



GPFS Best Practices

Programming, Configuration, Environment and Performance Perspectives

Tutorial for GPFS versions 3.3 and earlier



"A supercomputer is a device for turning compute-bound problems into I/O-bound problems." Ken Batcher

Raymond L. Paden, Ph.D.
HPC Technical Architect
IBM Deep Computing
raypaden@us.ibm.com
877-669-1853

Version 17.2.0
13 Apr 10

Special Notices from IBM Legal

This presentation was produced in the United States. IBM may not offer the products, programs, services or features discussed herein in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the products, programs, services, and features available in your area. Any reference to an IBM product, program, service or feature is not intended to state or imply that only IBM's product, program, service or feature may be used. Any functionally equivalent product, program, service or feature that does not infringe on any of IBM's intellectual property rights may be used instead of the IBM product, program, service or feature.

Information in this presentation concerning non-IBM products was obtained from the suppliers of these products, published announcement material or other publicly available sources. Sources for non-IBM list prices and performance numbers are taken from publicly available information including D.H. Brown, vendor announcements, vendor WWW Home Pages, SPEC Home Page, GPC (Graphics Processing Council) Home Page and TPC (Transaction Processing Performance Council) Home Page. IBM has not tested these products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

IBM may have patents or pending patent applications covering subject matter in this presentation. The furnishing of this presentation does not give you any license to these patents. Send license inquiries, in writing, to IBM Director of Licensing, IBM Corporation, New Castle Drive, Armonk, NY 10504-1785 USA.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Contact your local IBM office or IBM authorized reseller for the full text of a specific Statement of General Direction.

The information contained in this presentation has not been submitted to any formal IBM test and is distributed "AS IS". While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. The use of this information or the implementation of any techniques described herein is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. Customers attempting to adapt these techniques to their own environments do so at their own risk.

IBM is not responsible for printing errors in this presentation that result in pricing or information inaccuracies.

The information contained in this presentation represents the current views of IBM on the issues discussed as of the date of publication. IBM cannot guarantee the accuracy of any information presented after the date of publication.

IBM products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

Any performance data contained in this presentation was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements quoted in this presentation may have been made on development-level systems. There is no guarantee these measurements will be the same on generally-available systems. Some measurements quoted in this presentation may have been estimated through extrapolation. Actual results may vary. Users of this presentation should verify the applicable data for their specific environment.

Microsoft, Windows, Windows NT and the Windows logo are registered trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

LINUX is a registered trademark of Linus Torvalds. Intel and Pentium are registered trademarks and MMX, Itanium, Pentium II Xeon and Pentium III Xeon are trademarks of Intel Corporation in the United States and/or other countries.

Other company, product and service names may be trademarks or service marks of others.



Author, Revisions and TBDs

Author: Raymond L. Paden
Date: 6 June 2008
Version: v15.1

TBDs:

- ▶ Add SoFS example under NAS in Taxonomy
- ▶ Add SoFS to section on NFS and CNFS
- ▶ Add snapshots
- ▶ GPFS SNMP support
- ▶ Guide lines on where to use NFS
- ▶ pros and cons for using GPFS for home directories

- ▶ OS Commands
 - see pp.79-80 in *Concepts, Planning, Installation Guide*.
- ▶ OS Calls
 - see pp.80-83 in *Concepts, Planning, Installation Guide*.
- ▶ GPFS Command Processing
 - see pp. 83-84 in *Concepts, Planning, Installation Guide*.

- ▶ GPFS Port Usage
 - see pp. 122-124 of *Advanced Admin Guide*





Abstract and Biographical Sketch



Abstract

GPFS (General Parallel File System) is IBM's clustered/parallel file system commonly used for HPC and cluster applications. It has been generally available since 1998 giving it both maturity and market presence. This 2-day seminar is divided into 4 sessions, which are both flexible and dynamic. They examine GPFS's features, semantics, programming considerations, configuration procedures, tuning and optimization guidelines, best practices and environment. If supported hardware is available, it includes a "hands on" lab exercise or a live demonstration. A planning and design session can also be included to help customers deploy GPFS for their specific circumstances. Specific topics are emphasized on the basis of attendee interest. This seminar is delivered in a comfortable environment encouraging question and answer dialogue.

Biographical Sketch

Dr. Ray Paden is currently an HPC Technical Architect with world wide scope in IBM's Deep Computing organization, a position he has held since June, 2000. His particular areas of focus include HPC storage systems, performance optimization and cluster design. Before joining IBM, Dr. Paden worked as software engineer doing systems programming and performance optimization for 6 years in the oil industry. He also served in the Computer Science Department at Andrews University for 13 years, including 4 years as department chair. He has a Ph.D. from the Illinois Institute of Technology in Computer Science. He has done research and published papers in the areas of parallel algorithms and combinatorial optimization, performance tuning, file systems, and computer education. He has served in various capacities on the planning committee for the Supercomputing conference since 2000. He is currently a member of ACM, IEEE and Sigma Xi. As a professor, he has won awards for excellence in both teaching and research. He has also received the Outstanding Innovation Award from IBM.



Sample 2 Day Agenda

■ Day 1

- Session 1 (8:30 AM - 10:00 AM, 10:30 AM - NOON)
 - Introduction: Parallel I/O, Clustered Files Systems and a Cluster Storage Taxonomy
 - Overview of GPFS and Various Design Motivations
 - GPFS Architecture
- Session 2 (1:30 PM - 3:00 PM, 3:30 PM - 5:00 PM)
 - GPFS Organization and Topology
 - GPFS Environment: servers, disk controllers, disk technology, networks
 - Example GPFS Configurations
- GPFS Design Exercise: optional "paper and pencil" exercise

■ Day 2

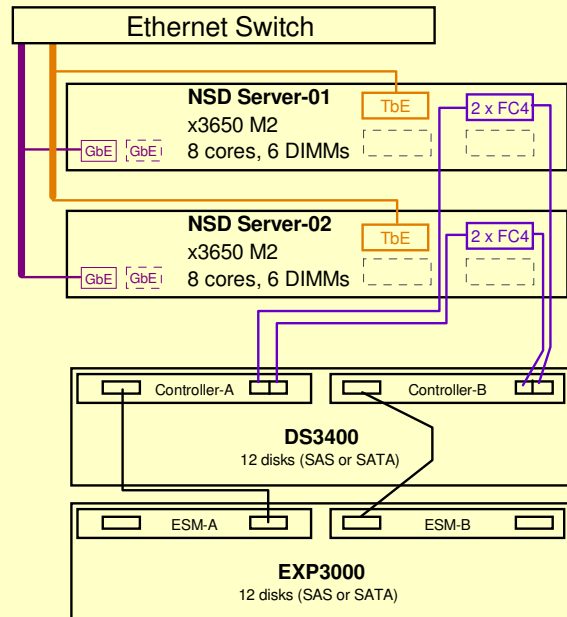
- Session 1 (8:30 AM - 10:00 AM, 10:30 AM - NOON)
 - Review written exercise from previous day
 - GPFS System Administration
 - GPFS Configuration Example
 - Optional GPFS lab exercise: install and configure GPFS
- Session 2 (1:30 PM - 3:00 PM, 3:30 PM - 5:00 PM)
 - Specific topics selected based on attendee interests; topics include
 1. GPFS planning and design (intended for customers who have purchased GPFS)
 2. Information Life Cycle Management (ILM) and HSM Product Integration
 3. Clustered NFS (CNFS)
 4. SoNAS and SoFS
 5. Snapshots
 6. Disaster Recovery
 7. SNMP Support
 8. Miscellaneous Best Practices
 9. GPFS Roadmap (requires NDA)

COMMENT: The material in this slide set is detailed and comprehensive; it requires 3 full days to cover it in its entirety (including the hands on lab). However, this tutorial is generally covered in 2 days at customer sites by including only the material relevant to the customer.



Sample Test Configurations

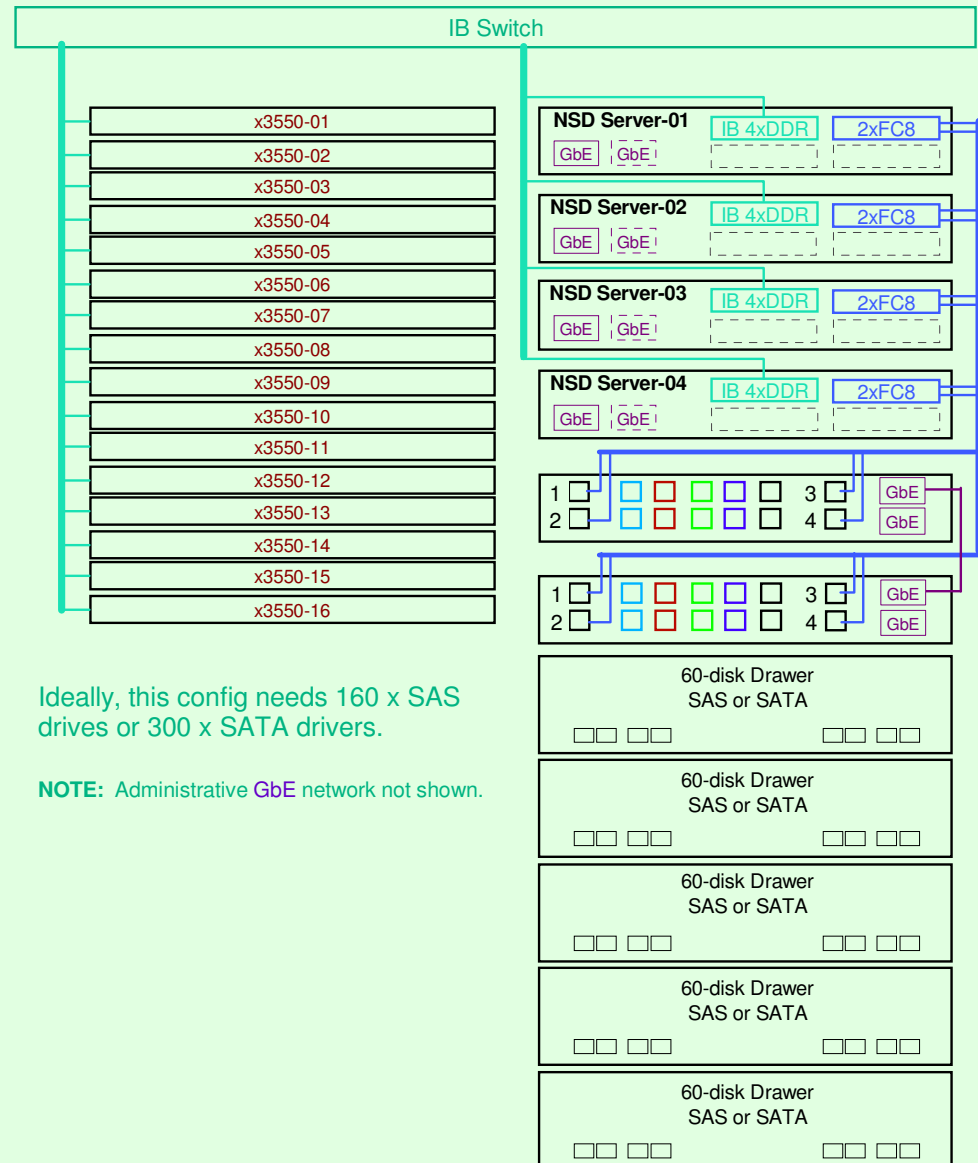
Example #1 - Simple



COMMENT:

These diagrams are intended to illustrate the range of possibilities for configurations that could be used for a test system to do the lab exercise. There are many other possibilities, including the use of "internal" SCSI or SAS drives.

Example #2 - Elaborate

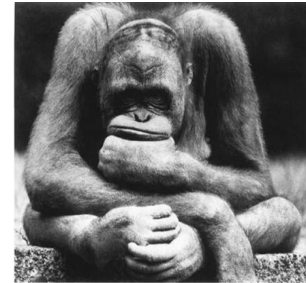




Tutorial Objectives

■ Conceptual understanding of GPFS

- With a conceptual understanding and a man page, a sysadm can do anything!



theory

■ Practical understanding of how to use GPFS

- GPFS integration with other products
 - servers, disk controllers, networks, OSs
- "Hands on" introduction
 - assumes appropriate HW resources are available



practical

■ Targeted Audience

- system administrators
- systems and application programmers
- system architects
- computer center managers

■ Requirements

- cluster experience in keeping with one of the previous backgrounds

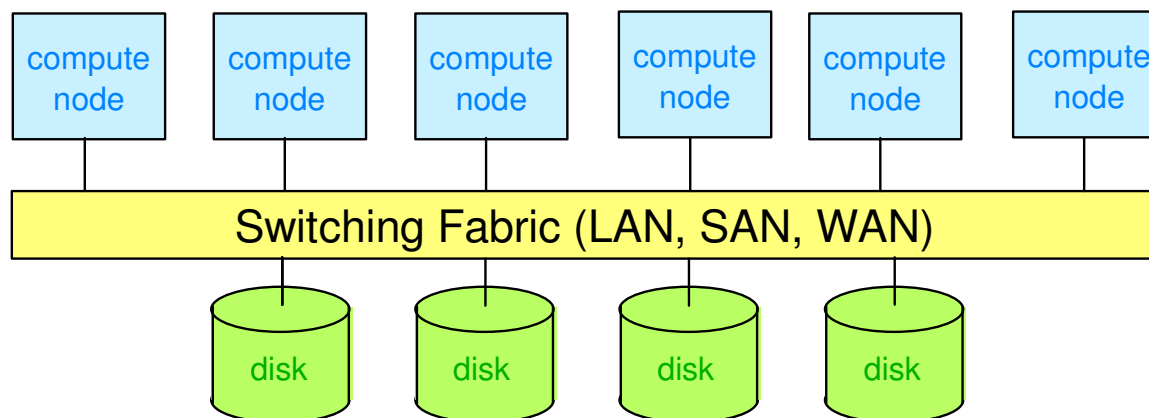
An educated customer is a good customer!



1. Introduction

GPFS is a *shared disk, parallel clustered file system*.

- ▶ Shared disk
 - all user and meta data are accessible from any disk to any node
- ▶ Parallel
 - user data and metadata flows between all nodes and all disks in parallel
- ▶ Clustered
 - 1 to 1000's of nodes under common rubric

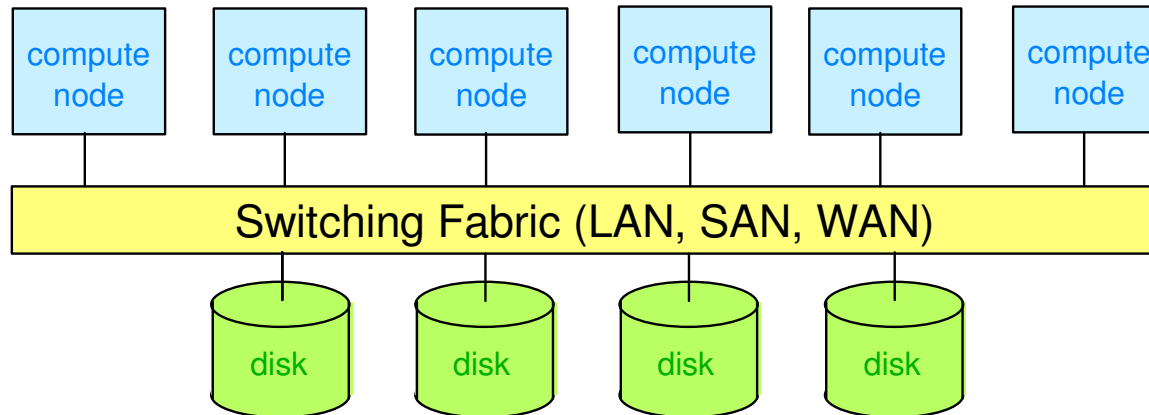




Parallel I/O in a Cluster

User data and metadata flows between all nodes and all disks in parallel

- ▶ Multiple tasks distributed over multiple nodes simultaneously access file data
- ▶ Multi-task applications access common files in parallel
- ▶ Files span multiple disks
- ▶ File system overhead operations are distributed and done in parallel
- ▶ Provides a consistent global name space across all nodes of the cluster

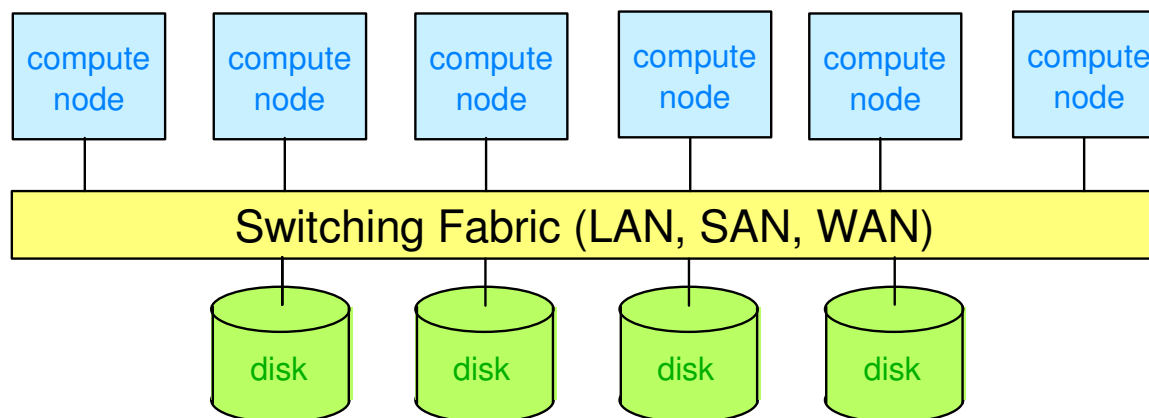




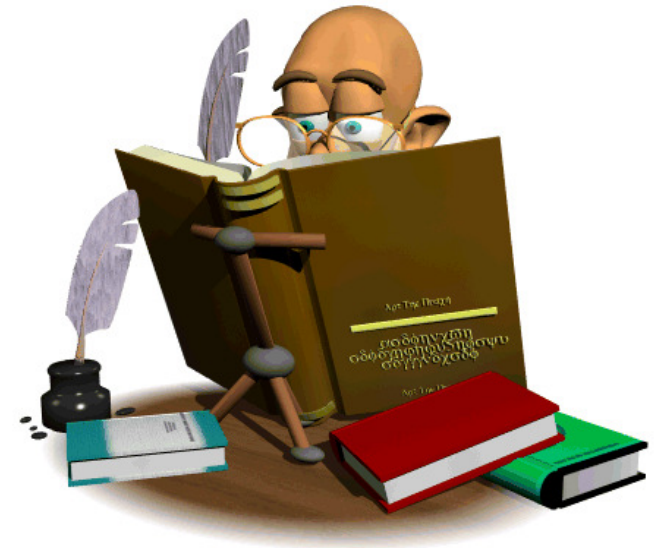
Parallel I/O in a Cluster

The promise of parallel I/O is increased performance and robustness in a cluster and it naturally maps to the architecture of a cluster.

The challenge of parallel I/O is that it is a more complex model of I/O to use and manage.



Textbook examples are great. But in practical terms, what is GPFS?





What is GPFS?



General Parallel File System

All of GPFS's rivals do some of these things, none of them do all of them!

- ▶ **General:** supports wide range of applications and configurations
- ▶ **Cluster:** from large (4000+ in a multi-cluster) to small (only 1 node) clusters
- ▶ **Parallel:** user data and metadata flows between all nodes and all disks in parallel
- ▶ **HPC:** supports high performance applications
- ▶ **Flexible:** tuning parameters allow GPFS to be adapted to many environments
- ▶ **Capacity:** from high (4+ PB) to low capacity (only 1 disk)
- ▶ **Global:** Works across multiple nodes, clusters and labs (*i.e.*, LAN, SAN, WAN)
- ▶ **Heterogenous:**
 - Native GPFS on AIX, Linux, Windows as well as NFS and CIFS
 - Works with almost any block storage device
- ▶ **Shared disk:** all user and meta data are accessible from any disk to any node
- ▶ **RAS:** reliability, accessibility, serviceability
- ▶ **Ease of use:** GPFS is not a black box, yet it is relatively easy to use and manage
- ▶ **Basic file system features:** POSIX API, journaling, both parallel and non-parallel access
- ▶ **Advanced features:** ILM, integrated with tape, disaster recovery, SNMP, snapshots, robust NFS support, hints



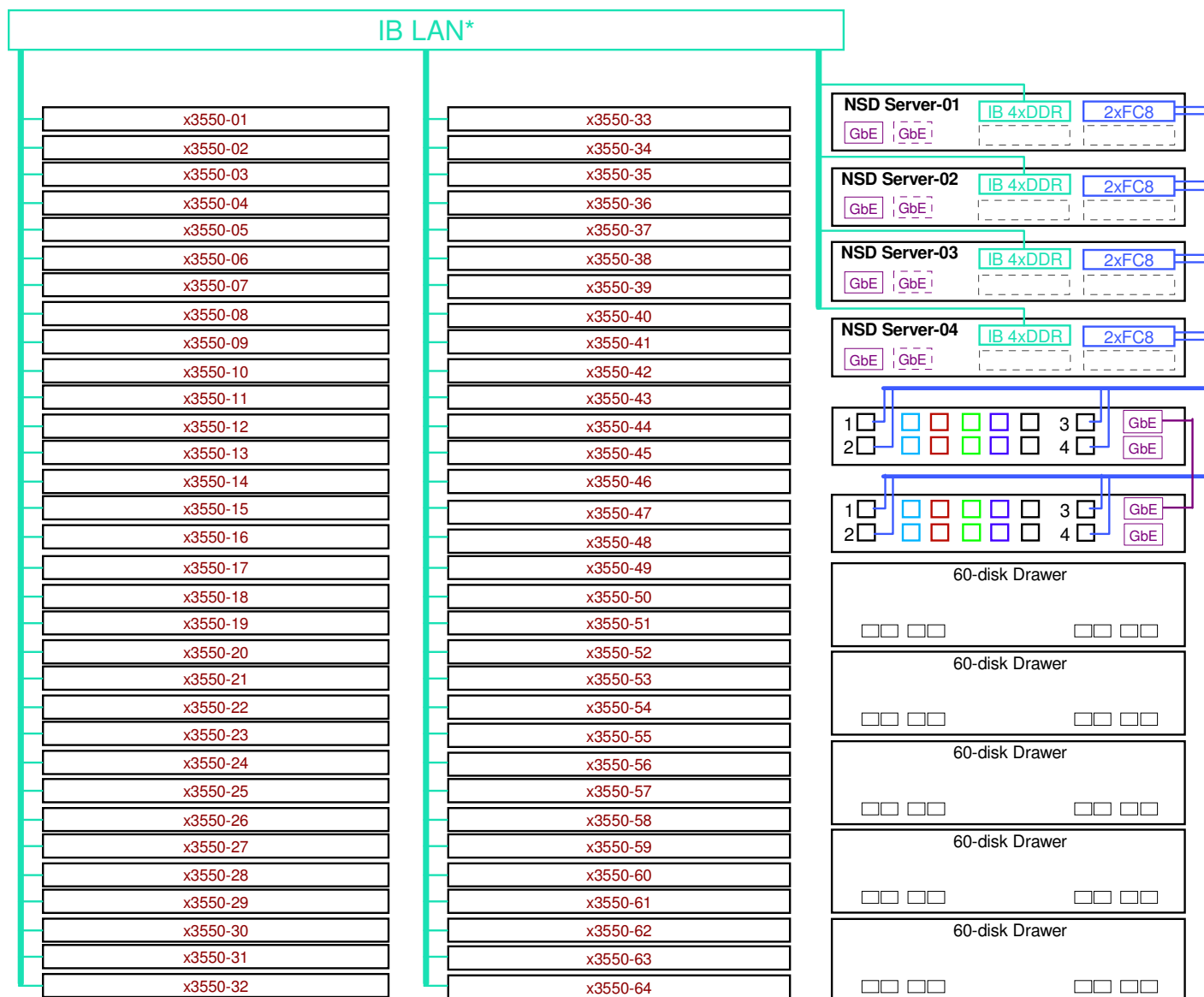
What is GPFS?

Typical Example

Aggregate Performance and Capacity

Data rate: streaming rate < 5 GB/s, 4 KB transaction rate < 40,000 IOP/s

Usable capacity < 240 TB



LAN Configuration

- ▶ Performance scales linearly in the number of storage servers
- ▶ Add capacity without increasing the number of servers
- ▶ Add performance by adding more servers and/or storage
- ▶ Inexpensively scale out the number of clients

★Though not shown, a cluster like this will generally include an administrative GbE network.



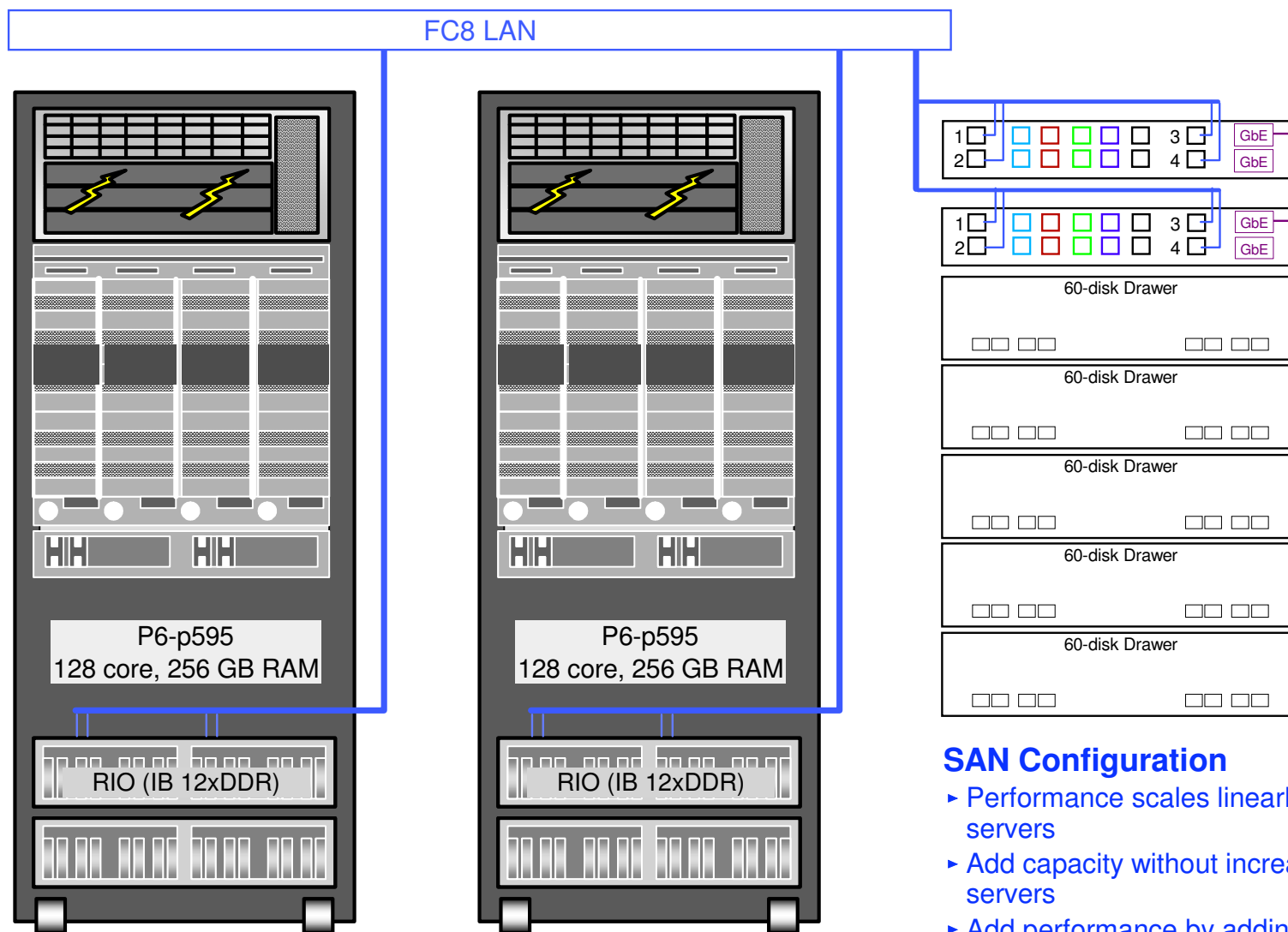
What is GPFS?

Another Typical Example

Aggregate Performance and Capacity

Data rate: streaming rate < 5 GB/s, 4 KB transaction rate < 40,000 IOP/s

Usable capacity < 240 TB



SAN Configuration

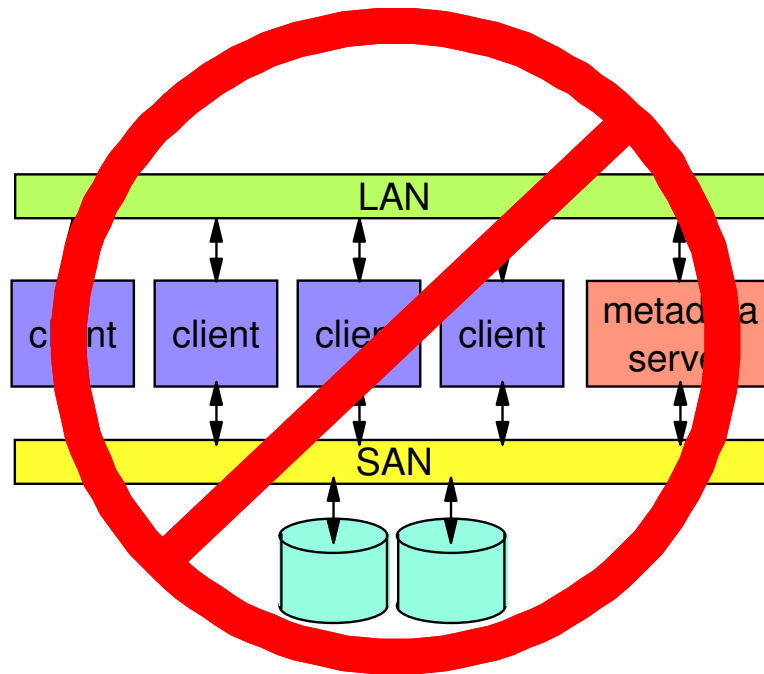
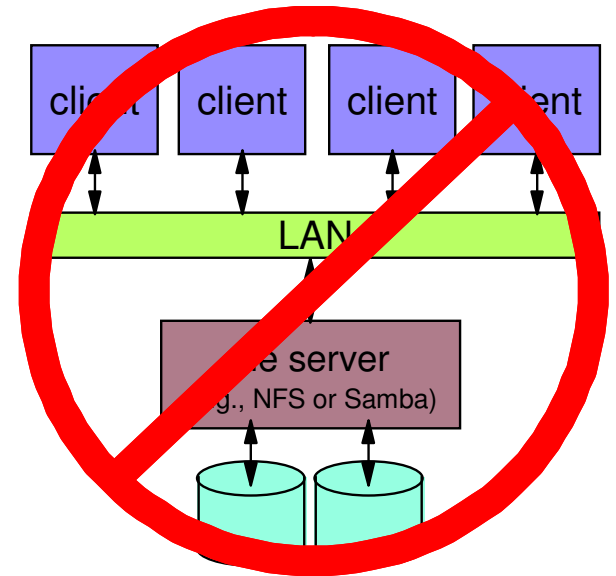
- ▶ Performance scales linearly in the number of servers
- ▶ Add capacity without increasing the number of servers
- ▶ Add performance by adding more servers and/or storage



What GPFS is *Not*

GPFS is *not* a client/server file system like NFS, CIFS (Samba) or AFS/DFS with a single file server.

- ▶ GPFS *nodes can be* an NFS or CIFS server, but GPFS treats them like any other application.



GPFS is *not* a SAN file system with dedicated metadata server.

- ▶ GPFS *can* run in a SAN file system like mode, but it does *not* have a dedicated metadata server.

GPFS avoids the bottlenecks introduced by centralized file and/or metadata servers.



What GPFS is *Not*

GPFS is **not** a niche file system for IBM system P products

- ▶ Yesterday
 - GPFS **was** a parallel file system for IBM SP systems
- ▶ Today
 - GPFS **is** a general purpose clustered parallel file system tunable for many workloads on many configurations.



Winterhawk



BlueGene/P



P6 p595



iDataPlex



BladeCenter/H



Where GPFS Is Used Today

GPFS is a mature product with established market presence. It has been generally available since 1998 with research development starting in 1991. Applications include...

- Aerospace and Automotive
- Banking and Finance
- Bio-informatics and Life Sciences
- Defence
- Digital Media
- EDA (Electronic Design Automation)
- General Business
- National Labs
- Petroleum
- SMB (Small and Medium sized Business)
- Universities
- Weather Modeling



Where GPFS Is Used Today

LARGE Clusters

Smaller Number of Big Nodes



155 p575 Nodes



Herd of Elephants

Larger Number of Small Nodes



3780 iDataPlex Nodes
105 P6p575 Nodes



Army of Ants



Where GPFS Is Used Today

small Clusters



smaller Clusters





2. Cluster Storage Taxonomy



The following pages examine a taxonomy of file systems commonly used with clusters. They may or may not be a clustered file system and they support varying degrees of parallelism. They do not represent mutually exclusive choices.

- Conventional I/O
- Asynchronous I/O
- Networked File Systems
- Network Attached Storage (NAS)
- Basic Clustered File Systems
- SAN File Systems
- Multi-component Clustered File Systems
- High Level Parallel I/O



Conventional I/O

- ▶ Used generally for "local file systems"
 - the basic, "no frills, out of the box" file system
- ▶ Supports POSIX I/O model
- ▶ Generally supports limited forms of parallelism
 - intra-node process parallelism
 - disk level parallelism possible via striping
 - not truly a parallel file system
- ▶ Journal, extent based semantics
 - *journaling* (AKA *logging*): to log information about operations performed on the file system meta-data as atomic transactions. In the event of a system failure, a file system is restored to a consistent state by replaying the log and applying log records for the appropriate transactions.
 - *extent*: a sequence of contiguous blocks allocated to a file as a unit and is described by a triple consisting of <logical offset, length, physical>
- ▶ If they are a native FS, they are integrated into the OS (*e.g.*, caching done via VMM)
- ▶ Examples: ext3, JFS, NTFS, ReiserFS, XFS



Asynchronous I/O

- ▶ Abstractions allowing multiple threads/tasks to *safely* and simultaneously access a common file
 - non-blocking I/O
 - built on top of a base file system
- ▶ Parallelism available if its supported in the base file system
- ▶ Included in the POSIX 4 standard
 - not *necessarily* supported on all Unix operating systems
- ▶ Examples:
 - commonly available under real time operating systems
 - Supported today on various "flavors" of standard Unix
 - AIX, Solaris, Linux (starting with 2.6)



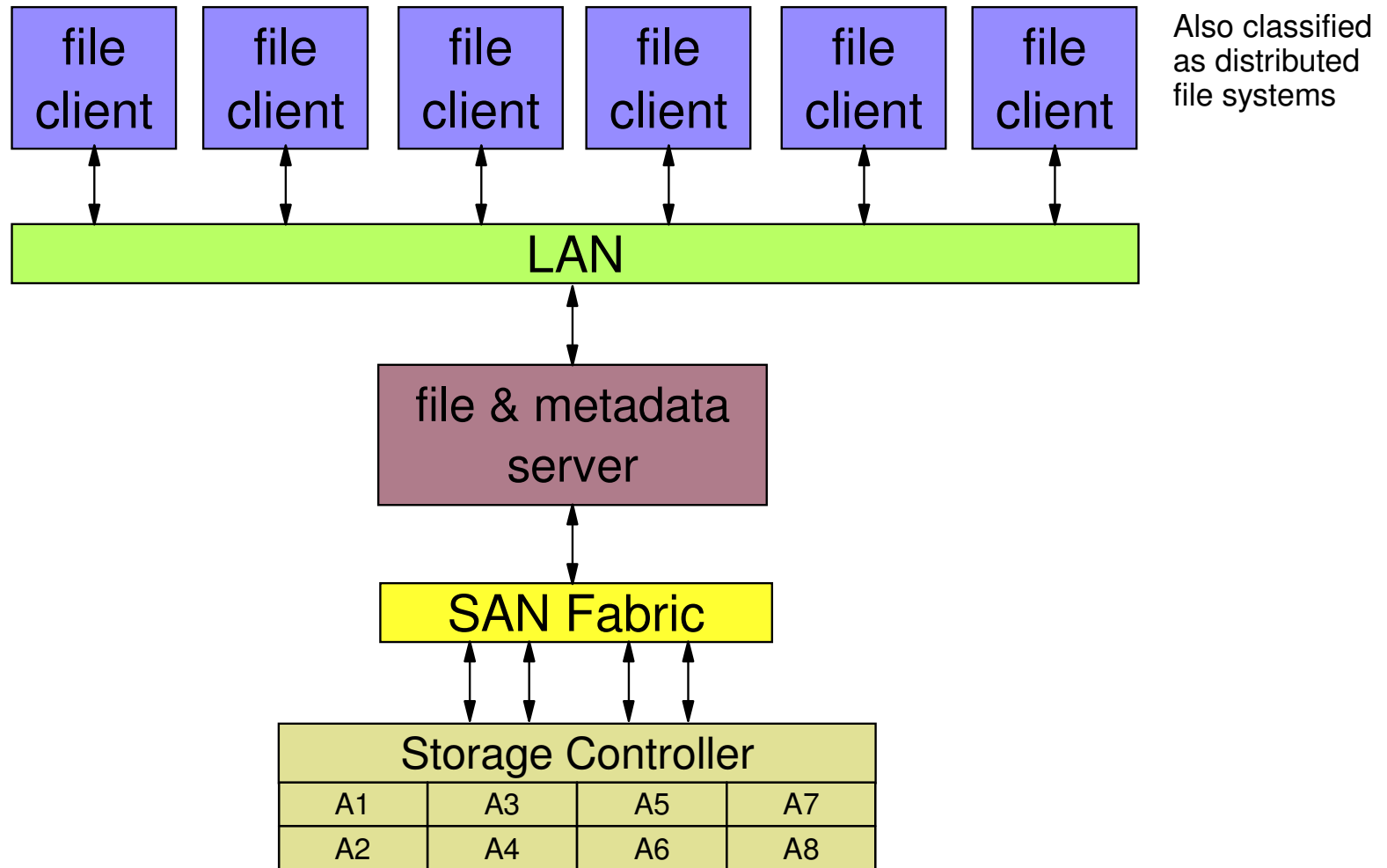
Networked File Systems

- ▶ Disk access from remote nodes via network access
 - generally based on TCP/IP over Ethernet
 - Useful for on-line interactive access (*e.g.*, home directories)
- ▶ NFS is ubiquitous in Unix/Linux environments
 - does not provide a genuinely parallel model of I/O
 - it is not cache coherent (will future versions like pNFS correct this?)
 - parallel write requires `O_SYNC` and `-noac` options to be safe
 - poorer performance for HPC jobs, especially parallel I/O
 - write: only 90 MB/s on system capable of 400 MB/s (4 tasks)
 - read: only 381 MB/s on system capable of 740 MB/s (16 tasks)
 - uses POSIX I/O API, but not its semantics
 - traditional NFS configurations limited by "single server" bottleneck
 - while NFS is not designed parallel file access, by placing restrictions on an application's file access and/or doing non-parallel I/O, it may be possible to get "good enough" performance
 - NFS clients available for Windows, but POSIX to NTFS mapping is awkward
 - GPFS provides a high availability version of NFS called Clustered NFS
- ▶ CIFS is ubiquitous in Windows environments
 - Samba is a CIFS server available under Unix/Linux that maps a POSIX based file system to the Windows/NTFS model.

Also classified
as distributed
file systems



Networked File Systems



COMMENT:

Traditionally, a single NFS/CIFS file server manages both user data and metadata operations which "gates" performance/scaling and presents a single point of failure risk. Products (e.g., CNFS) are available that provide multiple server designs to avoid this issue.



Network Attached Storage (NAS)

► Appliance Concept

- Traditionally focused on the CIFS and/or NFS protocols
- Integrated HW/SW storage product
 - integrates servers, storage controllers, disks, networks, file system, protocol, etc. all into single product
 - main advantage: "black box" design (i.e., ease of use at the expense of flexibility)
 - not intended for high performance storage
- Provides an NFS server and/or CIFS/Samba solution
 - these are **server** based products; they do **not** improve client access or operation
 - may support other protocols (e.g., iSCSI, http)
- Generally based on Ethernet LANs
- Is this just a subclass of the networked file systems level?

► Examples

- Netapps (also rebranded as IBM nSeries)
 - Provides excellent performance for IOPS and transaction processing workloads with favorable temporal locality.
- Scale-out File System (SoFS, SoNAS)
 - An IBM product supporting CIFS, http, iSCSI, NFS, NSD (i.e., GPFS) protocols

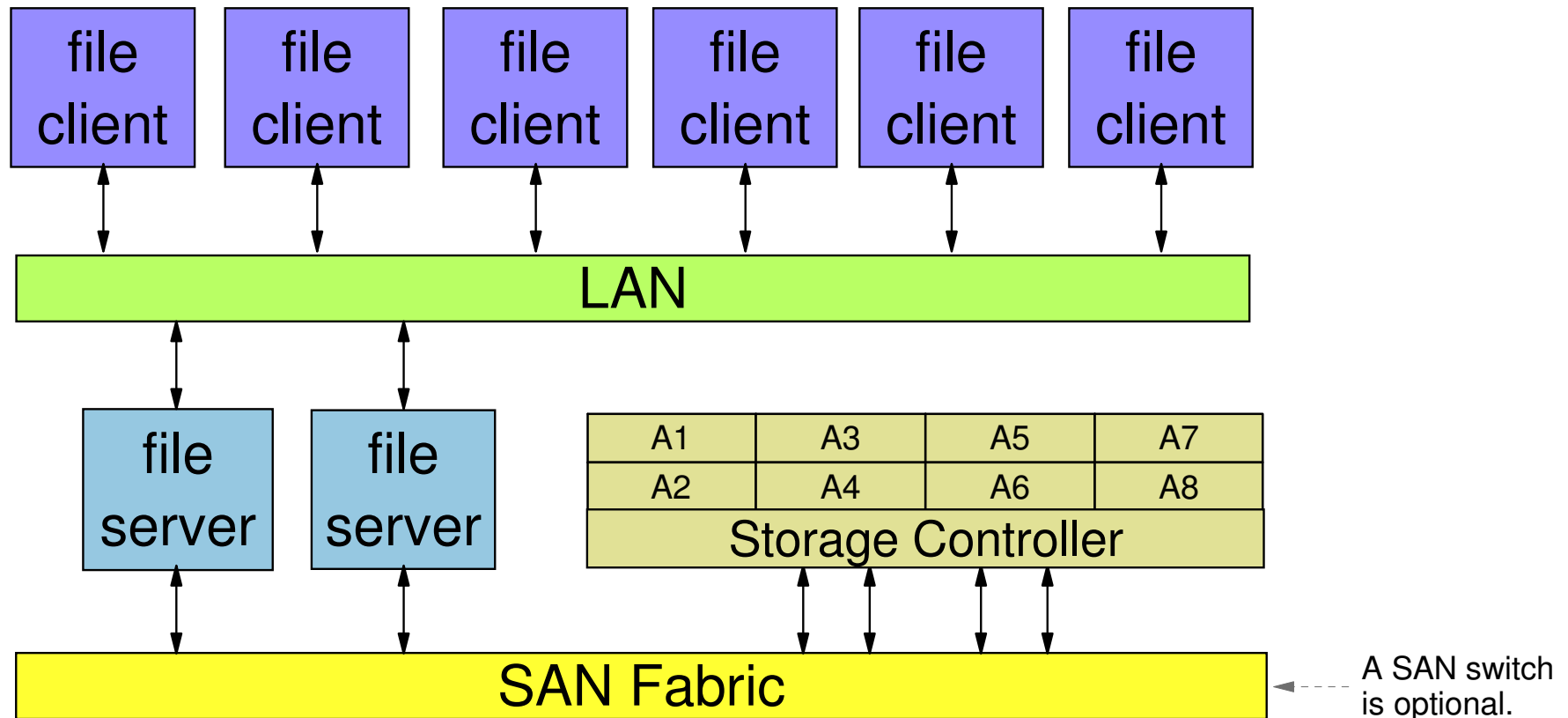


Basic Clustered File Systems

- ▶ Satisfies the definition of a clustered file system
- ▶ File access is parallel
 - supports POSIX API, but provides safe parallel file access semantics
 - guarantees portability to other POSIX based file systems
- ▶ File system overhead operations
 - file system overhead operations is distributed and done in parallel
 - there are no single server bottlenecks
 - *n.b.*, no metadata servers
- ▶ Common component architecture
 - commonly configured using separate file clients and file servers
 - this is common for reasons of economy; for many storage systems, it costs too much to have a separate storage controller for every node
 - some FS's allow a single component architecture where file clients and file servers are combined
 - yields very good scaling for asynchronous applications
- ▶ file clients access file data through file servers via the LAN
- ▶ Example: GPFS (IBM), GFS (Sistina/Redhat), IBRIX Fusion



Basic Clustered File Systems



File system overhead operations are *distributed* across the entire cluster and is done in parallel; it is **not** concentrated in any given place. There is no single server bottleneck. User data and metadata flows between all nodes and all disks via the file servers.

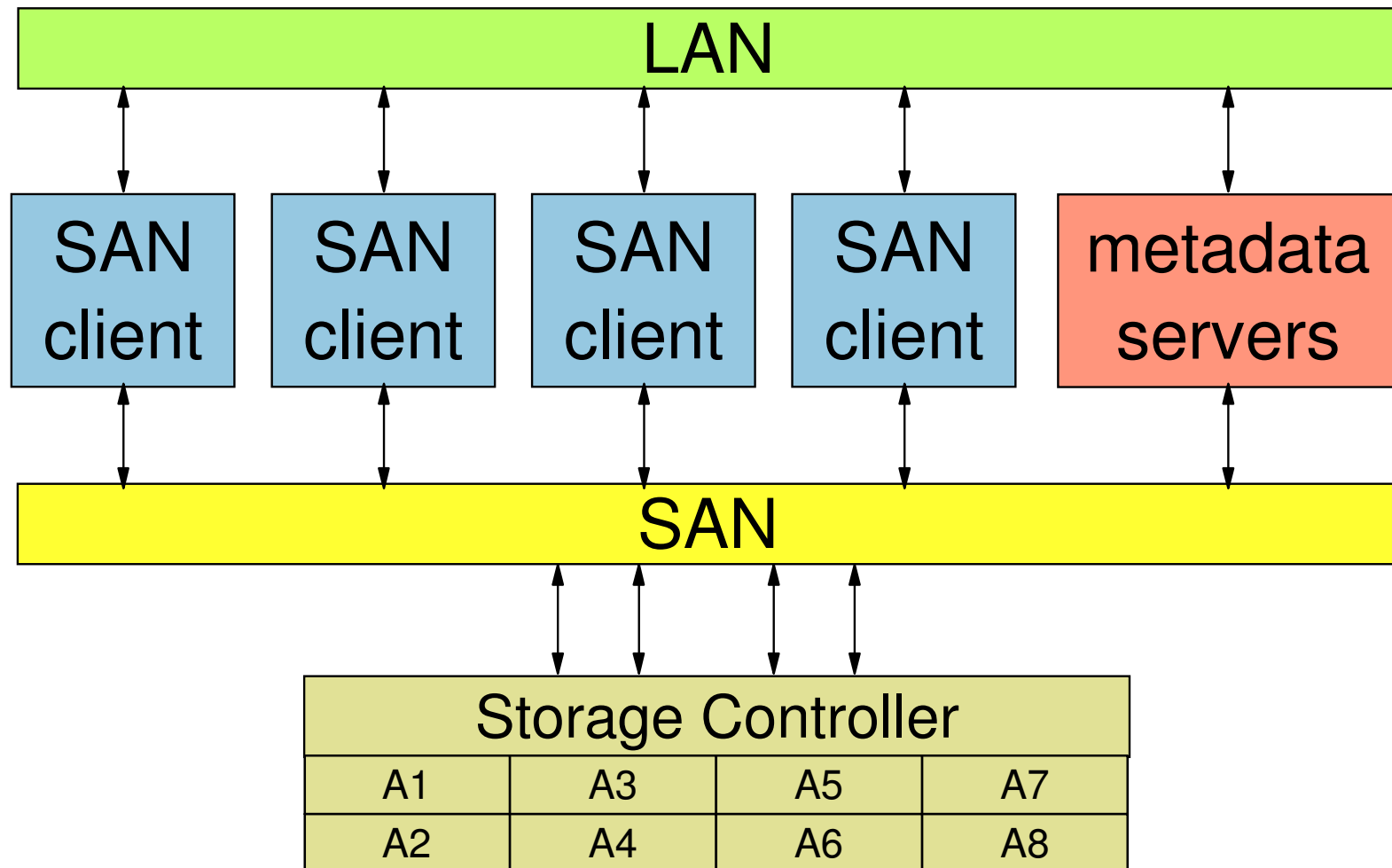


SAN File Systems

- ▶ File access is parallel
 - supports POSIX API, but provides safe parallel file access semantics
 - guarantees portability to other POSIX based file systems
- ▶ File system overhead operations
 - it is NOT done in parallel
 - single metadata server with a backup metadata server
 - metadata server is accessed via the LAN
 - metadata server is a potential bottleneck, but it is not considered a limitation since these FS's are generally used for smaller clusters
- ▶ Dual component architecture
 - file client/server and metadata server
- ▶ All disks connected to all file client/server nodes via the SAN
 - file data accessed via the SAN, not the LAN
 - removes need for expensive LAN where high BW is required (*e.g.*, IB, Myrinet)
 - inhibits scaling due to cost of FC Switch Tree (*i.e.*, SAN)
- ▶ Example: CXFS (SGI), SNFS (Quantum, formerly ADIC), QFS (Sun)
 - ideal for smaller numbers of nodes
 - SNFS scales to 50+ nodes
 - CXFS scales up to 64+ nodes (appropriate for many-processor Altix systems)



SAN File Systems



File system protocol is *concentrated* in the metadata server and is **not** done in parallel; all file client/server nodes must coordinate file access via the metadata server. There are generally no client only nodes in this type of cluster, and hence the need for large scaling is not needed.



Multi-component Clustered File Systems

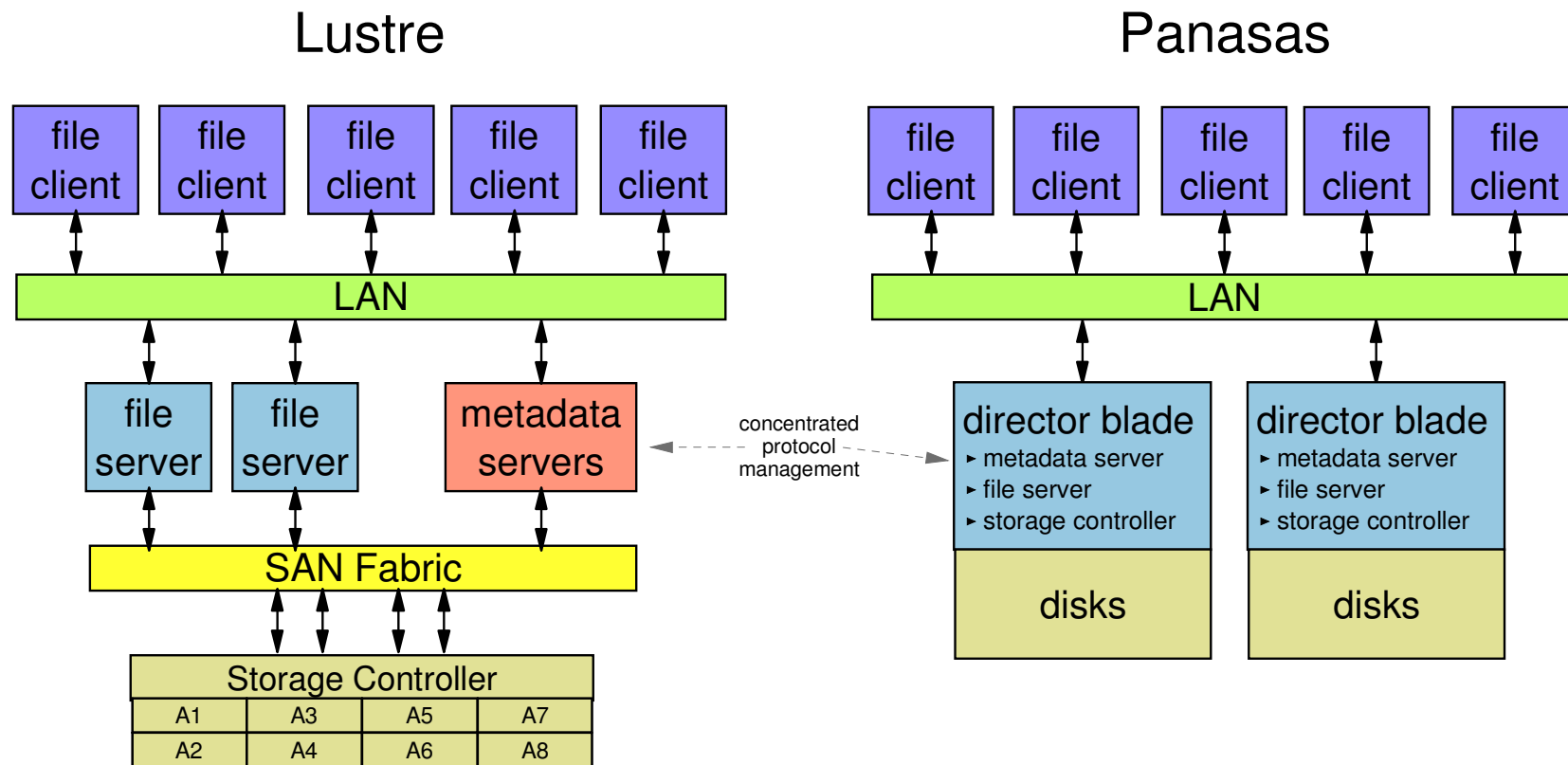
- ▶ Satisfies the definition of a clustered file system
- ▶ File access is parallel
 - supports POSIX API, but provides safe parallel file access semantics
 - guarantees portability to other POSIX based file systems
- ▶ File system overhead operations
 - Lustre: 1 metadata server per file system (with backup) accessed via LAN
 - potential bottleneck (deploy multiple file systems to avoid backup)
 - Panasas: the "director blades" manages protocol
 - each "shelf" contains a director blade and 10 disks accessible via Ethernet
 - this provides multiple metadata servers reducing contention
- ▶ Multi-component architecture
 - Lustre: file clients, file servers, metadata server
 - Panasas: file clients, director blade
 - director blade encapsulates file service, metadata service, storage controller operations
- ▶ file clients access file data through file servers or director blades via the LAN
- ▶ Examples: Lustre, Panasas
 - Lustre: Linux only, Panasas: Linux and Windows.
 - object oriented disks
 - Lustre emulates object oriented disks
 - Panasas uses actual OO disks; user can only use Panasas disks

Will they improve
this in the future?

Do OO disks really add
value to the FS? Other
FS's efficiently
accomplish the same
thing at a higher level.



Multi-component Clustered File Systems



While different in many ways, Lustre and Panasas are similar in that they both have concentrated file system overhead operations (*i.e.*, protocol management). The Panasas design, however, scales the number of protocol managers proportionally to the number of disks and is less of a bottleneck than for Lustre.



Higher Level Parallel I/O

- ▶ High level abstraction layer providing a parallel I/O model
- ▶ Built on top of a base file system (conventional or parallel)
- ▶ MPI-I/O is the ubiquitous model
 - parallel disk I/O extension to MPI in the MPI-2 standard
 - semantically richer API and semantics
 - can do things that POSIX I/O was never designed to do
 - applications using MPI-I/O are portable
- ▶ Requires significant source code modification for use in legacy codes, but it has the advantages of being a standard (e.g., syntactic portability)
- ▶ Examples: IBM MPI, MPICH, OpenMPI, Scali MPI



Which File System Architecture is Best?

There is no concise answer to this question.

- ▶ It is application/customer specific.
- ▶ All of them serve specific needs.
- ▶ All of them work well *if properly deployed and used according to their design specs.*
- ▶ Issues to consider are
 - application requirements
 - often requires compromise between competing needs
 - how the product implements a specific architecture



3. GPFS Design Motivation

Consider miscellaneous observations to motivate the design of GPFS.



Efficiency Is Critical in a Cluster

Clusters are intended to provide cost effective performance scaling. Thus it is imperative that I/O and computational performance keep pace with each other.

- ▶ A cluster designed to perform TFLOP calculations must be able to access up to 100's of GB of data per second.
- ▶ A cluster is no faster than its slowest component (large or small!)



Anecdote:

200 years ago, when a tree fell across the road and your ox wasn't big enough to move it out of the way, you didn't go grow a bigger ox; you got more oxen.

Rear Admiral Grace Murray Hopper
Computer Pioneer
1906-1992



Efficiency Is Critical in a Cluster

Amdahl's Law Applied to a Cluster

Speedup = $1 / (f + F/n)$ where

- F = fraction time that can utilize parallelism
- f = fraction of time that can NOT utilize parallelism (n.b., $f = 1 - F$)
- n = number of nodes (also called *ideal speedup*)

Parallel Efficiency is then

- Efficiency = $100 * \text{Speedup} / n$

I/O *can, but need not* be a large contributor to f in clusters. I call the inefficiency represented by the term f in Amdahl's law "[Amdahl inefficiency](#)" or "Amdahl overhead".

Consider a job on a 32 node/64 CPU Linux Cluster (LC). This job, when executed on a single node accessing a local scratch disk, devotes 10% of its job time writing to a file. By contrast, the LC writes via NFS to a single file server preventing parallel I/O operation. Assume the following...

- the file server is the same "out of the box" Linux system used for the sequential test
- the Ethernet connection rate used for NFS exceeds the sequential job's write rate
- number crunching and file reading phases of the job runs perfectly parallel (i.e., are small enough to be ignored)

In other words, the writes are sequentialized.

What are the speedup and efficiency values for this job?

Number of Tasks	Speedup	Efficiency
8	4.71	58.9%
16	6.40	40.0%
32	7.85	24.5%
64	9.86	15.4%



Let's examine a simple disk I/O program and modify it to do parallel disk I/O so that we can better appreciate the tasks that a parallel file system *must do* and that GPFS *does* to allow a programmer to do parallel I/O *safely*.



Simple I/O Program

```
int main()
{
    int  fd, k, nrec = 1024, bsz = 16384;
    char *fid_out = "myfile", buf[bsz];
    offset_t soff;      /* 64 bit seek offset */

    fd = open(fid_out, O_WRONLY | O_CREAT | O_TRUNC, 0777);
    for (k = 0; k < nrec; k++)
    {
        do_something(buf, bsz);
        soff = (offset_t)k * (offset_t)bsz;
        llseek(fd, soff, SEEK_SET);
        write(fd, buf, bsz);
    }
    close(fd);

    return 0;
}
```



What do we need to do to parallelize disk I/O?

1. mapping function (i.e., locating proper data across multiple disks over different nodes)
2. message passing (i.e., shipping data between client node task and disk located on remote server)
3. caching system (e.g., coherence, aging, swapping, "data-shipping", etc.)
4. parallel programming model (e.g., data striping, data decomposition, node to disk access patterns)
5. critical section programming
6. performance tuning
7. maintain state information
8. provide an API



That's a lot of work!
GPFS has 100's of KLOCs



GPFS Design Goal

Provide a parallel I/O system conforming to the POSIX API standard.

This allows you to write an application code to access one file without worrying *too much* about what the other tasks are doing.

You can't be blind, but you can focus on application needs without worrying too much about system issues. If the code is sequential, you can get the full benefits of a parallel file system without worrying at all about it!!



A Simple Parallel I/O Program

```
int main()
{
    int  fd, k, nrec = 1024, bsz = 16384, ntask = 2, tid;
    char *fid_out = "myfile", buf[bsz];
    offset_t soff;      /* 64 bit seek offset */

    tid = spawn_task(ntask);

    fd = open(fid_out, O_WRONLY | O_CREAT, 0777);
    for (k = tid; k < nrec; k+=ntask)
    {
        do_something(buf, bsz);
        soff = (offset_t)k * (offset_t)bsz;
        llseek(fd, soff, SEEK_SET);
        write(fd, buf, bsz);
    }
    close(fd);

    return 0;
}
```

The proper way to open a file is to set a barrier and use O_TRUNC; however, this seems to work OK for this simple example.

If O_TRUNC is used without a barrier or some equivalent timing primitive, records written to the file before subsequent tasks open the file will be "clobbered".

In reality you will need more than this, but it will be application oriented ONLY!



You Do Not Have to Worry About...

- ▶ Which disk/file to write to (there is only one file seen by all tasks)
- ▶ If some other task/job has opened the file
- ▶ If somebody else is writing to the file right now
- ▶ Cache coherence
- ▶ If its portable ... its POSIX compliant!



Now Consider This: What Can Go Wrong if the FS is not Parallel?

```
int main()
{
    int  fd, k, nrec = 1024, bsz = 16384, ntask = 2, tid;
    char *fid_out = "myfile", buf[bsz];
    offset_t soff;      /* 64 bit seek offset */

    tid = spawn_task(ntask);

    fd = open(fid_out, O_RDWR | O_CREAT, 0777);

    while ((soff = find_record()))    /* assume soff%bsz == 0 */
    {
        critical section begin
        llseek(fd, soff, SEEK_SET);
        read(fd, buf, bsz);
        for (k = ntask; k < bsz; k += ntask)
            buf[k] = do_something(...);
        llseek(fd, soff, SEEK_SET);
        write(fd, buf, bsz);
        critical section end
    }
    close(ofd);

    return 0;
}
```



Now Consider This: What Can Go Wrong if the FS is not Parallel?

Task 0, node J acquires lock

Task 0, node J reads record N from disk

Task 0, node J modifies buf[] at indices 0, 2, 4, 6, 8, ...

Task 0, node J writes record N to local cache

Task 0, node J releases lock

Task 1, node K acquires lock

Task 1, node K reads record N from disk he does not know its in node J's cache

Task 1, node K modifies buf[] at indices 1, 3, 5, 7, 9, ...

Node J flushes cache

Task 1, node K writes record N to local cache

Task 1, node K releases lock

Node K flushes cache **clobbering Task 0's modifications!**



Now Consider This: What Can Go Wrong if the FS is not Parallel?

This scenario is quite possible under NFS, for example, since it is not cache coherent (after all, its not truly parallel!).

GPFS maintains cache coherence (among many other parallel tasks) making parallel access to a common file safe (by taking the usual concurrency precautions such as using locks or semaphores).

Comments on NFS

- ▶ NFS V3 has cleaned much of this up. By using -noac option opening the file with the O_SYNC flag, parallel writes can be done more safely, though this contributes to Amdahl inefficiency by sequentializing parallel writes. However, this is not fool proof. Some customer codes fail under NFS using these options where they run safely without error under GPFS.
- ▶ NFS V4 holds more promise, but the verdict is still out.
- ▶ And parallel NFS (pNFS) is on the horizon...



Parallel Access from Multiple Nodes

Earlier I said that GPFS...

provides a parallel I/O system conforming to the POSIX standard; therefore, you can write an application code to access one file without worrying *too much* about what the other tasks are doing.



Well there are 2 things to worry about:

1. Normal precautions against RAW, WAR, WAW errors
2. Performance issues
 - overlapping records sequentializes file access and contributes to "Amdahl inefficiency"

RAW = Read After Write
WAR = Write After Read
WAW = Write After Write



Simplicity vs. Flexibility



GPFS is simple to use, *but* it is not a black box!

Design Philosophy:

- ▶ Unlike a black box, GPFS provides many tuning parameters so that it can be adapted to many and changing environments.
- ▶ Over the years, GPFS has become simpler to use/administer.
- ▶ Its complexity comes from sitting on top of a complex stack.



GPFS is simplest complex product you will ever use!



GPFS salient feature - million knobs
GPFS problem - million knobs



The next several sections survey basic architectural, organizational and topological features of GPFS. This provides a conceptual understanding for GPFS.

This helps

- ▶ applications and systems programmers to more effectively utilize GPFS
- ▶ system administrators and architects to more effectively design and maintain a GPFS infrastructure



4. GPFS Architecture

1. Client vs. Server
2. LAN Model
3. SAN Model
4. Mixed SAN/LAN Model



Is GPFS a Client/Server Design?



Software Architecture Perspective: **No**

There is no single-server bottleneck, no protocol manager for data transfer. The mmfsd daemon runs symmetrically on all nodes. All nodes can and do access the file system via virtual disks (i.e., NSDs). All nodes can, if disks are physically attached to them, provide physical disk access for corresponding virtual disks.



Is GPFS a Client/Server Design?



Practical Perspective: Yes

1. GPFS is commonly *deployed* having dedicated storage servers ("NSD servers") and distinct compute clients ("NSD clients") running applications that access virtual disks (*i.e.*, "NSD devices" or "NSDs") via the file system.
 - this is based on economics (its *generally* too expensive to have 1 storage controller for every 2 nodes)
2. Nodes are designated as clients or servers for licensing.
 - client nodes only consume data
 - server nodes produce data for other nodes or provide GPFS management functions
 - producers: NSD servers, application servers (e.g., CIFS, NFS, FTP, HTTP)
 - management function: quorum nodes, manager nodes, cluster manager, configuration manager
 - server functions are commonly overlapped ◀ -- This reduces cost, but use caution!
 - example: use NSD servers as quorum and manager nodes
 - client licenses cost less than server licenses ◀ -- The new licensing model is much cheaper!
 - server nodes **can** perform client actions, but client nodes can **not** perform server actions

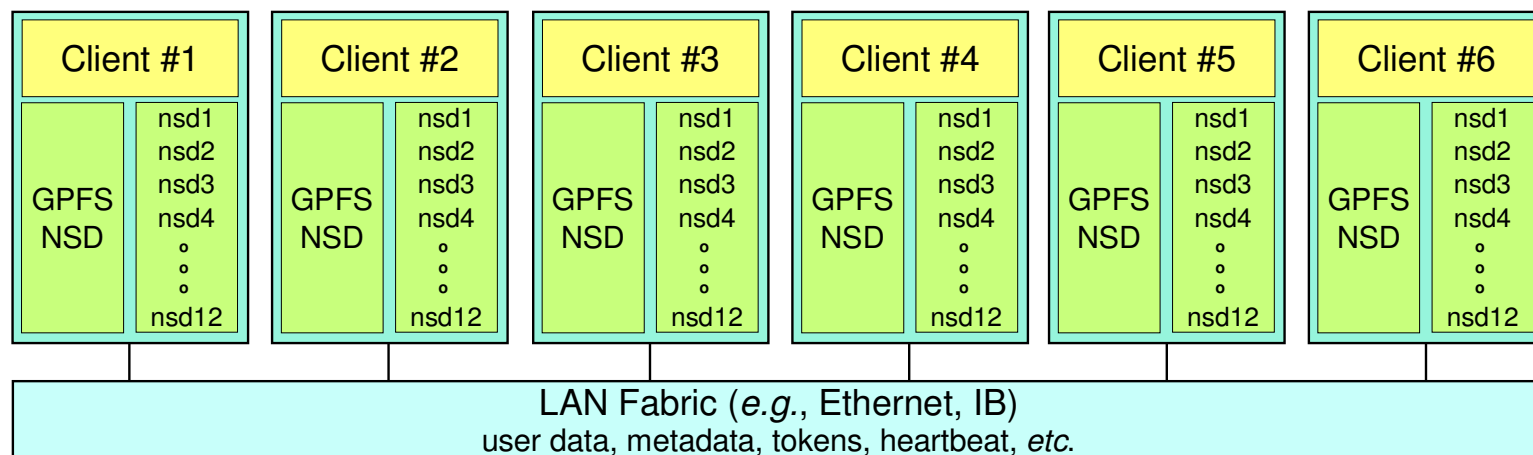


Local Area Network (LAN) Topology

Clients Access Disks Through the Servers via the LAN

NSD

- ▶ SW layer in GPFS providing a "virtual" view of a disk
- ▶ virtual disks which correspond to LUNs in the NSD servers with a bijective mapping



LUN

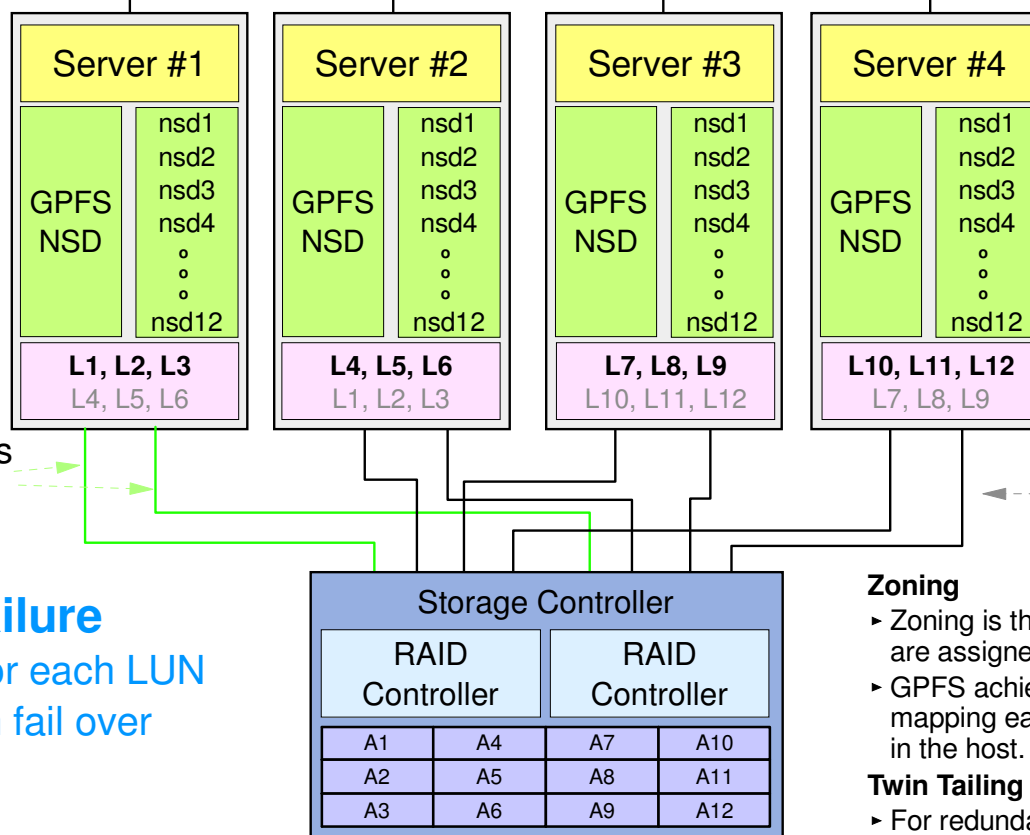
- ▶ Logical Unit
- ▶ Abstraction of a disk
 - AIX - hdisk
 - Linux - SCSI device
- ▶ LUNs map to RAID arrays in a disk controller or "physical disks" in a server

Redundancy

Each server has 2 connections to the disk controller providing redundancy

No single points of failure

- ▶ primary/backup servers for each LUN
- ▶ controller/host connection fail over
- ▶ Dual RAID controllers



Redundancy

Each LUN can have upto 8 servers. If a server fails, the next one in the list takes over.

There are 2 servers per NSD, a primary and backup server.

SAN switch can be added if desired.

Zoning

- ▶ Zoning is the process by which RAID sets are assigned to controller ports and HBAs
- ▶ GPFS achieves its best performance by mapping each RAID array to a **single** LUN in the host.

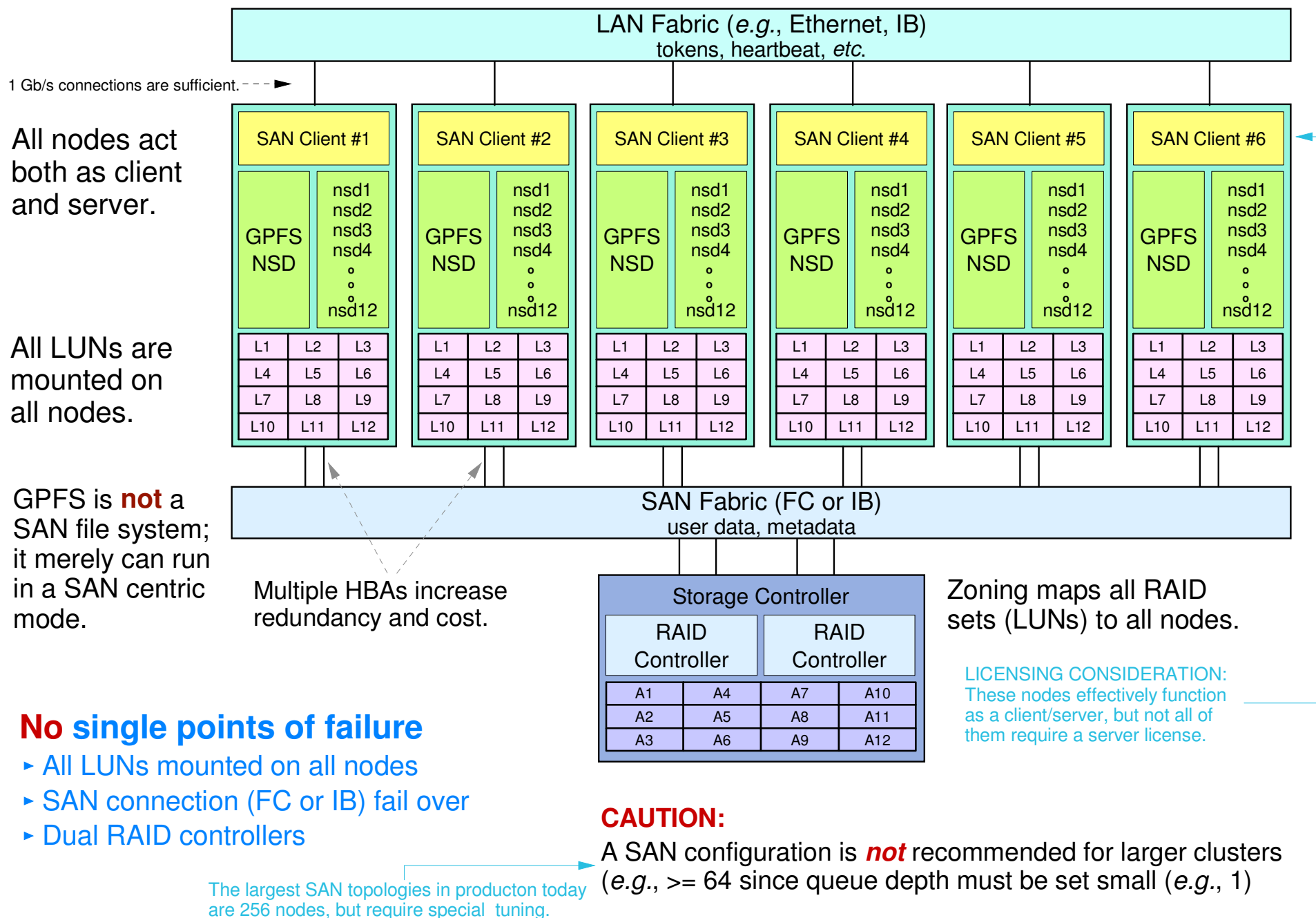
Twin Tailing

- ▶ For redundancy, each RAID array is zoned to appear as a LUN on 2 or more hosts.



Storage Area Network (SAN) Topology

Client/Servers Access Disk via the SAN





Comparing LAN and SAN Topologies

■ LAN Topology

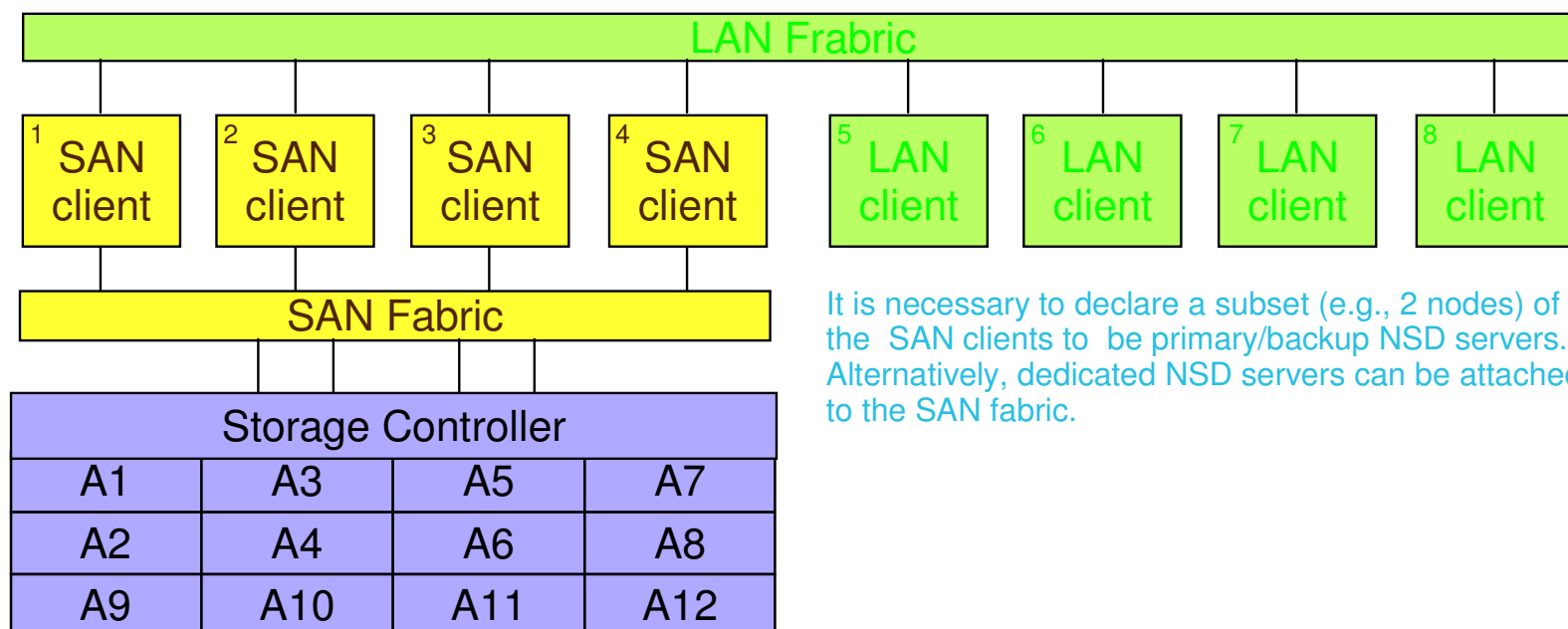
- All GPFS traffic (user data, metadata, overhead) traverses LAN fabric
- Disks attach only to servers (also called NSD servers)
- Applications generally run only on the clients (also called GPFS clients); however, applications can also run on servers
 - cycle stealing on the server can adversely affect synchronous applications
- Economically scales out to large clusters
 - ideal for an "army of ants" configuration (*i.e.*, large number of small systems)
- Potential bottleneck: LAN adapters
 - *e.g.*, GbE adapter limits peak BW per node to 80 MB/s; "channel aggregation" improves BW

■ SAN Topology

- User data and metadata only traverse SAN; only overhead data traverses the LAN
- Disks attach to all nodes in the cluster
- Applications run on all nodes in the cluster
- Works well for small clusters
 - too expensive to scale out to large clusters (*e.g.*, largest production SAN cluster is 250+ nodes)
 - ideal for a "herd of elephants" configuration (*i.e.*, small number of large systems)
- Potential bottleneck: HBA (Host Bus Adapters)
 - *e.g.*, assume 180 MB/s effect BW per 4 Gb/s HBA; multiple HBAs improves BW



Mixed LAN/SAN Topology



COMMENTS:

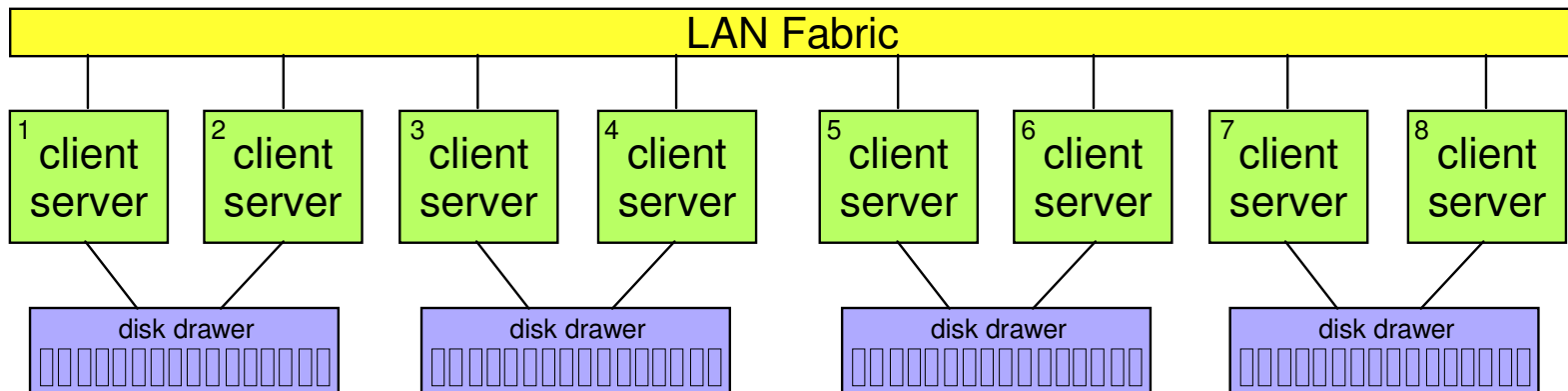
- ▶ Nodes 1 - 4 (*i.e.*, **SAN** clients)
 - GPFS operates in **SAN** mode
 - User and meta data traverse the **SAN**
 - Tokens and heartbeat traverse the **LAN**
- ▶ Nodes 5 - 8 (*i.e.*, **LAN** clients)
 - GPFS operates in **LAN** mode
 - User data, meta data, tokens, heartbeat traverse the **LAN**

COMMON EXAMPLE

- ▶ **Nodes 1 - 4**: P6p575 or P6p595
- ▶ **Nodes 5 - 8**: iDataPlex or blades



Symetric Clusters



COMMENTS

- ▶ No distinction between NSD clients and NSD servers ← Requires special bid pricing under new licensing model
 - not well suited for synchronous applications
- ▶ Provides excellent scaling and performance
- ▶ Not common today given the cost associated with disk controllers
- ▶ Use "twin tailed disk" to avoid single point of failure risks
 - does not necessarily work with any disk drawer ← New products may make this popular again.
 - do validation test first
 - example: DS3200 - yes, EXP3000 - no
- ▶ Can be done using internal SCSI
 - Problem: exposed to single point of failure risk
 - Solution: use GPFS mirroring



Which Organization is Best?

Its application/customer dependent!

Each configuration has its limitations and its strong points.
And each one is commonly used.

The following pages illustrate specific GPFS configurations.



5. Performance Features

Six related performance features in GPFS

1. Multithreading
2. Striping
3. File caching
4. Byte range locking
5. Blocks and sub-blocks
6. Access pattern optimization



Multithreaded Architecture

■ GPFS can spawn up to

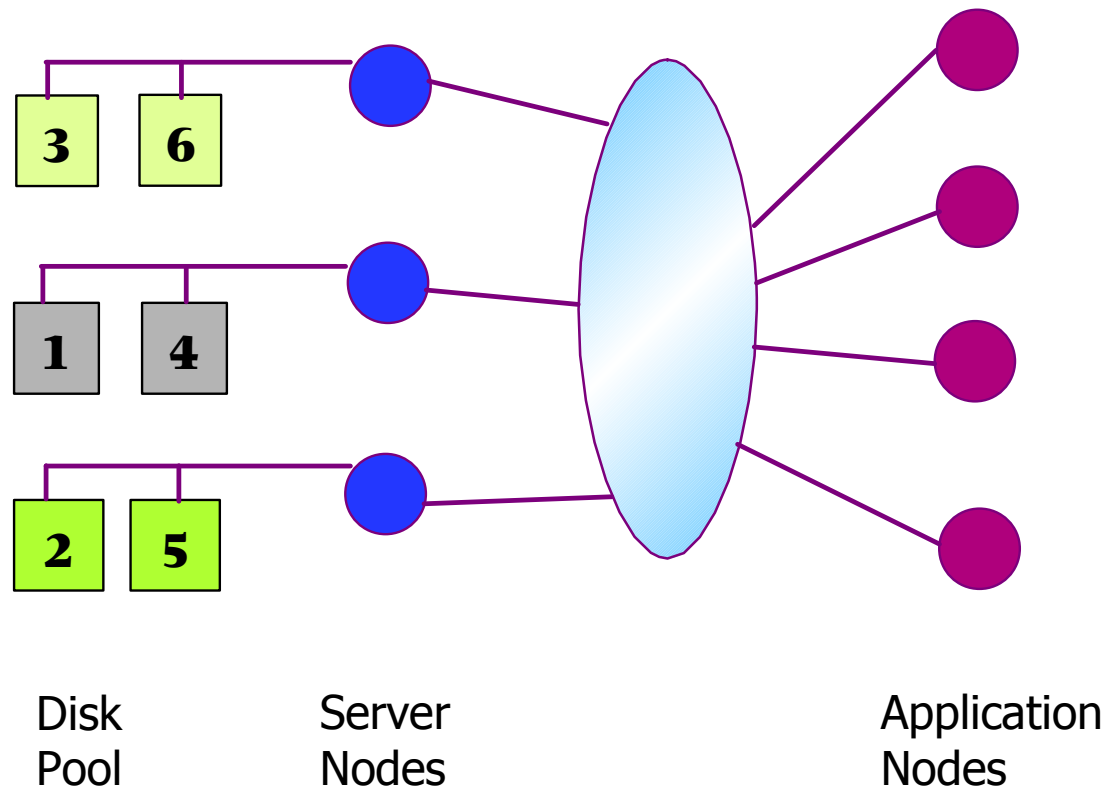
- 512 threads/node for 32 bit kernels
- 1024 threads/node for 64 bit kernels
- there is one thread per block (i.e., each block is an IOP)
 - large records may require multiple threads

■ The key to GPFS performance is "deep prefetch" which due its multithreaded architecture and is facilitated by

- GPFS pagepool
- striping (which allows multiple disks to spin simultaneously)
- access pattern optimizations for sequential and strided access or the explicit use of hints

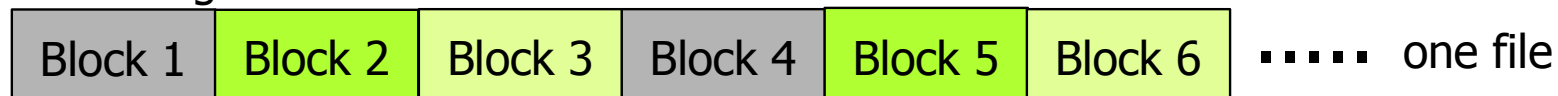


Data Striping



- GPFS *stripes* successive blocks of each file across successive disks
- Disk I/O for sequential reads and writes is done in parallel (prefetch, write behind)
- Make no assumptions about the striping pattern
- Block size is configured when file system is configured, and is *not* programmable
- transparent to programmer

increasing file offset ---->



3 I/Os executed in parallel

Job reads at 120 MB/s

Each disk reads at 40 MB/s



GPFS File Caching

Spatial vs Temporal Locality

■ Definitions

- working set: a subset of the data that is actively being used
- spatial locality: successive accesses are clustered in space (*e.g.*, seek offset)
 - this is used for predictable access patterns (*e.g.*, sequential, strided)
- temporal locality: successive accesses to the same record are clustered in time

■ To effectively exploit locality it is necessary to have a cache large enough to hold the working set.

- good spatial locality generally requires a smaller working set
 - ideally, adjacent records are accessed once and not needed again
- good temporal locality often requires a larger working set
 - the longer a block stays in cache, the more times it can be accessed without swapping

■ GPFS locality

- GPFS caching is optimized for spatial locality, but can accomodate temporal locality
- HPC applications more commonly demonstrate spatial locality

■ VM based caching systems are used in the "generic file systems"

- favors temporal locality, but can accomodate spatial locality
 - tuned using vmtune on Unix/Linux OSs
- temporal locality is common in commerical applications
- VM based caches can be as large as all free main memory
- examples: ext3, JFS, ReiserFS, XFS



GPFS File Caching

GPFS Pagepool

- ▶ What is the pagepool?
 - It is a pinned memory cache used exclusively by GPFS
 - The pagepool is independent of the VMM subsystem
 - vmtune has no direct impact on the pagepool
 - GPFS uses mmap, schmat or kernel calls to do the pinning operation
 - It is used by GPFS for file data, indirect blocks and "system metadata blocks"

- ▶ Pagepool size

- Set by `mmchconfig pagepool= {value}`

- default: 64M
 - min value: 4M
 - max value: 256G for 64 bit OS, 2G for 32 bit OS

These values are generally too small, especially for large blocks (e.g., 4M)

map blocks
inodes in transit
recovery log buffers
emergency buffers

- **BUT** GPFS will **not**

- allocate more than the pagepool parameter setting
 - allocate more than 75% of physical memory
 - request more memory than the OS will allow

This can be changed to values between 10% to 90%
e.g., `mmchconfig pagepoolMaxPhysMemPct=90`

- Optimum size of the pagepool

- Best determined empirically
 - Optimum pagepool sizing is partially workload dependent
 - File systems with a large blocksize and/or a larger number of LUNs requires a larger pagepool
 - assume blocksize <= 1 MB and number of LUNs <= 12, then let `sizeof(pagepool) <= 256 MB`
 - assume blocksize >= 2 MB and number of LUNs >= 24, then let `sizeof(pagepool) >= 512 MB`
 - Optimizing streaming access requires a smaller pagepool (e.g., up to 1 GB)
 - Optimizing irregular access requires a larger pagepool (e.g., > 1 GB, enough to hold working set)
 - Once max performance is achieved, larger pagepools yield diminishing returns

This is not an exact measurement; it is only an example.

This requires temporal locality.



GPFS File Caching

GPFS Pagepool

► Pagepool Semantics

- GPFS provides a client side caching model with cache coherency
 - Pagepool can be viewed as a single entity rather than separate caches for each node
- Regular access patterns use write-behind or prefetch caching ◀ -- spatial locality
 - Applies to sequential or other predictable access patterns
 - Write-behind: write cache policy
 - write back (write to cache only)
 - write allocate (allocate cache block before writing)
 - Prefetch: read cache policy
- Irregular (*i.e.*, random) access patterns use LRU caching ◀ -- temporal locality
- Prefetch threads
 - GPFS maintains pool of threads to be dispatched for active transactions
 - performs write behind for writers
 - performs prefetch for readers ◀ -- Despite the name, "prefetch" threads perform both write and read tasks.
 - Set by `mmchconfig prefetchThreads={value}`
- Miscellaneous observations
 - Pagepool creates implicit asynchronous operation
 - It is an open question as to whether POSIX AIO provides additional benefit under GPFS.
 - GPFS write operations are atomic
 - Given this and atomic writes, you can avoid accessing partially written records or corrupting records by having 2 tasks write to it at the same time.



GPFS File Caching

GPFS Pagepool

► Determine targeted portion of cache space used for streaming access

- `mmchconfig prefetchPct={ integer <= 70 }` ← Do not make this too large; remember, metadata access is random!
- default = 20% (e.g., 200 MB if the pagepool is 1 GB)
- uniformly distribute cache over active **sequential** streams
- adaptively determine if a stream is sequential ← -allocate 1 buffer for first access
-allocate 2 buffers if 2nd access is sequential
-allocate more buffers up to the target for continued sequential access
- Allocate buffers for each filesystem with active **sequential** streams

- adaptively determine the target number of buffers per active stream

- `sizeof(buffer)` = GPFS blocksize

- D_{FS} = desired number of buffers per file system = $\min(v1, v2) + v3$

To fully utilize a NIC, set
 $\text{maxMBpS} = 2 * \text{NIC_speed}$

$v1 = 2 * \text{LUNs} * \text{factorPerBlockSize}$ ← This is a scaling factor from a lookup table to compensate for non-linear scaling associated with block sizes.

→ $v2 = \text{maxMBpS} * t_{i0} / \text{blocksize}$ where t_{i0} = avg time over last 16 **sequential** transactions

$v3$ = number of the active **sequential** streams

- determine the allowed buffer count per file system (A_{FS})

- $A_{FS} = \min(v4, D_{FS})$

$v4 = \text{sizeof(pagepool)} * \text{prefetchPct} / \text{number of filesystems} / \text{blocksize(FS)}$

- determine the target number of buffers per active **sequential** stream per file system

- $T_B = A_{FS} / \text{number of streams for this file system}$

- Active streams receive 1 to A_{FS} buffers as they demonstrate **sequential** access

- a reader with A_{FS} buffers can not receive a new buffer until it has consumed an old buffer
- a writer with A_{FS} buffers can not dirty a new buffer until it's completed a write-behind on a old buffer
- once a buffer transaction is done, the buffer is placed on the "done list" where it is recycled

- If a stream (i.e., **sequential** user) becomes random or inactive after 5 seconds, then its buffers are disowned and given a LRU status where they "age out"

Sequential means "strictly sequential", but these algorithms can be *adapted* to other regular (i.e., "predictable") access patterns.



GPFS File Caching

GPFS Pagepool

- ▶ GPFS recognizes the following regular access patterns and sets appropriate prefetching strategies.

- Regular Access Patterns

- **Sequential:** This is a strictly sequential pattern.
- **Fuzzy Sequential:** The `nfsPrefetchStrategy` parameter defines a window of 3 to 12 (the default value is 2) contiguous blocks that can be accessed out of order, but cached using write-behind/prefetch semantics (except that write-behind buffers are returned to the LRU pool). While intended to handle out of order NFS accesses (due to thread scheduling of `nfsd` workers) this algorithm will work with any access pattern demonstrating similar locality).
- **Strided:** This applies to records of the same size with a consistent offset (forward or backward, including backward sequential) from the previous record. Prefetch threads only access the sectors encapsulating the record.
- **Mmap Strided:** Applies where a small set of contiguous pages are accessed that are roughly the same length before each "gap". The prefetch algorithm tries to predict how many pages will be needed for the next stride, but only works within a single GPFS block at a time.
- **Multi-block Random:** Applies when 3 or more blocks are accessed in one request that is not sequential. The prefetch algorithm will be applied to the blocks after the first block up to the end of the request.
 - example: if `blocksize = 256K` and `recordsize = 1024K`, then `access = multi-block random`
- **User Defined:** Apply the prefetch algorithm to records allocated via the GPFS multiple access hint (discussed later).

- Irregular (*i.e.*, random)

- Apply LRU (Least Recently Used) caching to access patterns not included in the previous list.
- Access only the sectors encapsulating the record.

Remember that despite its name, the *prefetch* algorithm applies to *both* write-behind and read-prefetch.



GPFS File Caching

GPFS Pagepool

► Miscellaneous observations

– If you change **both** the maxblocksize and pagepool parameters at the same time

- specify pagepool first if you increase the values
- specify maxblocksize first if you decrease the values

– Large pagepools are most helpful when

- writes can overlap computation
- heavy reuse of file records (*n.b.*, good temporal locality)
- semi-random access patterns with acceptable temporal locality
- a GPFS node is used as a login node or an NFS server for large clusters

– Pagepool size on NSD servers

- general principle:

NSD servers do **not** cache data; they use the pagepool for transient buffers

- Formula

pagepool_size = largest blocksize * NSDThreads ← -- Largest blocksize is not necessarily the same as the maxblocksize. This the largest blocksize from any GPFS file system on the cluster.

NSDThreads = min(A1, max(A2, A3))

A1 = nsdMaxWorkerThreads

A2 = nsdMinWorkerThreads ← -- Determine these parameters as follows: mmfsadm dump config | grep -i nsd

A3 = K * nsdThreadsPerDisk

K = number of LUNs per NSD server

- Heuristic: **Don't worry about it!** Pick a value that is not too large (e.g., 64 to 128 MB)

COMMENT:

This NSD server issue is most important for application environments where some subset of the application nodes have larger pagepools. Since the pagepool is easy to change, empirical methods can also be used to determine an optimum setting.



GPFS File Caching

i-node and Stat Cache

i-node cache

- part of the shared segment
- size = $\text{maxFilesToCache} * 2.5 \text{ KB}$
- $1 \leq \text{maxFilesToCache} \leq 100,000$
 - set large enough to accomodate the number of concurrently open files plus caching for recently used files
 - the default is 1000, but a value as small as 200 is adequate for traditional HPC applications
 - larger values (e.g., 1000) may improve performance on systems with many small files
 - larger values (e.g., 1000) are needed for a GPFS node used as a login node or an NFS server for large clusters

These parameters are easy to change.
Empirical evaluation is the most effective means for determining optimum settings.

stat cache

- part of the shared segment
- size = $176 \text{ B} * \text{maxStatCache}$
- best practice: $\text{maxStatCache} \leq 100,000$
 - default = $4 * \text{maxFilesToCache}$
 - larger values are needed when a GPFS node is used as a login node or an NFS server (e.g., 50,000)
 - mmfsd will only allocate as much space as it thinks is safe; if an excessive request is made, it will request at most $4 * \text{maxFilesToCache}$. This is at best only a heuristic algorithm.
- Avoid setting this value unnecessarily large. Remember that it only is helpful where temporal locality of stat operations (e.g., `ls -l`) can be exploited.

According to the GPFS documentation, this value can be set as large as 10,000,000 (n.b., 1.7 GB), but such a large value will exceed the shared segment size.

The Shared Segment

- memory shared by the GPFS daemon and the OS kernel
 - AIX: a 256M segment of unpinned memory
 - Linux: vmalloc space (n.b., set by a boot parameter) which is pinned memory
 - GPFS uses at most 80% of the shared segment
 - use the `mmfsadm dump fs` command for the calculation of how much will fit in the shared segment
 - If `maxFilesToCache` or `maxStatCache` are set too large, mmfsd will not start.



Supporting Parallel File Access from Multiple Nodes

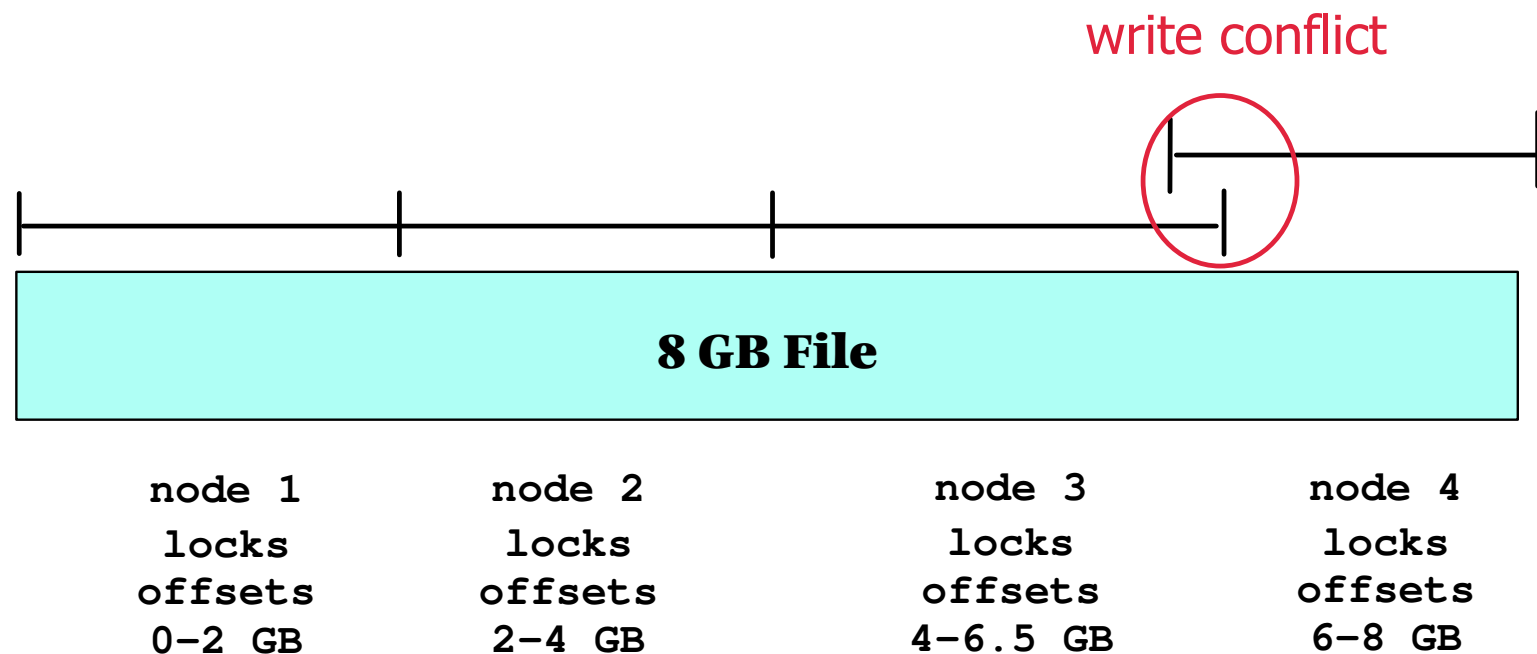
Traditionally file systems have allowed safe concurrent access to a single file from multiple tasks, but only with one task at a time. This was inefficient.

GPFS provides a finer grained approach to this allowing multiple tasks to read and write to a file at the same time. GPFS does this using a feature called "byte range locking" which is facilitated by tokens.



Byte Range Locking

- GPFS allows parallel applications on multiple nodes to access non-overlapping ranges of file without conflict or performance loss
- But byte range locks serialize access to overlapping ranges of a file



- byte range locks preserve data integrity
- byte range locks are transparent to the user
- byte range lock patterns can be much more intricate



Token Management

■ Byte range locking facilitated by tokens

- A task can access a byte range within a file (*e.g.*, read or write) iff it holds a token for that byte range.

■ Token management is distributed between 2 components

■ Token Server

- There can be 1 or more nodes acting as token servers
 - distributedTokenServer = yes by default (see mmchconfig)
 - designate multiple manager nodes (using mmcrcluster or mmchnode)
 - token load is uniformly distributed over the manager nodes
 - 1 token manager can process $\geq 500,000$ "tokens" using default settings
 - total tokens = number of nodes * (maxFilesToCache + maxStatCache) + all currently open files
- Manages tokens on behalf of a particular file system
 - distributes tokens to requesting token clients
 - distributes lists of nodes to token clients requesting conflicting tokens
- Tokens are processed via the kernel

tokenMemLimit controls the number of tokens per token manager. Default is 512M. As a rule of thumb allow for ~ 600 bytes of token per file per node. In this context, each token is a set of tokens adding up to 600 bytes.

EXAMPLE: Using default settings, a cluster with 256 nodes will have more than 1,200,000 tokens.

■ Token Client

- There is one token client per node per file system running on behalf of all application tasks running on that node
- It requests/holds/releases tokens on behalf of the task accessing the file



Token Management

■ The Process

- Offload as much work as possible from the token manager
- Token semantics
 - tokens allow either read or write access within a byte range
 - token manager responsibility
 - "coordinates" access to files
 - distributes tokens or a list of nodes holding conflicting tokens to requesting token clients
 - token client responsibility
 - token clients act on behalf of their tasks (*n.b.*, token operations are blind to the application programmer)
 - a task can access a given byte range without further access to the token manager once it has received a token until another task attempts to access the same byte range
 - the task requests other tasks holding conflicting tokens to release their tokens
 - the task releases a token to the token manager at the request of another task, but it will not do this until it has released the byte range lock for the file (this may include waiting for an I/O operation to complete)

Reality is far more complex. Tokens are associated with lock objects. Tokens support 12 modes of access and there are 7 lock object types. As a "rule of thumb" allow for about 600 bytes of token (e.g., typically 3 tokens) per file per node.

■ COMMENTS

- Accessing overlapping byte ranges where a file is being modified will **sequentialize file operations** (*n.b.*, this contributes to Amdahl inefficiency)
- GPFS write operations are atomic
- There are 9 classes of tokens in GPFS, but an open file on any node will generally have only 3 classes of tokens associated with it for ~ 600 bytes per file per node.



Token Management: FAQs

■ FAQs

- What happens if a manager node fails that is running a token server?
 - A new token server will be automatically spawned on a manager node.
 - selected from the list of manager nodes specified by mmcrcluster, mmchnode
 - if there are no free manager nodes, GPFS will redistribute the tokens across the existing manager nodes
 - File operations are suspended until the new token server is ready.
 - The new token server re-creates its token set by collecting the token state from each node in the cluster.
 - If there are multiple manager nodes are running token servers, a simple algorithm using token IDs sort out which tokens belong to which server.
- What happens if a node or task fails that holds byte range locks?
 - A log corresponding to the failed node is re-played
 - metadata is restored to consistent state
 - locks are released
- How long does token server recovery take?
 - Many variables to consider.
 - complexity of token state
 - network design and robustness
 - example: 10's of minutes in extreme cases (e.g., cluster with 4000 nodes)



Access Patterns

An application's I/O access pattern describes its I/O transaction sizes and the order in which they are accessed. It is determined by *both* the application and the file system.

Sequentially accessed large application records based on large file system blocks provide the best performance for GPFS (as well as any other file system), but applications can not always do I/O this way.

Let's examine GPFS features, tuning and best practices that can determine and compensate (to varying degrees) for access pattern variations.



GPFS Blocks

■ What is a block?

- The largest "chunk" of contiguous data in a GPFS file system
- The largest "transfer" unit in a GPFS file system
- If $\text{sizeof}(\text{record}) \geq \text{sizeof}(\text{block})$, then GPFS will simultaneously access multiple blocks for that transaction
 - example: if $\text{sizeof}(\text{record}) = 4 \text{ MB}$ and $\text{sizeof}(\text{block}) = 1 \text{ MB}$, then this transaction will result in 4 simultaneous GPFS IOPs

■ Supported block sizes

- 4MB, 2MB, 1MB, 512 KB, 256 KB, 128 KB, 64 KB, 16 KB
- A large blocksize optimizes performance when large record accesses are common (by reducing the number of IOPs)

Blocksize (KB)	128	256	512	1024	2048	4096
write (MB/s)	611.0	1226.1	2344.2	4137.9	5482.5	5477.6
read (MB/s)	502.3	1041.4	1988.7	2994.7	3418.1	3630.6

Nodes(4): P6p520, RAM = 8G, 2 x FC8, 2 x TbE, 2 x HCA (12xDDR)

DCS9900: SATA, 64 tiers, cache size = GPFS blocksize, cache writeback = ON, cache prefetch = 0, NCQ = OFF

GPFS: blocksize = DCS9900 cache size, block allocation = scatter, pagepool = 4 GB, maxMBpS = 4000

Application: 16 tasks, record size = GPFS blocksize, file size = 256 GB



GPFS Sub-blocks

■ GPFS blocks can be divided into 32 sub-blocks

- A sub-block is the smallest "chunk" of contiguous data in a GPFS file system
 - a file smaller than a sub-block will occupy the entire sub-block
 - large files begin on block boundary
 - files smaller than a block can be stored in fragments of 1 or more sub-blocks
 - files larger than a sub-block have very little internal fragmentation

■ Sub-blocks vs. sectors

- A sector (512 bytes) is the smallest "transfer" unit
 - e.g., a 1-byte read request will result in a 512 byte transfer
- If the access pattern is irregular, the record sizes are smaller than a block, and the data is not in the pagepool, then GPFS will access only the sectors that enclose the record
- If a file smaller than a block is accessed in a single transaction, then GPFS will access only the sectors that enclose the file

■ Caching small irregular transactions

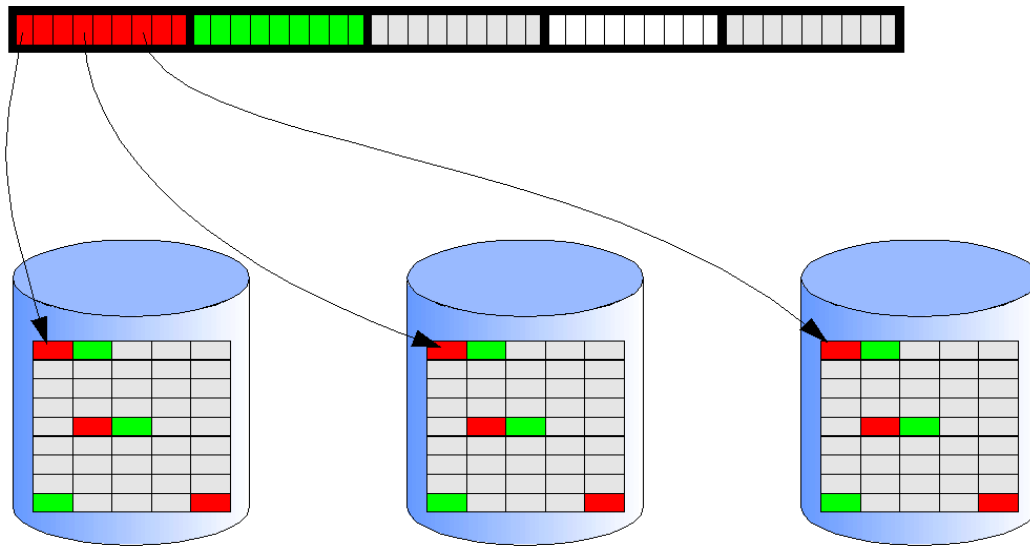
- Suppose $\text{sizeof}(\text{record}) < \text{sizeof}(\text{block})$
- Suppose that the record is in the pagepool
- If a second record is accessed within the same block, then
 - if "time" ≤ 32 , then GPFS will access the entire block
 - if "time" > 32 , then GPFS flushes cache and accesses the sectors of the new record only
 - "time" is measured by a random access counter differential
 - a global counter is bumped every time there is a random access to any file in the file system
 - a local counter for a given block in the pagepool is initialized to the global value upon the first random access to that block
 - time is the difference between the global and local counter on the next random access to a given block



Allocation Map and Allocation Regions

Segmented Block Allocation MAP:

- map types: scatter (default), cluster
- selected using `mmcrfs -j` parameter



- Each segment contains bits representing blocks on all disks
- Each segment is a separately lockable unit
- Minimizes conflicts between multiple writers
- Allocation manager provides hints which segments to try
- $\text{sizeof}(\text{segment}) < \text{blocksize}$

Allocation Regions

- The block allocation map is divided into k regions where $k > 32 * \text{number of nodes}$
 - the value of k is based on the number of nodes estimate from `mmcrfs -n` parameter
 - there are at least 32 allocation regions per node
 - there is one or more allocation map segments per allocation region
- Guarantees there are 1 or more allocation regions per node if file system $< 97\%$ capacity
 - if `mmcrfs -n` is set too small, nodes run out of allocation regions prematurely
 - nodes start sharing allocation regions which hurts performance
 - **WARNING:** it is not easy to change the `mmcrfs -n` setting... get it right the first time!



Block Allocation

Block Allocation Map Type

► File data distribution

- GPFS distributes file blocks to a file system's LUNs in a round-robin pattern
- file blocks are then distributed across each LUN according to the block allocation map type

► Type: scatter

- randomly distribute file blocks over the LUN (*i.e.*, scattered over the disk)
- guarantees uniform performance of multitask jobs accessing a common file
 - compensates for Poisson arrivals
- default = scatter if number of nodes > 8 or number of LUNs > 8

[Scatter] won't get you the best possible performance out of the disk subsystem, but it also avoids getting the worst of it.

Yuri Volobuev

► Type: cluster

- write file data in clusters of contiguous disk blocks (*i.e.*, clustered together on the disk)
- yields better performance in restricted circumstances
 - "small" clusters and/or "small" file systems
 - "small" transactions (*e.g.*, 4K)
- default = cluster if number of nodes ≤ 8 and number of LUNs ≤ 8

► COMMENT:

- There is no guarantee that contiguous file blocks on disk will be accessed in the same order that they are mapped to disk. Factors contributing to the randomness of "arrivals" include
 - a larger number of tasks and/or nodes simultaneously accessing a file
 - a larger number of files simultaneously being accessed
 - the stochastic nature of queueing systems
- Given the variabilities of clustered block allocation, validation testing is recommended before adopting it.

WARNING: This parameter can only be changed with a destructive rebuild.

► Example

blocksize	block allocation		write - number of nodes					read - number of nodes				
			1	2	4	8	14	1	2	4	8	14
256 KB	scatter	MB/s	554	780	799	799	808	635	818	1026	1180	1205
256 KB	cluster	MB/s	554	1074	1117	1091	1078	799	1321	1510	1485	1470
1024 KB	scatter	MB/s	624	1145	1145	1145	1142	661	1321	1527	1535	1541
1024 KB	cluster	MB/s	624	1145	1145	1145	1145	661	1347	1561	1561	1551

transaction size		nodes/tasks	
		write	read
		4/16	4/16
4 KB ¹	IOP/s	51931	21664
4 KB ¹	IOP/s	48113	37001
4096 KB ²	MB/s	5576	3474
4096 KB ²	MB/s	5394	4147

Hartner, *et. al.* Sequential I/O Performance of GPFS on HS20 Blades and DS4800 Storage Server. *Technical Report.*, IBM.
22 x 4+P RAID 5 arrays, 14 x HS20 blades using GPFS as a SAN over FC4

DCS9900, 64 SATA tiers
4 x P6p520 nodes
GPFS SAN over FC8

1. blocksize = 256 KB
2. blocksize = 4096 KB



Well Formed I/O

■ I/O transactions are *well formed* if the application record is aligned with GPFS block or sub-block boundaries.

- Records \geq blocksize
 - $\text{seek_offset} \% \text{blocksize} = 0$
 - $\text{record_size} / \text{blocksize} = k$, where k is an integer > 0
- Records $<$ blocksize
 - $\text{seek_offset} \% \text{sub-blocksize} = 0$
 - $\text{record_size} / \text{sub-blocksize} = k$, where k is an integer > 0

■ Caveates and Warnings

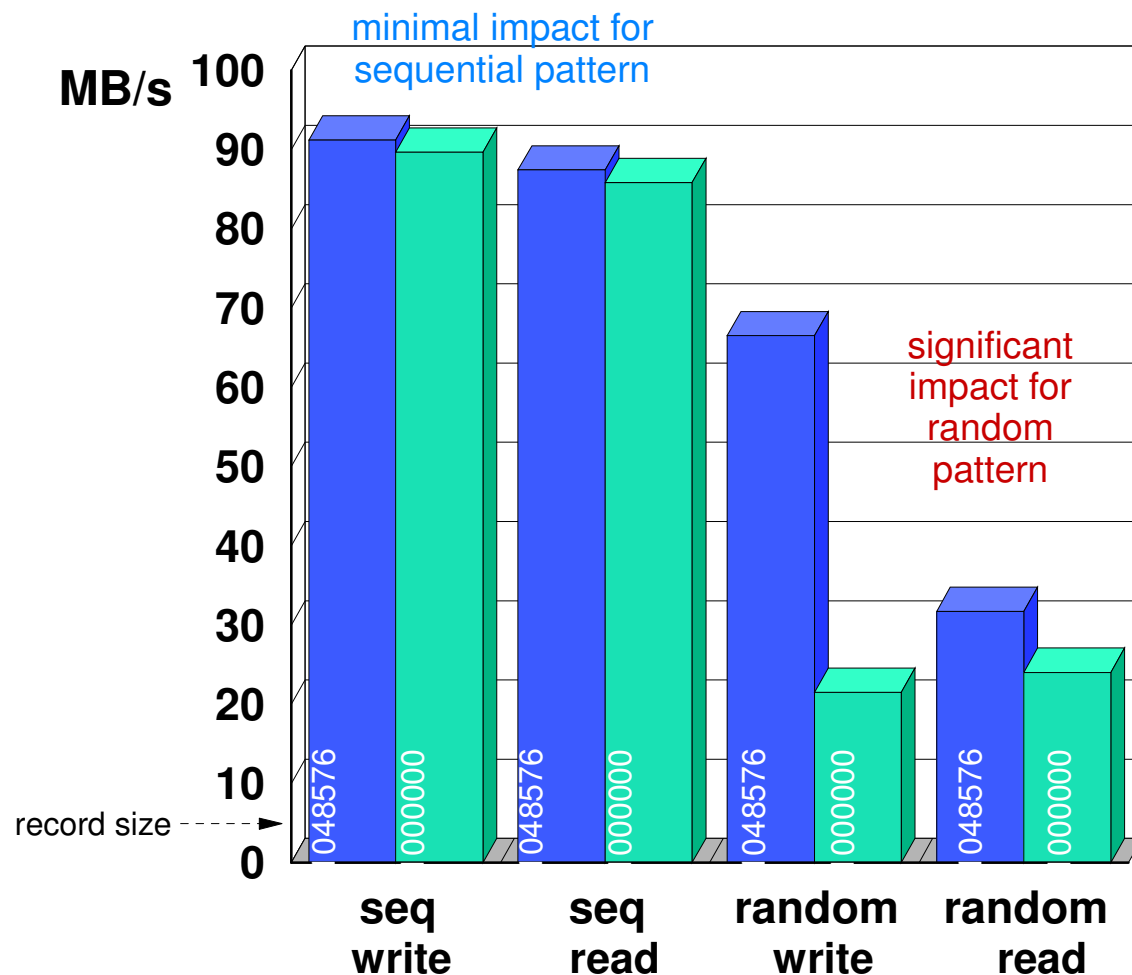
- Sequential access compensates for non-well formed I/O
 - full blocks will be preallocated in cache upon write and prefetched upon read
- Irregular access patterns with non-well formed IO will often require extra IOPs
 - Example (2^n vs. 10^n): If $\text{record_size} = 1000000$ and $\text{blocksize} = 1048576$, then each record will generally span 2 blocks requiring 2 IOPs to read 1 record that fits in 1 block. If the application reads a full block (*i.e.*, 1048576), it will have significantly improved performance even if it does not use all of the data it reads.

■ COMMENT

- $\text{seek_offset} = 0$ is well formed in GPFS



Well Formed I/O Benchmark Example



■ well formed
■ non well formed

COMMENT

- ▶ Well formed I/O has a *minimal* impact on a sequential access pattern.
- ▶ Well formed I/O has a *significant* impact on random access pattern since it must do nearly 2X more IOPs when its transactions are not well formed.

App: 4 tasks, 2 nodes, record size = variable, file size = 8 GB

GPFS: version 3.2, blocksize = 1 MB, pagepool = 256 MB

Nodes: P4p615, 2 cores, 4 GB RAM

Disk: SAN attached SSA (16 disks @ 10 Krpm), JBOD



Direct I/O

■ Direct I/O

- Open the file the O_DIRECT flag
 - this flag is considered advisory, not mandatory
 - the FS can ignore it, but GPFS accepts it (n.b., "buyer beware!")
- I/O Buffer must be memory page aligned
 - for most systems, 4 KB alignment (*i.e.*, $\text{buffer_address} \% 4096 = 0$)
- Seek offset must be sector aligned for GPFS
 - 512 B alignment (*i.e.*, $\text{seek_offset} \% 512 = 0$)
 - *n.b.*, most file systems require 4K alignment Direct I/O
- Direct I/O bypasses the FS cache mechanism; therefore, the programmer must compensate by manually doing the aligning.

COMMENT:

Use Direct I/O when the GPFS caching mechanism can not compensate for the access pattern. This is not trivial!

■ EXAMPLE:

```
int      bsz;          /* size of record */
off_t    soff = 0;     /* seek offset */
char     *buf;         /* 4K aligned buffer */
void     *b1;          /* needed for pointer swizzling */
unsigned b2;           /* needed for pointer swizzling */
. . . . .
b1 = malloc(bsz + 4096);
b2 = (unsigned)b1;
b2 = b2 & (unsigned)0xFFFFF000;
b2 = b2 + (unsigned)0x00001000;
buf = (char*)b2;
if (bsz%512 != 0) printf("ERROR:  buffer is not block aligned\n");
else             soff += (off_t)bsz;
```




Common Application Access Patterns

■ Streaming

- records are accessed once and not needed again
- generally the file size is quite large (e.g., GB or more)
- good spatial locality occurs if records are adjacent
- performance is measured by BW (e.g., MB/s, GB/s)
- operation counts are low compared to BW
- most common in digital media, HPC, scientific/technical applications

COMMENT:

These access patterns are from the application's perspective. The actual access pattern on the media is also determined by the file system and storage controller architecture and configuration.

■ IOP Processing

- small transactions (*i.e.*, less than FS block size)
 - small records irregularly distributed over the seek offset space
 - small files
- poor spatial locality and often poor temporal locality
- performance is measured in operation rates (e.g., IOP/s)
- operation counts are high compared to BW
- common examples: bio-informatics, EDA, rendering, home directories

■ Transaction Processing

- small transactions (*i.e.*, files or records less than the blocksize), but often displaying good temporal locality
 - access efficiency can often be improved by database technology
- performance is measured in operation rates (e.g., IOP/s)
- operation counts are high compared to BW
- common examples: commercial applications



Access Pattern Optimizations

GPFS uses cache to improve the performance of various access patterns:

- sequential (write and read, forward and backward)
- strided (write and read, forward and backward)
- small file (write only)

If the pattern is recognized, then the relevant records can be asynchronously pre-loaded into cache.

If the access pattern is not recognized by GPFS, then hints can be provided informing GPFS which records can be pre-loaded into cache.

COMMENT: These optimizations assume spatial locality



Sequential Access Pattern

When GPFS detects a forward or backward sequential order, it either preallocates cache blocks on write or prefetches disk blocks on reads generating the peak performance sustainable by the disk controller.

App: 4 tasks, 2 nodes, record size = 1 MB, file size = 8 GB, well formed I/O

GPFS: version 3.2, blocksize = 1 MB, pagepool = 1 GB

Nodes: P4p615, 2 cores, 4 GB RAM

Disk: SAN attached SSA (16 disks @ 10 Krpm), JBOD

Access Pattern	Write	Read
Sequential	85.8 MB/s	88.8 MB/s
Backward Sequential	89.4 MB/s	93.0 MB/s
Random	37.8 MB/s	32.8 MB/s

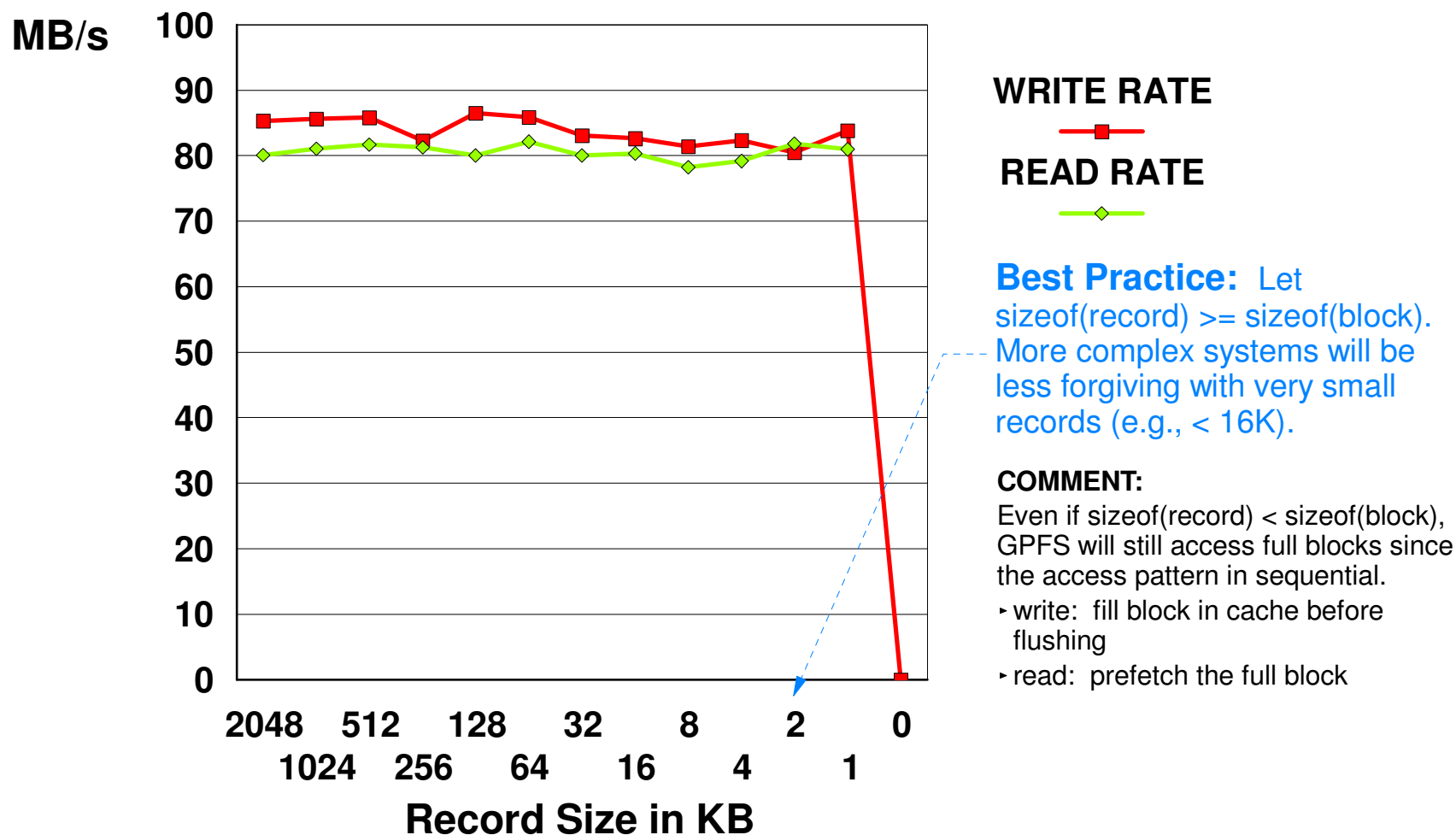
← -- minimal caching



Sequential Access Pattern

GPFS Blocksize vs. Application Record Size*

*I/O is well formed.



App: 4 tasks, 2 nodes, record size = variable, file size = 2 GB, well formed I/O

GPFS: version 3.2, blocksize = 256 KB, pagepool = 256 MB

Nodes: P4p615, 2 cores, 4 GB RAM

Disk: SAN attached SSA (16 disks @ 10 Krpm), JBOD



Strided Access Pattern

When GPFS detects a strided order, it prefetches along the stride thus improving performance.



8 tasks @ 1 per node, blocksize = 256 KB, record size = 16 KB, file size = 5 GB
WH2 with 14 clients and 2 VSD servers, using 36 GB, 10 Krpm, SSA drives

	Write	Read
strides under GPFS 1.2	less than 1 MB/s*	11.1 MB/s*
strides under GPFS 1.3	17 MB/s	58 MB/s

* The strided rate under GPFS 1.2 is the same as the random (without hints) rate under GPFS 1.3.



Improving Strided Access Rates

But notice that increasing the record size from 16KB to 1024 KB, the rates increase.

8 tasks @ 1 per node, blocksize = 256 KB, file size \geq 5 GB

WH2 with 14 clients and 2 VSD servers, using 36 GB, 10 Krpm, SSA drives

	Write	Read
record size = 1024 KB	172 MB/s	211 MB/s
record size = 16 KB	17 MB/s	58 MB/s



Irregular Access Patterns Are Slow

An irregular access pattern does not allow GPFS to anticipate the seek pattern. Therefore it can not prefetch records for reading or preallocate cache blocks for writing.





POSIX I/O is a simple standard covering the basics. Early versions of GPFS stuck closely to this standard. But because of its shortcomings in many environments, IBM has added API extensions to GPFS that go beyond the POSIX I/O API.

These extensions are a mixed blessing. While they improve performance and facilitate important semantics not part of POSIX I/O, they are generally not portable.



GPFS Multiple Access Hint

Improving Random I/O



- ▶ Some applications are intrinsically based on small records
 - sorting jobs
 - dmostack in seismic processing
- ▶ The GPFS multiple access hint allows the programmer to post future accesses and then prefetches them asynchronously.
- ▶ Reads are improved substantially, writes not as much.

	write rate	read rate
without hints	33.1 MB/s	18.5 MB/s
using hints*	38.8 MB/s	63.7 MB/s

★ The impact of using hints is more significant given a larger number of nodes.

App: 8 tasks, 2 nodes, record size = 128 KB file size = 2 GB, well formed I/O

GPFS: version 3.2, blocksize = 256 KB, pagepool = 256 MB

Nodes: P4p615, 2 cores, 4 GB RAM

Disk: SAN attached SSA (16 disks @ 10 Krpm), JBOD

- ▶ The multiple access hint interface is tedious to use, but a simple to use interface can be crafted.



GPFS Multiple Access Hint: An Example of a Simple Interface

A simple GPFS multiple access hint interface can be designed by the user (hiding the low level tedium) making it easier for high level applications to use hints.

For example...

public:

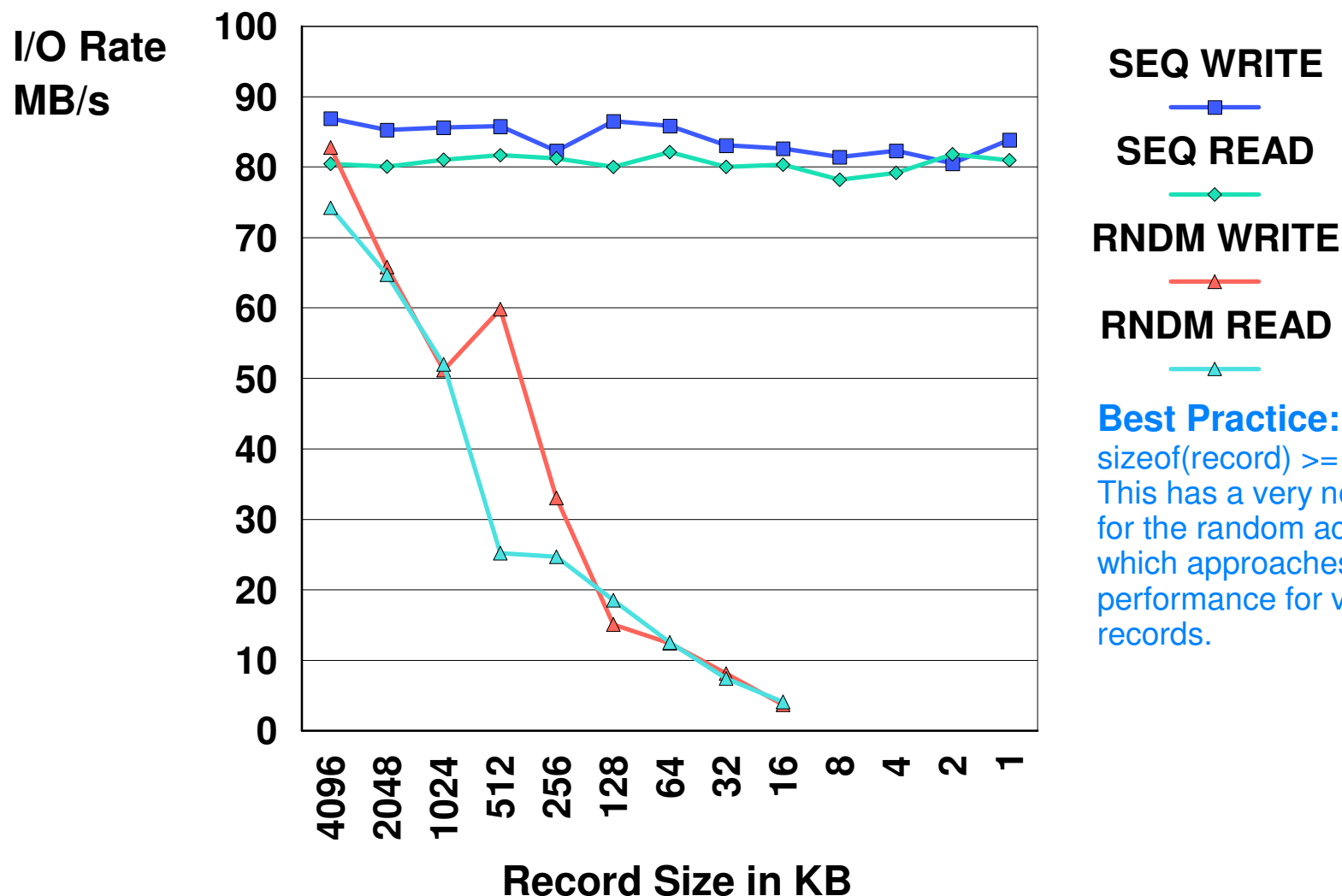
```
int pio_init_hint(struct pio *p, int maxbsz, int maxhint);  
int pio_post_hint(struct pio *p, offset_t soff, int nbytes, int nth, int isWrite);  
int pio_declare_1st_hint(struct pio *p);  
int pio_xfer(struct pio* p, char* buf, int nth);
```

private:

```
int pio_gen_blk(struct pio *p, int nth, int isWrite);  
int pio_issue_hint(struct pio*p, int nth);  
int pio_cancel_hint(int fd);
```



PUTTING IT ALL TOGETHER: Block Size vs. Record Size vs. Caching



App: 4 tasks, 2 nodes, record size = variable, file size = 2 GB, well formed I/O
GPFS: version 3.2, blocksize = 256 KB, pagepool = 256 MB
Nodes: P4p615, 2 cores, 4 GB RAM
Disk: SAN attached SSA (16 disks @ 10 Krpm), JBOD



Small Files

■ Small Files

- Increasingly common in clusters today
- Small blocks work best when the average file size is small (e.g., less than 256K)
- But do not make GPFS blocks too small
 - select blocksize so that the sub-block size \approx average file size
 - reduces internal fragmentation
 - produces optimum small file performance
 - block still large enough to support larger files (not every file will be small)
- Small file optimization
 - allocate small files "close together" by filling one full block on one disk before moving to the next
 - flushes small files to disk as individual small IOPs (i.e., in units of sub-blocks)
 - use controller cache to block the small IOPs into larger transaction
 - produces 7.2% improvement in DS4800 benchmarks ←----- Will this be better on DCS9900?
 - improves write performance, less likely to improve read performance

■ Small file access performance is not as good as streaming access

- Small file access is an IOP access pattern
- Should only be considered when all else fails



Tuning GPFS for IOP Processing

When Small Transactions Are Inevitable



COMMENT:

This slide needs further refinement and some of its guidelines need to be tested more rigorously.

■ worker1Threads

- Consider an N+P RAID set. The set
 - $\text{worker1Threads} = 2 * N * \text{number_of_LUNs}$

■ prefetchThreads

- You need roughly twice as many prefetchThreads as LUNs. Suppose you have K LUNs, then
 - $\text{prefetchThreads} = 2 * K$

■ pagepool

- Set the pagepool large enough so that 20% can hold the buffers for the prefetchThreads
 - $\text{prefetchThreads} * \text{blocksize} < 0.2 * \text{pagepool}$

■ When there are many small files...

- Increasing the maxFilesToCache and the maxStatCache may help
 - example (EDA): $\text{maxFilesToCache} = 10000$, $\text{maxStatCache} = 40000$

■ When there is good temporal locality

- $\text{pagepool} < 0.5 * \text{sizeof}(\text{memory})$
 - $\text{max pagepool} < 256 \text{ GB}$

■ Direct I/O

- Bypasses the FS cache mechanism
 - Since GPFS is optimized for large records, can this reduce overhead for small records?
- Use knowledge of application to exploit locality not detected by the file system

**COMMENT:**

This slide needs further refinement doing tests with a more genuinely random pattern for small files.



IOP Processing vs. Streaming

IOP Workload

ntasks	16
tree depth	12
files per directory	8
total directories	65520
total files	524160
record size	2K
total data	2600 MB

GPFS Config

- version 3.2
- blocksize: 256 KB
- subblocksize: 8 KB
- pagepool: 1024 MB

DS4800 Config

- 64 x 15Krpm disks
- 4+P RAID 5
- segment size: 64 KB
- cache page size: 16 KB
- read cache: ON
- read ahead: ON
- write cache: ON
- write cache mirroring: OFF

function	write	read
job time (sec)	163.9 ²	90.7
average time (sec)	112.1	86.4
total time (sec)	1793.3	1383.1
# IOPs	2,620,800	2,555,280
% directory ops	10.0%	7.7%
% open/close	40.0%	41.0%
% write	50%	
% read		51.3%
IOP rate (IOP/s)	23384 ²	29561
Data rate (MB/s)	6.0	29.6

These values may benefit unaturally from cache.³

Streaming Workload

ntasks	16
tree depth	1
files per directory	1
total directories	1
total files	1
record size	1M
total data	65536 MB

GPFS Config

- version 3.2
- blocksize: 1024 KB
- pagepool: 256 MB

DS4800 Config

- 64 x 15Krpm disks
- 4+P RAID 5
- segment size: 256 KB
- cache page size: 16 KB
- read cache: ON
- read ahead: ON
- write cache: OFF

function	write	read
job time (sec)	60.7	47.8
average time (sec)	59.0	46.3
total time (sec)	943.6	740.8
# IOPs	65540	65540
% directory ops	0%	0%
% open/close	0%	0%
% write	99.9%	
% read		99.9%
IOP rate (IOP/s)	1111	1415
Data rate (MB/s)	1111.2	1415.4

COMMENTS:

1. For the most part these are non-cached IOP rates. Moreover, the IOP rates quoted here are based on application transactions. The non-cached IOP rates quoted for storage controllers are based on consistant 4K transactions measured by the controller and not the application.
2. The write IOP rate is based on the harmonic aggregate; however, this measure is *slightly* compromised by the large job time variance of 24.1%. By comparison, the natural aggregate is 15988. The "true" IOP rate (i.e., when all 16 tasks were active) would be closer to 20000. The variance for the read rate was only 4.0%.
3. The tree traversal algorithm used by this benchmark may lend itself to an unaturally cache friendly situation not typical for many small file access patterns.

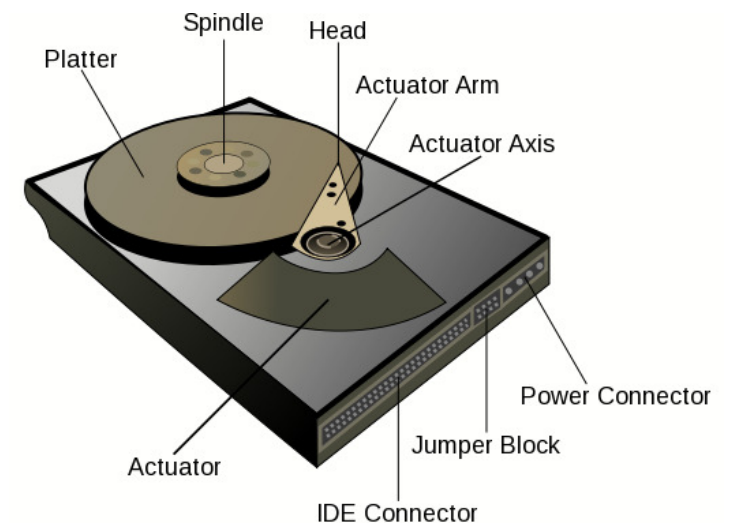


The Big Picture

Use large files and streaming access where possible.

Seek Arm Mechanics

The more data transferred for every seek arm movement the better performance is!





6. GPFS Management and Overhead Functions

In general, GPFS is designed to perform the same functions on each node and the functions performed on behalf of an application are executed on the node where it is generated.

However, there are specialized management and overhead operations which are performed globally that affect the operation of the other nodes in the cluster.



GPFS Management Functions

Like any file system, GPFS has several classes of overhead functions, but it does *not* concentrate them into a dedicated "metadata" server. Rather, it distributes them over several to many nodes reducing the impact of their overhead and risk exposure. Some aspects of these functions are concentrated on specific nodes* where the cost of its distribution outweighs its value.

■ Metanode⁺

■ Configuration Manager

- AKA "cluster configuration manager"

■ Cluster Manager

■ Manager Nodes

- File System Managers
 - File system configuration
 - Manage disk space allocation
 - Quota management
 - Security services
- Token Managers

■ Quorum Nodes

*These functions are generally overlapped with other dedicated nodes (*e.g.*, NSD servers, login nodes) though in very large clusters (*e.g.*, over 2000 nodes) this must be done carefully so that their function is not impacted by network congestion.

+Does not require a server license



Metanodes

■ Problem

- Can't afford exclusive inode lock to update file size and mtime
- Can't afford locking whole indirect blocks

■ Solution

- Metanode (one per file) collects file size, mtime/atime, and indirect block updates from other nodes

■ How it works

- Metanode is elected dynamically and can move dynamically
- Only the metanode reads & writes inode and indirect blocks
- Merges inode updates by keeping largest file size and latest mtime
- Synchronization
 - Shared write lock allows concurrent updates to file size and mtime.
 - Operations that require exact file size/mtime (e.g., stat) conflict with the shared write locks.
 - Operations that may decrease file size or mtime

■ Comments

- This is **not** a metadata server concentrating metadata operations on 1 or a small number of dedicated nodes. Rather, its a distributed algorithm processing metadata transactions across clients in the cluster.
- There is minimal overhead per metanode



Configuration Manager

- **There is a primary and backup configuration manager per GPFS cluster**
 - Specified when the cluster is created using mmcrcluster
 - Common practice
 - assign to manager nodes and/or NSD servers.
- **Function**
 - Maintains the GPFS configuration file `/var/mmfs/gen/mmsdrfs` on all nodes in the GPFS cluster.
 - This configuration file can not be updated unless both the primary and backup configuration managers are functioning.
- **Minimal overhead**



Cluster Manager

- There is one cluster manager per GPFS cluster

- Selected by election from the set of quorum nodes

- can be changed using `mmchmgr`

- **Functions**

- Monitors disk leases (*i.e.*, "heartbeat")
- Detects failures and directs recovery within a GPFS cluster
 - determines whether quorum exists
 - This guarantees that a consistent token management domain exists; if communication were lost between nodes without this rule, the cluster would become partitioned and the partition without a token manager would launch another token management domain (*i.e.*, "split brain")
- Manages communications with remote clusters
 - distributes certain configuration changes to remote clusters
 - handles GID/UID mapping requests from remote clusters
- Selects the file system manager node
 - by default, it is chosen from the set of designated manager nodes
 - choice can be overridden using **`mmchmgr`** or **`mmchconfig`** commands

- **Network Considerations**

GbE is adequate!

- Heartbeat network traffic is light and packets are small
 - default heartbeat rate = 1 disk lease / 30 sec per node
 - cluster manager for a 4000 node cluster receives 133 disk leases per second
- But network congestion must **not** be allowed to interfere with the heartbeat
 - by default, disk lease lasts 35 sec, but a node has last 5 sec to renew lease
 - best practice: assign to a lightly used or dedicated node in clusters over 1000 nodes



Manager Nodes

■ Designating manager nodes

- They are specified when a cluster is created (using mmcrcluster)
 - can be changed using mmchnode
- Can specify up to 128 manager nodes
 - If not specified, GPFS selects 1 node to be a combined file system and token manager node.

■ Function

- File system managers
- Token managers

■ Best practices

- smaller clusters (less than 1000 nodes):
 - commonly overlapped with NSD servers and/or quorum nodes
- larger clusters (more than 1000 nodes):
 - assign to lightly used or dedicated nodes
 - do **not** overlap with NSD servers

Some customers overlap quorum and manager nodes.
Be careful overlapping them with login nodes.



File System Managers

■ There is exactly one file system manager per file system

- File system managers are uniformly distributed over manager nodes
 - A file system manager never spans more than 1 node, but if there are more file systems than manager nodes, there will be multiple file system managers per manager node.
- Choice can be overridden by `mmchmgr` command
 - any node can be chosen (*n.b.*, it does not have to be a manager node)

■ There are 4 file system management functions

1. file system configuration

- adding disks
- changing disk availability
- repairing the file system
- mount/umount processing (this is also done the node requesting the operation)

2. disk space allocation management

- controls which regions of each disk are allocated to each node (striping management)

3. quota management

- enforces quotas if it has been enabled (see **mmcrfs** and **mmchfs** commands)
- allocates disk blocks to nodes writing to the file system
- generally more disk blocks are allocated than requested to reduce need for frequent requests

4. security services

- see manual for details
- some differences *appear* to exist between AIX and Linux based systems

■ Low overhead



Token Managers

■ Token managers run on manager nodes

- GPFS selects some number of manager nodes to run token managers
 - GPFS will only use manager nodes for token managers
 - the number of manager nodes selected is based on the number of GPFS client nodes
- Token state for each file system is uniformly distributed over the selected manager nodes
 - there is 1 token manager per mounted file system on each selected manager node
- 1 manager node can process $\geq 500,000$ "tokens" using default settings
 - In this context, a token is a set of several tokens ≈ 600 bytes on average.
 - total number of tokens = number of nodes * (maxFilesToCache + maxStatCache) + all currently open files
 - If the selected manager nodes can not hold all of the tokens, GPFS will revoke unused tokens, but if that does not work, the token manager will generate an ENOMEM error.

This usually happens when not enough manager nodes were designated.

■ Function

- Maintain token state (see earlier slides)

■ Overhead

- CPU usage is light
- Memory usage is light to moderate (e.g., at most 512 MB by default)
 - can be changed using mmchconfig tokenMemLimit=<value>
- Message traffic is variable, but not excessive. It is characterized by many small packets
 - If network congestion impedes token traffic, performance will be compromised, but it will not cause instability.
 - If NSD servers and GPFS clients are also used for token management, large block transfers (e.g., ≥ 512 KB) may impede token messages.

If these issues are impeding token response, chances are good that users will never notice.



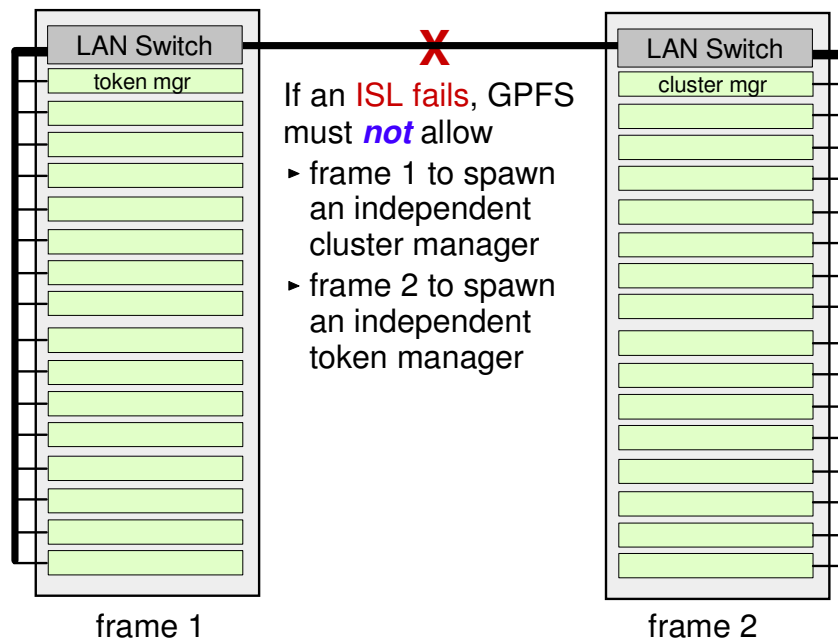
Quorum

■ Problem

- If a key resource fails (e.g., cluster manager or token manager) GPFS will spawn a new one to take over. But if the other one is not truly dead (e.g., network failure), this could create 2 independent resources and corrupt the file system.

■ Solution

- *Quorum* must be maintained to recover failing nodes.
- 2 options
 - **Node quorum (default):** must have at least 3 quorum nodes
 - **Node quorum with tiebreaker disks:** used in 1 or 2 node clusters

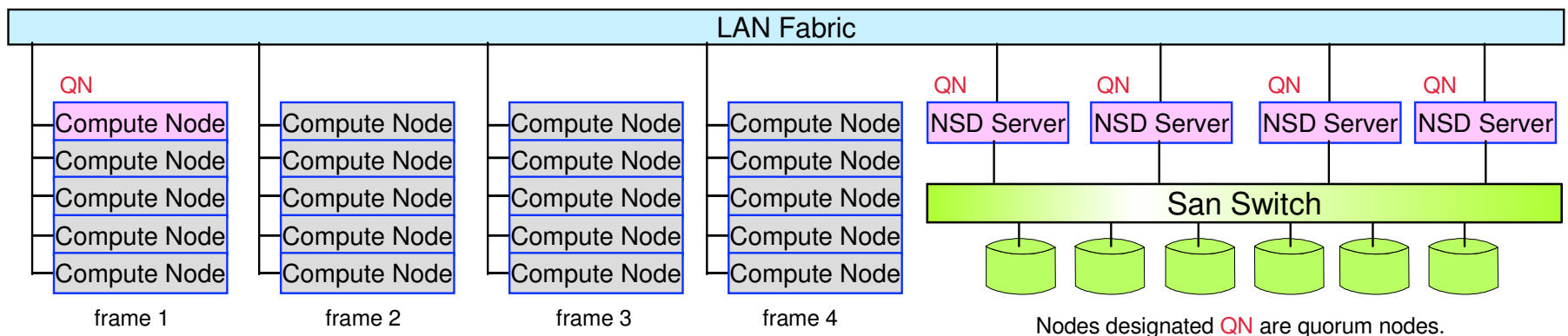




Node Quorum

How it Works

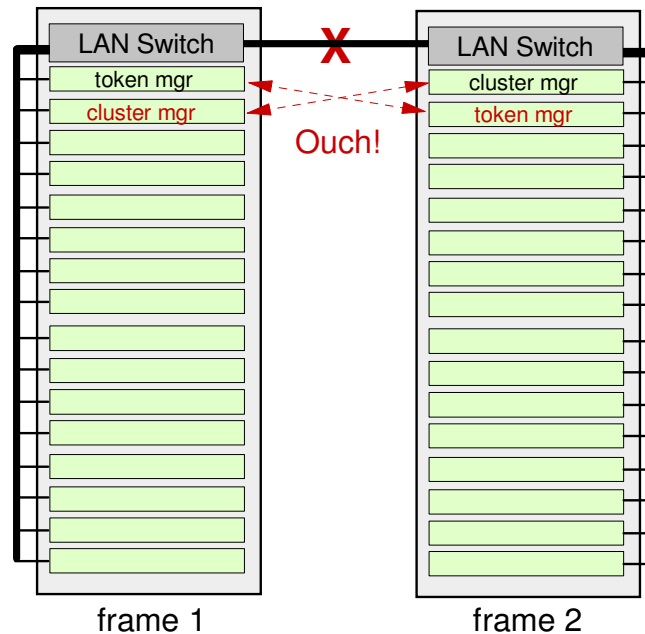
- Node quorum is defined as one plus half of the explicitly defined quorum nodes in the GPFS cluster. There are no default quorum nodes. The smallest node quorum is 3 nodes.
- Selecting quorum nodes: best practices
 - Use caution in a "five 9's" environment
 - Select nodes most apt to remain active
 - Select nodes that rely on different failure points
 - example: select nodes in different racks or on different power panels.
 - In smaller clusters (e.g., < 1000 nodes) select administrative nodes
 - common examples: NSD servers, login nodes
 - In large clusters, either select dedicated nodes or overlap with manager nodes.
 - do **not** overlap with NSD servers
 - Select an odd number of nodes (e.g., 3, 5, or 7 nodes)
 - More than 7 nodes is not necessary; it increases failure recovery time without increasing availability.





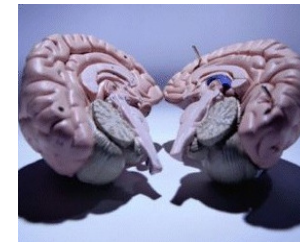
Node Quorum Example

No Quorum Rule



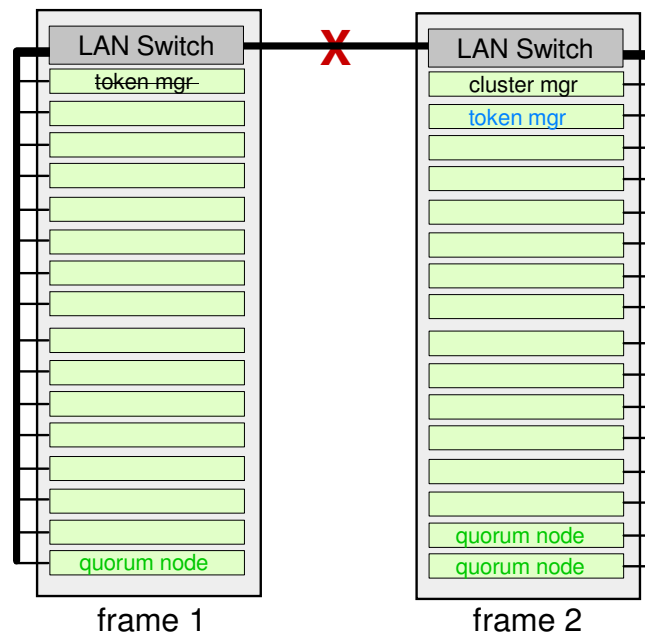
Without the quorum rule:

- ▶ frame 1 could start an independent cluster manager
- ▶ frame 2 could start an independent token manager
- ▶ file system would be corrupted!



This is sometimes called "split brain".

Maintaining Quorum



Maintaining quorum semantics

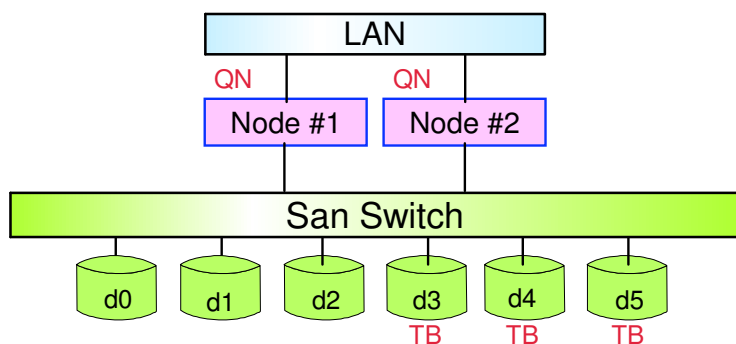
- ▶ guarantees frame 1
 - can **not** start an independent cluster manager
 - token manager remains inactive
- ▶ allows frame 2 to spawn a new token manager and remain active



Node Quorum with Tiebreaker Disks

How it Works

- **Node quorum with tiebreaker disks runs with as little as one quorum node available so long as there is access to a *majority* of the quorum disks.**
 - there can be a maximum of only 2 quorum nodes
 - the number of non-quorum nodes is unlimited (can be as small as zero)
 - there can be 1 to 3 tiebreaker disks (n.b., odd number of disks is best)
 - tiebreakers disks must be directly accessible from the quorum nodes, but do not have to belong to any particular file system
 - must have a cluster-wide NSD name as defined through the **mmcrnsd** command
 - tiebreaker disks must be SAN attached (FC or IP) or VSDs
 - same rules apply in selecting quorum nodes for both quorum options (see previous page)
 - select quorum nodes with **mmcrcluster** or **mmchconfig** commands
 - select tiebreaker disks with **mmchconfig** command



Nodes designated **QN** are quorum nodes
Disks designated **TB** are tiebreaker disks

EXAMPLE: GPFS remains active with the minimum of a single available quorum node and two available tiebreaker disks.



7. Node Failure Recovery

Several of the GPFS management functions we have just considered are designed for node failure recovery and maintaining data integrity.

In this section, let's take a closer look at how this all fits together.



Disk Lease



■ Disk Lease (AKA, "heartbeat")

- Disk leasing is the mechanism facilitating failure recovery in GPFS.
 - Its a GPFS-specific fencing mechanism.
- A node can only access the file system if it has a disk lease.
 - If a node fails or can not access the LAN, it can not renew its lease.
 - Recovery begins after a lease expires.
 - It gives time for I/O "in flight" to complete.
 - It guarantees consistent file system metadata logs.
 - It reduces the risk of data corruption during failure recovery processing.
- Failure recovery will have little or no effect on other nodes in the cluster.

■ Heartbeat Related Tuning Parameters

- mmchconfig parameters
 - failureDetectionTime (default = 35 sec)
 - starting with GPFS v3.2, leaseDuration is derived from other parameters and should be left at its default value
 - leaseRecoveryWait (default = 35 sec)
 - minMissedPingTimeout (default = 3 sec)
 - maxMissedPingTimeout (default = 60 sec)
 - Best practice: do not alter these defaults without guidance from IBM
 - There's a reason they are not documented!



Node Failure Recovery

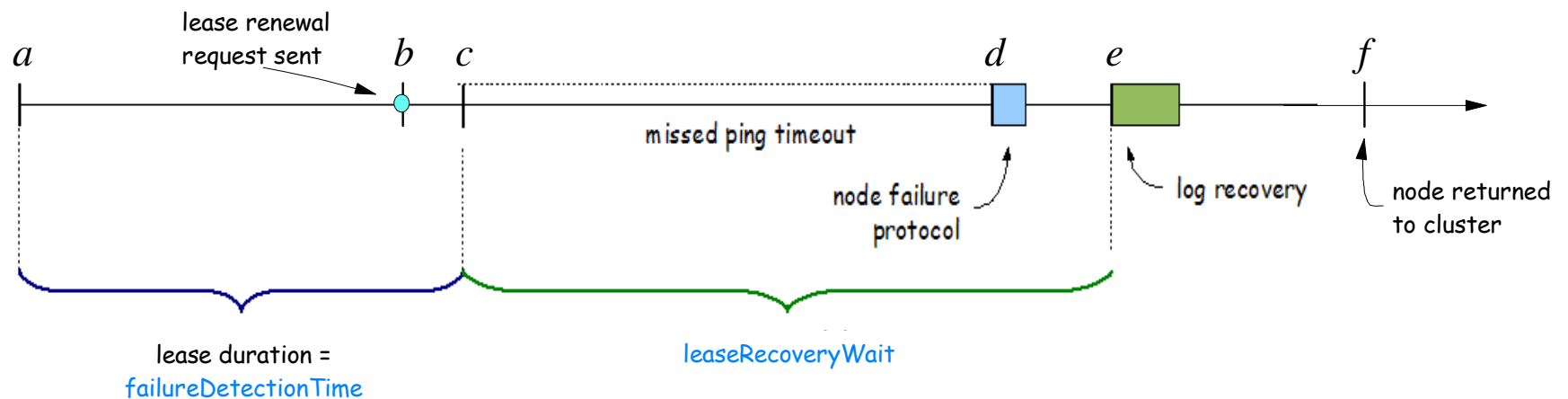
The next few slides take a careful look at node failure recovery under several scenarios. There are more cases than this, but the illustrated scenarios cover the basic concepts.



Node Failure Recovery

Non-Cluster Manager*

★Any node other than the cluster manager.



Suppose this node has just failed.

- Last time this node renewed its lease
- This node sends lease renewal request to cluster manager
- The cluster manager detects that the lease has expired, and starts pinging this node
- The cluster manager decides that the node is dead and runs the node failure protocol
- The file system manager starts log recovery
- The revived node returns to the cluster.
 - It must validate it has access to quorum to return to the cluster.

This is the process where a failed node is removed from the cluster.

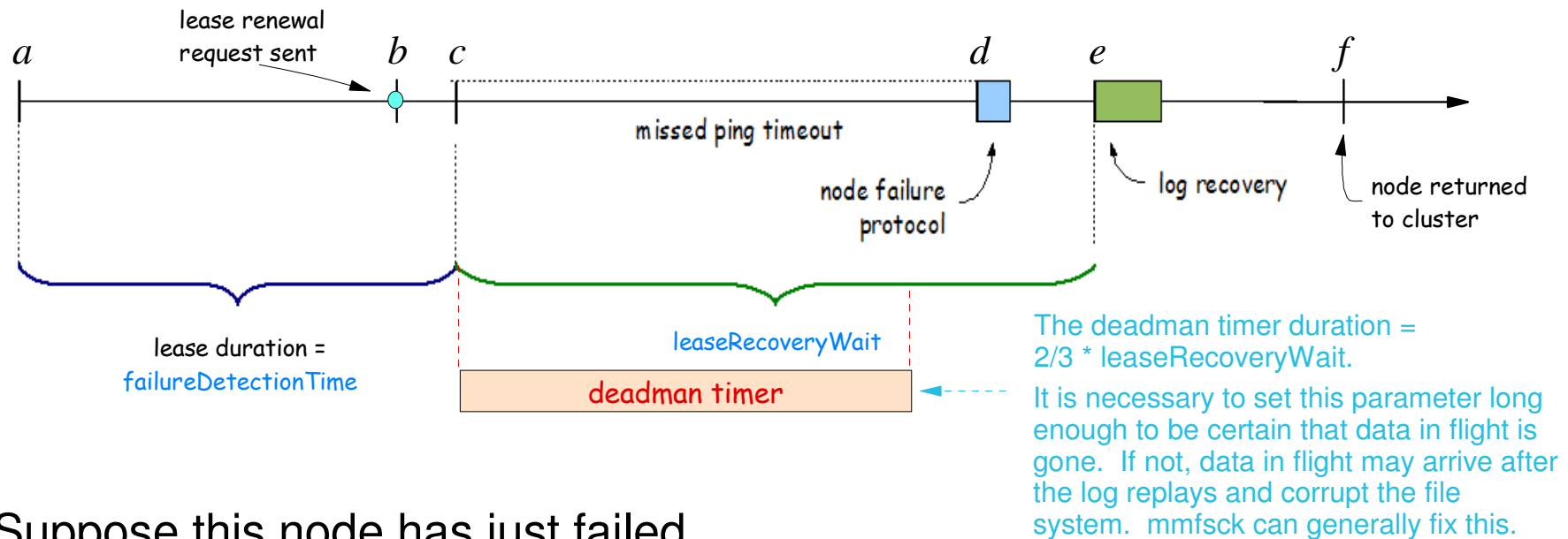
Failure processing is completed **before** recovery processing begins.



Node Failure Recovery: the Deadman Timer

Non-Cluster Manager*

*Any node other than the cluster manager.



Suppose this node has just failed

Problem

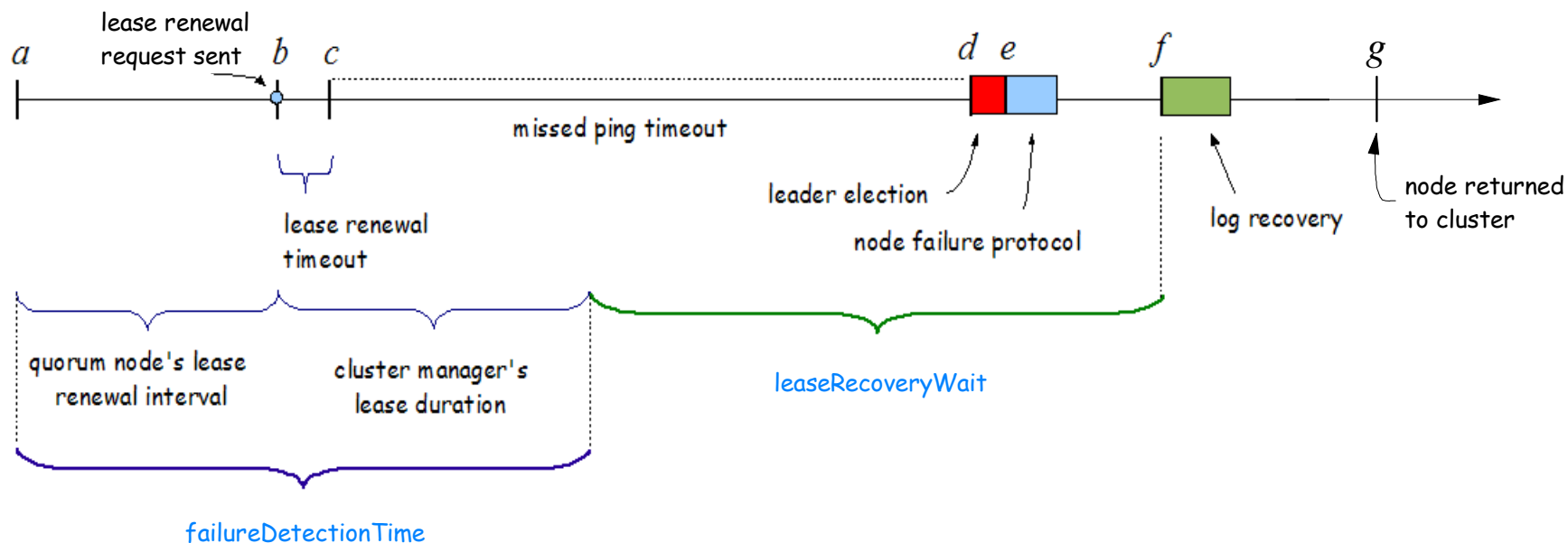
- ▶ If an IOP in flight arrives after the log has been replayed, it would arrive out of order and corrupt the file system.
- ▶ This can only happen on nodes which are writing and have direct access to disk.

Solution

- ▶ If this node is not completely dead, it starts a "deadman timer" thread once its lease duration expires at time *c*.
- ▶ If there is an IOP in flight (e.g., hung IOP) when the deadman timer expires, it will panic the kernel to prevent it from completing.



Node Failure Recovery Cluster Manager



Suppose the cluster manager has just failed

- Last time that the old cluster manager answered a lease renewal request from another quorum node.
- Last time a quorum node sent a new lease request to the old cluster manager. This is also the last time that the old cluster manager could have renewed its own lease.
- A quorum node detects that it is unable to renew its lease and starts pinging the old cluster manager.
- The quorum node decides that the old cluster manager is dead and runs an election to take over as new cluster manager (assuming quorum can still be maintained).
- The election completes and the new cluster manager runs the node failure protocol.
- The file system manager starts log recovery.
- The revived node returns to the cluster.
 - Without manual intervention, it will come back as a quorum node.



Notes on the Recovery Process

■ Recovering from a node failure (includes daemon failures)

- cluster manager broadcasts failed node status to cluster
 - uses heartbeat to detect node failures
 - uses join/leave messages to manage a node failure
- initiate recovery process for each file system mounted on a failed node
 - ensure that failed node no longer has access to FS disks
 - use logs to rebuild metadata that was being modified at the time of failure to a consistent state
 - release locks held by failed node are released
 - mmfsck recovers blocks that have been allocated but assigned to a file during recovery
- recover the failed node and add it back to the cluster

■ Logs (*i.e.*, GPFS Recovery Logs)

- created at file system creation time, additional logs created as needed
- file system manager assigns a log to each node accessing the file system
 - logs are replicated
- logging (and rigid sequencing of operations) preserve atomicity of on-disk structures
- data blocks are written to disk before control structures referencing them
 - prevents contents of previous data block being accessed in new file
- metadata blocks are written/logged so that there will never be a pointer to a block marked unallocated that is not recoverable from the log
- log recovery is run as part of recovery to node failure affecting locked objects



8. Miscellaneous



GPFS Memory Usage and Accounting

■ Heap

- Moderate amount (e.g., 40 MB) of general purpose memory (e.g., thread stacks) which **is** accounted as belonging to GPFS (i.e., mmfsd)
- Similar for both AIX and Linux

■ Token Heap

- Memory used for processing tokens which **is** accounted as belonging to GPFS (i.e., mmfsd)
- Size is negligible except for token manager nodes
- Similar for both AIX and Linux

■ Shared Segment

- Chunk of common memory available to all tasks; used by GPFS for the inode/stat caches
- Since it is available to all tasks, the portion used by GPFS is **not** accounted as belonging to GPFS
- AIX: unpinned, allocated via shmat (32 bit) or kernel call (64 bit)
- Linux: pinned, allocated via mmap

■ Pagepool

- AIX: pinned, allocated via shmat (32 bit) or kernel call (64 bit), but **not** accounted as belonging to GPFS
- Linux (32/64 bit): pinned, allocated via mmap and **is** accounted as belonging to GPFS

COMMENT: When using general tools to measure memory usage (e.g., top), the difference in memory allocation mechanisms for a particular GPFS/OS combination leads to different memory accounting. In particular, GPFS memory usage appears larger under Linux than AIX since the pagepool is attributed to GPFS under Linux, but not under AIX.



GPFS Code Structure

■ All nodes equal in principle (same code installed on all nodes)

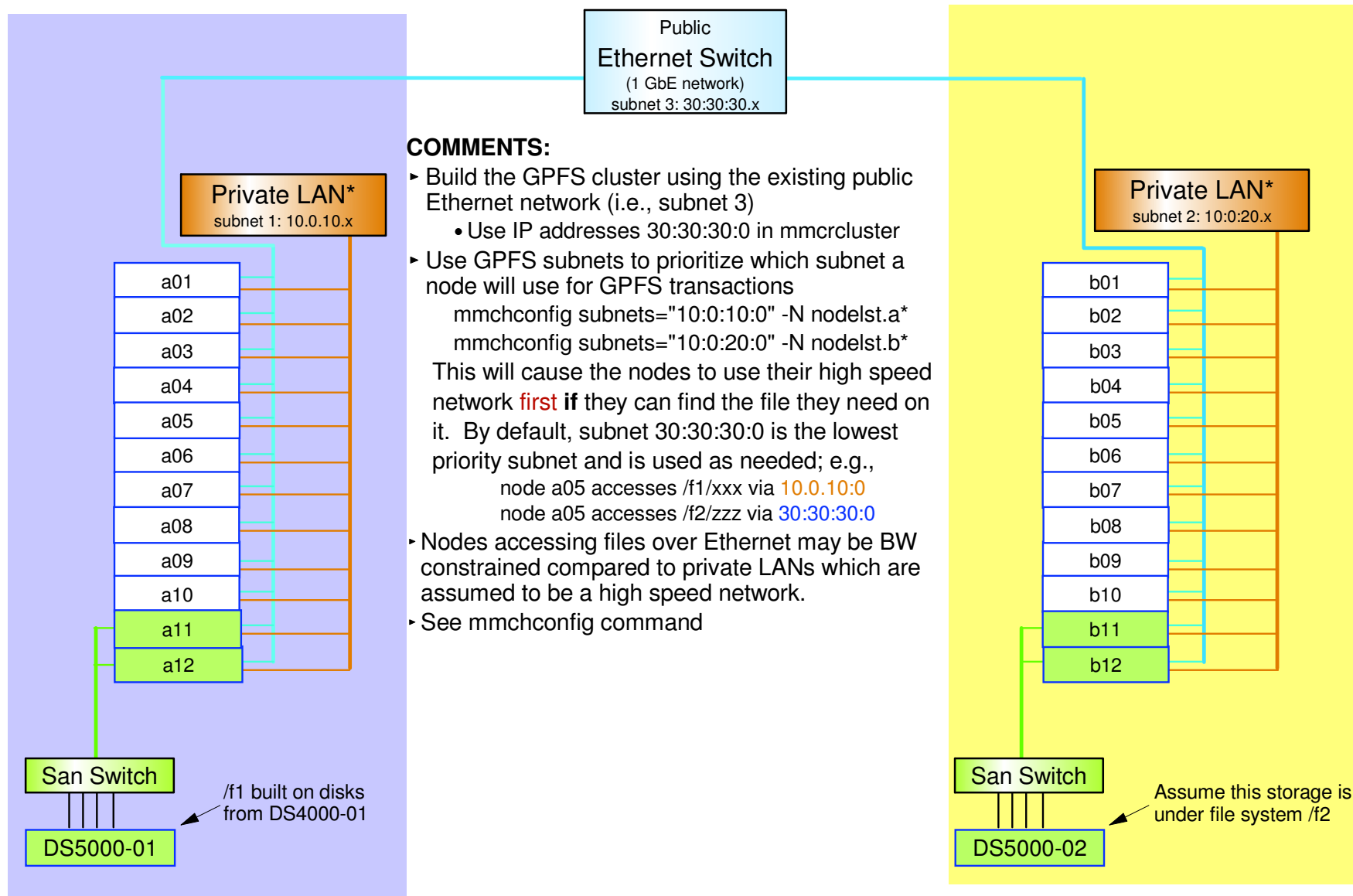
- Some nodes perform special roles (dynamically elected)
- Controllable via config options

■ Components:

- Kernel extension / kernel modules
 - AIX: single kernel extension
 - Linux: three kernel modules
 - tracing, portability/GPL layer, I/O
- Daemon (*i.e.*, mmfsd)
- Commands and scripts
 - “ts-command”: just a stub that sends params to daemon
 - “mm-scripts”:
 - Call ts-commands
 - Some are just wrappers around ts-commands with additional error checking
 - Some do more: manage cluster configuration (update mmsdrfs), gpfs startup/shutdown, etc.



GPFS Subnets



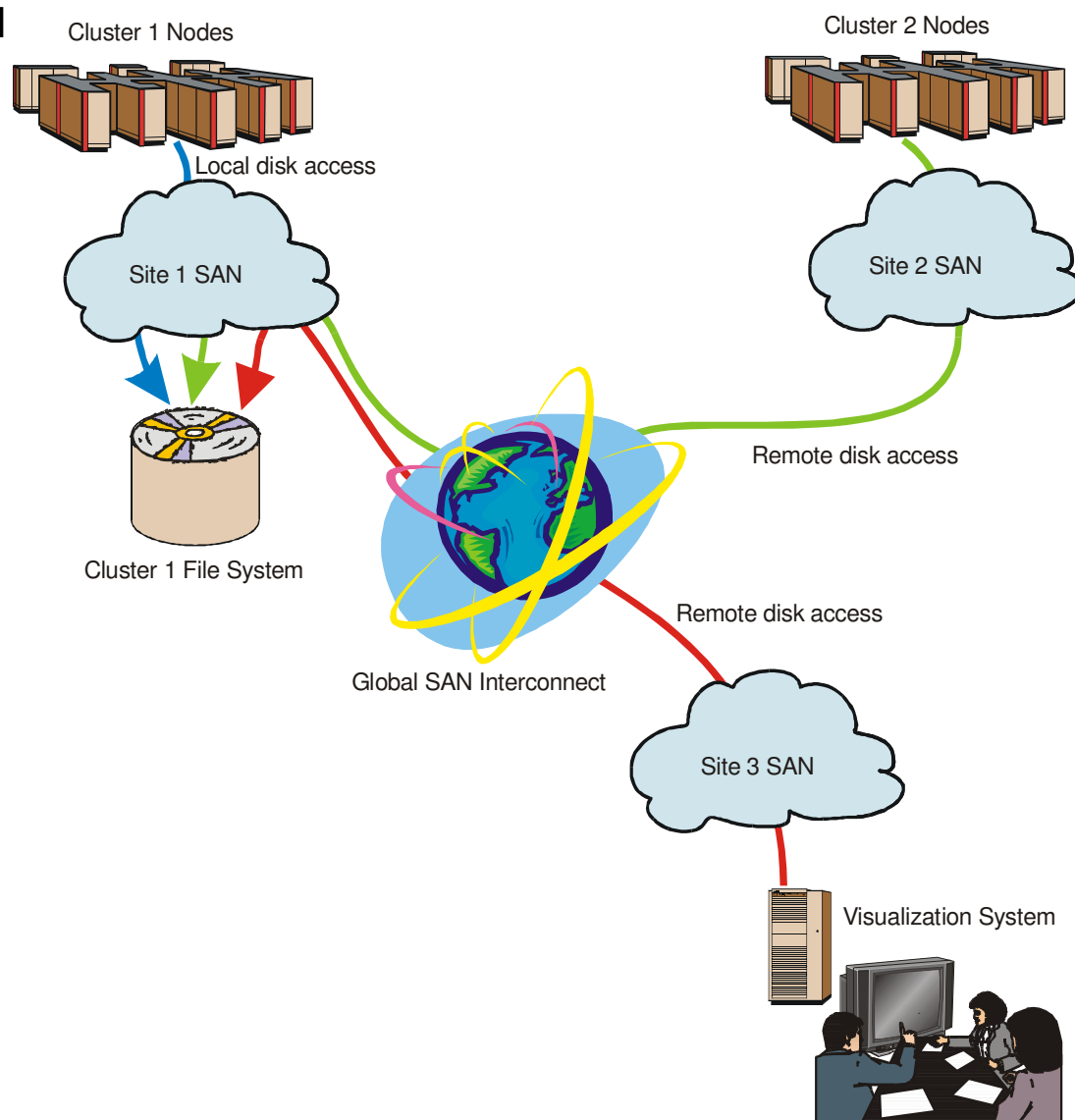
* The private LAN is generally a high speed switch; e.g., IB, Myrinet, Federation
Assume nodelst.a contains the nodes a01-a12 and nodelst.b contains nodes b01-b12.



GPFS Multi-Cluster Feature

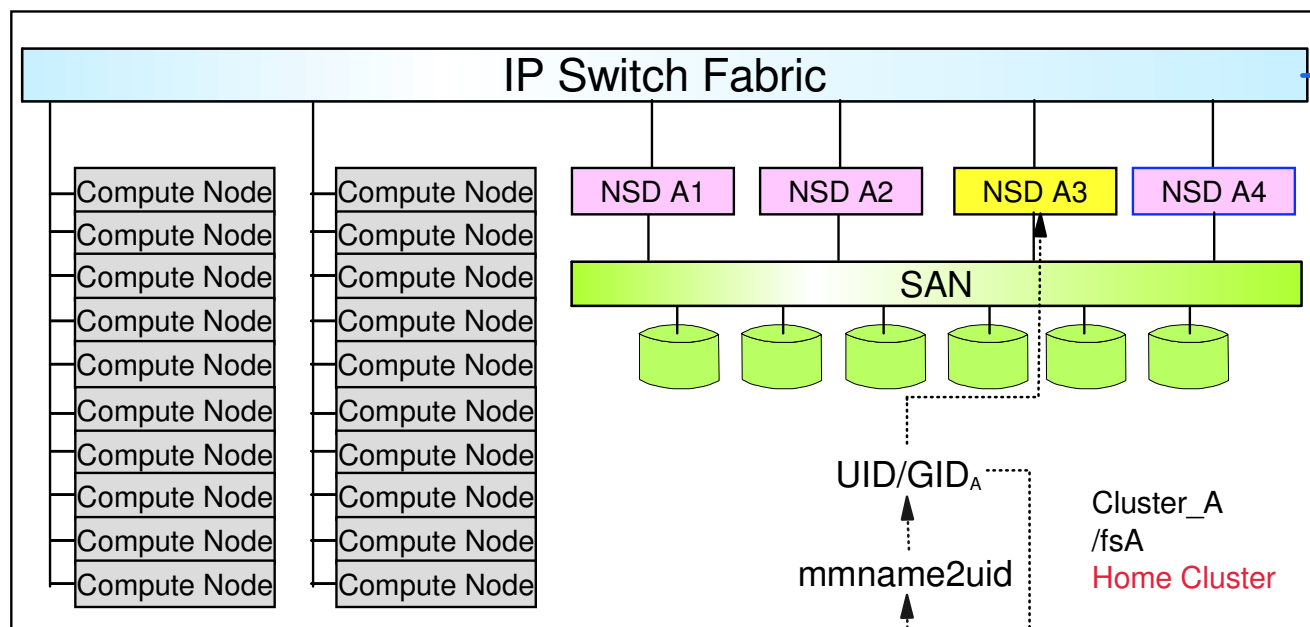
The Big Picture

- Problem: nodes outside the cluster need access to GPFS files
- Solution: allow nodes outside the cluster to natively (i.e., no NFS) mount the file system
 - "Home" cluster responsible for admin, managing locking, recovery, etc.
 - Separately administered remote nodes have limited status
 - Can request locks and other metadata operations
 - Can do I/O to file system disks over global SAN
 - Are trusted to enforce access control, map user Ids, ...
- Uses:
 - High-speed data ingestion, postprocessing (e.g. visualization)
 - Sharing data among clusters
 - Separate data and compute sites (Grid)
 - Forming multiple clusters into a "supercluster" for grand challenge problems
- Scaling: max supported GPFS cluster size





GPFS Multi-Cluster Example



Inter-Switch Link (at least GbE speed!)

COMMENTS:

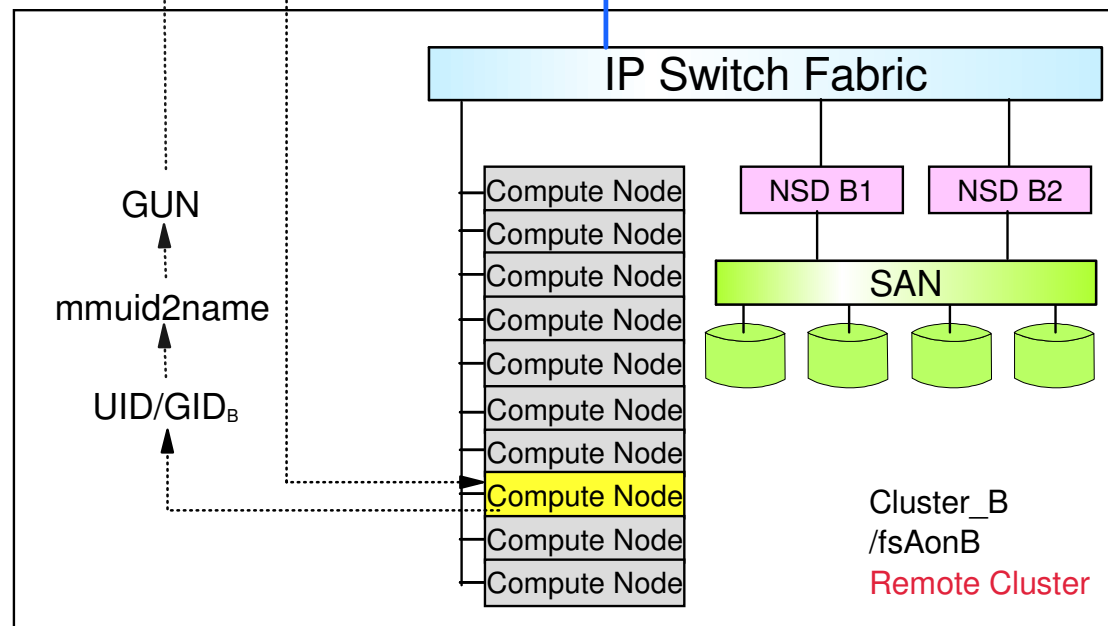
- Cluster_B accesses /fsA from Cluster_A via the NSD nodes
 - see example on next page
- Cluster_B mounts /fsA locally as /fsAonB
- OpenSSL (secure socket layer) provides secure access between clusters for "daemon to daemon" communication using the TCP/IP based GPFS protocol. However, nodes in the remote cluster do not require ssh contact to nodes in the home cluster or vice-versa.

UID MAPPING EXAMPLE (i.e., Credential Mapping)

- pass Cluster_B UID/GID(s) from I/O thread node to mmuid2name
- map UID to GUN(s) (Globally Unique Name)
- send GUN(s) to mmname2uid on node in Cluster_A
- generate corresponding CLUSTER_A UID/GID(s)
- send Cluster_A UID/GIDs back to Cluster_B node running I/O thread (for duration of I/O request)

COMMENTS:

- mmuid2name and mmname2uid are user written scripts made available to *all* users in /var/mmfs/etc; these scripts are called *ID remapping helper functions* (IRHF) and implement access policies
- simple strategies (e.g. text based file with UID <-> GUN mappings) or 3rd party packages (e.g., Globus Security Infrastructure from Teragrid) can be used to implement the remapping procedures





GPFS Multi-Cluster Example

Mount a GPFS file system from Cluster_A onto Cluster_B

On Cluster_A

1. Generate public/private key pair

```
mmauth genkey new
```

COMMENTS

- key pair is placed in /var/mmfs/ssl
- public key default file name id_rsa.pub

2. Enable authorization

```
mmauth update . -l AUTHONLY
```

3. Sysadm gives following file to Cluster_B

```
/var/mmfs/ssl/id_rsa.pub
```

COMMET: rename as cluster_A.pub

7. Authorize Cluster_B to mount file systems owned by Cluster_A

```
mmauth add cluster_B -k cluster_B.pub
```

8. Authorize Cluster_B to mount a particular FS owned by Cluster_A

```
mmauth grant cluster_B -f /dev/fsA
```

On Cluster_B

4. Generate public/private key pair

```
mmauth genkey
```

COMMENTS

- key pair is placed in /var/mmfs/ssl
- public key default file name id_rsa.pub

5. Enable authorization

```
mmauth update . -l AUTHONLY
```

6. Sysadm gives following file to Cluster_A

```
/var/mmfs/ssl/id_rsa.pub
```

COMMENT: rename as cluster_B.pub

9. Define cluster name, contact nodes and public key for cluster_A

```
mmremotecluster add cluster_A -n
```

```
nsd_A1,nsd_A2 -k Cluster_A.pub
```

10. Identify the FS to be accessed on cluster_A

```
mmremotefs add /dev/fsAonB -f /dev/fsA -C  
Cluster_A -T /fsAonB
```

11. mount FS locally

```
mmmount /dev/fsAonB
```

Communication Between Clusters

All nodes in both clusters must have TCP/IP connectivity between each other; this is used only for "daemon to daemon" (*n.b.*, mmfsd) communication via the GPFS protocol. This does not allow remote shell (e.g., ssh or rsh) access between nodes; remote users can **not** access the home cluster by this mechanism. OpenSSL guarantees secure communications.

Contact Nodes

The contact nodes are used only when a remote cluster first tries to access the home cluster; one of them sends configuration information to the remote cluster after which there is no further communication. It is recommended that the primary and backup cluster manager be used as the contact nodes.



GPFS is Available for AIX, Linux and Windows

- GPFS has been designed so that its architecture, commands, programming APIs and other GPFS specific entities are nearly identical for AIX, Linux and Windows.
- Same source code, but...
 - Linux requires building something called the "portability layer"; it does **not** change the kernel.
 - Mapping Unix permissions to Windows
 - The core of GPFS continues to operate on Unix UID/GID values. Windows GPFS nodes perform the task of mapping to Windows SIDs: explicit Unix-Windows ID maps are defined in Active Directory; implicit (default) maps for Windows SIDs are created from a reserved range of UID/GID values; and unmapped Unix IDs are cast into a foreign domain for Windows. Explicit maps persist only in the Active Directory. Implicit maps persist in the file system.
- GPFS under Windows can be used only as a client.
 - GPFS 3.2 supports Windows Server 2003 R2 SP2 x64
 - GPFS 3.3 supports Windows Server 2008 SP2 x64

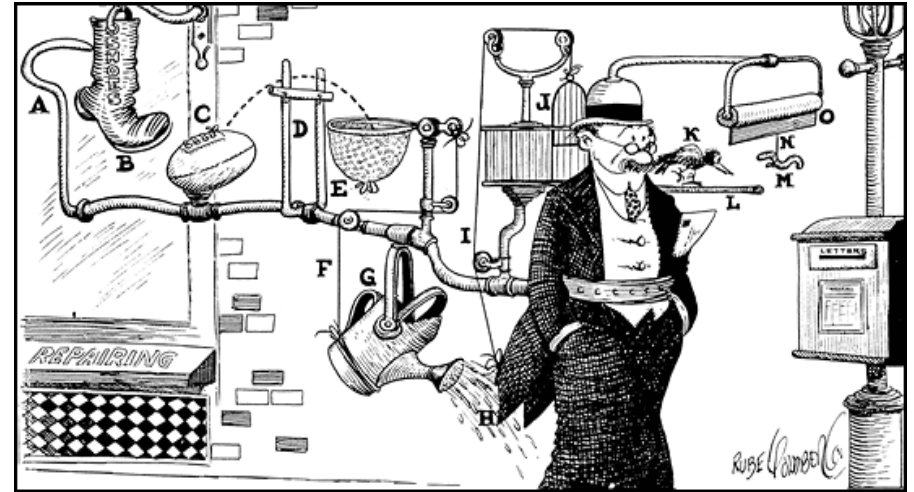


Mixed OS Clusters

- **A single GPFS cluster or GPFS multi-cluster can have nodes running under AIX, Linux and/or Windows at the same time!**
- **Restriction**
 - All LUNs for a particular file system must run under the same OS
 - Corollary: the NSD nodes for *a given file system* must run under same OS
 - Special request (i.e., RPQ) is required to use Windows for an NSD server

9. GPFS Environment

GPFS does not exist in isolation; it must be integrated with other components when designing an overall solution.



The following pages look at selected hardware and software components (disk controllers, disks and storage servers) commonly used with GPFS file systems.



Tested Scaling Limits

■ Largest File System

- tested - 4 PB
- architectural limit - 2^{99}

■ Largest Tested Cluster

- AIX
 - tested - 1530 nodes
 - over 128 nodes requires review by IBM
- Linux
 - tested
 - 3500+ nodes in a multi-cluster (including one cluster with 2560 nodes)
 - 4000+ nodes in a single cluster
 - over 512 nodes requires review by IBM





Tested Disk Products

GPFS v2.3 - v3.2

■ AIX

- DCS9900, DCS9550
- DS3000, DS4000, DS5000 series systems
- ESS and DS8000 series (i.e., shark)
- SAN Volume Controller (V1.1 and V1.2, V2.1)
- 7133 Serial Disk System (i.e., SSA)
- EMC Symmetrix DMX (FC attach only)
- Hitachi Lightning 9900 (HDLM required)

-9910, 9960, 9970V, 9980V

■ Linux

- DCS9900, DCS9550
- DS3000, DS4000, DS5000 series systems
- EMC Symmetrix DMX 1000 with PowerPath v3.06 or v3.07

COMMENT:

- ▶ GPFS does not rely on the SCSI persistent reserve for failover. This reduces the risk associated with using non-tested storage controllers with GPFS.
- ▶ However, starting with GPFS 3.2, SCSI persistent reserve is available as an option on NSD servers under AIX (see mmchconfig).

See <http://publib.boulder.ibm.com/clresctr/library/gpfsclustersfaq.html> for the complete list of tested disk

This is NOT the only disk that will work with GPFS. In general, any reasonable block device will work with GPFS. According to the FAQ page, the "GPFS support team will help customers who are using devices outside of this list of tested devices, to solve problems directly related to GPFS, but not problems deemed to be issues with the underlying device's behavior including any performance issues exhibited on untested hardware." Before adopting such devices for use with GPFS, it is urged that the customer first run proof of concept tests.



Redundant Array of Inexpensive Disk (RAID)

RAID Concept and definitions

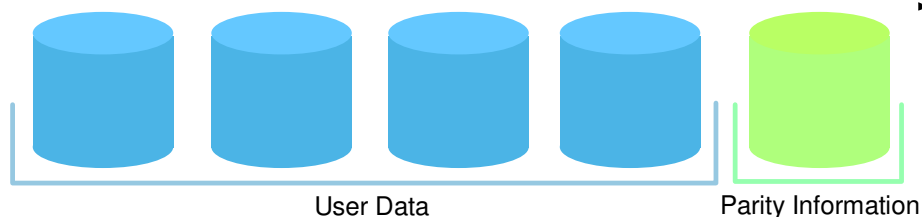
- RAID Set: a set of disks containing user data and parity information
- Parity: information used to reconstruct user data lost when a disk in a RAID group fails

Common RAID Levels:

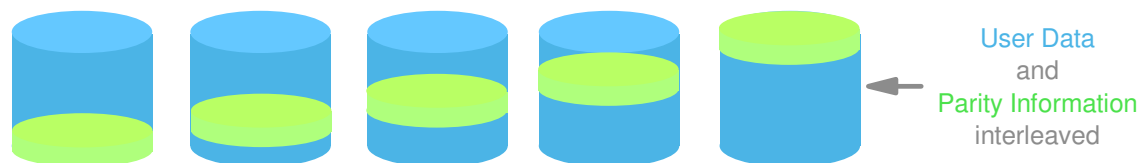
Other RAID Levels

- ▶ JBOD: Just a Bunch Of Disk
- ▶ RAID 0: striping without redundancy
e.g., 4+0P
- ▶ RAID 1: mirroring
- ▶ RAID 10: striping across mirrored groups
m1m1 - m2m2 - m3m3 - m4m4

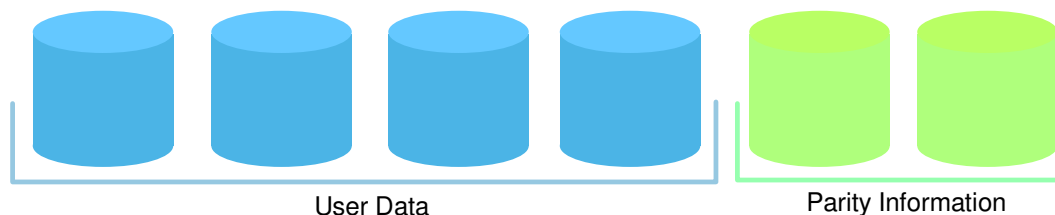
■ RAID 3, 4+P



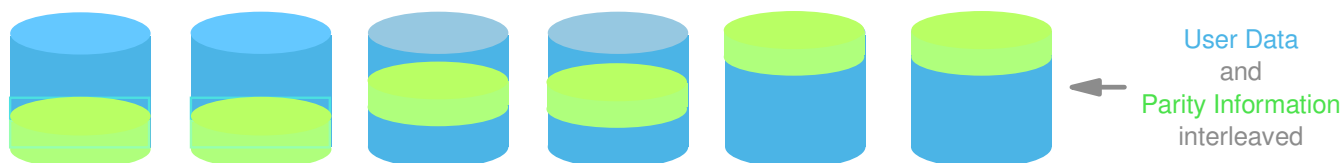
■ RAID 5, 4+P



■ RAID 6*, 4+2P RAID 3 extension



■ RAID 6*, 4+2P RAID 5 extension



* Since RAID 6 group has 2 redundant disks, it is common to make the RAID group larger (e.g., 8+2p)



Redundant Array of Inexpensive Disk (RAID)

Comments on Terminology and Best Practices

■ RAID Sets

- Different OEM vendors use different names for the grouping of disks that I am calling a "RAID Set". I am using this term as a generic alternative.
 - DDN (DCS9000): "Tier"
 - IBM (DS8000): "Array"
 - "array sites" and "ranks" are closely related terms
 - LSI (DS3000, DS4000, DS5000): "Array"



A rose by any other name has just as many thorns. ;->

■ LUNs

- A LUN (logical unit) is an entry in /dev for Unix based OSs
 - examples
 - Linux: /dev/sdb
 - AIX: /dev/hdisk2

■ Best Practice: LUNs vs. RAID sets

- There should be 1 LUN per RAID set.
 - Multiple LUNs per RAID set can lead to "LUN thrashing" where the seek arm "bounces" between LUNs to service uncoordinated requests.
 - Older versions of Linux and storage controller microcode did not support LUN sizes large enough to accommodate today's large disks. This forced users configure multiple LUNs per RAID set.
 - This issue applies to the DS3000, DS4000, DS8000, DCS9000. Storage controllers from some other vendors are designed to create very large RAID sets presented as multiple LUNs to the OS as a best practice.
- All examples in this presentation are configured with 1 LUN per RAID set.



Disk Technology

■ FC, SAS, SCSI

- Enterprise Class
 - different protocols, same mechanical standards
- Rotational speed: 15 Krpm
- Common drive sizes
 - 300 GB, 450 GB, 600 GB
- 90% duty cycle
- MTBF = 2.0 MHour⁴
- Single drive IOP performance, no caching¹
 - 420 IOP/s
- Single drive BW²
 - cache disabled
 - write = 50.8 MB/s
 - read = 95.4 MB/s
 - cache enabled
 - write = 154.6 MB/s
 - read = 123.6 MB/s

■ Some Risks to be Considered

- Probability of "dual disk failures"
 - SATA: TBD
 - enterprise class storage: 1 failure / 73 yrs
 - calculation based on binomial distribution with ~ = 6200 SSA disks
- Sensitivity to "head wobble"
 - compounded by very high density drives

■ SATA/2



Cheap
and
Cheerful

- Cost Optimized
- Rotational speed: 7200 rpm
- Common drive sizes
 - 750 GB, 1 TB, 2 TB
- Duty cycle is generally ignored now.
- MTBF < 1.6 MHour with 50% duty cycle⁴
- MTBF < 0.9 MHour with 90% duty cycle⁴
- Single drive IOP performance, no caching¹
 - with command tag queueing: 120 IOP/s
 - without command tag queueing: 60 to 70 IOP/s
- Single drive BW³
 - cache disabled
 - write = 18.5 MB/s
 - read = 59.2 MB/s
 - cache enabled
 - write = 30.3 MB/s
 - read = 74.9 MB/s

Footnotes:

1. IOP rates assume 4K records
2. DS4800
 - dd buffer size = 1024K
 - cache block size = 16K
 - [segment size = 256K](#)
3. DS4700
 - dd buffer size = 1024K
 - cache block size = 16K
 - [segment size = 64K](#)
4. [MTBF based on Hitachi disk](#)



Comments on the Proper Use of SATA

Market segment:

- ▶ "scalable SATA storage for small to midrange businesses at an *affordable price*"
- ▶ "for data archival, data reference, and near-line storage applications" (e.g., lower tier storage in HSM)

Risk Management Strategies

- ▶ do not use in large, I/O intensive storage systems or as first tier storage
- ▶ partition storage into several file systems or storage pools
 - reduces collateral damage, but compromise peak data rate for an application
- ▶ for GPFS, designate FC disk as metadataOnly and SATA disk as dataOnly
 - reduces risk of metadata loss and increases access rate to metadata
- ▶ adopt RAID6 configuration



Storage Controllers

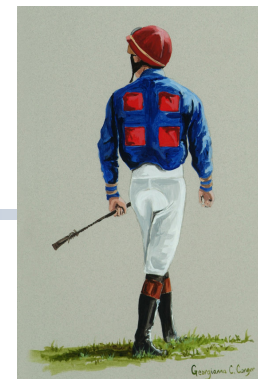
HPC systems requiring high bandwidth and/or large capacities use disk controllers to manage external disk.

Common IBM choices for HPC include

- ▶ DS3000
 - low cost of entry
- ▶ DS4000
 - DS4800 replaced by the DS5300
 - DS4700 is higher end product cf. DS3000, yet with a low cost of entry
- ▶ DS5000
 - DS5300 balanced streaming and IOP performance
 - DS5100 lower performance with a lower cost of entry
- ▶ DS8000
 - DS8300 provides very high reliability with good IOP performance
- ▶ DCS9000
 - designed specifically for HPC optimizing streaming BW and capacity



DS3000 Series



DS3200



- 3-Gbps SAS connect to host
- Direct-attach
- For System x
- 2U, 12 disks
- Dual Power Supplies
- Support for SAS or SATA disks
- Expansion via EXP3000

DS3400



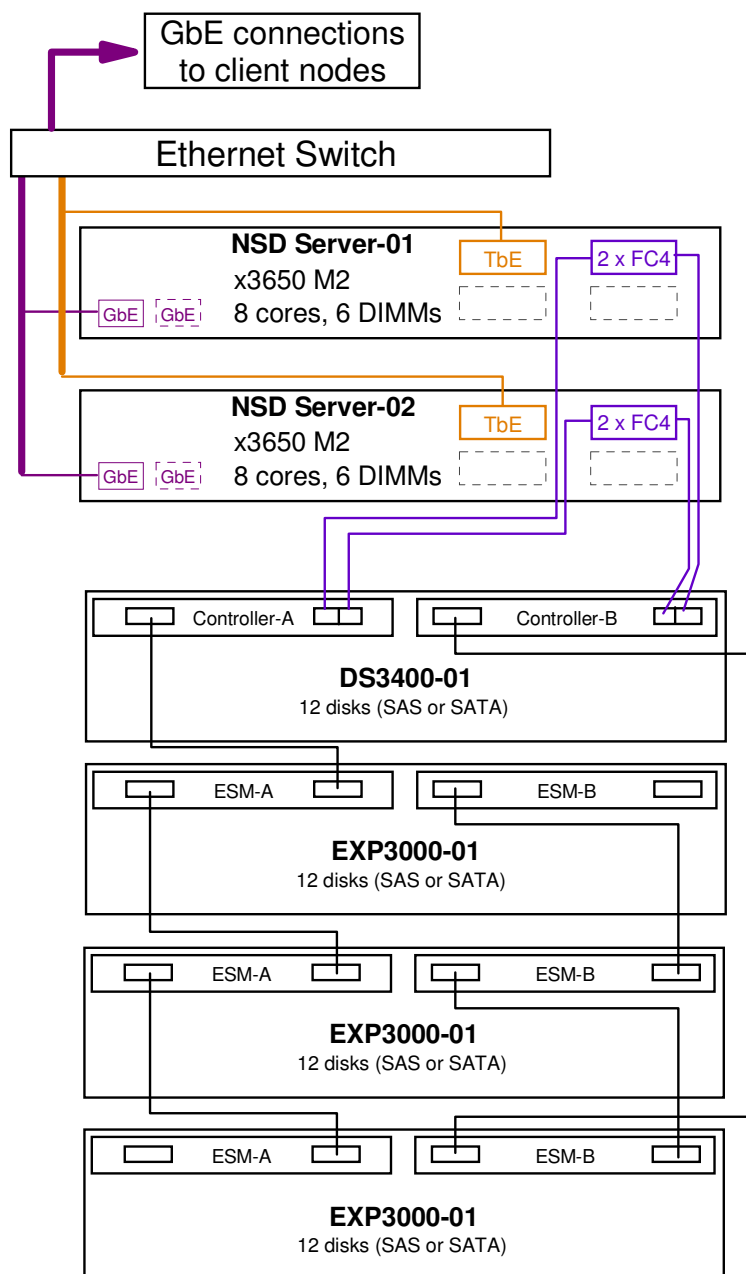
- 4-Gbps Fibre connect to host
- Direct-attach or SAN
- For System x & BladeCenters
- 2U, 12 disks
- Dual Power Supplies
- Support for SAS or SATA disks
- Expansion via EXP3000

COMMENT: DS3300 provides an iSCSI interface for the same basic hardware.
Its not commonly used with GPFS.



DS3400

Example Configuration



- ▶ NSD Server: x3650 M2
 - at least 8 Cores
 - at least 6 GB RAM
 - 2 dual-port 4 Gb/s FC HBAs (2xFC4)
 - at most 760 MB/s per adapter
 - single 10 GbE (TbE) adapter per node
 - at most 725 MB/s per adapter (recommend Myricom TbE)
- ▶ Disk Controller: DS3400 with EXP3000
 - Peak sustained performance (theoretical)
 - streaming: write < 700 MB/s, read < 900 MB/s
 - IOP rate: write < 4500 IOP/s, read < 21,000 IOP/s
 - 12 disks per DS3400 plus 12 disks per EXP3000
 - up to 48 disks
 - Example: 15Krpm SAS disks @ 450 GB/disk
 - 4 x 4+P RAID 5 + 2 hot spares (optimize streaming performance)
 - raw ~ 10 TB, usable ~ 7 TB
 - 9 x 4+P RAID 5 + 3 hot spares (optimize IOP performance)
 - raw ~ 21 TB, usable ~ 16 TB
 - Example: SATA disks @ 1 TB/disk
 - 4 x 8+2P RAID 6 + 2 hot spares (optimize capacity)
 - raw ~ 42 TB, usable ~ 32 TB

WARNING: The DS3400, while relatively fast and inexpensive, is **not** well suited for large configurations, especially when using SATA drives. RAID array rebuilds are common in large configurations (e.g., 10 x DS3400s with 480 drives) especially for SATA. But a DS3400's performance is significantly compromised during a rebuild. Therefore at any given time in file systems aggregated across many DS3400s, a RAID array rebuild will be in progress and the expected value of file system performance will be significantly less than the maximum possible sustained rates.



DS3400

Benchmark Results

GPFS Parameters

- ▶ blocksize = 256K or 1024K
- ▶ pagepool = 1G
- ▶ maxMBpS = 2000

DS3400 Parameters

- ▶ RAID 5 array = 4+P
- ▶ segment size = 64K or 256K
- ▶ cache page size = 16K
- ▶ read ahead = default
- ▶ write cache = enabled
- ▶ write cache mirroring = disabled

Bandwidth Scaling

- ▶ BW per 4+P RAID 5 array using 15 Krpm disks
- ▶ read cache: ON with default read ahead
- ▶ write cache: ON

RAID sets	1	2	4	8
write (MB/s)	239	471	655	647
read (MB/s)	294	615	860	875

↑
The most efficient
streaming occurs
using only 4 arrays.

← This is **not** a best practice

Streaming Job*

- ▶ record size = 1024K
- ▶ file size = 4G
- ▶ number of tasks = 8
- ▶ access pattern = seq

IOP Job*

- ▶ record size = 2K
- ▶ total data accessed = 1G
- ▶ number of tasks = 16
- ▶ access pattern = small file

8 RAID sets	write	read
stream*	650 MB/s	875 MB/s
IOP*	19,100 IOP/s	23,000 IOP/s

* Configuration was optimized differently for each test.

Benchmark tool: ibm.v4a

Theoretical max IOP rates for the DS3400

- ▶ cached < 96,000 IOP/s (512 B transactions)
- ▶ uncached: write < 4,200 IOP/s, read < 19,000 IOP/s (4 KB transactions)

Benchmark System:

These results were produced using the configuration from the previous page with 16 client nodes (x3550) connected to a LAN via GbE.

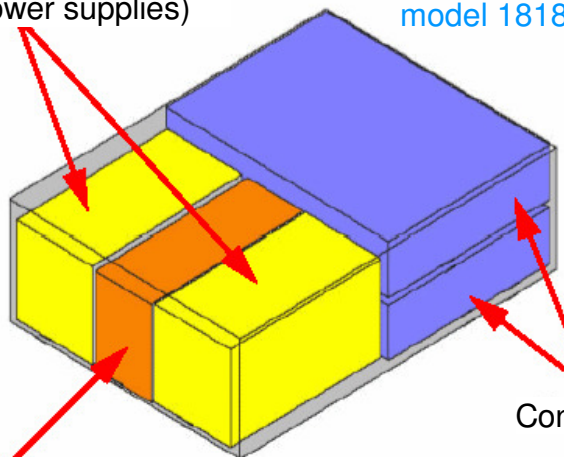


DS5000 Series



Controller Support Modules
(Fans, power supplies)

DS5300
model 1818-53A



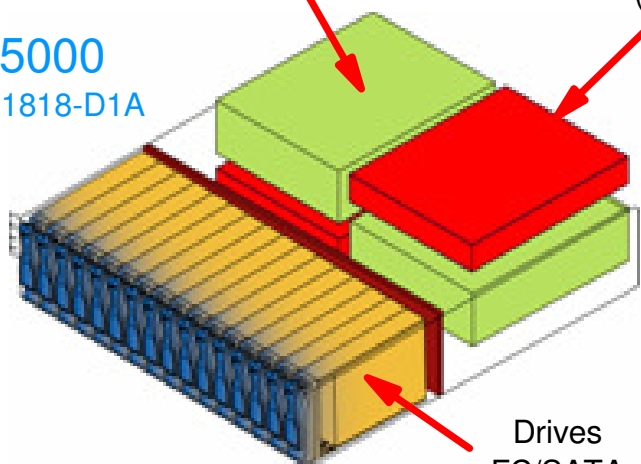
Controllers

Interconnect Module
(batteries, midplane)

Power/cooling

Controllers
(ESMs)

EXP5000
model 1818-D1A



Drives
FC/SATA

4u

3u

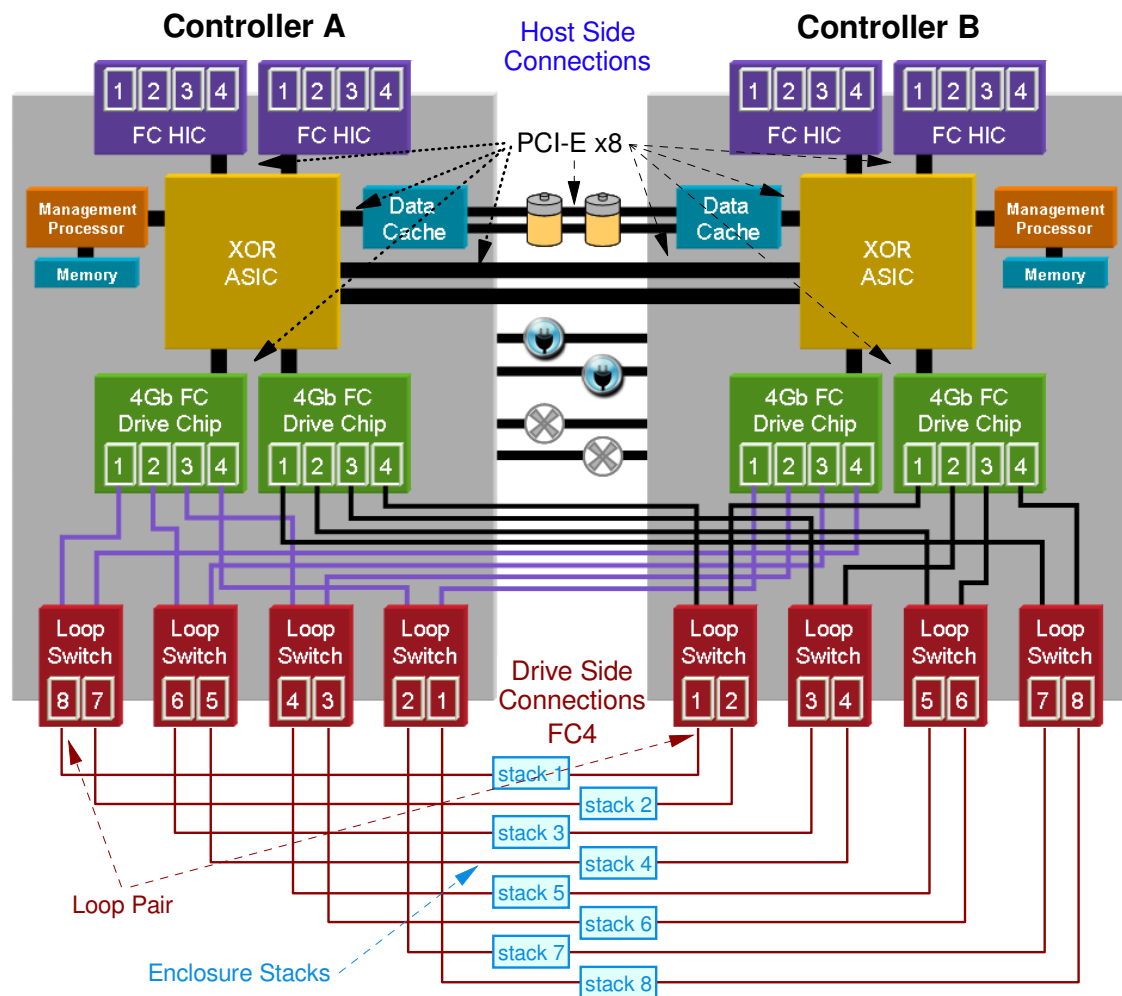


16 Disks per Disk Enclosure



DS5300

Controller/Enclosure Overview



A *loop pair* is a set of redundant drive side cables as shown in this diagram. A *stack* is a set of enclosures along a loop pair. A DS5300 supports *at most* 4 x EXP5000 enclosures per stack, but never more than 28 x EXP5000 total. Be sure to balance the number of enclosures across the stacks.

- Dual, redundant RAID controllers
- Dual, redundant power, battery backup and fans
- Internal busses (theoretical)
 - PCI-E x8 simplex rate = 2 GB/s
- Host side connections (measured)
 - 16 host-side connections
 - FC4 < 380 MB/s
 - FC8 < 760 MB/s
 - Active/passive architecture
- Drive side connections (measured)
 - 16 drive-side connections
 - FC4 < 380 MB/s
- Supported enclosures
 - EXP5000 (FC switched, 4 Gb/s)
 - EXP810 (FC switched, 2 Gb/s)
 - Maximums
 - 28 enclosures, 16 disks per enclosure
 - 448 disks
- Disk Technology
 - 15 Krpm FC disk (300, 450 GB)
 - SATA (750, 1000 GB)
- Peak sustained rates (theoretical)
 - Streaming (to media)
 - requires 192 x 15Krpm drives
 - write < 6 GB/s
 - read < 6 GB/s
 - IOP Rate (to media)
 - requires 448 x 15Krpm drives
 - write < 45,000 IOP/s
 - read < 172,000 IOP/s

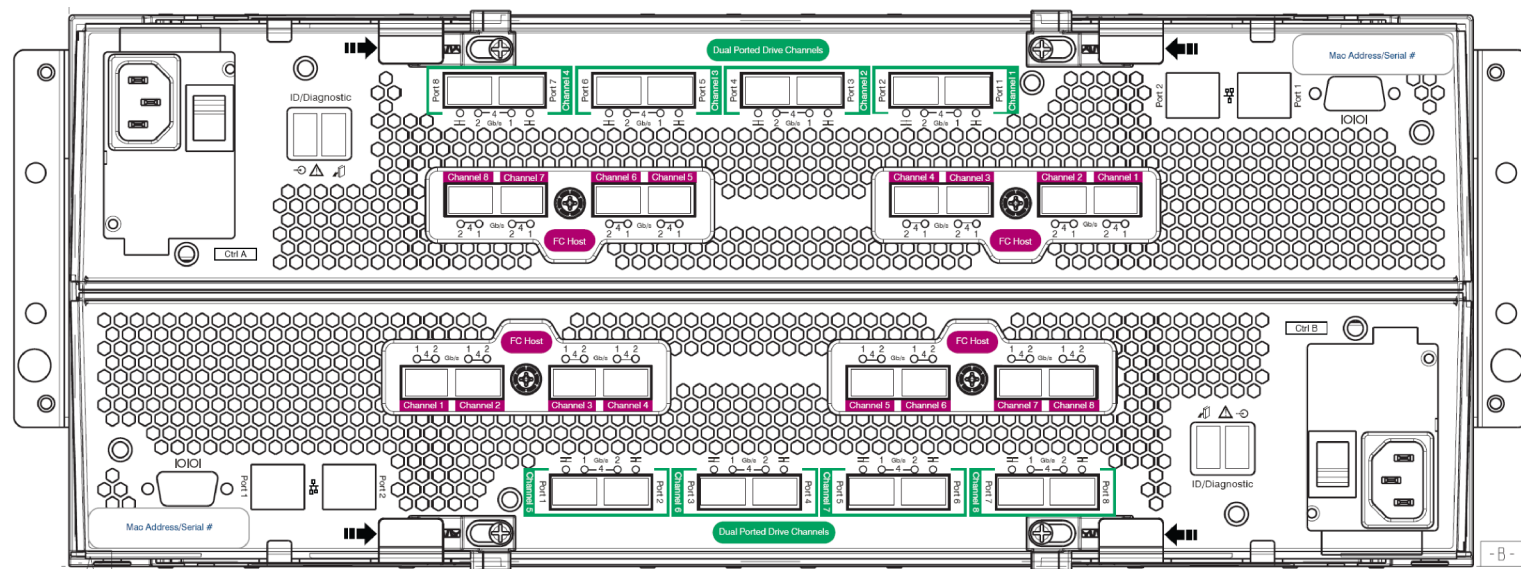


DS5300

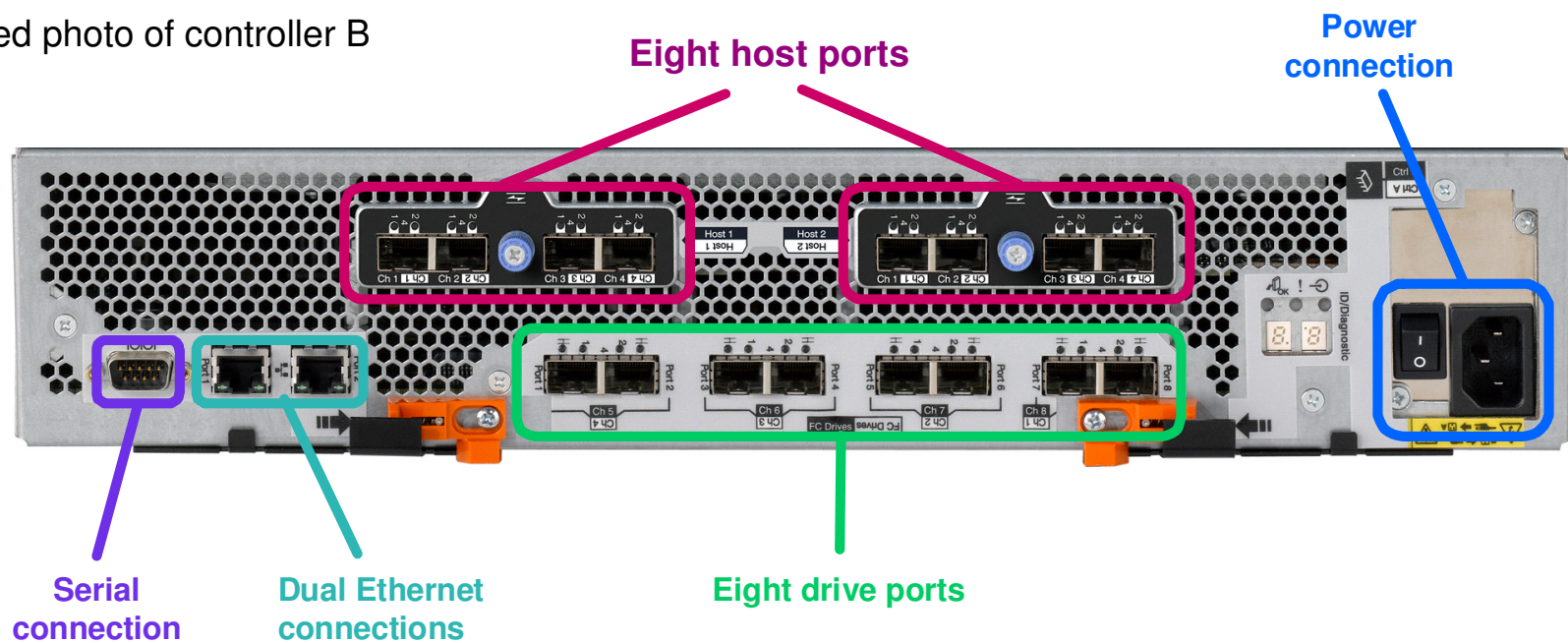
Rearview

Controller A

Controller B

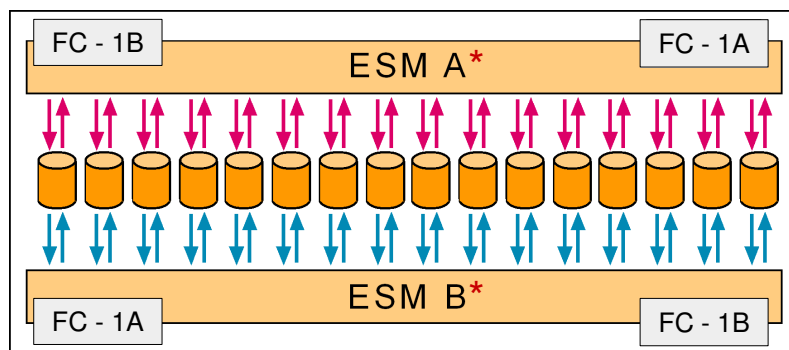


Annotated photo of controller B





EXP5000



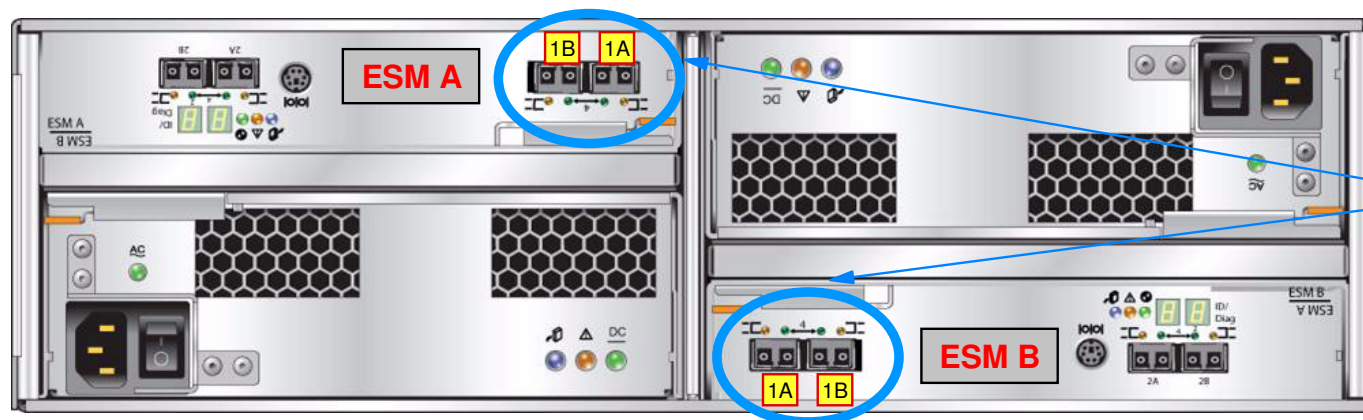
logical layout

- ▶ 16 drives in 3U enclosure
- ▶ 4 Gbps FC interfaces / ESMs
 - High-speed, low-latency interconnect from controllers to drives
- ▶ Supports intermixing FC and SATA drives
- ▶ Unique speed-matching technology
 - 3 Gbps SATA II drives effectively run at 4 Gbps speeds
- ▶ Switched architecture
 - Higher performance, lower latency
 - Drive isolation, better diagnostics
- ▶ RoHS compliant
- ▶ NEBS level 3 certified

FOOTNOTES

- ★ ESM A is the primary path for the odd drives
- ESM B is the primary path for the even drives

If an ESM fails, the other ESM can access all of the drives.



Only use highlighted ports
(EXP5000 does not support "trunking")



DS5300

Cabling and Disk to Array Mapping

Careful attention must be given to cabling and disk to array mapping on the DS5300 in order guarantee optimum streaming performance. This issue is less significant for IOP performance.

WARNINGS:

- ▶ Default array mappings (*e.g.*, created by SMclient) are *not* guaranteed to be optimum!
- ▶ Rules and best practices for the DS4800 do *not always* apply to the DS5300.

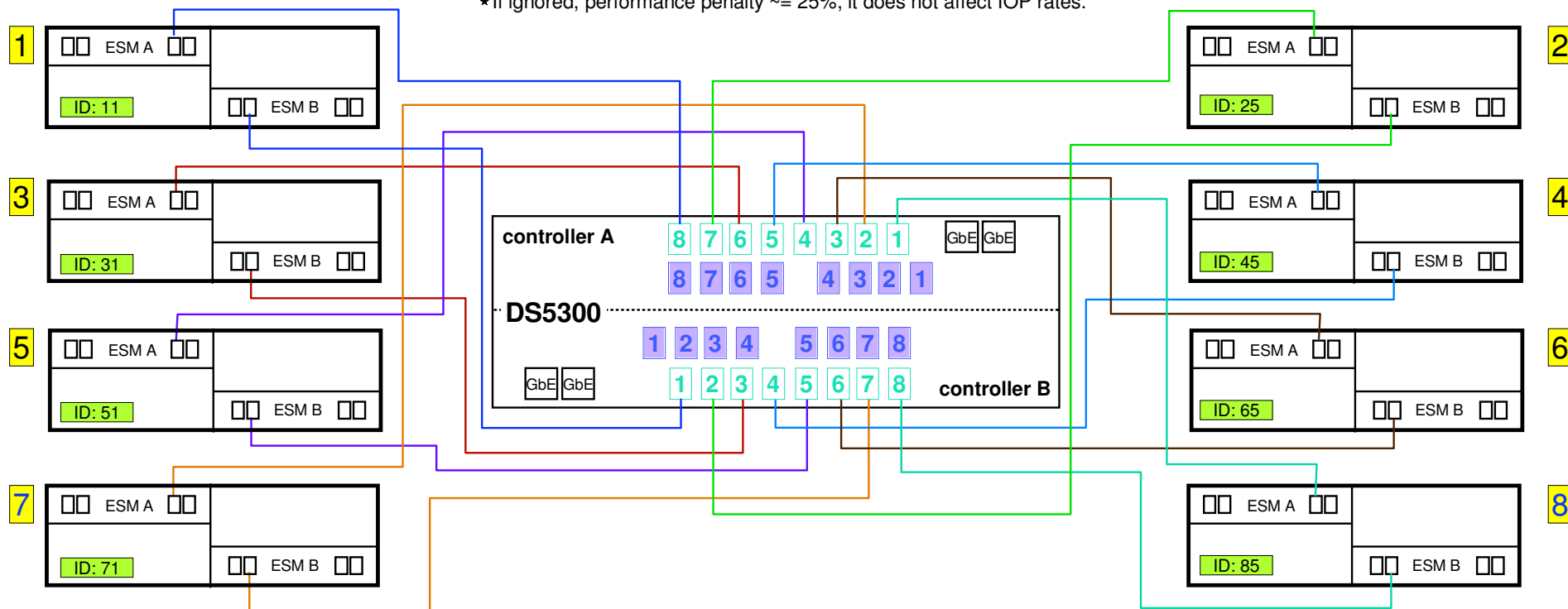


DS5300

Drive Side Cabling - 8 Enclosures

Balance*: Best streaming performance is achieved using a multiple of 8 x EXP5000 drawers with the same number of drawers per stack. Optimum performance is achieved using 8, 16 or 24 stacks.

* If ignored, performance penalty $\sim 25\%$; it does not affect IOP rates.



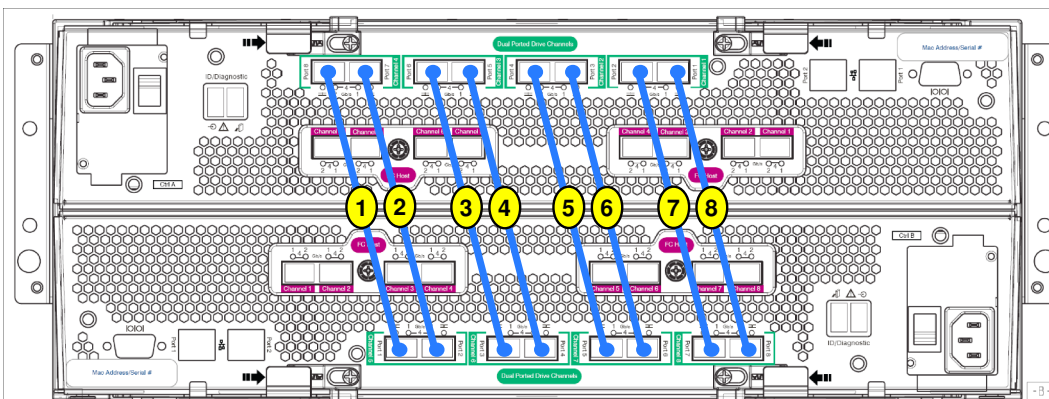
Stacks

When attaching enclosures, drive loops are configured as redundant pairs (*i.e.*, loop pairs) utilizing one port from each controller; the enclosures along a loop pair are called a stack.

Tray ID

Tray ID is assigned during system configuration. The values are not arbitrary. **Best practice:**

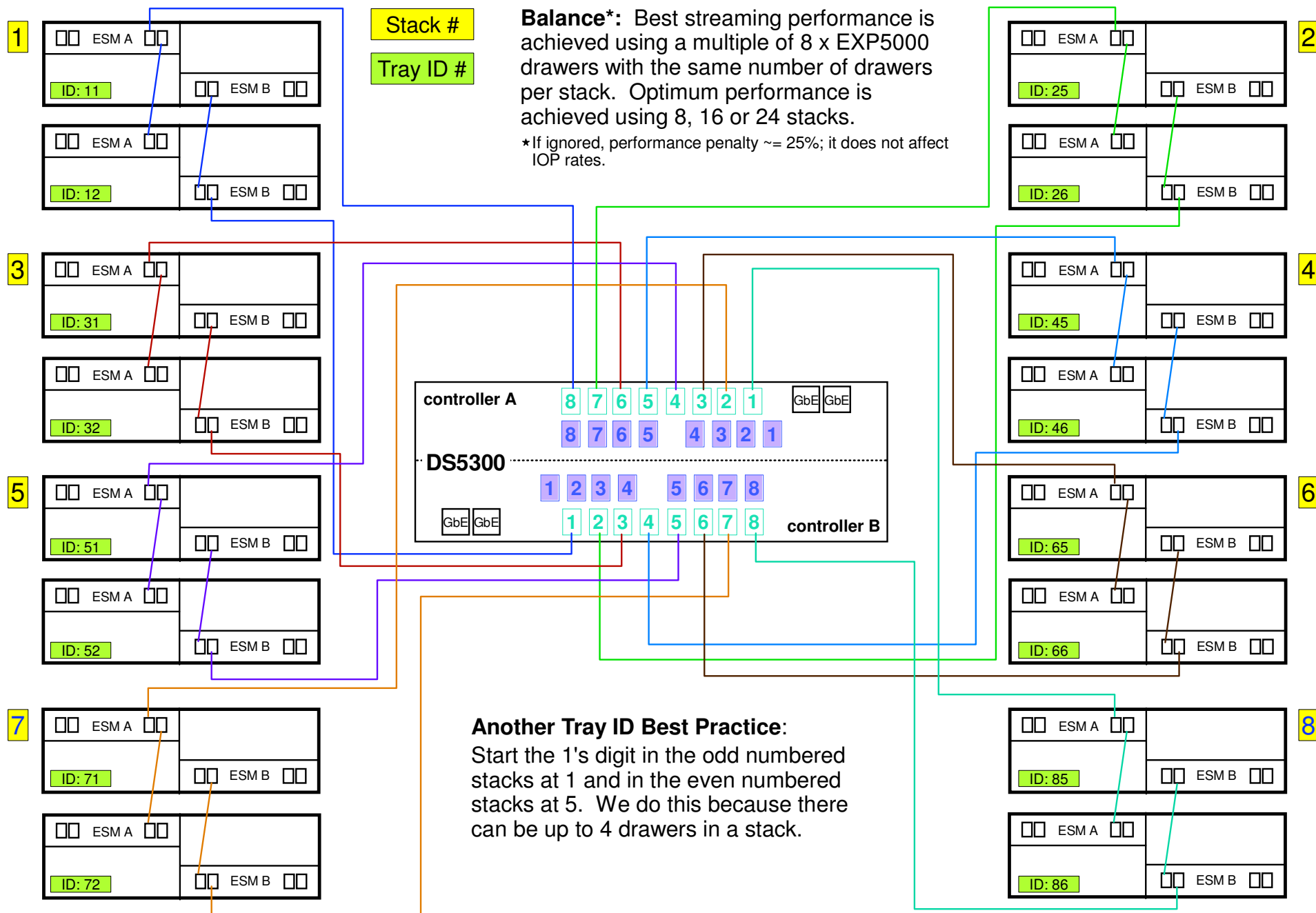
- 10's digit: stack number
- 1's digit: ordinal number within a stack





DS5300

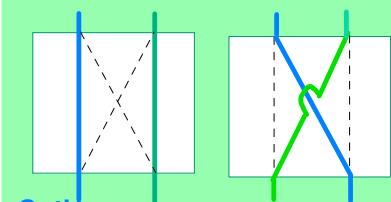
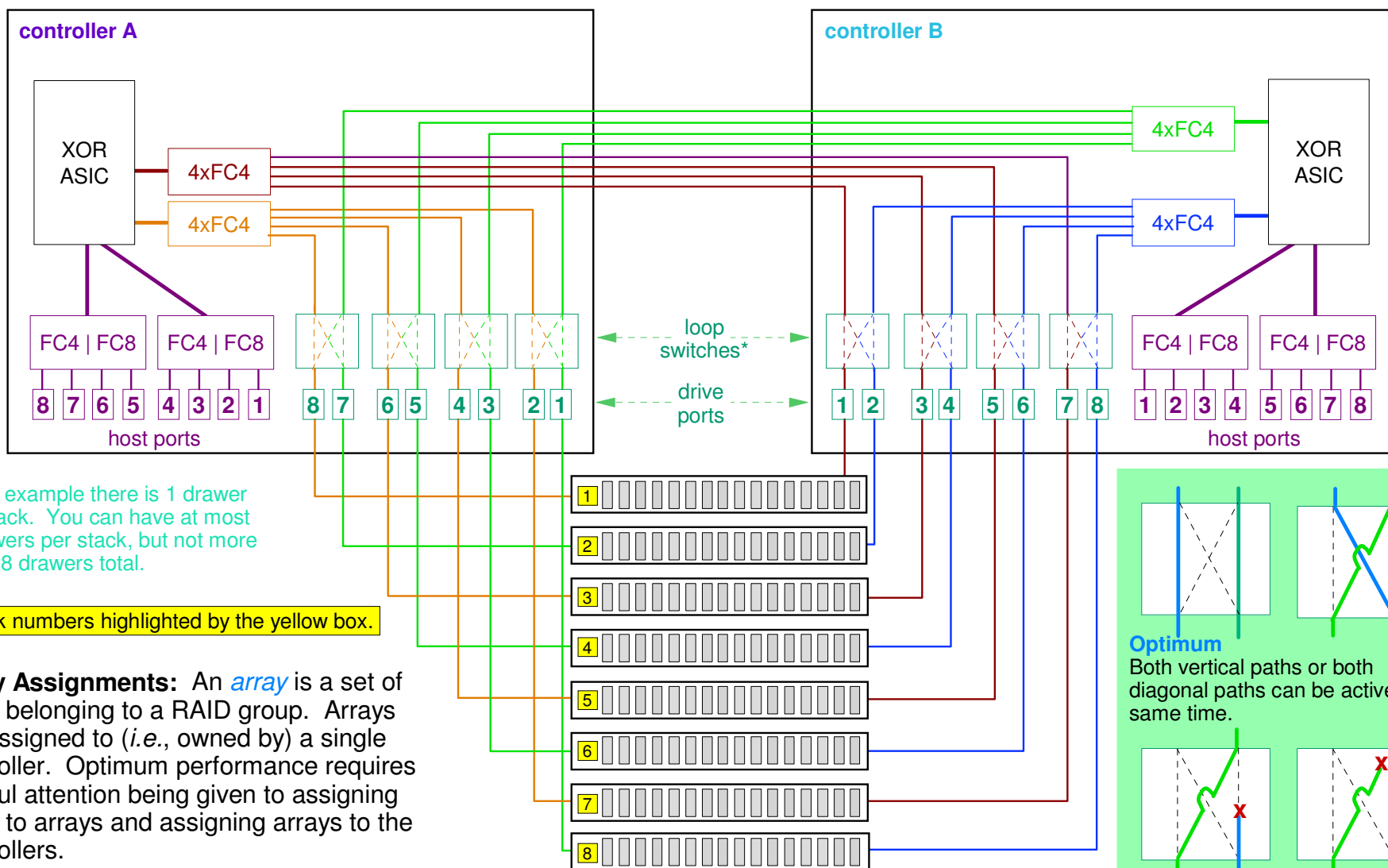
Drive Side Cabling - 16 Enclosures





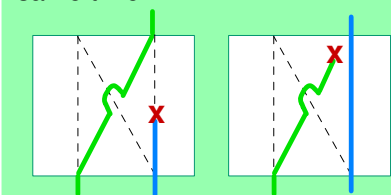
DS5300

Drive Side Cabling and Disk to Array Mapping



Optimum

Both vertical paths or both diagonal paths can be active at same time.



Sub-Optimum

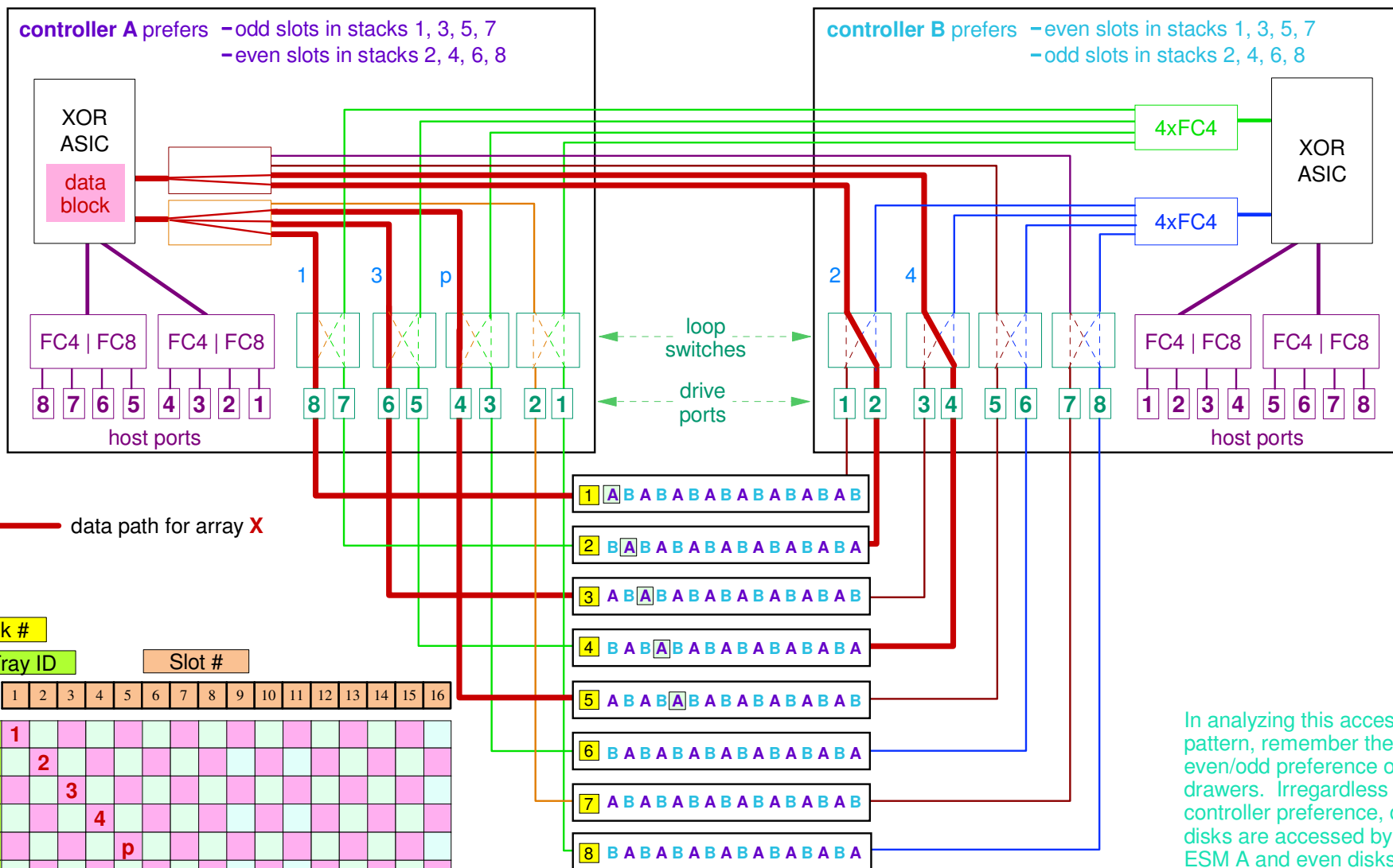
A vertical path and a diagonal path can not both be active at the same time.

★ Loop Switches



DS5300

Data Flow Example #1A



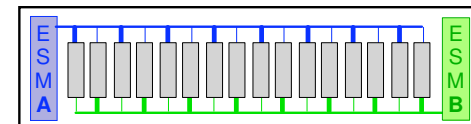
Array X: 4+P RAID 5, owned by controller A
Array Y: 4+P RAID 5, owned by controller B
Tray protected ("barber pole"), optimum performance

Mapping Disks to Array Rule:

Assign disks to arrays diagonally with 1 per tray as shown.

Array Ownership Rule:

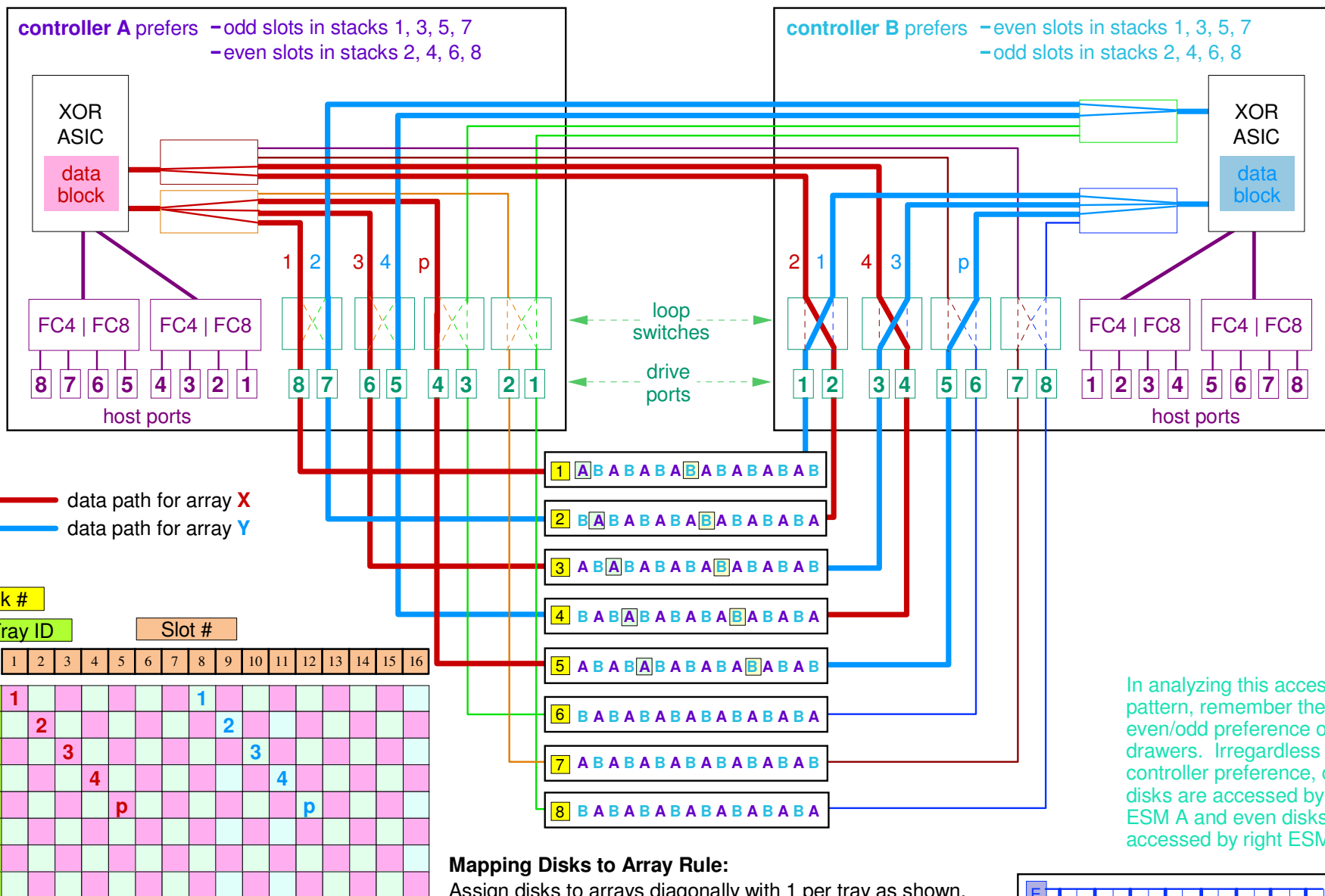
Assign array to controller accessing the first disk in the array.





DS5300

Data Flow Example #1B



Array X: 4+P RAID 5, owned by controller A
Array Y: 4+P RAID 5, owned by controller B
Tray protected, optimum performance

Mapping Disks to Array Rule:
Assign disks to arrays diagonally with 1 per tray as shown.

Array Ownership Rule:
Assign array to controller accessing the first disk in the array.



DS5300



Sample Complete Disk to Array Mappings for Example #1

4+P RAID 5, Tray Protected with optimum performance

11	A1	B2	A3	B4	A7	B8	A9	B10	A13	B14	A17	B18	A19	B20	A23	B24
25	B24	A1	B2	A3	B4	A7	B8	A11	B12	A13	B14	A17	B18	A19	B20	A23
31	A23	B24	A1	B2	A5	B6	A7	B8	A11	B12	A13	B14	A17	B18	A21	B22
45	B22	A23	B24	A1	B2	A5	B6	A7	B8	A11	B12	A15	B16	A17	B18	A21
51	A21	B22	HS	HS	A1	B2	A5	B6	A9	B10	A11	B12	A15	B16	A17	B18
65	B20	A21	B22	HS	HS	A3	B4	A5	B6	A9	B10	A11	B12	A15	B16	A19
71	A19	B20	A21	B22	HS	HS	A3	B4	A5	B6	A9	B10	A13	B14	A15	B16
85	B16	A19	B20	A23	B24	HS	HS	A3	B4	A7	B8	A9	B10	A13	B14	A15

Arrays
owned by
Controller A

Arrays
owned by
Controller B

8+2P RAID 6, Tray Protected with optimum performance

11	A1	B2	A3	B4	A7	B8	A9	B10	A13	B14	A17	B18	A19	B20	A23	B24
12	B24	A1	B2	A3	B4	A7	B8	A9	B10	A13	B14	A17	B18	A19	B20	A23
25	A23	B24	A1	B2	A3	B4	A7	B8	A11	B12	A13	B14	A17	B18	A19	B20
26	B20	A23	B24	A1	B2	A3	B4	A7	B8	A11	B12	A13	B14	A17	B18	A19
31	A21	B22	A23	B24	A1	B2	A5	B6	A7	B8	A11	B12	A13	B14	A17	B18
32	B18	A21	B22	A23	B24	A1	B2	A5	B6	A7	B8	A11	B12	A13	B14	A17
45	A17	B18	A21	B22	A23	B24	A1	B2	A5	B6	A7	B8	A11	B12	A15	B16
46	B16	A17	B18	A21	B22	A23	B24	A1	B2	A5	B6	A7	B8	A11	B12	A15
51	A15	B16	A17	B18	A21	B22	HS	HS	A1	B2	A5	B6	A9	B10	A11	B12
52	B12	A15	B16	A17	B18	A21	B22	HS	HS	A1	B2	A5	B6	A9	B10	A11
65	A11	B12	A15	B16	A19	B20	A21	B22	HS	HS	A3	B4	A5	B6	A9	B10
66	B10	A11	B12	A15	B16	A19	B20	A21	B22	HS	HS	A3	B4	A5	B6	A9
71	A9	B10	A13	B14	A15	B16	A19	B20	A21	B22	HS	HS	A3	B4	A5	B6
72	B6	A9	B10	A13	B14	A15	B16	A19	B20	A21	B22	HS	HS	A3	B4	A5
85	A7	B8	A9	B10	A13	B14	A15	B16	A19	B20	A23	B24	HS	HS	A3	B4
86	B4	A7	B8	A9	B10	A13	B14	A15	B16	A19	B20	A23	B24	HS	HS	A3

There is little need for hot spares under RAID 6. The 16 "extra" disks could be configured as 2 x 4+4 RAID 10 arrays.

It is an open question as to whether these should be used as GPFS metadataOnly disks. While their capacity is more than enough, there may not be enough available to sustain the required metadata IOP rates for the other 24 arrays under IOP intensive workloads. But they could be used as a cache under GPFS ILM for frequently accessed files.

Best Practice: Adopt tray protection using the following configurations.

8 trays using 4+P RAID 5 or 4+2P RAID 6

16 trays using 4+P or 8+P RAID 5, or 8+2P RAID 6

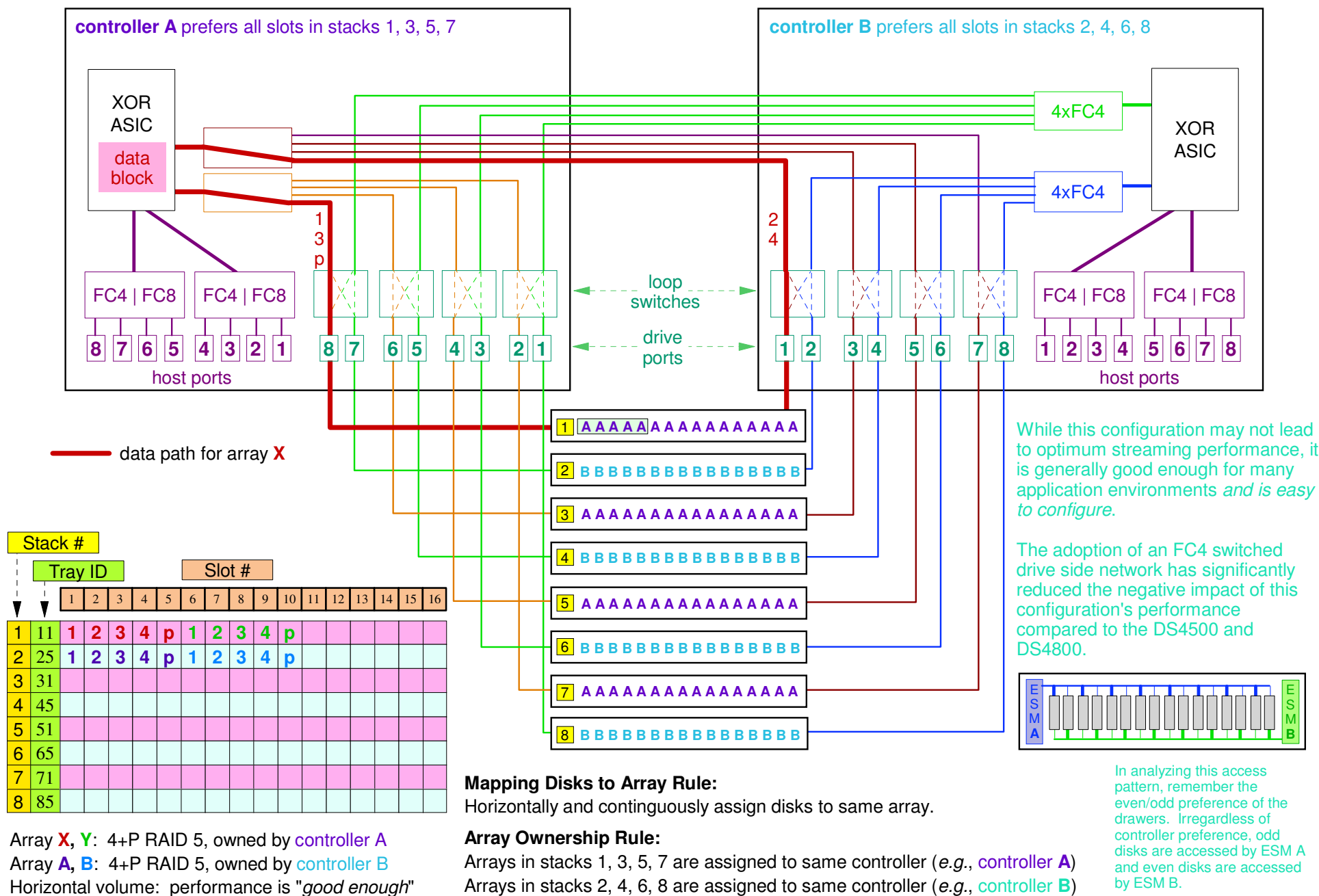


This is generally called a "barber pole" configuration.



DS5300

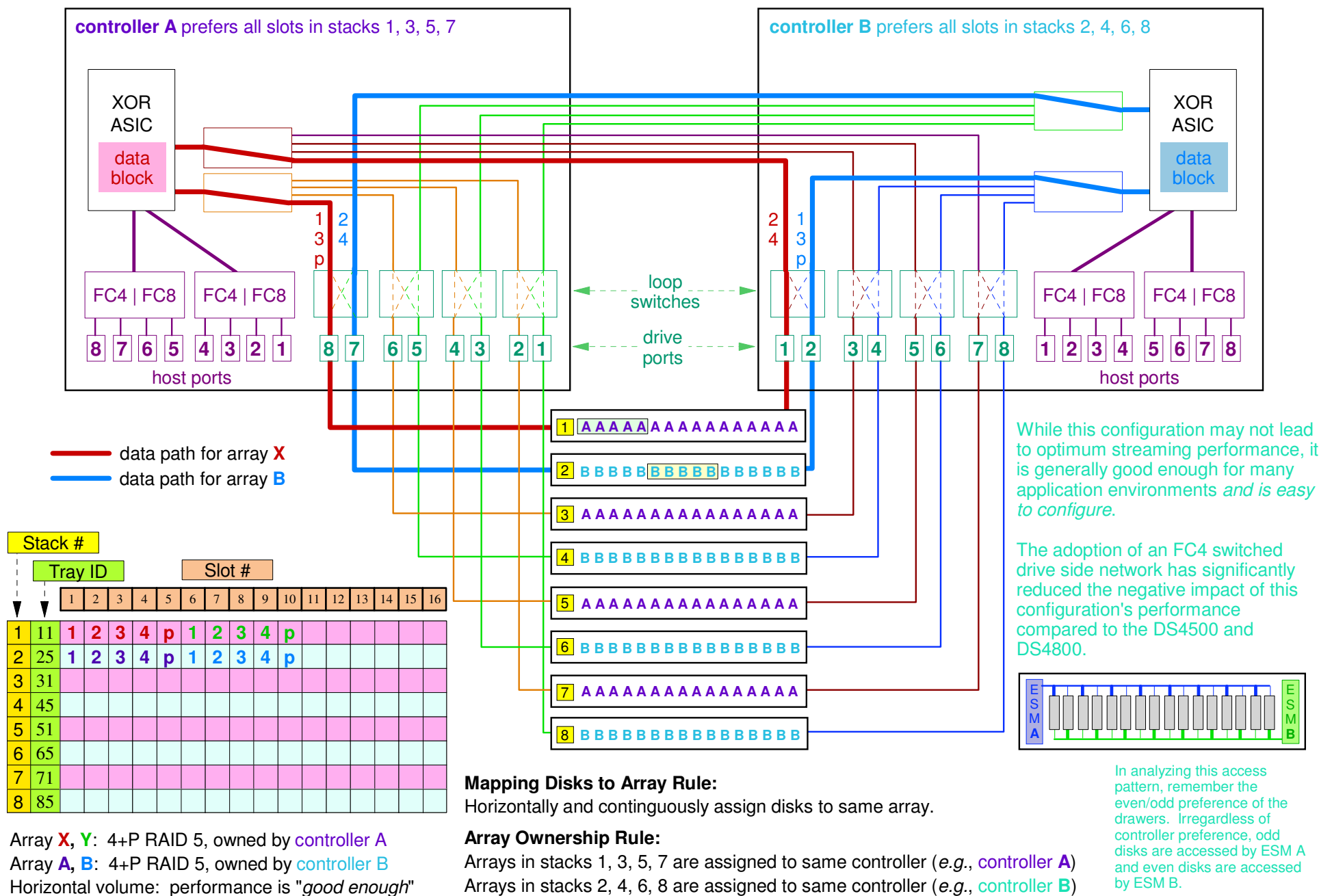
Data Flow Example #2A





DS5300

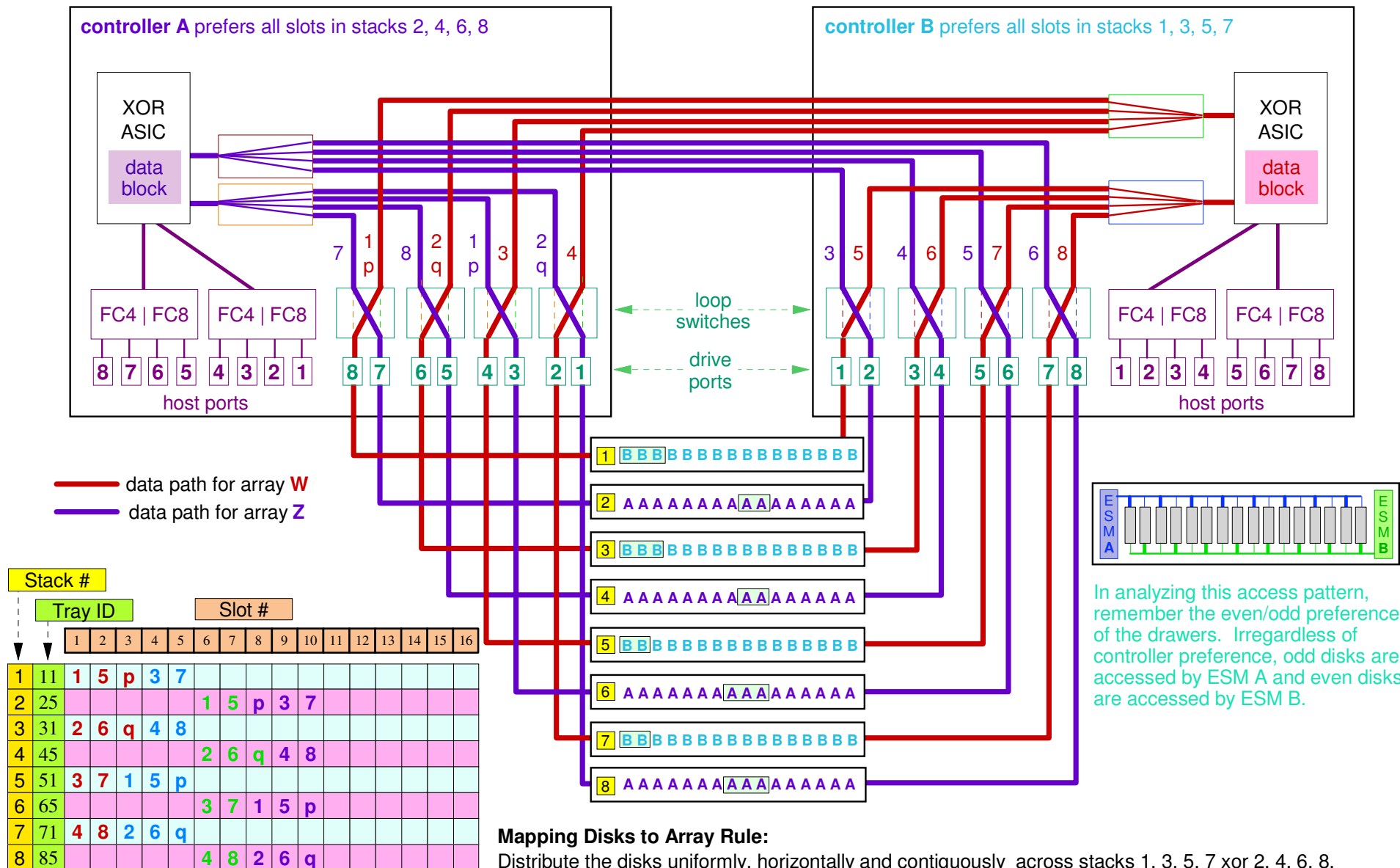
Data Flow Example #2B





DS5300

Data Flow Example #2C



Array **W**, **X**: 8+P+Q RAID 6, owned by **controller B**
Array **Y**, **Z**: 8+P+Q RAID 6, owned by **controller A**
Stack oriented: performance is "good enough"

Mapping Disks to Array Rule:

Distribute the disks uniformly, horizontally and contiguously across stacks 1, 3, 5, 7 xor 2, 4, 6, 8.

Array Ownership Rule:

Arrays in stacks 1, 3, 5, 7 are assigned to same controller (e.g., **controller B**)
Arrays in stacks 2, 4, 6, 8 are assigned to same controller (e.g., **controller A**)

I swapped the controllers around to make a point.



DS5300



Sample Complete Disk to Array Mappings for Example #2C

8+2P RAID 6, Stack Oriented with "good enough" performance

11	A1	A1	A1	A3	A3	A5	A5	A7	A7	A7	A9	A9	A11	A11	A11	HS
25	B2	B2	B2	B4	B4	B6	B6	B8	B8	B8	B10	B10	B12	B12	B12	HS
31	A1	A1	A1	A3	A3	A5	A5	A5	A7	A7	A9	A9	A11	A11	A11	HS
45	B2	B2	B2	B4	B4	B6	B6	B6	B8	B8	B10	B10	B12	B12	B12	HS
51	A1	A1	A3	A3	A3	A5	A5	A5	A7	A7	A9	A9	A9	A11	A11	HS
65	B2	B2	B4	B4	B4	B6	B6	B6	B8	B8	B10	B10	B10	B12	B12	HS
71	A1	A1	A3	A3	A3	A5	A5	A7	A7	A7	A9	A9	A9	A11	A11	HS
85	B2	B2	B4	B4	B4	B6	B6	B8	B8	B8	B10	B10	B10	B12	B12	HS

Arrays
owned by
Controller A

Arrays
owned by
Controller B

Alternatively, use the "extra" disks to create 4+4 RAID 10 arrays instead of using them as hot spares.

8+2P RAID 6, Stack Oriented with "good enough" performance

11	A1	A1	A3	A5	A7	A9	A11	A13	A15	A15	A17	A17	A21	A23	A23	HS
12	B2	B2	B4	B6	B8	B10	B12	B14	B16	B16	B18	B18	B22	B24	B24	HS
25	A1	A1	A3	A5	A7	A9	A11	A13	A15	A15	A17	A19	A19	A21	A23	HS
26	B2	B2	B4	B6	B8	B10	B12	B14	B16	B16	B18	B20	B20	B22	B24	HS
31	A1	A3	A3	A5	A7	A9	A11	A13	A13	A15	A17	A19	A19	A21	A23	HS
32	B2	B4	B4	B6	B8	B10	B12	B14	B14	B16	B18	B20	B20	B22	B24	HS
45	A1	A3	A3	A5	A7	A9	A11	A13	A13	A15	A17	A19	A21	A21	A23	HS
46	B2	B4	B4	B6	B8	B10	B12	B14	B14	B16	B18	B20	B22	B22	B24	HS
51	A1	A3	A5	A5	A7	A9	A11	A11	A13	A15	A17	A19	A21	A21	A23	HS
52	B2	B4	B6	B6	B8	B10	B12	B12	B14	B16	B18	B20	B22	B22	B24	HS
65	A1	A3	A5	A5	A7	A9	A11	A11	A13	A15	A17	A19	A19	A21	A23	HS
66	B2	B4	B6	B6	B8	B10	B12	B12	B14	B16	B18	B20	B20	B22	B24	HS
71	A1	A3	A5	A7	A7	A9	A9	A11	A13	A15	A17	A19	A19	A21	A23	HS
72	B2	B4	B6	B8	B8	B10	B10	B12	B14	B16	B18	B20	B20	B22	B24	HS
85	A1	A3	A5	A7	A7	A9	A9	A11	A13	A15	A17	A17	A21	A23	A23	HS
86	B2	B4	B6	B8	B8	B10	B10	B12	B14	B16	B18	B18	B22	B24	B24	HS

However, be sure to distribute them in a more optimal pattern across the trays, such as a "barber pole" distribution.

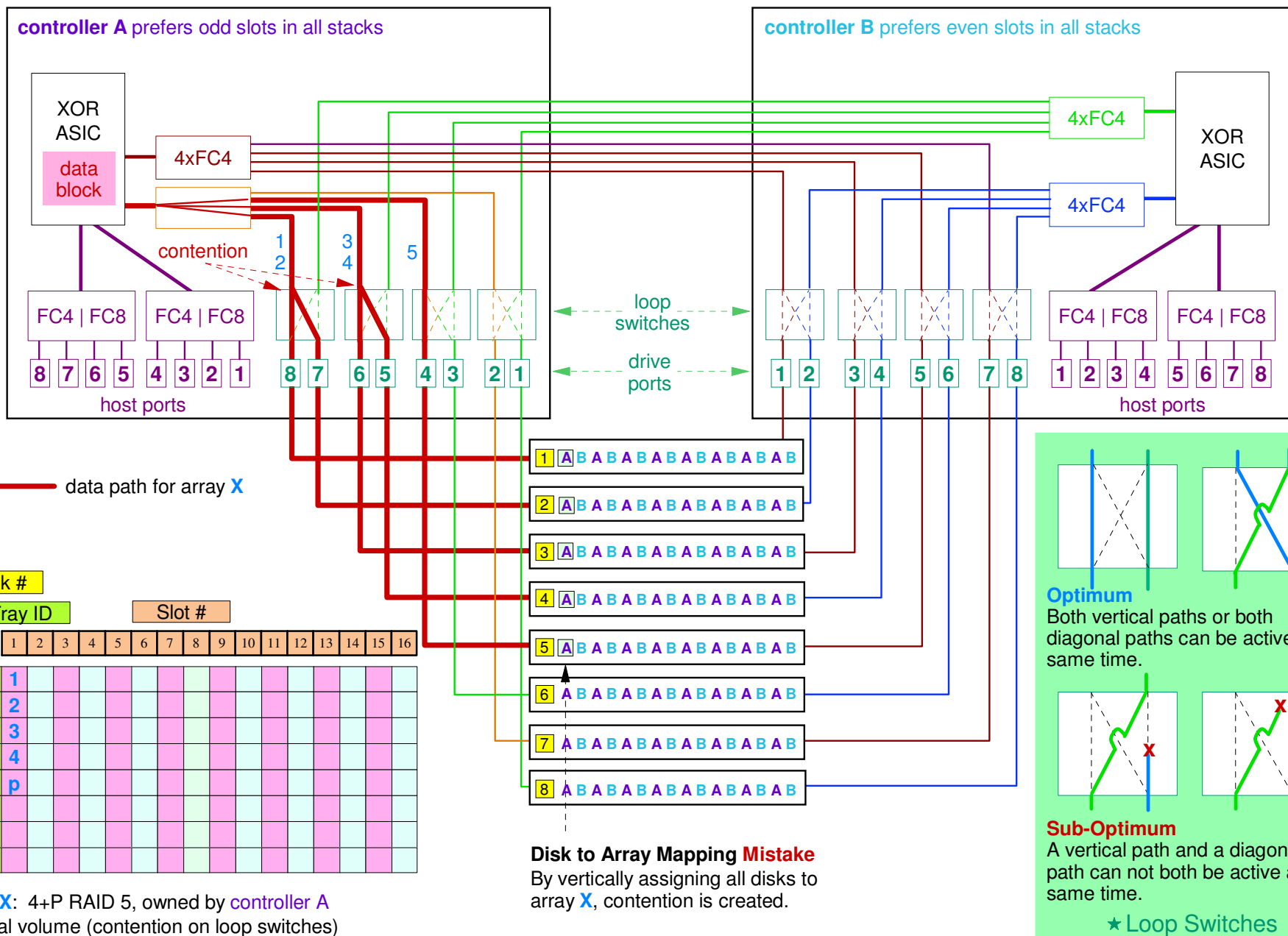
COMMENT: Hot spares vs. RAID 6

There is little need for hot spares with RAID 6, but in a 8 tray configuration, there is not room for another 8+2P RAID 6 array. Therefore, configure the other 8 disks as a 4+4 RAID 10 array. In the 16 tray configuration, there is room for 1 more 8+2P RAID 6 array, but this will create an imbalance that will hurt GPFS performance. Therefore configure the other 16 disks as 2 x 4+4 RAID 10 arrays.



DS5300

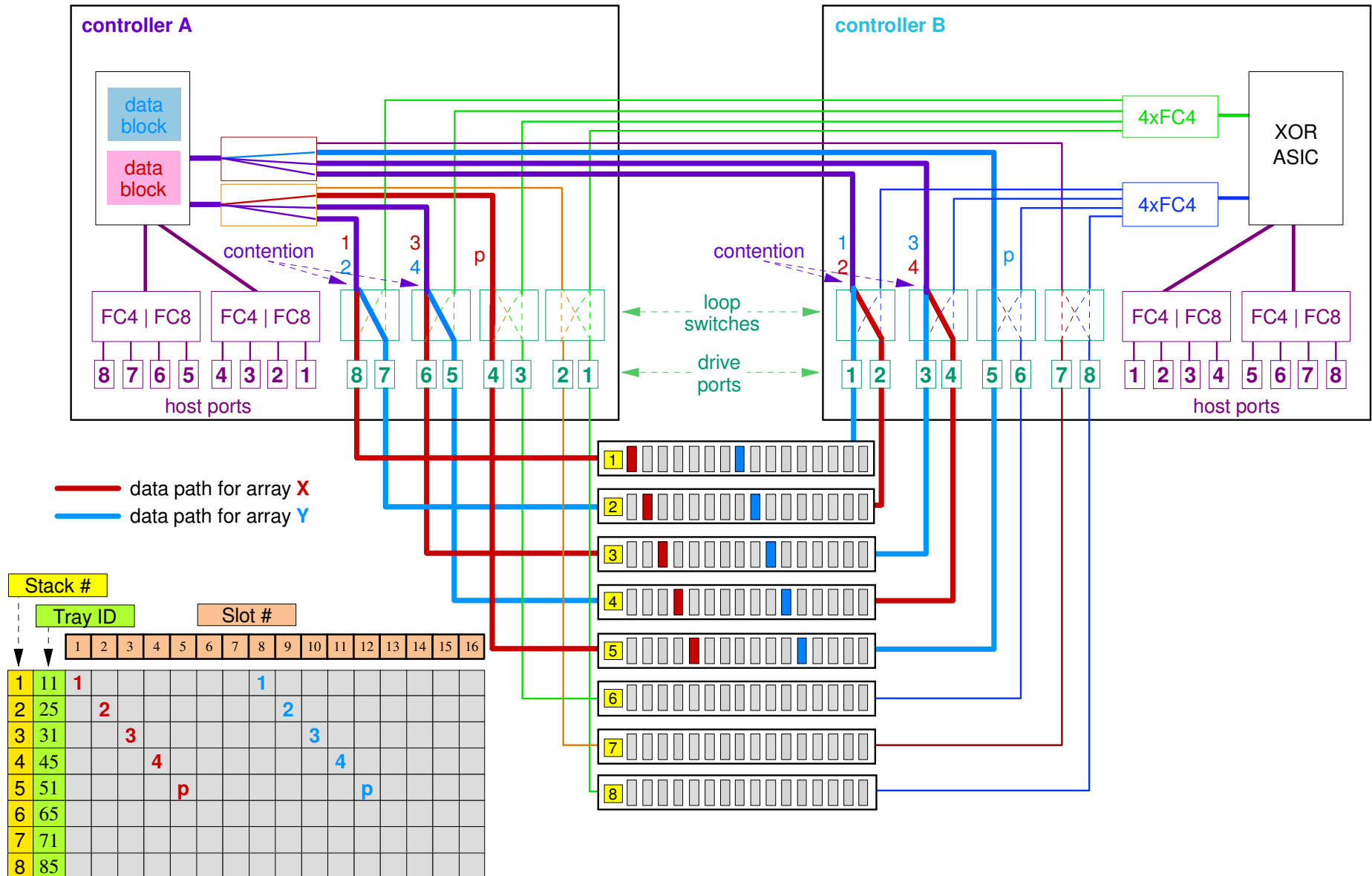
Data Flow Example #3





DS5300

Data Flow Example #4



Array X: 4+P RAID 5, owned by controller A

Array Y: 4+P RAID 5, owned by controller A

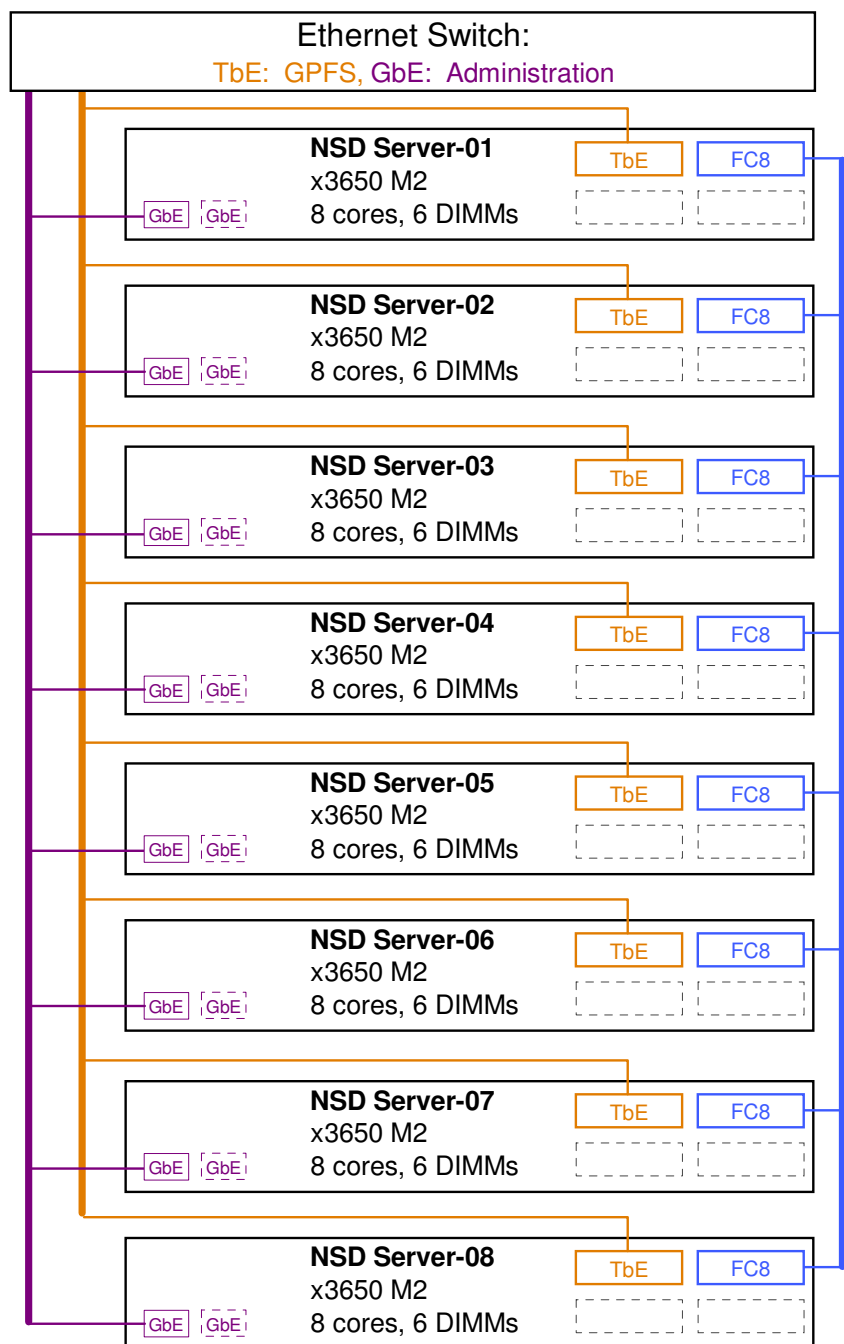
Array Ownership Mistake

By assigning array Y to controller A, contention is created.



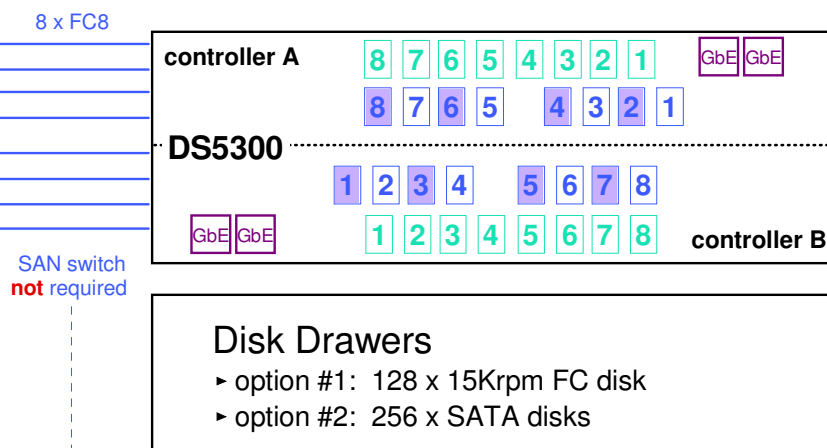
DS5300

Example Configuration



Performance Analysis

- ▶ DS5300 streaming data rate
 - 256 x SATA or 128 x 15Krpm disks: write < 4.5 GB/s, read < 5.5 GB/s
- ▶ DS5300 IOP rate
 - 256 x SATA disks: write < 3600 IOP/s, read < 12,000 IOP/s
 - 128 x 15Krpm disks: write < 9,000 IOP/s, read < 36,000 IOP/s
- ▶ potential aggregate TbE rate: 8 x TbE < 5.6 GB/s
 - 725 MB/s per TbE is possible, but 700 MB/s is required
- ▶ potential aggregate FC8 rate: 8 x FC8 < 6.0 GB/s
 - 780 MB/s per FC8 is possible, but 700 MB/s is required



COMMENT:

- ▶ This is a "safe" configuration in the sense that meeting projected performance rates can reasonably be expected (n.b., there are more than enough servers, FC8 and TbE ports to do the job).
- ▶ If HBA failover is required, then 8 dual port HBAs may be adopted (thereby requiring a SAN switch). If 2xFC8 adapters are adopted, then peak performance can be maintained during failure conditions.



DS5300

Benchmark Results



To Be Completed

These and other measurements to be validated using GPFS

- ▶ Peak Write Rates

- Write cache enabled, mirroring disabled : **TBD**
- Write cache enabled, mirroring enabled, FSWT enabled: **TBD**
- Write cache disabled: **TBD**

- ▶ Peak Read Rates

- Read cache enabled: **TBD**



DS5020



- **The DS5020 is an upgrade from the DS4700**
- **Its performance profile is roughly equivalent to the earlier generation DS4800**
 - peak streaming rates < 1500 MB/s
- **It supports a maximum of 112 disks**
- **It uses the EXP5000 disk drawers with FC and/or SATA disk**
- **Compared with the DS5300**
 - RAID 6 overhead comprises a greater percentage of processing time (*e.g.*, ~ = 25%) cf. the DS5300 (*e.g.*, 10%)
 - Write cache mirroring is not as effective
- **Best practice: use increments of 2 drawers**





DS3000/DS5000 Series Comparison

DS model	DS3200 ¹	DS3400 ¹	DS5020	DS5100	DS5300
Max disks	48	48	112	256	448
Internal disks	12	12	16	0	0
Host interfaces	SAS @ 3 Gb/s 6 ports	FC @ 1/2/4 Gb/s 4 ports		FC @ 4/8 Gb/s 8 ports	FC @ 4/8 Gb/s 16 ports
Drive interfaces	SAS @ 3 Gb/s 2 ports	SAS @ 3 Gb/s 2 ports		4 Gb/s 16 ports	4 Gb/s 16 ports
Max cache memory	2 GB	2 GB		8 GB	16 GB
IOPS from cache read ²	96,000	120,000		700,000	700,000
IOPS from disk read ²	19,000	21,500		75,000	172,000
IOPS from disk write ²	4,200	4,600		20,000	45,000
BW from cache read (MB/s) ²	1670	1630		3200	6,400
BW from disk read (MB/s) ²	895	940		3200	6,400
BW from disk write (MB/s) ²	695	725		2,500	5,300
RAID Levels	0, 1, 3, 5, 6, 10	0, 1, 3, 5, 6, 10		0, 1, 3, 5, 6, 10	0, 1, 3, 5, 6, 10
FC 15 Krpm				146, 300, 450 @ 4Gb/s	146, 300, 450 @ 4Gb/s
SAS 15 Krpm	146, 300, 450 @ 3Gb/s	146, 300, 450 @ 3Gb/s			
SATA 7.2 Krpm	750, 1000 GB @ 3Gb/s	750, 1000 GB @ 3Gb/s		750, 1000 GB @ 4Gb/s	750, 1000 GB @ 4Gb/s
Disk Enclosures	EXP3000	EXP3000		EXP5000, EXP810	EXP5000, EXP810

1. Not intended for use in large capacity storage systems. Best practices suggest not using more than 4 units under the control of a single file system.
2. Data rates are reported as peak theoretical values and are not feasible in a production environment; they are intended for comparison purposes only.
3. This refers specifically to the 1814-72A.



DCS9900



4u



Couplet Front View

2u



Rear View of "Half of a Couplet"

A single couplet can support up to 1200 disks (i.e., 2 frames).

45u



Couplet
"dual RAID controller"

Disk Enclosure

10 Disk Enclosures per Frame
60 Disks per Enclosure
4u per Enclosure



DCS9900



I can't find an image of the IBM tray.

- ▶ Couplet
 - Dual RAID controller design
 - Active/active design
 - 5 GB of cache
 - RAID level: 8+2P RAID 6 only
 - 8 host side ports
 - FC8 or IB 4x DDR2
 - 20 drive side ports
 - 3 GB/s SAS connections
 - facilitates fast RAID rebuild

Can sustain up to 4 RAID rebuilds at the same time without a noticeable impact on performance.

- ▶ Disk Trays
 - Up to 60 disks per tray
 - Up to 20 trays (1200 disks) per couplet
 - Supports SAS and SATA
 - SAS: 450 GB
 - SATA: 1 TB, 2 TB*

- ▶ Peak Performance (**theoretical**⁺)
 - Streaming (using 300 x SATA disks)
 - write < 4.5 GB/s (4M/transaction)
 - read < 5.9 GB/s (4M/transaction)

2 TB @ 7200 RPM will be available in Q1/10

- IOP Rates
 - 40,000 IOP/s (4K/transaction)

FOOTNOTES:

★ Currently, 2 TB drives are only available at 5400 RPM. They will be available at 7200 RPM in Q1/10.

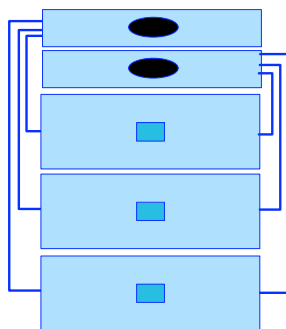
✦ These are upper bound rates based on lab measurements using specialized tuning parameters and workload assumptions. They demonstrate what the DCS9900 can do. Actual performance rates will not exceed these values.



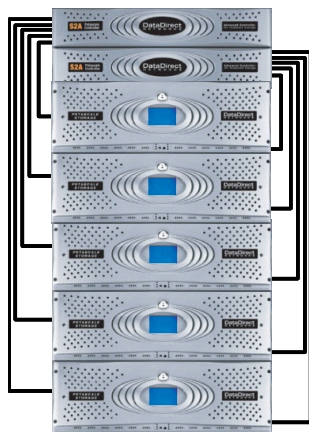
DCS9900

Physical View

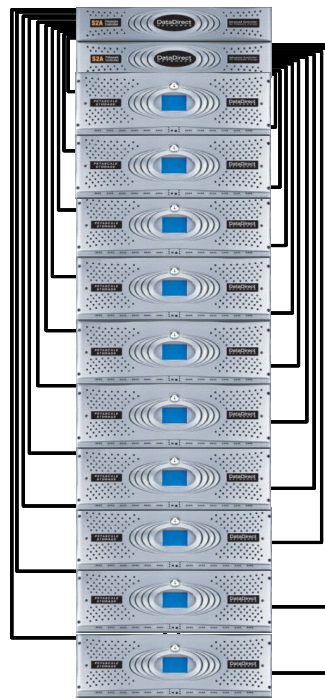
Illustrations shown using 60-bay enclosures (model 3S1).
IBM also supports a 16-bay enclosure though it is seldom used.



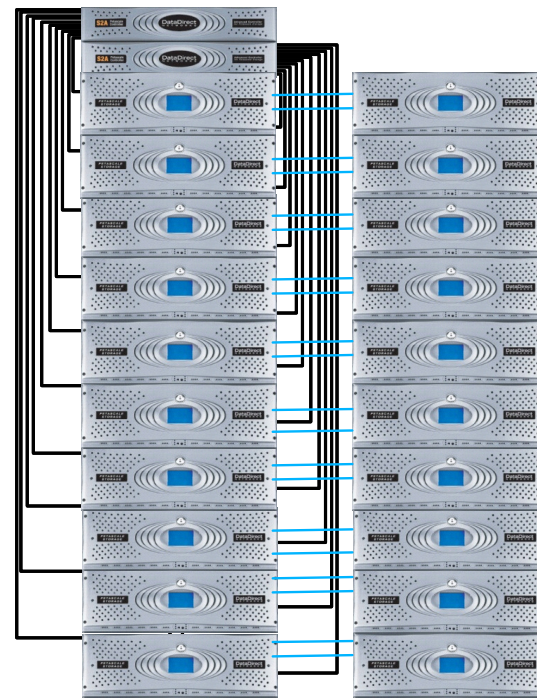
3 x 60-slot Trays
SAS xor SATA
150 disks
Capacity
► 300 TB using 2 TB SATA
► 66 TB using 450 GB SAS



5 x 60-slot Trays
SAS xor SATA
150 to 300 disks
Capacity
► 600 TB using 2 TB SATA
► 135 TB using 450 GB SAS



10 x 60-slot Trays
SAS and/or SATA
150 to 600 disks
Capacity
► 1200 TB using 2 TB SATA
► 270 TB using 450 GB SAS



20 x 60-slot Trays
SAS and/or SATA
150 to 1200 disks
Capacity
► 2400 TB using 2 TB SATA
► 540 TB using 450 GB SAS

COMMENT: To maximize performance per capacity, peak performance can be achieved using as few as 160 x 15 Krpm SAS drives or with 300 SATA drives. To minimize cost per capacity, the number of drives can be increased up to 1200.



DCS9900

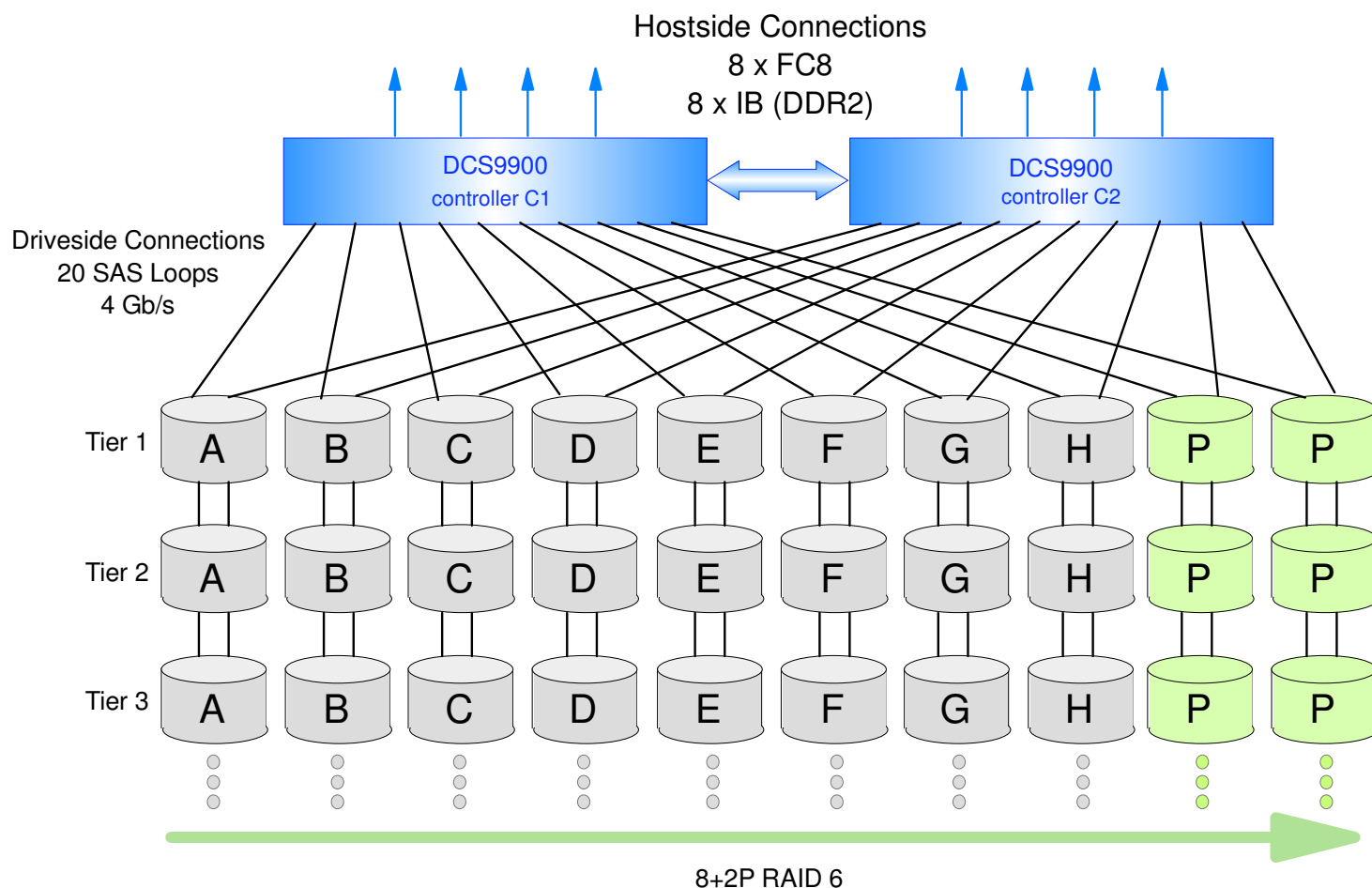
Controller Overview

DCS9900 RAID configuration

- 8+2P RAID 6
- Data accessed using a "byte striping algorithm"

Supported sector sizes are

- 512, 1024, 2048, 4096 bytes
- GPFS only supports 512 bytes



COMMENTS

- Recommend creating only 1 LUN per tier for GPFS
- Parity is computed for each write I/O operation
- Parity is checked for each read I/O operation



DCS9900

Configuration and Parameter Explanation and Guidelines

■ DCS9900 Cache Organization

- There is 2.5 GB of cache per RAID controller for a total of 5 GB
- The cache page size is a configurable parameter
 - set this using the command "cache size=<int>"
 - valid choices are 64, 128, 256, 512, 1024, 2048, 4096 (units are in KB)
 - **Best Practice:** set the cache size to 1024, 2048 or 4096
 - optimum streaming performance occurs for any of these values
 - if GPFS blocksize is 2M or 4M, then cache size = 1024 or 2048 or 4096 gives same performance (n.b., a cache size smaller than the GPFS blocksize is OK!)

■ Setting the OS transfer size

- Set cache page size \geq OS transfer size
 - **Best Practice:** set them to be the same value
- AIX
 - `chdev -l fcs<int> -a max_xfer_size=<hex value>`
 - default = 0x100000 (i.e., 1 MB)
- Linux
 - set the `max_sectors_kb` parameter to the DCS9900 cache size
 - located in `/sys/block/<SCSI device name>/queue/max_sectors_kb`
 - typical SCSI device names are `sdb`, `sdc`, `sdd`, `sde`, ...
 - These changes are not persistent, therefore this must be reset after every reboot.



DCS9900

Configuration and Parameter Explanation and Guidelines

■ Write Caching: Write Back vs. Write Thru

- Enabling write back caching instructs the DCS9900 to write data blocks to cache and return control to the OS; data in the cache is actually written to disk later. If write thru caching is enabled, all data is written *both* to cache and disk before control is returned to the OS.
- Enable write back caching using the command "cache writeback=on"
 - warning: if a controller fails, all data in its cache will be lost, possibly before it is written to disk
 - This can corrupt the file system since metadata can be lost.
 - Adopt proper risk management procedures if write back caching is enabled.
- Enable write thru by setting "cache writeback=off"
 - **Best Practice:** disable write back caching
 - this will significantly degrade performance

■ Cache coherence

- The DCS9900 has the concept of "LUN ownership" or "LUN affinity". The controller in the couplet that created the LUN owns that LUN.
- Both controllers in a DCS9900 couplet can *both* see a given LUN (even though only one of them created it) iff cache coherence is enabled
- Cache coherence generally has a minimal performance degradation
- **Best Practice:** Enable cache coherence using the command "dual coherency=on"



DCS9900

Configuration and Parameter Explanation and Guidelines

■ Read Caching: Prefetch

- DCS9900 read caching uses a prefetch algorithm. The setting for this parameter is dependent on the GPFS block allocation map setting.
- **Best practice:** set GPFS block allocation map to scatter
 - assumes that number of nodes > 8 or number of LUNs > 8
 - scatter vs. cluster
 - There are 2 block allocation map types for GPFS: scatter or cluster
 - scatter: randomly distribute file blocks over the LUN
 - cluster: write file data in clusters of contiguous disk blocks
 - COMMENT: There is no guarantee that contiguous file blocks on disk will be accessed in the same order that they are mapped to disk. This problem is exacerbated by increasing "disk entropy" through repeated create/delete cycles. Scatter guarantees uniform performance for multitask jobs accessing a common file.
 - set MF bit using the command "cache mf=on"
 - disable prefetching using the command "cache prefetch=0"
 - since file blocks are randomly distributed, prefetching hurts performance



DCS9900

Configuration and Parameter Explanation and Guidelines

■ User Data vs. Meta Data

- **Best Practice: Segregate User Data and Meta Data**
 - DCS9900 does streaming well, but randomly distributed small transactions not as well. Since meta data transactions are small, segregating user data and meta data can improve performance for meta data intensive operations.
- Caveats and warnings
 - Most beneficial in environments with significant meta data processing
 - Must have enough dedicated metadataOnly LUNs on controllers with good enough IOP rates to keep pace with DCS9900 LUNs.

■ FC Drivers

- Linux
 - GPFS uses the qla2400 driver for FC access to the DCS9900
 - this driver is AVT, **not** RDAC
 - Supports the DCS9900 active:active access model
- AIX
 - GPFS uses the MPIO driver in failover mode
 - Only supports an active:passive access model

■ Linux Multipathing

- While not officially supported, customers familiar with Linux multipathing report being able to get it work with GPFS and the DCS9900.



DCS9900

Configuration and Parameter Explanation and Guidelines

■ Summary of Selected DCS9900 Best Practice Settings

- 1 LUN per tier
- LUN block size = 512 (set interactively when the LUN is created)
- dual coherency=on
- cache size=1024 (assumes OS transfer size = 1MB)
- cache writeback=off
- cache mf=on
- cache prefetch=0
- ncq disabled

■ Summary of Selected GPFS Best Practice Settings

- pagepool >= 256M
- maxMBpS ~= 2X to 3X LAN connection data rate
- maxblocksize = 4096K
- blocksize = 4M (assumes objective is to optimize streaming access)
- if feasible, segregate user data and meta data
- set GPFS block allocation = scatter (i.e., mmcrfs -j scatter)



DCS9900

Logical Configuration

The following page illustrates an optimum scheme for

- ▶ LUN to Port Mapping (i.e., zoning)
- ▶ selecting primary and backup servers for each LUN
- ▶ cabling

These criteria assume an NSD configuration defined later in these slides.

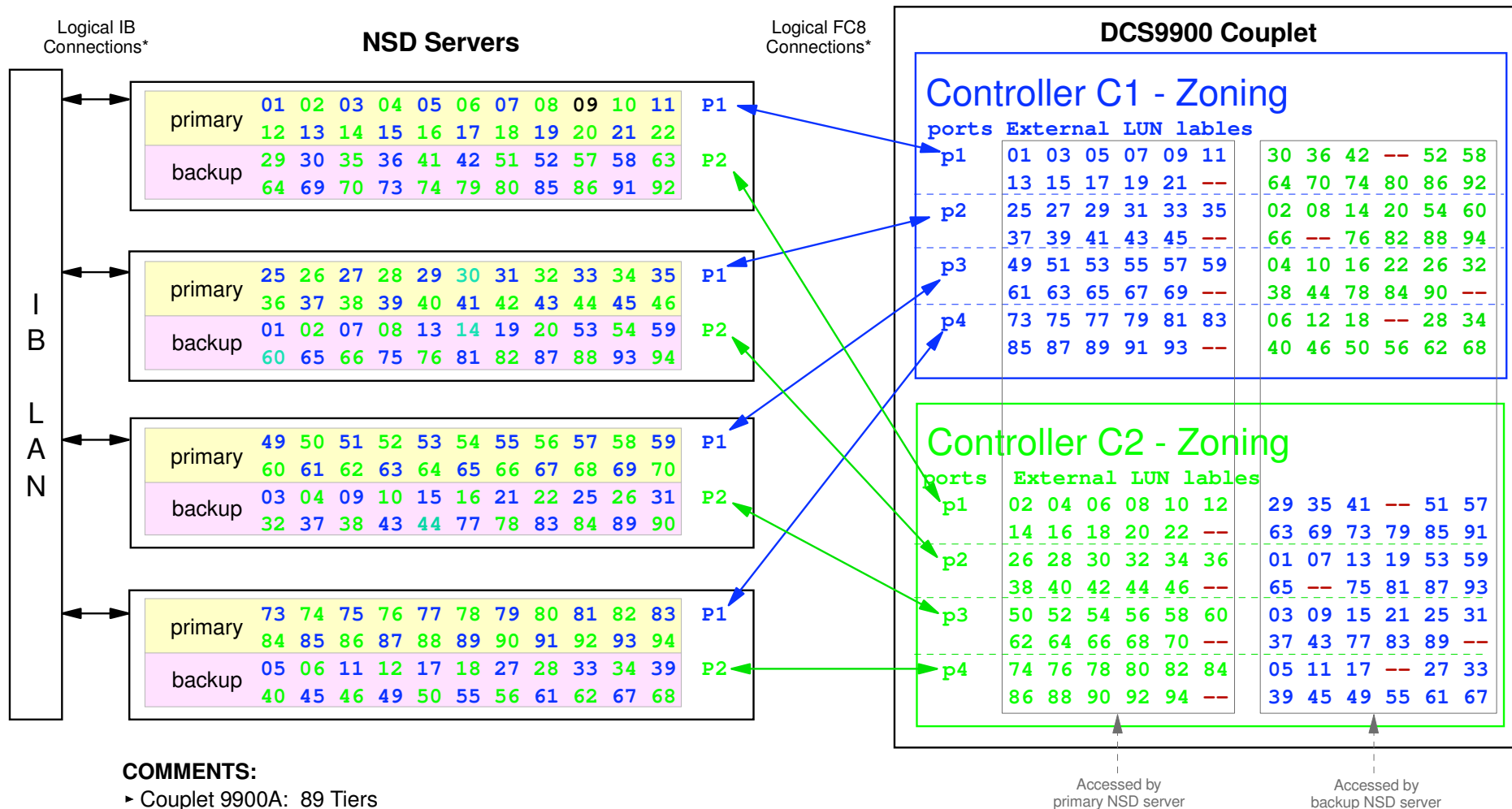
These schema are based on the following design criteria:

1. Guarantee that all controller ports and all HBA ports are uniformly active.
 - ▶ *n.b.*, the DCS9900 supports an active:active protocol
2. If a NSD server fails, its backup server can access its LUNs.
3. Consider LUNs associated with a given HBA or controller port. If an HBA or controller port fails, GPFS failover can access the associated LUNS over alternative paths from a backup NSD server for a given LUN *in a balanced manner (i.e., do not access all of the affected LUNs from a single NSD server)*.
 - ▶ This balance condition applies to performance under degraded conditions. It results in a slightly more complex logical configuration.
4. If one of the controllers in a couplet fails, the file system remains viable using backup NSD servers to access the LUNs of the failed controller over the other controller.



DCS9900

A Proper Logical Configuration



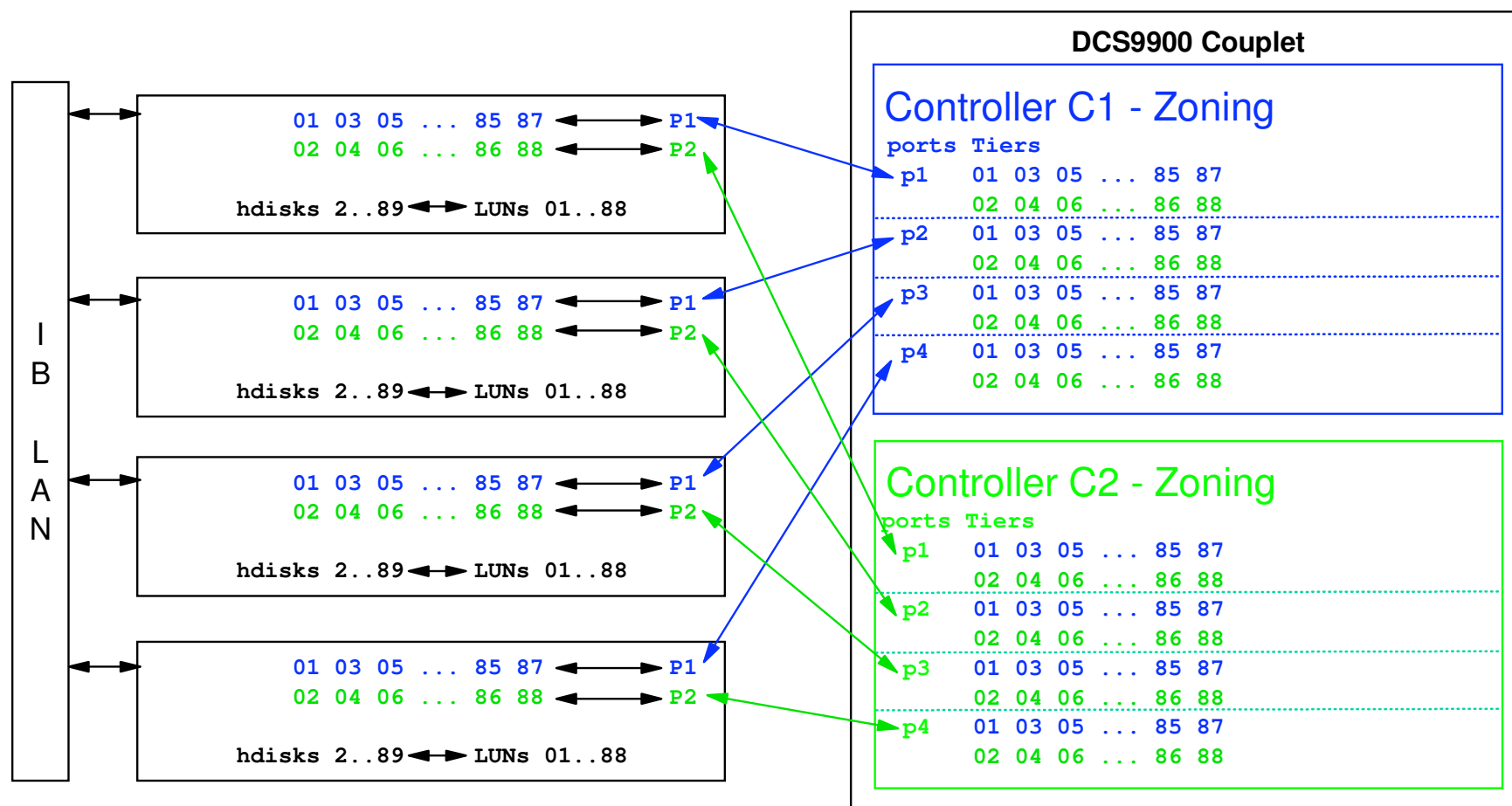
COMMENTS:

- ▶ Couplet 9900A: 89 Tiers
 - External LUN labels: 001..022, 025..046, 049..070, 073..094, 097
- ▶ Couplet 9900B: 90 Tiers
 - External LUN labels: 001..022, 025..046, 049..070, 073..094, 097, 098
- ▶ In order to improve managability skip external LUN lables 23, 24, 47, 48, 71, 72, 95, 96
- ▶ Controller C1 owns the odd LUNs, controller C2 owns the even LUNs
- ▶ In order to allow controller failover it is nessary to enable cache coherence
 - Command: dual coherency = ON



DCS9900

The Actual Logical Configuration



COMMENTS:

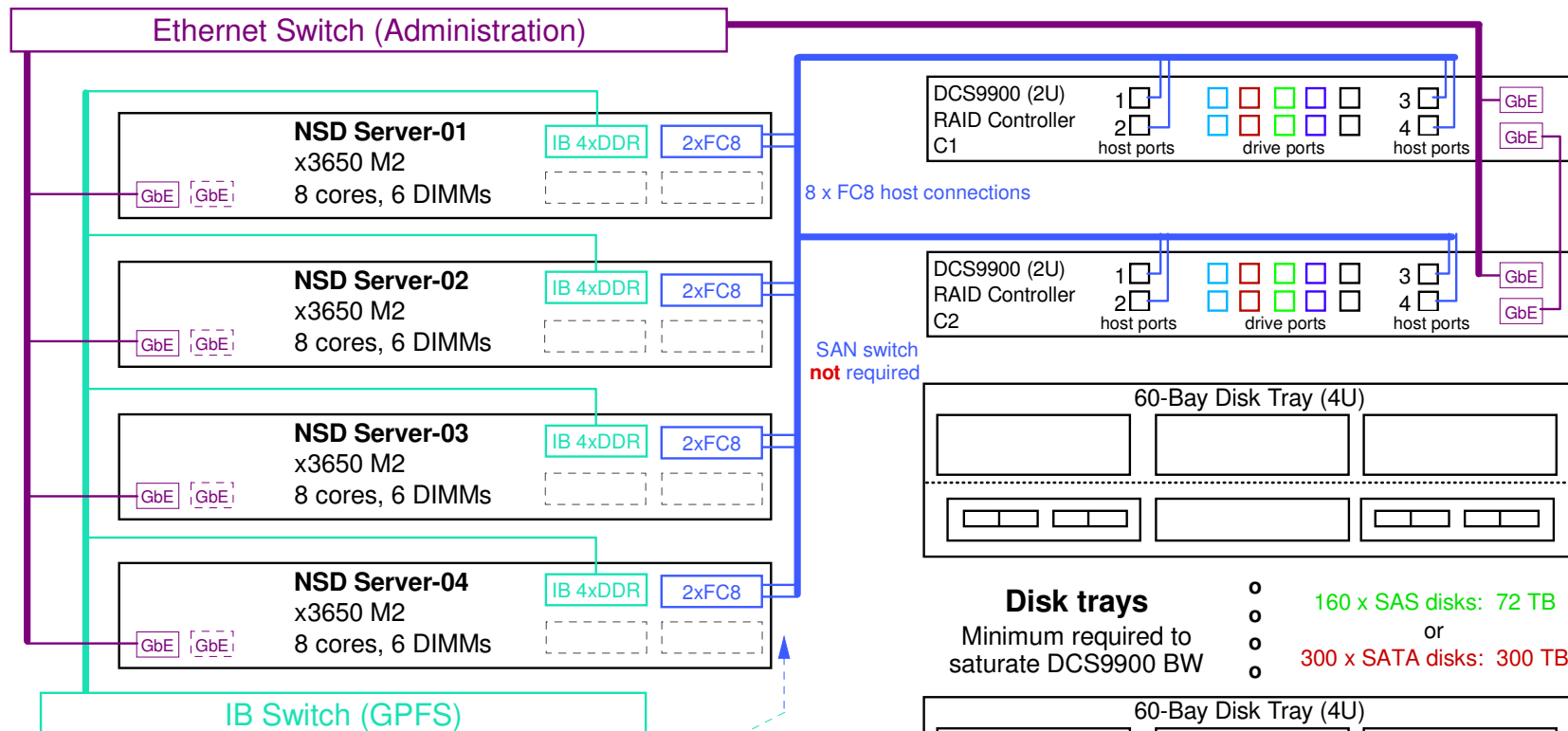
- ▶ GPFS is configured in a SAN mode.
- ▶ Since dual coherency=OFF for these tests, P1 sees only the LUNs owned by controller 1 and P2 sees only the LUNs owned by controller 2. **So there is no HBA failover.** This is simply a configuration error. This does not affect performance..

Controller 1 "owns" LUNs 01, 03, 05, ..., 87
Controller 2 "owns" LUNs 02, 04, 06, ..., 88
LUNs 89, 90 were not used.
dual coherency = OFF



DCS9900

Example Configuration



The **2xFC8 HBAs** can be replaced by dual port **4xDDR IB HCAs** using SRP. The IB host ports can either be directly attached to the servers or connected to a dedicated IB SAN switch. It is also possible to use an IB switch for a combined LAN and SAN, but this has been discouraged in the past. As a best practice, it is not recommend to use an IB SAN for more than 32 ports.

- * These are consistent well formed 4K transactions. A typical GPFS small transaction work load has a mixed transaction sizes resulting from metadata transactions.

Peak sustained DCS9900 performance

- ▶ streaming data rate < **5.6 GB/s**
 - ▶ noncached IOP rate < **40,000 IOP/s***
- 4xDDR IB HCA** (Host Channel Adapter)
- ▶ Potential peak data rate per **HCA** < 1500 MB/s
 - ▶ Required peak data rate per **HCA** < 1400 MB/s
- 2xFC8** (dual port 8 Gbit/s Fibre Channel)
- ▶ Potential peak data rate per **2xFC8** < 1500 MB/s
 - ▶ Required peak data rate per **2xFC8** < 1400 MB/s



DCS9900

Benchmark Results

GPFS Parameters

- ▶ blocksize(streaming) = 4096K
- ▶ blocksize(IOP) = 256K
- ▶ pagepool = 1 G
- ▶ maxMBpS = 4000

DCS9900 Parameters

- ▶ 8+2P RAID 6
- ▶ SATA
- ▶ cache size = 1024K
- ▶ cache prefetch = 0
- ▶ cache writeback = ON

Streaming Job

- ▶ record size = 4M
- ▶ file size = 32G
- ▶ number of tasks = 1 to 16
- ▶ access pattern = seq

COMMENT

The disparity between read and write performance observed below is much less pronounced when using 15Krpm SAS drives. For example, using 160 SAS tiers...

write ~= 5700 MB/s, read ~= 4400 MB/s

This disparity can be removed using cluster block allocation for SATA disk, but this not recommended.

4 NSD Servers, no GPFS clients

- ▶ P6-p520, 4 cores, 4.2 GHz, 8 GB RAM
- ▶ 2xFC8

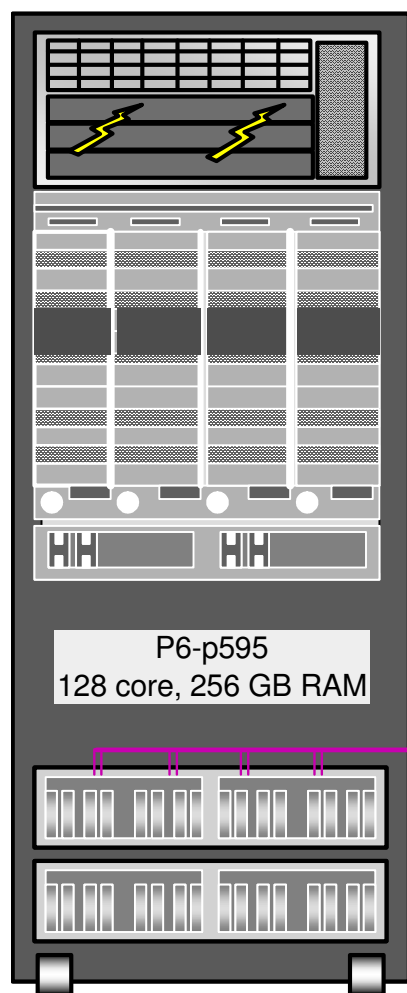
IOP Job

- ▶ record size = 4K
- ▶ total data accessed = 10G
- ▶ number of tasks = 32
- ▶ access pattern = small file (4K to 16K)

Access Patt	Tier	1	4	8	16	32	64
Streaming	write (MB/s)	270	790	1400	2700	4800	5400
Streaming	read (MB/s)	220	710	1200	1600	2900	3600
IOP*	write (IOP/s)	7,500	13,500	30,000	30,400	41,000	
IOP*	read (IOP/s)	3,800	5,900	27,300	27,300	33,500	

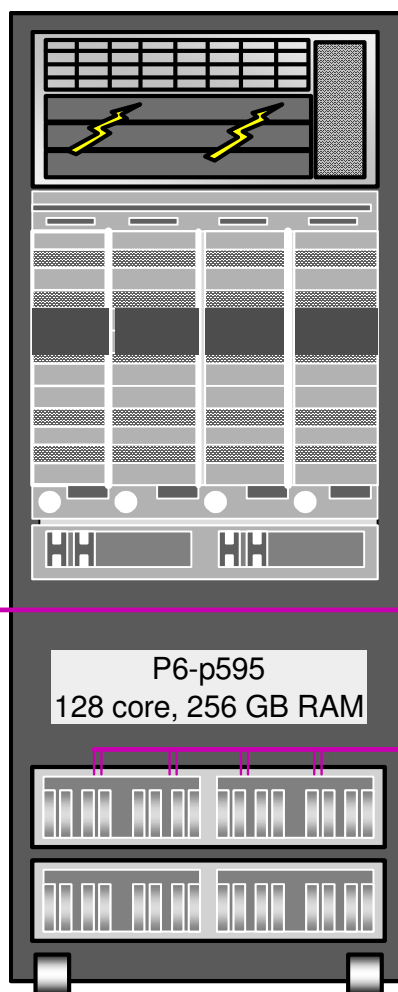


DS8000 Series



P6-p595 FC Ports

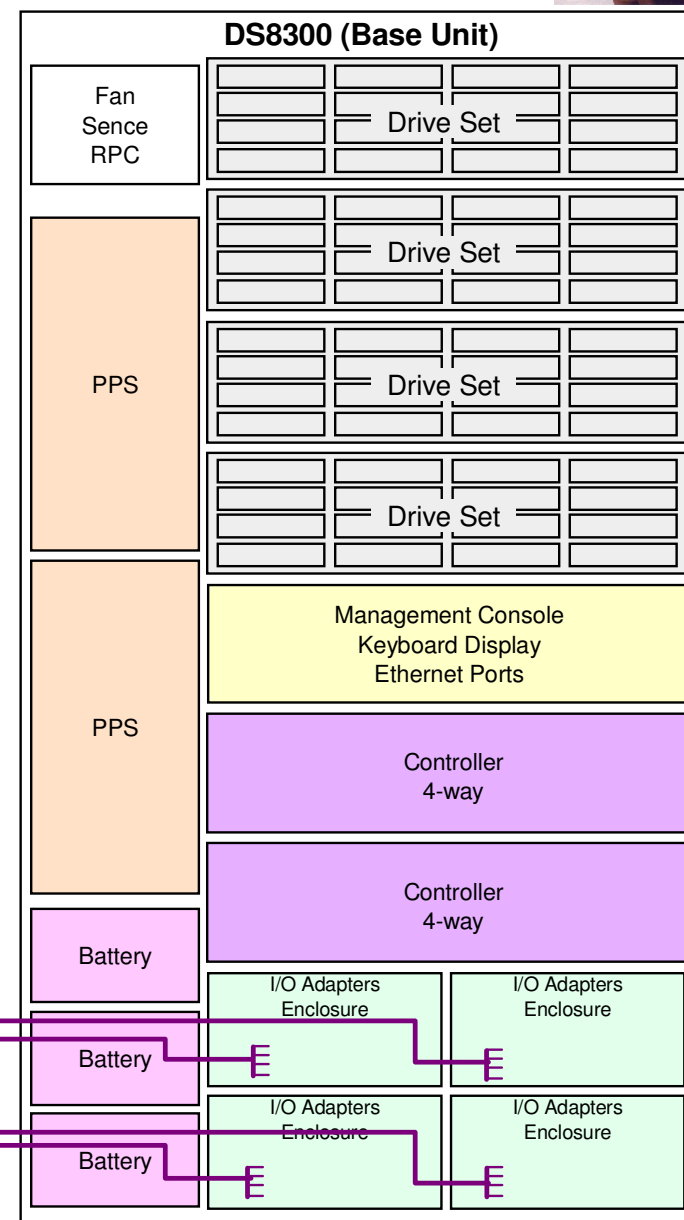
- ▶ 8 x 4 Gb/s FC ports per system
- ▶ ~ = 3 GB/s per system



8
S
A
N
S
W
I
T
C
H
8

DS8300 ANALYSIS

- ▶ Peak BW
 - read < 3500 MB/s
 - write < 1900 MB/s
 - duplex < 2300 MB/s
- ▶ FC Ports
 - 16 @ 4 Gb/s
- ▶ Disks
 - 128
 - max per base unit
 - 300 GB/disk @ 15Krpm
 - raw capacity ~ = 38 TB



COMMENT: The DS8000 RAID architecture is RAID 5 organized in a combination of 6+P and 7+P RAID sets. While this makes configuration easier, it hurts GPFS streaming performance.



Which Storage System is Best?



Which storage system is the best?

What is the best number of disks?

What is the best size of disks?

There is no unequivocal answer to these questions. The next page presents a feature comparison with a heuristic evaluation of these feature's value to HPC applications.



Which Storage System is Best for HPC?



Feature Comparison

	DS3400	DS5300 ²	DS8300	DCS9900
streaming BW	good	best	acceptable	best
IOP rate	acceptable	best	best	good
performance:capacity ¹	best	good	acceptable	good
fast RAID rebuild	no	no	no	yes
RAID 6 support	yes	yes	yes	yes
parity check on read	yes	yes	yes	yes
controller organization ²	active/passive	active/passive	active/active	active/active
disk technology	SAS, SATA	FC, SATA	FC, FATA	SAS, SATA
max number of disks	48	448	1024	1200
floor space utilization	acceptable	best	acceptable	best
remote mirroring	no	yes	yes	no
RAID N+P where $N = 2^k$ ³	yes	yes	no	yes

Footnotes:

1. The performance:capacity ratio assessment is based on the minimum number of disks *commonly* deployed in order to achieve peak streaming BW. Increasing capacity behind a controller will decrease this ratio. See the analysis on following pages.
2. Most storage controllers are based a "dual RAID controller" design in order to avoid single point of failure risks. The RAID controllers are generally associated with the RAID sets in either an active/passive or active/active organization.
3. RAID architecture is described using the expression N+P where N is the number of data disks and P is the number of parity disks in a RAID set. For optimum GPFS performance, $N = 2^k$. This category declares whether $N = 2^k$.



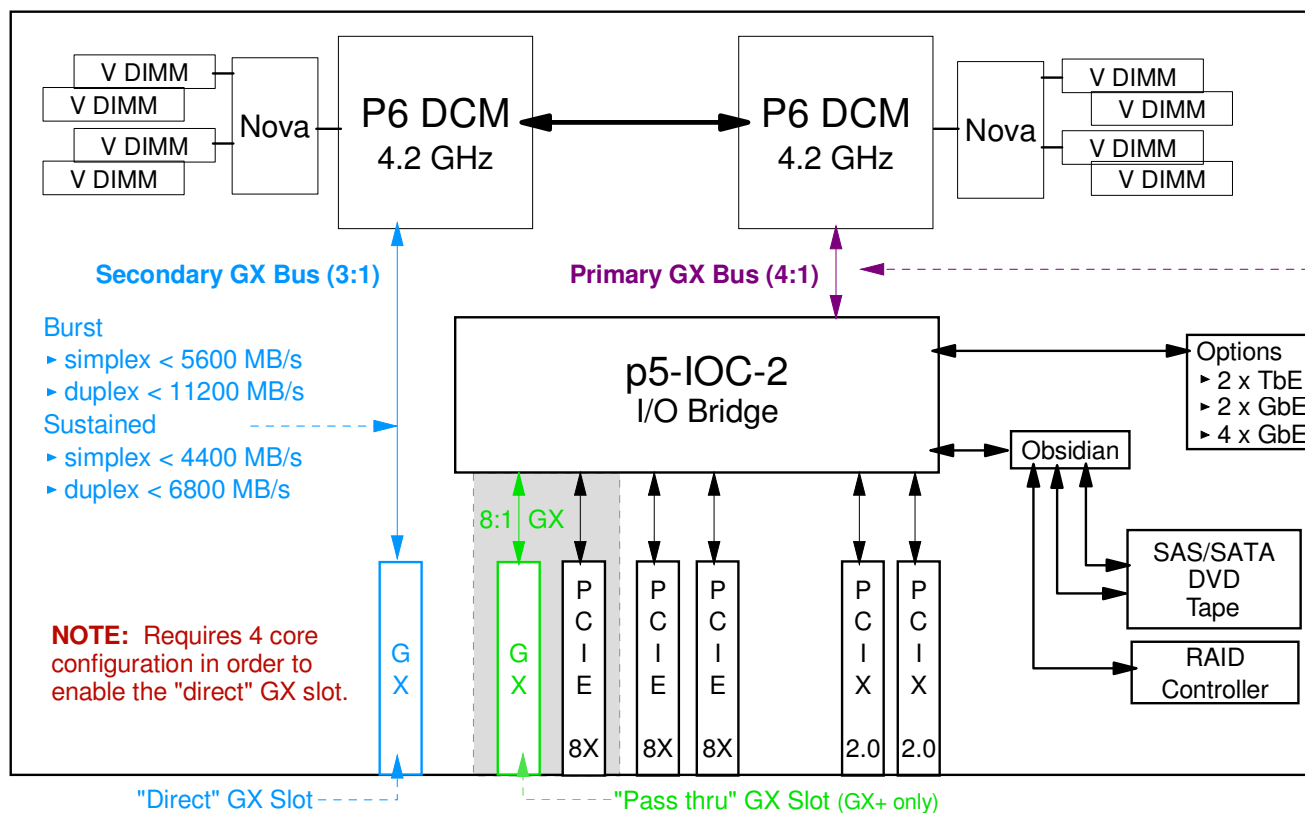
Storage Servers

There are many options for storage servers (i.e., NSD servers) with GPFS clusters. The following pages provide examples illustrating some of the more common choices.



P6-p520

System Architecture



"DIRECT" GX Slot

- IB cards are **only** supported in this slot.
- Card options:
 - dual port, IB 12xSDR @ 6:1 ratio (GX+)
 - dual port, IB 12xDDR @ 3:1 ratio (GX++)
 - RIO2 card @ 8:1 (GX+)
- 12x IB ports 1X and 4X cables
 - requires special "width changer" cable
- GX Bus width: 32 bits
- Rules of thumb:
 - Sustained simplex rates < 80% of simplex burst rate
 - Sustained duplex rates < 60% of duplex burst rate
 - single SDR "lane"
 - burst < 250 MB/s, sustained < 185 MB/s
 - single DDR "lane"
 - burst < 500 MB/s, sustained < 375 MB/s

"Pass Thru" GX Slot

- The pass thru GX slot occupies the same physical space as the 1st PCI-E slot. Therefore you can not use both of these slots.
- Supports the RIO2 card @ 8:1 (GX+). It does **not** support IB card.

Single PCI Adapter Data Rates

- PCI-E 8x:
 - Simplex: Burst < 2000 MB/s, Sustained < 1400 MB/s
 - Duplex: Burst < 4000 MB/s, Sustained < 2100 MB/s
- PCI-X 2.0
 - Burst < 2000 MB/s, Sustained < 1400 MB/s (this is not a duplex protocol)

Burst

- simplex < 4200 MB/s
 - duplex < 8400 MB/s
- Sustained
- simplex < 3400 MB/s
 - duplex < 5000 MB/s

Overview

The P6-p520 is cost effective storage server for GPFS in most pSeries clusters using Ethernet. This diagram illustrates those features most useful to its function as a storage server.

Alternative Solution

The P6-p550 can be used in place of the P6-p520. It provides the same number of I/O slots and bandwidth, but it also has more CPUs; GPFS does not need these extra CPUs, therefore the P6-p520 is recommended.



P6-p520

RIO Architecture

COMMENT:

This data rate analysis is based on the assumption that the G30 connects to Secondary GX Bus.

6:1 GX Bus on 4.2 GHz system

- Burst
- ▶ simplex < 2800 MB/s
 - ▶ duplex < 5600 MB/s
- Sustained
- ▶ simplex < 2200 MB/s
 - ▶ duplex < 3400 MB/s

2 x IB HCAs

12X IB Ports

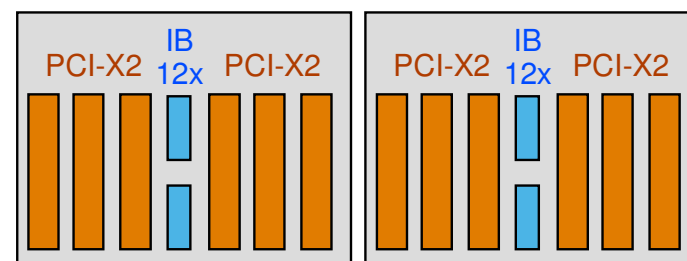
Single IB 12X link*

- Burst
- ▶ simplex < 3000 MB/s
 - ▶ duplex < 6000 MB/s
- Sustained
- ▶ simplex < 2400 MB/s
 - ▶ duplex < 3600 MB/s

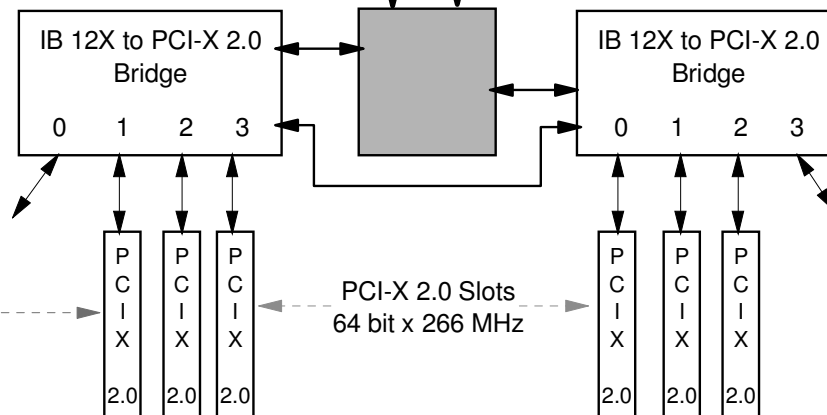
* The IB link performance is constrained by the GX bus.

Physical Dimensions

- ▶ Height: 4U
 - ▶ Width: 9.5 inches
- 2 G30s fit side by side in a 19 inch rack



7314-G30



Single PCI-X 2.0 Adapter

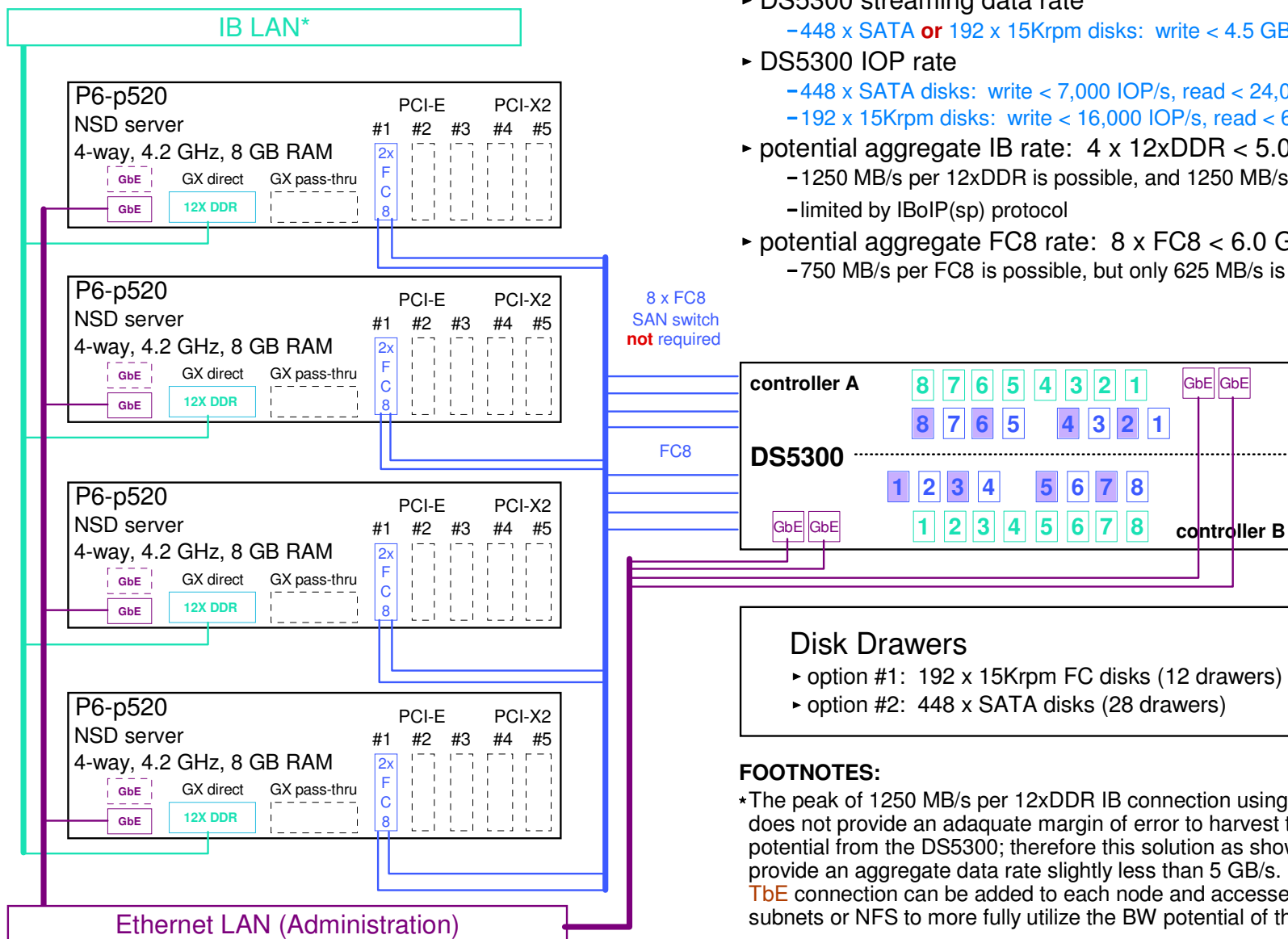
Burst < 2000 MB/s
Sustained < 1400 MB/s



P6-p520

Example Configuration

*The P6-520 offers only 12xDDR, while 4xDDR is more common, so cables supporting 12xDDR -> 4xDDR conversion are available.



NOTE: Order the P6-p520 in the 19" form factor so that they do not require special frames.

Performance Analysis

- ▶ DS5300 streaming data rate
 - 448 x SATA **or** 192 x 15Krpm disks: write < 4.5 GB/s, read < 5.0 GB/s
- ▶ DS5300 IOP rate
 - 448 x SATA disks: write < 7,000 IOP/s, read < 24,000 IOP/s
 - 192 x 15Krpm disks: write < 16,000 IOP/s, read < 64,000 IOP/s
- ▶ potential aggregate IB rate: 4 x 12xDDR < 5.0 GB/s
 - 1250 MB/s per 12xDDR is possible, and 1250 MB/s is required*
 - limited by IBolP(sp) protocol
- ▶ potential aggregate FC8 rate: 8 x FC8 < 6.0 GB/s
 - 750 MB/s per FC8 is possible, but only 625 MB/s is required

FOOTNOTES:

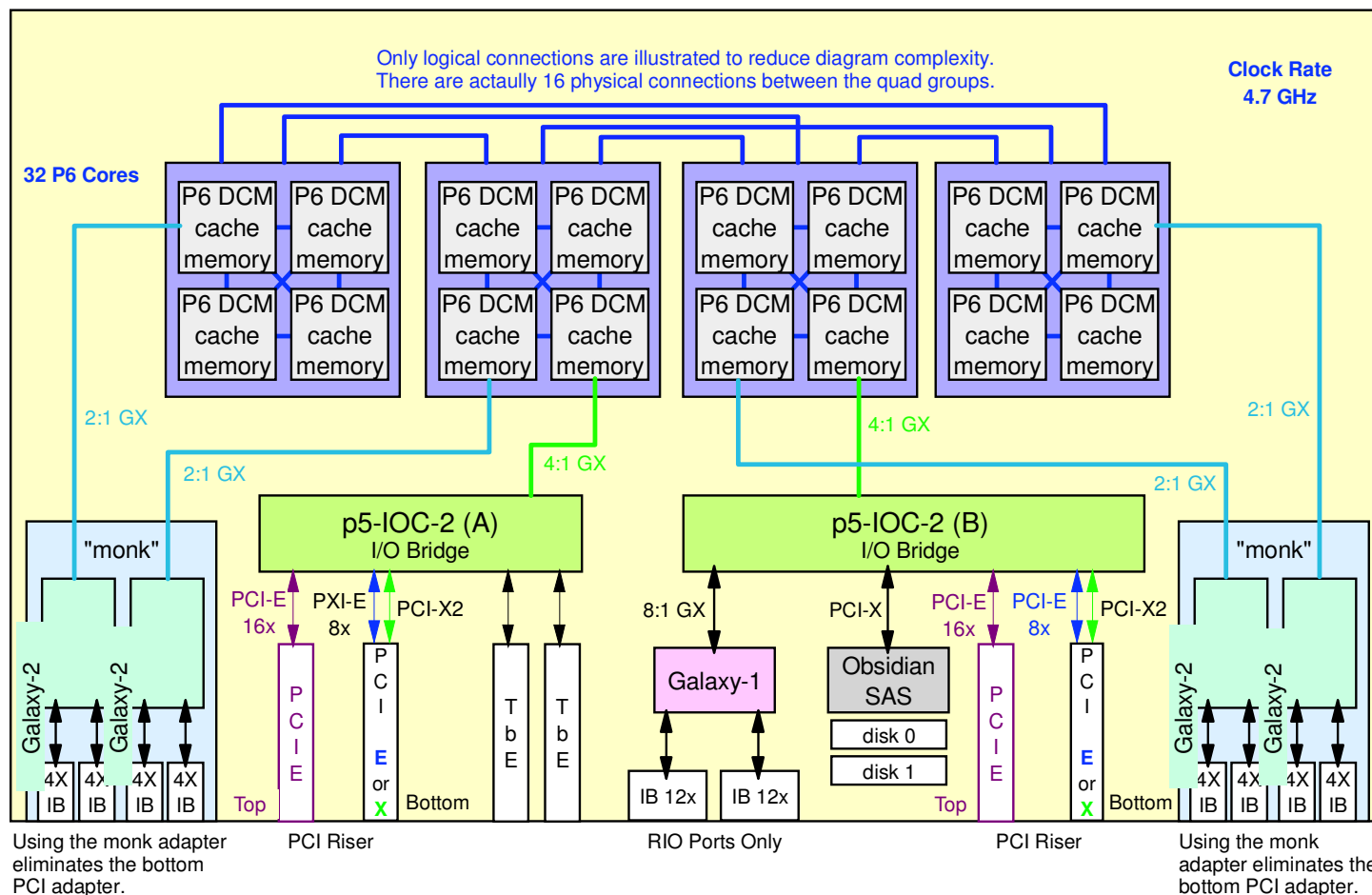
*The peak of 1250 MB/s per 12xDDR IB connection using IPoIB(sp) does not provide an adequate margin of error to harvest the 5 GB/s potential from the DS5300; therefore this solution as shown may provide an aggregate data rate slightly less than 5 GB/s. However, a **TbE** connection can be added to each node and accessed via GPFS subnets or NFS to more fully utilize the BW potential of the DS5300.



P6-p575

System Architecture

The P6-p575 is used as a storage server in HPC oriented pSeries clusters using Infiniband. This diagram illustrates those features most useful to its function as a storage server.



4:1 GX Bus Data Rates

- Burst
- ▶ simplex < 4700 MB/s
- ▶ duplex < 9400 MB/s
- Sustained
- ▶ simplex < 3700 MB/s
- ▶ duplex < 5600 MB/s

2:1 GX Bus Data Rates

- Burst
- ▶ simplex < 9400 MB/s
- ▶ duplex < 18,800 MB/s
- Sustained
- ▶ simplex < 7500 MB/s
- ▶ duplex < 11,300 MB/s

Technical Notes

- ▶ The GX bus is 32 bits wide
- ▶ Rules of thumb:
 - Sustained simplex rates ~ 80% of simplex burst rate
 - Sustained duplex rates ~ 60% of duplex burst rate

IB Performance Comments:

- ▶ 4X DDR IB port BW
 - simplex < 1500 MB/s, duplex < 2600 MB/s
- ▶ protocol limitations
 - AIX supports IPoIB(sp) which is a high performance version of IPoIB
 - simplex < 1250 MB/s, duplex < 2150 MB/s

8:1 GX Bus for the RIO ports

- Burst
- ▶ simplex < 2400 MB/s
- ▶ duplex < 4800 MB/s
- Sustained
- ▶ simplex < 1900 MB/s
- ▶ duplex < 2900 MB/s

COMMENTS:

- ▶ The 8:1 bus servicing the RIO ports severely restricts the data rate possible using 12x SDR IB.
- ▶ While this server has limited I/O connectivity, its I/O BW is outstanding. The monk IB ports in particular provides the greatest potential for high speed I/O.

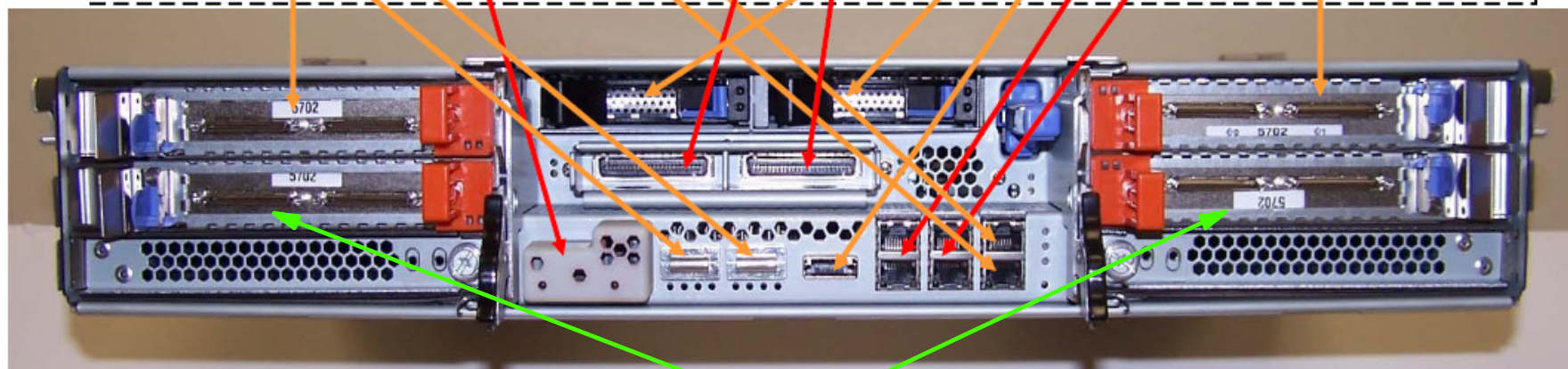
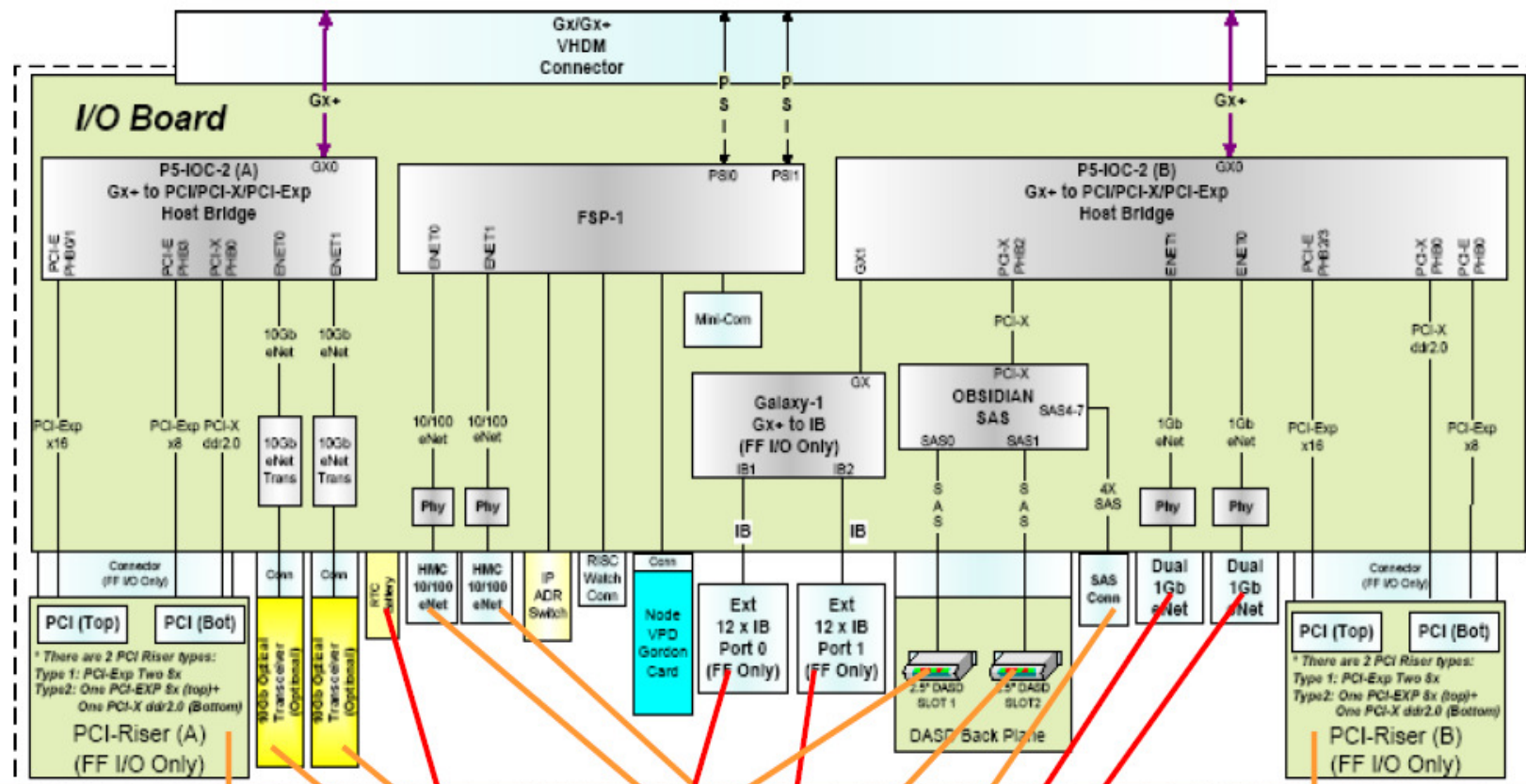


P6-p575

Physical View



"the whole enchilada"



COMMENT: The monk 4X DDR IB HCAs are not shown. If they were, the bottom PCI-E slot would not be available.



P6-p575

RIO Architecture

GX++ Bus @ 5.0 GHz

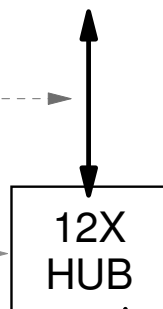
Burst

- ▶ simplex < 10.0 GB/s
- ▶ duplex < 20.0 GB/s

Sustained

- ▶ simplex < 8.0 GB/s
- ▶ duplex < 12.0 GB/s

2 x 12xDDR



COMMENTS:

- ▶ A P6-p595 provides 4 GX card slots per node. With 8 nodes per CEC, there is a max of 32 GX cards.
- ▶ Maximum Bandwidth Configuration: attach upto 16 PCI-E drawers in a dual loop configuration (as shown).
- ▶ Maximum Capacity Configuration: attach upto 32 PCI-E drawers in a single loop configuration.

Physical Dimensions

- ▶ Height: 2U
- ▶ Width: 24 inches

Single IB 12X link

Burst

- ▶ simplex < 6.0 GB/s
- ▶ duplex < 12.0 GB/s

Sustained

- ▶ simplex < 4.0 GB/s
- ▶ duplex < 6.0 GB/s

Model - 5802

Per Planar (HUB and bridge limited)

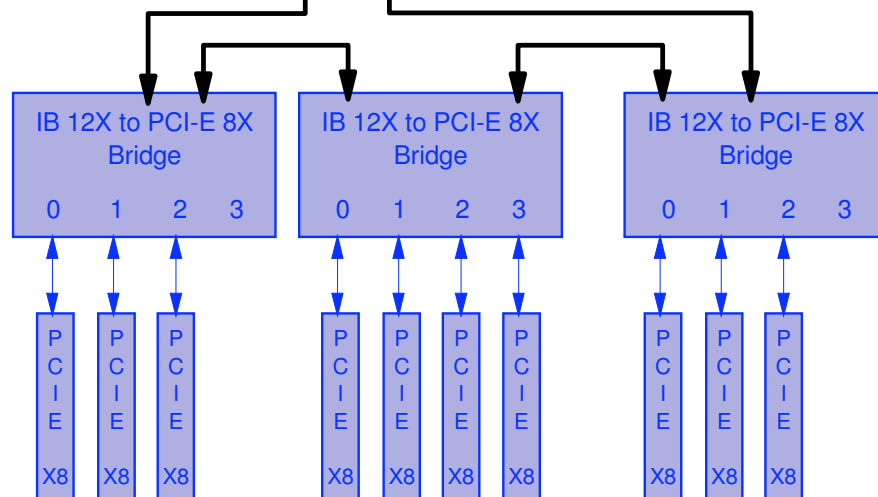
Burst

- ▶ simplex < 10.0 GB/s
- ▶ duplex < 20.0 GB/s

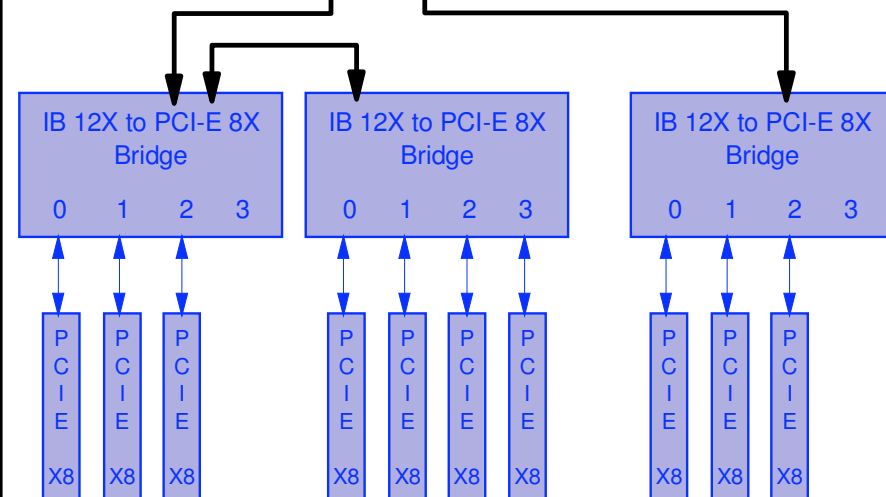
Sustained

- ▶ simplex: write < 5.0 GB/s, read < 6.0 GB/s
- ▶ duplex < 9.0 GB/s

Planar 1



Planar 2



PCI-E X8

Burst rate

- ▶ simplex < 2.0 GB/s
- ▶ duplex < 4.0 GB/s

Sustained rate

- ▶ simplex < 1.2 GB/s
- ▶ duplex < 1.8GB/s

Internal Disks:

- ▶ Support upto 26 SAS SFF Drives in 1, 2, or 4 groups.



P6-p575

Example Configuration



Server benchmark
test needed.

Performance Analysis

Peak sustained DCS9900 performance

- ▶ streaming data rate < **5.6 GB/s**
- ▶ noncached IOP rate < **40,000 IOP/s**

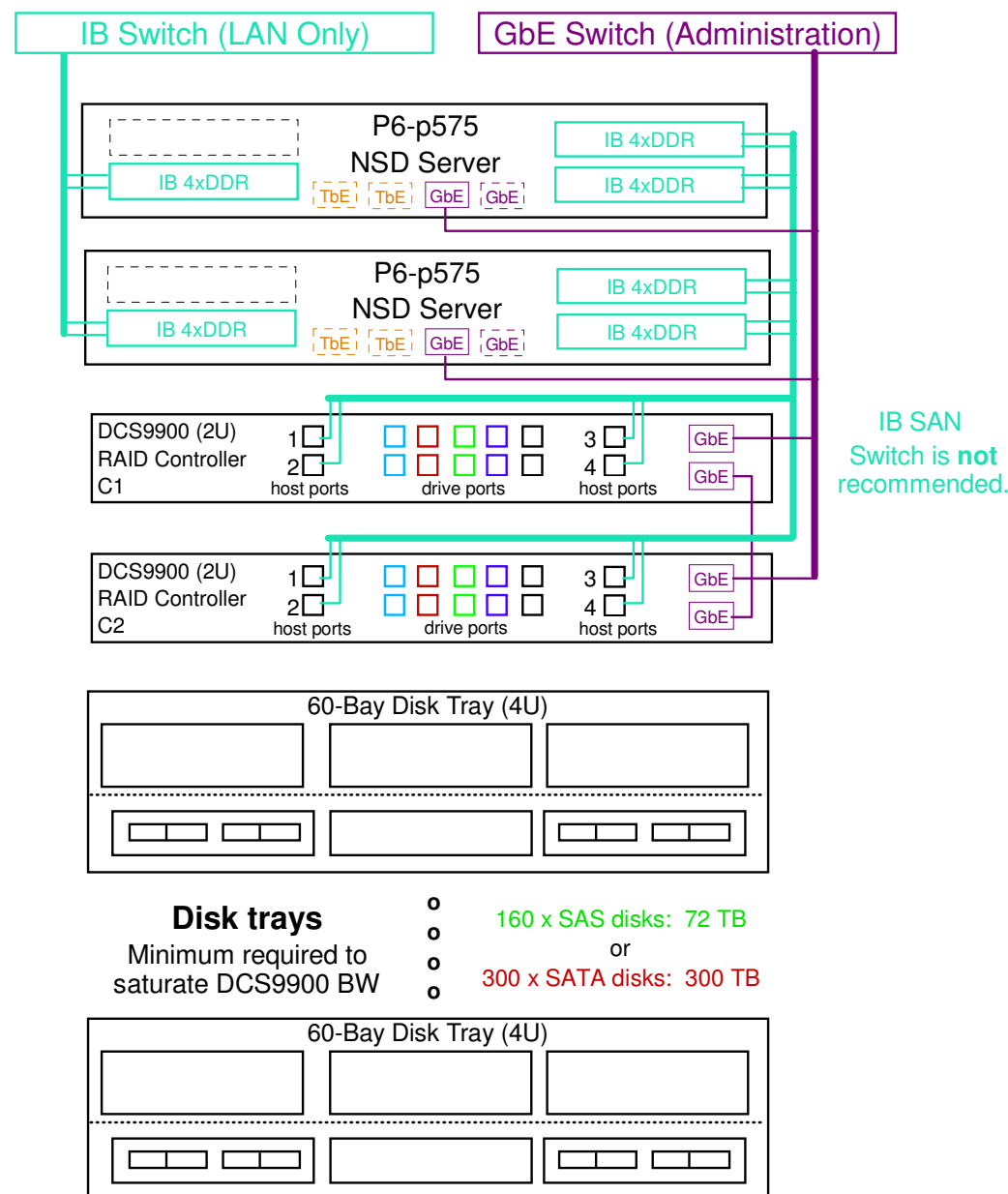
IB 4xDDR IB - LAN via IBolP(sp)

- ▶ Potential peak data rate per port < 1250 MB/s
–limited by IBolP(sp) protocol
- ▶ Required peak data rate per port < 1400 MB/s
- ▶ The peak of 1250 MB/s per IB port comes close, but is insufficient to harvest to full BW potential of the couplet. Additional IB ports are needed to fully utilize the BW potential of the couplet.

IB 4xDDR IB - SAN via SRP

- ▶ Potential peak data rate per **IB** port < 760 MB/s
- ▶ Even though the couplet host side ports are IB, they can not exceed 760 MB/s. Therefore, to harvest the full BW potential of the couplet, all 8 host side ports must be used.

COMMENT: The P6-p575 is best suited for use as an NSD server in p575 or p595 clusters with an IB based LAN. Otherwise, use the P6-p520.





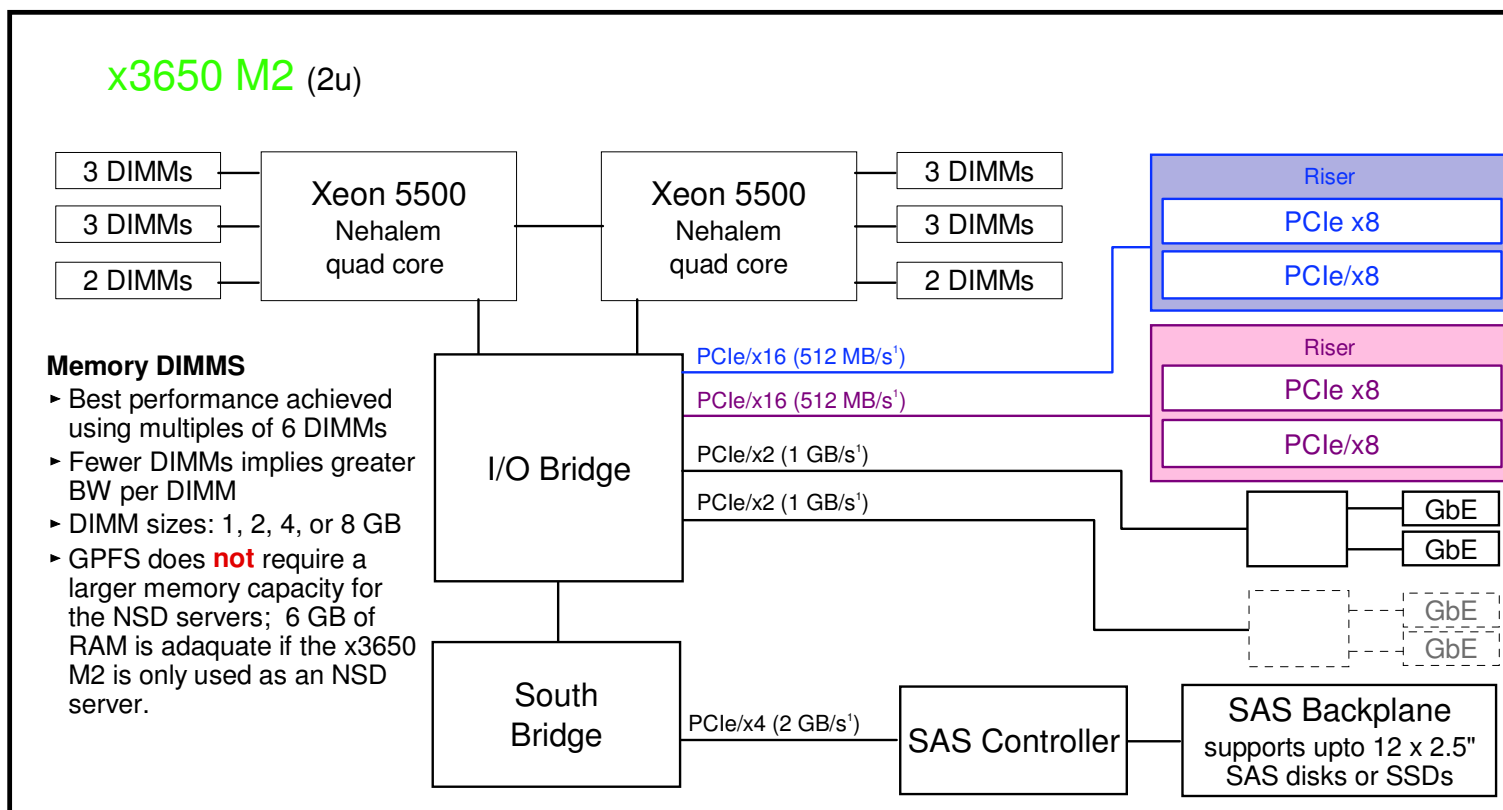
x3650 M2 System Architecture

The work horse...



The x3650 M2 is a common and cost effective storage server for GPFS in System X environments.

This diagram illustrates those features most useful to its function as a storage server.



1. Listed bus rates are theoretical *duplex* rates assuming 512 MB/s per link. Production data rates will be less.

2. Peak duplex rates for PCIe x8 adapters

Gen 1 adapters < 3.2 GB/s

Gen 2 adapters < 6.4 GB/s

3. Aggregate I/O rate over 4 x PCIe x8 adapters < 10 GB/s

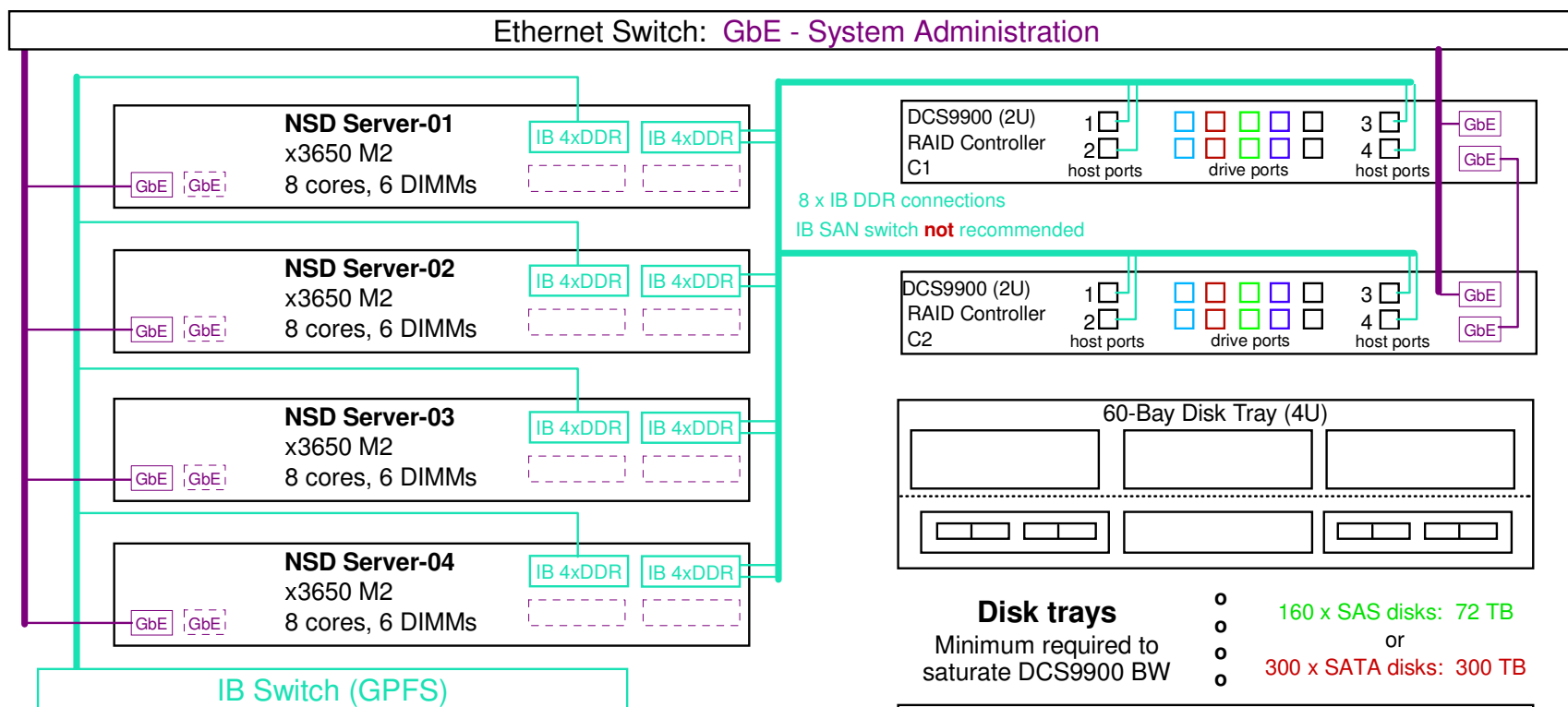
These are the data rates as they would be measured from an application perspective. Actual data rates with overhead are much greater.

* See <http://en.wikipedia.org/wiki/PCIe> for details on the PCI Express standard



x3650 M2

Example Configuration

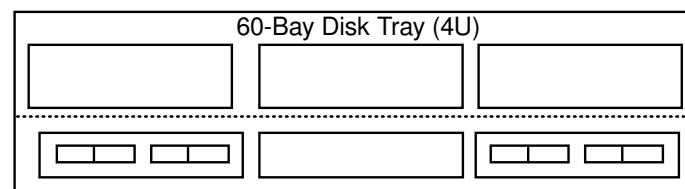


COMMENTS - DCS9900 Host Connections

- ▶ Dual port IB 4xDDR HCAs are necessary since the DCS9900 host side ports can deliver at most 760 MB/s .
- ▶ Sharing the LAN based IB switch is **not** recommended, especially if there are more than 32 NSD servers. The host ports can either be directly attached to the servers or separate IB switch can be used.
- ▶ While IB 4xDDR (RDMA) can deliver rates upto 1500 MB/s over a LAN, in practice IB 4xDDR (SRP) delivers closer to 1300 MB/s over a SAN. The peak data rate for this solution may therefore be closer to 5.2 GB/s.

Disk trays

Minimum required to saturate DCS9900 BW	160 x SAS disks: 72 TB
	or
	300 x SATA disks: 300 TB



Peak sustained DCS9900 performance

- ▶ streaming data rate < **5.6 GB/s**
 - ▶ noncached IOP rate < **40,000 IOP/s**
- LAN: 4xDDR IB HCA (RDMA)
- ▶ Potential peak data rate per HCA < 1500 MB/s
 - ▶ Required peak data rate per HCA < 1400 MB/s

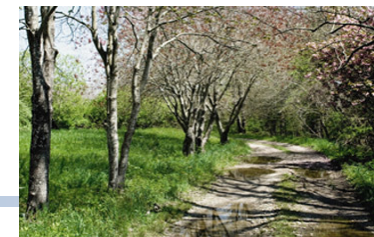
Host connections: 4xDDR IB HCA (SRP)

- ▶ Potential peak data rate per IB connection < 760 MB/s

COMMENT:
More disks (for a total of 1200) can be added to this solution but it will **not** increase performance.



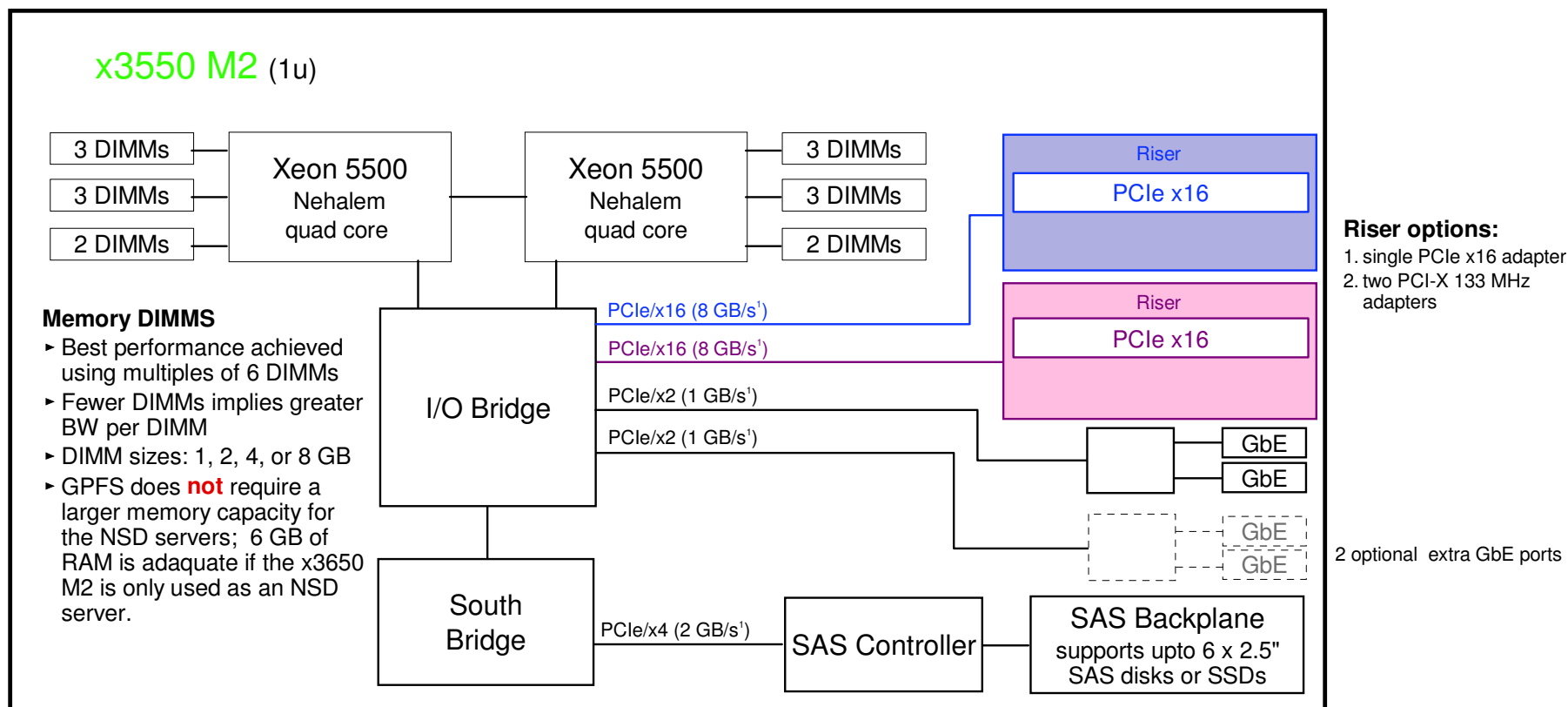
The road less traveled...



x3550 M2 System Architecture

The x3550 may be a cost effective storage server for GPFS in some cases. It's main limitation is a lack of PCIe slots.

This diagram illustrates those features most useful to its function as a storage server.



1. Listed bus rates are theoretical *duplex* rates assuming 512 MB/s per link. Production data rates will be less.

2. Peak duplex rates for PCIe x8 adapters

Gen 1 adapters < 3.2 GB/s

Gen 2 adapters < 6.4 GB/s

3. Aggregate I/O rate over 4 x PCIe x8 adapters < 10 GB/s

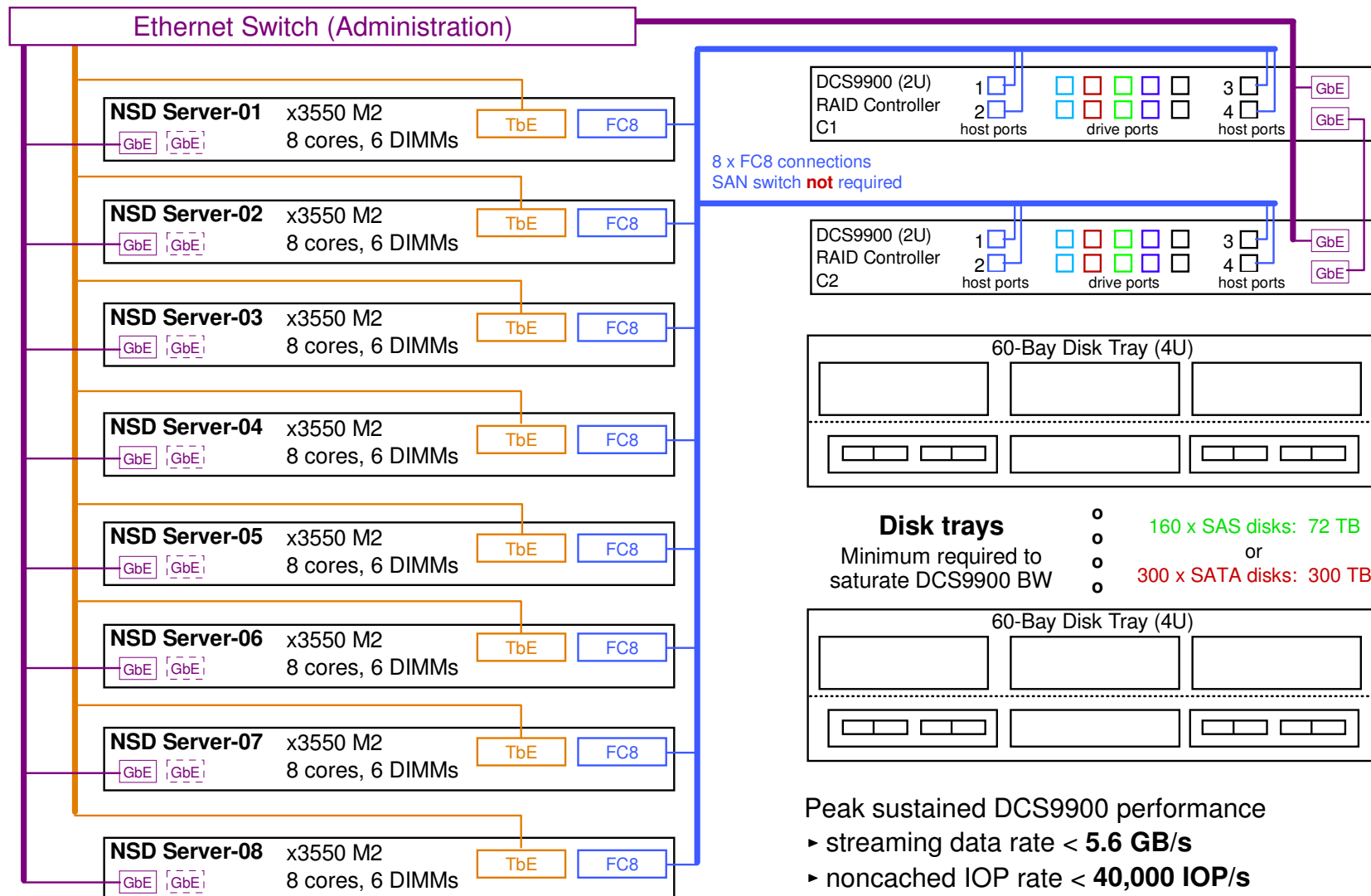
These are the data rates as they would be measured from an application perspective. Actual data rates with overhead are much greater.

* See <http://en.wikipedia.org/wiki/PCIe> for details on the PCI Express standard



x3550 M2

Example Configuration



COMMENT: Do not underestimate the I/O capability of the x3550 M2. It has the same busses and mother board as the x3650 M2, just fewer I/O ports. For example, an NSD server configuration similar to the previous x3650 M2 example could be effectively used instead of the one illustrated in this example.

COMMENT: More disks (for a total of 1200) can be added to this solution but it will **not** increase performance.

Peak sustained DCS9900 performance

- ▶ streaming data rate < **5.6 GB/s**
- ▶ noncached IOP rate < **40,000 IOP/s**

TbE (10 Gbit Ethernet Adapter)

- ▶ Potential peak data rate per **TbE** < 725 MB/s
- ▶ Required peak data rate per **TbE** < 700 MB/s

FC8 (single port 8 Gbit/s Fibre Channel)

- ▶ Potential peak data rate per **FC8** < 760 MB/s
- ▶ Required peak data rate per **FC8** < 700 MB/s

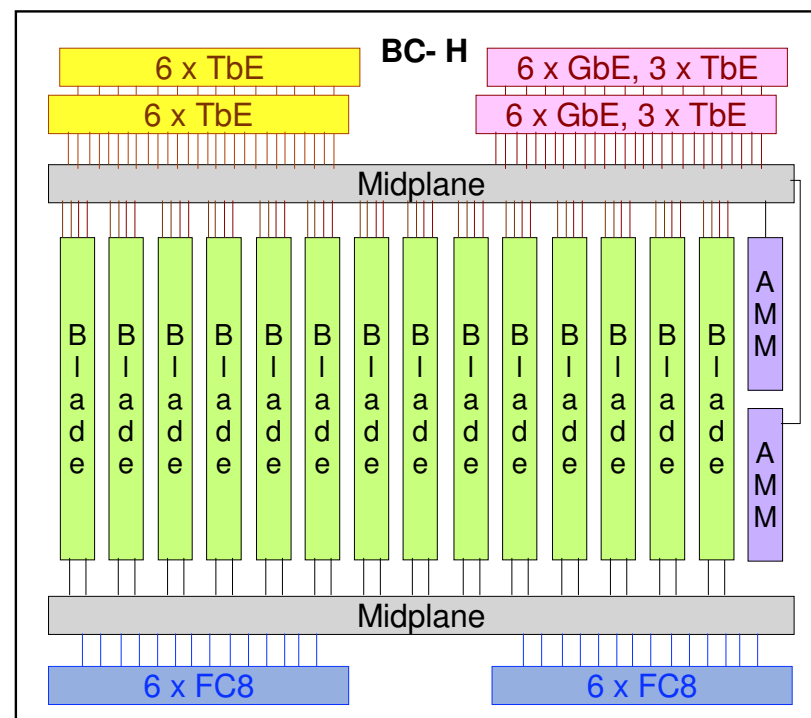


Blade Center (BC)

Features Useful for Storage Service



- Scalability
 - BC chassis supports up to 14 NSD servers and includes the power, cooling, networking and management infrastructure.
- Management efficiency
 - Nodes, cabling, switches, and management modules form integrated package
 - Management modules provide a common interface for managing all BC components
- Space efficiency
 - A single 9U chassis supports up to 14 NSD servers (plus associated infrastructure!)
- Power efficiency
 - BC power modules are as much as 50% more efficient than the smaller power supplies used in rack-mounted servers.
- Price
 - By comparison, it requires a large initial investment if only a small storage server infrastructure is needed. But the incremental costs of scaling out are small.
- Potential storage I/O bandwidth (BW)
 - Server to client BW (Ethernet)
 - TbE < 12 GB/s
 - GbE < 1 GB/s
 - Server to storage controller BW (FC)
 - **FC8 < 8 GB/s**
 - The external FC ports gate the effective BW to 8 GB/s; this means that we can only effectively use 12 of the 14 blades as NSD servers. The "extra" 2 blades can be used as spares or for other things.



COMMENT:
This is a maxed out configuration. Typically a smaller number of NSD servers is sufficient.

FC8 < 750 MB/s

I/O Ports

- **TbE module:**
 - in: 14 x TbE
 - out: 6 x TbE
- **Mixed Ethernet module:**
 - in: 14 x GbE
 - out: 6 x GbE, 3 x TbE
- **FC module:**
 - in: 14 x FC4
 - out: 6 x FC8

Blade	HS21 or HS21XM
Cores	► 4 cores
RAM	► RAM: 4 to 8 GB is adequate up to 32 GB is possible
I/O Ports	Sustained I/O Rates
PCI-E 2 x TbE	PCI-E (8x) < 1400 MB/s TbE < 725 MB/s
GbE	GbE < 80 MB/s
PCI-X 2 x FC4	PCI-X < 700 MB/s FC4 < 380 MB/s

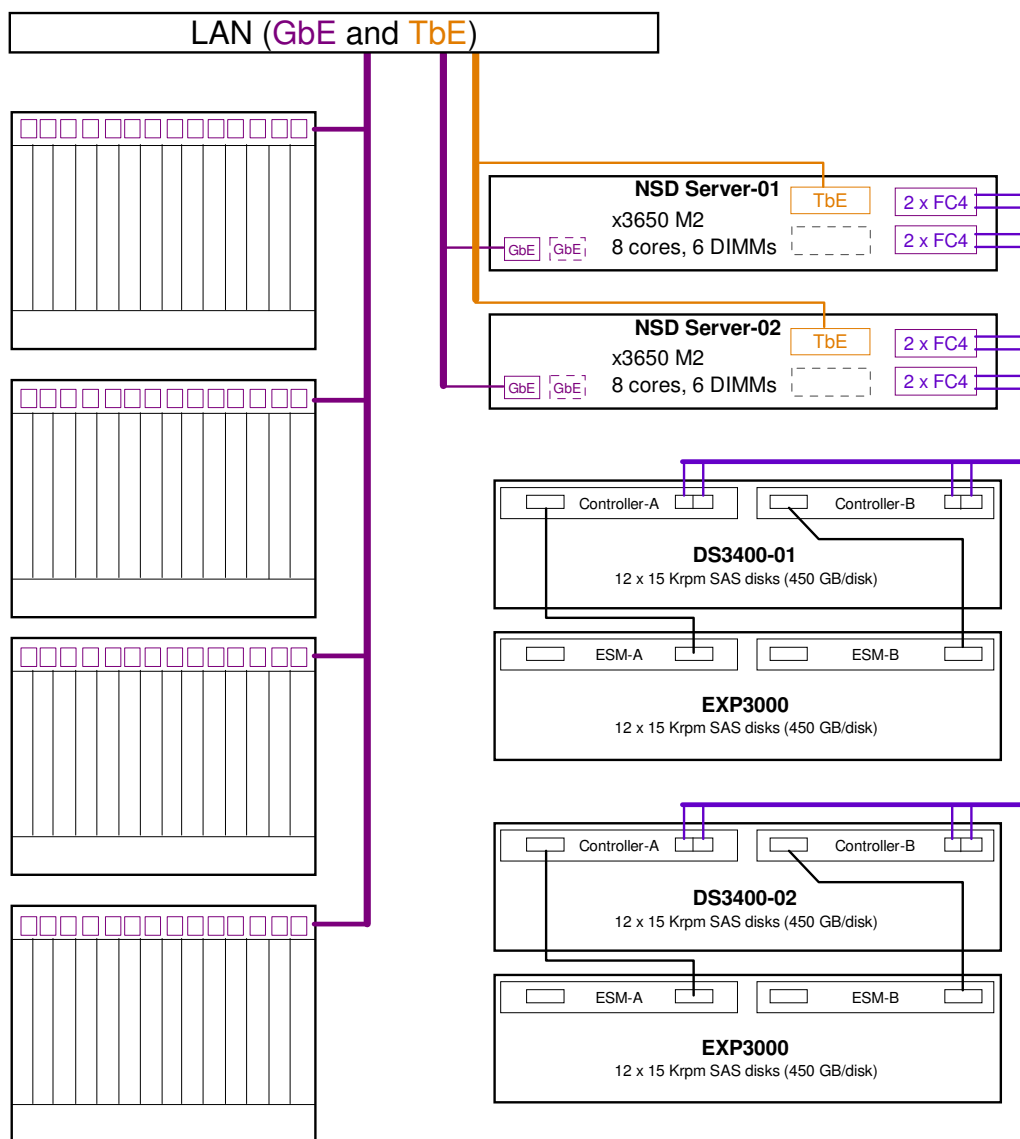


BladeCenter Configuration

Using External Nodes as Storage Servers



The recommended best practice for using GPFS with blades is to use external nodes as the NSD servers.



ANALYSIS

Storage

- ▶ 2 x DS3400
 - disk: 15Krpm SAS
 - 48 disks
 - 4+P RAID 5
 - 8 arrays + 8 hot spares
 - usable capacity < 14 TB

Blades (56)

- ▶ GbE to each blade
 - up to 80 MB/s per blade
- ▶ Aggregate BW
 - write < 1300 MB/s
 - read < 1450 MB/s
- ▶ Average BW
 - write < 20 MB/s per blade
 - read < 25 MB/s per blade

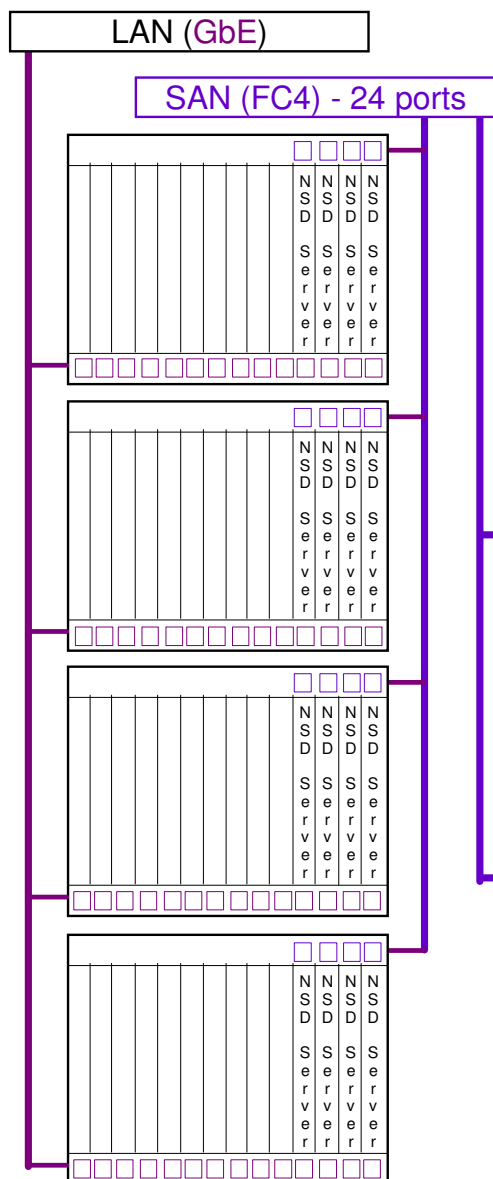


BladeCenter Configuration

Using Blades as NSD Servers



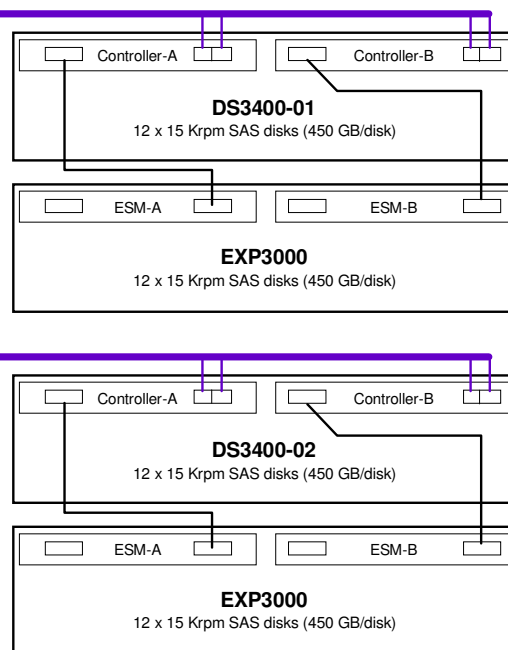
While not as effective as external servers, blades can be used as NSD servers.



ANALYSIS

Storage

- ▶ 2 x DS3400
 - disk: 15Krpm SAS
 - 48 disks
 - 4+P RAID 5
 - 8 arrays + 8 hot spares
 - usable capacity < 14 TB



Blades (56)

- ▶ GbE to each blade
 - up to 80 MB/s per blade
- ▶ Aggregate BW
 - write < 1300 MB/s
 - read < 1450 MB/s
- ▶ Average BW
 - write < 20 MB/s per blade
 - read < 25 MB/s per blade

COMMENT:

- ▶ Given a GbE of LAN, it is necessary to use 20 nodes as NSD servers.
- ▶ Requires SAN switch and 1 FC4 per NSD server.
- ▶ NSD servers can also be used to run applications as GPFS clients.
- ▶ Blades have less utility as storage servers due to more limited I/O capabilities.

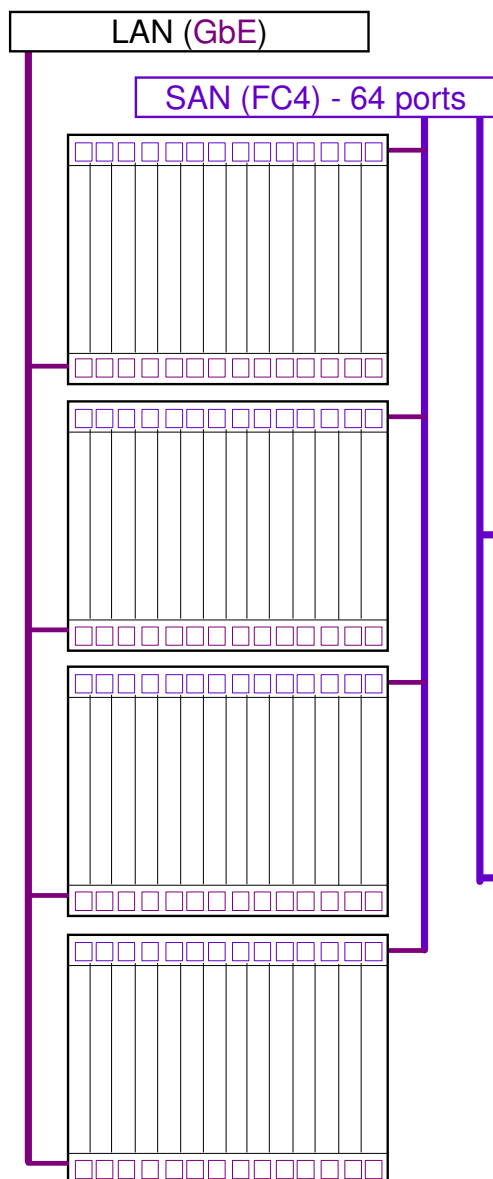


Blade Configuration

Using Blades as NSD Servers



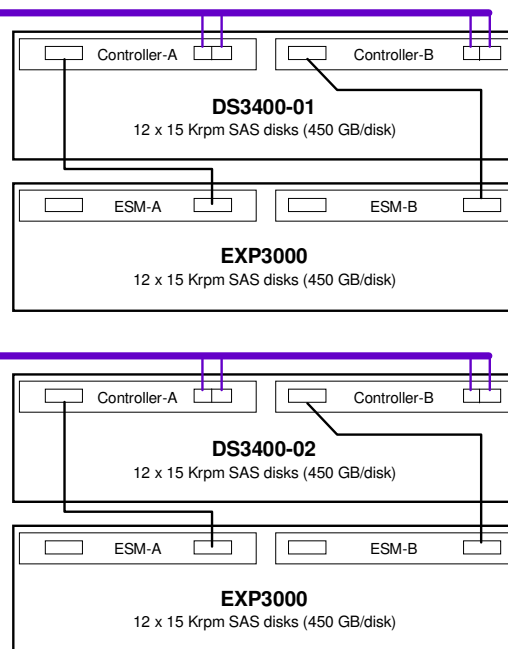
Blades can use GPFS as a SAN file system, but since blade clusters tend to be large, a larger SAN and special SAN tuning is necessary.



ANALYSIS

Storage

- ▶ 2 x DS3400
 - disk: 15Krpm SAS
 - 48 disks
 - 4+P RAID 5
 - 8 arrays + 8 hot spares
 - usable capacity < 14 TB



Blades (56)

- ▶ FC4 per blade
 - up to 380 MB/s per blade
- ▶ Aggregate BW
 - write < 1300 MB/s
 - read < 1450 MB/s
- ▶ Average BW
 - write < 20 MB/s per blade
 - read < 25 MB/s per blade

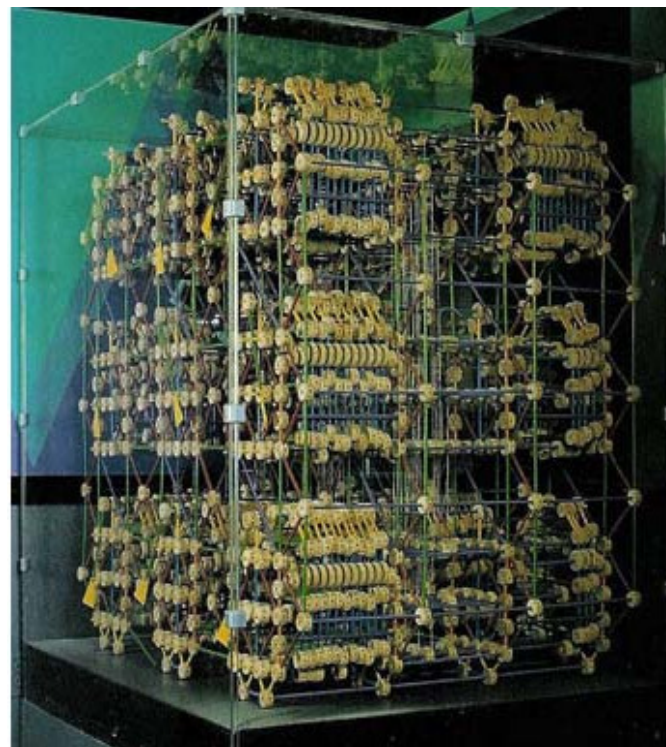
COMMENT:

- ▶ Requires larger SAN (1 FC4 per blade) along with a SAN switch.
- ▶ Set queue depth to 1 or 2.
- ▶ There are no hard rules saying that GPFS can not be used for a large SAN, nor are there rules regarding the size of a GPFS SAN, but generally SANs spanning more than 32 nodes are less common for GPFS.

10. GPFS Configurations

This section contains numerous examples of actual and proposed GPFS configurations illustrating the versatility of GPFS.

They show both ordinary and unusual configurations.



Tinkertoy Computer
On display at Museum of Science, Boston



Disclaimers

- The configurations shown in this section are only examples illustrating and suggesting GPFS possible configurations. In some cases, they merely illustrate how systems *have been* configured, not necessarily how they *should be* configured.
- These slides are not intended to be "wiring diagrams"; rather, they are to illustrate basic concepts when integrating various components into an overall solution.
- Unless stated otherwise, "feeds and speeds" are based on *realistic* upper bound estimates as measured by the application, but under ideal benchmarking conditions.
Performance will vary "according to actual driving conditions".





Balance

The I/O Subsystem Design Goal

Ideally, an I/O subsystem should be balanced. There is no point in making one component of an I/O subsystem fast while another is slow. Moreover, overtaxing some components of the I/O subsystem (e.g., HBAs) may disproportionately degrade performance.

However, this goal can not always be perfectly achieved. A common imbalance is when capacity is more important than bandwidth; then the aggregate bandwidth based on the number of disks may exceed the aggregate bandwidth supported by the electronics of the controllers and/or the number HBAs and storage servers.

"Performance is inversely proportional to capacity."
-- Todd Virnoche





GPFS Building Blocks



A convenient design strategy for GPFS solutions is to define a "storage building block", which is the "smallest" increment of storage and servers by which a storage system can grow.

Therefore, a storage solution consists of 1 or more storage building blocks. This allows customers to conveniently expand their storage solution in increments of storage building blocks (*i.e.*, "build as you grow" strategy)

This solution is made feasible since GPFS scales linearly in the number of disks, storage controllers, NSD servers, GPFS clients, and so forth.



But the Building Blocks Are Getting Larger

FC and SAS disks: 450 GB/disk

SATA Disks: 2 TB/disk

Storage Controllers: 0.5 PB to 1.0 PB

Storage Servers: several GB/s

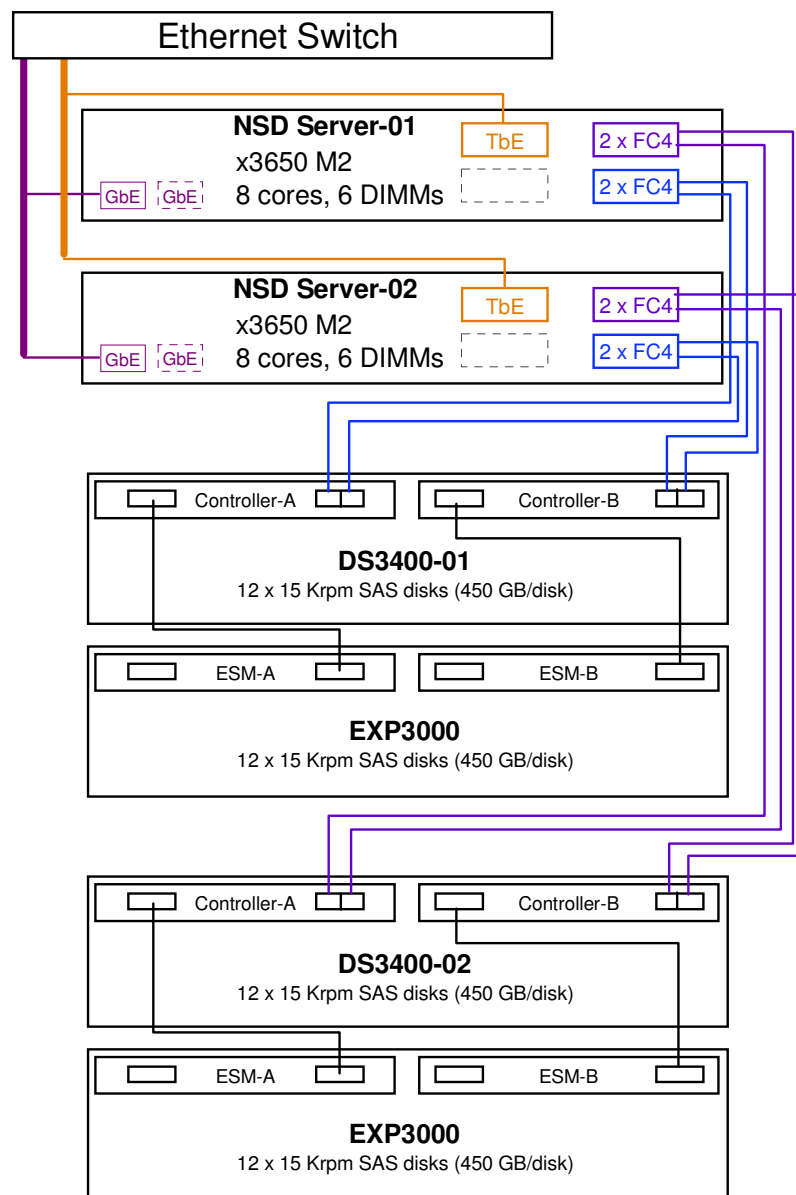
This presents a challenge for smaller storage systems.





Building Block #1A

Performance Optimized

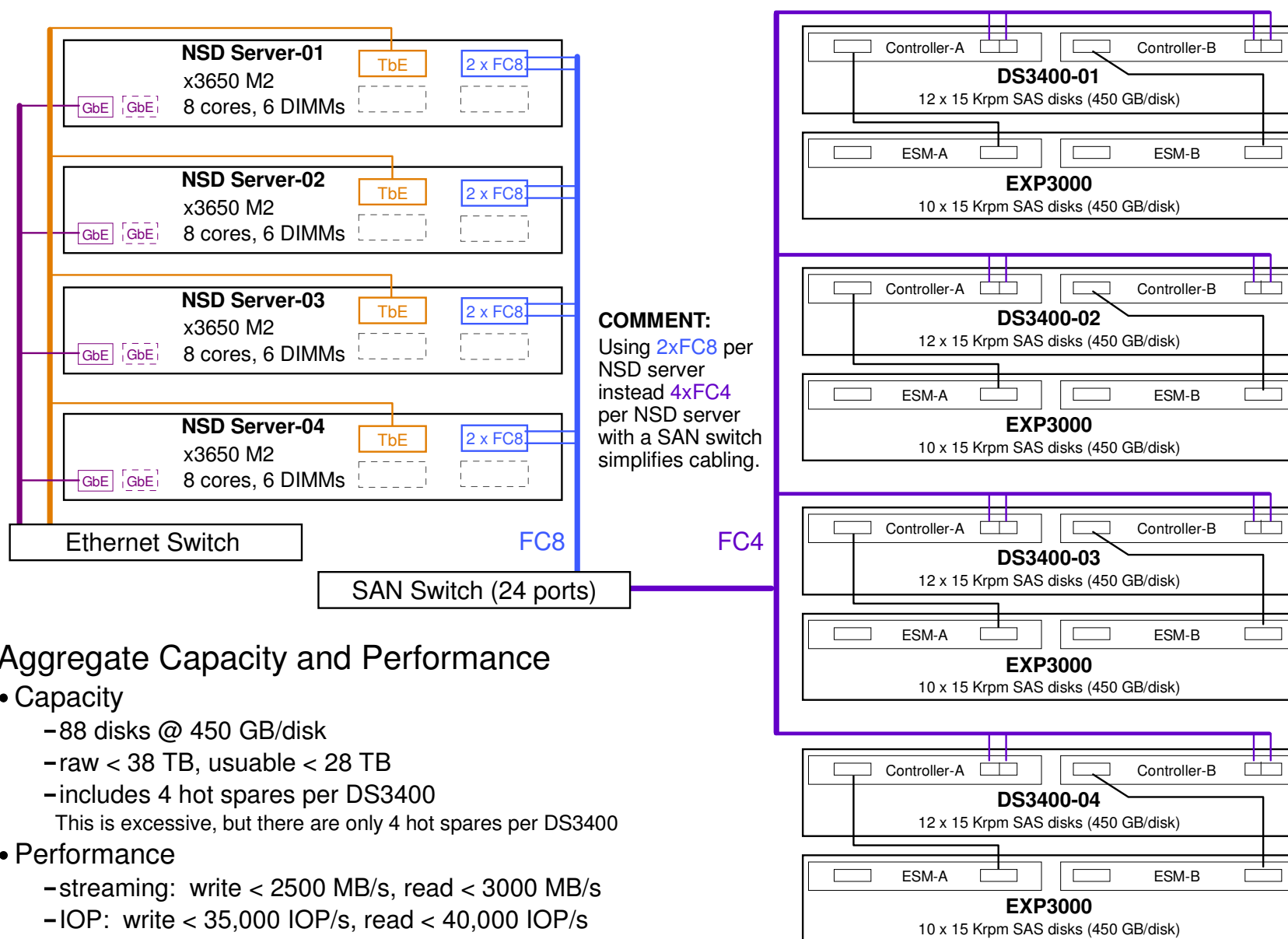


- ▶ **NSD Server: x3650-M2**
 - 8 Cores, 6 GB RAM
 - 2 dual-port 4 Gb/s FC HBAs (2xFC4)
 - at most 760 MB/s per adapter
 - Single 10 GbE (TbE) adapter per node
 - at most 750 MB/s per adapter (Myricom adapter)
- ▶ **Disk Controller: DS3400 with EXP3000**
 - 12 disks per DS3400 plus 12 disks per EXP3000
 - SAS disk, 450 GB/disk @ 15Krpm
 - LUNs: 4+P RAID5 sets
- ▶ **Aggregate Capacity and Performance**
 - Capacity
 - 48 disks @ 450 GB/disk
 - raw = 21 TB, usable = 14.4 TB
 - includes 8 hot spares
 - This is excessive, but there are only 4 hot spares per DS3400
 - Performance
 - streaming: write < 1300 MB/s, read < 1600 MB/s
 - IOP: write < 18,000 IOP/s, read < 22,000 IOP/s
- ▶ **Alternative: Capacity Optimized**
 - Use 4 drawers of 1 TB SATA disk per DS3400
 - Capacity
 - 84 disks @ 1 TB/disk in 8+2P RAID 6 configuration
 - raw = 84 TB, usable = 64 TB
 - includes 4 hot spares
 - Performance: TBD (perhaps 1200 MB/s?)
 - **COMMENT:**
 - Usable capacity could be increased to 72 TB using 4+P RAID5 arrays, but this is not a best practice.



Building Block #1A

2 Building Blocks



Aggregate Capacity and Performance

• Capacity

- 88 disks @ 450 GB/disk
- raw < 38 TB, usable < 28 TB
- includes 4 hot spares per DS3400
- This is excessive, but there are only 4 hot spares per DS3400

• Performance

- streaming: write < 2500 MB/s, read < 3000 MB/s
- IOP: write < 35,000 IOP/s, read < 40,000 IOP/s

WARNING: Scaling beyond 2 building blocks (i.e., 4 x DS3400) is **not** recommended when performance is critical because RAID rebuilds over multiple DS3400s significantly impede performance. If scaling beyond this is required, then deploy multiple GPFS file systems or storage pools to limit the impact of RAID rebuilds.



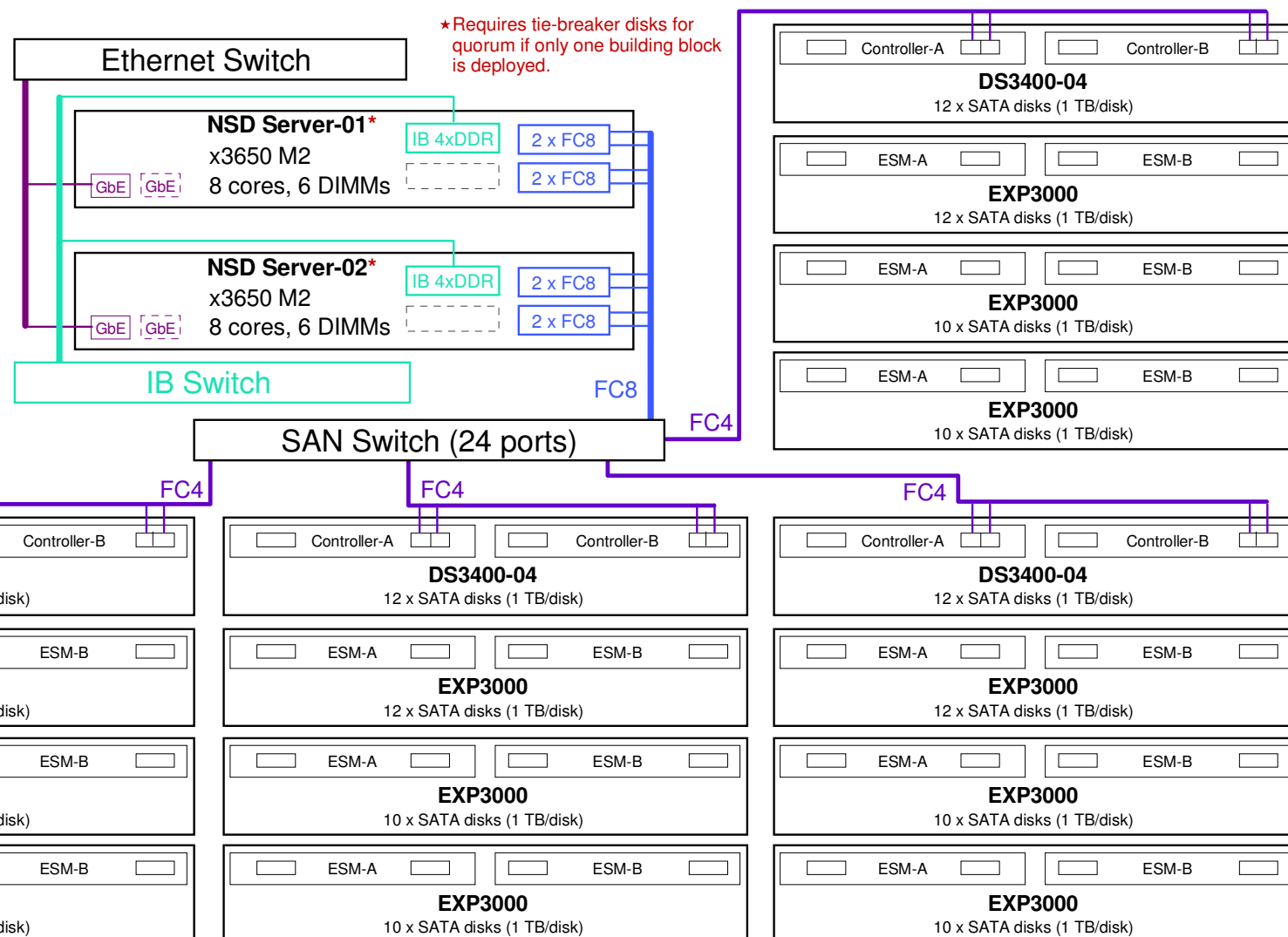
Building Block #1B

Maximizing Capacity and Harvesting Unused Server Bandwidth

General Idea:

x3650 M2 bandwidth is not fully utilized in building block #1a. This larger building block supports 2X more DS3400s in order to efficiently use the server bandwidth.

WARNING: Scaling beyond 1 building block (i.e., 4 x DS3400) is **not** recommended. See WARNING on previous page.



NSD Server: x3650-M2

- 8 Cores, 6 GB RAM
- 2 dual-port 8 Gb/s FC HBAs (2xFC8)
 - at most 1500 MB/s per adapter
- IB HCA (4xDDR2)
 - at most 1500 MB/s per HCA

Capacity Optimized

- Use 4 drawers of 1 TB SATA disk per DS3400
- Capacity per DS3400
 - 42 disks @ 1 TB/disk in 8+2P RAID 6 configuration
 - raw = 42 TB, usable = 32 TB
 - includes 2 hot spares

- Aggregate Capacity
 - raw = 168 TB, usable = 128 TB
 - includes 8 hot spares
- Aggregate Performance
 - streaming rate < 3 GB/s



Building Block #1A vs. #1B

Performance vs. Capacity

15Krpm FC disk @ 450 GB/disk

- Capacity
 - raw < 38 TB
 - usable < 28 TB
- Performance
 - streaming rate
 - write < 2500 MB/s
 - read < 3000 MB/s
 - IOP rate
 - write < 35,000 IOP/s
 - read < 40,000 IOP/s
- Performance to Usable Capacity Ratio
 - streaming rate
 - write < 89 MB/s / TB
 - read < 107 MB/s / TB
 - IOP rate
 - write < 1250 IOP/s / TB
 - read < 1430 IOP/s / TB
- Floor Space⁺
 - Racks (42u x 19"): 1
 - Usable Capacity per rack: 29 TB/rack

SATA @ 1 TB/disk

- Capacity
 - raw < 168 TB
 - usable < 128 TB
- Performance
 - streaming rate
 - write < 2500 MB/s
 - read < 3000 MB/s
 - IOP rate^{*}
 - write < 15,000 IOP/s
 - read < 20,000 IOP/s
- Performance to Usable Capacity Ratio
 - streaming rate
 - write < 19 MB/s / TB
 - read < 23 MB/s / TB
 - IOP rate
 - write < 117 IOP/s / TB
 - read < 156 IOP/s / TB
- Floor Space⁺
 - Racks (42u x 19"): 1
 - Usable Capacity per rack: 128 TB/rack

FOOTNOTES:

★ SATA IOP rates need validation testing (n.b., they are a SWAG ;->)

† This ratio is misleading in this case since a rack is not fully utilized for this solution.

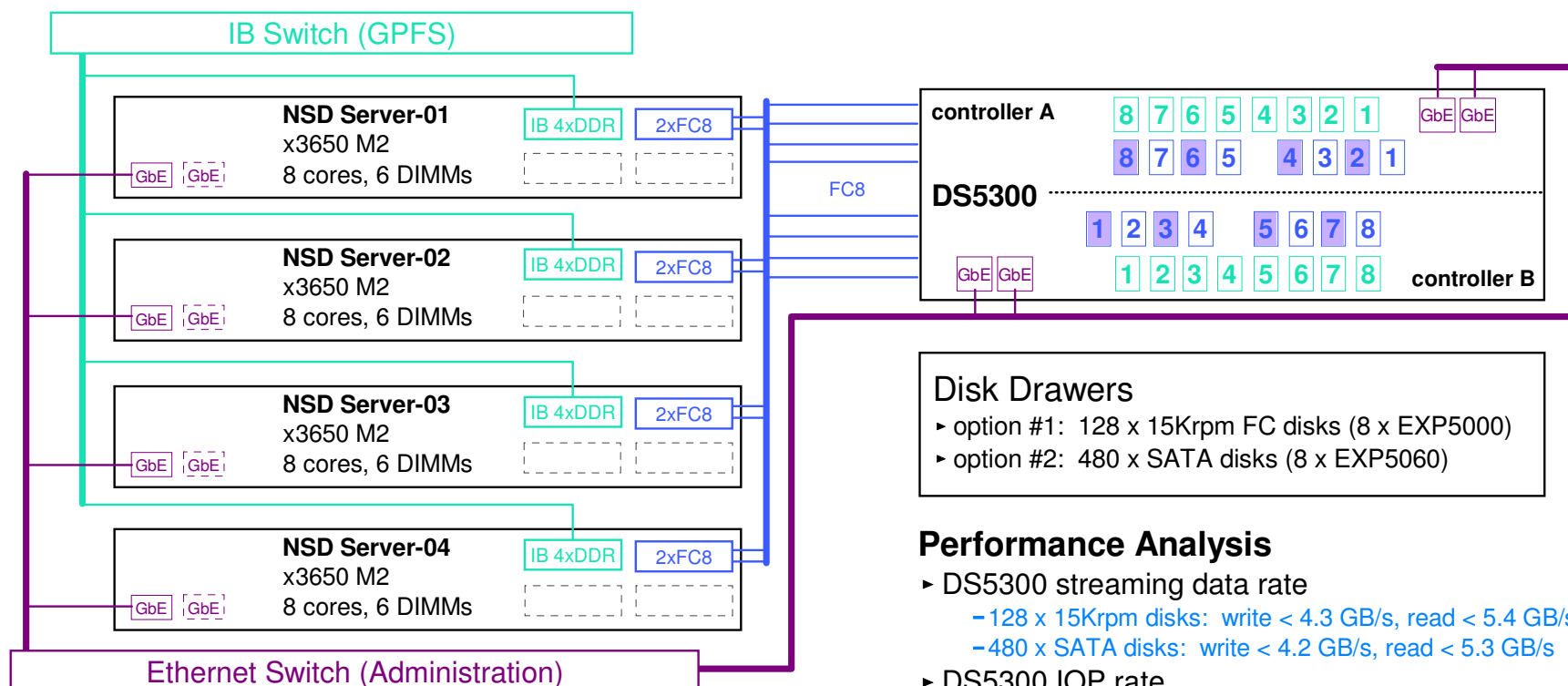


Building Block #2A

Performance vs. Capacity



The plain vanilla
Linux configuration



FOOTNOTE:

★ The 15Krpm IOP rates assume of good locality.
Assuming poor locality, these rates could be:
write < 9,000 IOP/s, read < 15,000 IOP/s.

Performance Analysis

- DS5300 streaming data rate
 - 128 x 15Krpm disks: write < 4.3 GB/s, read < 5.4 GB/s
 - 480 x SATA disks: write < 4.2 GB/s, read < 5.3 GB/s
- DS5300 IOP rate
 - 128 x 15Krpm disks: write < 26,000 IOPs, read < 35,000 IOP/s*
 - 480 x SATA disks: write < 7,000 IOP/s, read < 24,000 IOP/s
- potential aggregate IB rate: 4 x 12xDDR < 5.0 GB/s
 - 1500 MB/s per 12xDDR is possible, and 1350 MB/s is required
- potential aggregate FC8 rate: 8 x FC8 < 6.0 GB/s
 - 750 MB/s per FC8 is possible, but only 675 MB/s is required

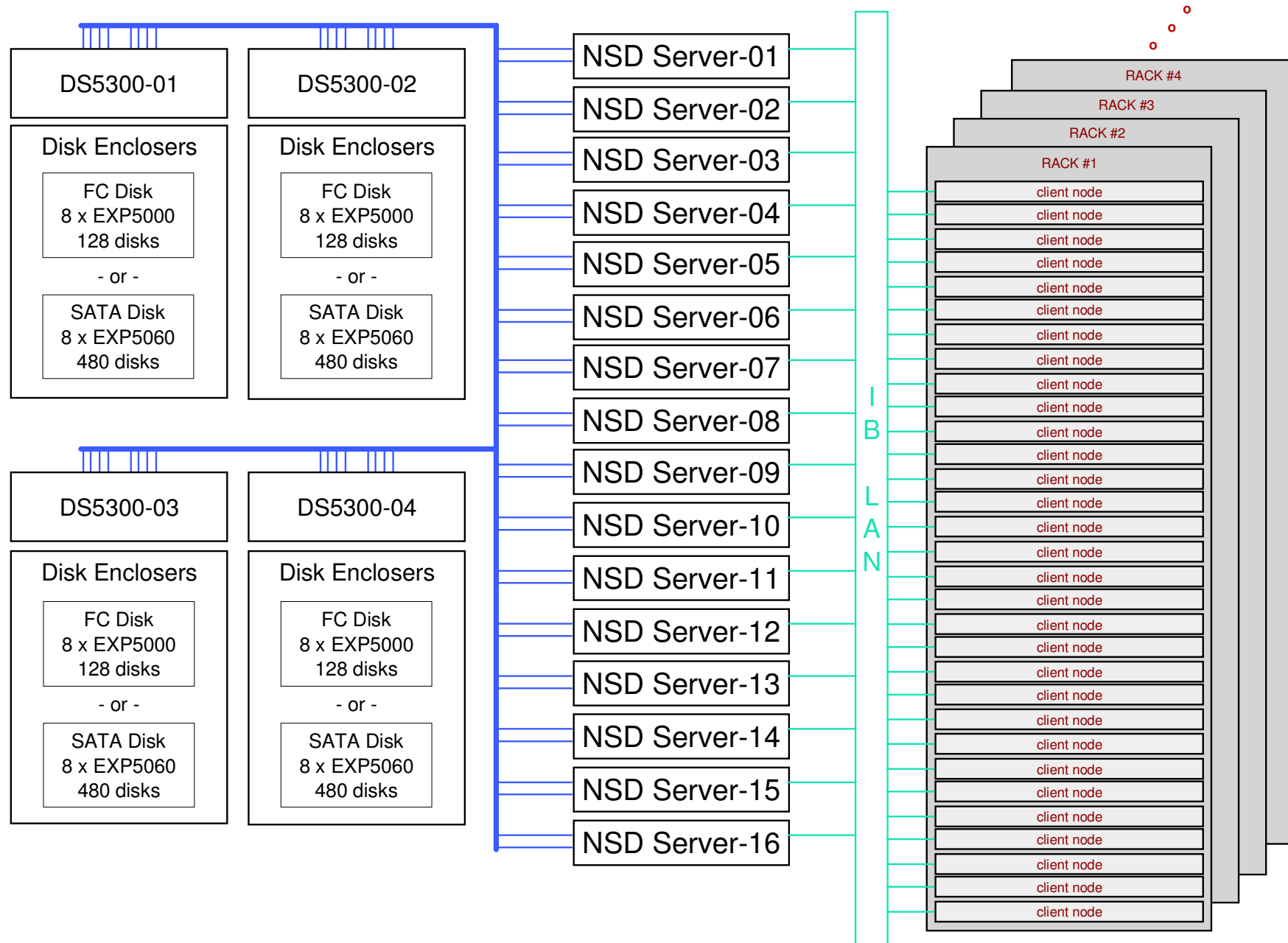
Capacity Analysis

- 15Krpm FC Disk
 - 128 disks @ 450 GB/disk
 - 24 x 4+P RAID 5 arrays + 8 hot spares
 - raw capacity < 56 TB, usable capacity < 42 TB
- SATA disk
 - 480 disks @ 2 TB/disk
 - 48 x 8+2P RAID 6 arrays
 - raw capacity < 960 TB, usable capacity < 768 TB



Building Block #2A or #2B

4 Building Blocks - Performance vs. Capacity





Building Block #2A vs. #2B

4 Building Blocks - Performance vs. Capacity

15Krpm FC disk @ 450 GB/disk

- Capacity
 - raw < 224 TB
 - usable < 168 TB
- Performance
 - streaming rate
 - write < 16 GB/s
 - read < 20 GB/s
 - IOP rate
 - write < 104,000 IOP/s
 - read < 140,000 IOP/s
- Performance to Usable Capacity Ratio
 - streaming rate
 - write < 97 MB/s / TB
 - read < 122 MB/s / TB
 - IOP rate
 - write < 620 IOP/s / TB
 - read < 830 IOP/s / TB
- Racks
 - Storage Racks (42u x 19"): 4
 - Server Racks (42u x 19"): 5

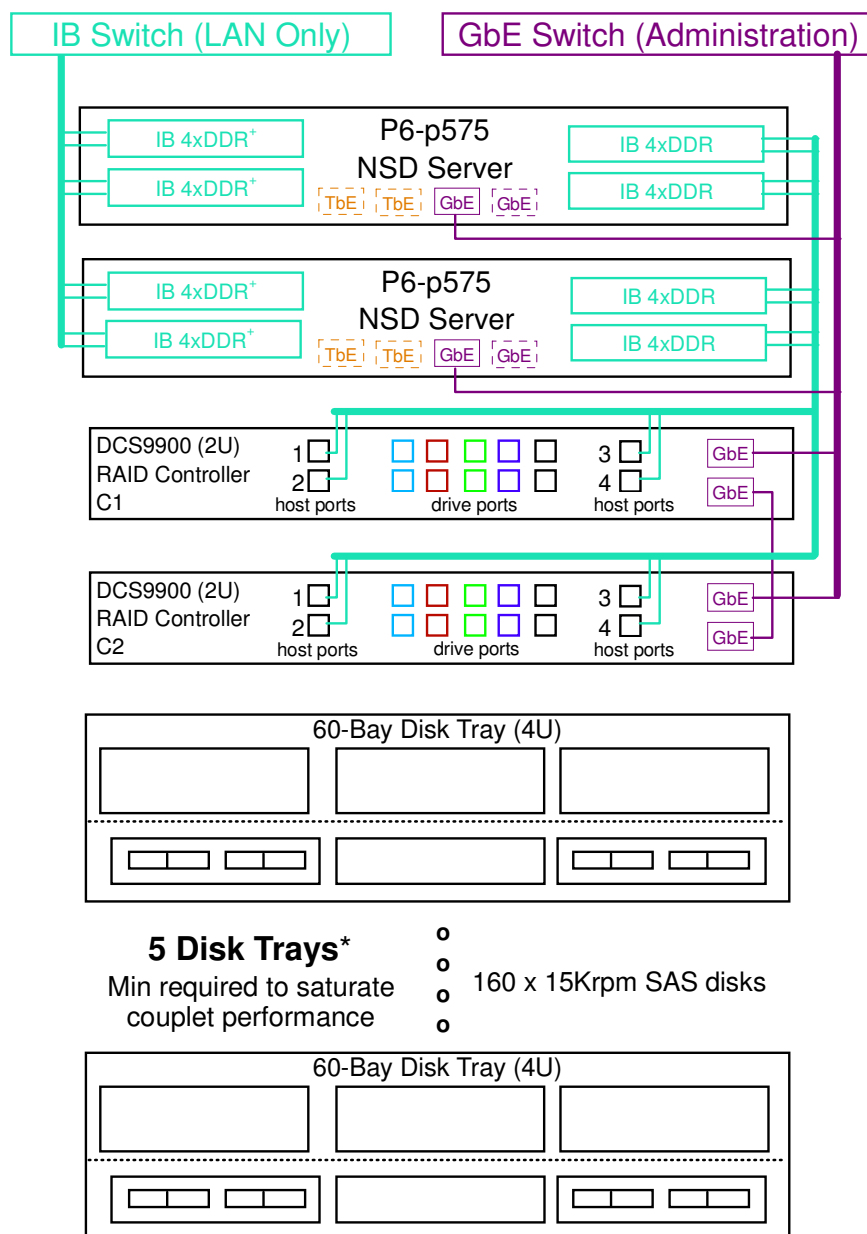
SATA @ 1 TB/disk

- Capacity
 - raw < 3840 TB
 - usable < 3072 TB
- Performance
 - streaming rate
 - write < 16 GB/s
 - read < 20 GB/s
 - IOP rate
 - write < 28,000 IOP/s
 - read < 96,000 IOP/s
- Performance to Usable Capacity Ratio
 - streaming rate
 - write < 5.3 MB/s / TB
 - read < 6.7 MB/s / TB
 - IOP rate
 - write < 9.1 IOP/s / TB
 - read < 31 IOP/s / TB
- Racks
 - Storage Racks (42u x 19"): 5
 - Server Racks (42u x 19"): 5



Building Block #3A

Performance Optimized



Performance Analysis

DCS9900 Performance

- ▶ Streaming data rate
 - write < 5.7 GB/s
 - read < 4.4 GB/s
- ▶ Noncached IOP rate (4K transactions)
 - write < 40,000 IOP/s
 - read < 65,000 IOP/s

LAN: 4xDDR IB HCA (RDMA)⁺

- ▶ Potential peak data rate per port < 1250 MB/s
 - Limited by IBolP(sp) protocol.
- ▶ Required peak data rate per port < 700 MB/s
 - The reason this value is so low is that the NSD servers are configured with 4 IB LAN ports.⁺

SAN: 4xDDR IB HCA (SRP)

- ▶ Potential peak data rate per **host connection** < 780 MB/s
 - Limited by the busses in the DCS9900.
- ▶ Required peak data rate per **host connection** < 715 MB/s

Capacity Analysis

- ▶ 15Krpm FC Disk
 - 160 disks @ 450 GB/disk
 - 16 x 8+2P RAID 6 tiers
 - raw capacity < 70 TB
 - usable capacity < 56 TB

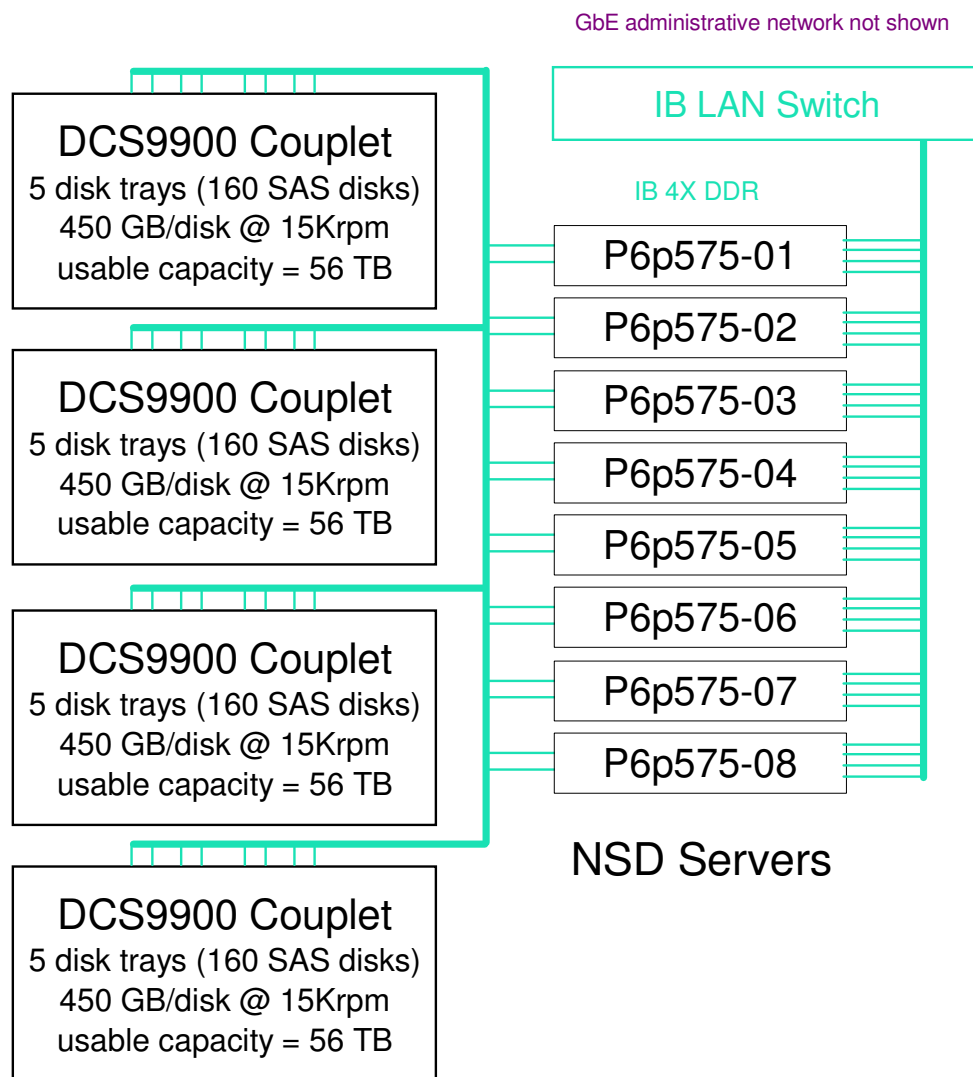
FOOTNOTES:

- ⁺4 IB LAN ports per NSD server is overkill, but 2 IB LAN ports are not quite enough. Since peak performance is the objective of this design, the "extra" IB LAN ports are recommended.
- ★ If you need more than 300 SAS disks to meet capacity requirements, a SATA solution may be sufficient; n.b., data is secure on SATA given the DCS9900 RAID 6 architecture.



Building Block #3A

4 Building Blocks, Performance Optimized



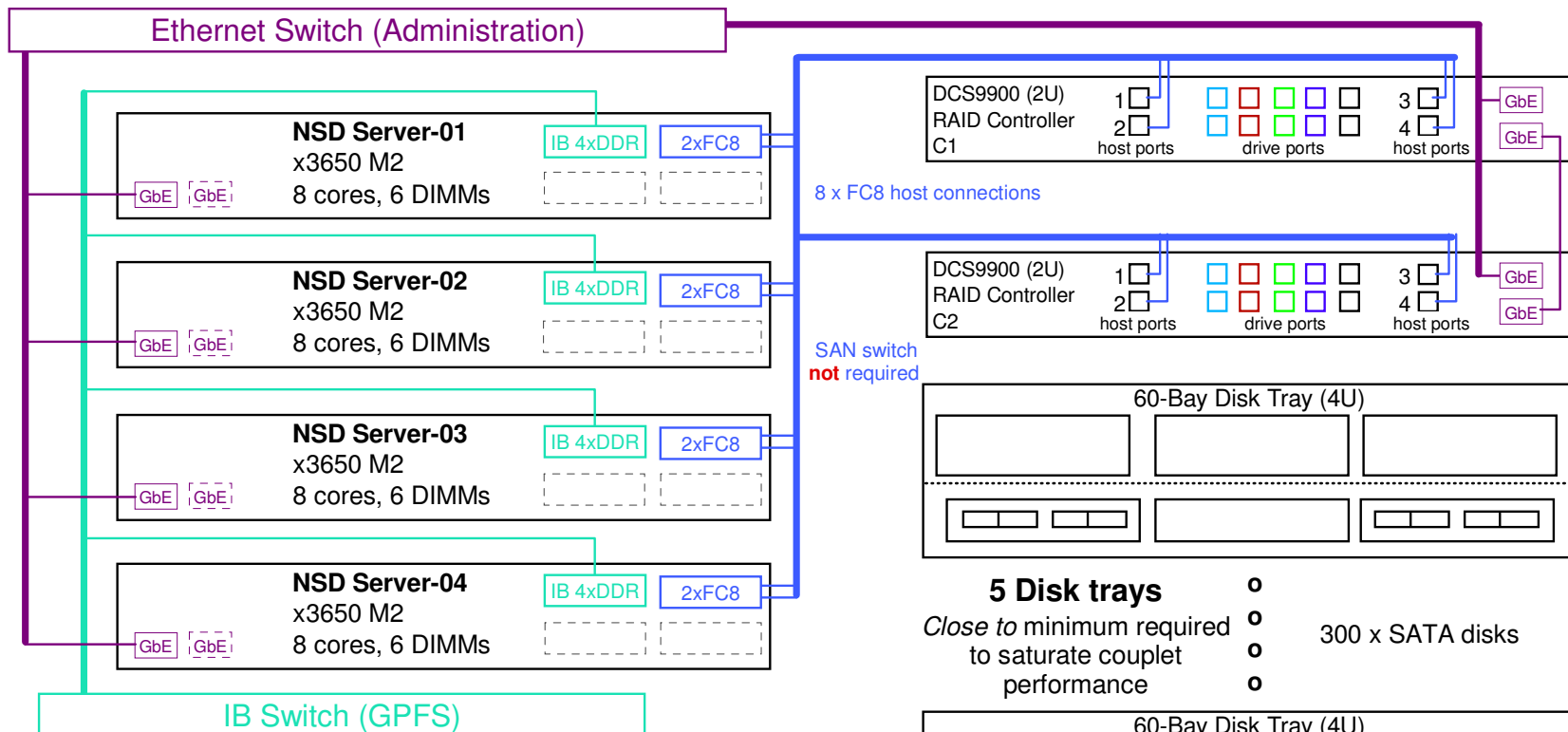
Analysis

- Capacity
 - raw < 280 TB
 - usable < 224 TB
- Performance
 - streaming rate
 - write < 20 GB/s
 - read < 16 GB/s
 - IOP rate
 - write < 160,000 IOP/s
 - read < 260,000 IOP/s
- Performance to Usable Capacity Ratio
 - streaming rate
 - write < 91 MB/s / TB
 - read < 77 GB/s / TB
 - IOP rate
 - write < 714 IOP/s / TB
 - read < 1160 IOP/s / TB
- Racks
 - Storage Racks (45u x 19"): 2
 - Server Racks: 1



Building Block #3B

Balanced Capacity/Performance



Performance Analysis

DCS9900 Performance

- ▶ Streaming data rate
 - write < 4.8 GB/s
 - read < 3.1 GB/s
- ▶ Noncached IOP rate
 - write < 47,000 IOP/s
 - read < 33,000 IOP/s

LAN: 4xDDR IB HCA (RDMA)

- ▶ Potential peak data rate per HCA < 1500 MB/s
 - ▶ Required peak data rate per HCA < 1200 MB/s
- SAN: 2xFC8 (dual port 8 Gbit/s Fibre Channel)
- ▶ Potential peak data rate per 2xFC8 < 1500 MB/s
 - ▶ Required peak data rate per 2xFC8 < 1200 MB/s

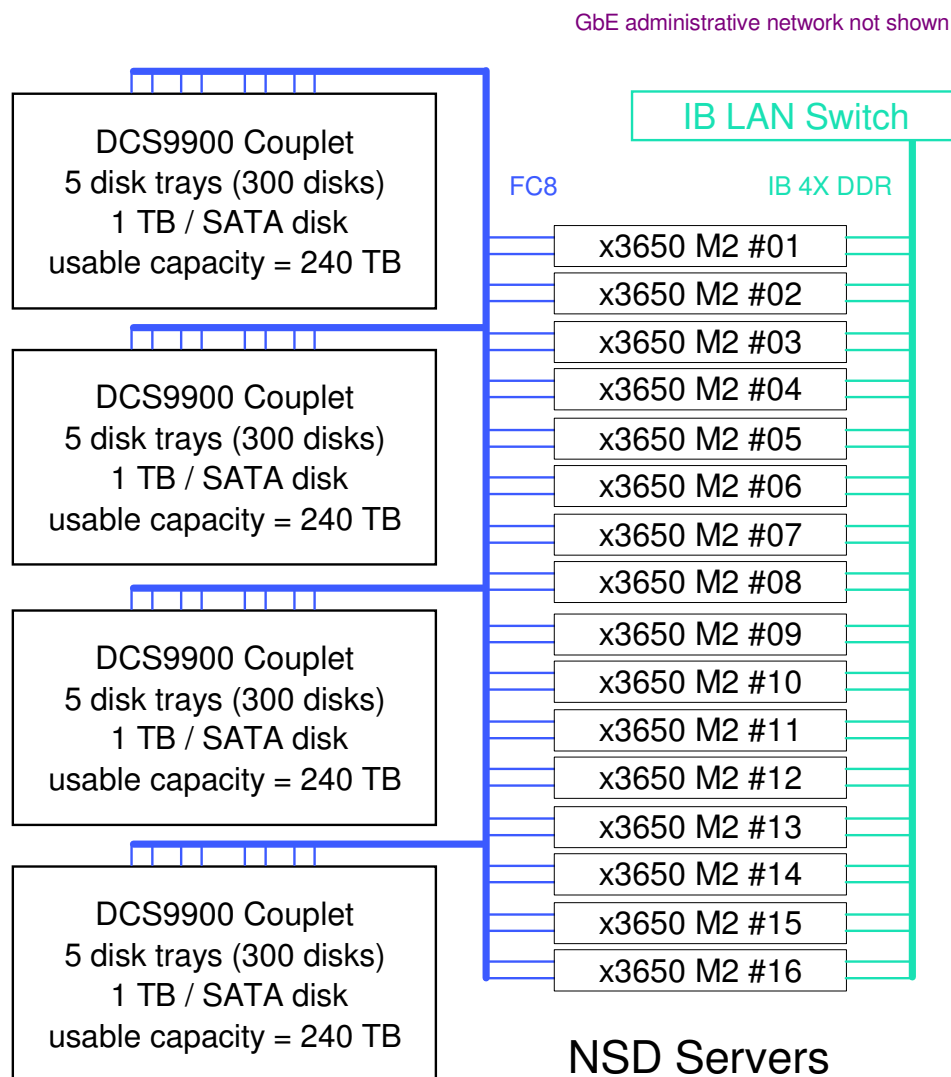
Capacity Analysis

- ▶ SATA
 - 300 disks @ 1 TB/disk
 - 30 x 8+2P RAID 6 tiers
 - raw capacity < 300 TB
 - usable capacity < 240 TB



Building Block #3B

4 Building Blocks, Balanced Capacity/Performance



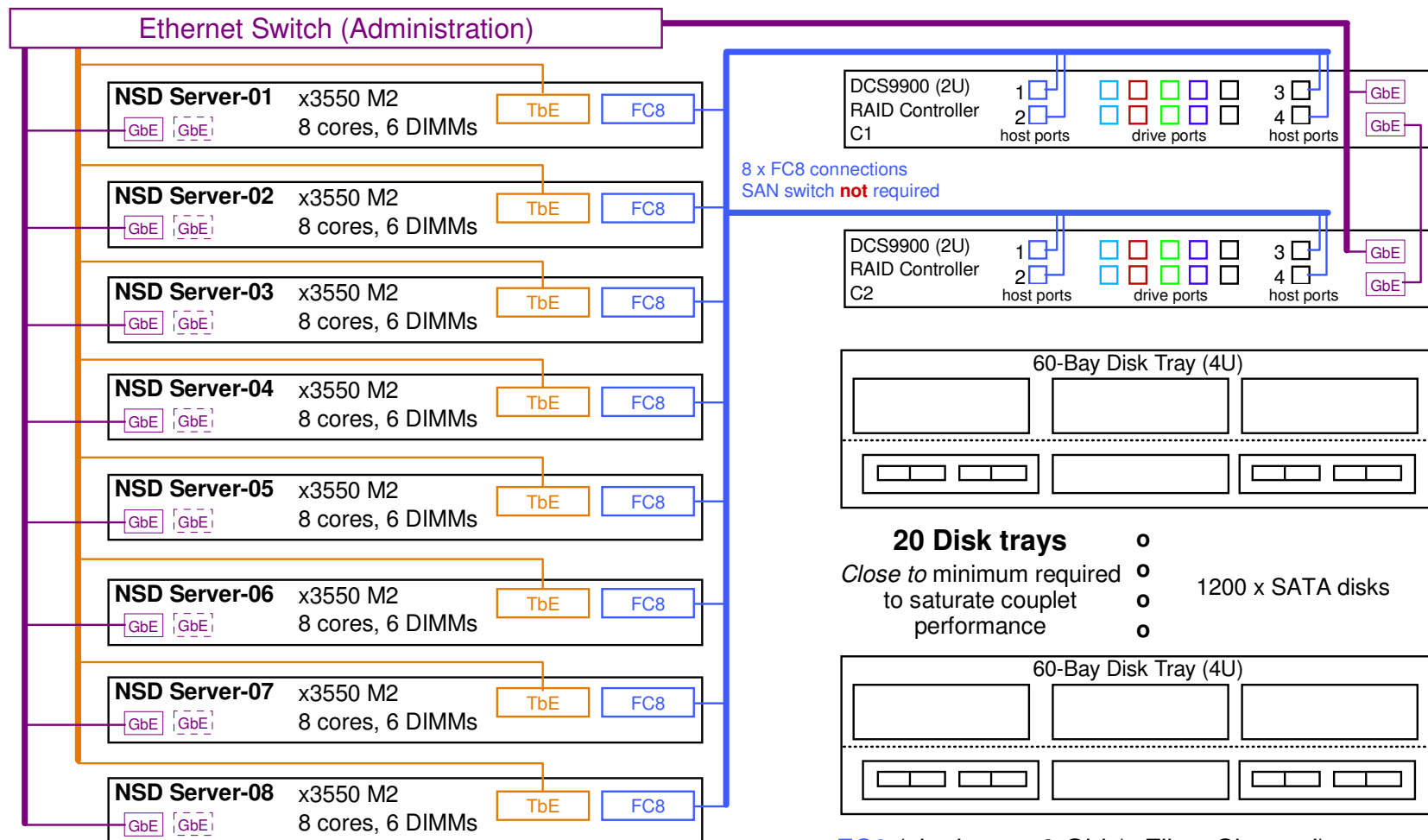
Analysis

- Capacity
 - raw < 1200 TB
 - usable < 960 TB
- Performance
 - streaming rate
 - write < 18 GB/s
 - read < 12 GB/s
 - IOP rate
 - write < 180,000 IOP/s
 - read < 130,000 IOP/s
- Performance to Usable Capacity Ratio
 - streaming rate
 - write < 19 MB/s / TB
 - read < 12 MB/s / TB
 - IOP rate
 - write < 187 IOP/s / TB
 - read < 135 IOP/s / TB
- Racks
 - Storage Racks (45u x 19"): 2
 - Server Racks (42u x 19"): 1



Building Block #3C

Capacity Optimized



Performance Analysis

DCS9900 Performance

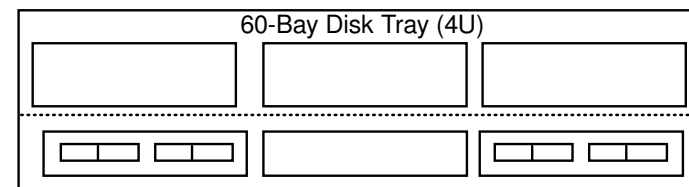
- ▶ Streaming data rate
 - write < 5.4 GB/s
 - read < 3.5 GB/s
- ▶ Noncached IOP rate*
 - write < 52,000 IOP/s
 - read < 33,000 IOP/s

TbE (10 Gbit Ethernet Adapter)

- ▶ Potential peak data rate per TbE < 725 MB/s
- ▶ Required peak data rate per TbE < 700 MB/s

20 Disk trays

- Close to minimum required to saturate couplet performance
- 1200 x SATA disks



FC8 (single port 8 Gbit/s Fibre Channel)

- ▶ Potential peak data rate per FC8 < 760 MB/s
- ▶ Required peak data rate per FC8 < 700 MB/s

Capacity Analysis

- ▶ SATA
 - 1200 disks @ 2 TB/disk - raw capacity < 2400 TB
 - 120 x 8+2P RAID 6 tiers - usable capacity < 1920 TB

FOOTNOTES:

- ★ Validation testing needed

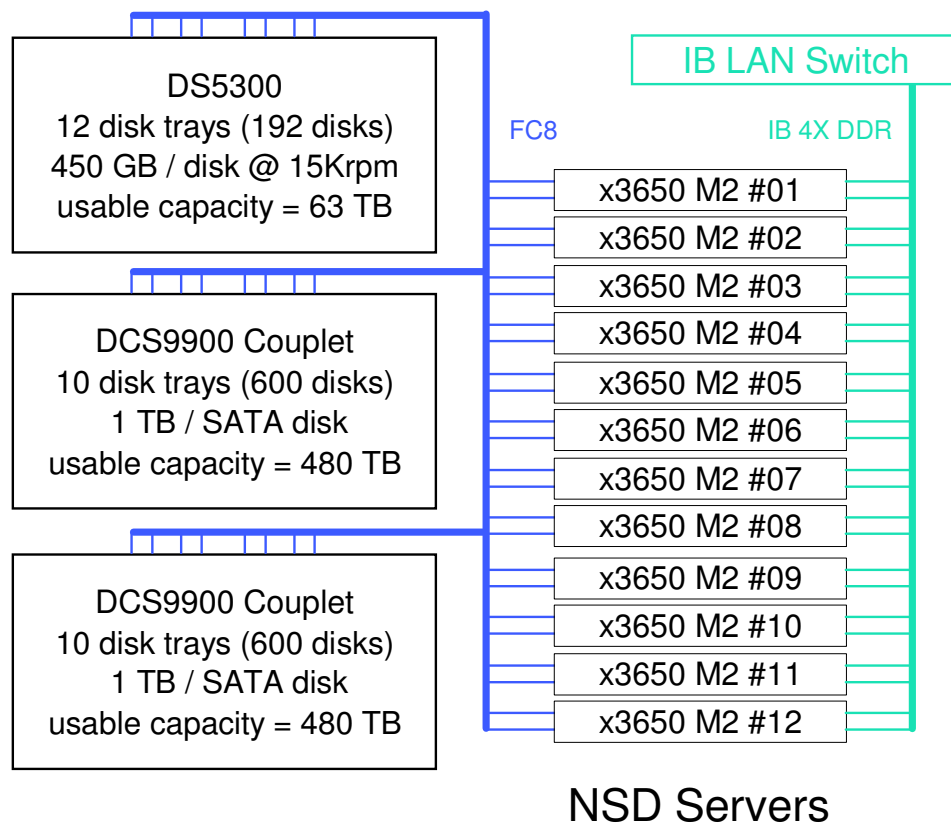


Multi-tiered Storage

Example: Building Blocks 2A and 3B



GbE administrative network not shown





SAN Configurations

The concept of integrating storage servers and controllers into building blocks does not generalize as well for SAN file systems.

The following pages illustrate how GPFS can be deployed using a SAN configuration.

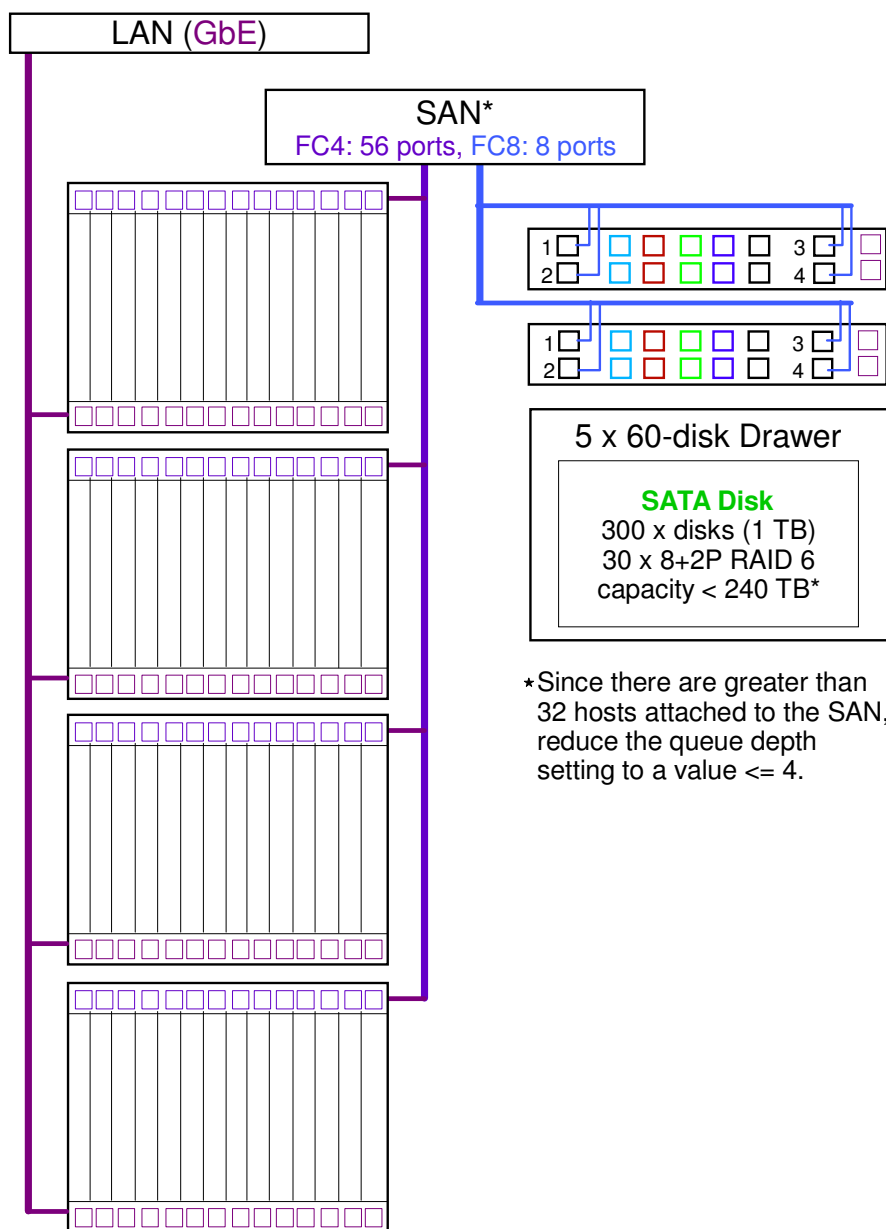
COMMENT:

If the configuration is small enough, a SAN switch (e.g., Brocade or McData) is not needed.



SAN #1

Linux/Blades



Blades (56)

- ▶ Ports per blade
 - 1 x GbE
 - 1 x FC4
- ▶ FC4 (4 Gbit/sec)
 - up to 380 MB/s per blade
- ▶ Max Aggregate FC4 BW
 - ~ 20 GB/s
- ▶ Average I/O BW per blade for *this* configuration
 - write < 90 MB/s per blade
 - read < 50 MB/s per blade

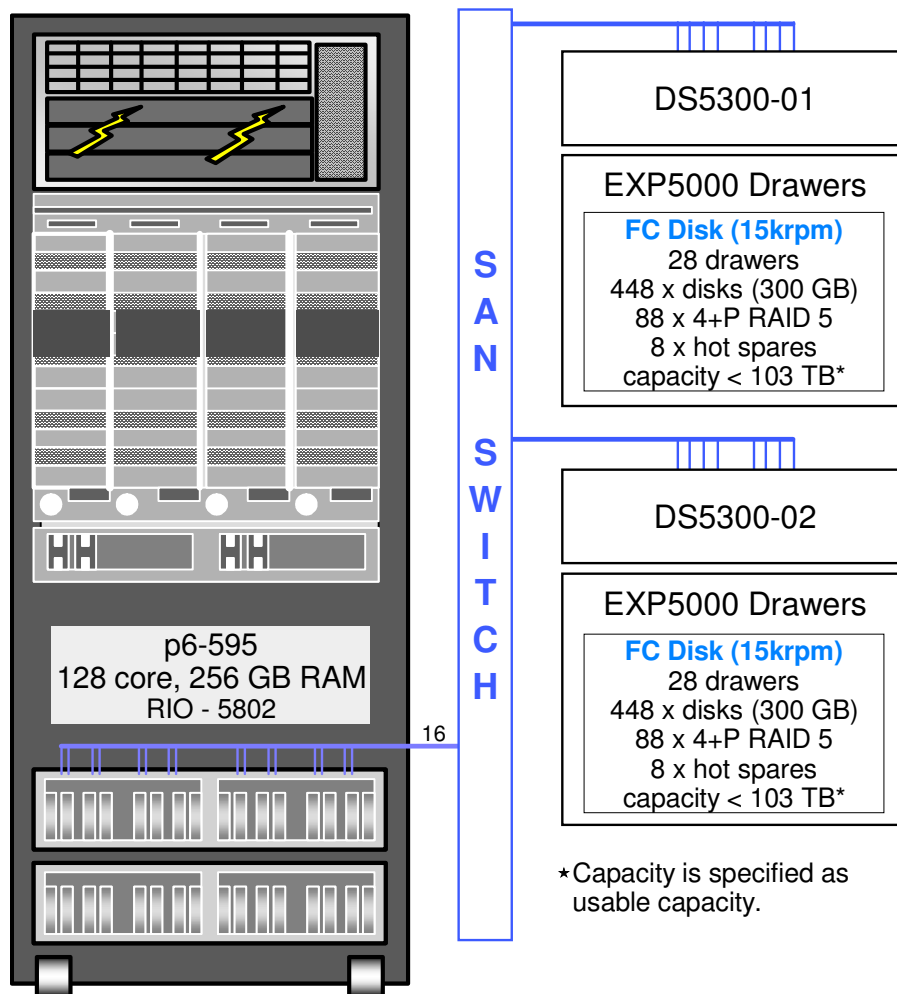
Storage

- ▶ DCS9900
 - data rate
 - write < 5 GB/s, read < 3 GB/s
 - disk: SATA
 - 300 disks
 - 8+2P RAID 6
 - raw capacity < 300 TB
 - usable capacity < 240 TB



SAN #2A

AIX/System P - Optimize IOP Performance



COMMENTS

- ▶ Since the objective is to optimize the IOP rate per DS5300, faster (15Krpm) but smaller (300 GB/disk) FC disks were chosen.
- ▶ Max IOP performance requires using all of the disks supported by a single DS5300 (*i.e.*, 448) and specialized tuning (*e.g.*, "short stroking"); this tuning will decrease the usable capacity.
- ▶ The number of FC8 ports are configured to also support peak streaming BW.
- ▶ Best practice: Configure with at least 4 partitions for use with GPFS.

P6-p595 FC Ports

- ▶ FC8 = 8 Gbit/s;
 - usable BW < 760 MB/s
- ▶ 16 x FC8 ports per system
- ▶ Aggregate data rate
 - at most 12 GB/s

Configured with enough RIO drawers, peak sustained BW of a P6-p595 is much greater than this.

DS5300 ANALYSIS

- ▶ Assume **4+P RAID 5**
- ▶ Data Rate
 - per DS5300
 - write < 4.5 GB/s
 - read < 5.0 GB/s
 - aggregate
 - write < 9.0 GB/s
 - read < 10.0 GB/s
- ▶ IOP Rate
 - per DS5300
 - write < 30 Kiop/s
 - read < 150 Kiop/s
 - aggregate
 - write < 60 Kiop/s
 - read < 300 Kiop/s
- ▶ Capacity
 - per DS5300
 - raw < 130 TB
 - usable < 103 TB
 - aggregate
 - raw < 260 TB
 - usable < 206 TB

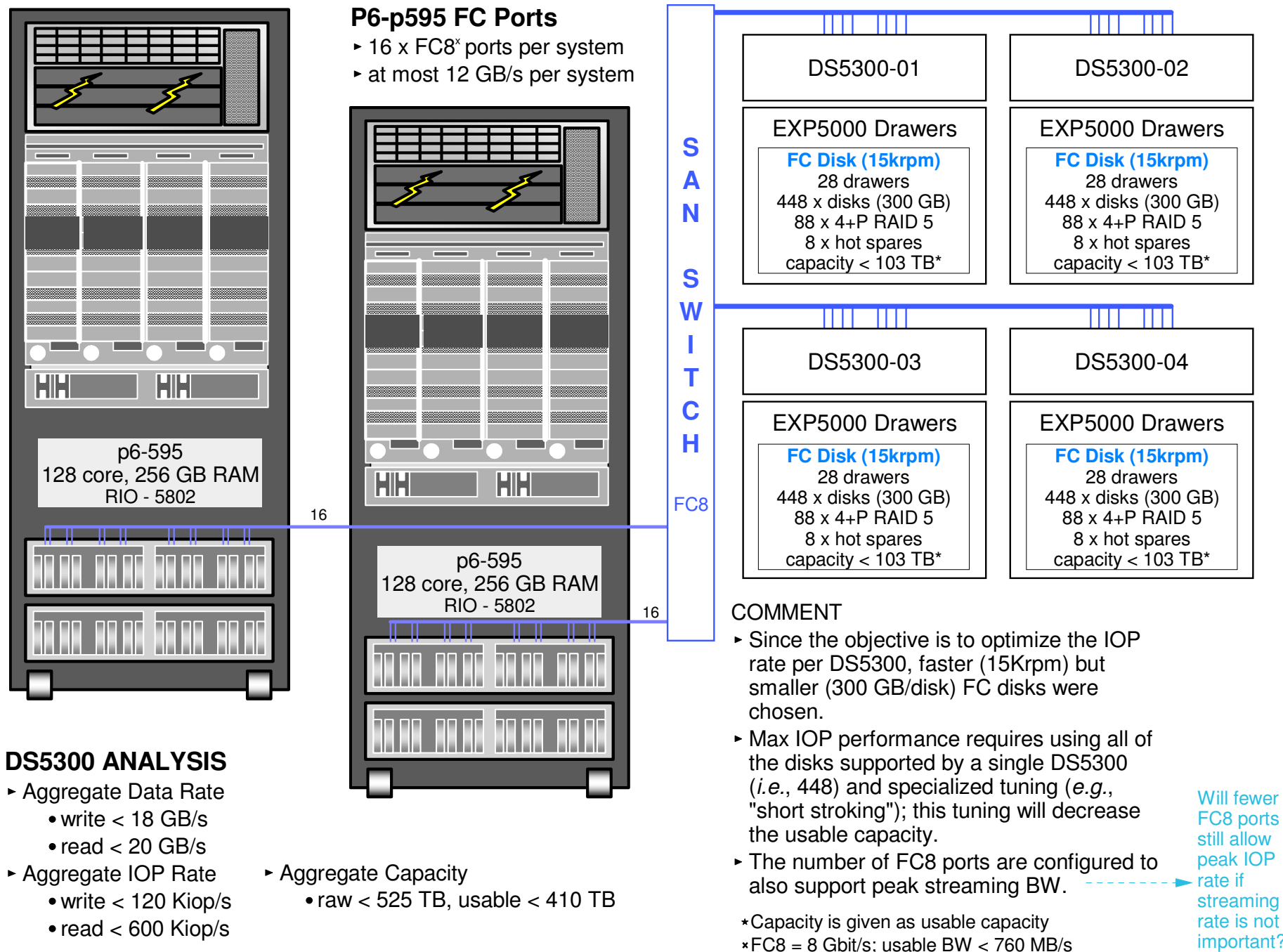
Quoted data rates are a conservative estimate, especially for the read rates. Validation is required.

Quoted IOP rates are derived from benchmark tests using different configurations are provided for informational purposes only. Validation is required.



SAN #2B-1

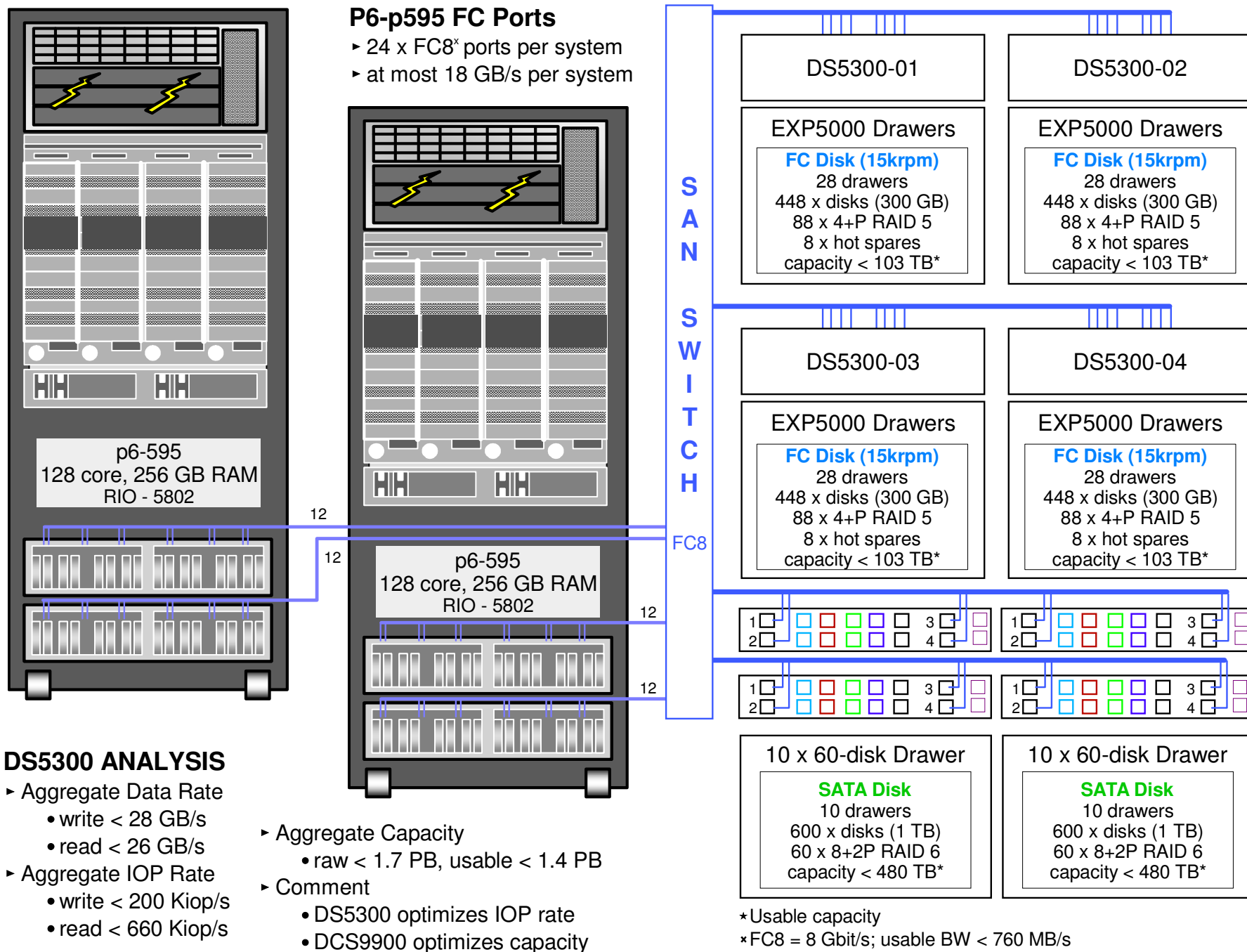
AIX/System P - Optimize IOP Performance





SAN #2B-2

AIX/System P - Multi-tiered Solution





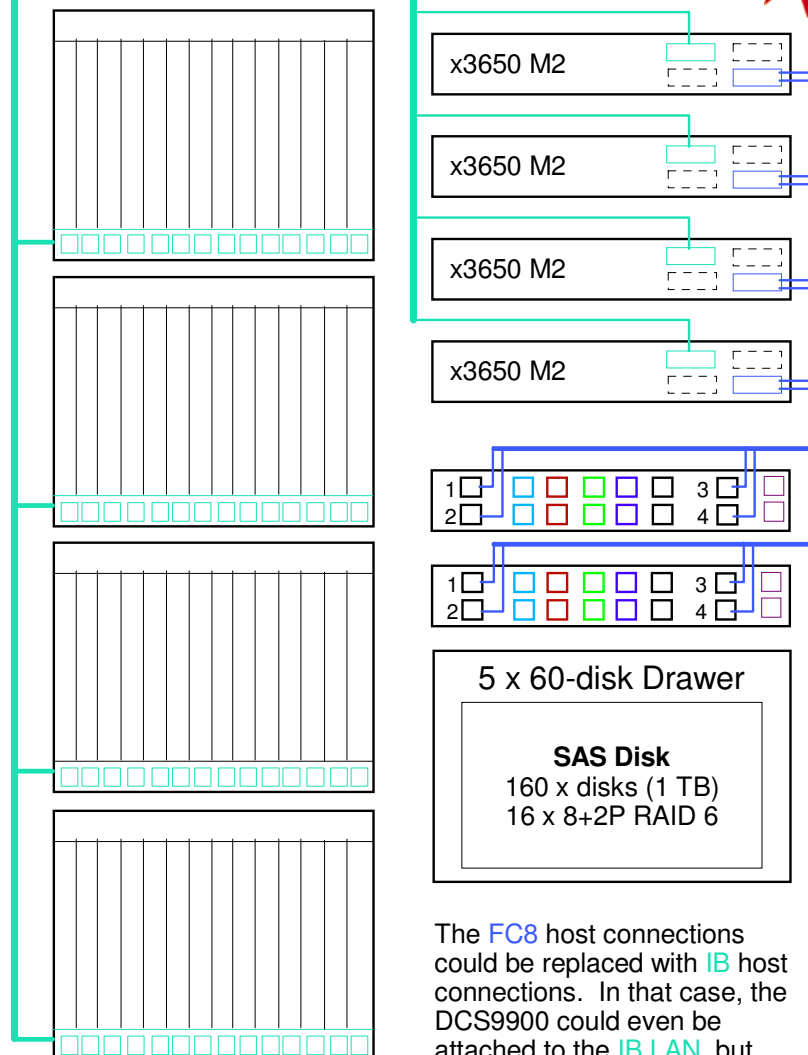
LAN vs. SAN

An Example



IB LAN

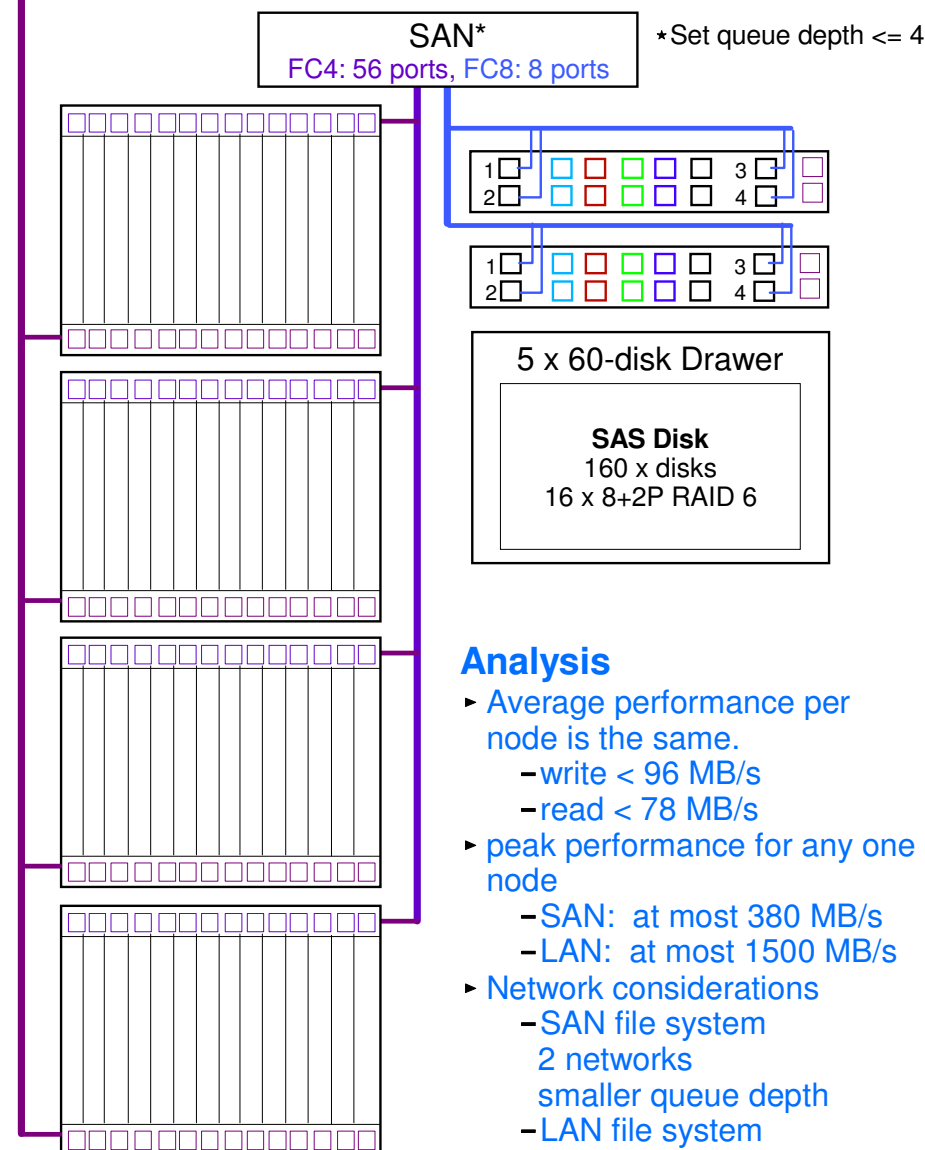
IB can be replaced by Ethernet. In that case, use 2 x TbE in place of IB HCAs in the NSD servers.



The FC8 host connections could be replaced with IB host connections. In that case, the DCS9900 could even be attached to the IB LAN, but that only increases the IB switch port count with little added benefit.



Ethernet LAN



*Set queue depth <= 4

Analysis

- Average performance per node is the same.
 - write < 96 MB/s
 - read < 78 MB/s
- peak performance for any one node
 - SAN: at most 380 MB/s
 - LAN: at most 1500 MB/s
- Network considerations
 - SAN file system
 - 2 networks
 - smaller queue depth
 - LAN file system
 - simpler network
 - larger queue depth



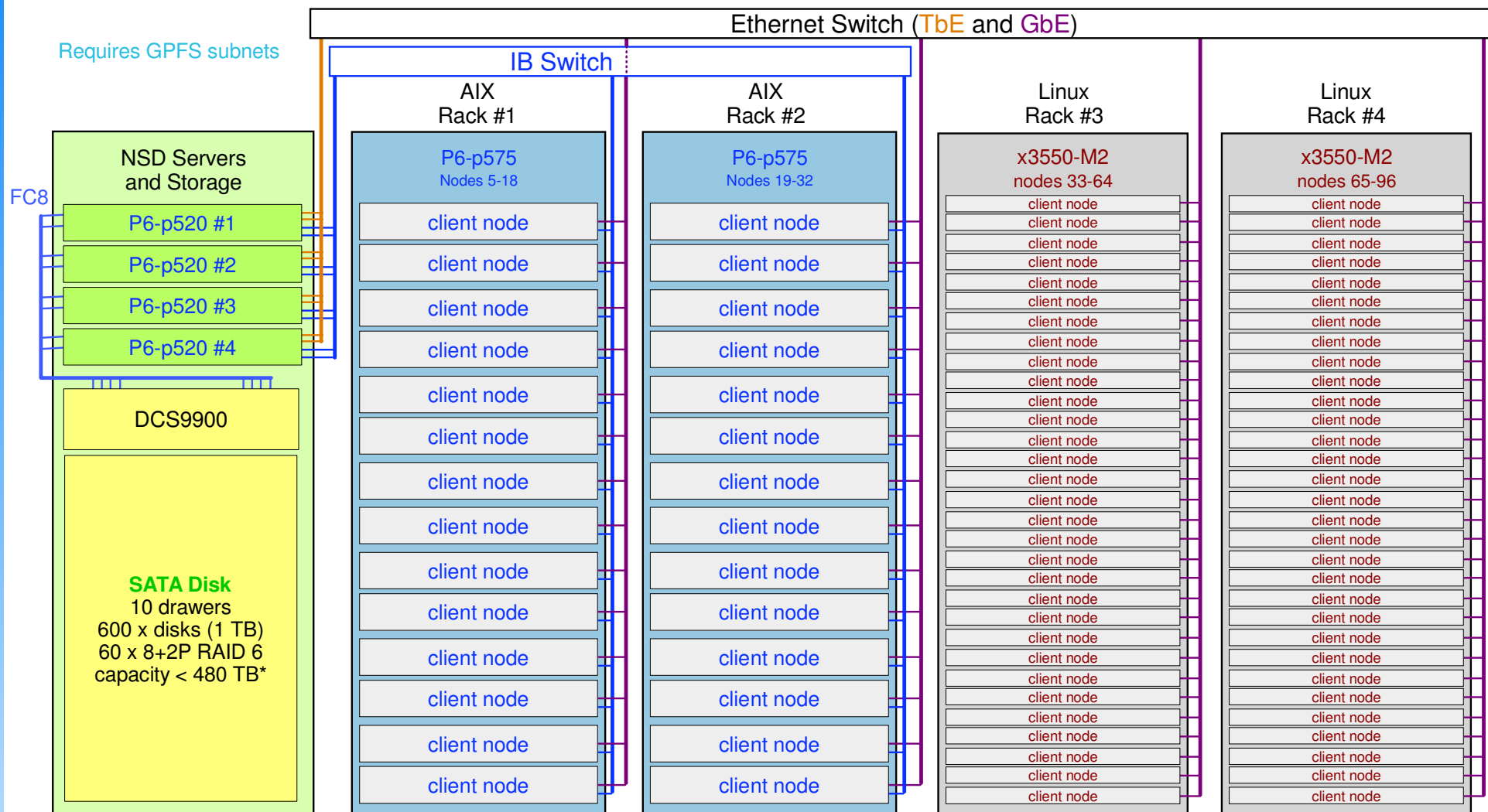
Other Miscellaneous GPFS Configurations

The following pages contain other examples of GPFS configurations further illustrating the versatility of GPFS.



Mixed AIX/Linux Environment

System P and System X with Mixed IB/Ethernet Fabric



DCS5300 Data Rate

- ▶ write < 5.4 GB/s
- ▶ read < 3.5 GB/s

BW per client node

- ▶ 92 client nodes
- ▶ write: ~ 60 MB/s per node
- ▶ read: ~ 40 MB/s per node

Capacity

- ▶ raw: 600 TB
- ▶ usable: 480 TB

COMMENTS: This is an egalitarian network in the sense that

- ▶ any client node (pSeries or xSeries) can consume upto the max BW of its IB adapter
- ▶ if all of the pSeries nodes are quiescent, then the xSeries nodes can use all of the potential aggregate DCS9900 BW
- ▶ if all of the xSeries nodes are quiescent, then the pSeries nodes can use all of the potential aggregate DCS9900 BW

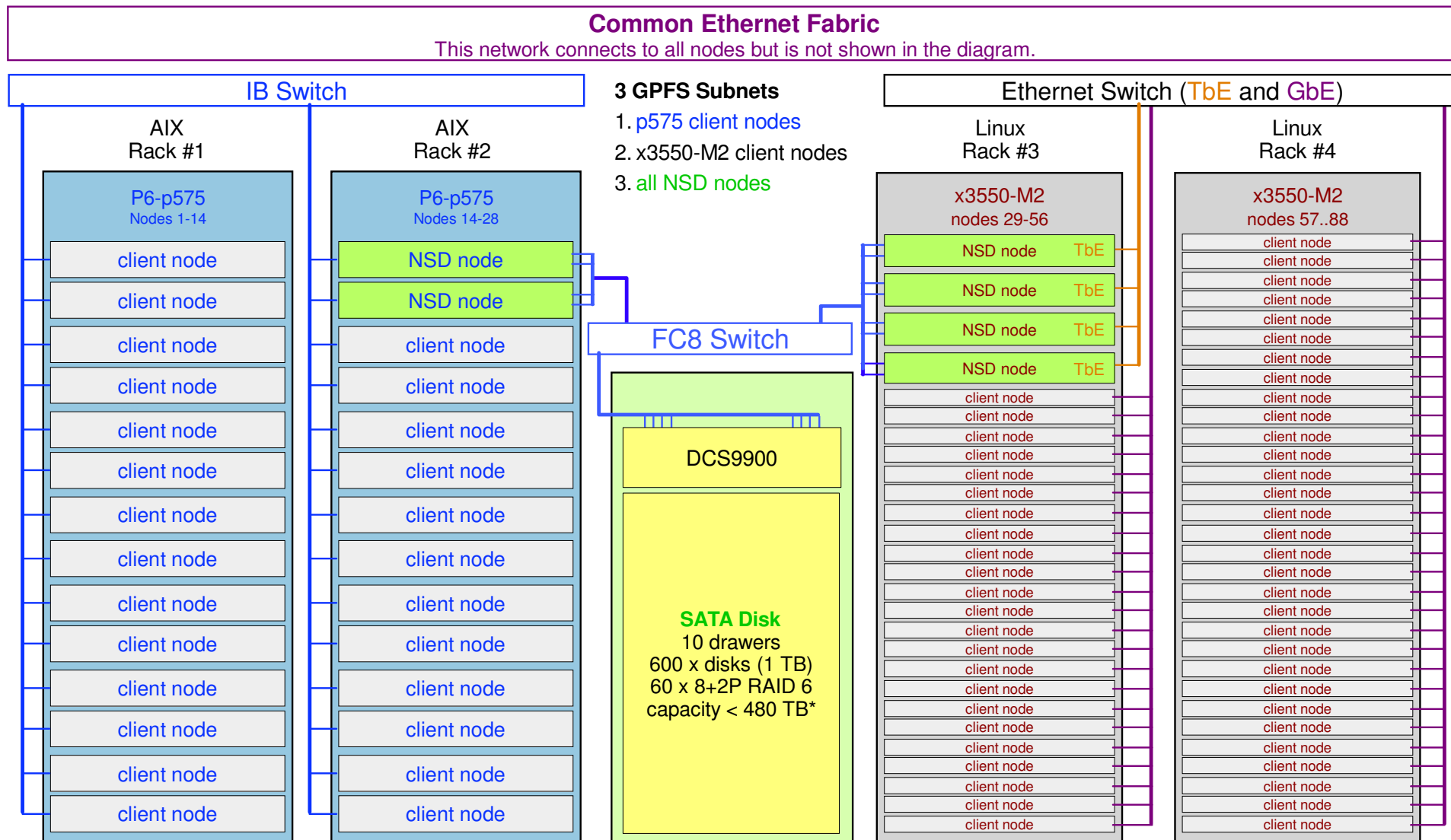
COMMENTS: Switch Fabric

- ▶ IB can **not** be used to all nodes since AIX uses IBolP(sp) and Linux used RDMA.
- ▶ Alternatively, all nodes could use an homogenous Ethernet fabric.



Mixed AIX/Linux Environment

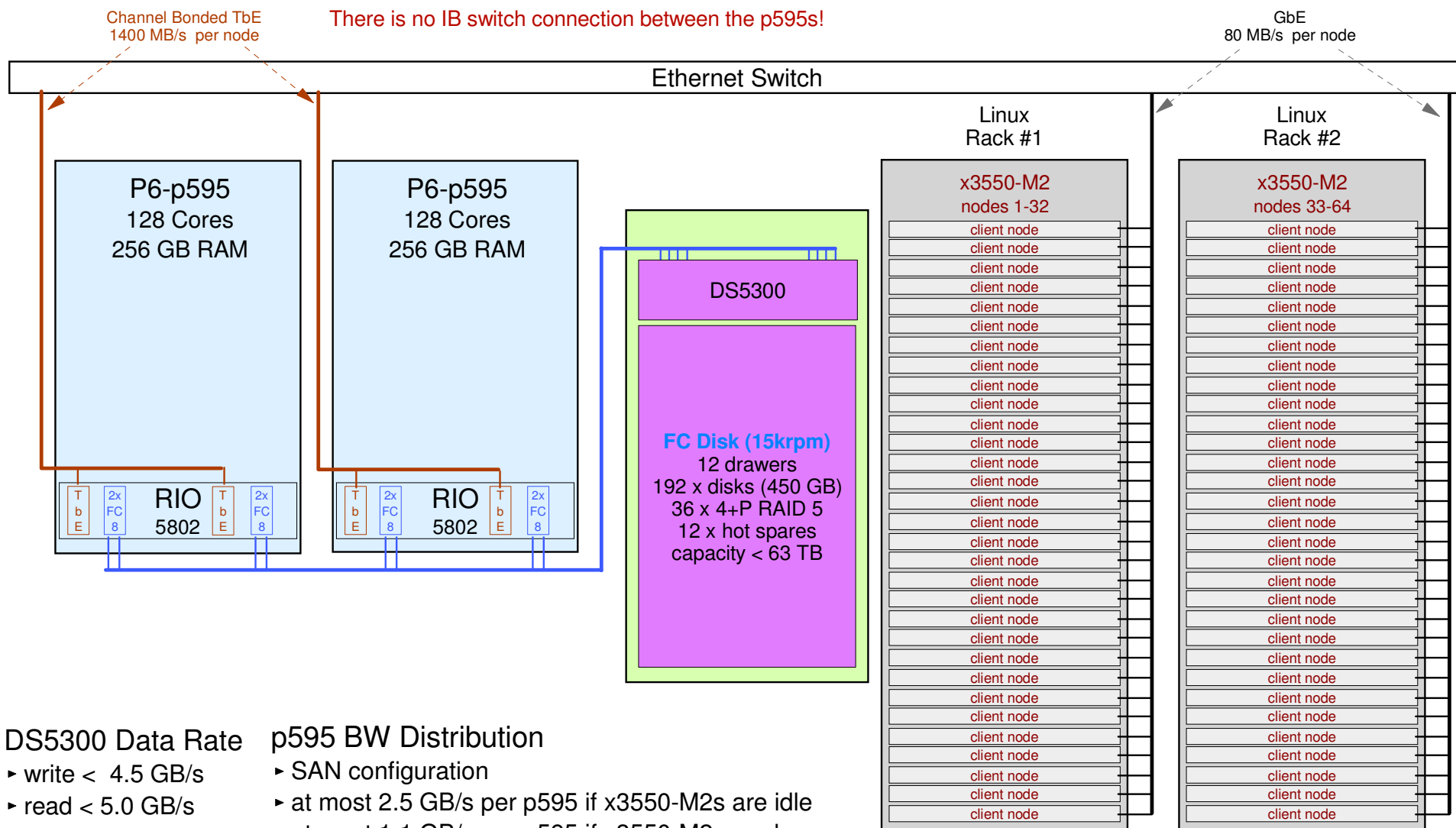
System P and System X with a Mixed LAN





Mixed AIX/Linux Environment

System P SAN with System X NSD Cluster



DS5300 Data Rate

- ▶ write < 4.5 GB/s
- ▶ read < 5.0 GB/s

Capacity

- ▶ raw: 84 TB
- ▶ usable: 63 TB

p595 BW Distribution

- ▶ SAN configuration
- ▶ at most 2.5 GB/s per p595 if x3550-M2s are idle
- ▶ at most 1.1 GB/s per p595 if x3550-M2s are busy

x3550-M2 Data Rate Distribution

- ▶ NSD configuration
- ▶ at most 2.8 GB/s over all x3550s if p595s are idle
- ▶ at most 1.4 GB/s over all x3550s if p595s are busy

COMMENT:

By design, under load this system is load balanced between both classes of nodes. The TbE network can provide up to half of the potential DS5300 BW to the x3550-M2 nodes leaving the other half for use locally on the p595 nodes.



GPFS on Blue Gene

BG/P Architecture

BG/P scales up to

- ▶ 256 racks
- ▶ Peak rate 3.56 PF/s

Terminology:

a "compute card" is also called a "node".

Full System

72 Racks - 72x32x32 Torus



1 PF/s
144 TB

Full Rack

Cabled
8x8x16 Torus

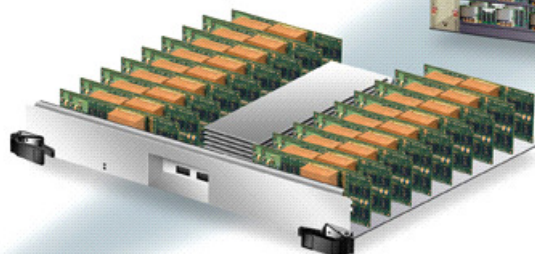
32 Node Cards



13.9 TF/s
2 TB
8 to 64 x TbE

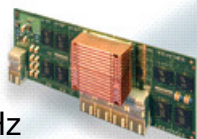
Node Card

32 compute cards - 4x4x2 torus
2 I/O cards



Compute Card

1 chip per card
20 DRAMs



435 GF/s
64 GB
0 to 2 x TbE

Chip

4 cores @ 850 MHz



13.6 GF/s
8 MB EDRAM

13.6 GF/s
2.0 GB DDR2

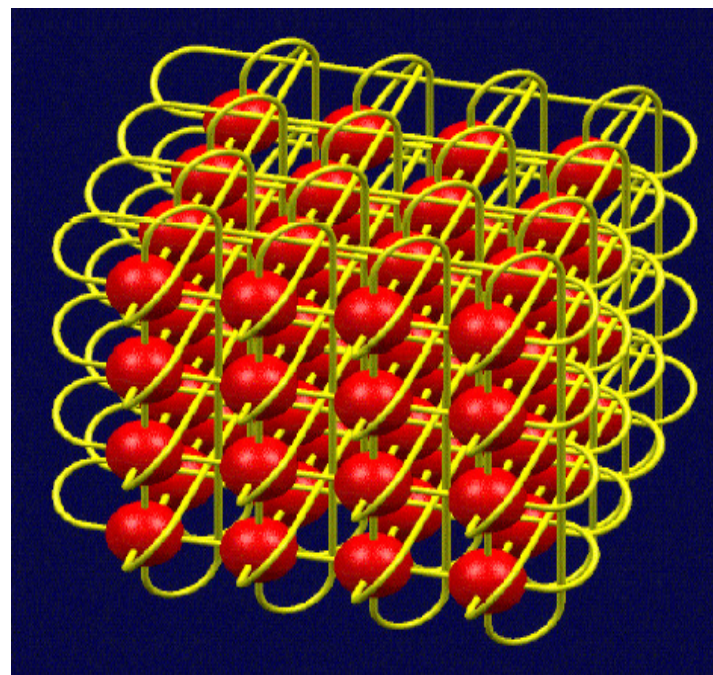
2 compute cards
fit back to back

Formula:

$$0.85 \text{ GHz} * 4 \text{ cores} * 4 \text{ pipes per core} * 1 \text{ FLOP/pipe} = 13.6 \text{ GF/s}$$

3 Networks in BG/P

1. **3D Torus** for point-to-point communications
2. **Global tree** for reduction, all-to-one communications and file I/O between the I/O and compute nodes
3. 10 Gbit/sec **Ethernet** (TbE) for file I/O, host interface, control and monitoring



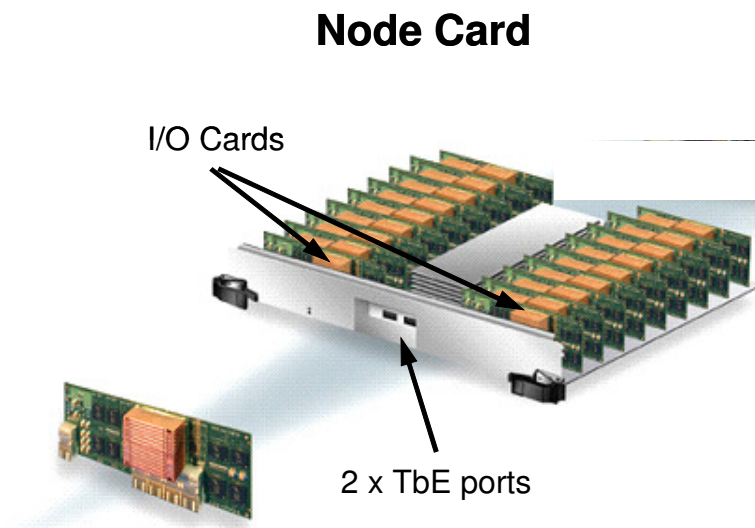
3D Torus

Each node is connected to its 6 nearest neighbors.



GPFS on Blue Gene BG/P Storage I/O

- ▶ An I/O card is similar to a compute card except that it has a single TbE port
- ▶ Each node card can be configured with 1, 2 or no I/O cards
- ▶ Each rack can be configured with 8 to 64 I/O cards
 - default = 16 I/O cards
- ▶ I/O cards connect to compute cards over the tree network
- ▶ Each I/O card acts as a storage client; external nodes act as storage servers

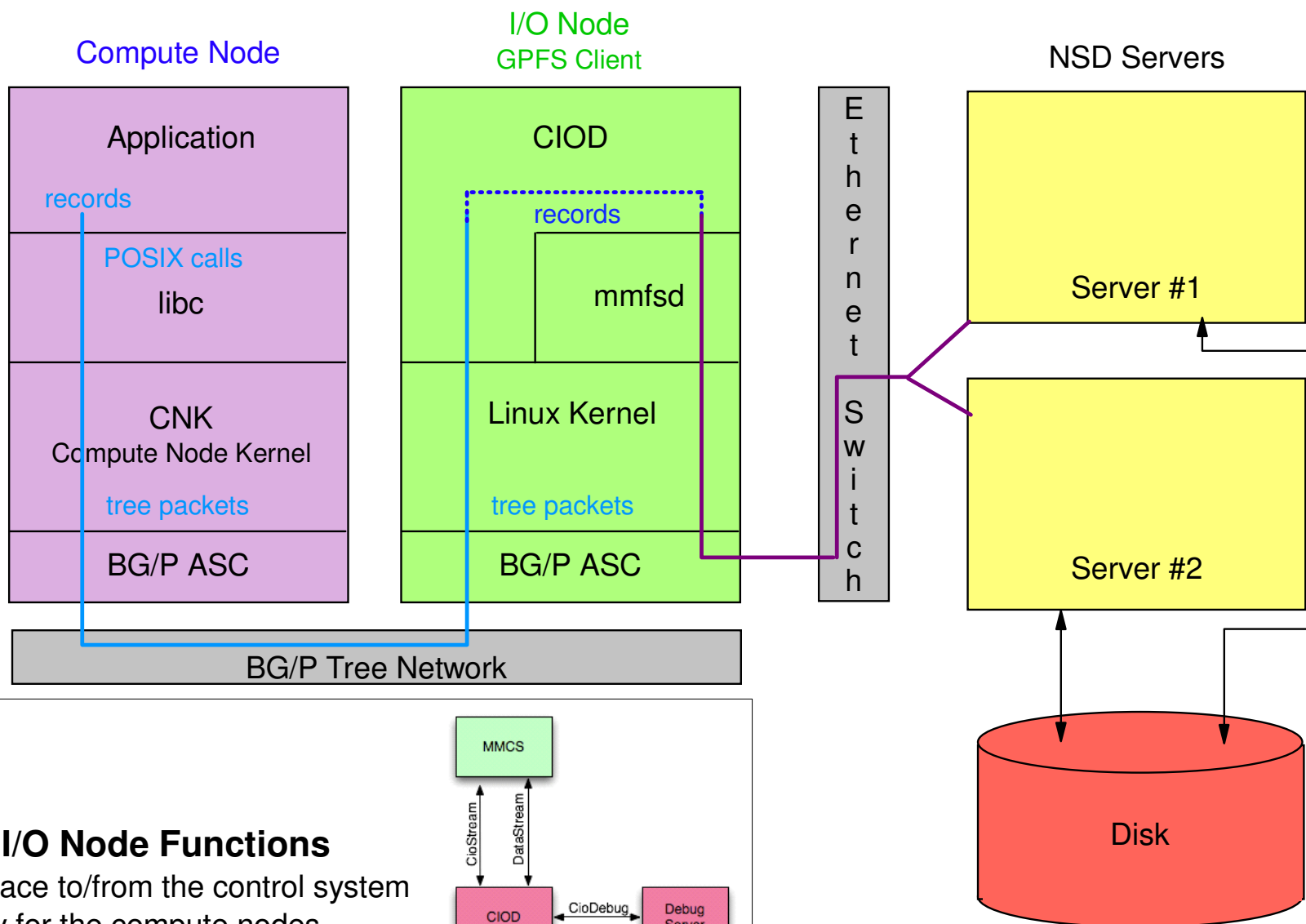




GPFS on Blue Gene

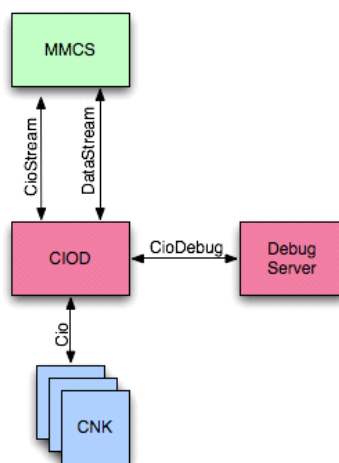
BG/P Storage I/O

COMMENT:
GPFS does
not run on
the compute
nodes.



BG/P I/O Node Functions

- Interface to/from the control system
- Proxy for the compute nodes
- Proxy for the debug server





GPFS on Blue Gene BG/P Storage Building Block

redo with
DS5300



TbE BW per Server

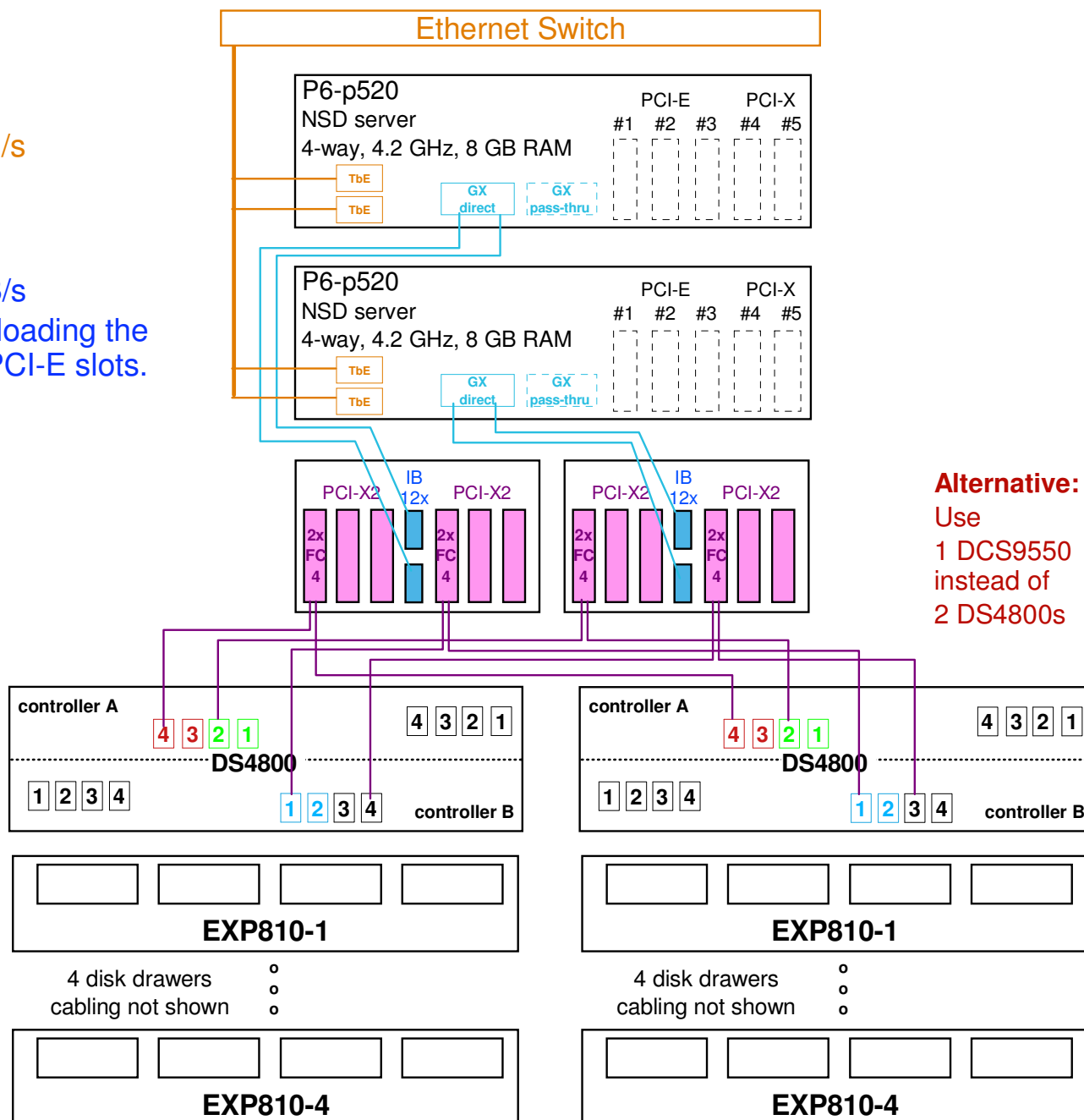
- ▶ 2 TbE per server
- ▶ data rate per server < 1500 MB/s

FC BW per Server

- ▶ 2 dual port HBAs per server
- ▶ data rate per server < 1560 MB/s
- ▶ Use RIO drawers to avoid overloading the common GX bus for TbE and PCI-E slots.

DS4800

- ▶ aggregate data rate
 - write < 2.2 GB/s
 - read < 3.0 GB/s
- ▶ aggregate capacity
 - raw: 37.5 TB
 - usable: 28.0 TB

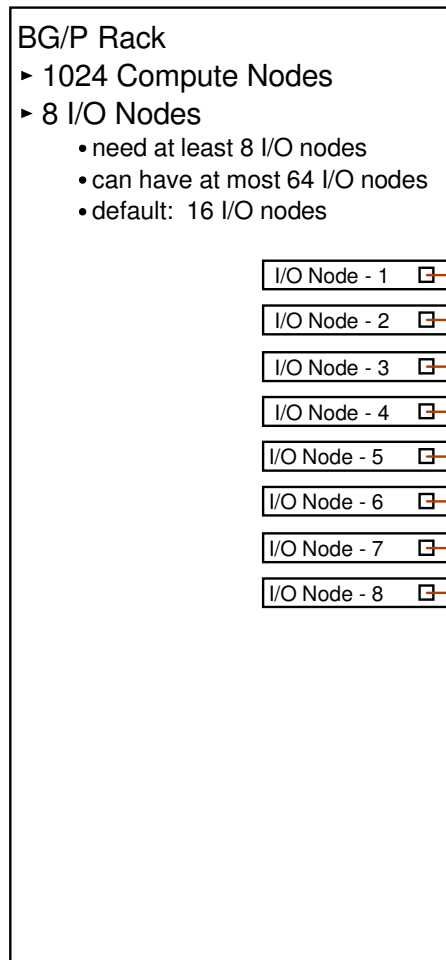




GPFS on Blue Gene

Single BG/P Frame Configuration

redo with
DS5300



Balanced Design
8 TbE connections from the BW source
8 TbE connections from the BW sync



GPFS Parameters

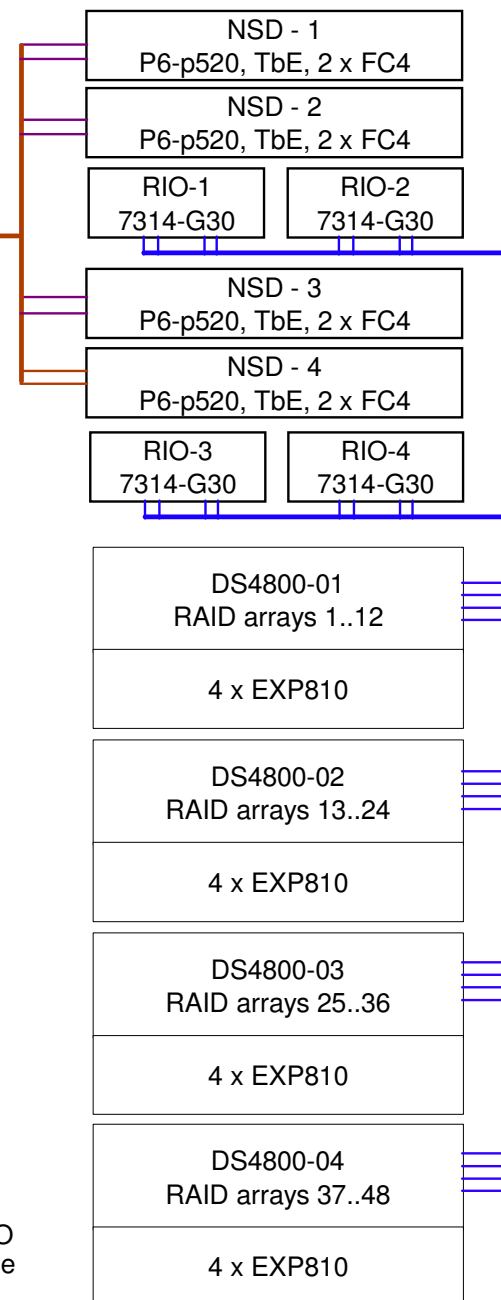
- ▶ page pool = 4 GB
- ▶ maxMBpS >= 2000 MB/s
- ▶ block size = 1024 KB

DS4800 Parameters

- ▶ RAID config = 4+P
- ▶ read ahead multiplier = 0
- ▶ write caching = off
- ▶ write mirroring = off
- ▶ read caching = on
- ▶ segment size = 256 KB
- ▶ cache block size = 16 KB

Bandwidth

- ▶ Aggregate
 - write < 4.0 GB/s
 - read < 5.6 GB/s
- ▶ BW per compute node
 - write < 4.0 MB/s per compute node
 - read < 6.0 MB/s per compute node



Alternative:

This is an "I/O poor" design using the only 8 I/O nodes (the minimum requirement). We could make this an "I/O rich" design by adding all 32 I/O nodes (the maximum allowed), but to be usefull we would need to increase the number of building blocks to 8.



The following pages contain some successful legacy designs that are still relevant today.



Smaller Building Blocks Using Different

The previous building blocks all assume the existence of a high BW switch fabric (i.e. TbE or IB).

However, many users have existing networks based on GbE only (n.b., no TbE switch ports). This leads to a building block with different granularity.



Smaller Building Blocks

Optimized for NFS and **NO** TbE

■ Basic Building Block

- 2 NSD nodes
 - x3550
 - dual core, dual socket (4 CPUs) per NSD node
 - 16 GB RAM per NSD node
 - 1 dual port FC HBA @ 4 Gb/s per NSD node
 - 2 internal GbE ports per NSD node
 - dual port GbE adapter per NSD node
- 2 SAN switches
 - SAN32B (32 ports)
 - 4 Gb/s fabric
- 1 disk controller
 - DS4800 using 4 Gb/s host side and drive side ports
- 2 disk enclosures ("drawers")
 - EXP810 with at most 16 disks per drawer
 - disk options
 - SATA/2 @ 500 GB/disk
 - 10 Krpm FC @ 146 or 300 GB/disk
 - 15 Krpm FC @ 73 or 146 GB/disk

COMMENT

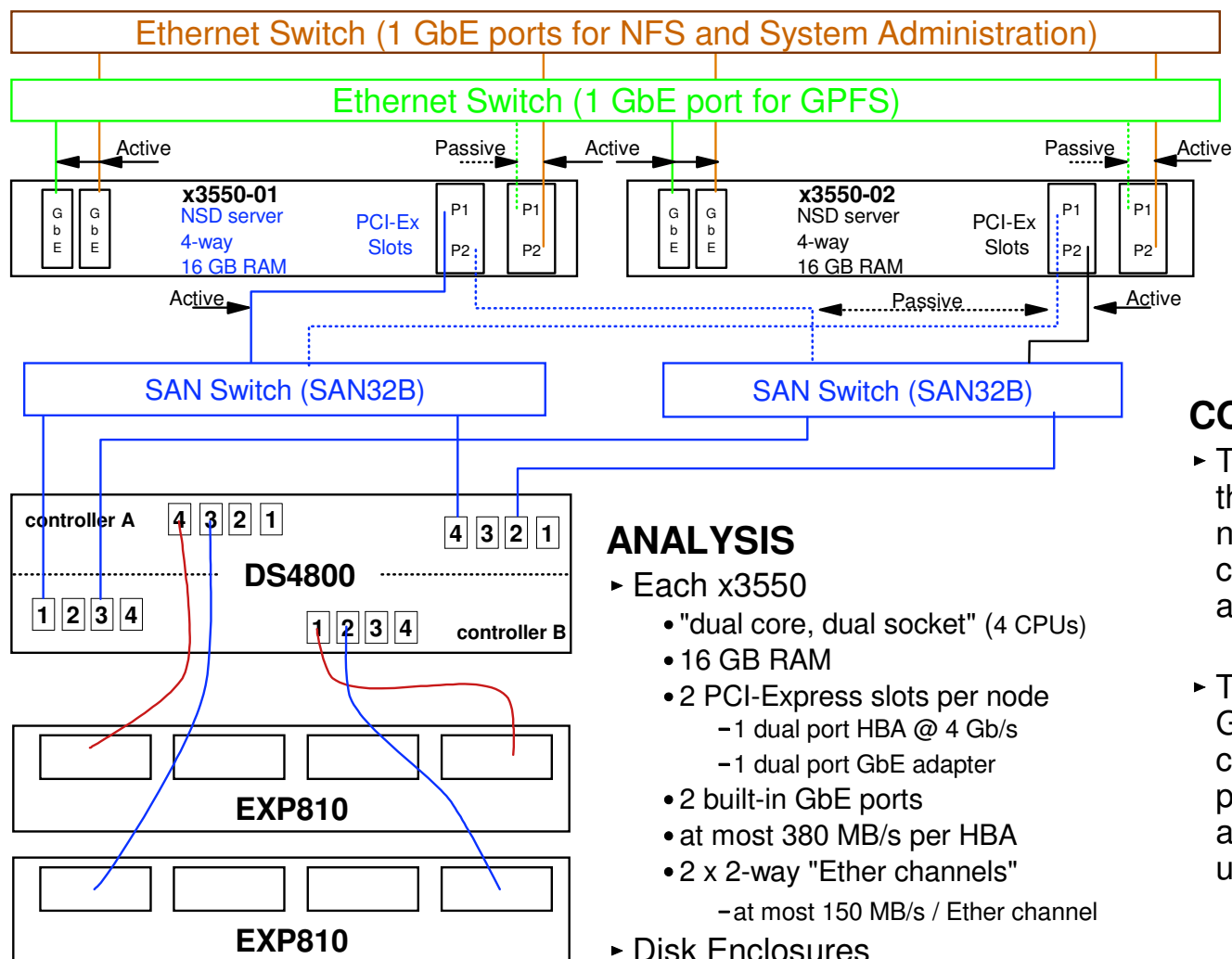
If a tape backup system is added to the storage cluster, use 2 x3650s with the following configuration for each x3650:

- dual core, dual socket
- 16 GB RAM
- 2 internal GbE ports
- dual port GbE
- dual port FC HBA
 - for disk
 - 4 Gb/s
- dual port FC HBA
 - for tape
 - 4 Gb/s



Smaller Building Blocks

1 Building Block



Sustained NFS performance:
at most 240 MB/s per building block

COMMENTS

- The 2 Ether ports dedicated to the the NFS and sysadm networks is a 2-way channel-bond, but it has 2 IP addresses.
 - sustained peak BW < 240 MB/s
- The 2 ports dedicated to GPFS are not a channel-bond; using ethernet protocols, they are configured as an active/passive bond under the same IP address.
 - sustained peak BW < 80 MB/s
 - the GPFS network is only used for GPFS overhead traffic (e.g., tokens, heartbeat, etc.) and thus minimal BW is used

ANALYSIS

- Each x3550
 - "dual core, dual socket" (4 CPUs)
 - 16 GB RAM
 - 2 PCI-Express slots per node
 - 1 dual port HBA @ 4 Gb/s
 - 1 dual port GbE adapter
 - 2 built-in GbE ports
 - at most 380 MB/s per HBA
 - 2 x 2-way "Ether channels"
 - at most 150 MB/s / Ether channel
- Disk Enclosures
 - 2 EXP810 enclosures
 - at most 16 disks per enclosure
 - 6 x 4+P RAID 5 arrays

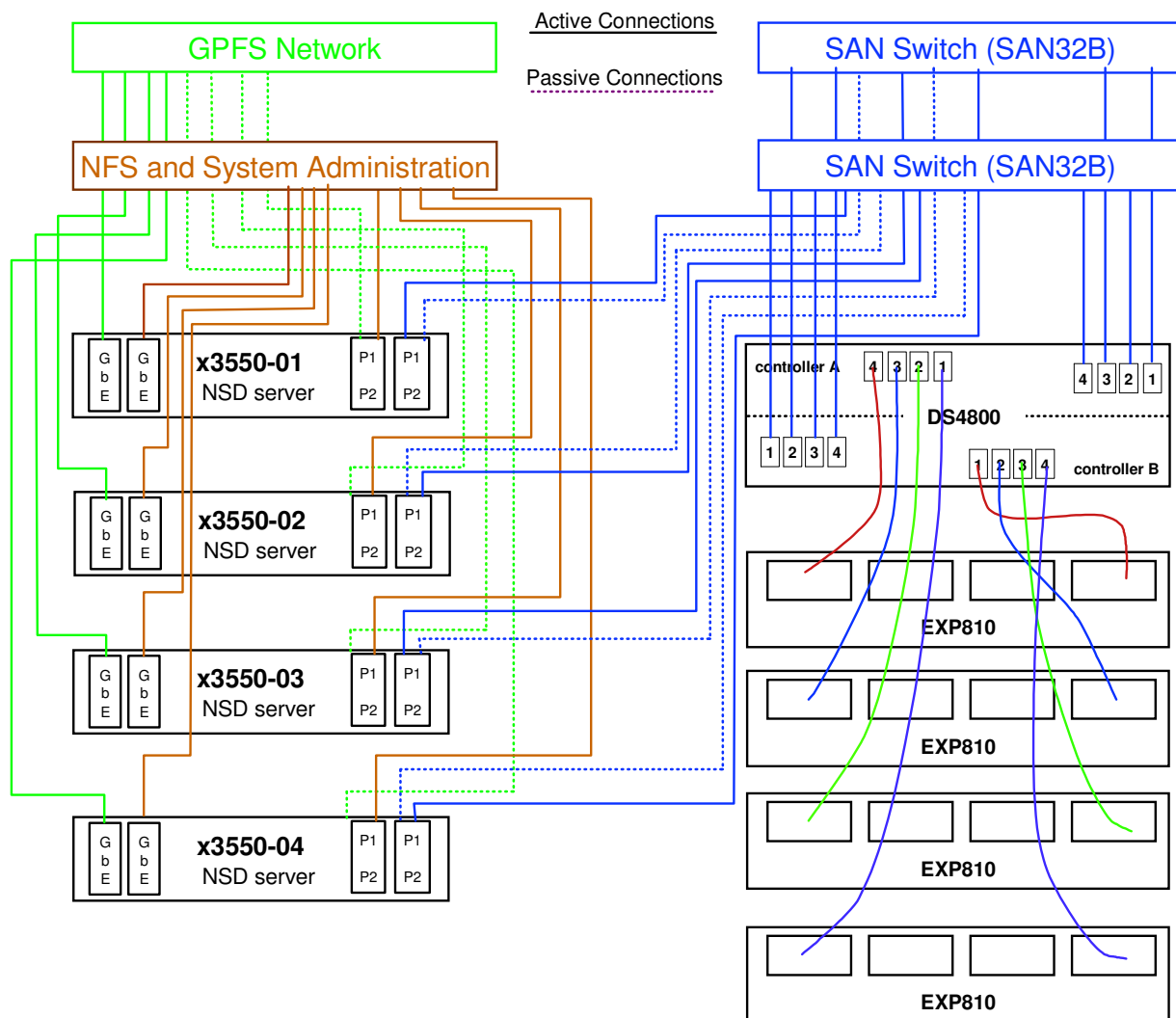
- assume 32 x 300 GB/disk @ 10 Krpm
 - write: 540 MB/s (read caching off)
 - read: 540 MB/s (write caching off)
 - 9.4 TB (raw)

- assume 32 x 500 GB/disk SATA
 - write: 180 MB/s (read caching off)
 - read: 360 MB/s (write caching off)
 - 16 TB (raw)



Smaller Building Blocks

2 Building Blocks



ANALYSIS

- 2 Building Blocks
 - at most 480 MB/s NFS BW
 - limited by the GPFS GbE adapters
- Disk Enclosures
 - 4 EXP810 enclosures
 - at most 16 disks per enclosure
 - 12 x 4+P RAID 5 arrays
 - assume 64 x 500 GB/disk **SATA**
 - write: 720 MB/s (read caching off)
 - read: 720 MB/s (write caching off)
 - 32 TB (raw)
 - assume 64 x 300 GB/disk @ **10 Krpm**
 - write: 1000 MB/s (read caching off)
 - read: 1400 MB/s (write caching off)
 - 19 TB (raw)
 - assume 64 x 146 GB/disk @ **15 Krpm**
 - write: 1100 MB/s (read caching off)
 - read: 1400 MB/s (write caching off)
 - 9 TB (raw)



Smaller Building Blocks

6 Building Blocks

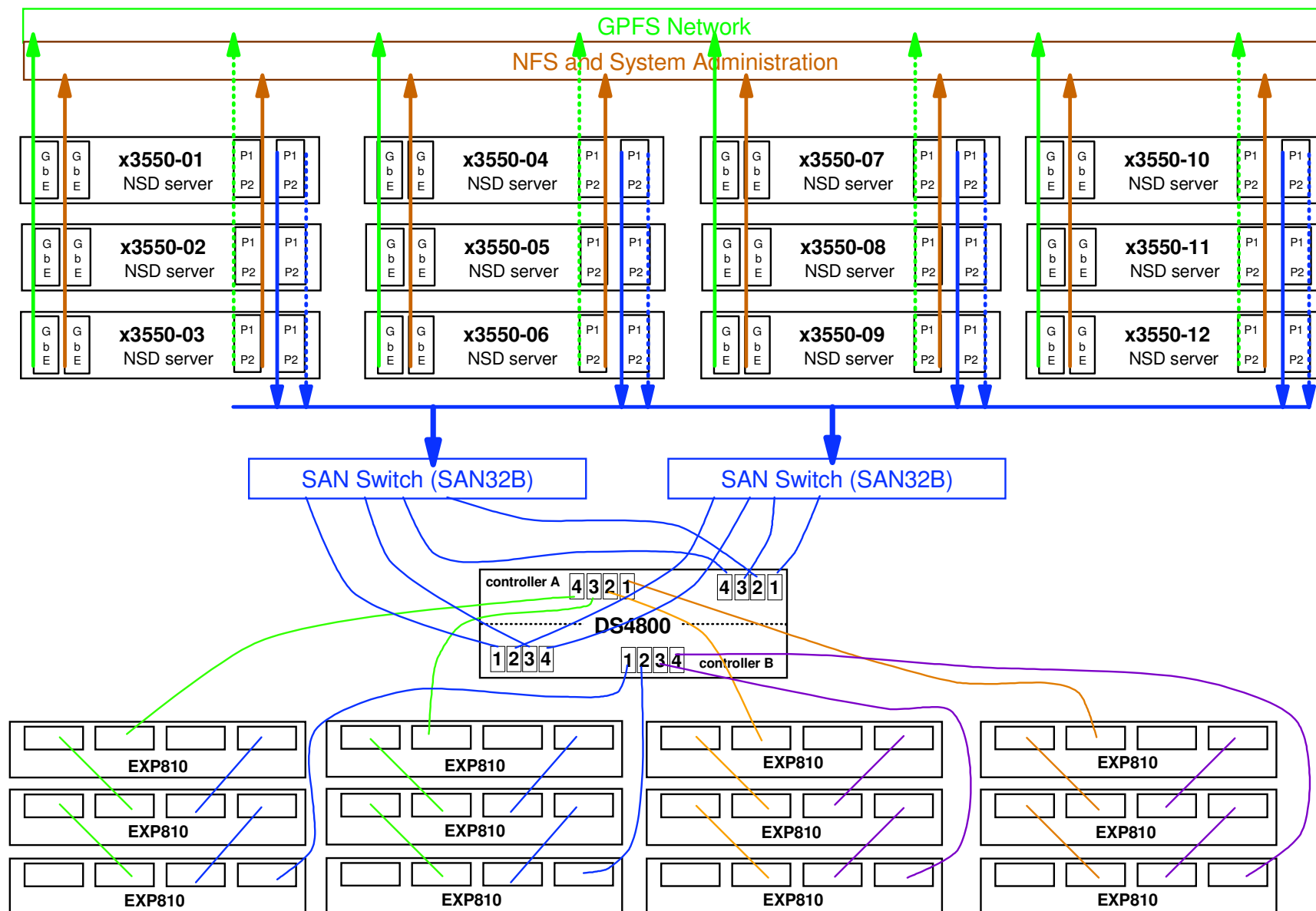
ANALYSIS

► 6 Building Blocks

- at most 1400 MB/s NFS BW

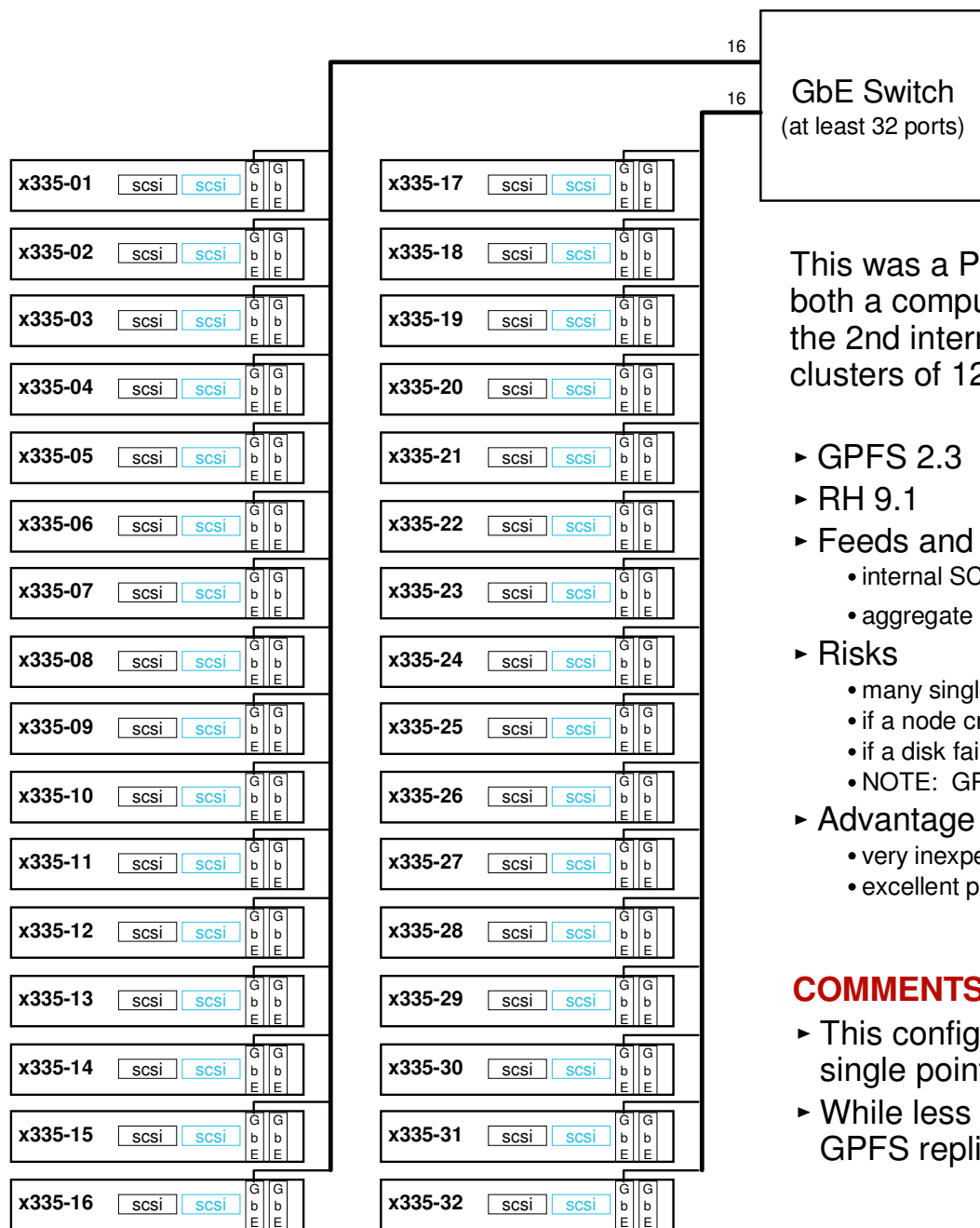
- assume 192 x 500 GB/disk **SATA**
 - write: 1400 MB/s (read caching off)
 - read: 1100 MB/s (write caching off)
 - 96 TB (raw)

- assume 192 x 300 GB/disk @ 10 Krpm
 - write: 1100 MB/s (read caching off)
 - read: 1400 MB/s (write caching off)
 - 56 TB (raw)





Linux Cluster Using Internal SCSI Drives



This was a POC test done by a customer. Each node is both a compute client and NSD node. GPFS was built on the 2nd internal SCSI disk. They now use it in production on clusters of 128 nodes.

- GPFS 2.3
- RH 9.1
- Feeds and Speeds
 - internal SCSI disk ~ 30 MB/s
 - aggregate ~ 1 GB/s
- Risks
 - many single points of failure
 - if a node crashes, GPFS file system is unavailable until the node is on-line
 - if a disk fails, the file system will be corrupted and data will be lost
 - NOTE: GPFS robustness design requires "twin tailed disk"
- Advantage
 - very inexpensive
 - excellent performance scaling

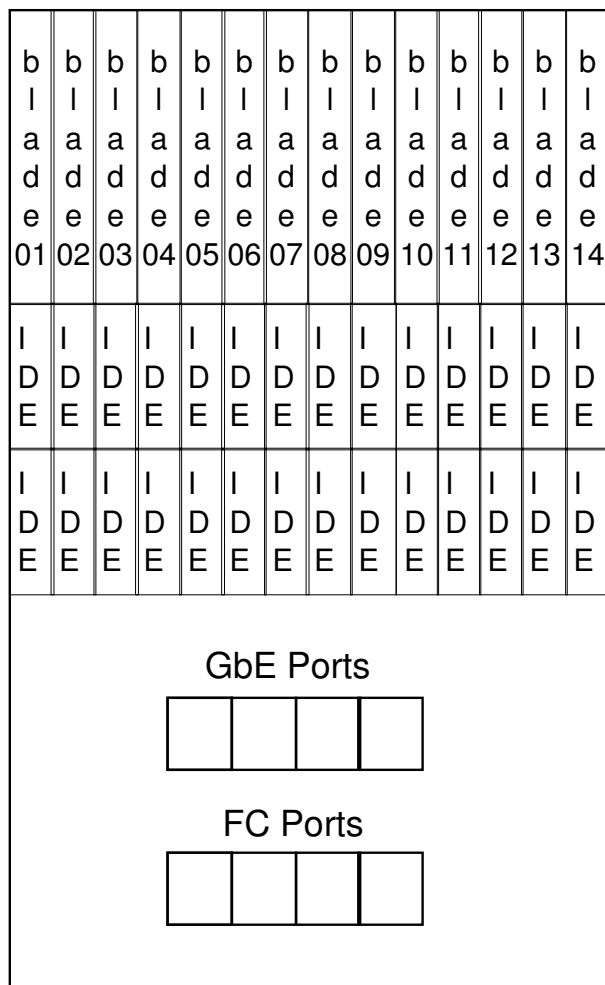
COMMENTS:

- This configuration is **not** recommended since it presents a single point of failure risk.
- While less than optimal, this risk can be eliminated using GPFS replication.



Blades Using Internal IDE Drives

Blade Center



Blade Specs

- ▶ IBM HS20
- ▶ dual Xeon @ 2.8 GHz
- ▶ 4 GB RAM
- ▶ 2 IDE drives (40 GB at 5400 RPM)

GPFS Configuration Parameters

- ▶ Block size = 256 KB
- ▶ Pagepool = 64 MB
- ▶ NSD configuration (via internal GbE network)
- ▶ file system scope limited to single chassis

Disk System Specs

- ▶ internal IDE drives (see Blade Specs)
- ▶ 1 full disk and a 30 GB partition from the other drive in each blade is used for the file system
- ▶ 28 LUNs
- ▶ JBOD

Benchmark Results

- ▶ **bonnie** (see <http://linux.maruhn.com/sec/bonnie.html>)
- ▶ single task
 - read rate = **80 MB/s**
- ▶ aggregate (1 task per blade)
 - read rate = **560 MB/s** (i.e., 40 MB/s per blade)
- ▶ baseline test (read from a single local disk using ext2)
 - read rate = **30 MB/s**

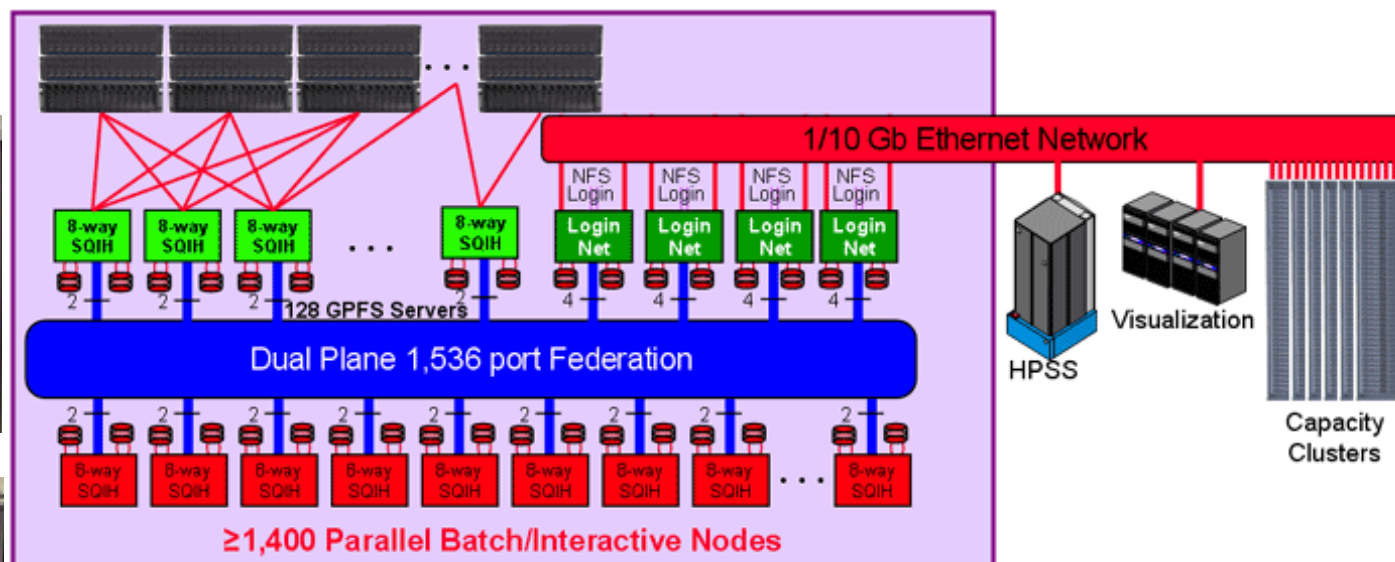
COMMENTS:

- ▶ This configuration is **not** recommended since it presents a single point of failure risk.
- ▶ While less than optimal, this risk can be eliminated using GPFS replication.

COMMENT: the primary application is blast



Heterogenous Cluster



System

- At least 1,400 parallel batch/interactive nodes
- 4 Login/network nodes from 2 SQH
- Clustered I/O with 128 SQIH for global I/O
- Dual plane 1,536 port Federation switch
- External networking
 - Login/network nodes for login/NFS/PFTP
 - All external networking is 1-10Gb/s Ethernet

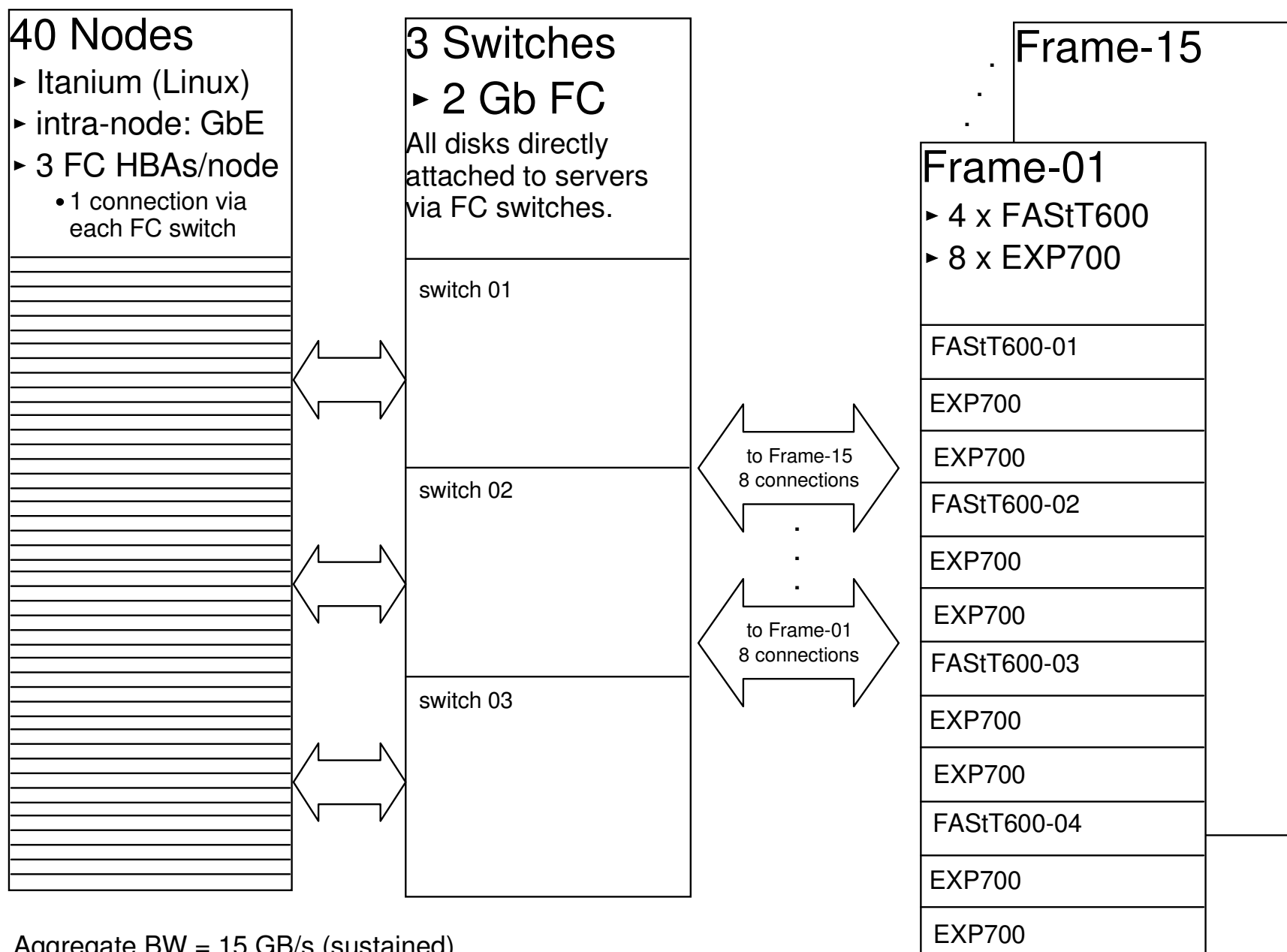
Programming/Usage Model

- Application launch over all compute nodes
- 1 MPI task/CPU and Shared Memory, full 64b support
- Scalable MPI (MPI_allreduce, buffer space) to 8,192 tasks
- Likely usage
 - multiple MPI tasks/node with 2-4 OpenMP/MPI task
- Single STDIO interface
- Parallel I/O to single file, multiple serial I/O (1 file/MPI task)

- 1536-node, 100 TF pSeries cluster
- 2 PB GPFS file system (one mount point)
- 500 RAID controller pairs, 11000 disk drives
- 126 GB/s parallel I/O measured to a single file (134GB/s to multiple files)
- 125 NSD/VSD nodes



SAN Organization with Excellent Scaling



Aggregate BW = 15 GB/s (sustained)

goto: http://www.sdsc.edu/Press/2004/11/111204_SC04.html



SAN Organization with Excellent Scaling

- **40 Linux Nodes (SDSC Booth)**

- **3 FC HBAs per Node**

- **15 Storage frames**

- **60 FAStT600s**

- **2520 disks**

- **240 LUNs**

- **8+P**

- **4 LUNs per FAStT600**

- **73 GB/disk @ 15 Krpm**

- **Sustained Aggregate Rate**

 - 15 GB/s

 - 380 MB/s per node

 - 256 MB/s per FAStT600



Written Exercise #1

Suppose you have been asked to design the storage subsystem. The cluster will be running Linux with 256 4-way compute nodes (i386 or x86-64). The application job mix will be varied. The message passing traffic will vary from light to moderate (bursts of large messages that are latency tolerant) to heavy (numerous small packet messages that are latency *intolerant* for the duration of the job, or jobs that will have large packet transfers upon startup and close to termination that are latency tolerant); the customer believes that node message passing BW will be at most 50 to 80 MB/s. The storage I/O will also be variable. Some jobs will require lots of BW at the beginning and end of the jobs using a streaming access pattern (large records, sequential access), others will required sustained, moderate access over the life of the job using a streaming pattern, and 1 job will require sustained, but light access over the life of the job, but the access pattern will be small records, irregularly distributed over the seek offset space. The jobs on the cluster are parallel. Finally, there are about 200 users with Windows or Linux based PCs in their office that must access this cluster's file system. Typically, the aggregate file system BW for the cluster will be in the neighborhood of 1 GB/s, though aggregate burst rates could be as high as 2 GB/s. Individual nodes must be able to sustain storage I/O BW up to 60+ MB/s, though more typical node rates are less than 5 MB/s. The file system must start at 50 TB, but be expandable to 100 TB in the future.

Design the cluster by specifying

- network (GbE, Myrinet, IB or mixed) and its topology
- storage nodes
- disk and controllers

What additional information do you need in order to make a better specification?



The dog ate my homework!



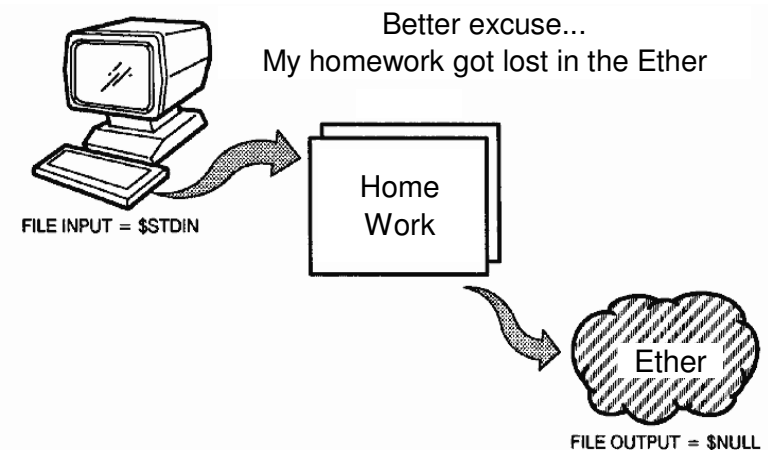
Written Exercise #2

Suppose you have been asked to design a new storage subsystem to be shared by 2 clusters. The first cluster is a new one that consists of 32 P6-p575 nodes (32 cores per node with 64 GB or RAM) using IB (4x DDR) for the LAN. The other cluster is a legacy system that was designed for written exercise #1. The new storage subsystem needs to be accessible by both clusters though it will be primarily used by the new pSeries cluster (85% usage) as a scratch file system; the legacy cluster accesses on the new file system will mostly be reads. This new pSeries cluster must also be able to access the storage subsystem in the legacy cluster, though it will only account for 15% of the storage work load on that storage subsystem with most of the accesses being reads. The job mix is varied on the new pSeries cluster; it will be used for parallel jobs with heavy message passing requirements, but the storage I/O access pattern will be largely streaming oriented (large records sequentially accessed in large files), but there will be one job flow that must access many small files (2K to 256K with the average being 8K). The small file workload will account for 25% of the overall workload on the pSeries cluster. The new storage system must be able to support high data rates (8 GB/s) for the streaming work load and high IOP rates for the small file work load (80,000 files per second). The capacity of the new storage system must be 250 TB.

Design the cluster by specifying

- storage network (GbE, Myrinet, IB or mixed) and its
- storage nodes
- disk and controllers

What additional information do you need in order to make a better specification?





11. GPFS System Administration

Let's take a look at some selected sysadm details. Much of this this information is also relevant to programmers.

This discussion is not intended to be exhaustive; rather it is intended to provide general guidance and examples (i.e., "give me an example and I will figure out how to do it").

The emphasis is more on concept than syntax. Nor are all options explained. See manuals for further details.

COMMENT:

Unless stated otherwise, **specific examples are based on GPFS 3.1**. *For the most part*, there is little or no change between GPFS versions 2.3, 3.1 and 3.2 for these commands.



Where to Find Things in GPFS

■ Some useful GPFS directories

- /usr/lpp/mmfs
 - /bin... commands (binary and scripts)
 - most GPFS commands begin with "mm"
 - /gpfsdocs... pdf and html versions of basic GPFS documents
 - /include... include files for GPFS specific APIs, etc.
 - /lib... GPFS libraries (e.g., libgpfs.a, libdmapi.a)
 - /samples... sample scripts, benchmark codes, etc.
- /var/adm/ras
 - error logs
 - files... mmfs.log.<time stamp>.<hostname> (new log every time GPFS restarted)
 - links... mmfs.log.latest, mmfs.log.previous
- /tmp/mmfs
 - used for GPFS dumps
 - sysadm must create this directory
 - see **mmchconfig**
- /var/mmfs
 - GPFS configuration files
- same directory structure for both AIX and Linux systems

■ Today's trivia question...

Question: What does mmfs stand for?

Answer: Multi-Media File System... predecessor to GPFS in the research lab



GPFS Information and Downloads from the Web

■ GPFS FAQs (for versions 3.1, 3.2, 3.3)

- <http://publib.boulder.ibm.com/clresctr/library/gpfsclustersfaq.html>
- In addition to FAQs, this web page has links to technical documentation for
 - CSM
 - GPFS
 - RSCT
 - *etc.*

■ GPFS upgrades

- <https://www14.software.ibm.com/webapp/set2/sas/f/gpfs/download/home.html>
- Once the base version of GPFS is installed, upgrades can be freely downloaded and installed for AIX and Linux



Comments on Selected GPFS Manuals

■ GPFS Documentation

- *Concepts, Planning and Installation Guide*

- One of the most helpful manuals on GPFS... it provides an excellent conceptual overview of GPFS.
- If this were a university class, this manual would be your assigned reading. :->

- *Administration and Programming Reference* and *Advanced Administration Guide*

- Documents GPFS related administrative procedures and commands as well as an API guide for GPFS extensions to the POSIX API. The command reference is identical to man pages.

- *Problem Determination Guide*

- Many times, GPFS error messages in the mmfs.log files have an error number. You can generally find these referenced in this guide with a brief explanation regarding the cause of the message. They will often point to likely earlier error messages helping you to find the cause of the problem as opposed to its symptom.

- *DMAPI Guide*

- Documentation available online at...

- http://publib.boulder.ibm.com/clresctr/windows/public/gpfsbooks.html#aix_rsctpd22wo

- Available with the GPFS SW distribution in `/usr/lpp/mmfs/gpfsdocs`

- note to sysadm's... Be sure to install this directory! Also install the man pages!

■ IBM Redbooks and Redpapers

- <http://www.redbooks.ibm.com/>... do a search on GPFS



GPFS provides a number of commands to list parameter settings, configuration components and other things.

I call these the "mmls" or "mm list" commands.

COMMENT:

By default, nearly all of the mm commands require root authority to execute. However, many sysadm's reset the permissions on mmls commands to allow programmers and others to execute them as they are very useful for the purposes of problem determination and debugging.



Selected "mmls" Commands

■ mmlsfs <file system device name>

```
[root@gpfs_node1 gpfs_install]# mmlsfs gpfs1      <-- you could also type "mmlsfs /dev/gpfs1"
```

```
flag value          description
```

```
-----
-s  roundRobin      Stripe method
-f  131072          Minimum fragment size in bytes
-i  512             Inode size in bytes
-I  32768           Indirect block size in bytes
-m  2              Default number of metadata replicas
-M  2              Maximum number of metadata replicas
-r  1              Default number of data replicas
-R  2              Maximum number of data replicas
-j  scatter         Block allocation type
-D  posix           File locking semantics in effect
-k  posix           ACL semantics in effect
-a  1048576         Estimated average file size
-n  32             Estimated number of nodes that will mount file system
-B  4194304         Block size
-Q  none            Quotas enforced
    none           Default quotas enabled
-F  91798862        Maximum number of inodes
-V  9.03            File system version. Highest supported version: 9.03
-u  yes             Support for large LUNs?
-z  no             Is DMAPI enabled?
-E  yes            Exact mtime mount option
-S  no             Suppress atime mount option
-K  whenpossible    Strict replica allocation option
-P  system          Disk storage pools in file system
-d  nsd_meta1;nsd_meta2;nsd_meta3;nsd_meta4;nsd_lun1;nsd_lun2;nsd_lun3;nsd_lun4;nsd_lun5;
    nsd_lun6;nsd_lun7;nsd_lun8;nsd_lun9;nsd_lun10;nsd_lun11;nsd_lun12;
-A  yes            Automatic mount option
-o  none            Additional mount options
-T  /gpfs1          Default mount point
```



Selected "mmls" Commands

■ mmlsconfig

```
[root@gpfs_node1 gpfs_install]# mmlsconfig
Configuration data for cluster gpfs_node1:
```

```
-----
clusterName gpfs_node1
clusterId 13882392465829736019
clusterType lc
autoload yes
useDiskLease yes
maxFeatureLevelAllowed 913
maxblocksize 4096k
pagepool 256m
maxMBpS 2000
[gpfs1]
takeOverSdrServ yes
```

```
File systems in cluster gpfs_node1:
-----
/dev/gpfs1
```

COMMENTS:

- Lists configuration parameters applying to the cluster.
- Generally only lists configuration parameters that have been changed.

Unless explicitly specified, the cluster name is the primary cluster configuration server hostname.

■ An undocumented alternative to mmlsconfig that provides more info

- mmfsadm
 - 2 modes: console mode or command line mode
- example using command line mode
 - mmfsadm dump config
 - list all configuration parameters (including defaults)
 - more information than mmlsconfig



Selected "mmls" Commands

mmlscluster

```
[root@gpfs_node1 gpfs_install]# mmlscluster
```

```
GPFS cluster information
```

```
=====
```

```
GPFS cluster name:      gpfs_node1
GPFS cluster id:        13882392465829736019
GPFS UID domain:        gpfs_node1
Remote shell command:   /usr/bin/ssh
Remote file copy command: /usr/bin/scp
```

```
GPFS cluster configuration servers:
```

```
-----
```

```
Primary server:   gpfs_node1
Secondary server: gpfs_node2
```

Node	Daemon node name	IP address	Admin node name	Designation

1	gpfs_node1	192.168.42.101	gpfs_node1	quorum
2	gpfs_node2	192.168.42.102	gpfs_node2	quorum
3	gpfs_node3	192.168.42.103	gpfs_node3	quorum
4	gpfs_node4	192.168.42.104	gpfs_node4	quorum



Selected "mmls" Commands

■ mmgetstate -a

```
[root@gpfs_node2 gpfs_install]# mmgetstate -a
```

Node number	Node name	GPFS state
1	gpfs_node1	arbitrating
2	gpfs_node2	active
3	gpfs_node3	active
4	gpfs_node4	down

If the mmgetstate command is issued too soon after the mmstartup command, some nodes will be listed with an "arbitrating" state, meaning only that the daemon has not had enough time to start. This is common in very large clusters. (It may take a couple minutes for all daemons to start in this case).

If the state is "down", that generally means that there is a problem or that the daemon has not yet been started.



Selected "mmls" Commands

■ mmlsdisk <file system device name>

- display current configuration and state of the disks in a file system

```
[root@gpfs_node1 gpfs_install]# mmlsdisk gpfs1
```

disk name	driver type	sector size	failure group	holds metadata	holds data	status	availability	storage pool
nsd_lun1	nsd	512	4001	no	yes	ready	up	system
nsd_lun2	nsd	512	4001	no	yes	ready	up	system
nsd_lun3	nsd	512	4001	no	yes	ready	up	system
nsd_lun4	nsd	512	4002	no	yes	ready	up	system

■ status

- **suspended**: indicates that data is to be migrated off this disk
- **being emptied**: transitional status in effect while a disk deletion is pending
- **replacing**: transitional status in effect for old disk while replacement is pending
- **replacement**: transitional status in effect for new disk while replacement is pending

■ availability

- **up**: disk is available to GPFS for normal read and write operations
- **down**: no read and write operations can be performed on this disk
- **recovering**: an intermediate state for disks coming up
 - during this state GPFS verifies and corrects data
 - read operations can be performed, but write operations cannot
- **unrecovered**: the disks was not successfully brought up



Other Selected "mmls" Commands

■ **mmlsattr <file name>**

- query file attributes

■ **mmlsmgr <file system device name(s)>**

- display which node is the file system manager for the specified file systems

■ **mmlsnsd**

- display current NSD information in the GPFS cluster
- -X
 - cool new option for GPFS 3.2
 - Maps the NSD name to its disk device name in /dev on the local node and, if applicable, on the NSD server nodes.
 - Using the -X option is a slow operation and is recommended only for problem determination.

```
[root]# mmlsnsd -X -d "hd3n97;sdfnsd;hd5n98"
```

Disk name	NSD volume ID	Device	Devtype	Node name	Remarks
hd3n97	0972846145C8E927	/dev/hdisk3	hdisk	c5n97g.ppd.pok.ibm.com	server node,pr=no
hd3n97	0972846145C8E927	/dev/hdisk3	hdisk	c5n98g.ppd.pok.ibm.com	server node,pr=no
hd5n98	0972846245EB501C	/dev/hdisk5	hdisk	c5n97g.ppd.pok.ibm.com	server node,pr=no
hd5n98	0972846245EB501C	/dev/hdisk5	hdisk	c5n98g.ppd.pok.ibm.com	server node,pr=no
sdfnsd	0972845E45F02E81	/dev/sdf	generic	c5n94g.ppd.pok.ibm.com	server node
sdfnsd	0972845E45F02E81	/dev/sdm	generic	c5n96g.ppd.pok.ibm.com	server node

AIX

LINUX



GPFS provides a number of commands needed to create the file system.

These commands of necessity require root authority to execute.



Selected "mm" Commands

■ mmcrcluster

- create a GPFS cluster
- parameters and options
 - -n specifies a file with a list of node descriptors - NodeName:NodeDesignations
 - NodeName is the IP address or hostname
 - NodeDesignators specifies client or server, quorum or non-quorum
 - -p primary GPFS cluster configuration server node
 - -s secondary GPFS cluster configuration server node
 - -R specify remote file copy command (e.g., rcp or scp)
 - -r specify remote shell command (e.g., rsh or ssh)
 - The remote copy and remote shell commands must adhere to the same syntax format as the rcp and rsh commands, but may implement an alternate authentication mechanism.



Selected "mm" Commands

■ **mmstartup -a**

- manually start up the **mmfsd** daemons on all nodes in cluster
 - setting autoload=yes via **mmchconfig** command will automatically launch **mmfsd**
 - if mmfsd can not start automatically, you will see **runmmfs** running and a lot of messages in /var/adm/ras/mmfs.log.latest

■ **mmshutdown -a**

- unmount all GPFS file systems and shut down **mmfsd** daemons on all nodes
- always do this before rebooting nodes if possible

■ **If you do not need to do this on all nodes, the -W or -w parameters will allow you specify which nodes in the cluster to start up/shut down mmfsd.**



Selected "mm" Commands

■ mmcrnsd (version 3.1)

- Creates and globally names Network Shared Disks for use by GPFS.
- mmfsd daemon must be running to execute mmcrnsd (i.e., do mmstartup first)
- mmcrnsd -F disk.lst
 - (version 3.1) disk.lst is a "disk descriptor file whose entries are in the format
 - DiskName:PrimaryServer:BackupServer:DiskUsage:FailureGroup:DesiredName:StoragePool
 - ◆ **DiskName:** The disk name as it appears in /dev
 - ◆ **PrimaryServer:** The name of the primary NSD server node.
 - ◆ **BackupServer:** The name of backup NSD server node. Specifying primary and secondary servers is recommended even in SAN mode.
 - ◆ **DiskUsage:** dataAndMetadata (default) or dataOnly or metadataOnly
 - ◆ **FailureGroup:** GPFS uses this information during data and metadata placement to assure that no two replicas of the same block are written in such a way as to become unavailable due to a single failure. All disks that are attached to the same adapter or NSD server should be placed in the same failure group. Applies only to GPFS in non-SAN mode.
 - ◆ **DesiredName:** Specify the name you desire for the NSD to be created. Default format... **gpfs<integer>nsd**
 - ◆ **StoragePool:** Specify name of the storage pool that the NSD is assigned to; this parameter is used by mmcrfs command
 - disk.lst is modified for use as the input file to the mmcrfs command
- -v no
 - verify the disk is not already formatted as an NSD; a value of no means do NOT verify



Selected "mm" Commands

■ mmcrnsd (version 3.2 change)

- Creates and globally names Network Shared Disks for use by GPFS.
- mmfsd daemon must be running to execute mmcrnsd (i.e., do mmstartup first)
- mmcrnsd -F disk.lst
 - disk.lst is a "disk descriptor" file whose entries are in the format
 - DiskName:ServerList::DiskUsage:FailureGroup:DesiredName:StoragePool
 - ♦ **DiskName:** The disk name as it appears in /dev
 - ♦ **ServerList:** Is a comma separated list of NSD server nodes. You may specify up to eight NSD servers in this list. The defined NSD will preferentially use the first server on the list. If the first server is not available, the NSD will use the next available server on the list.
 - ♦ **DiskUsage:** dataAndMetadata (default) or dataOnly or metadataOnly
 - ♦ **FailureGroup:** GPFS uses this information during data and metadata placement to assure that no two replicas of the same block are written in such a way as to become unavailable due to a single failure. All disks that are attached to the same adapter or NSD server should be placed in the same failure group. Applies only to GPFS in non-SAN mode.
 - ♦ **DesiredName:** Specify the name you desire for the NSD to be created. Default format... **gpfs<integer>nsd**
 - ♦ **StoragePool:** Specify name of the storage pool that the NSD is assigned to; this parameter is used by mmcrfs command
 - disk.lst is modified for use as the input file to the mmcrfs command
- -v no
 - verify the disk is not already formatted as an NSD; a value of no means do NOT verify

LOOK::

2 colons



Selected "mm" Commands

Disk Descriptor Files Are **Both** Input and Output Files

WARNING: The mmcrnsd command modifies the disk.lst file. Therefore, make a backup copy of it.

```
[root@gpfs1 gpfs_install]# cat disk.lst                                     Version 3.1 Example
/dev/sdb:gpfs1:gpfs2:::nsd_lun1:
/dev/sdc:gpfs1:gpfs2:::nsd_lun2:
/dev/sdd:gpfs2:gpfs1:::nsd_lun3:
/dev/sde:gpfs2:gpfs1:::nsd_lun4:
[root@gpfs_node1 gpfs_install]# mmcrnsd -F disk.lst -v no
. . . . .
[root@gpfs_node1 gpfs_install]# cat disk.lst
# /dev/sdb:gpfs1:gpfs2:::nsd_lun1:
nsd_lun1:::dataAndMetadata:4001::
# /dev/sdc:gpfs1:gpfs2:::nsd_lun2:
nsd_lun2:::dataAndMetadata:4001::
# /dev/sdd:gpfs2:gpfs1:::nsd_lun3:
nsd_lun3:::dataAndMetadata:4002::
# /dev/sde:gpfs2:gpfs1:::nsd_lun4:
nsd_lun4:::dataAndMetadata:4002::
```

NOTE: This is a Linux example. Under AIX disk names are generally of the form hdisk<x>



Selected "mm" Commands

■ mmcrvsd

- only useful with pSeries/AIX systems using the HPS (*i.e.*, "federation") or SP switch (*i.e.*, "colony"); it replaces IP with a more efficient protocol
 - GPFS no longer requires RSCT, but RSCT must be installed to use this protocol
- syntax is similar to mmcrnsd
 - see *Administration and Programming Reference* for details (n.b., it has several more optional parameters)
- The mmcrvsd output disk descriptor file can no longer be used as input to the mmcrfs command to build the file system. It is necessary to create NSDs (via the mmcrnsd command) using the output disk descriptor file from the mmcrvsd command after creating the VSDs.

■ mmcrlv

- no longer required and no longer exists
 - If you do create LVs manually using crlv, GPFS will **not** configure properly!



Selected "mm" Commands

■ **mmcrfs <mountpoint> <device name> <options>**

■ Create a GPFS file system

- -F specifies a file containing a list of disk descriptors (one per line)
 - this is the output file from **mmcrnsd**
- -A yes -> mount after starting mmfsd, no -> manually mount, automount -> mount at first use
 - default is yes
- -B block size (16K, 64K, 256K (default), 512K, 1024K, 2048K, 4096K)
 - If you choose a block size larger than 256 KB, you must run mmchconfig to change the value of maxblocksize to a value at least as large as BlockSize.
- -E specifies whether or not to report exact mtime values
- -m default number of copies (1 or 2) of i-nodes and indirect blocks for a file
- -M default max number of copies of inodes, directories, indirect blocks for a file
- -n estimated number of nodes that will mount the file system
- -N max number of files in the file system (default = sizeof(file system)/1M)
- -Q activate quotas when the file system is mounted (default = NO)
- -r default number of copies of each data block for a file
- -R default maximum number of copies of data blocks for a file
- -S suppress the periodic updating of the value of atime
- -v verify that specified disks do not belong to an existing file system
- -z enable or disable DMAP1 on the file system (default = no)
- -D specify nfs4 to allow "deny-write open lock" to block writes for NFS V4 exported file systems
default=posix
- -k specify the authorization protocol; the options are <posix | nfs4 | all>

■ **Typical example**

- mmcrfs /fs fs -F disk.lst -A yes -B 1024k -v no



GPFS provides a number of commands to change configuration and file system parameters after being initially set.

I call these the "mmch" or "mm change" commands.

There are some GPFS parameters which are initially set only by default; the only way to modify their value is using the appropriate mmch command.

n.b., There are restrictions regarding changes that can be made to many of these parameters; be sure to consult the *Concepts, Planning and Installation Guide* for tables outlining what parameters can be changed and under which conditions they can be changed. See the *Administration and Programming Reference* manual for further parameter details.



Selected "mmch" Commands

■ mmchconfig <attributes> <parameters>

- change GPFS default configuration attributes
- parameters
 - node list
 - specify node file using -n <node file> or a comma separated list of nodes on the command line
 - the default is all nodes in the cluster
 - -i changes take effect immediately and are permanent
 - -l changes take effect immediately but do not persist after GPFS is restarted
 - parameters do **not** apply to all attributes; carefully review *Administration and Programming Guide* for details
- attributes (selection of the more common or nettlesome ones)
 - autoload: Start mmfsd automatically when nodes are rebooted. Valid values are yes or no.
 - dataStructureDump: the default is /tmp/mmfs
 - do not use a GPFS directory (it may not be available)
 - warning: files can be large (200 MB or more)... be sure to delete them when done
 - designation: explicitly designate client, manager, quorum, or nonquorum nodes
 - maxblocksize: default is 1024K; n.b., mmcrfs blocksize (-B) can not exceed this.
 - maxMBpS (data rate estimate (MB/s) on how much data can be transferred in or out of 1 node)
 - The value is used in calculating the amount of IO that can be done to effectively prefetch data for readers and write-behind data from writers. By lowering this value, you can artificially limit how much IO one node can put on all of the disk servers. This is useful in environments in which a large number of nodes can overrun a few storage servers.
 - **The default is 150 MB/s which can severely limit performance on HPS ("federation") based systems.**
 - pagepool: minimum = 4M, default = 64M, max = 50% of memory
 - Unnecessarily large pagepools result in a law of diminishing returns.
 - maxFilesToCache: values in range 1 to 100,000
 - size = 2.5 KB * maxFilesToCache
 - maxStatCache: values in range 1 to 100,000
 - size = 176 B * maxStatCache
 - tiebreakerDisks: to use this feature, provide a list of disk names (i.e., there NSD name)



Selected "mmch" Commands

■ **mmchfs <device name> <options>**

- Change attributes of an existing GPFS file system
 - Files created after issuing the mmchfs assume the new attributes. Existing files are not affected.
- Options the same as for mmcrfs
 - -A, -E, -D, -k, -m, -Q, -r, -S, -z
- Options not available under mmcrfs
 - -F changes the max number of files that can be created
 - -T change mountpoint of file system starting at the next mount of the file system.
 - -V change the file system format to the latest format supported by this version
 - -W assign a new device name to the file system (i.e., change the name in /dev)
- Options available under mmcrfs, but **not** available under mmchfs
 - -B, -M, -n, -N, -R, -v
 - changing these parameters requires rebuilding the FS
- Carefully review the following documents for more details
 - *Concepts, Planning and Installation Guide*
 - *Administration and Programming Reference*



Selected "mmch" Commands

■ mmchlicense





GPFS is provides dynamic means to add and/or remove many components. This allows the sysadm a convient means to grow the current infrastructure or to re-allocate resources to other places by deleting them from an existing system.



mmadddisk and mmdeldisk

GPFS provides the means to dynamically add and remove disks from a GPFS cluster.

- ▶ `mmadddisk <device> -F disk.lst -r -v {yes | no}`
- ▶ `mmdeldisk <device> -F disk.lst -r -c`
 - <device> is the GPFS device in /dev
 - disk.lst entries in the form `DiskName:::DiskUsage:FailureGroup`
 - see documentation for details
 - v: verify that disk does not already belong to file system
 - if using the yes option and if the disk has had a file system on it before, GPFS will not add it to file system
 - r: rebalance disks (see also -N parameter in documentation)
 - this can take a lot of overhead for a dynamic environment and is probably unnecessary
- ▶ notes and caveats
 - mmadddisk: if a NSD does not exist for the disk, must first create it
 - mmdeldisk: if disk is bad and cannot be read, use -c



mmaddnode and mmdelnode

GPFS provides the means to dynamically add and remove nodes from a GPFS cluster.

- ▶ **mmaddnode -n node.lst**
 - add nodes to an existing cluster and create mount points and device entries on the new nodes
 - under some circumstances (e.g., re-installing node), it may be necessary copy the mmsdrfs file to the new nodes (*n.b.*, get copy from primary cluster server... see mmlscluster)
- ▶ **mmdelnode -n node.lst**
 - remove nodes from an existing cluster
 - notes and caveats
 - primary/secordary cluster servers
 - primary/secordary NSD servers
 - must first unmount GPFS file system
 - use caution when deleting quorum nodes



There are a number of ways to measure file system performance. There are some very simple techniques that provide useful insight while there are other more elaborate alternatives. Some are common and some are unique to GPFS.



Measuring Performance

See Hennessy & Patterson, *Computer Architecture: A Quantitative Approach*, 2nd ed., Morgan Kaufmann, 1996 (pp. 20-21)

Benchmarking

- Real programs with instrumentation
- Synthetic benchmarks
 - good ones
 - IOR, xdd
 - not recommended for HPC
 - bonnie, iозone
 - GPFS specific benchmarks
 - gpfsperf, ibm.v4b
- Kernels
 - nsdperf
 - very few of these exist for measuring file system performance.
- Toy benchmarks
 - dd, home grown varieties

Measurement Tools

- System tools (iostat, nmon)
- GPFS commands (mmpmon, nsdperf)
- Controller tools



Measuring Bandwidth

iostat

Measuring I/O time within the application using timing functions like `rtc()` or `gettimeofday()` are useful from a job perspective, but do not accurately measure actual I/O rates (e.g., they can overlook locking delays, include PVM message passing overhead, ignore variance).

The **iostat** command shows actual disk activity (this is the AIX version).

`iostat <time interval> <number of samples>`

Use `dsh` to collect from multiple nodes.

`export WCOLL=/wcoll`

`dsh -a`

`> iostat 10 360 > `hostname -s`.iostat`

rsh can also be used





Measuring Bandwidth

iostat

```
flash008> iostat 10 360
```

tty:	tin	tout	avg-cpu:	% user	% sys	% idle	% iowait
	0.0	0.0		49.8	5.2	44.9	0.1

hdisk0, hdisk1
▶ local JFS directory

Disks:	% tm_act	Kbps	tps	Kb_read	Kb_wrtn
hdisk1	0.0	0.0	0.0	0	0
hdisk0	0.2	3.3	0.4	5537695	4424062
hdisk3	16.5	940.8	19.3	2119811596	709528896
hdisk5	0.0	0.0	0.0	5262	0
hdisk2	0.0	0.0	0.0	5262	0
hdisk4	16.3	249.5	19.4	39842779	710476158

hdisk3, hdisk4
▶ mounted locally on this
VSD server node

hdisk2, hdisk5
▶ mounted locally on this
VSD server node **only in
failover mode**

tty:	tin	tout	avg-cpu:	% user	% sys	% idle	% iowait
	0.0	0.0		0.3	8.1	91.6	0.0

Disks:	% tm_act	Kbps	tps	Kb_read	Kb_wrtn
hdisk1	0.0	0.0	0.0	0	0
hdisk0	0.0	0.0	0.0	0	0
hdisk3	67.6	10041.6	39.6	0	100416
hdisk5	0.0	0.0	0.0	0	0
hdisk2	0.0	0.0	0.0	0	0
hdisk4	67.4	10036.8	39.4	0	100368

tty:	tin	tout	avg-cpu:	% user	% sys	% idle	% iowait
	0.0	0.0		0.1	3.4	96.5	0.0

Disks:	% tm_act	Kbps	tps	Kb_read	Kb_wrtn
hdisk1	0.0	0.0	0.0	0	0
hdisk0	0.0	0.0	0.0	0	0
hdisk3	56.7	8111.9	32.0	0	81200
hdisk5	0.0	0.0	0.0	0	0
hdisk2	0.0	0.0	0.0	0	0
hdisk4	57.0	8062.3	31.9	0	80704



Measuring Bandwidth

iostat

Meaning of iostat columns.

- %usr - percent application CPU time
- %sys - percent of kernel CPU time
- %idle - percent of CPU idle time during which there were **no** outstanding disk I/O requests
- %iowait - percent of CPU idle time during which there were outstanding disk I/O requests
- %tm_act - percent of time that the hdisk was active (i.e., bandwidth disk utilization)
- Kbps - volume of data read and/or written to the hdisk in kilobytes per second
- tps - transfers (i.e., I/O requests) per second to the hdisk
- Kb_read - total data read from the given hdisk over the last time interval in KB
- Kb_wrtn - total data written to the given hdisk over the last time interval in KB

n.b., $\text{Kbps} = (\text{Kb_read} + \text{Kb_wrtn}) / \text{time_interval}$



Measuring Latency

mmpmon

mmpmon

- ▶ Performance monitoring tool
- ▶ It can be used to
 - display I/O statistics per mounted file system for each file system on a node
 - display aggregate I/O statistics from multiple nodes
 - see the nlist option
 - display latency measurements in a histogram format
- ▶ Command syntax for with output in human readable form
 - `mmpmon -i command_file`
 - commands specifying what is displayed and how they are displayed is done by a command file
 - see chapter 5 in the GPFS 3.1 Advanced Administration Guide for details
- ▶ Miscellaneous
 - Starting with version 3.1, it can collect statistics from multiple nodes
 - It requires root access
 - Up to 5 instances of mmpmon can be run on 1 node at one time
 - It is effective, but still a little cumbersome to use



Measuring Latency

mmpmon

Latency Histogram Example

```
> cat command_file                                     <-- be sure to login as root while an application is running
rhist on                                              <-- enable histograms (n.b., rhist off disables histograms)
> mmpmon -i command_file
mmpmon node 192.168.1.2 name gandalf rhist on OK
> cat command_file
rhist nr 64k;256k;1m; 1;10;30;100                  <-- this is in the form rhist nr <packet sizes> <space> <time intervals>
> mmpmon -i command_file                               time is specified in units of milliseconds
mmpmon node 192.168.1.2 name gandalf rhist nr 64k;256k;1m; 1;10;30;100 OK
> cat command file
rhist s                                              <-- display histogram data for non-empty bins
> mmpmon -i command_file
mmpmon node 10.10.111.4 name c4f01plg rhist s OK read timestamp 1159397896/443573
size range           0 to           65536 count           512
  latency range      0.0 to           1.0 count           511
  latency range      30.1 to          100.0 count             1
size range      262145 to      1048576 count          4096
  latency range      0.0 to           1.0 count          3945
  latency range      1.1 to          10.0 count             27
  latency range      10.1 to          30.0 count             27
  latency range      30.1 to          100.0 count             43
  latency range      100.1 to           0.0 count             54
mmpmon node 10.10.111.4 name c4f01plg rhist s OK write timestamp 1159397896/443616
size range      262145 to      1048576 count          2048
  latency range      0.0 to           1.0 count          1636
  latency range      1.1 to          10.0 count           214
  latency range      10.1 to          30.0 count           162
  latency range      30.1 to          100.0 count            34
  latency range      100.1 to           0.0 count             2
c4f01plm:/u/cmisp/home/padenr/ibm.v3g/ibm_mpi/mmpmon >
```



Measuring Latency Multi-node mmpmon Usage

Latency Histogram Example Across Multiple Nodes

When using the nlist option, you must combine it within the command file of the associated command each time for the command

```
> cat command_file_1
nlist new n01 n02 n03 n04
rhist nr 64k;256k;1m; 1;10;30;100      <-- apply rhist nr to nodes n01 n02 n03 n04
rhist on                                <-- apply rhist on to nodes n01 n02 n03 n04
> cat command_file_2
nlist new n01 n02 n03 n04
rhist s                                  <-- apply rhist s to nodes n01 n02 n03 n04
> cat command_file_3
nlist new n01 n02 n03 n04
rhist off                                <-- apply rhist off to nodes n01 n02 n03 n04
```



Measuring Bandwidth

nsdperf





Measuring Bandwidth

Code Instrumentation





Problem Determination Tools





Other Topics



■ Waiters (i.e., waiting threads)

- see p. 48, 89, of *Problem Determination Guide*

■ mmfsadm

- see p. 9, of *Problem Determination Guide*
 - **mmfsdam dump config**
 - **mmfsadm dump all**
 - **WARNING:** Creates a file up to 100's of MB in size
 - **mmfsadm dump tscomm**
 - list of GPFS sockets and their status
 - **mmfsadm dump threads**
 - active thread tracebacks (useful to find threads that are looping instead of waiting)
 - **mmfsadm cleanup**
 - alternative to mmshutdown; designed to recycle mmfsd on a node without hanging

■ **mmddsh -N { Node[,Node,...] | Nodefile | nodeclass} { command }**

- **nodeclass**
 - all, clientnodes, managernodes, mount, nonquorumnodes, nsdnodes, quorumnodes

■ gpfs.snap

- see pp. 6-8, of *Problem Determination Guide*



Creating Trace Reports under Loss of Quorum Condition

- **If a node loses GPFS Quorum, it will execute the following user defined script**
 - /var/mmfs/etc/mmQuorumLossExit
 - this script must exist on all nodes in the cluster
 - for example...

Launch mmtrace on all nodes in cluster.

When a node loses quorum, GPFS executes the following script which recycles the tracing (there-by generating a trace report). The tracefile is called lxtrace.trc.<hostname>.

```
> cat mmQuorumLossExit
echo `hostname` LOST GPFS QUORUM
echo RECYCLING mmtrace
date
/usr/lpp/mmfs/bin/mmtrace
```

COMMENT:

A GPFS trace is fixed size (default is 16M... set environment variable TRCFILESIZE to change size). Trace data wraps around once it hits the end of the file. The time duration represented by the tracefile is proportional to its size.



GPFS Security

GPFS Security Model

GPFS administration is based on an environment of trust; this facilitates performance and convenience. It relies on external measures to insure security.

- ▶ version 3.3: implicitly define the scope of trust
- ▶ prior to version 3.3: scope of trust spans the cluster

Security vs. performance and convenience...
a classic example of being caught between a
rock and a hard place





GPFS Security

Defining Administrative Domain





GPFS Security

Administrative Access



- **It is necessary to properly configure security in order to administer GPFS; this includes the following...**
 - Provide standard root access to designated system administrators.
 - most GPFS commands require root authority
 - Establish an authentication method between nodes in the GPFS cluster.
 - Designate a remote communication program for remote shell and remote file copy commands.
 - a *subset* of nodes must allow root level communication without the use of a password and without any extraneous messages
 - common choices are ssh/scp and rsh/rcp
 - designated using the mmcrcluster and mmchcluster commands
 - the selected option must use the rsh/rcp CLI
 - GPFS uses remote shell and remote file copy commands to do things like...
 - GPFS commands executed by a system administrator on a given node propagate configuration information to and perform administrative tasks on other nodes in the cluster.
 - GPFS automatically communicates changes of system state across the nodes of a cluster
- **Further information can be found in**
 - GPFS Administration and Programming Reference, version 3.3, pp. 1-3.



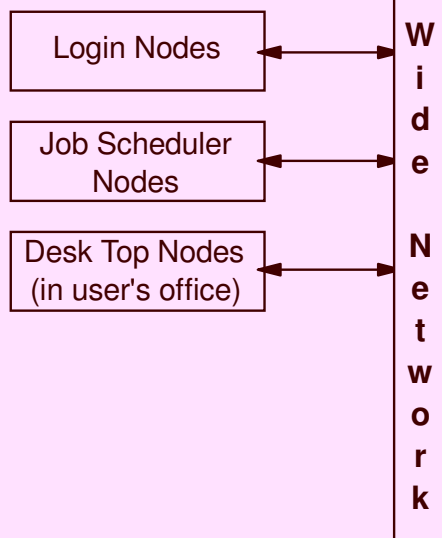
GPFS Security - Traditional

Example: Restricting Access to Trusted Environment

NON-TRUSTED

Nodes outside the trusted network are not part of the GPFS cluster

These nodes can be given *restricted* access to the GPFS cluster via NFS.



COMMENTS:

- ▶ Only the nodes within trusted network have direct access to GPFS file system
- ▶ User accounts do NOT exist on nodes in the trusted network
- ▶ User access is indirect via job schedulers, login nodes, etc.

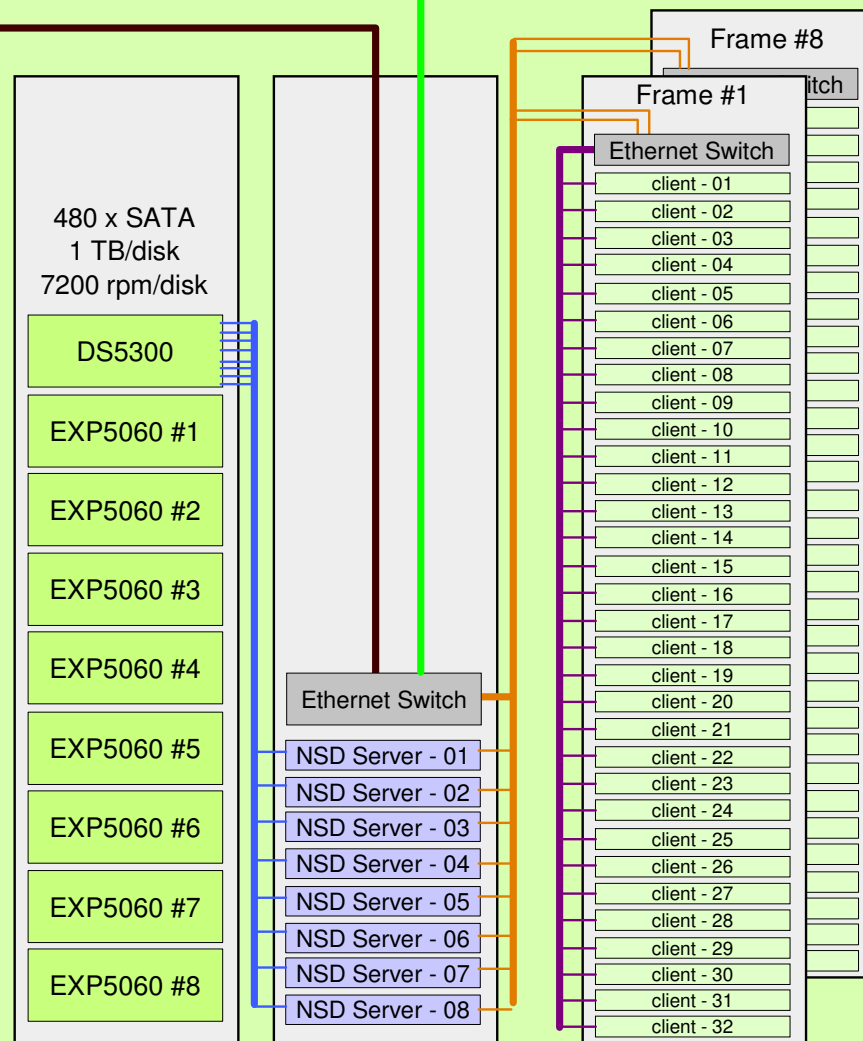
FOOTNOTE

* Prior to version 3.3, this was the only option for a GPFS cluster.

TRUSTED

mmchconfig adminMode=allToAll

The domain of trust can be extended over a WAN via GPFS multi-cluster. Use OpenSSL for access security. (n.b., root squash option)

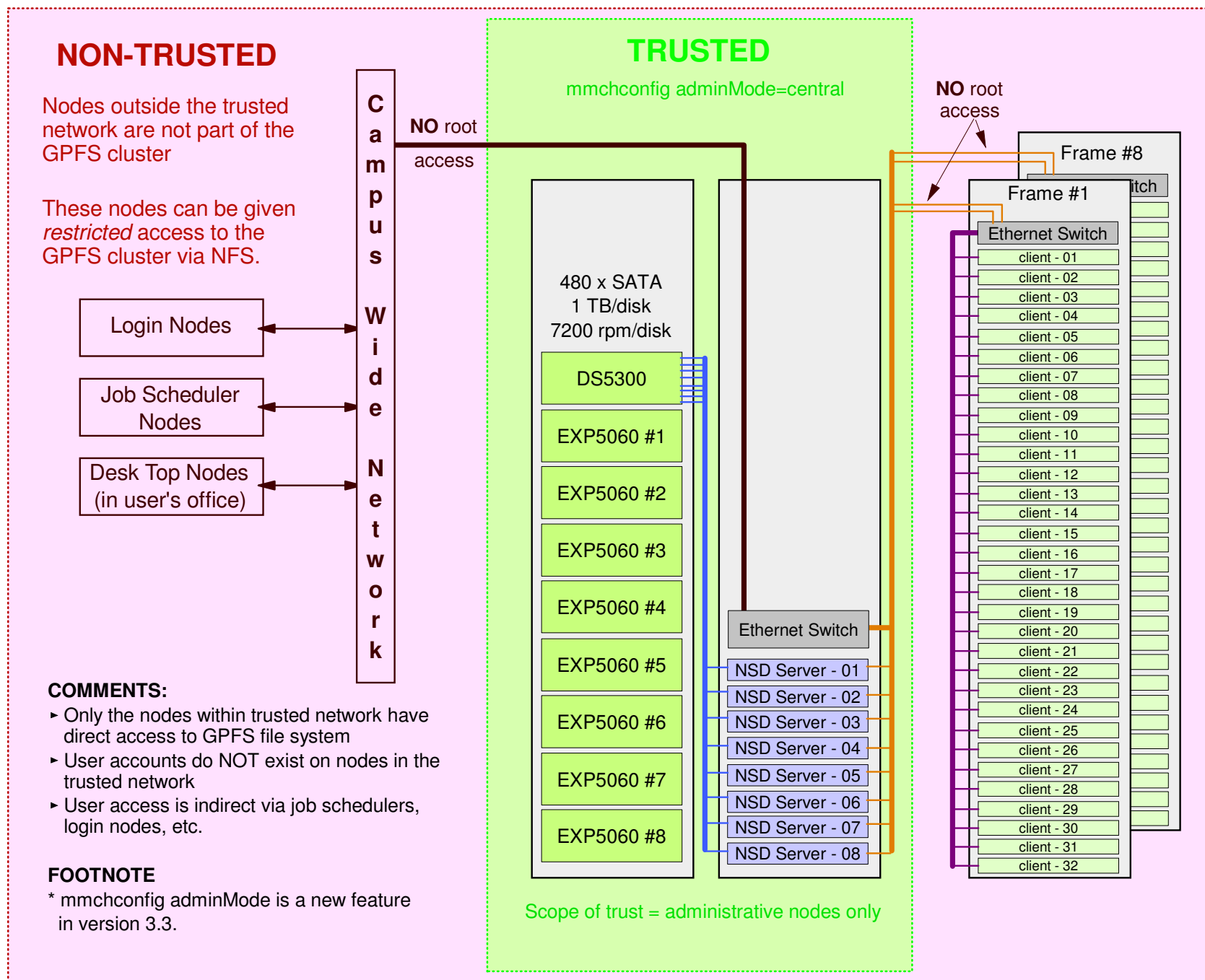


Scope of trust = all nodes in GPFS cluster



GPFS Security - New

Example: Restricting Access to Trusted Environment





GPFS Security

Example Configuring Passwordless ssh/scp Authentication

```
[root@nsd1 ~]# cd .ssh <--- Create the /root/.ssh directory if it does not exist.
[root@nsd1 .ssh]# ssh-keygen -t rsa -f id_rsa <--- Generate the public/private key pair (the other option is dsa)
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase): <--- Leave these responses blank to avoid passwords.
Enter same passphrase again: <---
Your identification has been saved in id_rsa.
Your public key has been saved in id_rsa.pub.
The key fingerprint is:
dc:74:17:46:0e:ea:ad:96:50:df:d3:bf:99:86:d6:c8 root@nsd1
[root@nsd1 .ssh]# cat id_rsa.pub >> authorized_keys <--- Append public key file to the authorized_keys file.
[root@nsd1 .ssh]# ssh nsd1 <--- Be sure you can ssh to yourself without a password; all nodes must be able to do this.
The authenticity of host 'nsd1 (172.31.1.78)' can't be established.
RSA key fingerprint is d8:4a:cd:96:45:25:34:19:34:fa:23:98:36:c0:ed:7e.
Are you sure you want to continue connecting (yes/no)? yes <--- This is normal the first time you ssh to a node.
Warning: Permanently added 'nsd1,172.31.1.78' (RSA) to the list of known hosts.
Last login: Thu Oct  9 17:01:06 2008 from nsd1
[root@nsd1 .ssh]# exit
Connection to nsd1 closed.
[root@nsd1 .ssh]# dir
total 20
-rw----- 1 root root 391 Oct  9 17:06 authorized_keys <--- Be sure the permissions are 600
-rw----- 1 root root 1675 Oct  9 17:05 id_rsa <--- and the owner/group is root.
-rw-r--r-- 1 root root 391 Oct  9 17:05 id_rsa.pub <--- Be sure the permissions are 644
-rw-r--r-- 1 root root 398 Oct  9 17:06 known_hosts <--- and the owner/group is root.
[root@nsd1 .ssh]#
```

The known_hosts file is generated "automagically" when a remote node first logs into the local node via ssh. In this example, it was created when we "ssh'd" to ourselves and answered "yes".



GPFS Security

Example Configuring Passwordless ssh/scp Authentication

```
[root@nsd1 .ssh]# for i in 2 3 4
```

```
> do
```

```
> scp authorized_keys id_rsa id_rsa.pub known_hosts nsd$i:.ssh
```

```
> done
```

It is necessary for all nodes in the GPFS cluster to have ssh keys. It is common practice to generate the keys on one node and copy them to all other nodes in the GPFS cluster.

The authenticity of host 'nsd2 (172.31.1.79)' can't be established.

RSA key fingerprint is 48:db:31:71:76:4f:25:f0:37:b1:62:29:d6:87:5e:4e.

Are you sure you want to continue connecting (yes/no)? **yes**

Warning: Permanently added 'nsd2,172.31.1.79' (RSA) to the list of known hosts.

root@nsd2's password: *********

authorized_keys	100%	391	0.4KB/s	00:00
-----------------	------	-----	---------	-------

id_rsa	100%	1675	1.6KB/s	00:00
--------	------	------	---------	-------

id_rsa.pub	100%	391	0.4KB/s	00:00
------------	------	-----	---------	-------

known_hosts	100%	796	0.8KB/s	00:00
-------------	------	-----	---------	-------

Answering "yes" to this request causes ssh to "automagically" append encrypted public keys to the local known_hosts file. Subsequent logins to any of these remote nodes will no longer encounter this request.

The authenticity of host 'nsd3 (172.31.1.80)' can't be established.

RSA key fingerprint is e9:96:bc:31:a6:7f:e5:29:92:06:f3:ac:3d:5a:2b:3c.

Are you sure you want to continue connecting (yes/no)? **yes**

Warning: Permanently added 'nsd3,172.31.1.80' (RSA) to the list of known hosts.

root@nsd3's password: *********

authorized_keys	100%	391	0.4KB/s	00:00
-----------------	------	-----	---------	-------

id_rsa	100%	1675	1.6KB/s	00:00
--------	------	------	---------	-------

id_rsa.pub	100%	391	0.4KB/s	00:00
------------	------	-----	---------	-------

known_hosts	100%	1194	1.2KB/s	00:00
-------------	------	------	---------	-------

The first ssh access to remote nodes requires a password. After properly copying the keys to these other nodes, a password challenge will no longer happen.

The authenticity of host 'nsd4 (172.31.1.81)' can't be established.

RSA key fingerprint is e2:3d:1b:3f:ef:6f:b8:bd:5e:0a:ab:e0:56:1b:83:39.

Are you sure you want to continue connecting (yes/no)? **yes**

Warning: Permanently added 'nsd4,172.31.1.81' (RSA) to the list of known hosts.

root@nsd4's password: *********

authorized_keys	100%	391	0.4KB/s	00:00
-----------------	------	-----	---------	-------

id_rsa	100%	1675	1.6KB/s	00:00
--------	------	------	---------	-------

id_rsa.pub	100%	391	0.4KB/s	00:00
------------	------	-----	---------	-------

known_hosts	100%	1592	1.6KB/s	00:00
-------------	------	------	---------	-------

```
[root@nsd1 .ssh]#
```



GPFS Security

Example Configuring Passwordless ssh/scp Authentication

```
[root@nsd1 .ssh]# ssh nsd1 ssh nsd2 ssh nsd3 ssh nsd4 ssh nsd1 date
```

Host key verification failed.

This is a simple test to be sure ssh is configured properly.

It failed because the known_hosts file was incomplete on the other nodes in the GPFS cluster.

```
[root@nsd1 .ssh]# for i in 2 3 4
```

```
> do
```

```
> scp known_hosts nsd$i:.ssh
```

```
> done
```

```
known_hosts
```

```
100% 1592 1.6KB/s 00:00
```

```
known_hosts
```

```
100% 1592 1.6KB/s 00:00
```

```
known_hosts
```

```
100% 1592 1.6KB/s 00:00
```

```
[root@nsd1 .ssh]# ssh nsd1 ssh nsd2 ssh nsd3 ssh nsd4 ssh nsd1 date
```

```
Thu Oct 9 17:14:45 EDT 2008
```

---The test completed properly this time.
This test is not fool proof, however.

```
[root@nsd1 .ssh]# ls -al
```

```
drwx----- 2 root root
```

```
4096 Oct 3 18:23 .ssh
```

---Be sure the permissions are 700 and the owner/group is root.

```
drwxr-x--- 12 root root
```

```
4096 Sep 24 03:25 root
```

---Be sure the permissions are 750 and the owner/group is root.

```
[root@nsd1 ~]#
```

WARNING: Some implementations of ssh/scp may not allow passwordless access if the permissions are not set properly.

COMMENT: This is a tedious process! For large clusters, automated tools are used to do this task.

The following pages is a potpourri of practical sysadm and tuning experience (often learned late at night under duress :->)



Who says an old dog can't learn new tricks?!?!



Read/Modify/Write Penalty

Choosing N in N+P in RAID 5 Configurations

RAID 5 "disk arrays" have N data disks and 1 parity disk

- ▶ This is called "N+P" (e.g., 4+P)

A RAID 5 "stripe" is $N * \text{segment_size}$ where `segment_size` is the size of the block of data written to 1 physical disk in the RAID 5 array

- ▶ If `segment_size` = 256K with 4+P RAID 5 array, then the `stripe_size` = 1024K

GPFS `block_size` should equal RAID 5 stripe size for best performance. Since GPFS `block_size` is not arbitrary (GPFS blocksize is 2^k), this implicitly restricts choices for N and the `segment_size` if optimum performance is to be achieved. For example...

- ▶ On a DS4000 system, $N = \{ 4 \mid 8 \}$
- ▶ If GPFS `block_size`=1024K, then
 - if $N == 4$, then `segment_size`=256K
 - if $N == 8$, then `segment_size`=128K
- ▶ If GPFS `block_size`=256K, then
 - if $N == 4$, then `segment_size`=64K
 - if $N == 8$, then `segment_size`=32K

If N+P and GPFS `block_size` are

- ▶ consistent, then `block_size` == $N * \text{segment_size}$ and the stripe is "over written"
 - this yields best performance
 - this is sometimes called a "full stride write"
- ▶ not consistent, then it is necessary read RAID 5 stripe, then update it, then write it
 - this significantly reduces performance



atime and mtime

■ **mmchfs -E { yes | no }**

■ **mmchfs -S { yes | no }**

First ls does not update atime. atime is updated when a file is actually accessed (ls accesses the directory). I do not have data on cost of atime - it really depends on the workload. The reason of recommending changing -E no (for mtime) is that on some systems we have observed impact of mtime in shared file updates (and variability in performance). My guess would have been to expect atime to be a lesser issue than mtime (what is their workload that makes them concerned about performance impact atime updates?).

-E no means no exact mtime.

For suppressing atime updates you would say -S yes

Requires remounting the file system

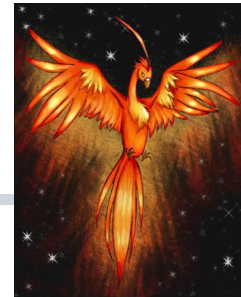


Importance of Stable LAN and SAN





MMFS_PHOENIX Error Log Message



AIX: `errpt -a`

Linux: `less /var/log/messages`

■ Common error message in the OS error log

Mar 15 15:55:15 bm-dell-10 mmfs: Error=**MMFS_PHOENIX**, ID=0xAB429E38, Tag=10383382: Reason code 668 Failure Reason Lost membership in cluster bm-dell-10. Unmounting file systems.

■ What is PHOENIX?

- It is the "high availability" layer in GPFS today.
- Replaces the RSCT service used by GPFS in its AIX days

■ Significance of error message

- MMFS_PHOENIX messages occur when a node joins, leaves, changes cluster membership
 - this could merely be a normal response to an external event (*e.g.*, `mmdelnod`)
 - this could be a response to anomalous event (*e.g.*, removing node due to loss of network access)



Testing Adapters and LUNs Using dd

OBSERVATIONS:

- ▶ A slow or improperly configured LAN adapter (e.g., Ethernet, Myrinet, IB) may adversely affect GPFS performance
- ▶ Use dd to isolate performance of a given adapter as follows:

```
dd if=/dev/zero bs=1024k count=1024 | ssh <hostname> dd of=/dev/null
```

- ▶ A slow or improperly configured LUN in a GPFS file system can slow down performance for the entire file system
- ▶ Use dd to isolate performance of a given LUN
for example, read a SCSI device in Linux

```
time dd if=/dev/sdc of=/dev/null bs=1024K count=2048
```

for example, read a raw device in AIX

```
time dd if=/dev/rhdisk2 of=/dev/null bs=1024K count=2048
```

- ▶ Use caution when writing to SCSI device... it will "clobber" a file system



Mixed GPFS Code Levels

■ Co-existence defined

- Nodes with different GPFS code levels may be active in the same cluster and simultaneously access the same file system.

■ Co-existence makes it possible to

- upgrade GPFS within a cluster without shutting down GPFS on all nodes
 - this is also called "rolling upgrades"
- mount GPFS file systems from other GPFS clusters that may be running a different GPFS code level

■ Beginning with GPFS 2.3.0.6

- Nodes running with different 2.3 *maintenance levels* may co-exist
- Nodes running the 2.3 and 3.1 releases cannot co-exist

■ Beginning with GPFS 3.1

- Release-to-release co-existence is officially supported
 - this includes the co-existence of maintenance as well as release levels
 - e.g., 3.1 and 3.2 may co-exist
 - Once all nodes are upgraded to latest version, it is necessary to run following commands
 - mmchconfig release=LATEST
 - mmchfs -V all
 - See the *GPFS: Concepts, Planning, and Installation Guide* for further information.



Suspending GPFS

If it is necessary to take down a node to repair something, do the following...

```
> mmfsctl <FS name> suspend
```

do something

```
> mmfsctl <FS name> resume
```

Fine Print:

Use the mmfsctl command to issue control requests to a particular GPFS file system. The command is used to temporarily suspend the processing of all application I/O requests, and later resume them, as well as to synchronize the file system's configuration state between peer clusters in disaster recovery environments.



SAN Conjestion

An Example

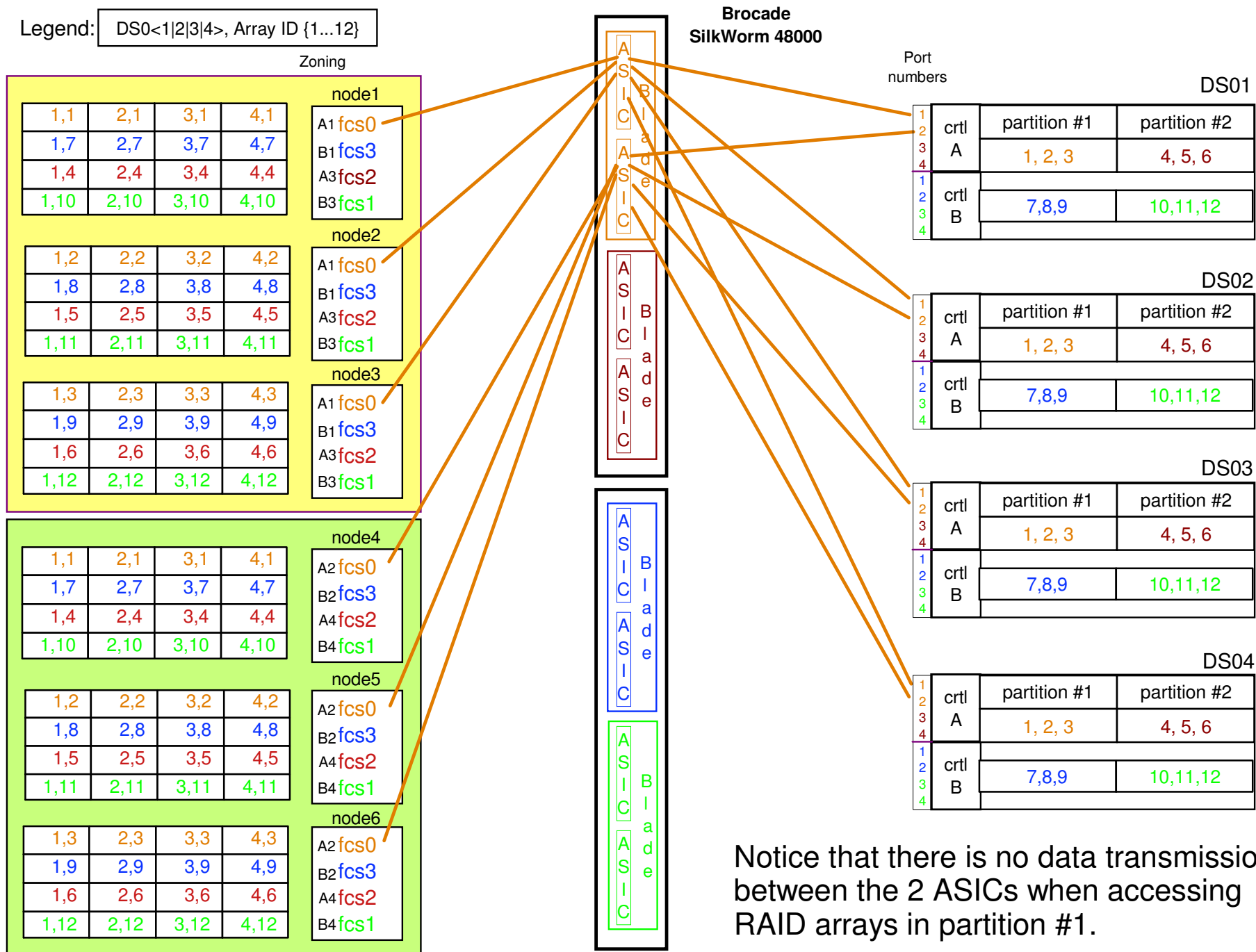
- **This analysis is based on the Brocade SilkWorm 48000 with 4 Gb/s FC fabric**
 - 2 SAN switches with 2 x 32 port blades
 - 2 ASICs per blade with 16 ports per ASIC
 - total ports available = 128
 - total ports used = 56
- **Empirical tests show that the effective ASIC BW < 1100 MB/s**
 - test code: dd to raw disks, read 4096 records with sizeof(record) = 1M
 - effective BW is the BW *measured by the application*
 - total BW through the ASIC is 2200 MB/s (accounting for the data in and data out streams)
- **Properly distribute host and controller connections across *all* ASICs to avoid ASIC saturation**
 - see cabling example on next page
 - requires using all DS4800 host side ports in this example
 - **BEST PRACTICE:** deploy cabling to avoid all "inter-ASIC" traffic
 - For completeness, each ASIC connects to a control processor enabling 32 Gb/s simplex or 64 Gb/s duplex inter-ASIC communication, however, the electronics of the ASIC does not *appear* to be able to handle that much aggregate BW.
- **This SAN cabling issue does not impact standard GPFS NSD configuration.**
 - In the standard NSD configuration, a SAN is not necessary. Moreover, each host port typically will be accessed by only 1 HBA; *i.e.*, there is a 1:1 HBA to host port ratio.
 - In this multi-cluster VSD configuration, there is a 3:1 HBA to host port ratio.



SAN Congestion

The View from the Perspective of fcs0

Legend: DS0<1|2|3|4>, Array ID {1...12}

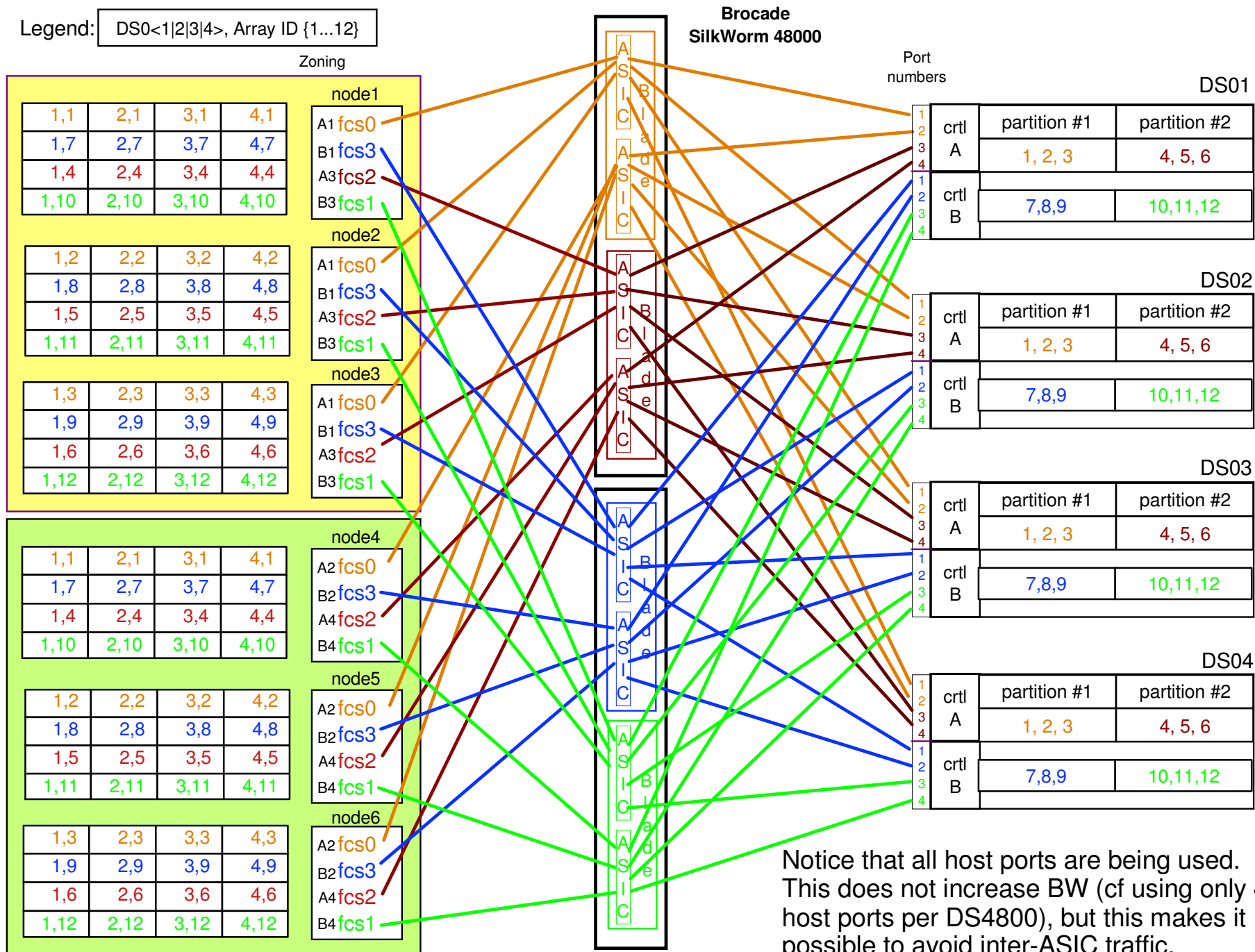




SAN Conjection

The Complete Cabeling View

Legend: DS0<1|2|3|4>, Array ID {1...12}





Non-unique Device Names

Due to sysadm errors, it is possible for a given LUN (RAID array) to have multiple device names.

e.g., /dev/sdy and /dev/sdba are the same RAID array

This can be seen when trying to create a new NSD on /dev/sdy

```
[root@gpfs01 gpfs1pp]# cat disk.lst.more
/dev/sdy:::::nsd14
[root@gpfs01 gpfs1pp]# mmcrnsd -F disk.lst.more
mmcrnsd: Processing disk sdy
mmcrnsd: Disk descriptor /dev/sdy:::::nsd14 refers to an existing NSD nsd11
```

Doing an mmlsnsd -f <device> -m shows nsd11 is assigned to /dev/sdba

```
nsd11      C0A8010142828D9C    /dev/sdba      gpfs01          directly attached
```

To confirm this, dump the NSD record on the LUN to stdout

```
[root@gpfs01 gpfs1pp]# dd if=/dev/sdy count=10
```

The dump is mostly binary, but the following text record can be seen

```
NSD descriptor for /dev/sdba created by GPFS Wed May 11 17:56:17 2005
```

Under Linux, this discrepancy can be seen by comparing the output between

```
mmlsnsd -f <device> -m                                <--output omitted to excessive length
```

and

```
[root@gpfs01 gpfs1pp]# ps -ef | grep mmfsd | grep -v grep
root      19625 19510  0 May11 ?                00:00:40 /usr/lpp/mmfs/bin//mmfsd
[root@gpfs01 gpfs1pp]# lsof -p 19625                <--output omitted to excessive length
```

which lists the devices associated with mmfsd



Fine Grain Directory Locking

■ The problem

- Multiple nodes changing the contents of a single directory at the same time hurts performance.
 - multiple nodes changing the same directory block simultaneously forces nodes to serialize operations to maintain block consistency
 - **alternative schemes**: if an application runs on multiple nodes where every node creates a file, it is recommended...
 - precreate all files (empty) on one node before all the other nodes open/access their file
 - have each node create their file in its own private directory
- very common in digital media and bio-informatics applications

■ This issue was corrected in patch release 3.2.1.6 (Sep 08)



LTG and VSD Buddy Buffers

AIX Tuning Parameter Affecting GPFS when Using VSD

When using large pSeries clusters with HPS (*i.e.*, "federation") or SP (*i.e.*, "colony") switch, VSD provides more efficient switch protocol than TCP/IP. But be sure to set the following AIX tuning parameters appropriately for GPFS.

- Set LTG size to be \geq GPFS blocksize
 - *e.g.*, if blocksize is 1024K, then increase LTG size to 1024K
 - requires AIX 5.2 or later
 - early AIX 5.2 releases may require patch(?)
 - modify mmvsdhelper script
 - see /usr/lpp/mmfs/bin
 - default = 128K
- Set buddy buffer size to be \geq GPFS blocksize
 - *e.g.*, if blocksize is 1024K, then set buddy buffer size to 1024K
 - modify via smitty
 - default = 256K



Consistent GID/UID





Linux Memory Management Issue



■ If memory is oversubscribed, Linux "shoots" large memory users.

- mmfsd is common target since under Linux, the pagepool is accounted as belonging to mmfsd
- Reducing the risk
 - reduce the size of the pagepool
 -



Installing GPFS Under CentOS



- GPFS is officially tested with RHEL and SUSE Linux *only*, but it *generally* works with other RHEL like distributions.

- CentOS is the most common; others include Scientific Linux, Rocks.

- If non-RHEL distributions may require some "tweaking".

- e.g., `make Autoconfig` fails when building the portability layer

- CentOS 5.4 example.

- Change the Red Hat version identifier

```
# echo "Red Hat Enterprise Linux Server release 5.4 (Tikanga)" > /etc/redhat-release
```

- Install RPMs for kernel-devel, compat-libstdc++-33

- Fix a "broken" symbolic link

- The link may appear as follows...

```
# ls -la /lib/modules/2.6.18-164.el5/build*  
lrwxrwxrwx 1 root root 46 Feb  2 16:14 build -> ../../usr/src/kernels/2.6.18-164.el5-x86_64
```

- It should look like this...

```
# ls -la /lib/modules/2.6.18-164.el5/build*  
lrwxrwxrwx 1 root root 44 Feb  4 13:08 build -> /usr/src/kernels/2.6.18-164.11.1.el5-x86_64/
```

- Fix it with the following steps...

```
# unlink /lib/modules/2.6.18-164.el5/build  
# ln -s /usr/src/kernels/2.6.18-164.11.1.el5-x86_64/ /lib/modules/2.6.18-164.el5/build
```



Queue Depth

What is the queue depth?

- Storage controllers can process up to a maximum number of concurrent I/O operation requests, sometimes called the maximum command queue depth (MQD)
 - DS5300: MQD = 4096 requests (plus a small number of active requests)
 - up to 2048 requests per RAID controller
 - up to 2048 requests per port up to the maximum allowed on a RAID controller
 - DCS9900: MQD = 4608 = 4096 queued requests + 512 active requests
 - up to 576 requests per port (*n.b.*, you must use all ports to get all 4608 requests)
- When the MQD has been reached, the controller will respond with a "queue full" status until some number of active requests have been processed and there is room for new requests.
- Storage adapter (e.g., HBA) drivers provide a device queue depth (DQD) parameter controlling the number of I/O requests submitted to a disk device on a given host
 - DQD sets the number of I/O requests per device (e.g., `sd<char>` or `hdisk<int>`)

Purpose of the queue depth parameters

- If a cluster submits more I/O requests than a storage controller can process, erratic behavior occurs
 - e.g., lost I/O requests
- The DQD is used to limit the amount of I/O received by a storage controller

How it works

- Parameters:
 - NN = Number of nodes submitting IOPs to a storage controller
 - NLUN = Number of LUNs per node
- For reliable operation, set DQD such that $MQD > NN * NLUN * DQD$

This formula ignores the fact that Linux may break a GPFS "packet" into several transactions. This is especially true for larger block sizes. This only makes the problem worse!

Consequences for a SAN configuration

- For a large SANs, DCD should be set small
 - e.g., set DQD = 1 for cluster with 128 nodes and 24 LUNs
- Problems:
 - a small DQD limits the number of I/O requests per node
 - setting DCD to 1 limits the size of the SAN cluster

Scratch Paper:

Consider cluster with 128 nodes and a DS5300

MQD = 4096

Set DQD = 1 and let NLUN = 24

$NN * NLUN * DQD = 128 * 24 * 1 = 3072 < MQD = 4096$: OK!

Set DQD = 1 and let NLUN = 32

$NN * NLUN * DQD = 128 * 32 * 1 = 4096 \sim < MQD = 4096$: Ouch!





12. GPFS Configuration Example

The following pages provide an actual example of installing and configuring GPFS under Linux using 4 x3650 NSD servers and a DCS9550 storage controller and disk enclosures. The steps for doing this under AIX are very similar; differences are explained in the annotations.

This example can be used as a hands on guide for a lab exercise.

Note the following:

red arial font is used for annotations

blue courier font is used to highlight commands and parameters

black courier font is used for screen text

COMMENT: This example is based on GPFS 3.1, but the steps for GPFS 3.2 are nearly identical. Key differences are highlighted in context.



Lab Exercise

■ Install GPFS from media

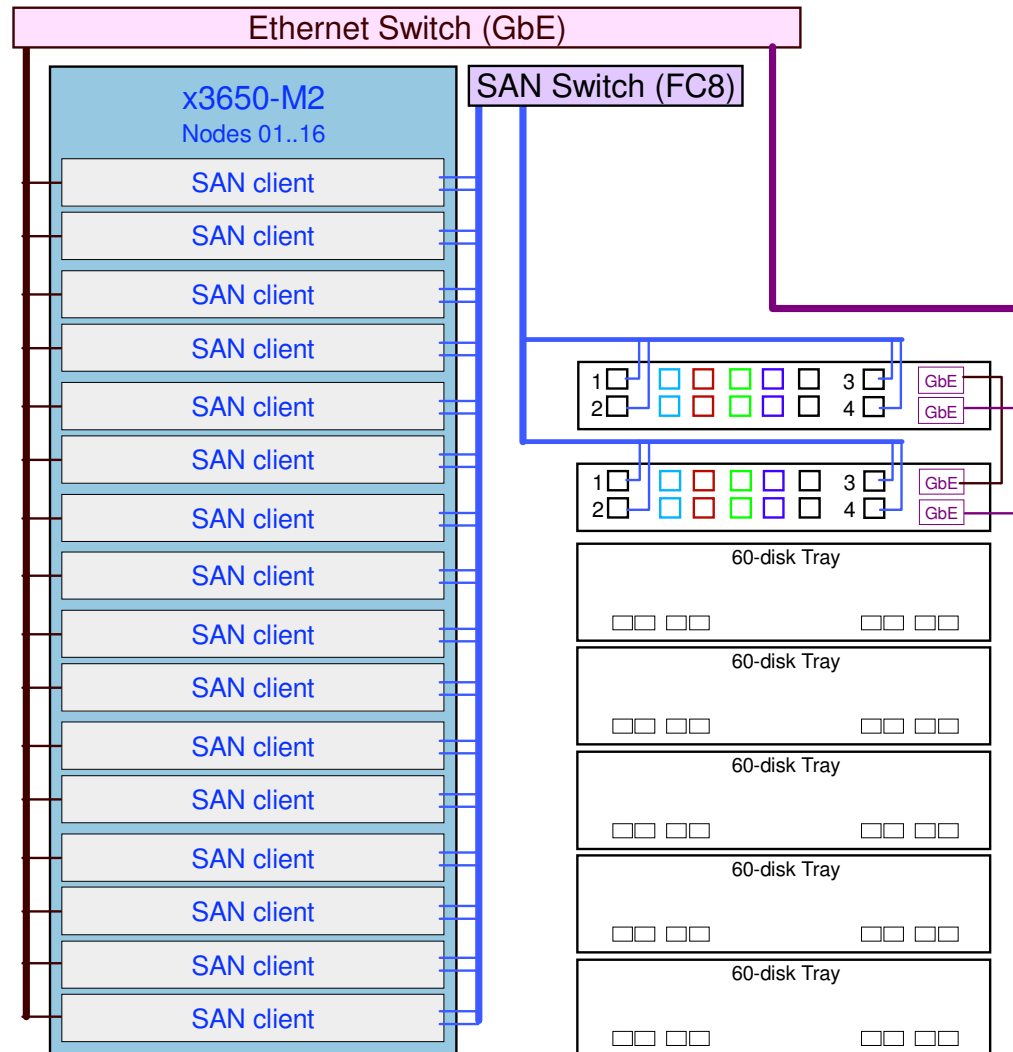
- if using Linux, build portability layer

■ Configure GPFS appropriate for lab cluster

■ As time allows

- experiment with "mmls" and "mmch" commands
- examine /var/adm/ras/mmfs.log.<extension>
- examine /var/mmfs
 - look at /var/mmfs/gen/mmsdrfs file
- run dd or other benchmark tests
 - monitor performance using iostat, vmstat, SMclient
 - iostat, vmstat not installed by default in Linux
 - must have Windows or AIX client to run SMclient
 - examine packet size and latency using mmpmon

PHYSICAL CONFIGURATION



- 450 GB/disk, 15 Krpm
- 56 x 4+P RAID 5

* Since this is CentOS and not RHEL, it's necessary to create a configuration file as follows so that the portability layer build procedures will work.

```
[root@node-01]# echo "Red Hat Enterprise Linux Server release 5.4 (Tikanga)" > /etc/redhat-release
```



EXAMPLE: Installing and Configuring GPFS

Outline of Steps to Install and Configure GPFS

1. Establish administrative control and scope
 - a. *e.g.*, Enable ssh with passwordless root access to designated nodes.
2. Install the GPFS code
 - a. Base version (*e.g.*, 3.3.0.0)
 - b. PTF version (*e.g.*, 3.3.0.4)
3. Build portability RPM It is only necessary to build the portability layer under Linux.
4. Configure a GPFS File System
 - a. Create cluster
 - b. Declare client and server licenses
 - c. Change global GPFS parameters and start the GPFS daemon
 - d. Create the NSDs
 - e. Create and Mount the file system



EXAMPLE: Installing and Configuring GPFS

Steps to Install the GPFS Code under Linux

Installing GPFS under AIX and Windows is quite different. See the note below for references.

1. Create an NFS mounted installation directory for extracting the RPMs
 - ▶ Alternatively, copy the RPMs to all nodes
 - ▶ Example installation directory: `/gpfs_install/gpfs_3.3.0.0`
2. Copy base version RPMs to the installation directory and extract the RPMs on all nodes in the GPFS cluster.
 - ▶ Sample RPM names:
 - gpfs.base-3.3.0-0.x86_64.rpm
 - gpfs.docs-3.3.0-0.noarch.rpm *gpfs.docs... contains the man pages. They will be installed in `/usr/share/man/`*
 - gpfs.gpl-3.3.0-0.noarch.rpm
 - gpfs.gui-3.3.0-0.x86_64.rpm *It is not necessary to install the GUI.*
 - gpfs.msg.en_US-3.3.0-0.noarch.rpm
3. Download the latest update package. This comes as a tar/gzip file from
 - ▶ <https://www14.software.ibm.com/webapp/set2/sas/f/gpfs/download/home.html>
 - ▶ `gpfs-3.3.0-4.x86_64.update.tar.gz`
 - This file contains a different version of the same RPMs as the base version.
4. Copy this file to the installation directory, then `gunzip/untar` this file, and extract the RPMs on all nodes in the GPFS cluster.
 - ▶ Example installation directory: `/gpfs_install/gpfs_3.3.0.4`

The steps for doing this are more thoroughly documented in chapter 5 of the *GPFS Concepts, Planning and Installation Guide*

The steps for installing the GPFS code under AIX and Windows are also documented in this guide. It can be found at

http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp?topic=/com.ibm.cluster.gpfs.doc/gpfs_faqs/gpfs_com_faq.html

This link may not take you directly to the current GPFS FAQ, but by drilling down, you can get there.



EXAMPLE: Installing and Configuring GPFS

Steps to Install the GPFS Code under Linux: Building Portability RPM

5. Select a node in the cluster and do the following:

- ▶ `cd /usr/lpp/mmfs/src`
- ▶ `make Autoconfig`
- ▶ `make World`
- ▶ `make InstallImages`
- ▶ `make rpm`
 - sample rpm name: `gpfs.gplbin-2.6.18-164.11.1.el5-3.3.0-4.x86_64.rpm`

Use `echo $?` to verify each make operation completes normally.

6. Copy portability rpm to all nodes and extract.

7. Warnings and Caveates

- ▶ If a cluster has mixed architectures and/or kernel levels, it is necessary build a portability rpm for each instance and copy it to like nodes.
- ▶ Required Linux patches for GPFS can be found at:
<http://www.ibm.com/developerworks/opensource/>



EXAMPLE: Installing and Configuring GPFS

```
[root@node-01 GPFS_install]# cat node_spec.lst
```

```
node-01:manager-quorum
```

```
node-02:manager-quorum
```

The manager-quorum nodes must be licensed as servers.

```
node-03:manager-quorum
```

```
node-04
```

```
node-05
```

```
node-06
```

```
node-07
```

```
node-08
```

```
node-09
```

The remaining nodes must be licensed as clients.

```
node-10
```

```
node-11
```

```
node-12
```

```
node-13
```

```
node-14
```

```
node-15
```

```
node-16
```

Administrative domain spans all nodes.
(i.e., the traditional security model)



EXAMPLE: Installing and Configuring GPFS

```
[root@node-01 GPFS_install]# mmcrcluster -n node_spec.lst -p node-01 -s node-02 -R  
/usr/bin/scp -r /usr/bin/ssh
```

Create the
GPFS cluster.

```
Thu Feb  4 21:56:06 EST 2010: mmcrcluster: Processing node node-01  
Thu Feb  4 21:56:06 EST 2010: mmcrcluster: Processing node node-02  
Thu Feb  4 21:56:06 EST 2010: mmcrcluster: Processing node node-03  
Thu Feb  4 21:56:07 EST 2010: mmcrcluster: Processing node node-04  
Thu Feb  4 21:56:07 EST 2010: mmcrcluster: Processing node node-05  
Thu Feb  4 21:56:08 EST 2010: mmcrcluster: Processing node node-06  
Thu Feb  4 21:56:08 EST 2010: mmcrcluster: Processing node node-07  
Thu Feb  4 21:56:09 EST 2010: mmcrcluster: Processing node node-08  
Thu Feb  4 21:56:09 EST 2010: mmcrcluster: Processing node node-09  
Thu Feb  4 21:56:10 EST 2010: mmcrcluster: Processing node node-10  
Thu Feb  4 21:56:10 EST 2010: mmcrcluster: Processing node node-11  
Thu Feb  4 21:56:11 EST 2010: mmcrcluster: Processing node node-12  
Thu Feb  4 21:56:11 EST 2010: mmcrcluster: Processing node node-13  
Thu Feb  4 21:56:12 EST 2010: mmcrcluster: Processing node node-14  
Thu Feb  4 21:56:12 EST 2010: mmcrcluster: Processing node node-15  
Thu Feb  4 21:56:13 EST 2010: mmcrcluster: Processing node node-16  
mmcrcluster: Command successfully completed  
mmcrcluster: Warning: Not all nodes have proper GPFS license designations.  
    Use the mmchlicense command to designate licenses as needed.  
mmcrcluster: Propagating the cluster configuration data to all  
affected nodes. This is an asynchronous process.
```

This is a common and routine message. GPFS is using ssh and scp to copy configuration information to all of the nodes asynchronously. It may be that the node on which the command is executed may complete before all of the other nodes are ready.

mmcrcluster parameters

- n: list of nodes to be included in the cluster
- p: primary GPFS cluster configuration server node
- s: secondary GPFS cluster configuration server node
- R: remote copy command (e.g., rcp or scp)
- r: remote shell command (e.g., rsh or ssh)



EXAMPLE: Installing and Configuring GPFS

[root@node-01 GPFS_install]# `mmlscluster` "List" the cluster to verify that the cluster is created as intended

GPFS cluster information
=====

GPFS cluster name: node-01
GPFS cluster id: 12402633858572060870
GPFS UID domain: node-01
Remote shell command: /usr/bin/ssh
Remote file copy command: /usr/bin/scp

GPFS cluster configuration servers:

Primary server: node-01
Secondary server: node-02

Node	Daemon node name	IP address	Admin node name	Designation
1	node-01	172.31.1.200	node-01	quorum-manager
2	node-02	172.31.1.201	node-02	quorum-manager
3	node-03	172.31.1.202	node-03	quorum-manager
4	node-04	172.31.1.203	node-04	
5	node-05	172.31.1.204	node-05	
6	node-06	172.31.1.205	node-06	
7	node-07	172.31.1.206	node-07	
8	node-08	172.31.1.207	node-08	
9	node-09	172.31.1.210	node-09	
10	node-10	172.31.1.211	node-10	
11	node-11	172.31.1.212	node-11	
12	node-12	172.31.1.213	node-12	
13	node-13	172.31.1.214	node-13	
14	node-14	172.31.1.215	node-14	
15	node-15	172.31.1.216	node-15	
16	node-16	172.31.1.217	node-16	



EXAMPLE: Installing and Configuring GPFS

```
[root@node-01 GPFS_install]# cat license_server.lst
```

```
node-01
```

```
node-02
```

```
node-03
```

```
[root@node-01 GPFS_install]# mmchlicense server --accept -N license_server.lst
```

The following nodes will be designated as possessing GPFS server licenses:

```
node-01
```

```
node-02
```

```
node-03
```

mmchlicense: Command successfully completed

mmchlicense: Propagating the cluster configuration data to all affected nodes. This is an asynchronous process.

```
[root@node-01 GPFS_install]# mmchlicense client --accept -N license_client.lst
```

The following nodes will be designated as possessing GPFS client licenses:

```
node-04
```

```
node-05
```

```
node-06
```

```
node-07
```

```
node-08
```

```
node-09
```

```
node-10
```

```
node-11
```

```
node-12
```

```
node-13
```

```
node-14
```

```
node-15
```

```
node-16
```

mmchlicense: Command successfully completed

mmchlicense: Propagating the cluster configuration data to all affected nodes. This is an asynchronous process.

mmchlicense parameters

server: server license type

client: client license type

--accept: suppress the license prompt
(implies you accept license terms)

-N: list of nodes for a given license type

It is necessary to explicitly declare both license types.



EXAMPLE: Installing and Configuring GPFS

```
[root@node-01 GPFS_install]# mmfslslicense
```

```
Summary information
-----
Number of nodes defined in the cluster:                16
Number of nodes with server license designation:        3
Number of nodes with client license designation:        13
Number of nodes still requiring server license designation: 0
Number of nodes still requiring client license designation: 0
```

mmchlicense parameters

-L: Displays the license type for each node, using an * to designate node with licenses out of compliance.

```
[root@node-01 GPFS_install]# mmfslslicense -L
```

Node name	Required license	Designated license
node-01	server	server
node-02	server	server
node-03	server	client *
node-04	client	client
node-05	client	node *

```
Summary information
-----
Number of nodes defined in the cluster: 5
Number of nodes with server license designation: 2
Number of nodes with client license designation: 2
Number of nodes still requiring server license designation: 1
Number of nodes still requiring client license designation: 1
```



EXAMPLE: Installing and Configuring GPFS

```
[root@node-01 GPFS_install]# mmchconfig maxMBpS=2000, maxblocksize=4m, pagepool=256m,
autoload=yes, adminMode=allToAll
```

```
Verifying GPFS is stopped on all nodes ...
```

```
mmchconfig: Command successfully completed
```

```
mmchconfig: Propagating the cluster configuration data to all
affected nodes. This is an asynchronous process.
```

Change selected
global parameters.

```
[root@node-01 GPFS_install]# mmlsconfig
Configuration data for cluster bm-dell-10:
```

“List” the global parameters to verify that they are set as intended.

```
-----
clusterName node-01
clusterId 12402633858572060870
autoload yes
minReleaseLevel 3.3.0.2
dmapiFileHandleSize 32
maxMBpS 2000
maxblocksize 4m
pagepool 256m
adminMode allToAll
```

```
File systems in cluster node-01:
```

```
-----
(none)
```

The **mmchconfig** parameters are

maxMBpS: Limit the LAN BW per node. To get peak rate, set it
~= 2X the desired BW; do NOT set it excessively large.

maxblocksize: Maximum file system block size allowed. This
parameter can **not** be easily changed.

pagepool: Size of GPFS cache.

autoload: yes -> start mmfsd when a node is rebooted

adminMode: allToAll -> all nodes allow passwordless root access

client -> subset of nodes allow passwordless root access

NOTES:

- ▶ **mmchconfig** parameters can be set differently on different nodes using the **-N** option.
- ▶ There are many more **mmchconfig** parameters possible, most of which are undocumented.



EXAMPLE: Installing and Configuring GPFS

```
[root@node-01 GPFS_install]# mmstartup -a
```

```
Thu Feb  4 21:58:56 EST 2010: mmstartup: Starting GPFS ...
```

Start the GPFS daemon (aka, `mmfsd`)
on all nodes in the cluster.

```
[root@node-01 GPFS_install]# mmgetstate -a
```

Be sure `mmfsd` is active on all nodes before proceeding.

Node number	Node name	GPFS state

1	node-01	active
2	node-02	active
3	node-03	active
4	node-04	active
5	node-05	active
6	node-06	active
7	node-07	active
8	node-08	active
9	node-09	active
10	node-10	active
11	node-11	active
12	node-12	active
13	node-13	active
14	node-14	active
15	node-15	active
16	node-16	active



EXAMPLE: Installing and Configuring GPFS

```
[root@node-01 GPFS_install]# cat disk.lst
```

```
dm-3:::metadataOnly::ssd0
dm-4:::metadataOnly::ssd1
dm-5:::metadataOnly::ssd2
dm-6:::metadataOnly::ssd3
```

```
dm-7:::dataOnly::sas4
dm-8:::dataOnly::sas5
dm-9:::dataOnly::sas6
dm-10:::dataOnly::sas7
dm-11:::dataOnly::sas8
dm-12:::dataOnly::sas9
dm-13:::dataOnly::sas10
dm-14:::dataOnly::sas11
```

Since GPFS is configured as a SAN topology, primary and backup NSD servers are not specified.

Using the Linux multi-pathing driver.

Create a NSD specification file.

The format for each line is as follows

f1:f2:f3:f4:f5:f6:f7:

where

f1 = scsi device

f2 = comma separate NSD server list
there can be upto 8 NSD servers

f3 = NULL (retained for legacy reasons)

f4 = usage

f5 = failure group

f6 = NSD name

f7 = storage pool name

Fields left blank are filled with default value

```
[root@node-01 GPFS_install]# cp disk.lst disk.lst.orig
```

Back up this specifications since its an input/output file for the `mmcrnsd`.

```
[root@node-01 GPFS_install]# mmcrnsd -F disk.lst -v no
```

```
mmcrnsd: Processing disk dm-3
mmcrnsd: Processing disk dm-4
mmcrnsd: Processing disk dm-5
mmcrnsd: Processing disk dm-6
mmcrnsd: Processing disk dm-7
mmcrnsd: Processing disk dm-8
mmcrnsd: Processing disk dm-9
mmcrnsd: Processing disk dm-10
mmcrnsd: Processing disk dm-11
mmcrnsd: Processing disk dm-12
mmcrnsd: Processing disk dm-13
mmcrnsd: Processing disk dm-14
```

```
mmcrnsd: Propagating the cluster configuration data to all
affected nodes. This is an asynchronous process.
```

The `mmcrnsd` parameters are

`-F`: name of the NSD specification file (*n.b.*, the file is changed by this command... keep a back up!)

`-v`: check if this disk is part of an existing GPFS file system or ever had a GPFS file system on it (*n.b.*, if it does/did and the parameter is yes, then `mmcrnsd` will not create it as a new NSD)



EXAMPLE: Installing and Configuring GPFS

```
[root@node-01 GPFS_install]# cat disk.lst
# dm-3:::metadataOnly::ssd0
ssd0:::metadataOnly:-1::
# dm-4:::metadataOnly::ssd1
ssd1:::metadataOnly:-1::
# dm-5:::metadataOnly::ssd2
ssd2:::metadataOnly:-1::
# dm-6:::metadataOnly::ssd3
ssd3:::metadataOnly:-1::

# dm-7:::dataOnly::sas4
sas4:::dataOnly:-1::
# dm-8:::dataOnly::sas5
sas5:::dataOnly:-1::
# dm-9:::dataOnly::sas6
sas6:::dataOnly:-1::
# dm-10:::dataOnly::sas7
sas7:::dataOnly:-1::
# dm-11:::dataOnly::sas8
sas8:::dataOnly:-1::
# dm-12:::dataOnly::sas9
sas9:::dataOnly:-1::
# dm-13:::dataOnly::sas10
sas10:::dataOnly:-1::
# dm-14:::dataOnly::sas11
sas11:::dataOnly:-1::
```

This shows on the `mmcrnsd` command modifies the NSD specification file.



EXAMPLE: Installing and Configuring GPFS

```
[root@node-01 GPFS_install]# mmlsnsd -X
```

 Verify that the NSDs were properly created.

Disk name	NSD volume ID	Device	Devtype	Node name	Remarks
sas10	AC1F01C84B6C5755	/dev/dm-13	dmm	node-01	Since GPFS is configured as SAN topology in this example, the node names are not unique. In a LAN configuration, there is one line for each LUN and each node where it is mounted. mmlsnsd parameters -X: list extended NSD information
sas11	AC1F01C84B6C5756	/dev/dm-14	dmm	node-01	
sas4	AC1F01C84B6C574F	/dev/dm-7	dmm	node-01	
sas5	AC1F01C84B6C5750	/dev/dm-8	dmm	node-01	
sas6	AC1F01C84B6C5751	/dev/dm-9	dmm	node-01	
sas7	AC1F01C84B6C5752	/dev/dm-10	dmm	node-01	
sas8	AC1F01C84B6C5753	/dev/dm-11	dmm	node-01	
sas9	AC1F01C84B6C5754	/dev/dm-12	dmm	node-01	
ssd0	AC1F01C84B6C574B	/dev/dm-3	dmm	node-01	
ssd1	AC1F01C84B6C574C	/dev/dm-4	dmm	node-01	
ssd2	AC1F01C84B6C574D	/dev/dm-5	dmm	node-01	
ssd3	AC1F01C84B6C574E	/dev/dm-6	dmm	node-01	

```
[root@bm-dell-10 GPFS_install]# mmlsnsd
```

 Assume this command was issued after a file system is built.

File system	Disk name	NSD servers	
gpfs1	sas10	(directly attached)	By omitting the -X parameter, a different view is presented. For example, if a file system exists, it would then show the LUN to file system mapping. If GPFS used a LAN topology, it would show primary and backup NSD servers.
gpfs1	sas11	(directly attached)	
gpfs1	sas4	(directly attached)	
gpfs1	sas5	(directly attached)	
gpfs1	sas6	(directly attached)	
gpfs1	sas7	(directly attached)	
gpfs1	sas8	(directly attached)	
gpfs1	sas9	(directly attached)	
gpfs1	ssd0	(directly attached)	
gpfs1	ssd1	(directly attached)	
gpfs1	ssd2	(directly attached)	
gpfs1	ssd3	(directly attached)	



EXAMPLE: Installing and Configuring GPFS

```
[root@node-01 GPFS_install]# mmcrfs /gpfs1 gpfs1 -F disk.lst -A yes -B 256k -v no -n 32
```

The following disks of gpfs1 will be formatted on node node-01:

```
ssd0: size 192937984 KB
ssd1: size 192937984 KB
ssd2: size 192937984 KB
ssd3: size 192937984 KB
sas4: size 2264924160 KB
sas5: size 2264924160 KB
sas6: size 2264924160 KB
sas7: size 2264924160 KB
sas8: size 2264924160 KB
sas9: size 2264924160 KB
sas10: size 2264924160 KB
sas11: size 2264924160 KB
```

Formatting file system ...

Disks up to size 21 TB can be added to
storage pool 'system'.

Creating Inode File

Creating Allocation Maps

Clearing Inode Allocation Map

Clearing Block Allocation Map

Formatting Allocation Map for storage pool 'system'

Completed creation of file system /dev/gpfs1.

mmcrfs: Propagating the cluster configuration data
to all affected nodes. This is an asynchronous
process.

Parameters for mmcrfs

/gpfs1: mount point

gpfs1: device entry in /dev for the file system

-F: output file from the mmcrnsd command

-A: mount the file system automatically every time
mmfsd is started

-B: actual block size for this file system; it can not be
larger than the maxblocksize set by the mmchconfig
command

-v: check if this disk is part of an existing GPFS file
system or ever had a GPFS file system on it (*n.b.*,
if it does/did and the parameter is yes, then

mmcrfs will not include this disk in this file system)
-n: estimated number of nodes that will mount this file
system (see **note** below).

The optimum value for the actual block size is both
application and controller dependent. Experimentation is
recommended to determine the best choice for this value.
The options are 16k, 64k, 128K, 256k, 512k, 1M, 2M, 4M.

The /etc/fstab is automatically updated by this command.

COMMENT:

- Do **not** forget to set the -n parameter. Since it provides an estimate for the number of nodes that will mount the file system, try estimate future growth without wildly overestimating. While it can be off quite a bit with minimal impact, after it crosses a certain threshold performance can be severely impacted (e.g., performance will be impacted when it is off by an order of magnitude and the file system is over 70% capacity) and this parameter can not be easily changed.
- If you configure GPFS with a SAN topology on a cluster that you anticipate will exceed 32 nodes, seek technical assistance from IBM.



EXAMPLE: Installing and Configuring GPFS

```
[root@node-01 GPFS_install]# mmlsfs gpfs1 Verify file system status..
```

flag	value	description
-f	8192	Minimum fragment size in bytes
-i	512	Inode size in bytes
-I	16384	Indirect block size in bytes
-m	1	Default number of metadata replicas
-M	2	Maximum number of metadata replicas
-r	1	Default number of data replicas
-R	2	Maximum number of data replicas
-j	scatter	Block allocation type
-D	nfs4	File locking semantics in effect
-k	all	ACL semantics in effect
-a	1048576	Estimated average file size
-n	32	Estimated number of nodes that will mount file system
-B	262144	Block size
-Q	none	Quotas enforced
	none	Default quotas enabled
-F	18448390	Maximum number of inodes
-V	11.05 (3.3.0.2)	File system version
-u	yes	Support for large LUNs?
-z	no	Is DMAPI enabled?
-L	4194304	Logfile size
-E	yes	Exact mtime mount option
-S	no	Suppress atime mount option
-K	whenpossible	Strict replica allocation option
-P	system	Disk storage pools in file system
-d	ssd0;ssd1;ssd2;ssd3;sas4;sas5;sas6;sas7;sas8;sas9;sas10;sas11	Disks in file system
-A	yes	Automatic mount option
-o	none	Additional mount options
-T	/gpfs1	Default mount point



EXAMPLE: Installing and Configuring GPFS

```
[root@node-01 GPFS_install]# mmlsdisk gpfs1 Verify disk status..
```

disk name	driver type	sector size	failure group	holds metadata	holds data	status	availability	storage pool
ssd0	nsd	512	-1	yes	no	ready	up	system
ssd1	nsd	512	-1	yes	no	ready	up	system
ssd2	nsd	512	-1	yes	no	ready	up	system
ssd3	nsd	512	-1	yes	no	ready	up	system
sas4	nsd	512	-1	no	yes	ready	up	system
sas5	nsd	512	-1	no	yes	ready	up	system
sas6	nsd	512	-1	no	yes	ready	up	system
sas7	nsd	512	-1	no	yes	ready	up	system
sas8	nsd	512	-1	no	yes	ready	up	system
sas9	nsd	512	-1	no	yes	ready	up	system
sas10	nsd	512	-1	no	yes	ready	up	system
sas11	nsd	512	-1	no	yes	ready	up	system



EXAMPLE: Installing and Configuring GPFS

```
[root@node-01 GPFS_install]# mmmount /gpfs1 -a
Fri Feb  5 12:50:17 EST 2010: mmmount: Mounting file systems ...
[root@bm-dell-10 GPFS_install]# chmod 777 /gpfs1           Permissions propagate to mount points on all nodes.
[root@bm-dell-10 GPFS_install]# touch /gpfs1/test_file     Sanity check...
[root@bm-dell-10 GPFS_install]# dir /gpfs1
total 0
-rw-r--r-- 1 root root 0 Feb  5 12:51 test_file

[root@node-01 GPFS_install]# time dd if=/dev/zero of=/gpfs1/buggs_bunny bs=256k count=16384
16384+0 records in
16384+0 records out
4294967296 bytes (4.3 GB) copied, 7.50776 seconds, 572 MB/s

real    0m7.511s
user    0m0.010s
sys     0m2.102s

[root@node-01 GPFS_install]# dir /gpfs1
total 4194304
-rw-r--r-- 1 root root 4294967296 Feb  5 13:01 buggs_bunny
-rw-r--r-- 1 root root           0 Feb  5 12:51 test_file

[root@node-01 GPFS_install]# cat /etc/fstab
/dev/VolGroup00/LogVol100 / ext3 defaults 1 1
LABEL=/boot /boot ext3 defaults 1 2
tmpfs /dev/shm tmpfs defaults 0 0
devpts /dev/pts devpts gid=5,mode=620 0 0
sysfs /sys sysfs defaults 0 0
proc /proc proc defaults 0 0
/dev/VolGroup00/LogVol101 swap swap defaults 0 0
/dev/gpfs1 /gpfs1 gpfs rw,mtime,atime,dev=gpfs1,autostart 0 0
```

 mmcrfs automatically adds a GPFS stanza to the fstab file

EXAMPLE: Installing and Configuring GPFS

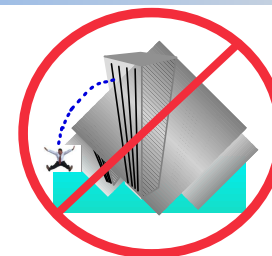
What if I screw up?



Clean up and start over again.

Don't jump!

It's easier the second time!



Option #1 The proper way to do it.

1. Unmount the GPFS file system

```
[root@gpfs1 gpfs_install]# mmunmount /gpfs1 -a
```

2. Delete GPFS file system

```
[root@gpfs1 gpfs_install]# mmdelfs /gpfs1
```

3. Delete GPFS NSDs

```
[root@gpfs1 gpfs_install]# for i in `seq 1 24`
> do
> mmdelnsd nsd_lun$i
> done
```

4. Shutdown GPFS daemons

```
[root@gpfs1 gpfs_install]# mmshutdown -a
```

5. Delete the GPFS cluster

```
[root@gpfs1 gpfs_install]# mmdelnode -a
```

Properly deleting the file system ensures that the file system descriptors are deleted from the disks so that they will not create issues upon a subsequent file system creation attempt.

Properly deleting the NSDs ensures that the NSD descriptors are deleted so that they will not create issues upon a subsequent NSD creation attempt.

Option #2 But what if I really screw things up?

1. Unmount the GPFS file system and shutdown the GPFS daemons

```
[root@gpfs1 gpfs_install]# mmunmount /gpfs1 -a
[root@gpfs1 gpfs_install]# mmfsadm cleanup
```

2. Delete selected configuration files on all nodes

```
[root@gpfs1 gpfs_install]# rm -f /var/mmfs/etc/mmfs.cfg
[root@gpfs1 gpfs_install]# rm -f /var/mmfs/gen/*
[root@gpfs1 gpfs_install]# rm -f /var/mmfs/tmp/*
```



WARNING: Use with extreme caution! Once the configuration files have been deleted, the GPFS cluster no longer exists and any data on the disks will likely be lost. Use this method only when Option #1 fails.

Deleting /var/mmfs/tmp may not be necessary; if so, then skip this step since it keeps backup copies of the mmsdrfs file.





EXAMPLE: Installing and Configuring GPFS

Verifying Multipath Driver Settings

```
[root@node-01 GPFS_install]# multipath -ll
LUN11 (360001ff08000a000000000258bb1000b) dm-14 DDN,SFA 10000
[size=2.1T][features=1 queue_if_no_path][hwhandler=0][rw]
\_ round-robin 0 [prio=50][active]
  \_ 1:0:0:12 sdz 65:144 [active][ready]
\_ round-robin 0 [prio=10][enabled]
  \_ 2:0:0:12 sdn 8:208 [active][ready]
LUN10 (360001ff08000a000000000248bb0000a) dm-13 DDN,SFA 10000
[size=2.1T][features=1 queue_if_no_path][hwhandler=0][rw]
\_ round-robin 0 [prio=50][active]
  \_ 2:0:0:11 sdm 8:192 [active][ready]
\_ round-robin 0 [prio=10][enabled]
  \_ 1:0:0:11 sdy 65:128 [active][ready]
```

SAS Drives

.... partial listing

```
LUN1 (360001ff08000a0000000001b8ba70001) dm-4 DDN,SFA 10000
[size=184G][features=1 queue_if_no_path][hwhandler=0][rw]
\_ round-robin 0 [prio=50][active]
  \_ 1:0:0:2 sdp 8:240 [active][ready]
\_ round-robin 0 [prio=10][enabled]
  \_ 2:0:0:2 sdd 8:48 [active][ready]
LUN0 (360001ff08000a0000000001a8ba60000) dm-3 DDN,SFA 10000
[size=184G][features=1 queue_if_no_path][hwhandler=0][rw]
\_ round-robin 0 [prio=50][active]
  \_ 2:0:0:1 sdc 8:32 [active][ready]
\_ round-robin 0 [prio=10][enabled]
  \_ 1:0:0:1 sdo 8:224 [active][ready]
```

SSD Drives



EXAMPLE: Installing and Configuring GPFS

Script to Modify the Linux Transfer Size

```
[root@node-01 bin]# cat setDDNioparams.sh
#!/bin/bash
#
# setDDNioparams
#

maxsectkb=8192
readaheadkb=8192
nrrequests=512

DDNdevstmp="/var/tmp/DDNdevs.tmp"
DDNdevslog="/var/tmp/DDNdevs.log"

lsscsi > $DDNdevstmp

while read line; do
    devtype=`echo $line | cut -d" " -f3`
    if [ "$devtype" = "DDN" ]; then
        devname=`echo $line | cut -d"/" -f3`
        curmaxsectkb=`cat /sys/block/$devname/queue/max_sectors_kb`
        curreadaheadkb=`cat /sys/block/$devname/queue/read_ahead_kb`
        curnrrequests=`cat /sys/block/$devname/queue/nr_requests`
        if [ $curmaxsectkb -ne $maxsectkb -o \
            $curreadaheadkb -ne $readaheadkb -o \
            $curnrrequests -ne $nrrequests ]; then
            echo $maxsectkb > /sys/block/$devname/queue/max_sectors_kb
            echo $readaheadkb > /sys/block/$devname/queue/read_ahead_kb
            echo $nrrequests > /sys/block/$devname/queue/nr_requests
            echo "Device=$devname" | tee -a $DDNdevslog
            echo "Old max_sectors_kb=$curmaxsectkb" | tee -a $DDNdevslog
            echo "New max_sectors_kb=$maxsectkb" | tee -a $DDNdevslog
            echo "Old read_ahead_kb=$curreadaheadkb" | tee -a $DDNdevslog
            echo "New read_ahead_kb=$readaheadkb" | tee -a $DDNdevslog
            echo "Old nr_requests=$curnrrequests" | tee -a $DDNdevslog
            echo "New nr_requests=$nrrequests" | tee -a $DDNdevslog
            echo | tee -a $DDNdevslog
        fi
    fi
done < $DDNdevstmp
```



EXAMPLE: Installing and Configuring GPFS

Script to Modify the Linux Transfer Size

```
[root@node-01 queue]# pwd
/sys/block/sdm/queue
[root@node-01 queue]# ls -l
total 0
drwxr-xr-x 2 root root    0 Feb  5 10:32 iosched
-rw-r--r-- 1 root root 4096 Feb  5 13:28 iostats
-r--r--r-- 1 root root 4096 Feb  5 13:28 max_hw_sectors_kb
-rw-r--r-- 1 root root 4096 Feb  5 13:28 max_sectors_kb
-rw-r--r-- 1 root root 4096 Feb  5 13:28 nr_requests
-rw-r--r-- 1 root root 4096 Feb  5 13:28 read_ahead_kb
-rw-r--r-- 1 root root 4096 Feb  5 13:28 scheduler
[root@node-01 queue]# cat max_sectors_kb
512
[root@node-01 queue]# cat nr_requests
128
[root@node-01 queue]# cat read_ahead_kb
128
```

WARNING:

These values resort to their defaults every time a node is rebooted.
Therefore it is necessary to reset them everytime a node is rebooted.



13. Advanced Features in GPFS

The following pages examine newer features that are discussed in the *Advanced Administration Guide*.



Beginning with version 3.1 (and moving forward), GPFS is exploiting existing features and creating new features that make it more than an HPC file system... GPFS is becoming a general purpose clustered file system where HPC is a key and pervasive feature.

GPFS will never forget HPC!

The following pages examine some of the new (or newly exploited) GPFS features making it more suitable as a general purpose file system. Today, these features include

- ▶ ILM with Integrated HSM
- ▶ Robust NFS/CIFS support
- ▶ Scale-out File System (SoFS) - TBD
- ▶ Storage Virtualization
- ▶ Disaster Recovery
- ▶ Snapshots - TBD
- ▶ GPFS SNMP Support - TBD



Information Lifecycle Management (ILM)

GOALS

- ▶ Manage data over its life cycle ("cradle to grave")
- ▶ Keep active data on highest performing media and inactive data on tape of low cost, high capacity disk
- ▶ Migration of data is automatic and transparent to the client
- ▶ Lower levels can serve as backup for higher levels

Tier-1

- ▶ Performance Optimized Disk
 - e.g., FC, SAS disk
- ▶ Scratch Space

Tier-2


- ▶ Capacity Optimized
 - e.g., SATA
- ▶ Infrequently used files

Tier-3

- ▶ Local tape libraries

Tier-4

- ▶ Remote tape libraries



frequent use
smaller capacity
high BW/low latency
more expensive

infrequent use
larger capacity
lower BW
higher latency
less expensive



Archive vs. Backup



■ Archive: Maintain Only 1 Copy of File

- By definition, an **archive** requires multiple tiers of storage and maintains only 1 copy of a file in one of the tiers
- Example: in a combined disk/tape archive, a file resides either on disk or on tape, but *not both*!

■ Backup: Maintain a Second Copy of a File

- A **backup** system maintains a second copy of a file.
- Best practice guideline: the files of an archive should be backed up, including files archived on tape.
 - some archive products integrate backup into the archive function

■ GPFS ILM is primarily an archive tool, but...

- GPFS ILM policy supports file replication
 - while similar, replication is not the same thing as backup
- HSM products integrated with GPFS support both archive and backup functions



Information Lifecycle Management in GPFS

ILM provides

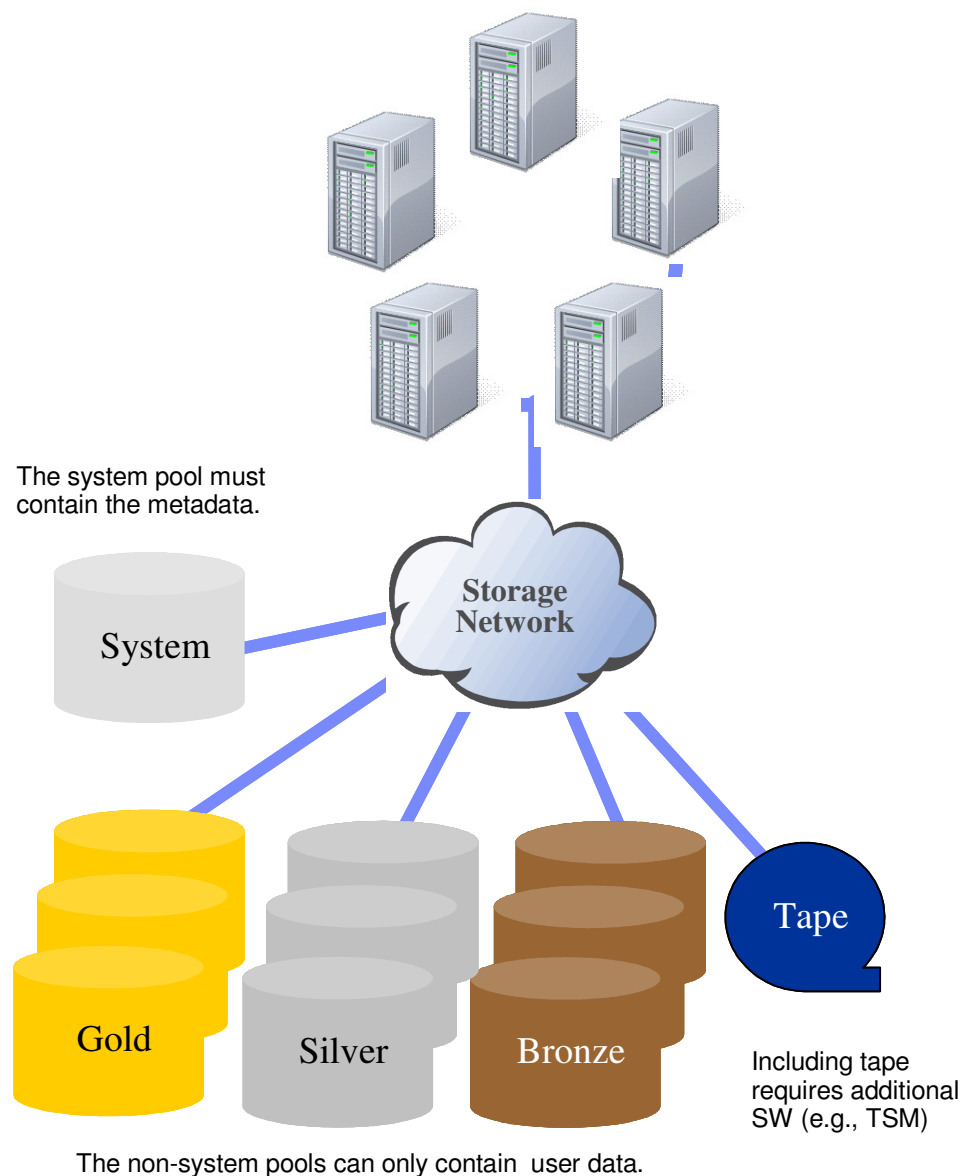
- ▶ Storage pool - group of LUNs
- ▶ Fileset - define subtrees of a file system
- ▶ Policies - for rule based management of files inside the storage pools

Examples of policy rules

- ▶ Place new files on fast, reliable storage, move files as they age to slower storage, then tape
- ▶ Place media files on video-friendly storage (fast, smooth), other files on cheaper storage
- ▶ Place related files together; e.g., for failure containment

Comments

- ▶ One global name space across pools of independent Storage
- ▶ Files in the same directory can be in different pools





Storage Pools

■ ILM Manages sets of storage called "storage pools"

■ What is a storage pool?

- A named subset of disks and tapes
 - within the context of GPFS, new appropriate SW to include tape (e.g., HPSS)
- Each file is assigned to a storage pool based upon *policy rules*
 - placement policies (where to place files upon creation)
 - migration policies (moving files from one pool to another)
 - deletion policies (removing files from the storage system)

■ What are they are good for?

- Tiered storage (files aged to slower/cheaper disk)
- Dedicated storage (e.g., per user or per project or per directory subtree)
- Failure containment
 - To limit the amount of data lost due to a failure
 - To bound the performance impact of RAID rebuild
- Appropriate use of special-purpose storage
 - Different RAID levels
 - Enterprise grade disk vs. consumer-grade disk
 - Multimedia friendly storage



GPFS Filesets

- **What they are:**

- A *named* subtree of a GPFS file system
- Somewhat like a distinct file system, i.e. a fileset can be unlinked without deleting it, and it can subsequently be linked using its name as a handle

- **What they are good for:**

- Filesets can have quotas associated with them (global; not per-pool).
 - Fileset quotas are independent of user and group quotas
- Filesets can be used to restrict the effect of policies to specific files

- **Side effects:**

- Unlinked filesets can confuse programs that scan the file system (e.g. incremental backup programs)
- Moving and linking between filesets is not allowed, in keeping with their being like little file systems



GPFS ILM/HSM Integration

■ GPFS Integrates its ILM Policies with tape based HSM Products

- GPFS extends its Information Lifecycle Management (ILM) functionality to integrate with HSM (Hierarchical Storage System) products.
 - A single set of policies is used to move data between GPFS storage pools and tape storage pools.
- Supported HSM products include
 - High Performance Storage System (HPSS)
 - Tivoli Storage Manager (TSM)
- Cool Feature: very fast file scans
 - 1 million files in 13 seconds
 - 1 billion files in 75 minutes

Tape is not dead!



Never underestimate the BW in a pickup load of magnetic tape!



... or a cargo plane for that matter.

A lab with insufficient tape BW was forced to use DHL to move 200 TB of data on disk!



HPSS/GPFS Integration Project

- **GPFS/HPSS is a collaborative project to develop synergy between IBM's General Parallel File System and the HPSS Collaboration's High Performance Storage System**

- **Purpose**

- To create a hierarchical disk and tape storage solution with unequaled parallelism and scalability
- Extend GPFS Information Lifecycle Management functionality to include tape

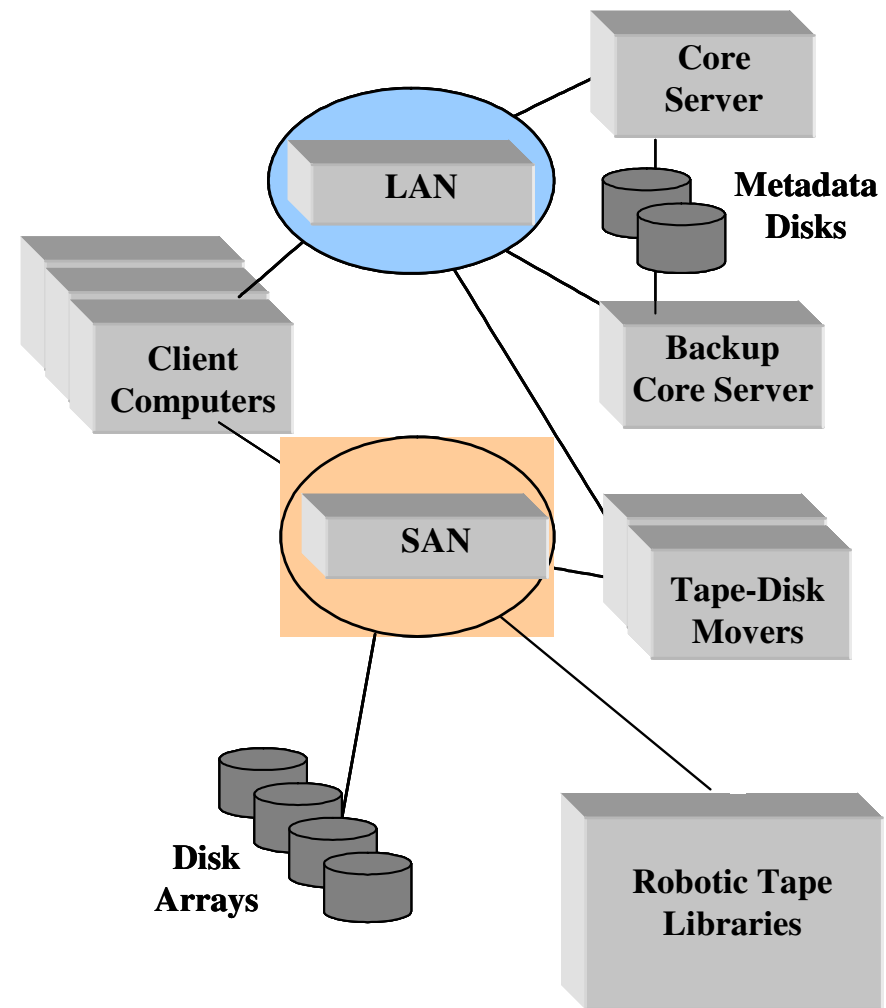
- **Participants**

- HPSS Collaboration member NERSC/Lawrence Berkeley Lab
- IBM Research, Almaden Lab
- IBM GPFS Product Development in Poughkeepsie NY
- IBM HPSS Development and Support in Houston TX



What is HPSS?

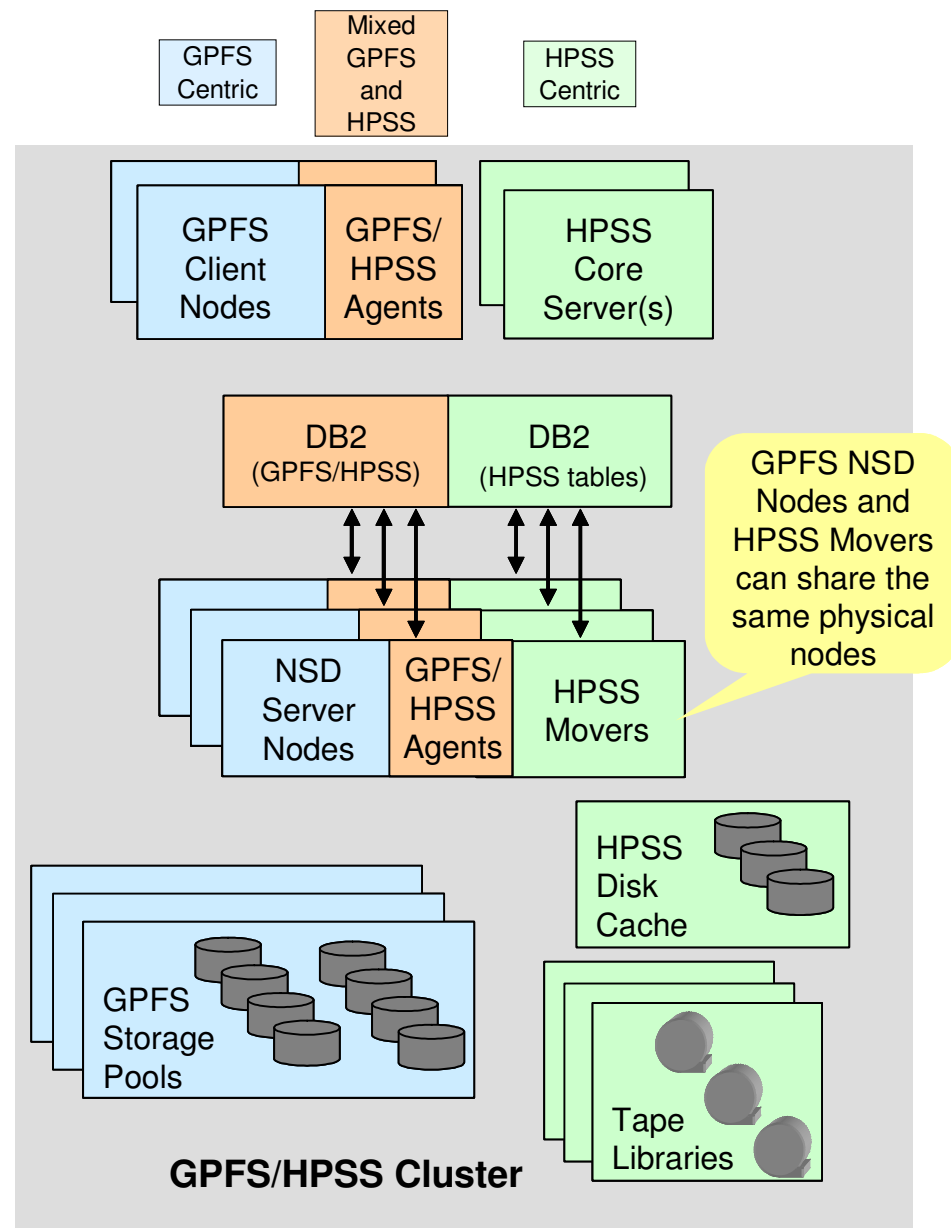
- ▶ HPSS
 - High Performance Storage System
- ▶ HPSS is a disk and tape hierarchical storage system with a cluster architecture similar in many ways to GPFS architecture
- ▶ HPSS can be used alone as a cluster hierarchical storage system or as the tape component of GPFS
- ▶ Versatile native HPSS interfaces:
 - Traditional HPSS APIs
 - Linux file system interface
 - New GridFTP interface available
- ▶ Rugged DB2 metadata engine assures reliability and quick recovery
- ▶ Like GPFS, HPSS supports horizontal scaling by adding disks, tape libraries, movers, and core servers to:
 - 10s of petabytes
 - 100s of millions of files
 - gigabytes per second
- ▶ Jointly developed by the five US Department of Energy labs and IBM





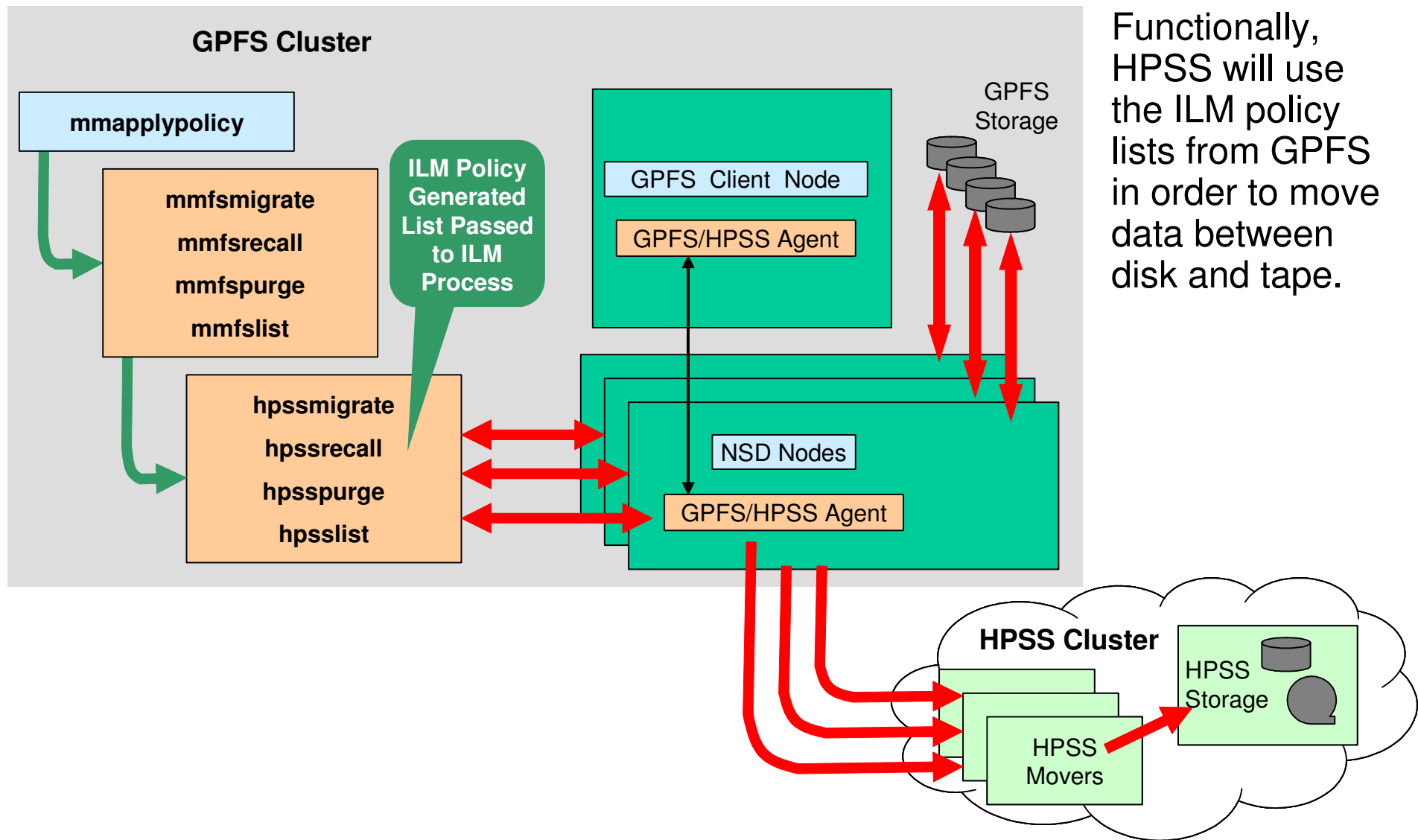
Integrated GPFS/HPSS Architecture

- ▶ GPFS/HPSS is software that connects GPFS and HPSS together under the GPFS ILM policy framework
- ▶ GPFS/HPSS agents (processes and daemons) run on the GPFS Session Node and I/O Manager Nodes
- ▶ GPFS/HPSS uses DB2 to contain a reference table that maps between GPFS file system objects and HPSS storage objects
- ▶ GPFS/HPSS is distributed with and supported by HPSS





GPFS/HPSS ILM Semantics





Sample GPFS ILM Policy Statements

Initial Placement

Rule name

Storage pool name
(corresponds to class of storage)

Fileset name
(corresponds to subdirectory)

Qualifiers

```
RULE 'SlowDBase' SET STGPOOL 'sata' FOR FILESET('dbase') WHERE NAME LIKE '%.data'
RULE 'SlowScratch' SET STGPOOL 'sata' FOR FILESET('scratch') WHERE NAME LIKE '%.mpg'
RULE 'default' SET STGPOOL 'system'
```

Movement by Age

```
RULE 'MigData' MIGRATE FROM POOL 'system' THRESHOLD(80,78)
  WEIGHT( TIME_SINCE_LAST_ACCESS ) TO POOL 'sata' FOR FILESET('data')
RULE 'HsmData' MIGRATE FROM POOL 'sata' THRESHOLD(95,80)
  WEIGHT( TIME_SINCE_LAST_ACCESS ) TO POOL 'hsm' FOR FILESET('data')
RULE 'Mig2System' MIGRATE FROM POOL 'sata' WEIGHT(ACCESS_TIME) TO POOL 'system' LIMIT(85)
  FOR FILESET('user','root') WHERE DAYS_SINCE_LAST_ACCESS_IS_LESS_THAN( 2 )
```

Rule to
move files
to HPSS

Lock in place

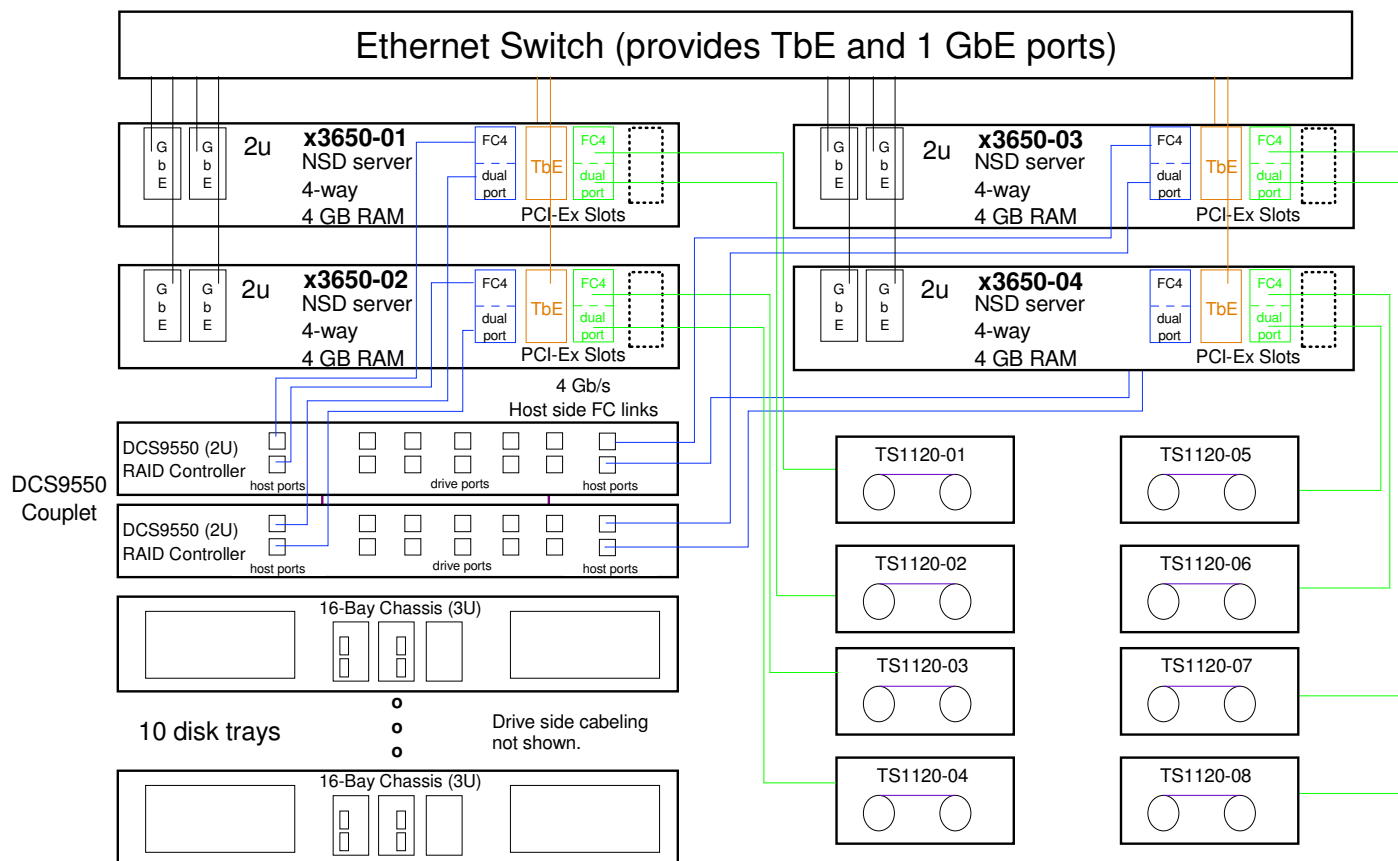
```
RULE 'ExcDBase' EXCLUDE FOR FILESET('dbase')
```

Life expiration

```
RULE 'DelScratch' DELETE FROM POOL 'sata' FOR FILESET('scratch') WHERE
  DAYS_SINCE_LAST_ACCESS_IS_MORE_THAN( 90 )
```



GPFS/HPSS Building Block



Aggregate BW per server

- TbE
 - peak < 750 MB/s
 - sustained < 700 MB/s
- HBA
 - peak < 780 MB/s
 - sustained < 700 MB/s
- NSD service (GPFS)
 - peak < 700 MB/s
 - sustained < 600 MB/s
- HPSS mover
 - peak < 200 MB/s
 - sustained < 100 MB/s

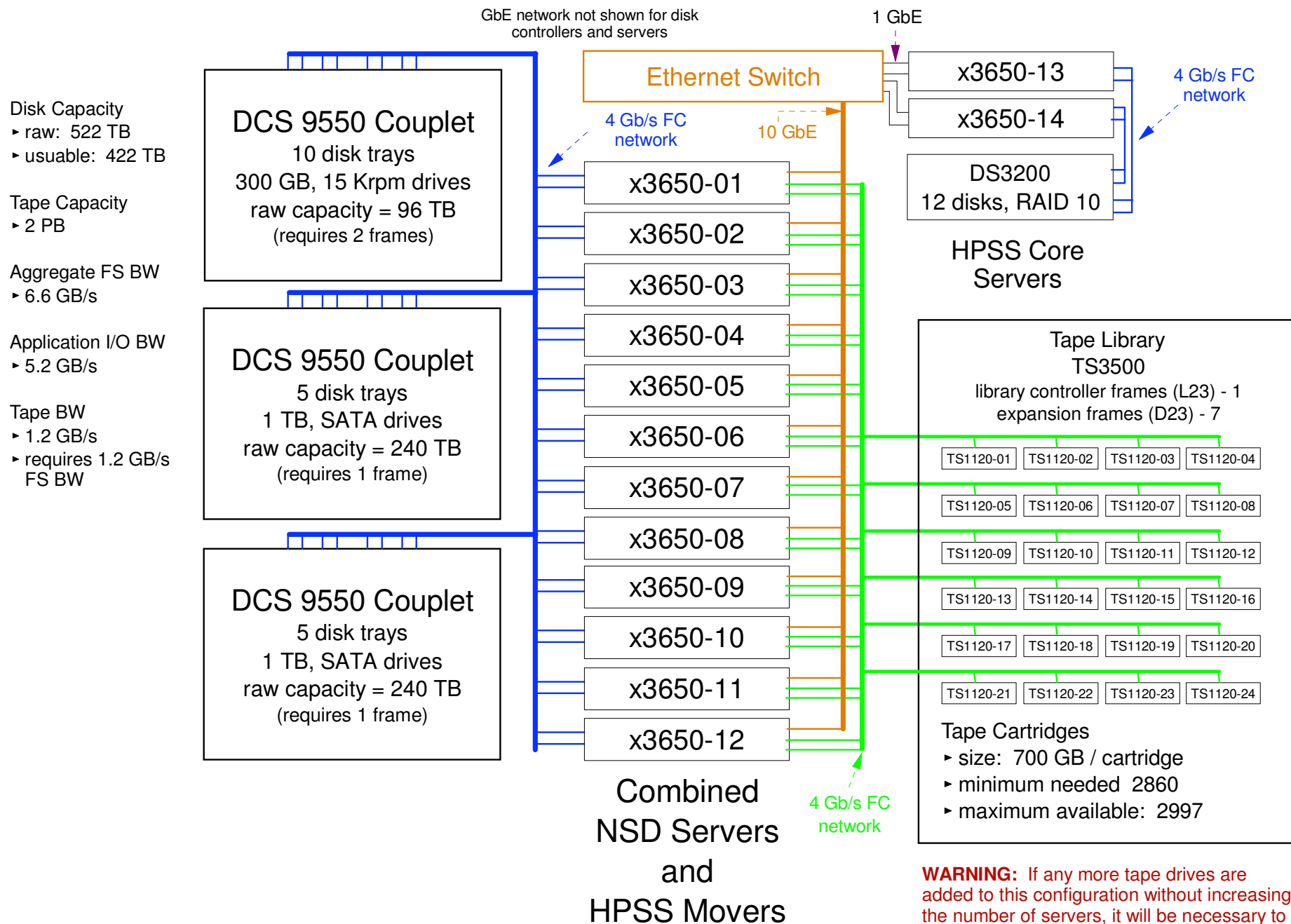
In terms of sustained rates assuming well designed I/O application in a production environment, applications will be able to draw *up to* 450 MB/s per server.

ANALYSIS

- Servers
 - 4 x3650s
 - 2 dual core sockets
 - at least 4 GB RAM
 - 2 dual port, 4 Gb/s HBA
 - 380 MB/s per port
 - 1 TbE (10 GbE) adapter per node
 - 800 MB/s per adapter
- Disk and Disk Controller
 - DCS 9550 couplet
 - 10 disk trays with 16 disks/tray
 - FC, 300 GB/disk @ 15 Krpm
 - 35 LUNs
 - 8+P RAID sets
 - capacity: raw = 48 TB, usable < 38 TB
 - Peak BW
 - write < 2.8 GB/s, read < 2.6 GB/s
 - Sustained BW
 - write < 2.6 GB/s, read < 2.2 GB/s
- Disk and Disk Controller
 - 8 x TS1120 (Jaguar)
 - Peak BW per drive < 100 MB/s
 - Sustained BW < 50 MB/s



A Multi-tiered GPFS/HPSS Solution





What is Tivoli Storage Manager?

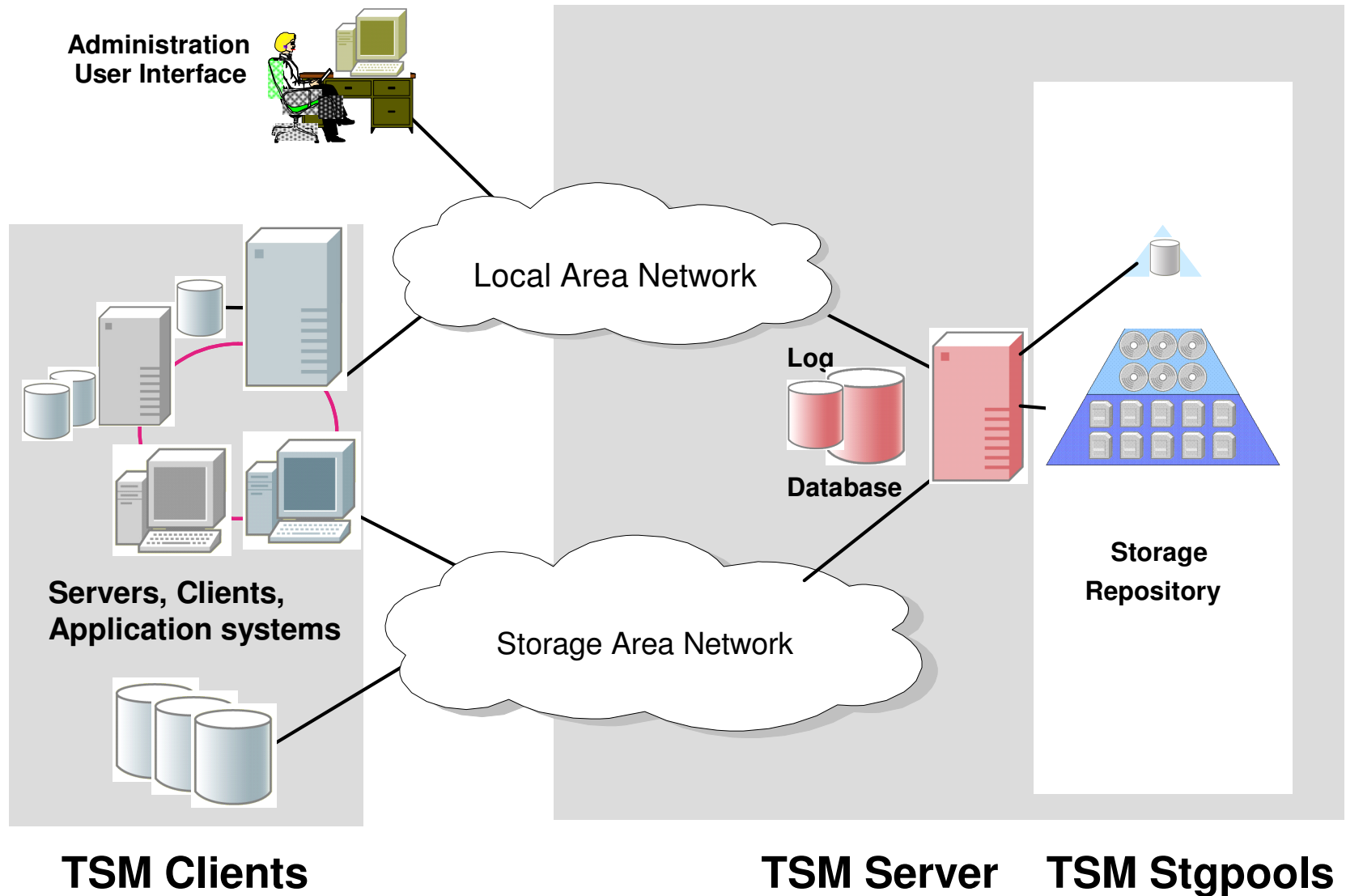
Tivoli Storage Manager (TSM) is a comprehensive software suite that manages storage. It provides

- ▶ Backup / restore
- ▶ Archive / retrieve
- ▶ Disaster recovery
- ▶ Database & application protection
- ▶ Space management (HSM)
- ▶ Bare machine recovery
- ▶ Continuous data protection
- ▶ Content Management

It is a client/server design with separate server products and client products implementing this list of functions.



TSM Architecture

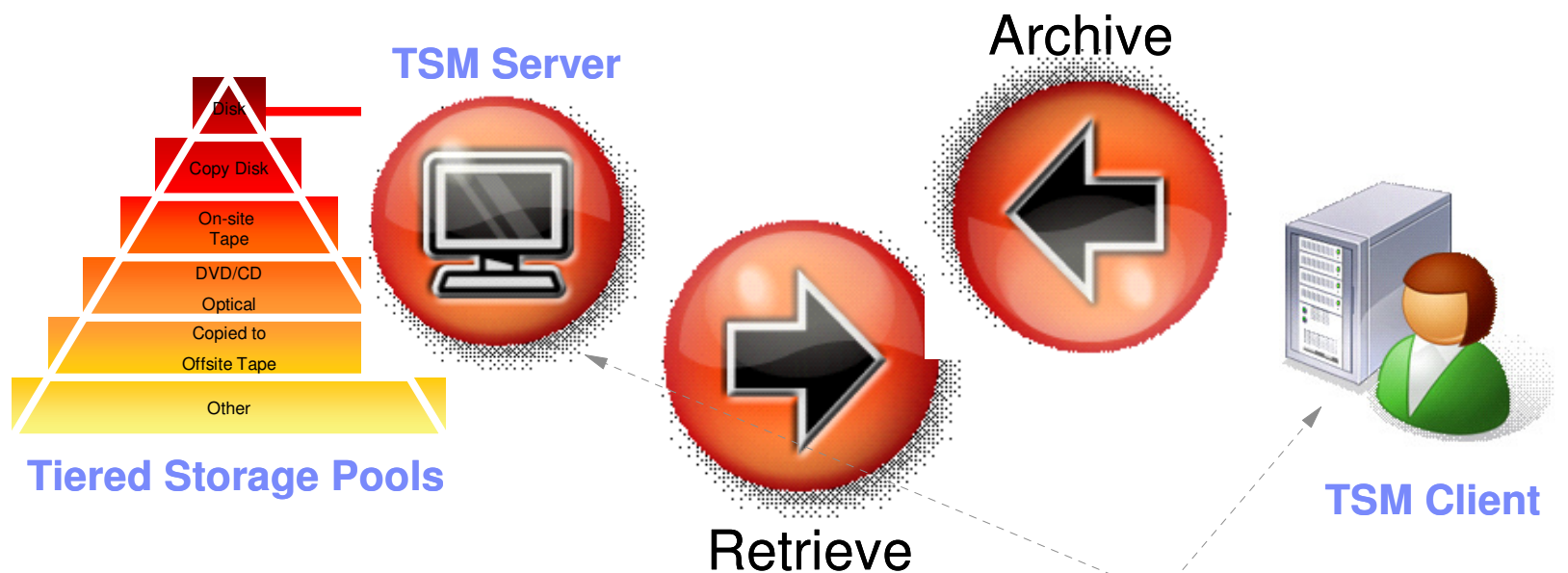


COMMENT

It is a common practice to combine the TSM server and client functions into a common node in GPFS clusters.



TSM Archive



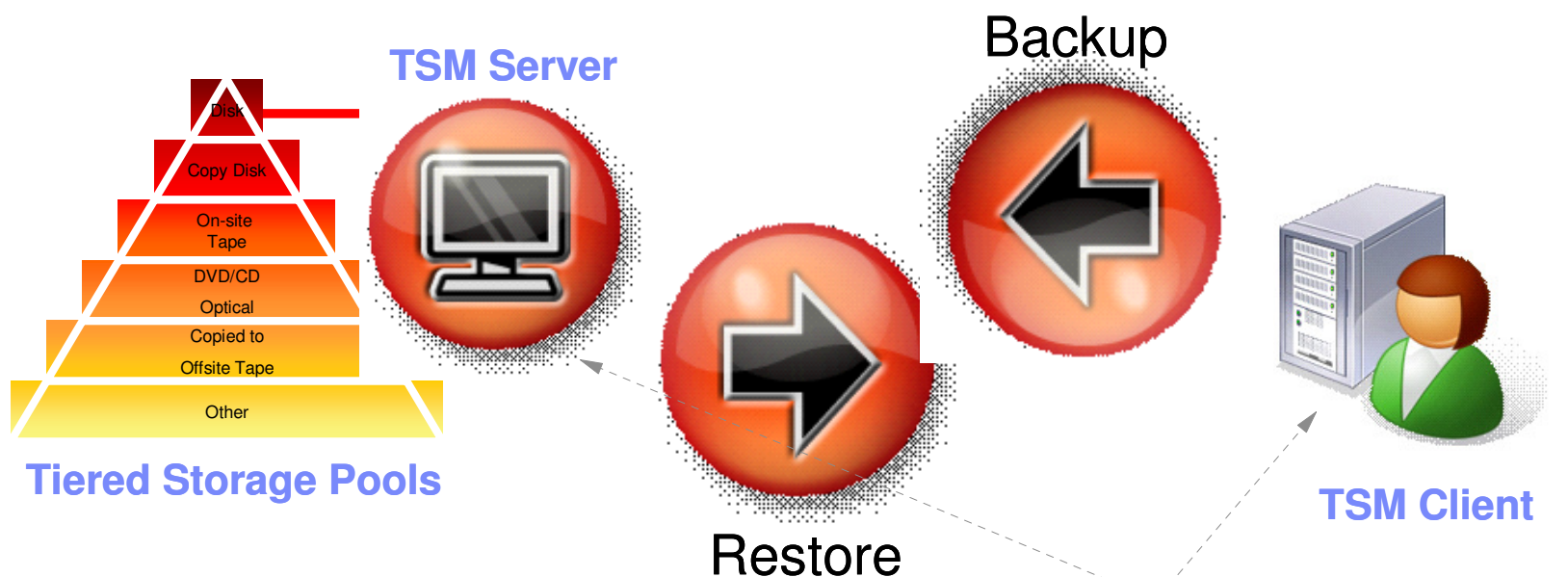
It is a common practice to combine the TSM server and client functions into a common node in GPFS clusters.

Archive Features

- Long-term storage
- Point in time copy
- Retention period
- Policy managed
- Index archives with descriptive metadata expedite locating historical information
- Allows focus to be placed on active data
 - Recover only active data
 - Reduce backup time by focusing on active files only



TSM Backup



It is a common practice to combine the TSM server and client functions into a common node in GPFS clusters.

Backup Features

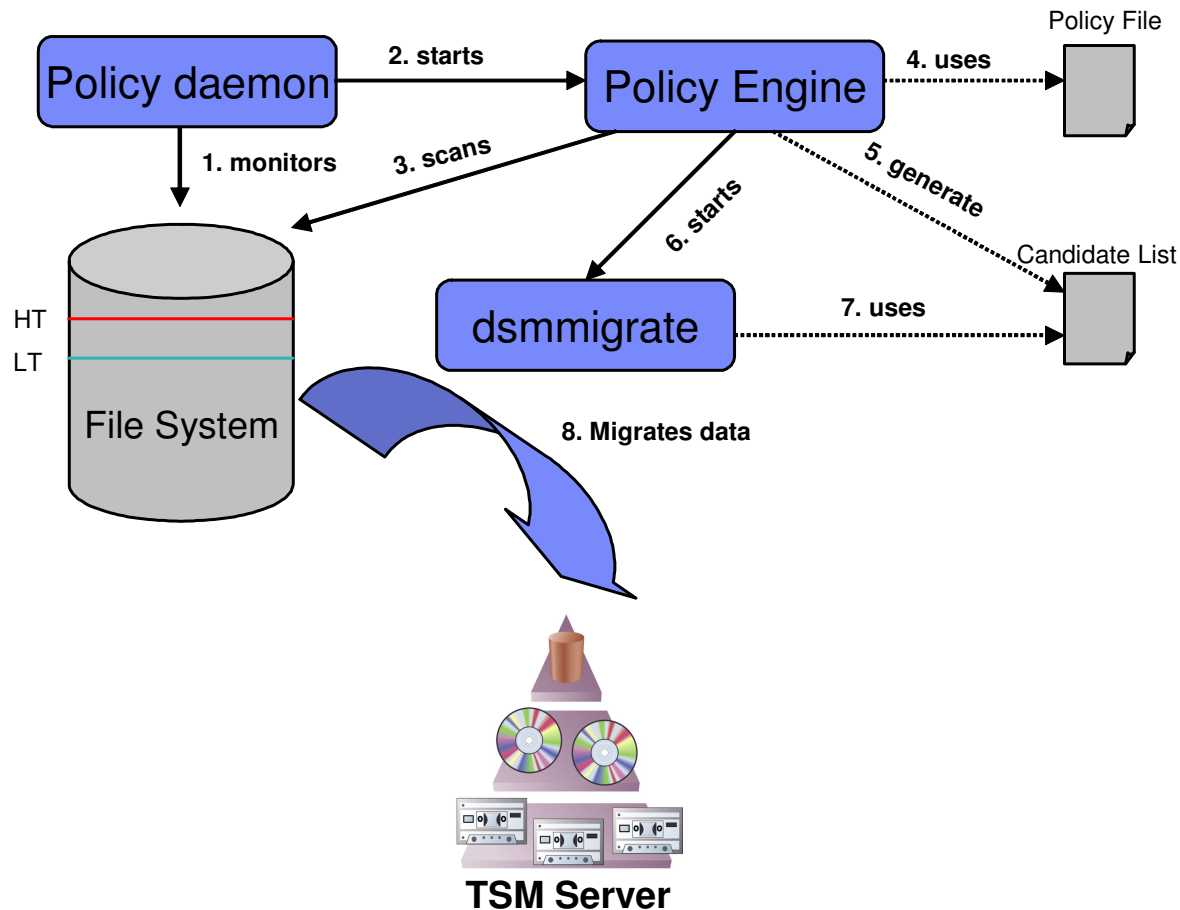
- ▶ Progressive incremental backup
 - Backup only new/changed files avoiding wasteful full backups
 - Data tracked at file level
 - Accurately restores files to a point in time
- ▶ Adaptive subfile differencing
- ▶ Volume level
- ▶ Multiple versions kept
- ▶ Policy managed
- ▶ System assisted restore
- ▶ Automated scheduling

COMMENT:

For TSM, a recommended best practice is to explicitly backup archived data.



Integrated GPFS/TSM Architecture



The Process

- ▶ GPFS policy daemon monitors HT/LT based on enabled policy.
- ▶ Policy daemon starts policy engine.
- ▶ Policy engine scans file system and generates candidate list based on enabled migration policy.
- ▶ dsmmigrate is called and migrates all files in candidate list to the TSM server.



Setting up Integrated TSM/GPFS Migration

To enable the TSM/HSM client to integrate with the GPFS policy engine it is necessary to

- ▶ Install GPFS 3.2 and TSM/HSM 5.5 as described in the books
- ▶ The HSM automigration capability needs to be disabled. This is done by the following command :

```
dsmmigfs add -HT=100 <file_system>
```

```
dsmmigfs update -HT=100 <file_system>
```

- ▶ Add an external HSM Pool to your placement policy file.

Example:

```
RULE EXTERNAL POOL 'hsm pool'
```

```
EXEC '/var/mmfs/etc/mmpolicyExec-hsm.sample' OPTS '-v'
```

- ▶ Add a threshold migration rule to your placement policy file

Example:

```
RULE 'HsmData' MIGRATE
```

```
FROM POOL 'StoragePool1'
```

```
THRESHOLD (90, 80, 70)
```

```
WEIGHT ( CURRENT_TIMESTAMP - ACCESS_TIME )
```

```
TO POOL 'hsm pool'
```

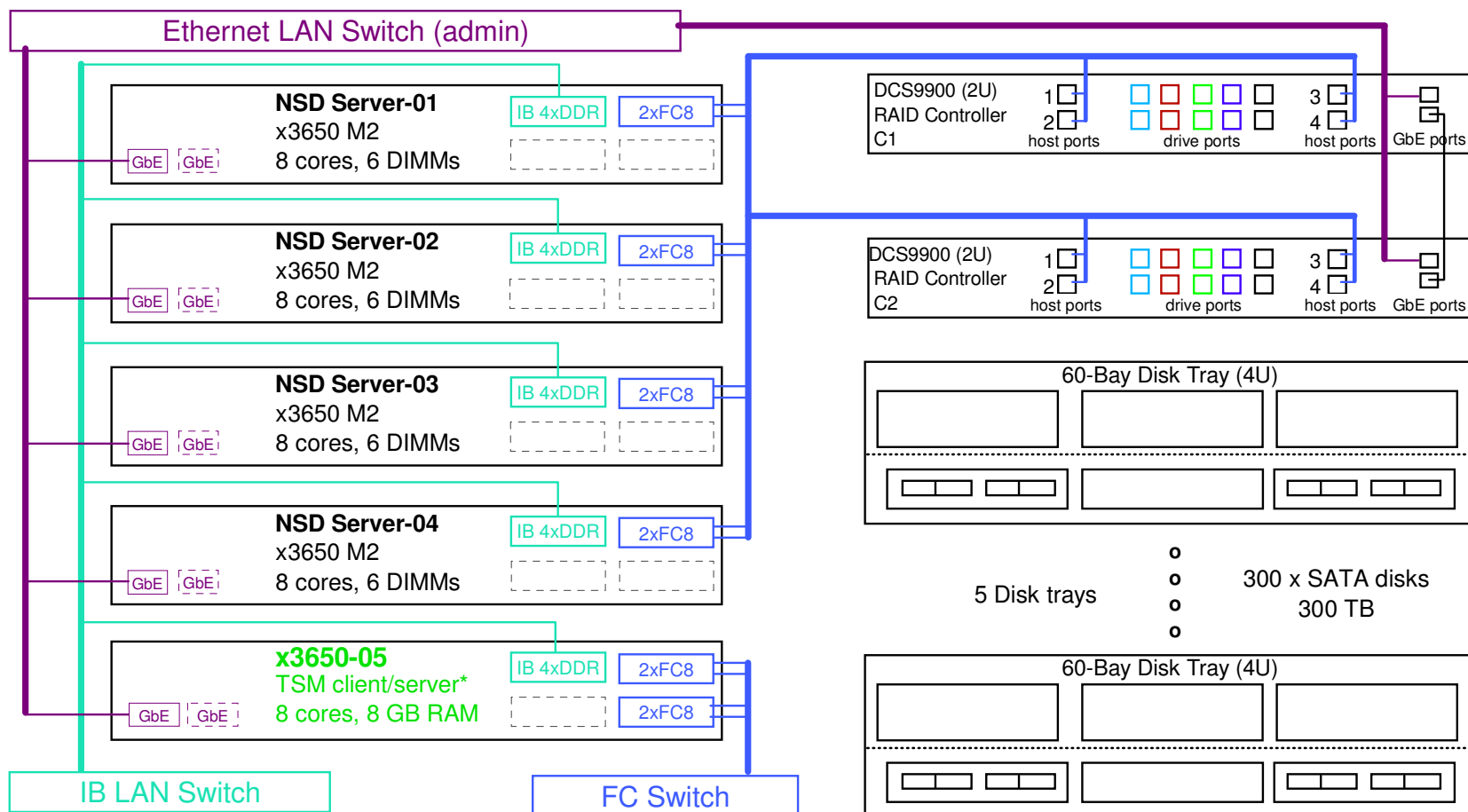
```
WHERE FILE_SIZE > 1024
```

- ▶ Install placement policy with

```
mmchpolicy <filesystem> <policy file>
```



A Two-tiered GPFS/TSM Solution



2nd Tier TSM Tape Archive

- ▶ 10 x LTO-4 drives (TS1040)
 - 1 FC4 port per tape drive
 - at most 120 MB/s per tape drive
 - assumes no compression
- ▶ 800 x 800 GB cartridges
 - usable capacity < 640 TB
 - assumes no compression
- ▶ aggregate data rate < 1 GB/s
 - provides 4 GB/s application to file system BW and 1 GB/s file system to TSM BW

4xDDR IB HCA (Host Channel Adapter)

- ▶ Peak data rate per HCA < 1500 MB/s
- ▶ Require RDMA

FC8 (single port 8 Gbit/s Fibre Channel)

- ▶ Peak data rate 2 FC8 HBAs < 1500 MB/s

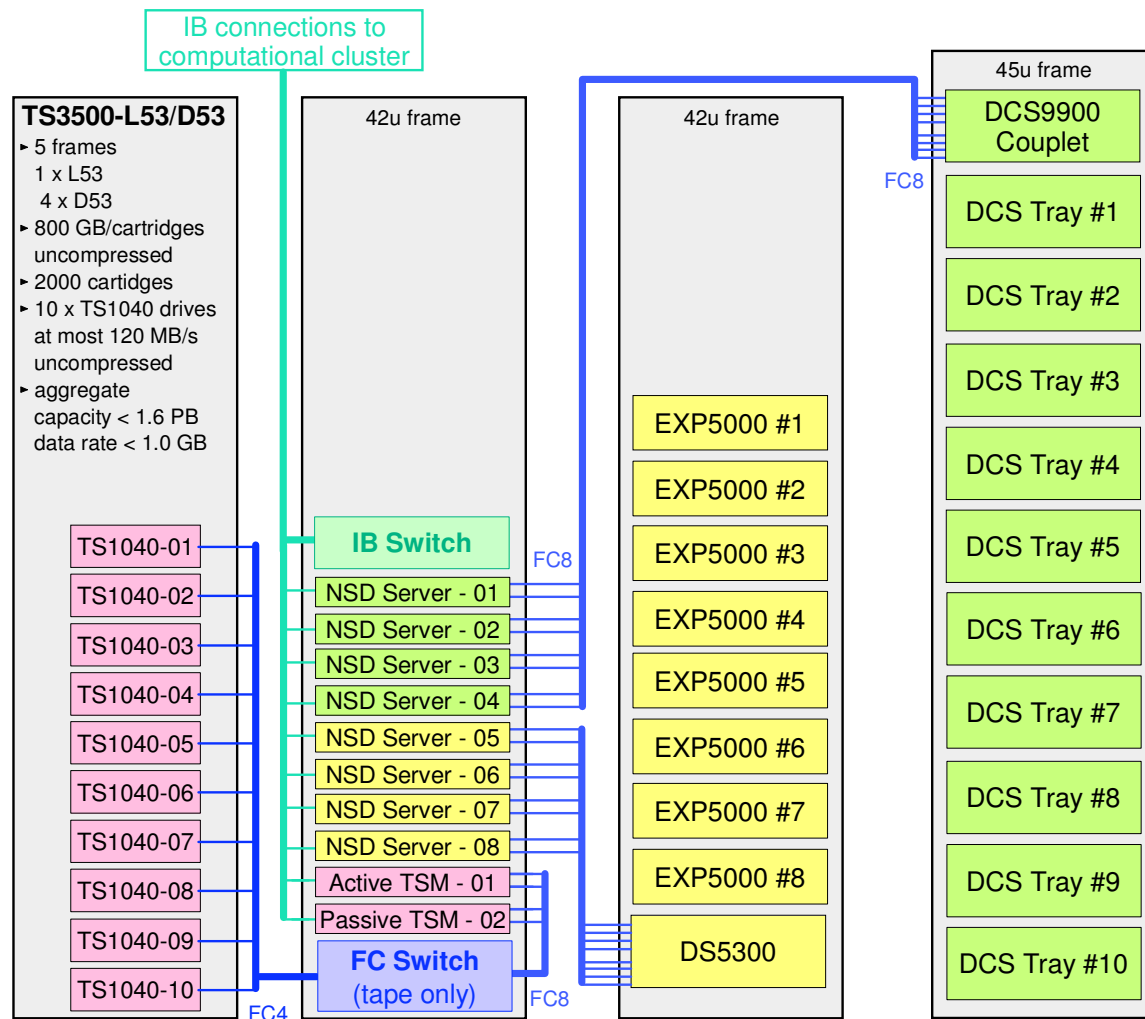
DCS9900 Performance

- ▶ streaming data rate < 5000 MB/s
- ▶ noncached IOP rate < 40,000+ IOP/s

★ In a production system there would normally be 2 TSM client/server systems; the active one and a passive one for redundancy.



A Three-tiered GPFS/TSM Solution



Tier 3 - archive/backup

- Capacity
1.6 PB
- ILM Bandwidth
receive < 0.8 GB/s
restore < 0.2 GB/s

Tier 1 - scratch

- Capacity < 58 TB
128 x 450 GB drives
15 Krpm FC drives
- Application Bandwidth
write < 1.0 GB/s
read < 2.0 GB/s
- ILM Bandwidth
transfer < 1.0 GB/s
restore < 0.5 GB/s

Tier 2 - archive

- Capacity < 600 TB
600 x 1 TB drives
SATA drives
- ILM Bandwidth
receive from tier 1 < 1.0 GB/s
transfer to tier 3 < 1.0 GB/s
restore to tier 1 < 0.5 GB/s
- Unused Bandwidth < 2.5 GB/s

COMMENTS

- Tier 1
Scratch storage used for application processing
- Tier 2
Archive storage indirectly accessed by applications
- Tier 3
Archive/backup storage indirectly accessed by applications.

Footnotes

- The passive TSM client/server is a "hot spare" backup for the active TSM client/server.
- It is assumed that the NSD server and TSM client/server nodes are x3650 M2. Alternatively, the P6-p520 could be used instead. Likewise, the IB LAN could be replaced by TbE where each server has a channel bonded 2xTbE.
- There is upto 2.5 GB/s of unused bandwidth in the tier 2 storage. If applications directly access this storage to create data, then additional tape bandwidth is needed archive and/or backup this data. This will require more TSM client/server nodes which means creating a 2nd TSM repository, or selecting a more powerful node that can handle the increased bandwidth.
- Generally, the archive rate = data creation rate which is assumed to be 80% write rate for this example. (n.b., not all written data is retained)



HPSS vs. TSM

Which is Best?



■ Capacity

- HPSS is sized and priced for systems with over 1 PB of storage.
- TSM has an upper limit of 1 PB per TSM instance.

■ Backup

- HPSS integrates backup into the archive function.
- TSM requires a separate backup procedure in addition to the archive function.

■ Parallelism

- HPSS is designed as a parallel archive tool; it supports multiple tape servers (i.e., "tape movers") per HPSS instance.
- TSM is not parallel; to scale TSM beyond a single server requires multiple TSM instances.

■ Metadata management

- HPSS requires a separate metadata subsystem (e.g., 2 "core servers" plus external disk storage for its metadata database).
- TSM integrates the metadata operations into its server operations.

■ Market segment

- HPSS was designed for the high end HPC market by a consortium of HPC labs.
- TSM was designed for commercial applications, but is commonly adapted to scientific and technical environments.

Yesterday

► NFS was a



with GPFS on Linux!

Today...

- Improved performance, robustness, server farm features for NFS
- Clustered NFS (CNFS)
 - Provides high availability NFS server functionality using GPFS
 - only under Linux
 - RHEL and SuSE
 - includes Linux on pSeries (LoP)
 - Serves most any NFS client
 - examples: AIX, Linux, Solaris, *etc.*



Clustered NFS (CNFS)

■ CNFS

- Provides a High Availability implementation of NFS under Linux
- Two primary uses
 - can be used like a NAS device providing robust NFS service
 - facilitate a transition from a current NFS deployment to GPFS

■ Create a GPFS cluster whose nodes can provide NFS service

- Generally use NSD servers as the NFS servers

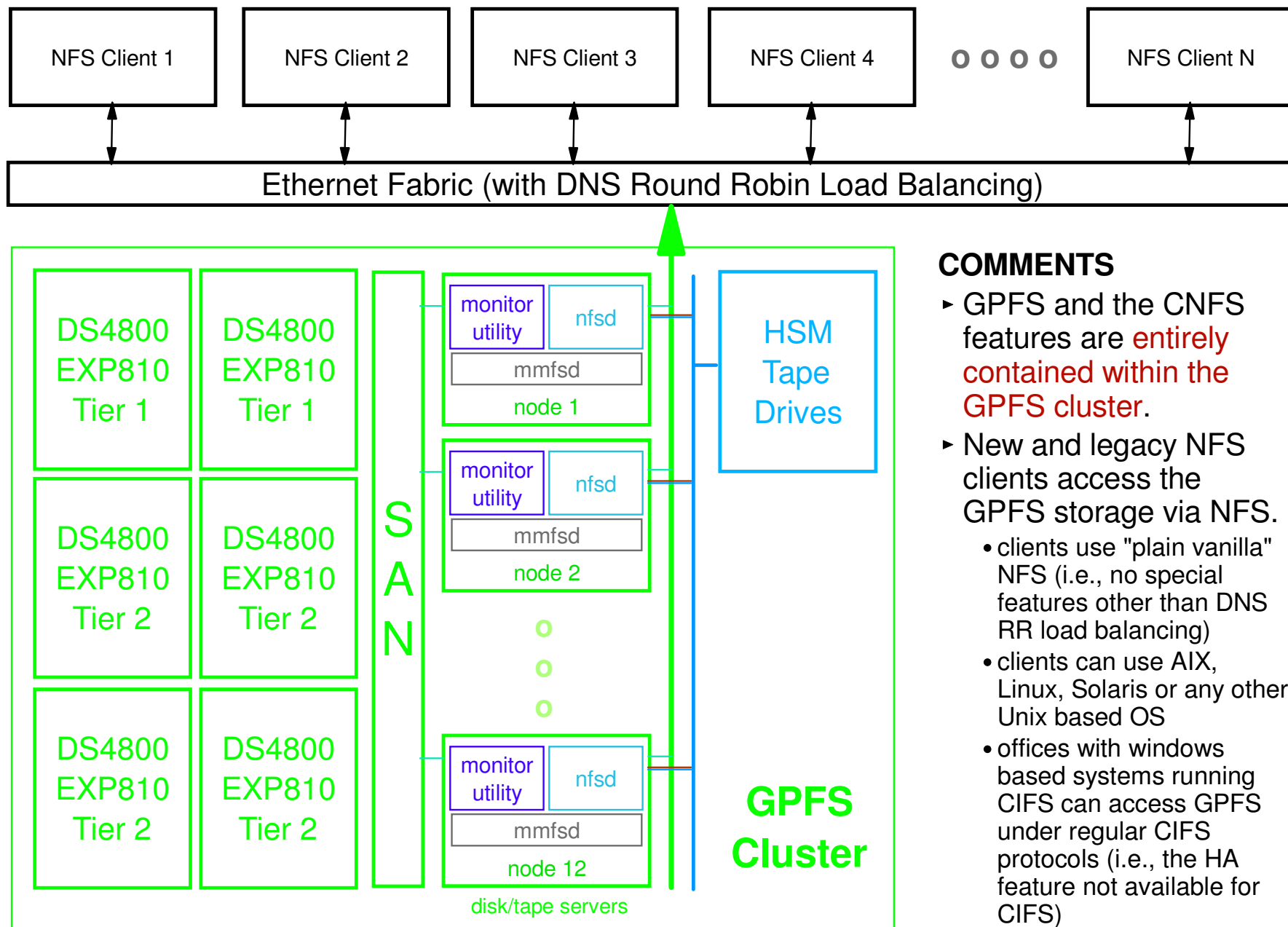
■ The CNFS feature provides the following features

- Monitoring
 - all NSD/NFS server nodes monitor key components of the "stack" and upon a failure initiate failover recovery
- Failover
 - exploits IP address failover and GPFS failover to perform NFS recovery
- Load balancing
 - utilizes DNS round robin
- Honors NFS lock consistency
 - NLM locks are passed thru to GPFS lock manager



CNFS

"A Picture is Worth a Thousand Words"



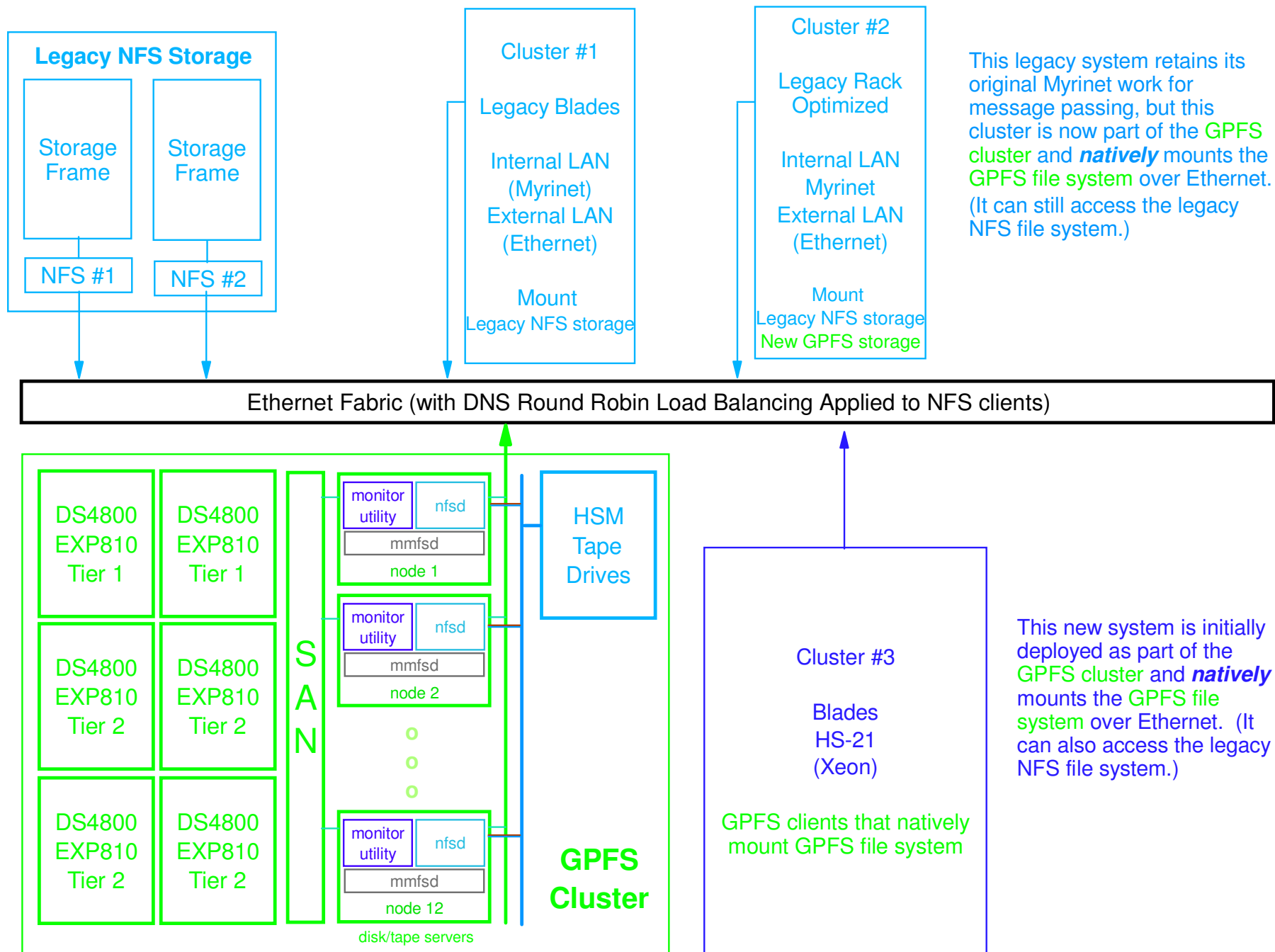
COMMENTS

- ▶ GPFS and the CNFS features are **entirely contained within the GPFS cluster**.
- ▶ New and legacy NFS clients access the GPFS storage via NFS.
 - clients use "plain vanilla" NFS (i.e., no special features other than DNS RR load balancing)
 - clients can use AIX, Linux, Solaris or any other Unix based OS
 - offices with windows based systems running CIFS can access GPFS under regular CIFS protocols (i.e., the HA feature not available for CIFS)



CNFS

"A Picture is Worth a Thousand Words"





TBD

The core of GPFS continues to operate on Unix UID/GID values. Windows GPFS nodes perform the task of mapping to Windows SIDs: explicit Unix-Windows ID maps are defined in Active Directory; implicit (default) maps for Windows SIDs are created from a reserved range of UID/GID values; and unmapped Unix IDs are cast into a foreign domain for Windows. Explicit maps persist only in the Active Directory. Implicit maps persist in the file system.

(So how did we do on that explainable principle?)



CNFS Details

Monitoring

Every node in the CNFS cluster runs an NFS utility that monitors GPFS, NFS, and networking components on the node. Upon failure detection and based on your configuration, the monitoring utility might invoke a failover.

Failover

As part of GPFS recovery, the CNFS cluster failover mechanism is invoked. It transfers the NFS serving load that was served by the failing node to another node in the CNFS cluster. Failover is done using recovery groups to help choose the preferred node for takeover. The failover mechanism is based on IP address failover. In addition, it guarantees NFS lock (NLM) recovery.

Load balancing

CNFS supports a failover of all of the node's load together (all of its NFS IP addresses) as one unit to another node. However, if no locks are outstanding, individual IP addresses can be moved to other nodes for load balancing purposes. CNFS is based on round robin DNS for load balancing of NFS clients among the NFS cluster nodes.



GPFS Supports Storage Virtualization

- ***Storage Virtualization:*** software provides a consistent "look and feel" for different disk technologies
- **GPFS provides a common command interface to manage any supported storage technology**
 - Commands of particular interest:
 - ILM
 - mmadddisk, mmdeldisk, mmaddnode, mmdelnode
 - GPFS client access is made consistent by the NSD layer
 - GPFS can be used as a virtualization tool for NFS clients
 - does not require CNFS
- **Other storage virtualization tool: SVC**
 - SVC is **not** necessary to achieve virtualization if GPFS is being used.



Storage Virtualization

An Abstract Example with GPFS Using Best Practices

This example illustrates two **possible** ways to virtualize very different types of storage technologies.

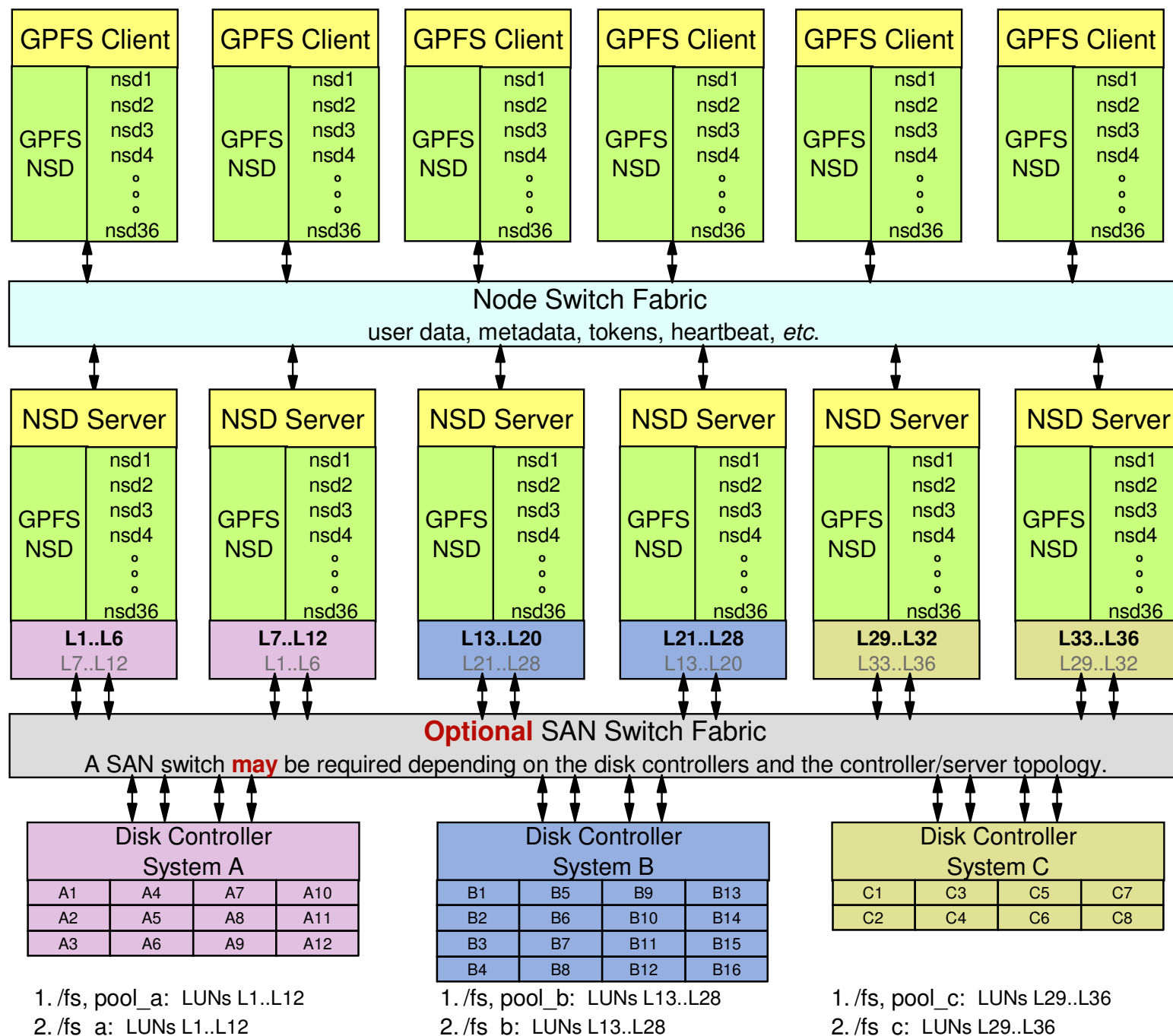
Segregate different disk systems to improve performance

1. Use GPFS ILM placing each disk system into its own storage pool
2. Different disk systems are segregated into separate file systems.

Segregation is not required, but is merely a "best practice".

COMMENT:

GPFS can virtualize *almost* any block device under a common rubric.





Storage Virtualization

Another Abstract Example with GPFS Using Best Practices

This example illustrates two **possible** ways to virtualize very different types of storage technologies.

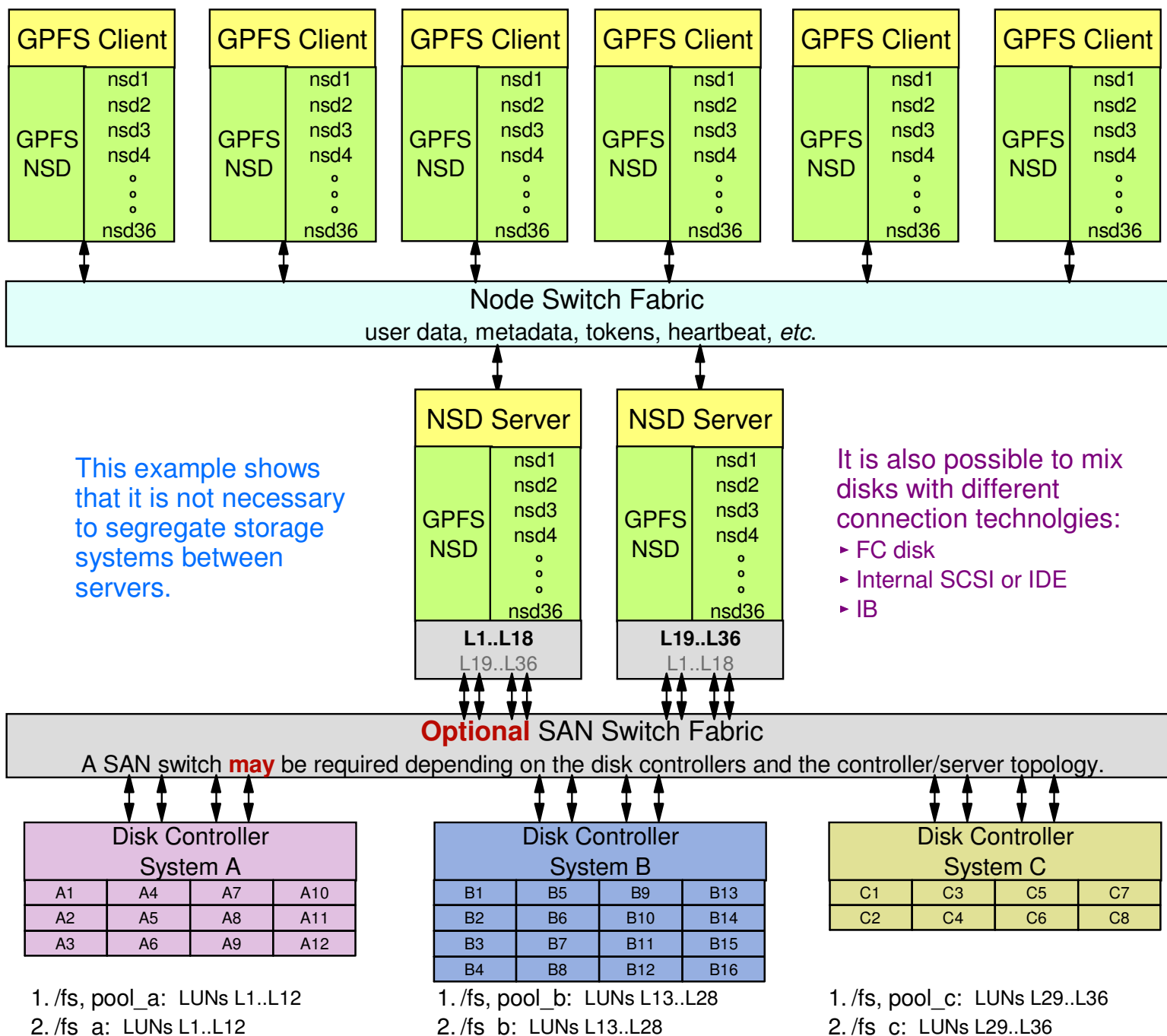
Segregate different disk systems to improve performance

1. Use GPFS ILM placing each disk system into its own storage pool
2. Different disk systems are segregated into separate file systems.

Segregation is not required, but is merely a "best practice".

COMMENT:

GPFS can virtualize *almost* any block device under a common rubric.





Storage Virtualization

GPFS Can Provide Storage Virtualization to Non-GPFS Clusters

This example illustrates two **possible** ways to virtualize very different types of storage technologies.

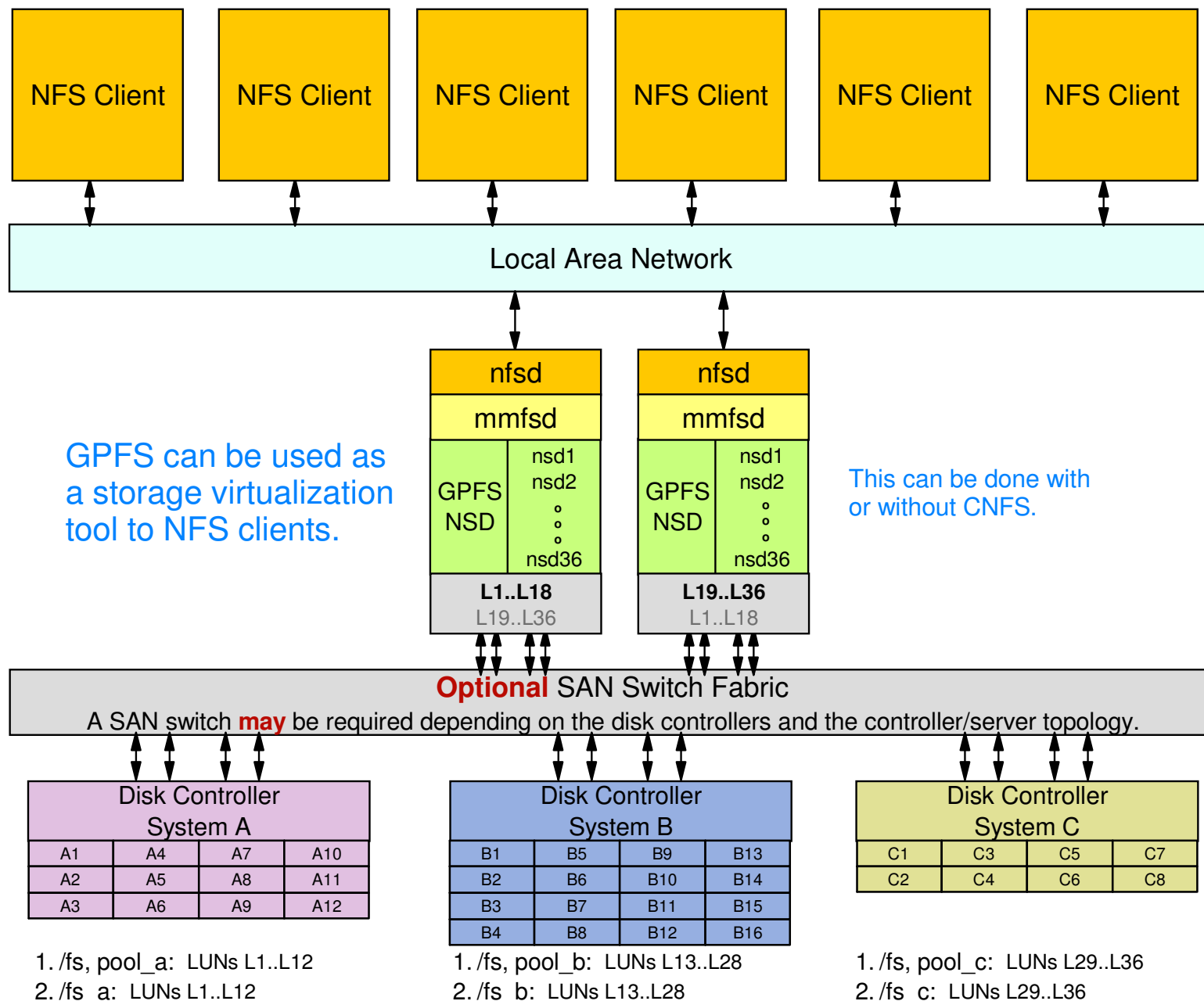
Segregate different disk systems to improve performance

1. Use GPFS ILM placing each disk system into its own storage pool
2. Different disk systems are segregated into separate file systems.

Segregation is not required, but is merely a "best practice".

COMMENT:

GPFS can virtualize *almost* any block device under a common rubric.





Operational Continuity under Adverse Conditions

"Disaster Recovery"

■ General Concept

- Redundant storage technology is deployed that enables operational continuity in the event of a disaster or other unrecoverable error. This is achieved by maintaining duplicate copies of a data set at two different locations, each with a redundant storage system, and enabling the "other" storage system to take over responsibility.
- This commonly called "disaster recovery" in the literature.

■ Two Proposed Alternatives

- Software based solution
 - synchronous mirroring using GPFS replication
- Hardware based solution
 - **a**synchrnous mirroring using DS4000 *Enhanced Remote Mirroring*



TWAIN
Technology Without An
Interesting Name



Operational Continuity under Adverse Conditions

First Alternative

Synchronous Mirroring Using GPFS Replication

- **Software replication using GPFS mirroring**
 - the replica is not a sector image copy
- **Reduces risk of permanent data loss or inconsistency**
- **Provides automatic failover**
- **Failback procedures vary according to the cause and magnitude of the failure**
- **Allows an active/active configuration**
- **Synchronous operation explicitly exposes BW and latency issues**
- **Does not require inter-SAN connectivity nor local SAN switches**
- **Included in GPFS at no extra cost**





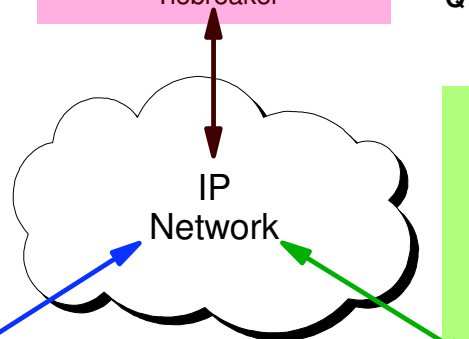
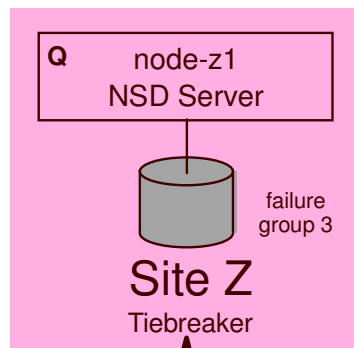
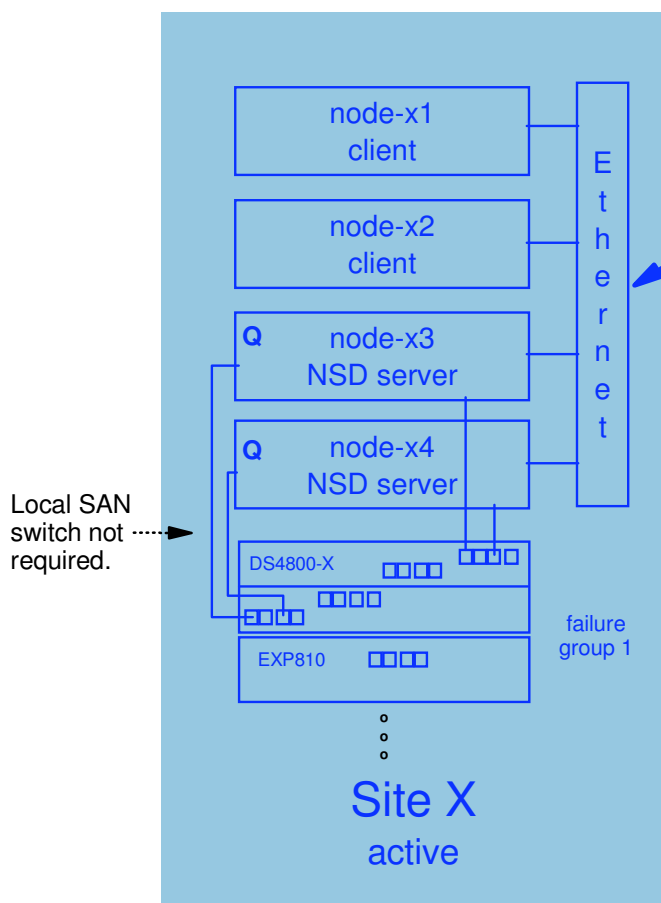
Synchronous Mirroring Using GPFS Replication

Best Practice

- Maintain 3 sites at geographically distributed locations
- Maintain 3 sub-sites at same geographic location, but supplied by power from 3 independent sources

Requirement

- This infrastructure is deployed as a single GPFS cluster (*n.b.*, can not use GPFS multi-cluster feature)



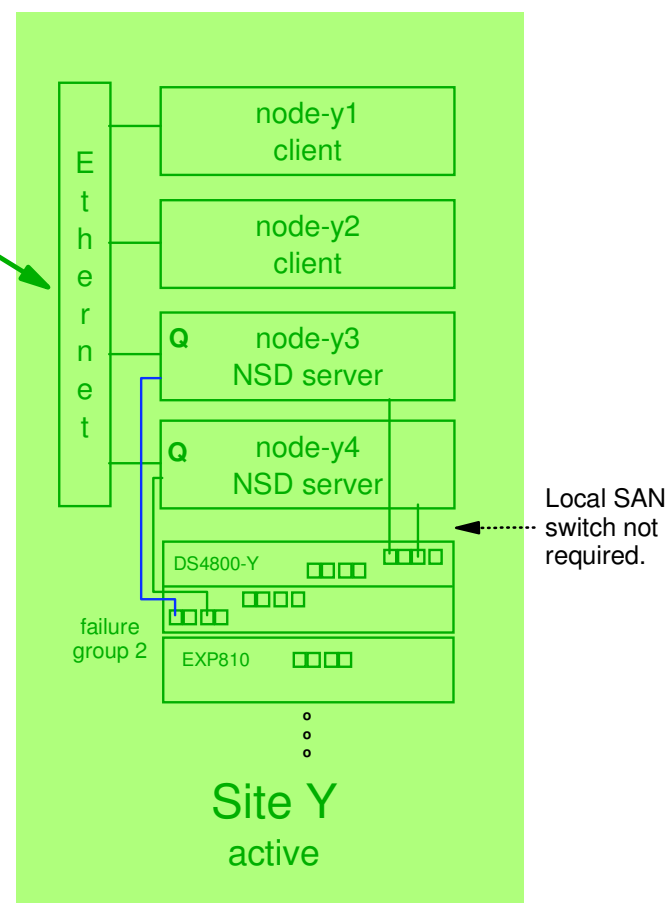
GPFS uses *mirroring* to *synchronously* copy user data and meta data to both sites (e.g., the write() call blocks until the data is copied to both sites).

- minimizes the risk of permanent data loss
- must provide sufficient BW between sites so regular I/O operations are not impeded
- must have extra BW to allow quick recovery (*n.b.*, restriping) following a failure
- Does not require SAN switches or SAN connectivity between sites

Maintaining Quorum

- Site C guarantees quorum under degraded operation. It does not participate in regular operations.
- The disk contains only file system descriptor info (descOnly) needed to maintain quorum
- Site-C is **not** required, but it improves your chances of surviving an outage automatically by 50%

Q designates quorum nodes





Operational Continuity under Adverse Conditions

Second Alternative

DS4800 Enhanced Remote Mirroring (ERM)

■ ERM is hardware based mirroring provided by the IBM DS4000

■ Three alternatives

- metro mirroring
 - full **s**ynchronous copy requiring a response from the secondary storage device to continue
 - distance limited to metropolitan areas
- global mirror
 - **a**synchronous copy with guaranteed in order delivery and the ability to create a “consistency group” of LUN’s that will be mirrored together
 - distance limited (but not as much as metro mirroring)
- global copy
 - **a**synchronous copy with no guaranteed in order delivery and no consistency groups
 - very long distances theoretically possible



Enhanced Remote Mirroring

- **Hardware mirroring integrated with GPFS**
 - focus on the **a**synchronous methods
- **Asynchronous operation avoids latency issues**
 - caution: BW must be able to keep up with data creation rate as well as re-synchronization following an outage in a timely manner after a failure
- **Failover can be automated with scripts**
- **Failback procedures vary according to the cause and magnitude of the failure**
- **Deployed in an active/passive configuration**
 - LUNs at the secondary site can **not** be written to
- **Increased risk of permanent data loss**
 - Due to **a**synchronous operation, if the primary site fails, data that has not been replicated to the secondary site will be lost.
- **Requires inter-SAN connectivity and local SAN switches**
- **Premium feature that must be licensed**



Enhanced Remote Mirroring

Best Practice

- ▶ Maintain 3 sites at geographically distributed locations
- ▶ Maintain 3 sub-sites at same geographic location, but supplied by power from 3 independent sources

Requirement

- ▶ This infrastructure is deployed as a single GPFS cluster (*n.b.*, can not use GPFS multi-cluster feature)

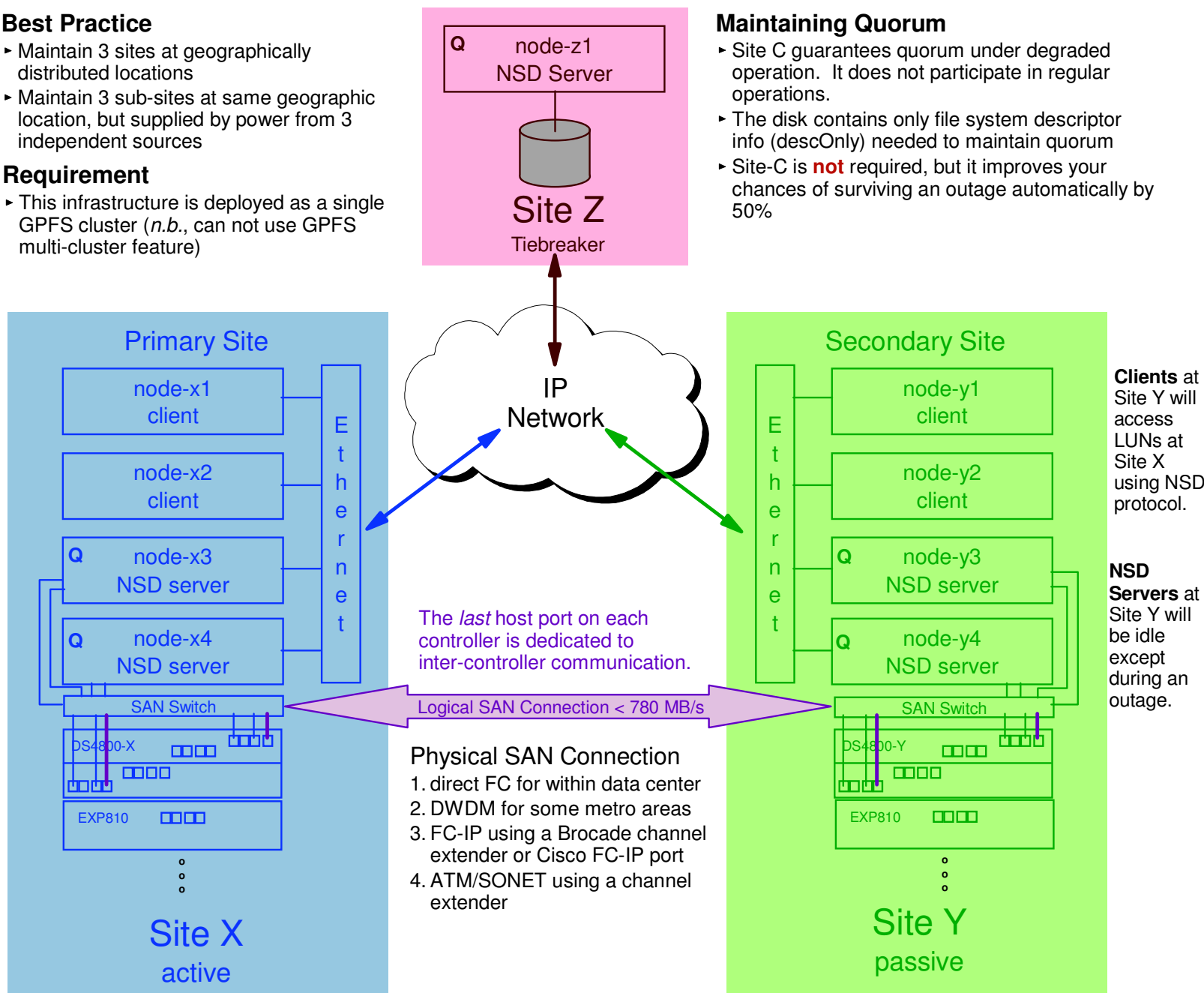
Maintaining Quorum

- ▶ Site C guarantees quorum under degraded operation. It does not participate in regular operations.
- ▶ The disk contains only file system descriptor info (descOnly) needed to maintain quorum
- ▶ Site-C is **not** required, but it improves your chances of surviving an outage automatically by 50%



Some Terms

- ▶ primary/secondary storage system
- ▶ mirroring storage controller pair
- ▶ mirror FC connection
- ▶ primary/secondary logical drive
- ▶ mirrored logical drive pair
- ▶ mirror role
- ▶ role reversal
- ▶ write consistency group
- ▶ full synchronization





Enhanced Remote Mirroring Integration with GPFS

■ ERM has **not** been tested with GPFS

- While ERM has not been tested with GPFS, the mechanisms of ERM are embodied in the storage controller and completely independent of GPFS. For the most part, GPFS operations are unaware of the ERM functionality. There is no reason why GPFS should not work with ERM.
- A proof of concept (POC) test is recommended.

■ Operational Procedures

- Each logical drive in a mirrored pair presents itself to the local host(s) as a SCSI device (e.g., /dev/sdb).
- Write requests can be received only by the primary logical drive in a mirrored pair.
- Read requests can be received by both the primary and logical drive; reading secondary logical drives is primarily intended for administrative purposes.
- Two file systems are created for the mirrored storage controller pair, a primary and secondary file system.
- During normal operation, applications access the primary file system.
- If operation to the primary file system is lost, the secondary site must go through a role reversal, and the secondary file system unmounted/mounted.
- Full synchronization is needed after an outage.



Snapshots



TBD



GPFS SNMP support



TBD



14. Best Practices

Consider an eclectic assortment of GPFS "best practices" (some collective wisdom).

While these are simple, common sense things, they are easily overlooked, especially when you are working with legacy codes developed under different conditions and assumptions.



Miscellaneous Best Practices

Performance Hierarchy

1. large record sequential order
2. large record strided order or small record sequential order
3. large records in random order or small records in strided order
4. issue hints when reading in small records in random order
5. small record random order without hints

NOTE: large records are \geq GPFS block size
small records are $<$ GPFS block size (*e.g.*, 2K to 16K)



Miscellaneous Best Practices

Performance Hierarchy

Example illustrating how code can be rewritten to eliminate small record accesses.

- **Suppose you are sorting directly a set of *small*, randomly or semi-randomly distributed records.**
- **Because records are small, GPFS will perform poorly.**
- **Rewrite code sort as follows:**
 - divide file into N subsets and assign each subset to a node
 - choose the subset size so that it can fit entirely within RAM
 - sort each subset
 - depending on file size, a node may need to sort several subsets
 - merge all of the subsets together
- **This improves performance by**
 - performing all small record references in memory
 - sequentially accessing records on disk in the merge step
- **This is a variation of the mergesort algorithm**



Miscellaneous Best Practices

Read:Write Ratio for HPC Processing

General rule of thumb in CS textbooks

- read: 90%
- write: 10%

But this generalization is more typical of commercial applications than technical HPC applications.

For example, the ratio for many scientific applications is

- read: 60% to 70%
- write: 40% to 30%

Therefore, plan accordingly.



Miscellaneous Best Practices

Response Time vs Bandwidth

■ Storage systems optimized for response time generally

- have larger variance
- complete **less** work per unit time
- are best suited to support online users

■ Storage systems optimized for bandwidth generally

- have smaller variance
- complete **more** work per unit time
- are best suited for batch systems



Miscellaneous Best Practices

Avoid "Gold Plating"

■ Rule of thumb

- Configure a file system to handle peak performance up to 3 or 4 standard deviations above the mean to avoid "gold plating". (John Watts, IBM)
- Programmers worried about performance will often over architect a system

■ When sizing the disk I/O subsystem, programmers...

- should *not* ask
 - What is the peak load?
- should ask
 - What are the highest mean loads?
 - What are their standard deviations?
 - What loads are you prepared to pay to meet? Once a month? Once a year?



Miscellaneous Best Practices

Mixing Home and Scratch Directories Under GPFS

■ Home Directory Usage

- Many small files
- Support online, interactive transactions
- Large variance in access patterns, data rates and times of usage
 - *e.g.*, used a lot during business hours and quiet during the night
- `stat()` call are frequent
 - *e.g.*, `ls -l`
- Rate of change is small
 - *e.g.*, many files remain untouched for long periods of time

■ Scratch Directory Usage

- Principle working directory for applications
- Support batch processing, often under job scheduler
- 24x7 usage with consistent access patterns and data rates
 - *n.b.*, does not imply low variance... its just a different kind of variance
- `stat()` calls are infrequent
- Rate of change can be large
 - HSM systems move untouched files to lower storage tiers (*e.g.*, tape)

■ Best Practice

- Avoid mixing home and scratch directories under GPFS



Miscellaneous Best Practices

When to Use NFS



- Where NFS works well
- Where NFS is a challenge
- NFS *vs.* GPFS





Miscellaneous Best Practices

Myth: GPFS is Hard to Manage



One of GPFS's salient features is that it has a million knobs...

One of GPFS's problems is that it has a million knobs...

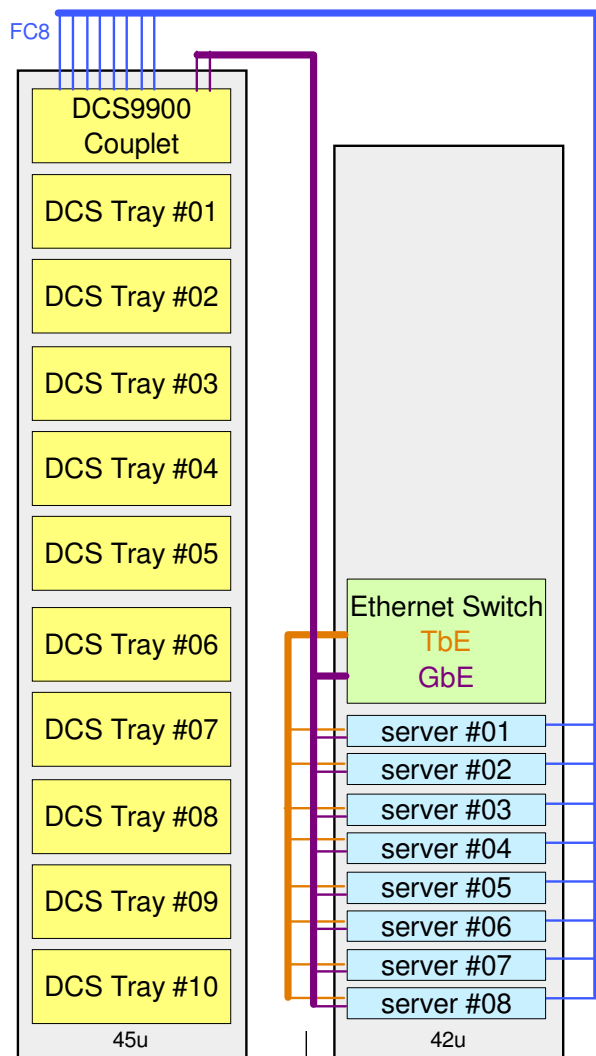


But do not worry!



Miscellaneous Best Practices

Myth: GPFS is Hard to Manage



GPFS is simplest complex product you will ever use!

Only 6 Steps to install, build and configure a GPFS file system

1. Install from media (e.g., rpm or smitty)
2. For Linux only build portability layer
 - see /usr/lpp/mmfs/src/README
3. Create the GPFS cluster
4. Startup the GPFS daemons
5. Create the logical disks (i.e., NSDs)
6. Create and mount the file system

An experienced sysadm can do this in as little as 5 to 10 minutes!

GPFS provides convenient sysadm tools

1. Adding and deleting disk
2. Adding and deleting nodes
3. Changing disk, cluster, configuration and file system attributes
4. Monitoring performance (including latency)

Many sysadm tasks like adding and deleting nodes or disks can be completed without shutting down GPFS, rebooting or interfering with production.



15. GPFS Road Map

16. Conclusion and Observation

- ▶ GPFS is a best of class product with good features, but it is not a "silver bullet"
- ▶ Without careful design, I/O can seriously degrade parallel efficiency (e.g., Amdahl's law)
- ▶ Good I/O performance requires hard work, careful design and the intelligent use of GPFS
- ▶ I/O is not the entire picture; improving I/O performance will uncover other bottle necks

