Parallel storage and Parallel FS

- Stefano Cozzini
- CNR-IOM and eXact lab srl

Agenda

- I/O infrastructure for HPC
- Distributed FS
    - NFS
- Parallel File System
- Lustre
- How to use Lustre FS

Building block for HPC I/O systems

- A HPC I/O system should:
  - Present storage as a single, logical storage unit
    - We do not want to look for different storage on different nodes
  - Tolerate failures (in conjunction with other HW/SW)
    - We do not want to stop production when a disk/server/inc card) breaks
  - Provide a standard interface (i.e. Posix compliant)
    - We do not want to change your code when you use an HPC
  - Stripe files across disks and nodes for performance
    - We do want to get parallel performance on parallel system

HPC I/O infrastructure

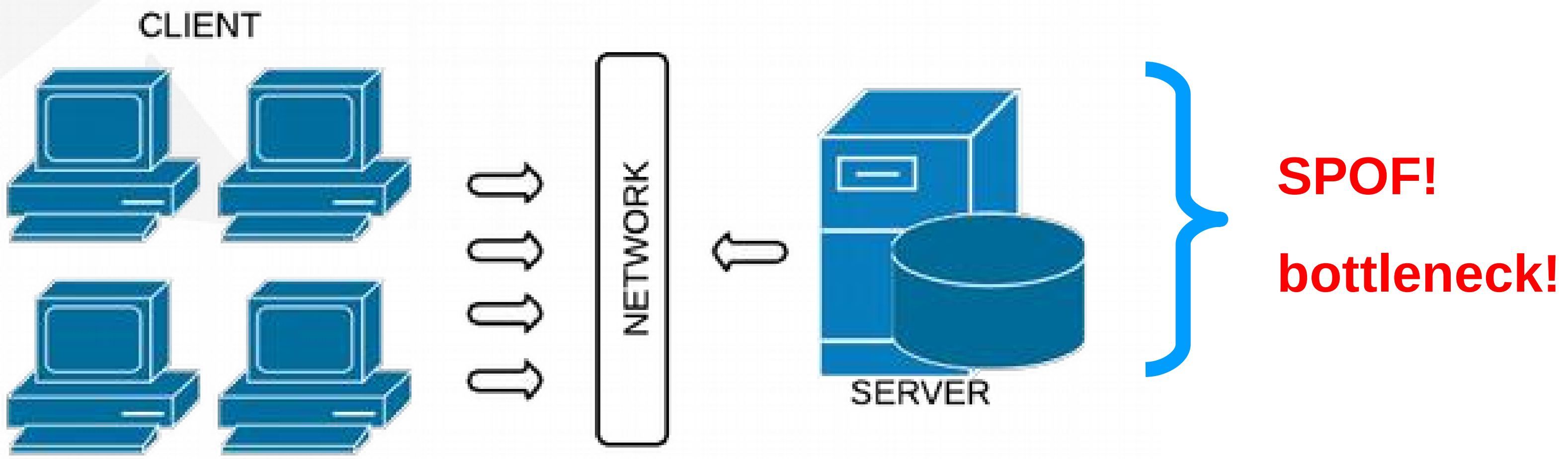
- HW:
  - Disks/ disk enclosure/ disk controllers
  - Server,
  - Networks etc..etc..
- Software:
  - distribute/parallel Filesystem,
  - libraries
  - some parts of O.S.

Scaling the Filesystem…

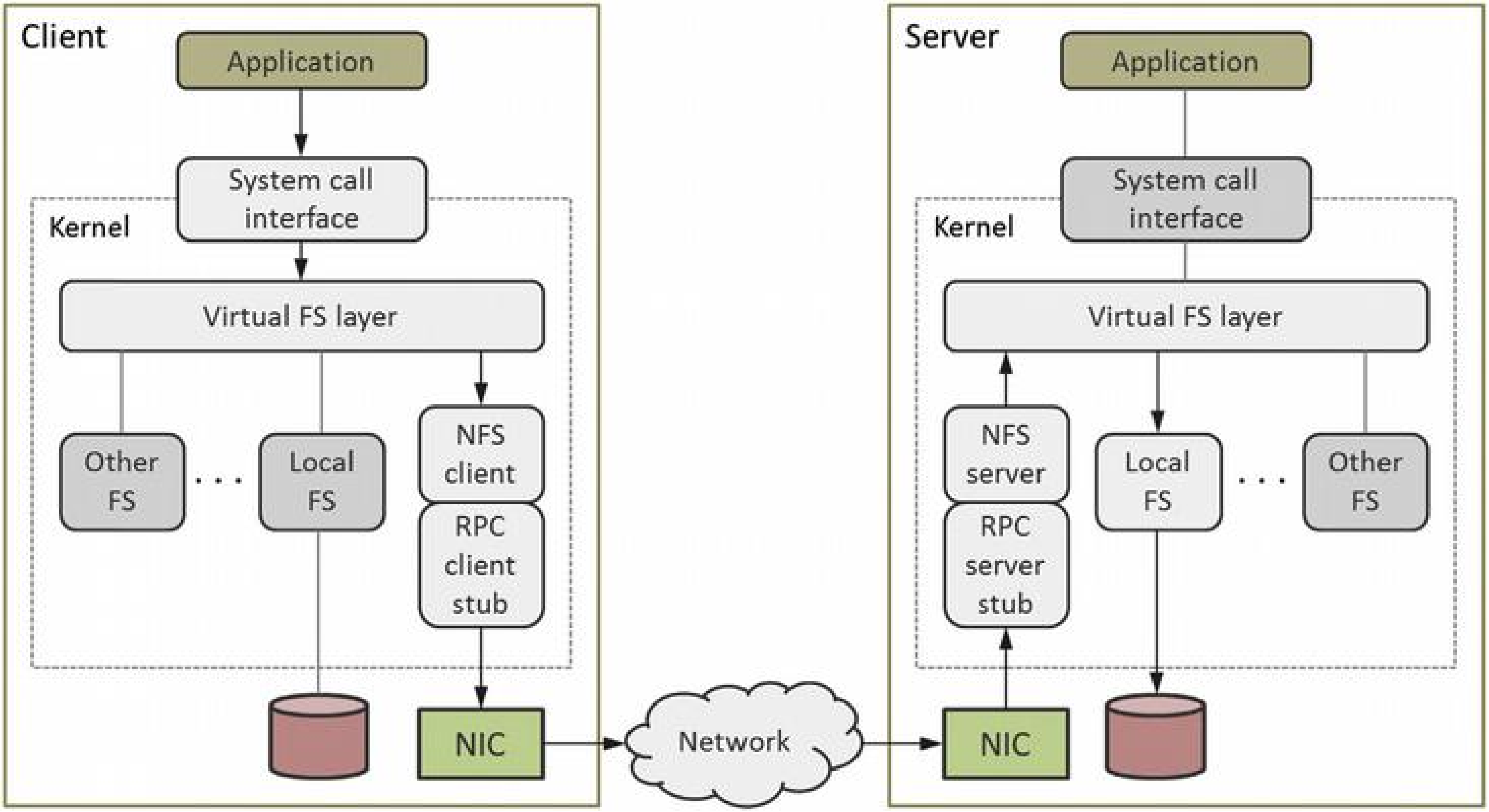
- Original POSIX environment was unshared, direct-attached storage
- RAID and Volume Managers aggregate devices safely
  - Scale performance within the same machine
- Distributed FS introduces a Network (Ethernet) between clients and server
  - Able to coordinate access from multiple clients: scales over many client
- Parallel FS coordinates many clients and many servers
  - A special kind of networked file system that provides high-performance I/O when multiple clients share the file system
  - Able to scale in both capacity and performance

Distributed file system

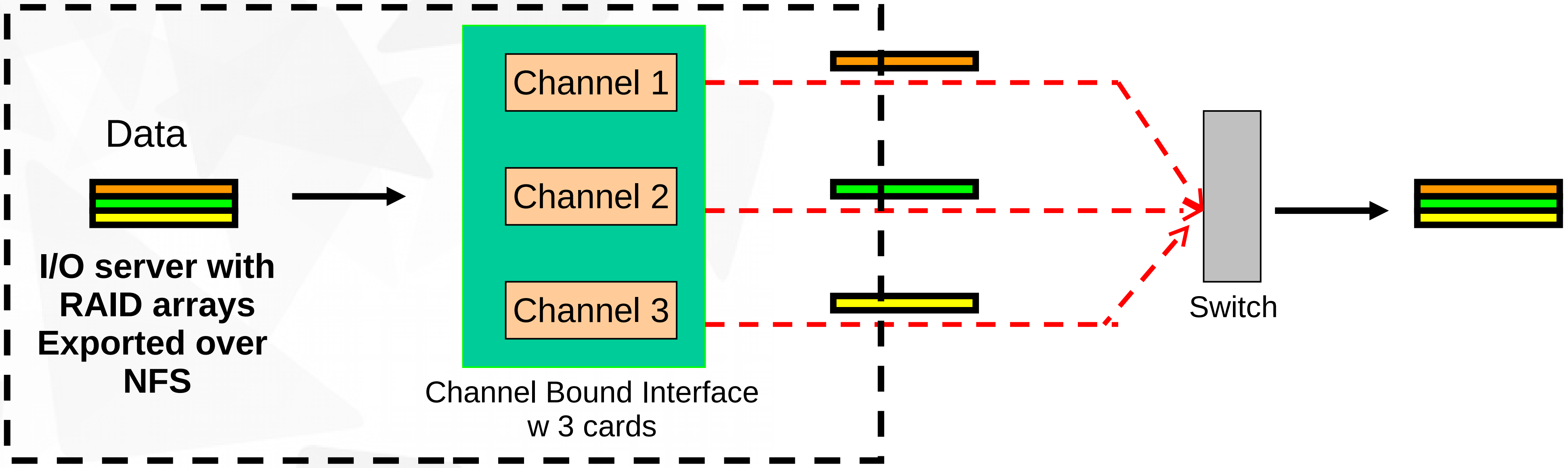
- Distributed file systems are file systems that are capable of handling I/O requests issued by multiple clients over the network.
- FS is “mounted” by several clients (compute nodes/login nodes..)
- Example: Network File System
  - Parallel access is possible but:
  - Network bandwith limited..
  - Locking issues



NFS architecture

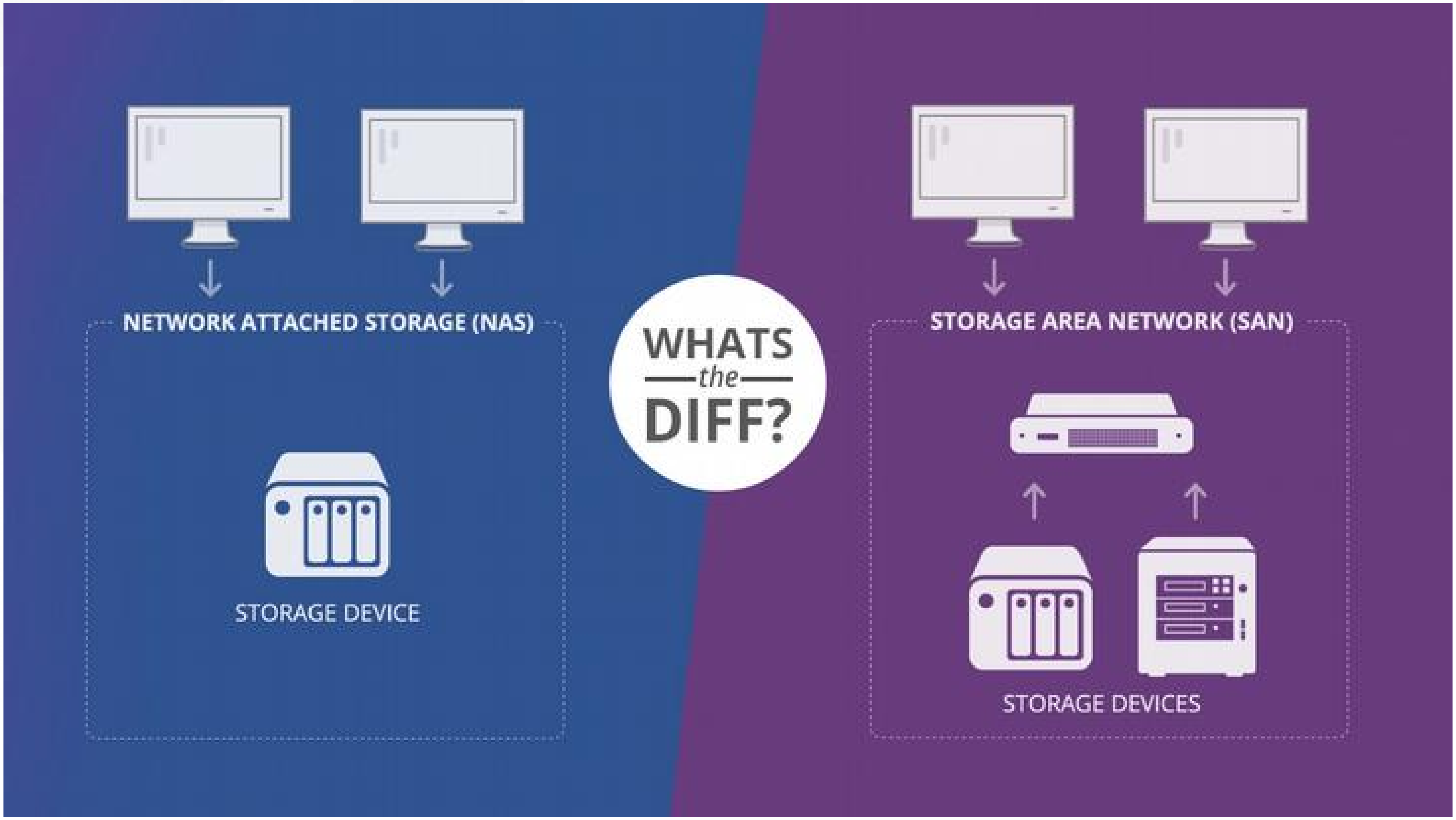


NFS I/O server (poor man PFS)


Data
I/O server with RAID arrays Exported over NFS
Channel 1
Channel 2
Channel 3
Channel Bound Interface w 3 cards
Switch

- I/O performance can be obtained/ increased by combing RAID (for I/O and/or security and channel bonding (for increased bandwidth).
- Good: cheap and easy to maintain and configure.
- Bad:
  - performance is marginal (N disks give N I/O speedup at best), .
  - Need a high end network switch.
  - Performance degrades fast as the function of the number of users.
  - Needs lots of memory on I/O server to allow for good buffering.
  - Needs good UPS/Shutdown solution, to prevent disk corruption.

SAN vs NAS:



Picture from: https://www.backblaze.com/blog/whats-the-diff-nas-vs-san/

SAN vs NAS

- SAN
  - Block level data access
  - Fiber channel is the primary media used with SAN.
  - SCSI is the main I/O protocol
  - SAN storage appears to the computer as its own storage

    NAS

- NAS
  - File Level Data access
  - Ethernet is the primary media used with NAS
  - NFS is used as the main I/O protocol in NAS
  - appears as a shared partition to the computer

Issues in building HPC I/O systems

- “Management problem”: many disks/ HW around our cluster but not easy to make them available to user in a clean/safe/cheap way.
- “Performance problems” : large dataset requires high performance I/O solutions

Scalability Limitation of I/O

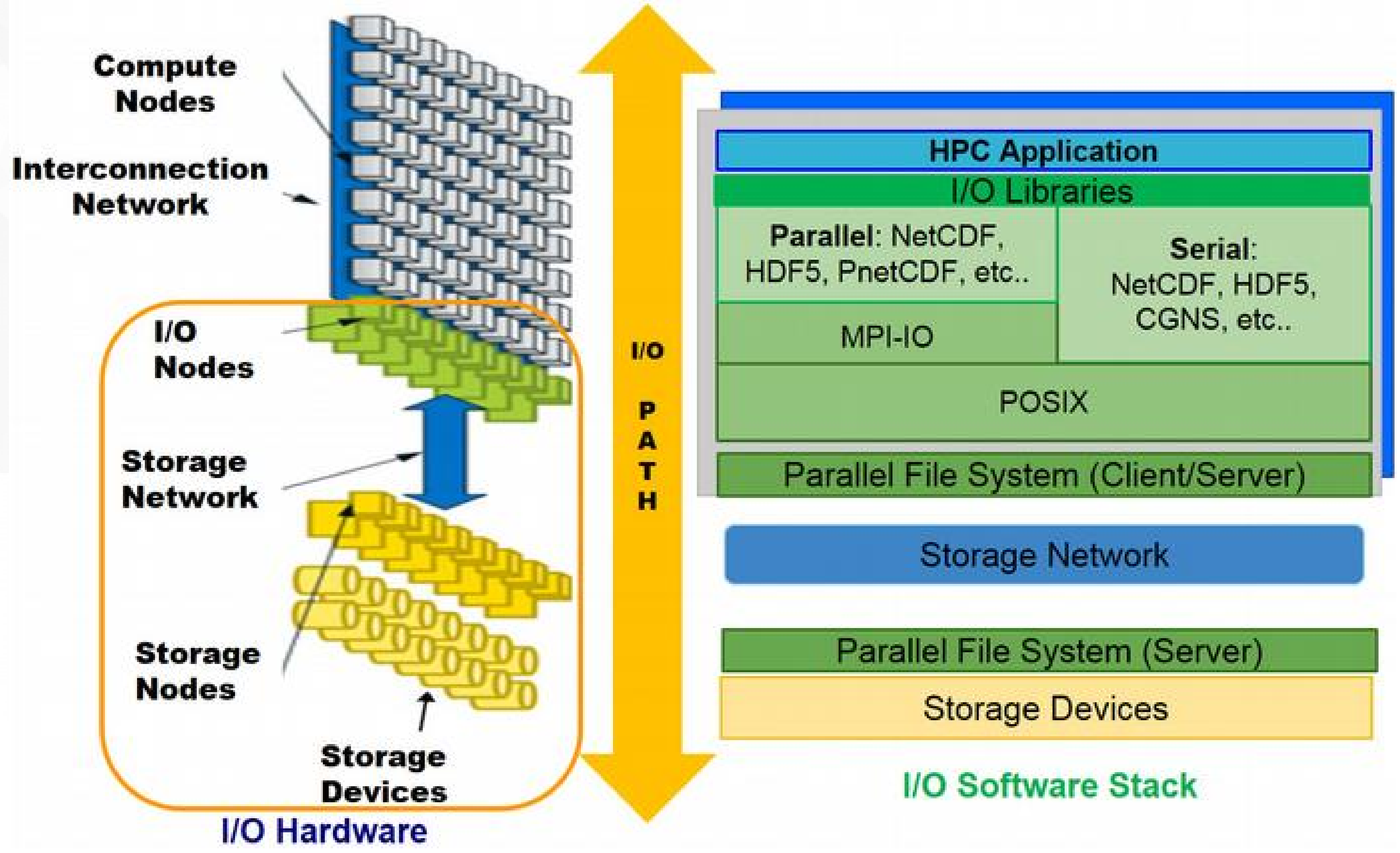
- I/O subsystems are typically very slow compared to other parts of a supercomputer
  - You can easily saturate the bandwidth
- Once the bandwidth is saturated scaling in I/O stops
  - Adding more compute nodes increases aggregate memory bandwidth and flops/s, but not I/O

Parallel File System

Parallel FS solution

- A parallel solution usually is made of
  - several Storage Servers that hold the actual filesystem data
  - one or more Metadata Servers that help clients to identify/manage data stored in the file system
  - a redundancy layer that replicates in some way information in the storage cluster, so that the file system can survive the loss of some component server
- and optionally:
  - monitoring software that ensures continuous availability of all needed components

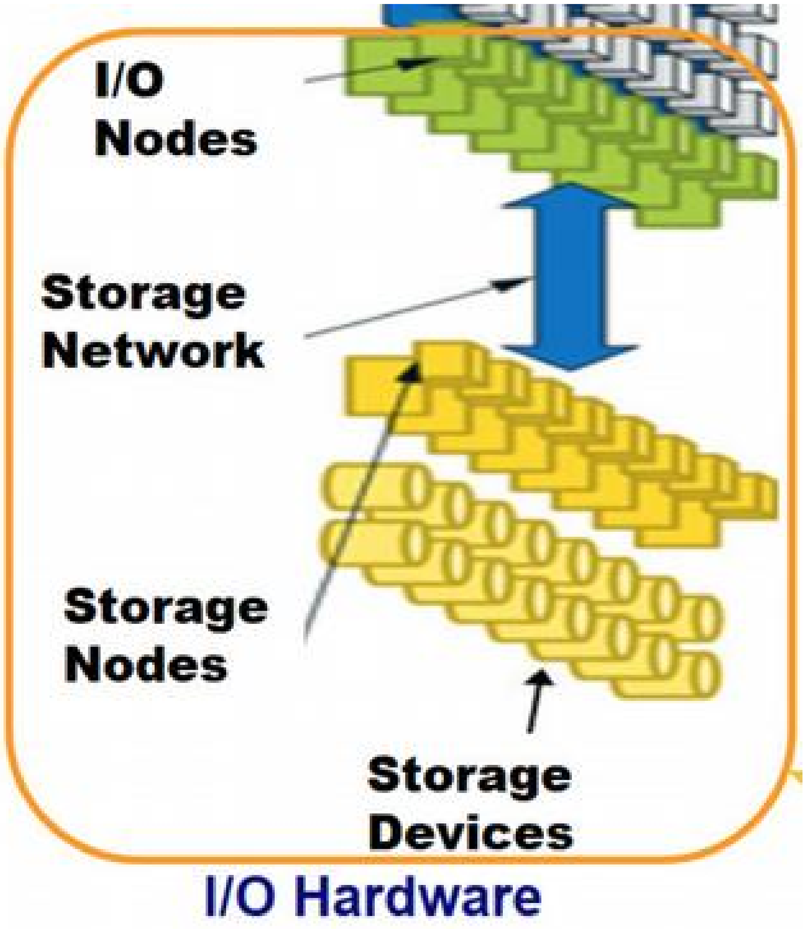
A graphical view:



Picture from: http://www.prace-ri.eu/best-practice-guide-parallel-i-o/#id-1.3.5

Parallel File System: I/O hardware

- I/O nodes = Storage Nodes (Ulysses)
- Metadata server hosted on I/O server (dedicated and/ or shared)
- Storage nodes hosts some data:
- Metadata server coordinates access by the clients to the data
-



Parallel filesystem approach

- In a parallel solution..
  - you add storage servers, not only disk space
  - each added storage server brings in more memory, more processor power, more bandwidth
  - software complexity however is much higher
- there is no «easy» solution, like NFS...

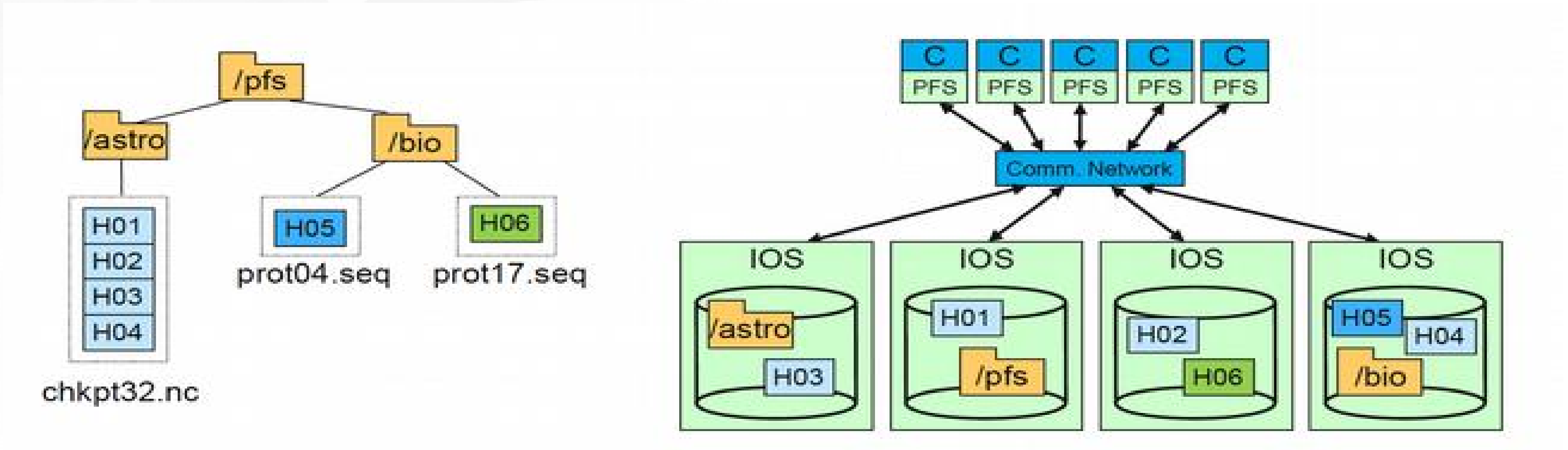
A disclaimer...

- Parallel File Systems are usually optimized for high performance rather than general purpose use,
- Optimization criteria:
  - –Large block sizes (≥ 64kB)
  - –Relatively slow metadata operations (eg. fstat()) compared to reads and writes..)
  - –Special APIs for direct access and additional optimizations

Hardware to build a PFS

- Disks, controllers, and interconnects
- Hardware defines the peak performance of the I/O system:
  - raw bandwidth
  - Minimum latency
- At the hardware level, data is accessed at the granularity of blocks, either physical disk blocks or logical blocks spread across multiple physical devices such as in a RAID array
- Parallel File Systems takes care of
  - managing data on the storage hardware,
  - presenting this data as a directory hierarchy,
  - coordinating access to files and directories in a consistent manner

Parallel FS



An example parallel file system, with large astrophysics checkpoints distributed across multiple I/O servers (IOS) while small bioinformatics files are each stored on a single IOS

What is available on the market ?

- BeeGFS
  - Developed at Fraunhofer Institute, freely available not open
  - http://www.fhgfs.com/cms/
- Lustre
  - open and Free owned by Intel DDN
  - Intel no longer sells tools to manage and support ($$$)
  - http://lustre.opensfs.org/
- GPFS (now known as Spectrum Scale)
  - IBM proprietary $$$
  - Very nice solution and expensive ones !
- And many others (WekaIO/MooseFS/Panasas... etc)

Comparing parallel FS

- Sys Adm:
  - Block Management
  - How Metadata is stored
  - What is cached and where
  - Fault tolerance mechanisms
  - Management/Administration

- User:
  - Performance
  - Reliability
  - Manageability
  - Cost

Recap on Metadata

- Metadata names files and describes where they are located in the distributed system
  - Inodes hold attributes and point to data blocks
  - Directories map names to inodes
- Metadata updates can create performance problems
- Different approaches to metadata are illustrated via the File Create operation

Create a file on local FS

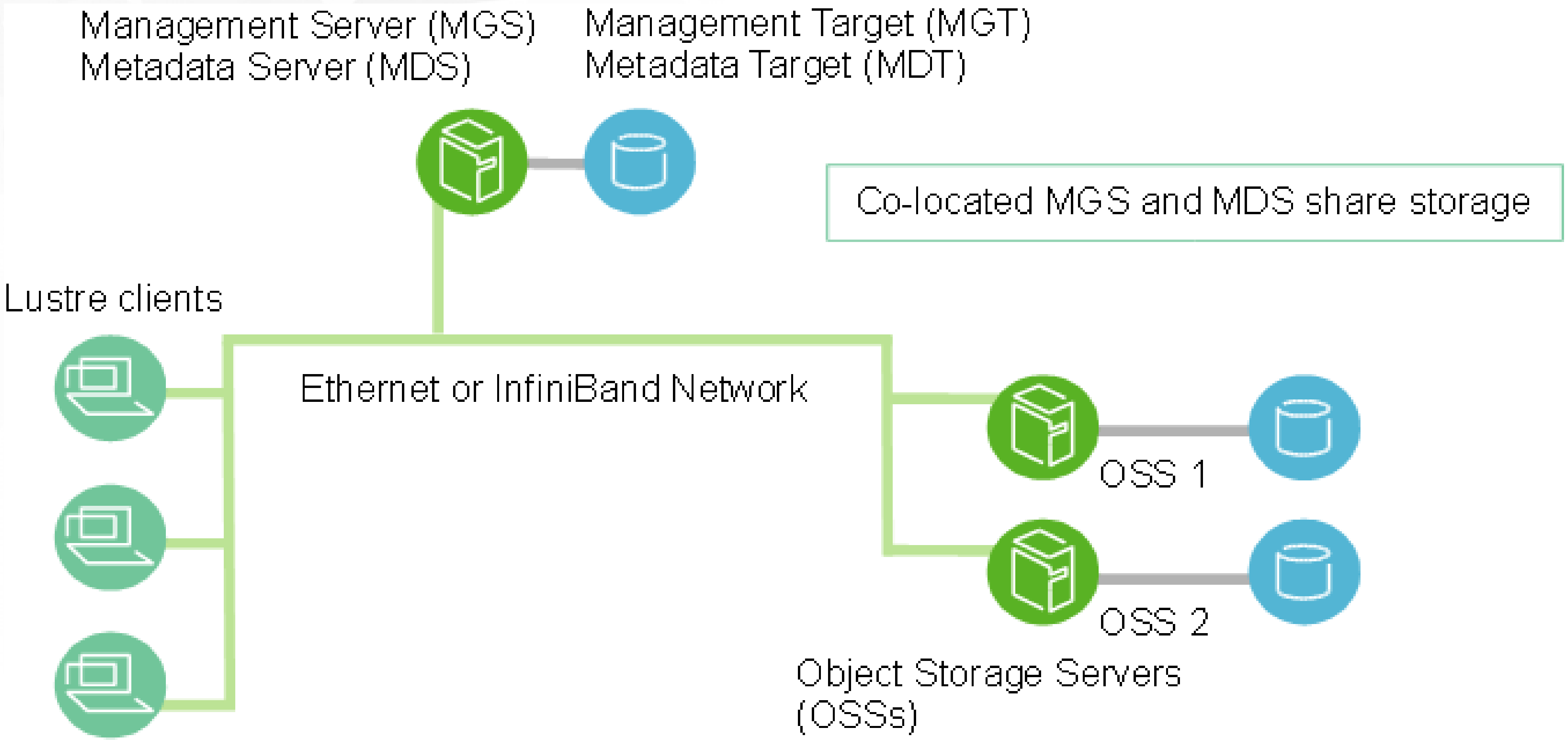
- 4 logical I/Os
  - Journal update
  - Inode allocation
  - Directory insert
  - Inode update
- Performance determined by journal updates
  - Details vary among systems

Lustre File System

Lustre Principles

- Lustre is centralized
  - No copy of data or metadata.
  - Metadata stored on a shared storage called Metadata Target (MDT)
  - Attached to two Metadata Servers (MDS)
- MDS provides the illusion of shared name space
- Lustre is an overlay file system
- Data are stored on back-end file-systems: Object Storage Target (OST)
  - Attached to several Object Storage Servers (OSS) for active/active failover.
  - OSS are the servers that handle I/O requests.
- Remote clients that can mount the Lustre filesystem, e.g. your

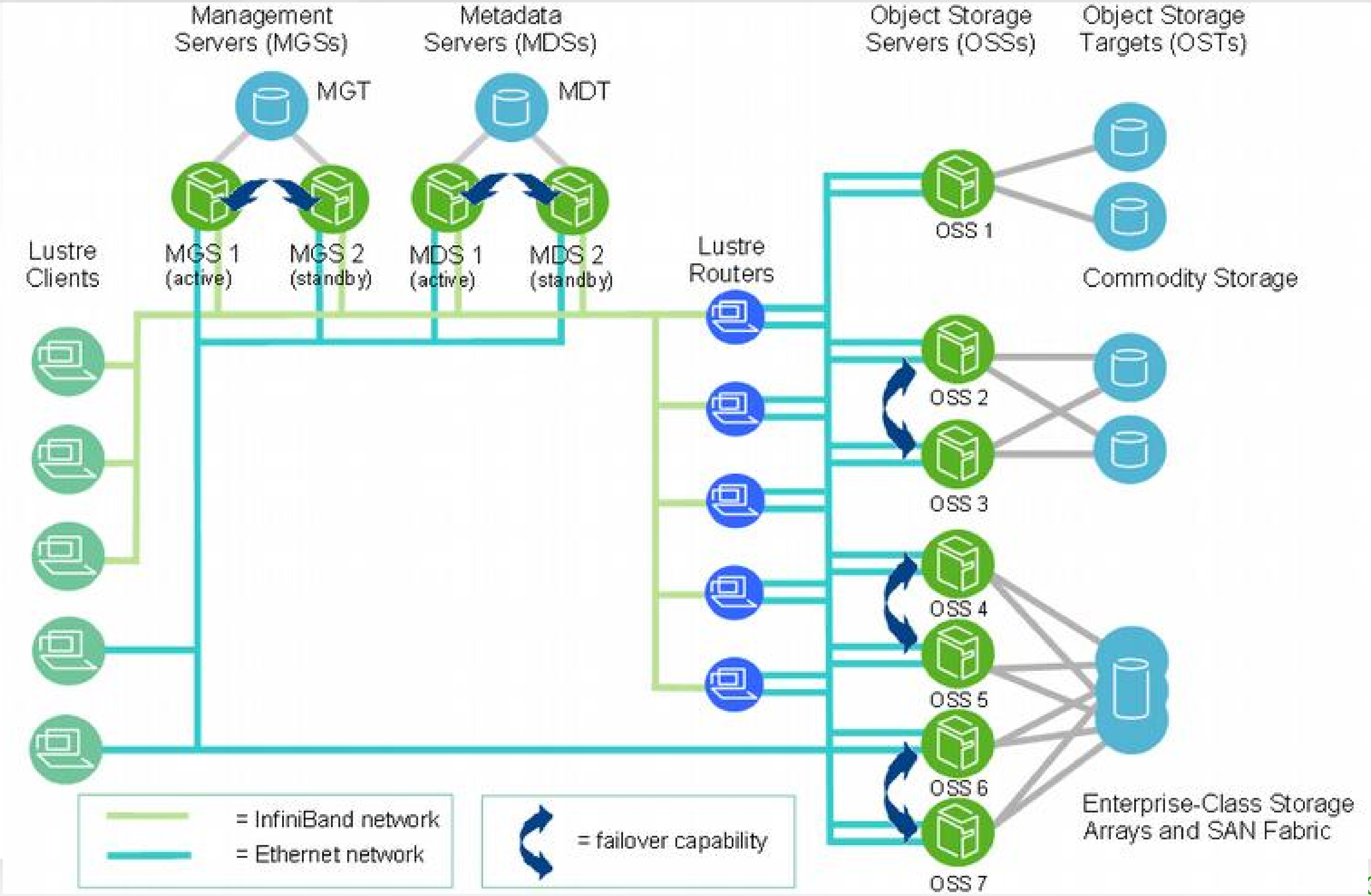
Lustre layout (simple cluster)



Building blocks of Lustre FS

- Metadata Servers (MDS) :
  - The MDS makes metadata stored in one or more Metadata Targets (MDTs) available to Lustre clients. Each MDS manages the names and directories in the Lustre file system(s) and provides network request handling for one or more local MDTs. Operations such as opening and closing a file can require dedicated access to the MDS and it can become a serial bottleneck in some circumstances.
- Metadata Targets (MDT) : .
  - The MDT stores metadata (such as filenames, directories, permissions and file layout) on storage attached to an MDS. An MDT on a shared storage target can be available to multiple MDSs, although only one can access it at a time. If an active MDS fails, a standby MDS can serve the MDT and make it available to clients. This is referred to as MDS failover.
- Object Storage Servers (OSS) :
  - The OSS provides file I/O service and network request handling for one or more local Object Storage Targets (OSTs). Typically, an OSS serves between two and eight OSTs, up to 16 TB each. A typical Lustre configuration consists of an MDT on a dedicated node, two or more OSTs on each OSS node
- Object Storage Target (OST) :
  - User file data is stored in one or more objects, each object on a separate OST in a Lustre file system. Each OST can write data at around 500 MB/s. You can think of an OST as being equivalent to a disk, although in practice it may comprise multiple disks, e.g. in a RAID array. An individual file can be stored across multiple OSTs; this is called striping. The default is dependent on the particular system (1 is a common choice), although this can be changed by the user to optimize performance for a given workload.

Lustre layout (complex infrastructure)



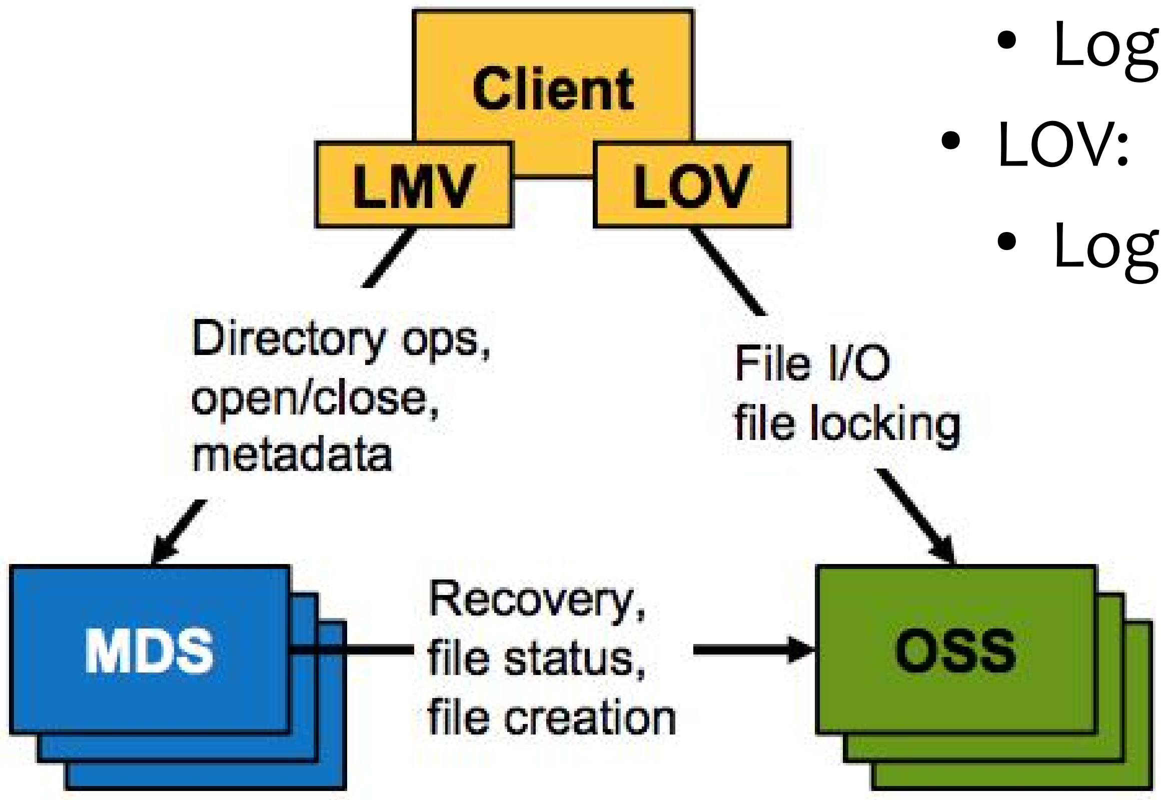
Lustre component characteristics

- Clients
  - 1 - 100,000 per system
  - Data bandwidth up to few GB/s; 1000’s of metadata ops/s
  - No particular hardware or attached storage characteristics required
- Object Storage Servers (OSS)
  - 1 - 1000 per system
  - 500 MB/s..2.5 GB/s I/O bandwidth
  - Require good network bandwidth and adequate local storage capacity coupled with sufficient number of OSSs in the system
- Metadata Servers (MDS)
  - 2 (for redundancy) per system
  - 3,000 - 15,000 metadata ops/s
  - Utilize 1 – 2% of total file system capacity
  - Require memory-rich nodes with lots of CPU processing power

Create a File on Lustre

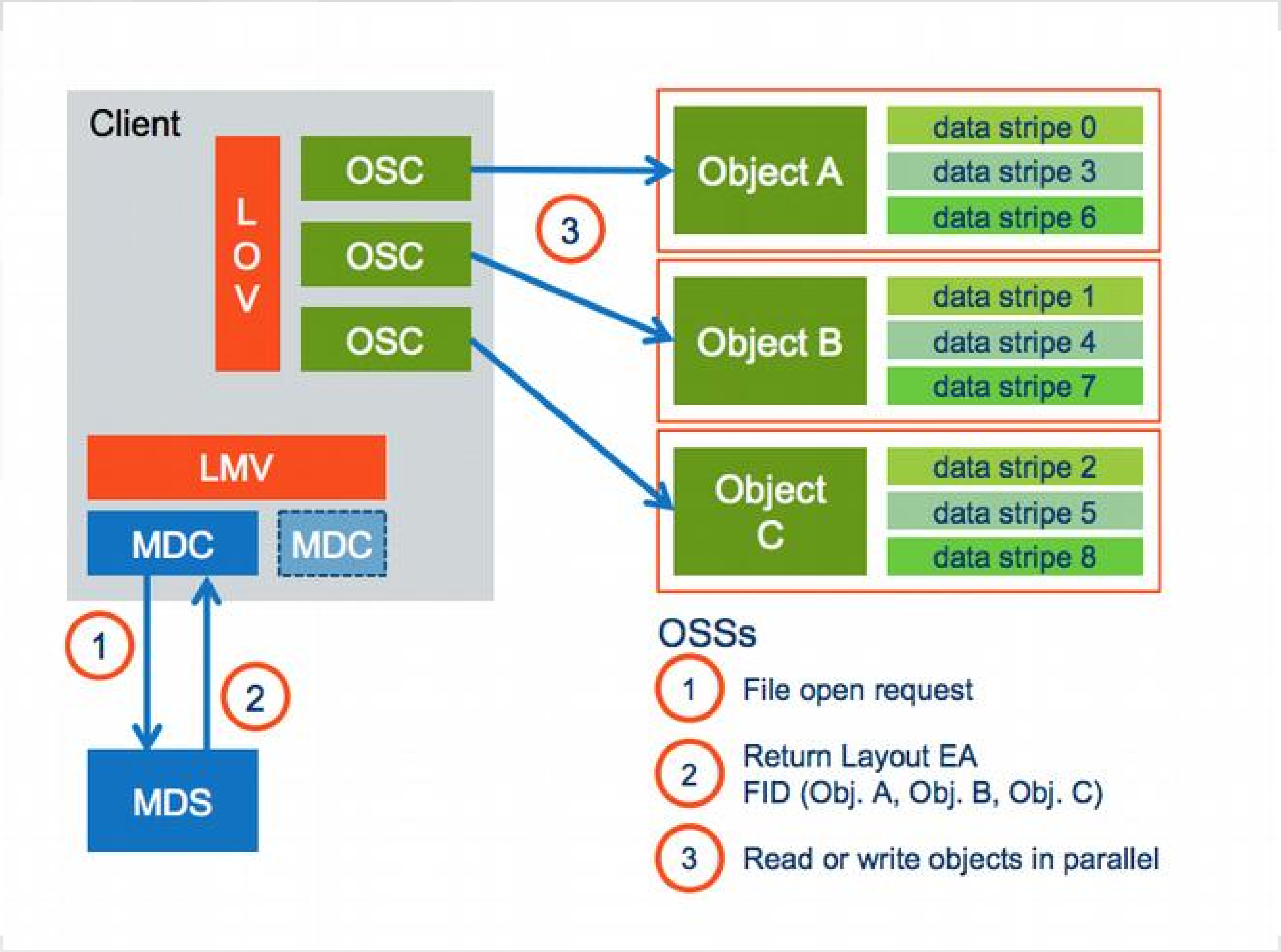
- Client to Server RPC (remote procedure call)
- Server creates local file to store metadata
  - Journal update, local disk I/O
- Server creates container object(s)
  - Object create transaction with OSS
  - OSS creates local file for object
- Performance dependent on
  - Local file systems on metadata server and OSS/OST

Lustre operations


Client
LMV
LOV
Directory ops, open/close, metadata
File I/O file locking
MDS
Recovery, file status, file creation
OSS

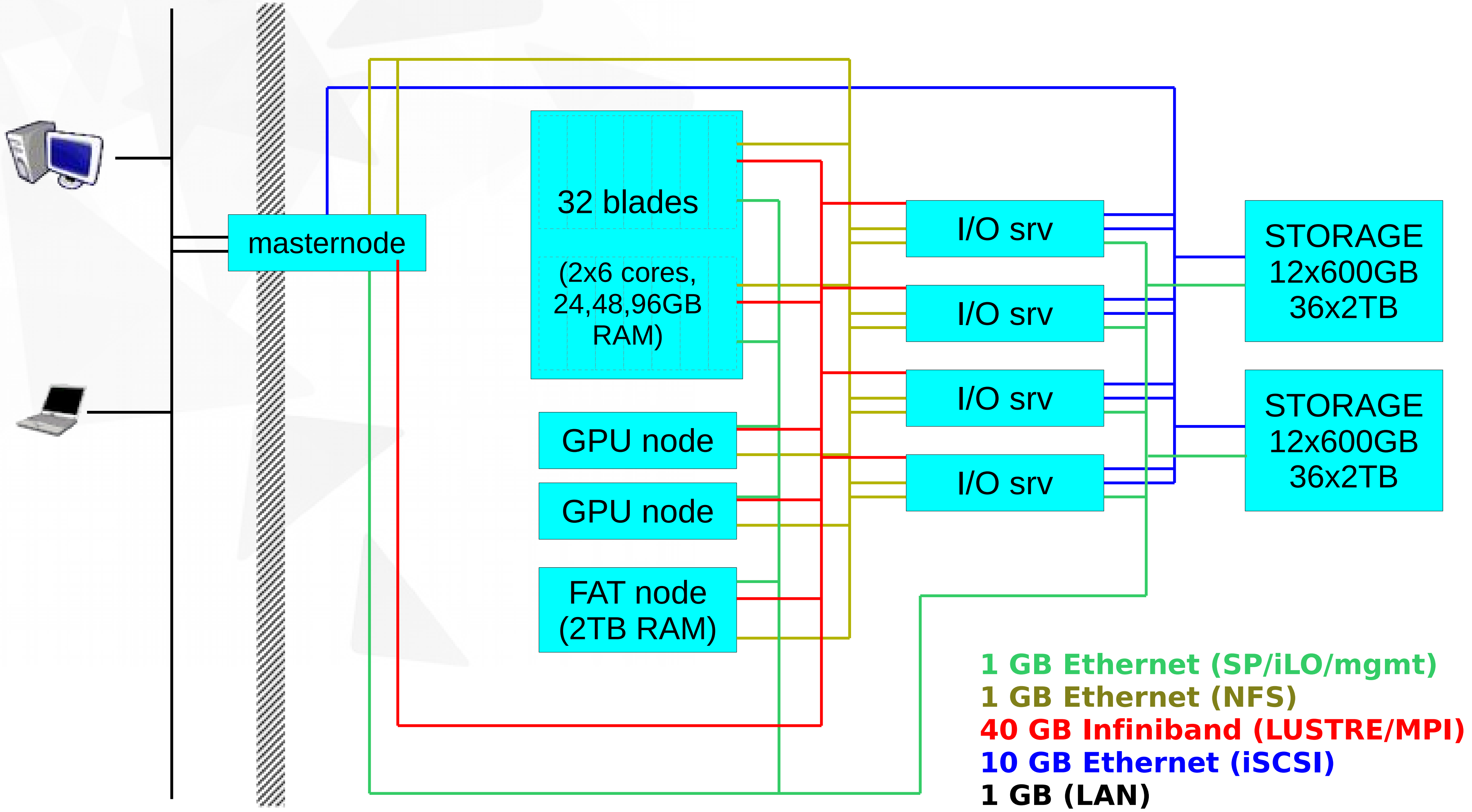
- LMV:
  - Logical Metadata Volume
- LOV:
  - Logical Object Volume

Lustre operations

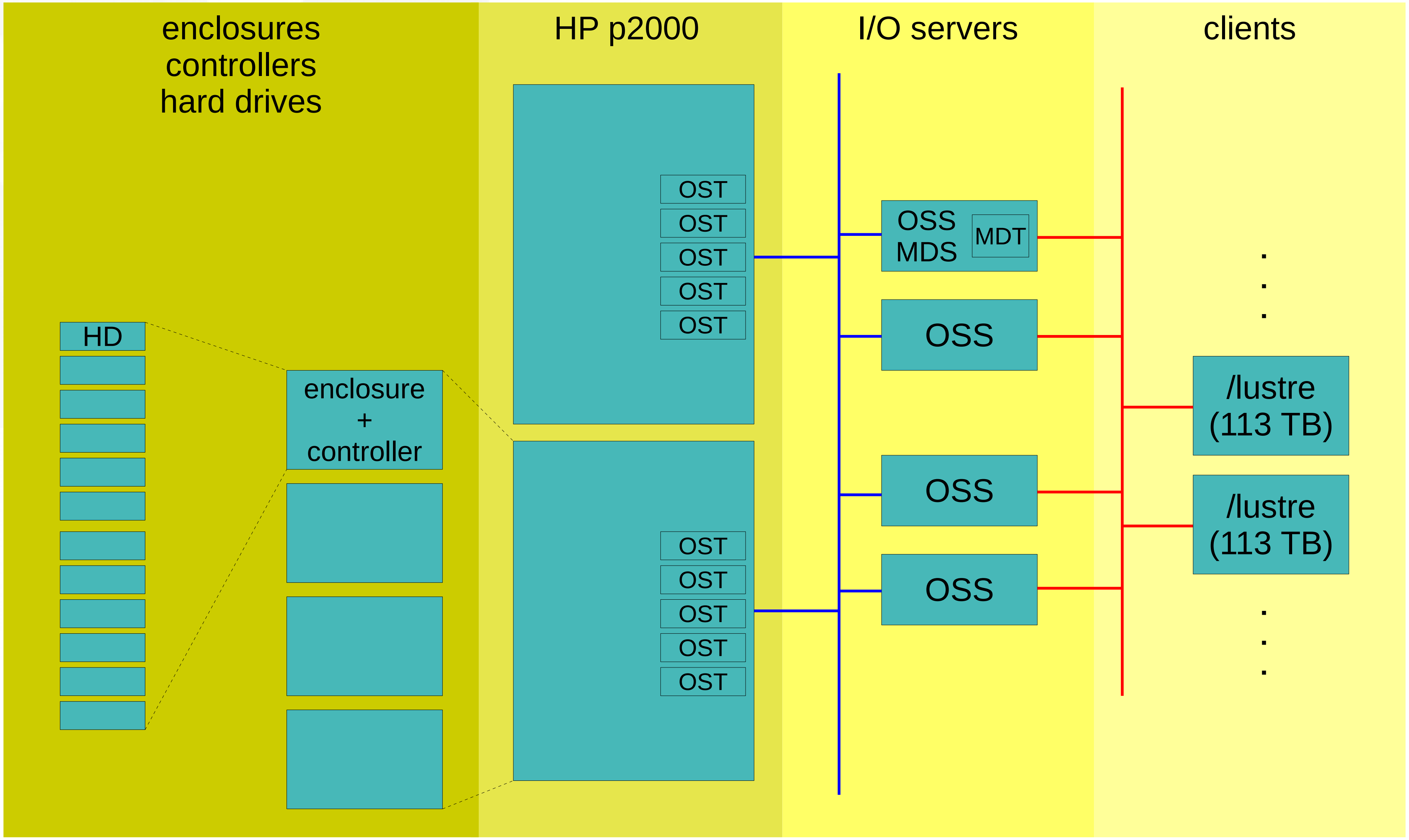


Lustre example

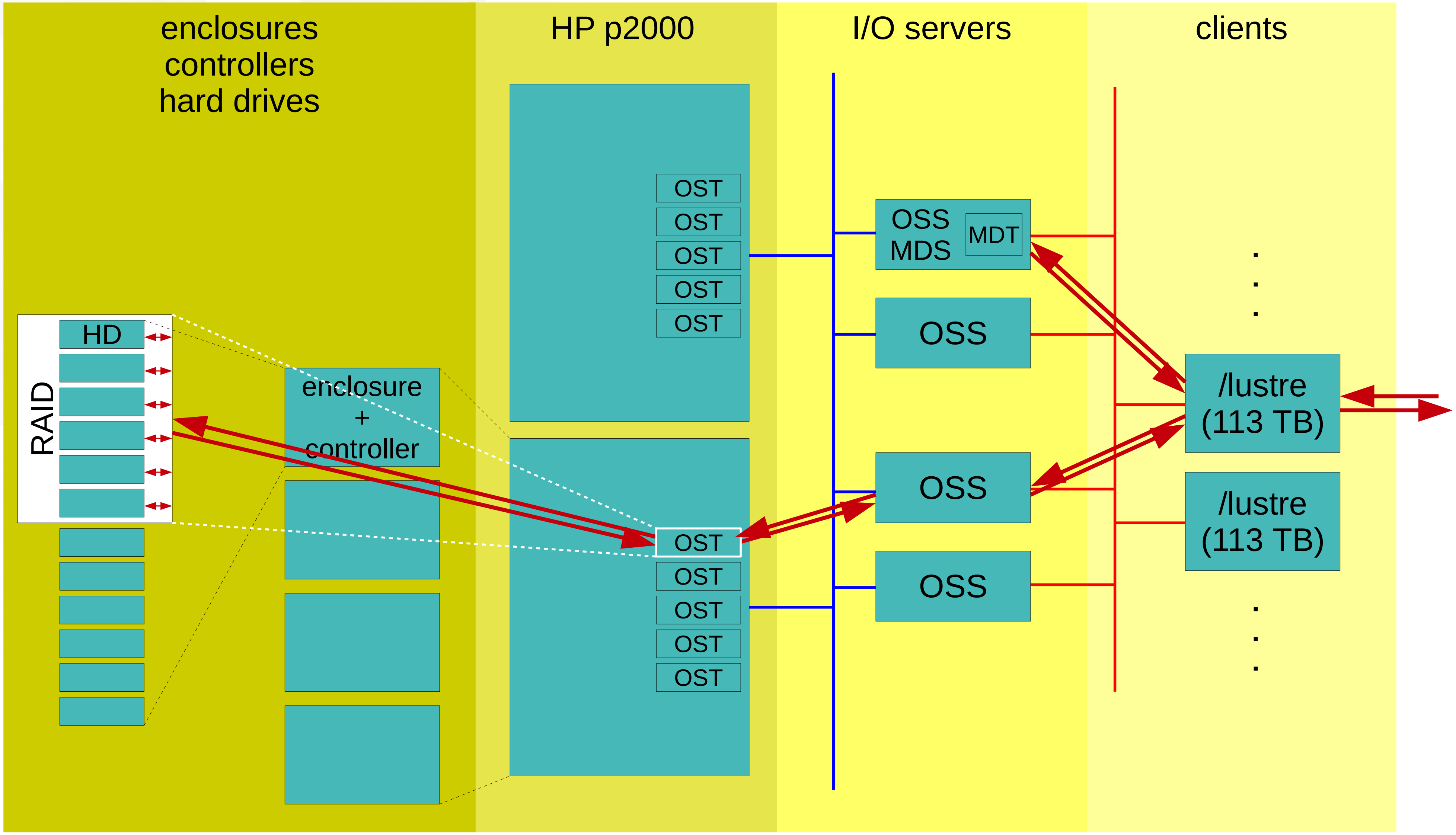
HPC infrastructure @ CRIBI



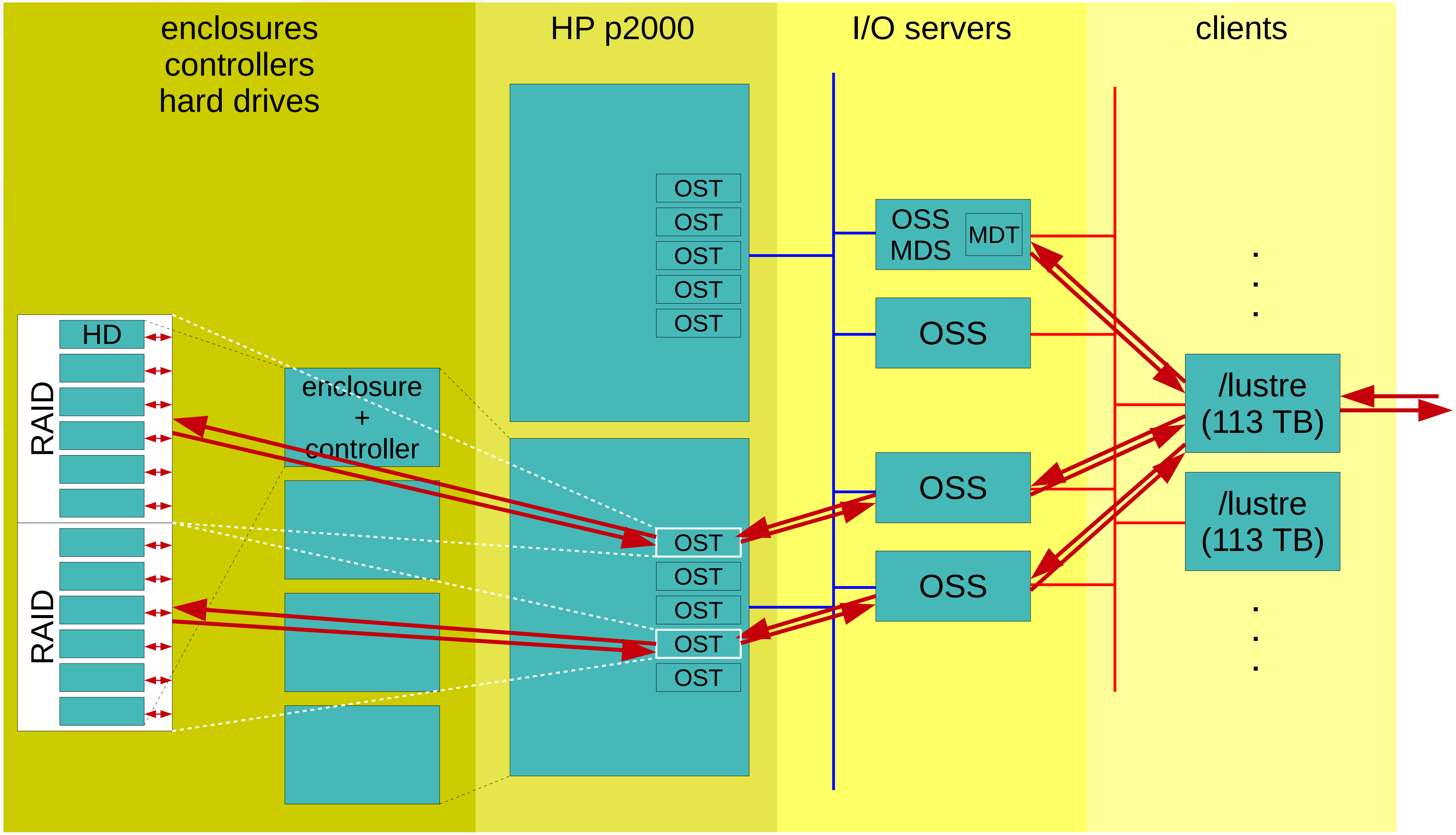
LUSTRE and storage solution



accessing LUSTRE filesystem



why “parallel” filesystem?



Expected performance

- Elements of the infrastructure:
  - Network Speed: Infiniband QDR :3.2GB/sec for server
    - Network aggregate bandwith: 3.2 x 4 ~ 12GB/se
  - 4 IO-SRV
    - two OST each
      - Each OST: RAID 6 6 disks
        - OST Aggregate bandwith: (6-2)*120 = 400 Mb/seconds
        - Disk speed: 120 Mb/seconds
    - Node Aggregate bandwith 400x 2 = 800 Mb/sec
  - Peak performance : 4 GB/sec read/write

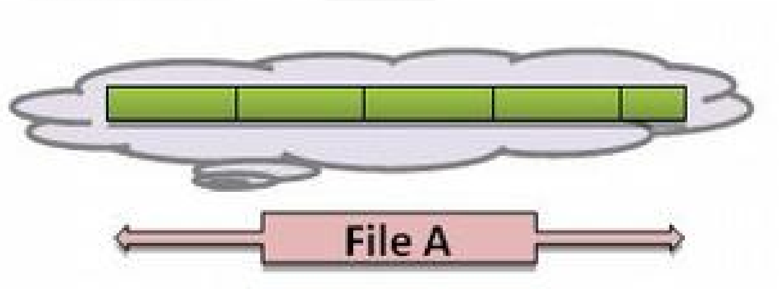
How to use Lustre

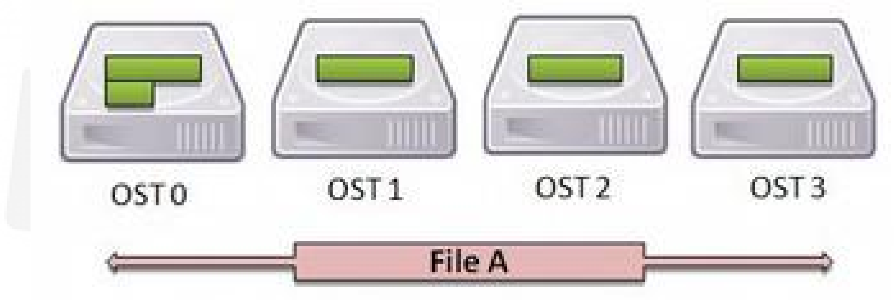
Lustre file striping

- Each file in Lustre has its own unique file layout, comprised of 1 or more objects in a stripe equivalent to RAID
- File layout is allocated by the MDS
- Layout is selected by the client, either
  - by policy (inherited from parent directory)
  - by the user or application
- Layout of a file is fixed once created

Lustre file striping

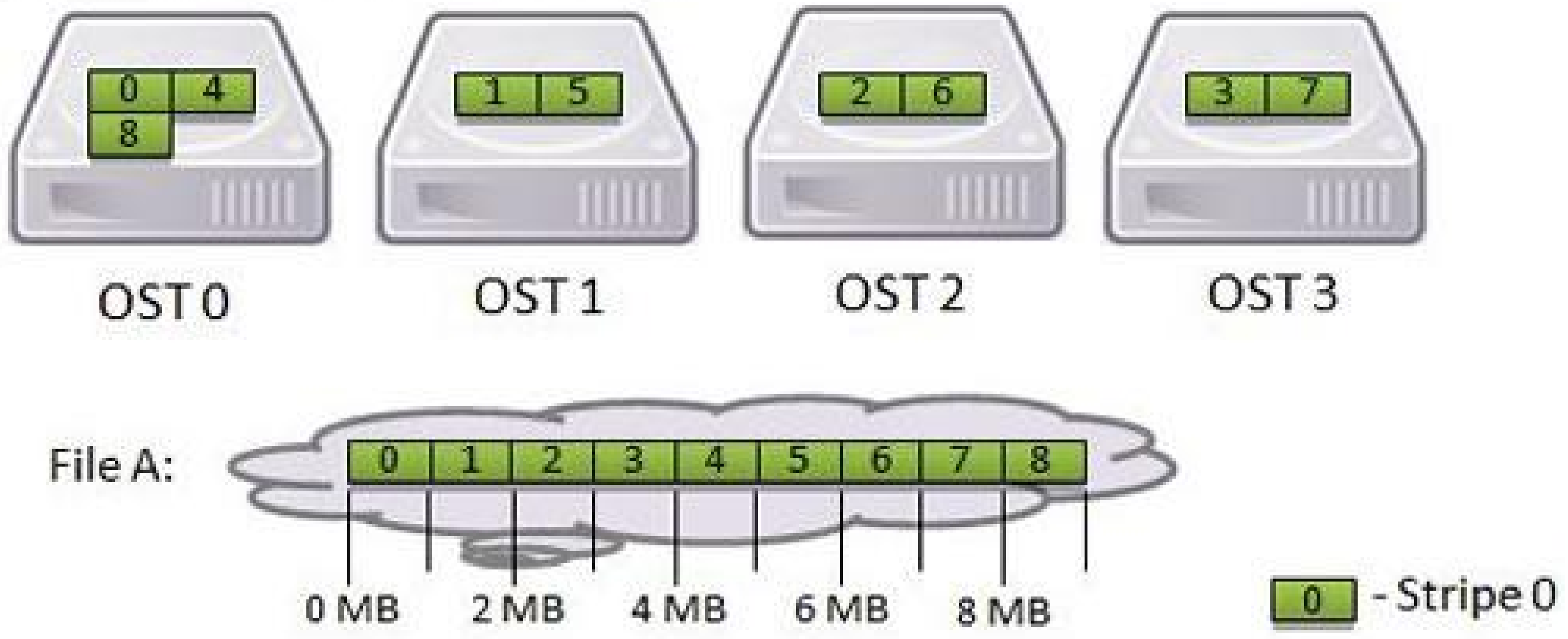
- Distribute a file over several OST..





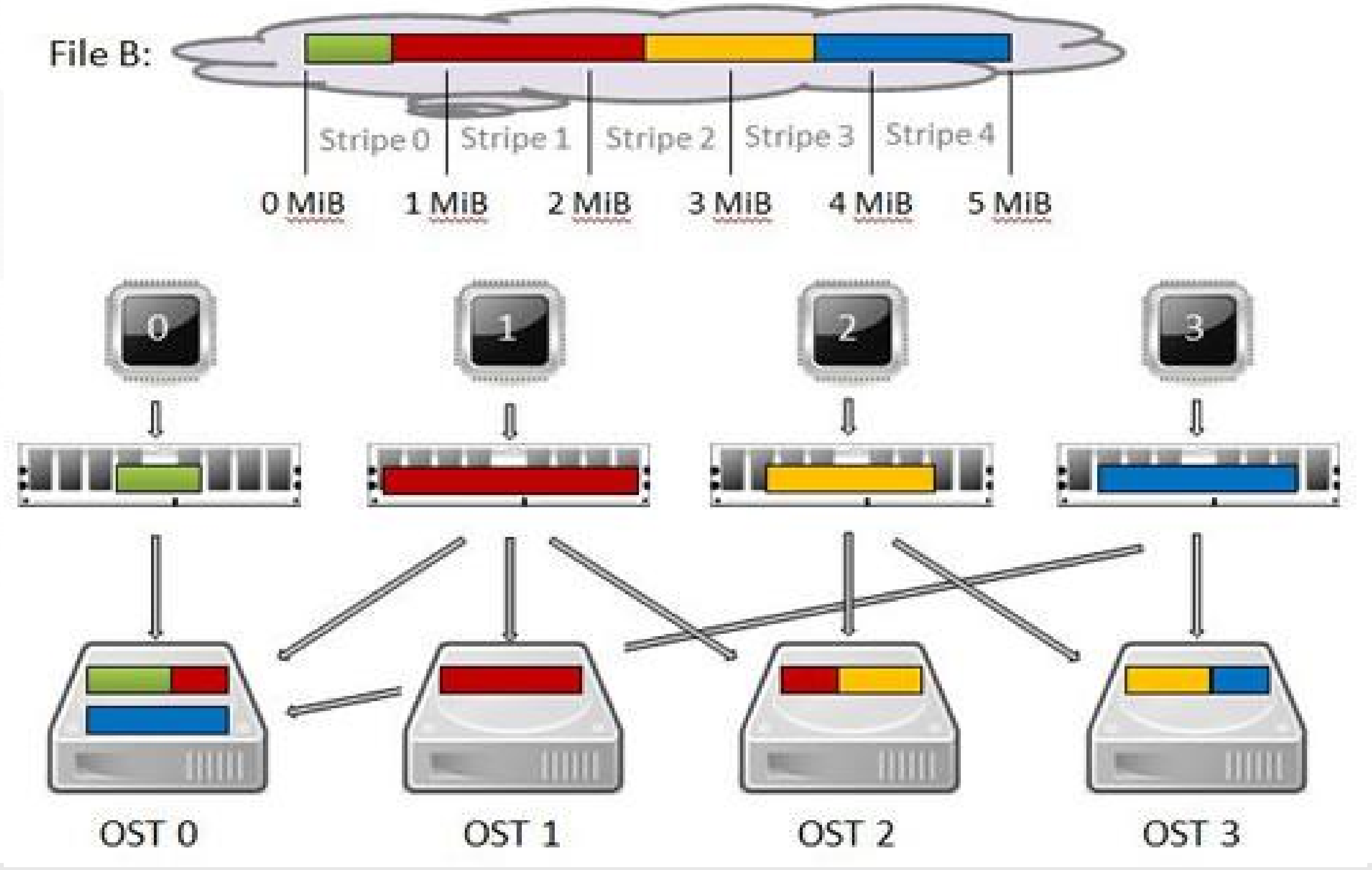
Aligned stripes

- A file large 9Mb
- Stripe size: 1Mb
- Four OST to stripe:



Non aligned stripes

- A file B 5Mb
- Written in parallel with different offset:



Basic Lustre User command

- Lustre's lfs utility provides several options for monitoring and configuring your Lustre environment.
- You can:
  - List OSTs in the File System
  - Search the Directory Tree
  - Check Disk Space Usage
  - Get Striping Information
  - Set Striping Patterns

```
>lfs -help
```

lfs commands: check FS size

-
```
[exact@login ~]$ lfs df -h
UUID                           bytes        Used   Available Use% Mounted on
lustre-MDT0000_UUID            26.3G        9.4G       16.9G  36% /lustre[MDT:0]
lustre-OST0000_UUID             7.3T        4.9T        2.3T  68% /lustre[OST:0]
lustre-OST0001_UUID             7.3T        5.6T        1.7T  77% /lustre[OST:1]
lustre-OST0002_UUID             7.3T        5.6T        1.7T  77% /lustre[OST:2]
lustre-OST0003_UUID             7.3T        5.7T        1.6T  78% /lustre[OST:3]

filesystem_summary:            29.1T       21.8T        7.3T  75% /lustre
```

lfs commands: getstripe

```
 [exact@login exact]$ lfs getstripe  striped2
striped2
stripe_count:  4 stripe_size:   1048576 stripe_offset: -1
striped2/striped_over_how-many
lmm_stripe_count:  4
lmm_stripe_size:   1048576
lmm_pattern:       1
lmm_layout_gen:    0
lmm_stripe_offset: 3
obdidx  objid    objid    group
     3       39804142      0x25f5cee              0
     1       40327224      0x2675838              0
     0       34000522      0x206ce8a              0
     2       41127105      0x2738cc1              0

striped2/striped_over_2
lmm_stripe_count:  2
lmm_stripe_size:   1048576
lmm_pattern:       1
lmm_layout_gen:    0
lmm_stripe_offset: 2
obdidx  objid    objid    group
     2       41127104      0x2738cc0              0
     0       34000521      0x206ce89              0
```

What you can do with setstripe ?

- setting striping on a single file
- setting stripe on a directory
  - Files within directory inherits the same striping policy
- Note: you cannot change the striping policy of a file..
  - To do this:
    - create a new file with appropriate striping policy and then move the original file over this.

lfs commands: setstripe

```
$ lfs setstripe -c 2 striped_over2/
$ lfs getstripe striped_over2
striped_over2
stripe_count:  2 stripe_size:   1048576
stripe_offset: -1
```

Lustre – some advices to check

- Use ls -l only where absolutely necessary
- Avoid using wild cards with GNU commands,
- Place small files on single OST
- Place directories containing many small files on single OSTs
- Set the stripe count and size appropriately for shared files
- Set the stripe count appropriately for applications which utilize a file-per-process

tips to use Lustre at best

identify bottlenecks

- HDs, RAID, controllers, volumes and virtual disks, 10GE, iscsi, LUSTRE fs, local caching/buffers, InfiniBand or Gigabit network, LUSTRE client, access patterns

when accessing files:

- read/write in large chunks, not line by line

when running multiple I/O operations:

- optimize data distribution and disk access (i.e. mpi-i/o)