DATA SCIENCE & SCIENTIFIC COMPUTING

# Lecture 14: storage for HPC systems

Stefano Cozzini
AreaSciencePark
07.12.2021

# Agenda of today's lectures

- ~~Intro:~~
  - ~~Basic concepts on storage~~
  - ~~Basic concept on File Systems~~
- Storage and I/O for HPC
- I/O stack for HPC system
- Parallel FS
- CEPH fs
- ORFEO storage
- Benchmarking I/O storage on ORFEO…

# I/O in HPC

# A couple of citations

“Very few large scale applications of practical importance are NOT data intensive.”

A supercomputer is a device for converting a CPU-bound problem into an I/O bound problem." [Ken Batcher]

# HPC I/O ecosystem

- HPC I/O system is the hardware and software that assists in accessing data during simulations and analysis and keeping data between these activities
- It composed by
  - Hardware: disks, disk enclosures, servers, networks, etc.
  - Software: parallel file system, libraries, parts of the OS
  - Brainware: people who take care of it

# ORFEO storage: hardware

| | FAST storage (NVMe) | FAST storage (SSD) | Standard storage (HDD) | Long term preservation |
|---|---|---|---|---|
| # of server | 4 | | 6 | 1 |
| RAM | 6 x 16GB | | 6 x 16GB | 6 x 16GB |
| Disk per node | 2x 1.6TB NVMe PCIe card | 20 x 3.84TB | 15 x 12TB | 84 x 12TB + 42 x 12TB |
| Storage provider | CEPH parallel FS | CEPH parallel FS | CEPH parallel FS | Network FS (NFS) |
| **RAW storage** | **12TB** | **320 TB** | **1080 TB** | **1,512 TB** |

# I/O subsystem on ORFEO:

- Home
  - once logged in, each user will land in its home in `/u/[name_of_group]/[name_of_user]
  - e.g. the home of user area is in /u/area/[name_of_users]
  - it’s physically located on ceph large FS, and exported via infiniband to all the computational nodes
  - quotas are enforced with a default limit of 2TB for each users
  - soft link are available there for the other areas

```
[cozzini@login ~]$ ls -lrt
total 548398
lrwxrwxrwx 1 cozzini area        18 Apr  7  2020 fast -> /fast/area/cozzini
lrwxrwxrwx 1 cozzini area        21 Apr  7  2020 storage -> /storage/area/cozzini
lrwxrwxrwx 1 cozzini area        21 Apr 16  2020 scratch -> /scratch/area/cozzini
```

# I/O subsystem on ORFEO:

- /Scratch
  - it is large area intended to be used to store data that need to be elaborated
  - it is also physically located on ceph large FS, and exported via infiniband to all the computational nodes

```
[cozzini@login ~]$ df -h /scratch
Filesystem
Size  Used Avail Use% Mounted on
10.128.6.211:6789,10.128.6.213:6789,..:/  598T   95T  503T  16% /large
```

- /fast
  - is a fast space available for each user, on all the computing nodes
  - is intended to be a **fast scratch area** for data intensive application

```
[cozzini@login ~] df -h /fast
Filesystem
Size  Used Avail Use% Mounted on
10.128.6.211:6789,10.128.6.212:6789,..:/   88T  4.3T   83T   5% /fast
```

# I/O subsystem on ORFEO:

- Long term storage:
  - it is NFS mounted via 50bit ethernet link
  - it is intended for long-term storage of final processed dataset
  - Plenty of room to be allocated..

```
[cozzini@login ~]$ df -h  | grep 231
10.128.6.231:/illumina_run          128T   58T   70T  46% /illumina_run
10.128.2.231:/storage                37T   27T  9.9T  74% /storage
10.128.6.231:/long_term_storage     128T  112T   17T  88% /long_term_storage
10.128.6.231:/analisi_da_consegnare 100T   33T   68T  33%
/analisi_da_consegnare
10.128.6.231:/onp_run_1             117T   27T   91T  23% /onp_run
10.128.2.231:/lage_archive          128T   68T   60T  54% /lage_archive
```

# Why do I need I/O for scientific computing ?

Scientific applications use I/O:

- to load initial conditions  or datasets for processing (input)
- to store dataset from simulations for later analysis (output)
- checkpointing to files that save the state of an application in case of system failure

# Flavors of I/O applications

- Two “flavors” of I/O from applications:
  - Defensive: storing data to protect results from data loss due to system faults
  - Productive: storing/retrieving data as part of the scientific workflow
  - Note: Sometimes these are combined (i.e., data stored both protects  from loss and is used in later analysis)
- “Flavor” influences priorities:
  - Defensive I/O: Spend as little time as possible
  - Productive I/O: Capture provenance, organize for analysis

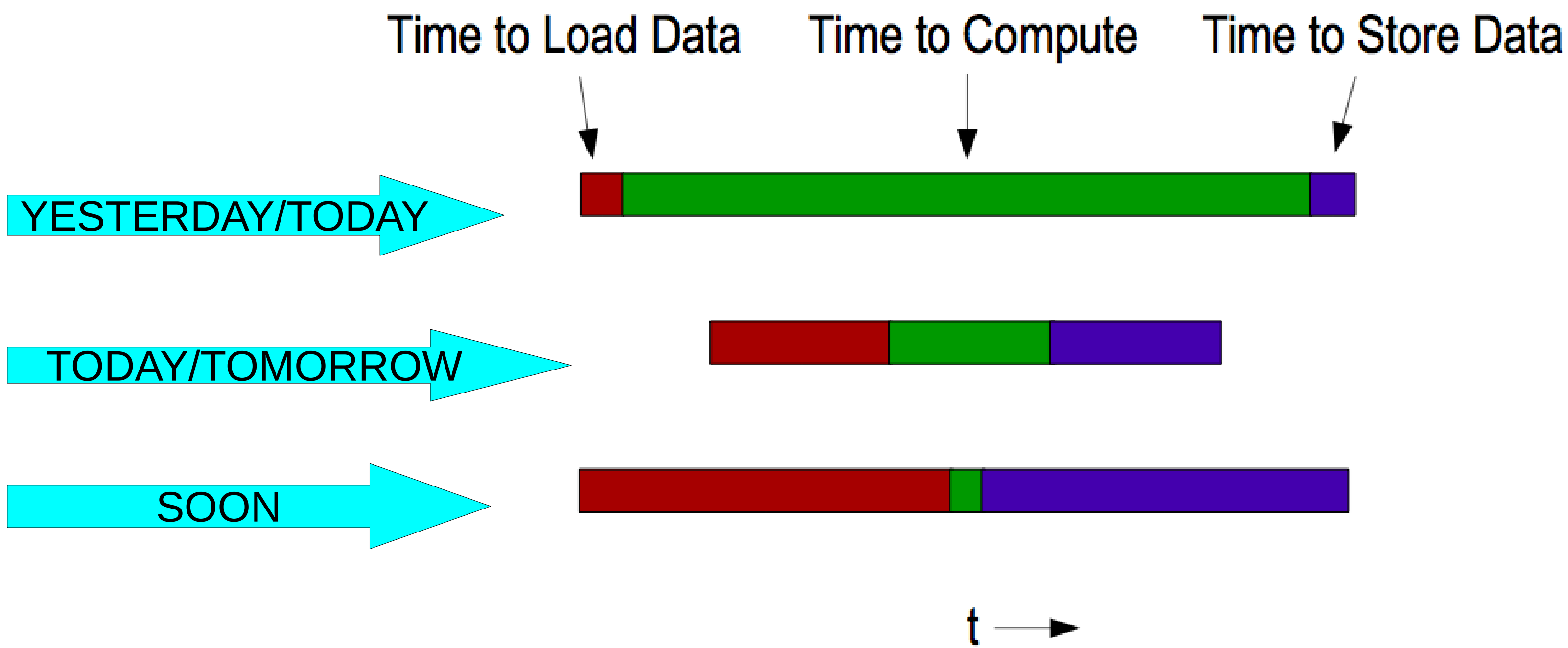
# Preprocessing/Post-processing phases..

- Pre-/post processing:
  - Preparing input
  - Processing output
- These phases are becoming comparable or even larger in time than the computational phases..

# HPC optimization works

- Most optimization work on HPC applications is carried out on:
  - Single node performance
  - Network performance (communication)
  - I/O only when it becomes a real problem

# Do we need to start optimizing I/O ?




We are not counting here pre/post processing phases !!

# I/O challenge in HPC

Large parallel machines should perform large calculations

=> Critical to leverage parallelism in all phases including I/O

(do you remember Amdahl law ?)

# Factors which affect I/O

- How is I/O performed?
  - I/O pattern
  - Number of processes and files.
  - Characteristics of file access.
- Where is I/O performed?
  - Characteristics of the computational system.
  - Characteristics of the file system.

# Challenges in Application I/O

- Leveraging aggregate communication and I/O bandwidth of clients
  - but not overwhelming a resource limited I/O system with uncoordinated accesses!
- Limiting number of files that must be managed
  - Also a performance issue
- Avoiding unnecessary post-processing
- Often application teams spend so much time on this that they never get any further:
  - Interacting with storage through convenient abstractions
  - Storing in portable formats

Parallel I/O software is available  to help fixing ALL these problem

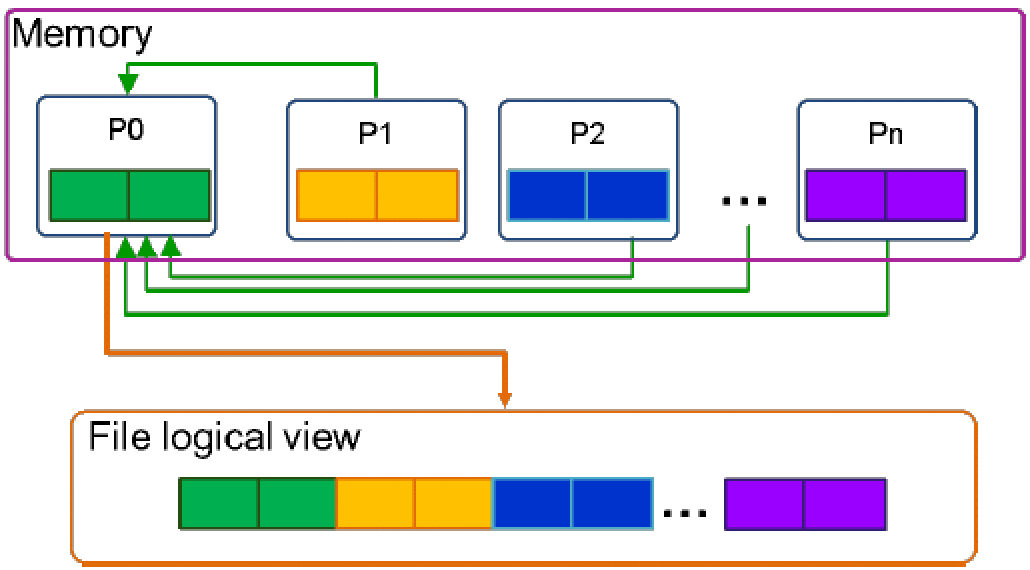
# Application dataset complexity vs I/O

- I/O systems have very simple data models
  - Tree-based hierarchy of containers
  - Some containers have streams of bytes (files)
  - Others hold collections of other containers (directories or folders)
- Applications have data models appropriate to domain
  - Multidimensional typed arrays, images composed of scan lines, variable length records
  - Headers, attributes on data
- How to map from one to the other ?

# How to perform input/output on HPC

# Serial I/O : spokeperson

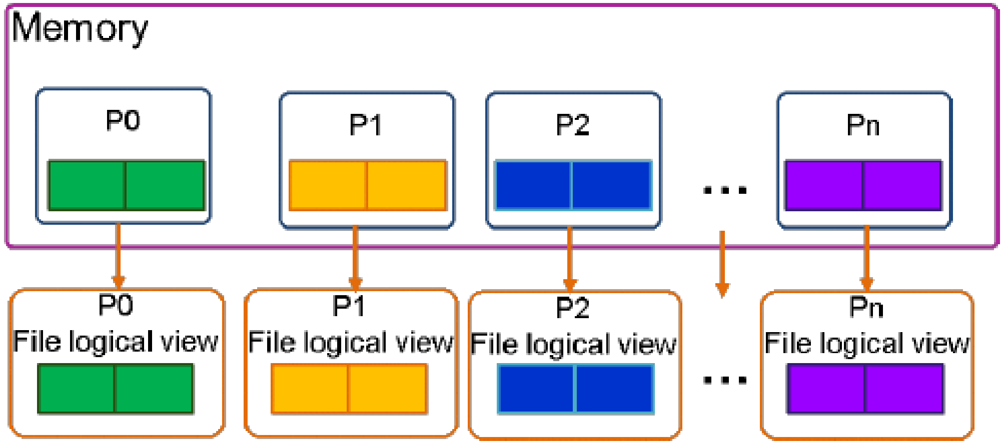
- One process performs I/O.
  - Data Aggregation or Duplication
  - Limited by single I/O process.
- Simple solution, easy to manage, but Pattern does not scale.
  - Time increases linearly with amount of data.
  - Time increases with number of processes.



# Parallel I/O: File-per-Process

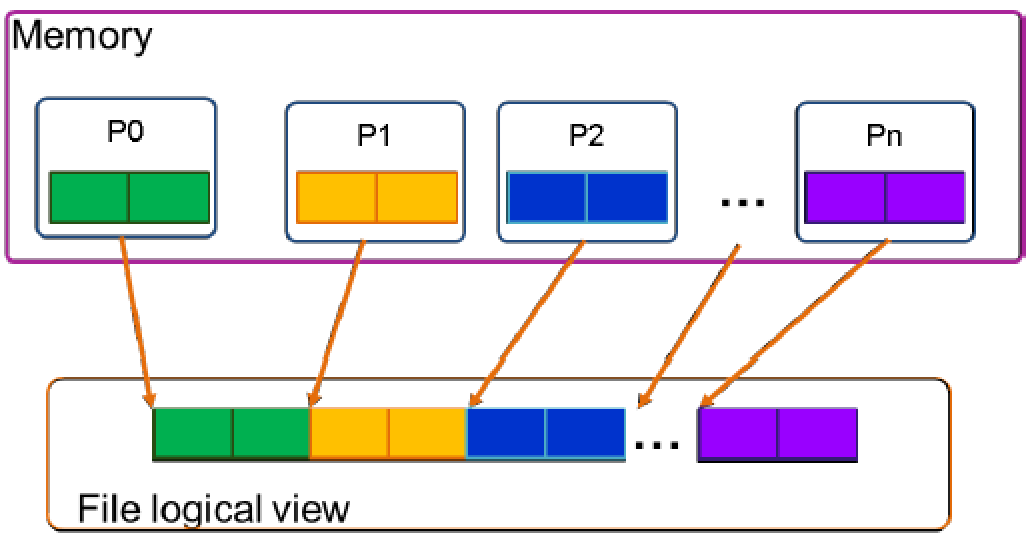
All processes perform I/O to individual files.

- Limited by file system.
  - Pattern does not scale at large number of processes
    - Number of files creates bottleneck with metadata operations.
    - Number of simultaneous disk accesses creates contention for file system resources.
- Manageability issues:
    - What about managing thousand of files ???
    - What about checkpoint/restart procedures on different number of processors ?



# Parallel I/O

- Each process performs I/O to a single file which is shared.
- Performance Data layout within the shared file is very important.
- Possible contention  for file system resources when large number of processors involved..

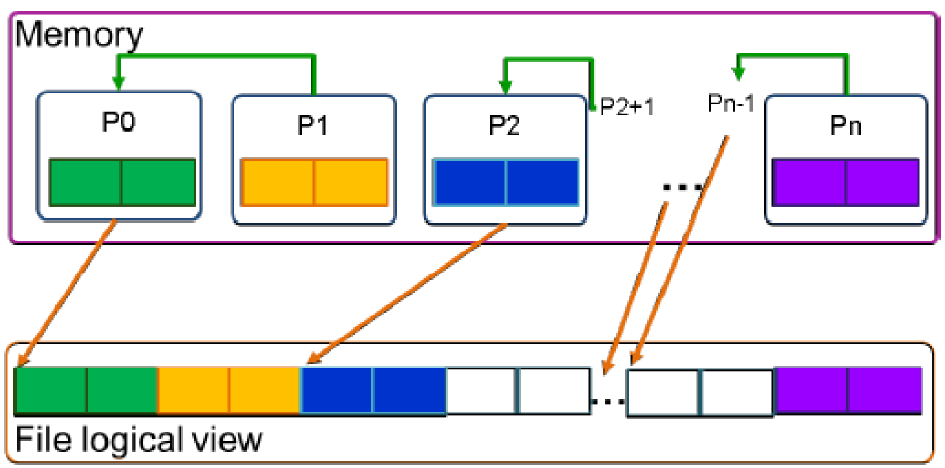


# What does Parallel I/O mean ?

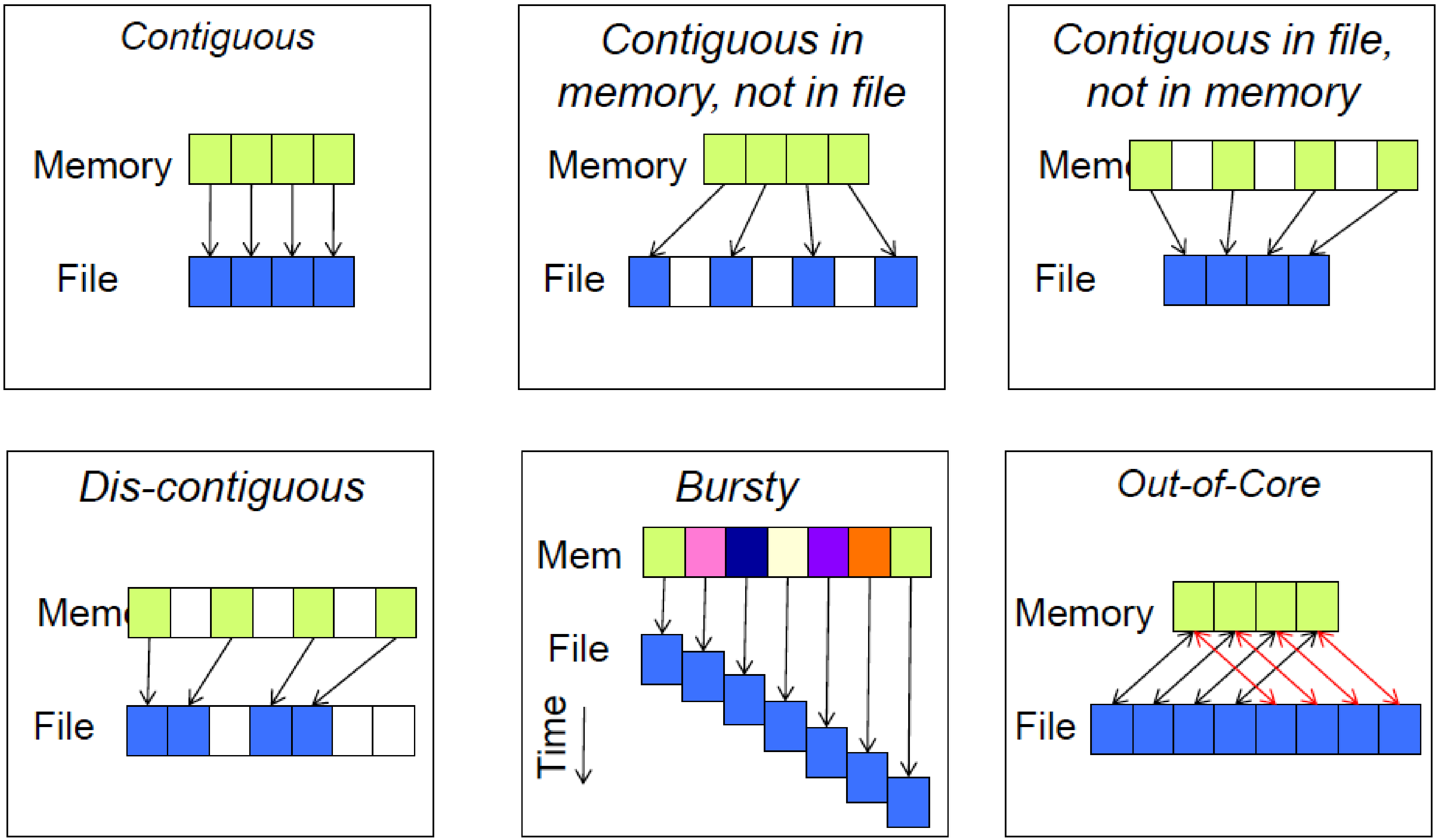
- At the program level:
  - Concurrent reads or writes from multiple processes to a common file
- At the system level:
  - A parallel file system and hardware that support such concurrent access

# Parallel I/O on very large system..

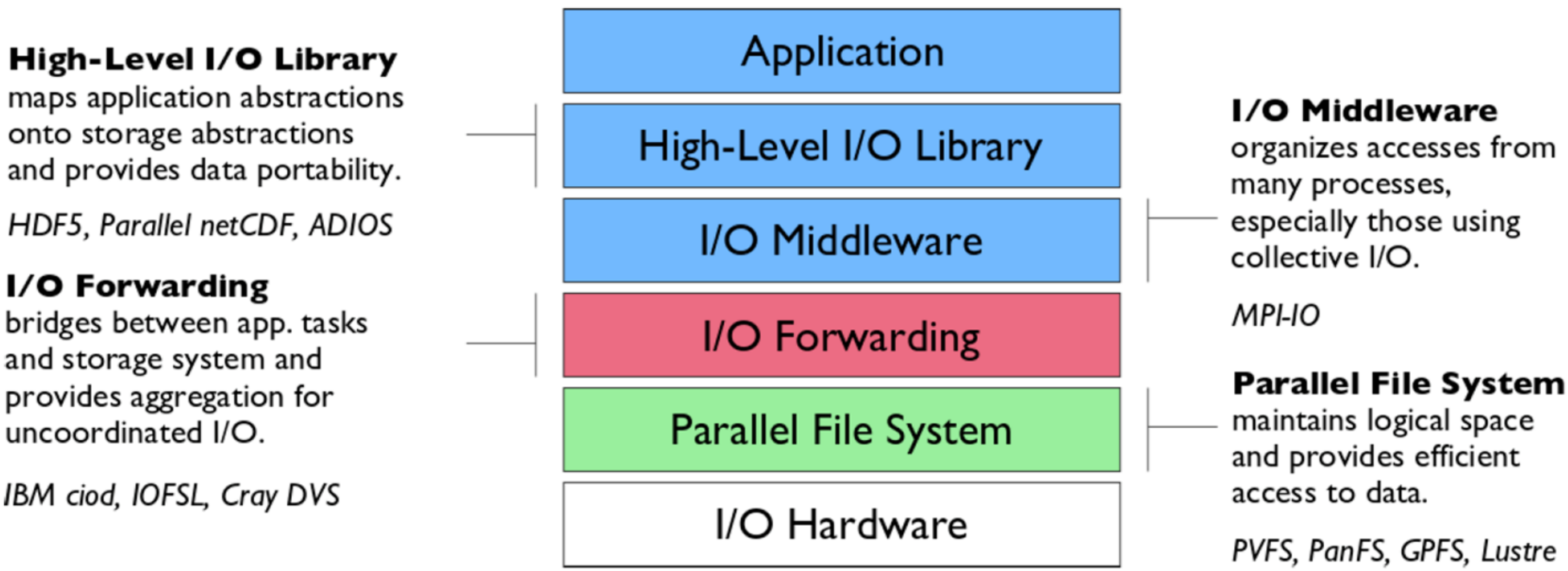
- Accessing  a shared filesystem from large numbers of processes could potentially overwhelm the storage system and not only..
- In some cases we simply need to reduce the number of processes accessing the storage system in order to match number of servers or limit concurrent access.



# Access Patterns



# Software/Hardware stack for I/O



# I/O middleware

- Match the programming model (e.g. MPI)
  - Facilitate concurrent access by groups of processes
  - Collective I/O
  - Atomicity rules
- Expose a generic interface
- Good building block for high-level libraries
- Efficiently map middleware operations into PFS ones
- Leverage any rich PFS access constructs, such as
  - Scalable file name resolution
  - Rich I/O descriptions

# Overview of MPI I/O

- I/O interface specification for use in MPI apps
- Available in MPI-2.0 standard on
- Data model is a stream of bytes in a file
- Same as POSIX and stdio
- Features:
  - Noncontiguous I/O with MPI datatypes and file views
  - Collective I/O
  - Nonblocking I/O
- Fortran/C bindings (and additional languages)
- API has a large number of routines..

NOTE: you simply compile and link as you would any normal MPI program.

# Why MPI is good for I/O ?

- Writing is like sending a message and reading is like receiving one.
- Any parallel I/O system will need to
  - define collective operations (*MPI communicators*)
  - define noncontiguous data layout in memory and file (*MPI datatypes*)
  - Test completion of nonblocking operations (*MPI request objects*)
- i.e., lots of MPI-like machinery needed

NOTE: you simply compile and link as you would any normal MPI program.

# Parallel  I/O using MPI ?

- Why do I/O in MPI?
- Why not just POSIX?
  - Parallel performance
  - Single file (instead of one file / process)
- MPI has replacement functions for POSIX I/O
- Multiple styles of I/O can all be expressed in MPI
  - Contiguous vs non contiguous etc….

# Building blocks for HPC I/O system

- 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 system

- 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

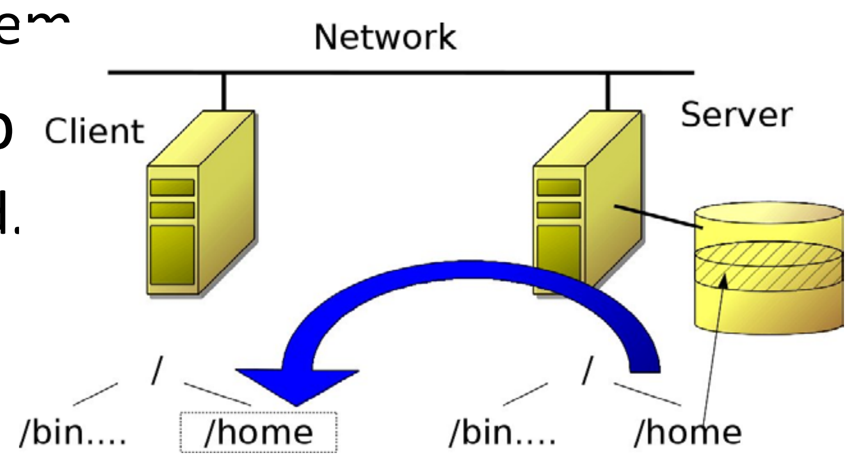
→ Able to coordinate access from multiple clients: scales over many client

- 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

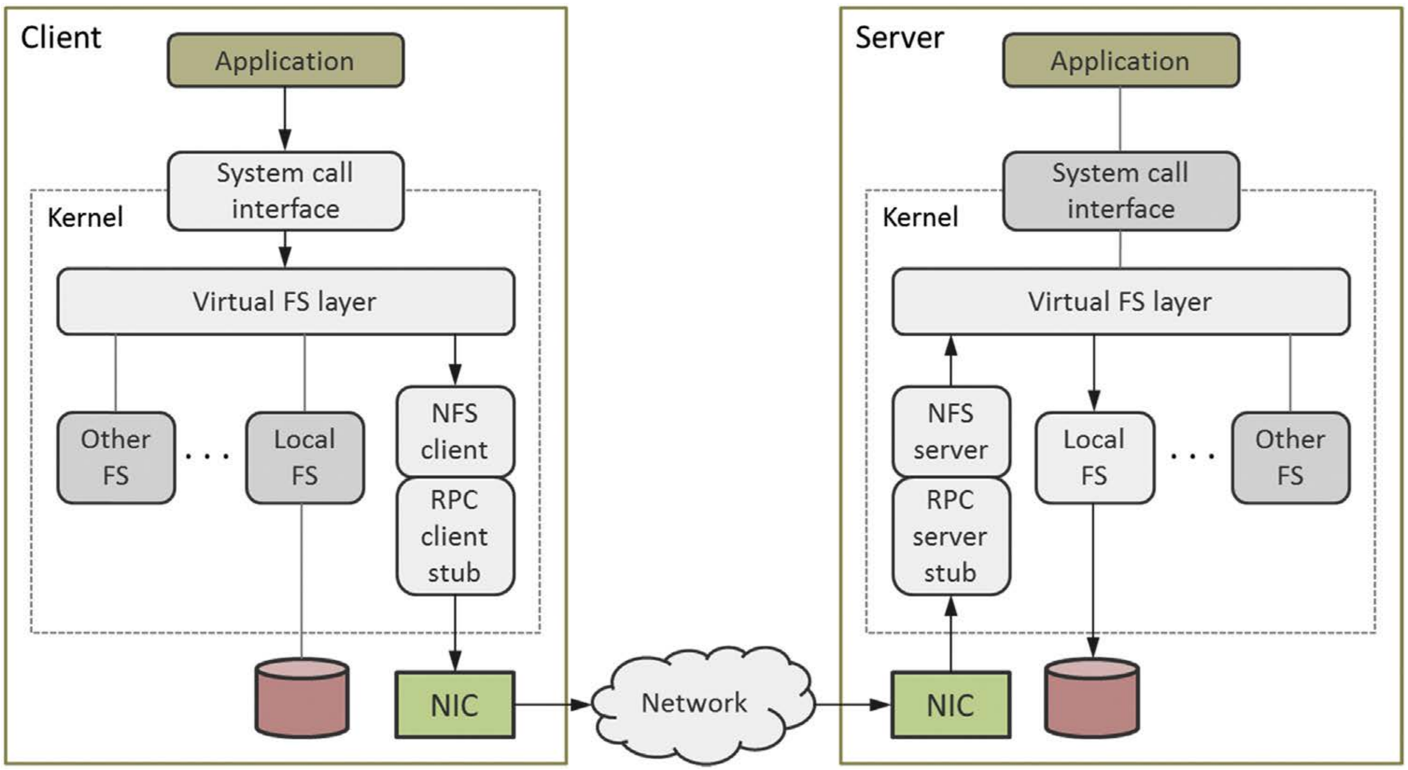
→ Able to scale in both capacity and performance

# Distributed file system

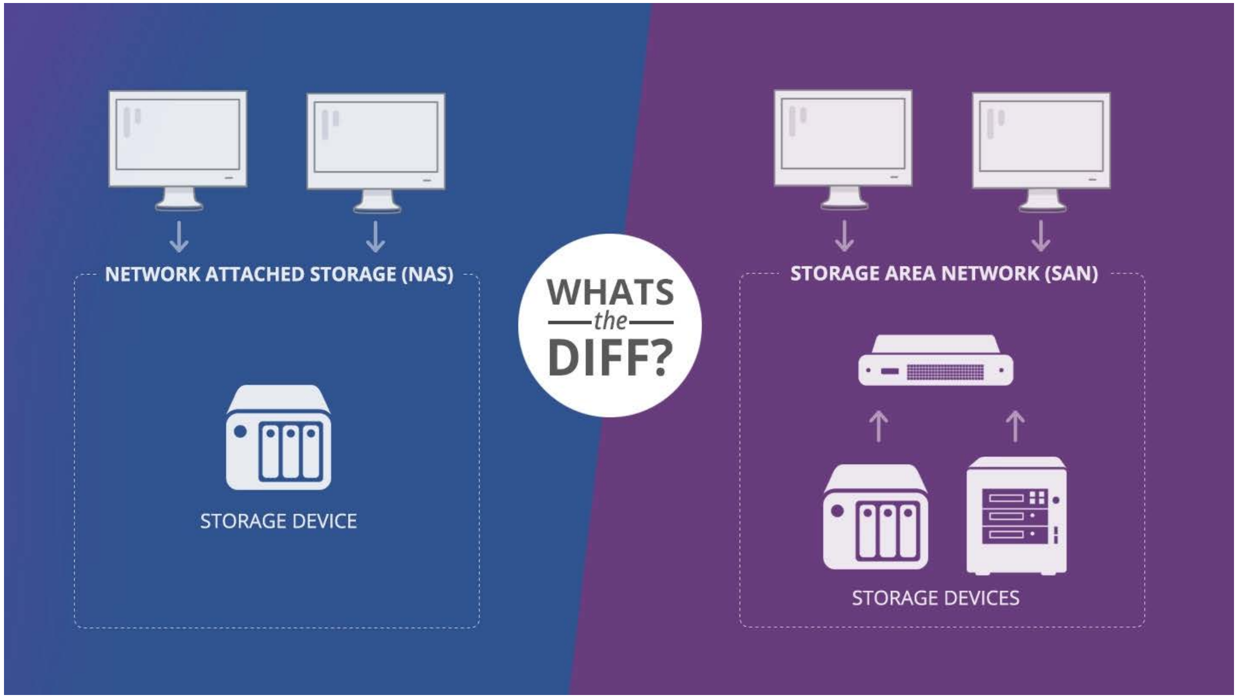
- Distributed file systems are file systems that are capable of handling I/O requests issued by multiple clients over the network.
- FS is “mounted” by several clients (compute nodes/login nodes..)
  - Example: Network File Syste
- Parallel access is possible b
  - Network bandwidth limited.
- Locking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



# NFS architecture



# 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:
  - 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

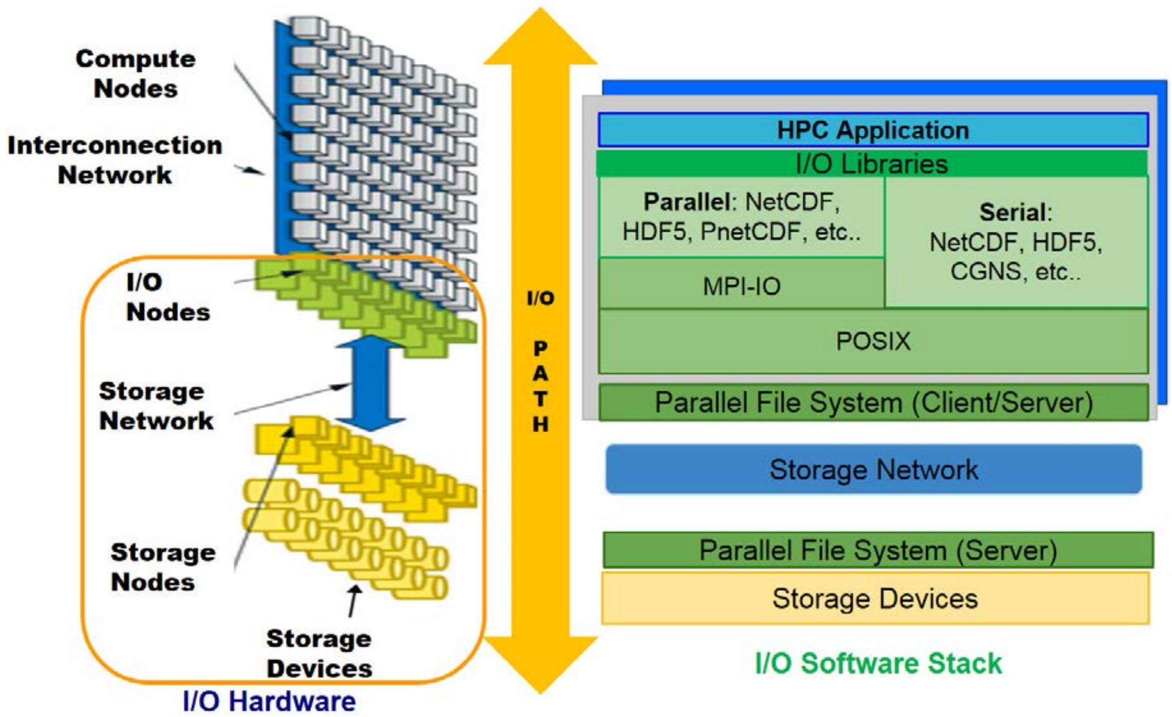
- I/O subsystems are typically very slow compared to other parts of a supercomputer
  → You can easily saturate the bandwidth
- Once the bandwidth is saturated scaling in I/O stops
- Adding more compute nodes increases aggregate memory bandwidth and flops/s, but not I/O

# Parallel File System

# Elements of a PFS

- 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

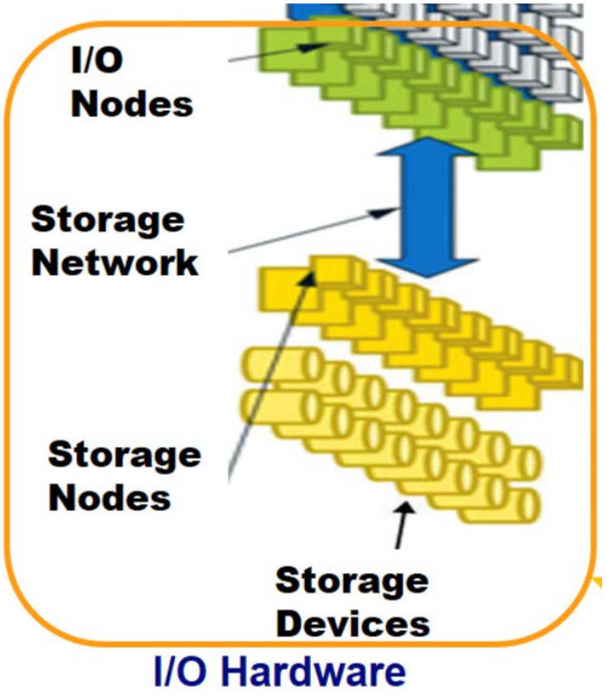
# A graphical view:




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

# Parallel File System: I/O hardware

- Within ORFEO:
  - I/O nodes = Storage Nodes () = CEPH nodes
  - Storage Network= INFINIBAND network for  CEPH
  - 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

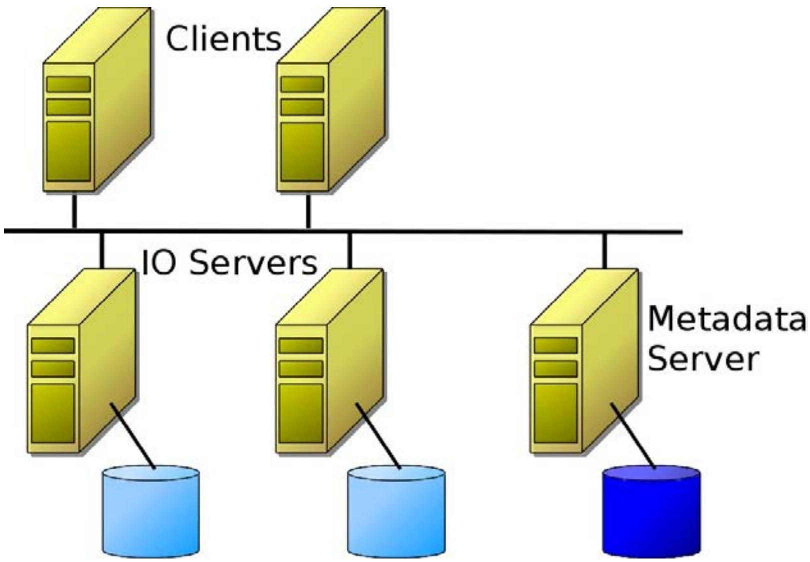


# Hardware to build a PFS:

- Nodes, 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 File System: components

- In general, a Parallel File Systems has the following components
  - Metadata Server
  - I/O Servers
  - Clients

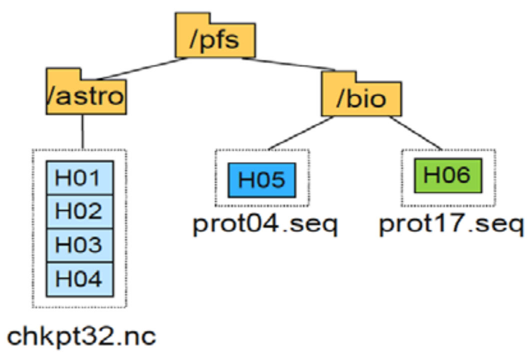


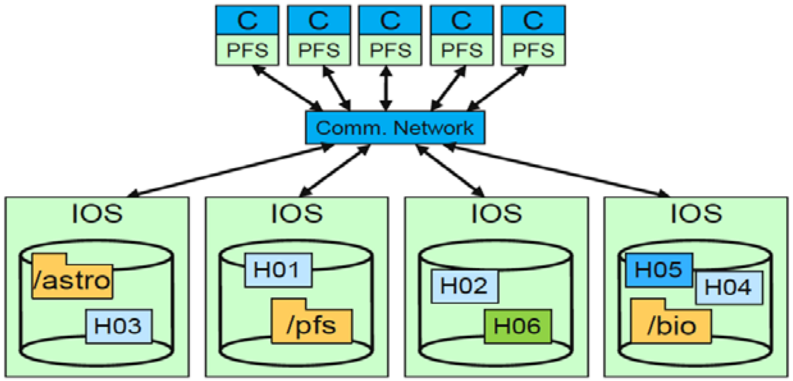
# An important disclaimer..

- 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

# Parallel FS approaches..

- 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

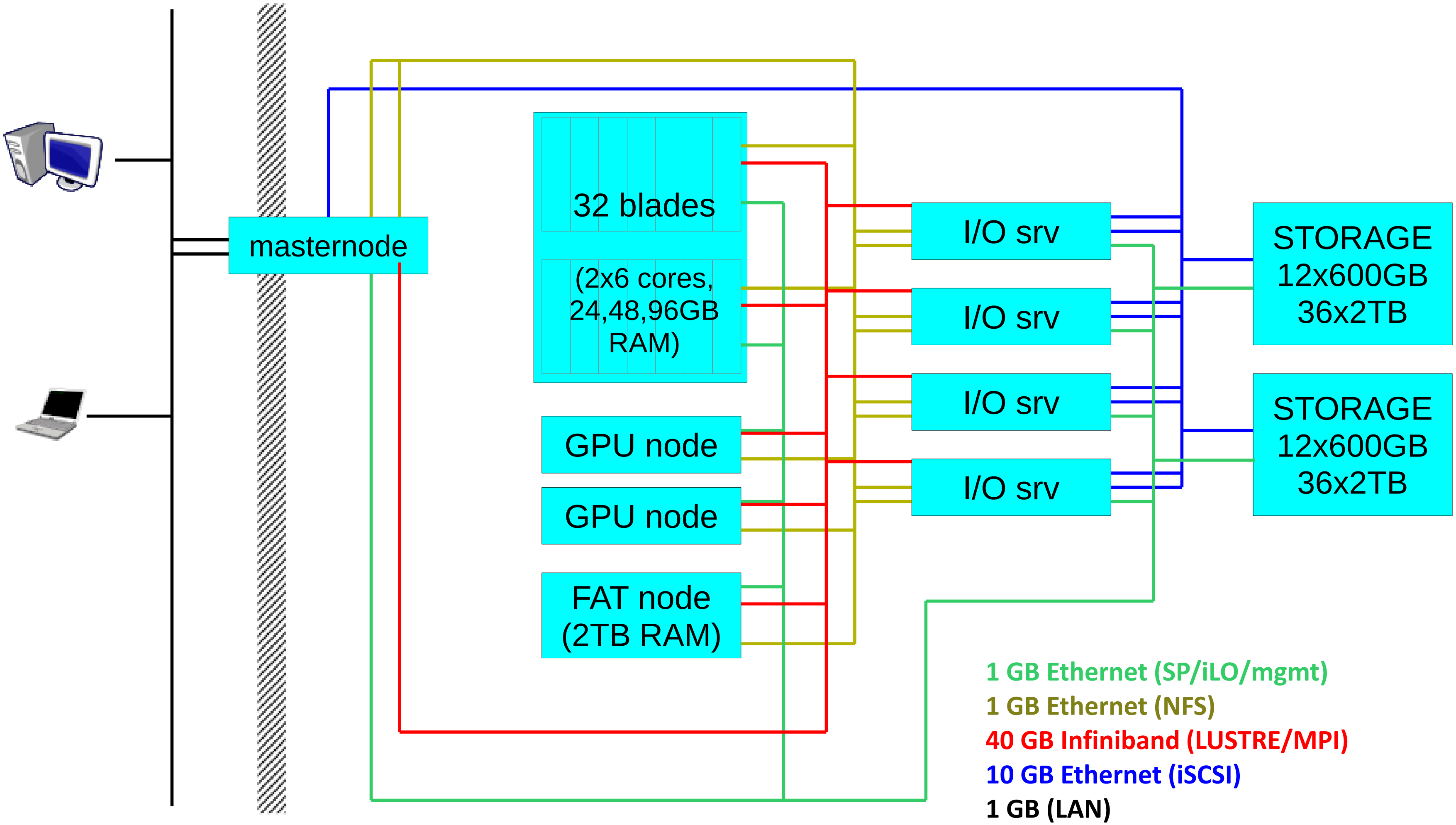


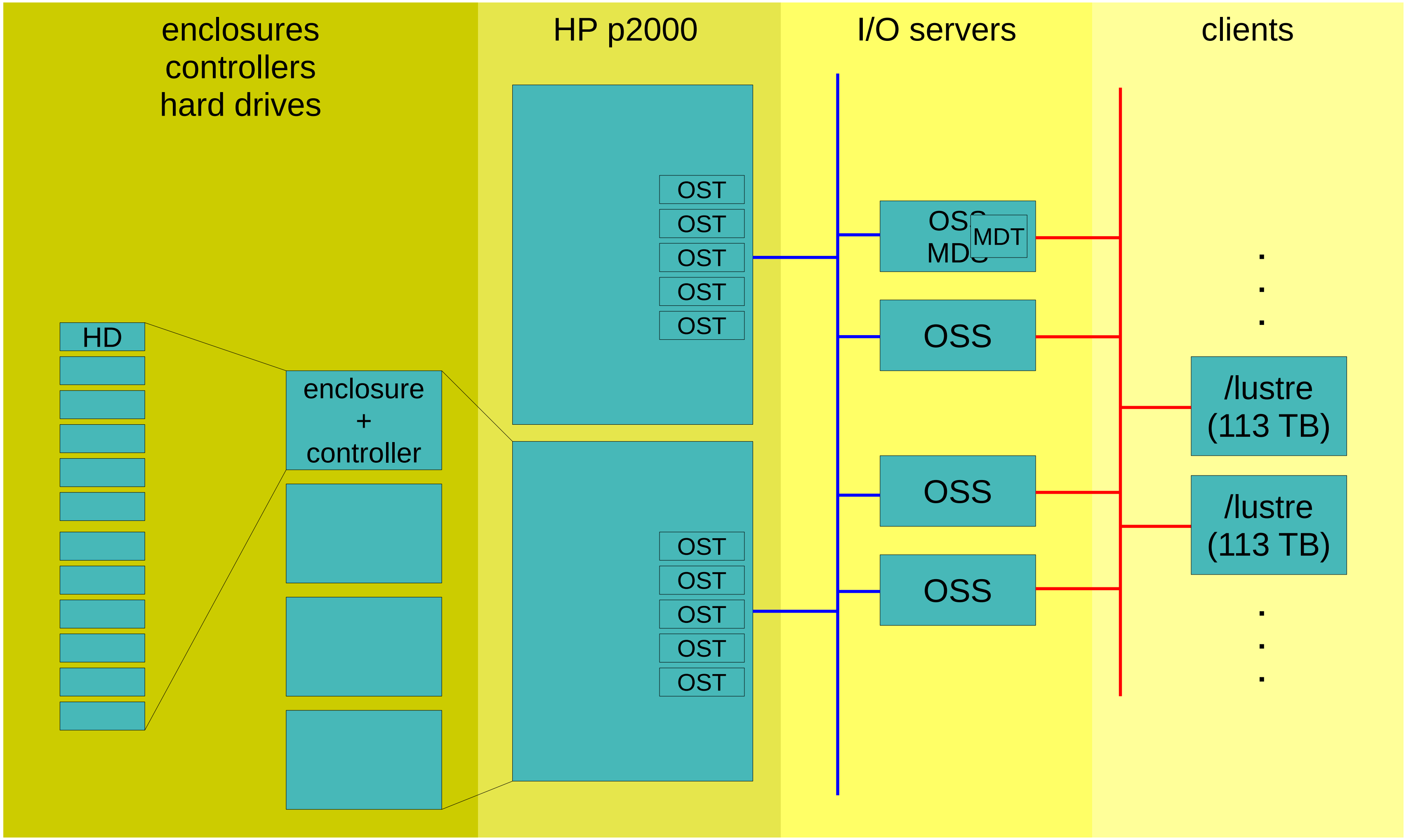
# What is available on the market ?

- 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)

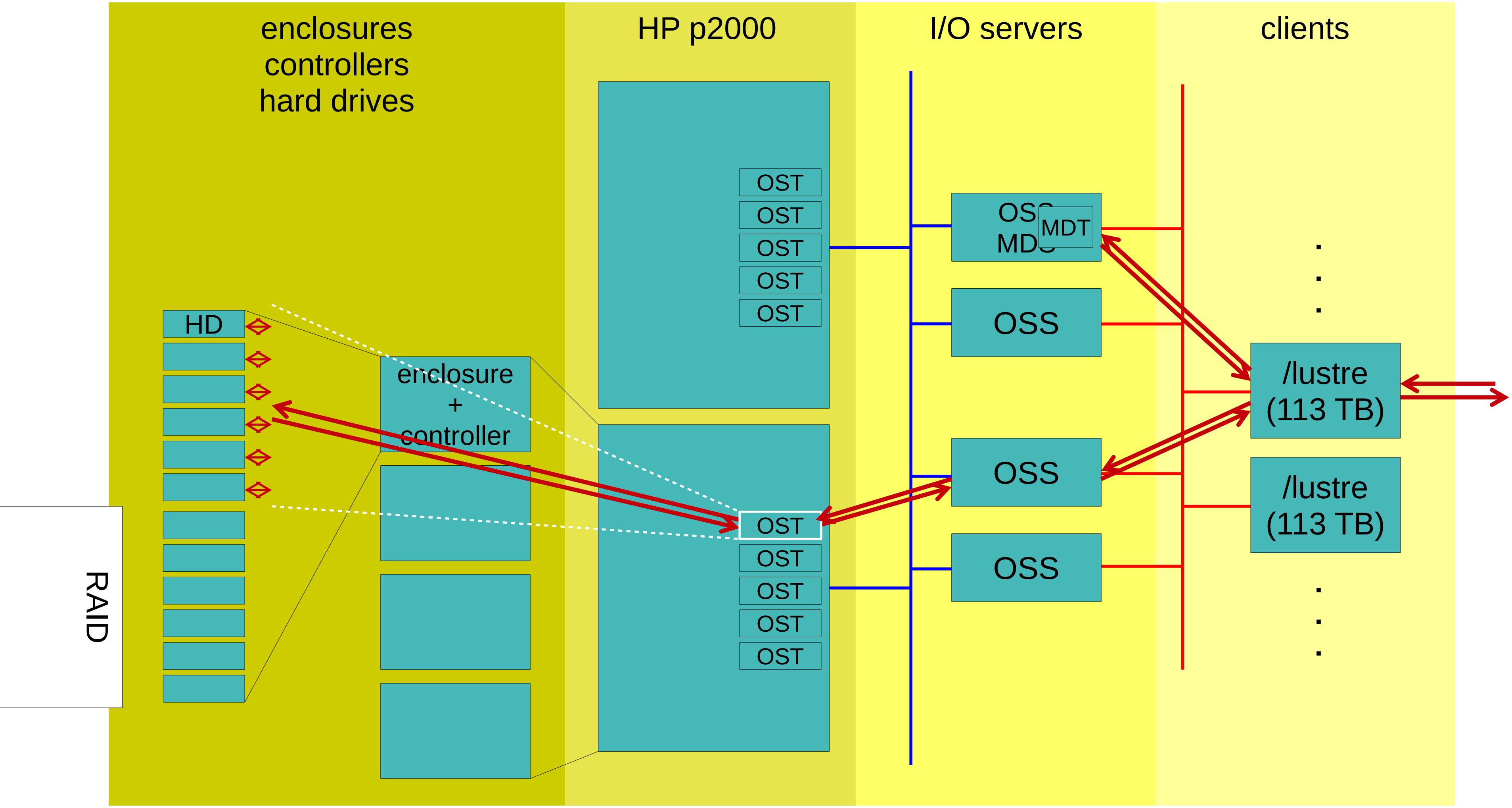
# HPC infrastructure @ CRIBI



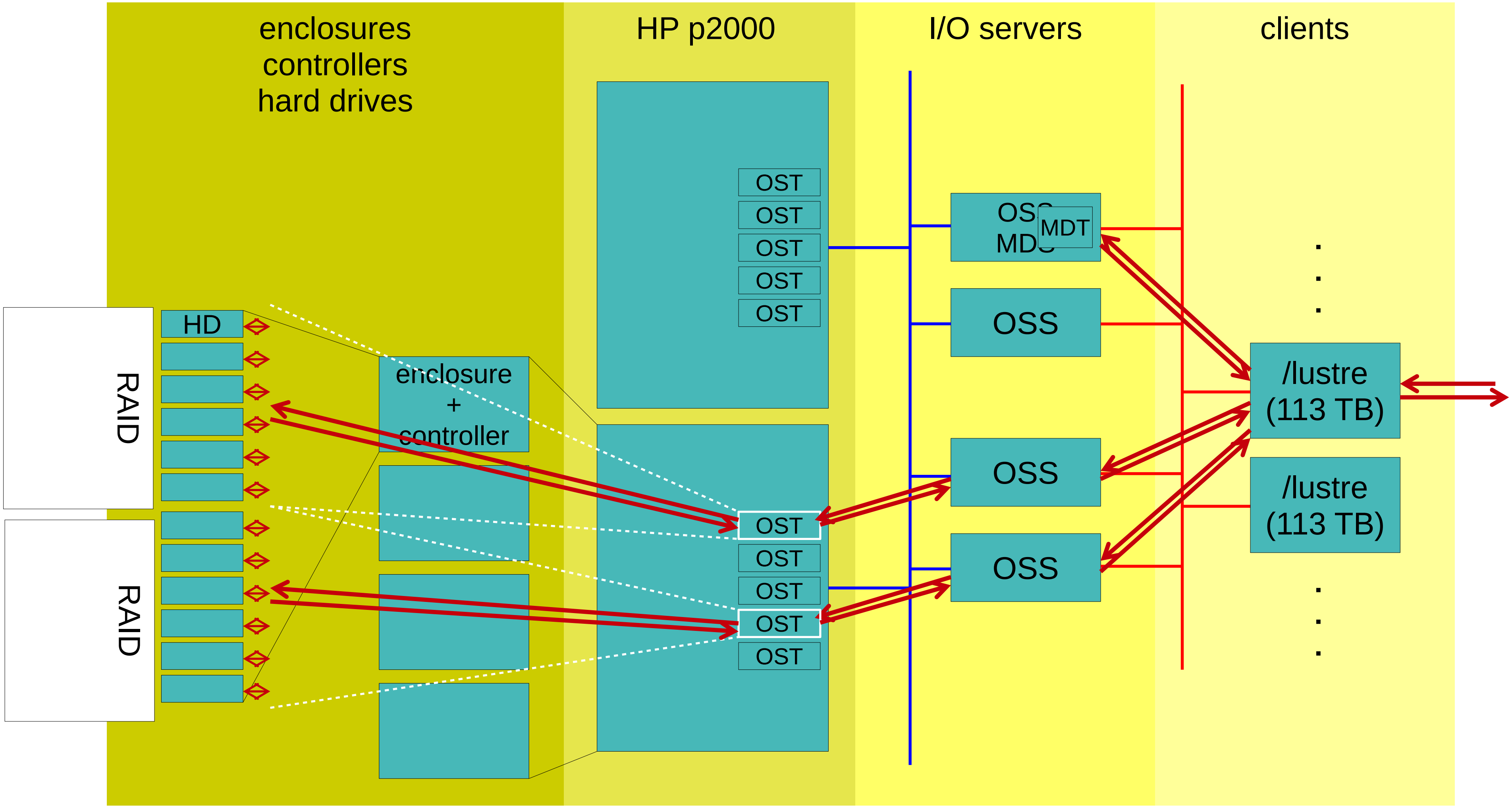
# LUSTRE@CRIBI as storage solution



# accessing LUSTRE filesystem



# why “parallel” filesystem?

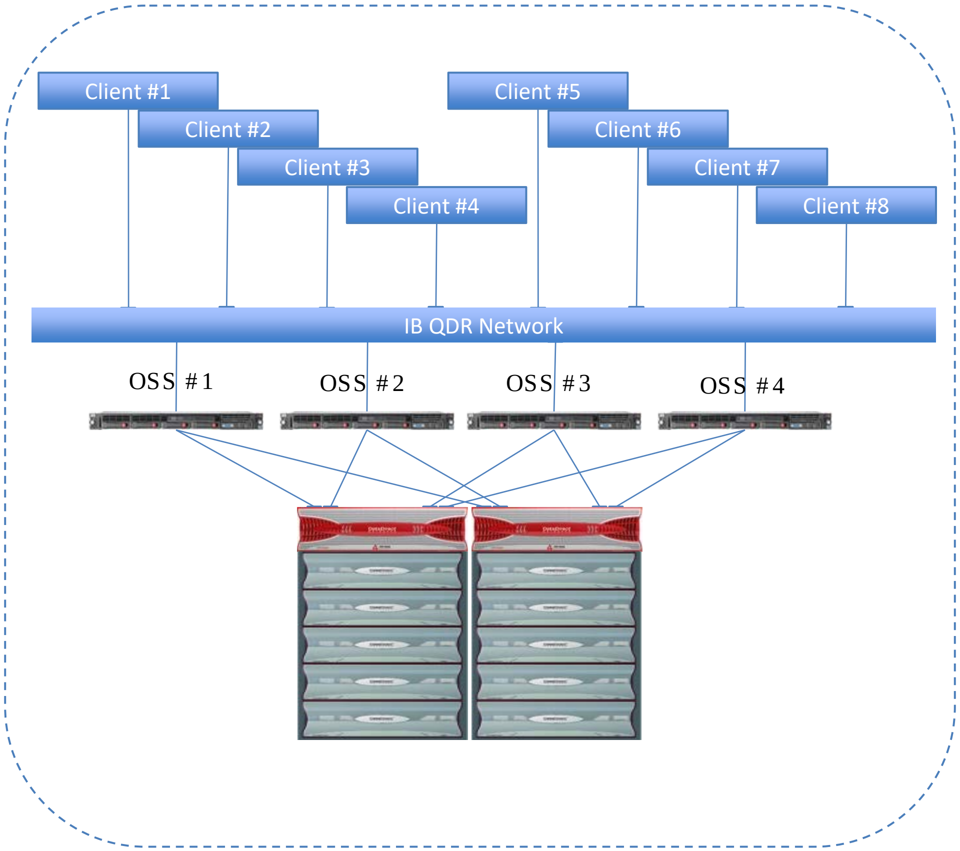


# Expected performance

- 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)*100 = 400 Mb/seconds
      - Disk speed: 100 Mb/seconds
  - Node Aggregate bandwith 400x 2 = 800 Mb/sec

Peak performance : 4 x 800 = 3.2 GB/sec read/write

# overall LUSTRE performance

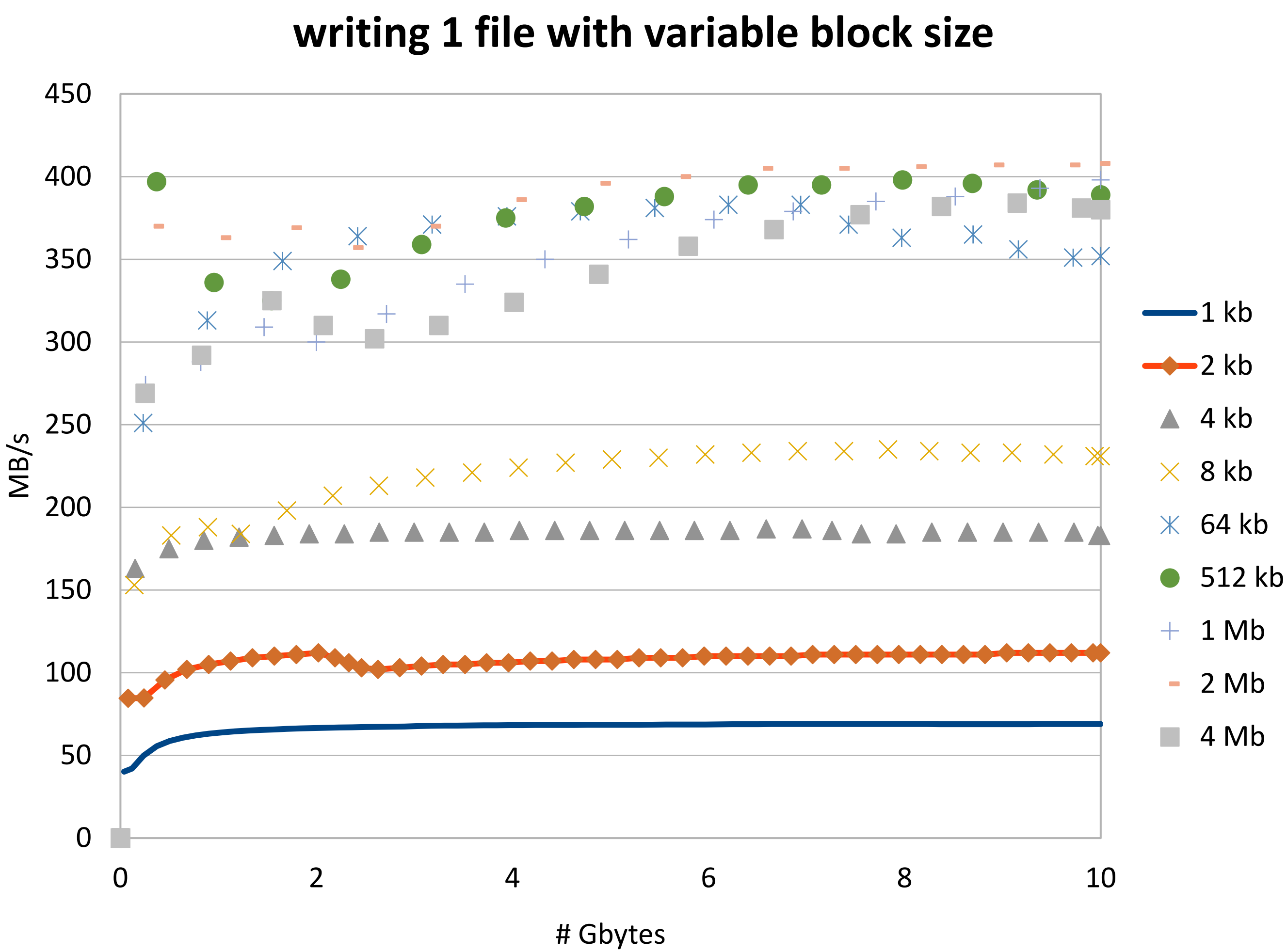



- ➢ sequential write/read by iozone
- ➢ 1 ~ 8 clients, 1 ~ 4 proc/client
- ➢ 32 GB files writing
- ➢ 64 GB files reading

- **~ 1.7 GB/sec writing**
- **32 clients, 32 GB files**

- **~ 1.2 GB/sec reading**
- **32 clients, 64 GB files**

# LUSTRE can be disappointing too...



# ORFEO choice: CEPH

- A unique storage solution for both HPC and Cloud infrastructure
- Main Users: Bioinformatics with many files
- Open and free
- Scalable..