DATA SCIENCE &
SCIENTIFIC COMPUTING

Storage for HPC systems part 2

Stefano Cozzini

AreaSciencePark

16.04.2024

Agenda of these lectures

- Intro
  - Basic concepts on storage
    - ORFEO storage
  - Basic concept on File Systems
  - ORFEO filesystems 11.04

- Storage and I/O for HPC
- I/O stack for HPC system
- Parallel FS 16.04

- CEPH fs
- ORFEO CEPH fs 18.04

Lectures final schedule

- Data management services and tools on the top of data infrastructure: benchmarking storage and FS
  - May 9th
- Seminar: NFFA-DI ecosystem for data (Material science)
  - May 10th
- Seminar: PRP ecosystem for data (Life Science)
  - May 16th
- data management services and tools on the top of data infrastructure: parallel I/O with hdf5
  - May 17th

Intro: Basic concepts on storage

Key metrics

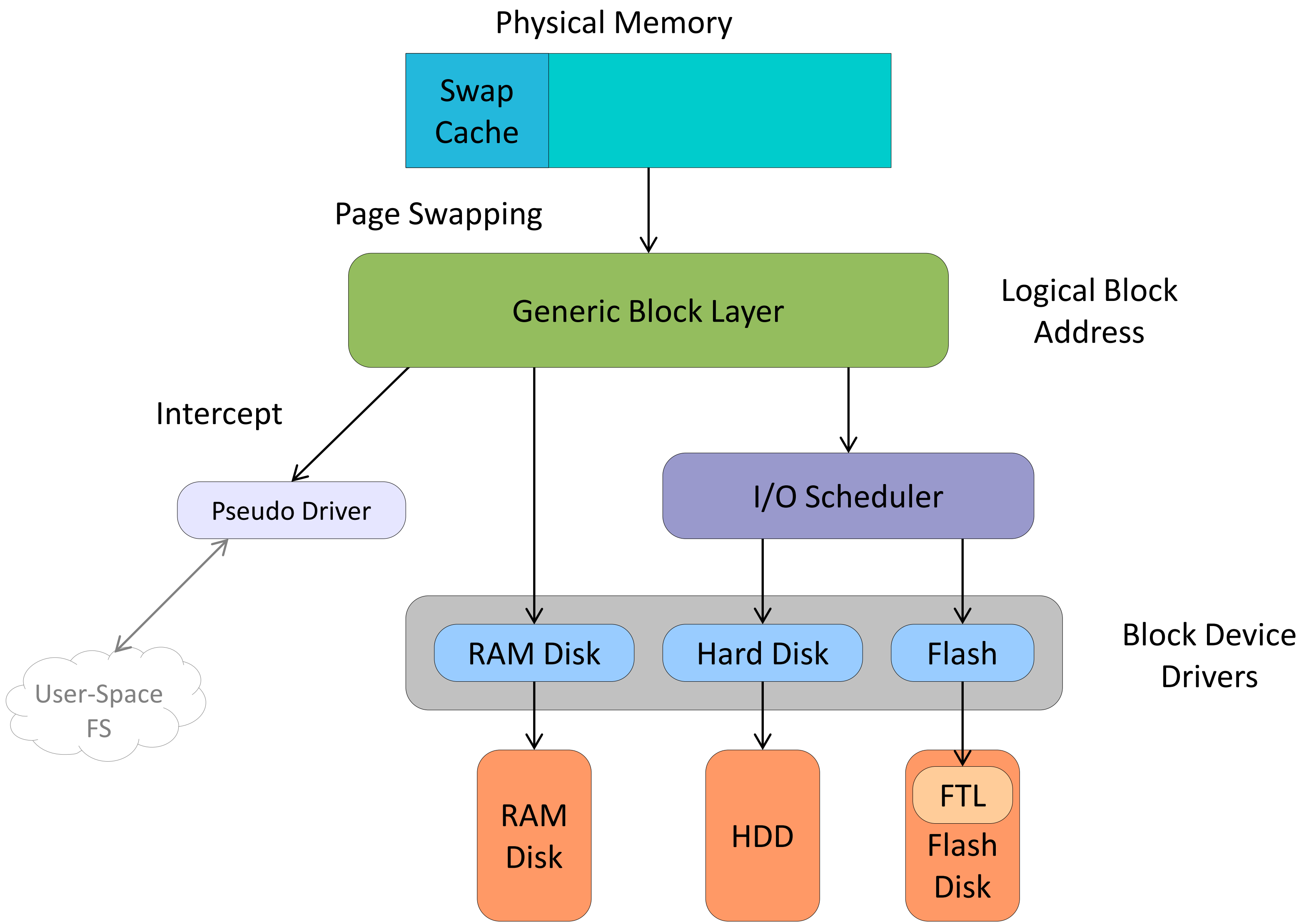
- Bandwidth: volume of data read/written in a second
  → throughput metric
- IOPs: number of I/O request processed by second
  → Is it a latency or a throughput metric ?
- Order of magnitudes:
  - Intel v2/v3 CPU-DRAM: 80/200 GB/s
  - Epyc node CPU-DRAM: 300 GB/sec
  - IB link: 12 GB/s
  - Hard Drive: ~100- 400 MB/s

Storage Hierarchy

- Storage follows a hierarchy with multiple levels:
  - RAM disk, I/O buffers or file system cache
  - Local disk (flash based, spinning disk) (SATA, SAS, RAID, SSD, JBOD, ...)
  - Local network attached device or file system server (NAS, SAN NFS, CIFS, PFS,Lustre, GPFS,CEPH)
  - Tape based archival system (often with disk cache)
  - External, distributed file systems (Cloud storage)

Same as with the memory hierarchy:
Register -> Cache (L1->L2->L3) -> RAM

Storage Hierarchy



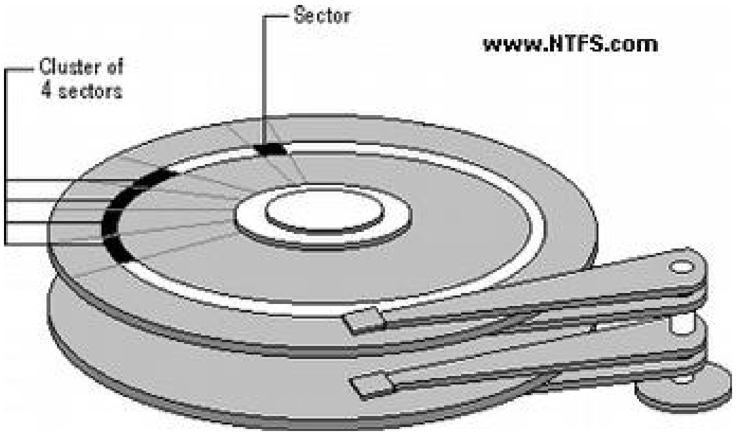
RAM Disk

- Unix-like OS environments very frequently create (small) temporary files in /tmp, etc.
    - faster access and less wear with RAM disk
    - Linux provides "dynamic RAM disk" (tmpfs)
    - only existing files consume RAM
    - automatically cleared on reboot (-> volatile)

```
[cozzini@login ~]$ df
Filesystem                    1K-blocks         Used    Available Use% Mounted
on
devtmpfs                       1915112            0      1915112   0% /dev
tmpfs                          1939960            0      1939960   0% /dev/shm
tmpfs                          1939960        25316      1914644   2% /run
tmpfs                          1939960            0      1939960   0%
/sys/fs/cgroup
/dev/vda1                     41931756     11442916     30488840  28% /
```

Traditional disk: Hard Disk Drive (HDD)

- Rotating mechanical device
  - 7200, 10000, 15000 rpm.
- Head on the right track
  - (seek time) 4 ms
- Head on the right sector
  - (latency) 2ms
  - Capacity: 4-12 TB
- Bandwidth: Read / Write ~ 150/250 MB/s



At constant rotating speed, where should I put my data to get max bandwidth ?

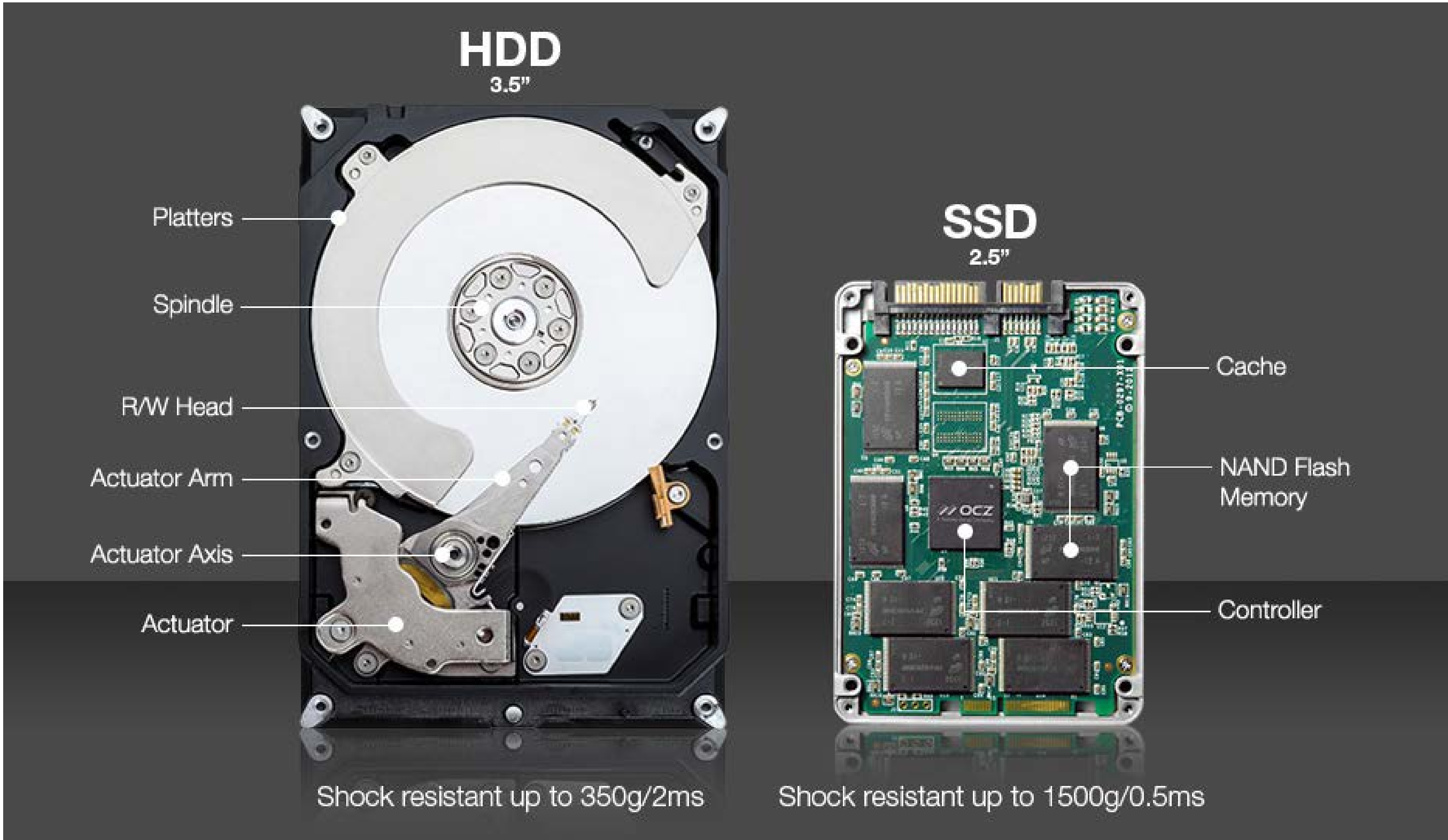
Current HDD technology

- Two main technologies today:
  - Serial Advanced Technology Attachment (SATA)
    - less expensive, and it’s better suited for desktop file storage.
    - Up to 6 Gbit/sec
  - Serial Attached SCSI (SAS)
    - more expensive, and it’s better suited for use in servers or in processing-heavy computer workstations.
    - Up to 12Gbit/sec

Solid State Drive: SDD

- pros:
  - lower access time and latency
  - no moving parts (silent, less susceptible to physical shock, low power consumption and heat production)
  - available over SATA, SAS, PCIe
- cons:
  - expensive, low capacity; usage limited to special purposes only (hardly used for big data-servers)
  - limited write-cycle durability (depending on technology and price)
    - SLC NAND flash ~ 100K erases per cell
    - MLC NAND flash ~ 5K-30K erases per cell
    - TLC NAND flash ~ 300-500 erases per cell

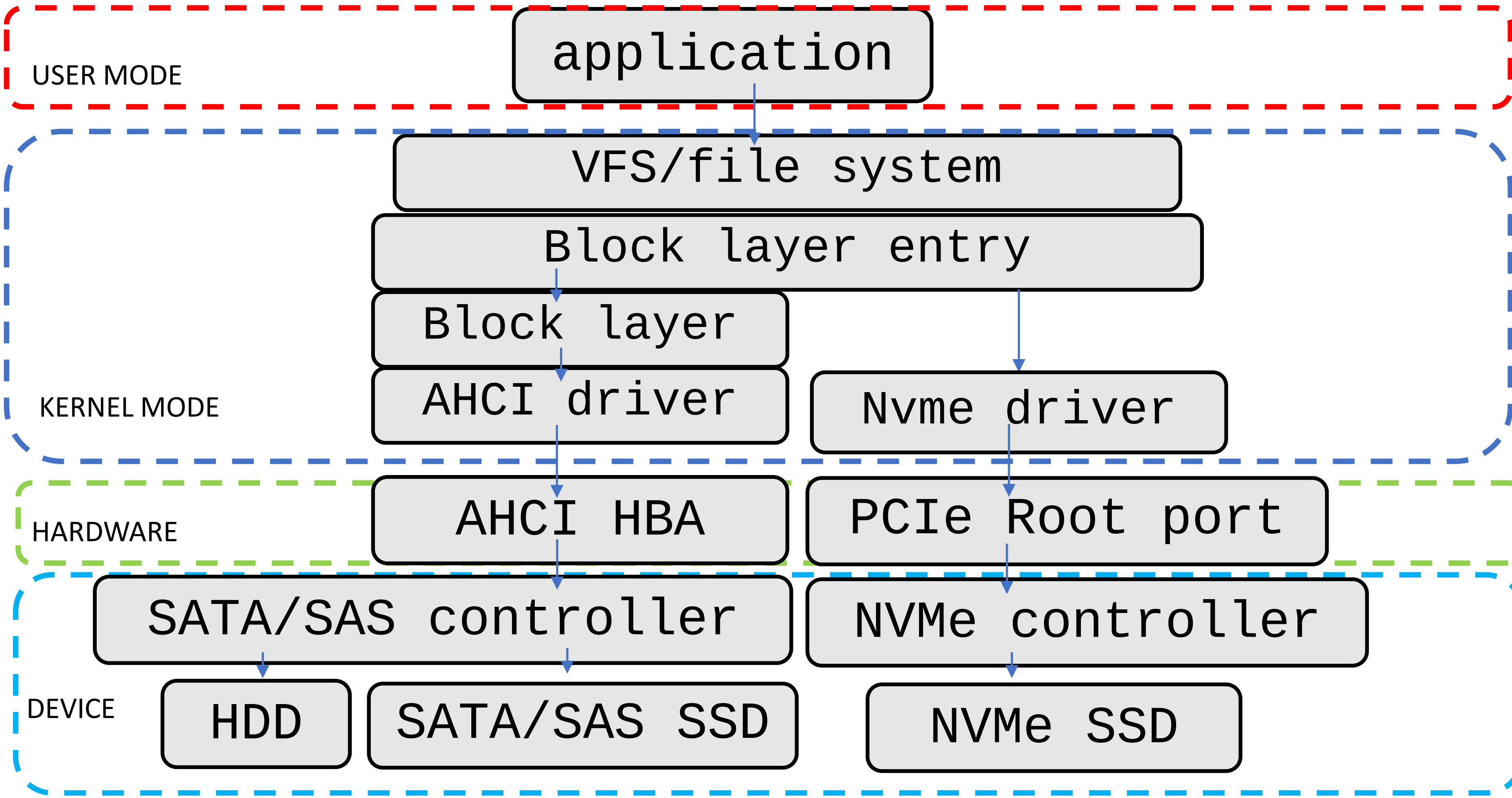
HDD vs SSD


HDD
3.5"
Platters
Spindle
R/W Head
Actuator Arm
Actuator Axis
Actuator
SSD
2.5"
Cache
NAND Flash Memory
Controller
Shock resistant up to 350g/2ms
Shock resistant up to 1500g/0.5ms

NVMe (Non-volatile Memory express)

- NVMe is an “*optimized, high-performance, scalable host controller interface with a streamlined register interface and command set designed for non-volatile memory based storage*.”
- Designed to fix many of the issues of legacy SAS/SATA.
  - SATA /SAS protocols for mechanical drive
  - Now the bottleneck
- Physical connectivity is much simplified, with devices connected directly on the PCIe bus

NVMe (Non-volatile Memory express)



A recent comparison

- UltraStar DC HC620 with SAS 12GB/s interface
  - Sustained transfer rate: 255 MBps read and write
- Samsung 970 Evo with PCIe 3 interface
  - Read speed 3,500 MBps
  - Write speed 2,500 MBps

From https://www.enterprisestorageforum.com/storage-hardware/ssdvs-hdd-speed.html

ORFEO storage: hardware

	FAST storage (NVMe)	FAST storage (SSD)	Standard storage (HDD)	Long term preservation
# of server	4		**10**	1
RAM	6 x 16GB		6 x 16GB	6 x 16GB
Disk per node	-4x 1.6TB NVMe PCIe card	20 x 3.84TB	18 x 12TB + 18x16TB + (on the 2 new server)	84 x 12TB + 84 x 12TB+ 84x 12TB
Storage provider	CEPH parallel FS	CEPH parallel FS	CEPH parallel FS	Network FS (NFS)
RAW storage	**24TB**	**320 TB**	**1872 TB**	**3024 TB**

The ORFEO basic brick: NVME

- Device Type
  - SSD –NVME no hot swap
  - Samsung PM1725b HHHL
- Capacity
  - 1.6 TB
- Form Factor
  - PCI-express
- Performance
  - 6,3 GB/s read
  - 3,3 GB/s write

See:

http://image-us.samsung.com/SamsungUS/PIM/Samsung_1725b_Product.pdf

The ORFEO basic brick: SDD drive

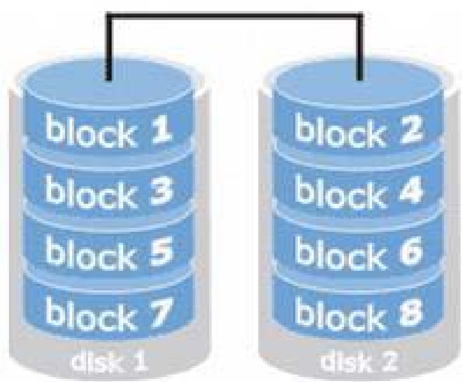
- Device Type
  - SSD driver nearline hot swap
  - Intel SSDSC2KB038T8R
- Capacity
  - 3.84 TB
- Form Factor
  - 2.5"
- Interface
  - SATA 6 Gb/s
- Performance
  - 560 MB/s read
  - 510 MB/s write

The ORFEO basic brick: HDD drive

- Device Type
  - Hard drive - hot-swap - nearline
- Capacity
  - 12 TB
- Form Factor
  - 3.5"
- Interface
  - SAS 12Gb/s
- Performance
  - 255MB/s

The disk bandwidth/reliability problem

- Disks are slow: use lots of them in a parallel file system
- However, disks are unreliable, and lots of disks are even more unreliable



This simple two-disk system is twice as fast, but half as reliable, as a single-disk system

RAID

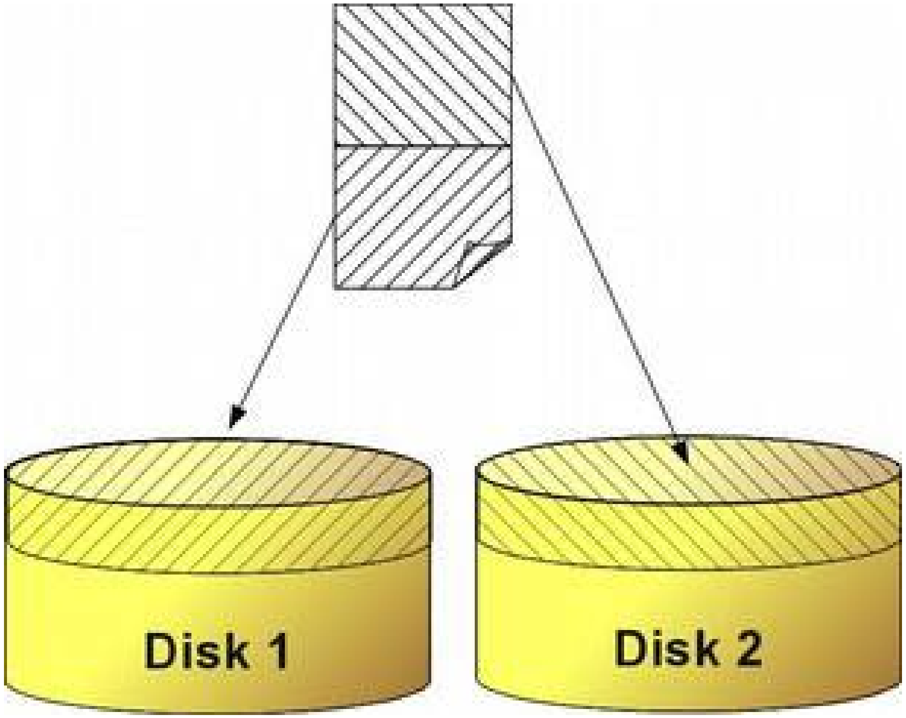
- RAID is a way to aggregate multiple physical devices into a larger virtual device
  - Redundant Array of Inexpensive Disks
  - Redundant Array of Independent Devices
- Invented by Patterson, Gibson, Katz, et al
  - hTtp://www.cs.cmu.edu/~garth/RAIDpaper/Patterson88.pdf
- Redundant data is computed and stored so the system can recover from disk failures
- RAID was invented for bandwidth
- RAID was successful because of its reliability

RAID reliability and performance..

- Reliability or performance (or both) can be increased using different RAID “levels”.
- Let us examine some of the most important:
- Definitions:
  - S: Hard disk drive size.
  - N: Number of hard disk drives in the array.
  - P: Average performance of a single hard disk drive (MB/sec).

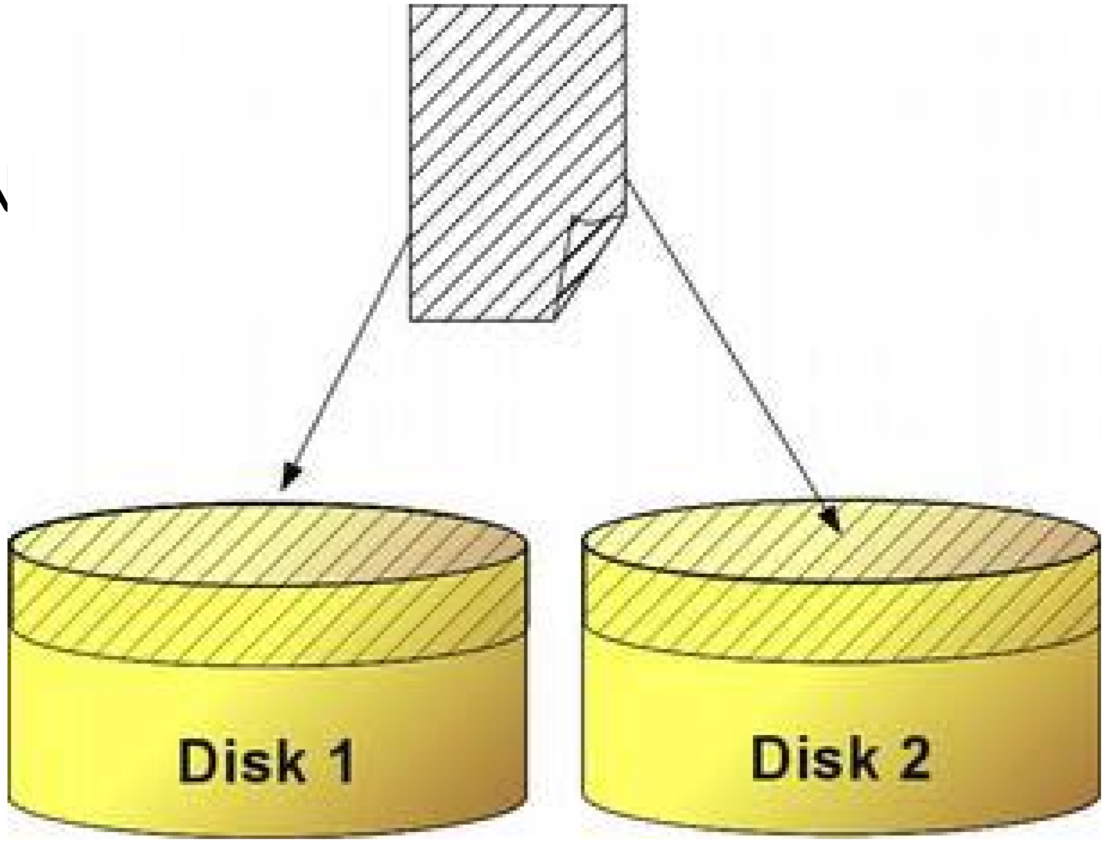
RAID 0: striping

- Performance = P * N
- Capacity = N * S



RAID 1: redundancy

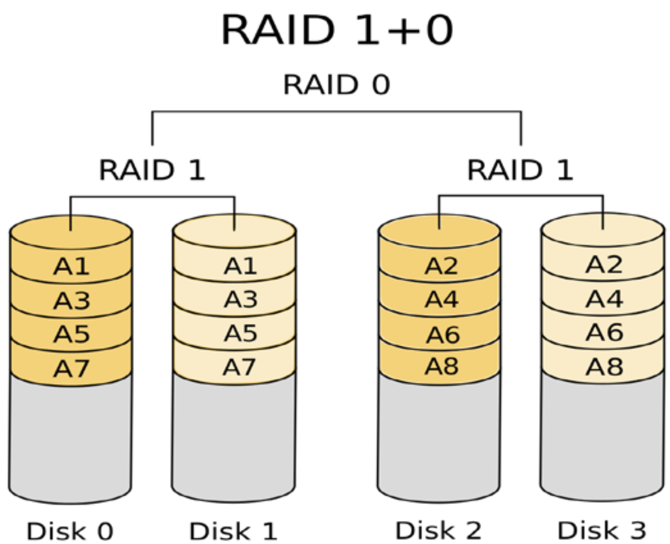
- Write Perf. = P
- Read Perf. = P * N
- Capacity = S

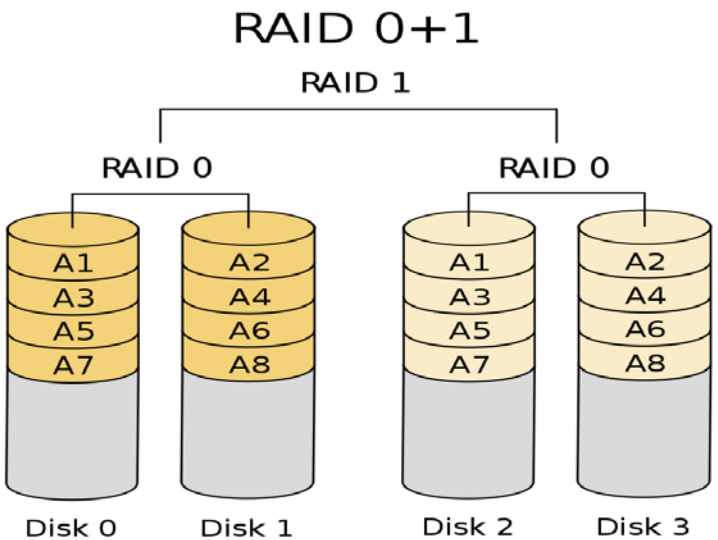


RAID 10: striping +redundancy (1+0 / 0+1)

- Raid 1+0 : mirrored sets in a striped set
- the array can sustain multiple drive losses so long as no mirror loses all its drives

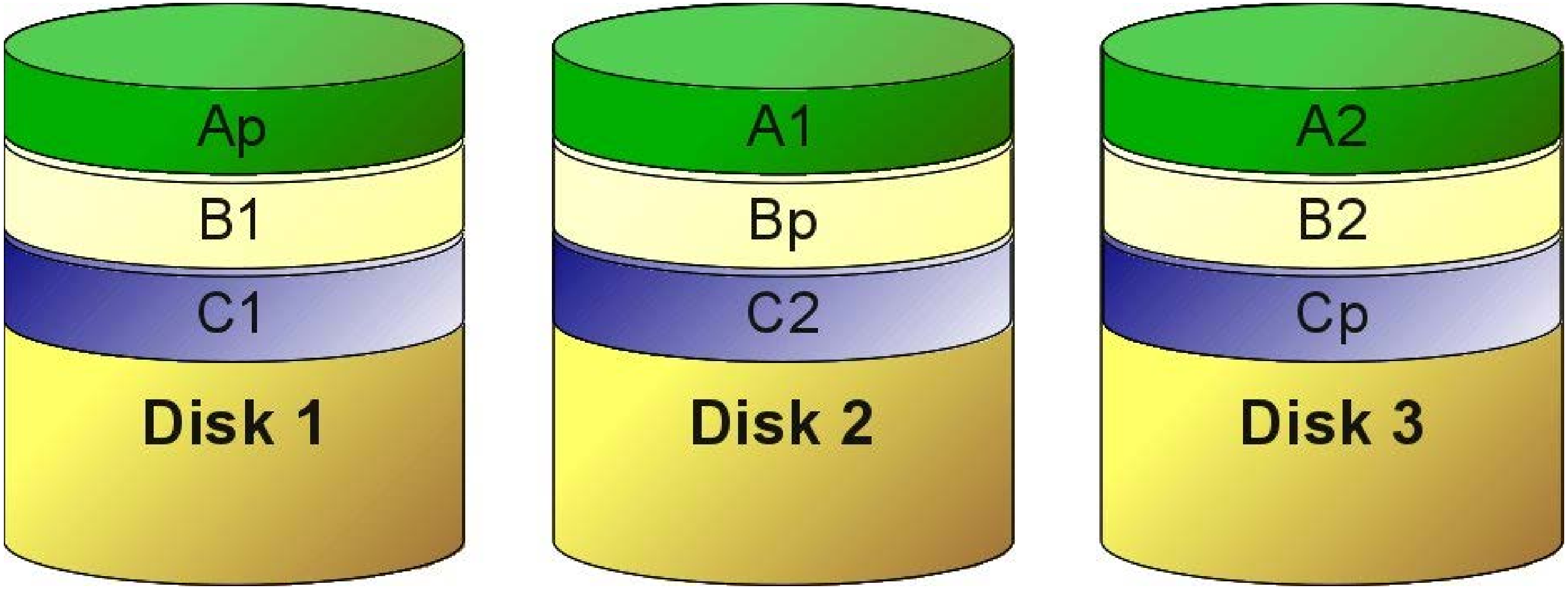
- Raid 0+1: striped sets in mirrored set
- if drives fail on both sides of the mirror the data are lost





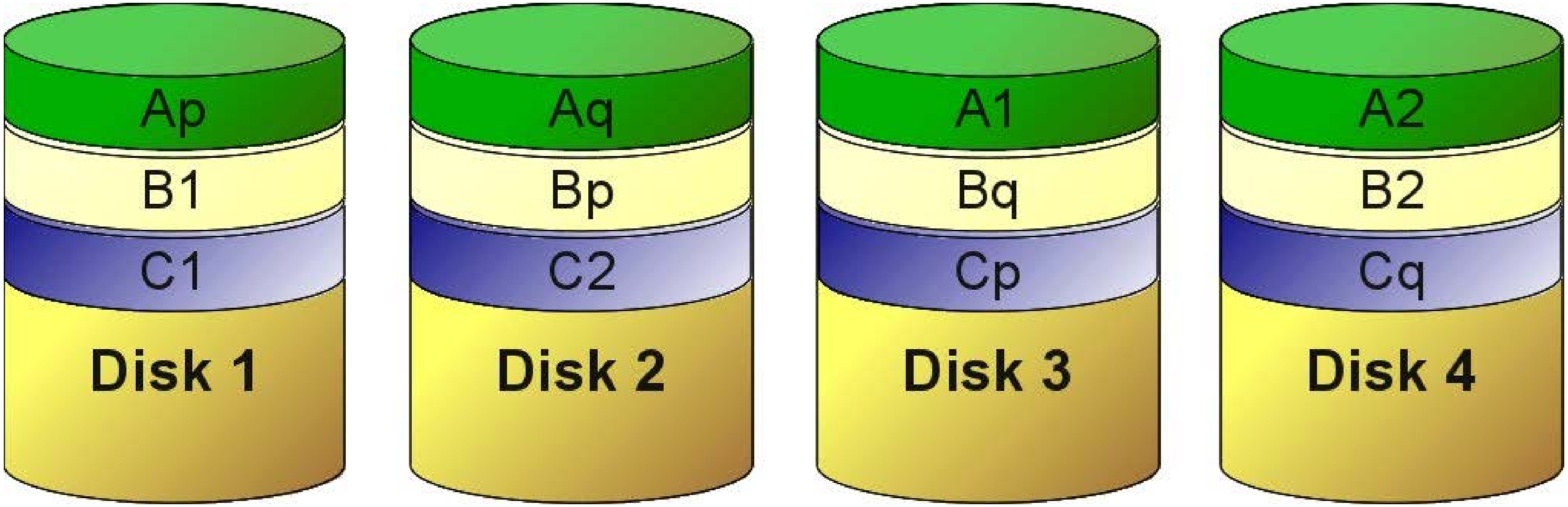
RAID 5

- One disk can fail
- Distributed parity



RAID 6

- Two disks can fail
- Double distributed parity code



RAID Parameters

Level	Description	Minimum # of drives	Space Efficiency	Fault Tolerance	Read Benefit	Write Benefit
RAID 0	Block-level striping without parity or mirroring.	2	1	0 (none)	nX	nX
RAID 1	Mirroring without parity or striping.	2	1/n	n-1 drives	nX	1X
RAID 4	Block-level striping with dedicated parity.	3	1-1/n	1 drive	(n-1)X	(n-1)X
RAID 5	Block-level striping with distributed parity.	3	1-1/n	1 drive	(n-1)X	(n-1)X
RAID 6	Block-level striping with double distributed parity.	4	1-2/n	2 drives	(n-2)X	(n-2)X
RAID 1+0/10	Striped set of mirrored sets.	4	*	needs 1 drive on each mirror set	*	*
RAID 0+1	Mirrored set of striped sets.	4	*	needs 1 working striped set	*	*

* depends on the # of mirrored/striped sets and # of drives

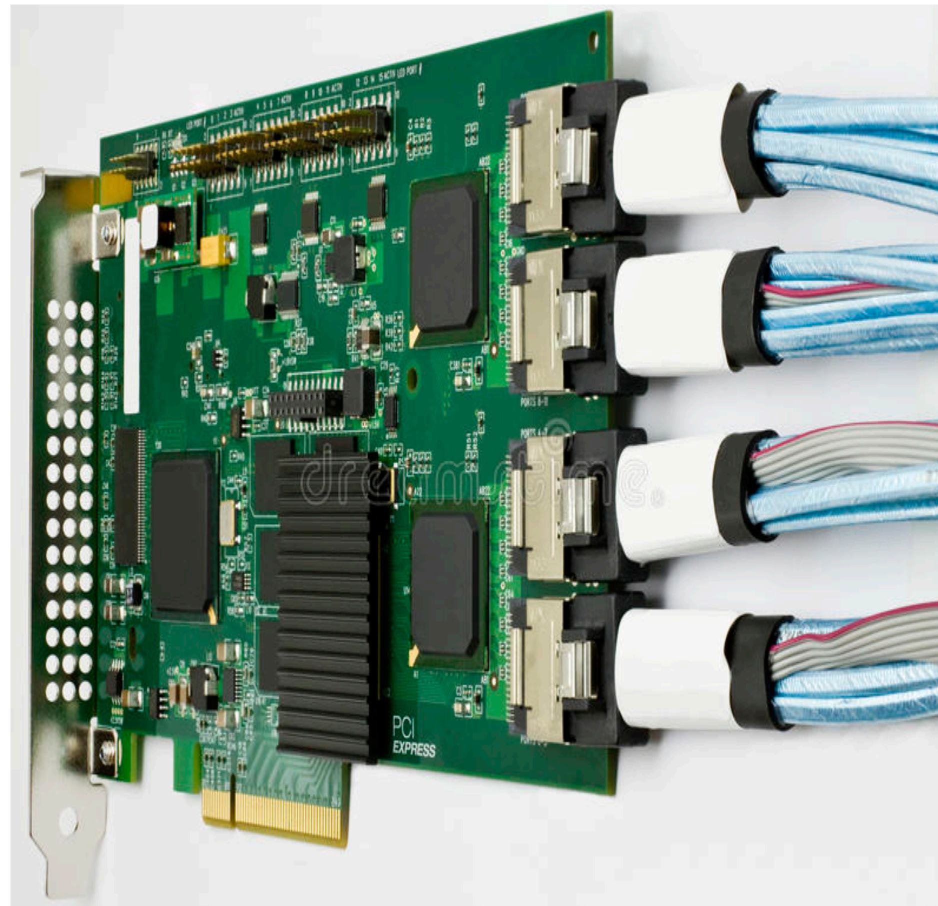
From http://en.wikipedia.org/wiki/RAID

Notes on redundancy

- Computing and updating parity negatively impact the performance. Upon drive failure, though, lost data can be reconstructed, and any subsequent read can be calculated from the distributed parity such that the drive failure is masked to the end user.
- However, a single drive failure results in reduced performance of the entire array until the failed drive has been replaced and the associated data rebuilt.
- The larger the drive, the longer the rebuild takes (up to several hours (even days) on busy systems or large disks/arrays).

Implementing RAID

- RAID is implemented both in hardware and software.
- RAID controller is the hardware part.
- Totally transparent to the users
- Configured when the system is installed
- No way to change it on the fly..



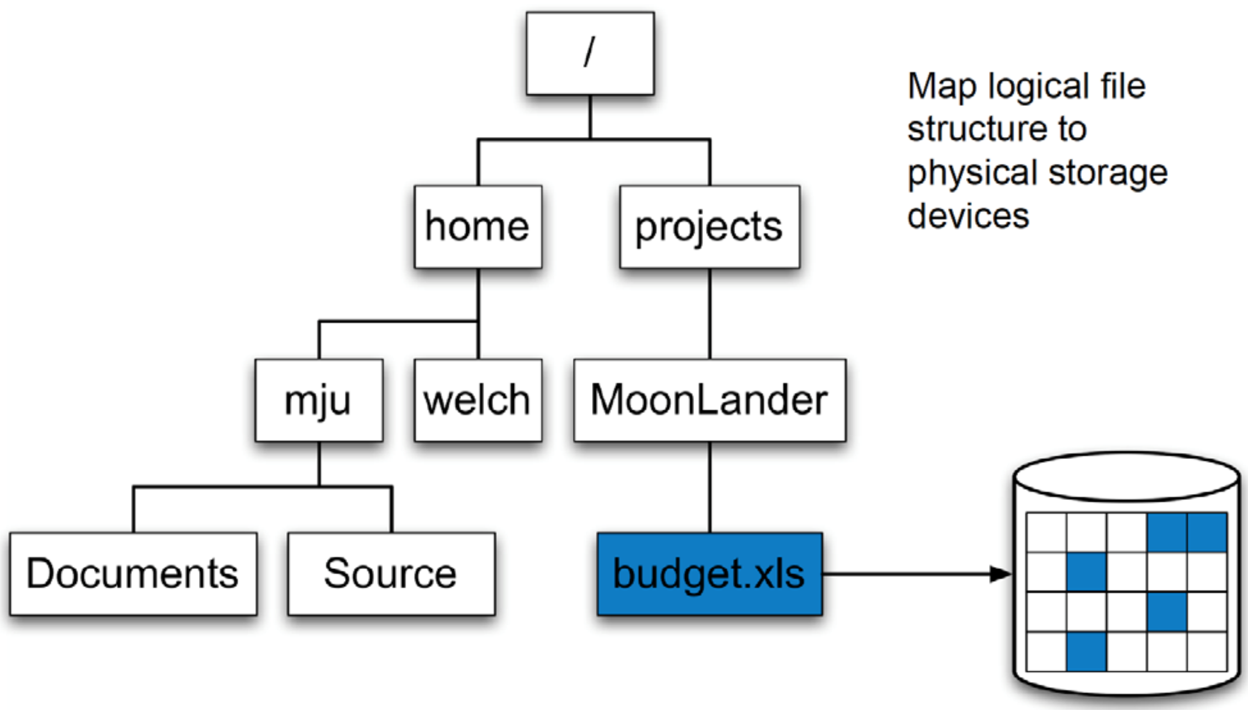
RAID on ORFEO storage

- RAID 1 on all nodes for OS reliability
- For actual storage: NONE
  - For CEPH FS redundancy managed at disk level (see later)
  - For long term storage redundancy managed at hardware/software layer within the NAS (see later)

Intro: Filesystems

Filesystem

- Provide a unique namespace (i.e. organize and maintain the file name space)
- Store your data on the medium (disk/array of disks etc)



File Systems: Basic Concepts (1/2)

- **Disk**: A permanent storage medium of a certain size.
- **Block**: The smallest unit writable by a disk or file system. Everything a file system does is composed of operations done on blocks.
- **Partition**: A subset of all the blocks on a disk.
- **Volume**: The term is used to refer to a disk or partition that has been initialized with a file system.
- **Superblock**: The area of a volume where a file system stores its critical data.

File Systems: Basic Concepts (2/2)

- **Metadata**: A general term referring to information that is about something but not directly part of it.
- **Journaling:** write data to journal, commit to file system when complete in atomic operation
  - reduces risk of corruption and inconsistency
- **Attribute**: A name and value associated with the name. The value may have a defined type (string, integer, etc.).

Filesystem: data layout

```
[root@elcid ~]# tune2fs -l /dev/sda1
 tune2fs 1.41.12 (17-May-2010)
 Filesystem volume name:   <none>
 Last mounted on:          /boot
 Filesystem UUID:          72228245-8322-4b2f-b043-317f5d9653df
 Filesystem magic number:  0xEF53
 Filesystem revision #:    1 (dynamic)
 Filesystem features:      has_journal ext_attr resize_inode
dir_index filetype
 // needs_recovery extent flex_bg sparse_super large_
 // file huge_file uninit_bg dir_nlink extra_isize
 Filesystem flags:         signed_directory_hash
 Default mount options:    user_xattr acl
 Filesystem state:         clean
 Errors behavior:          Continue
 Filesystem OS type:       Linux
 Inode count:              38400
 Block count:              153600
 Reserved block count:     7680
 Free blocks:              116833
 Free inodes:              38336
 First block:              0
 Block size:               4096
 Fragment size:            4096
 Reserved GDT blocks:      37
 Blocks per group:         32768 [...]    c
```

File System: data layout and inode

- Data structure pointed by the inode number, a unique identifier of a file in the file system
  - address of data block on the storage media description of the file (POSIX)
  - Size of the file
  - Storage device ID
  - User ID of the file's owner.
  - Group ID of the file.
  - File type
  - File access right
  - Inode last modification time (ctime)
  - File content last modification time (mtime),
  - Last access time (atime).
  - Count of hard links pointed to the inode.
  - Pointers to the disk blocks that store the file's contents

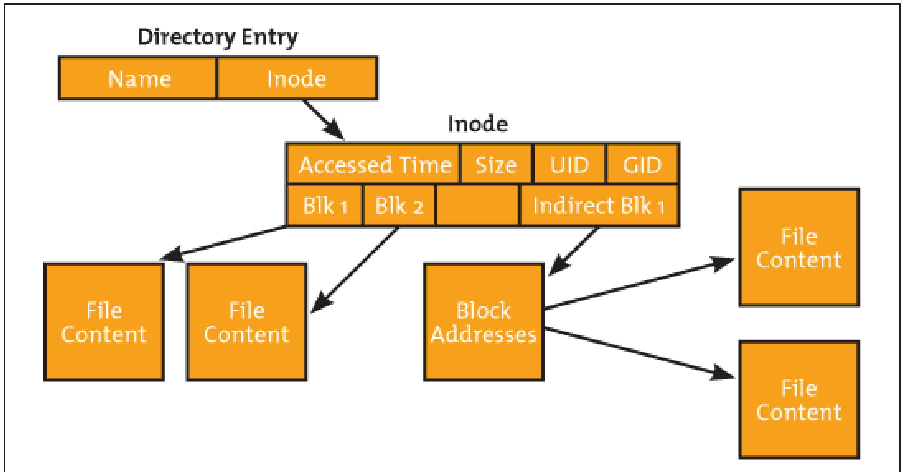


FIGURE 1 RELATIONSHIP BETWEEN THE DIRECTORY ENTRY, AN INODE, AND BLOCKS OF AN ALLOCATED FILE

Useful command to interact with FS

- ls -i
- stat filename
- df -i

```
[cozzini@login ~]$ df -ih
Filesystem                                               Inodes IUsed IFree IUse% Mounted on
10.128.6.211:6789,10.128.6.212:6789,10.128.6.213   969K     -     -     - /fast
10.128.6.211:6789,10.128.6.213:6789,10.128.6.212/   48M     -     -     - /large
10.128.4.201:/opt/area                              191M  797K  190M    1% /opt/area
10.128.2.231:/illumina_run                          4.6G  1.9M  4.6G    1%
/illumina_run
10.128.2.231:/storage                                3.7G  462K  3.7G    1% /storage
```

Data and metadata

- Meta-data : Data to describe data attribute (and extended attribute)
  - size, owner, creation date
- Meta-data are the bottleneck of scalability
  - How many times do you type `ls` in a day?
    How many times to you write a file?
- `ls` means a scanning of all the files in the directory !

Posix interface

- API to access data and metadata (1988)
- POSIX interface is a useful, ubiquitous interface for building basic I/O tools.
- Standard I/O interface across many platforms.
- open, read/write, close functions in C/C++/Fortran
- It allows buffered file I/O (streams) within (c/sdtio)

Posix interface (2)

- Posix assumes atomicity and ubiquity
  - Changes are visible immediately to all clients
- Problem for parallel accesses:
  - POSIX requires a strict consistency to sequential order : lock
    - (Create a directory is an atomic operation with immediate global view)
  - No support for non-continuous I/O
  - No hint / prefetching

MPI-IO can be useful here. (see later..)

Local FS: some examples

- Linux
  - Ext2
  - Ext3
  - ext4
  - Raiserfs
  - Jfs
  - Xfs…

I/O FS 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 16  2020 scratch -> /scratch/area/cozzini
```

I/O FS 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 FS on ORFEO:

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

```
[root@login ~]# df -h | grep 231
10.128.6.231:/illumina_run       128T  109T   20T  85% /illumina_run
10.128.6.231:/lage_archive       128T   94T   34T  74% /lage_archive
10.128.6.231:/onp_run_1          117T   56T   61T  48% /onp_run
10.128.6.231:/burlo_lon          91T  8.6T   83T  10% /burlo_long_term_storage
10.128.6.231:/analisi_da_cons    100T   56T   45T  56% /analisi_da_consegnare
10.128.6.231:/lala_storage       4.6T  2.4T  2.3T  52% /lala_storage
10.128.6.231:/opt/area           477G  210G  267G  45% /opt/area
```

Measure (raw) performance on FS

- dd command..

```
$dd if=/dev/zero  of=/dev/null count=1
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.000242478 s, 2.1 MB/s
$dd if=/dev/zero  of=~/big-write count=1M
1048576+0 records in
1048576+0 records out
536870912 bytes (537 MB) copied, 3.43889 s, 156 MB/s
```

- Questions:
  - Why such a difference between the two runs?
  - Why copying unit of 512B ?

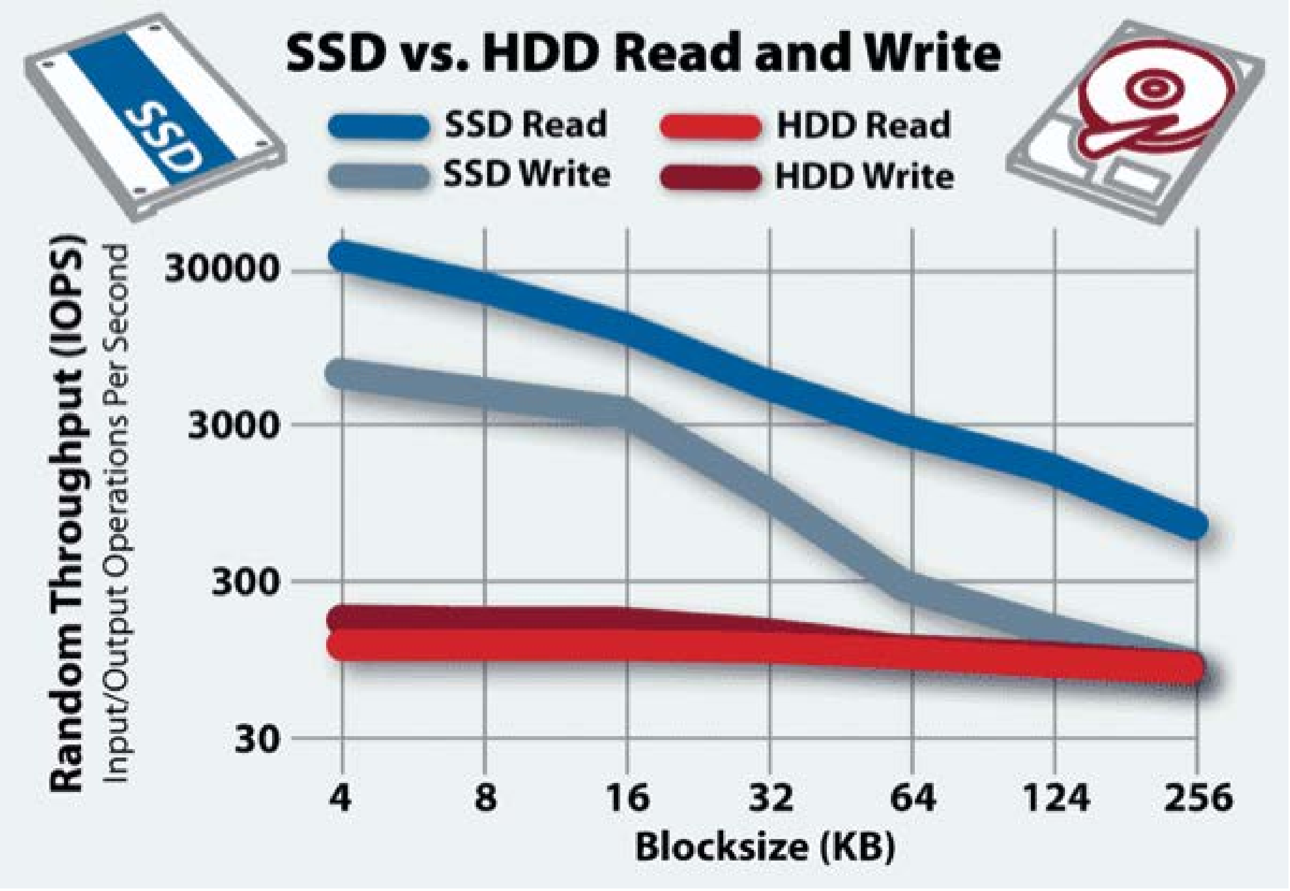
Blocksize on FS

- 512 byte is a typical block-size of the disk:
- It cannot read less than 512 bytes, if you want to read less, read 512 bytes and discard the rest.
- File System block-size can be different

```
[exact@login ~]$ stat -f .
  File: "."
    ID: 9d0420af3cbc070e Namelen: 255     Type: ext2/ext3
Block size: 4096       Fundamental block size: 4096
Blocks: Total: 372561982  Free: 51012529   Available: 32646449
Inodes: Total: 94633984   Free: 90641935
```

Blocksize effect in the Random access

- The performance DISK is not a single number



Proposed exercise

- Identify your FileSystem and its properties
- Measure/Estimate the rough performance of your hard-drive
- Compare it with the ramfs on your linux box and on your cluster system

```
cozzini@login ~]$ df
Filesystem                       1K-blocks        Used  Available Use% 
Mounted on
/dev/mapper/SysVG-Root            51474912    33126208   15710880  68% /
devtmpfs                          16358128           0   16358128   0% 
/dev
tmpfs                             16371480      501024   15870456   4% 
/dev/shm
```

Lecture 03: Parallel I/O in HPC

A couple of citations

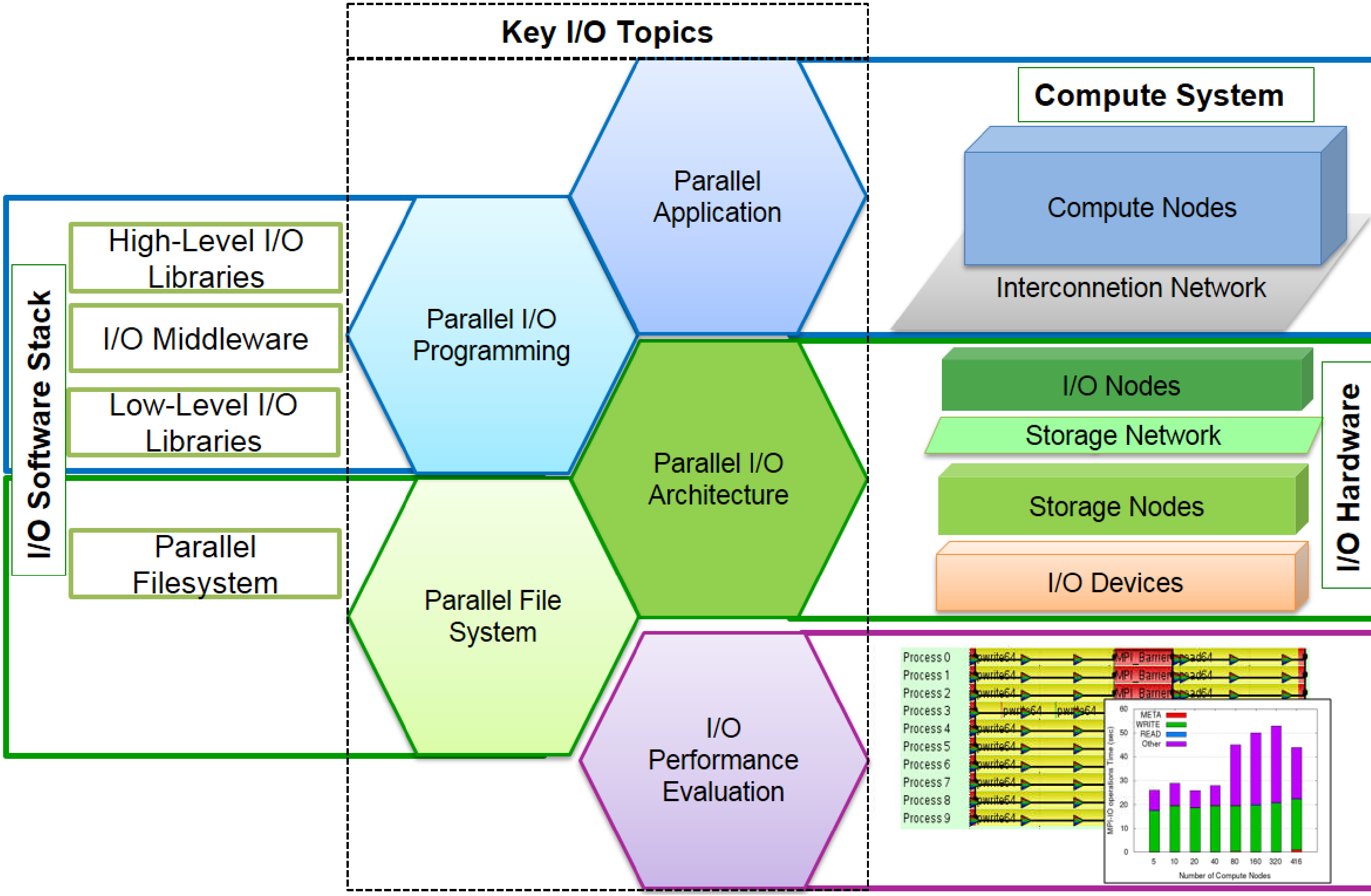
“Very few large scale applications of practical importance are NOT data intensive.”

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

Parallel I/O in HPC



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
- (Implementing ”out-of-core” techniques for algorithms that process more data than can fit in system memory)

Checkpoint/restart

- For long-running applications, the cautious user checkpoints
- Application-level checkpoint involves the application saving its own state
  - –Portable!
- A canonical representation is preferred
  - –Independent of number of processes
- Restarting is then possible
- Canonical representation aids restarting with a different number of processes
- Also eases data analysis (when using same output)

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

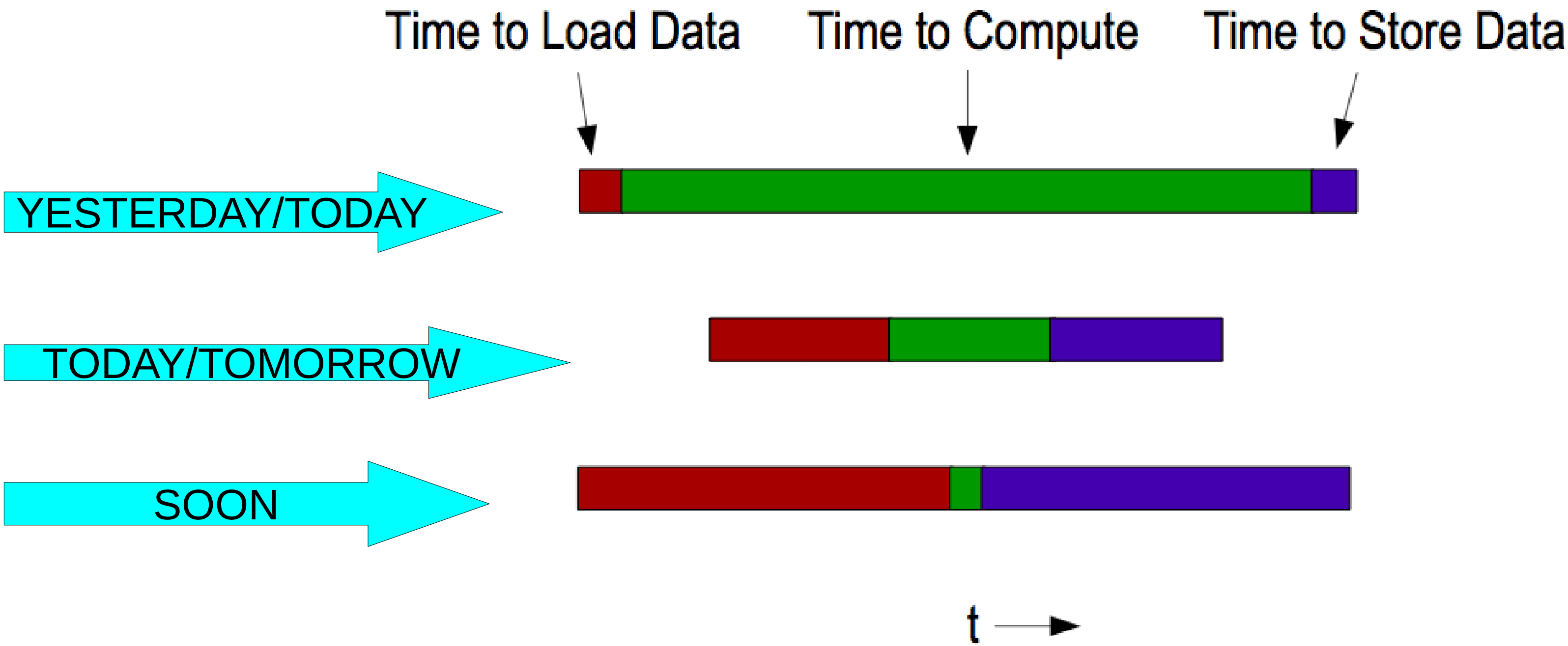
Preprocessing/Post-processing phases..

- 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

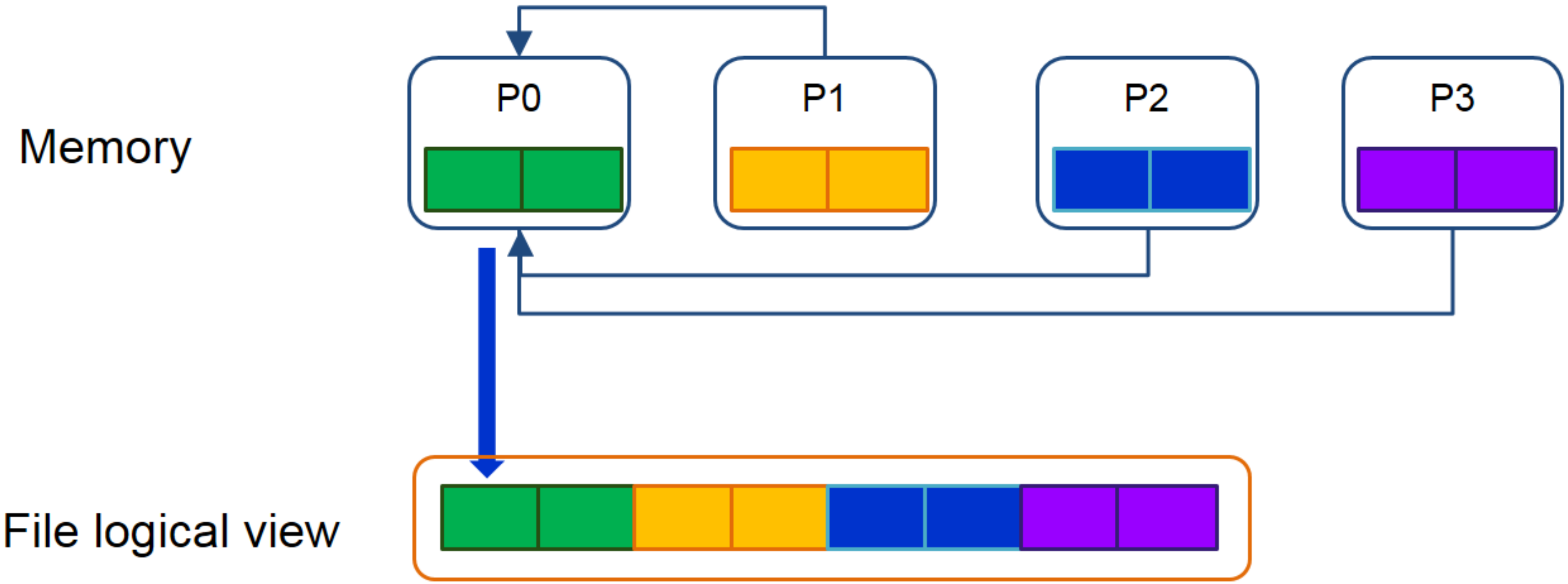
Application dataset complexity vs I/O

- 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

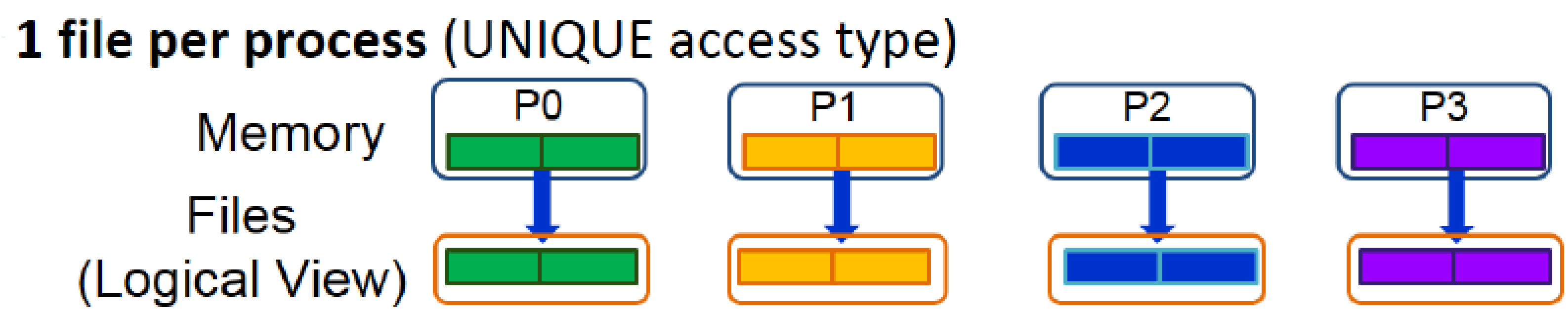
- One process performs I/O.
  - Data Aggregation or Duplication
  - Limited by single I/O process.
- Simple solution, easy to manage, but Pattern does not scale.
  - Time increases linearly with amount of data.
  - Time increases with number of processes.



Parallel I/O: File-per-Process

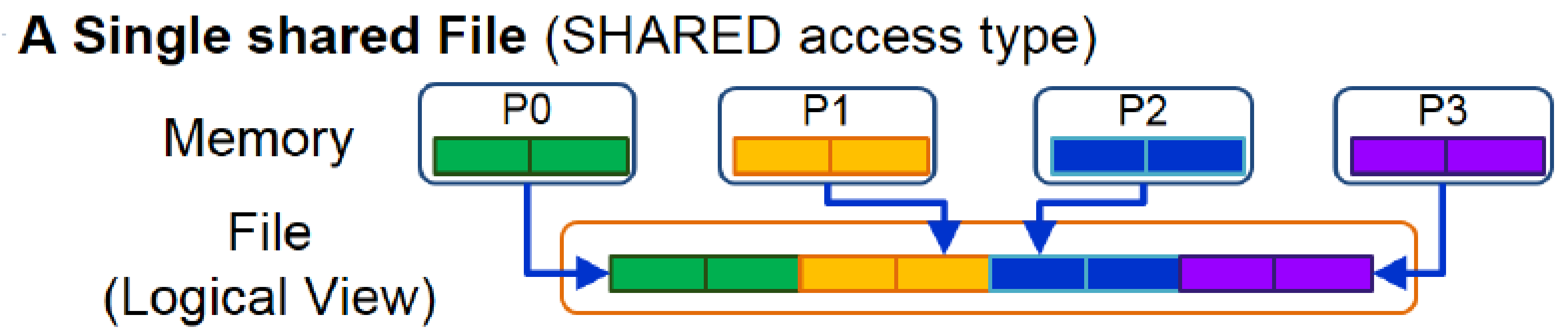
All processes perform I/O to individual files.

- 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 ?



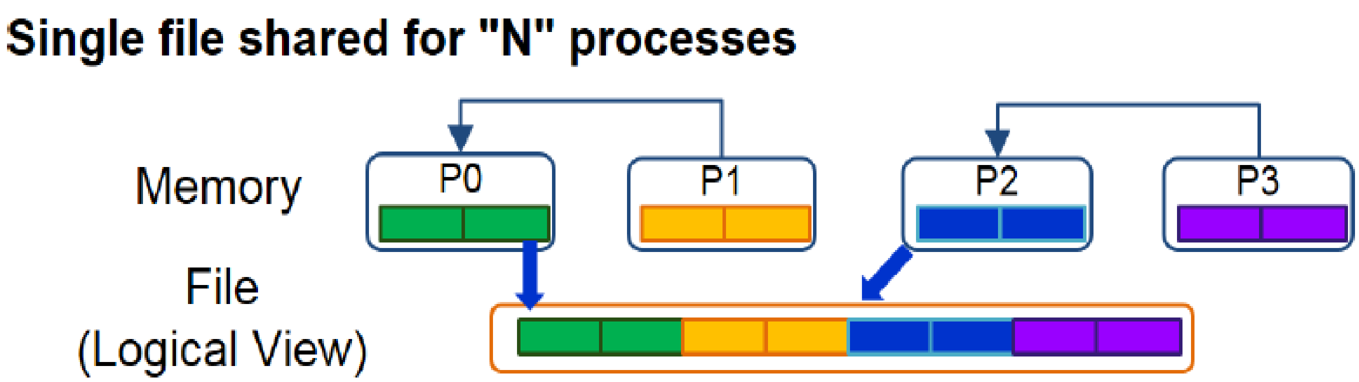
Parallel I/O

- 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..



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.



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

I/O access patterns

I/O Operation

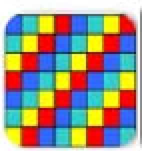
- Read only
- Write only

Request size

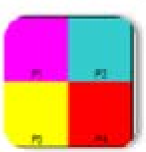
- Fixed / Variable
- Small / Medium / Large

Spatial Patterns

Contiguous



Non-contiguous: Strided
Fixed strided / 2D-strided / Negative strided /Random strided/ kD-strided



Combination of contiguous and non-contiguous patterns

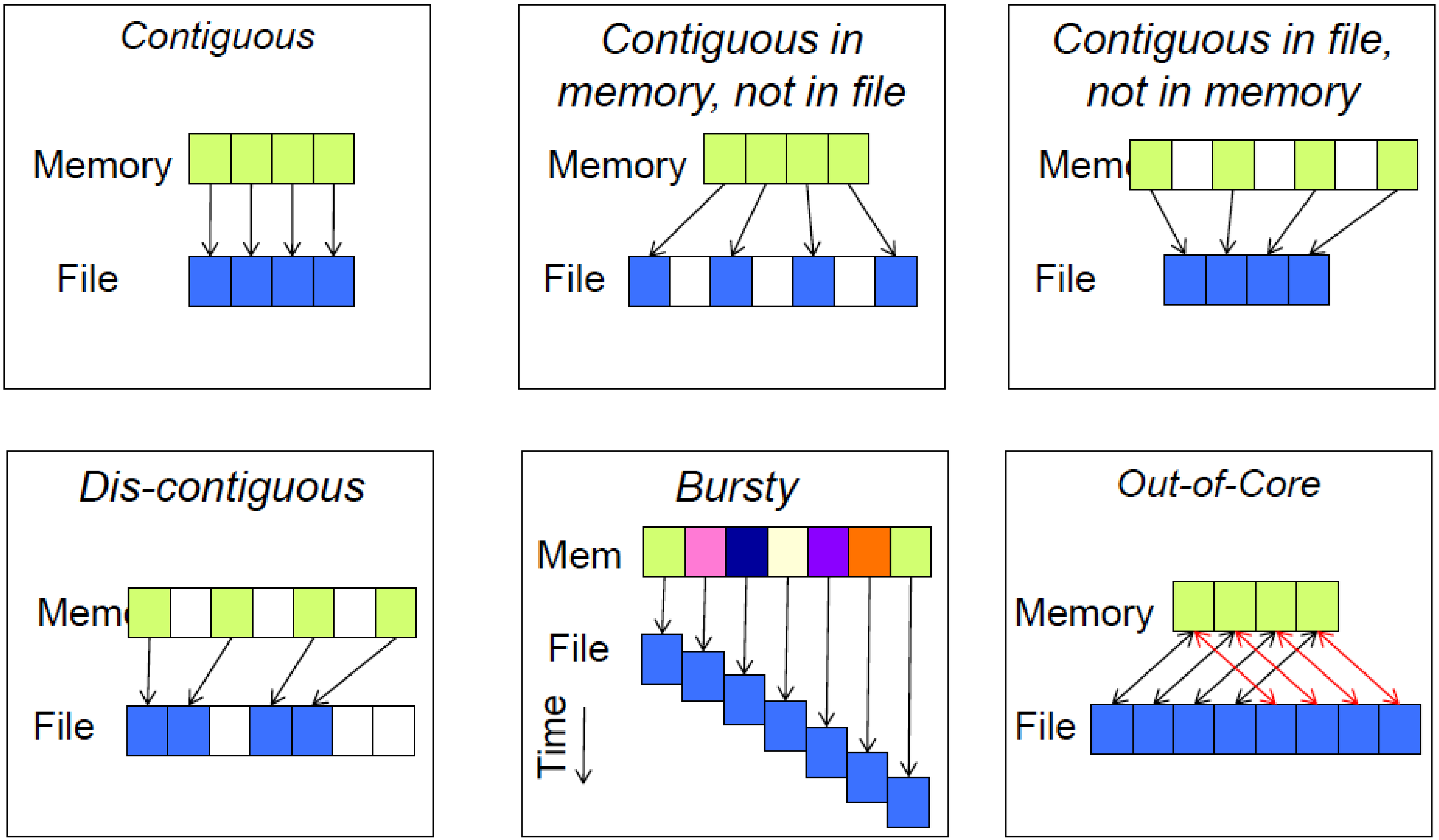
Temporal intervals

- Fixed
- Random

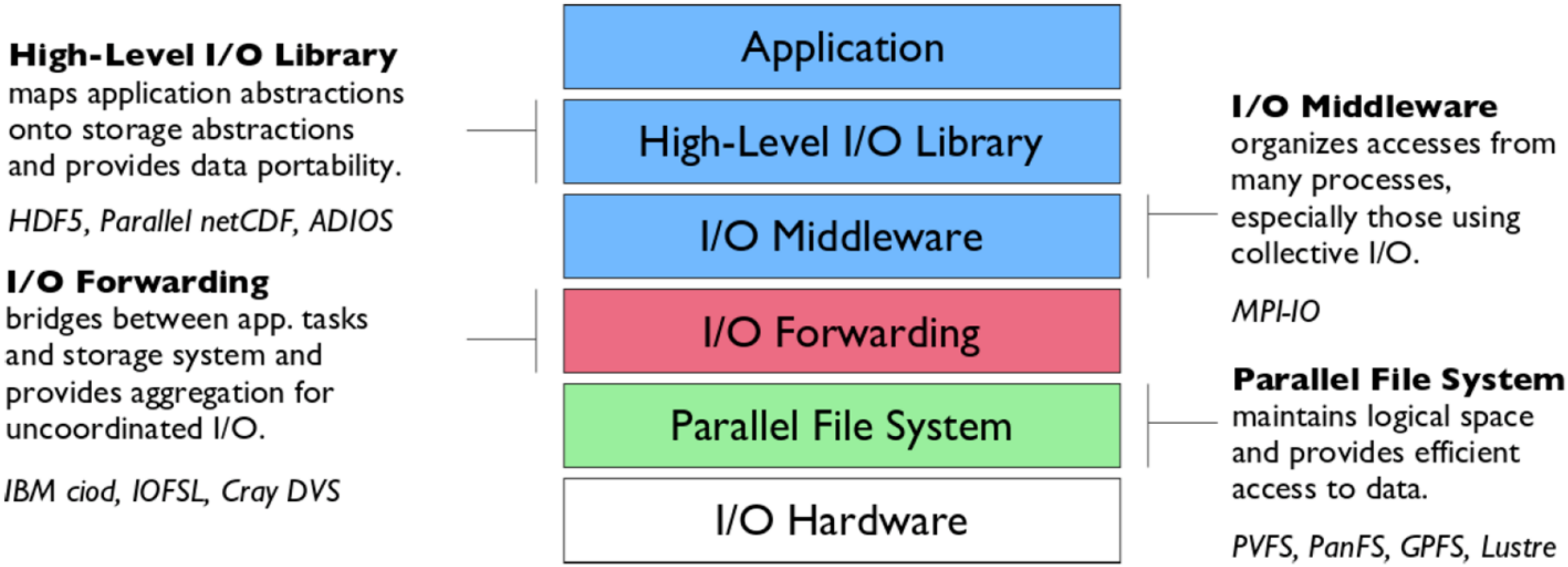
Repetition

- Single Occurrence
- Multiple Ocurrence

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.

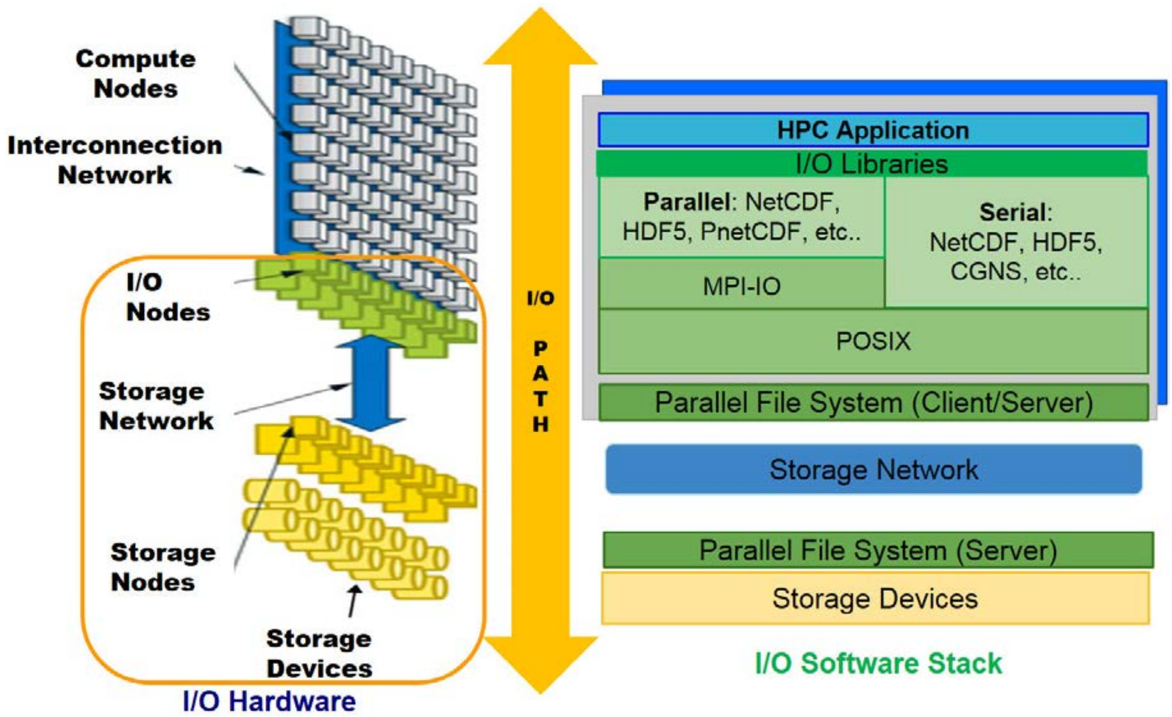
Parallel I/O using MPI ?

- 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….

Elements of a PFS

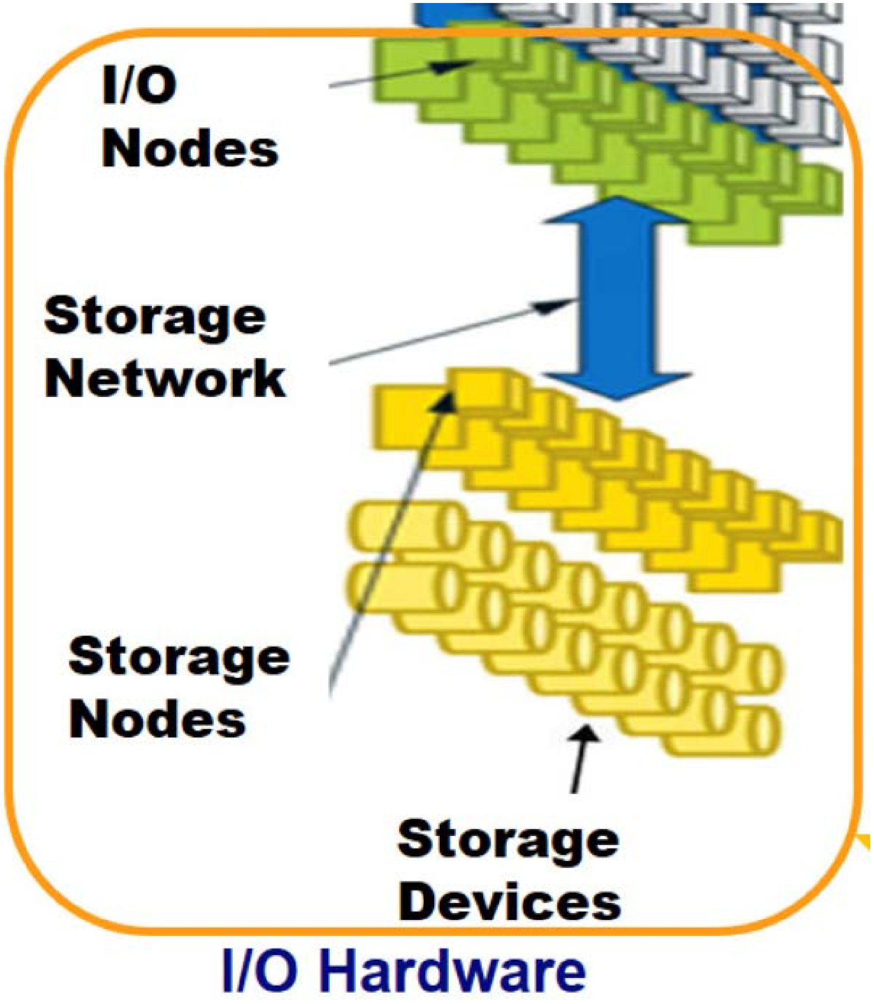
- A parallel solution usually is made of
  - several Storage Servers that hold the actual filesystem data
  - one or more Metadata Servers that help clients to identify/manage data stored in the file system
  - a redundancy layer that replicates in some way information in the storage cluster, so that 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:



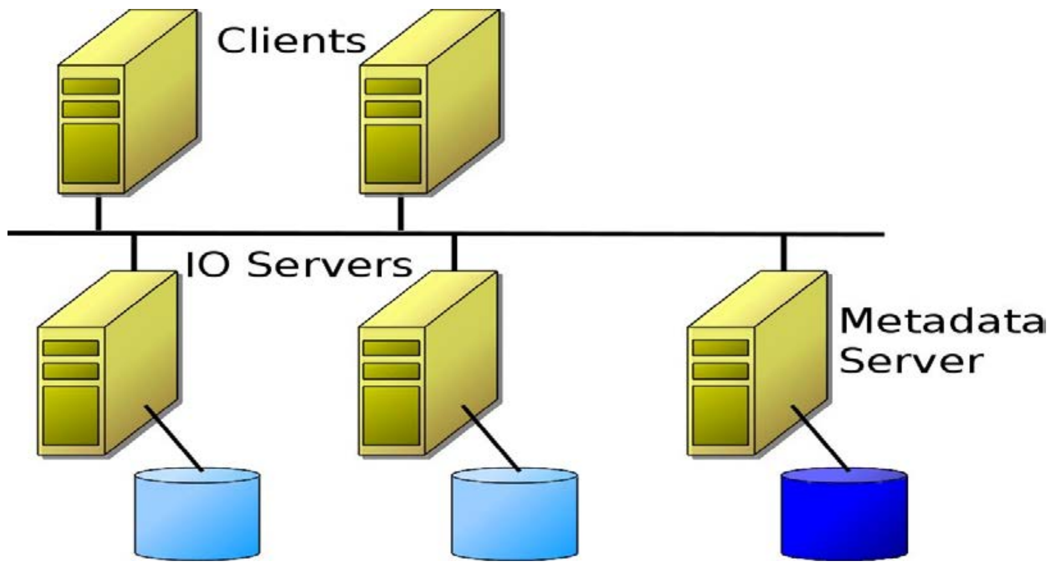
Picture from: http://www.prace-ri.eu/best-practice-guide-parallel-i-o/#id-1.3.5

Parallel File System: I/O hardware



Parallel File System: components

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



Hardware to build a PFS:

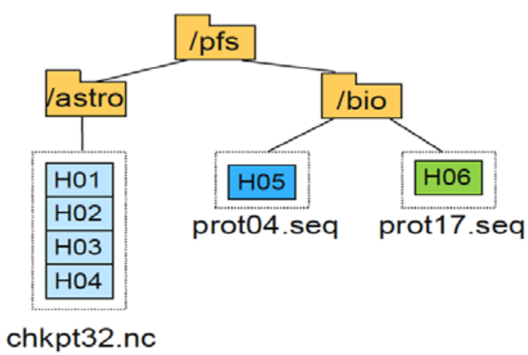
- Nodes, Disks, controllers, and interconnects
- Hardware defines the peak performance of the I/O system:
  - raw bandwidth
  - Minimum latency
- At the hardware level, data is accessed at the granularity of blocks, either physical disk blocks or logical blocks spread across multiple physical devices 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

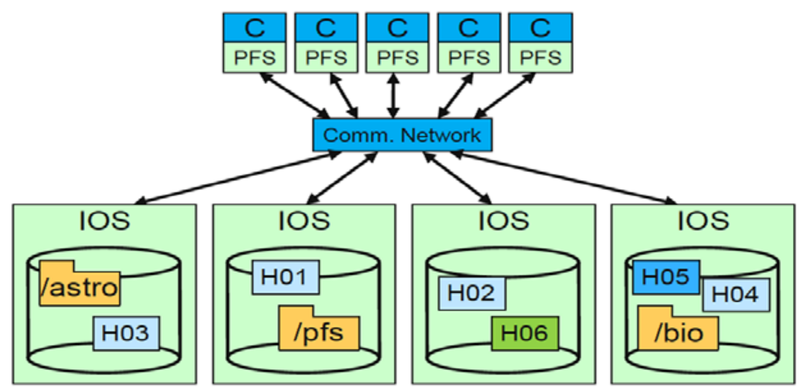
An important disclaimer..

- 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. i.e. no Posix sometime/somewhere

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

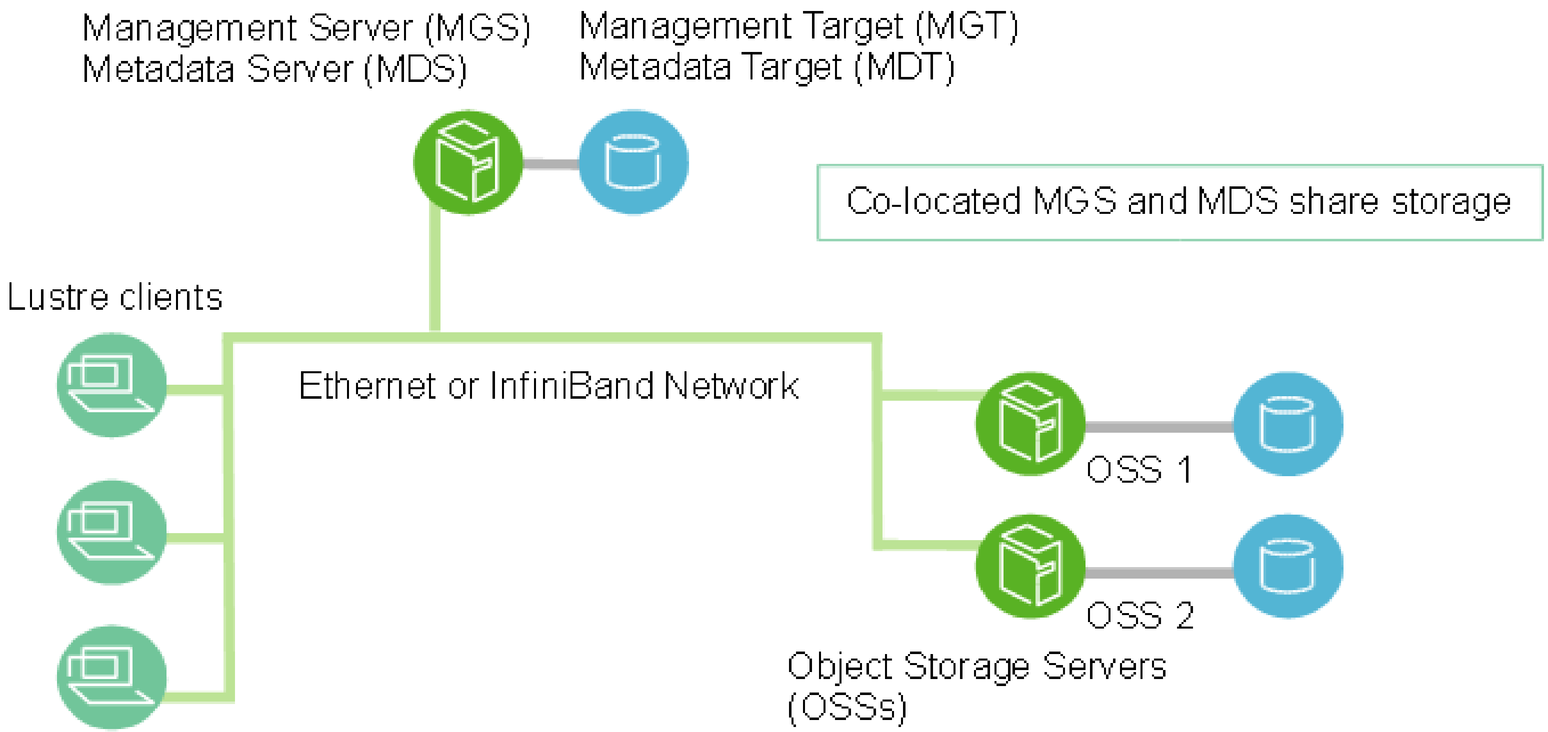




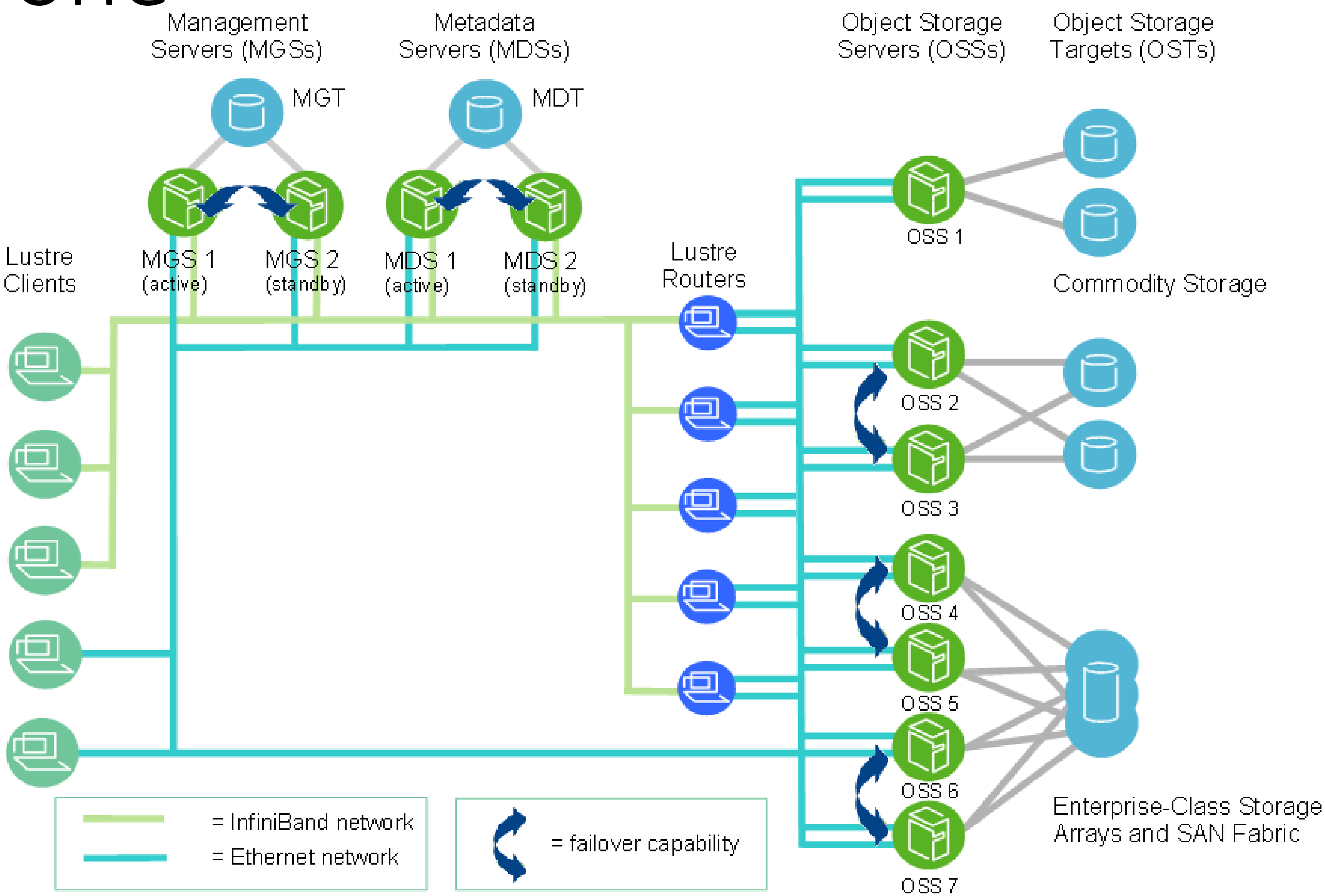
What is available on the market ?

- 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)

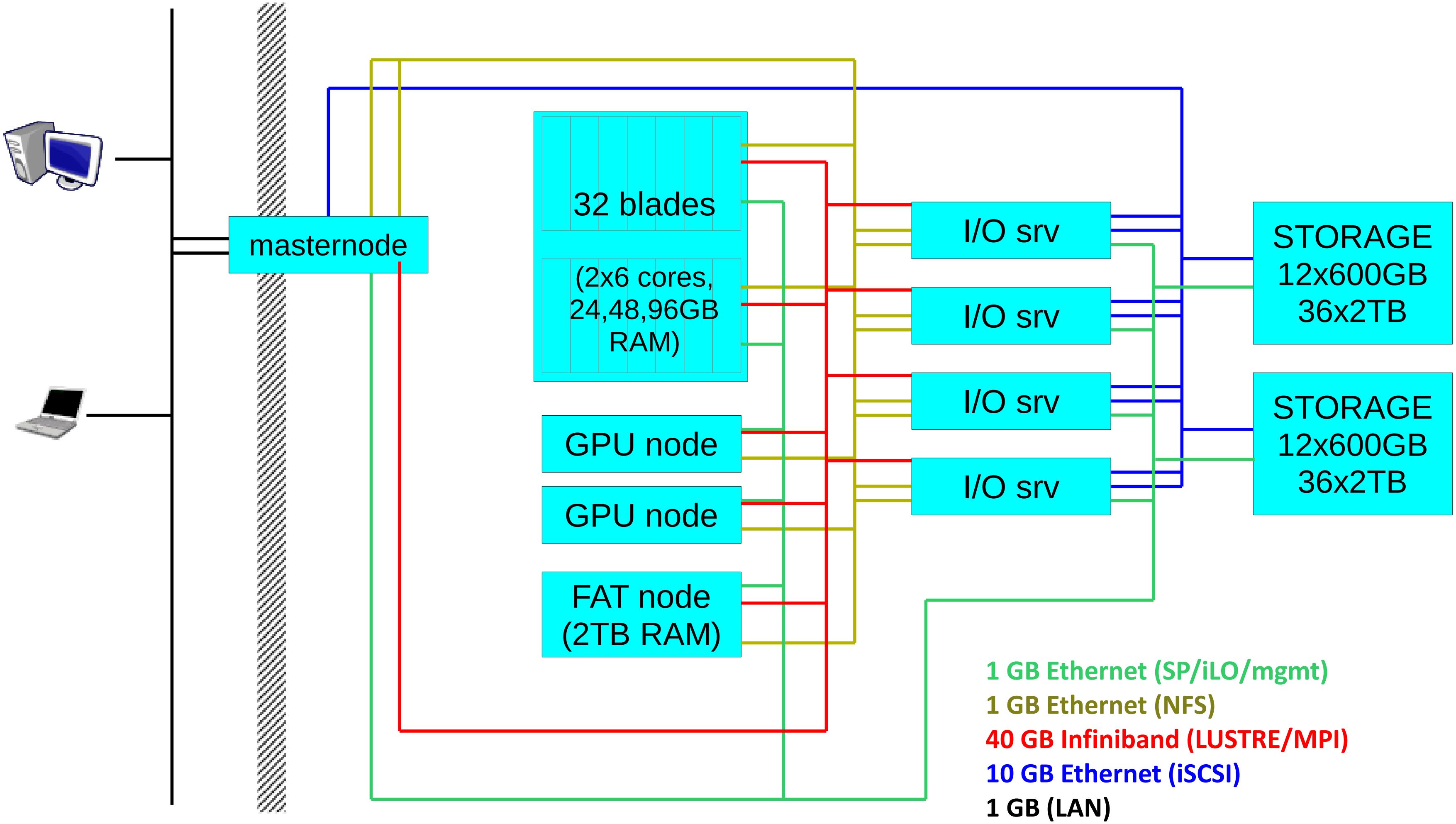
Lustre in two pictures: simple one



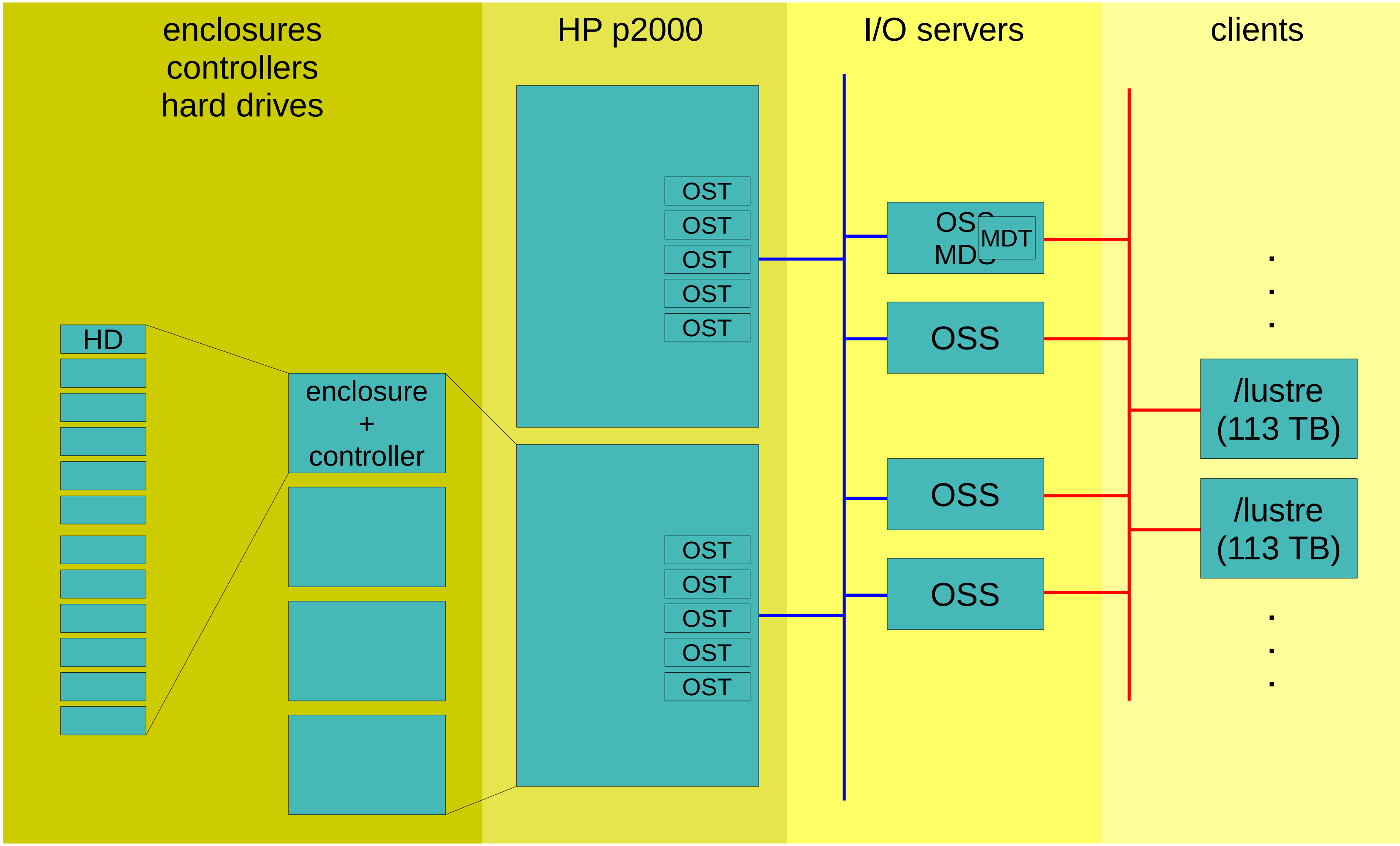
Lustre in two pictures: complex one



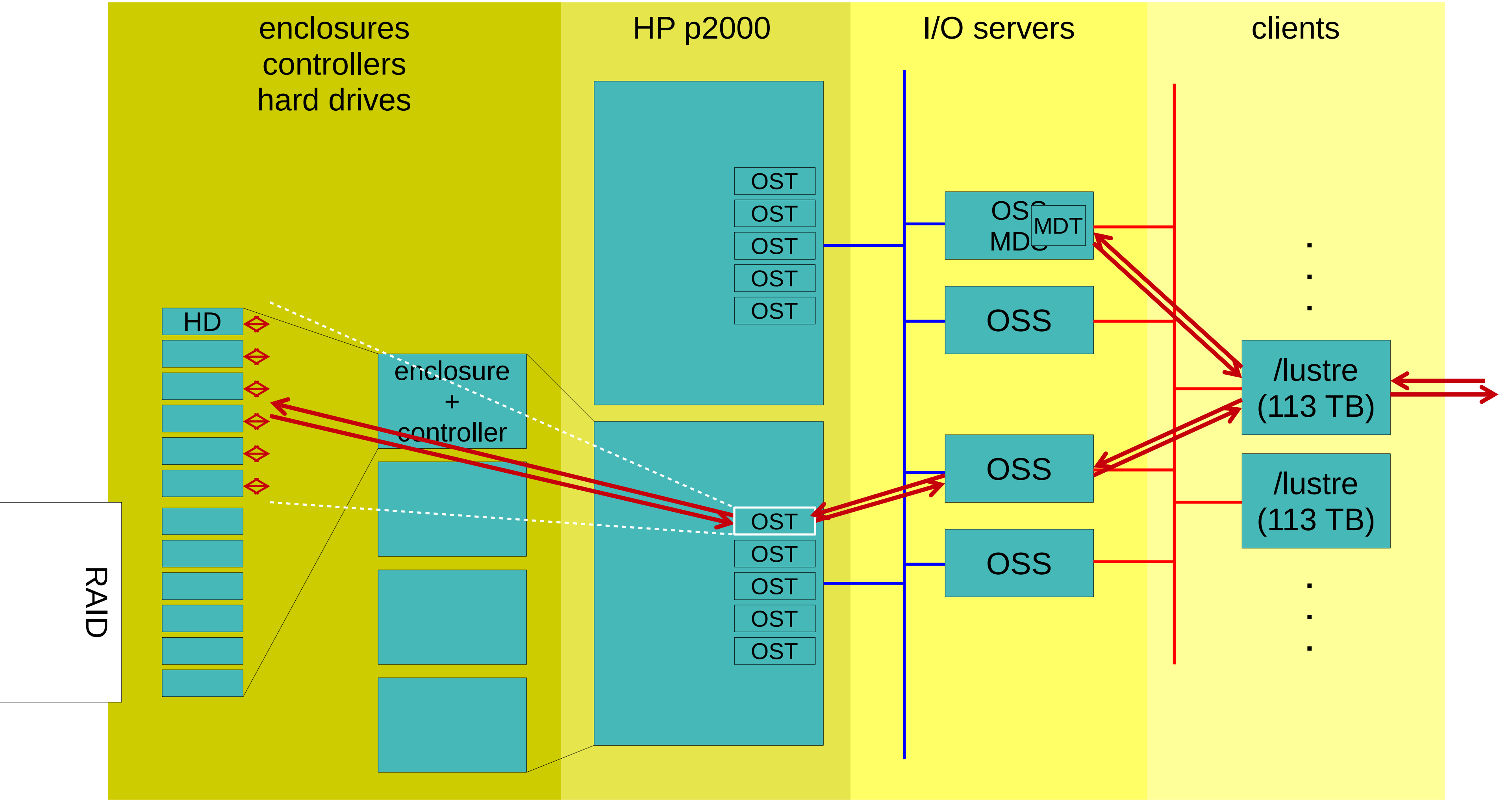
HPC infrastructure @ CRIBI



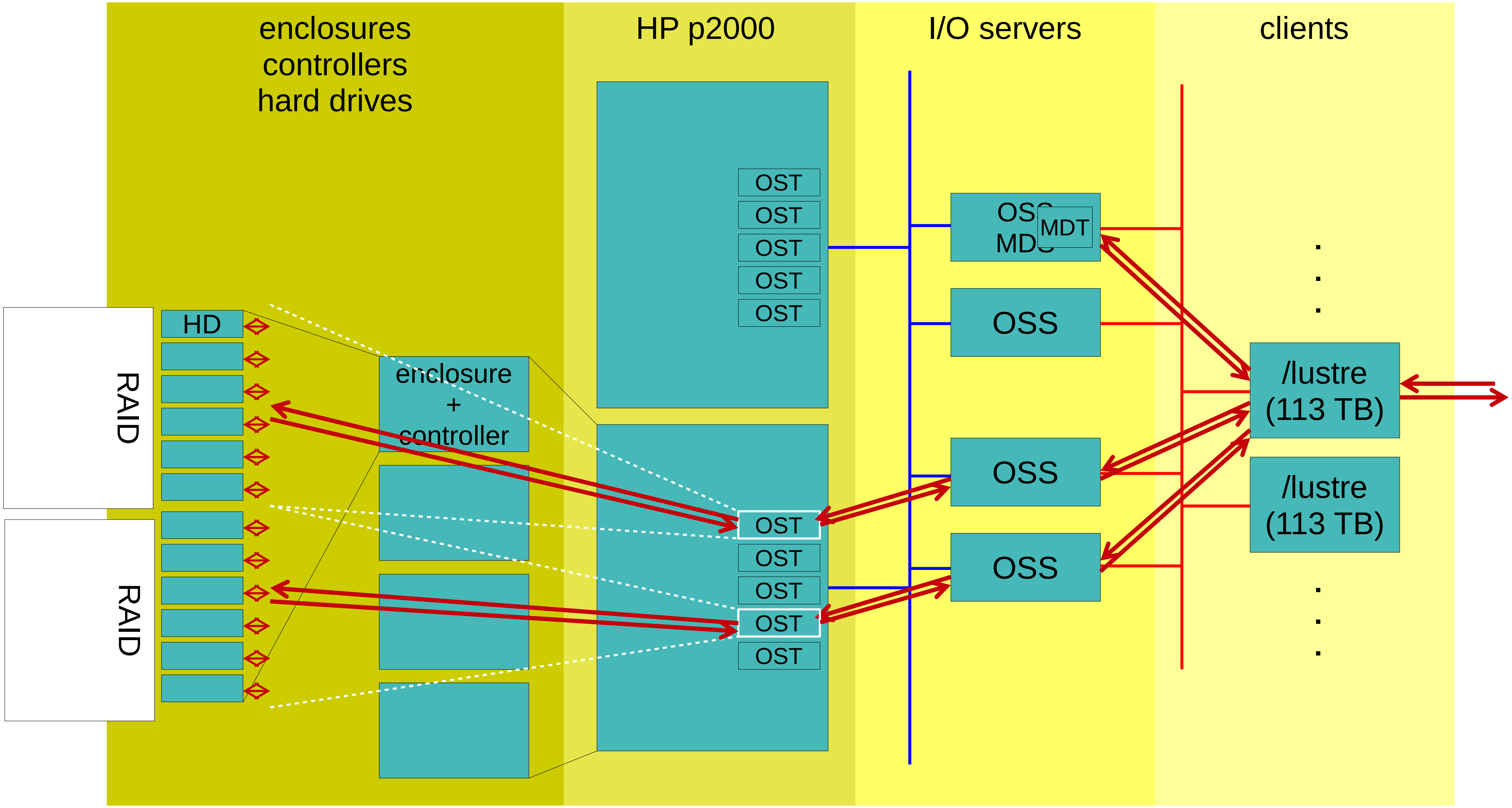
LUSTRE@CRIBI as storage solution



accessing LUSTRE filesystem



why “parallel” filesystem?

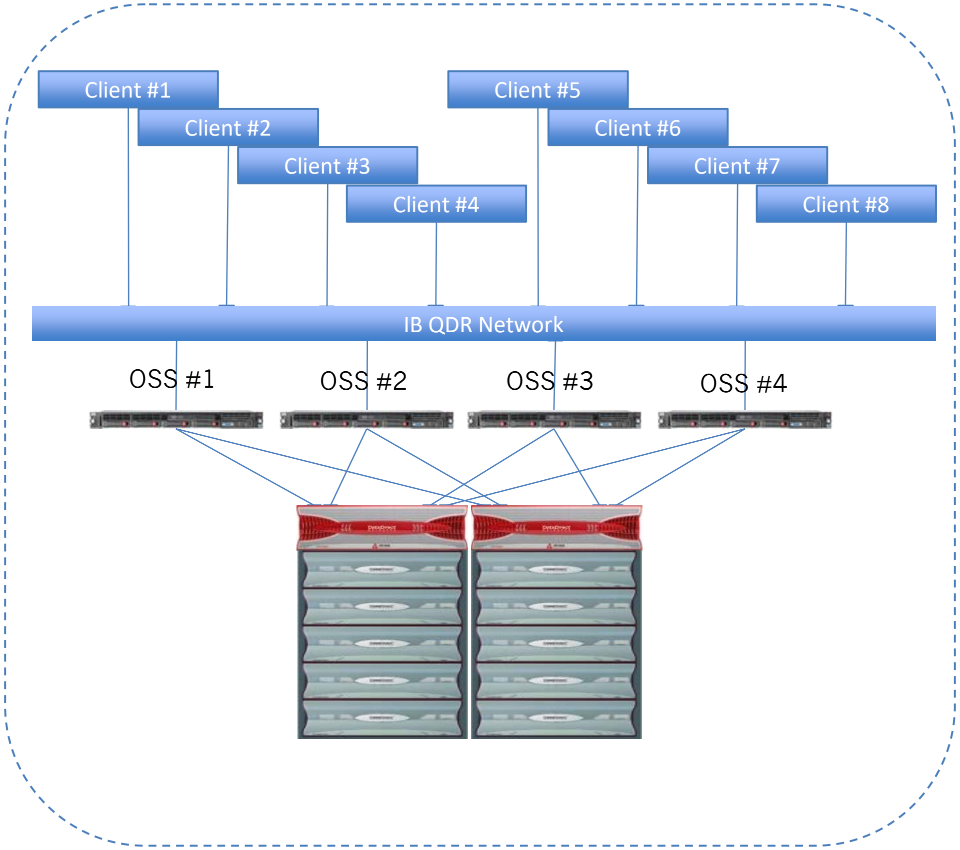


Expected performance

- 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

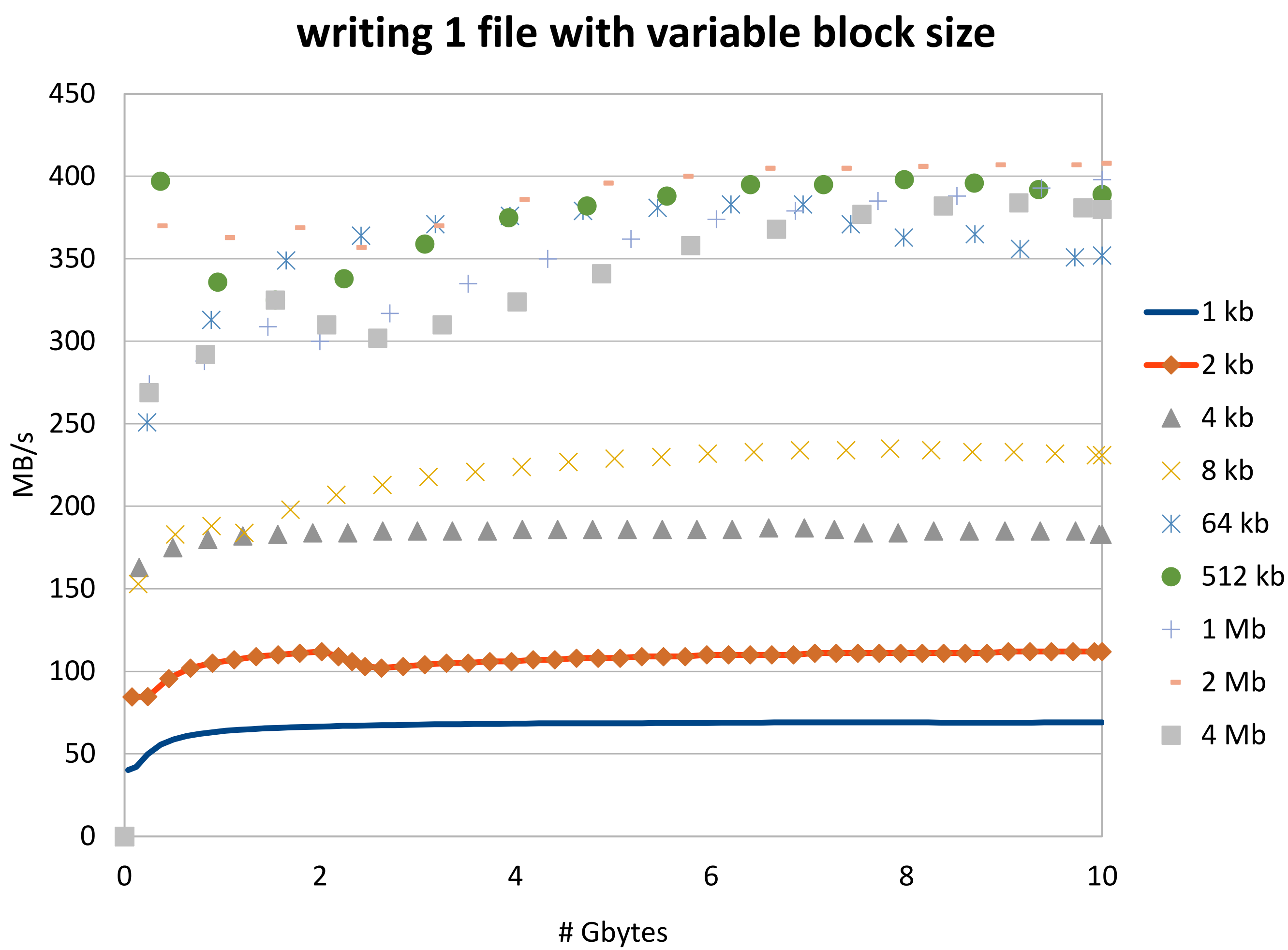


- ➢ sequential write/read by iozone
- ➢ 1 ~ 8 clients, 1 ~ 4 proc/client
- ➢ 32 GB files writing
- ➢ 64 GB files reading

- **~ 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...



To be continued