



DATA SCIENCE &  
SCIENTIFIC COMPUTING

# Storage for HPC systems

Stefano Cozzini  
AreaSciencePark  
28.11.2022

# Agenda of these lectures

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

# 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

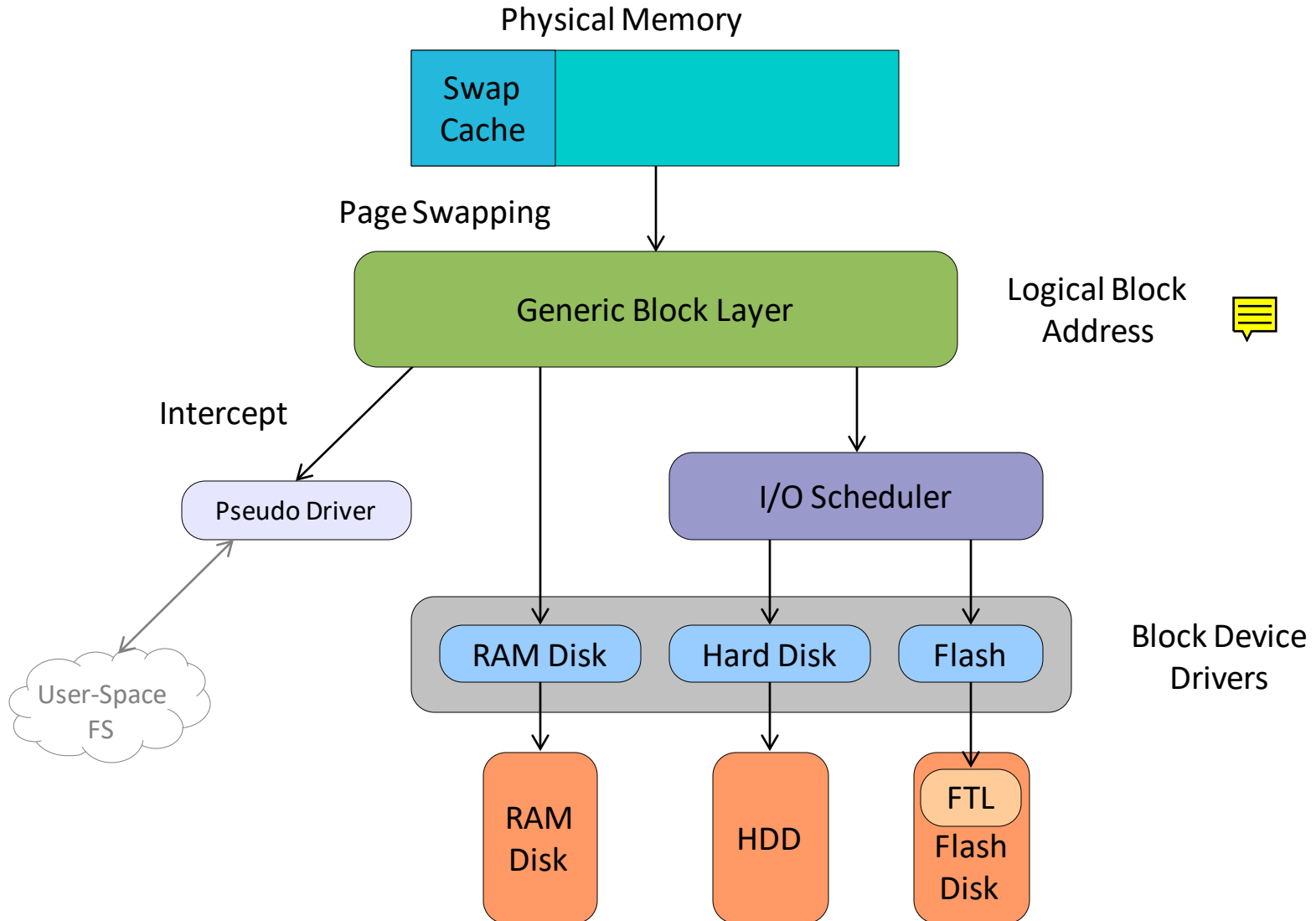


# Storage Hierarchy

- 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

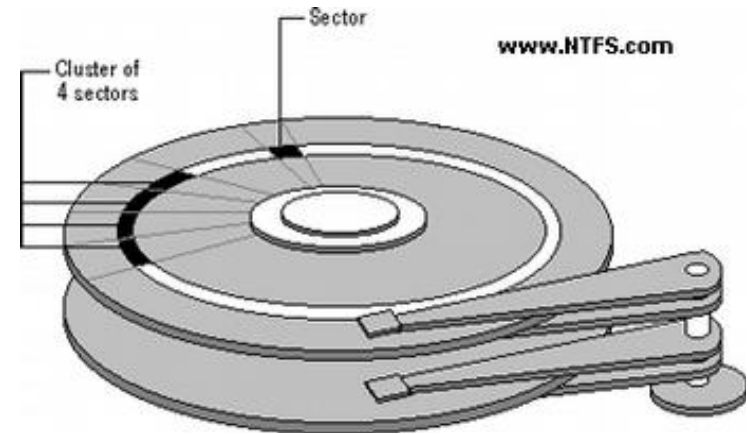
- Unix-like OS environments very frequently create (small) temporary files in /tmp, etc.
  - faster access and less wear with RAM disk
  - Linux provides “dynamic RAM disk” (tmpfs)
  - only existing files consume RAM
  - automatically cleared on reboot (-> volatile)

```
[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

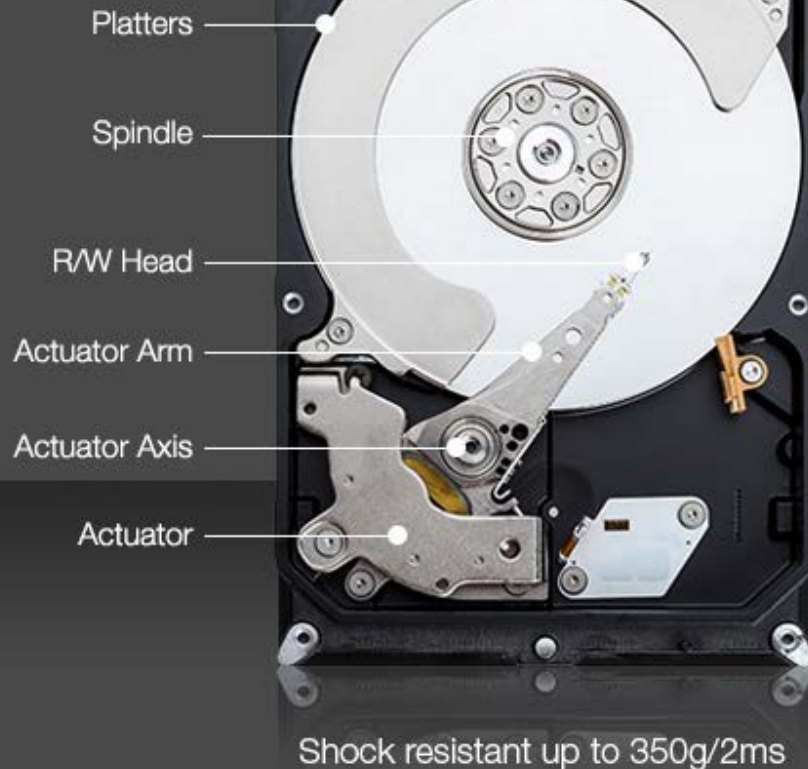
- Two main technologies today:
  - Serial Advanced Technology Attachment (SATA)
    - less expensive, and it's better suited for desktop file storage.
    - Up to 6 Gbit/sec
  - Serial Attached SCSI (SAS)
    - more expensive, and it's better suited for use in servers or in processing-heavy computer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

# Solid State Drive: SDD

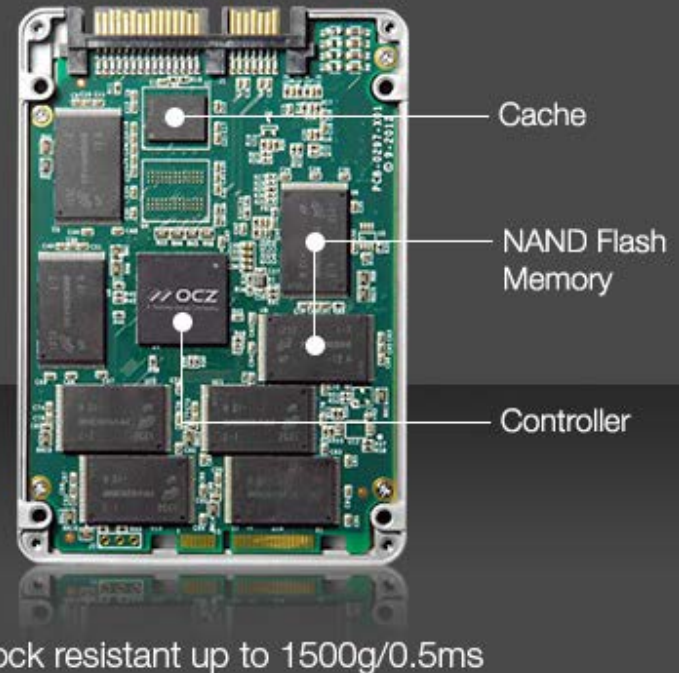
- pros:
  - lower access time and latency
  - no moving parts (silent, less susceptible to physical shock, low power consumption and heat production)
  - available over SATA, SAS, PCIe
- cons:
  - expensive, low capacity; usage limited to special purposes  only (hardly used for big data-servers)
  - 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"



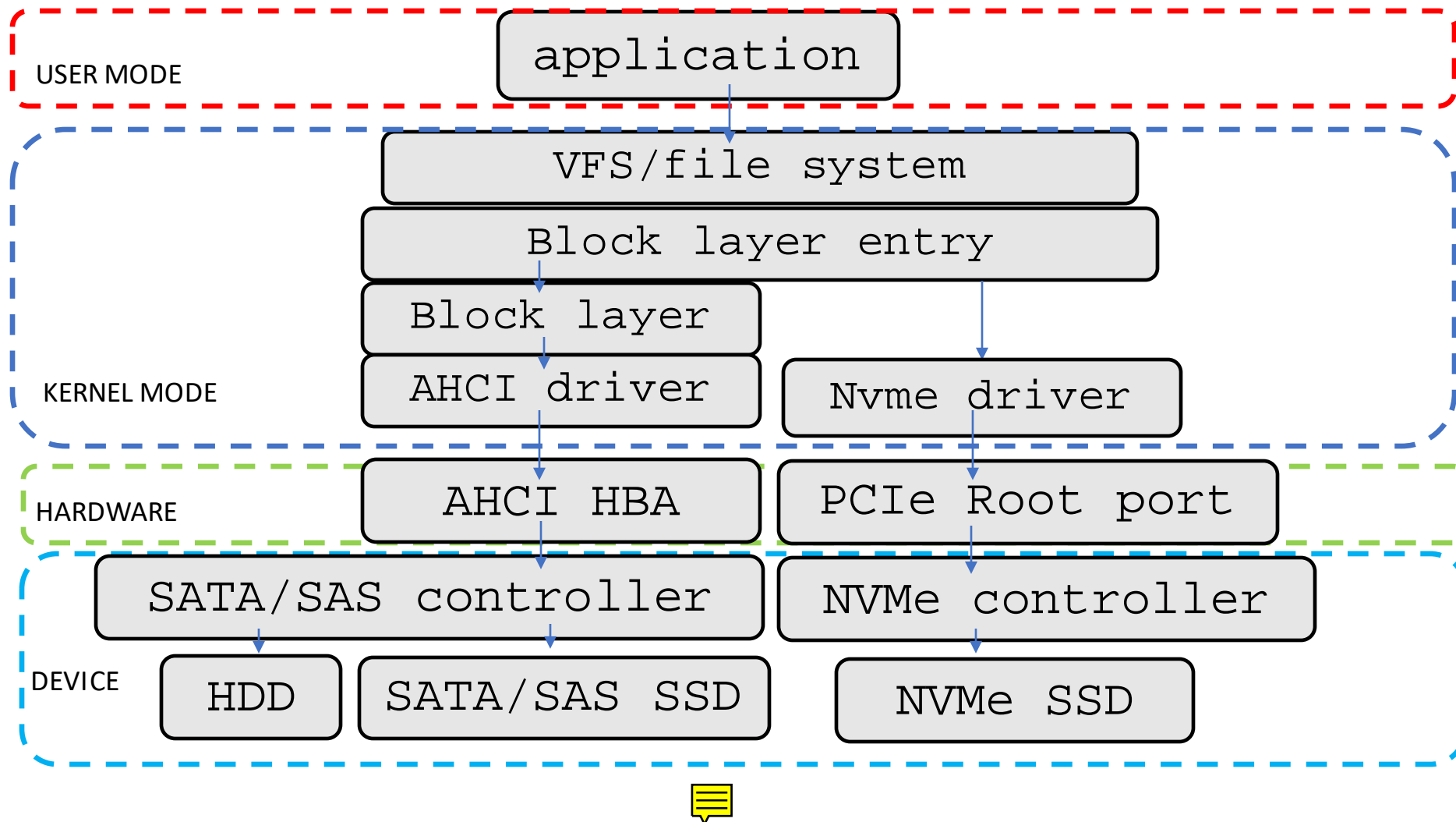
**SSD**  
2.5"



# 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 as today...

	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 + 42x 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



# ORFEO storage: hardware as Christmas

	FAST storage (NVMe)	FAST storage (SSD)	Standard storage (HDD)	Long term preservation
# of server	4		<del>6</del> 8	1
RAM	6 x 16GB		6 x 16GB	6 x 16GB
Disk per node	<del>2</del> 4 x 1.6TB NVMe PCIe card	20 x 3.84TB	<del>15</del> 18 x 12TB + 18x16TB + (on the 2 new server)	84 x 12TB + <del>84</del> 42x 12TB+ 84x12TB
Storage provider	CEPH parallel FS	CEPH parallel FS	CEPH parallel FS	Network FS (NFS)
RAW storage	<del>12</del> 24TB	320 TB	<del>1080</del> 1872 TB	<del>1,512</del> 3024 TB



# The ORFEO basic brick: NVME


- Device Type
  - SSD –NVME no hot swap
  - Samsung PM1725b HHL
- 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](http://image-us.samsung.com/SamsungUS/PIM/Samsung_1725b_Product.pdf)

# The ORFEO basic brick: SSD 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



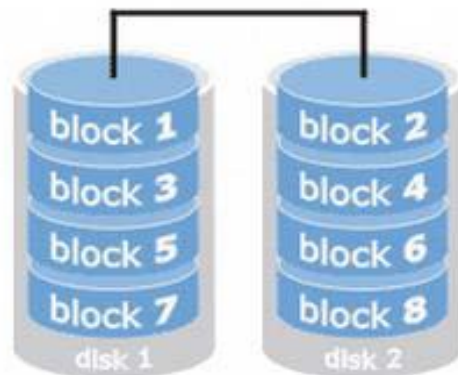
# The ORFEO basic brick: HDD drive

- 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](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..

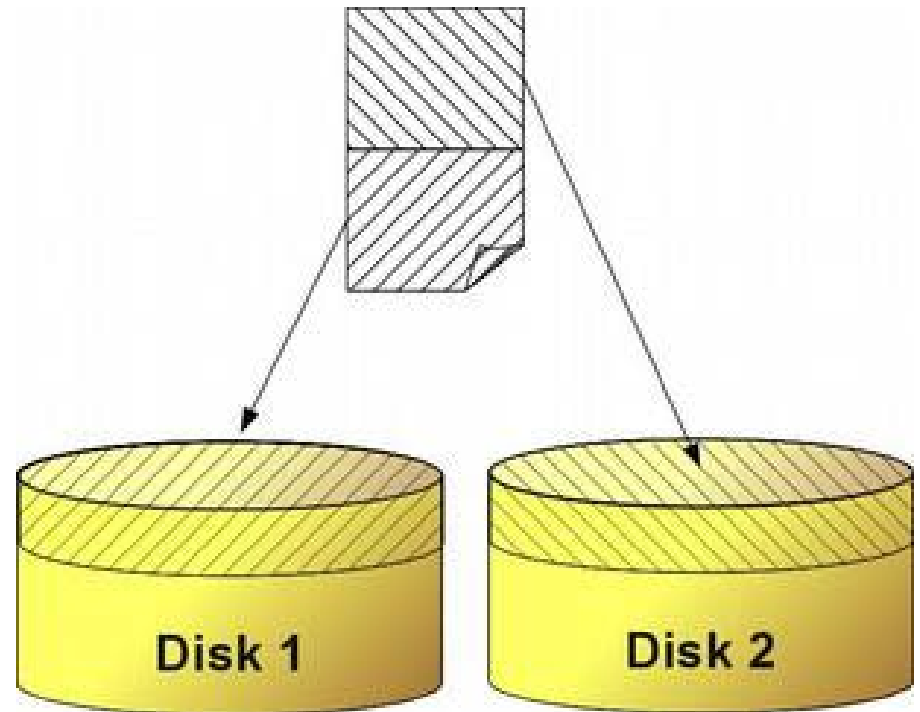
- Reliability or performance (or both) can be increased using different RAID “levels”.
- Let us examine some of the most important:
- Definitions:
  - S: Hard disk drive size.
  - N: Number of hard disk drives in the array.
  - P: Average performance of a single hard disk drive (MB/sec).

# RAID 0: striping

- Performance =  $P * N$



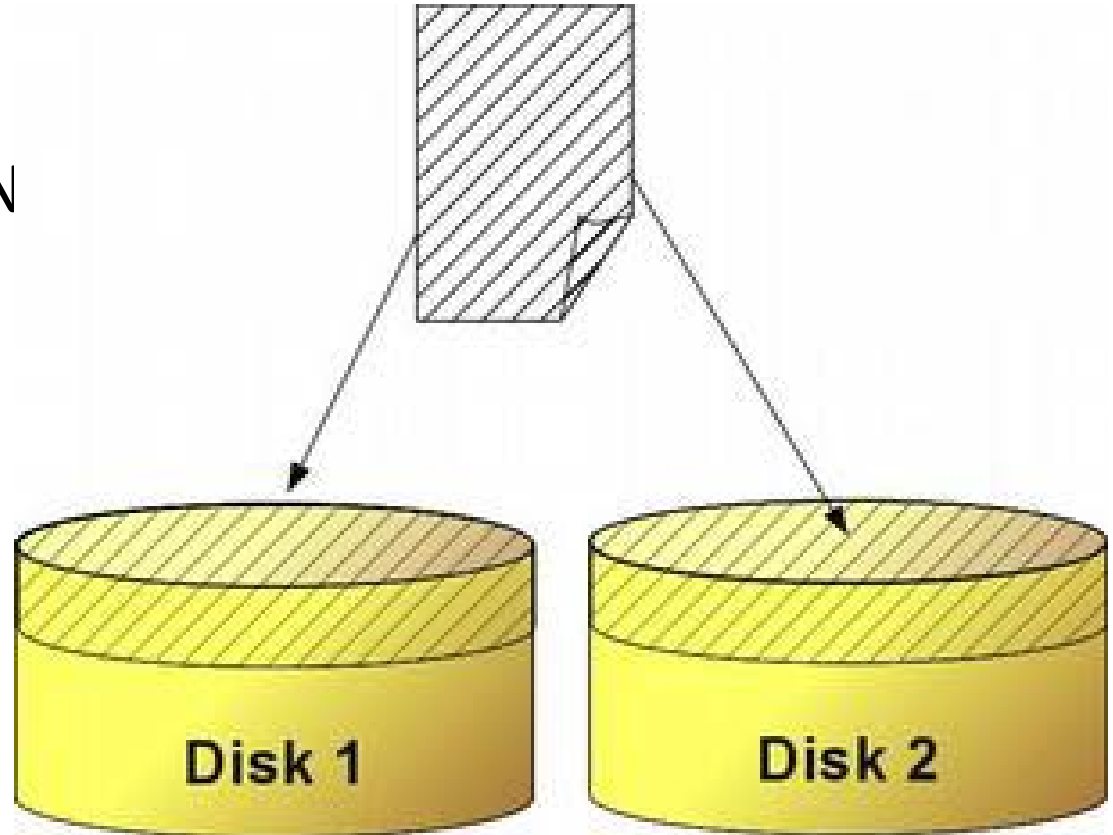
- Capacity =  $N * S$





# RAID 1: redundancy

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



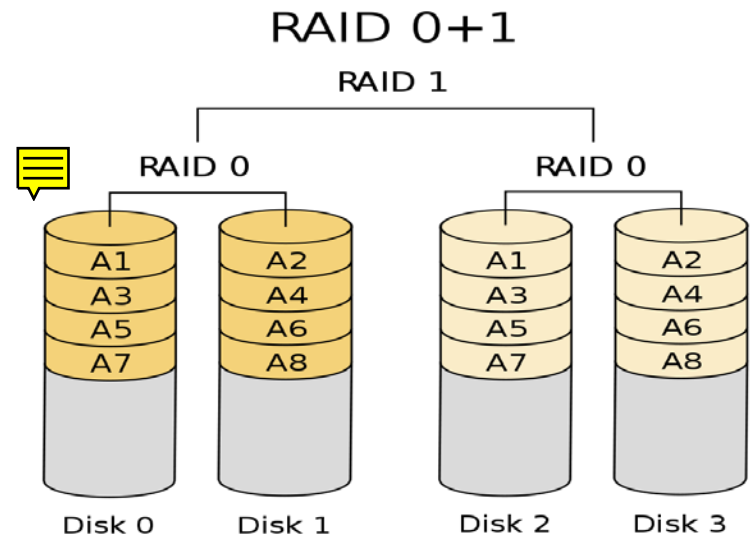
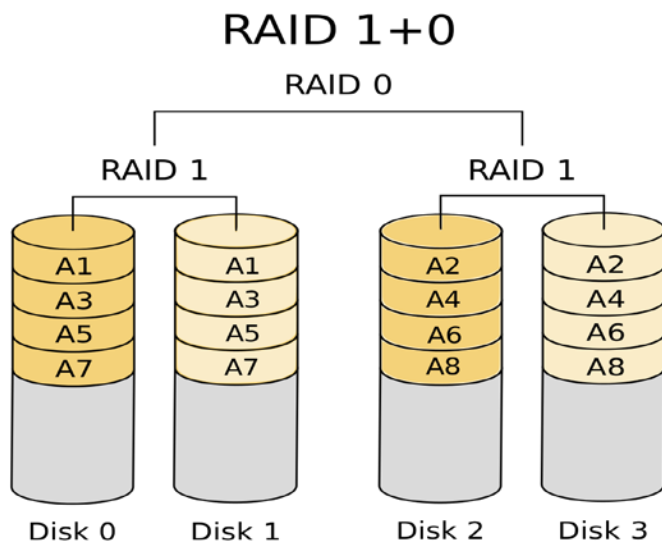


# RAID 10: striping + redundancy

## (1+0 / 0+1)



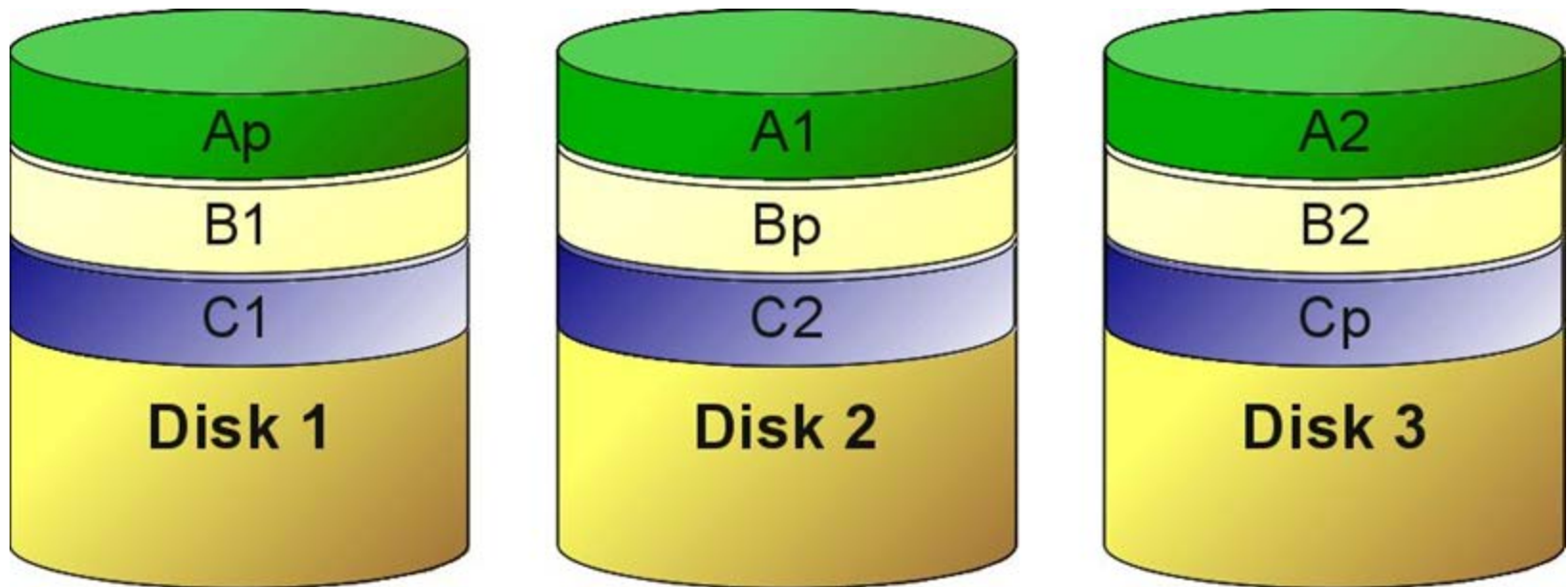
- Raid 1+0 : mirrored sets in a striped set
- the array can sustain multiple drive losses so long as no mirror loses all its drives
- Raid 0+1: striped sets in mirrored set
- if drives fail on both sides of the mirror the data are lost



# RAID 5



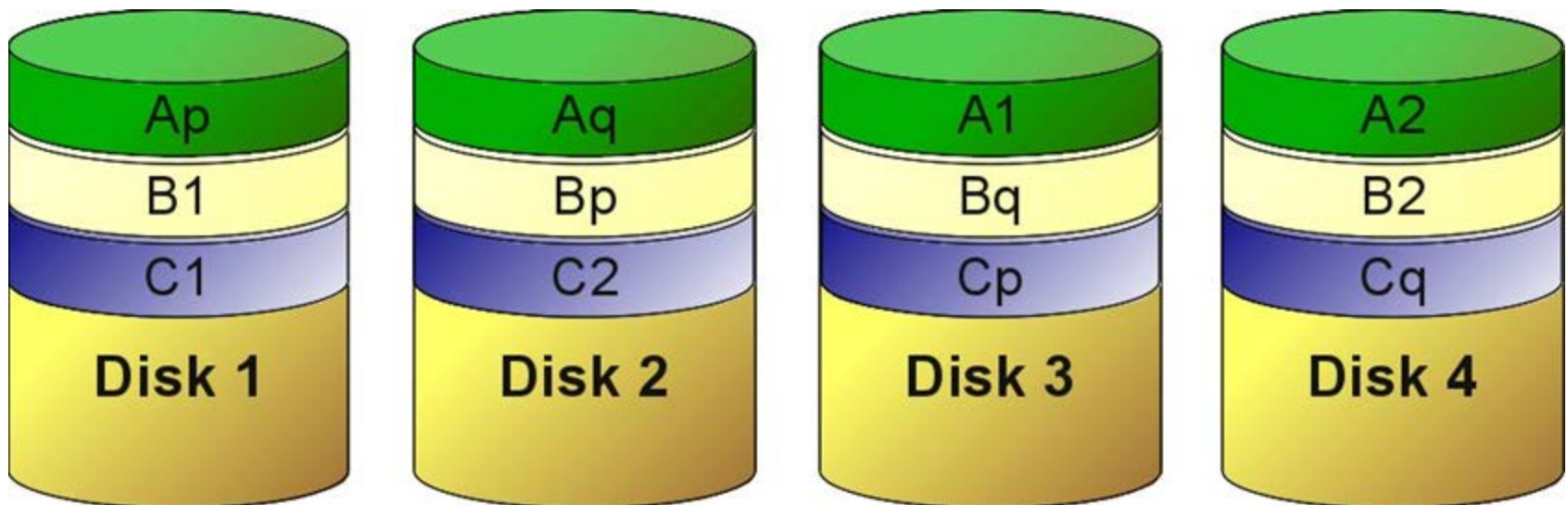
- One disk can fail
- Distributed parity



# 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

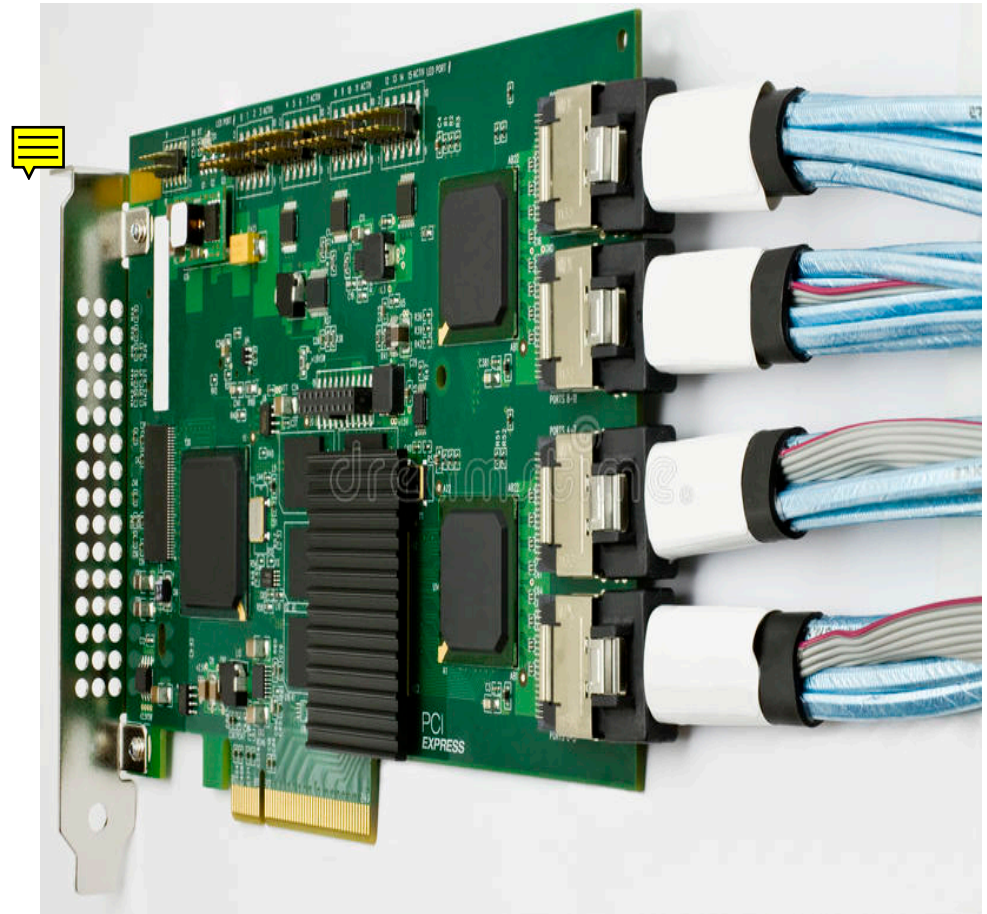
From <http://en.wikipedia.org/wiki/RAID>

# 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)

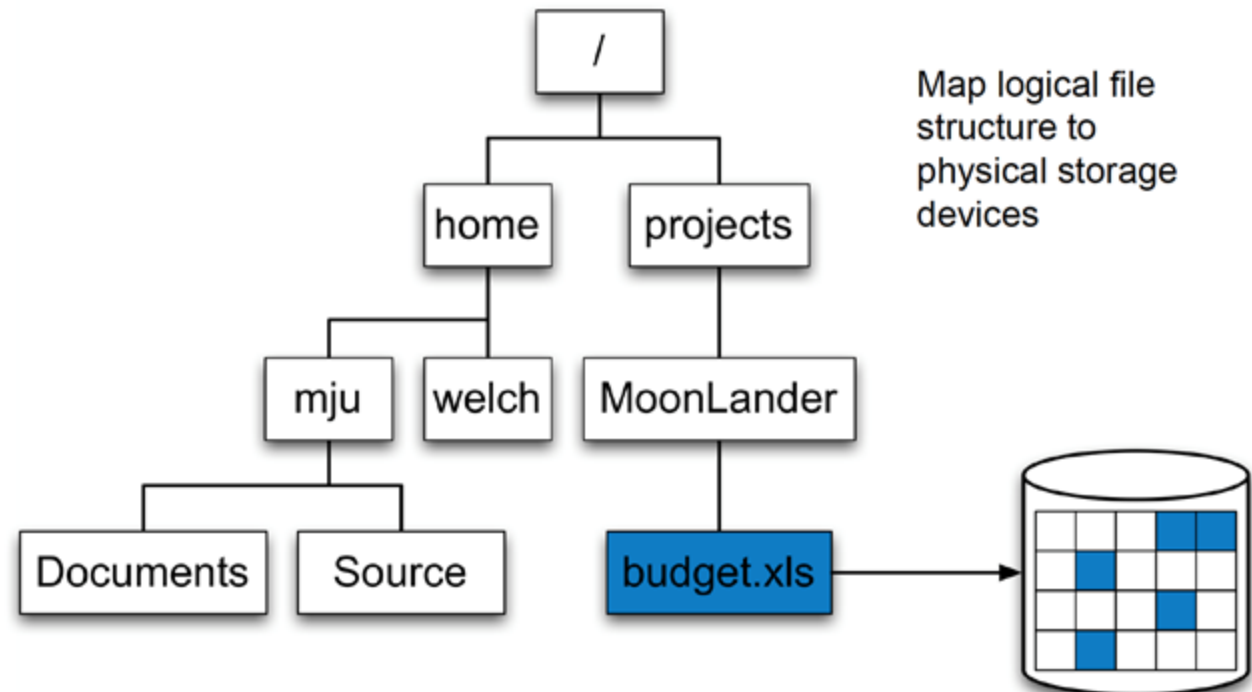


# Intro: Filesystems



# 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)

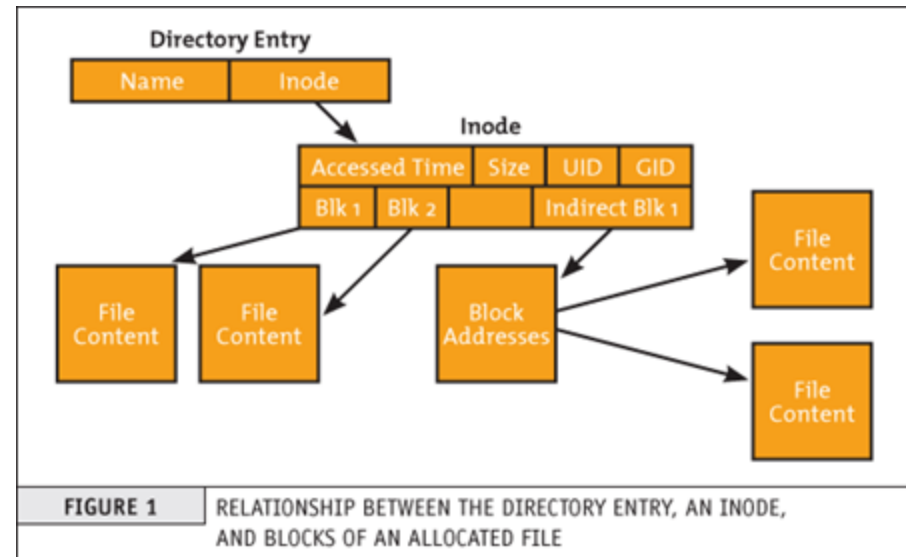
- **Metadata:** A general term referring to information that is about something but not directly part of it.
- **Journaling:** write data to journal, commit to file system when complete in atomic operation
  - reduces risk of corruption and inconsistency
- **Attribute:** A name and value associated with 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



# 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 /illumina_run	4.6G	1.9M	4.6G	1%	
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
```

# The messy situation on nfs01

<del>/dev/mapper/test_vol</del>	<del>187G</del>	<del>33M</del>	<del>187G</del>	<del>1%</del>	<del>/test_vol</del>
/dev/mapper/orfeo_repo	94G	33M	94G	1%	/orfeo_repo
/dev/mapper/read_the_docs	94G	218M	93G	1%	/read_the_docs
/dev/mapper/illumina_decode	1.9T	936G	927G	51%	/illumina_decode
/dev/mapper/nep	94G	94G	148K	100%	/nep
/dev/mapper/opt_area	477G	210G	267G	45%	/opt/area
<del>/dev/mapper/storage</del>	<del>37T</del>	<del>1.1T</del>	<del>36T</del>	<del>3%</del>	<del>/storage</del>
/dev/mapper/orfeo_replicated_share	9.1T	3.8T	5.4T	42%	/orfeo_replicated_share
/dev/mapper/borg_repos	14T	8.2G	14T	1%	/borg_repos
/dev/mapper/lala_storage	4.6T	2.4T	2.3T	52%	/lala_storage
<del>/dev/mapper/tesi_fabricei</del>	<del>9.1T</del>	<del>3.1T</del>	<del>6.1T</del>	<del>34%</del>	<del>/tesi_fabricei</del>
/dev/mapper/illumina_run	128T	109T	20T	85%	/illumina_run
/dev/mapper/burlo_long_term_storage	91T	8.6T	83T	10%	/burlo_long_term_storage
/dev/mapper/onp_run_1	117T	56T	61T	48%	/onp_run_1
/dev/mapper/lage_archive	128T	94T	34T	74%	/lage_archive
/dev/mapper/analisi_da_consegnare	100T	56T	45T	56%	/analisi_da_consegnare
/dev/mapper/long_term_storage	128T	112T	17T	88%	/long_term_storage
/dev/mapper/onpLVMVolGroup-onpLV	510T	62M	510T	1%	/TEST_onp_run

# 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

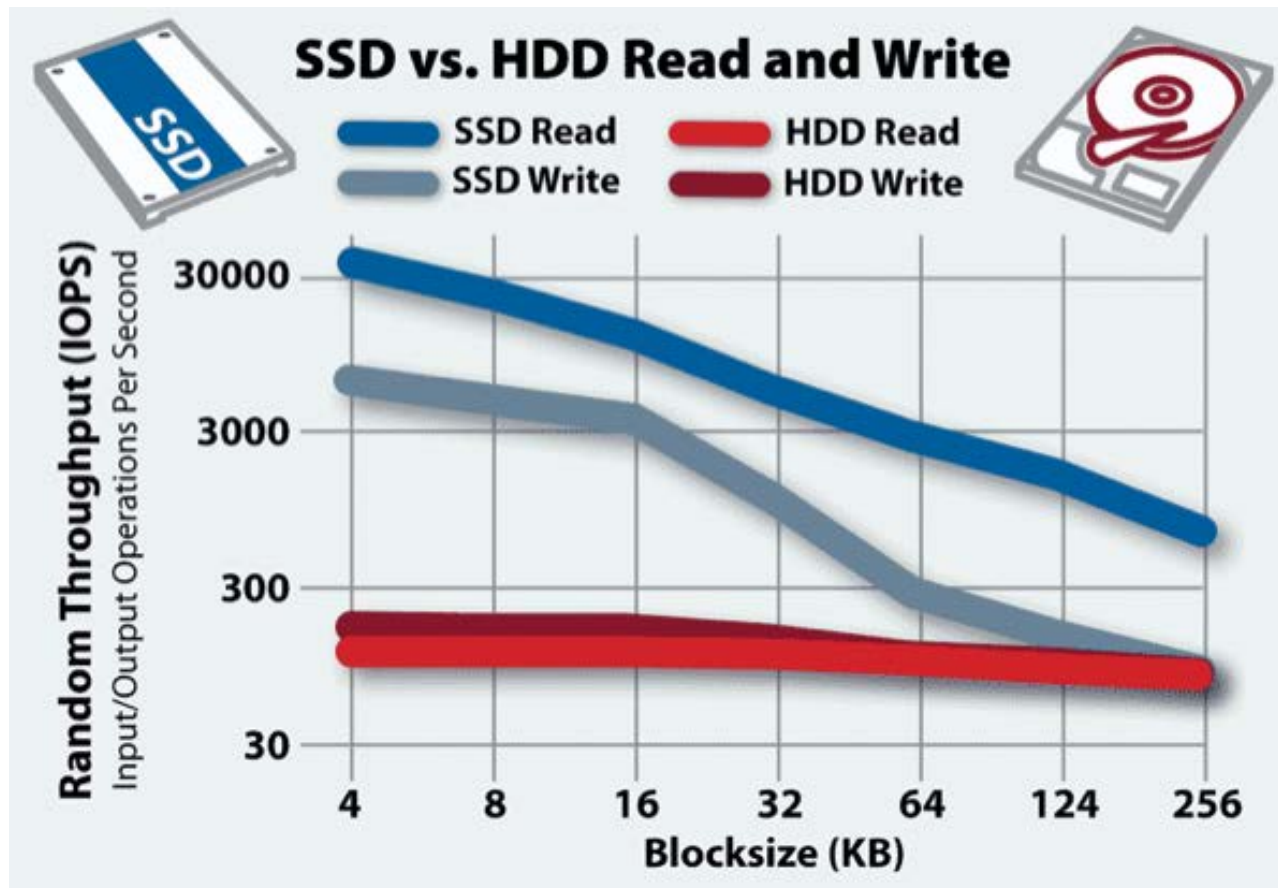
- 512 byte is a typical block-size of the disk:
- It cannot read less than 512 bytes, if you want to read less, read 512 bytes and discard the rest.
- File System block-size can be different

```
[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
```

# Parallel I/O in HPC

# A couple of citations

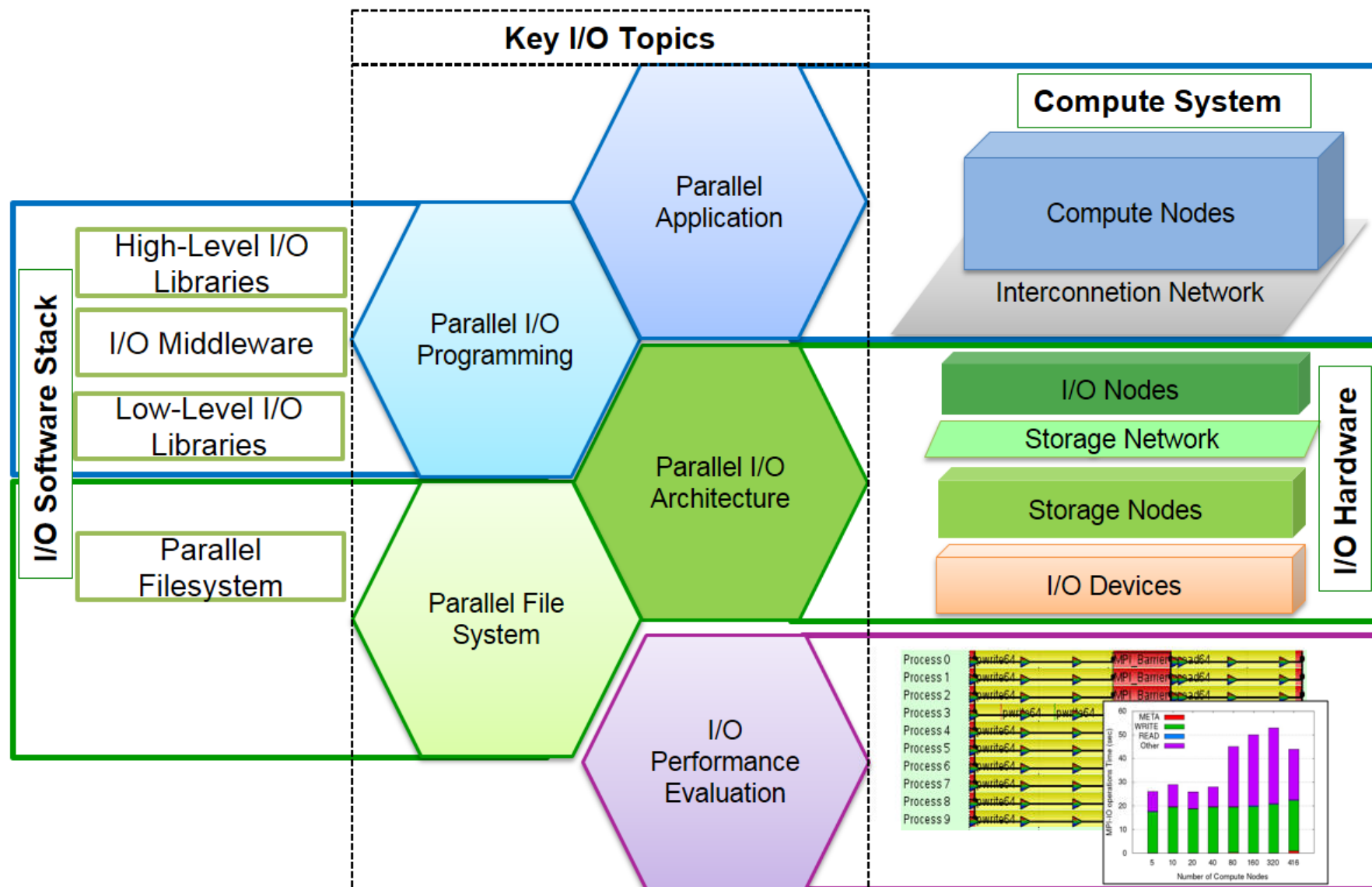
“Very few large scale applications of practical importance are NOT data intensive.”

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

# 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

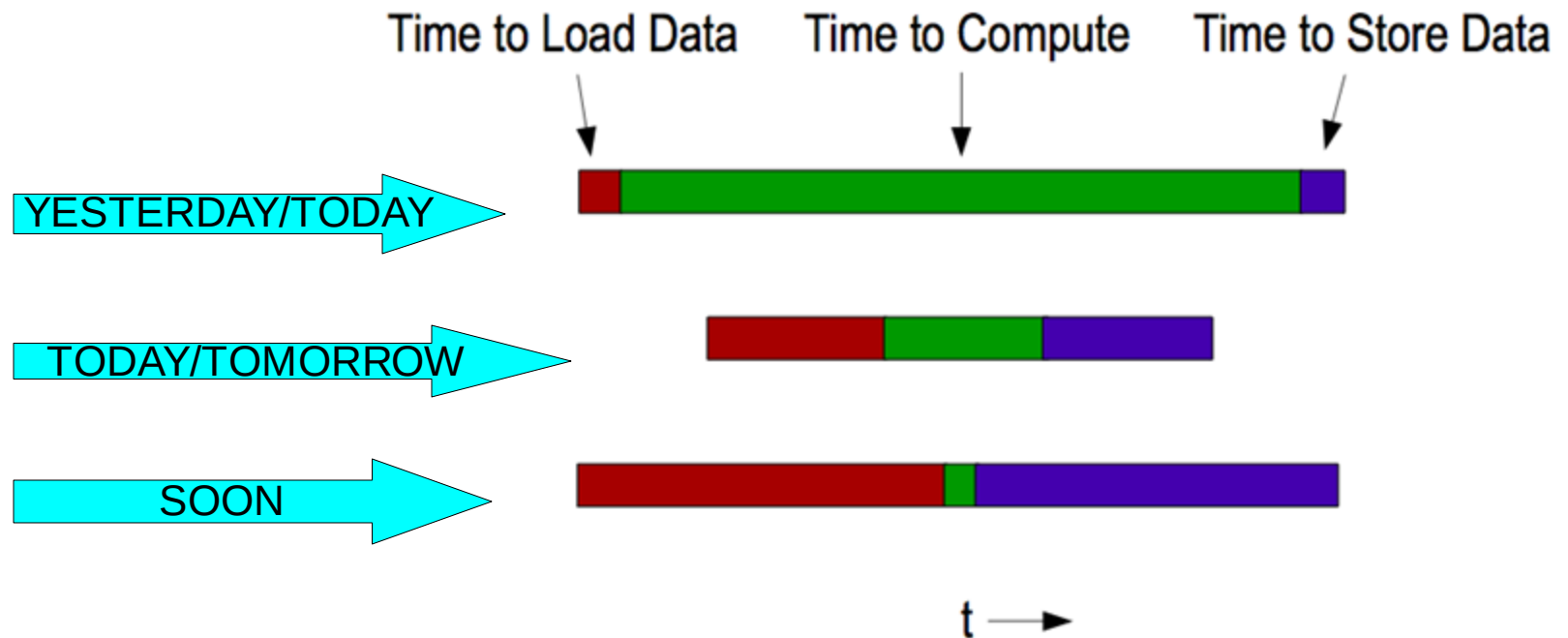
# Preprocessing/Post-processing phases..

- 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

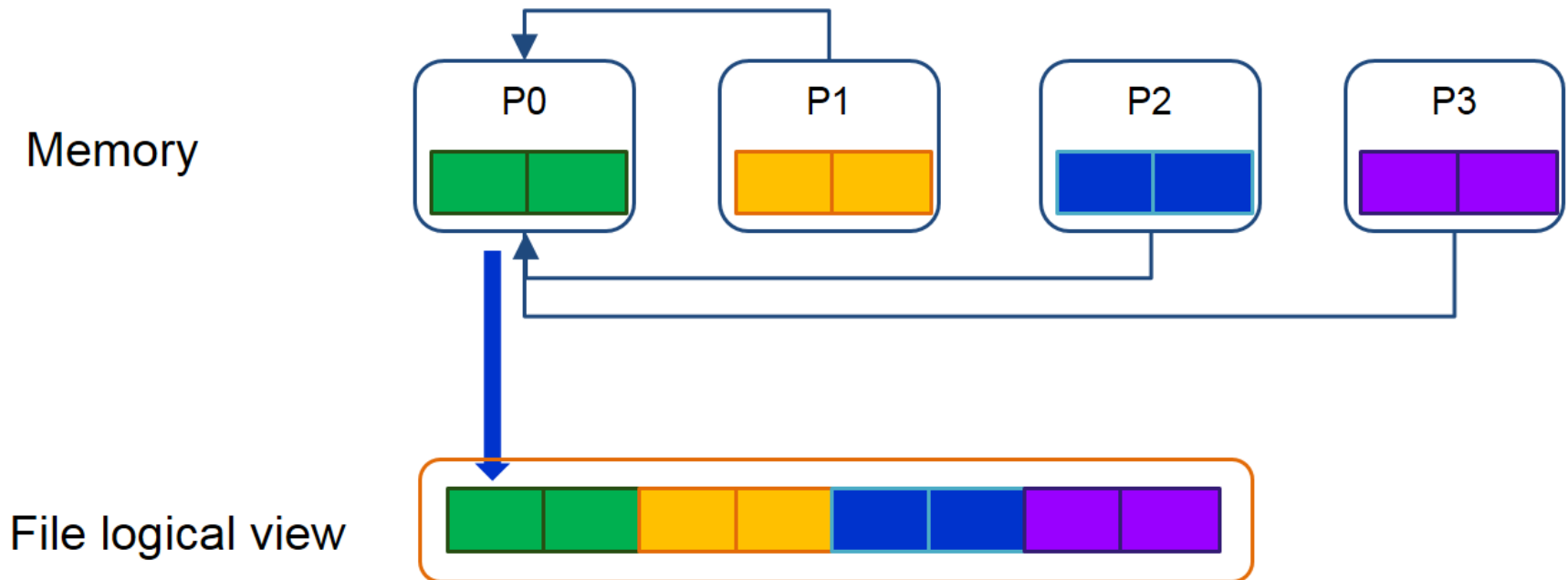
- I/O systems have very simple data models
  - Tree-based hierarchy of containers
  - Some containers have streams of bytes (files)
  - Others hold collections of other containers (directories or folders)
- Applications have data models appropriate to domain
  - Multidimensional typed arrays, images 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 : spokesperson

- 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.**

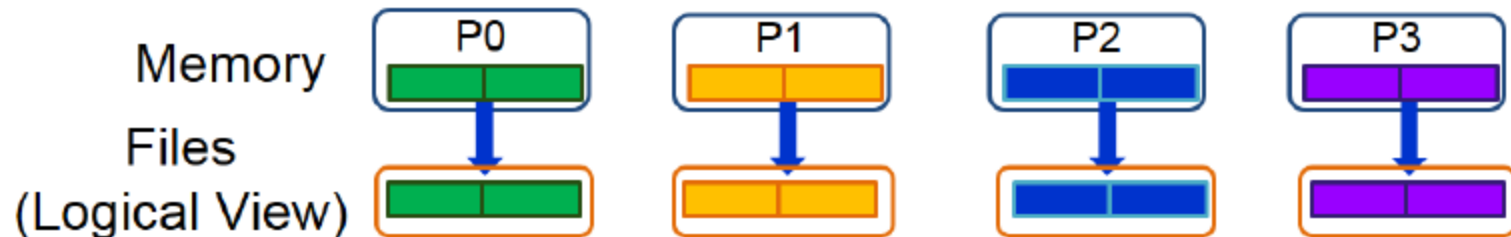


# Parallel I/O: File-per-Process

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 ?

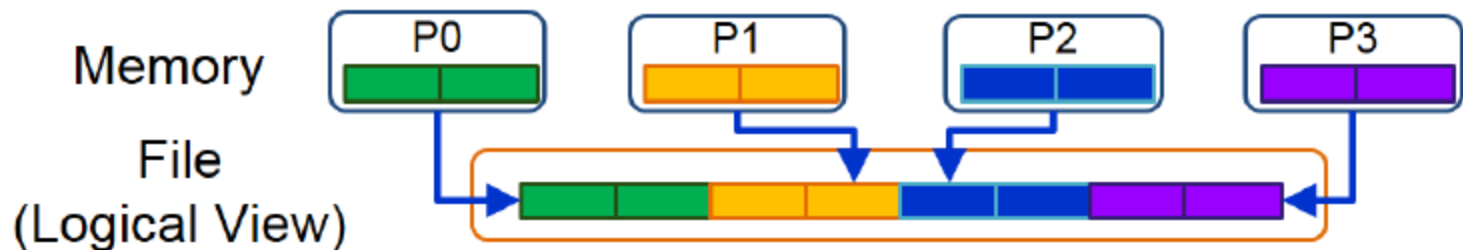
**1 file per process (UNIQUE access type)**



# 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..

## A Single shared File (SHARED access type)



# 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.

## Single file shared for "N" processes



# 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

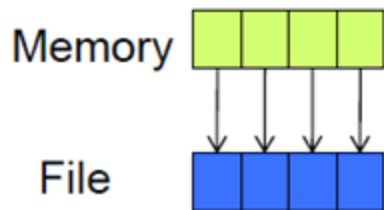
## Repetition

Single  
Occurrence

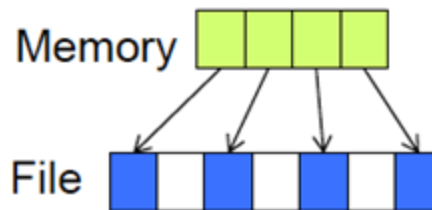
Multiple  
Occurrence

# Access Patterns

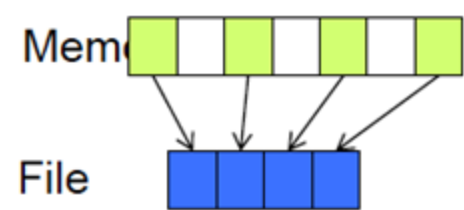
*Contiguous*



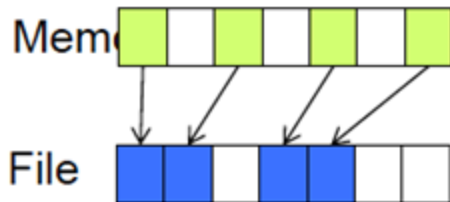
*Contiguous in memory, not in file*



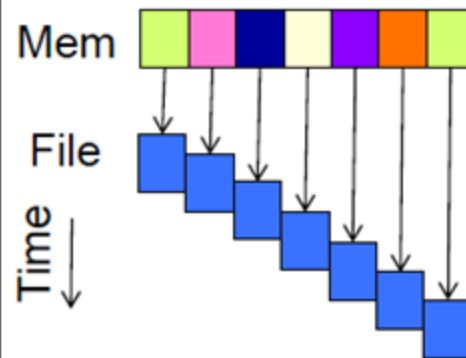
*Contiguous in file, not in memory*



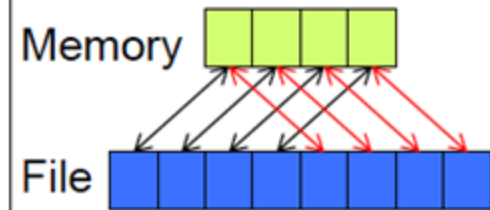
*Dis-contiguous*



*Bursty*



*Out-of-Core*





# Software/Hardware stack for I/O

## High-Level I/O Library

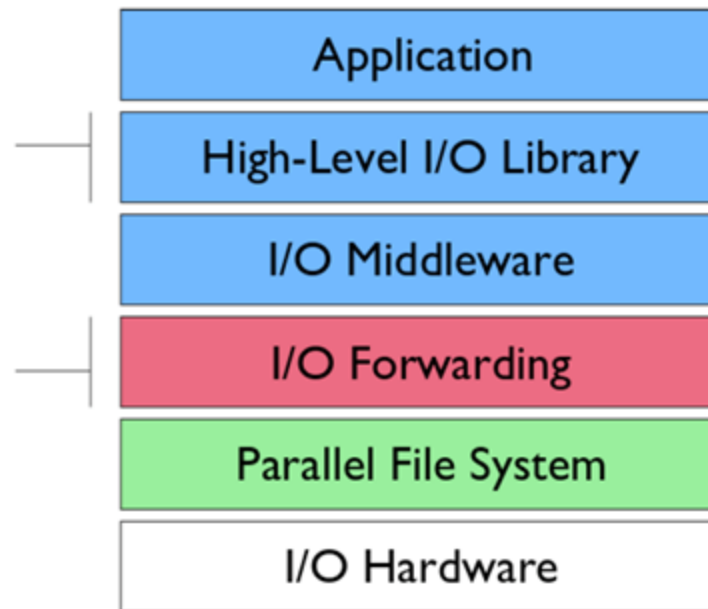
maps application abstractions onto storage abstractions and provides data portability.

*HDF5, Parallel netCDF, ADIOS*

## I/O Forwarding

bridges between app. tasks and storage system and provides aggregation for uncoordinated I/O.

*IBM ciod, IOFSL, Cray DVS*



## I/O Middleware

organizes accesses from many processes, especially those using collective I/O.

*MPI-IO*

## Parallel File System

maintains logical space and provides efficient access to data.

*PVFS, PanFS, GPFS, Lustre*

# 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 ?

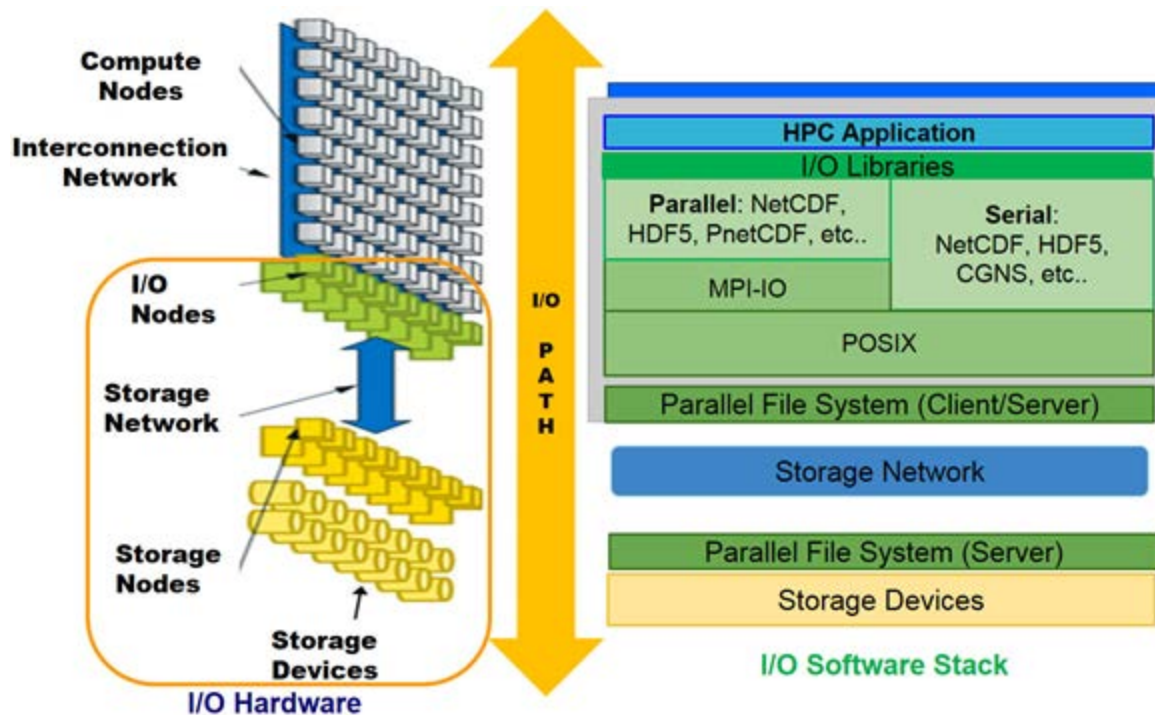
- Why do I/O in MPI?
- Why not just POSIX?
  - Parallel performance
  - Single file (instead of one file / process)
- MPI has replacement functions for POSIX I/O
- Multiple styles of I/O can all be expressed in MPI
  - Contiguous vs non contiguous etc....

To be continued...

# 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

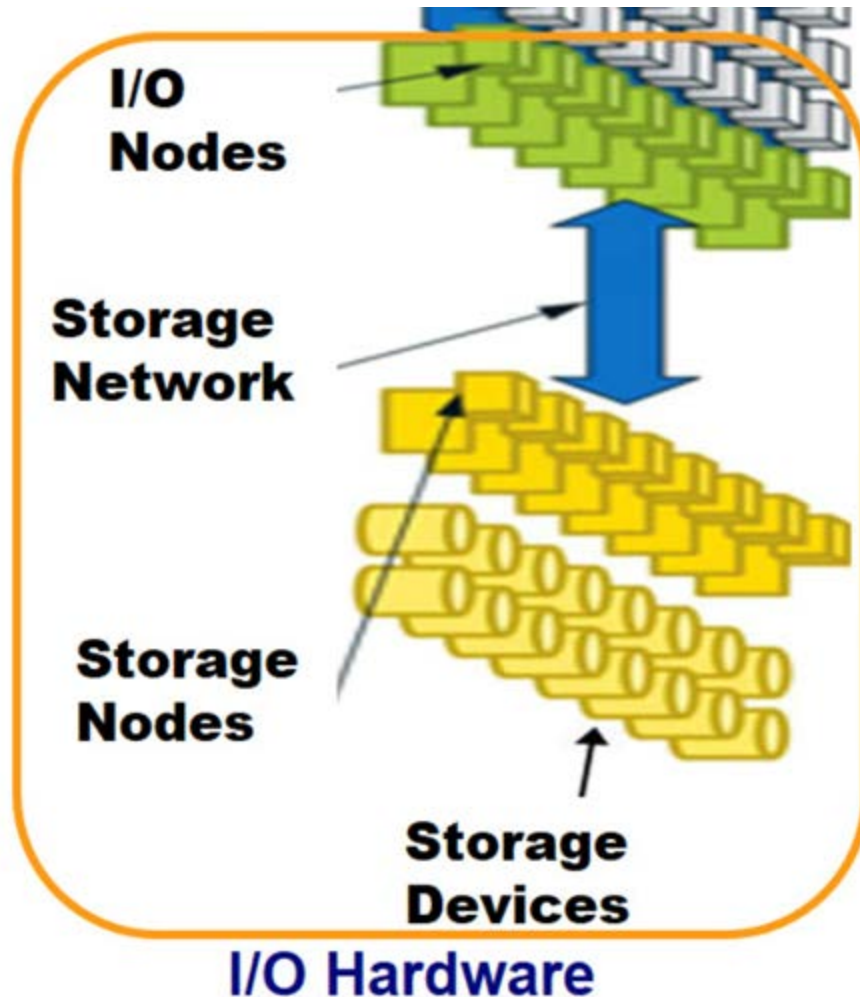
# A graphical view:



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

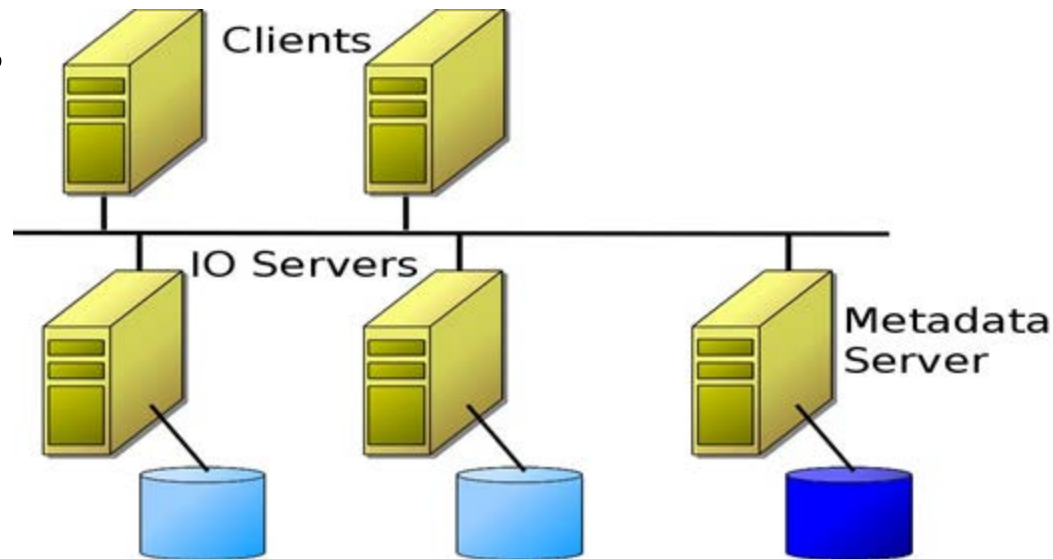


# Parallel File System: I/O hardware



# 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:

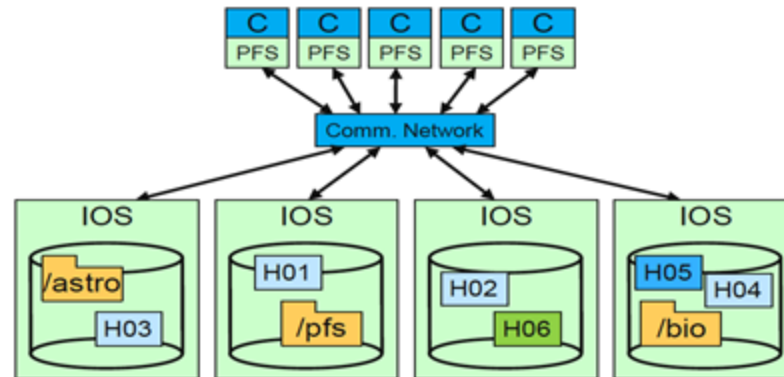
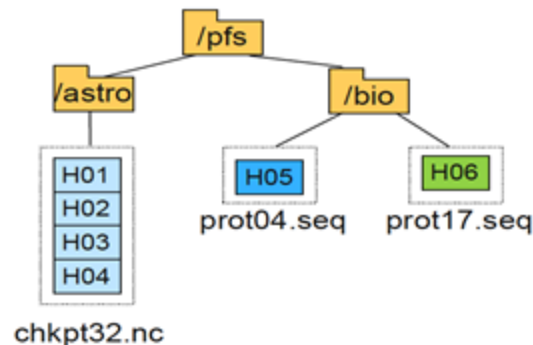
- Nodes, Disks, controllers, and interconnects
- Hardware defines the peak performance of the I/O system:
  - raw bandwidth
  - Minimum latency
- At the hardware level, data is accessed at the granularity of blocks, either **physical disk blocks** or **logical blocks spread across multiple physical devices** 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 ( $\geq 64\text{kB}$ )
  - 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..

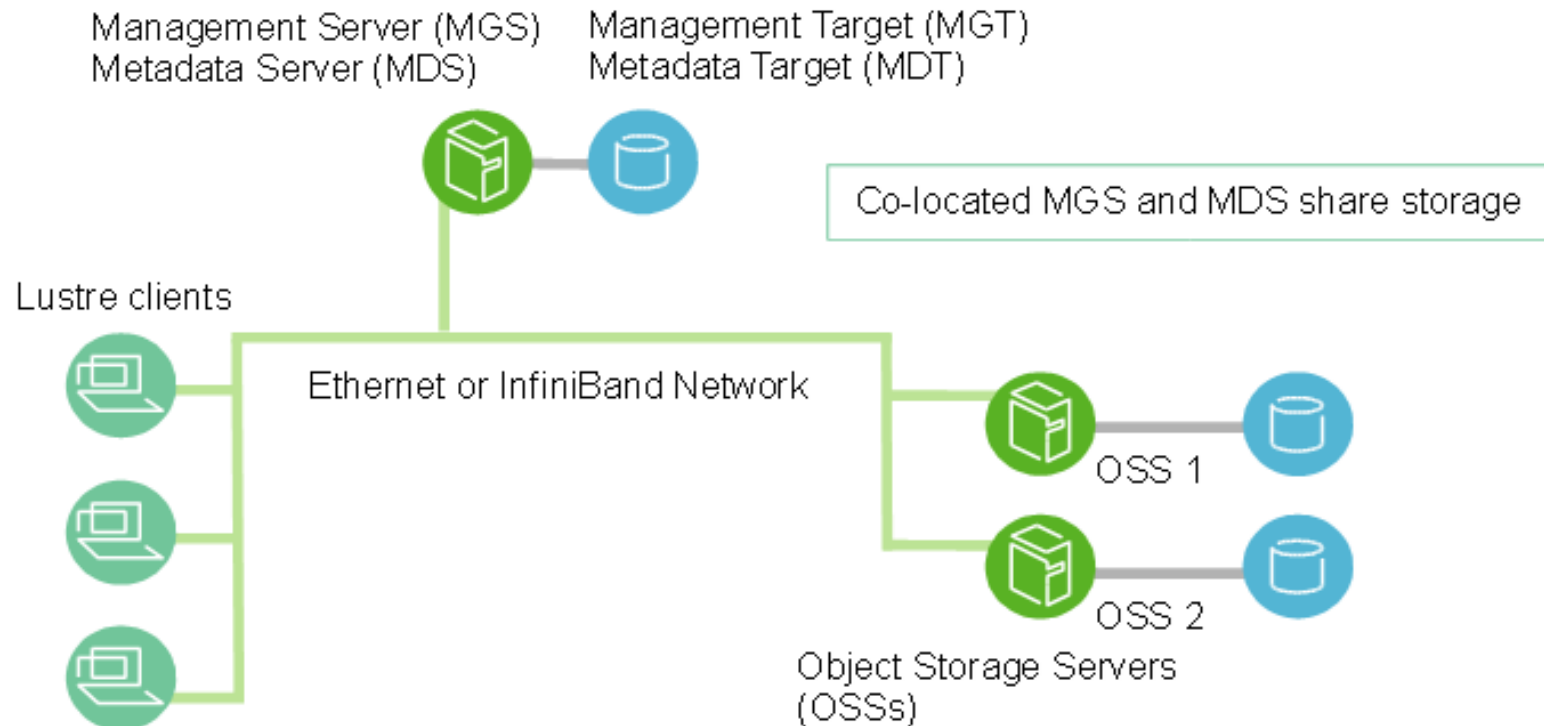
- An example parallel file system, with large astrophysics checkpoints distributed across multiple I/O servers (IOS) while small bioinformatics files are each stored on a single IOS



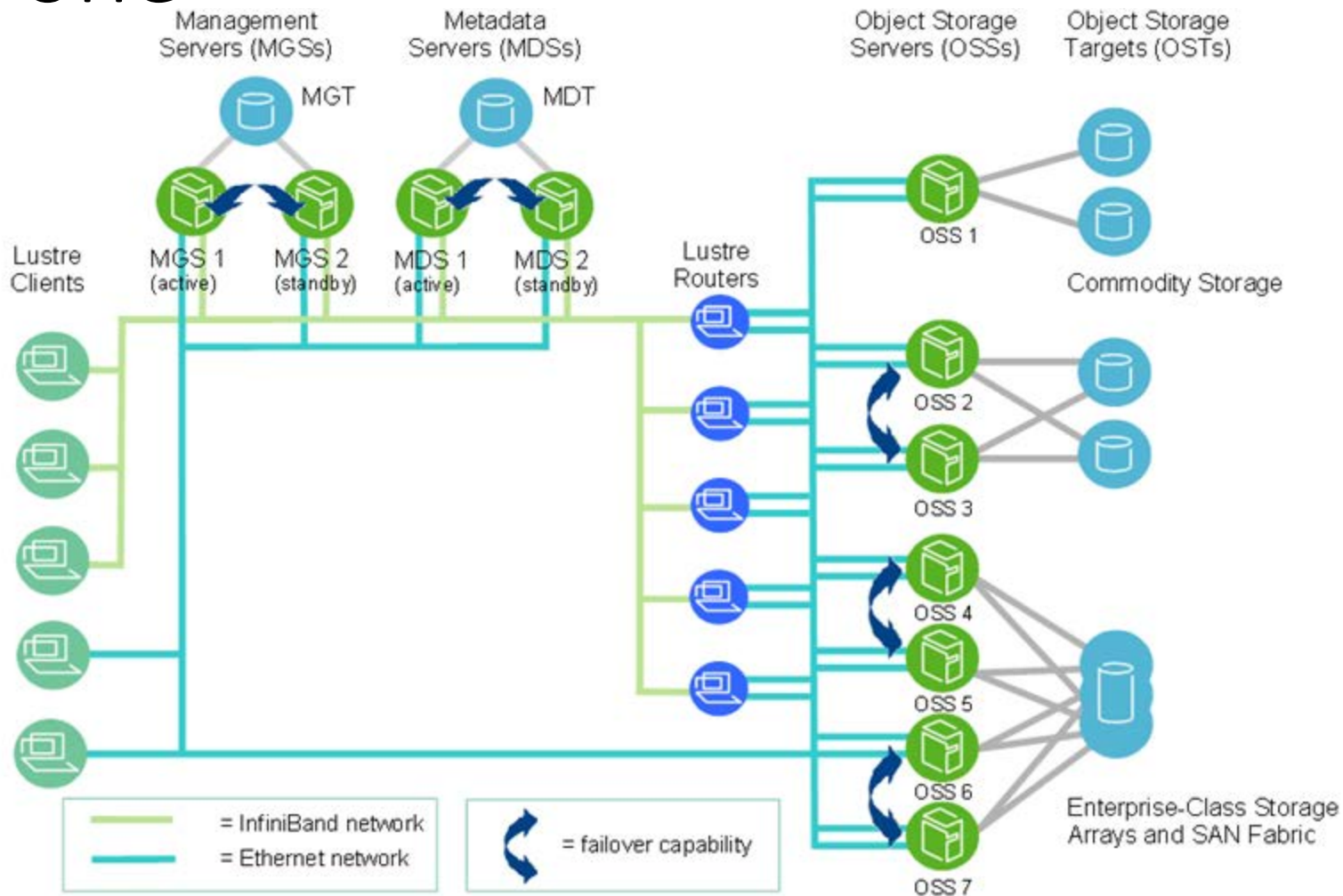
# What is available on the market ?

- 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

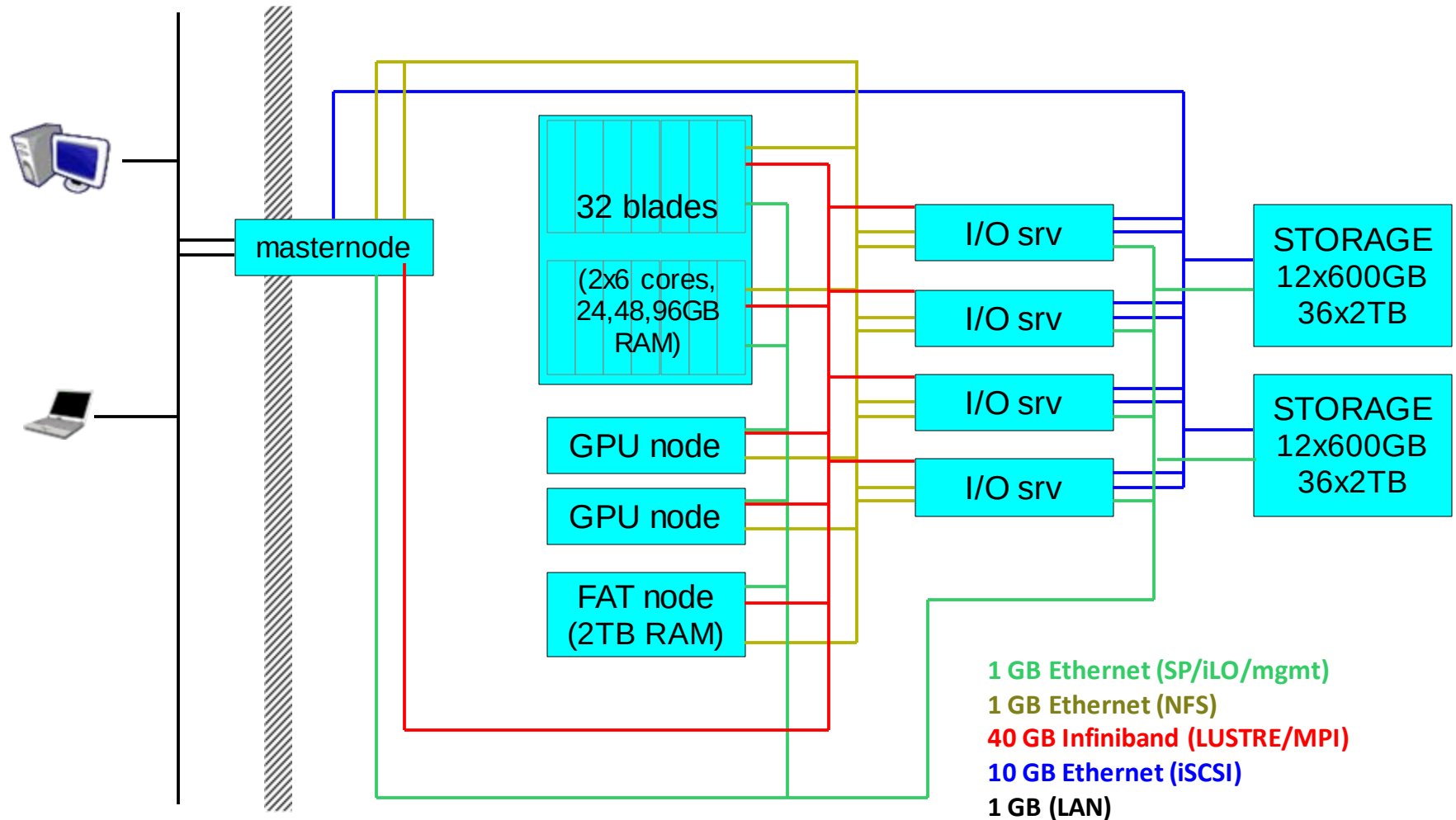


# Lustre in two pictures: complex one

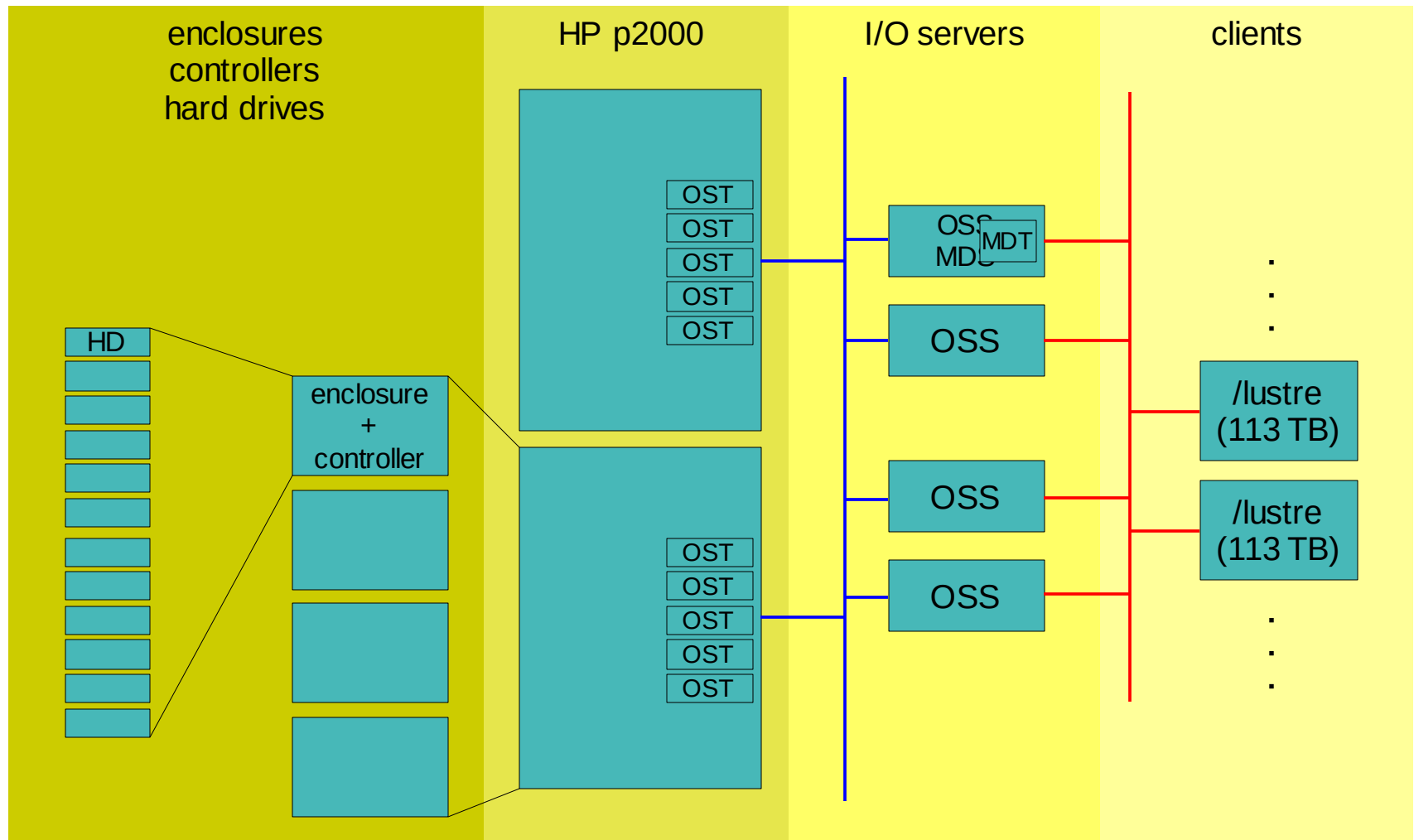




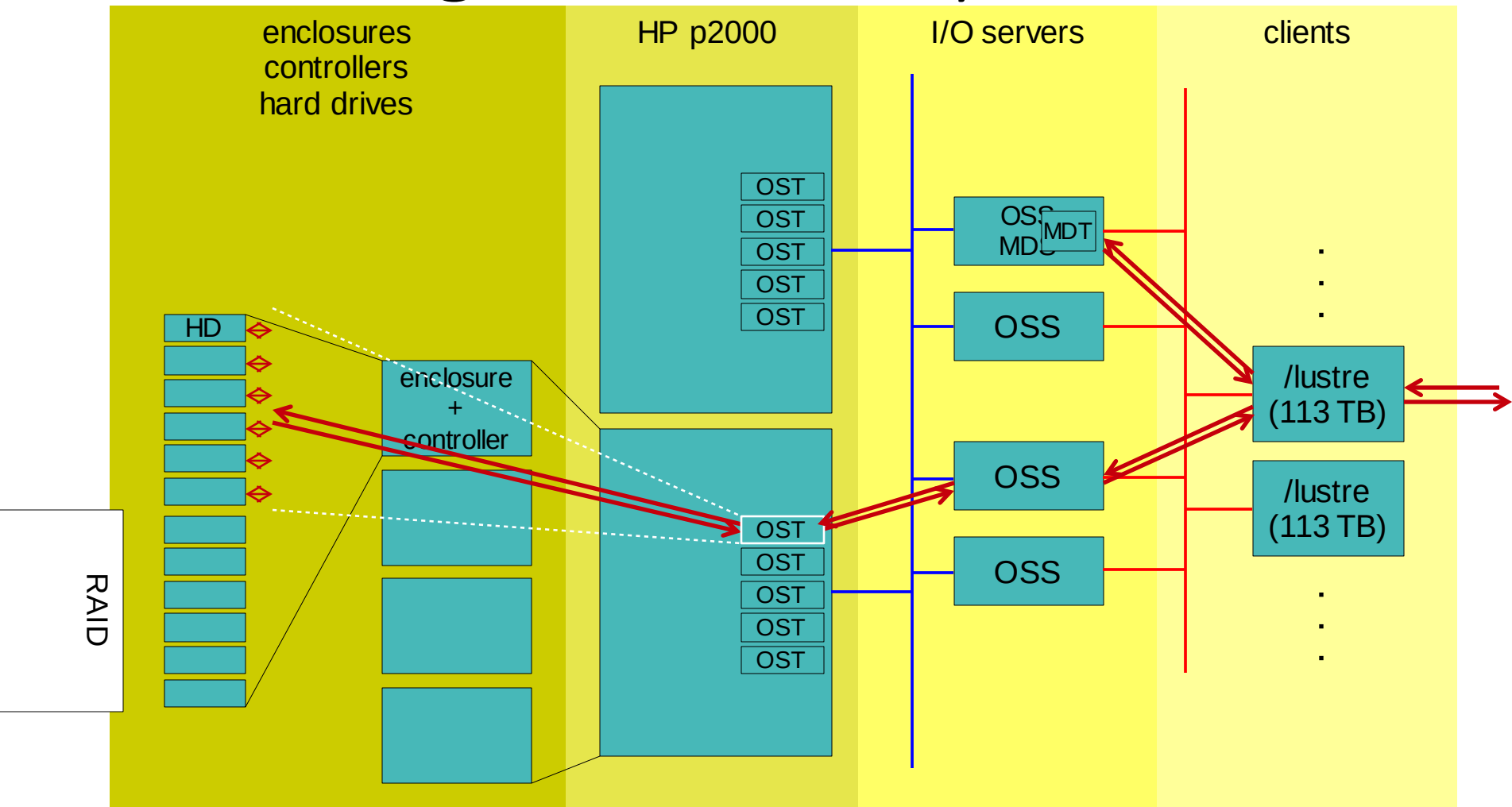
# HPC infrastructure @ CRIBI



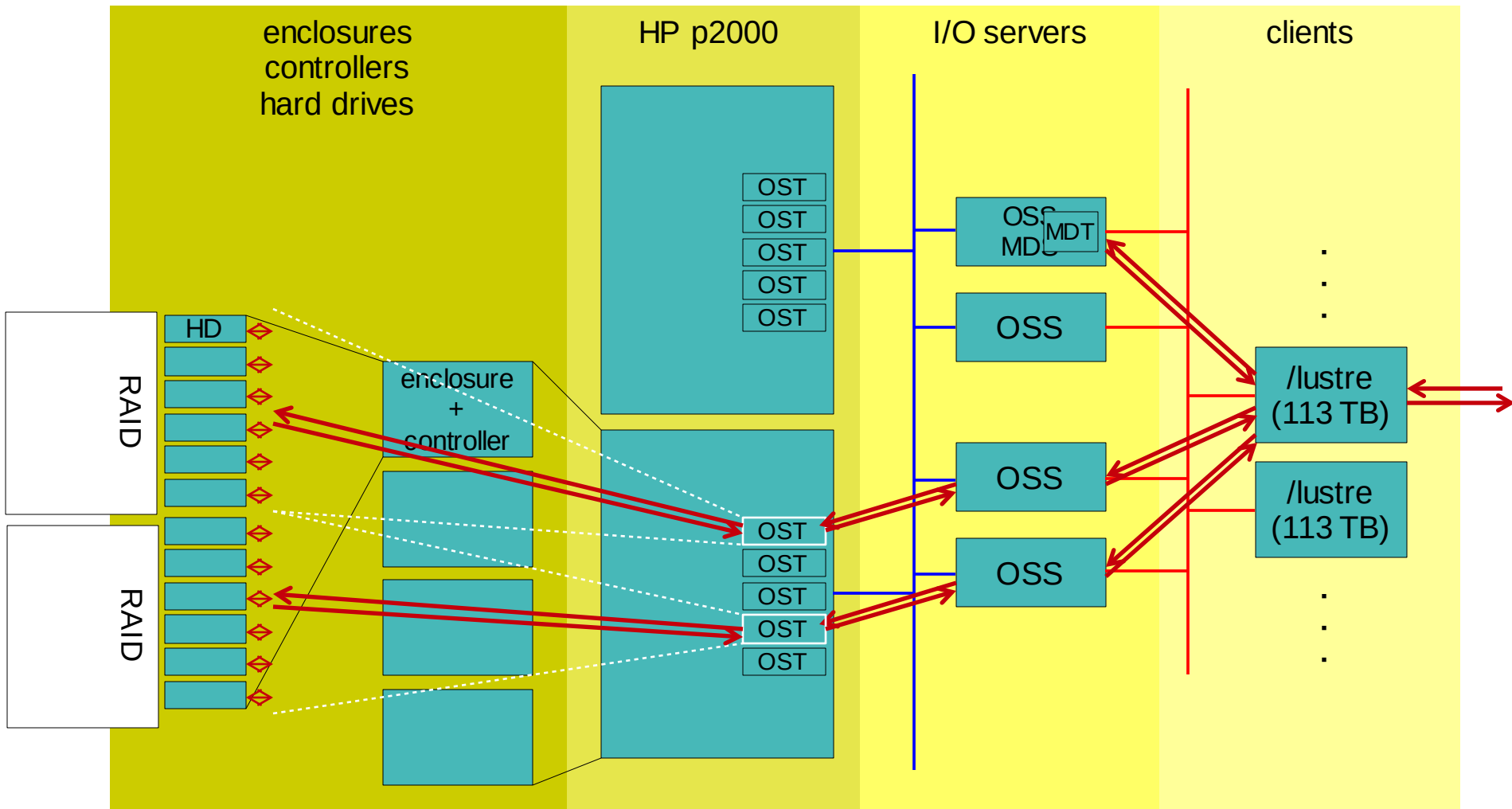
# LUSTRE@CRIBI as storage solution



# accessing LUSTRE filesystem



# why “parallel” filesystem?



# Expected performance

- Elements of the infrastructure:
  - Network Speed: Infiniband QDR :3.2GB/sec for server  
--> Network aggregate bandwidth:  $3.2 \times 4 \sim 12\text{GB/se}$
  - 4 IO-SRV two OST each
    - Each OST: RAID 6 6 disks
    - OST Aggregate bandwidth:  $(6-2)*100 = 400 \text{ Mb/seconds}$ 
      - [Disk speed: 100 Mb/seconds]
  - Node Aggregate bandwidth  $400 \times 2 = 800 \text{ Mb/sec}$

Peak performance :  $4 \times 800 = 3.2 \text{ GB/sec read/write}$

# overall LUSTRE performance



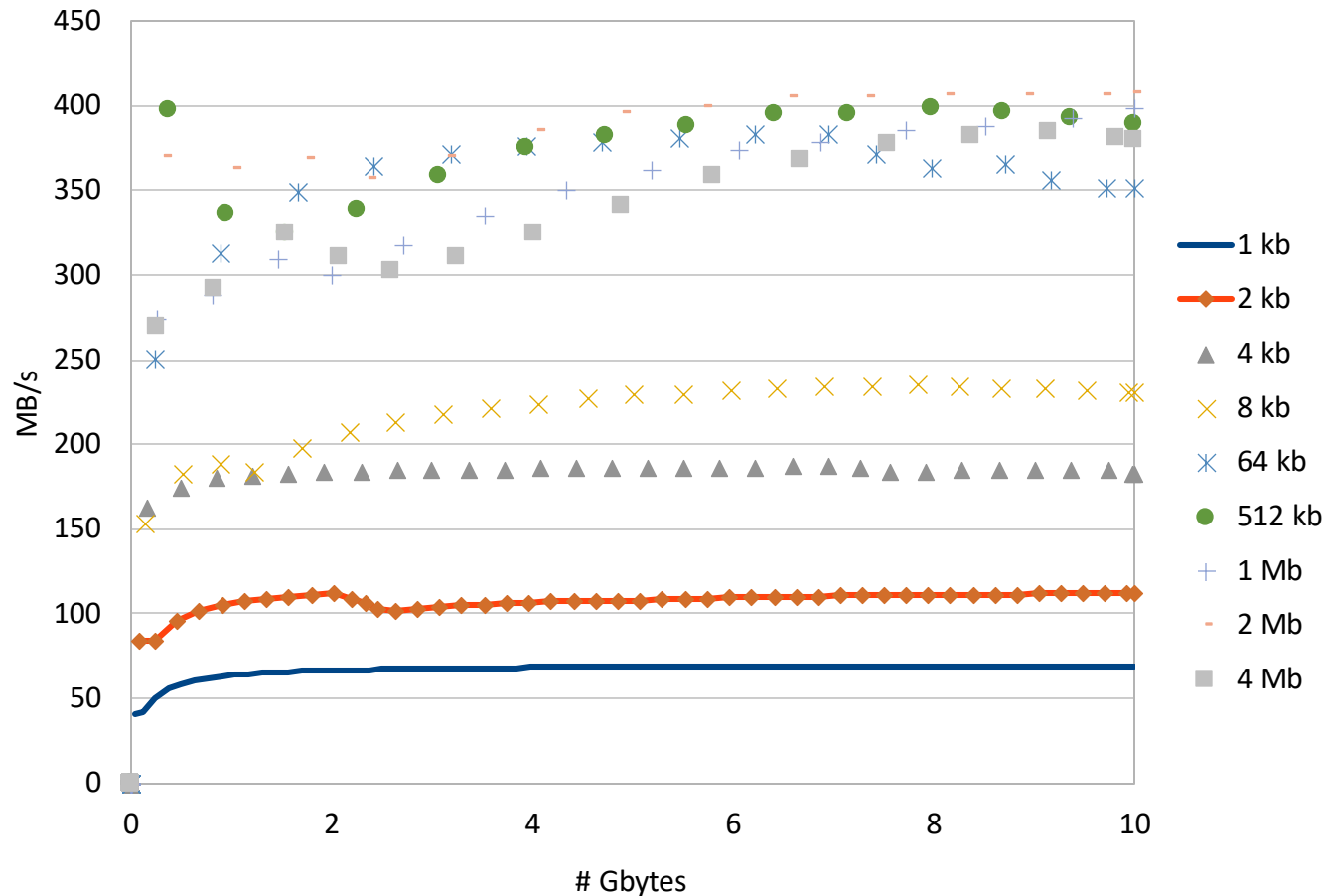
- sequential write/read by iotune
- 1 ~ 8 clients, 1 ~ 4 proc/client
- 32 GB files writing
- 64 GB files reading



- ~ 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...

writing 1 file with variable block size



# ORFEO choice: CEPH

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

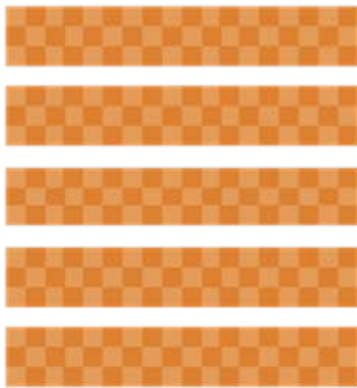


# A short introduction to CEPH

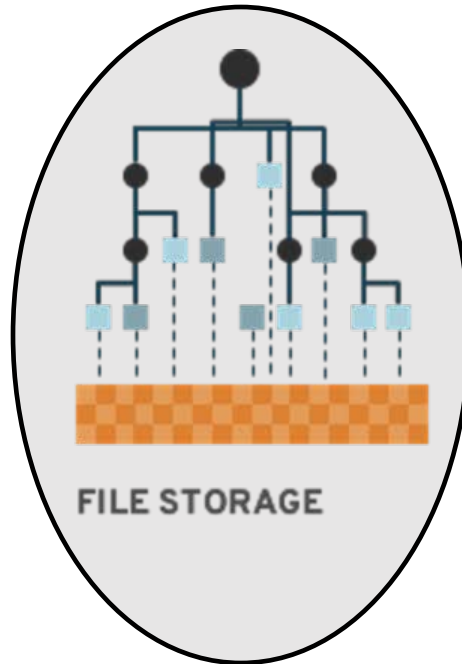
# CEPH storage

- Open-source distributed storage solution
- Object based storage
- Highly scalable
- Built around the CRUSH algorithm, by Sage Weil – <http://ceph.com/papers/weil-crush-sc06.pdf>
- Supports multiple access methods [File, Block, Object]

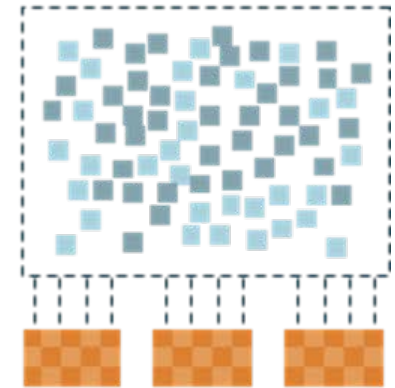
# Access methods:



**BLOCK STORAGE**

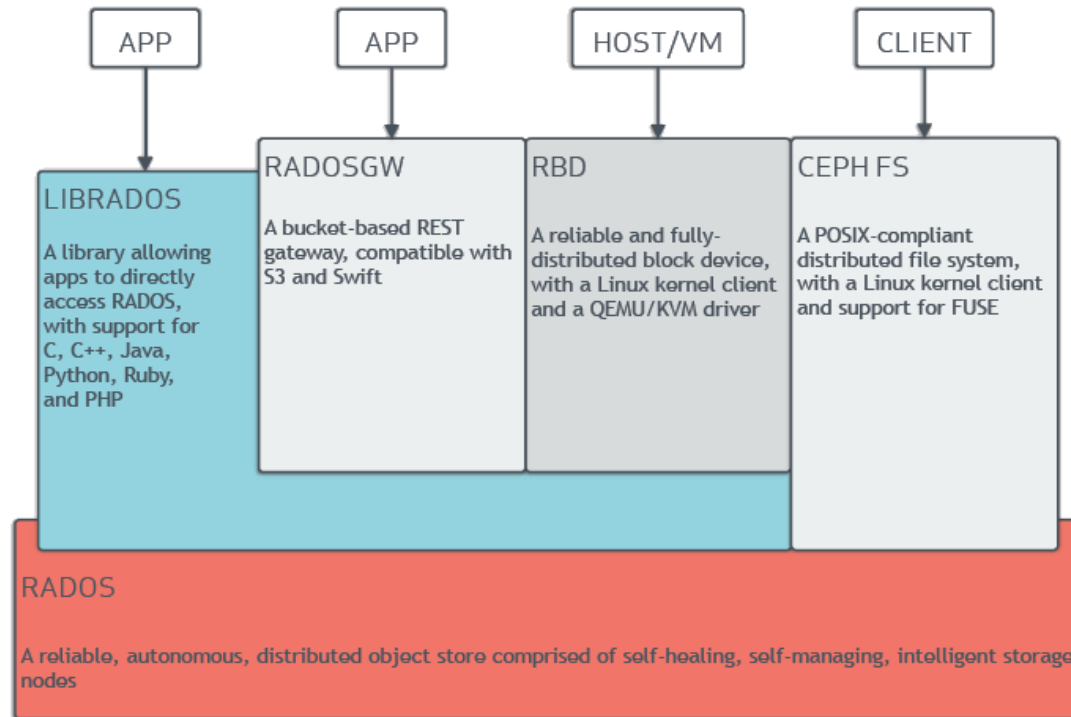


**FILE STORAGE**



**OBJECT STORAGE**

# CEPH Storage Architecture



# CEPH storage cluster: RADOS

- **RADOS** (Reliable Autonomic Distributed Object Store)
  - This layer provides the CEPH software defined storage with the ability to store data (serve IO requests, protect the data, check the consistency and the integrity of the data through built-in mechanisms).
- The RADOS layer is composed of the following daemons:
  - MONs or Monitors
  - OSDs or Object Storage Devices
  - MGRs or Managers
  - MDSs or Meta Data Servers (only for CEPHfs)

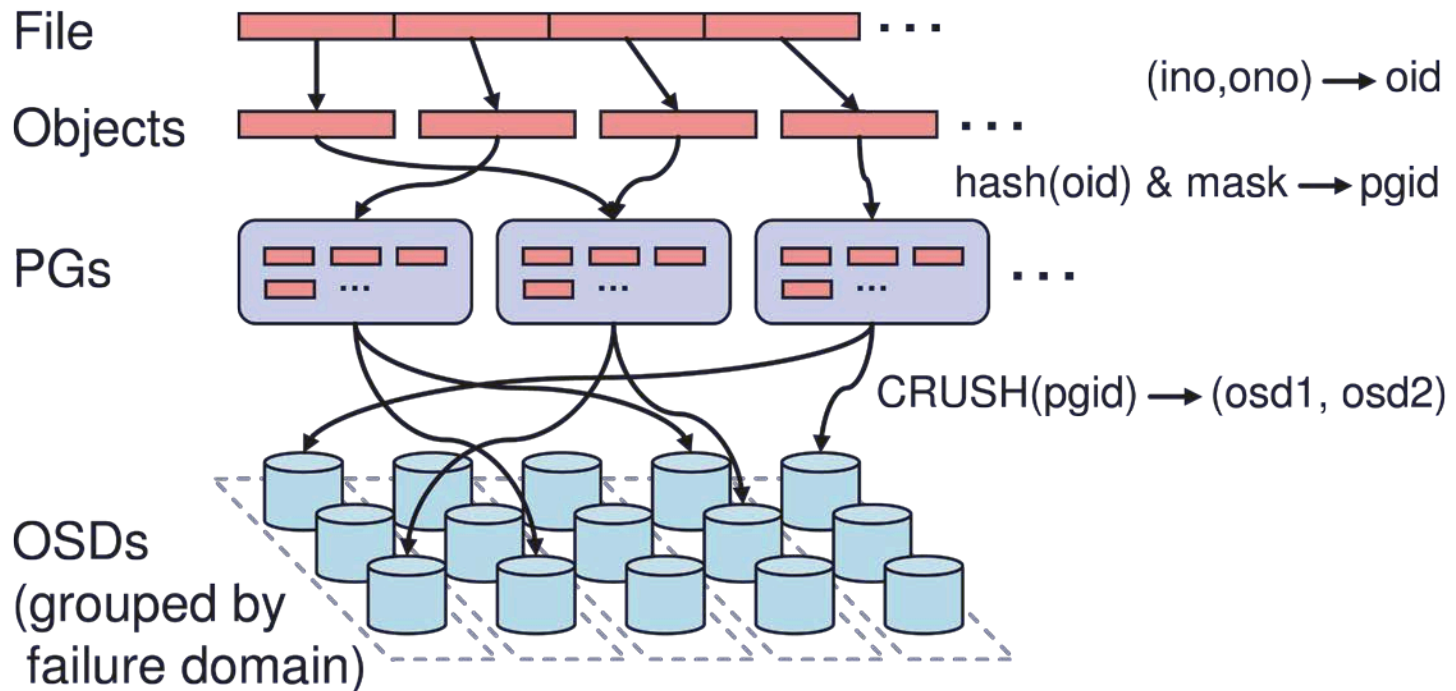
# What are they doing ?

- A CEPH Monitor maintains a master copy of the cluster map. A cluster of CEPH monitors ensures high availability should a monitor daemon fail. Storage cluster clients retrieve a copy of the cluster map from the CEPH Monitor.
- A CEPH OSD Daemon checks its own state and the state of other OSDs and reports back to monitors.
- A CEPH Manager acts as an endpoint for monitoring, orchestration, and plug-in modules.
- A CEPH Metadata Server (MDS) manages file metadata when CephFS is used to provide file services.

# Distributed Object Storage

- Files are split across objects
- Objects are members of placement groups
- Placement groups (PG) are distributed across OSDs.
- CRUSH (Controlled Replication Under Scalable Hashing) algorithm takes care of distributing objects and uses rules to determine the mapping of the PGs to the OSDs.

# Distributed Object Storage





# CRUSH

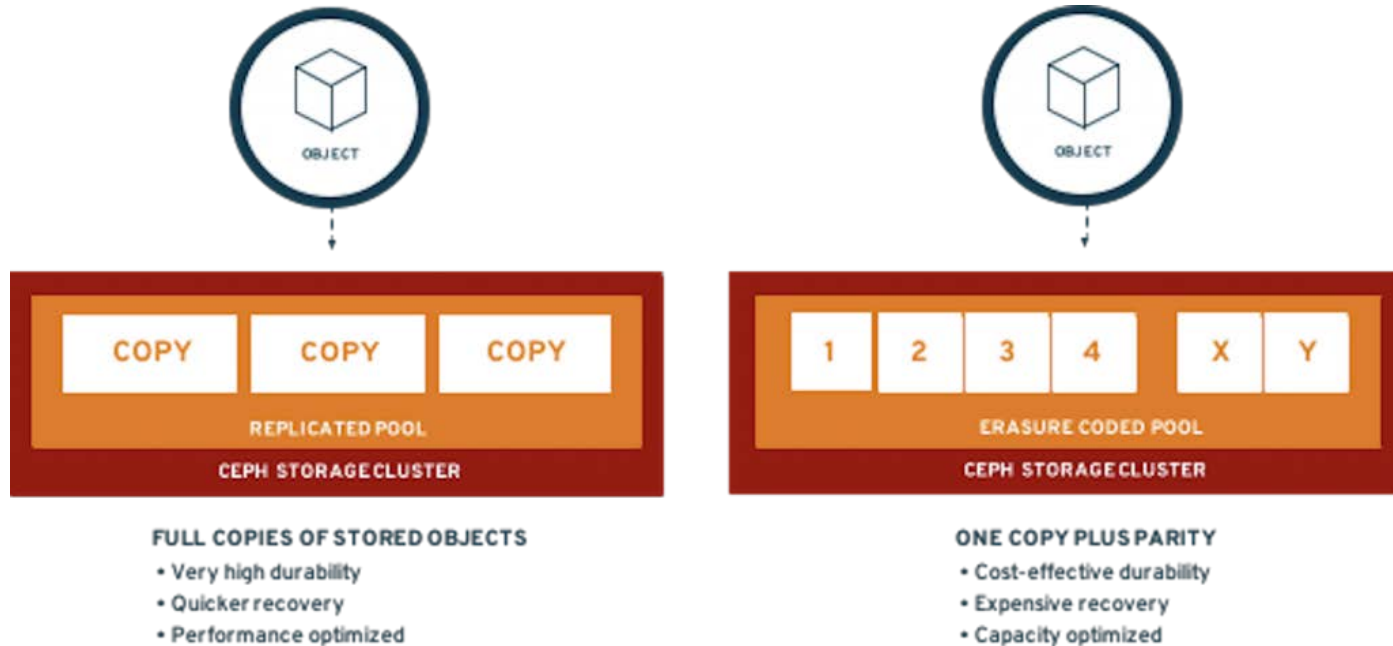
- CRUSH(x) -> (osdn1, osdn2, osdn3)
  - Inputs
    - x is the placement group
    - Hierarchical cluster map
    - Placement rules
  - Outputs a list of OSDs
- Advantages
  - Anyone can calculate object location
  - Cluster map infrequently updated

# Cluster partitions

- The CEPH cluster is separated into logical partitions, known as pools. Each pool has the following properties that can be adjusted:
  - An ID (immutable)
  - A name
  - A number of PGs to distribute the objects across the OSDs
  - A CRUSH rule to determine the mapping of the PGs for this pool
  - Parameters associated with the type of protection
    - Number of copies for replicated pools
    - K and M chunks for Erasure Coding

# Data protection

- Support two types: redundancy and erasure code

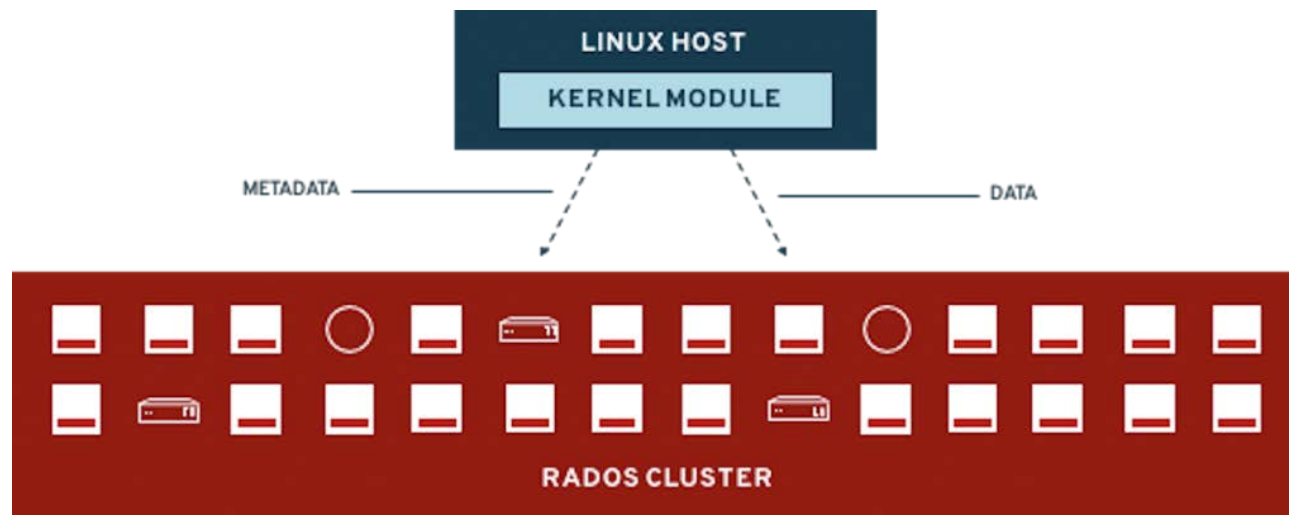


# Erasure code vs replication

- Replicated pools provide better performance in **almost** all cases at the cost of a lower usable to raw storage ratio (1 usable byte is stored using 3 bytes of raw storage by default)
- Erasure Coding provides a cost-efficient way to store data with less performance.
- Standard Erasure Coding profiles
  - 4+2 (1:1.666 ratio)
  - 8+3 (1:1.375 ratio)
  - 8+4 (1:1.666 ratio)

# CEPHfs

- clients access a shared POSIX compliant filesystem.



# Client access example:

1. Client sends *open* request to MDS
2. MDS returns capability, file inode, file size and stripe information
3. Client read/write directly from/to OSDs
4. MDS manages the capability
5. Client sends *close* request, relinquishes capability, provides details to MDS

# Synchronization

- Adheres to POSIX
- Includes HPC oriented extensions
  - Consistency / correctness by default
  - Optionally relax constraints via extensions
  - Extensions for both data and metadata
- Synchronous I/O used with multiple writers or mix of readers and writers

ORFEO storage



# 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

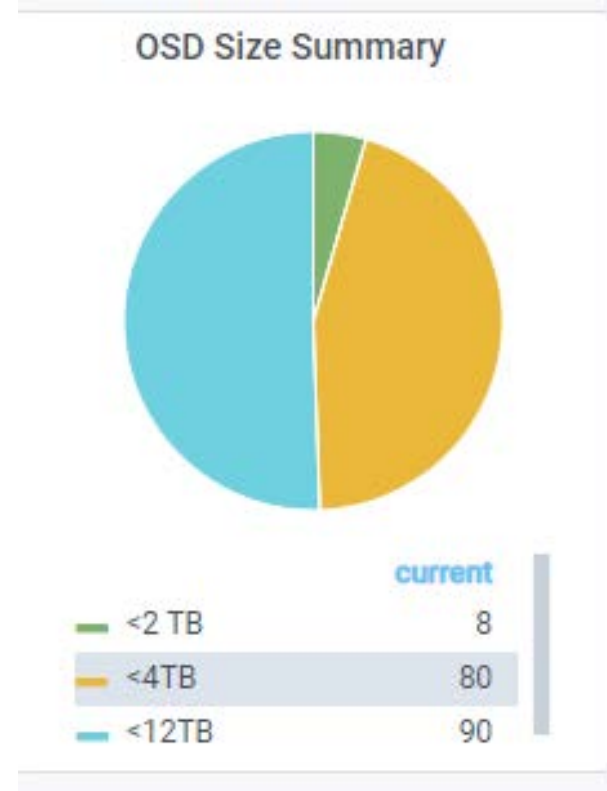
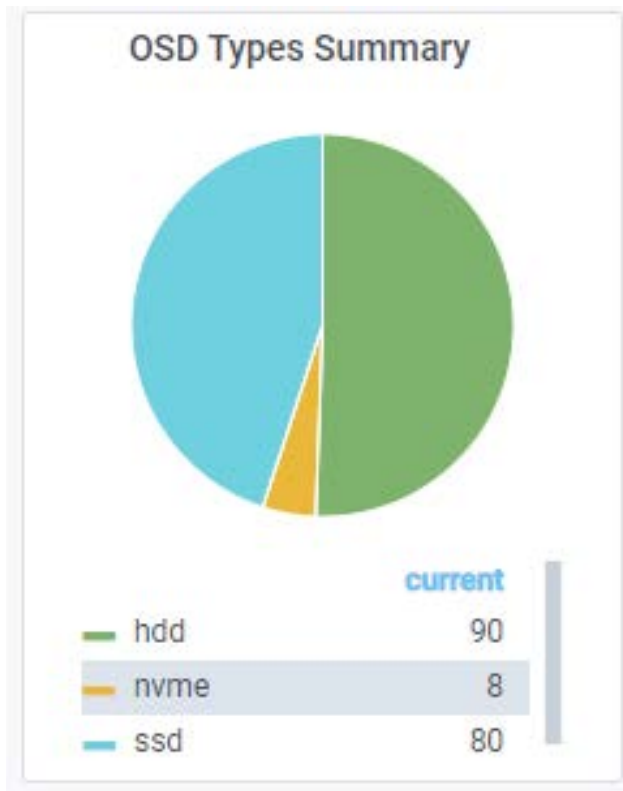
# ORFEO CEPH storage cluster

- 10 nodes

Hosts List	
Overall Performance	
<div> <input type="text" value="10"/> <input type="text" value="Search"/> </div>	
Hostname ↕	Services ⌵
ct1ps-ceph005	osd.63, osd.64, osd.65, osd.66, osd.67, osd.68, osd.69, osd.70, osd.71, osd.72, osd.73, osd.74, osd.75, osd.76, osd.77
ct1ps-ceph006	osd.78, osd.79, osd.80, osd.81, osd.82, osd.83, osd.84, osd.85, osd.86, osd.87, osd.88, osd.89, osd.90, osd.91, osd.92
ct1ps-ceph007	osd.100, osd.101, osd.102, osd.103, osd.104, osd.105, osd.106, osd.107, osd.93, osd.94, osd.95, osd.96, osd.97, osd.98, osd.99
ct1ps-ceph008	osd.108, osd.109, osd.110, osd.111, osd.112, osd.113, osd.114, osd.115, osd.116, osd.117, osd.118, osd.119, osd.120, osd.121, osd.122
ct1ps-ceph009	osd.148, osd.149, osd.150, osd.151, osd.152, osd.153, osd.154, osd.155, osd.156, osd.157, osd.158, osd.159, osd.160, osd.161, osd.162
ct1ps-ceph010	osd.163, osd.164, osd.165, osd.166, osd.167, osd.168, osd.169, osd.170, osd.171, osd.172, osd.173, osd.174, osd.175, osd.176, osd.177
ct1ps-ceph001	mds.ct1ps-ceph001, mgr.ct1ps-ceph001, mon.ct1ps-ceph001, osd.0, osd.1, osd.10, osd.11, osd.12, osd.124, osd.125, osd.126, osd.127, osd.128, osd.129, osd.13, osd.2, osd.3, osd.4, osd.5, osd.56, osd.6, osd.60, osd.7, osd.8, osd.9, rgw.ct1ps-ceph001
ct1ps-ceph002	mds.ct1ps-ceph002, mgr.ct1ps-ceph002, mon.ct1ps-ceph002, osd.130, osd.131, osd.132, osd.133, osd.134, osd.135, osd.14, osd.15, osd.16, osd.17, osd.18, osd.19, osd.20, osd.21, osd.22, osd.23, osd.24, osd.25, osd.26, osd.27, osd.57, osd.61, rgw.ct1ps-ceph002
ct1ps-ceph003	mds.ct1ps-ceph003, mgr.ct1ps-ceph003, mon.ct1ps-ceph003, osd.136, osd.137, osd.138, osd.139, osd.140, osd.141, osd.28, osd.29, osd.30, osd.31, osd.32, osd.33, osd.34, osd.35, osd.36, osd.37, osd.38, osd.39, osd.40, osd.41, osd.58, osd.62, rgw.ct1ps-ceph003
ct1ps-ceph004	mds.ct1ps-ceph004, mgr.ct1ps-ceph004, mon.ct1ps-ceph004, osd.123, osd.142, osd.143, osd.144, osd.145, osd.146, osd.147, osd.42, osd.43, osd.44, osd.45, osd.46, osd.47, osd.48, osd.49, osd.50, osd.51, osd.52, osd.53, osd.54, osd.55, osd.59, rgw.ct1ps-ceph004
0 selected / 10 total	

# ORFEO CEPH storage cluster

- 178 OSDs



# ORFEO CEPH Crush map

Cluster » CRUSH map

## CRUSH map viewer

- ▼ default (root)
  - ▶ ct1ps-ceph005 (host)
  - ▶ ct1ps-ceph006 (host)
  - ▶ ct1ps-ceph007 (host)
  - ▼ ct1ps-ceph001 (host)
    -  osd.0 (osd)
    -  osd.1 (osd)
    -  osd.10 (osd)
    -  osd.11 (osd)
    -  osd.12 (osd)
    -  osd.124 (osd)
    -  osd.125 (osd)
    -  osd.126 (osd)
    -  osd.127 (osd)
    -  osd.128 (osd)
    -  osd.129 (osd)
    -  osd.13 (osd)
    -  osd.2 (osd)
    -  osd.3 (osd)
    -  osd.4 (osd)
    -  osd.5 (osd)
    -  **osd.56 (osd)**

## osd.56 (osd)

crush_weight	1.454986572265625
depth	2
device_class	nvme
exists	1
id	56
primary_affinity	1
reweight	1
type_id	0

# ORFEO CEPH storage cluster

- 4 monitors:

## In Quorum

			<div><div><div></div></div></div>	<div>10</div>	<div><div></div></div>	<div></div>	<div></div>
Name	Rank	Public Address	Open Sessions				
ct1ps-ceph001	0	10.128.6.211:6789/0					
ct1ps-ceph002	1	10.128.6.212:6789/0					
ct1ps-ceph003	2	10.128.6.213:6789/0					
ct1ps-ceph004	3	10.128.6.214:6789/0					
4 total							

Not In Quorum

[illegible]

# ORFEO CEPH pools..

- 17 different pools

fast  
large

Pools List										Overall Performance	
+ Create											
Name	Type	Applications	PG Status	Replica Size	Erasure Coded Profile	Crush Ruleset	Usage	Read bytes	Write bytes		
cephfs_data	replicated	cephfs	2048 active+clean	3		repl_ssd	17%				
cephfs_metadata	replicated	cephfs	256 active+clean	3		repl_ssd	0%				
cephfs_spin_data	erasure	cephfs	2 active+clean+scrubbing+deep 1 active+clean+scrubbing, 1021 active+clean	6	ec_hdd_4km2	cephfs_spin_data	23%				
cephfs_spin_metadata	replicated	cephfs	128 active+clean	3		repl_ssd	0%				
kub_metadata	replicated	cephfs	8 active+clean	3		repl_nvme	0%				
kub_data	replicated	cephfs	32 active+clean	3		repl_ssd	1%				
os-images	replicated	rbd	64 active+clean	3		repl_ssd	0%				
os-volumes-ssd	replicated	rbd	1024 active+clean	3		repl_ssd	0%				
os-vms	replicated	rbd	32 active+clean	3		repl_ssd	0%				
os-backups	replicated	rbd	32 active+clean	3		repl_ssd	0%				
0 selected / 17 total											

# ORFEO /fast and /large from CEPH

- /large
  - 90 disks (12TB each) → 90 OSDs
  - Erasure code: 4+2 (1:1.666 ratio)
  - 1080 raw capacity → 648 useful size  
converti in Tib
- /fast
  - 80 disks (4TB each) → 80 OSDs
  - Replication: 3 copies each object
  - 320TB raw capacity →  $320/3 \approx 100$ TB  
useful size

# ORFEO CEPH file system

- 3 different file-system

Filesystems

fast

large

Name <a href="#">↗</a>	Created <a href="#">↕</a>	Enabled <a href="#">↕</a>
cephfs	2020-01-25 15:06:37.434728	true
cephfs_spin	2020-04-06 15:24:22.897778	true
kubefs	2020-05-12 16:34:13.390075	true

1 selected / 3 total

Details Clients: 19 Performance Details

## Ranks

Rank <a href="#">↗</a>	State <a href="#">↕</a>	Daemon <a href="#">↕</a>	Activity <a href="#">↕</a>	Dentries <a href="#">↕</a>	Inodes <a href="#">↕</a>
0	active	ct1ps-ceph003	Reqs: 5.2 /s	1.7 M	1.5 M

1 total

Standby daemons ct1ps-ceph004

## Pools

Pool <a href="#">↗</a>	Type <a href="#">↕</a>	Size <a href="#">↕</a>	Usage <a href="#">↕</a>
cephfs_spin_data	data	652.8 TiB	<div><div></div></div> 23%
cephfs_spin_metadata	metadata	81.5 TiB	<div><div></div></div> 0%

2 total



# ORFEO: long term storage

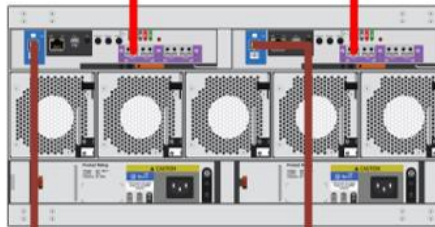
- A NAS for ORFEO Cluster
- Internally:
  - An entry level SAN

# ORFEO: long term storage

- Dell EMC PowerEdge R640

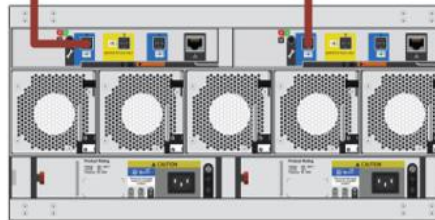


- Dell EMC PowerVault ME4084

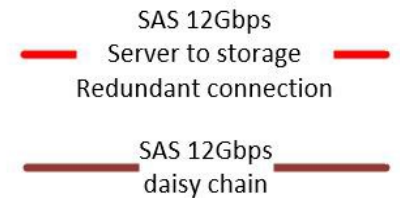


84 x 12TB HDD

- + Dell EMC PowerVault ME4084



42 x 12TB HDD



```
>df -h
10.128.2.231:/storage          37T   18T   20T   48% /storage
10.128.2.231:/illumina_run    46T   42T   4.2T   92% /illumina_run
```

Action

HOME

  
0 Host Groups

  
1 Hosts

  
2 Initiators

Ports A

  
A0 - SAS

  
A1 - SAS

  
A2 - SAS

  
A3 - SAS

0 IOPS  
0 MB/s

Ports B

  
B0 - SAS

  
B1 - SAS

  
B2 - SAS

  
B3 - SAS





Spares

0

0

0



## SYSTEM

Front Rear Table

Turn On LEDs Turn Off LEDs



## POOLS

<input type="text"/>	Clear Filters	Export to CSV	Show <b>All</b>	Showing 1 to 2 of 2 entries(1 selected)		
Name	Health	Size	Class	Avail	Volumes	Disk Groups
A	OK	356.9TB	Virtual	248.9TB	3	1
B	OK	112.7TB	Virtual	112.7TB	0	1

### Related Disk Groups

Clear Filters

Export to CSV

Show 

All

Showing 1 to 1 of 1 entries(1 selected)

◀

▶

Name	Health	Pool	RAID	Class	Disk Description	Size	Free	Current Job	Status	Disks
dgB01	OK	B	ADAPT	Virtual	SAS MDL	112.7TB	112.7TB	VRSC (58%)	FTOL	14

### Related Disks

<input type="text"/>	Clear Filters	Export to CSV	Show <b>All</b>	Showing 1 to 14 of 14 entries		
Location	Health	Description	Size	Usage	Disk Group	Status
0.21	OK	SAS MDL	11.7TB	VIRTUAL POOL	dgB01	Up
0.22	OK	SAS MDL	11.7TB	VIRTUAL POOL	dgB01	Up
0.23	OK	SAS MDL	11.7TB	VIRTUAL POOL	dgB01	Up
0.24	OK	SAS MDL	11.7TB	VIRTUAL POOL	dgB01	Up
0.25	OK	SAS MDL	11.7TB	VIRTUAL POOL	dgB01	Up
0.26	OK	SAS MDL	11.7TB	VIRTUAL POOL	dgB01	Up
0.27	OK	SAS MDL	11.7TB	VIRTUAL POOL	dgB01	Up
0.28	OK	SAS MDL	11.7TB	VIRTUAL POOL	dgB01	Up
0.29	OK	SAS MDL	11.7TB	VIRTUAL POOL	dgB01	Up
0.30	OK	SAS MDL	11.7TB	VIRTUAL POOL	dgB01	Up
0.31	OK	SAS MDL	11.7TB	VIRTUAL POOL	dgB01	Up
0.32	OK	SAS MDL	11.7TB	VIRTUAL POOL	dgB01	Up
0.33	OK	SAS MDL	11.7TB	VIRTUAL POOL	dgB01	Up
0.34	OK	SAS MDL	11.7TB	VIRTUAL POOL	dgB01	Up

Action

## VOLUMES

<input type="text"/>	<input type="button" value="Clear Filters"/>	<input type="button" value="Export to CSV"/>	Show <input type="text" value="10"/>	Showing 1 to 3 of 3 entries(1 selected)	<input type="button" value="◀"/>	<input type="button" value="▶"/>
Group	Name	Pool	Type	Size	Allocated	
-	illumina_run	A	base	50.4TB	45.9TB	
-	long_term_storage	A	base	109.9TB	40.5TB	
-	storage	A	base	39.9TB	21.5TB	

Snapshots

Maps

Replication Sets

Schedules

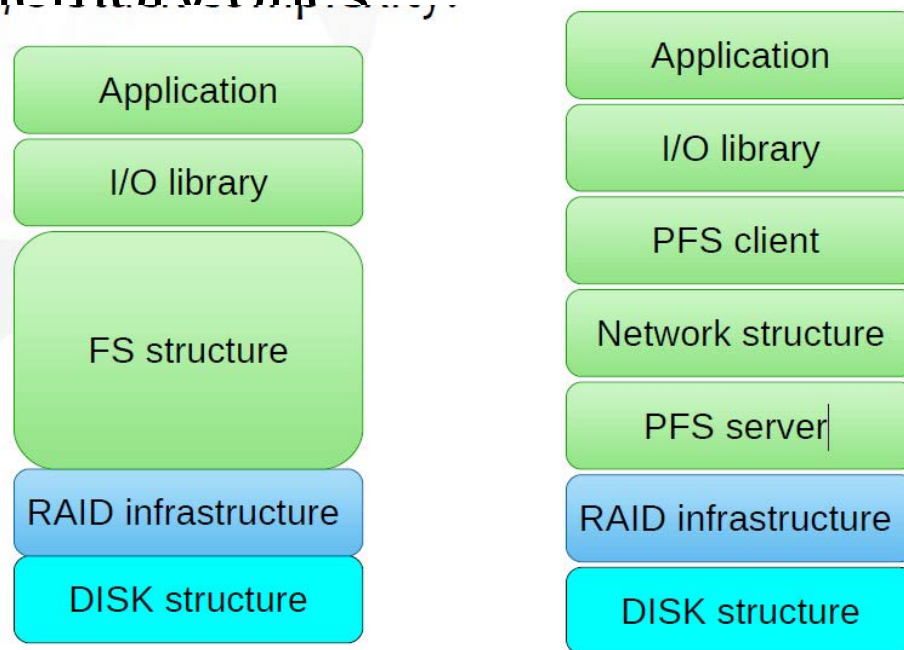
<input type="text"/>	<input type="button" value="Clear Filters"/>	<input type="button" value="Export to CSV"/>	Show <input type="text" value="10"/>	Showing 1 to 1 of 1 entries	<input type="button" value="◀"/>	<input type="button" value="▶"/>
Group.Host.Nickname	Volume	Access	LUN	Ports		
ceph9.*	storage	read-write	2	0,1,2,3		

```
[root@login bin]# df -h | grep storage
10.128.2.231:/storage          37T   18T   20T   48% /storage
```

# Benchmarking I/O on ORFEO

# I/O benchmarking...

- It is becoming more and more important
- I/O performance tends to be trickier than CPU/memory ones...





# How to test a complex I/O infrastructure ?

- Benchmark all the single component of the infrastructure
- Compare simple component Peak performance with measured numbers
- Combine all numbers together to get a performance model and some expected value
- Perform the high level benchmark and compare against what you evaluated.

# I/O microbenchmarks to play..

- Measures one fundamental operation in isolation
  - Read throughput, write throughput, creates/sec, etc.
- Good for:
  - Tuning a specific operation
  - Post-install system validation
  - Publishing a big number in a press release
- Not as good for:
  - Modeling & predicting real application performance
  - Measuring broad system performance characteristics
- Example to play
  - IOR: <https://github.com/hpc/ior>
  - iozone ([www.iozone.org](http://www.iozone.org))
  - Mdtest (included in the IOR )

# Estimate I/O performance of ORFEO storage..

- Peak performance estimate:
  - Network:
    - Infiniband Network from server toward clients: 12GB/sec
  - Disks:
    - HDD: 150 MB/sec (estimate)
    - SDD: 600 MB/sec (estimate)



/fast:  $80 \times 600 = 32$  GB/sec without replicas

/scratch:  $90 \times 150 = 13$  GB/sec without erasure code

# Measure performance of ORFEO storage..

- Acceptance tests:
  - /fast without replica with 56 disks:
    - ~ 20 GB/seconds

# iozone

- Compilation trivial: see tutorial.
- Things to try:
- Test to run:
  - `iozone -a` (basic testing)
  - Large file (large than memory to avoid caching effects)

```
iozone -i 1 -i 0 -s 32g -r 1M -f ./32gzero2
```

- Short introduction of basic flags:  
<http://www.thegeekstuff.com/2011/05/iozone-examples/>

# IOR: the de-facto I/O benchmark for HPC

## HPC IO Benchmark Repository build error

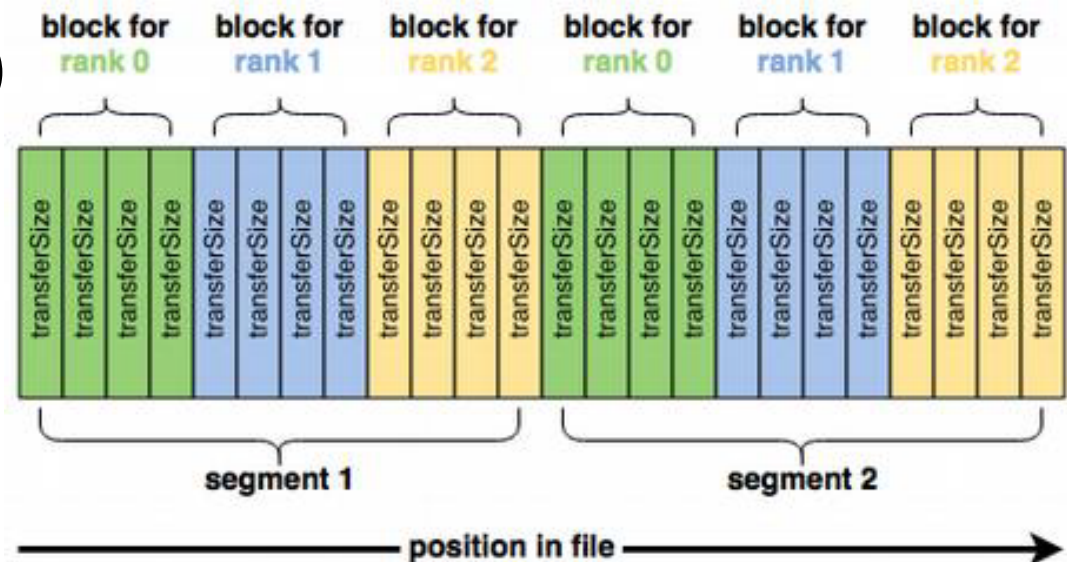
This repository contains the IOR and mdtest parallel I/O benchmarks. The [official IOR/mdtest documentation](#) can be found in the `docs/` subdirectory or on Read the Docs.

### Building

1. If `configure` is missing from the top level directory, you probably retrieved this code directly from the repository. Run `./bootstrap` to generate the configure script. Alternatively, download an [official IOR release](#) which includes the configure script.
2. Run `./configure`. For a full list of configuration options, use `./configure --help`.
3. Run `make`
4. Optionally, run `make install`. The installation prefix can be changed via `./configure --prefix=...`.

# IOR basic usage:

- IOR writes data sequentially with the following parameters:
  - blockSize (-b)
  - transferSize (-t)
  - segmentCount
  - numTasks (-n)



# IOR number to collect..

- Compare performance of HDF5 vs MPIIO vs POSIX..
- Possible experiments:
  - `mpirun -np 32 IOR -a [POSIX|MPIO] -i 3 -d 32 -k -r -E -o yourfile_name -s 1 -b 60G -t 1m`
  - `mpirun -np 32 IOR -a [POSIX|MPIO] -i 3 -d 32 -k -r -E -o yourfile_name -s 1 -b 16G -t 1m`
  - `mpirun -np 32 IOR -a [POSIX|MPIO] -i 3 -d 32 -k -r -E -o yourfile_name -s 1 -b 4G -t 1m`



# MD test

- How much does it cost metadata operations ?
- Example to run:

```
mdtest -n 10 -i 200 -y -N 10 -t -u -d $test_directory
```

- -n: every process will creat/stat/remove # directories and files
- -i: number of iterations the test will run
- -y: sync file after writing
- -N: stride # between neighbor tasks for file/dir stat (local=0)
- -t: time unique working directory overhead
- -u: unique working directory for each task
- -d: the directory in which the tests will run

The end