Gilberto Díaz

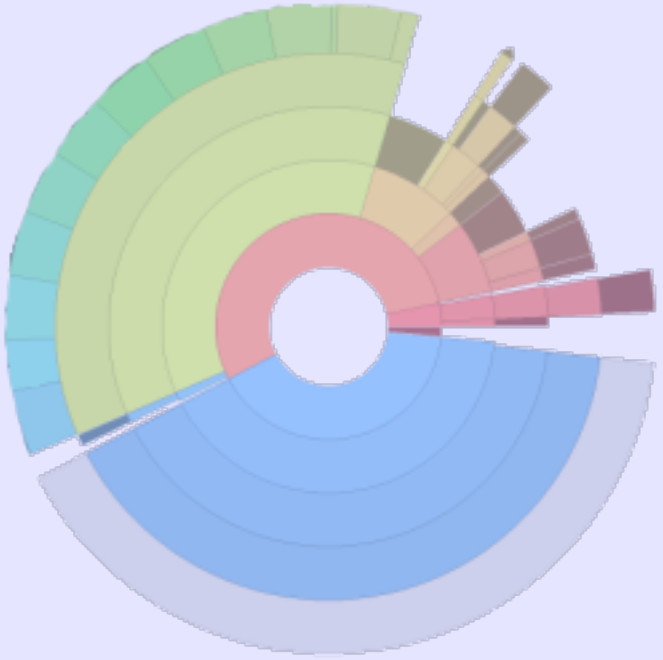
ULA
Merida, VENEZUELA

Moreno Baricevic
Stefano Cozzini

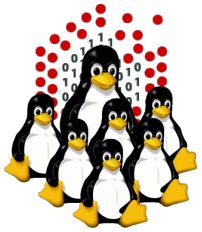
CNR-IOM DEMOCRITOS
Trieste, ITALY

Axel Kohlmeyer

ICTP
Trieste, ITALY



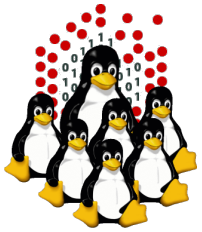
Introduction to storage and filesystems



Introduction

Many applications perform relatively simple operations on vast amounts of data. In such cases, the performance of a computer's data storage devices impact overall application performance more than processor performance.

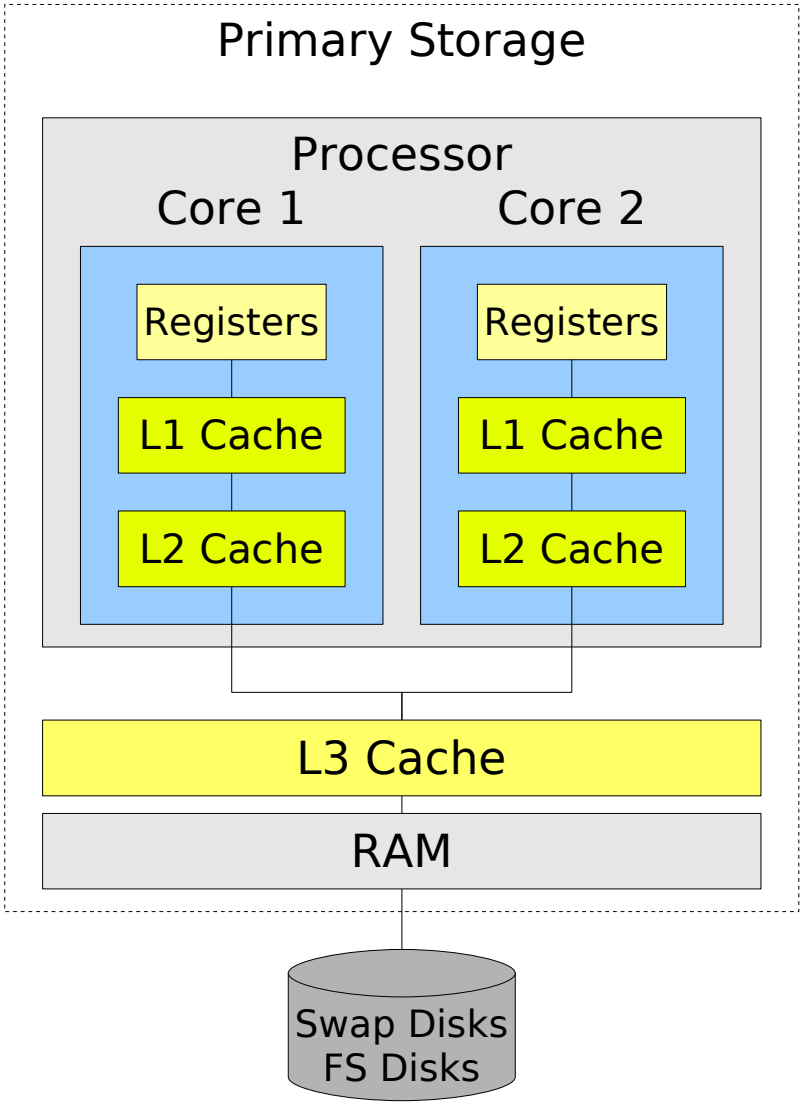
- HPC workflow will be soon bounded by the speed of the storage system.
- *You can only compute it as fast as you can move it.*



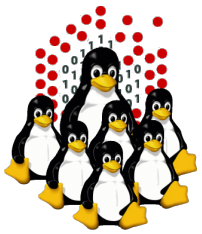
Memory Hierarchy

Computers architectures try to keep data close to the processors in order to feed them continuously.

However, while the capacity of storage devices increases, the distance to the processors also increases.



Internal Memory - processor registers and cache
Main Memory - system RAM and controller cards
On-line mass storage - secondary storage
Off-line bulk storage - tertiary and off-line storage



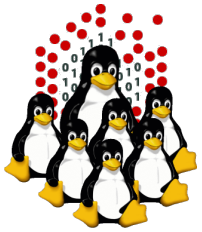
Storage Hierarchy

Same as with the memory hierarchy of

Register -> Cache (L1->L2->L3) -> RAM

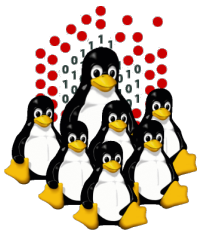
storage follows a hierarchy with multiple levels:

- 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, Lustre, GPFS, ...)
- Tape based archival system (often with disk cache)
- External, distributed file systems (Cloud storage)



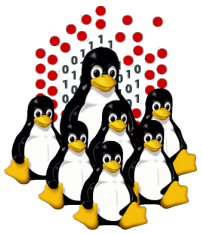
Cache / Swap

- Disk I/O is much slower than main memory I/O, typically about a 100x (varied with hardware):
  - typically applications use buffers (libc/stdio)
- In typical workloads certain data is accessed repeatedly **beyond** an application lifetimes:
  - OS maintains buffer of recently used data
  - buffer competes with applications for RAM
  - OS can substitute swap disk for RAM
- Memory management unit (MMU) organizes address space in pages (RO, RW, COW)

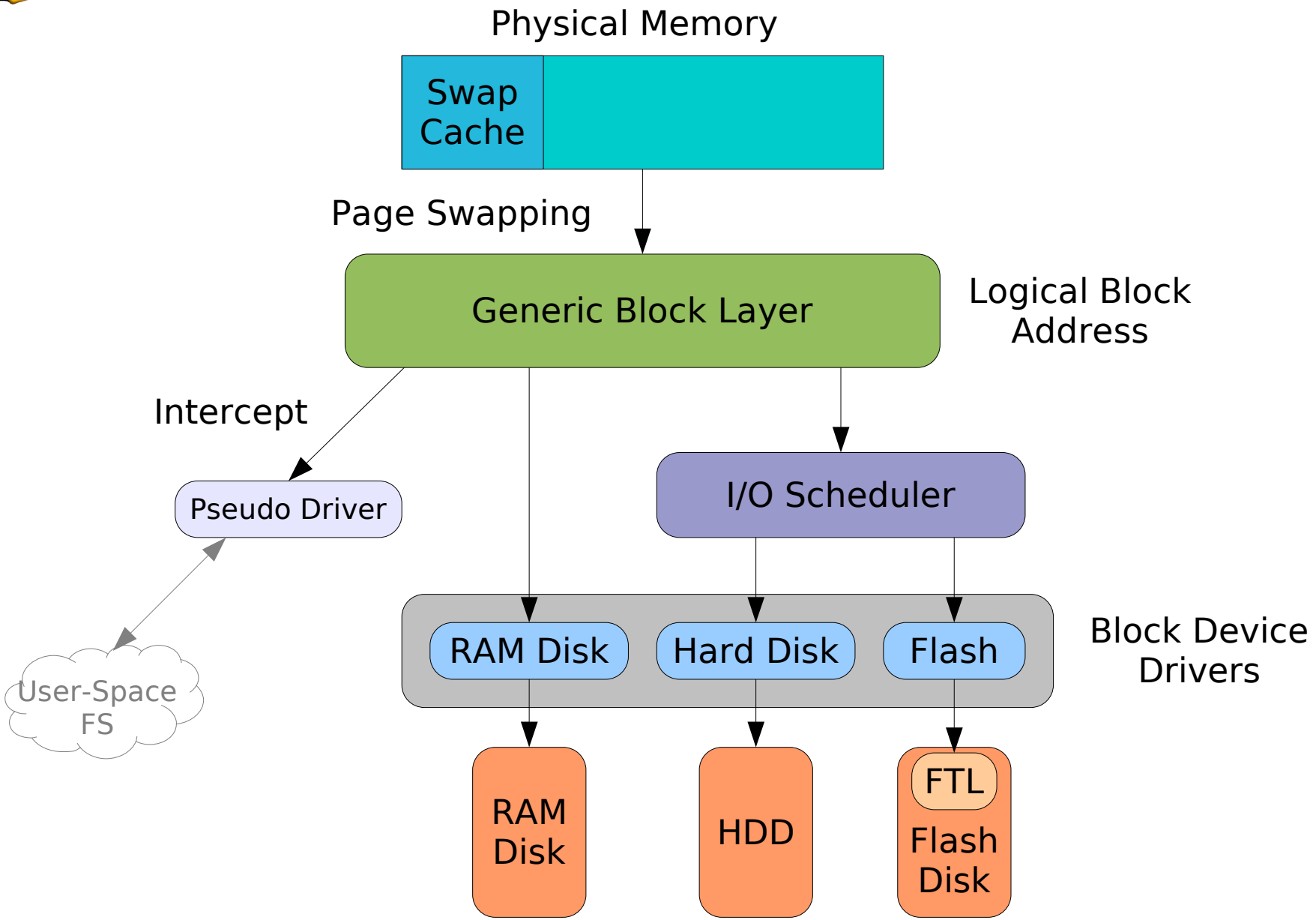


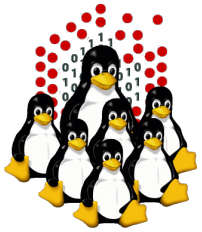
RAM Disk / Solid State Drive

- 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)
- Solid state drive is a **non**-volatile RAM disk
  - uses same interface as (spinning) hard drive
    - Battery buffered DRAM (fast, no wear, expensive)
    - Flash based (varied speed, wears out, varied cost)



Storage Hierarchy





Storage Hierarchy
About size, bandwidth and latency

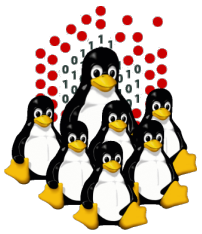
Processor Registers	• 1 CPU cycle • few KB (x core)
Cache L1	• ~5 CPU cycles • <= 128KB (x core) • 700 GiB/s
Cache L2	• ~10 CPU cycle • <= 2MB (x core) • 200 GiB/s
Cache L3	• <100 CPU cycles • <= 8MB (x numanode/socket) • 100 GB/s
RAM	• <300 CPU cycles • <= few GB typical, up to 2TB (x machine) • 10GB/s
FLASH	• <= 128GB (x device) • bw is bus dependent
SSD	• <= 800GB (x disk) • <= 700MB/s (PCIe)
HDD	• <= 4TB (x disk) (+ <=64MB cache) • <= 200MB/s • 4~48 disks x machine, more x storage appliances
TAPE	• <= 8TB (x cartridge), ~40TB soon • ~160MB/s • PB/EB x archive libraries (robots)

→ Hardware

→ Kernel
→ Optimizing compilers
→ Programmers
(Assembly, C registers)

→ Programmers

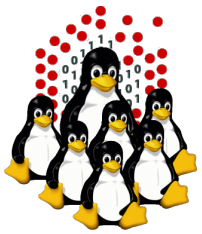
>1.000.000 CPU cycles
the CPU spends much of its time **idling, waiting for memory I/O to complete**



Current Mass Storage Devices

We are interested particularly in low cost storage devices with big capacity and high performance.

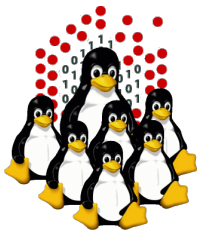
Nowadays, magnetic hard disk drives still are the technology which include all these features.



Current Hard Disk Drives Technologies

We can find several magnetic hard disk technologies today:

- Serial Advanced Technology Attachment (SATA)
- Serial Attached SCSI (SAS)
- Advanced Technology Attachment ([P]ATA/[E]IDE) (obsoleted by SATA)
- Small Computer System Interface (SCSI) (obsoleted by SAS)



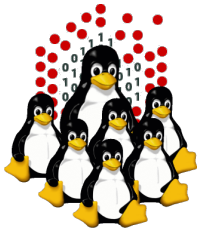
Rising Hard Disk Drives Technologies Solid-State Drive (SSD)

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, FC buses

cons:

- extremely expensive, low capacity; usage limited to special purposes only (hardly used for 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

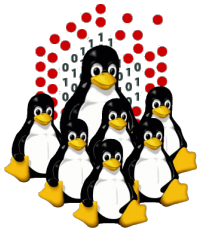


Performance vs Capacity vs Price

Today disk space is cheap, a single (SATA) disk drives provides up to 4TB (SSD still limited below 1TB for 10 times the price of SATA counterparts).

However, performance is another story. Fastest hard disk drive bandwidth is around 6Gbps (SAS 600, SATA3), with real-life speed that spans roughly from 100 to 200MB/s.

Up to 700MB/s for enterprise-level SSD over PCIe 8x bus, around 160MB/s for cheap ones over SATA bus.

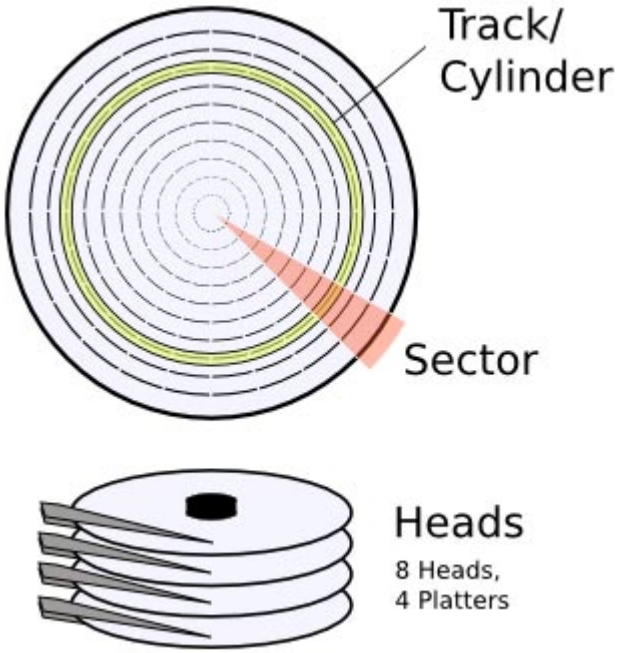


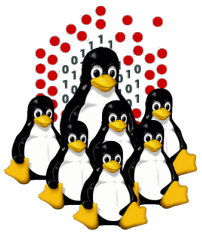
HDD components

A typical HDD includes a plurality of **magnetic disks** spun by a **spindle motor**.

Read/Write **heads** supported by the **slider suspension** assembly which are moved by some actuators in radial direction.

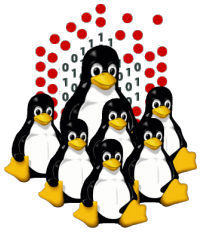
We can identify, on each **plate** (usually two or more, **two sided**), specific zones: cylinders and sectors. The data are stored on the disk in thin, concentric bands, each **cylinder** correspond to a single head position on the disk. A **sector** is the smallest physical storage unit on the disk. The data size of a sector is always a power of two (used to be 512 bytes, it's now 4k on the new TB hard-disks).





Local Spinning Disk Storage

- Data stored in concentric circles on fast rotating (3-15K RPM) metal plates with magnetic coating
- Increased capacity through stacking of plates
- Lower cost per capacity than RAM or Flash
- Read-write head positioned over track, wait until over requested sector(s) and read data
  - random data access incurs latency
  - wait time depends on rotation speed
- Susceptible to mechanical failures (head crash)

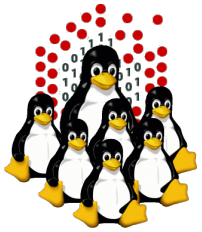


Redundant Array of Independent Disks (RAID)

One way to improve the bandwidth and overcome the limitation of a single mechanic is to define a logical device which consists of multiple disks.

With this sort of approach a single I/O transaction can simultaneously move blocks of data to multiple disks.

For example, if a logical device is created from eight disks, each of which is capable of sustaining 100 MB/sec, then this logical device is capable of delivering up to 800 MB/sec of I/O bandwidth.



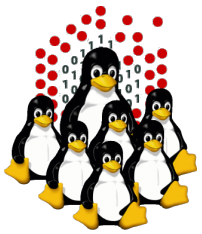
Redundant Array of Independent Disks (RAID)

Reliability or performance (or both) can be increased using different RAID “levels”.

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).

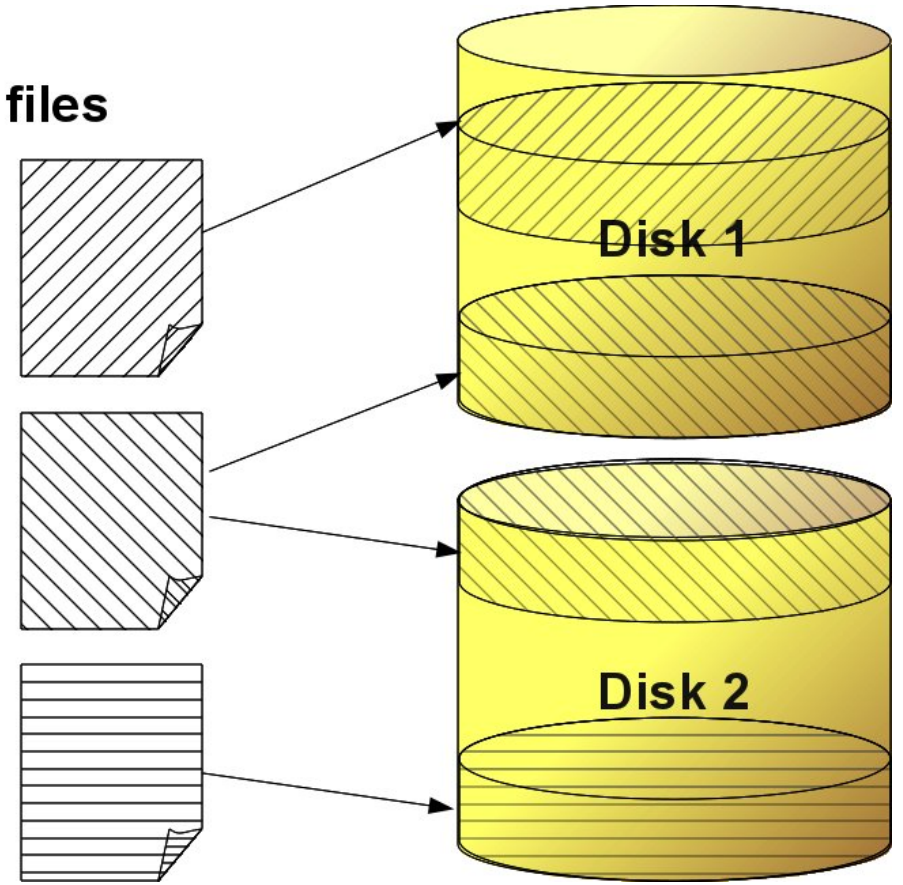


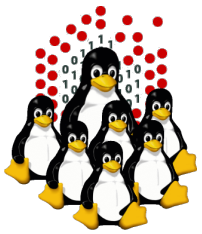
LINEAR RAID

Performance = P

NO REDUNDANCY

Capacity = N * S



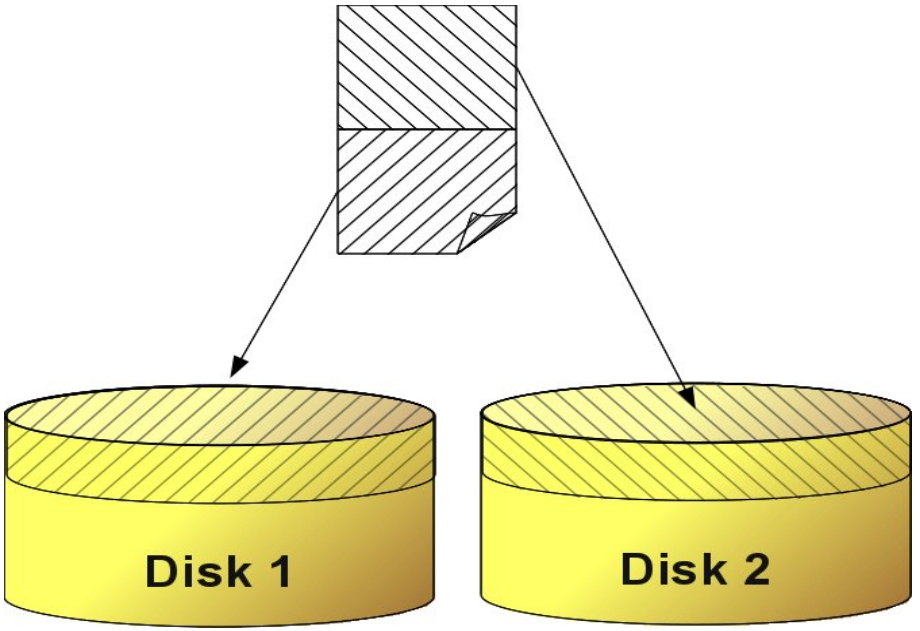


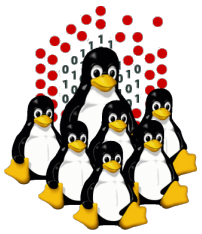
RAID 0

Performance = P * N

STRIPING

Capacity = N * S





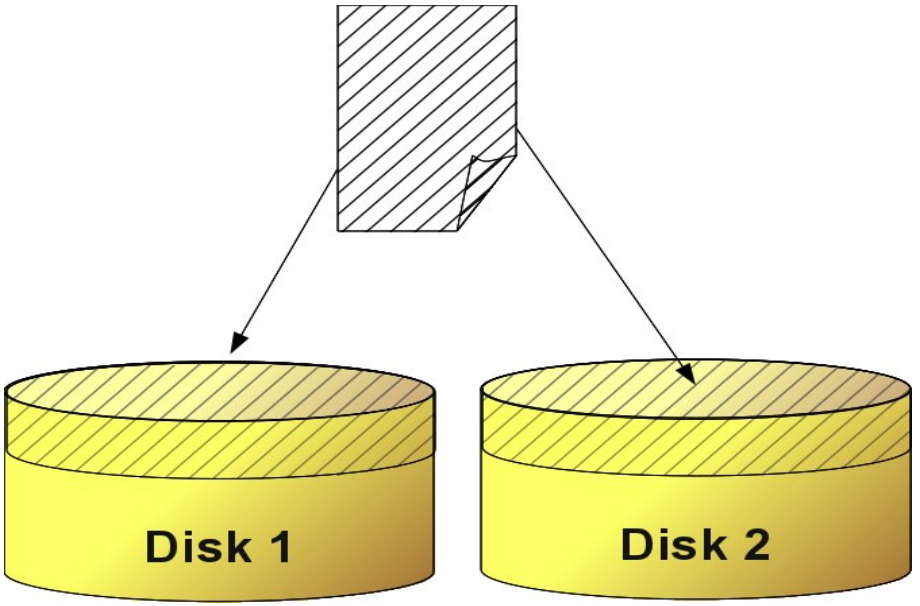
RAID 1

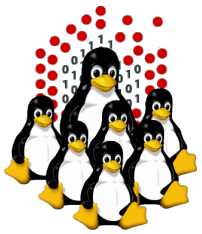
Write Perf. = P

Read Perf. = P * N

REDUNDANCY

Capacity = S

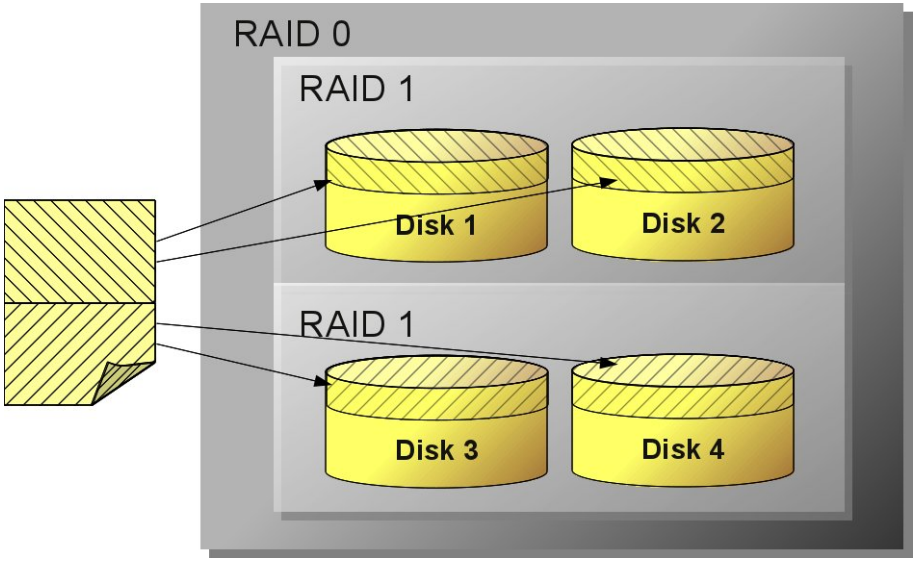




Nested RAID levels
RAID 10 / RAID 1+0 and RAID 0+1

REDUNDANCY

STRIPING

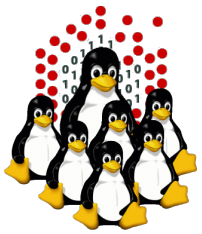


Raid 1+0 / 10: mirrored sets in a striped set

the array can sustain multiple drive losses so long as no mirror loses all its drives

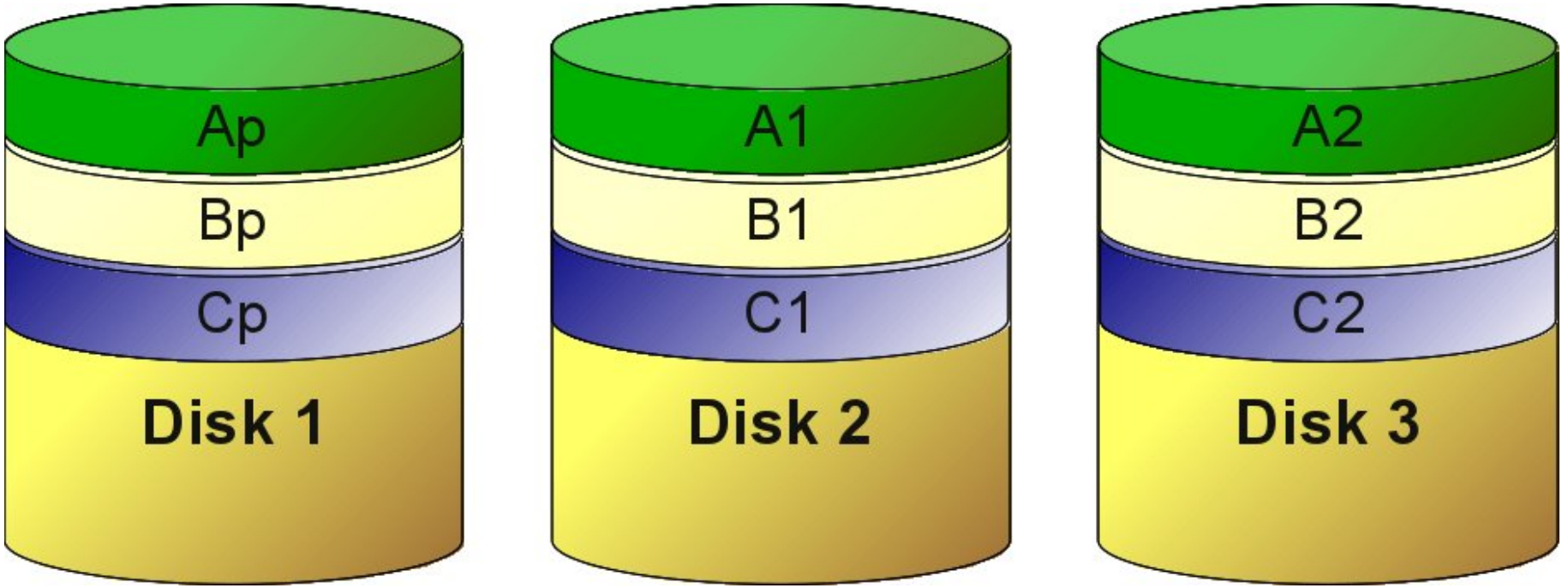
Raid 0+1: striped sets in a mirrored set

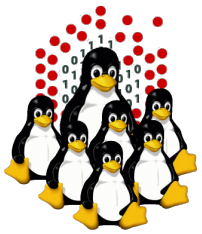
if drives fail on both sides of the mirror the data on the RAID system is lost



RAID 4

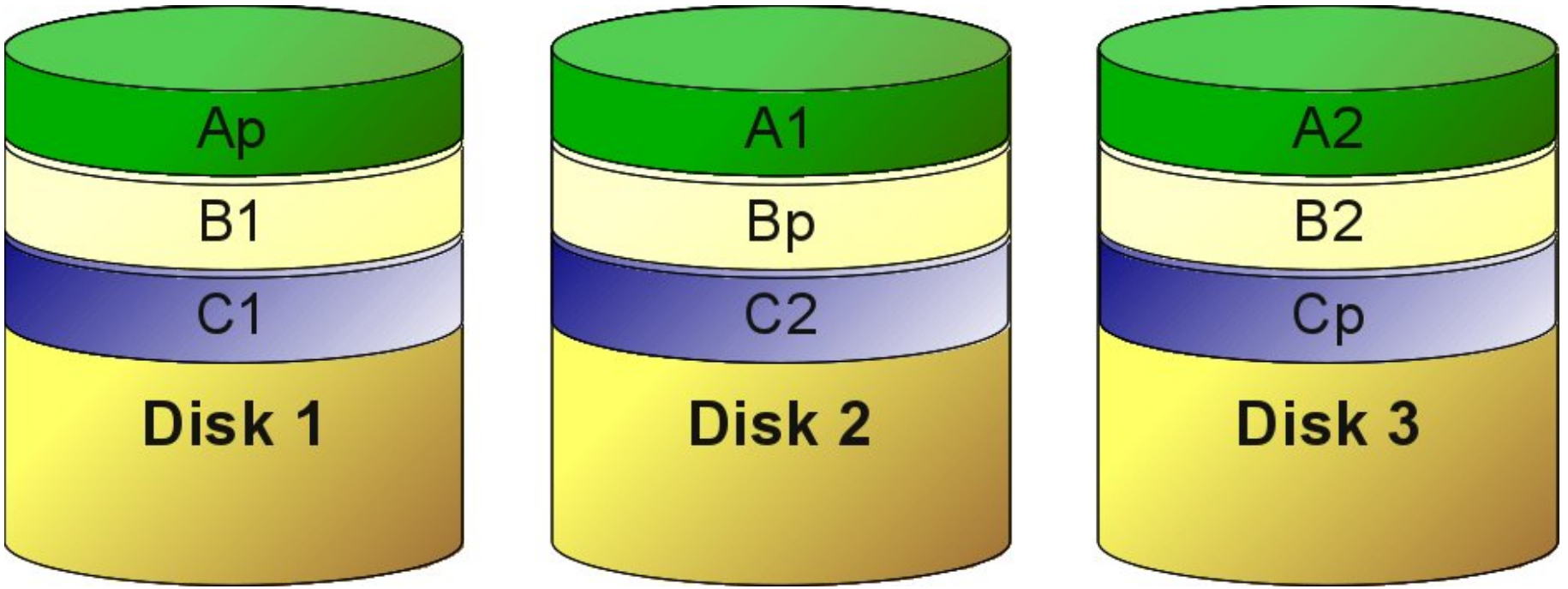
Parity Disk
Bottleneck

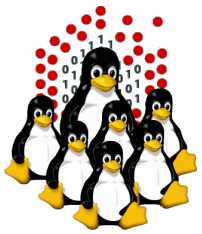




RAID 5

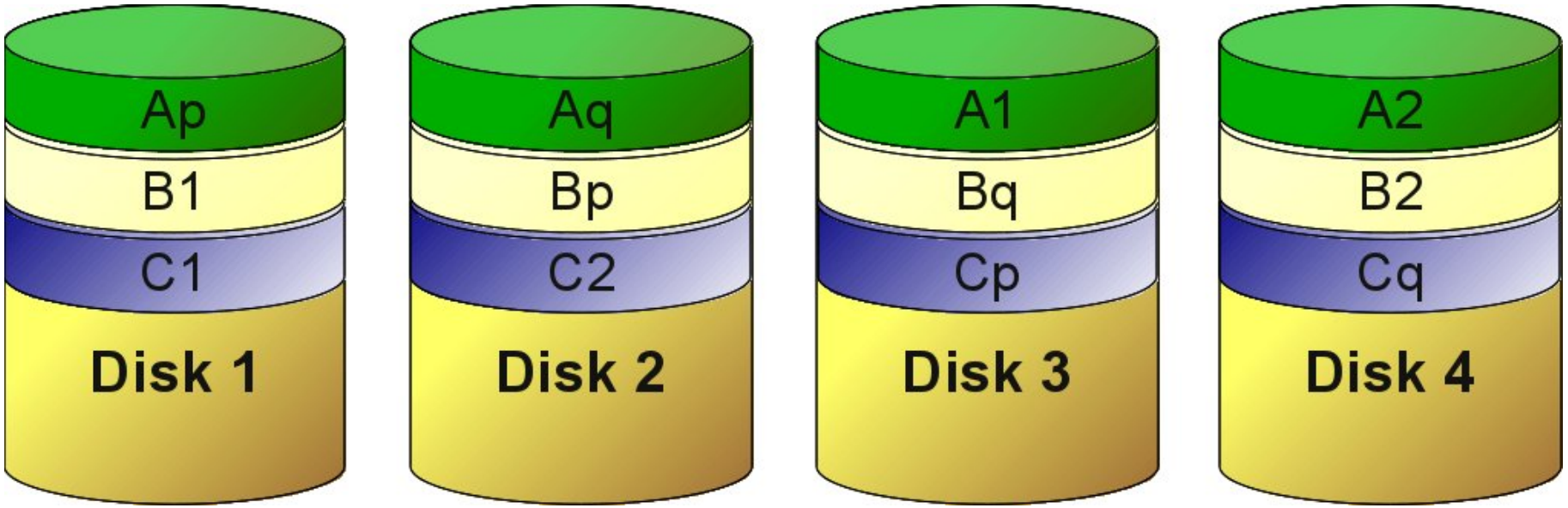
One disk can fail
Distributed parity

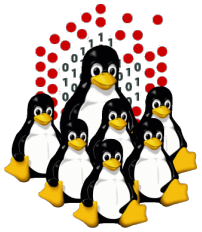




RAID 6

Two disks can fail
Double distributed parity code



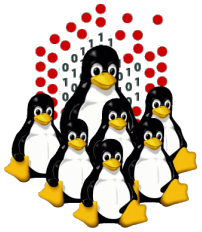


Notes on redundancy

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 on busy systems or large disks/arrays).

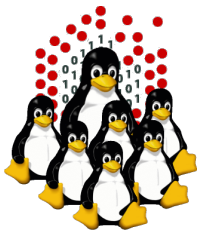


Hot-spare

Both hardware and software RAIDs with redundancy may support the use of a hot spare drive, a drive physically installed in the array which is inactive until an active drive fails, when the system automatically replaces the failed drive with the spare, rebuilding the array with the spare drive included. A hot spare can be shared by multiple RAID sets.

Subsequent additional failure(s) in the same RAID redundancy group before the array is fully rebuilt can cause data loss.

RAID 6 without a spare uses the same number of drives as RAID 5 with a hot spare and protects data against failure of up to two drives, but requires a more advanced RAID controller and may not perform as well.

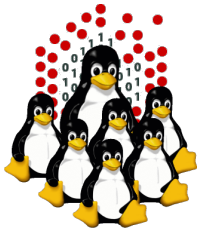


RAID Parameters

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

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

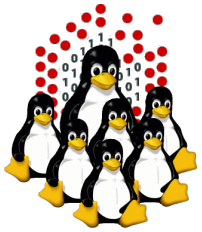


Software RAID

Several Operating Systems provide software RAID. Linux Software RAID devices are implemented through the *md* (Multiple Devices) device driver.

Currently, Linux supports:

- LINEAR md devices
- RAID0 (*striping*)
- RAID1 (*mirroring*)
- RAID4
- RAID5
- RAID6
- RAID10
- MULTIPATH
- FAULTY

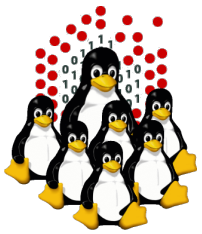


Logical Volume Manager

LVM is a software layer on top of the hard disks and partitions, which creates an illusion of continuity and ease-of-use for managing hard-drive replacement, repartitioning, and backup.

LVM is suitable for creating single logical volumes of multiple physical volumes or entire hard disks (somewhat similar to RAID 0, but more similar to JBOD*), allowing for dynamic volume resizing.

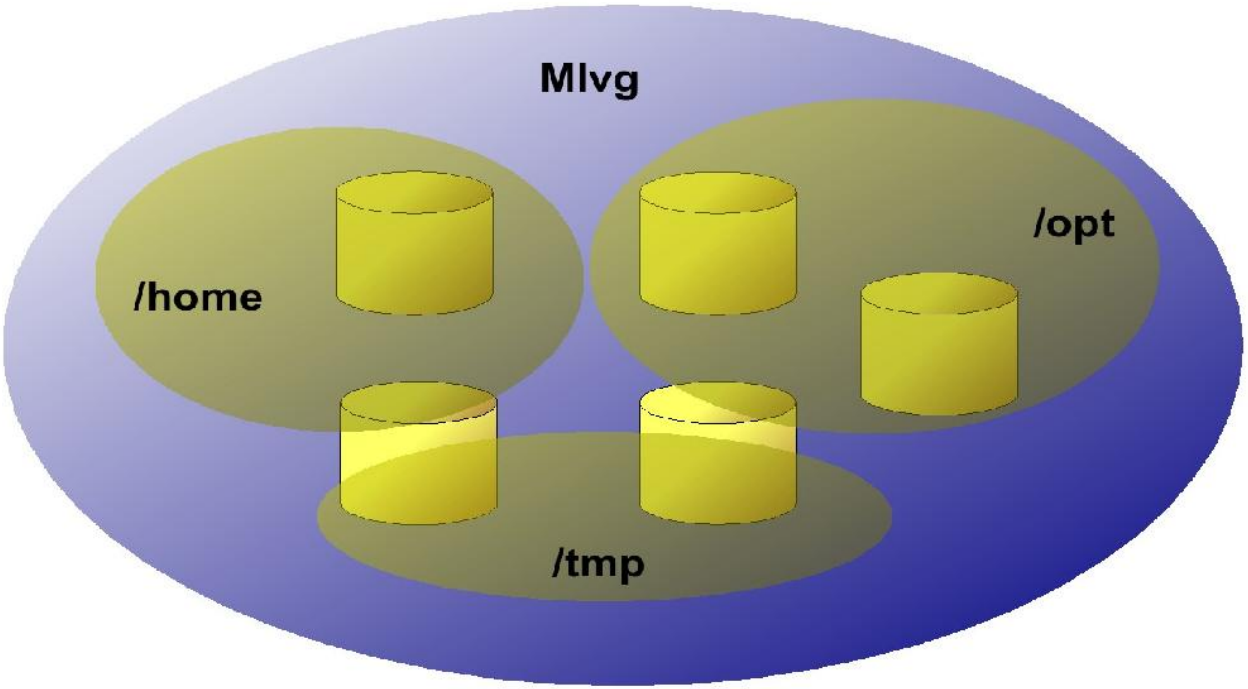
(*) JBOD: *Just a Bunch Of Disks*; an array of drives, each of which is accessed directly as an independent drive.

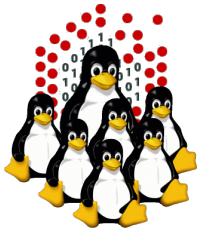


Logical Volume Manager

From physical devices we can create:

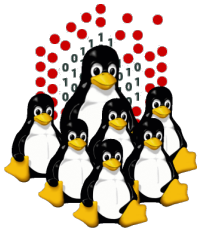
- Volume groups (Mlvg)
- Logical volumes (logical partitions):
  - /home
  - /opt
  - /tmp





File System

A file system is a set of methods and data structures used to organize, store, retrieve and manage information in a permanent storage medium, such as a hard disk. Its main purpose is to represent and organize resources storage.



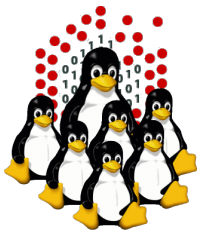
File System: elements

Name space: is a way to assign names to the items stored and organize them hierarchically.

API: is a set of calls that allow the manipulation of stored items.

Security Model: is a scheme to protect, hide and share data.

Implementation: is the code that couples the logical model to the storage medium.



File Systems: Basic Concepts (1/2)

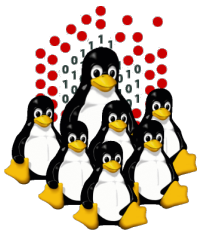
Disk: A permanent storage medium of a certain size.

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.

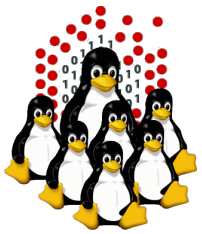


File Systems: Basic Concepts (2/2)

Metadata: A general term referring to information that is about something but not directly part of it. For example, the size of a file is very important information about a file, but it is not part of the data in the file. Ownerships, access permissions, creation/modification/access time, are also part of the metadata information pertaining a file.

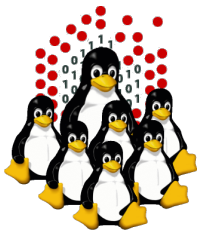
Journaling: A method of insuring the correctness of file system metadata even in the presence of power failures or unexpected reboots (atomic write).

Attribute: A name and value associated with the name. The value may have a defined type (string, integer, etc.).



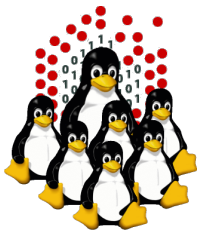
Modern File System Features

- *Journaling*: write data to journal, commit to file system when complete in *atomic* operation
  - reduces risk of corruption and inconsistency
- Faster file lookups through *balanced tree*
- *Snapshot*: retain status of file system at given point in time by copying metadata and marking object data referred as *copy-on-write*
- *Deduplication*: identify identical storage objects, consolidate and mark them *copy-on-write*



(Local) File Systems: few examples

- FAT
- NTFS
- ext2, ext3, ext4
- Reiserfs
- xfs
- jfs
- ...

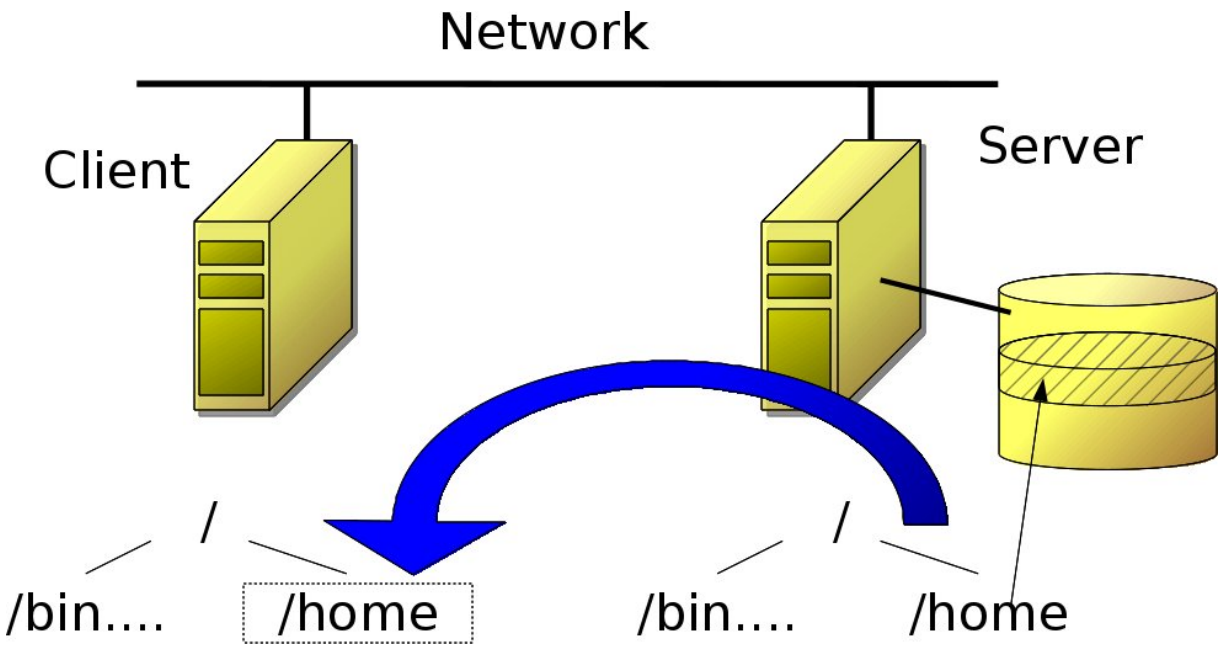


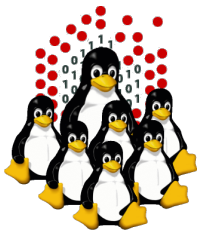
Distributed File Systems

A distributed file system makes the data available across the network.

Some examples:

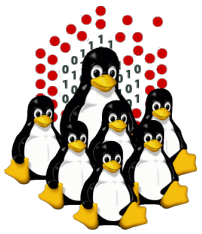
- NFS
- coda
- AFS
- ...
- samba/CIFS





Network Attached Storage (NAS)

- Centralized storage provisioning / management
- Emulate local file system or local block device
- Use specialized appliance or general server
  - (SMB/CIFS -> Windows, NFS -> Unix/Linux)
- Multiple clients access same device / storage
  - conflicts from concurrent access
- Performance limited through network
  - consistency -> limited write performance
  - client and server side caching possible



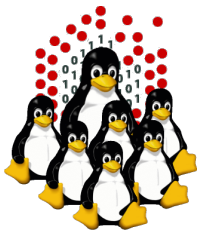
Distributed File Systems (Network File Systems)

Centralized

- Principle: one server contains all data, multiple clients
- Advantages: Simple, easy to setup, only one machine required, fast if few clients
- Drawbacks: not efficient, concurrency problems, slow if many clients
- Example: NFS

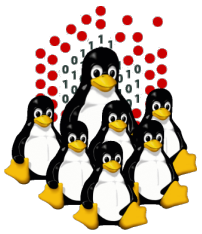
Parallel

- Principle: several data servers, (opt: metadata server), multiple clients, data is striped.
- Advantages: very efficient, configure striping, fast
- Drawbacks: setup more complicated, several machines required, metadata inefficiency at large scale
- Example: Lustre, GPFS



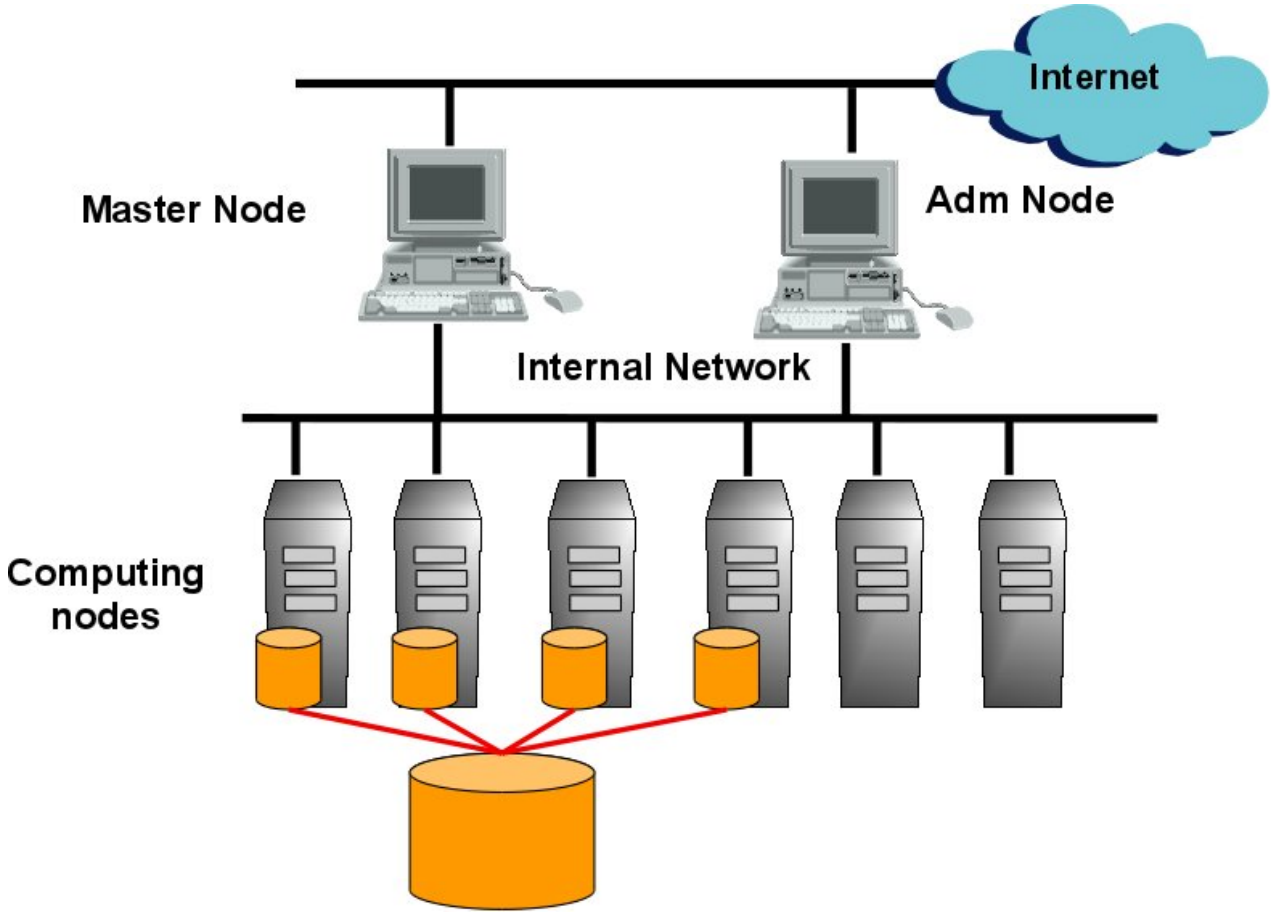
Parallel File Systems

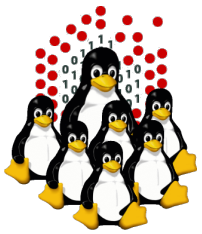
- Often combined with large HPC deployments
- Data distributed across multiple servers
- Multiple servers are accessed in parallel
- *Lustre*: filename based; dedicated metadata and storage servers on regular file system
- *GPFS*: block based; metadata and object storage distributed, based local RAID storage
- *HDFS*: block based; failure protection built in; improved performance through redundancy



Parallel File Systems

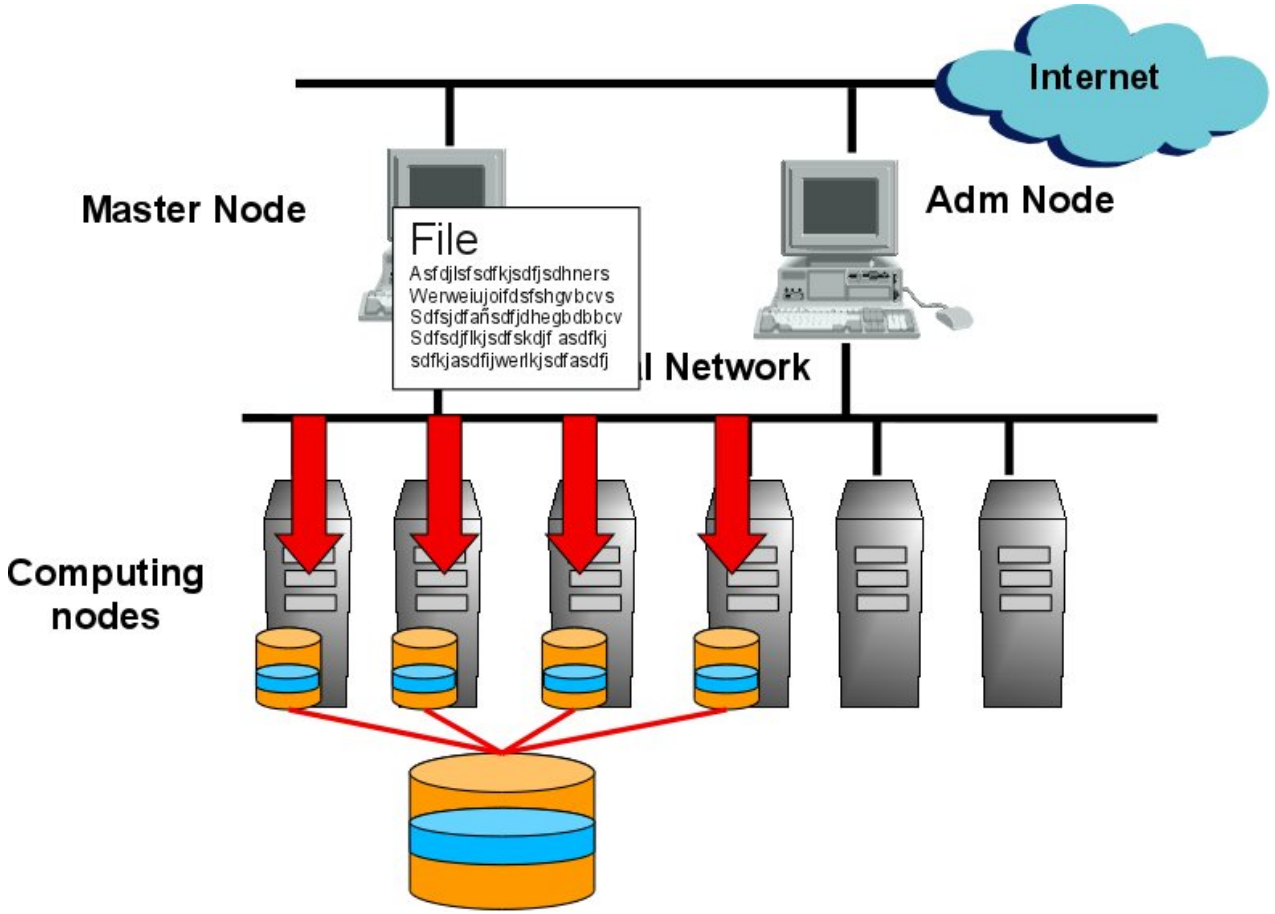
A parallel file system leverages all disks available across the network.

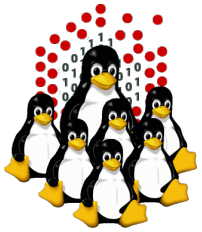




Parallel File Systems

A parallel file uses all I/O controllers at the same time.





Parallel File Systems

We can find several implementations of Parallel File Systems:

Lustre from Cluster File System, then from Sun, then from Oracle, now from WhamCloud (acquired by INTEL) (GPL)

GPFS from IBM (Private)

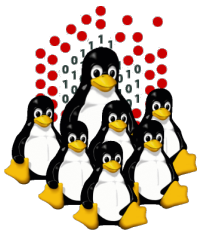
PVFS from ANL (GPL)

GlusterFS from Gluster Inc. (GPL)

PanFS from Panasas (Private)

IBRIX from Ibrix (Private)

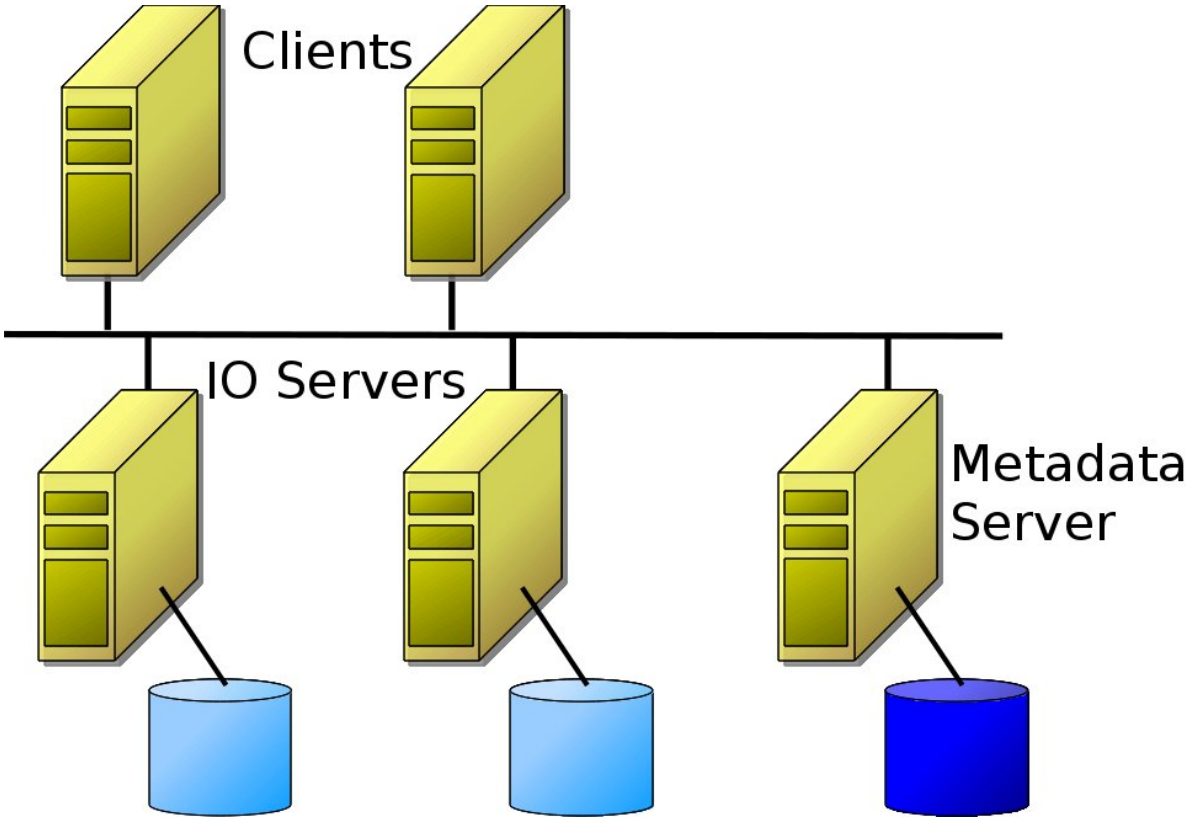
...

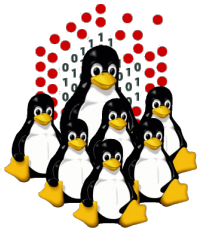


Parallel File System: components

In general, a Parallel File Systems has the following components

- Metadata Server
- I/O Servers
- Clients





Parallel I/O

How to write on disk?

- a file for each process
- shared file + offset

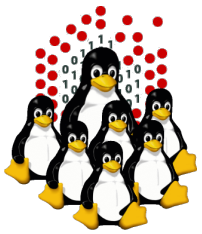
Libraries

- Parallel format HDF5, NetCFD
- MPI I/O (ROMIO, etc.)

Benchmarks

- Spec Suite ($)
- FileBench (Go- for Linux)
- NAS BTIO
- IOZone
- IOR
- Many others...

  (even some real applications: DEUS (Dark Energy Universe Simulations))

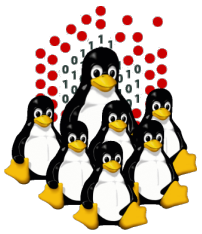


Tape Based Storage

- High density, high capacity, high latency storage
- (Still) best long-term reliability of all storage media (better than CD/DVD! much better than hard drives)
- Typical deployment as hierarchical storage management system (HSM):

  disk array (as cache) + tape robot(s)

  – data is moved to/from tape based on usage pattern
  – transparent integration into centralized storage
- Data transfer **rate** of "tape via courier" unbeatable
- Disk based alternatives have to assume failures and limited lifetime and depend on continuous replication

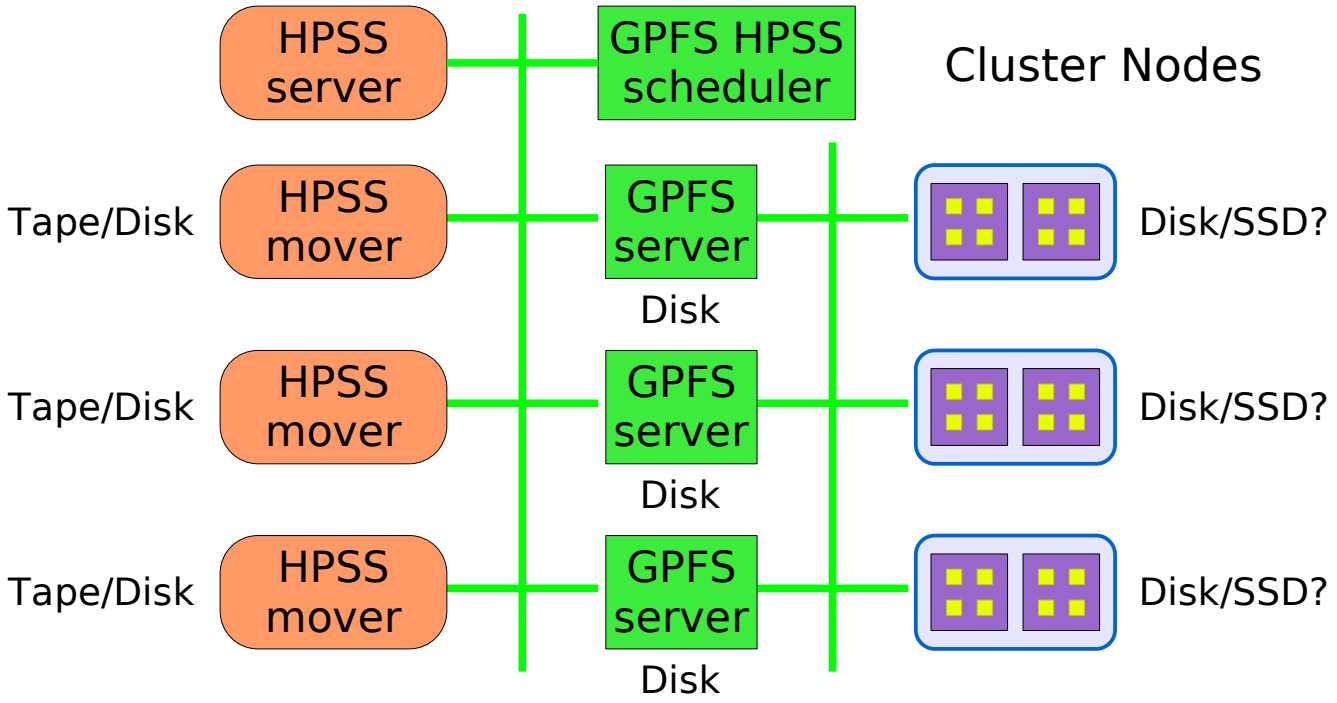


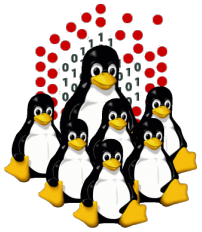
Hierarchical Storage Management

Archiving Storage

Hierarchical storage management (HSM) provides mechanisms to automatically migrate and recall files from disk to tape

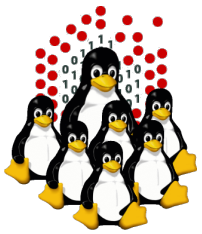
- Principle: store less frequently used data on tape. Archiving only.
- Advantages: Gb is not expensive, huge amounts of data, low power consumption
- Drawbacks: specific machine and tape-robot/silo required, very slow (order of sec.)
- Example: High Performance Storage System (HPSS)
- HPSS provides both striping (for large files) and file aggregation (for little ones)





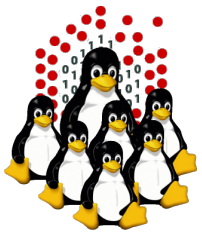
Cloud Storage

- Use http / https protocol for data transfer
  - build application service protocol on top
  - allows for service virtualization, location optimization, transparent failover (“always on”)
- Favor download performance over upload
  - typical usage: concurrent data access
- Typical is a *stateless* interface, i.e. only server defines the state of data and resolves conflicts



Storage vs. HPC Workflows

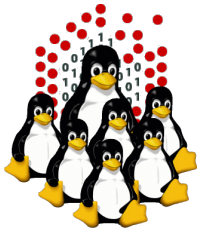
- Entirely compute bound HPC tasks are not affected at all: read input, compute, write result
- Entirely data bound tasks focus on location:
  - computation is sent to data (MapReduce)
  - replication allows speculative execution
- Everything in between can be “tricky”
  - need to consider data lifetime
  - need to consider effort to recompute data
  - use in-memory buffers, local/global storage
  - need to consider parallel performance



File Management in HPC Cluster

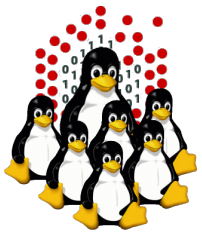
Data Staging

- Job's data requirements are identified and provided by user in submitted script.
- *Stage-in*: Input Files are transfered to local disk of compute nodes before job starting.
- *Stage-out*: Output Files are transfered from nodes to mass storage after execution.
- Nowadays, rarely used on clusters, as stage-in/out are based on shared/parallel FS, mainly used in Grid context.
- Explicit Control.



Wrap-up

- Storage hardware technologies
- RAID implementations:
  - hardware
  - software
- Fault tolerance vs efficiency vs performance
- File systems:
  - local
  - distributed
  - parallel
- Data workflow



Questions?


```
( questions ; comments ) | mail -s uheilaaa baro@democritos.it

( complaints ; insults ) &>/dev/null
```