

Computing Infrastructures

Polytechnic University of Milan
Master's Degree - Computer Science and Engineering

Professor: [Manuel Roveri](#)

A.A 2019/2020 – 2° Semester



POLITECNICO
MILANO 1863

! Notes based on the slides provided during the course. I take no responsibility for any errors.

by Simone Staffa

LinkedIn: <https://www.linkedin.com/in/simone-staffa-8b3b79158/>

Storage

Files

Disks can be seen by a OS as a collection of data blocks that can be read or written independently. To allow the ordering/management among them, each block is characterized by a unique numerical address called **LBA (Logical Block Address)**.

Typically, the OS groups blocks into **clusters** to simplify the access to the disk. Clusters are the minimal unit that a OS can read from or write to a disk.

Files are just a better way to address data on the clusters than LBA numbers.

Clusters contains:





- **File data**: the actual content of the files
- **Meta data**: the information required to support the file system

Meta data contains:

- File names
- Directory structures and symbolic links
- File size and file type
- Creation, modification, last access dates
- Security information
- **Links to the LBA where the file content can be located on the disk**

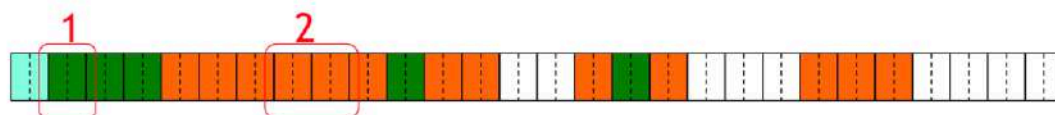
The disk can thus contain several types of clusters:



-  Meta data – fixed position (to bootstrap the entire file system)
-  Meta data – variable position (to hold the folder structure)
-  File data (actual content of the files)
-  Unused space (available to contain new files and folders)

Files - reading

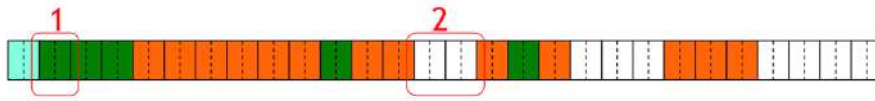
Reading a file requires to:



1. Accessing the meta-data to locate its blocks.
2. Access the blocks to read its content

Files - writing

Writing a file requires to:



1. Accessing the meta-data to locate free space.
2. Write the data in the assigned blocks

Since the file system can only access clusters, the real occupation of space on a disk for a file is always a multiple of the cluster size

- s – the file size
- c – the cluster size
- a – the actual size

Then we have

$$a = \text{ceil}(s / c) * c \quad w = a - s$$

Where w is the wasted disk space due to the organization of the file into clusters.

This waste of space is called **internal fragmentation** of files.

Example:

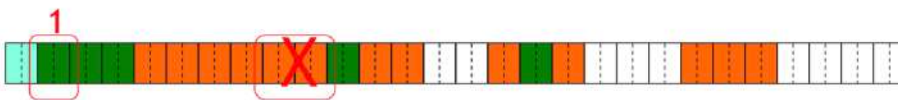
- s - file size = 27 byte
- c – cluster size = 8 byte
- actual size on the disk

$$a = \text{ceil}(27 / 8) * 8 = \text{ceil}(3.375) * 8 = 4 * 8 = 32 \text{ byte}$$

- Wasted disk space = $32 - 27 = 5$ byte

Files – deleting

Deleting a file requires:

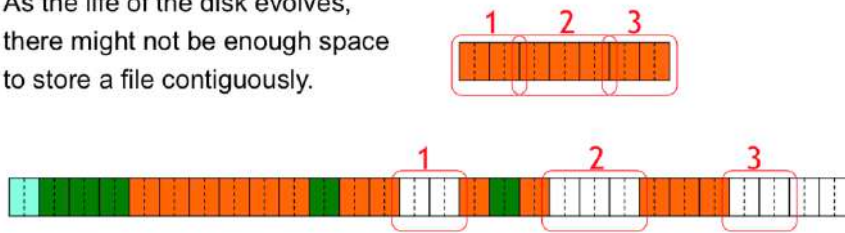


1. Only to update the meta-data to say that the blocks where the file was stored are no longer in use by the O.S.

Deleting a file never actually deletes the data on the disk: when a new file will be written on the same clusters, the old data will be replaced by the new one.

Files – external fragmentation

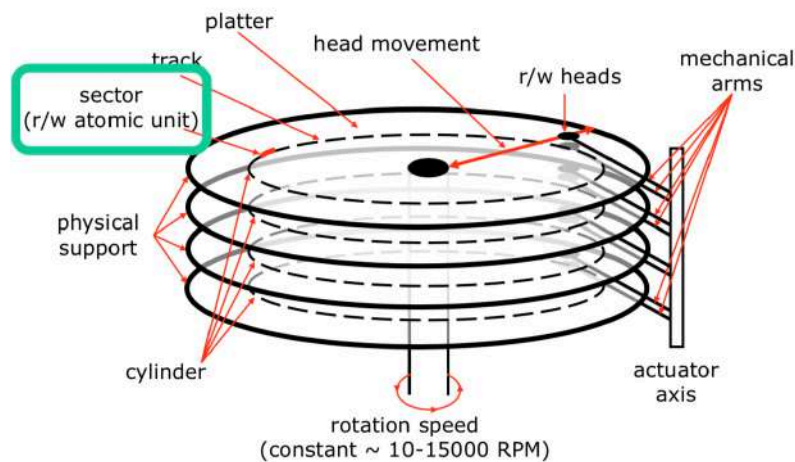
As the life of the disk evolves, there might not be enough space to store a file contiguously.



In this case, the file is splitted into smaller chunks that are inserted into the free clusters spread over the disk. The effect of splitting a file into non-contiguous clusters is called **external fragmentation**. As we will see this can reduce a lot the performance of an HDD.

Hard Disk Drives

An HDD is a data storage using rotating disks coated with magnetic material. It consists of one or more rigid rotating disks with magnetic heads arranged on a moving actuator arm to read and write data to the surfaces.



Here we see the problem of external fragmentation. To read a file with the head is simpler if it is on the same track. External fragmentation cause file to be on different track so there are delays on moving the head.

LBA and CHS

Sectors are usually grouped by the OS.

Clusters are normally power-of-two multiple of the sector size.

Each sector on a HDD is identified by three numbers (**CHS**).

- Cylinder
- Head
- Sector

A function in the BIOS of the PC converts the LBA index into the three CHS numbers that identify the sector on the HDD.

If we call:

- *HPC* – heads per cylinder: number of heads in the disk
- *SPT* – sector per track
- *C* – number of the cylinder (starting from 0)
- *H* – number of the head (from 0 to *HPC*-1)
- *S* – number of the sector (from 1 to *SPT*)

Position within
the HDD

Then:

$$LBA = (C * HPC + H) * SPT + (S-1)$$

And:

$$C = \text{floor}(LBA / (HPC * SPT))$$

$$H = \text{floor}(LBA / SPT) \bmod HPC$$

$$S = (LBA \bmod SPT) + 1$$

Example:

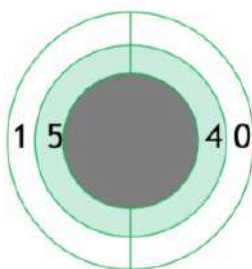
Table 7.9 CHS and LBA Sector Numbers for an Imaginary Drive with Two Cylinders, Two Heads, and Two Sectors per Track (Eight Sectors Total)

| Mode | Equivalent Sector Numbers | | | | | | | |
|------|---------------------------|-------|-------|-------|-------|-------|-------|-------|
| CHS: | 0,0,1 | 0,0,2 | 0,1,1 | 0,1,2 | 1,0,1 | 1,0,2 | 1,1,1 | 1,1,2 |
| LBA: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

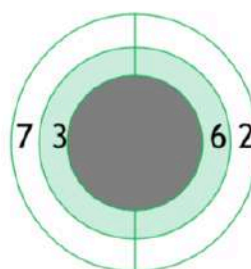
- 2 *HPC* – heads per cylinder: number of heads in the disk (Up/Down)
- 2 *SPT* – sector per track

C = 0

C = 1

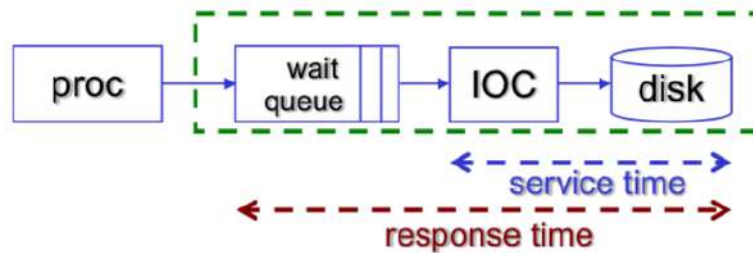


Up (H=0)



Down (H=1)

HDD Performances



service time s_{disk}

seek time + rotational latency + data transfer time + controller overhead

- **seek** time: head movement time (\approx ms), is a function of the number of cylinders traversed
- **latency** time: time to wait for the sector (\approx ms, $\frac{1}{2}$ round)
- **transfer** time: is a function of rotation speed, storing density, cylinder position (\approx MB/sec)
- **controller overhead** : buffer management (data transfer) and interrupt sending time

response time R : time spent in queue waiting for the resource (that is busy executing other requests) and for the execution of the request

Example:

data **transfer** rate: 50 MB/sec

rotation speed: 10000 RPM (round per minute)

mean **seek** time: 6ms Time required to move the head

overhead **controller**: 0.2ms

mean latency: $(60s/min) \times 1000 / (2 \times 10000 \text{ rpm}) = 3.0ms$ (time for $\frac{1}{2}$ round)

transfer time: $(0.5KB) \times 1000 / (50 \times 1024KB/s) = 0.01ms$

$$\text{mean I/O service time} = \underset{\text{seek}}{6ms} + \underset{\text{latency}}{3ms} + \underset{\text{transfer}}{0.01ms} + \underset{\text{controller}}{0.2ms} = \mathbf{9.21ms}$$

This considers only the very pessimistic case where sectors are fragmented on the disk in worst possible way. In many circumstances, this is not the case: files are larger than one block, and they are stored in a contiguous way to decrease the need of seek and rotational latency times.

Suppose we can measure the **data locality** of a disk as the percentage of block that do not need seek or rotational latency to be found.

Example:

- **locality**: the effect is that the seek time and transfer times are zero
 - **data locality l=75%**: seek+latency affect only **25%** of the operations
 - $T = (1 - l) * (T_s + T_l) + T_c + T_t$
 $(6.0 + (0.5 \times 60 \times 10^3 / 10000)) \times 0.25 + (0.5 \text{ KB} / 50\text{MB} \times 2^{10}) + 0.2$
 - mean time of one I/O op. = $(\underbrace{0.25}_{\text{seek+latency}} \times (6+3)) + \underbrace{0.01}_{\text{transfer controller}} + 0.2 = \mathbf{2.46 \text{ ms}}$

Solid State Disks (SSD)

A solid-state drive is a non-volatile storage device.

A controller is included in the device with one or more solid state memory components.

As we will see it's not faster and more reliable than an HDD.

Data is stored in array of NAND cells:

- SLC: cells store one bit
- MLC: cells store more bit by multilevel voltage

NAND flash is organized into:

- Pages: smallest unit that can be read/written (512-byte LBAs).
- Blocks: smallest unit that can be erased, typically consisting of multiple pages (64 pages).

The conversion from LBA to physical page is done thanks to special look-up tables (LUT).

With LUT we can change the assignments between LBA and page/blocks, so that I'm able to move pages without moving them physically.

LBA LUT

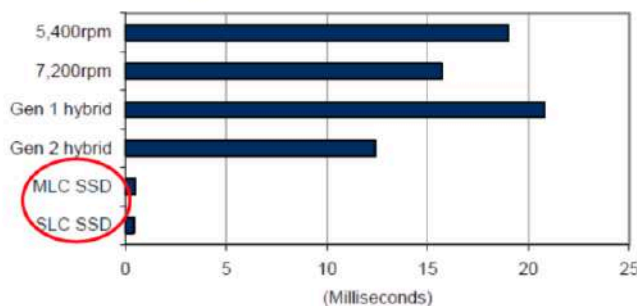
| LBA | Plane | Block | Page |
|-----|-------------------|-------------------|-------------------|
| 0 | 0 | 1 | 1 |
| 1 | 0 1 | 0 1 | 1 0 |
| 2 | 3 | 4097 | 0 |
| 3 | 2 | 4096 | 1 |
| 4 | 1 | 0 | 1 |

Pages can be in three states:

- Empty: they do not contain data.
 - Dirty: they contain data, but this data is no longer in use
 - In use: the page contains data that can be actually read
-
- Only empty pages can be written.
 - Only dirty pages can be erased (at block level).
 - If no empty page exists, some dirty page must be erased.
 - If no block containing just dirty or empty pages exists, then special procedures should be followed to gather empty pages over the disk.

We can write and read a single page of data from a SSD but we have to delete an entire block to release it.

HD Tach — Access Speeds

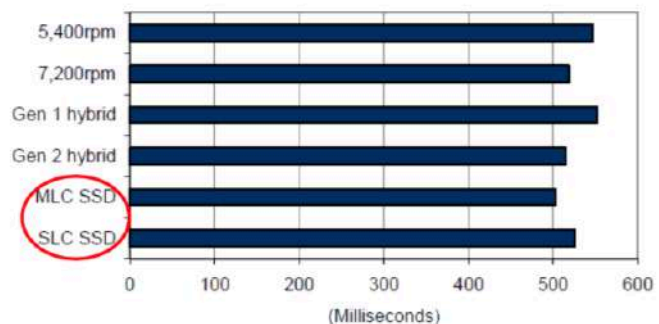


Access speed
measured at device level in a
controlled laboratory environment
HDD vs SSD
~ 15-20x improvement

Same device
integrated into a
notebook PC with a real
application test
~ same performance !!!

SSD are not faster!
Why such behavior ?

Internet Explorer Launch



After the drive is filled up, the amount of time to write a block of data increase a lot since it also had to delete blocks. This actually made the SSDs in such conditions slower than a regular HDD.

Example:

- Hypothetical SSD:
 - Page Size: 4KB
 - Block Size: 5 Pages
 - Drive Size: 1 Block
 - Read Speed: 2 KB/s
 - Write Speed: 1 KB/s



Let's write a 4Kb text file to the brand new SSD. Then a 8Kb dog pic file and finally a 12Kb pic.



We expect 12s to write, but we need more time to delete the dirty page. To erase the dirty page, we have to delete the whole block. IT ACTUALLY TOOK 26s.



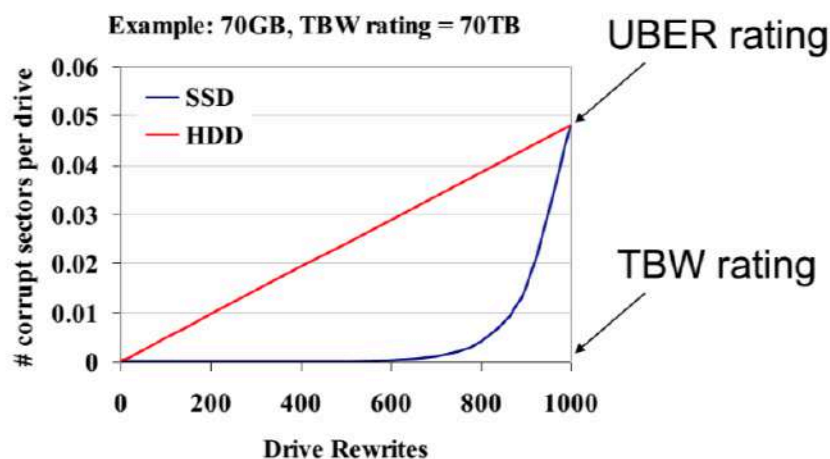
- ⦿ Step 1: Read block into cache
- ⦿ Step 2: Delete page from cache
- ⦿ Step 3: Write new pic into cache
- ⦿ Step 4: Delete the old block on SSD
- ⦿ Step 5: Write cache to SSD

SSD – drawbacks

They cost more than the conventional HDD. They have a shorted lifetime and they have a different read/write speed. Write performances degrades over time. SSD are not affected by data-locality and must not be defragmented.

HDD vs SSD

- **Retention failure:** A data error occurring when the SSD is read after an extended period of time following the previous write.
- **Endurance failure:** A failure caused by endurance stressing.
- **Endurance rating (TBW rating):** The number of terabytes that may be written to the SSD while still meeting the requirements.
- **Write amplification factor (WAF):** The data written to the NVM divided by data written by the host to the SSD.
- **Unrecoverable Bit Error Ratio (UBER):** A metric for the rate of occurrence of data errors, equal to the number of data errors per bits read



Some large storage servers use SSD as a cache for several HDD. Some mainboards of the latest generation have the same feature: the combine a small SSD with a large HDD to have a faster disk.



Some HDD manufacturers produce Solid State Hybrid Disks (SSHD) that combine a small SSD with a large HDD in a single unit.

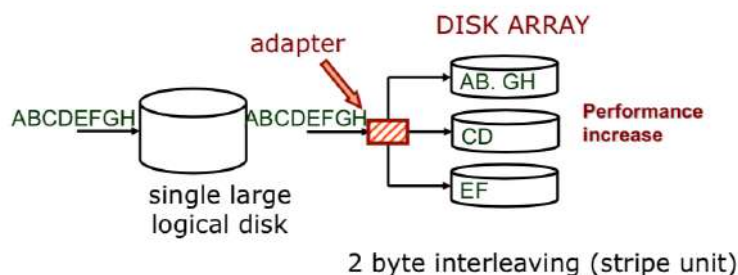
RAID Disks

In 1980's disk arrays were proposed because there was the need to increase the performance, the size and the reliability of storage systems. Disk arrays are composed by several independent disks that are considered as a single, large, high performance logical disk. Data are striped across several disks accessed in parallel in order to achieve:

- **High data transfer rate**: large data accesses
- **High I/O rate**: small but frequend data accesses
- **Load balancing** across the disks

We have two orthogonal techniques:

- **Data striping**: to improve performance
- **Redundancy**: to improve reliability



Data Striping

- **Striping**: data are written sequentially in units on multiple disks according to a cyclic algorithm
 - **Stripe unit**: dimension of the unit of data that are written on a single disk
 - **Stripe width**: number of disks considered by the striping algorithm
1. **Multiple independent I/O requests** will be executed in parallel by several disks decreasing the queue length of the disks.
 2. **Single multiple-block I/O requests** will be executed by multiple disks in parallel increasing of the transfer rate of a single request

The more physical disks in the array the larger the size and performance gains but the larger the probability of failure of a disk, that is the main motivation for the introduction of **REDUNDANCY**

The probability of failure increase with the number of disks in the array. To overcome failures, we may apply redundancy. Redundancy consists in error correcting codes computed to tolerate loss due to disk failure. This technic reduces the performances because write operations must update also the redundant information.

| | Disk 0 | Disk 1 | Disk 2 | redundant data | Sum of the data |
|---------------------------------------------|--------|--------|--------|----------------|-----------------|
| Values | 10 | 8 | 2 | 20 | |
| | 10 | failed | 2 | 20 | |
| $failed = 20 - (10 + 2) = 8$ | | | | | |
| | Disk 0 | Disk 1 | Disk 2 | parity | |
| bits | 1 | 1 | 0 | 0 | |
| | 1 | failed | 0 | 0 | |
| Parity calculation/reconstruction using XOR | | | | | |
| $failed = 1$ | | | | | |

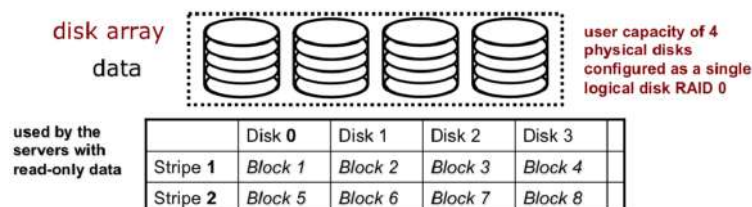
In the first example we apply a sum-based redundancy in which to recover data, we check the sum of the whole blocks. The second one consists in controlling the block's parity using a XOR op.

RAID architectures

- RAID 0 striping only
- RAID 1 mirroring only (RAID 0+1 and RAID 1+0)
- RAID 2 bit interleaving
- RAID 3 byte interleaving (not used)
- RAID 4 block interleaving - redundancy (parity disk)
- RAID 5 block interleaving – redundancy (parity block distributed)
- RAID 6 greater redundancy (2 failed disks are tolerated)
- RAID 7 (proprietary solutions)

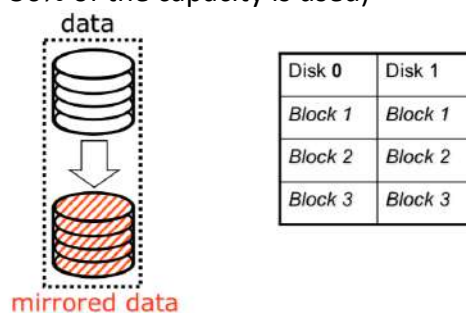
RAID level 0 – striping, no redundancy

Data are written on a single logical disk and splitted in several blocks distributed across the disks according to a striping algorithm. This is used where **performance** and **capacity**, rather than reliability, are the primary concerns and is required a minimum of two drives. Since there is no redundancy, we have **low cost** and we have the **best write performance**. The drawback is that a **single disk failure will result in data loss**



RAID level 1 – mirroring

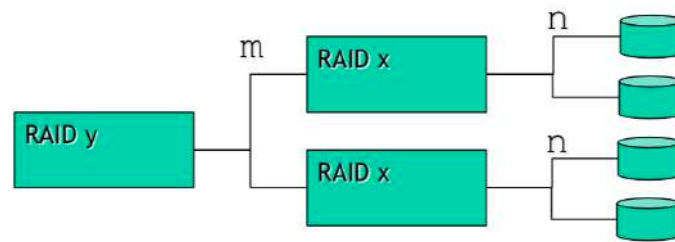
Whenever data is written to a disk it is also duplicated (**mirrored**) to a second disk. This one also requires minimum 2 disk drives. Here we have a **high reliability** because when a disk fails the second copy is used. We have a fast **read** of data and a fast **write** since no correcting code should be computed (**no redundancy**) but still slower than standard disks due to duplication. Main drawback is the high cost (only 50% of the capacity is used)



If several disks are available, disks could be coupled. Each disk has a mirror and the total capacity is halved.



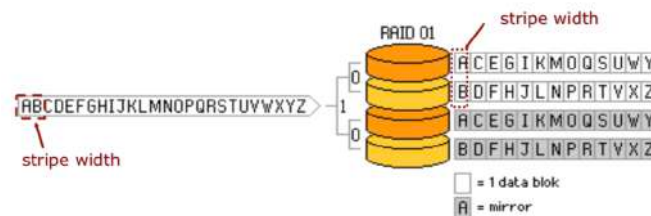
RAID levels (combined)



RAID x + y

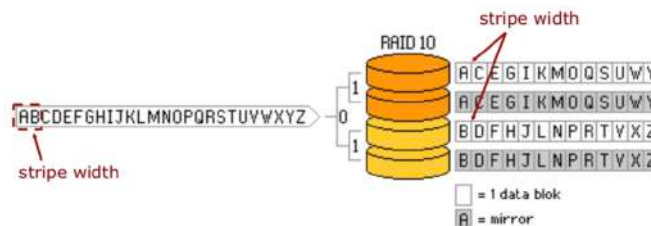
- $n \times m$ disks in total
- Consider m groups of n disks
- Apply RAID x to each group of n disks
- Apply RAID y considering the m groups as single disks

RAID 0 + 1

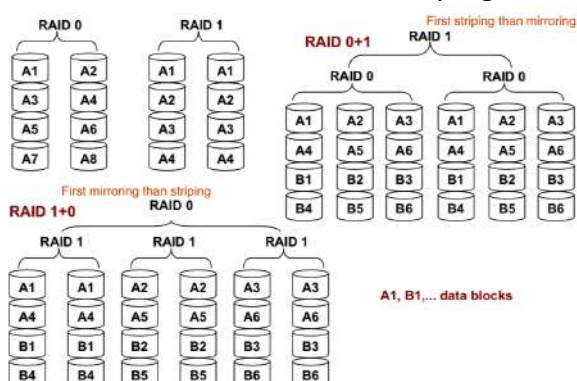


- High performance and reliability
- Minimum 4 drives
- After the first failure the model becomes as a RAID 0
- High overhead (disk duplication)

RAID 1 + 0



- High performance and higher reliability than RAID 0+1
- Minimum 4 drives
- Used in databases with very high workload (fast writes)



In these two different combined configurations the blocks are the same but are allocated in a different order and the performance on both RAID 10 and RAID 01 will be the same, as the storage capacity. The main difference is the fault tolerance level: on most implementations, RAID 01 fault tolerant is less since we have only two groups of RAID 0, that's why if two drives fail (one in each group), the entire RAID 01 will fail. While in RAID 10 the fault tolerance is larger since there are many groups (as the individual group is only two disks), and even if three disks fail (one in each group), the RAID 10 is still functional.

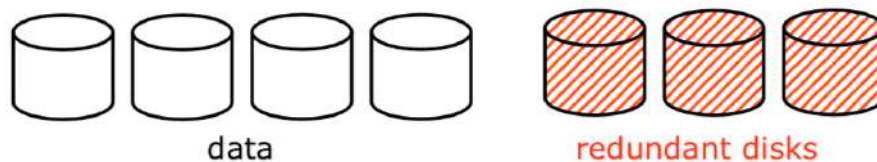
RAID level 2 – bit interleaved parity (not used)

Here we assume that the failed component is unknown. We apply the ECC memory style (Hamming codes), that has less cost than mirroring but is still very high.

- 4 data disks => 3 redundant disks (one less than mirroring)
- 11 data disks => 4 redundant disks (seven less than mirroring)

[#redundant_disks is proportional to $\log(\text{\#total_disks})$]

Parity blocks are computed for several subset of overlapped data, and when disk fails several of the parity blocks will have inconsistent values, the failed component is the one held in common by each incorrect subset.



RAID level 3 – bit interleaved parity (less used)

Here we assume that the failed component is known. Data are interleaved byte-wise over the data disk and is required a single redundant bits disk for the parity bytes. So that if we lose one disk, we can reconstruct its byte thanks to parity bits.

This configuration is used in applications that required high bandwidth by not high I/O. Each read accesses all data disks. Each write accesses all data disks and the parity disks.

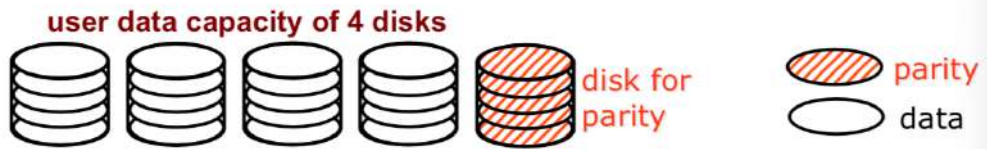


Byte parity = $\text{byte_0 XOR byte_1 XOR byte_2 XOR byte_3}$

| | |
|------------------------------|-----------------------------------|
| Having the following 3 bytes | Byte a) is lost and reconstructed |
| a) 01100011 XOR | p) 00000011 XOR |
| b) 10101010 XOR | b) 10101010 XOR |
| c) 11001010 = | c) 11001010 = |
| p) 00000011 | a) 01100011 |

RAID level 4 – block interleaved parity

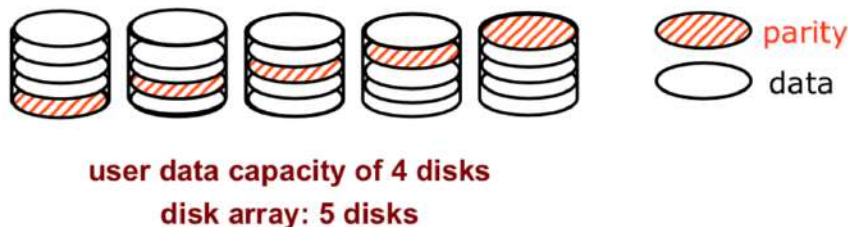
Similar to level 3 but striping here is at the block level and the parity is calculated at the block layer. One disk is used to store the parity of the strips that have the same position in the disks. Write must update the requested data blocks and must compute and update the parity block. The bottleneck here is the single disk for redundant data (parity disk). RAID4 encounters data loss only if two disks fail concurrently.



| | Disk 0 | Disk 1 | Disk 2 | Disk 3 | Disk 4 redundancy |
|----------|---------|----------|----------|----------|----------------------|
| Stripe 1 | Block 1 | Block 2 | Block 3 | Block 4 | Parity 1-4 |
| Stripe 2 | Block 5 | Block 6 | Block 7 | Block 8 | Parity 5-8 |
| Stripe 3 | Block 9 | Block 10 | Block 11 | Block 12 | Parity 9-12 |

RAID level 5 – block interleaved distributed parity

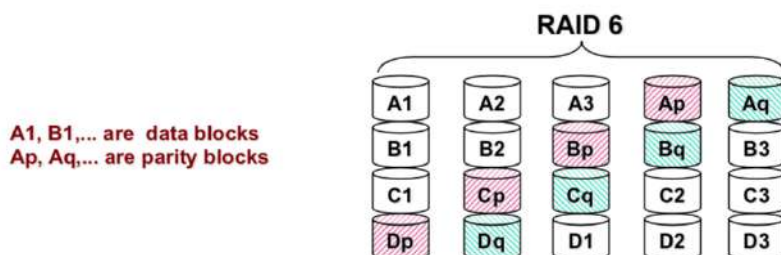
Similar to level 4 but parity is uniformly distributed over all of the disks (avoiding the bottleneck on the parity disk). Data are also distributed over all the disks. In case of disk fail, reconstruction of its content is more complex since it stores both data and parity. It encounters data loss only if two disks failed concurrently.



RAID level 6

The last level is able to withstand the failure of two disks simultaneously, combining two levels of parity distributed on the various disks (as level 5). For this system we need a disk more than level 5 to store the same amount of data, and we will have a greater computational overhead. This solution is adopted for very critical applications.

$N+2$ disks required \Rightarrow each write required $N+2$ accesses due the need to update bot the P and Q parity blocks.



Not efficient when the number of disks is small.

Calculating Parity P

$$P = D_0 + D_1 + \dots + D_{n-1} = \sum_{i=0}^{n-1} D_i$$

Calculating Parity Q using a generator value g (common choice is g=2)

$$Q = D_0 + g \cdot D_1 + \dots + g^{n-1} \cdot D_{n-1} = \sum_{i=0}^{n-1} g^i \cdot D_i, \quad g \neq 1$$

| | Disk 0 | Disk 1 | Disk 2 | Disk 3 | Disk 4 |
|----------|---------|---------|---------|--------|------------|
| Stripe 1 | Block 1 | Block 2 | Block 3 | P 1-3 | Q 1-3 |
| | | | | 2+10+7 | 2+10*2+7*4 |
| Values | 2 | 10 | 7 | 19 | 50 |

What if we write new data in block 1?

1. Read old Block 1, old P 1-3 and old Q 1-3 (3 reads)
2. Compute new P 1-3 from old Block 1, new Block 1, old P 1-3
3. Compute new Q 1-3 from old Block 1, new Block 1, old Q 1-3
4. Write new Block 1, and P 1-3 and Q 1-3 (3 write)

6 disk I/O are required

| | Disk 0 | Disk 1 | Disk 2 | Disk 3 | Disk 4 |
|----------|---------|---------|---------|---------|---------|
| Stripe 1 | Block 1 | Block 2 | Block 3 | P 1-3 | Q 1-3 |
| Stripe 2 | Block 4 | Block 5 | P 4-6 | Q 4-6 | Block 6 |
| Stripe 3 | Block 7 | P 7-9 | Q 7-9 | Block 8 | Block 9 |

new data in Block 1

New parity P

$$P^{new} = P^{old} + D^{new} - D_i^{old}$$

New parity Q

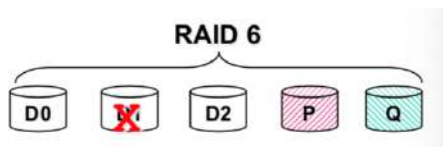
$$Q^{new} = Q^{old} + g^i \cdot (D_i^{new} - D_i^{old})$$

| | Disk 0 | Disk 1 | Disk 2 | Disk 3 | Disk 4 |
|------------|---------|---------|---------|--------------|------------------|
| Stripe 1 | Block 1 | Block 2 | Block 3 | P 1-3 | Q 1-3 |
| Values | 2 | 10 | 7 | 19 | 50 |
| New values | 2 | X 5 | 7 | 19+5-10 = 14 | 50+2*(5-10) = 40 |

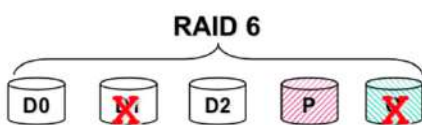
RAID 6 failure and repair

Raid 6 can recover from up to two failed disks. They can be divided in 3 different cases:

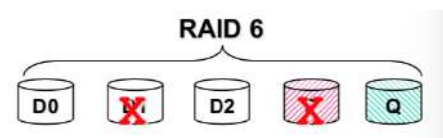
- Repairing a data disk
- Repairing a data disk and Q
- Repairing a data disk and P
- Repairing two data disks



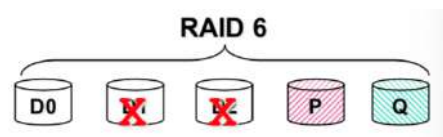
$$D_i = P - \sum_{j=0, j \neq i}^{n-1} D_j$$



$$D_i = P - \sum_{j=0, j \neq i}^{n-1} D_j \quad Q = \sum_{i=0}^{n-1} g^i \cdot D_i$$



$$D_i = \left(Q - \sum_{j=0, j \neq i}^{n-1} g^j \cdot D_j \right) / g^i \quad P = \sum_{i=0}^{n-1} D_i$$

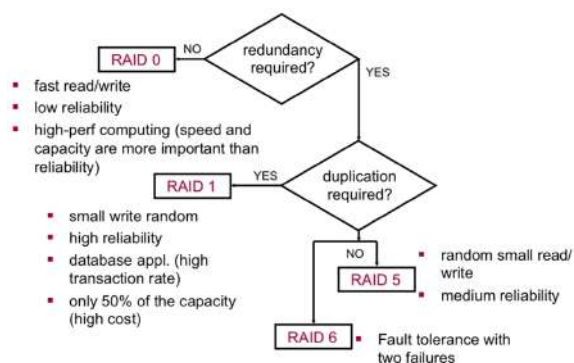


$$P^* = \sum_{k=0, k \neq i, k \neq j}^{n-1} D_k \quad Q^* = \sum_{i=0, k \neq i, k \neq j}^{n-1} g^k \cdot D_k$$

then calculating D_i and D_j from this system

$$\begin{cases} P = P^* + D_i + D_j \\ Q = Q^* + g^i \cdot D_i + g^j \cdot D_j \end{cases}$$

Selection of RAID level



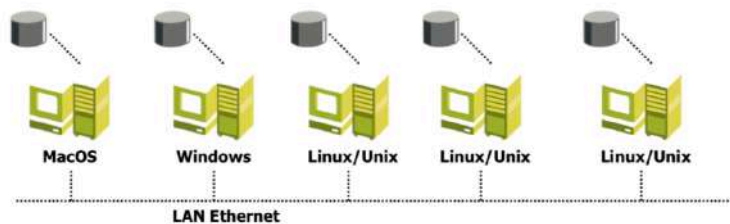
| RAID level | Capacity | Reliability | R/W performance | Rebuild performance | Suggested applications |
|------------|-----------|-------------|------------------|---------------------|------------------------------------------------------|
| 0 | 100% | N/A | Very good | Good | Non critical data |
| 1 | 50% | Excellent | Very good / good | good | Critical information |
| 3 | $(n-1)/n$ | Good | Good / fair | Fair | Single user, large file processing, image processing |
| 5 | $(n-1)/n$ | Good | Good/ fair | Poor | Database, transaction based applications |
| 6 | $(n-2)/n$ | Excellent | Very good/ poor | Poor | Critical information, w/minimal |
| 1+0 | 50% | Excellent | Very good/ good | Good | Critical information, w/better performance |
| 3+0/5+0 | $(n-1)/n$ | Excellent | Very good/ good | Fair | Critical information w/fair performance |

Storage Systems

DAS - Direct Attached Storage

DAS is a storage system directly attached to a server or workstation.

- limited scalability
- complex manageability
- limited performance
- To read files in other machines, the “file sharing” protocol of the OS must be used

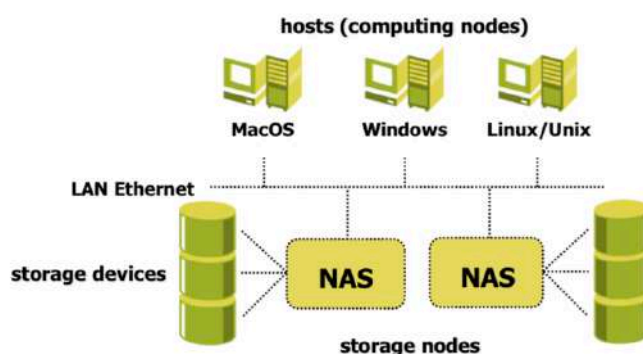


DAS does not necessarily mean “internal drives”, all the external disks connected with a point-to-point protocol to a PC can be considered as DAS. USB is an example of DAS

NAS – Network Attached Storage

A NAS unit is a computer connected to a network that provides only file-based data storage services to other devices on the network. They can contain one or more hard disks, often organized into logical redundant storage containers or RAID. NAS uses file-based protocols such as NFS and Samba.

- each NAS element has its own IP address
- good scalability (incrementing the devices in each NAS element or incrementing the number of NAS elements)



NAS vs DAS

- DAS is simply an extension of an existing server and is not necessarily networked.
- NAS is designed as an easy and self-contained solution for sharing files over the network.

Their performance are not comparable since NAS is networked and its performances depends mainly on the speed and congestion of the network

SAN – Storage Area Network

Remote storage units that are connected to PC using a specific networking technology.

NAS vs SAN 1

- NAS provides both storage and a file system.
- SAN provides only block-based storage and leaves file system concerns on the client side

NAS appears to the client OS as a file server, while SAN appears as a disk.

NAS vs SAN 2

- NAS is used for low-volume access to a large amount of storage by many users.
- SAN is the solution for petabytes of storage and multiple, simultaneous access to files, such as streaming audio/video.

Dependability

The *Dependability* is the collective term used to describe the availability performance of a system and its influencing factors: reliability performance, maintainability performance and maintenance support performance

It encompasses:

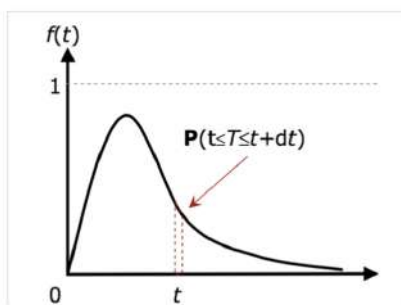
- Reliability: continuity of correct service
- Availability: readiness for correct service
- Maintainability: reparation to restore correct service

Dependability and system performance can't be described by deterministic laws, these phenomena must be approached with statistical techniques.

Probability of failure

Probability that a component fails at a given time instant:

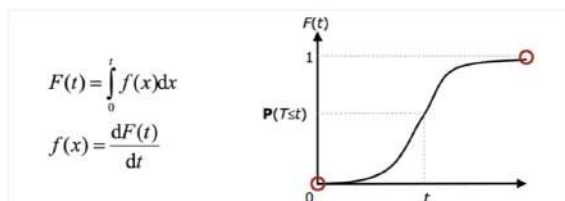
- $f(t)$ = probability density function of the failure
- T = time instant where the first failure occurs
- $f(t)dt = P(t \leq T \leq t + dt)$



Unreliability - F(t)

Probability of the component failure in the interval 0...t, knowing that for t=0 the component was working correctly.

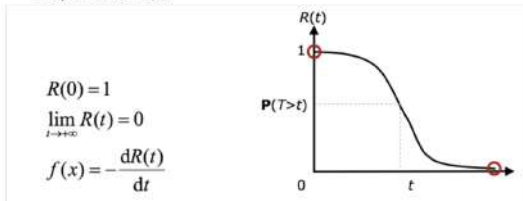
- $F(t) \equiv P(T \leq t)$
- $F(t)$ is the cumulative distribution



Reliability - R(t)

Probability of the component does not fail in the interval in the interval 0...t, knowing that for t=0 the component was working correctly.

- $R(t) \equiv 1 - F(t) = P(T > t)$
- Properties of R(t):



Computing Reliability

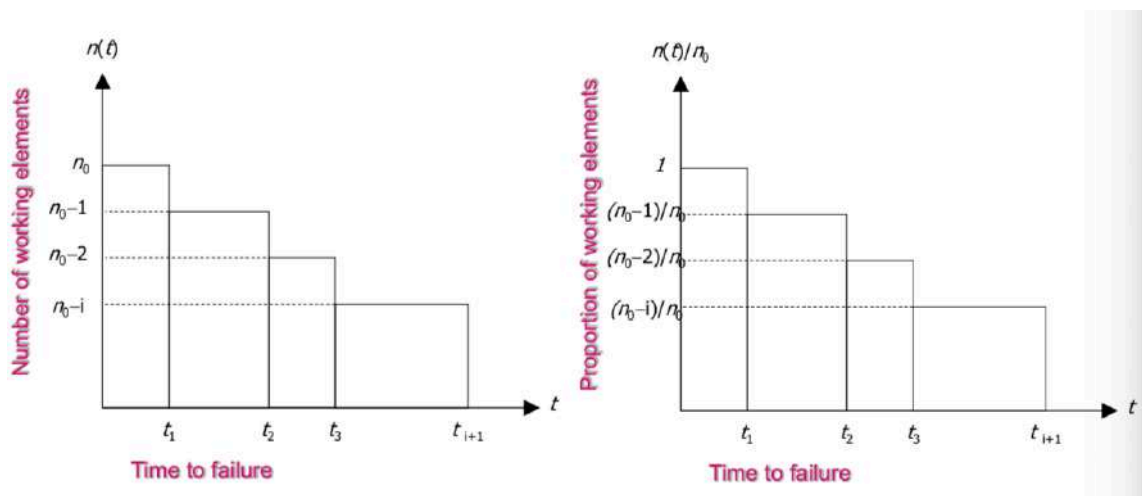
Let's consider n_0 independent and statistically identical elements deployed at time $t = 0$ in identical conditions

- $n(0) = n_0$

At time t , $n(t)$ elements *are not* failed

t_1, t_2, \dots, t_{n_0} are the times to failure of the n_0 elements (i.e., the times between $t = 0$ and the instant at which the failure occur – times to failure)

times to failure are independent occurrences of the random quantity τ (t_i : time to failure j^{th})



- Function $n(t) / n_0$ is the empirical function of reliability that as $n_0 \rightarrow \infty$ converges to the value:

$$n(t) / n_0 \rightarrow R(t)$$

- Represents the probability that at time t a given component is still working

If a component has reliability $R(10000) = 99\%$ then it has the probability of 99% do not fail in 10000hours of operation. Such components is more reliable than another with $R(10000) = 90\%$ operating under the same conditions.

A component with $R(20000) = 90\%$ is not comparable with the previous due to different timing.

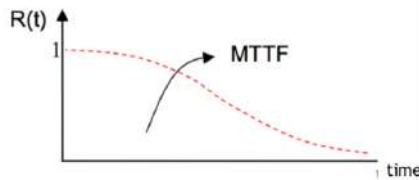
Mean Time to Failure (MTTF)

Mean time of working for a component, or mean time before a failure:

$$MTTF = \int_0^{\infty} t * f(t) dt$$

Interestingly

$$MTTF = \int_0^{\infty} R(t) dt$$



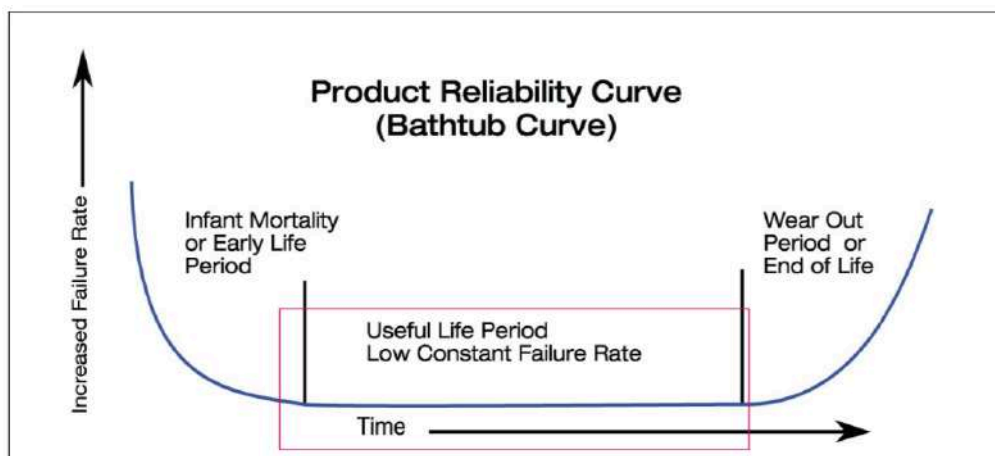
Failure Rate

The failure rate can be interpreted as the number of faults in the unit of time, it is a measure of the speed of occurrence of the fault.

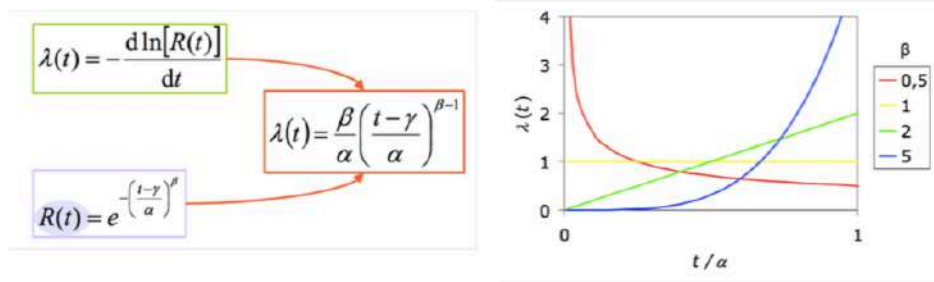
$\lambda(t)$: failure rate - $\lambda(t)dt$ probability of component failure at instant t assuming that the component was working at time t

Source of failures

- **Design failures:** caused by project or manufacturing errors
- **Infant Mortality:** failures showing up in new systems.
- **Random failures:** showing up randomly during the entire life of a system.
- **Wear out:** at the end of its life, some components can cause the failure of a system.



Weibull distribution (or combination of)



From this point on, all failure and repair time distribution will be considered to be **independent** and **exponential**.

In case the distribution is exponential we have:

$$F(t) = 1 - e^{-\lambda t} \quad f(t) = \lambda e^{-\lambda t} \quad R(t) = e^{-\lambda t}$$

$$MTTF = E[X] = \int_0^{\infty} t \cdot f(t) dt = \int_0^{\infty} t \cdot \lambda e^{-\lambda t} dt = \frac{1}{\lambda}$$

$$\lambda = \frac{1}{E[X]} = \frac{1}{MTTF}$$

$$F(t) = 1 - e^{-\frac{t}{MTTF}} \quad f(t) = \frac{e^{-\frac{t}{MTTF}}}{MTTF} \quad R(t) = e^{-\frac{t}{MTTF}}$$

Example: Compute the probability that a disk with MTTF = 100000 hours fails at least once in 3 years (i.e., $t = 3 \times 365 \times 24 = 26280$ hours)

$$P(X \leq t) = 1 - e^{-\frac{26280}{100000}} \quad \text{That is } \approx 23\%$$

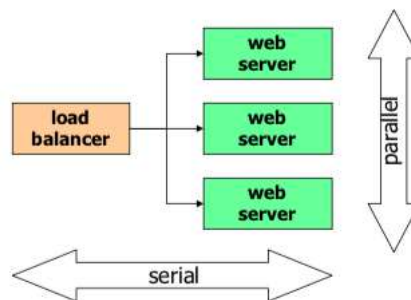
considering the 3 years period **small** with respect to the MTTF, probability can be approximated with: $26280/100000 \approx 26\%$

We can introduce redundant elements so that a system can tolerate multiple failed components. To increase reliability, we may:

- Use multiple redundant components
- Use element internally highly reliable (high MTTF)
- Have spare components at disposal
- Reduce MTTR

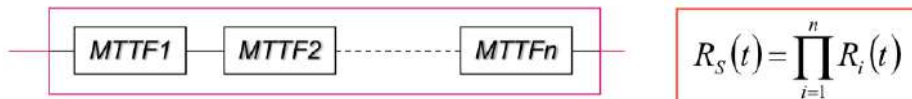
The picture below is called RBD (Reliability Block Diagram) and its used to represents the behavior of a system with many operative components disposed in different configurations:

- Serial connection between components requires that all must be operative
- Parallel connections require that at least one must be operative



MTTF (Mean Time to Failure)

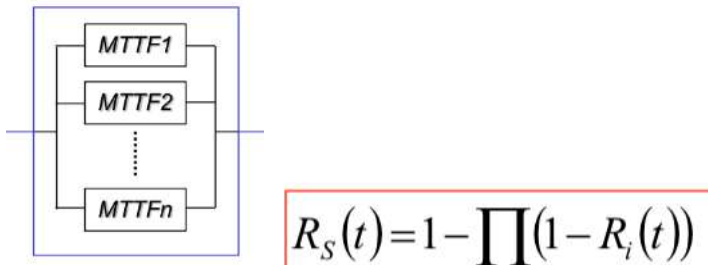
Serial Components



$$F_{SERIE} = 1 - \prod_{i=1}^n (1 - F_i(t)) = 1 - \left(e^{-\left(\frac{1}{MTTF1} + \frac{1}{MTTF2} + \dots \right) t} \right)$$

$$MTTF_{SERIE} = \frac{1}{\frac{1}{MTTF1} + \frac{1}{MTTF2} + \dots + \frac{1}{MTTFn}}$$

Parallel Components



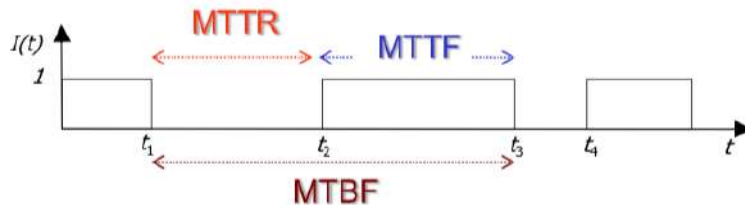
$$F_S(t) = \prod_{i=1}^n F_i(t) \quad MTTF_{PARALLEL} = MTTF \left(\frac{1}{n} + \frac{1}{n-1} + \dots + \frac{1}{2} + 1 \right)$$

MTTR (Mean Time to Repair)

Mean time required to repair a failed component => mean time during which a component is not working.

MTBF (Mean Time Between Failure)

Mean time between two failure, which is equal to the sum of the MTTF and the MTTR



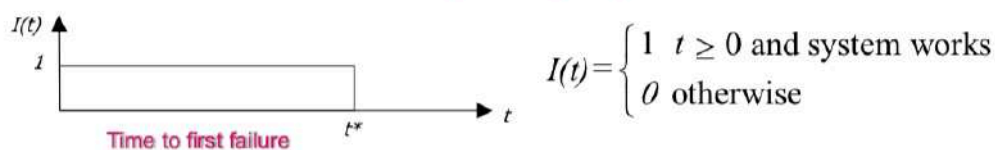
Availability

The availability represents the probability that a given instant of time the system is working.

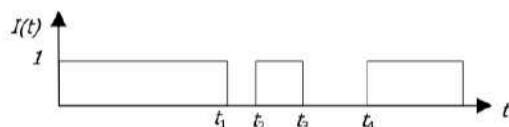
$$A = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

Availability vs Reliability

Reliability represents the probability of components, parts and systems to perform their required functions for a desired period of time **without failure (and repair)**



Availability represents the probability that the (repairable) system is capable of conducting its required function when it is called upon, given that it is not failed or undergoing a repair action.



Failure conditions

$$A = \prod_{i=1}^n A_i \quad \blacksquare \text{ One component (serial)}$$

$$A = 1 - \prod_{i=1}^n (1 - A_i) \quad \blacksquare \text{ All components (parallel)}$$

$$A_{SERIE} = \prod_k \frac{MTTF_k}{MTTF_k + MTTR_k}$$

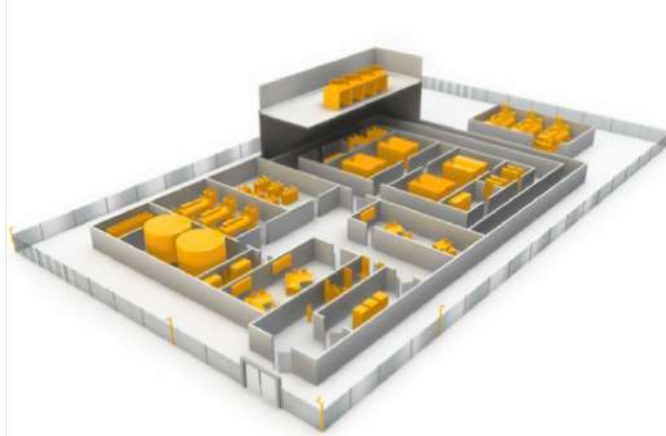
$$MTTR_{SERIE} = \frac{(1 - A_{SERIE}) MTTF_{SERIE}}{A_{SERIE}}$$

$$A_{PAR} = 1 - \prod_k \left(1 - \frac{MTTF_k}{MTTF_k + MTTR_k} \right)$$

$$MTTF_{PAR} = \frac{A_{PAR} MTTR_{PAR}}{1 - A_{PAR}}$$

Data Center

Beside the IT equipment, a data-center must have several physical characteristics to ensure connectivity, reliability, security and safety. Data centers are not just servers!

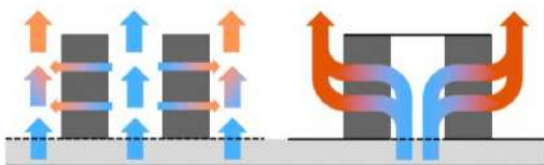


The IT equipment is stored into corridors, and organized into racks. The equipment generates a lot of heat: the cooling system is usually a very expensive component of the datacenter, and It is composed by coolers, heat-exchangers and cold water tanks (placed on top of data center because the hot air goes up).

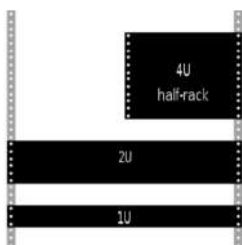
Batteries and diesel generators are used to backup the external supply in order to protect from power failures.

Corridors where servers are located are split into *cold aisle*, where the front panels of the equipment is reachable, and *warm aisle*, where the back connections are located.

Cold air flows from the cool aisle, cools down the equipment and leave the room from the warm aisle.



Racks are special shelves that accommodate all the IT equipment and allow their interconnection. IT equipment must conform to specific standar sizes to fit into the rack shelves.



We can distinguish 4 types of equipment present into racks:

- Servers
- Communication equipment
- Storage units
- Power distribution units

Servers

Servers are the main processing equipment. They are like ordinary PC, but with a form factor that allows to fit them into racks.

Communication equipment

Communication equipment allow network interconnections among the devices.

Storage units

Storage units holds large amount of data. They can use both HDD, SDD or RAM disks.

Power distribution

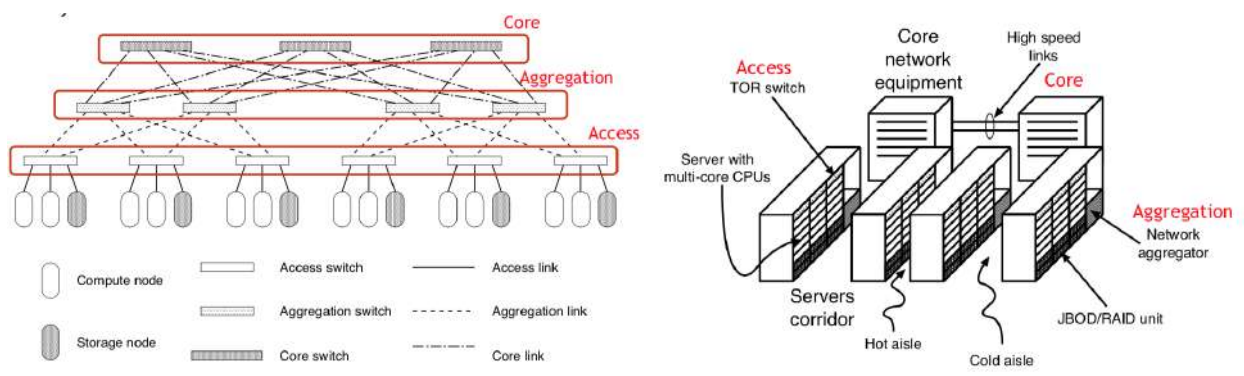
Power distribution units are used to distribute energy to the devices inside the rack. They can provide additional feature such as energy consumption monitoring and remote turn on/off.

Data-center network architectures

It has been proven that the main bottleneck on the performances of the infrastructure is the network. The most important and used network configuration are:

- Three Layers
- Fat-tree
- D-Cell

Three Layer



Switches can be placed into two positions:

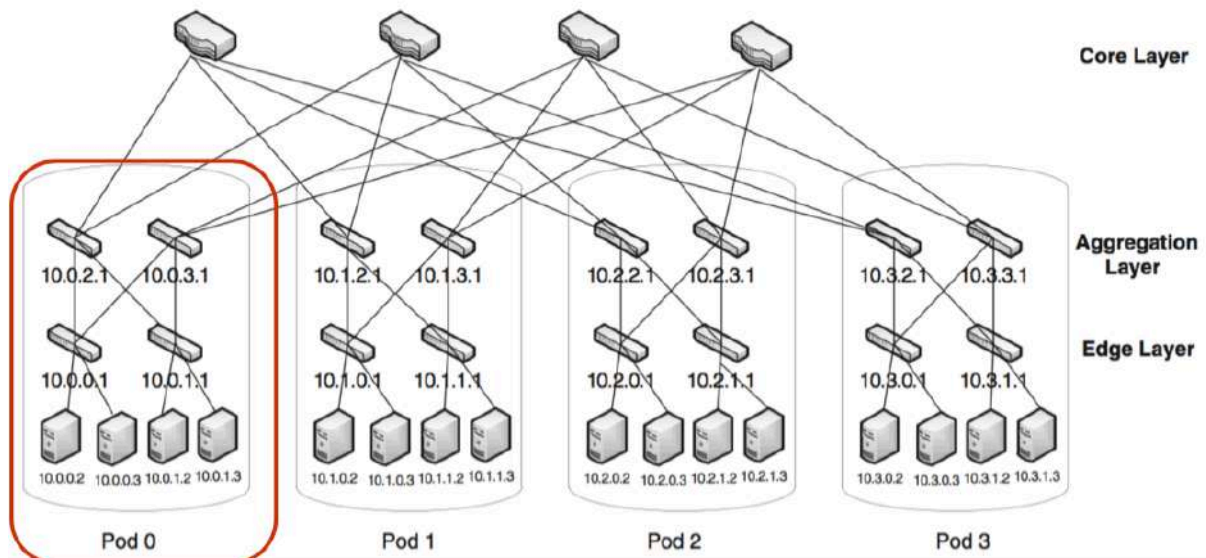
- **Top-Of-the-Rack (TOR)**: the number of cables, ports and scalability are limited, however the cost is lower.
- **End-Of-the-Line (EOL)**: switches must have a larger number of ports and longer cables (higher cost), however the system can scale to have a larger number of machines.

Bandwidth can be increased by increasing the switches at the core and aggregation layer by using routing protocols such as Equal Cost Multiple Path (ECMP) that equally shares the traffic among different routes.

This solution is very simple but can be very expensive for large data-centers since the cost in term of acquisition and energy consumption can be very high.

Fat-tree

Fat-tree topologies use a larger number of slow speed switches and connections. In particular, nodes are divided into pods characterized by the same number of nodes and switches.

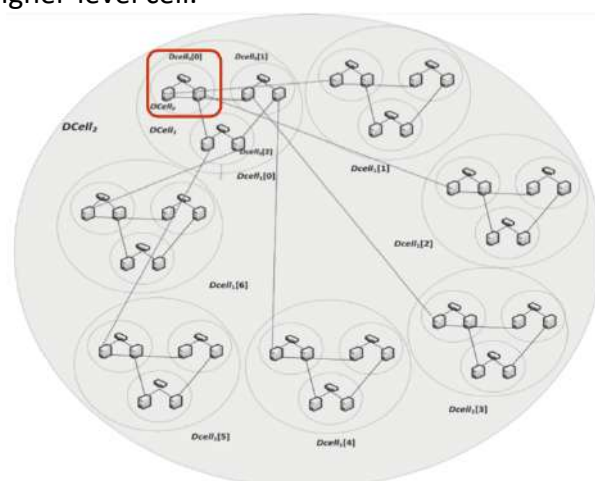


In every pods, switches are divided into two layers and they are connected to half of the nodes.

The advantage of **Fat-tree** over **Three-layer** is that it only uses slower switches which are much less expensive than the one required at the aggregation or core layers. Its limit is that It requires a high number of switches and cables.

D-Cell

D-Cell topology defines the network in recursive way. Cells are organized in levels. Switches connects nodes at the lower level. Some nodes belong to different cells: they perform routing among them to create a higher-level cell.



The advantage of **D-Cell** is that it requires fewer switches and less cables than the **Fat-Tree**. However, some nodes must be configured as router and this can limit their performance and also introduce network administration problems.

Data-center power consumption

Data-center power consumption is an issue, since it can reach several MWs. Cooling usually requires about half the energy required by the IT equipment (servers + network + disks). Energy transformation creates also a large amount of energy wasted for running a datacenter.

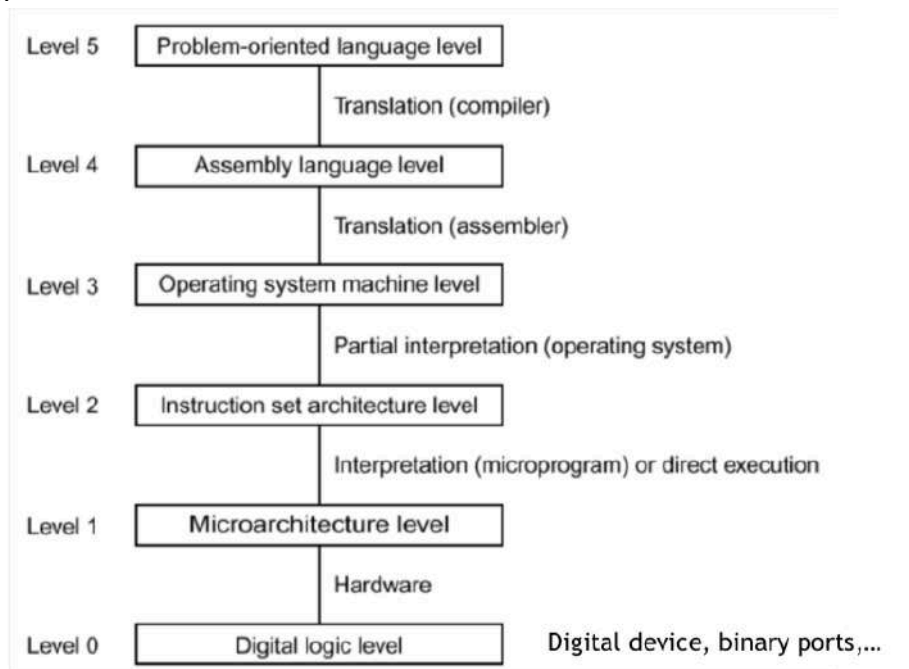
| Amortized Cost | Component | Sub-Components |
|----------------|----------------|----------------------------------|
| ~45% | Servers | CPU, memory, disk |
| ~25% | Infrastructure | UPS, cooling, power distribution |
| ~15% | Power draw | Electrical utility costs |
| ~15% | Network | Switches, links, transit |

Virtualization A

Virtual Machines

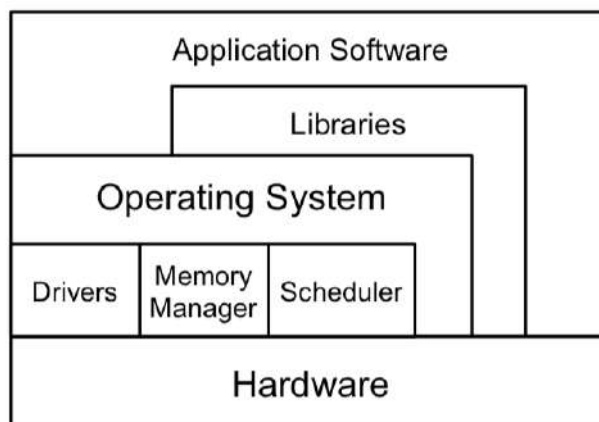
A machine is an execution environment capable of running a program.

In computer architectures, the set of instruction that a program can use might be structured at different levels.



We usually program the software part (3-4-5), exploiting the hardware (0-1-2).

An application software running on a system can be built using libraries, OS functionalities or hardware facilities. The OS can perform I/O using drivers, control process via the scheduler, and allocate RAM through the memory manager.

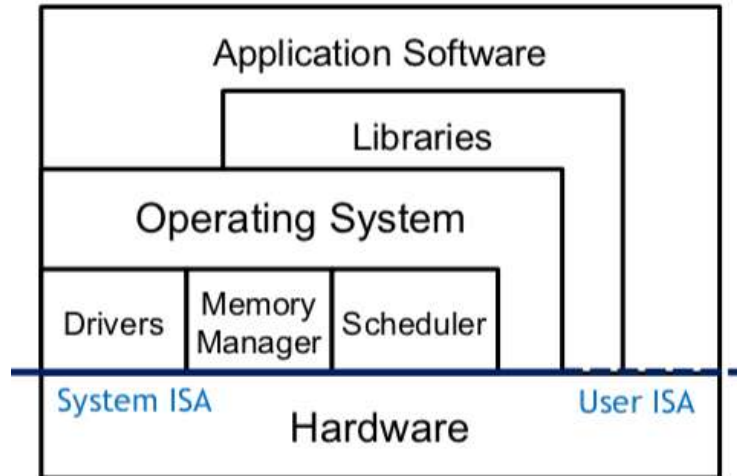


Instructions set architecture (ISA)

ISA is used to connect hardware and software. ISA corresponds to level 2 and each type of machine has its ISA.

It is composed by:

- User ISA: aspects visible to an application program (ex. Read one data from memory, sum..)
- System ISA: aspects visible to supervisor software (OS) which is responsible for managing hardware resources (hide the complexity of CPUs, communicate with the HW, read HW register).

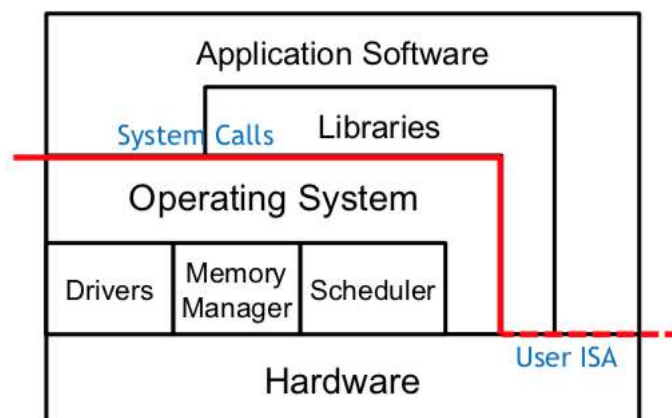


Application binary interface (ABI)

ABI corresponds to level 3 and provides a program access to:

- The hardware resources (via the OS)
- Services available in a system

We have access to system calls but we don't have access to the System ISA. System calls allow programs to interact with shared hardware resources indirectly by OS.



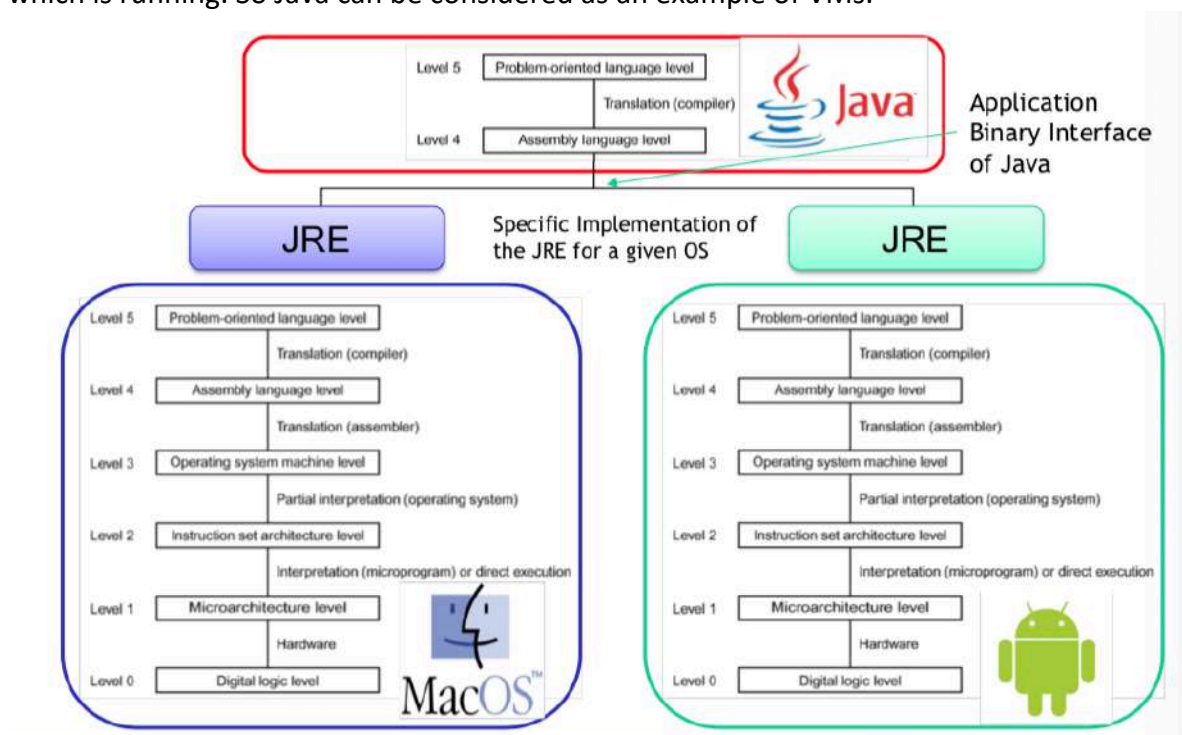
Machine level

One machine level can only run instructions that were meant for it. Nobody will try to run an Android program on Mac OS because for sure it will not work because the ABI is different. However, there are several technologies that allows to run a program on whatever architecture is able to support it (ex. Java).

Java can work on every architecture for which an interpreter, called the Java Runtime Environment (JRE), exists because:

- All I/O devices and peripherals have been standardized (all the computers have screens, keyboard, disks etc...)
- It use a new machine language Java Bytecode that does not use instructions for a specific architecture.

JRE is also addressed as the **Java Virtual Machine**. It depends/interacts on the particular OS in which is running. So Java can be considered as an example of VMs.



Emulation

Emulation is a way of executing a software written for one hardware on a different one. An emulator reads all the bytes included in the memory of system it is going to reproduce. It understand the behavior of the CPU, the RAM and all the I/O devices connected to the emulated system. It reproduces the corresponding behavior in the system where it its running.

Emulator reproduce the machine: not only ABI but also ISA.

Machine Virtualization

Both emulators and JRE can be considered as VMs.

A **Virtual Machine** (VM) is a logical abstraction able to provide a virtual execution environment or a full system environment.

- It provides identical software behavior
- It consists in a combination of physical machine and virtualizing software
- It may appear as different resources than physical machine
- It may result in different level of performances

VMs can be divided in:

- Process VMs



- System VMs



Process VMs

Machine as seen by Processes.

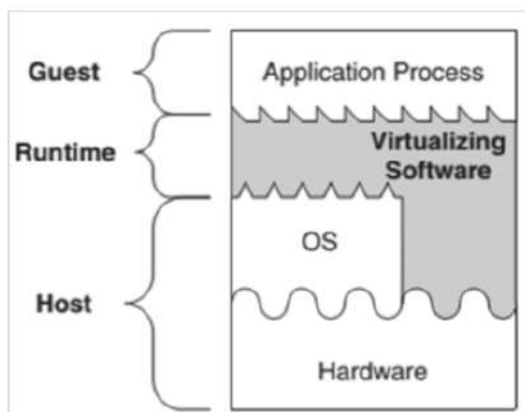
It consists of:

- a logical memory address space assigned to the process
- user-level registers
- the interpreter that allows the execution of the application code
- the I/O is visible only through the OS

The virtualizing software called **Runtime Software** is placed at the ABI interface, on top of the OS/hardware combination and it emulates both user-level instructions and operating system calls. The ABI provides the interface between the process and the machine.

The **ABI** provides the interface between the process and the machine.

Process Virtual Machine



System VMs

Machine as seen by OS

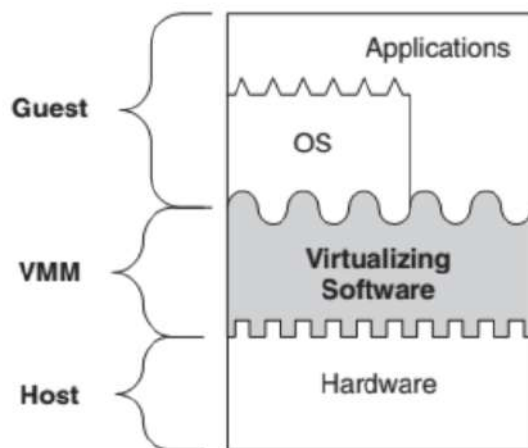
A system (full execution environment) is supported by the underlying machine:

- it simultaneously support different user processes/applications running at the same time. All processes share file system and I/O resources.
- It allocates physical memory and I/O resources to the processes that interact with their resources via an OS that is part of the system.
- It can be either implementd on the underlying hardware alone, or on top of another OS.

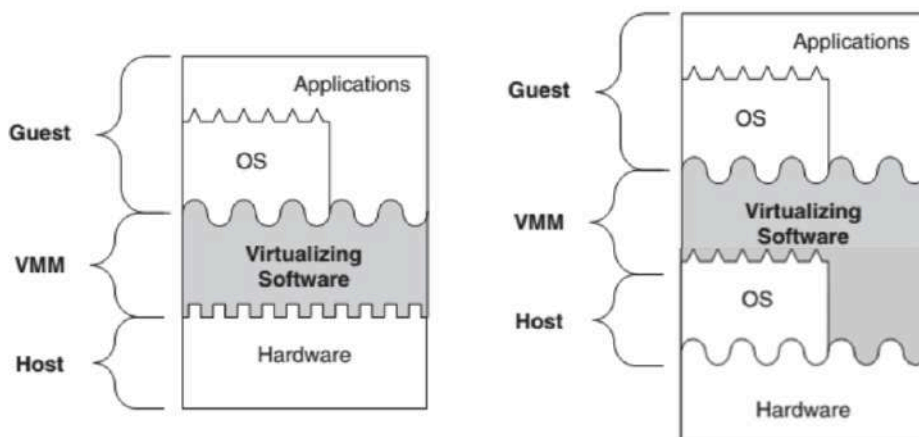
It provides a complete system environment that can support an operating system running in it access to underlying hardware resources. The virtualization software that emulates the ISA interface is called **Virtual Machine Monitor (VMM)**

The **ISA** provides the interface between the system and the machine

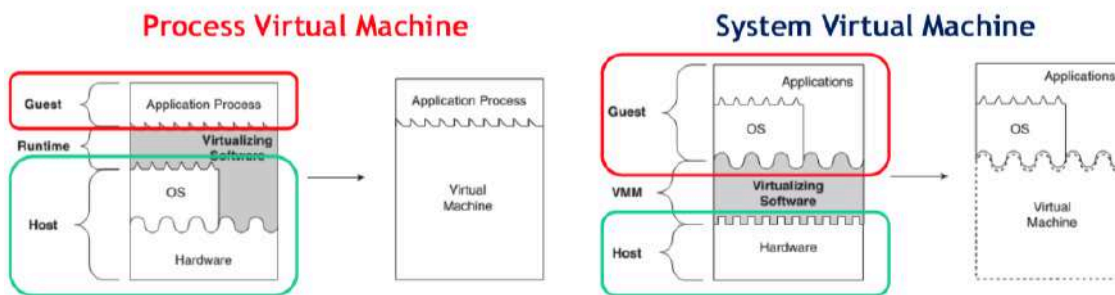
System Virtual Machine



The VMM can provide its functionality either **working directly on the hardware**, or **running on another OS**.



Terminology (Host and Guest)



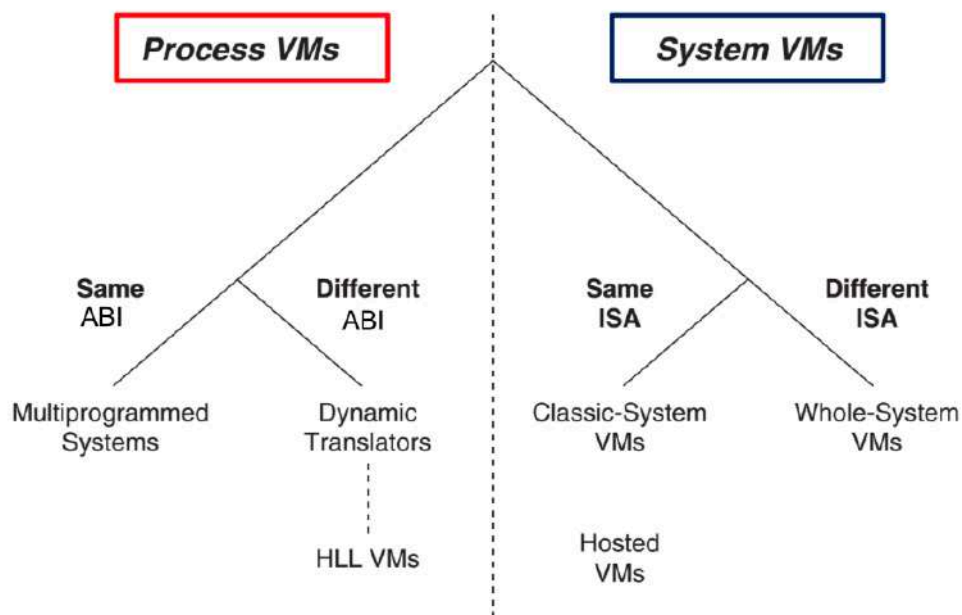
what is physical

Host: the underlying platform supporting the environment/system

Guest: the software that runs in the VM environment as the guest.

what is executed in VM

Different types of virtualization



Multiprogrammed Systems

Each user process is given the illusion of having a complete machine to itself and they have their own address space and file structure. OS timeshares and manages HW resources to permit this.

Emulation

Refers to those software technologies developed to allow an application to run in an environment different from that originally intended. It is required when the VMs have a different ISA / ABI from the architecture where they are running on.

We can distinguish two types of emulation:

- **Interpretation:** interpreter program fetches, decodes, emulate the execution of individual source instruction (can be a slow process)
- **Binary Translation:** blocks of source instructions are converted at run time. It has an high overhead on the translation process, but once a block is translated the performance are high and much faster than the interpreter.

High-Level Language VM

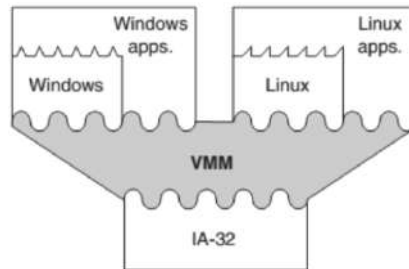
Here the goal is to have an isolated execution environment for each application.

So, applications run normally but are sandboxed and can “migrate, they are not conflicting one another and are not installed in a strict sense. They are just meant to run in a virtual environment.

Example: Java Virtual Machine

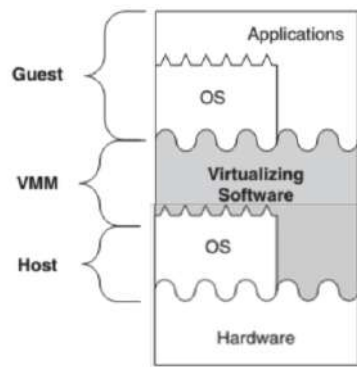
System VM for same ISA

The VMM is on bare hardware and virtual machines fit on top. It can intercept guest OS's interaction with hardware resources. This is the most efficient VM architectures and can handle two different Oss on the same HW.



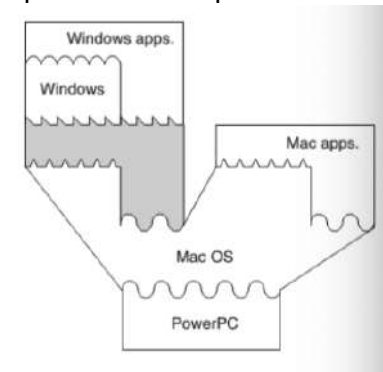
Hosted VM

Virtualizing software is on top of an existing host operating system.



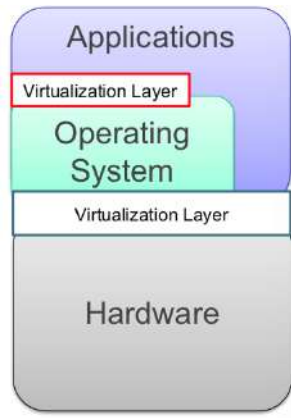
Whole-System VMs

It virtualizes all software. ISAs are different so both application and OS code required emulation via binary translation. It usually implements the VMM and guest software on top of a conventional host OS running on the hardware. The VM software must emulate the entire hardware environment and all the guest ISA operations to equivalent OS call to the Host.



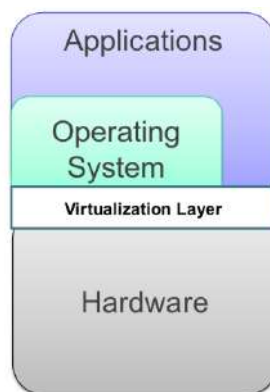
How is Virtualization implemented?

Given a typical layered architecture of a system it is implemented by adding layers between execution stack layers. Depending on where the new layer is placed, we obtain different types of Virtualization.



Hardware-level virtualization:

Virtualization layer is placed between hardware and OS. The interface seen by OS and application might be different from the physical one.



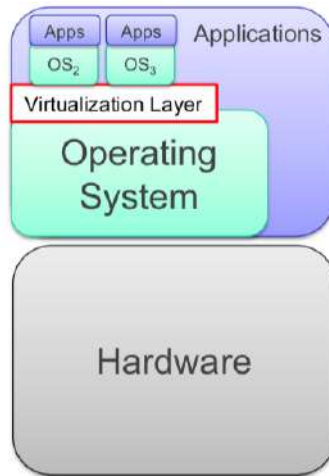
Application-level virtualization

Virtualization layer is placed between the OS and some applications (ex. JVM). It provides the same interface to the applications. Applications run in their environment, independently from OS.



System-level virtualization

Virtualization layer provides the interface of a physical machine to a secondary OS and a set of application running in it, allowing them to run on top of an existing OS. It is placed between the system's OS and other OS enabling several Oss to run on a single HW.



Virtual Machine Manager

An application that:

- Manages the virtual devices
- Mediates access to the hardware resources on the physical host system
- Intercepts and handles any privileged or protected instructions issued by the virtual machines

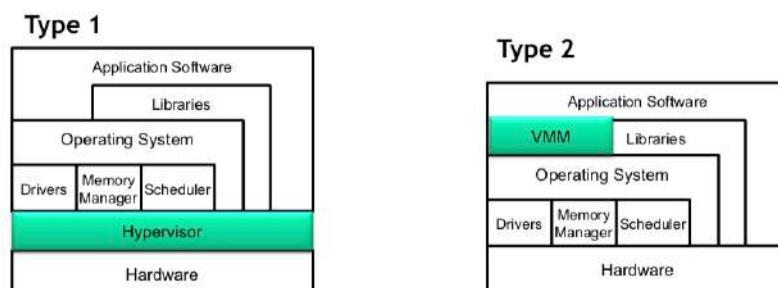
This type of virtualization typically runs virtual machines whose OS, libraries and utilities have been compiled for the same type of processor and instruction set as the physical machine on which the virtual systems are running.

These are three terms to identify the same thing:

- Virtual Machine Manager
- Virtual Machine Monitor
- Hypervisor

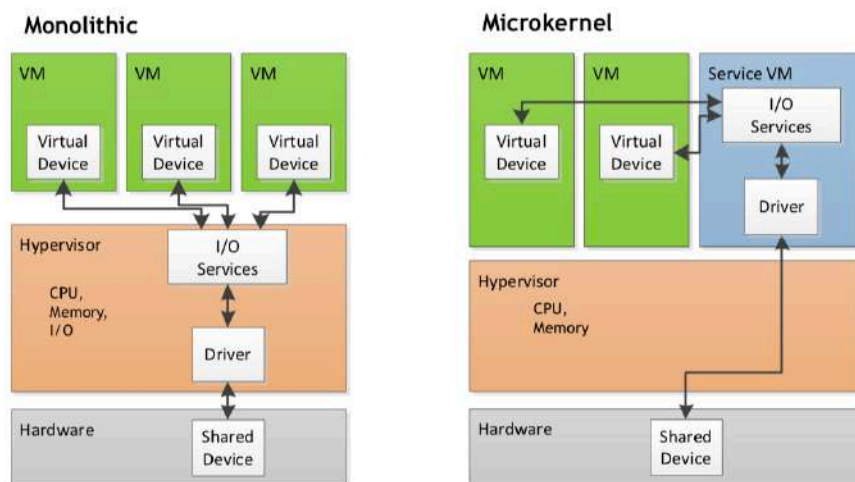
We may distinguish two types of Hypervisors

1. Bare-metal (type 1)
Takes direct control of the hardware
2. Hosted (type 2)
Reside within a host operating system and leverage code



Type 1 hypervisor

- Monolithic
 - Device drivers compiled and run within the hypervisor
 - + Better performance because it's already compiled
 - + Better performance isolation because hypervisor can manage it all
 - - Can run only on hardware for which the hypervisor has driver
- Micro-kernel
 - Device drivers run within a service VM
 - + Smaller hypervisor
 - + Leverages driver ecosystem of an existing OS
 - + Can use 3rd party driver
 - - Less performance because we need an additional service



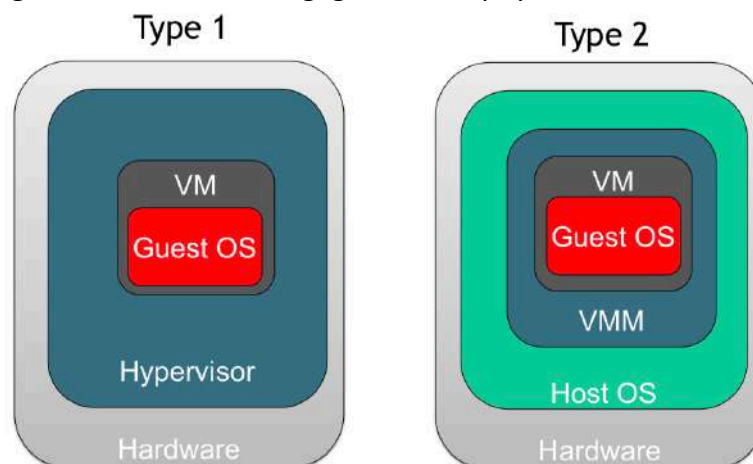
Type 2 hypervisor

Characterized by at least two OSs running on the same hardware:

- The Host OS controls the hardware of the system
- The Guest OS is the one that runs in the VM

The VMM runs in the Host OS while applications run in the Guest OS.

- + Most flexible in terms of underlying hardware
- + Simpler to manage and configure
- - Special care must be taken to avoid conflict between Host OS and Guest OS
- - The Host OS might consume a non-negligible set of physical resources

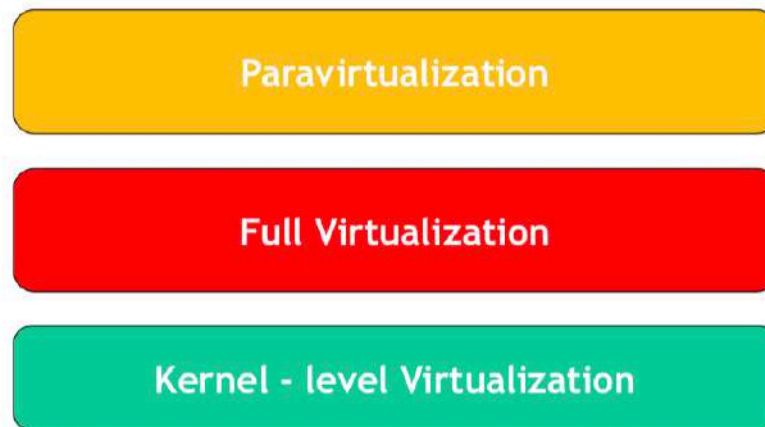


Virtualization technologies: properties

- **Partitioning**
Execution of multiple OSs on a single physical machine. Partitioning of resources between the different VMs.
- **Isolation**
Fault tolerance and security. Advanced resource control to guarantee performance.
- **Encapsulation**
The entire state of a VM can be saved in a file (freeze and restart the execution), that can be copied and moved (as all other files 😊)
- **HW-independence**
Provisioning/migration of a given VM on a given physical server

Virtualization techniques

Different ways to implement system level / same ISA virtualization:



Paravirtualization

Guest OS and VMM collaborates:

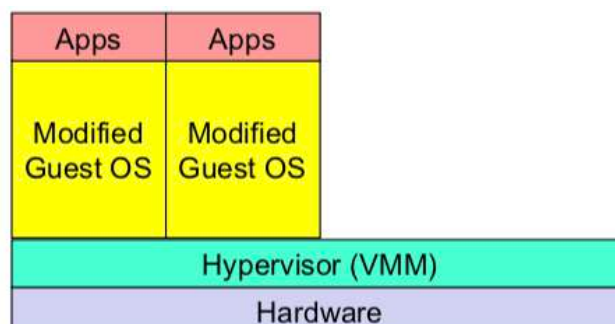
- VMM present to VMs an interface similar but not identical to that of the underlying hardware
- To reduce guest's executions of tasks too expensive for the virtualized environment

Pros:

- Simpler VMM
- High Performance

Cons:

- Modified Guest OS



Full Virtualization

Provides a complete simulation of the underlying hardware:

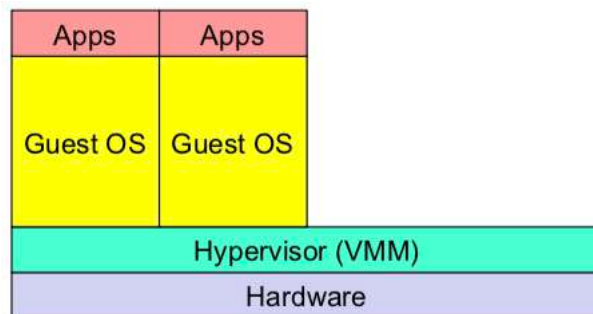
- The full instruction set
- Input/output
- Interrupts
- Memory access

Pros:

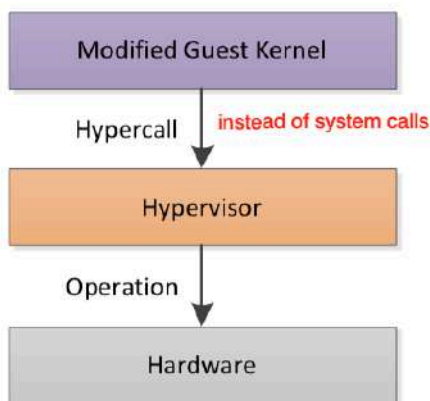
- Running unmodified OS

Cons:

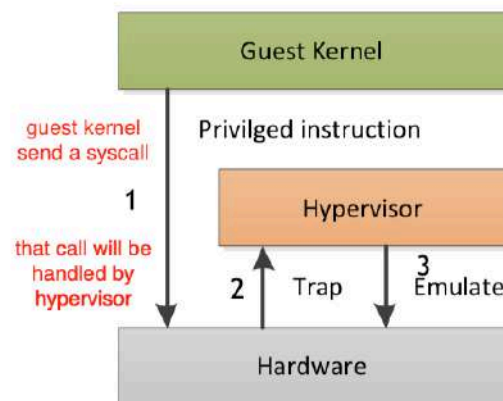
- Hypervisor mediation (reduce performance)
- Requires some hardware support (not on every architecture)



Para-virtualization



“Classical” Full-virtualization



Kernel-level Virtualization

Virtualization happens at OS-level. No guest OS but private Servers virtualized on top of the operating system instead.

Pros:

- Native OS performances
- Native HW support through host
- No VMM or Hypervisor

Cons:

- Single OS
- Host Kernel modifications required

Virtualization B

Virtual Resources

The main contribution of a VMM is handling virtual resources.

- Present to the Guest OS the physical interface for a real resource
- Understand the requests issued by the Guest OS to the resource
- Emulate the requests on the Host hardware

Virtualization layer maps virtual resources to a set of managed resources that are often physical resources, such as a disk, or may be virtual resources when multiple virtualization layers are stacked.

Virtual resources may provide

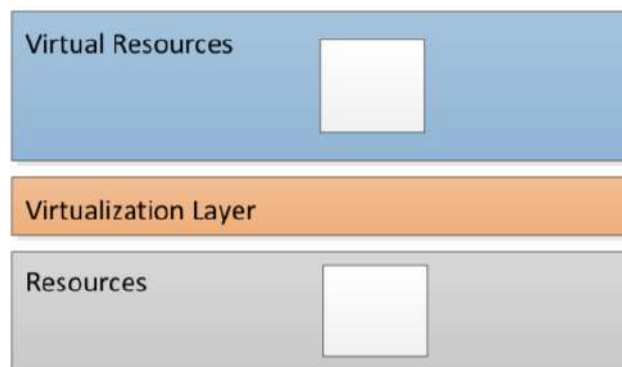
- the same interface of physical resources (**virtualization**)
- a different interface (**emulation**)

Virtualization

Resource mapped to a virtual resource with the **same interface**

+ Quality of performance

- Not always possible

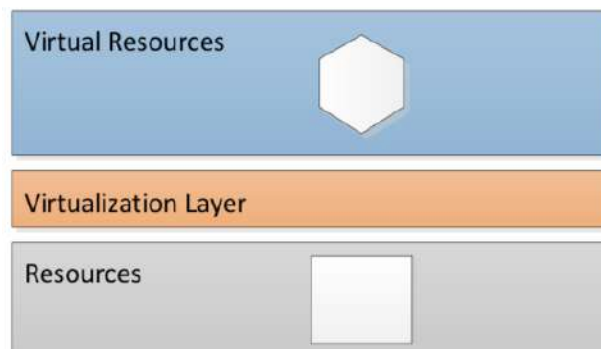


Emulation

Present clients resources with a **different interface**

+ Compatibility, interoperability, flexibility

- Limited performances



Mapping

The mapping can be:

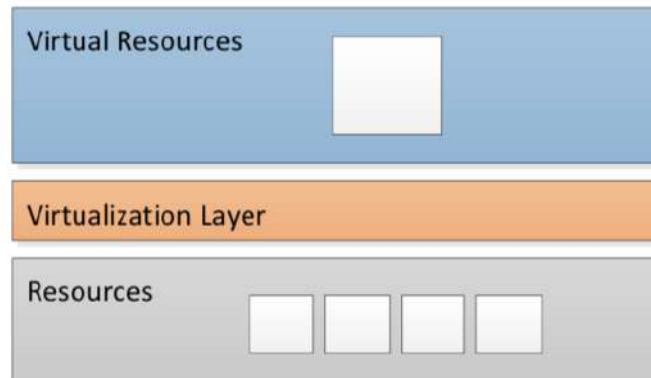
- One virtual resource corresponds to a physical resource
- One virtual resource and many physical resources ([aggregation](#))
- Multiple virtual resources and a single physical resource ([sharing](#))

Aggregation

Pool of resources presented as a single one.

Benefits: scalability, reliability, simplification

Example: RAID (OS sees the array as a single block device)

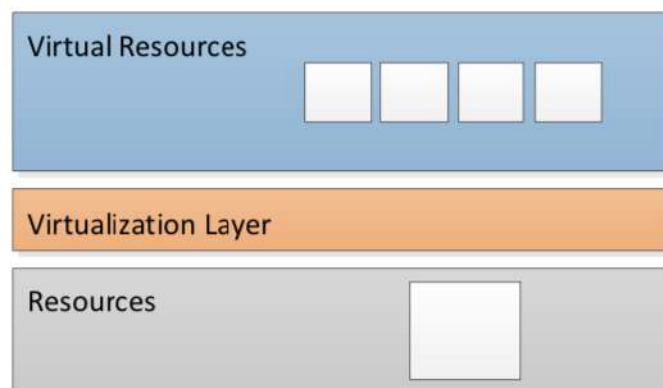


Sharing

Present multiple, smaller resources multiplexing access to a single resource. Time or space multiplexing (partitions of one disk).

Benefits: isolation, flexibility, resource management

Example: sharing a printer between several PCs



Virtual Resources of a System VM

- Processor
- Memory
- I/O devices

Processors

Must be able to execute the guest instructions (system and user-level).

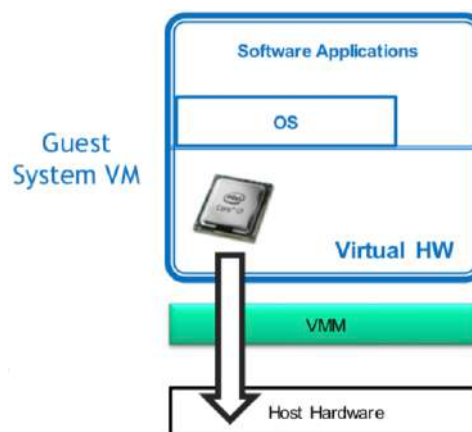
It has two ways to accomplish this:

- Emulation (different ISA)
Interpretation or Binary translation
- Direct Native Execution (same ISA)
Better performances

CPU Virtualization is based on the fact that each VM can be assigned a number of physical cores of the physical machine.

Although in principle a VMM can simulate more cores than the one actually available, this is a bad practice because more virtual cores than physical cores can be only achieved by time-sharing a core, and it can only reduce performance. However, a VMM can simultaneously run several VMs using a total number of cores greater than the one available on the physical machine.

We can set an *execution cap* to define the time a core is assigned to a VM, it specifies the percentage of time the VM can run. This can be used to limit the CPU resources assigned to a VM, allowing to run a larger number of VMs on the same PM. However, it can create timing problems on the Guest OSs.



Memory

The VMM can assign a portion of the physical memory to each VM. A good practice is to make sure that the sum of the memory allocated for each VM is less than the total memory available on the system. This however can lead to poor physical memory utilization. For this reason, VMM allows to assign VMs a total quantity of memory larger than the one physically available.

Memory ballooning and *overcommit* are two techniques to assign VMs more memory than available physical RAM.

- **Memory Ballooning**

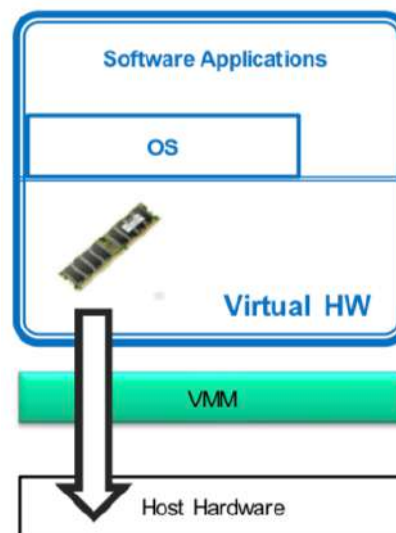
Requires support from the Guest OS (paravirtualize).

When VMs requires more memory than available pages, the VMM interrogates the Guest on the VMs. If there are pages no longer used, they are returned to the VMM to be assigned to the VMs in need of memory.

- **Memory Overcommit**

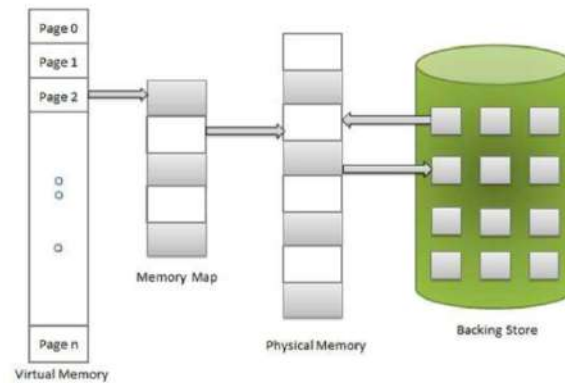
No support for the guest OS is required.

The VMM tries to determine the pages that are less used, to use their space (guessing how much memory the OS is using).

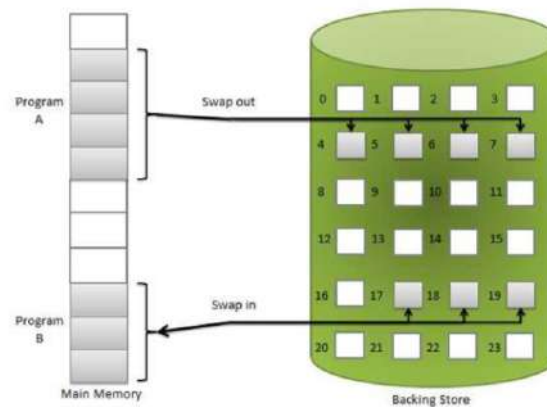


Conventional OSs usually exploit a technique called *Virtual Memory* to better use the available RAM. Virtual memory uses part of the storage space to increase system's memory, allowing the execution of processes which are not completely loaded into ram. The main visible advantage is that programs can be larger than physical memory. It separates user logical memory from physical memory, and this separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available.

Memory is divided into pages of equal size (which must be a power of two). Only a fraction of the pages is in RAM, the other is allocated on a storage device (a disk partition or a file called “**swap**”).

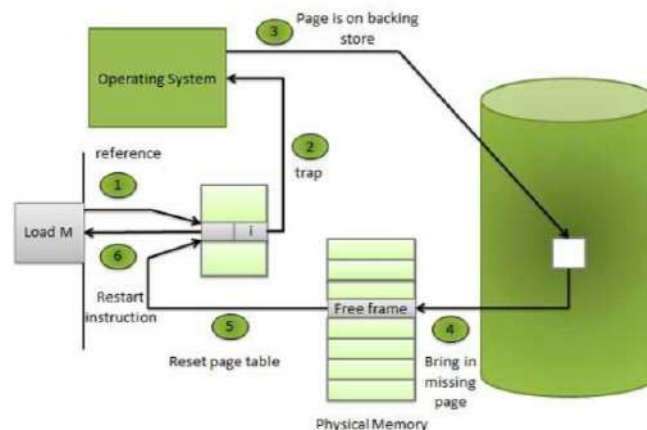


CPU can only access data on pages that are in RAM. Pages must then be moved on the disk and reloaded from the disk when needed.



Virtual memory is supported by the CPU: it uses the trap technique to identify when a process tries to access a portion of memory that is not in RAM.

The OS must determine which page on RAM store on disk to free space, and which page load to satisfy the request (this action is called **swap**)

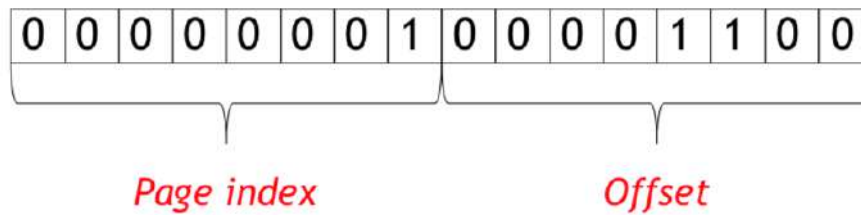


However trap alone is not enough to support virtual memory. The CPU must also have a special unit called **Memory Management Unit (MMU)**. The MMU allows to separate the process address space from the physical address space.



The binary address of each memory cell accessed by the CPU is divided in two parts:

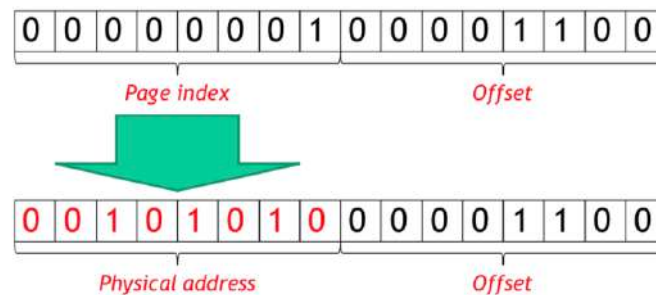
- Page index: indicates the memory page.
- Offset: contains the offset from the start of the page.



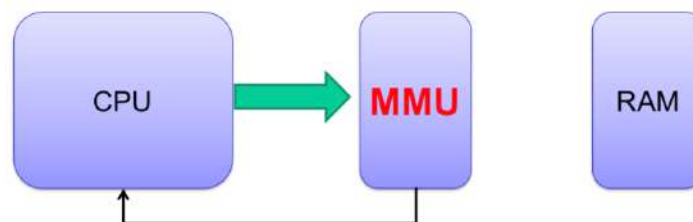
The MMU has a special data structure that is called the **Page Table**. It contains for each page index at least the following data:

- Physical address of the page.
- A flag that tells whether the page is in RAM or disk.
- A flag that tells if the page has been changed from the last saving.
- A flag that tells if the page is free or if it is being used.

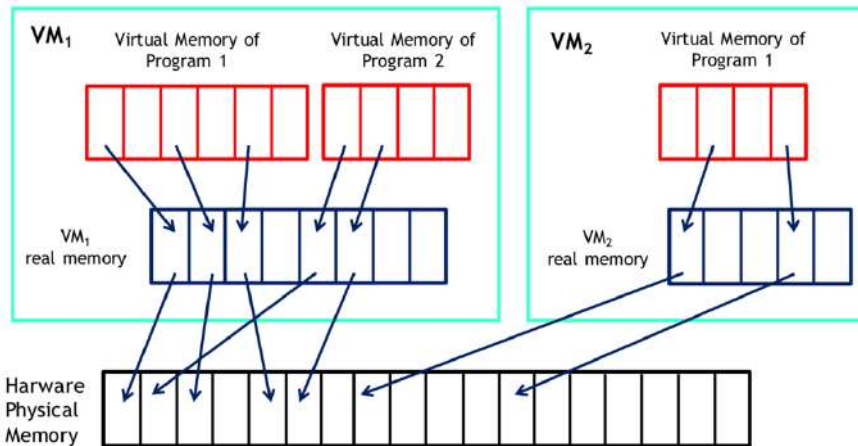
The MMU looks at the page index in the table. If the page is in RAM, it replaces the page index with the hardware address on the RAM.



If the page is not in RAM, the MMU causes a trap, stopping the execution of the program, and calling the OS to load the page from the disk. When the page has been loaded, the page table is updated, and the execution of the application continues from the point in which it was stopped.



In a system VM environment, each of the guest VMs has its own set of virtual memory tables.



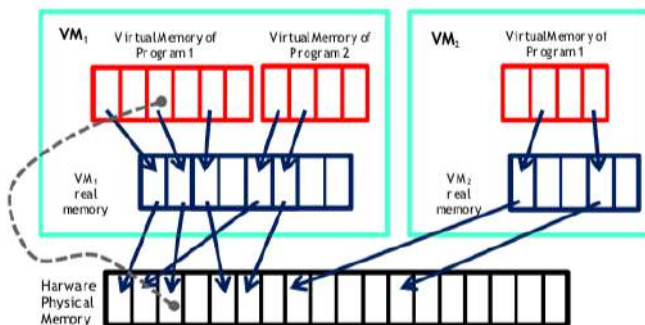
The addition of another level in the memory hierarchy necessitates additional mechanisms for memory management. In particular, there must be consistency and synchronization between Host and Guest page tables.

Two main approaches are used:

- Shadow pages (SW solution)
- Nested pages (HW assisted)

Shadow Page Tables

The VMM maintains them, one for each of the guest VMs, to map Virtual Memory Pages of VMs to physical memory of the host HW. These tables are the ones actually used by hardware to translate virtual addresses. VMM keeps Shadow Pages Table in sync with Guest OS Page Table, to allow guest processes to access proper host HW addresses. This synchronization process introduces a lot of overhead each time Guest OS updates its Page Table.



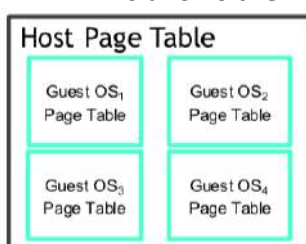
Eliminates one level of indirection jumping from the virtual to the physical without passing through the real memory.

Nested Pages

Nested paging removes the overheads associated with shadow paging. It requires HW support and an additional table (**NPT**) that is used to translate guest real addresses to system physical.

The TLB is able to cache this 2-step translation:

- A new tag is added to the TLB called ASID (address space identifier)
- This allows the TLB to keep track of which TLB entry belongs to which VM.



I/O devices

A number of different types of I/O devices:

- Display
- Keyboard
- Mouse
- Speakers

If required, VMM construct a virtual version of the device, and the I/O activity directed at the device is intercepted by VMM and converted to equivalent request for underlying physical device.



Virtual Disks

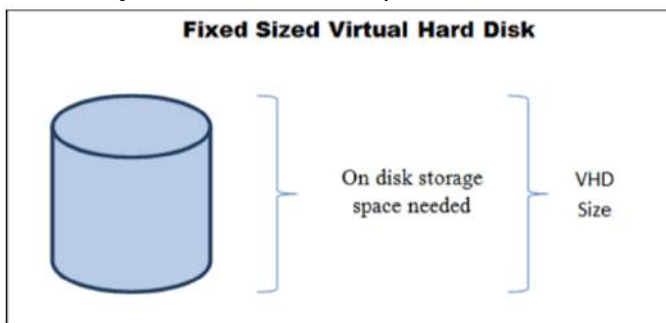
Virtual disks save the entire content of an HDD into a file.

There are several different formats:

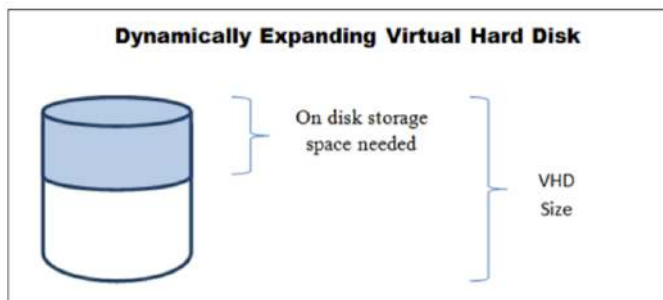
- VDI – The format for VirtualBox
- VMDK – The format of VMware
- VHD – The format used on Windows Virtual PC

Formats are different from each other, however they all share some similarities

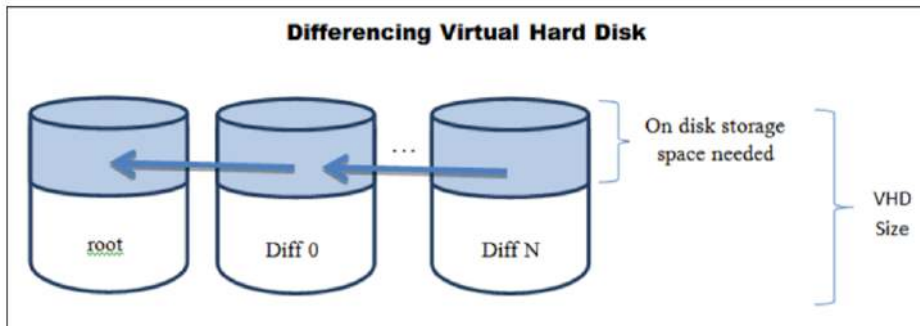
Fixed size formats immediately create a file whose size corresponds to the size of the HDD.



Variable size formats create disks that start with a reduced size and can grow up to the total HDD size. Compression is obtained by dividing the space into blocks, and by storing only the used ones.



All formats allow creating **snapshots**: save the content of the disk at a given point in time to allow to return to it when required. After a snapshot as been created, all formats store just the differences from the previous snapshot.



Network Virtualization

Network virtualization must allow all the VMs plus the Host to share the same connection, and to operate as separate entities. Since all VMs are operated and controlled through the network, this increase the importance of these virtualized components. A VMM can attach to a VM several virtual network cards, each one characterized by its own configuration. Each virtual network cards is addressed as **Virtual Network Adapter (VNA)**.

If more than one VNA are connected, the virtualized system can perform routing among the cards, and accept connections through different interfaces.

There are many ways in which a VNA can operate:

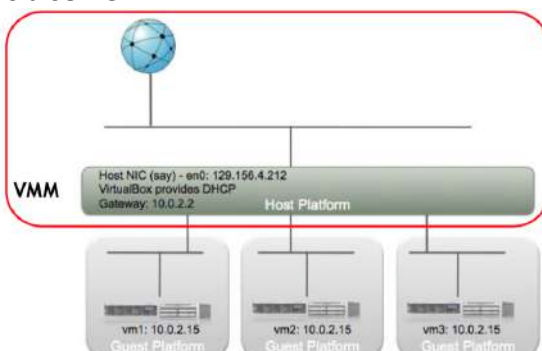
- Network Address Translation (NAT)
- Bridged networking
- Host networking
- Internal networking

Network Address Translation

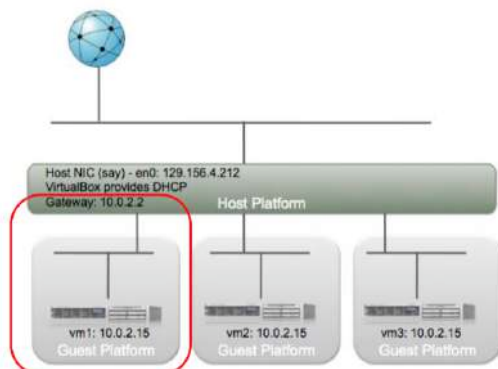
This is the simplest way of accessing an external network from within a virtual machine. A virtual machine with NAT enabled acts much like a real computer that connects to the Internet through a router.

- Each VM as a different private virtual network that connects it just to the host OS.
- The router is in the VMM networking engine, placed between each virtual machine and the host.
- VMs can easily access the external network.
- Port forwarding is needed to let VM be visible and reachable.

From the **Host side**, the VMM is treated has any other application that can acts either as a client or as a server.



From the **Guest side**, the OS sees a virtual network, with a gateway, a DNS and a given subnet mask.

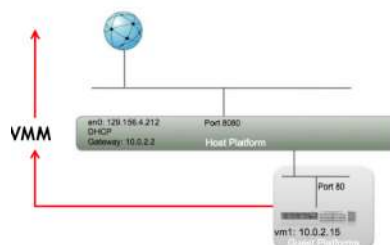


The VMM simulates the presence of a DHCP server, that always assigns to the Host VM the same address. The actual address assigned, depends on the virtual network card to which the considered NAT-type network is attached.

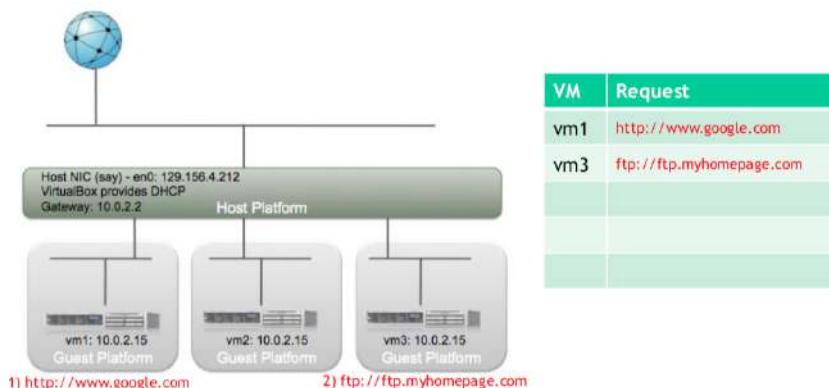
| | IP | Gateway | DNS |
|-------|-----------|----------|----------|
| VNA 1 | 10.0.2.15 | 10.0.2.2 | 10.0.2.3 |
| VNA 2 | 10.0.3.15 | 10.0.3.2 | 10.0.3.3 |
| VNA 3 | 10.0.4.15 | 10.0.4.2 | 10.0.4.3 |
| VNA 4 | 10.0.5.15 | 10.0.5.2 | 10.0.5.3 |

VirtualBox example where the range of the DHCP server, and the behavior of the DNS can be usually changed using VMM specific configuration tools.

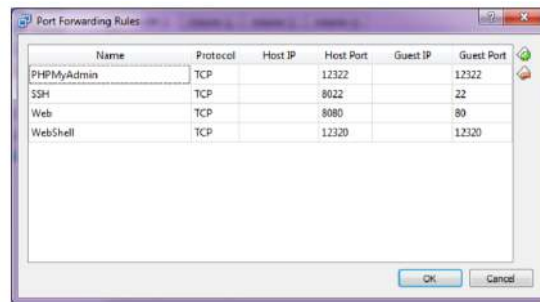
All the traffic that is generated inside the VM, is repeated by the VMM, as it was originated by itself.



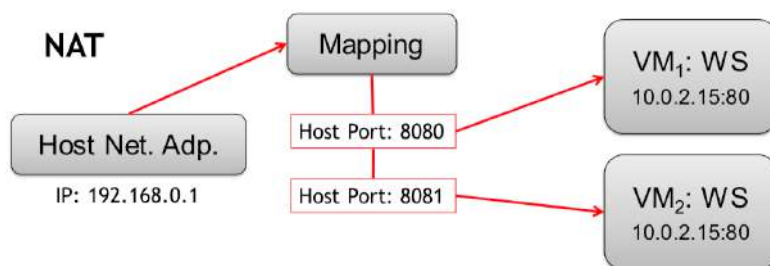
Each time a VM sends a request, the VMM stores in an internal table from which VM it was originated. When the response arrives, the VMM looks back into this table to find the appropriate VM to which forward the arrived packet.



In order to contact a server on the Guest OS, port forwarding must be enabled. Port forwarding creates a correspondence between a free port on the Host OS and a port on the Guest OS.



The VMM acts as a server on the corresponding port: all the packets directed to the Host port, are forwarded to the Guest port of the corresponding VM. The same port on two different VMs must then be mapped to two different ports on the Guest OS. **Each port on the Host can be used by only a single VM.**



- The address used to access the WS on the VM1 is 192.168.0.1:8080
- The address used to access the WS on the VM2 is 192.168.0.1:8081

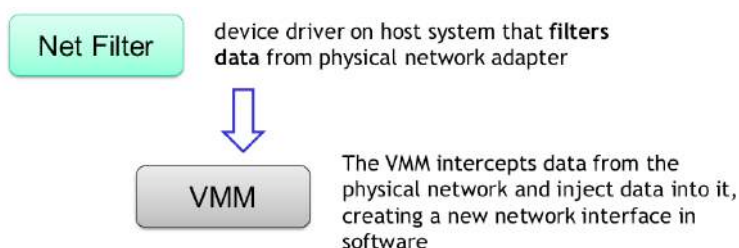
This method consumes PORTS.

Bridged networking

Bridged networking allows the Guest to send and receive packets directly from a Physical Network Adapter on the Host. It exploits the fact that a physical network card can be characterized by multiple IP addresses on the network to which it connects.

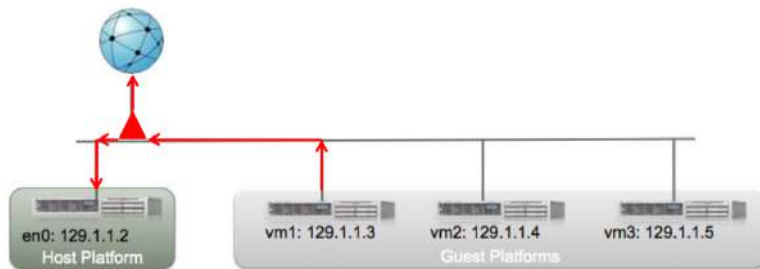
- The VM seems to be actually connected to the real network
- No port-forwarding should be setup
- A DHCP should be present, or static addresses should be assigned

It uses a special network feature of the Host OS called *Net Filter*:

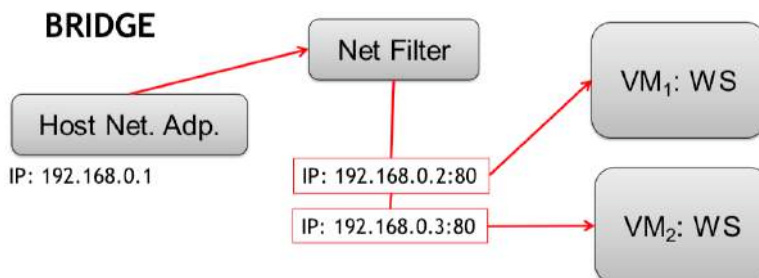
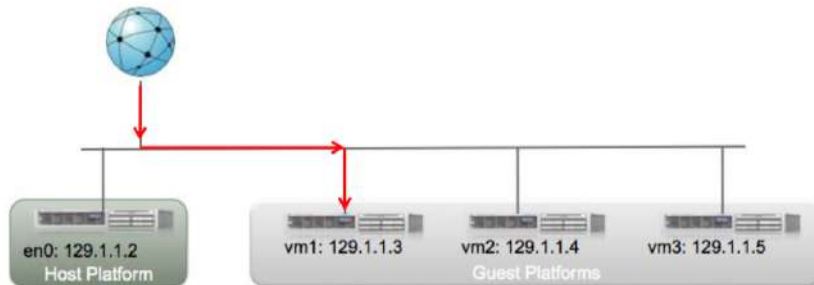


To the host system guests behave as physically connected to the interface using a network cable. To configure a VNA in Bridged mode, a real Network Adapter connected to the Host machine must be selected, in order to specify the network card on which Net Filter operates and the subnetwork in which VMs will be placed.

Traffic produced from inside the VMs, thanks to the Net Filter interface is sent both on the network connected to selected adapter, and to the Host machine. The Host machine in this way sees the traffic produced from inside a VM as coming from the associated Network adapter.



Traffic coming from the outside, can be directed to the corresponding VM, exploiting the fact that the physical network adapter is characterized by all the IP addresses of the host and the VMs running on the system.



I can access both the WS at their own IPs. Both WSs cannot have the same IP (but might have the same port)

This method consumes IPs

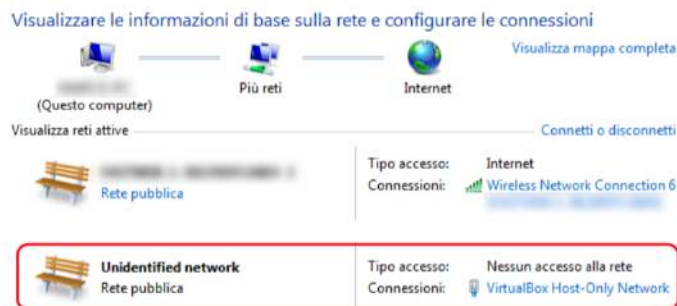
In bridged mode the VMM does not provide a DHCP service. However, since Host OS are connected directly to an external network, they can obtain IP address from a real external DHCP server.

Host-only networking

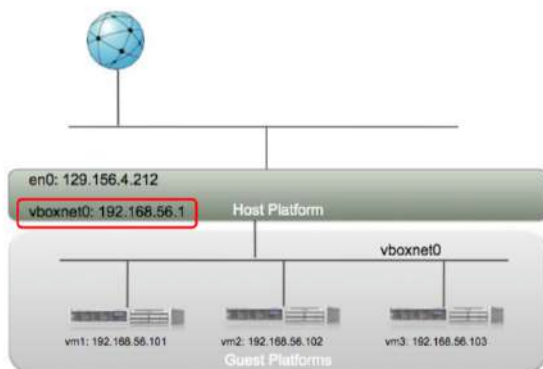
Host-only Networking allows the connection between the Host and the Guest VMs only.

- VM are not directly connected to the external network.
- It is more secure
- Routing mechanisms should be configured on the Host to allow guests to connect to the external network

The Host has one special virtual network adapter (per host-only network) to the Host OS, which behaves exactly as any other network driver for a physical connection.



The host and the VMs can communicate among each other using this special virtual network card.



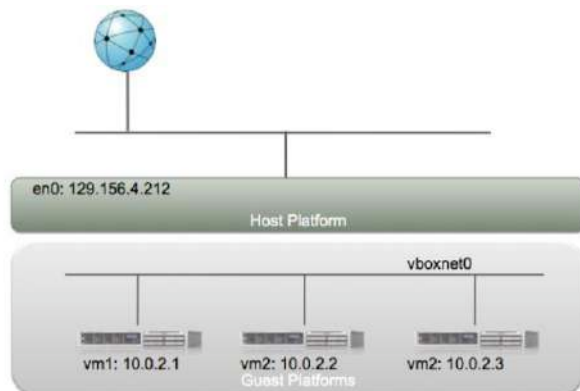
The system can have more than one virtual host adapter: the one to which the VM will be connected must be selected during its configuration. If needed, a virtual DHCP server can be added to automatically configure the parameters of the VNA on the Guest OS.

Internal Networking

Internal Networking creates a network that connects only the VMs, and it is not reachable by outside.

- The most secure
- The most limited
- VMs cannot contact a real server
- Useful to test complex network configuration

The network in this case is created only between the VMs: they cannot communicate either with the Host or with the outside world. The host OS can be contacted only through keyboard, mouse and graphic adapter emulation of the VMM.



More than one virtual network can be usually defined: each one can be identified by a different name.

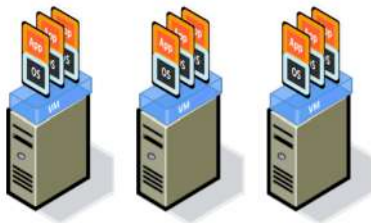
Virtualization and Cloud

Server Consolidation

Without Virtualization, software would be strongly linked/related with hardware so that moving/changing an application would not be an easy task. This implies low flexibility.



With Virtualization, software and hardware are no longer strongly related, this corresponds to a high flexibility thanks to pre-built VMs. OS and applications can be handled as a single entity.



Consolidation Management: migrating from physical to virtual machines.

Scalability: It is possible to move virtual Machines, without interrupting the applications running inside.

Automatic Scalability: It is possible to automatically balance the Workloads according to set limits and guarantees.

High Availability: servers and applications are protected against component and system failure.

Advantages of consolidation:

- Different OS can run on the same hardware
- Higher hardware utilization (less hardware is needed, green IT-oriented)
- Continue to use legacy software (software for WIN on Linux machines thanks to VMs)
- Application independent from the hardware

Cloud Computing

Cloud computing is a computing model in which resources (CPU and storage) are provided as general utilities that can be leased and released by users through the Internet in an on-demand fashion.

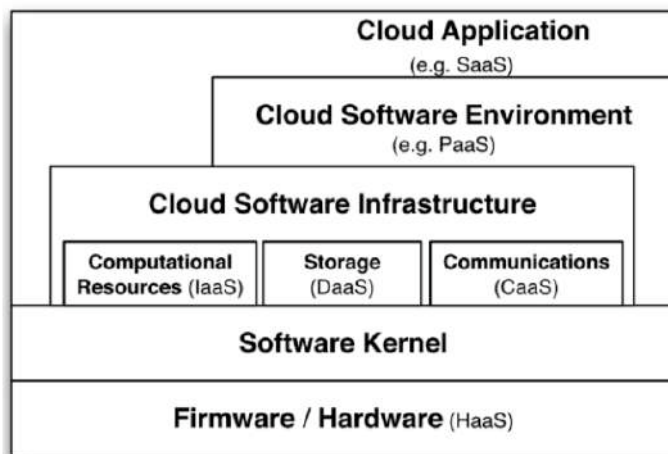
Cloud computing is a model for enabling convenient and on-demand network access to a shared pool of configurable computing resources, like for example:

- Networks
- Servers
- Storage
- Applications
- Services

That can be rapidly provisioned and released with minimal management effort or service provider interaction.

We can distinguish six models of services:

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)
- Data Storage as a Service (DaaS)
- Hardware as a service (HaaS)
- Communication as a Service (CaaS)



Cloud Application Layer (SaaS)

Users access the services provided by this layer through web-portals and are sometimes required to pay fees to use them. Cloud applications can be developed on the cloud software environments or infrastructure components.

Example: GMail, Google Docs, SalseForce.com

Cloud Software Environment Layer (PaaS)

Users are application developers. Providers supply developers with a programming-language-level environment with a well-defined API. This facilitates interaction between environment and apps, accelerating the deployment and it also supports scalability.

Examples in Deep Learning: Amazon Machine Learning, Google AI: Tensor Flow

Cloud Software Infrastructure Layer (IaaS, DaaS, CaaS)

Provides resources to the higher-level layers.

IaaS:

Virtual Machines (VM) vs dedicated hardware

VM's benefits:

- Flexibility
- Super-user access to VM for fine granularity settings and customization of installed SW

VM's issues:

- Performance Interference
- Inability to provide strong guarantees about SLAs

Example: Amazon Elastic Cloud (EC2), Microsoft Azure

DaaS:

Allows users to store their data at remote disks and accessing it anytime from anywhere. It facilitates cloud applications to scale beyond their limited servers' requirements:

- High dependability: availability, reliability, performance
- Replication
- Data consistency

Example: DropBox, iCloud, GoogleDrive

CaaS:

Communications becomes a vital component in guaranteeing QoS.

Requirements:

- Service-oriented
- Configurable
- Schedulable
- Predictable
- Reliable
- Network security
- Network monitoring

Software Kernel Layer

Basic SW management for the physical servers:

- OS Kernel
- Hypervisor
- VMM
- Clustering middleware

Firmware/Hardware layer (HaaS)

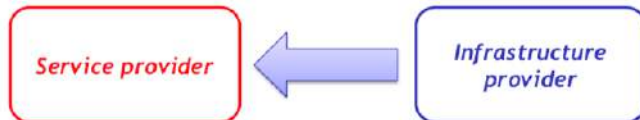
Physical HW (servers, switches...)

Users of this layer are big enterprise with huge IT requirements.

HaaS provides, operates, manages and upgrades the HW on behalf of its customers, for the life-time of the sublease.

Users and providers

In a cloud computing environment, the traditional role of service provider is divided into two:

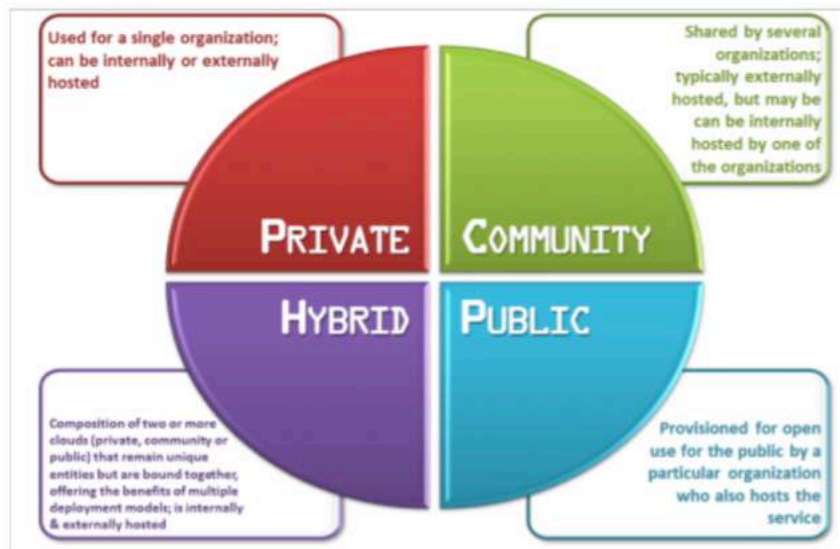


- *Infrastructure providers* who manage cloud platforms and lease resources according to a usage-based pricing model
- *Service providers* who rent resources from one or many infrastructure providers to serve the end users.

Due to the different layers of services, the infrastructure provider can be a user itself of a lower-level service provider.



Types of Cloud



Public Clouds

Large scale infrastructure available on a rental basis

- OS virtualization provides CPU isolation
- “Roll-your-own” network provisioning provides network isolation

Full customer self-service

- Service Level Agreements (SLAs) are advertised
- Requests are accepted and resources granted via web services
- Customers access resources remotely via the Internet

Accountability is e-commerce based

- Web-based transaction
- “Pay-as-you-go” and flat-rate subscription

Private Clouds

Internally managed data centers. The organization sets up a virtualization environment on its own servers. Useful for companies that have significant existing IT investments.

Key benefits:

- You have total control over every aspect of the infrastructure
- You gain advantages of virtualization

Issues:

- It lacks freedom from capital investment and flexibility.

Community Clouds

A single cloud managed by several federated organizations allowing economy of scale. Resources can be shared and used by one organization, while the others are not using them.

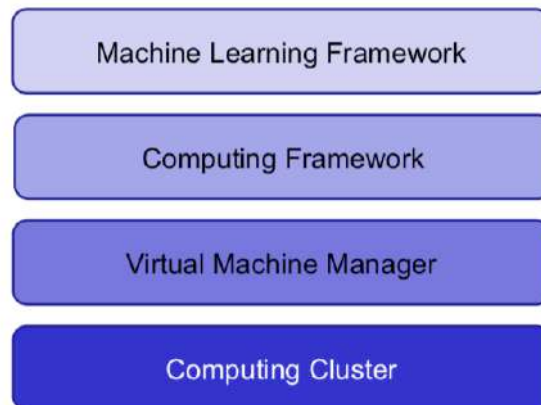
Technically similar to private cloud because they share the same software and the same issues, however a more complex accounting system is required.

Can be hosted locally or externally. Typically, community clouds shares infrastructures of the participants, however they can be hosted by a separate specific organization.

Hybrid Cloud

Hybrid Clouds are the combination of the previous types. Usually are companies that holds their private cloud, but that they can be subject to unpredictable peaks of load. In this case, the company rents resources from other types of cloud.

Machine Learning as a Service



Computing Cluster

- Servers (with parallel and scalability properties) like GPUs
- Storage (multiple storage systems included distributed/parallel file systems)
 - Direct Attached Storage
 - NAS
 - SAN
- Network (applications are generally non-iterative) -> Ethernet

Virtual Machine Manager

- Virtualization is carried out through hypervisor or containers.
- Resources are increased by adding more virtual machines.
- User can design personalized software environments and scale instances to the needs.

Computing Framework

Computing Frameworks are composed by several modules:

- Cluster Manager
- Data Storage
- Data Processing Engine
- Graph Computation
- Programming Languages (Java, Python, Scala, R)

An application operating in a cluster is distributed among different computing (virtual) machines.



Machine/Deep Learning Frameworks

Machine Learning frameworks cover a variety of learning methods for classification, regression, clustering, anomaly detections and it may or may not include neural network methods

Deep Learning frameworks cover a variety of neural network topologies with many hidden layers.

| | Computing Framework | Stand-alone (OS) | |
|------------------|------------------------------------------|-------------------------------------------------------------------------|--------------------|
| Machine Learning | Spark Mllib, BigDL, Mahout | Scikit-learn, Torch, Pandas, Numpy, Matplotlib | |
| Deep Learning | TensorFlowOnSpark, Deeplearning4j, BigDL | Tensorflow, Caffe, Apache MXNet, Keras, Theano, Microsoft CNTK, Pytorch | Exploit GPU access |
| | | Generally organized as libraries/shells | |

Why Machine Learning in the Cloud?

Cloud computing simplifies the access to ML capabilities for

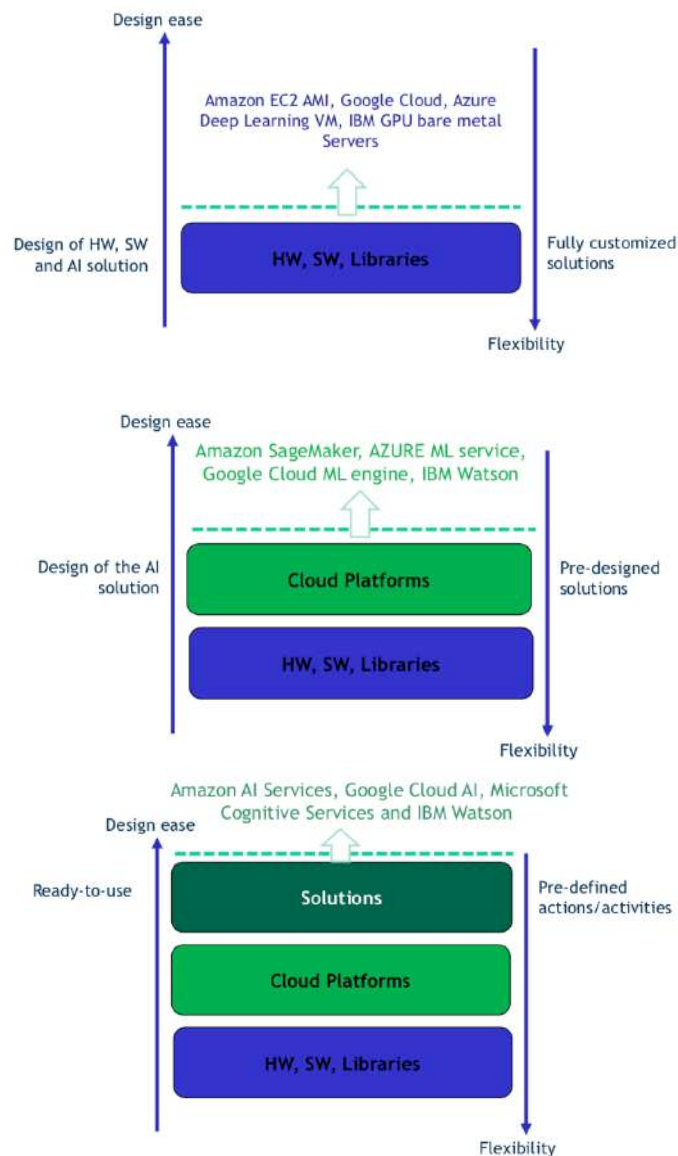
- Designing a solution (without required a deep knowledge of ML)
- Setting up a project (managing demand increases and IT solution)

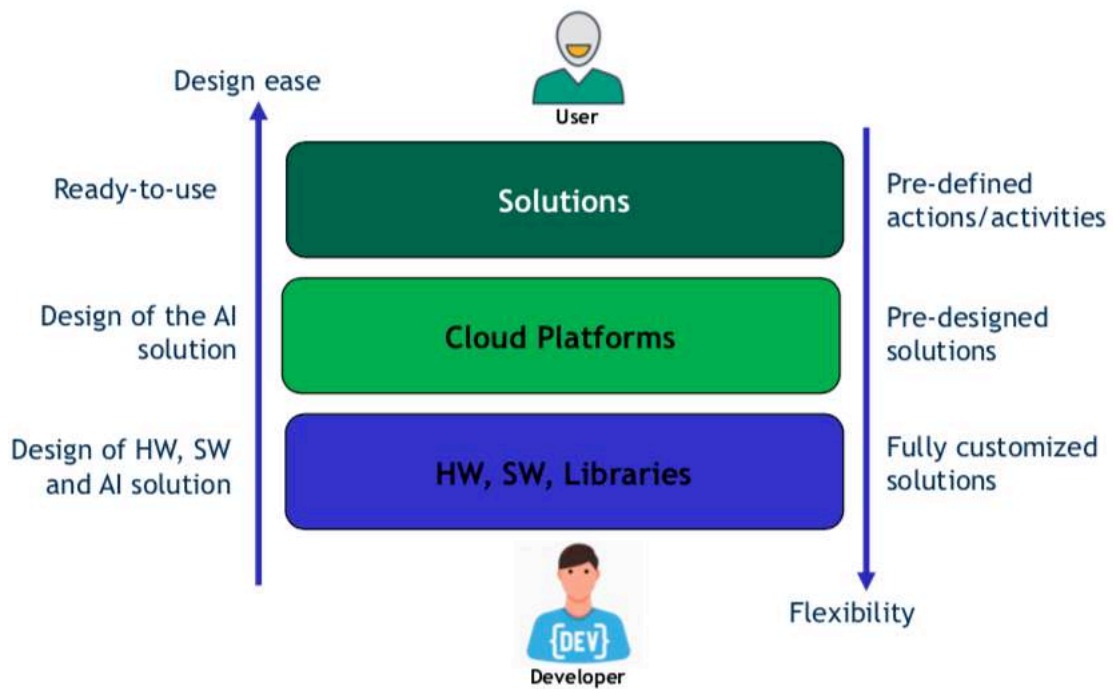
Amazon, Microsoft and Google provide

- Solutions → ML Solutions as a service
- Platforms → ML Platforms as a service
- Infrastructures → ML Infrastructures as a service

to support ML in the Cloud

Machine Learning as a Service





Disadvantages of Cloud Computing

- Requires a constant Internet Connection
- Does not work well with low-speed connections
- Features might be limited
- Can be slow
- Stored data might not be secure
- Stored data can be lost