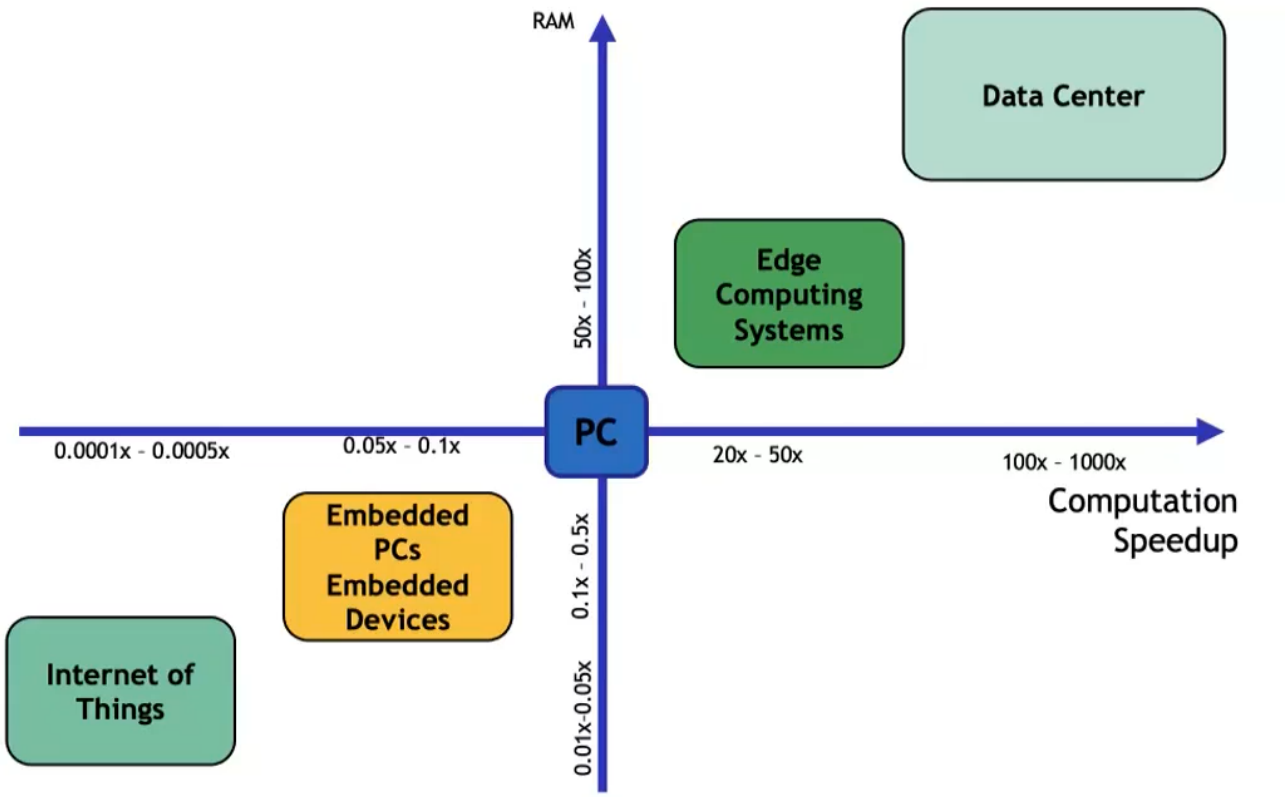
Computing Infrastructures

github.com/martinopiaggi/polimi-notes

2022-2023

Contents

1 Data centers



1.1 Traditional data centers

The data center approach gained popularity in computing. DCs shift has led to the rise of Software as a Service (SaaS) and its widespread use. DCs also allow for more efficient and cost-effective execution of computationally intensive tasks, like training neural networks.

ADVANTAGES	DISADVANTAGES
Lower IT costs	Require a constant Internet connection (so not work well with low-speed connections and latency problem)
High performance, "unlimited" storage capacity and backup	Privacy and security issues
Device independence (no need to manually configure . . .)	High Power Consumption

1.2 Warehouse-scale Computer

From a DCs are moving towards Warehouse Scale Computers (WSCs). Actually WSCs can be considered as a type of DCs, the main concept is that:

- DCs consist of a collection of different servers
- WSCs use a relatively homogeneous hardware and system software platform to simplify management and reduce cost.

The machine is itself this large cluster or aggregation of servers and needs to be considered as a single computing unit.

Here the main differences:

- Traditional data centers:
  - DCs usually house many small- or medium-sized applications.
  - Each application runs on dedicated hardware, isolated from other systems.
  - Applications within a data center typically do not communicate with each other.
  - Data centers can host hardware and software for multiple organizational units or companies.
- WSCs:
  - WSCs run few large applications or internet services
  - There is a common resource management infrastructure which provides deployment flexibility
  - The main requirements are homogeneity, single-organization control, and cost efficiency.

2 Servers

Servers are organized in racks, blades, or towers and are the computational unit in data centers or WSC.

2.1 Tower

A tower server looks and feels much like a traditional tower PC.

PROs	CONs
Scalability and ease of upgrade	Consumes a lot of space
Cost-effective (cheapest servers)	Provides a basic level of performance
Cools easily (low component density)	Complicated cable management

2.2 Rack server

PROs	CONs
Failure containment	Power usage
Simplified cable management	Maintenance
Cost-effective	

Server racks use rack units, or “U”s, as a measurement. 1U equals 44.45 mm (1.75 inches).

2.3 Blade server

Blade servers are a new and advanced type of server that can be described as a hybrid rack server. These servers are housed in blade enclosures, creating a blade system. Blade servers are the smallest type of server available, making them ideal for conserving space.

PROs	CONs
Load balancing and failover Centralized management, all blades connected to a single interface Cabling Size and form-factor	Expensive configuration high component density -> effort to avoid overheated

2.4 Technologies inside servers

	Advantages	Disadvantages
CPU	Easy to be programmed and * support any programming framework.	Most suited for simple models that do not take long to train and for small models with small training set.
GPU	Ideal for applications in which data need to be processed in parallel like the pixels of images or videos.	Programmed in languages like CUDA and OpenCL and therefore provide limited flexibility compared to CPUs.
TPU	Very fast at performing dense . vector and matrix computations and are specialized on running very « fast ML workloads	For applications and models based on TensorFlow/PyTorch/ JAX Lower flexibility compared to CPUs and GPUs
FPGA	Higher performance, lower cost . and lower power consumption compared to other options like . CPUs and GPU	Programmed using OpenCL and High-level Synthesis (HLS). Limited flexibility compared to other platforms.

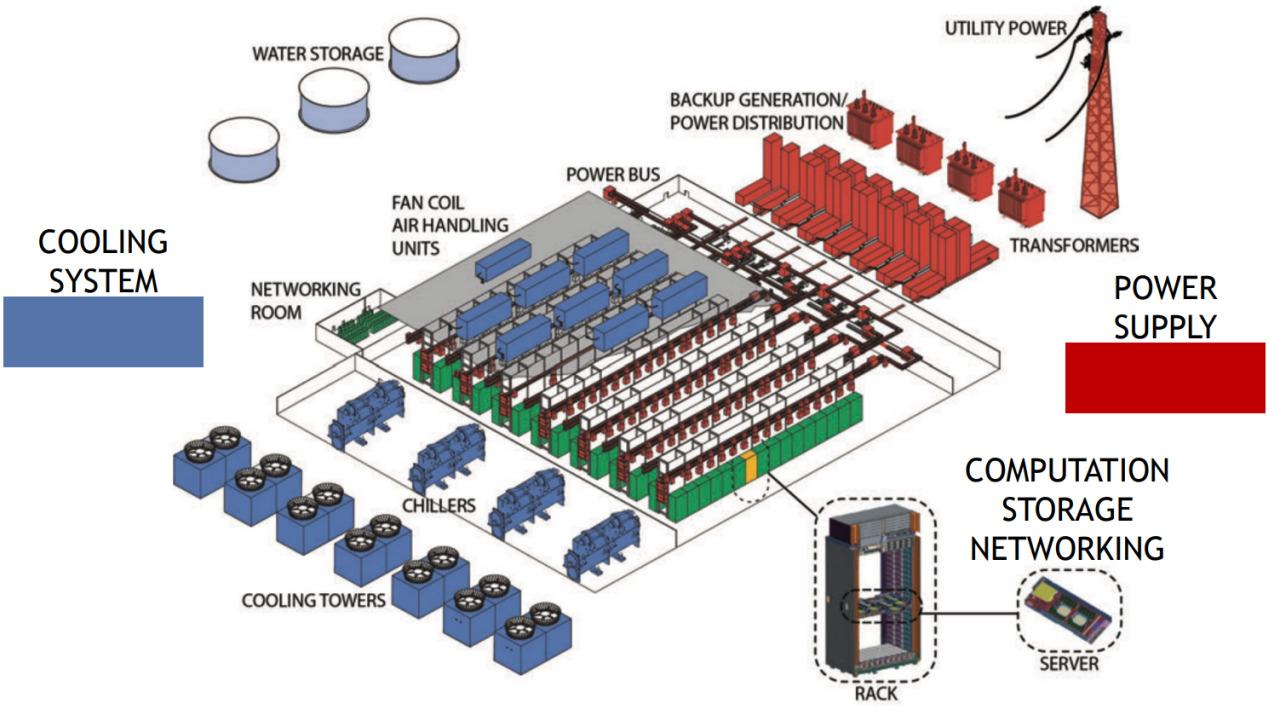
Power usage effectiveness (PUE) is the ratio of the total amount of energy used by a DC facility to the energy delivered to the computing equipment

$$PUE = \frac{\text{Total Facility Power}}{\text{IT Equipment power}}$$

PUE	DCiE	Level of Efficieny
3.0	33%	Very Inefficient
2.5	40%	Inefficient
2.0	50%	Average
1.5	67%	Efficient
1.2	83%	Very Efficient

3 Infrastructure

A high-performance data center WSC (Web-Scale Computing) must operate continuously at optimal conditions, requiring efficient and reliable power delivery and building infrastructure.



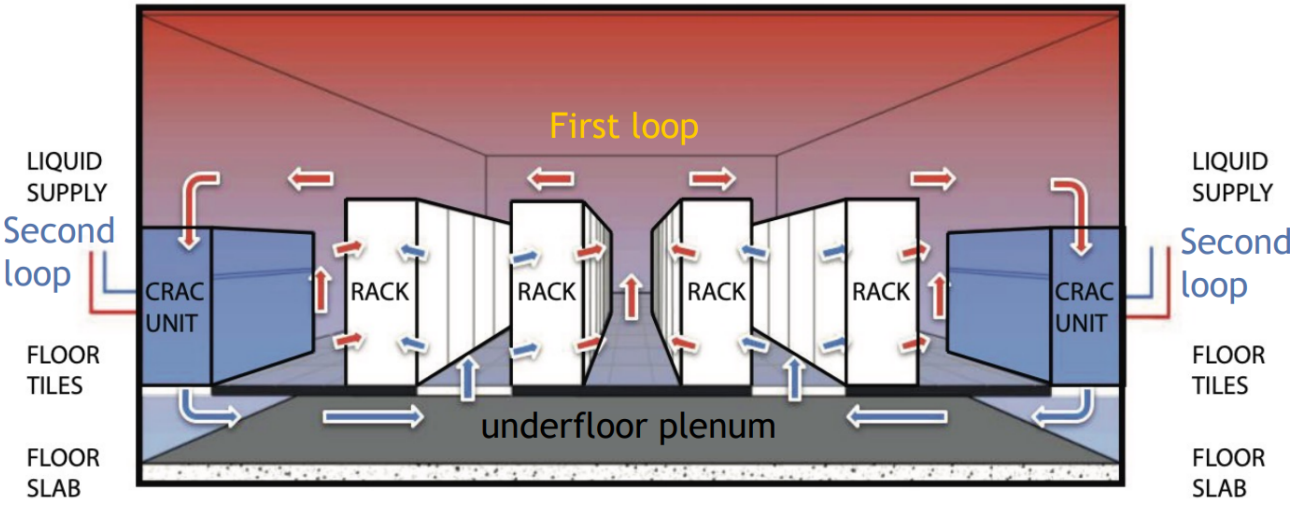
3.1 Cooling

Cooling usually requires about half the energy required by the IT equipment (servers + network + disks).

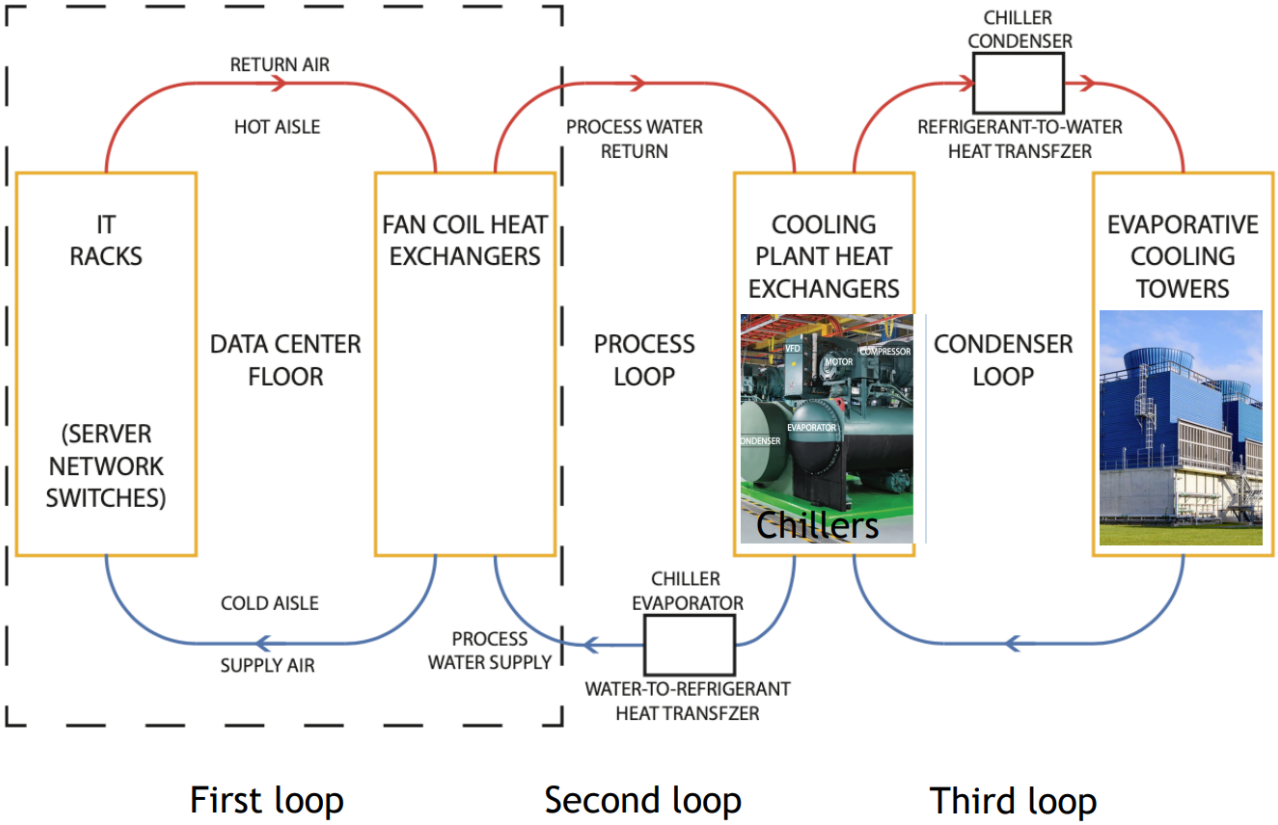
Amortized Cost	Component	Sub-Components
~ 45%	Servers	CPU, memory, disk
~ 25%	Infrastructure	UPS, cooling, power distribution
~ 15%	Power draw	Electrical utility costs
~ 15%	Network	Switches, links, transit

Cooling topologies have tradeoffs in complexity, efficiency, and cost.

- Open loop is the the simplest cooling approach: it's an open-the-windows approach. It requires filtering of airborne particles, and may introduce complex control problems.
- Closed loop cooling systems are suitable for high-density datacenters where heat dissipation is critical. However, their operational efficiency is usually not as high as other systems. Here an example of 2-loops:



- 3-loop system is an evolution of 2-closed-loops. It's expensive and has complex controls but provides good efficiency.

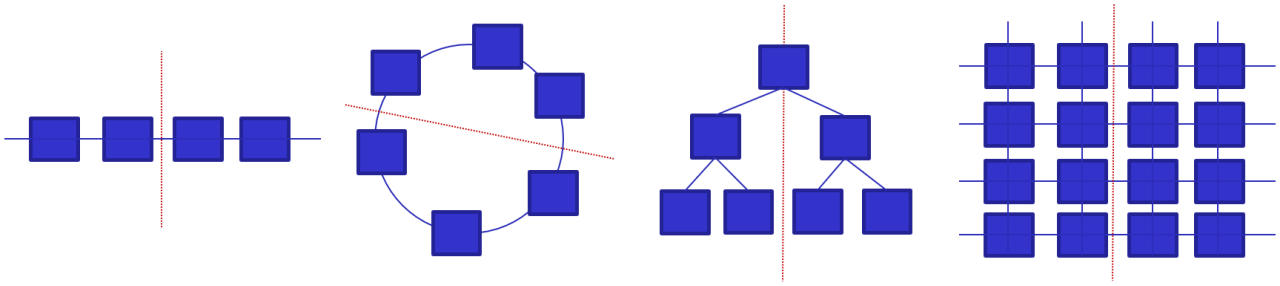


3.2 Networking

The evolution of computing infrastructures can be presented in the following steps: client-server, web applications, microservices and datacenters/warehouse scale computing. In terms of horizontal scaling, networking lacks a straightforward solution.

- Increasing the leaf bandwidth is simple: doubling the number of servers will provide twice the network ports and bandwidth.
- However, if every server needs to communicate with the others, **bisection bandwidth** must be considered.

The bisection bandwidth is the bandwidth across the narrowest line that divides the newtork into two parts equally, and it is important for characterizing the network capacity.



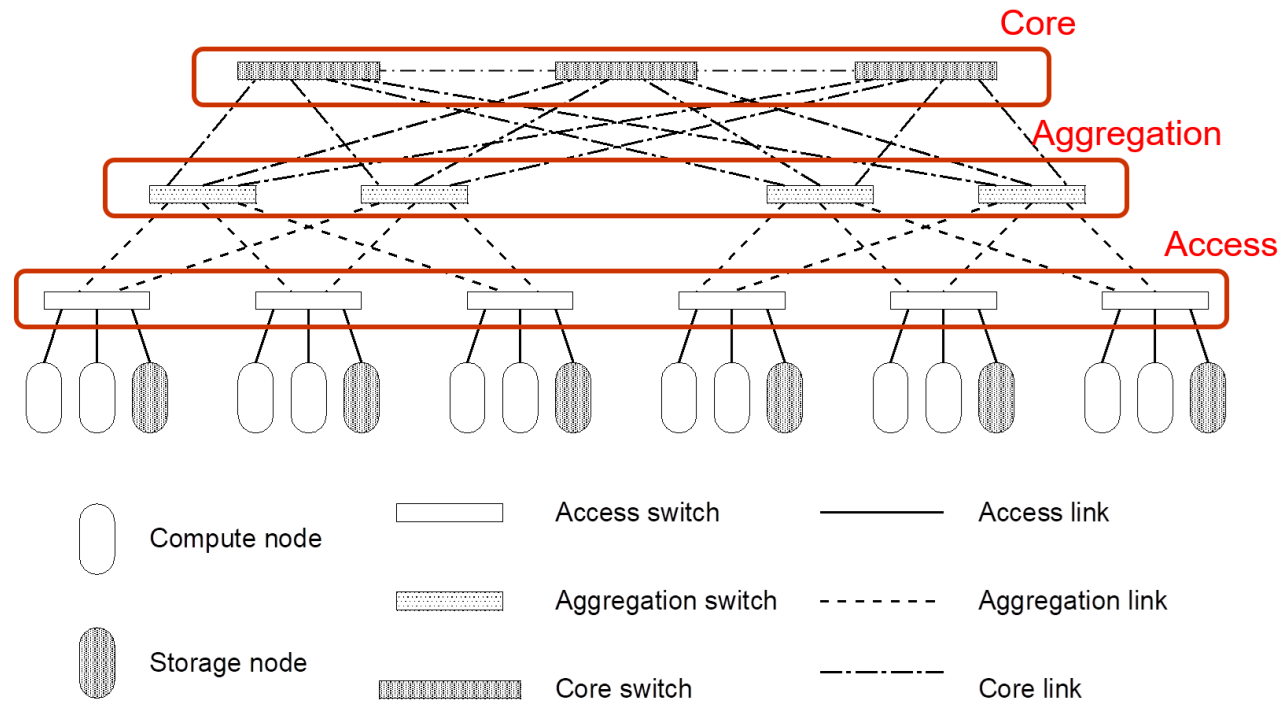
Data center's networking can be classified in the following ways:

- **Switch-centric** (also called router-centric): alle the nodes are implemented as specialized hardware which can manage traffic performing packet forwarding. Examples are:
  - Classical 3-tier architecture
  - Leaf-Spine architectures
- **Server-centric**: Uses servers with multiple Network Interface Cards (NICs) to act as switches in addition to performing other computational functions. This approach might cause slow performance.
- **Hybrid architectures**: combining switches and servers for packet forwarding is a current trend in computing infrastructures, but it can be expensive.

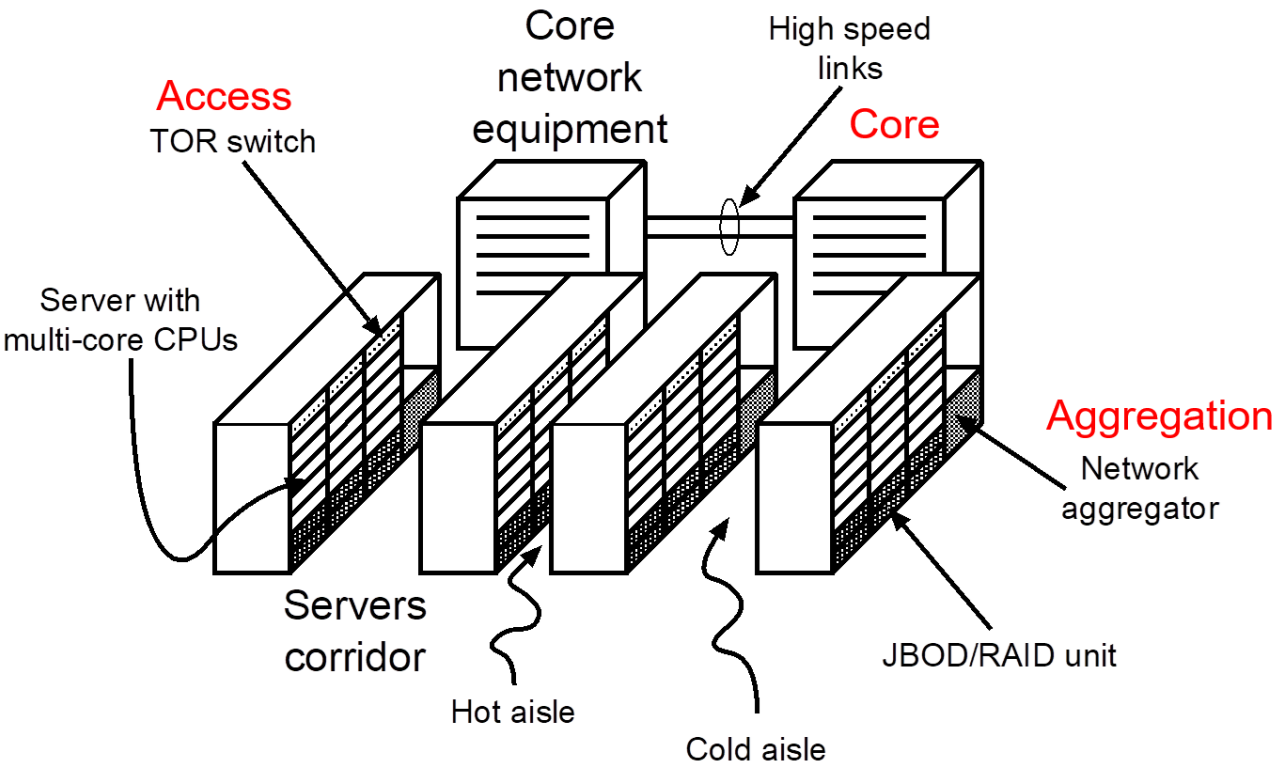
3.2.1 Switch-centric: Classical 3-tier architecture

The internal network is divided into three levels.

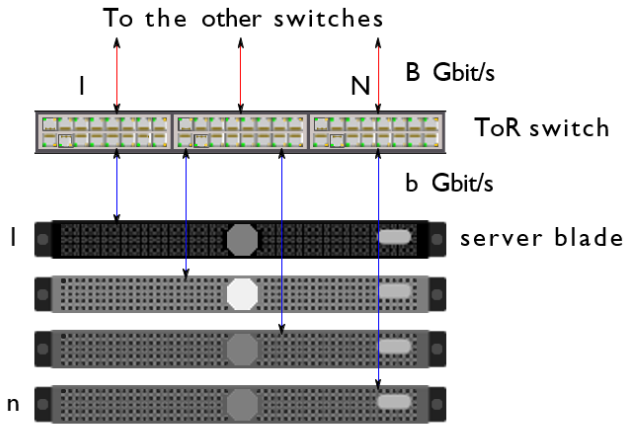
1. **Access**: This network layer connects the servers to the network.
2. **Aggregation**: This level groups the servers together. It's usually implemented through TOR (Top Of the Rack) switches.
3. **Core**: This level is the furthest from the individual servers and aggregates the entities of the "aggregation" level.



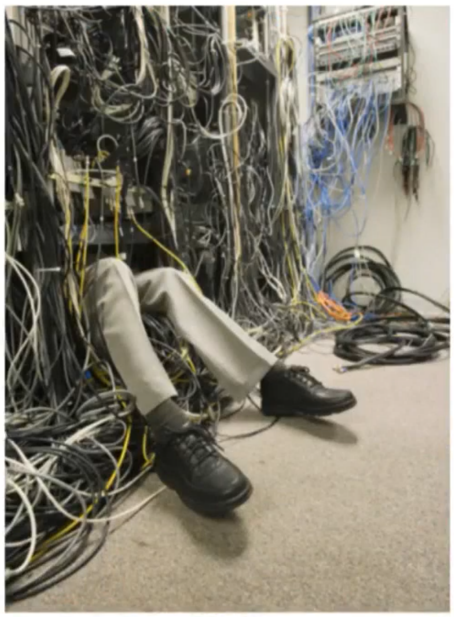
Three layer architecture reflects the topology of the data center.

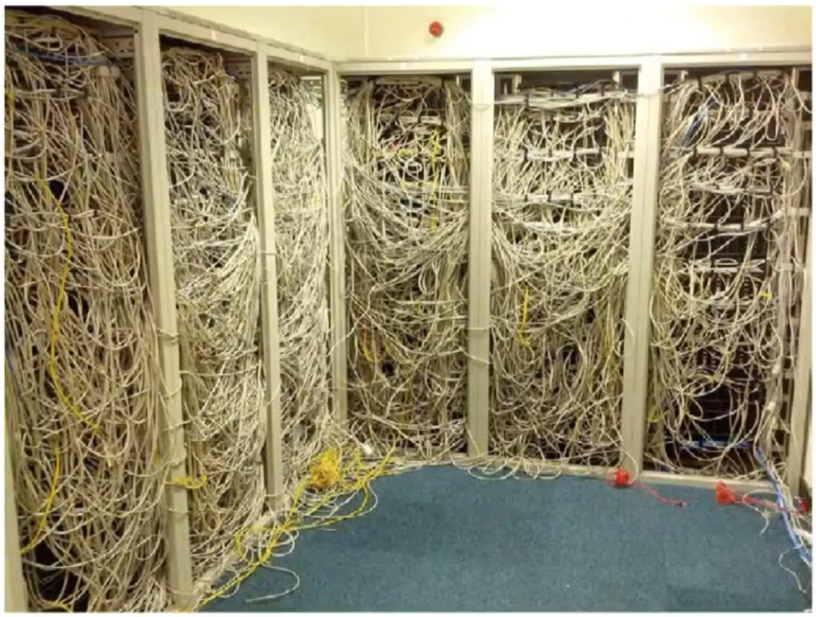


3-tier architecture is an generic approach to networking: in a rack, all servers are connected to a ToR access switch. the servers and the ToR switch are in the same rack.



Aggregation switches are either in dedicated racks or shared racks with other Top-of-Rack (ToR) switches and servers. This setup has limited scalability and can lead to higher complexity for switch management.



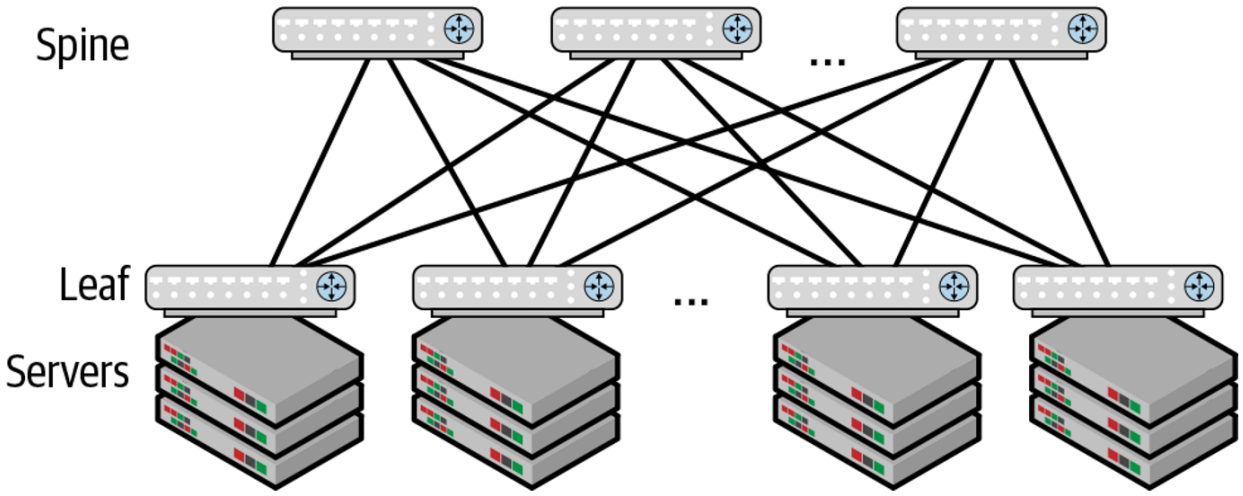


End-of-Row (EoR) is an alternative to Top-of-Rack (ToR) where an aggregation switch is placed at the end of a line of racks, one per corridor. Servers in one rack are directly connected to the aggregation switch in another rack. However, EoR requires aggregation switches with more ports and complex cabling, resulting in longer cables and higher costs.

3.2.2 Switch-centric: Leaf-Spine architectures

It's an architecture based on two-stage interconnections based on a circuit switching architecture, once used in telephony (an alternative architecture to the "classical" packet switching). There are two levels:

- **Leaf** switches act as top-of-rack (ToR) switches
- **Spine** switches are dedicated aggregation switches.



Pros:

- Leaf and spine topology is bidirectional for each switching module.
- Number of hops is the same for any pair of nodes
- Small blast radius: if you have a failure in the architecture, the blast (the number of interested machines) is limited.

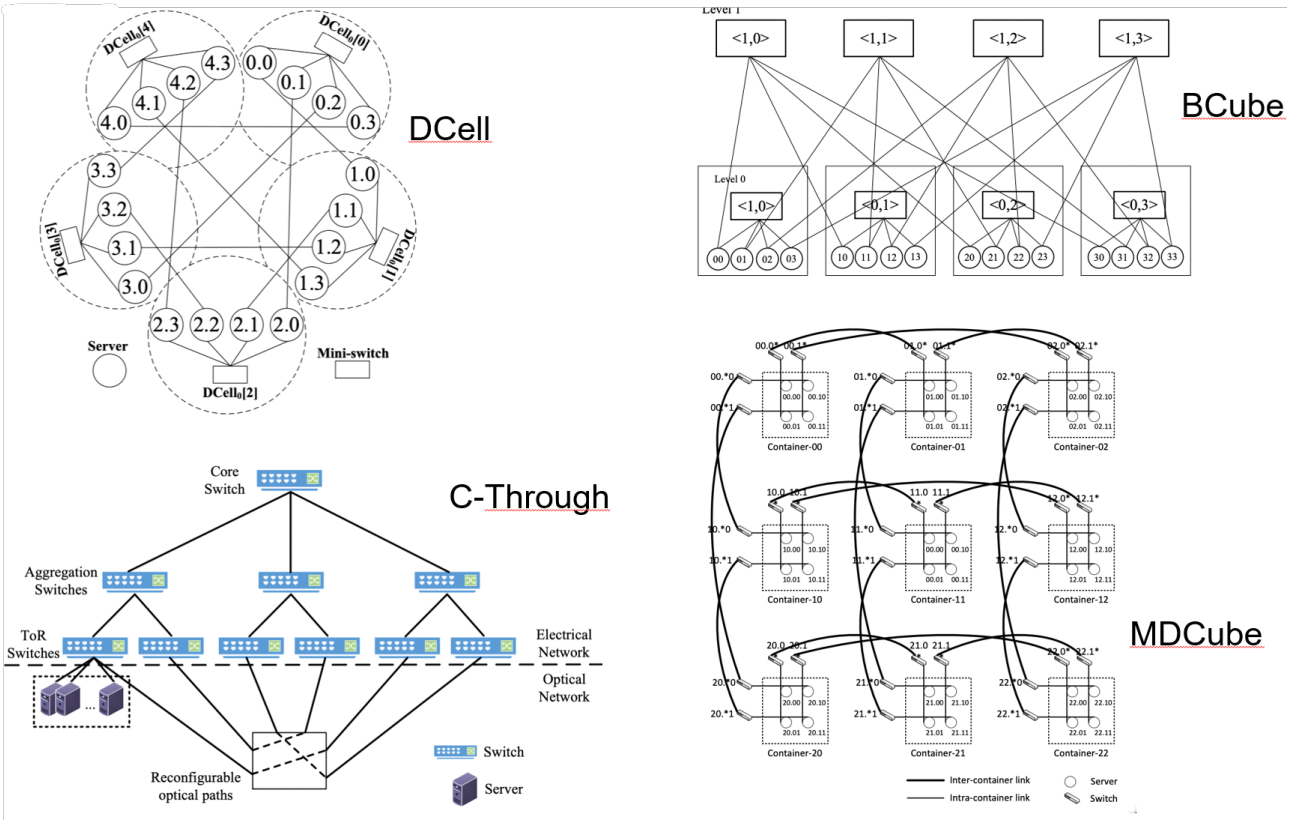
3.2.3 Server-centric

- A server-centric architecture proposed for container-sized data centers.
- Only servers used to build the data center, reducing costs.
- 3D-Torus topology used for direct server interconnection.
- Network locality exploited to increase communication efficiency.
- Drawbacks are servers requiring multiple NICs, long paths, and high routing complexity.

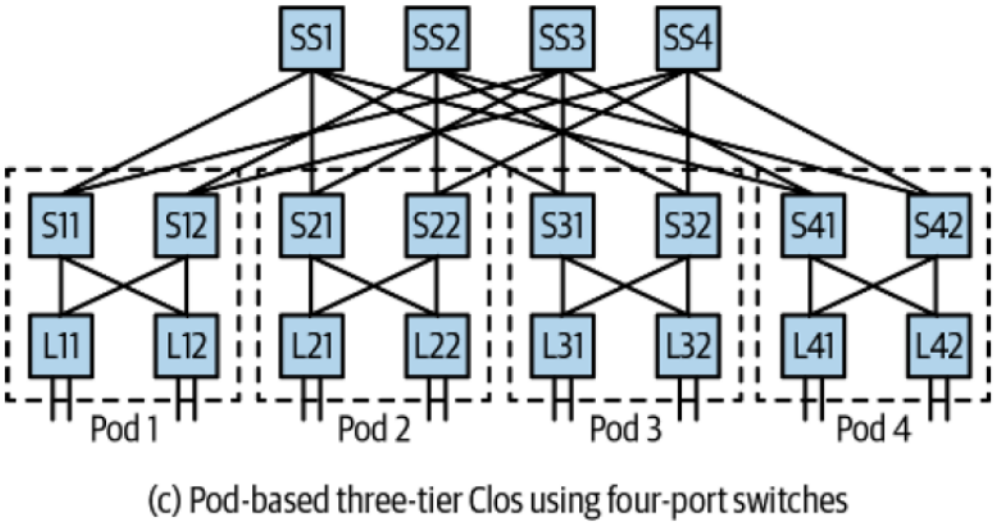
3.2.4 Hybrid architectures

Many network topologies are out there:

1. Fat tree (classic hierarchical topology, clearly visible access aggregation and core levels
2. D-Cell, recursive topology. Cells are organized in levels and the network "builds itself" through a discover mechanism
3. Hypercube topologies

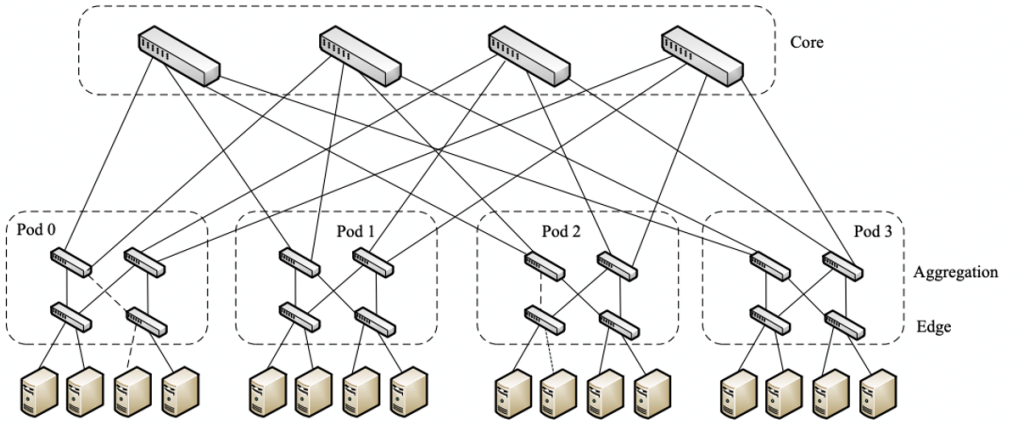


- Point Of Delivery (POD): a group of several components that work together to deliver network services. It enhances modularity, scalability, and manageability of data.
- Gigabit Ethernet switches with the same number of ports can be used.



(c) Pod-based three-tier Clos using four-port switches

This architecture is known to be used by Microsoft and Amazon for example..



4 Storage

Disks can be seen by an 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, which can range from 1 disk sector (512 B) to 128 sectors (64 KB), are the minimal unit that an OS can read from or write to a disk.

Clusters contains:

- File data
- Meta data:
  - File names
  - Directory structures and symbolic links
  - File size and file type
  - Creation, modification, last access dates
  - Security information (owners, access list, encryption)
  - Links to the LBA where the file content can be located on the disk

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.

$$\text{actual size on disk} = ceil(\frac{\text{the file size}}{\text{the cluster size}}) * \text{the cluster size}$$

The waste of space is called internal fragmentation of files.

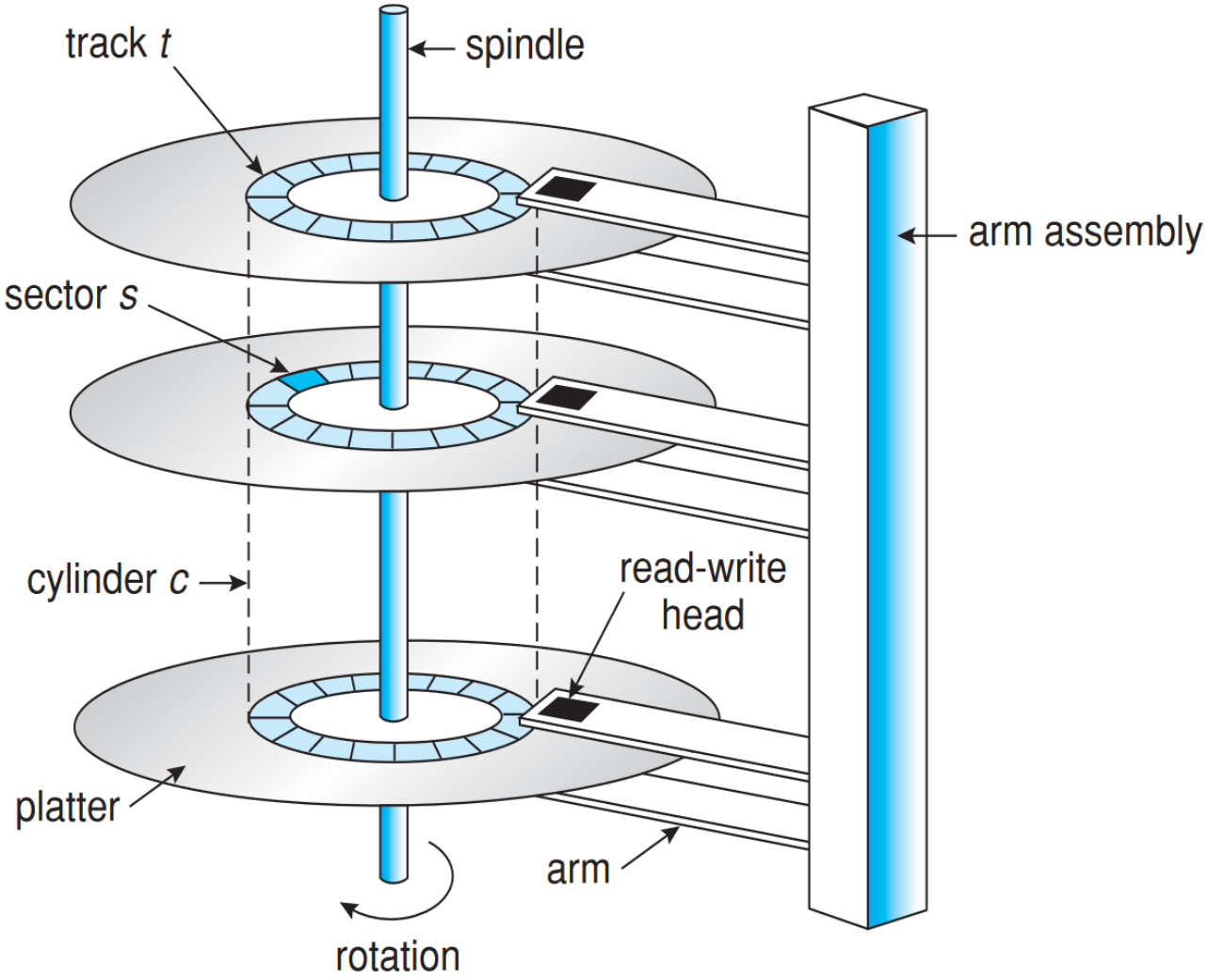
$$\text{waste disk space} = \text{actual size on disk} - \text{file size}$$

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.

As the life of the disk evolves, there might not be enough space to store a file contiguously. In this case, the file is split 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.

4.1 HDD

An HDD consists of one or more rigid ("hard") rotating disks (platters) with magnetic heads arranged on a moving actuator arm to read and write data to the surfaces.



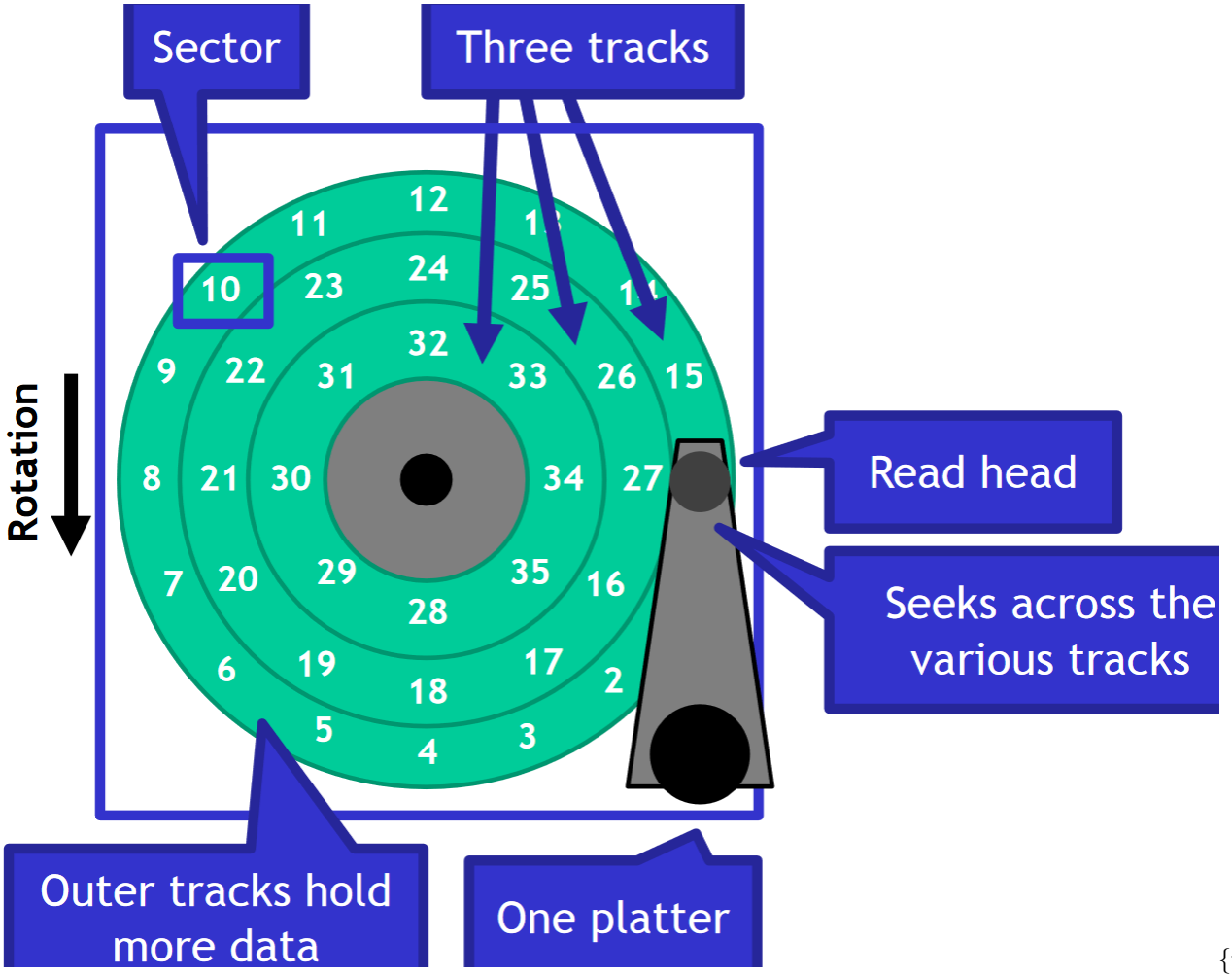
HDD geometry:

- Sectors arranged into tracks
- A cylinder is a particular track on multiple platters
- Tracks arranged in concentric circles on platters

- A disk may have multiple, double-sided platters

To understand well this:

- Externally, hard drives expose a large number of sectors (blocks)
- Typically 512 or 4096 bytes
- Individual sector writes are atomic
- Have a header and an error correction code
- Multiple sectors writes may be interrupted (torn write)



{width = 50%}

The external track are "faster" to read since angular velocity considerations and are more dense.

Four types of delay:

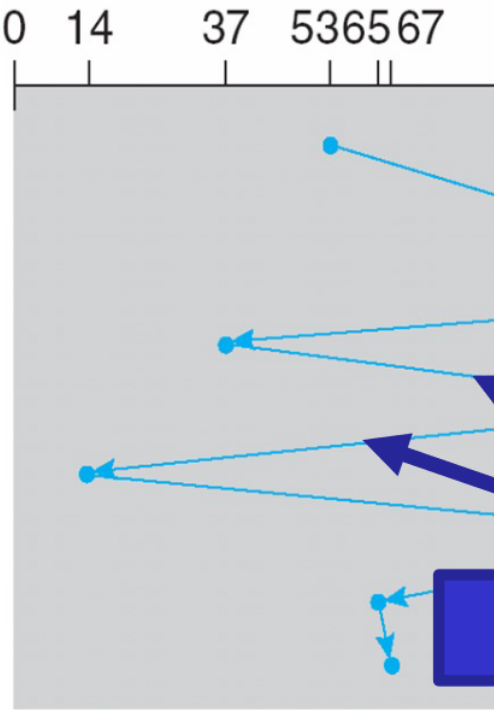
1. Seek delay: The seek time represents the time necessary to align the head over the track $T_{\text{seek AVG}} = \frac{T_{\text{seek MAX}}}{3}$
2. Rotational Delay: Time to rotate the desired sector to the read head related to RPM 2 $T_{rotation} = \frac{R_{period}}{2}$

with $R_{period} = \frac{1}{RPM}$. there is the time where the sector is waited to be under the head (and it's based on rotational speeds ... just an estimation)
3. Transfer time: then there is the time given by the controller to transfer data
4. Controller Overhead: Overhead for the request management

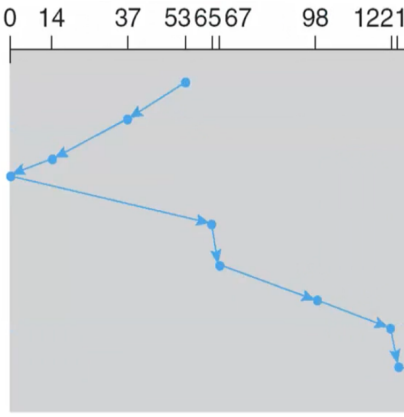
So at the end:$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer} + T_{overhead}$. Or, considering data locality: $T_{I/O} = (1 - DL)(T_{seek} + T_{rotation}) + T_{transfer} + T_{overhead}$.

4.1.1 Disk Scheduling

Since there is a queue of requests to the disk, they can be reordered to improve performance in HDD. Several scheduling algorithms:

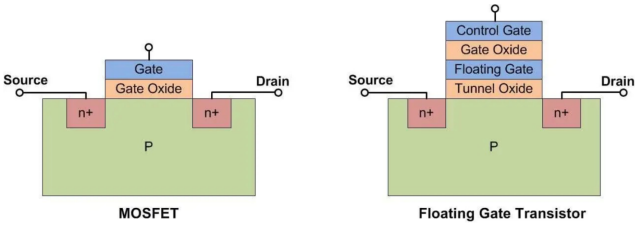


- **First come, first serve** (FCFS): Most basic scheduler, serve requests in order
= 50%}
- **Shortest seek time first** (SSTF): Minimize seek time by always selecting the block with the shortest seek time

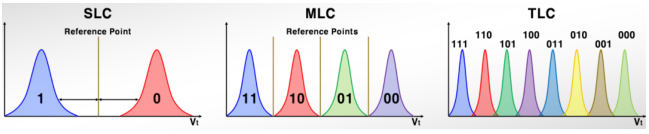


- **SCAN** (the elevator algorithm): Head sweeps across the disk servicing requests in order.
- **C-SCAN** (circular scan), C-LOOK: variants of Scan

5 SSD



- Single-level cell (SLC): simplest solution. For each cell just a voltage. Single bit per cell.
- Multi-level cell (MLC): Two bits per cell
- Triple-level cell (TLC): three bits per cel
- QLC, PLC...



NAND flash is organized into Pages and Blocks:

- A page contains multiple logical block (e.g. 512B-4KB) addresses (LBAs)
  - A block typically consists of multiple pages (e.g. 64) with a total capacity of around 128-256KB. Pages can be in three states:
    * Empty or erased
    * dirty or invalid
    * in use or valid

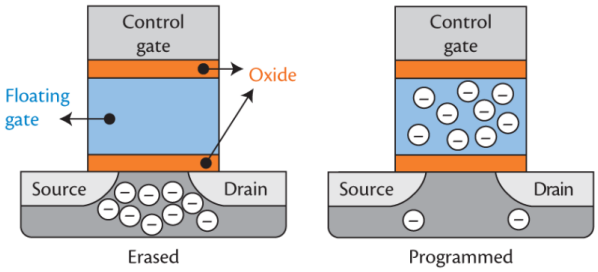
Actually there are some constraints:

- Pages: smallest unit that can be read/written. Can't be erased.
- Blocks (or Erase Block): smallest unit that can be erased.

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

SSDs have a limited lifespan due to NAND flash memory limitations. A whole block cannot be erased until all pages within the block are not used. The SSD must move data from other pages to another block to ensure all

pages in a block are no longer in use before erasing it. *Why erase an entire block and not just erase a single page?* Erasing a single page would require the memory controller to keep track of which pages have been erased and which ones haven't (slow) and erasing only blocks helps to **extend the overall lifespan of the device**. This because when flash cells undergo numerous write-erase cycles, their electrical properties deteriorate, causing them to become unreliable. The oxide layer in floating-gate transistors of NAND flash memory breaks down during the erasing process since this one involves a sizable electrical charge. So basically **when a block is erased, it degrades the silicon material** due to a large electrical charge.



5.0.1 FTL

Flash Translation Layer (FTL) is an SSD component that make the SSD "look as HDD"

- **Data Allocation and Address translation**: efficient methods to reduce Write Amplification effects and program pages within an erased block.
- **Garbage collection**: reuse of pages with old data
- **Wear leveling**: distributing writes and erases evenly across all blocks of the device, ensuring that they wear out roughly at the same time. Erasing a block only happens when all the pages within it are unused, and any valid data from other pages is moved to another block before erasing the block entirely.

5.0.2 Block Mapping problem

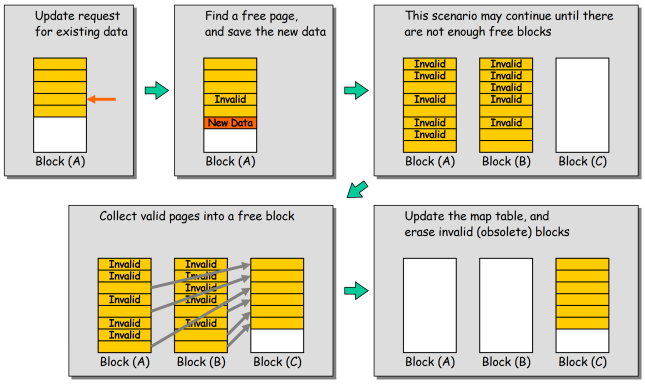
Data Allocation and Address translation: efficient methods to reduce Write Amplification effects and program pages within an erased block.

Also Mapping Table Size is a problem ... With a 1TB SSD with a 4byte entry per 4KB page, 1GB of DRAM is needed for mapping. Some approaches to reduce the costs of mapping:

- Block-based mapping: Mapping at block granularity, to reduce the size of a mapping table. But this can cause an increment of operations necessary.
- Hybrid mapping: When looking for a particular logical block, the ftl will consult the page mapping table and block mapping table in order: log blocks (page mapped) and then data blocks (block-mapped). Also FTL can perform merge operations to reduce the numbers of occupied blocks.
- Page mapping plus caching: cache the active part of the page-mapped FTL. If a given workload only accesses a small set of pages, the translations of those pages will be stored in the FTL memory

5.1 Garbage Collection

Garbage collection: reuse of pages with old data

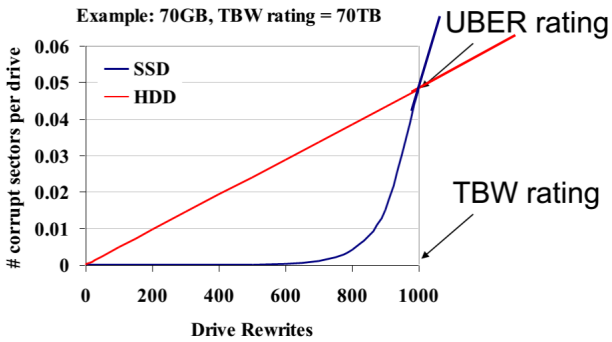


Garbage collection is costly. Also many file systems do not truly erase data, for instance, on Linux, the "delete" function eliminates the file meta-data but not the actual file, and as a result, the garbage collector wastes resources duplicating useless pages. However, having an OS that supports TRIM is a possible solution.

5.2 Wear leveling

Wear leveling: distributing writes and erases evenly across all blocks of the device, ensuring that they wear out roughly at the same time. Erasing a block only happens when all the pages within it are unused, and any valid data from other pages is moved to another block before erasing the block entirely.

6 Summary SSD vs HDD



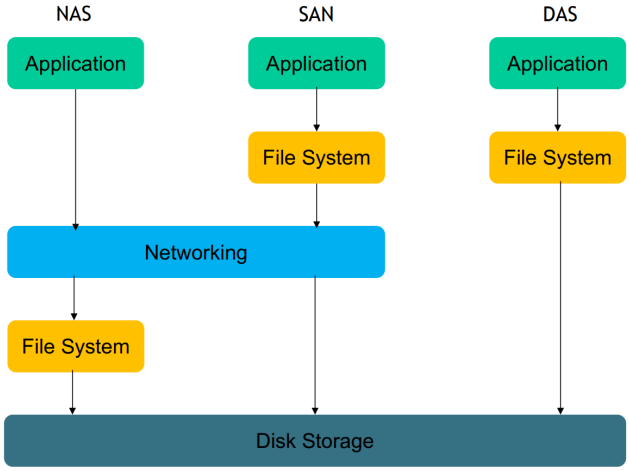
- UBER is a measure of the number of data errors per bits read
- TBW is the total amount of data that a SSD can sustain before failing

6.1 Storage Systems

Number	Domain and Advantages	Disadvantages
DAS (Direct Attached Storage)	Suitability for budget and constraint-heavy environments. Simple storage solutions	Limited accessibility and scalability. No central management or backup
NAS (Network Attached Storage)	File storage and sharing and big data storage. Scalability and greater accessibility	Increased LAN traffic, performance limitations and security and reliability concerns

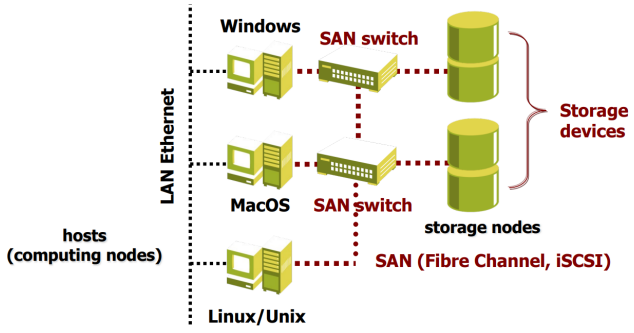
Number	Domain and Advantages	Disadvantages
SAN (Storage Area Network)	Suitable for DBMS and virtualized environments, have best performance, best scalability and best availability	High costs

- A Direct Attached Storage (DAS): A DAS is a storage system directly connected to a server or workstation and appears as disks/volumes to the client operating system.
- A Network Attached Storage (NAS): a computer connected to a network that offers file-based data storage services such as FTP, Network File System, and SAMBA to other devices on the network. It appears as a File Server to the client OS.
- Storage Area Networks (SAN): Remote storage units are connected to PCs via networking technologies such as Fiber Channel and appear as disks/volumes on the client's OS.



6.2 NAS vs SAN

San (Storage Area Network) has a dedicated network for accessing storage devices.



- NAS provides storage and file system, while SAN only provides block-based storage.
- NAS is seen as a file server by the client OS, while SAN appears as a disk.
- NAS is used for low-volume access by many users, while SAN is used for petabytes of storage and simultaneous file access, like audio/video streaming.

Another factor contributing to write amplification is wear-leveling algorithms employed by SSD controllers. These algorithms distribute writes evenly across all available flash memory cells to prevent certain cells from

wearing out faster than others.

d # Dependability

Dependability refers to the ability of a system to perform its intended functionality while also exposing several key properties, including:

- **Reliability**: Continuity of correct service
- **Availability**: Readiness for correct service
- Maintainability: Ability for easy maintenance
- Safety: Absence of catastrophic consequences
- Security: Confidentiality and integrity of data

6.2.0.1 Failure Rate

Type of malfunctioning:

- Fault: physical defect or software bugs.
- Error: program incorrectness that can result from a fault.
- Failure: catastrophic scenario

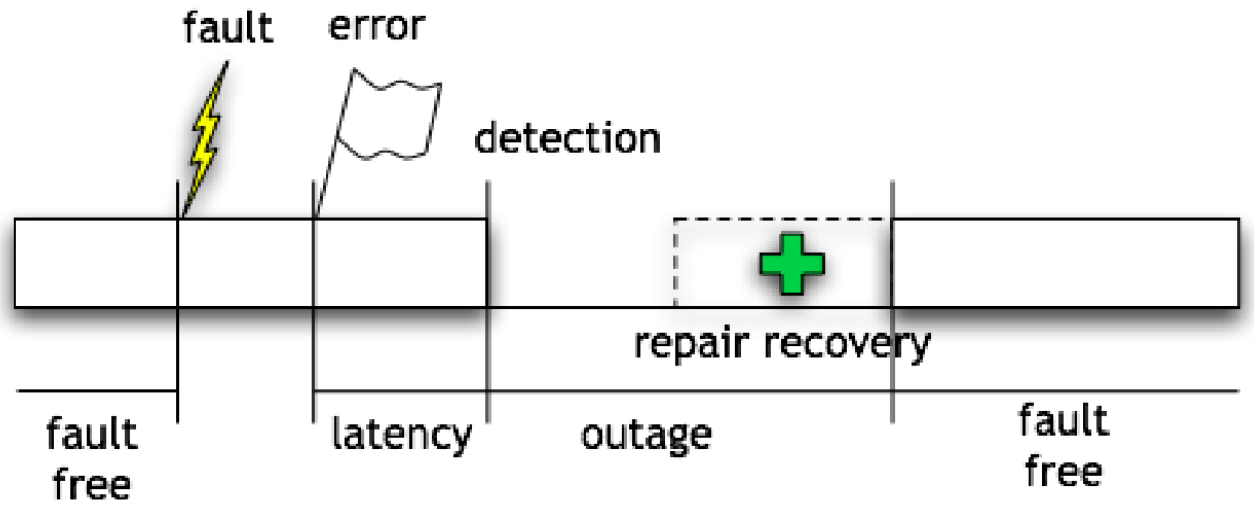
A fault can cause

6.3 Reliability

$R(t)$ = probability that the system will operate correctly **until** time t.

$$R(t) = e^{-\lambda t} = e^{-\frac{1}{MTTF}t}$$

In other terms: $R(t) = P(\text{not failed during } [0, t])$ assuming it was operating at time $t = 0$.



We can also define the unReliability:

$$S(t) = 1 - R(t)$$

$S(t)$, gives the probability that the system will fail before time t.

6.3.1 Reliability Block Diagrams

An inductive model where a system is divided into blocks that represent distinct elements such as components or subsystems (each with its own reliability).

Serie:

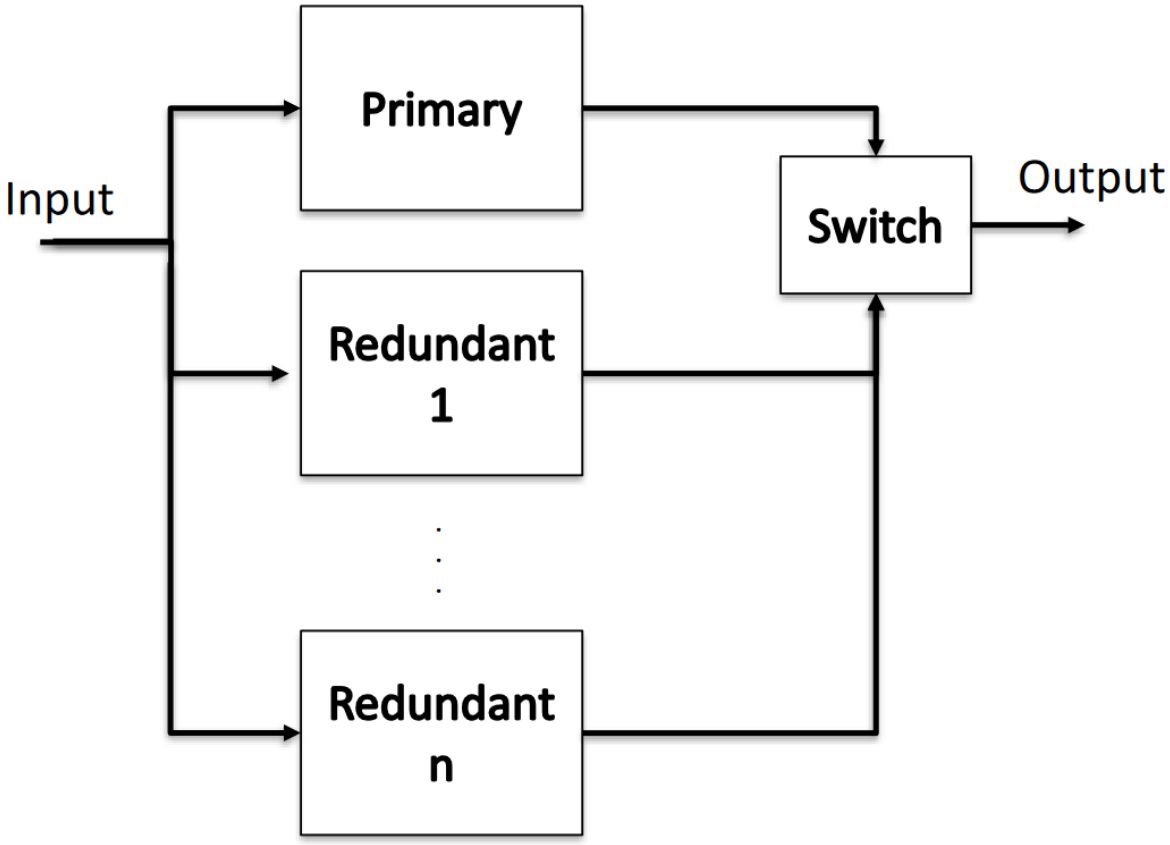
$$R_s = R_a * R_b * ...R_n$$

Parallel:

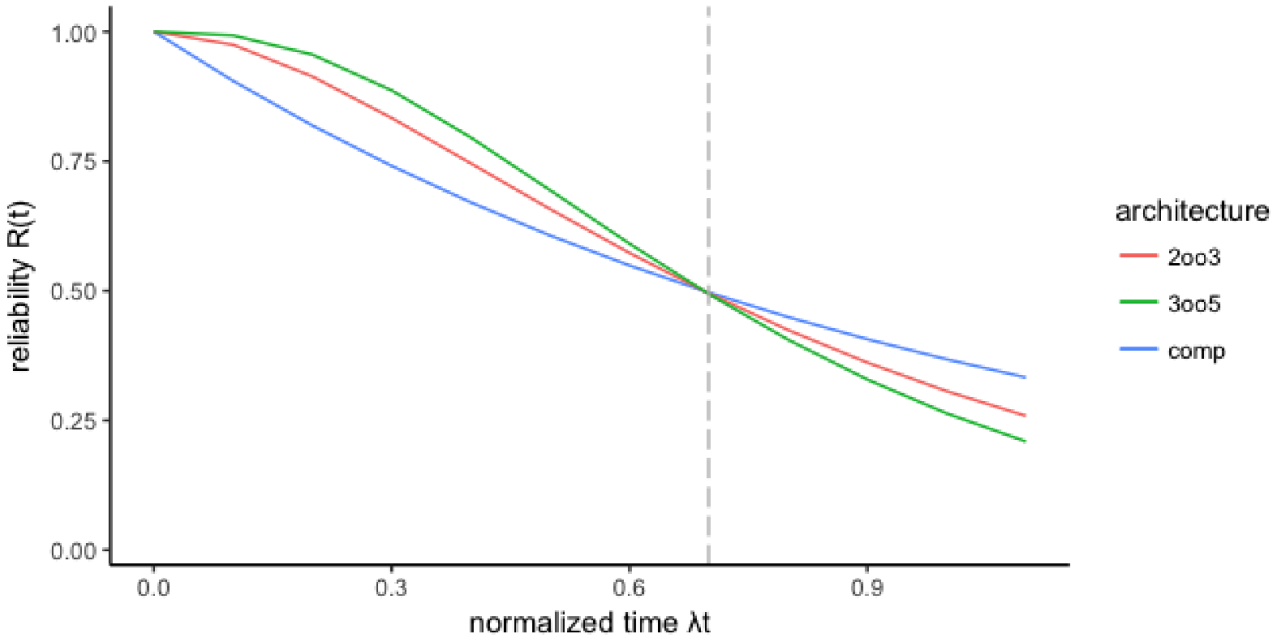
$$R_p = 1 - (1 - R_a) * (1 - R_b) * ...(1 - R_n)$$

6.3.2 RooN

A system having one primary replica and n redundant replicas (with identical replicas and perfect switching).



6.3.2.1 Triple Modular Redundancy - TMR Triple Modular Redundancy (TMR) is a system that relies on three components and a voter to ensure proper functioning. The system is considered to be working properly if at least **two** out of the **three** components are functioning correctly, in addition to the voter. TMR is particularly useful for missions with shorter durations, as it provides a higher level of reliability than the single component's one.



The important concept here is that the more components you add the higher will be the probability of failures at the limit of infinite of time. It depends on the time horizon. There is always a point in time where the reliability of basic components are more realiable than any kind of redundancy. And this point in time is:

$$t = \frac{ln2}{\lambda_m}$$

$R_{TMR}(t) > RC(t)$ when the mission time is shorter than 70% of $MTTF$. So it's always important to know the goal and the length of the mission time to make correctly rendundant systems.

$$R_S(t) = R_V \sum_{i=r}^{n} R_c^i \left(1 - R_C\right)^{n-i} \frac{n!}{i!(n-i)!}$$

Where:

- R_s System reliability
- R_c Component reliability }
- R_V Voter Reliability
- n Number of components
- i Minimum number of components which must survive

6.4 Availability

$A(t)$: probability that the system will be operational at time t

Fundamentally different from reliability since it admits the possibility of brief outages.

Availability	System	
99%	~ 4 days	Generic web site
99.9%	~ 9 hours	Amazon.com

Availability	System	
99.99%	~ 1 hour	Enterprise server
99.999%	~ 5 minutes	Telephone system
99.9999%	~ 30 seconds	Phone switches

Same definitions seen in Software Engineering 2 :

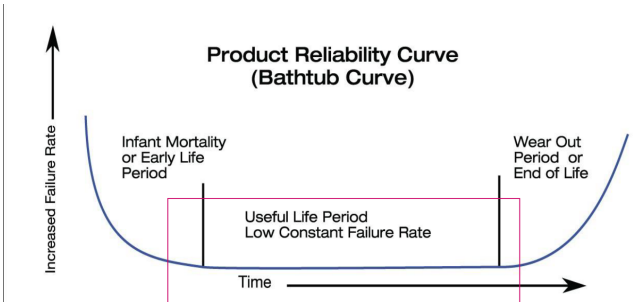
- Mean Time To Failures ($MTTF$): time between the recovery from one incident and the occurrence of the next incident, also known as up-time.
- Mean Time Between Failures ($MTBF$): Mean time between the occurrences of two consecutive incidents

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

$$MTBF = \frac{\text{total operating time}}{\text{number of failures}}$$

sometimes we also use failure rate $\lambda = \frac{\text{number of failures}}{\text{total operating time}}$.

$$A = \frac{MTTF}{(MTTF + MTTR)} = \frac{\text{uptime}}{\text{uptime} + \text{downtime}}$$



6.4.1 Data Center Availability

Data-center availability is defined by in four different tier level. Each one has its own requirements:

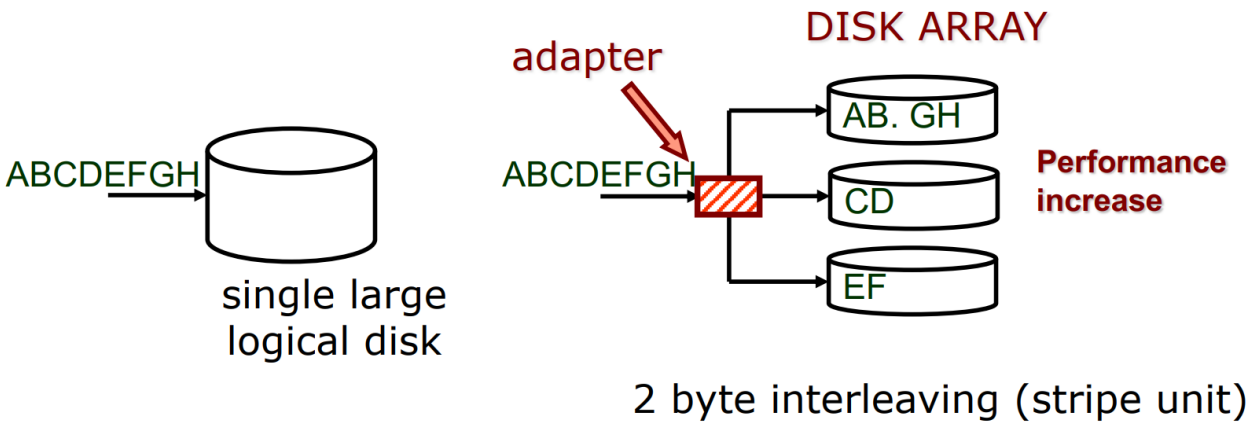
- **Tier 1**: Basic infrastructure with 99.671% availability.
- **Tier 2**: Redundant infrastructure with 99.741% availability.
- **Tier 3**: Multiple independent paths and concurrent maintenance with 99.982% availability.
- **Tier 4**: Fault-tolerant infrastructure with 99.995% availability. with independently dual-powered cooling equipment and backup electrical power storage and distribution facilities

7 Raid prologue

RAID increases performance, size, and reliability of storage systems by utilizing several independent disks as a single, large, high-performance logical disk. Data is striped across multiple disks, allowing for high data transfer rates as well as load balancing across the disks. 2023-07-04 Two orthogonal techniques:

- data striping: to improve performance
- redundancy: to improve reliability

7.1 Data striping

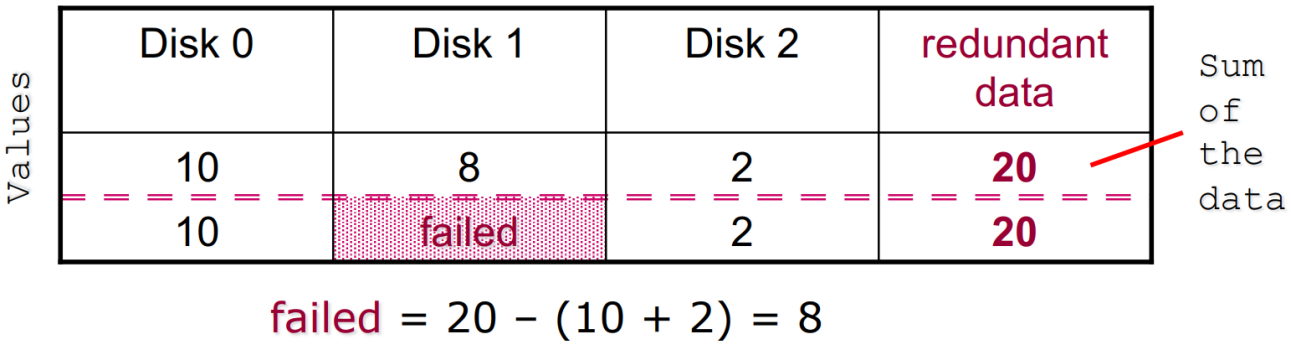


Data is written on multiple disks according to a **cyclic algorithm** (round robin).

8 RAID

Redundancy is used to tolerate disk failures with **error correcting codes**. The codes are calculated and stored separately from the data and this permits the **magic** of RAID 3, 4,5,6 etc. Write operations also update the redundant information, so **performance is worst** than traditional writes.

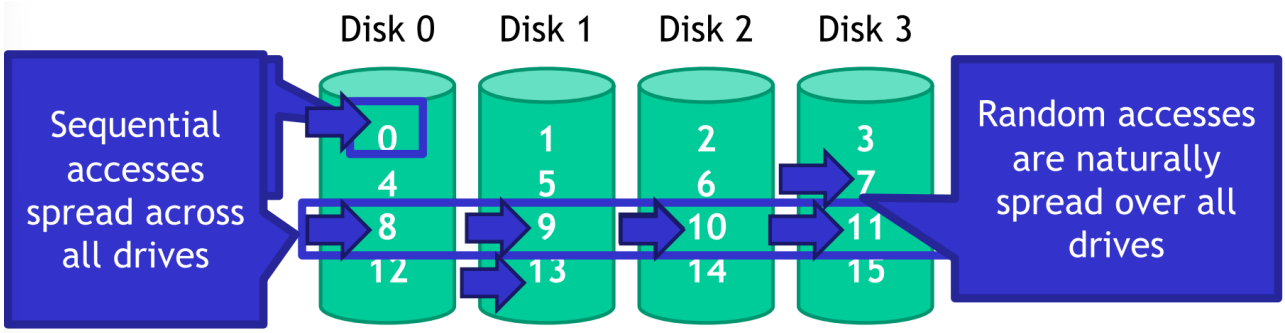
Super simple example of the idea that is the foundation of RAID:



8.1 RAID levels

8.1.1 RAID 0

Faster than using a single disk, but reliability is reduced because two disks are more likely to fail than one. Best write performance config since there is no need to update redundant data but still it's a parallelized approach.



Formulas:

$$Disk = logical_{block\_number} mod(number_{disks})$$

$$Offset = \frac{logical_{block\_number}}{(number_{disks})}$$

Capacity:

- All space on drives can be filled with data

Reliability:

- If a drive fails, data is permanently lost
- $MTTDL = MTTF$

Sequential read and write (where S is sequential transfer rate):

- Full parallelization across drives $N * S$

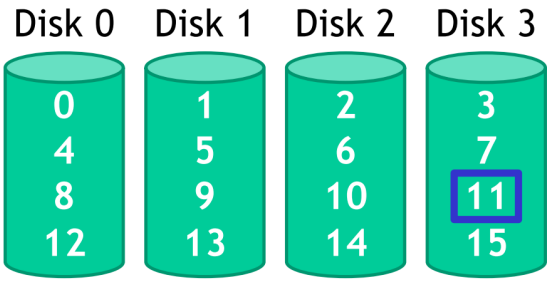
Random read and write (where R is sequential transfer rate):

- Full parallelization across all drives $N * R$

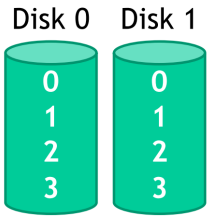
read block 11

11 % 4 = Disk 3

11 / 4 = Physical Block 2 (starting from 0)



8.1.2 RAID 1



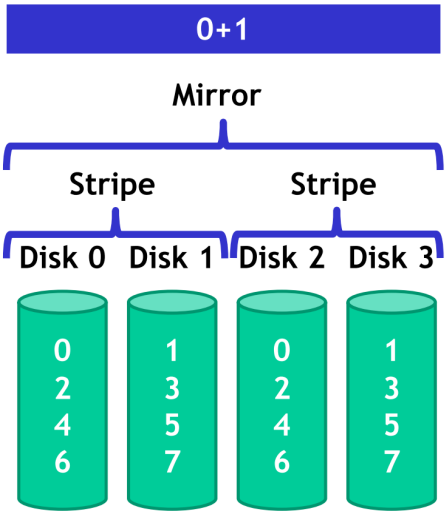
8.1.3 RAID x + y

We can use the notation RAID $x + y$ to say:

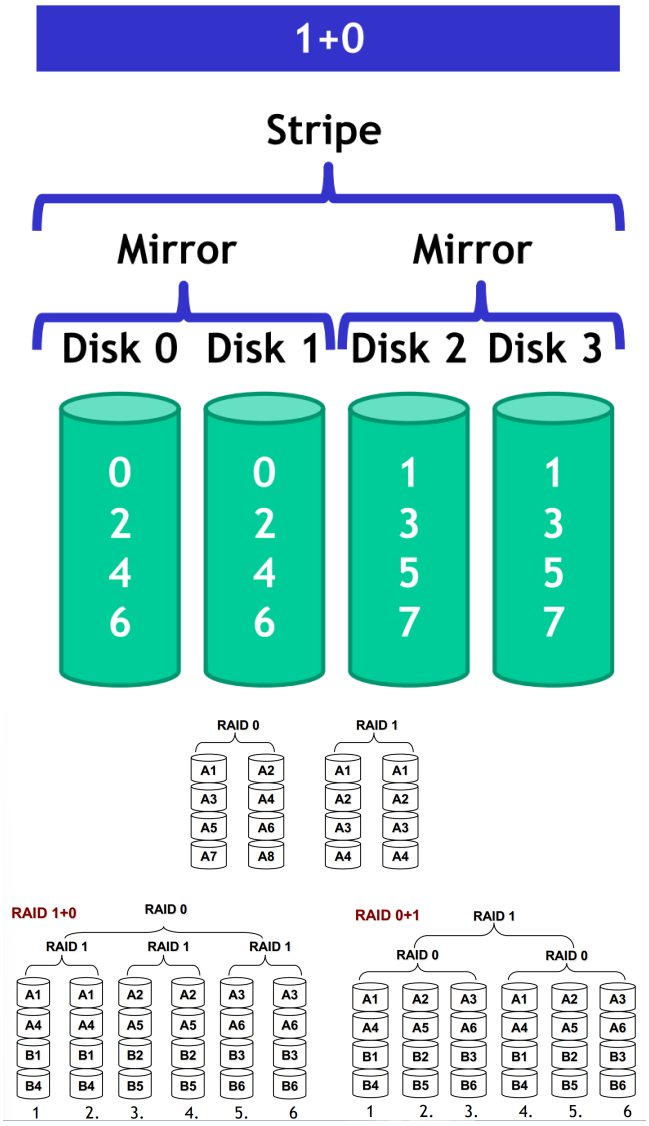
- $n * m$ is the total number of disks
- m groups of n disks and applying RAID x to each group before applying RAID y to treat the m groups as single disks.

Basically the number y in the notation RAID $x + y$ indicates the **upper** organization.

Example of RAID 0 (striped disks) + RAID 1 (mirrored) :



Example of RAID 1 (mirrored disks) + RAID 0 (striped) :



RAID $1+0$ and RAID $0+1$ have the same blocks but allocated differently. The result is that both have the same performance and storage but **RAID $1+0$ has a higher fault tolerance compared to RAID $0+1$ on most RAID controllers**.

> "The problem is that in RAID 0 if one disk fails the overall RAID 0 system fails. So in case of the two different configurations, the one with less disks in RAID 0 has an higher reliability."

For this motivation RAID $0+1$ is never used.

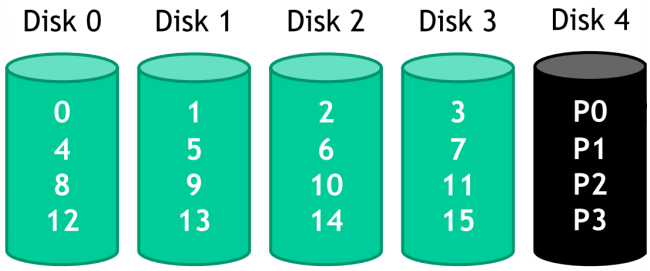
Formulas:

- Capacity is $\frac{N}{2}$
- If you are lucky, $\frac{N}{2}$ drives can fail without data loss
- the sequential write is $(\frac{N}{2}) * S$
- the random **reading** doesn't suffer of block skipping so $N * R$
- since we have to copy two time the same information we have that the random **writing** is $(\frac{N}{2}) * R$

8.1.3.1 The problem with RAID x+y

The issue with these systems is that information have to be or committed or not committed at all. **Partially commited** data is very dangerous since it gives inconsistencies to the system. This is a problem when there are power failures.

8.1.4 RAID 4

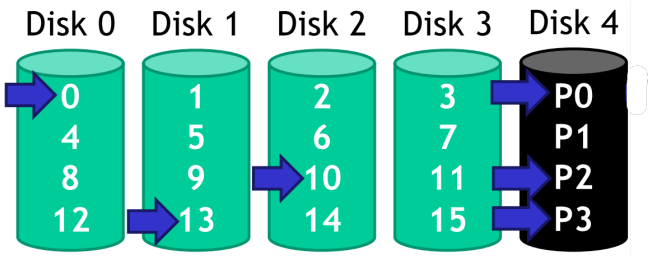


In this case the Disk 4 onlyu stores parity information for the other disks. The parity information can be computed in mainly two different ways:

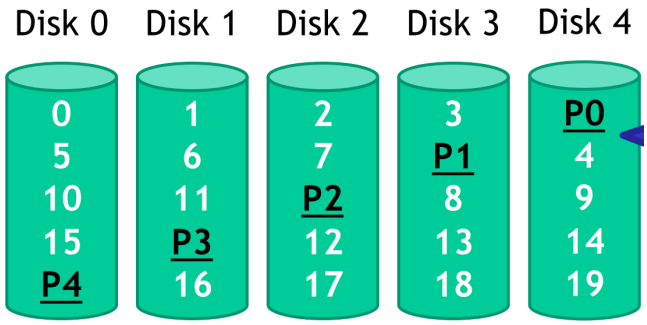
- **Additive parity**: when you modify $x-th$ disk, you have to read also the information of the other disks to compute the parity block.
- **Subtractive parity**: when you modify $x-th$ disk you are not required to read the other blocks to compute the parity block. To compute the new parity bit I use a different formula and not the "standard" one (example: $0 \oplus 0 \oplus 0 \oplus 1 = 1$ where $\oplus$ is `xor` operation) that permits to the raid controller to not read the others disks data: $P_{new} = C_{old} \oplus C_{new} \oplus P_{old}$.

In RAID 4, serial reads are not an issue. Serial writes only update the parity drive once, while random writes require updating the parity drive for each write (with three writes in three different disks can results other three writes in the parity disk). As a result, RAID 4 experiences poor write performance and is bottlenecked by the parity drive.

Case of random writes (the real bottleneck of RAID 4):



8.1.5 RAID 5



- Parity blocks are spread across all N disks, solving the issue of RAID 4 as writes are evenly spread across all drives.
- For random writes in RAID 5:
  1. Read the target and parity blocks.
  2. Use subtraction to calculate the new parity block.
  3. Write the target and parity blocks.

8.1.6 RAID 6

RAID 6 improves reliability beyond RAID 5 by **tolerating two simultaneous disk failures**. It uses Solomon-Reeds codes with two redundant schemes, resulting in two parity blocks. To implement RAID 6, an additional two disks are required, for a total of $N + 2$ disks.

8.1.7 RAID recap

RAID types:

- RAID 0: Striping only
- RAID 1: Mirroring only
  - RAID 0+1
  - RAID 1+0
- RAID 2: Bit interleaving (not used)
- RAID 3: Byte interleaving with redundancy (parity disk)
- RAID 4: Block interleaving with redundancy (parity disk)
- RAID 5: Block interleaving with redundancy (distributed parity block)
- RAID 6: Greater redundancy with tolerance for 2 failed disks.

Formulas:

	RAID 0	RAID 1	RAID 4	RAID 5	RAID 6
Capacity	N	$N/2$	$N-1$	$N-1$	$N-2$
Seq. Read	$N*S$	$(N/2)*S$	$(N-1)*S$	$(N-1)*S$	
Seq. Write	$N*S$	$(N/2)*S$	$(N-1)*S$	$(N-1)*S$	
Random Read	$N*R$	$N*R$	$(N-1)*R$	$N*R$	
Random Write	$N*R$	$(N/2)*R$	$R/2$	$(N/4)*R$	
How many disks can fail?	-	1	1	1	2

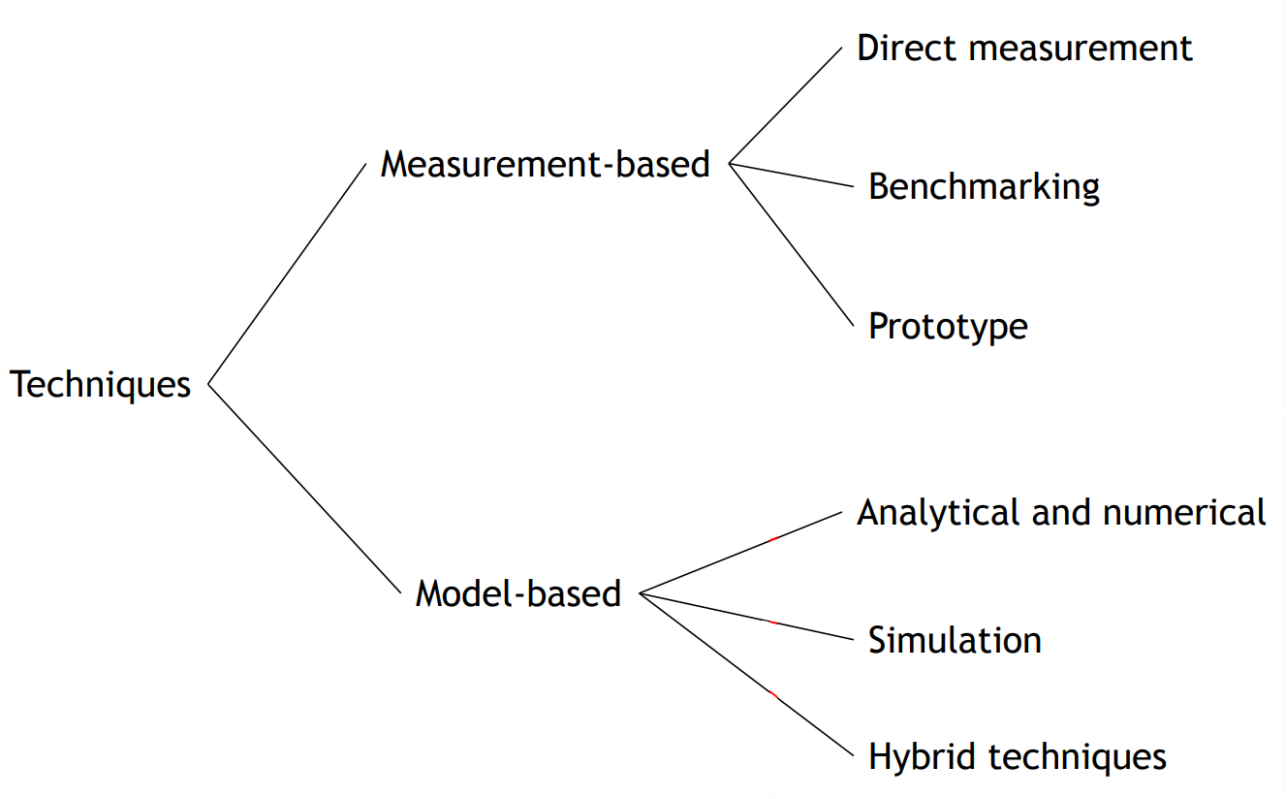
Best performance and most capacity ? raid 0 Greatest error recovery raid 1 or (1+0) or RAID 6 Most popular and well balanced RAID 5

RAID 6 is a configuration where there are, two Logical Block Address that work as parity LBA, for each row of data striped in the disks So, if you have N disks, a total amount of 2 disks of memory is used as parity

Then, when you need to write a datum, you need to read from the disk where the datum is, plus the two other disks where the parity blocks are Now, you can calculate the two new values of the parity blocks, and write them on the two respective disks, and finally write the new datum, replacing the old one

9 Performance

Measuring the quality of a system is a classic engineering question.



Possible models that we will se are:

- Queueing networks

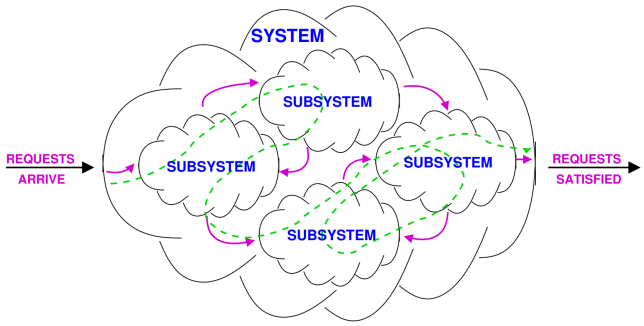
9.0.1 Queueing networks

Queueing theory is the theory behind what happens when you have a lot of jobs, scarce resources, and so long queue and delays. Examples of queues in computer systems:

- CPU uses a time-sharing scheduler
- Disk serves a queue of requests waiting to read or write blocks
- A router in a network serves a queue of packets waiting to be routed
- Databases have lock queues, where transactions wait to acquire the lock on a record

Different aspects characterize queuing models:

- **Arrival**: frequency of jobs is important: the average arrival rate λ (req/s).
- **Service**: the service time is the time which a computer spends satisfying a job: the average duration $\frac{1}{\mu}$ and μ is the maximum service rate.
- **Queue**: jobs or customers which cannot receive service immediately must wait in a buffer.
- **Population**: we can both have members of the population indistinguishable or divided into classes, whose members differs in one or more characteristics, for example, arrival rate or service demand.
- **Routing**: the policy that describes how the next destination is selected after finishing service at a station must be defined. It can be:
  - probabilistic: random destination
  - round robin: the destination chosen by the job rotates among all the possible ones
  - shortes queue: jobs are moved to the one other subsystem with the mallest number of jobs waiting



9.1 Operational laws

Operational laws represent the average behavior of any system, without assumptions about random variables. They are simple easy to apply equations.

If we observe a system we might measure these quantities: - T the length of time we observe the system - A the number of request arrivals we observe - C the number of request completions we observe - B the total amount of time during which the system is busy ($B \leq T$) - N the average number of jobs in the system - R the average time a request remains in the system (also called **residence time**). It corresponds to the period of time from when a user submits a request until that user's response is returned. - $\tilde{R}_k$ of a sub-system k is the **response time** and it's the average time spent by a job in k when enters the node. - Z is the think time which represents the time between processing being completed and the job becoming available as a request again, basically the time where in interactive systems the app is waiting the user.

From the quantities we can derive these equations:

- $\lambda = A/T$, the arrival rate
- $X = C/T$, the throughput or completion rate where C
- $U = B/T$, the utilization
- $S = B/C$, the mean service time per completed job. Note that is the average time that a job spends when IT IS SERVED.
- $U = XS$ which is useful in situations like "consider a serving rate of x where each serving need S time to be completed".
- $N = XR$ which is called the **little law**.

- $R = \frac{N}{X} - Z$ in case of **interactive** systems.

- $V_k = \frac{C_k}{C}$ is the visit count of sub-system k in the system with C as completion rate. It's possible that $C_k > C$ when there are "loops" in the model
- $C_k = V_k C$
- $X_k = V_k X$ is the force flow law and it captures the relationship between sub-system k with the system.
- $D_k = V_k S_k$ is the service demand and it represents the total time that the k sub-system is needed for a generic request. It's a crucial characteristic of the sub-system/resource.
- $\tilde{R}_k$ is the time spent by a job in a service station, counting the queue time.
- $R_k = V_k \tilde{R}_k$ is the residence time and is the time spent by a job in a service station *during all system time*: so accounting the visits of the component.
- $U_k = D_k X$ is the utilization of a resource/sub-system.

If the system is job flow balanced $C = A$ which means that $\lambda = X$. Also the above formulas can also be applied to a single sub system k.

9.2 Performance bounds

We only consider single class systems and determine asymptotic bounds on a system's performance indices. Our focus will be on highlighting the **bottleneck** and quantify its impact.

The considered models and the bounding analysis make use of the following parameters:

- K, the number of service centers
- $D = \sum_k D_k$
- $Dmax$, the largest service demand at any single center
- Z, the average think time, for interactive systems

And the following performance quantities are considered:

- X, the system throughput
- R, the system response time

We distinguish between **open** model and **closed** model.

9.2.1 Open Models

In open models we have the following bounds:

$$X(\lambda) \leq \frac{1}{D_{\max}}$$

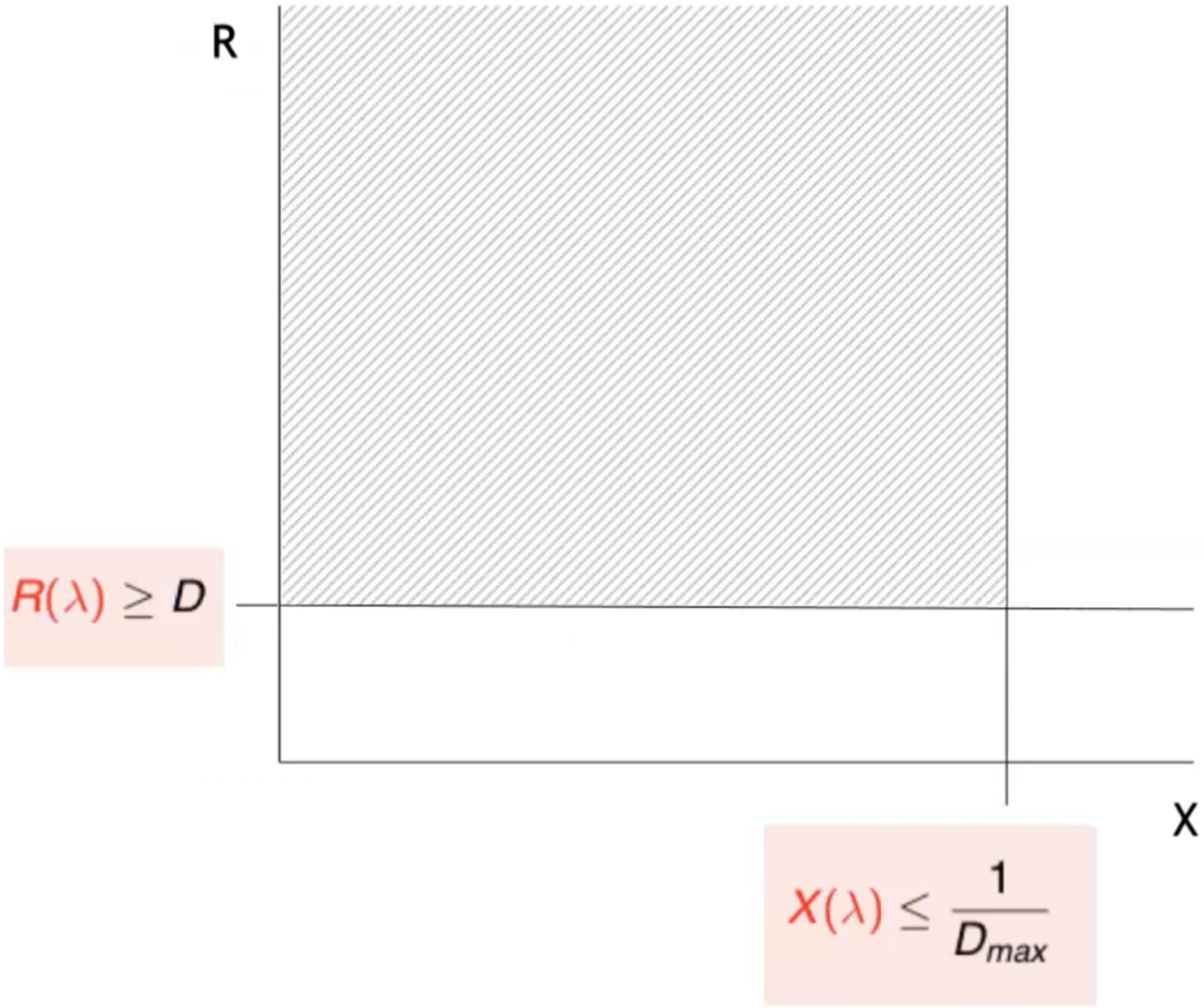
which is essentially derived from these relations:

$$D_{\max} = \max_k D_k$$

$$U \leqslant 1 \quad X \cdot D_k \leqslant 1$$

And the for the response time:

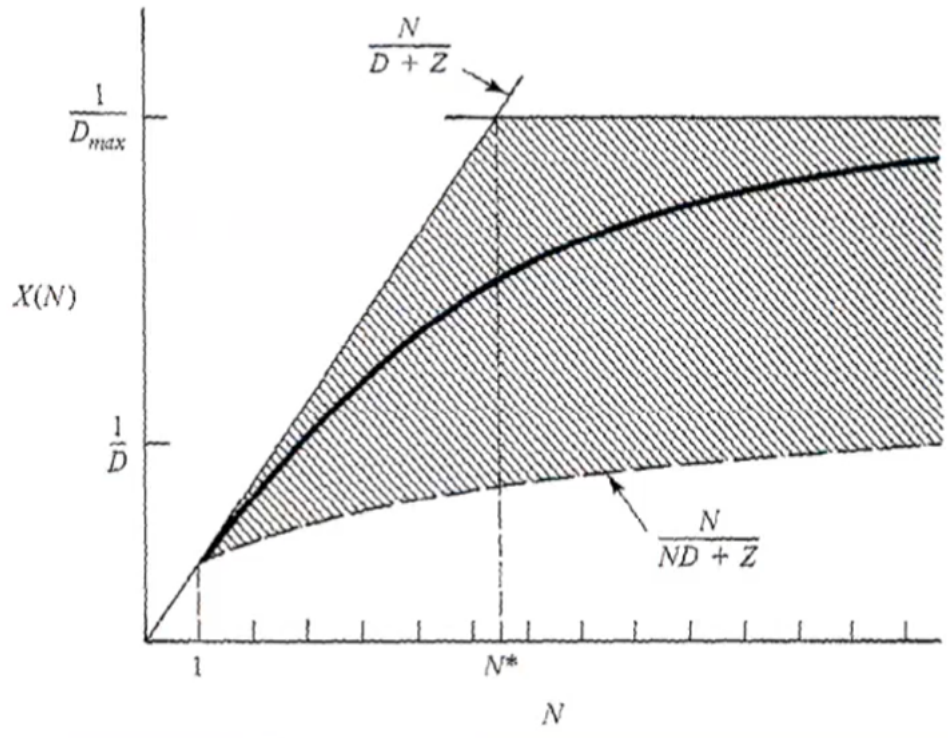
$$D \leq R(\lambda)$$



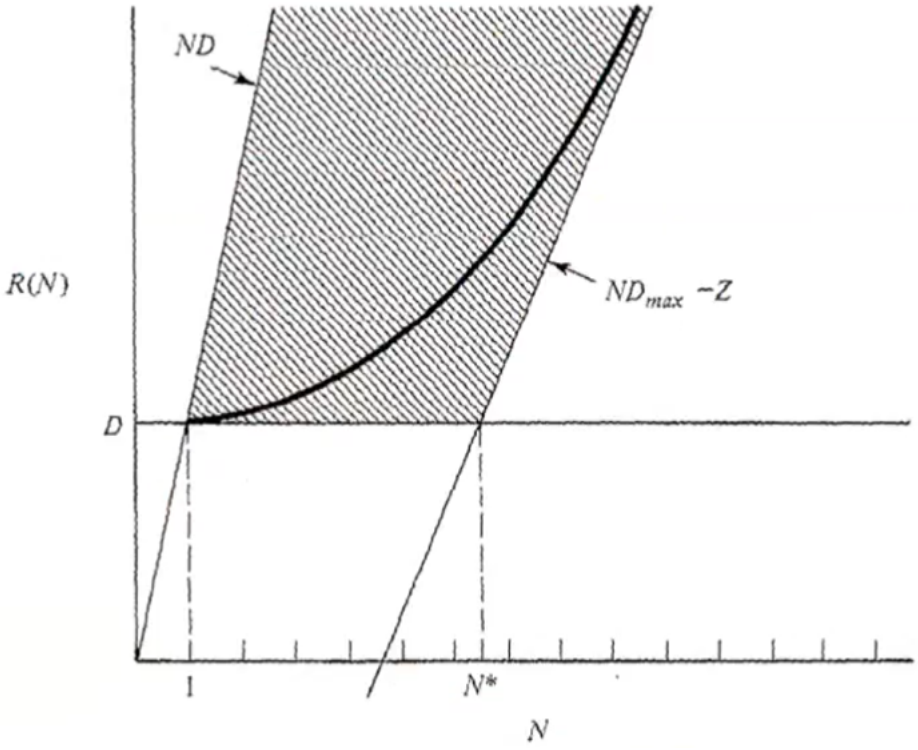
The upper bound for R cannot be defined while minimum response time is the sum of all the demands of all components..

9.2.2 Closed models

$$\frac{N}{ND+Z} \leq X(N) \leq \min\left(\frac{N}{D+Z}, \frac{1}{D_{\max}}\right)$$



$$\max(D, ND_{\max} - Z) \leq R(N) \leq ND$$



We can distinguish when the system is operating at "light load" from an heavy load.

- Light load: $\frac{N}{ND+Z} \leq X \leq \frac{N}{D+Z}$
- Heavy load: $\frac{N}{ND+Z} \leq X \leq D_{max}$

10 Virtualization

Virtualization adds a layer of abstraction between hardware and software for different purposes.

Virtualization Benefits:

- Provides flexibility and super-user (root) access to the Virtual Machine (VM) for fine-tuning settings and software customization.

Virtualization Issues:

- Suffers from performance interference and inability to deliver strong Service Level Agreement (SLA) guarantees.

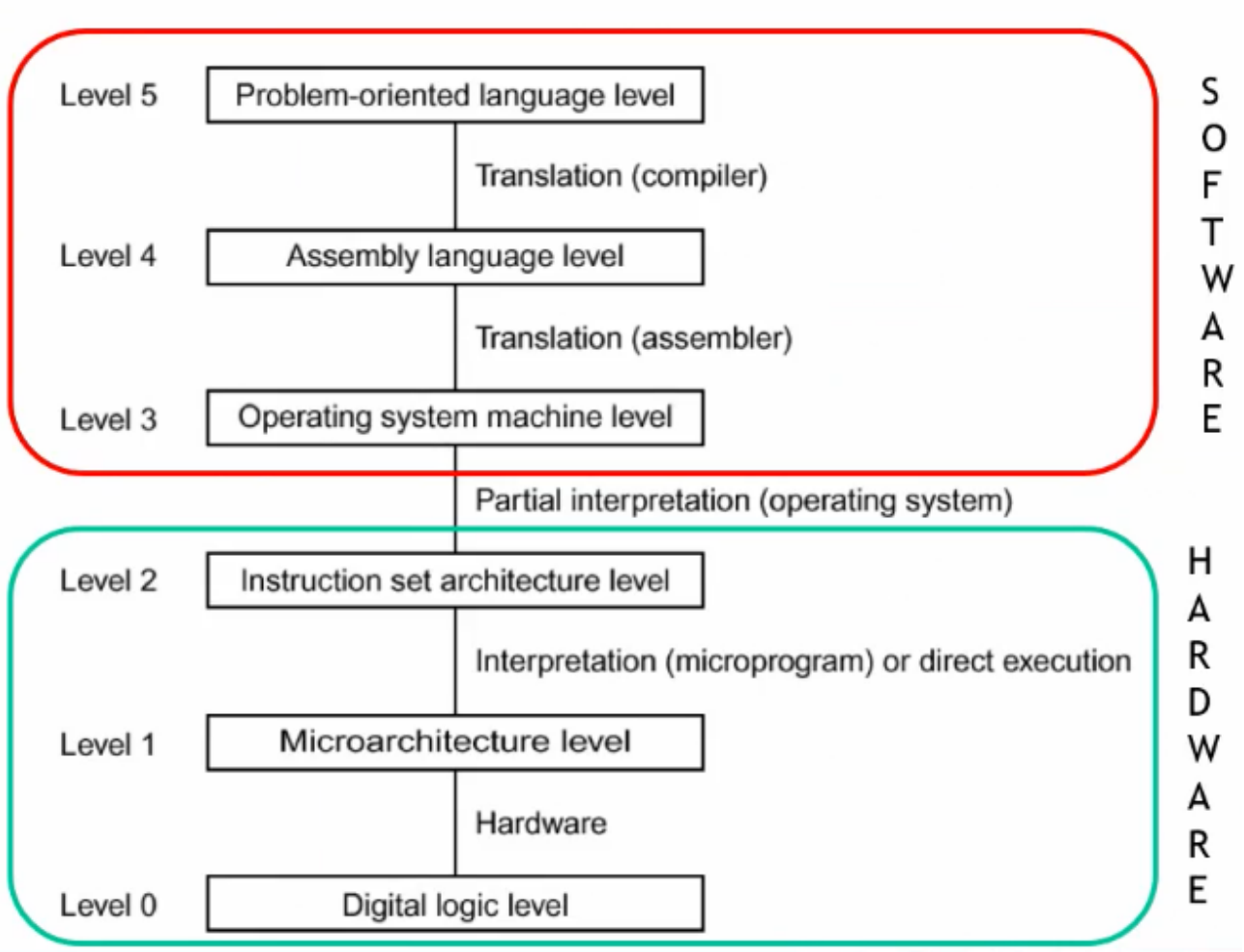
Properties of Virtual Machines (VMs):

- Partitioning: Divide a computer into separate, isolated segments where each segment functions as an individual system.
- Isolation: Every VM runs independently of the others, ensuring that activity (or failure) of one VM does not affect others.
- Encapsulation: The entire state of a VM can be captured into a file, which can be replicated or moved just like any other file. This simplifies the process of VM replication and migration.
- Hardware Independence: All VMs observe the same virtual hardware specifications independent of the host system. This homogeneity greatly simplifies VM provisioning and migration.

Other definitions:

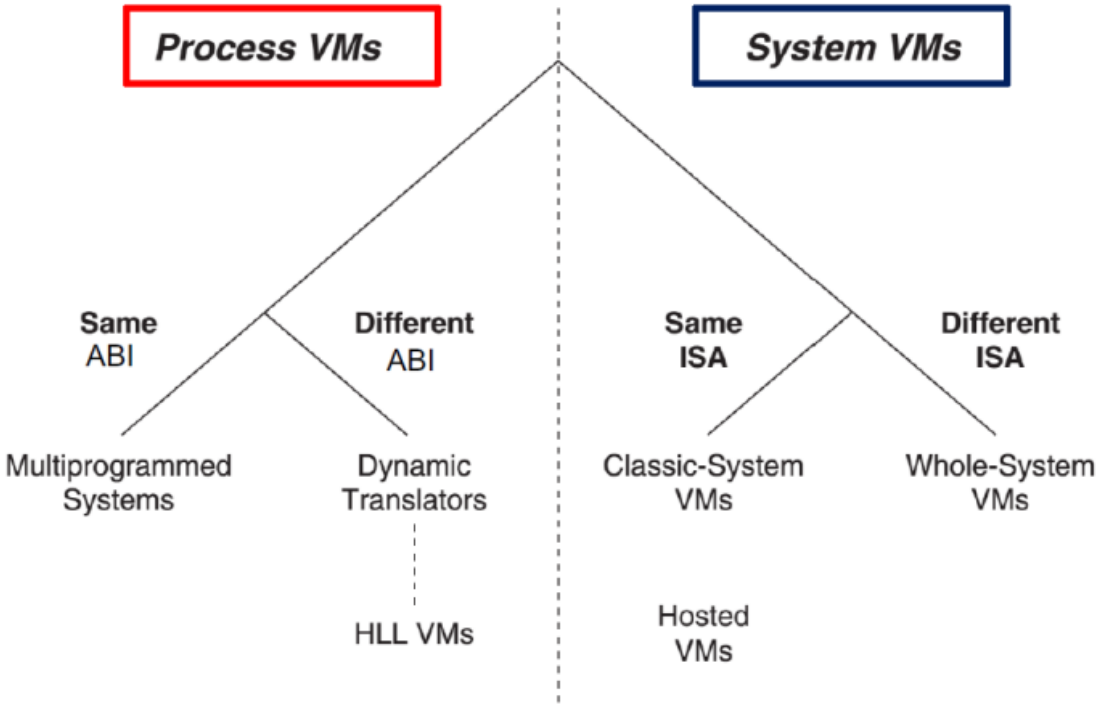
- Virtual Machine Monitor:
  - The virtualization software
- Hypervisor:
  - A virtualization software that runs directly on the hardware. It can be divided into:
    - type 1: no Host OS
    - type 2: the VMM is executing over an Host OS
- Virtual Machine Manager:
  - A VMM or Hypervisor that is also used to create, configure and maintain virtualized resources.
  - It provides a user-friendly interface to the underlying virtualization software

Recall ISA:



- User ISA: the ISA visible to application includes operations like sum, multiplication, logic operations, and branches.
- System ISA: this ISA is used when the OS interacts with hardware through drivers, memory management, and scheduling.

10.0.1 Type of VMs



10.0.1.1 Process VMs The process VMs "emulates" levels 0-3 of the architecture and has the capability to support individual processes. The virtualizing software is positioned at the ABI interface, sitting right above the OS. In addition to emulating user-level instructions, this software also mimics operating system calls.

- same ABI: basically a sandboxed application. Containers are exactly this. Process level VMs running on the same hw architecture.
- different ABI: what is generally called "emulation". Also an example of High Level Language VMs is the Java VM. JRE emulates everything up to the third level of the architecture.

10.0.1.2 System VMs Classic-System Virtual Machine

- Virtual Machine Monitor (VMM) directly installed on hardware
- Has the ability to intercept guest Operating System's (OS) interaction with hardware resources
- Among the most efficient VM architectures, allowing hardware to execute the same instructions of VMs
- Enables running two different OSs on the same hardware

Whole-System VM

- Entire software system is virtualized
- Involves different Instruction Set Architectures (ISAs) requiring emulation of both application and OS code, for instance, through binary translation
- No native execution is possible
- VMM and guest software are executed on top of a conventional host OS
- VM software needs to emulate the entire hardware environment and convert all the guest ISA operations into equivalent host OS calls

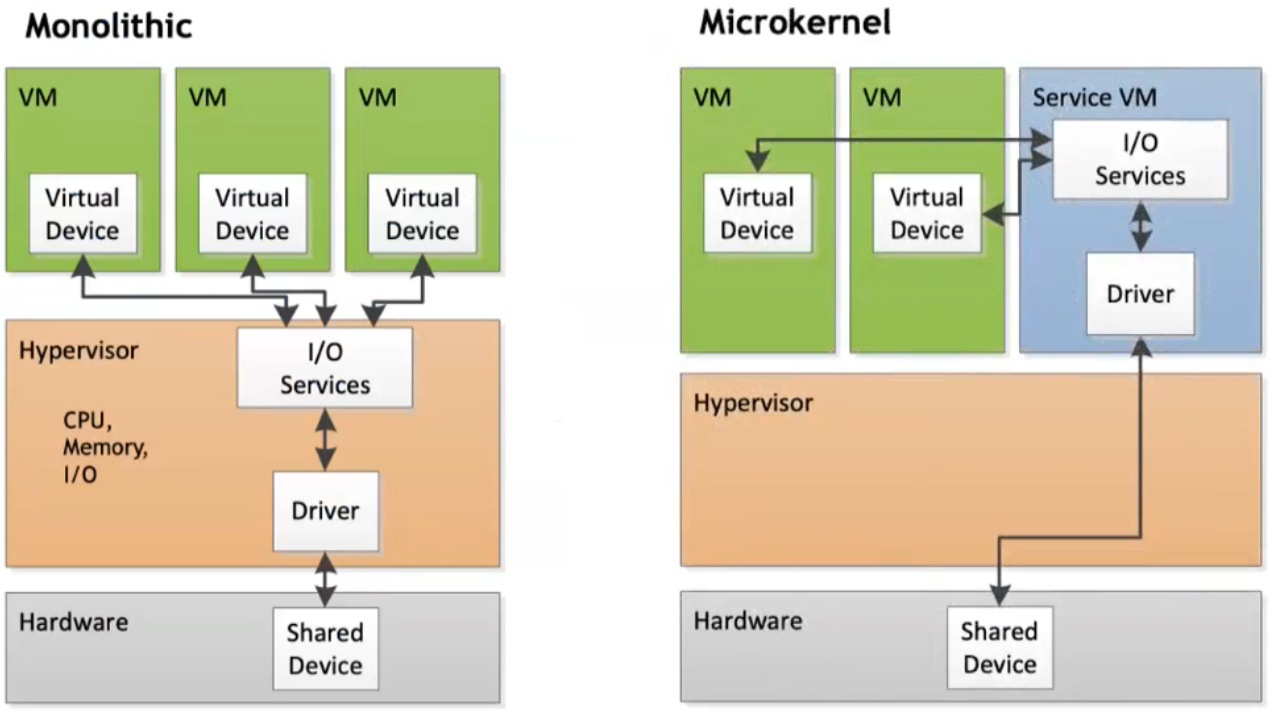
10.0.1.3 Type 1 hypervisor Type-1 hypervisor presents two fundamental approaches to managing I/O:

1. Monolithic:

In this approach, device drivers run within the hypervisor itself. Its main advantages include better overall performance and improved performance isolation. However, the limiting factor is that monolithic hypervisors can only run on hardware for which they already have drivers.

2. Microkernel (e.g. XEN - used by Amazon):

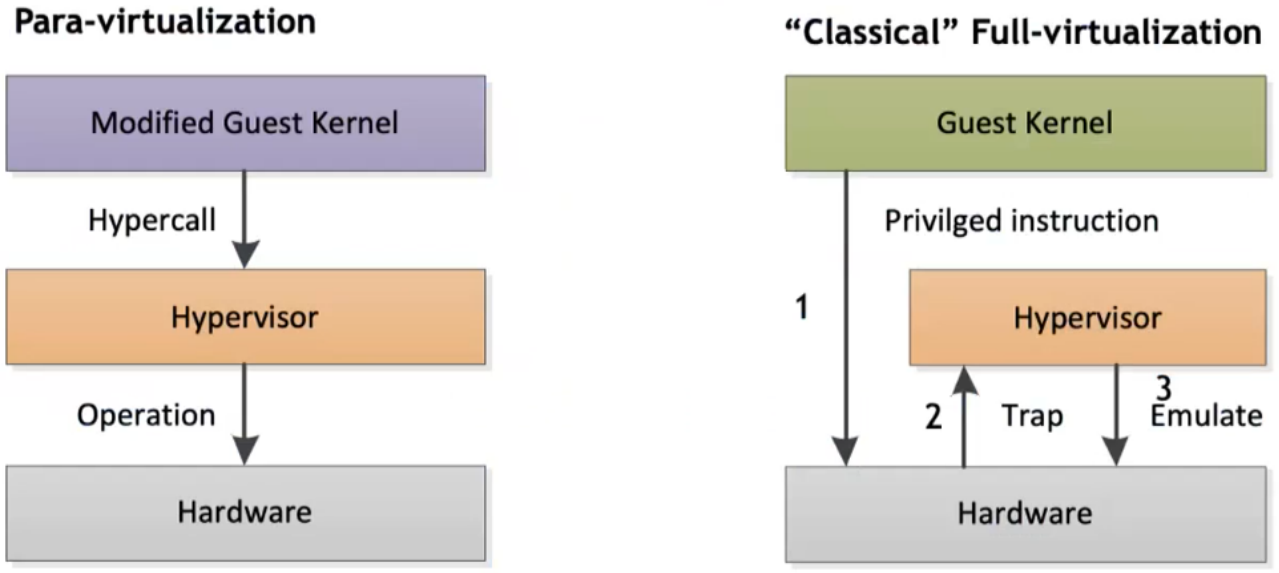
The microkernel approach is more flexible, with device drivers running inside a service Virtual Machine (VM's). This makes the hypervisor smaller and allows for the leveraging of an existing operating system (OS) driver's ecosystem. It can also use 3rd party drivers, although recompiling may occasionally be required.



10.0.1.4 Type 2 hypervisors Type-2 feature is that there is a Host OS that manages the hardware where the Virtual Machine Manager (VMM) runs. The benefits include its flexibility with regard to underlying hardware and simplicity in management and configuration, as the VMM can utilize the Host OS for GUI provision. However, conflicts might arise between the Host OS and Guest OS (e.g., Virtual Memory), and the Host OS can take up a significant set of physical resources, such as one core.

10.0.1.5 System VMs–Same ISA Virtualization Techniques Two main techniques:

- Paravirtualization offers an interface resembling but not identical to hardware, with the aim of reducing costly tasks, simplifying the Virtual Machine Manager (VMM) and enhancing performance. The Guest OS must be specifically ported to the para-API in paravirtualized systems, so it's incompatible with an unmodified OS.
- Full Virtualization completely imitates the underlying hardware with an unmodified Guest OS and Hypervisor mediation of guest's tasks and requests. However, it's not compatible with all architecture.



Almost all virtualization solutions today are grounded on full virtualization to avoid modifying the OS, and they generally incur a CPU overhead of 1-2%.

10.0.1.6 Containers Containers are pre-configured packages for executing code that offer predictable, repeatable, and immutable functionality when duplicated. They act as kernel-based VMs, sharing the host system kernel with other containers, which makes them extremely lightweight. Due to this, containers are often used in DevOps procedures like CI/CD. Containers are also useful for creating multi-user Platform-as-a-Service (PaaS) or Software-as-a-Service (SaaS).

10.0.1.6.1 Docker Docker is an open-source platform for building, shipping, and running applications inside containers using a Docker File. Docker Swarm is a container orchestration tool that competes with Kubernetes. It is developed by Docker and closely integrated within the Docker ecosystem. Docker is the standard for building a single container.

10.0.1.6.2 Kubernetes Kubernetes is for managing medium-large clusters and complex applications. Can run containers on many heterogeneous machines, increase/decrease performance, share load between containers, and start new containers on different machines if one fails.

10.1 Cloud Computing

Cloud Computing is a model that provides convenient and on-demand network access. Access is provided to a shared pool of configurable computing resources. These include networks, servers, storage, applications, and services. These resources can be swiftly provisioned and released with minimal management effort or interaction with the service provider.

Cloud Benefits

1. Cost Efficiency: Reduces IT expenditure.
2. Enhanced Performance: Swift software updates ensure improved functionality.
3. Infinite Storage: Provides virtually unlimited data storage.
4. Reliable Data: Promotes data reliability and integrity.
5. Universal Accessibility: Ensures access to documents anywhere, anytime.
6. Device Independence: Operates independently of the device used.

Cloud Limitations

1. Internet Dependence: Requires continuous internet connectivity.
2. Low-Speed Connection Inefficiency: May deliver poor performance with slow connections.
3. Limited Features: Might have feature restrictions.
4. Data Security Concern: Potential risks associated with data security.

10.1.1 Type of clouds

Public Cloud Infrastructure

- Accessible on a rental basis.
- Completely self-service for customers.
- Accountability is rooted in e-commerce.

Private Cloud Infrastructure

- An internally managed virtualization environment set up on organization's own servers.
- Provides total control over every aspect of the infrastructure.
- Involves capital investment and flexibility concerns.

Community Clouds

- Managed by several federated organizations.
- Economies of scale achieved by combining multiple organizations.
- Resources can be shared and used by one organization when others are not using them.
- Similar to private clouds.
- Either hosted locally or externally; common practice is to share participants' infrastructure.

10.1.1.1 Edge and Fog Computing Edge and fog computing are two "emerging" paradigms that exploits intelligence at the edge of the network (of sensors, of data hotspots ecc) in order to provide faster communications and services. The progressive convergence between Cloud Computing and the Internet of Things (IoT is resulting in a Computing Continuum.