Contents

1 Data Center and Computing Infrastructure **5**

2 Hardware Infrastructures **6**
2.1 System-level . . . 6
2.1.1 Computing Infrastructures and Data Center Architectures 6
2.1.1.1 Overview of Computing Infrastructures . . . 6
2.1.1.2 The Datacenter as a Computer . . . 13
2.1.1.3 Warehouse-Scale Computers . . . 16
2.1.1.4 Multiple Data Centers . . . 19
2.1.1.5 Availability in WSCs and DCs . . . 21
2.1.1.6 Architectural Overview of WSCs . . . 22
2.2 Node-level . . . 25
2.2.1 Server (computation, HW accelerators) . . . 25
2.2.1.1 Tower Server . . . 27
2.2.1.2 Rack Servers . . . 28
2.2.1.3 Blade Servers . . . 30
2.2.1.4 Machine Learning . . . 31
2.2.2 Storage (type, technology) . . . 34
2.2.2.1 Files . . . 35
2.2.2.2 HDD . . . 38
2.2.2.3 SSD . . . 41
2.2.2.4 RAID . . . 49
2.2.2.5 DAS, NAS and SAN . . . 63
2.2.3 Networking (architecture and technology) . . . 66
2.2.3.1 Fundamental concepts . . . 66
2.2.3.2 Switch-centric: classical Three-Tier architecture 68
2.2.3.3 Switch-centric: Leaf-Spine architectures . . . 70
2.2.3.4 Server-centric and hybrid architectures . . . 74
2.3 Building level . . . 75
2.3.1 Cooling systems . . . 77
2.3.2 Power supply . . . 80
2.3.3 Data Center availability . . . 81

3 Software Infrastructure **82**
3.1 Virtualization . . . 82
3.1.1 What is a Virtual Machine? . . . 82
3.1.1.1 Process VM . . . 84
3.1.1.2 System VM . . . 85
3.1.2 Virtualization Implementation . . . 86
3.1.3 Virtual Machine Managers (VMM) . . . 87
3.1.3.1 Full virtualization . . . 90
3.1.3.2 Paravirtualization . . . 91
3.1.3.3 Containers . . . 93
3.2 Computing Architectures . . . 95
3.2.1 Cloud Computing . . . 95
3.2.1.1 Server Consolidation . . . 95
3.2.1.2 Services provided by cloud . . . 96
3.2.1.3 Types of clouds . . . 99

4 Methods **100**
4.1 Reliability and availability of data centers . . . 100
4.1.1 Introduction . . . 100
4.1.2 Reliability and Availability . . . 103
4.1.3 Reliability Block Diagrams . . . 109
4.1.3.1 R out of N redundancy (RooN) . . . 114
4.1.3.2 Triple Modular Redundancy (TMR) . . . 115
4.1.3.3 Standby redundancy . . . 116
4.2 Disk performance . . . 117
4.2.1 HDD . . . 117
4.2.2 RAID . . . 122
4.3 Scalability and performance of data centers . . . 126
4.3.1 Evaluate system quality . . . 126
4.3.2 Queueing Networks . . . 128
4.3.2.1 Definition . . . 128
4.3.2.2 Characteristics . . . 129
4.3.3 Operational Laws . . . 133
4.3.3.1 Basic measurements . . . 133
4.3.3.2 Utilization Law . . . 135
4.3.3.3 Little's Law . . . 135
4.3.3.4 Interactive Response Time Law . . . 138
4.3.3.5 Visit count . . . 138
4.3.3.6 Forced Flow Law . . . 139
4.3.3.7 Utilization Law with Service Demand . . . 139
4.3.3.8 Response and Residence Times . . . 140
4.3.4 Bounding Analysis . . . 141
4.3.4.1 Introduction . . . 141
4.3.4.2 Asymptotic bounds . . . 142

Index **149**

2.2 Node-level

2.2.1 Server (computation, HW accelerators)

A **Server** is a computing system designed to manage, process, and deliver data or services to other computers (clients) over a network. In the context of datacenters and Warehouse-Scale Computers (WSCs), servers are the **atomic units of computation**, the fundamental building blocks of the entire system architecture.

Though **conceptually similar to a desktop PC**, servers **differ** in critical ways:

- They are **significantly more powerful**, scalable, and modular.
- They are designed for **continuous operation**, high availability, and **dense physical packaging** within a rack structure.

Servers in modern datacenters must balance **performance**, **density**, **power efficiency**, and **maintainability**.

Server Types

Server form factors **define how** servers are **physically organized** and **deployed** in datacenter environments. There are three principal types:

1. **Tower Server**s (section 2.2.1.1, page 27). Resemble traditional desktop PCs. They are ideal for small-scale deployments or low-density use cases.
   - ✔ **Pros**: Easy to upgrade, good cooling, low cost.
   - ✖ **Cons**: Large footprint, not optimized for rack deployment.
2. **Rack Server**s (section 2.2.1.2, page 28). Designed to slide **into a rack** (standardized shelves) in units (U); e.g., 1U = 1.75 inches. It is the most common server format in datacenters.
   - ✔ **Pros**: High compute density, easy to cable and scale.
   - ✖ **Cons**: Requires dedicated infrastructure (rack, cooling, power).
3. **Blade Server**s (section 2.2.1.3, page 30). Extremely compact and multiple blades share power, cooling, and networking through a **blade enclosure**. It is excellent for environments where space and energy are at a premium.
   - ✔ **Pros**: Highest density and modularity, centralized management.
   - ✖ **Cons**: Higher initial cost, vendor lock-in, increased heat density.

Server Architecture

Servers are typically **integrated into a tray or blade enclosure**, which contains:

- **Motherboard**: The central PCB that **interconnects all components**.
- **Chipset**: **Manages data flow** between CPU, RAM, storage, and peripherals.
- **Expansion slots**: For GPUs, network cards, and other **accelerators**.

Servers in WSCs tend to **use homogeneous hardware/software platforms** to simplify large-scale orchestration and maintenance.

Server Architecture

The **Motherboard** acts as a **central nervous system for the server**, it hosts:

- **CPU sockets** (e.g., up to 2 for dual Xeon systems)
- **DIMM slots** for RAM
- **Storage connectors** (e.g., SATA, NVMe)
- **NIC slots** (Network Interface Cards)

This level of configurability allows tailoring servers for compute-heavy, memory-bound, or I/O-intensive applications.

2.2.1.1 Tower Server

A **Tower Server** is a type of server designed in a vertical, standalone chassis that closely resembles a **standard tower desktop computer**. Unlike blade or rack servers, which are designed for high-density environments, tower servers prioritize **simplicity and accessibility**, often at the cost of physical footprint.

- **Structure**: Independent, **vertical** case (not meant for rack mounting).
- **Deployment**: Common in small businesses, branch offices, or settings where only a few servers are needed.
- **Internal layout**: Lots of **space for expansion components** like disks or PCIe cards.

Advantages

- ✔ **Scalability & Ease of Upgrade**. **Easy** to open and **upgrade**, users can add storage, memory, or cards as needed.
- ✔ **Cost-Effective**. Usually the **cheapest server type**, suitable for budget-constrained environments.
- ✔ **Easy Cooling**. Due to **low component density**, natural airflow is often sufficient. Less need for specialized cooling systems.

Limitations

- ✖ **Space Consumption** Tower servers consume **significant physical space** and don't scale well in quantity.
- ✖ **Basic Performance** They usually **offer lower performance and redundancy** compared to enterprise-grade rack or blade servers.
- ✖ **Cable Management** Not ideal for structured environments, cables can become messy and hard to manage.

2.2.1.2 Rack Servers

A **Rack Server** is a server built specifically to be **mounted vertically in standardized racks**, which are metallic shelves designed to hold multiple servers and IT components. Rack servers are the **default choice** in medium to large-scale datacenters, balancing compute density, modularity, and serviceability.

Physical Standardization

- **Servers are stored in racks** which follow a global standard:
  - **1U (Rack Unit)** = 1.75 inches (44.45 mm) in height.
  - Servers may come in 1U, 2U, 4U, up to 10U formats depending on power and component density.
- Racks also house other components: networking switches, storage arrays, power distribution units (PDUs), and cooling units.

This **standardization allows for efficient vertical stacking** of servers, optimizing physical space and simplifying cabling.

Racks as More Than Just Shelves

A rack is not just a mechanical holder, it is **part of the power, networking, and management infrastructure of the datacenter**:

- **Power Infrastructure**:
  - Shared power distribution units.
  - Battery backup (UPS).
  - Power conversion units.
- **Networking**:
  - Top of Rack (ToR) switches connect all servers in the rack to the datacenter network fabric.
  - Simplifies cabling and reduces latency.
- **Cooling**: designed for front-to-back airflow, aligned with datacenter cooling strategy (e.g., cold aisle containment).
- **Dimensions** can vary, but the classic rack is 19 inches wide and up to 48 inches deep.

Advantages

- ✔ **Modularity**: Individual servers can be **hot-swapped**, upgraded, or replaced **without disrupting others**.
- ✔ **Failure Containment**: Easy to **isolate and service a failed node** without bringing down the system.
- ✔ **Cable Management**: Organized by rear/backplanes or Top-of-Rack (ToR) switches.
- ✔ **Cost-Efficient Scaling**: **Scales vertically** at relatively lower incremental cost compared to other formats.

Challenges

- ✖ **High Power Demand**: **Higher component density** requires more energy and advanced cooling systems.
- ✖ **Thermal Hotspots**: Tight stacking can cause **hot zones**, especially with accelerator-heavy nodes.
- ✖ **Maintenance Overhead**: Large racks with tens of servers can become **complex to manage** physically as systems scale.

2.2.1.3 Blade Servers

Blade Servers represent the **most advanced evolution** in server form factors. They are designed to **maximize space efficiency** and **centralized manageability**, making them ideal for **large-scale enterprise datacenters** and **high-performance computing environments**.

A blade server is essentially a **stripped-down**, **ultra-thin server board** (the "blade") that fits into a blade enclosure, a shared chassis providing:

- Power
- Networking
- Cooling
- Centralized management

The enclosure conforms to the same **rack unit standard** (U), allowing it to integrate seamlessly with existing rack infrastructure. We can think of a blade system as a server equivalent of a modular bookshelf, where each "book" is a full server, and the "bookshelf" provides shared power, ventilation, and data connectivity.

Advantages

- ✔ **Compactness & Density**: The **smallest physical form factor** among all servers, allowing high-density deployments within a minimal footprint.
- ✔ **Minimal Cabling**: The **shared backplane** removes the need for complex cabling; power and network connections are centralized.
- ✔ **Centralized Management**: Blade systems typically include **unified management interfaces** (e.g., iLO, iDRAC) to monitor and configure blades collectively.
- ✔ **Scalability & Reliability**: New **blades can be added** with minimal disruption; enclosures support **load balancing** and **failover mechanisms**.
- ✔ **Uniform Infrastructure**: Simplifies deployment with **shared cooling**, **network fabrics**, and **power redundancy**.

Disadvantages

- ✘ **High Initial Cost**: Blade enclosures and vendor-specific blades often demand **significant upfront investment**.
- ✘ **Vendor Lock-In**: Typically, only blades from the **same manufacturer** (e.g., HPE, Dell, Cisco) **are compatible** with a given enclosure.
- ✘ **Thermal Density**: The compact form causes **higher heat output per rack unit**, requiring advanced HVAC design and monitoring.
- ✘ **Limited Flexibility**: While modular, blade systems trade off flexibility for density, upgrades and replacements may be **constrained by the enclosure's architecture**.

2.2.1.4 Machine Learning

Deepening: Machine Learning (supervised learning)

Machine learning (ML) is a branch of artificial intelligence (AI) and computer science that focuses on the using data and algorithms to enable AI to imitate the way that humans learn, gradually improving its accuracy (source).

UC Berkeley breaks out the learning system of a machine learning algorithm into three main parts:

1. A Decision Process: In general, machine learning algorithms are used to make a prediction or classification. Based on some input data, which can be labeled or unlabeled, your algorithm will produce an estimate about a pattern in the data.
2. An Error Function: An error function evaluates the prediction of the model. If there are known examples, an error function can make a comparison to assess the accuracy of the model.
3. A Model Optimization Process: If the model can fit better to the data points in the training set, then weights are adjusted to reduce the discrepancy between the known example and the model estimate. The algorithm will repeat this iterative "evaluate and optimize" process, updating weights autonomously until a threshold of accuracy has been met.

The main goal is to learn a target function that can be used for prediction. Given a training set of labeled examples $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, estimate the prediction function f by minimizing the prediction error on the training set:

$$y = f(x)$$

Where y is the output, f is the prediction function and the x is an image feature.

Deepening: Artificial Neural Network

The **Artificial Neural Network** is a computational model inspired by the human brain (perceptron). It consists of interconnected nodes (neurons) organized in layers to process and analyze data and used to learn data representation from data (learn features and the classifier/regressor).

The learning process of a Neural Network is as follows: Neurons make decisions (activation functions). There are wights, so the connections between neurons are strengthened or weakened through training- randomly initialized.
The (training data) Neural Networks learn from historical data and ex-

amples. Then, labeled data are provided.

Deepening: effects of ML and ANN

Deep learning models began to appear and be widely adopted, enabling specialized hardware to power a broad spectrum of machine learning solutions.
Since 2013, AI learning compute requirements have doubled every 3.5 months (vs. 18-24 months expected from Moore's Law).
To satisfy the growing compute needs for deep learning, **WSCs deploy specialized accelerator hardware**:

- Graphical Processing Units (GPUs) are used for data-parallel computations (the same program is executed on many data elements in parallel). In order to use parallel programming, high-level languages such as CUDA, OpenCL, OPENACC, OpenMP, and SYCL exist. This technique allows up to 1000x faster than CPU.
- Tensor Processing Unit (TPU), where Tensor is a n-dimensional matrix, are used for training and inference.
- Field-Programmable Gate Array (FPGA) are programmable hardware devices. The device user can program an array of logic gates ("configured") in the field instead of the people who designed it. An array of carefully designed and interconnected digital subcircuits that efficiently implement common functions, offering very high levels of flexibility. The digital subcircuits are called configurable logic blocks (CLBs).

  FPGA Applications in Data Centers:

  - Network acceleration: FPGAs can offload specific processing tasks from CPUs, improving overall network performance and reducing CPU workload.
  - Security acceleration: Encryption, decryption, and other security-related tasks can be accelerated using FPGAs, enhancing data centre security while maintaining performance.
  - Data analytics: FPGAs can accelerate specific algorithms in data analytics workloads, leading to faster data processing and analysis.
  - Machine learning: FPGAs can be configured to efficiently implement specific machine learning algorithms, potentially offering performance advantages for specialized tasks.

	Advantages	Disadvantages
CPU	• Easy to be programmed and support any programming framework. • Fast design space exploration and run your applications.	• Suited only for simple AI 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 programming based on Tensorflow.	• For applications and models based on the TensorFlow. • 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.

2.2.2 Storage (type, technology)

Data has significantly grown in the last few years due to sensors, industry 4.0, AI, etc. The growth favours the **centralized storage strategy** that is focused on the following:

- **Limiting redundant data**
- **Automatizing replication and backup**
- **Reducing management costs**

The *storage technologies* are many. One of the oldest but still used is the **Hard Disk Drive (HDD)**, a magnetic disk with mechanical interactions. Due to its mechanical movement, the **solid-state drive (SSD)** is the best solution (quality-price) because there are no mechanical or moving parts, and they are built out of transistors (NADN flash-based devices). The **non-volatile memory express (NVMe)** also exists, which is the **latest industry standard** for running PCIe[2] SSDs.

As for price classification, we can see that the NVMe is the most expensive solution:

1. NVMe (between 100€ and 200€ for 1TB)
2. SSD (between 70€ and 100€ for 1TB)
3. HDD (between 40€ and 60€ for 1TB)

For these reasons, it is reasonable to use a **hybrid solution** (HDD + SSD):

- A speed storage technology (**SSD or NVMe**) as **cache** and **several HDDs for storage**. It is a combination used by some servers: 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 SDD with a large HDD in a single unit.

[2] **PCIe (peripheral component interconnect express)**. is an interface standard for connecting high-speed components

2.2.2.1 Files

An **OS can see the disks 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**[3] **to simplify the access to the disk**. Typical cluster sizes range from 1 disk sector (512 B, or 4 KB) to 128 sectors (64 KB).

Each *cluster* contains:

- **File data**. The actual content of the files.
- **Metadata**. The information required to support the file system:
  - File names
  - Directory structures and symbolic links
  - Creation, modification, and last access dates
  - Security information (owners, access list, encryption)
  - **Links to the LBA where the file content can be located on the disk**

The disk can thus contain **several types of clusters**:

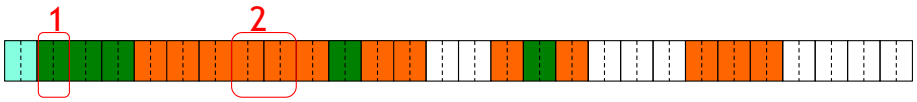
- Metadata:
  - Fixed position (to bootstrap the entire file system)
  - Variable position (to hold the folder structure)
- File data (the actual content of the files)
- Unused space (available to contain new files and folders)

Figure 4: A cluster can be seen visually as an array. In this image, for example, we've shown three types of cluster: metadata fixed position (azure), metadata variable position (green), file data (orange), unused space (white).

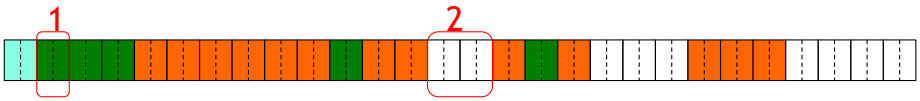
[3] **Clusters** are the minimal units an OS can read from or write to a disk.

The following explanation introduces some basic operations on the files to see what happens inside the disks.

- **Reading**. To read a file:
  1. Access the metadata, variable position (because it contains the folder structure), to locate its block;
  2. Access the blocks to read the contents of the file.



- **Writing**. To write a file:
  1. Access the metadata, variable position (because it contains the folder structure), to find free space.
  2. Write the data in the allocated blocks (cluster type: unused space).



Since the *file system can only access clusters*, the **actual space taken up by a file on a disk is always a multiple of the cluster size**. Given:

- s, the *file size*
- c, the *cluster size*

Then the **actual size on the disk** a can be calculated as:

$$a = \text{ceil}\left(\frac{s}{c}\right) \times c \tag{1}$$

Where ceil rounds a number up to the nearest integer. It's also possible to calculate the **amount of disk space wasted by organising the file into clusters** (**wasted disk space** w):

$$w = a - s \tag{2}$$

A formal way to refer to wasted disk space is **internal fragmentation** of files.

Example 4: internal fragmentation

- File size: 27 byte
- Cluster size: 8 byte

The *actual size* on the disk is:

$$a = \text{ceil}\left(\frac{27}{8}\right) \cdot 8 = \text{ceil}\,(3.375) \cdot 8 = 4 \cdot 8 = 32 \text{ byte}$$

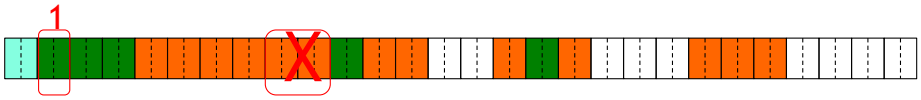
And the internal fragmentation w is:

$$w = 32 - 27 = 5 \text{ byte}$$

- **Deleting**. To delete a file:

  1. Just update the metadata, variable position (because it contains the folder structure), to say that the blocks where the file was stored are no longer used by the OS.

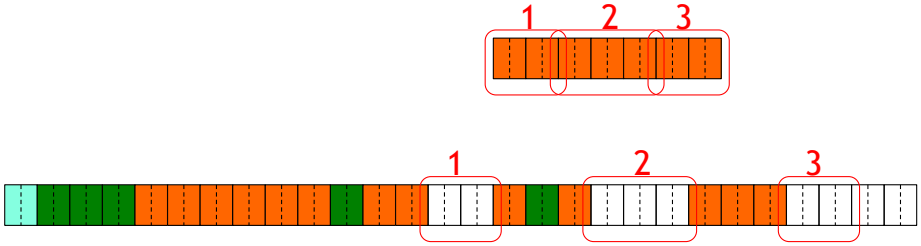
  Deleting a file never actually deletes the data on the disk: if a new file is written to the same clusters, the old data is replaced by the new.



- **External fragmentation**. As the disk's life evolves, there might **not be enough space to store a file contiguously**.

  In this case, the file is split into smaller chunks and inserted into the free clusters spread over the disk.

  The effect of **splitting a file into non-contiguous clusters** is called **external fragmentation**.



2.2.2.2 HDD

A **Hard Disk Drive (HDD)** is a **data storage device that uses rotating disks (platters) coated with magnetic material**.

Data is read randomly, meaning individual data blocks can be stored or retrieved in any order rather than sequentially.

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.

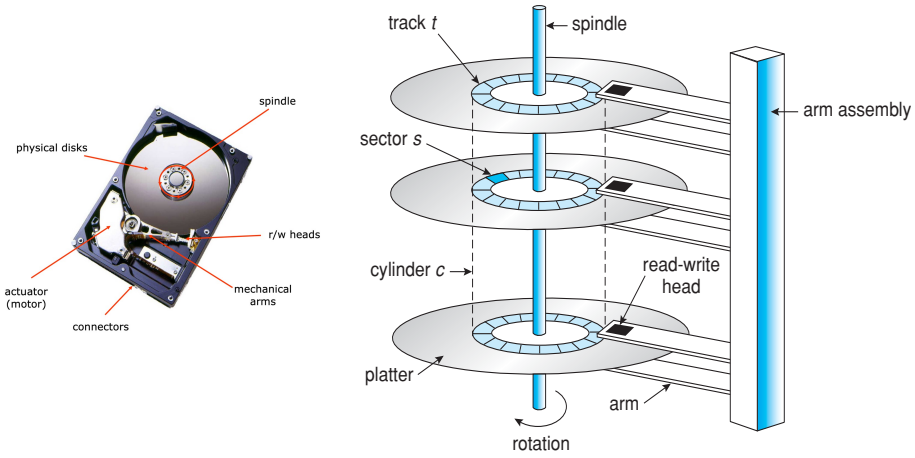


Figure 5: Hard Drive Disk anatomy.

Externally, hard drives expose a large number of **sectors** (blocks):

- Typically, 512 or 4096 bytes.
- Individual **sector writes** are **atomic**.
- Multiple sectors write it may be interrupted (**torn write**[4]).

The geometry of the drive:

- The sectors are arranged into **tracks**.
- A **cylinder** is a particular track on multiple platters.
- Tracks are arranged in concentric circles on **platters**.
- A disk may have multiple double-sided platters.

The **driver motor spins the platters at a constant rate**, measured in **Revolutions Per Minute (RPM)**.

[4]Torn writes happen when only part of a multi-sector update is written successfully to disk.

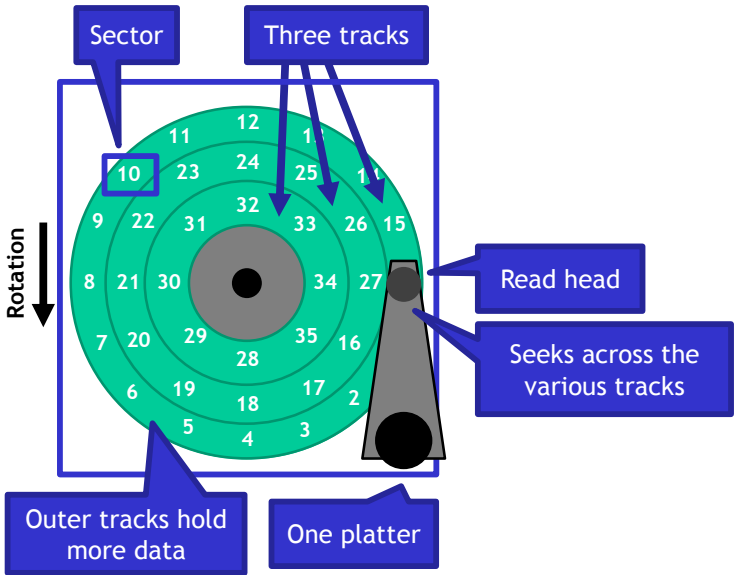


Figure 6: Example of HDD geometry.

Given the architecture of the HDD, there are **four types of delay**:

- **Rotational Delay** is the **time to rotate the desired sector to the read head**, and it's related to RPM.
- **Seek Delay** is the **time to move the read head to a different track**.
- **Transfer time** is the **time to read or write bytes**.
- **Controller Overhead** is the **overhead for the request management**.

In order to reduce the delay in the HDD, the companies use a high-speed and tiny memory (8, 16 or 32 MB) called **cache** (also called **track buffer**). The cache is used when there is a:

- **Read caching**. It reduces read delays due to seeking and rotation.
- **Write caching**. It is divided into two different implementations:
  - **Write Back cache**: the drive reports that writes are complete after they have been cached. The disadvantage is that it has an inconsistent state if the power goes off before the write-back event.
  - **Write Through cache**: the drive reports that writes are complete after they have been written to disk.

So, the **caching helps improve disk performance**. The critical idea under caching is that if there is a **queue of requests to the disk, they can be reordered to improve performance**. The estimation of the request length is feasible, knowing the position of the data on the disk.

There are several **scheduling algorithms**:

- **First Come, First Serve (FCFC)**. It is the most basic scheduler, serving requests in order. The ***disadvantage*** is that there is much time spent seeking.
- **Shortest Seek Time First (SSTF)**. It is primary purpose is to minimize seek time by always selecting the block with the shortest seek time. The ***advantage*** is that it is optimal and can be easily implemented. The main ***disadvantage*** is that it is prone to starvation.
- **SCAN**, otherwise known as the Elevator Algorithm. The head sweeps across the disk, servicing requests in order. The ***advantage*** is that it performs reasonably well and does not suffer starvation. The ***disadvantage*** is that the average access times are higher for requests at high and low addresses.
- **C-SCAN (Circular SCAN)**. It is like the SCAN algorithm, but only service requests in one direction. The ***advantage*** is fairer than SCAN. However, the ***disadvantage*** is that it has worse performance than SCAN.
- **C-LOOK**. It is a C-SCAN variant that peeks at the upcoming addresses in the queue. The head only goes as far as the last request.

2.2.2.3 SSD

The **solid-state drive (SSD)** does not have mechanical or moving parts like an HDD. It is built out of transistors (like memory and processors). It has higher performance than HDD.

It stores bits in cells. Each cell can have:

- Single-Level Cell (SLC), a single bit per cell.
- Multi-Level Cell (MLC), two bits per cell.
- Triple-Level Cell (TLC), three bits per cell.
- And so on... QLC, PLC, etcetera.

Internally, the SSD has a lot of NAND flashes, which are organized into Pages and Blocks. Some terminology:

- A **Page** contains **multiple logical block** (e.g. 512 B - 4 KB) **addresses** (LBAs). It is the **smallest unit that can be read/written**. It is a sub-unit of an erase block and consists of the number of bytes which can be read/written in a single operation. The states of each page are:
  - **Empty** (`ERASED`): it does not contain data.
  - **Dirty** (`INVALID`): it contains data, but this data is no longer in use (or never used).
  - **In use** (`VALID`): the page contains data that can be read.
- A Block (or **Erase Block**) typically consists of **multiple pages** (e.g. 64) with a total capacity of around 128-256 KB. It is the **smallest unit that can be erased**.

When passing from the HDD to SDD, there is a problem known as Write Amplification (WA). **Write amplification (WA)** is an **undesirable phenomenon associated with flash memory and solid-state drives (SSDs)** where the actual amount of information physically written to the storage media is a multiple of the logical amount intended to be written.

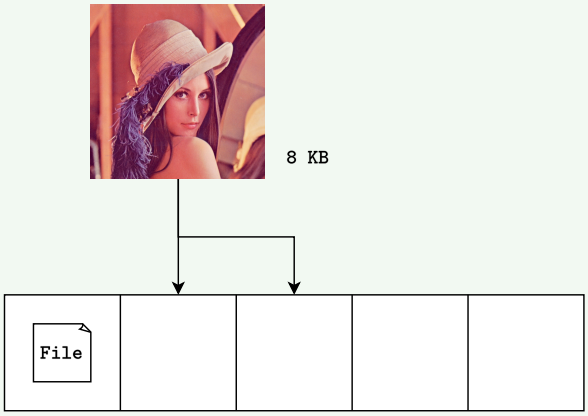
Example 5

Given a hypothetical SSD:

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

Let us write a 4 KB text file to the brand-new SSD. The overall writing time is 4 seconds (write speed $\times$ file dimension, 1 KB/s $\times$ 4 KB).

Now, let us write an 8 KB picture file for the almost brand-new SSD; thankfully, there is space. The overall Writing time is 8 seconds, and the calculation is the same as above.



Now, consider that the first file inserted on the first page is unnecessary.

Finally, let's write a 12 KB pic to the SSD. Theoretically, the image should take 12 seconds. However, it is wrong! The SSD has only two empty pages and one dirty page (invalid). Then, the operations are:

1. Read block into cache.
2. Delete page from cache (set dirty page).
3. Write a new picture into the cache.
4. Erase the old block on the SSD.
5. Write cache to SSD.

The OS only thought it was writing 12 KBs of data when the SSD had to read 8 KBs (2 KB/s) and then write 20 KBs (1 KB/s), the entire block. The writing should have taken 12 seconds but took $4 + 20 = 24$ seconds, resulting in a write speed of 0.5 KB/s, not 1 KB/s.

A direct mapping between Logical and Physical pages is not feasible inside the SSD. Therefore, each SSD has an FTL component that makes the SSD *look like an HDD*.

The **Flash Translation Layer (FTL)** is placed in the hierarchy between the **File System** and **Flash Memory**. It aims to do **three main actions**:

1. **Data Allocation** and **Address Translation**: It efficiently reduces Write Amplification effects (see page 41); the program pages within an erased block in order (from low to high pages), called **Log-Structured FTL**.
2. **Garbage collection**: reuse of pages with old data (Dirty/Invalid).

3. **Wear levelling**: FTL should spread across the blocks of the flash, ensuring that all of the blocks of the device wear out roughly simultaneously.

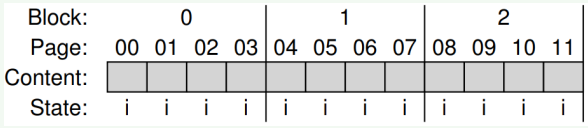
Example 6: Log-Structured FTL

Assume that a page size is 4 KB and a block consists of four pages. The write list is (`Write(pageNumber, value)`):

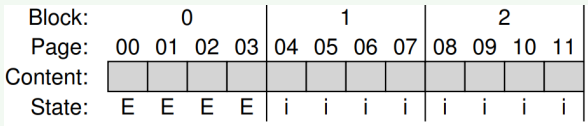
- `Write(100, a1)`
- `Write(101, a2)`
- `Write(2000, b1)`
- `Write(2001, b2)`
- `Write(100, c1)`
- `Write(101, c2)`

The steps are:

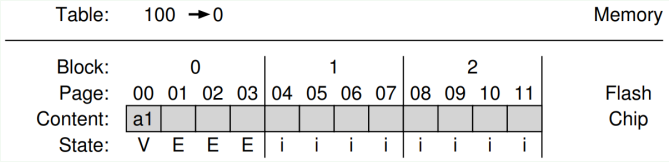
1. The initial state is with all pages marked as `INVALID(i)`:



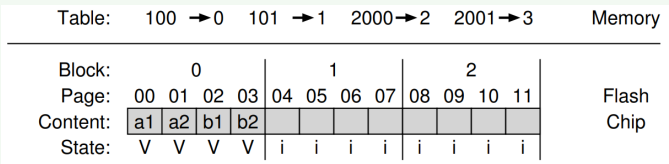
2. Erase block zero:



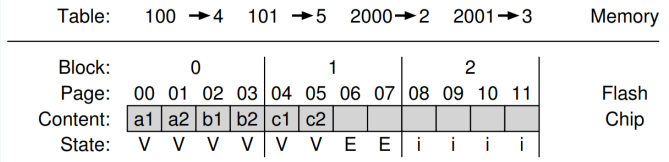
3. Program pages in order and update mapping information (`Write(100, a1)`):



4. After performing four writes (`Write(100, a1)`, `Write(101, a2)`, `Write(2000, b1)`, `Write(2001, b2)`):



5. After updating 100 and 101:



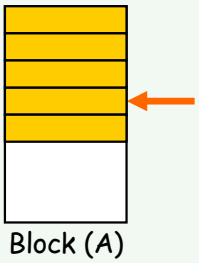
When an existing page is updated, then the old data becomes obsolete. The **old versions of data are called garbage**, and (sooner or later) garbage pages must be reclaimed for new writes to take place.

The **Garbage Collection** is the **process of finding garbage blocks and reclaiming them**. It is a simple process for fully garbage blocks but more complex for partial cases.

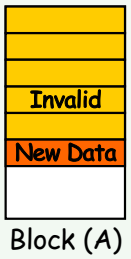
Example 7: how garbage collection works

The steps are:

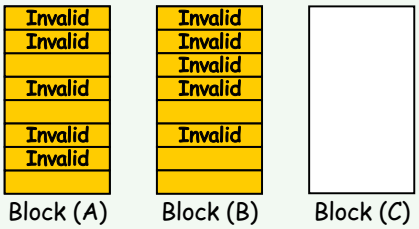
1. Update request for existing data:



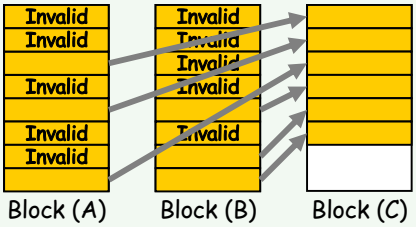
2. Find a free page, and save the new data:



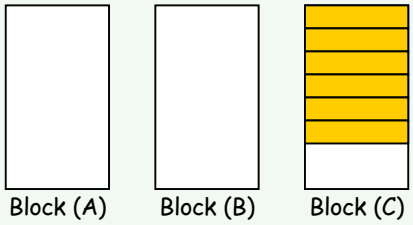
3. This scenario may continue until there are not enough free blocks:



4. Collect valid pages into a free block:



5. Update the map table and erase invalid (obsolete) blocks:



⚠ Problem 1: the Garbage Collection is too expensive!

The Garbage Collection is **expensive**. It requires reading and rewriting of live data. Ideal garbage collection is a reclamation of a block that consists of only dead pages.

✔ Partial solution

Garbage Collection **costs depend on the amount of data blocks that must be migrated**. The **solution** to alleviate the problem is to overprovision the device by **adding extra flash capacity** (cleaning can be delayed) and **running the garbage collection in the background** using less busy periods for the disk.

⚠ Problem 2: the Ambiguity of Delete

When performing background Garbage Collection, the SSD assumes to know which pages are invalid. However, most file systems do not truly delete data. For example, on Linux, the "delete" function is `unlink()`, removing the file meta-data but not the file itself.

1. File is written on SSD
2. File is deleted
3. The Garbage Collection executes:
   - 9 pages look valid to the SSD;
   - BUT the OS knows only 2 pages are valid.

✔ Partial solution

New SSD SATA command TRIM (SCSI - UNMAP). The **OS tells the SSD that specific LBAs** (page 35) **are invalid and may be garbaged by the Garbage Collection**.

⚠ Problem 3: Mapping Table Size

The **size of the page-level mapping table is too large**. In fact, with a 1 TB SSD with a 4-byte entry per 4 KB page, 1 GB of DRAM is needed for mapping!

✔ Partial solution

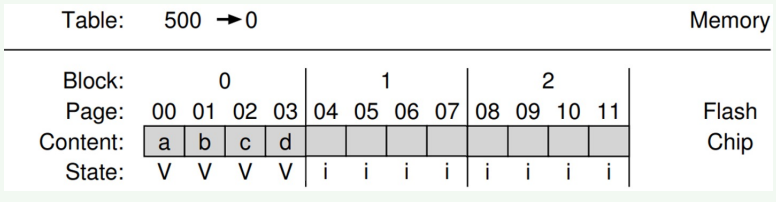
Exists some approaches to reduce the costs of mapping:

- **Block Mapping** (block-based mapping). Mapping at block granularity to reduce the size of a mapping table. With this technique, there is a small writing problem: the FTL must read a large amount of live data from the old block and copy it into a new one.

Example 8: Block Mapping

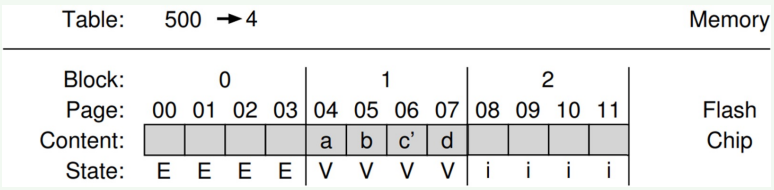
The first four writes:

- `Write(2000, a)`
- `Write(2001, b)`
- `Write(2002, c)`
- `Write(2003, d)`



And finally the last one:

- `Write(2002, c')`

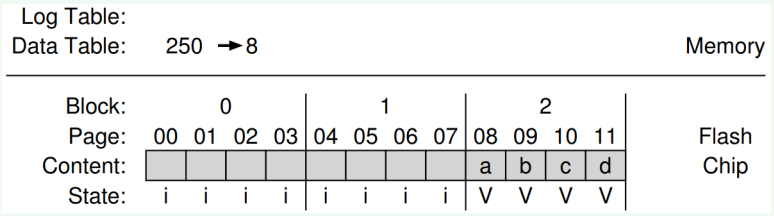


- **Hybrid Mapping**. FTL maintains two tables: **log blocks** (page mapped) and **data blocks** (block mapped). The FTL will consult the page mapping table and block mapping table when looking for a particular logical block.

Example 9: Hybrid Mapping

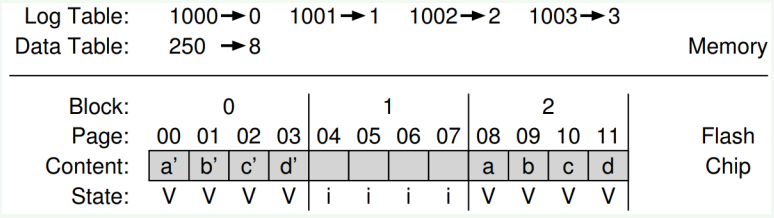
Let's suppose the following sequence:

- `Write(1000, a)`
- `Write(1001, b)`
- `Write(1002, c)`
- `Write(1003, d)`

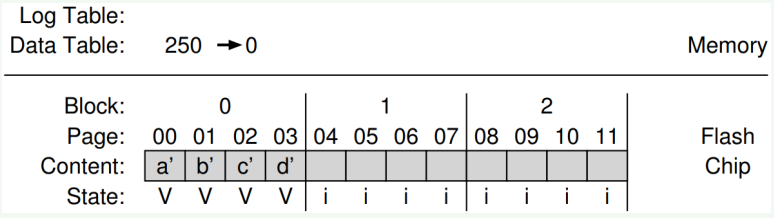


Let's update some pages:

- `Write(1000, a')`
- `Write(1001, b')`
- `Write(1002, c')`
- FTL updates only the page mapping information



When needed, FTL can perform MERGE operations:



- **Page Mapping plus Caching**. The basic idea is to **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. It will perform well without high memory cost if the cache can contain the necessary working set. Cache miss overhead exists; we need to accept it.

✔ The importance of Wear Leveling

As we have mentioned, the wear leveling is essential. The erase/write cycle is limited in Flash Memory. All blocks should wear out roughly at the same time.

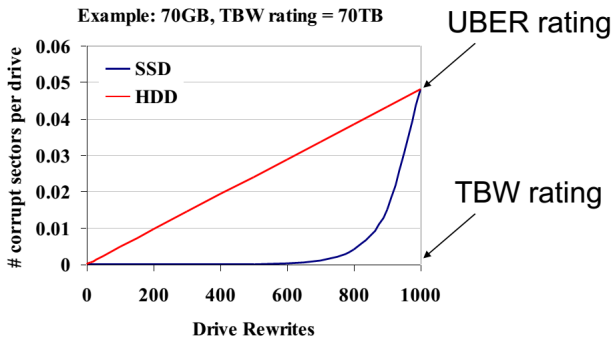
The log-structured approach and garbage collection help spread writing. However, a block may contain cold data: the FTL must periodically read all the live data from such blocks and re-write it elsewhere. A **disadvantage** is that the wear levelling **increases the write amplification** of the SSD and consequently decreases performance. However, to **partially fix** this, a simple policy to apply is that each flash block has an **Erase/Write Cycle Counter** and maintains the value of:

$$|\text{Max}\,(\text{EW cycle}) - \text{Min}\,(\text{EW cycle})| < e \tag{3}$$

HDD vs SSD

Exists two metrics:

- **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**.
- **Endurance rating: Terabytes Written (TBW)**. It is the total amount of data that can be written into an SSD before it is likely to fail. **The number of terabytes that may be written to the SSD while still meeting the requirements**.



2.2.2.4 RAID

RAID (Redundant Array of Independent Disks) is a **data storage virtualization technology**[5] that **combines multiple physical disk drive components into one or more logical units for the purposes of data redundancy**, **performance improvement**, or both. This contrasts the previous concept of highly reliable mainframe disk drives, which were referred to as **Single Large Expensive Disks (SLED)**, also called **Just a Bunch of Disks (JBOD)** method where each disk is a separate device with a different mount point.

The data are striped across several disks accessed in parallel:

- **High data transfer rate**: large data accesses (heavy I/O operations).
- **High I/O rate**: small but frequent data accesses (light I/O operations).
- **Load Balancing** across the disks.

Two techniques exist to guarantee these features: **data striping** (improve performance) and **redundancy** (improve reliability).

Data Striping is the technique of **segmenting logically sequential data**, such as a file, so that **consecutive segments are stored on different physical storage devices**. A small quantity of terminology:

- **Striping**: **data are written sequentially** (a vector, a file, a table, etc) in units (stripe units such as bit, byte, and blocks) **on multiple disks** according to a cyclic algorithm (round robin).
- **Stripe unit**: the **dimension of the data unit 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 disks' queue length (and time).
  2. Multiple disks will execute **single Multiple-Block I/O requests** in parallel, increasing the transfer rate of a single request.

The **redundancy technique is introduced because** the more physical disks in the array, the more significant the size and performance gains, but the **larger the probability of failure of a disk**.

In fact, the *probability of a failure* (assuming independent failures) in an array of 100 disks is 100 higher than the probability of a failure of a single disk! For **example**, if a disk has a **Mean Time To Failure (MTTF)** of 200.000 hours (23 years), an array of 100 disks will show an MTTF of 2000 hours (3 months).

[5] I/O virtualization: data are distributed transparently over the disks, then no action is required of the users.

The **Redundancy** is the **technique of data duplication or error correcting codes** (stored on disks different from the ones with the data) **that are computed to tolerate loss due to disk failures**. Since write operations must also update the redundant information, their **performance is worse than traditional writing**.

Data is distributed across the drives in one of several ways, referred to as **RAID levels**, depending on the required level of redundancy and performance. The different schemes, or data distribution layouts, are named by the word *RAID* followed by a number, for example RAID 0 or RAID 1. Each scheme, or RAID level, provides a different balance among the key goals: reliability, availability, performance, and capacity. The RAID levels are:

- `RAID 0` striping only
- `RAID 1` mirroring only
  - `RAID 0+1` nested levels
  - `RAID 1+0` nested levels
- `RAID 2` bit interleaving (not used)
- `RAID 3` byte interleaving - redundancy (parity disk)
- `RAID 4` block interleaving - redundancy (parity disk)
- `RAID 5` block interleaving - redundancy (parity block distributed)
- `RAID 6` greater redundancy (2 failed disks are tolerated)

Note: these notes do not study the levels `RAID 2` and `RAID 3`.

Topic	Page
`RAID 0`	51
`RAID 1`	53
`RAID 0 + 1`	54
`RAID 1 + 0`	54
`RAID 4`	57
`RAID 5`	59
`RAID 6`	60
Comparison and characteristics of RAID levels	61

Table 3: RAID - Table of Contents.

RAID 0

In RAID 0, the data are **written on a single logical disk and split into several blocks distributed across the disks** according to a striping algorithm.

When is it used?

It is used where **performance** and **capacity** are the primary concerns. These mean that a minimum of two drives is required.

Advantages

- **Lower cost** because it does not employ redundancy (no error-correcting codes are computed and stored).
- **Best write performance** (it does not need to update redundant data and is parallelized).

Disadvantages

Single disk failure will result in data loss.

How does it work?

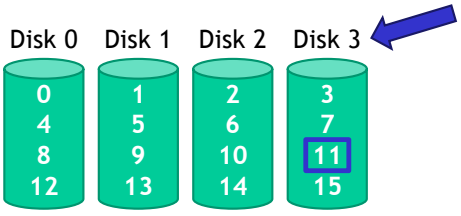
The key idea is to present an **array of disks as a single large disk** and maximize parallelism by striping data across all N disks.

Now, we will give some metrics to understand how it is possible to **calculate access to a specific data block** and compare the **performance** of different RAID technologies.

To **access to a specific data blocks**, the formulas are:

$$\begin{aligned} \texttt{Disk} &= \texttt{logical\_block\_number} \,\%\, \texttt{number\_of\_disks} \\ \texttt{Offset} &= \frac{\texttt{logical\_block\_number}}{\texttt{number\_of\_disks}} \end{aligned} \tag{4}$$

For example, given the following schema:



If it is requested, the logical block is 11, and the disks are 4:

$$\begin{aligned} \texttt{Disk} &= 11 \% 4 = 3 \\ \texttt{Offset} &= 11 \div 4 = 2.75 \approx 3, \text{ then physical block 2 starting from 0} \end{aligned}$$

Note that the **chunk size is critical** because it impacts disk array performance. With **smaller chunks**, there is **greater parallelism** than with **big chunks**, which have **reduced seek times**. The typical arrays use 64 KB chunks.

To measure RAID **performance**, we focus on sequential and random workloads. Assume disks in the array have sequential transfer rate S, and the info about the disk is:

- Average seek time: 7 ms
- Average rotational delay: 3 ms
- Transfer rate: 50 MB/s

For a single large transfer (10 MB):

$$S = \frac{\texttt{transfer\_size}}{\texttt{time\_to\_access}}$$

$$S = 10\ \text{MB} \div (7\ \text{ms} + 3\ \text{ms} + 10\ \text{MB} \div 50\ \text{MB/s}) = 47.62\ \text{MB/s}$$

If the disks in the array have a random transfer rate R, and for a set of small files (10 KB):

$$R = \frac{\texttt{transfer\_size}}{\texttt{time\_to\_access}}$$

$$R = 10\ \text{KB} \div (7\ \text{ms} + 3\ \text{ms} + 10\ \text{KB} \div 50\ \text{MB/s}) = 0.98\ \text{MB/s}$$

Analysis

- **Capacity**: N, where N is the number of disks. Then, everything can be filled with data.
- **Reliability**: non-existent. If any drive fails, data is permanently lost. Then, the Mean Time To Failure (MTTF) equals the **Mean Time To Data Loss (MTTDL)**:
$$\texttt{MTTF} = \texttt{MTTDL}$$
- **Sequential read and write**: $N \times S$
- **Random read and write**: $N \times R$

Where N is the number of disks, S the sequential transfer rate and R the random transfer rate.

RAID 1

Although RAID 0 offers high performance, it has zero error recovery. For this reason, RAID 1 makes **two copies of all data** (again, a minimum of 2 disk drives are required).

When is it used?

It is used when **zero error recovery is not allowed**.

Advantages

- **High reliability**. When a disk fails, the *second copy is used*.
- **Read of a data**. It can be *retrieved* from the *disk with shorter queueing, seek, and latency delays*.
- **Fast writes** (no error correcting code should be computed). But *still slower than standard disks* (due to duplication).

Disadvantages

- **High costs** because only 50% of the capacity is used.

How does it work?

In principle, a RAID 1 can mirror the content over more than one disk. This feature gives resiliency to errors even if more than one disk breaks. Also, it allows a **voting mechanism to identify errors not reported by the disk controller**. Unfortunately, this is **never used in practice because the overhead and costs are too high**.

However, disks could be coupled if several disks are available (always in an even number). Nevertheless, the total capacity is halved, and each disk has a mirror. Then, the question is simple: *How do we organize this architecture?* The answer is the nested RAIDs: `RAID 0 + 1` and `RAID 1 + 0`.

We define the `RAID x + y` (or `RAID xy`) as:

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

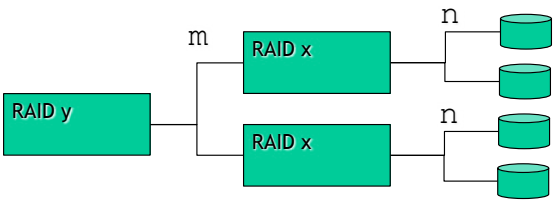
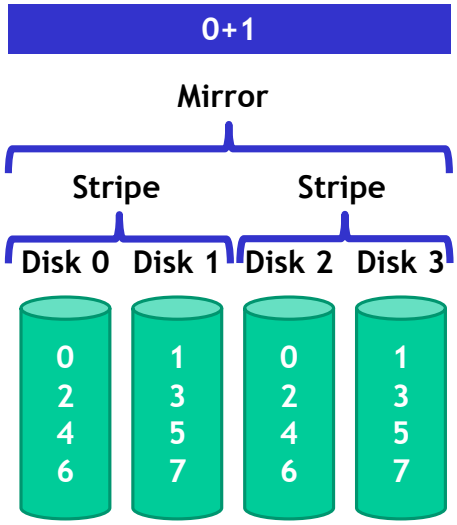


Figure 7: `RAID x + y` general architecture.

The `RAID 0 + 1` is a **group of striped disks (RAID 0) that are then mirrored (RAID 1)**. So:

1. The mirroring first (RAID 1)
2. Then the striping (RAID 0)

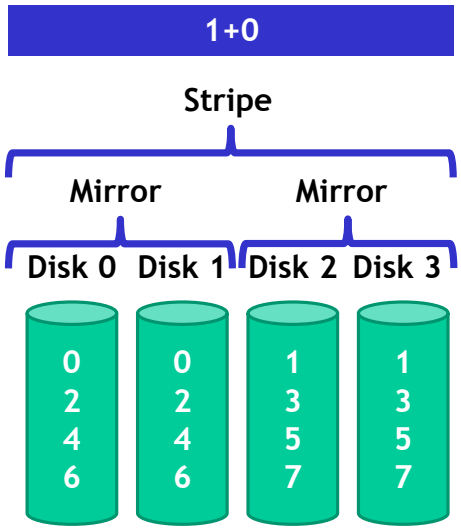
There are necessary almost four drives. Note that after the first failure, the model becomes a RAID 0.



The `RAID 1 + 0` is a **group of mirrored disks (RAID 1) that are then striped (RAID 0)**. So:

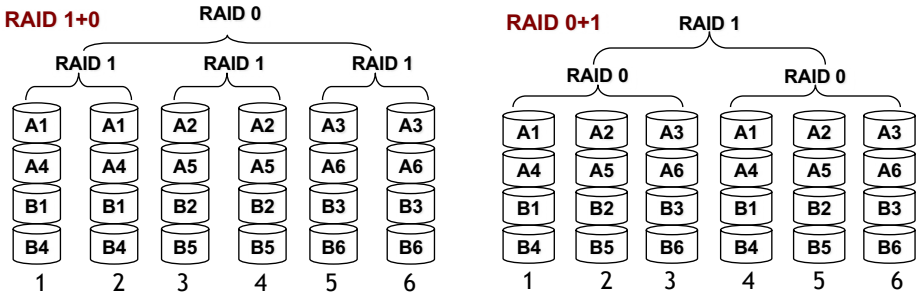
1. The striping first (RAID 0)
2. Then the mirroring (RAID 1)

There are necessary almost four drives. Usually, it is used in databases with very high workloads (fast writes).



≠ Differences between RAID 01 and RAID 10

Look at the following architectures:



What we can say is:

- The performance of RAID 01 and RAID 10 are the same.
- The **main difference is the fault tolerance**[6]!

On most implementations of RAID controllers, **RAID 01 fault tolerance is less**. Since we have only two groups of RAID 0, if two drives (one in each group) fail, the entire RAID 01 will fail. Looking at the architecture above, if Disk 1 and 4 fail, both groups will be down. So, the whole RAID 01 will fail.

On the contrary, RAID 10 since there are many groups (as the individual group is only two disks), even if three disks fail (one in each group), the RAID 10 is still functional. Looking at the architecture above, even if Disk 1, 3, and 5 fail, the RAID 10 will still be functional.

Fault Tolerance: `RAID 01` $\ggg$ `RAID 10`

[6]Fault tolerance is the ability of a system to maintain proper operation despite failures or faults in or more of its components.

So, given a choice between RAID 10 and RAID 01, it should be better to choose RAID 10 to have more fault tolerance.

Analysis

- **Capacity**: $N \div 2$, where N is the number of disks. Then, two copies of all data, thus half capacity.
- **Reliability**: 1 drive can fail, sometimes more! In an optimistic scenario, $N \div 2$ drives can fail without data loss.
- **Sequential write**: $(N \div 2) \times S$; two copies of all data, thus half throughput.
- **Sequential read**: $(N \div 2) \times S$; half of the read blocks are wasted, thus halving throughput.
- **Random read**: $N \times R$; it is the best scenario for RAID 1 because the read can be parallelized across all disks.
- **Random write**: $(N \div 2) \times R$; two copies of all data, thus half throughput.

Where N is the number of disks, S is the sequential transfer rate, and R is the random transfer rate.

It is essential to observe RAID 1. There is a notorious **problem** called the **Consistent Update Problem**.

Mirrored writes should be atomic. Then, all copies are written, or none are written. Unfortunately, this is very **difficult to guarantee** sometimes, for example, in a power failure scenario. To **solve** this problem, many RAID controllers include a **write-ahead log**, a **battery backend**, and **non-volatile storage of pending writes**. With this system, a **recovery procedure ensures the recovery of the out-of-sync mirrored copies**.

RAID 4

RAID 4 consists of **block-level striping with a dedicated parity disk**.

When is it used?

RAID 1 offers highly reliable data storage, but it uses half the space of the array capacity. To achieve the **same level of reliability without wasting capacity**, it is possible to use RAID 4, which uses **information coding techniques to build lightweight error recovery mechanisms**.

Advantages

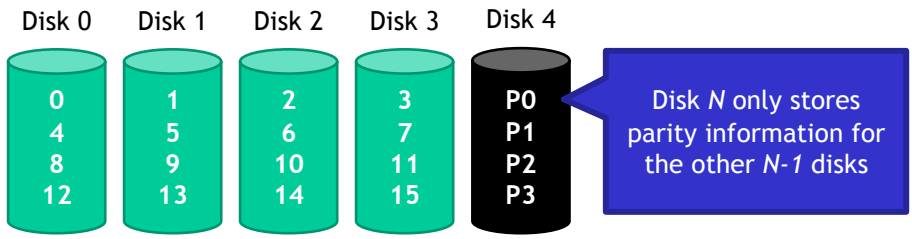
- **Good performance** of random reads.

Disadvantages

- **Random Write performance is terrible** due to being *bottlenecked* by the parity drive.

How does it work?

Disk N only stores parity information for the other $N-1$ disks. The parity is calculated using the `XOR` operation[7].



Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	0	1	1	0 ^ 0 ^ 1 ^ 1 = 0
0	1	0	0	0 ^ 1 ^ 0 ^ 0 = 1
1	1	1	1	1 ^ 1 ^ 1 ^ 1 = 0
0	1	1	1	0 ^ 1 ^ 1 ^ 1 = 1

Parity calculated using XOR

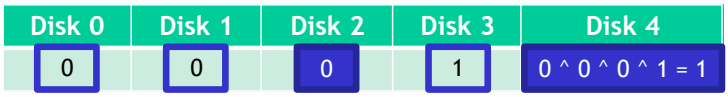
Figure 8: RAID 4 - *How does it work?*

The **serial** or **random read** is not a problem in RAID 4 because there is a **parallelization across all non-parity blocks** in the stripe despite a tiny performance reduction due to the parity disk.

[7] "A or B, but not A and B"

During the parity writing, the blocks are updated. The **random write** performance is affected by the approach used:

- **Additive parity** (as known as reconstruct-writes):
  1. Read all other blocks in a stripe in parallel;
  2. `XOR` those with the new block to form a new parity block;
  3. Write the new data block and new parity block to disks.



- **Subtractive parity** (as known as read-modify-writes):
  1. Read only the old data block to be updated and the old parity block;
  2. Compute the new parity block using:

$$P_{new} = (D_{new} \veebar D_{old}) \veebar P_{old}$$

     Where $\veebar$ is the logical `XOR`.
  3. Write the new data block and new parity block to disks.

Despite the **sequential write** does **not suffer any performance effect** from the parity disk. Because it uses full-stripe write:

1. Buffer all data blocks of a stripe
2. Compute the parity block
3. Write all data and parity blocks in parallel [8]

Analysis

- **Capacity**: $N-1$, where N is the number of disks, and the -1 is because one is dedicated to the parity disk.
- **Reliability**: 1 drive can fail. Massive performance degradation during partial outage.
- **Sequential read/write**: $(N-1)\times S$; parallelization across all non-parity blocks in the stripe (parity disk has no effect).
- **Random read**: $(N-1) \times R$; reads parallelize over all but the parity drive (parity disk has no effect).
- **Random write**: $R \div 2$; writes serialize due to the parity drive, and each write requires one read and one write of the parity drive.

Where N is the number of disks, S is the sequential transfer rate, and R is the random transfer rate.

RAID 5

RAID 5 **rotates parity blocks across stripes**.

When is it used?

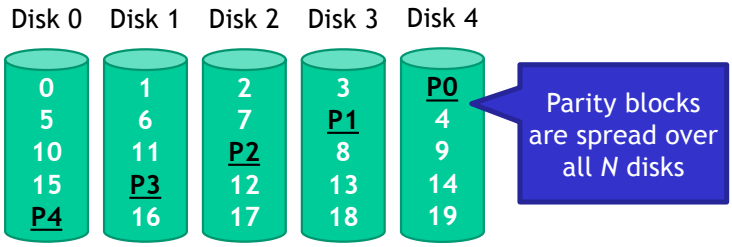
Unlike in RAID 4, parity information is distributed among the drives. This technique is **used to improve significantly the random write throughput** against the RAID 4 system.

Advantages

- Improved random write despite the RAID 4 system.

How does it work?

The writes are spread roughly evenly across all drives.



Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	0	1	1	0 ^ 0 ^ 1 ^ 1 = 0
1	0	0	0 ^ 1 ^ 0 ^ 0 = 1	0
1	1	1 ^ 1 ^ 1 ^ 1 = 0	1	1
1	0 ^ 1 ^ 1 ^ 1 = 1	0	1	1

The **random write** in RAID 5 is:

1. Read the target block and the parity block
2. Use subtraction to calculate the new parity block
3. Write the target block and the parity block

Thus, **four total operations** (two reads, two writes) **are distributed across all drives**.

Analysis

- **Capacity**: $N - 1$ (**same as RAID 4**), where N is the number of disks.
- **Reliability**: 1 drive can fail (**same as RAID 4**). Massive performance degradation during partial outage.

- **Sequential read/write**: (N - 1) * S (**same as RAID 4**); parallelization across all non-parity blocks in the stripe (parity disk has no effect).
- **Random read**: $N \times R$; unlike RAID 4, **reads parallelize over all drives**.
- **Random write**: $(N \div 4) \times R$; unlike RAID 4, **writes parallelize over all drives**, and each write requires two reads and two writes, hence $N \div 4$.

Where N is the number of disks, S is the sequential transfer rate, and R is the random transfer rate.

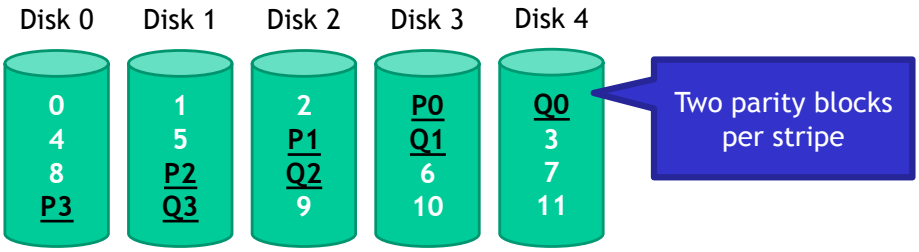
RAID 6

RAID 6 can tolerate multiple disk faults (**more fault tolerance**) concerning RAID 5. It tolerates two concurrent failures.

How does it work?

It uses Solomon-Reeds codes with two redundancy schemes and requires $N + 2$ disks (where N is the number of disks). Note that the **minimum set is 4 data disks**.

Unfortunately, it has a **high overhead for writes** (computation of parities) because each write requires six disk accesses due to the need to update both the P and Q parity blocks (slow writes).



Comparison and characteristics of RAID levels

The following table shows eight fundamental properties of the RAID levels.

- N: number of drives
- R: random access speed
- S: sequential access speed
- D: latency to access a single disk

	`RAID 0`	`RAID 1`	`RAID 4`	`RAID 5`
Capacity	N	$N \div 2$	$N - 1$	$N - 1$
Reliability	0	$1 \vee : N \div 2$	1	1
Sequential Read	$N \times S$	$(N \div 2) \times S$	$(N - 1) \times S$	$(N - 1) \times S$
Sequential Write	$N \times S$	$(N \div 2) \times S$	$(N - 1) \times S$	$(N - 1) \times S$
Random Read	$N \times R$	$N \times R$	$(N - 1) \times R$	$N \times R$
Random Write	$N \times R$	$(N \div 2) \times R$	$R \div 2$	$(N \div 4) \times R$
Read	D	D	D	D
Write	D	D	$2 \times D$	$2 \times D$

Table 4: Comparison of RAID levels.

Where the throughput is:

- Sequential Read
- Sequential Write
- Random Read
- Random Write

And the latency is:

- Read
- Write

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
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

Figure 9: Characteristics of RAID levels.

RAID 0 has the best performance and most capacity.

RAID 1 (10 better than 01) or **RAID 6** have the greatest error recovery.

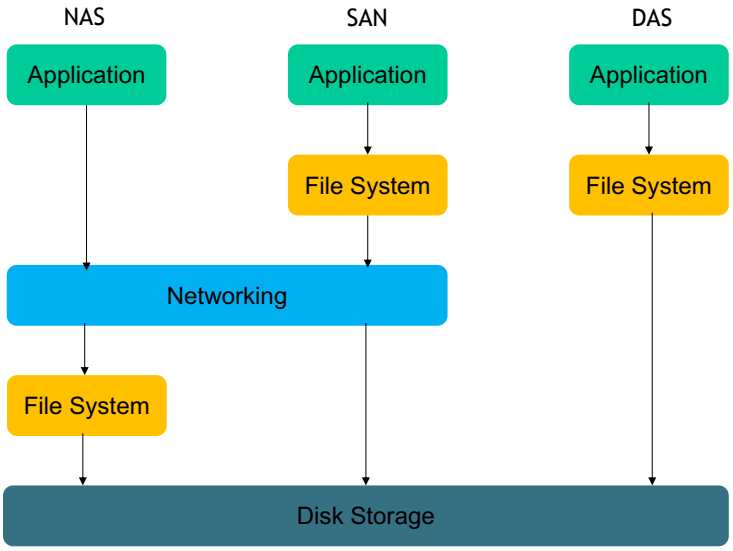
RAID 5 has the better ***balance*** between space, performance, and recoverability.

2.2.2.5 DAS, NAS and SAN

As the last argument, we introduce **three different typologies** of storage systems:

- **Direct Attached Storage (DAS)** is a **storage system directly attached to a server or workstation**. They are visible as disks/volumes by the client OS.
- **Network Attached Storage (NAS)** is a **computer connected to a network that provides only file-based data storage services** (e.g. FTP, Network File System) to other devices on the network and is visible as File Server to the client OS.
- **Storage Area Networks (SAN)** are **remote storage units connected to a PC using a specific networking technology** (e.g. Fiber Channel) and are visible as disks/volumes by the client OS.

In the following schema, we can see a simple architectural comparison.



✔ **DAS features**

DAS is a **storage system directly attached to a server or workstation**. The term is used to differentiate non-networked storage from SAN and NAS. The **main features** are:

- Limited scalability.
- Complex manageability.
- Limited performance.
- To read files in other machines (the "file sharing" protocol of the OS must be used).

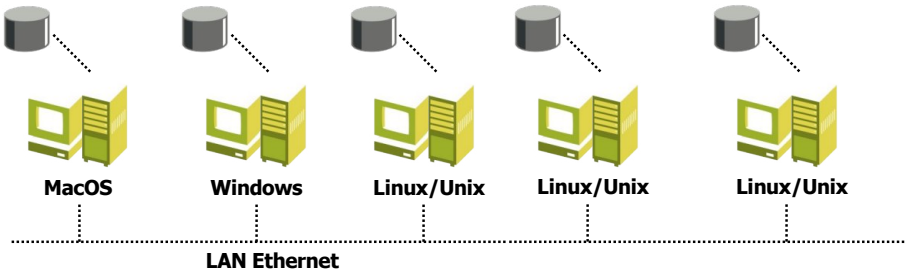


Figure 10: DAS architecture.

Note that all the **external disks connected to a PC with a point-to-point protocol can be considered DAS**.

✔ NAS features

A NAS unit is a **computer connected to a network that provides only file-based data storage services to other devices on the network**. NAS systems contain one or more hard disks, often organized into logical redundant storage containers or RAID. Finally, **NAS provides file-access services to the hosts connected to a TCP/IP network through Networked File Systems/SAMBA**. Each NAS element has its IP address. Furthermore, each NAS has good scalability.

The **main differences between DAS and NAS** are:

- 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**.

Regarding **performance**, NAS depends mainly on the speed and congestion of the network.

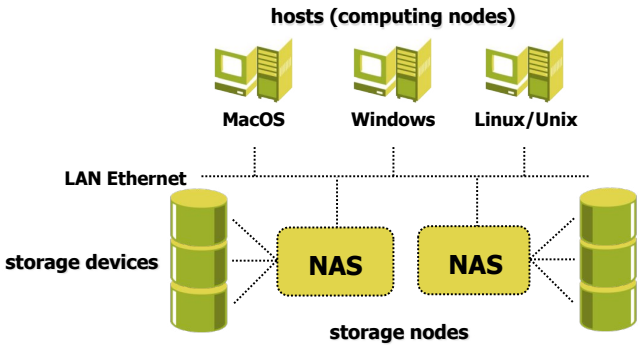


Figure 11: NAS architecture.

✔ SAN features

SANs are remote **storage units connected to servers using a specific networking technology**. SANs have a particular network dedicated to accessing storage devices. It has two distinct networks: one TCP/IP and another dedicated network (e.g. Fiber Channel). It has a high scalability.

The **main difference between a NAS and a SAN** is that:

- **NAS appears to the client OS as a file server**. Then, the client can map network drives to shares on that server.
- **A disk available through a SAN still appears to the client OS as a disk**. It will be visible in the disks and volumes management utilities (along with the client's disks) and available to be formatted with a file system.

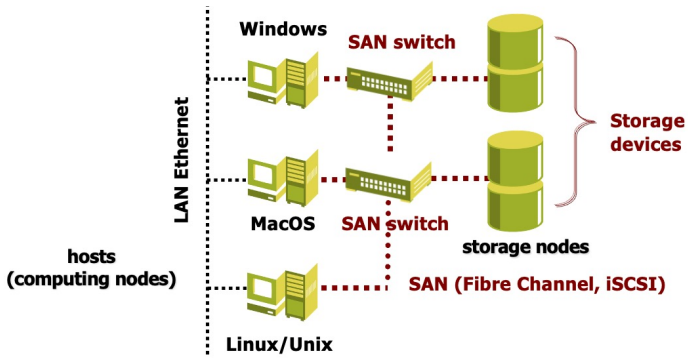


Figure 12: SAN architecture.

	Application Domain	Advantages	Disadvantages
DAS	• Budget constraints • Simple storage solutions	• Easy setup • Low cost • High performance	• Limited accessibility • Limited scalability • No central management and backup
NAS	• File storage and sharing • Big Data	• Scalability • Greater accessibility • Performance	• Increased LAN traffic • Performance limitations • Security and reliability
SAN	• DBMS • Virtualized environments	• Improved performance • Greater scalability • Improved availability	• Costs • Complex setup and maintenance

Figure 13: DAS vs. NAS vs. SAN

2.2.3 Networking (architecture and technology)

2.2.3.1 Fundamental concepts

In the data centre, servers' *performance increases* over time, and the demand for inter-server *bandwidth also increases*.

A **solution** can be to double the aggregate compute capacity or the aggregate storage simply by **doubling the number of compute or storage elements**.

The **doubling leaf bandwidth** is used since the networking has no straightforward horizontal scaling solution. Then, with **twice as many servers**, we will have **twice as many network ports and thus twice as much bandwidth**.

What is a bisection bandwidth?

Bisection bandwidth is a **measure of network performance**, **defined as the bandwidth available between two equal-sized partitions when a network is bisected**. This measure accounts for the *bottleneck bandwidth of the entire network*, providing a representation of the actual bandwidth available in the system. The bisection should be done to minimize the bandwidth between the two partitions. It is often used to evaluate and compare networks for parallel architectures, including point-to-point communication systems or on-chip micro-networks.

Assuming that every server needs to talk to every other server, we need to double not just leaf bandwidth but bisection bandwidth.

How to design a data centre network?

There are many design principles to follow:

- **Very scalable** in order to support a vast number of servers;
- **Minimum cost** in terms of basic building blocks (e.g. switches);
- **Modular** to reuse simple basic modules;
- **Reliable** and **resilient**;
- It may exploit novel/proprietary technologies and protocols incompatible with legacy Internet.

The **Data Center Network (DCN)** connects a data centre's computing and storage units to achieve optimum performance. It can be **classified** into **three main categories**:

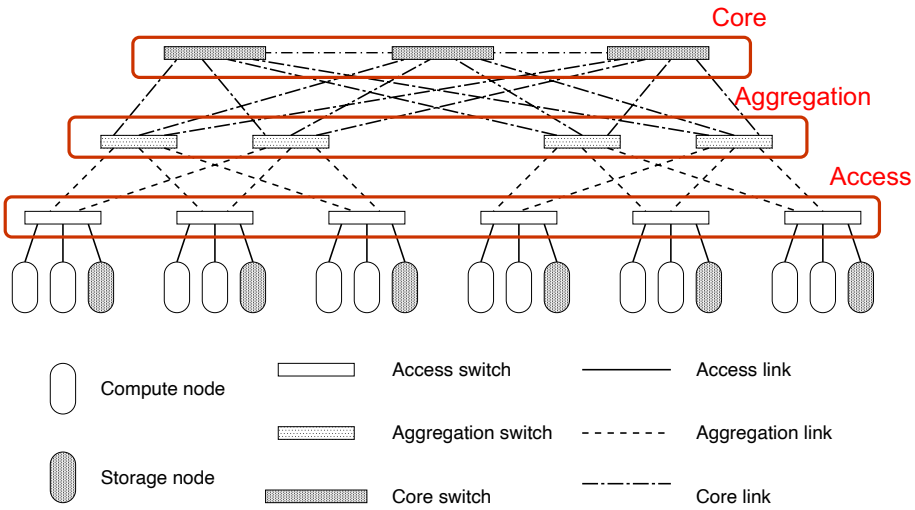
- **DCN Switch-centric architectures**. DCN uses **switches to perform packet forwarding**.[8]
- **DCN Server-centric architectures**. DCN **uses servers with multiple Network Interface Cards (NICs)**[9] **to act as switches** and perform other computational functions.
- **DCN Hybrid architectures**. DCN **combines switches and servers** for packet forwarding.

[8] **Packet forwarding** is the passing of packets from one network segment to another by nodes on a computer network.

[9] A **Network Interface Cards (NICs)** is a computer hardware component that connects a computer to a computer network.

2.2.3.2 Switch-centric: classical Three-Tier architecture

The **Three-Tier architecture**, also called **Three Layer architecture**, **configures the network in three different layers**:



It is a simple Data Center Network topology.

- The **servers** are **connected to the DCN through access switches**.
- **Each access-level switch is connected to at least two aggregation-level switches**.
- **Aggregation-level switches are connected to core-level switches** (gateways).

✔ **Advantages**

1. **Bandwidth can be increased** by **increasing the switches at the core and aggregation layers**, and by using routing protocols such as Equal Cost Multiple Path (ECMP) that equally shares the traffic among different routes.
2. **Very simple** solution.

👎 **Cons**

1. **Very expensive** in large data centers because the upper layers require faster network equipments.
2. **Cost very high** in term of acquisition and energy consumption.

In the **access layer**, there are two possible architectures:

- **ToR (Top-of-Rack) architecture**. All servers in a rack are connected to a ToR access switch within the same rack. The aggregation switches are in dedicated racks or shared racks with other ToR switches and servers.

✔ **Advantages**

1. **Simpler cabling** because the number of cables is limited.
2. **Lower costs** because the number of ports per switch is limited.

👎 **Cons**

Higher complexity for switch management.

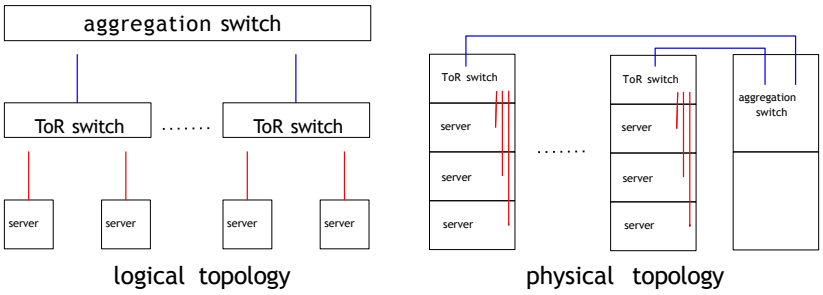


Figure 14: ToR (Top-of-Rack) architecture.

- **EoR (End-of-Row) architecture**. Aggregation switcher are positioned one per corridor, at the end of a line of rack. Servers in a racks are connected directly to the aggregation switch in another rack. Exists a patch panel to connect the servers to the aggregation switch.

✔ **Advantages**

Simpler switch management.

👎 **Cons**

The aggregation switches must have a larger number of ports, then:

1. **Complex cabling**.
2. **Longer cables** then **higher costs**.

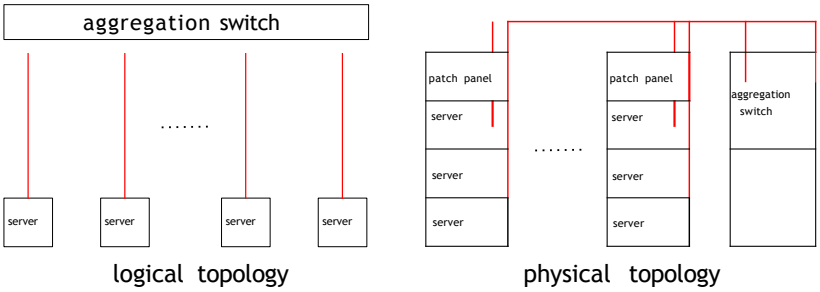


Figure 15: EoR (End-of-Row) architecture.

2.2.3.3 Switch-centric: Leaf-Spine architectures

In the following section we present two Leaf-Spine architectures: the Leaf-Spine model and the Pod-based model (Fat Tree Network).

The **Leaf-Spine architecture** consists of **two levels of interconnection**:

1. The **leaf** (which is a ToR switch);
2. The **spine** (which has dedicated switches, aggregation switches).

In practice, servers have two interfaces connected to two ToR switches to provide fault tolerance.

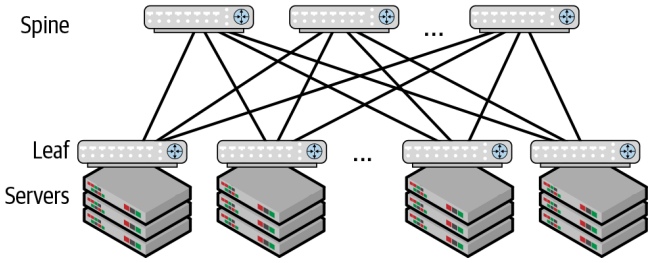


Figure 16: Leaf-Spine architecture.

Now we will explain the Leaf-Spine architecture. If m is the number of **mid-stage switches** and n is the **number of inputs and outputs**, the Leaf-Spine topology is as follows:

- Each switch module is bi-directional.
  - *Leaf* has $2k$ bidirectional ports per module;
  - *Spine* has k bidirectional ports per module.
- Each path traverses either 1 or 3 modules.

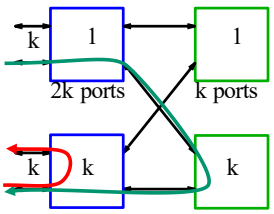


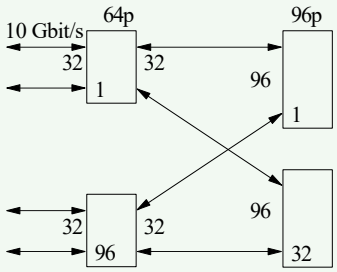
Figure 17: Explanation of Leaf-Spine architecture.

The **advantages** are: use of homogeneous equipment, simple routing, the number of hops is the same for any pair of nodes, small blast radius.

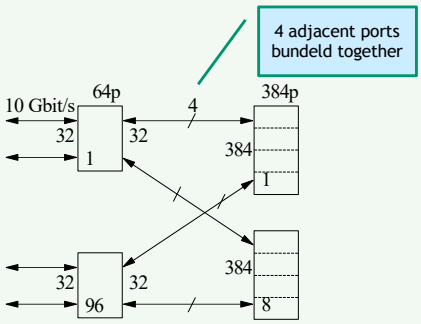
Example 10

There are 3072 servers and 3072 ports available at 10 Gbit/s. Provides a leaf-spine design.
There are **two possible designs**.

1. The first consists of 96 switches with 64 ports and 32 switches with 96 ports.



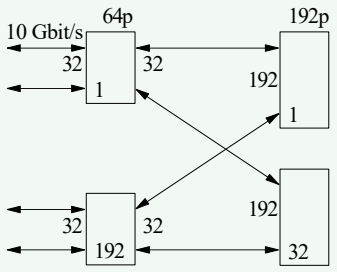
2. The second has only 8 switches but they have more ports: 384 ($8 \times 384 = 3072$).



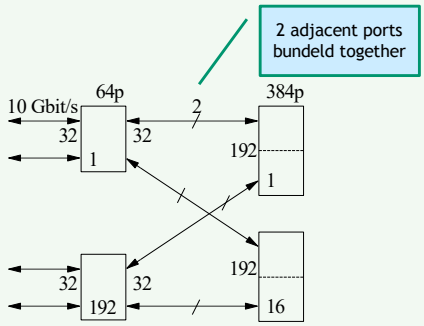
Example 11

There are 6144 servers and 6144 ports available at 10 Gbit/s. Provides a leaf-spine design.
There are **two possible designs**.

1. The first consists of 192 switches with 64 ports and 32 switches with 192 ports.



2. The second has only 16 switches but they have more ports: 384 ($16 \times 384 = 6144$).



The **Pod-based model**, also called **Fat Tree Network**, is another network architecture used to **increase the scaling** feature respecting the leaf-spine.

It transforms each group of spine-leaf into a **PoD (Point of Delivery)**[10] and adds a super spine layer.

It is a **highly scalable** and **cost-effective** DCN architecture designed to **maximise bisection bandwidth**. It can be built using standard Gigabit Ethernet switches with the same number of ports.

It is composed by a ***leaf*** of $2k^2$ bidirectional ports:

- k^2 ports to the servers;
- k^2 ports to the data center network.

In general, let k^2P servers: there are $2kP$ switches with $2k$ ports and k^2 switches with P ports. Using the Fat-Tree model, **the P value is 2k, so for $2k^3$ servers, there are $5k^2$ switches with 2k ports** ($k^2 + 2k \cdot 2k$).

At the **edge layer**, there are $2k$ pods (groups of servers), each with k^2 servers.

- Each edge switch is directly connected to k servers in a pod and k aggregation switches.
- A Fat-Tree network with $2k$-port commodity switches can accomodate $2k^3$ servers in total.
- k^2 core switches with $2k$-port each, each one connected to $2k$ pods.
- Each aggregation switch is connected to k core switches.

[10] A Point Of Delivery is a module or group of network, compute, storage and application components that work together to deliver a network service. The PoD is a repeatable pattern and its components increase the modularity, scalability and manageability of data.

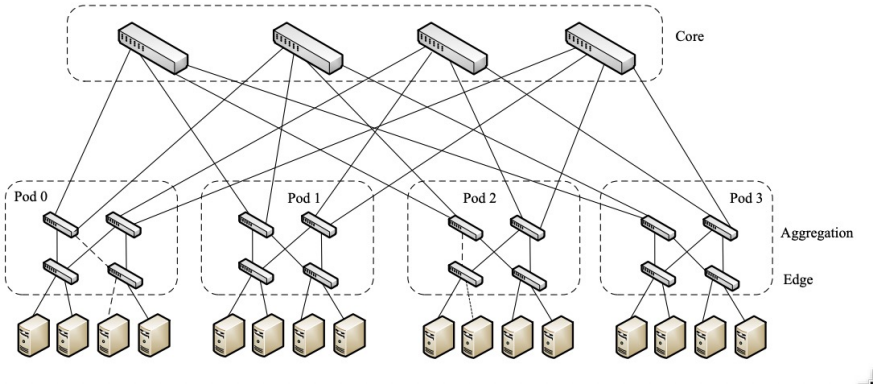


Figure 18: Fat-Tree Network, with $k = 2$, 4 pods, 16 servers, 20 switches.

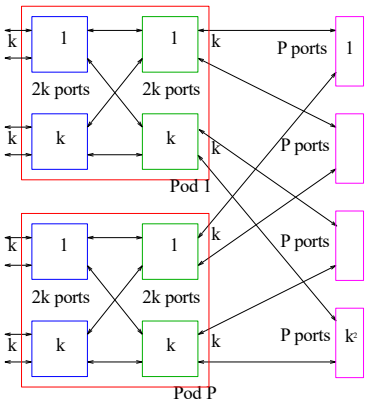


Figure 19: Explanation of Fat-Tree Network.

2.2.3.4 Server-centric and hybrid architectures

CamCube is a **server-centric architecture** typically proposed for building container-sized data centres.

Advantages

It can **reduce implementation and maintenance costs** by using only servers to build the Data Center Network. It also exploits network locality to **increase communication efficiency**.

Disadvantages

It **requires servers with Multiple Network Interface cards** to build a 3D tours network, **long paths** and **high routing complexity**.

The hybrid architectures are **DCell**, **BCube** and **MDCube**.

A **DCell** is a **scalable and cost-effective hybrid architecture** that uses switches and servers for packet forwarding. It is a recursive architecture and uses a basic building block called $DCell_0$ to construct larger DCells.

$DCell_k$ ($k > 0$) denotes a level-k DCell **constructed by combining** $n + 1$ **servers** in $DCell_0$. A $DCell_0$ has n ($n < 8$) **servers directly connected by a commodity switch**.

Disadvantages: **long communication paths**, many required Network Interface Cards and **increased cabling costs**.

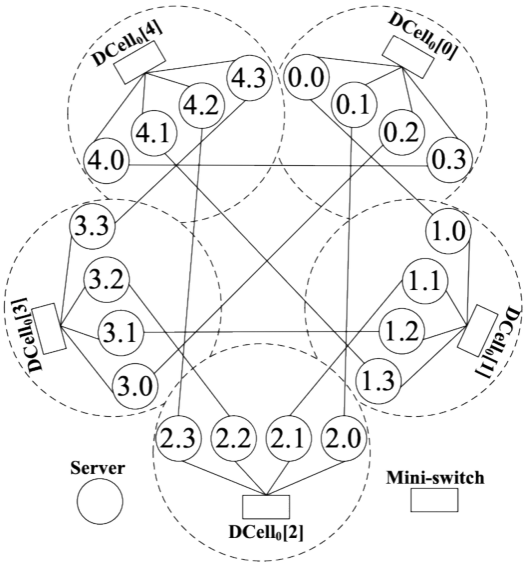


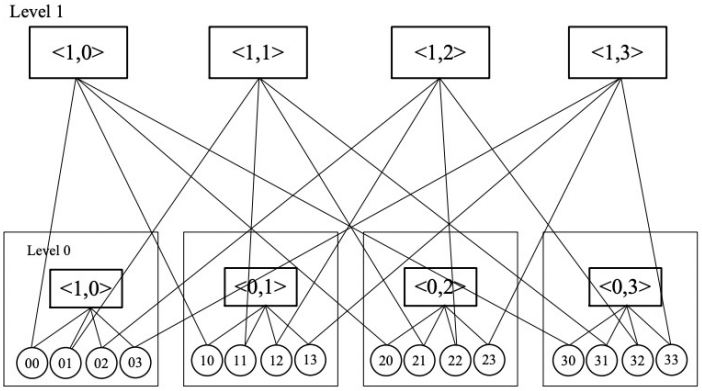
Figure 20: DCell hybrid architecture.

BCube is a **hybrid and cost-effective architecture** that scales through recursion. It provides **high bisection bandwidth** and **graceful throughput degradation**.

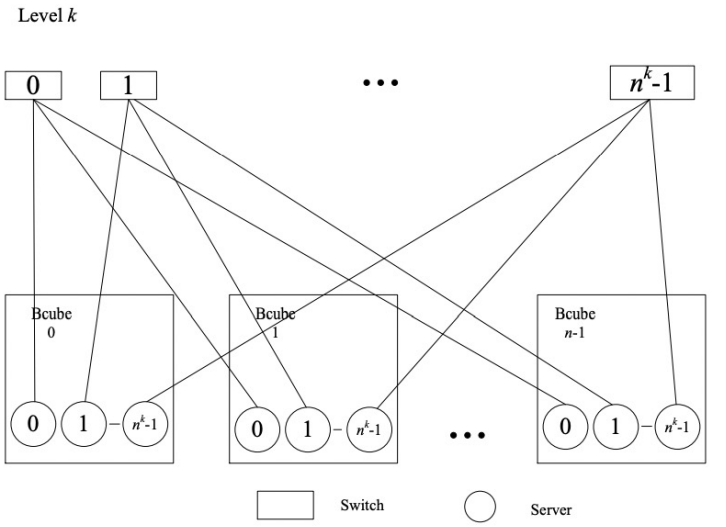
It uses BCube as a building block, consisting of n servers connected to an n-port switch.

A $BCube_k$ $(k > 0)$ is constructed with n $BCube_{k-1}s$ and n^k n-port switches. In a $BCube_k$ there are $n^{(k+1)}$ $k+1$-port servers and $k+1$ layers of switches.

Disadvantages: **limited scalability** and **high cabling costs** (NICs reason).



(a) Request phase.



(b) Grant phase.

Figure 21: BCube hybrid architecture.

MDCube is designed to **reduce the number of cables used to connect containers**.

- Each container has an ID which is mapped to a multidimensional tuple.
- Each container is connected to a neighbouring container with a different tuple in one dimension.
- There are two types of connections: Intra-container links and high-speed inter-container links.

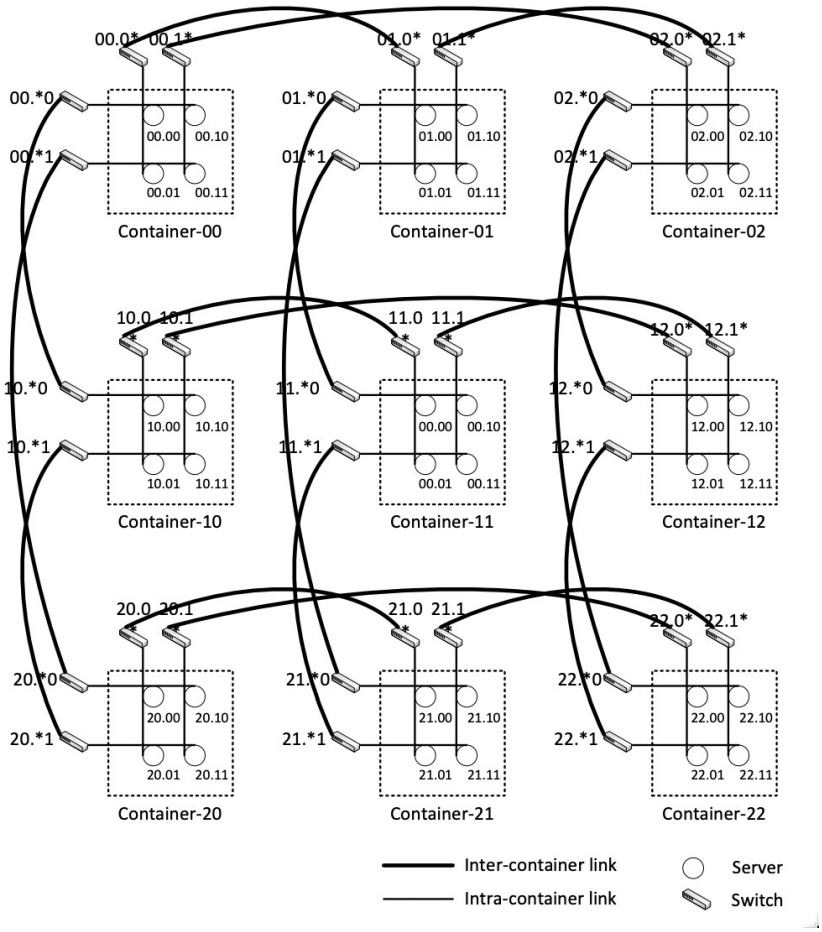


Figure 22: MDCube hybrid architecture.