Data Infrastructure

Lecture 2: Introduction & Storage

Federica Bazzocchi
13 And 17/04/2026

Organization:

- Frontal lessons
- Active participation from you
- Interactive session on an example of data management in material science: OFED and its services
- Seminar to discuss of an example of FAIR data management

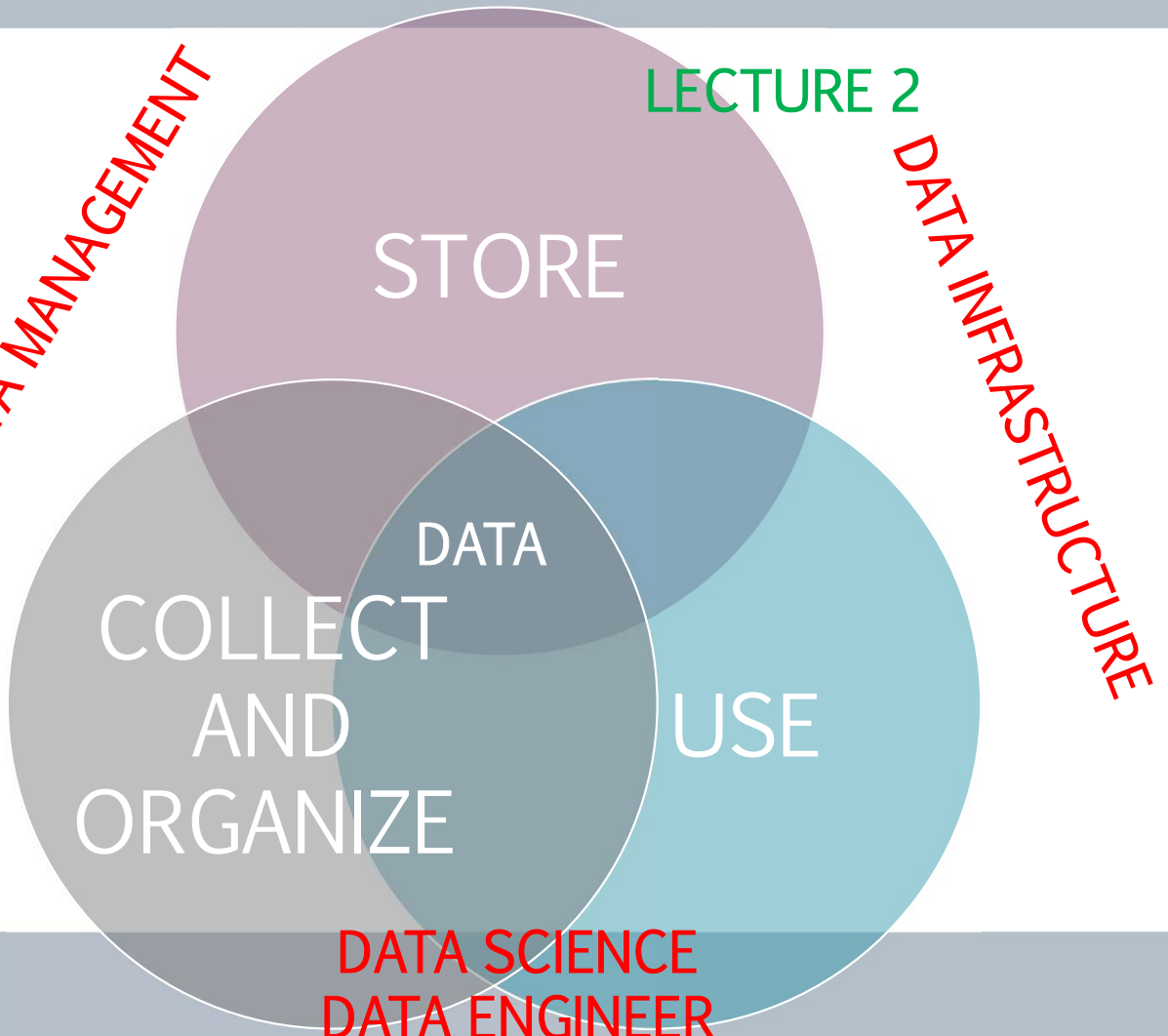
On 8 of May, 11:00-13:00, in presence

LECTURE 1

RESEARCH DATA MANAGEMENT

LECTURE 2

DATA INFRASTRUCTURE



DATA SCIENCE
DATA ENGINEER
AI ENGINEER

π

LECTURE 2 OUTLINE:

- Introduction to data infrastructure
- Hardware component
- Storage

INTRODUCTION TO DATA INFRASTRUCTURE

WHAT IS DATA INFRASTRUCTURE?

- The set of technologies and processes for collecting, storing, processing, and managing data
- Includes hardware, software, and policies to handle structured and unstructured data
- Aims to provide reliable, scalable, and secure data management solutions
- A well-structured data infrastructure enables efficient data management, security, and innovation

THE RISE OF DATA INFRASTRUCTURE

1960s-1970s: Emergence of databases (e.g., IBM's IMS, relational databases by E.F. Codd)

1980s-1990s: Data Warehousing and ETL processes

2000s: Big Data explosion and cloud computing

2010s-present: Modern data infrastructure with scalable architectures and AI integration

DATA INFRASTRUCTURE

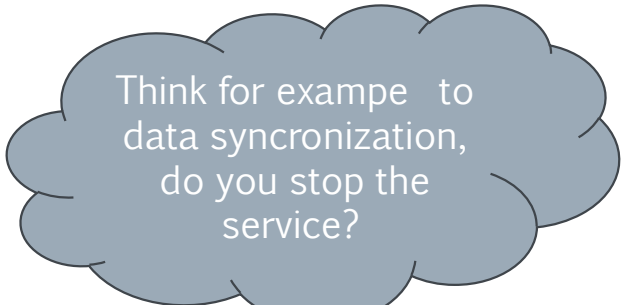
- Store large datasets and large data rates from experiments/business activities
- Allow reliability by replicating data sources
- Allow accessibility by copying source to several places
- Monitor and check resource usage
- Provide a set of integrated services which are compatible between domains
- Increase interoperability through common standard schemes
- Guarantee secure, broadband, remote access to data

WHY DATA INFRASTRUCTURE IS RELEVANT

- Data preservation to allow long-term availability of data
- Ensures data quality, security, and compliance
- High quality of data and metadata to enable advanced and cross-disciplinary access and enrichment operations
- Economic justification: as the scientific community is operating on increasingly larger datasets and want to preserve the information concerned, the infrastructure provided should have a clear roadmap of technology exchange and backwards compatibility.
- Provide the infrastructure to allow fine-grained access control
- Facilitates scalability and efficiency in modern enterprises
- Supports data-driven decision-making
- Enables real-time analytics and business intelligence

CRITERIA TO SCORE DATA INFRASTRUCTURE

- Scalability: horizontal scaling, adding nodes to distribute the load more evenly (vertical scaling single node's resources are increased) flexibility and avoid bottleneck (single point of failure)
- Reliability: measure of a system to perform without failure (redundancy and fault tolerance) replication data and services across multiple nodes. By replicating the system can withstand failure, if one node fails other can take over its tasks. Implementing robust error handling, monitoring and automated recovery process enhance reliability
- Availability: refers to proportion of time a system is operational and accessible to users. Load balancing, failover mechanism, data replications contribute to high availability



Together are essential to create robust distributed systems for modern applications and services

KEY COMPONENTS OF DATA INFRASTRUCTURE

- Data Storage: Databases, Data Lakes, Cloud Storage
- Data Processing: ETL Pipelines, Batch & Streaming Processing
- Data Governance/Policy: Security, Compliance, Data Quality
- Data Access & Analytics Services: Dashboards, AI/ML Models, BI Tools

DATA INFRASTRUCTURE DESIGN

DATA INFRASTRUCTURE ARCHITECTURE

- High-Level Architectures: define the global structure of the system, strategy for organizing the entire system
- Architectural Models: define how components interact and are deployed, define interaction and distribution roles
- Logical Models: describe internal organization of components, independent of physical deployment, organize responsibilities and logic flow

HIGH-LEVEL ARCHITECTURE

Monolithic Architecture

- All components of the system are tightly integrated into a single unit
- Centralized deployment, usually a single executable or application
- Easier to develop initially, but difficult to scale or updates or change single parts (full redeploy may affect all the system)
- Examples: Traditional desktop software, early web applications
- Pros: Simple to develop and deploy
- Cons: Hard to scale, maintain, or update parts independently

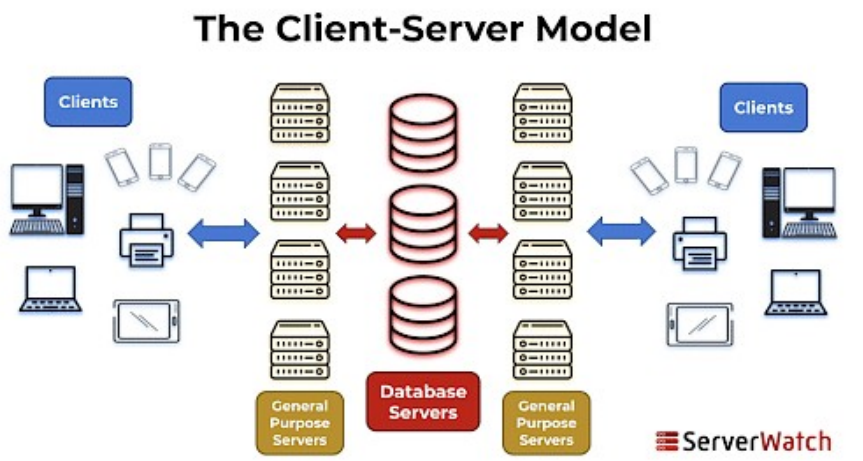
Distributed Architecture

- System components are spread across multiple physical or virtual machines
- Components communicate via network protocols (e.g., REST, gRPC)
- Designed for scalability, fault tolerance, and flexibility
- Examples: Cloud-native apps, microservices systems
- Pros: Scalable, resilient, flexible
- Cons: Increased complexity, requires orchestration and monitoring

ARCHITECTURAL MODELS

Client-Server Architecture

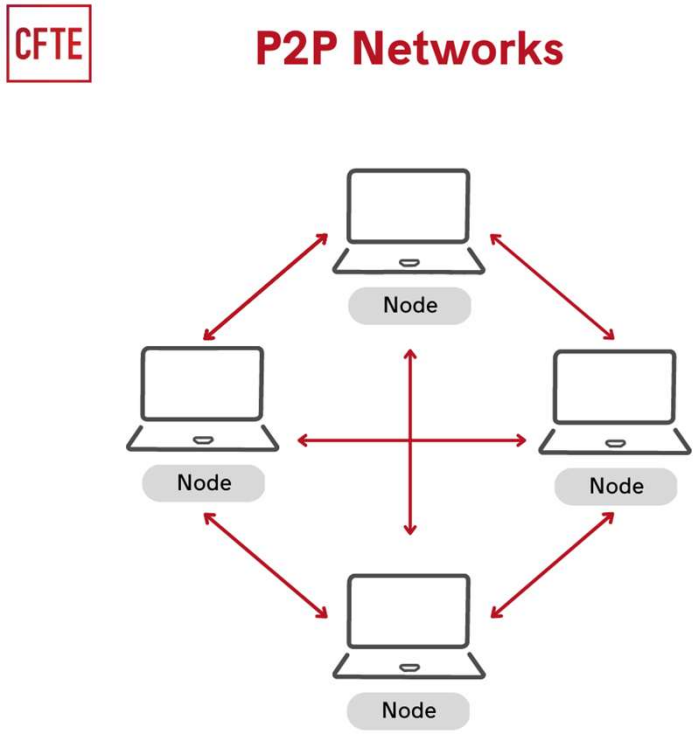
- Centralized architecture with clients inititate requests for services or sources (anything from web browser to mobile apps to desktop applications)
- Server: central componets that provide resources or services to the clients. Server powerful machine or sw system designed to handle multiple requests simultaneously (resources managed databases, fiels, applications)
- Easy to implement and manage (centralized at server level)
- Examples: Web servers, Database servers
- Pros: Central control, easier maintenance
- Cons: Single point of failure (bottleneck at server levels), scalability limitations



ARCHITECTURAL MODELS

Peer-to-Peer (P2P) Architecture

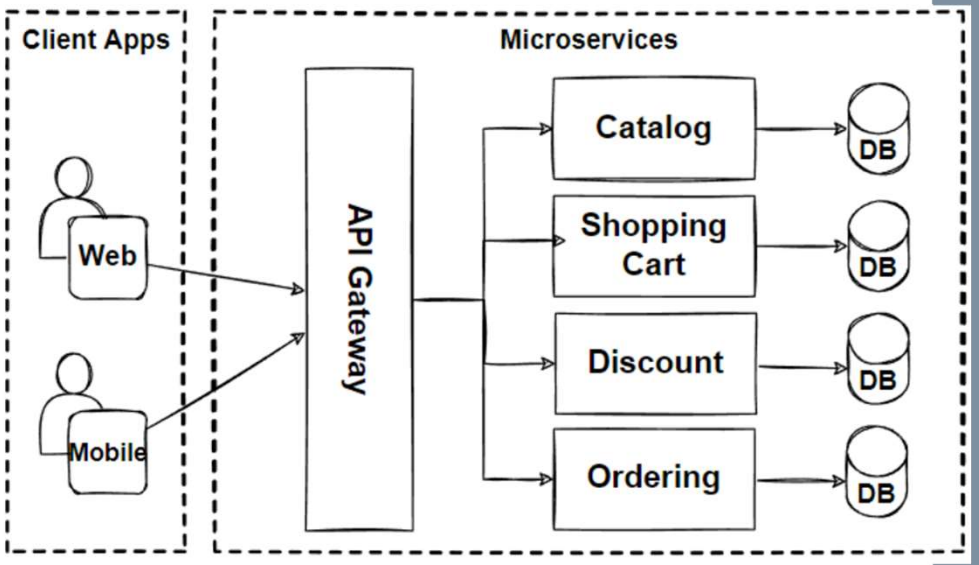
- Decentralized model where each node acts as both client and server
- Nodes equal authority, share resources and communicate directly
- Each peer can send request, provide data, share computational power.
- Examples: BitTorrent, Blockchain, Skype
- Pros: High resilience, scalability, fault-tolerant (no single point of failure) decentralization makes it well suited for app that requires scalability abd resilience
- Cons: Complex coordination, managing security challenging (required reputation system or encryption)



ARCHITECTURAL MODELS

Microservices Architecture

- SW development approach where a large app is broken into small, independent services
- Each service handles a specific function and communicates over APIs, independent modules (independent dev and deployment, acceleration)
- Examples: Netflix, Amazon
- Pros: Modular, scalable, independently development and deploy, fault isolation, reliability and resilience, fexibility
- Cons: Operational complexity, requires orchestration tools (e.g., Kubernetes), data consistency and integration issues



ARCHITECTURAL MODELS

Comparison Overview

- Client-Server:
  - Centralized
  - Easier to manage
  - Single point of failure

- Peer-to-Peer:
  - Fully decentralized
  - Fault-tolerant
  - Complex coordination

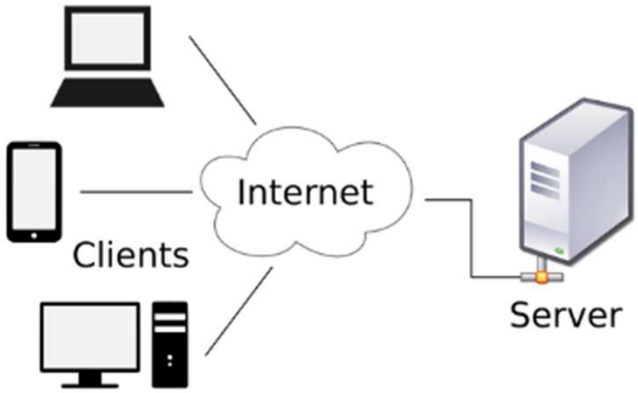
- Microservices:
  - Modular, loosely coupled
  - Scalable and flexible
  - Requires DevOps and orchestration

ARCHITECTURAL MODELS

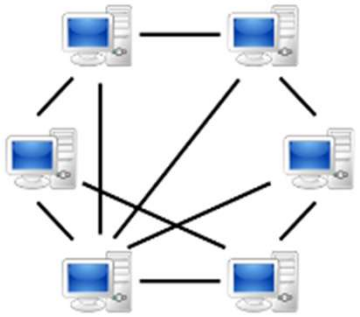
Comparison Overview

Feature	Client-Server	Peer-to-Peer (P2P)	Microservices
Hierarchy	Strict: Centralized Server vs. Passive Clients.	Flat: All nodes (Peers) are equal.	Functional: Decentralized services with specialized roles.
Data Control	Centralized: Single source of truth managed by the server.	Distributed: Data is spread across many nodes.	Decentralized: Each service owns its specific database (Polyglot Persistence).
Scalability	Vertical: Upgrading the central server (CPU/RAM).	Organic: Capacity increases as more peers join.	Horizontal: Scaling specific services based on demand.
Failure Impact	Total: Server down = System down.	Minimal: Network is resilient to node departures.	Partial: One service failure doesn't necessarily crash the system.
Complexity	Low: Easy to manage and secure.	High: Hard to ensure data consistency/consensus.	Medium-High: Requires complex orchestration (Kubernetes).
Typical Use Case	Web Hosting, Traditional SQL DBs.	BitTorrent, Blockchain, IPFS.	Netflix, Amazon, Modern Enterprise Apps.

COMPARING ARCHITECTURE AND DESIGN PRINCIPLES



Client-Server



Peer-to-Peer

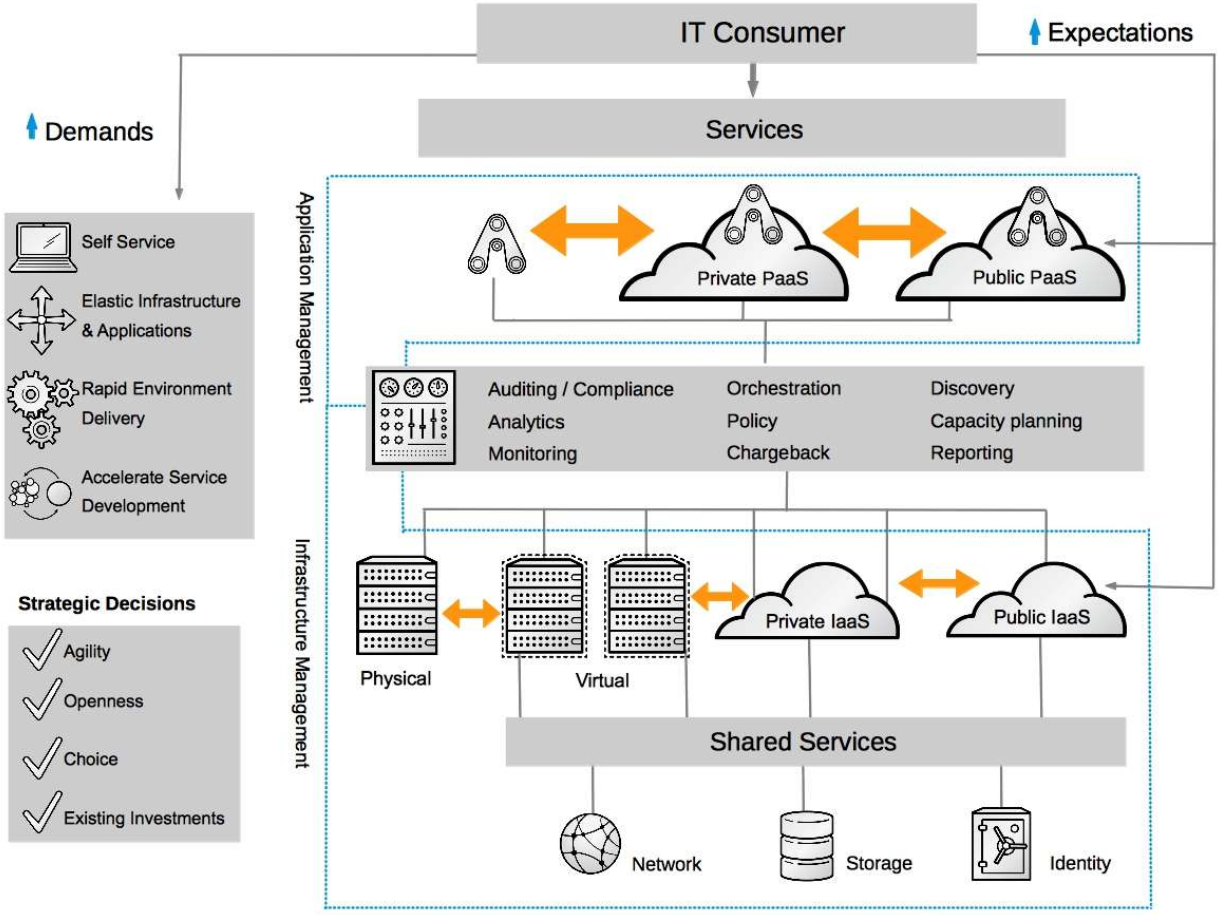
ARCHITECTURAL MODELS

Hybrid Infrastructures: Combining advantages of **Client-Server** with the **resiliency of P2P**.

- **Leader-Follower (Master-Slave) Clusters:** it looks like Client-Server to the user, the internal nodes act as **Peers** that use P2P protocols. If the "Leader" (Server) fails, the "Followers" (Peers) hold an election to promote a new leader.
  - *Examples:* Redis Sentinel, MongoDB Replica Sets.
- **Content Delivery Networks (CDNs):** a central server (Client-Server) originates the data, but a global network of "Edge" nodes (P2P-like distribution) delivers it to users. This reduces latency by moving data physically closer to the requester.
  - *Examples:* Cloudflare, Akamai.
- **Distributed Hash Tables (DHT) in Metadata Management:** use a Client-Server interface for queries but manage the actual file locations across thousands of drives using DHT (a P2P technology). This prevents a single metadata server from becoming a bottleneck.
  - *Examples:* Amazon S3's internal architecture, Apache Cassandra.
- **Service Mesh in Microservices:** they communicate in a P2P fashion (Service-to-Service) rather than going back to a central "Hub." A "Sidecar" proxy manages this communication, creating a peer-like network of services that is managed by a centralized control plane.
  - *Examples:* Istio, Linkerd.

ARCHITECTURAL MODELS

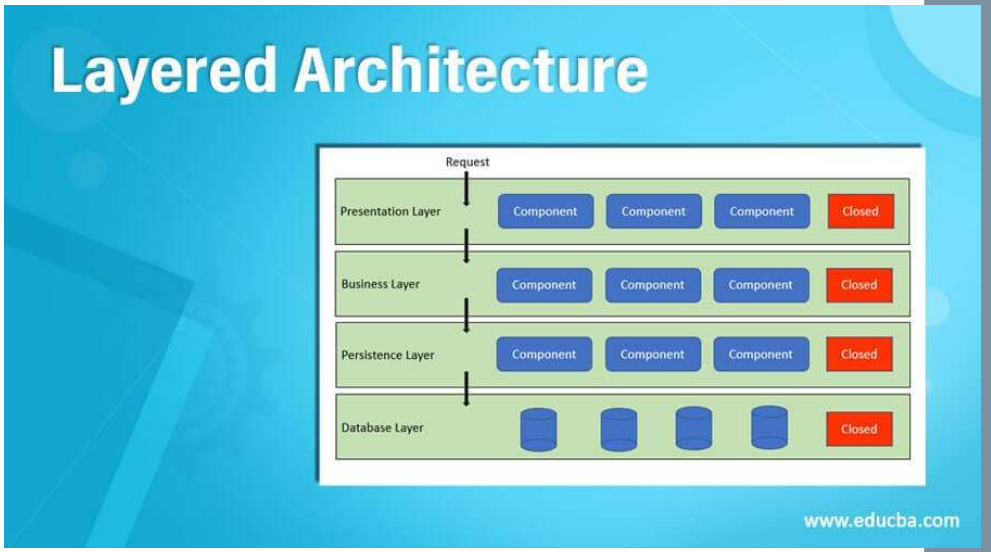
Hybrid Infrastructures: Combining advantages of **Client-Server** with the **resiliency of P2P.**



LOGICAL MODEL ARCHITECTURES

Layered Model

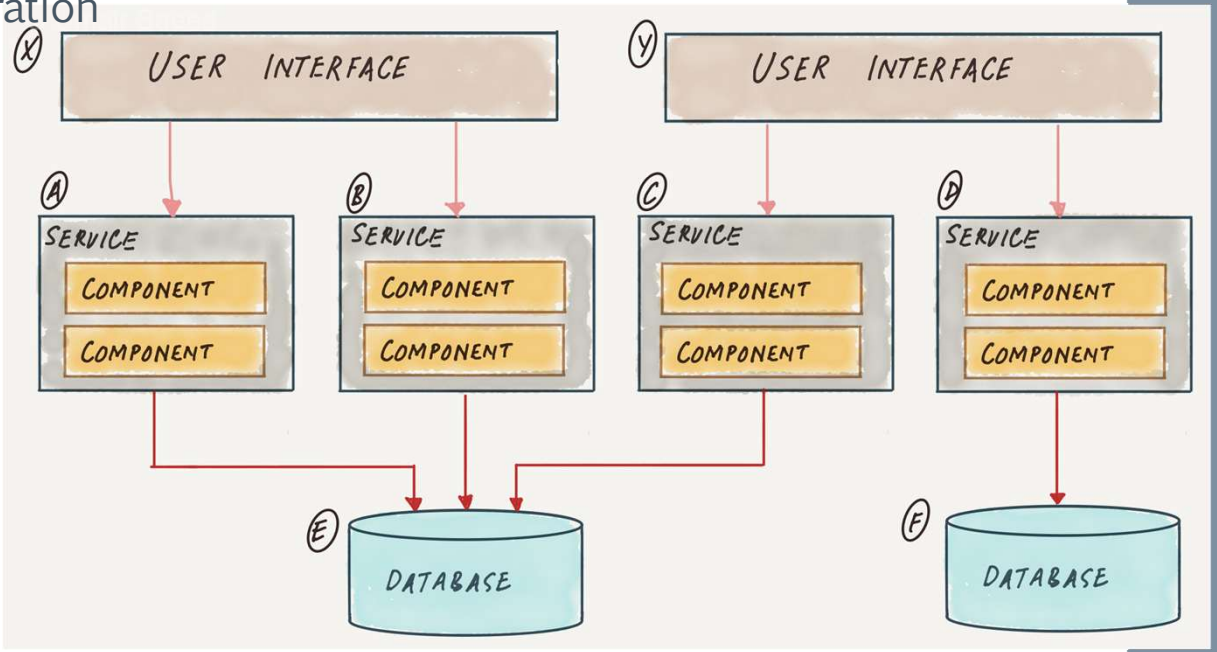
- System is organized into logical layers (e.g., presentation, business, data)
- Each layer has a specific responsibility and interfaces with adjacent layers
- Promotes separation of concerns and modularity
- Examples: MVC (Model-View-Controller), n-tier applications
- Pros: Easier to maintain and test
- Cons: Can become rigid and slow due to strict layer boundaries



LOGICAL MODEL ARCHITECTURES

Service-Oriented Architecture (SOA)

- System is built as a collection of services that communicate over a network
- Each service is self-contained and performs a specific business function
- Uses standard protocols (e.g., SOAP, REST)
- Encourages reusability and integration
  - Pros: Modular, reusable, interoperable
  - Cons: Overhead in service communication, governance complexity



COMBINING ARCHITECTURES/MODELS

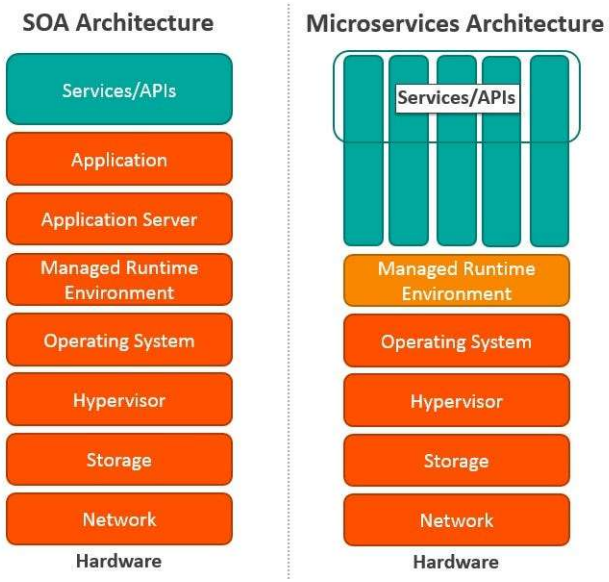
Architectural Model	Only Distributed ?	Typical of Distributed System
Client-Server	NO	YES
P2P	YES	YES
Microservices	NO, BUT ALMOST ALWAYS	YES

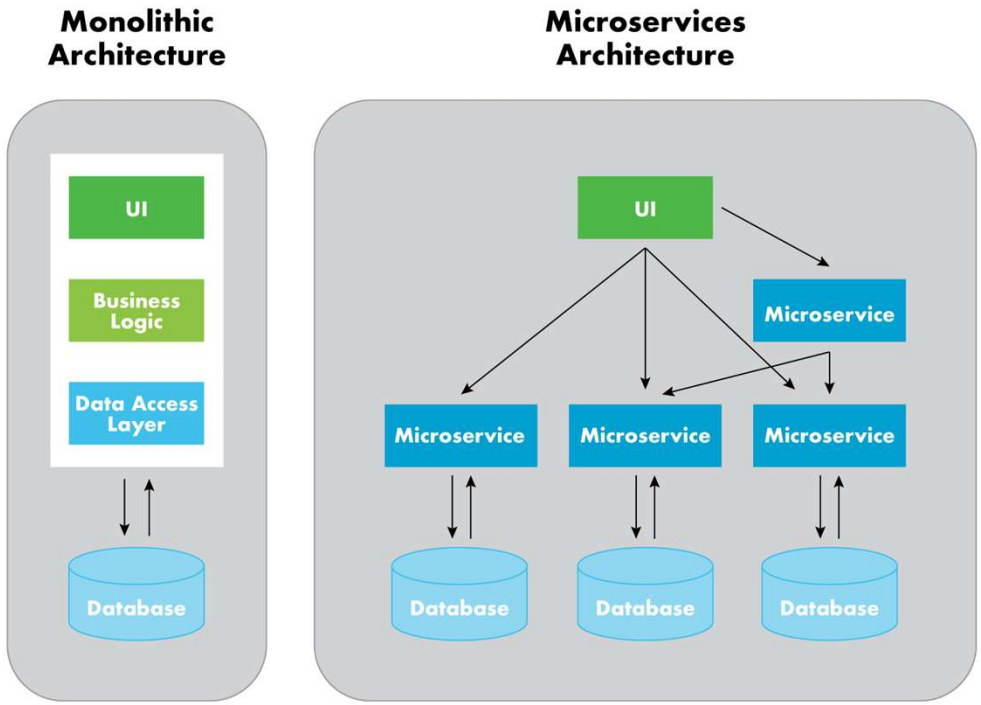
Architectural Model	Only Distributed ?	Notes
Layered	NO	Can be implemented on a single system (e.g.web app)
SOA	YES	Requires communication between decoupled services

COMBINING ARCHITECTURES/MODELS

Architectural Model	Compatible with Layered ?	Compatible with SOA ?
Client-Server	YES	YES
P2P	NOT TYPICAL	LIMITED
Microservices	CAN IMPLEMENT LAYERS	YES (microservices are the evolution of SOA)

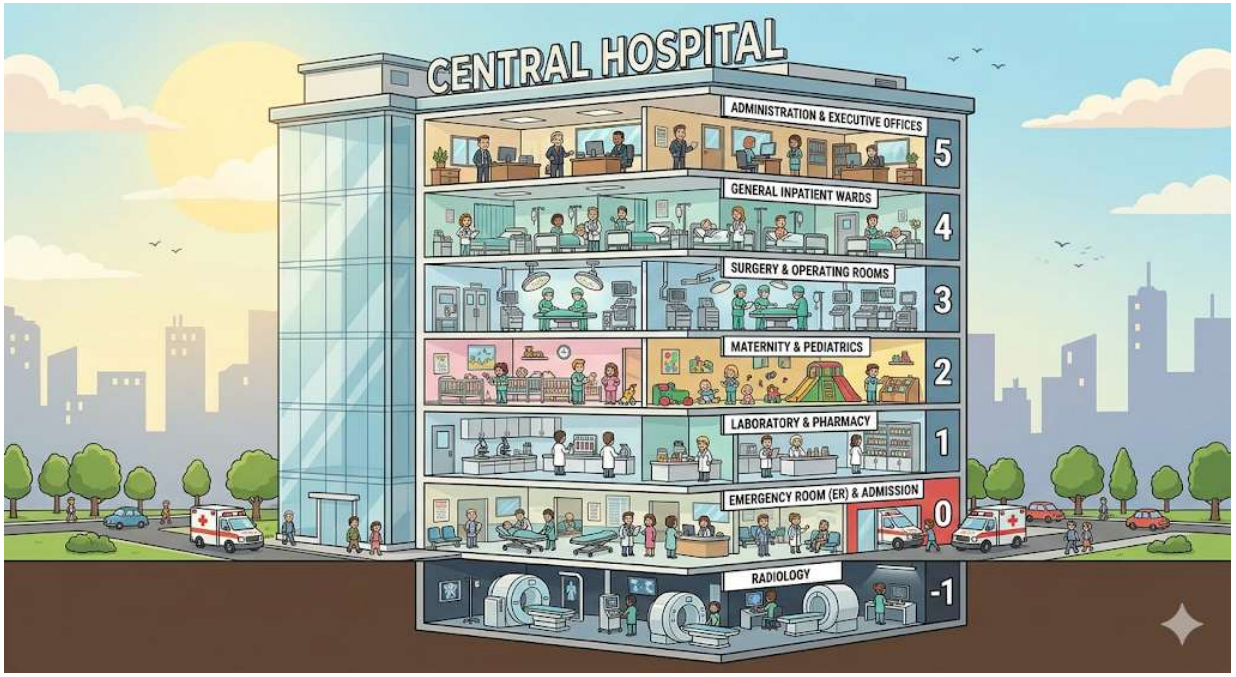
COMPARING ARCHITECTURE AND DESIGN PRINCIPLES





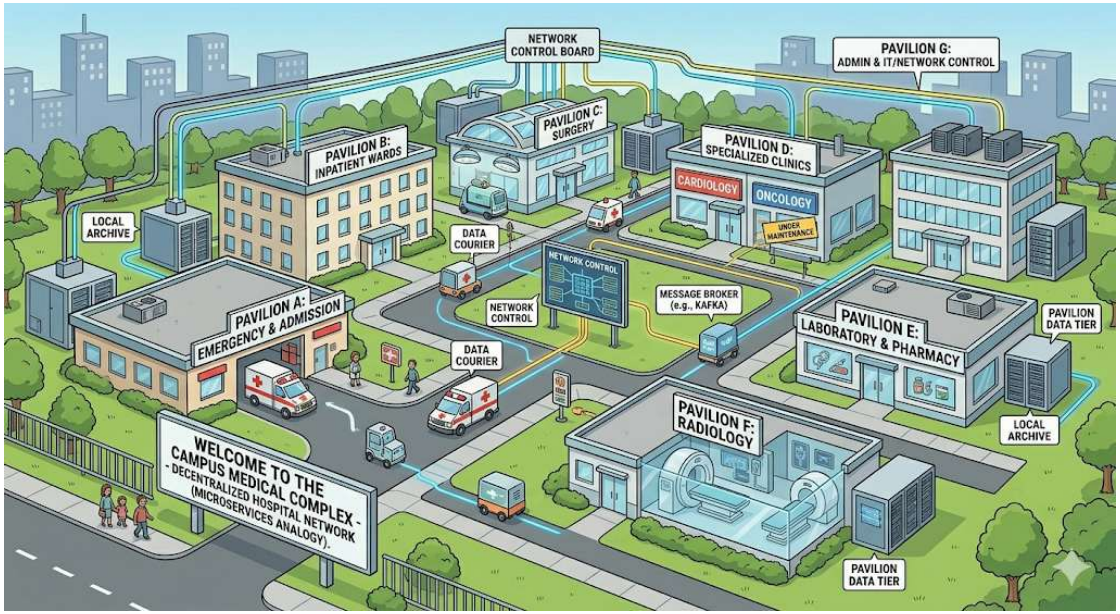
AN ANALOGY

- Hospital in a single building ->Monolitich high level architecture
- Different Departments/Units in each floor-> Logical model layers that may communicate only among first near
- Patient List/ Information -> if shared we have a single storage (client-server), if not shared we have a peer-to-peer/microservices approach (synchronization problem!)



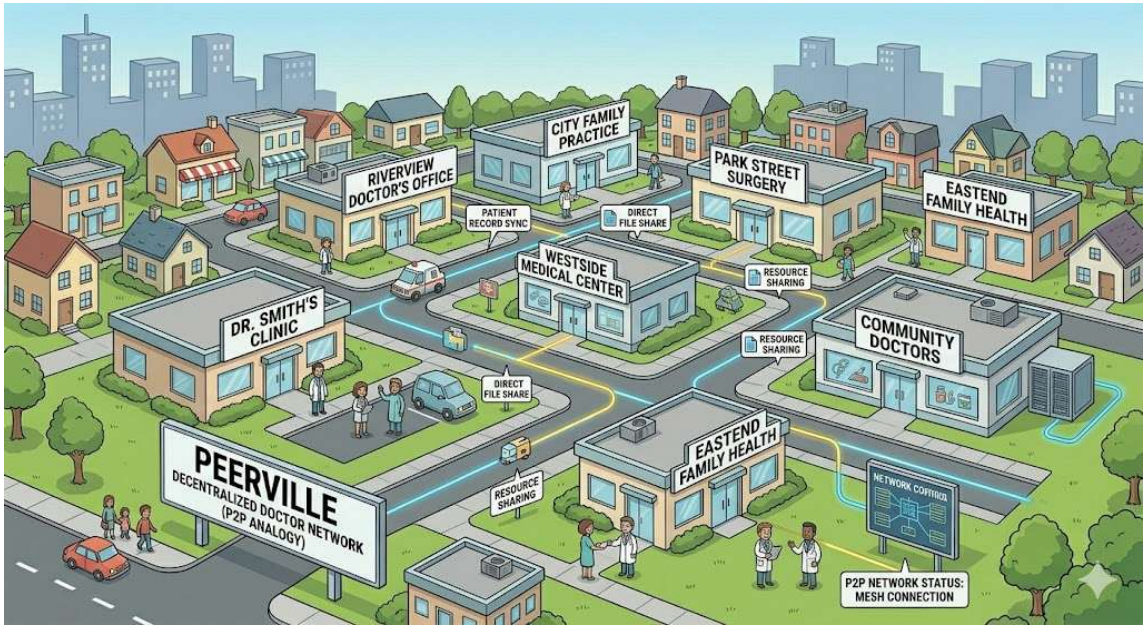
AN ANALOGY

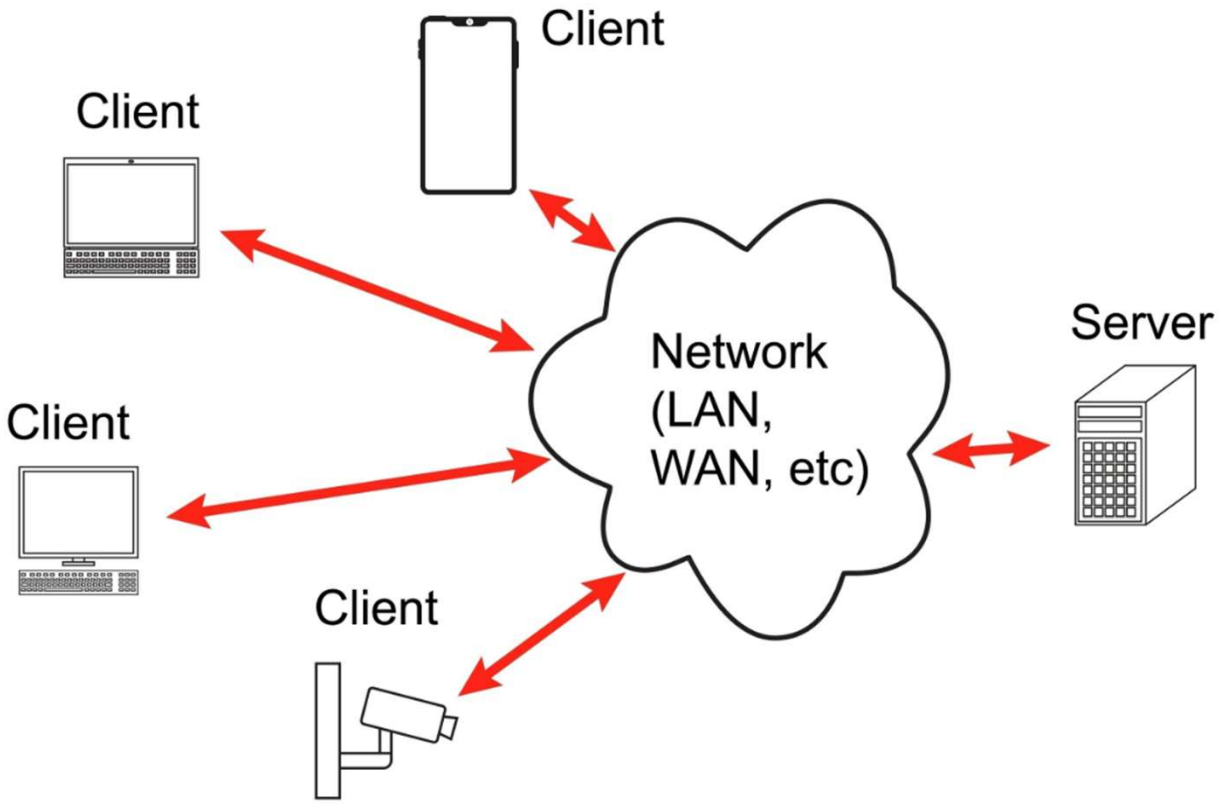
- Hospital in blocks -> Distributed high level architecture
- Different Departments/Units per block-> if they may communicate only following a specific order corresponds to a layer logical model, if they communicate as a network they corresponds to a service oriented architecture
- Patient List/ Information -> if shared we have a single storage (client-server), if not shared we have a peer-to-peer/microservices approach (synchronization problem!)



AN ANALOGY- DOCTORVILLE

- Atown of General Practitioners-> Peer to Peer
- Each doctor has the same kit-> they do the same kind of analysis/examinations
- Who are the patients?-> the doctors themselves! They go to the other “peers”





Client–server model (architecture)

CLIENT-SERVER EXAMPLE

A Retail Banking System

In a bank, we need control and central authority.

The bank system is Single Source of Truth for all account balances.

- The Server: database cluster.
- The Clients: Hundreds of branch terminals, thousands of ATMs, and millions of mobile banking apps.

When somebody uses an ATM (client) it asks to the Central Server for customer's balance and then allow the withdraw. The central server validates the request and updates the master record. Strict hierarchy, central control.

PEER-TO-PEER EXAMPLE

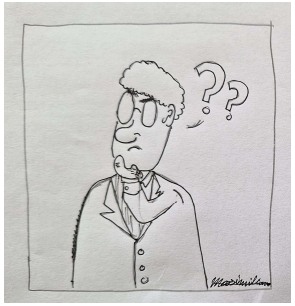
BitTorrent File Sharing

In file sharing, resilience and decentralized distribution are crucial. All nodes are equal and simultaneously act as content providers and content consumers.

- **The Peers:** user computers globally.
- **When using it:** A user joins the "swarm" and downloads pieces of a large file from multiple other peers. At the same moment, they begin uploading the pieces they have already received to other peers who need them. All nodes operate on the same level, sharing and consuming resources directly, without a single central file server. High availability, decentralized content.

I used to be a bit-torrent user…so was I one of the peers?

I never knew! When did I give my agreement?



PEER-TO-PEER EXAMPLE

The "Hidden" P2P Social Contract

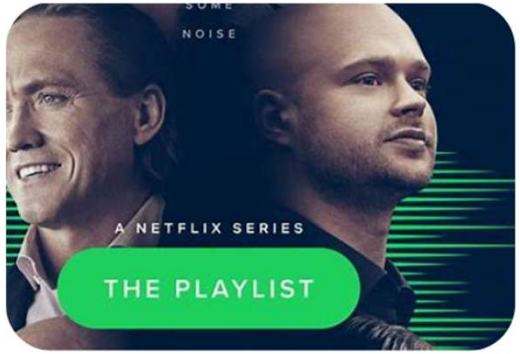
- **The Agreement**: I accepted the role of becoming a **node** in their network in exchange for free files or significantly higher download speeds.
- **Implicit Consent**: By downloading and running the software, I granted my laptop permission to "listen" for requests from other peers and transmit data back to them.
- **The P2P "Contract"**: A reciprocal exchange of resources— *"I consume bandwidth from you, and in return, you consume bandwidth from me."*

GOOD TO KNOW !!! (maybe is always better reading the licence agreement!)

PEER-TO-PEER EXAMPLE

The "Classic" Spotify (2008–2014): Hybrid
When Spotify first launched, they faced a massive infrastructure challenge: streaming high-quality audio to millions of users without bankrupting themselves on server costs.

- They used a Hybrid P2P/Client-Server model.
- How it worked: When somebody played a song, Spotify's "Client" software would first check if that song (or chunks of it) existed on a nearby user's computer. If it did, his/her computer would "suck" the data from his/her neighbor instead of the central Spotify server.
- In numbers: About 80% of the music delivered to users was served via P2P. This massively reduced Spotify's bandwidth bills and made the service very snappy (great scalable business model!)



PEER-TO-PEER EXAMPLE

The Pivot (2014–Present): Pure Client-Server
In 2014, Spotify officially retired their P2P backend.

- Mobile Dominance: P2P is not efficient for smartphones. It drains the battery (because the phone has to stay "active" to serve others) and eats up expensive mobile data plans (think at 2014!)

- Server Efficiency: Cloud providers became so efficient and cheap that the complexity of managing a P2P mesh was no longer worth the savings.
- Content Control: In a Client-Server model, it is much easier to manage digital rights (DRM) and ensure every stream is perfectly tracked for royalties.

as infrastructure shifted from desktop PCs (with unlimited wired internet) to mobile phones (with limited battery and data), the Client-Server model regained its role via Content Delivery Networks (CDNs).

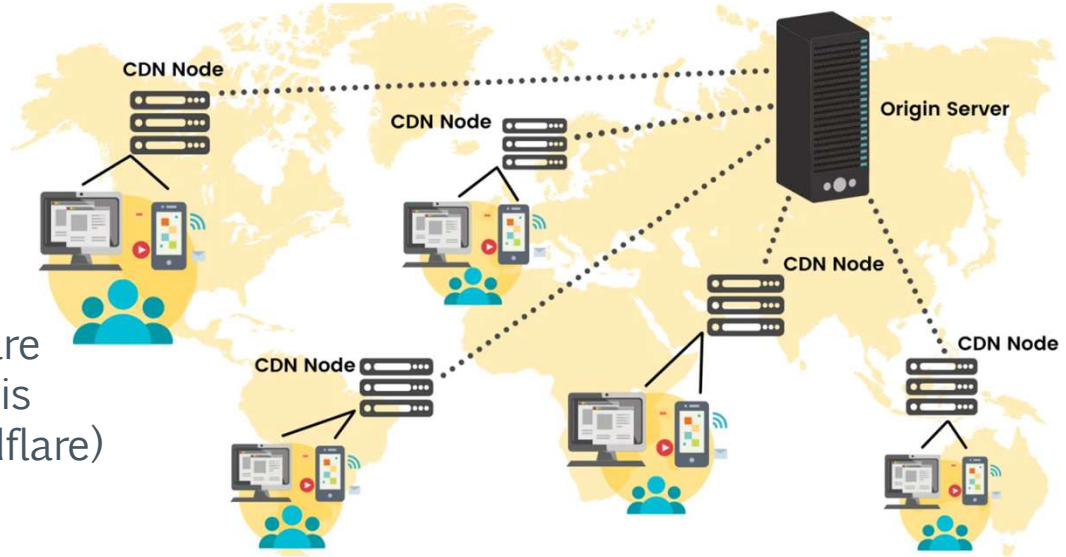
2 WORDS ON CONTENT DELIVERY NETWORKS

Origin vs. Edge

- **The Origin Server**: The "Source of Truth." This is where the original file (the song, the video, the data) is stored.
- **Edge Servers (PoPs - Points of Presence)**: These are smaller servers located in data centers globally
- **Caching**: When the first person in a given place ask for a resource, the CDN fetches it from the Origin. It then saves a copy on the Closest Ege Server. When the next other people close to that person watch it, they get it directly from the Edge Server.

Netflix has its own CDN, but the are enterprises whose core business is "renting" the networks (like Cloudflare)

Edge Computing !



CASE STUDY: CLOUDFLARE vs ITALY “PIRACY SHIELDS”

The "Piracy Shield" Controversy (2024-2025)

Italy implemented a platform called Piracy Shield (managed by AGCOM) to combat illegal live streaming of sports events (Serie A). The law requires providers to block offending IP addresses within 30 minutes.
Cloudflare uses Shared IP addresses (Anycast). One single IP can host thousands of legitimate websites alongside one illegal stream.

- The automated blocking led to "over-blocking," where legal sites (including hospitals, government portals, and small businesses) were accidentally shut down.

New regulations (October 2024) mandated that even DNS/CDN providers (like Cloudflare and Google) must automate these blocks and report suspicious activities or face criminal liability.

The company expressed “concerns” about the technical danger of blocking backbone infrastructure instead of specific URLs.

2 WORDS ON CONTENT DELIVERY NETWORKS

Feature	Peer-to-Peer (P2P)	Content Delivery Network (CDN)
Who serves the data?	Other users (unreliable).	Professional servers (highly reliable).
Battery/Data Impact	High (your phone works for others).	Low (your phone only receives).
Performance	Variable (depends on "Swarm" size).	Guaranteed high-speed throughput.
Security/Control	Hard to manage digital rights.	Total control over content and access.

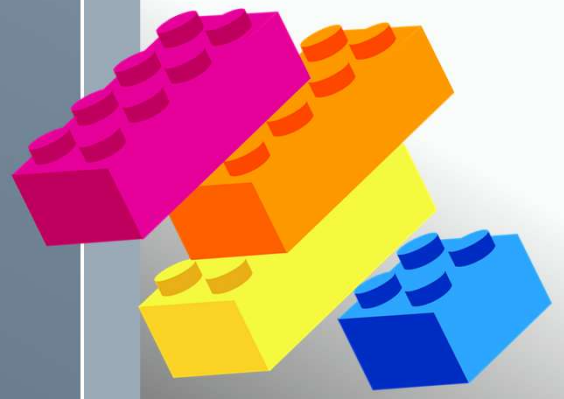
HARDWARE COMPONENTS OF A DATA INFRASTRUCTURE

HARDWARE

DATA INFRASTRUCTURE?

- Physical backbone: storing, processing, and transmitting data
- Critical for performance: scalability, reliability, and energy efficiency
- Composition: It includes servers, storage devices, networking equipment, and data centers

Generic Data Infrastructure



SOFTWARE COMPONENT

HARDWARE COMPONENT

Core Hardware Components

- Storage Systems: HDD, SSD, NAS, SAN
- Servers: Compute power for running applications and processing data
- Networking: Switches, routers, firewalls, load balancers
- Data Centers: Physical facilities for computing and storage
- Additional Systems: Backup and Disaster Recovery Systems

Core Hardware Components

- Storage Systems:
  - HDDs for cost-effective bulk storage
  - SSDs for fast, low-latency data access
  - NAS for file-level sharing across networks
  - SAN for high-speed, block-level access in enterprise environments

- Servers:
  - Compute power for running applications and processing data
  - Rack-mounted, blade, and tower servers
  - Equipped with CPUs, RAM, storage, and optionally GPUs
  - Handle computing workloads from applications to databases

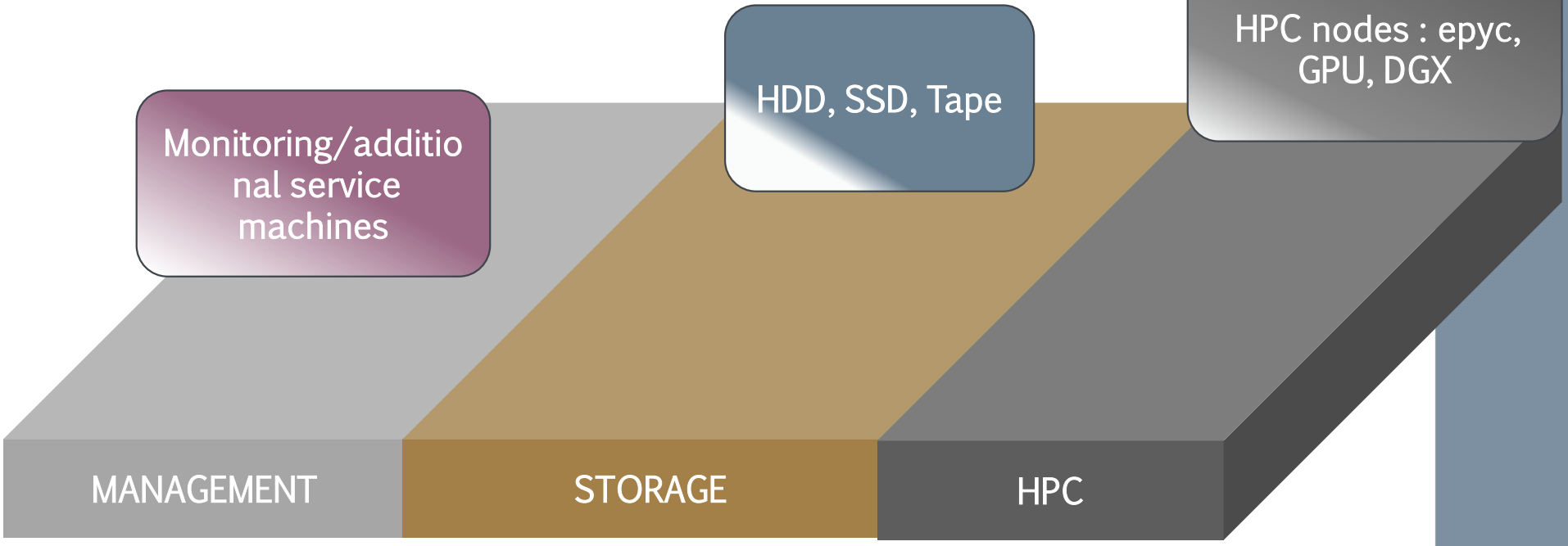
Core Hardware Components

- Networking Equipment:
  - Switches route data internally
  - Routers connect external/internal networks
  - Firewalls secure the perimeter
  - Load balancers optimize traffic and redundancy

- Data Centers:
  - Controlled environments for hosting infrastructure
  - Include power, cooling, and physical security systems
  - Rated by tiers based on uptime and redundancy levels

- Backup and Disaster Recovery:
  - Backup servers, tapes, or cloud solutions
  - Ensure continuity during hardware failure or disasters

Hardware Component



Features and Capabilities

- High Availability: Redundant systems, RAID, clustering

- Scalability: Modular and swappable components

- Performance: CPU/GPU specs, IOPS, bandwidth

- Energy Efficiency: Cooling, green hardware

- Security: Encryption, secure boot, access controls

Technical Challenges

- Hardware failures and downtime
- Integration of multi systems
- I/O latency and network bottlenecks
- Budget constraints: CapEx vs OpEx
- Environmental concerns: energy, e-waste

HARDWARE COMPONENT: STORAGE

WHAT IS STORAGE ?

- A system used to save and retrieve digital data persistently
- May be local (HDD/SSD), networked (NAS/SAN), or cloud-based
- Designed for durability, availability, and data access

STORAGE HIERARCHY

- Storage in data infrastructure includes multiple layers with different speeds and roles
- Organized by proximity to CPU and data volatility
- Trade-off between speed, capacity, and persistence

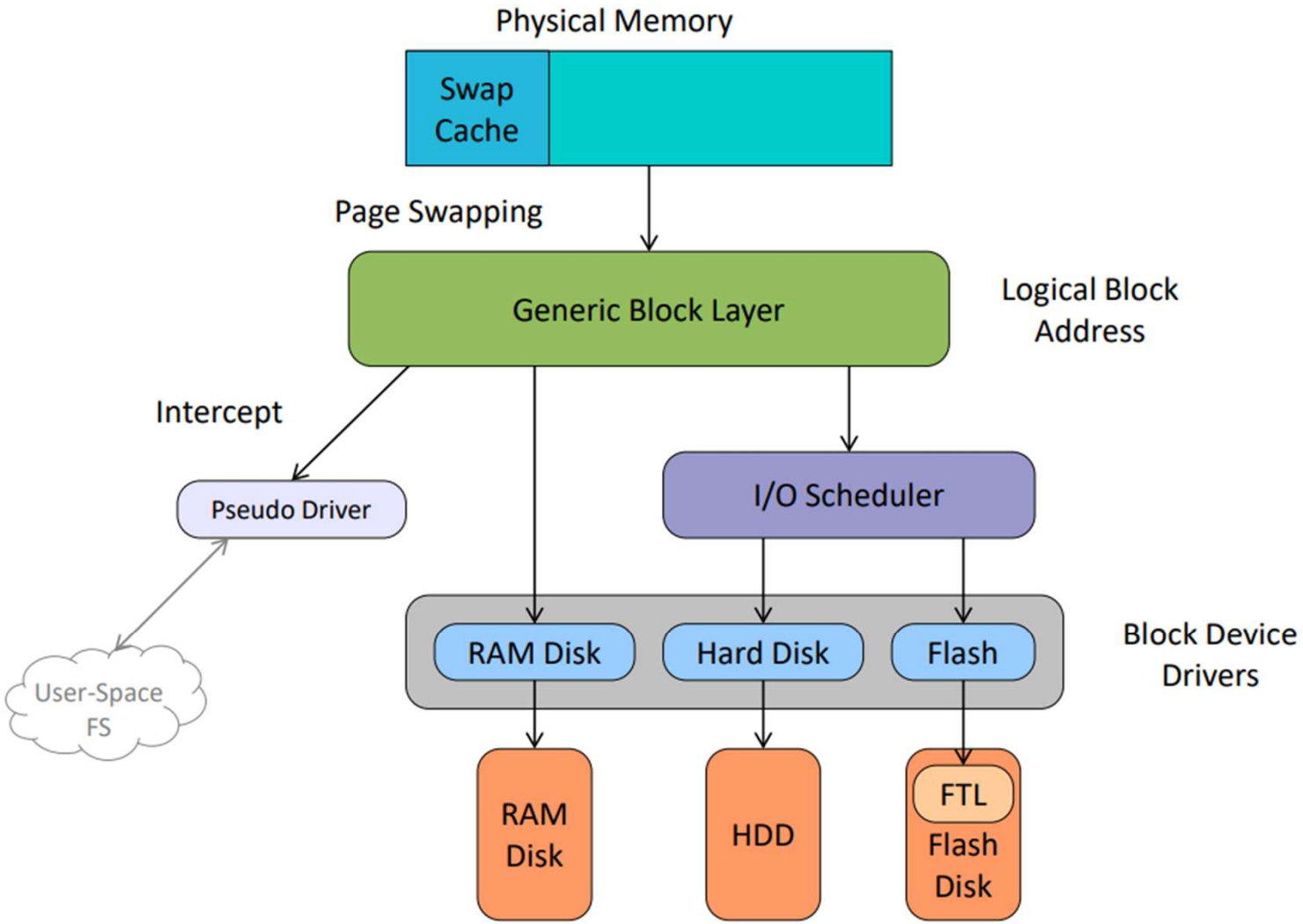
STORAGE:

- RAM disk, I/O buffers or file system cache
- Local disk (flash based, spinning disk,SATA, SAS, RAID, SSD, JBOD, …)
- Local network attached device or file system server (NAS, SAN NFS, CIFS, PFS,Lustre, GPFS,CEPH)
- Tape based archival system (often with disk cache)
- External, distributed file systems (Cloud storage)

KEY METRICS

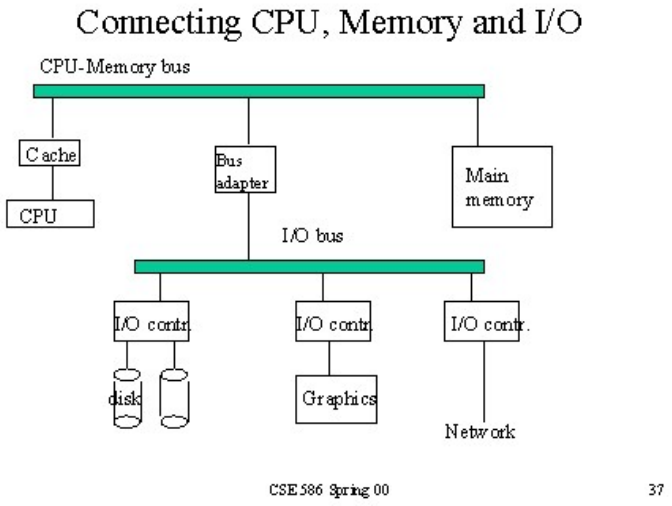
- Bandwidth: volume of data read/written in a second
  → throughput metric
- IOPs: number of I/O request processed by second
- Order of magnitudes: ·
  - Intel v2/v3 CPU-DRAM: 80/200 GB/s
  - Epyc node CPU-DRAM: 300 GB/s
  - IB link: 12 GB/s ·
  - Hard Drive: ~100- 400 MB/s

STORAGE HIERARCHY MULTIPLE LEVELS



RAM and I/O Operations

- RAM: Fast, volatile memory used for active data processing
- Temporary and non-persistent; data is lost on power-off
- I/O: Interface between CPU and external storage devices
- Affects performance in reading/writing to disks or network



File System Cache

- Resides in RAM and buffers frequently accessed disk data
- Reduces latency by minimizing direct disk access
- Automatically managed by the OS (Unix-like OS)
- Crucial for speeding up file operations in active applications
- faster access and less wear with RAM disk
- Linux provides “dynamic RAM disk” (tmpfs)
- only existing files consume RAM
- automatically cleared on reboot (-> volatile)

```
Filesystem  1K-blocks    Used    Available   Use%   Mounted on

devtmpfs       4096        0        4096     0%     /dev
tmpfs      20530316        0    20530316     0%     /dev/shm
tmpfs       8212128     1724     8210404     1%     /run
```

I/O Requests

- I/O (Input/Output) requests refer to operations made by a process to read from or write to a device
- in data storage, an I/O request typically means accessing or saving data on a disk, SSD, or network storage
- Read Request: Fetches data from storage
- Write Request: Sends data to be saved on storage
- Measured in IOPS (I/O Operations Per Second)

I/O Request Lifecycle

1. Application requests file access
2. OS generates an I/O request
3. File system resolves data location
4. Storage device performs read/write
5. Data returned to application

Local Disk Storage

- Non-volatile, persistent storage medium (HDD/SSD)
- Stores OS, applications, files, and databases
- SSDs offer faster access than traditional HDDs
- Limited by local device capacity and I/O speed

TRADITIONAL HARD DISK DRIVE (HDD)

- Mechanical device with spinning magnetic platters
- Data accessed by a moving read/write head
- Interfaces: SATA, SAS
- Rotating mechanical device (7200, 10000, 15000 rpm)
- Head on the right track · (seek time) 4 ms
- Head on the right sector
  - rotational latency 2ms
  - Capacity: 4-12 TB
- Bandwidth: Read / Write ~ 150/250 MB/s
- Strength: high capacity, low cost for GB
- Weakness: high latency, mechanical wear

CURRENT HDD TECHNOLOGIES

Two main technologies today:

- Serial Advanced Technology Attachment (SATA)
  - less expensive, and it's better suited for desktop file storage.
  - Up to 6 Gbit/sec

- Serial Attached SCSI (SAS)
  - more expensive, and it's better suited for use in servers or in processing-heavy computer workstations.
  - Up to 12Gbit/sec

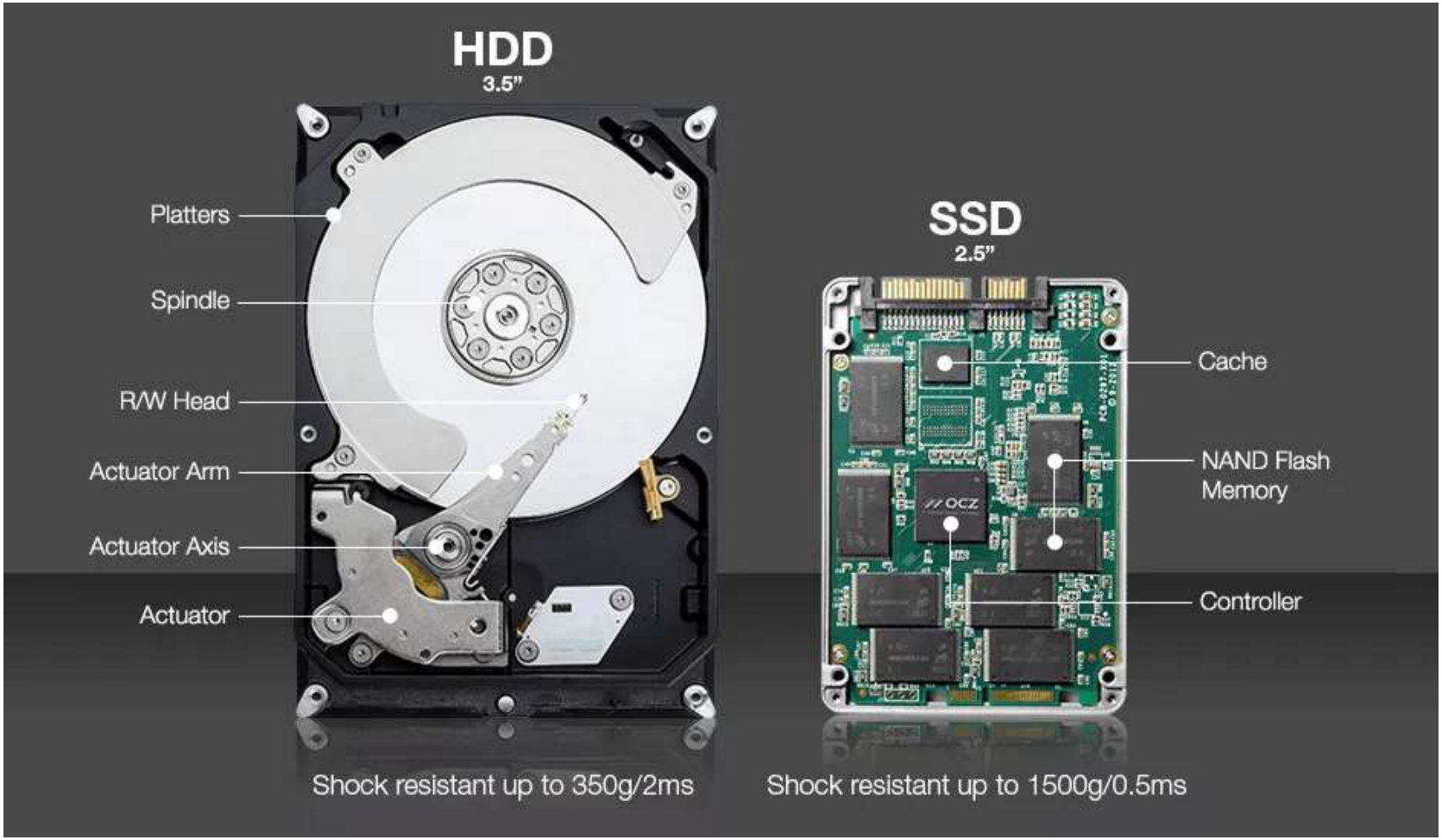
SOLID STATE DRIVE (SSD)

- Solid-state device with no moving parts
- Uses NAND flash memory for data storage
- Access time: ~0.1 ms (much faster than HDD)
- Interfaces: SATA, NVMe (PCIe)

SOLID STATE DRIVE (SSD)

- Strength:
  - lower access time and latency
  - no moving parts (silent, less susceptible to physical shock, low power consumption and heat production)
  - available over SATA, SAS, PCIe

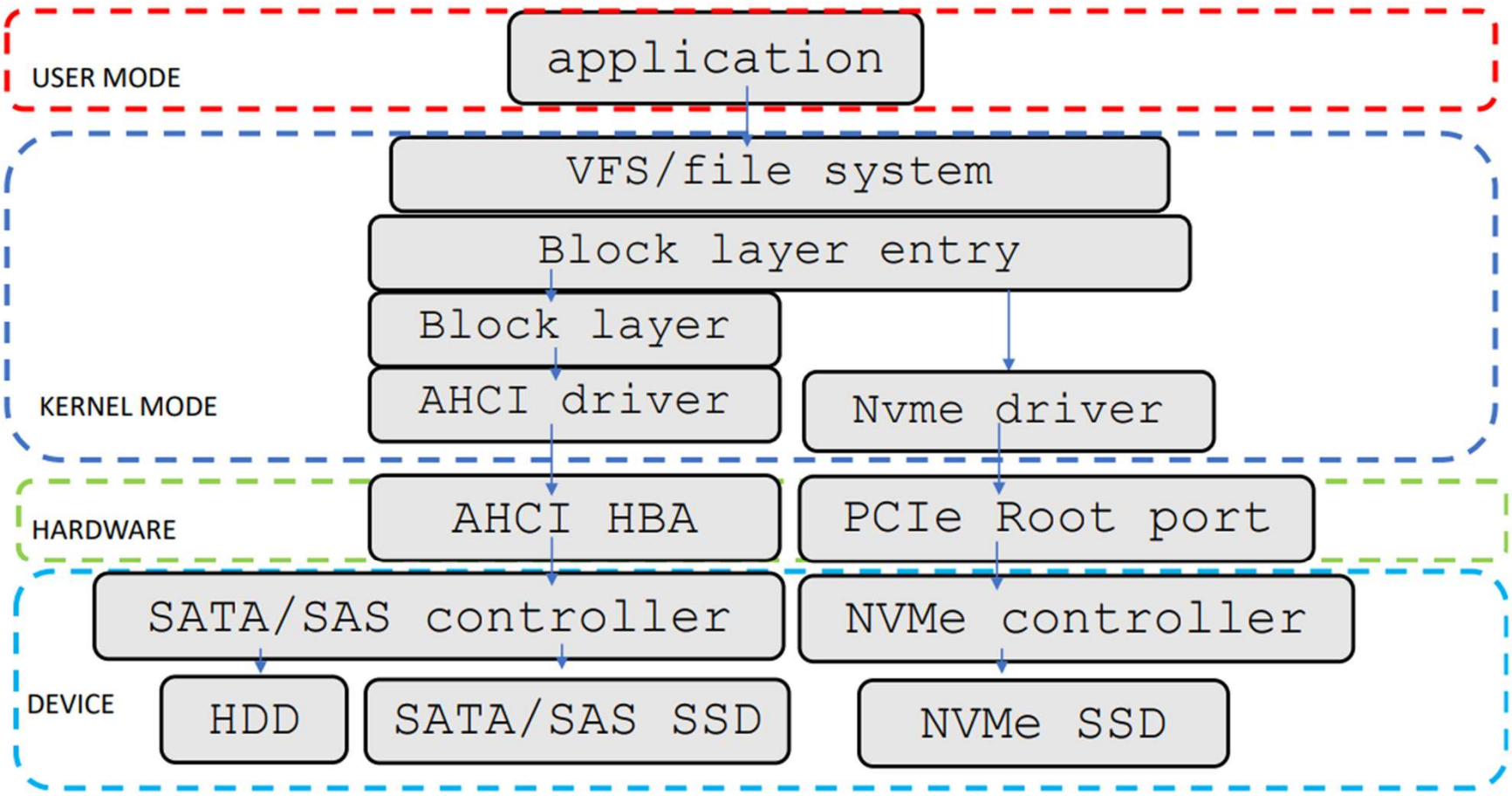
- Weakness:
  - expensive, low capacity;
  - usage limited to special purposes only (hardly used for big data-servers)
  - limited write-cycle durability (depending on technology and price)
    - SLC NAND flash ~ 100K erases per cell
    - MLC NAND flash ~ 5K-30K erases per cell ·
    - TLC NAND flash ~ 300-500 erases per cel


HDD
3.5”
Platters
Spindle
R/W Head
Actuator Arm
Actuator Axis
Actuator
Shock resistant up to 350g/2ms
SSD
2.5”
Cache
NAND Flash
Memory
Controller
Shock resistant up to 1500g/0.5ms

NON-VOLATILE MEMORY EXPRESS (NVMe)

- NVMe is an "*optimized, high-performance, scalable host controller interface with a streamlined register interface and command set designed for non-volatile memory based storage.*"
- Designed to fix many of the issues of legacy SAS/SATA.
  - SATA /SAS protocols for mechanical drive
  - Now the bottleneck
- Physical connectivity is much simplified, with devices connected directly on the PCIe bus

NON-VOLATILE MEMORY EXPRESS (NVMe)



COMPARISON

- UltraStar DC HC620 with SAS 12GB/s interface
  - Sustained transfer rate: 255 MBps read and write
- Samsung 970 Evo with PCIe 3 interface
  - Read speed 3,500 MBps
  - Write speed 2,500 MBps

Type	Sequential Read Speed
HDD 7200 RPM	~ 100 MB/s
SSD SATA	~ 500-550 MB/s
SSD NVMe PCIe 3.0 x4	~ 3500 MB/s
SSD NVMe PCIe 4.0 x4	~ 7000 MB/s

COMPARISON

Type	Cost
HDD 1 TB	~ <50 €
SSD 1 TB SATA	~ 50-80 €
SSD 1 TB NVMe	~ 100 €

April 2025 prices

TRADEOFF

- Performances
- Dimension
- costs

COMPARISON

DATA Temperature: based on the frequence we need to access to the data

- **Hot Data:** Constantly accessed → **RAM / NVMe.**
- **Warm Data:** Accessed weekly/monthly → **SATA SSD.**
- **Cold Data:** Rarely accessed → **HDD / Cloud Cold Storage.**

Storage Type	Temperature	Latency	Cost per GB	Primary Use Case
RAM	Hot	< 100 ns	Extreme	Cache, Real-time transient data
NVMe SSD	Hot	~10-50 µs	High	Transactional DBs, AI Training
SATA SSD	Warm	~100-500 µs	Medium	Active Data Warehousing
HDD	Warm/Cold	~5-10 ms	Low	Data Lakes, Massive Big Data storage
Tape	Cold	Mins / Hours	Minimal	Long-term Backup, Compliance

HARDWARE SELECTION AND STORAGE STRATEGIES

Storage choice is never binary (SSD vs. HDD), but a **Tiering Strategy** is implemented.

Core Principle : The cost of storage must scale proportionally with the business/"research" value/needs of the data

Scenario A : Real-Time Analytics (High Performance)
Example: fraud detection systems or stock exchange monitoring.
For this system latency is the enemy. Every millisecond could correlate to financial loss. **Hot Data!**

- ❑ **Hardware Choice: NVMe SSD** (Non-Volatile Memory Express).
- ❑ **Rationale:** Traditional SSDs are bottlenecked by the SATA interface. NVMe communicates directly with the CPU via the PCIe bus.
- ❑ **Configuration:**
  - **Hot Tier:** The last hour/day of data resides in RAM (In-memory DBs like Redis) or NVMe.

HARDWARE SELECTION AND STORAGE STRATEGIES

Scenario B: Enterprise Data Warehouse (Balanced)
Example: daily BI (Business Intelligence) reporting.
In this case it is neccessary balancing massive capacity and analytical speed.
Warm and Cold Data!

- ❑ Hardware Choice: SATA/SAS SSD for active compute; Object Storage (HDD-based) for persistence.
- ❑ Tiering Strategy:
  - Warm Tier (SSD): Holds the last 3–6 months of data. Allows scanning terabyte-scale tables in seconds.
  - Cold Tier (HDD): Historical data is moved to Object Storage

Scenario C: Archive & Compliance (Deep Archive)
Example: 10-year legal log retention or medical imaging backups.
In this case with high probability Data is "Write Once, Read Never" (WORN). (Very) Cold Data!

- ❑ Hardware Choice: HDD (less performant) or LTO Tape (Magnetic Tape).
- ❑ Rationale: HDD is unbeatable for density/price. Tape has zero power consumption
- ❑ Latency: Acceptable at minutes to hours for retrieval.

HARDWARE SELECTION AND STORAGE STRATEGIES

and how a Research Data Center is designed?

See Lecture 5

AI INFRASTRUCTURE

Storage Balancing (Data Feeding)

Three-layer architectural approach:

- **Layer 1: Local NVMe (Hot):** Every AI node is equipped with local NVMe to provide immediate caching for training data. This minimizes latency by keeping the most frequent data chunks physically close to the GPUs.
- **Layer 2: Parallel File System (Warm/Active):** Utilizing systems for parallel computation this layer aggregates hundreds of storage servers to deliver throughput in the range of several Terabytes per second. Its primary role is to "feed" GPUs without interruption.
- **Layer 3: Data Lake (Cold):** This is where the raw and massive datasets reside. It utilizes cost-effective HDDs for long-term retention.

NETWORK ATTACHED STORAGE (NAS)

- A file-level storage device connected to a local network
- Allows multiple users and systems to access data over the network
- Appears as a network drive or shared folder

- Common use cases:
  - File sharing in home or office
  - Centralized backups
  - Media servers

- Supports protocols like NFS, SMB/CIFS, FTP

NETWORK ATTACHED STORAGE (NAS)

› ✅ Advantages:

- Easy to deploy and use
- Centralized data access
- Cost-effective for small setups

› ❌ Disadvantages:

- Limited by local network speed
- Less scalable than enterprise SAN systems
- May require maintenance and configuration

CLOUD STORAGE

› A storage service provided over the internet by third-party providers

Examples: Google Drive, Dropbox, Amazon S3, Microsoft Azure Blob Storage

› Characteristics:

- Elastic and scalable
- Accessible globally
- Pay-as-you-go pricing model
- High durability and availability

CLOUD STORAGE

- ✅ Advantages:
  - Scalable and flexible
  - No physical maintenance required
  - High availability and redundancy

- ❌ Disadvantages:
  - Depends on internet connection
  - Latency for large file transfers
  - Data security and privacy considerations

DIRECT ATTACHED STORAGE (DAS)

- **DAS** is digital storage that is **physically connected** to a single computer or server. Unlike NAS or SAN, it does **not** use a network (no switches, no routers) to communicate.

- Connection: Internal (SATA/NVMe slots) or External (SAS/Thunderbolt cables).
- The "Zero-Latency" Path: Data travels directly via the PCIe bus to the CPU. There are no "network hops" to slow it down.
- Protocol: Usually communicates at the Block Level, which is the fastest way for a CPU to talk to a disk.

  - In a **Modern Data Infrastructure**, DAS is used
    - Edge Computing: CDN nodes that need to serve video *now*.
    - AI Training: "Scratch space" for GPUs to read training data at Terabytes per second.
    - High-Performance Databases: Where every microsecond of "seek time" matters.

CONTENT DELIVERY NETWORKS

- Edge Tier : DAS (NVMe SSD)
  Minimum latency. At the Edge, we need the speed of Direct Attached Storage.
- Origin Tier (The Central Warehouse): Object Storage (S3/GCS)
  Infinite capacity and low cost. Object Storage, however, scales horizontally to infinity, making it the perfect "Source of Truth."
- Office/Internal Apps (Administration) : NAS
  Seamless file sharing between colleagues (e.g., PDFs, Excel spreadsheets). A latency of a few milliseconds is unnoticeable to a human opening a document. In this context, the ease of collaboration and centralized file management is the primary advantage.

RAID (Redundant Array of Independent Disks)

- RAID is a way to aggregate multiple physical devices into a larger virtual device
- Redundant Array of Inexpensive Disks · Redundant Array of Independent Devices · Invented by Patterson, Gibson, Katz, et al · hTtp://www.cs.cmu.edu/~garth/RAIDpaper/Patterson88.pdf
- Redundant data is computed and stored so the system can recover from disk failures
- RAID was invented for bandwidth
- RAID was successful because of its reliability
- Improves data redundancy, performance, or both depending on the RAID level
- Commonly used in servers, NAS, and storage arrays

Key goals:

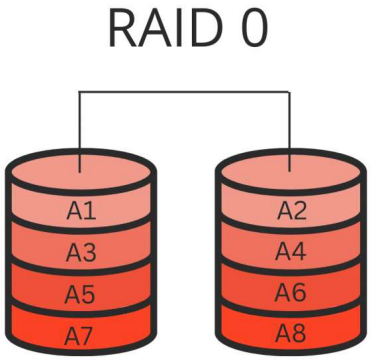
- Redundancy (fault tolerance)
- Speed (read/write performance)
- Scalability (larger combined capacity)

RAID RELIABILITY AND PERFORMANCE

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

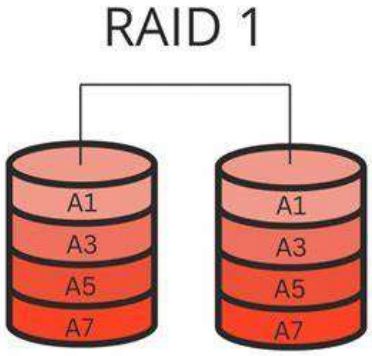
RAID 0 : striping

- Performance= P*N
- Capacity=N*S



RAID 1 : redundacy

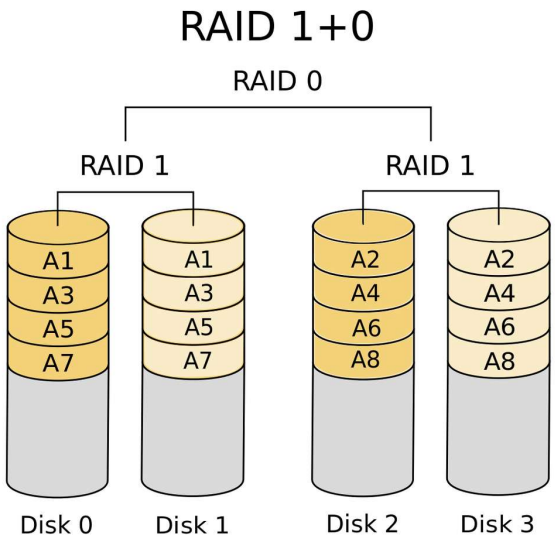
- Write Performance=P
- Read Performance= P*N
- Capacity=S



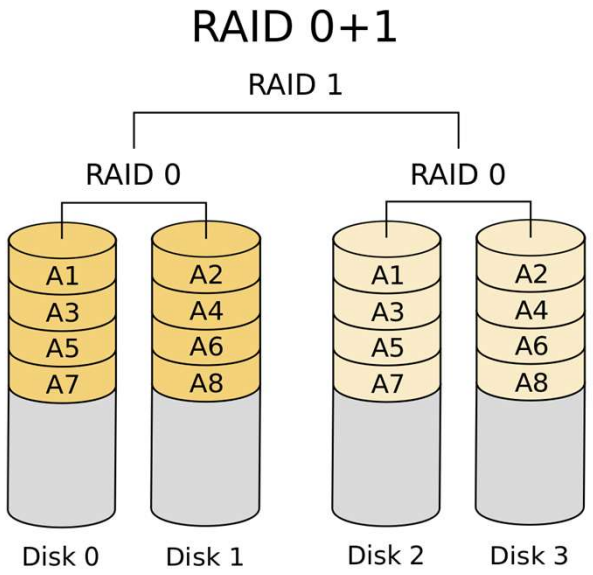
But higher reliability!

RAID 10 : striping + redundancy (1+0 or 0+1)

- Raid 1+0 : mirrored sets in a striped set
- the array can sustain multiple drive losses so long as no mirror loses all its drives ·

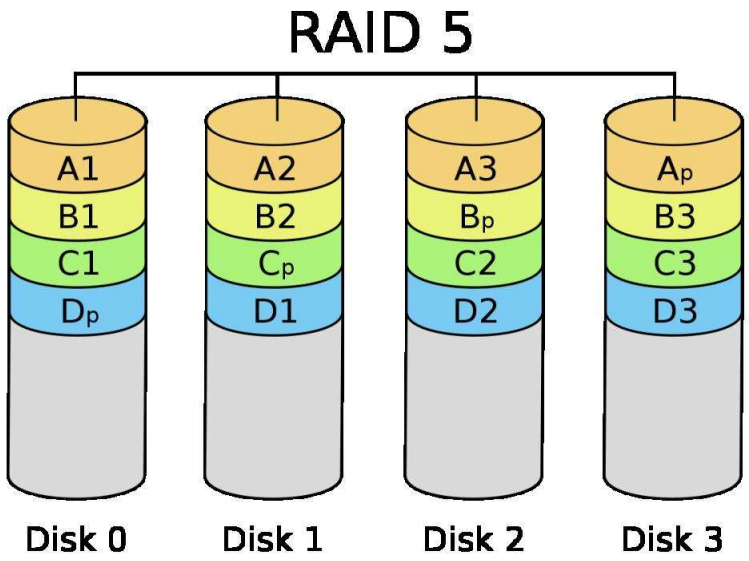


- Raid 0+1: striped sets in mirrored set
- if drives fail on both sides of the mirror the data are lost



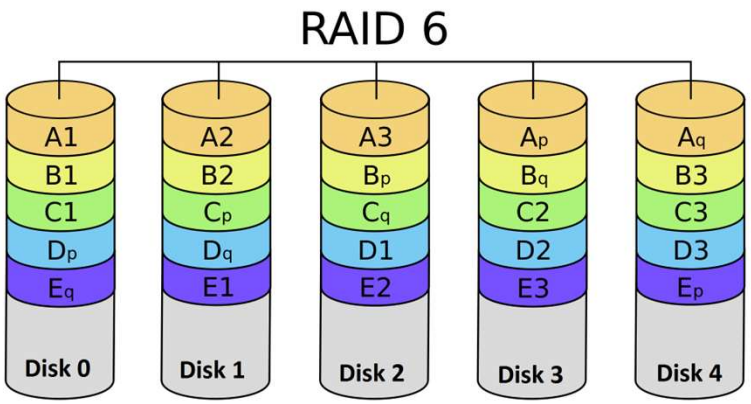
RAID 5

- One disk can fail
- Distributed Parity



RAID 6

- Two disks can fail
- Double distributed parity



RAID COMPARISON

RAID Level	Description	Redundancy	Performance	Min Disks
RAID 0	Striping	No	High	2
RAID 1	Mirroring	Yes	Moderate	2
RAID 5	Striping + Parity	Yes (1 disk)	Balanced	3
RAID 6	Striping + Double Parity	Yes (2 disks)	Lower Write	4
RAID 10	Mirror of Stripes (RAID 1+0)	Yes	High	4

OBSERVATIONS ON RAID

- Computing and updating parity negatively impact the performance. Upon drive failure, though, lost data can be reconstructed, and any subsequent read can be calculated from the distributed parity such that the drive failure is masked to the end user.
- However, a single drive failure results in reduced performance of the entire array until the failed drive has been replaced and the associated data rebuilt.
- The larger the drive, the longer the rebuild takes (up to several hours (even days) on busy systems or large disks/arrays)

HOT SPARE

- A drive physically installed in the array which is inactive until an active drive fails, when the system automatically replaces the failed drive with the spare, rebuilding the array with the spare drive included.
- A hot spare can be shared by multiple RAID sets. Subsequent additional failure(s) in the same RAID redundancy group before the array is fully rebuilt can cause data loss.
- RAID 6 without a spare uses the same number of drives as RAID 5 with a hot spare and protects data against failure of up to two drives

IMPLEMENTING RAID

- RAID is implemented both in hardware and software. RAID controller is the hardware part.
- Totally transparent to the users
- Configured when the system is installed
- No way to change it on the fly

PARQUET FORMAT STORAGE STANDARD FOR DISTRIBUTED BIG DATA

Columnar vs. Row-Based

Traditional databases store data in Rows (like a list of patient names). Parquet stores data in Columns.

- Row-Based: Name, Age, Diagnosis | Name, Age, Diagnosis.
- Columnar (Parquet): All Names | All Ages | All Diagnoses.

- Parquet was designed to be handled by many machines at once
- Horizontal Splittability: A single Parquet file is divided into "Row Groups." This allows a distributed engine (like Spark or Presto) to work in parallel without interference (what happens with iterations?)
- Predicate Pushdown (Intelligence): Parquet stores metadata (Min/Max values) for every column, the system skips the file entirely without reading it if the metadata did not match

Parquet is compressed, intelligent, and designed to be 'eaten' by a hundred computers!

EXERCISE: PLAN YOUR STORAGE!

You are running a social media app. Where would you store posts from the last 24 hours? And where would you store the posts of a user who has not logged in since 2018? Justify your hardware choice based on costs and predictions.