MDMC
Master in Data Management and Curation

AREA SCIENCE PARK

SISSA

Introduction to Data Infrastructure

CC BY

Trieste, 19 & 21/10/2026
Federica Bazzocchi, RIT – Area Science Park

Why this module:

- Any platform, any tool we use are deployed on a well defined infrastructure
- A deep understanding of data life cycle goes through knowing where data lives, moves and understanding why infrastructural choices reflect on quality, accessibility and persevreance



Module's objective:

- Understanding the key components of Data Infrastructure
- Basic knowledge of different architectural models
- Understanding the connexion between managing data and designing a data infrastructure

When you will apply these competences:

- If you will have to plan and/or deploy a service in production

MDMC
Master in Data Management and Curation

AREA SCIENCE PARK

SISSA

Agenda

- ❑ Some reflexion on DATA
- ❑ What is Data Infrastructure
- ❑ Data Infrastructure's Design
- ❑ Hardware Components
- ❑ Data Architecture and Data Stack
- ❑ Few words on Identity Providers



CC BY

REFLEXION ON DATA

WHAT IS

DATA

?

My old (ingenuous) BIAS as young physics' student (that I am not anymore) and science enthusiast (that I still am)

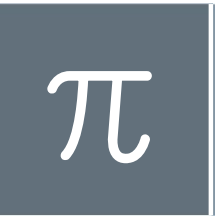


- Data is a results of a measurement;
- It is objective;
- It is quantitative;
- It is related to a physical law/phenomena;
- It is analyzed by mean of statistics and is used to validate a/make prediction by a (analytical) mathematical model.



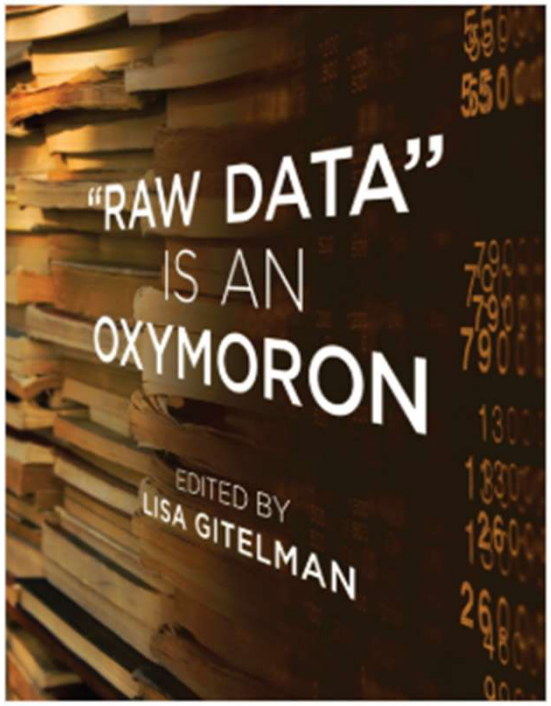
DATA is a wider (and not clear) concept

- "Data is any set of characters that has been gathered and translated to some purpose, usually analysis. It can be any character, including text and numbers, pictures, sound, or video"
  (https://www.computerhope.com/jargon/d/data.html)
- "Data is information that has been translated into a form that is efficient for movement or processing"
  (https://searchdatamanagement.techtarget.com/definition/data)
- · "Data is a collection of facts, such as numbers, words, measurements, observations or even just description of things"
  (https://www.mathsisfun.com/data/data.html)



DATA journey

- The word “data” derives from the Latin word “datum” (singular), which means the “thing given”
- A **data** can be defined as a fundamental unit of **raw unstructured information**, represented in different form that can be transferred and then **recorded, processed, analyzed** and then **interpreted**.
- 19000 B.C-**calculation**- Ishango bone (baboon tool) : the first **mathematical data** (intended as information registered)
- I640s –**medical data-** John Graunt started collecting information regarding deaths in London (number of death, mortality rate per age, causes)
- 1880s-**data processing-** the German-American statistician Herman Hollerith had the idea of using punch cards in writing and processing data. With this invention Hollerith helped the American government complete the US census within the same year.
- 1928 – **magnetic tape**- German engineer Fritz Pfleumer patented a magnetic tape that he used to replace wire recording for storing data.
- 1960s – **relational database** – idea introduced by the computer scientist Codd
- 1990s **–** Internet and then Google **– the rise of big data**

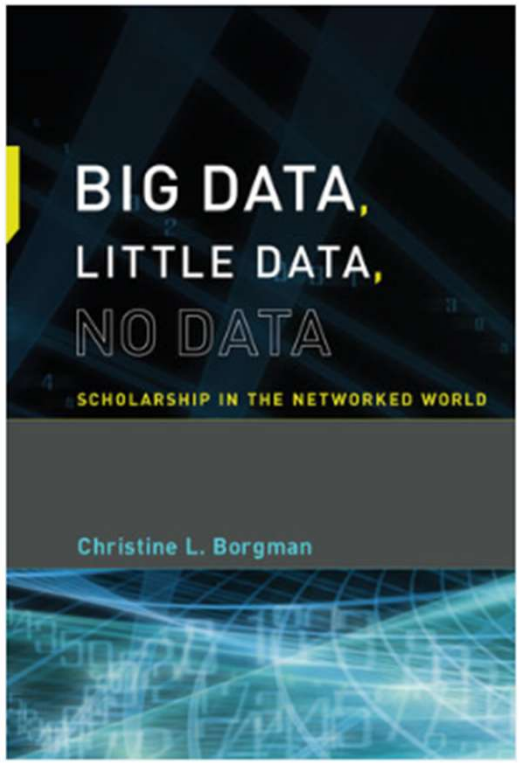


The MIT Press **DOI:** https://doi.org/10.7551/mitpress/9302.001.0001 **ISBN electronic:** 9780262312325 **Publication date:** 2013

Johanna Drucker on data vs. capta

Johanna Drucker in Humanities Approaches to Graphical Display:

> *Capta is "taken" actively while data is assumed to be a "given" able to be recorded and observed. From this distinction, a world of differences arises. Humanistic inquiry acknowledges the situated, partial, and constitutive character of knowledge production, the recognition that knowledge is constructed, taken, not simply given as a natural representation of pre-existing fact.*



Data are “representations of observations, objects, or other entities used as evidence of phenomena”

(and here we could open a discussion on what we mean with phenomena)

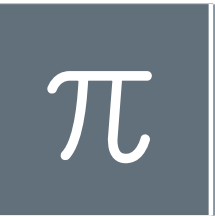
The MIT Press **DOI:** https://doi.org/10.7551/mitpress/9963.001.0001 **ISBN electronic** 9780262327862**Publication date:** 2015

Types of Data

- Structured Data: Organized in tables or databases (e.g., name, age, address).
- Unstructured Data: Texts, images, videos, audio, without a predefined structure.
- Semi-structured Data: JSON, XML, which have some organization but are not as rigid as relational databases.
- Qualitative vs. Quantitative Data: Words vs. numbers.

Data Sources

- Data are not only created anymore to write scientific papers or doing analysis, but they are created with the notion of being reused in different contexts which is revolutionary in many disciplines;
- Data are produced by almost everything
- Advanced statistical methods (machine learning/deep learning) are required and have allowed to detect the patterns and correlations hidden in the data



DATA ≠ **INFORMATION**

DATA is raw and processed values, it becomes INFORMATION when it is enriched by metadata

If we take the general definition

INFORMATION= DATA + MEANING

Data may be seen as raw and processed substrate, to which we add metadata to get information

What is «information»?

Not a easy definition…

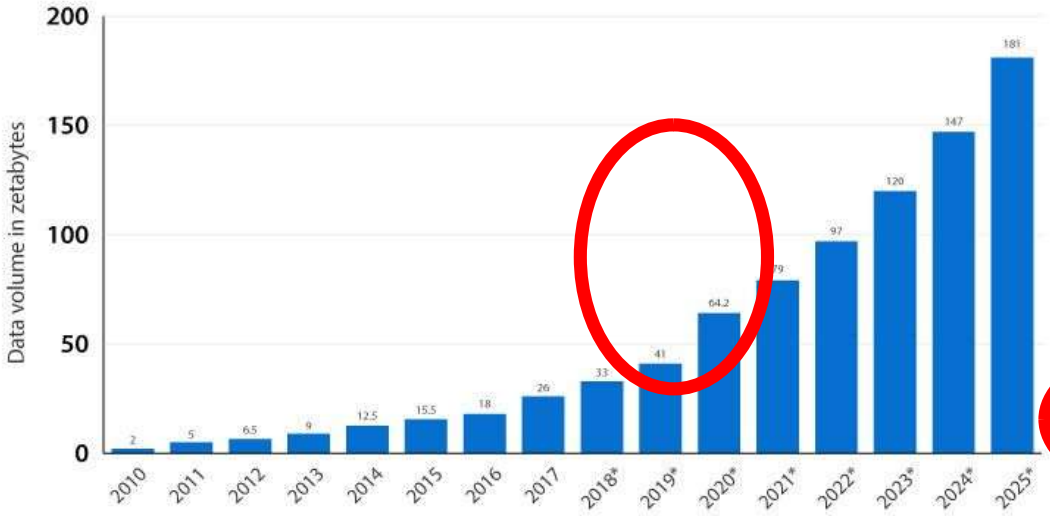
«Information is data that has been processed into a form that is meaningful to the recipient.» (Davis, Olson, 1985, p. 200)
«Data are the raw material that is processed and reworked to generate information.» (Silver, Silver, 1989, p. 6)
«Information equals data plus meaning.» (Checkland, Scholes, 1990, p. 303)
«Information consists of data that have been interpreted and understood by the message recipient.» (Lucey, 1991, p. 5)
«Data must be interpreted or manipulated [to] become information.» (Warner, 1996, p. 1)

Filosofia dell'informazione, L. Floridi

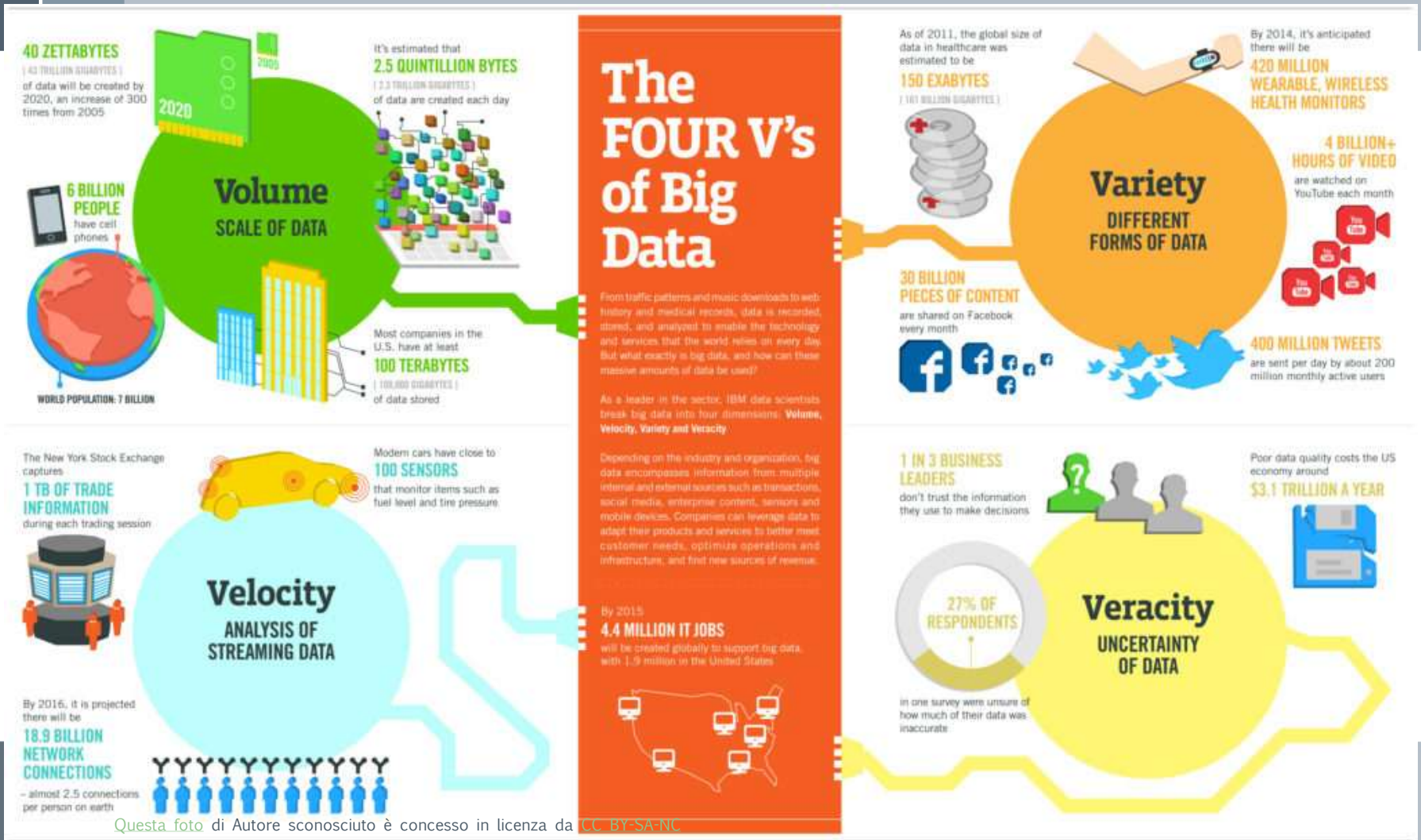
DATA explosion in the last decades

The volume of data/information created, captured, copied and consumed worldwide from 2010 to 2025

Storage is rouhghly 10% of all traffic



Year	World Storage Size (Exabytes)
1986	2.6 EB
1993	15.8 EB
2000	54.5 EB
2007	295 EB
2014	5000 EB
2020	6800 EB



Questa foto di Autore sconosciuto è concesso in licenza da CC BY-SA-NC

HOW WE MANAGE THIS

HUGE AMOUNT OF DATA

?



WHERE WE STORE THIS

HUGE AMOUNT OF DATA

?

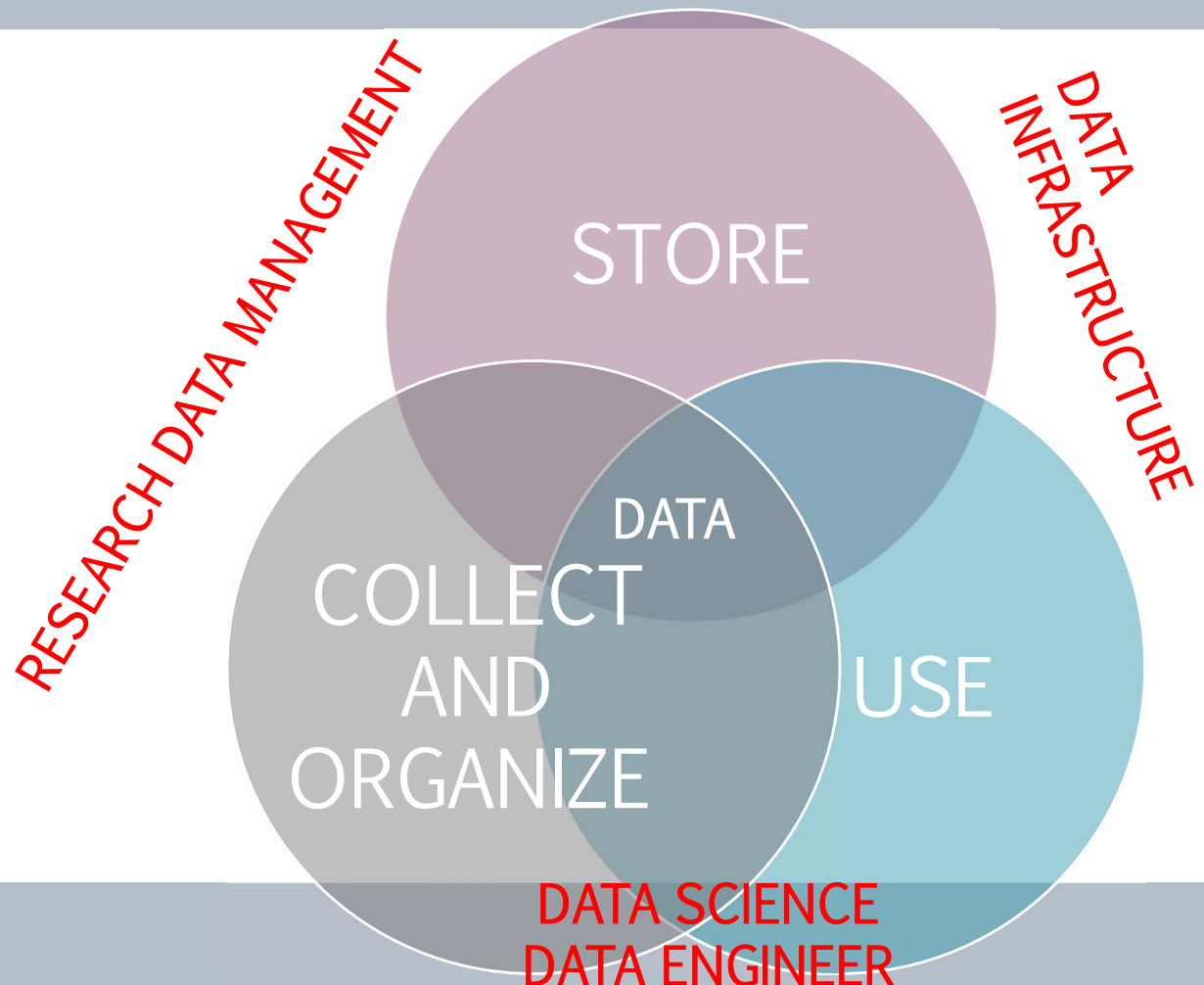
π

HOW WE USE (AND WHAT WE MAY DO WITH) THIS

HUGE AMOUNT OF DATA

π

?


RESEARCH DATA MANAGEMENT
DATA INFRASTRUCTURE
STORE
DATA
COLLECT AND ORGANIZE
USE
DATA SCIENCE
DATA ENGINEER
AI ENGINEER

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

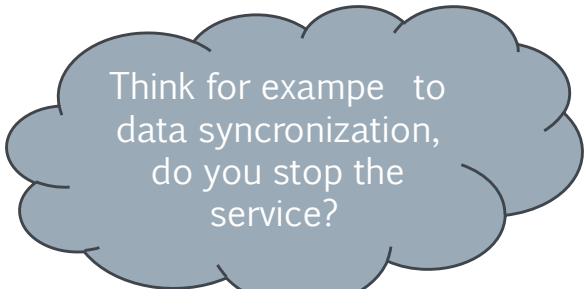
- Store large datasets and large data rates from experiments/business activities
- Allow reliability by replicating data sources
- Allow accessibility by copying source to several places
- Monitor and check resource usage
- Provide a set of integrated services which are compatible between domains
- Increase interoperability through common standard schemas
- 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 -PHYSICAL MODEL (DEPLOYMENT)

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

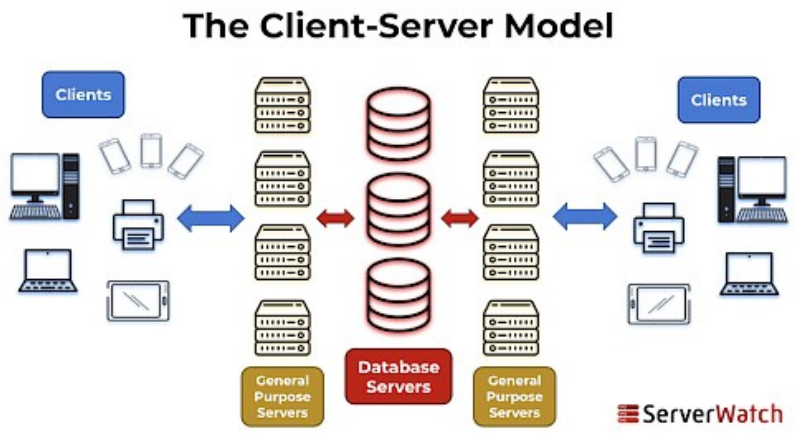
Distributed Architecture

- 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

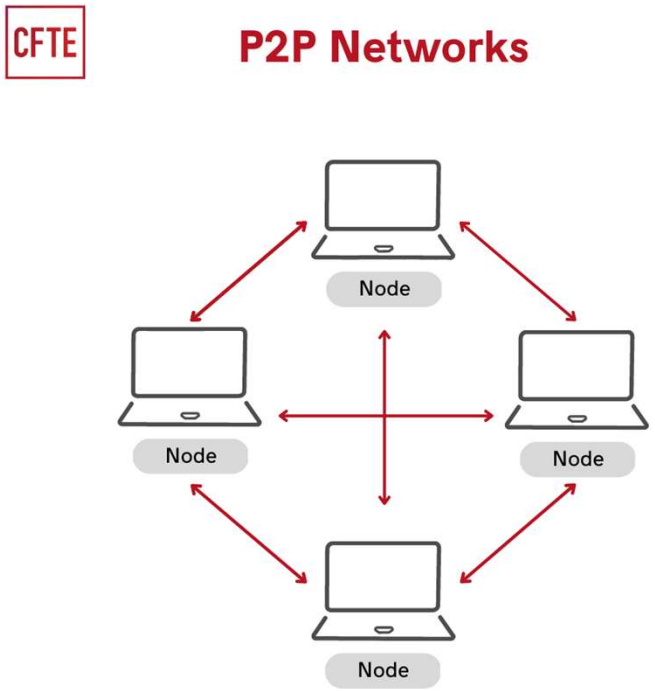
- Centralized architecture with clients inititate requests for services or sources (anything from web browser to mobile apps to desktop applications)
- Server: central componets that provide resources or services to the clients. Server powerful machine or sw system designed to handle multiple requests simultaneously (resources managed databases, files, 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

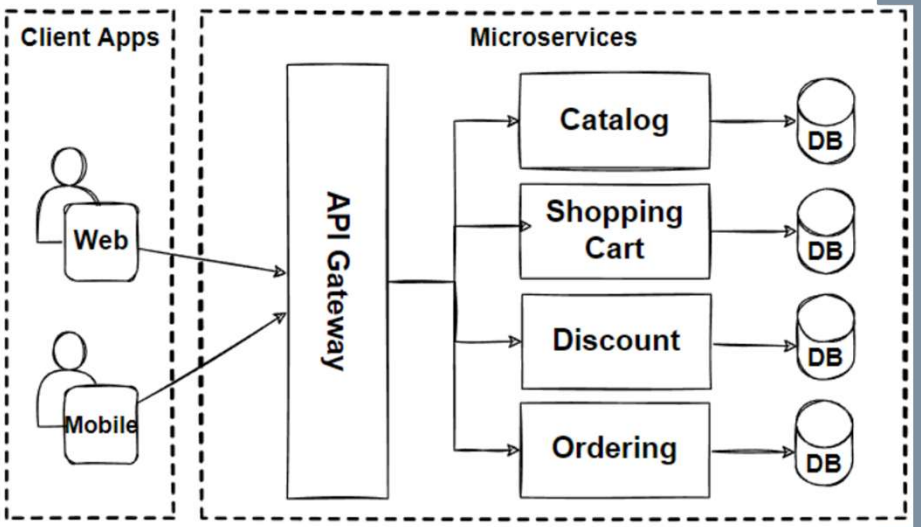
- Decentralized model where each node acts as both client and server
- Nodes equal authority, share resources and communicate directly
- Each peer can send request, provide data, share computational power.
- Examples: BitTorrent, Blockchain, Skype
- Pros: High resilience, scalability, fault-tolerant (no single point of failure) decentralization makes it well suited for app that requires scalability and 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 (which role have the components and how they interact)

Comparison Overview

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

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

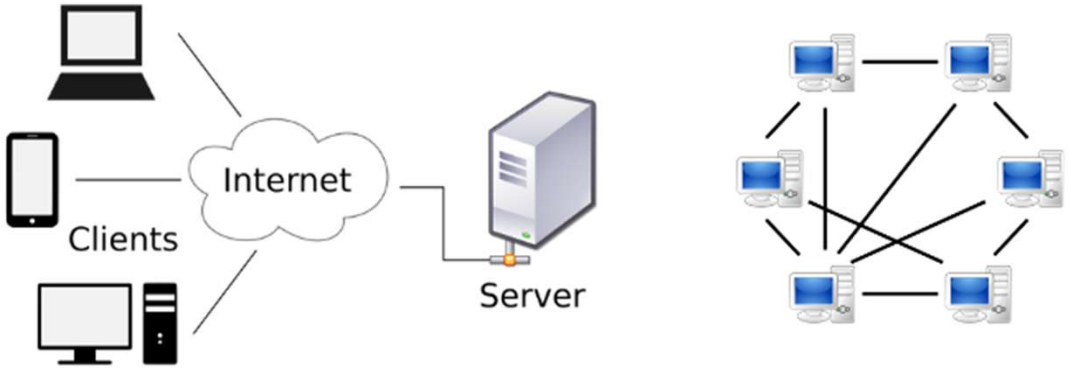
- Microservices:
  - Modular, loosely coupled
  - Scalable and flexible
  - Requires DevOps and orchestration

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



ARCHITECTURAL MODELS

Hybrid Infrastructures:

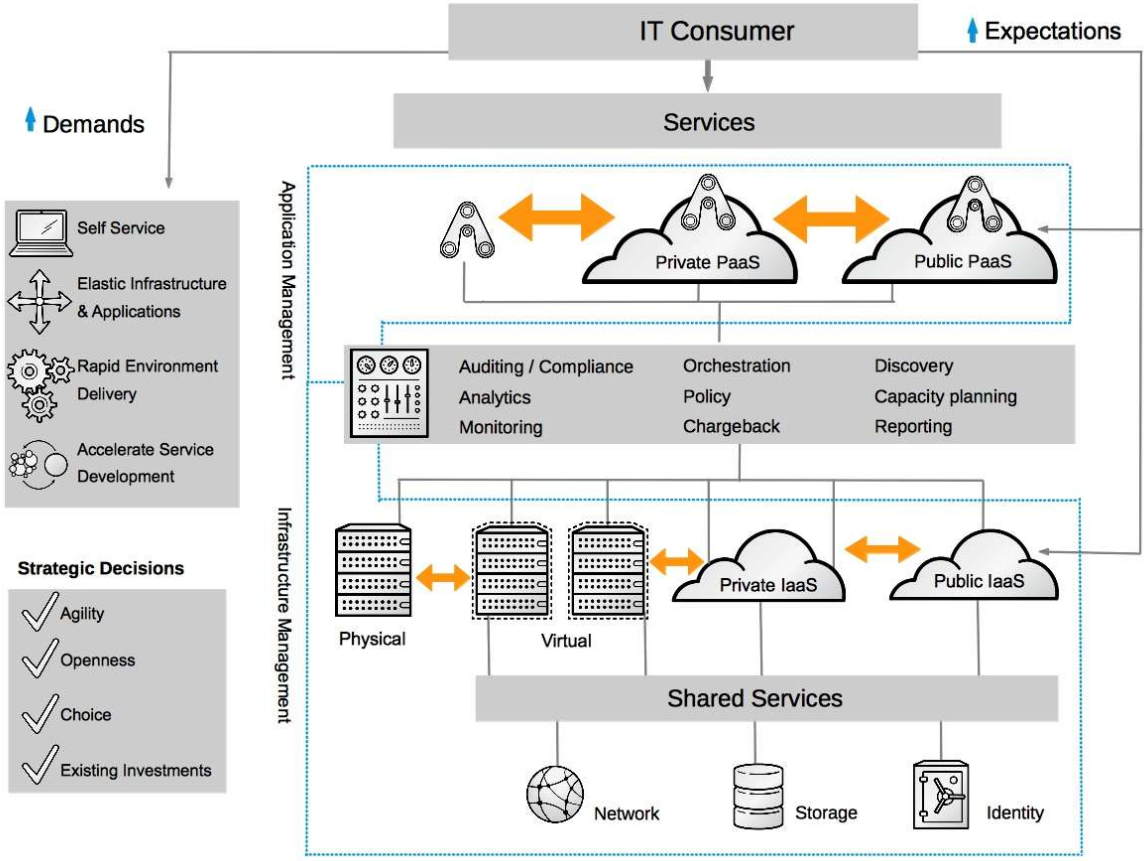
Combining advantages of **Client-Server** with the **resiliency of P2P**.

- **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:

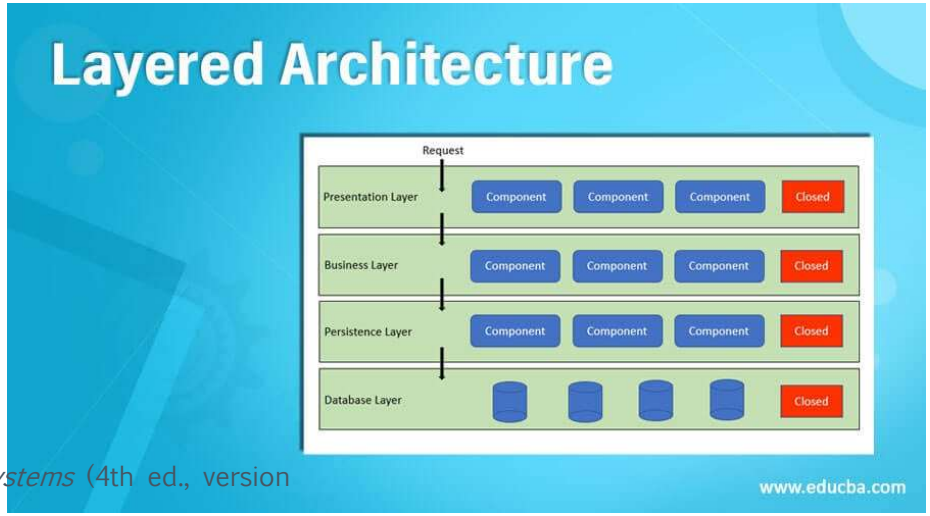
Combining advantages of **Client-Server** with the **resiliency of P2P**.



LOGICAL MODEL ARCHITECTURES (What the system does and how it is organised, regardless of where it runs)

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



van Steen, M., & Tanenbaum, A. S. (2023). *Distributed systems* (4th ed., version 4.01). Maarten van Steen. ISBN 978-90-815406-3-6.

LOGICAL MODEL ARCHITECTURES

Layered Model - Example

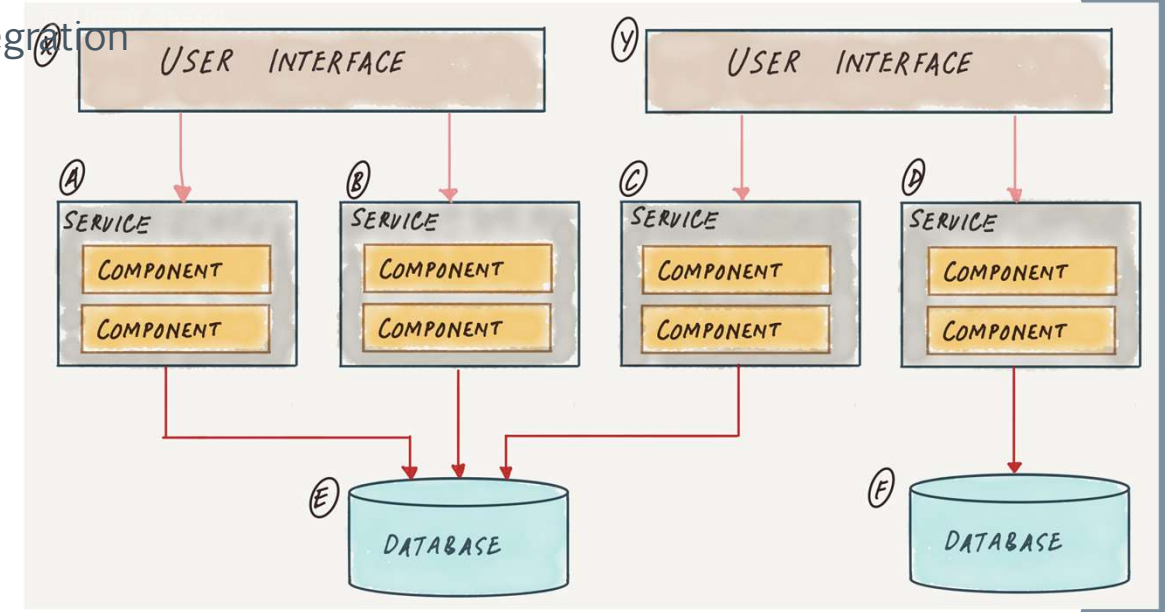
- **Presentation layer**: the user interface; how users interact with the system
- **Application (logic) layer**: rules, processing and decisions
- **Data layer**: storage, retrieval and management of data

the same logical model can be deployed in very different ways, from a single laptop to thousands of servers worldwide.

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



LOGICAL MODEL ARCHITECTURES

Resource Based (REST)

- **Everything is a resource with its own address:** data are exposed as resources identified by a URL (e.g. a dataset, a record, a file), and handled with a small set of standard operations (GET, POST, PUT, DELETE).
- **Simple, stateless and interoperable:** each request contains all the information needed, so any client can access the data without special software. This is how most data repositories and catalogues offer access through APIs.

LOGICAL MODEL ARCHITECTURES

Event-Based (PUBLISH-SUBSCRIBE)

- **Components don't call each other directly:** producers publish events (e.g. "new record added") and consumers subscribe to the types of events they care about, without knowing who sent them.
- **Loose coupling and real-time flows:** components can be added or removed independently, and data move as soon as something happens. It underpins many real-time data pipelines and notification systems.

COMBINING ARCHITECTURES/MODELS

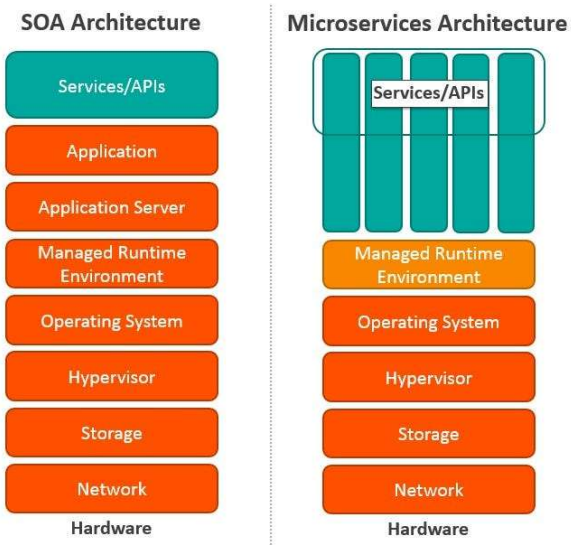
Architectural Model	Only Distributed ?	Typical of Distributed System
Client-Server	NO	NO
P2P	YES	YES
Microservices	NO, BUT ALMOST ALWAYS	YES

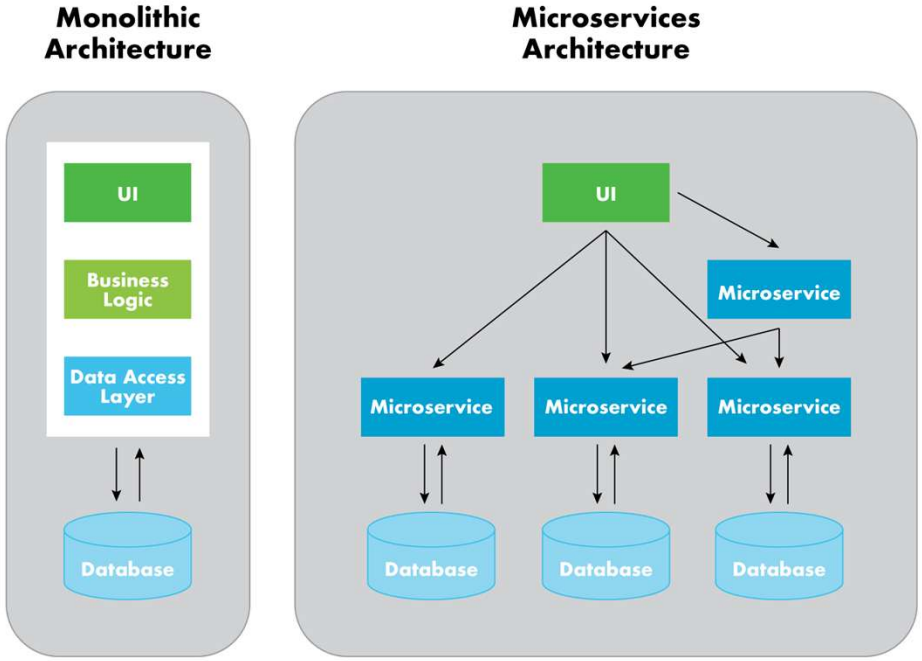
Architectural Model	Only Distributed ?	Notes
Layered	NO	Can be implemented on a single system (e.g.web app)
SOA	YES	Requires communication between decoupled services

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 architectural 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!)

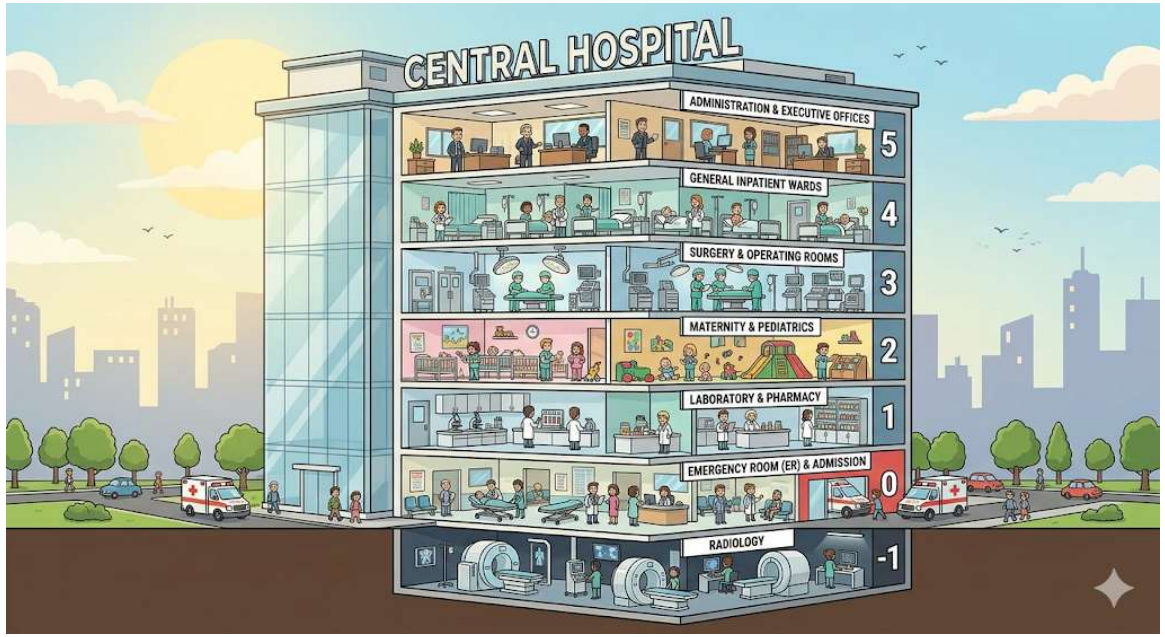


Image generated by AI

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!)

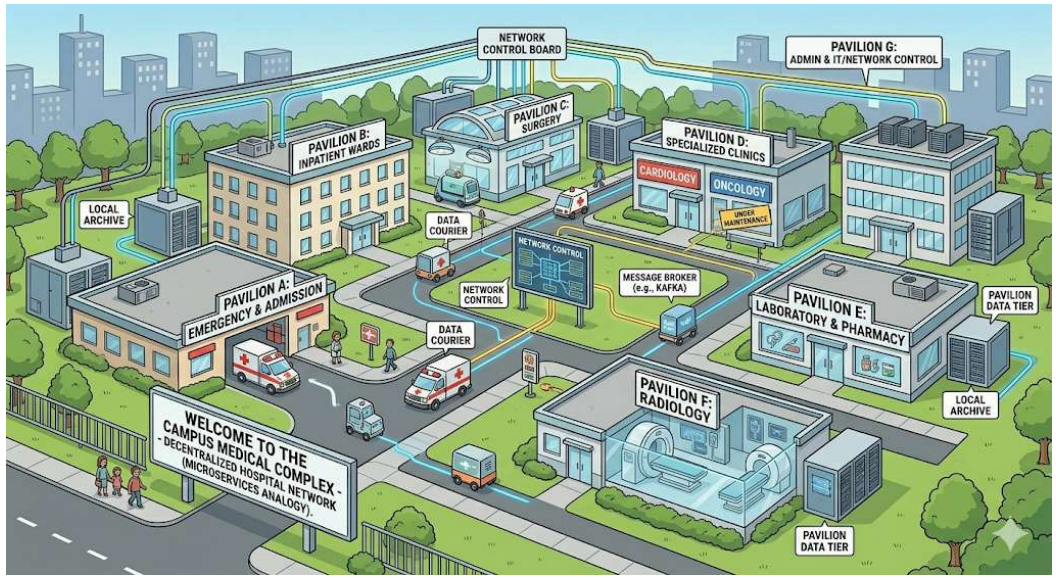


Image generated by AI

AN ANALOGY- DOCTORVILLE

- A town 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"

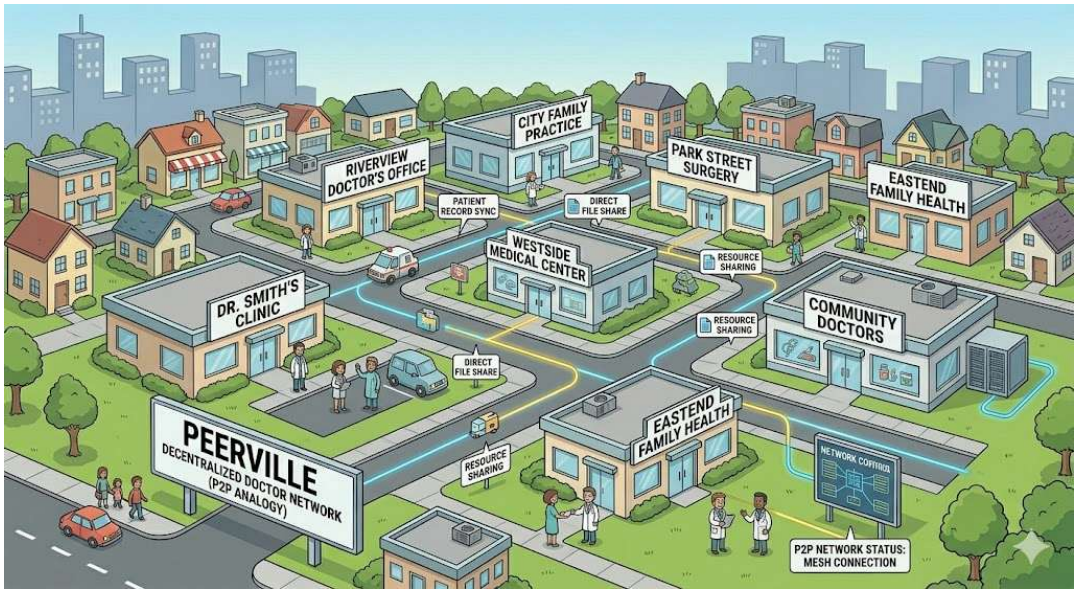
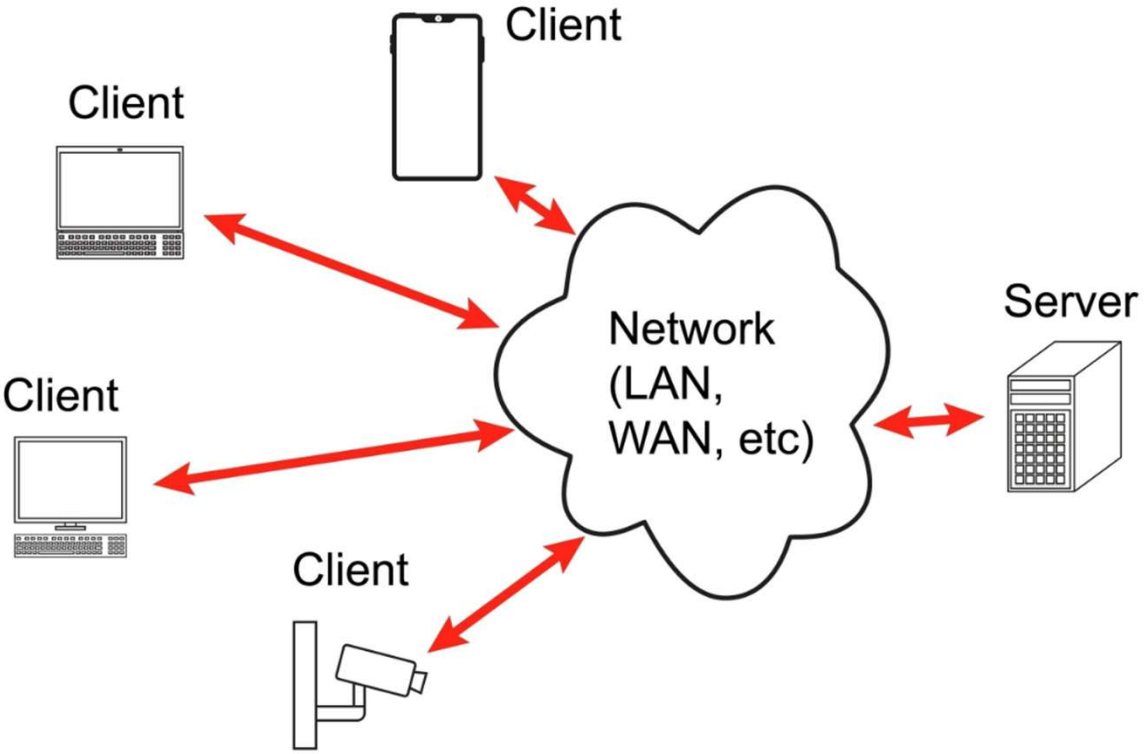


Image generated by AI



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 BIT TORRENT USE CASE



- **The ideal**: every peer is both client and server, with no central authority
- **Obstacle**: most devices sit behind NAT, with private addresses that are unreachable from outside (NAT=Network Address Translation)
- **The workarounds**:
  - Outbound connections always work, and once open they carry data both ways
  - Routers can be asked to open ports automatically (UPnP / NAT-PMP) or manually (port forwarding)
  - Two peers behind NAT can connect via *hole punching*, coordinated by a third node
- **Finding peers**: trackers (centralised), DHT (distributed), peer exchange
    - **Step 1, Discovery:** find who is sharing the file (tracker, DHT, peer exchange)
    - **Step 2, Connectivity:** reach them despite NAT (open ports, outbound connections, hole punching)
    - **Not every pair connects, yet the swarm still works:** reachable peers carry more of the load

Takeaway:

- "pure" P2P is an abstraction; real systems adapt to the infrastructure beneath them
- who opens the connection doesn't matter: once connected, both peers can upload and download.
- every decentralised network needs a known entry point: trackers or bootstrap nodes.

PEER-TO-PEER EXAMPLE

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



I never knew! When did I give my agreement?

The "Hidden" P2P Social Contract

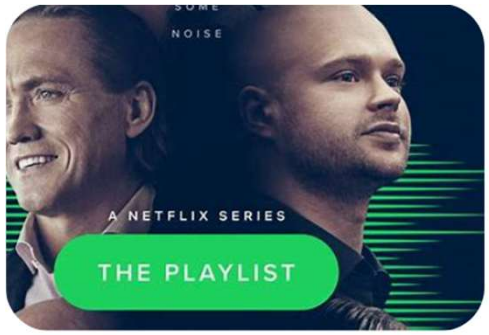
- **The Agreement:** I accepted the role of becoming a **node** in their network in exchange for free files or significantly higher download speeds.
- **Implicit Consent:** By downloading and running the software, I granted my laptop permission to "listen" for requests from other peers and transmit data 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.

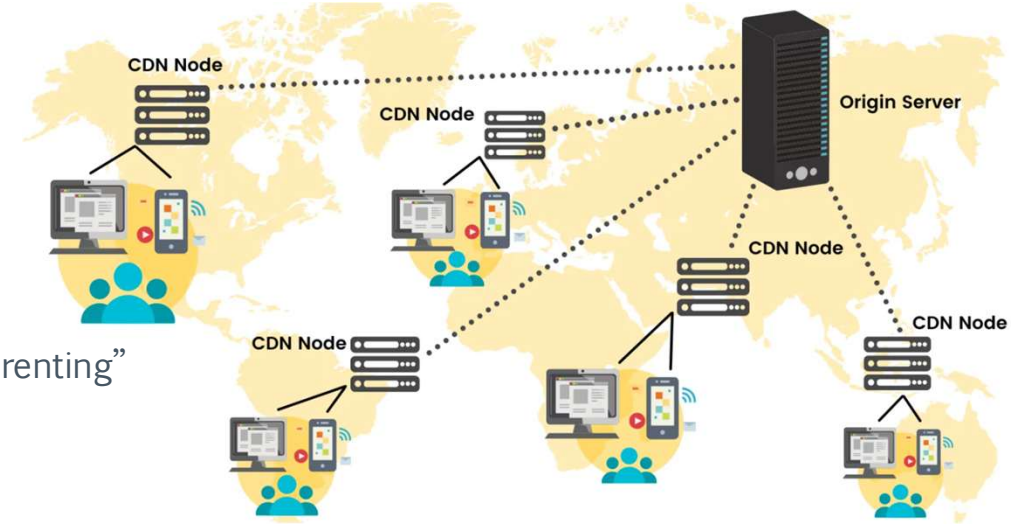
as infrastructure shifted from desktop PCs (with unlimited wired internet) to mobile phones (with limited battery and data), the **Client-Server model** regained its role via **Content Delivery Networks (CDNs).**

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.

COMPARING THREE USE CASES

Similar Problem solved by three Architectures

How do you deliver large amounts of data to many users?

- **BitTorrent:** distributed among users, out of necessity, no central infrastructure; peers share the load
- **Spotify:** hybrid at first, then centralised once it made economic sense, P2P cut bandwidth costs, until mobile and scale changed the equation
- **Netflix:** centralised control, distributed at the network edge

In each case the architecture follows the constraints!

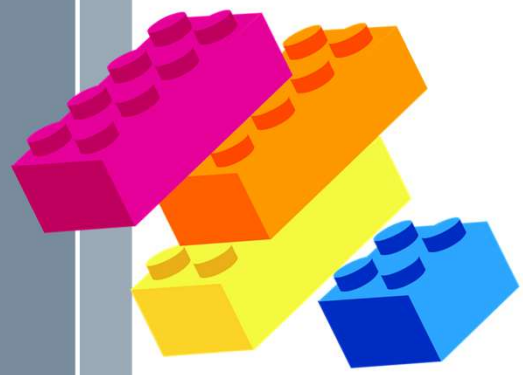
HARDWARE COMPONENTS OF A DATA INFRASTRUCTURE

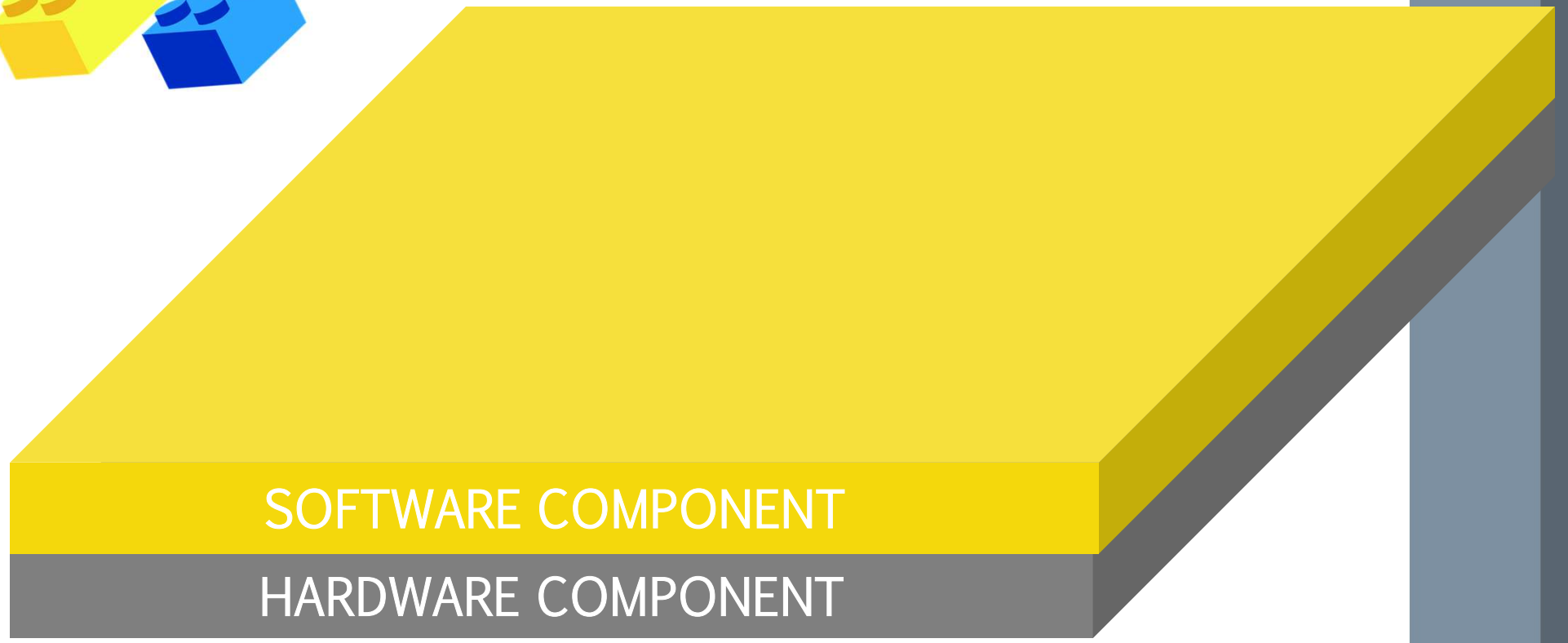
HARDWARE

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





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

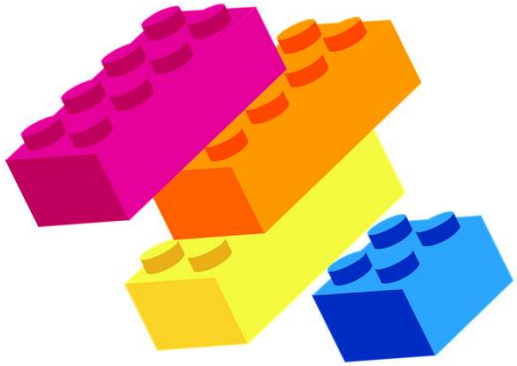
Features and Capabilities

- 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

Where we left ...

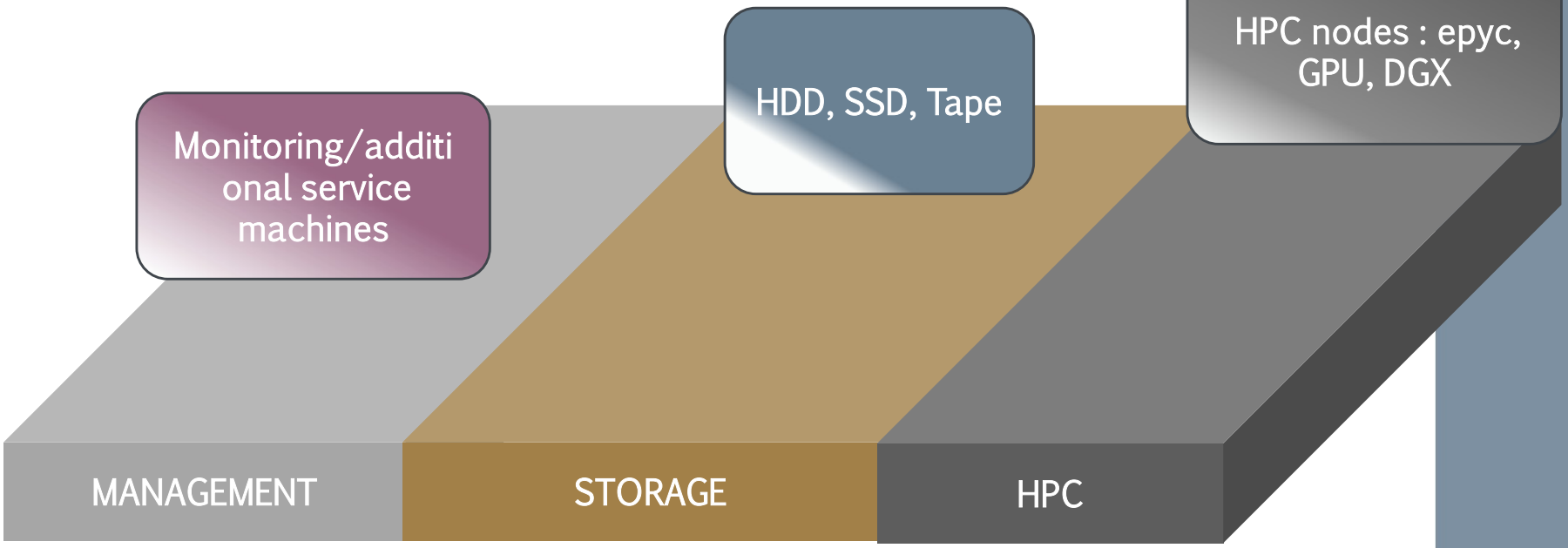


Generic Data Infrastructure

SOFTWARE COMPONENT

HARDWARE COMPONENT

Hardware Component



HARDWARE COMPONENT: STORAGE



Image generated by AI

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

COMPARISON

TRADEOFF

- Performances
- Dimension
- costs

Type	Sequential Reading	Reference
HDD 7200 RPM	~ 150–210 MB/s	Seagate BarraCuda 1 TB (ST1000DM010): 156 MB/s medi, 210 MB/s max
SSD SATA	~ 500–550 MB/s	Crucial BX500: 540 MB/s (interface limit SATA 6 GB/s)
SSD NVMe PCIe 3.0 x4	~ 3500 MB/s	Newegg, PCIe 5.0 2026
SSD NVMe PCIe 4.0 x4	~ 7000–7450 MB/s	Samsung 990 Pro, WD SN850X
SSD NVMe PCIe 5.0 x4	~ 11000–14900 MB/s	WD_BLACK SN8100, Samsung 9100 Pro, Crucial T705

Type	Price	Reference
HDD 1 TB	~ 50–60 €	Seagate BarraCuda, WD Blue
SSD 1 TB SATA	~ 140–160 €	Tuttotech
SSD 1 TB NVMe	~ 140–150 € (entry Gen4) / ~ 230–270 € (fascia alta)	Tuttotech

October 2026 prices

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

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)

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)

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

COMPARISON

October 2026 prices

Type	Price	Reference
16 GB DDR4 (2x8 GB)	~ 130–190 €	Yeppon
16 GB DDR5 (2x8 GB)	~ 260–370 €	Trovaprezzi; Pangoly
32 GB DDR5 (2x16 GB)	~ 400–450 €	Trovaprezzi

Type	Price	Reference
16 GB DDR4 (2x8 GB)	~ 70–85 €	DropReference
32 GB DDR5 (2x16 GB)	~ 140–240 €	DropReference; MisterGadget

October 2025 prices

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 has to be minimized. 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?

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.

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



There are rules to organize the warehouse…

FILE SYSTEM

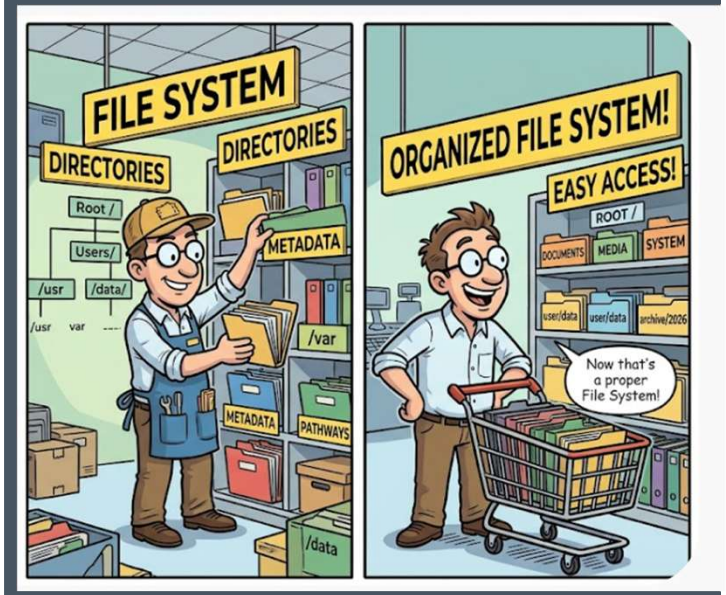


Image generated by AI

WHAT IS A FILE SYSTEM?

- it is a storage system on a memory support that structures files and directories and organizes different actions on the files (writing, reading, modifying, deleting,...etc)
- The structure is such that file identification and access are as fast as possible
- It maintains data integrity and it is essential for file operations, access control, metadata management, and backups

A file system is the set of rules, structures, and tools that turn "chaos" into accessible files.

Ceph: Open source distributed storage system



CephFS

- CephFS is a distributed file system built on top of Ceph
- POSIX-compliant: supports standard file operations
- Designed for scalability, redundancy, and high availability
- Used in HPC, cloud, and data center environments
- High scalability
- No single point of failure
- Seamless integration with Ceph object and block storage
- Open-source and community supported

Use Cases of CephFS

- Scientific computing and HPC
- Backup and archival solutions
- Large-scale content repositories
- Hybrid cloud infrastructure

Flexible!

Object Storage System vs FS

- At the physical level, both store bits on HDDs/SSDs using similar block-level operations.
- The difference lies in how the OS manages the mapping between the user's request and the physical sectors.

File System (organized warehouse)

- **Structure:** Hierarchical (Tree-based).
- **Access:** POSIX-compliant (Open, Read, Write, Close).
- **Limit:** Highly dependent on the OS kernel; managing millions of nested folders creates massive metadata overhead.

Object Storage (The Warehouse):

- **Structure:** Flat Namespace (No folders, just "Buckets").
- **Access:** RESTful APIs (HTTP GET, PUT, POST, DELETE).
- **Philosophy:** Data is treated as an "Object" containing the Data + Metadata + Global ID.

ORFEO storage after PNRR

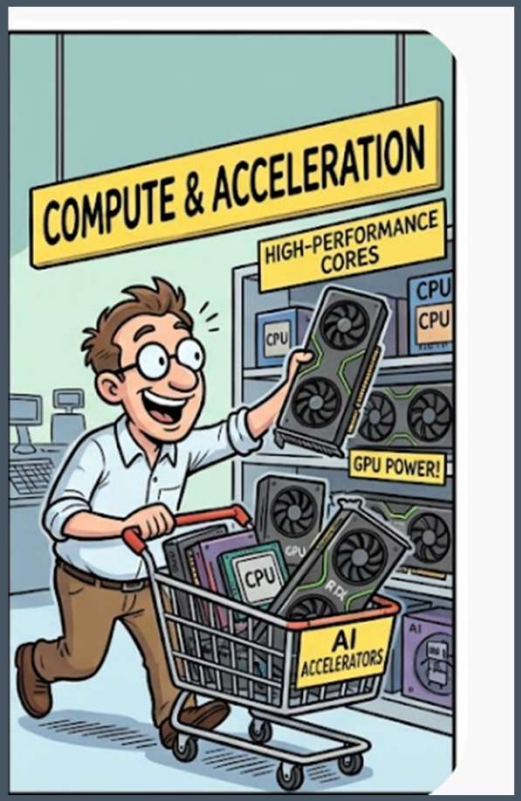
	Standard storage (HDD)	Fast storage (SSD)	Super Fast storage (NVME)	Long Term Storage
Node capacity	(Gen1) 212 TB (Gen2) 294 TB (Gen3) 294 TB	84 TB	2 x 1.6 TB 2 x 14 TB (Gen3 nodes)	2962.5 TB
# of Nodes	6 x Gen1 + 2 x Gen2 + 11 x Gen3	4 nodes	2 devices each node, **Gen3 nodes have 2x 14 TB !**	Served via NFS server with a SAN.
Storage provider	CEPH	CEPH	CEPH	Network FS (NFS)
Increment wrt before PNRR	+ 130 % more HDD storage		+ 1340 % more NVME storage	
RAW storage	~ 5684 TB	~ 340 TB	~ 340 TB	~ 2962 TB

CEPHFS

LTS

~ 9,2 PB

HARDWARE COMPONENTS: PROCESSING UNITS

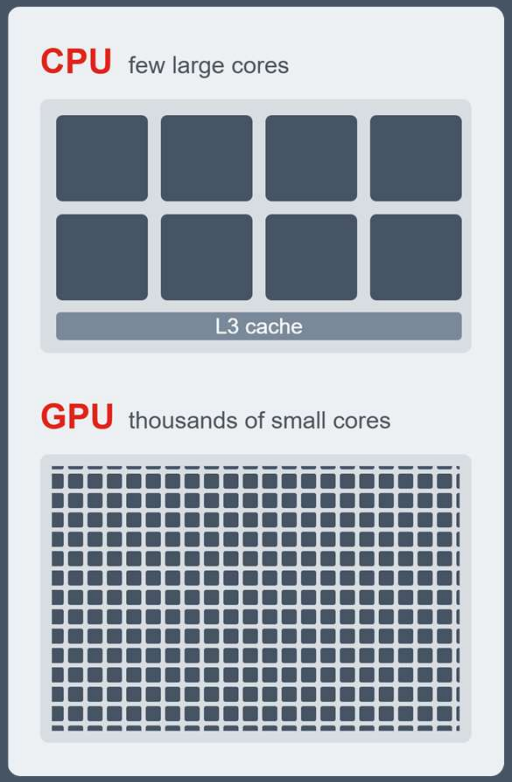


PROCESSORS AND CORES

- Fundamental components of high-performing-computing (HPC) system
- Central processing unit (CPU) primary units executing instructions (the ‘brain” for general instructions)
- Choice of the processor can have significant role for the performance of specific applications
- Cores are the individual processing units within a processor
- A thread is the single sequence of programmed instructions

Image generated by AI

HARDWARE COMPONENTS: GPU AND (OTHER) ACCELERATORS



GPUS AND ACCELERATORS

CPU · latency-optimized

- Few powerful cores (8–192 per socket)
- Built for complex, branching code
- e.g. Ryzen 9 9950X: 16 cores

GPU · throughput-optimized

- Thousands of simple cores
- Same operation on many data at once
- e.g. RTX 5090: 21,760 CUDA cores

Memory bandwidth is often the real bottleneck: RTX 5090 (GDDR7) ≈ 1.8 TB/s vs. dual-channel DDR5-6000 ≈ 96 GB/s. Data-center GPUs use stacked HBM.

TPU
Google's matrix engine for AI training and inference

NPU
Low-power AI inference built into laptops and phones

FPGA
Reconfigurable chips for custom, low-latency pipelines

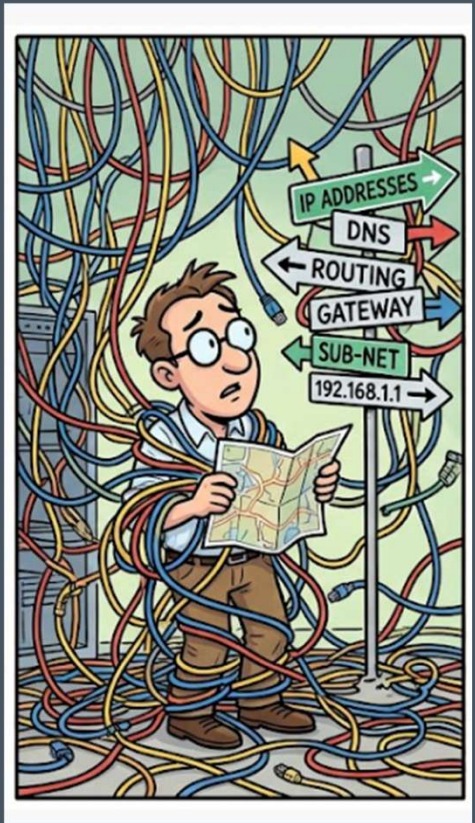
DPU / SmartNIC
Offloads network, storage and security work from the CPU

ORFEO HPC nodes after PNRR

TYPE OF NODE	RAM per node	COREs per node	GPUs per node
10 x THIN intel nodes	768 GB	24	-
2 x FAT intel nodes	1536 GB	36	-
4 x GPU intel nodes	256 GB	24	2 x Nvidia V100 (32GB)
8 x EPYC Amd nodes	512 GB	128	-
2 x DGX Nvidia Station	1024 GB	128	8 x Nvidia A100 (40GB)
13 x GENOA Amd nodes	512 GB	64	-
3 x Dell XE9680	1024 GB	112	8 x Nvidia H100 (80GB)
42 Compute Nodes	**~ 25 TB of RAM**	**2856 cores**	**48 GPUs**



NETWORKING AND COMMUNICATION

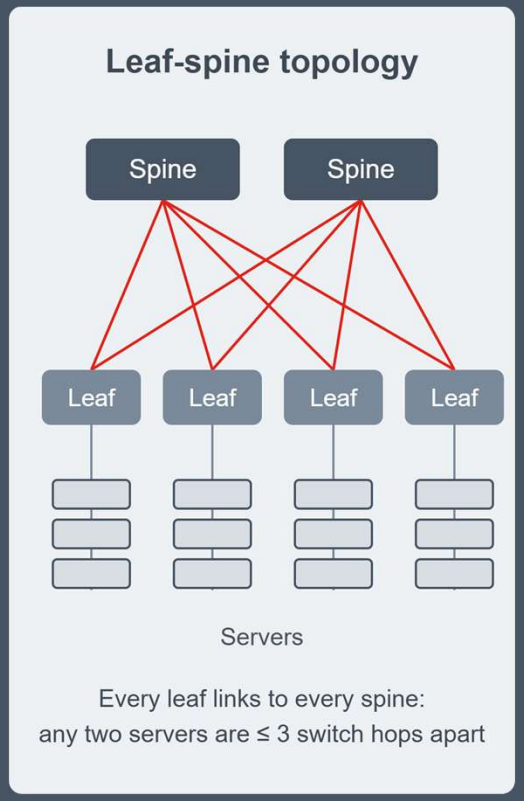


WHY THEY MATTER

- Enable communication between distributed components, internet and other neywork
- Support data sharing, replication, and remote access
- Affect performance, reliability, and scalability of infrastructure
- Critical metrics:
    - Latency: time to begin data transfer
    - Bandwidth: amount of data per unit time

Image generated by AI

NETWORKING AND COMMUNICATION



DATA CENTER INTERCONNECTS

Link	Line rate	≈ Max GB/s	Roughly matches
1 GbE	1 Gb/s	0.125	Less than 1 HDD
10 GbE	10 Gb/s	1.25	2 SATA SSDs
25 GbE	25 Gb/s	3.1	1 NVMe PCIe 3.0
100 GbE	100 Gb/s	12.5	≈ 1 NVMe PCIe 5.0
400 GbE / IB NDR	400 Gb/s	50	≈ 7 NVMe PCIe 4.0

Networks count bits, storage counts bytes: divide Gb/s by 8 to compare a link with the drives behind it.

RDMA / InfiniBand
Memory-to-memory transfers that bypass CPU and OS; standard in HPC and AI clusters

RoCE
RDMA over Ethernet: low latency on standard data center switches

NVMe-oF
Remote SSDs accessed over the network with near-local latency

Theoretical line rates; protocol overhead lowers real throughput by roughly 5–10%.

ORFEO network after PNRR

• HPC Network • I/O Network	100 Gbit & 200 Gbit Infiniband QM8700 + QM9700 Mellanox - 16 + 51.2 Tb/s aggregate throughput
In band Management network	2 x 25Gbit Ethernet link 10x dell S5148F switches: - 3.6 Tbps aggregate throughput - Fully redundant network
Out of band Management network	1Gbit Ethernet

IP Addressing

- A system that assign unique identifiers, IP addresses, to devices connected to a network.
- These serve as primary means for devices to locate and communicate with each other

Protocols

- Standardized rules and conventions that governs how data is trasmitted and received over a network
- Transmission Control Protocol (TCP): ensures reliable, ordered and error-checked delivery of data
- Internet Protocol (IP): routing of data across networks
- HyperText Transfer Protocol (Secure) (HTTP/HTTPS):
  - Used for REST APIs, web-based data services
  - Common in cloud-based and microservices architectures
- File Transfer Protocol (FTP) simple but insecure
- Secure File Transfer Protocol (SFTP): encrypted, uses SSH
  - Used for batch transfers, data ingestion pipelines

Which FAIR principle is strongly related to?

CAP theorem (Brewer theorem 2000, proved by Gilbert and Lynch 2002))

Parenthesis

Consistency-Availability-Partition Tolerance

Trade-offs between three key properties in distributed system

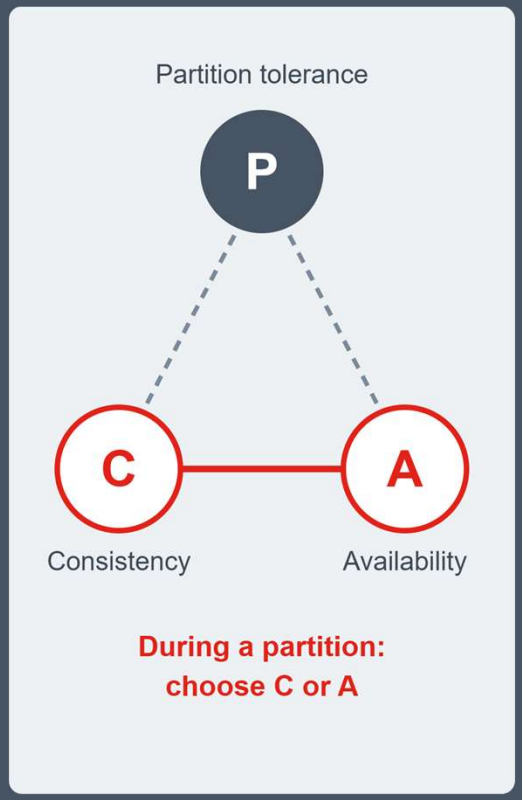
Consistency: all nodes in the system have the same data view

Availability: guartantees that every request receives a response. Even if some nodes fail, system remains operational and responsive

Partition Tolerance: system ability to continue functioning despite network partitions or communication breakdowns between nodes

The CAP theorem states that a distributed system can only guarantee two of these three properties at any given time, not all three simultaneously.

DISTRIBUTED SYSTEMS: CAP THEOREM



(Brewer theorem 2000, proved by Gilbert and Lynch 2002) Parenthesis

Consistency
Every read returns the latest write

Availability
Every request gets a valid response

Partition tolerance
Keeps working when nodes can't talk

The CAP theorem states that a distributed system can only guarantee two of these three properties at any given time, not all three simultaneously.

CAP: THE REAL TRADE-OFF

> **Common misreading:** "pick any two of the three"
> Partitions happen anyway (a cable, a switch, a link between data centers), so P is not optional.

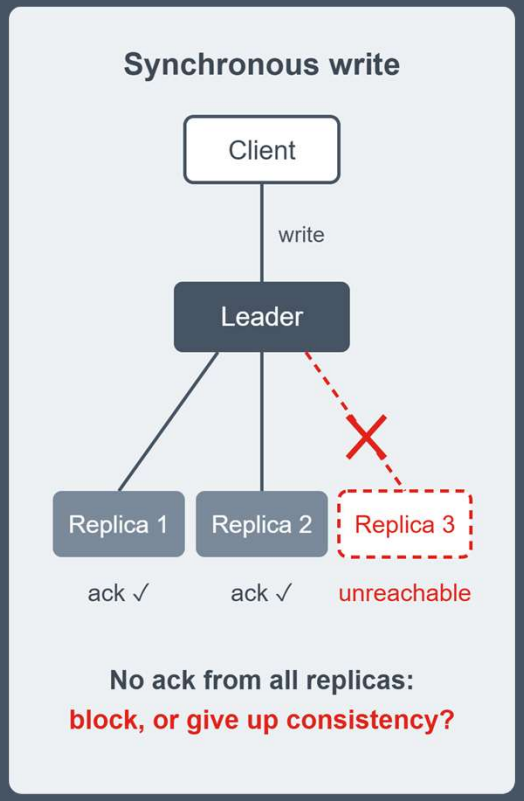
When a partition occurs, you must choose between C and A.

An isolated node that receives a request has only two options:

Choose A
Answer with the data it has, at the risk of returning stale values

Choose C
Refuse or wait until the network is back, so data stays correct

DISTRIBUTED SYSTEMS: CAP THEOREM



INSTANT SYNC IS IMPOSSIBLE

< 1 ms
within a rack

10s of ms
across continents

Every message takes time: no system can update all nodes at the same instant.

> **Synchronous replication:** a write is acknowledged only after **every** replica confirms it. Nobody ever reads stale data, so you get **Consistency**.

Now one node becomes unreachable. The system can no longer collect every confirmation, so it must choose:

Strategy	During the partition	CAP choice
Synchronous replication to all nodes	Writes block or fail until the node is back	CP
Majority quorum (Paxos, Raft)	The majority side keeps working, the minority stops	CP, more fault-tolerant
Asynchronous replication	Every node accepts writes; the sides diverge and must be reconciled later	AP (eventual consistency)

DATA ARCHITECTURE VS DATA INFRASTRUCTURE

WHAT IS DATA ARCHITECTURE?

- the set of rules that defines within an infrastructure how data is gathered, kept, managed, and utilized.
- Includes the toolset, policies, and standards that help in managing the handling of data assets properly.
- the framework that regulates how an organization's IT infrastructure enables its data strategy
- provides a framework for and is a relevant part of data management.

What is Data Architecture? | GeeksforGeeks

COMPONENTS OF DATA ARCHITECTURE

- **Data Models** : data models are abstract representations of data objects and their relationships. They provide a structured framework for organizing data
- **Data Governance**: involves the policies, procedures, and standards that ensure data quality, security, and compliance
- **Data Integration**: Data integration involves combining data from different sources to provide a unified view.
- **Data Storage:** storage solutions are critical for maintaining and managing data.
- **Data Access and Analytics** : Data access and analytics tools enable users to retrieve, analyze, and visualize data.

TYPES OF DATA ARCHITECTURE

- Centralized Data Architecture: In this framework, all data (being stored and managed) are done in a central repository, which might be a data warehouse. It combines data from many sources into one place, attempting to facilitate data management, analysis, and integrity maintenance. A common association of centralized data architecture is with conventional monolithic data infrastructure,.
- Decentralized Data Architecture: In this framework data processing and storage are distributed among many nodes or systems, enabling each domain to handle its own. Data is spread all over different servers or databases.

What is Research Data Management (RDM)?

Data management refers to all aspects of creating, housing, delivering, maintaining, and archiving and preserving data. It is one of the essential areas of responsible conduct of research

WHAT IS DATA INFRASTRUCTURE?

› The set of technologies and processes for collecting, storing, processing, and managing data

WHAT IS DATA ARCHITECTURE?

❑ the set of rules that defines within an infrastructure how data is gathered, kept, managed, and utilized.

ALWAYS THE SAME CONCEPTS?



A BUILDING METAPHOR (FOR A DATA ECOSYSTEM)

Data Infrastructure:

The 'shell' of the system:

- Includes servers, storage, networks, orchestrators
- Provides the physical and technical backbone



shell structure, electrical and plumbing installation

Data Architecture:

The 'essence' inside:

- Includes data models, flows, governance, access logic
- Gives meaning, shape, and function to the infrastructure



Room division/Space planning

Data Management:

The ‘rule' inside:

- Includes all the policy to according to which data are organized
- includes all the facility operations

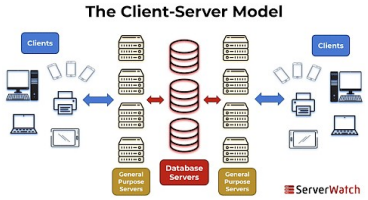


Furniture and space planning, ‘house’ policy

ARCHITECTURAL MODELS

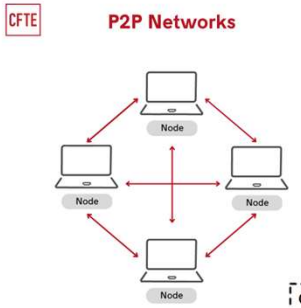
Comparison Overview

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

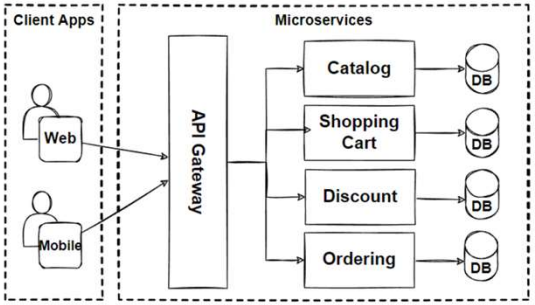


Are connected to the type of data architecture

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



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



COMBINING ARCHITECTURES/MODELS

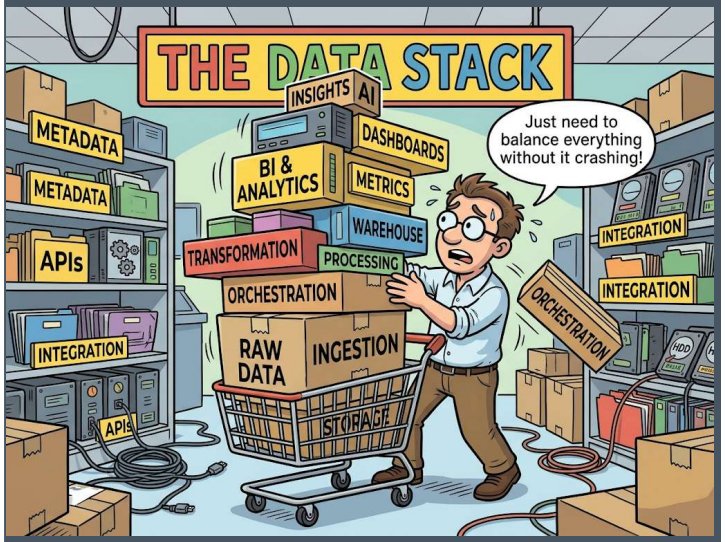
Architectural Model	Only Distributed ?	Typical of Distributed System
Client-Server	NO	NO
P2P	YES	YES
Microservices	NO, BUT ALMOST ALWAYS	YES

Not all the architectures are compatible with all data architectures



A building may become a private house or a B&B, another building may become a bank or a hotel (see Hilton Hotel in TS) but not all the buildings may become a bank!

DATA STACK



WHAT IS A DATA STACK?

- A data stack is a collection of technologies and tools used to collect, store, process, analyze, and visualize data
- It forms the core of modern data infrastructure
- Enables scalable, reliable, and efficient data pipelines
- Supports collaboration across engineering, analytics, and business teams
- Data Stack is the ‘soul’ of Data Architecture

Image generated by AI

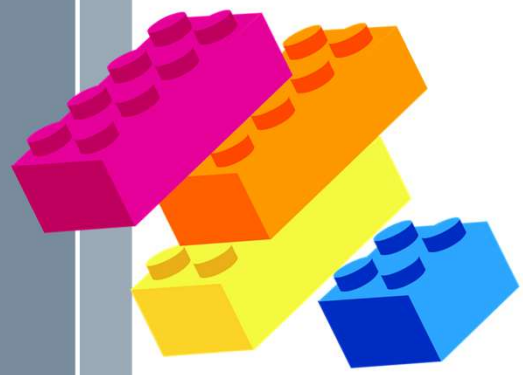
DATA STACK TOOLS

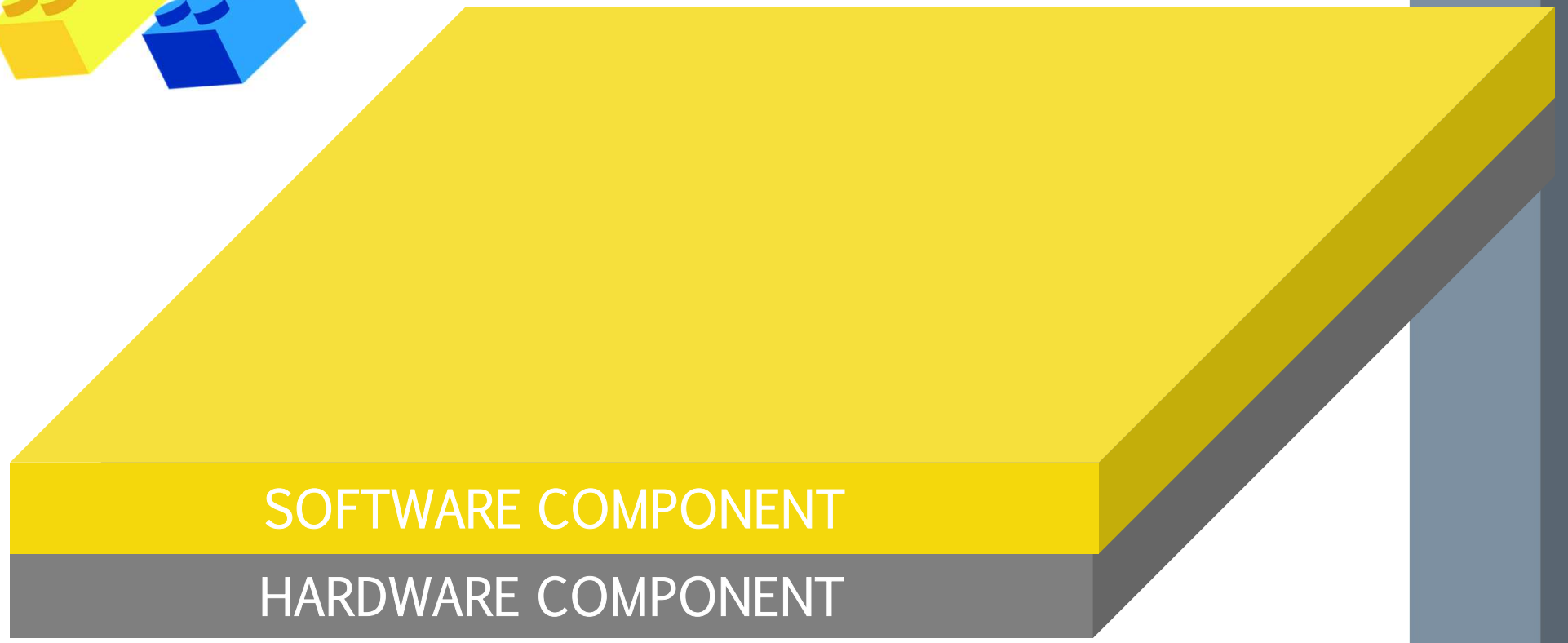
These tools are essential for turning data from 'inedible data' (data that cannot be worked with) to 'edible data' (data that can be worked with).

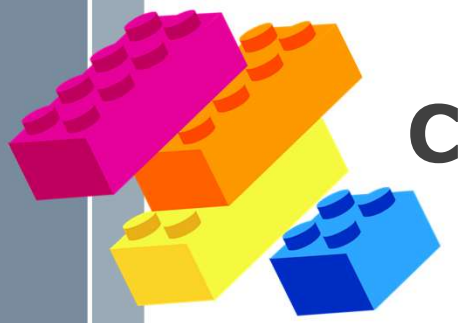
The faster that data can be accessed, prepared, and analyzed, the faster they can be used (decision-making or research)

An effective modern data stack architecture is therefore crucial for any data infrastructure hoping to extract value from their data.

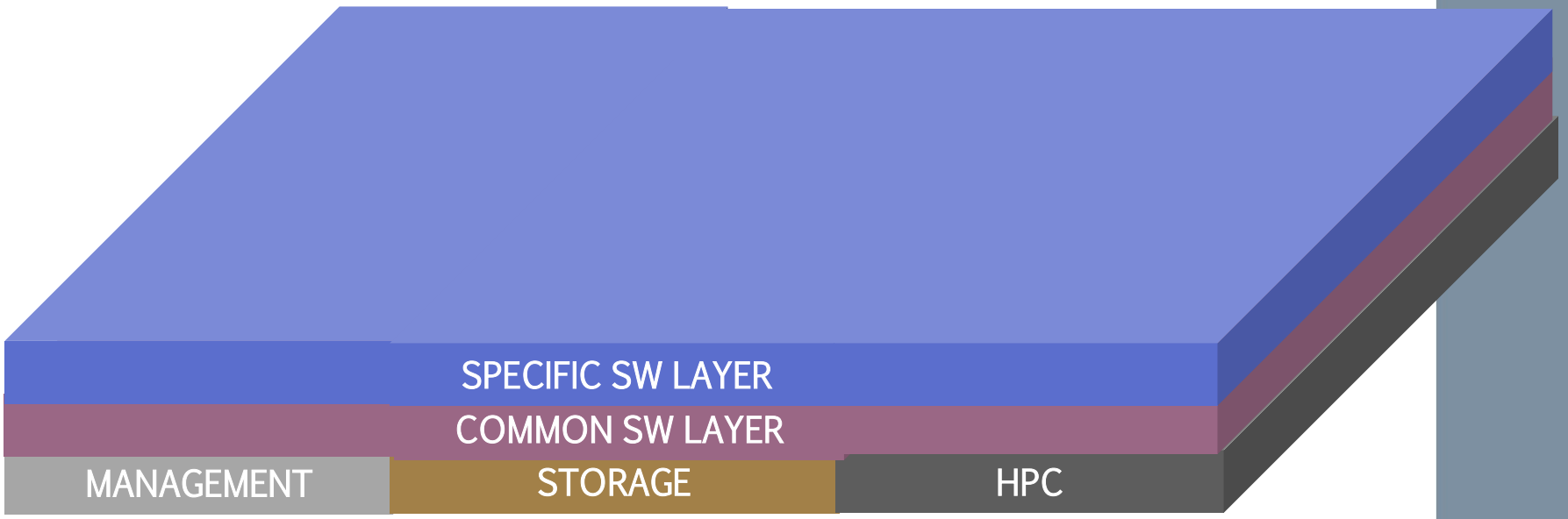
Generic Data Infrastructure







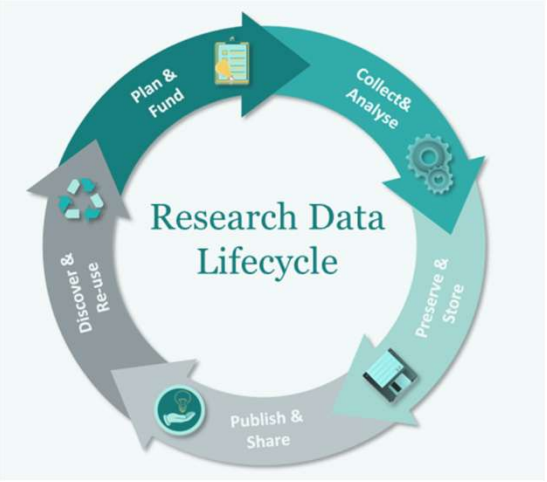
Common + Specific SW Layers



Common Layers in a Modern Data Stack

1. Ingestion: Data collection from various sources
2. Storage: Raw and structured data storage
3. Processing: ETL/ELT and transformation logic
4. Analytics: Query engines and data modeling
5. Visualization: Business Intelligence tools
6. Governance: Data quality, lineage, and security

See de Luca's Lectures



Enterprise vs Scientific Data Stack Layers

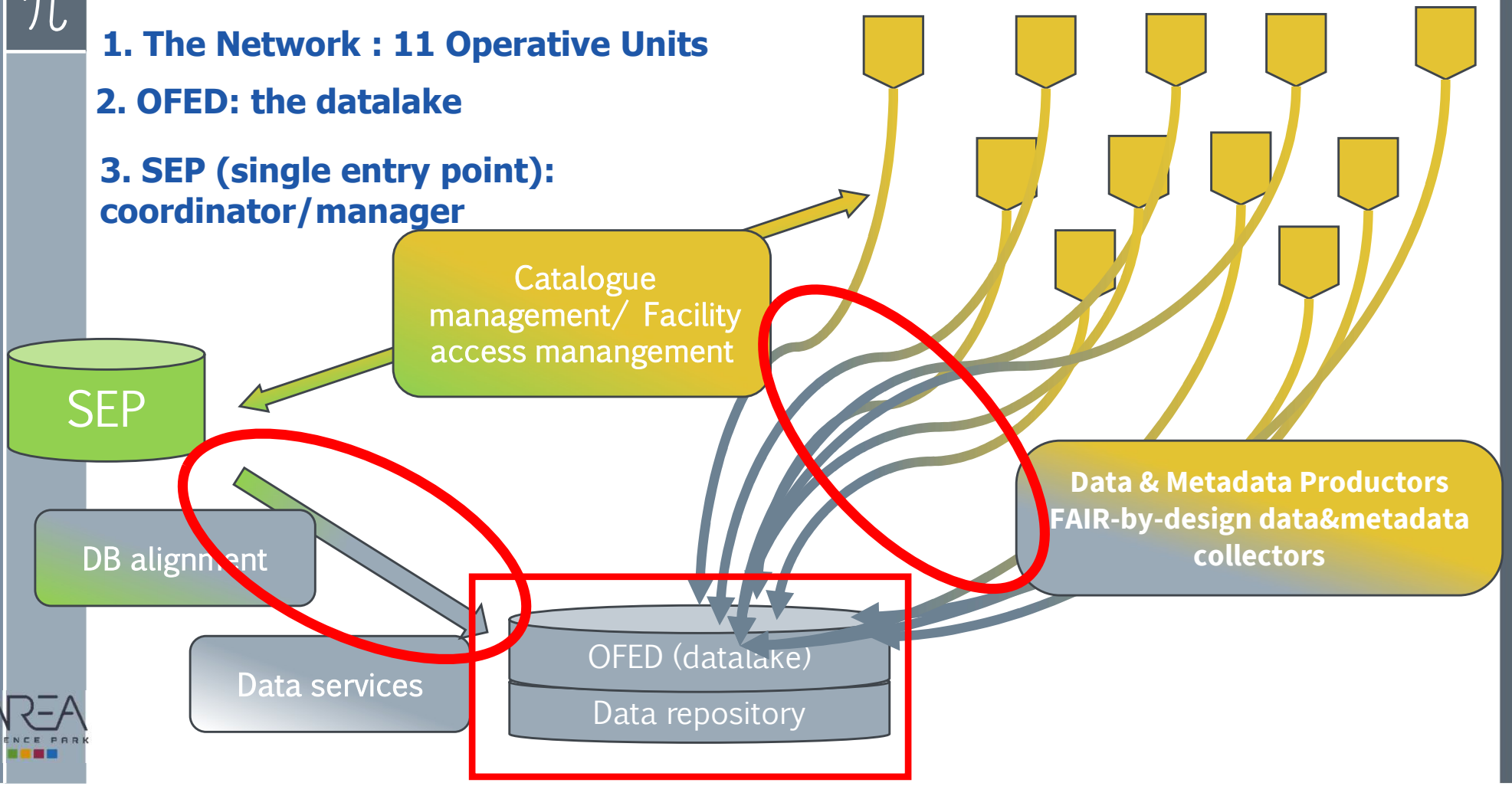
Layer	Enterprise	Research
Ingestion & ETL	ETL/ELT pipelines (Kafka, Airbyte, dbt)	FAIR-by-design acquisition workflows
Storage	Data Lakes, Data Warehouse, S3	Open Source Storage SW
Processing/Analysing	Spark, SQL engines, Jupyter workflows	HPC clusters, Jupyter workflows, scientific libraries (NumPy, SciPy)
Governance & Access	Internal Data Governance	Provenance tracking, persistent identifiers (DOI), open standards
Goal	Business intelligence and performance (enhance profit)	Reproducibility, transparency, data reusability--FAIR principles–publication (enhance scientific discoveries)

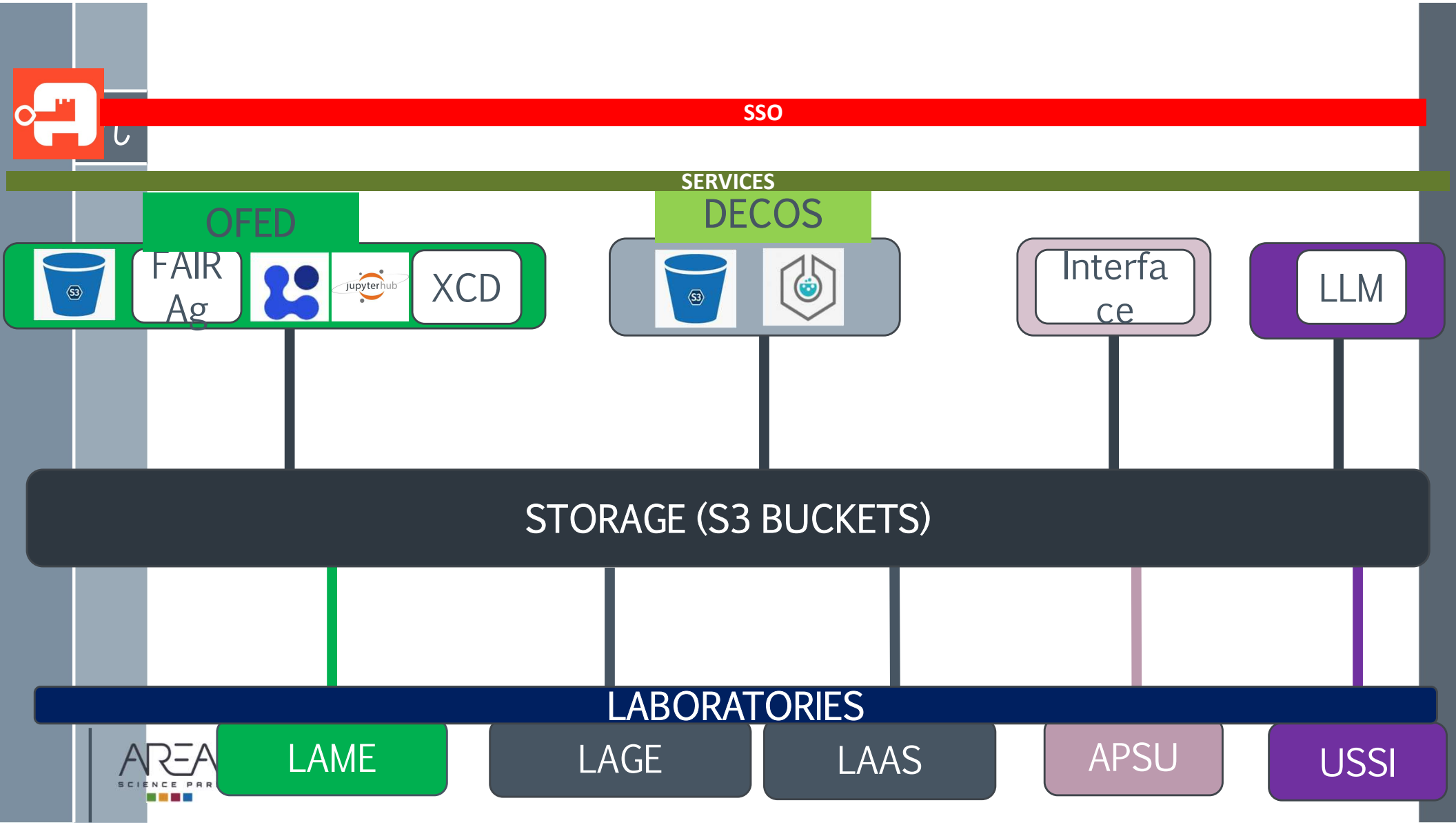
Elements of the Research (Distributed) Digital Infrastructure

1. The Network : 11 Operative Units

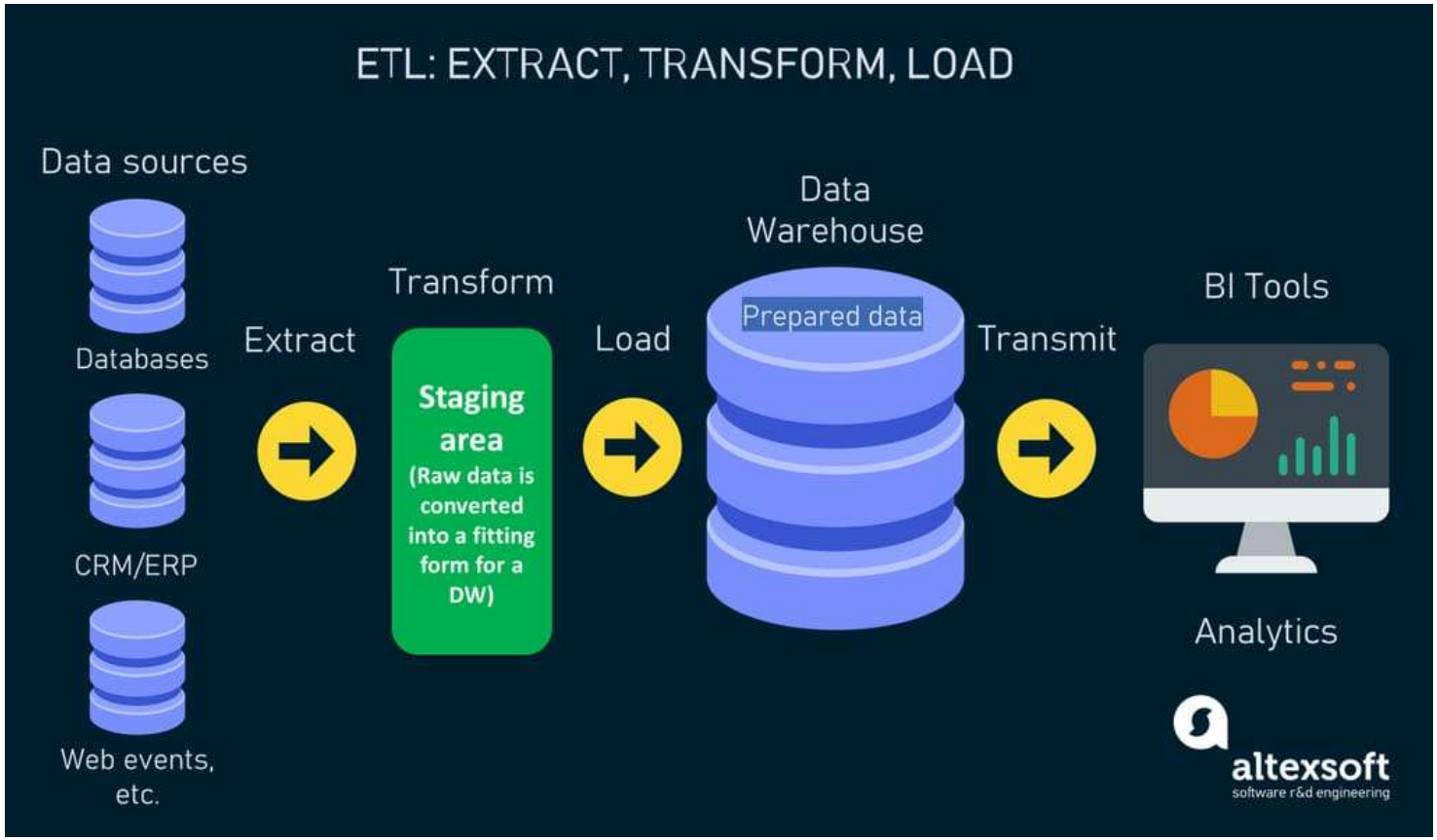
2. OFED: the datalake

3. SEP (single entry point): coordinator/manager




SSO
SERVICES
OFED
FAIR Ag
jupyterhub
XCD
DECOS
Interfa ce
LLM
STORAGE (S3 BUCKETS)
LABORATORIES
LAME
LAGE
LAAS
APSU
USSI

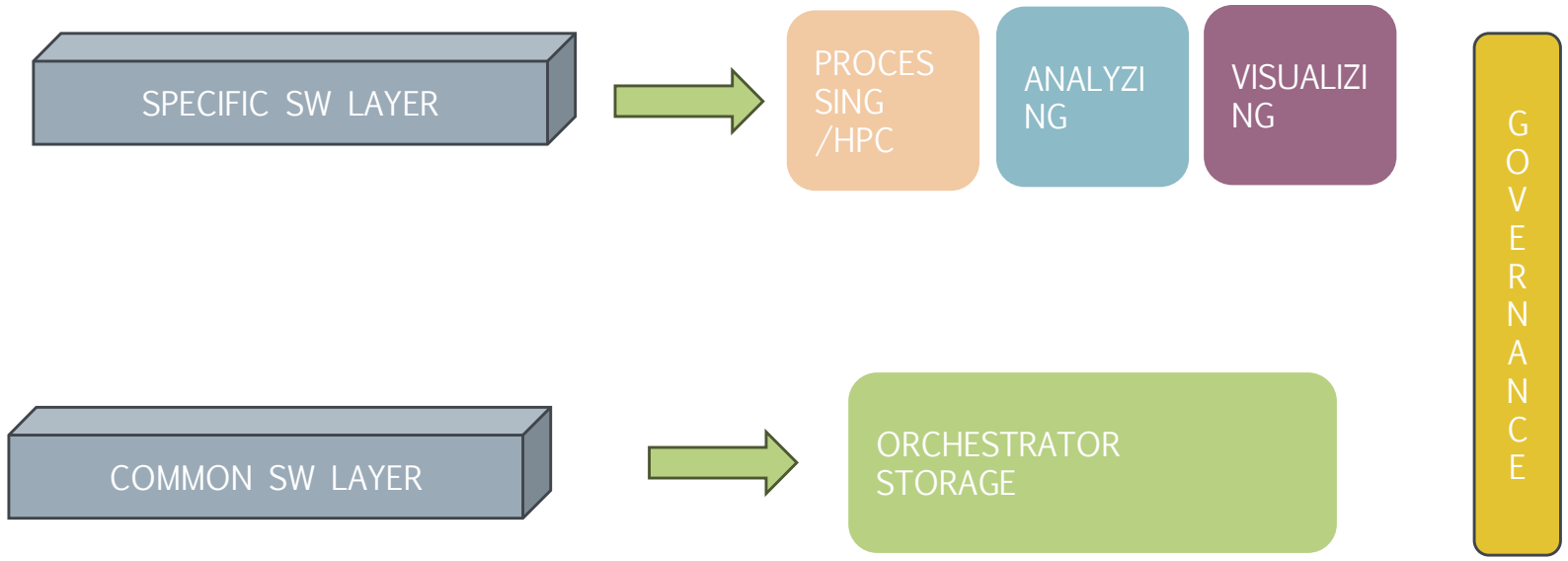
Look at the similarities...



Traditional vs Modern Data Stack

› Traditional:
- On-prem databases
- Batch ETL
- Static reports

› Modern:
- Cloud data lakes and warehouses
- Streaming ingestion and ELT
- Interactive BI dashboards
- Integrated monitoring and governance


SPECIFIC SW LAYER
PROCES SING /HPC
ANALYZI NG
VISUALIZI NG
COMMON SW LAYER
ORCHESTRATOR STORAGE
GOVERNANCE

COMMON SOFTWARE LAYER



Container: what they are

- A container is a standard package of software that bundles an application's code together with the related configuration files and libraries, and with the dependencies required for the app to run. This allows developers and IT pros to deploy applications seamlessly across environments.
- Containers solve the problem of an application failing to run correctly when moved from one environment to another. They ensure consistency across environments.
- Containers provide a lightweight, immutable infrastructure for application packaging and deployment. An application or service, its dependencies, and its configuration are packaged together as a container image. It is used for reproducible data pipelines.
- This way, containers enable developers and IT professionals to deploy applications across environments with little or no modification.

What is a container? | Microsoft Azure

What is a Container Orchestrator?

See Pasianotto's Lectures

Container orchestration is the process of automating the deployment, management, scaling, and networking of containers throughout their lifecycle, making it possible to deploy software consistently across many different environments at scale.

Container orchestration makes it possible to build **continuous integration and continuous deployment (CI/CD) pipelines**, which improve software delivery throughout the software development lifecycle via automation. Container orchestration also connects to a **DevOps** approach, which aims to accelerate the processes of bringing an idea from development to deployment.

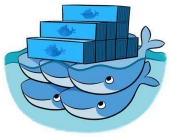
Running containers at scale requires orchestration and management of distributed, containerized applications via an orchestration platform such as Kubernetes.

Example of Container Orchestrator

Kubernetes: **a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. Originally developed by Google**



Docker Swarm: a native clustering and orchestration tool for Docker containers. It enables you to manage a cluster of Docker hosts (also called nodes) as a single virtual host.



Apache Mesos: is built using the same principles as the Linux kernel, provides applications with API's for resource management. Native support for launching containers with Docker and AppC images.



What is a Data Orchestrator?

- A data orchestrator is a tool used to automate, schedule, and manage data workflows
- Helps coordinate multiple tasks across ingestion, transformation, and storage
- Commonly used in both batch and streaming data pipelines

- Ensures dependencies are respected and failures are tracked
- Supports alerting, retries, and monitoring for robust data operations

Few examples of Data Orchestrator

General purpose Orchestrator:

Apache Airflow is an open-source platform for developing, scheduling, and monitoring batch-oriented workflows in Python code strong integration with cloud tool



Prefect is an **open-source orchestration engine** that turns your Python functions into production-grade data pipelines with minimal friction.



ML and Kubernetes native Orchestrator:

Argo Workflows is an open source container-native workflow engine for orchestrating parallel jobs on Kubernetes. Define workflows where each step is a container.



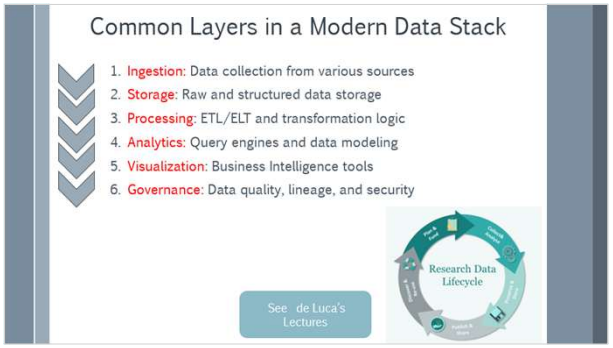
Kubeflow an open-source project that aims to make it easy to deploy and manage ML workflows on Kubernetes. This platform provides a set of tools to develop and maintain the machine learning lifecycle on Kubernetes.



SPECIFIC SOFTWARE LAYER



Layer 1: Data Ingestion



Tools that collect and bring data into the platform

- Real-time and batch ingestion
- Examples: Apache Kafka, Fivetran, Airbyte

Notice that for FAIR-by-design data pipelines, data ingestion depends on instrumentations (they may expose API or it is necessary realizing ad hoc solutions). In general we have custom solutions.

Layer 2: Data Storage (I/III)

Technically it belongs to the common sw layer, but we preferred privileging the data stack horizontal conceptual layers

- Centralized location for raw and structured data
- Durable, scalable, and secure

File Storage (file= finite-length stream of bytes):

The FS storage files into directory tree. At each file/directory are associated metadata and when we look for a file in a given path, starting from the root '/' the FS looks sat metadata at each hierarchy level.

Object Storage (object=a specialized file-like construct, any type of file):

an object store is a key-value store for immutable data objects. We loose writing flexibility (no append operation, we must rewrite) but they allow data storage across massive disk clusters (support parallel stream writes and reads). There is no hierarchical structure (the 'folder-tree' is fictitious)

Layer 2: Data Storage (II/III)

Data Warehouse:

central data hub used for reported and analysis. Data is highly formatted and structured for analytics use cases

Cloud Data Warehouse:

Amazon redshifts kicked off such revolution. Data is hosted in object storage allowing virtually limitless storage.

Data Marts: subset of data warehouse designed to serve departments

Data Lake:

emerged in big data era, the idea is dumping data without rigid structure (any size, any tipe). Data lake 1.0 started with HDFS. Processing data is challenging, even common operation in SQL.

Layer 2: Data Storage (III/III)

See Brancaleon's Lectures next week

What Is a Database? From Tables to Bits

Logically a set of tables; physically a set of files made of fixed-size pages of bytes.

LOGICAL

Logical model
tables, rows, columns, SQL

DBMS engine
parser, optimizer, executor, transactions

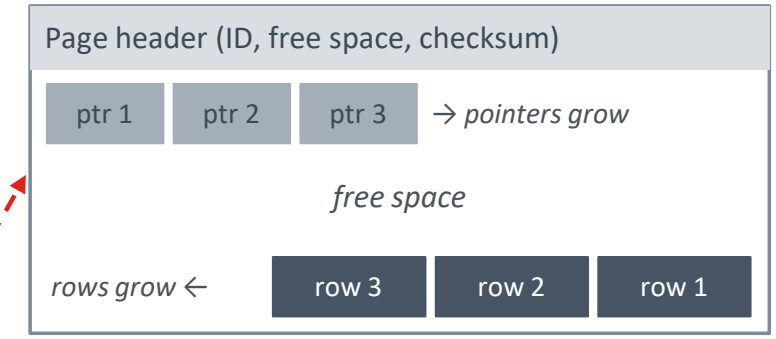
Storage engine
pages, B-tree indexes, buffer pool, log (WAL)

Files (OS file system)
data, index and log files

Block device
4 KB blocks on SSD/HDD → physical bits

PHYSICAL

Inside one page (8 KB in PostgreSQL)



One row as bytes: (id = 42, name = 'ANNA')



42 as 4-byte integer

"Anna" as UTF-8 text (hex)

Amazon S3

- Cloud object storage from AWS
- Highly available and durable
- Ideal for raw data, backups, data lake storage
- Pay-as-you-go model



MinIO

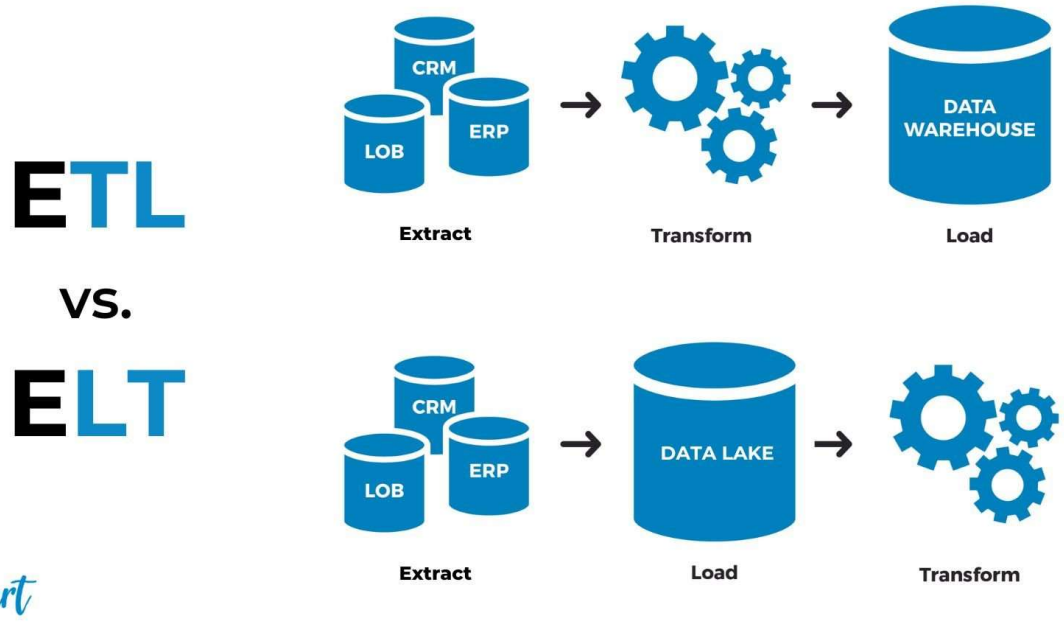


- Fully compatible with Amazon S3 API
- Kubernetes Native (in ORFEO WAS delpoyed as a Kubernetes POD)-> then we moved to cheph RDG
- High Performance (speed!)
- distributed under the GNU AGPL v3 license, supported by a company and with a large community

Layer 3: Data Processing

See Piompioni's Lectures next week

Transforming, cleaning, and modeling data

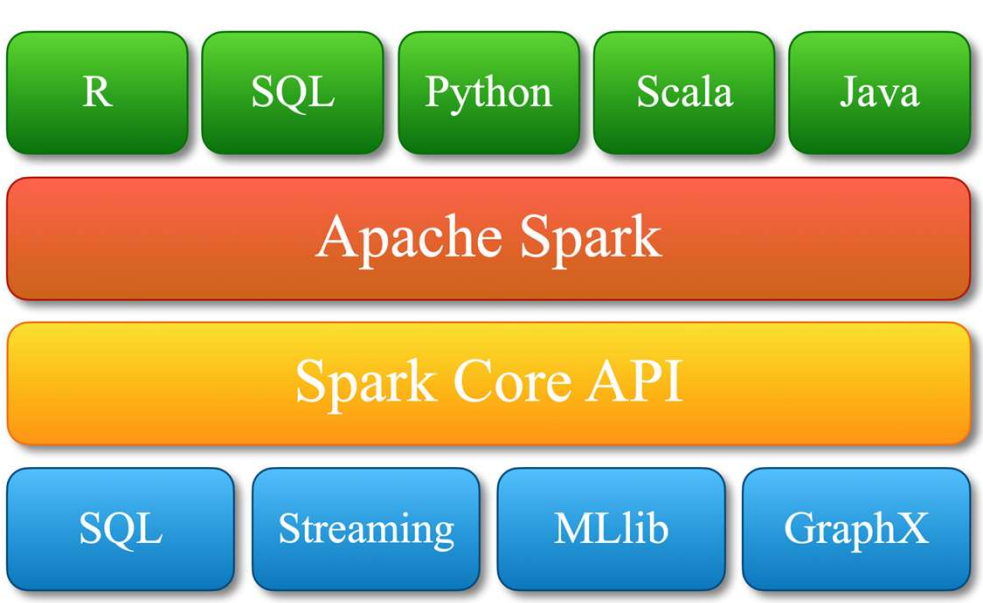


Apache Spark

Apache Spark™ is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters.

Unify the processing of data in batches and real-time streaming,

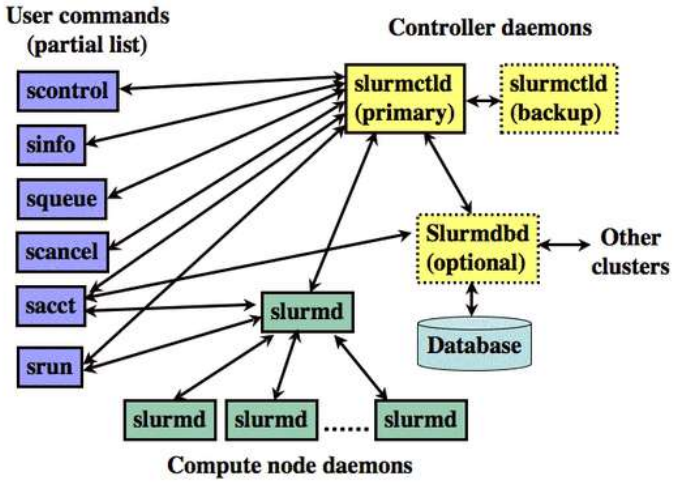
Perform Exploratory Data Analysis (EDA) on petabyte-scale data without having to resort to downsampling



Apache Spark™ - Unified Engine for large-scale data analytics

Slurm

Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system l Linux clusters



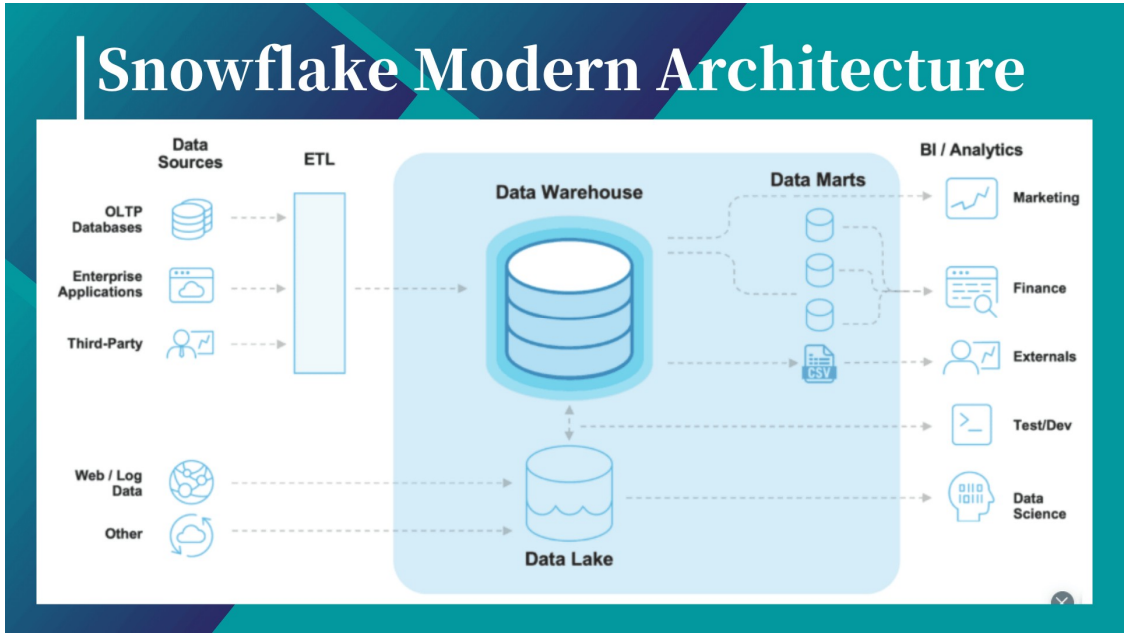
Other workload manager: Stonebranch (free) Red Hat OpenShift (pay)

Layer 4: Data Analytics

- Query engines
- Interactive exploration

Snowflake

Snowflake is a cloud-based data warehousing platform known for its scalability, flexibility, and ease of use. It is designed to handle large volumes of data efficiently and cost-effectively by leveraging the benefits of cloud computing platforms. It is built specifically for the cloud, allowing it to scale resources up or down based on demand.



Layer 5: Data Visualization and Business Intelligence

- Reporting, extract final value from the big data
- May be descriptive, diagnostic, predictive

Tableau

powerful, secure, and flexible end-to-end analytics platform.
Both in desktop and cloud version



Power BI

Collection of sw, services, connectors created by Microsoft for BI.
Compatible (of course!) with all Microsoft services and storage system (Azure, Access,..)



Notice that in a research context Layers 3-4-5 'collapse' in 1 or 2 layers according to the domain, and are done by the same researcher/group.

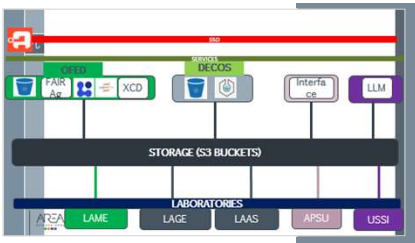
Layer 6: Data Governance

- Data governance tools are essential for managing the availability, integrity, and confidentiality of enterprise data.
- These tools help organizations ensure that data is handled consistently and not misused, addressing data privacy and security issues effectively[1].

Key Data Governance Tools

- Alation Data Catalog: it offers a collaborative data catalog with automated metadata ingestion.
- Ataccama: A Platform-as-a-Service for self-driven data management and governance, uses AI-driven automated capabilities to analyze data quality and classify data
- Collibra: it automates data operations and keeps cross-functional teams on the same page. It provides features like natural language search, data governance automation, and data stewardship

Identity Providers (IdP)



What it is

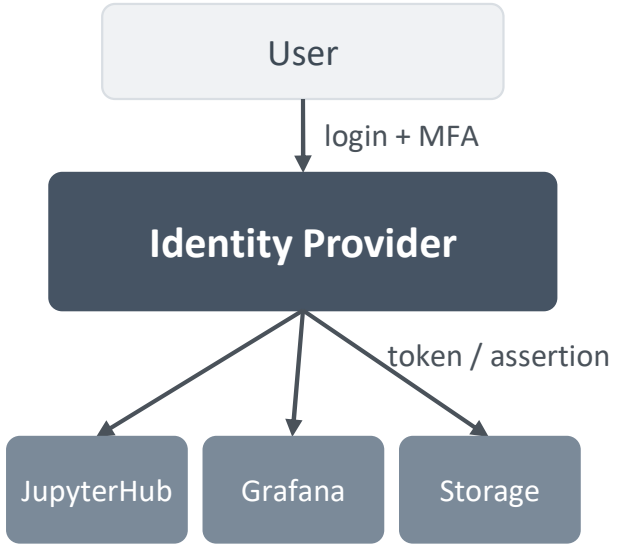
Manages and verifies user identities: the central authentication authority for applications, systems and networks

What users get

Single sign-on (SSO): log in once, reach every connected app. Multi-factor authentication (MFA) on top

How apps trust it

Standard protocols: SAML 2.0, OpenID Connect (built on OAuth 2.0), LDAP, Kerberos



one login, many applications

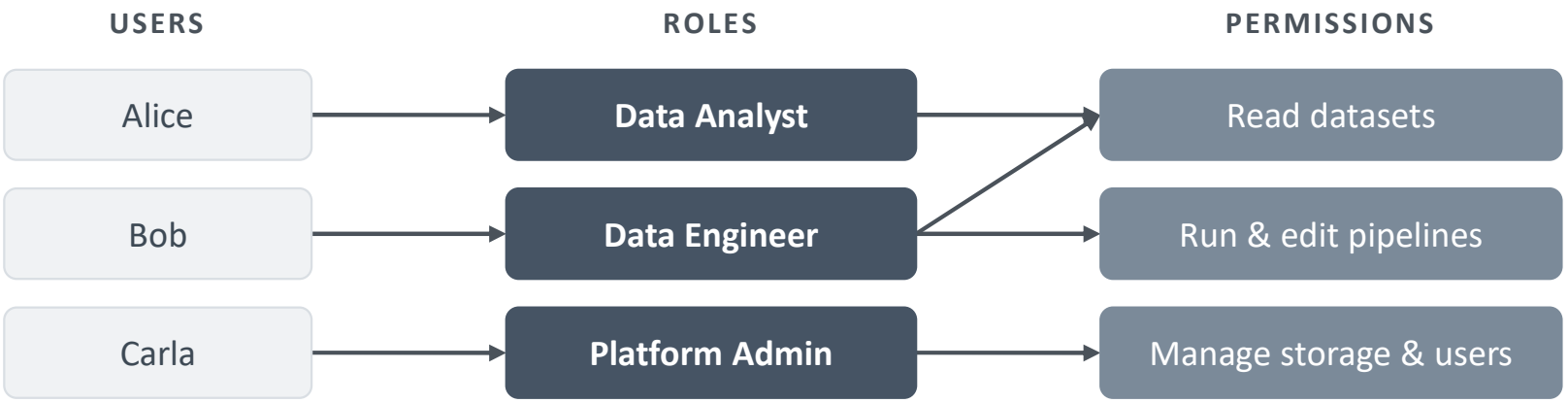
Authentication (AuthN)
Who are you? → the IdP

Authorization (AuthZ)
What can you do? → roles, e.g. RBAC

Source: "What Is an Identity Provider (IdP) and How Does It Work?" – Security Boulevard

Role-Based Access Control (RBAC)

Permissions go to roles, users get roles: never permissions directly to users.



Least privilege
Each role gets only what its job needs

Easy on/offboarding
Change one role, not dozens of permissions

Auditable
"Who can do what" is readable from the roles

In practice: the IdP stores roles or groups and sends them as claims in the token; each application enforces them. A variant, ABAC, decides on attributes (department, time, location).

Identity Provider Examples

	FreeIPA	Keycloak	Authentik
Backed by	Red Hat (upstream of RHEL IdM)	CNCF project; Red Hat build available	Authentik Security (open core)
Focus	Identity for Linux hosts and users	SSO and IAM for web apps and APIs	All-in-one IdP with access portal
Protocols	LDAP (389 DS), Kerberos, DNS, CA	OIDC, OAuth 2.0, SAML 2.0; LDAP/AD federation	OIDC, OAuth 2.0, SAML; LDAP, RADIUS, proxy
Strengths	Kerberos SSO, sudo and host-based access rules	Mature, extensible; social and external IdP brokering	Modern UI, visual flows, self-service
RBAC	Admin roles and privileges, HBAC rules	Realm and client roles, composite roles, groups	Groups and policies bound to applications
Best for	Linux servers, HPC clusters	Enterprise apps, microservices	Self-hosted, cloud-native setups

Common pattern: FreeIPA as the user directory, Keycloak federating it via LDAP to give web apps SSO over OIDC/SAML.

Groups vs Roles + Attributes

Scenario: User A manage the buckets; 3 users each need access to their own bucket. Later, a new bucket D must be readable by A, user1 and user2.

Traditional: Linux groups	
`grp_bucket_a`	A, user1
`grp_bucket_b`	A, user2
`grp_bucket_c`	A, user3
`grp_bucket_d`	A, user1, user2

New bucket D → yet another group

IdP (e.g. Keycloak): roles + attributes

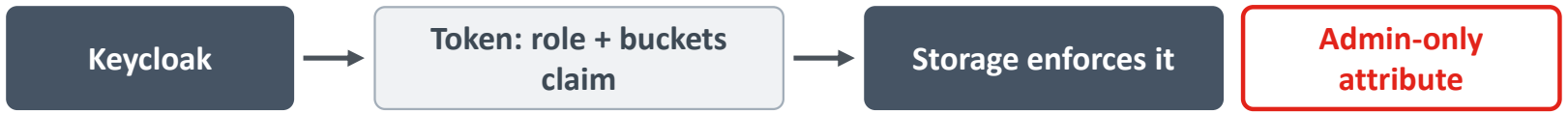
`global_read` A

`restricted_read` user1–3

user1	buckets:	`bucket-a`	`bucket-d`
user2	buckets:	`bucket-b`	`bucket-d`
user3	buckets:	`bucket-c`	

New bucket D → edit two attributes

Roles say what kind of access, attributes say where (RBAC + ABAC)



AREA SCIENCE PARK

SISSA

OGS Istituto Nazionale di Oceanografia e di Geofisica Sperimentale

ARCTUR

REGIONE AUTONOMA FRIULI VENEZIA GIULIA

E-ARGO

Thank you