Ministry of Higher Education and Scientific Research
University of Mohamed Boudiaf, Msila

Faculty of Mathematics and Computer Science
Department of Computer Science

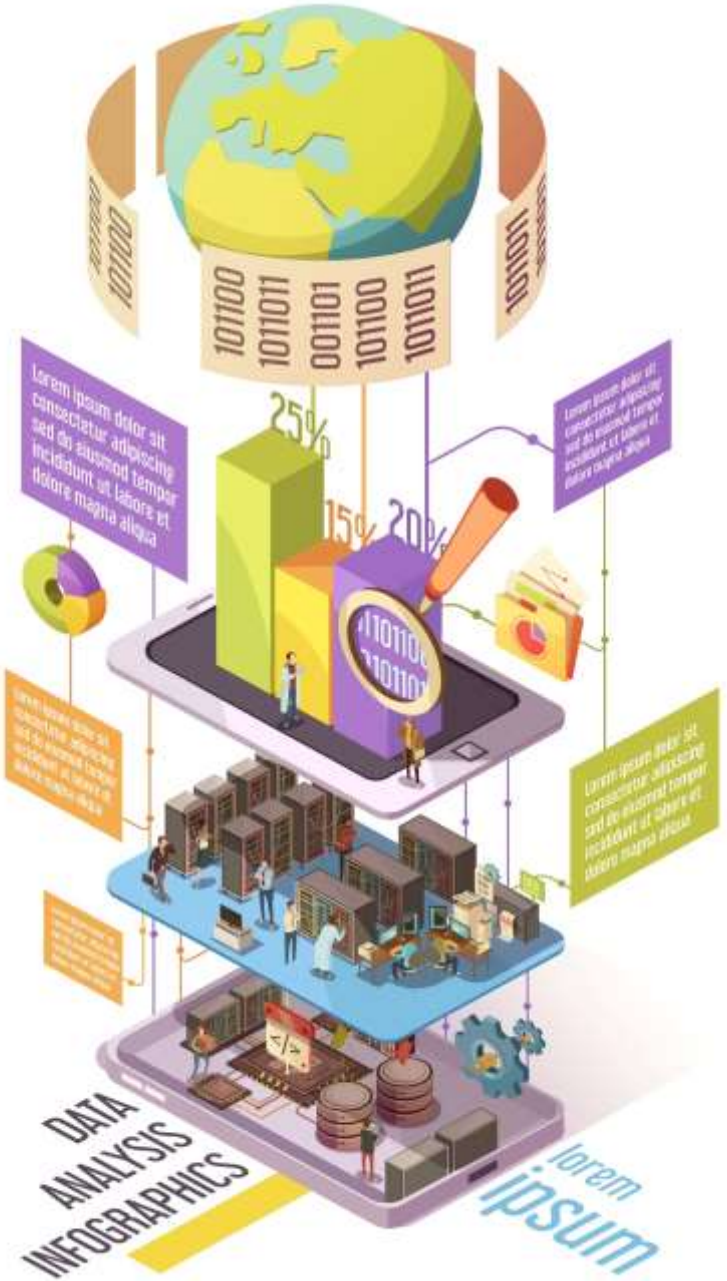
Mater 1

Big Data Analytics

Dr. N.CHALABI

October 2025

Big Data Architecture & Ecosystem



In the 21st century, data is being generated at a rate faster than any other human-produced commodity in history. Yet, raw data is only potential — it must be processed, organized, and analyzed to yield insight.

Why Do We Need New Architectures

Big Data Architecture = framework to collect, store, process & analyze huge, diverse, and fast data.

Data systems evolved:

- 1960s → Mainframes
- 1980s → Relational Databases
- Today → Distributed & Cloud Systems

Why traditional systems fail:

⊘ Single machine = limited storage & processing

⊘ Fixed schemas don't fit unstructured data

⊘ Businesses need **real-time analytics**, not static reports

✓ **New architectures** → designed for **scale, flexibility, and speed**

What Do We Mean by "Ecosystem"

- In tech, an **ecosystem** = a set of tools, frameworks, and systems that work **together** to handle data.
- Think of it like a **city**:
  - 🏢 **Storage systems** = warehouses/archives (where goods = data are kept)
  - 📚 **Databases** = libraries/banks (organize goods in special ways)
  - 🚇 **Processing frameworks** = transport networks (move goods quickly)

👉 No single tool does everything.

👉 Together, they form the **Big Data ecosystem** — designed for **scale, reliability, and efficiency.**

STORAGE SYSTEMS

WAREHOUSES / ARCHIVES
(Where goods = data are kept)

DATABASES

LIBRARIES / BANKS
(Organize goods in special ways)

PROCESSING FRAMEWORKS

TRANSPORT NETWORKS
Move goods quickly

Historical Evolution of Big Data Architectures

Early Systems (1970s–80s)

- Mainframes + central databases
- **Vertical scaling**: faster CPU, more memory, bigger storage (on one machine)

Limits of Vertical Scaling (2000s)

- 📈 **Volume**: Google's crawler collected billions of pages (too big for one DB)
- ⚡ **Velocity**: Real-time search updates needed faster tools than batch systems
- 🌐 **Variety**: HTML, images, PDFs, video → relational DBs couldn't handle it

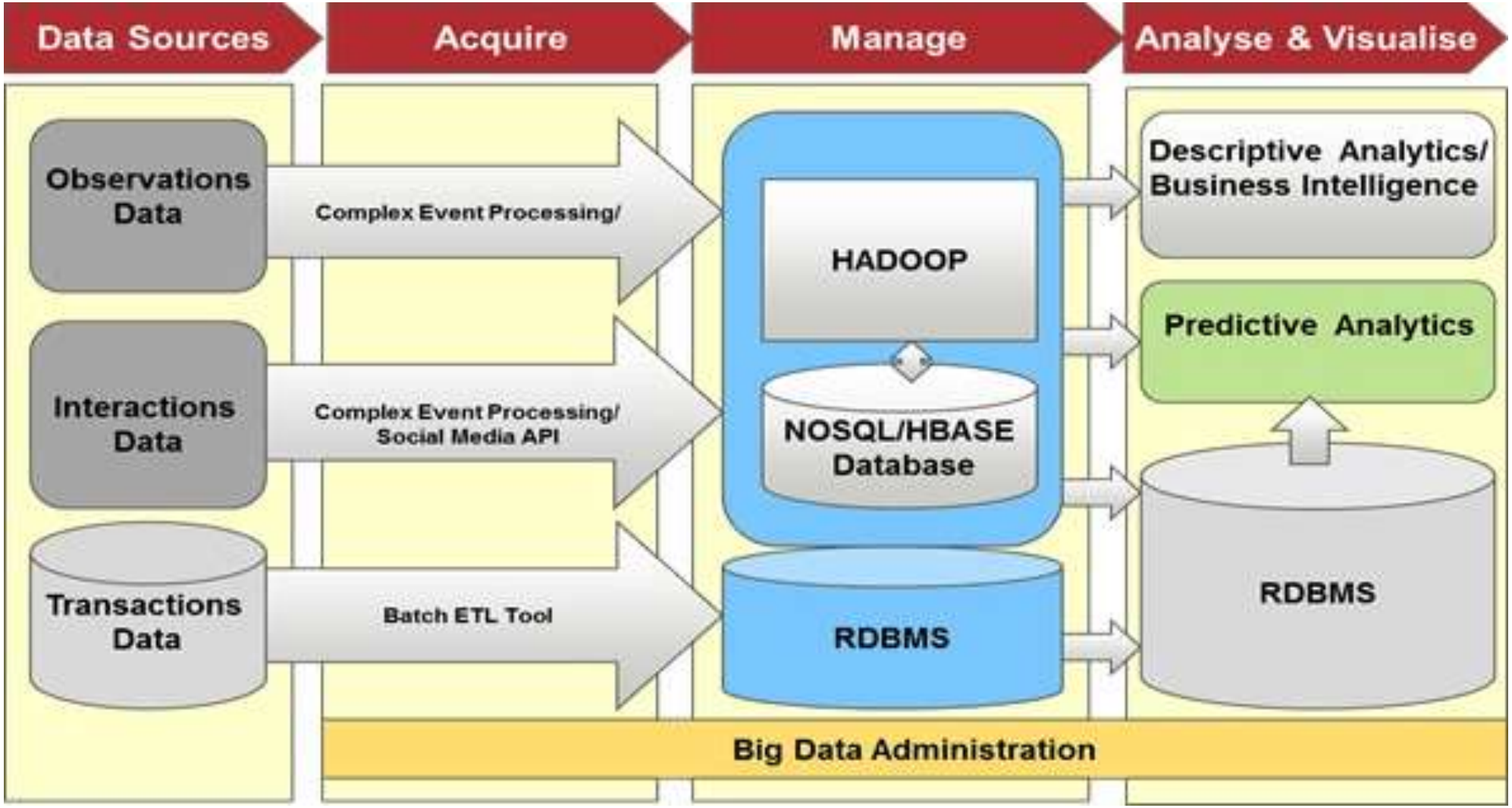
The Breakthrough (2003–2004)

- 📄 *Google File System (GFS)* → scalable distributed storage
- 📄 *MapReduce* → parallel processing model

Inspired **Hadoop**:

- HDFS (storage) + MapReduce (batch processing)
- Made scalable Big Data tools accessible to everyone

Layers of Big Data Architecture



A modern Big Data system works like a **multi-lane highway**.
Each **layer** has a specific role, moving data from **sources → insights**.

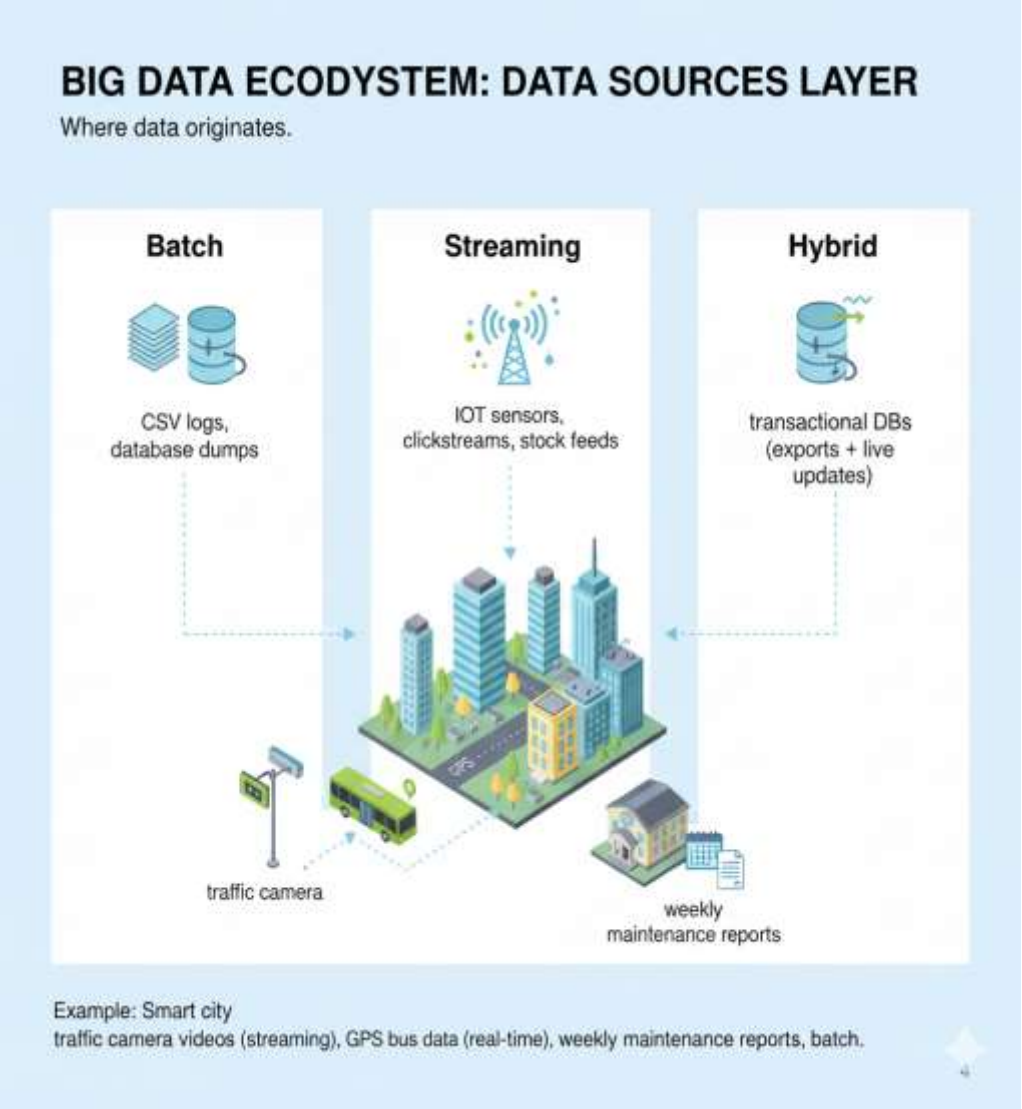
1. Data Sources Layer

Where data originates.

Types of sources:

- **Batch** → CSV logs, database dumps
- **Streaming** → IoT sensors, clickstreams, stock feeds
- **Hybrid** → transactional DBs (exports + live updates)

Example: Smart city → traffic camera videos (streaming), GPS bus data (real-time), weekly maintenance reports (batch).



2. Ingestion / Acquire Layer

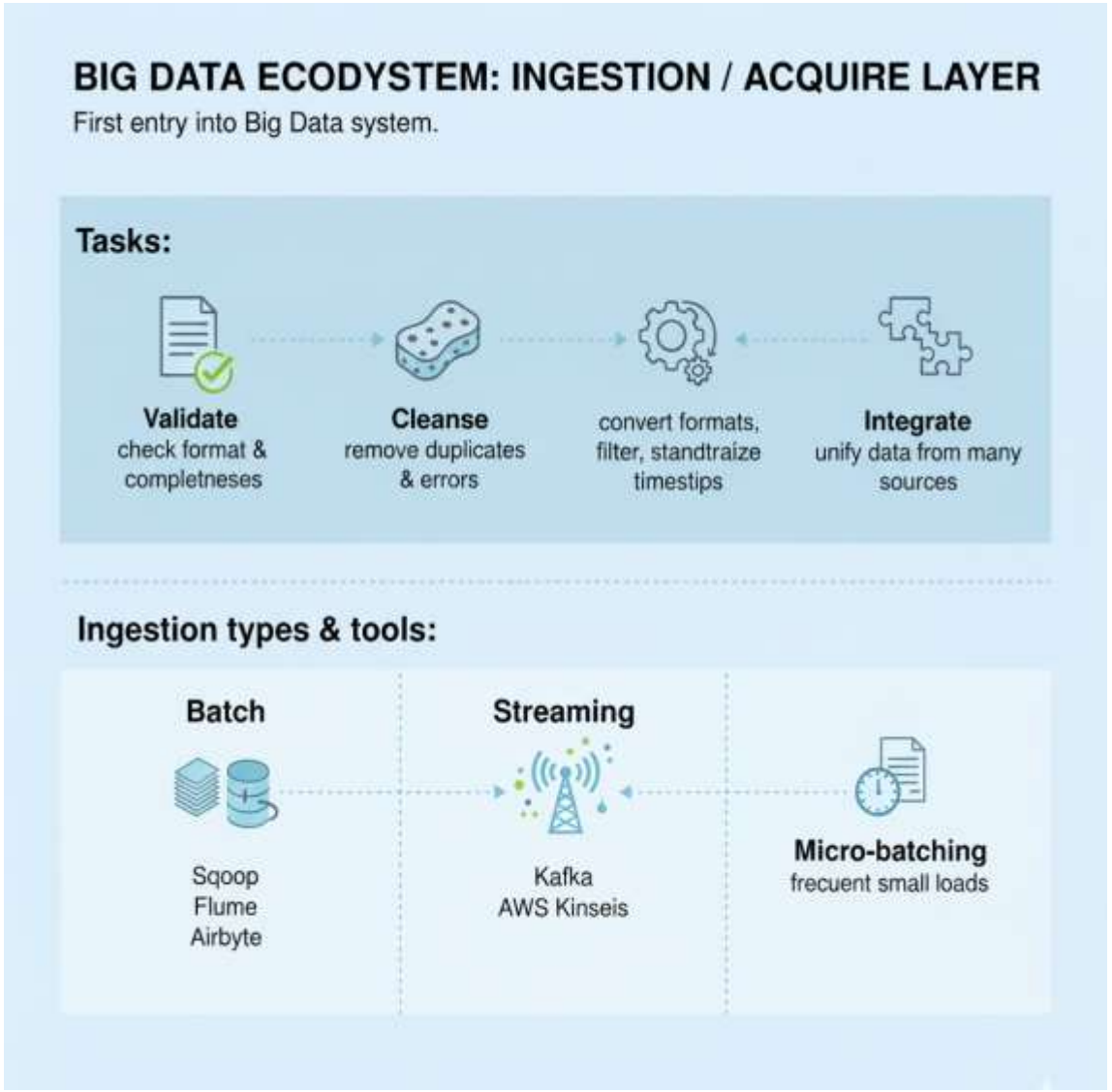
First entry into the Big Data system.

Tasks:

- Validate → check format & completeness
- Cleanse → remove duplicates & errors
- Transform → convert formats, filter, standardize timestamps
- Integrate → unify data from many sources

Ingestion types:

- **Batch** (large chunks at scheduled times) – tools: Apache Sqoop, Flume, Airbyte
- **Streaming** (continuous, real-time flow) – tools: Apache Kafka, AWS Kinesis
- **Micro-batching** (small, frequent batches for near real-time)



3. Storage Layer

The storage layer is where all collected data is **kept safely** so it can be used later for **analysis, reports, or machine learning**. It must be able to:

- Store **huge amounts of data**
- Work across **many machines** (scalable)
- Give **fast access** when needed

◆ a) Distributed File Systems

Work like a **library split across many buildings**. If one building fails, books (data) are still safe in others.

Examples: HDFS, Amazon S3, Azure Data Lake, Google Cloud Storage

Used for: storing very large datasets (raw or processed files)

Used for: media files, backups, and very large files

◆ b) NoSQL Databases

Flexible databases — no fixed "table rules" like traditional databases.

Types:

Column-family (HBase, Cassandra) → like spreadsheets with millions of columns, good for analytics

Document (MongoDB) → stores JSON-like files, easy to change structure

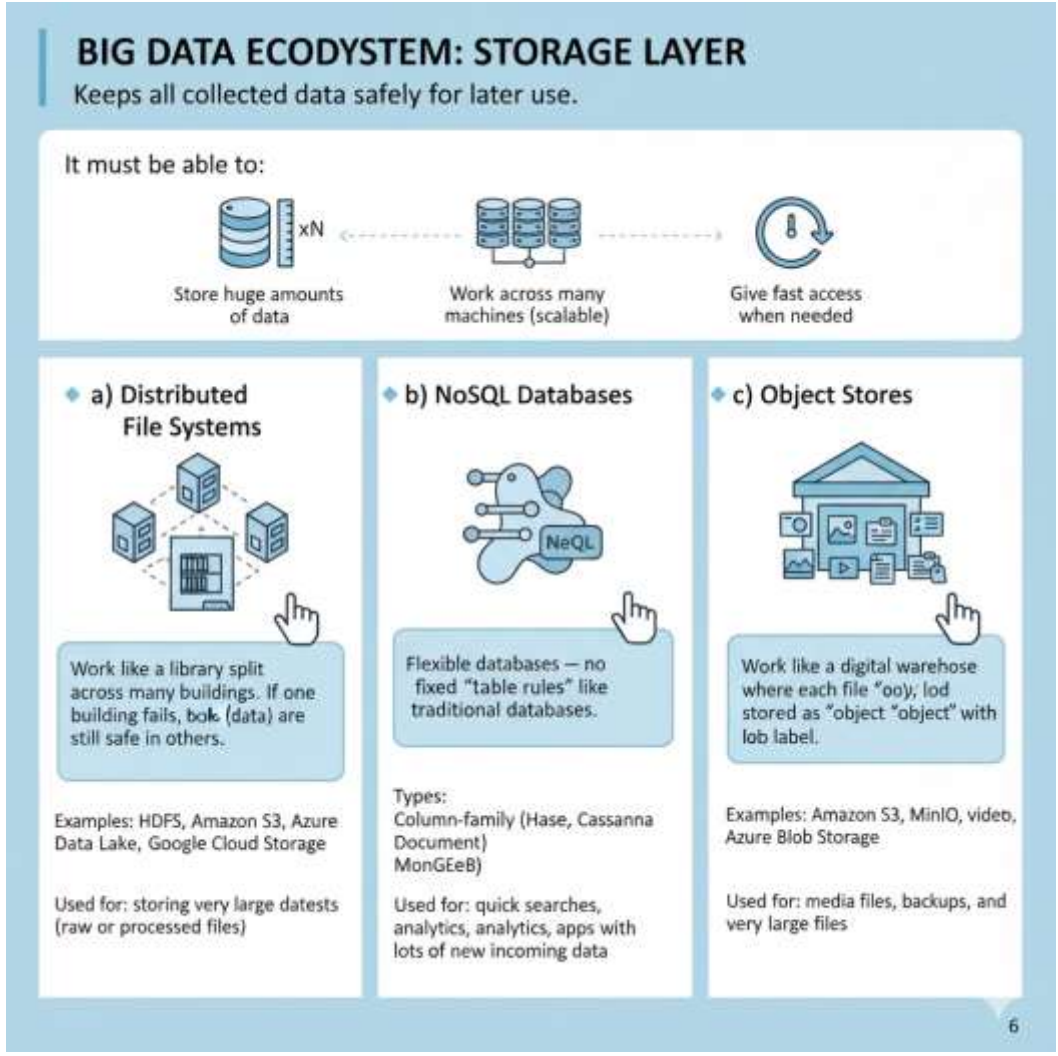
Used for: quick searches, analytics, apps with lots of new incoming data

◆ c) Object Stores

Work like a **digital warehouse** where each file (photo, video, log) is stored as an "object" with a label.

Examples: Amazon S3, MinIO, Azure Blob Storage

Together, these storage options make sure **any kind of data** — text, numbers, images, or video — can be stored and accessed efficiently.



4. Processing Layer

The **“brain”** of a Big Data system

Takes stored data → **processes, analyzes, and transforms** it into results

Must handle:
✓ Huge volumes of data
✓ Both **bulk** data and **real-time** data

◆ **a) Batch Processing** – handles large datasets collected over a period of time.

- Tools: **MapReduce**, **Apache Spark (batch mode)**
- Use cases: monthly reports, historical trend analysis, data aggregation

◆ **b) Streaming Processing** – processes data as it arrives, with low latency.

- Tools: **Spark Streaming**, **Apache Flink**, **Apache Storm**, **Kafka Streams**
- Use cases: fraud detection, sensor monitoring, real-time analytics

◆ **c) Hybrid / Unified Processing** – supports both batch and streaming in one model.

- Tools: **Apache Beam**, **Spark Structured Streaming**
- Use cases: pipelines that need both historical and real-time data combined

BIG DATA ECOCYSTEM: PROCESSING LAYER

The computational heart of a Big Data system.

It takes the data stored in a storage layer and processes, analyses, and transforms it into meanigful results. This layer mundle large volumes large edata efficiently, whether it arrivees in bulk or in real time.

Batch Processing

handles large datastes collected over a period of time.

- MapReouce, Apache Spark (batch (batch mode)
- monthly reports, historical trend analysis, data aggregation

Streaming Processing

processes data as it arrives, with low latency.

- Spark Streaming, Apache Flink Flink, Apache Storm, Kaffa Streams
- Fraud detection
- Fraud detection, sensor monitoring -ealime analytics

Hybrid / Unifed Processing

supports both batch and streaming in one model.

- Apache Beam, Spark Structured Streaming
- pirpelines that need need need both historical and reime data combined

5. Analytics Layer

Where **processed data → insights** for decision-making

- Builds on the **Processing Layer results**

Provides tools for:
✓ Querying
✓ Advanced analysis
✓ Visualization & BI

◆ a) SQL-on-Hadoop Engines

- Let analysts use **familiar SQL queries** on massive datasets
- Work like using **Excel on a huge distributed system**

Examples: Hive, Impala, Presto (Trino)

Use cases: interactive queries, BI dashboards, data exploration

◆ b) Data Science & Machine Learning Tools

- Go **beyond SQL** → enable predictions & automation
- Work like **teaching the system to learn patterns**

Examples: Python (pandas, scikit-learn, PySpark), R, Spark MLlib, TensorFlow

Use cases: anomaly detection, forecasting, recommendation systems, building AI models

BIG DATA ECOSSTEM: ANALYTICS LAYER

Where procesed data → insights for decision-making
Builds on the Processing Layer results
Provides tools for:

- ☑ Querying
- ☑ Advanced analysis
- ☑ Visulization & BI

a) SQL-on-Hadood Engines

Let analysts use familiar SQL queries on massive datesets

- Work like using Excel on a huge distributed system
- Examples: Hive, Impala, Presto (Trino)
- Use cases: interactive queriss, BI dashboards, data exploration

b) Data Science & Machine Learning Tools

Go beyond SQL → enable predictions & automation

- Work like teaching the system to learn patterns
- Examples: Python (pandas, scitt- PyPsark) R, Spark MLLlib, TensƐFrow
- Uses: anamily detection, forecating, recommendation systems, building AI models

6. Visualization Layer

- Final step of Big Data architecture → shows **insights visually**
- Makes results **easy to understand** for managers, analysts, and decision-makers
- Helps spot **trends, patterns, and anomalies** at a glance

◆ 1) Dashboards

- Ready-made, **interactive reports** that auto-update
- **Examples**: Tableau, Power BI, Apache Superset, Looker
- **Use cases**: executive KPIs, sales trends, real-time monitoring

◆ 2) Custom Visualizations

- **Tailor-made charts** for special needs
- **Examples**: D3.js, Chart.js, Plotly, ECharts
- **Use cases**: network diagrams, scientific plots, complex analytics

BIG DATA ECOSYSTEM: VISUALIZATION LAYER

Final step of Big Data architecture → shows insights visually

Makes results easy to understand for managers, analysts, and decision-makers.
Helps spot trends, patterns, and anomiles at a glance

BIG DATA ECOCYSTEM: VISUALIZATION LAYER

1) Dashbaards

Ready-made, interactive reports that auto-update
Examples: Tabelau, Power BI Apache Superset, Looker
Use cases: executive KPIS, sales trends aor-reatme monitoring

BIG DATA ECOSYTEM: VISULLIZATION LAYER

2) Custom Visualizations

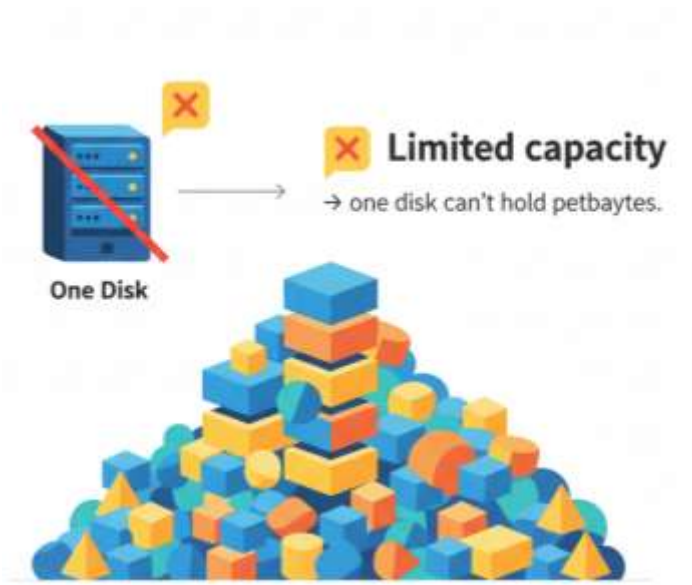
Tailor-made charts for special needs
Examples: D3 js, Chart js, Plotly, EChartsnetwork diagrams, scientific plots, complex analytics

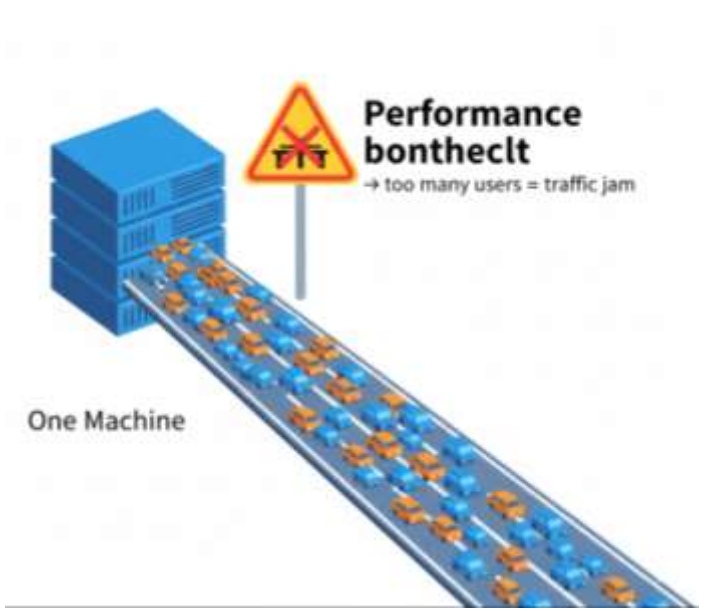
Distributed Storage

Why Do We Need Distributed Storage?

- In the past, data fit into one computer or one database.
- As data grew (logs, videos, sensor data…), one machine was no longer enough

Problems with single-machine storage:





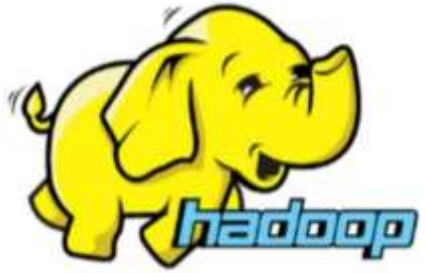
Solution → Distributed Storage:

Data is spread across many machines working together.

Result: scalable, reliable, fault-tolerant system

Provides unlimited capacity, high availability, and protection against data loss.

Hadoop Distributed File System (HDFS)



Origins and Design Philosophy

- The History of Apache Hadoop

2002:

a. **Doug Cutting** (Director at Archive.org) and **Mike Cafarella** (a student) created **Nutch**, an open-source search engine that used distributed computing.
b. **Problem**: Nutch could only run on a few machines and faced challenges with file access and sharing.

2003-2004: Google's research team published two key whitepapers:

a. **GFS (Google File System)**: A distributed file system for handling large-scale data.
b. **MapReduce**: A framework for processing and generating large datasets in a distributed environment.

2004:

a. Cutting and Cafarella were inspired by Google's papers and developed an early framework for distributed computing.
b. They adapted Nutch to work on this new framework.

2006:

a. Doug Cutting joined Yahoo and worked on improving Yahoo's search engine.
b. He enhanced the framework, transforming it into an **open-source project** for the Apache Foundation.
c. He named it **Hadoop**, after his son's toy elephant.

Early Development:

Initially, Hadoop could only manage clusters of **5 to 20 machines**, making it a work in progress.

2008:

a. Hadoop became a mature framework and was widely used by Yahoo's search engine and other divisions.

2011:

a. Hadoop was adopted by many companies and universities worldwide.

b. Yahoo's Hadoop cluster grew to **42,000 machines** and stored hundreds of **petabytes** of data.

Its design reflects a few central principles:

1. **Scalability** – The system must be able to expand from a few machines to thousands without redesign.
2. **Fault Tolerance** – Data must remain available even if hardware fails.
3. **Throughput over Latency** – Optimized for reading and writing very large files in bulk, not small, random-access queries.

HDFS follows a **write-once, read-many** model, meaning data is typically written once and then read repeatedly for analysis.

Architecture of HDFS

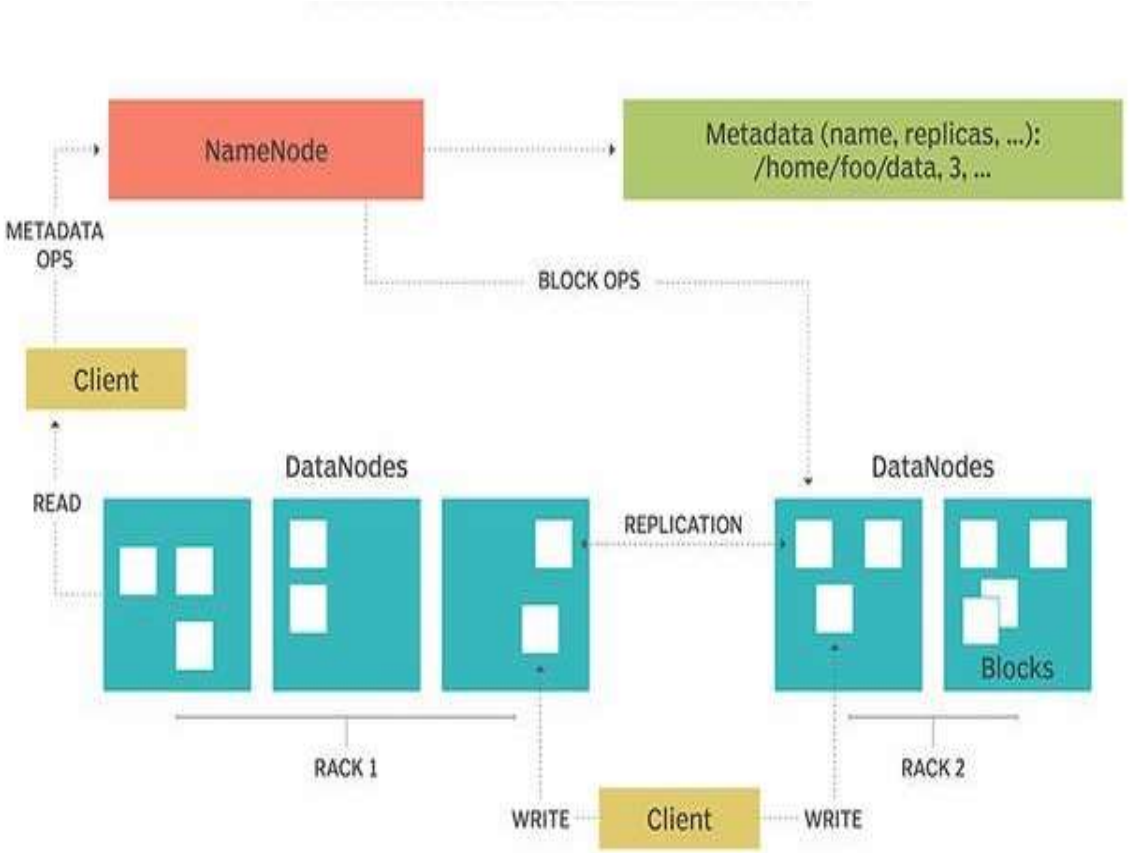
HDFS uses a **master–slave architecture**:

- **NameNode (Master)** – Manages *metadata*: file names, directory structure, block locations, and permissions.
- **DataNodes (Slaves)** – Store the actual data blocks

and serve them to clients upon request.

When a file is uploaded:

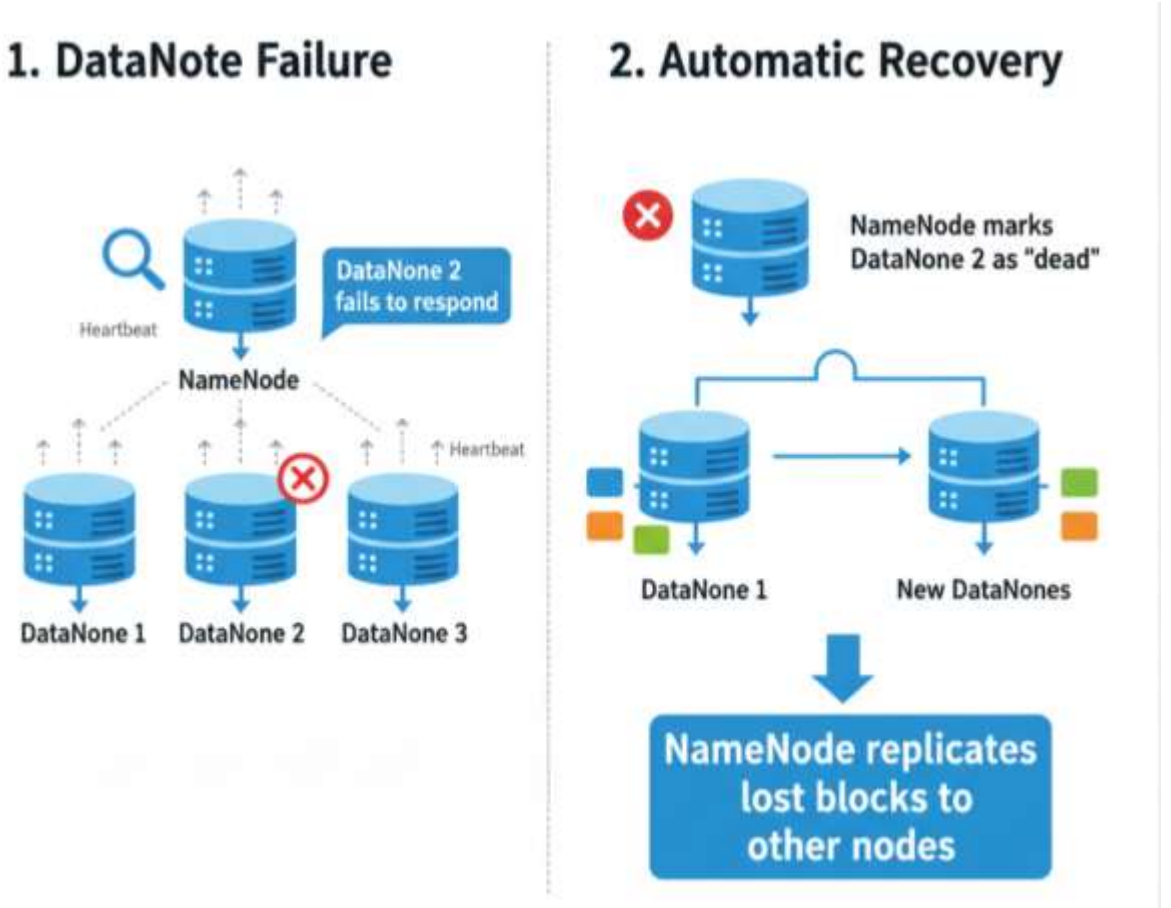
1. It's divided into **large blocks** (default: 128 MB).
2. Each block is **replicated** across multiple DataNodes (default: 3 copies).
3. The NameNode records where each block's replicas are stored.



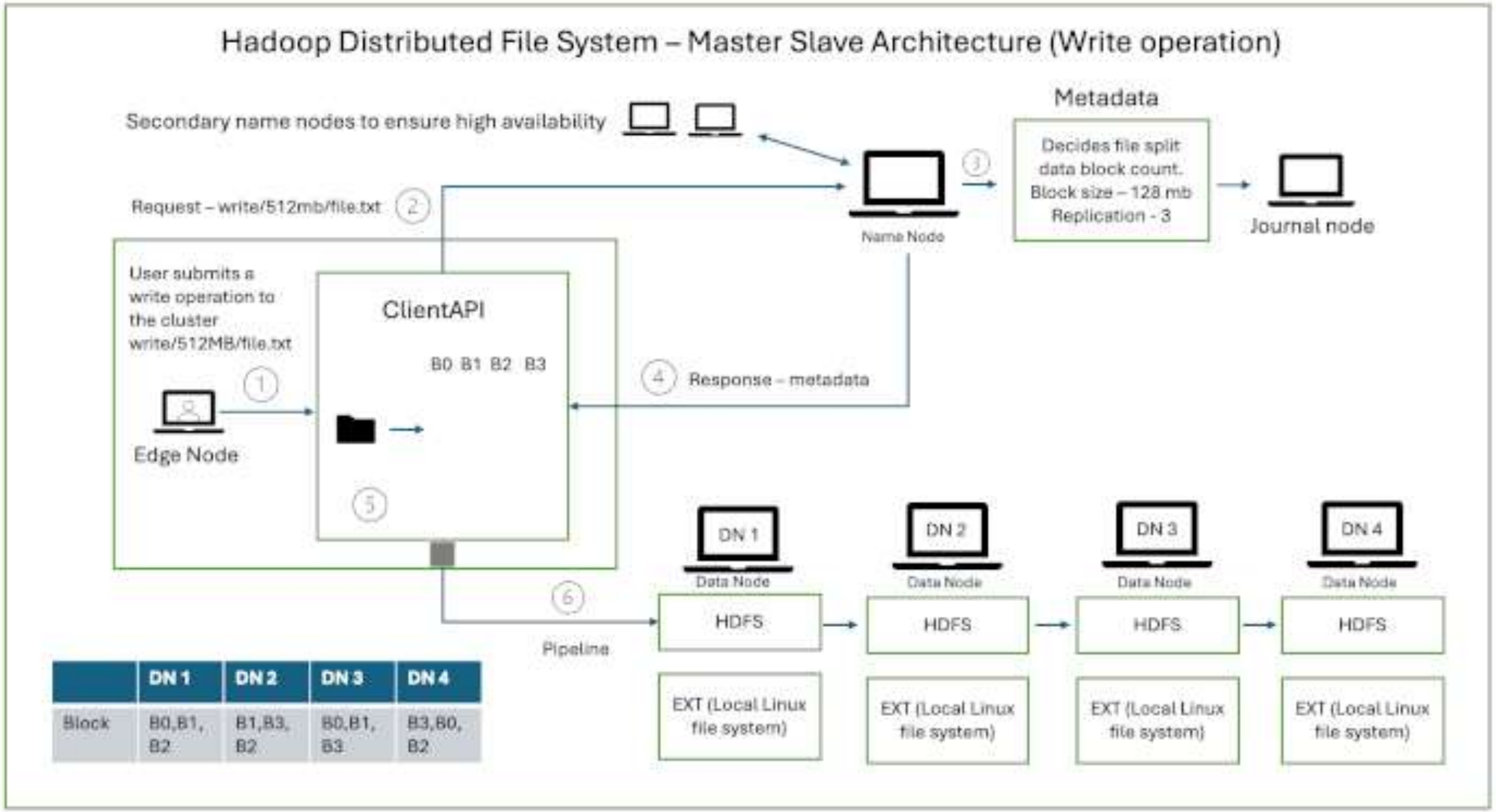
Fault Tolerance and Data Recovery

Hardware failures are expected in large clusters. HDFS deals with them automatically:

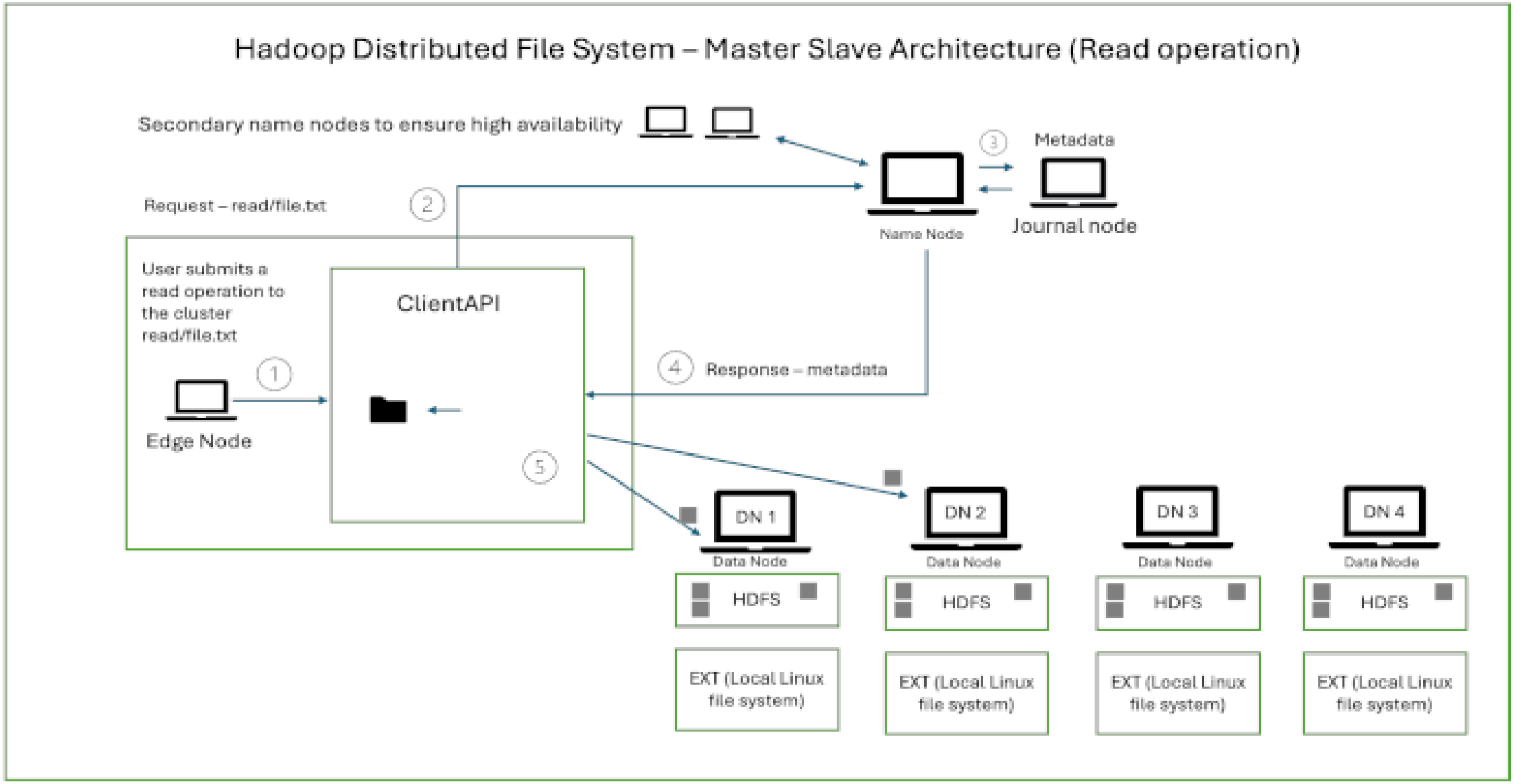
- Each DataNode regularly sends a **heartbeat** to the NameNode.
- If a DataNode fails to respond, the NameNode marks it as “dead” and replicates its lost blocks to other nodes.



Writing a File to HDFS



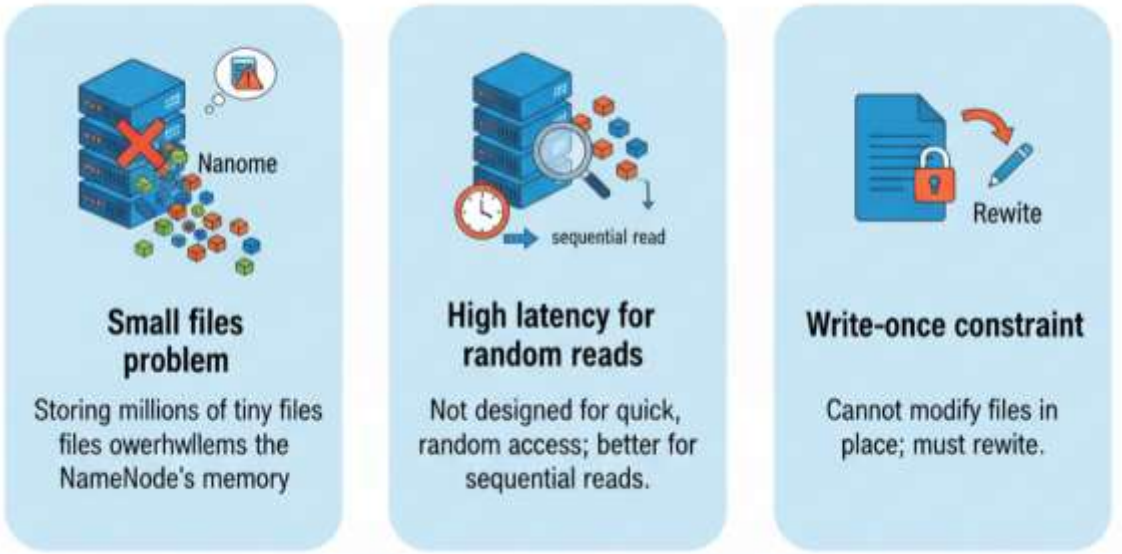
Reading a File from HDFS



Limitations of HDFS

Despite its strengths, HDFS is not perfect:

- **Small files problem** – Storing millions of tiny files overwhelms the NameNode's memory.
- **High latency for random reads** – Not designed for quick, random access; better for sequential reads.
- **Write-once constraint** – Cannot modify files in place; must rewrite.



Object Storage: Amazon S3 (Simple Storage Service)

What is Object Storage?

•**Definition**: Object storage is a way of saving data where each piece of data is stored as an **object**.

•An **object** includes:

- **Data** (the actual file, e.g., photo, video, log).
- **Metadata** (details like size, type, timestamp, tags).
- **Unique Identifier (Key)** that lets you find it.

Why Object Storage?

•Traditional HDFS works well on **on-premise clusters**, but cloud platforms are needed:

- **Infinite scalability** – handle petabytes or more.
- **Durability** – data is never lost (11 “nines” reliability).
- **Web access** – simple API calls from anywhere.

Definition: Amazon S3 is a cloud-based object storage service that lets you store and retrieve virtually unlimited amounts of data from anywhere on the internet.

Unlike block or file storage, which organize data into fixed units or hierarchical folders, S3 organizes data as **objects** inside **buckets**. Each object contains the raw data plus metadata (e.g., timestamps, tags, owner information) and is referenced by a unique key.

Key features

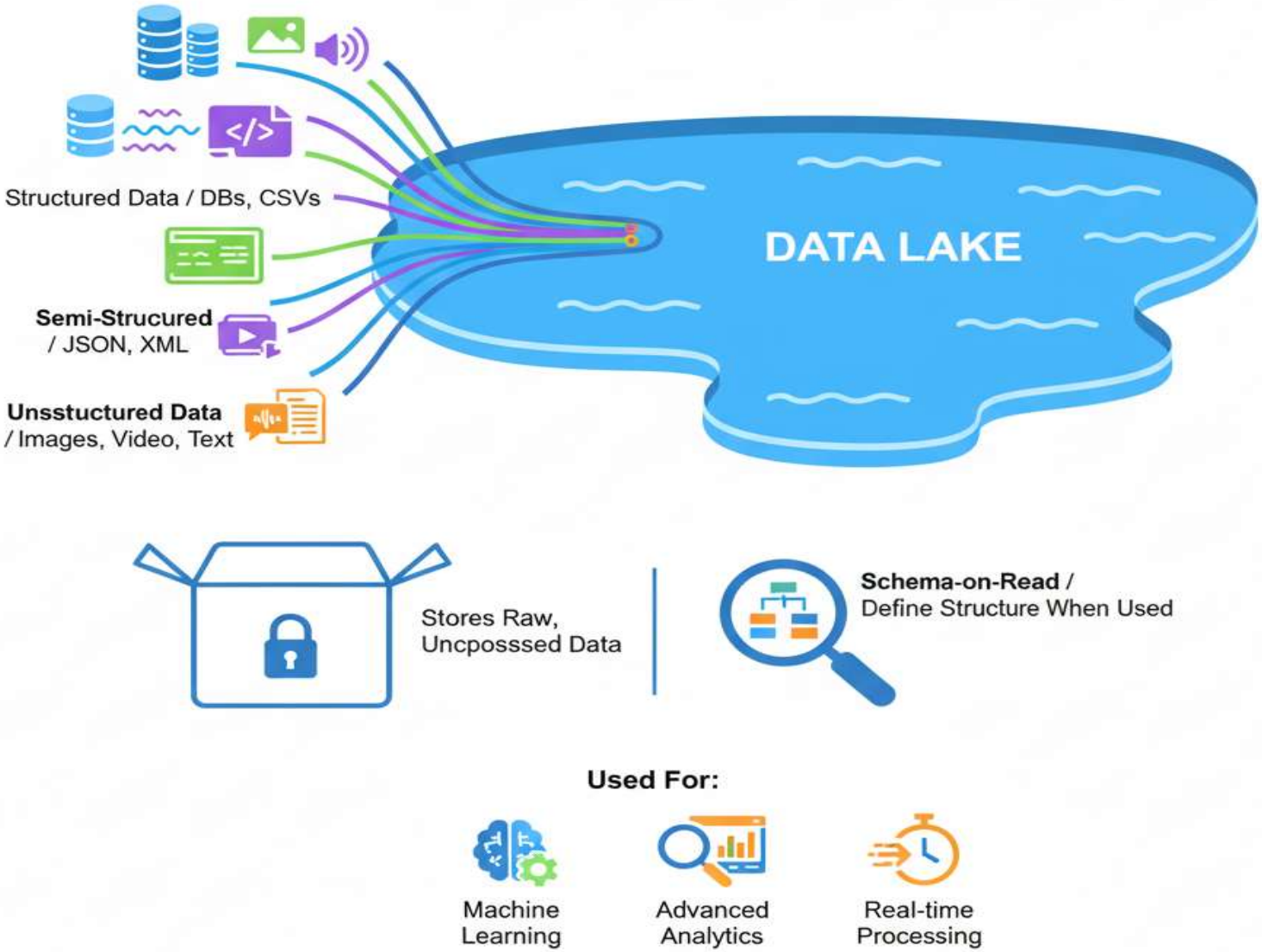
Feature	Description
Scalability & Elasticity	You can store virtually unlimited data (object count or total size). S3 automatically scales as you add data or delete data. You don't need to pre-provision capacity.
Durability & Availability	S3 is designed for **11 nines** of durability (99.999999999%) and **99.99% availability** in most regions by default. This means data loss due to physical failures is extremely unlikely.
Security & Access Control	Data in S3 is encrypted by default. Access is controlled via IAM (Identity and Access Management), bucket policies, access control lists (ACLs), etc. Logging and auditing are supported.
Multiple Storage Classes	S3 offers different *storage classes* depending on how often and how quickly you need access. For example, frequent access vs infrequent vs archival. Helps optimize cost vs performance.
Lifecycle Management	S3 allows you to automate the transition of objects between different storage classes (e.g. migrate older items to cheaper, less-accessible classes) or set up expiration. This helps control cost and data age.

How It Works

1. **Buckets** act as top-level containers, like folders.
2. **Objects** are files (up to 5 TB each) plus metadata, identified by a key.
3. **Regions** determine where data is physically stored, affecting latency, compliance, and cost.
4. **Versioning** allows recovery of older object versions if files are deleted or overwritten.

Strengths and Limitations

Data lake


Structured Data / DBs, CSVs
Semi-Strucured
/ JSON, XML
Unsstuctured Data
/ Images, Video, Text
DATA LAKE
Stores Raw,
Uncposssed Data
Schema-on-Read /
Define Structure When Used
Used For:
Machine
Learning
Advanced
Analytics
Real-time
Processing

What is a Data Lake?

•**Definition**: A low-cost storage environment for huge volumes of **raw data** in any format:

- Structured (tables)
- Semi-structured (JSON, XML)
- Unstructured (images, logs, audio, video)

•**Foundation**: Built mostly on **cloud object storage** (AWS S3, Google Cloud Storage, Azure Blob).

•**Why?** Scalable, durable, and cost-efficient for modern big data needs.

Key Characteristics

1.Schema-on-read

1. Data stored in raw form.
2. Schema applied only when reading/processing → flexibility.

2.Separation of storage & compute

1. Storage = where data sits.
2. Compute = where analysis happens.
3. Each scales independently → lower costs.

3.Supports all data types

1. Structured + semi-structured + unstructured.
2. Enables AI/ML with access to diverse datasets.

4.Cloud object storage backbone

4. Uses S3, GCS, Azure Blob, etc.
5. Provides scalability, durability, simple management.

5.Cost-effectiveness

4. Store raw data cheaply.
5. Transform/process only when needed.
6. Tiered storage further reduces costs.

6.Governance & metadata

4. Prevents “data swamps.”
5. Uses catalogs & metadata for discovery, management, and trust.

Use Cases

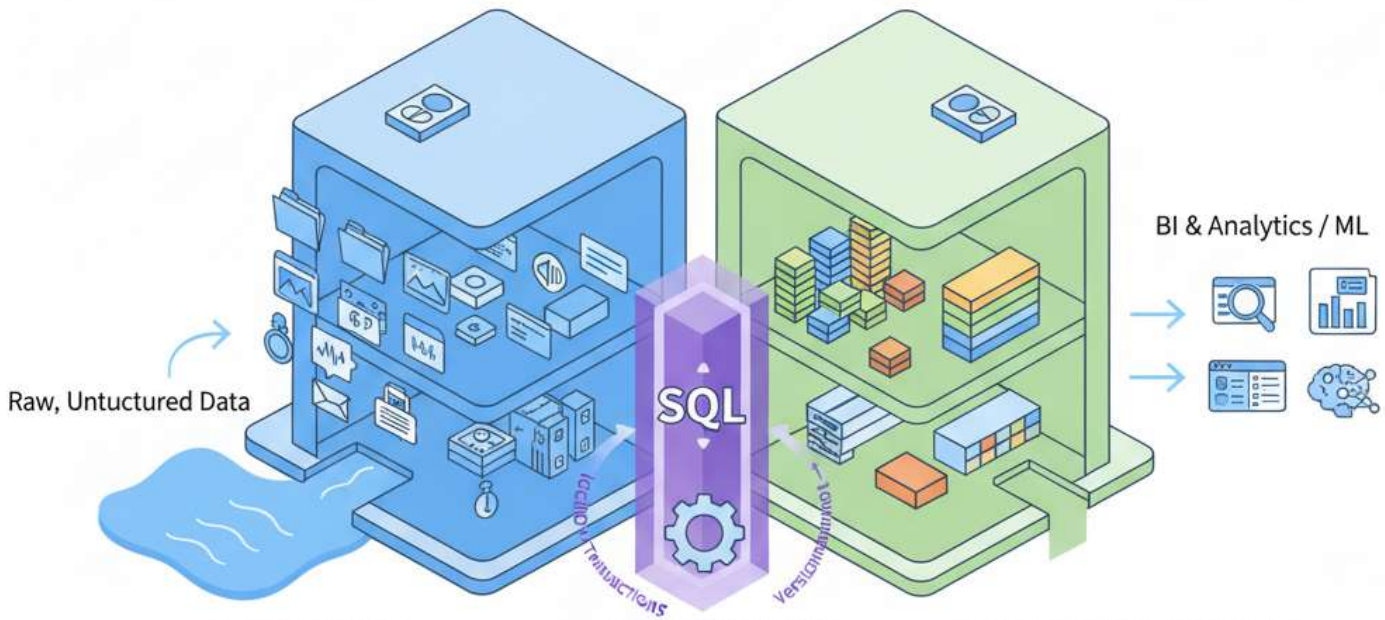
- AI / ML work: For training models you often need huge volumes of raw data (images, logs, text) — data lakes support that.

- All-purpose storage: Data lakes serve as a central repository where all incoming data types land. Later, data can be moved to specialized systems if needed (e.g. data warehouses) or processed with external tools.

- Backups, archives, cold (inactive) data: Because storage is relatively cheap and durable, often data lakes are used for historical data that may be needed for compliance or future analytics.

DATA LAKEHOUSE

Unified Platform for All Your Data Needs



DATA LAKE

Scalable, Cheap Storage

Metdato / Compute
Versining

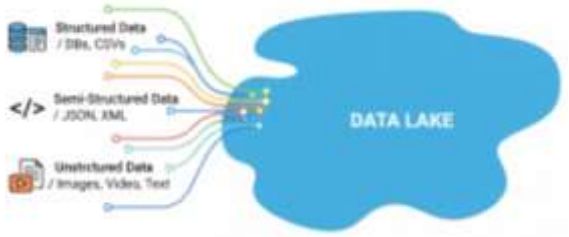
DATA WAREHOUSE

Clean, Structured Data
Fast Queriys

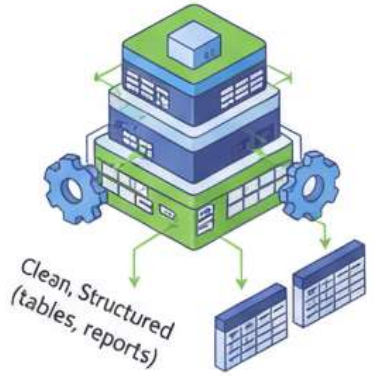
Lakehouse (Bridging Data Lakes & Warehouses)

A Data Lakehouse is a centralized repository that allows you to store structured and unstructured data at any scale. It is a hybrid approach that combines the best aspects of a data lake and a data warehouse.

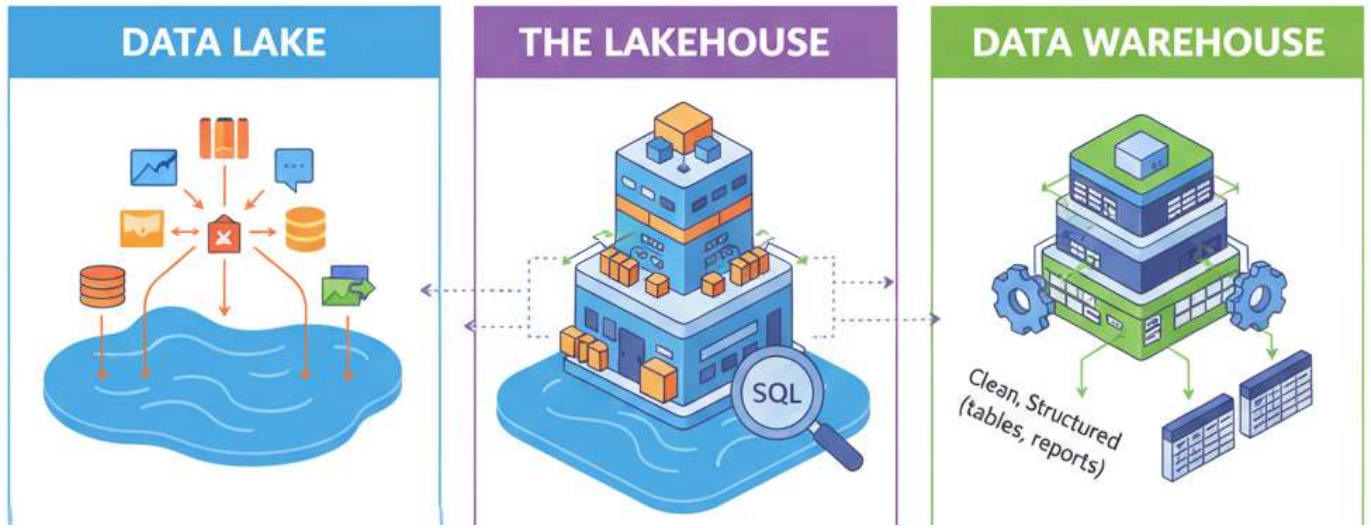
A Data Lake is a storage system that allows you to store large amounts of raw data in its native format, without the need to structure it upfront. This makes it an ideal platform for storing data that is diverse and hard to classify.



A Data Warehouse, on the other hand, is designed to store structured data that has been cleaned, transformed, and integrated from multiple sources. It is optimized for fast querying and analysis, and is typically used for business intelligence and reporting purposes.



A Data Lakehouse combines the scalability and flexibility of a data lake with the structured data storage and fast querying capabilities of a data warehouse. It allows you to store both structured and unstructured data in a single repository, and to use SQL-based tools to query and analyze the data. This makes it a powerful platform for data analytics, and allows you to gain insights from all of your organization's data, regardless of its structure.



Key Features of a Lakehouse

•**Handles all data types**
Can store tables, documents, images, videos → all in one place.

•**Open formats & storage**
Saves data in standard formats (like Parquet) on cloud storage (like S3) → easy to use with many tools.

•**Reliable & flexible**
Makes sure data is consistent , and lets you change data structure (schema) without breaking things.

•**Storage and compute are separate**
You can add more storage or more processing power independently → saves money and improves speed.

•**One system for many tasks**
Same data can be used for reports, dashboards, AI/ML, or real-time streams.

•**Strong management**
Has catalogs, metadata, and access control → keeps data organized and avoids messy “data swamp.”

How It Works (Components / Layers)

To deliver these features, lakehouse architectures typically implement several layers/components:

1. **Storage Layer** Uses scalable, low-cost object storage (e.g. S3, Azure Blob). Raw data, semi-processed data, and curated data are stored there. Open file formats like Parquet / ORC help with compatibility.
2. **Table & Metadata Layer** Adds a layer on top of raw storage: catalogs, table formats (Delta Lake, Apache Iceberg, Apache Hudi) that track schema, support time-travel, versioning, ACID transactions.
3. **Compute / Engine Layer** Engines (Spark, Flink, Presto/Trino, etc.) read from the storage and metadata layers. Different engines may be used for batch, streaming, interactive queries.
4. **Consumption / Query Layer** Business intelligence tools, dashboards, ML pipelines, reporting layers that consume data. Because data is structured/managed, queries are faster and more reliable.
5. **Governance / Quality / Access Control** Built-in cataloging, permissions, auditing, data validation. Without this, the lakehouse risks becoming disorganized.

Advantages vs. Challenges

Advantages

- **Reduced duplication**: no need to move data between lakes and warehouses.
- Freshness and agility: ability to run analytics or ML directly on recent data, even streamed data.
- **Cost efficiency**: storage remains cheap; compute is used only when needed.
- **Strong consistency, reliability**: thanks to ACID transactions, versioning, and schema enforcement.

Challenges

- **Complexity**: building and maintaining components like metadata catalogs, transactional table formats, schema evolution tools adds overhead.
- **Performance tuning**: query performance depends on layout (partitioning, clustering), indexing, compaction. Poor configuration can degrade performance.
- **Tooling maturity**: though many tools are now well-developed, integrating streaming, BI, ML, governance seamlessly can still be challenging.

Use Cases

- One System for All Needs

Business reporting (SQL dashboards) + advanced ML on raw log files → without separate platforms.

- Real-time + Historical Analytics

Process streaming data as it arrives, while also analyzing stored historical data.

- Governance & Compliance

Strong governance, versioning, and auditing → useful for regulated industries (finance, healthcare, government).

- Support for Many Data Types

Works with structured (tables), semi-structured (JSON, XML), and unstructured (logs, images).