



# GBDi

Government Big Data Institute

สถาบันส่งเสริมการวิเคราะห์และบริหารข้อมูลขนาดใหญ่ภาครัฐ (สวช.)



# Intermediate Data Engineering

## Data Sources

กฤษณะกาน เพชรสุวรรณ

# Before we start

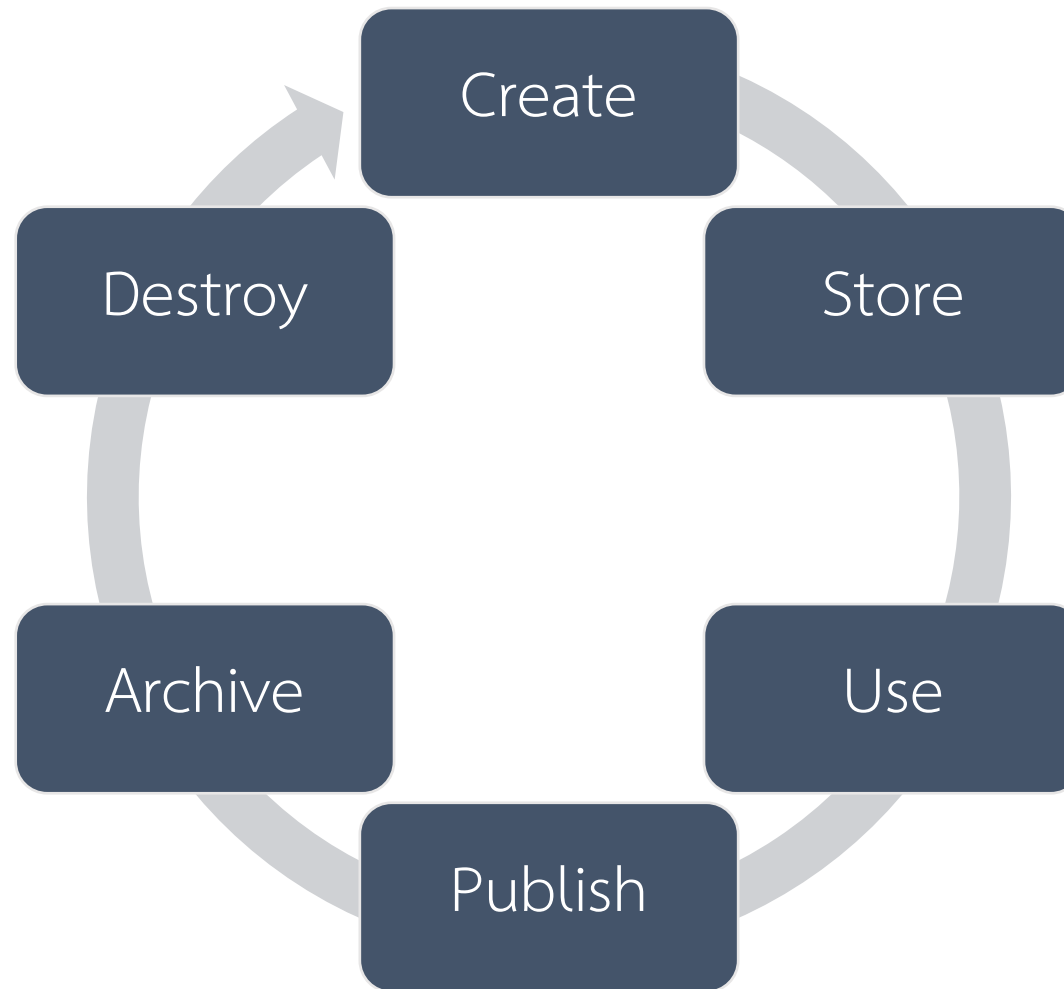
- Data Engineering คืออะไร ?
- หน้าที่ของ Data Engineer ?
- แหล่งข้อมูลมาจากไหน
- เก็บข้อมูลไว้ที่ไหน

# What is Data Engineering ?

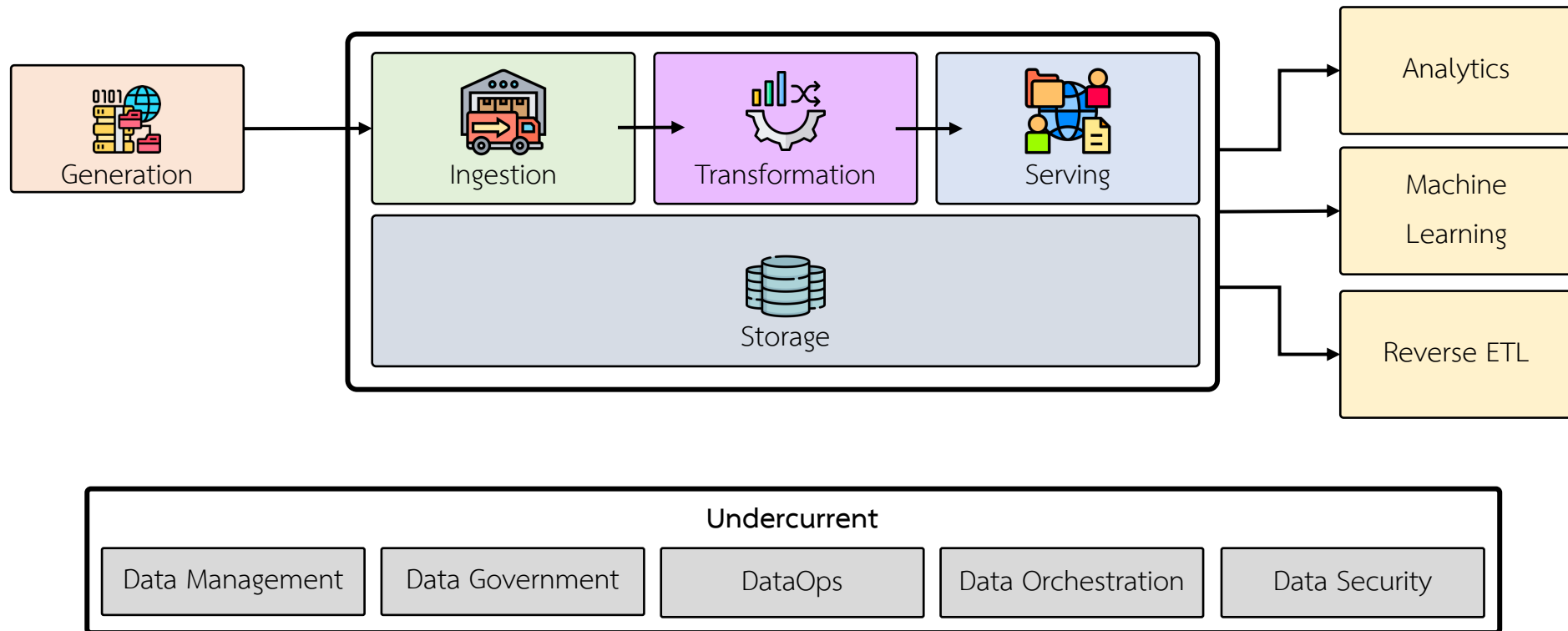
# What is Data Engineering

- *Data engineering* is the development, implementation, and maintenance of systems and processes that take in raw data and produce high-quality, consistent information that supports downstream use cases, such as analysis and machine learning.
- *Data engineering* is the intersection of security, data management, DataOps, data architecture, orchestration, and software engineering.
- A *data engineer* manages the data engineering lifecycle, beginning with getting data from source systems and ending with serving data for use cases, such as analysis or machine learning.

# Data Life Cycle



# Data Engineering Pipeline



# Outline

- Data Engineering Lifecycle
- Data Generation in Source Systems
  - Type of Data
  - Source System In Practical
- Data Storage
  - Data Storage Systems
  - Data Engineering Storage Abstraction
  - Big Ideas and trends in Storage



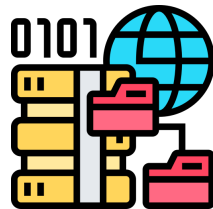
# GBDi

Government Big Data Institute

สถาบันส่งเสริมการวิเคราะห์และบริหารข้อมูลขนาดใหญ่ภาครัฐ (สวช.)



## Data Generation





# Type of Data

## Structured data

ID	Name	Age	Degree
1	John	18	B.Sc.
2	David	31	Ph.D.
3	Michael	51	Ph.D.
4	Jenny	24	B.Sc.
5	Cathie	25	M.Sc.
...	...	...	...

## Semi-structured data

```
<University>
  <Student ID="1">
    <Name>John</Name>
    <Age>18</Age>
    <Degree>B.Sc.</Degree>
  </Student>
  <Student ID="2">
    <Name>David</Name>
    <Age>31</Age>
    <Degree>Ph.D.</Degree>
  </Student>
  ...
</University>
```

## Unstructured data

The university has 5600 students.

John's ID is number 1, he is 18 years old and already holds a B.Sc. degree.

David's ID is number 2, he is 31 years old and holds a Ph.D. degree.

Michael's ID is number 3, he is 51 years old and also holds the same degree as David, a Ph.D. degree.

...

## Structured data

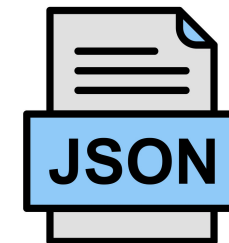
ID	Name	Age	Degree
1	John	18	B.Sc.
2	David	31	Ph.D.
3	Michael	51	Ph.D.
4	Jenny	24	B.Sc.
5	Cathie	25	M.Sc.
...	...	...	...





## Semi-structured data

```
<University>
  <Student ID="1">
    <Name>John</Name>
    <Age>18</Age>
    <Degree>B.Sc.</Degree>
  </Student>
  <Student ID="2">
    <Name>David</Name>
    <Age>31</Age>
    <Degree>Ph.D.</Degree>
  </Student>
  ...
</University>
```





## Unstructured data

The university has 5600 students.

John's ID is number 1, he is 18 years old and already holds a B.Sc. degree.

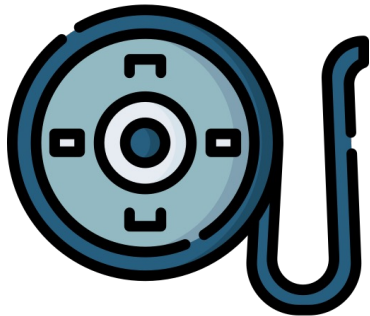
David's ID is number 2, he is 31 years old and holds a Ph.D. degree.

Michael's ID is number 3, he is 51 years old and also holds the same degree as David, a Ph.D. degree.

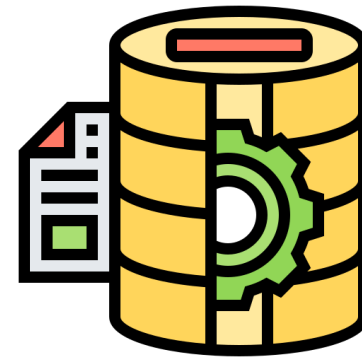
...

# How is data created

- *Analog data* creation occurs in the real world, such as vocal speech, sign language, writing on paper, or playing an instrument. This analog data is often transient.



- *Digital data* is either created by converting analog data to digital form or is the native product of a digital system. An example is a mobile texting app that converts analog speech into digital text, a credit card transaction on an ecommerce platform.



# Source System Main Idea

- File and unstructured data
- APIs
- Application Databases (OLTP)
- Online Analytical Processing System (OLAP)
- Change Data Capture (CDC)
- Logs
- Messages and Streams

# Online Transaction Processing (OLTP)

- Purpose: OLTP systems are designed for real-time transaction processing and day-to-day operations of an organization.
- Use: OLTP systems handle a large number of short, simple, and frequent transactions, such as adding, updating, or deleting records in a database.
- Data Model: OLTP databases typically have a normalized data model to minimize data redundancy and maintain data integrity.
- Performance: Emphasizes quick response times and high throughput for individual transactions.
- Example: E-commerce websites, banking systems, airline reservation systems.

# Online Analytical Processing (OLAP)

- Purpose: OLAP systems are designed for complex data analysis and business intelligence purposes.
- Use: OLAP systems handle complex queries that involve aggregations, grouping, and multidimensional analysis to gain insights from historical data.
- Data Model: OLAP databases usually have a denormalized or multidimensional data model to facilitate efficient analytical queries.
- Performance: Emphasizes fast query performance for analytical processing, even if individual transactions may take longer.
- Example: Data warehouses, business intelligence applications, decision support systems.



Aspect	OLTP (Online Transaction Processing)	OLAP (Online Analytical Processing)
Purpose	Real-time transaction processing for day-to-day operations	Complex data analysis and business intelligence
Use	Handles frequent short transactions	Handles complex queries for data analysis
Data Model	Typically normalized data model	Denormalized or multidimensional data model
Performance	Emphasizes quick response times for individual transactions	Emphasizes fast query performance for analytical processing
Example	E-commerce websites, banking systems, reservation systems	Data warehouses, business intelligence applications
Query Complexity	Simple and frequent queries	Complex queries with aggregations and multidimensional analysis
Data Structure	Stores current state of data	Stores historical and aggregated data
Data Volume	Handles small amounts of data with high transaction rates	Handles large volumes of data for analysis
Data Update	Constantly updated with real-time transactions	Mostly read-only with periodic data refresh
Data Usage	Supports day-to-day operations	Facilitates strategic decision-making

# ACID Properties

- **Atomicity** requires a transaction to execute completely or not at all.
- **Consistency** requires that when a transaction has been committed, the data must conform to the database schema.
- **Isolation** requires that concurrent transactions execute separately from each other.
- **Durability** requires the ability to recover from an unexpected system failure or power outage to the last known state.

# Source System Practical

- Database
- APIs
- Data Sharing
- Thirds Party Data Source
- Message Queues and Event-Streaming Platforms

# Relational Database







- Relational database management systems (RDBMS) have a long history, originating in the 1970s at IBM and gaining widespread popularity in the 1980s through Oracle's efforts.
- The growth of the internet led to the emergence of the LAMP stack, which further propelled the adoption of RDBMS solutions like MySQL in combination with Linux, Apache web server, and PHP.
- RDBMS systems are typically *ACID compliant*. Combining a *normalized schema*, *ACID compliance*, and support for *high transaction rates* makes relational database systems ideal for storing rapidly changing application states.

# How RDBMS store

- Data in relational databases is stored in tables of relations (rows) with multiple fields (columns) and a consistent schema.
- Tables are typically indexed by a primary key, allowing for efficient data retrieval and joining with other tables through foreign keys.
- Normalization is employed to avoid data duplication and maintain consistency, making relational databases suitable for storing rapidly changing application states.

# Non-Relational Database: NoSQL

- NoSQL databases are purpose built for specific data models and have flexible schemas for building modern applications.
- NoSQL databases are widely recognized for their ease of development, functionality, and performance at scale.

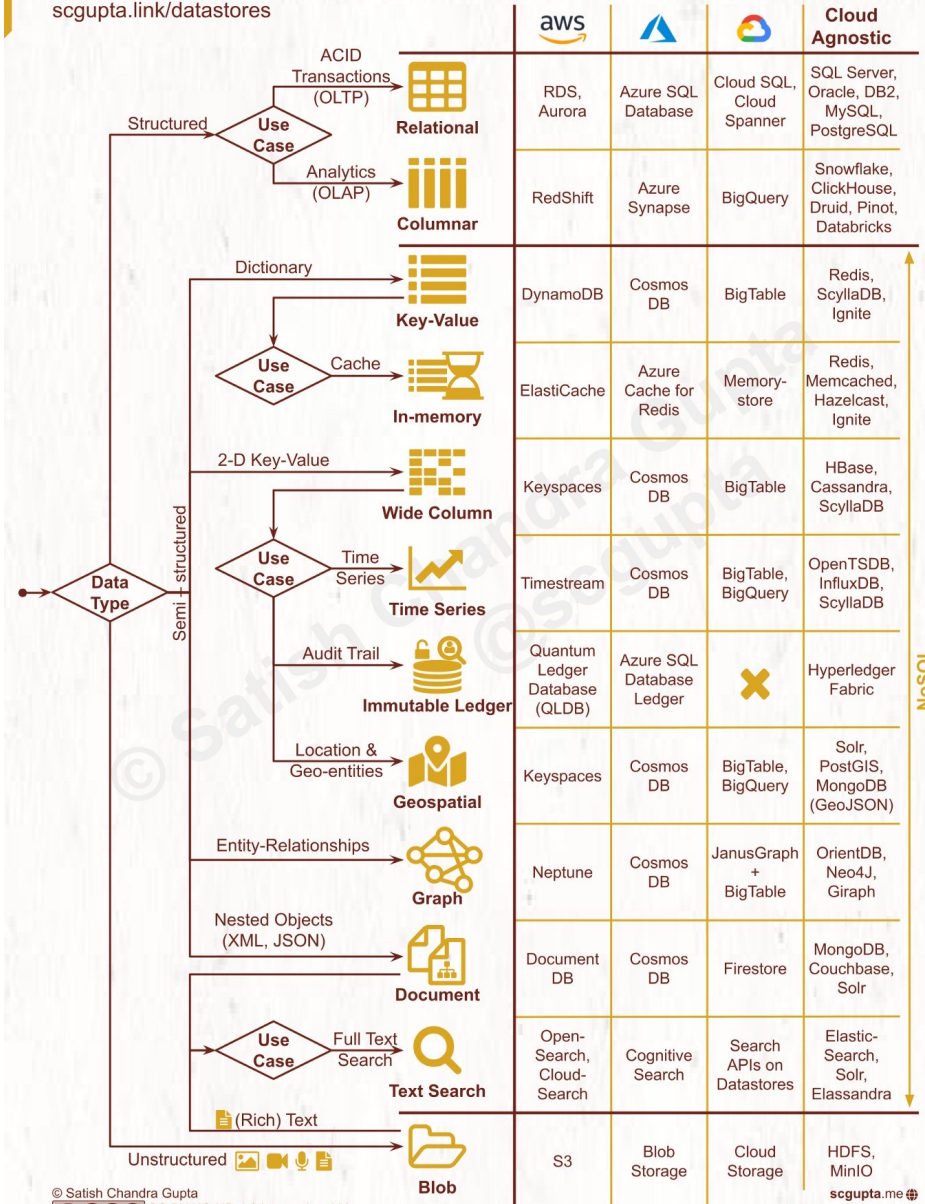
NoSQL Database Type	Description	Example Databases
Key-Value 	Stores data in key-value pairs, where each key is a unique identifier for the associated value.	Redis, Amazon DynamoDB, Riak
Wide-Column 	Organizes data in columns rather than rows, suitable for handling large amounts of data with high write and read throughput.	Apache Cassandra, HBase
Document 	Stores data in documents, typically using JSON or BSON formats, allowing for flexible schema.	MongoDB, Couchbase, RavenDB
Graph 	Stores data using graph structures (nodes, edges, properties) to represent and manage relationships between data.	Neo4j, Amazon Neptune, ArangoDB
In-Memory 	Stores data in memory for rapid access and low-latency operations.	Redis, Memcached, Couchbase In-Memory
Search 	Designed for efficient full-text search and indexing capabilities.	Elasticsearch, Apache Solr, Amazon CloudSearch

Aspect	Relational (SQL) Databases	Non-Relational (NoSQL) Databases
Data Model	Structured data with fixed schemas (tables, rows, columns)	Flexible data model (documents, key-value pairs, columns, graphs)
Schema	Predefined schema with strict data types and relationships	Dynamic or schema-less; each record can have its own structure
Query Language	SQL (Structured Query Language)	NoSQL databases use various query languages (e.g., MongoDB uses MongoDB Query Language)
Scalability	Vertical scaling (hardware upgrades)	Horizontal scaling (distributed across multiple servers or nodes)
Performance	Excellent for complex queries and joins	High performance for simple queries and large-scale read/write operations
ACID Transactions	Supports ACID transactions for data integrity	NoSQL databases may support ACID, but often prioritize eventual consistency and partition tolerance
Data Integrity	Enforces data integrity through foreign keys and constraints	May have looser data integrity due to flexible schema
Use Cases	Well-suited for complex, structured data with strict relationships	Ideal for handling unstructured, semi-structured, and rapidly changing data
Examples	MySQL, PostgreSQL, Oracle	MongoDB, Redis, Cassandra, Neo4j



# SQL vs. NoSQL: Cheatsheet for AWS, Azure, and Google Cloud

scgupta.link/datastores



Communication Protocol	Description	Usage and Characteristics
REST (Representational State Transfer)	REST is an architectural style for designing networked applications. It uses HTTP methods (GET, POST, PUT, DELETE) to perform CRUD operations on resources.	<ul style="list-style-type: none"> <li>- Well-established and widely used protocol.</li> <li>- Suitable for simple and CRUD-based APIs.</li> <li>- Each endpoint represents a resource, and state changes are made using HTTP methods.</li> <li>- Stateless and cacheable.</li> </ul>
GraphQL	GraphQL is a query language for APIs that allows clients to request exactly the data they need. It provides a single endpoint for multiple data-fetching operations.	<ul style="list-style-type: none"> <li>- Client-centric approach; clients request only the data they need.</li> <li>- Provides a flexible and hierarchical data model.</li> <li>- Reduces over-fetching and under-fetching of data.</li> <li>- Easier versioning of APIs.</li> </ul>
Webhook	A webhook is an HTTP callback initiated by an external system when a specific event occurs. It sends data to a predefined URL or endpoint for processing.	<ul style="list-style-type: none"> <li>- Used for real-time event-driven communication.</li> <li>- Ideal for integrating systems asynchronously.</li> <li>- Enables server-to-server communication.</li> <li>- Used in web applications and APIs to receive notifications.</li> </ul>
RPC (Remote Procedure Call)	RPC is a communication protocol that allows a program on one device to call a function or procedure on another device over a network.	<ul style="list-style-type: none"> <li>- Primarily used for inter-process communication (IPC).</li> <li>- Suitable for distributed systems and microservices architecture.</li> <li>- Often employed in client-server architectures.</li> </ul>

# How to ingest, collect data?

- Direct Database Connect
- CDC
- API
- Message Queues and Event-Streaming Platforms
- Databases and File Export
- Shell
- SSH
- SFTP, SCP
- Webhook
- Webscraping
- Transfer Appliances for Data Migration



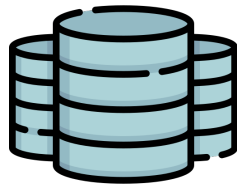
# GBDi

Government Big Data Institute

สถาบันส่งเสริมการวิเคราะห์และบริหารข้อมูลขนาดใหญ่ภาครัฐ (สวช.)

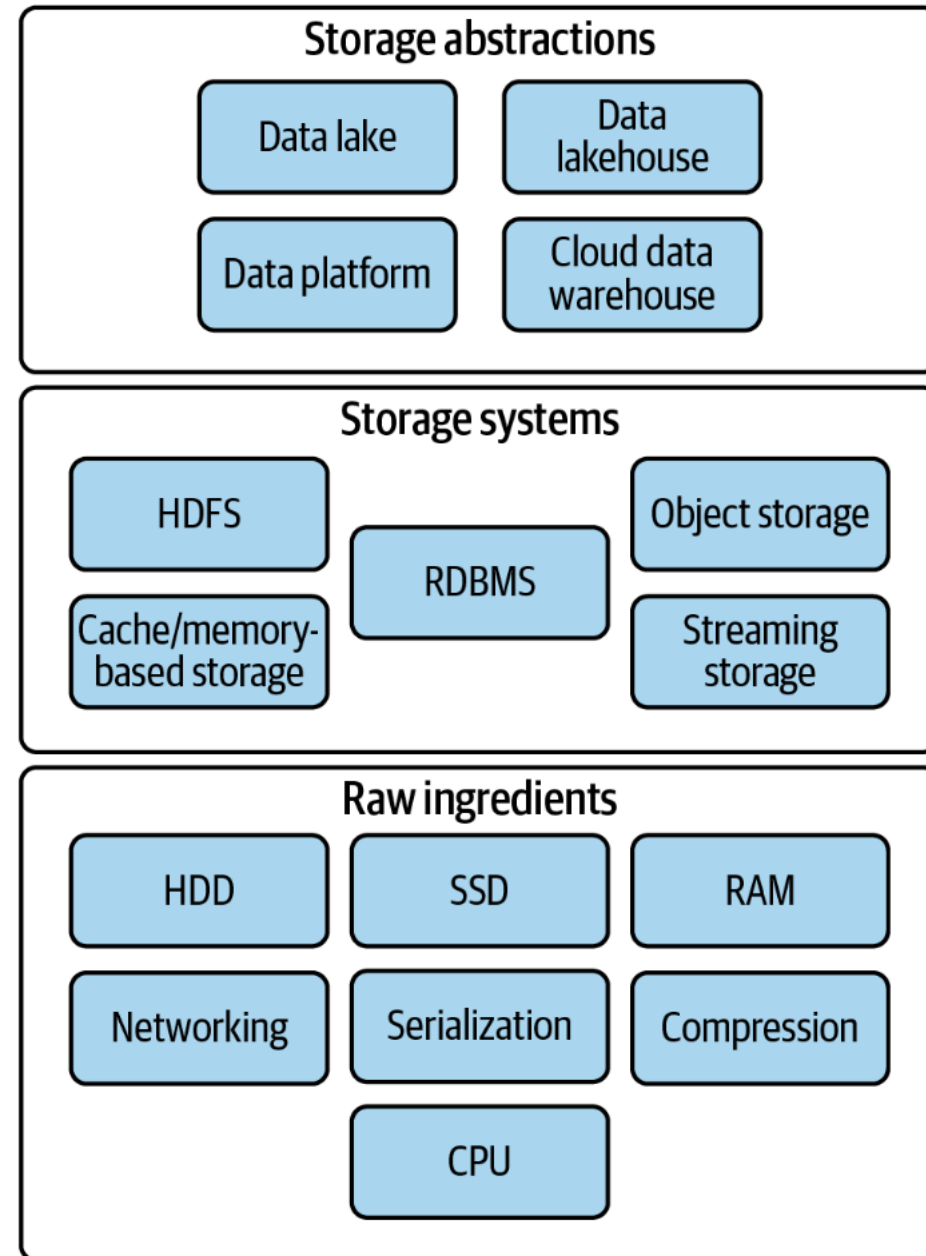


## Data Storage



# Form of Storage

- Raw Ingredients of Data Storage
- Storage System
- Storage Abstraction



# Raw Ingredients

- Magnetic Disk Drive
- Solid-State Drive (SSD)
- Random Access Memory (RAM)
- Network and CPU
- Caching

Storage type	Data fetch latency <sup>a</sup>	Bandwidth	Price
CPU cache	1 nanosecond	1 TB/s	N/A
RAM	0.1 microseconds	100 GB/s	\$10/GB
SSD	0.1 milliseconds	4 GB/s	\$0.20/GB
HDD	4 milliseconds	300 MB/s	\$0.03/GB
Object storage	100 milliseconds	10 GB/s	\$0.02/GB per month
Archival storage	12 hours	Same as object storage once data is available	\$0.004/GB per month

<sup>a</sup> A microsecond is 1,000 nanoseconds, and a millisecond is 1,000 microseconds.

# Data Storage System

- File Storage
- Block Storage
- Object Storage
- Cache and Memory-based Storage
- Hadoop Distributed File System (HDFS)
- Streaming Storage



# File Storage

- Data organized into hierarchical directories and files
- Accessed through a file system interface
- Suitable for various file-based applications and user data
- Provides a user-friendly way to organize and access data
- Files are typically of different sizes and formats
- Allows for easy management of data using file names and directories
- Commonly used in personal computers, file servers, and network-attached storage (NAS)
- May have limitations in handling high-performance workloads with random access patterns
- Well-suited for applications requiring sequential access, streaming, or sharing files between users

# Block Storage

- Data organized into fixed-sized blocks
- Block size typically ranges from kilobytes to megabytes
- Each block has a unique identifier
- Used in storage area networks (SANs) and cloud computing
- Enables low-level access for direct read and write operations
- Ideal for applications requiring random access, e.g., databases, virtual machines
- Offers flexibility in data management and scalability
- Easily expandable without affecting the underlying file system
- Contrasts with file storage that uses hierarchical directories and file systems

# Object Storage

- Data organized as objects, each with a unique identifier or key
- Objects store data, metadata, and a unique identifier
- Provides a flat namespace for scalable and efficient data access
- Suitable for unstructured data, such as images, videos, documents
- Scalable and ideal for storing vast amounts of data
- Can be geographically distributed for redundancy and disaster recovery
- Accessible over HTTP(S) using RESTful APIs (e.g., Amazon S3, Azure Blob Storage)
- Well-suited for cloud-based applications and data storage in distributed systems
- Offers seamless horizontal scaling without impacting performance
- Object storage systems automatically handle data replication and data integrity

Feature	File Storage	Block Storage	Object Storage
Data Organization	Hierarchical directories and files	Fixed-sized blocks with unique identifiers	Objects with unique identifiers (keys)
Access Method	File system interface	Low-level direct read/write operations	HTTP(S) RESTful API
Use Case	Personal computers, file servers, NAS	Databases, virtual machines, high-performance	Unstructured data (images, videos, documents)
Scalability	Limited scalability due to hierarchy	Easily expandable without impacting system	Highly scalable, suitable for vast data volumes
Redundancy	Limited redundancy options	RAID configurations provide data redundancy	Geographically distributed for redundancy
Performance	Efficient for sequential access	Excellent for random access patterns	Good for large-scale distributed applications
Access Protocol	Paths (e.g., /folder/file.txt)	Raw storage access	HTTP(S) RESTful API (e.g., Amazon S3, Azure Blob)
Example Services	Local file storage, NAS	SANs, Cloud block storage (e.g., AWS EBS)	Amazon S3, Azure Blob Storage, Google Cloud Storage
Use in Cloud	Yes, widely used	Common in cloud environments	Yes, widely used in various cloud services

# On cloud storage

Storage Service	AWS (Amazon Web Services)	Azure (Microsoft Azure)	GCP (Google Cloud Platform)
File Storage	Amazon Elastic File System (EFS)	Azure Files	Cloud Filestore
Block Storage	Amazon Elastic Block Store (EBS)	Azure Managed Disks	Google Cloud Persistent Disks
Object Storage	Amazon Simple Storage Service (S3)	Azure Blob Storage	Google Cloud Storage
Archive Storage	Amazon Glacier	Azure Archive Storage	Google Cloud Storage Nearline/Archive

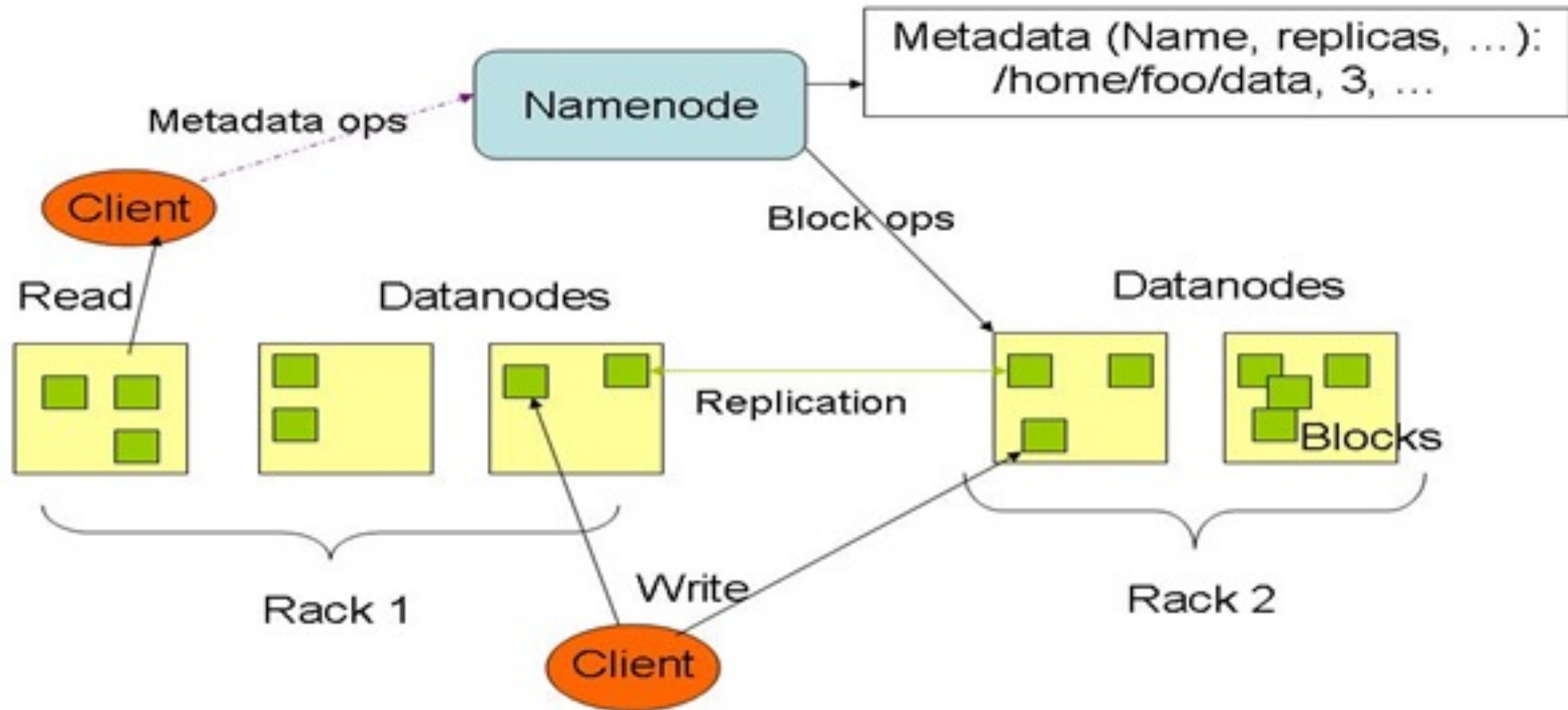
# Cache and Memory-Based Storage Systems

- RAM offers excellent latency and transfer speeds.
- Traditional RAM is vulnerable to data loss during power outages.
- RAM-based storage systems focus on caching for quick access and high bandwidth.
- Data should be written to more durable media for long-term retention.
- Example : Memcached, Redis

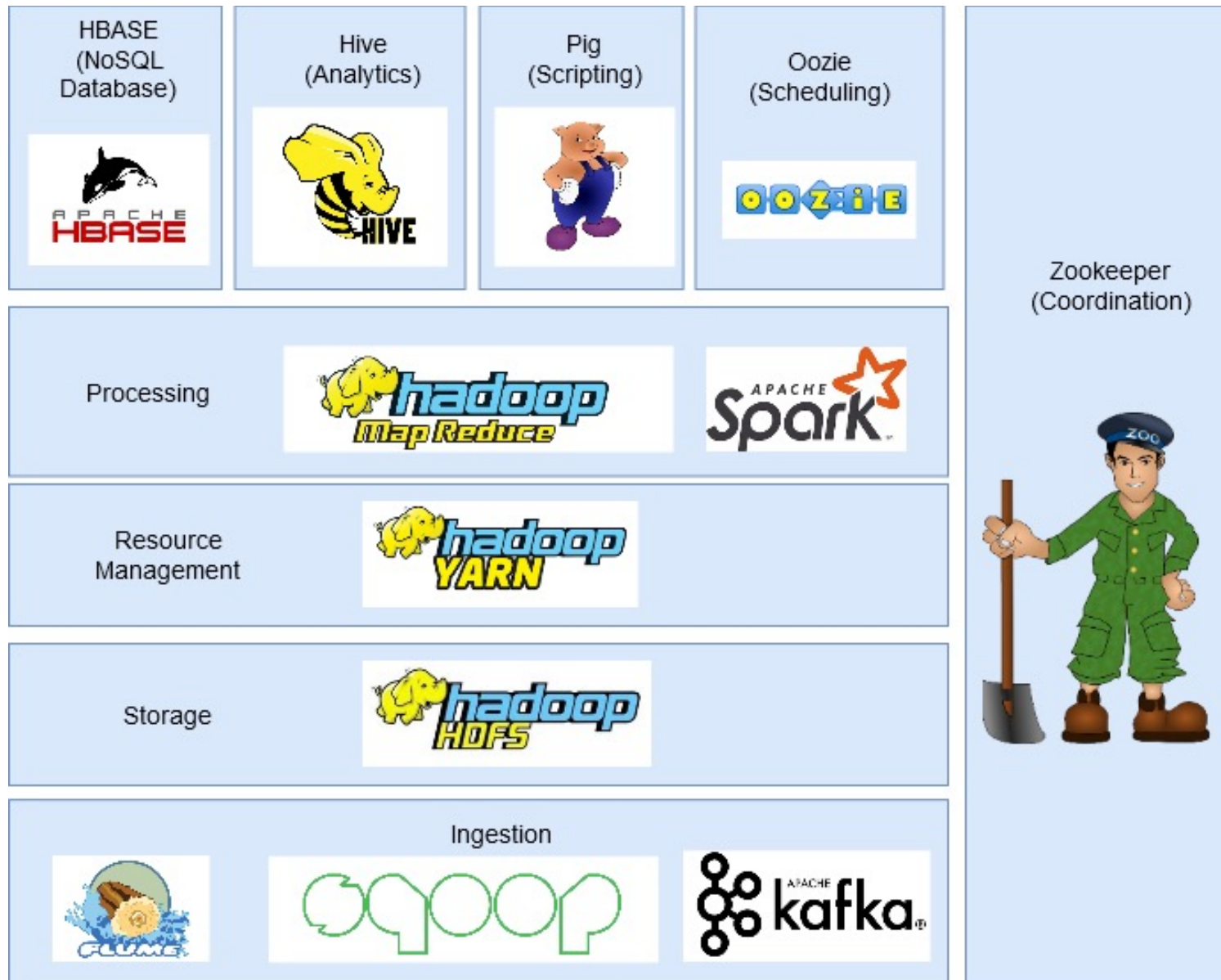
# Hadoop Distributed File System (HDFS)

- Hadoop was synonymous with big data in the recent past.
- Based on Google File System (GFS) and designed for MapReduce programming.
- Combines compute and storage on the same nodes, enabling in-place data processing.
- Breaks large files into blocks of data (a few hundred megabytes in size).
- Managed by the NameNode, which maintains file metadata and block locations.
- Each data block is replicated to three nodes for increased durability and availability.

# HDFS Architecture







# Streaming Storage

- Streaming data has different storage requirements than non-streaming data.
- Message queues store temporal data that is expected to disappear after a certain duration.
- Apache Kafka and similar scalable streaming frameworks now support extremely long-duration streaming data retention.
- Kafka enables indefinite data retention by pushing old, infrequently accessed messages down to object storage.
- Competitors like Amazon Kinesis, Apache Pulsar, and Google Cloud Pub/Sub also support long data retention.

# Data Storage Abstraction

- Data Warehouse
- Data Lake
- Data Lakehouse

# Data Abstraction Consideration

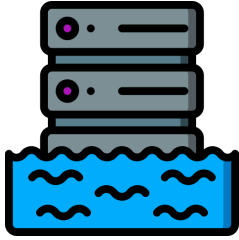
- **Purpose and Use Case:** Identify the purpose of storing the data and its use.
- **Update Patterns:** Is it optimized for bulk updates, streaming inserts, or upserts?
- **Cost:** Assess direct and indirect financial costs, time to value, and opportunity costs.
- **Separate Storage and Compute:** Trend towards separation, but many systems hybridize.

# Data Warehouse



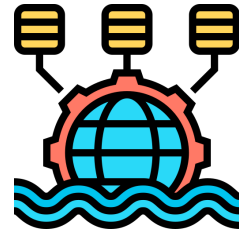
- OLAP data architecture for centralizing and organizing data.
- Evolved from conventional transactional databases to cloud data warehouses.
- Cloud data warehouses handle massive amounts of raw text and complex JSON data.

# Data Lake



- Originally conceived as a massive store for raw, unprocessed data.
- Initially built on Hadoop systems due to cheap storage and no proprietary MPP system cost overhead.

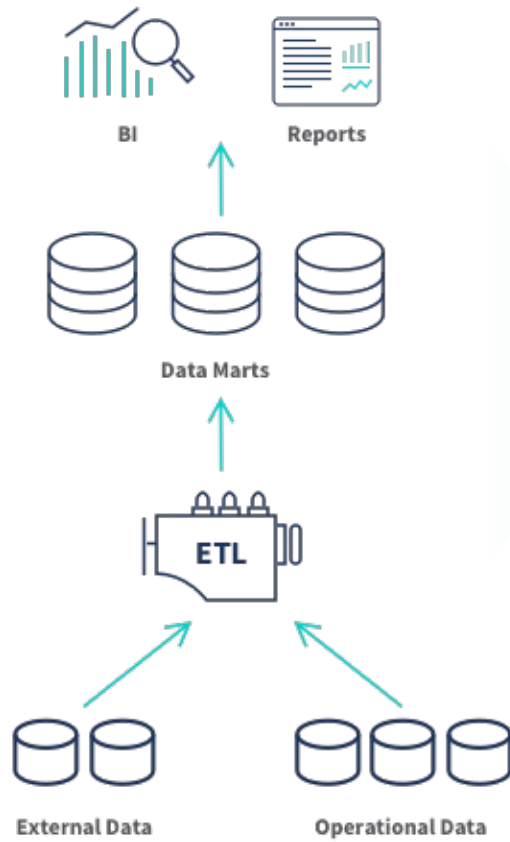
# Data Lakehouse



- Architecture combining data warehouse and data lake aspects.
- Stores data in object storage like a data lake.
- Features designed to streamline data management and create a data warehouse-like engineering experience.

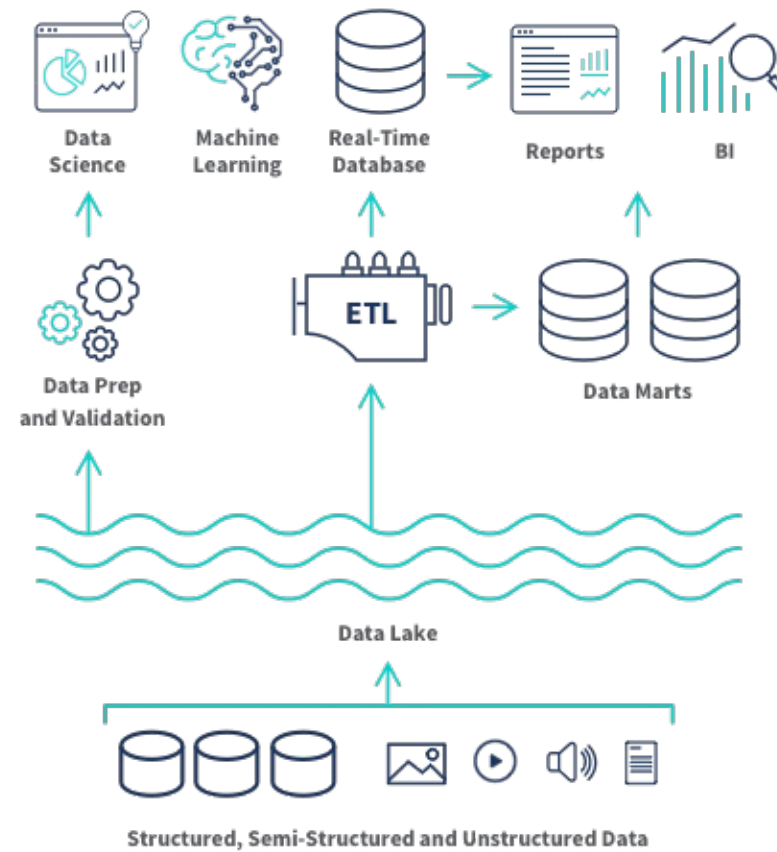
LATE 1980'S

## Data Warehouse



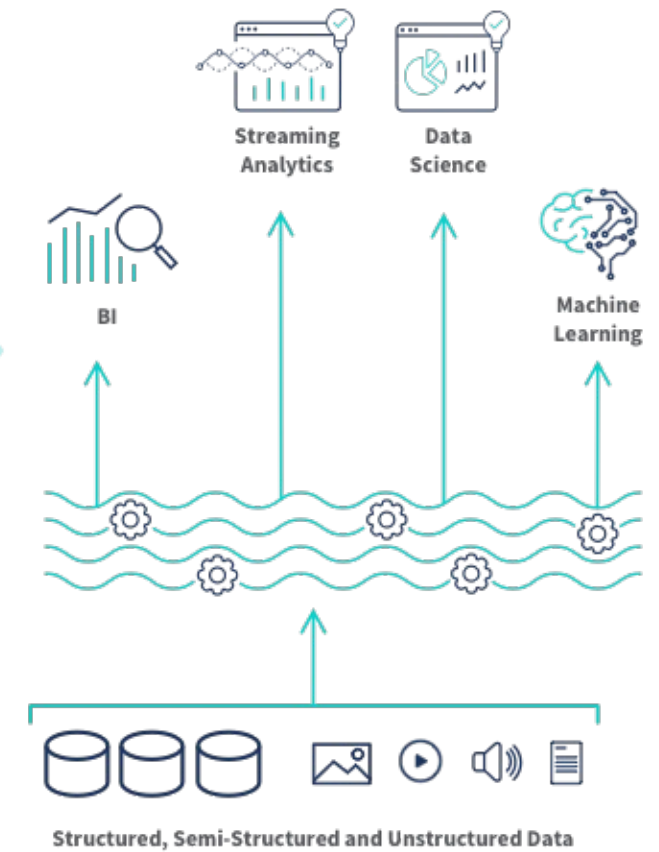
2011

## Data Lake



2020

## Lakehouse





Aspect	Data Warehouse	Data Lake	Data Lakehouse
Purpose	OLAP data architecture	Raw data storage	Combination of both
Data Organization	Structured data	Structured and unstructured data	Structured and unstructured data
Update Patterns	Batch updates	Bulk storage	Batch and incremental updates
Storage	Relational databases, MPP systems	Initially Hadoop, now migrating to cloud object storage	Metadata and file-management layer with object storage
Advantages	Fast query performance	Cost-effective for raw data storage	Flexible with structured querying and data management
Limitations	Scalability and locking limitations for large volumes of data	Limited advanced data management features	Still evolving, varying implementations

# Data Catalog

- Centralized metadata store for all data across an organization.
- Integrates with various systems and data storage abstractions.
- Works across operational and analytics data sources.
- Provides data lineage and data relationship presentation.
- Allows user editing of data descriptions.



DataHub



Amundsen

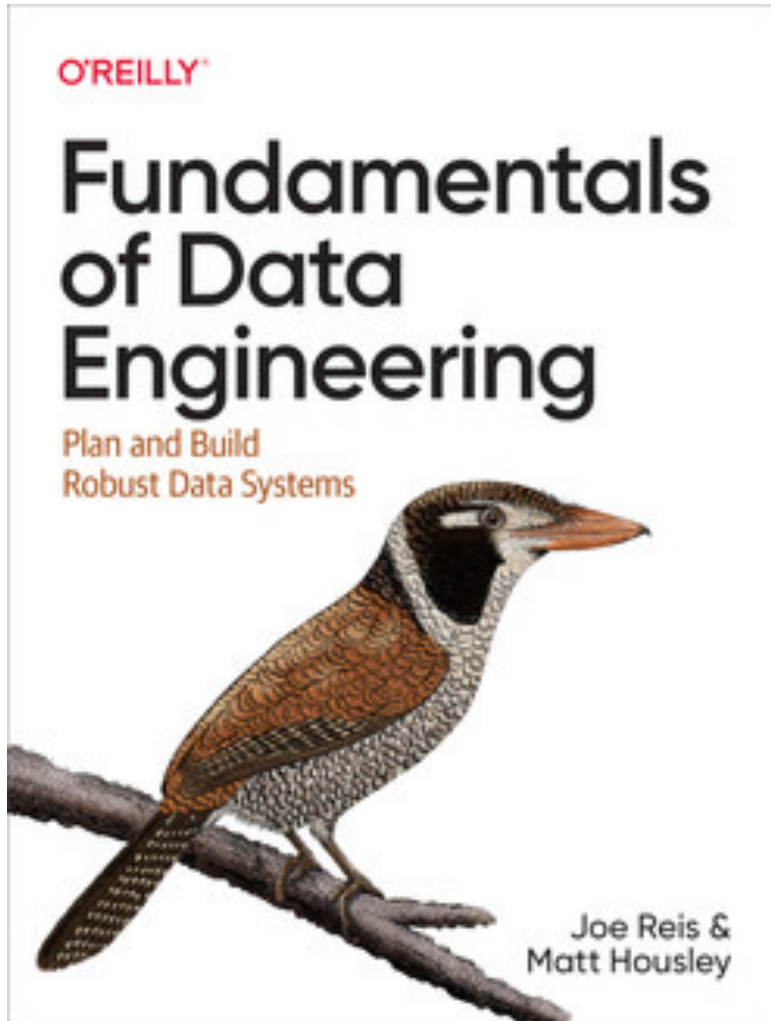


ckan

# Big Idea Trend in Storage

- Data Catalog
- Data Sharing
- Schema
- Separation of Compute and Storage
- Data Storage Lifecycle and Data Retention

# Ref : Additional Resource



- Fundamentals of Data Engineering
  - by Joe Reis, Matt Housley
  - Released June 2022
- Publisher(s): O'Reilly Media, Inc.
- ISBN: 9781098108304