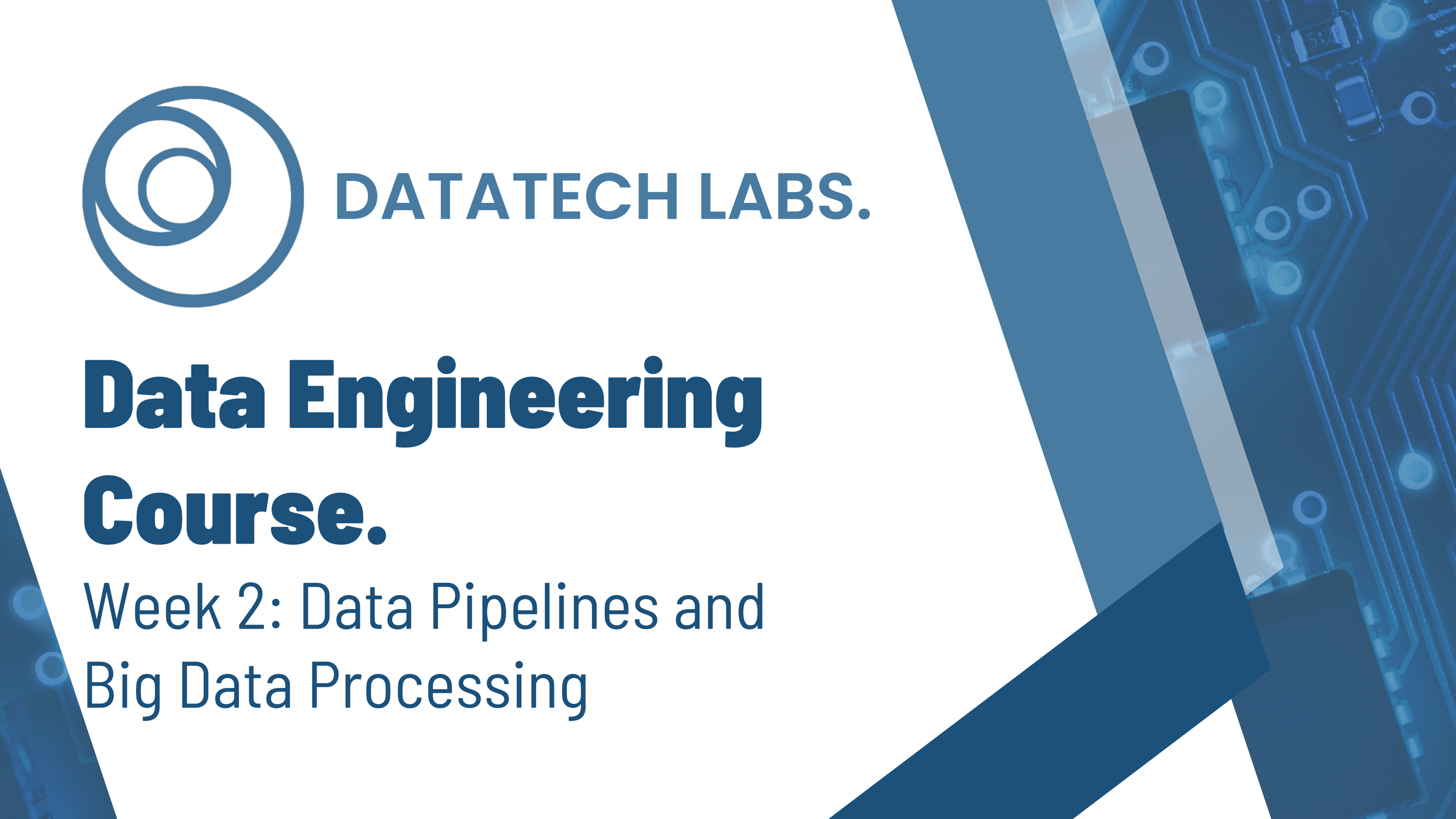


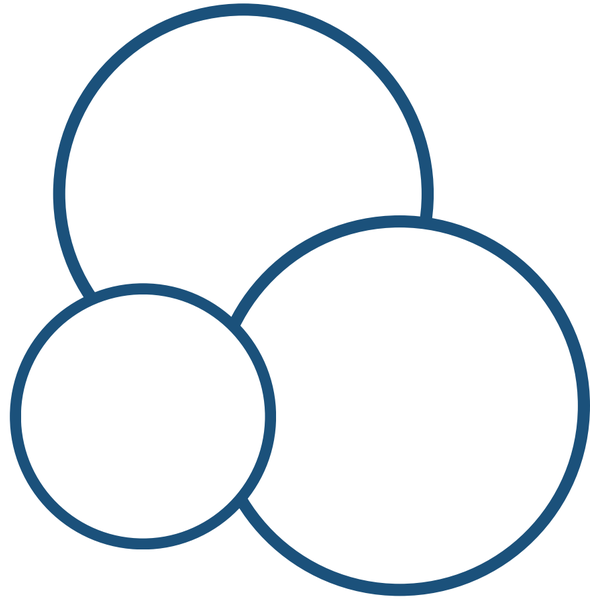
DATATECH LABS.

# Data Engineering Course.

## Week 2: Data Pipelines and Big Data Processing

# Outline: Week 2

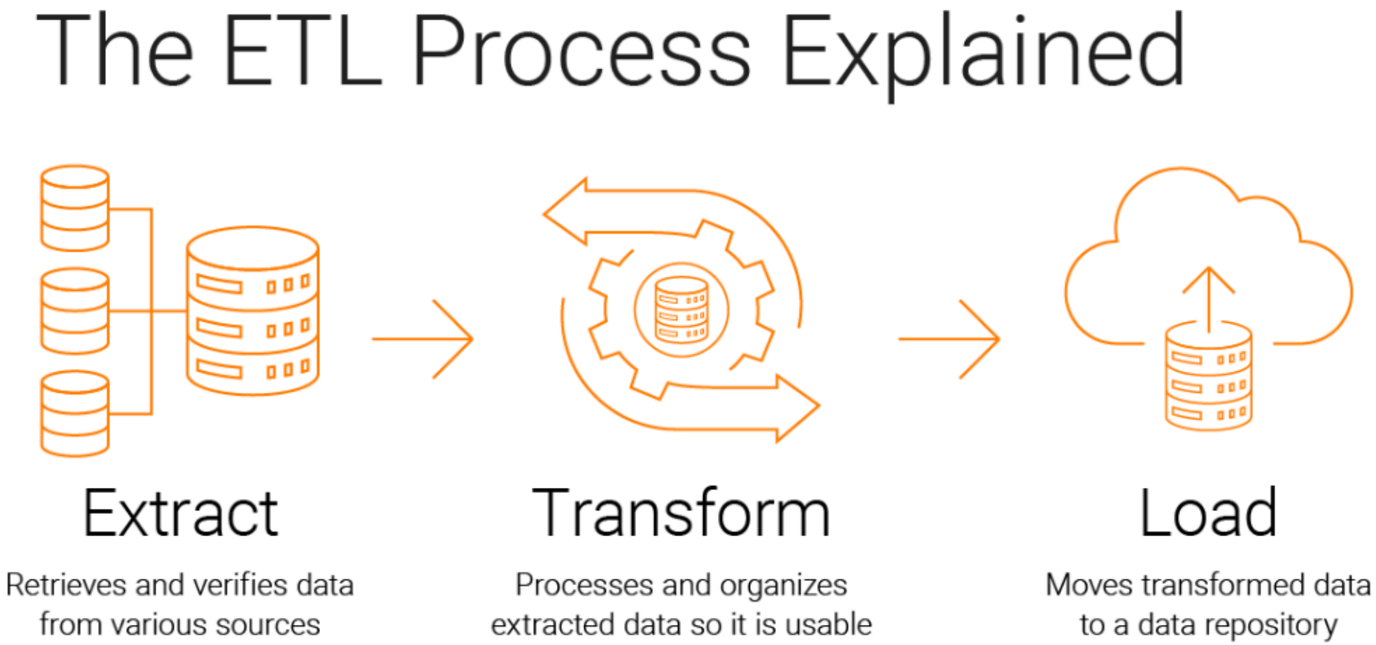
- ETL (Extract, Transform, Load) Processes
- Data Pipeline Design
- ETL Tools and Frameworks
- Introduction to Big Data
- Hadoop Ecosystem: HDFS, MapReduce, YARN
- Apache Spark



# ETL and Data Pipelines

# What is ETL?

- **Definition**: ETL stands for Extract, Transform, Load.
- It is a data integration process that involves:
  - **Extract**: Pulling data from various sources.
  - **Transform**: Converting the data into a format suitable for analysis.
  - **Load**: Storing the transformed data into a data warehouse or database.



# The Importance of ETL in Data Engineering

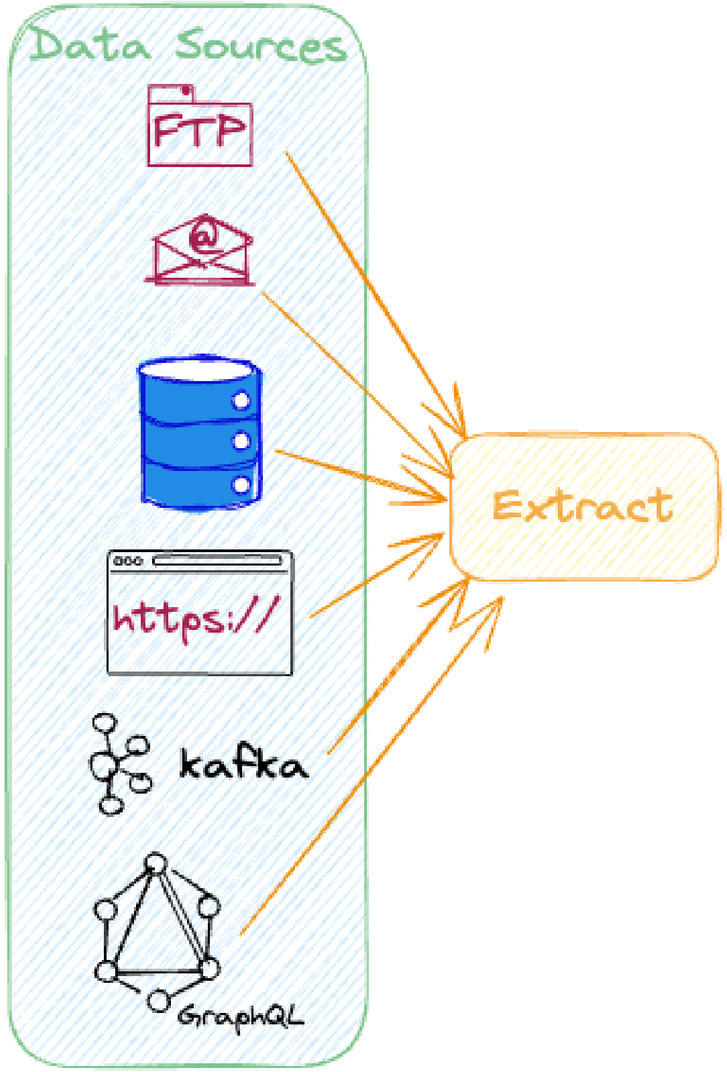
- ETL processes are critical for ensuring that data is accessible, accurate, and usable for decision-making.
- They allow businesses to integrate data from disparate sources and prepare it for analysis.

- **Discussion Point**: What are the potential challenges in each step of the ETL process?

# Components of ETL: Extraction

- Data Extraction:
  - Common sources: databases queries, APIs, flat files, etc.
  - Data extraction techniques:
    - Full extraction
    - incremental extraction
    - Update notification

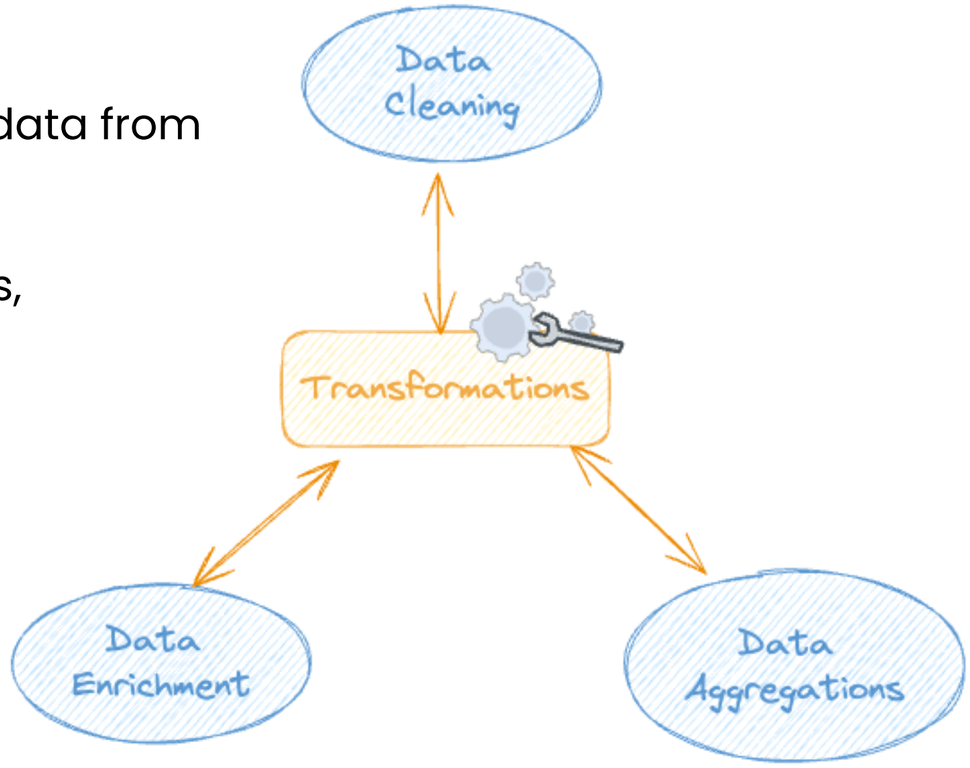
- Challenges:
  - Data volume and velocity (Complex data sources)
  - Variety of data sources for the same application
  - Data format inconsistencies
  - Data quality issues at source



# Components of ETL: Transformation

- Types of Transformations:
  - Data cleaning: Removing duplicates, handling missing values.
  - Data standardization: Consistent naming conventions, Data type conversion, ..
  - Data enrichment: Adding context to raw data, combining data from multiple sources, adding derived columns.
  - Data transformation: Aggregation, sorting, joining datasets,

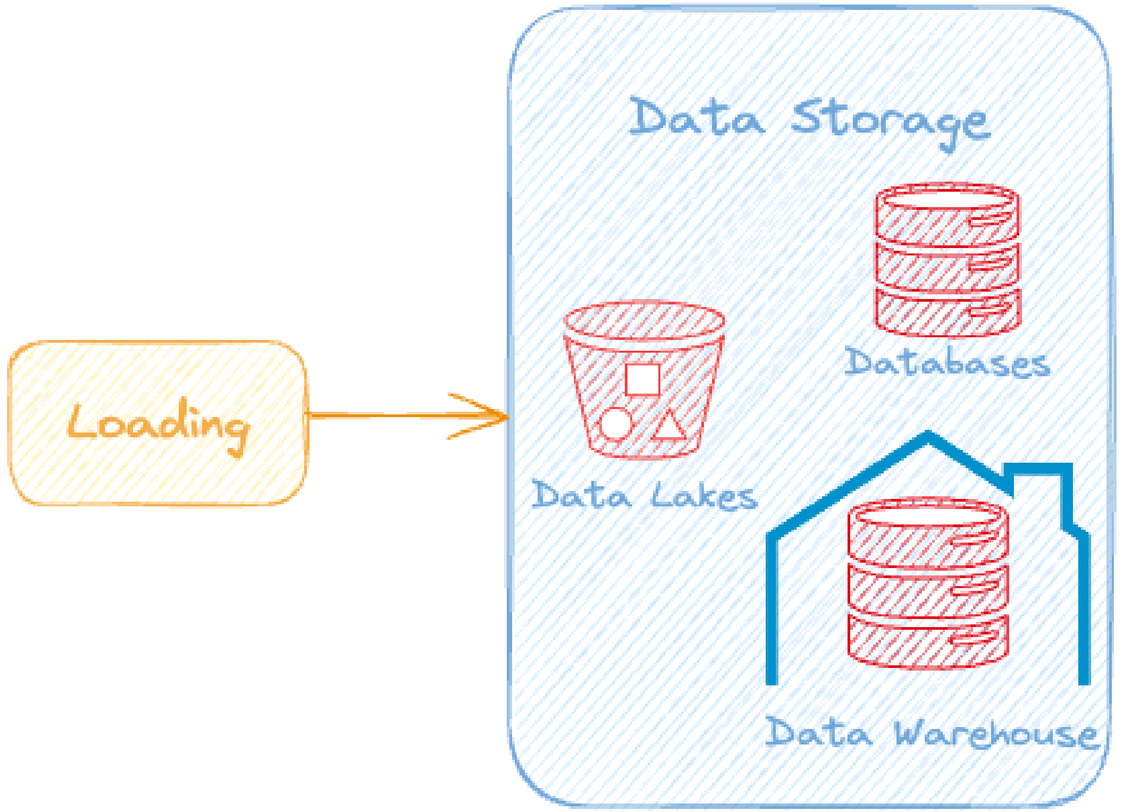
- Challenges:
  - Handling missing data
  - Dealing with duplicates.



# Components of ETL: Loading

- Loading data into target systems like:
  - Databases
  - Data warehouses
  - Data lakes.

- Data Loading Considerations
  - Data volume
  - Loading frequency
  - Target system constraints
  - Data integrity and consistency
  - Error handling and recovery



# ETL Tools and Frameworks

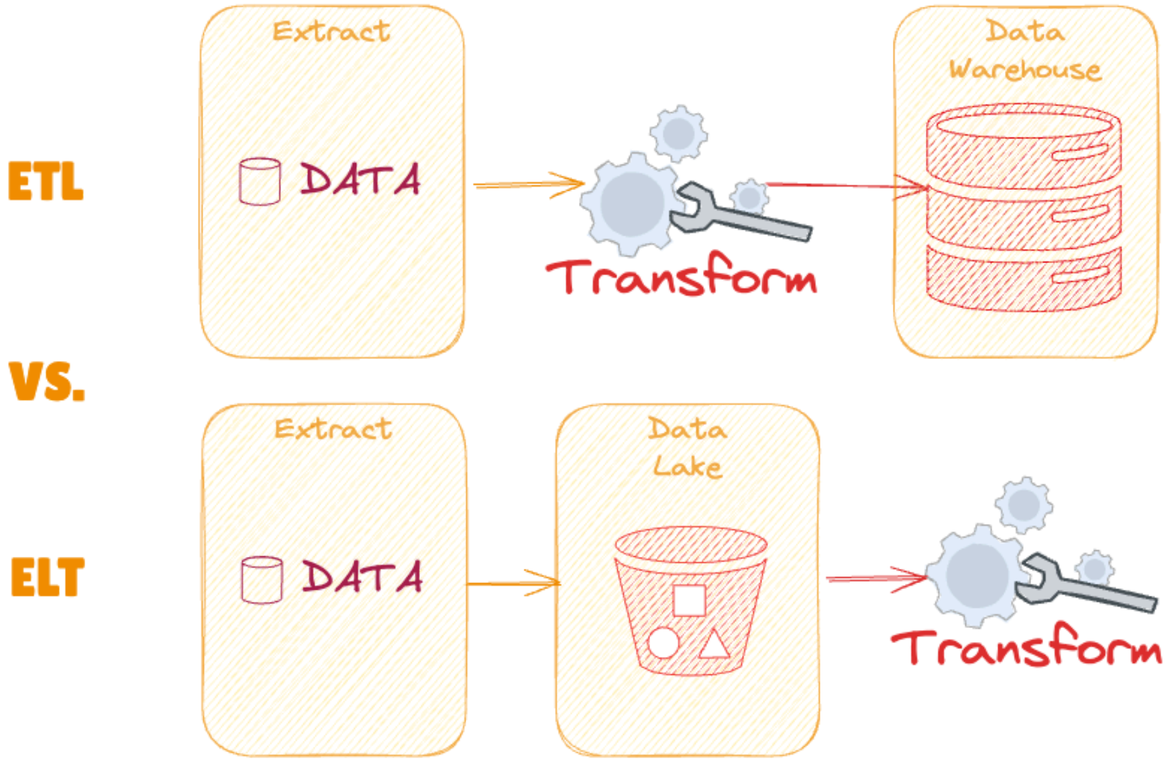
- **Native Code script**: Using one of the programming languages to write the ETL logic, like Python, or Java.
- **Apache NiFi**: A powerful data integration tool that supports data routing, transformation, and system mediation.
- **Talend**: An open-source ETL tool that provides a comprehensive set of tools for data integration.
- **Informatica**: A widely used enterprise ETL tool that provides data integration solutions for various types of data.




# ETL vs. ELT

- ETL: Data is transformed before loading into the data warehouse.
- ELT: Data is loaded first, and then transformations are applied.

- Discussion: When to choose ETL over ELT and vice versa?

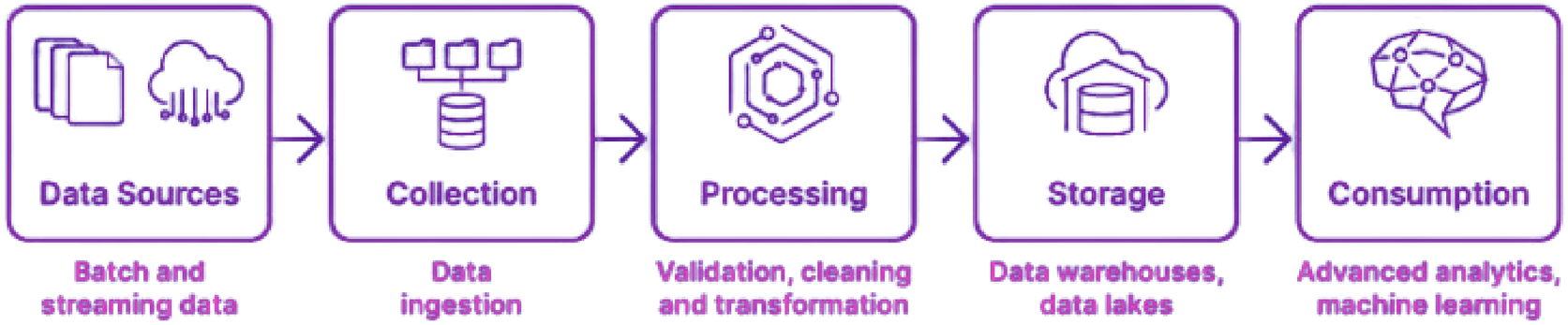


# Best Practices in ETL

- Maintain a robust logging mechanism.
- Implement retry mechanisms.
- Ensure data validation at each stage.
- Automate, automate, automate
- **Use parallel processing**

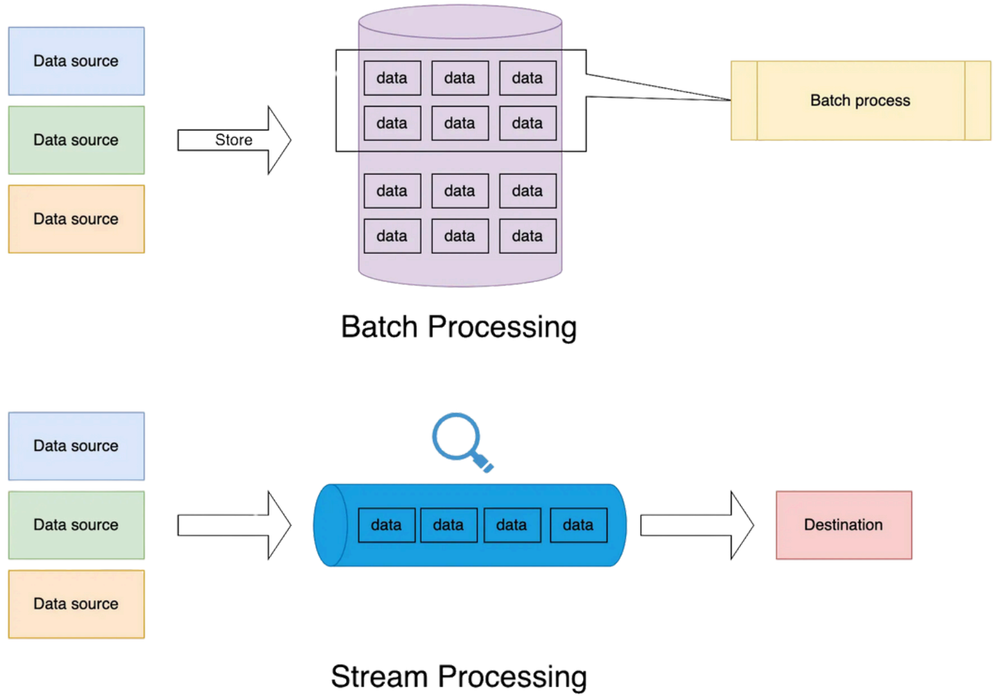
# Data Pipeline Design

- **Definition**: A data pipeline is a series of data processing steps that collect raw data from different sources, process it, and deliver it to a destination system.
- Pipeline Stages:
  - **Ingestion**: Collecting raw data from various sources.
  - **Processing**: Transforming the data into a usable format.
  - **Storage**: Storing the processed data in a data warehouse, database, or data lake.
  - **Analysis (Consumption)**: Making the data available for analysis and reporting.



# Batch vs. Streaming Data Pipelines

- **Batch Processing**: Collecting and processing data in large blocks at scheduled intervals.
  - Process data in chunks (Daily reports, weekly analytics).
  - Typically run at scheduled intervals
  - Suitable for large volumes of data
- **Streaming Processing**: Processing data as it is generated, providing immediate insights (Real-time monitoring, fraud detection).
  - Process data in real-time or near real-time
  - Continuous data flow
  - Low-latency processing
- **Lambda architecture** (combination of batch and streaming)



# Case Study: ETL in a Retail Environment

- Scenario:
  - A retail company needs to integrate data from online sales, physical stores, and customer feedback systems.
  - **Discussion Point**: How would you design an ETL pipeline for this scenario?

# Challenges in ETL Processes

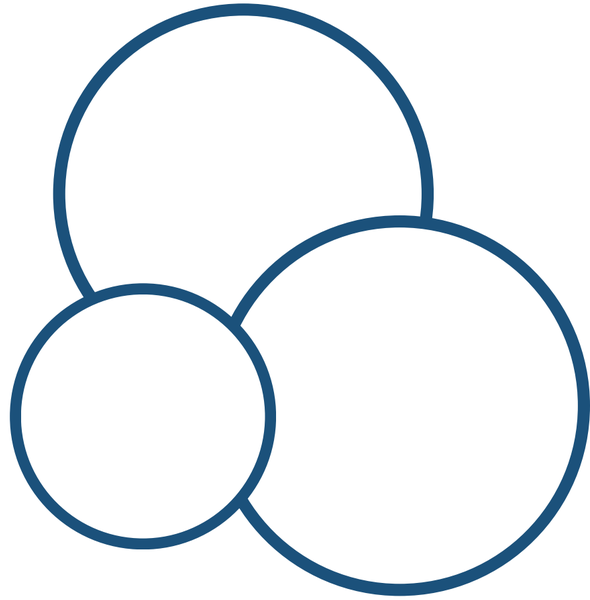
- Data Quality Issues
- Handling Large Volumes of Data
- Managing Multiple Data Sources

- **Discussion Point**: How can these challenges be mitigated in a production environment?

# Hands-on:

Implementing ETL Pipeline.

# Q&A and Discussion

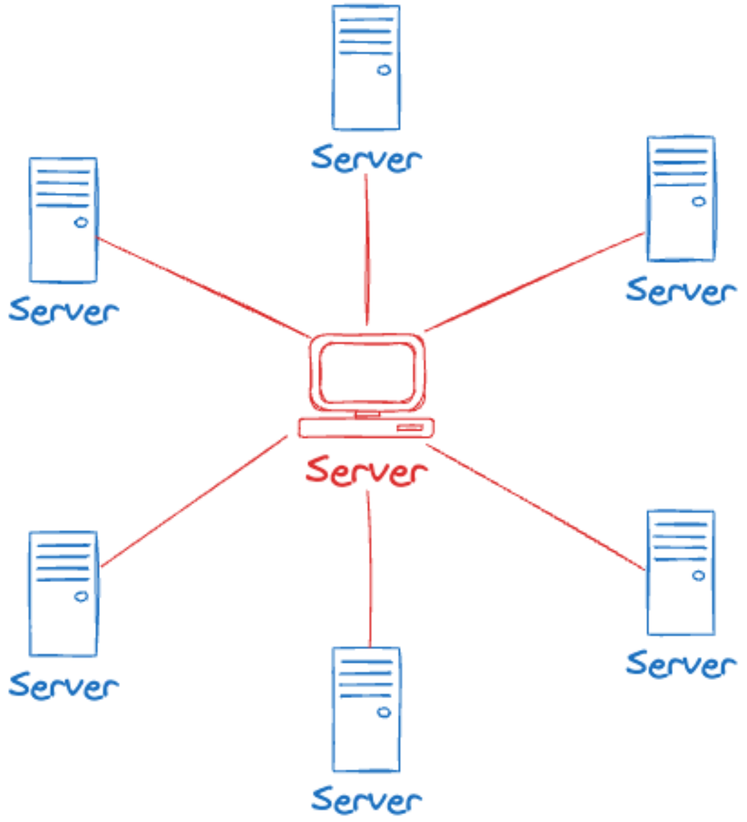


# Big Data Technologies and Ecosystems

# Introduction to BIG DATA

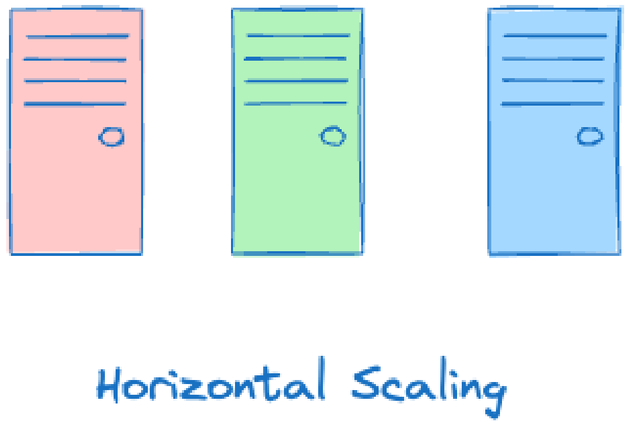
# Introduction to Distributed Computing

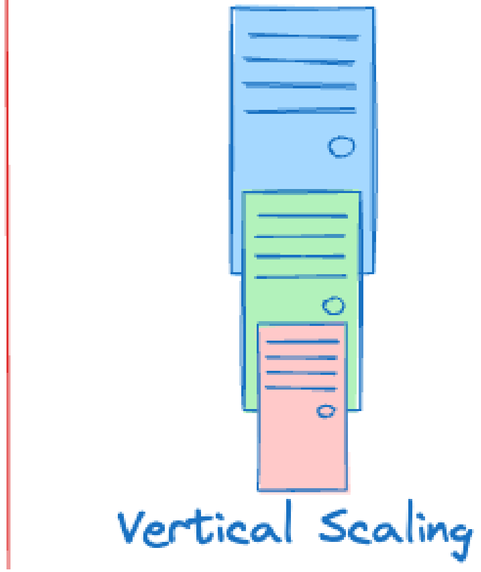
- **Definition**: A model in which components of a software system are shared among multiple computers to improve efficiency and performance.
- Why Distributed Computing?
  - Handle large-scale data processing
  - Improve fault tolerance and reliability
  - Enable parallel processing
  - Provide scalability



# Key Concepts & Scalability in Distributed Computing

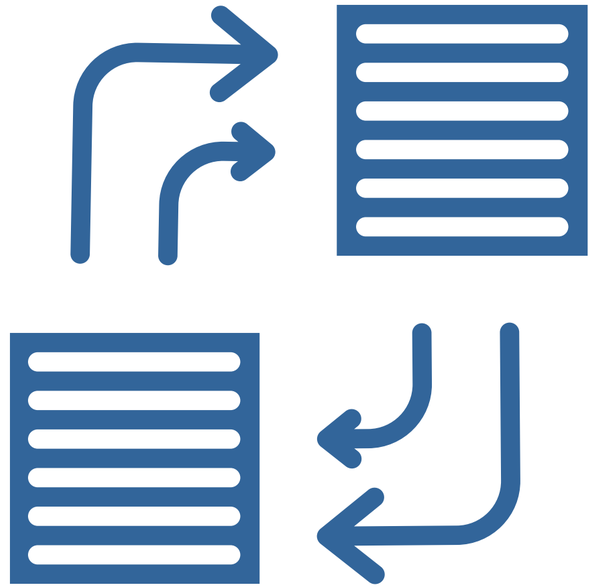
- Key Concepts:
  - Scalability
  - Fault tolerance
  - Consistency
  - Availability
  - Partition tolerance
- Scaling methods:
  - Vertical scaling (scale-up)
  - Horizontal scaling (scale-out)



# Challenges in Distributed Computing

- Network latency
- Synchronization
- Partial failures
- Consistency maintenance
- Load balancing



# Introduction to Big Data

- **Definition**: Big Data refers to the vast volumes of data generated by various sources that are too large, complex, and dynamic for traditional data processing tools.
- Overview of the components of a big data ecosystem:
  - Data sources
  - Ingestion
  - Processing
  - Storage
  - And analysis.

# The Big Data Ecosystem

- Explanation: Discuss the 5 V's of Big Data:
  - Volume
  - Velocity
  - Variety
  - Veracity
  - Value

# Introduction to Hadoop

# Hadoop Ecosystem Overview

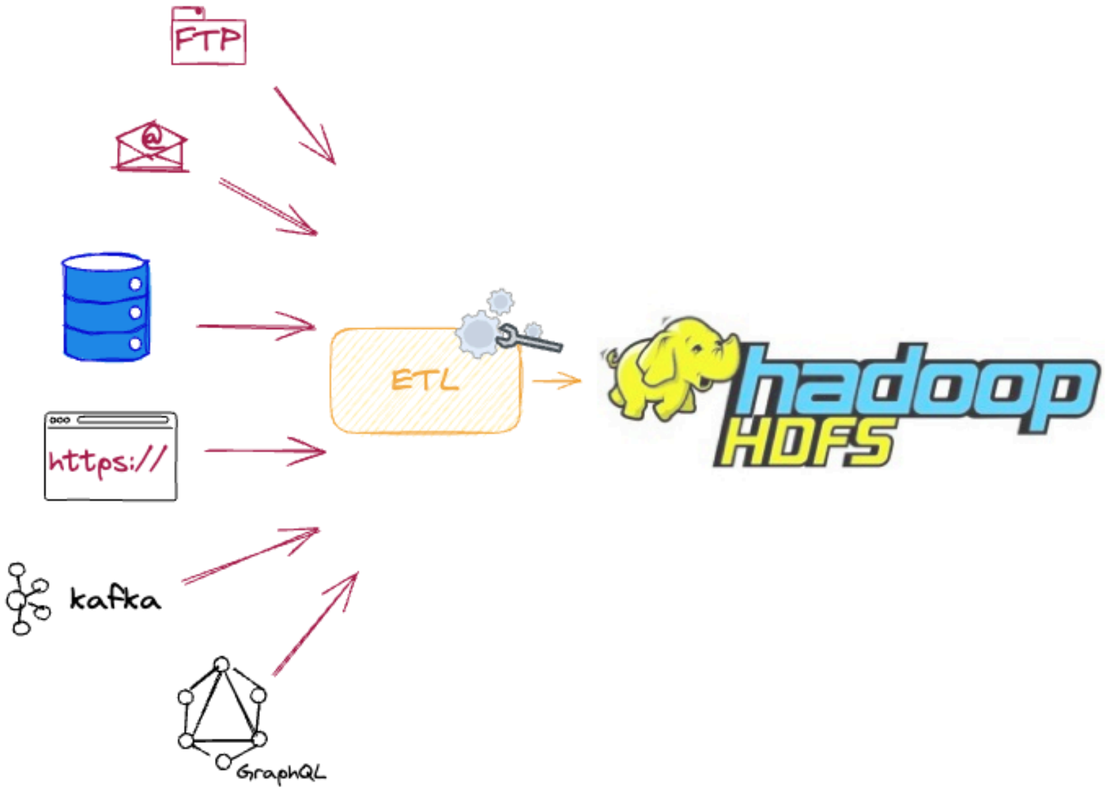
- Hadoop is an open-source framework for distributed storage and processing of large data sets.

- Components:
  - **HDFS**: *“Hadoop Distributed File System”*. for distibuted storage.
  - **MapReduce**: A programming model for processing large data sets.
  - **YARN**: *“Yet Another Resource Negotiator”*, manages cluster resources.



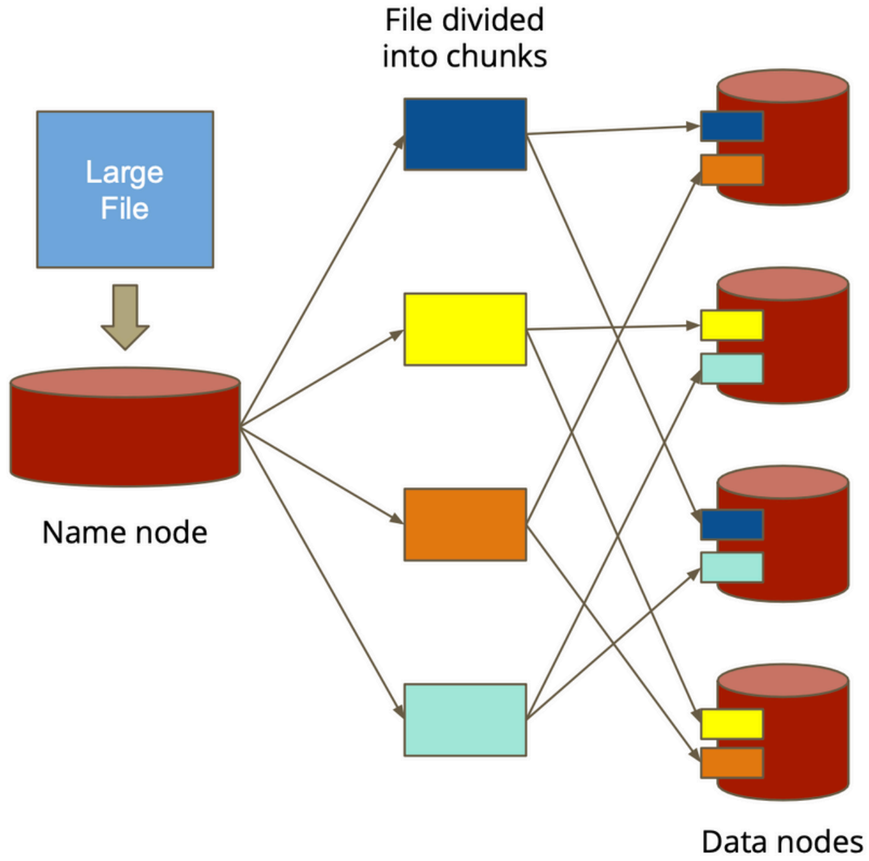
# HDFS: The Storage Backbone

- HDFS is designed to store very large files across multiple machines. It is a distributed storage system for big data
- Data is split into blocks and distributed across the cluster.
- Architecture:
  - NameNode (manages filesystem namespace)
  - DataNodes (store actual data)



# HDFS Features

- Large file storage
- Streaming data access
- Fault tolerance
- Portability across heterogeneous hardware and software platforms

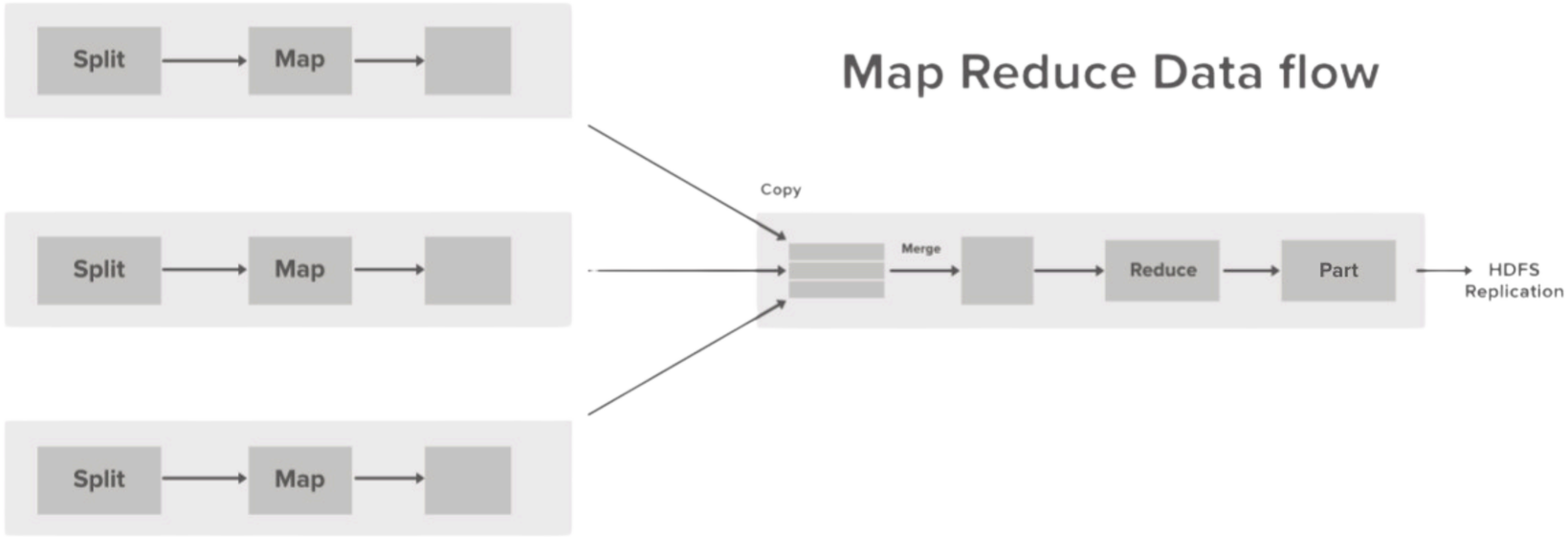


# Big Data Storage Solutions

- On-premise:
  - HDFS: Hadoop’s distributed file system.
- Cloud-based:
  - Amazon S3: Cloud storage service.
  - Google Cloud Storage: Object storage service.

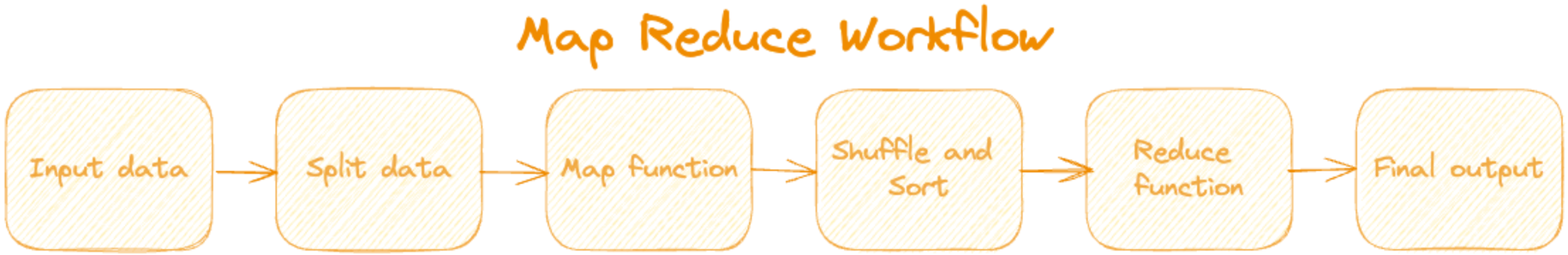

# MapReduce: Distributed Data Processing

- Programming model for processing large data sets.
- Two main tasks (steps):
  - **Map Phase**: Processes input data and generates key-value pairs.
  - **Reduce Phase**: Aggregates key-value pairs to produce the final output.



# MapReduce Workflow

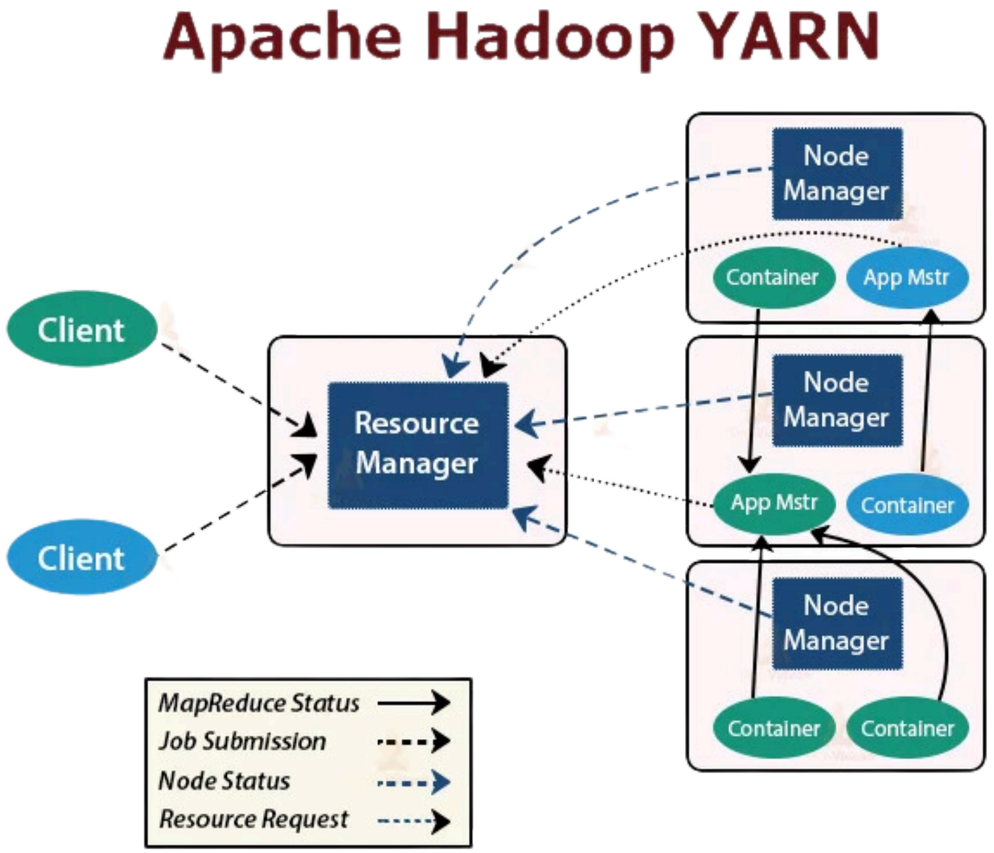
1. Input data
2. Split data
3. Map function
4. Shuffle and Sort
5. Reduce function
6. Final output



# YARN: Resource Management in Hadoop

- YARN: *"Yet Another Resource Negotiator"*
- Cluster resource management and job scheduling.

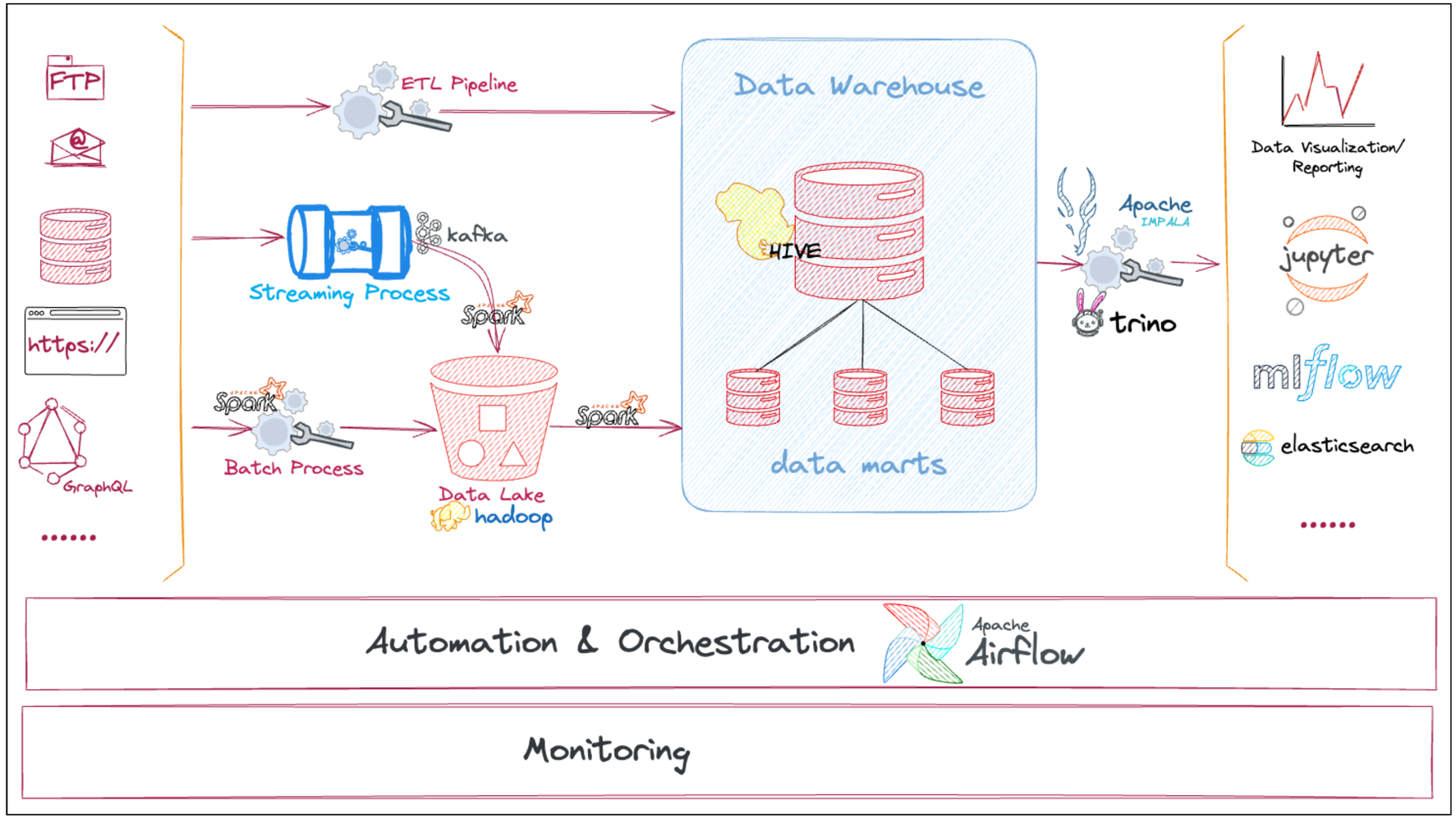
  Means resource allocations for different applications.
- Components:
  - ResourceManager: Global resource scheduler
  - NodeManager: Per-node agent
  - ApplicationMaster: Per-application manager



# Hadoop Ecosystem

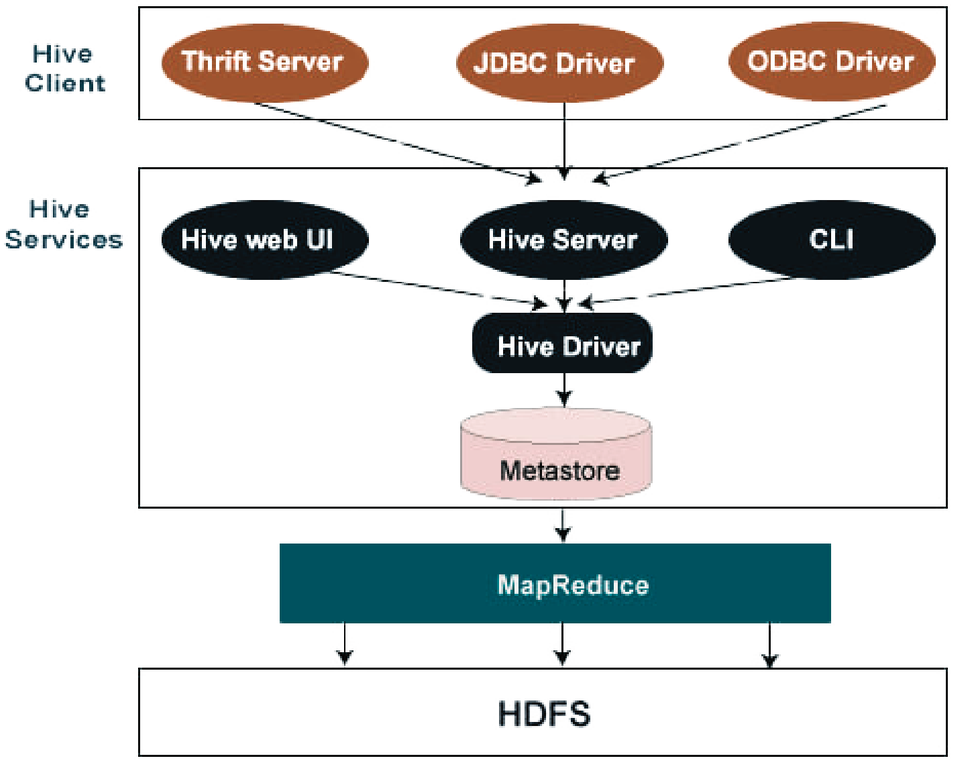
# Hadoop Ecosystem

- **Hive**: SQL-like queries.
- **HBase**: NoSQL database.
- **Spark**: In-memory data processing.
- **Kafka**: Distributed streaming platform.
- **Impala/Trino**: Querying engines.



# Big Data Warehousing

- **Apache Hive** is a data warehouse infrastructure built on top of Hadoop.
- Features:
  - SQL-like query language (HiveQL).
  - Supports complex queries on large datasets.



# NoSQL Database in Hadoop Ecosystem

- **Apache HBase** is a distributed, scalable, NoSQL database built on top of HDFS.
- Features:
  - Real-time read/write access to large datasets.

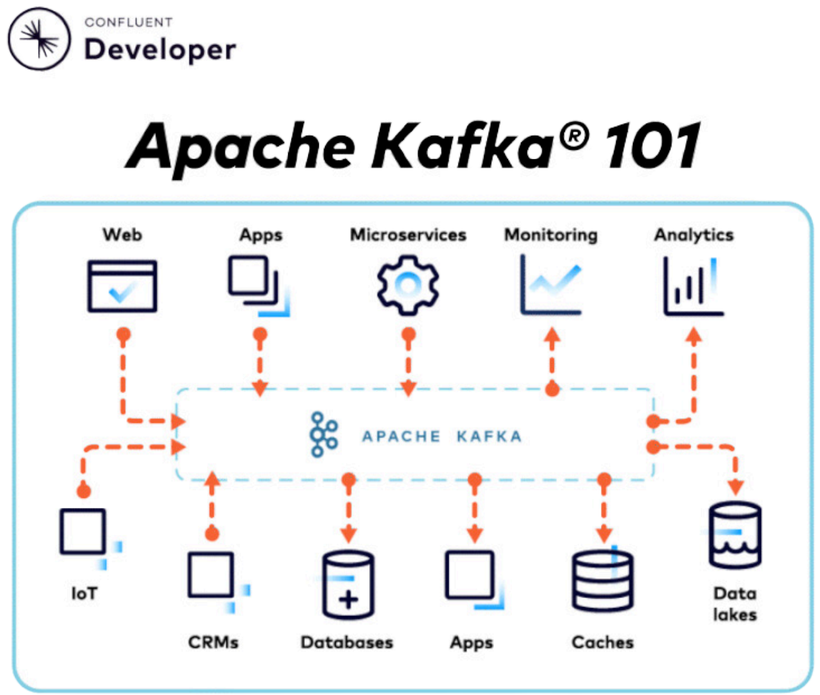
# Apache Spark: In-Memory Data Processing

- **Spark** is a fast, in-memory data processing engine that allows for fast and/or real-time analytics.

- Spark vs. Hadoop:
  - **Hadoop MapReduce**: Disk-based processing, suited for batch processing.
  - **Apache Spark**: In-memory processing, suited for both batch and real-time processing.

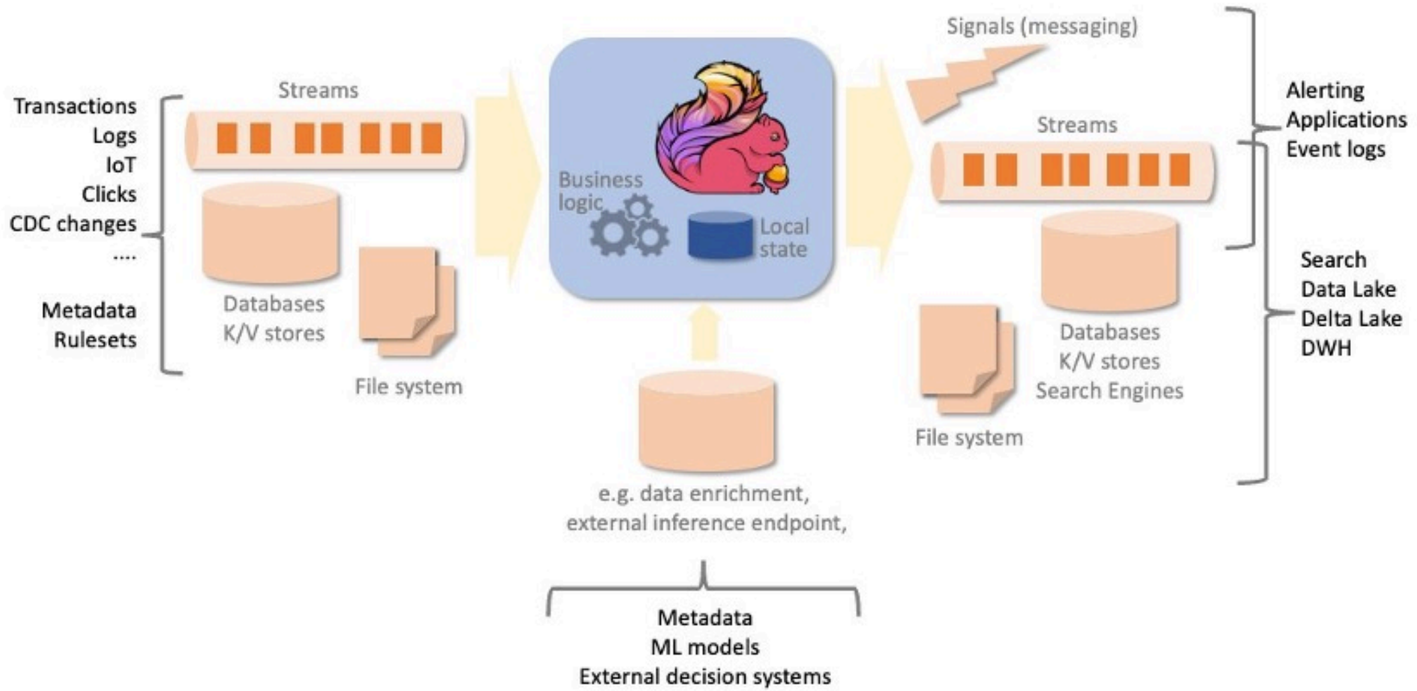
# Apache Kafka: Distributed Streaming Platform

- **Kafka** is a distributed messaging system for data transfer.
- Used as streaming platform for building real-time data pipelines.
- Components:
  - Producers: Publish messages to Kafka topics.
  - Consumers: Subscribe to Kafka topics to process messages.



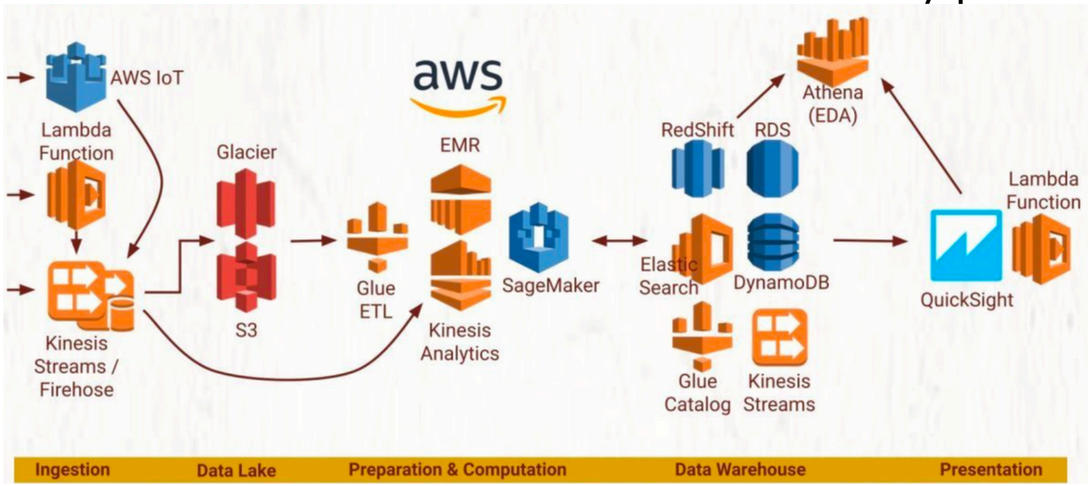
# Apache Flink: Real-Time Stream Processing

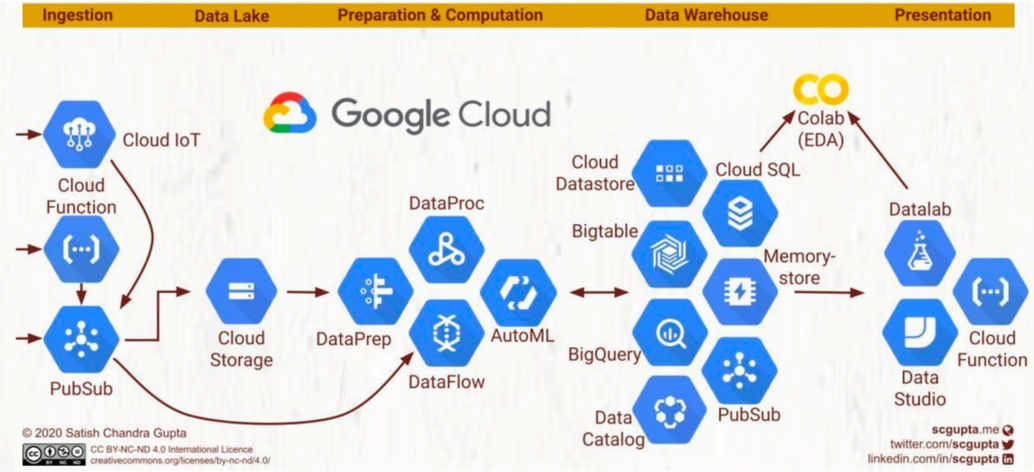
- **Flink** is a stream processing framework for real-time data processing.
- Features:
  - Stateful computation, low-latency processing.
  - Event-driven applications

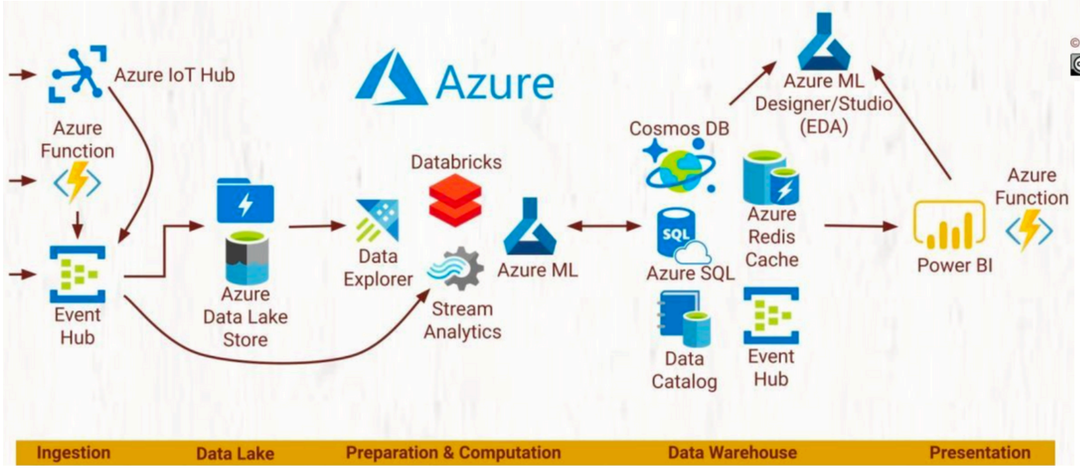


# Big Data on-Cloud Services

- Cloud Computing providers (ex: AWS, GCP, MS Azure) often utilize a “software as a service” model to allow customers to easily process normal and big data.



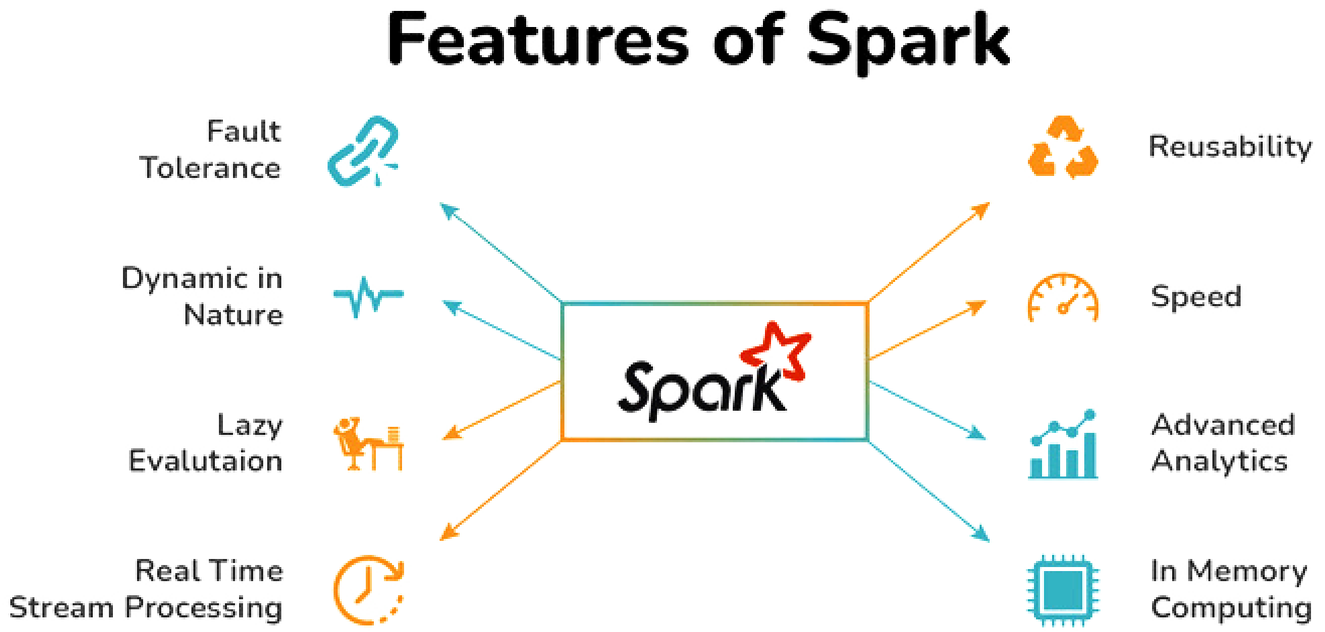
# Hands-on:

Set up a Hadoop cluster on your local machine.
Implement a MapReduce job to process data on HDFS.

# Apache Spark

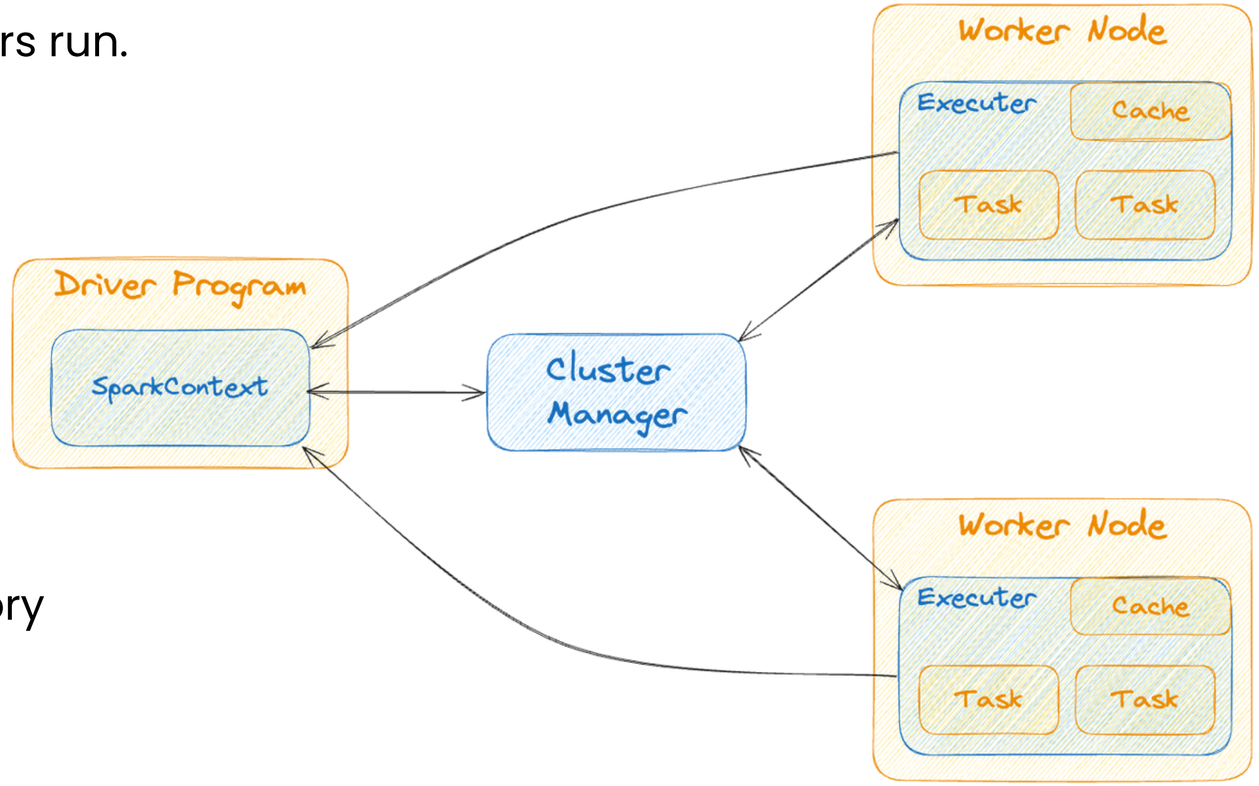
# Introduction to Apache Spark

- Key features:
  - In-memory computation
  - Fault tolerance
  - Lazy evaluation
  - Support for multiple languages (Scala, Java, Python, R)



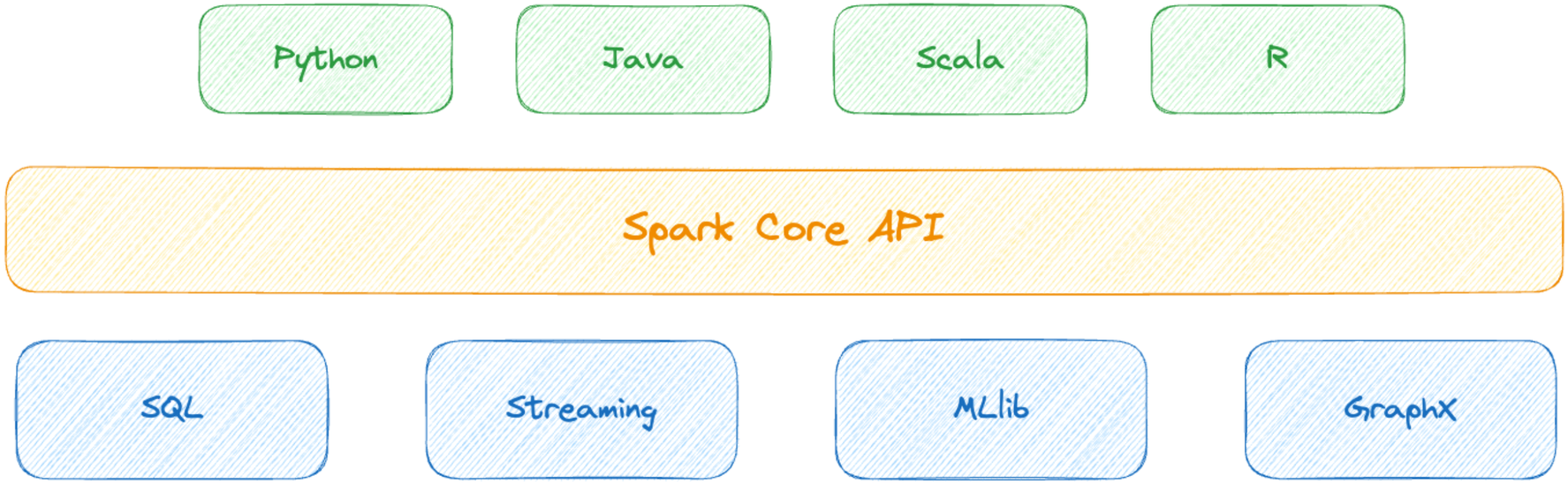
# Spark Architecture

- **Driver Program**: The process running the main() function of the application and creating the SparkContext.
- **Cluster Manager**: is responsible for allocating resources and managing the cluster on which the Spark application runs.
- **Worker Nodes**: The slave nodes where the executors run.
- **Executors**:
  - Are worker processes responsible for executing tasks.
  - Are launched on worker nodes and communicate with the driver program and cluster manager.
  - Run tasks concurrently and store data in memory or disk for caching and intermediate storage.



# Spark Components

- Spark Core
- Spark SQL
- Spark Streaming
- MLlib (Machine Learning)
- GraphX (Graph computation)



# Spark RDDs (Resilient Distributed Datasets)

- Definition: Fundamental data structure of Spark
- Properties:
  - Immutable
  - Partitioned
  - Ability to be operated on in parallel

```python
from pyspark import SparkContext, SparkConf

conf = SparkConf() \
    .setAppName("wordcount") \
    .setMaster("local[*]")

sc = SparkContext(conf=conf)
data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
rdd = sc.parallelize(data)
print(rdd.collect())
# [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

# Spark DataFrames and Datasets

- **DataFrames**:
  - Distributed collection of data organized into named columns
  - Create by loading structured data from various sources like.
  - Perform operations similar to a SQL table by utilizing DataFrame APIs.
  - Benefit from the optimization techniques incorporated within DataFrames.
- **Datasets**:
  - Provides a type-safe, object-oriented programming interface
  - Perfect for complex operations.
  - It combines the advantages of RDDs and DFs.
  - It excels efficiently at grouping data.

```
1 df = spark.read.option('header', 'true').csv('test.csv')
```

```
1 df.show()
```

```
+--------+---+-----+------+
|Category| ID|Truth| Value|
+--------+---+-----+------+
|       A|  1| True|121.44|
|       B|  2|False|300.01|
|       C|  3| null| 10.99|
|       E|  4| True| 33.87|
+--------+---+-----+------+
```

# Best Practices in Big Data Engineering

- Optimize storage and processing costs.
- Ensure data security and privacy.
- Monitor and maintain data pipelines.

- **Discussion**: Challenges in implementing best practices in big data.

# Case Study: Building a Big Data Ecosystem

- Scenario:
  - Setting up a big data ecosystem for a retail business.
  - **Discussion**: Components required and the tools to be used.

# Hands-on:

## Spark Basics

# Q&A and Discussion

# Spark RDDs (Resilient Distributed Datasets)

- Definition: Fundamental data structure of Spark
- Essentially refers to a distributed collection of data records
- Resilience Advantage is their fault-tolerant nature

- Properties:
  - Immutable
  - Partitioned
  - Ability to be operated on in parallel

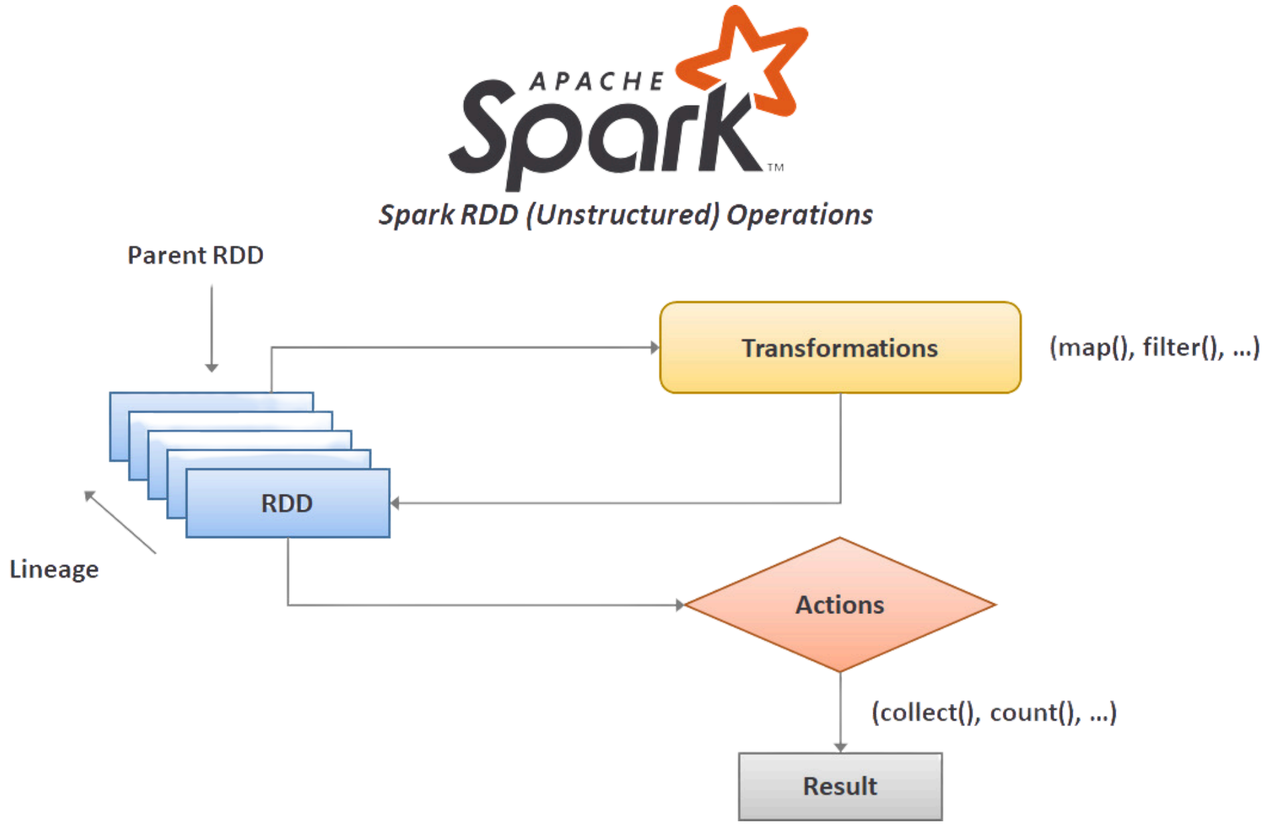
***should move to week2***

```python
from pyspark import SparkContext, SparkConf

conf = SparkConf() \
 .setAppName("parallelize_example") \
 .setMaster("local[*]")

sc = SparkContext(conf=conf)
data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
rdd = sc. parallelize (data)
print(rdd.collect())

```

# RDD Operations

- Transformations (lazy operations):
  - map, flatMap, filter, distinct, sample
- Actions (trigger execution):
  - collect, count, first, take, reduce

***should move to week2***



# MEET OUR TEAM

**Omar AlSaghier**

Sr. Data Engineer

# DATATECH LABS.

# THANK YOU

## OUR CONTACT


DataTechLabs

datatechlabs.ai@gmail.com

datatechlabs.ai

Amman, Jordan