

## Unit # 2

### - Introduction to Hadoop

# Introduction - Hadoop

- Apache Hadoop is an open-source framework that allows for the distributed processing of large datasets across clusters of computers using simple programming models. It is designed to scale up from a single server to thousands of machines, with a high degree of fault tolerance.

There are four core components of Hadoop.

1. HDFS (Hadoop Distributed File System)
2. YARN (Yet Another Resource Negotiator)
3. Map Reduce
4. Hadoop Common

## Difference between RDBMS and Hadoop

RDBMS systems like MySQL, PostgreSQL, and Oracle are designed for real-time, transactional queries where low-latency access is critical whereas Hadoop and similar systems like HDFS, Hive, HBase are optimized for large-scale data processing and batch operations, focusing on handling big data and Online Analytical Processing workloads (OLAP) . The specific differences are discussed below :

### *1. Data Size and Type:*

RDBMS: Suited for structured, relational data; handles gigabytes to terabytes of data. Hadoop: Designed for both structured and unstructured data; capable of scaling to petabytes or exabytes of data.

# Introduction – Hadoop ( contd)

## *2. Read/Write Performance:*

RDBMS: Optimized for fast, small read/write operations; handles real-time transactional queries efficiently. Data retrieval and modification are quick due to the use of B-trees, indexes, and row-based storage.

Hadoop: Optimized for sequential access and batch processing. It's not designed for real-time, frequent, or small transactions. The overhead of distributing tasks across the cluster makes it slower for individual queries but highly efficient for processing huge datasets.

## *3. Data Access Speed:*

RDBMS: Read Latency is typically in milliseconds to seconds for a single row or small dataset query. Write Latency as well in milliseconds to seconds depending on whether indexes, constraints, or transactions are involved. Eg. Queries on MySQL or PostgreSQL to retrieve a few thousand records can typically return results in less than 100 milliseconds.

Hadoop: Read Latency is in seconds to minutes due to its distributed nature and the overhead of accessing data stored across multiple nodes in HDFS. While write Latency involves replication and coordination across the cluster, so it can take seconds to minutes as well. Eg. A single query to retrieve data from HDFS using Hive or HBase could take 5–20 seconds for a moderately sized query (e.g., querying millions of rows).

# Introduction – Hadoop ( contd)

## 4. Concurrency :

RDBMS: Supports high concurrency, handling thousands of simultaneous small transactions efficiently with ACID compliance. Typically, RDBMS systems can handle hundreds to thousands of queries per second (QPS) for small transactional workloads. For eg. MySQL can handle 10,000+ QPS for simple reads/writes depending on the hardware and optimizations.

Hadoop: Hadoop is not designed for high-concurrency random access. It is optimized for batch processing and parallel processing over large datasets, so concurrent small queries will experience significant delays. For eg. A typical Hadoop MapReduce job could take several minutes to hours depending on the size of the data and the complexity of the task.

Activity	RDBMS	Hadoop
SELECT 1000 rows from a table with an index on the query key.	< 100 ms	10–20 sec
SELECT with a complex JOIN and aggregation over a 1 TB dataset.	Several hours.	10–30 minutes.
Real-time transactional query	Milliseconds	Minutes to hours

# Introduction – Hadoop ( contd)

## Ecosystem of Hadoop

*Oozie*: A workflow scheduler system to manage Hadoop jobs.

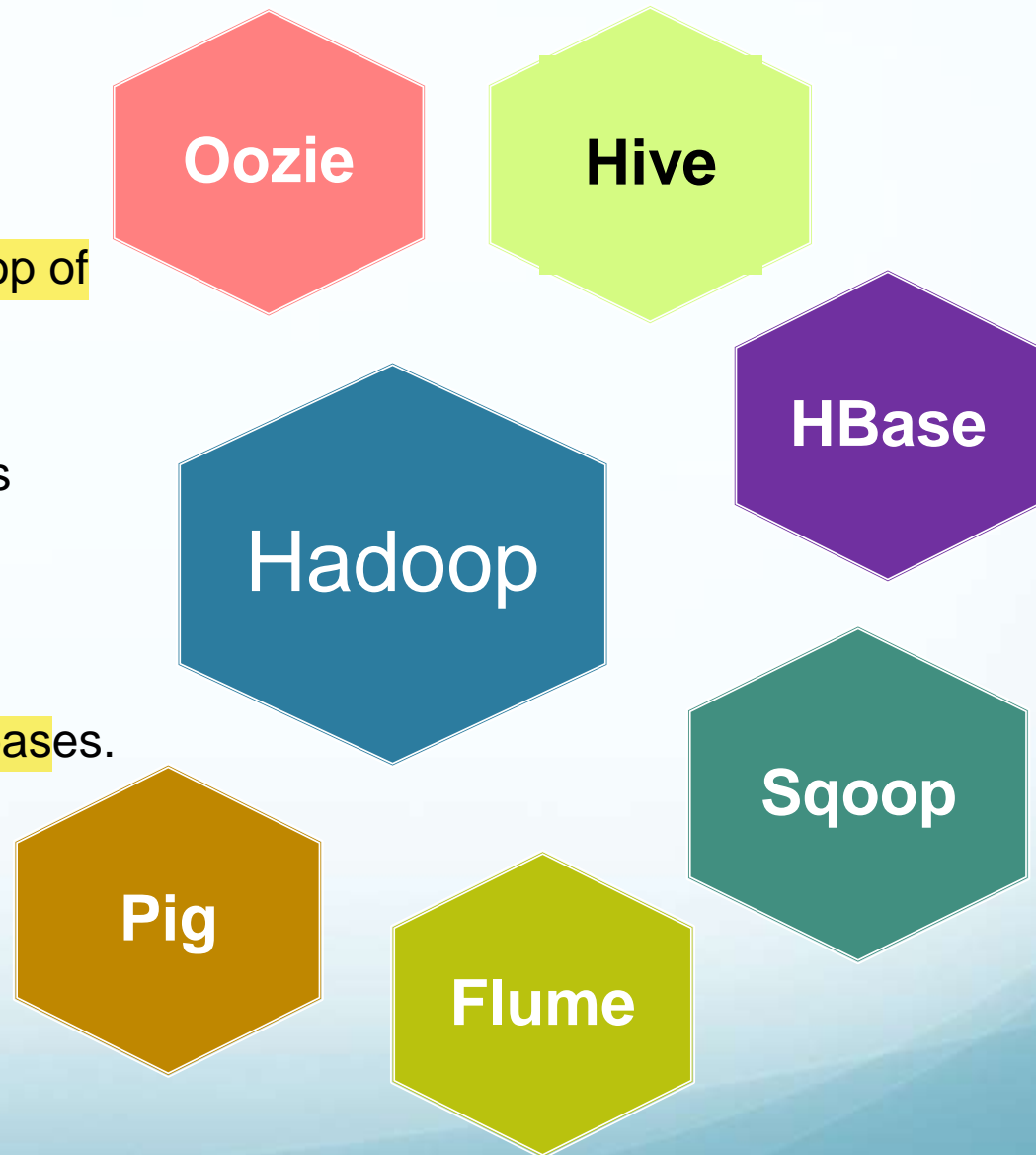
*Hive*: A data warehousing layer on top of Hadoop that allows SQL-like queries (HiveQL) on large datasets.

*HBase*: A NoSQL database that runs on top of HDFS, allowing real-time read/write access to large datasets.

*Sqoop*: A tool for transferring data between Hadoop and relational databases.

*Flume*: A distributed service for collecting and moving large amounts of log data into HDFS.

*Pig* : A platform for analyzing large datasets with a scripting language (Pig Latin).



# Analyzing data with Hadoop

Data analysis with Hadoop involves multiple stages, from data ingestion to final output, each leveraging different components of the Hadoop ecosystem. The key Phases of Data Analysis with Hadoop are listed below:

## 1. Data Ingestion:

Data ingestion is the process of transferring data from multiple sources such as relational databases, log files, social media, IoT devices, and more into Hadoop's HDFS (Hadoop Distributed File System) or other Hadoop-compatible storage systems. This is the first and foundational step for any big data analysis pipeline, as it allows Hadoop to process large volumes of structured, semi-structured, or unstructured data coming from disparate sources. The data ingestion can happen in batch mode i.e moving large amounts of data at once or in real-time/streaming mode i.e continuous flow of data as it's generated. Both modes are supported by different tools and frameworks in the Hadoop ecosystem. Various data sources are

- ✓ Relational databases (e.g., MySQL, Oracle, SQL Server)
- ✓ NoSQL databases (e.g., MongoDB, Cassandra)
- ✓ Flat files (e.g., CSV, JSON, XML)
- ✓ Log files from web servers, applications, or devices
- ✓ APIs or social media feeds
- ✓ IoT devices generating sensor data
- ✓ Data streams (e.g., clickstreams from websites)

# Analyzing data with Hadoop (contd)

## Key Ingestion Tools:

1. *Apache Sqoop*: A tool to import data from traditional databases (RDBMS) like MySQL, PostgreSQL, and Oracle into Hadoop's HDFS, or vice versa. Sqoop allows you to bring large datasets stored in relational databases into the Hadoop ecosystem.
2. *Apache Flume*: A service for collecting, aggregating, and moving large amounts of log and event data from different sources to HDFS. It's ideal for ingesting real-time streaming data from sources such as social media, web logs, or IoT devices.

## 2. Data Storage in HDFS :

Hadoop Distributed File System (HDFS) is the primary storage layer for Hadoop, designed to store large datasets across a distributed cluster. It allows for scalable and fault-tolerant storage of data by splitting large files into smaller blocks typically 128 MB or 256 MB in size and replicating those blocks across multiple nodes. Fault tolerance is achieved as HDFS replicates each data block across multiple nodes (by default, three replicas per block). This replication ensures that data remains available even if a node fails, thus maintaining data reliability. The storage and retrieval of data in HDFS form the backbone of big data processing, as it enables high-throughput access and efficient data analysis through parallel processing. HDFS is optimized for write-once, read-many access patterns, which means once data is written, it is rarely updated. This design improves performance for reading and analyzing large-scale datasets.



# Analyzing data with Hadoop (contd)

If a file is larger than the block size (128 MB), it will be split into several blocks. For example, a 512 MB file will be split into 4 blocks of 128 MB each and saved as follows.

Example: A 512 MB file will be split into 4 blocks:

Block 1 → DataNode A, DataNode B, DataNode C

Block 2 → DataNode B, DataNode C, DataNode D

Block 3 → DataNode C, DataNode D, DataNode A

Block 4 → DataNode D, DataNode A, DataNode B

## Data Storage Formats :

The data in HDFS is stored in efficient formats for big data analysis. Few of the popular data formats are :

1. CSV (Comma-Separated Values) and TSV (Tab-Separated Values)
2. JSON
3. Sequence Files
4. Avro
5. Parquet
6. Optimised Row Columnar (ORC)
7. Protobuf



# Analyzing data with Hadoop (contd)

## 3. Data Processing with MapReduce :

MapReduce is a **core component** of Hadoop designed for **processing large-scale data** across distributed clusters. It is a programming model that **enables parallel and distributed processing of vast datasets** by **dividing tasks into two distinct phases**.

The **concept** behind MapReduce is to **break** down a **large task** into **smaller, independent subtasks**, which are **processed in parallel** and **later aggregated**. This model is highly **scalable** and **fault-tolerant**, making it suitable for processing datasets that span **petabytes** and are distributed across **hundreds or thousands of machines**.

The two phases viz Map and reduce works as follows :

### *Map Phase:*

In the **Map** phase, the **input dataset** is **split into smaller chunks**, and **each chunk** is **processed in parallel** by a **mapper** function. The **mapper reads** the data, **processes** it, and **outputs key-value pairs**. A simple **example** of the Mapper function is can be , say If we want to **count the number of times each word appears** in a large collection of text files (e.g., **web logs** or social media posts):

*We takes a **block of text**, **splits it into words**, and **outputs each word as a key**, with a **value of 1***

*(e.g., "apple" → (apple, 1)).*

### *Reduce Phase:*

The **Reduce** function takes the **sorted intermediate data** and **processes it to generate the final output**. **apple, 1), (apple, 1), (apple, 1) → (apple, 3)).**

The reducer aggregates, combines, or summarizes the results based on the keys. The

# Analyzing data with Hadoop (contd)

## 4. Higher-Level Data Processing :

MapReduce model can be complex for non-programmers. To make Hadoop more accessible, higher-level abstractions such as Apache Hive and Apache Pig were introduced.

**Apache Hive** is a data warehousing solution built on top of Hadoop. It allows analysts to run SQL-like queries (HiveQL) over large datasets stored in HDFS without needing to write complex Java code. Hive translates HiveQL queries into MapReduce under the hood. It supports various file formats (ORC, Parquet) and can partition large datasets for faster querying.

**Apache Pig** uses a scripting language called **Pig Latin** that is more flexible than SQL and allows users to perform complex transformations and data flows on large datasets. Pig Latin scripts are compiled into MapReduce jobs and executed on Hadoop clusters.

## 5. Workflow Scheduling :

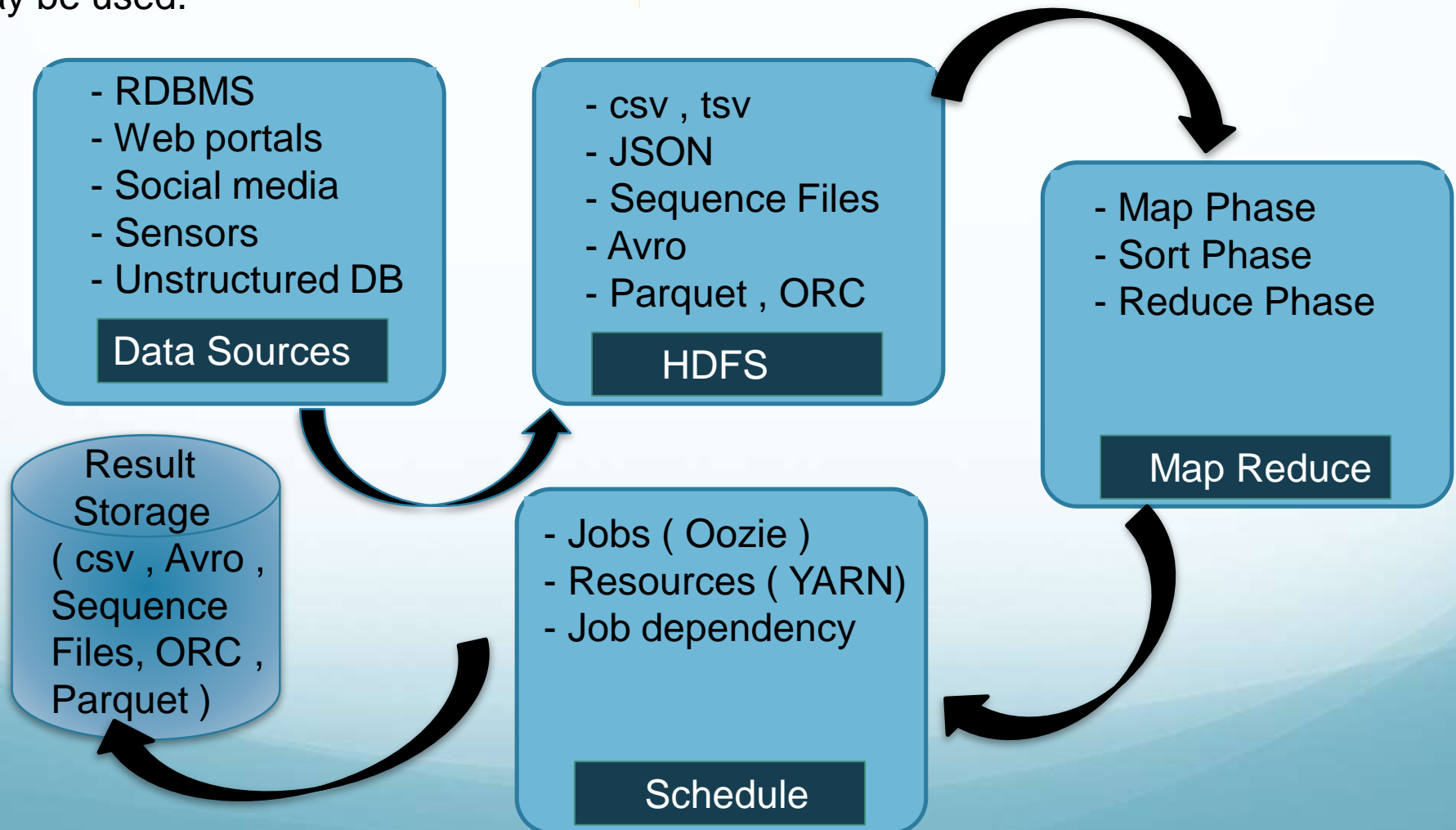
Once we have imported the data from various data sources, saved it on HDFS and ready to perform MapReduce operation using tools the next step is to automate and schedule these jobs using tools like Oozie. We can use this tool to automates and manages the execution of multiple Hadoop jobs. It allows users to define dependencies between jobs and schedule them to run in a particular sequence.

## 6. Result Storage :

The Result Storage step is the final stage in a Hadoop workflow, particularly after a MapReduce or other distributed processing job has been completed. This step

# Analyzing data with Hadoop (contd)

involves saving the output produced by the reducer tasks to HDFS (Hadoop Distributed File System) or another storage system such as HBase, S3, etc. The output files from different reducers may need to be concatenated or combined to create a single file, depending on the use case. To store appropriate storage format like csv , tab , Avro etc may be used.



# Design of Hadoop distributed file system (HDFS)

- HDFS (Hadoop Distributed File System) is a core component of the Hadoop ecosystem, designed to store massive amounts of data reliably and efficiently across a distributed cluster. It follows a set of principles and design ideas tailored to handle big data workloads, ensuring scalability, fault tolerance, and high throughput.
- The architecture of HDFS is optimized for write-once, read-many patterns, making it suitable for storing large files and supporting distributed data processing frameworks like MapReduce, Hive, and Spark.
- Following are the Key Design Principles that are followed in HADOOP.

## 1. Master-Slave Architecture

HDFS follows a **master-slave architecture**, where the master node manages the metadata and overall operation of the system, while the slave nodes store the actual data. The master node does not store actual data but rather tracks where the blocks of each file are stored across the DataNodes. It only stores data such as file locations, permissions, block mapping etc .

The data nodes are the worker nodes that store the actual data blocks. Each DataNode is responsible for managing the storage attached to it and periodically reporting the status of its blocks to the Master node. They perform read and write requests from clients and replicate data blocks as instructed by the Master node. This architecture allows Master node to focus on meta data while data nodes on storing and scaling up for data storage and performance.

# Design of Hadoop distributed file system (contd)

## 2. Rack Aware

HDFS is rack-aware, meaning that it is designed to consider the physical topology of the network when replicating data. It ensures that replicas of data blocks are stored on different machines to improve fault tolerance. In a large data center, machines are often grouped into racks, with high-bandwidth connections within a rack but slower cross-rack communication. By storing replicas across different racks, HDFS ensures that if an entire rack fails (e.g., due to a network switch failure), the data is still available from nodes in other racks.

## 3. Write Once, Read Many

HDFS allows files to be written once but read multiple times. This model reduces the complexity of managing concurrent write operations in a distributed system. This means that once a file is written to HDFS, it cannot be modified. It can only be read or appended to. This simplifies the design of HDFS and improves data consistency and integrity eliminating the need for locking and concurrency control mechanisms.

## 4. Portability

HDFS is designed to be hardware-agnostic and portable across various hardware configurations. It can run on commodity hardware, allowing organizations to build large-scale storage systems without expensive infrastructure. This reduces the cost of setting up a Hadoop cluster.

# Design of Hadoop distributed file system (contd)

## 5. Data Integrity and Consistency

HDFS ensures that data stored in the system is consistent and remains intact even in the case of failures. It uses **checksums** to validate the integrity of the data blocks stored on DataNodes. Each block is checked for corruption, and if a corrupt block is found, it is replaced with a replica from another node. HDFS supports atomic file writes. Once a file is written to HDFS, it cannot be modified. This write-once design ensures data consistency and simplifies file management in a distributed system.

## 6. Scalability & Load Balancing

HDFS is designed to scale out easily, meaning it can handle more data and more processing by simply adding more nodes to the cluster. This makes HDFS suitable for storing and processing petabyte-scale datasets.













# Thanks

# Data Storage Formats

There are several commonly used data storage formats in HDFS (Hadoop Distributed File System). These formats cater to different data requirements, such as binary storage, schema support, compression, and ease of use

1. CSV (Comma-Separated Values) and TSV (Tab-Separated Values)
2. JSON
3. Sequence Files
4. Avro
5. Parquet
6. Optimised Row Columnar (ORC)
7. Protobuf

## 1. CSV (Comma-Separated Values) and TSV (Tab-Separated Values)

CSV and TSV are plain-text formats that store tabular data, where fields are separated by commas or tabs. They are simple and widely supported but are not optimized for storage or processing in big data environments due to their text-based nature. Also they do not natively support compression or can be split. They do not include any schema information, which makes it harder to validate data consistency. Hence this format are useful for storing simple, flat table data, such as customer lists, sales records, or transaction logs.

# Data Storage Formats (contd)

## 2. JSON (JavaScript Object Notation)

JSON is a text-based, human-readable format for representing structured data. It is commonly used in big data environments due to its flexibility and ease of use. It is easy to read and debug, which makes it popular for data exchange between systems (e.g., REST APIs). JSON is a schema-less format, allowing it to store flexible, semi-structured data. This is useful for cases where the structure of the data may vary between records. It is widely used across all programming languages and platforms, making it a popular format for data interchange. However like CSV format JSON also does not support native compression nor naturally splittable which becomes a limitation for distributed processing in Hadoop. When large JSON files are stored in HDFS, it's challenging to split them into smaller pieces for parallel processing.

## 3. Sequence Files

A Sequence File is a binary format that stores data as key-value pairs. It is an older Hadoop-native format primarily used for intermediate data in MapReduce jobs and can store any arbitrary Java objects. Sequence files store data as key-value pairs, where the keys and values are serialized in binary format. This makes them ideal for scenarios where data needs to be processed as key-value pairs, such as in MapReduce jobs. They can be compressed using block-level or record-level compression and also are splittable, meaning that large files can be split into multiple smaller pieces, enabling parallel processing across multiple nodes.

# Data Storage Formats (contd)

## 4. Avro

Apache Avro is a row-based binary data serialization format that is optimized for performance and supports schema evolution. It's widely used in big data ecosystems because of its compactness, versatility, and ability to handle both structured and semi-structured data. Avro stores data row by row, making it well-suited for write-heavy workloads. It can carry their schema along with the data, which allows for easy serialization and deserialization across systems. Since it uses compact binary encoding, it reduces storage overhead. Also, Avro files are splittable, which makes them suitable for distributed processing frameworks like Hadoop MapReduce or Apache Spark. Avro is often used to store logs or sensor data that might evolve over time, as it efficiently handles schema changes.

## 5. Parquet

Apache Parquet is a columnar storage format, optimized for efficient reading and writing of large datasets. It was developed to provide high performance for both read-heavy and write-heavy workloads in data analytics systems. Data in Parquet is stored in a column-oriented format, meaning that the values from the same column are stored together. This enables efficient querying and retrieval of specific columns without scanning entire rows. Parquet uses efficient compression algorithms, such as **Snappy**, **GZIP**, or **LZO**, which reduce storage space while maintaining high performance. Parquet files are splittable, meaning large files can be divided into smaller pieces for parallel processing across different nodes in a Hadoop cluster.

# Data Storage Formats (contd)

Parquet enforces a schema for the data, ensuring that the data structure is well-defined and validated. It supports rich data types like nested structures (arrays, maps), making it a great fit for complex data models.

## 6. ORC (Optimized Row Columnar)

ORC is another columnar storage format, designed specifically for the Hadoop ecosystem, particularly for use with Apache Hive. It was introduced to improve on earlier storage formats like Sequence File by providing better compression, faster query performance, and more efficient data storage. Similar to Parquet, ORC stores data in a column-oriented format. This allows for efficient reading and writing of specific columns during query execution. ORC provides lightweight and advanced compression algorithms like **Zlib** and **Snappy** and can compress both individual columns and the entire file at different levels thus reducing the size of data stored on HDFS. ORC files are splittable also, making them suitable for parallel processing across nodes in Hadoop clusters. One speciality of ORC files are that they supports **predicate pushdown**, which allows filtering of rows during the data scan phase thereby improving query performance by reducing the amount of data processed. ORC automatically creates lightweight indexes for each column, which allows queries to be executed more efficiently by skipping irrelevant data during scans. All these features makes ORC highly optimized for large-scale, read-heavy analytics workloads.



# Data Storage Formats (contd)

The background of the slide features a series of overlapping, wavy, horizontal bands in shades of light blue and white, creating a soft, abstract, and modern aesthetic.

# Data Storage Formats (contd)

The background of the slide features a series of overlapping, wavy, horizontal bands in shades of light blue and white, creating a soft, abstract, and modern aesthetic.

# Data Storage Formats (contd)

In the Hadoop ecosystem, choosing the right data format for storage depends on the specific needs of your data processing tasks. Parquet and ORC are the top choices for columnar storage in analytical workloads, with Parquet being more general-purpose and optimized for Spark, while ORC offers deep integration with Hive and excels in structured, tabular data scenarios. For row-based storage, Avro is a great option, especially when schema evolution is needed. Sequence Files work well for intermediate MapReduce data, while JSON and CSV are popular for human-readable formats but lack the compression and efficiency of more advanced formats like Parquet or ORC.

# Data Storage Formats (contd)

The background of the slide features a series of overlapping, wavy, horizontal bands in shades of light blue and white, creating a soft, abstract, and modern aesthetic.

# Data Storage Formats (contd)



# Data Storage Formats (contd)

