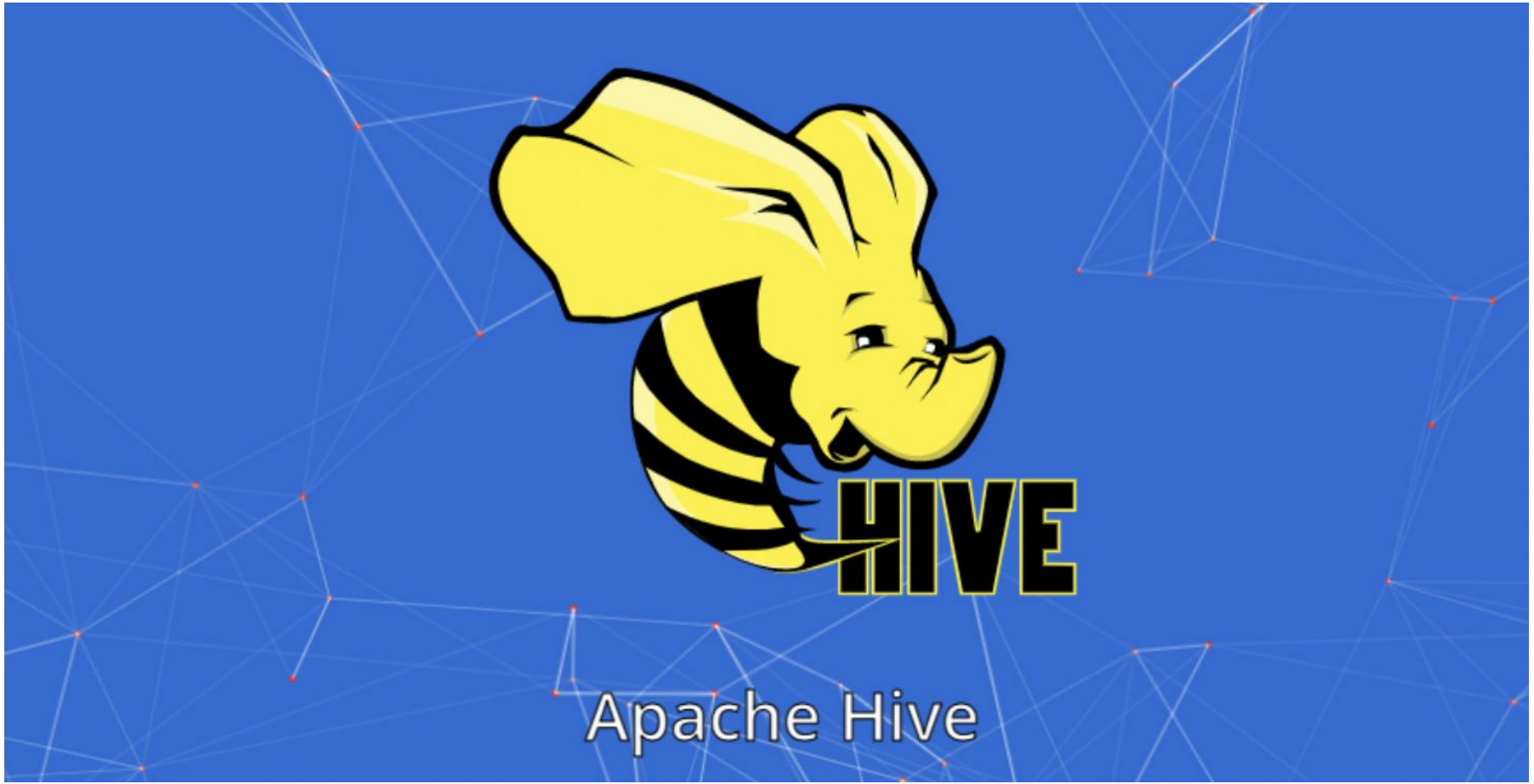

HIVE
Apache Hive

Plain Text Data Storage vs Byte Data Storage

GROW DATA SKILLS

Storing data as plain text and storing data as bytes are two different ways of representing data, each with their advantages and disadvantages.

Plain Text Data Storage:

- In plain text data storage, data is stored as readable characters.
- For example, consider the number 12345. In plain text, it's stored as the characters '1', '2', '3', '4', and '5'.
- Each character typically uses 1 byte of memory (in ASCII), or 2 bytes (in UTF-16), so this number would use 5 to 10 bytes of memory.
- The advantage of plain text storage is that it's human-readable and easy to interpret without any conversion.
- The disadvantage is that it's not space-efficient. Larger numbers or data types other than integers (like floating-point numbers) will use more space.

Byte (Binary) Data Storage:

- In byte (or binary) data storage, data is stored as binary values, not as readable characters. Each byte consists of 8 bits, and each bit can be either 0 or 1.
- Using our previous example, the number 12345 can be represented in binary format as 11000000111001, which is 14 bits or 2 bytes (with 6 unused bits). In a more memory-optimized format, it could use only the necessary 14 bits.
- The advantage of binary storage is that it's very space-efficient. Each type of data (integer, float, etc.) has a standard size, regardless of its value.
- The disadvantage is that binary data is not human-readable. You need to know the type of data and how it's encoded to convert it back to a readable format.

SerDe in Hive

- SerDe stands for Serializer/Deserializer.
- It's a crucial component of Hive used for IO operations, specifically for reading and writing data.
- It helps Hive to read data in custom formats and translate it into a format Hive can process (deserialization) and vice versa (serialization).

Role of SerDe in Hive:

- When reading data from a table (input to Hive), deserialization is performed by the SerDe.
- When writing data to a table (output from Hive), serialization is performed by the SerDe.

Serialization - Process of converting an object in memory into bytes that can be stored in a file or transmitted over a network.

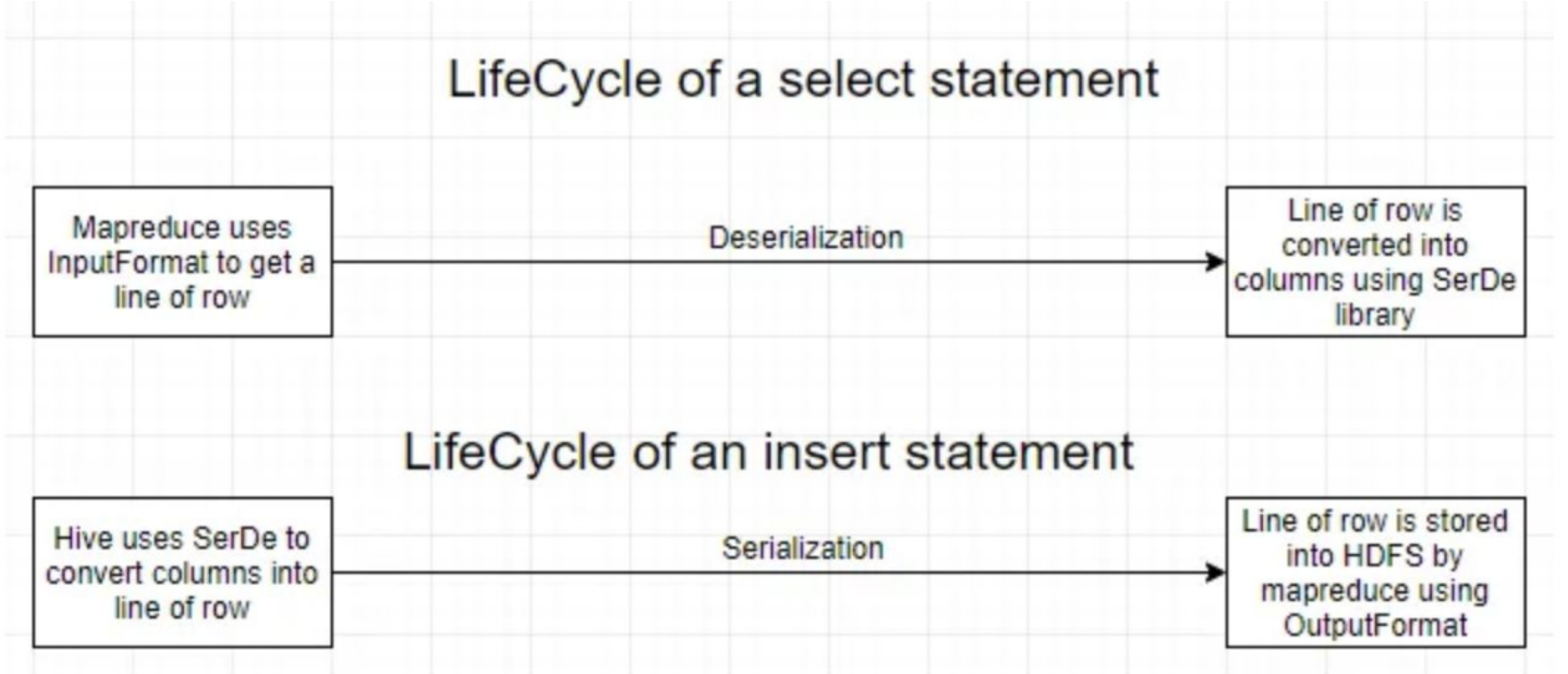
Deserialization - Process of converting the bytes back into an object in memory.

“A select statement creates deserialized data(columns) that is understood by Hive. An insert statement creates serialized data(files) that can be stored into an external storage like HDFS”.

Hive Row format and Map-reduce Input/Output format

```
CREATE TABLE my_table(a string, b string, ...)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
   "separatorChar" = "\t",
   "quoteChar"     = "'",
   "escapeChar"    = "\\"
)
STORED AS TEXTFILE;
```

In any table definition, there are two important sections. The **“Row Format”** describes the **libraries** used to convert a given row into columns. The **“Stored as”** describes the **InputFormat** and **OutputFormat** libraries used by map-reduce to read and write to HDFS files.



File formats in Hive with SerDe Library

GROW DATA SKILLS

TextFile:

- This is the default file format.
- Each line in the text file is a record. Hive uses the *LazySimpleSerDe* for serialization and deserialization.
- It's easy to use but doesn't provide good compression or the ability to skip over not-needed columns during read.

SequenceFile:

- It's a flat file format consisting of binary key/value pairs.
- It provides a SequenceFileInputFormat for reading files and SequenceFileOutputFormat for writing files.
- Mainly used for data between MapReduce jobs in Hadoop.
- Default SerDe is ***org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe***

JSON:

- Hive supports reading and writing JSON format data.
- Note that JSON files typically do not have a splittable structure, which can affect performance as only one mapper can read the data when not splittable.
- Default SerDe is ***org.apache.hive.hcatalog.data.JsonSerDe***

CSV (Comma Separated Values):

- CSV is a simple, human-readable file format used to store tabular data. Columns are separated by commas, and rows by new lines.
- CSV files can be read and written in Hive using the ***org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe*** or ***org.apache.hadoop.hive.serde2.OpenCSVSerde*** for advanced CSV parsing.
- CSV lacks a splittable structure, which may affect performance due to limited parallel processing.

RCFile (Record Columnar File):

- Developed by Facebook, RCFile provides an efficient way to store data in a columnar format.
- It's a hybrid format that keeps a sync marker to delineate rows into groups (row splits), and within each split, it stores data column-wise.
- Default SerDe is ***org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe***

ORCFile (Optimized Row Columnar File):

- Introduced by Hortonworks, ORC is a highly efficient way to store Hive data.
- It provides efficient compression and encoding schemes with enhanced performance to handle complex data types.
- Default SerDe is **org.apache.hadoop.hive.ql.io.orc.OrcSerde**

Parquet:

- Columnar storage format, available to any project in the Hadoop ecosystem.
- It's designed to bring efficient columnar storage of data compared to row-based like CSV or TSV files.
- Default SerDe is ***org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe***

Avro:

- It's a row-oriented format that is highly splittable.
- It also supports schema evolution - you can have Avro data files where each file has a different schema but all are part of the same table.
- Default SerDe is ***org.apache.hadoop.hive.serde2.avro.AvroSerDe***

ORC vs Parquet vs AVRO

GROW DATA SKILLS

ORC (Optimized Row Columnar):

- Built for the Hive query engine, ORC is a columnar storage format that allows Hive to read, write, and process data faster.
- It allows for efficient compression, which saves storage space, and adds improvements in the speed of data retrieval, making it suitable for performing high-speed queries.
- It stores collections of rows, not individual rows.
- Each file consists of row index, column statistics, and stripes (a row of data consisting of several rows) that contain the column data.
- Supports complex types: Structs, Lists, Maps, and Unions.
- Also supports advanced features like **bloom filters** and indexing. A bloom filter is a data structure that can identify whether an element might be present in a set, or is definitely not present. In the context of ORC, bloom filters can help skip unnecessary reads when performing a lookup on a particular column value.

ORC vs Parquet vs AVRO

GROW DATA SKILLS

Parquet:

- It is columnar in nature and designed to bring efficient columnar storage of data.
- Provides efficient data compression and encoding schemes with enhanced performance to handle complex data in comparison to row-based files like CSV.
- Schema evolution is handled in the file metadata allowing compatible schema evolution.
- It supports all data types, including nested ones, and integrates well with flat data, semi-structured data, and nested data sources.

Avro:

- It's a row-based format that's ideal for write-heavy workloads.
- It is also schema-based, and its schemas are defined with JSON, facilitating implementation in languages with JSON libraries.
- Avro serializes the data which has a built-in schema. This serialized data can be deserialized by any application.
- Avro is highly integrated with both Hadoop and Schema registry and supports a lot of data types.

ORC vs Parquet vs AVRO

GROW DATA SKILLS

	Avro	Parquet	ORC
Schema Evolution Support			
Compression			
Splitability			
Most Compatible Platforms	Kafka, Druid	Impala, Arrow Drill, Spark	Hive, Presto
Row or Column	Row	Column	Column
Read or Write	Write	Read	Read

How to decide which file format to choose?

GROW DATA SKILLS

- **Columnar vs Row-based:** Columnar storage like Parquet and ORC is efficient for read-heavy workloads and is especially effective for queries that only access a small subset of total columns, as it allows skipping over non-relevant data quickly. Row-based storage like Avro is typically better for write-heavy workloads and for queries that access many or all columns of a table, as all of the data in a row is located next to each other.

- **Schema Evolution:** If your data schema may change over time, Avro is a solid choice because of its support for schema evolution. Avro stores the schema and the data together, allowing you to add or remove fields over time. Parquet and ORC also support schema evolution, but with some limitations compared to Avro.

- **Compression:** Parquet and ORC, being columnar file formats, allow for better compression and improved query performance as data of the same type is stored together. Avro also supports compression but being a row-based format, it might not be as efficient as Parquet or ORC.

- **Splittability:** When compressed, splittable file formats can still be divided into smaller parts and processed in parallel. Parquet, ORC, and Avro are all splittable, even when compressed.

- **Complex Types and Nested Data:** If your data includes complex nested structures, then Parquet is a good choice because it provides efficient encoding and compression of nested data.