Query large datasets in near real time with Hive and Pig

Hive

- **Challenges at Facebook: Exponential Growth of Data**

➢ Before 2008, all the data processing infrastructure in Facebook was built around a data warehouse based on commercial RDBMS. These infrastructures were capable enough to suffice the needs of Facebook at that time. But, as the data started growing very fast, it became a huge challenge to manage and process this huge dataset. According to a Facebook article, the data scaled from a 15 TB data set in 2007 to a 2 PB data in 2009. Also, many Facebook products involve analysis of the data like Audience Insights, Facebook Lexicon, Facebook Ads, etc. So, they needed a scalable and economical solution to cope up with this very problem and, therefore started using the Hadoop framework.

Hive

- **Birth of Hive**

➢ Facebook played an active role in the birth of Hive as Facebook uses Hadoop to handle Big Data. Hadoop uses MapReduce to process data. Previously, users needed to write lengthy, complex codes to process and analyze data. Not everyone was well-versed in **Java and other complex programming languages**. Also, for performing simple analysis one has to write a hundred lines of MapReduce code. On the other hand, many individuals were **comfortable with** writing queries in **SQL**. For this reason, there was a need to develop a language similar to SQL, which was well-known to all users. This is how the Hive Query Language, also known as HiveQL, came to be.

Hive

- **What is Hive in Hadoop?**

➢ Hive is a data warehouse system used to **query and analyze large datasets stored in HDFS**. Hive uses a query language called HiveQL, which is similar to SQL.

➢ The image above demonstrates a user writing queries in the HiveQL language, which is then converted into MapReduce tasks. Next, the data is processed and analyzed. **HiveQL works on structured and semi-structured data**, such as numbers, addresses, dates, names, and so on. HiveQL allows **multiple users to query data simultaneously**.

Hive

- **What is Hive in Hadoop?**

- Hive is query engine because hive does not has it's own storage to store the data.
- Require knowledge of simple SQL query (Create, update, delete, insert, select, sub queries and join). No need of complex query like stored procedure, trigger these kind of sequel knowledge is not required.
- Hive act as a vehicle and runs on a engine map-reduce. Therefore we say hive is abstraction of map-reduce.
- Hive internally used map reduce engine to process the query. So instead of Java; communicate with sequels. (Bypassing this with sequel via hive)
- Hive is just replacing Java part; Hive is not replacing map reduce part.

Hive

- **What is Hive in Hadoop?**

- Hive **reads** data from **HDFS** and then **process** it in **hive** with the help of **map reduce** and then the **output** again stored back to **HDFS.**
- When someone say we are not using map reduce then what I understood is that You are using some other engine to run hive.
- Similar to map reduce We have one more engine which is **Spark**. Spark is replacement of map reduce. But it’s not replacement of whole hadoop framework.
- Hive can also run on Spark engine instead of map reduce. So hive is a vehicle and map reduce/spark engine which operates vehicle.
- Question: We can do everything with sequel itself; no need to write code for every situation?

Hive

- No, We can’t do everything with sequel itself. There will be some complex transformation (like require extra optimization, extra performance, extra complex logic etc.) then we have to go for spark or map reduce and to write programs. But Hive is more matured in recent days and we can do so many things with hive.
- Amazon use Hive with S3 (Database). So Hive can also be use with S3 also instead of HDFS

Hive

- **Why Hive?**
- Apache Hive saves developers from **writing complex** Hadoop MapReduce jobs for ad-hoc requirements. Hence, hive provides summarization, analysis, and query of data.
- Hive is very **fast and scalable**. It is highly extensible. Since Apache Hive is similar to SQL, hence it becomes very easy for the SQL developers to learn and implement Hive Queries.
- Hive reduces the complexity of MapReduce by providing an interface where the user can submit SQL queries. So, now business analysts can play with **Big Data** using Apache Hive and generate insights.
- It also provides file access on various data stores like HDFS and HBase. The most important feature of Apache Hive is that to learn Hive we don’t have to learn Java.

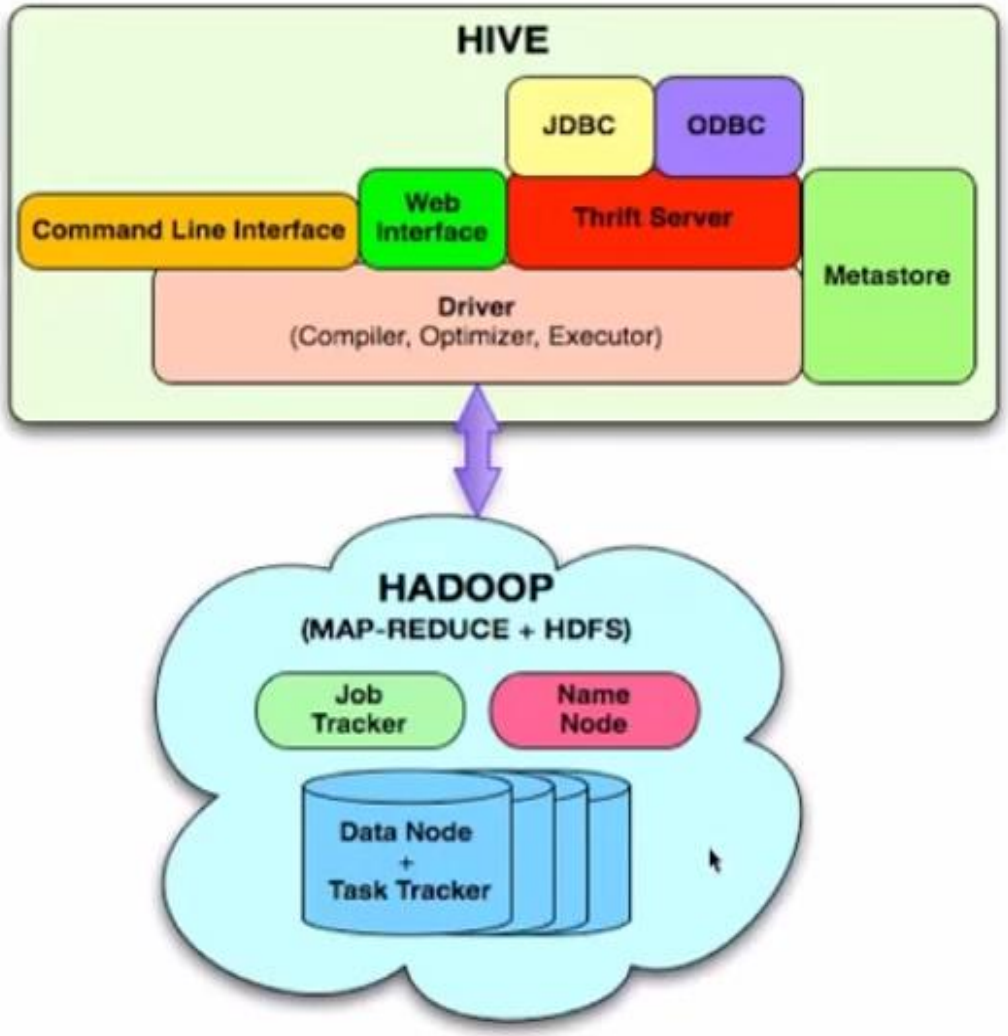
Architecture

Externel Interfaces- CLI, WebUI, JDBC, ODBC programming interfaces

Thrift Server – Cross Language service framework .

Metastore - Meta data about the Hive tables, partitions

Driver - Brain of Hive! Compiler, Optimizer and Execution engine



Hive components

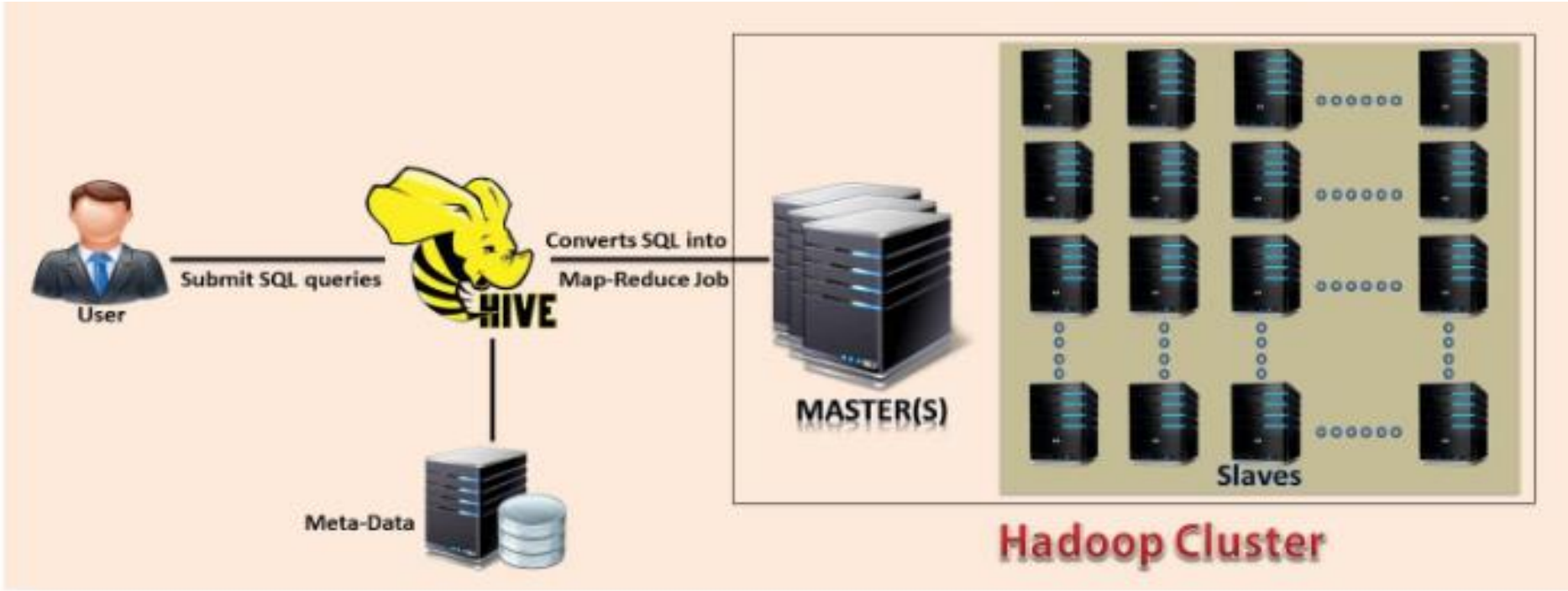
- **Metastore –** It stores metadata for each of the tables like their schema and location. Hive also includes the partition metadata. This helps the driver to track the progress of various data sets distributed over the cluster. It stores the data in a traditional RDBMS format. Hive **metadata helps the driver to keep a track of the data** and it is highly crucial. Backup server regularly replicates the data which it can retrieve in case of data loss.
- **Driver –** It acts like a controller which **receives the HiveQL statements**. The driver starts the execution of the statement by creating sessions. It monitors the life cycle and progress of the execution. Driver stores the necessary metadata generated during the execution of a HiveQL statement. It also acts as a collection point of data or query result obtained after the Reduce operation.

Hive components

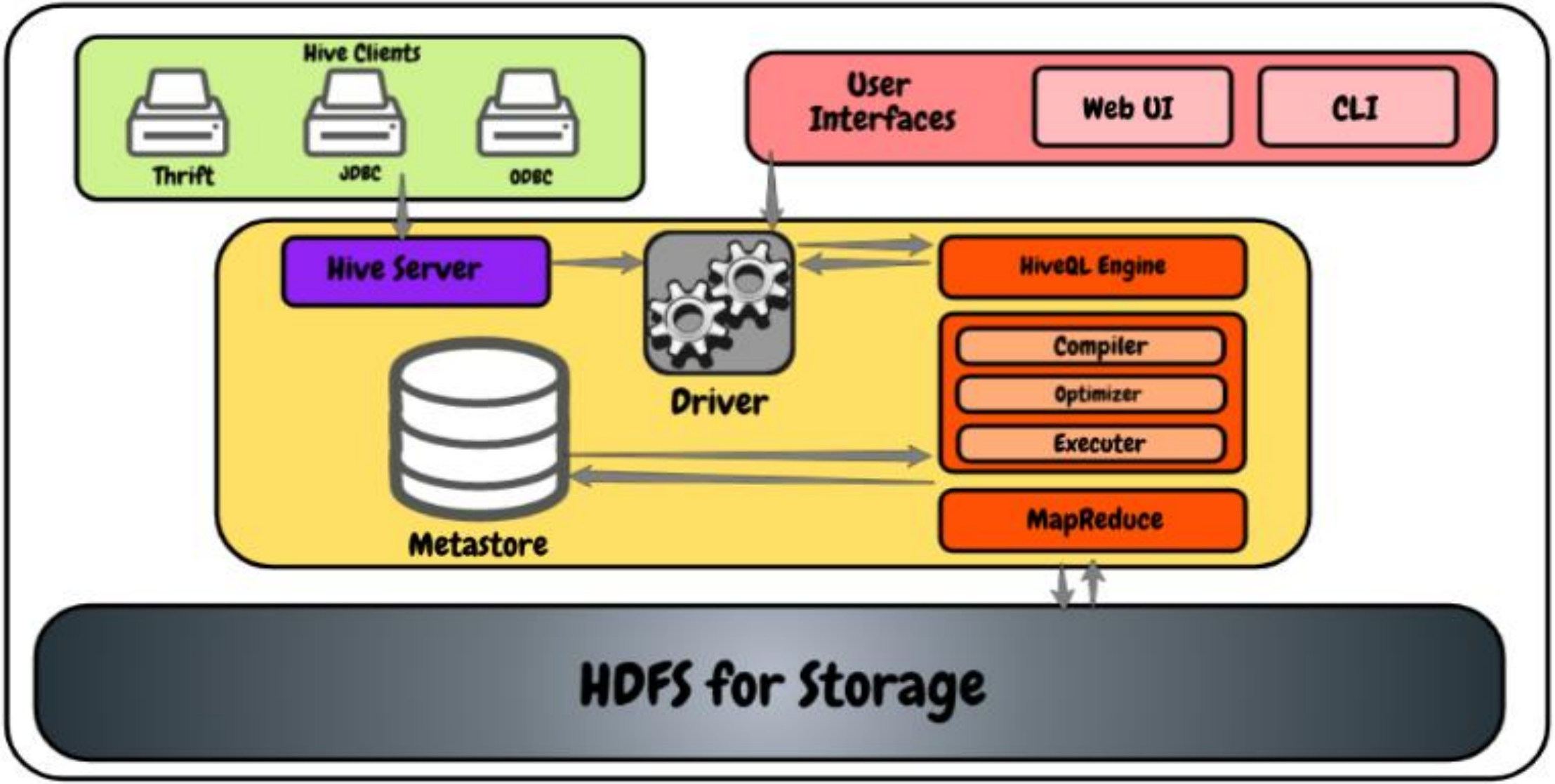
- **Compiler** – It performs the compilation of the HiveQL query. This converts the query to an execution plan. The plan contains the tasks. It also contains steps needed to be performed by the MapReduce to get the output as translated by the query. The compiler in Hive converts the query to an **Abstract Syntax Tree** (AST). First, check for compatibility and compile-time errors, then converts the AST to a **Directed Acyclic Graph (DAG).**
- **Optimizer** – It performs various transformations on the execution plan to provide optimized DAG. It **aggregates** the transformations together, such as converting a pipeline of joins to a single join, for better performance. The optimizer can also split the tasks, such as applying a transformation on data before a reduce operation, to provide better performance.

Hive components

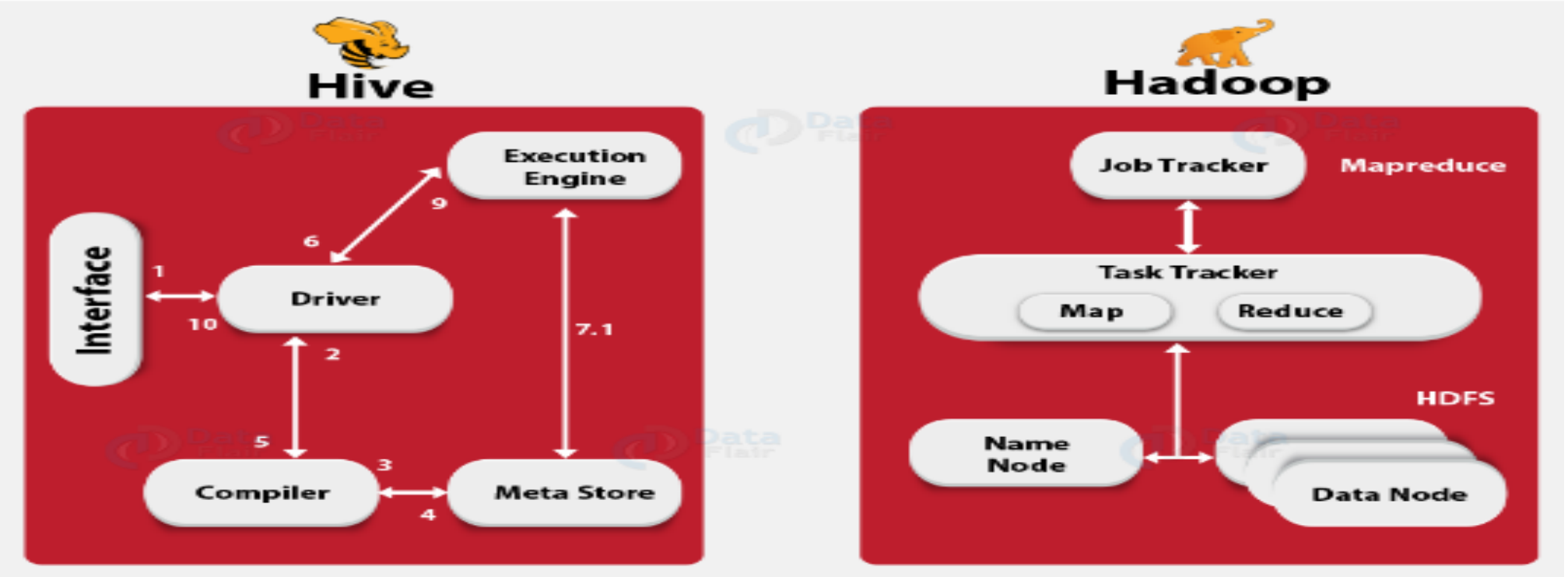
- **Executor** – Once compilation and optimization complete, the executor executes the tasks. Executor takes care of pipelining the tasks.
- **CLI, UI, and Thrift Server** – CLI (command-line interface) provides a user interface for an external user to interact with Hive. Thrift server in Hive allows external clients to interact with Hive over a network, similar to the JDBC or ODBC protocols.



Hive Architecture



How Does Hive Works?



- **Step-1 Execute Query:** In the first step, we write down the query using the web interface or the command-line interface of the hive. It sends it to the driver to execute the query.

How Does Hive Works?

- **Step-2 Get Plan**: In the next step, the driver sends the received query to the compiler where the compiler verifies the syntax.
- **Step-3 Get Metadata**: And once the syntax verification is done, it requests metadata from the meta store.
- **Step-4 Send Metadata**: Now, the metadata provides information like the database, tables, data types of the column in response to the query back to the compiler.
- **Step-5 Send Plan**: The compiler again checks all the requirements received from the meta store and sends the execution plan to the driver.
- **Step-6 Execute Plan**: Now, the driver sends the execution plan to the HiveQL process engine where the engine converts the query into the map-reduce job.

How Does Hive Works?

- **Step-7 Execute Job**: After the query is converted into the map-reduce job, it sends the task information to the Hadoop where the processing of the query begins and at the same time it updates the metadata about the map-reduce job in the meta store.
- **Step-8 Fetch Result**: Once the processing is done, the execution engine receives the results of the query.
- **Step-9 Send Results**: The execution engine transfers the results back to the driver and at last, the driver sends the results to Hive user interfaces from where we can see results.

Hive Shell

- The shell is the primary way with the help of which we interact with the Hive; we can issue our commands or queries in HiveQL inside the Hive shell. Hive Shell is almost similar to MySQL Shell.
- It is the **command line interface** for Hive. In Hive Shell users can run HQL queries. HiveQL is also **case-insensitive** (except for string comparisons) same as SQL.
- We can run the Hive Shell in two modes which are: **Non-Interactive mode and Interactive mode**
- **Hive in Non-Interactive mode –** Hive Shell can be run in the non-interactive mode, with **-f option** we can specify the location of a file which contains HQL queries. For example- hive -f my-script.q

Hive Shell

- **Hive in Interactive mode –** Hive Shell can also be run in the interactive mode. In this mode, we directly need to go to the hive shell and run the queries there. In hive shell, we can submit required queries manually and get the result. For example- $bin/hive, go to hive shell.

HiveQL

DDL :
- CREATE DATABASE
- CREATE TABLE
- ALTER TABLE
- SHOW TABLE
- DESCRIBE

DML:
- LOAD TABLE
- INSERT

QUERY:
- SELECT
- GROUP BY
- JOIN
- MULTI TABLE INSERT

Metastore Configuration

- **Metastore Configuration**
- Metastore stores the meta data information using RDBMS and an open source ORM (Object Relational Model) layer called Data Nucleus which converts the object representation into relational schema and vice versa. The reason for choosing RDBMS instead of HDFS is to achieve low latency.
- We can implement metastore in following two configurations:

➢ Embedded Metastore

➢ Remote Metastore

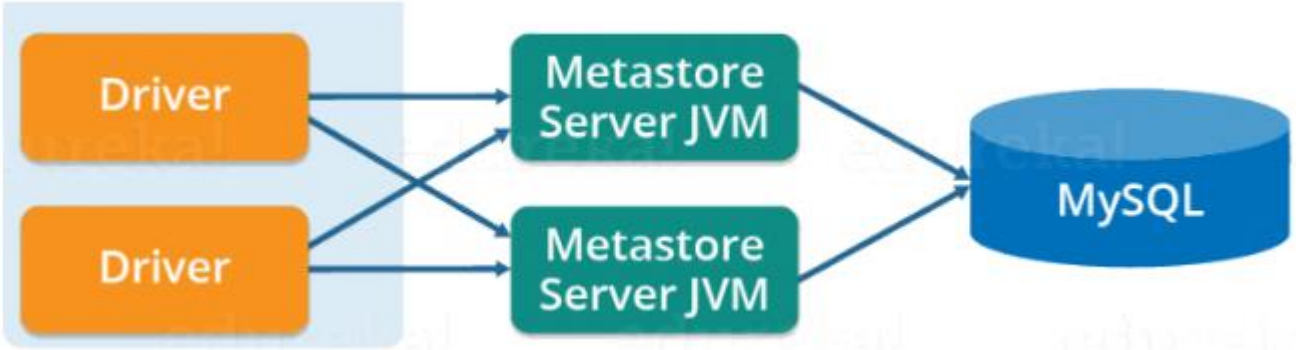
Metastore Configuration

- **Embedded Metastore:**

➢ Both the metastore service and the Hive service runs in the same JVM by default using an embedded Derby Database instance where metadata is stored in the local disk. This is called embedded metastore configuration. In this case, only one user can connect to metastore database at a time. If you start a second instance of Hive driver, you will get an error. This is good for unit testing, but not for the practical solutions.

Metastore Configuration

- **Remote Metastore:**
- In the remote metastore configuration, the metastore service runs on its own separate JVM and not inthe Hive service JVM. Other processes communicate with the metastore server using Thrift Network APIs. You can have one or more metastore servers in this case to provide more availability. The main advantage of using remote metastore is you do not need to share JDBC login credential with each Hive user to access the metastore database.



Hive Data Model

- Data in Hive can be categorized into three types on the granular level:

➢ Table

➢ Partition

➢ Bucket

- **Tables:**
- Tables in Hive are the same as the tables present in a Relational Database. You can perform filter, project, join and union operations on them. There are two types of tables in Hive:

1. Managed Table:

- *Command:*

➢ CREATE TABLE <table_name> (column1 data_type, column2 data_type);

➢ LOAD DATA INPATH <HDFS_file_location> INTO table managed_table;

Hive Data Model

- As the name suggests (managed table), Hive is responsible for managing the data of a managed table. In other words, “Hive manages the data”, is that if you load the data from a file present in HDFS into a Hive Managed Table and issue a DROP command on it, the table along with its metadata will be deleted. So, the data belonging to the dropped managed_table no longer exist anywhere in HDFS and you can’t retrieve it by any means. Basically, you are moving the data when you issue the LOAD command from the HDFS file location to the Hive warehouse directory.
- Note: The default path of the warehouse directory is set to/user/hive/warehouse. The data of a Hive table resides in warehouse_directory/table_name (HDFS). You can also specify the path of the warehouse directory in the hive.metastore.warehouse.dir configuration parameter present in the hive-site.xml.

Hive Data Model

2. External Table:

- CREATE EXTERNAL TABLE <table_name> (column1 data_type, column2 data_type) LOCATION '<table_hive_location>';
- LOAD DATA INPATH '<HDFS_file_location>' INTO TABLE <table_name>;
- For external table, Hive is not responsible for managing the data. In this case, when you issue the LOAD command, Hive moves the data into its warehouse directory. Then, Hive creates the metadata information for the external table. Now, if you issue a DROP command on the external table, only metadata information regarding the external table will be deleted. Therefore, you can still retrive the data of that very external table from the warehouse directory using HDFS commands.

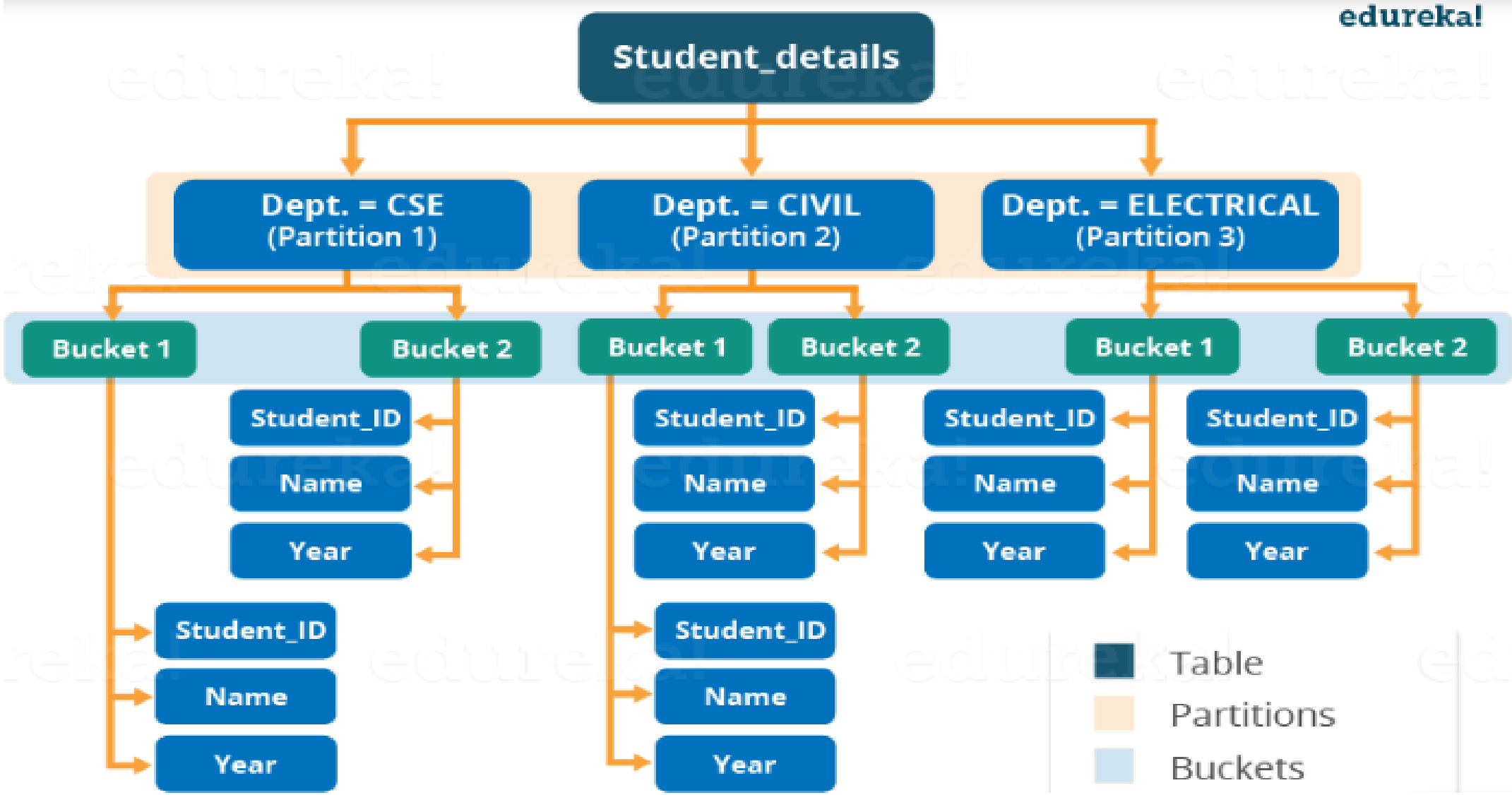
Hive Data Model

- **Partitions:**
- CREATE TABLE table_name (column1 data_type, column2 data_type) PARTITIONED BY (partition1 data_type, partition2 data_type,….);
- Hive organizes tables into partitions for grouping similar type of data together based on a column or partition key. Each Table can have one or more partition keys to identify a particular partition. This allows us to have a faster query on slices of the data.
- **Note:** Remember, the most common mistake made while creating partitions is to specify an existing column name as a partition column. While doing so, you will receive an error – “Error in semantic analysis: Column repeated in partitioning columns”.

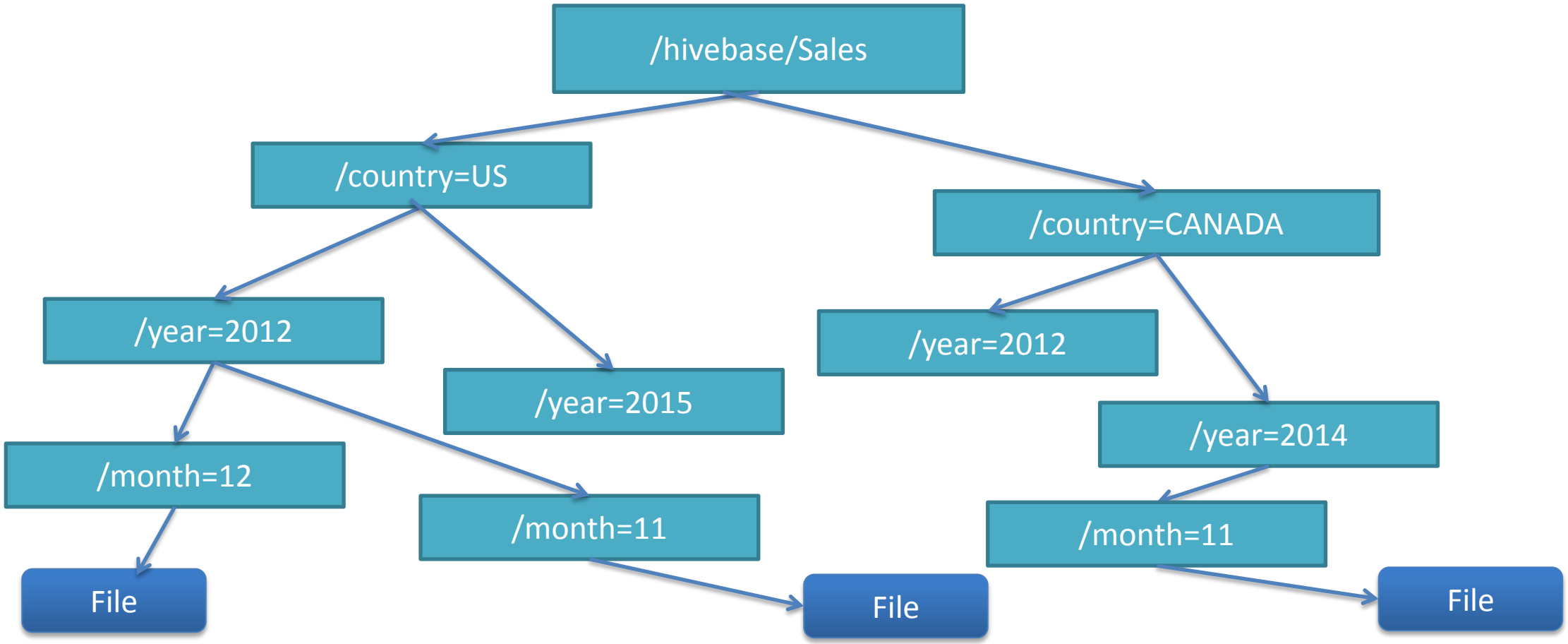
Hive Data Model

- Let us understand partition by taking an example where I have a table student_details containing the student information of some engineering college like student_id, name, department, year, etc. Now, if I perform partitioning based on department column, the information of all the students belonging to a particular department will be stored together in that very partition. Physically, a partition is nothing but a sub-directory in the table directory.
- Let's say we have data for three departments in our student_details table – CSE, ECE and Civil. Therefore, we will have three partitions in total for each of the departments as shown in the image below. And, for each department we will have all the data regarding that very department residing in a separate sub – directory under the Hive table directory. For example, all the student data regarding CSE departments will be stored in user/hive/warehouse/student_details/dept.=CSE. So, the queries regarding CSE students would only have to look through the data present in the CSE partition. This makes partitioning very useful as it reduces the query latency by scanning only **relevant** partitioned data instead of the whole data set. In fact, in real world implementations, you will be dealing with hundreds of TBs of data. So, imagine scanning this huge amount of data for some query where **95%** data scanned by you was un-relevant to your query.

Hive Data Model



Hierarchy of Hive Partitions



Hive Data Model

- **Buckets:**
- CREATE TABLE table_name PARTITIONED BY (partition1 data_type, partition2 data_type,….) CLUSTERED BY (column_name1, column_name2, …) SORTED BY (column_name [ASC|DESC], …)] INTO num_buckets BUCKETS;
- Now, you may divide each partition or the unpartitioned table into Buckets based on the hash function of a column in the table. Actually, each bucket is just a file in the partition directory or the table directory (unpartitioned table). Therefore, if you have chosen to divide the partitions into n buckets, you will have n files in each of your partition directory. For example, you can see the above image where we have bucketed each partition into 2 buckets. So, each partition, say CSE, will have two files where each of them will be storing the CSE student’s data.

Hive Data Model

- **How Hive distributes the rows into buckets?**

➢ Hive determines the bucket number for a row by using the formula: **hash_function (bucketing_column) modulo (num_of_buckets)**. Here, hash_function depends on the column data type. For example, if you are bucketing the table on the basis of some column, let's say user_id, of INT datatype, the hash_function will be – **hash_function (user_id)= integer value of user_id**. And, suppose you have created two buckets, then Hive will determine the rows going to bucket 1 in each partition by calculating: (value of user_id) modulo (2). Therefore, in this case, rows having user_id ending with an even integer digit will reside in a same bucket corresponding to each partition. The hash_function for other data types is a bit complex to calculate and in fact, for a string it is not even humanly recognizable.

Hive Data Model

- **Why do we need buckets?**
- There are two main reasons for performing bucketing to a partition:

➢ A **map side join** requires the data belonging to a unique join key to be present in the same partition. But what about those cases where your partition key differs from join? Therefore, in these cases you can perform a map side join by bucketing the table using the join key.

➢ Bucketing makes the sampling process more efficient and therefore, allows us to decrease the query time.

Data Types in Apache Hive

- Hive data types are divided into the following 5 different categories:

➢ Numeric Type: TINYINT, SMALLINT, INT, BIGINT

➢ Date/Time Types: TIMESTAMP, DATE, INTERVAL

➢ String Types: STRING, VARCHAR, CHAR

➢ Complex Types: STRUCT, MAP, UNION, ARRAY

➢ Misc Types: BOOLEAN, BINARY

Data Type	Description
TINYINT	It is a 1-byte signed integer ranges from **-128 to 127**
SMALLINT	It is a 2-byte signed integer ranges from **-10^{15} to $10^{15} - 1$**
INT	It is a 4-byte signed integer ranges from **-10^{31} to $10^{31} - 1$**
BIGINT	It is a 8-byte signed integer ranges from **-10^{63} to $10^{63} - 1$**
FLOAT	It is a 4-byte single precision floating point number.
DOUBLE	It is a 8-byte double precision floating point number.
TIMESTAMP	It follows the format "YYYY-MM-DD HH:MM:SS.fffffffff" with 9 decimal place precision and also has the option of UNIX timestamp with nanosecond precision.
STRING	String literals can be expressed with either single quotes (') or double quotes (").
VARCHAR	It is a variable length type that ranges 1 and 65535, which specifies the maximum number of characters.
CHAR	It is a fixed-length type whose maximum length is fixed at 255.
MAP	It contains the key-value tuples where the fields are accessed using array notation.
ARRAY	It is a collection of similar types of values that are indexable using zero-based integers.

Features of Apache Hive

- Hive is designed for querying and managing mostly structured data stored in tables
- Hive is scalable, fast, and uses familiar concepts
- Schema gets stored in a database, while processed data goes into a Hadoop Distributed File System (HDFS)
- Tables and databases get created first; then data gets loaded into the proper tables
- Hive uses an SQL-inspired language, sparing the user from dealing with the complexity of MapReduce programming. It makes learning more accessible by utilizing familiar concepts found in relational databases, such as columns, tables, rows, and schema, etc.
- The most significant difference between the Hive Query Language (HQL) and SQL is that Hive executes queries on Hadoop's infrastructure instead of on a traditional database

Features of Apache Hive

- Since Hadoop's programming works on flat files, Hive uses directory structures to "partition" data, improving performance on specific queries
- Hive supports partition and buckets for fast and simple data retrieval
- Hive supports custom user-defined functions (UDF) for tasks like data cleansing and filtering. Hive UDFs can be defined according to programmers' requirements

Advantages/Disadvantages of Apache Hive

- Uses SQL like query language which is already familiar to most of the developers so makes it easy to use.
- It is highly scalable, you can use it to process any size of data.
- Supports multiple databases like MySQL, derby, Postgres, and Oracle for its metastore.
- Supports multiple data formats also allows indexing, partitioning, and bucketing for query optimization.
- Can only deal with cold data and is useless when it comes to processing real-time data.
- It is comparatively slower than some of its competitors. If your use-case is mostly about batch processing then Hive is well and fine.

REFERENCES

- https://hive.apache.org/
- http://www.qubole.com/blog/big-data/hive-best-practices/

Pig

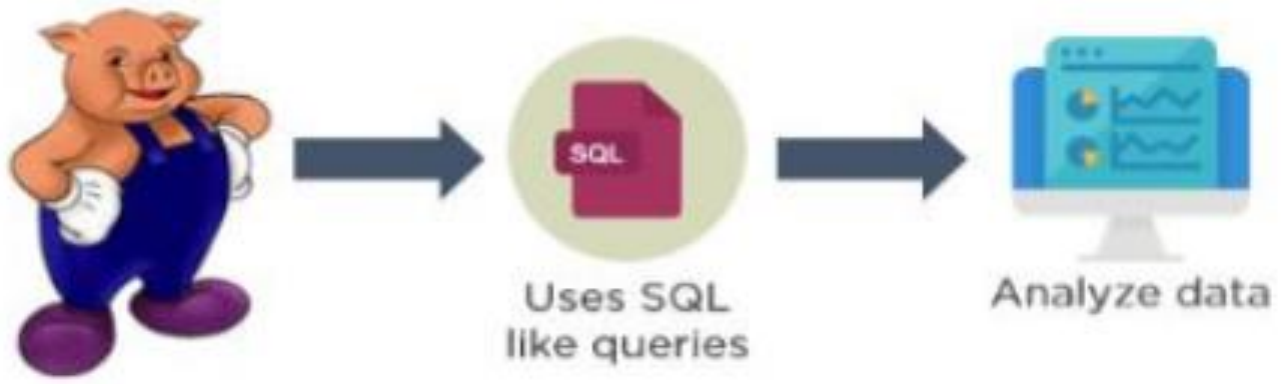
- Pig came into existence to solve issues with MapReduce.
- **Birth of Pig**

➢ Although MapReduce helped process and analyze Big Data faster, it had its flaws. Individuals who were unfamiliar with programming often found it challenging to write lengthy Java codes. Eventually, it became a difficult task to maintain and optimize the code, and as a result, the processing time increased.

➢ This was the reason Yahoo faced problems when it came to processing and analyzing large datasets. Apache Pig was developed to analyze large datasets without using time-consuming and complex Java codes. Pig was explicitly developed for non-programmers.

Pig

- **What is Pig in Hadoop?**
- Pig is a scripting platform that runs on Hadoop clusters, designed to process and analyze large datasets. Pig uses a language called **Pig Latin**, which is similar to SQL. This language does not require as much code in order to analyze data. Although it is similar to SQL, it does have significant differences. In Pig Latin, **10 lines of code is equivalent to 200 lines in Java**. This, in turn, results in shorter development times.



Pig

- What stands out about Pig is that it operates on various types of data, including **structured, semi-structured, and unstructured** data. Whether you're working with structured, semi-structured, or unstructured data, Pig takes care of it all.
- To analyze data using Apache Pig, programmers need to write scripts using Pig Latin language.
- All these scripts are internally converted to Map and Reduce tasks.
- Apache Pig has a component known as Pig Engine that accepts the Pig Latin scripts as input and converts those scripts into MapReduce jobs.

Why do we need Apache Pig?

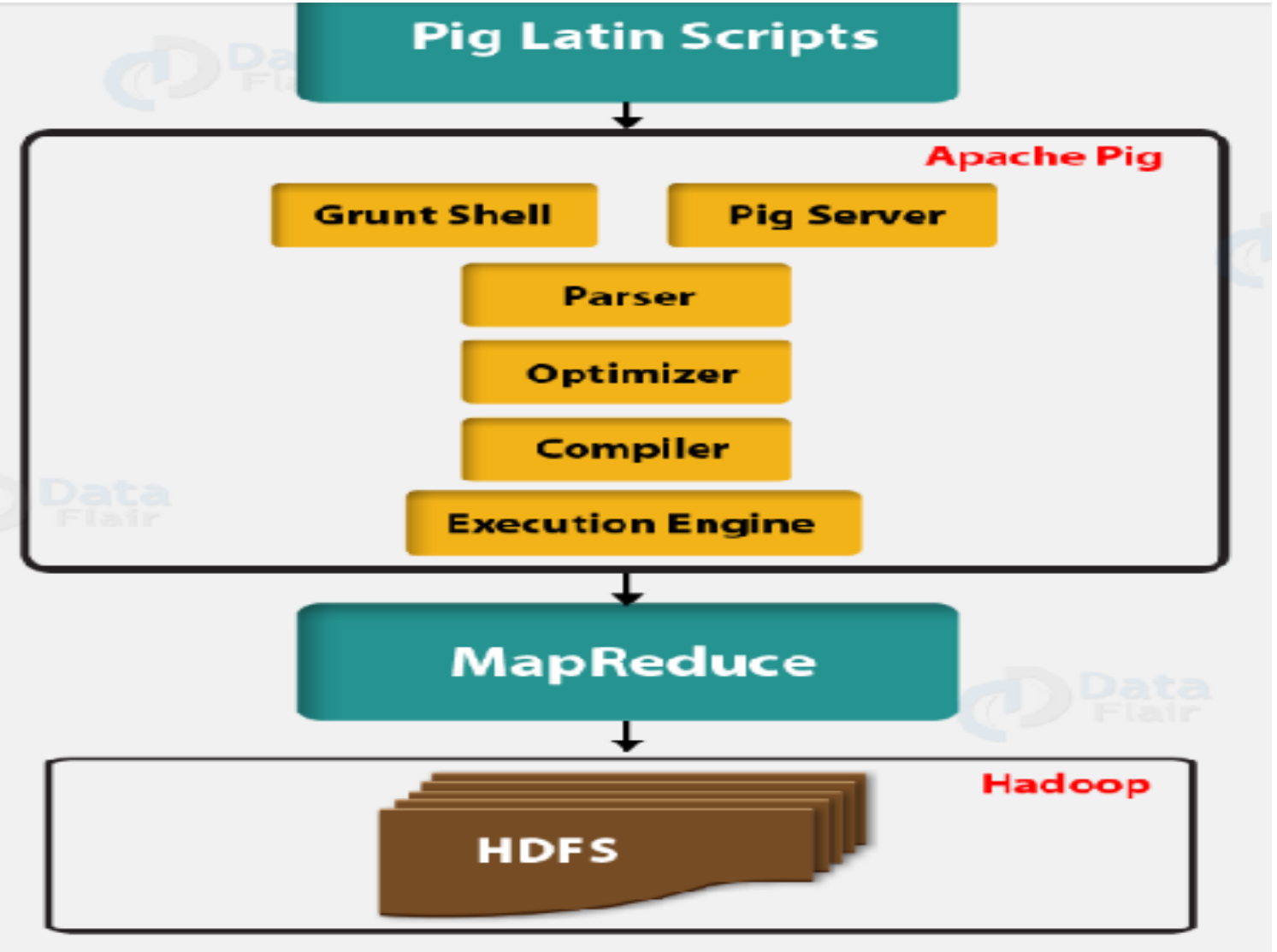
- Using Pig Latin, programmers can perform MapReduce tasks easily without having to type complex codes in Java.
- Apache Pig uses multi-query approach, thereby reducing the length of codes. For example, an operation that would require you to type 200 lines of code (LoC) in Java can be easily done by typing as less as just 10 LoC in Apache Pig. Ultimately, Apache Pig reduces the development time by almost 16 times.
- Pig Latin is SQL-like language and it is easy to learn Apache Pig when you are familiar with SQL.
- Apache Pig provides many built-in operators to support data operations like joins, filters, ordering, etc. In addition, it also provides nested data types like tuples, bags, and maps that are missing from MapReduce.

Pig Architecture

- In Pig, there is a language we use to analyze data in **Hadoop**. That is what we call Pig Latin. Also, it is a high-level data processing language that offers a rich set of data types and operators to perform several operations on the data.
- Moreover, in order to perform a particular task, programmers need to write a **Pig script using the Pig Latin language** and execute them using any of the execution mechanisms (Grunt Shell, UDFs, Embedded) using Pig.
- To produce the desired output, these scripts will go through a series of transformations applied by the Pig Framework, after execution.
- Further, Pig converts these scripts into a series of **MapReduce** jobs internally. Therefore it makes the programmer’s job easy. Here, is the architecture of Apache Pig.

Pig Architecture

- Architecture

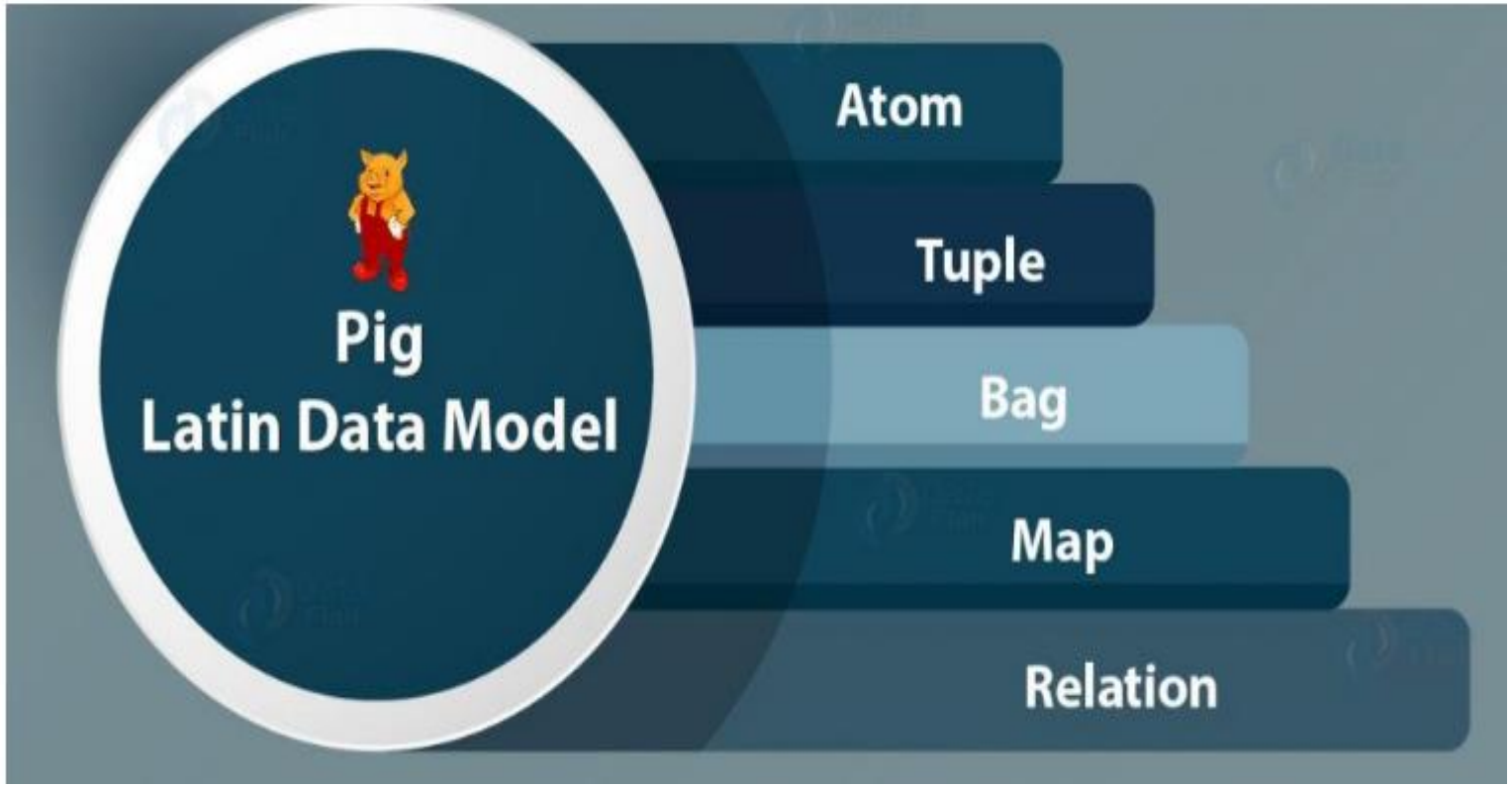


Apache Pig Components

- **Parser:** At first, all the Pig Scripts are handled by the Parser. Parser basically checks the syntax of the script, does type checking, and other miscellaneous checks. Afterwards, Parser's output will be a DAG (directed acyclic graph) that represents the Pig Latin statements as well as logical operators.
- The logical operators of the script are represented as the nodes and the data flows are represented as edges in DAG (the logical plan)
- **Optimizer:** Afterwards, the logical plan (DAG) is passed to the logical optimizer. It carries out the logical optimizations further such as projection and push down.
- **Compiler:** Then compiler compiles the optimized logical plan into a series of MapReduce jobs.
- **Execution engine**: Eventually, all the MapReduce jobs are submitted to Hadoop in a sorted order. Ultimately, it produces the desired results while these MapReduce jobs are executed on Hadoop.

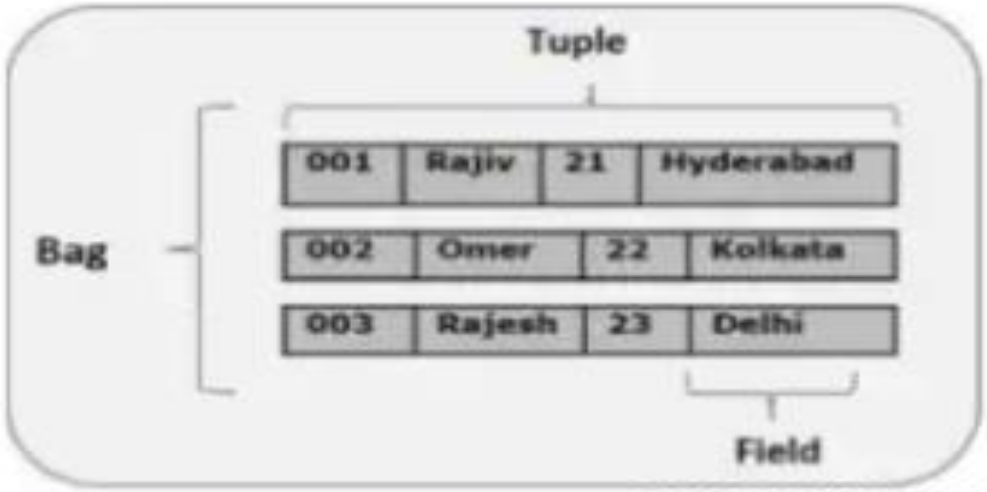
Pig Latin Data Model

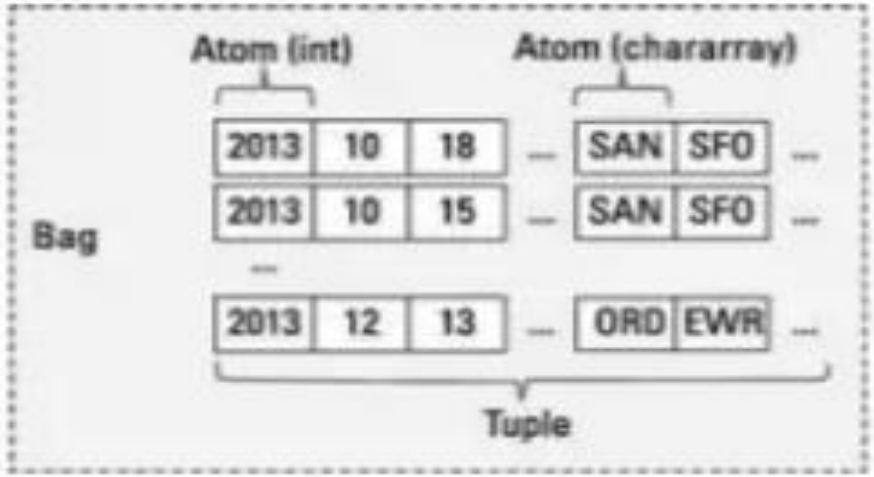
- Pig Latin data model is fully nested. Also, it allows complex non-atomic data types like map and tuple.

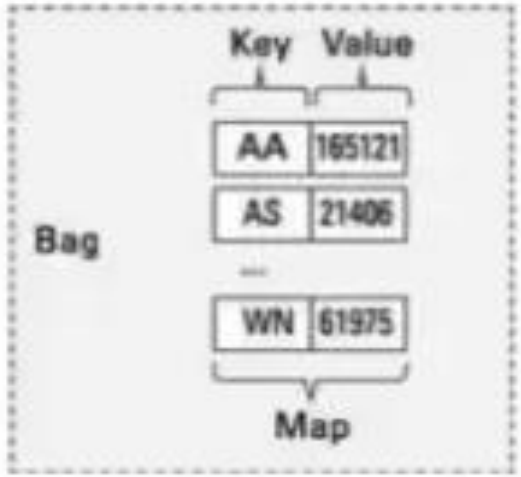


Pig Latin Data Model

- Apache Pig – Elements







Pig Latin Data Model

- **Atom**

➢ Atom is defined as any single value in Pig Latin, irrespective of their data. Basically, we can use it as string and number and store it as the string. Atomic values of Pig are int, long, float, double, char array, and byte array. Moreover, a field is a piece of data or a simple atomic value in Pig.

➢ For Example – 'Shubham' or '25'

- **Tuple**

➢ Tuple is a record that is formed by an ordered set of fields. However, the fields can be of any type. In addition, a tuple is similar to a row in a table of RDBMS.

For Example – (Shubham, 25)

Pig Latin Data Model

- **Bag**

➢ An unordered set of tuples is what we call Bag. To be more specific, a Bag is a collection of tuples (non-unique). Moreover, each tuple can have any number of fields (flexible schema). Generally, we represent a bag by '{}'.

➢ For Example − {(Shubham, 25), (Pulkit, 35)}

➢ In addition, when a bag is a field in a relation, in that way it is known as the **inner bag**.

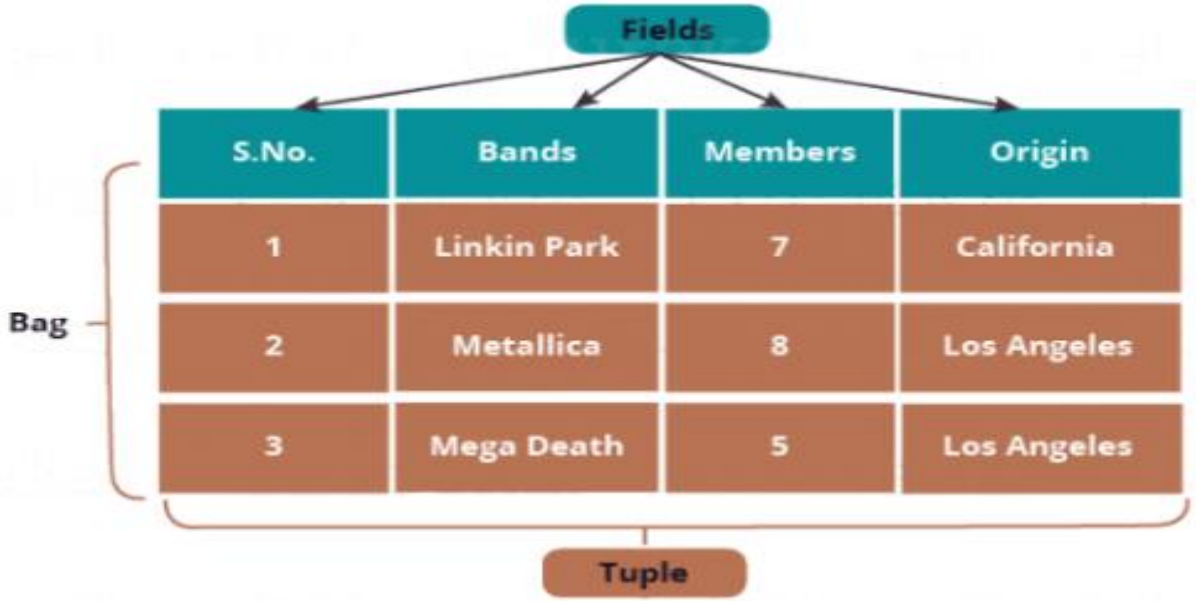
➢ Example − {Shubham, 25, {9826022258, Shubham@gmail.com,}}

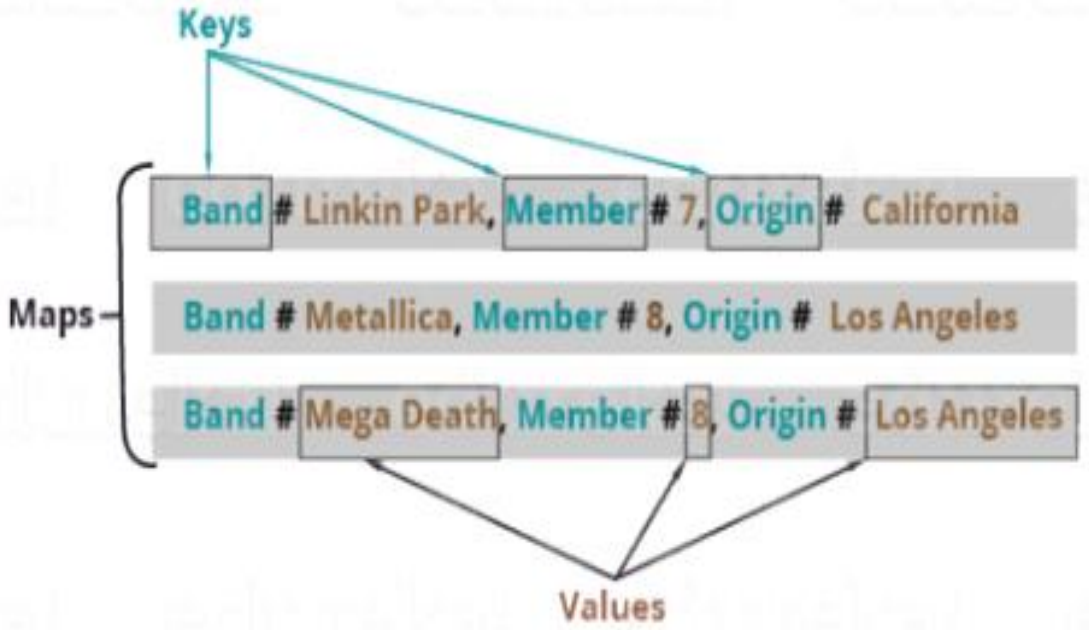
- **Map**

➢ A set of key-value pairs is what we call a map (or data map). Basically, the key needs to be of type char array and should be unique. Also, the value might be of any type. And, we represent it by '[]'

➢ For Example − [name#Shubham, age#25]

Pig Latin Data Model





Pig Latin Scripts

- **Pig Latin Scripts**
- We submit Pig scripts to the Apache Pig execution environment which can be written in Pig Latin using built-in operators.
- There are three ways to execute the Pig script:

➢ **Script File:** Write all the Pig commands in a script file and execute the Pig script file. This is executed by the Pig Server.

➢ **Grunt Shell:** This is Pig's interactive shell provided to execute all Pig Scripts.

➢ **Embedded Script:** If some functions are unavailable in built-in operators, we can programmatically create User Defined Functions to bring that functionalities using other languages like Java, Python, Ruby, etc. and embed it in Pig Latin Script file**.** Then, execute that script file.

Job Execution Flow

- The developer creates the scripts, and then it goes to the local file system as functions. Moreover, when the developers submit Pig Script, it contacts with Pig Latin Compiler.
- The compiler then splits the task and run a series of MR jobs. Meanwhile, Pig Compiler retrieves data from the **HDFS**. The output file again goes to the HDFS after running MR jobs.

Pig Execution Modes

- We can run Pig in two execution modes. These modes depend upon where the Pig script is going to run. It also depends on where the data is residing. We can thus store data on a single machine or in a distributed environment like Clusters.
- **Pig Local mode**

➢ In this mode, pig implements on **single JVM** and access the file system. This mode is better for dealing with the **small data sets**.

- Start the pig in local mode:

➢ **pig -x local**

- While the user can provide –x local to get into Pig local mode of execution. Therefore, Pig always looks for the **local file system path while loading data**.

Pig Execution Modes

- **Pig Map Reduce Mode**

➢ In this mode, a user could have proper **Hadoop cluster setup** and installations on it. By default, Apache Pig installs as in MR mode. The Pig also translates the queries into Map reduce jobs and runs on top of Hadoop cluster. Hence, this mode as a Map reduce runs on a **distributed cluster**.

- Start the pig in mapreduce mode (needs hadoop datanode started):

➢ **pig -x mapreduce**

- The statements like LOAD, STORE read the data from the HDFS file system and to show output. These Statements are also used to process data.

Features of Pig

- **Rich set of operators**: It provides many operators to perform operations like join, sort, filer, etc.
- **Ease of programming**: Pig Latin is similar to SQL and it is easy to write a Pig script if you are good at SQL.
- **Optimization opportunities**: The tasks in Apache Pig optimize their execution automatically, so the programmers need to focus only on semantics of the language.
- **Extensibility**: Using the existing operators, users can develop their own functions to read, process, and write data.
- **UDF's**: Pig provides the facility to create User-defined Functions in other programming languages such as Java and invoke or embed them in Pig Scripts.
- **Handles all kinds of data**: Apache Pig analyzes all kinds of data, both structured as well as unstructured. It stores the results in HDFS.

Applications of Apache Pig

- To process huge data sources such as web logs, streaming online data, etc.
- To perform data processing for search platforms (different types of data needs to be processed) like ***Yahoo uses Pig for 40% of their jobs including news feeds and search engine***.
- To process time sensitive data loads. Here, data needs to be extracted and analyzed quickly. E.g. machine learning algorithms requires time sensitive data loads, like twitter needs to quickly extract data of customer activities (i.e. tweets, re-tweets and likes) and analyze the data to find patterns in customer behaviors, and make recommendations immediately like trending tweets.

Twitter Case Study

- Their major aim was to analyse data stored in Hadoop to come up with the following insights on a daily, weekly or monthly basis.
- **Counting operations**:

➢ How many requests twitter serve in a day?
➢ What is the average latency of the requests?
➢ How many searches happens each day on Twitter?
➢ How many unique queries are received?
➢ How many unique users come to visit?

- **Correlating Big Data**:

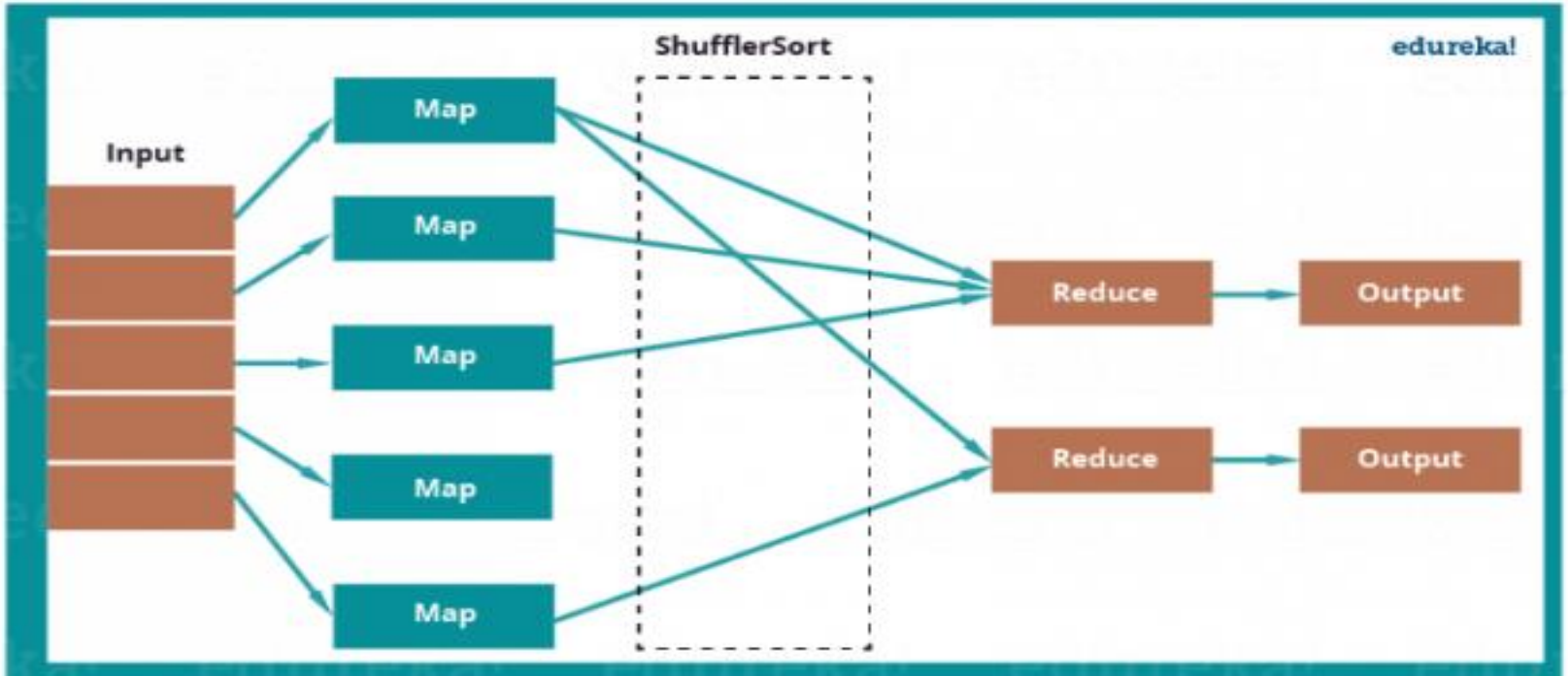
➢ Cohort analysis: analyzing data by categorizing user, based on their behavior.
➢ What goes wrong while site problem occurs?
➢ Search correction and search suggestions.

- **Research on Big Data & produce better outcomes like:**

➢ What is the ratio of the follower to following?

Twitter Case Study

- So, for analyzing data, Twitter used MapReduce initially, which is parallel computing over HDFS. For example, they wanted to analyse how many tweets are stored per user, in the given tweet table?
- Using MapReduce, this problem will be solved sequentially as shown in the below image:



Twitter Case Study

- MapReduce program first inputs the key as rows and sends the tweet table information to mapper function. Then the Mapper function will select the user id and associate unit value (i.e. 1) to every user id. The Shuffle function will sort same user ids together. At last, Reduce function will add all the number of tweets together belonging to same user. The output will be user id, combined with user name and the number of tweets per user.
- But while using MapReduce, they faced some limitations:

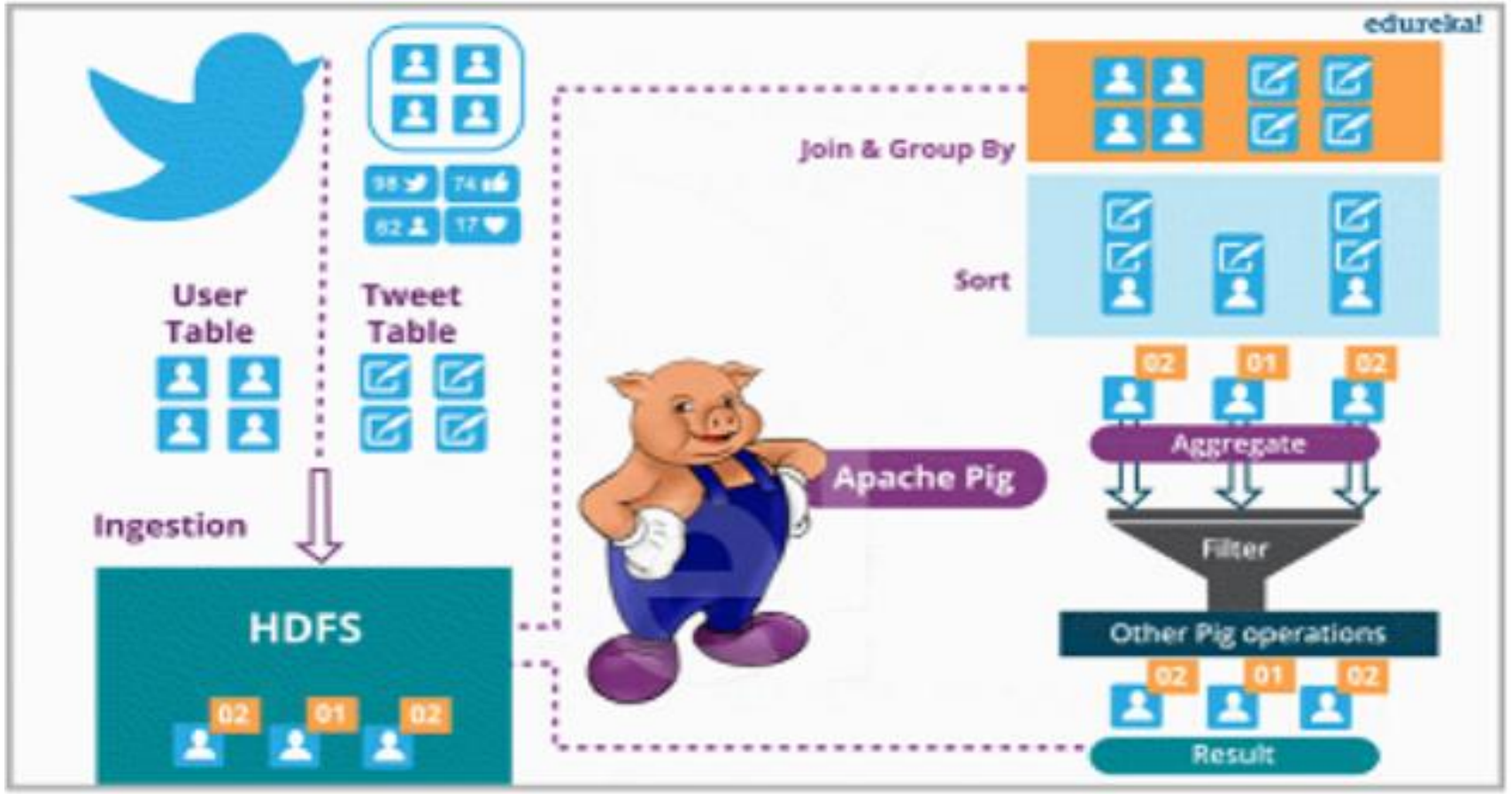
➢ Analysis needs to be typically done in Java.

➢ Joins, that are performed, needs to be written in Java, which makes it longer and more error-prone.

➢ For projection and filters, custom code needs to be written which makes the whole process slower.

Twitter Case Study

- So, Twitter moved to Apache Pig for analysis. Now, joining data sets, grouping them, sorting them and retrieving data becomes easier and simpler.

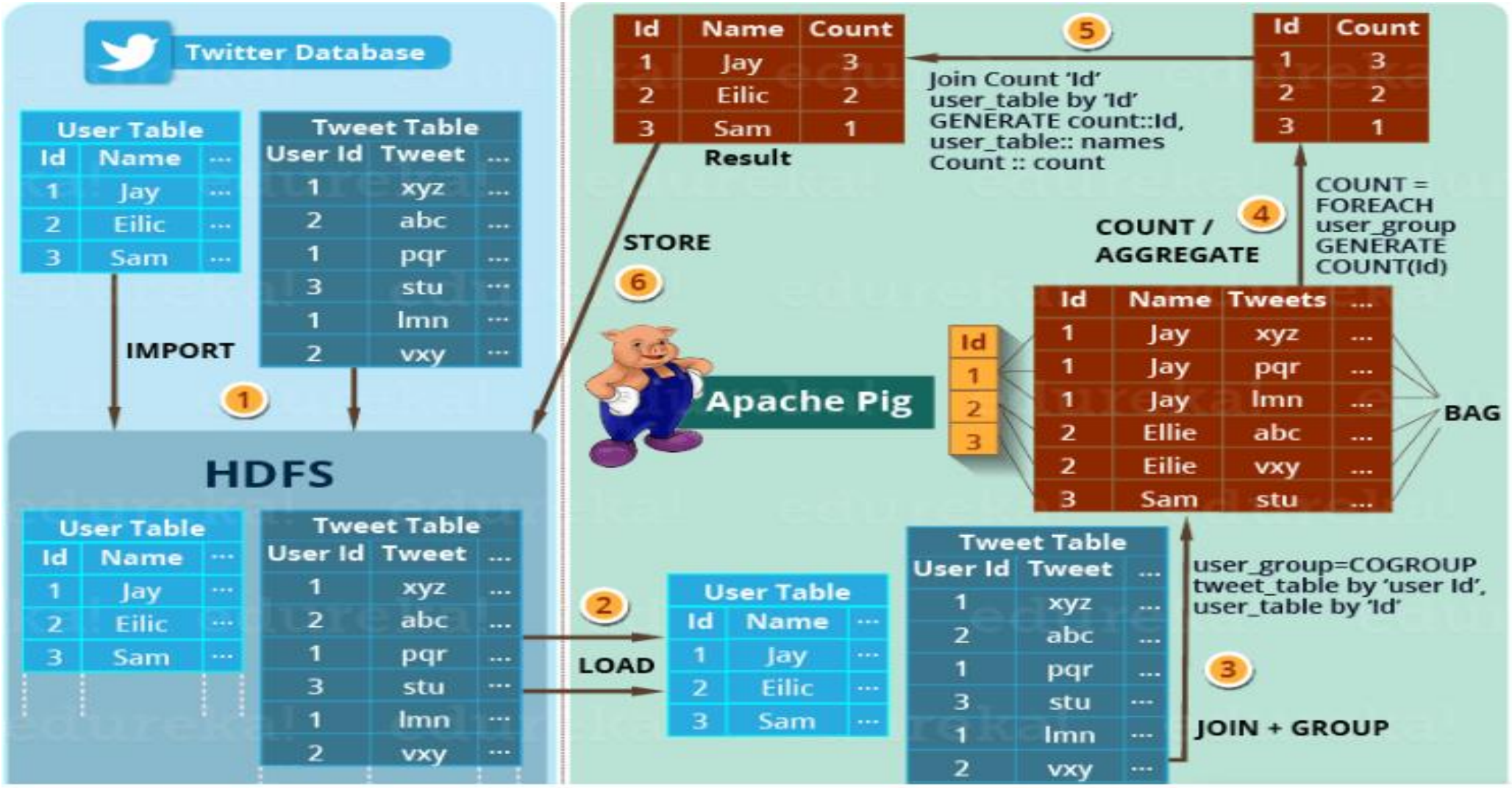


Twitter Case Study

- Twitter had both semi-structured data like Twitter Apache logs, Twitter search logs, Twitter MySQL query logs, application logs and structured data like tweets, users, block notifications, phones, favorites, saved searches, re-tweets, authentications, SMS usage, user followings, etc. which can be easily processed by Apache Pig.
- Twitter dumps all its archived data on HDFS. It has two tables i.e. user data and tweets data. User data contains information about the users like username, followers, followings, number of tweets etc. While Tweet data contains tweet, its owner, number of re-tweets, number of likes etc. Now, twitter uses this data to analyse their customer's behaviors and improve their past experiences.

Twitter Case Study

- Q: Analyzing how many tweets are stored per user, in the given tweet tables?



Twitter Case Study

- **STEP 1**– First of all, twitter imports the twitter tables (i.e. user table and tweet table) into the HDFS.
- **STEP 2**– Then Apache Pig loads (**LOAD**) the tables into Apache Pig framework.
- **STEP 3**– Then it joins and groups the tweet tables and user table using **COGROUP** command. This results in the inner Bag Data type.
- Example of Inner bags produced–

➢ (1,**{(1,Jay,xyz),(1,Jay,pqr),(1,Jay,lmn)}**)

➢ (2,**{(2,Ellie,abc),(2,Ellie,vxy)}**)

➢ (3, **{(3,Sam,stu)}**)

- **STEP 4**– Then the tweets are counted according to the users using **COUNT** command. So, that the total number of tweets per user can be easily calculated.

Twitter Case Study

- Example of tuple produced as (id, tweet count) (refer to the above image) –

➢ (1, **3**)

➢ (2, **2**)

➢ (3, **1**)

- **STEP 5**– At last the result is joined with user table to extract the user name with produced result.
- Example of tuple produced as (id, name, tweet count)

➢ (1, **Jay,** 3)

➢ (2, **Ellie,** 2)

➢ (3, **Sam**, 1)

- **STEP 6**– Finally, this result is stored back in the HDFS.

Hive vs. Pig

Features	Hive	Pig
Language	Hive uses a declarative language called HiveQL	With Pig Latin, a procedural data flow language is used
Data Processing	Hive is used for batch processing	Pig is a high-level data-flow language
Partitions	Yes	No. Pig does not support partitions although there is an option for filtering
Web interface	Hive has a web interface	Pig does not support web interface
User Specification	Data analysts are the primary users	Programmers and researchers use Pig
Used for	Reporting	Programming

Hive vs. Pig

Features	Hive	Pig
Type of data	Hive works on structured data. Does not work on other types of data	Pig works on structured, semi-structured and unstructured data
Operates on	Works on the server-side of the cluster	Works on the client-side of the cluster
Avro File Format	Hive does not support Avro	Pig supports Avro
Loading Speed	Hive takes time to load but executes quickly	Pig loads data quickly
JDBC/ ODBC	Supported, but limited	Unsupported

Hive v/s Pig

Similarities:

- Both High level Languages which work on top of map reduce framework
- Can coexist since both use the under lying HDFS and map reduce

Differences:

- **Language**
  - Pig is a procedural ; (A = load ‘mydata’; dump A)
  - Hive is Declarative (select * from A)

- **Work Type**
  - Pig more suited for adhoc analysis (on demand analysis of click stream search logs)
  - Hive a reporting tool (e.g. weekly BI reporting)

Hive v/s Pig

Differences:

- **Users**
  - Pig – Researchers, Programmers (build complex data pipelines, machine learning)
  - Hive – Business Analysts
- **Integration**
  - Pig - Doesn’t have a thrift server(i.e no/limited cross language support)
  - Hive - Thrift server
- **User’s need**
  - Pig – Better dev environments, debuggers expected
  - Hive - Better integration with technologies expected(e.g JDBC, ODBC)

Hive vs. Pig vs. Hadoop MapReduce

	Hive	Pig	Hadoop MapReduce
Language	It has SQL like Query language.	It has the scripting language.	compiled language.
Abstraction	It has a Low level of Abstraction.	has the High level of Abstraction.	has the High level of Abstraction.
Line of codes	Comparatively less no. of the line of codes from both MapReduce and Pig.	Comparatively less no. of the line of codes from MapReduce.	It has More line of codes.
Development Efforts	Comparatively fewer development efforts from both MapReduce and Pig.	Comparatively less development effort.	More development effort is involved.
Code Efficiency	Code efficiency is relatively less.	Code efficiency is relatively less.	It has high Code efficiency.

Distributed Database

- Distributed databases are a collection of data stored at different sites of a computer network.
- A distributed database can be created by splitting and scattering the data of an existing database over different sites or by federating together multiple existing databases.
- Distributed database may be required when a particular database needs to be accessed by various users globally.
- It needs to be managed such that for the users it looks like one single database.

Types of Distributed Database

1. Homogeneous Database:

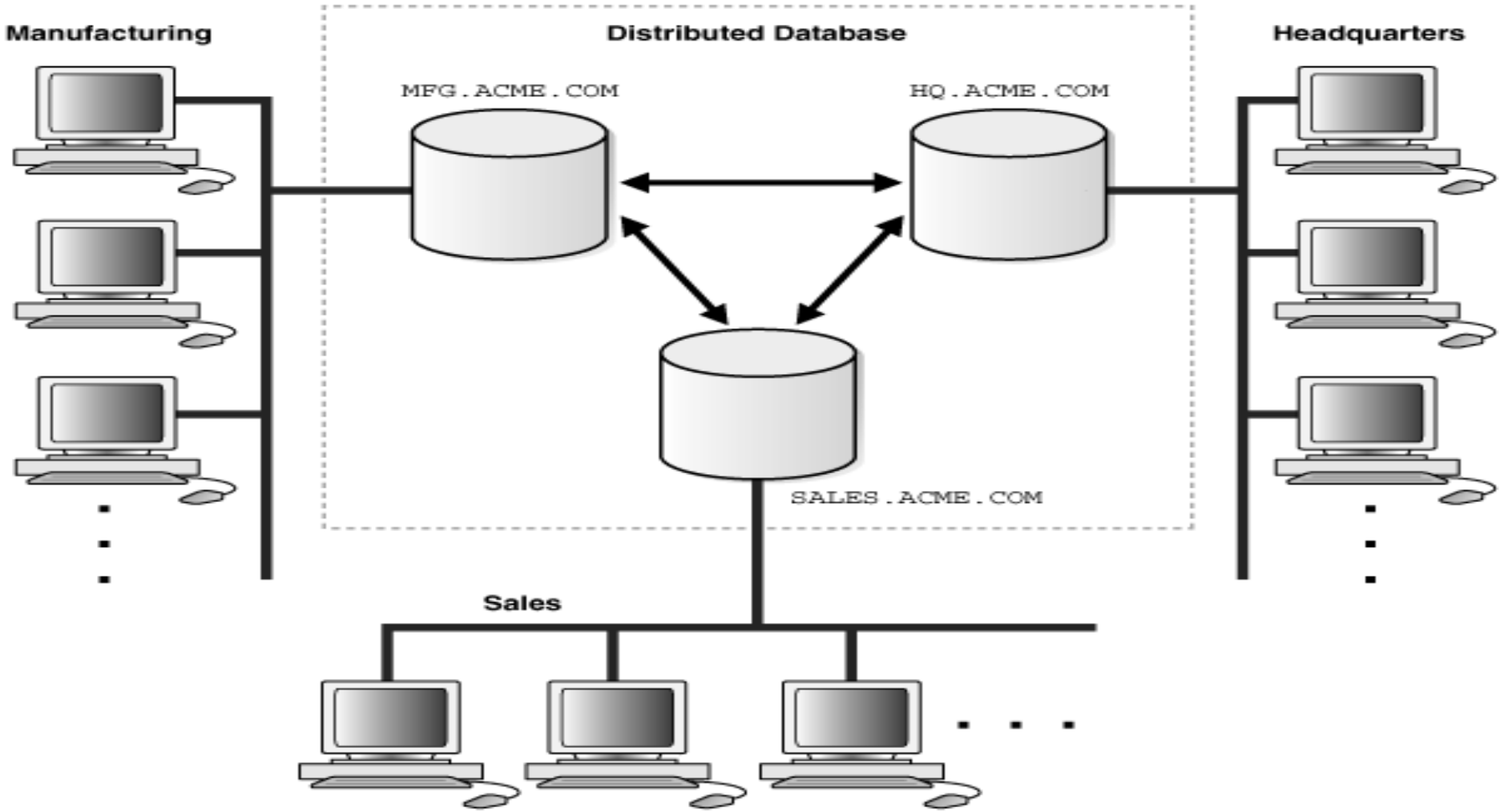
- In a homogeneous database, all different sites store database identically. The operating system, database management system and the data structures used – all are same at all sites. Hence, they're easy to manage.

- Types of Homogeneous Distributed Database

- **Autonomous** − Each database is independent that functions on its own. They are integrated by a controlling application and use message passing to share data updates.
- **Non-autonomous** − Data is distributed across the homogeneous nodes and a central or master DBMS co-ordinates data updates across the sites.

Types of Distributed Database

1. Homogeneous Database:



Types of Distributed Database

2. Heterogeneous Database:

- In a heterogeneous distributed database, different sites can use different schema and software that can lead to problems in query processing and transactions. Also, a particular site might be completely unaware of the other sites. Different computers may use a different operating system, different database application. They may even use different data models for the database.
- Hence, translations are required for different sites to communicate.

Distributed Data Storage

- There are 2 ways in which data can be stored on different sites. These are:

1. Replication –

- In this approach, the entire relation is stored redundantly at 2 or more sites. If the entire database is available at all sites, it is a fully redundant database. Hence, in replication, systems maintain copies of data.
- This is advantageous as it increases the availability of data at different sites. Also, now query requests can be processed in parallel. However, it has certain disadvantages as well. Data needs to be constantly updated. Any change made at one site needs to be recorded at every site that relation is stored or else it may lead to inconsistency. This is a lot of overhead. Also, concurrency control becomes way more complex as concurrent access now needs to be checked over a number of sites.

Distributed Data Storage

2. Fragmentation –

- In this approach, the relations are fragmented (i.e., they're divided into smaller parts) and each of the fragments is stored in different sites where they're required. It must be made sure that the fragments are such that they can be used to reconstruct the original relation (i.e, there isn't any loss of data).
- Fragmentation is advantageous as it doesn't create copies of data, consistency is not a problem.

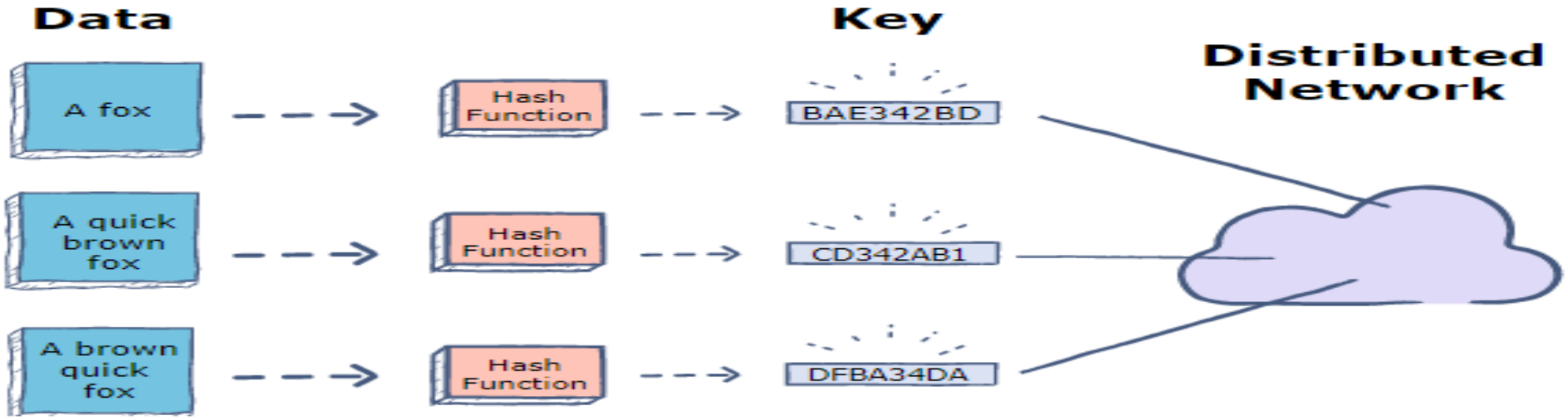
Distributed Hash Tables

- A distributed hash table (DHT) is a **decentralized** storage system that provides lookup and storage schemes similar to a hash table, storing key-value pairs.
- Each node in a DHT is responsible for keys along with the mapped values. Any node can efficiently retrieve the value associated with a given key. Just like in hash tables, values mapped against keys in a DHT can be any arbitrary form of data.
- Distributed hash tables are decentralized, so all nodes form the collective system without any centralized coordination.
- They are generally fault-tolerant because data is replicated across multiple nodes. Distributed hash tables can scale for large volumes of data across many nodes.

Distributed Hash Table

- Distributed Hash Tables are a form of a distributed database that can store and retrieve information associated with a key in a network of peer nodes that can join and leave the network at any time.
- The nodes coordinate among themselves to balance and store data in the network without any central coordinating party.
- DHT’s require that information to be evenly distributed across the network. To achieve this goal, the concept of consistent hashing is used. A key is passed through a hash algorithm that serves as a randomization function. This ensures that each node in the network has an equal chance of being chosen to store the key/value pair.

Distributed Hash Tables



- DHTs have the following properties:

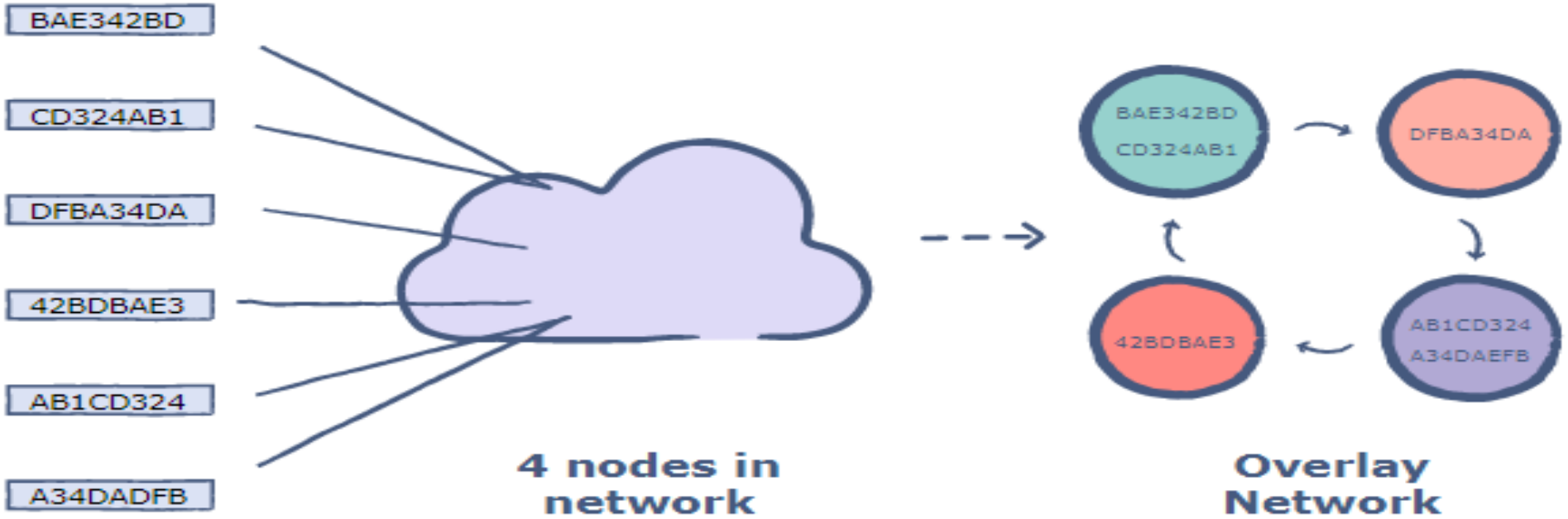
➢ **Decentralised & Autonomous:** Nodes collectively form the system without any central authority.

➢ **Fault Tolerant:** System is reliable with lots of nodes joining, leaving, and failing at all times.

➢ **Scalable:** System should function efficiently with even thousands or millions of nodes.

Distributed Hash Tables

- Just like hash tables, DHTs support the following 2 functions:
  - ➢ put (key, value)
  - ➢ get (key)
- The nodes in a DHT are connected together through an **overlay network** in which neighboring nodes are connected. This network allows the nodes to find any given key in the key-space.



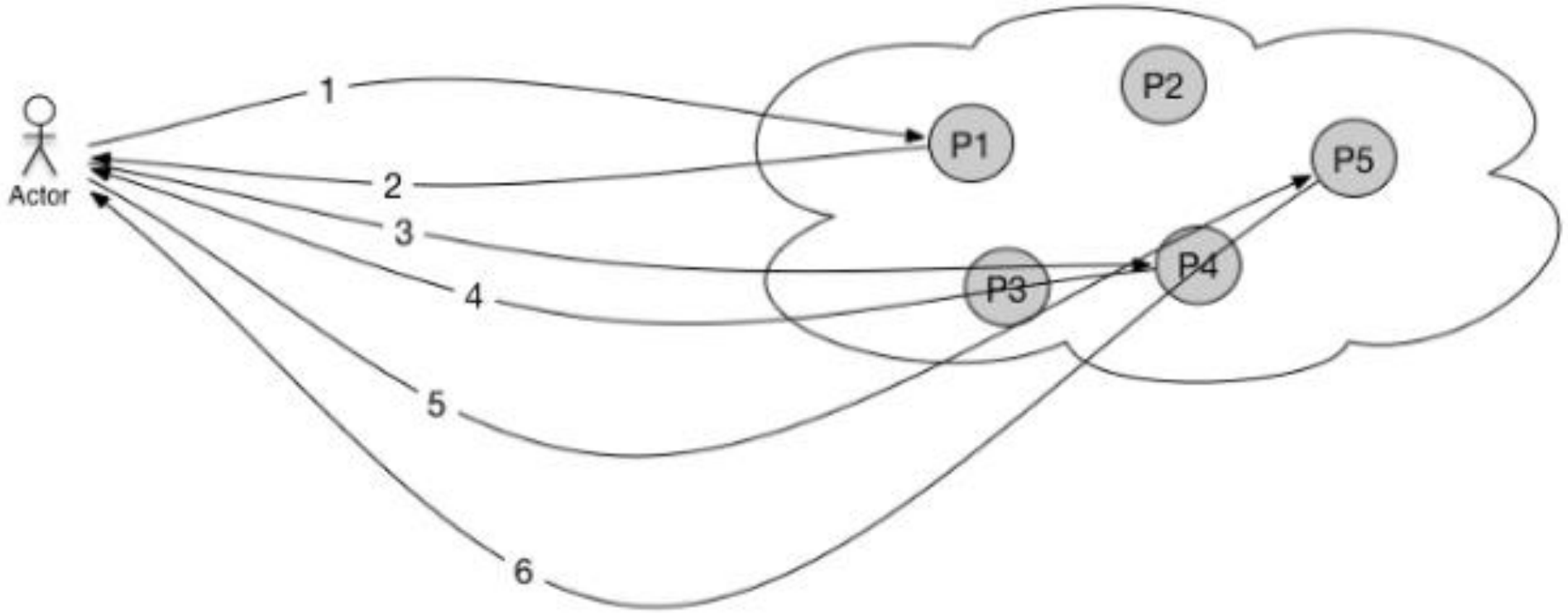
Why Is a Distributed Hash Table Used?

- Distributed hash tables provide an easy way to find information in a large collection of data because all keys are in a consistent format, and the entire set of keys can be partitioned in a way that allows fast identification on where the key/value pair resides.
- The nodes participating in a distributed hash table act as peers to find specific data values, as each node stores the key partitioning scheme so that if it receives a request to access a given key, it can quickly map the key to the node that stores the data. It then sends the request to that node.
- Also, nodes in a distributed hash table can be easily added or removed without forcing a significant amount of re-balancing of the data in the cluster. Cluster rebalancing, especially for large data sets, can often be a time-consuming task that also impacts performance. Having a quick and easy means for growing or shrinking a cluster ensures that changes in data size does not disrupt the operation of the applications that access data in the distributed hash table.

Iterative and Recursive Lookup

- There are two common styles of lookup operations.
- In an iterative lookup, a requesting node will query another node asking for a key/value pair. If that node does not have that node, it will return one or more nodes that are “closer”. The requesting node will then query the next closest node. This process repeats until either the key/value is found and returned, or the last queried node returns an error saying that the key simply cannot be found.
- With recursive lookups, the first requesting node will query the next closes node, which will then query the next closest node until the data is found. The data is then passed back along the chain of requests back to the requestor.

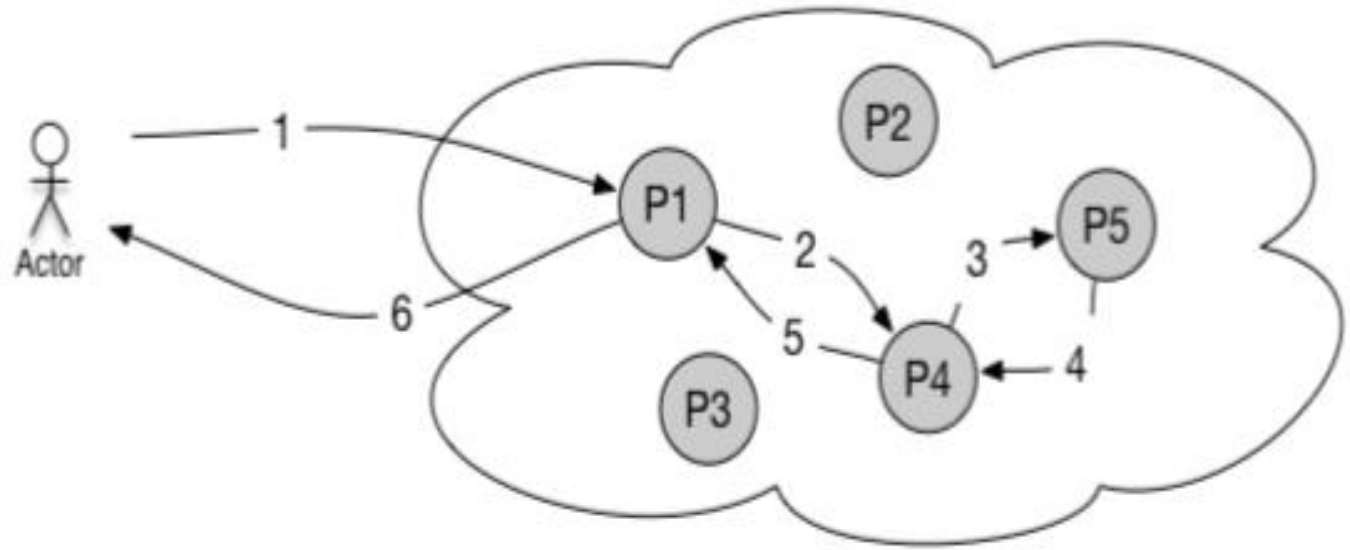
Iterative Lookup



1. STORE “MyKey” / “My Value”
2. I’m not responsible for “MyKey” - but P4 is closer
3. STORE “MyKey” / “My Value”
4. I’m not responsible for “MyKey” - but P5 is closer
5.STORE “MyKey” / “My Value”
6. OK - value is stored.

1. GET “MyKey”
2. I’m not responsible for “MyKey” - but P4 is closer
3. GET “MyKey”
4. I’m not responsible for “MyKey” - but P5 is closer
5.GET “MyKey”
6. OK - here is “My Value”

Recursive Lookup



1. STORE “MyKey” / “My Value”
2. I’m not responsible for “MyKey”, P4 is closer, so ask P4
3. I’m not responsible for “MyKey”, P5 is closer, so ask P5
4. OK, value is stored
5. OK, value is stored
6. OK, value is stored

1. GET “MyKey”
2. I’m not responsible for “MyKey”, P4 is closer, so ask P4
3. I’m not responsible for “MyKey”, P5 is closer, so ask P5
4. OK, here is “My Value”
5. OK, passing back “My Value”
6. OK, passing back “My Value”

Routing Table

- DHT nodes don’t store all the data, there needs to be a routing layer so that any node can locate the node that stores a particular key. The mechanics of how the routing table works, and how the table is updated as nodes join and leave the network is a key differentiator between **different DHT algorithms**.
- In general, DHT nodes will only store a portion of the routing information in the network. This means that when a node joins the network, routing information only needs to be disseminated to a small number of peers.
- **Because each node only contains a portion of the routing table, the process of finding or storing a key/value pair requires contacting multiple nodes.** The number of nodes that needs to be contacted is related to the amount of routing information each node stores. Common lookup complexity is O(log n) where n is the number of nodes in the network.

Routing Table

I want to find my key of 66

1. I query node 1 for the key of 66
1. Node 1 is not responsible for 66, but Node 58 is the closest so ask 58
3. Query node 58 for the key of 66
4. Node 58 is not responsible for the key 66, but node 63 is the closes, so ask 63
5. Query node 63 for key of 66
6. Node 66 has the key of 66, return the value

Node / ID	Keyspace Range	Routing Table Node ID:Range	Database
1	1-10	Node 22: 21-30 Node 58: 51-60	"5":"some value"
22	21-30	Node 36: 21-30 Node 63: 61-70	"23": "some value" "27": "some other value"
36	31-40	Node 41: 41-50 Node 77: 71-80	Nothing
41	41-50	Node 58: 51-60 Node 89: 81-90	"48":"some value"
58	51-60	Node 63: 61-70 Node 95: 91-100	"54":"some value" "56":some value"
63	61-70	Node 77: 71-80 Node 1: 1-10	"66":"some value"
77	71-80	Node 89: 81-90 Node 22: 21-30	"71":"some value"
89	81-90	Node 95: 91-100 Node 36: 31-40	"89":"some value"
95	91-100	Node 1: 1-10 Node 41: 41-50	"97":"some value"

Thank you.