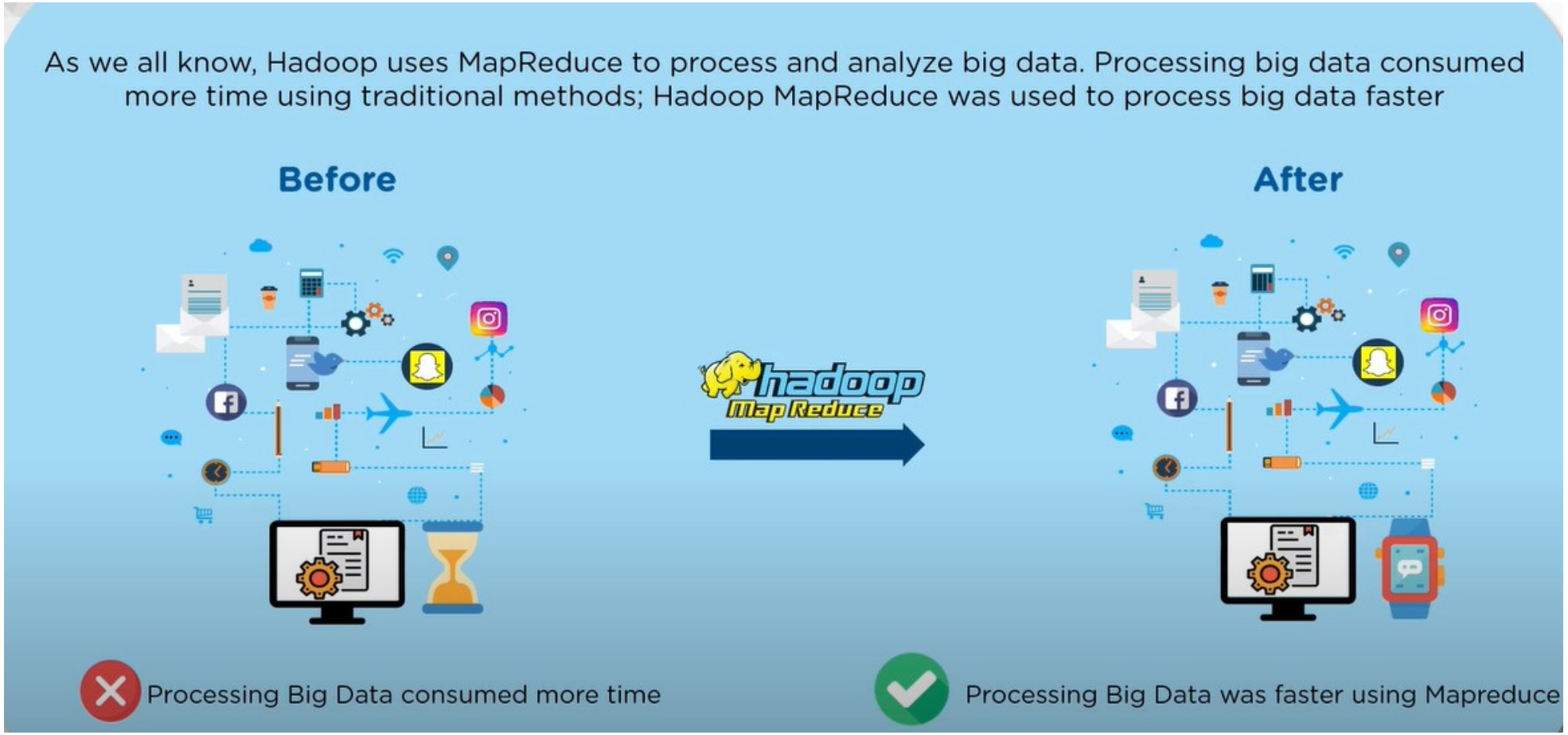
Pig and Hive

Praveen Kumar Chandaliya, DoAI, SVNIT

Pig and Hive



Source: https://www.youtube.com/watch?v=LEzQVPvNTNo

As we all know, Hadoop uses MapReduce to process and analyze big data. Processing big data consumed more time using traditional methods; Hadoop MapReduce was used to process big data faster

Before

MapReduce is primarily implemented using Java codes. Lengthy complex codes were written by programmers to process data

Processing Big Data consumed more time

After

This proved to be a disadvantage for users who were non-programmers. To overcome this issue, **Hive and Pig** were introduced

Processing Big Data was faster using Mapreduce

Need for Hive & Pig

What is Hive & Pig

HiveQL & Pig Latin

Data models

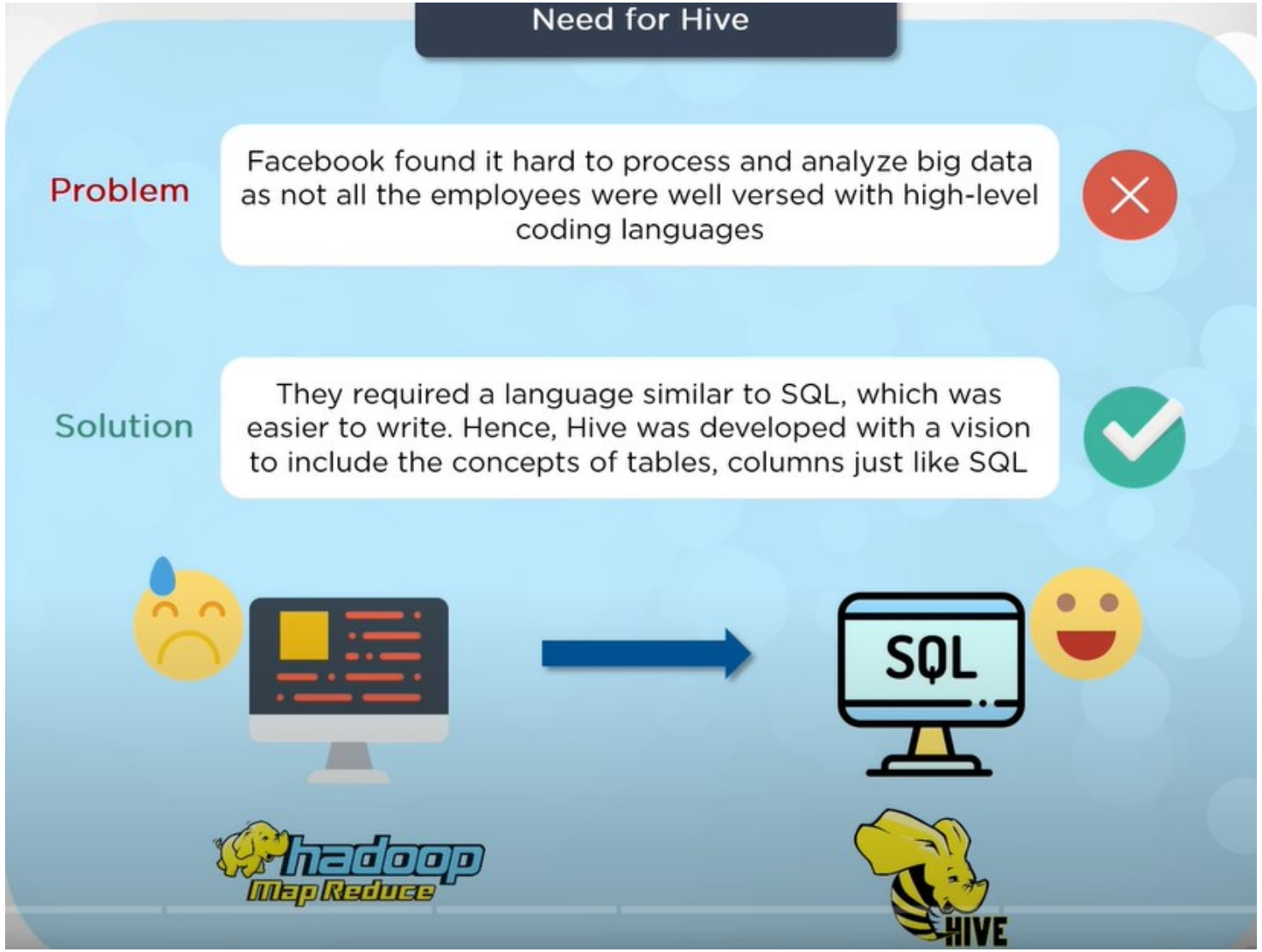
Execution Modes

Features

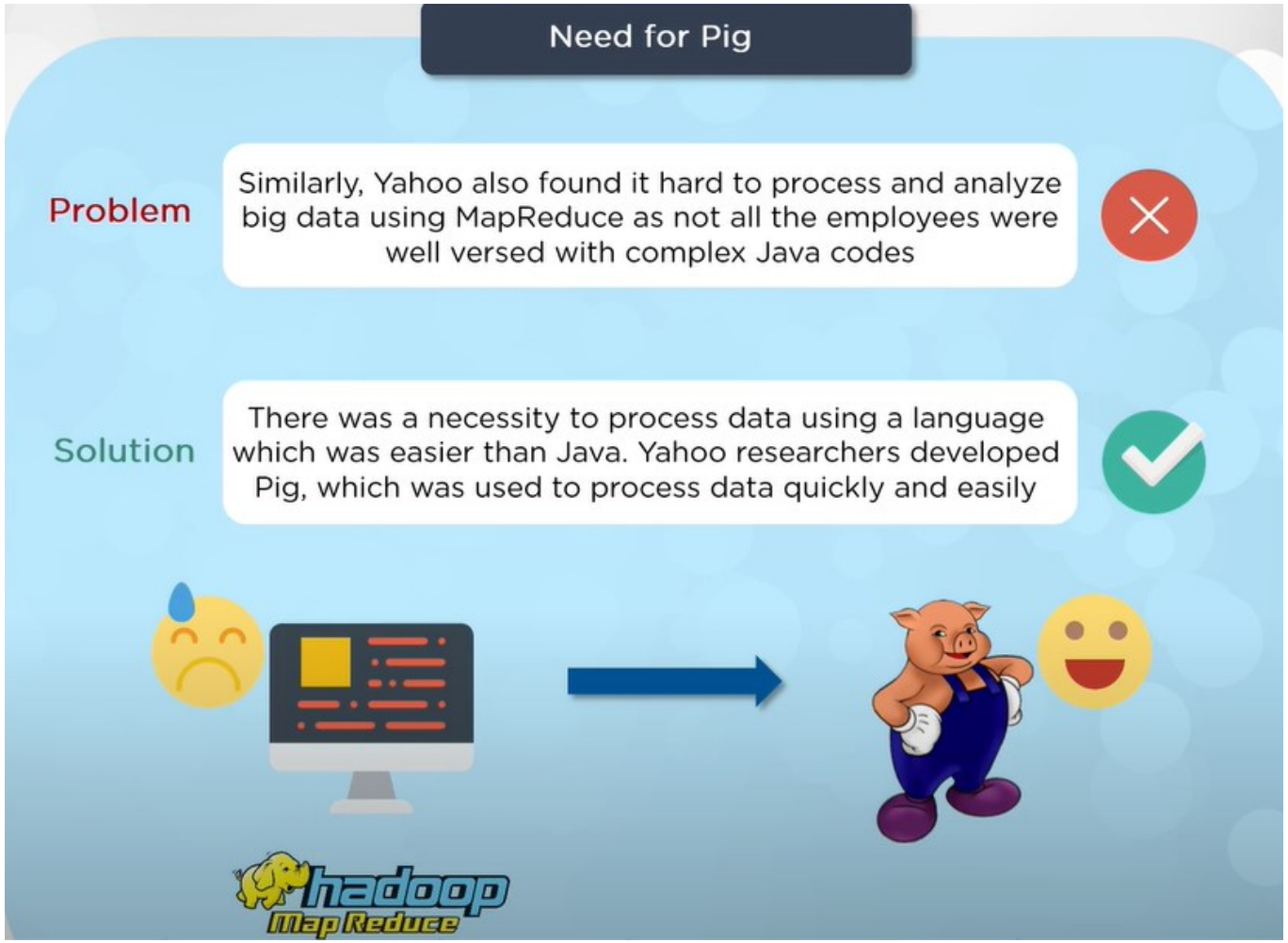
Commands

Source: https://www.youtube.com/watch?v=LEzQVPvNTNo https://www.youtube.com/watch?v=2q3F-mcmzGQ

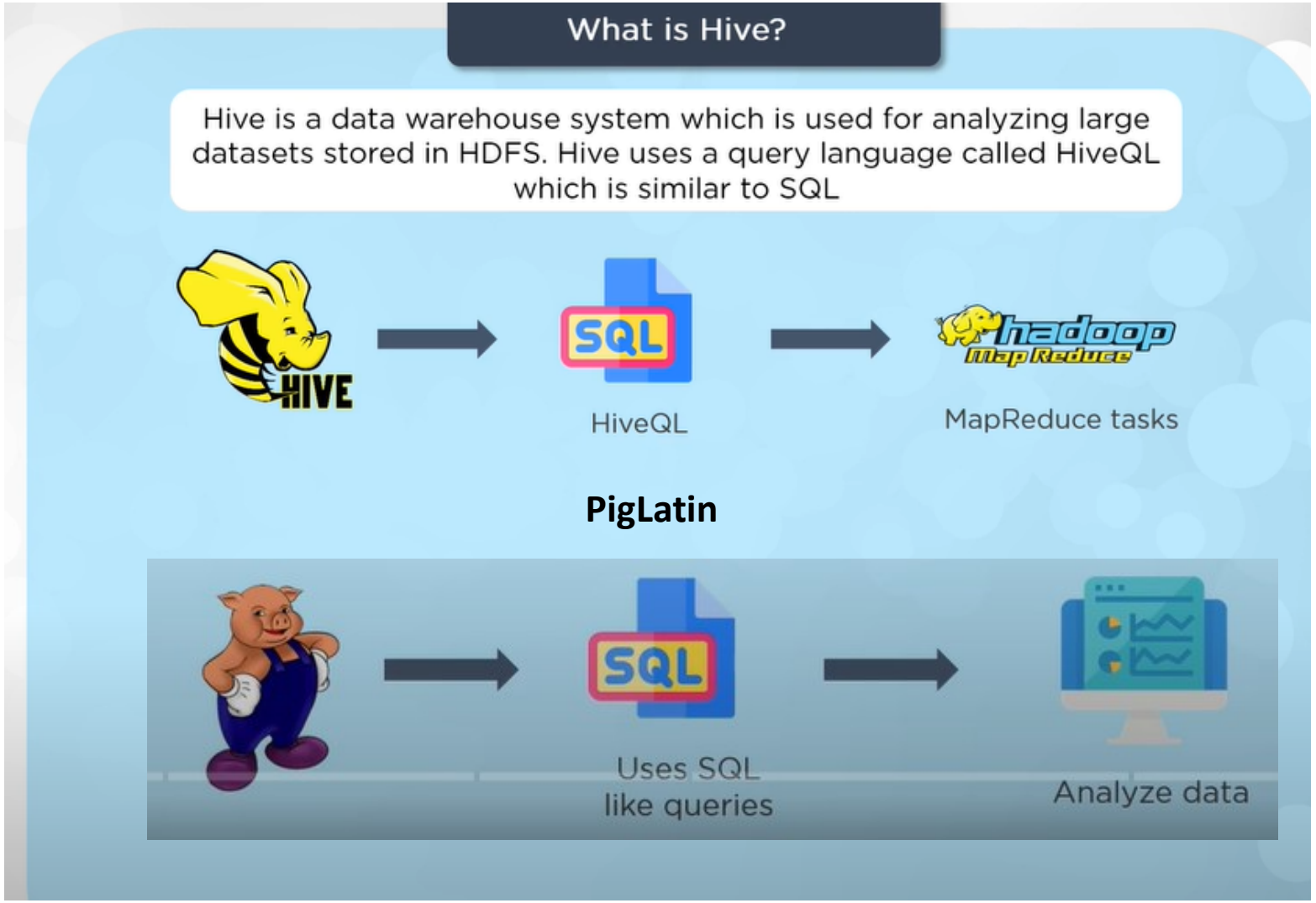
- Need for Hive & Pig
- What is Hive & Pig
- HiveQL & Pig Latin
- Data models
- Execution Modes
- Features
- Commands



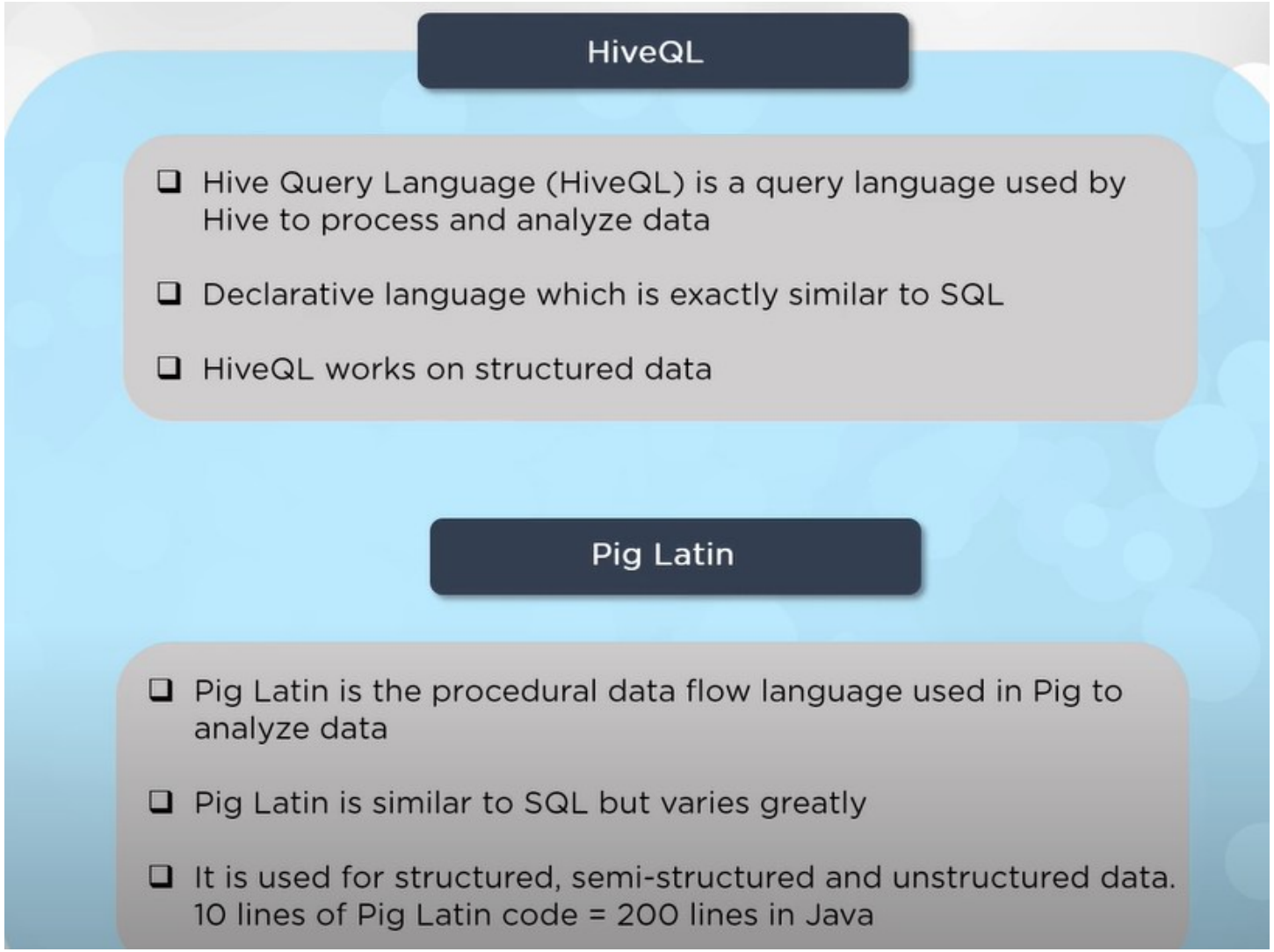
- Need for Hive & Pig
- What is Hive & Pig
- HiveQL & Pig Latin
- Data models
- Execution Modes
- Features
- Commands



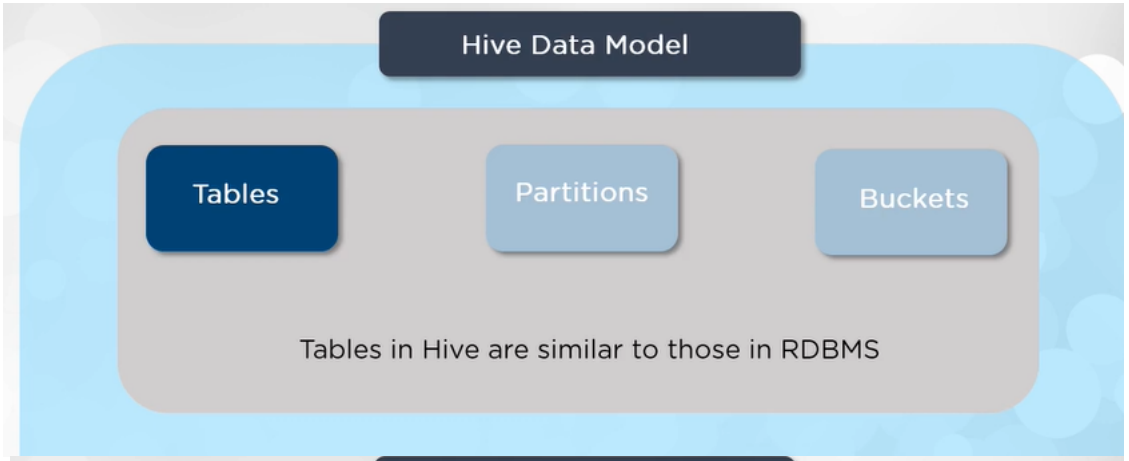
- Need for Hive & Pig
- What is Hive & Pig
- HiveQL & Pig Latin
- Data models
- Execution Modes
- Features
- Commands

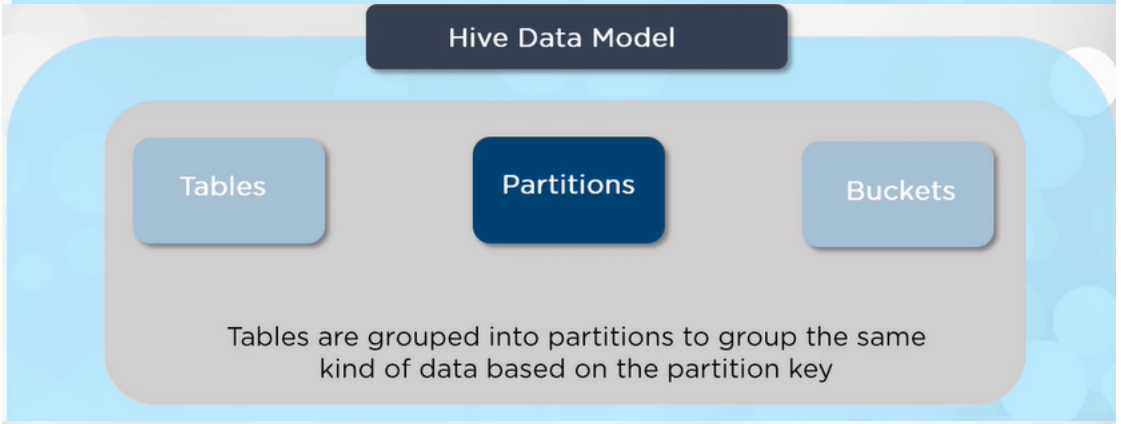


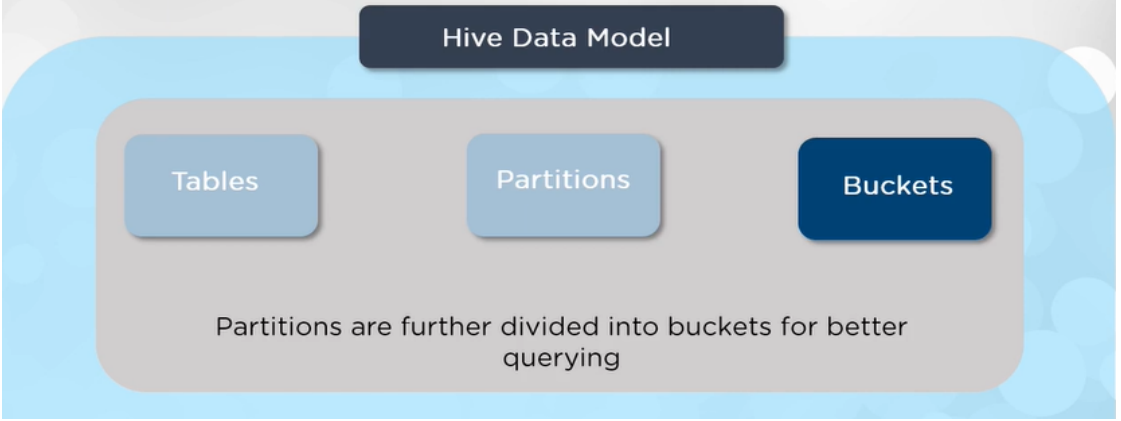
- Need for Hive & Pig
- What is Hive & Pig
- HiveQL & Pig Latin
- Data models
- Execution Modes
- Features
- Commands



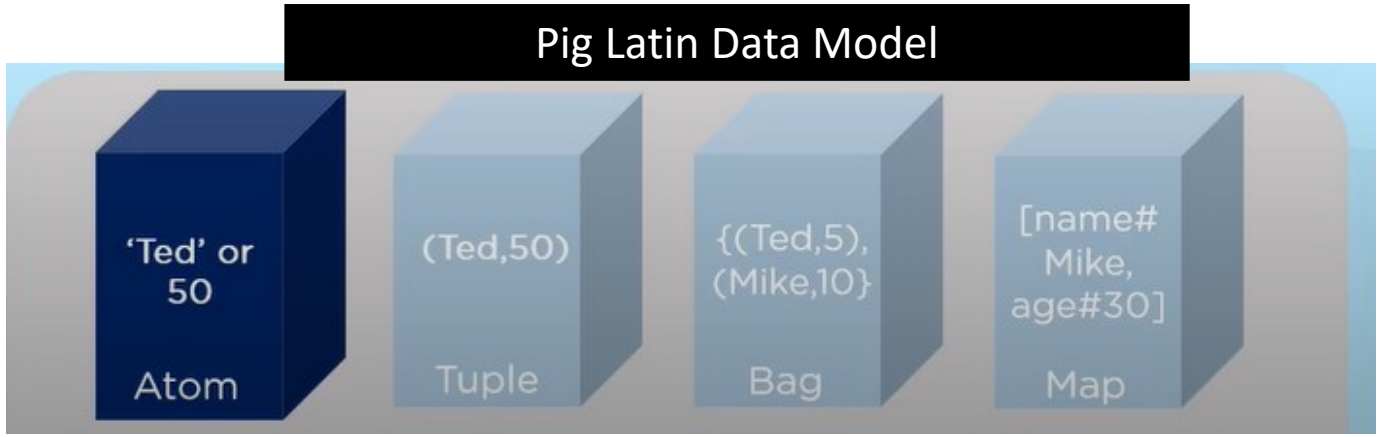
- Need for Hive & Pig
- What is Hive & Pig
- HiveQL & Pig Latin
- Data models
- Execution Modes
- Features
- Commands





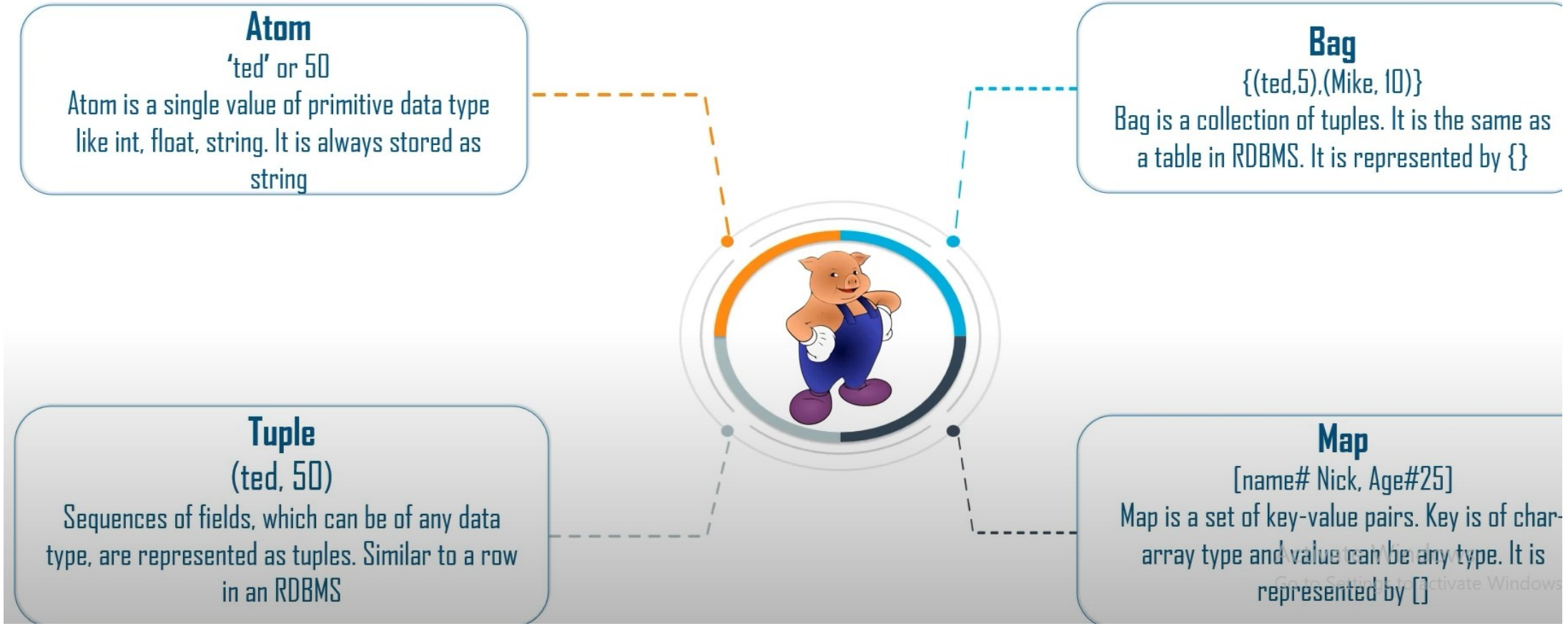


- Need for Hive & Pig
- What is Hive & Pig
- HiveQL & Pig Latin
- Data models
- Execution Modes
- Features
- Commands

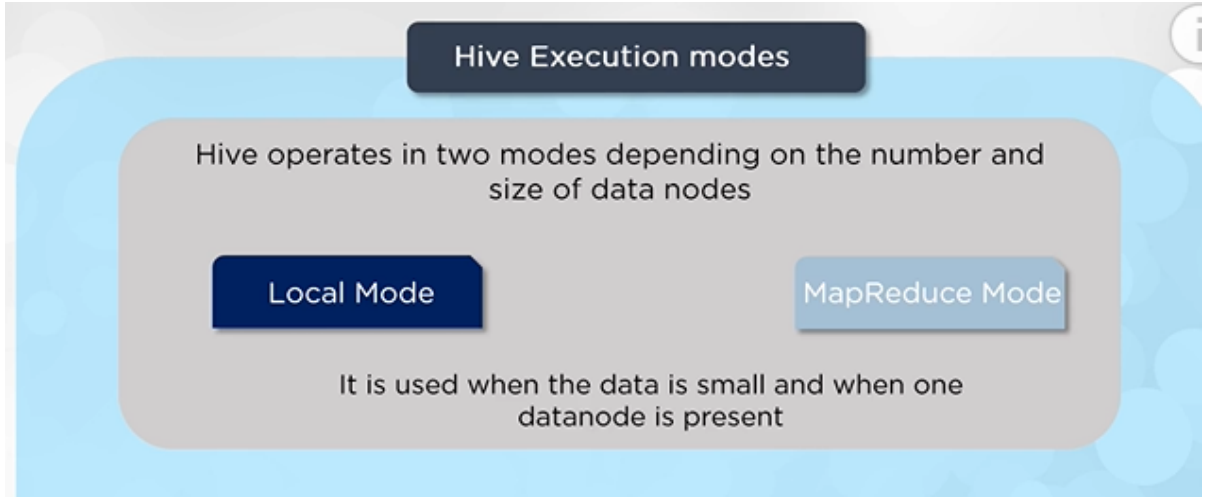


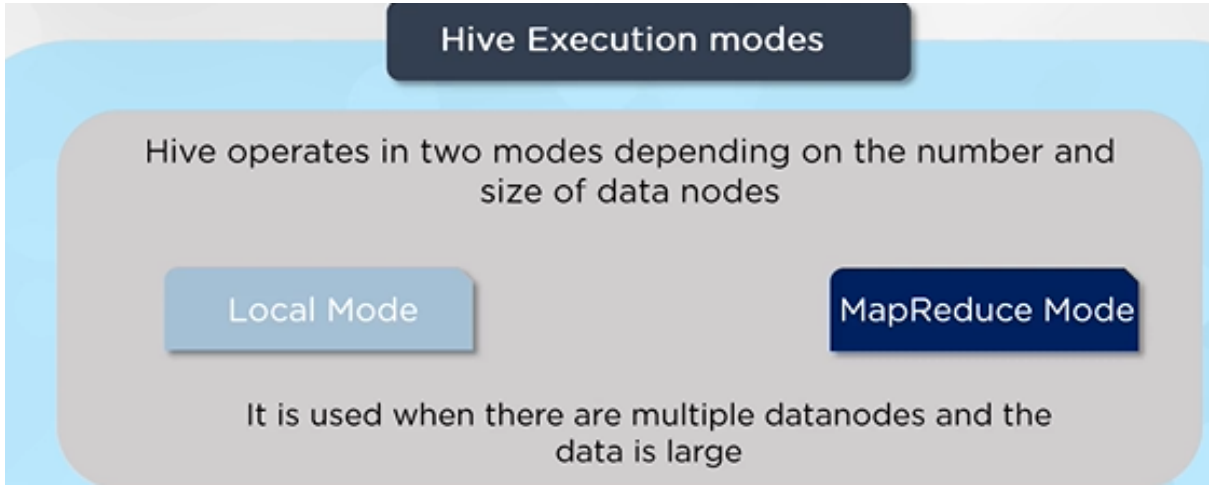
- **Atom** is a single value of primitive data type like int, float, string. It is always store as string.
- **Tuple** represents sequence of fields that can be of any data type. It is same as a row in RDDMS
- **Bag** is a collection of tuples. It is the same as a table in 'RDBMS'. It is represent by {}
- **Map** is a set of key-value pairs. Key is of chararray type and value can be of any type. It is represent by [].

Pig Latin Data Model

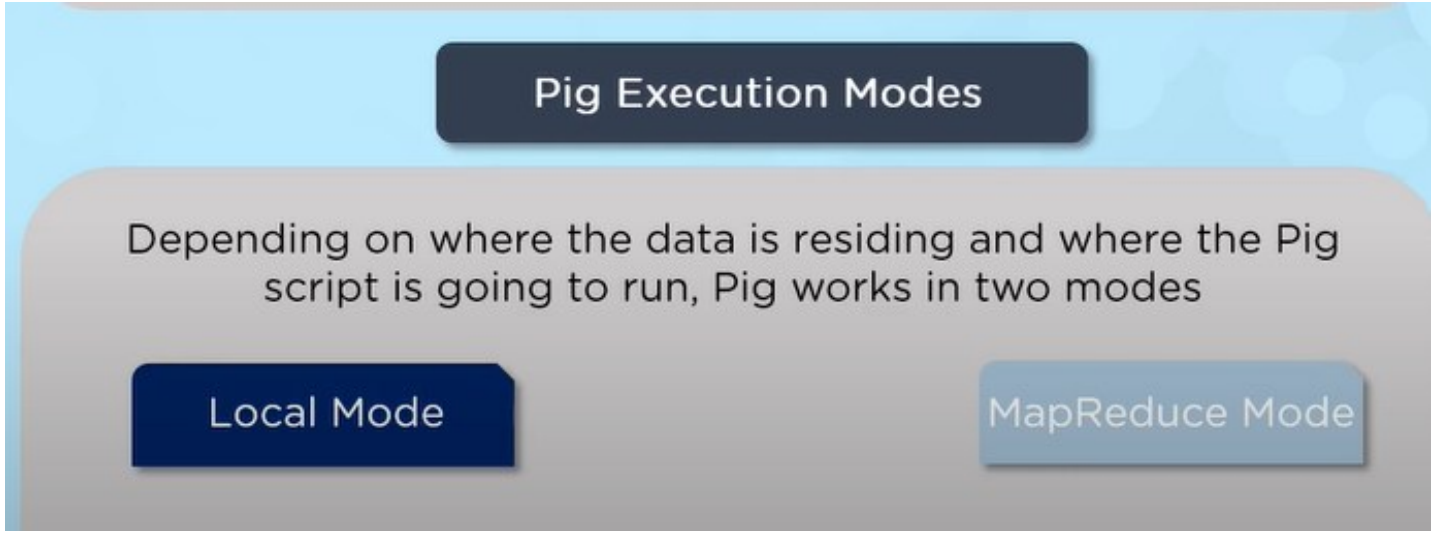


- Need for Hive & Pig
- What is Hive & Pig
- HiveQL & Pig Latin
- Data models
- Execution Modes
- Features
- Commands





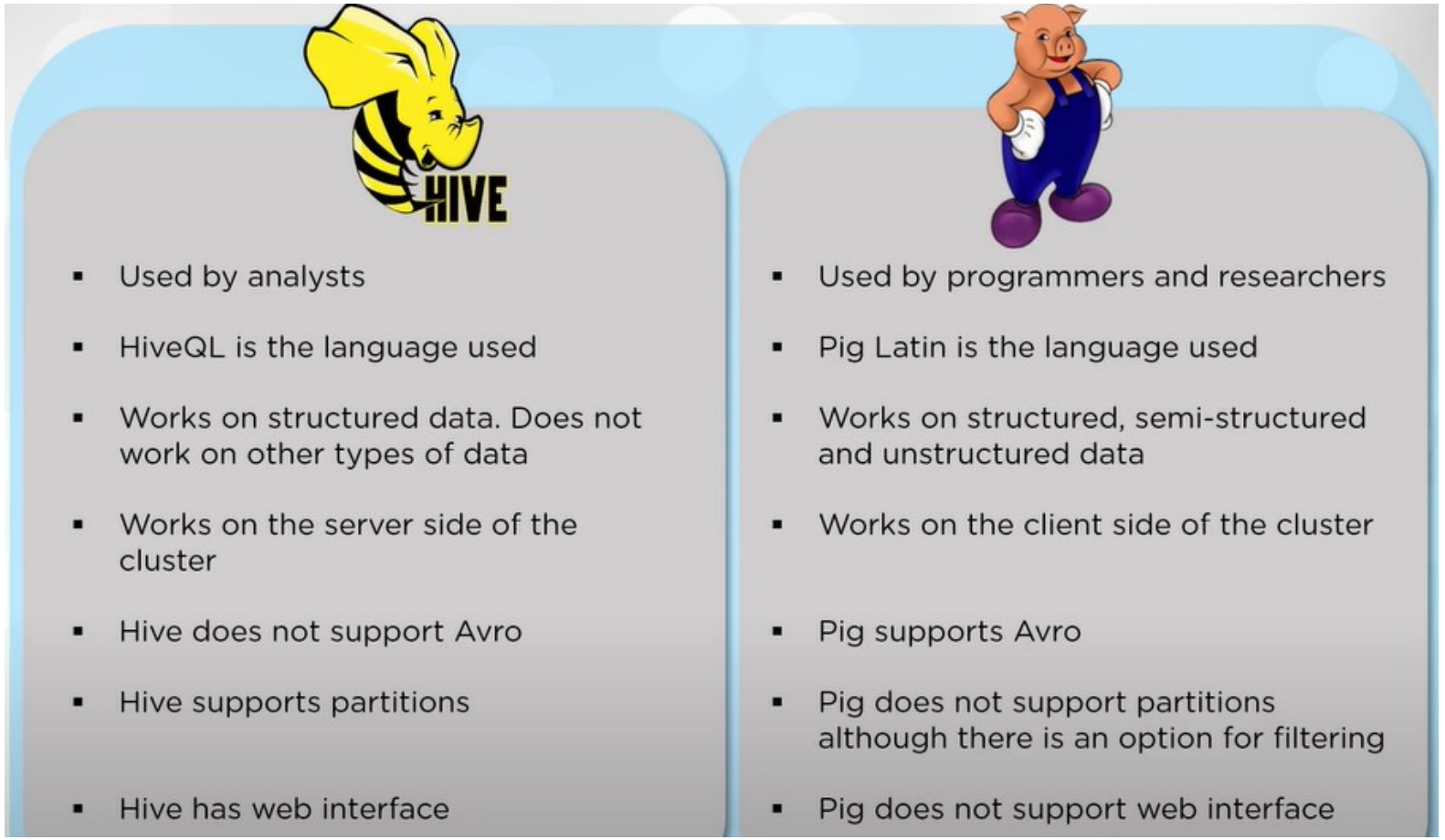
- Need for Hive & Pig
- What is Hive & Pig
- HiveQL & Pig Latin
- Data models
- Execution Modes
- Features
- Commands



Local Mode: Pig engine takes input from the Linux File System and the output is stored in the same file system. It is used when the data is small & when one data node is present.

Map Reduce Mode: Queries written in Pig Latin are translated into MapReduce jobs and are run on a Hadoop cluster. Pig runs in this mode by default. It is used when there are multiple data nodes and the data is large.

- Need for Hive & Pig
- What is Hive & Pig
- HiveQL & Pig Latin
- Data models
- Execution Modes
- Features
- Commands



- Need for Hive & Pig
- What is Hive & Pig
- HiveQL & Pig Latin
- Data models
- Execution Modes
- Features
- Commands

Few Hive Commands

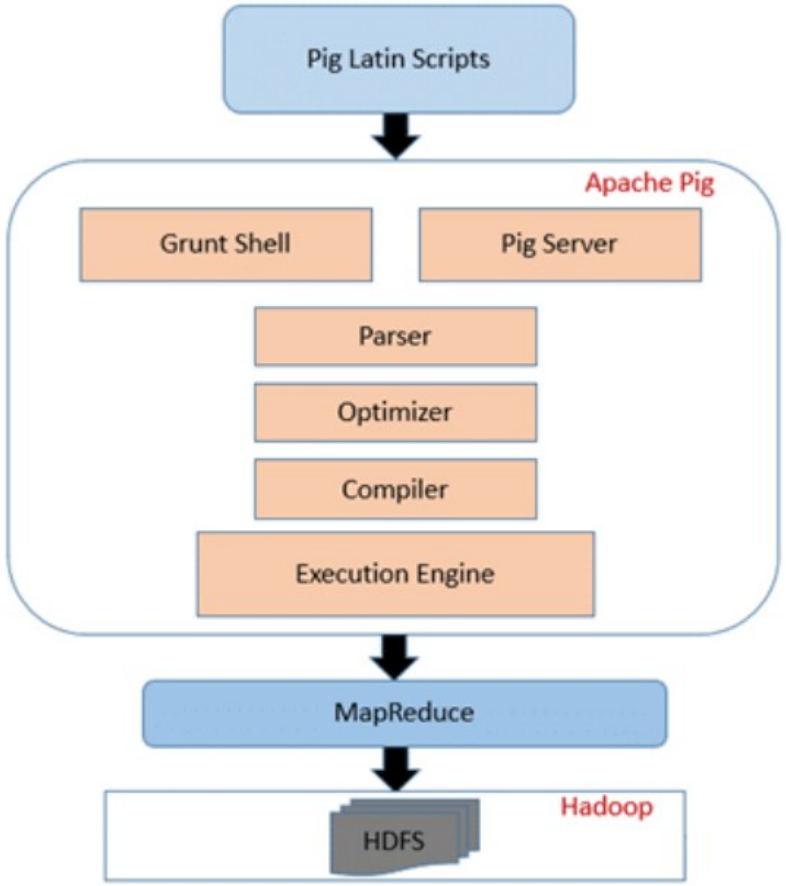
- create database database_name // used to create a new database
- show databases; //shows the list of existing databases
- Now, to create a table inside the database

  create table table_name(ID INT, Name STRING, DEPT STRING, YOJ INT) row format delimited fields terminated by ',';
- show tables; //Gives list of the created table
- hive> SELECT round(2.3) from temp; //Rounds off the value to the nearest highest integer -> 2.3 – 2
- hive> SELECT floor(2.3) from temp; //Rounds off any positive or negative decimal value down to the next least integer value -> 2.3 – 2
- hive> SELECT ceil(2.3) from temp; //This function is used to get the smallest integer which is greater than, or equal to, the specified numeric expression -> 2.3 - 3

- Need for Hive & Pig
- What is Hive & Pig
- HiveQL & Pig Latin
- Data models
- Execution Modes
- Features
- Commands

Few Pig Commands

- hadoop dfs -put 'path_name' /pigInput //For file to be moved into HDFS
- pig // To start the grunt shell mode
- relation1 = LOAD '/pigInput' USING PigStorage(',') AS (Id:chararray,Name:chararray,Profession:chararray,Age:chararray); //Loads the file from HDFS into Pig
- dump relation1; //The results from the previous load command is displayed using dump
- relation1_filter = filter relation1 by column_name == 'attribute_name';
- dump relation1_filter; //Filter command shows the result for that particular filter that we give



Apache Pig Architecture

Apache Pig Architecture is designed to convert **Pig Latin scripts** (a high-level data-flow language) into **MapReduce jobs** that run on Hadoop.

1. Pig Latin Scripts

- The input written by the user (your program).
- Describes *what to do* — not *how to do it*.
- Example:

```
A = LOAD 'data.txt' USING PigStorage(',') AS (id:int, name:chararray);
B = FILTER A BY id > 100;
STORE B INTO 'output';
```

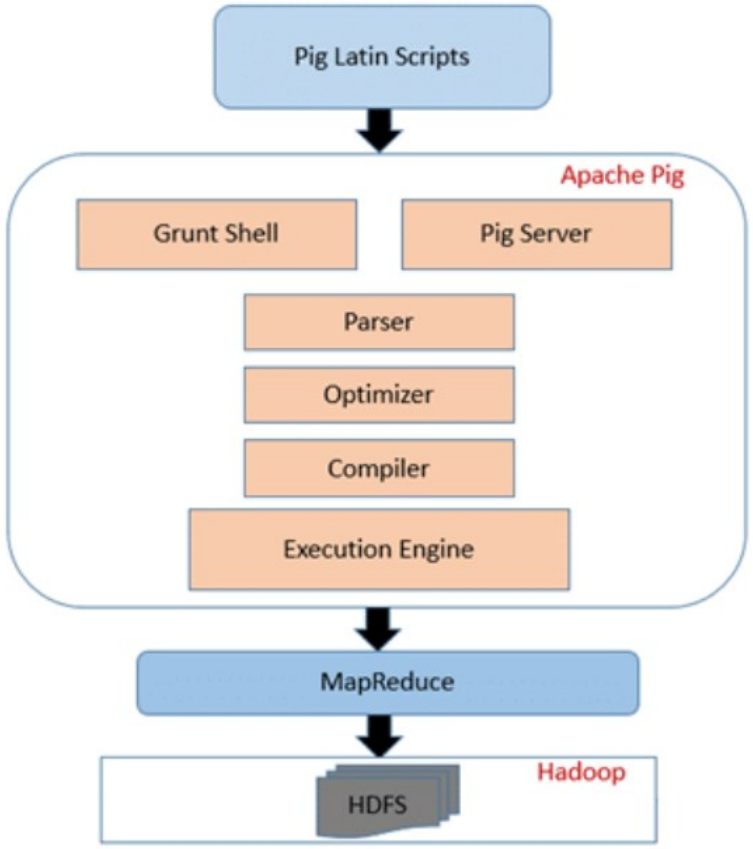
2. Parser

- The first component that processes your script.
- Checks for: Syntax errors, Type errors, Schema validations
- Creates a logical plan — a DAG (Directed Acyclic Graph) of logical operators (like LOAD, FILTER, JOIN, etc.).

Output : Logical Plan

3. Optimizer

- Takes the logical plan and optimizes it for performance.
- Common optimizations include:
  - **Combining multiple** operations (e.g., consecutive FOREACH)
  - **Projection pushdown** (only loading required columns)
  - **Filter pushdown**
  - **Join optimization**
- Optimization is logical, i.e., before any actual MapReduce code is generated.
- Output: Optimized Logical Plan



Apache Pig Architecture

4. Compiler (Logical-to-Physical Translator)

- Converts the optimized logical plan into a **physical plan** —a set of physical operators (e.g., POLoad, POFilter, POJoin) that correspond to MapReduce functions.
- **Output**: Physical Plan

5. MapReduce Compiler

- Converts the physical plan into one or more **MapReduce jobs**.
- Determines how each operator should be executed:
  - Which part should go into **Mapper**
  - Which part into **Reducer**
- Handles data shuffling, sorting, grouping, and partitioning internally.

Output: *Executable MapReduce Jobs (Job DAG)*

6. Execution Engine (Pig Runtime / Hadoop)

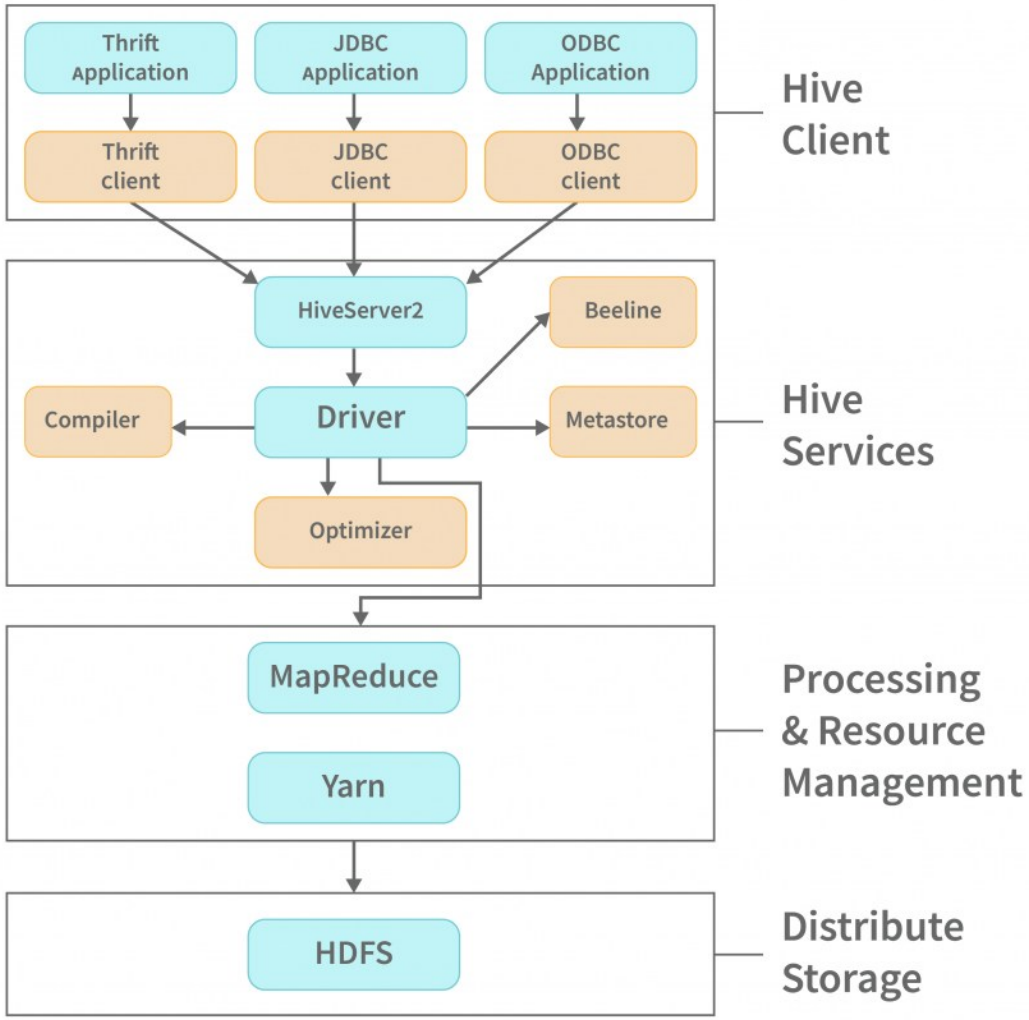
- The **MapReduce engine** (YARN or Hadoop) executes the generated jobs.
- Pig monitors job status, progress, and collects results.
- The intermediate results may be stored temporarily in HDFS between MR stages.

Output: *Final data in HDFS (or local file system)*

7. **Grunt Shell / Pig Server / UDFs**

Component	**Function**
Grunt Shell	Interactive shell to run Pig Latin commands line by line.
Pig Server	Provides programmatic access to run Pig scripts from Java.
UDFs (User-Defined Functions)	Let users extend Pig by writing custom logic in Java/Python.

Hive Architecture & Its Components



https://www.interviewbit.com/blog/hive-architecture/

Hive Client:

Hive drivers support applications written in any language like Python, Java, C++, and Ruby, among others, using JDBC, ODBC, and Thrift drivers, to perform queries on the Hive

The three types of Hive clients are referred to as Hive clients:

- **Thrift Clients**: The Hive server can handle requests from a thrift client by using Apache Thrift.
- **JDBC client**: A JDBC driver connects to Hive using the Thrift framework. Hive Server communicates with the Java applications using the JDBC driver.
- **ODBC client**: The Hive ODBC driver is similar to the JDBC driver in that it uses Thrift to connect to Hive. However, the ODBC driver uses the Hive Server to communicate with it instead of the Hive Server.

Hive Services:

Hive provides numerous services, including the Hive server2, Beeline, etc. The services offered by Hive are:

- **Beeline**: HiveServer2 supports the Beeline, a command shell that the user can submit commands and queries to. It is a JDBC client that utilises SQLLINE CLI (a pure Java console utility for connecting with relational databases and executing SQL queries). The Beeline is based on JDBC.
- **Hive Server 2**: HiveServer2 is the successor to HiveServer1. It provides clients with the ability to execute queries against the Hive. Multiple clients may submit queries to Hive and obtain the desired results. Open API clients such as JDBC and ODBC are supported by HiveServer2.

Hive Driver: The Hive driver receives the HiveQL statements submitted by the user through the command shell and creates session handles for the query.

Hive Compiler: Metastore and hive compiler both store metadata in order to support the semantic analysis and type checking performed on the different query blocks and query expressions by the hive compiler. The execution plan generated by the hive compiler is based on the parse results.
The DAG (Directed Acyclic Graph) is a DAG structure created by the compiler. Each step is a map/reduce job on HDFS, an operation on file metadata, and a data manipulation step.

Optimizer: The optimizer splits the execution plan before performing the transformation operations so that efficiency and scalability are improved.

Execution Engine: After the compilation and optimization steps, the execution engine uses Hadoop to execute the prepared execution plan, which is dependent on the compiler's execution plan.
Metastore: Metastore stores metadata information about tables and partitions, including column and column type information, in order to improve search engine indexing.