# Apache Hive

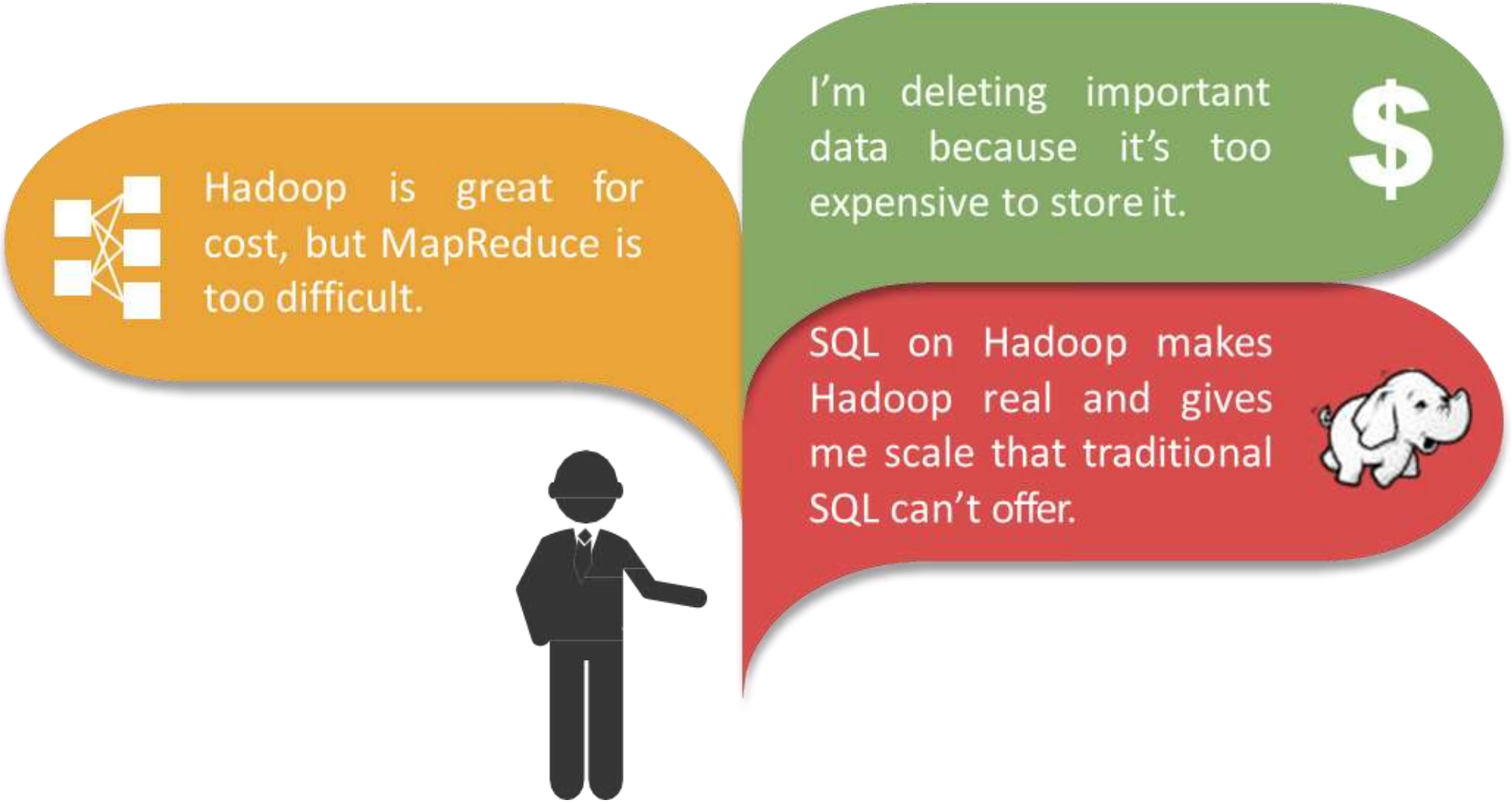



- Hive Introduction
- Hive vs SQL
- Hive Architecture Overview
- Interacting HDFS using HIVE
- Hive Hands on Queries
  - Hive Tables
  - Importing Data
  - Querying Data
  - Joins in Hive

# Introduction and Uses

# Need for Hive





# Introduction to Hive


facebook

- Developed Hive to address traditional RDBMS limitations. 300+ PB of data under management(1).
- 600+ TB of data loaded daily. 60,000+ Hive queries per day(2). More than 1,000 users per day. Initial Apache release in April 2009.

# Hive Vs SQL



| HIVE | SQL |
|---|---|
| Built with an analytical focus in consideration | General purpose database language used for all types of |
| Doesn't support update and delete functions | SQL supports update and delete |
| Faster than SQL in reading & processing huge volumes of data | Slow in processing huge volumes of data |
| Hive has less number of built in functions as compared to SQL | More number of built in functions |

## Hive Classic: Strengths and Challenges

- Familiar SQL Interface
- Economical Processing of Petabytes
- ⛔ Hive Classic tied to MapReduce, leading to latency


# Hive Architecture and Components

# Physical Layout of Hive



- Warehouse directory in HDFS
  - e.g., **/user/hive/warehouse**
- Table row data stored in subdirectories of warehouse
- Partitions form subdirectories of table directories
- Actual data stored in flat files
  - Control char-delimited text, or Sequence Files

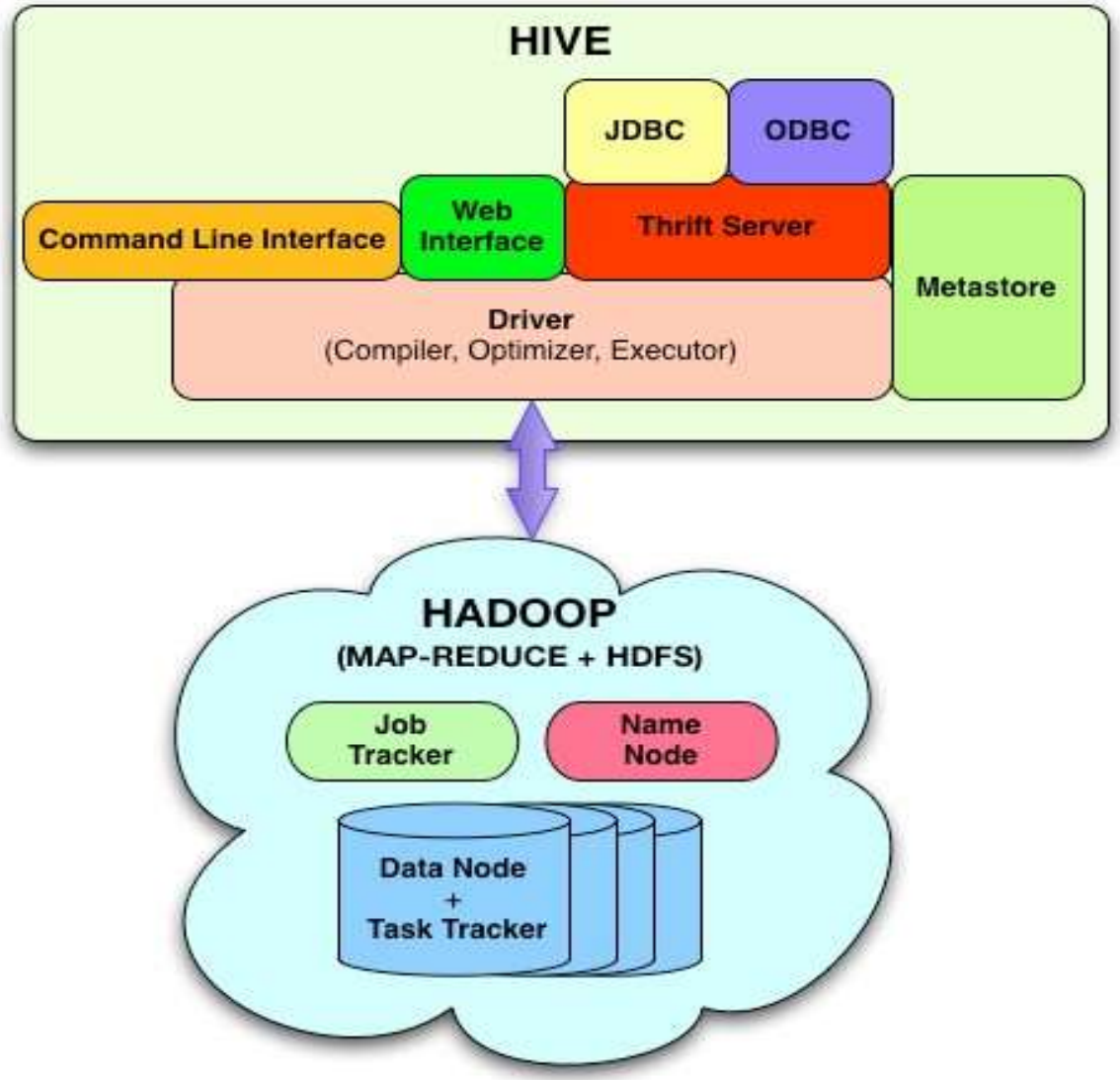
# Flow of Hive




User ask a data in  HIVE ( Kind of SQL)

# Hive Architecture





# Hive Architecture



# Hive Architecture




**Hive Client** supports different types of client applications in different languages for performing queries

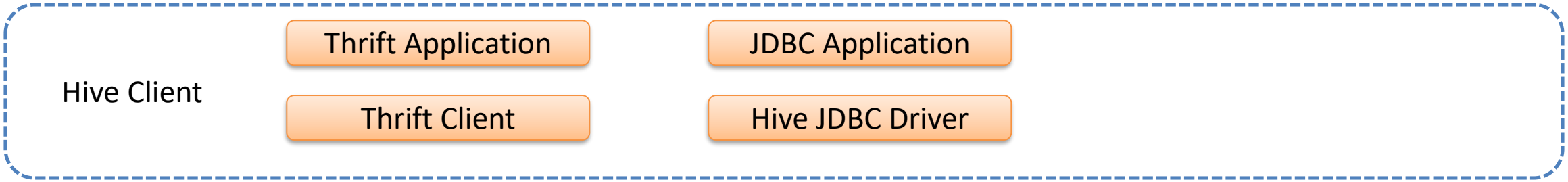
# Hive Architecture




**Thrift** is a software framework. Hive Server is based on Thrift, so it can support the request from all programing languages that support thrift

# Hive Architecture

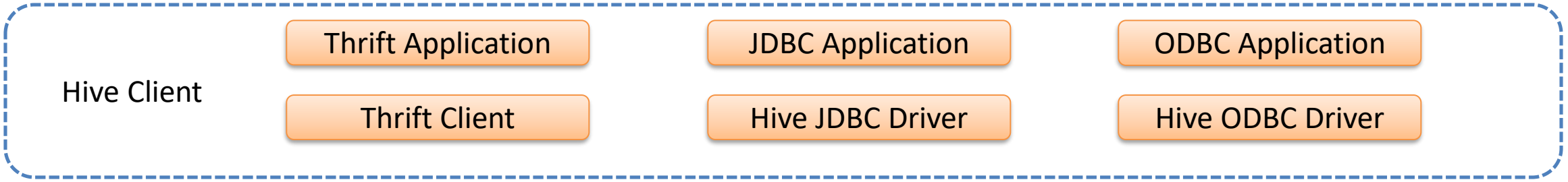





**JDBC –** Java Database Connectivity, helps to connect JDBC applications

# Hive Architecture

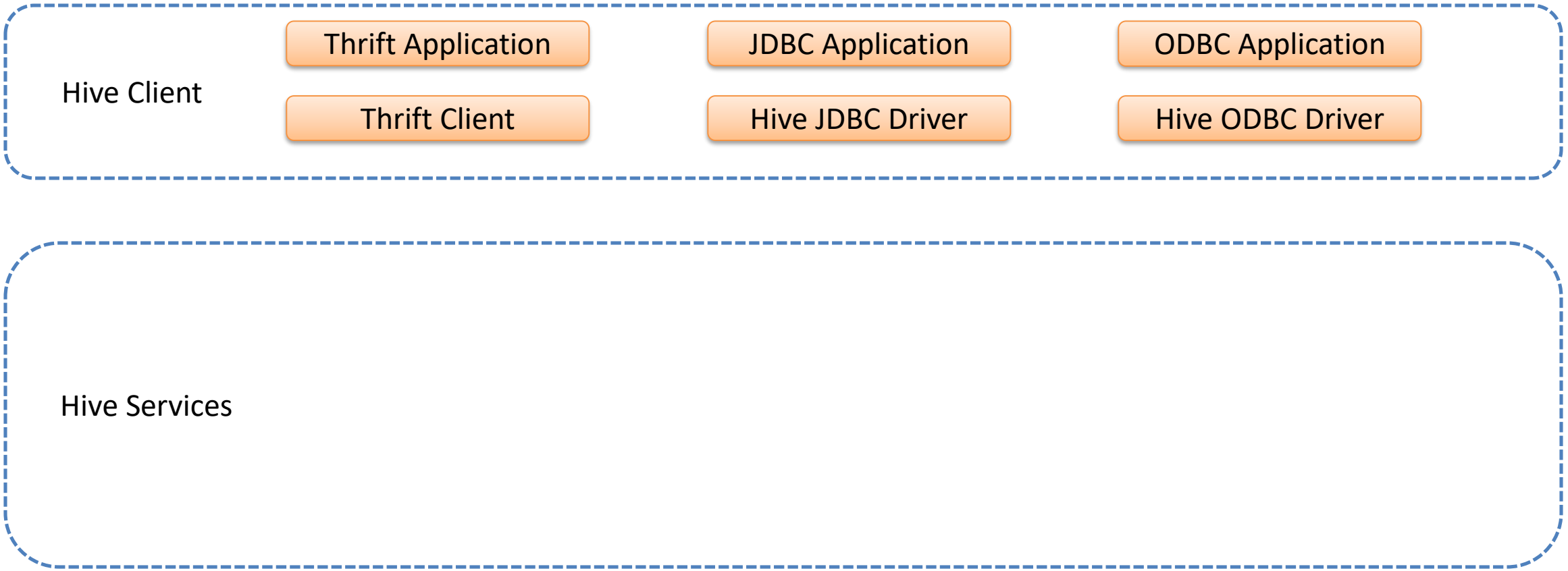





**ODBC–** Open Database Connectivity, helps to connect ODBC applications

# Hive Architecture

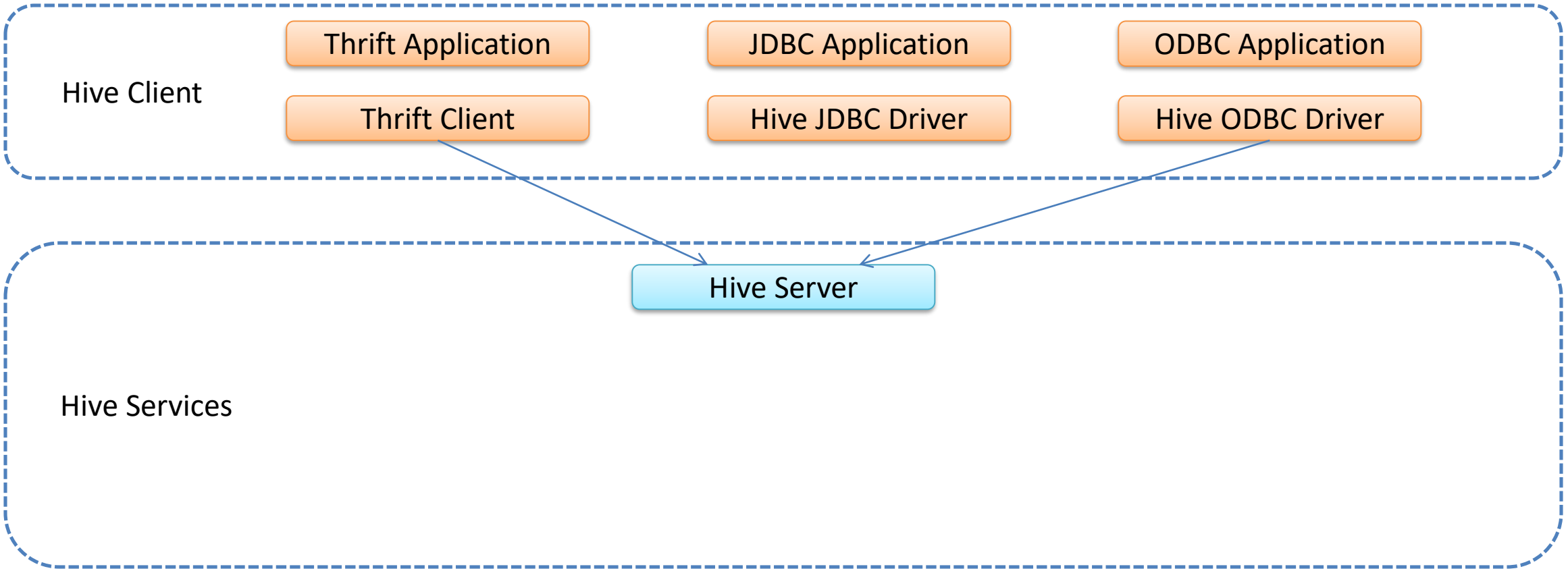





**Hive Supports various services**

# Hive Architecture

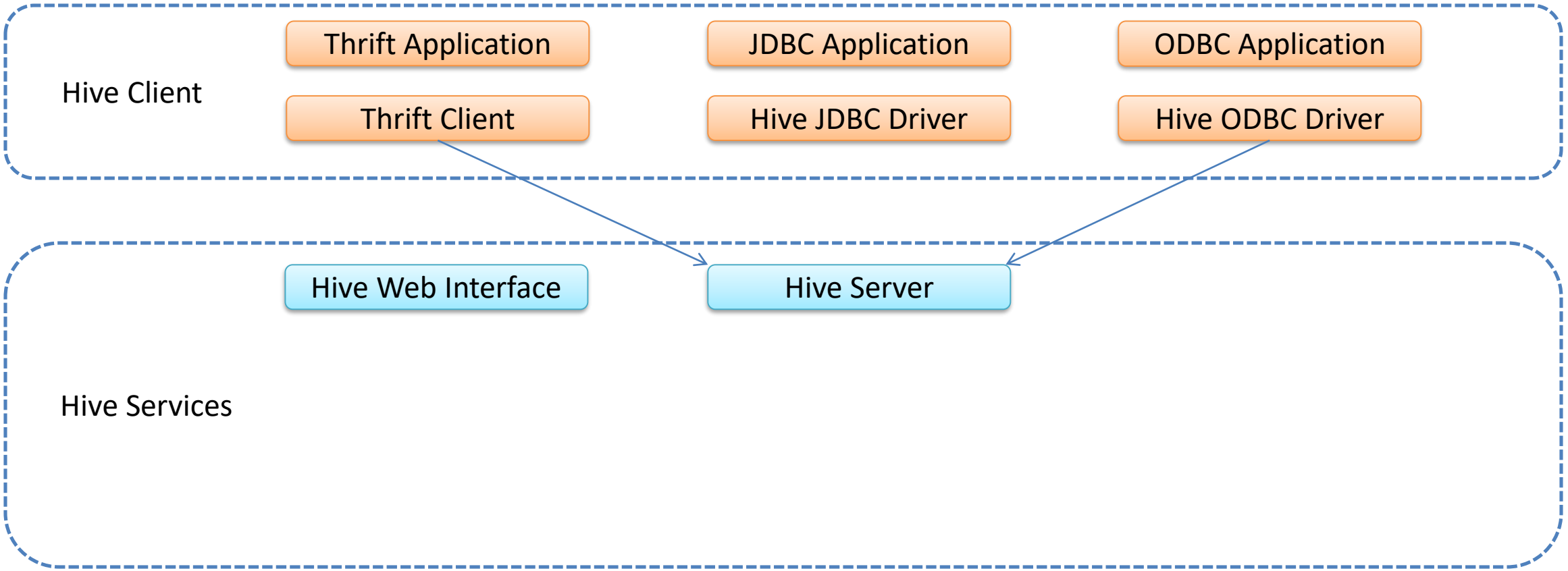





All the client requests are submitted to **Hive Server**

# Hive Architecture

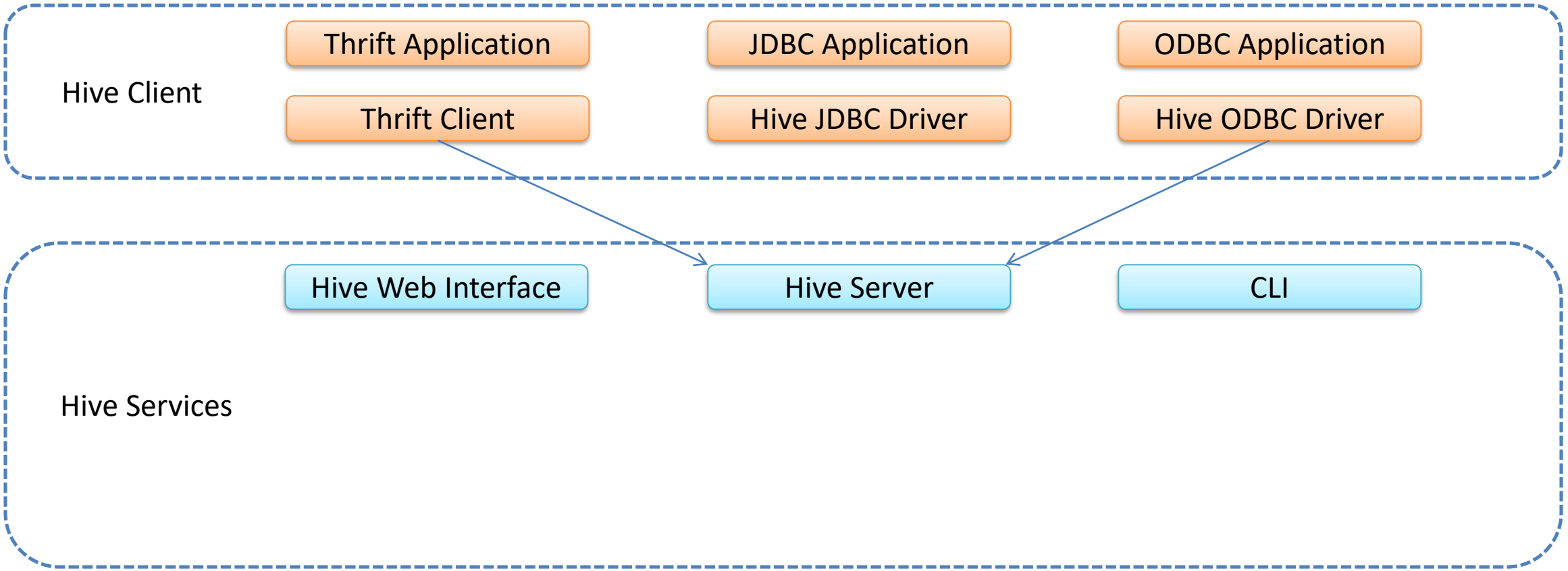





Web interface or GUI is provided to execute Quries

# Hive Architecture

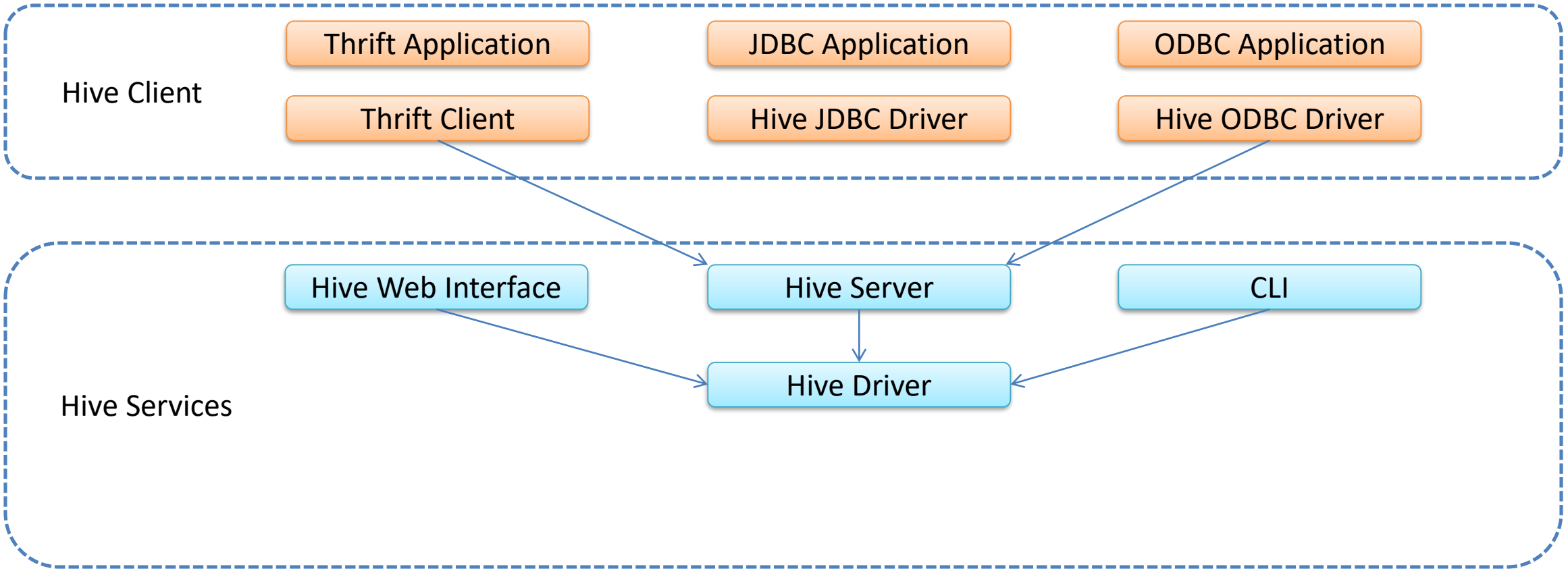





Commands are executed directly in CLI

# Hive Architecture

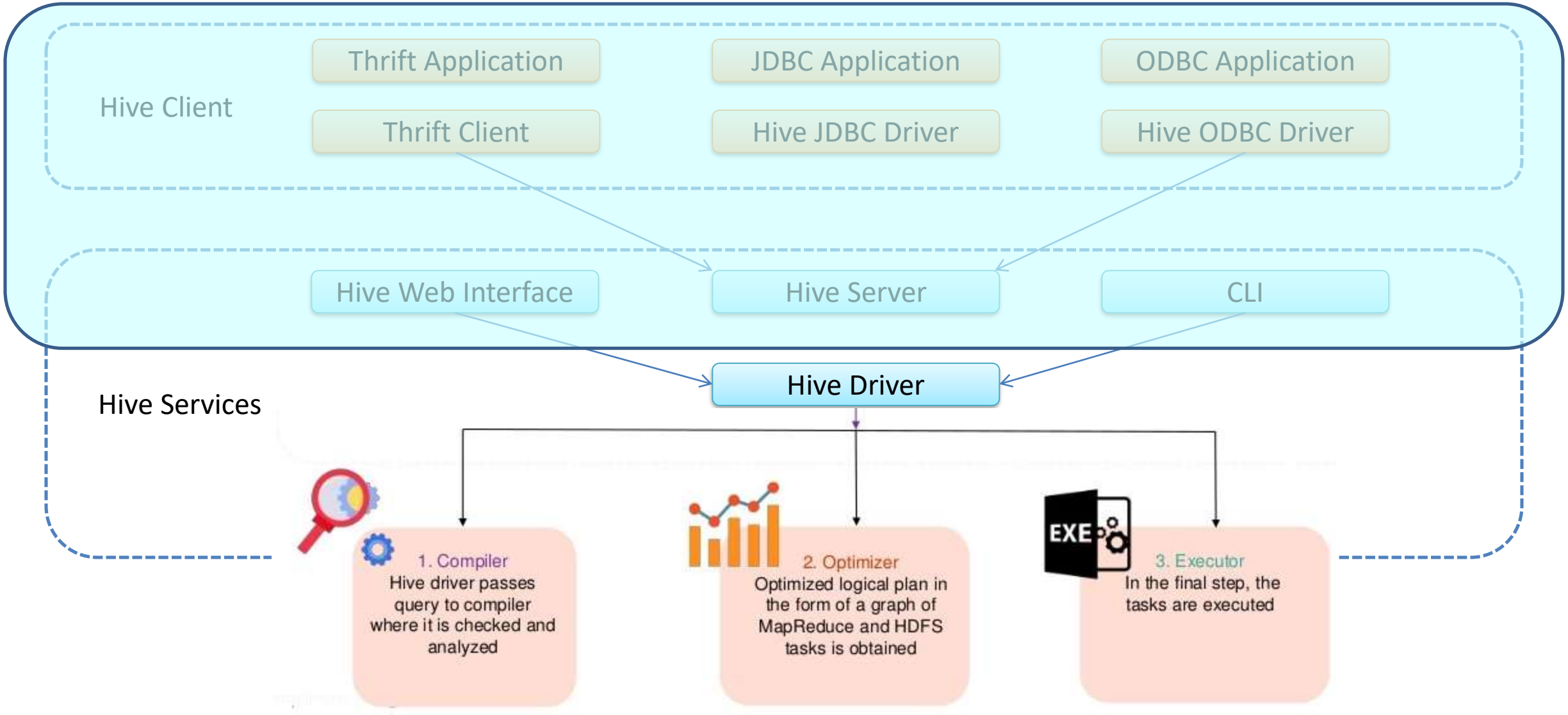





Hive Driver is responsible for all the queries submitted

# Hive Architecture

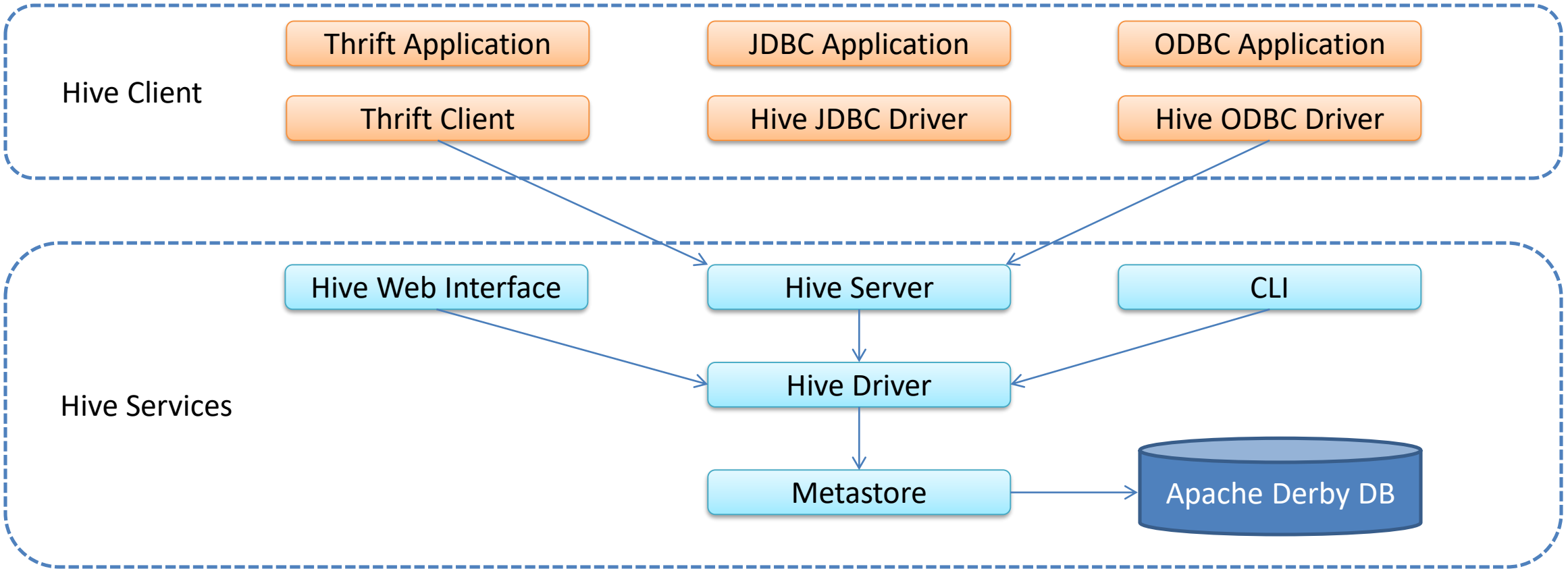





Hive Driver Performs 3 steps internally

# Hive Architecture

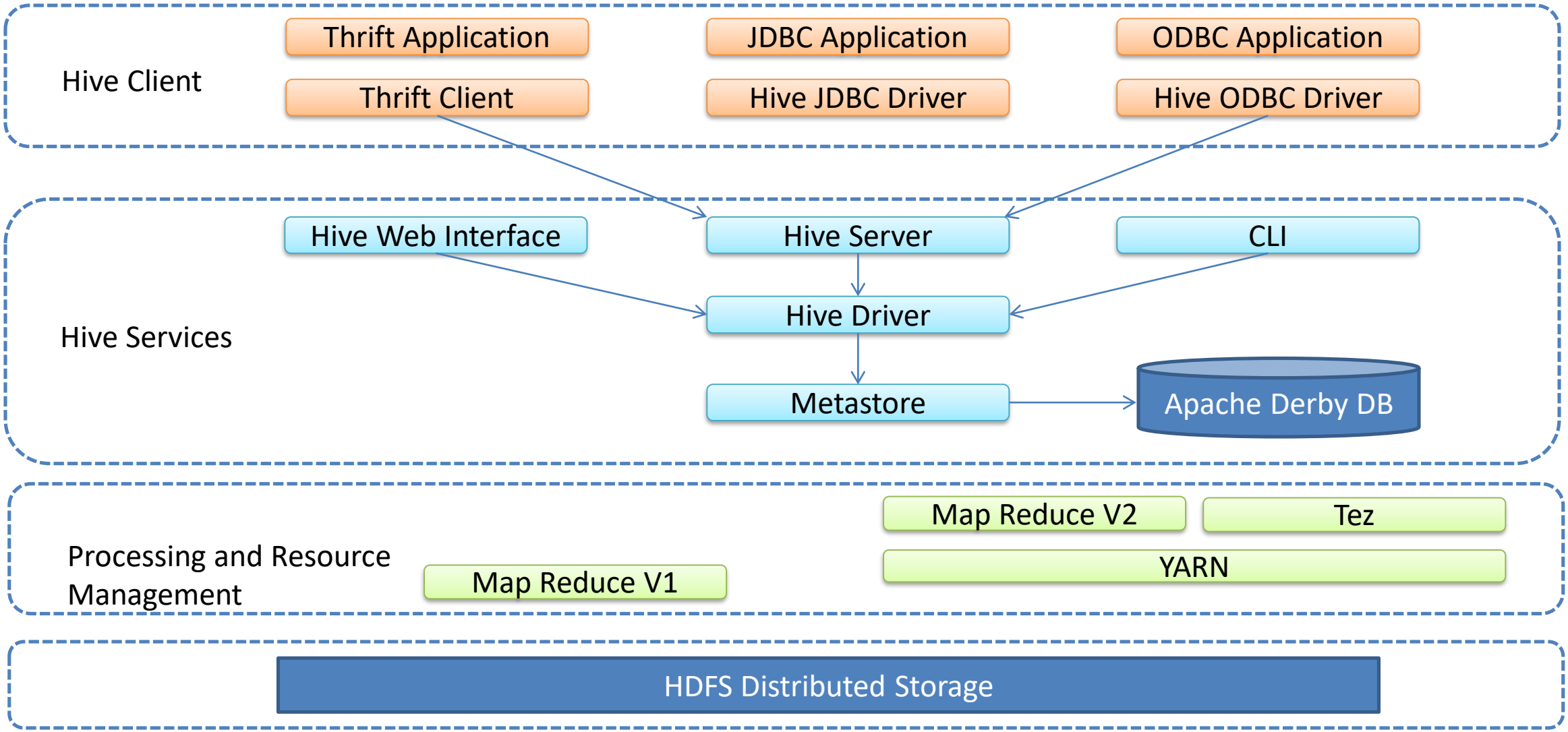





**Metastore** is repository for Hive Metadata. Stores metadata for Hive Tables

# Hive Architecture



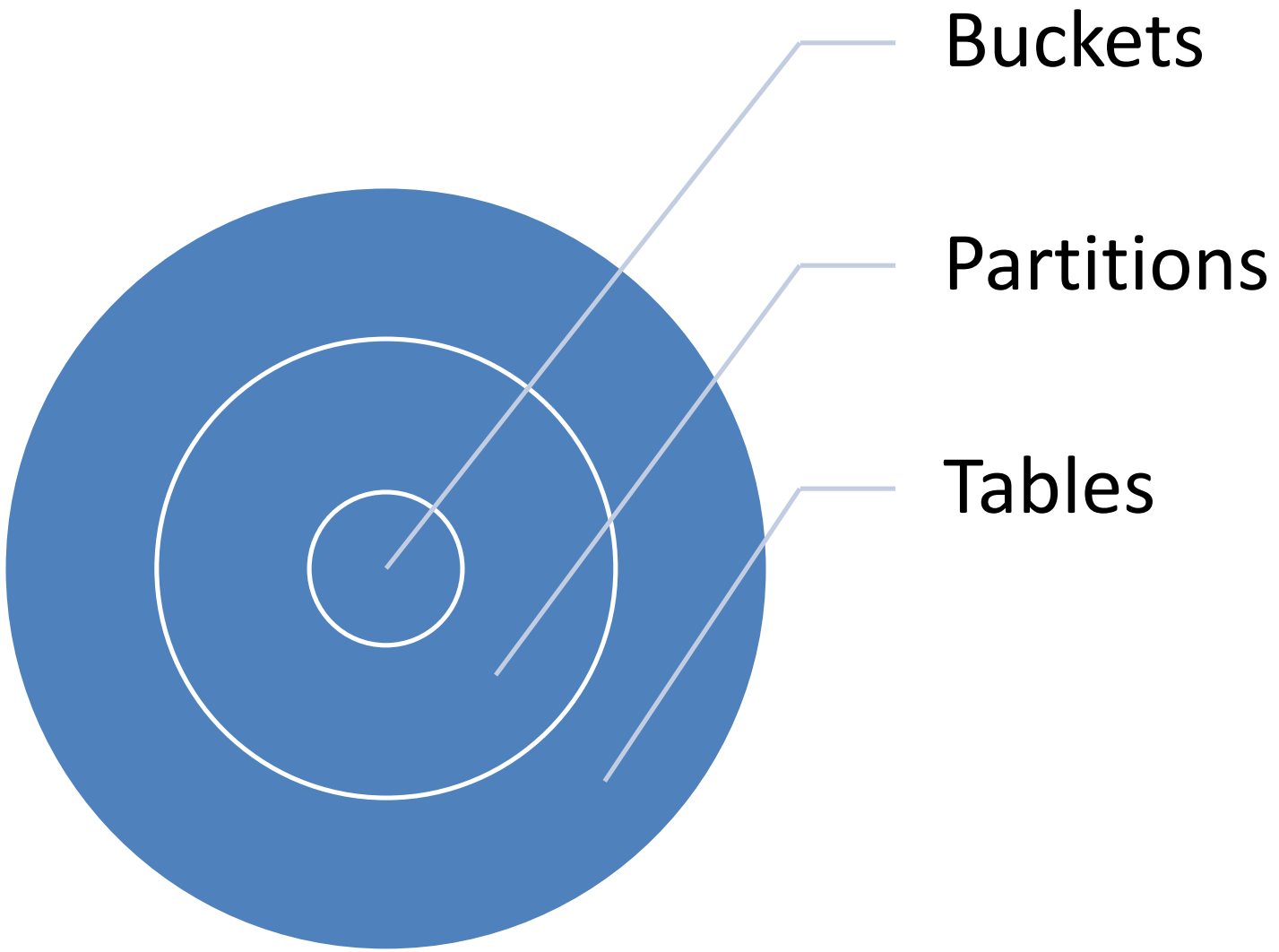



**We will not go in details of Processing and Storage Steps**

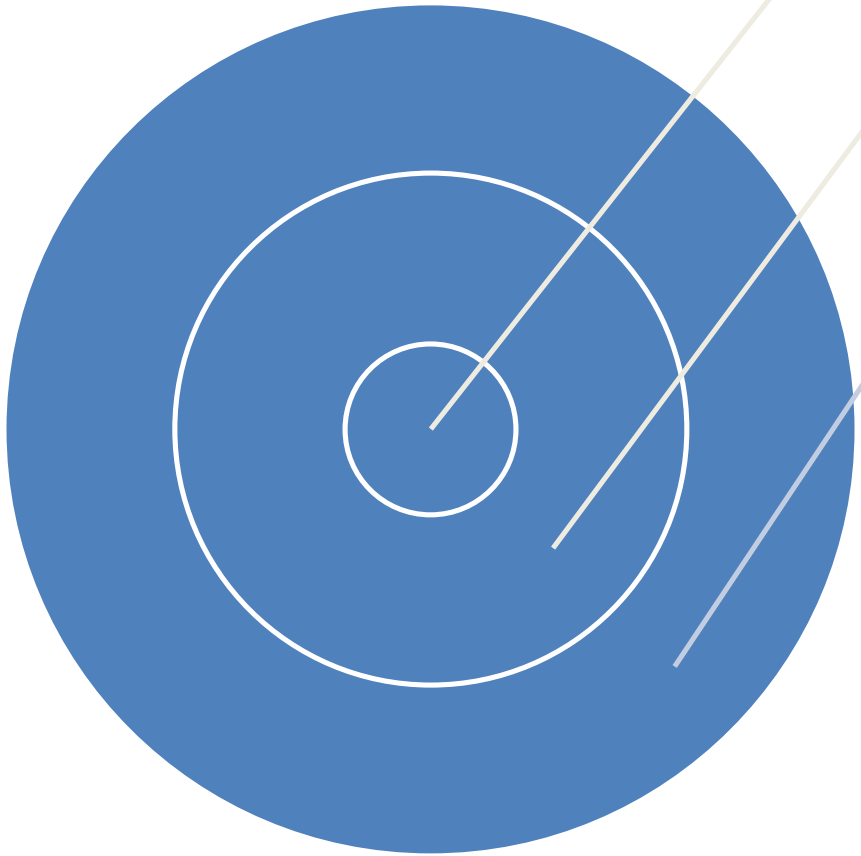
# Hive Data Modeling

# Hive Data Modeling





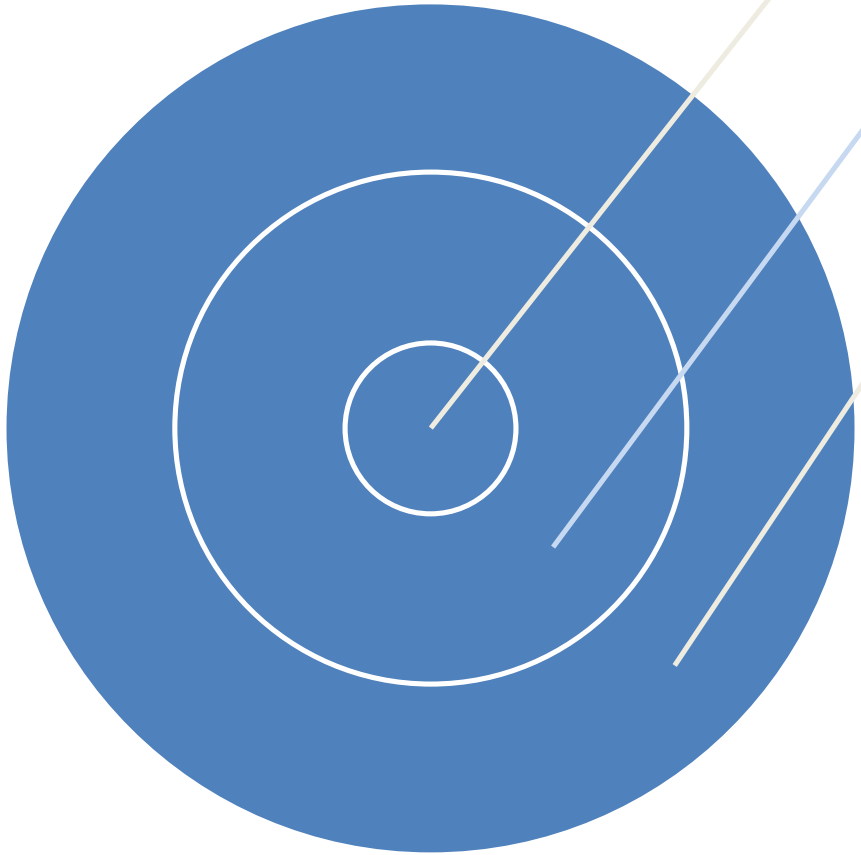
# Hive Data Modeling




Tables in Hive are created in the same way it is done in RDBMS

# Hive Data Modeling

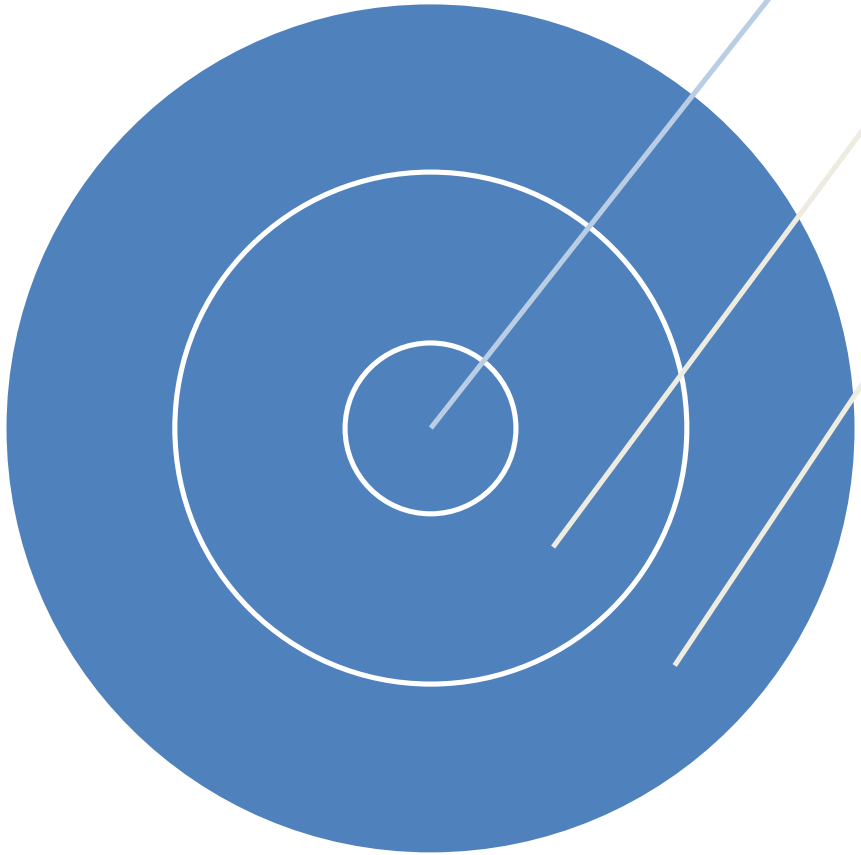





Here tables are organized into Partitions for grouping same kind of data based on Partition Key

# Hive Data Modeling





Buckets

Partitions

Tables

Data Present in Partition can be further divided into Buckets for efficient Queries


# Beginning of HQL

# Create Database in Hive



First Make sure we have a database created, if not then follow below steps

Then check there is database Created

Following are the commands to create database in Hive

```
hive> CREATE DATABASE [IF NOT EXISTS] userdb;
```

```
hive> SHOW DATABASES;

Output is as below:
      default
      userdb
```

# Create Table in Hive

proschool
An ims Initiative

Create table of following data type in Hive

```
Hive>CREATE TABLE IF NOT EXISTS employee ( eid int,
  name String, salary String, destination String)
COMMENT ‘Employee details’
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘\t’
LINES TERMINATED BY ‘\n’
STORED AS TEXTFILE;
```

| Field Name | Data Type |
|---|---|
| Eid | int |
| Name | String |
| Salary | Float |
| Designation | string |

Load Data Statement

Generally, after creating a table in SQL, we can insert data using the Insert statement. But in Hive, we can insert data using the LOAD DATA statement.

While inserting data into Hive, it is better to use LOAD DATA to store bulk records. There are two ways to load data: one is from local file system and second is from Hadoop file system.

# Managed Table Vs External Table



Managed Table

- When we create a table in Hive by default it is a managed table
- The table is managed by hive
- When a data is loaded in a managed table its moved to warehouse directory
- The table data will be deleted when table is dropped

External Table

- Managed by user
- Referring the data outside warehouse
- Applies only schema to the external file
- When table is dropped only the metadata/schema is deleted the file being referred is not deleted

# Load Data in Table



- Generally, after creating a table in SQL, we can insert data using the Insert statement.
- But in Hive, we can insert data using the LOAD DATA statement.
- While inserting data into Hive, it is better to use LOAD DATA to store bulk records.
- There are two ways to load data
  - one is from local file system
  - second is from Hadoop file system.

We are assuming the file is in local folder of following path

```
Hive>LOAD DATA LOCAL INPATH '/home/user/sample.txt'
          OVERWRITE INTO TABLE employee;
```

# Adding Partition

proschool
An ims Initiative

- Consider file Contains data

```
id, name, dept, yoj
1, gopal, TP, 2012
2, kiran, HR, 2012
3, kaleel,SC, 2013
4, Prasanth, SC, 2013
```

- Consider file is divided in the 2 files by year of joining

/tab1/employeedata/2012/file2

```
1, gopal, TP, 2012
2, kiran, HR, 2012
```

/tab1/employeedata/2013/file3

```
3, kaleel,SC, 2013
4, Prasanth, SC, 2013
```

- Following command is used to add a partition, give a try to rename also

```
hive> ALTER TABLE employee > ADD PARTITION
(year=’2012’) > location '/2012/part2012';
```


# Partitioning and Bucketing

# Partition



- What is Partitions?
  - ✓ A way of dividing a table into coarse-grained parts based on the value of a partition column, such as a date.
  - ✓ Using partitions can make it faster to do queries on slices of the data.
- A Table may be partitioned in multiple dimensions
  - ✓ it by location so as to get to know which location has what all profiles.
- Partitions are defined at table creation time using the PARTITIONED BY clause, which takes a list of column definitions

# Partitions



- At the filesystem level ,partitions are nested subdirectories of the table directories.
- For example we want to partition data of sales of an company by date and city :
  The files created after using **PARTITIONED BY(**dat STRING,city STRING) are :

```
|--dat = 2016-12-12/
| |--city=Pune/
| | |--file1
| | |-file2….
```

```
hive>SHOW PARTITIONS sales ;
dat=2016-12-12/city=Pune
dat=2016-12-12/city=Mumbai
dat=2016-12-13/city=Pune
dat=2016-12-13/city=Mumbai
```

# Partitions Contd.



- **Partition columns :** Columns defined in the PARTITIONED BY clause are full fledged table columns known as Partition columns
- Datafiles do not contain values for these columns ,since they are derived from directory names.

✓ Partition columns can be used in SELECT statements in usual way .

✓ Hive performs input pruning to scan only the relevant partitions. For eg : SELECT dat, company_name FROM sales

WHERE city = 'Pune'

✓ Only scans the files containing city = Pune

✓ The query returns the values of the dat partition columns ,which Hive reads from the directory names since they are not in datafiles

# Buckets



- Bucketing is a technique for decomposing data sets into more manageable parts
- It imposes extra structure on the table.
- For example :

```
hive> CREATE TABLE weblog (url STRING, source_ip STRING)
> PARTITIONED BY (dt STRING, user_id INT);
hive> FROM raw_weblog
> INSERT OVERWRITE TABLE page_view PARTITION(dt='2016-06-08', user_id)
> SELECT server_name, url, source_ip, dt, user_id;
```

- If use dynamic partitioning, above commands might fail because by default Hive limits the maximum number of dynamic partitions that may be created to prevent the extreme case that overwhelm the filesystem's ability.

# Buckets Contd.



- Instead, if we bucket the weblog table and use user_id as the bucketing column, the value of this column will be hashed by a user-defined number into buckets.

```
hive > CREATE TABLE weblog (user_id INT, url STRING, source_ip
STRING)
    > PARTITIONED BY (dt STRING)
    > CLUSTERED BY (user_id) INTO 96 BUCKETS;
```

- Records with the same user_id will always be stored in the same bucket, each bucket will have many users

# Buckets



- Creating Bucketed Table :

```
CREATE TABLE bucketed_users (emp_id INT, name STRING)
CLUSTERED BY (id) INTO 8 BUCKETS;
```

- Inserting Data :

```
INSERT OVERWRITE TABLE
bucketed_users  SELECT * FROM
users;
```

- To populate the bucketed table, we need to set the hive.enforce.bucketing property to true

# Bucket : working



- Hive bucketing works by hashing the value and reducing modulo number of buckets
- EG : hive>**SELECT * FROM STUDENTS**

  1 Ram
  2 Rajiv
  3 Rohit
  4 Nikhil

  Each bucket is a file in the table or partition directory  Hive > dfs –ls /user/hive/training_hive/bucketed_users  Output :
  00000_0
  00000_1
  00000_2
  ..

# Hive Partition : Features



- Hive Partitioning refers to dividing large amount of data into number pieces of folders based on table columns value.
- Hive Partition is often used for distributing load horizontally, this has performance benefit, and helps in organizing data in a logical fashion.
- We can perform partition on any number of columns in a table by using hive partition concept.
- Partitioning works better when the cardinality of the partitioning field is not too high .
- Partition is not solving responsiveness problem in case of data skewing towards a particular partition value.
  - ✓ Grouping of customers of Walmart will take long time compared to grouping of customers of Big Bazaar.

# Hive Bucketing : Features



- Hive bucketing is responsible for dividing the data into number of equal parts
- We can perform Hive bucketing optimization only on one column only not more than one.
- The value of this column will be hashed by a user-defined number into buckets.
- Bucketing works well when the field has high cardinality and data is evenly distributed among buckets
- Bucketing on the other hand, will result with a fixed number of files, since you do specify the number of buckets.
    - ✓ But, if you use let's assume 512 buckets and the field you're bucketing on has a low cardinality (for instance, it's a Australia state, so can be only 100 different values). Then we will have 100 buckets with data, and 412 buckets with no data.
- Due to equal volumes of data in each partition, joins at Map side will be quicker.

# Querying Bucket



- When querying a partition table when a query is issued only the required partitions are scanned.
- For Bucketed tables ,we need to provide an hint to the query using TABLESAMPLE clause ,if we want to scan few particular buckets else the whole set of files would be scanned.
- For example :

    If we want to choose data from only 5 buckets out of 50 buckets then we use

    -- SELECT * FROM table1 TABLESAMPLE (5 out of 50) WHERE dat = ‘2016-12-11’

# Comparing Queries with SQL and Hive

proschool An ims Initiative

| Function | MySQL | Hive QL |
|---|---|---|
| Retrieving information | SELECT from_columns FROM table WHERE conditions; | SELECT from_columns FROM table WHERE conditions; |
| All values | SELECT * FROM table; | SELECT * FROM table; |
| Some values | SELECT * FROM table WHERE rec_name = “value”; | SELECT * FROM table WHERE rec_name = "value"; |
| Multiple criteria | SELECT * FROM table WHERE rec1=”value1” AND rec2=”value2”; | SELECT * FROM TABLE WHERE rec1 = "value1" AND rec2 = "value2"; |
| Selecting specific columns | SELECT column_name FROM table; | SELECT column_name FROM table; |
| Retrieving unique output records | SELECT DISTINCT column_name FROM table; | SELECT DISTINCT column_name FROM table; |
| Sorting | SELECT col1, col2 FROM table ORDER BY col2; | SELECT col1, col2 FROM table ORDER BY col2; |
| Sorting backward | SELECT col1, col2 FROM table ORDER BY col2 DESC; | SELECT col1, col2 FROM table ORDER BY col2 DESC; |
| Counting rows | SELECT COUNT(*) FROM table; | SELECT COUNT(*) FROM table; |
| Grouping with counting | SELECT owner, COUNT(*) FROM table GROUP BY owner; | SELECT owner, COUNT(*) FROM table GROUP BY owner; |
| Maximum value | SELECT MAX(col_name) AS label FROM table; | SELECT MAX(col_name) AS label FROM table; |

# Joins in Hive

proschool
An ims Initiative

Following table named CUSTOMERS

| ID | NAME | AGE | ADDRESS | SALARY |
|---|---|---|---|---|
| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
| 2 | Khilan | 25 | Delhi | 1500.00 |
| 3 | kaushik | 23 | Kota | 2000.00 |
| 4 | Chaitali | 25 | Mumbai | 6500.00 |
| 5 | Hardik | 27 | Bhopal | 8500.00 |
| 6 | Komal | 22 | MP | 4500.00 |
| 7 | Muffy | 24 | Indore | 10000.00 |

Table ORDERS as follows

| OID | DATE | CUSTOMER_ID | AMOUNT |
|---|---|---|---|
| 102 | 2009-10-08 00:00:00 | 3 | 3000 |
| 100 | 2009-10-08 00:00:00 | 3 | 1500 |
| 101 | 2009-11-20 00:00:00 | 2 | 1560 |
| 103 | 2008-05-20 00:00:00 | 4 | 2060 |

Default Join is outer Join and query as follows

```
hive> SELECT c.ID, c.NAME, c.AGE, o.AMOUNT FROM
CUSTOMERS c JOIN ORDERS o ON (c.ID = o.CUSTOMER_ID);
```

# Joins in Hive



Default Join is outer Join and query as follows

```
hive> SELECT c.ID, c.NAME, c.AGE, o.AMOUNT FROM
CUSTOMERS c JOIN ORDERS o ON (c.ID = o.CUSTOMER_ID);
```

Output on successful completion is

```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
| 3  | kaushik  | 23  | 3000   |
| 3  | kaushik  | 23  | 1500   |
| 2  | Khilan   | 25  | 1560   |
| 4  | Chaitali | 25  | 2060   |
+----+----------+-----+--------+
```

# Joins in Hive



Left outer Join in Hive as follows

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE FROM
CUSTOMERS c LEFT OUTER JOIN ORDERS o ON (c.ID =
o.CUSTOMER_ID);
```

Output on successful completion is

```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
| 1  | Ramesh   | NULL   | NULL                |
| 2  | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 3  | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3  | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 4  | Chaitali | 2060   | 2008-05-20 00:00:00 |
| 5  | Hardik   | NULL   | NULL                |
| 6  | Komal    | NULL   | NULL                |
| 7  | Muffy    | NULL   | NULL                |
+----+----------+--------+---------------------+
```

# Joins in Hive



Right outer Join in Hive as follows

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE FROM
CUSTOMERS c RIGHT OUTER JOIN ORDERS o ON (c.ID =
o.CUSTOMER_ID);
```

Output on successful completion is

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
| 3    | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3    | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 2    | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 4    | Chaitali | 2060   | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

Please also explore the full outer join on your own

# Introduction to NOSQL




Neo4j

Google BigTable

mongoDB

CouchDB

amazon DynamoDB

# Introduction to NOSQL



- NOSQL – **N**ot **O**nly **SQL** (Structured Query Language)
- Non relational data storage system
- They do not require tabular relations used in relational databases
- Used in big data and real-time web applications
- They may support SQL like language
- *Horizontal scaling* of clusters of machines is achievable using NoSQL
- Data structures used in NoSQL databases are different from those in relational databases  making some operations faster in NoSQL

# HBase



- HBase began as a project by the company Powerset in order to process
- massive amounts of data for the purpose of natural language search.
- Facebook selected HBase to implement its new messaging platform in 2010
- The first HBase release was bundled as part of Hadoop 0.15.0 in October 2007.
- HBase is basically a database ,it’s a Hadoop database .

# Apache HBase: Introduction

proschool
An ims Initiative

- HBase is often described as a sparse, distributed, persistent, multidimensional sorted map, which is indexed by rowkey, columnkey, and timestamp.
- It's a platform for storing and retrieving data with random access .
- HBase stores structured and semi structured data naturally so you can load it with tweets and parsed log files and a catalogue of all .
- HBase can store unstructured data too, as long as it's not too large. It doesn't care about types and allows for a dynamic and flexible data model that doesn't constrain the kind of data you store.
- It is not a relational database like RDBMS ,hence doesn't include SQL like syntax
- It is designed to run on cluster of computers , it scales horizontally with the addition of new machines.

# Hbase VS HDFS

proschool
An ims Initiative

| HDFS | HBase |
| --- | --- |
| It is a distributed file storage system | It is a database built on top of HDFS |
| Doesn’t support fast individual record lookups | Provides fast record lookups |
| Sequential access of data | Uses Hash tables and provide random aceess of data |
| Provides high latency batch processing | Provides low latency access to single rows |

# Apache HBase

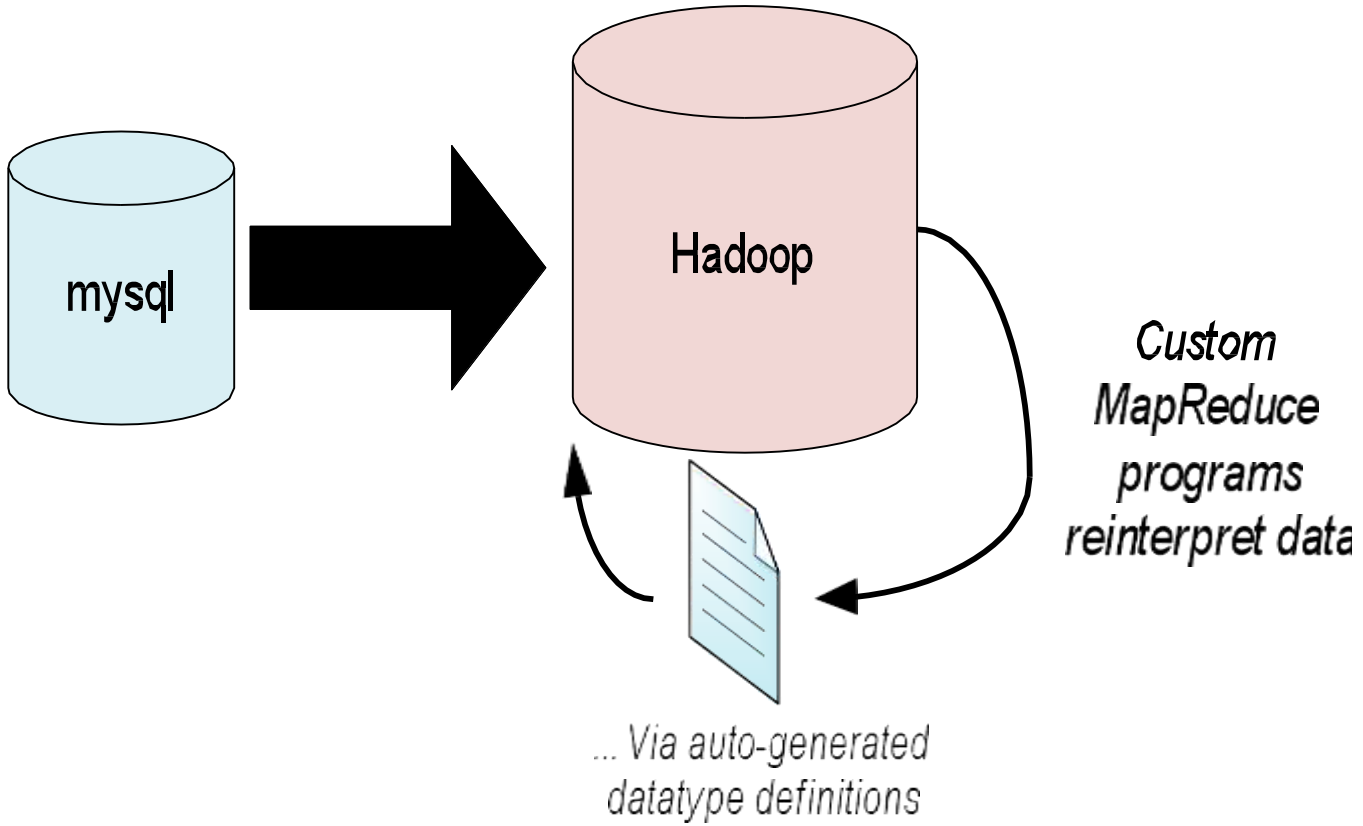


| HBase | RDBMS |
| --- | --- |
| It is schema-less ,doesn't have the concept of fixed columns schema. | RDBMS has well defined schema which describes the whole structure of tables |
| No transactions are there in HBase | RDBMS is transactional |
| It is built for wide tables and is horizontally scalable | It is built for small tables and is hard to scale |
| It is good for semi-structured as well as structured data | It is good for structured data |

# Introduction to Sqoop



## Sqoop = SQL-to-Hadoop



# Sqoop Features



- JDBC-based interface (MySQL, Oracle, PostgreSQL,  etc…)
- Automatic Datatype generation
  - Reads column info from table and generates Java classes
  - Can be used in further MapReduce processing passes
- Uses MapReduce to read tables from database
  - Can select individual table (or subset of columns)
  - Can read all tables in database
- Supports most JDBC standard types and null values

# Example of Sqoop

proschool
An ims Initiative

```
mysql> use corp;
Database changed

mysql> describe employees;
```

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| id | int(11) | NO | PRI | NULL | auto_increment |
| firstname | varchar(32) | YES | | NULL | |
| lastname | varchar(32) | YES | | NULL | |
| jobtitle | varchar(64) | YES | | NULL | |
| start_date | date | YES | | NULL | |
| dept_id | int(11) | YES | | NULL | |

# Import table in HDFS



```
sqoop --connect
jdbc:mysql://db.foo.com/corp \
      --table employees
```

- Imports “employees” table into HDFS directory

# Sqoop Hive Integration



```
$ sqoop --connect jdbc:mysql://db.foo.com/  corp      --hive-import --
table employees
```

- Auto-generates  CREATE TABLE / LOAD/ INPATH statements for Hive
- After data is imported to HDFS, auto-executes  Hive script
- Follow-up step: Loading into partitions

# Appendix



- Programming Hive : O'Reilly
- https://hive.apache.org/
- http://hortonworks.com/apache/hive/
- https://cwiki.apache.org/confluence/display/Hive/Home
- http://www.cloudera.com/products/apache-hadoop/apache-hive.html