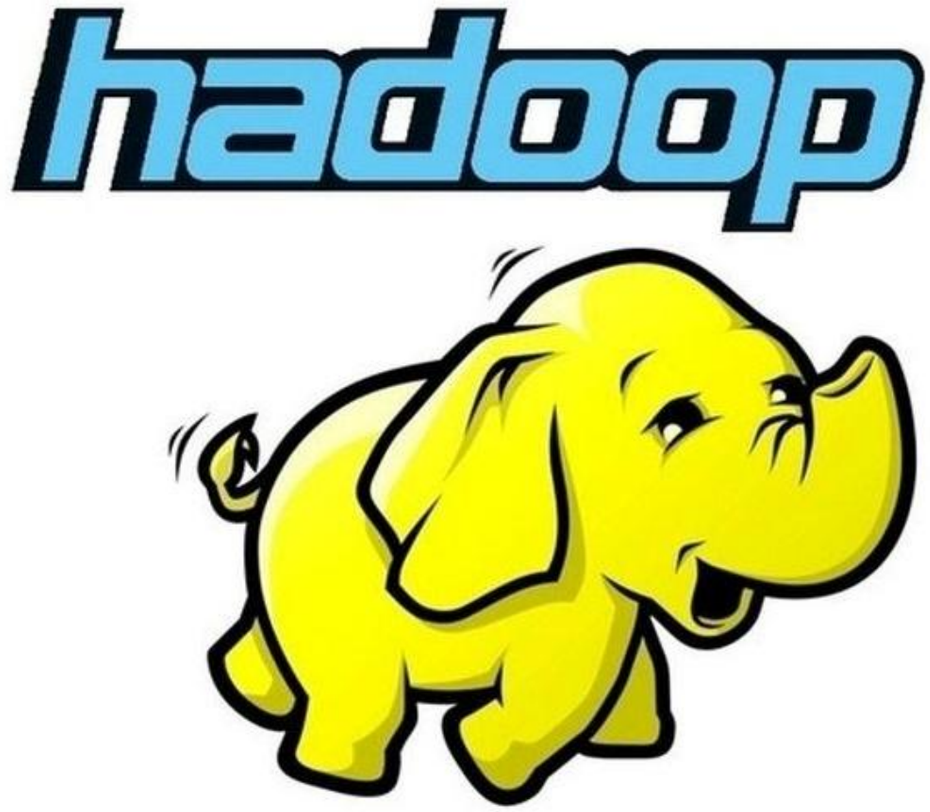
DICE
ANALYTICS

Empowering Data Analytics Ecosystem
DAY 03

https://www.facebook.com/diceanalytics/

https://www.linkedin.com/company/13294896/

DICE
ANALYTICS



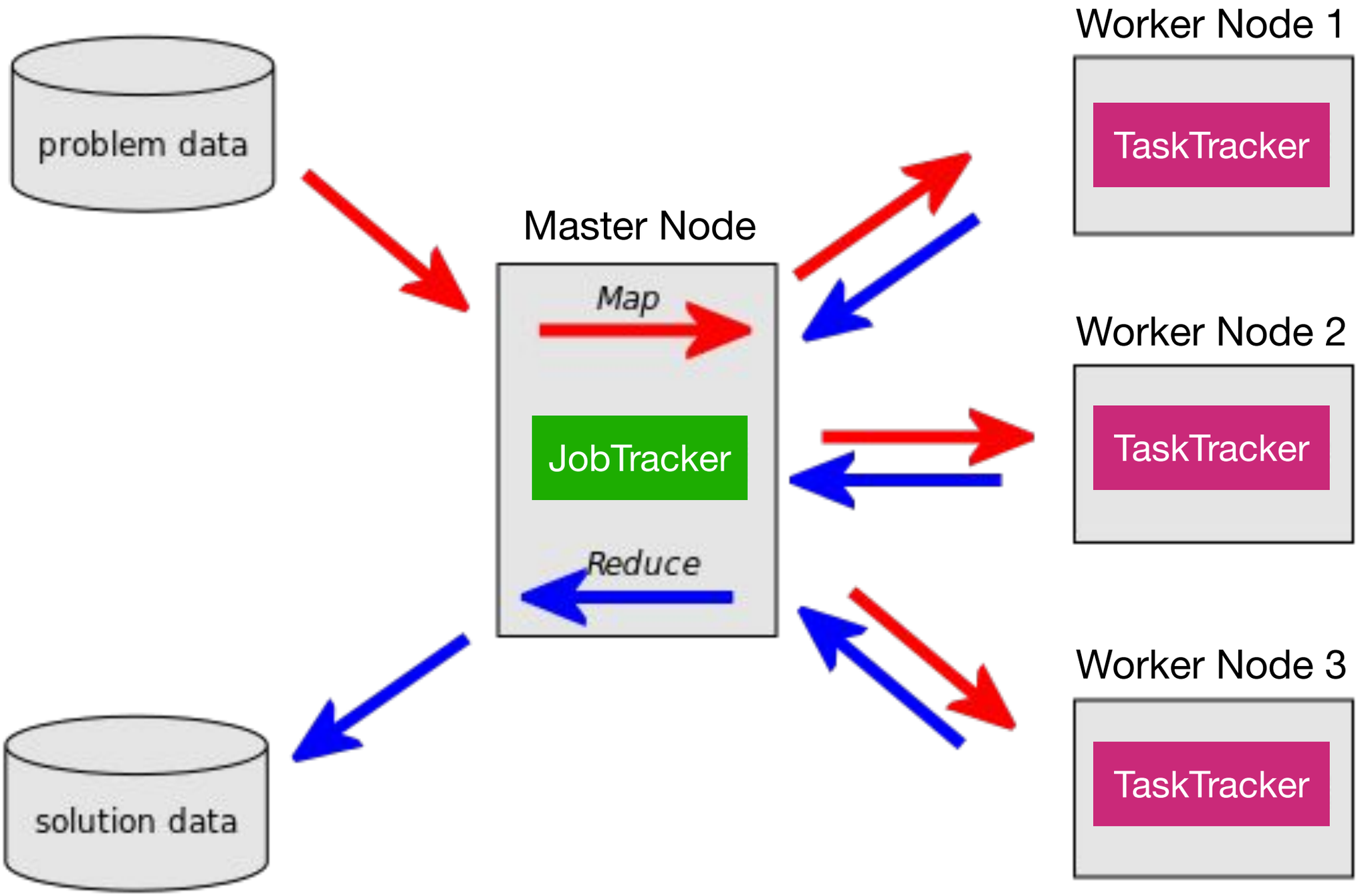
DICE
ANALYTICS

MapReduce

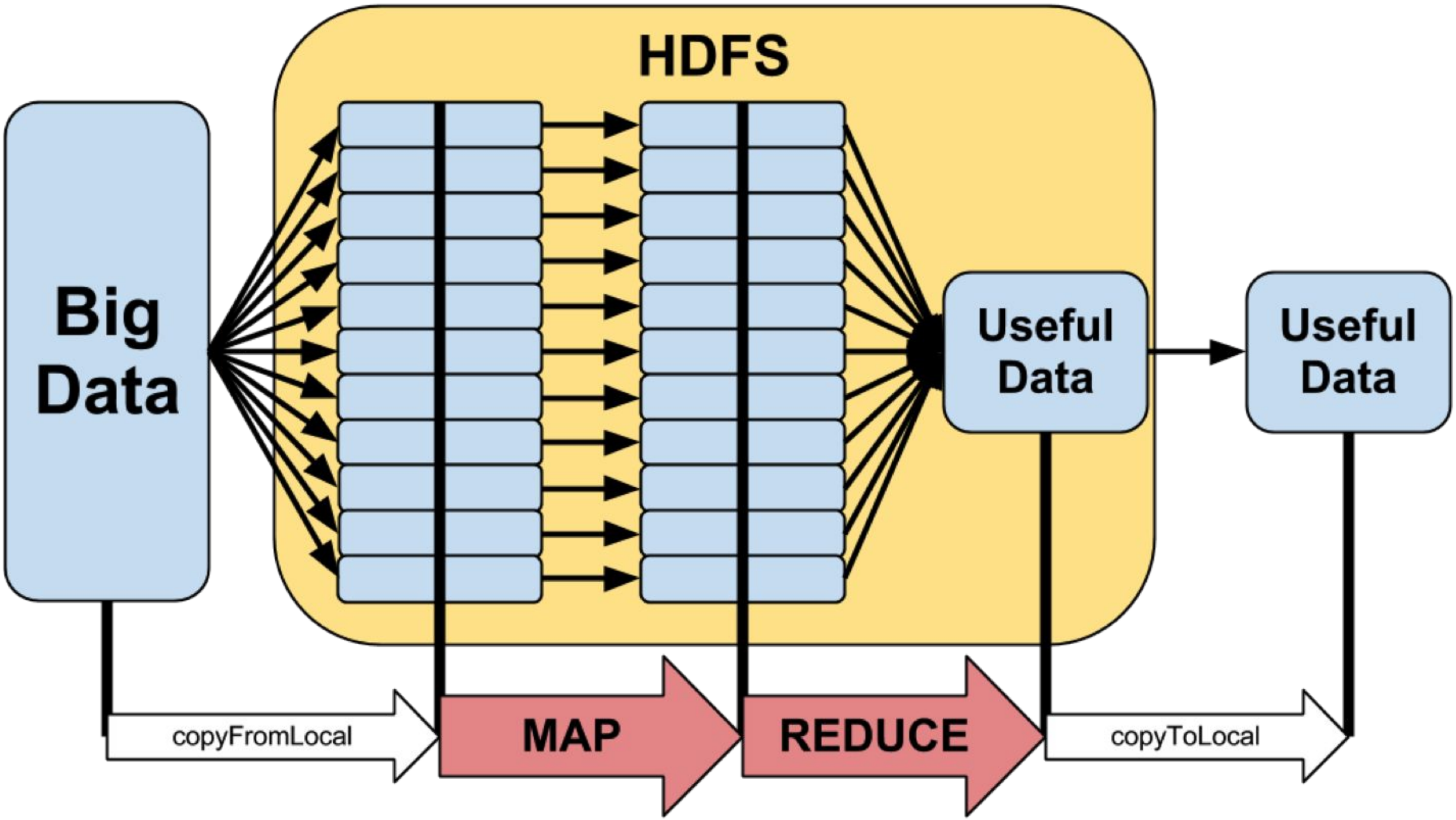
- Massive Parallel Processing technique for processing data which is distributed on a commodity cluster.
- Computations are sped up by breaking a big problem (Job) down into small problems (Tasks), which are then carried out in parallel by multiple computers.
- There are two components: Mappers & Reducers
- The number of Mappers is aligned by the number of blocks input data takes.
- The number of Reducers is set by the Application Developer.

http://www.fromdev.com/2010/12/interview-questions-hadoop-mapreduce.html

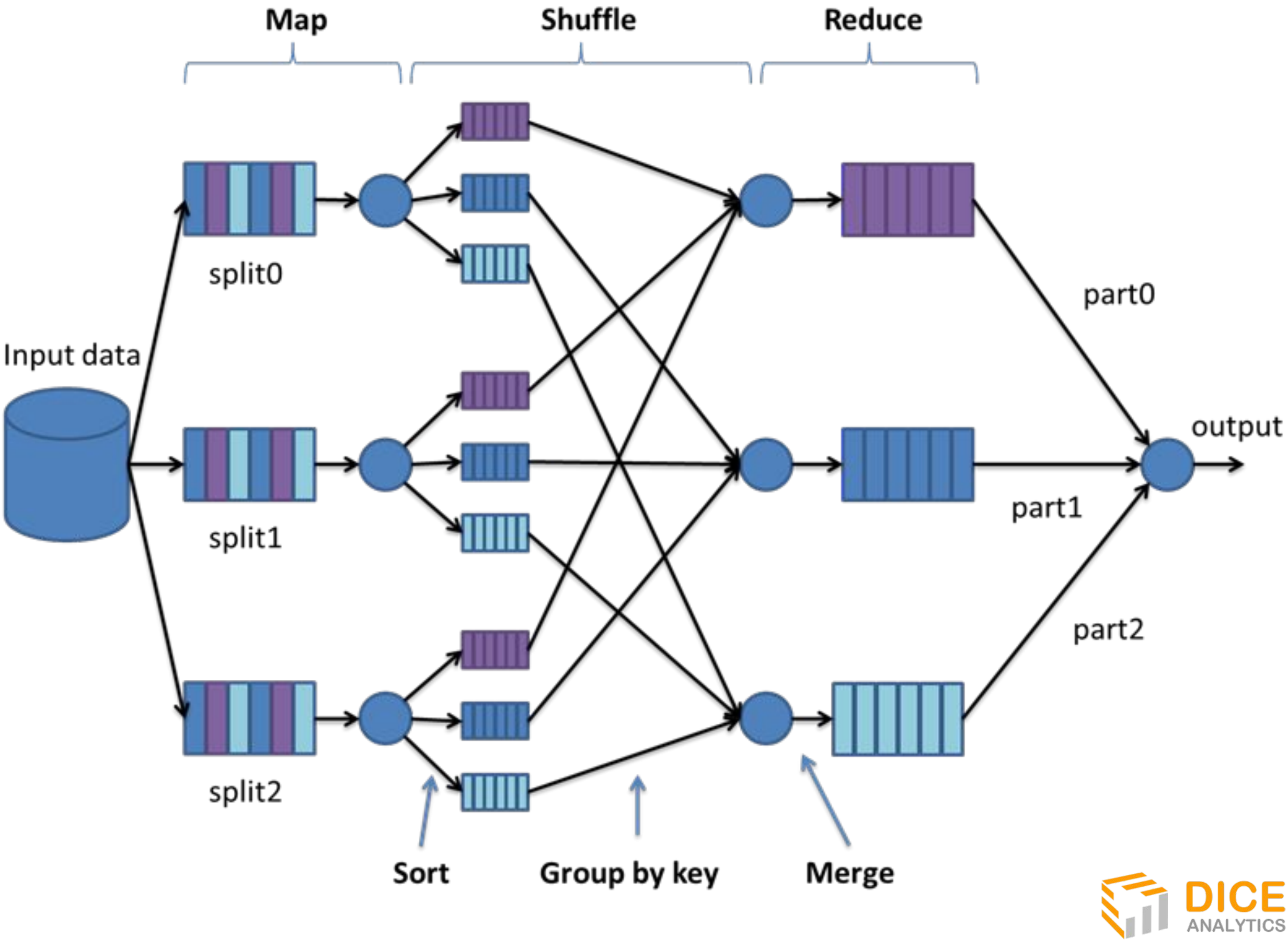
MapReduce



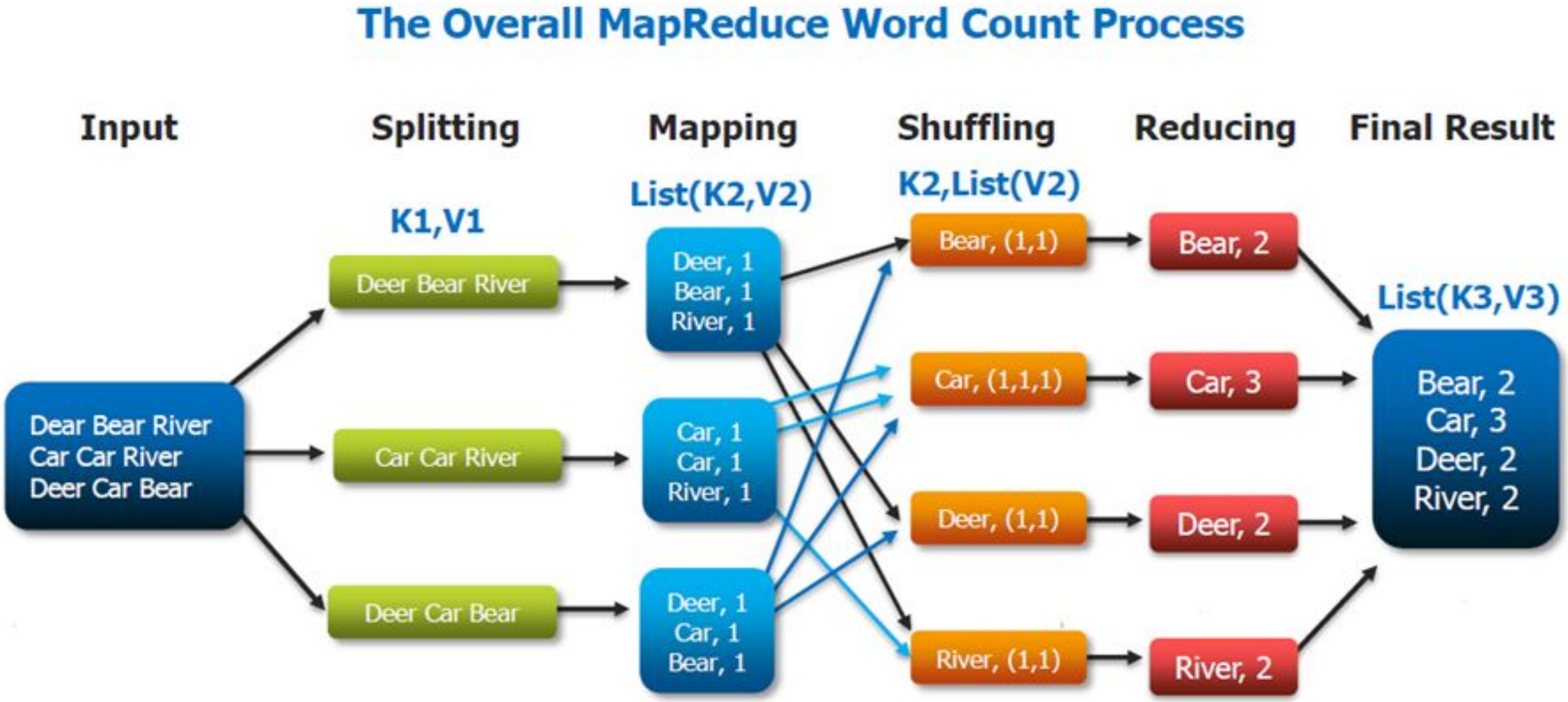
DICE ANALYTICS



DICE ANALYTICS


Map
Shuffle
Reduce
Input data
split0
split1
split2
part0
part1
part2
output
Sort
Group by key
Merge

Key Value Pair



https://wikis.nyu.edu/display/NYUHPC/Big+Data+Tutorial+1%3A+MapReduce

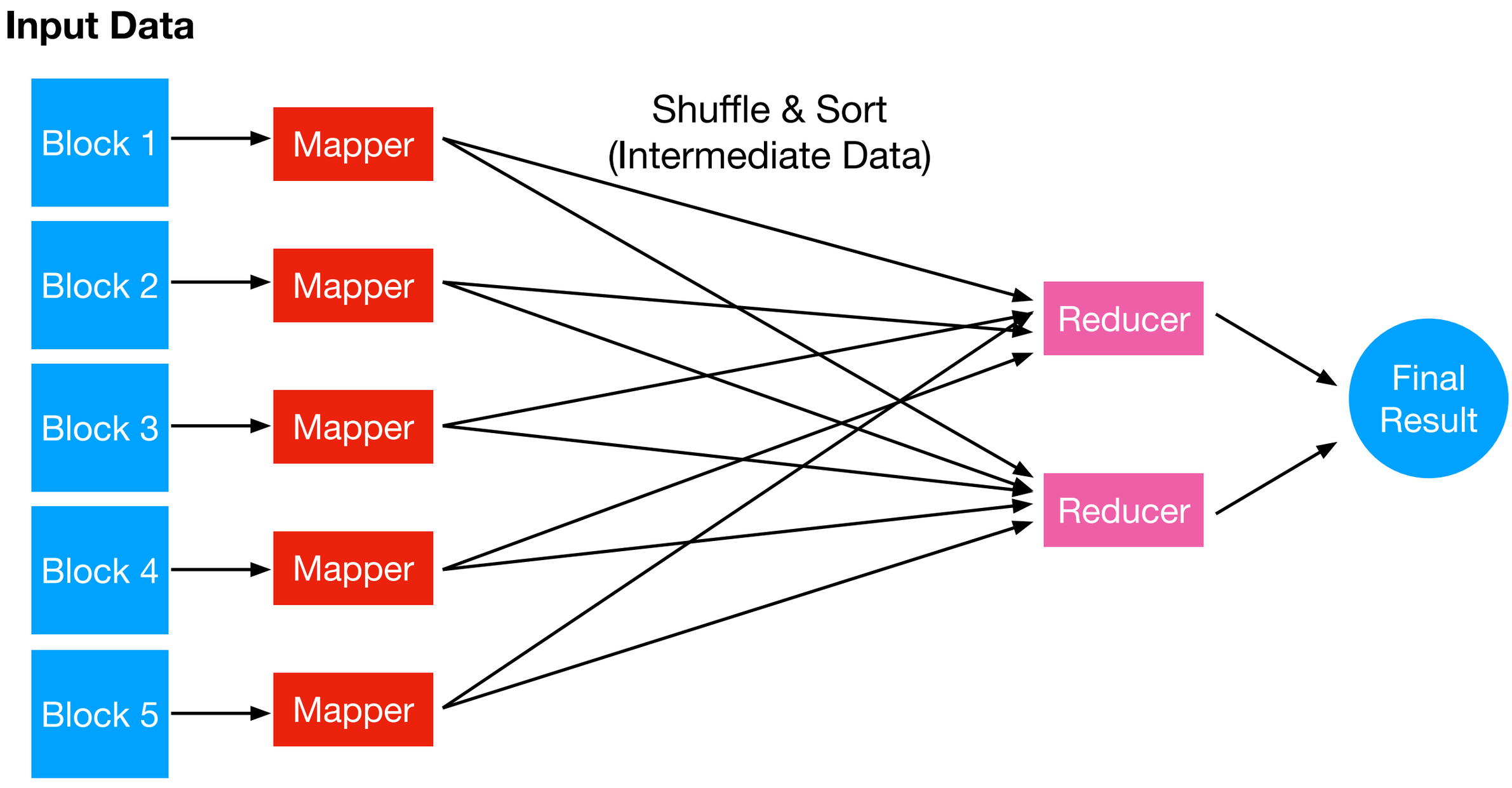
Mapper

- The Mappers reads data in the form of key/value pairs (KVPs).
- It outputs zero or more KVPs.
- The Mapper may use or completely ignore the inout keys:
  - For example, a standard pattern is to read a line of a file at a time.
  - In this case, the key is byte offset of line and value is actual line, so no point of reading key.
- Anything the Mapper writes must be in the form of KVPs:
  - This intermediate data is stored locally on Mapper node, not in HDFS.

Reducer

- After the Map phase is over, all the intermediate values for a given intermediate key are combined together into a list.
- This list is given to Reducer:
  - There may be one or more Reducers.
  - All values associated with a particular intermediate key are guaranteed to go to the same Reducer.
  - The intermediate keys, and their value lists, are passed in sorted order.
- The Reducer outputs zero or more KVPs, which are written to HDFS.

MapReduce

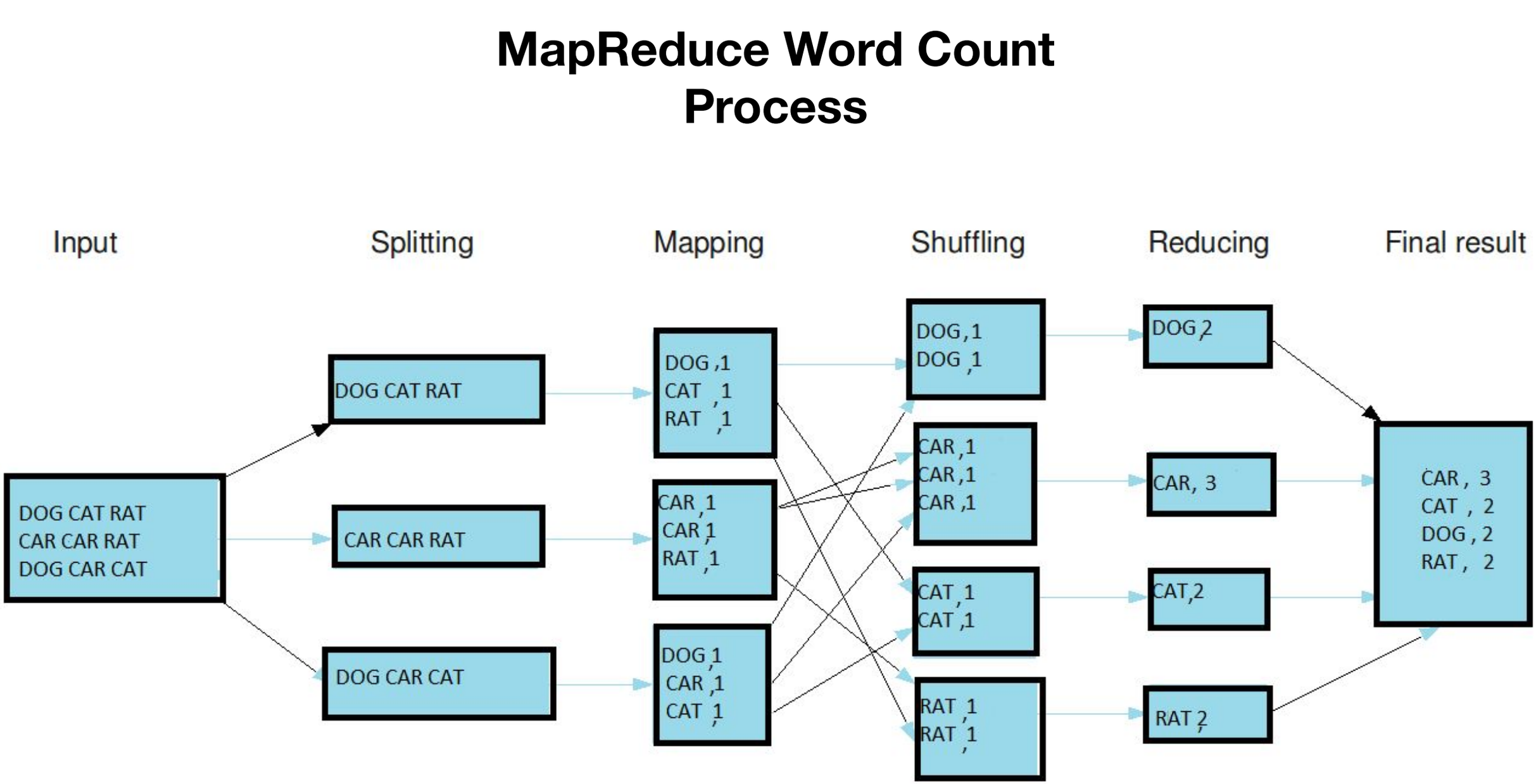


DICE ANALYTICS

MapReduce (Example)

MapReduce Word Count Problem

MapReduce (Example)



DICE ANALYTICS

One Mapper

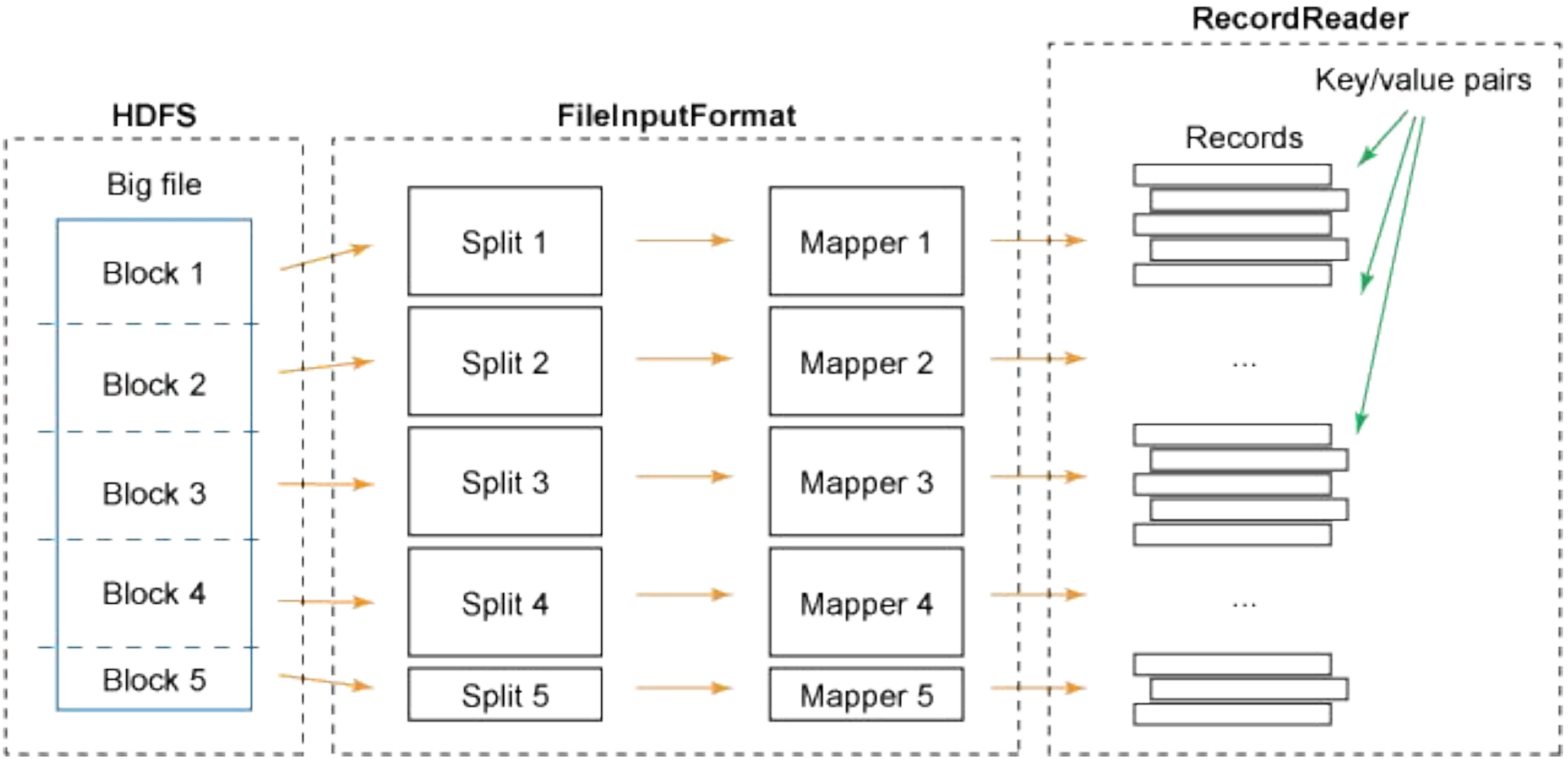
For

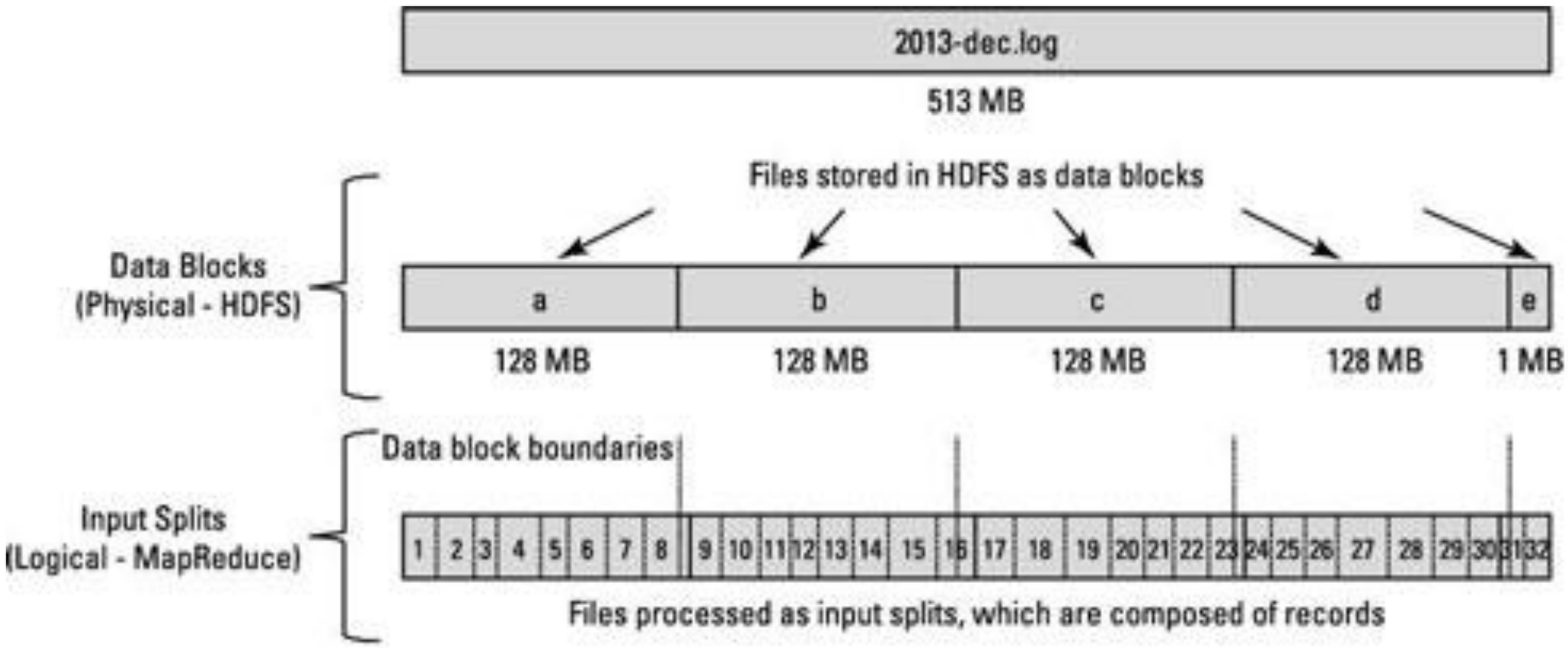
One Block

DICE ANALYTICS

Assignment

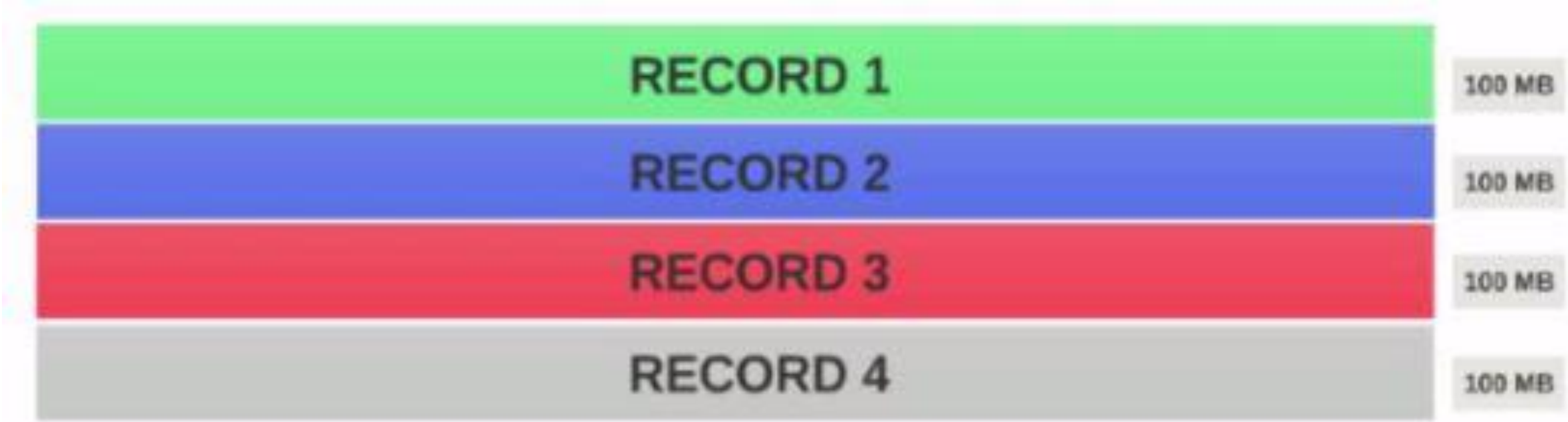
Difference between Block Size & Split Size


RecordReader
Key/value pairs
HDFS
FileInputFormat
Records
Big file
Block 1
Block 2
Block 3
Block 4
Block 5
Split 1
Split 2
Split 3
Split 4
Split 5
Mapper 1
Mapper 2
Mapper 3
Mapper 4
Mapper 5
...
...



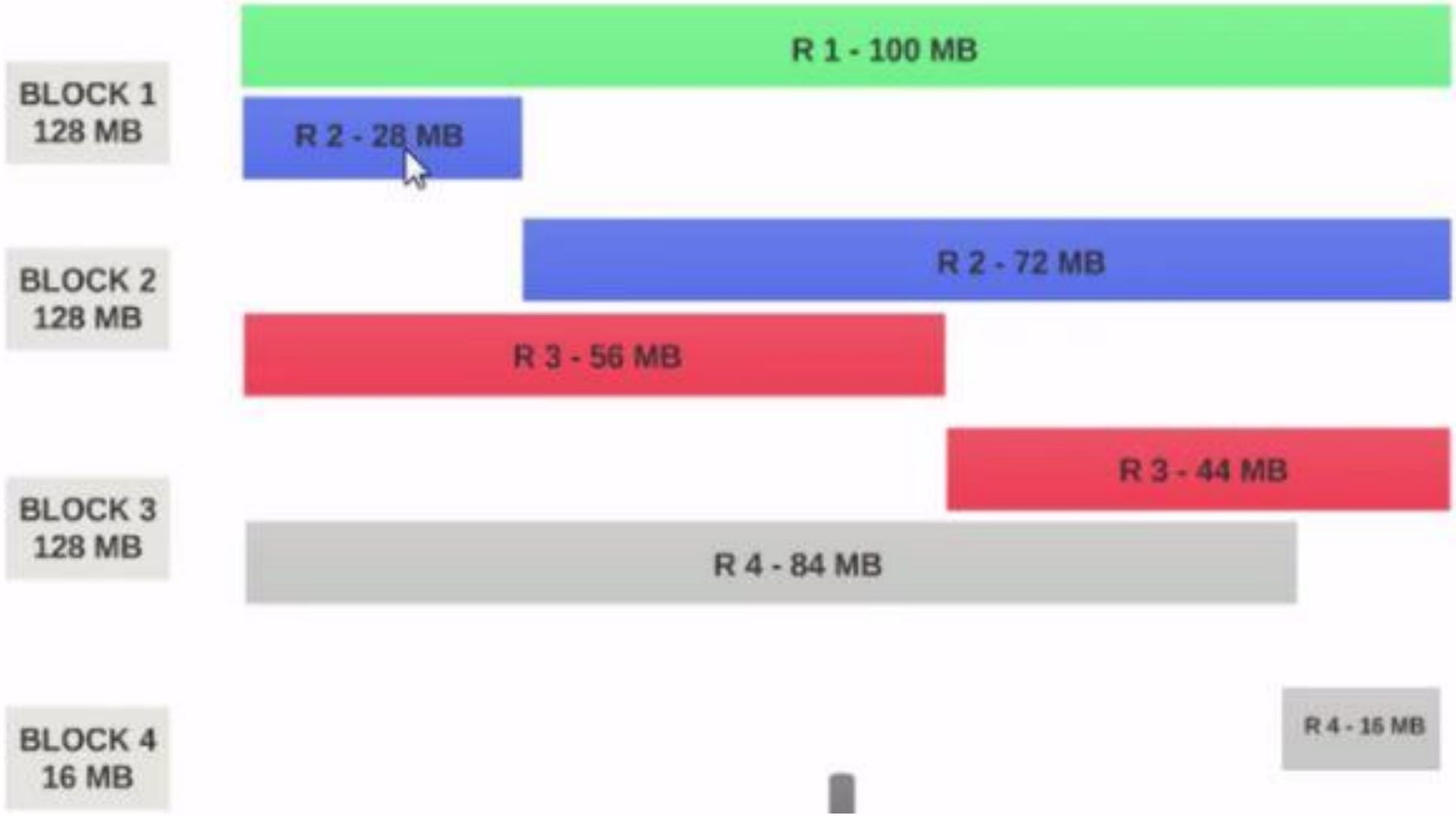
map: (K1, V1) → list(K2, V2)
reduce: (K2, list(V2)) → list(K3, V3)

Imagine a CSV File with 400 MB Size



Block Size 128 MB

DICE ANALYTICS

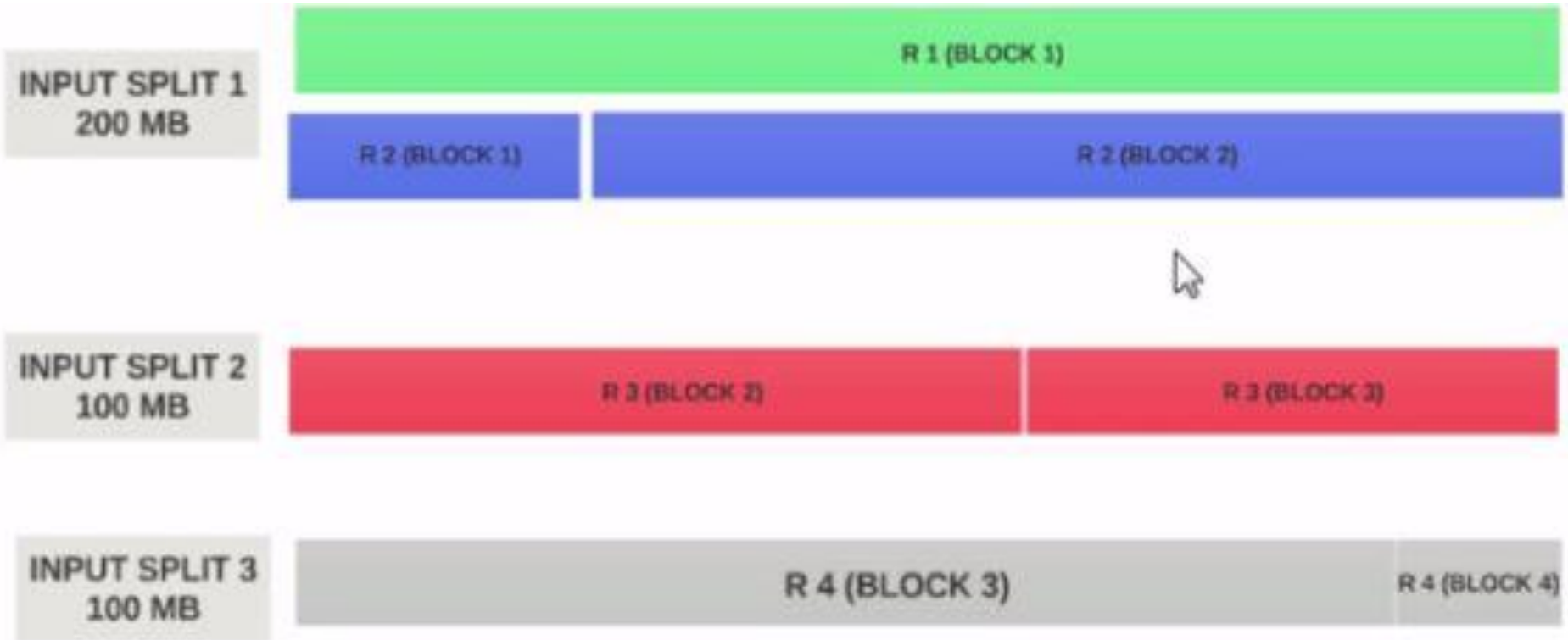


- **Block 1** contains the entire first record and a 28MB chunk of the second record.

- If a mapper is to be run on **Block 1**, the mapper cannot process since it won't have
  the entire second record.

- This is the exact problem that **input splits** solve. **Input splits** respects logical record boundaries.

- Lets Assume the **input split** size is **200MB**



- Therefore the **input split 1** should have both the record 1 and record 2. And input split 2 will not start with the record 2 since record 2 has been assigned to input split 1. Input split 2 will start with record 3.

- This is why an input split is only a **logical chunk** of data. It points to start and end locations with in blocks.

- If the input split size is n times the block size, an input split could fit multiple blocks and therefore less number of **Mappers** needed for the whole job and therefore less parallelism. (Number of mappers is the number of input splits)

- **input split size = block size** is the ideal configuration.

DICE ANALYTICS

One Mapper

For

One InputSplit

By default input split size = block size

DICE ANALYTICS

High Level Languages In Hadoop

Pig Latin

DICE ANALYTICS

What is Hive?

- Data Warehouse system for Hadoop.
- It can create schemas/table definitions that point to data in Hadoop, turning unstructured data into structured data.
- Helps to treat your data in Hadoop as Tables; which can be partitioned and bucketed.
- Helps to run Interactive SQL-like queries (HiveQL) at scale to query, summarize, explore and analyze data.
- JDBC/ODBC drivers are available.
- Integrable with custom MapReduce code.

Hive’s alignment with SQL

SQL Datatypes	SQL Semantics
INT	SELECT, LOAD, INSERT from query
TINYINT/BIGINT/SMALLINT	Expressions in WHERE and HAVING
BOOLEAN	GROUP BY, ORDER BY, SORT BY
FLOAT	CLUSTER BY, DISTRIBUTE BY
DOUBLE	Sub-queries in FROM clause
STRING	ROLL UP and CUBE
BINARY	UNION
TIMESTAMP	LEFT, RIGHT, FULL INNER/OUTER JOIN
ARRAY/MAP/STRUCT/UNION	CROSS JOIN, LEFT SEMI JOIN
DECIMAL	Windowing Functions (Over, Rank, etc.)
CHAR	Sub-queries for IN/NOT IN, HAVING
VARCHAR	EXISTS, NOT EXISTS
DATE	INTERSECT, EXCEPT

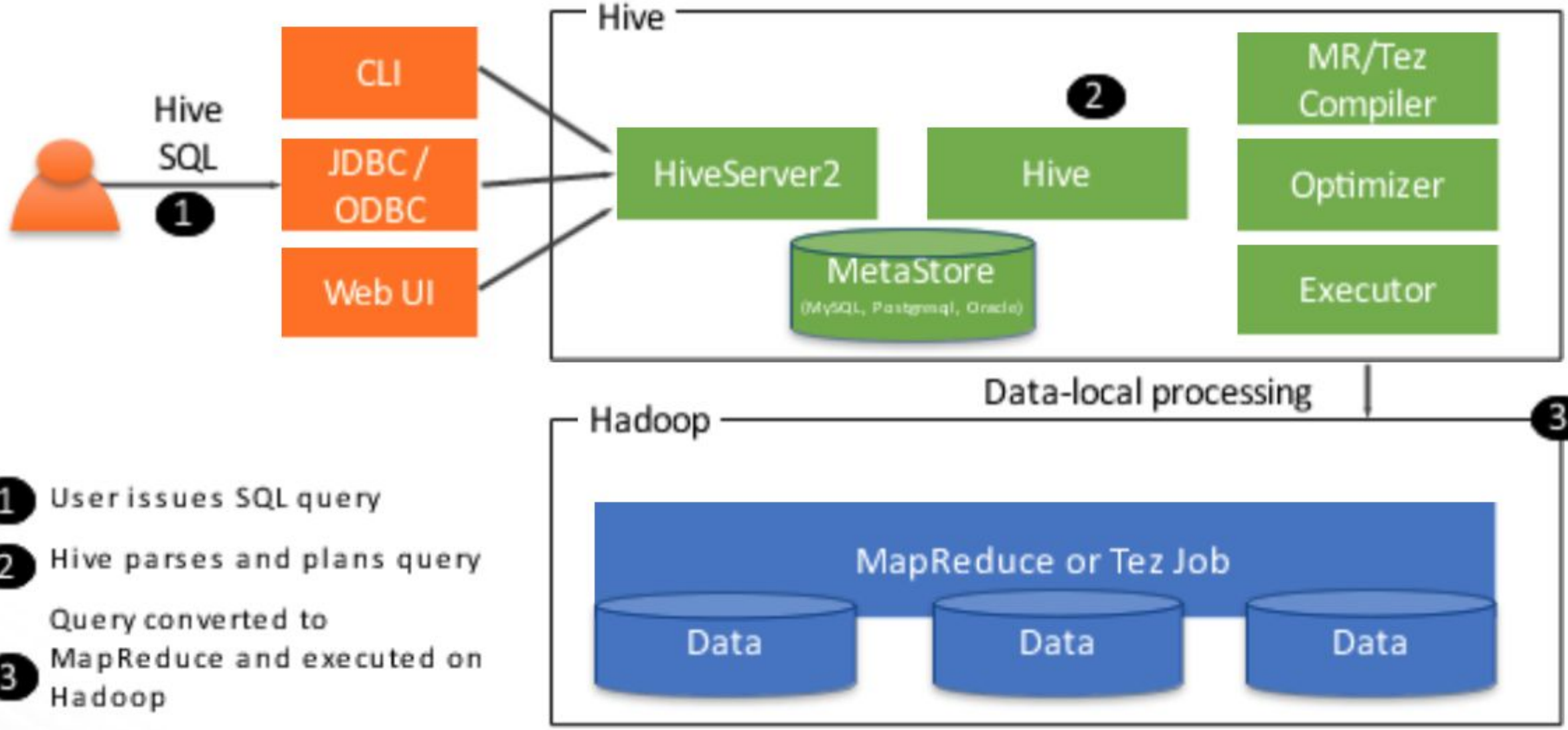
Hive is NOT…

- **… a relational database:**
  - Hive uses database store metadata only, the data processed by Hive is stored on HDFS.

- **… designed for on-line transaction processing:**
  - Latency & Overhead for Hive queries is generally high even for small jobs.

- **… suited for real-time queries and row-level updates:**
  - Hive is best used for batch jobs over large data sets of immutable data (web logs).

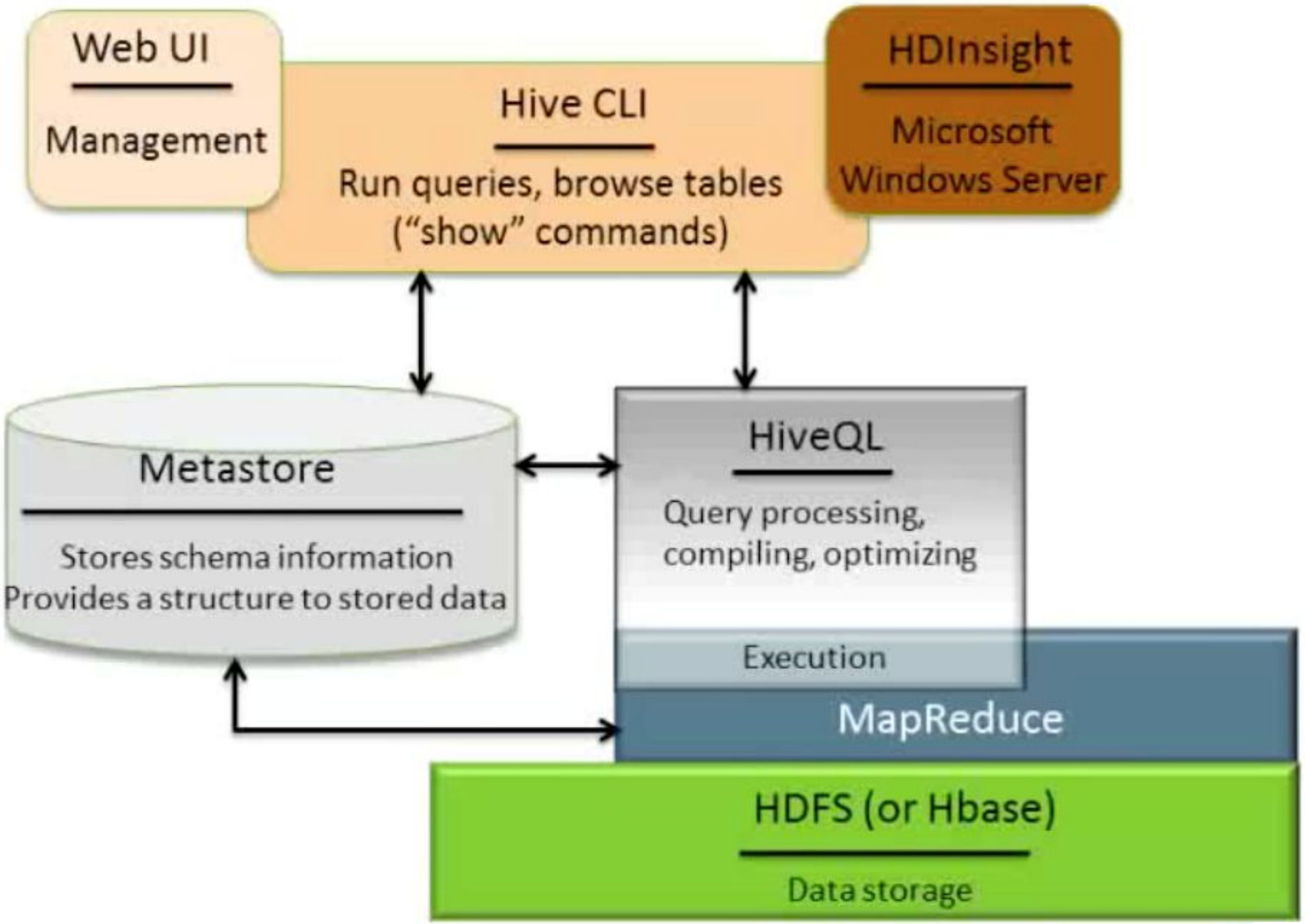
Hive Query Process



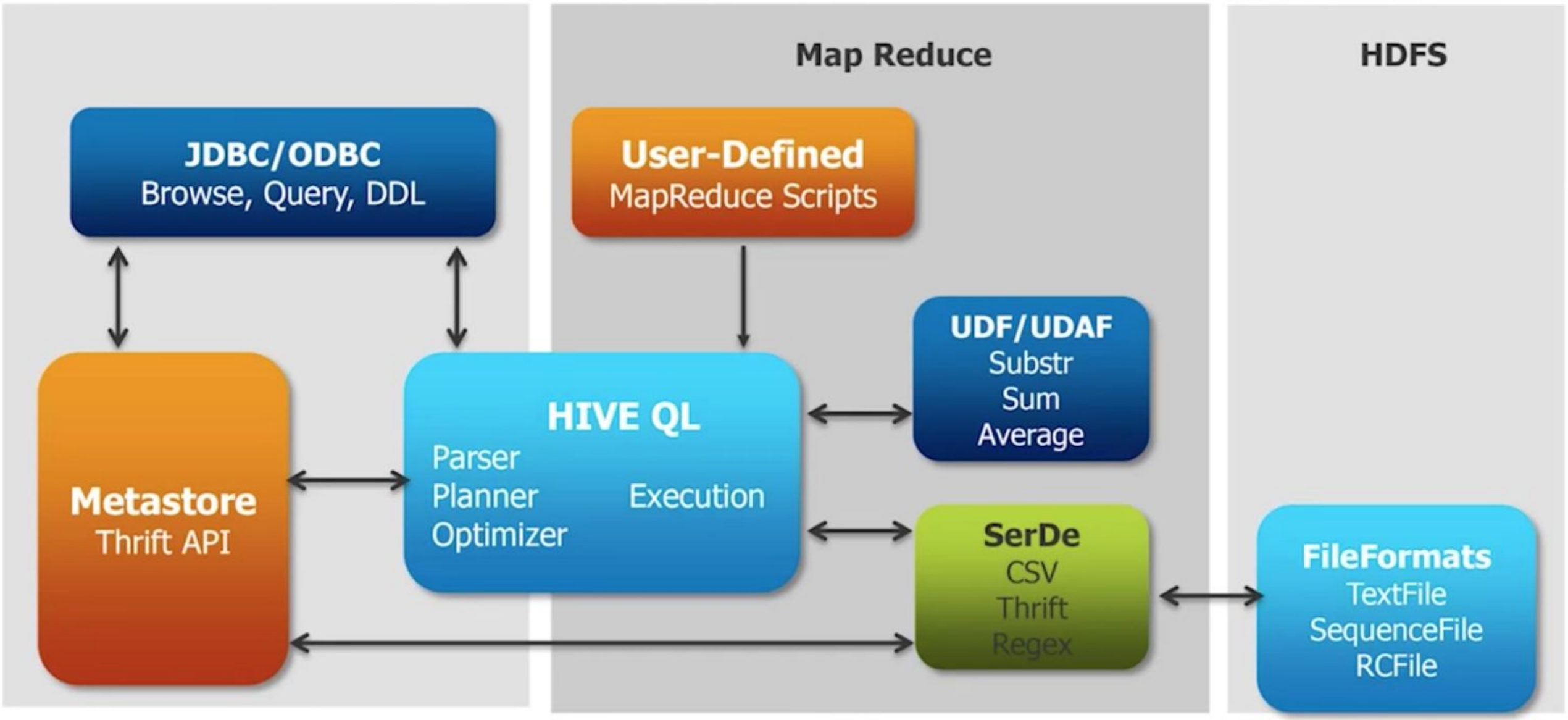
1. User issues SQL query
2. Hive parses and plans query
3. Query converted to MapReduce and executed on Hadoop

DICE ANALYTICS

Hive Architecture



Hive Architecture



DICE ANALYTICS

HiveQL

- HiveQL is similar to other SQLs.
  - Uses familiar relational database concepts (tables, rows, columns and schema).
  - Based on the SQL-92 specifications.
- Supports multi-table inserts via your code:
  - Access “Big Data” via tables.
- Converts SQL queries into MapReduce Jobs:
  - User does not need to know MapReduce.
- Also supports plugging custom MapReduce scripts into queries.

Submitting Hive Query

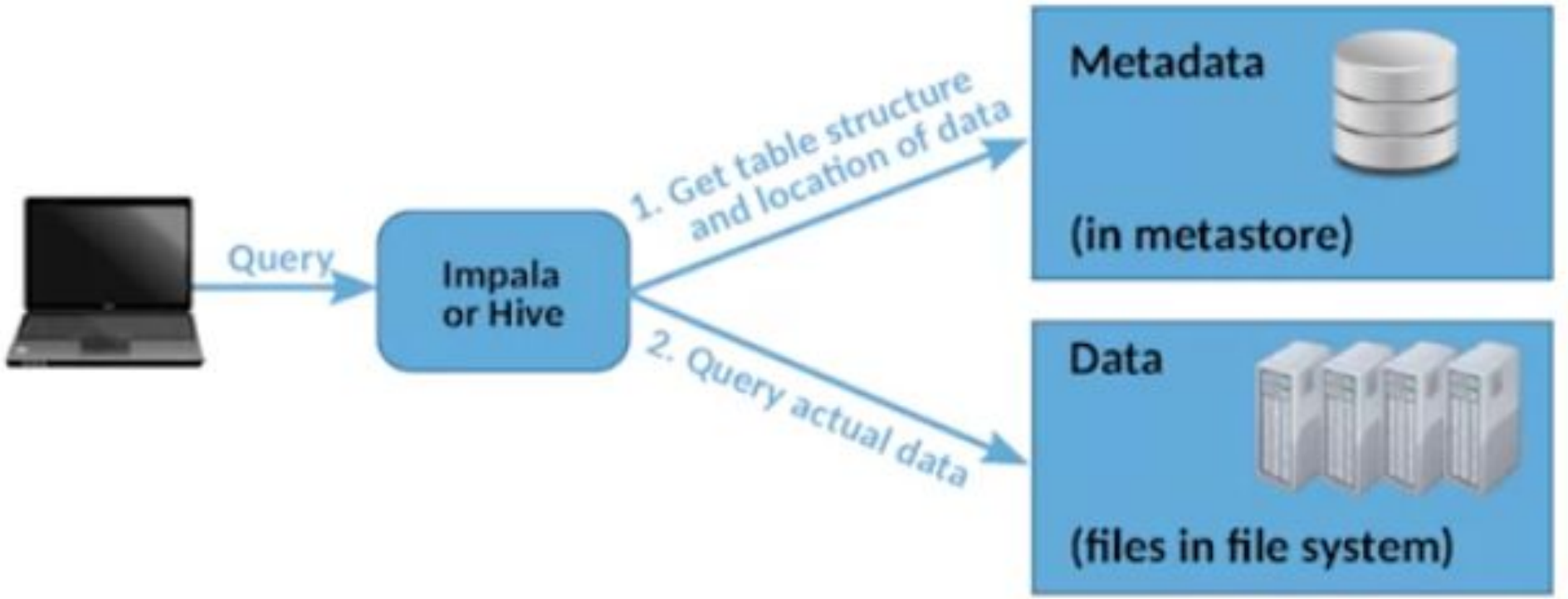
- Submitting Hive Queries through CLI:
  - Beeline of HiveServer2 (HS2)

- Submitting Hive Queries through GUI:
  - Ambari Hive View
  - Zeppelin
  - DBVisualizer
  - Dbeaver
  - SquirrelSQL
  - SQLWorkBench

Hive Table

- A Hive Table consists of:
  - Data: typically a file or group of files in HDFS.
  - Schema: In the form of metadata stored in a relational database.
- Schema and Data are separate:
  - A schema can be defined for existing data.
  - Data can be added or removed independently.
  - Hive can be “pointed” at existing data.
- You have to define a schema if you have existing data in HDFS that you want to use in Hive.

Hive Table



Managing Hive Table

Operation	Command Syntax
See Current Tables	SHOW TABLES
Check the Schema	DESCRIBE myTable;
Change the Table Name	ALTER TABLE myTable RENAME TO mt;
Add a Column	ALTER TABLE myTable ADD COLUMNS (myCol STRING);
Drop a Column	ALTER TABLE myTable DROP PARTITION (age = 17)

Hive Table (Example)

```
hive> CREATE TABLE myTable (name string, age int)
      ROW FORMAT DELIMITED
      FIELDS TERMINATED BY ','
      STORED AS TEXTFILE
```

Each row is comma delimited text

HiveQL statements are Terminated with semicolon

- Hive-managed Table
- When such table is dropped, underlying data is deleted

DICE ANALYTICS

Hive Table (Example)

```
hive> CREATE EXTERNAL TABLE myTable (name string, age int)
      ROW FORMAT DELIMITED
      FIELDS TERMINATED BY ';'
      STORED AS TEXTFILE
      LOCATION '/usr/train/mydata/';
```

Specifying Table Location

- External Table
- When such table is dropped, underlying data is not deleted

Hive Data Loading

- Loading data from local file system:

```
LOAD DATA LOCAL INPATH '/temp/customers.csv' OVERWRITE INTO TABLE customers;
```

- Loading data that is already in HDFS:

```
LOAD DATA INPATH '/temp/customers.csv' OVERWRITE INTO TABLE customers;
```

- Insert Into Select Case:

```
INSERT INTO birthdays
    SELECT firstName, lastName, birthday
    FROM customers
    WHERE birthday IS NOT NULL;
```

Hive Data Querying

```
SELECT * FROM customers;
```

```
FROM customers
    SELECT firstName, lastName, address, zip
    WHERE orderID > 0
    GROUPBY zip;
```

```
SELECT customers.*, orders.*
    FROM customers
    JOIN orders ON
    (customers.customerID = orders.customerID);
```

DICE ANALYTICS

Hive and MapReduce

Map Phase

Reduce Phase

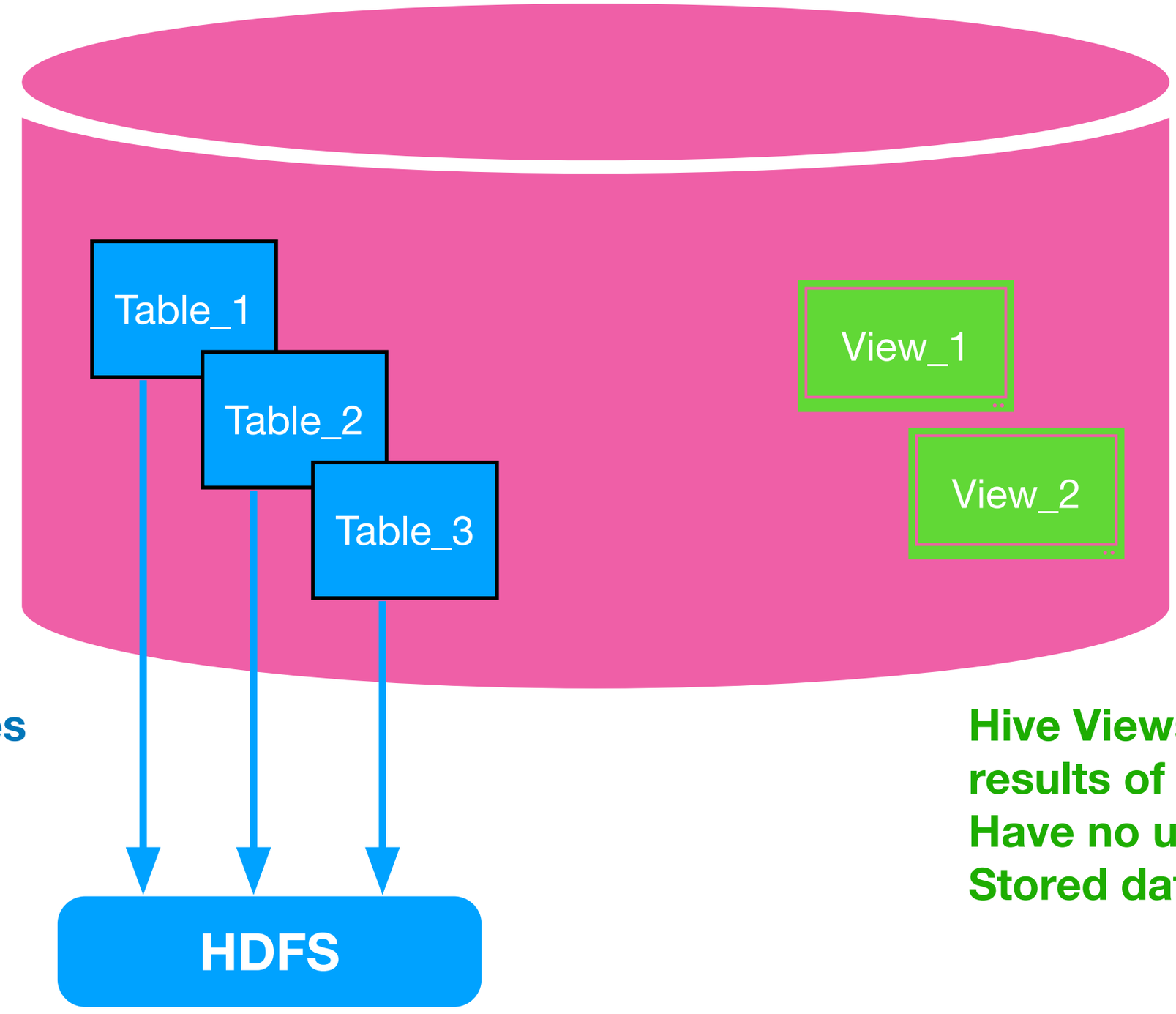
SORT & SHUFFLE

```
SELECT c.zip, COUNT(*)
FROM customers c
WHERE …
```

```
JOIN order o ON
(c.cid = o.cid)
GROUP BY c.zip
ORDER BY DISTINCT
```

DICE ANALYTICS

Hive Views



Hive Tables Map to Folders in HDFS

Hive Views are the results of queries and Have no underlying Stored data

DICE ANALYTICS

Defining Hive Views

```
CREATE VIEW 2010_visitors AS
    SELECT fname, lname, time_of_arrival, info_comment
    FROM wh_visitors
    WHERE
    cast(substring(time_of_arrival, 6, 4) AS int) >= 2010
    AND
    cast(substring(time_of_arrival, 6, 4) AS int) < 2011;
```

Using Hive Views

- Hive Views can be used just as Hive Tables:

```
FROM 2010_visitors
    SELECT *
    WHERE info_comment LIKE ‘%CONGRESS%’
    ORDER BY lname;
```

Hive Tutorial

1. show databases
2. Describe default.sample_07
3. Show create table default.sample_07
4. select * from xademo.call_detail_records limit 100;
5. select phone_type,count(*) from xademo.call_detail_records group by phone_type;
6. show create table xademo.call_detail_records;
7. CREATE TABLE myTable (name string, age int)
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ','
   LINES TERMINATED BY '\n'
   STORED AS TEXTFILE;

8. Check where a directory have been created ?

9. Create a CSV File with Name and Age in it load it through Hive View/HDFS Commands

10. Select * from default.myTable

11. Select count(*) from default.myTable

12. Upload the same file in directory again

13. Select count(*) from default.myTable

14. Delete table default.myTable

15. Check if the underlying directory delete or not ?

16.

```
CREATE External TABLE myTable (name string, age int)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

14. Upload the file again and execute select query

15. Now Delete the myTable and Check the directory again

16. CREATE External TABLE myTable3 (name string, age int)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
     STORED AS TEXTFILE
     Location '/usr/train/mydata/';

17. Check the directory whether the above directory created or not ?

18. LOAD DATA INPATH '/apps/hive/warehouse/mytable/Sample_File_Hive_2.csv'
     OVERWRITE INTO TABLE default.mytable3;

19. Now check the /usr/train/mydata directory and check if the file exists or not ?

20. Now check the directory /apps/hive/warehouse/mytable

Use Hive Over XML

STEP 1 : CREATING INPUT XML FILE WHICH WE WILL LOAD IN HIVE TABLE
Create XML file using below data and store it as students.xml

<student> <id>1</id> <name>Milind</name> <age>25</age> </student>
<student> <id>2</id> <name>Ramesh</name> <age>Testing</age> </student>

STEP 2: Store the XML file in HDFS directory /usr

STEP 3 : CREATING HIVE TABLE
create table student_xml(studinfo string) ;

STEP 4: LOADING DATA INTO HIVE TABLE
load data inpath '/usr/students.xml' into table student_xml;

STEP 5: QUERYING THE LOADED DATA
select * from student_xml;

STEP 7: CREATING A VIEW ON TOP OF NEWLY CREATED HIVE TABLE FOR GETTING NEWLY ADDED RECORDS
create view student_xml_view as SELECT xpath_int(studinfo ,'student/id'),xpath_string(studinfo ,'student/name')
,xpath_string(studinfo ,'student/age') FROM student_xml;

STEP 8: QUERYING THE CREATED VIEW
select * from student_xml_view;

Using Serde a Complex Route:

https://community.hortonworks.com/content/kbentry/972/hive-and-xml-pasring.html
https://www.safaribooksonline.com/library/view/hadoop-real-world-solutions/9781784395506/ch05s03.html

Does

Hive

Provides a Native Serde for

JSON and XML ?

https://cwiki.apache.org/confluence/display/Hive/SerDe
https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-RowFormats&SerDe

Assignment

Load a JSON File in Hive Table

https://www.guru99.com/data-extraction-hive.html

DICE ANALYTICS

Hive DDL Optional clauses

```
CREATE TABLE tablename (
    col1 TYPE,
    col2 TYPE,
    …)
  ROW FORMAT …
  STORED AS …
  LOCATION … ;
```

```
CREATE EXTERNAL TABLE dbname.tablename (col1 TYPE, col2 TYPE, …)
    ROW FORMAT …
    STORED AS …
    LOCATION …
    TBLPROPERTIES;
```

```
CREATE EXTERNAL TABLE dbname.tablename (col1 TYPE, col2 TYPE, …)
    ROW FORMAT …
    STORED AS …
    LOCATION …
    TBLPROPERTIES('skip.header.line.count'='1');
```

Create Table Statement: Hive SerDes

- Hive provides features for working with unstructured text data, semi-structured data in formats like JSON, and data that lacks consistent delimiters.
- You must explicitly specify the SerDe in the **CREATE TABLE** statement.
- The statement includes the clause **ROW FORMAT SERDE** followed by the fully qualified name of the Java class that implements the SerDe, enclosed in quotes.

Create Table Statement: Hive SerDes

- Try It!
- Do the following to create a table named **tunnels** in the **dig** database.
- 1. Examine the data in the file **training_materials/analyst/data/tunnels.csv** on the local file system of the VM. Note that it's a comma-delimited text file.
- 2. In the ***Hive*** query editor, execute the following statement. Note that it uses the **OpenCSVSerde** rather than the **ROW FORMAT DELIMITED** syntax you used in the previous lesson. Also you *must* use Hive, not Impala.
- **CREATE TABLE dig.tunnels**
- **(terminus_1 STRING, terminus_2 STRING, distance SMALLINT)**
- **ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde';**
- 3. In a Terminal window, run the following statement (all on one line) to move the **tunnels.csv** into the table directory. Do not include the **$**; that's the prompt to indicate this is a command-line shell command, not a query.
- **$ hdfs dfs -put ~/training_materials/analyst/data/tunnels.csv /user/hive/warehouse/dig.db/tunnels/**
- You'll learn more about this statement later in this course.
- 4. Use the data source panel or a Hive **SELECT *** statement to verify that the **tunnels** table has the data, with values in the correct columns. Remember that you can query this table only with Hive, not with Impala.

Hive Query Performance Patterns

- Only Metadata
- Fetch Tasks
- Only Map Phase
- Map and Reduce Phases
- Multiple Map and Reduce Phases

Hive Query Performance Patterns: Only Metadata

- Fastest type of queries
- Retrieves metadata from metastore.
- **DESCRIBE customers;**

Hive Query Performance Patterns: Fetch Tasks

- **SELECT** queries that do not require the underlying data processing engine.
- Hive server executes these queries internally and fetches data from file system.
- Eliminates the overhead of starting separate processes to execute the job

SELECT * FROM customers LIMIT 10;

Hive Query Performance Patterns: Only Map Phase

- Requires only a map phase and no reduce phase.
- For example, when a query inserts data into another table, Hive executes the query as a map-only job.

```
INSERT INTO TABLE ny_customers
SELECT * FROM customers
WHERE state = 'NY';
```

Hive Query Performance Patterns: Map and Reduce Phases

- Requires both map and reduce phases
- e.g. a query that performs aggregation

```
SELECT COUNT(cust_id)
FROM customers
WHERE zipcode=94305;
```

- Hive projects and filters the data in the map phase, then aggregates it using the **COUNT** function in the reduce phase.

Hive Query Performance Patterns: Multiple Map and Reduce Phases

- Slowest type of query
- Requires multiple map and reduce phases

```
SELECT zipcode, COUNT(cust_id) AS num
FROM customers
GROUP BY zipcode
ORDER BY num DESC
LIMIT 10;
```

Complex data types

- **Combines multiple values into a single column**
- **Array**
- **Map**
- **Struct**

Complex data types: ARRAY

- **An ordered list of values, all having the same data type.**
- **e.g. multiple phone numbers**

name	phones
Alice	[555-1111, 555-2222, 555-3333]
Bob	[555-4444]
Carlos	[555-5555, 555-6666]

- **Phones is an array with numbers in string data type**
- **All elements of an ARRAY must be of the same type.**

Complex data types: MAP

- **Represents key-value pairs e.g. which phone number is for what purpose**
- **All keys have the same data type and all values have the same data type**
- **e.g. key is STRING type and value is INT type**

name	**phones**
Alice	{home:555-1111, work:555-2222, mobile:555-3333}
Bob	{mobile:555-4444}
Carlos	{work:555-5555, home:555-6666}

Complex data types: STRUCT

- **Represents named fields, which can have different data types.**
- **For example, use a STRUCT to store addresses, with each part of the address a different field**

name	address
Alice	{street:742 Evergreen Terrace, city:Springfield, state:OR, zipcode:97477}
Bob	{street:1600 Pennsylvania Ave NW, city:Washington, state:DC, zipcode:20500}
Carlos	{street:342 Gravelpit Terrace, city:Bedrock}

Complex data types: Creating Tables

- **Use Hive to create tables**
- **Create a hive table for following data**

a,Alice,555-1111|555-2222|555-3333

b,Bob,555-4444

c,Carlos,555-5555|555-6666

Complex data types: Creating Tables

```
CREATE TABLE customers_phones_array
(cust_id STRING,
name STRING,
phones ARRAY<STRING>)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
COLLECTION ITEMS TERMINATED BY '|';
```

Complex data types: Creating Tables

- **MAP type is declared in the column list of the CREATE TABLE statement using MAP<*keytype, valuetype*>.**
- **Create a hive table for following data**

a,Alice,home:555-1111|work:555-2222|mobile:555-3333

b,Bob,mobile:555-4444

c,Carlos,work:555-5555|home:555-6666

Complex data types: Creating Tables

```
CREATE TABLE customers_phones_map
(cust_id STRING,
name STRING,
phones MAP<STRING,STRING>)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
COLLECTION ITEMS TERMINATED BY '|'
MAP KEYS TERMINATED BY ‘:’ ;
```

Complex data types: Creating Tables

- **STRUCT type is declared in the column list of the CREATE TABLE statement using STRUCT<*field1:TYPE1, field2:TYPE, …*>**
- **Order of the STRUCT fields in the table definition *must* match the order in the data files.**
- **Create a hive table for following data**

a,Alice,742 Evergreen Terrace|Springfield|OR|97477

b,Bob,1600 Pennsylvania Ave NW|Washington|DC|20500

c,Carlos,342 Gravelpit Terrace|Bedrock

Complex data types: Creating Tables

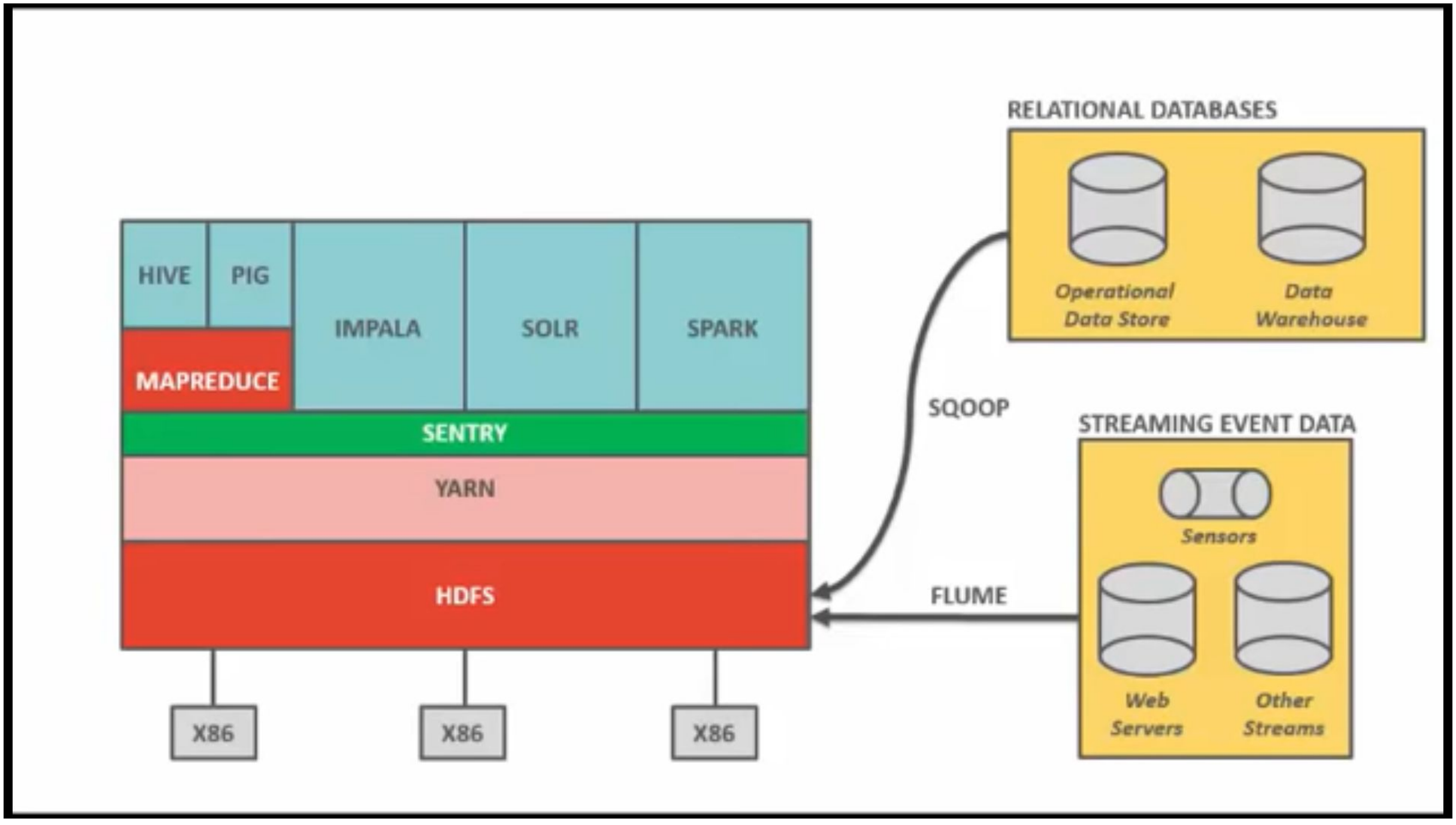
```
CREATE TABLE customers_addr
(cust_id STRING,
name STRING,
address STRUCT<street:STRING,
               city:STRING,
               state:STRING,
               zipcode:INT>)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
COLLECTION ITEMS TERMINATED BY '|';
```

Complex data types: Query tables with Hive

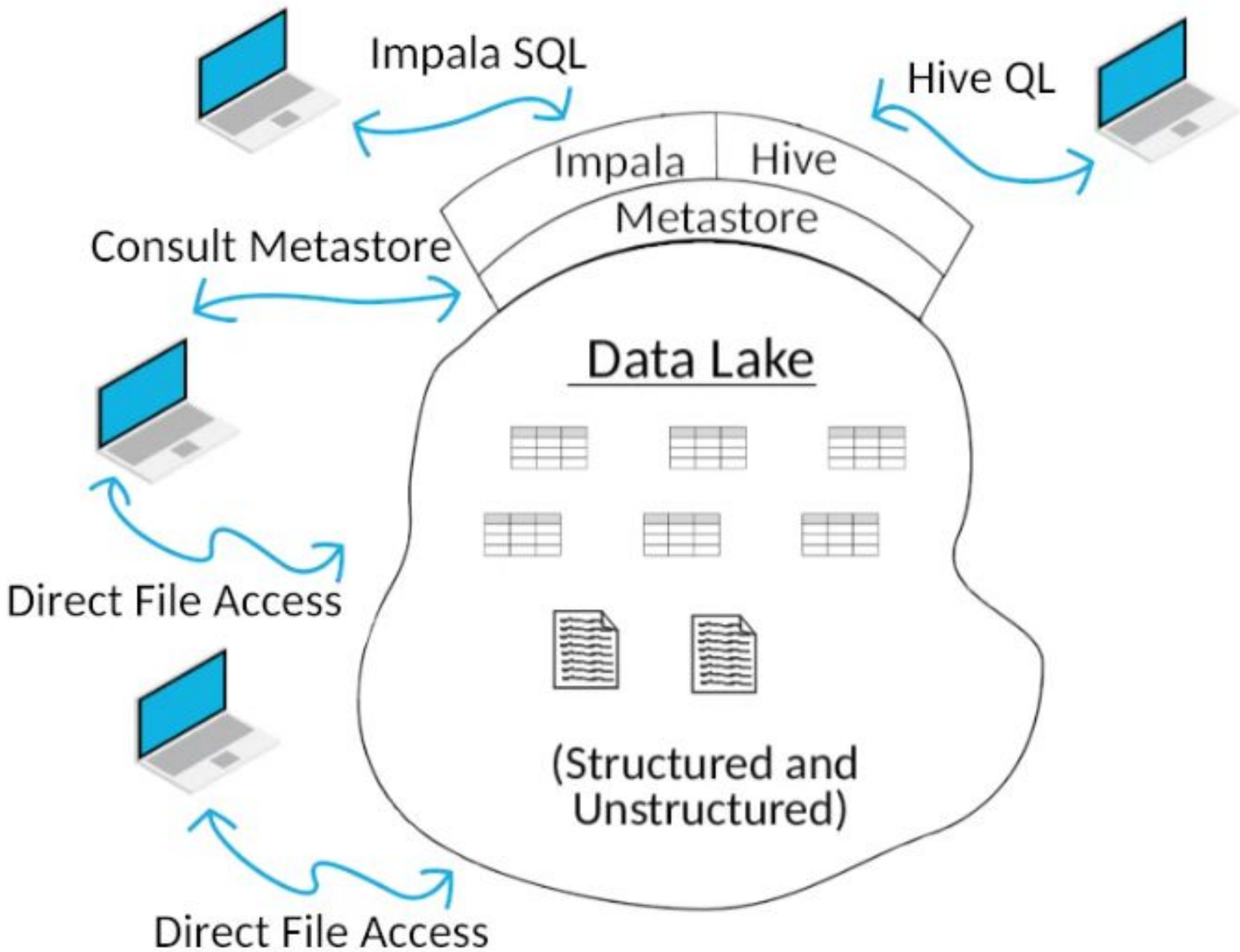
- **Run SELECT * FROM *tablename*; for each table and note how the complex column appears in the results.**
- **Try to return individual values in complex data types**

```
SELECT name, phones[0], phones[1]
FROM customers_phones_array;

SELECT name, phones['home'] AS home
FROM customers_phones_map;

SELECT name, address.state, address.zipcode
FROM customers_addr;
```

Hive vs. Impala

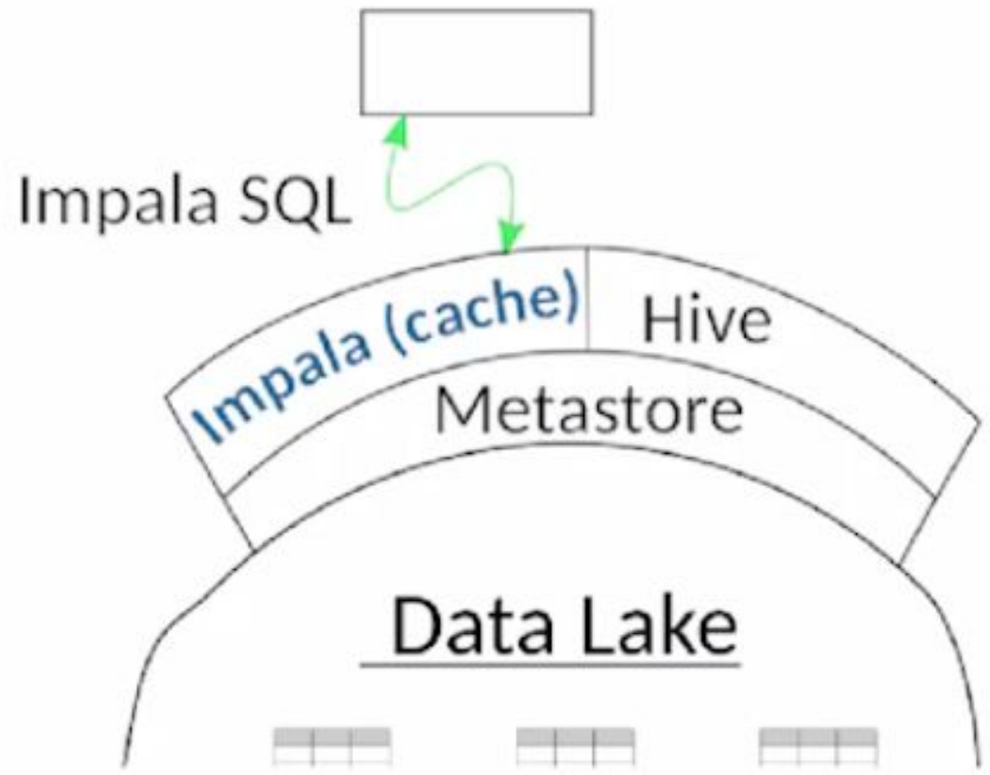


Hive vs. Impala



Hive vs. Impala

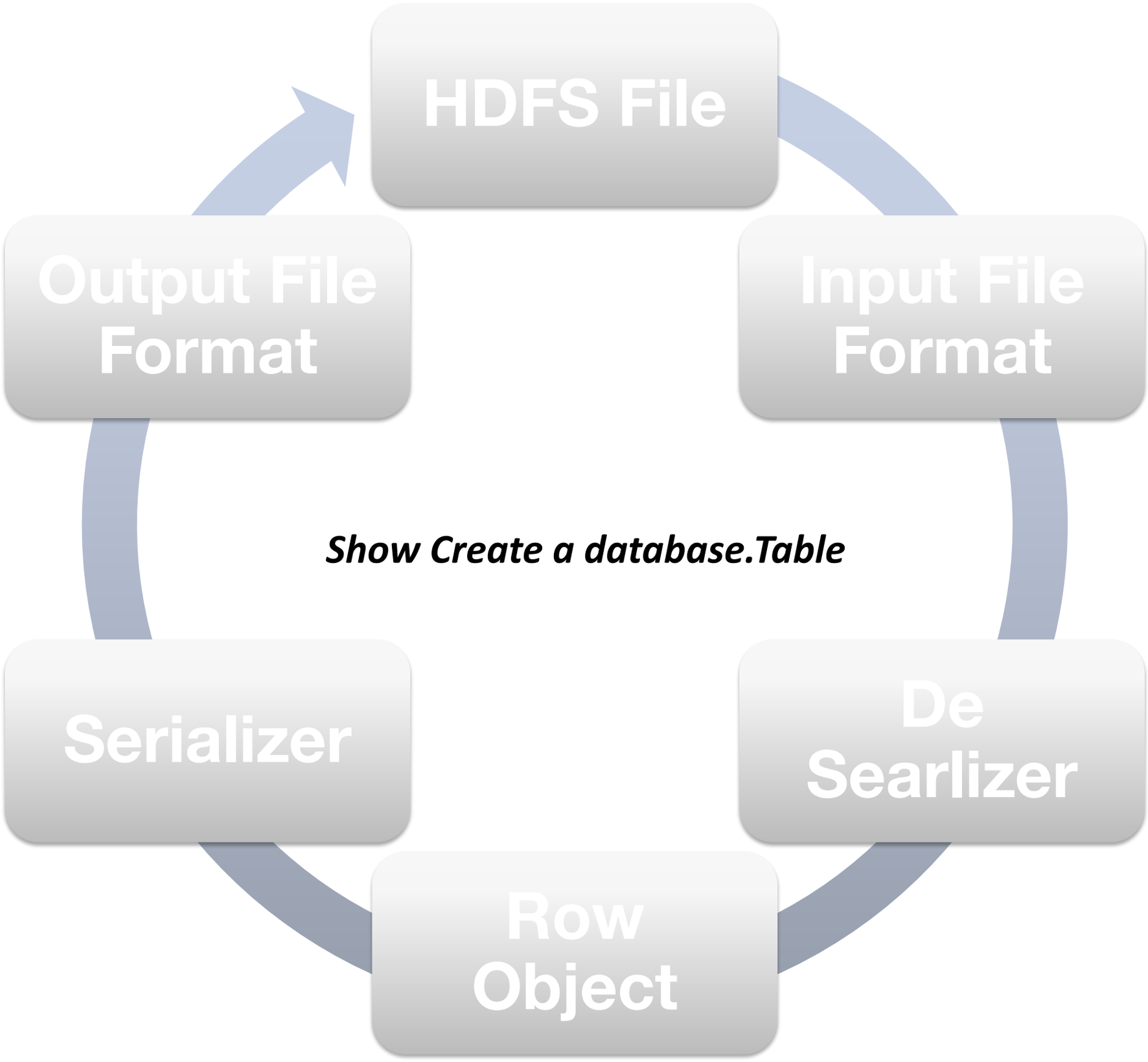
- **Hive converts SQL queries to map reduce jobs.**
- **Impala daemons are installed on data nodes of cluster.**
- **If there any changes outside of Impala, an impala metadata refresh is needed.**



Hive vs. Impala

External Metadata Change	Required Action	Effect on Local Caches
Table schema modified or new data added to a table	`REFRESH tablename;`	Reloads the metadata for one table immediately; reloads storage block locations for new data files only
New table added, or data in a table extensively altered, such as by HDFS balancing	`INVALIDATE METADATA tablename;`	Marks the metadata for a single table as stale; when the metadata is needed, all storage block locations are retrieved

- Caution: `INVALIDATE METADATA` with no table name affects *all* users
  - Marks the entire cache as stale, to be rebuilt completely when needed
  - Can be time-consuming with large tables or lots of tables
  - Use only when needed



Show Create a database.Table

File Formats & Hive

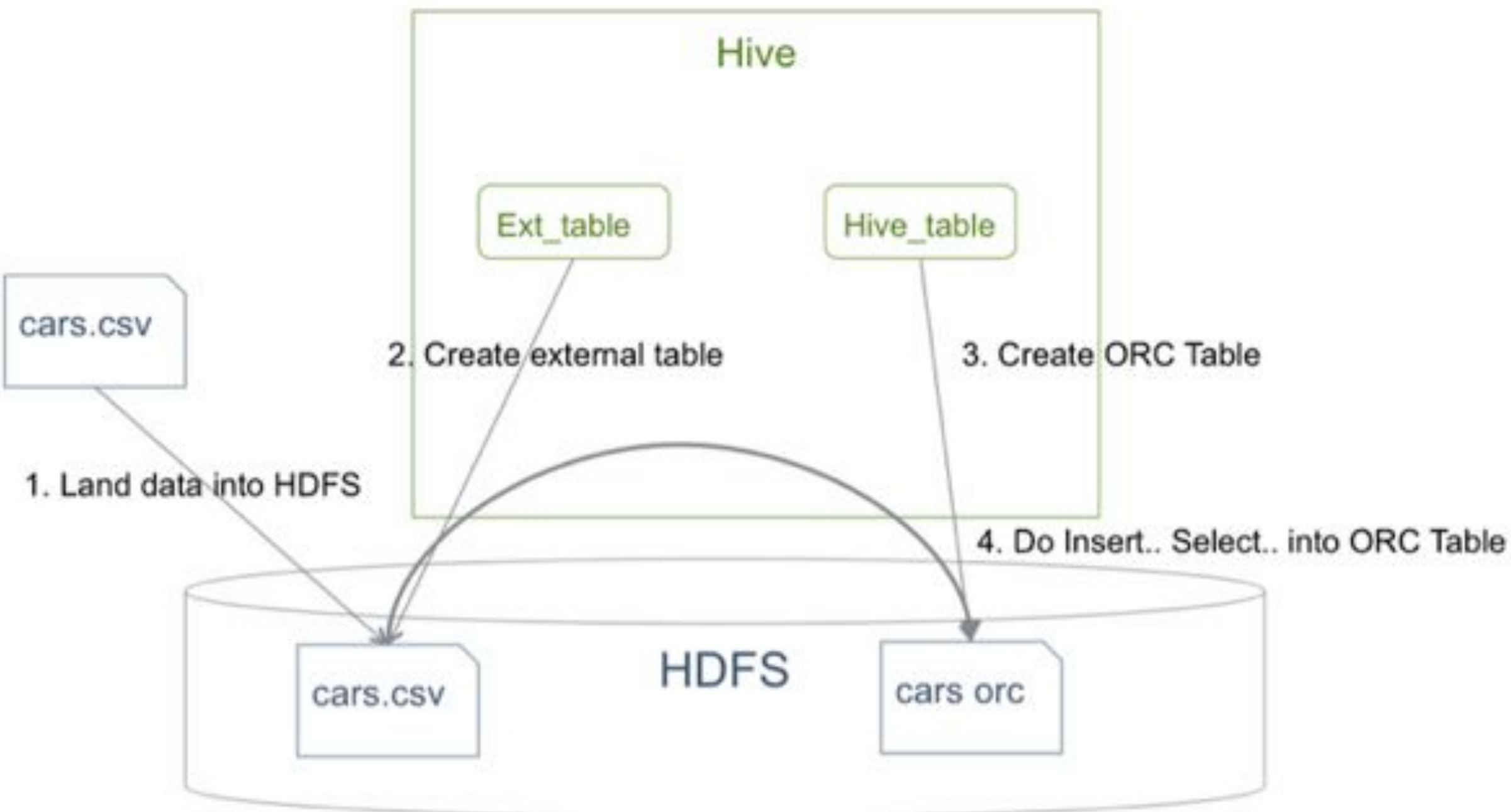
```
CREATE External TABLE myTable3 (name string, age int)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    STORED AS TEXTFILE
    STORED AS SEQUENCEFILE
    STORED AS RCFILE
    STORED AS ORCFILE
    STORED AS PARQUET
    Location '/usr/train/mydata/';
```

DICE ANALYTICS

Serdies & Hive

```
CREATE External TABLE myTable3 (name string, age int)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe‘
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe‘
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
STORED AS TEXTFILE
STORED AS TEXTFILE
Location '/usr/train/mydata/';
```

Create Hive Table with ORC Files



Step #1

```
CREATE EXTERNAL TABLE IF NOT EXISTS Cars(
        Name STRING,
        Miles_per_Gallon INT,
        Cylinders INT,
        Displacement INT,
        Horsepower INT,
        Weight_in_lbs INT,
        Acceleration DECIMAL,
        Year DATE,
        Origin CHAR(1))
COMMENT 'Data about cars from a public database'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
location '/user/<username>/visdata';
```

Step #2

Move Data to External Table (Creating External Table isn’t Mandatory)

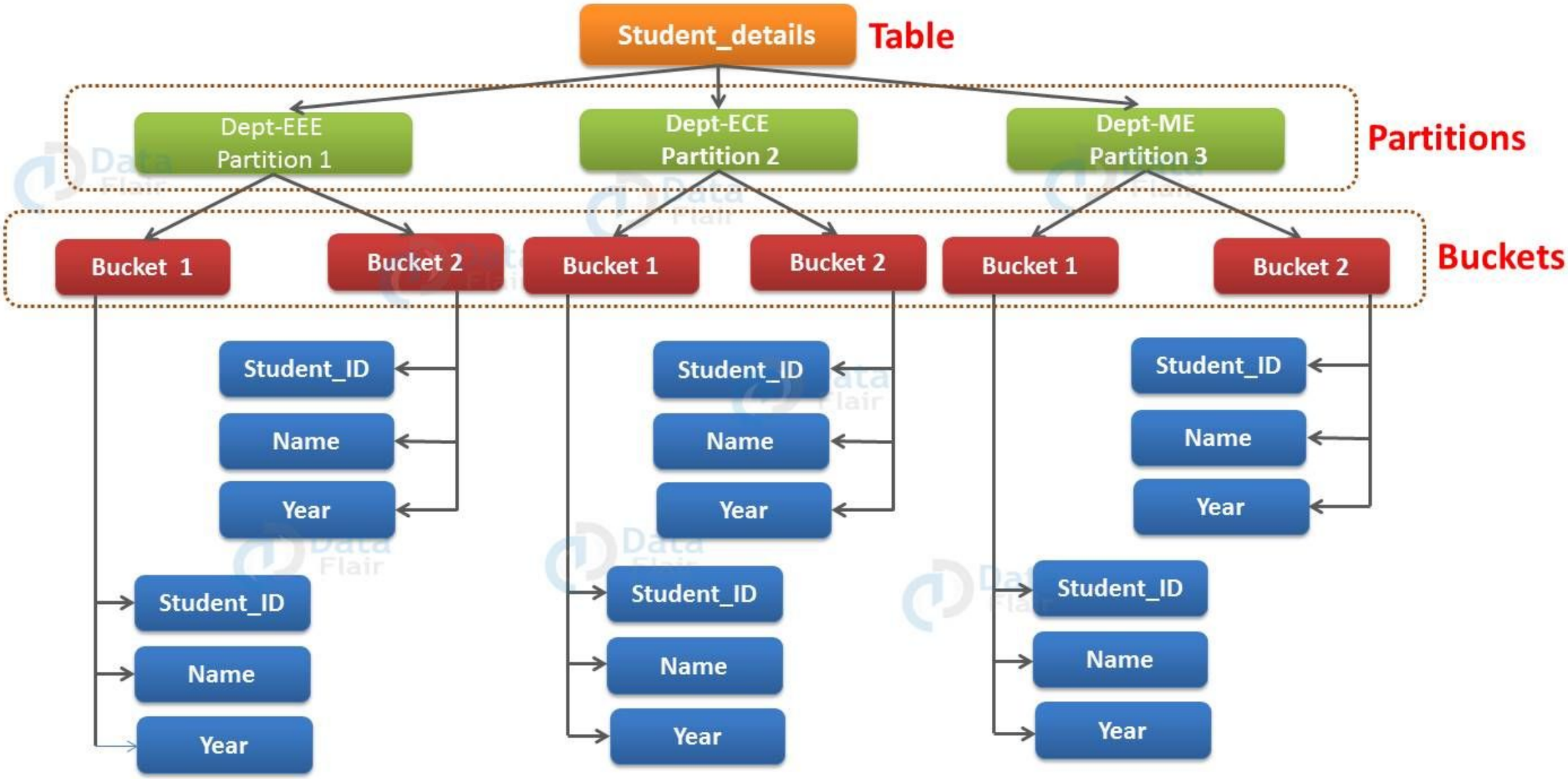
Step #3

```
CREATE TABLE IF NOT EXISTS mycars(
        Name STRING,
        Miles_per_Gallon INT,
        Cylinders INT,
        Displacement INT,
        Horsepower INT,
        Weight_in_lbs INT,
        Acceleration DECIMAL,
        Year DATE,
        Origin CHAR(1))
    COMMENT 'Data about cars from a public database'
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    STORED AS ORC;
```

Step #4

```
INSERT OVERWRITE TABLE mycars SELECT * FROM cars;
```

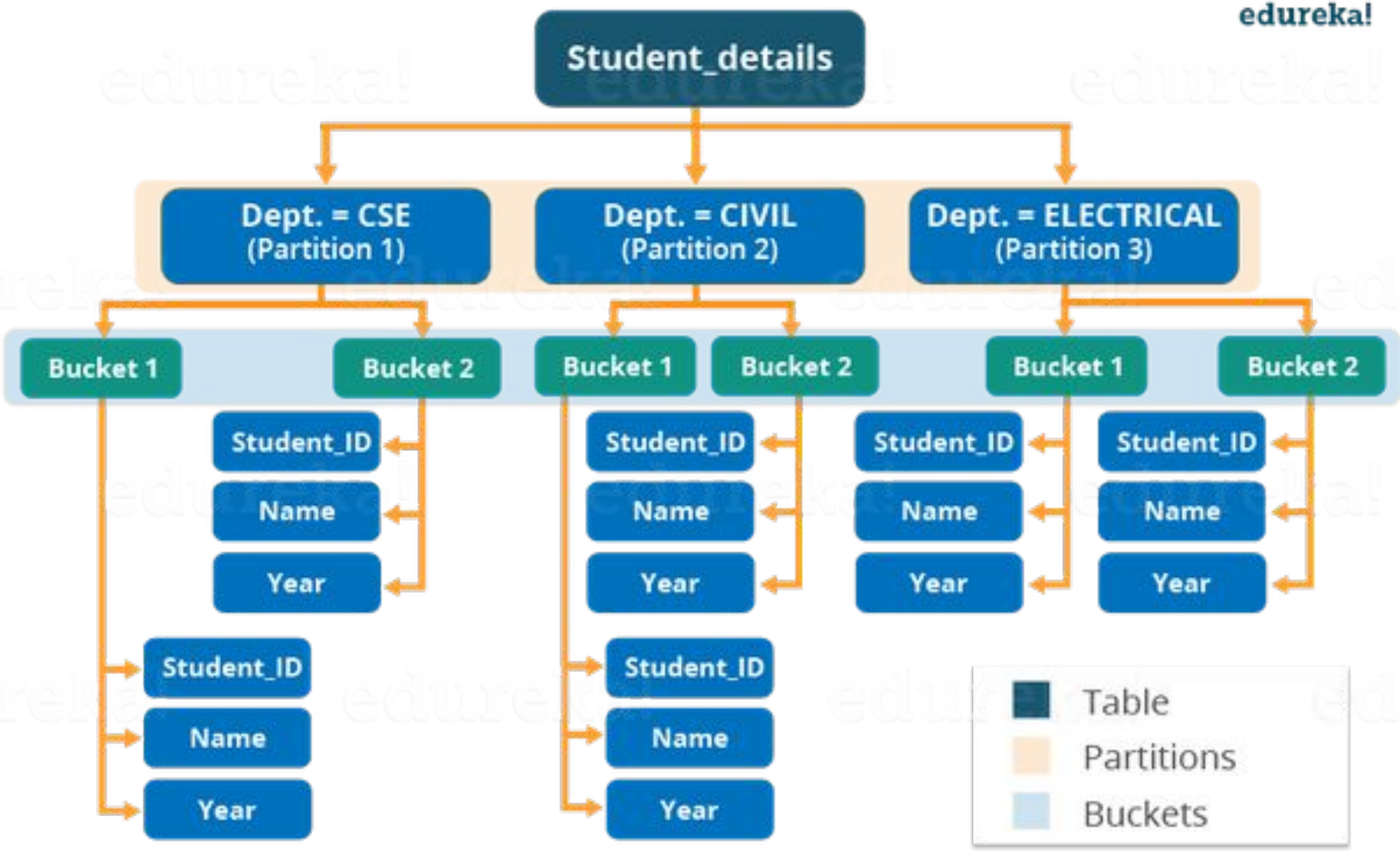
Hive Data Model



https://data-flair.training/blogs/hive-partitioning-vs-bucketing/

https://data-flair.training/blogs/hive-data-mode/

https://data-flair.training/blogs/apache-hive-partitions/



https://www.edureka.co/blog/hive-tutorial/

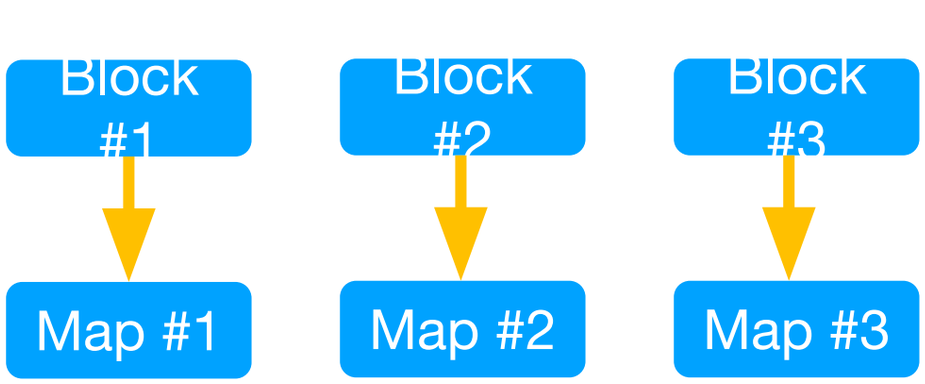
File

Split Ability

DICE ANALYTICS

File Combining in HDFS

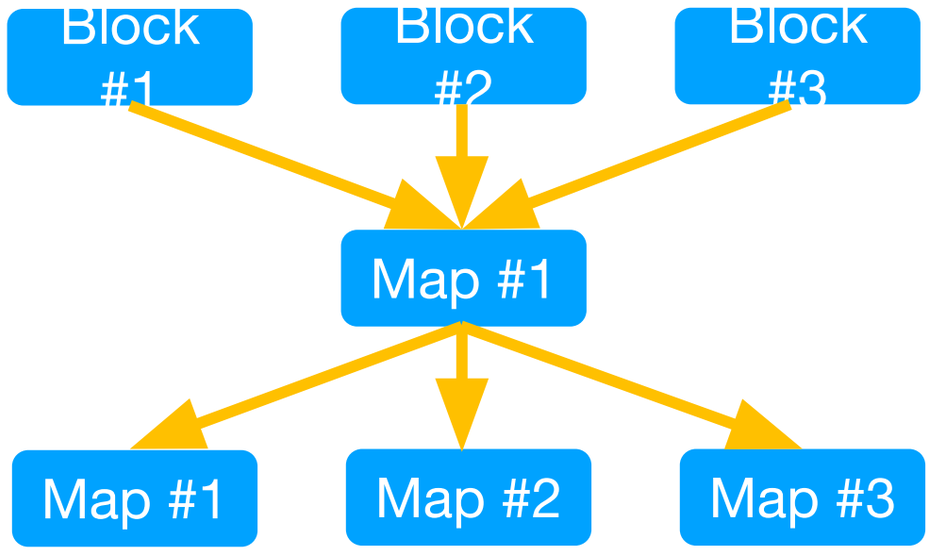
CSV



Mapreduce can achieve parallelism by default on CSV due to End of Line Characteristics helps to logically split the data incase of CSV

CSV is a Splittable Format

XML/JSON



Mapreduce cannot achieve parallelism by default on XML/JSON files as End of Line doesn't helps to logically split the XML/Json as XML data is splitted by start and end tags

Json/XML are non Splittable Format

MapReduce Works Well With Splittable File Formats

https://www.linkedin.com/pulse/hadoop-file-formats-chanchal-singh/

https://www.oreilly.com/library/view/hadoop-application-architectures/9781491910313/ch01.html

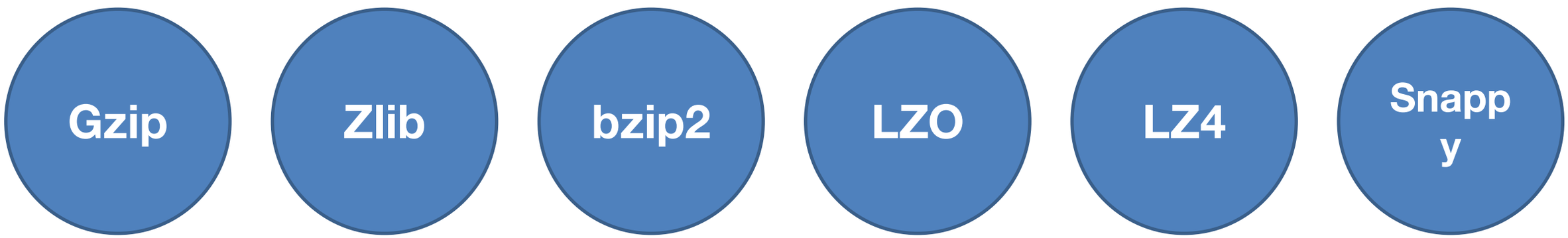
Apache Hive

Block Compression and File Formats

HDFS Compression & Hive File Formats

DICE ANALYTICS

HDFS Compression

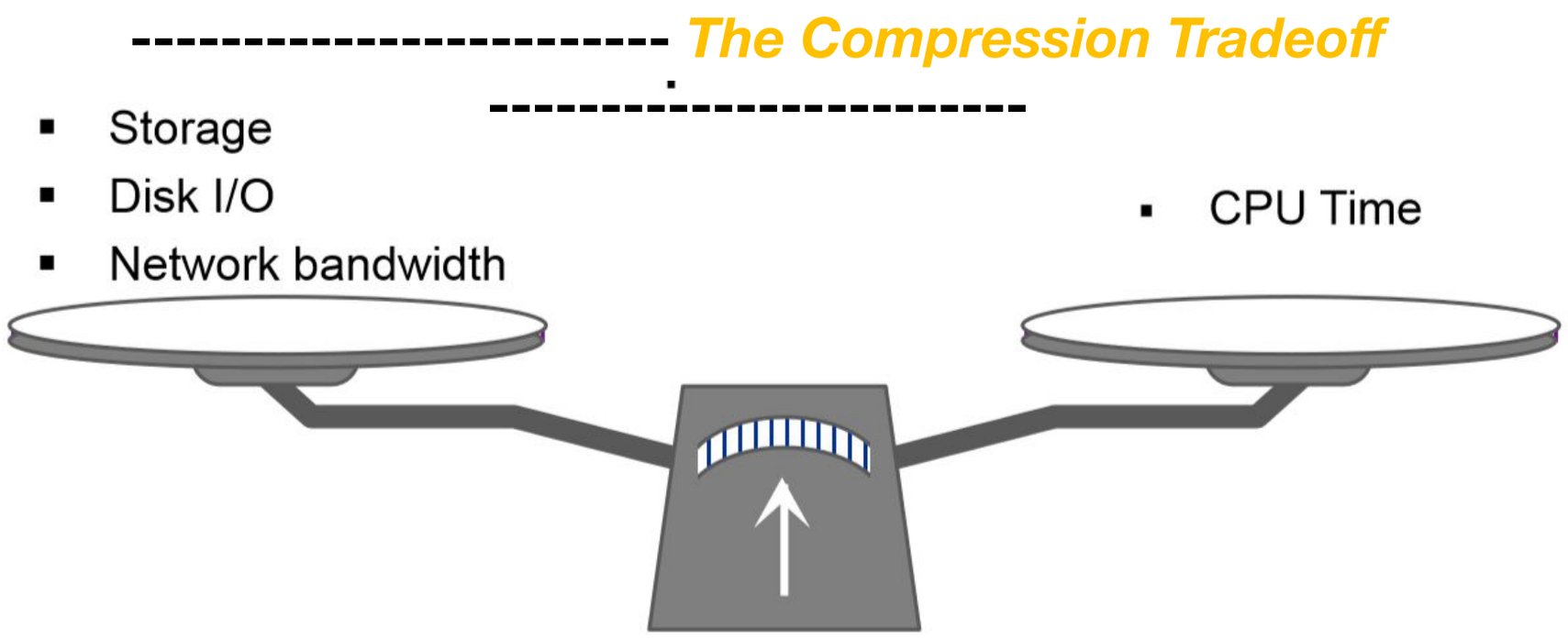


https://www.linkedin.com/pulse/data-compression-hadoop-framework-nitendra-gautam/

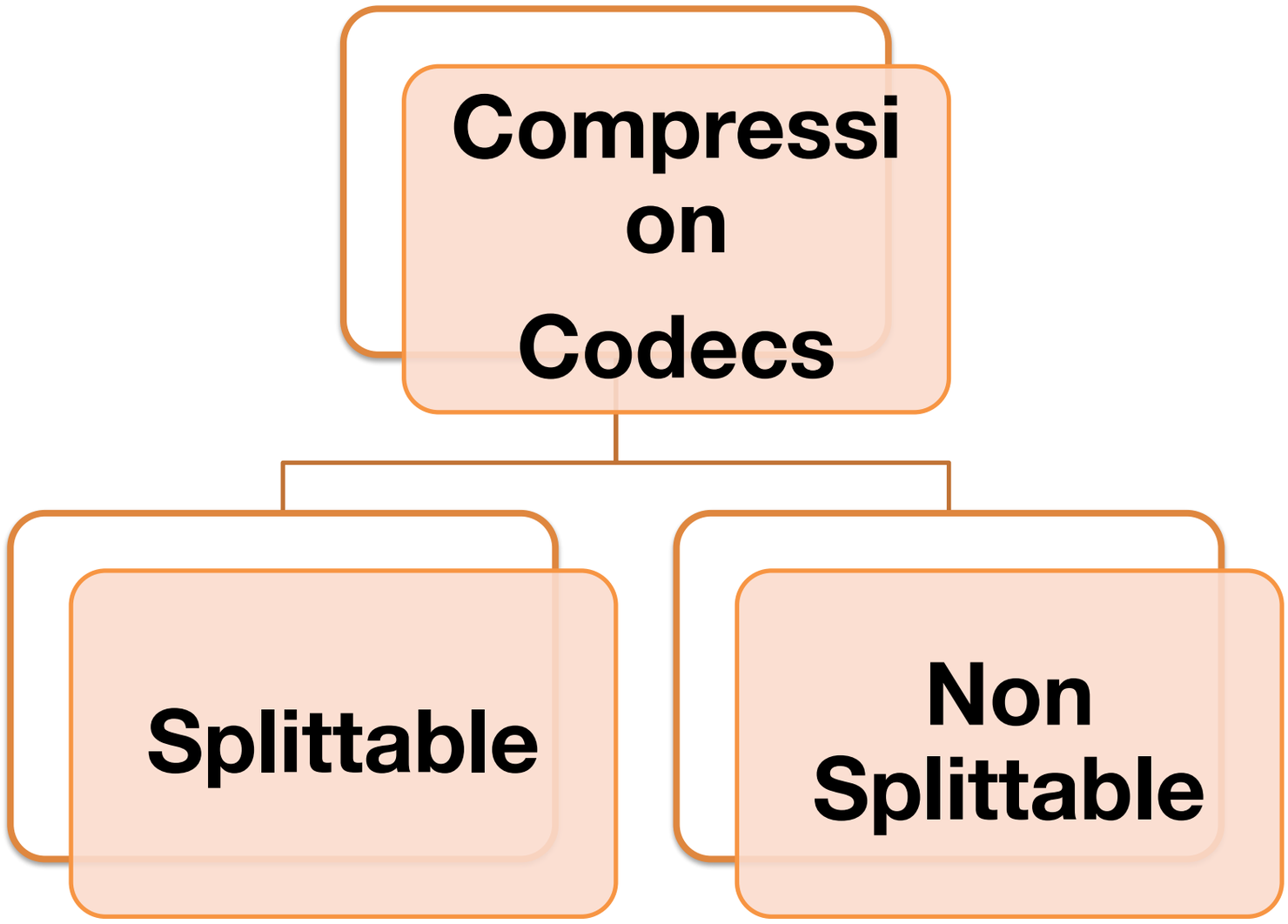
DICE ANALYTICS

Compression Tradeoffs in Hadoop

- **Mapreduce jobs are highly I/O Intensive**
- **Compression helps you to reduce the I/0 Intensity**
- **Compression helps you to reduce network traffic utilization**
- **But data must be uncompressed before processing**
- **And data un-compression comes at the Cost of CPU**
- **Compression also helps you to save the HDFS space**



https://www.cloudera.com/documentation/enterprise/5-3-x/topics/admin_data_compression_performance.htm



Codec	File Extension	Splittable?	Degree of Compression	Compression Speed
Gzip	.gz	No	Medium	Medium
Bzip2	.bz2	Yes	High	Slow
Snappy	.snappy	No	Medium	Fast
LZO	.lzo	No	Medium	Fast

http://comphadoop.weebly.com/

Load A Compressed File in Hive

http://www.hadooplessons.info/2017/07/loading-compressed-data-into-hive-table.html

http://hadoopmreduce.blogspot.com/2015/10/loading-tar-file-gzip-or-bzip2-into.html

Load a gzip File in Hive

1. **Create CSV File**
2. **Zip it as gzip format**
3. **Run following command**
   gzip filename

4. Hive automatically detects the compression extension

DICE ANALYTICS

Enable Hive for The Compression

DICE ANALYTICS

Block Compression

- Compression in hive is done in blocks
- Each block is compressed by the specified codec and stored
- Available codecs are LZO, gzip and snappy
- To enable compression:

-set hive.exec.compress.output=true;

-set mapred.output.compression=true;

-set mapred.output.compression.codec= org.apache.hadoop.io.compress.SnappyCodec;

http://upcouture.com/modules/ph_simpleblog/featured/15.jpg

DICE ANALYTICS

```
CREATE TABLE testsnappy_orc
STORED AS ORC
TBLPROPERTIES("orc.compress"="snappy")
```

DICE ANALYTICS

Supported Compressions in Hive/HDFS

https://acadgild.com/blog/hive-compression-codecs

DICE ANALYTICS

File Formats in Hive/HDFS

DICE ANALYTICS

Row Base File Formats

Vs

Column Base File Formats

DICE ANALYTICS

Column Format

Countries	Russia	Canada	China	USA	Brazil	Australia	India
Territory, km2	17,098,242	9,984,670	9,706,961	9,629,091	8,515,767	7,692,024	3,287,590
Land, km2	16,377,742	9,093,507	9,569,901	9,158,960	8,460,415	7,682,300	2,973,193
Water, km2	720,500.00	891,163.00	137,060.00	470,131.00	55,352.00	58,920.00	314,070.00
Water, %	4.21	8.93	1.41	2.23	0.65	0.76	9.55

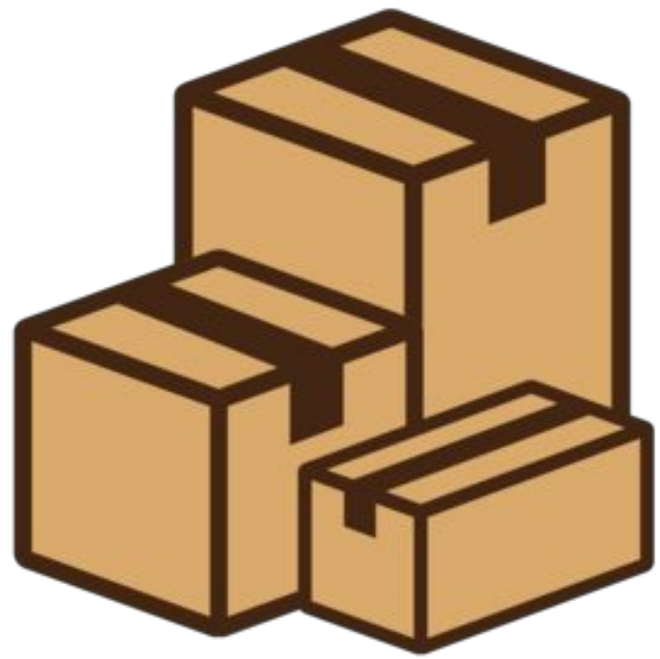
Countries	Territory, km2	Land, km2	Water, km2	Water, %
Russia	17,098,242	16,377,742	720,500	4.21
Canada	9,984,670	9,093,507	891,163	8.93
China	9,706,961	9,569,901	137,060	1.41
USA	9,629,091	9,158,960	470,131	2.23
Brazil	8,515,767	8,460,415	55,352	0.65
Australia	7,692,024	7,682,300	58,920	0.76
India	3,287,590	2,973,193	314,070	9.55
Argentina	2,780,400	2,736,690	43,710	1.57
Kazakhstan	2,724,900	2,699,700	25,200	0.92
Algeria	2,381,741	2,381,741	0	0.00
Democratic I	2,344,858	2,267,048	77,810	3.32
Denmark	2,166,086	2,166,086	0	0

Row Format

DICE ANALYTICS

Storage formats in HDFS

- Text
- JSON
- Sequence files
- RC file format
- Avro
- Parquet
- ORC file format



http://blog.trustpilot.com/blog/2015/11/5/storage-vertical

DICE ANALYTICS

RC File

- Row Columnar File Format
- Supports block compression
- Good compression rate
- good for querying
- uses more memory and computation while writing
- slow write
- Can be used by STORED AS RCFILE clause in the CREATE TABLE statement

RC File

	C1	C2	C3	C4	C5	C6
Row 1	1	-14	30	85	15	185
Row 2	2	-13	31	86	16	186
Row 3	3	-12	32	87	17	187
Row 4	4	-11	33	88	18	188
Row 5	5	-10	34	89	19	189
Row 6	6	-9	35	90	20	190
Row 7	7	-8	36	91	21	191
Row 8	8	-7	37	92	22	192
Row 9	9	-6	38	93	23	193
Row 10	10	-5	39	94	24	194
Row 11	11	-4	40	95	25	195
Row 12	12	-3	41	96	26	196
Row 13	13	-2	42	97	27	197
Row 14	14	-1	43	98	28	198
Row 15	15	0	44	99	29	199
Row 16	16	1	45	100	30	200

	C1	C2	C3	C4	C5	C6	
Row 1	1	-14	30	85	15	185	Group 1
Row 2	2	-13	31	86	16	186	
Row 3	3	-12	32	87	17	187	
Row 4	4	-11	33	88	18	188	

	C1	C2	C3	C4	C5	C6	
Row 5	5	-10	34	89	19	189	Group 2
Row 6	6	-9	35	90	20	190	
Row 7	7	-8	36	91	21	191	
Row 8	8	-7	37	92	22	192	

	C1	C2	C3	C4	C5	C6	
Row 9	9	-6	38	93	23	193	Group 3
Row 10	10	-5	39	94	24	194	
Row 11	11	-4	40	95	25	195	
Row 12	12	-3	41	96	26	196	

	C1	C2	C3	C4	C5	C6	
Row 13	13	-2	42	97	27	197	Group 4
Row 14	14	-1	43	98	28	198	
Row 15	15	0	44	99	29	199	
Row 16	16	1	45	100	30	200	

DICE ANALYTICS

	C1	C2	C3	C4	C5	C6	
Row 1	1	-14	30	85	15	185	Group 1
Row 2	2	-13	31	86	16	186	
Row 3	3	-12	32	87	17	187	
Row 4	4	-11	33	88	18	188	

	C1	C2	C3	C4	C5	C6	
Row 5	5	-10	34	89	19	189	Group 2
Row 6	6	-9	35	90	20	190	
Row 7	7	-8	36	91	21	191	
Row 8	8	-7	37	92	22	192	

	C1	C2	C3	C4	C5	C6	
Row 9	9	-6	38	93	23	193	Group 3
Row 10	10	-5	39	94	24	194	
Row 11	11	-4	40	95	25	195	
Row 12	12	-3	41	96	26	196	

	C1	C2	C3	C4	C5	C6	
Row 13	13	-2	42	97	27	197	Group 4
Row 14	14	-1	43	98	28	198	
Row 15	15	0	44	99	29	199	
Row 16	16	1	45	100	30	200	

	Row 1	Row 2	Row 3	Row 4
C1	1	2	3	4
C2	-14	-13	-12	-11
C3	30	31	32	33
C4	85	86	87	88
C5	15	16	17	18
C6	185	186	187	188
Group 1				

	Row 5	Row 6	Row 7	Row 8
C1	5	6	7	8
C2	-10	-9	-8	-7
C3	34	35	36	37
C4	89	90	91	92
C5	19	20	21	22
C6	189	190	191	192
Group 2				

	Row 9	Row 10	Row 11	Row 12
C1	9	10	11	12
C2	-6	-5	-4	-3
C3	38	39	40	41
C4	93	94	95	96
C5	23	24	25	26
C6	193	194	195	196
Group 3				

	Row 13	Row 14	Row 15	Row 16
C1	13	14	15	16
C2	-2	-1	0	1
C3	42	43	44	45
C4	97	98	99	100
C5	27	28	29	30
C6	197	198	199	200
Group 4				

ORC File Format

- Optimized Row Columnar File Format
- supports block compression
- fast querying
- good compression rate
- slow write (faster than RC)
- Can be used by STORED AS ORCFILE clause in the CREATE
  TABLE statement

https://orc.apache.org/

Sequence Files

- sequence files store data in a binary format
- supports block compression
- Used mainly for intermediate storage between jobs due to complexity
- **Not recommended for Hive**
- **As it stores compete row a Value**

- Can be used by STORED AS SEQUENCEFILE clause in the CREATE TABLE statement

DICE ANALYTICS

Avro

- Store metadata with the data
- Allow to load the schema independently
- Avro files are:
- splittable
- supports block compression
- has good tool support in hadoop ecosystem

https://seouniv.com/images/apache-avro.png

Parquet

- Columnar format similar to RC and ORC
- good compression rate
- fast querying
- slow write
- limited schema changing (new columns can be added only at the end of a structure)
- Can be used by STORED AS PARQUET clause in the CREATE TABLE statement

https://dribbble.com/shots/1625979-Parquet-logo

DICE ANALYTICS

Storage formats in HDFS

Non Columnar Formats

Text File
JSON File
Sequence File
Avro Format

Columnar Formats

RC File
ORC File
Parquet File

DICE ANALYTICS

Aspects to Keep in Mind while Selecting File Format

- **Schema Evaluation / Evolution**
- **Compression**
- **Splittability**
- **Data Processing**
  (Read/Write Performance, Partial Read Performance)

DICE ANALYTICS

Text Files: CSV, JSON, XML

Good for Begging But Not Good For Real Life

DICE ANALYTICS

CSV, XML, JSON, Avro, Parquet, ORC, RCFile, Sequence File

Gzip, LZIP, Snappy

Compression & File Formats are two separate concepts

DICE ANALYTICS

When to Use Which File Format

https://nxtgen.com/hadoop-file-formats-when-and-what-to-use

Hive Table Optimizations

https://www.spotx.tv/resources/blog/developer-blog/how-to-build-optimal-hive-tables-using-orc-partitions-and-metastore-statistics/

Hive Future

What’s Stringer and Stringer.next?

DICE ANALYTICS

Low Latency Analytical Processing

https://hortonworks.com/blog/top-5-performance-boosters-with-apache-hive-llap/