Data Management

CT051-3-M

Topic 8 – HIVE

Topic & Structure of Lesson

- What is Hive?
- Hadoop Framework
- Hadoop’s Architecture
- Hadoop in the Wild
- Data warehouse to Hadoop

About Hive

Store and Query all
Data in Hive
Operational
/ MPP
Weblog
Mobile
Sensor

Use Existing SQL Tools
and Existing SQL Processes

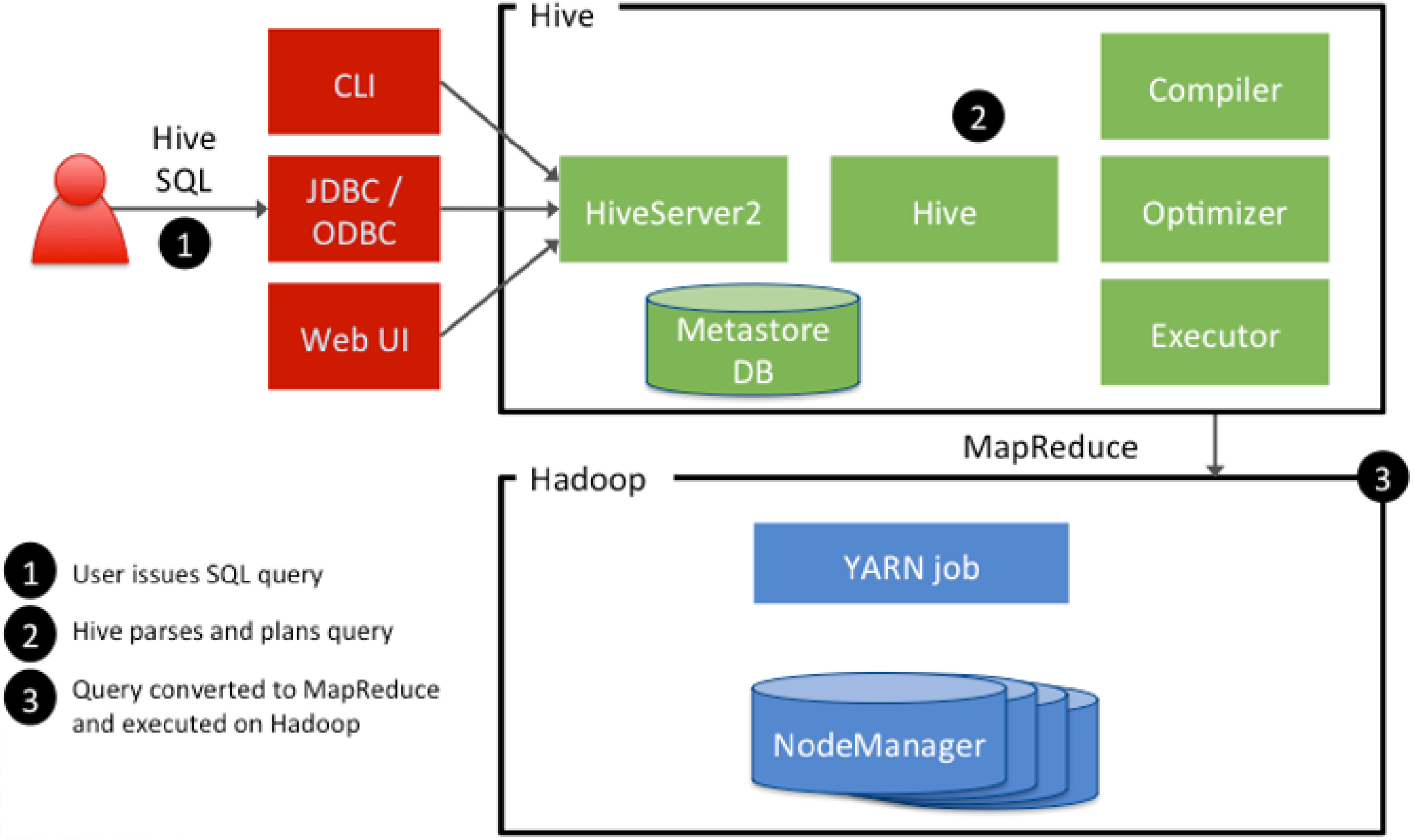
About Hive – cont.

- It is a data warehouse system for Hadoop
- It maintains metadata information about your big data stored on HDFS
- It treats your big data as tables
- It performs SQL-like operations on the data using a scripting language called **HiveQL**

Hive's Alignment with SQL

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

Hive Architecture



1 User issues SQL query

2 Hive parses and plans query

3 Query converted to MapReduce and executed on Hadoop

Submitting Hive Queries

- Hive CLI
  - Traditional Hive “thick” client

  -
    ```
    $ hive
    hive>
    ```

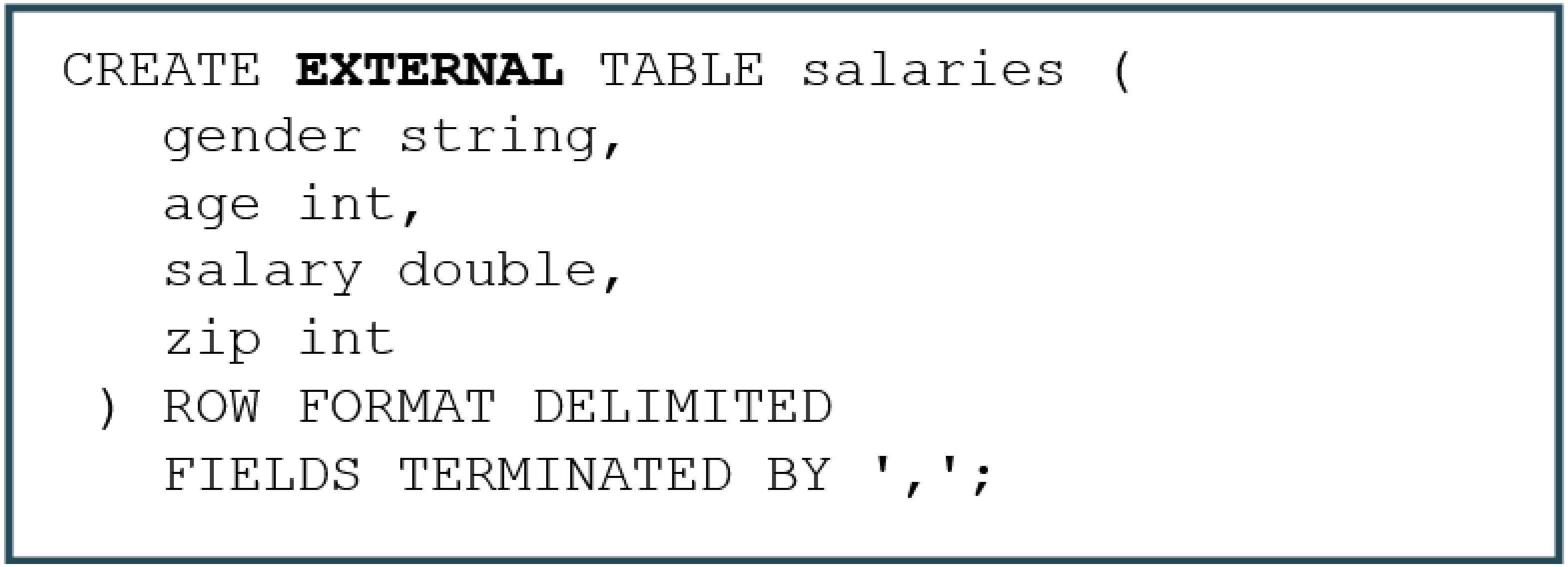
- Beeline
  - A new command-line client that connects to a HiveServer2 instance

  -
    ```
    $ beeline -u url -n username -p password
    beeline>
    ```

Defining a Hive-Managed Table

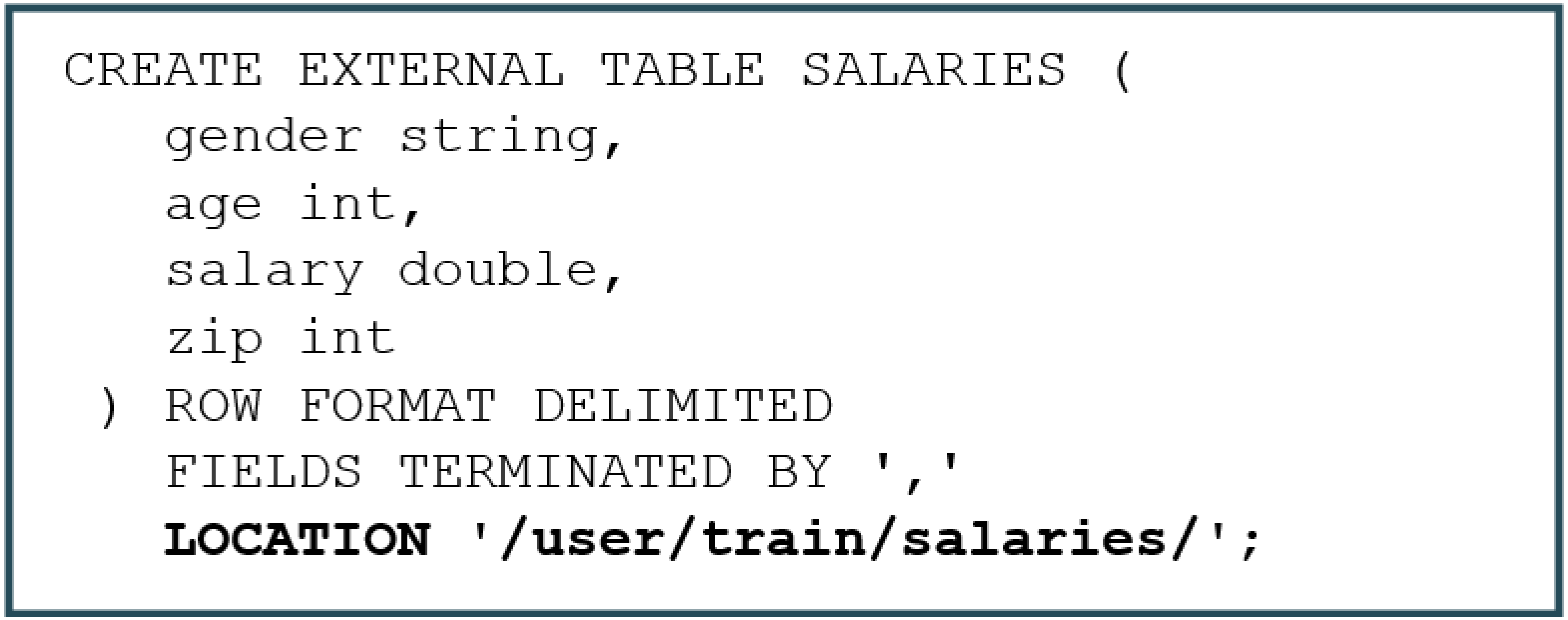
```
CREATE TABLE customer (
          customerID INT,
          firstName STRING,
          lastName STRING,
          birthday TIMESTAMP
    ) ROW FORMAT DELIMITED
      FIELDS TERMINATED BY ',';
```

Defining an External Table



```
CREATE EXTERNAL TABLE salaries (
   gender string,
   age int,
   salary double,
   zip int
 ) ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ',';
```

Defining a Table LOCATION



```
CREATE EXTERNAL TABLE SALARIES (
   gender string,
   age int,
   salary double,
   zip int
 ) ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ','
   LOCATION '/user/train/salaries/';
```

Loading Data into Hive

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

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

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

Performing Queries

```
SELECT * FROM customers;
```

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

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

Lab: Understanding Hive Tables

Hive Partitions

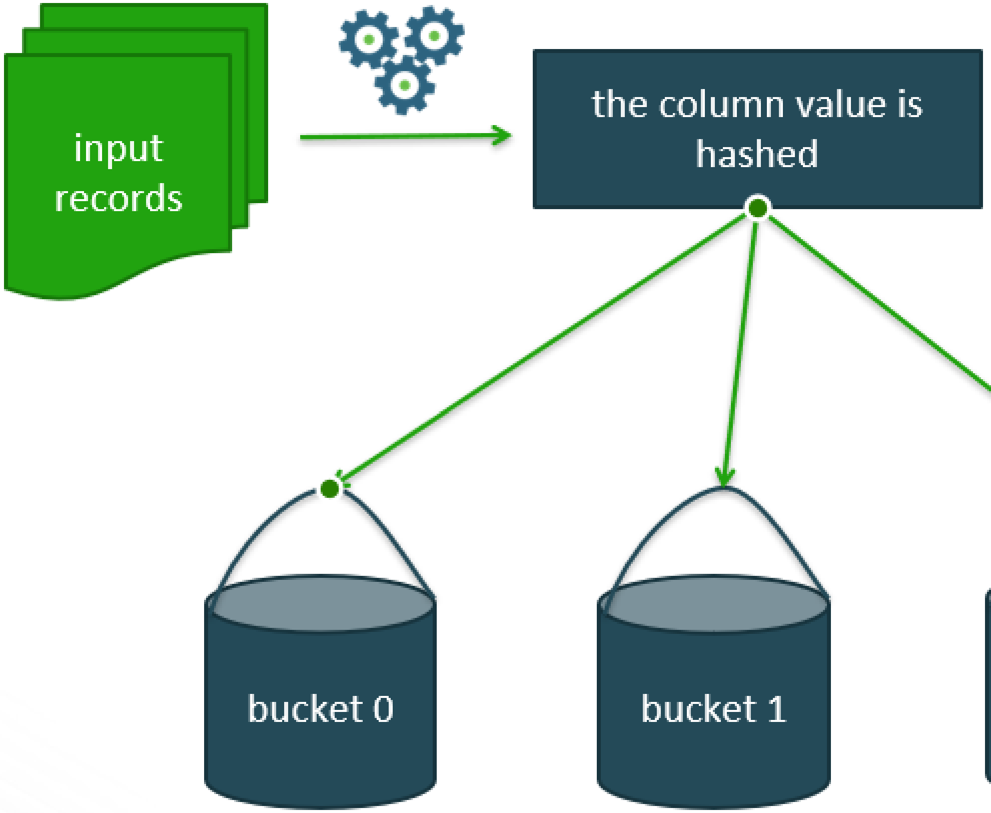
- Use the **partitioned by** clause to define a partition when creating a table:

```
create table employees (id int, name string, salary double)
partitioned by (dept string);
```

- Subfolders are created based on the partition values:

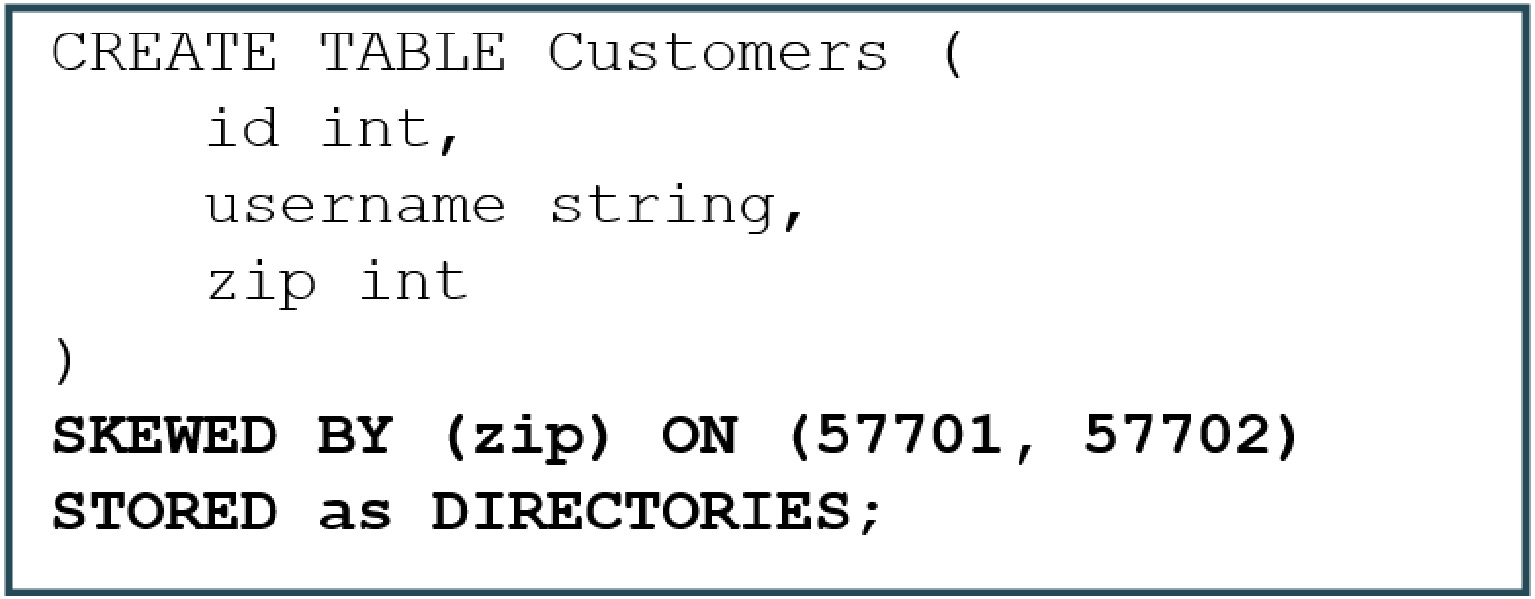
```
/apps/hive/warehouse/employees
   /dept=hr/
   /dept=support/
   /dept=engineering/
   /dept=training/
```

Hive Buckets



The table's data is divided up into buckets

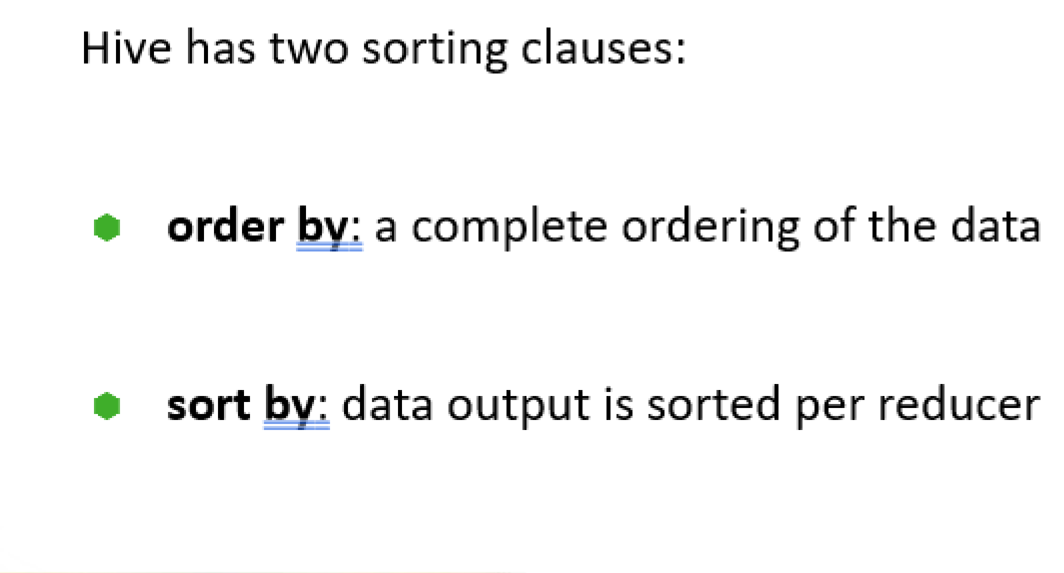
Skewed Tables



```
CREATE TABLE Customers (
    id int,
    username string,
    zip int
)
SKEWED BY (zip) ON (57701, 57702)
STORED as DIRECTORIES;
```

Understanding Partitions and Skew

- Sorting Data



Hive has two sorting clauses:

- **order by**: a complete ordering of the data
- **sort by**: data output is sorted per reducer

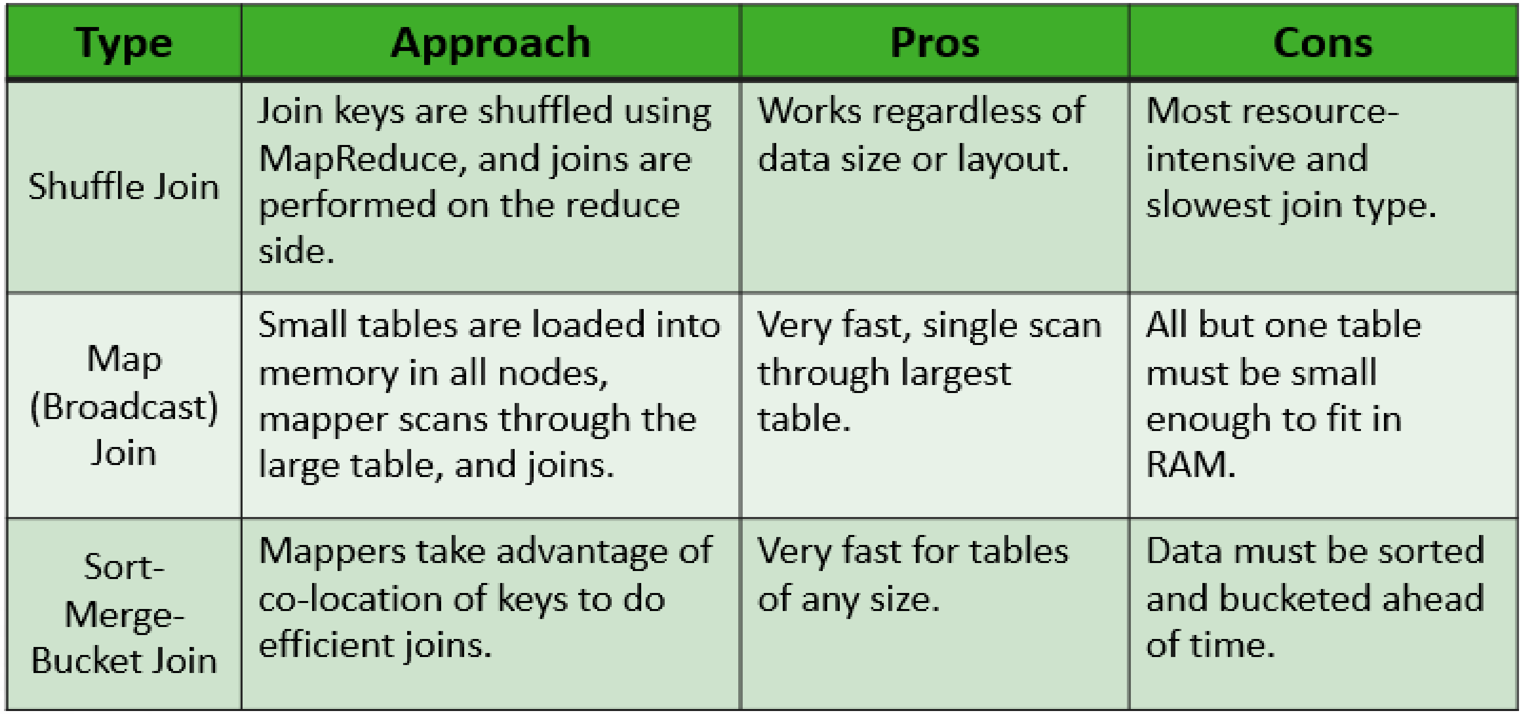
Using Distribute By

```
insert overwrite table mytable
    select gender,age,salary
    from salaries
    distribute by age;
```

```
insert overwrite table mytable
    select gender,age,salary
    from salaries
    distribute by age
    sort by age;
```

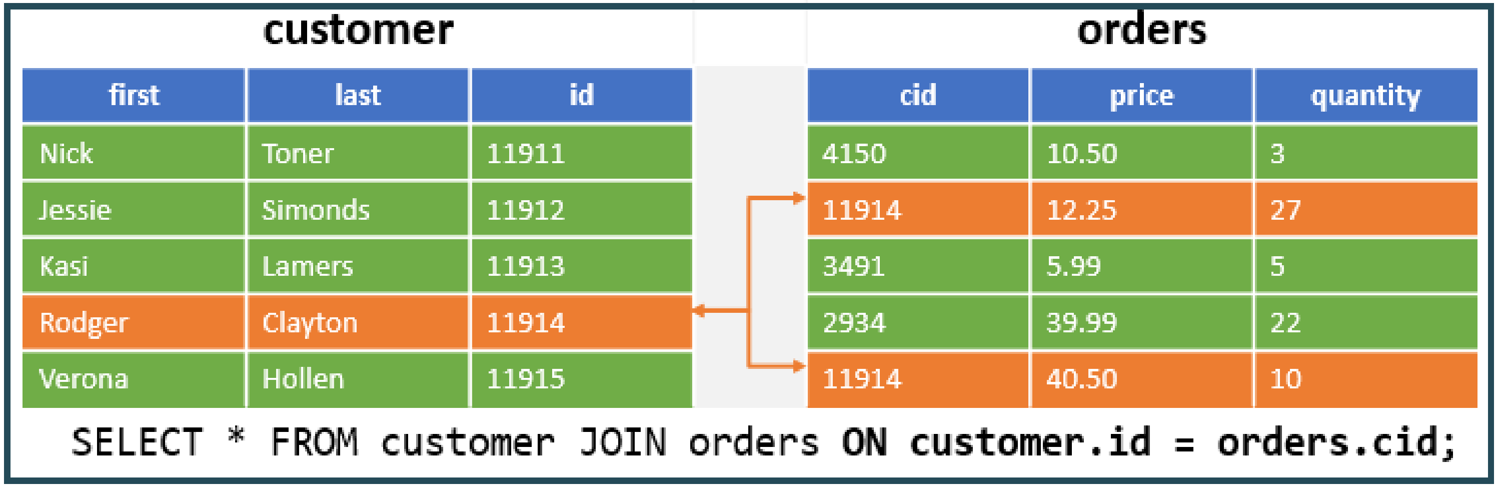
Analyzing Big Data with Hive

Hive Join Strategies



Type	Approach	Pros	Cons
Shuffle Join	Join keys are shuffled using MapReduce, and joins are performed on the reduce side.	Works regardless of data size or layout.	Most resource-intensive and slowest join type.
Map (Broadcast) Join	Small tables are loaded into memory in all nodes, mapper scans through the large table, and joins.	Very fast, single scan through largest table.	All but one table must be small enough to fit in RAM.
Sort-Merge-Bucket Join	Mappers take advantage of co-location of keys to do efficient joins.	Very fast for tables of any size.	Data must be sorted and bucketed ahead of time.

Shuffle Joins



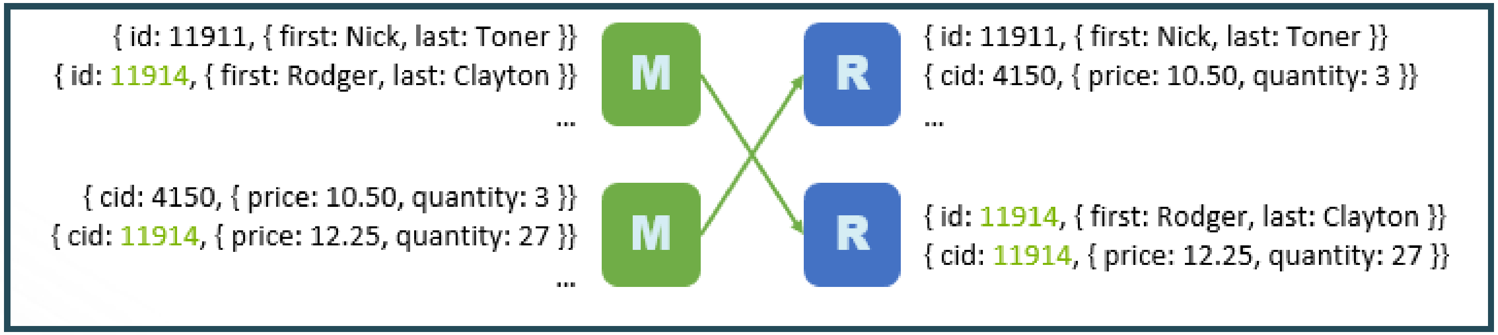
customer

first	last	id
Nick	Toner	11911
Jessie	Simonds	11912
Kasi	Lamers	11913
Rodger	Clayton	11914
Verona	Hollen	11915

orders

cid	price	quantity
4150	10.50	3
11914	12.25	27
3491	5.99	5
2934	39.99	22
11914	40.50	10

```
SELECT * FROM customer JOIN orders ON customer.id = orders.cid;
```



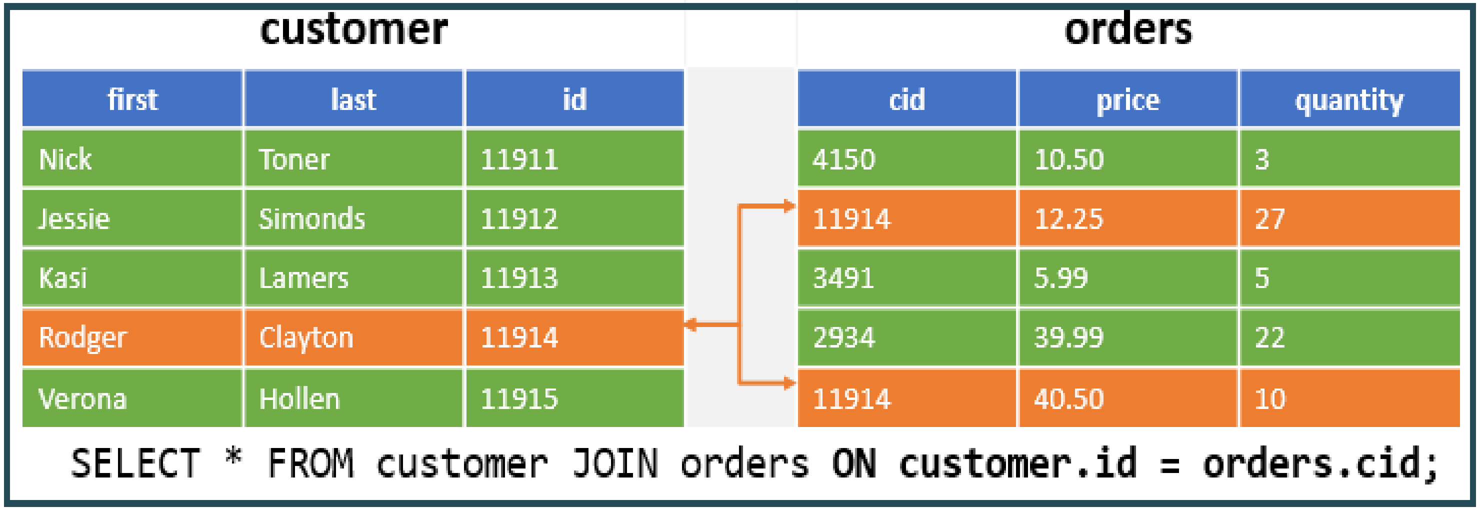
{ id: 11911, { first: Nick, last: Toner }}
{ id: 11914, { first: Rodger, last: Clayton }}
...

{ cid: 4150, { price: 10.50, quantity: 3 }}
{ cid: 11914, { price: 12.25, quantity: 27 }}
...

{ id: 11911, { first: Nick, last: Toner }}
{ cid: 4150, { price: 10.50, quantity: 3 }}
...

{ id: 11914, { first: Rodger, last: Clayton }}
{ cid: 11914, { price: 12.25, quantity: 27 }}

Map (Broadcast) Joins



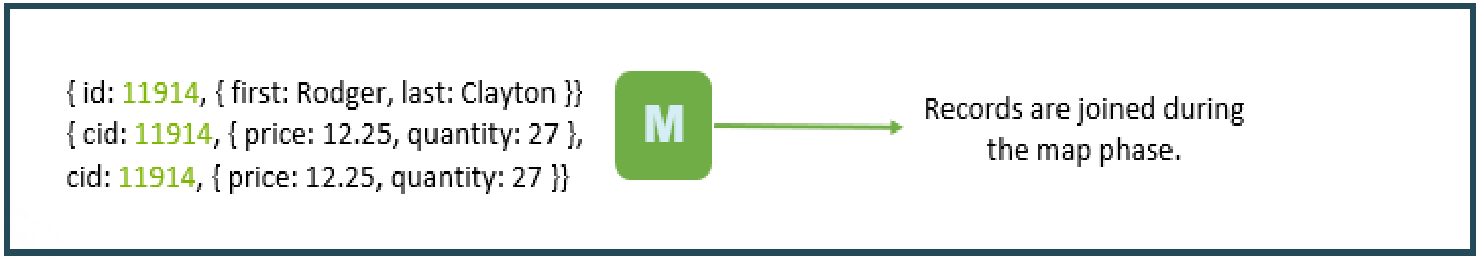
customer

first	last	id
Nick	Toner	11911
Jessie	Simonds	11912
Kasi	Lamers	11913
Rodger	Clayton	11914
Verona	Hollen	11915

orders

cid	price	quantity
4150	10.50	3
11914	12.25	27
3491	5.99	5
2934	39.99	22
11914	40.50	10

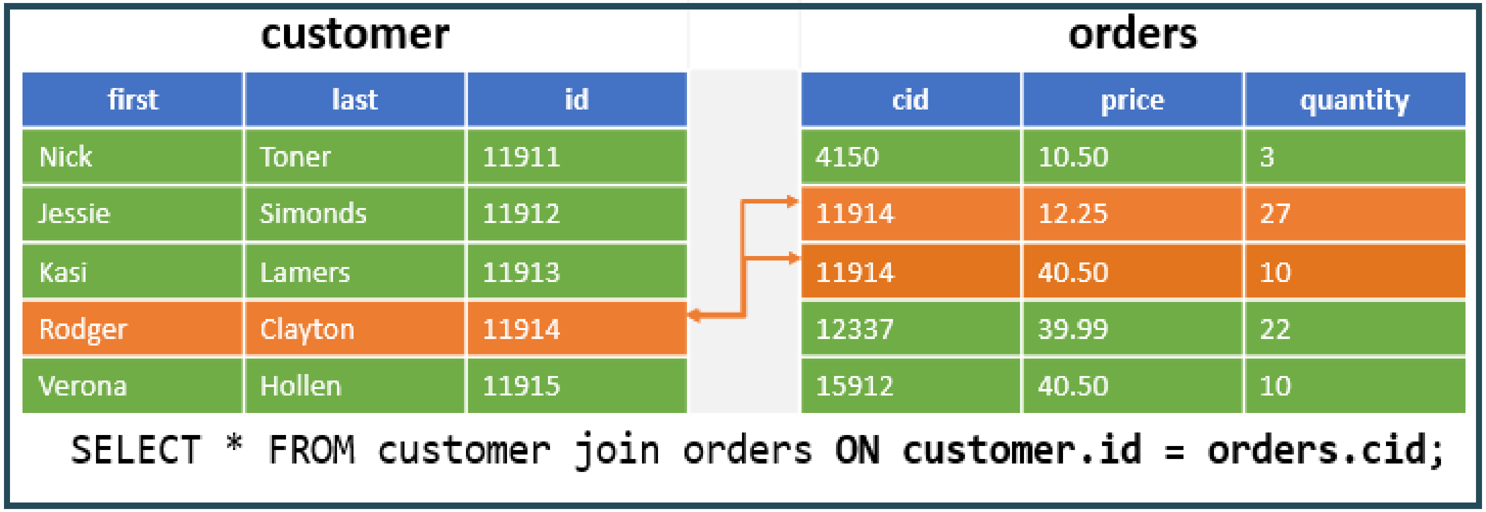
```
SELECT * FROM customer JOIN orders ON customer.id = orders.cid;
```



{ id: 11914, { first: Rodger, last: Clayton }}
{ cid: 11914, { price: 12.25, quantity: 27 },
cid: 11914, { price: 12.25, quantity: 27 }}

Records are joined during the map phase.

Sort-Merge-Bucket Joins



customer

first	last	id
Nick	Toner	11911
Jessie	Simonds	11912
Kasi	Lamers	11913
Rodger	Clayton	11914
Verona	Hollen	11915

orders

cid	price	quantity
4150	10.50	3
11914	12.25	27
11914	40.50	10
12337	39.99	22
15912	40.50	10

```
SELECT * FROM customer join orders ON customer.id = orders.cid;
```

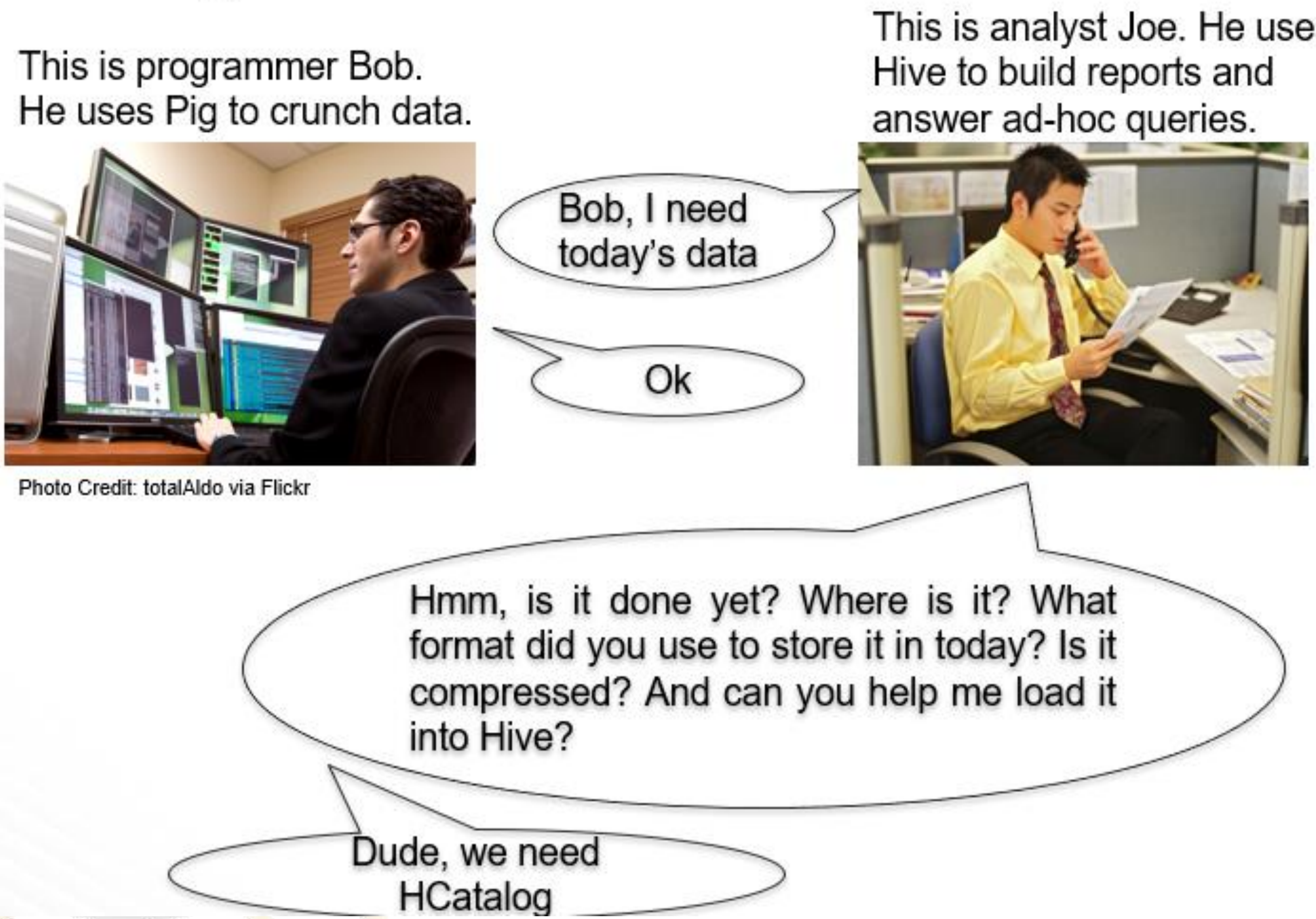
Distribute and sort by the most common join key.

```
CREATE TABLE orders (cid int, price float, quantity int)
CLUSTERED BY(cid) SORTED BY(cid) INTO 32 BUCKETS;


CREATE TABLE customer (id int, first string, last string)
CLUSTERED BY(id) SORTED BY(cid) INTO 32 BUCKETS;
```

Using HCatalog

About HCatalog



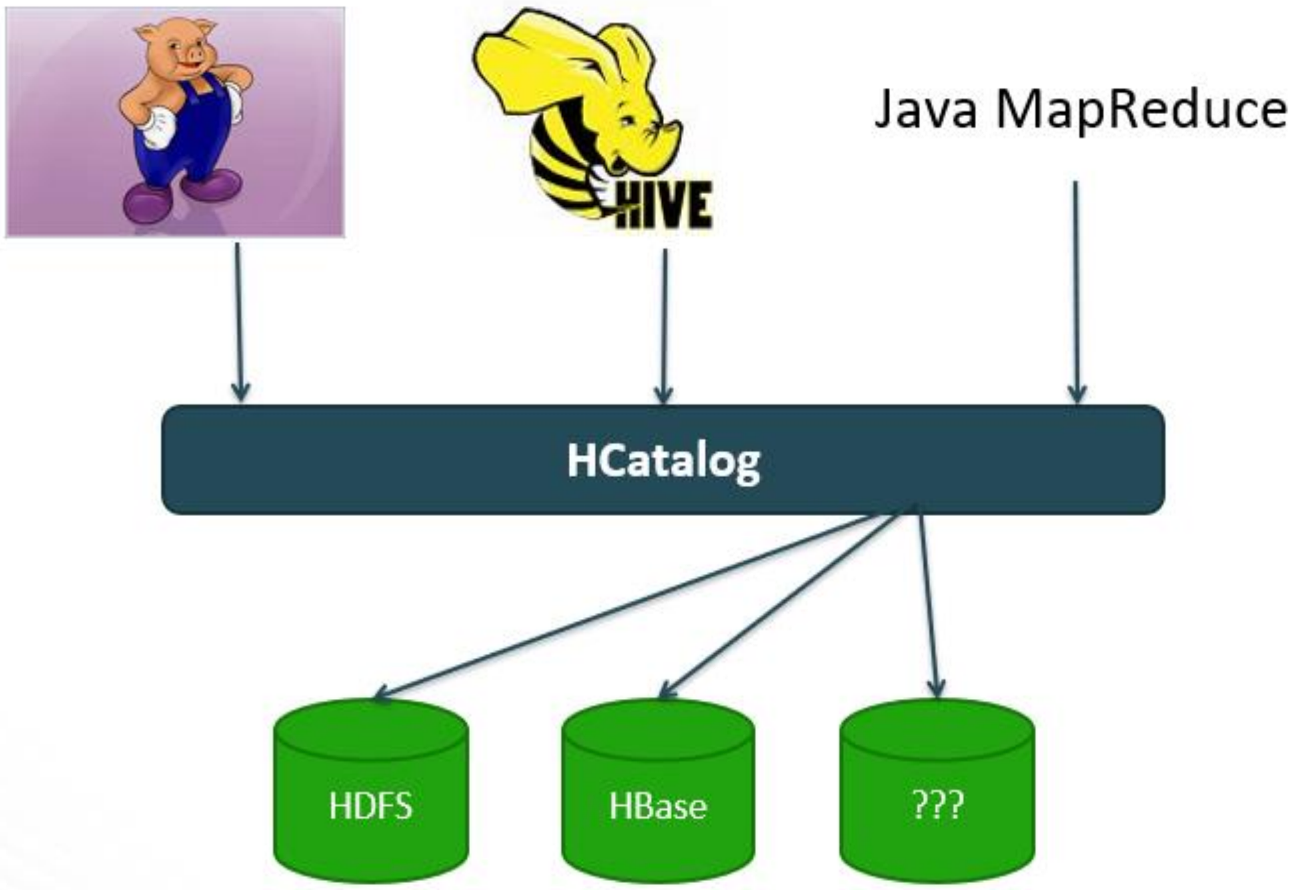
This is programmer Bob. He uses Pig to crunch data.

This is analyst Joe. He uses Hive to build reports and answer ad-hoc queries.

Photo Credit: totalAldo via Flickr

Bob, I need today's data
Ok
Hmm, is it done yet? Where is it? What format did you use to store it in today? Is it compressed? And can you help me load it into Hive?
Dude, we need HCatalog

HCatalog in the Ecosystem



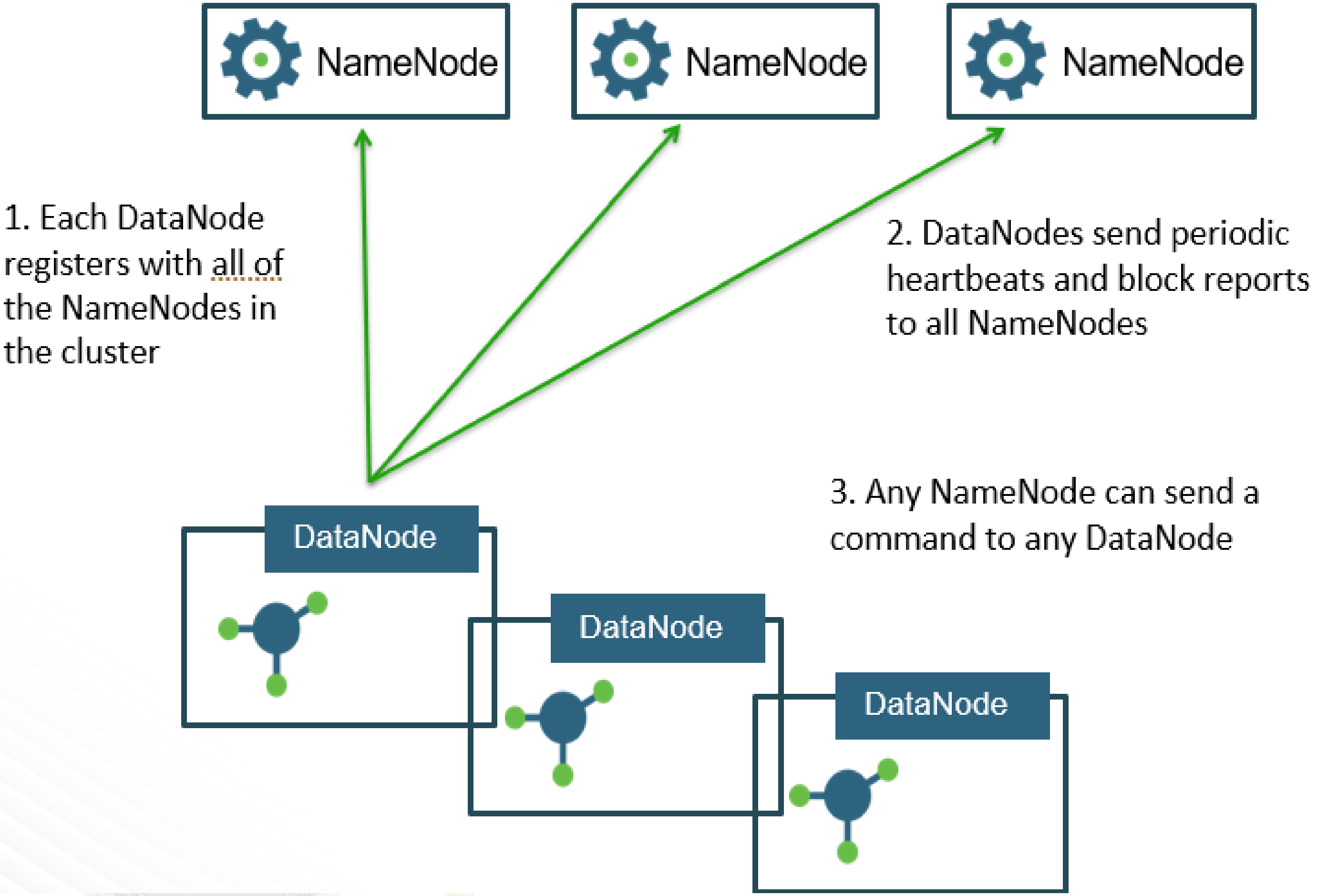
Hadoop 2 and YARN

About HDFS Federation

- According to Merriam-Webster's dictionary: a ***federation*** is an organization or group within which smaller divisions have some degree of internal autonomy

- **HDFS Federation** refers to the ability of NameNodes to work independently of each other

Multiple Federated NameNodes

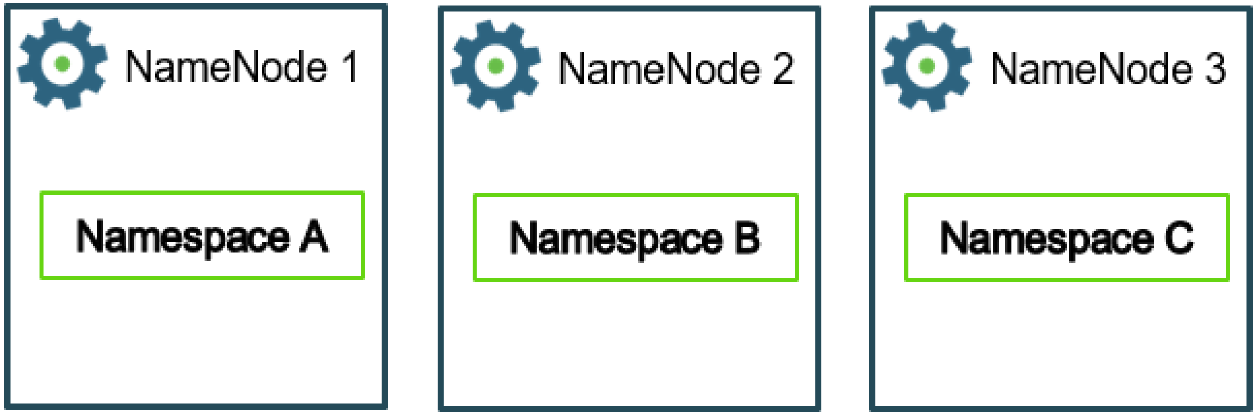


1. Each DataNode registers with all of the NameNodes in the cluster

2. DataNodes send periodic heartbeats and block reports to all NameNodes

3. Any NameNode can send a command to any DataNode

Multiple Namespaces



- Files and directories belong to a Namespace
- Prior versions of Hadoop only had a single Namespace
- Hadoop 2.x allows for multiple Namespaces
- A NameNode manages a single Namespace Volume

About YARN

YARN = Yet Another Resource Negotiator

YARN splits up the functionality of the JobTracker in Hadoop 1.x into two separate processes:

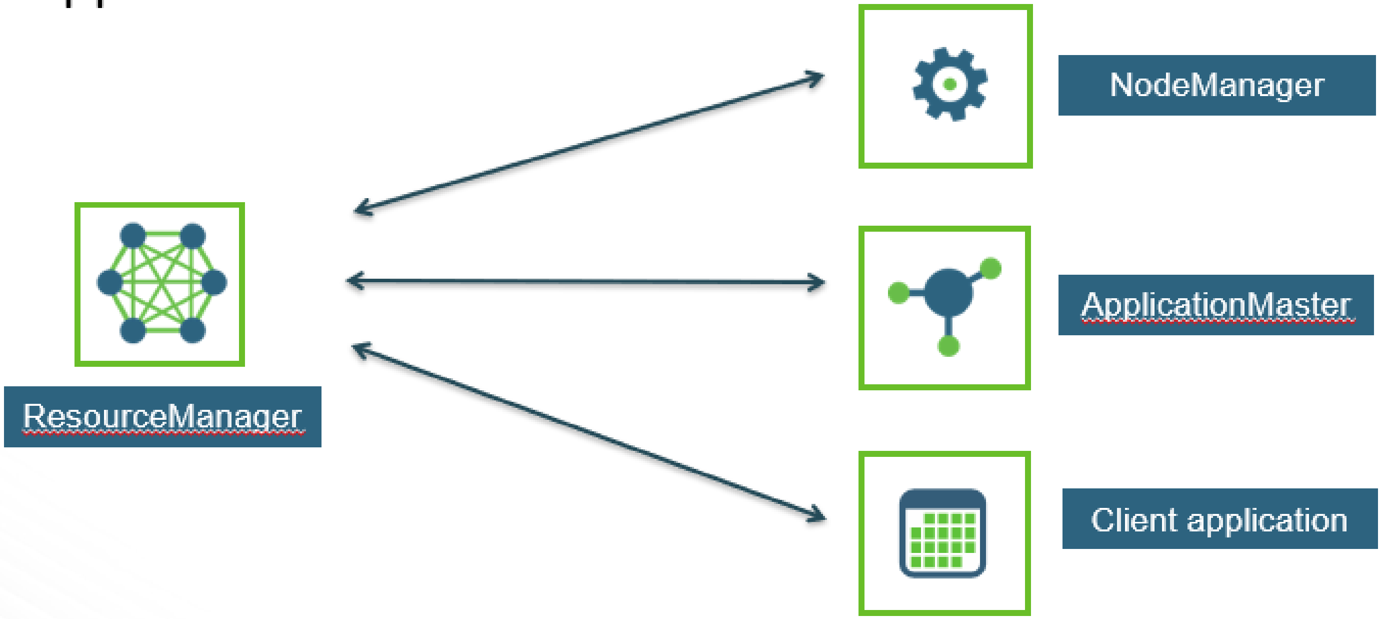
- **ResourceManager**: for allocating resources and scheduling applications
- **ApplicationMaster**: for executing applications and providing failover

Open-source YARN Use Cases

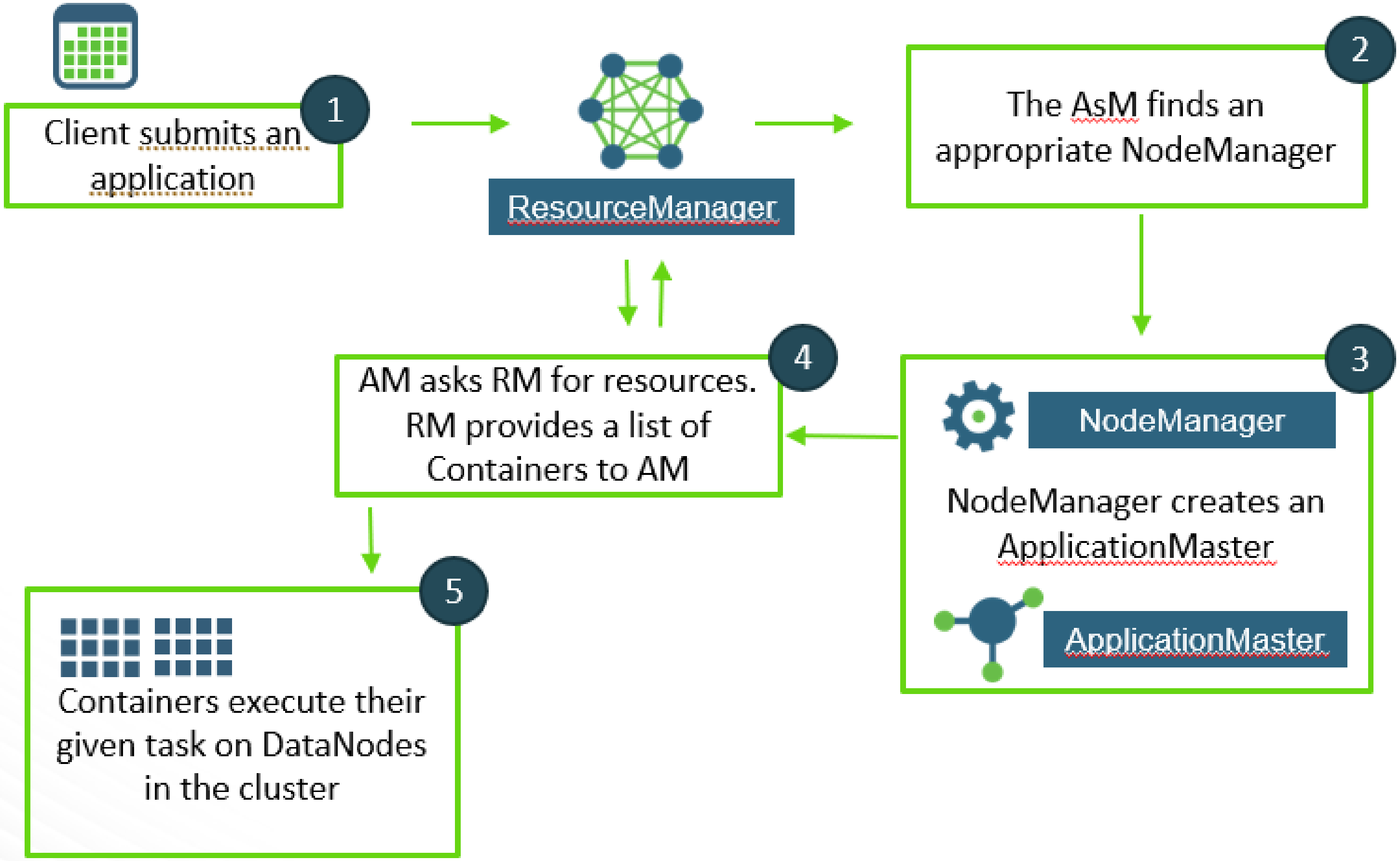
- **Tez:** improves the execution of MapReduce jobs
- **Slider:** deploy existing frameworks on YARN
- **Storm**: for real-time computing
- **Spark**: a MapReduce-like cluster computing framework designed for low-latency iterative jobs and interactive use from an interpreter
- **Apache Giraph**: a graph-processing platform

The Components of YARN

The **ResourceManager** communicates with the **NodeManagers**, **ApplicationMasters**, and **Client** applications



Lifecycle of a YARN Application



1 Client submits an application

ResourceManager

2 The AsM finds an appropriate NodeManager

3 NodeManager

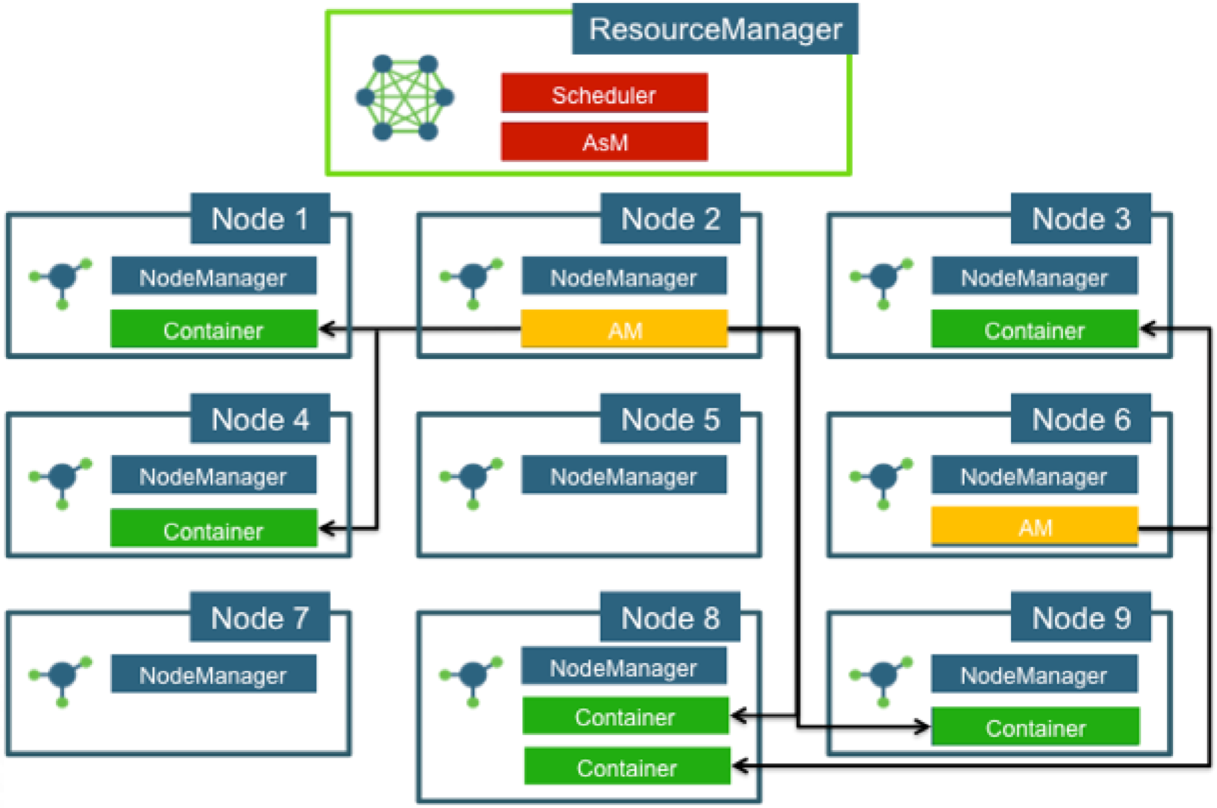
NodeManager creates an ApplicationMaster

ApplicationMaster

4 AM asks RM for resources. RM provides a list of Containers to AM

5 Containers execute their given task on DataNodes in the cluster

A Cluster View Example



Question & Answer Session

Q & A