Hadoop and Hadoop Ecosystem

Course Agenda

- Introduction to BigData
- Introduction to Apache Hadoop
- Components of ApacheHadoop
- Apache Hadoop Architecture
- Hadoop Ecosystem
- Apache Pig
- Apache Hive
- NoSQL and Apache HBase

Introduction to Big Data

Module Agenda

- What is Big Data?
- Why is Big Data Important?
- What is Big Data?
- Characteristics of Big Data
- How did data become so big?
- Why should you care about Big Data?
- Use Cases of Big Data Analysis
- What are possible options for analysing big data?

What is Big Data

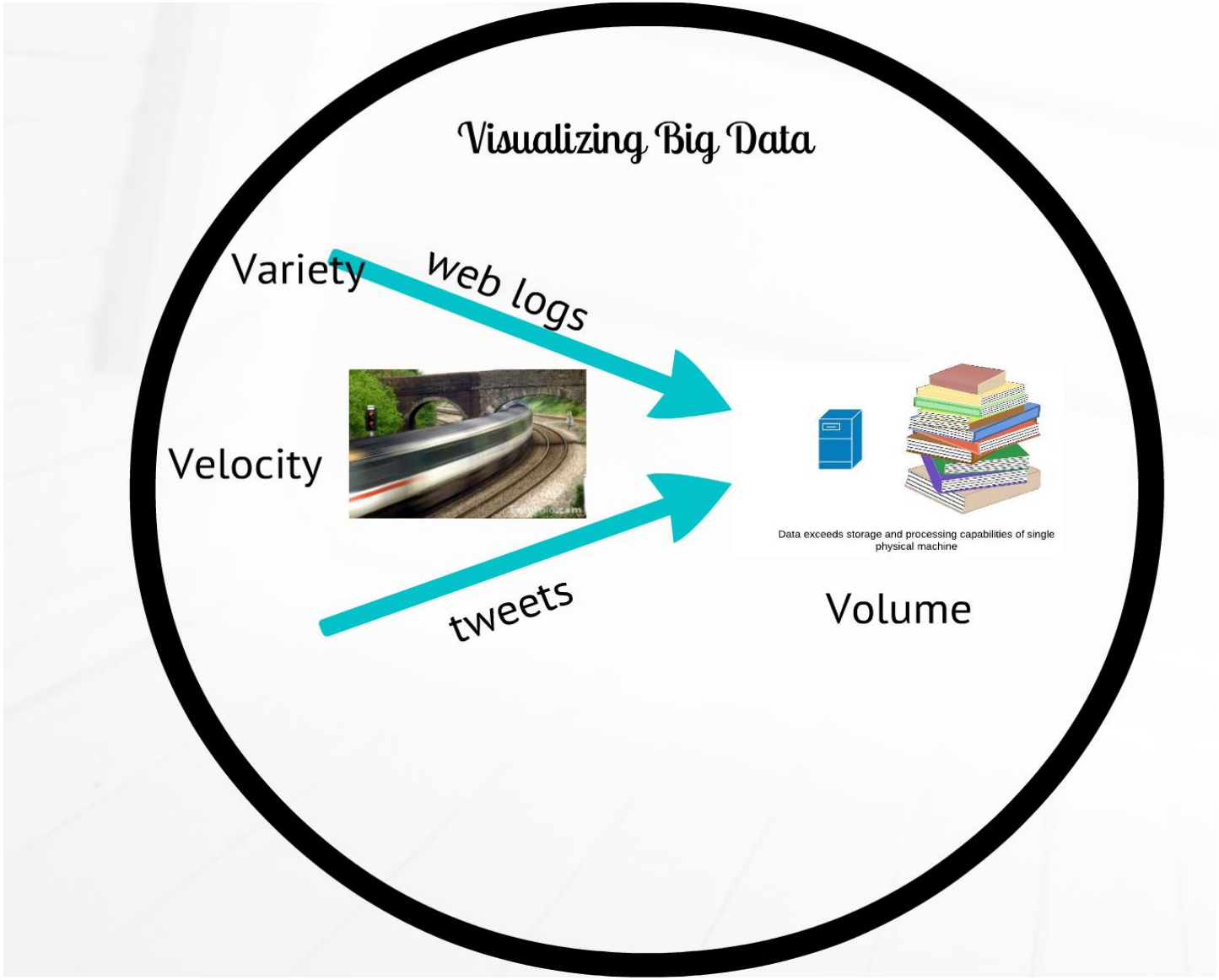
What is Big Data?

- Big Data is the amount of data that is beyond the storage and the processing capabilities of a single physical machine

- Data that has extra large volume, comes from variety of sources, variety of formats and comes at us with a great velocity it normally referred as Big Data

Characteristics of BigData

- **Volume:**
  - For data to be referred as Big Data, generally the volume of data has to be massive
- **Variety:**
  - For data to be referred as Big Data, generally the data would be semi-structured and rnay originate from variety of sources and in variety of formats
- **Velocity:**
  - For data to be referred as Big Data, the rate at which data comes into the system is really fast


Visualizing Big Data
Variety
web logs
Velocity
tweets
Data exceeds storage and processing capabilities of single physical machine
Volume

Next Logical Questions

- **Big Data is clear, but typically how this data looks like?**
  - There is no defined set as to how Big Data looks like. Big Data can look anything like:
    - RFID Systems generated files.
    - Web Logs
    - User Interaction logs.
    - User Transaction history.
    - User's tweet list data generated from Tweeter.
    - Facebook posts.
    - Climate sensors data.
    - And so on……..
- What do we do with this Big Data?
  - Analyse it!

Next Logical Questions

- **What is value in taking pain of analysing this data?**
  - It is in this huge heap of data the golden nuggets of vital information stay!
  - Analysing this data can give business an edge over the competitor and help them serving the user in a more personalized manner.

Next Logical Questions

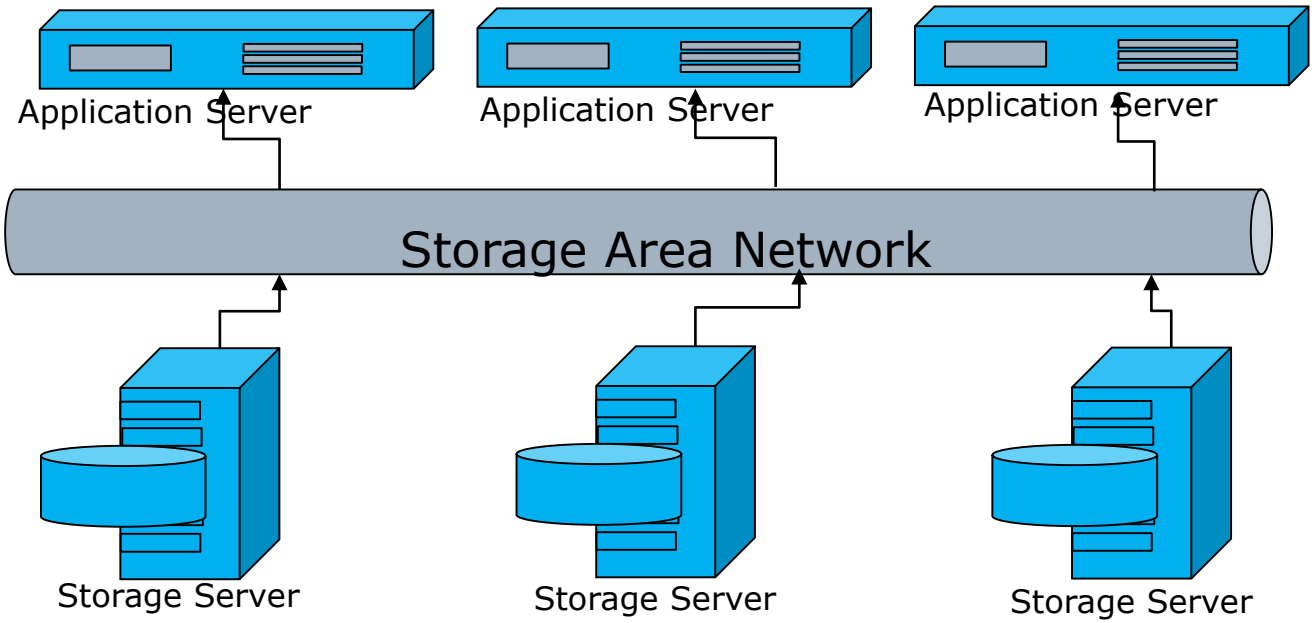
- **Can you give us some use cases of analysing BigData?**
  - Identify the customers who are most important.
  - Identify the best time to perform maintenance based on usage patterns.
  - Analysing your Brand's reputation by analysing the social media posts.
  - And so on….
- How can such a huge data be analysed?
  - As the volume is beyond the processing and storage ability of a Single physical machine, some distributed approach would have to be used!

Distributed Systems to analyse the Big Data.

- Distributed Systems like MPI have been around for more than a decade!
- So do we need to take this course further?
- Does the world need another distributed System?

YES!

Typical Distributed System



- Programs run on each app server
- All the data is on SAN
- Before execution each Server gets the data from SAN
- After execution each App Server writes the data to the SAN

Problems with Typical Distributed Systems

- Huge dependency on Network and huge bandwidth demand
- Scaling up and down is not a smooth process.
- Partial failures are difficult to handle.
- A lot of power is spent on transporting the data!
- Data Synchronization is required during exchange.

This is where HADOOP comes IN!

Apache Hadoop

What is Hadoop?

What is Hadoop?

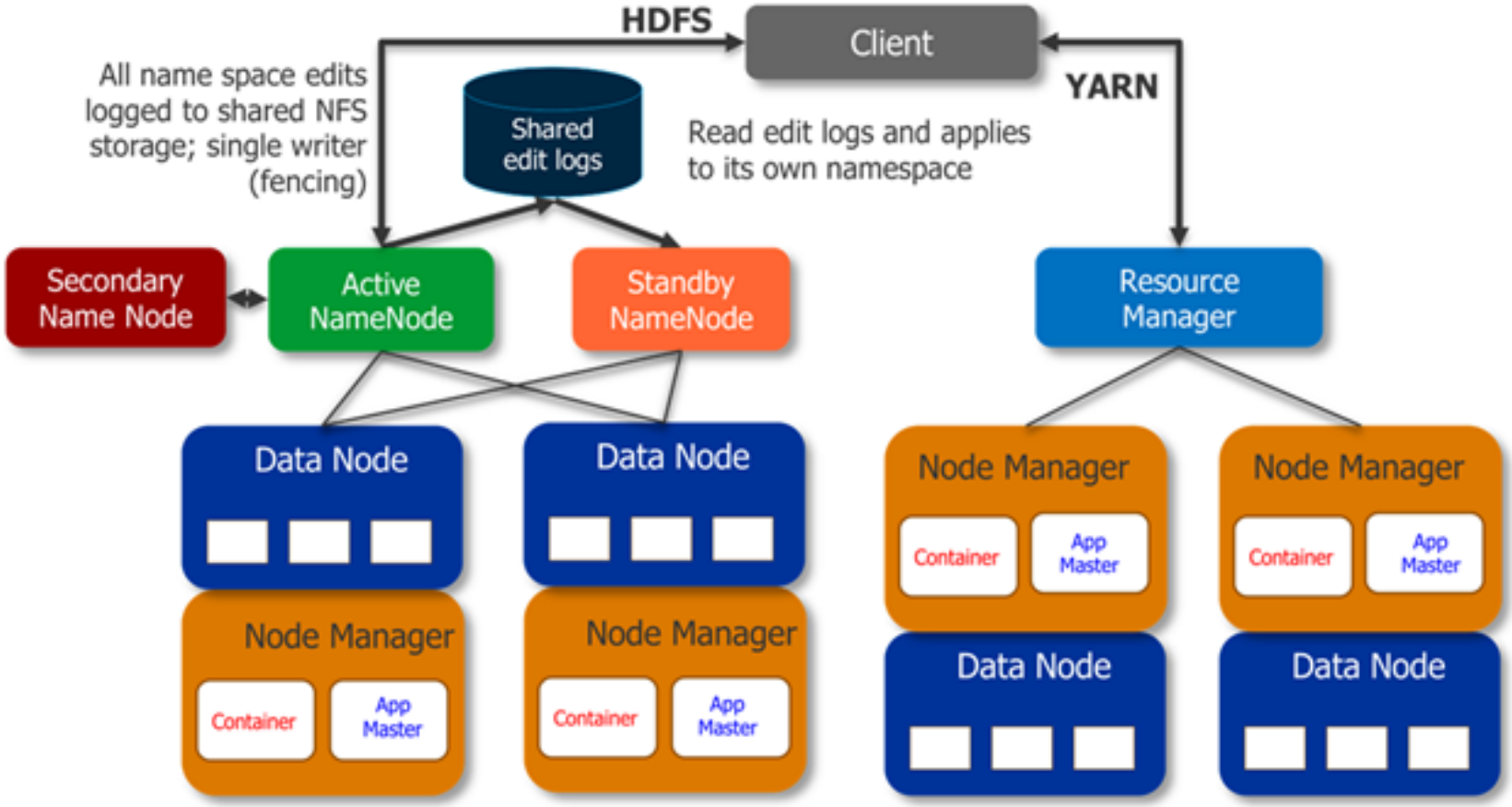
- Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using a simple programing model.
- In more simplistic terms, Hadoop is a framework that facilitates functioning of several machines together to achieve the goal of analyzing large sets of data.
- Google created its own distributed computing framework and published papers about the same. Hadoop was developed on the basis of papers released by Google
- Core Hadoop consists of 2 core components:
  - The Hadoop Distributed File System (HDFS)
  - MapReduce (YARN)
- A set of machines running HDFS and MapReduce is known as Hadoop Cluster
  - Individual machines are knows as nodes
  - A cluster can have as many as 1 node to several thousand nodes

Visualize Hadoop!

BigData

Processing + Data Storage

Visualize Hadoop



Quiz

Which one is not one of the big data feature?

A - Velocity

B - Veracity

C - volume

D - variety

The Hadoop Distributed File System

HDFS

What is Distributed File System?

- When a dataset outgrows the storage capacity of a single physical machine, it becomes necessary to partition it across a number of separate machines.
- Filesystems that manage the storage across a network of machines are called distributed filesystems.
- Since they are network-based, all the complications of network programming kick in, thus making distributed filesystems more complex than regular disk filesystems.

HDFS

- Hadoop comes with a distributed filesystem called HDFS, which stands for **H**adoop **D**istributed **F**ile **S**ystem.
- HDFS is Hadoop's flagship filesystem but Hadoop actually has a general-purpose filesystem abstraction,it can integrate with other storage systems (such as the local filesystem and Amazon S3).

The Design of HDFS

- Appears as a single disk
- Runs on top of a native filesystem
  - Ext3,Ext4,XFS
- Based on Google's Filesystem (GFS or GoogleFS)
- Fault Tolerant
  - Can handle disk crashes, machine crashes, etc...

Design of HDFS

- **Designed to use "Cheap" Commodity Server Hardware**
  - No need for super-computers, use commodity unreliable hardware.
  - Not desktops!
  - Ability to provide storage for huge amount of data using commodity hard ware is its biggest strength.
  - Works optimally for less number of large files and not large number of small files
  - Not optimized for random reads.
  - Files in HDFS are write once, once in HDFS files cannot be edit.

Design of HDFS

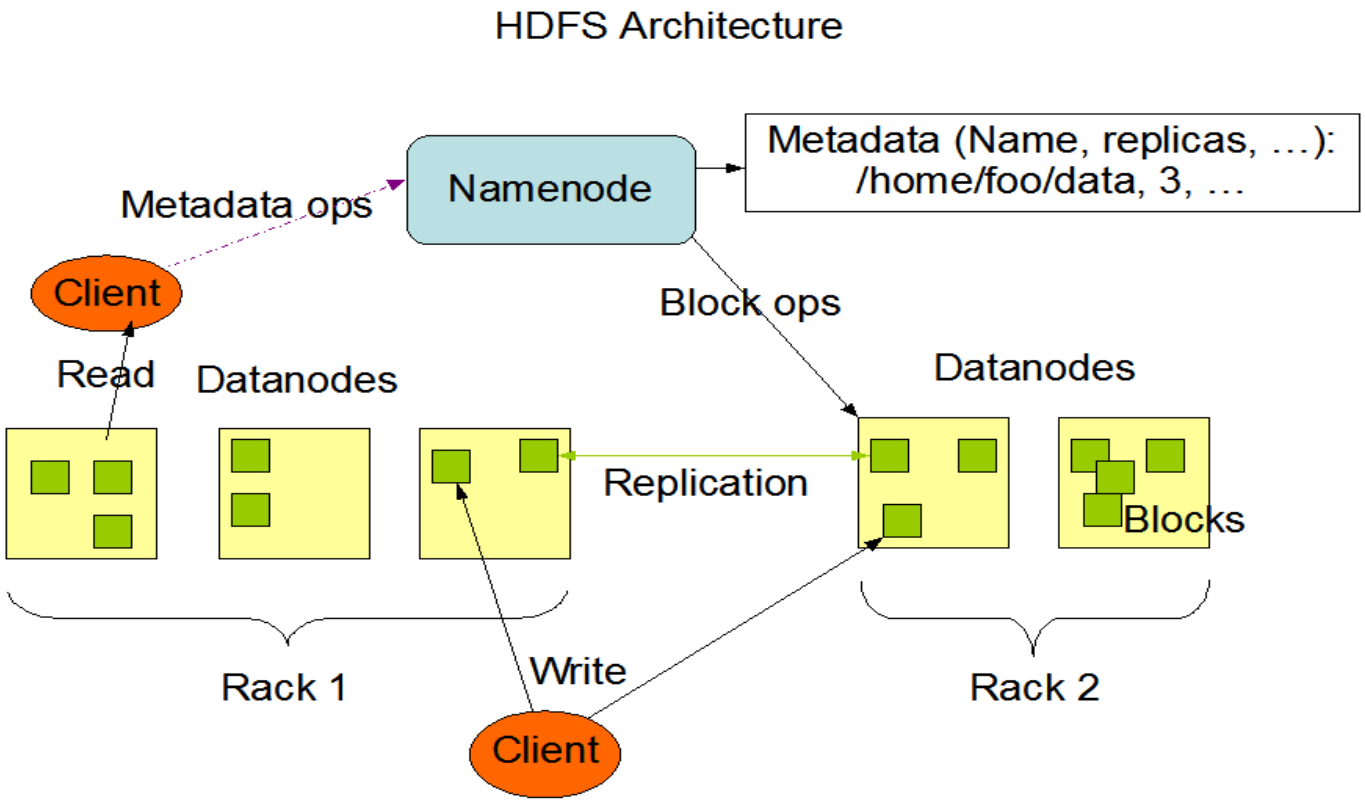
- HDFS is Good for
  - **Storing large files**
    - Terabytes, Petabytes, etc...
    - Millions rather than billions of files
    - 100MB or more per file
  - **Streaming data**
    - Write once and read-many times patterns
    - Optimized for streaming reads rather than random reads
    - Append operation added to Hadoop 0.21

Design of HDFS

- **HDFS is not so good for**
  - **Low-latency reads**
    - High-throughput rather than low latency for small chunks of data
    - HBase addresses this issue
  - **Large amount of small files**
    - Better for millions of large files instead of billions of small files
    - For example each file can be 100MB or more
  - **Multiple Writers**
    - Single writer per file
    - Writes only at the end of file, no-support for arbitrary offset

HDFS Architecture

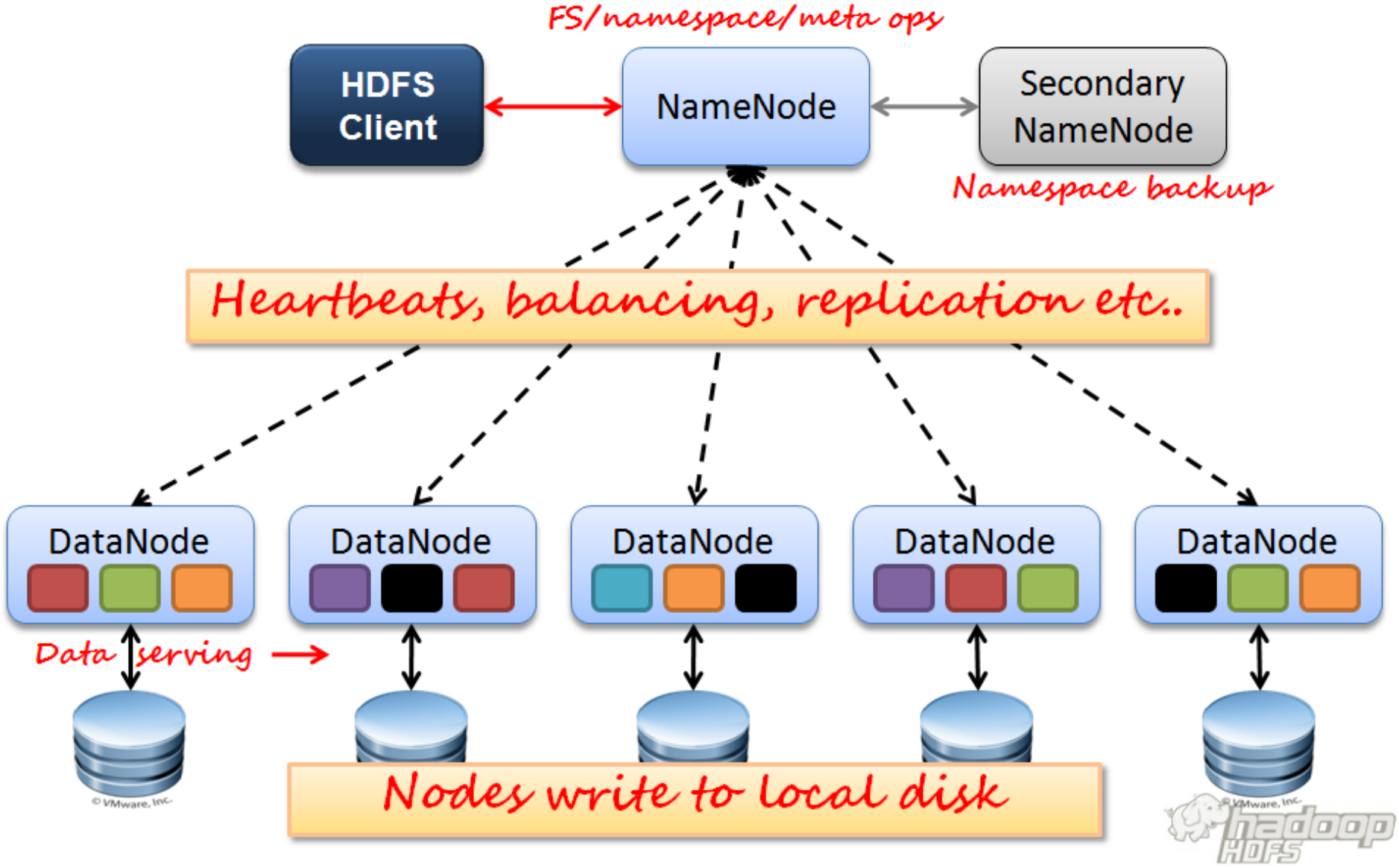
- HDFS has a master-slave architecture.
- An HDFS cluster has two types of nodes operating in a master-worker pattern: a **namenode** (the master) and a number of **datanodes** (workers).



HDFS Daemons

- **File system cluster is managed by three types of processes**
  - **Namenode**
    - manages the File System's namespace/meta-data/file blocks
    - Runs on 1 machine to several machines
  - **Datanode**
    - Stores and retrieves data blocks
    - Reports to Namenode
    - Runs on many machines
  - **Secondary Namenode**
    - Performs house keeping work so Namenode doesn't have to
    - Requires similar hardware as Namenode machine
    - Not used for high-availability – not a backup for Namenode

HDFS Daemons



Files and Blocks

- **Files are split into blocks (single unit of storage)**
  - Managed by Namenode, stored by Datanode
  - Transparent to user
- **Replicated across machines at load time**
  - Same block is stored on multiple machines
  - Good for fault-tolerance and access
  - Default replication is 3

HDFS Blocks

- **Blocks are traditionally either 64MB or 128MB**
  - Default is 64MB
- **The motivation is to minimize the cost of seeks as compared to transfer rate**

  'Time to transfer' > 'Time to seek'
- **For example, lets say**

  seek time = 10ms

  Transfer rate = 100 MB/s
- **To achieve seek time of 1% transfer rate**
  - Block size will need to be = 100MB

Block Replication

- **Namenode determines replica placement**
- **Replica placements are rack aware**
  - Balance between reliability and performance
    - Attempts to reduce bandwidth
    - Attempts to improve reliability by putting replicas on multiple racks
  - Default replication is 3
    - 1st replica on the local rack
    - 2nd replica on the local rack but different machine
    - 3rd replica on the different rack
- This policy may change/improve in the future

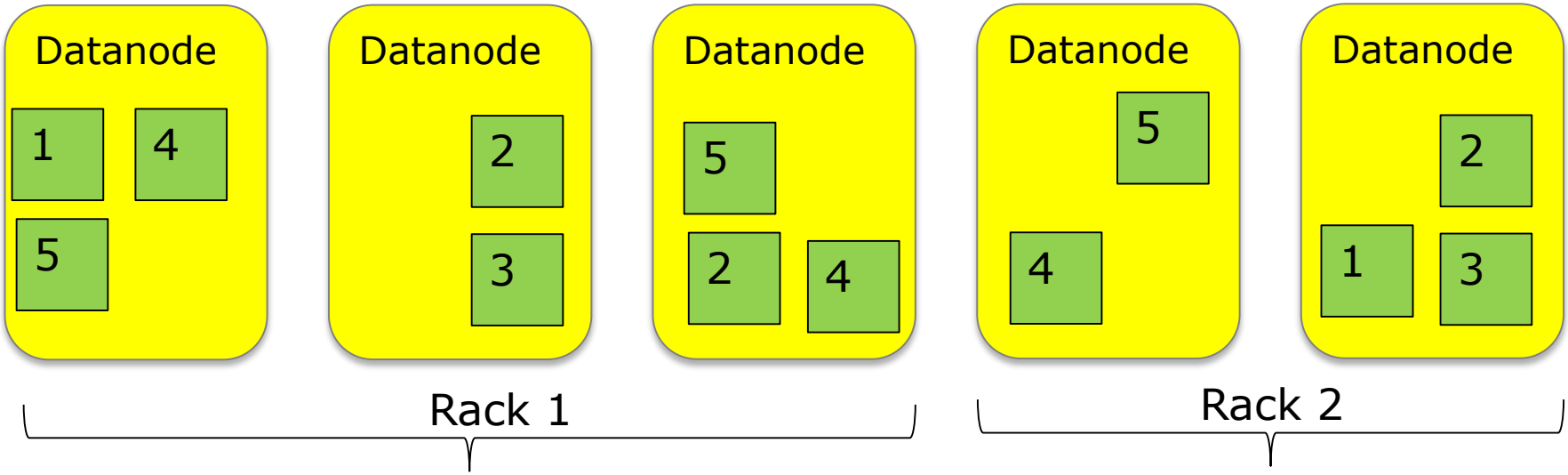
How files are stored in HDFS

Namenode

Filename…..numReplicas….block id

/user/shan/mylog1.txt..r:2..{1,3}

/user/shan/mylog2.txt..r:3..{2,4,5}



How files are written to HDFS

- Client, Namenode, and Datanodes
  - Namenode does NOT directly write or read data
    - One of the reasons for HDFS's Scalability
  - Client interacts with Namenode to update Namenode's HDFS namespace and retrieve block locations for writing and reading
  - Client interacts directly with Datanode to read/write data

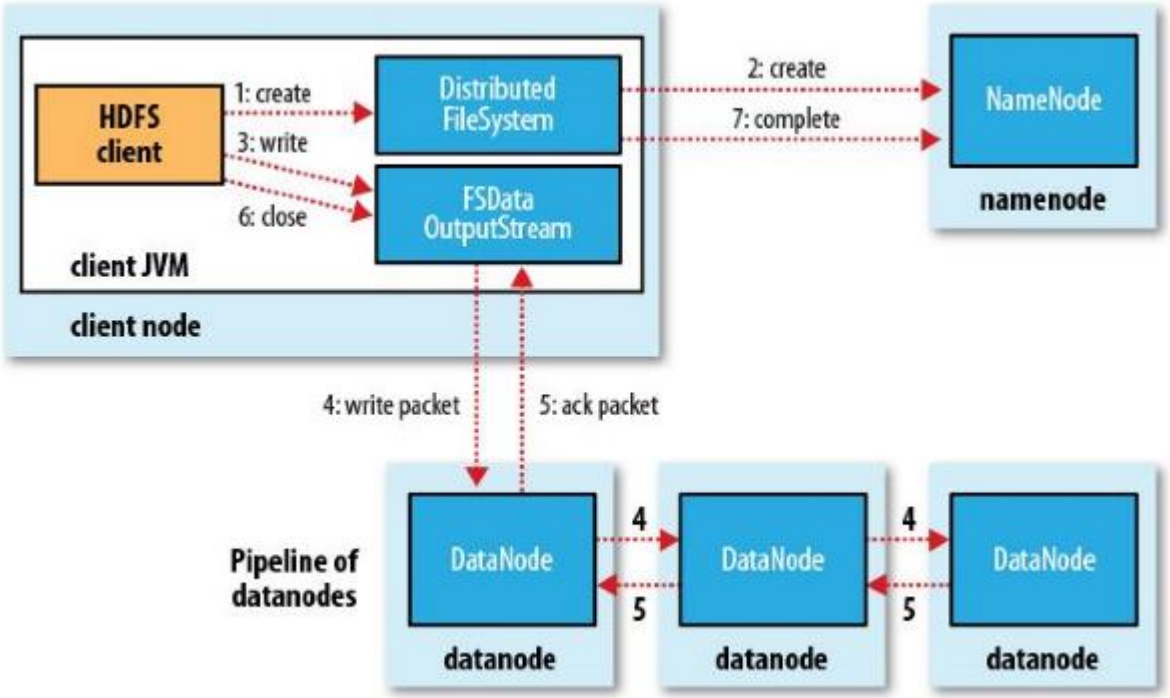
HDFS File Write

1,2: Create new file in the Namenode's Namespace; calculate block topology

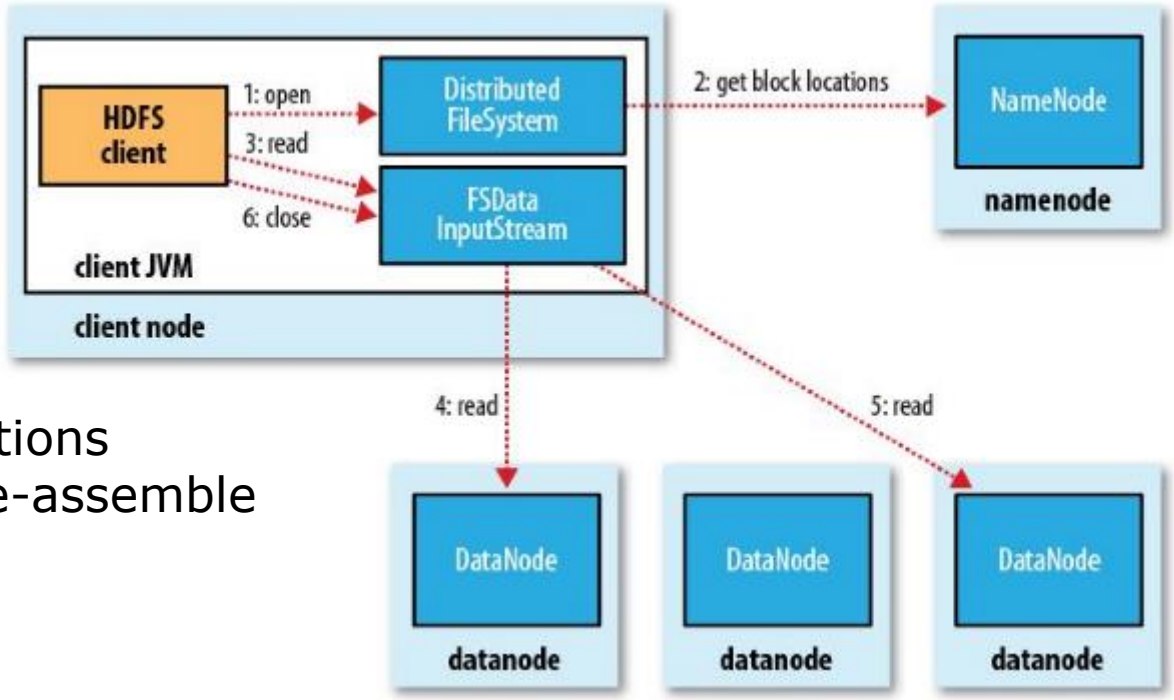
3,4: The client Stream data to the first Datanode, Stream data to the second node in the pipeline and Stream data to the third node

5: Success/Failure acknowledgment

6,7: Write complete



HDFS File Read



1,2 Retrieve Block Locations
3,4,5: Read blocks to re-assemble the file
6:done

Quiz

What is true about HDFS?

A - HDFS filesystem can be mounted on a local client’s Filesystem using NFS.

B - HDFS filesystem can never be mounted on a local client’s Filesystem.

C - You can edit a existing record in HDFS file which is already mounted using NFS.

D - You cannot append to a HDFS file which is mounted using NFS.

Quiz

The namenode knows that the datanode is active using a mechanism known as

A - heartbeats

B - datapulse

C - h-signal

D - Active-pulse

Quiz

When a client contacts the namenode for accessing a block, the namenode responds with

A - Size of the file requested.

B - Block ID of the file requested.

C - Block ID and hostname of any one of the data nodes containing that block.

D - Block ID and hostname of all the data nodes containing that block.

MapReduce

- MapReduce is the Data Processing component of Hadoop
- It attains the task of data processing by distributing tasks across the nodes
- Task on each node processes data present locally (Mostly)
- Consists of 2 phases
  - Map
  - Reduce
- In between Map and Reduce, there is small phase called Shuffle and Sort

MapReduce Example

- Assume we have a file that has some text and its named as data.txt
- The goal is to count number of times each word occurs

```
Hello How are you
Where have you been these days
Where can I find you
Are you there
```

- The required output is:

(Hello,1) (How 1) (Are,2) (You,4) (Where,2) (have 1) (been,1) (these,1) (days 1) (can 1) (I 1) (Find 1) (there 1)

- In MapReduce, every input is viewed as Key-Value Pair. In the above case

  Key --- line number i.e. 1

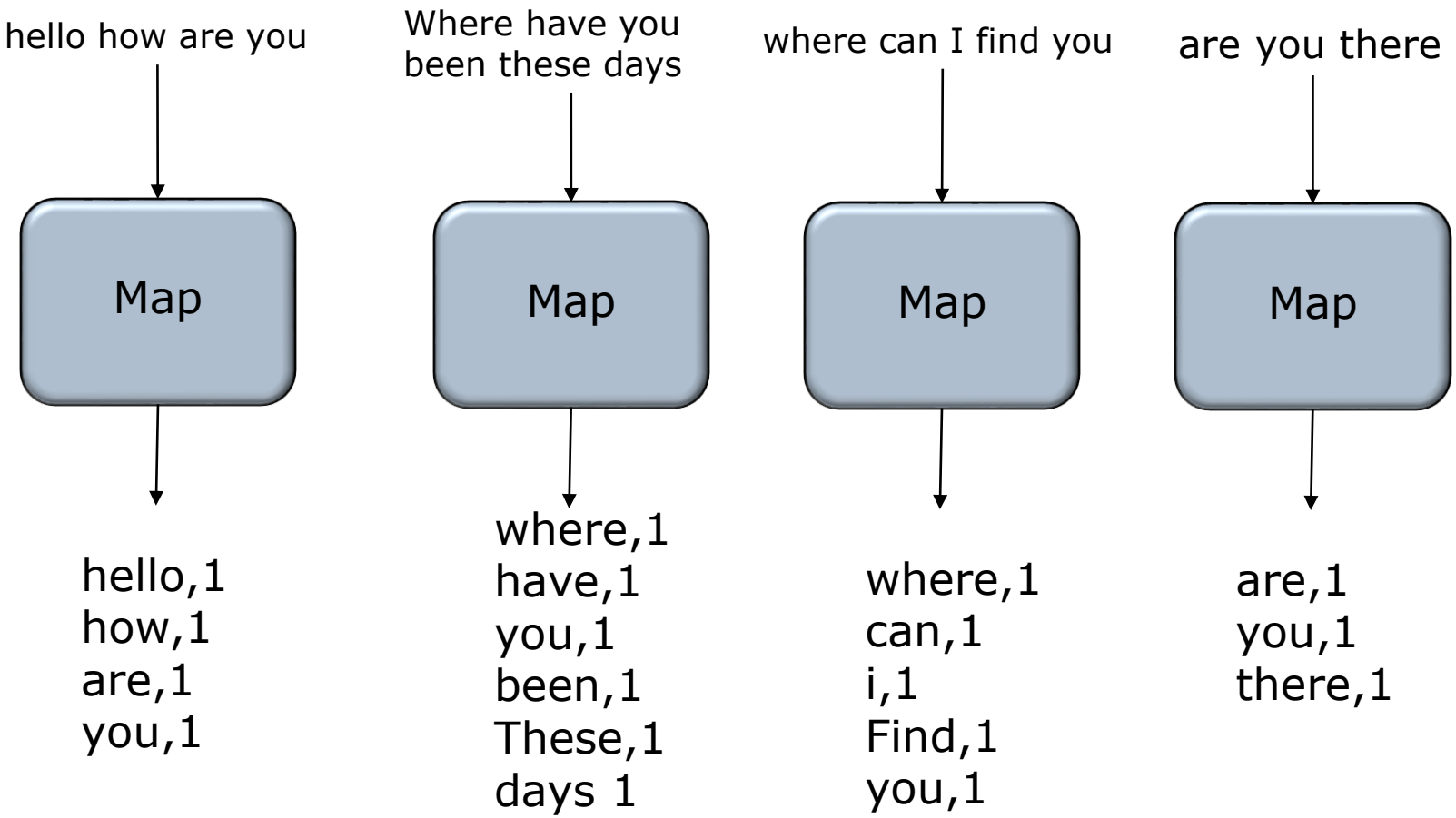
  Value--- the Line i.e. Hi How are you

The Map Phase

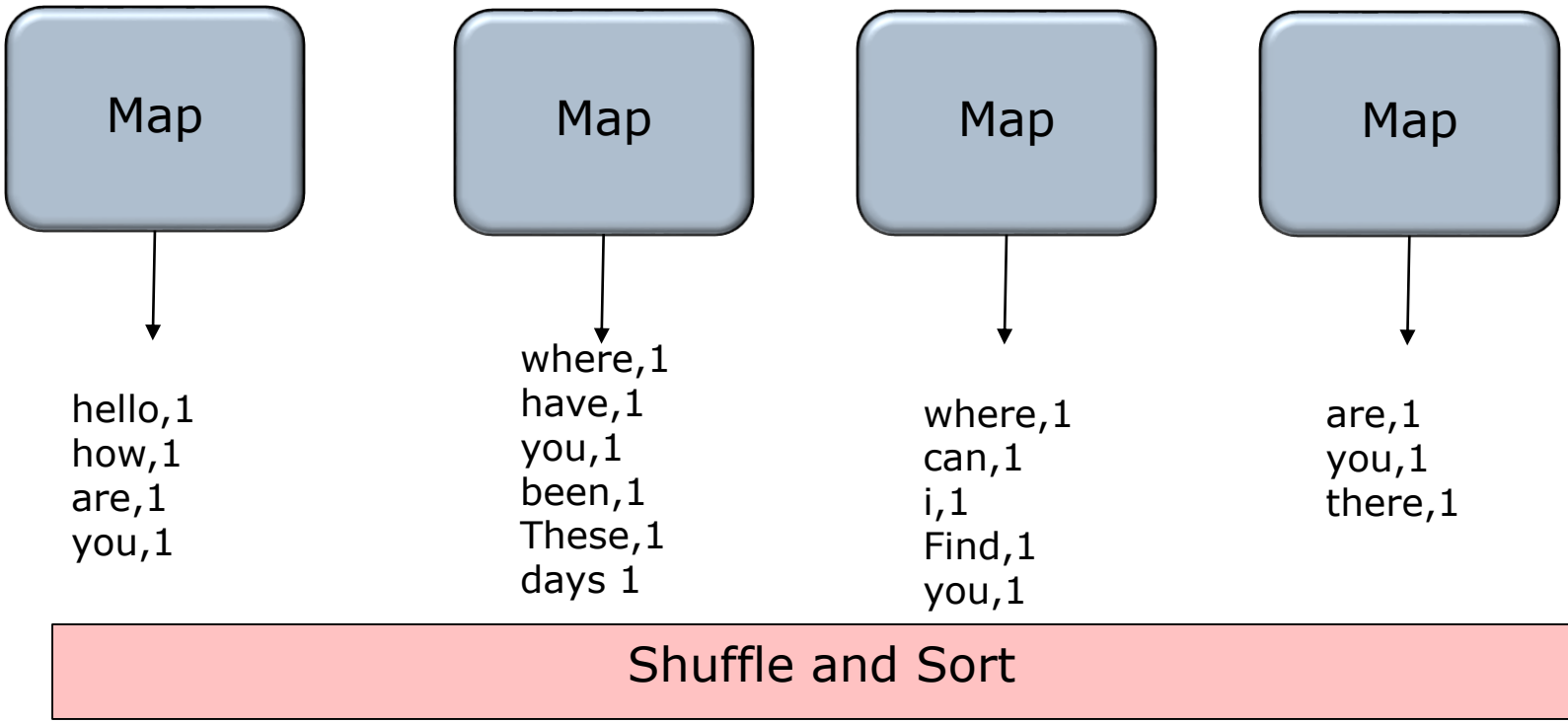
- Hypothetically this is how the map phase works

for each word in value

emit(word,1)



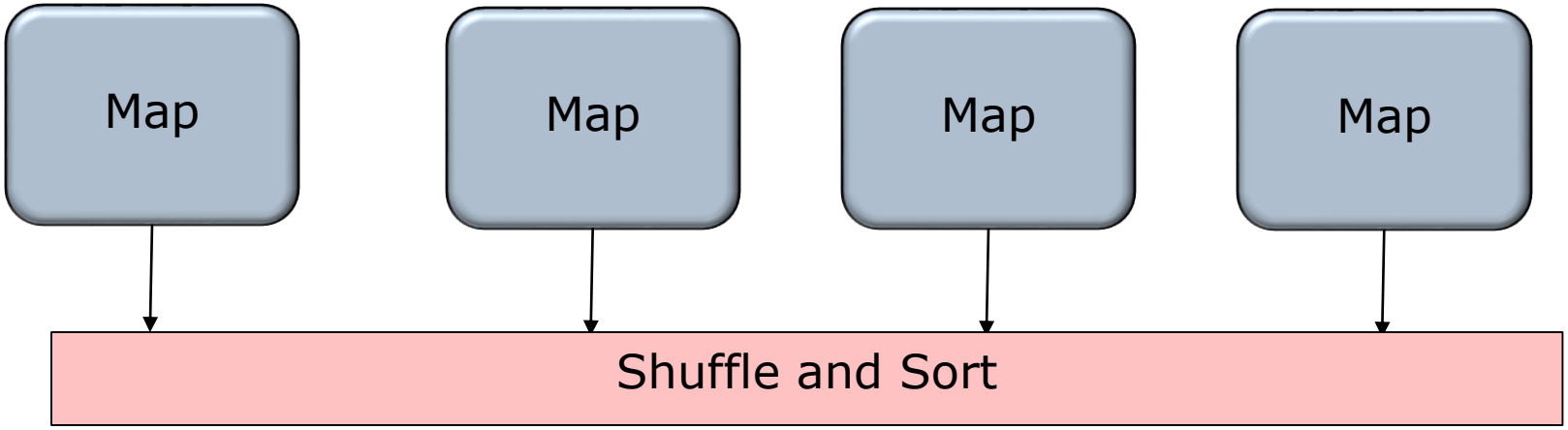
Shuffle and Sort



(are,1,1),(been,1),(can,1),(days,1),(find,1),(have,1),(hello,1),

(how,1),(I,1),(there,1),(there,1),(you,1,1,1,1),(where,1,1)

Shuffle and Sort



(are,1,1),(been,1),(can,1),(days,1),(find,1),(have,1),(hello,1),
(how,1),(I,1),(there,1),(there,1),(you,1,1,1,1),(where,1,1)

Reduce

```
for each v in value
sum= sum + v
emit(k, sum)
```

(are,2),(been,1),(can,1),(days,1),(find,1),(have,1),(hello,1),
(how,1),(I,1),(there,1),(there,1),(you,4),(where,2)

Quiz

The output of a mapper task is

A - The Key-value pair of all the records of the dataset.

B - The Key-value pair of all the records from the input split processed by the mapper

C - Only the sorted Keys from the input split

D - The number of rows processed by the mapper task.

Quiz

Arrange in Correct sequence

A. map,reduce,sort,shuffle

B. Map,sort, shuffle,reduce

C. Map,shuffle,sort,reduce

D. Any order

Quiz

For the given data set predict the out put after sort phase

river water fish water river bear fish water

A. (river,1),(water,1), (fish,1),(river,1),(bear,1) ,(fish,1), (water,1)

B. (river,2),(water,3), (fish,2),(bear,1)

C. (river,1,1),(water,1,1,1), (fish,1,1),(bear,1)

D. (bear,1), (fish,1,1),(river,1,1), (water,1,1,1)

Map Reduce Architecture(YARN)

Deep dive

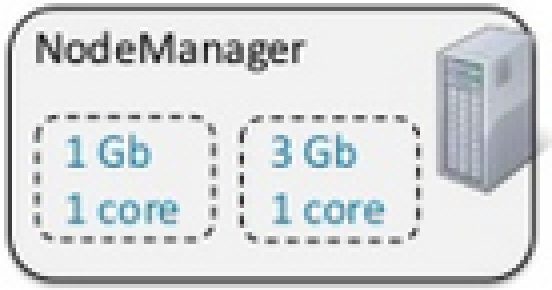
YARN Daemons

- **Resource Manager (RM)**
  - Runs on master node
  - Global resource scheduler
  - Arbitrates system resources between competing applications

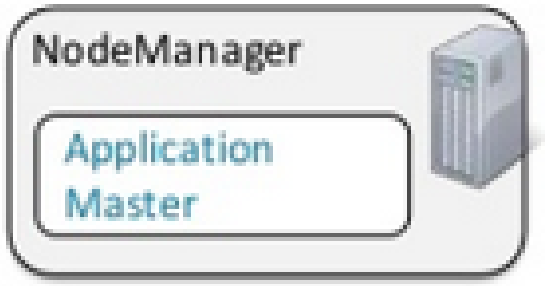
- Node Manager (NM)
  - Runs on slave nodes
  - Communicates with Resource Manager (RM)

Running an Application in YARN

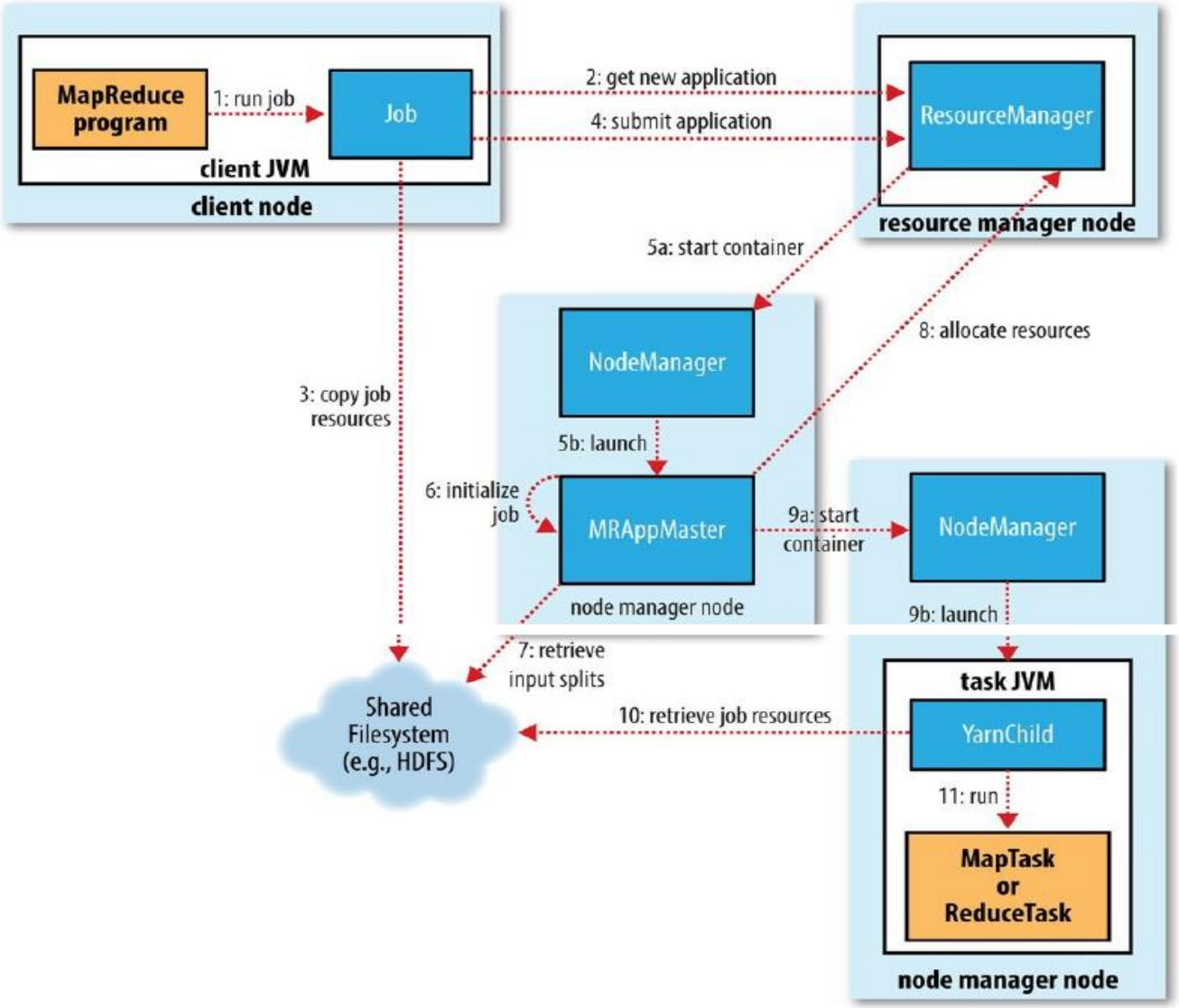
- Containers
  - Created by the RM upon request
  - Allocate a certain amount of resour
    - Memory,CPU on a slave node
  - Applications run in one or more containers



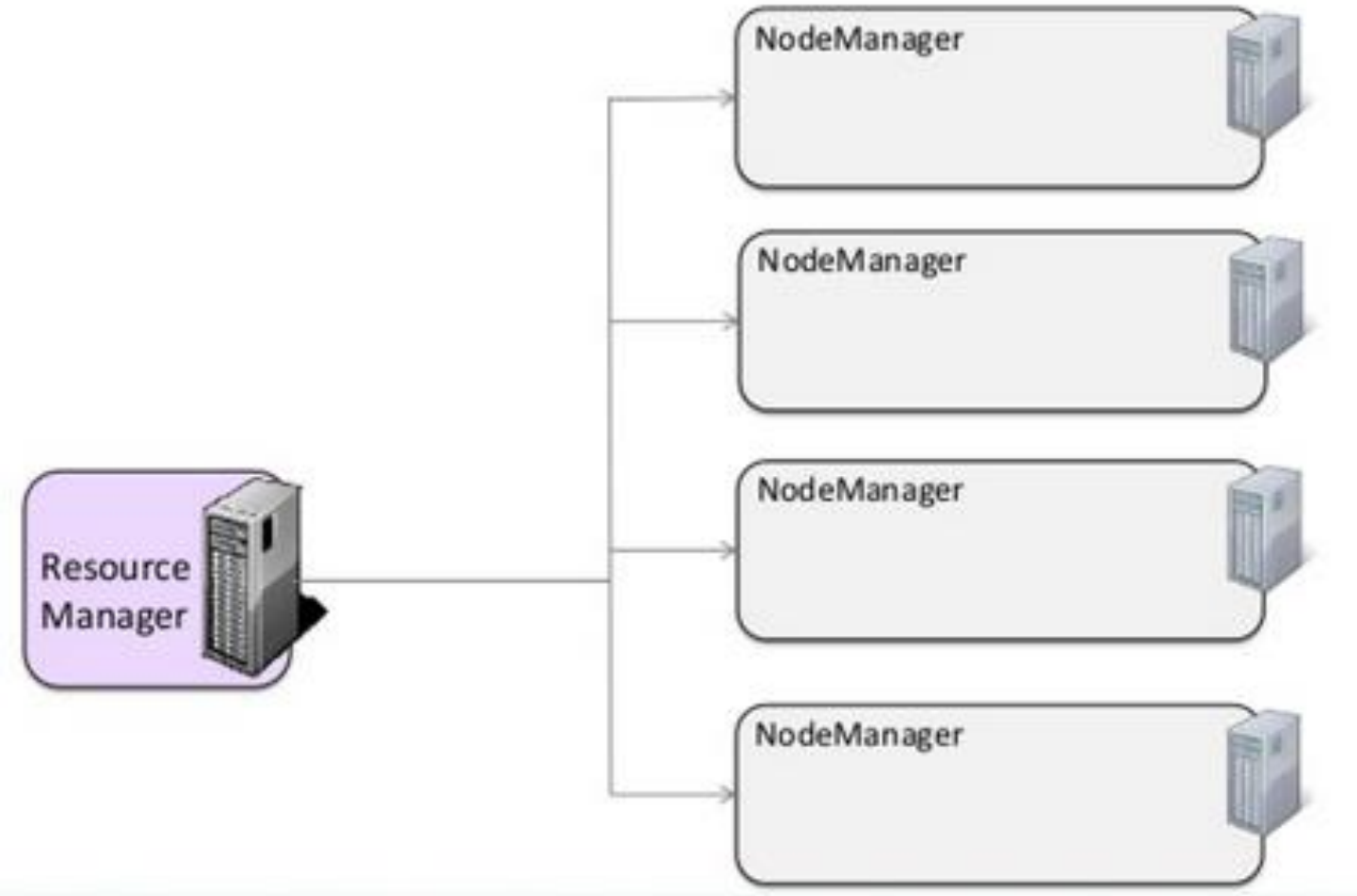
- Application Master (AM)
  - One per application
  - Framework/application specific
  - Runs in a container
  - Requests more containers to run application tasks



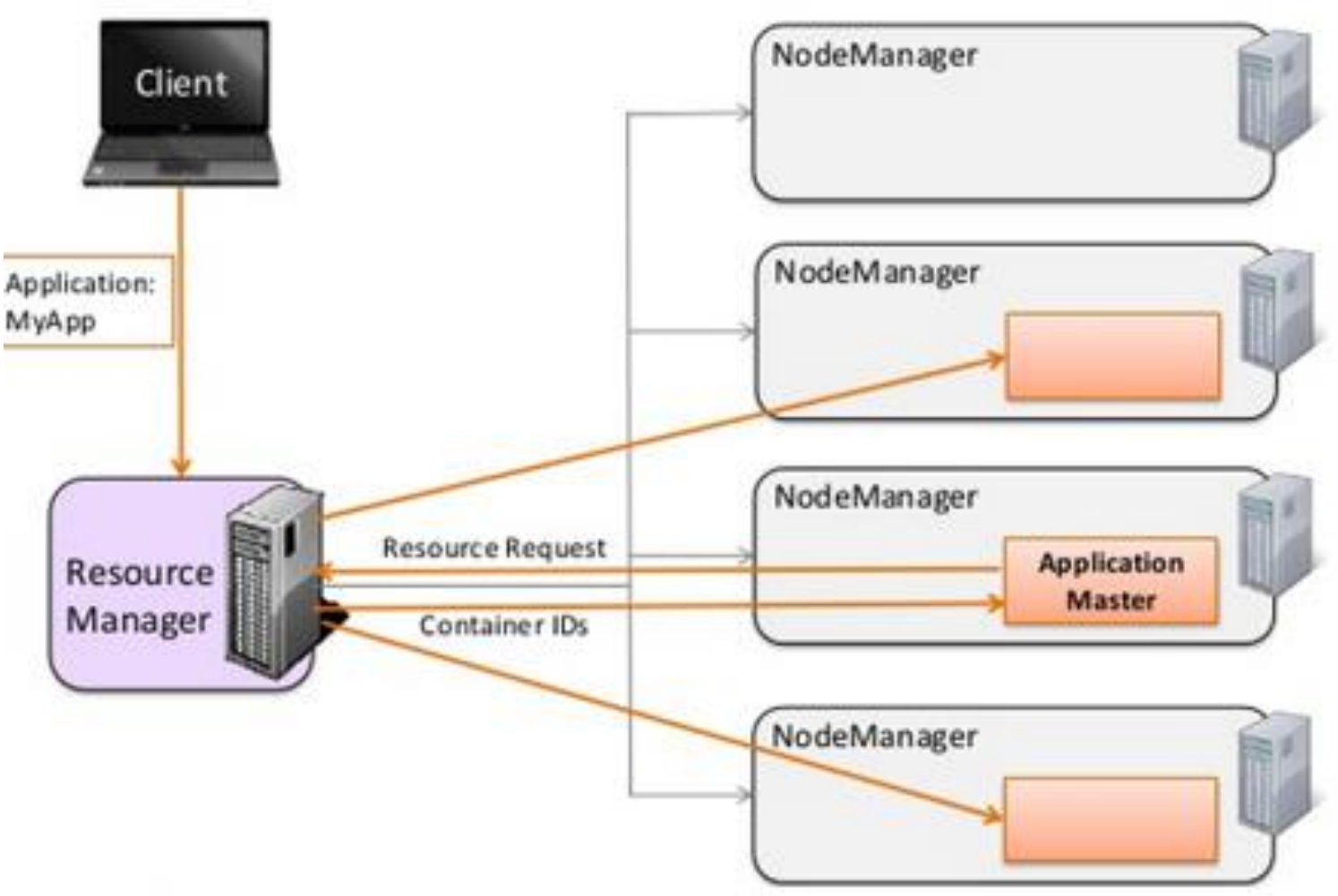
YARN Architecture



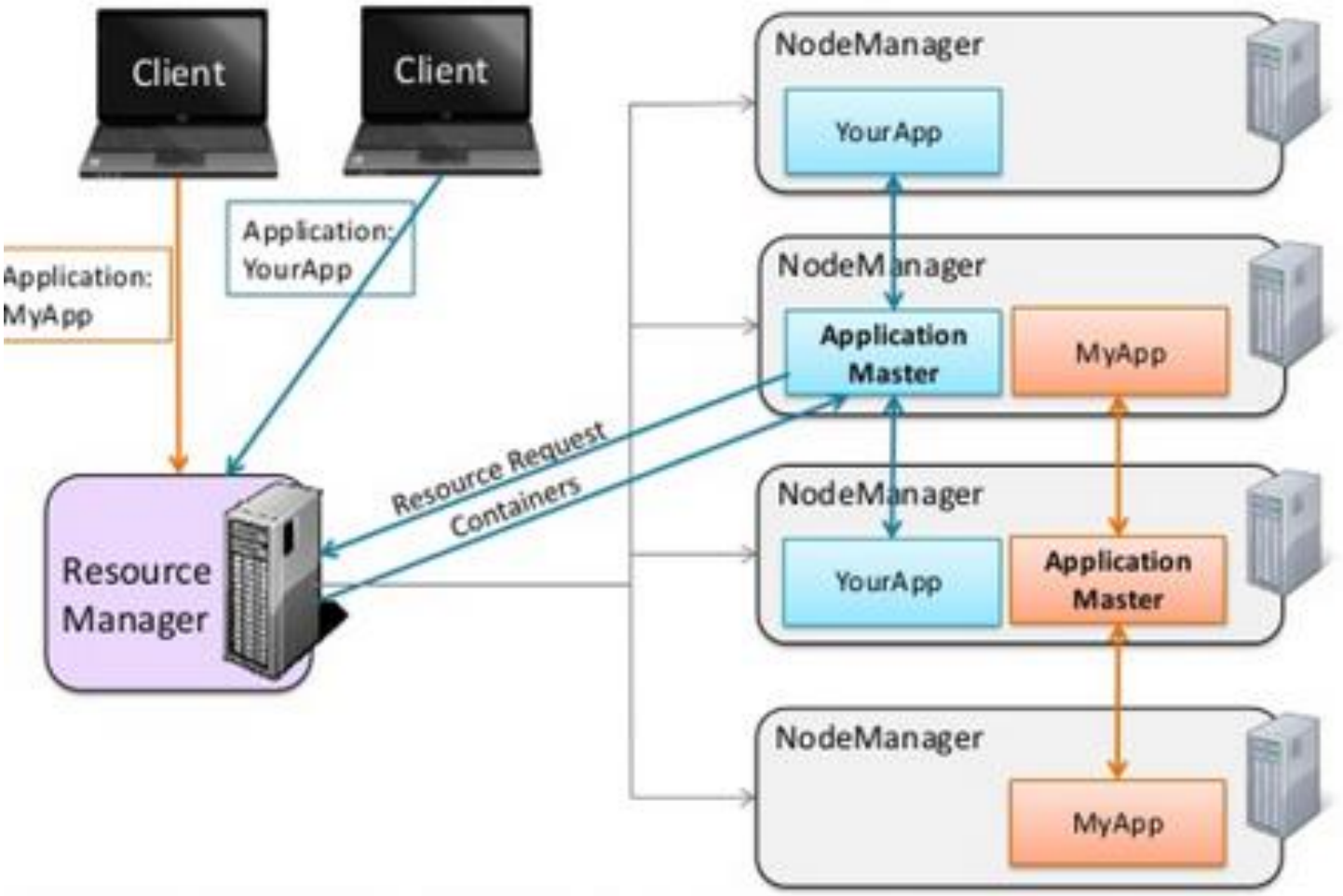
YARN Cluster



YARN Cluster : Running an application



YARN Cluster : Running an application



Resource Manager : Things to know

- What it does
  - Manages nodes
    - Tracks heartbeats from NodeManagers
  - Manages containers
    - Handles AM requests for resources
    - De-allocates containers when they expire or the application completes
  - Manages Application Masters
    - Creates a container for AM and tracks heartbeats
  - Manages security
    - Supports Kerberos

Node Manager: things to know

- What it does
  - Communicates with RM
    - Registers and provides info on node resources
      - Sends heartbeats and container status
    - Manages processes in containers
      - Launches AMs on request from the RM
      - Launches application processes on request from AM
      - Monitors resource usage by containers;kills run-away processes
    - Provides Logging services to aplications
      - Aggregates logs for an application and saves them to HDFS
    - Runs auxiliary services
    - Maintains node level security via ACLs

Quiz

Q. Which is true for Application Master

A. Min of one Application Master per data node

B. Min of one Application master per namenode

C. Min of one Application Master per JobTracker

D. Min of one Application master per application

Quiz

Q. Which of the component of YARN is responsible for allocation of CPU, Memory etc:

A. NodeManager

B. Application Master

C. Resource Manager

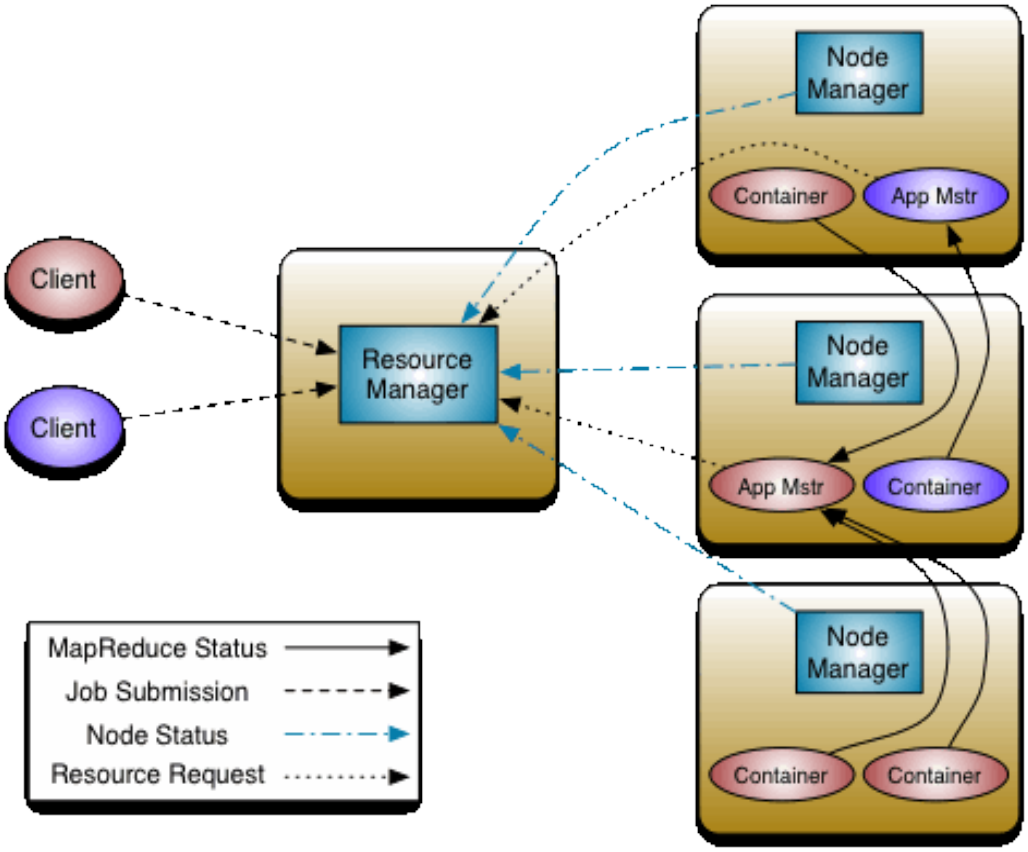
D. Node master

Hadoop Cluster and MapReduce

Hadoop Cluster

- A **Hadoop cluster** is a special type of computational **cluster** designed specifically for storing and analyzing huge amounts of unstructured data in a distributed computing environment.
- Components of the Cluster
  - Namenode,Secondary Name Node,data nodes
  - Resource Manager,NodeManager

Visualize Hadoop Cluster

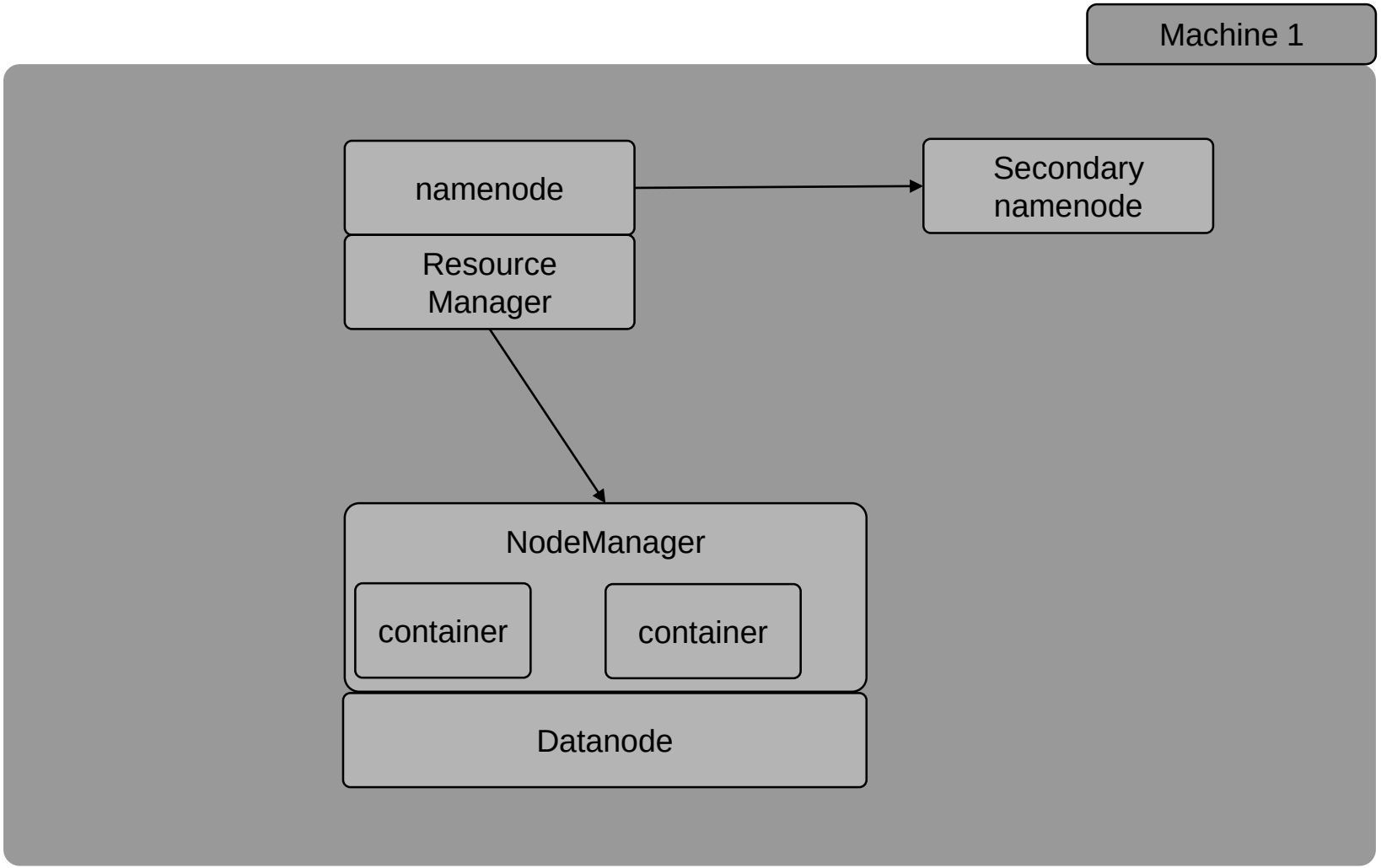


Hadoop 2.x Cluster

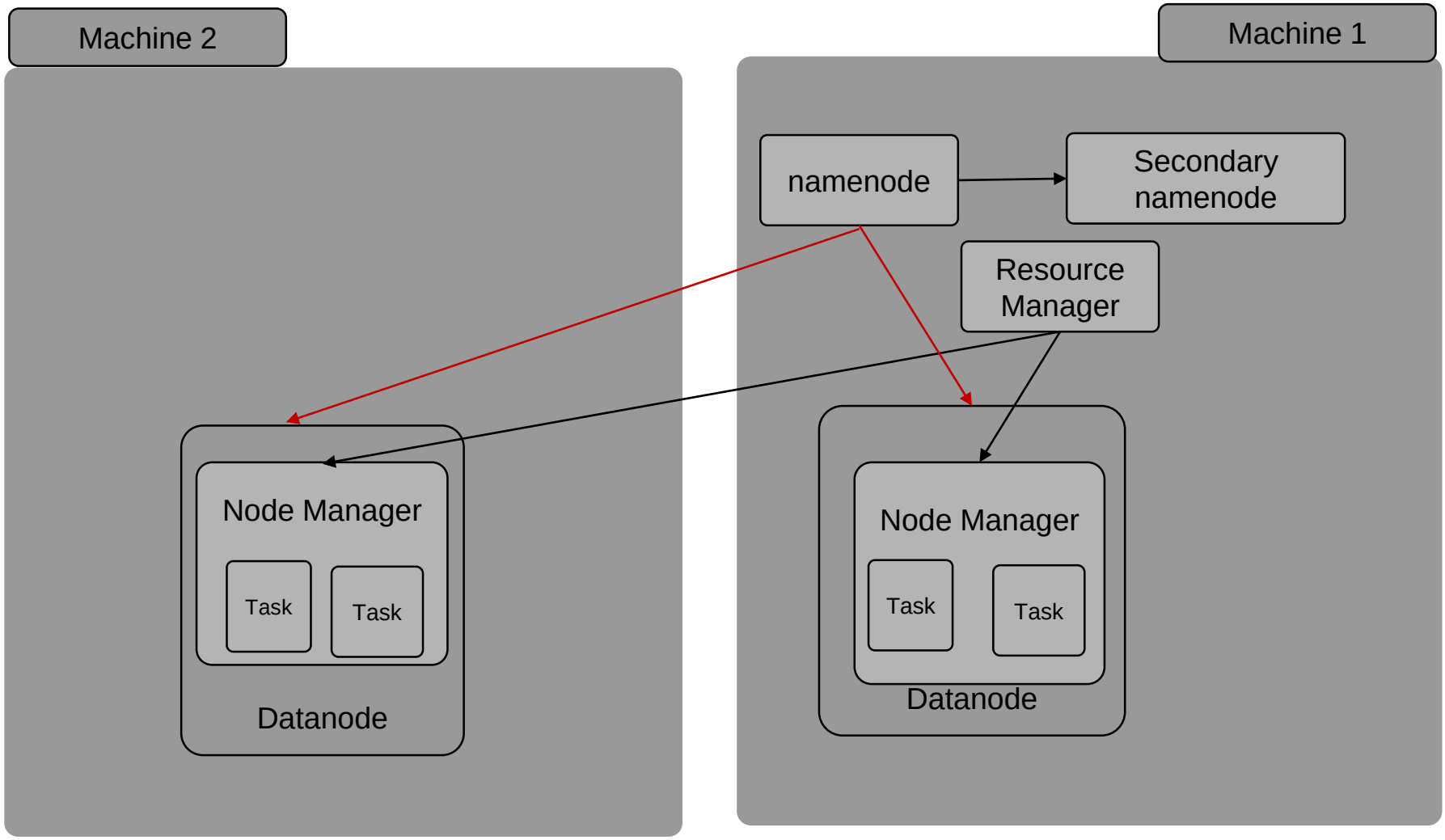
Installation

- Hadoop can be installed in 3 different ways
  - Standalone
  - Pseudo Distributed mode
  - Multi node cluster (Fully Distributed)

Hadoop in Standalone mode



Hadoop in Pseudo Distributed mode



Quiz

Which of the following is not a Hadoop operation mode?

A - Pseudo distributed mode

B - Globally distributed mode

C - Stand alone mode

D - Fully-Distributed mode

Quiz

The difference between standalone and pseudo-distributed mode is

A - Stand alone cannot use map reduce

B - Stand alone has a single java process running in it.

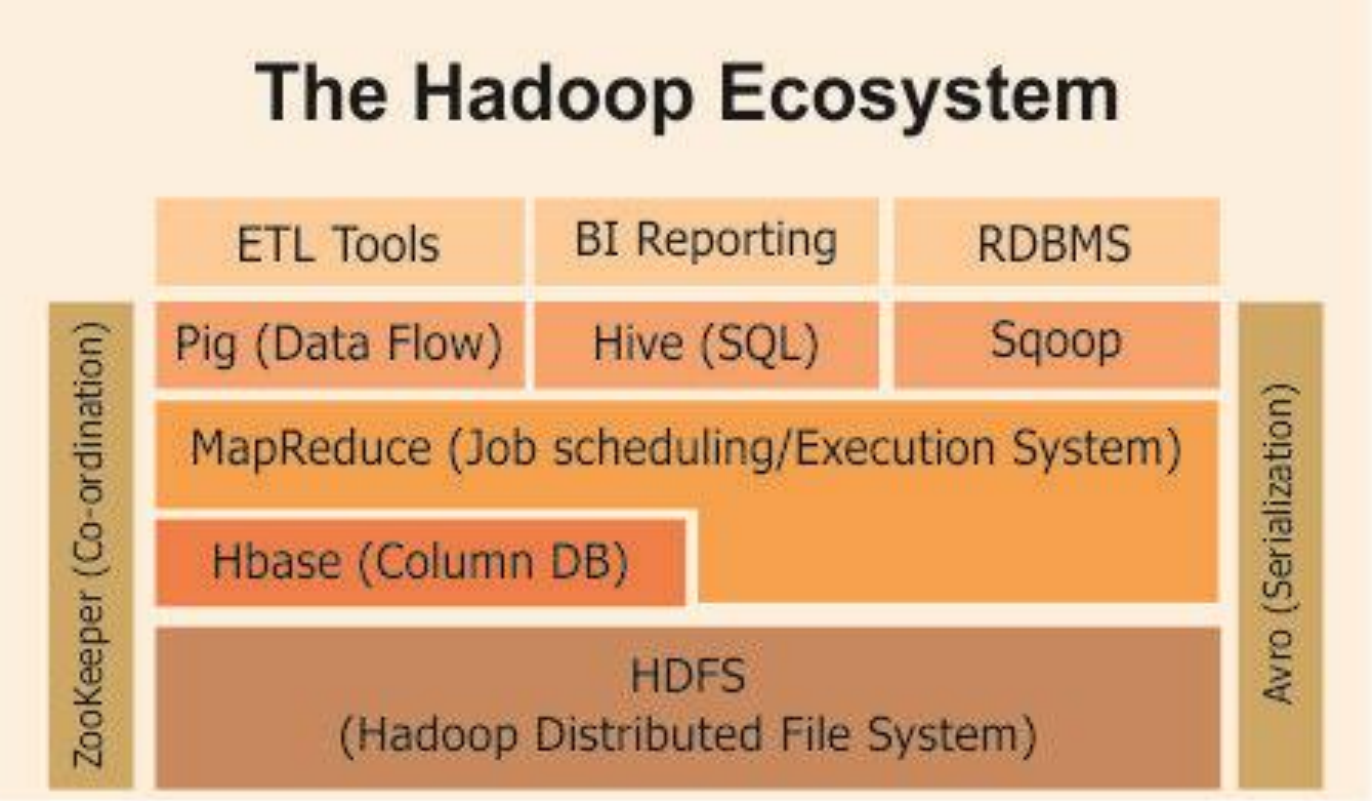
C - Pseudo distributed mode does not use HDFS

D - Pseudo distributed mode needs two or more physical machines.

How to run a MapReduce Application on Hadoop

- Demonstration by Trainer

Hadoop Ecosystem


The Hadoop Ecosystem
ETL Tools
BI Reporting
RDBMS
Pig (Data Flow)
Hive (SQL)
Sqoop
MapReduce (Job scheduling/Execution System)
Hbase (Column DB)
HDFS
(Hadoop Distributed File System)
ZooKeeper (Co-ordination)
Avro (Serialization)

Hadoop Ecosystem

Sqoop

What is Sqoop?

- Sqoop is a tool that is used to import data from SOL databases to HDFS
- Open source tool developed by Cloudera
- Can do SOL->Hadoop as well as Hadoop->SOL
- Uses JDBC to talk to the database
- Can import single table or all tables in database
- As a developer, you can specify the columns, or rows that need to be imported. It also facilitates SELECT query for selective import
- Can directly create Hive tables and push data, also supports incremental imports

How Sqoop Works?

- Sqoop takes the commandsfrom user, analyzes the tables and generates Java code that will take care of data import to HDFS
- After the Java code generation, a Map Only Mapreduce job is run to import the data
- By default 4 mappers are run with each mapper importing 25% of data

An Example

- List tables

```
sqoop list-tables --username root --password demo \
--connect jdbc:mysql://yourdb.database.com/dbname
```


OOZIE

Oozie

- Not all solutions can be attained with single Mapreduce program, at times output of one Mapreduce program would have to be given as an input to another program.
- Oozie does exactly that.
- Oozie is a workflow engine that runs on the server, it facilitates functionality like:
  - Run Program1
  - Take output of Program l and give it as input to Program2
- Not only restricted to MapReduce, but also does Pig, Hive and Sqoop jobs
- Jobs can be scheduled to run at specific intervals.

How Oozie works

- Oozie workflow is a set of actions to be performed in a particular sequence
- Oozie workflows are written in XML
- Oozie workflow has two main components:
  - Control flow nodes
  - Action nodes
- Control flow nodes define beginning and end of a work flow
- Action nodes are real execution is triggered

An example

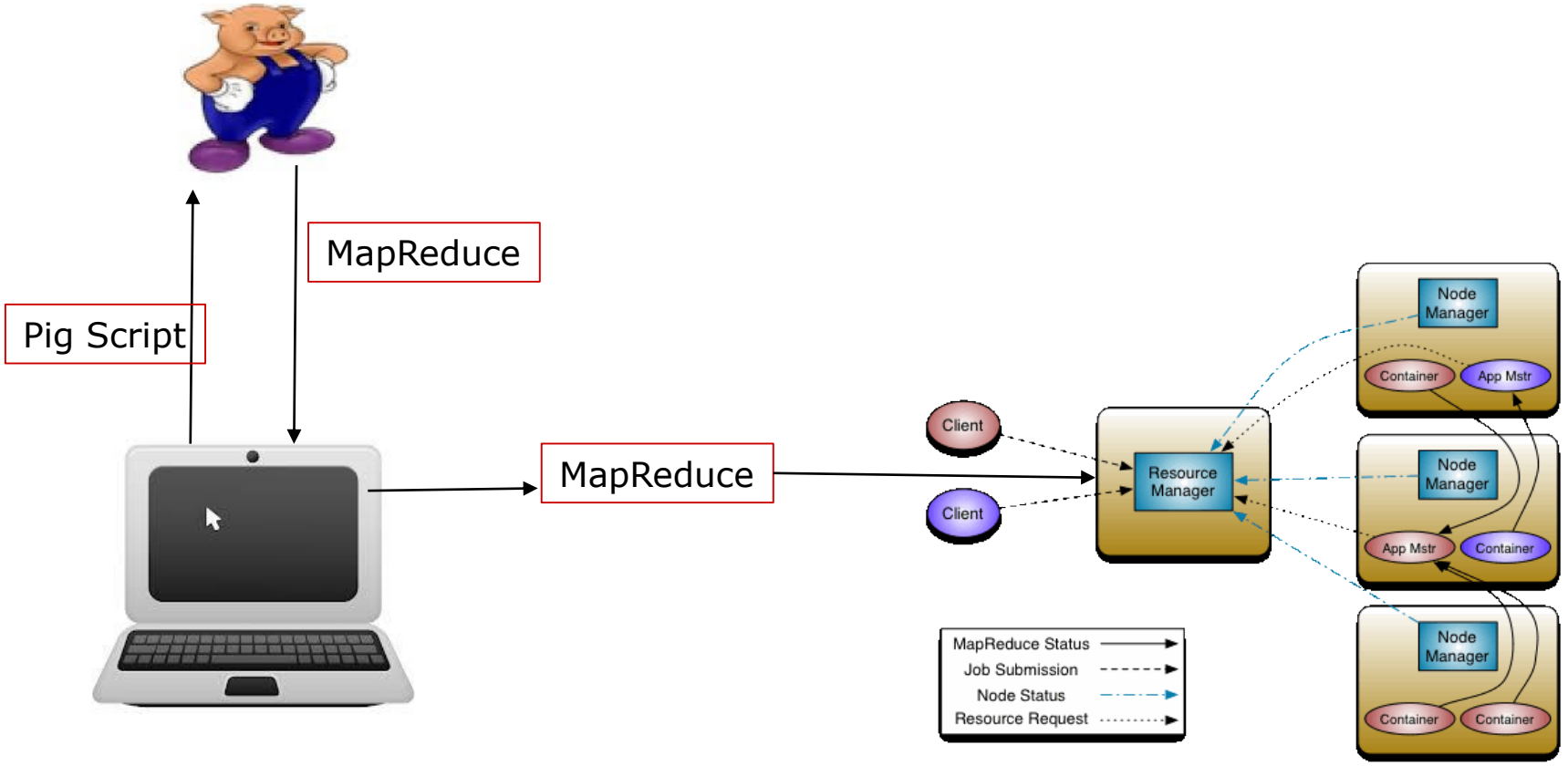
```
<workflow-app name='invertedindex-wf' xmlns="uri:oozie:workflow:0.1">
<start to='invertedindex'/>
        <action name='invertedindex '>
                <map-reduce>
                        <job-tracker>${jobTracker}</job-tracker>
                        <name-node>${nameNode}</name-node>
                <configuration>
                        <property>
                        <name>mapred.mapper.class</name>
                        <value>org.myorg.InvertedIndex.Map</value>
                        </property> .....
                </configuration>
                </map-reduce>
                        <ok to='end'/>
                        <error to='end'/>
                </action>
<end name='end'/>
</workflow-app>
```

Pig

What is Pig?

- Writing Mapreduce programs needs decent amount of]ava Knowledge.
- This entry point of learning Java would have restricted a lot of functional experts, who would need to query data.
- Here comes Pig Latin
- Pig Latin is a data flow language, with a simpler syntax
- Pig eventually gets converted to Mapreduce before it gets submitted to the Hadoop cluster
- Pig is ideal for Grouping, Filtering and Joins
- Also facilitates User defined functions

How Pig Works?



An Example

```
students= LOAD 'students.txt' AS (id, name, percentage);

distinction= FILTER students BY percentage> 75;

rankings = ORDER distinction BY percentage DESC;

STORE rankings INTO  'students_with_distinction
```

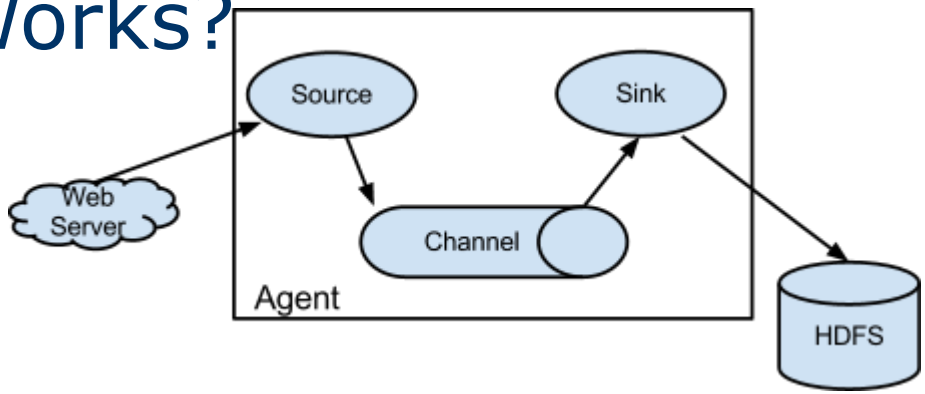
- In Pig, a Single data element is called as atom, collection of atoms is called as tuple, collection of tuples is called as bags
- Typically, Pig would load a dataset in a bag and then creates new bags by modifying the existing ones
- In the example above, the students file is loaded in students bag, then a new bag called distinction is created for students above 7 5%
- Later a new bag **rankings** is created and finally the results are stored in a file called **students_ with_distinction**


FLUME

What is Flume?

- Flume is a service that basically lets you ingest data (typically file data)into HDFS
- Defined as, distributed, reliable, available service for moving large amount of data as it is produced
- Open source service, developed by Cloudera
- Goals:
  - Reliability
  - Scalability
  - Manageability
  - Extensibility

How Flume Works?



- **Event**: A byte payload with optional string headers that represent the unit of data that Flume can transport from it's point of origination to it's final destination.
- **Flow:** Movement of events from the paint of origin to their final destination is considered a data flow, or simply flow. This is not a rigorous definition and is used only at a high level for description purposes.
- **Client:** An interface implementation that operates at the point of origin of events and delivers them to a Flume agent. Clients typically operate in the process space of the application they are consuming data from. For example, Flume Log4j Appender is a client.

How Flume Works?

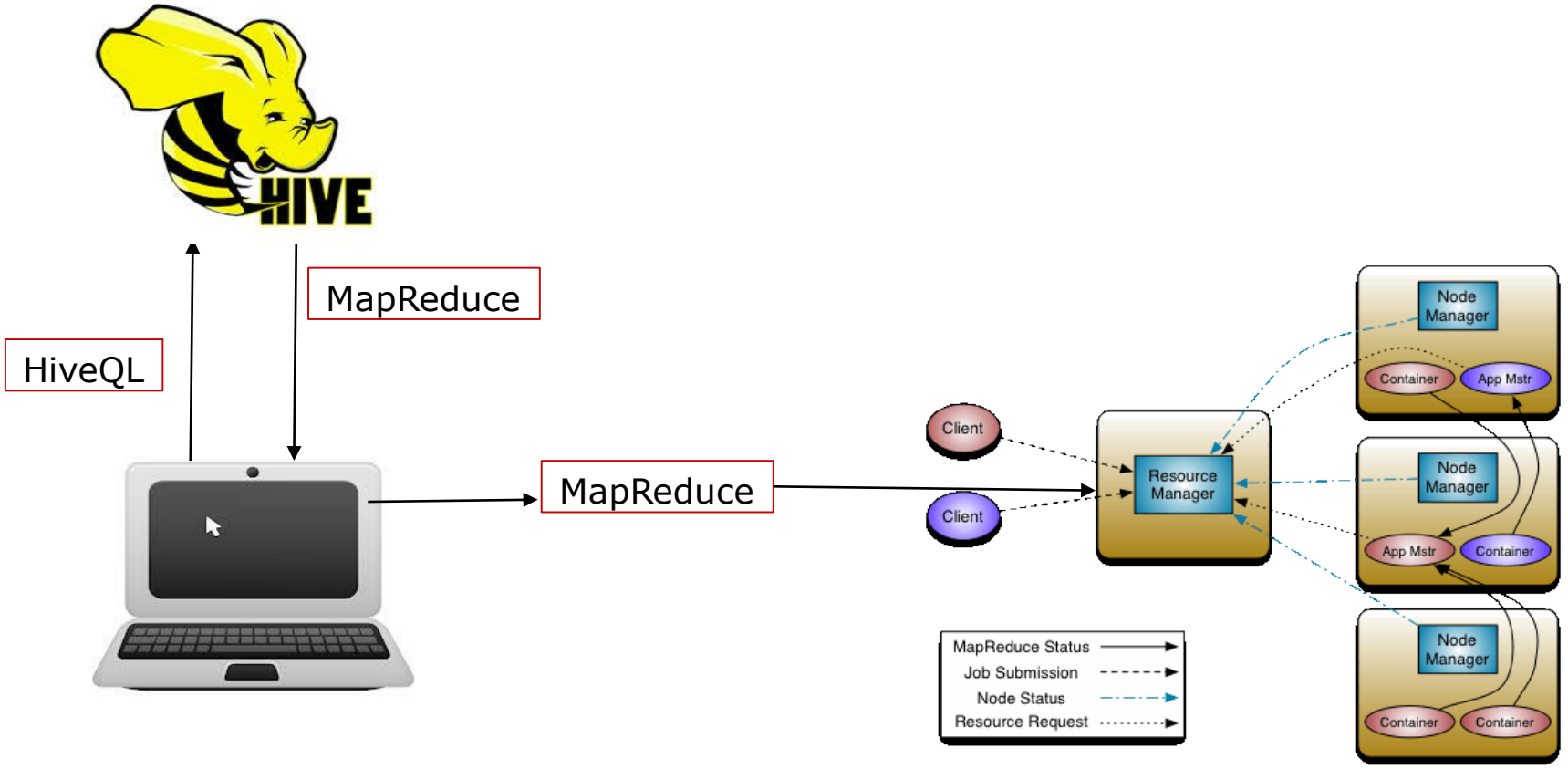
- **Agent:** An independent process that hosts flume components such as sources, channels and sinks, and thus has the ability to receive, store and forward events to their next-hop destination.
- **Source**: An interface implementation that can consume events delivered to it via a specific mechanism. For example, an Avro source is a source implementation that can be used to receive Avro events from clients or other agents in the flow. When a source receives an event, it hands it over to one or more channels.
- **Channel**: A transient store for events, where events are delivered to the channel via sources operating within the agent. An event put in a channel stays in that channel until a sink removes it for further transport. An example of channel is the JDBC channel that uses a file-system backed embedded database to persist the events until they are removed by a sink. Channels play an important role in ensuring durability of the flows.
- **Sink**: An interface implementation that can remove events from a channel and transmit them to the next agent in the flow, or to the event's final destination. Sinks that transmit the event to it's final destination are also known as terminal sinks. The Flume HDFS sink is an example of a terminal sink. Whereas the Flume Avro sink is an example of a regular sink that can transmit messages to other agents that are running an Avro source.


HIVE

What is Hive?

- Developed at Facebook.
- Provides a SQL like language
- Like Pig, it generates MapReduce jobs that run on the Hadoop cluster
- Hive has a layer of table definitions on top of HDFS, the table definitions are stored on the client machine. Table definitions storage is referred as metastore.
- Supports primitive types like INT, BIGINT, DOUBLE etc.
- Actual data is stored in flat files

How Hive Works?



An Example

```
hive> select * from students;
```

```
hive> show tables;
```

```
hive> describe students;
```

Quiz

Which of the below apache system deals with ingesting streaming data to hadoop

A - Oozie

B - Kafka

C - Flume

D - Hive

Quiz

Which technology is used to store data in Hadoop?

A - HBase

B - Avro

C - Sqoop

D - Zookeeper

Quiz

Q. A file of 1GB is subjected to wordcount job with 4 reducers and it emits 4 reduced output files. If I want to combine all these reduced files into one output, which of the following technologies would help:

1. ZooKeeper
2. Oozie
3. Flume
4. Hive

Apache Pig

Agenda

- **Pig Overview**
- **Execution Modes**
- **Installation**
- **Pig Latin Basics**
- **Developing Pig Script**
  - **Most Occurred Start Letter**
- **Resources**

Pig

- "is a platform for analysing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. "
- **Top Level Apache Project**
  - http://pig.apache.org
- **Pig is an abstraction on top of Hadoop**
  - Provides high level programming language designed for data processing
  - Converted into MapReduce and executed on Hadoop Clusters
- **Pig is widely accepted and used**
  - Yahoo!, Twitter, Netflix, etc...

Pig and MapReduce

- **MapReduce requires programmers**
  - Must think in terms of map and reduce functions
  - More than likely will require Java programmers
- **Pig provides high-level language that can be used by**
  - Analysts
  - Data Scientists
  - Statisticians
  - Etc...
- **Originally implemented at Yahoo! to allow analysts to access data**

Pig's Features

- Join Datasets
- Sort Datasets
- Filter
- Data Types
- Group By
- User Defined Functions
- Etc..

Pig's Use Cases

- **Extract Transform Load (ETL)**
  - Ex: Processing large amounts of log data
    - clean bad entries, join with other data-sets
- **Research of "raw" information**
  - Ex. User Audit Logs
  - Schema maybe unknown or inconsistent
  - Data Scientists and Analysts may like Pig's data transformation paradigm

Pig Components

- **Pig Latin**
  - Command based language
  - Designed specifically for data transformation and flow expression
- **Execution Environment**
  - The environment in which Pig Latin commands are executed
  - Currently there is support for Local and Hadoop modes
- **Pig compiler converts Pig Latin to MapReduce**
  - Compiler strives to optimize execution
  - You automatically get optimization improvements with Pig updates

Execution Modes

- **Local**
  - Executes in a single JVM
  - Works exclusively with local file system
  - Great for development, experimentation and prototyping
- **Hadoop Mode**
  - Also known as MapReduce mode
  - Pig renders Pig Latin into MapReduce jobs and executes them on the cluster
  - Can execute against semi-distributed or fully-distributed hadoop installation
    - We will run on pseudo-distributed cluster

Installation Prerequisites

- Java 6 or Higher
  - With $JAVA_HOME environment variable properly set
- Cygwin on Windows

Installation

- **Add pig script to path**
  - export PIG_HOME=/usr/local/pig
  - export PATH=$PATH:$PIG_HOME/bin
- **$ pig -help**
- **That's all we need to run in local mode**
  - Think of Pig as a 'Pig Latin' compiler, development tool and executor
  - Not tightly coupled with Hadoop clusters

Pig Installation for Hadoop Mode

- **Make sure Pig compiles with Hadoop**
  - Not a problem when using a distribution such as Cloudera Distribution for Hadoop (CDH)
- **Pig will utilize $HADOOP_HOME and**
- **$HADOOP_CONF_DIR variables to locate Hadoop configuration**
  - We already set these properties during MapReduce installation
  - Pig will use these properties to locate Namenode and Resource Manager

Running Modes

- **Can manually override the default mode via '-x' or '-exectype' options**
  - $pig -x local
  - $pig -x mapreduce
- **$ pig**

  2012-07-14 13:38:58,139 [main] INFO org.apache.pig.Main - Logging error messages to: /home/hadoop/Training/play_area/pig/pig_1342287538128.log
  2012-07-14 13:38:58,458 [main] INFO org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: **hdfs://localhost:8020**

- **$ pig -x local**

  2012-07-14 13:39:31,029 [main] INFO org.apache.pig.Main - Logging error messages to: /home/hadoop/Training/play_area/pig/pig_1342287571019.log
  2012-07-14 13:39:31,232 [main]

  INFO org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: **file:///**

Running Pig

- **Script**
  - **Execute commands in a file**
  - $pig scriptFile.pig
- **Grunt**
  - Interactive Shell for executing Pig Commands
  - Started when script file is NOT provided
  - Can execute scripts from Grunt via run or exec commands
- **Embedded**
  - Execute Pig commands using PigServer class
    - Just like JDBC to execute SQL
  - Can have programmatic access to Grunt via PigRunner class

Pig Latin Concepts

- **Building blocks**
  - Field – piece of data
  - Tuple – ordered set of fields, represented with "(" and ")"

    (10.4, 5, word, 4, field1)
  - Bag – collection of tuples, represented with "{" and "}"

    { (10.4, 5, word, 4, field1), (this, 1, blah) }
- **Similar to Relational Database**
  - Bag is a table in the database
  - Tuple is a row in a table
  - Bags do not require that all tuples contain the same number
    - Unlike relational table

DUMP and STORE statements

- **No action is taken until DUMP or STORE commands are encountered**
  - **Pig will parse, validate and analyze statements but not execute them**
- **DUMP – displays the results to the screen**
- **STORE – saves results (typically to a file)**

Nothing is executed; Pig will optimize this entire chunk of script

```
records = LOAD '/training/playArea/pig/a.txt' as
(letter:chararray, count:int);
...
...
...
...
...

DUMP final_bag;
```

Fun begins here!

Large Data

- **Hadoop data is usually quite large and it doesn't make sense to print it to the screen**
- **The common pattern is to persist results to Hadoop (HDFS, HBase)**
  - **This is done with STORE command**
- **For information and debugging purposes you can print a small sub-set to the screen**

```
grunt> records = LOAD '/training/playArea/pig/excite-small.log'
AS (userId:chararray, timestamp:long, query:chararray);
grunt> toPrint = LIMIT records 5;
grunt> DUMP toPrint;
```

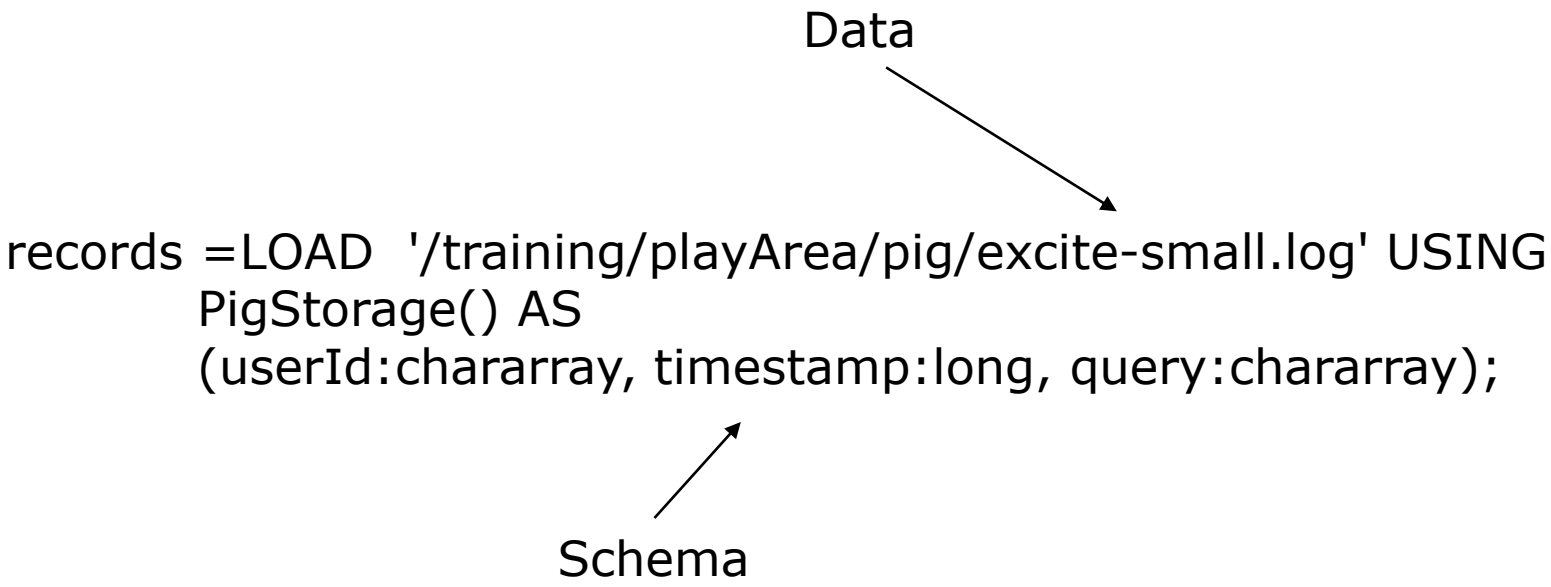
Only 5 records will be displayed

LOAD Command

```
LOAD 'data' [USING function] [AS schema];
```

- **data – name of the directory or file**
  - **Must be in single quotes**
- **USING – specifies the load function to use**
  - By default uses **PigStorage** which parses each line into fields using a delimiter
    - Default delimiter is tab (‘\t’)
    - The delimiter can be customized using regular expressions
- **AS – assign a schema to incoming data**
  - Assigns names to fields
  - Declares types to fields

LOAD Command Example



User selected Load Function,
there are a lot of choices or you
can implement your own

Schema Data Types

Type	Description	Example
	Simple	
int	Signed 32-bit integer	10
long	Signed 64-bit integer	10L or 10l
float	32-bit floating point	10.5F or 10.5f
double	64-bit floating point	10.5 or 10.5e2 or 10.5E2
	Arrays	
chararray	Character array (string) in Unicode UTF-8	hello world
bytearray	Byte array (blob)	
	Complex Data Types	
tuple	An ordered set of fields	(19,2)
bag	An collection of tuples	{(19,2), (18,1)}
map	An collection of tuples	[open#apache]

Pig Latin - Grouping

```
grunt> chars = LOAD '/training/playArea/pig/b.txt' AS
(c:chararray);
grunt> describe chars;
chars: {c: chararray}
grunt> dump chars;
(a)
(k)
...
...
(k)
(c)
(k)
grunt> charGroup = GROUP chars by c;
grunt> describe charGroup;
charGroup: {group: chararray,chars: {(c: chararray)}}
grunt> dump charGroup;
(a,{(a),(a),(a)})
(c,{(c),(c)})
(i,{(i),(i),(i)})
(k,{(k),(k),(k),(k)})
(l,{(l),(l)})
```

Creates a new bag with element named *group* and element named *chars*

The chars bag is grouped by "c"; therefore 'group' element will contain unique values

'*chars*' element is a bag itself and contains all tuples from '*chars*' bag that match the value form 'c'

Pig Latin - FOREACH

- **FOREACH <bag> GENERATE <data>**
  - Iterate over each element in the bag and produce a result
  - Ex: grunt> result = FOREACH bag GENERATE f1;

```
grunt> records = LOAD 'data/a.txt' AS (c:chararray, i:int);
grunt> dump records;
    (a,1)
    (d,4)
    (c,9)
    (k,6)
grunt> counts = foreach records generate i;
grunt> dump counts; (1)
(4)
(9)
(6)
```

FOREACH with Functions

- **FOREACH B GENERATE group, FUNCTION(A);**
  - Pig comes with many functions including COUNT, FLATTEN, CONCAT, etc...
  - Can implement a custom function

```
grunt> chars = LOAD 'data/b.txt' AS (c:chararray);
grunt> charGroup = GROUP chars by c;
grunt> dump charGroup; (a,{(a),(a),(a)})
(c,{(c),(c)})
(i,{(i),(i),(i)})
(k,{(k),(k),(k),(k)}) (l,{(l),(l)})
grunt> describe charGroup;
charGroup: {group: chararray,chars: {(c: chararray)}}
grunt> counts = FOREACH charGroup GENERATE group, COUNT(chars);
grunt> dump counts;
 (a,3)
(c,2)
(i,3)
(k,4)
(l,2)
```

Joins Overview

- **Critical Tool for Data Processing**
- **Will probably be used in most of your Pig scripts**
- **Pigs supports**
  - **Inner Joins**
  - Outer Joins
  - Full Joins

How to Join in Pig

- **Join Steps**
  - Load records into a bag from input #1
  - Load records into a bag from input #2
  - Join the 2 data-sets (bags) by provided join key
- **Default Join is Inner Join**
  - Rows are joined where the keys match
  - Rows that do not have matches are not included in the result

Simple Inner Join Example

1: Load records into a bag from input #1

```
posts = load '/training/data/user-posts.txt' using PigStorage(',') as
(user:chararray,post:chararray,date:long);
```

2:Load records into a bag from input #2

Use comma as a separator

```
likes = load '/training/data/user-likes.txt' using PigStorage(',') as
(user:chararray,likes:int,date:long);
```

3: Join the 2 data-sets

```
userInfo = join posts by user, likes by user;

dump userInfo;
```

Execute InnerJoin.pig

$ hdfs dfs -cat /training/data/user-posts.txt

user1,Funny Story,1343182026191

user2,Cool Deal,1343182133839

user4,Interesting Post,1343182154633

user5,Yet Another Blog,13431839394

$ hdfs dfs -cat /training/data/user-likes.txt

user1,12,1343182026191 user2,7,1343182139394 user3,0,1343182154633
user4,50,1343182147364

$ pig $PLAY_AREA/pig/scripts-samples/InnerJoin.pig

(user1,Funny Story,1343182026191,user1,12,1343182026191)

(user2,Cool Deal,1343182133839,user2,7,1343182139394)

(user4,Interesting Post,1343182154633,user4,50,1343182147364)

Outer Joins

- **Records which will not join with the ‘other’ record-set are still included in the result**
- **Left Outer**
  - Records from the first data-set are included whether they have a match or not. Fields from the unmatched (second) bag are set to null.
- **Right Outer**
  - The opposite of Left Outer Join: Records from the second data-set are included no matter what. Fields from the unmatched (first) bag are set to null.
- **Full Outer**
  - Records from both sides are included. For unmatched records the fields from the ‘other’ bag are set to null.

Left Outer Join Example

--LeftOuterJoin.pig

```
posts = load '/training/data/user-posts.txt' using
PigStorage(',')

as (user:chararray,post:chararray,date:long);

 likes = load '/training/data/user-likes.txt' using
PigStorage(',')

as (user:chararray,likes:int,date:long);

 userInfo = join posts by user left outer, likes by
user;

dump userInfo;
```

Execute LeftOuterJoin.pig

$ hdfs dfs -cat /training/data/user-posts.txt

user1,Funny Story,1343182026191
user2,Cool Deal,1343182133839
user4,Interesting Post,1343182154633
user5,Yet Another Blog,13431839394

$ hdfs dfs -cat /training/data/user-likes.txt

user1,12,1343182026191
user2,7,1343182139394
user3,0,1343182154633
user4,50,1343182147364

$ **pig $PLAY_AREA/pig/scripts/LeftOuterJoin.pig**

(user1,Funny Story,1343182026191,user1,12,1343182026191)
(user2,Cool Deal,1343182133839,user2,7,1343182139394)

(user4,Interesting Post,1343182154633,user4,50,1343182147364)
(user5,Yet Another Blog,13431839394,,,)

Right Outer

```
--RightOuterJoin.pig

posts = LOAD '/training/data/user-posts.txt' USING
PigStorage(',')

AS (user:chararray,post:chararray,date:long);

likes = LOAD '/training/data/user-likes.txt'

USING PigStorage(',')

AS (user:chararray,likes:int,date:long);

userInfo = JOIN posts BY user RIGHT OUTER, likes BY
user;

DUMP userInfo;
```

Full Join

```
--FullOuterJoin.pig

posts = LOAD '/training/data/user-posts.txt' USING
PigStorage(',')

AS (user:chararray,post:chararray,date:long);

likes = LOAD '/training/data/user-likes.txt'

USING PigStorage(',')

AS (user:chararray,likes:int,date:long);

userInfo = JOIN posts BY user FULL OUTER, likes BY user;

DUMP userInfo;
```

User Defined Function (UDF)

- **There are times when Pig's built in operators and functions will not suffice**
- **Pig provides ability to implement your own**
  - **Filter**
    - Ex: res = FILTER bag BY udfFilter(post);
  - Load Function
    - Ex: res = load 'file.txt' using udfLoad();
  - Eval
    - Ex: res = FOREACH bag GENERATE udfEval($1)
- **Choice between several programming languages**
  - Java, Python, Javascript

Implement Custom Filter Function

- **Our custom filter function will remove records with the provided value of more than 15 characters**
  - filtered = FILTER posts BY isShort(post);
- **Simple steps to implement a custom filter**
- Extend FilterFunc class and implement exec method
- Register JAR with your Pig Script
  - JAR file that contains your implementation
- Use custom filter function in the Pig script

1: Extend FilterFunc

- **FilterFunc class extends EvalFunc**
  - Customization for filter functionality
- **Implement exec method**
  - public Boolean exec(Tuple tuple) throws IOException
  - Returns false if the tuple needs to be filtered out and true otherwise
  - Tuple is a list of ordered fields indexed from 0 to N
    - We are only expecting a single field within the provided tuple
    - To retrieve fields use tuple.get(0);

1: Extend FilterFunc

```
public class IsShort extends FilterFunc {
private static final int MAX_CHARS = 15;

@Override
public Boolean exec(Tuple tuple) throws IOException {
if (tuple == null || tuple.isNull() || tuple.size() == 0) {
return false;

}
Object obj = tuple.get(0);
if (obj instanceof String) {
String st = (String) obj;
if (st.length() > MAX_CHARS) {
return false;
}
return true;
}
return false;
}
}
```

2: Register JAR with Pig Script

- **Compile your class with filter function and package it into a JAR file**
- **Utilize REGISTER operator to supply the JAR file to your script**
- REGISTER HadoopSamples.jar
  - **The local path to the jar file**
  - Path can be either absolute or relative to the execution location
  - Path must NOT be wrapped with quotes
  - Will add JAR file to Java's CLASSPATH

3: Use Custom Filter Function in the Pig Script

- **Pig locates functions by looking on CLASSPATH for fully qualified class name**

```
filtered = FILTER posts BY pig.IsShort(post);
```

- **Pig will properly distribute registered JAR and add it to the CLASSPATH**
- **Can create an alias for your function using DEFINE operator**

```
DEFINE isShort pig.IsShort();
...
...
filtered = FILTER posts BY isShort(post);
...
```

Script with Custom Function

```
--CustomFilter.pig

REGISTER HadoopSamples.jar

DEFINE isShort pig.IsShort();

posts = LOAD '/training/data/user-posts.txt' USING
PigStorage(',')

AS (user:chararray,post:chararray,date:long);


filtered = FILTER posts BY isShort(post);

dump filtered;
```

Execute CustomFilter.pig

```
$ hdfs dfs -cat /training/data/user-posts.txt

user1,Funny Story,1343182026191
user2,Cool Deal,1343182133839
user4,Interesting Post,1343182154633
user5,Yet Another Blog,13431839394

$ pig pig/scripts/CustomFilter.pig

(user1,Funny Story,1343182026191)

(user2,Cool Deal,1343182133839)
```

Quiz

Quiz

Quiz

Apache HIVE

Agenda

- **Hive Overview and Concepts**
- **Installation**
- **Table Creation and Deletion**
- **Loading Data into Hive**
- **Partitioning**
- **Bucketing**
- **Joins**

HIVE

- **Data Warehousing Solution built on top of Hadoop**
- **Provides SQL-like query language named HiveQL**
  - Minimal learning curve for people with SQL expertise
  - Data analysts are target audience
- **Early Hive development work started at Facebook in 2007**
- **Today Hive is an Apache project under Hadoop**
  - http://hive.apache.org

HIVE Provides

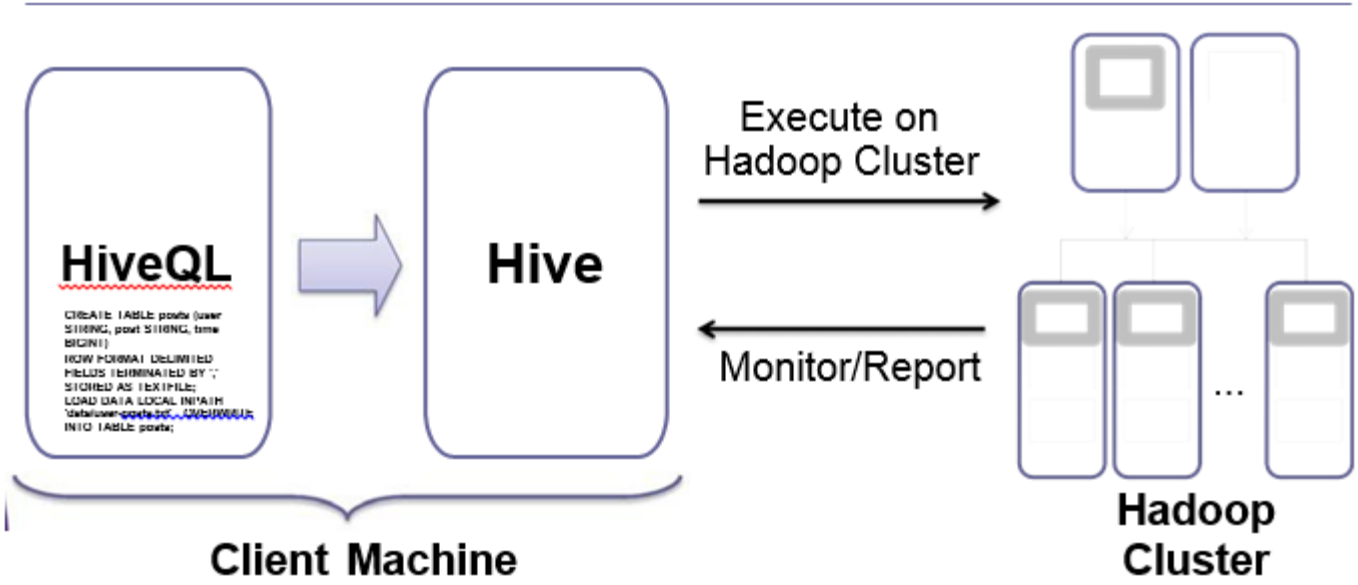
- **Ability to bring structure to various data formats**
- **Simple interface for ad hoc querying, analyzing and summarizing large amounts of data**
- **Access to files on various data stores such as HDFS and HBase**

Hive

- **Hive does NOT provide low latency or real-time queries**
- **Even querying small amounts of data may take minutes**
- **Designed for scalability and ease-of-use rather than low latency responses**

Hive

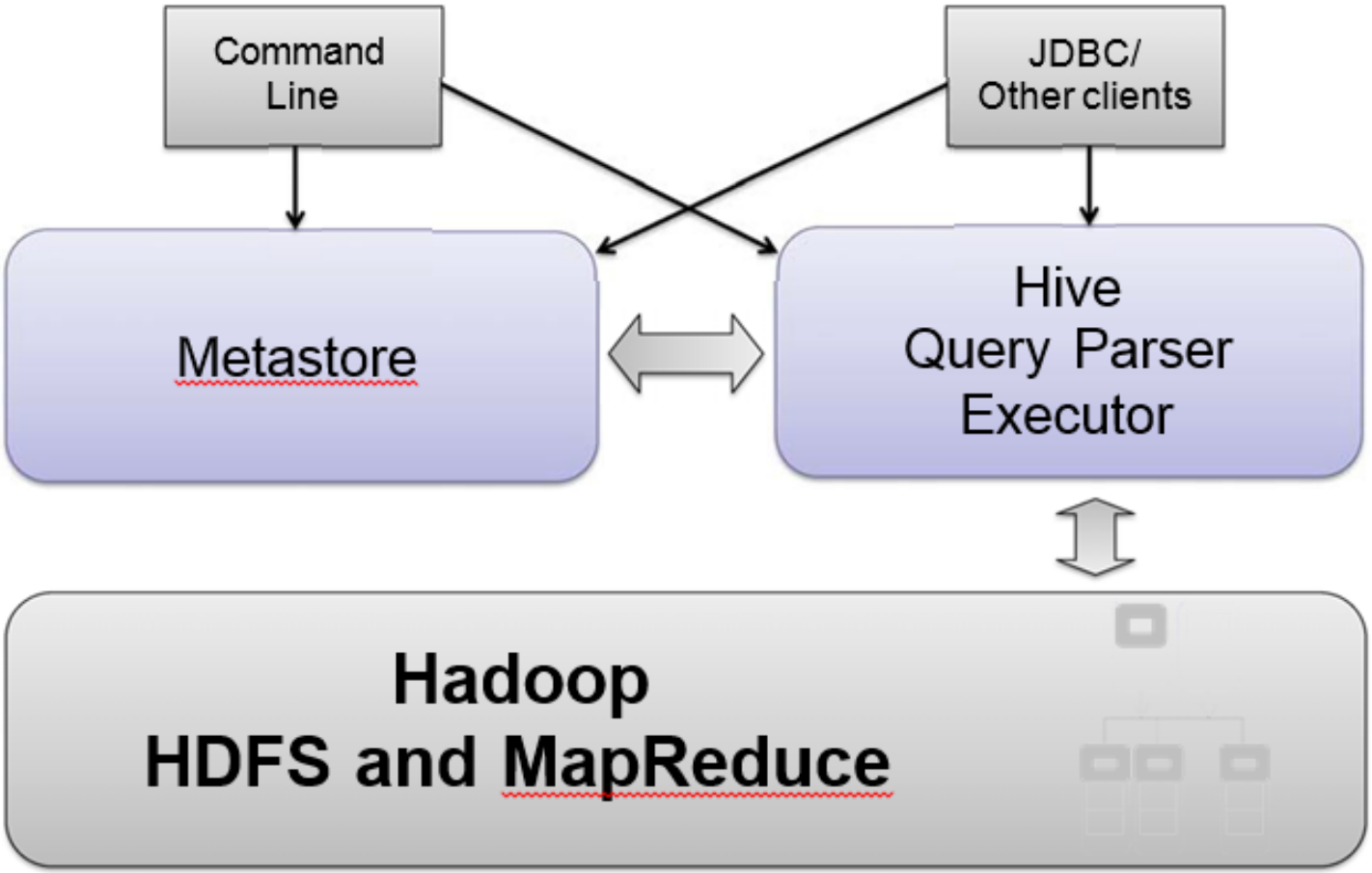
- **Translates HiveQL statements into a set of MapReduce Jobs which are then executed on a Hadoop Cluster**



Hive Metastore

- **To support features like schema(s) and data partitioning Hive keeps its metadata in a Relational Database**
  - Packaged with Derby, a lightweight embedded SQL DB
    - Default Derby based is good for evaluation an testing
    - Schema is not shared between users as each user has their own instance of embedded Derby
    - Stored in metastore_db directory which resides in the directory that hive was started from
  - Can easily switch another SQL installation such as MySQL

Hive Architecture

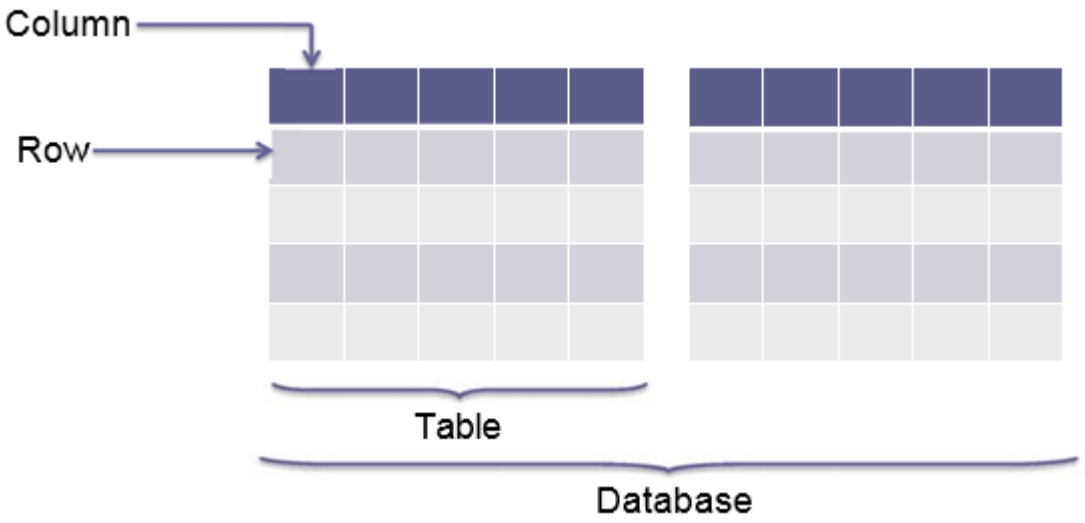


Hive Interface Options

- **Command Line Interface (CLI)**
  - Will use exclusively in these slides
- **Hive Web Interface**
  - https://cwiki.apache.org/confluence/display/Hive/HiveWebInterface
- **Java Database Connectivity (JDBC)**
  - https://cwiki.apache.org/confluence/display/Hive/HiveClient

Hive Concepts

- **Re-used from Relational Databases**
  - **Database**: Set of Tables, used for name conflicts resolution
  - **Table**: Set of Rows that have the same schema (same columns)
  - **Row**: A single record; a set of columns
  - **Column**: provides value and type for a single value



Run Hive

- HDFS and MapReduce Must be running

```
$ hive

Hive  history
file=/tmp/hadoop/hive_job_log_hadoop_201207312052_1402761030.txt

hive>
```

Simple Example

1. **Create a Table**
2. **Load Data into a Table**
3. **Query Data**
4. **Drop the Table**

1. Create a Table

- **Let's create a table to store data from $PLAY_AREA/data/user-posts.txt**

$ cd $PLAY_AREA

$ **hive**

Hive history file=/tmp/hadoop/hive_job_log_hadoop_201208022144_2014345460.txt

hive> **!cat data/user-posts.txt;**

```
user1,Funny Story,1343182026191
user2,Cool Deal,1343182133839
user4,Interesting  Post,1343182154633
user5,Yet Another Blog,13431839394
```

hive>

1. Create a Table

hive> CREATE TABLE posts (user STRING, post STRING, time BIGINT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',` STORED AS TEXTFILE;

hive> **show tables;**

OK

posts

Time taken: 0.221 seconds

hive> **describe posts;**

OK

user **string**

post **string**

time **bigint**

Time taken: 0.212 seconds

2: Load Data Into a Table

hive> **LOAD DATA LOCAL INPATH 'data/user-posts.txt'**

OVERWRITE **INTO TABLE posts;**

$ **hdfs dfs -cat /user/hive/warehouse/posts/user-posts.txt**

user1,Funny Story,1343182026191
user2,Cool Deal,1343182133839
user4,Interesting Post,1343182154633
user5,Yet Another Blog,13431839394

Under the hood Hive stores the data under **warehouse** folder

3: Query the data

hive> **select count (1) from posts;**

hive> **select * from posts where user="user2";**

hive> **select * from posts where time<=1343182133839 limit 2;**

4:Drop the Table

hive> **DROP TABLE posts;**

OK

Time taken: 2.182 seconds

hive> **exit;**

$ **hdfs dfs -ls /user/hive/warehouse/**

(The files are removed now!)

Loading Data

- **Several options to start using data in HIVE**
  - Load data from HDFS location

```
hive> LOAD DATA INPATH '/training/hive/user-posts.txt'
        > OVERWRITE INTO TABLE posts;
```

File is copied from the provided location to /user/hive/warehouse/ (or configured location)

- **Load data from a local file system**

```
hive> LOAD DATA LOCAL INPATH 'data/user-posts.txt'
        > OVERWRITE INTO TABLE posts;
```

- File is copied from the provided location to /user/hive/warehouse/ (or configured location)
  - Utilize an existing location on HDFS
    - Just point to an existing location when creating a table

Re-Use Existing HDFS Location

- hive> CREATE **EXTERNAL** TABLE posts

> (user STRING, post STRING, time BIGINT)

> ROW FORMAT DELIMITED

> FIELDS TERMINATED BY ','

> STORED AS TEXTFILE

> **LOCATION '/training/hive/'**;

OK

Time taken: 0.077 seconds

hive>

Partitions

- **To increase performance Hive has the capability to partition data**
  - The values of partitioned column divide a table into segments
  - Entire partitions can be ignored at query time
  - Similar to relational databases' indexes but not as granular
- **Partitions have to be properly crated by users**
  - When inserting data must specify a partition
- **At query time, whenever appropriate, Hive will automatically filter out partitions**

Creating Partitioned Table

```
hive> CREATE TABLE posts (user STRING, post STRING, time BIGINT)
> PARTITIONED BY(country STRING)
> ROW FORMAT DELIMITED

> FIELDS TERMINATED BY ','
> STORED AS TEXTFILE;
OK

hive> describe posts;
OK
user        string
post        string
time        bigint countrystring
hive> show partitions posts;
OK
hive>
```

Load data into Partitioned Table

```
hive> LOAD DATA LOCAL INPATH 'data/user-posts-US.txt'
> OVERWRITE INTO TABLE posts;
FAILED: Error in semantic analysis: Need to specify partition columns because the destination table is partitioned

hive> LOAD DATA LOCAL INPATH 'data/user-posts-US.txt'
> OVERWRITE INTO TABLE posts PARTITION(country='US');
OK

hive> LOAD DATA LOCAL INPATH 'data/user-posts-AUSTRALIA.txt'
> OVERWRITE INTO TABLE posts PARTITION(country='AUSTRALIA');
OK
hive>
```

Partitioned Table

- Partitions are physically stored under separate directories

hive> show partitions posts;

OK

country=AUSTRALIA country=US

Time taken: 0.095 seconds hive> exit;

There is a directory for each partition value

$ hdfs dfs -ls -R /user/hive/warehouse/posts

/user/hive/warehouse/posts/country=AUSTRALIA

/user/hive/warehouse/posts/country=AUSTRALIA/user-posts-AUSTRALIA.txt

/user/hive/warehouse/posts/country=US

/user/hive/warehouse/posts/country=US/user-posts-US.txt

Querying Partitioned Table

- **There is no difference in syntax**
- **When partitioned column is specified in the where clause entire directories/partitions could be ignored**
- Only "COUNTRY=US" partition will be queried, "COUNTRY=AUSTRALIA" partition will be ignored

hive> **select * from posts where country='US' limit 10;**

OK

user1 Funny Story 1343182026191 US

user2 Cool Deal 1343182133839 US

user2 Great Interesting Note 13431821339485 US

user4 Interesting Post 1343182154633 US

user1 Humor is good 1343182039586 US

user2 Hi I am user #2 1343182133839 US

Joins

- **Joins in Hive are trivial**
- **Supports outer joins**
  - left, right and full joins
- **Can join multiple tables**
- **Default Join is Inner Join**
  - Rows are joined where the keys match
  - Rows that do not have matches are not included in the result

Simple Inner Join

```
hive> select * from posts limit 10;

OK

user1  Funny Story  1343182026191
user2    Cool Deal     1343182133839
user4 Interesting Post 1343182154633
user5 Yet Another Blog 1343183939434


hive> select * from likes limit 10;

OK

user1,  12,  1343182026191
user2,  7,    1343182139394
user3,  0,    1343182154633
user4, 50,   1343182147364
```

We want to join these 2 data-sets and produce a single table that contains user, post and count of likes

```
hive> CREATE TABLE posts_likes (user STRING, post STRING, likes_count INT);

 OK
```

Simple Inner Join

```
hive> INSERT OVERWRITE TABLE posts_likes
> SELECT p.user, p.post, l.count
> FROM posts p JOIN likes l ON (p.user = l.user);
OK
```

Two tables are joined based on user column;

3 columns are selected and stored in posts_likes table

```
hive> select * from posts_likes limit 10;
OK
user1 Funny Story          12
user2 Cool Deal            7
user4 Interesting Post     50
hive>
```

Outer Join

- **Rows which will not join with the ‘other’ table are still included in the result**
- **Left Outer**
  - Row from the first table are included whether they have a match or not. Columns from the unmatched (second) table are set to null.
- **Right Outer**
  - The opposite of Left Outer Join: Rows from the second table are included no matter what. Columns from the unmatched (first) table are set to null.
- **Full Outer**
  - Rows from both sides are included. For unmatched rows the columns from the ‘other’ table are set to null.

Outer Join Examples

SELECT p.*, l.*

FROM posts p **LEFT OUTER JOIN** likes l ON (p.user = l.user) limit 10;

SELECT p.*, l.*

FROM posts p **RIGHT OUTER JOIN** likes l ON (p.user = l.user) limit 10;

SELECT p.*, l.*

FROM posts p **FULL OUTER JOIN** likes l ON (p.user = l.user) limit 10;

Quiz

Q. Each database created in hive is stored as

A - a directory

B - a file

C - a hdfs block

D - a jar file

Quiz

Q. On dropping an external table

A - The schema gets dropped without dropping the data

B - The data gets dropped without dropping the schema

C - An error is thrown

D - Both the schema and the data is dropped

Quiz

Q. Every Hive installation starts a local instance of Derby Database. It is used to store the metadata for hive. Which is called hive metastore. It is created in

A. The location from where hive was started for the first time

B. Under $HIVE_HOME/conf

C. Under $HIVE_HOME/bin

D. Under /var/lib/hive

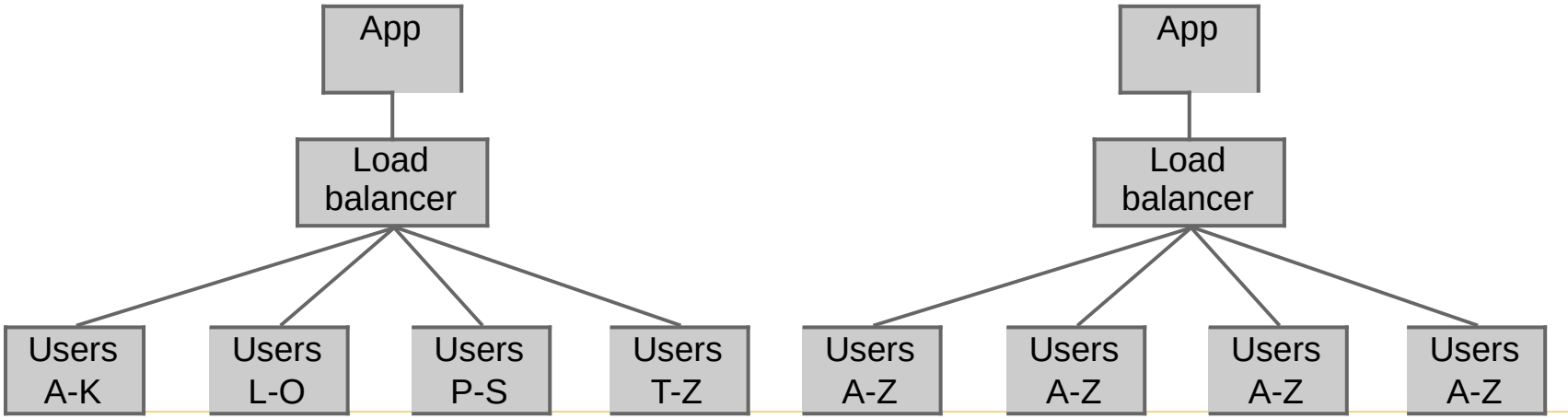
Introduction to NoSQL

What is NoSQL?

- Class of database management systems (DBMS)
- "*Not only SQL*"
  - Does not use SQL as querying language
  - Distributed, fault-tolerant architecture
  - No fixed schema (formally described structure)
  - No joins (typical in databases operated with SQL)
    - Expensive operation for combining records from two or more tables into one set
    - Joins require strong consistency and fixed schemas
      - Lack of these makes NoSQL databases more flexible
- It's not a replacement for a RDBMS but compliments it

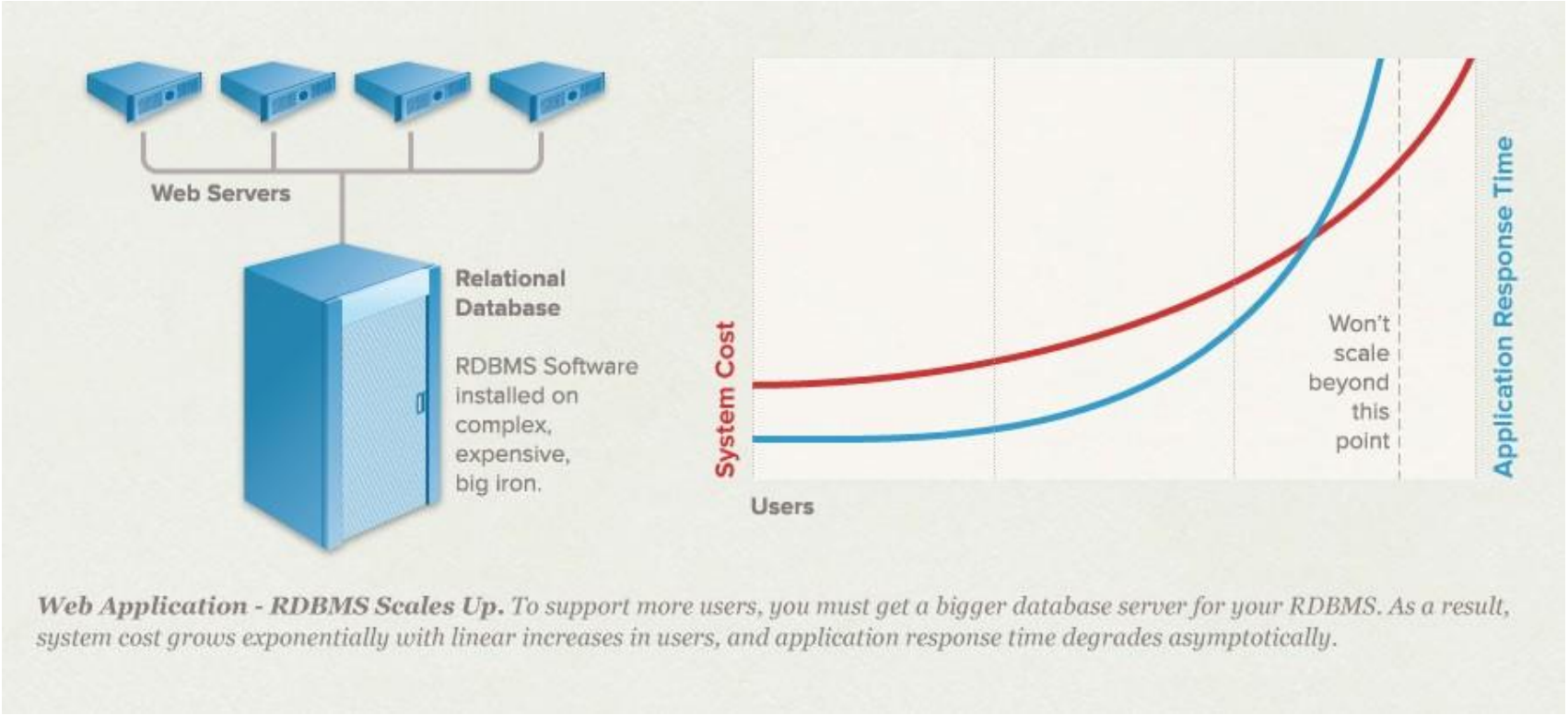
Database Scaling

- RDBMS are "scaled up" by adding hardware processing power
- NoSQL is "scaled out" by spreading the load
  - Partitioning (sharding) / replication



Relational DB Scaling

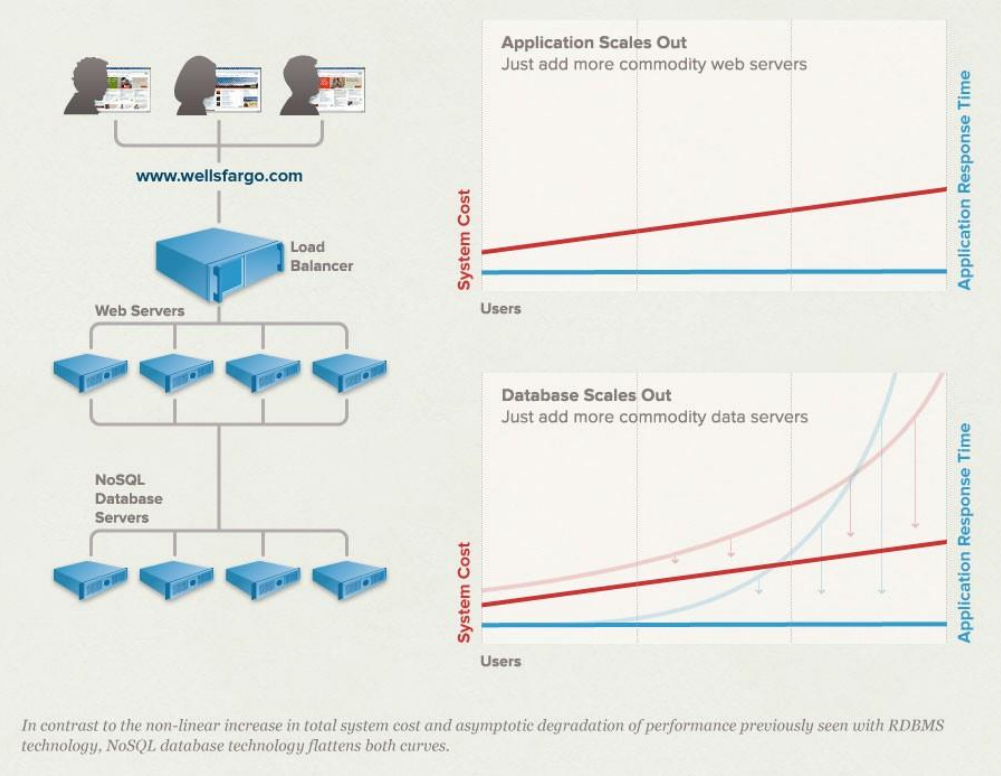
- At certain point relational database won't scale



Web Application - RDBMS Scales Up. *To support more users, you must get a bigger database server for your RDBMS. As a result, system cost grows exponentially with linear increases in users, and application response time degrades asymptotically.*

NoSQL DB Scaling

- Scaling horizontally is possible with NoSQL
- Scaling up / down is easy
  - Supports rapid production-ready prototyping
- Better handling of traffic spikes



Where NoSQL Is Used?

- Google (BigTable, LevelDB)
- LinkedIn (Voldemort)
- Facebook (Cassandra)
- Twitter (Hadoop/Hbase, FlockDB, Cassandra)
- Netflix (SimpleDB, Hadoop/HBase, Cassandra)
- CERN (CouchDB)

History of NoSQL

- MultiValue databases at TRW in **1965**.
- DBM is released by AT&T in **1979**.
- Lotus Domino released in **1989**.
- Carlo Strozzi used the term NoSQL in **1998** to name his lightweight, open-source relational database that did not expose the standard SQL interface.
- Graph database Neo4j is started in **2000**.
- Google BigTable is started in **2004**. Paper published in 2006.
- CouchDB is started in **2005**.
- The research paper on Amazon Dynamo is released in **2007**.
- The document database MongoDB is started in **2007** as a part of a open source cloud computing stack and first standalone release in 2009.
- Facebooks open sources the Cassandra project in **2008**.
- Project Voldemort started in **2008**.
- The term NoSQL was reintroduced in early **2009**.
- Some NoSQL conferences
  NoSQL Matters, NoSQL Now!, INOSA

CAP Theorem 1/2

- It is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:
  - Consistency (all nodes see the same data at the same time)
  - Availability (a guarantee that every request receives a response about whether it was successful or failed)
  - Partition tolerance (the system continues to operate despite arbitrary message loss or failure of part of the system)

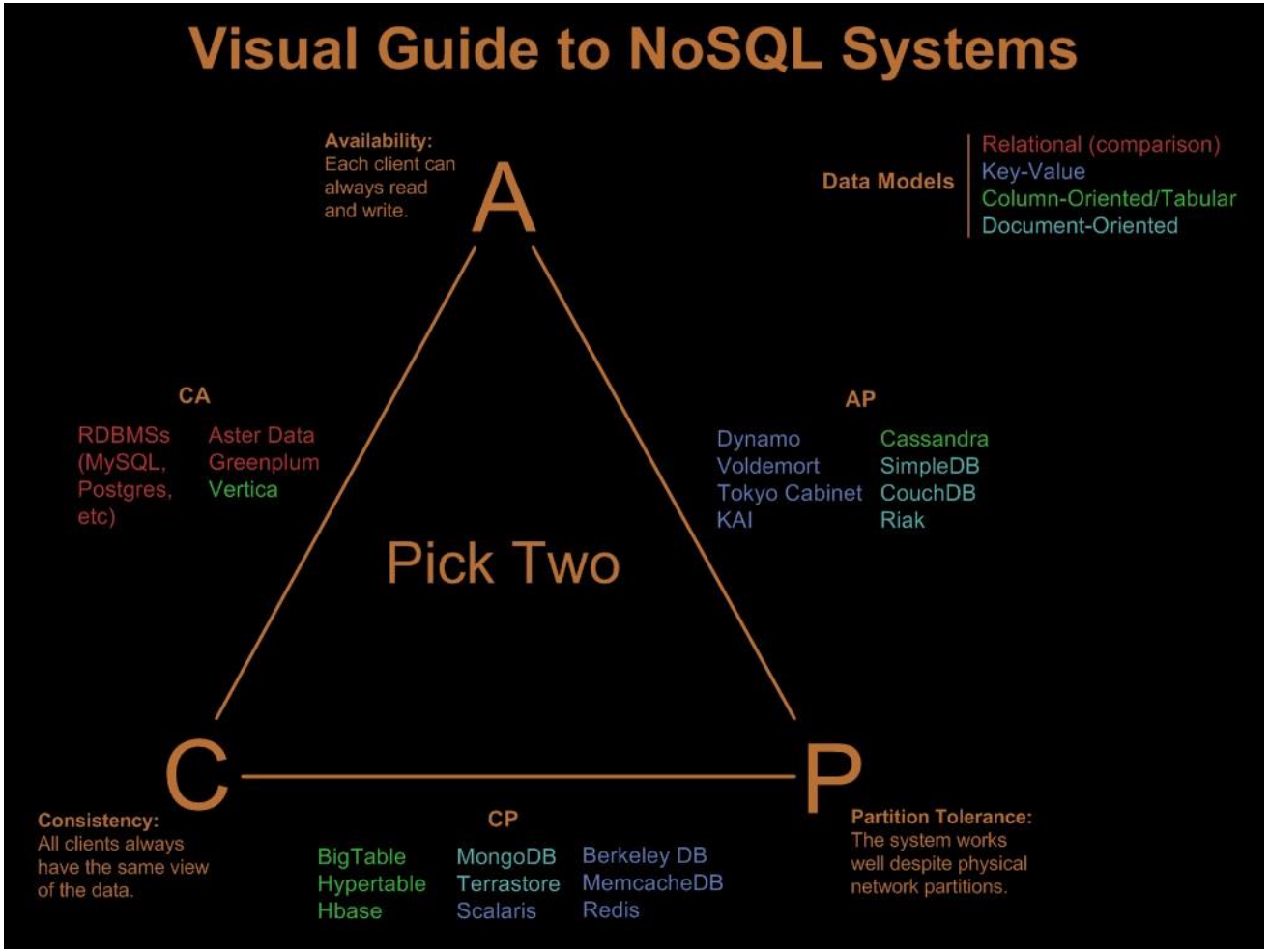
A distributed system can satisfy any two of these guarantees at the same time, but not all three.

Gilbert and Lynch, Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services, 2002

CAP Theorem 2/2

- In other words, CAP can be expressed as "If the network is broken, your database won't work"
  - "won't work" = down OR inconsistent
- In RDBMS we do not have P (network partitions)
  - Consistency and Availability are achieved
- In NoSQL we want to have P
  - Need to select either C or A
  - Drop A -> Accept waiting until data is consistent
  - Drop C -> Accept getting inconsistent data sometimes

NoSQL Systems and CAP



http://blog.nahurst.com/visual-guide-to-nosql-systems

ACID vs BASE

Scalability and better performance of NoSQL is achieved by sacrificing ACID compatibility.

Atomic, Consistent, Isolated, Durable

NoSQL is having BASE compatibility instead.

Basically Available, Soft state,
Eventual consistency

ACID -- Requirement for SQL DBs

- **Atomicity**. All of the operations in the transaction will complete, or none will.

- **Consistency**. Transactions never observe or result in inconsistent data.

- **Isolation**. The transaction will behave as if it is the only operation being performed upon the database (i.e. uncommitted transactions are isolated) **Durability**. Upon completion of the transaction, the

- operation will not be reversed (i.e. committed transactions are permanent)

BASE -- Basically Available

- Use replication and sharding to reduce the likelihood of data unavailability and use sharding, or partitioning the data among many different storage servers, to make any remaining failures partial.
- The result is a system that is always available, even if subsets of the data become unavailable for short periods of time.

BASE and Availability

- The availability of BASE is achieved through supporting partial failures without total system failure.
- Example. If users are partitioned across five database servers, BASE design encourages crafting operations in such a way that a user database failure impacts only the 20 percent of the users on that particular host.
  - This leads to higher perceived availability of the system. Even though a single node is failing, the interface is still operational.

BASE -- Eventually Consistent

- Although applications must deal with instantaneous consistency, NoSQL systems ensure that at some future point in time the data assumes a consistent state.
- In contrast to ACID systems that enforce consistency at transaction commit, NoSQL guarantees consistency only at some undefined future time.
  - Where ACID is pessimistic and forces consistency at the end of every operation, BASE is optimistic and accepts that the database consistency will be in a state of flux.

BASE and Consistency

- As DB nodes are added while scaling up, need for synchronization arises
- If absolute consistency is required, nodes need to communicate when read/write operations are performed on a node
  - Consistency over availability -> bottleneck
- As a trade-off, "eventual consistency" is used
- Consistency is maintained later
  - Numerous approaches for keeping up "distributed consistency" are available
    - Amazon Dynamo - consistent hashing
    - CouchDB - asynchronous master-master replication
    - MongoDB - auto-sharding+replication cluster with a master server

BASE -- Soft State

- While ACID systems assume that data consistency is a hard requirement, NoSQL systems allow data to be inconsistent and relegate designing around such inconsistencies to application developers.
- In other words, soft state indicates that the state of the system may change over time, even without input.
  - This is because of the eventual consistency model (the acronym is a bit contrived).

Some breeds of NoSQL solutions

- Key-Value Stores
- Column Family Stores
- Document Databases
- Graph Databases
- In addition: Object and RDF databases as well as Tuple stores

Key-Value Stores

- Dynamo, Voldemort, Rhino DHT ...
  - DeCandia et al. "Dynamo: Amazon's Highly Available Key-value Store", 2007
- Key-Value is based on a hash table where there is a unique key and a pointer to a particular item of data.
- Mappings are usually accompanied by cache mechanisms to maximize performance.
- API is typically simple -- implementation is often complex.

Column Family Stores

- BigTable, Cassandra, HBase, Hadoop ...
  - Chang et al. "Bigtable: A Distributed Storage System for Structured Data", 2006
- Store and process very large amounts of data distributed over many machines.
  - "Petabytes of data across thousands of servers"
- Keys point to multiple columns

jim_87	**Age**	**Name**	**Gender**	**Phone**
	25	Jim	M	123456
jill_90	**Age**	**Name**	**Gender**	**Phone**
	22	Jill	F	654321

- Cassandra example

Document Databases (Stores)

- CouchDB, MongoDB, Lotus Notes, Redis ...
- *Documents* are addressed in the database via a unique key that represents that document.
- Semi-structured documents can be XML or JSON formatted, for instance.
- In addition to the key, documents can be retrieved with queries.
- Redis is sometimes referred to as *data structure server* since keys can contain strings, hashes, lists, sets and sorted sets.

Graph Databases

- Neo4J, FlockDB, GraphBase, InfoGrip, ...
- Graph Databases are built with nodes, relationships between nodes (edges) and the properties of nodes.
  - Nodes represent entities (e.g. "Bob" or "Alice").
    - Similar in nature to the objects as in object-oriented programming.
  - Properties are pertinent information related to nodes (e. g. age: 18).
  - Edges connect nodes to nodes or nodes to properties.
    - Represent the relationship between the two.
- Scaling graph DBs is problematic
  - Neo4J: cache sharding, sharding strategy heuristics

Some NoSQL Challenges

- Lack of maturity -- numerous solutions still in their beta stages
- Lack of commercial support for enterprise users
- Lack of support for data analysis
- Maintenance efforts and skills are required -- experts are hard to find

Quiz

Q. Select the NoSQL Database

A. Oracle 12c

B. Neo4J

C. Hive

D. Derby

Quiz

Q. All NoSQL databases are inherently consistent

A. True

B. False

Quiz

Q. Which of the following is not a reason NoSQL has become a popular solution for some organizations?

A. Better scalability

B. Improved ability to keep data consistent

C. Faster access to data than relational database management systems (RDBMS)

D. More easily allows for data to be held across multiple servers

Quiz

Q. While NoSQL databases avoid the rigid schemas of relational databases, the types of NoSQL technologies vary and can be separated into the following primary categories:

A. Document databases, graph databases, key-value databases and wide column stores

B. CouchDB, MongoDB, Cassandra and HBase

C. Those that manage data in the cloud and those that don't

D. Oracle NoSQL database, NoSQL for Windows Azure and IBM DB2

HBase

The Hadoop database

Agenda

- Overview
- Architecture
- Installation
- Shell
  - Basic Commands

Overview

Overview

- Column-Oriented data store, known as “Hadoop Database”
- Supports random real-time CRUD operations (unlike HDFS)
- Distributed – designed to serve large tables
  - Billions of rows and millions of columns
- Runs on a cluster of commodity hardware
  - Server hardware, not laptop/desktops
- Open-source, written in Java
- Type of “NoSQL” DB
  - Does not provide a SQL based access
- – Does not adhere to Relational Model for storage

HBase

- Horizontally scalable
  - Automatic sharding
- Strongly consistent reads and writes
- Automatic fail-over
- Simple Java API
- Integration with Map/Reduce framework
- Thrift, Avro and REST-ful Web-services
- Based on Google's Bigtable
  - http://labs.google.com/papers/bigtable.html
- Just like BigTable is built on top of Google File System (GFS), HBase is implemented on top of HDFS

HBase History

Year	Event
Nov 2006	Google released the paper on BigTable.
Feb 2007	Initial HBase prototype was created as a Hadoop contribution.
Oct 2007	The first usable HBase along with Hadoop 0.15.0 was released.
Jan 2008	HBase became the sub project of Hadoop.
Oct 2008	HBase 0.18.1 was released.
Jan 2009	HBase 0.19.0 was released.
Sept 2009	HBase 0.20.0 was released.
May 2010	HBase became Apache top-level project.

Who Uses HBase?

- **Here is a very limited list of well known names**
  - Facebook
  - Adobe
  - Twitter
  - Yahoo!
  - Netflix
  - Meetup
  - Stumbleupon
  - You????

When To Use HBase

- **Not suitable for every problem**
  - Compared to RDBMs has VERY simple and limited API
- **Good for large amounts of data**
  - 100s of millions or billions of rows
  - If data is too small all the records will end up on a single node leaving the rest of the cluster idle

When To Use HBase

- **Have to have enough hardware!!**
  - At the minimum 5 nodes
    - There are multiple management daemon processes: Namenode, HBaseMaster, Zookeeper, etc....
    - HDFS won't do well on anything under 5 nodes anyway;particularly with a block replication of 3
    - HBase is memory and CPU intensive
- **Carefully evaluate HBase for mixed work loads**
  - Client Request vs. Batch processing (Map/Reduce)
    - SLAs on client requests would need evaluation
  - HBase has intermittent but large IO access
    - May affect response latency!!!

When To Use HBase

- **Two well-known use cases**
  - Lots and lots of data (already mentioned)
  - Large amount of clients/requests (usually cause a lot of data)
- **Great for single random selects and range scans by key**
- **Great for variable schema**
  - Rows may drastically differ
  - If your schema has many columns and most of them are null

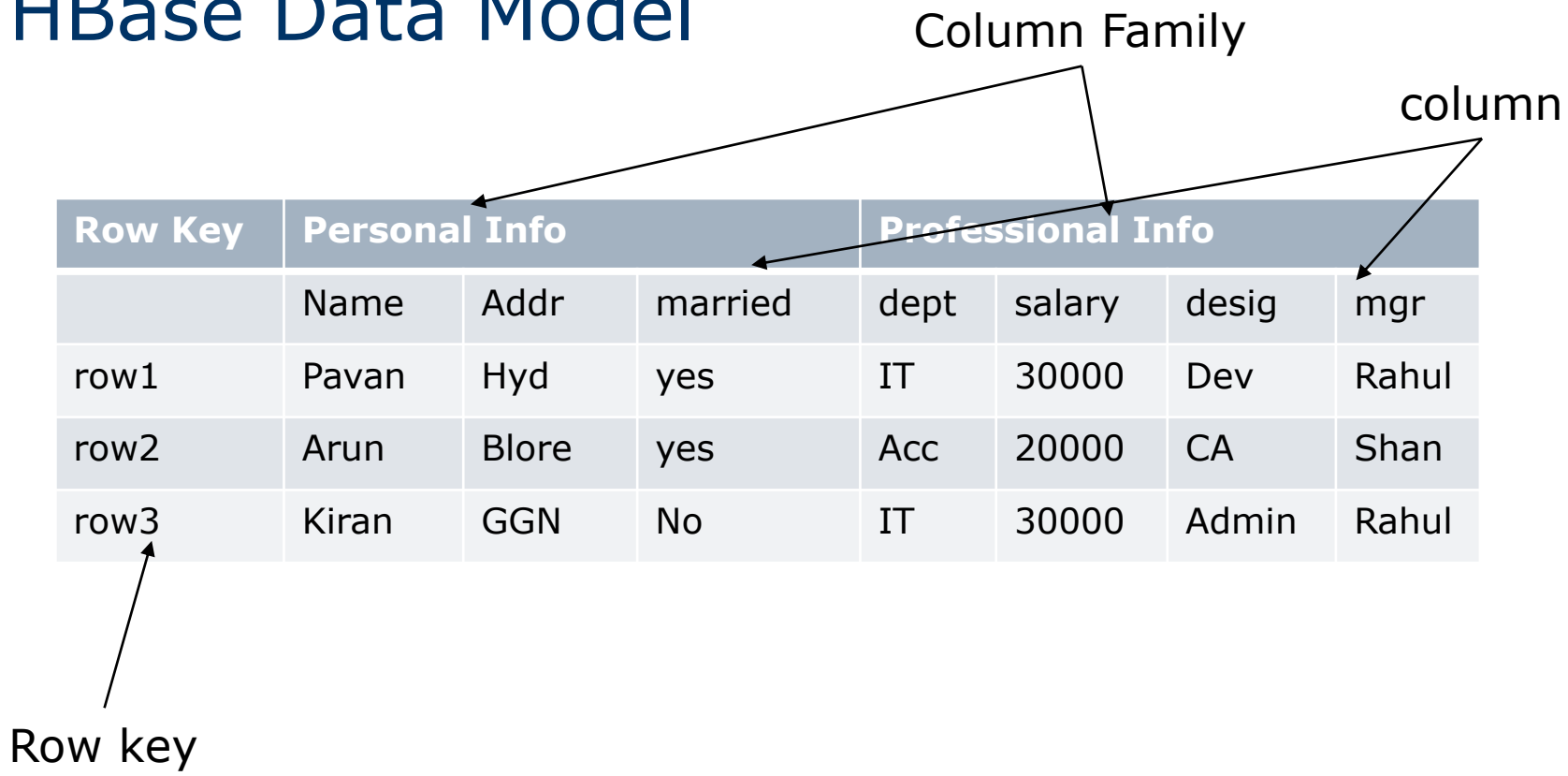
When NOT to Use HBase

- **Bad for traditional RDBMs retrieval**
  - Transactional applications
  - Relational Analytics
    - 'group by', 'join', and 'where column like', etc....
- **Currently bad for text-based search access**
  - There is work being done in this arena
    - HBasene: https://github.com/akkumar/hbasene/wiki
    - HBASE-3529: 100% integration of HBase and Lucene based on HBase' coprocessors
  - Some projects provide solution that use HBase
    - Lily=HBase+Solr http://www.lilyproject.org

HBase Data Model

- In HBase:
  - Table is a collection of rows.
  - Row is a collection of column families.
  - Column family is a collection of columns.
  - Column is a collection of key value pairs.

HBase Data Model



Row Key	Personal Info			Professional Info			
	Name	Addr	married	dept	salary	desig	mgr
row1	Pavan	Hyd	yes	IT	30000	Dev	Rahul
row2	Arun	Blore	yes	Acc	20000	CA	Shan
row3	Kiran	GGN	No	IT	30000	Admin	Rahul

HBase Data Model

- **Data is stored in Tables**
- **Tables contain rows**
  - Rows are referenced by a unique key
  - Key is an array of bytes – good news
  - Anything can be a key: string, long and your own serialized data structures
- **Rows made of columns which are grouped in column families**
- **Data is stored in cells**
  - Identified by **row x column-family x column**
  - Cell's content is also an array of bytes

HBase Timestamps

- **Cells' values are versioned**
  - For each cell multiple versions are kept
    - 3 by default
  - Another dimension to identify your data
  - Either explicitly timestamped by region server or provided by the client
    - Versions are stored in decreasing timestamp order
    - Read the latest first – optimization to read the current value
- **You can specify how many versions are kept**
  - More on this later....

HBase Row Keys

- **Rows are sorted lexicographically by key**
  - Compared on a binary level from left to right
  - For example keys 1,2,3,10,15 will get sorted as
    - 1, 10, 15, 2, 3
- **Somewhat similar to Relational DB primary index**
  - Always unique
  - Some but minimal secondary indexes support

Quiz

Q. Columns in Hbase are organized to

A - Column Group

B - Column families

C - Column list

D - Column base

Quiz

Q.To locate a piece of data Hbase uses coordinates. A coordinate is made-up of

A - rowId, table name, block address

B - rowid, table name, column name

C - rowkey, table name, column name

D - rowkey, table name, block address

HBase Architecture

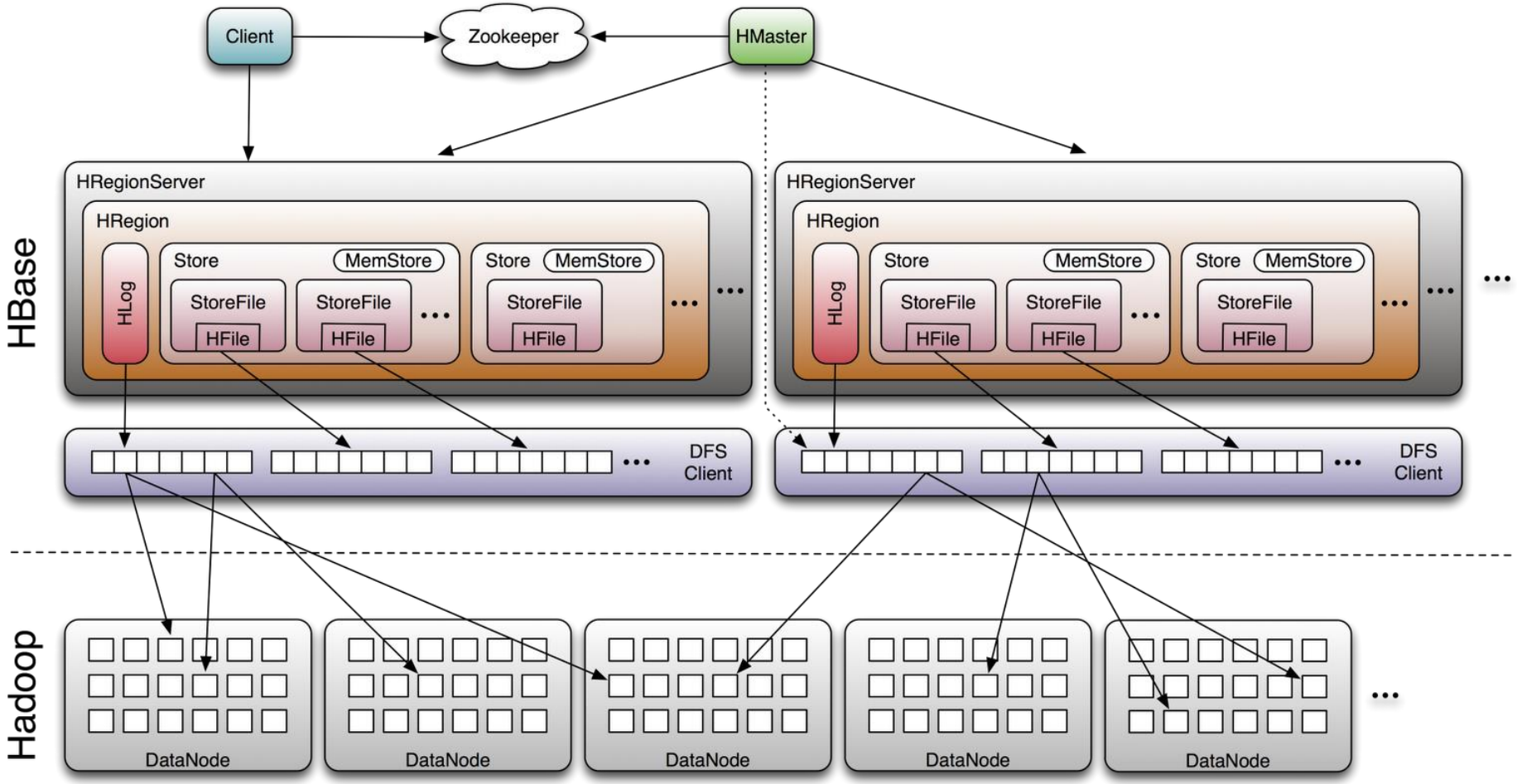
HBase Architecture

- Table is made of regions
- Region – a range of rows stored together
  - Single shard, used for scaling
  - Dynamically split as they become too big and merged if too small
- Region Server- serves one or more regions
  - A region is served by only 1 Region Server
  - Master Server – daemon responsible for managing HBase cluster, aka Region Servers
- HBase stores its data into HDFS
  - relies on HDFS's high availability and fault-tolerance features
- HBase utilizes Zookeeper for distributed coordination

Hbase Architecture

- HBase has three major components:
  - The client library,
  - A master server
  - Region servers
    - Region servers can be added or removed as per requirement.

HBase Architecture



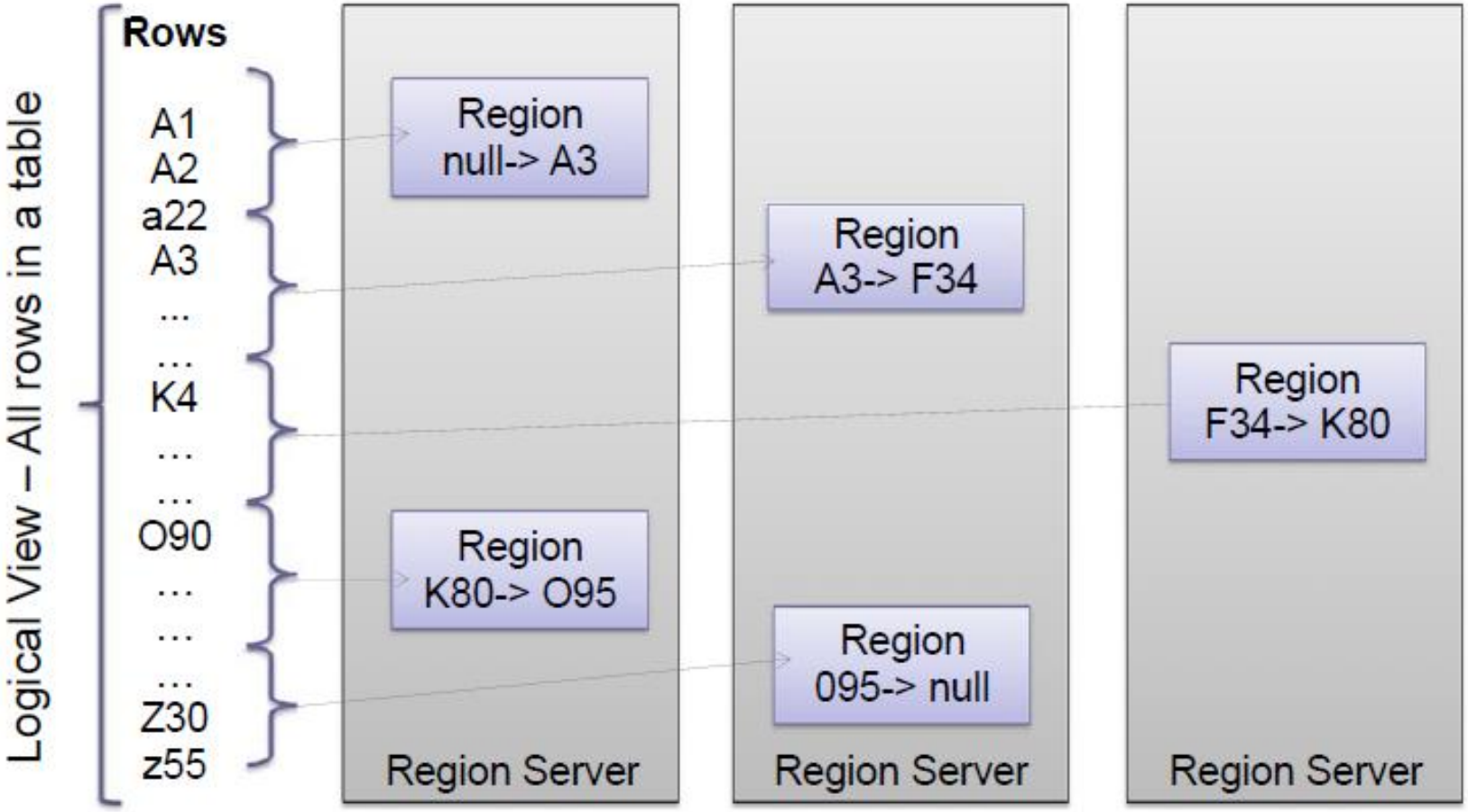
MasterServer

- Assigns regions to the region servers and takes the help of Apache ZooKeeper for this task.
- Handles load balancing of the regions across region servers.
- It unloads the busy servers and shifts the regions to less occupied servers.
- Maintains the state of the cluster by negotiating the load balancing.
- Is responsible for schema changes and other metadata operations such as creation of tables and column families.

Region Server

- **Regions**
  - Regions are nothing but tables that are split up and spread across the region servers.
- **Region server**
  - The region servers have regions that -
    - Communicate with the client and handle data-related operations.
    - Handle read and write requests for all the regions under it.
    - Decide the size of the region by following the region size thresholds.

Rows Distribution Between Region Servers



Zookeeper

- Zookeeper is an open-source project that provides services like maintaining configuration information, naming, providing distributed synchronization, etc.
- Zookeeper has ephemeral nodes representing different region servers.
- Master servers use these nodes to discover available servers.
- In addition to availability, the nodes are also used to track server failures or network partitions.
- Clients communicate with region servers via zookeeper.
- In pseudo and standalone modes, HBase itself will take care of zookeeper.

Quiz

Q . A small chunk of data residing in one machine which is part of a cluster of machines holding one Hbase table is known as

A - Split

B - Region

C - Rowarea

D - Tablearea

Quiz

Q . Servers that host regions of a Hbase table are called

A - RegionServers

B - Regional servers

C - Hbase Servers

D - Splitservers

Quiz

Q. Typically a Hbase Regionserver is collocated with

A - HDFS Namenode

B - HDFS datanode

C - As a client to HDFS server

D - TAsktrackers

HBase Shell

HBase Shell

- To interact with HBase , it provides a command line interface which we call HBase Shell
- It allows you to execute a number of HBase Commands.

HBase Shell – General Commands

- **status** - Provides the status of HBase, for example, the number of servers.
- **version** - Provides the version of HBase being used.
- **table_help** - Provides help for table-reference commands.
- **whoami** - Provides information about the user.

HBase Shell- DDL

- create - Creates a table.
- list - Lists all the tables in HBase.
- disable - Disables a table.
- is_disabled - Verifies whether a table is disabled.
- enable - Enables a table.
- is_enabled - Verifies whether a table is enabled.
- describe - Provides the description of a table.
- alter - Alters a table.
- exists - Verifies whether a table exists.
- drop - Drops a table from HBase.
- drop_all - Drops the tables matching the ‘regex’ given in the command.
- Java Admin API - Prior to all the above commands, Java provides an Admin API to achieve DDL functionalities through programming

HBase Shell- **Data Manipulation Language**

- put - Puts a cell value at a specified column in a specified row in a particular table.
- get - Fetches the contents of row or a cell.
- delete - Deletes a cell value in a table.
- deleteall - Deletes all the cells in a given row.
- scan - Scans and returns the table data.
- count - Counts and returns the number of rows in a table.
- truncate - Disables, drops, and recreates a specified table.
- Java client API - Prior to all the above commands, Java provides
- a client API to achieve DML functionalities

HBase Shell—How to start?

- Navigate to HBase home folder

  e.g. cd /usr/hbase

- Issue the command as

  ./bin/hbase shell

- On success it will show you message and a prompt one like :

```
Base Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.94.23, rf42302b28aceaab773b15f234aa8718fff7eea3c

hbase(main):001:0>
```

Creating a Table using HBase Shell

- Syntax

create '<table name>','<column family>‘

- Example:

hbase(main):002:0> create 'emp', 'personal', ’professional’

- Verify:

hbase(main):002:0> list

TABLE

emp

2 row(s) in 0.0300 seconds

Other DDL Commands

- **Check the existence of a table**

```
hbase(main):021:0> exists 'emp'
Table emp does exist
0 row(s) in 0.0650 seconds
```

- **Drop a Table**
  - Before dropping a table it has to be disabled

```
hbase(main):010:0> disable 'emp'
0 row(s) in 1.40 seconds

hbase(main):011:0> drop 'emp'
0 row(s) in 0.30 seconds
```

Other Commands
drop_all

Shutdown HBase

- Exit from the shell and then shutdown HBase

```
hbase(main):021:0> exit
```

```
./bin/stop-hbase.sh
```

Create Data in HBase Table

- Command –put
- Syntax

put '<tablename>','row1','<colfamily:colname>','<value>`

- Example:

```
hbase(main):005:0> put 'emp','row1','personal:name','Shantanu'
0 row(s) in 0.600 seconds
hbase(main):006:0> put 'emp','row1','personal:city','hyderabad'
0 row(s) in 0.040 seconds
hbase(main):007:0> put
'emp','row1','professional:designation','manager'
0 row(s) in 0.020 seconds
hbase(main):008:0> put 'emp','row1','professional:salary','80000'
0 row(s) in 0.0200 seconds
```

Display the inserted data

- scan <table name>

Example : ‘scan emp’

Updating Data using HBase Shell

- You can update an existing cell value using the put command. To do so, just follow the same syntax and mention your new value as shown below.

put `<table name>','row','Column family:column name','new value'

- Example:

```
hbase(main):006:0> put 'emp','row1','personal:city',’Bangalore'
0 row(s) in 0.040 seconds
```

Read Data

- Command: get
- Syntax: get <table name> <row>
- Example

```
hbase(main):012:0> get 'emp', '1`

   COLUMN                    CELL

personal : city timestamp = 1417521848375, value = Bangalore
personal : name timestamp = 1417521785385, value = Shantanu
professional: designation timestamp = 1417521885277, value =
manager
professional: salary timestamp = 1417521903862, value = 80000
4 row(s) in 0.0270 seconds
```

Reading a Specific Column

- Command: get
- Syntax:

hbase> get 'table name', ‘rowid’, {COLUMN ⇒ ‘column family:column name ’}

```
hbase(main):030:0> get 'emp', 'row1', {COLUMN ⇒ 'personal:name'}
  COLUMN                CELL
personal:name timestamp = 1418035791555, value = Shantanu
1 row(s) in 0.0080 seconds
```

Delete data

- Delete a cell

delete ‘<table name>’, ‘<row>’, ‘<column name >’, ‘<time stamp>’

- Delete the entire row

deleteall ‘<table name>’, ‘<row>’

Summary

- In this class we have Understood
  - what is HBase
  - How to install HBase
  - HBase architecture
  - HBase Shell
  - HBase Shell commands
  - HBase CRUD commands

Q&A