






# Data Engineering and Big Data Course

# Agenda

-  Hadoop Introduction
-  HDFS
-  Map Reduce
-  YARN
-  Hands On- HDFS and MapReduce

# HDFS - Hadoop Distributed File System

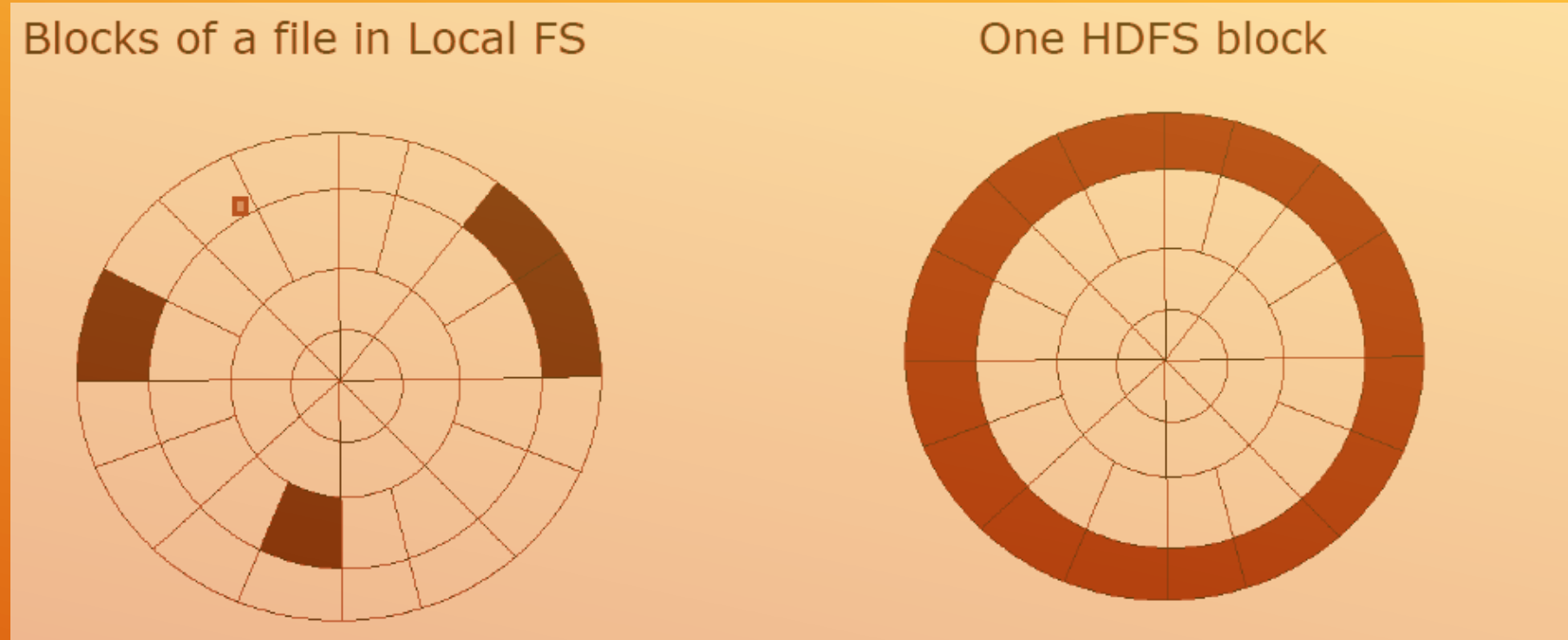
- The file store in HDFS provides scalable, fault tolerant storage at low cost.
- The HDFS software detects and compensates for hardware issues, including disk problems and server failure.
- HDFS stores file across the collection of servers in a cluster.
- Files are decomposed into the blocks and each block is written to more than one of the servers.
- The replication provides both fault tolerance and performance.
- HDFS is a filesystem written in Java
- Sits on top of a native filesystem such as “ext3”, “ext4” or “xfs”
- Provides redundant storage for massive amounts of data
- Using “readily/available,” “industry/standard” compute

# Design of HDFS

HDFS has been designed keeping in view the following features:

- **Very large files:** Files that are megabytes, gigabytes, terabytes or petabytes of size.
- **Data access:** HDFS is built around the idea that data is written once but read many times. A dataset is copied from source and then analysis is done on that dataset over time.
- **Commodity hardware:** Hadoop does not require expensive, highly reliable hardware as it is designed to run on clusters of commodity hardware.
- **Growth of storage vs read/write performance-** One hard drive in 1990 could store 1,370 MB of data and
- had a transfer speed of 4.4 MB/s so full data can be read in five minutes. Now with 1 terabyte drive transfer speed is around 100 MB/s But it takes more than two and a half hours to read all the data off the disk.
- Although the storage capacities of hard drives have increased, yet access speeds have not kept up with the same cost spending.

# HDFS Blocks

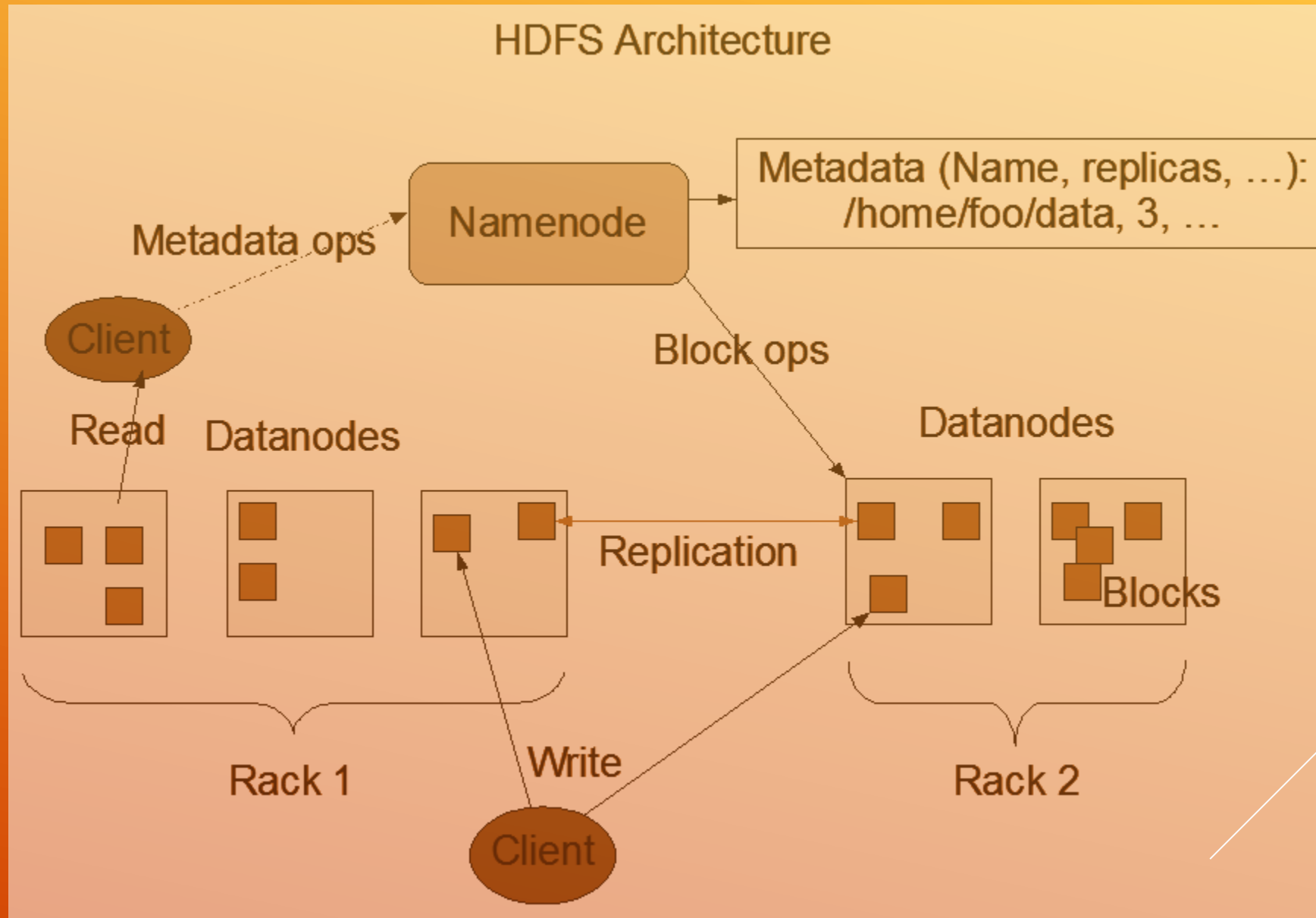


Hard Disk has concentric circles which form tracks.

- One file can contain many blocks. These blocks in local file system are nearly 512 bytes and not necessarily continuous.
- For HDFS, since it is designed for large files, block size is 128 MB by default. Moreover, it gets blocks of local file system contiguously to minimise head seek time



# HDFS Architecture



# HDFS Commands `hdfs dfs -help`



# Components of Hadoop 1.x

## NameNode

- Contains Hadoop FileSystem
- Tree and other metadata information about files and directories.
- Contains in memory mapping of which blocks are stored in which datanode

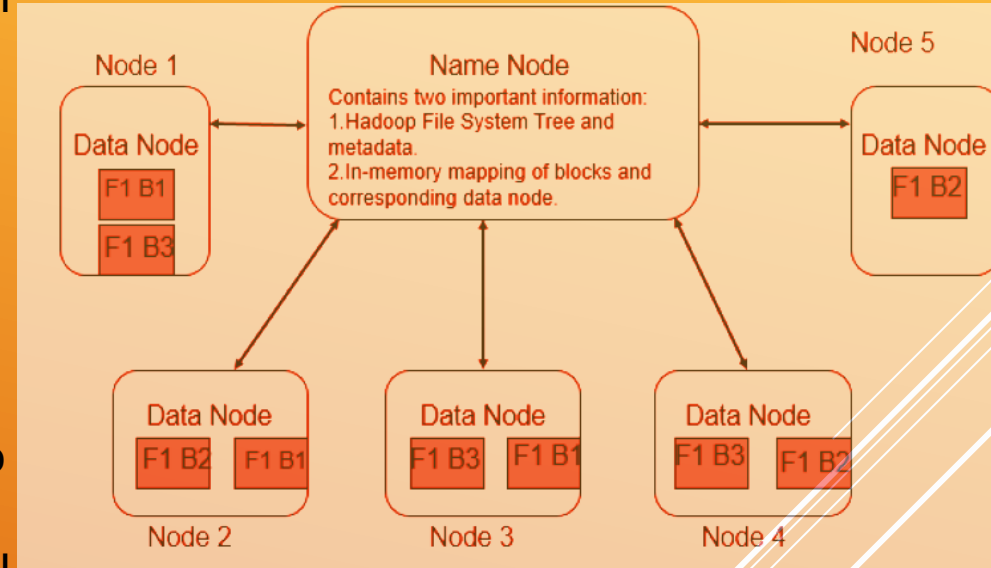
## Secondary Namenode

- Performs house-keeping activities for namenode, like periodic merging of namespace and edits.
- This is not a back up for namenode

## DataNode

- Stores actual data blocks of file in HDFS on its own local disk.
- Sends signals to NameNode periodically (called as Heartbeat) to verify it is active.
- Sends block reporting to the namenode on cluster startup as well as periodically at every 10th Heartbeat.
- The data node are the workhorse of the system.
- They perform all the block operation including periodic checksum. They receive instructions from the name node of where to put the blocks and how to put the blocks.

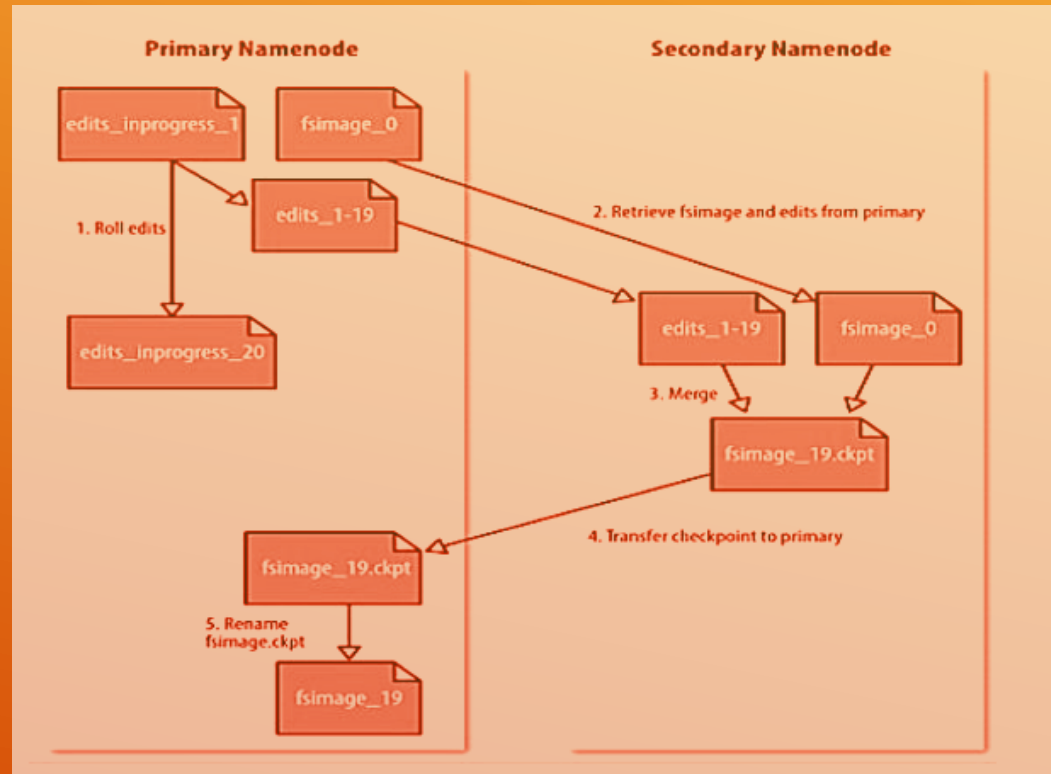
**Edge Node (Not mandatory)**– Actual client libraries to run the code/big data application, it is kept separate to minimize load on name node and data nodes.





# Daemons of Hadoop 1.x

NameNode contains two important files on its hard disk:



## 1.fsimage(file system image)

It contains:

- all directory structure of HDFS
- replication level of file
- modification and access times of files
- access permissions of files and directories
- block size of files
- the blocks constituting a file
- A Transaction Log-Records file creations, file deletions etc.

## 2.Edits

- When any write operation takes place in HDFS, the directory structure gets modified.
- These modifications are stored in memory as well as in edits files (edits files are stored on hard disk).
- If existing fsimage file gets merged with edits, we'll get updated fsimage file.
- This process is called checkpointing and is carried out by Secondary Namenode

# HDFS

## Safe Mode:

- During start up, the NameNode loads the file system state from the fsimage and the edits log file.
- It then waits for DataNodes to report their blocks. During this time, NameNode stays in Safemode
- Safemode for the NameNode is essentially a read-only mode for the HDFS cluster, where it does not allow any modifications to file system or blocks

## Replica Placement

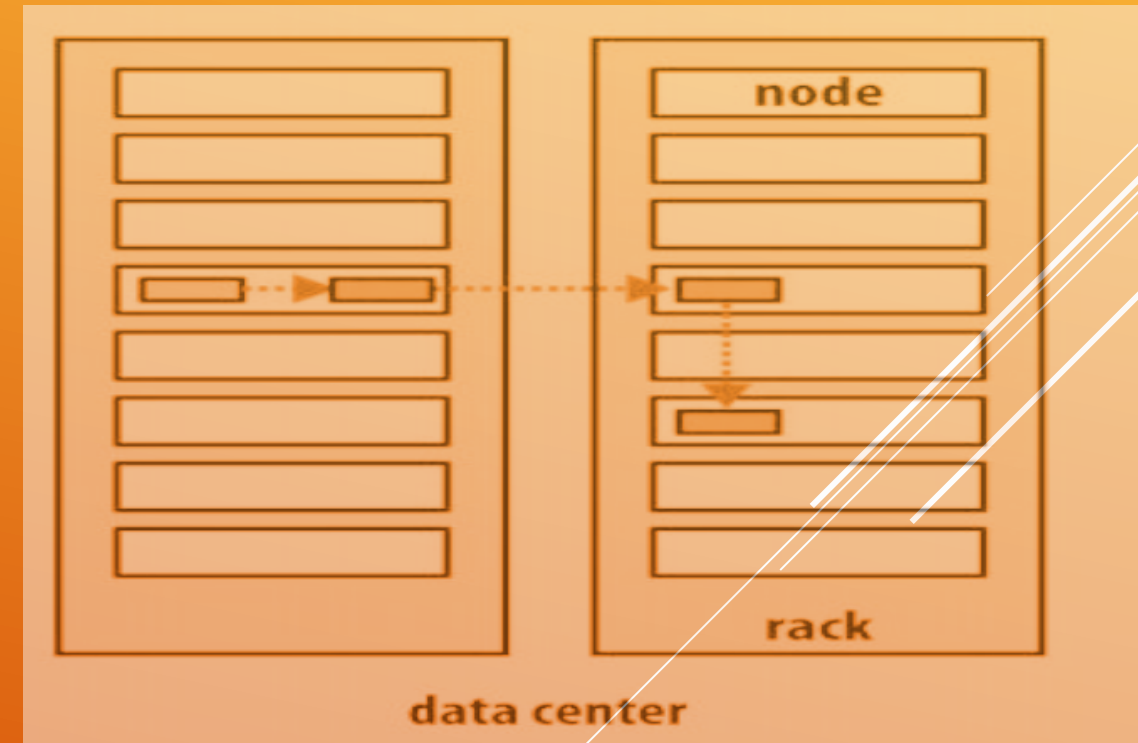
How does the namenode choose which data nodes to store replicas on?

- Placing all replicas on a single node incurs the lowest write bandwidth penalty (since the replication pipeline runs on a single node)
- But this offers no real redundancy (if the node fails, the data for that block is lost).
- Also, the read bandwidth is high for off-rack reads.
- At the other extreme, placing replicas in different data centres may maximize redundancy, but at the cost of write bandwidth.
- Hadoop's default strategy is to place the first replica on the same node as the client
- For clients running outside the cluster, a node is chosen at random.

Cluster = Name Node + Secondary Name Node+ Data Nodes

Cluster = Name Node + Secondary Name Node+ Data Nodes+ Edge Node

- The system tries not to pick nodes that are too full or too busy.
- The second replica is placed on a different rack from the first (off-rack), chosen at random.
- The third replica is placed on the same rack as the second, but on a different node chosen at random.
- Further replicas are placed on random nodes in the cluster, although the system tries to avoid placing too many replicas on the same rack.



# Benefits of Replica Placement and Rack Awareness

This strategy gives a good balance among:

- reliability (blocks are stored on two racks, so data is available even in case of node or rack failure)
- write bandwidth (writes only have to traverse a single network switch)
- read performance (there's a choice of two racks to read from)
- block distribution across the cluster (clients only write a single block on the local rack)

## **Balancer:**

A tool that analyzes block placement and re-balances data across the DataNodes.

**Goal:** disk full on DataNodes should be similar

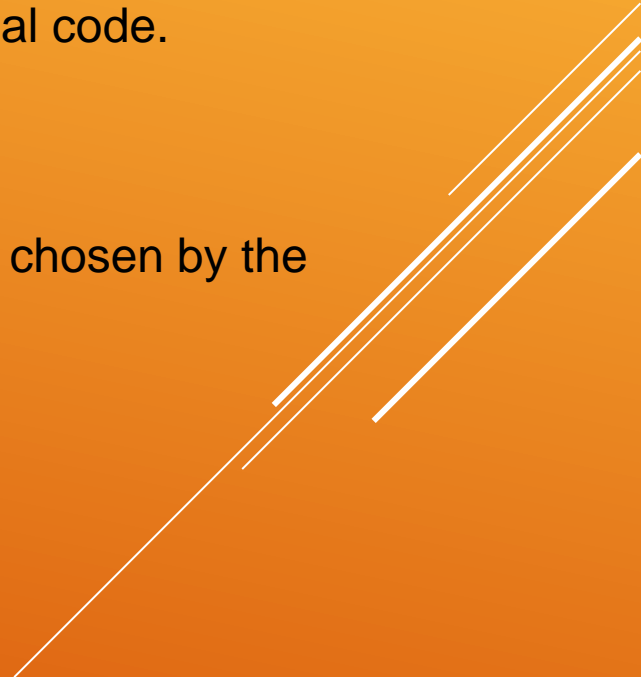
- Usually run when new DataNodes are added
- Cluster is online when rebalancer is active
- Rebalancer is throttled to avoid network congestion
- Command line tool

# Map Reduce and YARN

## Building Principles

- The individual concepts of functions called map and reduce have been derived from functional programming languages (like C & Java) where they were applied to lists of input data.
- Another key underlying concept is that of “divide and conquer”, where a single problem is broken into multiple individual sub tasks. This approach becomes even more powerful when the sub tasks are executed in parallel.
- The developer focuses on expressing the transformation between source and result data sets, and the Hadoop framework manages all aspects of job execution, parallelization, and coordination.
- MapReduce is a programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks.

# Some More Real-World Examples

- An address book relates a name (key) and to contact information (value).
  - A bank account uses an account number (key) to associate with the account details (value).
  - The index of a book relates a word (key) to the pages on which it occurs (value).
  - On a computer file system, filenames (keys) allow access to any sort of data, such as text, images, and sound (values).
- 
- Map Reduce sends code to distributed data, instead of bringing data to the actual code.
  - Map Reduce works by breaking the processing into two phases:
    - the map phase
    - the reduce phase
  - Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer.
  - The programmer also specifies two functions:
    - the map function
    - the reduce function
  - There is a shuffle and sort phase in between.
- 
- A series of three parallel white diagonal lines are positioned in the bottom right corner of the slide, extending from the middle of the right edge towards the bottom left.

# Broad Steps

Map phase takes input in Key-Value pairs

It produces output in the form of Key-Value pair.

Output from various Map tasks are grouped together on the basis of Key.



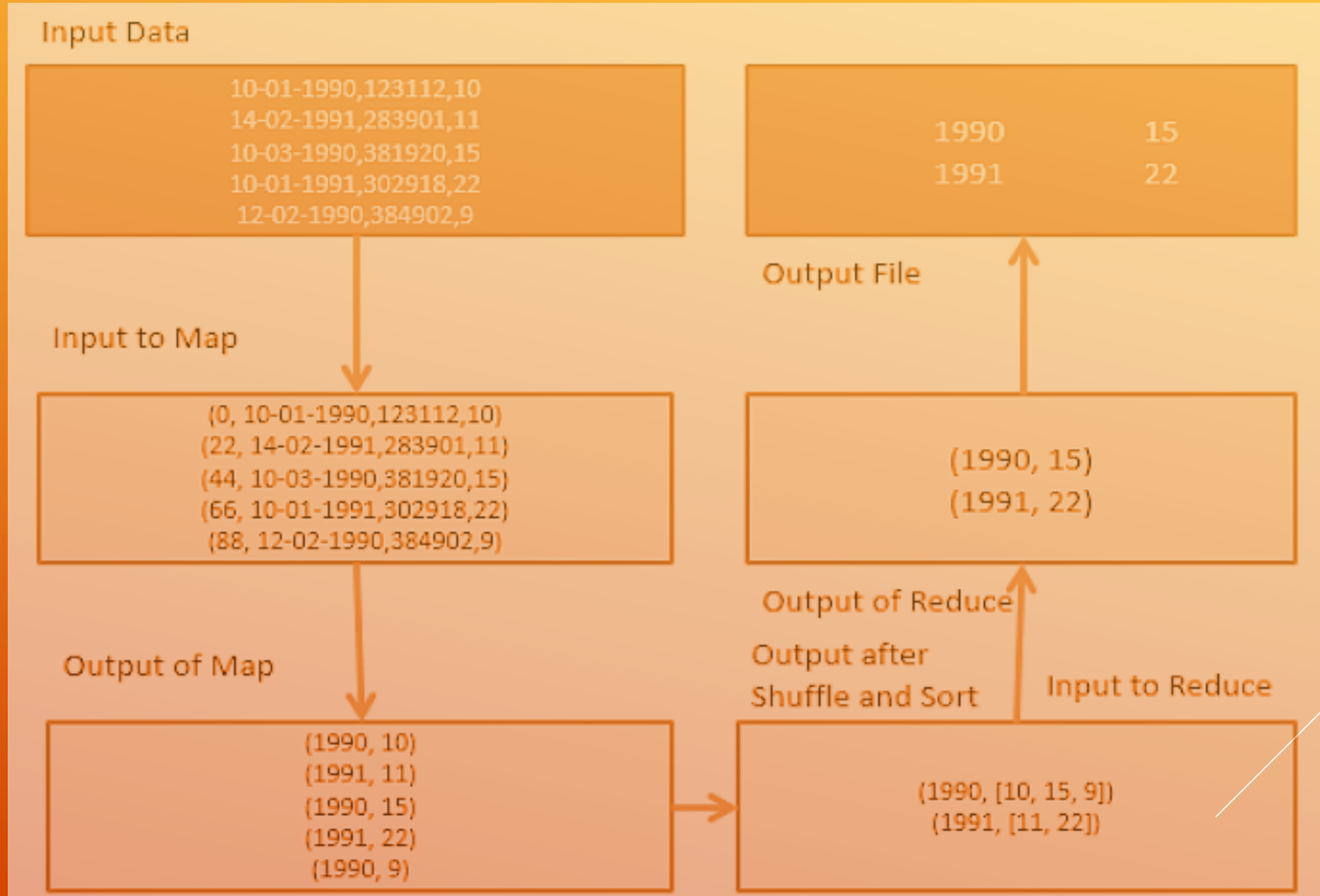
Key and its associated set of values are sent to the Reduce phase.

Reduce method operates on key and associated list of values.

Output of Reduce is written to HDFS.



# Example: Finding Out Maximum Temperature



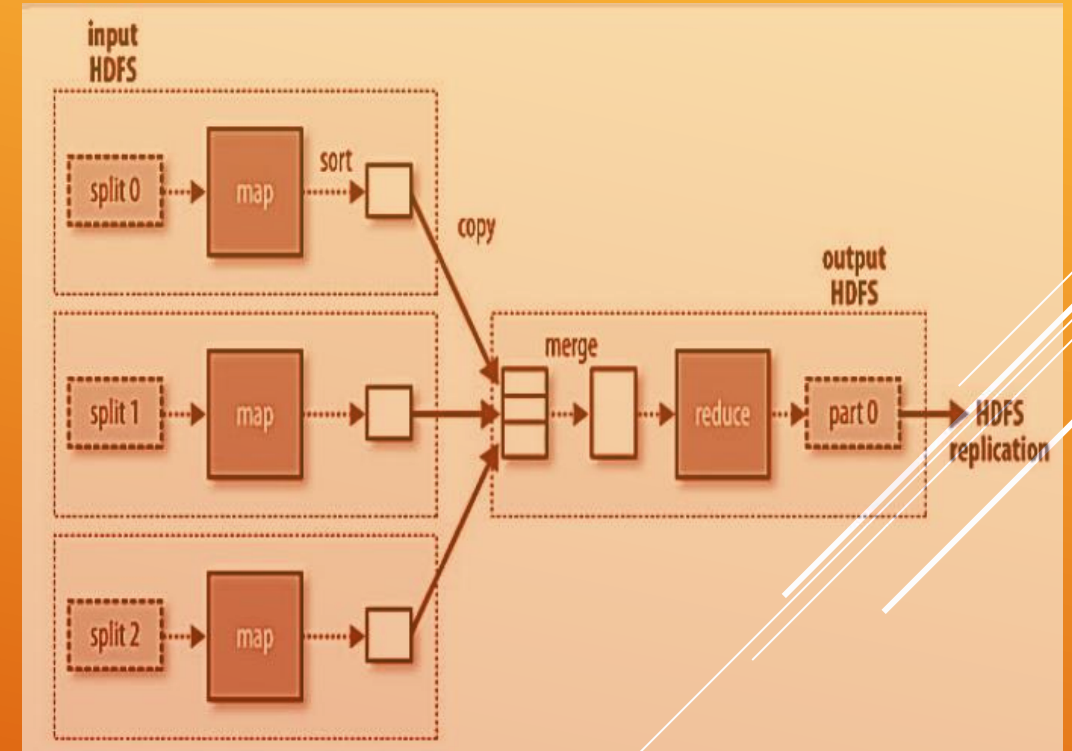
# Pseudo Code

## Map Phase

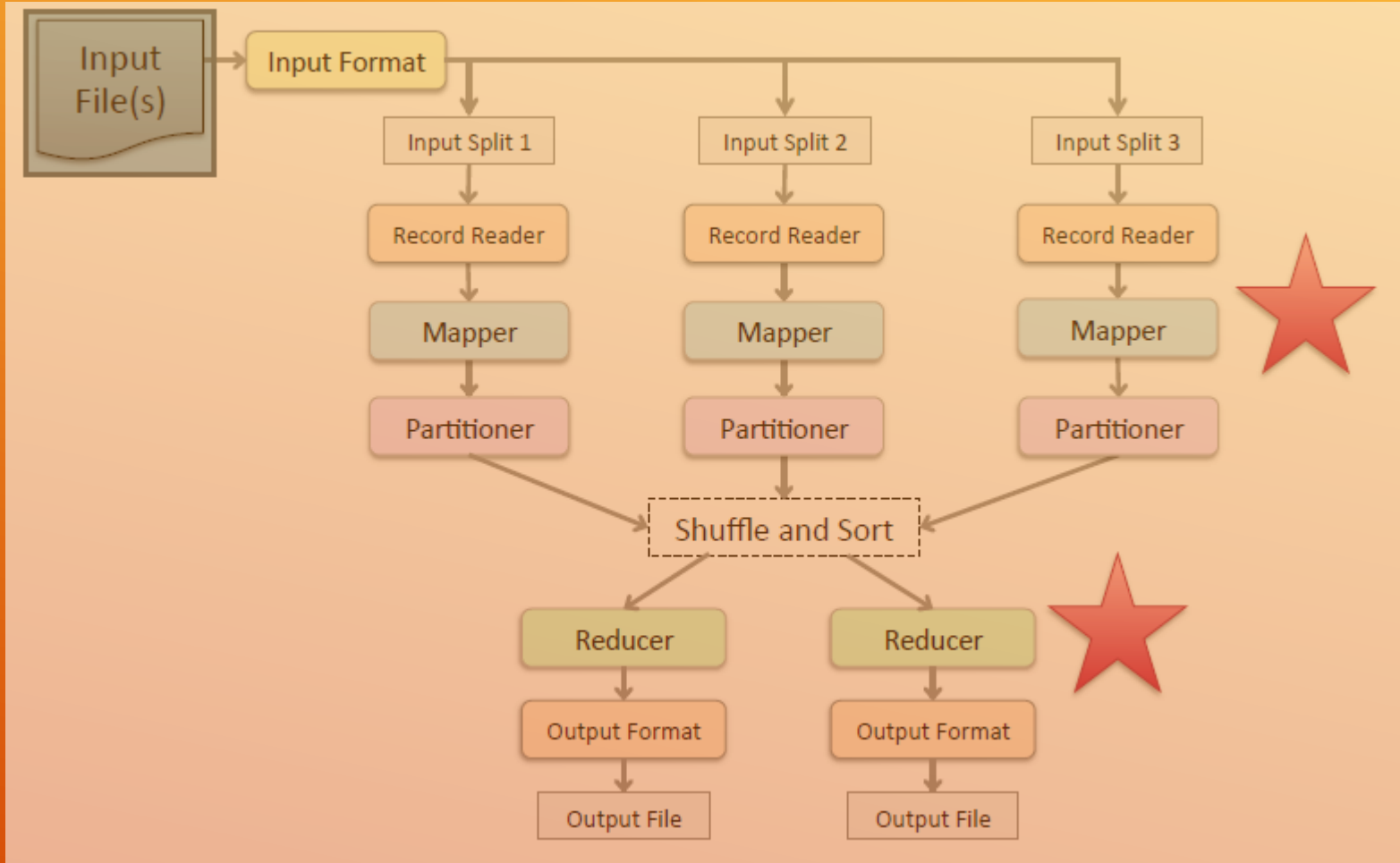
- Read K, V (By default, K is byte offset of the line, V is the whole line)
- Extract year and temperature from V
- Emit (year, temperature)

## Reduce Phase

- Read K, list (Here K is the year, as produced from output of map method)
- Iterate through list to get maximum element
- Emit (K, max)

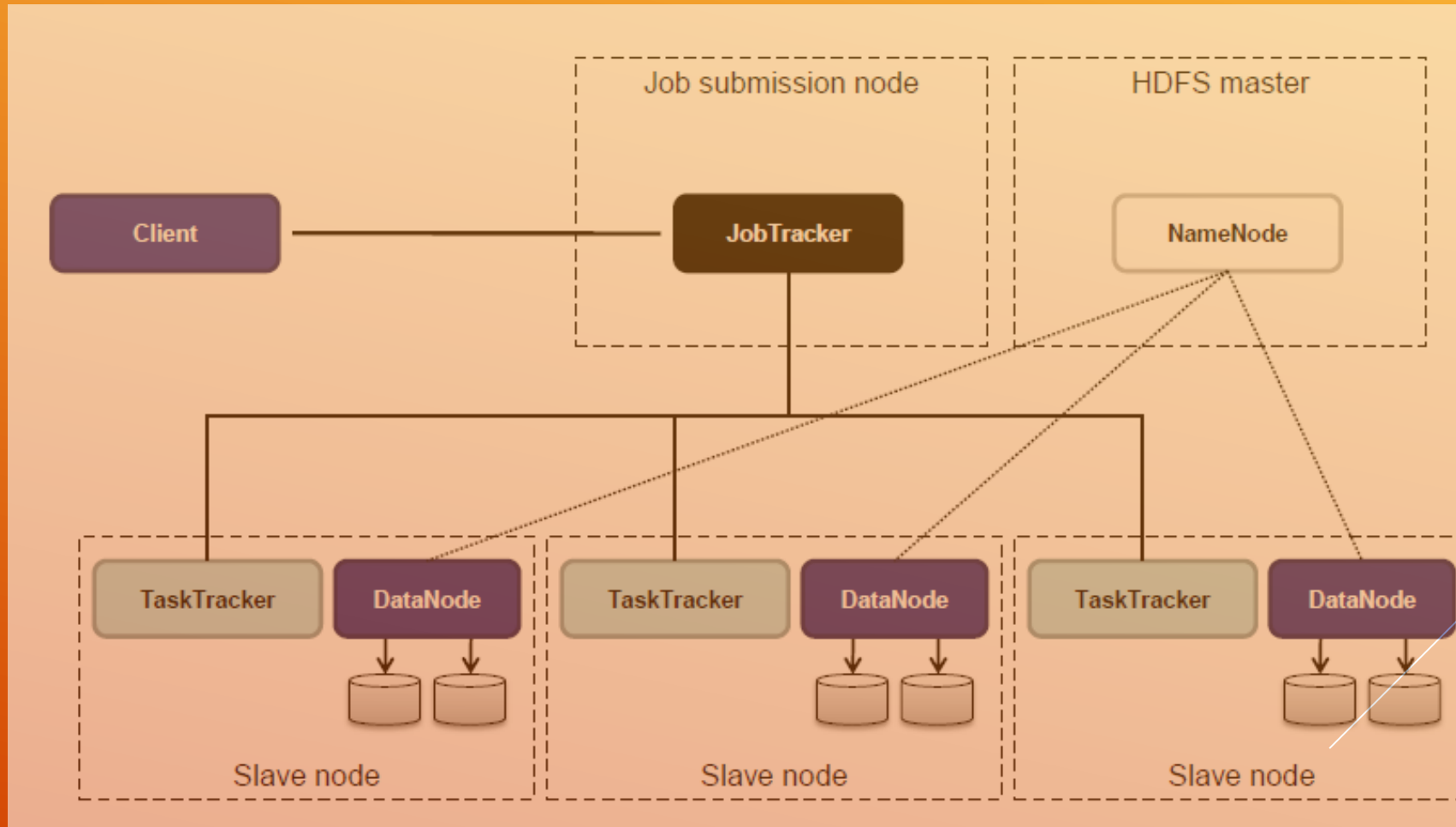


# MapReduce Architecture & Code



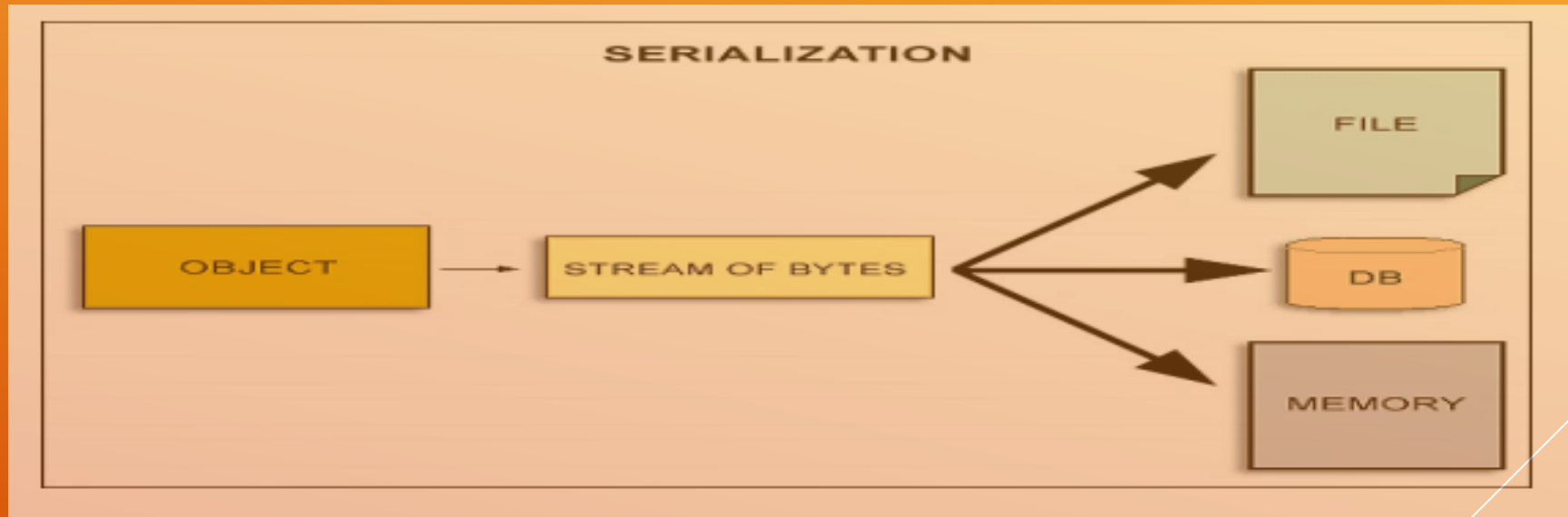
# Daemons of Hadoop 1.x (MapReduce)

- JobTracker(Not present in Hadoop 2.x) – Controls overall execution of map reduce jobs.
- TaskTracker(Not present in Hadoop 2.x)-Runs individual map-reduce jobs on datanodes




# Serialization

Serialization is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer, or transmitted across a network connection link).



# Serialization Classes in Hadoop

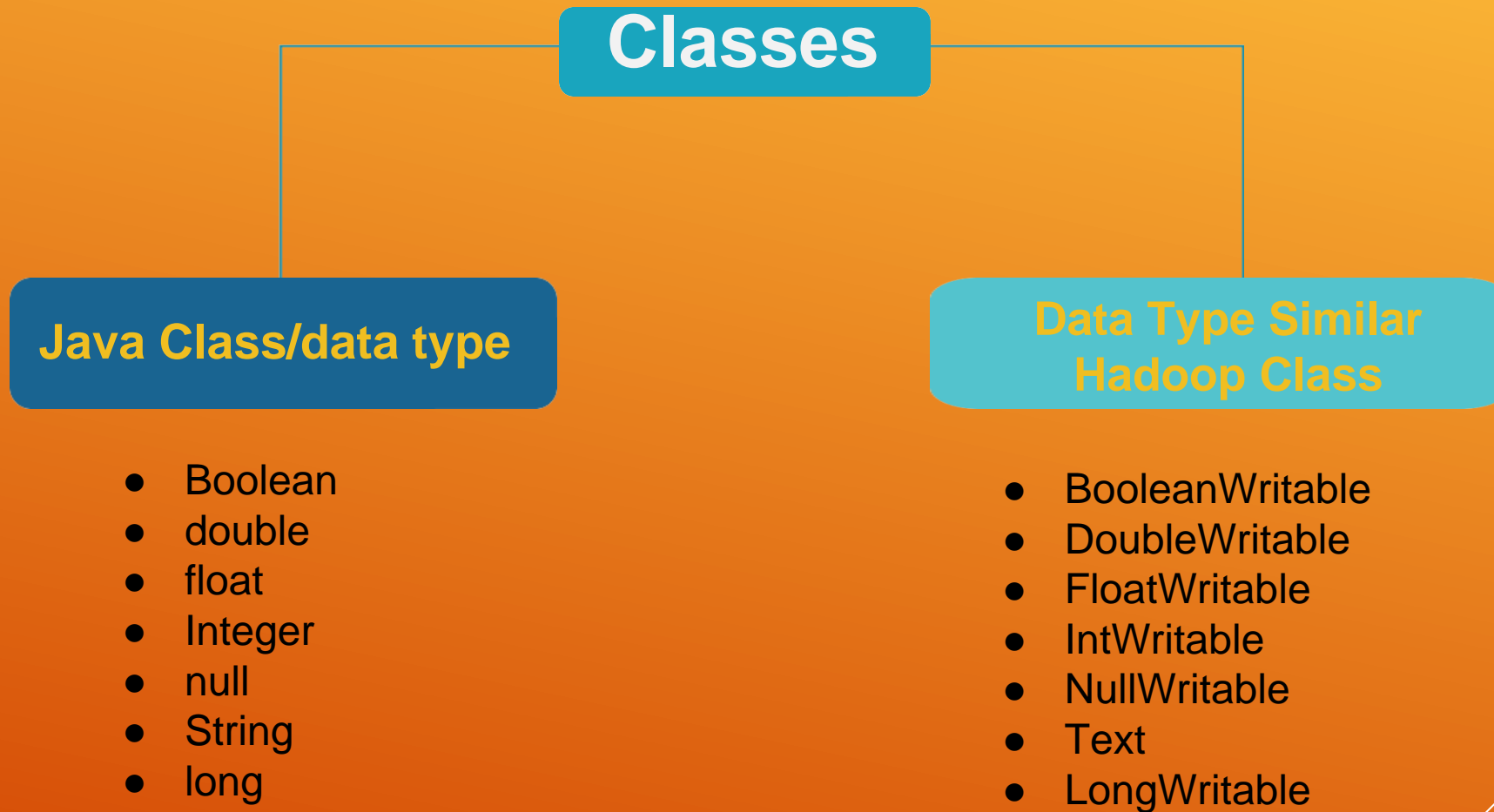
- Hadoop uses its own set of serializable objects which implements a simple, efficient, serialization protocol, based on DataInput and DataOutput.
  - Any key or value type in the Hadoop Map-Reduce framework uses these objects.
  - Hadoop serialization objects are mutable (unlike Java String objects) and are well suited to be sent across network
- 
- A series of three parallel white diagonal lines are located in the bottom right corner of the slide, extending from the middle of the right edge towards the bottom left.




# Serialization in MapReduce

- Data needs to be transmitted between different nodes in a distributed computing environment.
- This requires serialization and deserialization of data to convert the data that is in structured format to byte stream and vice-versa.
- Hadoop uses serializable objects for key-values which implements a simple, efficient, serialization protocol, based on DataInput and DataOutput.

# Serialization Classes in Hadoop (Contd.)



# YARN

- In Hadoop 1.x, the JobTracker has three major functions:
    1. Resource Management
    2. Job Scheduling
    3. Job Monitoring
  - YARN separates these functions into separate daemons.
- 
- A series of three parallel white diagonal lines are located in the bottom right corner of the slide, extending from the middle of the right edge towards the bottom left.

# YARN (Contd.)

Components of YARN:

1. **Global Resource Manager-** Assigns resources among applications for optimal resource utilization. One cluster has one instance of Resource Manager.
2. **Node Manager-** Runs on each node and communicates with Resource Manager about resource usage on the machine. It receives requests from resource manager about resource allocation to jobs and maintains life cycle of containers.
3. **Application-specific Application Master-** It is the actual instance which does processing. It requests Resource Manager for resources and works with NodeManager to get those resources for task execution. Application Master could be MapReduce or any other processing framework like Spark etc.

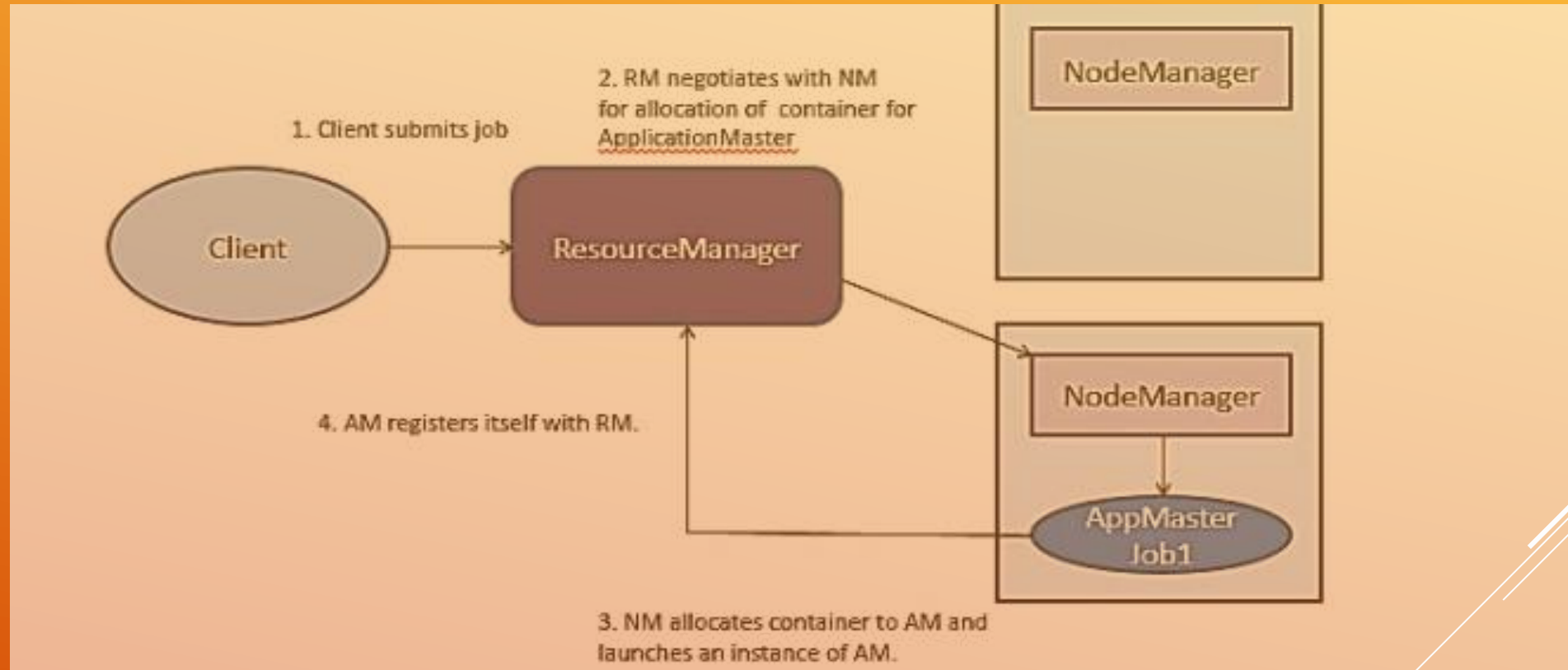
# YARN (Contd.)

- 4. **Scheduler-** It is plugged with Resource Manager to help in resource allocation. Different schedulers allocate resources using different algorithms.
- 5. **Container-** It is a set of allocated system resources (CPU Core and Memory).

Containers are allocated and managed by NodeManager and are used by tasks.

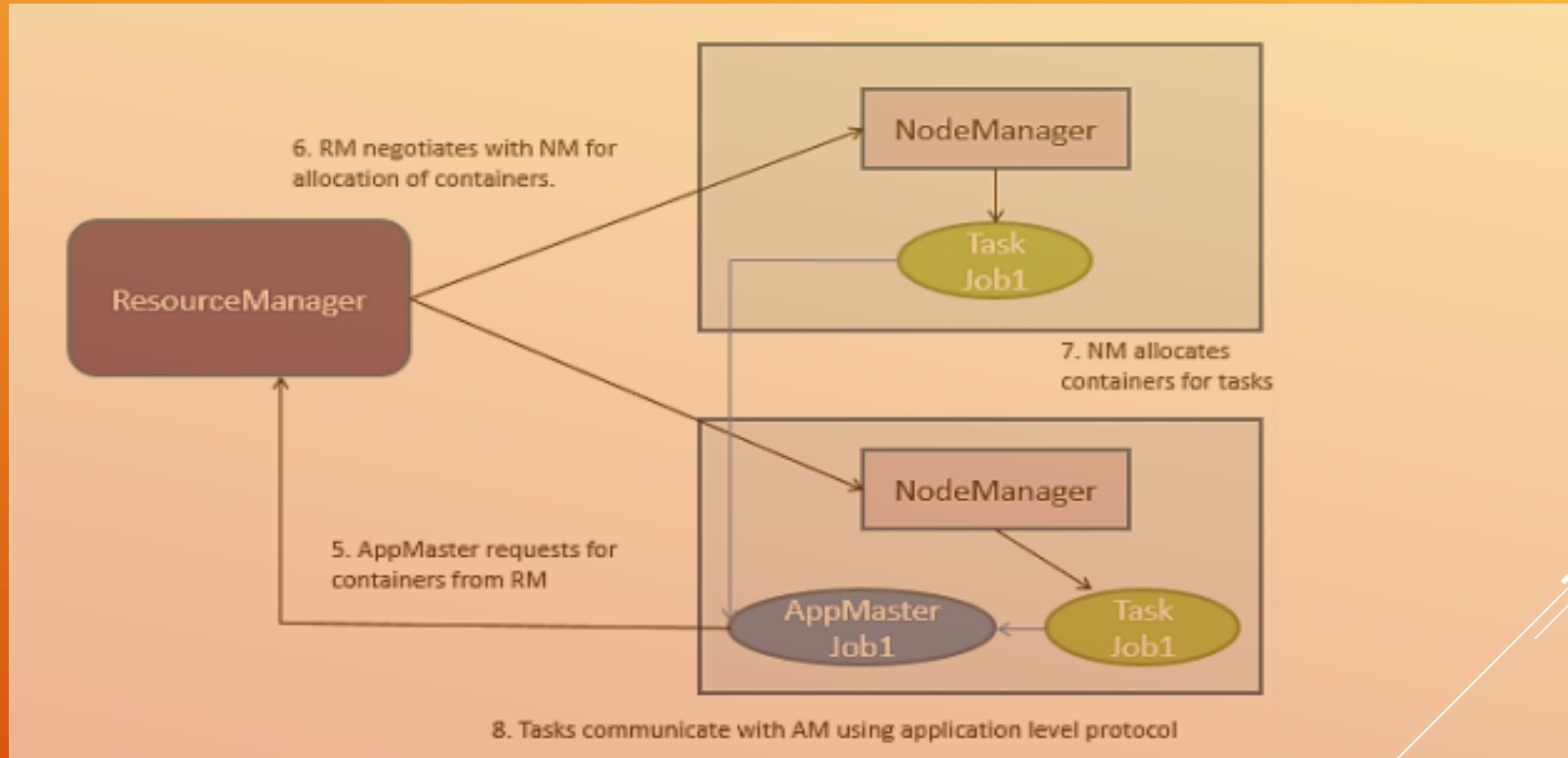
- Resource is handled by Resource Manager and Node Manager.
- Processing is handled by Application Master (MapReduce is one of the many possible types of Application Master). So, processing other than MapReduce is also possible.

# Antonomy of YARN Request





# Antonomy of YARN Request (Contd.)



# Overview

**Step 1:** Job/Application (which can be MapReduce, Java/Scala Application, DAG jobs like Apache Spark etc.) is submitted by the YARN client application to the ResourceManager daemon along with the command to start the ApplicationMaster on any container at NodeManager

**Step 2:** ApplicationManager process on Master Node validates the job submission request and hands it over to Scheduler process for resource allocation

**Step 3:** Scheduler process assigns a container for ApplicationMaster on one slave node

**Step 4:** NodeManager daemon starts the ApplicationMaster service within one of its container using the command mentioned in Step 1, hence ApplicationMaster is considered to be the first container of any application

**Step 5:** ApplicationMaster negotiates the other containers from ResourceManager by providing the details like location of data on slave nodes, required cpu, memory, cores etc.

**Step 6:** ResourceManager allocates the best suitable resources on slave nodes and responds to ApplicationMaster with node details and other details

**Step 7:** Then, ApplicationMaster send requests to NodeManagers on suggested slave nodes to start the containers

**Step 8:** ApplicationMaster then manages the resources of requested containers while job execution and notifies the ResourceManager when execution is completed

**Step 9:** NodeManagers periodically notify the ResourceManager with the current status of available resources on the node as to what information can be used by scheduler to schedule new application on the clusters

**Step 10:** In case of any failure of slave node, ResourceManager will try to allocate new container on other best suitable node so that ApplicationMaster can complete the process using new container

# Quiz

**Which of the following is not a duty of ApplicationMaster?**

1. Maintaining co-ordination among job tasks.
2. Allocating resources to the tasks.
3. Performing speculative execution.
4. Requesting to ResourceManager for resources.

# Hadoop Installation Guide

## HDP VM

- **Windows**

<https://medium.com/analytics-vidhya/hadoop-setting-up-a-single-node-cluster-in-windows-4221aab69aa6>

- **Linux**

<https://www.geeksforgeeks.org/how-to-install-hadoop-in-linux/>

- **Mac**

<https://towardsdatascience.com/installing-hadoop-on-a-mac-ec01c67b003c>

