UNIT - II : HADOOP BASICS

"Hadoop empowers organizations to store and process massive datasets across clusters of commodity hardware."

UNIT - II

1) Brief history of hadoop,
2) Apache hadoop and the hadoop ecosystem.
3) A weather dataset
4) Analyzing the data with unix tools,
5) Analyzing the data with hadoop ,
6) Understanding different Hadoop modes,
7) understanding Hadoop Features
    - 7.1) Understanding HDFS
    - 7.2) Understanding MapReduce
8) Learning the HDFS and Mapreduce Architecture
    - 8.1) Understanding the HDFS architecture
    - 8.2) Understanding the MapReduce Architecture.
    - 8.3) Understanding the HDFS and MapReduce architecture by plot.

Unit Overview

This unit introduces the history and evolution of Hadoop, the Apache Hadoop framework, and the Hadoop ecosystem. It explains how Hadoop solves Big Data storage and processing challenges using distributed computing. The unit covers the analysis of a weather dataset using Unix tools and Hadoop, different Hadoop operating modes, Hadoop features, the Hadoop Distributed File System (HDFS), MapReduce programming model, and the architectures of HDFS and MapReduce. Students will also understand the complete data processing workflow through architectural diagrams and real-world examples.

Learning Outcomes

- After completing this unit, students will be able to:
- Explain the history and evolution of Hadoop.
- Describe Apache Hadoop and its ecosystem.
- Analyze datasets using Unix commands and Hadoop.
- Differentiate between Hadoop operating modes.
- Explain the important features of Hadoop.
- Understand the architecture and working of HDFS.
- Explain the MapReduce programming model.
- Describe the HDFS and MapReduce architectures with suitable diagrams.
- Apply Hadoop concepts for distributed data storage and processing.

Importance of Studying this Unit

Modern organizations generate petabytes of data that cannot be processed efficiently using traditional systems. Hadoop provides a cost-effective, scalable, and fault-tolerant solution for storing and processing Big Data across clusters of commodity hardware. Understanding Hadoop and HDFS forms the foundation for learning Spark, Data Engineering, Cloud Computing, and Big Data Analytics.

Key Concepts

Hadoop, Apache Hadoop, Hadoop Ecosystem, HDFS, MapReduce, Distributed Computing, Cluster, NameNode, DataNode, JobTracker, TaskTracker, ResourceManager, NodeManager, Weather Dataset, Unix Commands, Hadoop Modes, Fault Tolerance, Scalability, Data Replication.

Brief History of Hadoop

In 2002, **Doug Cutting and Mike Cafarella** started to work on Apache Nutch project.

Nutch is a open source web crawler software project

After a lot of Research on Nutch, they concluded that this project is very expensive

0.5 million dollars for hardware , Running cost/month $30,000 approximately

In 2003, They came across Google's white paper on GFS(Google File System)

This paper described the architecture of Google's distributed file system, for storing the large data sets.

But it was just the half solution to their problem

In 2004, Google published one more paper on the Mapreduce Technique

This paper was another half solution and now they had complete solution for their Nutch

Brief History of Hadoop

So, they started implementing Google's Techniques (GFS &Mapreduce) as open-source in the Apache Nutch Project

In 2005, Cutting found that nutch is limited to only 20-to-40 node clusters

He soon realized 2 problems

1)Nutch wouldn't achieve its potential until it can reliably on the larger clusters

2)And that was looking impossible with just two people

Doug Cutting joined Yahoo along with Nutch Project

He wanted to provide the world with an open-source,reliable,scable,computing framework, with the help of yahoo

So at yahoo first, he separates the distributed computing parts from Nutch and formed a new project "Hadoop"

In 2007, Yahoo successfully tested Hadoop on a 1000node cluster and Started using it

In 2008, In January, Yahoo released Hadoop as an open source project to AFS(Apache Software Foundation)

Brief History of Hadoop

In July, Apache Software Foundation successfully tested a 4000 node cluster with Hadoop

In 2009, Hadoop was successfully tested to sort a PB (Petabyte) of data in less than 17 hours

Doug Cutting left the Yahoo and joined Cloudera to fulfill the challenge of spreading Hadoop to other industries

In December 2011

Apache Software Foundation released Apache Hadoop version 1.0

In August 2013

Version 2.0.6 was available

In December 2017

Apache Hadoop version 3.0 released which is currently we are having

Apache Hadoop and Hadoop Ecosystem

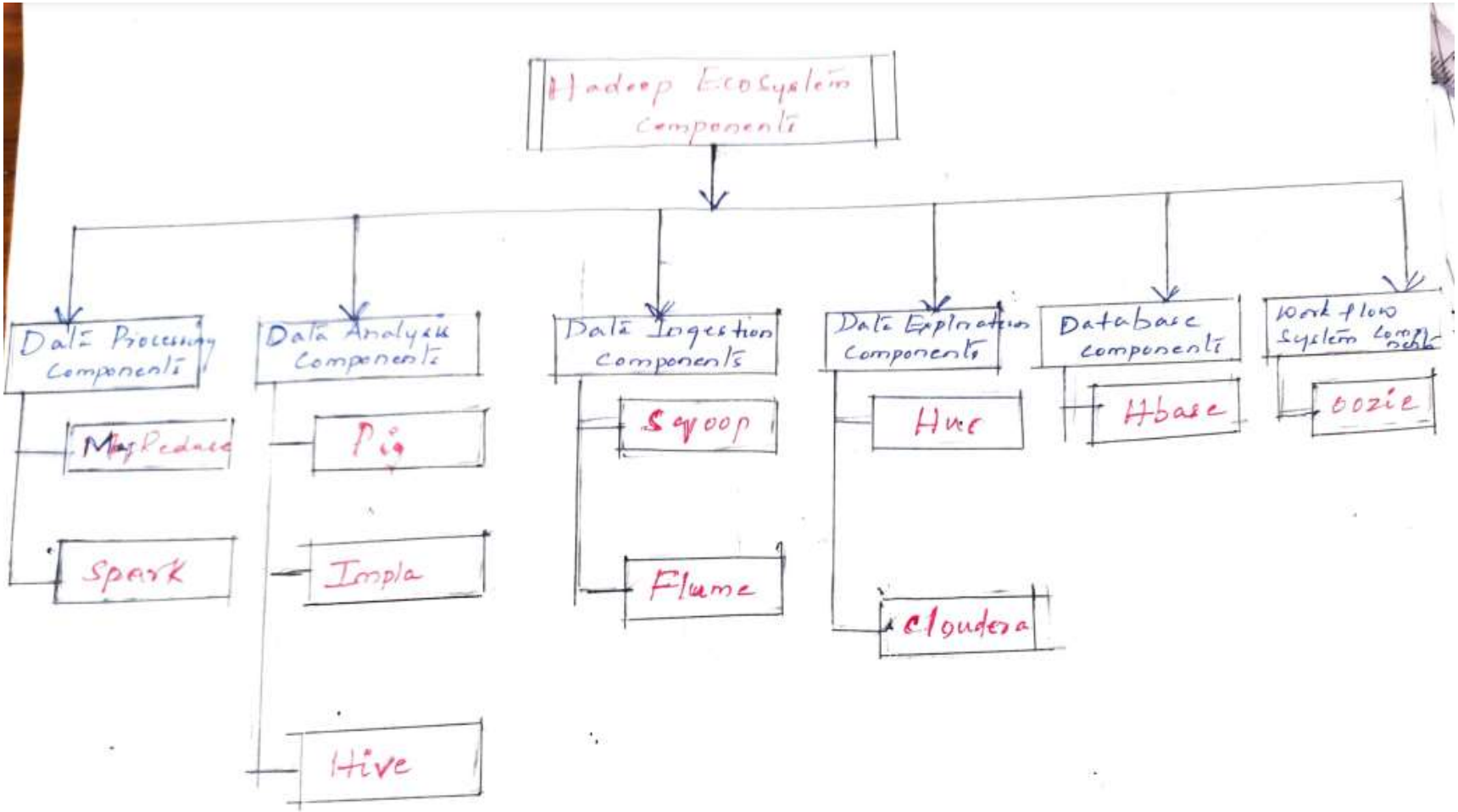
- Hadoop is a Big Data Processing Tool (or)
- Hadoop is s a open source framework that allows to store and process big data in a distributed environment across clusters of computer using simple programming models
- Two major components of Hadoop are
  1) **Hadoop Distributed File System** **(For Distributed Storage)**
  2) **Map Reduce** **(For Parallel Processing)**
- Besides these two components, many components have been introduced to deal with Big Data
- And these components are collectively called as Hadoop Ecosystem

Apache Hadoop and Hadoop Ecosystem

Hadoop Ecosystem components are categorized into

1) Data Processing Components
2) **Data Analysis Components**
3) Data Ingestion Components
4) **Data Exploration Components**
5) Data Base Components
6) **Work flow system Components**

Apache Hadoop and Hadoop Ecosystem



Apache Hadoop and Hadoop Ecosystem

1) Data Processing Components

1.1) Map Reduce

- ✓ It is a parallel programming model for processing large amounts of structured, semi-structured, and unstructured data on large clusters of commodity hardware.

1.2) Spark

- ✓ Spark is an open-source distributed engine for processing and analzing huge volumes of real time data
- ✓ Provides 100 times faster performance as compared to MapReduce
- ✓ Provides in-memory computation of data
- ✓ Used to process and analyze real time streaming data such as stock market and banking data
- ✓ Supports Machine Learning, Business Intelligence, Streaming and Batch Processing

Apache Hadoop and Hadoop Ecosystem

2) Data Analysis Components

2.1) Pig

- ✓ Pig is used to analyze data in Hadoop.
- ✓ It Provides a high level data processing language to perform numerous operations on the data.
- ✓ Language used in Pig is Pig Latin
- ✓ Pig Includes Pig Latin, a scripting language.
- ✓ Pig translates Pig Latin Scripts into MapReduce, which can ten process data in the HDFS cluster.
- ✓ Less Number of code is required to write in Pig when compared to MapReduce i.e 10 lines of pig latin script is around 200 lines of map Reduce job

2.2) Impla

- ✓ Impla is the highest performing SQL Engine which provides the fatest way to access data that I stored in Hadoop Cluster.
- ✓ Impla is Ideal for Interactive Analysis
- ✓ The Latency is very low and it is in measured in milli seconds

2.3) Hive

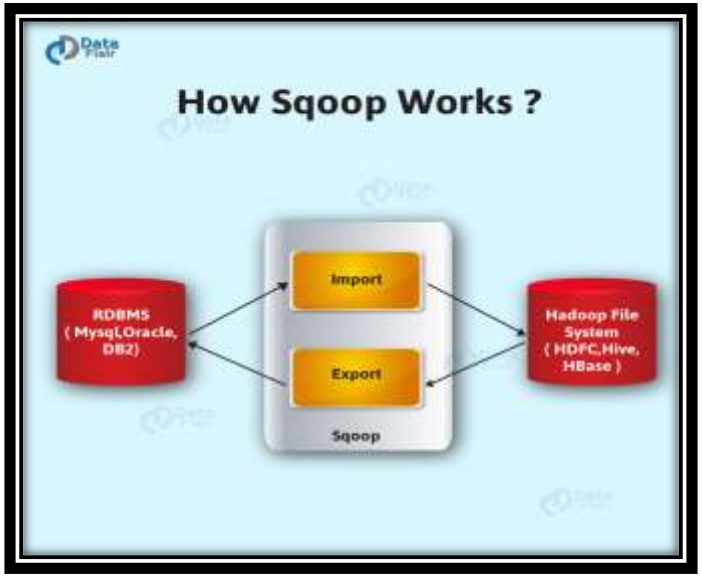
- ✓ The Hadoop ecosystem component, Apache Hive, is an open source data warehouse system for querying and analyzing large datasets stored in Hadoop files. (acts like a OLTP Tool)
- ✓ Hive do three main functions: *data summarization, query, and analysis*.
- ✓ Hive use language called HiveQL (HQL), which is similar to SQL. HiveQL automatically translates SQL-like queries into MapReduce jobs which will execute on Hadoop

Apache Hadoop and Hadoop Ecosystem

3) **Data Ingestion**

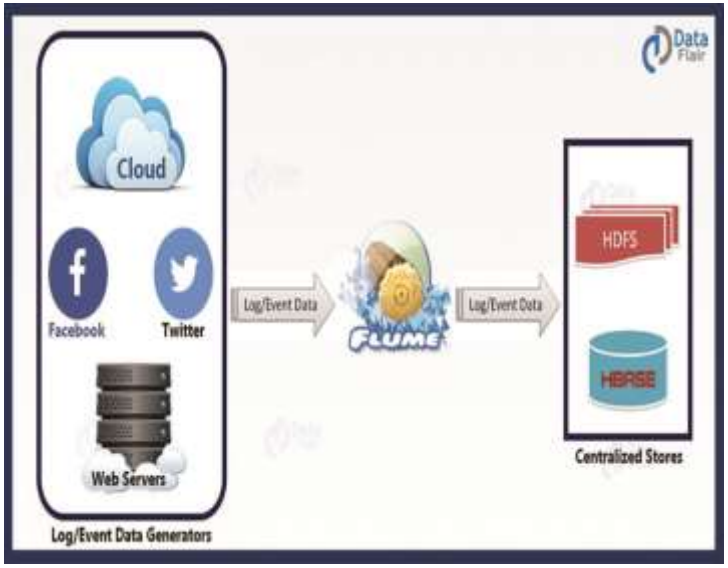
3.1) Sqoop

- ✓ Sqoop is a tool designed to transfer data between Hadoop and Relational Database Servers
- ✓ It is used to import data from relational databases such as Oracle, MYSQL to HDFS and Export data from HDFS to Relational databases



3.2) Flume

- ✓ To Ingests online streaming data from social media, log files, web server into HDFS
- ✓ **Flume** efficiently collects, aggregate and moves a large amount of data from its origin and sending it back to HDFS.

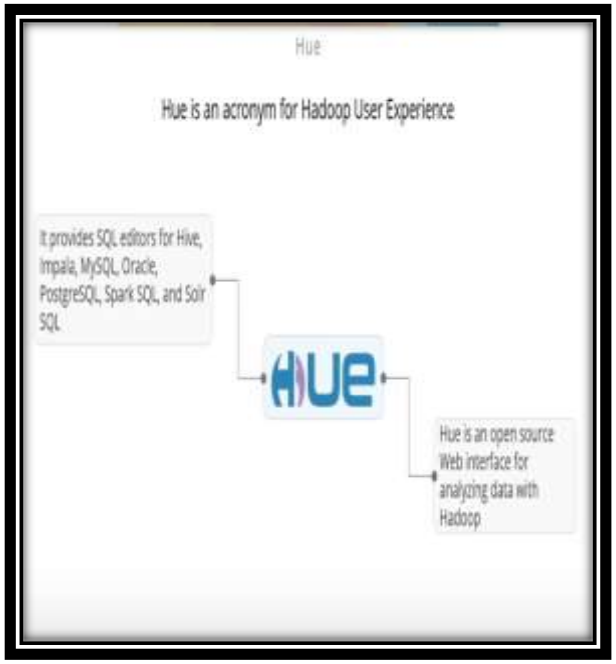


Apache Hadoop and Hadoop Ecosystem

4)Data Exploration

4.1) Hue

- ✓ Hue is an Acronym for Hadoop User Experience
- ✓ It provides SQL Editors fro Hive, Impla, MySQL, Oracle, PostgreSQL etc..
- ✓ Hue is an Open source web interface for analyzing data with Hadoop



4.2) Cloudera

- ✓ One of cloudera's near real time access products
- ✓ Users do not need SQL or Programming skills to use Cloudera Search
- ✓ Enables non-technical users to search and explore data stored in or ingested into Hadoop and HBase.

Apache Hadoop and Hadoop Ecosystem

5) Data Base Components

5.1) Hbase

- ✓ Stores data in HDFS
- ✓ A NoSQL database or Non-relational database
- ✓ Mainly used when you need random. Real-time, read/write access to your Big Data.
- ✓ Provides support to high volume of data and high throughput

Limitations of Hadoop

- ❑ Hadoop can perform only batch processing, and data will be accessed only in a sequential manner. That means one has to search the entire dataset even for the simplest of jobs.
- ❑ A huge dataset when processed results in another huge data set, which should also be processed sequentially. At this point, a new solution is needed to access any point of data in a single unit of time (random access).

Hadoop Random Access Databases

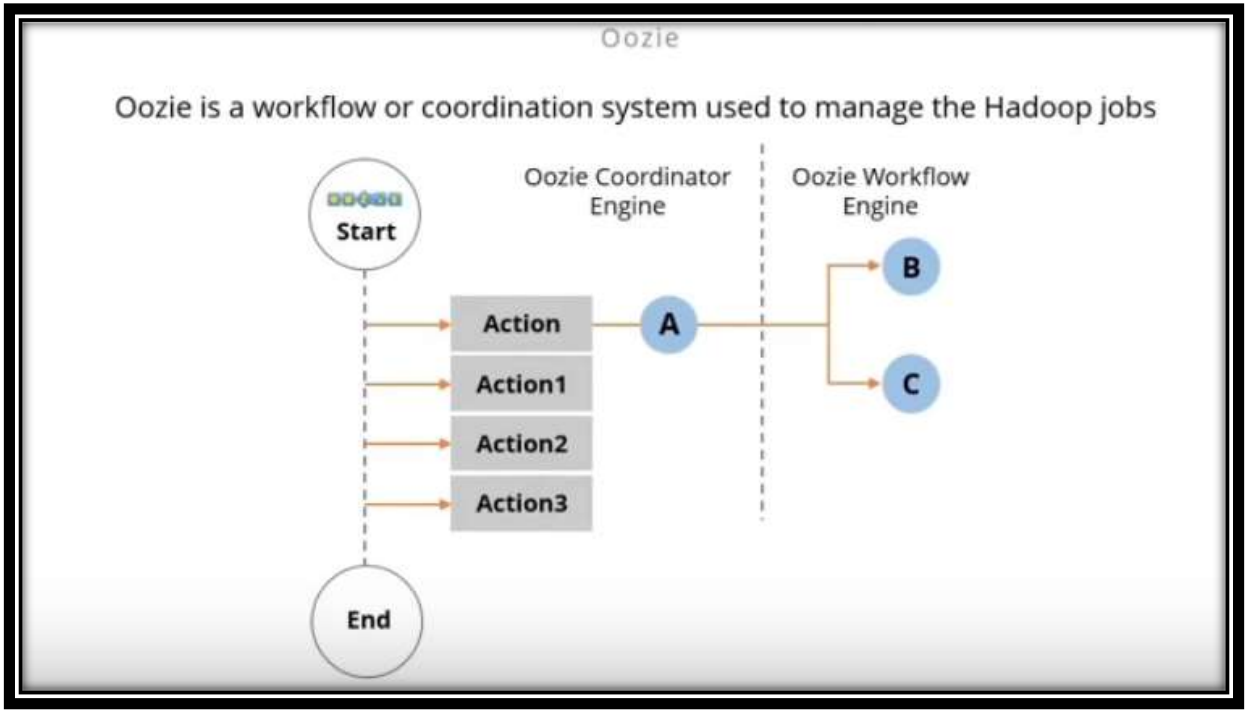
- ❑ Hbase stores huge amounts of data and access the data in a random manner.
- ❑ HBase is a distributed column-oriented database built on top of the Hadoop file system.
- ❑ One can store the data in HDFS either directly or through HBase. Data consumer reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System and provides read and write access.

Apache Hadoop and Hadoop Ecosystem

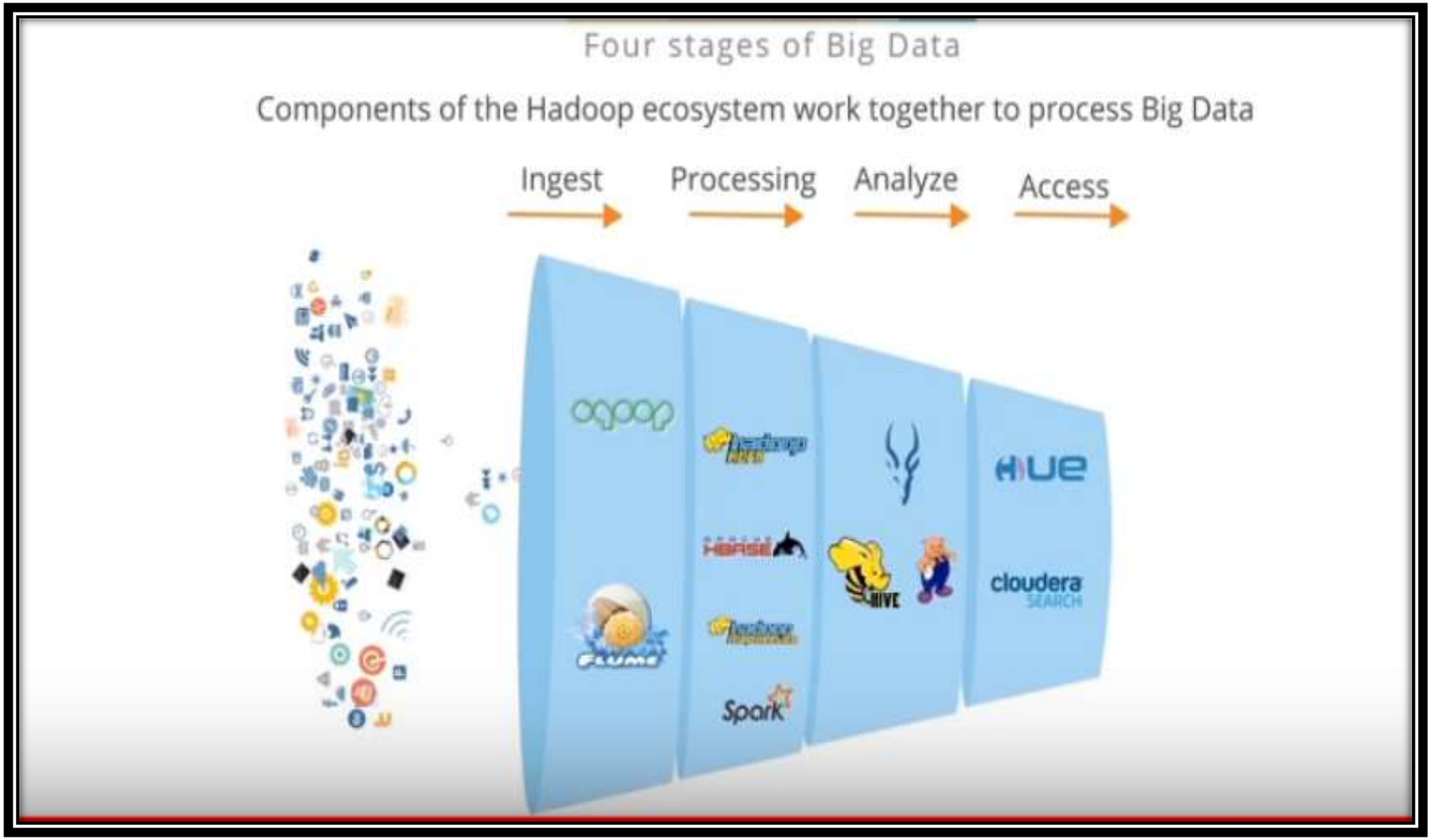
6) Work Fow System Components

6.1) Oozie

- ✓ Oozie is a workflow scheduler system to manage Hadoop Jobs



Apache Hadoop and Hadoop Ecosystem



3. A Weather Dataset

- Weather sensors collect data every hour at many locations across the globe and gather a large volume of log data.
- This data is very much suitable for analysis with MapReduce because it is **semi- structured** and **record-oriented**.
- **Data Format**

  - The data we will use is from the National Climatic Data Centre (NCDC).
  - The data is stored using a line-oriented ASCII format, in which each line is a record.

Example 2-1 shows a sample line with some of the salient fields highlighted. The line has been split into multiple lines to show each field; in the real file, fields are packed into one line with no delimiters.

3. A Weather Dataset

0067011990999991950051507004...9999999N9+00001+99999999999...

0043011990999991950051512004...9999999N9+00221+99999999999...

0043011990999991950051518004...9999999N9-00111+99999999999...

0043012650999991949032412004...0500001N9+01111+99999999999...

0043012650999991949032418004...0500001N9+00781+99999999999...

Example 2-1. Format of a National Climate Data Center record

```
0057
332130   # USAF weather station identifier
99999    # WBAN weather station identifier
19500101 # observation date
0300     # observation time
4
+51317   # latitude (degrees x 1000)
+028783  # longitude (degrees x 1000)
FM-12
+0171    # elevation (meters)
99999
V020
320      # wind direction (degrees)
1        # quality code
N
0072
1
00450    # sky ceiling height (meters)
1        # quality code
C
N
010000   # visibility distance (meters)
1        # quality code
N
9
-0128    # air temperature (degrees Celsius x 10)
1        # quality code
-0139    # dew point temperature (degrees Celsius x 10)
1        # quality code
10268    # atmospheric pressure (hectopascals x 10)
1        # quality code
```

4. Analyzing the data with unix tools

Example 2-2. A program for finding the maximum recorded temperature by year from NCDC weather records

```
#!/usr/bin/env bash
for year in all/*
do
  echo -ne `basename $year .gz`"\t"
  gunzip -c $year | \
    awk '{ temp = substr($0, 88, 5) + 0;
           q = substr($0, 93, 1);
           if (temp !=9999 && q ~ /[01459]/ && temp > max) max = temp }
         END { print max }'
done
```

4. Analyzing the data with unix tools

- The *awk* script extracts two fields from the data: the air temperature and the quality code.
- The air temperature value is turned into an integer by adding 0. Next, a test is applied to see whether the temperature is valid (the value 9999 signifies a missing value in the NCDC dataset) and whether the quality code indicates that the reading is not suspect or erroneous.
- If the reading is OK, the value is compared with the maximum value seen so far, which is updated if a new maximum is found.
- The END block is executed after all the lines in the file have been processed, and it prints the maximum value.

4. Analyzing the data with unix tools

OUTPUT

```
% ./max_temperature.sh
1901    317
1902    244
1903    289
1904    256
1905    283
```

4. Analyzing the data with unix tools

- The complete run for the century took 42 minutes in one run on a single EC2 High-CPU Extra Large Instance

- To speed up the processing, we need to run parts of the program in parallel but there are few **problems**.

1. Dividing the work into equal-size pieces isn't always easy, In this case, the file size for different years varies widely, so some processes will finish much earlier than others. Even if they pick up further work, the whole run is dominated by the longest file. A better approach is to split the file into equal size blocks.

2. Second, combining the results from independent processes may need further processing. In this case, the result for each year is independent of other years. If using the fixed-size chunk approach, the combination is more delicate. For this example, data for a particular year will typically be split into several chunks, each processed independently. We'll end up with the maximum `temperature` for each chunk, so the final step is to look for the highest of these maximums for each year.

3. Third, you are still limited by the processing capacity of a single machine

5) Analyzing the Data with Hadoop

- To take advantage of the parallel processing that Hadoop provides, we need to express our query as a MapReduce job.

Map and Reduce

- MapReduce works by breaking the processing into two phases: the map phase and 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 and the reduce function.
- The input to our map phase is the raw NCDC data.
- We choose a text input format that gives us each line in the dataset as a text **value**.
- The **key** is the offset of the beginning of the line from the beginning of the file.
- In map function, We pull out the year and the air temperature
- In this case, the map function is just a data preparation phase, setting up the data in such a way that the reducer function can do its work on it: finding the maximum temperature for each year.
- The map function is also a good place to drop bad records: here we filter out temperatures that are missing, suspect, or erroneous.
- To visualize the way the map works, consider the following sample lines of input data (Some unused

5) Analyzing the Data with Hadoop

0067011990999991950051507004...9999999N9+00001+99999999999...

0043011990999991950051512004...9999999N9+00221+99999999999...

0043011990999991950051518004...9999999N9-00111+99999999999...

0043012650999991949032412004...0500001N9+01111+99999999999...

0043012650999991949032418004...0500001N9+00781+99999999999...

These lines are presented to the map function as the key-value pairs:

(0, 0067011990999991950051507004...9999999N9+**0000**1+99999999999...)

(106, 0043011990999991950051512004...9999999N9+**0022**1+99999999999...)

(212, 0043011990999991950051518004...9999999N9-**0011**1+99999999999...)

(318, 0043012650999991949032412004...0500001N9+**0111**1+99999999999...)

(424, 0043012650999991949032418004...0500001N9+**0078**1+99999999999...)

5) Analyzing the Data with Hadoop

The map function merely extracts the year and the air temperature (indicated in bold text), and emits them as its output (the temperature values have been interpreted as integers):

(1950, 0)

(1950, 22)

(1950, −11)

(1949, 111)

(1949, 78)

5) Analyzing the Data with Hadoop

- The output from the map function is processed by the MapReduce framework before being sent to the reduce function. This processing sorts and groups the key-value pairs by key.
- So, in our example, our reduce function sees the following input:

  (1949, [111, 78])

  (1950, [0, 22, −11])

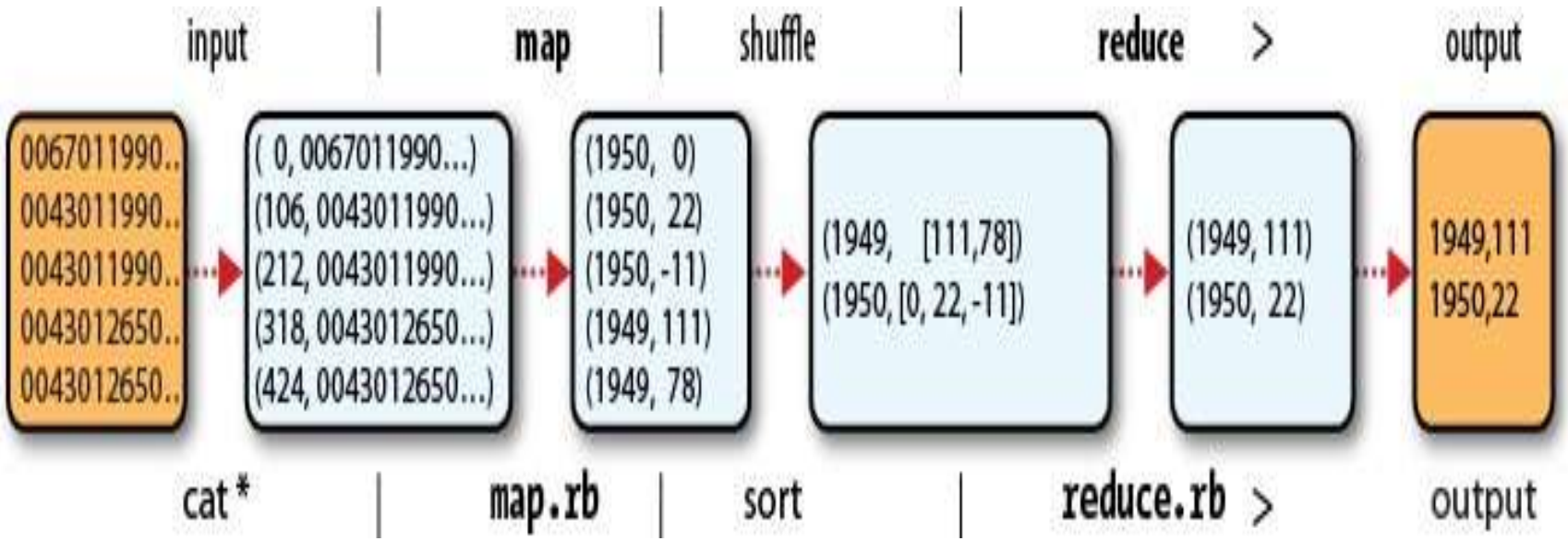
- Each year appears with a list of all its air temperature readings. All the reduce function has to do now is iterate through the list and pick up the maximum reading:

  (1949, 111)

  (1950, 22)

- This is the final output: the maximum global temperature recorded in each year.
- The whole data flow is illustrated in Figure 2-1. At the bottom of the diagram is a Unix pipeline, which mimics the whole MapReduce flow .

5) Analyzing the Data with Hadoop



5) Analyzing the Data with Hadoop

```
0029029070999991901010106004+64333+023450FM-12+000599999V0202701N015919999999N0000001N9-00781+99999102001ADDGF108991999999999999999999
0029029070999991902010113004+64333+023450FM-12+000599999V0202901N008219999999N0000001N9-00721+99999102001ADDGF104991999999999999999999
0029029070999991901010120004+64333+023450FM-12+000599999V0209991C000019999999N0000001N9-00941+99999102001ADDGF108991999999999999999999
0029029070999991902010206004+64333+023450FM-12+000599999V0201801N008219999999N0000001N9-00611+99999101831ADDGF108991999999999999999999
0029029070999991901010213004+64333+023450FM-12+000599999V0201801N009819999999N0000001N9-00561+99999101761ADDGF108991999999999999999999
0029029070999991902010220004+64333+023450FM-12+000599999V0201801N009819999999N0000001N9-00281+99999101751ADDGF108991999999999999999999
0029029070999991901010306004+64333+023450FM-12+000599999V0202001N009819999999N0000001N9-00671+99999101701ADDGF106991999999999999999999
0029029070999991902010313004+64333+023450FM-12+000599999V0202301N011819999999N0000001N9-00331+99999101741ADDGF108991999999999999999999
0029029070999991901010320004+64333+023450FM-12+000599999V0202301N011819999999N0000001N9-00281+99999101741ADDGF108991999999999999999999
0029029070999991902010406004+64333+023450FM-12+000599999V0209991C000019999999N0000001N9-00331+99999102311ADDGF108991999999999999999999
0029029070999991901010413004+64333+023450FM-12+000599999V0202301N008219999999N0000001N9-00441+99999102261ADDGF108991999999999999999999
0029029070999991902010420004+64333+023450FM-12+000599999V0202001N011819999999N0000001N9-00391+99999102231ADDGF108991999999999999999999
0029029070999991901010506004+64333+023450FM-12+000599999V0202701N004119999999N0000001N9+00001+99999101821ADDGF104991999999999999999999
0029029070999991901010513004+64333+023450FM-12+000599999V0202701N002119999999N0000001N9+00061+99999102591ADDGF104991999999999999999999
0029029070999991901010520004+64333+023450FM-12+000599999V0202301N004119999999N0000001N9+00001+99999102671ADDGF104991999999999999999999
0029029070999991902010606004+64333+023450FM-12+000599999V0202701N006219999999N0000001N9+00061+99999102751ADDGF103991999999999999999999
0029029070999991901010613004+64333+023450FM-12+000599999V0202701N006219999999N0000001N9+00061+99999102981ADDGF100991999999999999999999
0029029070999991901010620004+64333+023450FM-12+000599999V0203201N002119999999N0000001N9-00111+99999103191ADDGF100991999999999999999999
0029029070999991903010706004+64333+023450FM-12+000599999V0209991C000019999999N0000001N9-00331+99999103341ADDGF100991999999999999999999
0029029070999991901010713004+64333+023450FM-12+000599999V0209991C000019999999N0000001N9-00501+99999103321ADDGF100991999999999999999999
0029029070999991903010720004+64333+023450FM-12+000599999V0202001N009819999999N0000001N9-00441+99999103321ADDGF100991999999999999999999
0029029070999991901010806004+64333+023450FM-12+000599999V0202301N009819999999N0000001N9-00281+99999103221ADDGF108991999999999999999999
0029029070999991903010813004+64333+023450FM-12+000599999V0202301N011819999999N0000001N9-00331+99999103201ADDGF108991999999999999999999
0035029070999991901010820004+64333+023450FM-12+000599999V0202301N013919999999N0000001N9-00331+99999102991ADDGF108991999999999999999999MW17
0029029070999991901010906004+64333+023450FM-12+000599999V0209991C000019999999N0000001N9-00501+99999102871ADDGF108991999999999999999999
0029029070999991903009013004+64333+023450FM-12+000599999V0209991C000019999999N0000001N9-00331+99999102661ADDGF108991999999999999999999
0029029070999991901010920004+64333+023450FM-12+000599999V0201801N009819999999N0000001N9-00281+99999102391ADDGF108991999999999999999999
0029029070999991903011006004+64333+023450FM-12+000599999V0202301N009819999999N0000001N9-00441+99999101601ADDGF100991999999999999999999
0029029070999991901011013004+64333+023450FM-12+000599999V0202301N011819999999N0000001N9-00441+99999101481ADDGF100991999999999999999999
0029029070999991901011020004+64333+023450FM-12+000599999V0202301N013919999999N0000001N9-00441+99999101381ADDGF100991999999999999999999
0029029070999991904011106004+64333+023450FM-12+000599999V0202501N006219999999N0000001N9-00391+99999101061ADDGF100991999999999999999999
```

5) Analyzing the Data with Hadoop

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
//import org.apache.hadoop.mapreduce.Reducer.Context;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;
import java.util.Map;
import java.util.StringTokenizer;
```

5) Analyzing the Data with Hadoop

```
public class WeatherData
{
        public static class WDMapper extends Mapper<LongWritable,Text,Text,IntWritable>
        {
                private static final int MISSING = 9999;
                public void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException
                {
                        String line = value.toString();
                        String year= line.substring(15,19);
                        int airTemperature;
                        if(line.charAt(87) == '+')
                                airTemperature = Integer.parseInt(line.substring(88,92));
                        else
                                airTemperature = Integer.parseInt(line.substring(87,92));
                        String quality = line.substring(92,93);
                        if(airTemperature != MISSING && quality.matches("[01459]"))
                                context.write(new Text(year),new
                                IntWritable(airTemperature));
                }
        }
```

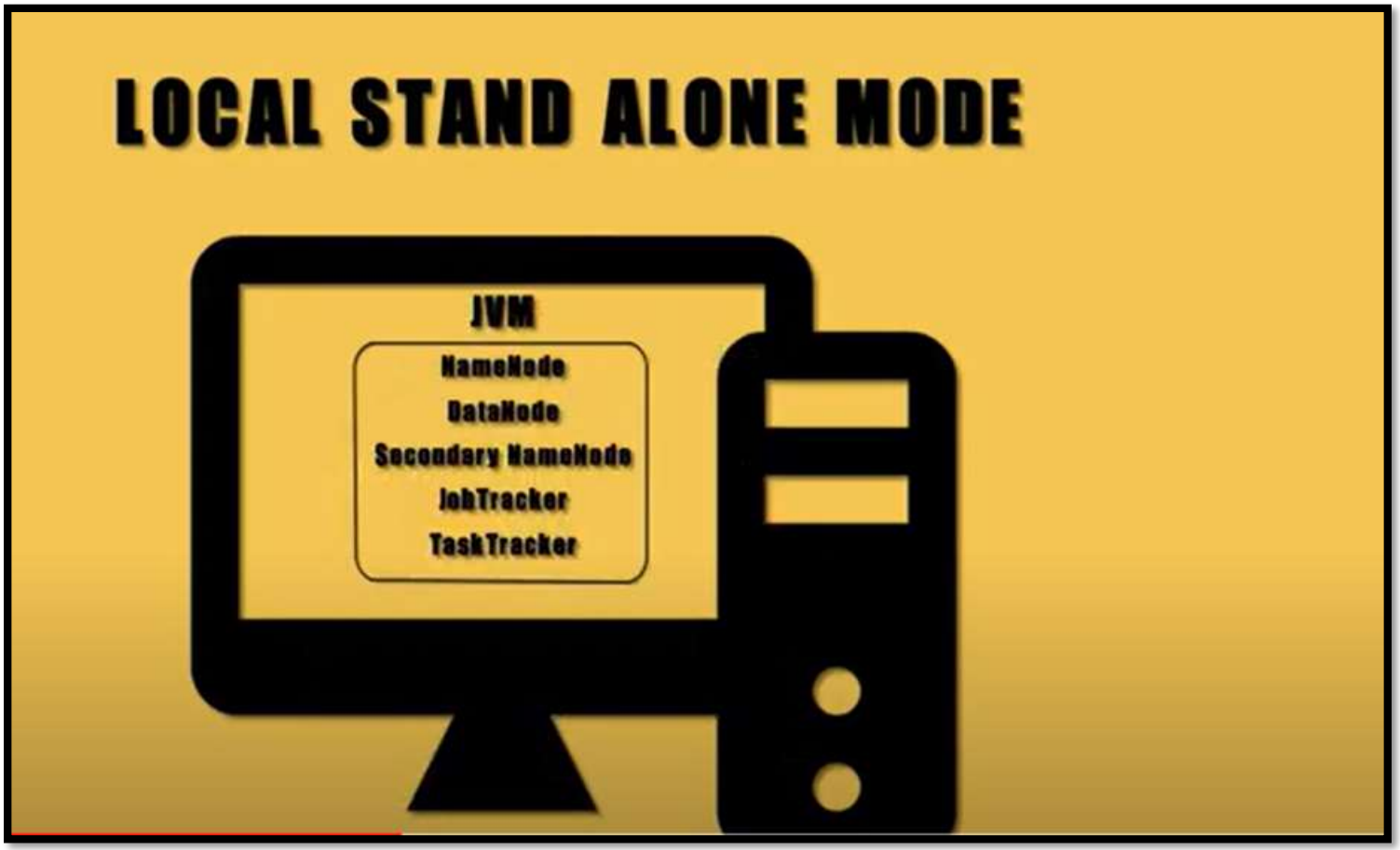
5) Analyzing the Data with Hadoop

```
public static class WDReducer extends Reducer<Text,IntWritable,Text,IntWritable>
{

        public void reduce(Text key, Iterable<IntWritable> values,
        Context context) throws IOException, InterruptedException
        {
                int maxValue = Integer.MIN_VALUE;
                for(IntWritable value : values)
                {
                        maxValue = Math.max(maxValue,value.get());
                }
                        context.write(key, new IntWritable(maxValue));

        }
}
```

7) Understanding different Hadoop modes

- Hadoop can run in three different modes as below-
  - ✓ Local (Standalone) Mode.
  - ✓ Pseudo-Distributed Mode
  - ✓ Fully-Distributed Mode

All the Components of HDFS like Name Node, Data Node, Job Tracker, Task Tracker in Single JVM



1. Local Mode or Standalone Mode

- Standalone mode is the default mode in which Hadoop run.
- Standalone mode is mainly used for debugging where you don't really use HDFS.
- You can use input and output both as a local file system in standalone mode.
- You also don't need to do any custom configuration in the files- **mapred-site.xml, core-site.xml, hdfs-site.xml**.
- Standalone mode is usually the fastest Hadoop modes as it uses the local file system for all the input and output.

- Here is the summarized view of the standalone mode
  - ✓ Used for debugging purpose
  - ✓ HDFS is not being used
  - ✓ Uses local file system for input and output
  - ✓ No need to change any configuration files
  - ✓ Default Hadoop Modes

2) Pseudo-distributed Mode

- The **pseudo-distribute mode** is also known as a **single-node cluster** where both Name Node and Data Node will reside on the same machine.
- In pseudo-distributed mode, all the Hadoop daemons will be running on a single node.
- Such configuration is mainly used while testing when we don't need to think about the resources and other users sharing the resource.

- Here is the summarized view of pseudo distributed Mode-
  - ✓ Single Node Hadoop deployment running on Hadoop is considered as pseudo distributed mode
    All the master & slave daemons will be running on the same node.
  - ✓ Mainly used for testing purpose.
  - ✓ Replication Factor will be ONE for blocks.
  - ✓ Changes in configuration files will be required for all the three files- **mapred-site.**xml**, core-site.**xml**,** hdfs**-site.xml**

The Java Virtual Machine is called JVM, is an **abstract computing machine** or **virtual machine interface** that drives the java code.

Each Component of HDFS like Name Node, Data Node, Job Tracker and Task Tracker will be running in separate a JVM of the single Machine

3) Fully-Distributed Mode (Multi-Node Cluster)

- This is the **production mode of Hadoop** where multiple nodes will be running. Here data will be distributed across several nodes and processing will be done on each node.
- Master and Slave services will be running on the separate nodes in fully-distributed Hadoop Mode.
  - ✓ Production phase of Hadoop
  - ✓ Separate nodes for master and slave daemons
  - ✓ Data are used and distributed across multiple nodes

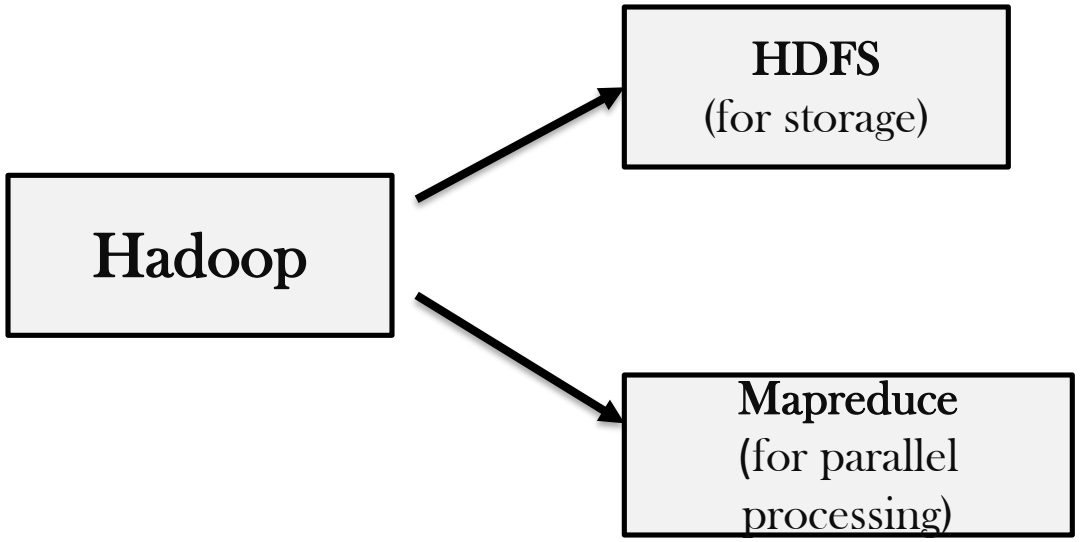

FULLY DISTRIBUTED MODE
JVM
NameNode
JVM
JobTracker
JVM
Secondary
NameNode
JVM
DataNode
JVM
TaskTracker

Conclusion

- Single mode runs a single process on one system and is not distributed. Pseudo-mode also runs on one system but it creates a cluster simulation.
- Single mode does not use hdfs, it used the local filesystem instead. But in pseudo-mode, hdfs is used. This how to storage and file system differs between these two modes.
- Pseudo mode runs virtual nodes on the same system. In single mode, only one node is run and this mode is mainly used for debugging process.

6) Understanding Hadoop Features

- Doug cutting" and "mike cafarella" introduced Hadoop in 2006.
- Hadoop is a open source framework for "storing and processing" huge dataset.
- Hadoop is a open source tool that allows to store and process big data in a distributed environment across cluster of computer using simple programming model for processing.
- Core components of Hadoop are: "HDFS" and "Mapreduce"



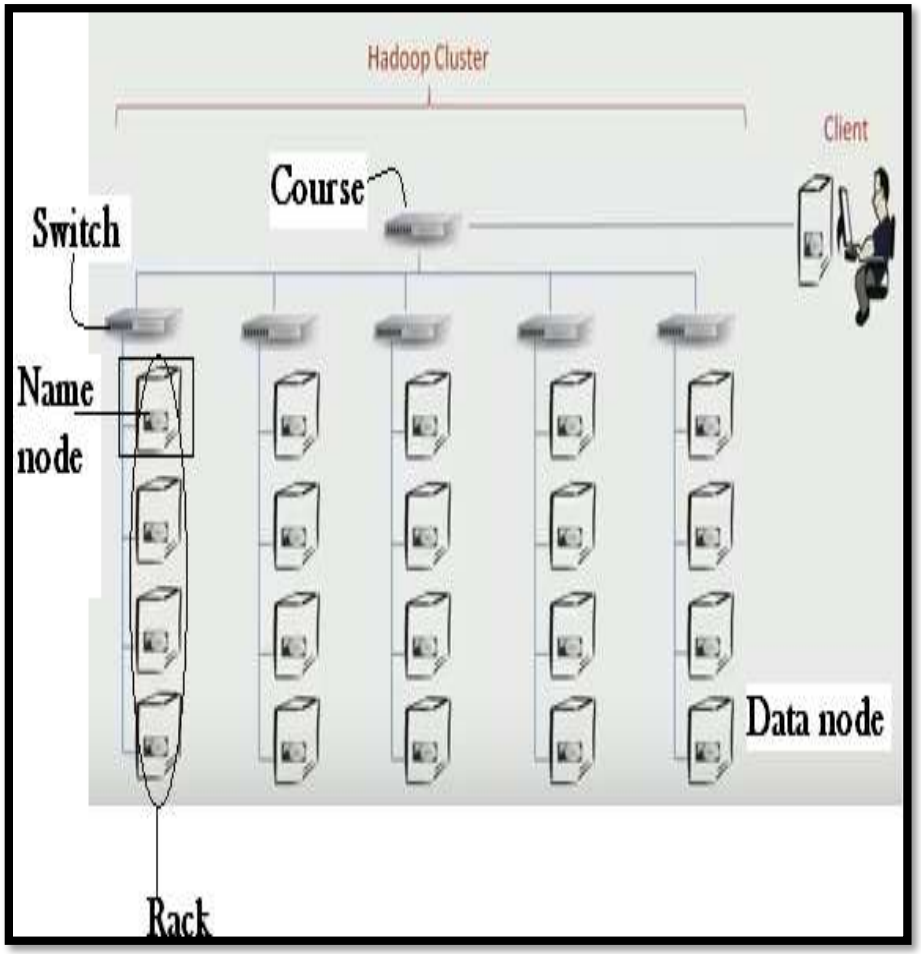
- The name “Hadoop” is not an acronym, it is madeup name. The projects creator “Doug cutting” explains how the name came about.
- The name, my kid gave to the stuffed yellow elephant. Short, relatively easy to pronounce, meaningless and not used else anywhere. Those are my naming criteria. Kids are good at generating search names

6.1 UNDERSTANDING HDFS

DISTRIBUTED STORAGE

Data is stored in a Distributed Manner

- Instead of relying on a single large machine , HDFS uses a network of several smaller machine and add them to a cluster this approach is called Horizontal scaling.
- So, we use a software that combines the storage capacity of the entire network into a single large system i.e network based file system and that is HDFS.



Hadoop client will send a request to name node that it want to create a file along **with target directory name and file name**

By receiving the request the **name node will perform various checks like is the file already exist and client has right permission to create a file.**

Name node does all this because it maintains an image of entire HDFS namespace into memory we call **it file system image.**

If all test pass the name node will **create an entry for new file and return success to client.**

Now, empty file is created.

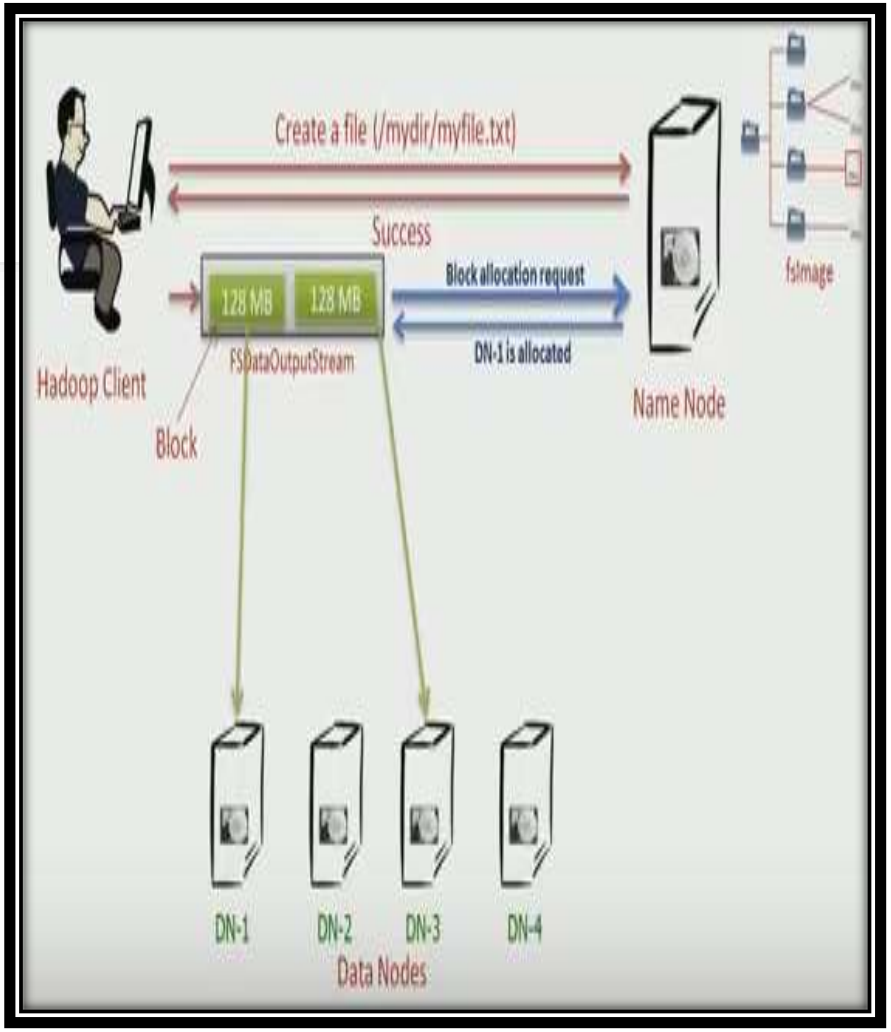
The client start writing **fsdata output stream.**

This stream internally buffers data until you accumulate the reasonable amount of data with default size **128 MB as a block**.

Each block send **block allocation request** to name node asking for location for storing that block name node will allocate a data node and send back data node name to the streamer.

Now streamer will send the **data to the data node.**

If file is large it creates a new block allocation and does the same work.



SCALABILITY

- As HDFS stores the large size data over multiple nodes, so when the requirement of data storing is increased or decreased the number of nodes can be scaled up or scaled down in a cluster.
- The vertical and Horizontal scalability is the 2 different mechanisms available to provide scalability in the cluster.
- Vertical scalability means adding the resource like Disk space, RAM on the existing node of the cluster.
- On another hand in Horizontal scaling, we increase the number of nodes in the cluster and it is more preferable since we can have hundreds of nodes in a cluster.

SCALABILITY

Scalability

- The primary benefit of Hadoop is its **Scalability.**
- One can easily scale the cluster by adding more nodes.
- There are two types of Scalability in Hadoop: **Vertical and Horizontal**

Vertical scalability

- It is also referred as "**scale up".** In vertical scaling, you can increase the **hardware capacity of the individual machine.**
- In other words, you can add more RAM or CPU to your existing system to make it more robust and powerful.

Horizontal scalability

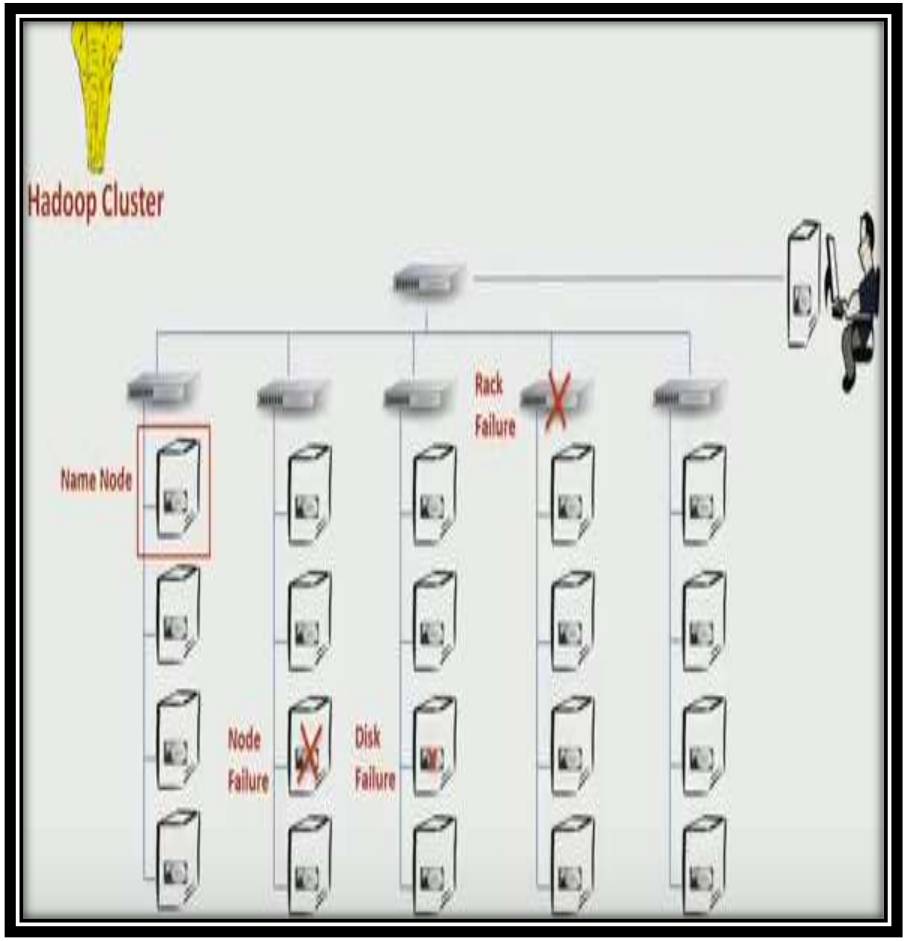
- It is also referred as "**scale out"** is basically the addition of more machines or setting up the cluster.
- In horizontal scaling instead of increasing hardware capacity of individual machine **you add more nodes to existing cluster**

FAULT TORELANCE

- HDFS is **highly fault-tolerant and reliable**.
- HDFS creates **replicas of file blocks** depending on the replication factor and stores them on different machines.
- If any of the machines containing data blocks fail, other Data Nodes containing the replicas of that data blocks are available.
- ***Thus ensuring no loss of data and makes the system reliable even in unfavorable conditions.***

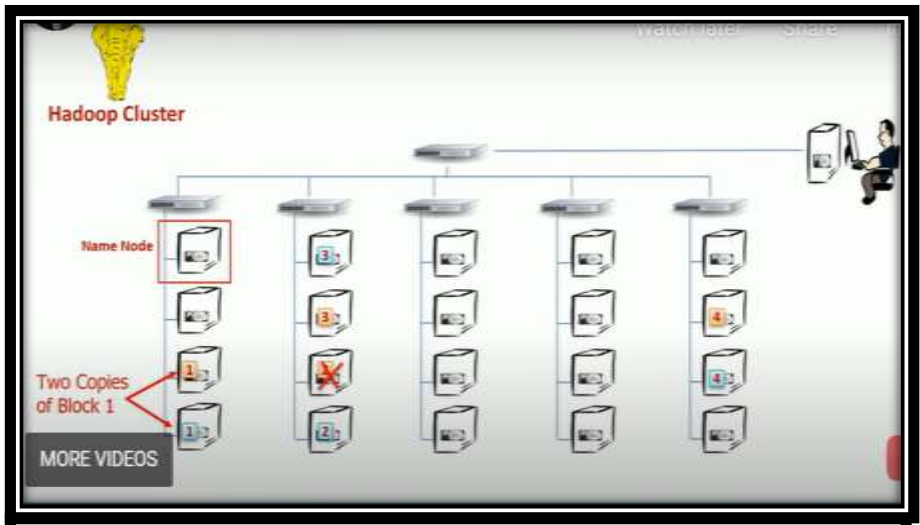
Types of Failures

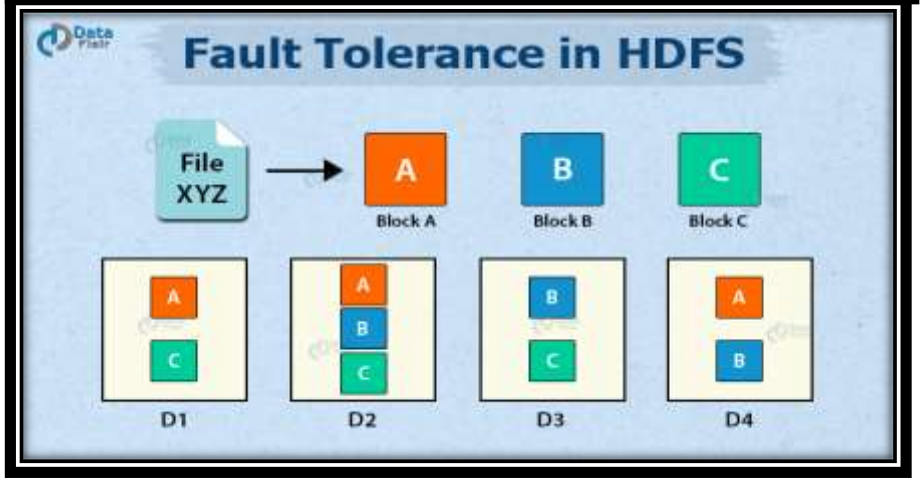
- Different Types of Failures that can happen in Hadoop Cluster are
  - ✓ Disk Failure
  - ✓ Node Failure
  - ✓ Rack Failure
  - ✓ Name Node Failure
- Replication helps to protect Data from **Disk and Node Failure**



FAULT TOLERANCE

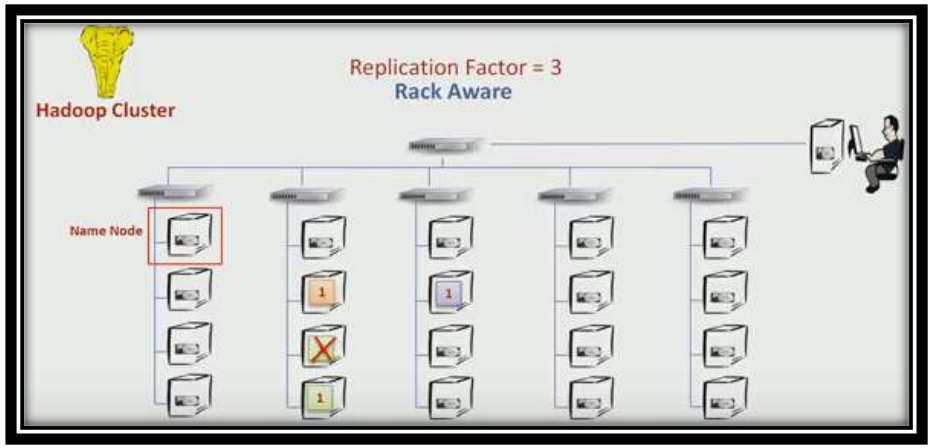
- **Fault tolerance refers to the ability of the system to work or operate even in case of unfavorable conditions (like components failure).**
- HDFS is **highly fault-tolerant**. it handles faults by the process of **replica creation**.
- It creates a replica of users' data on different machines in the HDFS cluster.
- So whenever if any machine in the cluster goes down, then data is accessible from other machines in which the same copy of data was created.
- We can set this replication on file to file basic, (by default its value is 3)
- if we set replication value as 3, HDFS will automatically makes 3 copies of each block for a file.
- Hadoop will also ensure that it keeps these three copies on rack Aware by placing atleast one node in different rack, so that if a complete rack get failed the data can be backed up easily.

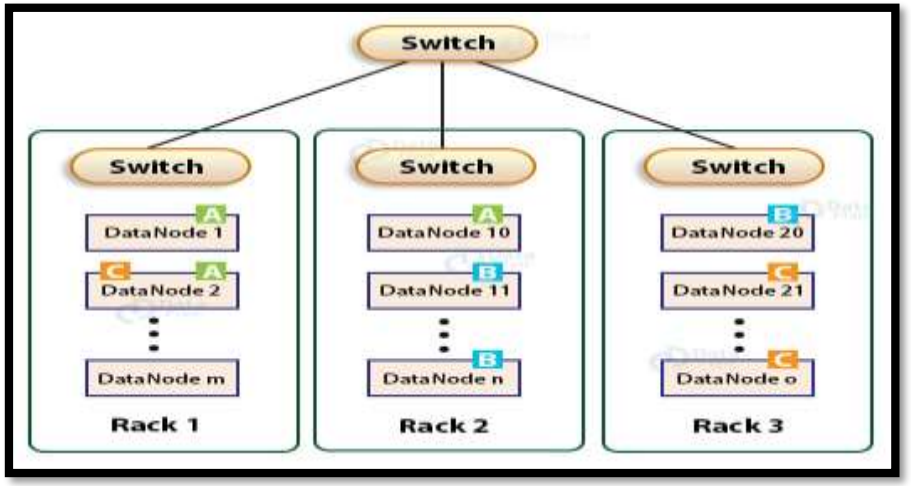




Replica placement via Rack awareness in Hadoop

- HDFS stores replicas of data blocks of a file to provide **fault tolerance and high availability.**
- ***If we store replicas on different nodes on the same rack, then it improves the network bandwidth, but if the rack fails (rarely happens), then there will be no copy of data on another rack.***
- Again, if we store replicas on unique racks, then due to the transfer of blocks to multiple racks while writes increase the cost of writes.
- Therefore, **NameNode on multiple rack cluster maintains block replication by using inbuilt Rack awareness policies which says:**
  - ✓ **Not more than one replica be placed on one node.**
  - ✓ **Not more than two replicas are placed on the same rack.**
- For the common case where the replication factor is three, the block replication policy put the first replica on the local rack, a second replica on the different DataNode on the same rack, and a third replica on the different rack.





HIGH AVAILABILITY

- The High availability feature of Hadoop ensures the availability of data even during NameNode or DataNode failure.
- Since HDFS creates replicas of data blocks, if any of the DataNodes goes down, the user can access his data from the other DataNodes containing a copy of the same data block.
- Also, if the active NameNode goes down, the passive node takes the responsibility of the active NameNode. Thus, data will be available and accessible to the user even during a machine crash.

HIGH AVAILABILITY

Although Rack, node and disk failure happens, these faults doesn't bring the entire system down your Hadoop cluster remains available, which shows Hadoop is rich in high availability.

What happens when the name node fails?

- We know that name node maintains the file system namespace.
- It also manages the file to block mapping.
- Every client interaction start with name node, so if name node fails we can't use Hadoop cluster.
- Name node is a single point of failure in cluster.
- To achieve high availability we need to protect it against name node failure

How to protect against name node(NN) failure?

- The solution is to back up the namespace content (HDFS namespace information)

HDFS namespace information:

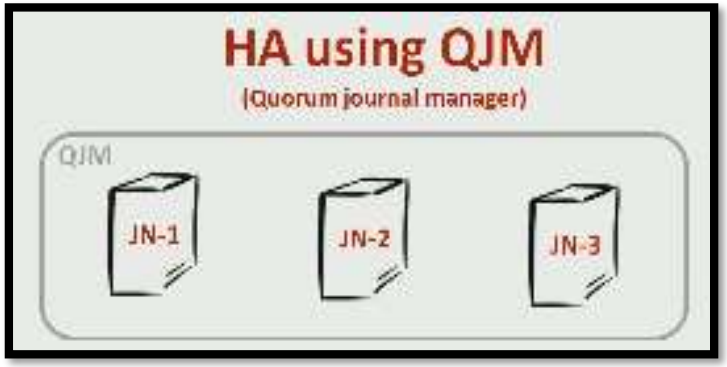
- All the information that name node maintains, should be continuously backed up at some other place.
- so that in the case of failure you have all necessary information to start a new name node on different machine.

HIGH AVAILABILITY

- There are many ways, one among them is “HA using QJM(quorum journal manager)”.
- QJM is a set of atleast 3 machines each of this is configured to execute the journal node.
- Journal node is a light weight software, to write edit log entries to the QJM.

Standby name node:

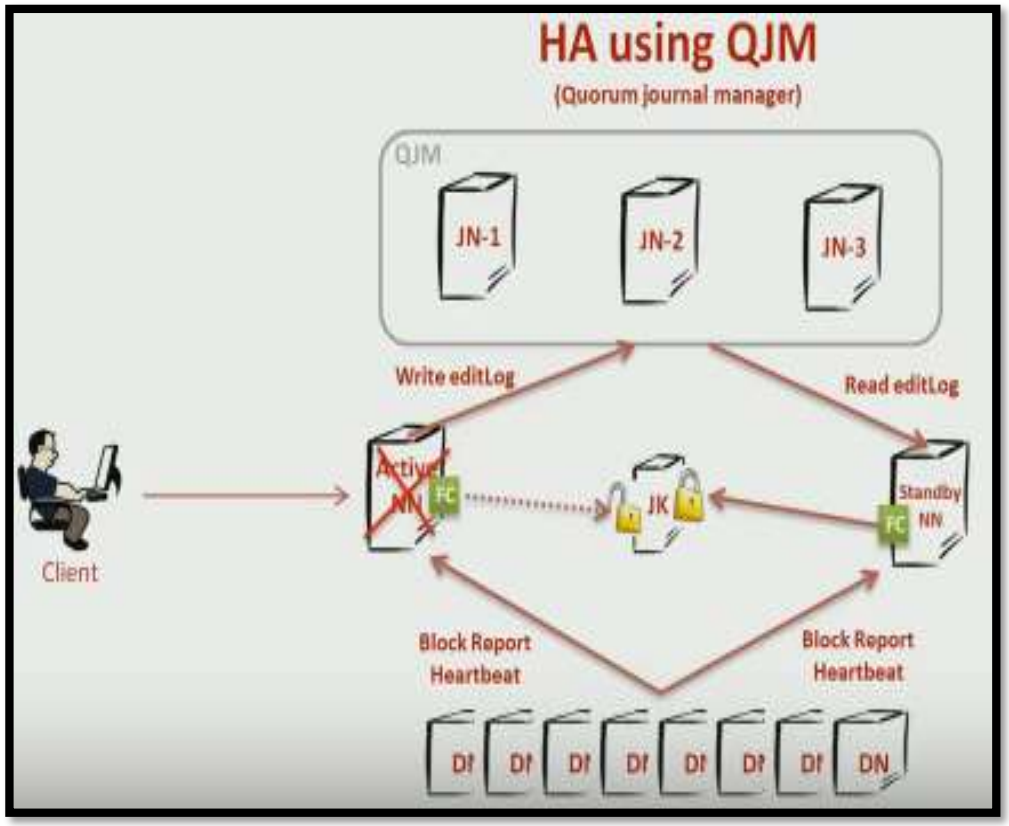
- To minimize the time to start new name node, we should already have a standby computer pre-configured and ready to take over the role of name node.
- We add a new machine to the cluster and make it stand by name node, we also configure it to keep reading the edit log from the QJM and keep itself updated.
- This configuration makes the standby ready to take up the active name node role in just a few seconds.
- All the data node are configured to send the block report to both active and standby Name node.
- The block report is a kind of health information of the block maintained by the data node.



HIGH AVAILABILITY

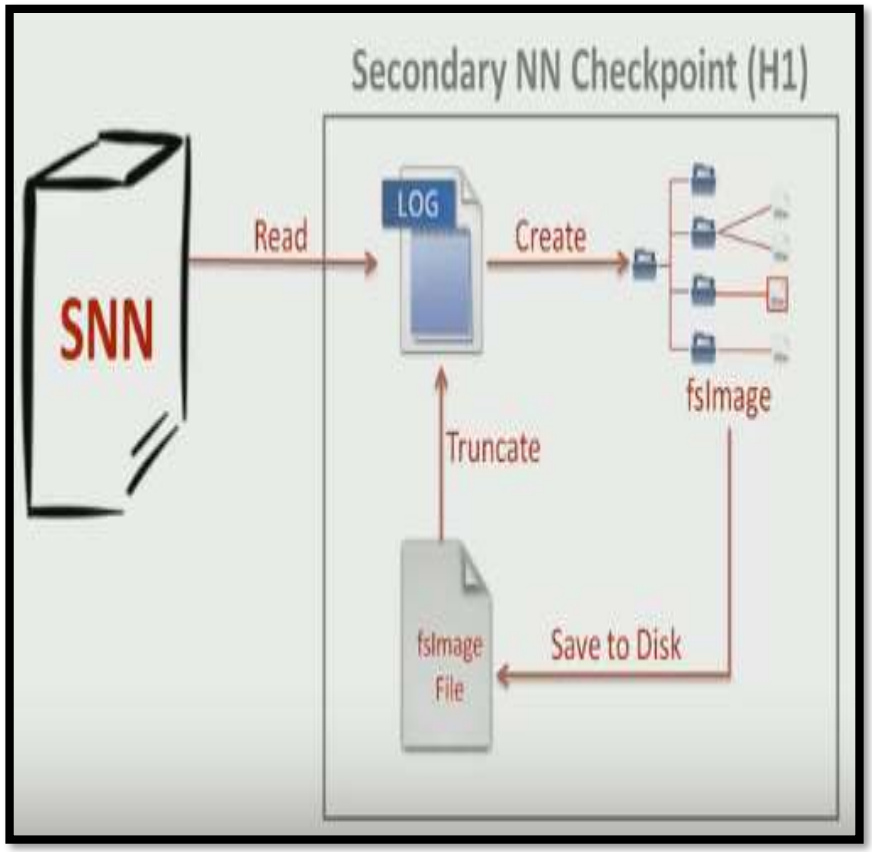
How standby NN knows that active NN failed?

- We achieve this by placing a zookeeper and two failover controller on each name node.
- The zookeeper failover controller of active NN maintains a lock in zookeeper, the standby NN keeps trying to get the lock.
- But since active NN already maintain it its not possible for standby NN.
- Incase if active fails or crashes the lock acquired by active NN expire and standby succeed to get a lock.
- As soon as standby get the lock it start to transition from extend by active NN because it knows that active has failed. i.e Whenever the active NN fails, standby NN act as active NN.



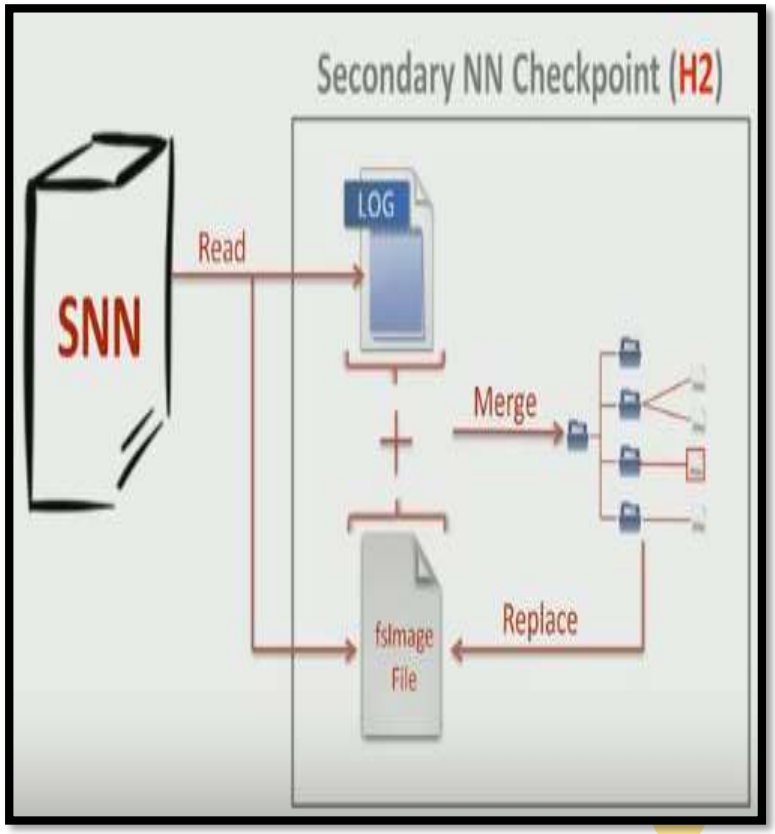
SECONDARY NAME NODE

- Secondary NN differs from standby NN.
- Standby NN is a backup for Active NN, whereas secondary is not for that.
- Whenever we restart the name node we will lose the fsImage because it is an in-memory image (In-memory fsImage is the latest and updated picture of Hadoop file system namespace).
- Of course we have editlog to create new fsImage, but it may take more time to read the editlog because the log keeps growing bigger & bigger everyday this directly impact the restart time for a name node which may take hours of time.
- So to solve this problem we came with secondary NN.



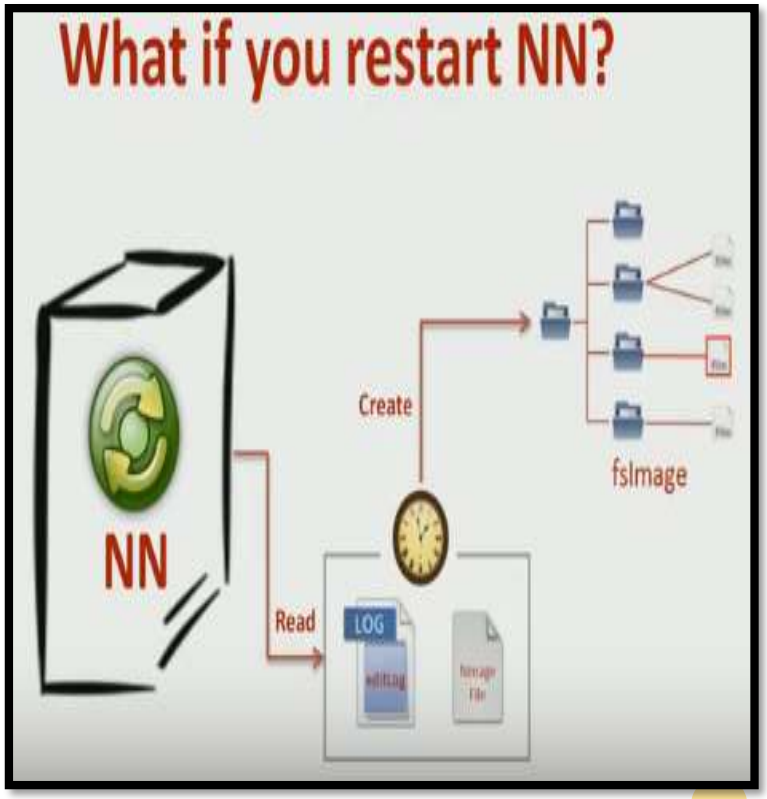
Secondary Name Node

- Secondary NN performs a **checkpoint activity every hour**.
- During the check point, the secondary NN will read the editlog and create the **latest file system image and save it to disk.**
- This state is exactly same as the In-memory fsImage. We call it on-disk fsimage.
- Once we have this on-disk fsImage the secondary NN truncate the edit log because all the changes are already applied, After an hour secondary NN will read the on-disk fsImage and apply all the changes from the editlog that we accumulated during the last 1 hour.
- It will replace the code on-disk fsImage with the new one and truncate the edit log once again. So the checkpoint activity is noting but the merging of an ondisk fsImage and editlog.



Secondary Name Node

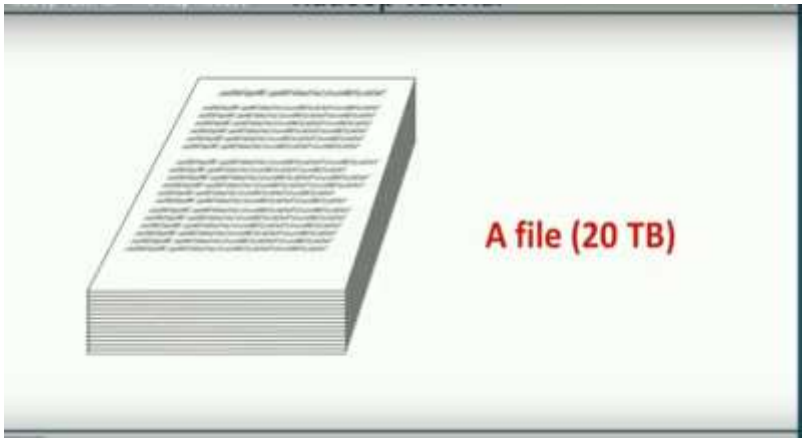
- The checkpoint doesn't take much time because we have just an hour old editlog to apply to the ondisk fsImage.
- The time to read fsImage is short because the fsImage is small compared to the editlog.
- The editlog is like a general ledger that records every transaction.
- The fsImage is like a balance sheet that shows the final state.
- So, the fsImage is small. In case of restart the name node also performs the same checkpoint activity that it can finish in a short time.
- Therefore the whole purpose of secondary NN and checkpoint is to minimize and restart time for the name node.



DATA LOCALITY

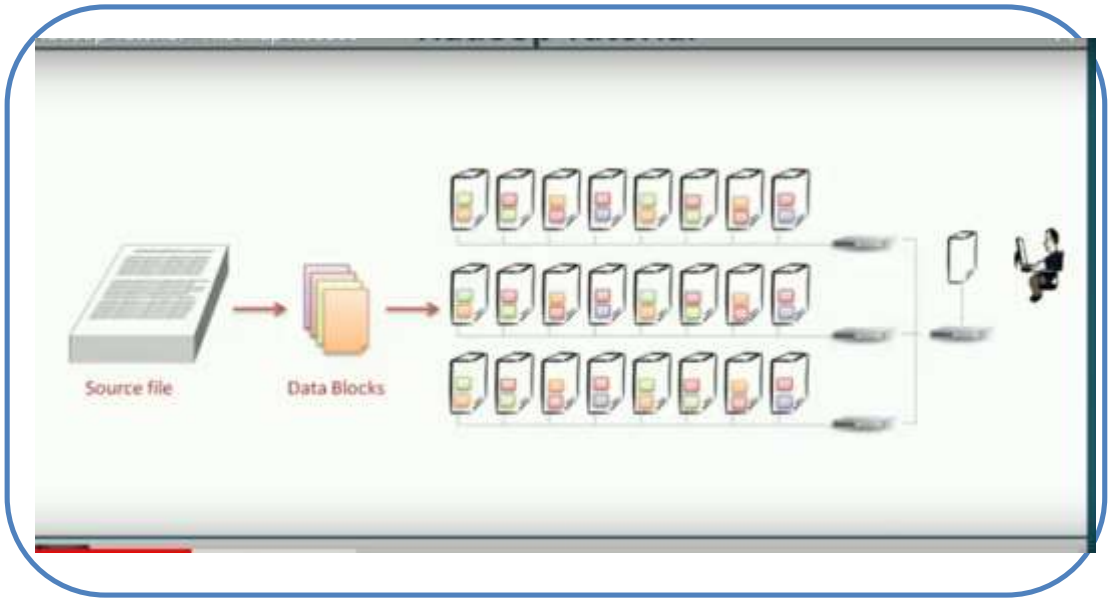
- In Hadoop, Data locality is the process of moving the computation close to where the actual data resides on the node, instead of moving large data to computation.

- This minimizes network congestion and increases the overall throughput of the system. .

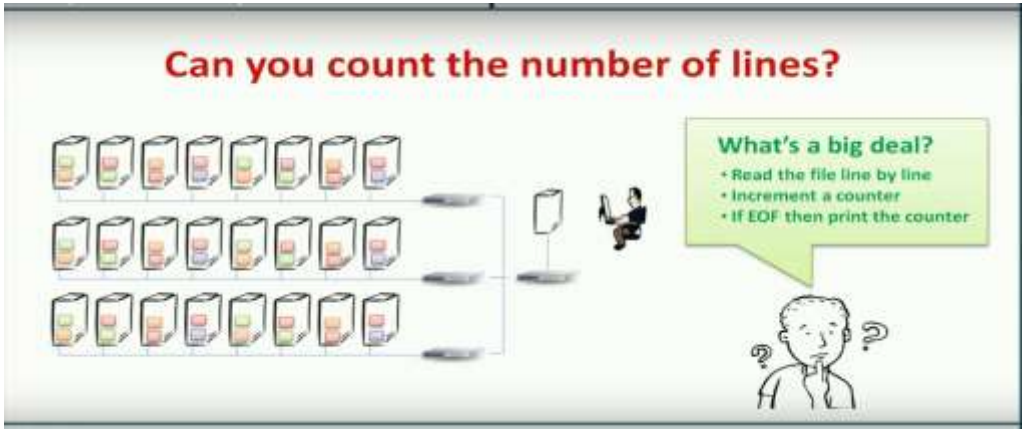


Lets us take a file of 20TB. We can't store the file in single machine .

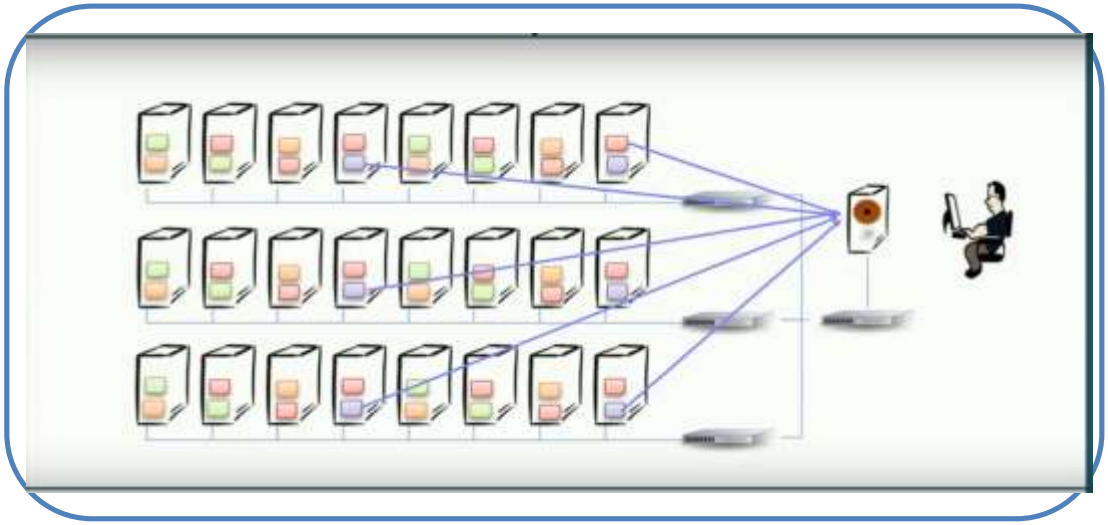
So we decided to store it in Hadoop Cluster



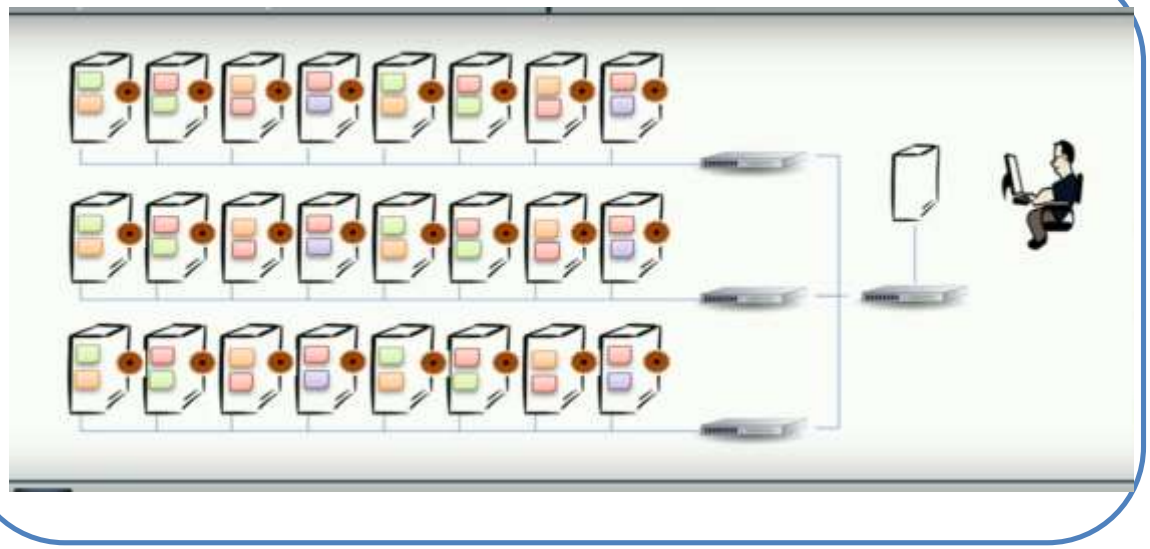
Divide the file into blocks . The Default Block size is 128 MB. Then store it in HDFS



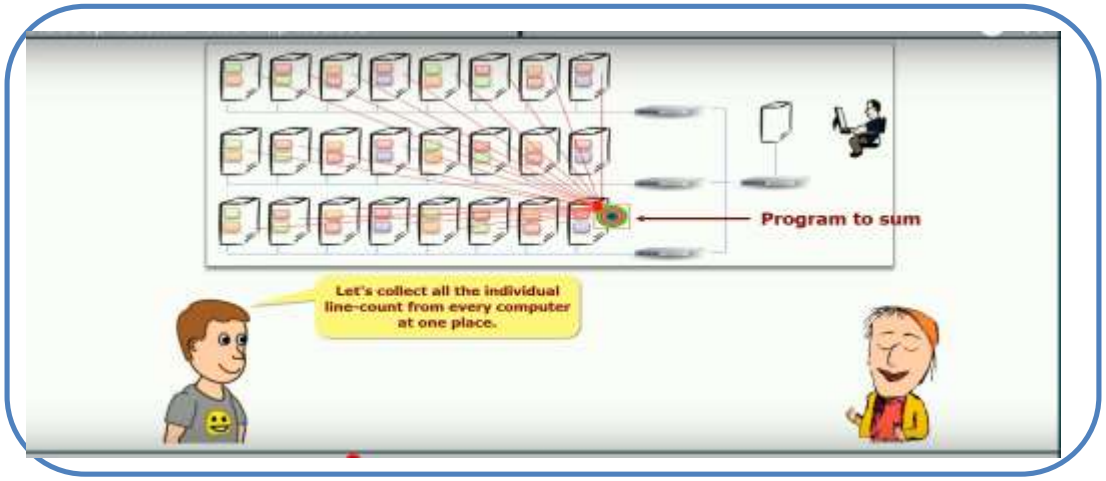
One way to count number of lines is to read a line and then increment the counter . Do the process until the EOF is Reached.



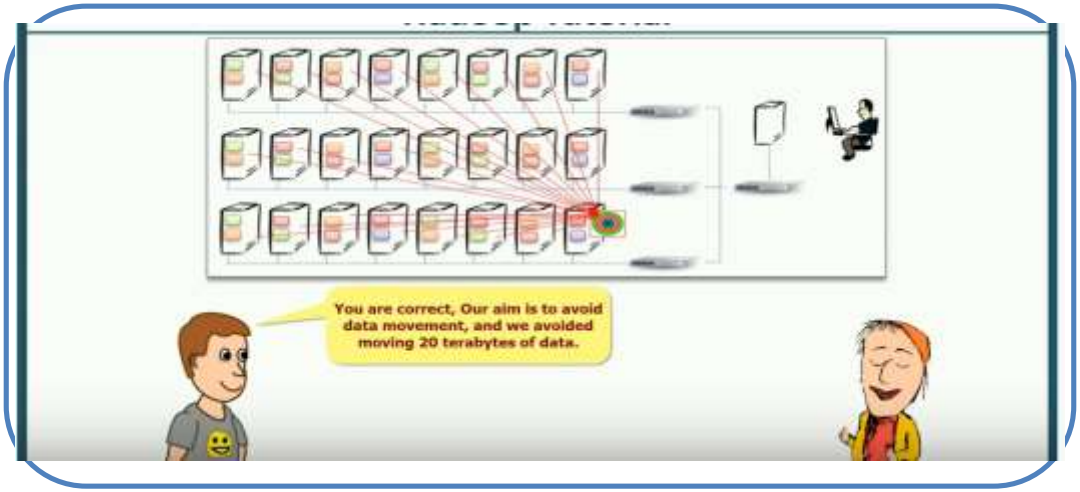
Bring all the Data to a single machine and execute the Java Program that counts the number of lines. This Process takes more Band width and more processing time .



Instead of Bringing data to processing , send processing to data and that is parallel processing



Collect all the Individual Line-count from every computer at one Place



Our aim is to avoid Data Movement and has avoided moving 20 terabytes of Data

6.2 UNDERSTANDING MAP REDUCE

What is Mapreduce

- **MapReduce** is a programming model used for processing huge amounts of data.
- **MapReduce** program work in two phases, namely, Map and Reduce.
  - ✓ Map tasks deal with splitting and mapping of data
  - ✓ while Reduce tasks shuffle and reduce the data.

6.2 UNDERSTANDING MAP REDUCE

What is Mapreduce

- ✓ The MapReduce program involves two phases, namely:

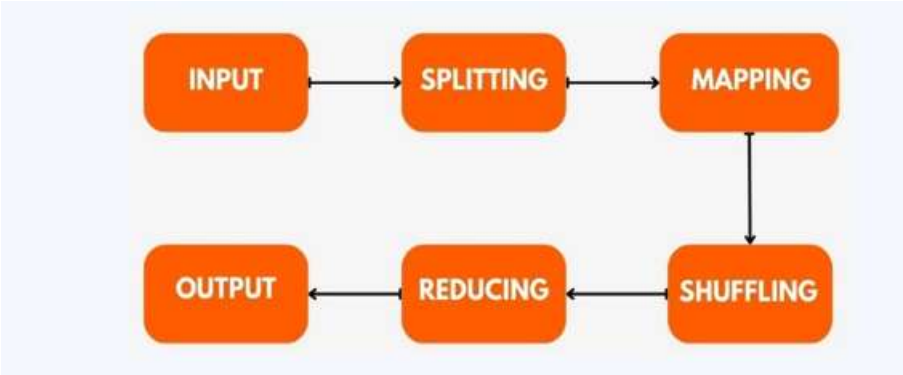
  Map and Reduce.

- ✓ These are sometimes also referred to as **Mapper and Reducer Phase**.
- ✓ Input to each phase is given **as key-value pairs.**
- ✓ ***The Mapper phase splits the data into chunks and processes these chunks parallelly.***
- ✓ ***The output of the mapper phase is fed as an input for the reducer phase. Reducer shuffles and reduces the data.***
- ✓ ***The output of the reducer phase is the final output.***

6.2 UNDERSTANDING MAP REDUCE

How MapReduce Works

The entire process is divided into four steps of execution which are splitting, mapping, shuffling and reducing.

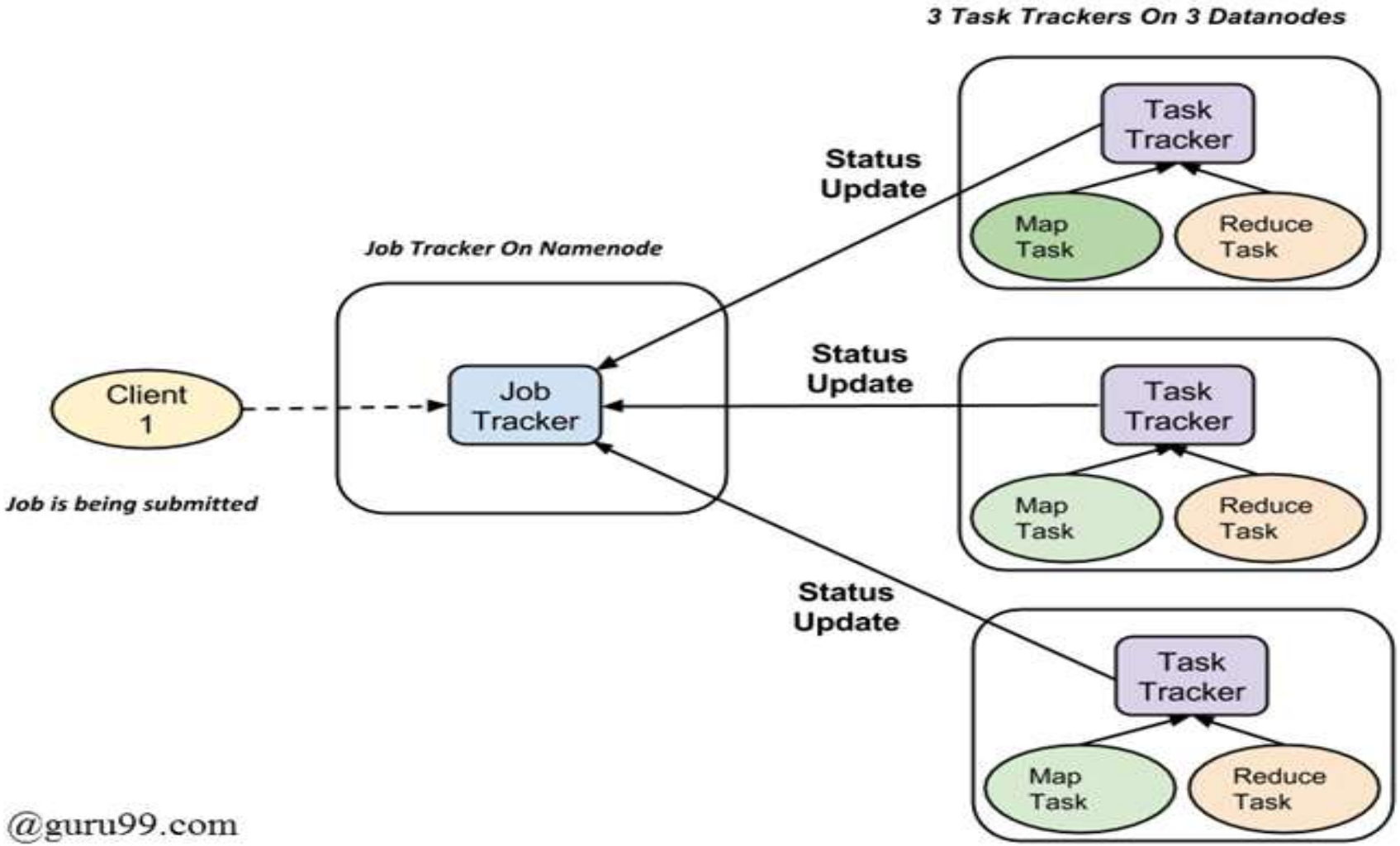


6.2 UNDERSTANDING MAP REDUCE

MapReduce Architecture

- MapReduce Architecture is a parallel processing framework for efficiently analyzing and processing large datasets across distributed computing clusters.
- The entire job is divided into tasks.
- There are two types of tasks namely, Mapping tasks and Reducing tasks.
- Mapping tasks splits the input data and performs mapping while reducing tasks performs shuffling and aggregates the shuffling values and returns a single output value, thereby reducing the data.
- The execution of these two tasks is controlled by two entities:
  1. **Job Tracker -** It acts like a master and plays the role of scheduling jobs and tracking the jobs assigned to Task Tracker.
  2. **Multiple Task Tracker -** It acts like slaves. It tracks the jobs and reports the status of the jobs to the master (job tracker)

6.2 UNDERSTANDING MAP REDUCE



6.2 UNDERSTANDING MAP REDUCE

Working of Job and Task Trackers

Below points shows the working of job and task trackers.

1. **Each job is divided into multiple tasks, which are executed on multiple data nodes in a cluster.**
2. **The Job tracker schedules the tasks to run on different data nodes.**
3. **The task tracker executes the task assigned to it by the job tracker. Task trackers are present on every data node executing a part of the job.**
4. Task Tracker is also responsible for sending status updates to the job tracker.
5. **Task Tracker periodically sends heartbeat signals to the Job tracker to notify it about the system's current state.**
6. Hence, the Job tracker tracks the progress of each job with the help of task trackers. In case of task failure, the job tracker reschedules the task to a different tasktracker.

6.2 UNDERSTANDING MAP REDUCE

Working of Job and Task Trackers

Below points shows the working of job and task trackers.

1. **Each job is divided into multiple tasks, which are executed on multiple data nodes in a cluster.**
2. **The Job tracker schedules the tasks to run on different data nodes.**
3. **The task tracker executes the task assigned to it by the job tracker. Task trackers are present on every data node executing a part of the job.**
4. Task Tracker is also responsible for sending status updates to the job tracker.
5. **Task Tracker periodically sends heartbeat signals to the Job tracker to notify it about the system's current state.**
6. Hence, the Job tracker tracks the progress of each job with the help of task trackers. In case of task failure, the job tracker reschedules the task to a different tasktracker.

6.2 UNDERSTANDING MAP REDUCE

Phases of MapReduce

- There are three main phases involved : **Mapping phase, Shuffling and Sorting phase, Reducing Phase.**
- Consider the example where we feed the below input data to MapReduce
- **Welcome Ninja Become a Coding Ninja Ninja Coder**

Mapping Phase

- **This is the first phase which involves splitting and mapping.**
- The input data set is split into equal units called input splits or chunks.
- **These input splits are then passed to a mapping function to generate output key-value pairs.**
- In our example, the job of the mapping phase is to count the frequency of each word from the input splits.

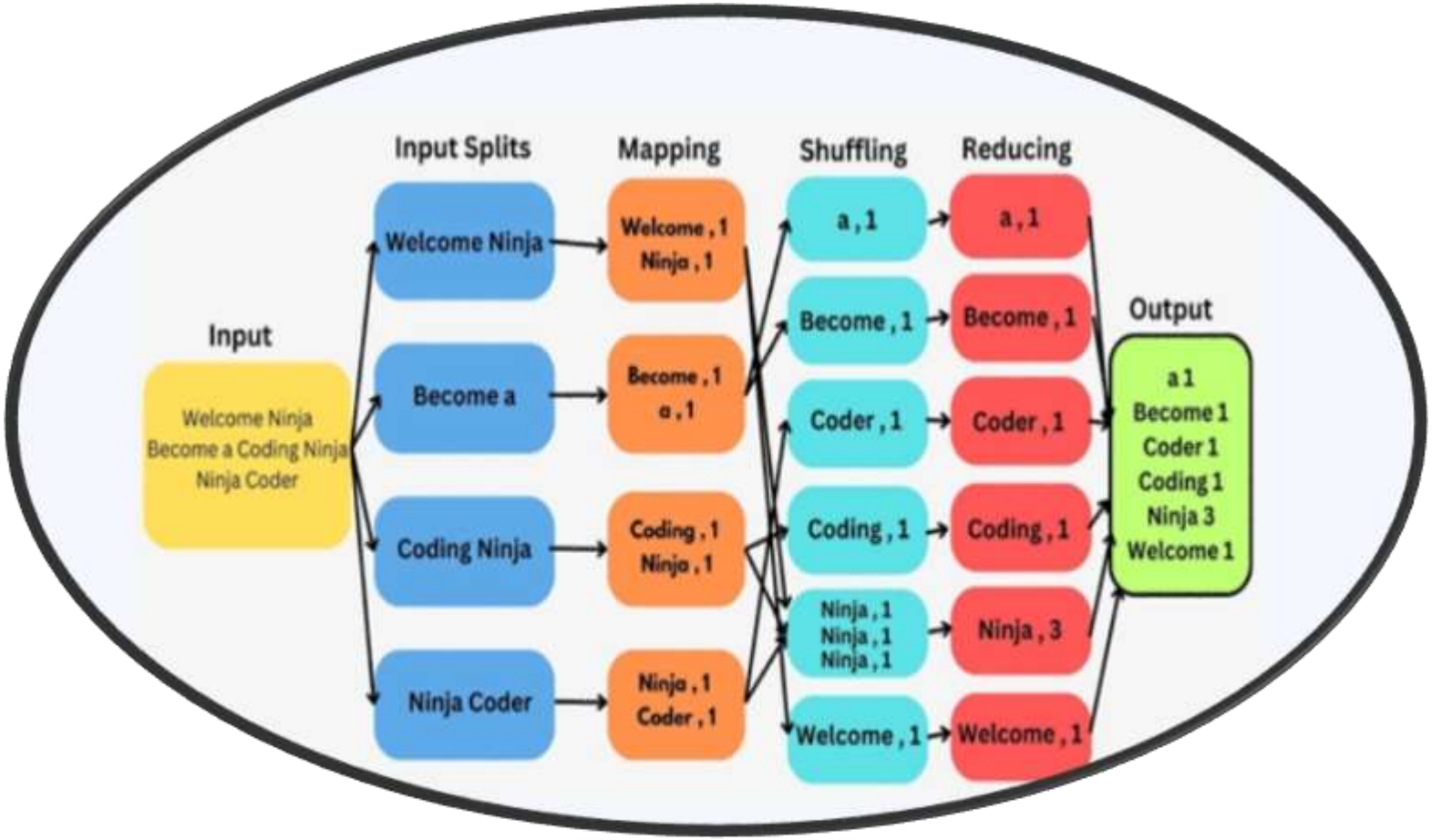
Shuffling and Sorting Phase

- **This is the second phase which takes place after the completion of the mapping phase.**
- The two primary functions of this phase are sorting and merging.
- **The key-value pairs received as an output from the Mapping Phase are sorted using the keys.**
- These sorted key-value pairs are then merged in the merging step.
- **Shuffling helps remove duplicate values and even helps in grouping them.**

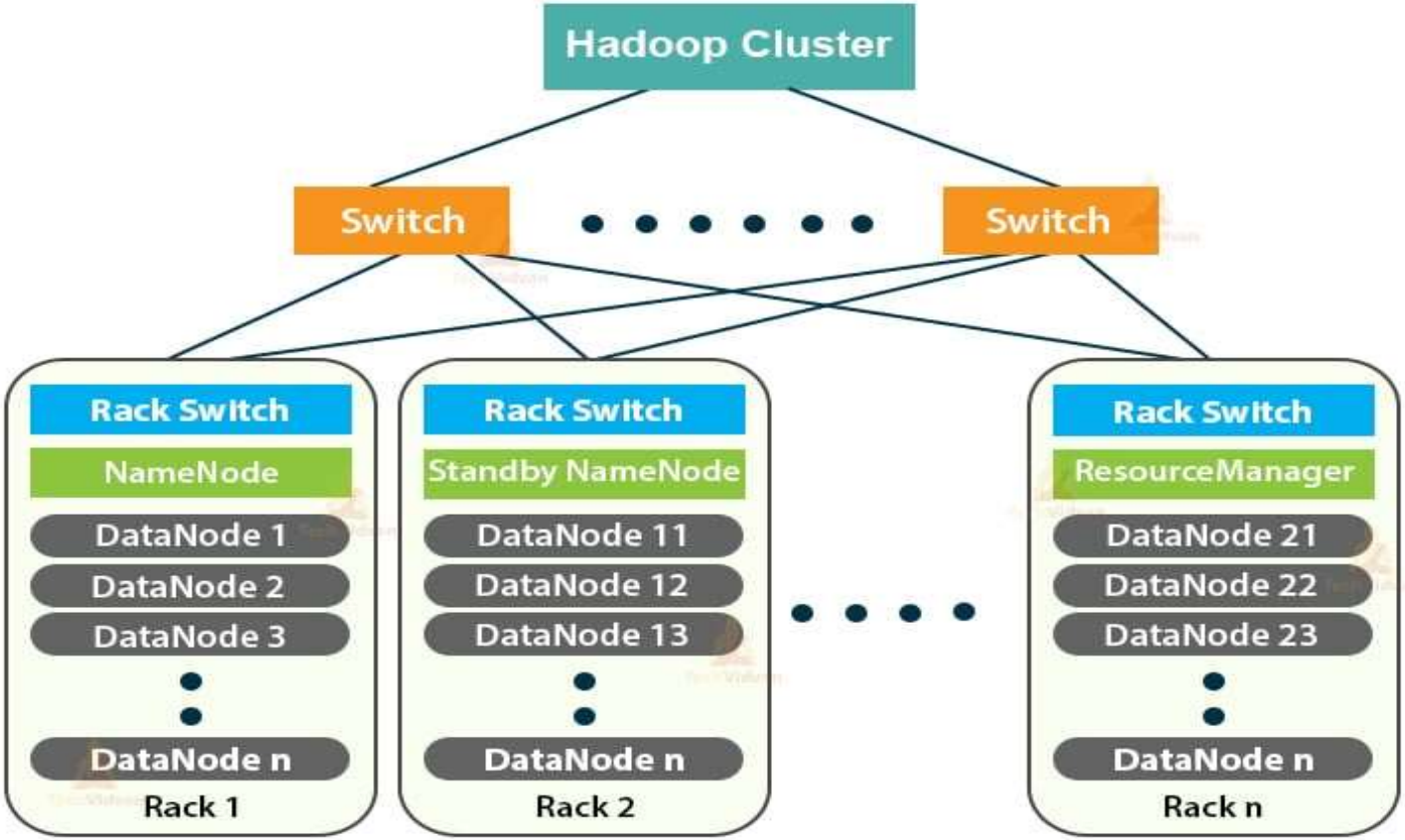
Reducing Phase

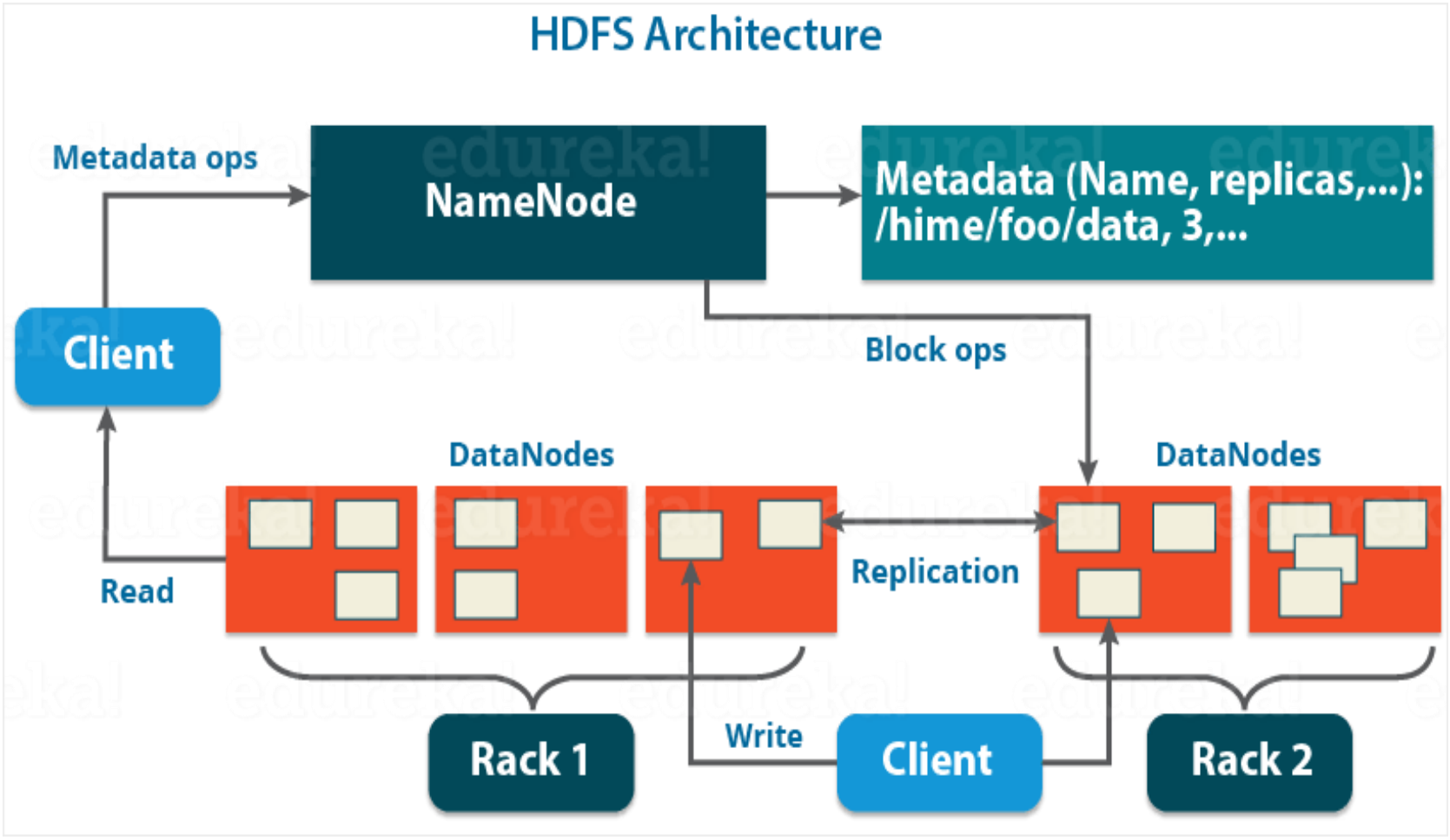
- **The output from the shuffling phase is used as an input for the Reducing Phase.**
- It aggregates/combines the shuffling values and gives a single output value.
- **The reducer processes the input by reducing the intermediate values into smaller values.**

6.2 UNDERSTANDING MAP REDUCE



HDFS Architecture




HDFS Architecture
Metadata ops
NameNode
Metadata (Name, replicas,...):
/hime/foo/data, 3,...
Client
Block ops
DataNodes
DataNodes
Read
Replication
Rack 1
Write
Client
Rack 2

HDFS Services

1. Name Node
2. Secondary Name Node
3. Job tracker
4. Data Node
5. Task Tracker

NameNode

- Metadata in Memory
  - The entire metadata is in main memory
- Types of metadata
  - List of files
  - List of Blocks for each file
  - List of DataNodes for each block
  - File attributes, e.g. creation time, replication factor
- A Transaction Log
  - Records file creations, file deletions etc

DataNode

- DataNodes are the main storages of data. Hadoop uses Low-Cost hardware to Store data.
- DataNodes are responsible for Storing, Replication Creating, Deleting these type of jobs according to the instruction of NameNode.
- These DataNodes send the health report to the NameNode periodically.
- The defalult time is 3second. So after every second these send the report to the NameNode.

Secondary NameNode

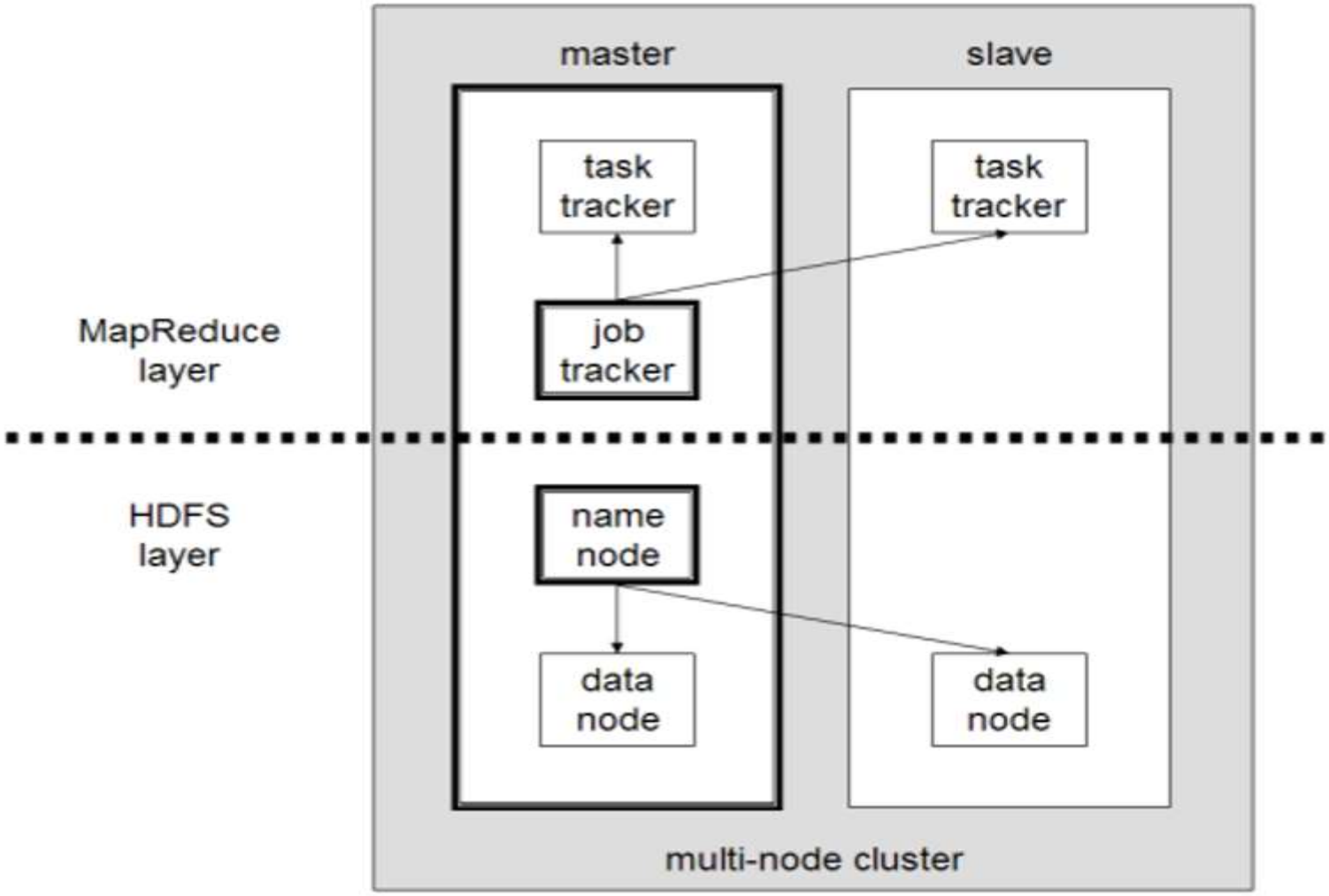
- SNN is another specially dedicated node, which is used to take the checkpoints of the File-System.
- The SNN is not substitute of the primary NameNode.

  It helps the NameNode but not replace of NameNode.
- Copies FsImage and Transaction Log from Namenode to a temporary directory
- Merges FSImage and Transaction Log into a new FSImage in temporary directory
- Uploads new FSImage to the NameNode
  - Fsimage is contains Dircetory Structure (NameSpace)

Job Tracker and Task Tracker

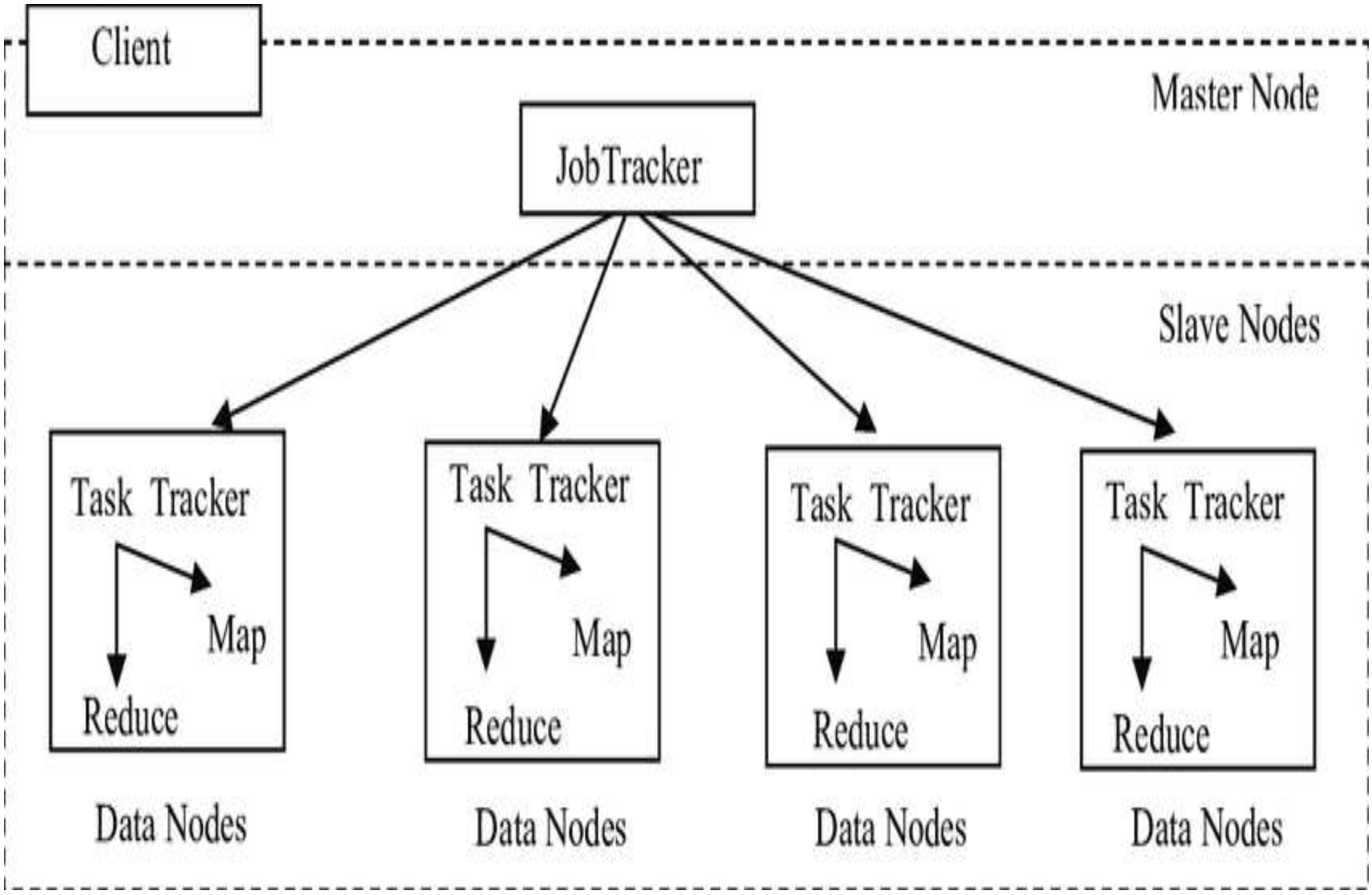
Job Tracker:

- Basically Job Tracker will be useful in the Processing the data.
- Job Tracker receives the requests from the client for MapReduce execution.
- Job Tracker talks with Name node to know about the location of the data like Job Tracker will request the Name Node for the processing the data.
- Name node in response gives the Metadata to job tracker.


master
slave
task tracker
task tracker
job tracker
MapReduce layer
HDFS layer
name node
data node
data node
multi-node cluster

Task Tracker

- It is the Slave Node for the Job Tracker and it will take the task from the Job Tracker.
- Task Tracker receives code from the Job Tracker and apply the code on the file for the process is known as Mapper.


Client
Master Node
JobTracker
Slave Nodes
Task Tracker
Map
Reduce
Data Nodes
Task Tracker
Map
Reduce
Data Nodes
Task Tracker
Map
Reduce
Data Nodes
Task Tracker
Map
Reduce
Data Nodes

Case Study: Weather Data Analysis Using Hadoop

Problem Statement

- Huge volume of weather data generated daily.
- Need for faster weather forecasting.
- Efficient storage of historical weather records.
- Real-time analysis for disaster management.

Unit Highlights

- Hadoop is an open-source Big Data framework.
- HDFS provides distributed and fault-tolerant storage.
- MapReduce enables parallel data processing.
- Hadoop supports Local, Pseudo-distributed, and Fully Distributed modes.
- The Hadoop ecosystem includes Hive, Pig, HBase, Sqoop, Flume, Oozie, ZooKeeper, and Spark.
- Weather datasets are a common example for demonstrating Big Data processing.
- HDFS and MapReduce architectures illustrate how Hadoop stores and processes large-scale data efficiently.
- Hadoop is widely used in industries such as weather forecasting, healthcare, finance, e-commerce, and social media.

TEXT BOOKS:

- Big Data Fundamentals: Concepts, Drivers & Techniques", 1/e, 2016, Thomas Erl, Wajid Khattak, Paul Buhler, Prentice Hall.
- "Hadoop:The Definitive Guide," 3/e, 2012, Tom White, O'REILLY Publications.
- "Learning Spark"

REFERENCE BOOKS:

- Taming the Big Data Tidal Wave: Finding Opportunities in Huge Data Streams with Advanced Analytics, 2012, Bill Franks, John Wiley & Sons..
- Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data, 2012, Paul C. Zikopoulos, Chris Eaton, Dirk deRoos, Thomas Deutsch, George Lapis, McGraw-Hill.
- Intelligent Data Analysis, 2007, Michael Berthhold, David J.Hand, Springer.
- Understanding Big Data: Analytics for Enterprise Class Hadoop and streaming data, 2011, Paul Zikopoulos, Chris Eaton, Paul Zikopoulos, McGraw hill.
- Big Data for Dummies, 2012, Hurwitz, Alan Nugent, Dr. Fern Halper, Marcia Kaufman,John Wiley & Sons