Big Data Analytics

IT-411

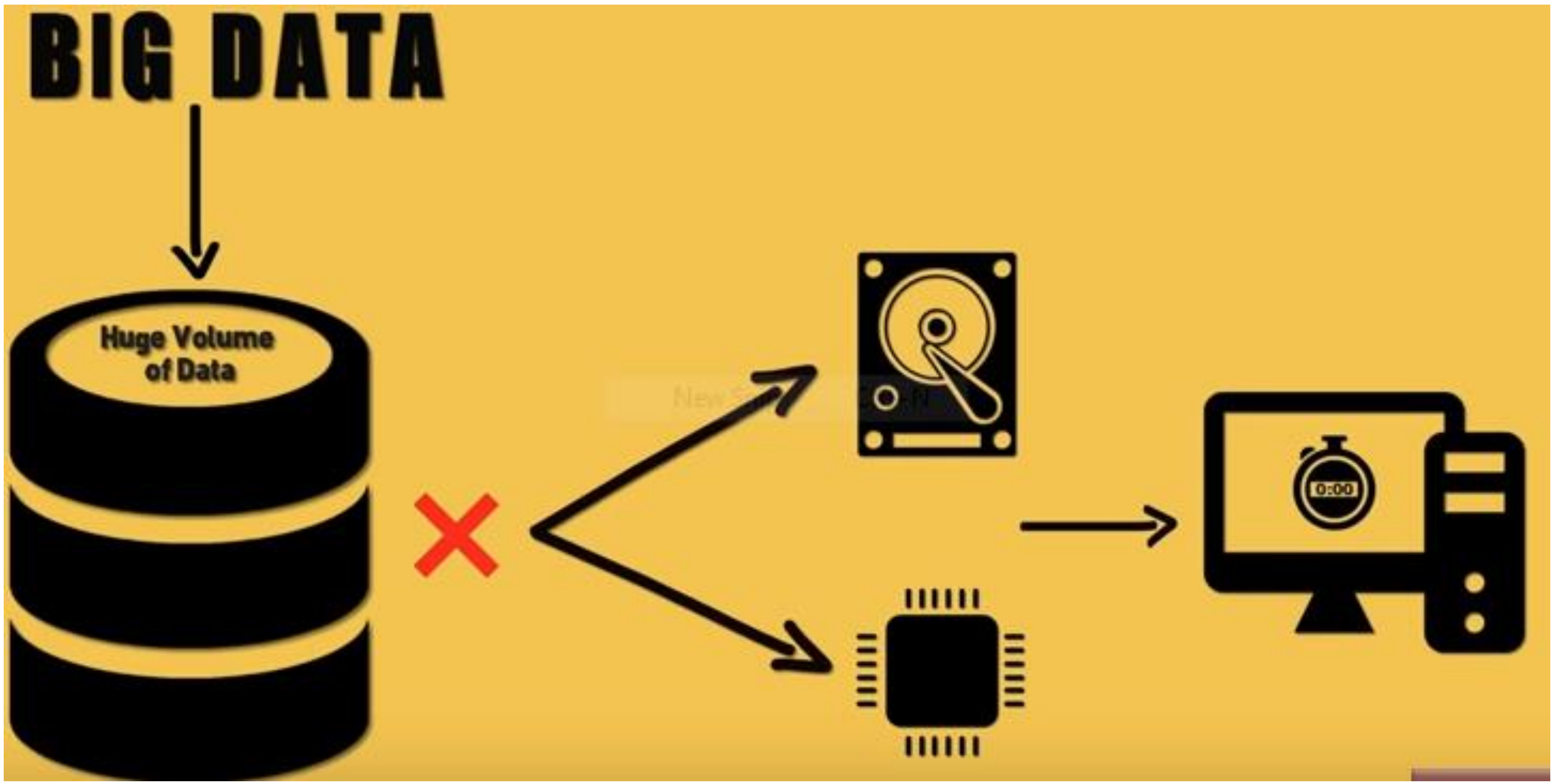
Kratika Sharma, Asst. Professor, IT Dept

Megabytes, Gigabytes, Terabytes… What Are They ?

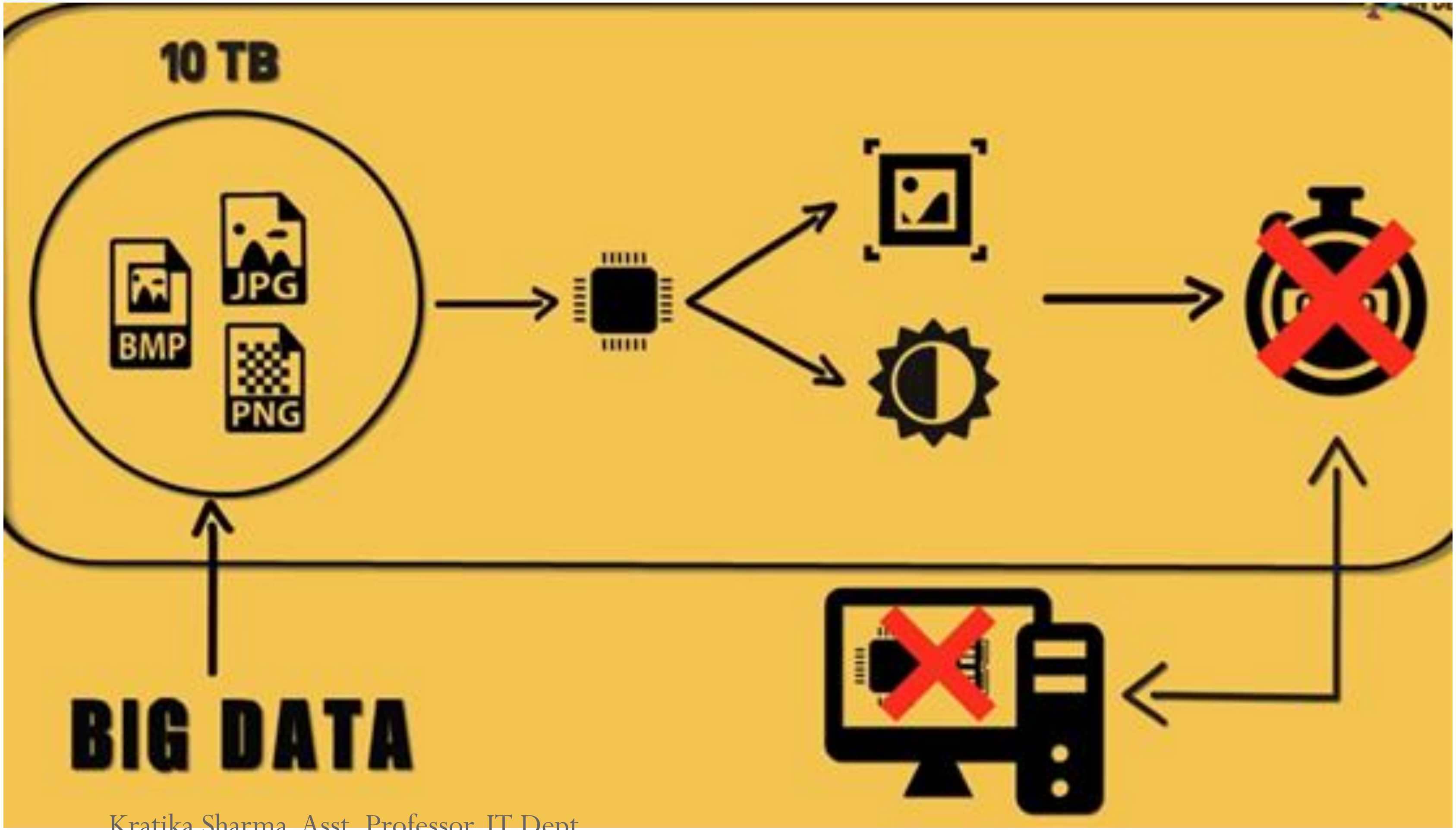
Bit = Binary Digit

- 8 Bits = 1 Byte
- 1024 Bytes = 1 Kilobyte
- 1024 Kilobytes = 1 Megabyte
- 1024 Megabytes = 1 Gigabyte
- 1024 Gigabytes = 1 Terabyte
- 1024 Terabytes = 1 Petabyte
- 1024 Petabytes = 1 Exabyte
- 1024 Exabytes = 1 Zettabyte
- 1024 Zettabytes = 1 Yottabyte
- 1024 Yottabytes = 1 Brontobyte
- 1024 Brontobytes = 1 Geopbyte

What is Big Data?



How Huge this data needs to be?




BIG DATA
You
Tube
$$

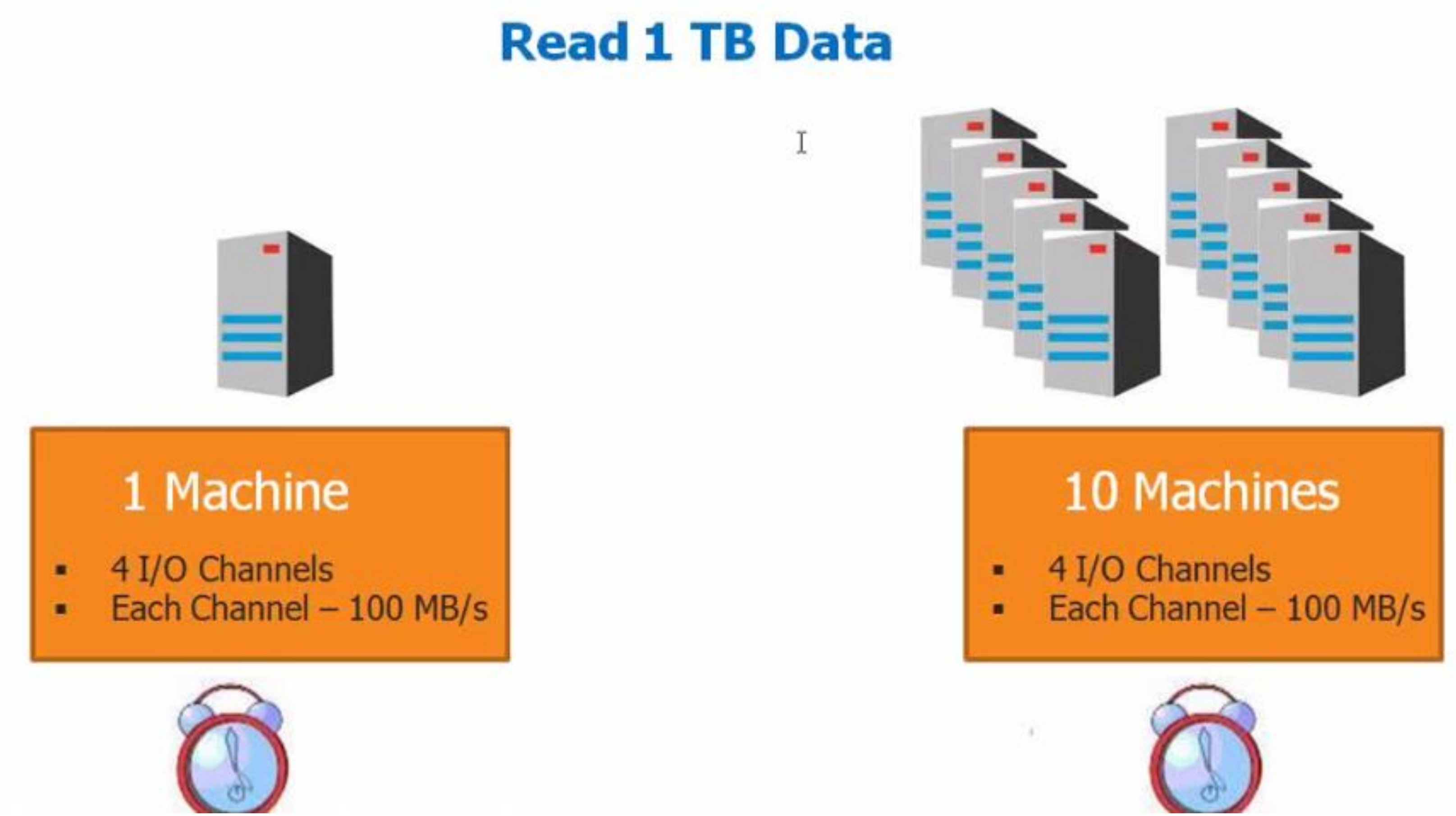
What is Big Data?

Classification of Big Data

- Structured

- Unstructured Data

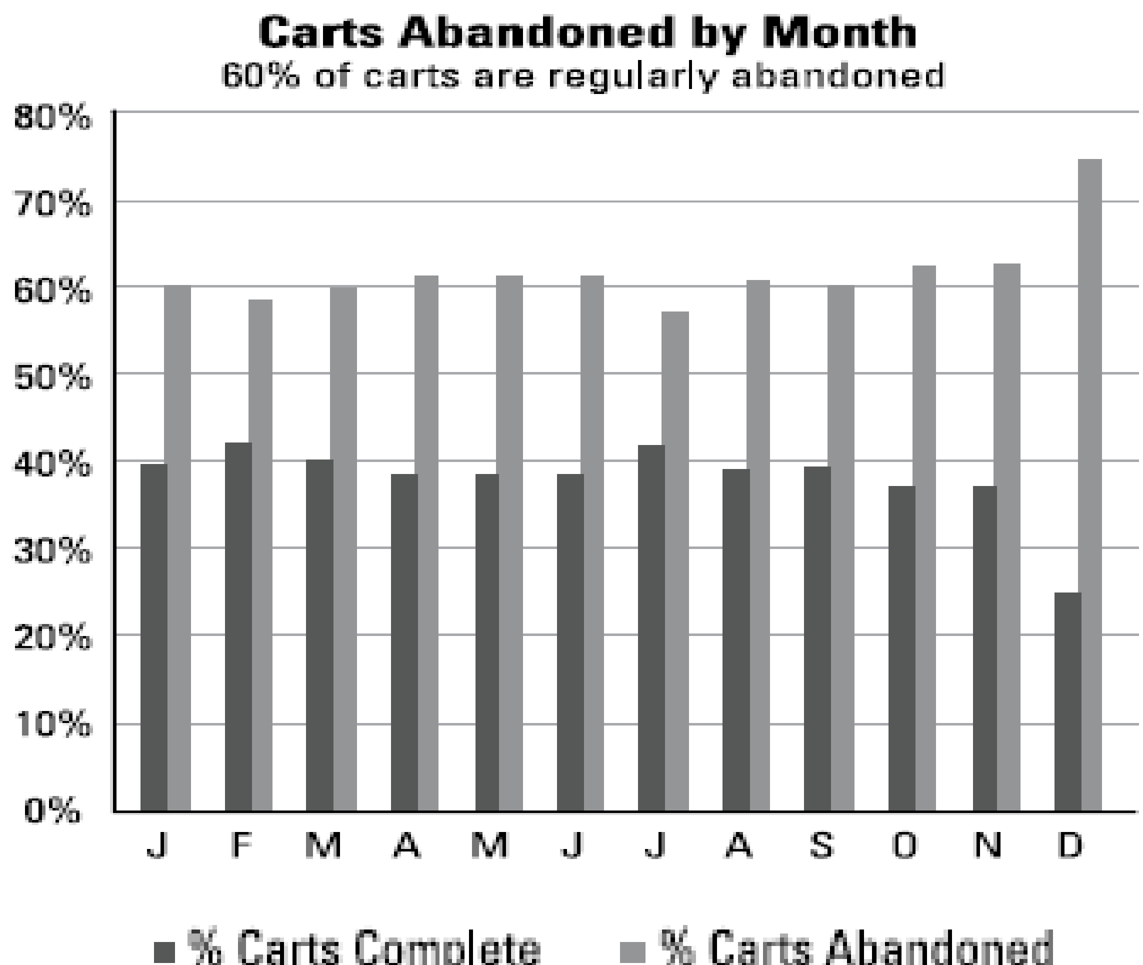
Why DFS??

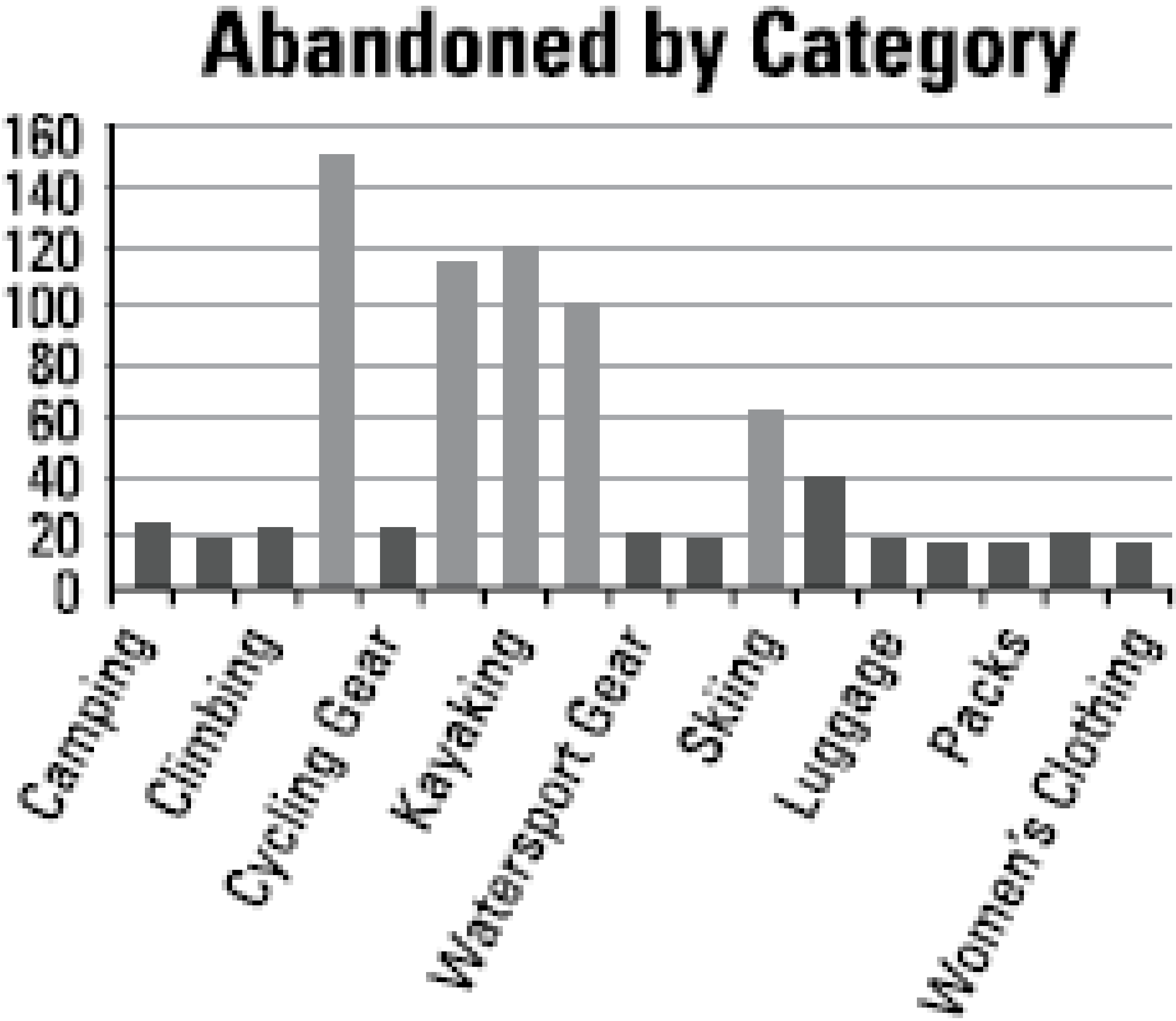

Read 1 TB Data
1 Machine
▪ 4 I/O Channels
▪ Each Channel – 100 MB/s
10 Machines
▪ 4 I/O Channels
▪ Each Channel – 100 MB/s

Big Data Use Cases

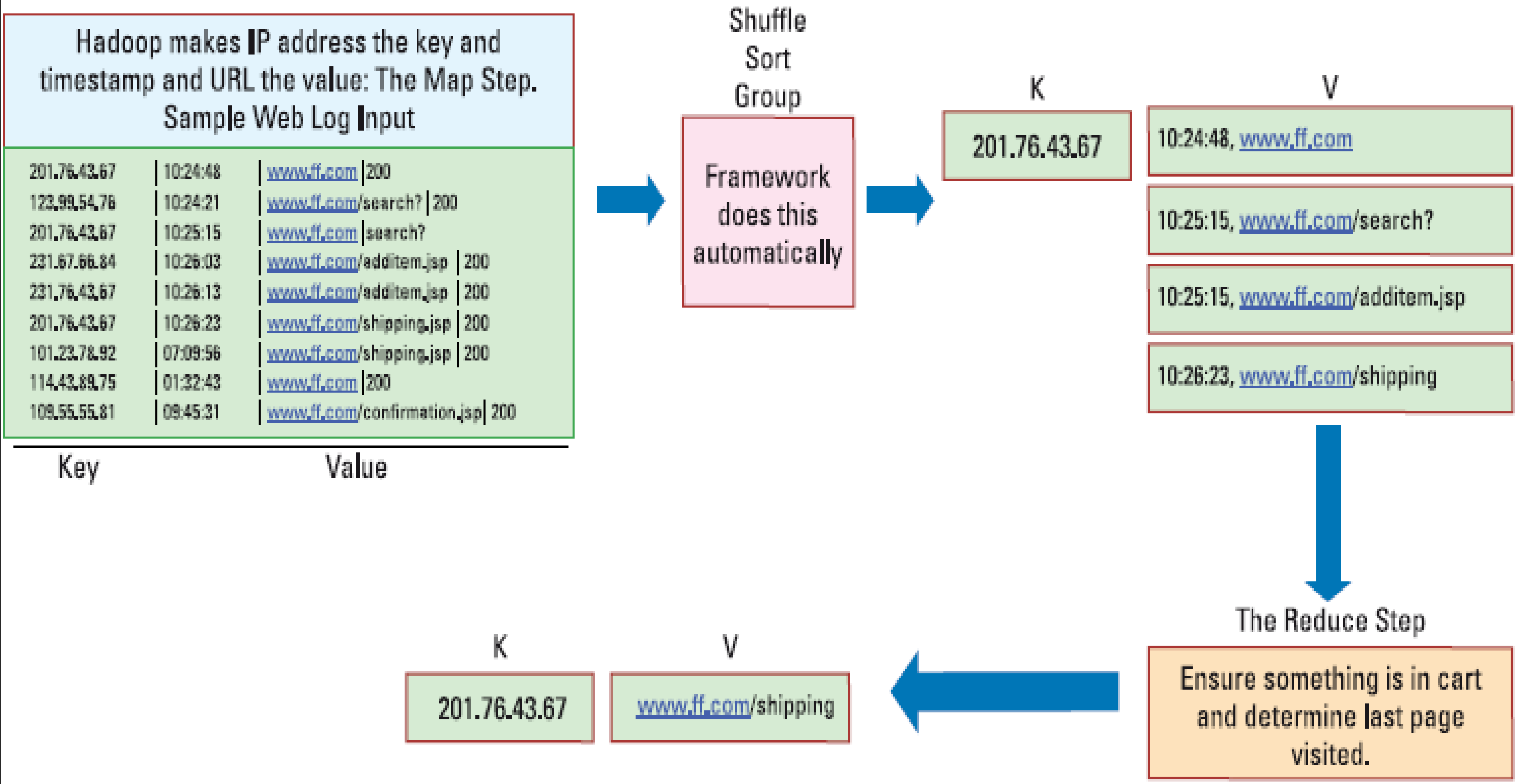
- **Log Data Analysis**
- **Fraud Detection**
- **Social sentiment analysis**

Log Data Analysis




Abandoned by Category
160
140
120
100
80
60
40
20
0
Camping
Climbing
Cycling Gear
Kayaking
Watersport Gear
Skiing
Luggage
Packs
Women's Clothing

- “Are certain products abandoned more than others?
- How much revenue can be recaptured if you decrease cart abandonment by 10 percent?”


Hadoop makes IP address the key and timestamp and URL the value: The Map Step.
Sample Web Log Input
201.76.43.67 | 10:24:48 | www.ff.com | 200
123.99.54.76 | 10:24:21 | www.ff.com/search? | 200
201.76.43.67 | 10:25:15 | www.ff.com | search?
231.67.66.84 | 10:26:03 | www.ff.com/additem.jsp | 200
231.76.43.67 | 10:26:13 | www.ff.com/additem.jsp | 200
201.76.43.67 | 10:26:23 | www.ff.com/shipping.jsp | 200
101.23.78.92 | 07:09:56 | www.ff.com/shipping.jsp | 200
114.43.89.75 | 01:32:43 | www.ff.com | 200
109.55.55.81 | 09:45:31 | www.ff.com/confirmation.jsp | 200
Key
Value
Shuffle
Sort
Group
Framework does this automatically
K
V
201.76.43.67
10:24:48, www.ff.com
10:25:15, www.ff.com/search?
10:25:15, www.ff.com/additem.jsp
10:26:23, www.ff.com/shipping
The Reduce Step
Ensure something is in cart and determine last page visited.
K
V
201.76.43.67
www.ff.com/shipping

- The final page that's visited
- A list of items in the shopping cart
- The state of the transaction for each user session (indexed by the IP address key)

Fraud Detection

- Hadoop-anchored fraud department that uses the full data set — no sampling — to build out the models, you can see the difference.

- For creating fraud-detection models, Hadoop is well suited **to**

✓ **Handle volume:That means processing the full data set — no data sampling.**

✓ **Manage new varieties of data: Examples are the inclusion of proximity**

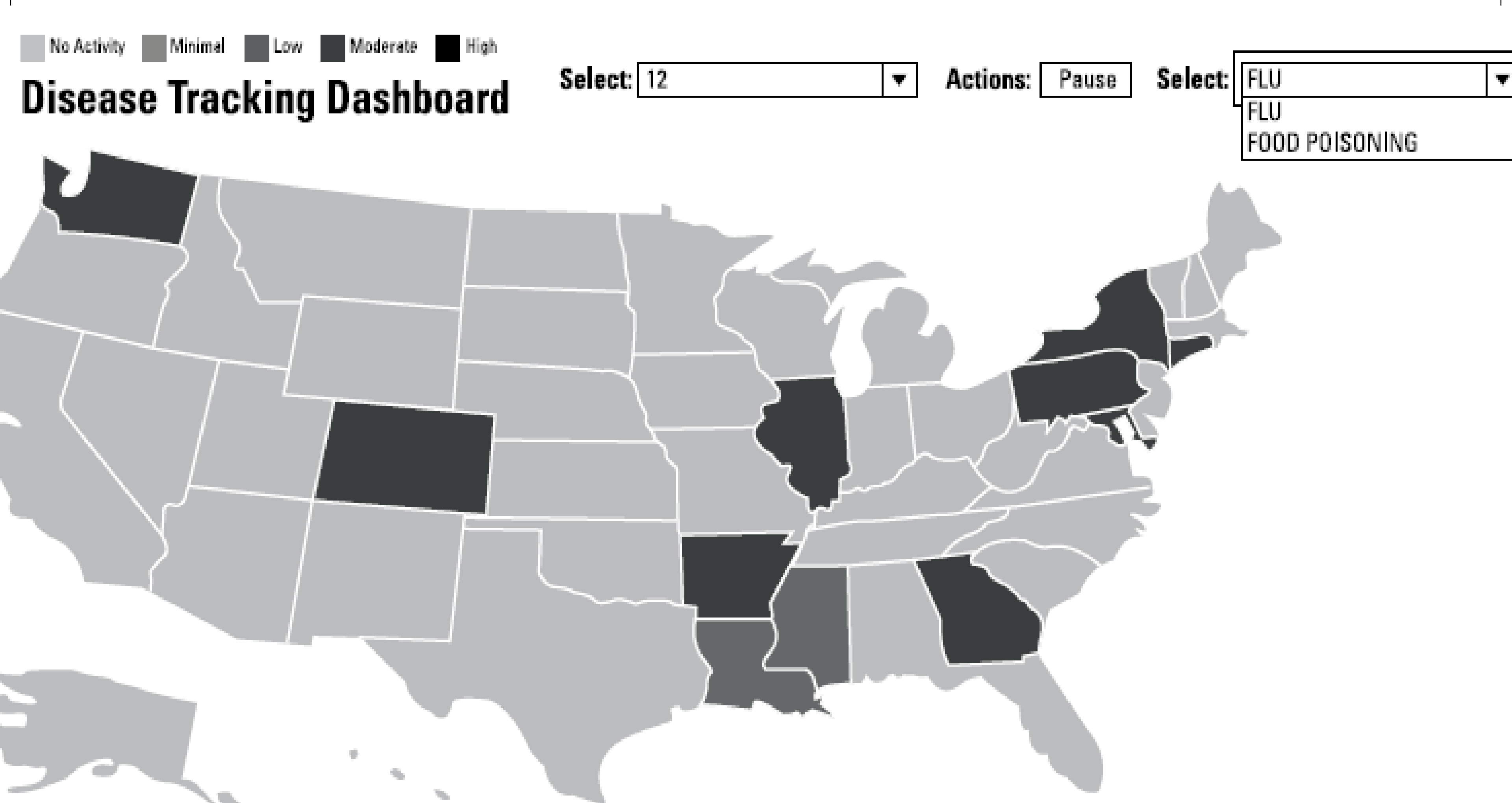
✓ **Maintain an agile environmto-care-services and social circles to decorate the fraud model.ent: Enable different kinds of analysis and changes to existing models.**

Social Sentiment Analysis

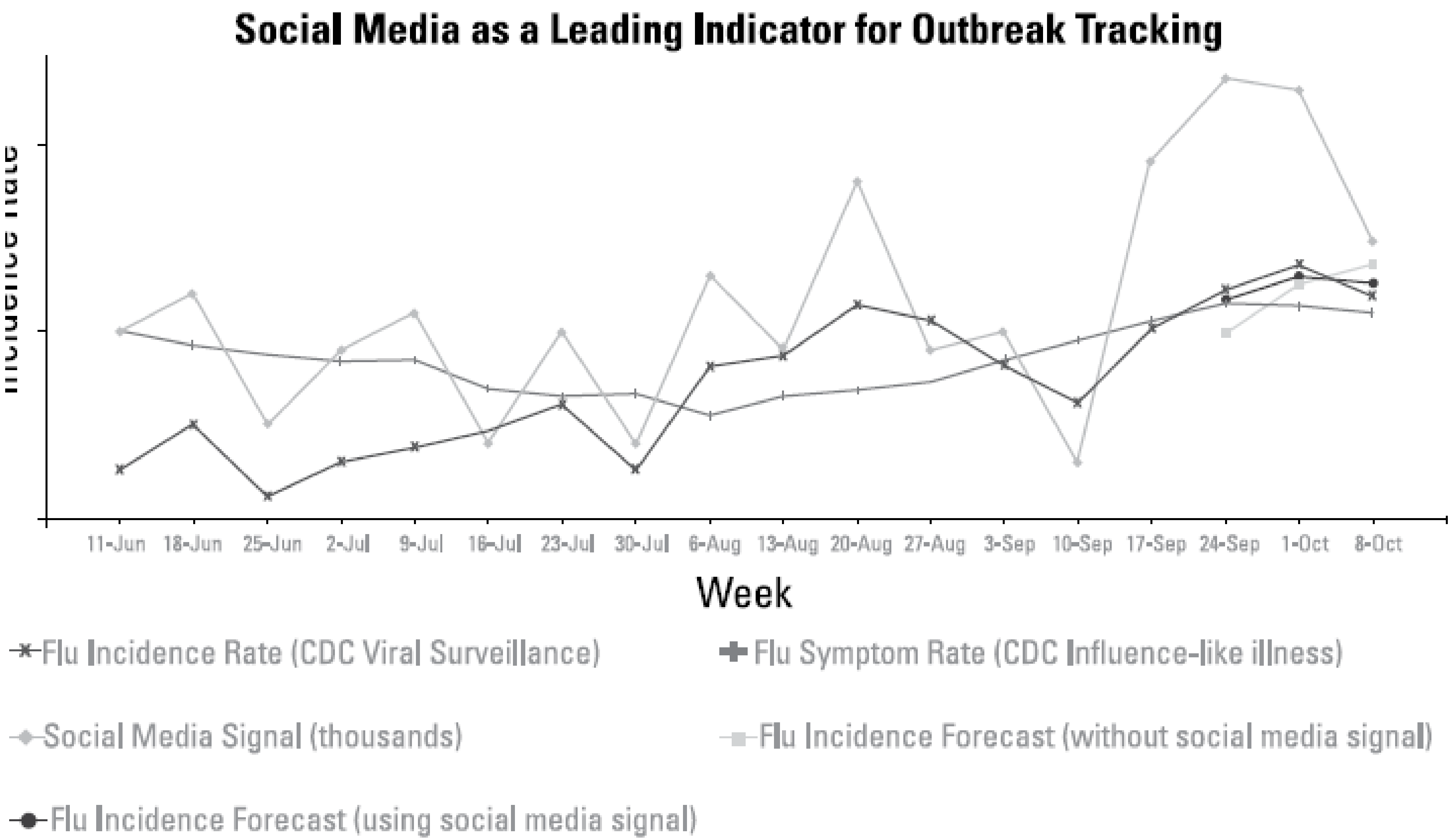
- Language is difficult to interpret, even for human beings at times.

- **Social sentiment analysis is in reality, text analysis**

Tweets
Sore throat has developed into full blown flu!
Though it was green apple two step - it's flu - both ends
Hands up for those who caught the flu bug as well
I think I got flu hate b in sick
I can feel flu coming on...hey that reminds me of a beer ad
awful headache, shocking weather, flu, could today get worse #complaintover
Never want to get stomach flu again!! Dad next door is feeling worse than me
Thera flu for you
Morning peeps big day ahead back to work had 4 days off man flu

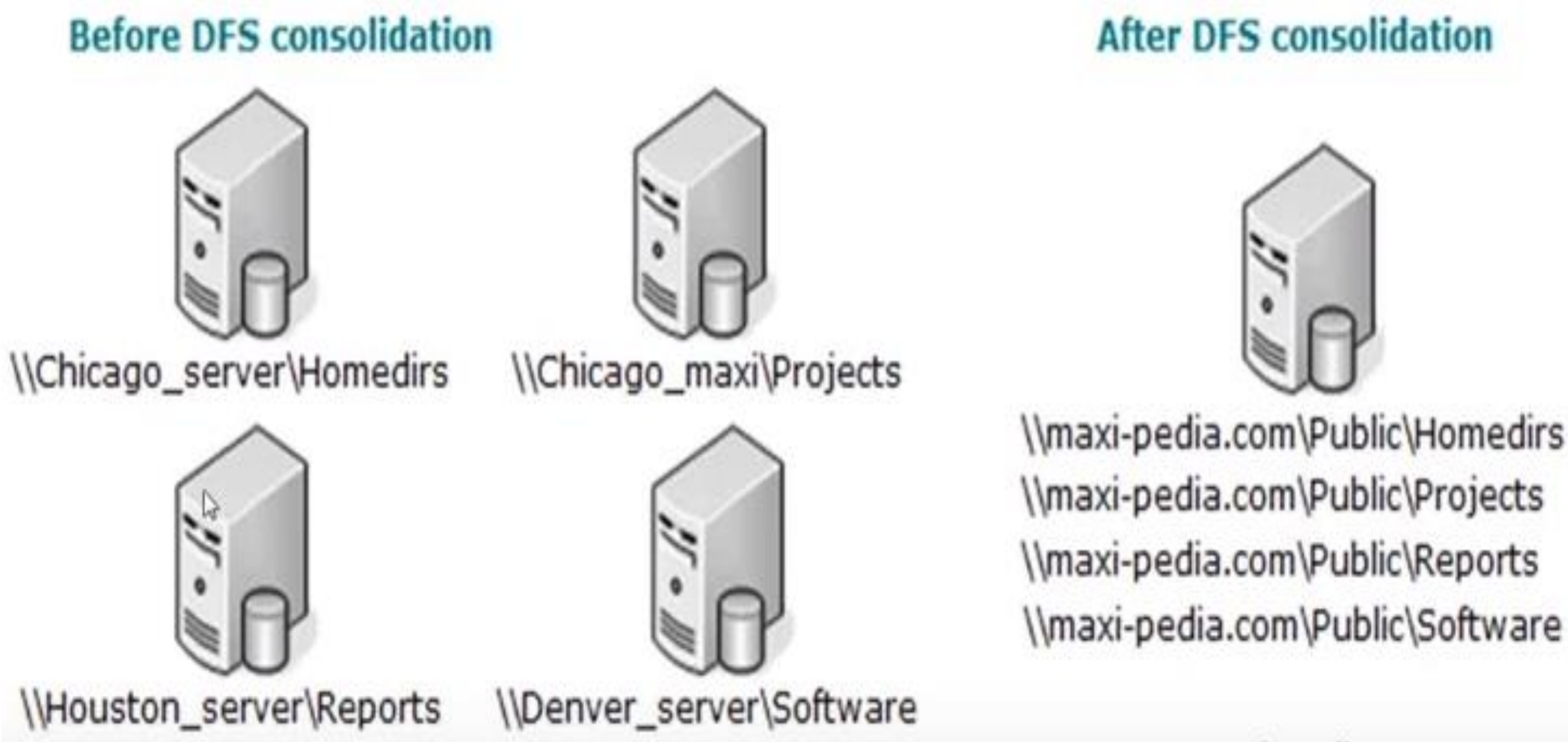

No Activity
Minimal
Low
Moderate
High
Disease Tracking Dashboard
Select: 12
Actions: Pause
Select: FLU
FLU
FOOD POISONING

- How good of a prediction engine can social media be for the outbreak of the flu or a food poisoning incident?



Social Media as a Leading Indicator for Outbreak Tracking
11-Jun
18-Jun
25-Jun
2-Jul
9-Jul
16-Jul
23-Jul
30-Jul
6-Aug
13-Aug
20-Aug
27-Aug
3-Sep
10-Sep
17-Sep
24-Sep
1-Oct
8-Oct
Week
Flu Incidence Rate (CDC Viral Surveillance)
Flu Symptom Rate (CDC Influence-like illness)
Social Media Signal (thousands)
Flu Incidence Forecast (without social media signal)
Flu Incidence Forecast (using social media signal)

What is Distributed File System



What is Hadoop?

Apache Hadoop is a framework that allows for the distributed processing of large data sets across clusters of commodity computers using a simple programming model.

Companies using Hadoop:

- Yahoo
- Google
- Facebook
- Amazon
- Ibm and many more..

Core Components of Hadoop

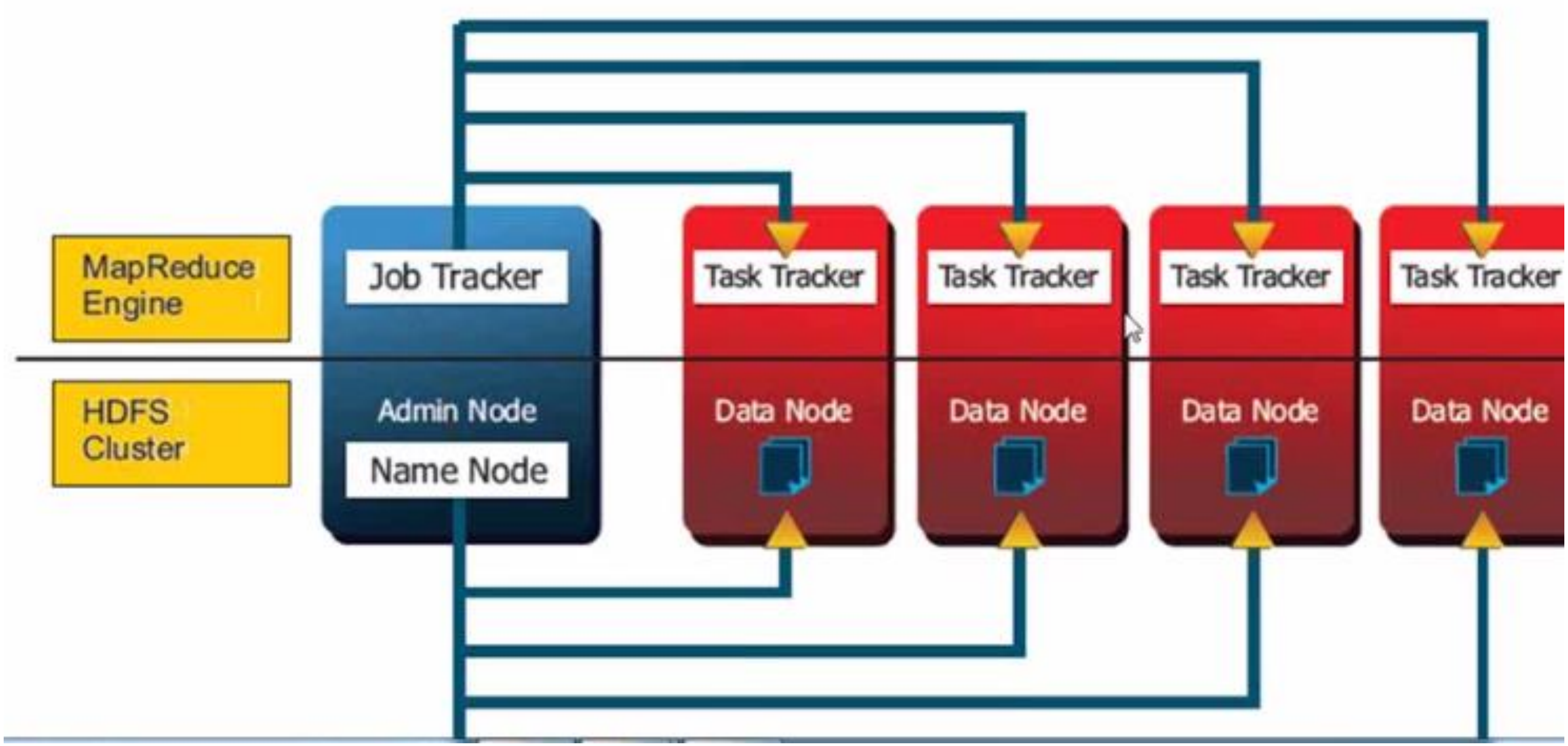
Hadoop Distributed File System (storage)

MapReduce (Processing)

Hadoop Services or architectural Components

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

Hadoop Core Components



HDFS Features:

- High Fault Tolerant
- High Throughput
- Suitable for applications with large data sets
- Streaming access to file system data
- Can be built out of commodity hardware

The Design of HDFS

HDFS is a file system designed for storing very large files with streaming data access patterns, running clusters on commodity hardware.

Areas where HDFS is not a good fit

- Low-latency data access
- Lots of small files
- Multiple writers, arbitrary file modifications

HDFS Components

- Name Node
- Data Node

Main Components of HDFS

- Name Node

➢ Master of the system

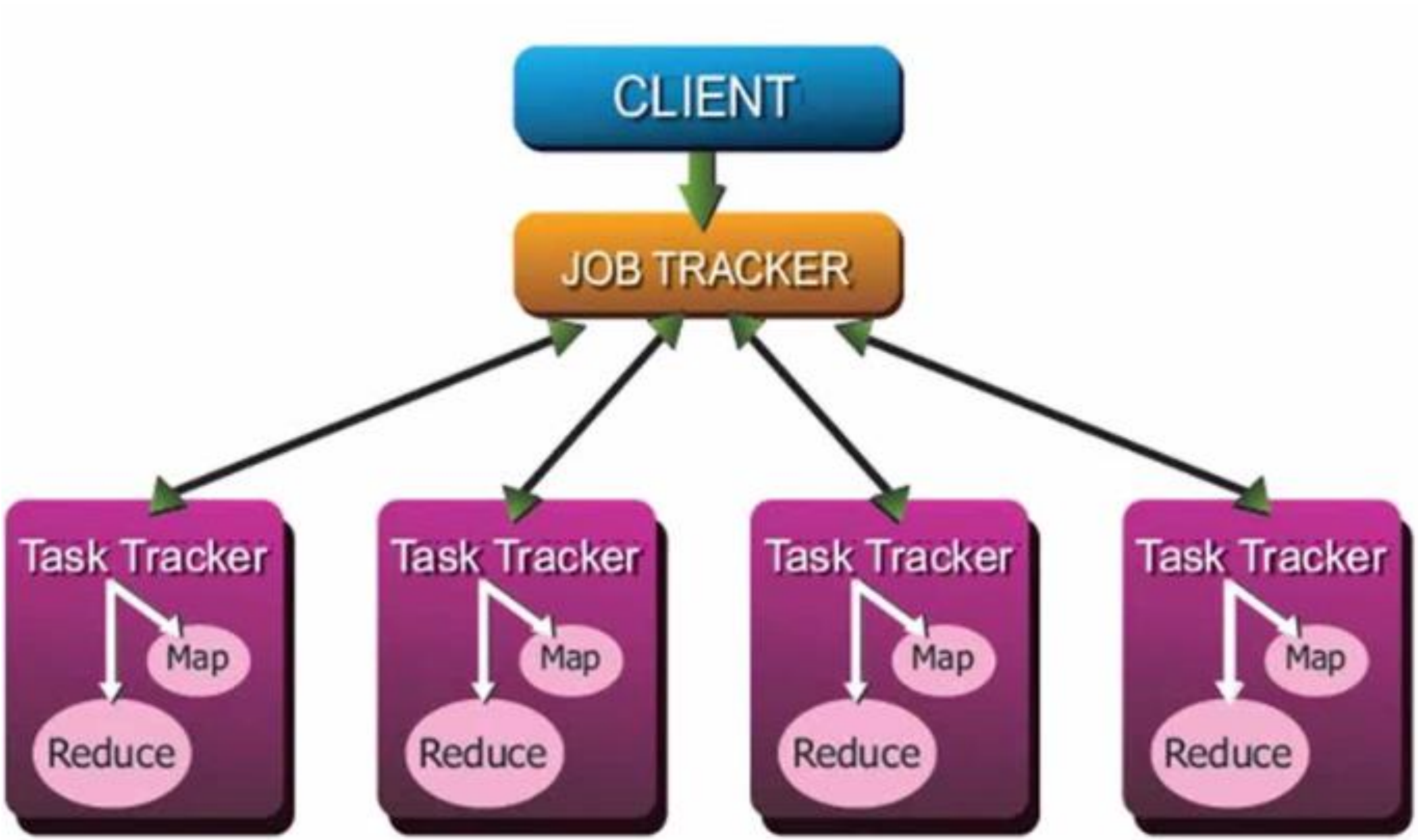
➢ Maintains and manages the blocks which are present in the data nodes

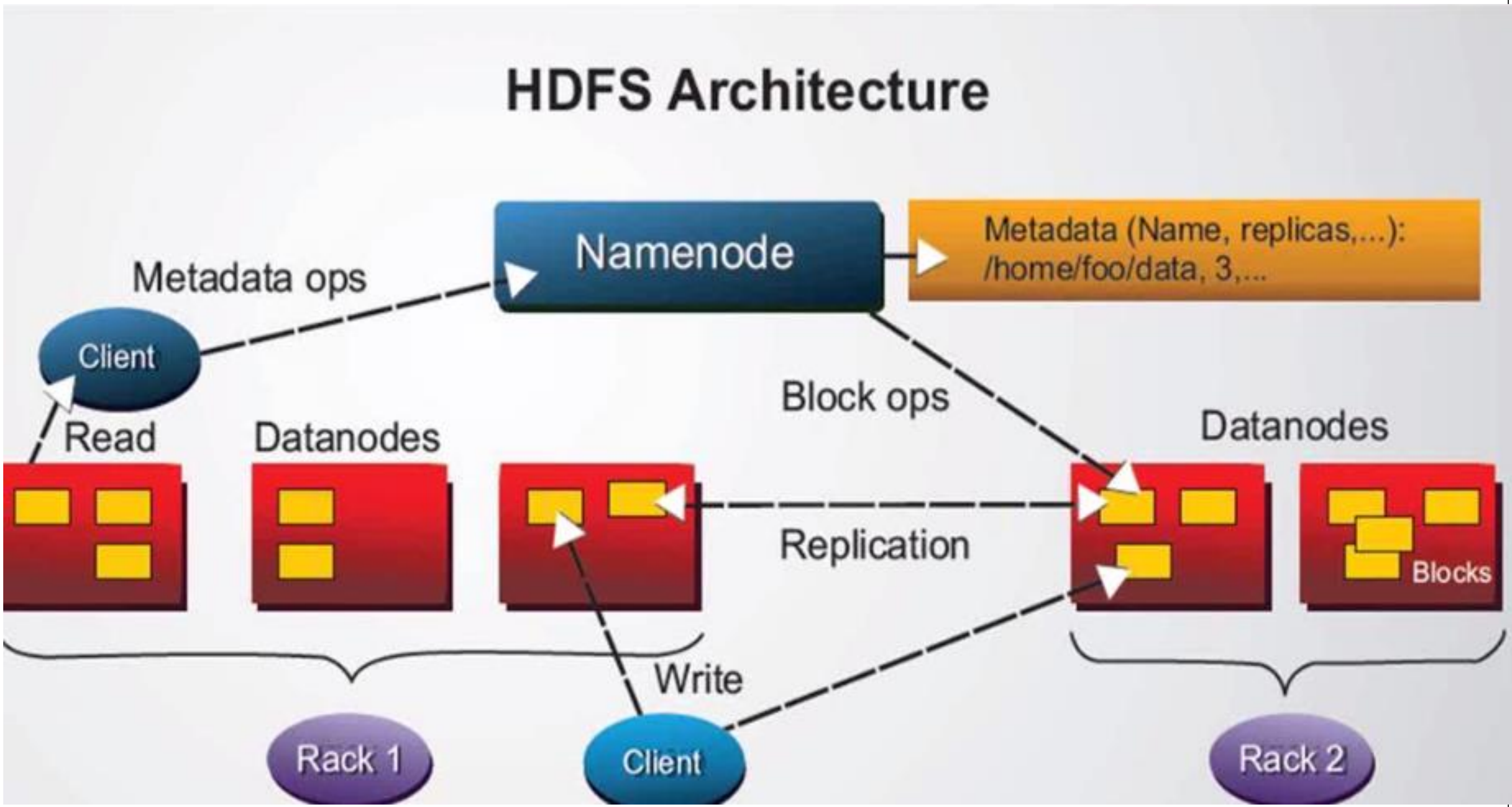
- Data Node

➢ Slaves which are deployed on each machine and provide the actual storage

➢ Responsible for serving read write request for the clients.

Job Tracker and Task Tracker



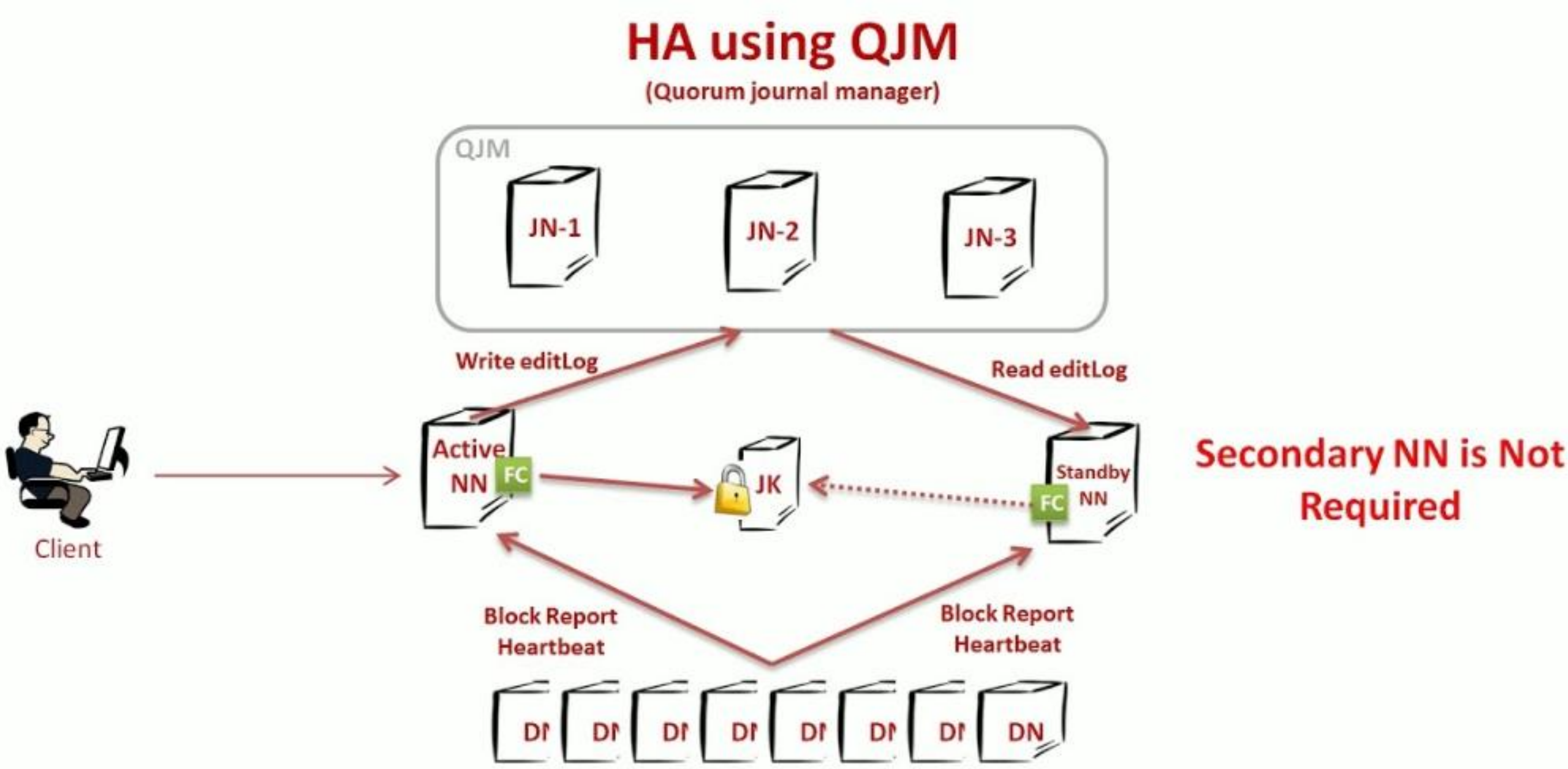

HDFS Architecture
Namenode
Metadata (Name, replicas,...):
/home/foo/data, 3,...
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Write
Rack 1
Client
Rack 2

Fault Tolerance :

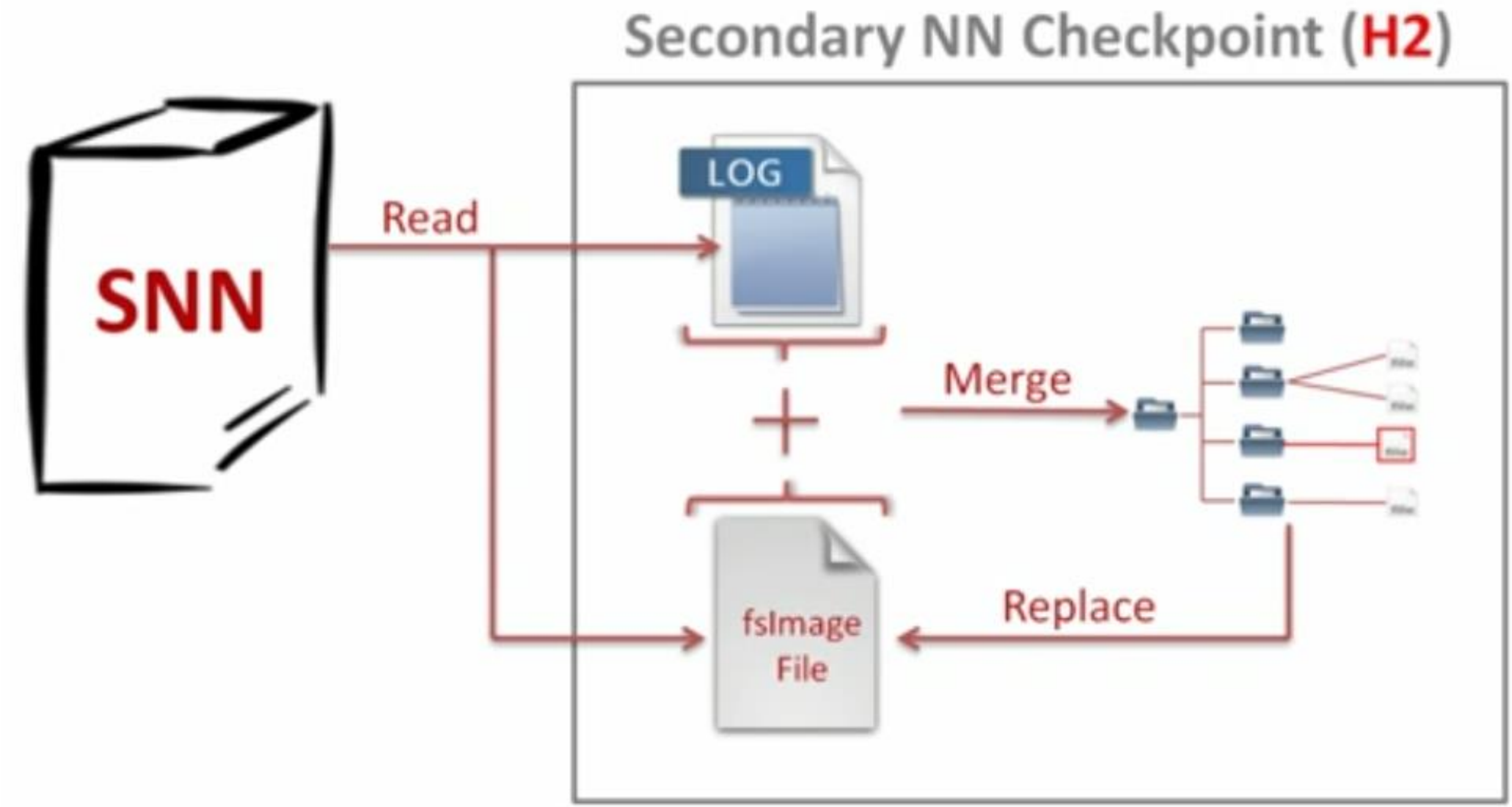
Rack Awareness, Failures: rack , data node and disc, Replication factor

High Availability : **It is the percentage of uptime of the system.**

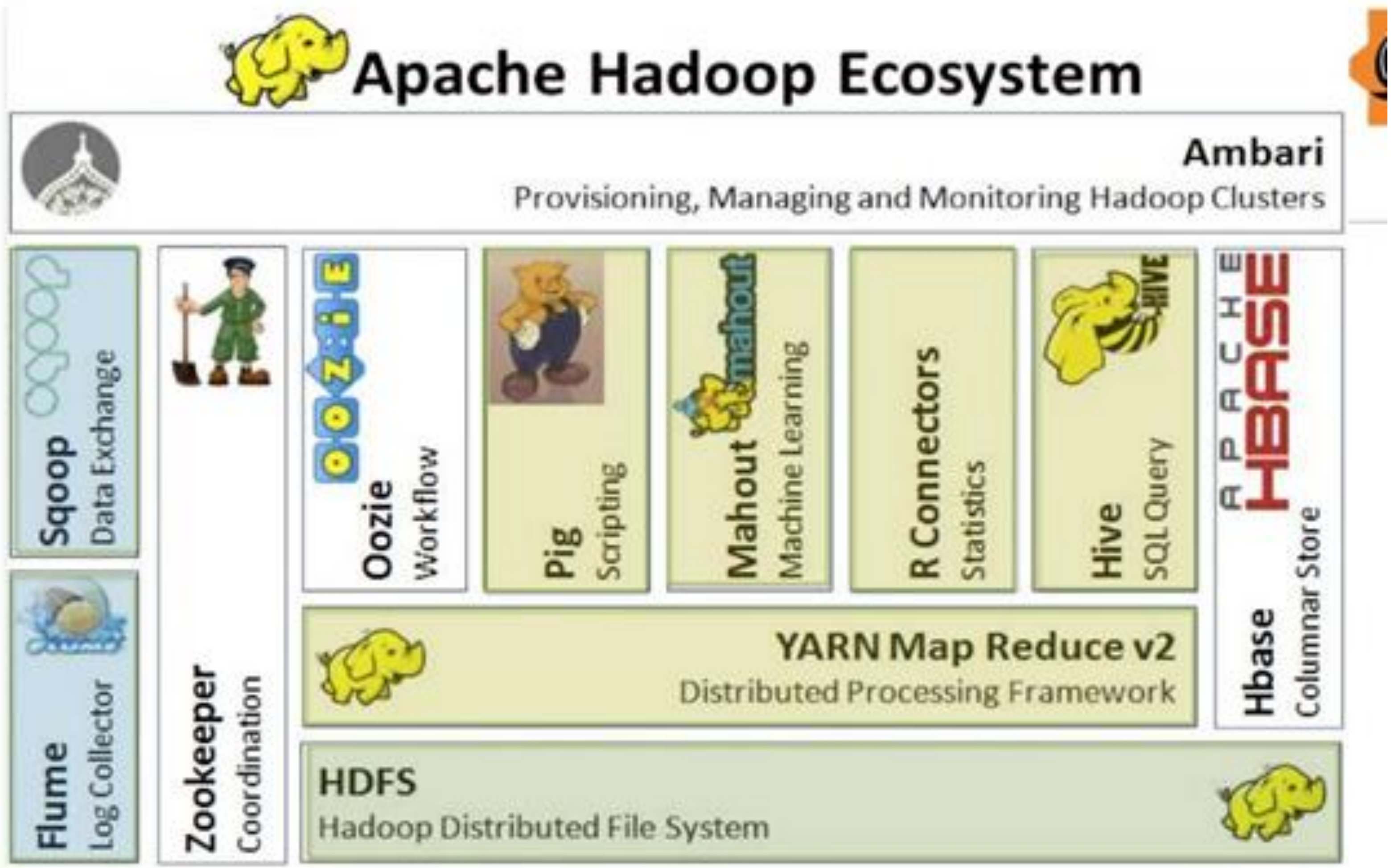
It shows the percentage of the time services are up and operational. Every industry require 99.99% uptime for their critical system.



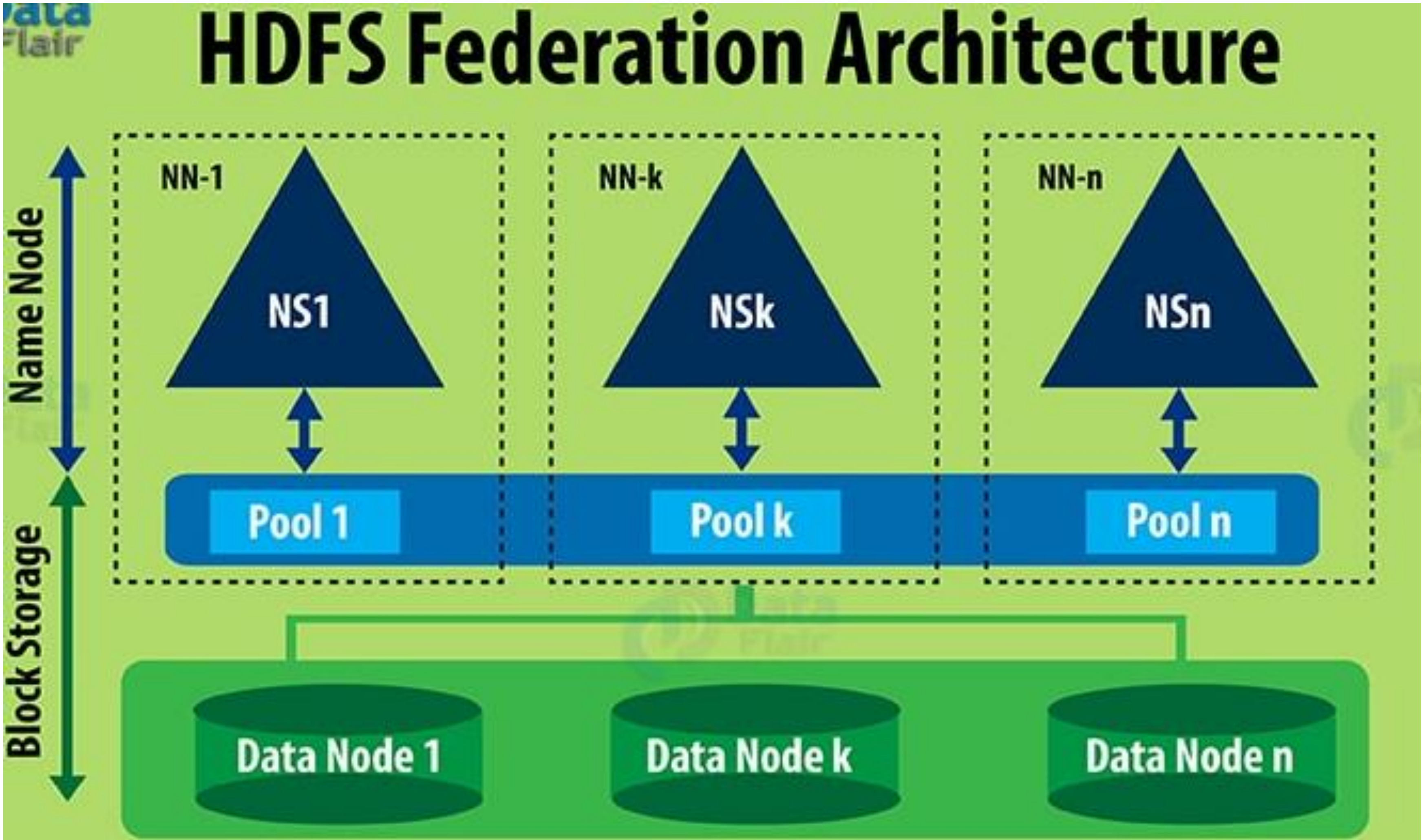
Secondary Name Node : **what will happen if we restart the name node?**



HDFS EcoSystem



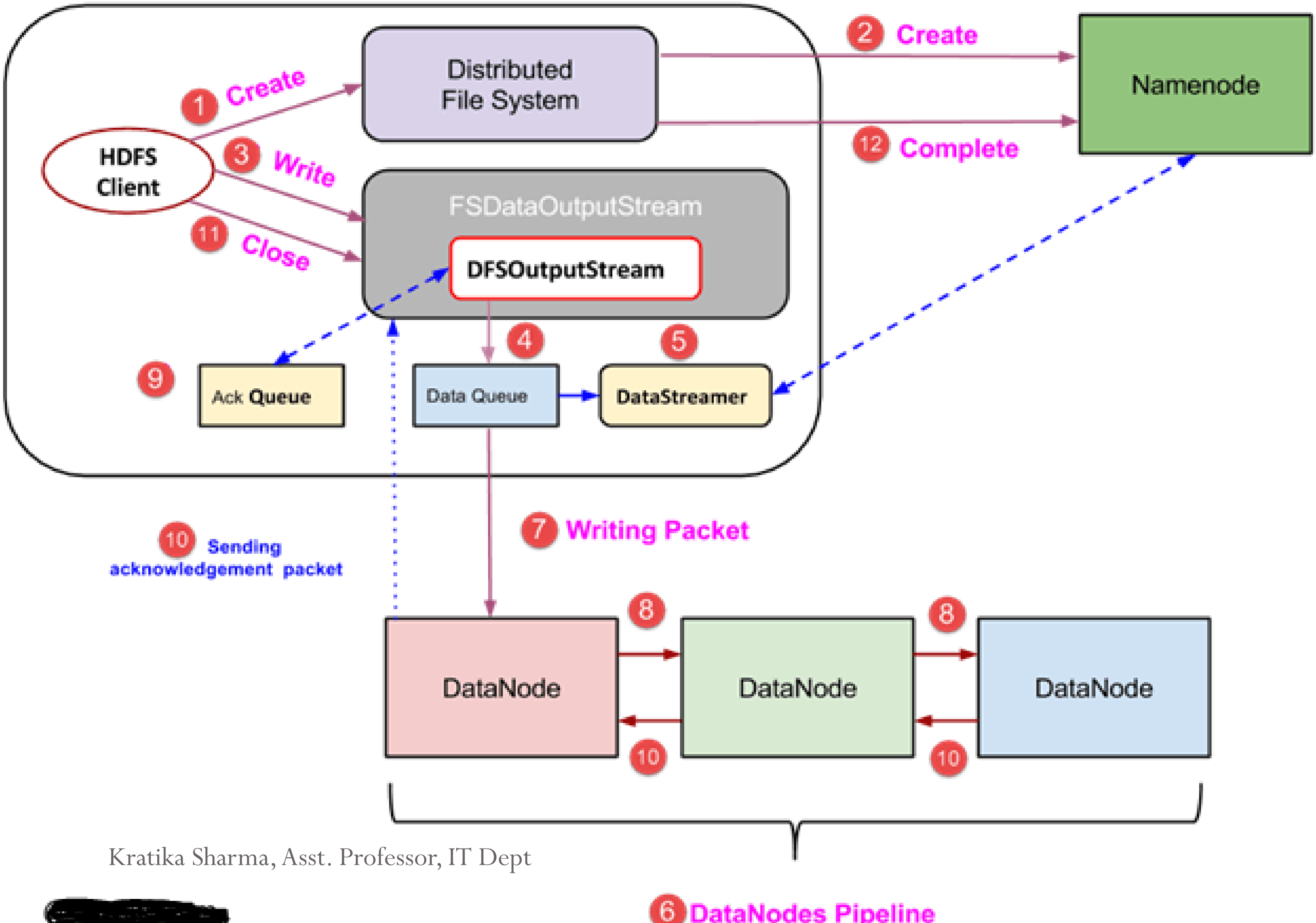
HDFS Federation



HDFS federation

- It is introduced in the 2.x release series, allows a cluster to scale by adding namenodes, each of which manages a portion of the filesystem namespace.
- For example, one namenode might manage all the files rooted under */user, say, and a second namenode might handle files* under */share*.

Anatomy of a File Write



Kratika Sharma, Asst. Professor, IT Dept

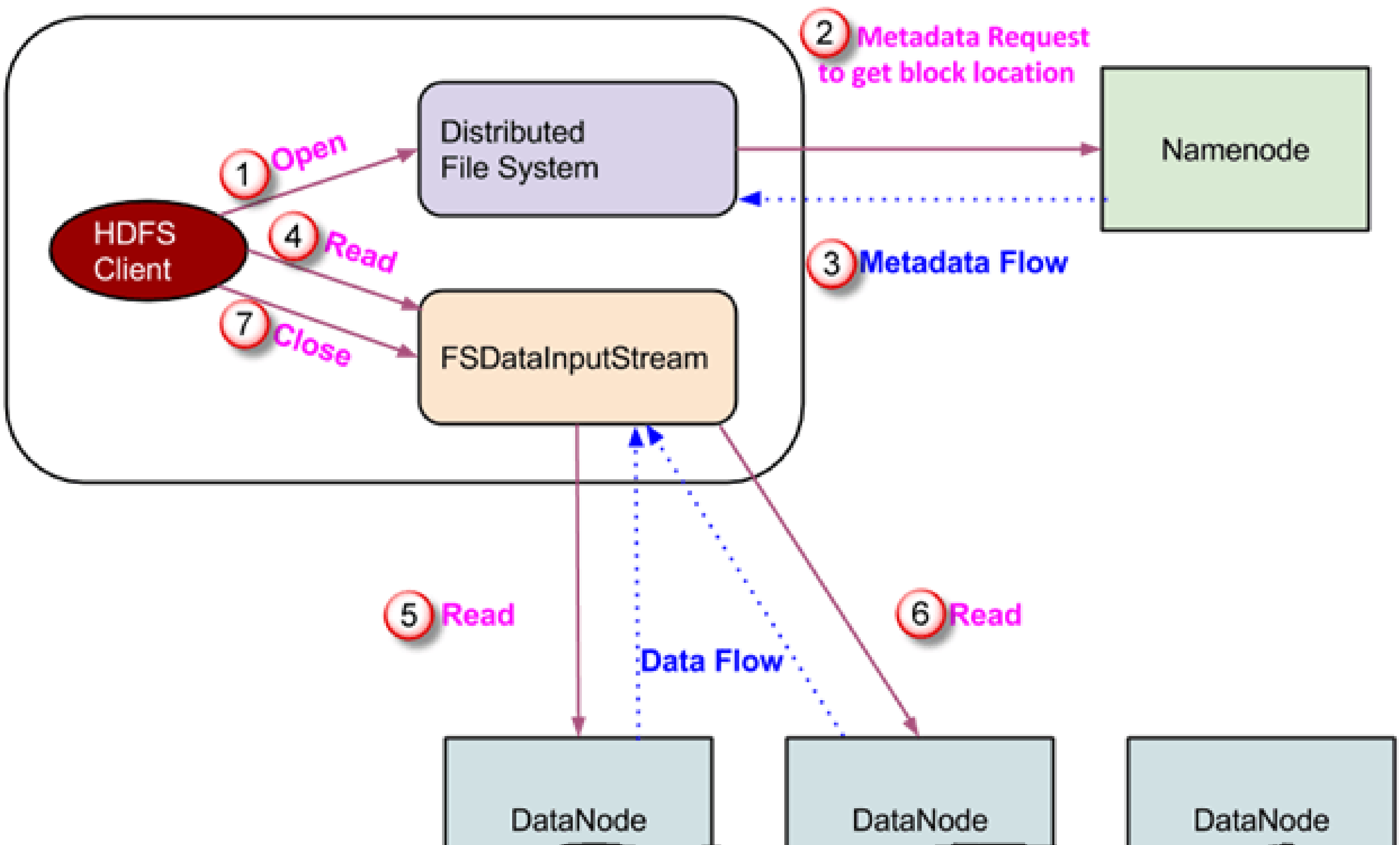
[redacted]

1. Client initiates write operation by calling 'create()' method of DistributedFileSystem object which creates a new file - Step no. 1 in above diagram.
2. DistributedFileSystem object connects to the NameNode using RPC call and initiates new file creation. However, this file create operation does not associate any blocks with the file. It is the responsibility of NameNode to verify that the file (which is being created) does not exist already and client has correct permissions to create new file. If file already exists or client does not have sufficient permission to create a new file, then **IOException** is thrown to client. Otherwise, operation succeeds and a new record for the file is created by the NameNode.
3. Once new record in NameNode is created, an object of type FSDataOutputStream is returned to the client. Client uses it to write data into the HDFS. Data write method is invoked (step 3 in diagram).
4. FSDataOutputStream contains DFSOutputStream object which looks after communication with DataNodes and NameNode. While client continues writing data, **DFSOutputStream** continues creating packets with this data. These packets are en-queued into a queue which is called as **DataQueue**.
5. There is one more component called **DataStreamer** which consumes this **DataQueue**. DataStreamer also asks NameNode for allocation of new blocks thereby picking desirable DataNodes to be used for replication.
6. Now, the process of replication starts by creating a pipeline using DataNodes. In our case, we have chosen replication level of 3 and hence there are 3 DataNodes in the pipeline.

Anatomy of file write continue

7. The DataStreamer pours packets into the first DataNode in the pipeline.
8. Every DataNode in a pipeline stores packet received by it and forwards the same to the second DataNode in pipeline.
9. Another queue, 'Ack Queue' is maintained by DFSOutputStream to store packets which are waiting for acknowledgement from DataNodes.
10. Once acknowledgement for a packet in queue is received from all DataNodes in the pipeline, it is removed from the 'Ack Queue'. In the event of any DataNode failure, packets from this queue are used to reinitiate the operation.
11. After client is done with the writing data, it calls close() method (Step 9 in the diagram) Call to close(), results into flushing remaining data packets to the pipeline followed by waiting for acknowledgement.
12. Once final acknowledgement is received, NameNode is contacted to tell it that the file write operation is complete.

Anatomy of File Read



Anatomy of a File Write

1. Client initiates read request by calling **'open()'** method of FileSystem object; it is an object of type **DistributedFileSystem**.
2. This object connects to namenode using RPC and gets metadata information such as the locations of the blocks of the file. Please note that these addresses are of first few block of file.
3. In response to this metadata request, addresses of the DataNodes having copy of that block, is returned back.
4. Once addresses of DataNodes are received, an object of type **FSDataInputStream** is returned to the client. **FSDataInputStream** contains **DFSInputStream** which takes care of interactions with DataNode and NameNode. In step 4 shown in above diagram, client invokes **'read()'** method which causes **DFSInputStream** to establish a connection with the first DataNode with the first block of file.
5. Data is read in the form of streams wherein client invokes **'read()'** method repeatedly. This process of **read()** operation continues till it reaches end of block.
6. Once end of block is reached, DFSInputStream closes the connection and moves on to locate the next DataNode for the next block
7. Once client has done with the reading, it calls **close()** method.

Parallel Copying with distcp

- Hadoop comes with a useful program called *distcp for copying data to and from Hadoop* filesystems in parallel.

- One use for *distcp is as an efficient replacement for hadoop fs -cp. For example, you can* copy one file to another with:

hadoop distcp file1 file2

- You can also copy directories:

hadoop distcp dir1 dir2

- You can also update only the files that have changed using the -update option.

hadoop distcp -update dir1 dir2

- A very common use case for *distcp is for transferring data between two HDFS clusters.*
- For example, the following creates a backup of the first cluster's */foo directory on the* second:

hadoop distcp -update -delete -p hdfs://namenode1/foo hdfs://namenode2/foo

- If the two clusters are running incompatible versions of HDFS, then you can use the webhdfs protocol to *distcp between them:*

hadoop distcp webhdfs://namenode1:50070/foo webhdfs://namenode2:50070/foo

Keeping an HDFS Cluster Balanced

Unit 2

- **MapReduce:** A Weather Dataset, Data Format, Analyzing the Data with Hadoop, Map and Reduce, Java MapReduce, Scaling Out, Data Flow, Combiner Functions, Running a Distributed MapReduce Job
- **Developing a MapReduce Application:** Writing a Unit Test with MRUnit, Mapper,Reducer, Running Locally on Test Data, Running a Job in a Local Job Runner, Testing the Driver, Running on a Cluster, Packaging a Job, Launching a Job, The MapReduce Web

A Weather Dataset

```
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
```

- % **ls raw/1990 | head**

010010-99999-1990.gz

010014-99999-1990.gz

010015-99999-1990.gz

010016-99999-1990.gz

010017-99999-1990.gz

010030-99999-1990.gz

010040-99999-1990.gz

010080-99999-1990.gz

Analyzing the Data with Unix Tools

```
#!/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
```

- Here is the beginning of a run:
- % **./max_temperature.sh**

1901 317

1902 244

1903 289

1904 256

1905 283…

Analyzing the Data with Hadoop

- To visualize the way the map works, consider the following sample lines of input data

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, 0067011990999991**950051507004...9999999N9+00001+9999 9999999...)**

(106,0043011990999991**950051512004...9999999N9+00221+99 999999999)**

(212, 0043011990999991**950051518004...9999999N900111+999999 99999...)**

(318,0043012650999991**949032412004...0500001N9+01111+99 999999999)**

(424,0043012650999991**949032418004...0500001N9+00781+99 999999999)**

- The keys are the line offsets within the file, which we ignore in our map function. The map function merely extracts the year and the air temperature (indicated in bold text), and emits them as its output:

(1950, 0)

(1950, 22)

(1950, −11)

(1949, 111)

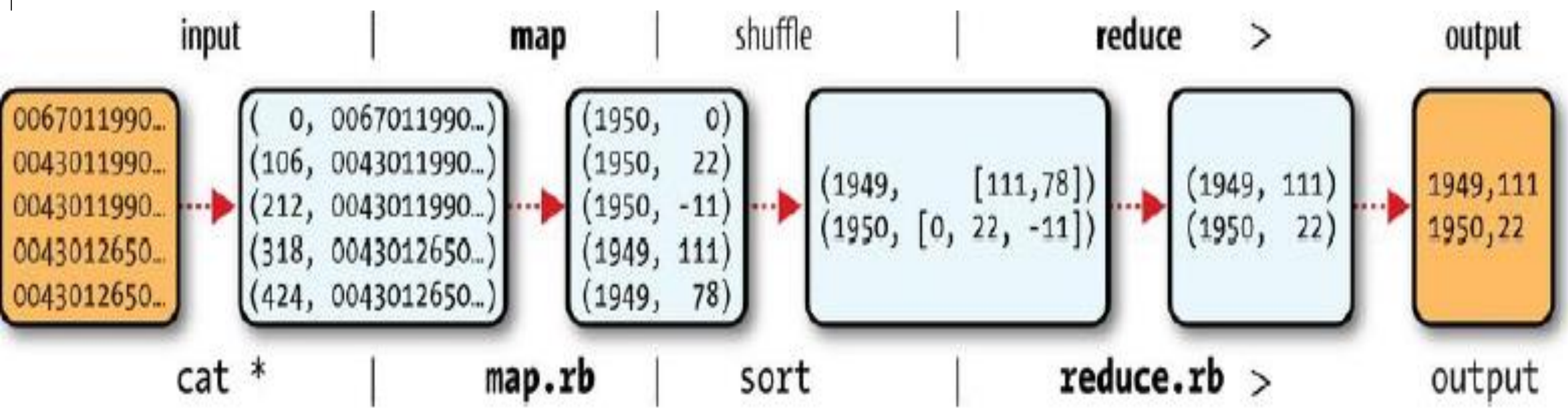
(1949, 78)

- 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, continuing the example, our reduce function sees the following input:

(1949, [111, 78])

(1950, [0, 22, −11])

MapReduce Logical Dataflow



Java MapReduce

```
public class MaxTemperatureMapper
extends Mapper<LongWritable,Text,Text, IntWritable> {
private static final int MISSING = 9999;
@Override
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) == '+') { // parseInt doesn't like leading plus signs
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));
```

Reducer for the maximum temperature example

```
public class MaxTemperatureReducer
extends Reducer<Text, IntWritable,Text, IntWritable> {
@Override
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));
```

Writing a Unit Test with MRUnit

```
import java.io.IOException;
import org.apache.hadoop.io.*;
import
   org.apache.hadoop.mrunit.mapreduce.MapDriver;
import org.junit.*;
```

```
public class MaxTemperatureMapperTest {
@Test
public void processesValidRecord() throws IOException,
   InterruptedException {
Text value = new
   Text("0043011990999991950051518004+68750+023550FM-
   12+0382" +
// Year ^^^^
"99999V0203201N00261220001CN9999999N9-
   00111+99999999999");
// Temperature ^^^^^
new MapDriver<LongWritable,Text,Text, IntWritable>()
.withMapper(new MaxTemperatureMapper())
.withInput(new LongWritable(0), value)
.withOutput(new Text("1950"), new IntWritable(-11))
.runTest();
}
}
```

Reducer Test

```
@Test
public void returnsMaximumIntegerInValues() throws IOException,
InterruptedException {
new ReduceDriver<Text, IntWritable, Text, IntWritable>()
.withReducer(new MaxTemperatureReducer())
.withInput(new Text("1950"),
Arrays.asList(new IntWritable(10), new IntWritable(5)))
.withOutput(new Text("1950"), new IntWritable(10))
.runTest();
```

Running Locally on Test Data :Local Job Runner

```
public class MaxTemperatureDriver extends Configured
   implements Tool {
@Override
public int run(String[] args) throws Exception {
if (args.length != 2) {
System.err.printf("Usage: %s [generic options] <input>
   <output>\n",
getClass().getSimpleName());
ToolRunner.printGenericCommandUsage(System.err);
return -1;
}
```

```
Job job = new Job(getConf(), "Max temperature");
job.setJarByClass(getClass());
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
job.setMapperClass(MaxTemperatureMapper.class);
job.setCombinerClass(MaxTemperatureReducer.class);
job.setReducerClass(MaxTemperatureReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
return job.waitForCompletion(true) ? 0 : 1;
}
```

```
public static void main(String[] args) throws
   Exception {
int exitCode = ToolRunner.run(new
   MaxTemperatureDriver(), args);
System.exit(exitCode);
}
}
```

Running Hadoop On Ubuntu Linux (Single-Node Cluster)

- Oracle java 8

- Adding a dedicated Hadoop system user

$ sudo addgroup hadoop

$ sudo adduser --ingroup hadoop hduser

- Configuring SSH

hduser@ubuntu:~$ ssh-keygen -t rsa -P ""

hduser@ubuntu:~$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys

- Hadoop

Installation

Update $HOME/.bashrc

- Starting your single-node cluster

Run the command:

hduser@ubuntu:~$ /usr/local/hadoop/bin/start-all.sh

hduser@ubuntu:/usr/local/hadoop$ jps

2287 TaskTracker 2149 JobTracker 1938 DataNode 2085 SecondaryNameNode 2349 Jps 1788 NameNode

- Stopping your single-node cluster

Run the command

hduser@ubuntu:~$ /usr/local/hadoop/bin/stop-all.sh

- Copy local example data to HDFS
- Run the MapReduce job
- Retrieve the job result from HDFS

Hadoop Web Interfaces

Hadoop comes with several web interfaces which are by default (see conf/hadoop-default.xml) available at these locations:

http://localhost:50070/ – web UI of the NameNode daemon

http://localhost:50030/ – web UI of the JobTracker daemon

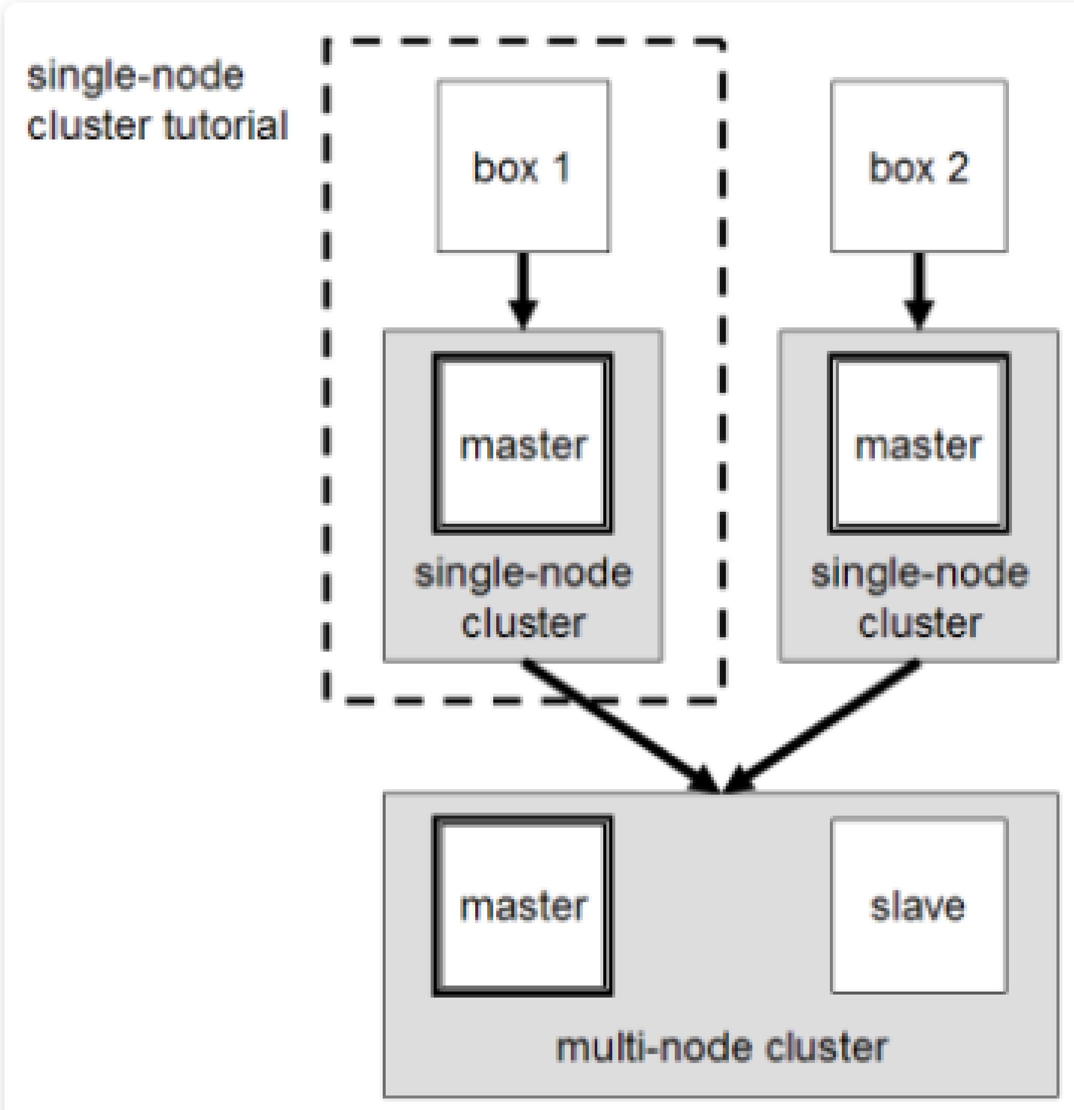
The JobTracker web UI provides information about general job statistics of the Hadoop cluster, running/completed/failed jobs and a job history log file.

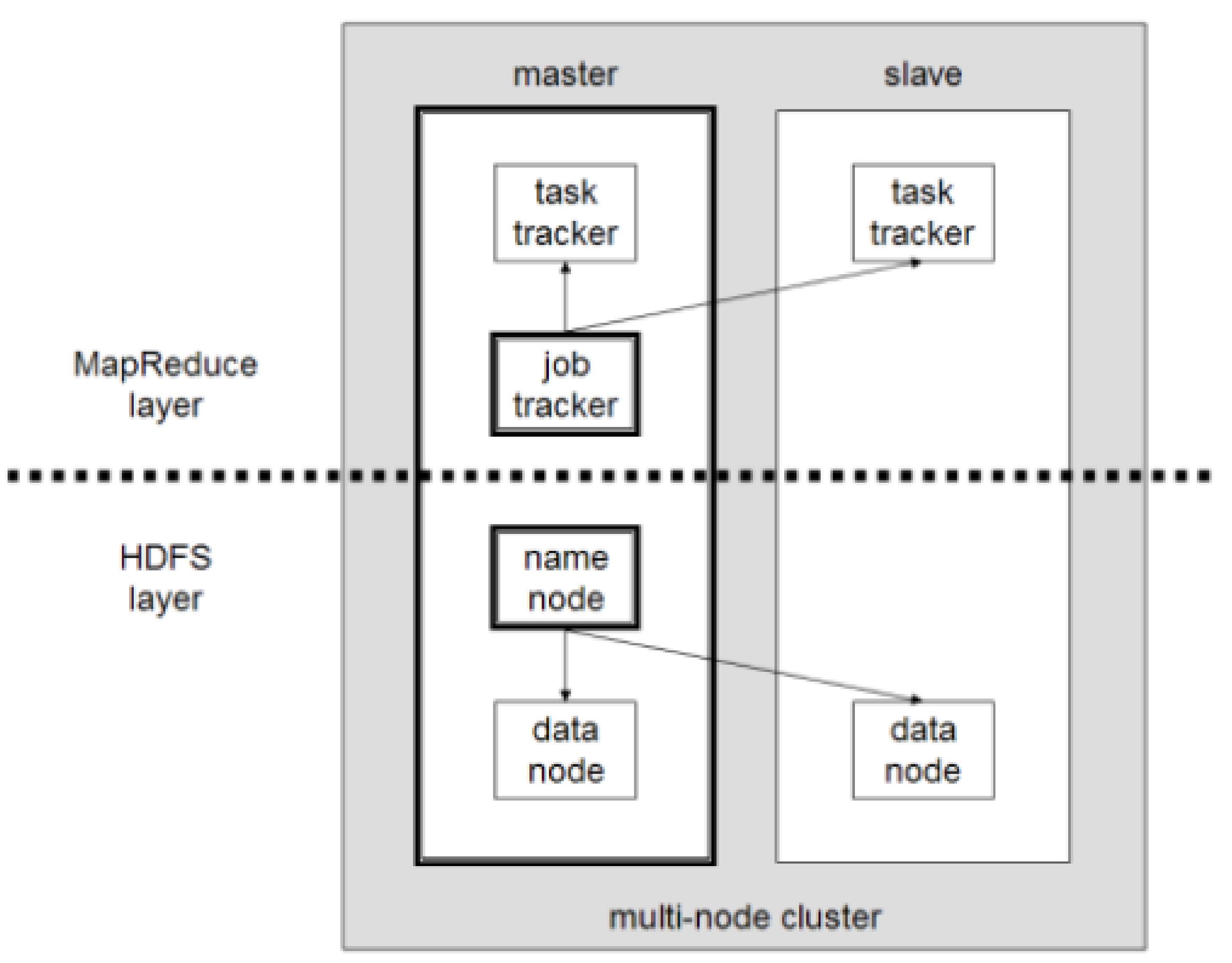
http://localhost:50060/ – web UI of the TaskTracker daemon

The task tracker web UI shows you running and non-running tasks. It also gives access to the “local machine’s” Hadoop log files.

Running on a Cluster

- Packaging a Job
- Launching a job


single-node
cluster tutorial
box 1
box 2
master
single-node
cluster
master
single-node
cluster
master
slave
multi-node cluster


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