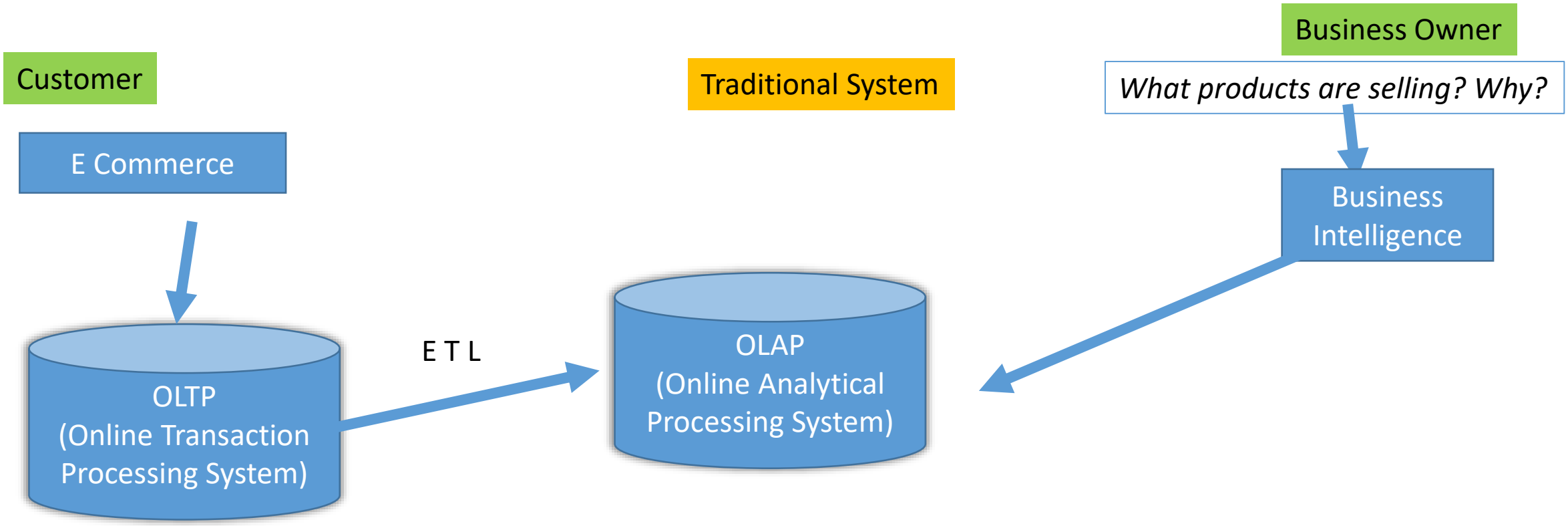
DSC 406 Big Data Technologies

Traditional System



What is the traditional system?

- *Architecture is based on RDBMS (Relational Database Management System)*
- *Handling data in the enterprise*
- *Classifications*

- *OLTP --> Online Transaction Processing System*
- *Ex. A sale, POS*
- *Transactional Data, Customer data*
- *Oracle, MySQL, SQL Server, DB2, Sybase, PostgreSQL*

- *OLAP --> Online Analytical Processing System*
- *Ex. How is my business doing, profit, revenue and business metrics*
- *Analytics data, business owner*
- *Teradata, Netezza, SAP Hana, Vertica, Oracle*

What is Big Data?

- *Big data is a broad term for data sets so large and complex that traditional data processing applications are inadequate*
- *Challenges include analysis, capture, curation, search, sharing, storage, transfer, visualization, and information privacy*
- *The term often refers simply to the use of predictive analytics or other certain advanced methods to extract value from data, and seldom to a particular size of data set*
- *We currently see the exponential growth in the data storage as the data is now more than text data.*
- *The current day data is far more than just the text data*
- *We see exponential growth in the data storage*
- *The data is found in the form of videos, music, large images etc. too*
- *Terabytes and Petabytes of storage has become a common storage size*

What is Big Data?

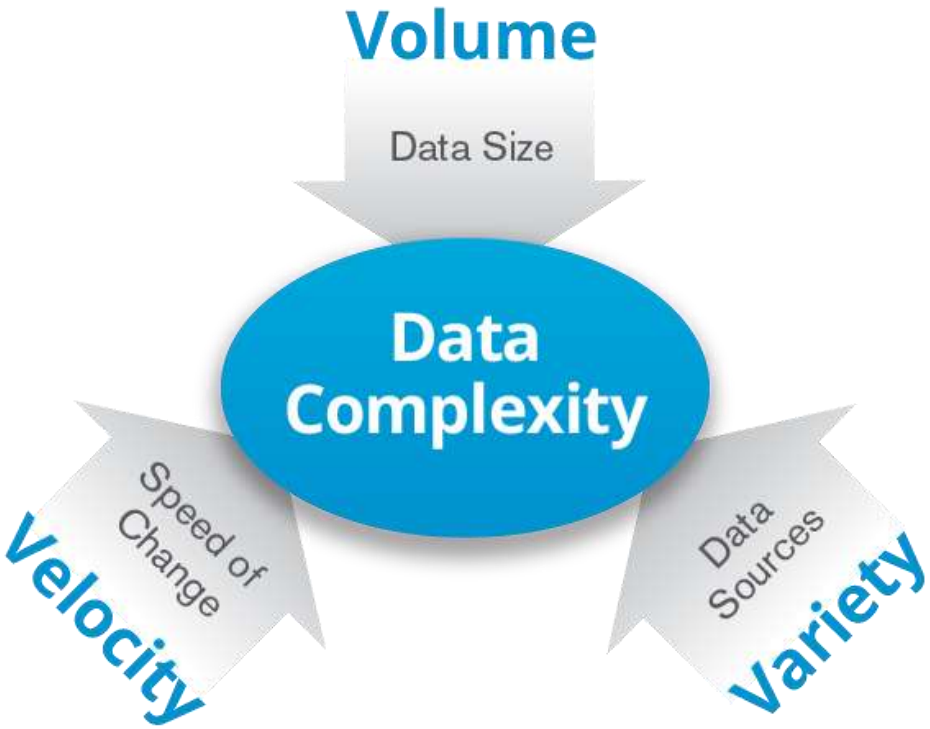
- *The data growth and social media explosion have changed how we look at the data*
- *Data is streaming in at unexpected speed and must be dealt with in a timely manner*
- *The Velocity is the speed at which the data is created, stored, analyzed and visualized*
- *Reacting quickly enough to deal with data velocity is a challenge for most organizations*
- *Data can be stored in multiple formats like database, excel, csv, access or a simple text file*
- *Sometimes the data is not even in the traditional format, it may be in the form of video, SMS, documents, PDF, WhatsApp, tweets, comments on facebook etc..*
- *The organizations need to capture it and arrange it to make it meaningful information.*
- *The variety of data is classified into 3 formats: structured data, semi-structured data and unstructured data*
- *The wide variety of data requires a different approach as well as different techniques to store all raw data*

What is Big Data? → Problem

- *3 Vs of Big Data*
  - *Volume → Size*
  - *Velocity → Speed*
  - *Variety → Different forms of data*

- *Hadoop's V → VALUE*

- *How to store Big Data? → HDFS*
- *How to process Big Data? → YARN*



Data Measurement Scale

• *1 Kilobyte*	*KB*	*1000*
• *1 Megabyte*	*MB*	*1000000*
• *1 Gigabyte*	*GB*	*1000000000*
• *1 Terabyte*	*TB*	*1000000000000*
• *1 Petabyte*	*PB*	*1000000000000000*
• *1 Exabyte*	*EB*	*1000000000000000000*
• ***1 Zettabyte***	***ZB***	***1000000000000000000000 X 16***
• *1 Yotabyte*	*YB*	*1000000000000000000000000*

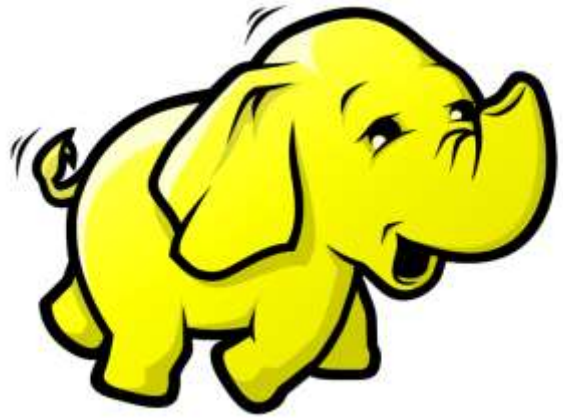
In Bytes

Challenges with the traditional system

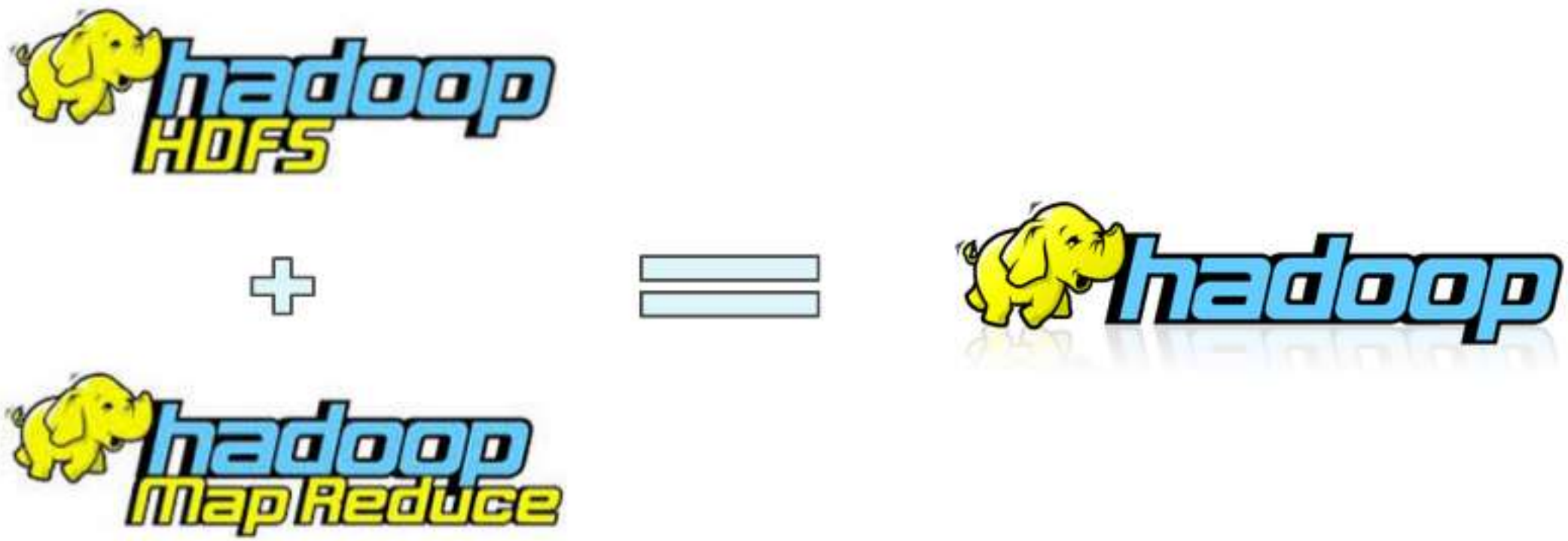
- *Handling Big Data*
- *Cost*
- *Scalability*
- *Latency*

What is Apache Hadoop?

- *The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models*
- *It is designed to scale up from single servers to thousands of machines, each offering local computation and storage*



What is Apache Hadoop?

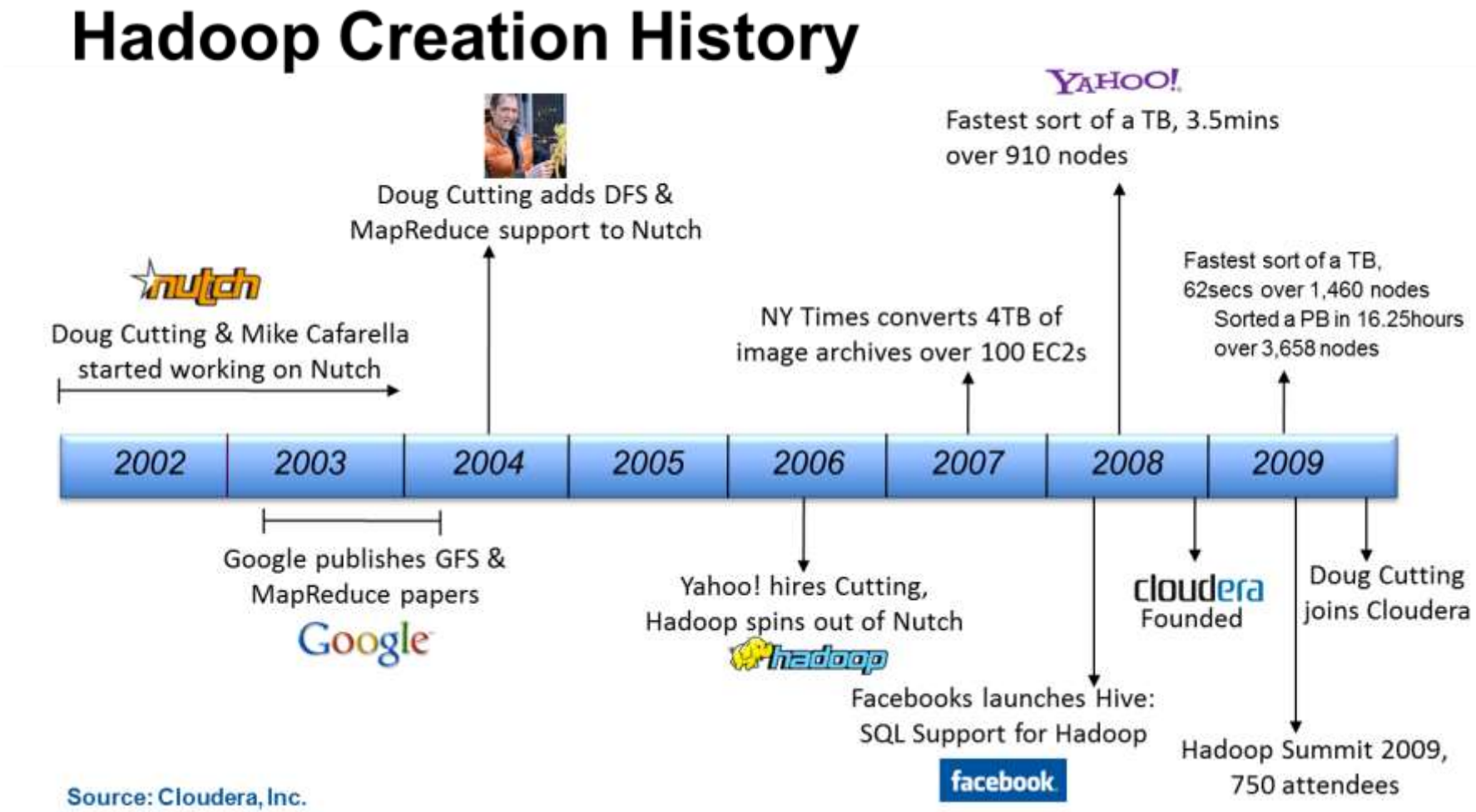


Features of Hadoop

- *Any infra*
- *Open source* http://hadoop.apache.org/
- *Distributed storage - Parallel processing*
- *Fault Tolerance*
- *Scale out architecture*
- *Each node is compute capable*
- *Data locality*
- *Hadoop can work on a single node up to 1000s of servers*
- *Distributed computing is reliable and scalable with Hadoop*
- *Hadoop framework is built in Java*

Note: HDFS datasets are immutable

History of Hadoop

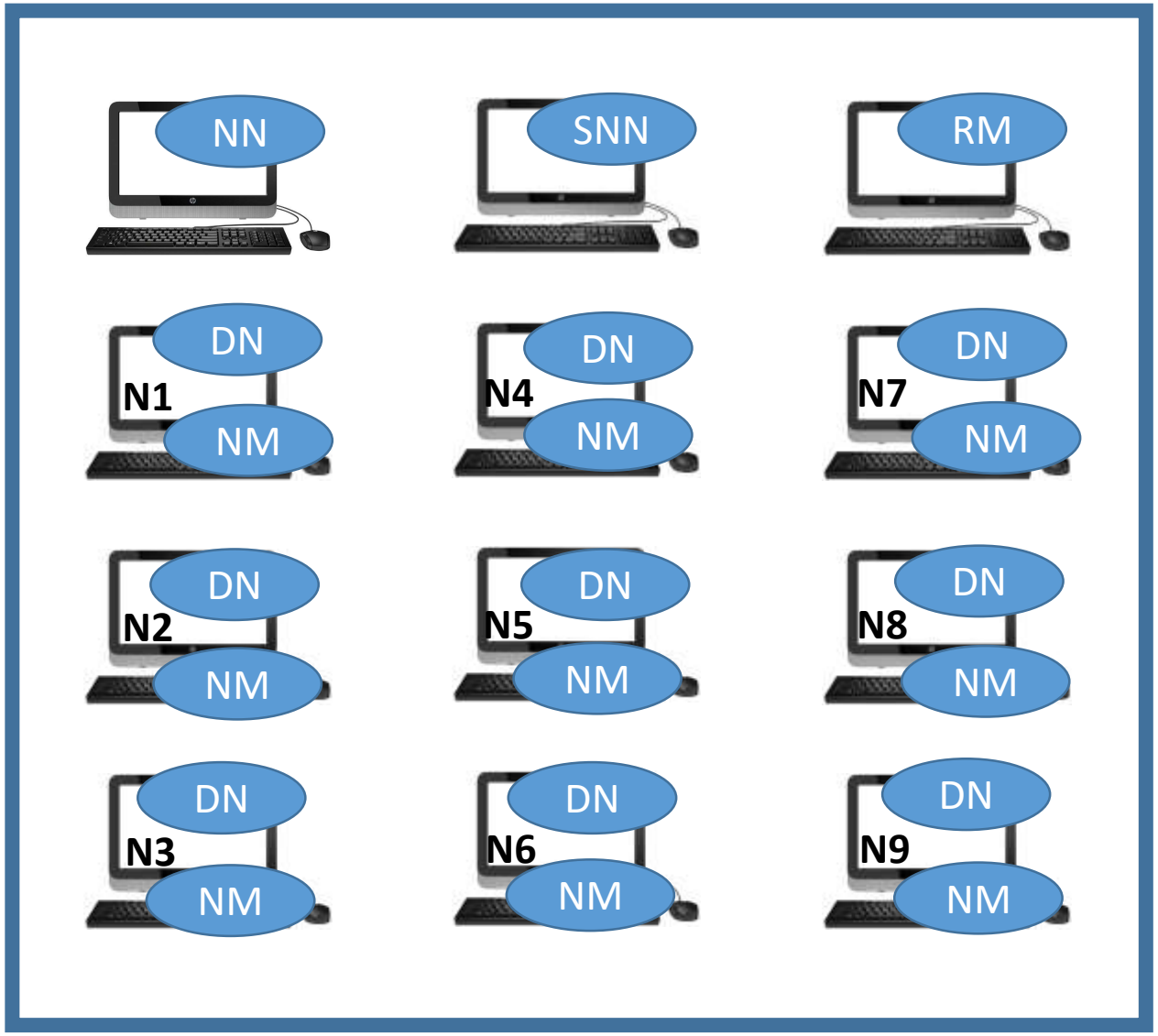


Hadoop Setup Modes

- Standalone Mode
  - Single node, non distributed (Default)
  - Hadoop works as a single Java Process
- *Pseudo Distributed Mode*
  - *Single node, distributed*
  - *Each Hadoop daemon runs inside a separate JVM*
    - *HDFS → 1 NN, 1 DN, 1 SNN*
    - *YARN → 1 RM, 1 NM*
- Fully Distributed Mode
  - Multi node, distributed
  - Hadoop daemons are distributed among many nodes → Typical production environment

Hadoop Daemons distributed over a cluster

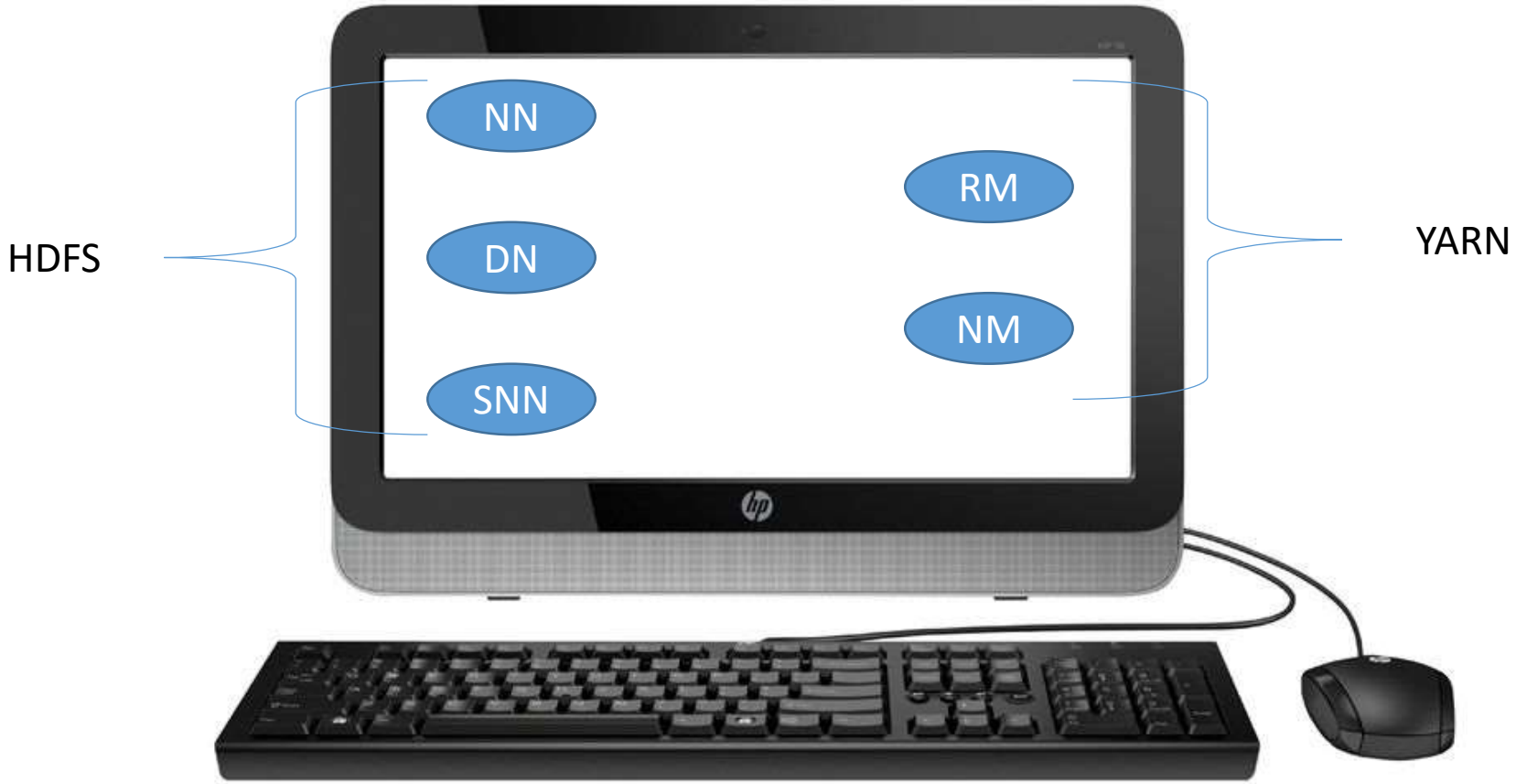
Gateway / Client node



DataNode and NodeManager co-exist

Hadoop Daemons distributed over a single node

Pseudo Distributed Mode



Hadoop FsShell

- *FsShell is a command line interface that lets user interact with data in HDFS*

Hadoop FsShell Reference

- http://hadoop.apache.org/docs/r2.7.5/hadoop-project-dist/hadoop-common/FileSystemShell.html

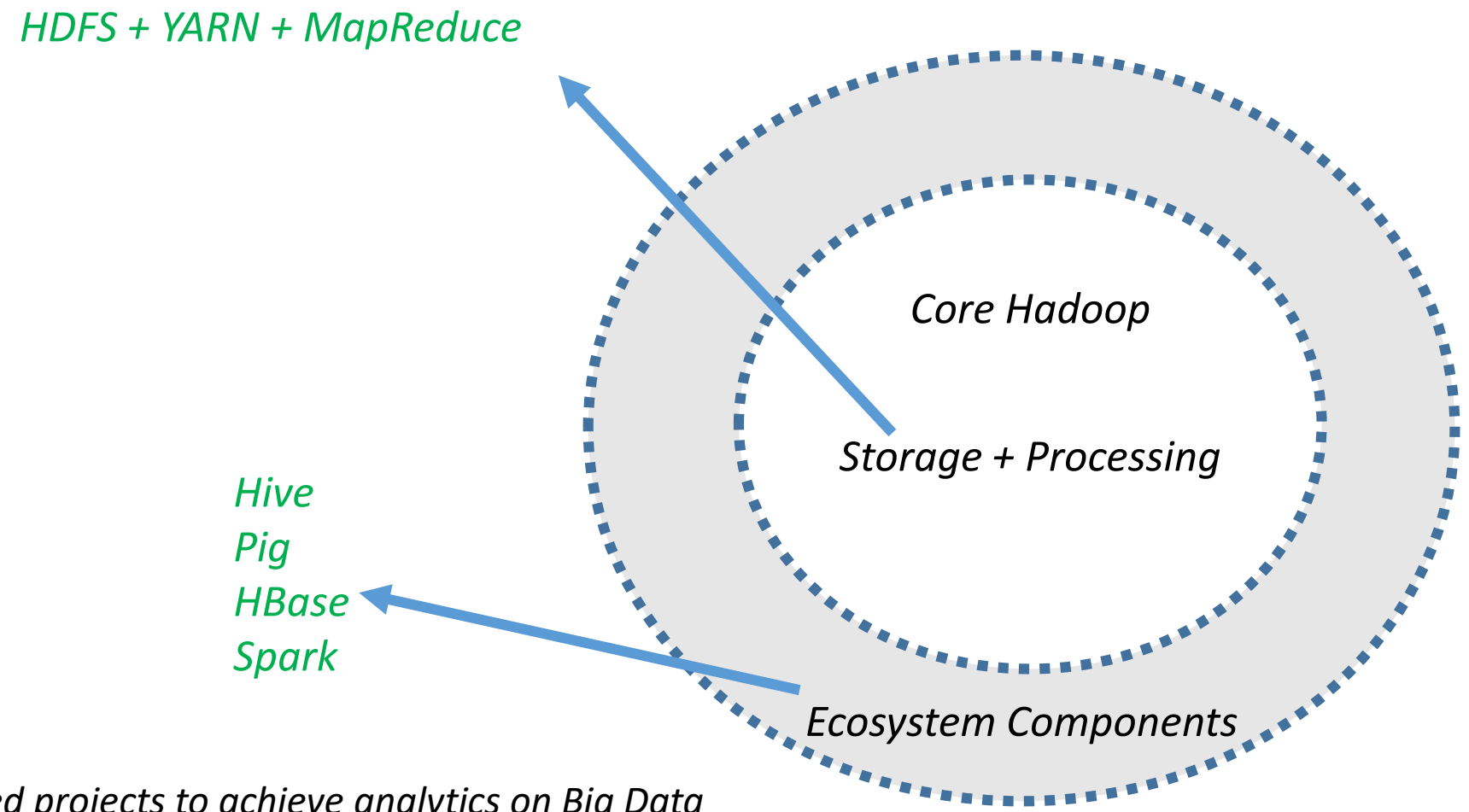
FsShell Commands

- `$ sudo jps`
- `$ hadoop fs -mkdir /Sample`
- `$ hadoop fs -ls /`
- `$ hadoop fs -put /home/cloudera/Desktop/Labs/SampleFile.txt /Sample/`
- `$ hadoop fs -cat /Sample/SampleFile.txt`
- `$ hadoop fs -mkdir /Sample1`
- `$ hadoop fs -cp /Sample/SampleFile.txt /Sample1/`
- `$ hadoop fs -cp /Sample/SampleFile.txt /Sample1/SampleFile1.txt`
- `$ hadoop fs -mv /Sample1/SampleFile.txt /Sample1/SampleFile2.txt`
- `$ hadoop fs -rm /Sample/SampleFile.txt`
- `$ hadoop fs -rm -r /Sample1`
- `$ hadoop fs -put /home/cloudera/Desktop/Labs/SampleFile.txt /Sample/`

WebUI Ports

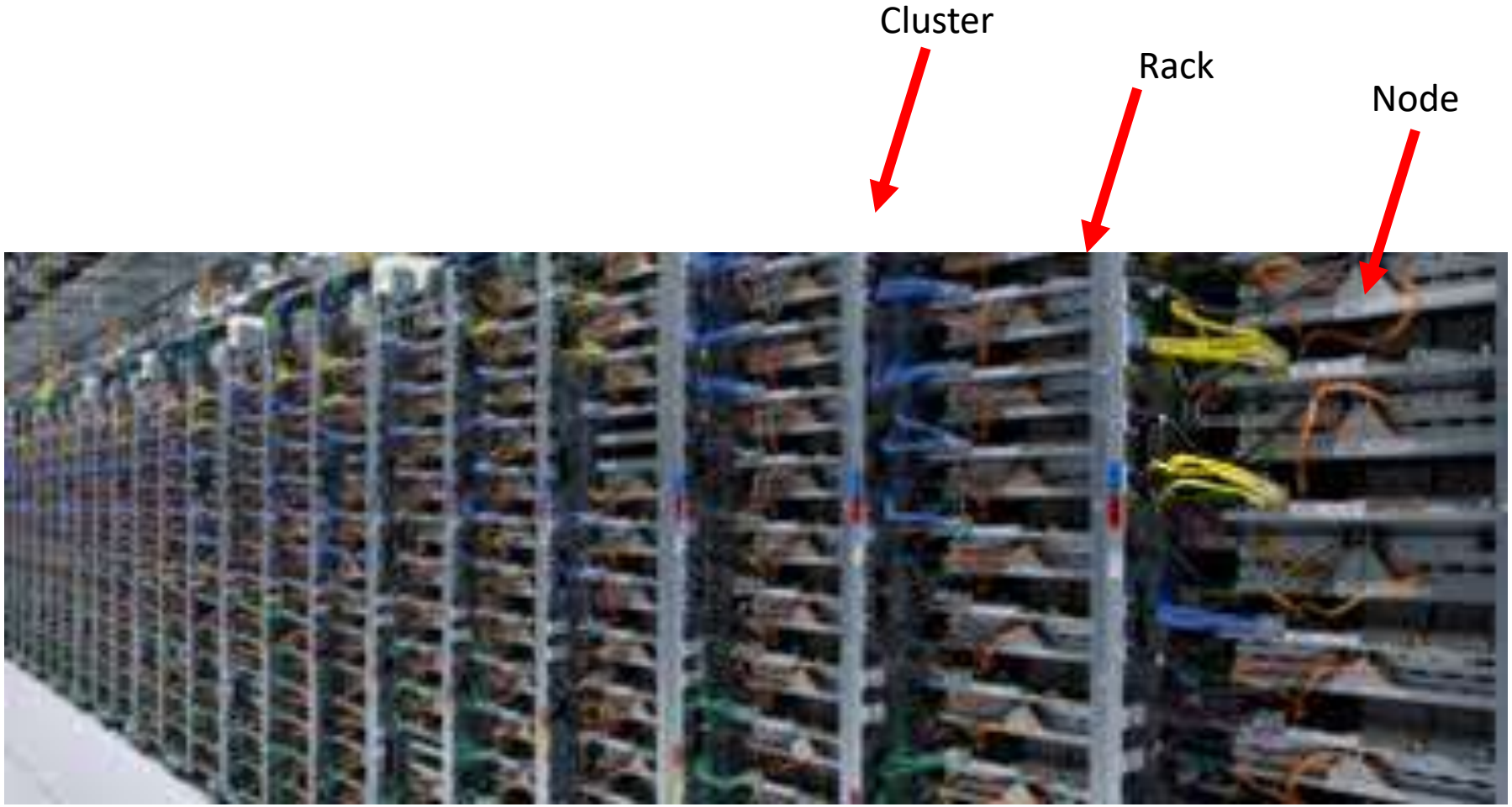
- *WebUI for accessing HDFS*
  - *NameNode WebUI - http://localhost:50070*
- *WebUI for accessing HDFS*
  - *ResourceManager WebUI - http://localhost:8088*
  - *JobHistoryServer WebUI - http://localhost:19888*

Hadoop Ecosystem



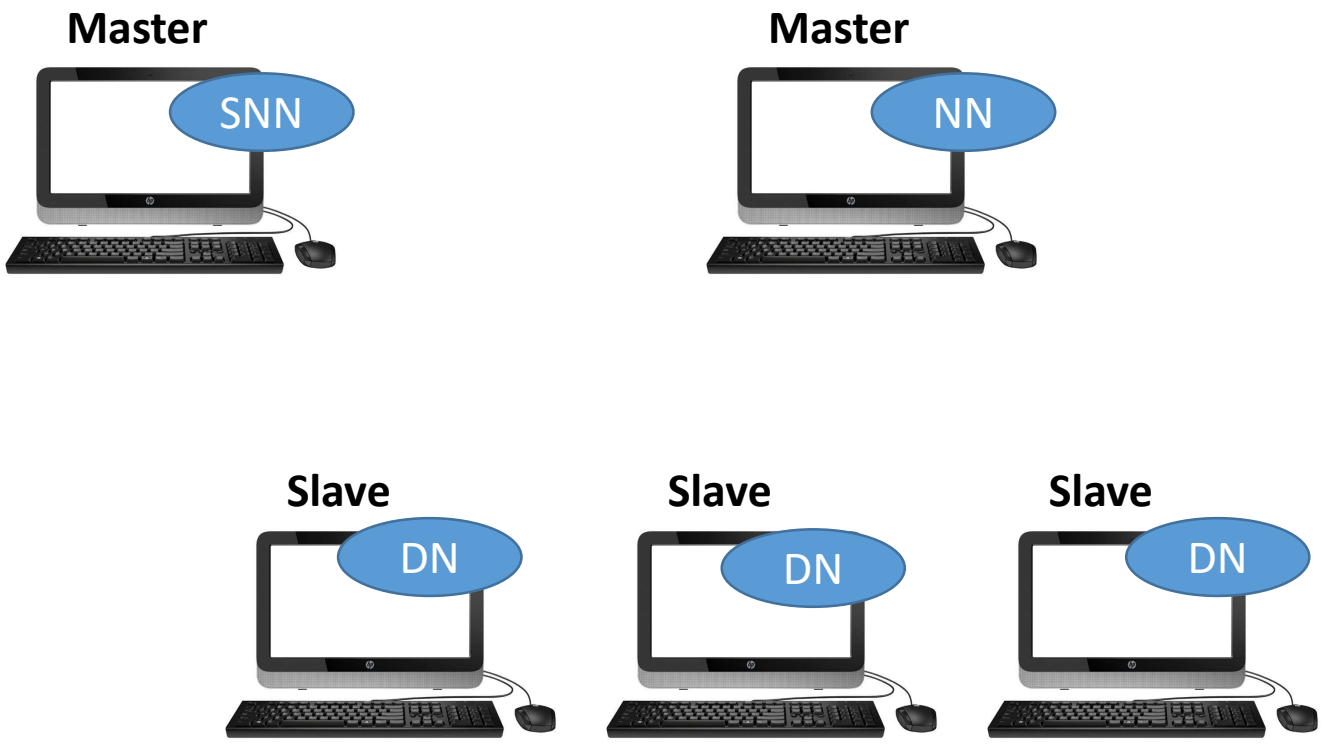
- *We can use Hadoop and related projects to achieve analytics on Big Data*
- *Hadoop systems are capable to query petabytes of scale of data*
- *A single Hadoop cluster can host 1000s of nodes*

Hadoop Terminologies



HDFS Daemons

Master – Slave Architecture

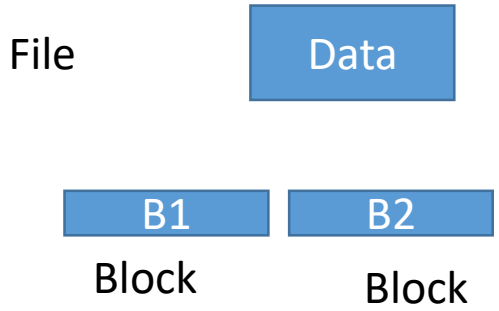


NN NameNode

DN DataNode

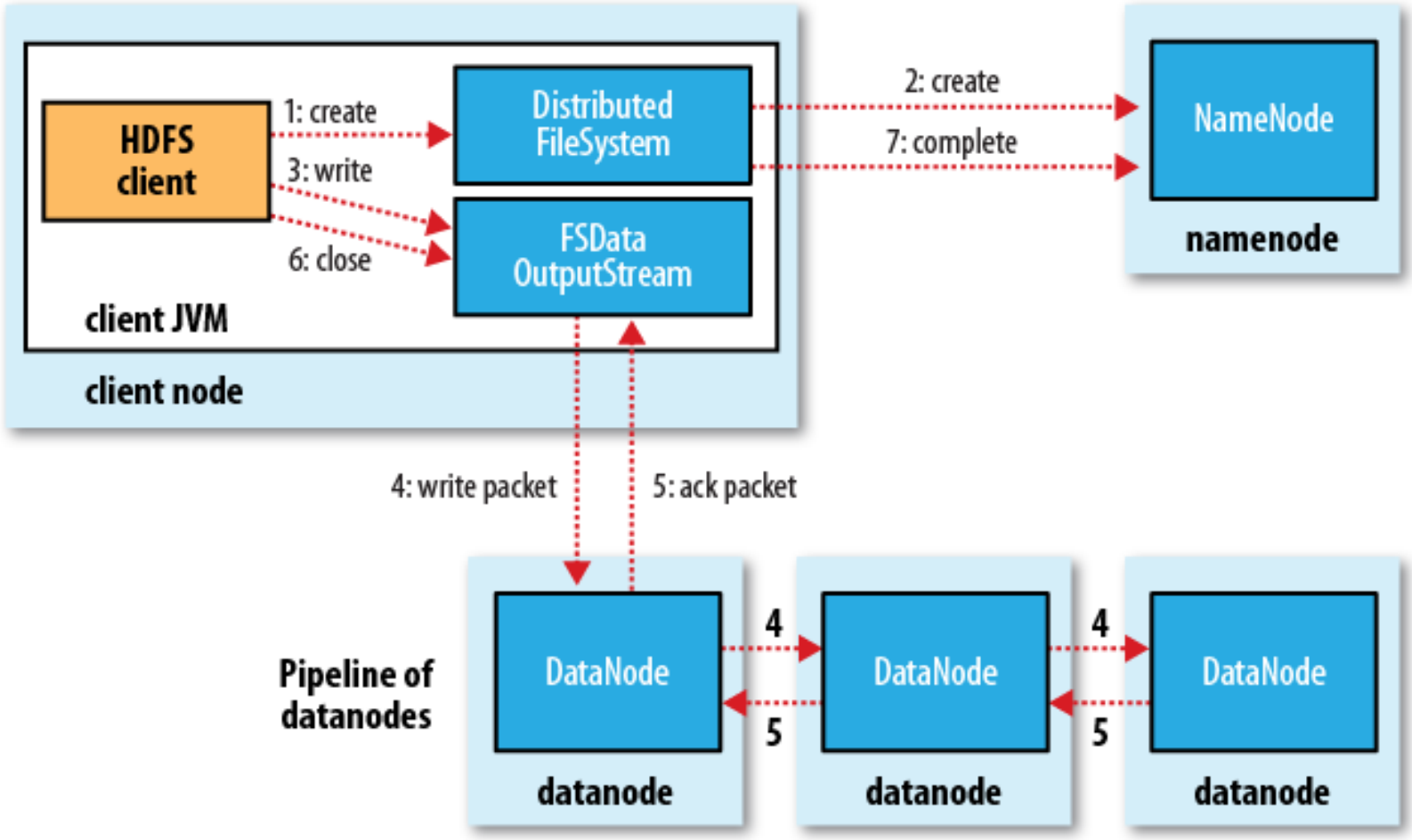
SNN Secondary NameNode

HDFS Components



- *NameNode* — *1 / cluster* — *Manage distribution, replication, maintain metadata*
- *Secondary NameNode* — *1 / cluster* — *Checkpoint Node*
- *DataNode* — *1 / Node* — *Store DataBlocks, Data Integrity*

HDFS Architecture – The Anatomy of a File Write



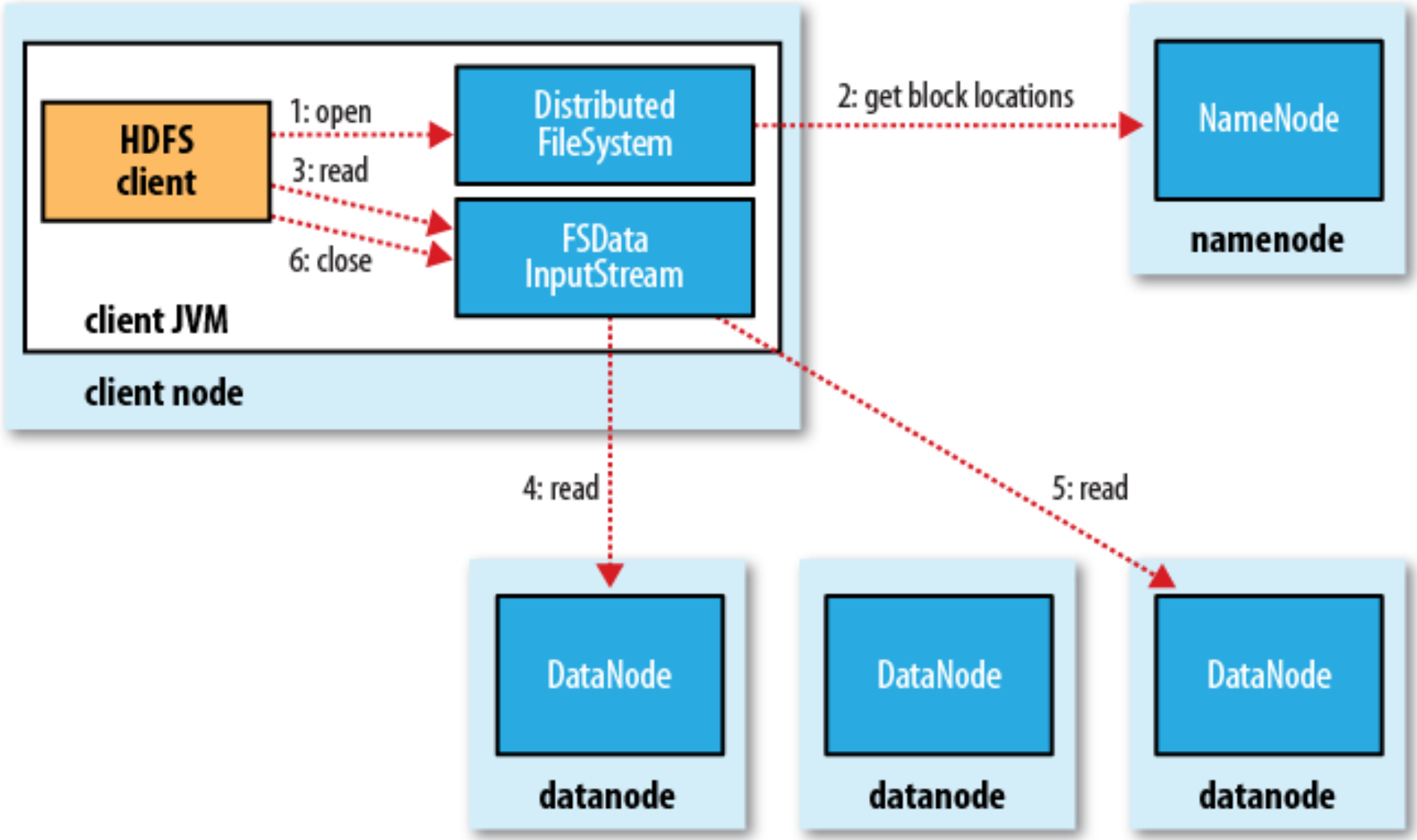
HDFS Architecture – The Anatomy of a File Write

- *Client connects to the NameNode, to create a new file in the HDFS, with no blocks associated with it initially*
- *The namenode makes a record of the new file, and returns an instruction for the client to start writing data*
- *As the client writes data, DFSOutputStream splits it into blocks, which is written to an internal queue initially (called data queue)*
- *The data queue is consumed by the DataStreamer, which is responsible for asking the NameNode to allocate new blocks by picking a list of suitable datanodes to store the replicas*
- *The list of datanodes allocated, forms a "pipeline of datanodes". For the standard replication factor of 3, there will be 3 datanodes in the pipeline*
- *The DataStreamer streams the blocks to the first datanode in the pipeline, and the first datanode stores it in its disk, and forwards it to the second datanode in the pipeline. Similarly, the second datanode stores the block and forwards it to the third (and last) datanode in the pipeline*
- *DFSOutputStream also maintains an internal queue of packets that are waiting to be acknowledged by datanodes, called the ack queue. A packet is removed from the ack queue only when it has been acknowledged by all the datanodes in the pipeline*
- *When the client has finished writing data, it calls close() on the DFSOutputStream. The metadata is committed and the file write is complete*

Rack Awareness

- *Common case where the replication factor is 3, HDFS block placement policy is to*
  - *Place the 1st block on a node in a local rack*
  - *Place the 2nd block on a different node in the same rack*
  - *Place the 3rd block on a different node in a remote rack*

HDFS Architecture – The Anatomy of a File Read



HDFS Architecture – The Anatomy of a File Read

- *The client opens the file it wishes to read by calling open() on the FileSystem object (is an instance of DistributedFileSystem)*
- *DistributedFileSystem calls the namenode, to determine the locations of the blocks for the file*
- *For each block, the namenode returns the addresses of the datanodes that have a copy of that block*
- *The DistributedFileSystem returns an FSDataInputStream (an input stream that supports file seeks) to the client for it to read data from*
- *The client then calls read() on the DFSInputStream*
- *DFSInputStream, then connects to the first datanode for the first block in the file*
- *Blocks are read in a sequence*
- *When the client has finished reading, it calls close() on the FSDataInputStream*

NameNode and SecondaryNameNode

- *The HDFS metadata is stored by the NameNode*
- *The NameNode uses a transaction log called the EditLog to persistently record every change that occurs to file system metadata (EditLog is a file on the OS file system of the machine running NN)*
- *The entire file system metadata, including the mapping of blocks to files and file system properties, is stored in another file called the FsImage (File System Image, also resides OS file system of the machine running NN)*
- *NN has 2 files -> EditLog, FsImage*
- *SNN does checkpointing*
    - *Regular checkpoint interval - 1 hour by default*
    - *Or 1 million transactions on the EditLog*

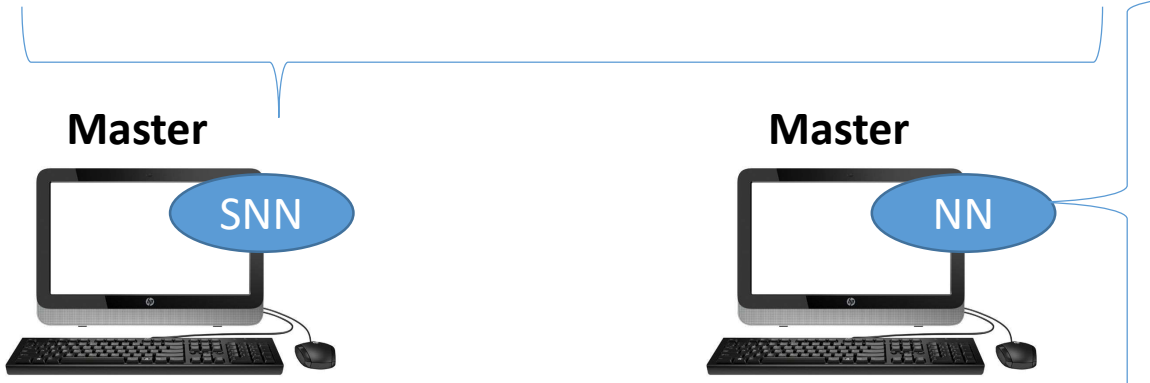
HDFS metadata backup

- *SNN is not a hot backup for the NN*
- *SNN periodically merges **edits in progress** with **FSImage***
  - *Every 1 hour (dfs.namenode.checkpoint.period = 3600) **OR***
  - *1 million txns on the edits_inprogress_xxx file (dfs.namenode.checkpoint.txns = 1000000)*
- ***NN is the Single Point of Failure. To minimize the risk workarounds are possible***
  - *Reduce the checkpoint interval*
  - *Run NN on a better hardware, also have manual copies of the metadata created at regular intervals*

HDFS Daemons - Responsibilities

Master – Slave Architecture

- *Checkpointing*
  - *Merge EditsInProgress with FSImage at regular intervals*



- *Stores metadata of the FileSystem*
- *Allocate DN for a file to be written*
- *Manages Fault Tolerance*
- *The metadata is stored on the local file system of the node running NN*

- *Stores DataBlocks of a file*
- *Sends regular heartbeats to the NN*
- *The DataBlocks reside on the local file system of the node running DN*

Introduction to MapReduce

- *A programming model for distributed datasets*
- *With Hadoop 2.x, MapReduce is YARN-based*
- *Involves 2 phases (Developer) – Map phase and a Reduce phase*
- *The MapReduce framework abstracts the complexity of IO from the developer*
- *MapReduce works on <key, value> pairs – Ex: Hadoop 5 → Hadoop is the key, 5 is the value*
- *Steps involved in MapReduce parallel processing*
  - *Input Split*
  - *Map*
  - *Shuffle & Sort*
  - *Reduce*
  - *Final Output*

Map → Transformation logic
Reduce → Aggregation logic

Understanding MapReduce with an example

Example Application: The MapReduce WordCount program - WordCount is a simple application that counts the number of occurrences of each word in a given input set

Input

/Sample/SampleFile.txt

Welcome to Hadoop
Learning Hadoop is fun
Hadoop Hadoop Hadoop is the buzz

(Key, Value) pairs

WordCount.java

Output

/Sample/WC/part-r-00000

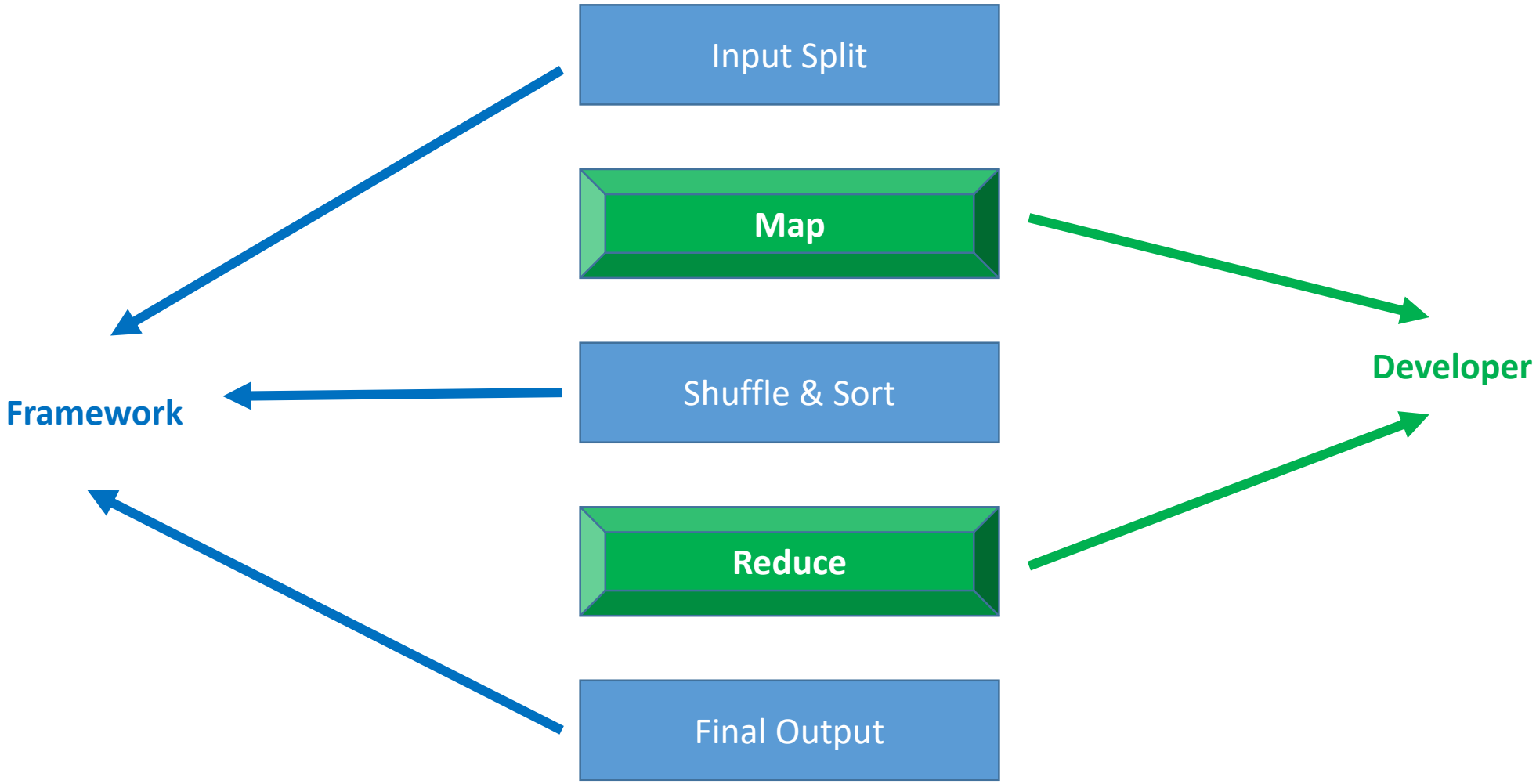
Hadoop	5
Learning	1
Welcome	1
buzz	1
fun	1
is	2
the	1
to	1

Steps in MapReduce

- Input Split
- Map
- Shuffle & Sort
- Reduce
- Final Output

- Map – Transformation → Convert into words
- Reduce – Aggregation → Count words

Steps in MapReduce



Steps in MapReduce – Input Split

- *InputSplit represents the data to be processed by an individual Mapper*
- *Typically InputSplit presents a byte-oriented view of the input, and it is the responsibility of RecordReader to process and present a record-oriented view*
- *RecordReader reads <key, value> pairs from an InputSplit*
- *Typically the RecordReader converts the byte-oriented view of the input, provided by the InputSplit, and presents a record-oriented to the Mapper implementations for processing. RecordReader thus assumes the responsibility of processing record boundaries and presents the tasks with keys and values*

Steps in MapReduce – Map

- *Mapper maps input key/value pairs to a set of intermediate key/value pairs*
- *Maps are the individual tasks that transform input records into intermediate records*
- *The Hadoop MapReduce framework spawns one map task for each InputSplit generated by the InputFormat for the job*
- *All intermediate values associated with a given output key are subsequently grouped by the framework, and passed to the Reducer(s) to determine the final output*
- *The number of maps is equal by the total number of the input splits*

Mapper Code – WordCount program

```
public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
```

Steps in MapReduce – Shuffle & Sort

- *MapReduce makes the guarantee that the input to the reducer is sorted by key*
- *The process by which the system performs the sort, and transfers the map outputs to the reducers as inputs, is known as the shuffle*

Steps in MapReduce – Reduce

- *In this phase the reduce() method is called for each key*
- *The list of value pairs / key are in the grouped inputs*
- *The output of the reduce task is typically written to the FileSystem via Context.write*
- *The output of the Reducer is not sorted*

Reducer Code – WordCount program

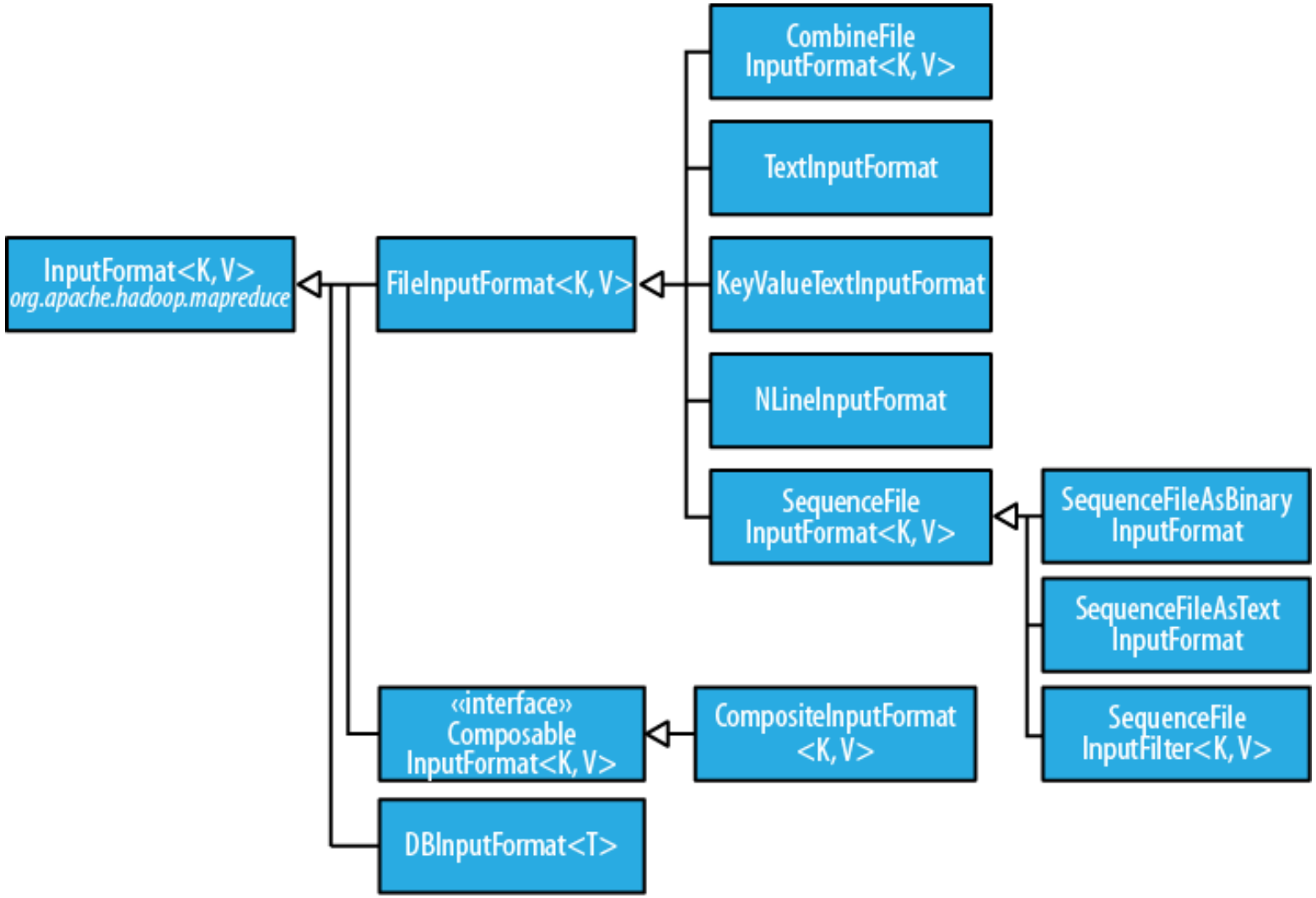
```
public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }
```

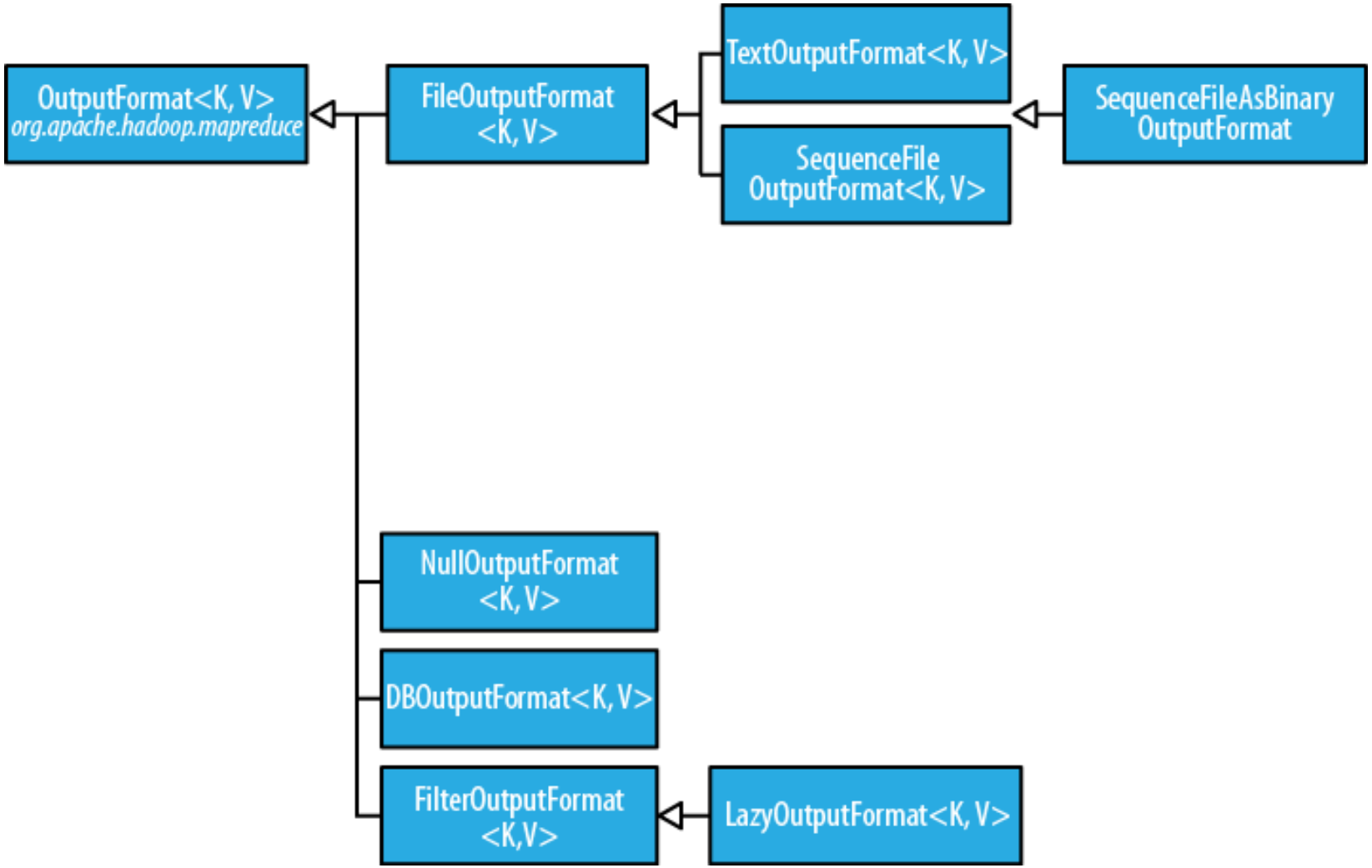
The main() class

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

MapReduce Input Formats



MapReduce Output Formats



MapReduce Datatypes

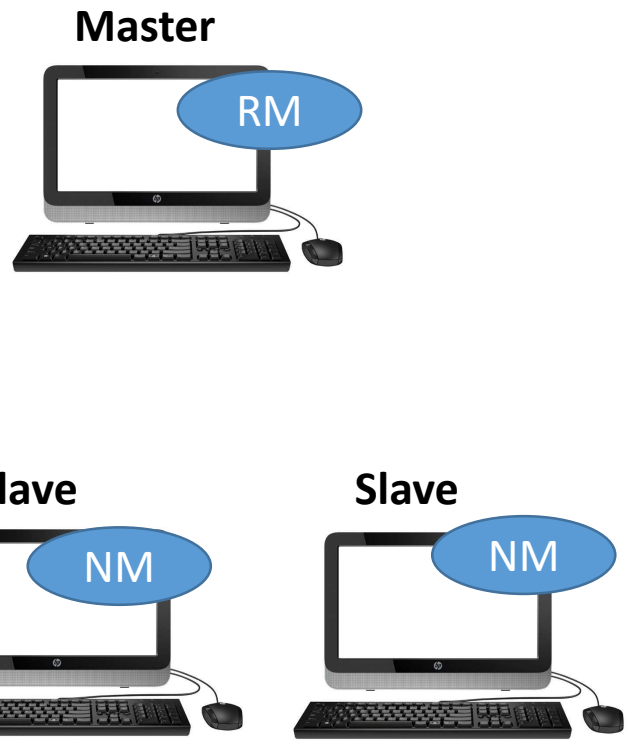
Java	MapReduce
int	IntWritable
float	FloatWritable
long	LongWritable
double	DoubleWritable
null	NullWritable
byte	BytesWritable
String	Text

WordCount execution steps

- **Terminal 1**
  - `$ sudo watch jps`
- **Terminal 2**
  - `$ yarn jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar wordcount /Sample/SampleFile.txt /Sample/WC`
- **Once the program completed, check the output**
  - `$ hadoop fs -cat /Sample/WC/part-r-00000`

YARN Daemons

Master – Slave Architecture



RM ResourceManager

NM NodeManager

YARN Components

Application | Job

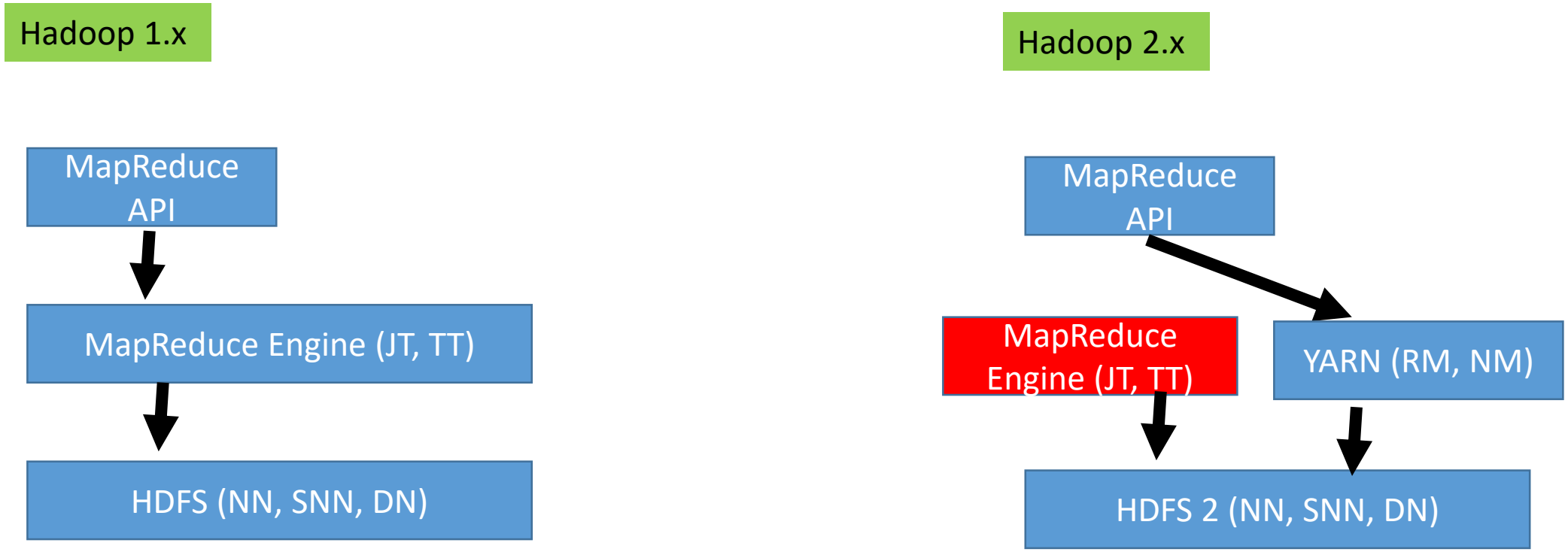
- *Resource Manager* — *1 / cluster* — *Scheduling + allocation of compute resources*
- *Application Master* — *1 / job* — *Monitoring*
- *Node Manager* — *1 / DN* — *Monitor containers on the node*
- *YarnChild*

Compute Resources on NM also termed as "Resource Containers" = (CPU + RAM)

Execution of tasks

Map Task | Reduce Task

Hadoop 1.x Vs Hadoop 2.x



Hadoop 1.x Vs Hadoop 2.x

Hadoop 1.x

- *Scalability 4000+ nodes*
- *No High Availability on masters (Ex. NN, JT)*
- *JobTracker over burdened – Scheduling + Monitoring*
- *Fixed slots for task execution*

Hadoop 2.x

- *Scalability 25000+ nodes*
- *High Availability on masters (Ex. NN, RM) can be setup*
- *RM for scheduling + ApplicationMaster for monitoring*
- *Containers are dynamic*