Big Data and Hadoop

What is Big Data

"Big data" is a field that treats ways to analyze, systematically extract information from, or otherwise deal with data sets that are too large or complex to be dealt with by traditional data-processing application software

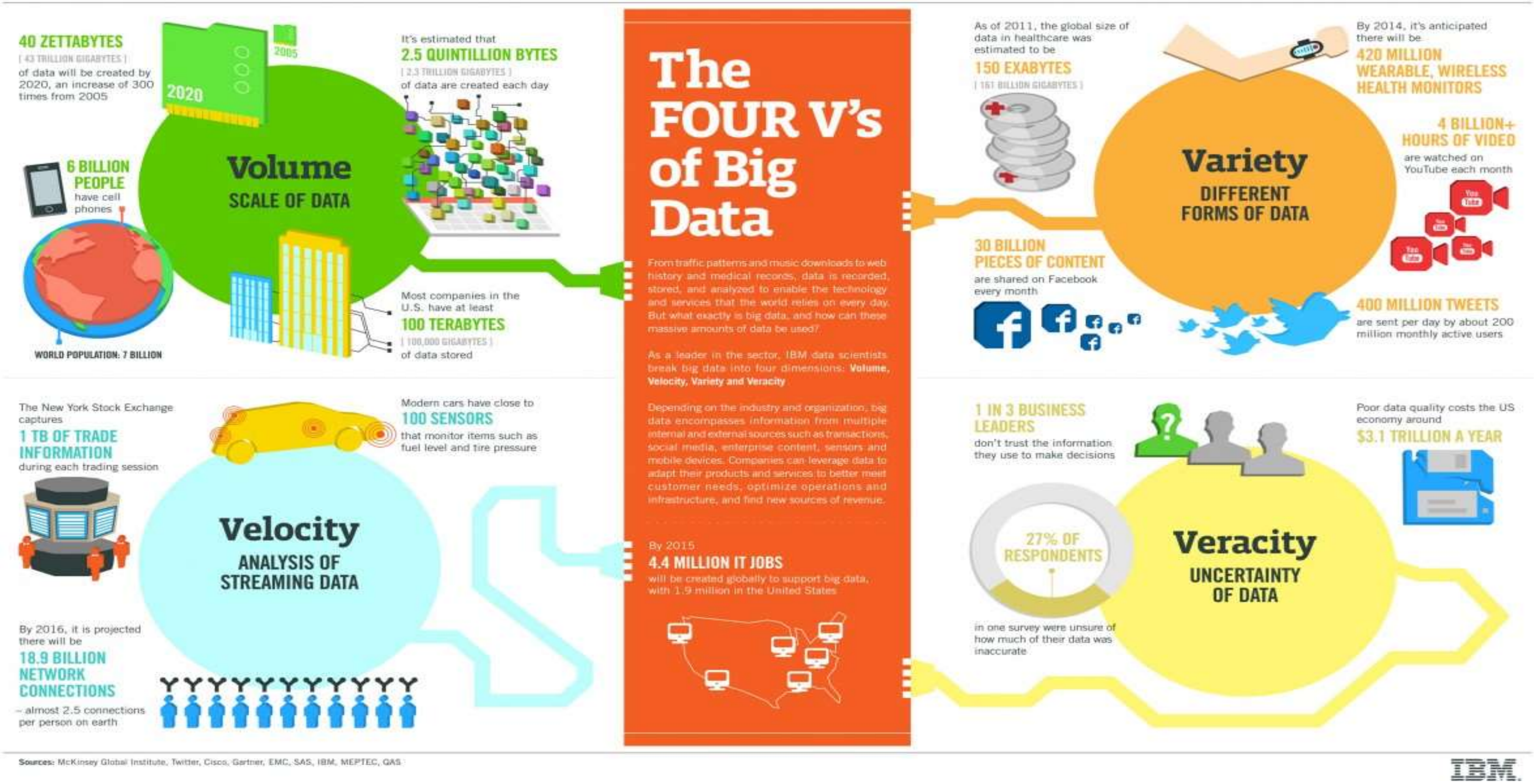
-- Wiki

Big Data is a term used to describe a collection of data that is huge in size and yet growing exponentially with time.

Difference Between RDBMS and Hadoop and

Feature	RDBMS	Hadoop
Data Variety	Mainly for Structured data.	Used for Structured, Semi-Structured and Unstructured data
Data Storage	Average size data (GBS)	Use for large data set (Tbs and Pbs)
Querying	SQL Language	HQL (Hive Query Language)
Schema	Required on write (static schema)	Required on reading (dynamic schema)
Speed	Reads are fast	Both reads and writes are fast
Cost	License	Free
Use Case	OLTP (Online transaction processing)	Analytics (Audio, video, logs etc), Data Discovery
Data Objects	Works on Relational Tables	Works on Key/Value Pair
Throughput	Low	High
Scalability	Vertical	Horizontal
Hardware Profile	High-End Servers	Commodity/Utility Hardware
Integrity	High (ACID)	Low

Attributes of Big Data



The FOUR V's of Big Data

From traffic patterns and music downloads to web history and medical records, data is recorded, stored, and analyzed to enable the technology and services that the world relies on every day. But what exactly is big data, and how can these massive amounts of data be used?

As a leader in the sector, IBM data scientists break big data into four dimensions: **Volume, Velocity, Variety and Veracity**

Depending on the industry and organization, big data encompasses information from multiple internal and external sources such as transactions, social media, enterprise content, sensors and mobile devices. Companies can leverage data to adapt their products and services to better meet customer needs, optimize operations and infrastructure, and find new sources of revenue.

By 2015
4.4 MILLION IT JOBS
will be created globally to support big data, with 1.9 million in the United States

Volume
SCALE OF DATA

40 ZETTABYTES
[43 TRILLION GIGABYTES]
of data will be created by 2020, an increase of 300 times from 2005

6 BILLION PEOPLE
have cell phones

WORLD POPULATION: 7 BILLION

It's estimated that
2.5 QUINTILLION BYTES
[2.3 TRILLION GIGABYTES]
of data are created each day

Most companies in the U.S. have at least
100 TERABYTES
[100,000 GIGABYTES]
of data stored

Velocity
ANALYSIS OF STREAMING DATA

The New York Stock Exchange captures
1 TB OF TRADE INFORMATION
during each trading session

Modern cars have close to
100 SENSORS
that monitor items such as fuel level and tire pressure

By 2016, it is projected there will be
18.9 BILLION NETWORK CONNECTIONS
– almost 2.5 connections per person on earth

Variety
DIFFERENT FORMS OF DATA

As of 2011, the global size of data in healthcare was estimated to be
150 EXABYTES
[161 BILLION GIGABYTES]

30 BILLION PIECES OF CONTENT
are shared on Facebook every month

By 2014, it's anticipated there will be
420 MILLION WEARABLE, WIRELESS HEALTH MONITORS

4 BILLION+ HOURS OF VIDEO
are watched on YouTube each month

400 MILLION TWEETS
are sent per day by about 200 million monthly active users

Veracity
UNCERTAINTY OF DATA

1 IN 3 BUSINESS LEADERS
don't trust the information they use to make decisions

27% OF RESPONDENTS
in one survey were unsure of how much of their data was inaccurate

Poor data quality costs the US economy around
$3.1 TRILLION A YEAR

Sources: McKinsey Global Institute, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS

Applications of Big Data

NetApp™

NetApp Improves Customer Support by Deploying Cloudera Enterprise

Home > Government IT

Opinion

Barack Obama's Big Data won the US election

Datameer®

Joint Success Story: Major Retail Bank

CASE STUDY
Intel® Xeon® Processors
Intel® Distribution for Apache Hadoop* Software

intel™

China Mobile Guangdong Gives Subscribers Real-Time Access to Billing and Call Data Records

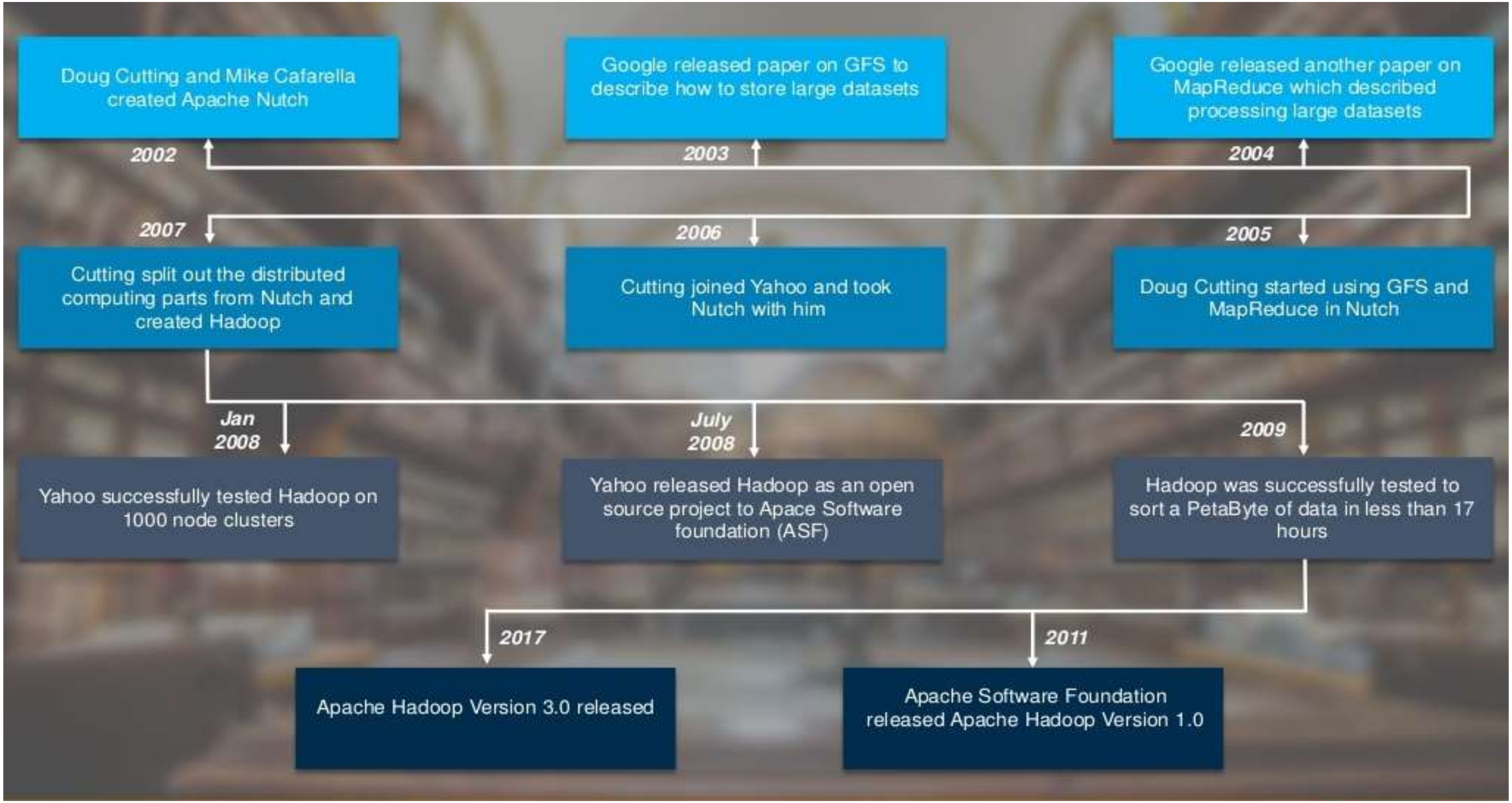
Streamlining Healthcare Connectivity with Big Data

What is Hadoop

- Big Data is a *Business Term* and *Hadoop* is the framework for implementation

- Hadoop
  - is an open-source software framework
  - used for distributed storage and
  - distributed processing of dataset of big data using the MapReduce programming model.
  - It consists of computer clusters built from commodity hardware

History of Hadoop



2002

Doug Cutting and Mike Cafarella created Apache Nutch

2003

Google released paper on GFS to describe how to store large datasets

2004

Google released another paper on MapReduce which described processing large datasets

2005

Doug Cutting started using GFS and MapReduce in Nutch

2006

Cutting joined Yahoo and took Nutch with him

2007

Cutting split out the distributed computing parts from Nutch and created Hadoop

Jan 2008

Yahoo successfully tested Hadoop on 1000 node clusters

July 2008

Yahoo released Hadoop as an open source project to Apace Software foundation (ASF)

2009

Hadoop was successfully tested to sort a PetaByte of data in less than 17 hours

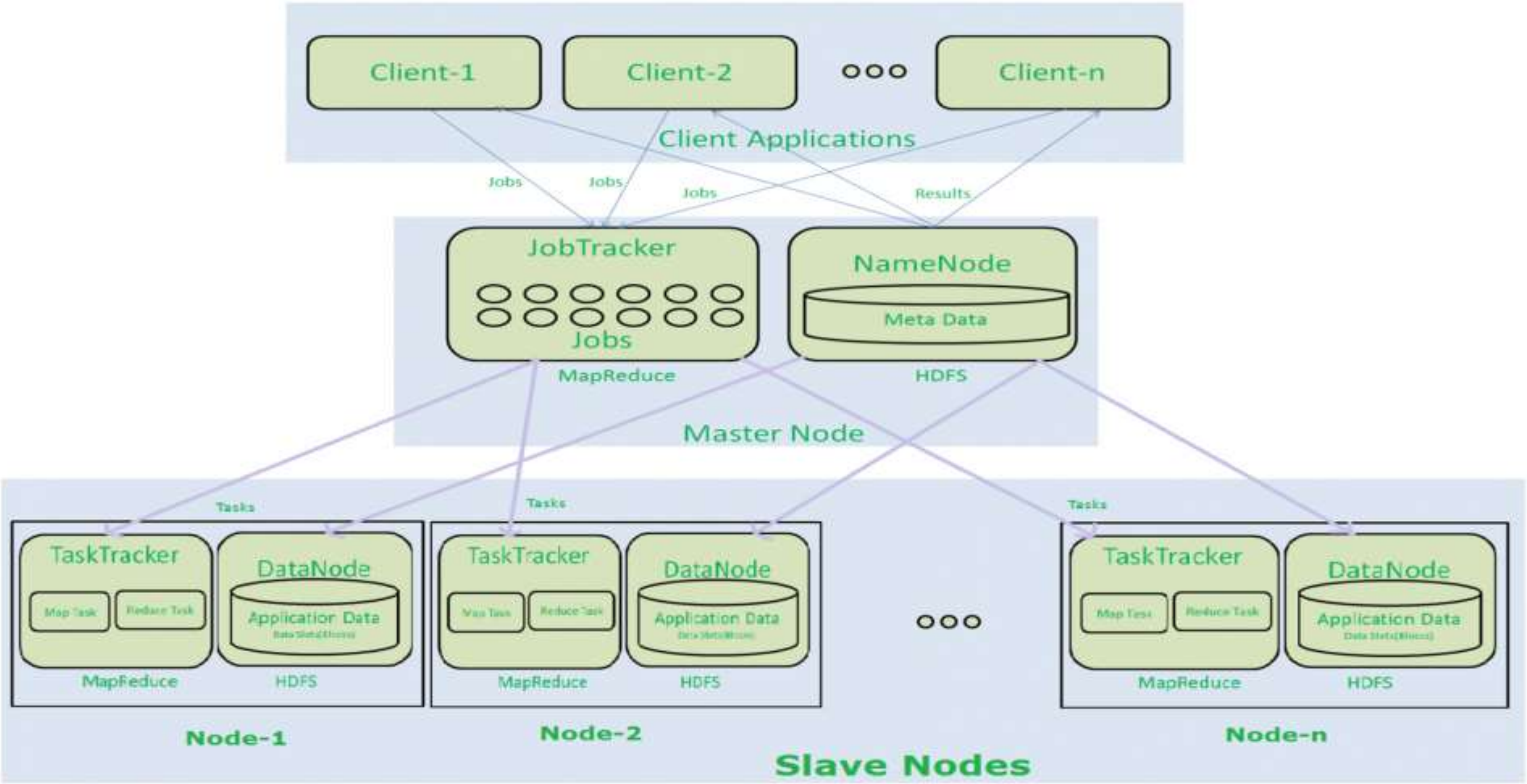
2011

Apache Software Foundation released Apache Hadoop Version 1.0

2017

Apache Hadoop Version 3.0 released

Hadoop 1.X Architecture



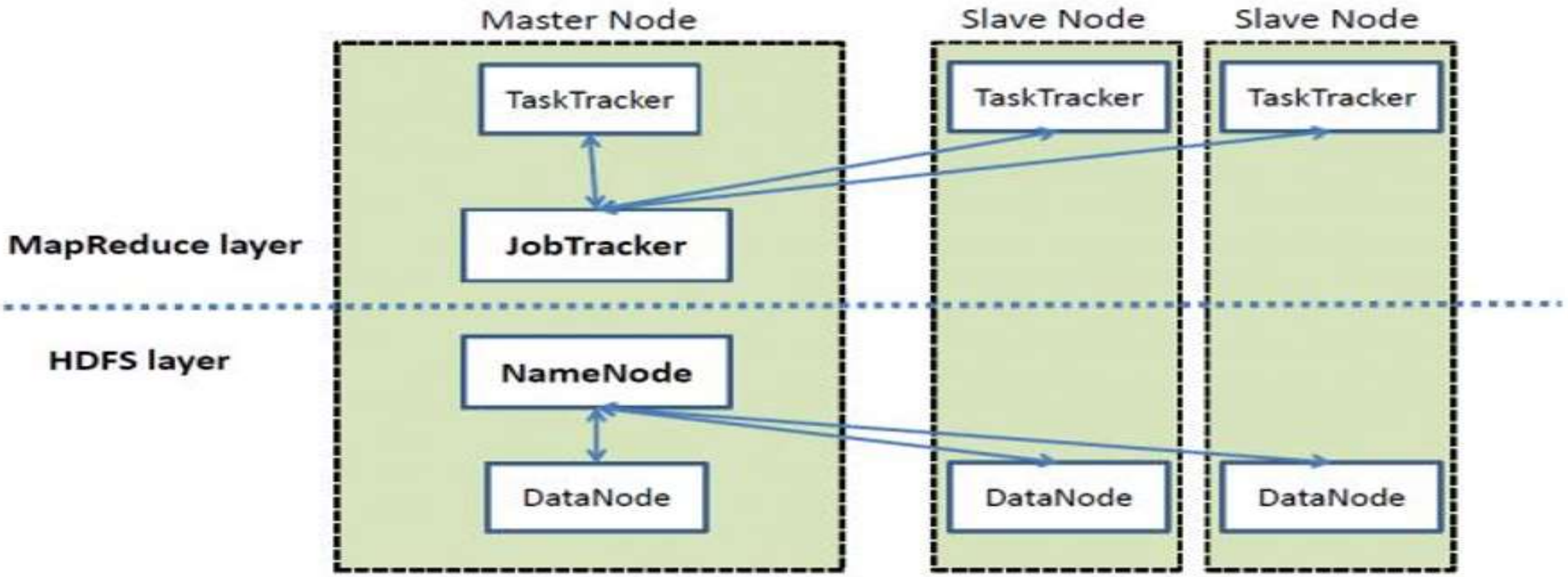
Hadoop 1.x Architecture

Hadoop Distributors

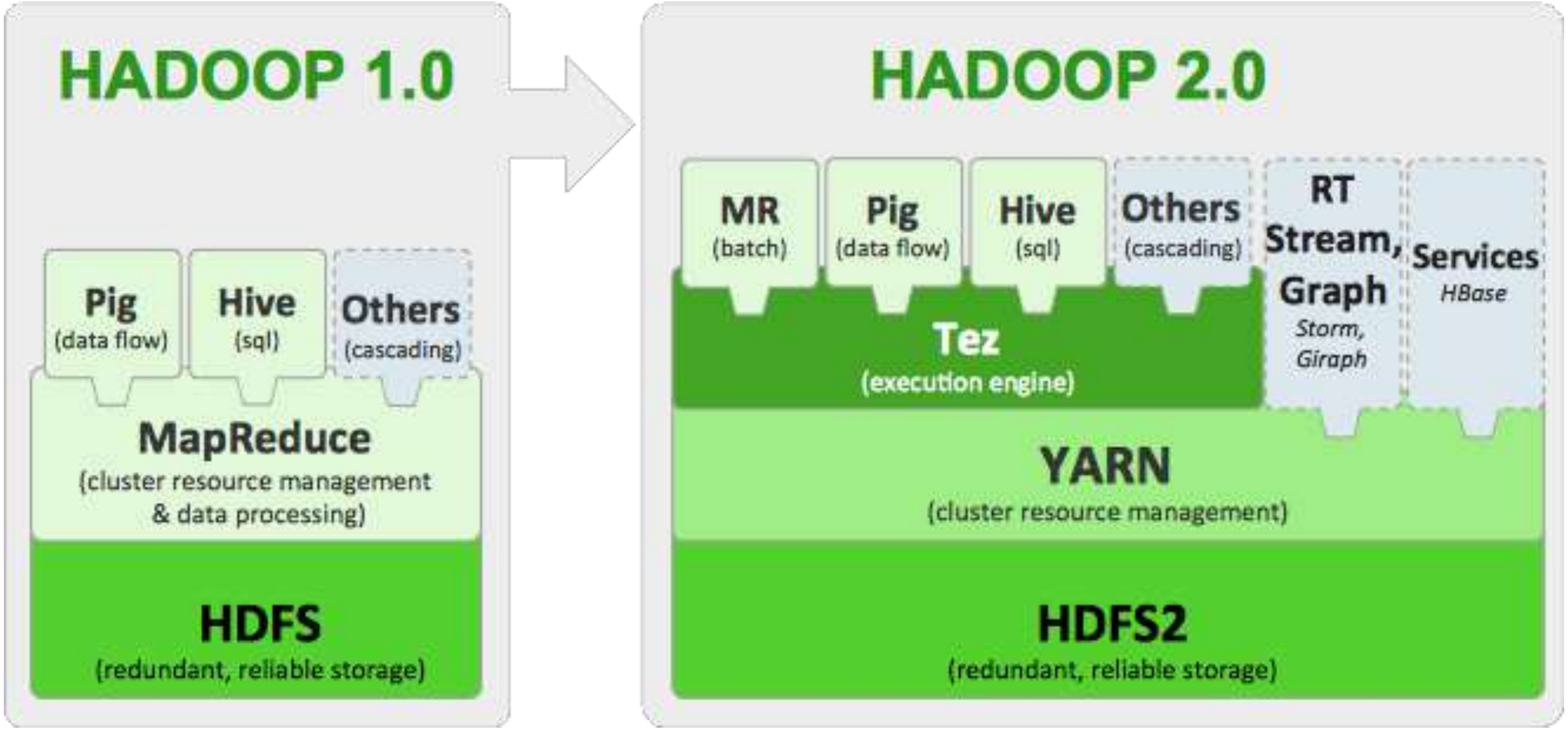
Difference between Hadoop 1 and Hadp2

Hadoop1	Hadoop2
Supports MapReduce (MR) processing model only. Does not support non-MR tools	Allows to work in MR as well as other distributed computing models like Spark, Hama, Giraph, Message Passing Interface) MPI & HBase coprocessors.
MR does both processing and cluster-resource management.	YARN (Yet Another Resource Negotiator) does cluster resource management and processing is done using different processing models.
Has limited scaling of nodes. Limited to 4000 nodes per cluster	Has better scalability. Scalable up to 10000 nodes per cluster
Works on concepts of slots – slots can run either a Map task or a Reduce task only.	Works on concepts of containers. Using containers can run generic tasks.
A single Namenode to manage the entire namespace.	Multiple Namenode servers manage multiple namespaces.
Has Single-Point-of-Failure (SPOF) – because of single Namenode- and in the case of Namenode failure, needs manual intervention to overcome.	Has to feature to overcome SPOF with a standby Namenode and in the case of Namenode failure, it is configured for automatic recovery.
Does not support Microsoft Windows	Added support for Microsoft windows

More on Master Slave



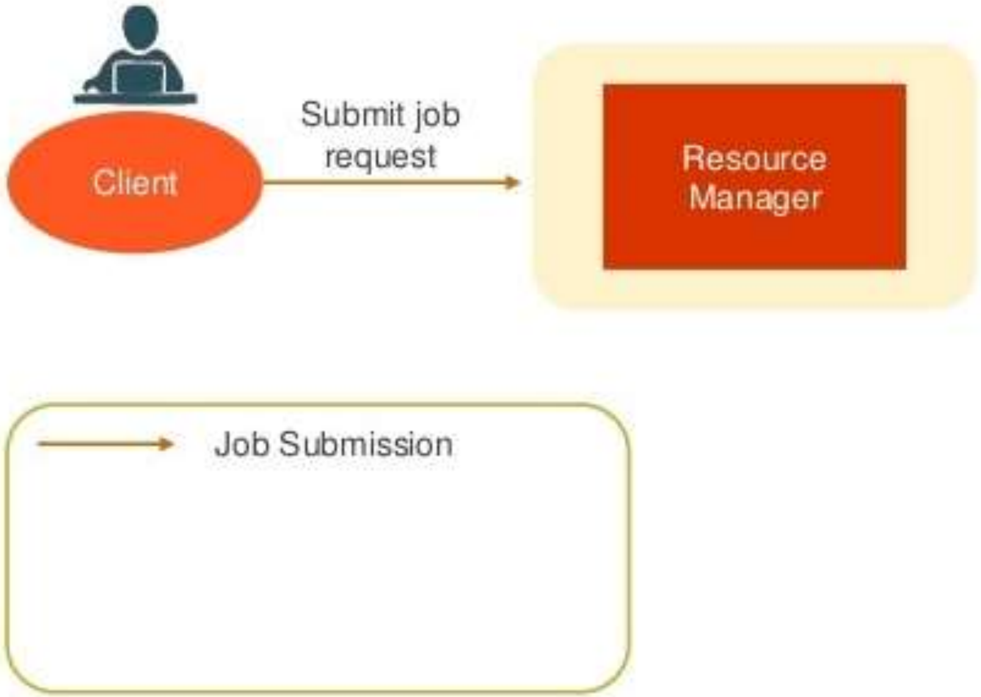
Hadoop 2.X Architecture



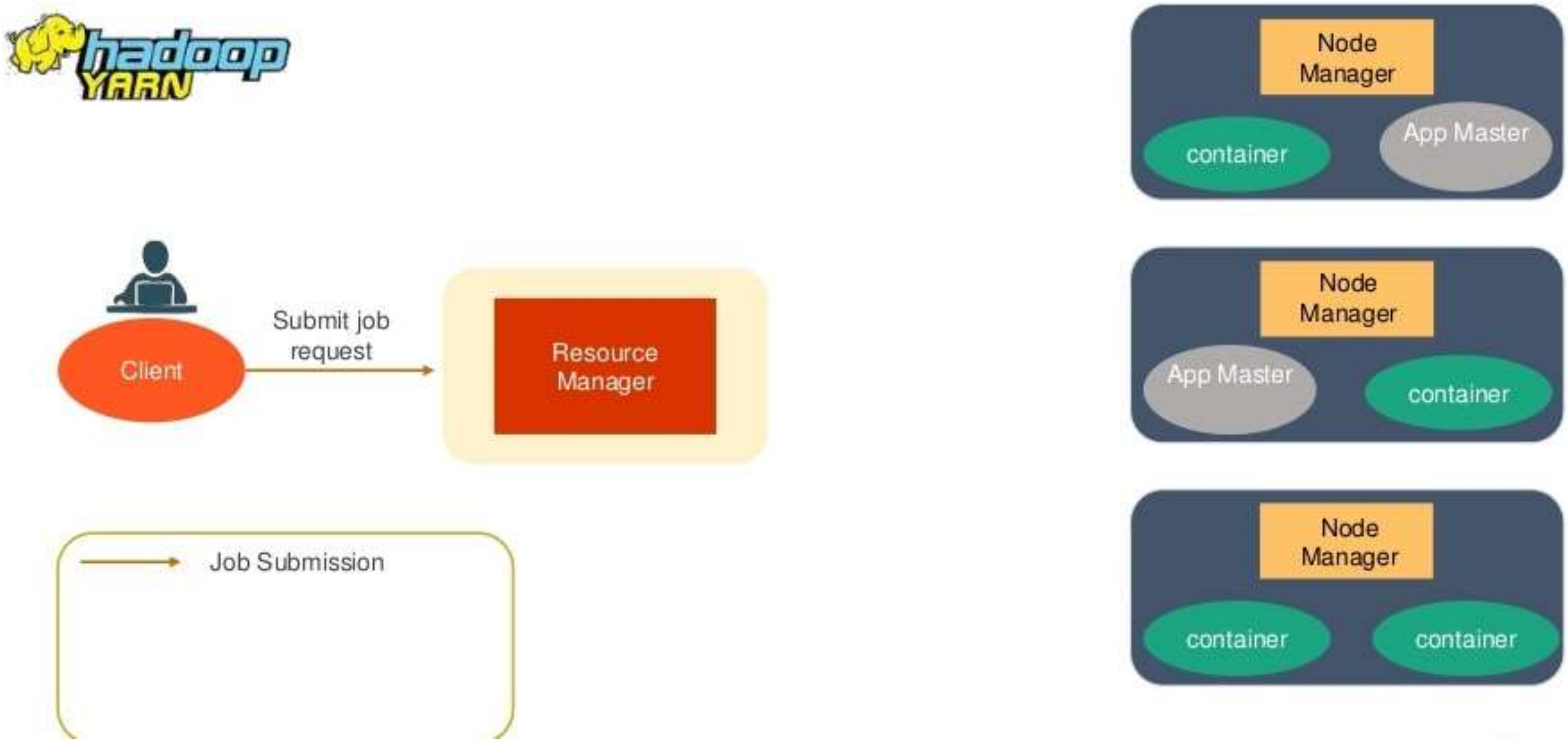
Hadoop 2.X Architecture Components

- **NameNode(HA) aka HDFS High Availability:** In Hadoop 1.0 NameNode was the single point of failure in a Cluster, resulting in data loss in case of a NameNode failure. Hadoop 2.0 Architecture supports multiple NameNodes to remove this bottleneck by using Standby NameNode.
- **HDFS :** enables support for multiple namespaces in the cluster to improve scalability and isolation
- **YARN(Yet Another Resource Negotiator) aka NextGen (MRv2):** This is next generation processing framework.

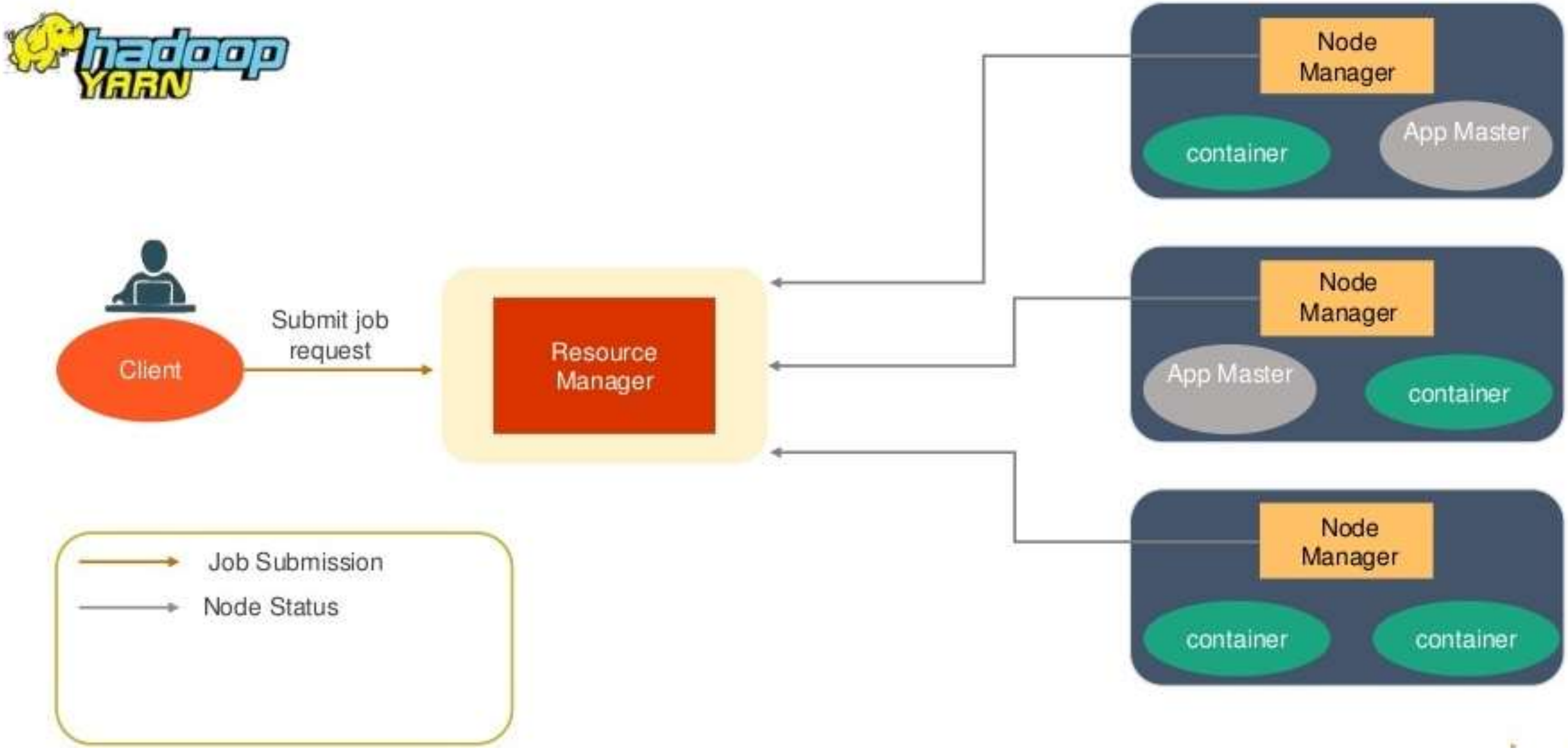
YARN Architecture



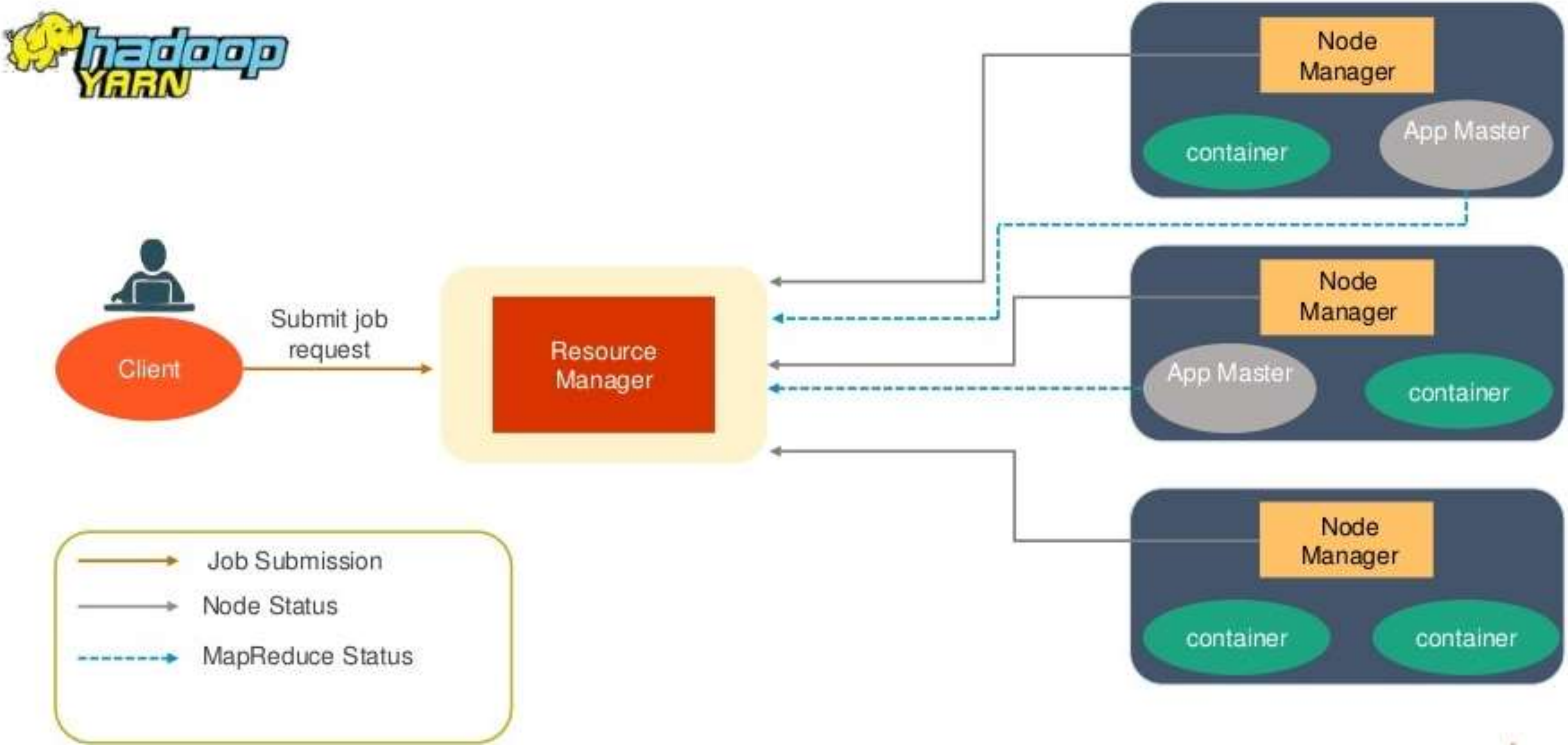
YARN Architecture



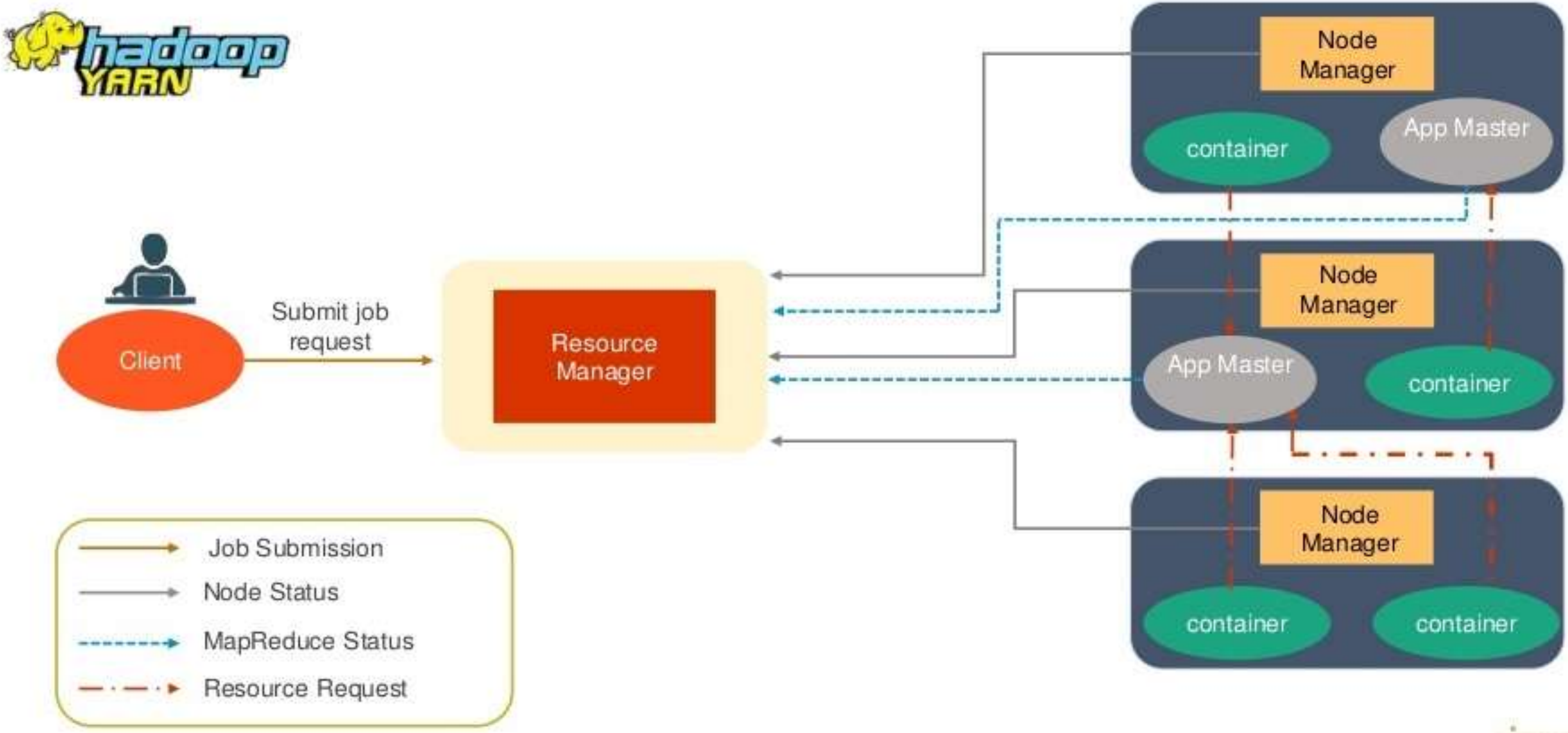
YARN Architecture



YARN Architecture



YARN Architecture



Client

Submit job request

Resource Manager

Node Manager

container

App Master

Node Manager

App Master

container

Node Manager

container

container

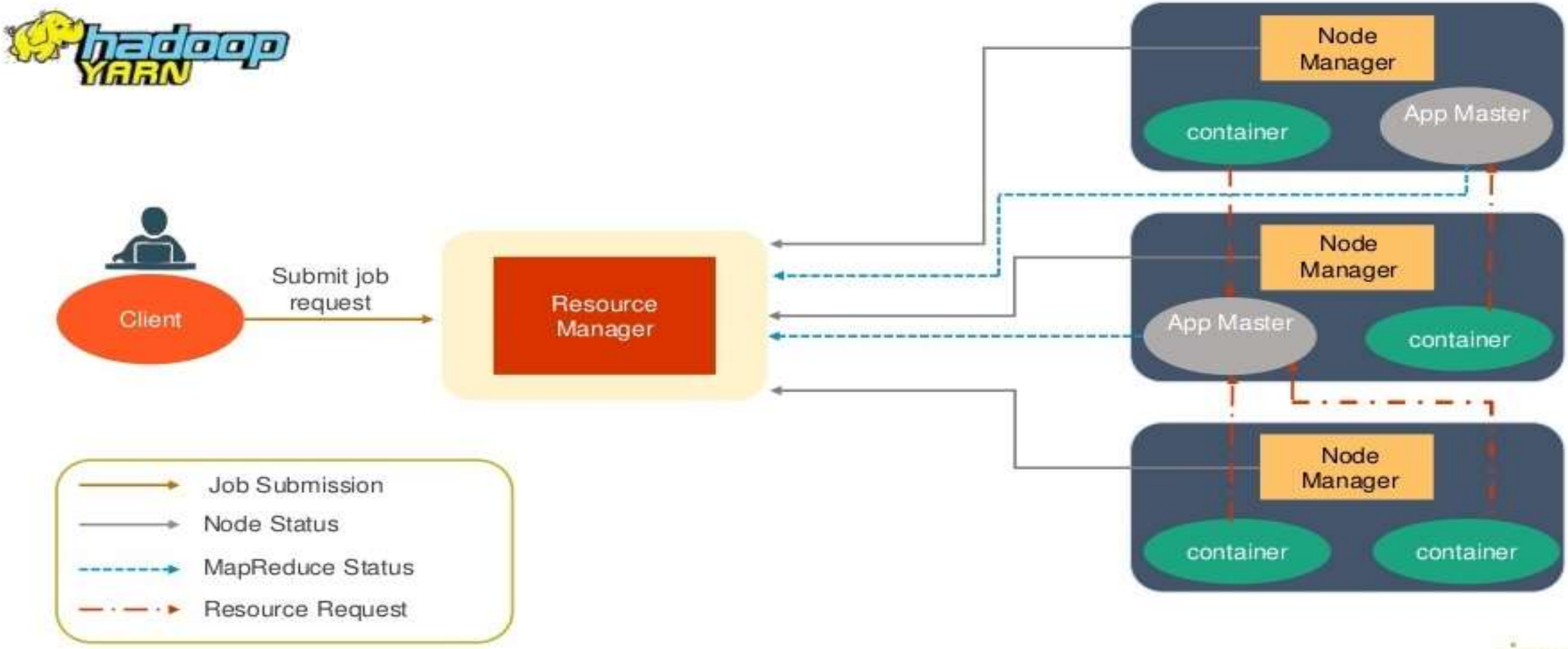
Job Submission

Node Status

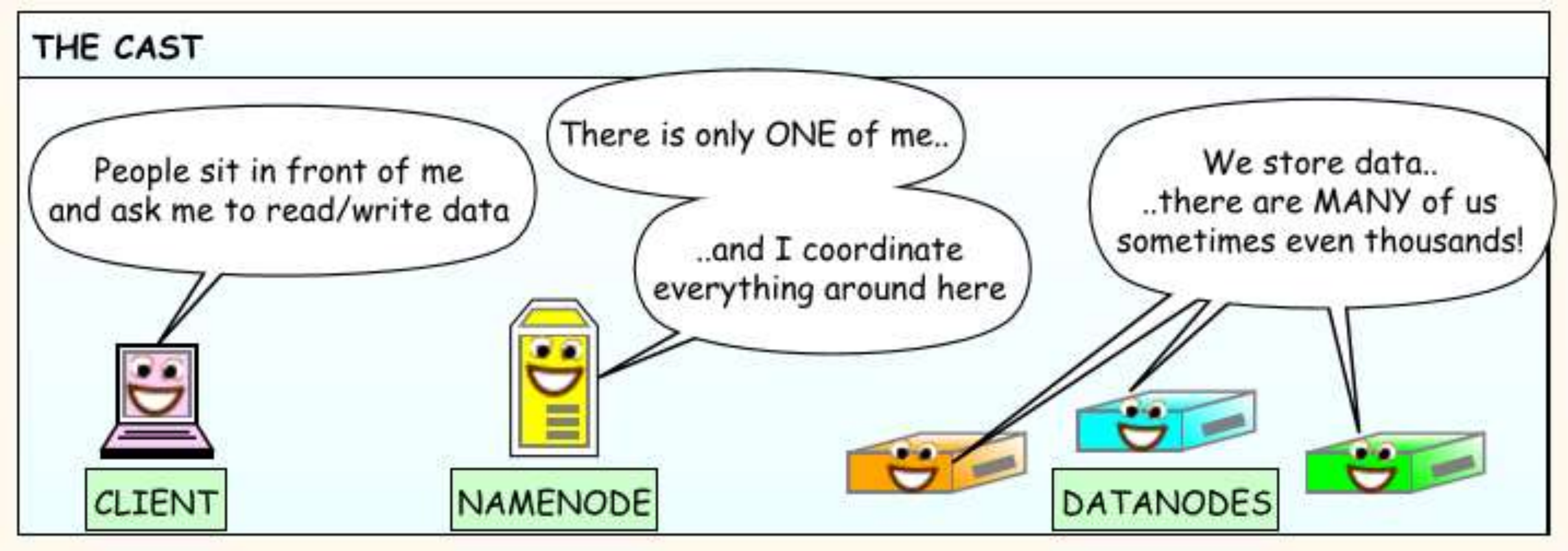
MapReduce Status

Resource Request

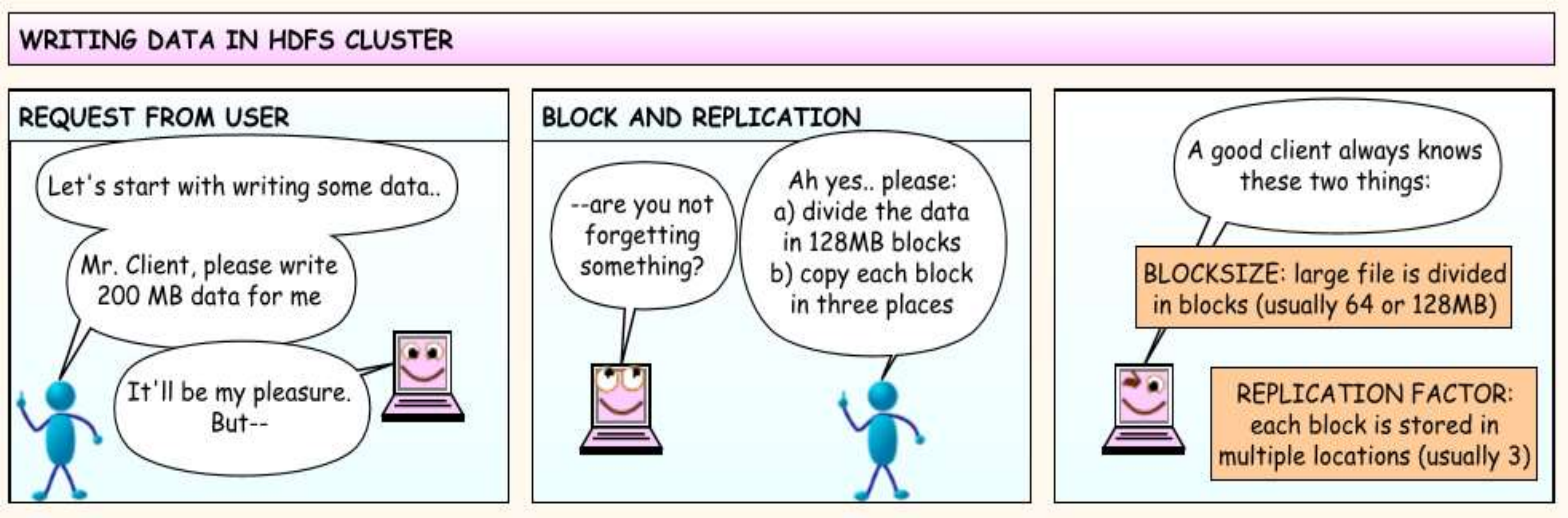
Hadoop Job Flow

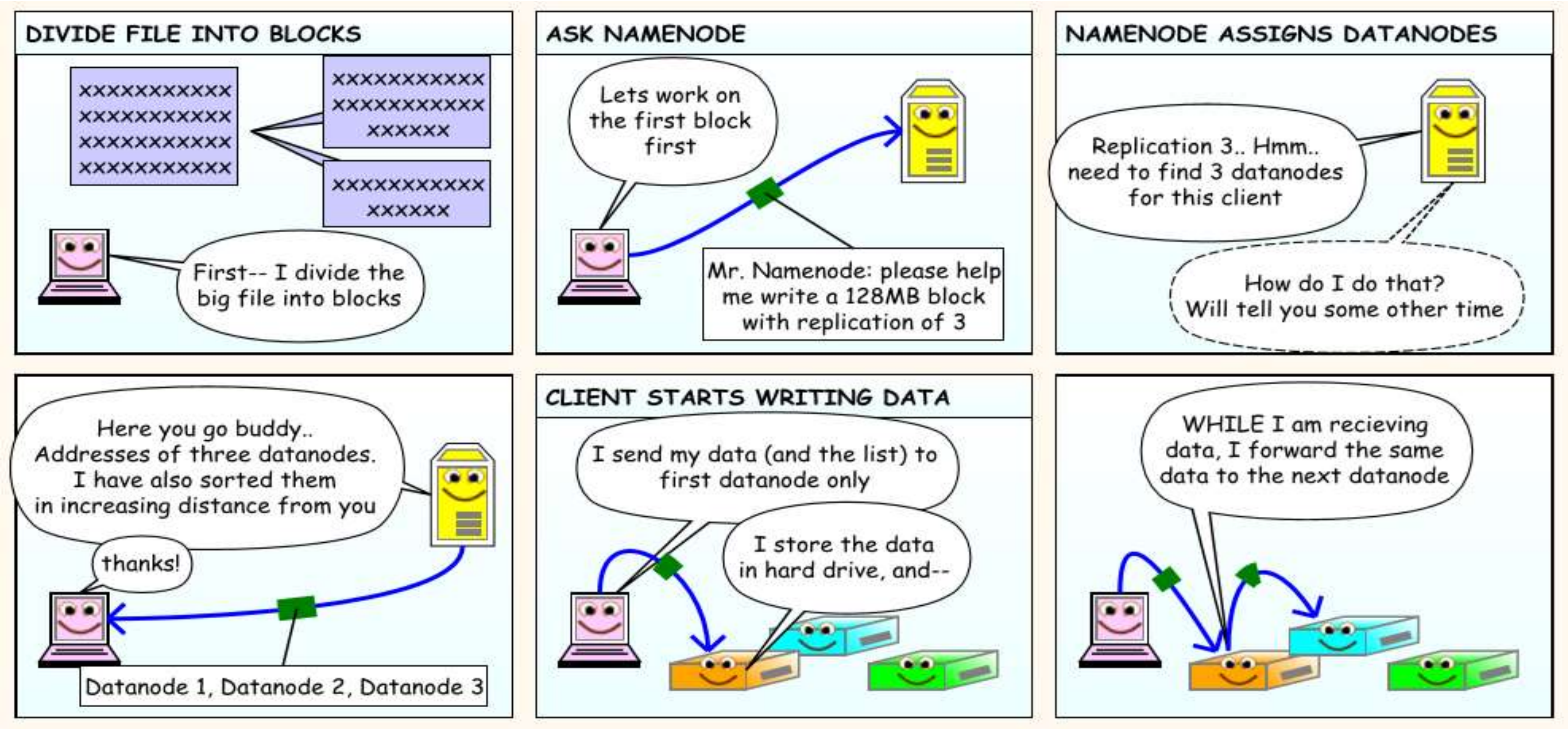


Hadoop Write Architecture

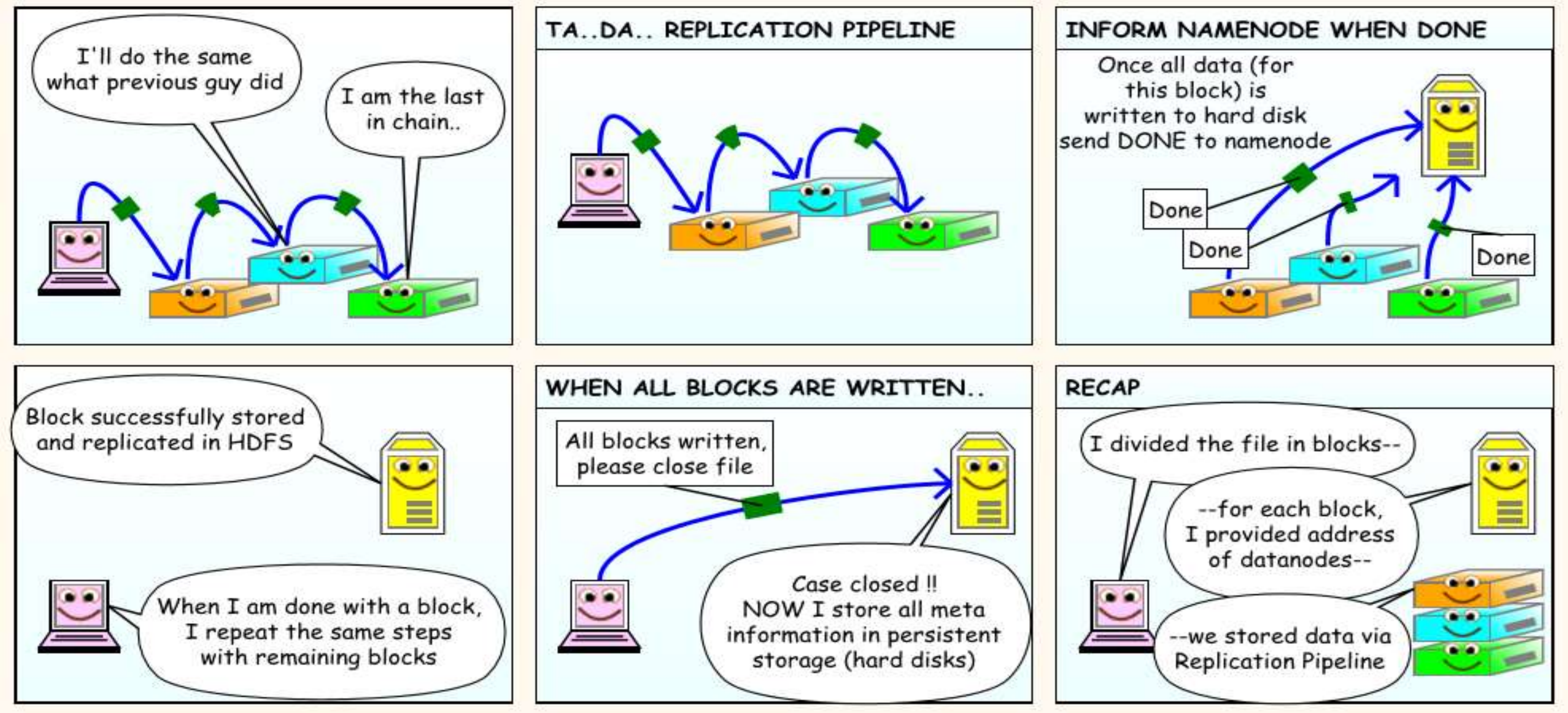

THE CAST
People sit in front of me and ask me to read/write data
There is only ONE of me..
..and I coordinate everything around here
We store data.. ..there are MANY of us sometimes even thousands!
CLIENT
NAMENODE
DATANODES

Hadoop Write Architecture

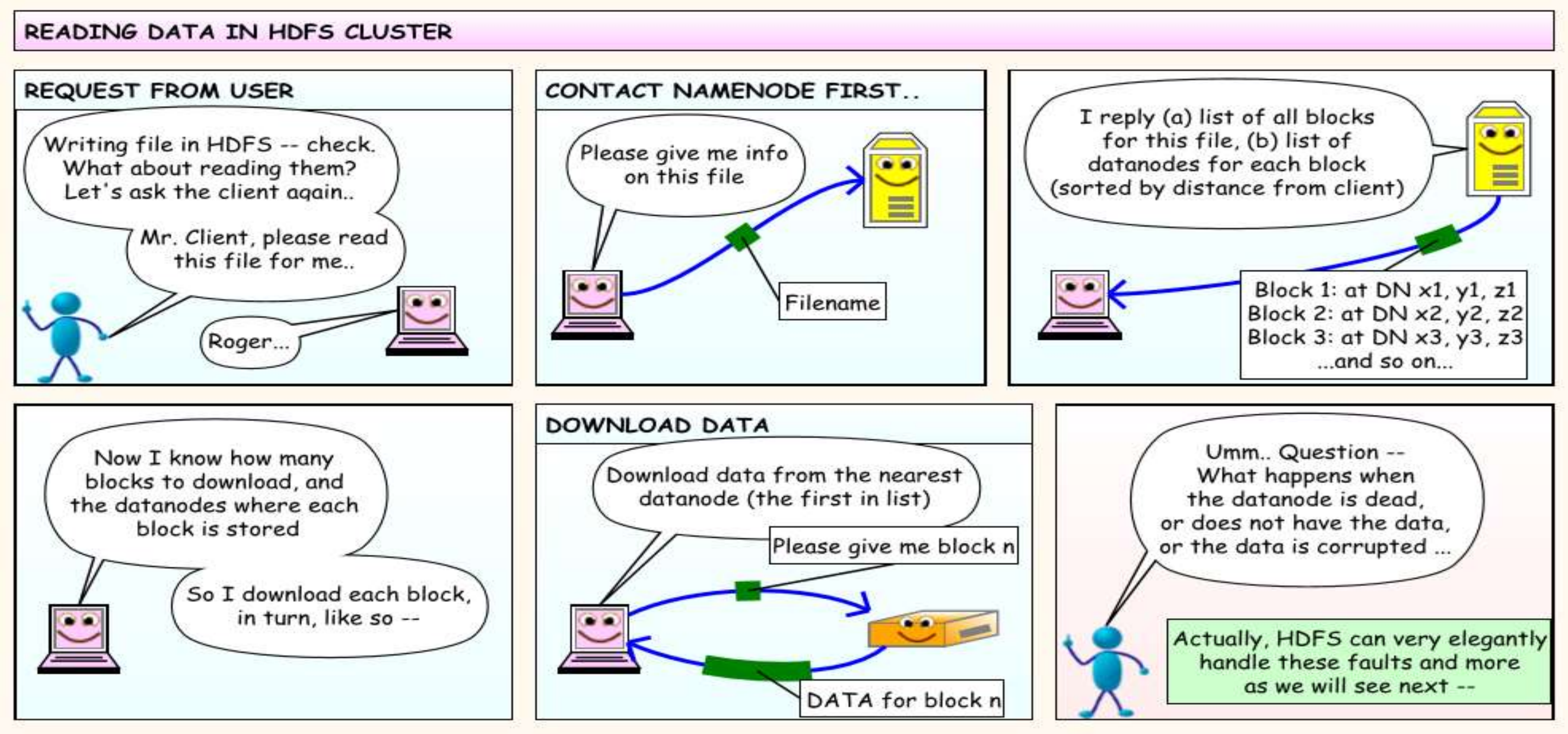

WRITING DATA IN HDFS CLUSTER
REQUEST FROM USER
Let's start with writing some data..
Mr. Client, please write 200 MB data for me
It'll be my pleasure. But--
BLOCK AND REPLICATION
--are you not forgetting something?
Ah yes.. please: a) divide the data in 128MB blocks b) copy each block in three places
A good client always knows these two things:
BLOCKSIZE: large file is divided in blocks (usually 64 or 128MB)
REPLICATION FACTOR: each block is stored in multiple locations (usually 3)


DIVIDE FILE INTO BLOCKS
xxxxxxxxxxx xxxxxxxxxxx xxxxxxxxxxx xxxxxxxxxxx
xxxxxxxxxxx xxxxxxxxxxx xxxxxx
xxxxxxxxxxx xxxxxx
First-- I divide the big file into blocks
ASK NAMENODE
Lets work on the first block first
Mr. Namenode: please help me write a 128MB block with replication of 3
NAMENODE ASSIGNS DATANODES
Replication 3.. Hmm.. need to find 3 datanodes for this client
How do I do that? Will tell you some other time
Here you go buddy.. Addresses of three datanodes. I have also sorted them in increasing distance from you
thanks!
Datanode 1, Datanode 2, Datanode 3
CLIENT STARTS WRITING DATA
I send my data (and the list) to first datanode only
I store the data in hard drive, and--
WHILE I am recieving data, I forward the same data to the next datanode

Hadoop Write Architecture

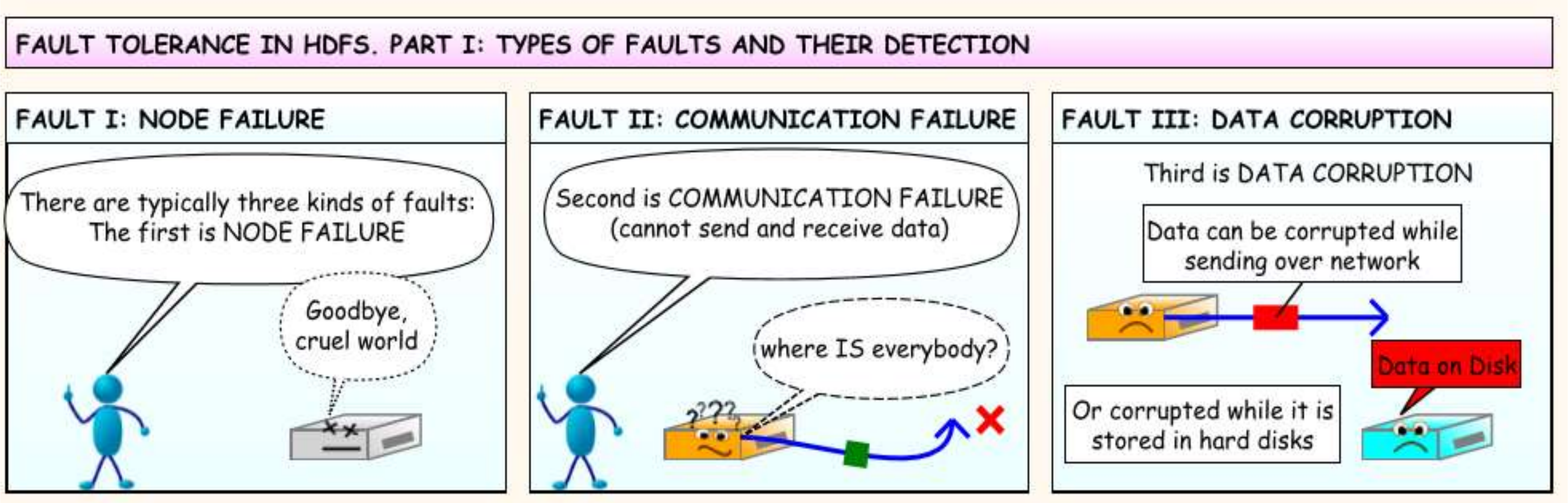

I'll do the same what previous guy did
I am the last in chain..
TA..DA.. REPLICATION PIPELINE
INFORM NAMENODE WHEN DONE
Once all data (for this block) is written to hard disk send DONE to namenode
Done
Done
Done
Block successfully stored and replicated in HDFS
When I am done with a block, I repeat the same steps with remaining blocks
WHEN ALL BLOCKS ARE WRITTEN..
All blocks written, please close file
Case closed !! NOW I store all meta information in persistent storage (hard disks)
RECAP
I divided the file in blocks--
--for each block, I provided address of datanodes--
--we stored data via Replication Pipeline

Hadoop Write Architecture

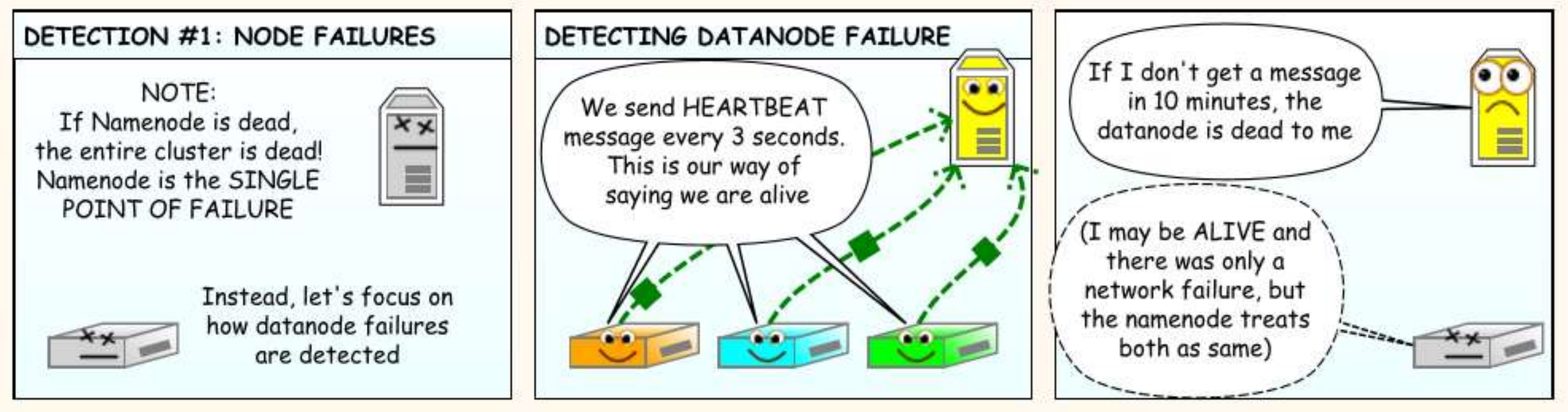


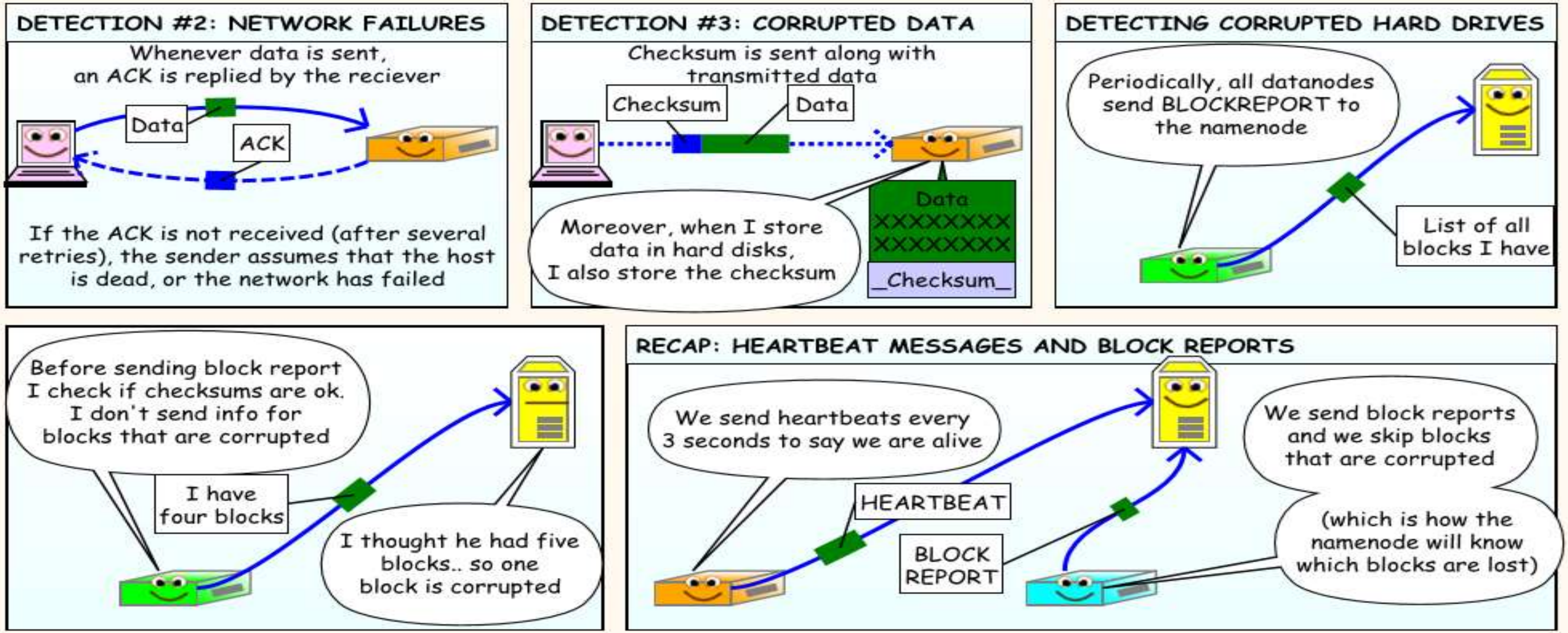
READING DATA IN HDFS CLUSTER
REQUEST FROM USER
Writing file in HDFS -- check. What about reading them? Let's ask the client again..
Mr. Client, please read this file for me..
Roger...
CONTACT NAMENODE FIRST..
Please give me info on this file
Filename
I reply (a) list of all blocks for this file, (b) list of datanodes for each block (sorted by distance from client)
Block 1: at DN x1, y1, z1
Block 2: at DN x2, y2, z2
Block 3: at DN x3, y3, z3
...and so on...
Now I know how many blocks to download, and the datanodes where each block is stored
So I download each block, in turn, like so --
DOWNLOAD DATA
Download data from the nearest datanode (the first in list)
Please give me block n
DATA for block n
Umm.. Question -- What happens when the datanode is dead, or does not have the data, or the data is corrupted ...
Actually, HDFS can very elegantly handle these faults and more as we will see next --

Hadoop Fault Tolerance Architecture


FAULT TOLERANCE IN HDFS. PART I: TYPES OF FAULTS AND THEIR DETECTION
FAULT I: NODE FAILURE
There are typically three kinds of faults: The first is NODE FAILURE
Goodbye, cruel world
FAULT II: COMMUNICATION FAILURE
Second is COMMUNICATION FAILURE (cannot send and receive data)
where IS everybody?
????
FAULT III: DATA CORRUPTION
Third is DATA CORRUPTION
Data can be corrupted while sending over network
Data on Disk
Or corrupted while it is stored in hard disks

Hadoop Fault Tolerance Architecture


DETECTION #1: NODE FAILURES
NOTE: If Namenode is dead, the entire cluster is dead! Namenode is the SINGLE POINT OF FAILURE
Instead, let's focus on how datanode failures are detected
DETECTING DATANODE FAILURE
We send HEARTBEAT message every 3 seconds. This is our way of saying we are alive
If I don't get a message in 10 minutes, the datanode is dead to me
(I may be ALIVE and there was only a network failure, but the namenode treats both as same)



DETECTION #2: NETWORK FAILURES

Whenever data is sent, an ACK is replied by the reciever
Data
ACK
If the ACK is not received (after several retries), the sender assumes that the host is dead, or the network has failed

DETECTION #3: CORRUPTED DATA

Checksum is sent along with transmitted data
Checksum
Data
Moreover, when I store data in hard disks, I also store the checksum
Data XXXXXXXX XXXXXXXX
Checksum

DETECTING CORRUPTED HARD DRIVES

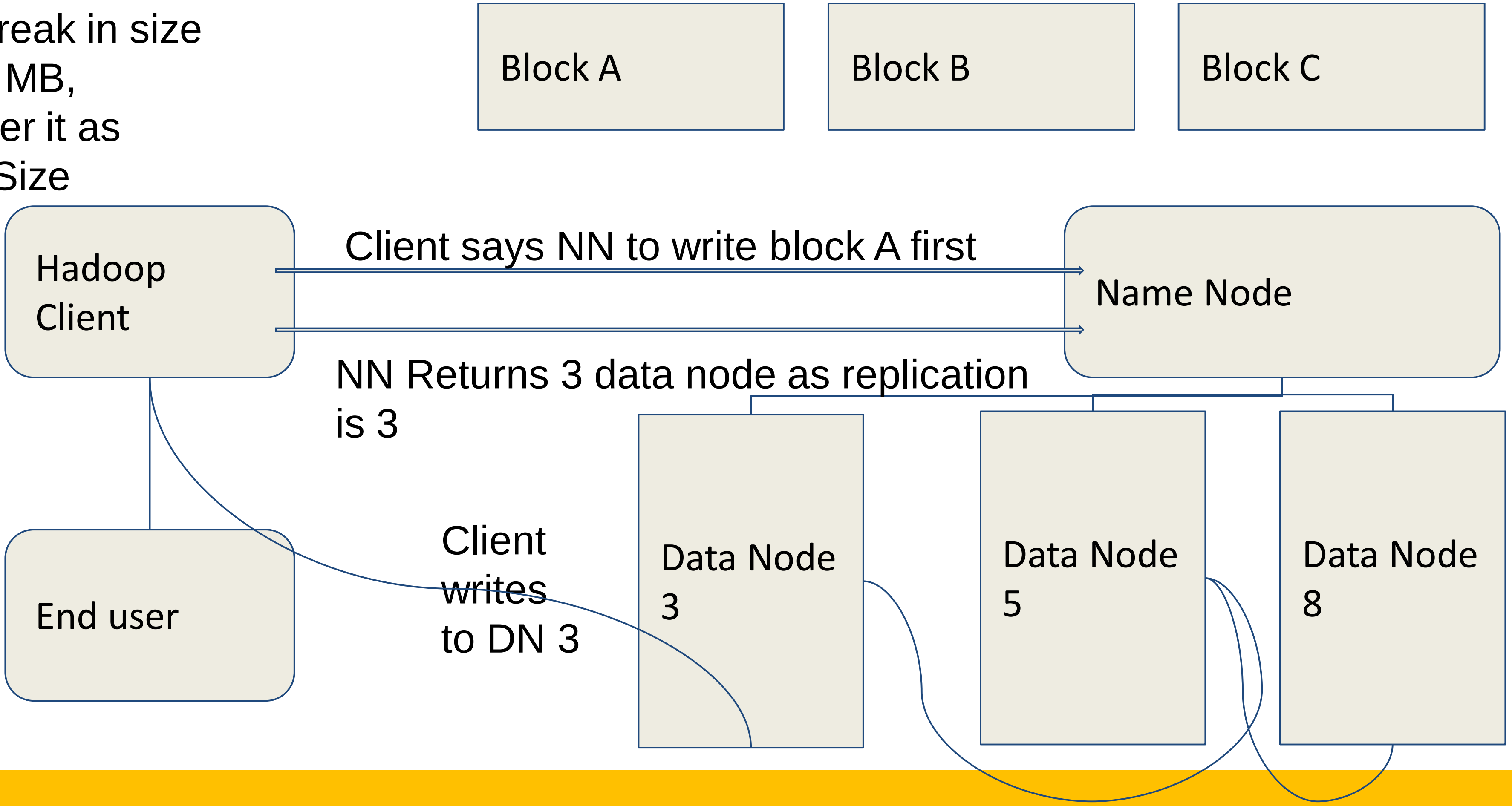
Periodically, all datanodes send BLOCKREPORT to the namenode
List of all blocks I have
Before sending block report I check if checksums are ok. I don't send info for blocks that are corrupted
I have four blocks
I thought he had five blocks.. so one block is corrupted

RECAP: HEARTBEAT MESSAGES AND BLOCK REPORTS

We send heartbeats every 3 seconds to say we are alive
HEARTBEAT
BLOCK REPORT
We send block reports and we skip blocks that are corrupted
(which is how the namenode will know which blocks are lost)

Hadoop Write Architecture

Lets Break in size of 256 MB, consider it as Block Size



Block A

Block B

Block C

Hadoop Client

Client says NN to write block A first

Name Node

NN Returns 3 data node as replication is 3

End user

Client writes to DN 3

Data Node 3

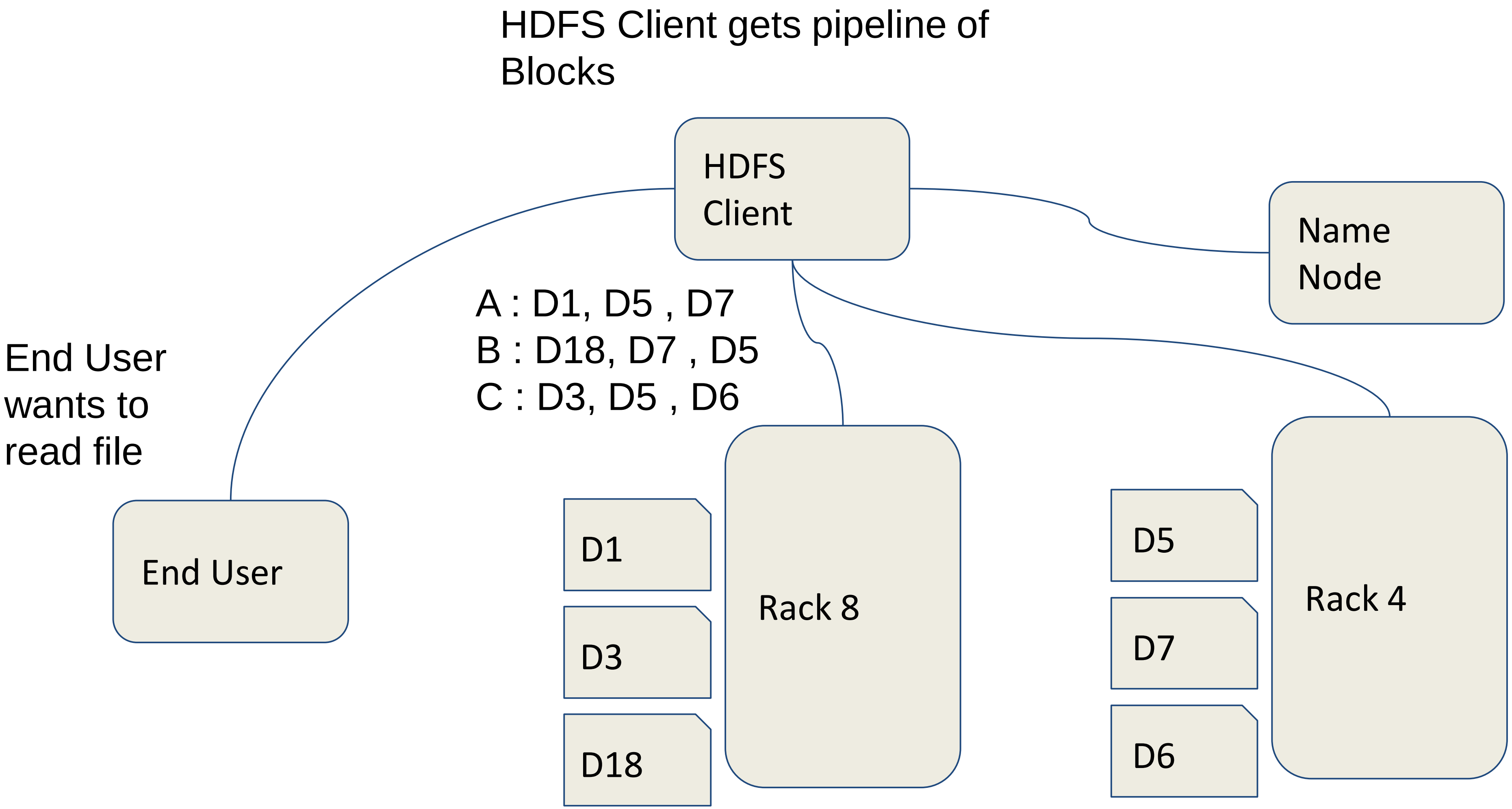
Data Node 5

Data Node 8

Write

768 MB FIle

Hadoop Read Architecture



HDFS Client gets pipeline of Blocks

A : D1, D5 , D7
B : D18, D7 , D5
C : D3, D5 , D6

End User wants to read file

HDFS Commands

a. HDFS Commands
   i. mkdir
   ii. Ls
   iii. Put
   iv. Get
   v. Cat
   vi. Cp
   vii. Mv

Microsoft Word
Document

Introduction to Map Reduce

MapReduce is a programming model whose libraries have been primarily written in Java

- Record-oriented data processing
- Operates on key and value pairs
- Consists of two phases :
  - ✓ Map
  - ✓ Reduce
- Every Map/Reduce program must specify a Mapper and optionally a Reducer
- Where ever possible, each node processes data stored on that node (Data Locality)
- Provides pluggable APIs and configuration mechanisms for writing
  applications Map and Reduce functions
  Input formats and splits
  Number of tasks, data types, etc…

MAPREDUCE

- MapReduce provides
  - Automatic parallelization, distribution
  - I/O scheduling
    - ✓ Load balancing
    - ✓ Network and data transfer optimization
- Fault tolerance
    - ✓ Handling of machine failures
- **Need more power: Scale out, not up!**
    - ✓ Large number of **commodity servers(**less processors and less RAM) as opposed to some high end specialized servers, this saves cost.

Technical Prerequisites

- It is expected that the students have knowledge about the following concepts in Java for understanding MapReduce Working

  - ✓ Class and Object
  - ✓ OOPS concepts : Abstraction ,Encapsulation ,Polymorphism and Inheritance
  - ✓ Abstract class and abstract methods
  - ✓ Keyword : Extends and implements meaning
  - ✓ Input formats (int,string,Boolean,double ,etc)
  - ✓ Knowledge of exceptions and exception handling (to understand code)
  - ✓ Use of loops like for,if,while etc in Java
  - ✓ Basic datastrucutres

- Apart from this, knowledge of basic Hadoop shell commands is required.

Motivation for MapReduce

- **Large-Scale Data Processing**
  - ✓ Need to manage large number of machines(CPU's) but the system should be hassle free.
- **MapReduce Architecture provides**
  - ✓ Automatic parallelization & distribution
  - ✓ Fault tolerance
  - ✓ I/O scheduling
  - ✓ Monitoring & status updates
  - ✓ Security and administration
  - ✓ Flexibility

Ways to use MAPREDUCE

Libraries

- HBase
- Hive
- Pig
- Sqoop
- Oozie
- Mahout

Languages

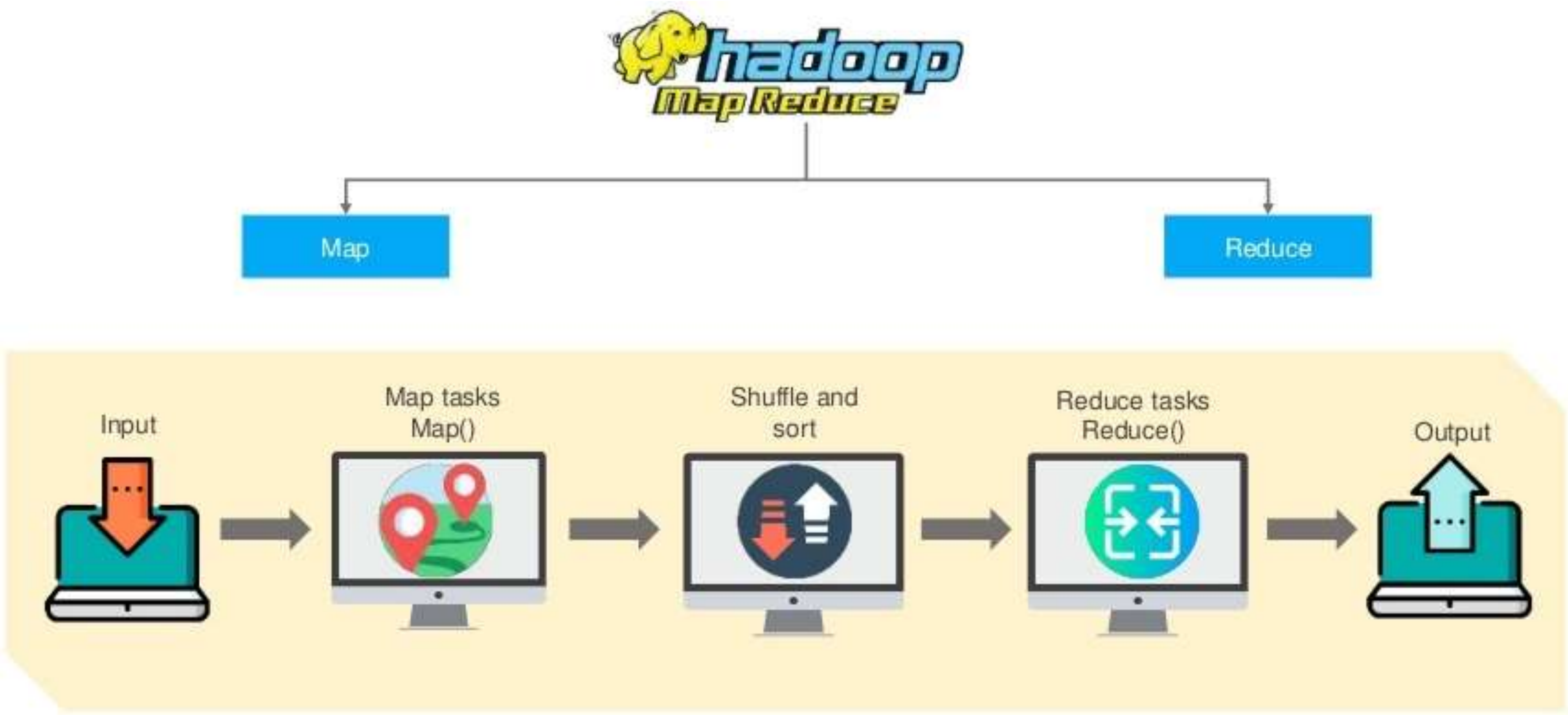
- Java
- HiveQL
- PigLatin
- Python
- JavaScript
- R

Keys and Values

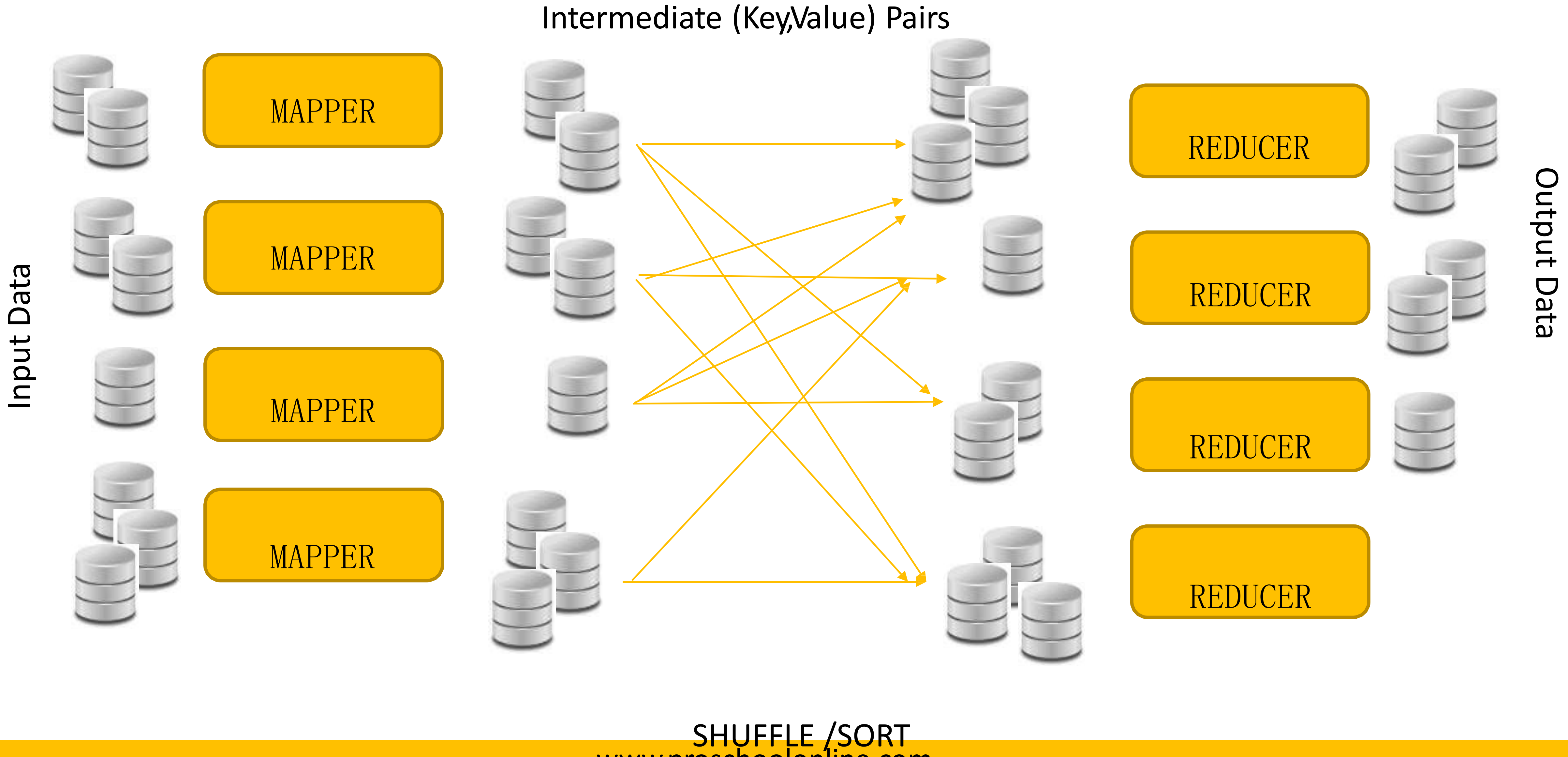
- Keys are objects which implement WritableComparable
  - ✓ A Writable Comparable is a Writable which is also Comparable.
  - ✓ Two WritableComparables can be compared against each other to determine their order.
  - ✓ Keys must be WritableComparables because they are passed to the reducer in sorted order.

- Values are objects which implement Writable interface
  - ✓ IntWritable for ints
  - ✓ LongWritable for longs
  - ✓ FloatWritable for floats
  - ✓ DoubleWritable for doubles
  - ✓ Text for strings

Map Reduce Steps



MAPREDUCE working



MAPREDUCE 1/3

Map

1) Key,Value(K1,V1)

Input (Data)

Input Split

2) list (K2, V2)

Key / Value out (intermediate values)

One list per local node

Can implement local Reducer (or Combiner)

MAPREDUCE 2/3

Map

1) Key,Value(K1,V1)

Input(Data)

Input Split

2) list (K2, V2)

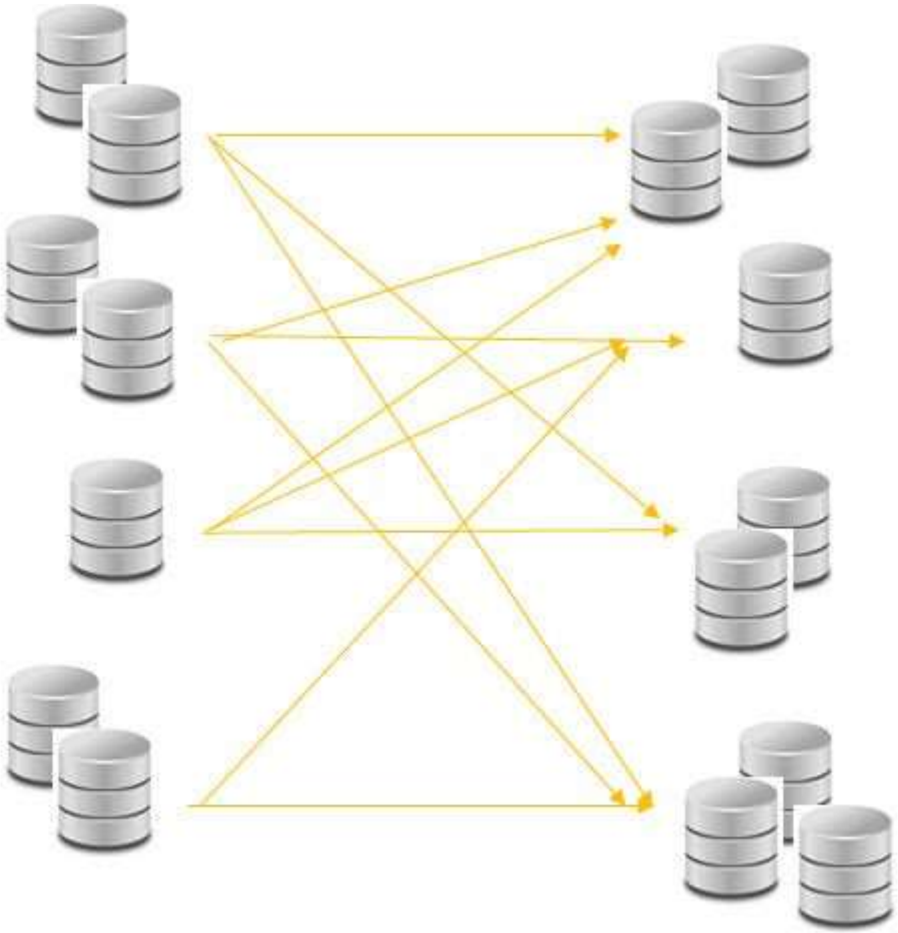
Key / Value out (intermediate values)

One list per local node

Can implement local

Reducer (or Combiner)

Shuffle/Sort



MAPREDUCE 3/3

Map

1) Key,Value(K1,V1)

Input (Data)

Input Split

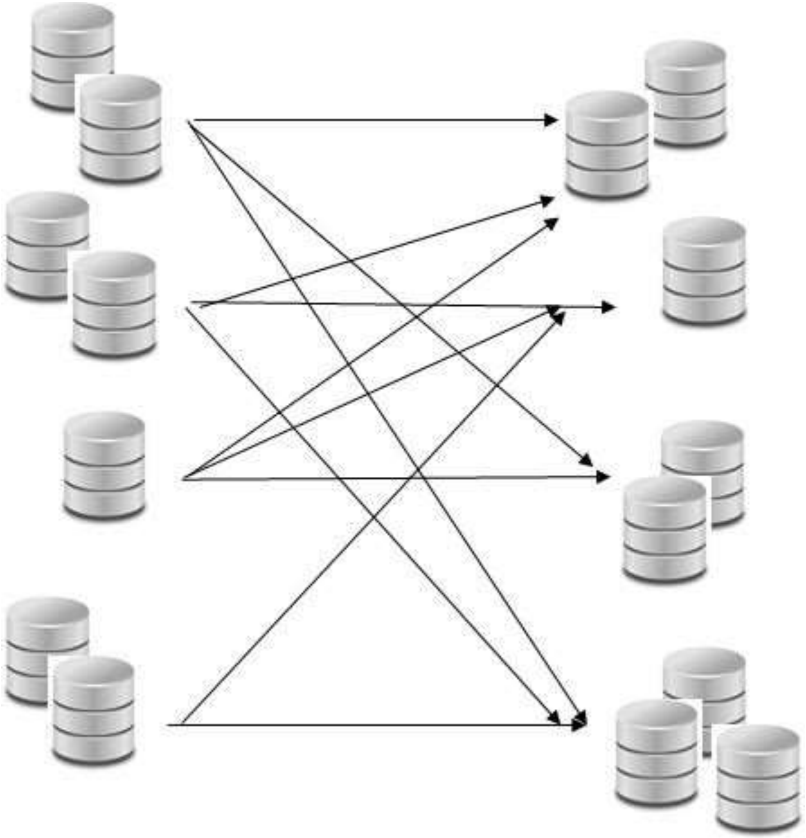
2) list (K2, V2)

Key / Value out (intermediate values)

One list per local node

Can implement local Reducer (or Combiner)

Shuffle/Sort



Reducer

1) (K2, list(V2))→

Shuffle / Sort phase precedes Reduce phase

Combines Map output into a list

2) list (K3, V3)

Usually aggregates intermediate values

Mapper

```
package mapper;

import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class WordCountMapper extends Mapper<LongWritable,Text,Text,LongWritable>  {
        @Override
        protected void map(LongWritable key,Text value,Context context)
        throws IOException,InterruptedException
        {
                LongWritable one = new LongWritable(1);

                String line = value.toString();
                String [] words = line.split(" ");

                for(String word : words)
                {
                        context.write(new Text(word),one);
                }}}
```

Reducer

```
package reducer;

import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer extends Reducer<Text,LongWritable,Text,LongWritable> {

        @Override
        protected void reduce(Text key,Iterable<LongWritable> values,Context context) throws IOException,
InterruptedException{
                long count = 0;

                for(@SuppressWarnings("unused") LongWritable value : values)
                {
                        count++;
                }

                LongWritable finalCount = new LongWritable(count);
                context.write(key,finalCount);
        }
}
```

Driver

```
package driver;

import java.io.IOException;
import mapper.WordCountMapper;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Job; import
org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.mapreduce.lib.output.*;
import reducer.WordCountReducer;

public class WordCountDriver {

@SuppressWarnings("deprecation")
public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException{

        Configuration conf = new Configuration(); // standard to be followed
        Job job= new Job(conf);                    //standard to be followed
        job.setJarByClass(WordCountDriver.class);  //starting point
        job.setMapperClass(WordCountMapper.class);
        job.setReducerClass(WordCountReducer.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(LongWritable.class);

        job.setOutputValueClass(LongWritable.class);
        job.setOutputKeyClass(Text.class);


        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);


        Path input = new Path(args[0]);
        Path output = new Path(args[1]);

        FileInputFormat.addInputPath(job,input);
        FileOutputFormat.setOutputPath(job,output);

        boolean isJobRunning = job.waitForCompletion(true);     //start the job and
                                                          wait for completion

        System.exit(isJobRunning ? 0:1); // return functions for exit

  }
}
```

Hadoop - Shell commands

- hadoop fs –cat file:///file2 (copies source paths to stdout)
- hadoop fs –mkdir /user/hadoop/dir1 /user/hadoop/dir2 (creates directory at the specified path).
- hadoop fs –copyFromLocal <fromDir> <toDir>
- hadoop fs –put <localfile> hdfs://nn.example.com/hadoop/hadoopfile
- sudo hadoop jar <jarFileName> <method> <fromDir> <toDir>
- hadoop fs –ls /user/hadoop/dir1
- Hadoop fs –cat hdfs://nn1.example.com/file1
- hadoop fs –get /user/hadoop/file <localfile>
- FOR OTHER COMMANDS REFER : https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/FileSystemShell.html
- **sudo refers to 'super user' i.e. run as super user(administrator)

Click on Icon for more commands

Microsoft Word
Document

Appendix

proschool
An ims Initiative

1) https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html
2) Hadoop – The Definitive Guide
3) https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn site/YARN.html