Big Data & Hadoop

Nagabhushan

Agenda – Day 1

- Introduction to Big Data & Hadoop
- Hadoop - Use Cases & History
- Commercial Distributions of Hadoop
- Hadoop Storage Architecture - HDFS
- Hadoop Setup
- Working with Hadoop FS shell

What is Big Data? → Problem

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

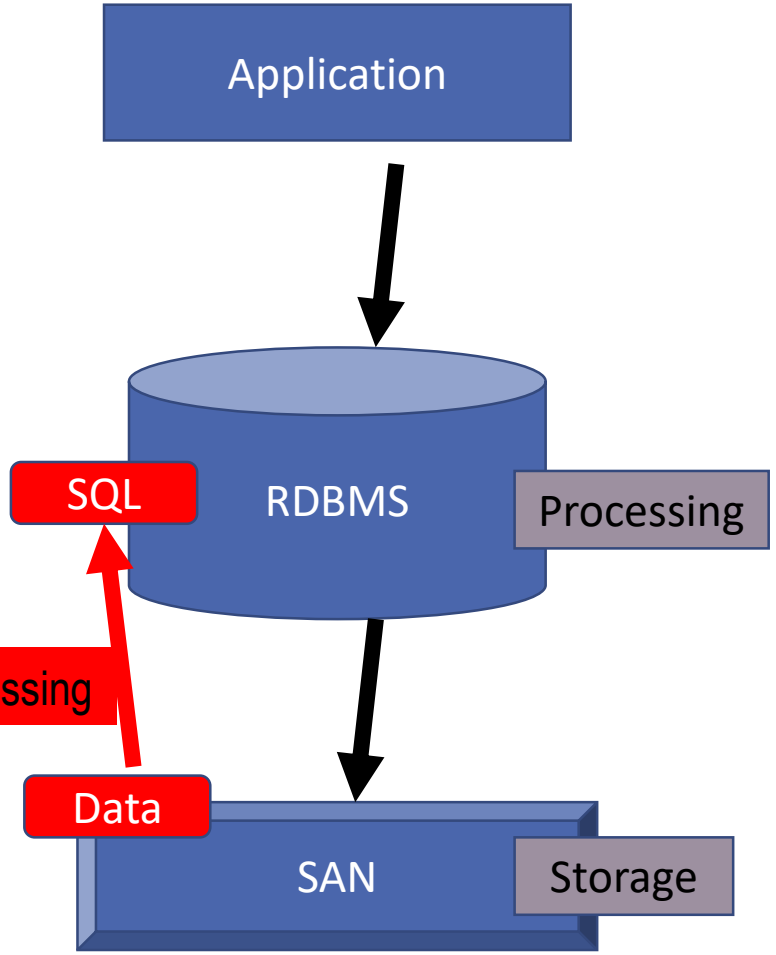
- *Hadoop's V → VALUE*

- *How to store Big Data? → HDFS*
- *How to process Big Data? → MapReduce (Hadoop 1.x) / YARN (Hadoop 2.x)*

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 5***
1 Yotabyte	*YB*	*1000000000000000000000000*

Problems with the traditional system



4 TB

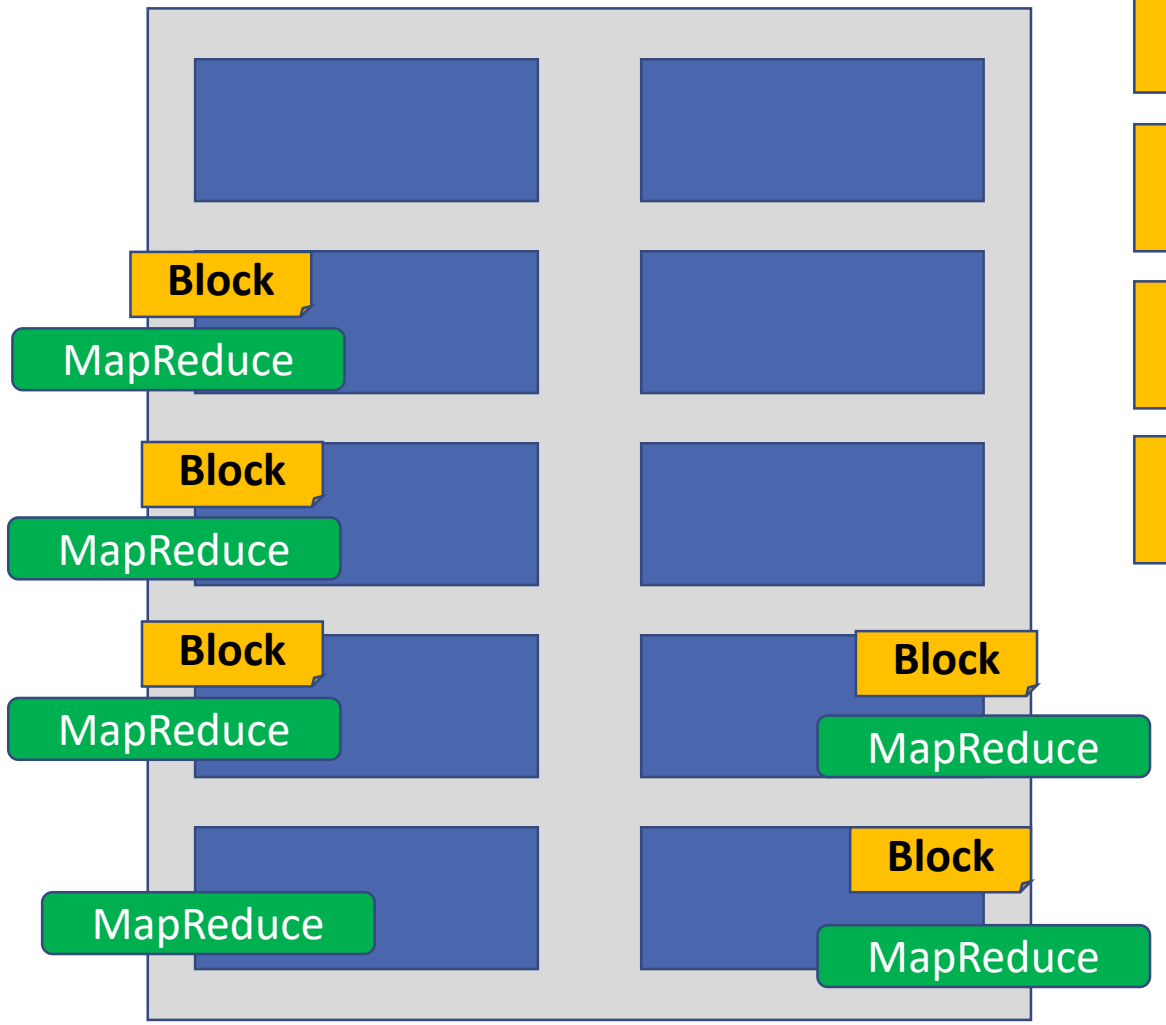
Proprietary Hardware

1 TB / day

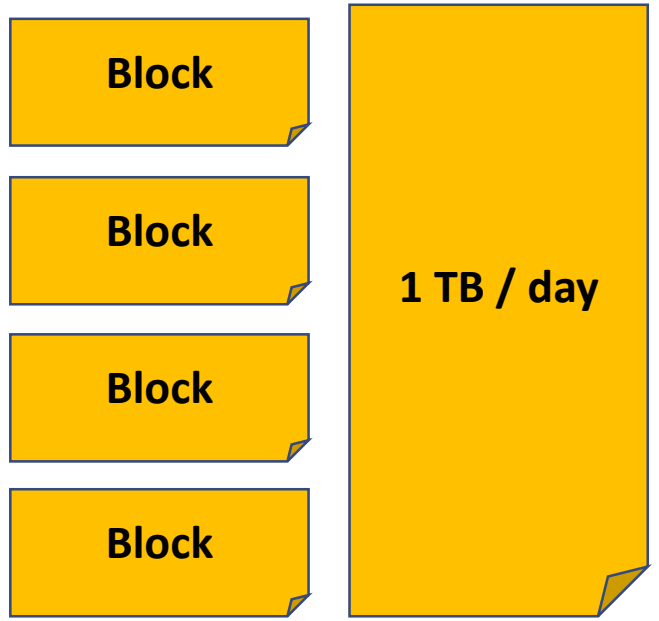
Big Data Systems to the rescue → Hadoop

Per Node
- 8 Cores of Xeon Processor
- 64 GB RAM
- 24 TB Storage

Per Cluster
- 80 Cores CPU
- 640 GB RAM
- 240 / 4 TB Storage



10 Node Hadoop Cluster



Moving Processing to Data

Commodity Hardware

Hadoop Layout / Node

Yet Another Resource Negotiator

Hadoop's Distributed File System

Java

Linux

Commodity Hardware

YARN — Processing

HDFS — Storage

J R E

OS

Infra

Features of Hadoop

- *Commodity Hardware*
- *Open Source*
- *Distributed Storage → HDFS*
- *Scale Out Architecture → Unlimited Nodes in a cluster*
- *Fault Tolerance → Replication*
- *Data Locality → a paradigm of moving processing to data for local computation → Parallel processing*
- *Java software library*
- *WORM → Write Once Read Many*

Hadoop project includes these modules:

- *Hadoop Common: The common utilities that support the other Hadoop modules* core-default.xml
- *Hadoop Distributed File System (HDFS™): A distributed file system that provides high-throughput access to application data* hdfs-default.xml
- *Hadoop YARN: A framework for job scheduling and cluster resource management* yarn-default.xml
- *Hadoop MapReduce: A YARN-based system for parallel processing of large data sets* mapred-default.xml

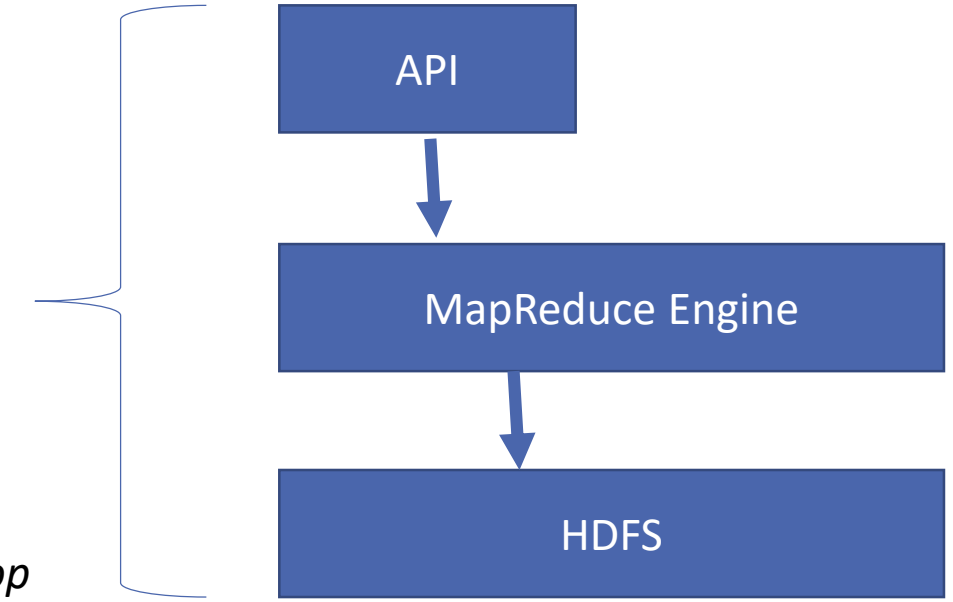
History of Hadoop

- *Google published whitepapers on GFS and MapReduce in 2004*
- *Yahoo hired Doug Cutting and Hadoop was born*
- *Yahoo handed over Hadoop project to Apache Software Foundation in 2006*

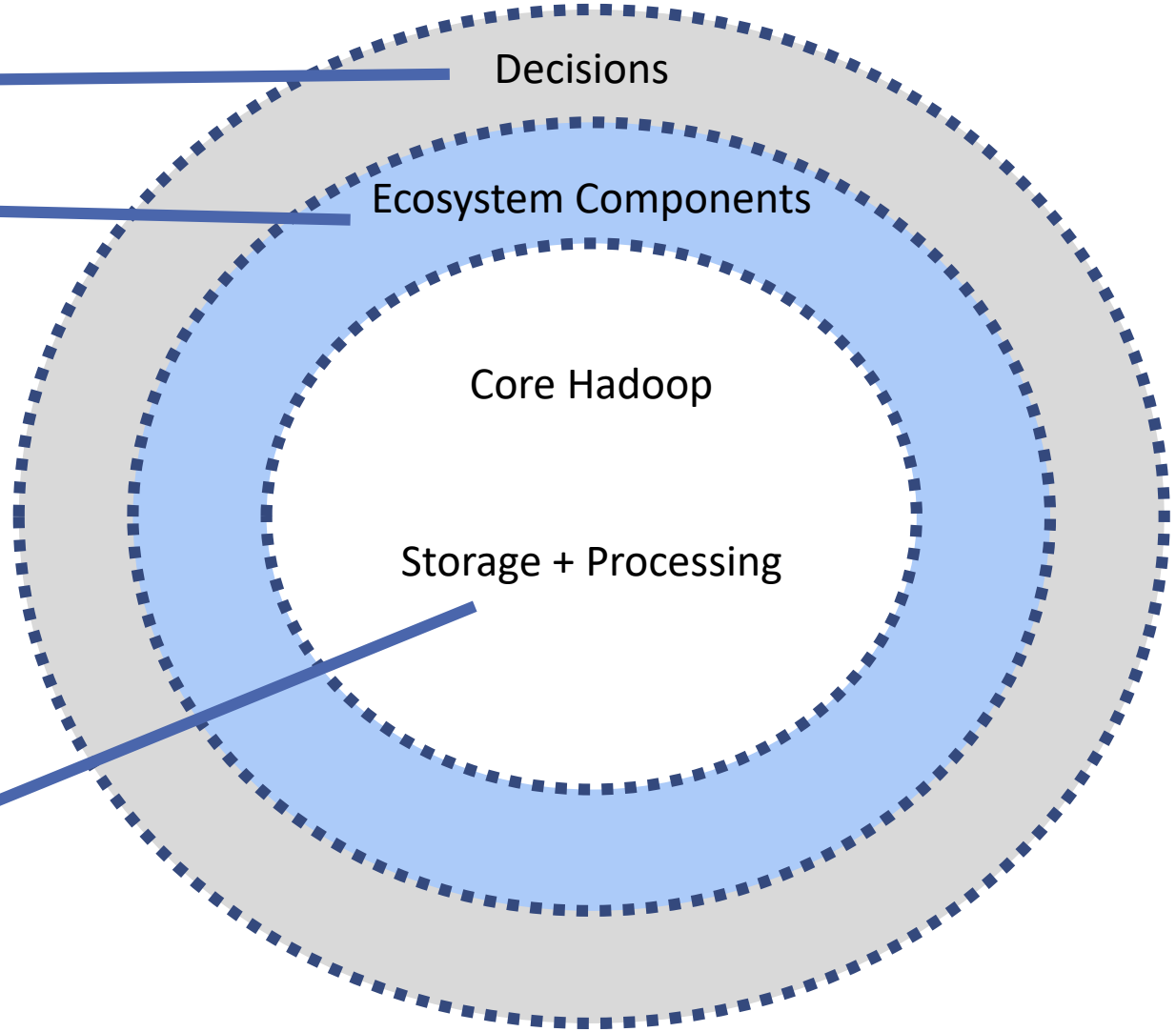
Java

Hadoop

Doug Cutting – Architect of Hadoop



Hadoop Ecosystem



Data Visualization

Analytics → **MR programs, Hive, Pig, Impala**, HBase, **Spark**...

Data Ingestion → **Sqoop, Flume,** Kafka

Administration → Ambari, **Cloudera Manager**

Security → Sentry, Kerberos

Synchronization → **ZooKeeper**

Workflows → Oozie

Web UI → **Hue**

HDFS + YARN

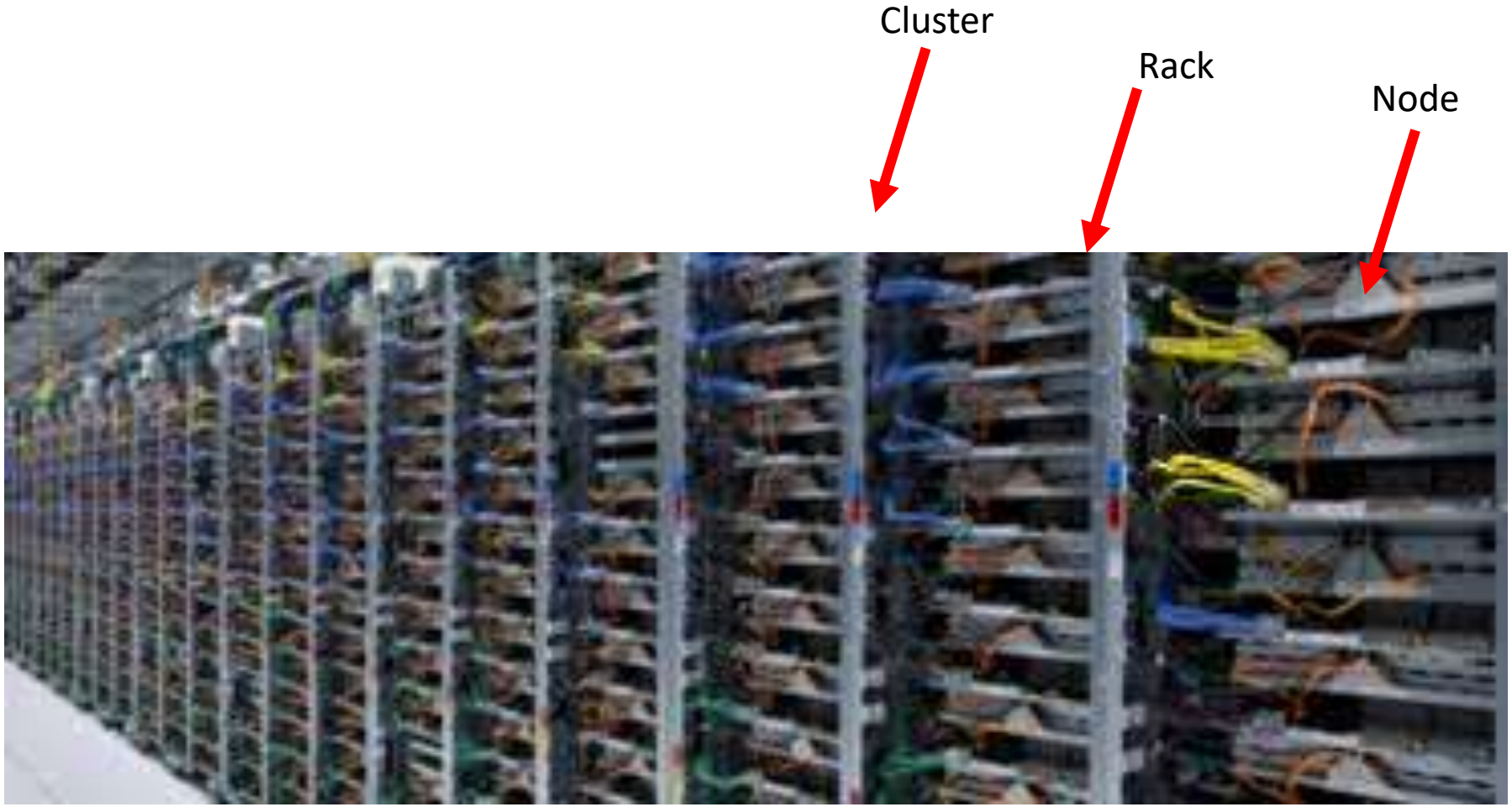
Commercial Distributions of Hadoop

- *Cloudera*
- *Hortonworks*
- *MAPR*
- *Big Insights (IBM)*

- *Apache Hadoop → Open Source*

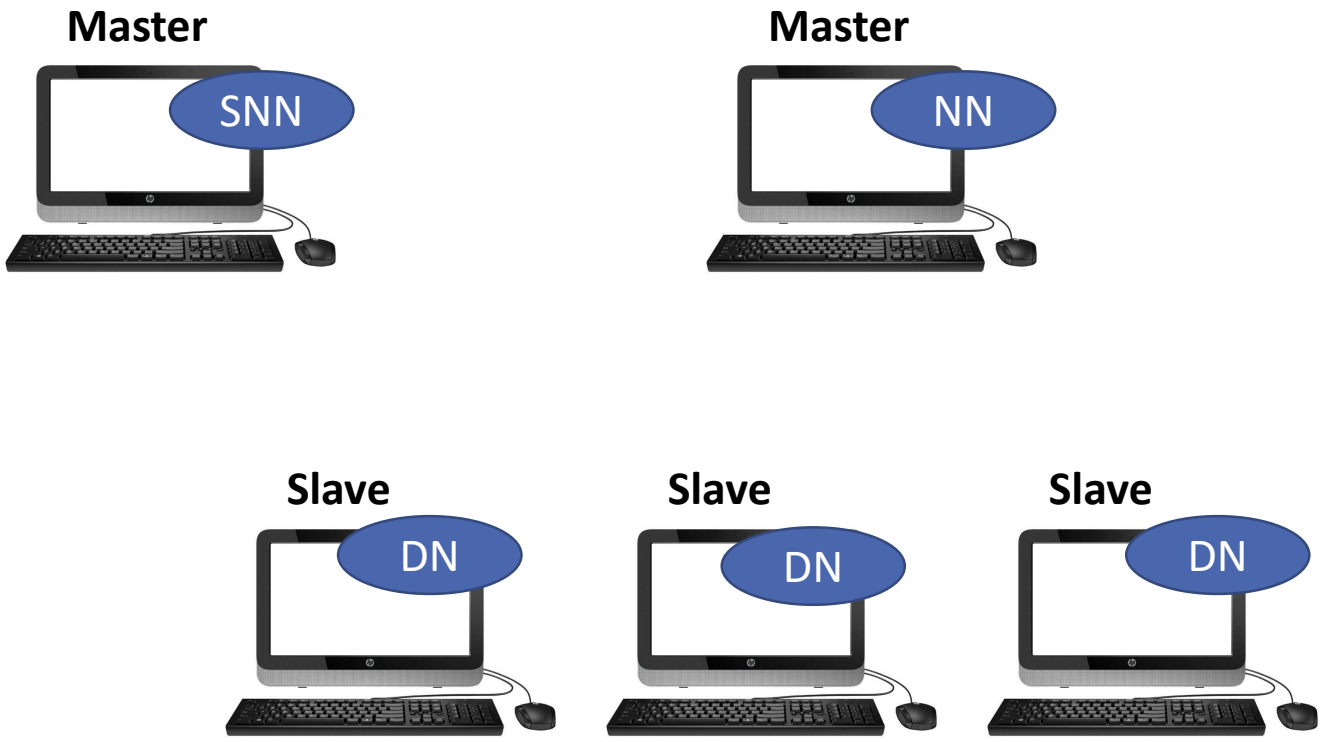
Hadoop Storage Architecture - HDFS

Hadoop Terminologies



HDFS Daemons

Master – Slave Architecture



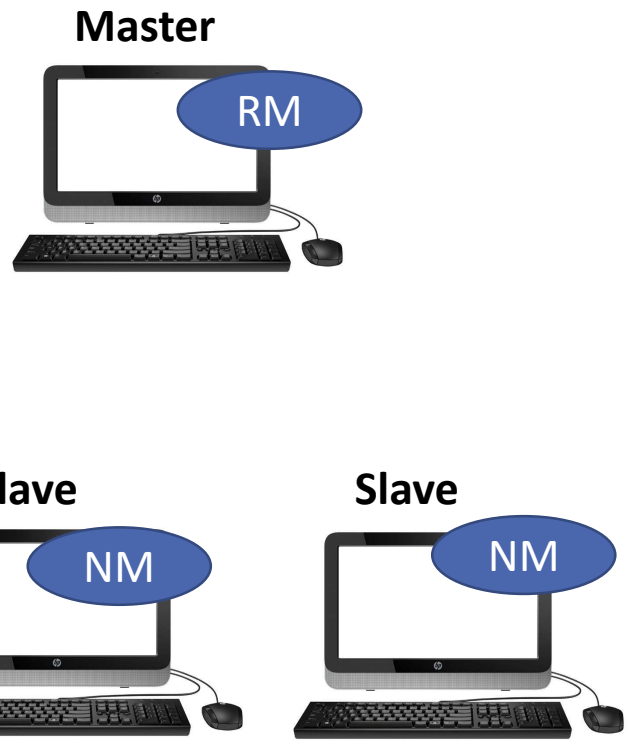
NN NameNode

DN DataNode

SNN Secondary NameNode

YARN Daemons

Master – Slave Architecture



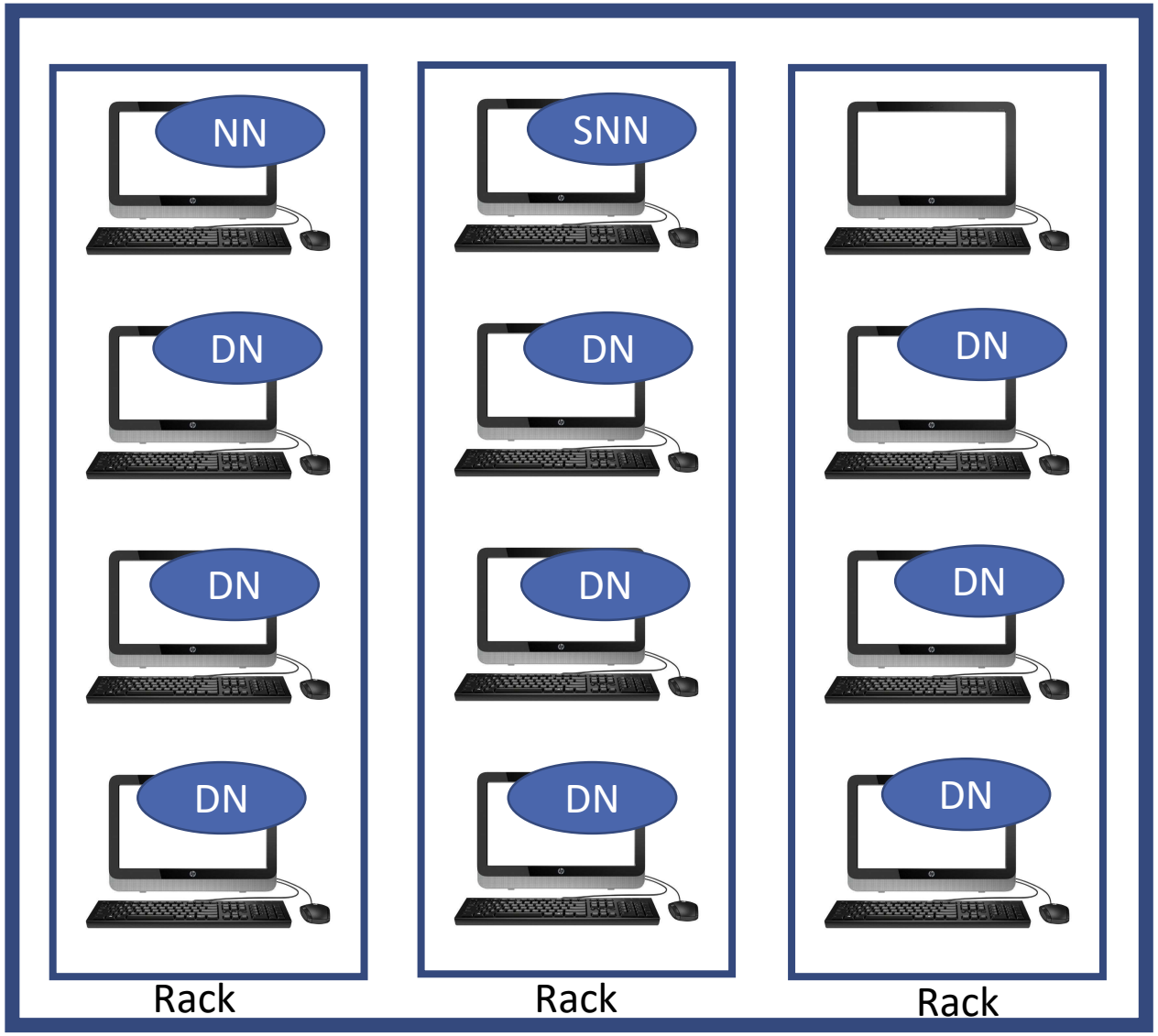
RM ResourceManager

NM NodeManager

HDFS Daemons distributed over a cluster

Hadoop binaries and configs

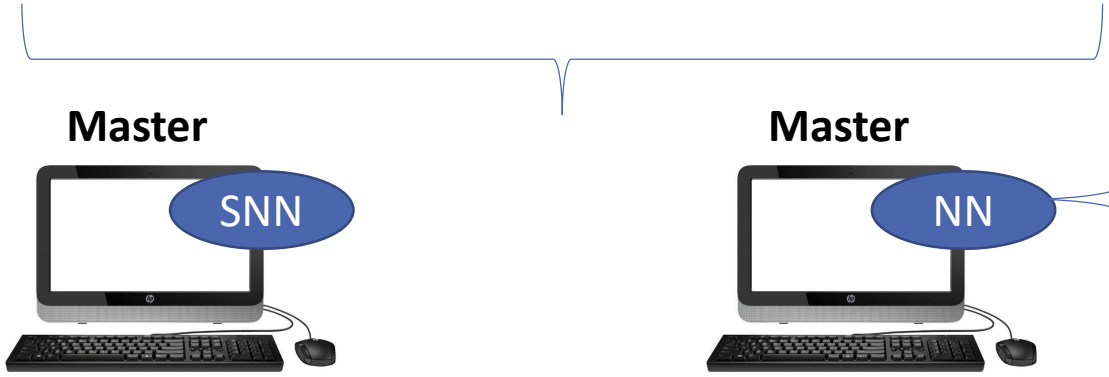
Gateway / Client Node



HDFS Daemons - Responsibilities

Master – Slave Architecture

- Checkpointing
  - Merge EditsInProgress with FSImage at regular intervals



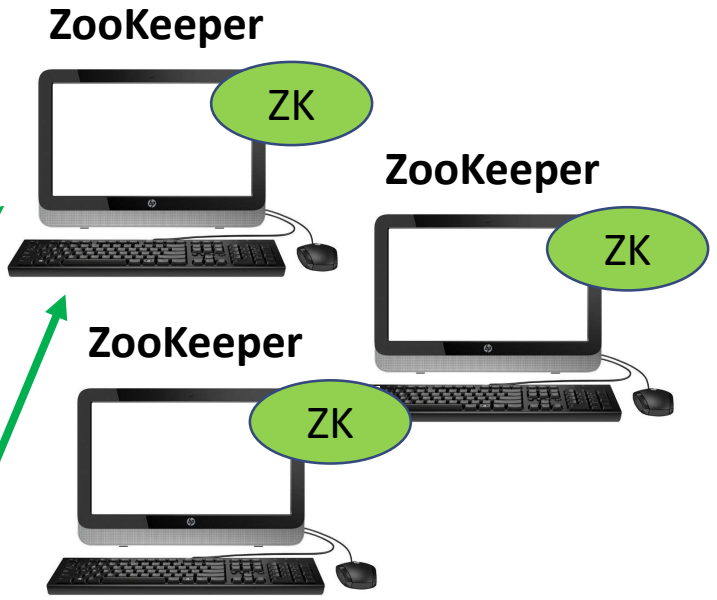
- Stores metadata of the FileSystem
  - File Vs Block Mapping
  - Block Vs Node Mapping
- Allocate DN for a file to be written
- Manages Fault Tolerance
- The metadata is stored on the local file system of the node running NN

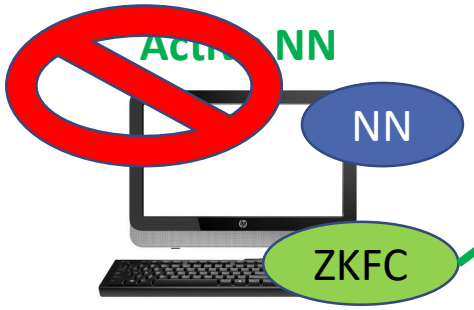
- Stores DataBlocks of a file
- Data Integrity
- Sends regular heartbeats to the NN
- The DataBlocks reside on the local file system of the node running DN

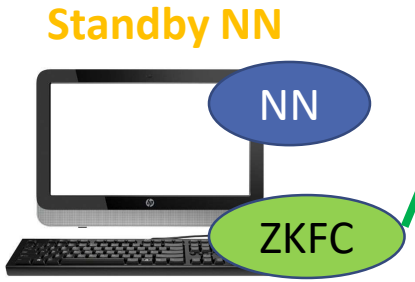
ZooKeeper - NN High Availability

Gateway / Client Node

Sync EditsInProgress with Active & Standby NN





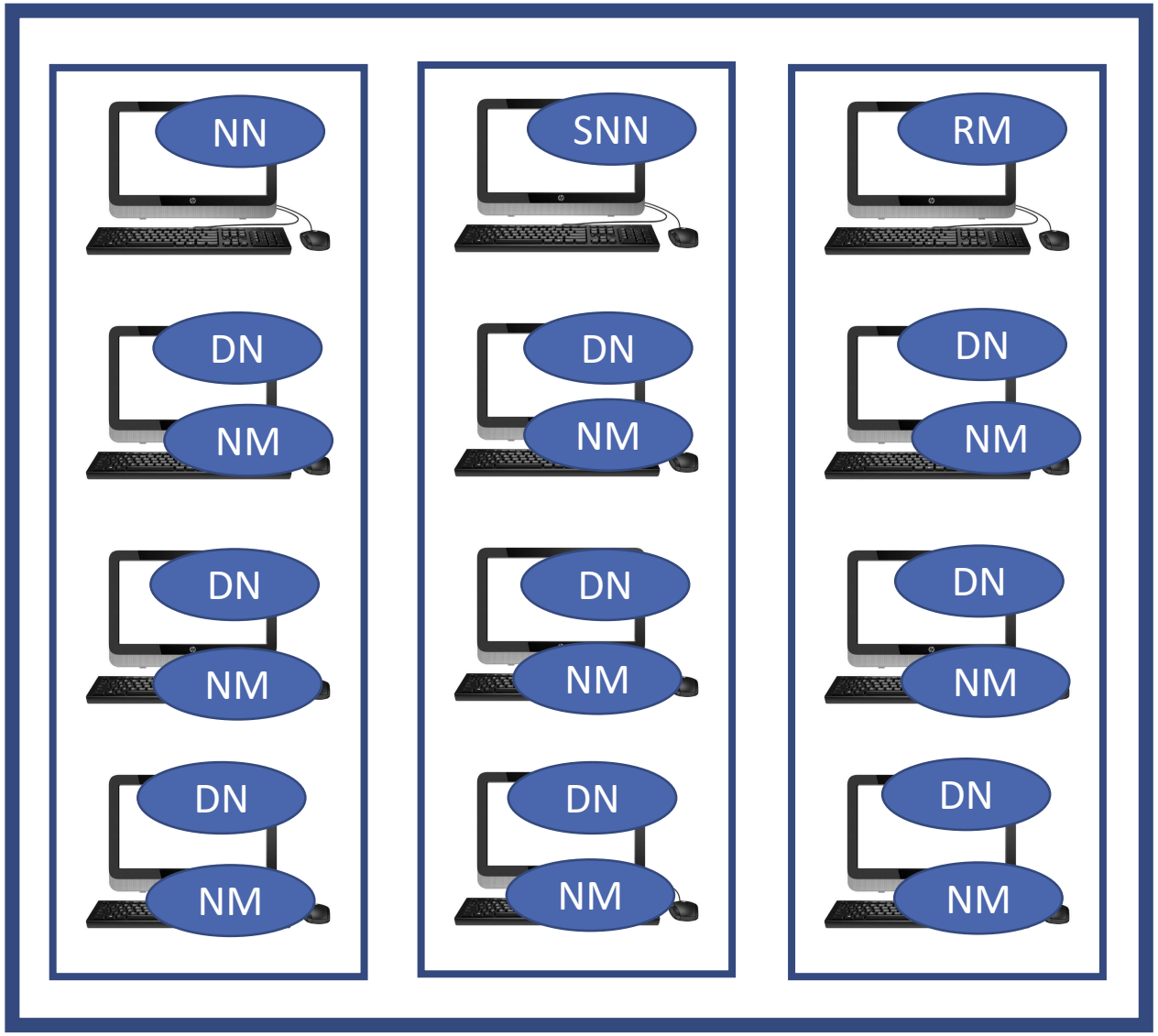


ZKFC – ZooKeeper Failover Controller

Hadoop Daemons distributed over a cluster

Hadoop Binaries and configs

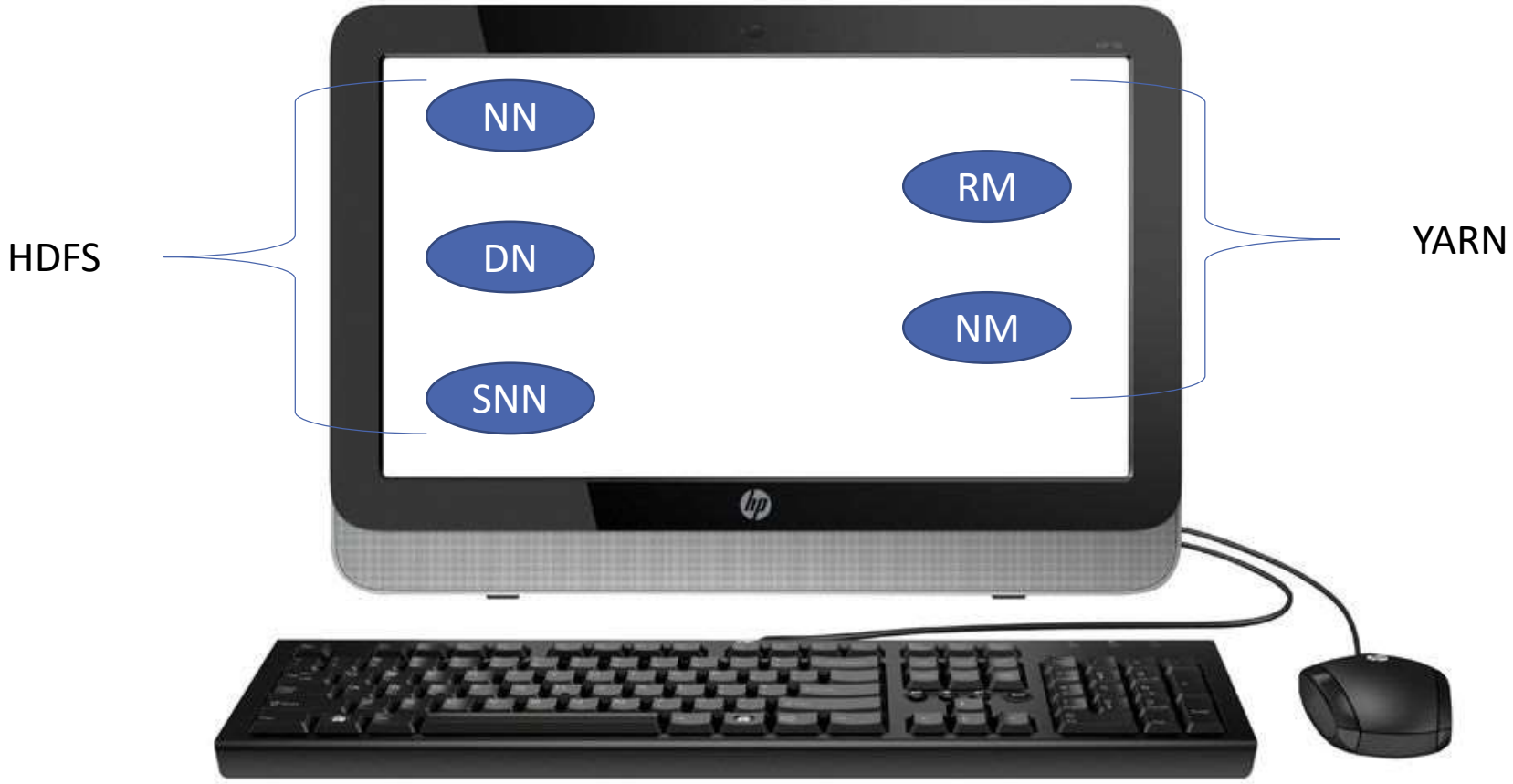
Gateway / Client Node



DataNode and NodeManager co-exist

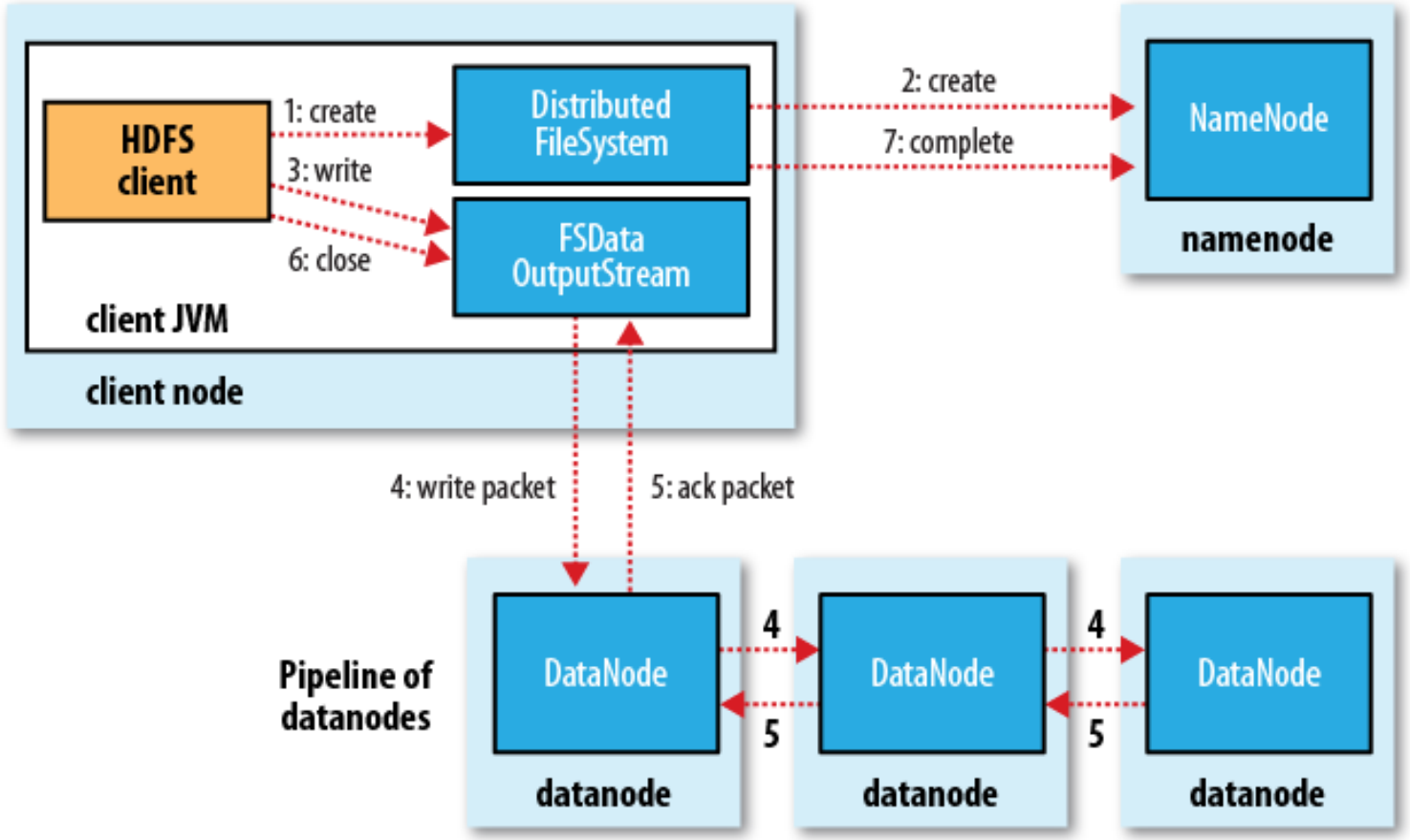
Hadoop Daemons distributed over a single node

Pseudo Distributed Mode

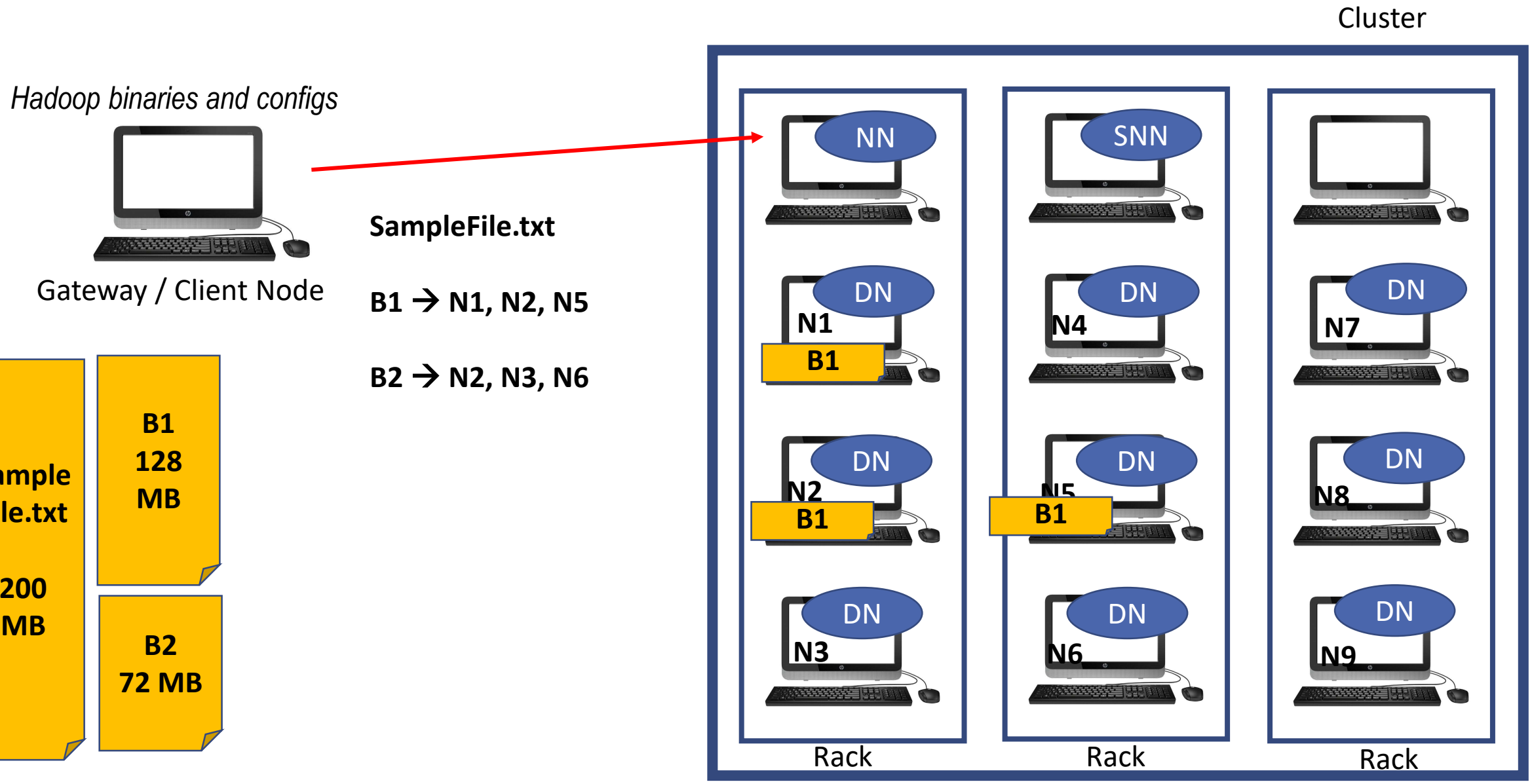


Anatomy of a File Write

```
$ hadoop fs –put <Source> <Destination>
```



HDFS Daemons distributed over a cluster



Anatomy of a File Write

1. *Client connects to the NameNode*
2. *NameNode places an entry for the file in the metadata, returns the block name and a list of DataNodes (Pipeline of DataNodes / block)*
3. *Client connects to the first DataNode and starts sending data*
4. *As the first DataNode receives the DataBlock, it will connect to the second and start sending data*
5. *Second DataNode connects with the third and replicates*
6. *ack packets are sent back to the client from the pipeline of DataNodes (in reverse direction)*
7. *Client reports to the NameNode that the blocks are written and then the metadata is committed (File write is complete)*

Hadoop Setup

Infra

- In premise
- Cloud – AWS / GCP / Azure…
- Virtualization

OS

- RHEL
- CentOS
- Ubuntu
- Fedora
- SUSE
- …..

JDK

- Open JDK
- Oracle JDK
- IBM JDK
- ….

Hadoop

- Cloudera
- Hortonworks
- Apache
- MapR
- Big Insights

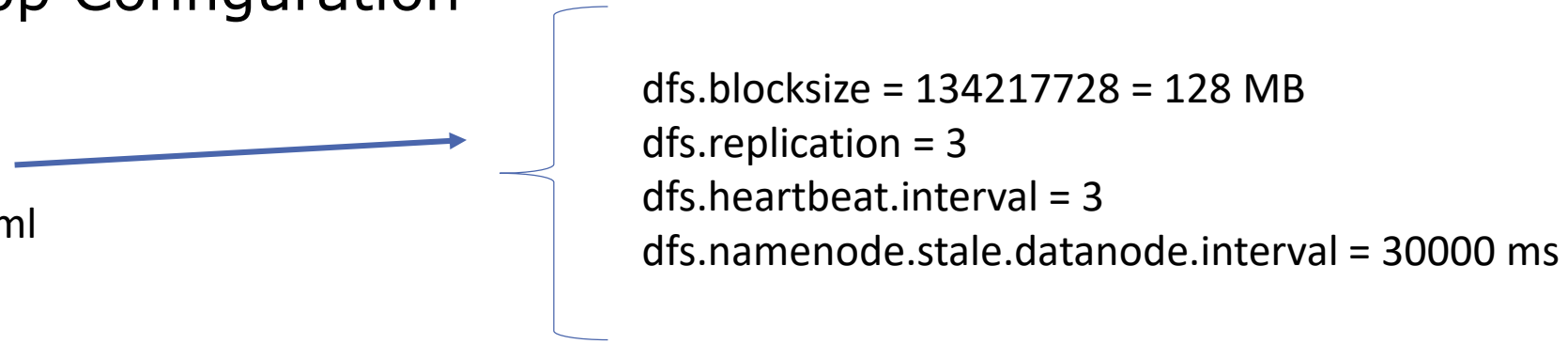
Hadoop Setup Mode

- Standalone Mode
- Pseudo Distributed Mode
- Fully Distributed Mode

Hadoop Configuration

Default Hadoop Configuration

core-default.xml
hdfs-default.xml
mapred-default.xml
yarn-default.xml



dfs.blocksize = 134217728 = 128 MB
dfs.replication = 3
dfs.heartbeat.interval = 3
dfs.namenode.stale.datanode.interval = 30000 ms

Customized Hadoop Configuration

core-site.xml
hdfs-site.xml
mapred-site.xml
yarn-site.xml

dfs.replication = 1

$HADOOP_HOME/etc/hadoop

Hadoop Setup Mode

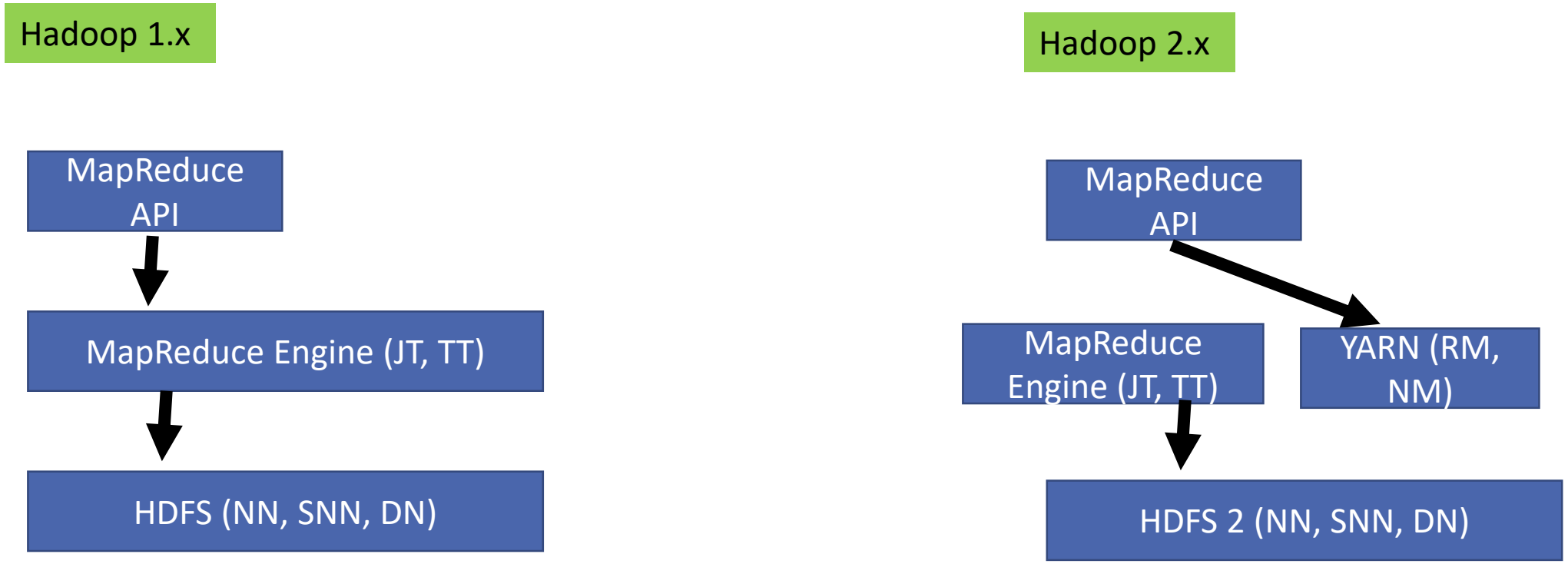
- Standalone Mode
  - Single Node, non distributed (Default)
  - Hadoop works as a single Java Process
- Pseudo Distributed Mode
  - Single Node, distributed
  - Each Hadoop daemon runs inside a separate JVM
    - HDFS → 1 NN, 1 DN, 1 SNN
    - YARN → 1 RM, 1 NM
- Fully Distributed Mode
  - Multi Node,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

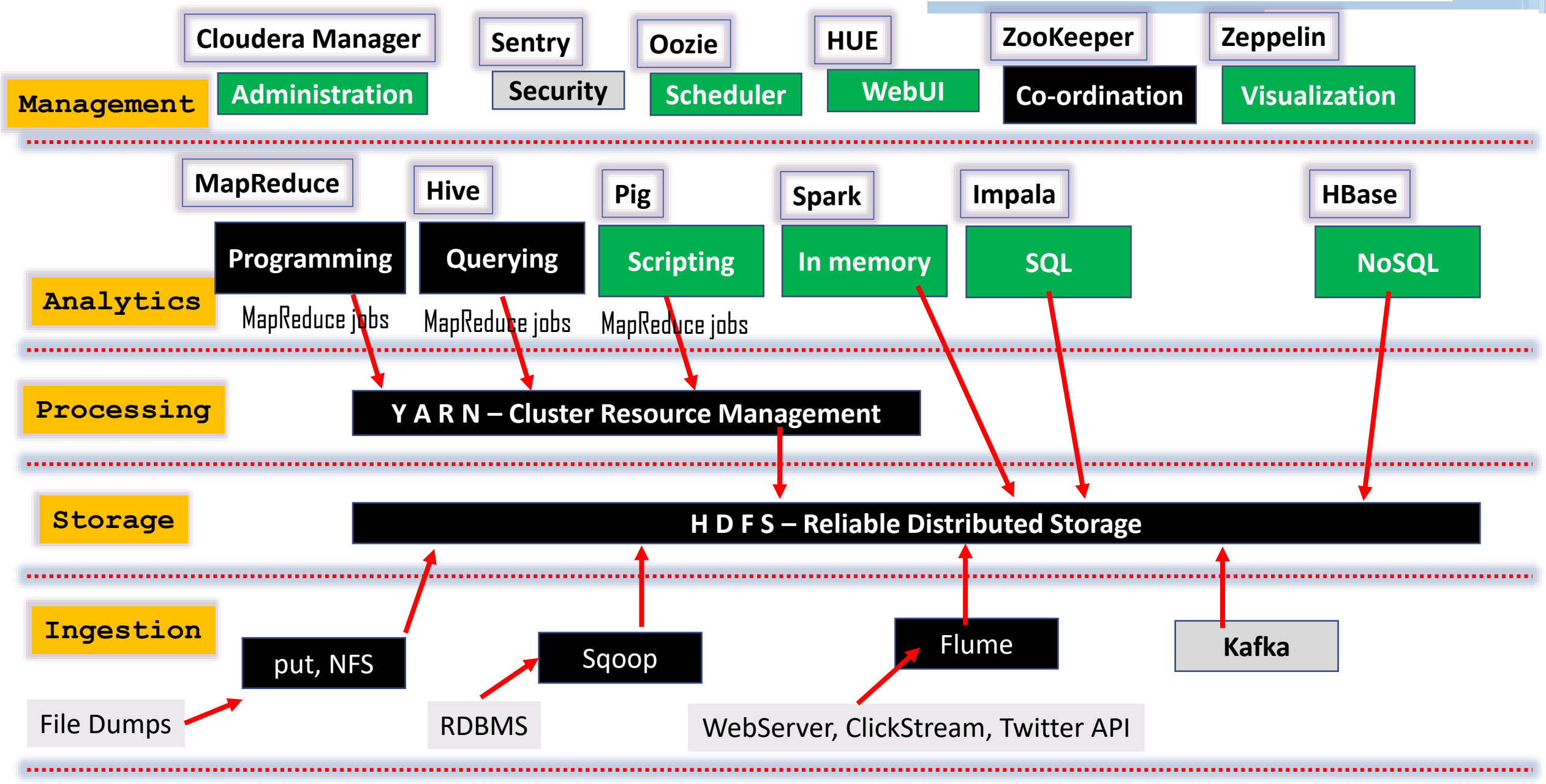
Rack Awareness

- *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*

http://hadoop.apache.org/docs/r2.7.3/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

Hadoop 1.x Vs Hadoop 2.x




Cloudera Manager
Sentry
Oozie
HUE
ZooKeeper
Zeppelin
Management
Administration
Security
Scheduler
WebUI
Co-ordination
Visualization
MapReduce
Hive
Pig
Spark
Impala
HBase
Programming
Querying
Scripting
In memory
SQL
NoSQL
Analytics
MapReduce jobs
MapReduce jobs
MapReduce jobs
Processing
Y A R N – Cluster Resource Management
Storage
H D F S – Reliable Distributed Storage
Ingestion
put, NFS
Sqoop
Flume
Kafka
File Dumps
RDBMS
WebServer, ClickStream, Twitter API

Hadoop Setup Steps – Pseudo Distributed Mode Setup

- Pre-Requisites
  - *Ubuntu OS*
  - *JDK*
  - *ssh (Passphraseless)*
- Download and unpack Hadoop
- Customize Hadoop
  - core-site.xml
  - hdfs-site.xml
  - mapred-site.xml
  - yarn-site.xml
  - hadoop-env.sh
- Format the NameNode
- Start Hadoop Services

Hadoop FS Shell Reference

- http://hadoop.apache.org/docs/r2.7.3/hadoop-project-dist/hadoop-common/FileSystemShell.html

NN & SNN

Master

Metadata is stored in these 2 files

- FileSystem Image ➔ fsimage_0000000000000000019
- Edit Log / Transaction log ➔ edits_inprogress_0000000000000000020

Master

FSImage is stored on SNN also, however no edits in progress

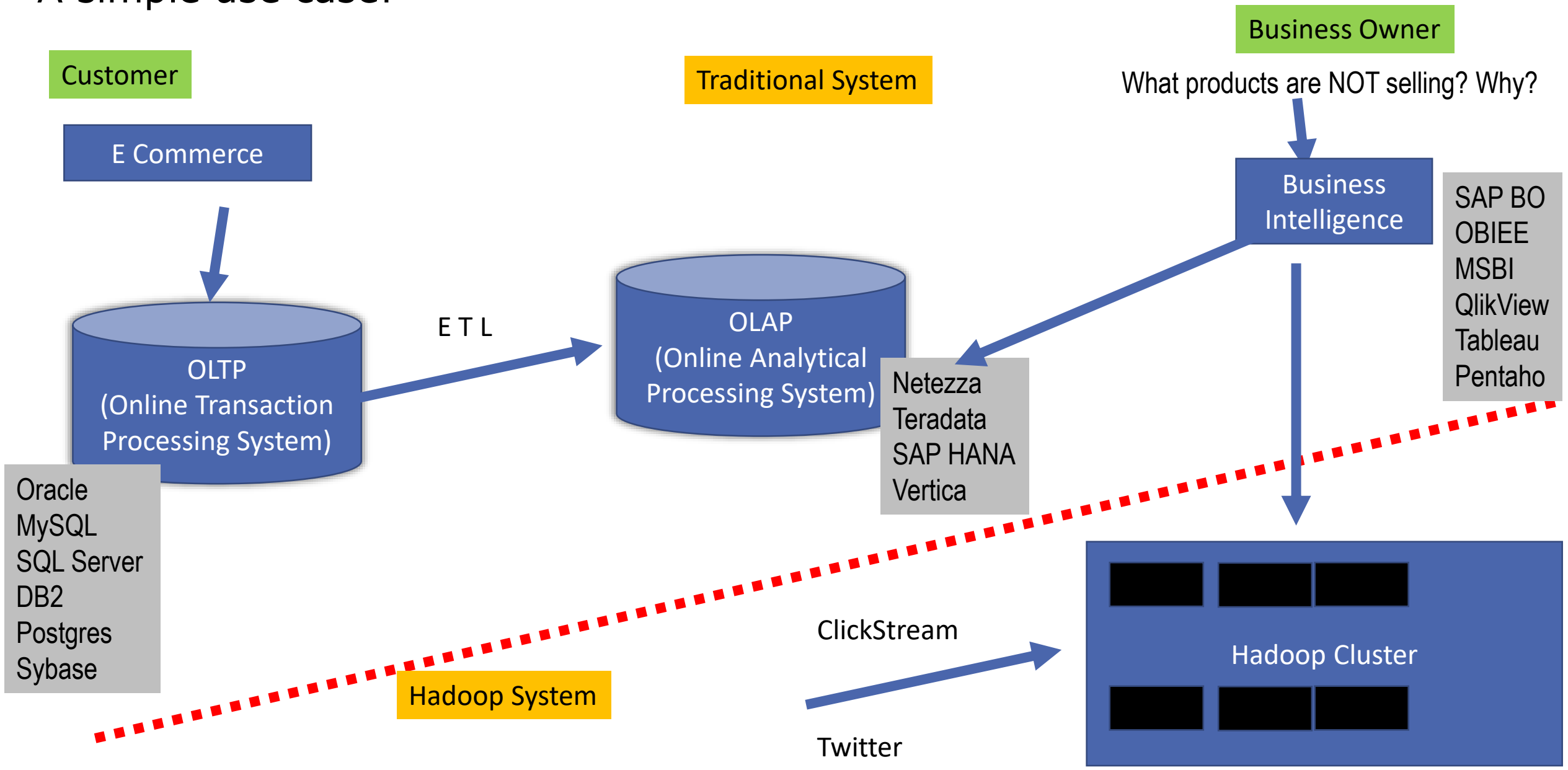
- FileSystem Image ➔ fsimage_0000000000000000019

- SNN is not a hot backup for the NN
- In a scenario where NN failed, the administrator can only recover the FSImage and restore the cluster to previous state
- 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

Agenda – Day 2

- Data Ingestion Techniques
  - Structured Data → Apache Sqoop
  - Unstructured Data → Apache Flume
- Hadoop Cluster Resource Management - YARN
- Hadoop Data Processing - MapReduce

A simple use case!



Apache Sqoop

http://sqoop.apache.org/

- *Sqoop is a tool to transfer data from RDBMS to Hadoop and vice versa*
- *Sqoop is "the SQL-to-Hadoop database import tool"*
- *Open-source Apache project*
- *Originally developed at Cloudera*
- *Designed to import data from RDBMSs into HDFS*
- *Can also send data from HDFS to an RDBMS*
- *Uses JDBC (Java Database Connectivity) to connect to the RDBMS*

How does Sqoop work?

- *Sqoop examines each table and automatically generates a Java class to import data into HDFS*
- *It then creates and runs a Map-only MapReduce job to import the data*
  - *By default, four Mappers connect to the RDBMS*
  - *Each imports a quarter of the data*
  - *We can also perform a sequential import*

Apache Sqoop

- *Sqoop Import ➔ From RDBMS to HDFS*
- *Sqoop Export ➔ From HDFS to RDBMS*

- *Sequential Import / Export ➔ m 1*
- *Parallel Import / Export ➔ multiple maps*

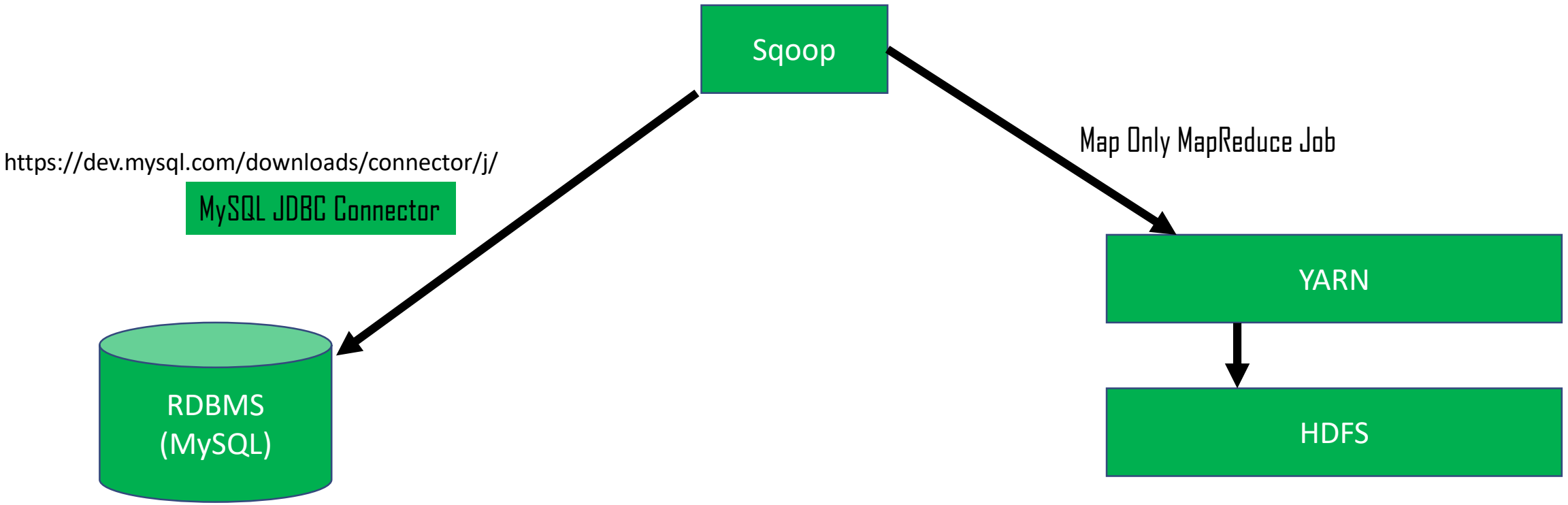
- *--as-textfile (Default) - csv*
- *--as-sequencefile (Binary)*
- *--as-avrodatafile - (JSON like)*
- *--as-parquetfile – (Columnar)*

http://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html

Apache Sqoop

http://sqoop.apache.org/

- Sqoop is a tool to transfer data from RDBMS to Hadoop and vice versa



Sqoop Installation and Configuration

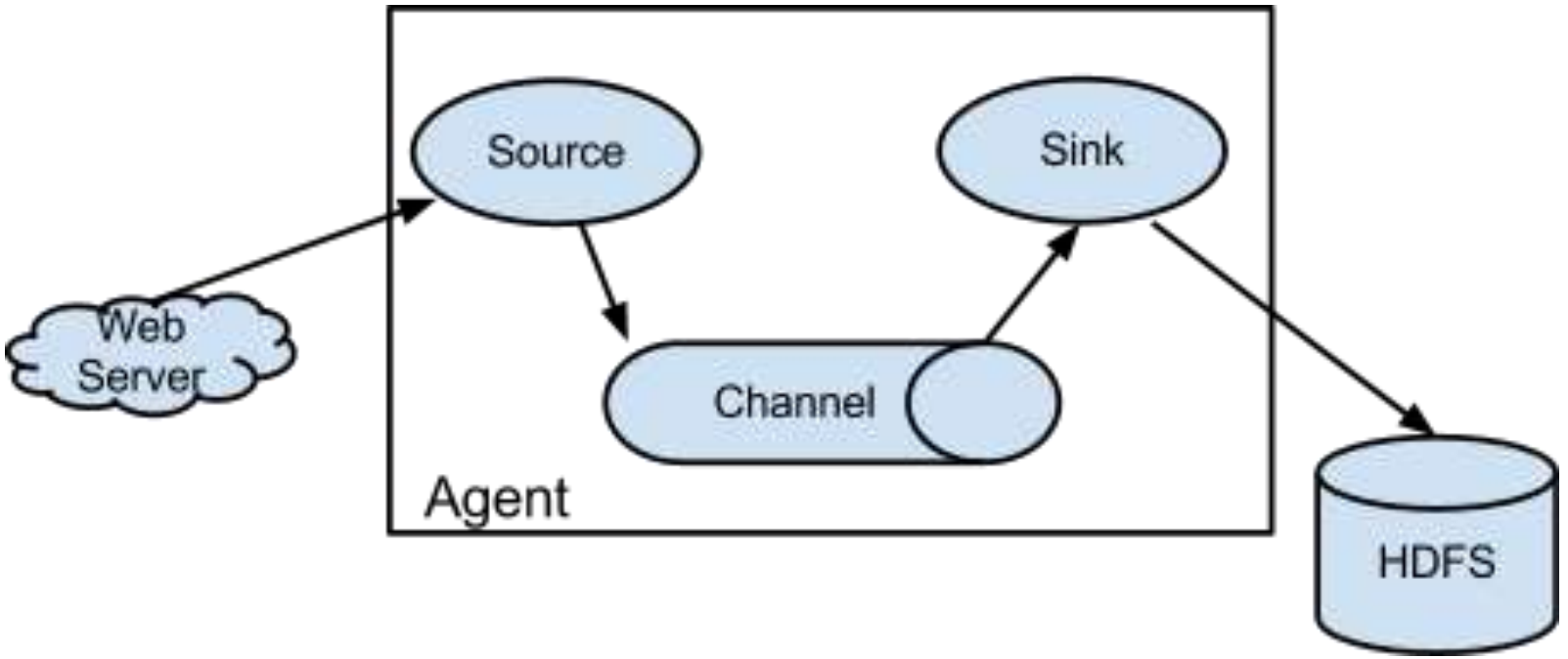
- *Connectors and JDBC drivers are installed on every client*
- *Database connectivity required for every client*
- *CLI is the client interface*
- *Every invocation requires credentials to RDBMS*

Apache Flume

http://flume.apache.org/

- *Flume is a distributed service for efficiently collecting and moving large amount of log data into HDFS*
- *Suitable for gathering logs from multiple systems and inserting them into HDFS as they get generated*
- *Flume is an open source Apache project*
- *Flume was initially developed at Cloudera*

Flume example



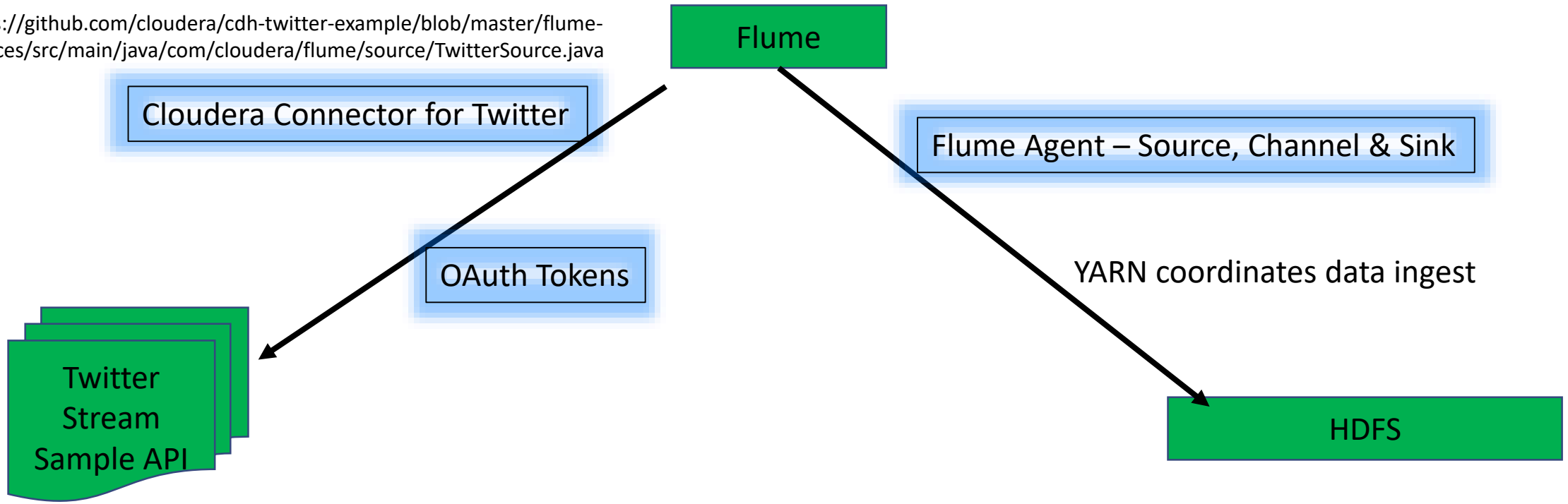
Apache Flume

http://flume.apache.org/

- *Flume configuration file is a Java property file with key-value pairs*
- *Since we can have multiple agents in Flume, we will configure each agent based on their unique name agent*
- *Each Flume agent has a source, channel and a sink*
- *Source*
  - *Tells the node where to receive data from*
- *Sink*
  - *Tells the node where to send data to*
- *Channel*
  - *A queue between the Source and Sink*
  - *Can be in-memory only or 'Durable'*
  - *Durable channels will not lose data if power is lost*
  - *In memory channels will lose data if power is lost*

Apache Flume – Twitter Exercise

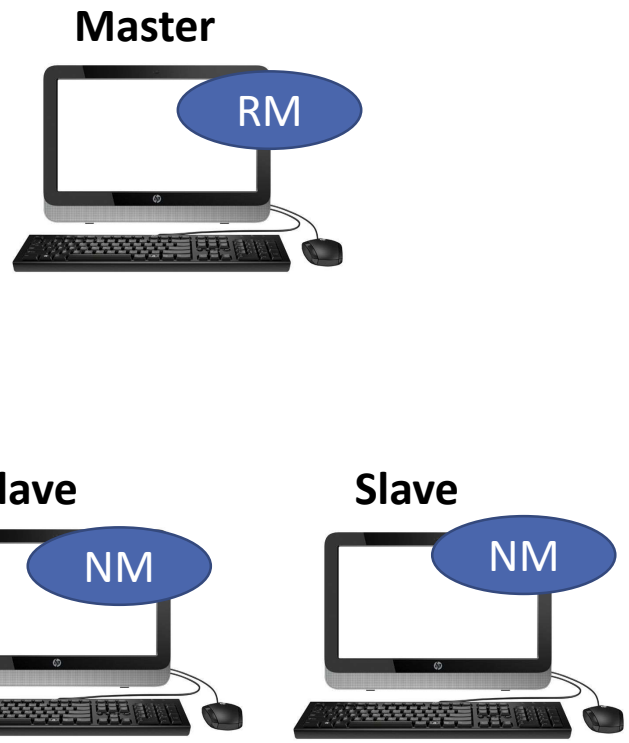
https://github.com/cloudera/cdh-twitter-example/blob/master/flume-sources/src/main/java/com/cloudera/flume/source/TwitterSource.java



Hadoop Data Processing Architecture - YARN

YARN Daemons

Master – Slave Architecture



RM ResourceManager

NM NodeManager

YARN Components

Application Job

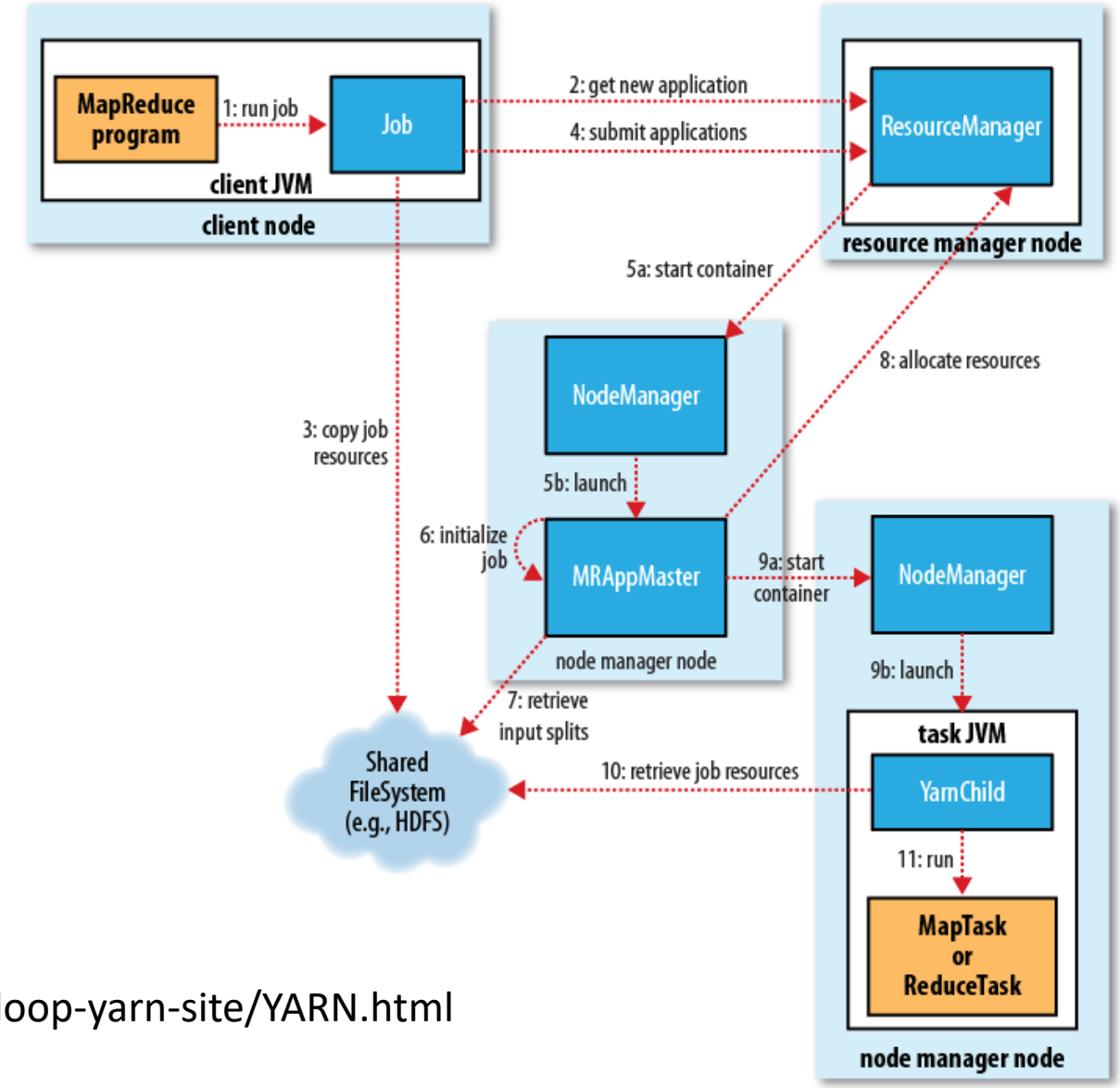
Map Task

Reduce Task

- *Resource Manager* — *1 / cluster* — *Scheduling + allocation of compute resources*
- *Application Master* — *1 / job* — *Monitoring*
- *Node Manager* — *1 / DN*
- *YarnChild* — *Compute Resources on NM also termed as "Resource Containers" = (CPU + RAM)*

  Execution of tasks

Anatomy of a MapReduce job run - YARN



http://hadoop.apache.org/docs/r2.7.4/hadoop-yarn/hadoop-yarn-site/YARN.html

MapReduce Programming

Introduction to MapReduce

- *Data processing paradigm*
- *With Hadoop 2.x, MapReduce is a YARN-based system for parallel processing of large data sets*
- *Involves 2 phases (Developer) – Map phase and a Reduce phase*
- *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

Introduction to MapReduce Programming

Problem Statement: WordCount → Count each word

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

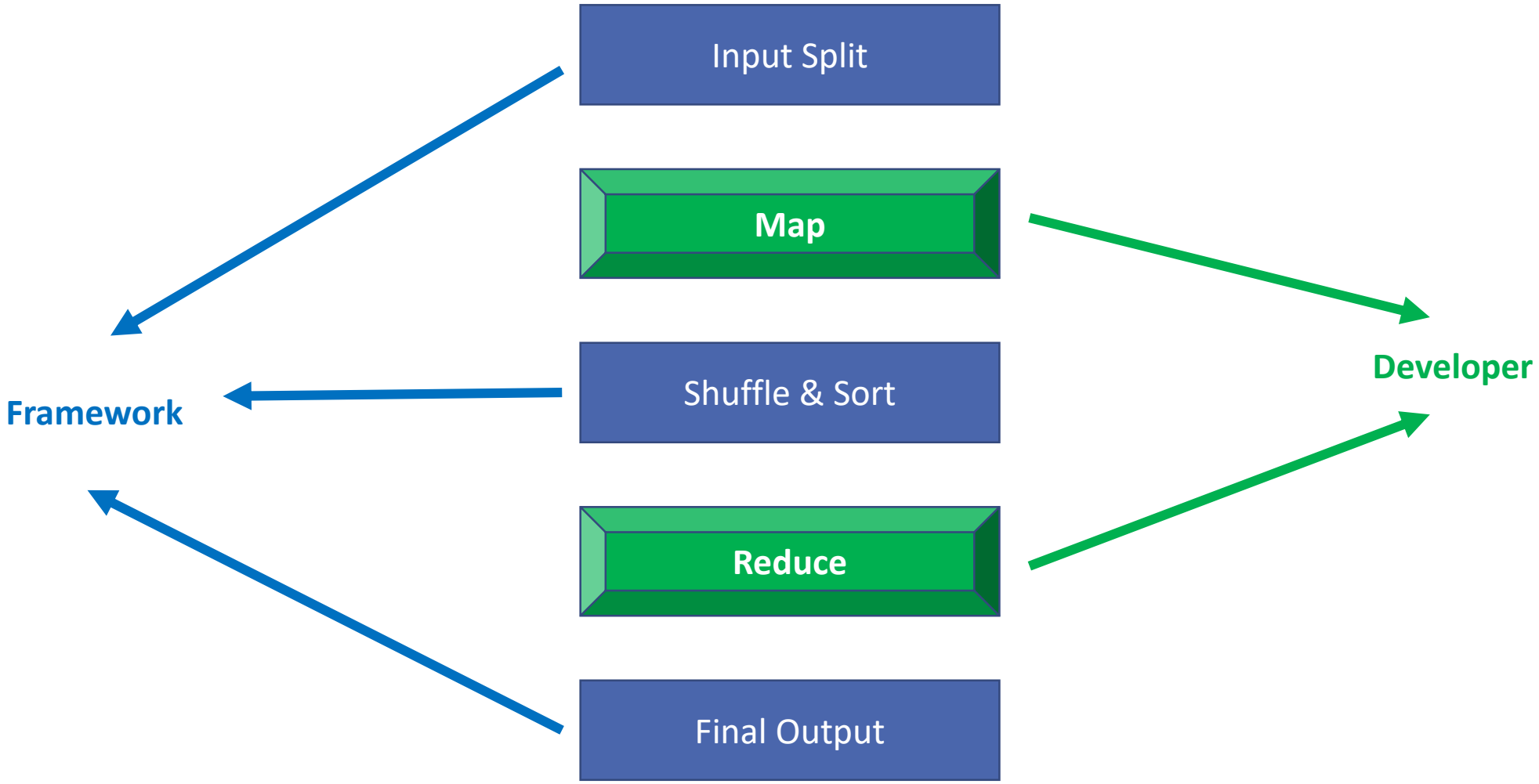
Hadoop	5
Learning	1
Welcome	1
buzz	1
fun	1
is	2
the	1
to	1

Steps in MapReduce

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

- Map – Transformation -> Convert into words
- Reduce – Aggregation -> Count the words

Steps in MapReduce



Hadoop API Documentation

VM Path : /home/user1/HadoopInstallations/hadoop-2.7.1/share/doc/hadoop/api/index.html

Map Signature

```
Mapper class {
        map (key, value, context) {
                ----------------------------------------
                ----------------------------------------
                    Logic for Transformation
                ----------------------------------------
                ----------------------------------------
                context.write(k, v);
        }
}
```

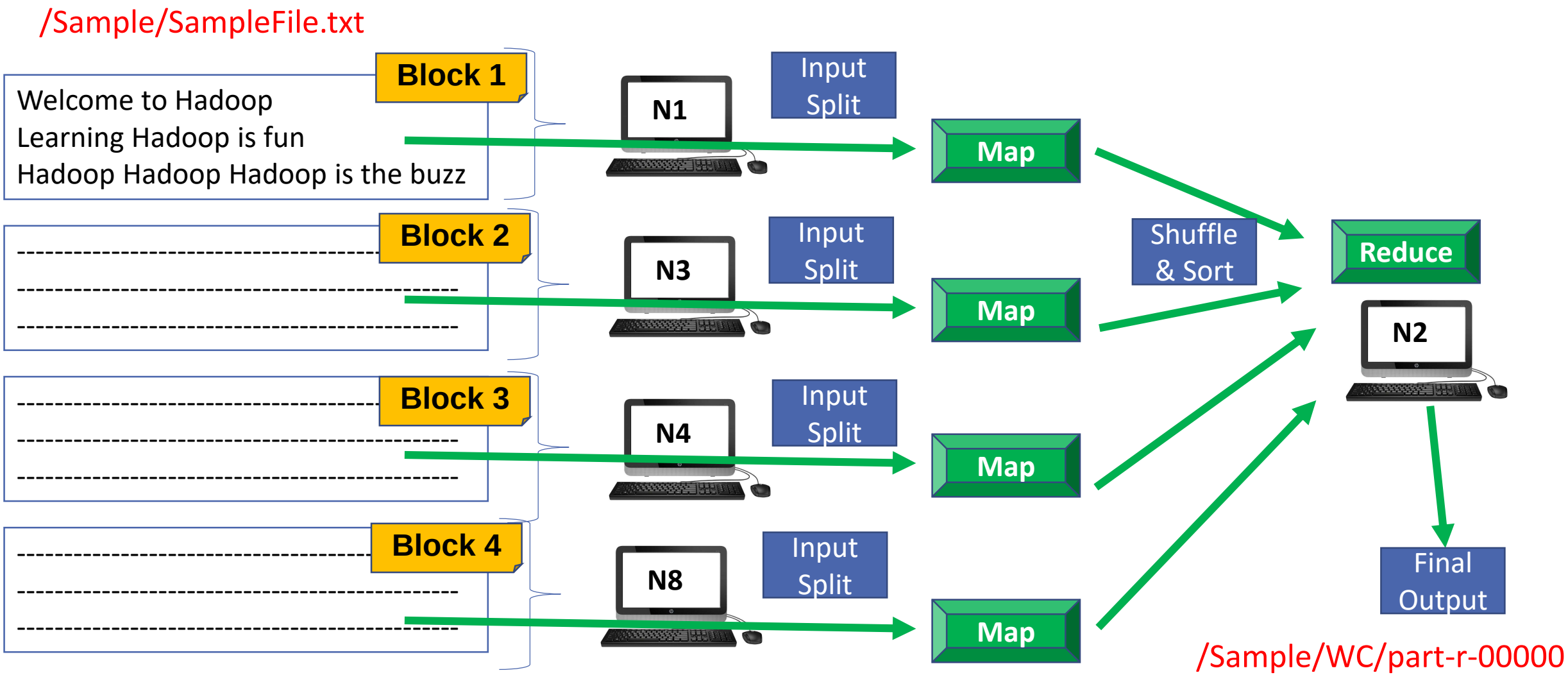
(K1, V1) ➔ Map ➔ List(K2, V2)

Reduce Signature

```
Reducer class {
        reduce (key, list{values}, context) {
                ----------------------------------------
                ----------------------------------------
                    Logic for Aggregation
                ----------------------------------------
                ----------------------------------------
                context.write(k, v);
        }
}
```

(K2, List{V2}) ➔ Reduce ➔ List(K3, V3)

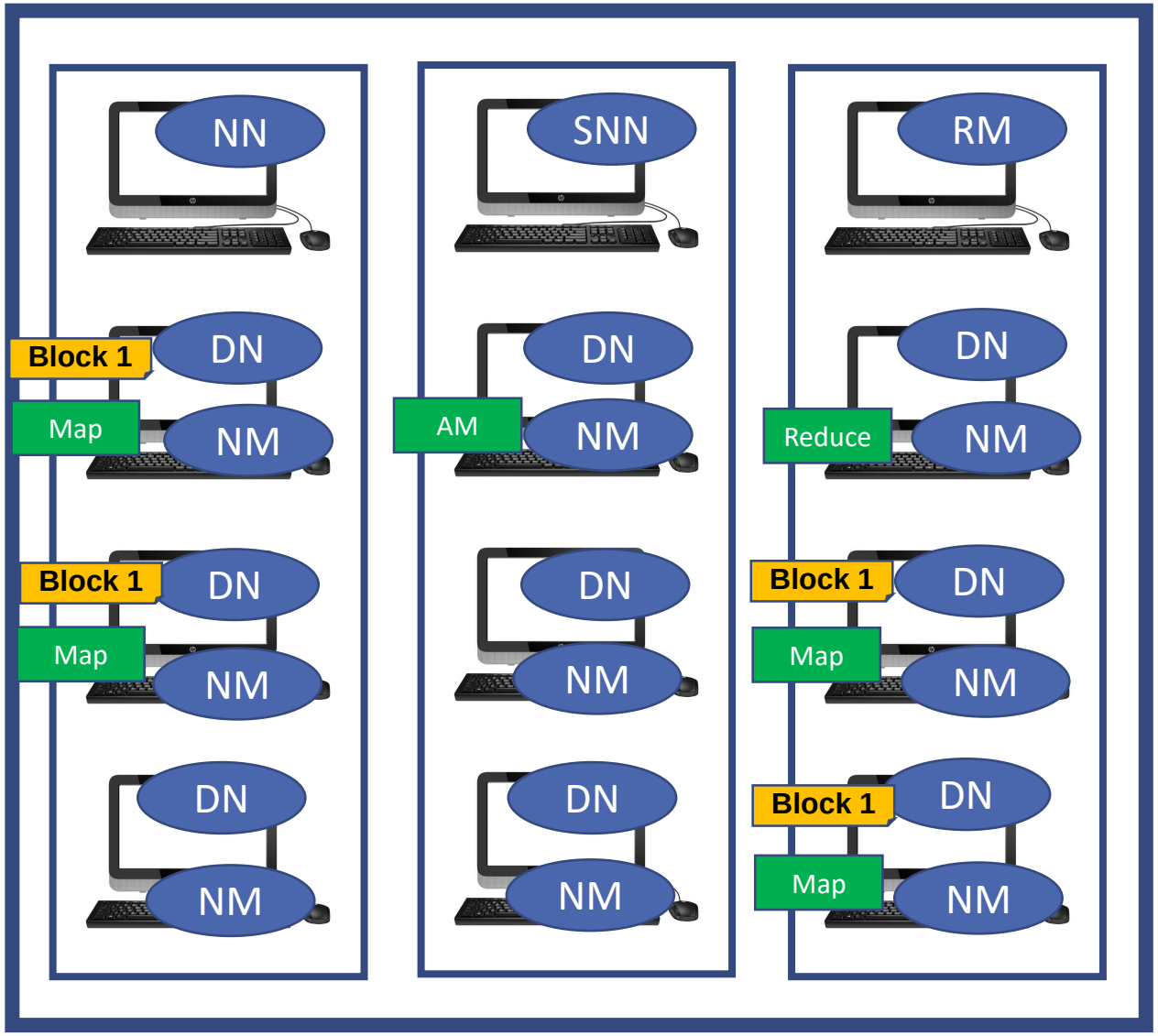
MapReduce Steps on a cluster



MapReduce program execution on a Hadoop Cluster

Hadoop binaries and configs

Gateway / Client Node



MapReduce Datatypes

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

MapReduce Execution Modes

- *Cluster Mode*
- *Local Mode*

How to execute a MapReduce program on a cluster?

```
$ yarn jar /home/user1/Documents/wc.jar WordCount /Sample/SampleFile.txt
/Sample/WC
```

Input Formats MapReduce

Input Format Type	Description
TextInputFormat	An InputFormat for plain text files. Files are broken into lines. Either linefeed or carriage-return are used to signal end of line. Keys are the position in the file, and values are the line of text.
SequenceFileInputFormat	SequenceFiles are flat files consisting of binary key/value pairs.
NLineInputFormat	NLineInputFormat which splits N lines of input as one split.
KeyValueTextInputFormat	Files are broken into lines. Either line feed or carriage-return are used to signal end of line. Each line is divided into key and value parts by a separator byte. If no such a byte exists, the key will be the entire line and value will be empty.
FixedLengthInputFormat	Input format used to read input files which contain fixed length records. The content of a record need not be text. It can be arbitrary binary data.
CombineFileInputFormat	Splits are constructed from the files under the input paths. A split cannot have files from different pools. Each split returned may contain blocks from different files.

Output Formats MapReduce

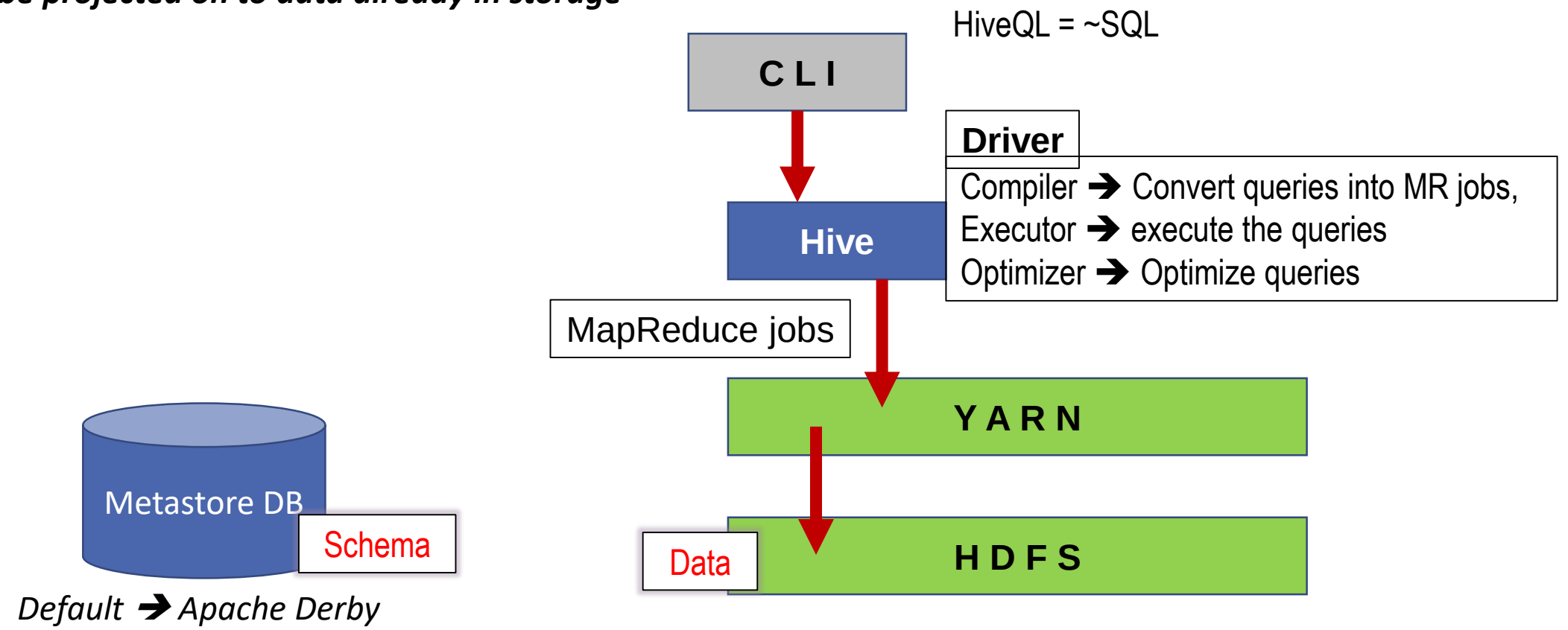
Output Format Type	Description
TextOutputFormat	An OutputFormat that writes plain text files
SequenceFileOutputormat	An OutputFormat that writes flat files consisting of binary key/value pairs.
MapFileOutputFormat	It writes MapFiles as the output. The keys in a MapFile must be added in an order, following which the reducer will emit keys in the sorted order

Agenda – Day 3

- Hive
- Introduction to NoSQL and HBase
- Fundamentals of Hadoop Administration – Cloudera Manager Hadoop Setup (Multi Node)

Hive Architecture

- ***Data and Schema are stored separately, the data is validated against the schema when it is queried, a technique called "Schema on Read" ➔ Flexibility***
- ***Structure can be projected on to data already in storage***



Hive Tables

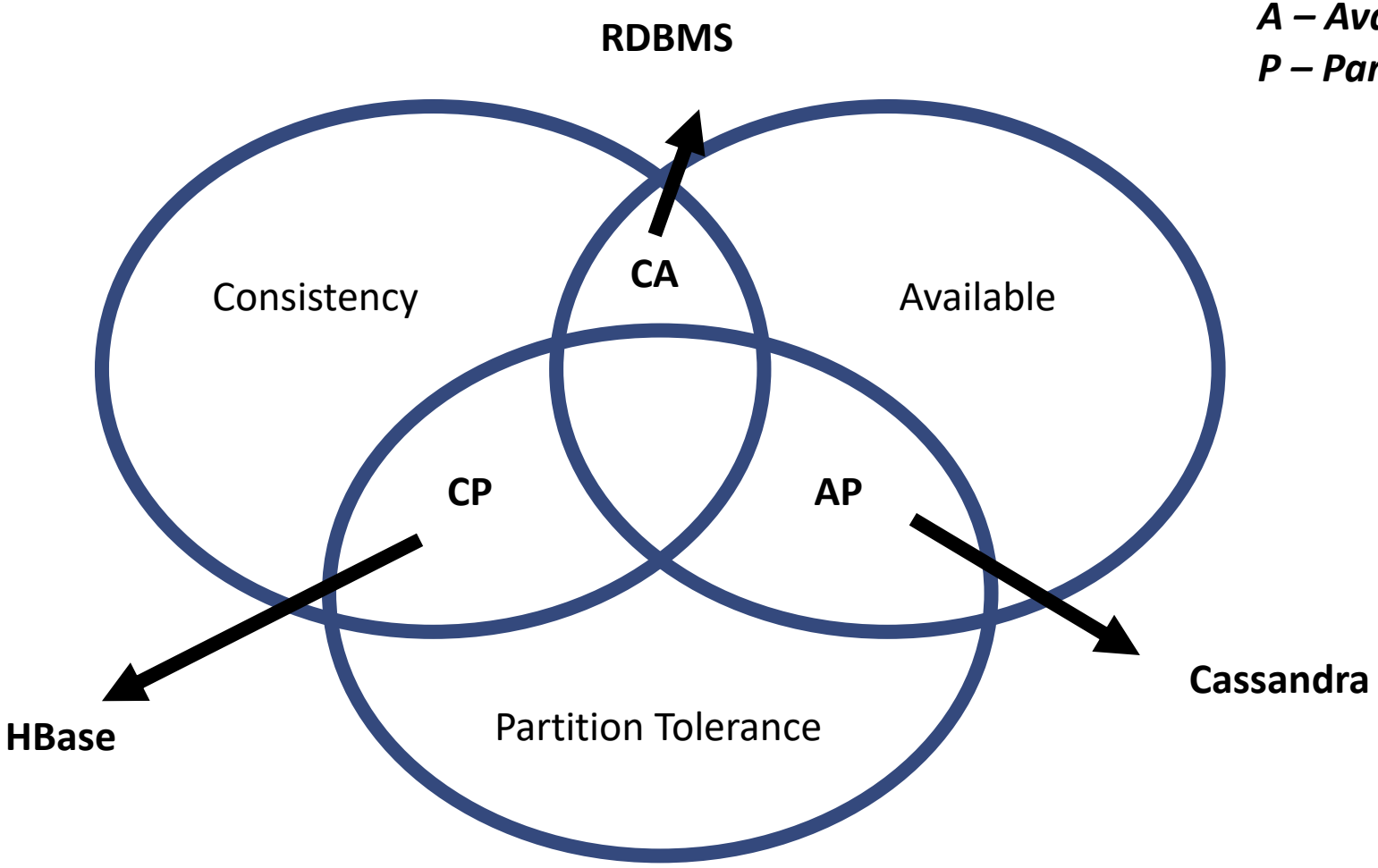
- *Managed Table*
  - *Default tables*
  - *Hive manages ‘schema’ and ‘data’*

/user/hive/warehouse ➔ Hive’s default warehouse dir

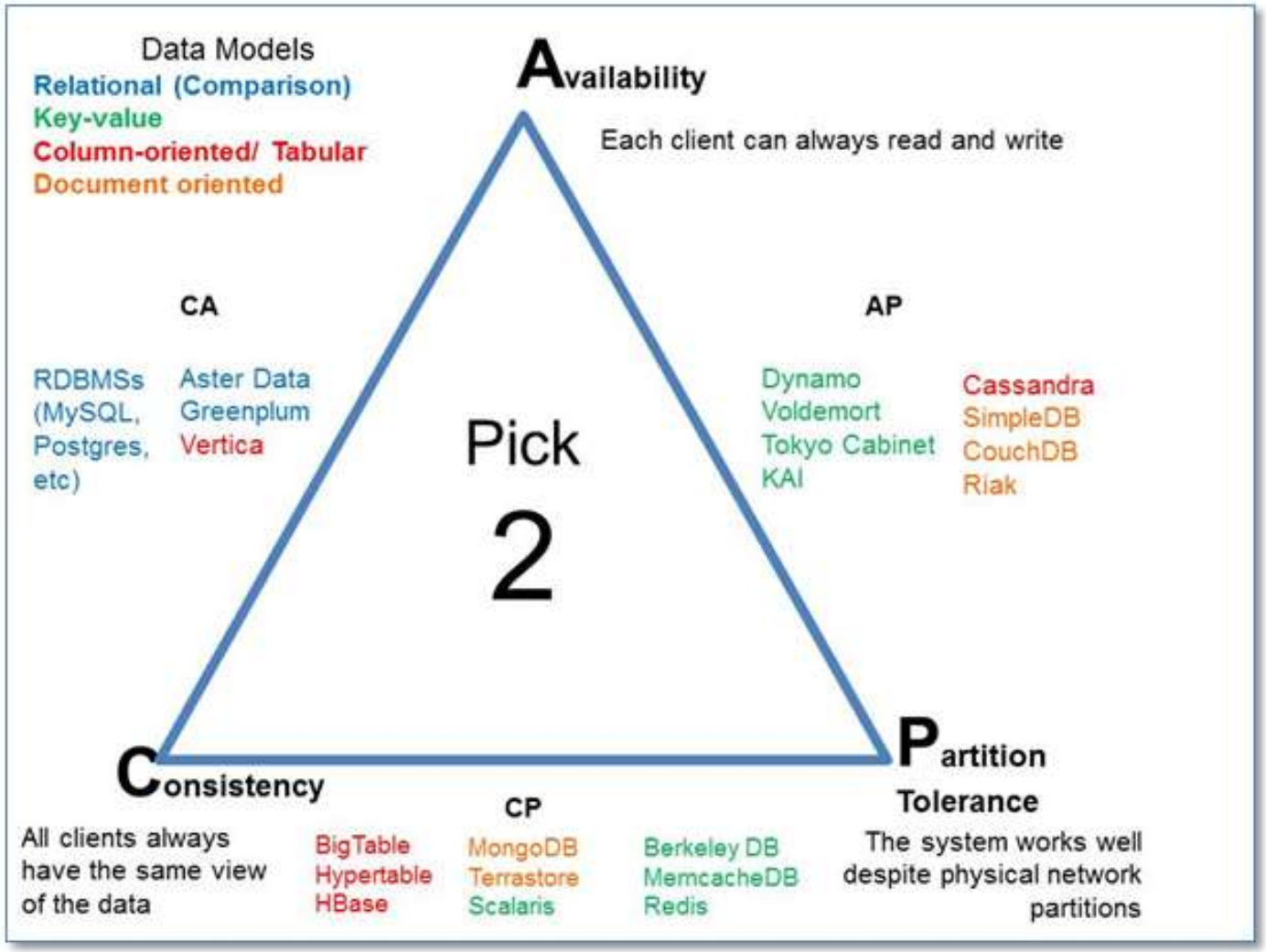
- *External Table*
  - *‘external’ keyword in DDL*
  - *Hive manages ‘schema’*

Brewer's Theorem ➔ CAP Theorem

C – Consistency
A – Availability
P – Partition Tolerance



CAP Theorem


Data Models
Relational (Comparison)
Key-value
Column-oriented/ Tabular
Document oriented
Availability
Each client can always read and write
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
Pick 2
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Consistency
All clients always have the same view of the data
CP
BigTable
Hypertable
HBase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance
The system works well despite physical network partitions

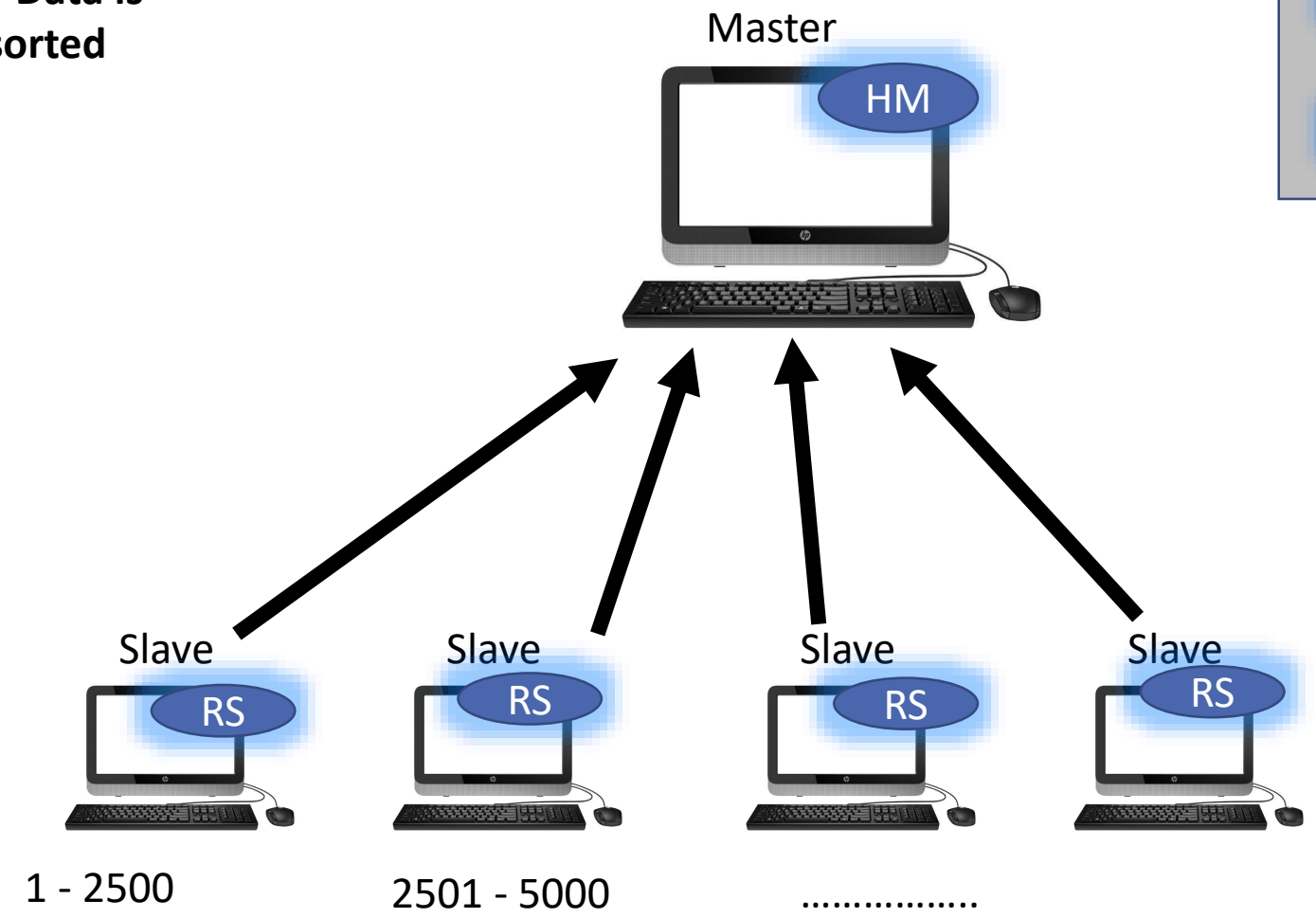
Apache HBase

http://hbase.apache.org

- *Apache HBase is Hadoop's NoSQL engine*
- *HBase is Google's Big Table clone*
- *No MapReduce (Batch Processing – High Latency)*
- *HBase offers Random Read / Write capabilities on HDFS*
- *Low Latency operations*
- *Imagine 1 Big Table with 1 billion rows X 1 million columns that are sparse (Row 1 can have 2 columns, Row 2 can have 200), with capabilities to add columns on the fly*
- *HBase data model introduces a concept of 'Column Family'*
- *Use HBase when you need "operational capabilities" to your existing Decision Support System*

Apache HBase

Row Key – Data is always sorted



WAL – Write Ahead Log
Memstore – (In mem WAL)
HFile - Storage

WebUI Port - HMaster

- *HMaster – http://localhost:16010*

Hadoop Setup

Infra

- In premise
- **Cloud – AWS** / GCP / Azure…
- Virtualization

OS

- RHEL
- CentOS
- **Ubuntu**
- Fedora
- SUSE
- …..

JDK

- Open JDK
- **Oracle JDK**
- IBM JDK
- ….

Hadoop

- **Cloudera**
- Hortonworks
- Apache
- MapR
- Big Insights

Hadoop Setup Mode

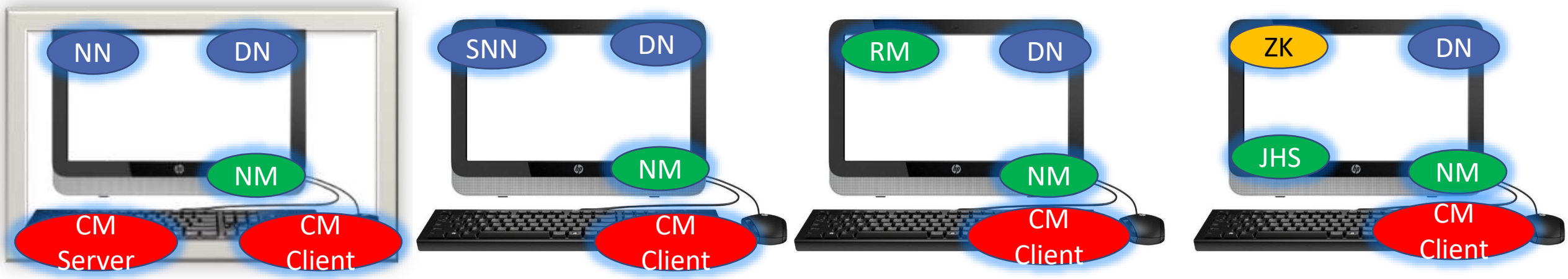
- Standalone Mode
- Pseudo Distributed Mode
- **Fully Distributed Mode**

https://www.cloudera.com/documentation/enterprise/5-9-x/topics/cm_ig_install_path_a.html#cmig_topic_6_5

Cloudera Manager Installer

wget https://archive.cloudera.com/cm5/installer/latest/cloudera-manager-installer.bin

Cluster Topology – Plan your cluster



N1 **N2** **N3** **N4**

References

https://wiki.apache.org/hadoop/Hadoop2OnWindows

https://www.cloudera.com/documentation/kafka/latest.html

https://github.com/cloudera/cdh-twitter-example

http://blog.cloudera.com/blog/2012/12/how-to-use-a-serde-in-apache-hive/