

# Big Data

## Understanding Hadoop and its ecosystem

Nagabhushan



# Agenda – Day 1

- Introduction to Big Data & Hadoop
- Hadoop - Use Cases & History
- Hadoop Setup
- Working with Hadoop FsShell
- Hadoop Architecture

# Analytics on traditional systems

Customer

E Commerce

Traditional System

Business Owner

What products are NOT selling? Why?

Business Intelligence

SAP BO  
OBIEE  
MSBI  
QlikView  
Tableau  
Pentaho

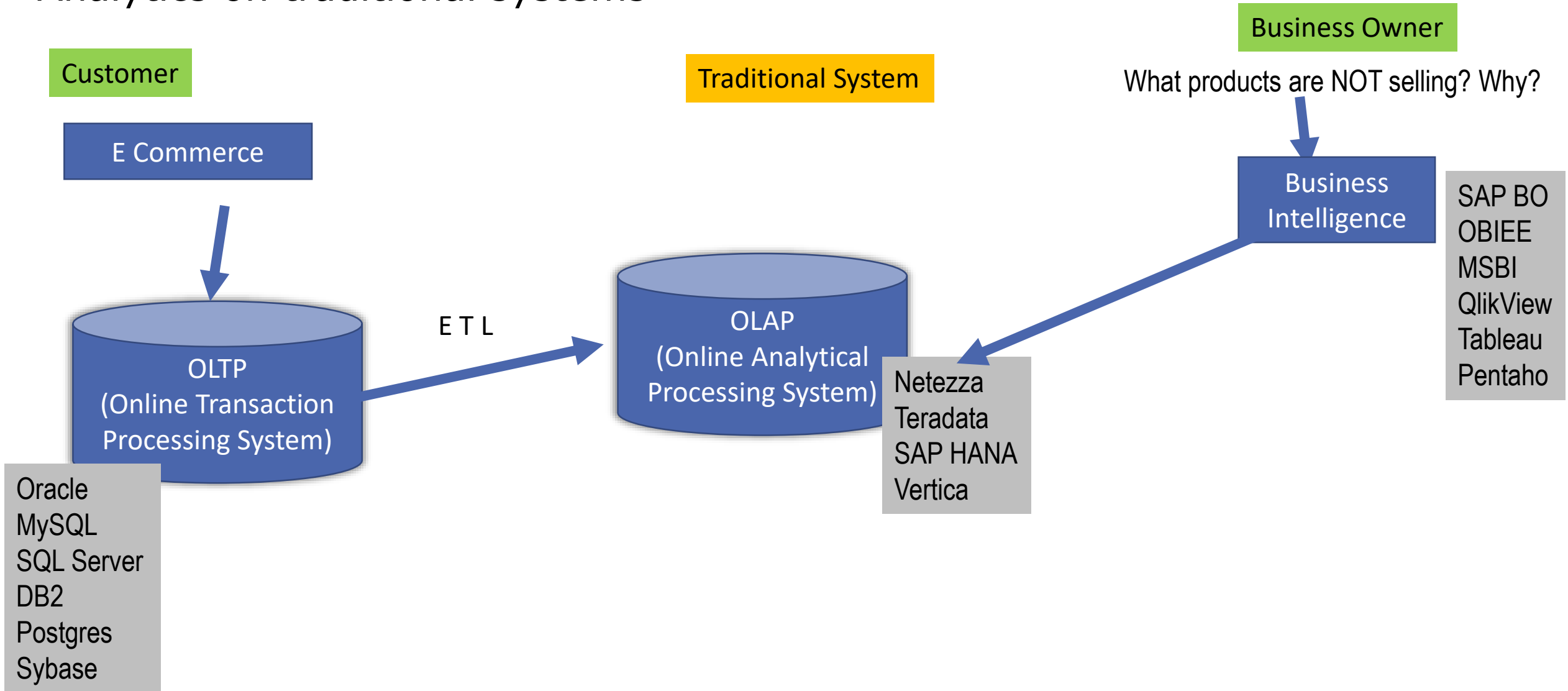
Netezza  
Teradata  
SAP HANA  
Vertica

ETL

OLAP  
(Online Analytical  
Processing System)

OLTP  
(Online Transaction  
Processing System)

Oracle  
MySQL  
SQL Server  
DB2  
Postgres  
Sybase



# What is the traditional system?

- *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*
  - *Architecture is RDBMS*
  
  - **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*
  - *Architecture is RDBMS*

# What is Big Data? → Problem

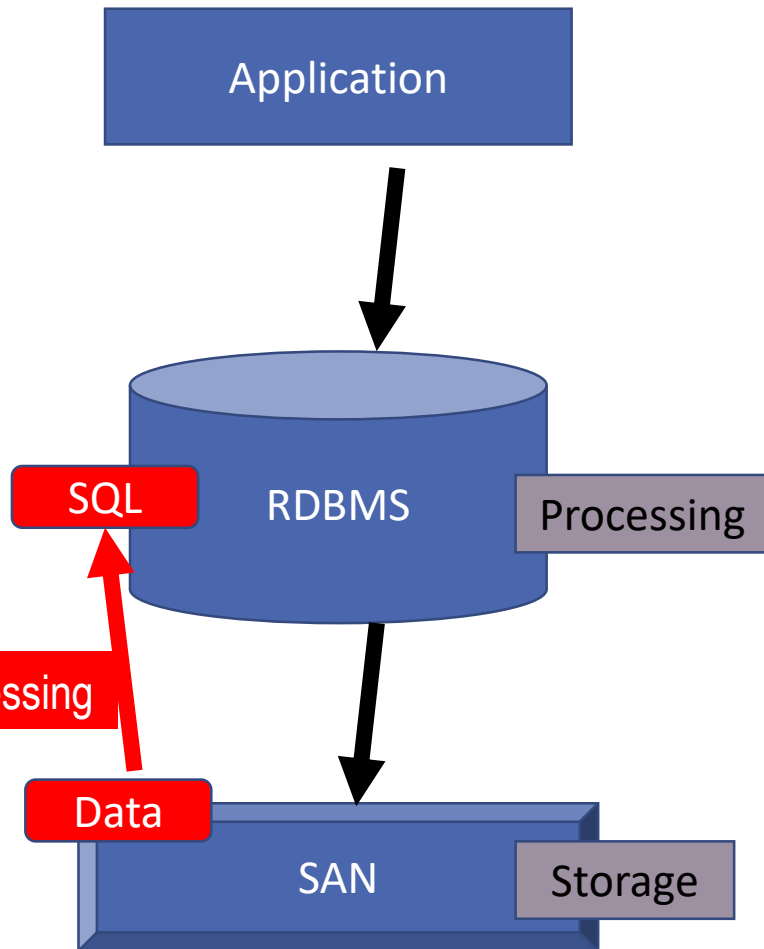
- *3 Vs of Big Data*
  - *Volume → Size*
  - *Velocity → Speed*
  - *Variety → Different forms of data*
- *Hadoop's V → VALUE*
- *How to store Big Data? → HDFS*
- *How to process Big Data? → MapReduce (Hadoop 1.x) / YARN (Hadoop 2.x)*

# Data Measurement Scale

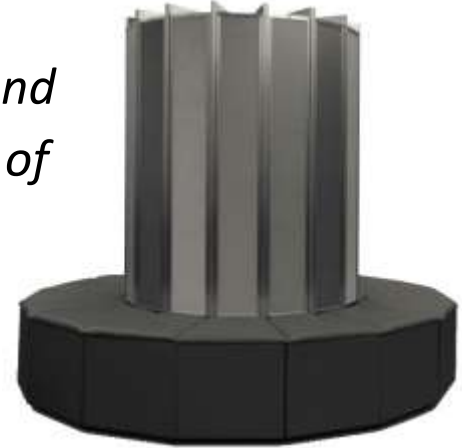
▣ 1 Kilobyte	KB	1000
▣ 1 Megabyte	MB	1000000
▣ 1 Gigabyte	GB	1000000000
▣ 1 Terabyte	TB	1000000000000
▣ 1 Petabyte	PB	1000000000000000
▣ 1 Exabyte	EB	1000000000000000000
▣ <u>1 Zettabyte</u>	<u>ZB</u>	<u>1000000000000000000000 X 16</u>
▣ 1 Yotabyte	YB	1000000000000000000000000000

# Problems with the traditional system

Monolithic Computing



- *Computation has been processor bound*
- *Intense processing on small amounts of data*
- *Goal was to have a bigger, powerful machine*
  - *Faster processor, more RAM*
- *Limitations to this approach*
  - *High Cost*
  - *Scalability*
  - *Latency (when the data is big)*



1 TB / day

Expensive Hardware

# Challenges with the traditional system

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



# Big Data Systems to the rescue → Hadoop

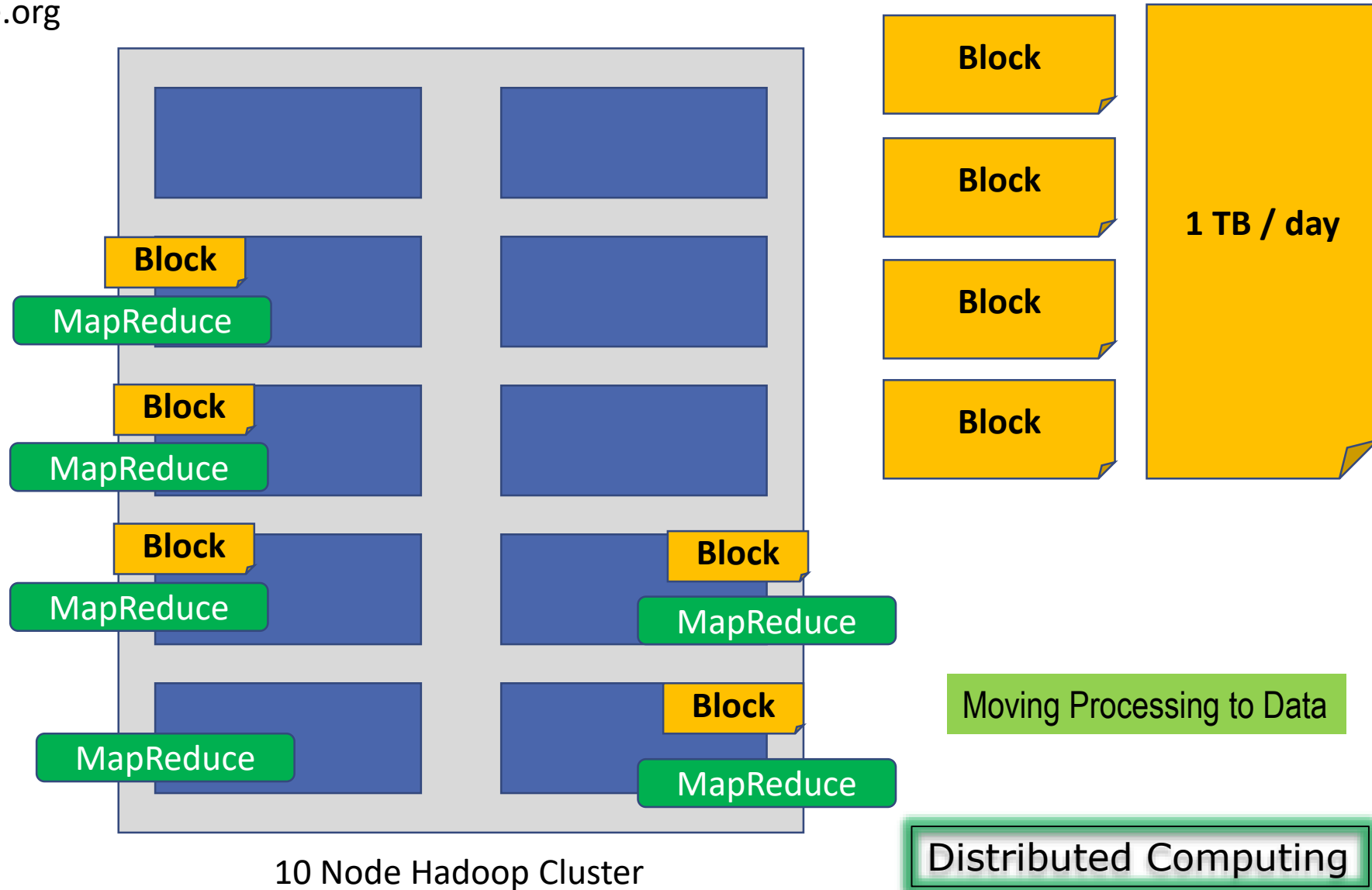
<http://hadoop.apache.org>

## 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



# Hadoop Layout / Node



Yet Another Resource Negotiator

Hadoop's Distributed File System

Java

Linux

Commodity Hardware

YARN

Processing

HDFS

Storage

J R E

OS

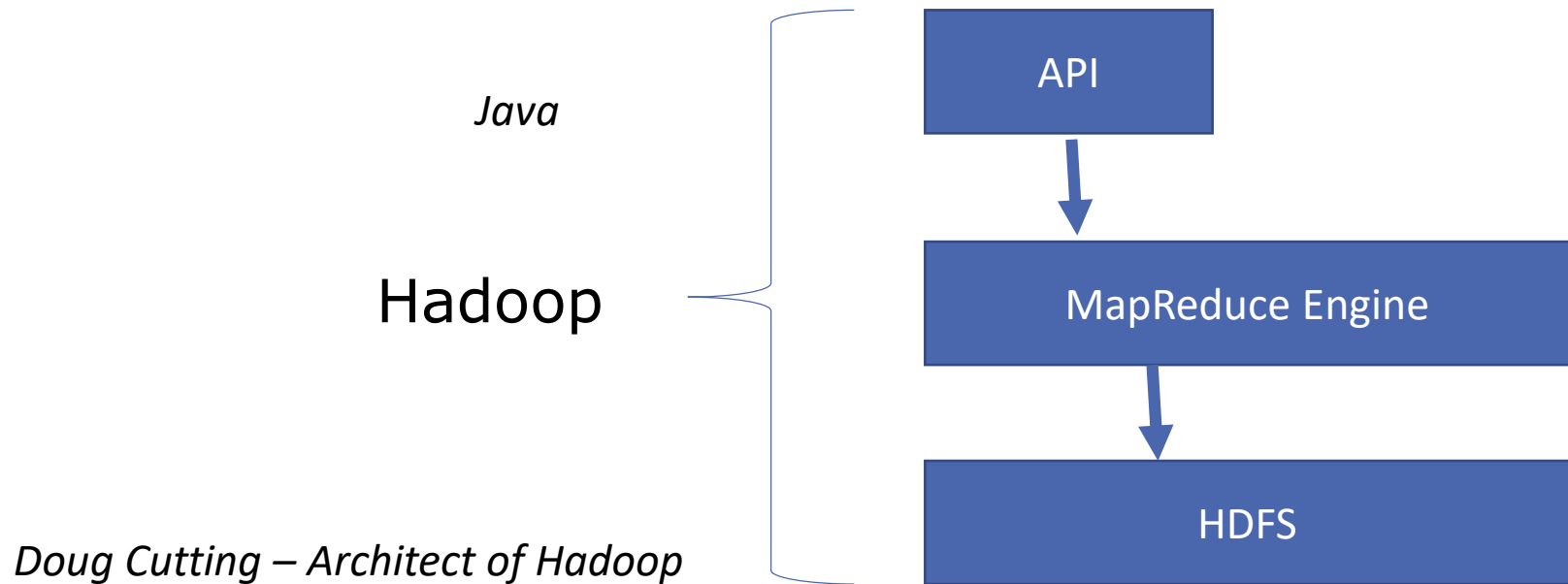
Infra

# Features of Hadoop

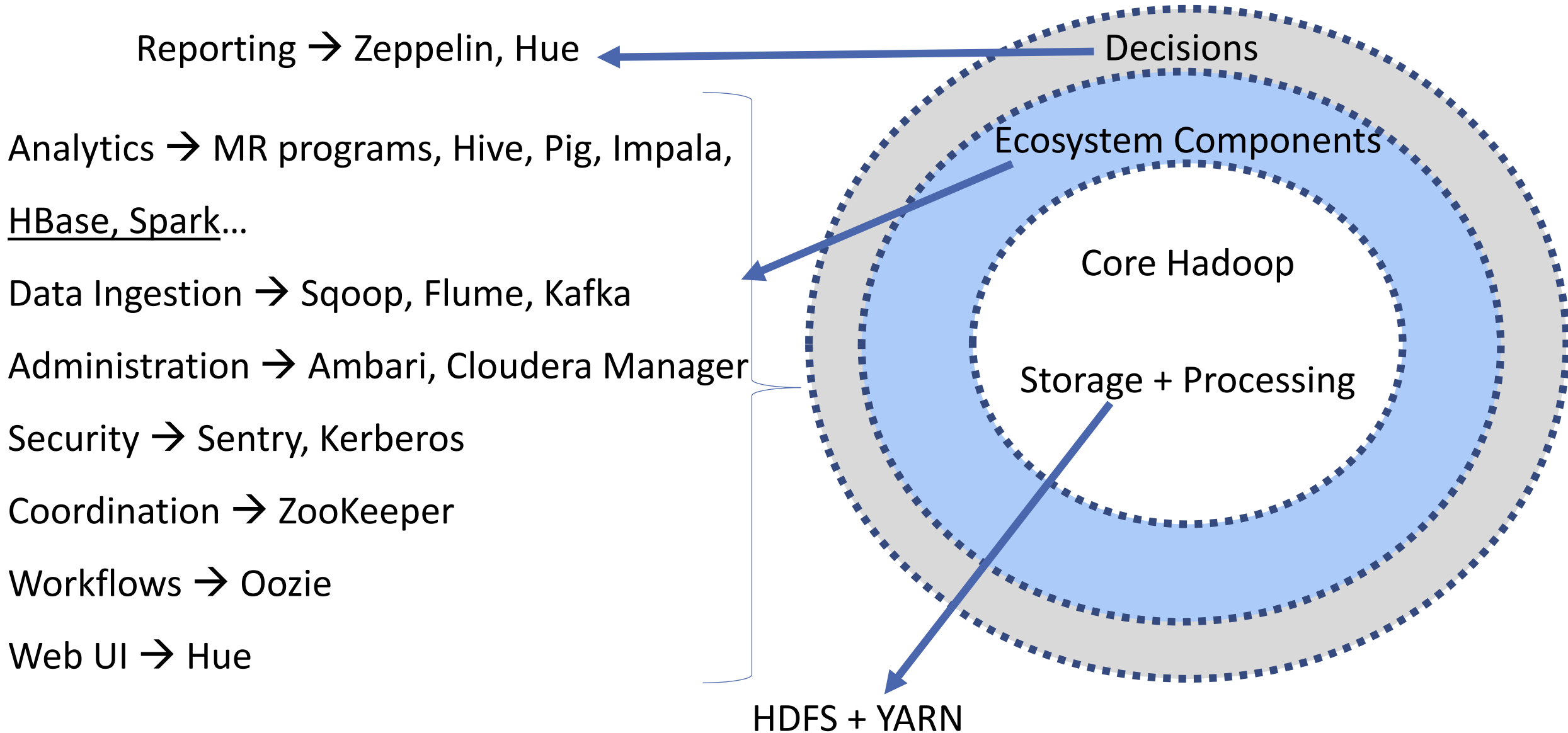
- *Any infra*
- *Open Source* <http://hadoop.apache.org/>
- *Distributed Storage - Parallel Processing*
- *Fault Tolerance - Replication - Hardware is prone to failure, high availability is handled by the framework*
- *Scale out architecture - Horizontal Scaling (Unlimited nodes in a cluster)*
- *Each node is compute capable*
- *Data Locality - a paradigm shift of moving compute to data*
- *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 (Write Once Read Many)*

# 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*



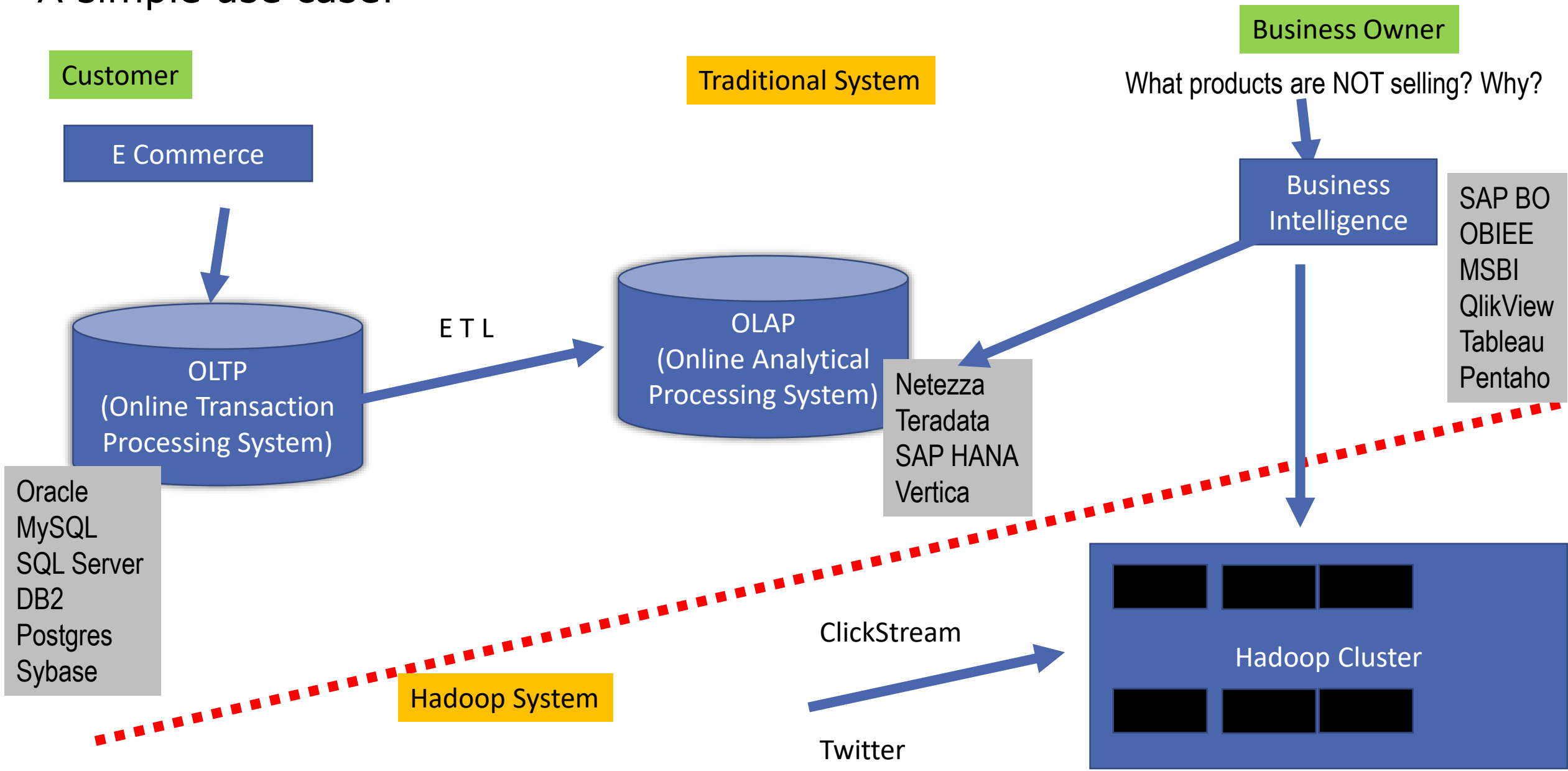
# Hadoop Ecosystem



# Commercial Distributions of Hadoop

- *Cloudera* <https://www.cloudera.com/>
- *Hortonworks* <https://hortonworks.com/>
- *MAPR* <https://mapr.com/>
- *Big Insights (IBM)* <https://www.ibm.com/analytics/us/en/technology/biginsights/>
- *Apache Hadoop → Open Source* <http://hadoop.apache.org/>

# A simple use case!



# Hadoop Terminologies

Cluster

Rack

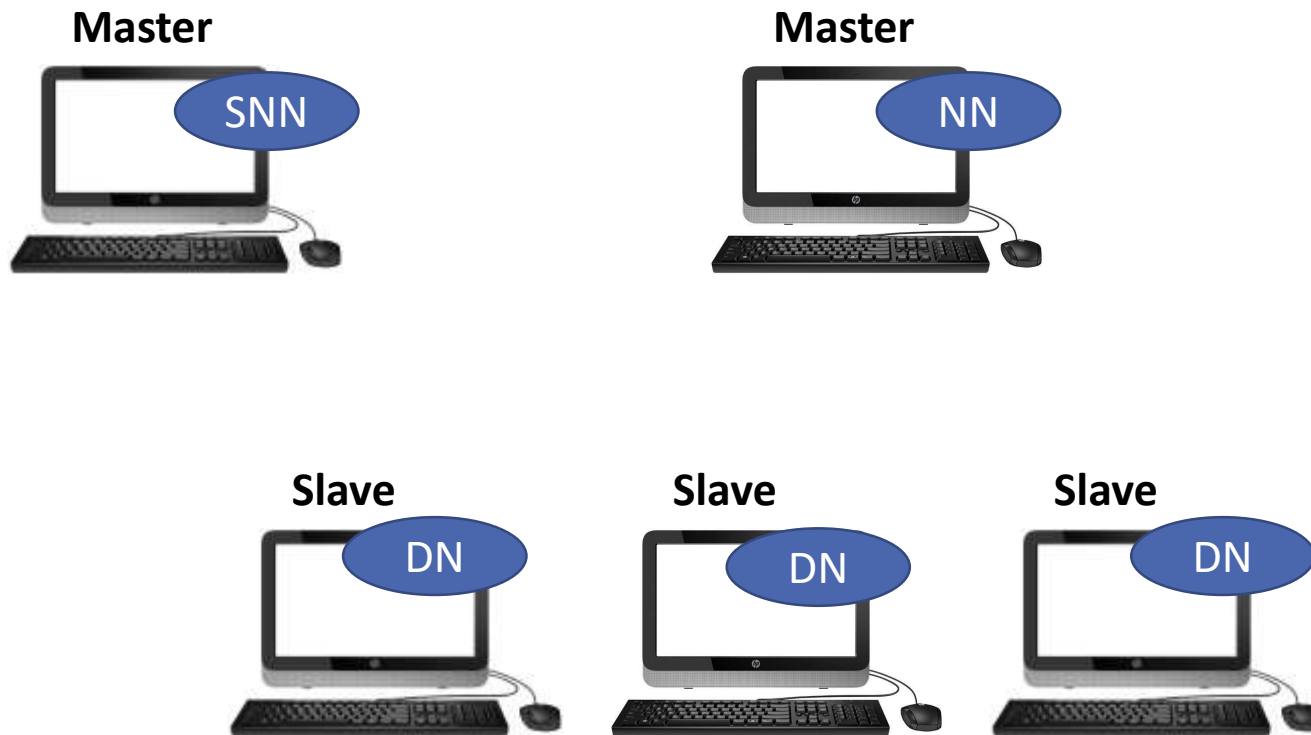
Node





# HDFS Daemons

Master – Slave Architecture



NN	NameNode
DN	DataNode
SNN	Secondary NameNode

# HDFS Components

File

Data

- *NameNode*

*1 / cluster*

*Manage distribution, replication,  
maintain metadata*

B1

Block

B2

Block

- *Secondary NameNode*

*1 / cluster*

*Checkpoint Node*

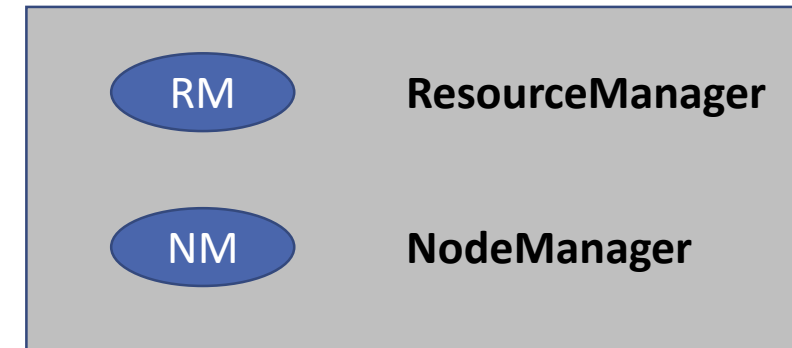
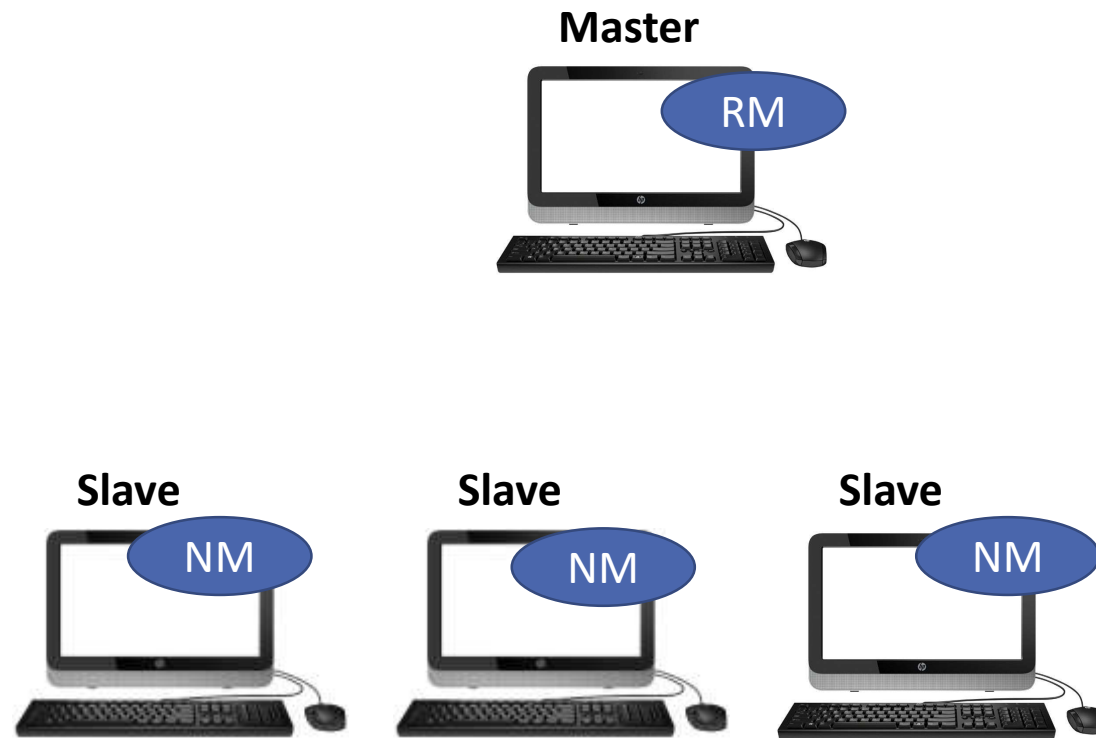
- *DataNode*

*1 / Node*

*Store DataBlocks, Data Integrity*

# YARN Daemons

Master – Slave Architecture



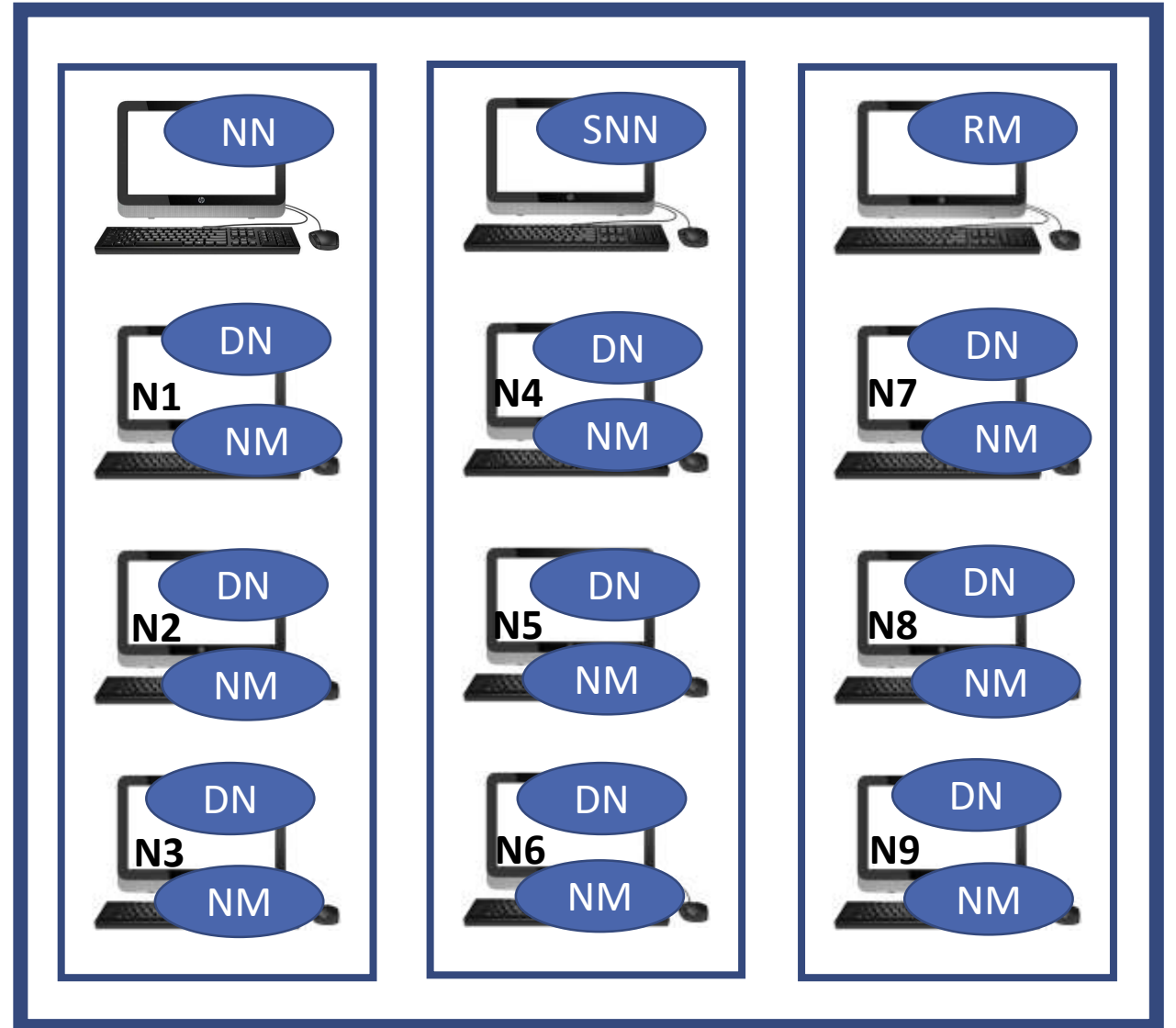
# Hadoop Daemons distributed over a cluster

*Hadoop Binaries and configs*



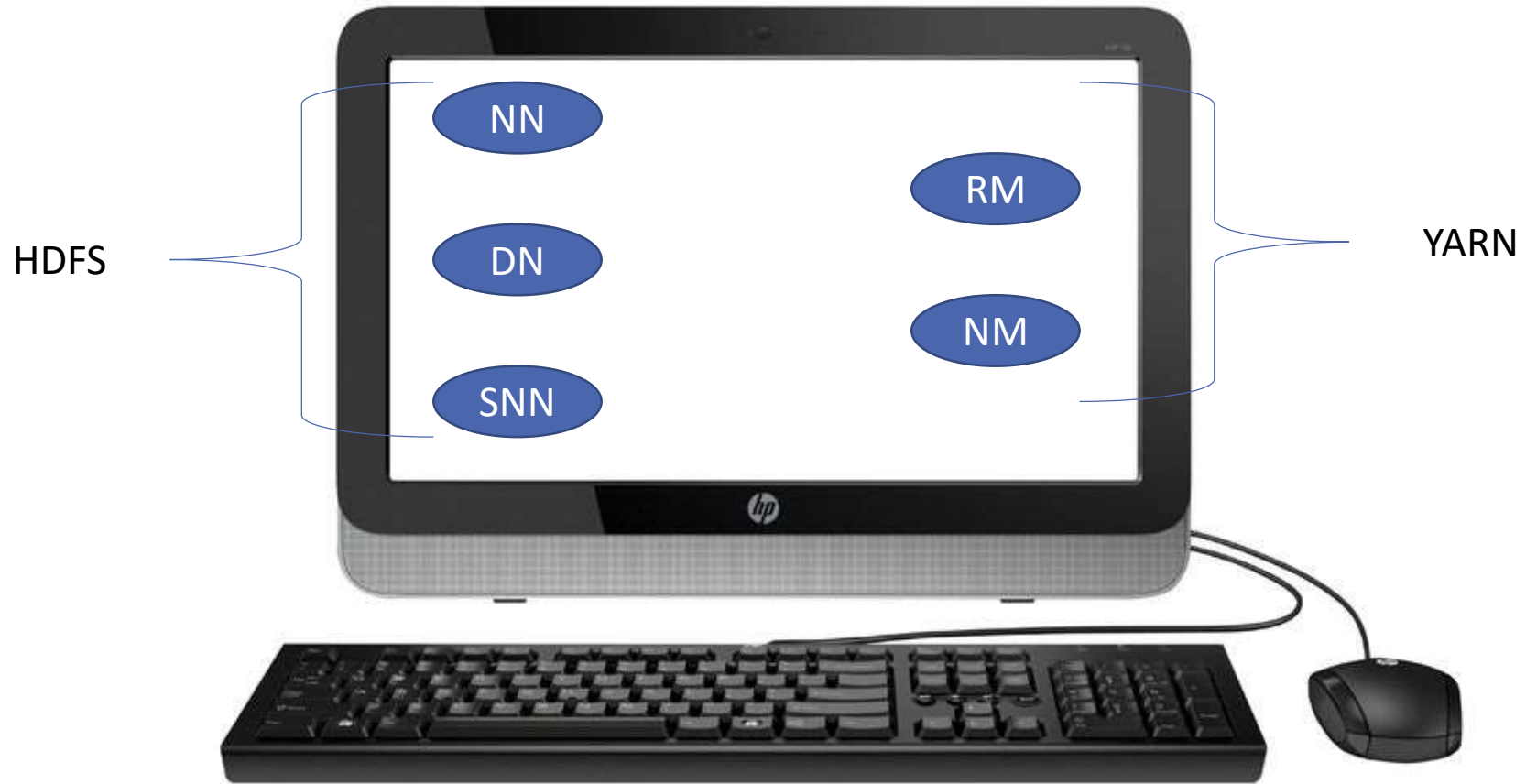
Gateway / Client Node

DataNode and NodeManager co-exist



# Hadoop Daemons distributed over a single node

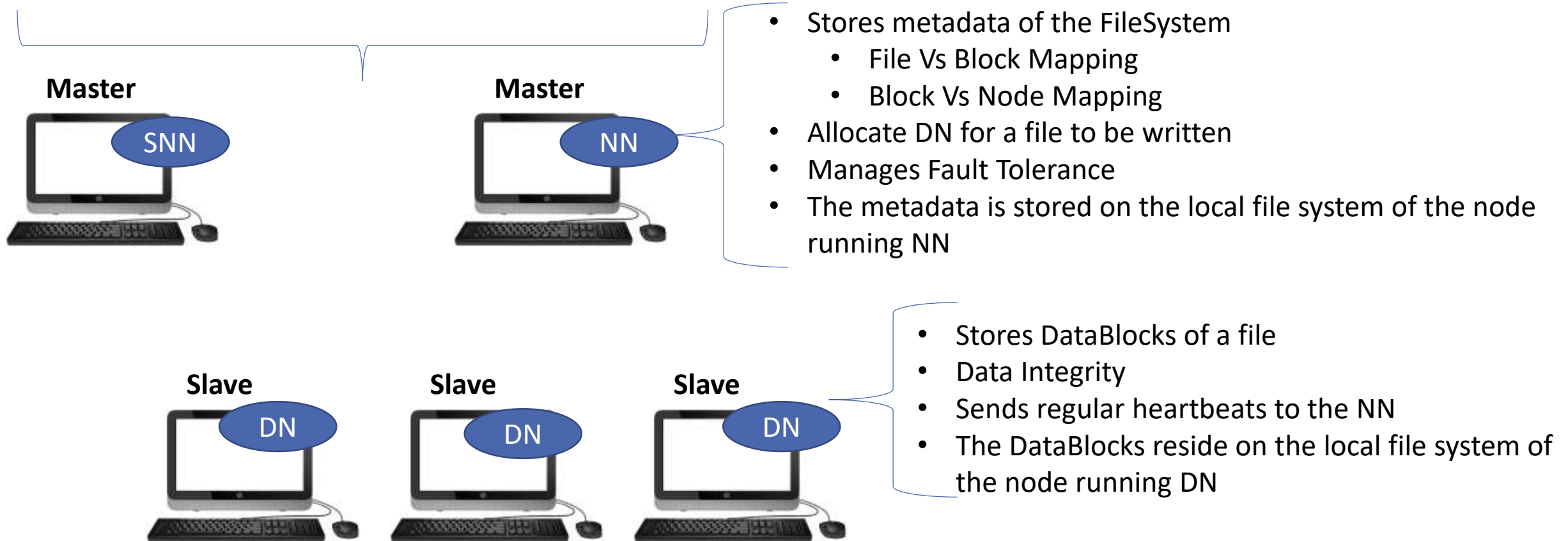
Pseudo Distributed Mode



# HDFS Daemons - Responsibilities

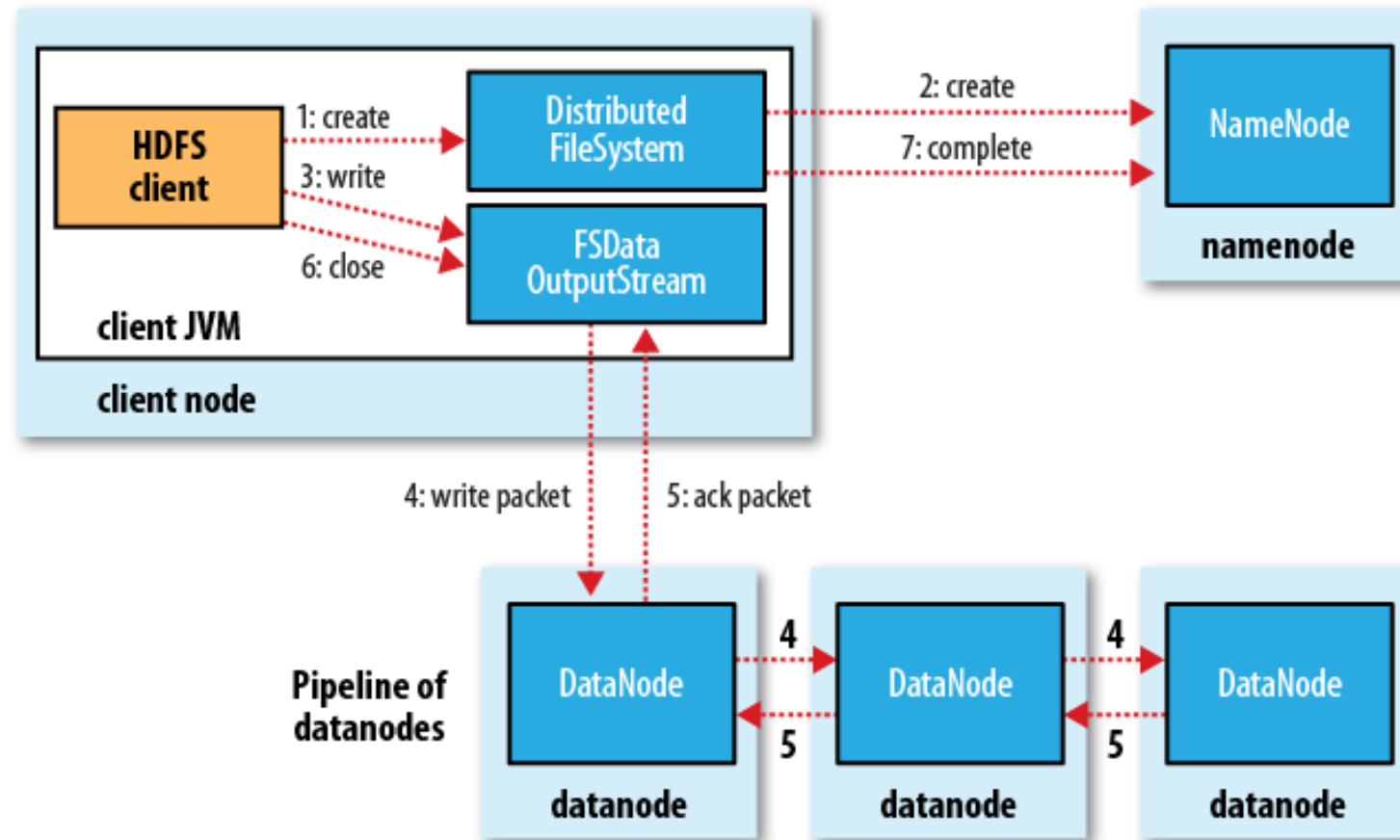
## Master – Slave Architecture

- Checkpointing
  - Merge EditsInProgress with FSImage at regular intervals

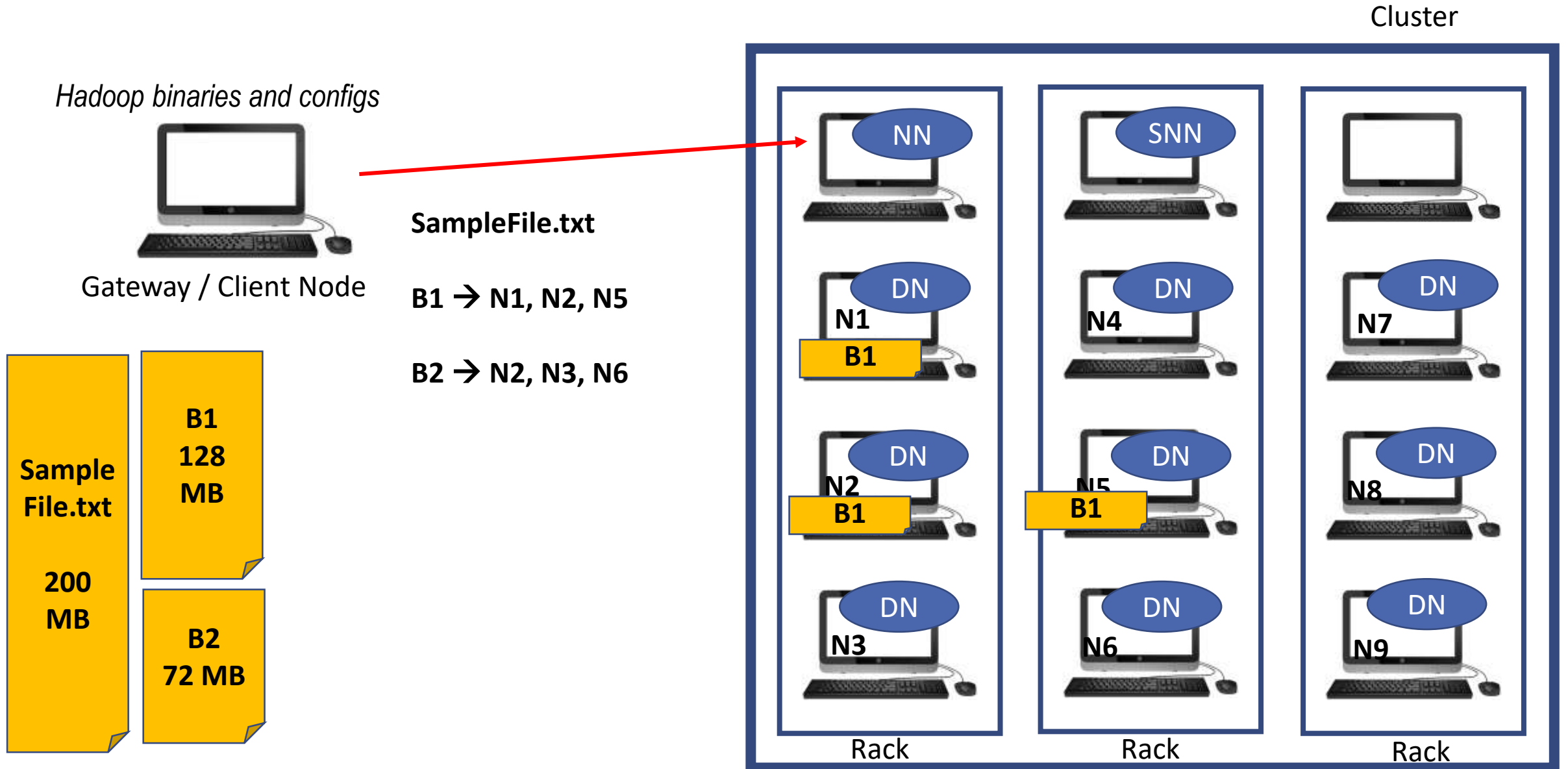


# Anatomy of a File Write

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



# HDFS Daemons distributed over a cluster





# Rack Awareness

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

## Infra

- In premise
- Cloud – AWS / GCP / Azure...
- **Virtualization**  
<http://virtualbox.org/>

## OS Distribution

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

## Java Distribution

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

## Hadoop Distribution

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

## Hadoop Setup Mode

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

# Lab Setup

- *Have a machine with following configurations*
  - *64 bit Hardware – CPU should be VT / Virtualization enabled in BIOS*
  - *64 bit operating System – Windows 7 & above or Mac*
  - *Minimum 8 GB RAM*
  - *25 GB of Hard disk space*
- *Oracle VM VirtualBox installed from <http://virtualbox.org/>*
- *Cloudera VM to be downloaded from [https://www.cloudera.com/downloads/quickstart\\_vms/5-12.html](https://www.cloudera.com/downloads/quickstart_vms/5-12.html)*

# Hadoop Setup Mode

- Standalone Mode

- Single Node, non distributed (Default)
- Hadoop works as a single Java Process

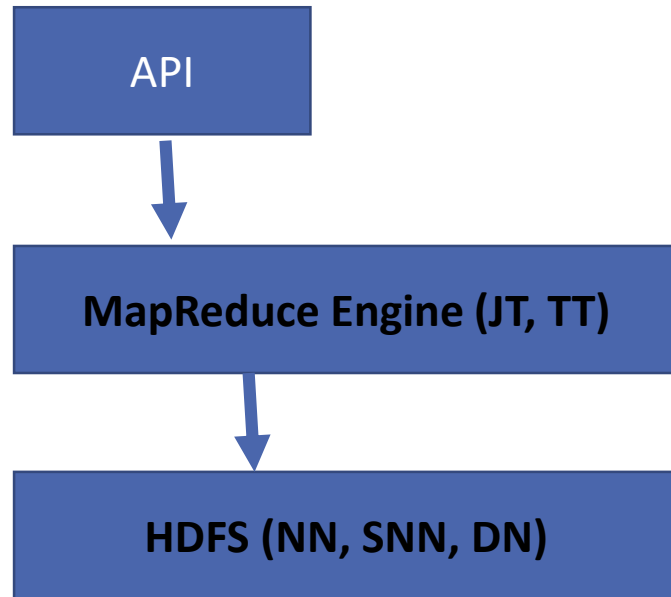
- *Pseudo Distributed Mode*

- *Single Node, distributed*
- *Each Hadoop daemon runs inside a separate JVM*
  - *HDFS → 1 NN, 1 DN, 1 SNN*
  - *YARN → 1 RM, 1 NM*

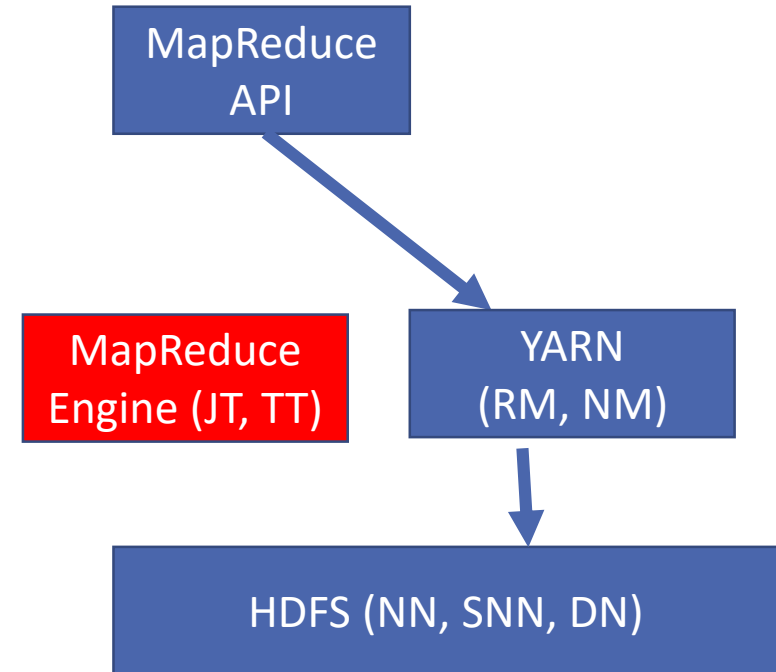
- Fully Distributed Mode

- Multi Node, distributed
- Hadoop daemons are distributed among many nodes → Typical production environment

# Hadoop 1.x



# Hadoop 2.x



# 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/ - Error`
- `$ 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/`
- `$ sudo watch 'jps | grep -v jps | sort -k 2'`

# NN & SNN

Master



Metadata is stored in these 2 files

- *FileSystem Image* → `fsimage_00000000000000000019`
- *Edit Log / Transaction log* → `edits_inprogress_00000000000000000020`

Master



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

- *FileSystem Image* → `fsimage_00000000000000000019`

- *SNN is not a hot backup for the NN*
- *In a scenario where NN failed, the administrator can only recover the FSImage from SNN 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*



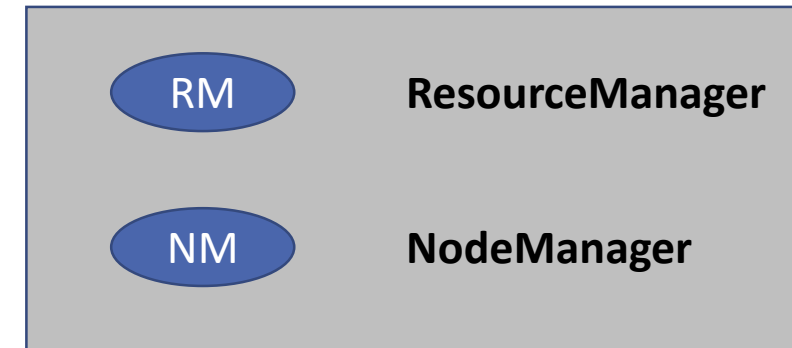
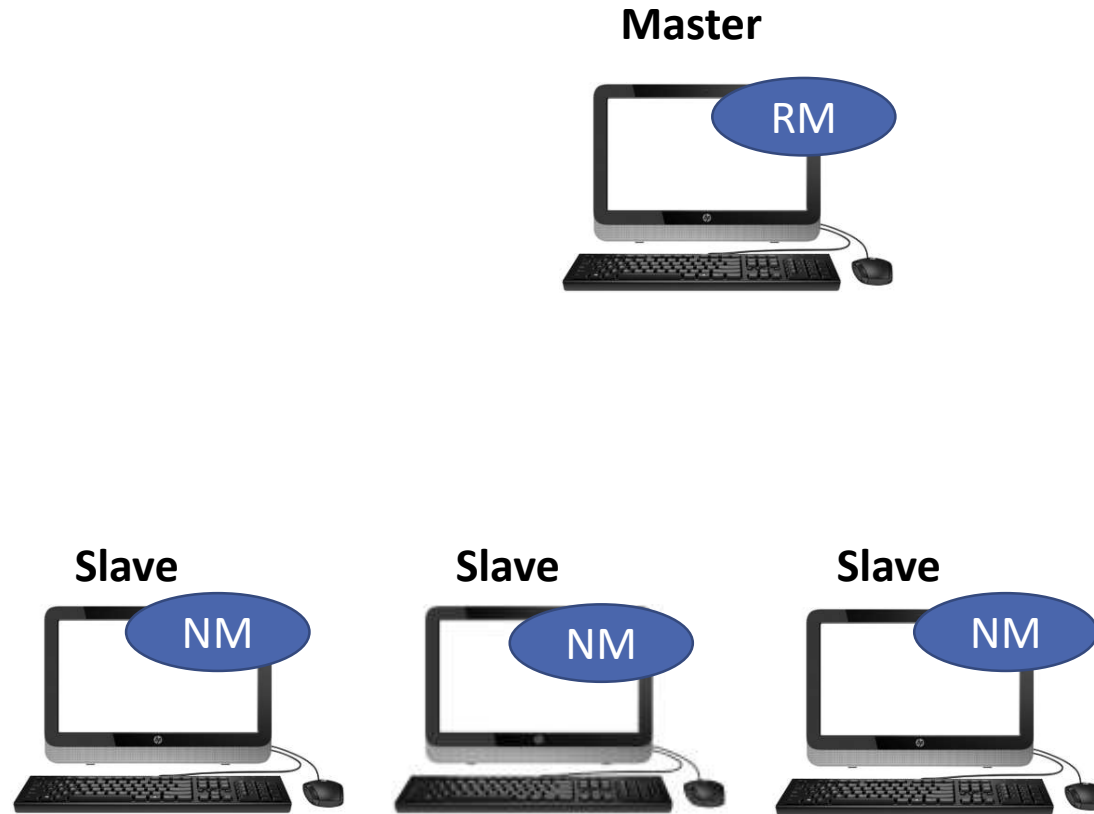
# WebUI Ports

- *WebUI for accessing HDFS*
  - *NameNode WebUI - <http://localhost:50070>*
  - *Secondary NameNode WebUI – <http://localhost:50090>*
  - *DataNode WebUI - <http://localhost:50075>*
- *WebUI for accessing HDFS*
  - *ResourceManager WebUI - <http://localhost:8088>*
  - *NodeManager WebUI - <http://localhost:8042>*
  - *JobHistoryServer WebUI - <http://localhost:19888>*
- *HUE (Cloudera)*
  - *WebUI - <http://localhost:8888>*

*Note: Port numbers are changing in Hadoop 3.x*

# YARN Daemons

Master – Slave Architecture

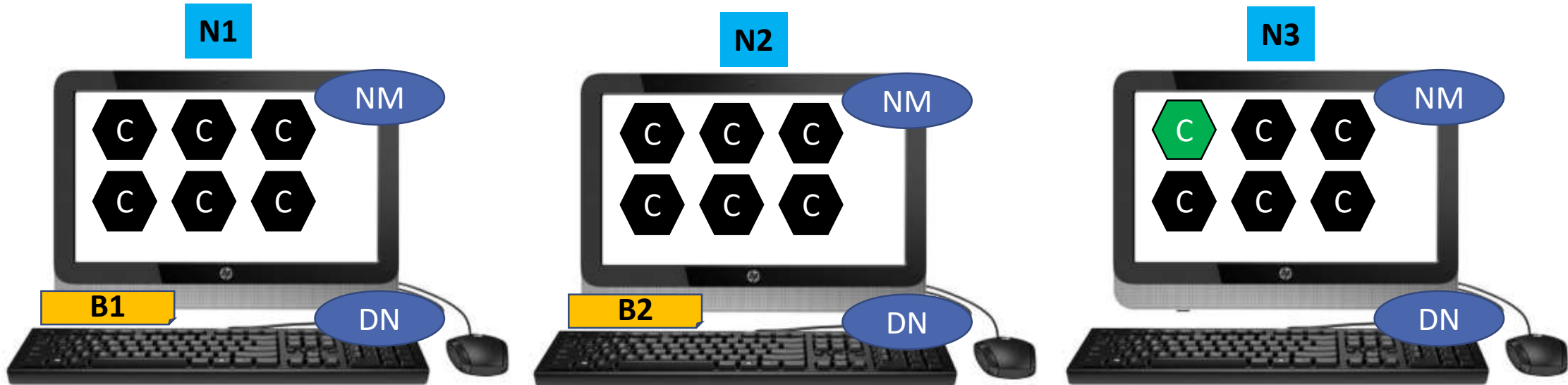


# YARN Anatomy

File1.txt – B1 and B2



\$ yarn jar .....



# YARN Components

Application

Job

- Resource Manager

1 / cluster

Scheduling + allocation of compute resources

Map

Reduce

Task

Task

- Application Master

1 / job

Monitoring

- Node Manager

1 / DN

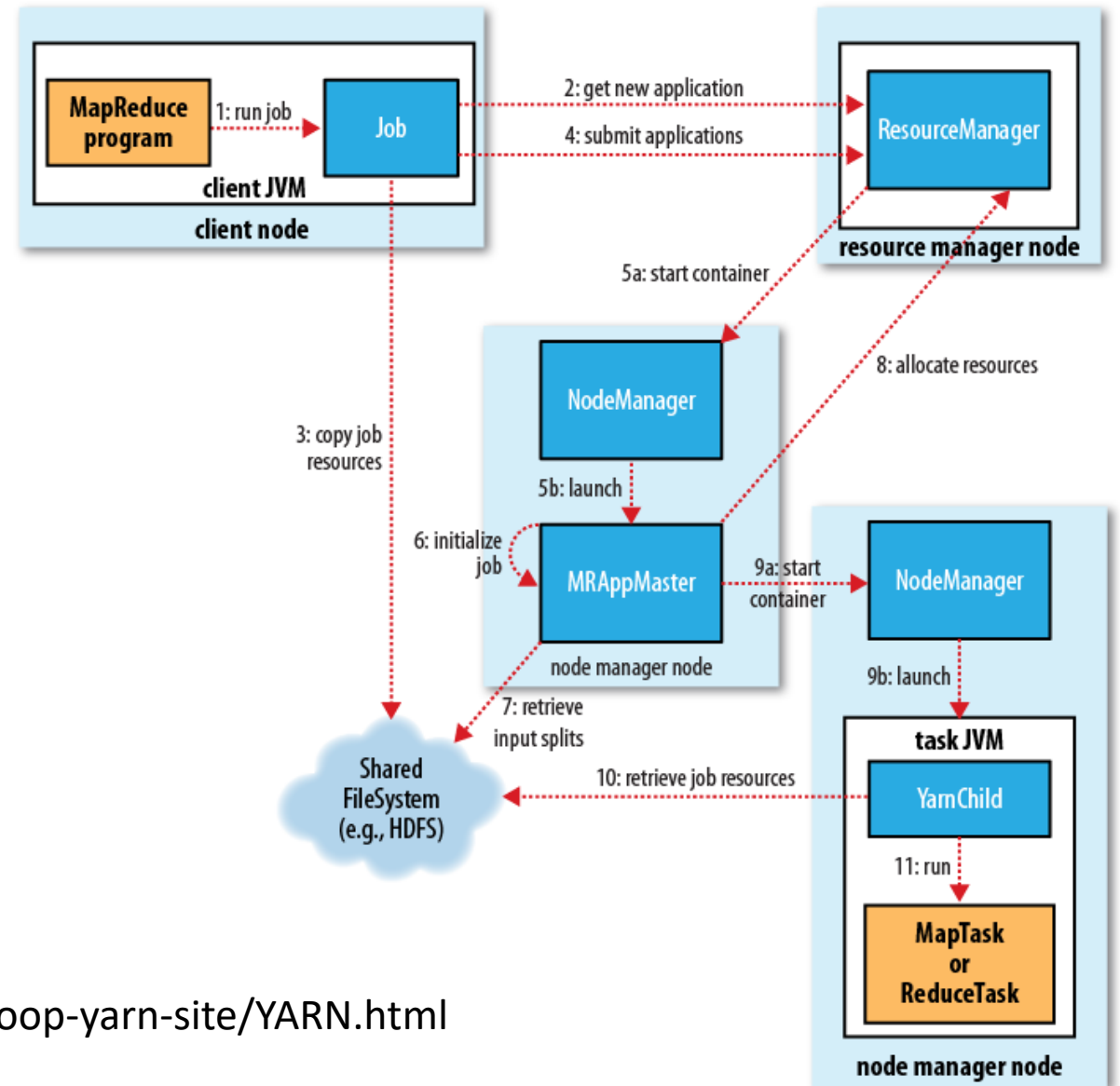
Monitor containers on the node

- YarnChild

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

Execution of tasks

# Anatomy of a MapReduce job run - YARN



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

# WordCount execution steps

- **Terminal 1**

- `$ sudo watch 'jps | grep -v jps | sort -k 2'`

- **Terminal 2**

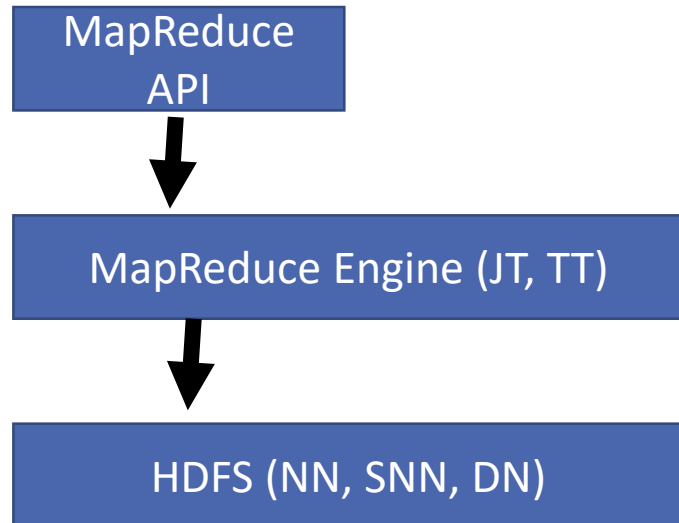
- `$ yarn jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar wordcount  
/Sample/SampleFile.txt /Sample/WC`

- **Once the program completed, check the output**

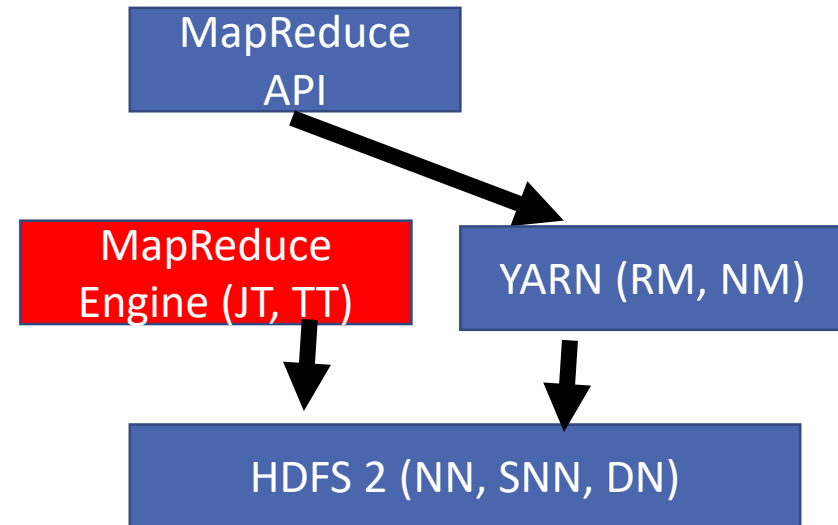
- `$ hadoop fs -cat /Sample/WC/part-r-00000`

# Hadoop 1.x Vs Hadoop 2.x

## Hadoop 1.x



## 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



# Agenda – Day 2

- Hadoop Architecture
- Data Analytics – Hive
- Data Analytics - Impala
- Data Ingest - Sqoop
- Data Ingest - Flume

# Introduction to MapReduce

- *Data processing paradigm for distributed datasets*
- *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*

# MapReduce with example

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

## Steps in MapReduce

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

WordCount.java



## Output

/Sample/WC/part-r-00000

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

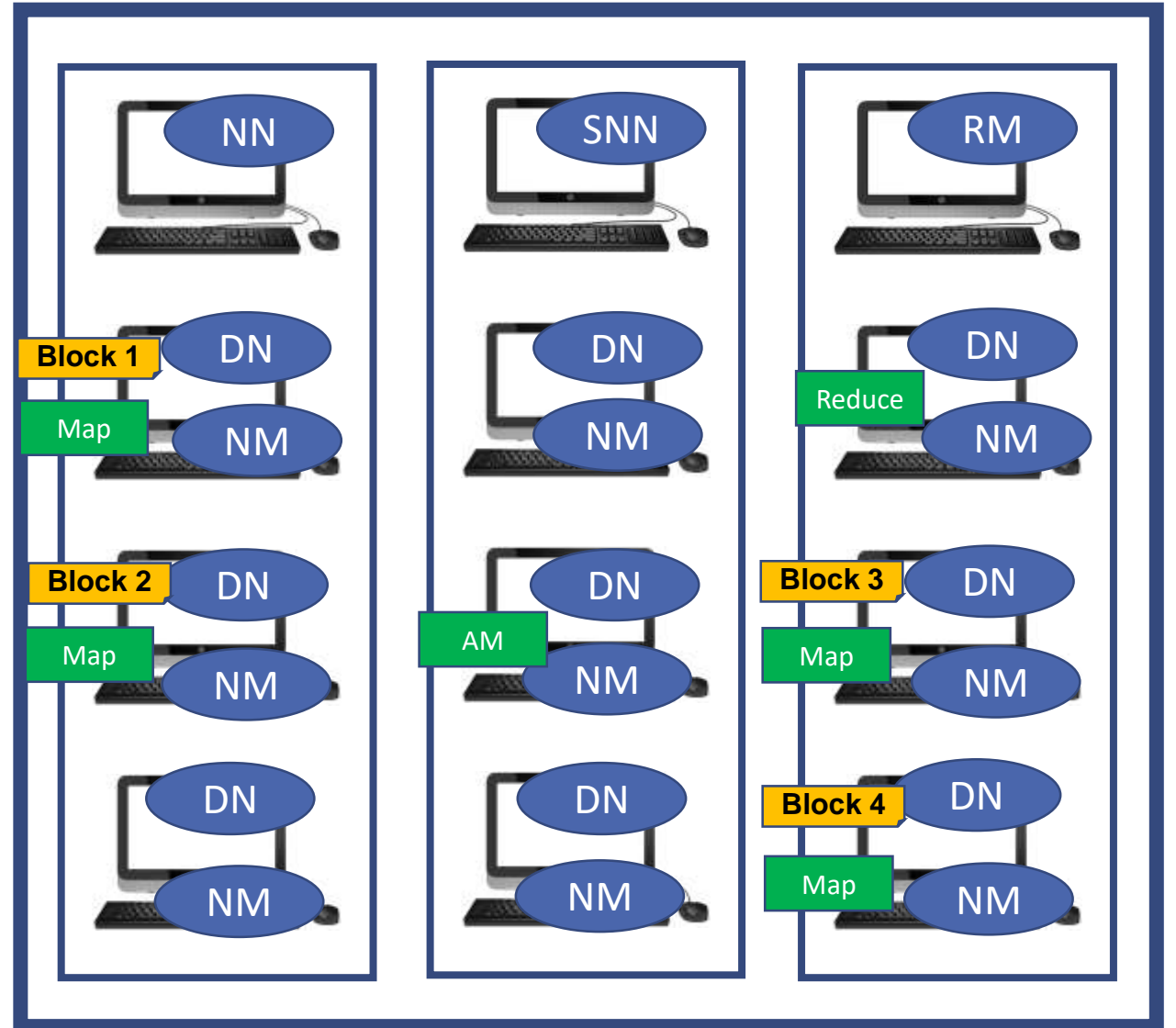
- Map – Transformation -> Convert into words
- Reduce – Aggregation -> Count the words

# MapReduce program execution on a Hadoop Cluster

*Hadoop binaries and configs*



Gateway / Client Node



# How to execute a MapReduce program on a cluster?

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

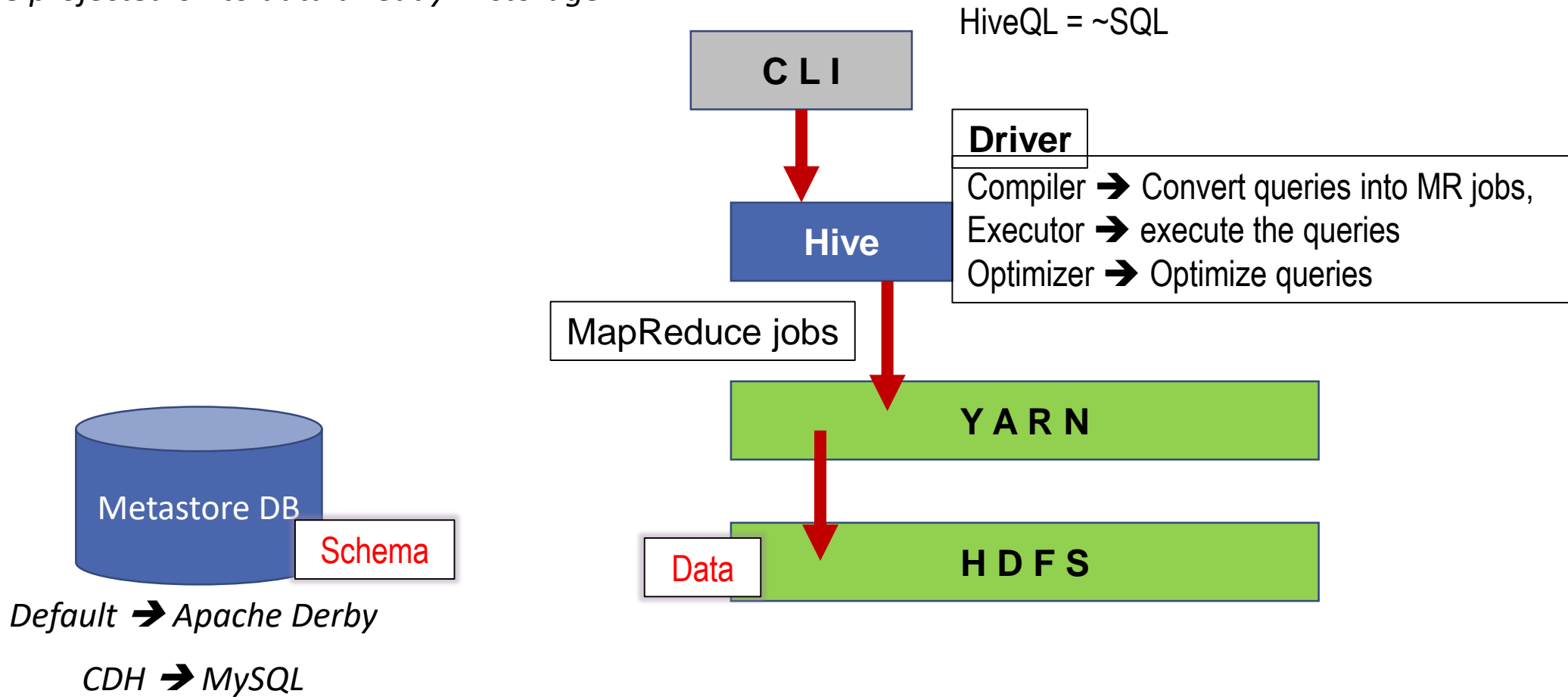
# Apache Hive

<http://hive.apache.org/>

- *Apache Hive is considered the standard for SQL queries over petabytes of data in Hadoop*
- *Open sourced by Facebook in 2008*
- *Data analysts use Hive to query, summarize, explore and analyze that data, then turn it into actionable business insight*
- *Hive easily integrates with other critical technologies using a JDBC interface*
- *On a cluster managed by Cloudera Manager, Hive comes along with the base CDH installation and does not need to be installed separately. With Cloudera Manager, you can enable or disable the Hive service, but the Hive component always remains present on the cluster*

# 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'*
- *External Table*
  - *'external' keyword in DDL*
  - *Hive manages 'schema'*

/user/hive/warehouse → Hive's default warehouse dir



# Hive WordCount

- **Terminal 1**

- `hadoop fs -mkdir /wc`
- `$ hadoop fs -cp /Sample/SampleFile.txt /wc/`

- **Terminal 2**

- `$ hive`
- `hive> create external table table1(col1 string) row format delimited fields terminated by '\n' location '/wc';`
- `hive> select word, count(*) from table1 LATERAL VIEW explode(split(col1, ' ')) ltable as word GROUP BY word;`

- **Once the query execution completed, the output would be printed on the console**

# Hive Joins

- `** emp.txt - Create file as /home/cloudera/Desktop/Labs/emp.txt`
- `swetha,250000,Bengaluru`
- `anamika,200000,Mysuru`
- `tarun,300000,Mangaluru`
- `anita,250000,Mumbai`
  
- `** email.txt - Create file as /home/cloudera/Desktop/Labs/email.txt`
- `swetha,swetha@gmail.com`
- `tarun,tarun@gmail.in`
- `nagesh,nagesh@yahoo.com`
- `venkatesh,venki@gmail.com`

# Hive Joins

- `hive> create table employee(name string, salary float,city string) row format delimited fields terminated by ',';`
- `hive> load data local inpath '/home/cloudera/Desktop/Labs/emp.txt' into table employee;`
- `hive> select * from employee where name='tarun';`
- `hive> create table mailid (name string, email string) row format delimited fields terminated by ',';`
- `hive> load data local inpath '/home/cloudera/Desktop/Labs/email.txt' into table mailid;`
- `hive> SET hive.auto.convert.join=false;`
- `hive> select a.name,a.city,a.salary,b.email from employee a inner join mailid b on a.name = b.name;`
- `hive> select a.name,a.city,a.salary,b.email from employee a left outer join mailid b on a.name = b.name;`
- `hive> select a.name,a.city,a.salary,b.email from employee a right outer join mailid b on a.name = b.name;`
- `hive> select a.name,a.city,a.salary,b.email from employee a full outer join mailid b on a.name = b.name;`

# Apache Impala

<http://impala.apache.org/>

- *Low latency SQL engine*
- *Developed by Cloudera*
- *Now an open source Apache project*
- *Impala provides high-performance, low-latency SQL queries on data stored in popular Apache Hadoop file formats*
- *The fast response for queries enables interactive exploration and fine-tuning of analytic queries, rather than long batch jobs traditionally associated with SQL-on-Hadoop technologies*
- *Impala integrates with the Apache Hive metastore database, to share databases and tables between both components*
- *The high level of integration with Hive, and compatibility with the HiveQL syntax, lets you use either Impala or Hive to create tables, issue queries, load data, and so on*

# Advantages of Impala

- *Impala integrates with the existing Hadoop ecosystem, meaning data can be stored, shared, and accessed using the various solutions included with Hadoop. This also avoids data silos and minimizes expensive data movement*
- *Impala provides access to data stored in Hadoop without requiring the Java skills required for MapReduce jobs. Impala can access data directly from the HDFS file system*
- *Impala also provides a SQL front-end to access data in the HBase database system, or in the Amazon Simple Storage System (S3)*

# Impala Architecture

- *Impala queries run on an additional set of daemons that run on the Hadoop cluster*
  - *Impala Daemon – typically runs on each worker node in the cluster*
    - *Co-located with the HDFS DataNode*
    - *Accepts queries from the impala-shell, HUE, JDBC, or ODBC*
    - *Checks the local metadata cache*
    - *Distributes the work to other Impala Daemons in the cluster*
  - *Two other daemons running on master nodes support query execution*
  - *Impala State Store (one per cluster)*
    - *Provides lookup service and status checks for Impala daemons*
  - *Impala Catalog Server (one per cluster)*
    - *Relays metadata changes to all the Impala Daemons in a cluster*

# Impala Vs Hive

- *Like Hive, Impala allows users to query data in HDFS using an SQL-like language*
- *Unlike Hive, Impala does not turn queries into MapReduce jobs*
- *Impala is pioneering the use of the Parquet file format, a columnar storage layout that is optimized for large-scale queries typical in data warehouse scenarios*
- *Impala returns results typically within seconds or a few minutes, rather than the many minutes or hours that are often required for Hive queries to complete*

# Running hive queries using impala

- `[quickstart.cloudera:21000] > invalidate metadata;`
- `[quickstart.cloudera:21000] > show tables;`
- `[quickstart.cloudera:21000] > select * from employee where name='tarun';`
- `[quickstart.cloudera:21000] > select a.name,a.city,a.salary,b.email from employee a inner join mailid b on a.name = b.name;`
- `[quickstart.cloudera:21000] > select a.name,a.city,a.salary,b.email from employee a left outer join mailid b on a.name = b.name;`
- `[quickstart.cloudera:21000] > select a.name,a.city,a.salary,b.email from employee a right outer join mailid b on a.name = b.name;`
- `[quickstart.cloudera:21000] > select a.name,a.city,a.salary,b.email from employee a full outer join mailid b on a.name = b.name;`



# Impala error – restarting impala services

- `quickstart.cloudera:21000] > show tables;`
- `Query: show tables`
- `ERROR: AnalysisException: This Impala daemon is not ready to accept user requests.  
Status: Waiting for catalog update from the StateStore.`
- Terminal 2:
  - `$ sudo service impala-state-store stop`
  - `$ sudo service impala-catalog stop`
  - `$ sudo service impala-server stop`
  - `$ sudo service impala-state-store start`
  - `$ sudo service impala-catalog start`
  - `$ sudo service impala-server start`

*Note: Use these commands only if Impala doesn't work*

# Apache Sqoop

<http://sqoop.apache.org/>

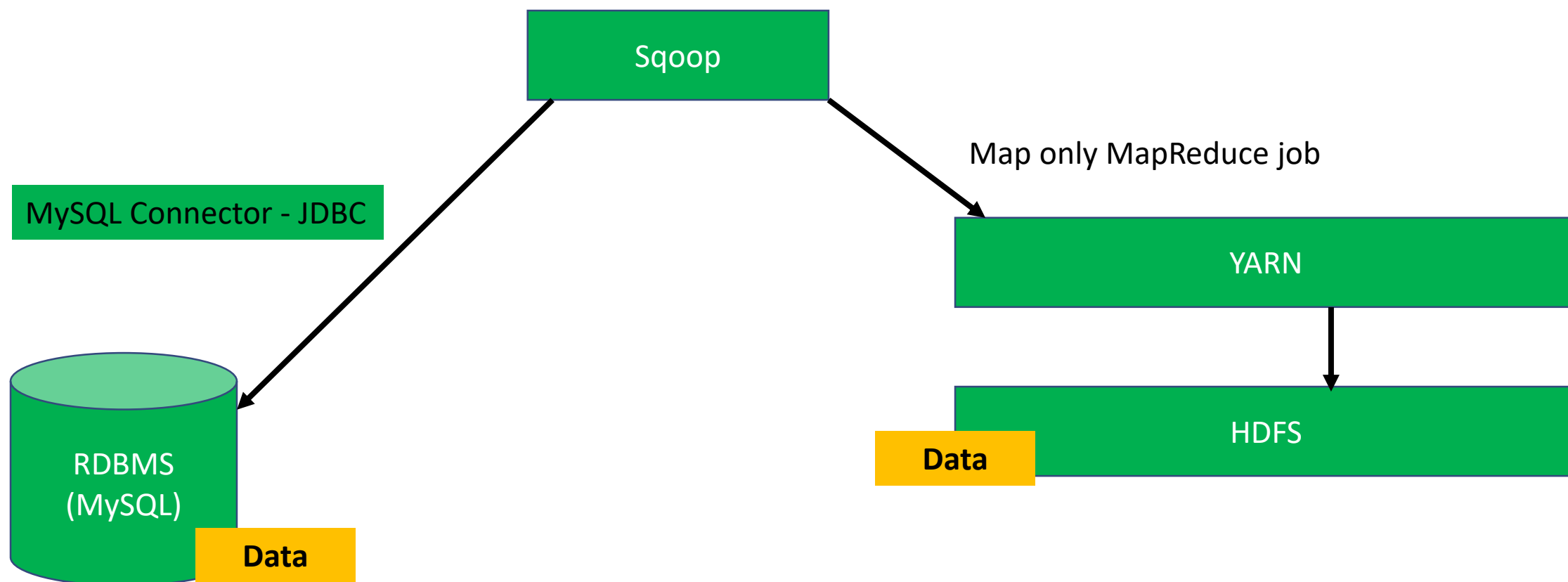
- *Sqoop is a tool to transfer data from RDBMS to Hadoop and vice versa*
- *Sqoop is “the SQL-to-Hadoop database import tool”*
- *Open-source Apache project*
- *Originally developed at Cloudera*
- *Designed to import data from RDBMSs into HDFS*
- *Can also send data from HDFS to an RDBMS*
- *Uses JDBC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

# 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

<http://sqoop.apache.org/>



# Apache Sqoop

- ▣ *Sqoop Import* → *From RDBMS to HDFS*
  - ▣ *Sqoop Export* → *From HDFS to RDBMS*
- 
- ▣ *Sequential Import / Export* → *m 1*
  - ▣ *Parallel Import / Export* → *multiple maps*

## Sqoop Installation and Configuration

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

# Sqoop commands

- **Terminal 1 - MySQL**

- `$ mysql -u root -p`
- `mysql> show databases;`
- `mysql> show databases;`
- `mysql> use retail_db;`
- `mysql> describe categories;`
- `mysql> select * from categories;`
- `mysql> select * from departments;`
- `mysql> select category_name from categories where category_department_id = 2;`

# Sqoop commands

- **Terminal 2 - Sqoop**

- `$ sqoop version`
- `$ sqoop list-databases --connect jdbc:mysql://localhost:3306 --username root -P`
- `$ sqoop list-tables --connect jdbc:mysql://localhost:3306/retail_db --username root -P`
- `$ sqoop import --connect jdbc:mysql://localhost:3306/retail_db --username root -P --table categories --target-dir /categories_import -m 1`
- `$ sqoop import --connect jdbc:mysql://localhost:3306/retail_db --username root -P --table categories --target-dir /categories_import_p -m 2`

# Sqoop commands

- **Terminal 2 - Sqoop**

- `$ sqoop version`
- `$ sqoop list-databases --connect jdbc:mysql://localhost:3306 --username root -P`
- `$ sqoop list-tables --connect jdbc:mysql://localhost:3306/retail_db --username root -P`
- `$ sqoop import --connect jdbc:mysql://localhost:3306/retail_db --username root -P --table categories --target-dir /categories_import -m 1`
- `$ sqoop import --connect jdbc:mysql://localhost:3306/retail_db --username root -P --table categories --target-dir /categories_import_p -m 2`



# Sqoop Export

- **Terminal 1 – MySQL**

- `mysql> create database sample;`
- `mysql> use sample;`
- `mysql> CREATE TABLE categories_new(category_id int(11) PRIMARY KEY, category_dept_id int(11), category_name varchar(50));`
- `mysql> select * from categories_new;`

- **On Terminal 2**

- `$ sqoop export --connect jdbc:mysql://localhost:3306/sample --username root --P --table categories_new --export-dir /categories_import_p`

- **On Terminal 1 (MySQL)**

- `mysql> select * from categories_new;`

# Sqoop, Hive and Impala

- **Terminal 3 - Hive**

- `hive> create table categories(cat_id int, dept_id int, cat_name string) row format delimited fields terminated by ',' location '/categories_import' ;`
- `hive> select * from categories;`
- `hive> select count(*) from categories;`

- **Terminal 4 - Impala**

- `[quickstart.cloudera:21000] > invalidate metadata;`
- `[quickstart.cloudera:21000] > select * from categories;`
- `[quickstart.cloudera:21000] > select count(*) from categories;`

# Sqoop Export

- **Terminal 1 - MySQL**

- `$ mysql -u root -p`
- `mysql> create database sample;`
- `mysql> show databases;`
- `mysql> use sample;`
- `mysql> CREATE TABLE categories_new(category_id int(11) PRIMARY KEY, category_dept_id int(11), category_name varchar(50));`
- `mysql> select * from categories_new;`

- **Terminal 2 - Sqoop**

- `sqoop export --connect jdbc:mysql://localhost:3306/sample --username root --P --table categories_new --export-dir /categories_import_p`

- **Terminal 1 - MySQL**

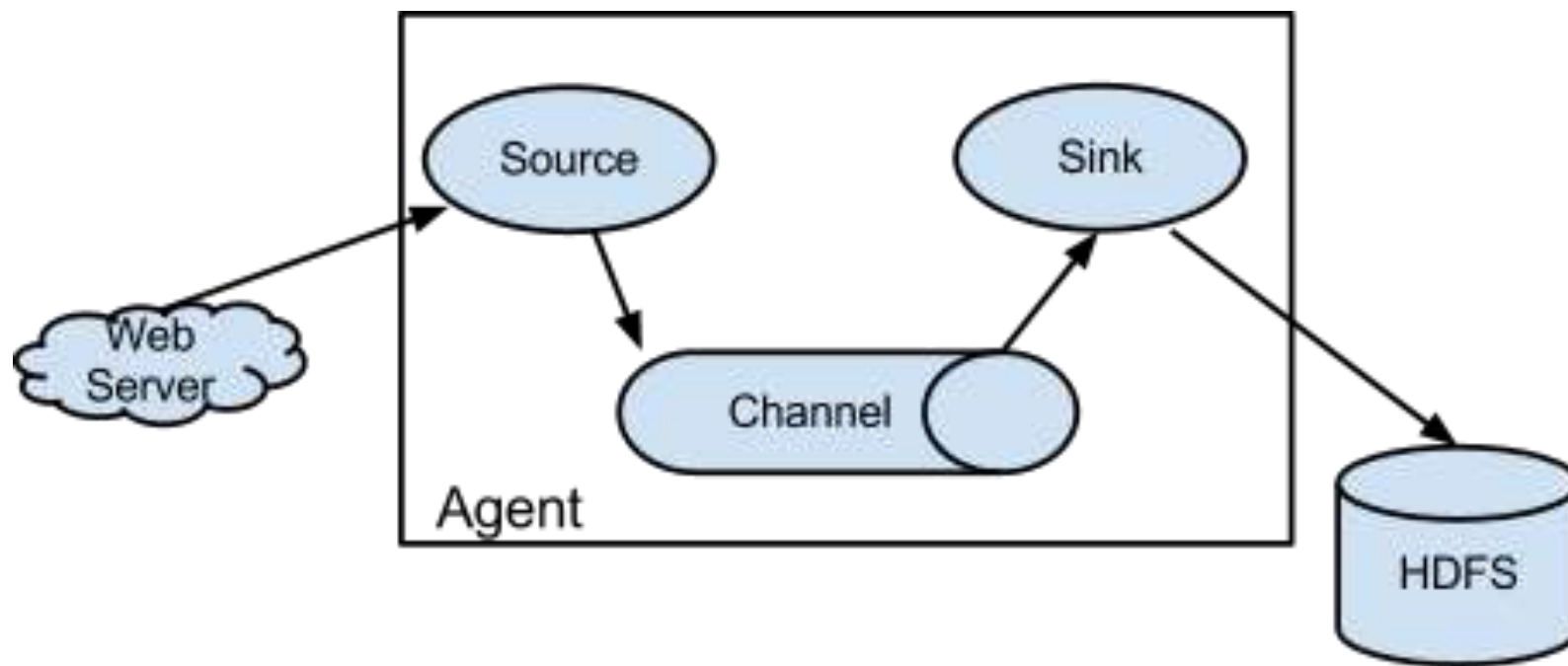
- `mysql> select * from categories_new;`

# 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, architecture is based on streaming data flows*
- *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*

# Flume example

- **Data simulator shell script - Save as /home/cloudera/Desktop/Labs/data\_gen.sh**

```
#!/bin/bash
```

```
while true
```

```
do
```

```
    echo "Welcome to Hadoop" >> /tmp/access_log
```

```
    sleep 1
```

```
done
```

# Flume example

- **Configuration file**

```
tail1.sources = src1  
tail1.channels = ch1  
tail1.sinks = sink1
```

```
tail1.sources.src1.type = exec  
tail1.sources.src1.command = tail -F /tmp/access_log  
tail1.sources.src1.channels = ch1
```

```
tail1.channels.ch1.type = memory
```

```
tail1.sinks.sink1.channel = ch1  
tail1.sinks.sink1.type = hdfs  
tail1.sinks.sink1.hdfs.path = hdfs://localhost:8020/access_log  
tail1.sinks.sink1.hdfs.fileType = DataStream  
tail1.sinks.sink1.hdfs.writeFormat = Text
```



# Flume example

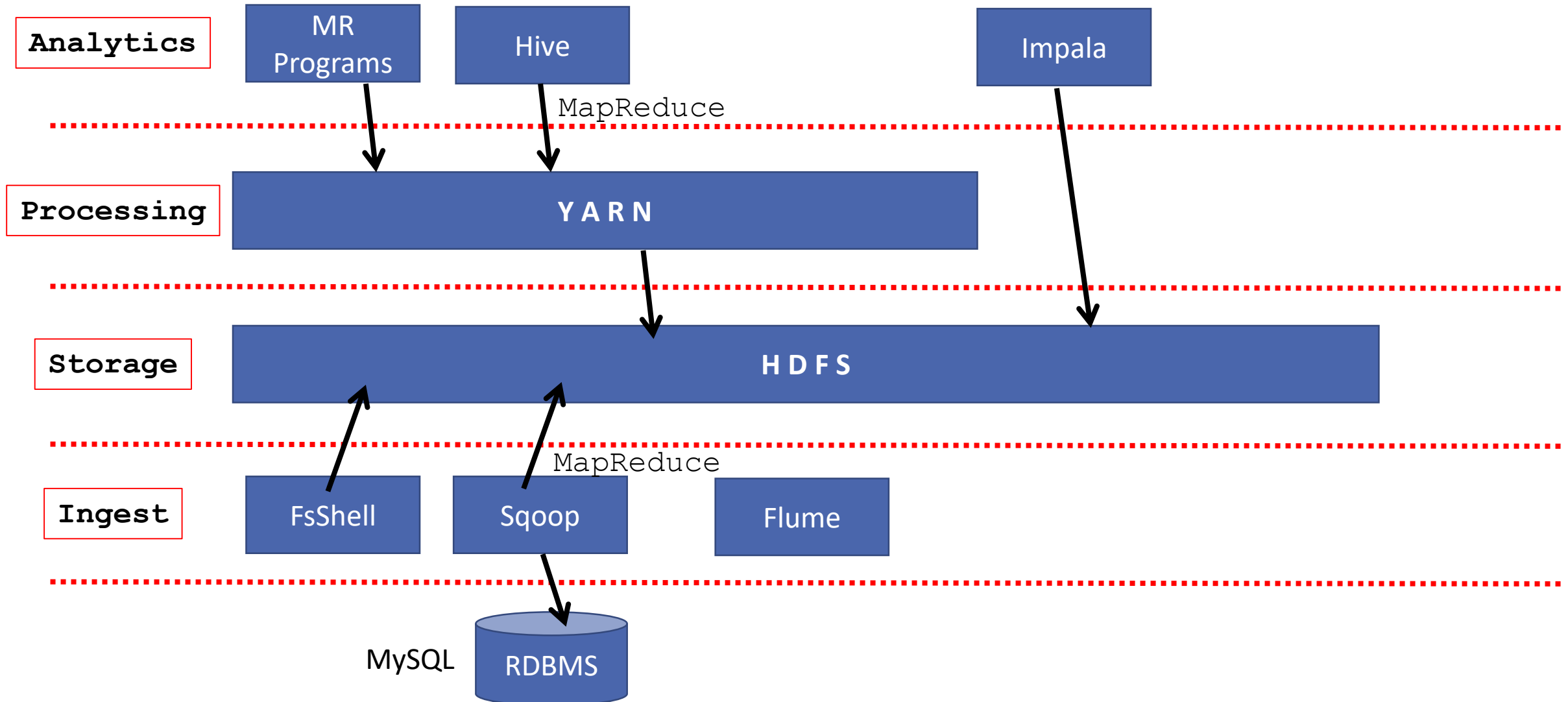
- **Terminal 1 - Run data\_gen**

- `$ chmod 777 /home/cloudera/Desktop/Labs/data_gen.sh`
- `$ ./home/cloudera/Desktop/Labs/data_gen.sh`

- **Terminal 2 - Run Flume agent**

- `flume-ng agent -n tail1 -c conf -f /home/cloudera/Desktop/Labs/tail.conf`

# What have we learnt?



# Further Reading

- <https://www.tatacommunications.com/press-release/cloudera-tata-communications-launch-big-data-platform-tackle-data-deluge/>
- <http://blog.cloudera.com/blog/2013/08/how-to-select-the-right-hardware-for-your-new-hadoop-cluster/>
- <http://hadoop.apache.org/docs/r2.7.5/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html>
- <http://hadoop.apache.org/docs/r2.7.5/hadoop-yarn/hadoop-yarn-site/YARN.html>