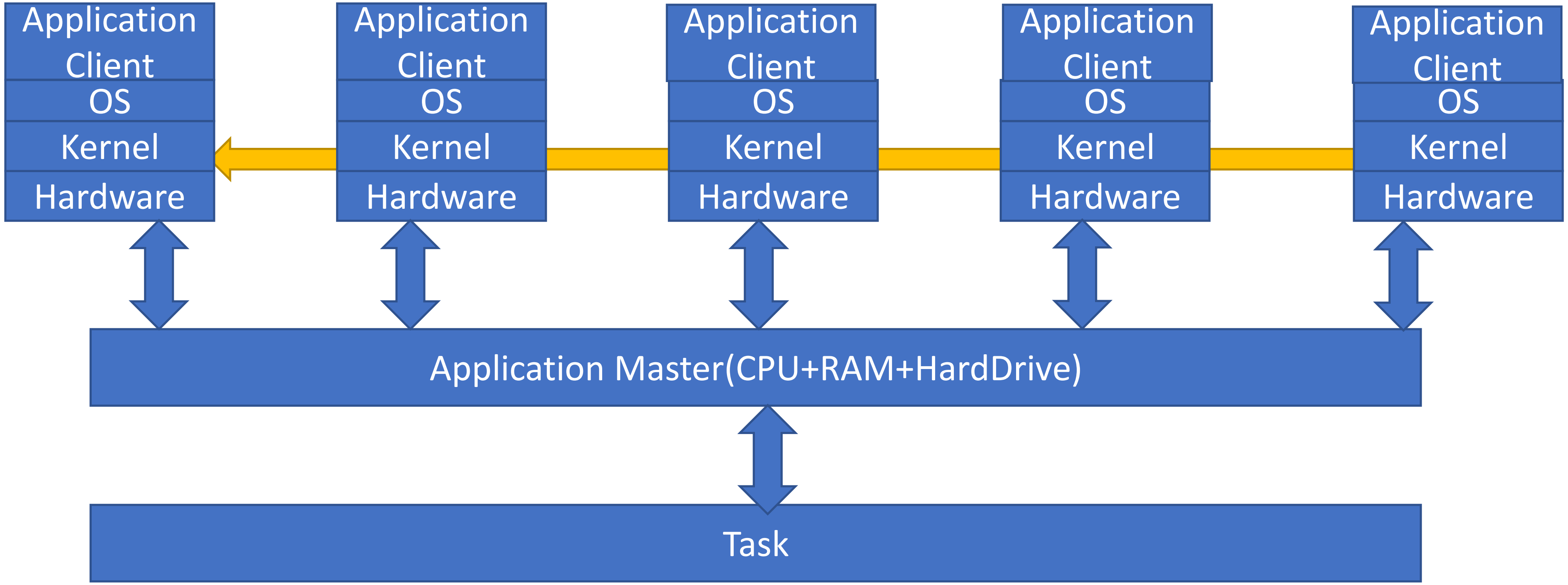
Terminologies

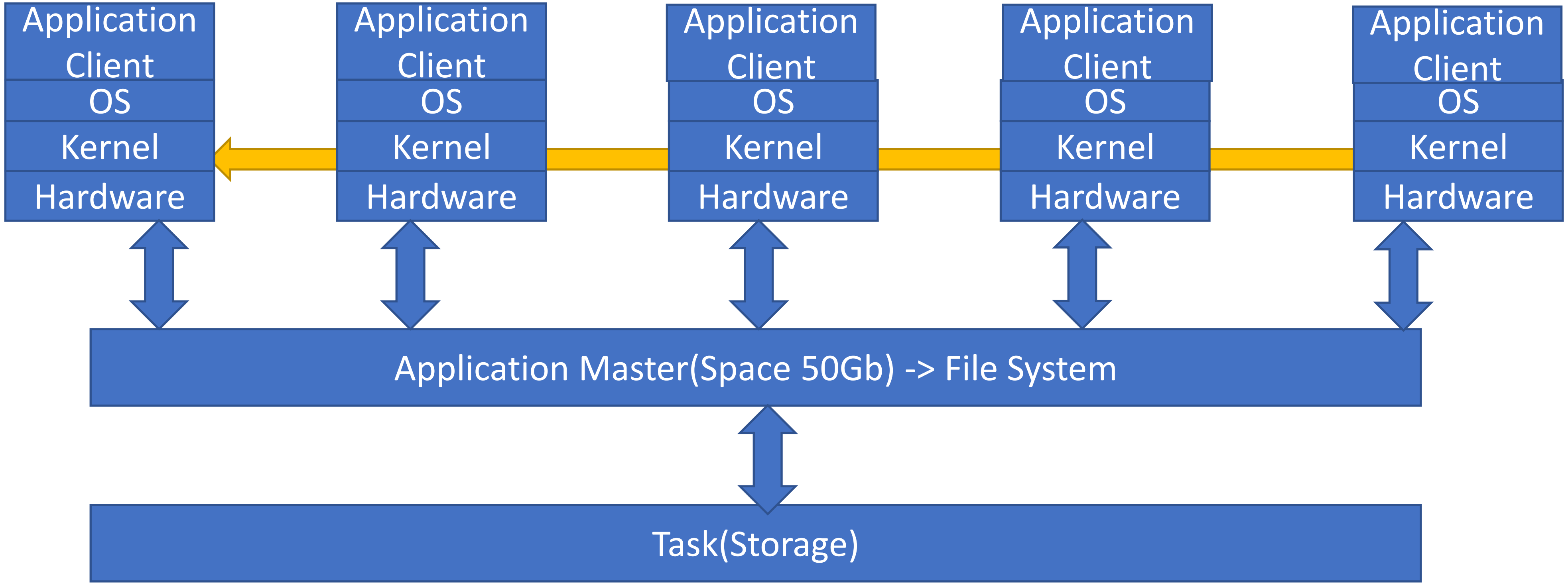
- Cluster computing -> is the kind of architecture that group of computers use their resources to perform the particular task by means of interconnected with them via network
- Distributed Computing
- Distributed Storage
- Auto Scaling
- Difference between Horizontal and Vertical Scaling

Cluster



Application Client
OS
Kernel
Hardware
Application Client
OS
Kernel
Hardware
Application Client
OS
Kernel
Hardware
Application Client
OS
Kernel
Hardware
Application Client
OS
Kernel
Hardware
Application Master(CPU+RAM+HardDrive)
Task

Distributed Storage



Application Client

OS

Kernel

Hardware

Application Client

OS

Kernel

Hardware

Application Client

OS

Kernel

Hardware

Application Client

OS

Kernel

Hardware

Application Client

OS

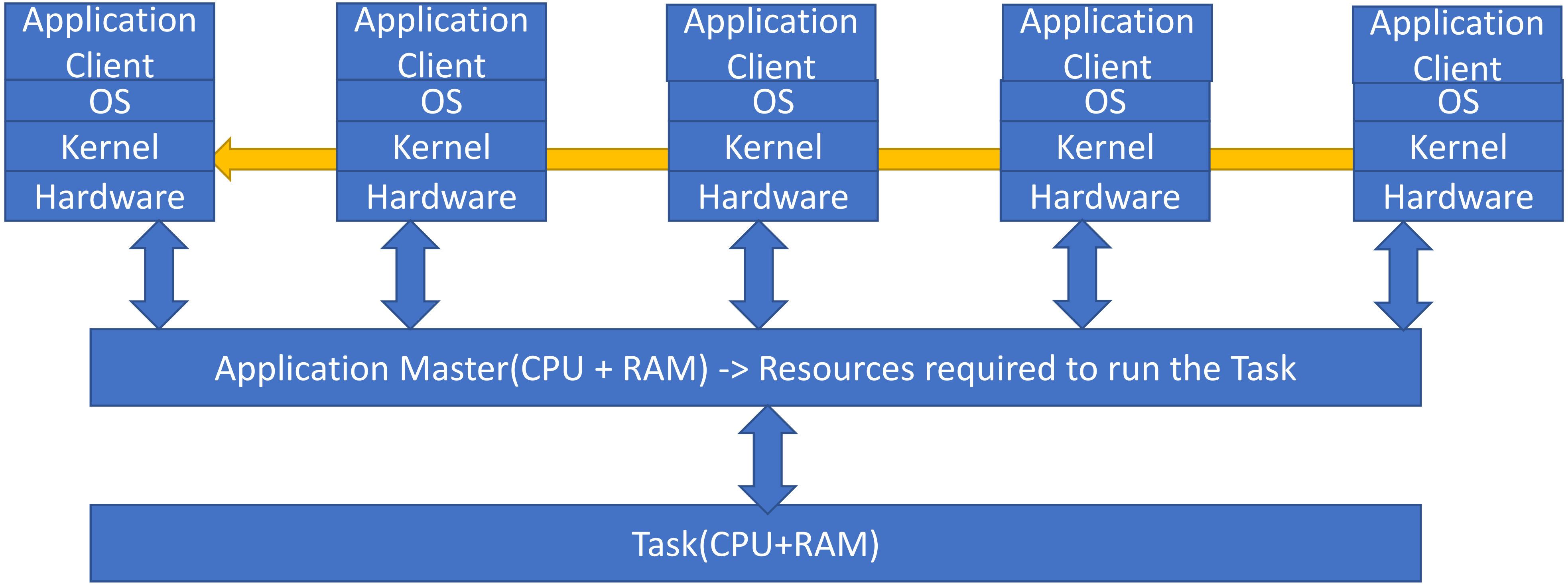
Kernel

Hardware

Application Master(Space 50Gb) -> File System

Task(Storage)

Distributed Computing



Application Client | OS | Kernel | Hardware

Application Client | OS | Kernel | Hardware

Application Client | OS | Kernel | Hardware

Application Client | OS | Kernel | Hardware

Application Client | OS | Kernel | Hardware

Application Master(CPU + RAM) -> Resources required to run the Task

Task(CPU+RAM)

Auto Scaling

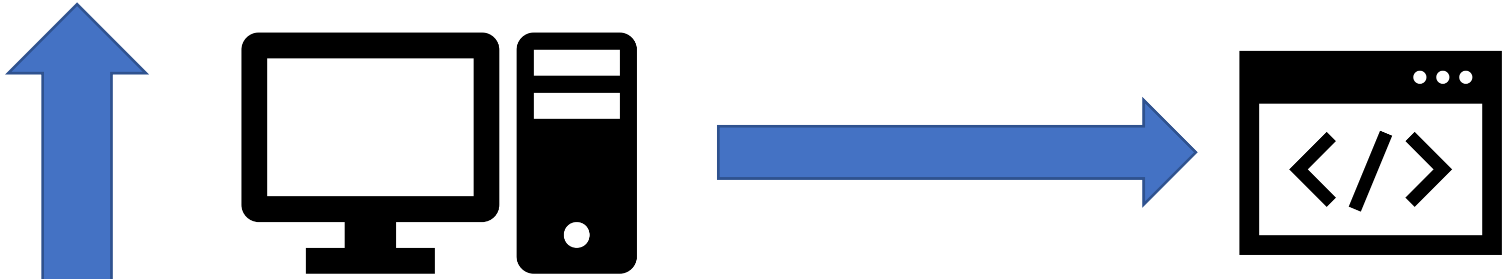
MYSQL Data Base

12th GEN i9(8 Core)
8GB Ram
1TB SSD

Vertical Scaling

While Scaling UP/Down ->
Shutdown -> Remove the Hardware

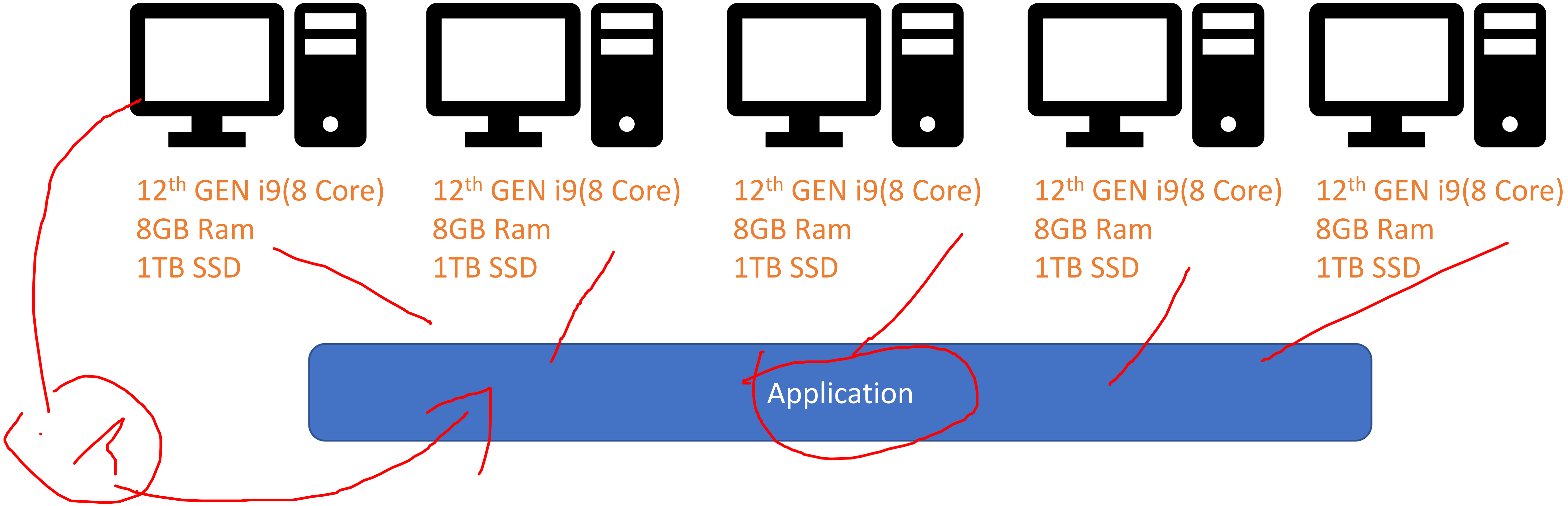
Application Downtime



12th GEN i9(8 Core)
8GB Ram
1TB SSD

MYSQL Data Base

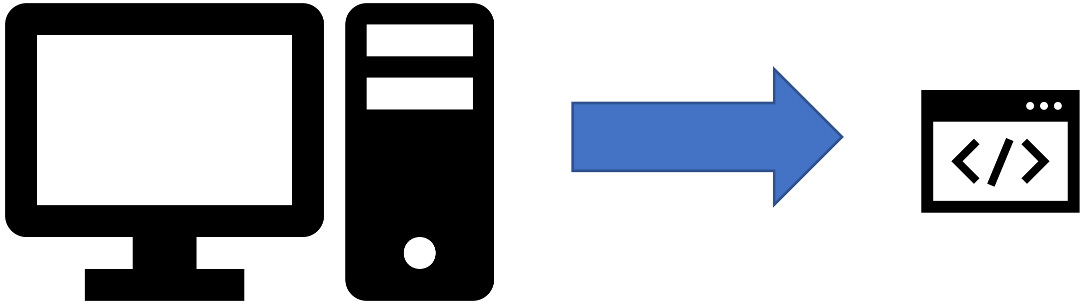
Horizontal Scaling

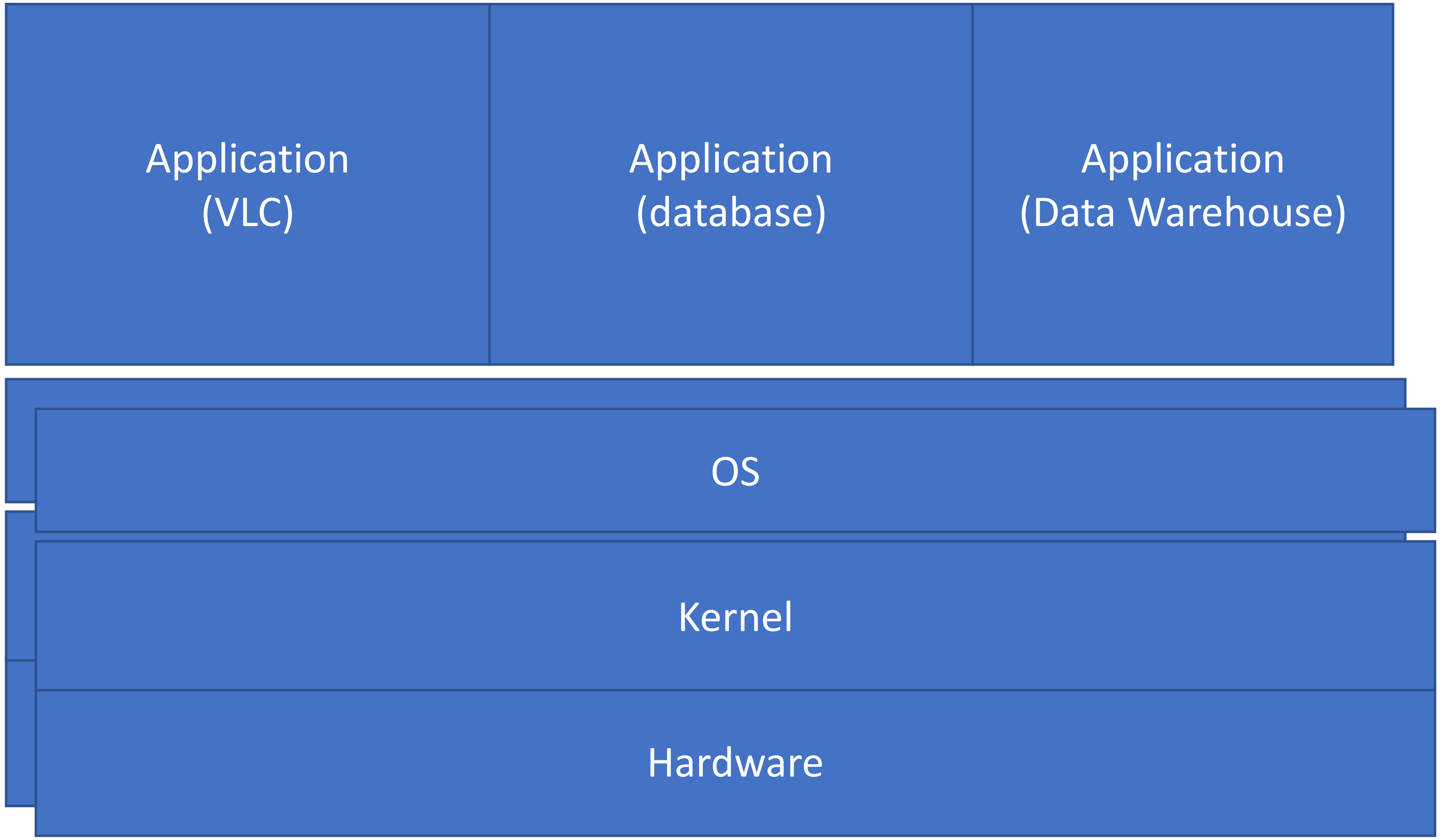


Bigdata

- Big data refers to data sets that are too large or complex to be dealt with by traditional data-processing application software(database).
- Volume -> doesn’t have any kind of limitation in handling the data volume(Distributed Storage)
- Velocity -> time taken to process the amount of data(Computational limitation) -> Distributed Computing
- Variety -> We don’t need to deal the files with ETL Concept -> As the Flat File -> it got variety of tools to deal with the data as same format without doing any kind of type conversion.

- 3 v
- Volume -> Storage
- Velocity -> Computation
- Variety ->





Application
(VLC)

Application
(database)

Application
(Data Warehouse)

OS

Kernel

Hardware

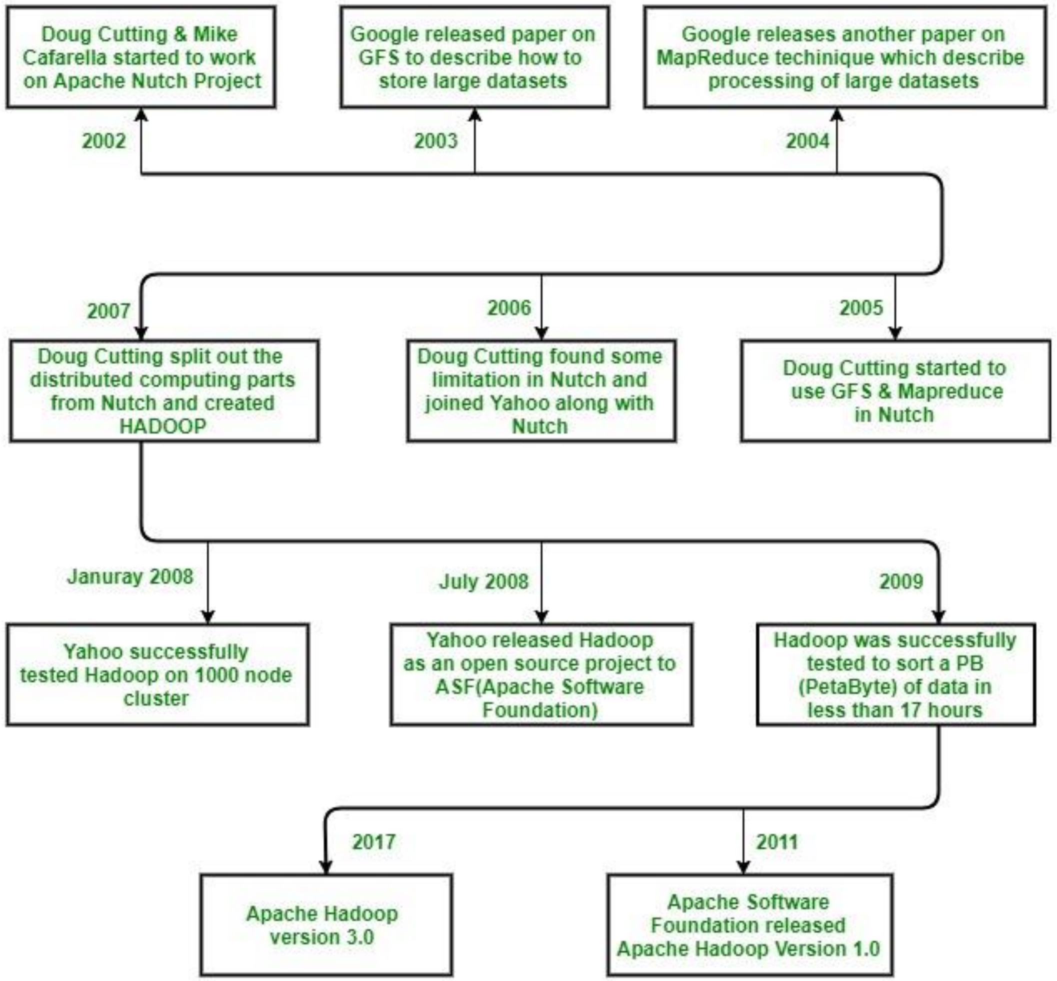
Bigdata -> Concept

- Hadoop
- Spark

Hadoop Frame Work

- Java Based Cluster application ->to deal with big data
- Is an open source tool

History of Hadoop



2002

Doug Cutting & Mike Cafarella started to work on Apache Nutch Project

2003

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

2004

Google releases another paper on MapReduce techinique which describe processing of large datasets

2005

Doug Cutting started to use GFS & Mapreduce in Nutch

2006

Doug Cutting found some limitation in Nutch and joined Yahoo along with Nutch

2007

Doug Cutting split out the distributed computing parts from Nutch and created HADOOP

Januray 2008

Yahoo successfully tested Hadoop on 1000 node cluster

July 2008

Yahoo released Hadoop as an open source project to ASF(Apache Software Foundation)

2009

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

2011

Apache Software Foundation released Apache Hadoop Version 1.0

2017

Apache Hadoop version 3.0

Hadoop Framework

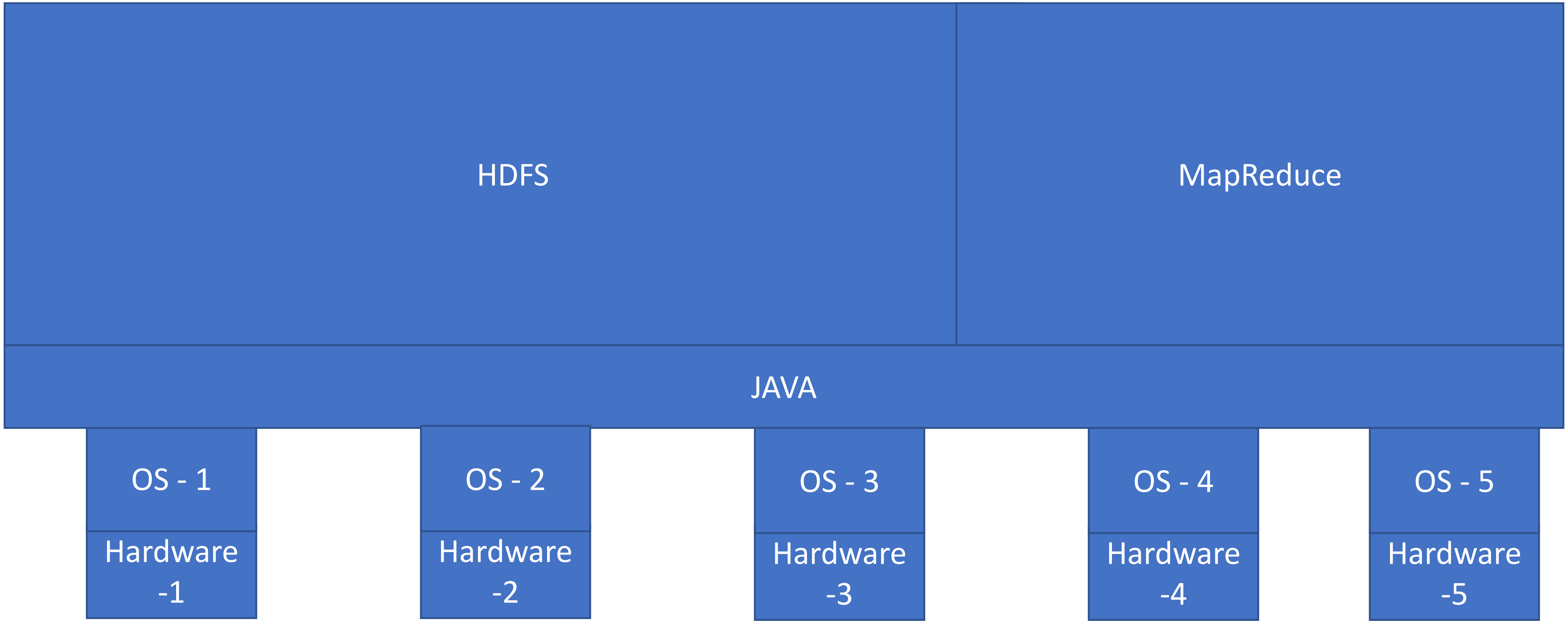
- Distributed Storage(HDFS -> Hadoop Distributed File system)
- Distributed Computing(MapReduce)
- Architecture -> JAVA
- Why JAVA?
- Code -> Compile -> .java -> .class -> JVM(Java Virtual Machine)
- Process -> threads

Hadoop Version 1

- TASK -> Commands -> JAVA Only
- TASK(Data Analytical) -> input(data -> HDFS) -> Command(JAVA -> MAPREDUCE) -> Output(data -> HDFS)
- Storage(HDFS)
  - Master -> Name Node
  - Slave -> Data Node
- CPU+RAM(MapReduce)
  - Master -> Job Tracker
  - Slave -> Task Tacker

Hadoop Version 2

- MapReduce issues
- Communication -> they did the major changes on MapReduce -> YARN(Yet Another Resource Negotiator) -> it will be supported for multiple API and Programming languages
- Way it will handle the job and resource allocation -> Job
- Storage(HDFS)
  - Master -> Name Node
  - Slave -> Data Node
- CPU+RAM(MapReduce)
  - Master -> Resource Manager
  - Slave -> Node Manager



HDFS

MapReduce

JAVA

OS - 1

Hardware -1

OS - 2

Hardware -2

OS - 3

Hardware -3

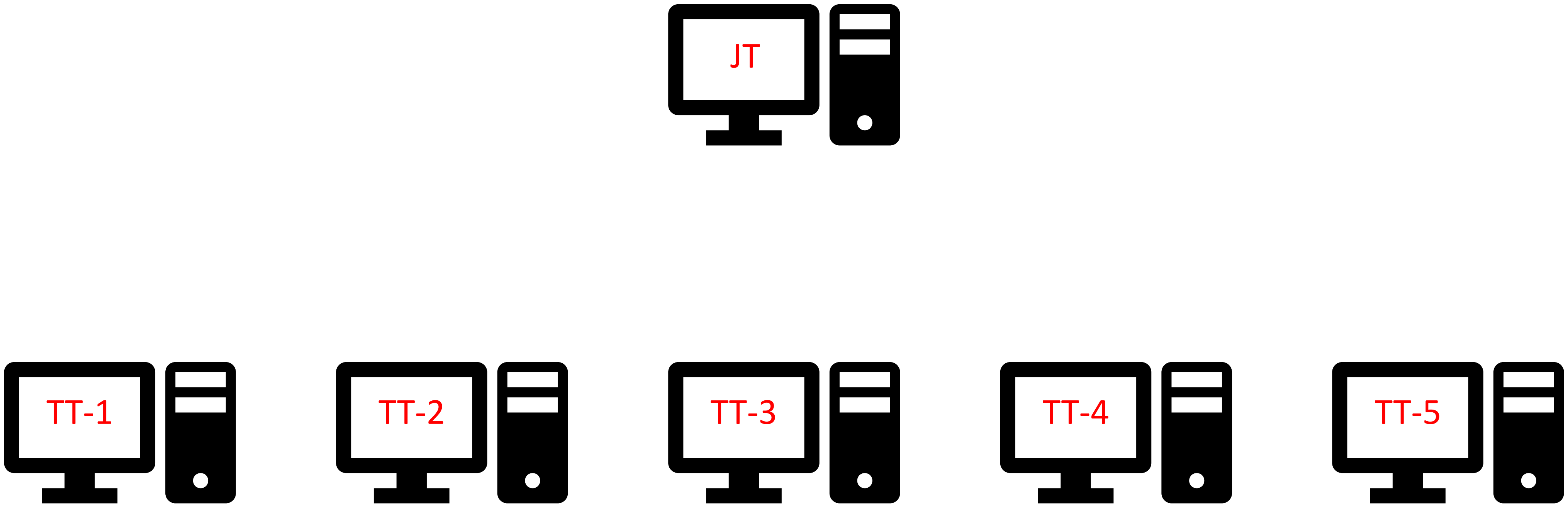
OS - 4

Hardware -4

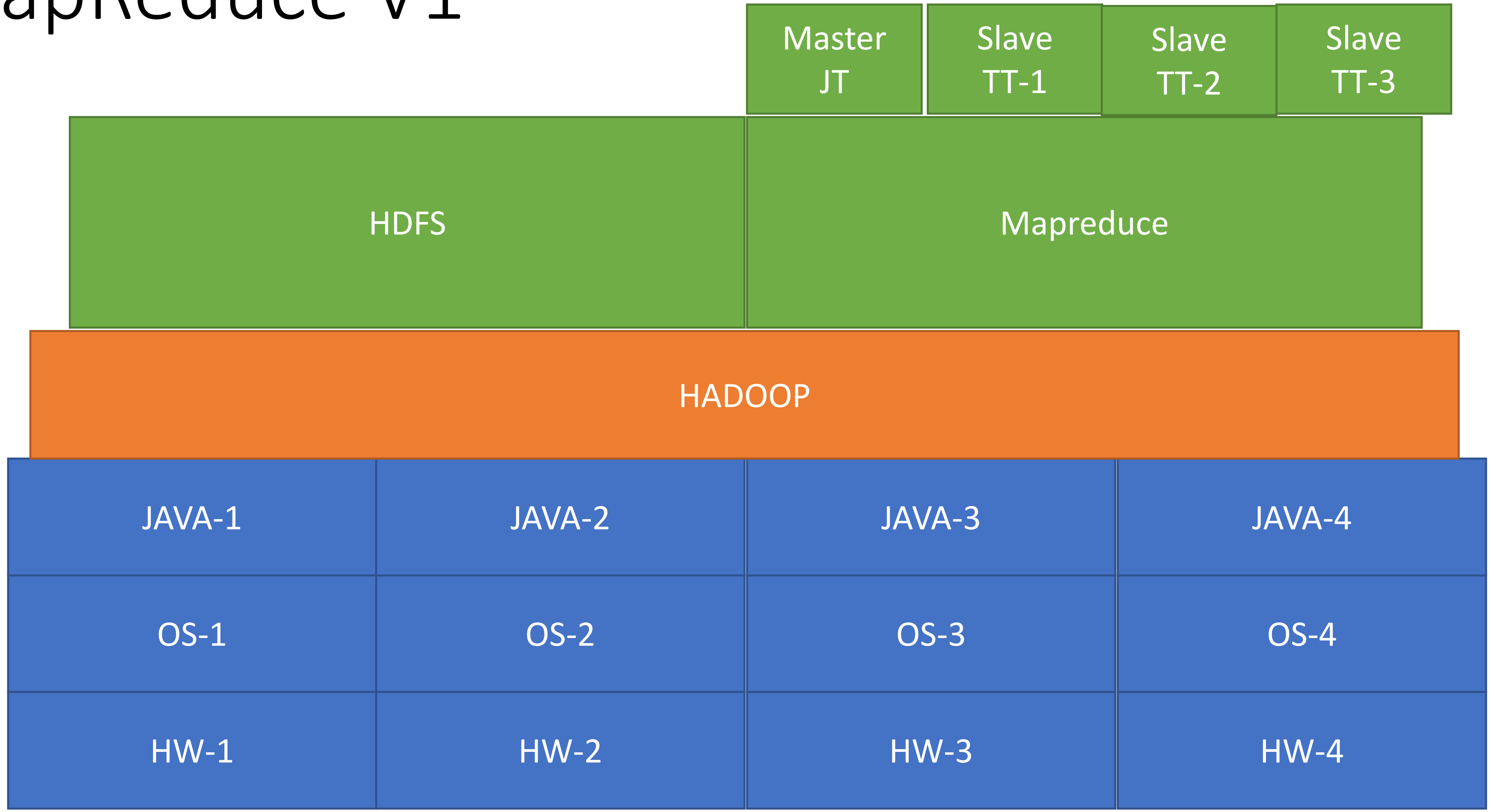
OS - 5

Hardware -5

MapReduce V1



MapReduce V1



Master
JT

Slave
TT-1

Slave
TT-2

Slave
TT-3

HDFS

Mapreduce

HADOOP

JAVA-1

JAVA-2

JAVA-3

JAVA-4

OS-1

OS-2

OS-3

OS-4

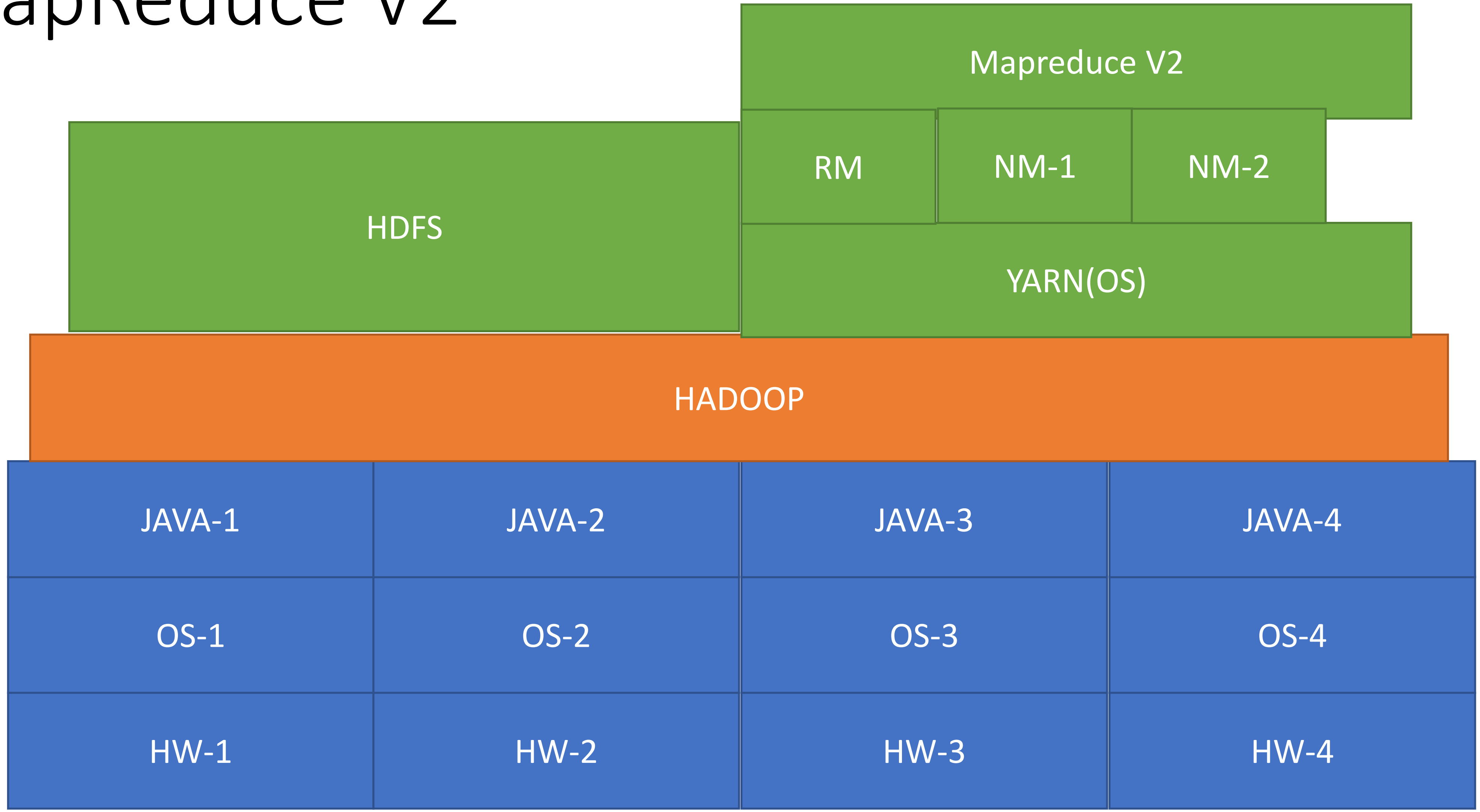
HW-1

HW-2

HW-3

HW-4

MapReduce V2



Mapreduce V2

RM

NM-1

NM-2

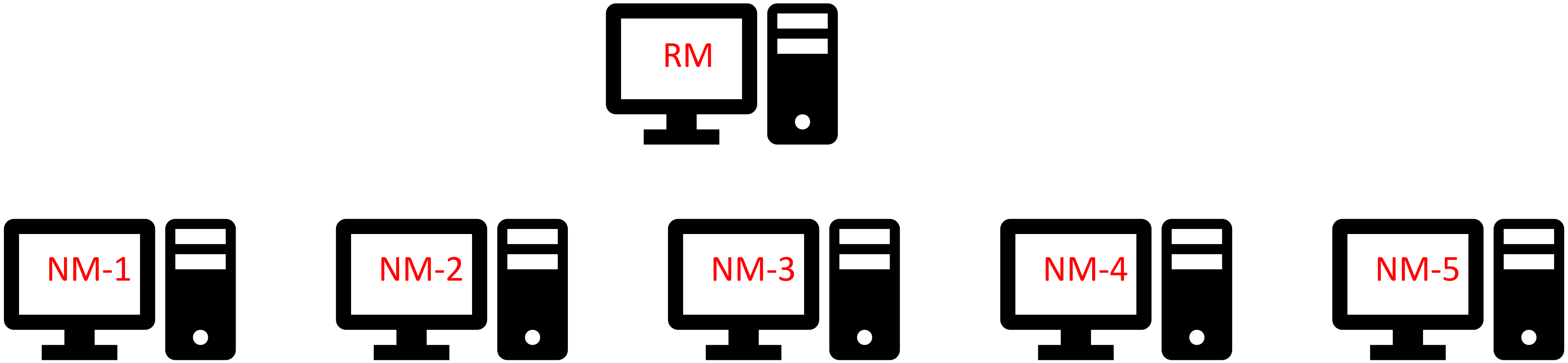
HDFS

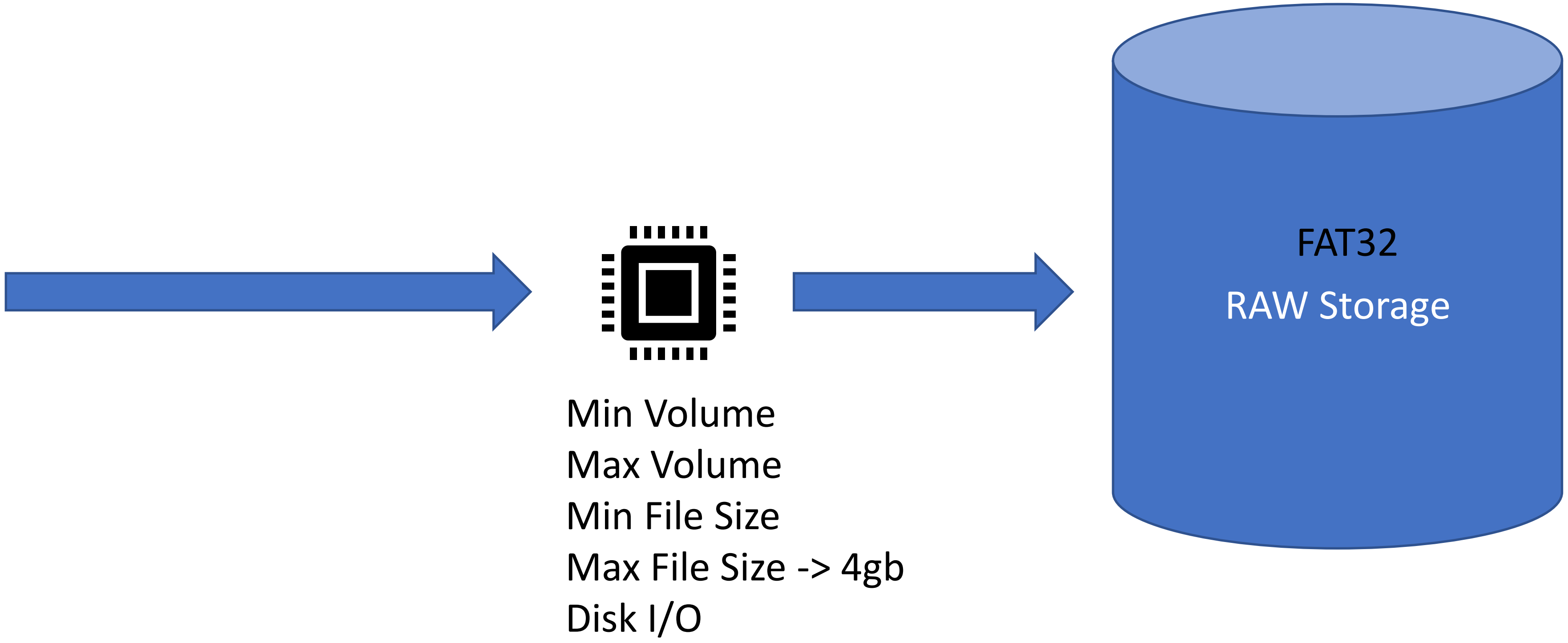
YARN(OS)

HADOOP

JAVA-1	JAVA-2	JAVA-3	JAVA-4
OS-1	OS-2	OS-3	OS-4
HW-1	HW-2	HW-3	HW-4

MapReduce V1





FAT32

RAW Storage

Min Volume
Max Volume
Min File Size
Max File Size -> 4gb
Disk I/O

HDFS

- Distributed File System -> Hadoop Framework -> Resources(Multiple Systems) -> Raw Storage(JAVA)
- If you want to us that raw storage -> Format -> HDFS(JAVA)
- File System -> distributed Storage
- Failover
- No min and max volume, no min and max file size
- HDFS -> Blocks -> Default
    - V1 -> 64MB
    - V2 -> 128MB
- Redundancy -> 1+2 Replication Factor -> Rack Awareness Algorithm
    - V2 -> 1 PB * 2 -> 3 PB
    - V3 -> 6GB -> 9GB

- 500 MB
- Version 1 -> ? Blocks
- Version 2 -> ? Blocks

Functions of Name Node

- It maintains the master daemon that maintains and manages the Data Nodes(Slaves)
- It records the meta data of all the files stored in the cluster e.g the location of the blocks stored, the size of the files, permissions, Hierarchy etc..
    - There are 2 special files are associated with the meta data
    - FsImage :- it contains the complete state of the file system namespace since the start of the Name node
    - EditLogs : - it contains all the recent modifications made to the file system with respect to the most recent FsImage

Functions of Name Node

- It records each change that take place to the system metadata ex: if the file is deleted in HDFS, the name node will immediately record in the EditLogs
- It regularly receives a Heartbeat and a block report form all the data nodes in the cluster to ensure the data node is alive
- It keeps a record of all the blocks in HDFS and which nodes these bocks are located
- The name node is also responsible to take care of the replication factor of all the blocks
- In case of the data node failure the name nodes choses new data node for new replicas, balance the disk usage and manages the communication traffic to the data nodes

↓

Metadata

fs image → (name node) (master)

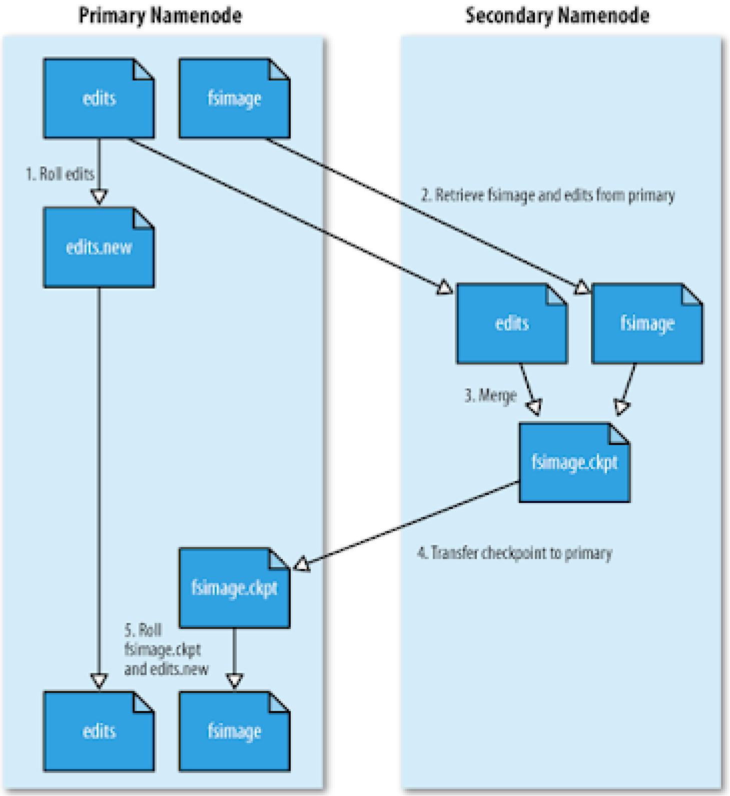
Edit logs (Recent changes)

(10 secs)

depends ⇒ Back up ⇒ fs image ⇒ name node

Recent → fs image ⇒ name node

↳ Secondary name → name node


Primary Namenode
Secondary Namenode
edits
fsimage
1. Roll edits
edits.new
2. Retrieve fsimage and edits from primary
edits
fsimage
3. Merge
fsimage.ckpt
4. Transfer checkpoint to primary
fsimage.ckpt
5. Roll fsimage.ckpt and edits.new
edits
fsimage

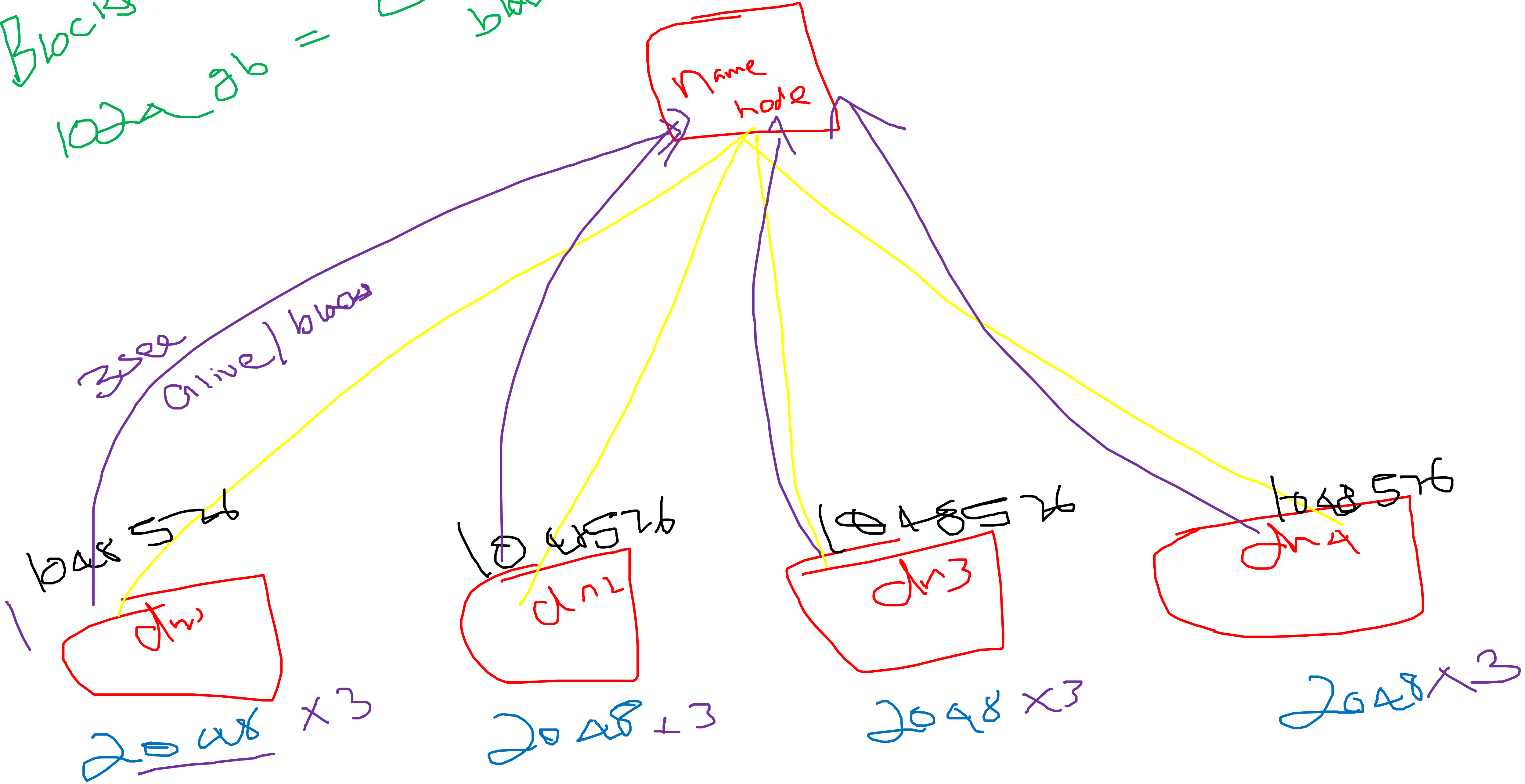
Functions of Data Node

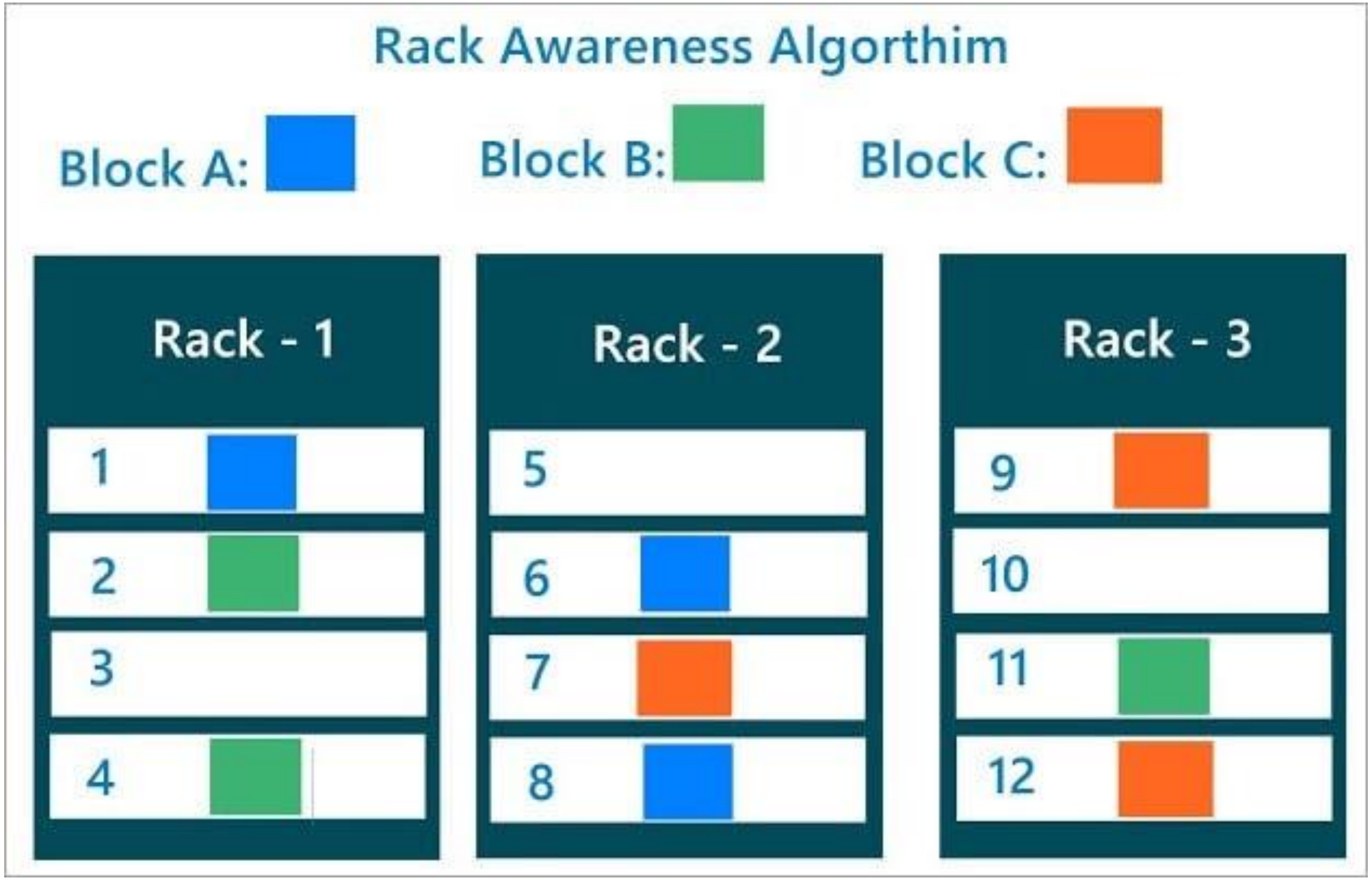
- These are slave daemons which runs on each slave machine
- The actual data is stored on data nodes
- The data nodes perform the low-level read and write requests from the clients directly
- They send the heart beats to the name node periodically to report overall health of HDFS, by default this frequency is set to 3 seconds

V2

Blocks -> 128 mb

1024 gb = 8192 blocks



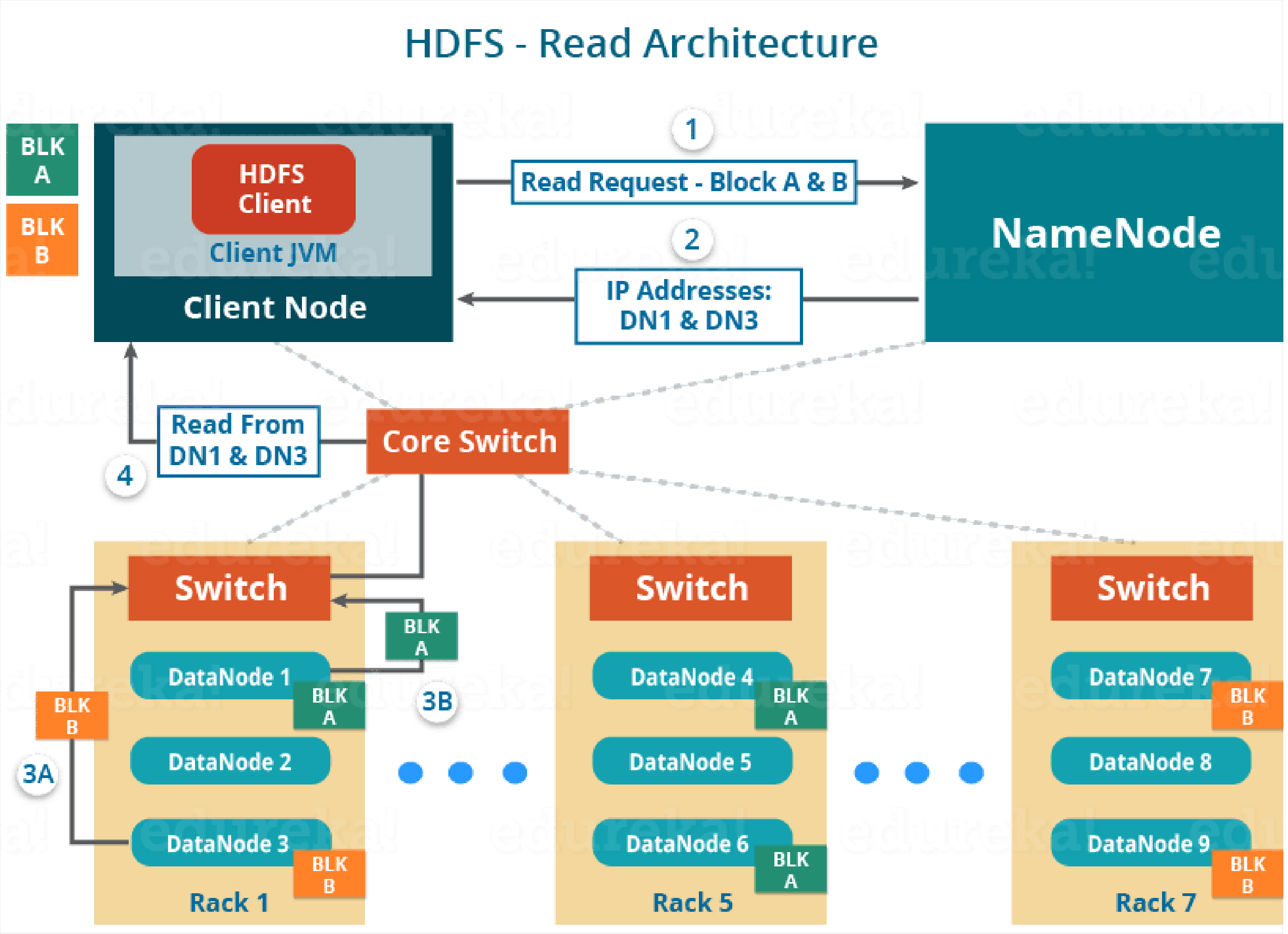


Rack Awareness Algorthim

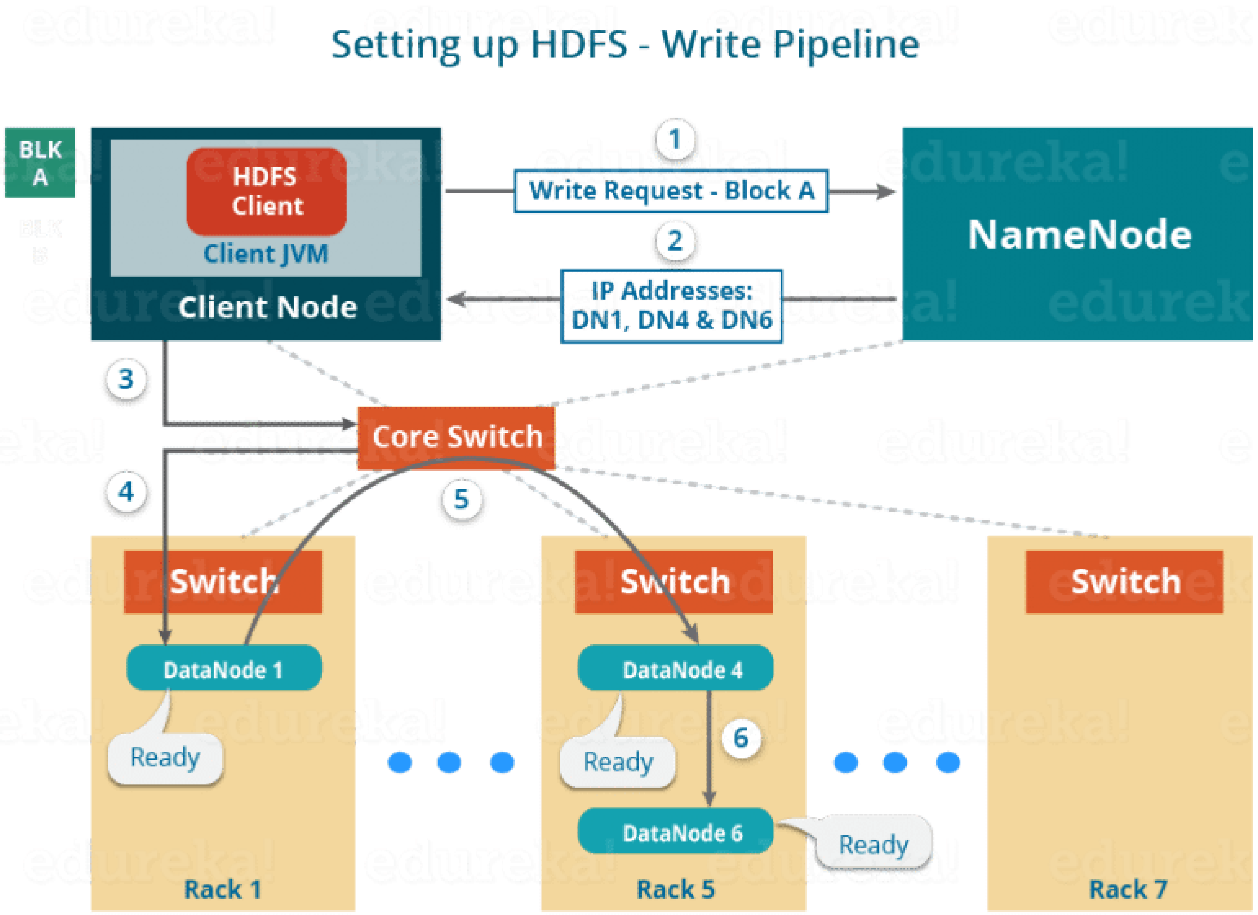
Block A: 🟦 Block B: 🟩 Block C: 🟧

Rack - 1	Rack - 2	Rack - 3
1 🟦	5	9 🟧
2 🟩	6 🟦	10
3	7 🟧	11 🟩
4 🟩	8 🟦	12 🟧

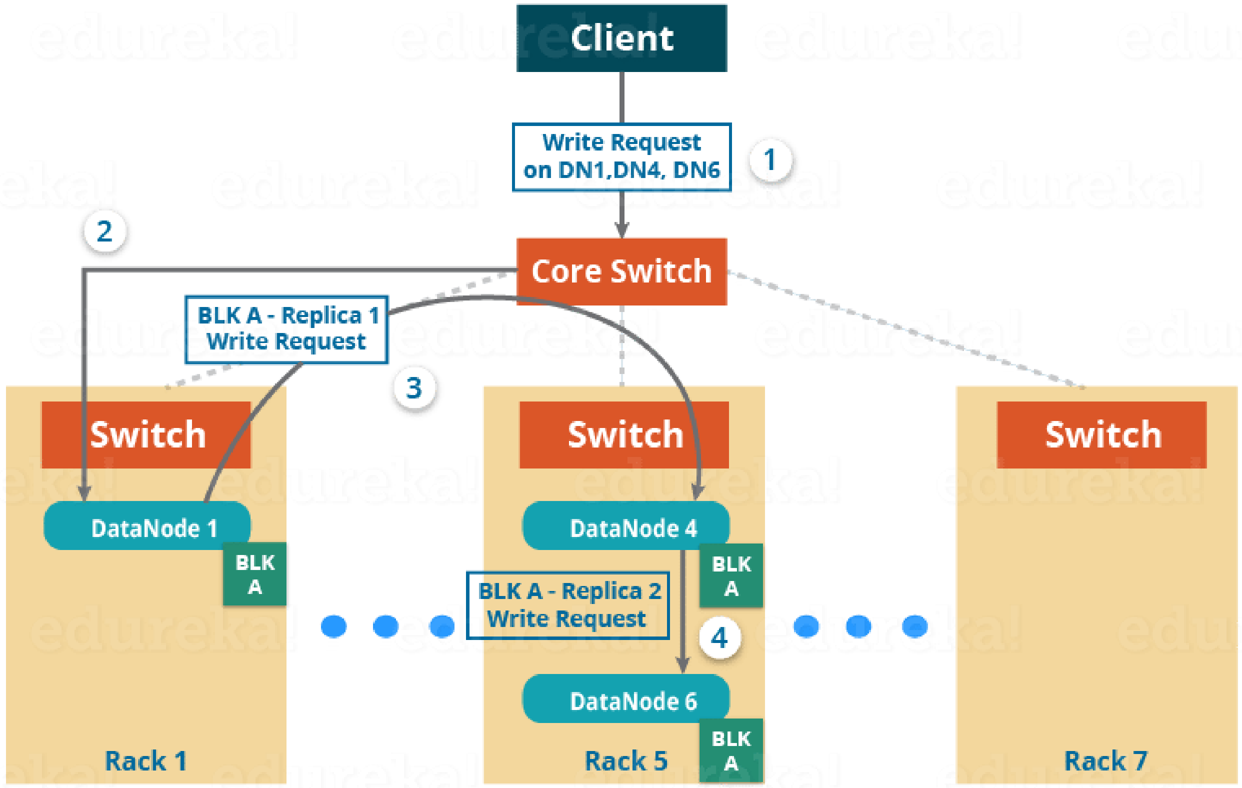
HDFS -> Read and Write



HDFS Write


Setting up HDFS - Write Pipeline
BLK A
BLK B
HDFS Client
Client JVM
Client Node
1
Write Request - Block A
2
IP Addresses: DN1, DN4 & DN6
NameNode
3
Core Switch
4
5
Switch
DataNode 1
Ready
Rack 1
Switch
DataNode 4
Ready
6
DataNode 6
Ready
Rack 5
Switch
Rack 7

HDFS - Write Pipeline



- OLAP -> Editing is not allowed

HDFS

MapReduce

- Distributed Computing Component that is available on Hadoop
- If you schedule any Mapreduce task
- V1
  - Job Tracker
  - Task Tracker
- V2
  - Resource Manager
  - Node Manager

How MapReduce Makes it possible?

Electral →

Banglore

2022 → adding & Removal

Car

→ ~~8 - 10~~

(1 - 2 year)

1000

96la

area 2nd

Cust 2nd

South 2g

North 2a

area

Ara

6 mont

(6 - 10 days)

10 core
12 gb
100 mbps

⇒ Same Job

18 hours

hadoop
map r

18 Computer → Cluster

+ 1 mas

+ sec

1 hour

YARN(Yet another Resource Negotiator)

- Version 1 -> Mapreduce V1 -> Resource allocation and Job Handling
- In version2 -> Hadoop -> they separated the job handler and the resource allocation by means of 2 different principles
- MapReduce v2 -> Actual job will run on MapReduce v2 under the guidance YARN
- YARN -> Allocating the resources that is required for running the job

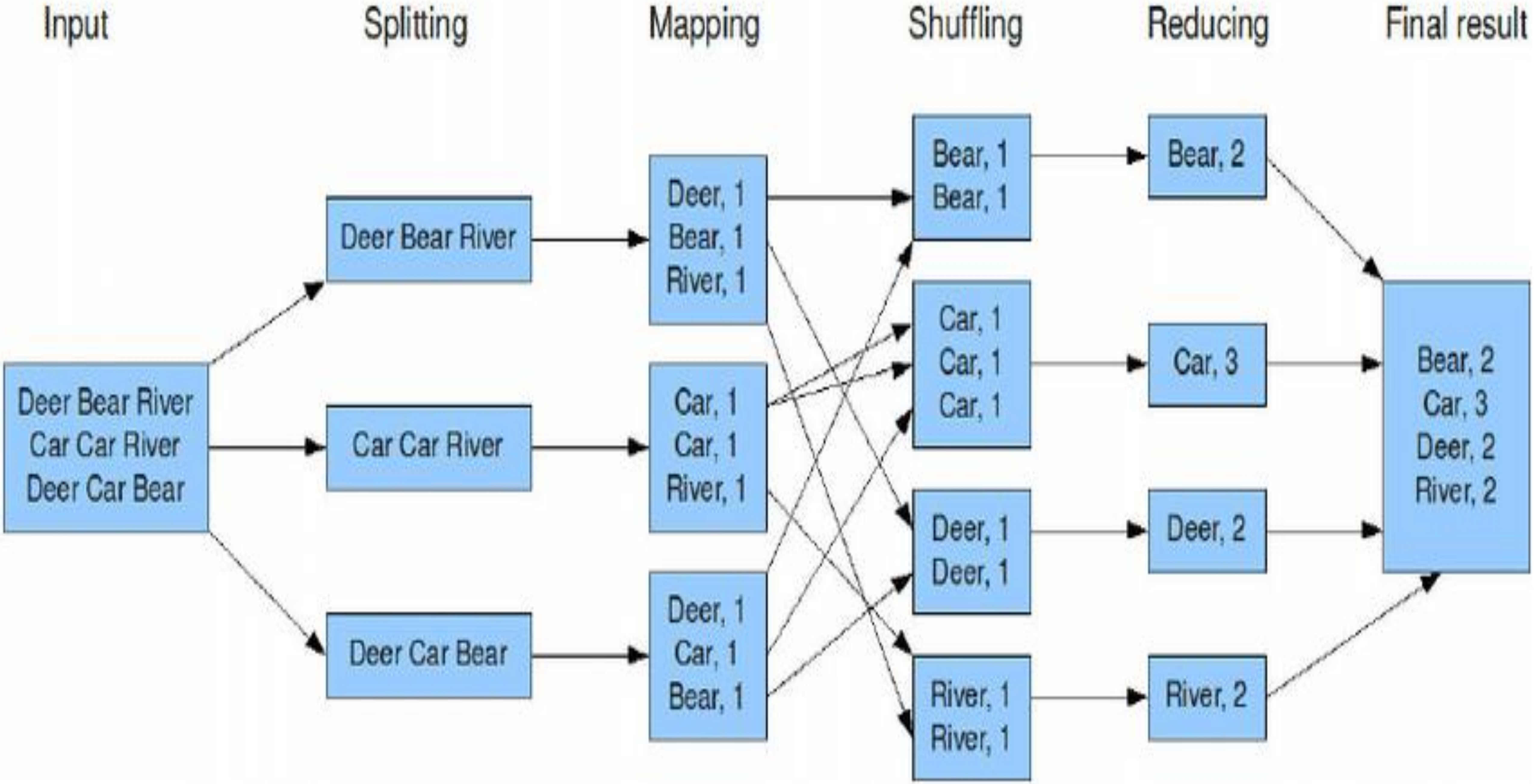
YARN(Yet another Resource Negotiator)

- It was developed by the company Horton Works
- Idea was introduced this technology which will act as OS
- YARN -> Distributed OS -> interface to manage the cluster hardware(entire cluster)
- Cluster Capacity -> only from the Worker Nodes

YARN

- Distributed OS
  - Managing the Cluster Hardware(Cluster Capacity)
  - Resource allocation for Apps/Tasks
  - Monitor the health of entire cluster(Worker Node down, it will try to run the job somewhere else, wait for heartbeat from the worker node which is down)
    - Heartbeat -> kind of communication message that is send from Worker node to Master node(inform the worker node is alive)
  - Logging -> log file aggregation property is there
  - Pluggable -> install(Spark, Tez, MapReduce)
  - Prioritization -> priority to job

The overall MapReduce word count process



Input

Splitting

Mapping

Shuffling

Reducing

Final result

Deer Bear River
Car Car River
Deer Car Bear

Deer Bear River

Car Car River

Deer Car Bear

Deer, 1
Bear, 1
River, 1

Car, 1
Car, 1
River, 1

Deer, 1
Car, 1
Bear, 1

Bear, 1
Bear, 1

Car, 1
Car, 1
Car, 1

Deer, 1
Deer, 1

River, 1
River, 1

Bear, 2

Car, 3

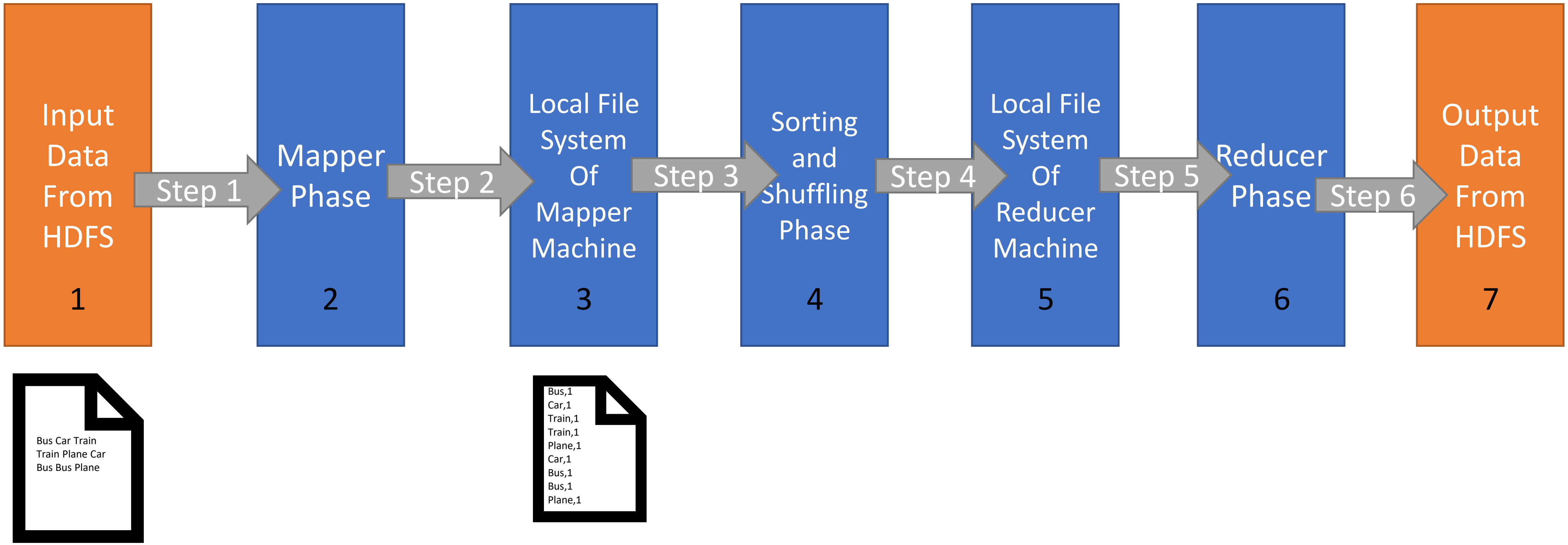
Deer, 2

River, 2

Bear, 2
Car, 3
Deer, 2
River, 2

MapReduce -> Logical View

- Input and Output data Only from HDFS -> why? -> Java based Distributed storage with Fault Tolerant, Horizontally Scalable



Input Data From HDFS 1

Step 1

Mapper Phase 2

Step 2

Local File System Of Mapper Machine 3

Step 3

Sorting and Shuffling Phase 4

Step 4

Local File System Of Reducer Machine 5

Step 5

Reducer Phase 6

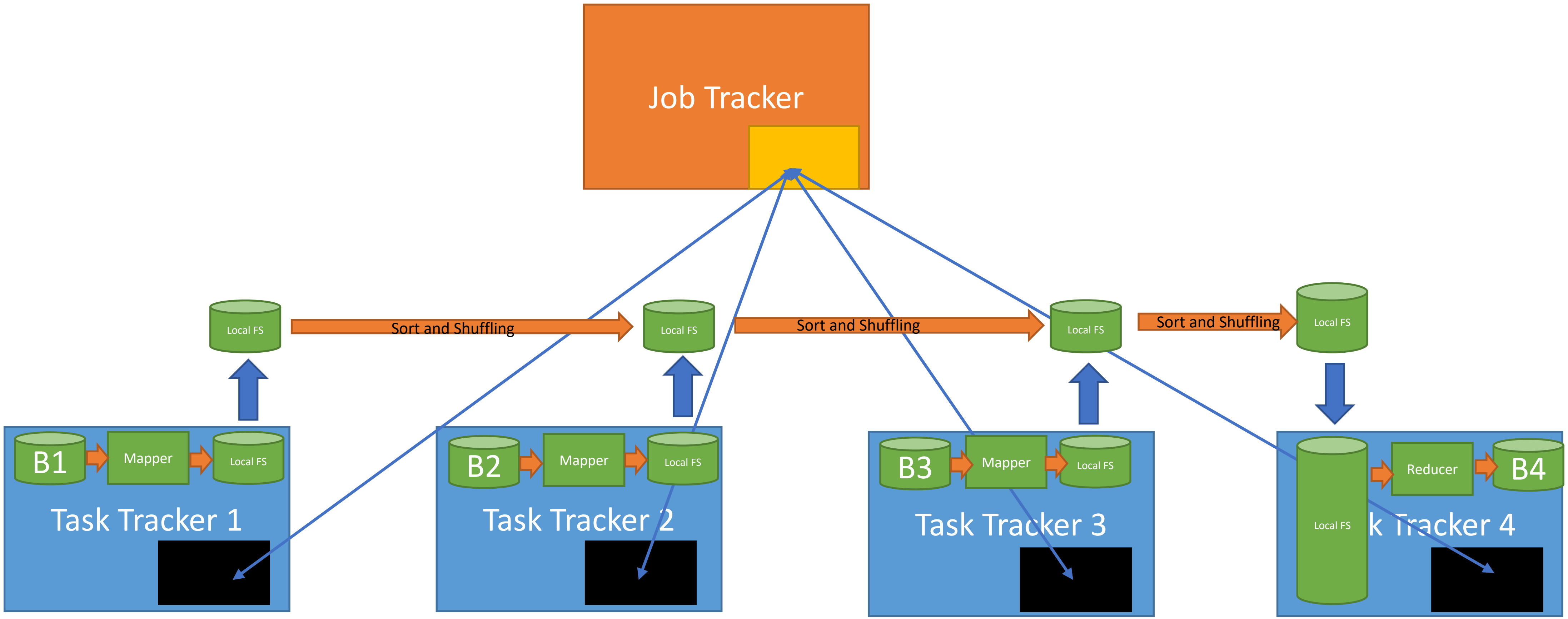
Step 6

Output Data From HDFS 7

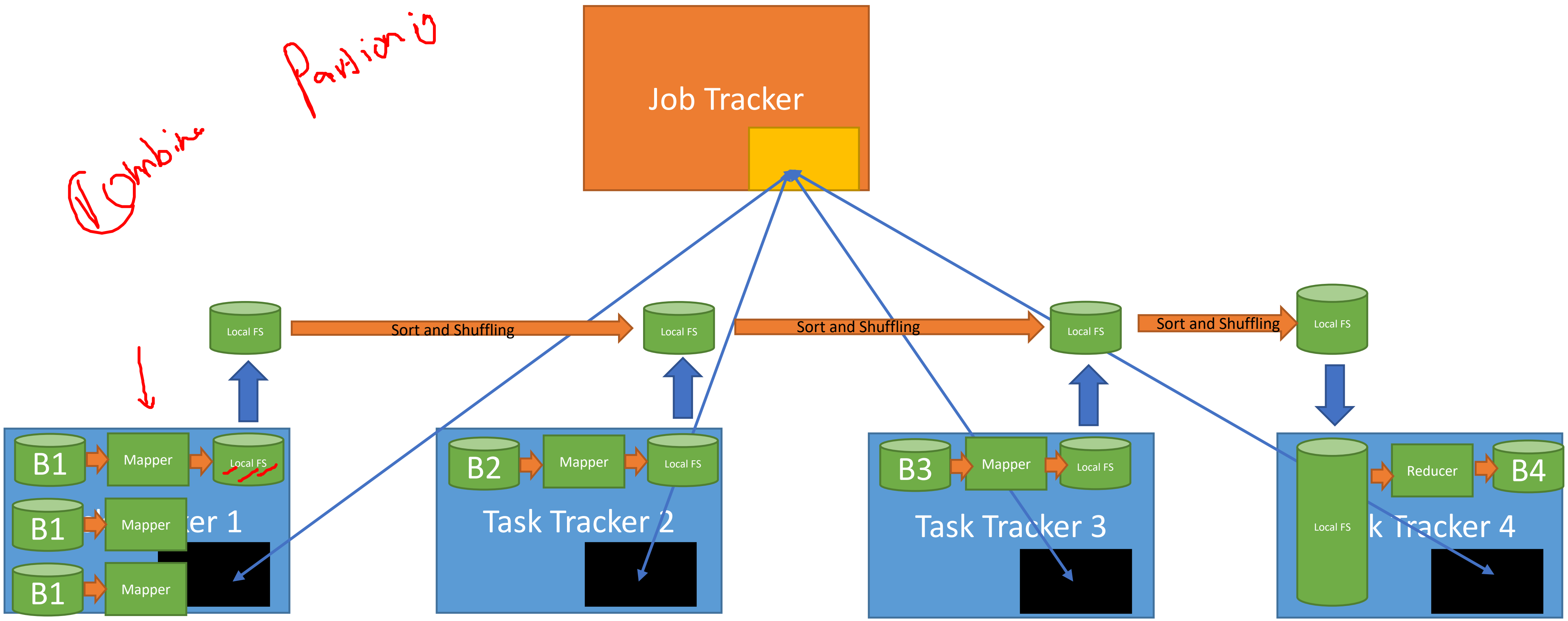
Bus Car Train
Train Plane Car
Bus Bus Plane

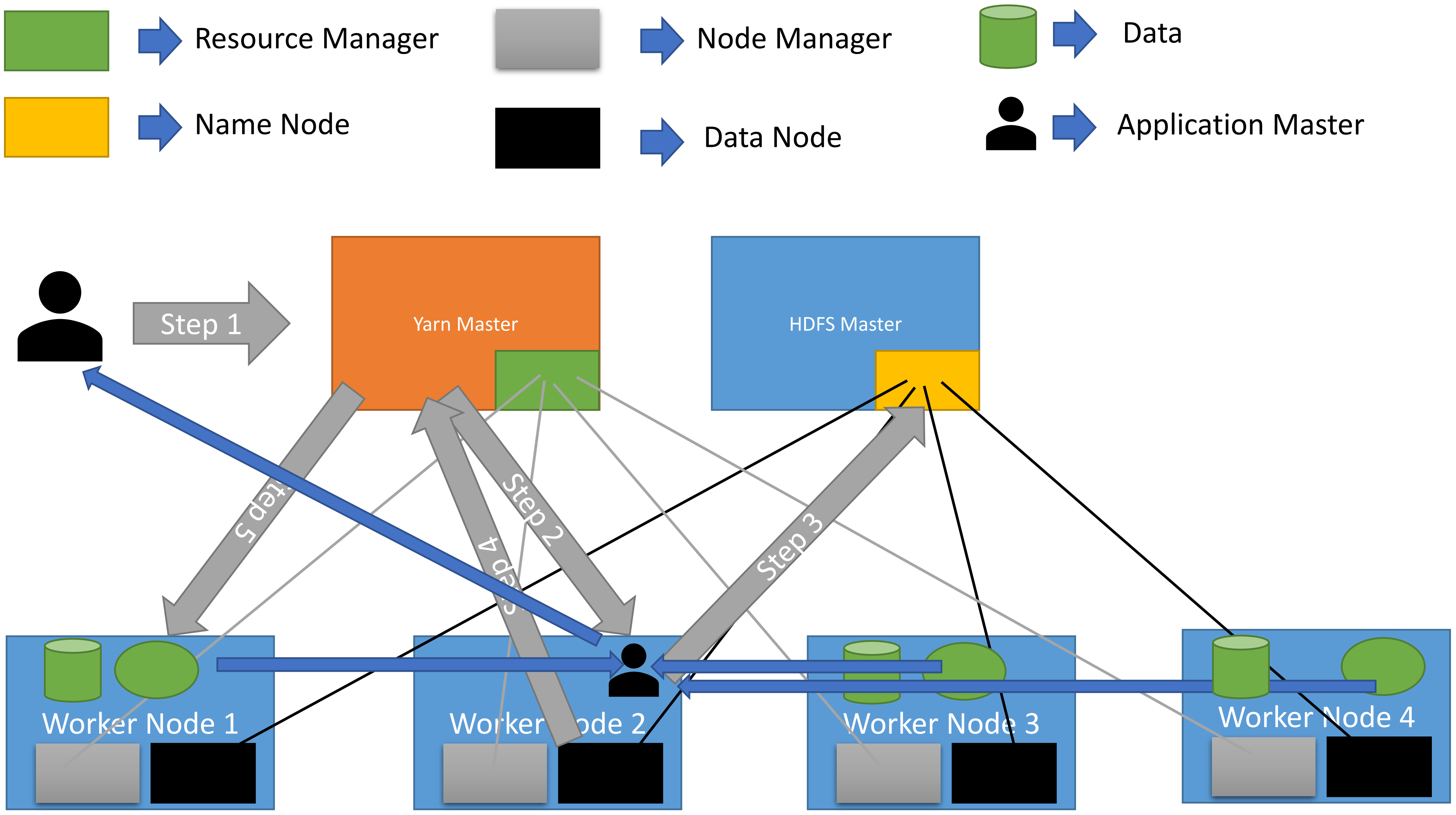
Bus,1
Car,1
Train,1
Train,1
Plane,1
Car,1
Bus,1
Bus,1
Plane,1

MapReduce -> Physical View



MapReduce -> Physical View




Resource Manager
Node Manager
Data
Name Node
Data Node
Application Master
Step 1
Yarn Master
HDFS Master
Step 2
Step 3
Step 4
Step 5
Worker Node 1
Worker Node 2
Worker Node 3
Worker Node 4