

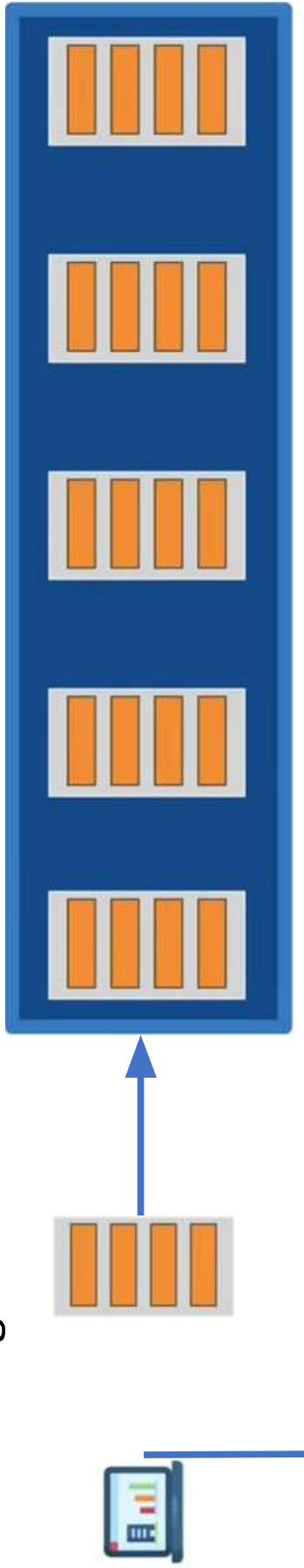
# Hadoop Architecture

Dr Milan Joshi

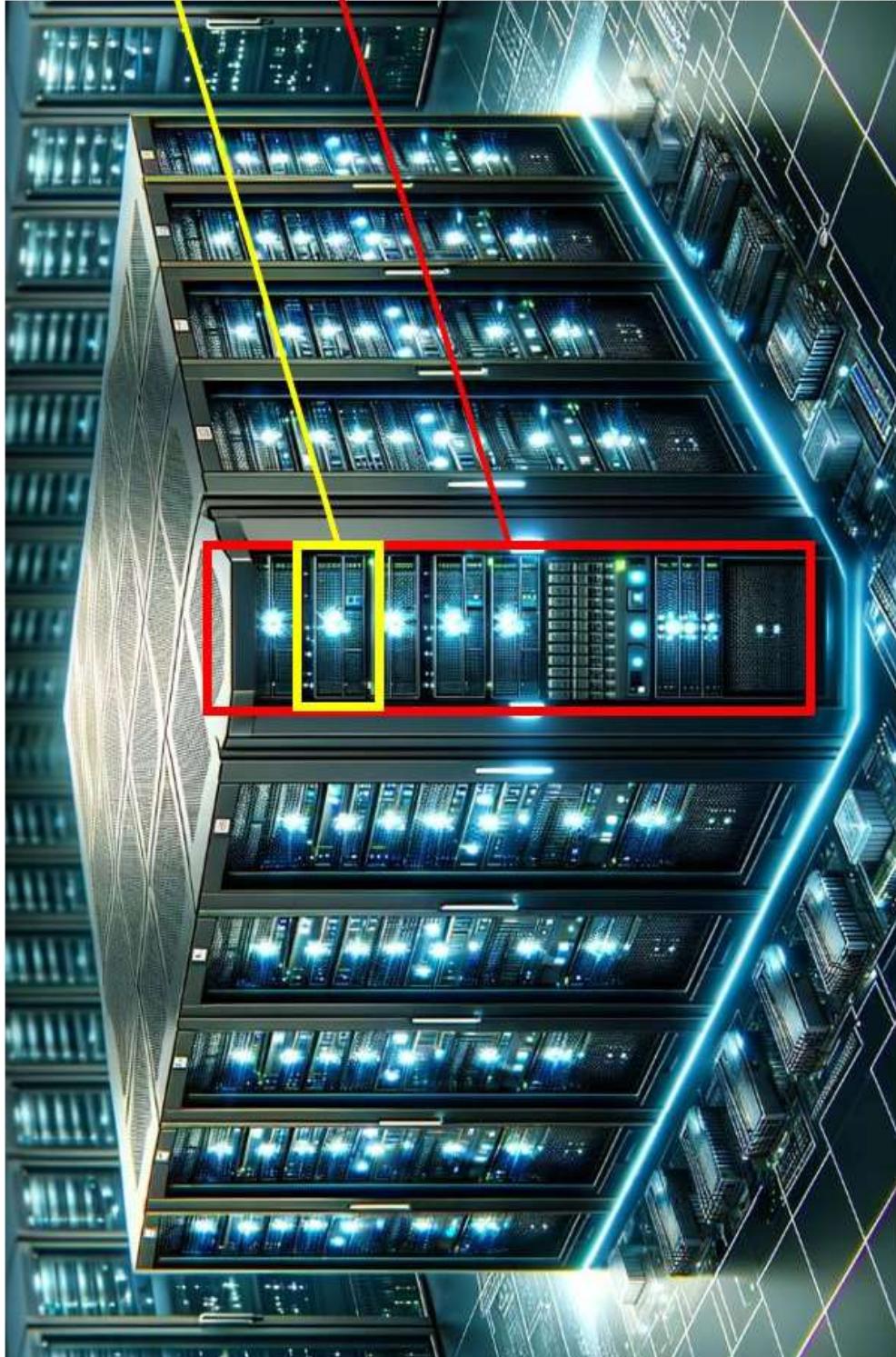
# Apache Hadoop



- **Node:** Node is a single **commodity hardware** which is used to store the actual data. Node can be physical computer or virtual machine.
- **Commodity hardware:** Inexpensive, easily accessible, and interchangeable computing systems or components that are widely available in the market.
- **Rack:** Rack is the collection of nodes. Mostly 30 to 40 nodes exist in a single rack.
- **Cluster:** Cluster is the collection of racks that are networked together to perform parallel computation on big data sets



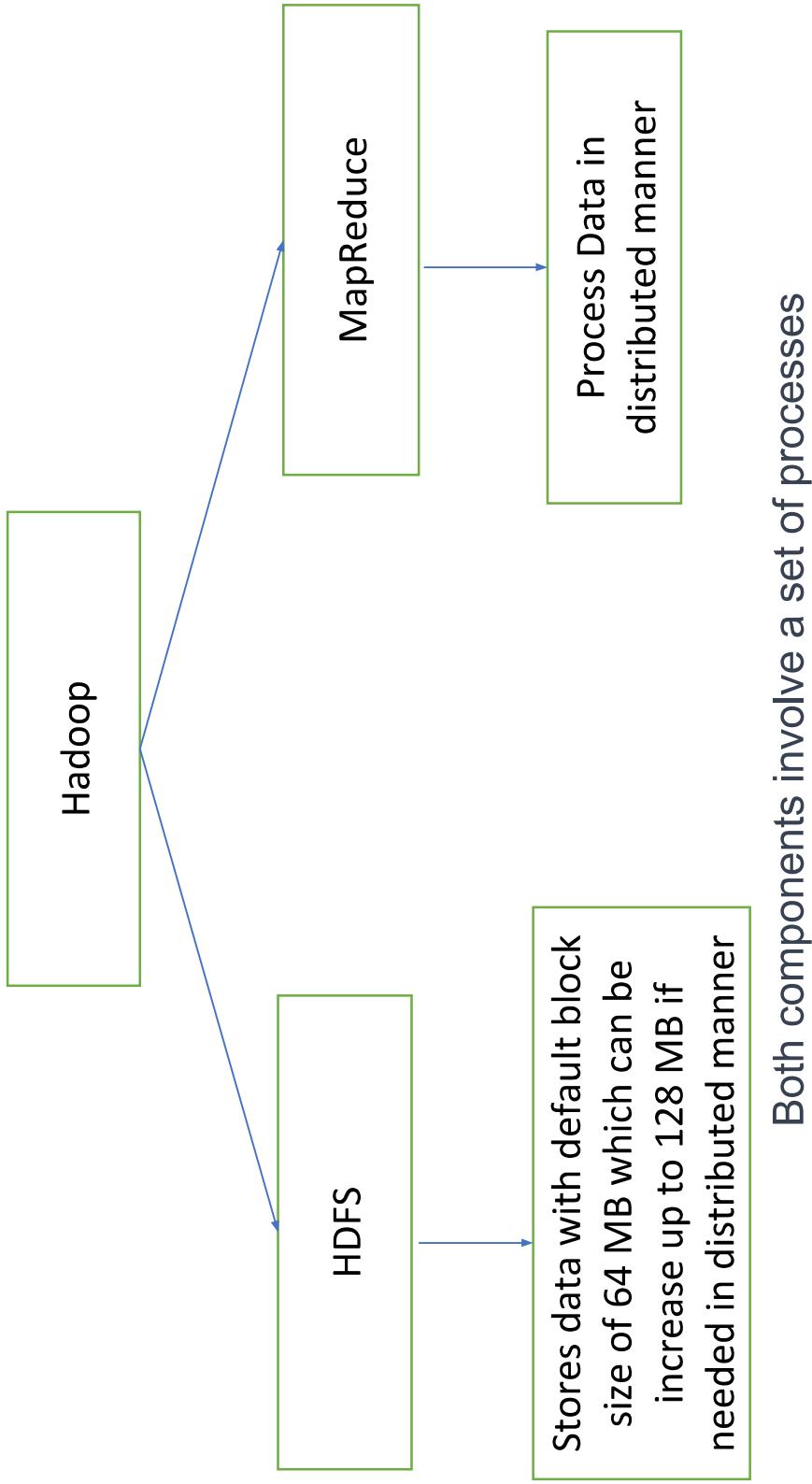
# Cluster



Node  
Rack

# Architecture

- **Hadoop 1.X** provides a distributed storage system called **HDFS** or Hadoop Distributed File System and **MapReduce** to tackle big data challenges.



# Architecture

- **Daemons** refers to the background processes that run on the servers in a Hadoop cluster.
- These daemons are continuously running services that perform the necessary tasks to manage and process data.
- Often, a daemon waits for specific events to occur or provides certain services.

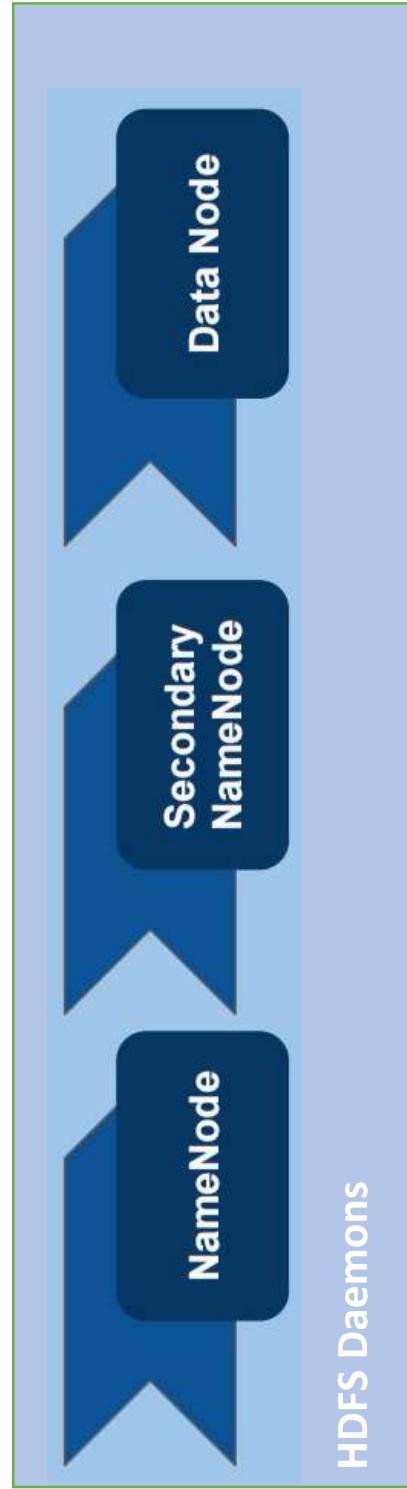
MapReduce Daemons

HDFS Daemons

# Architecture

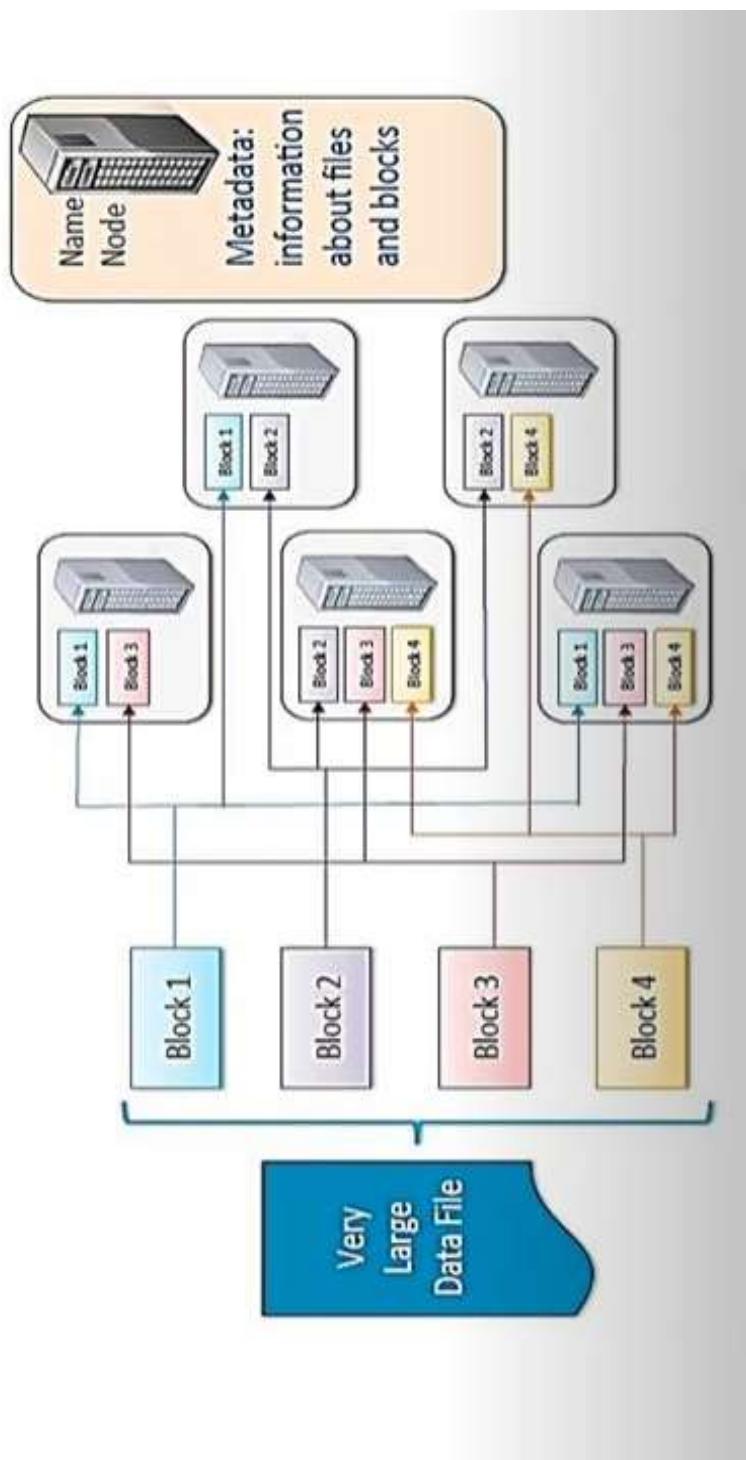
□ In Hadoop, there are three main daemons:

1. NameNode
2. Secondary NameNode
3. DataNode

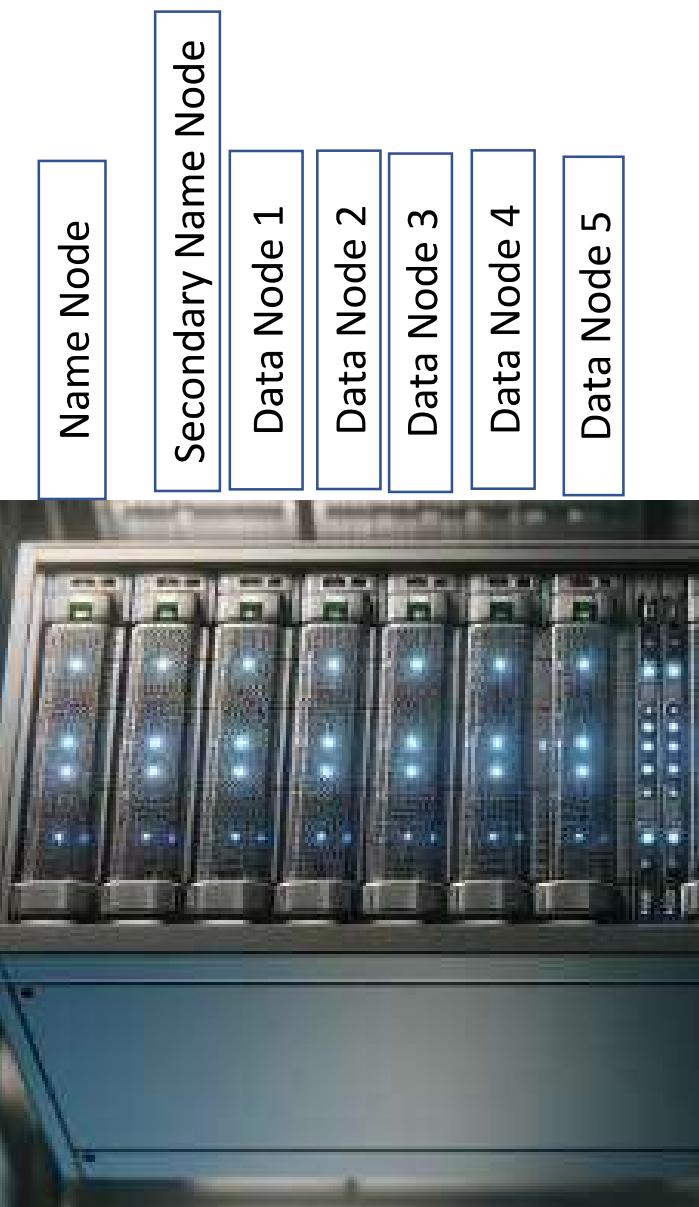


# How file is stored in HDFS

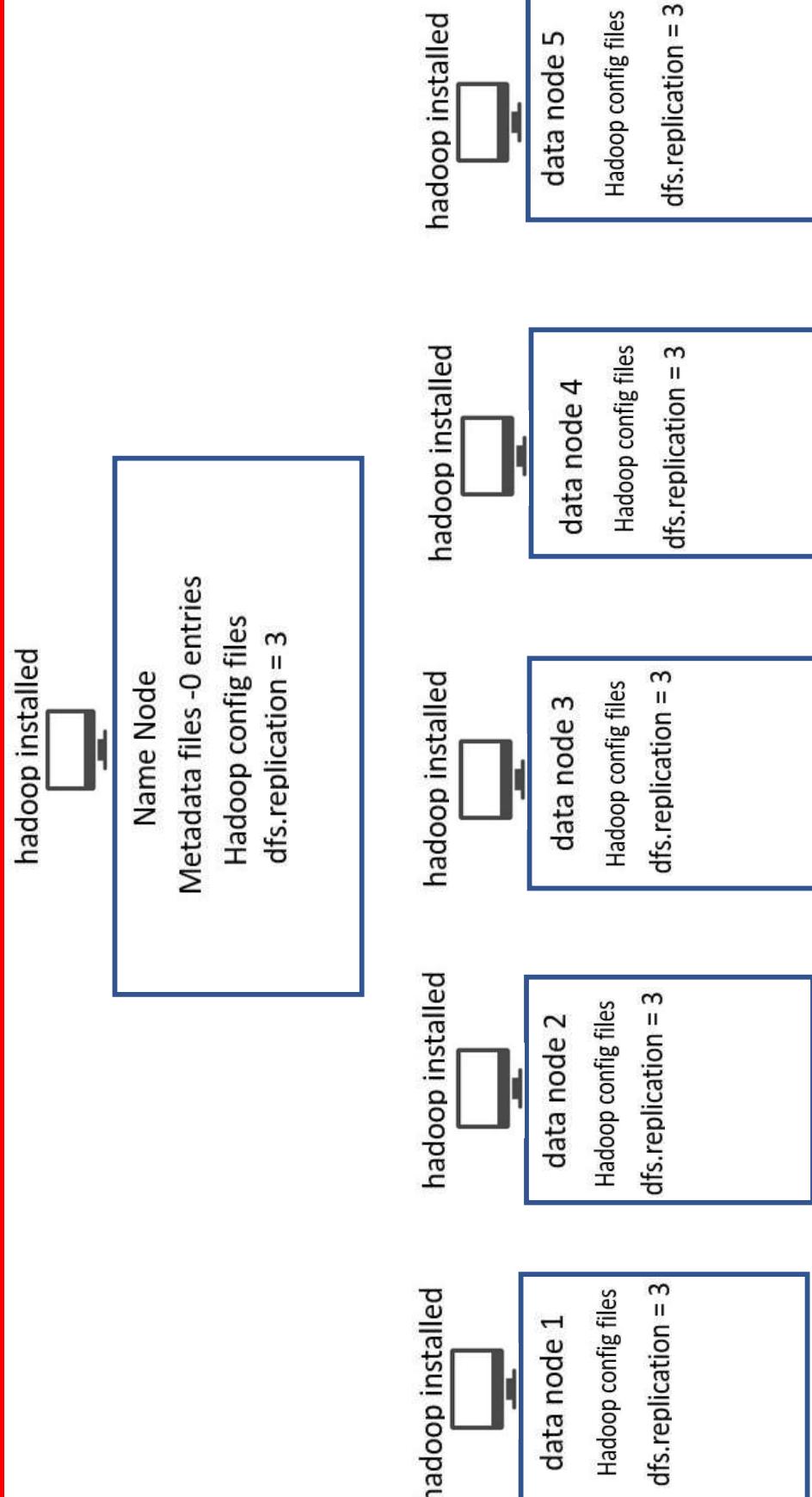
- A very large data file is split into four smaller blocks (Block 1, Block 2, Block 3, Block 4)
- Each block is then replicated 3 times across different data node machines
- A name node machine stores metadata about the files and their respective blocks



- When you install Hadoop it comes with configuration files, inside a file we have a property called `dfs.replication` with default value 3 , we can change it



# How file is stored in HDFS



hadoop installed



# How file is stored in HDFS(IP address)

dfs.replication = 3

IP address:192.168.56.181

dfs.blocksize = 64 MB

hadoop installed



Name Node

Metadata files -0 entries

Hadoop config files

dfs.replication = 3

IP address:192.168.56.181

hadoop installed



Name Node

Hadoop config files

dfs.replication = 3

Name node =192.168.56.181

IP Address :192.168.56.186

hadoop installed



Name Node

Hadoop config files

dfs.replication = 3

Name node =192.168.56.181

IP Address :192.168.56.185

hadoop installed



Name Node

Hadoop config files

dfs.replication = 3

Name node =192.168.56.181

IP Address :192.168.56.184

hadoop installed



Name Node

Hadoop config files

dfs.replication = 3

Name node =192.168.56.181

IP Address :192.168.56.183

hadoop installed



Name Node

Hadoop config files

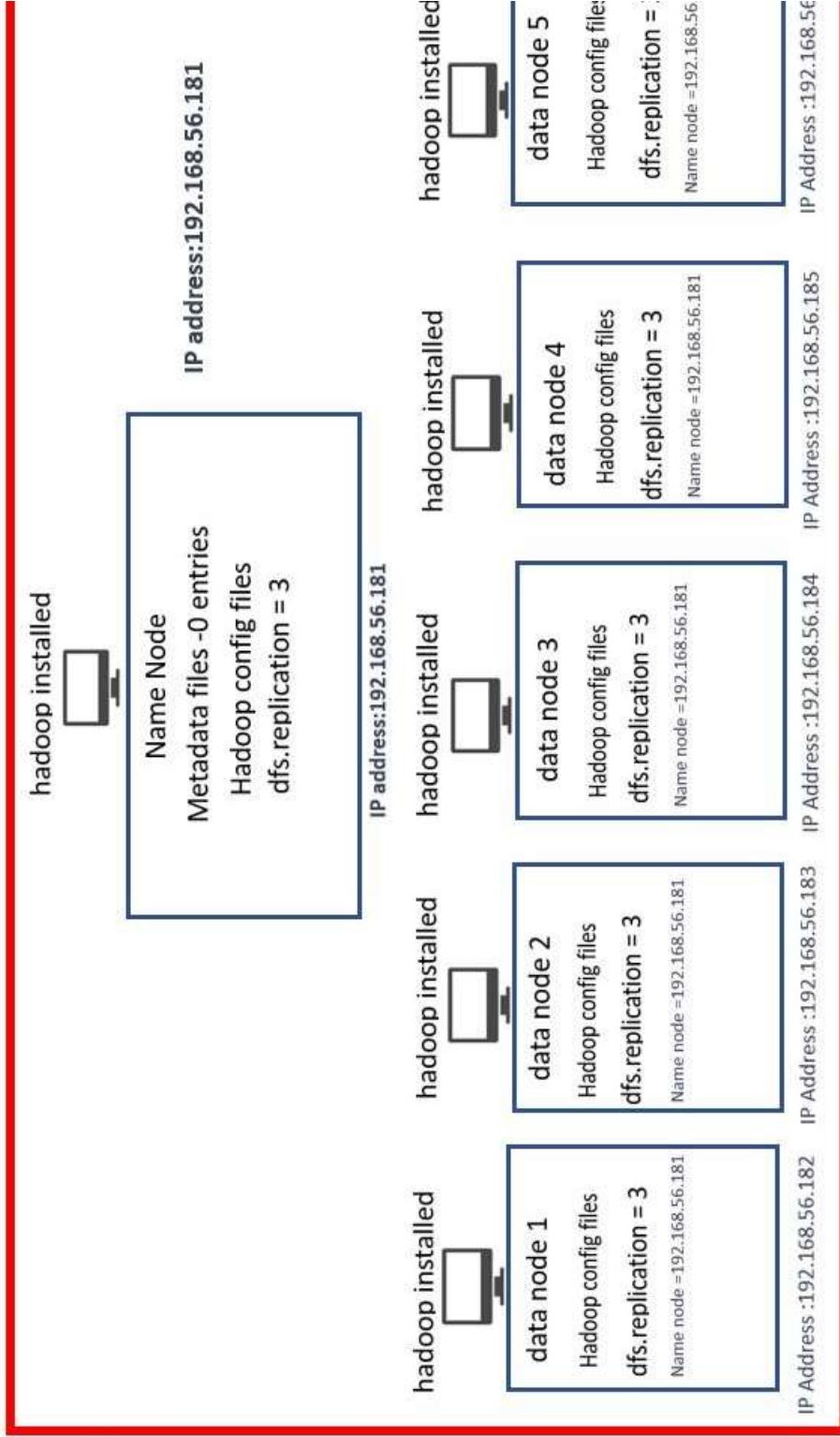
dfs.replication = 3

Name node =192.168.56.181

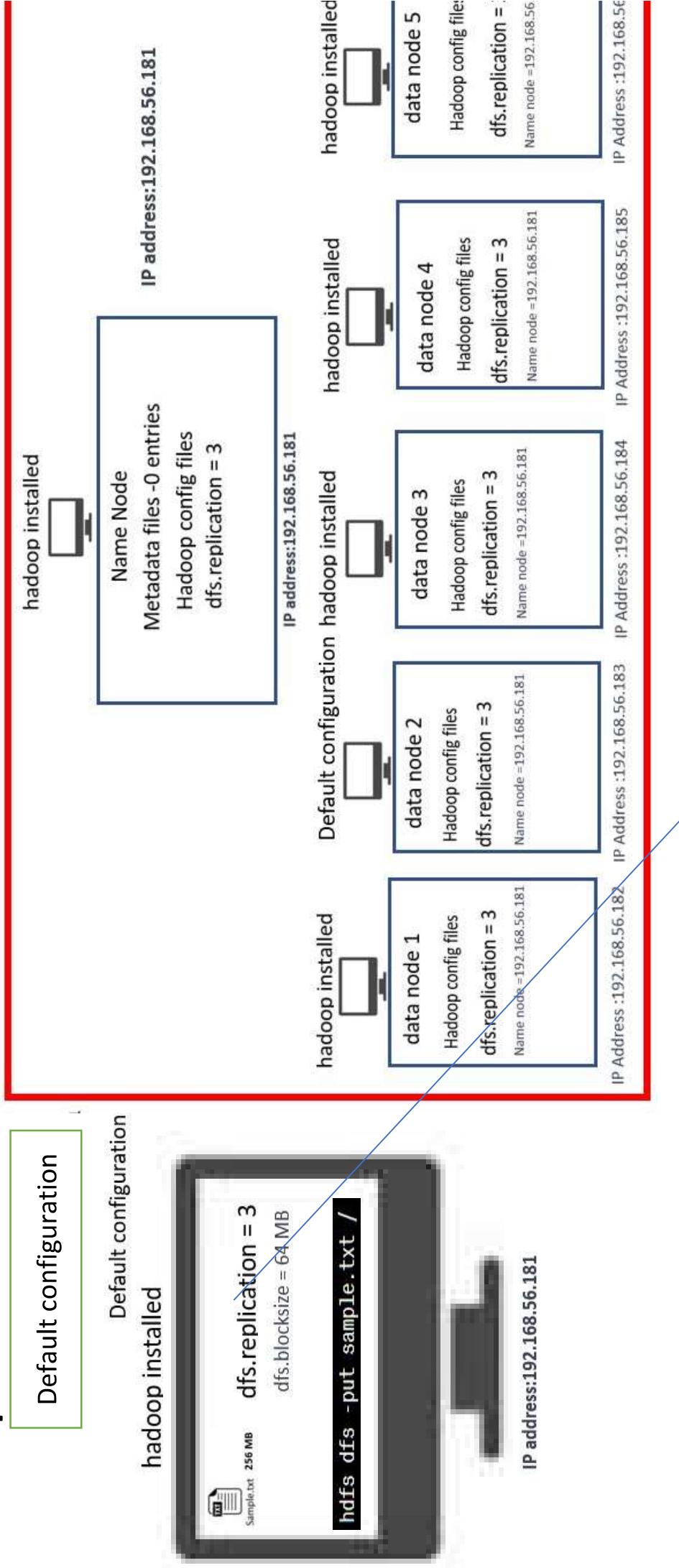
IP Address :192.168.56.182

## command

# How file is stored in HDFS-Hadoop



# How file is stored in HDFS-default replication



# How file is stored in HDFS

BLK003

Default configuration  
hadoop installed

```
Sample.txt 256 MB      dfs.replication = 3
           dfs.blocksize = 64 MB
hdfs dfs -put sample.txt /
```

IP address:192.168.56.181

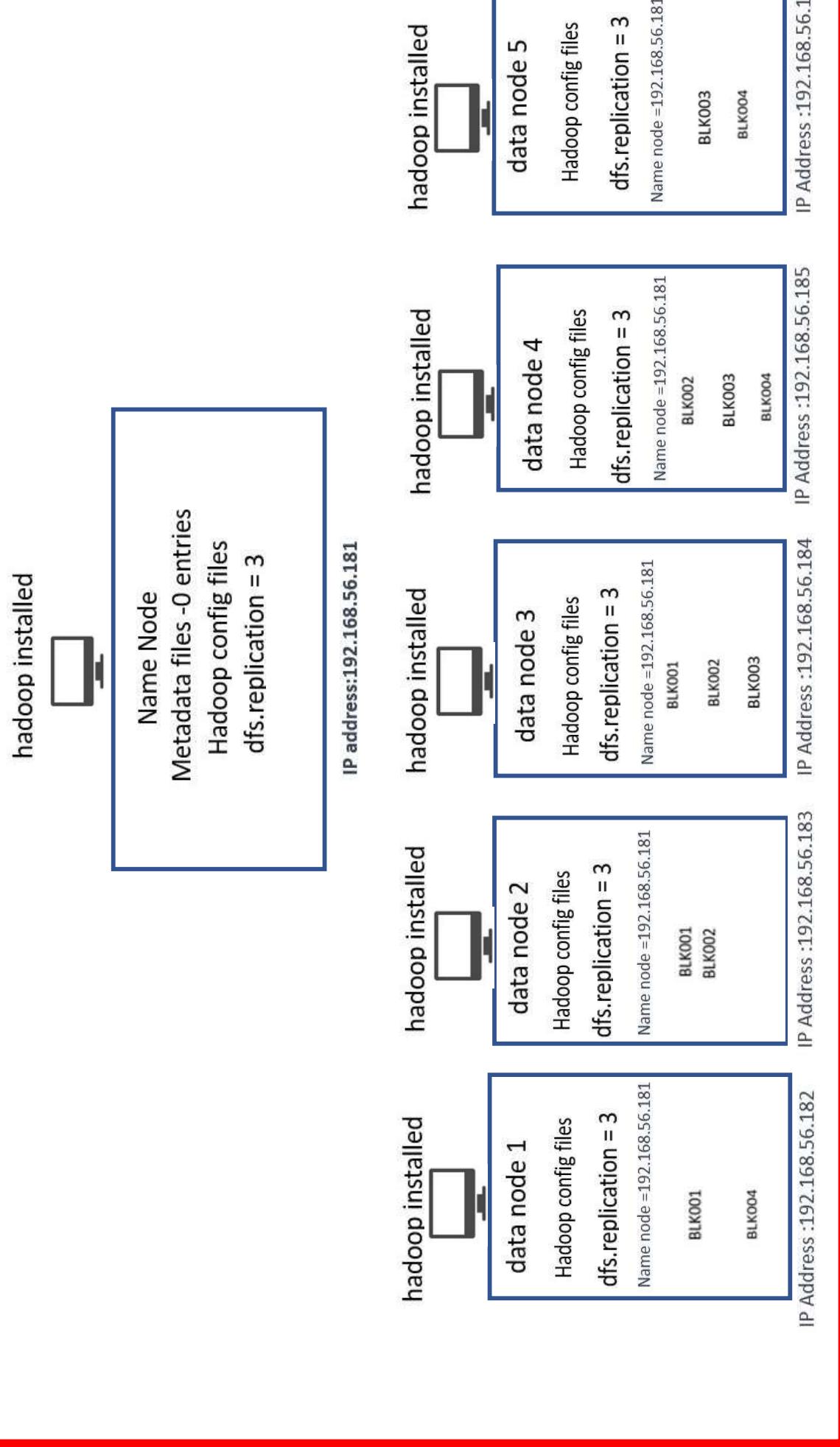
Default configuration

64 MB  
64 MB  
256 MB  
64 MB  
64 MB

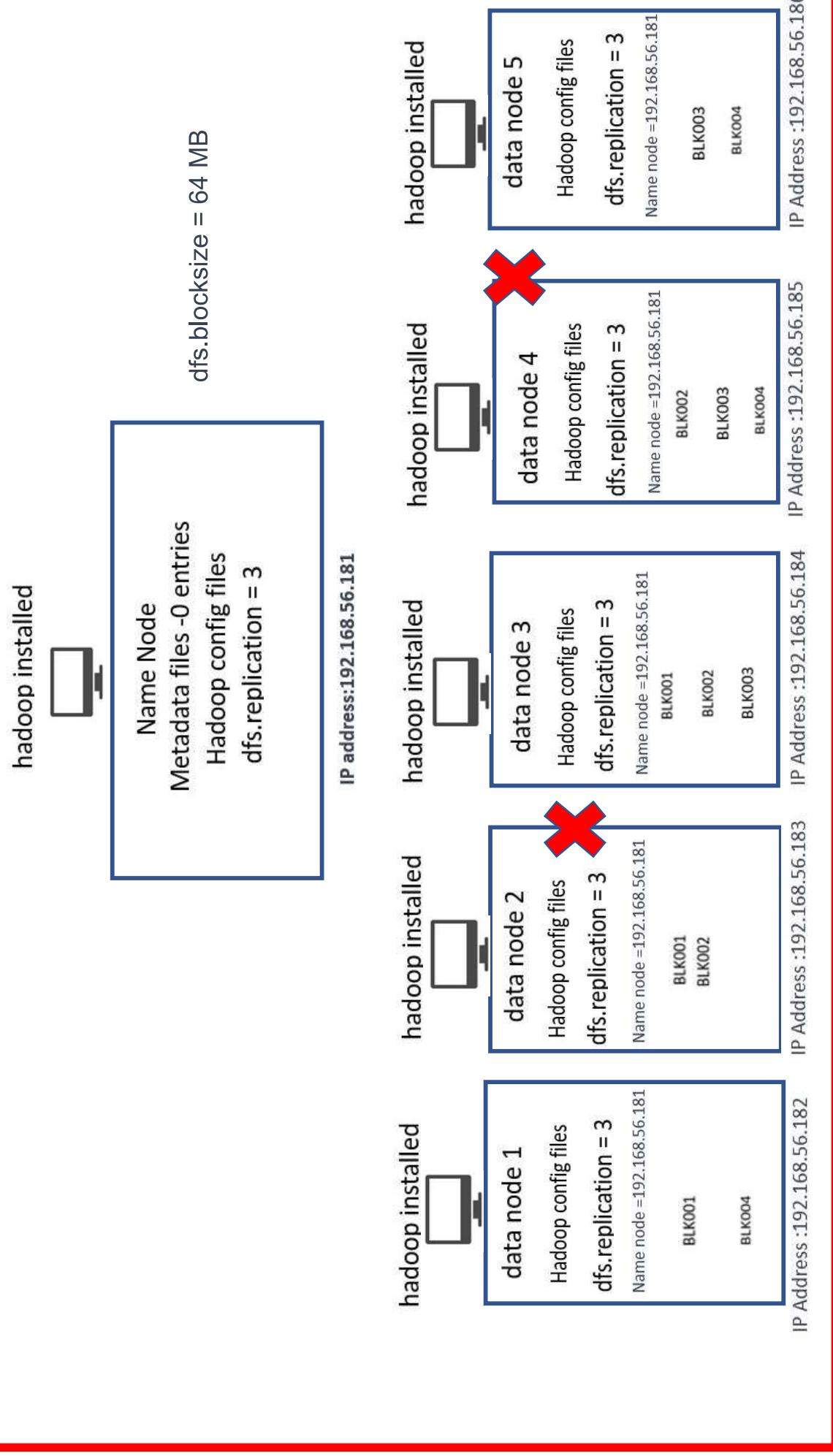
Default configuration

64 MB	BLK001
64 MB	BLK002
250 MB	
64 MB	BLK003
58 MB	BLK004

# How file is stored in HDFS(blocks)



# How file is stored in HDFS ( why replication )



# How file is stored in HDFS-custom config

hadoop installed

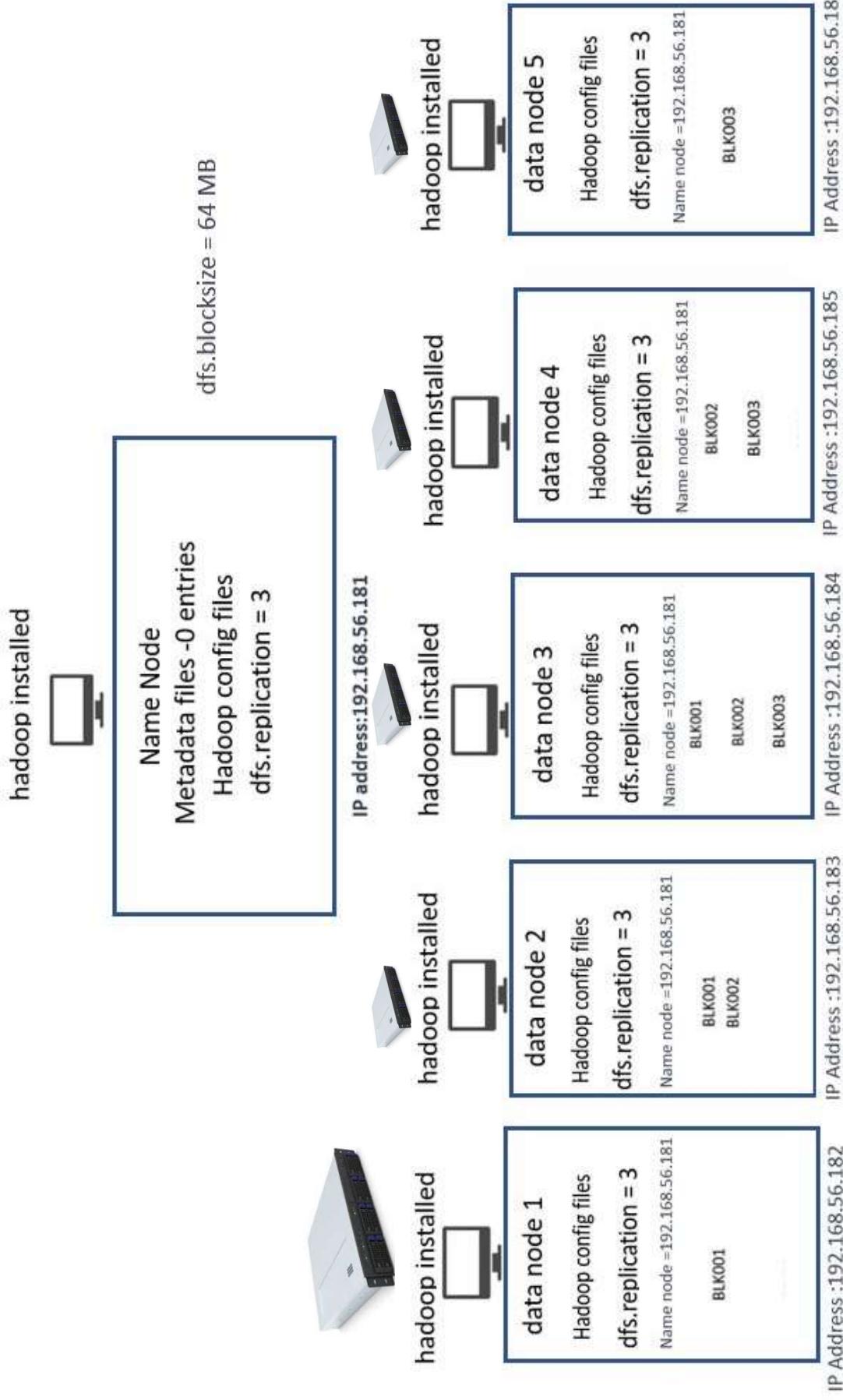
dfs.replication = 3

Sample.txt 256 MB	Sample.txt-256 MB	100MB	BLK001
		100MB	BLK002
		50MB	BLK003

```
hdfs dfs -D dfs.block.size=100M -put sample.txt /
```

IP address:192.168.56.181

# How file is stored in HDFS-blocks



# How file is stored in HDFS-custom config

hadoop installed

```
dfs.blocksize = 64 MB  
Sample.txt 256 MB  
  
hdfs dfs -D dfs.replication=4 -put sample.txt /
```

1 <= dfs.replication <= 512

	DN1	DN2	DN3	DN4
BLK001	BLK001	BLK001	BLK001	BLK001
	BLK002	BLK002	BLK002	BLK002
	BLK003	BLK003	BLK003	BLK003
BLK004	BLK004	BLK004	BLK004	BLK004

IP address:192.168.56.181

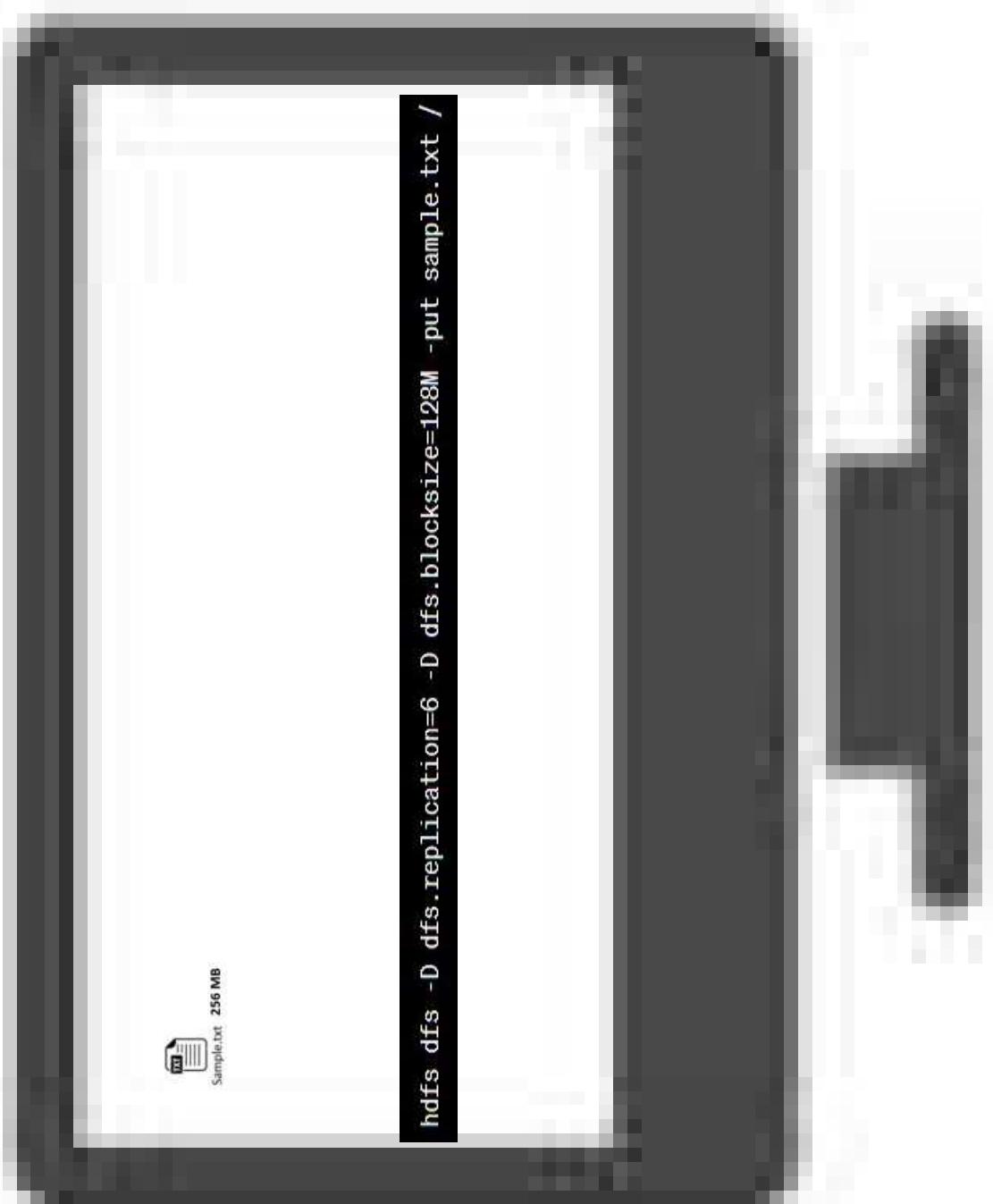
# How file is stored in HDFS-custom config



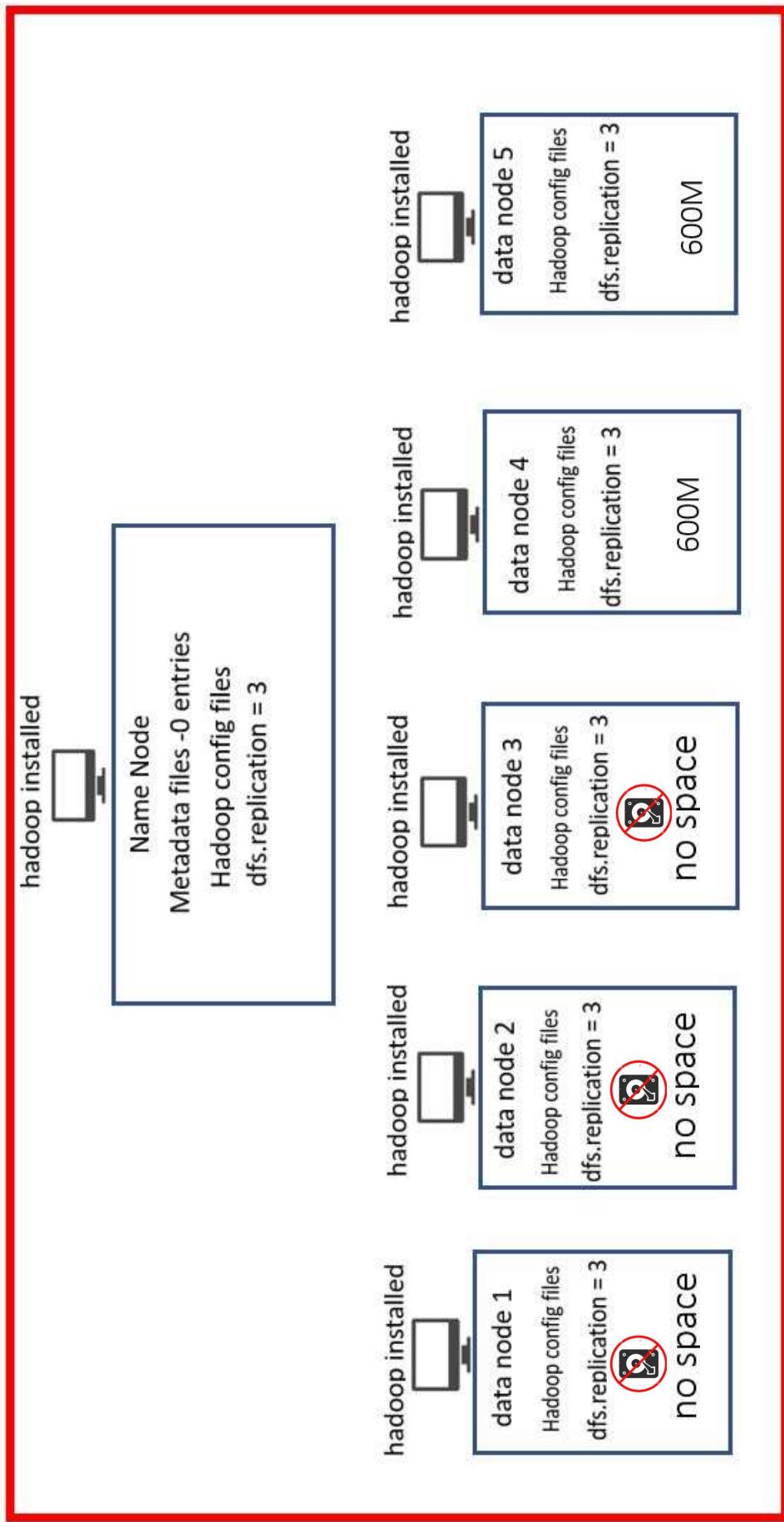
```
Sample.txt 256 MB  
hdfs dfs -D dfs.replication=2 -D dfs.blocksize=300M -put sample.txt /
```

# How file is stored in HDFS-custom config

- HDFS cannot replicate a block more than  $n$  times than there are nodes in the cluster.
- Therefore, in 5-node cluster, the maximum achievable replication factor is 5, not 6.
- It will still create a 5 copies and will show you warning that it is under replicated block
- We can change any replication factor at any time, if we think that its too much not needed and is getting wasted
- We can add new data node in the cluster
- Then automatically 6<sup>th</sup> copy will be created
- It will be updated in name node



# How file is stored in HDFS-custom config-no space in

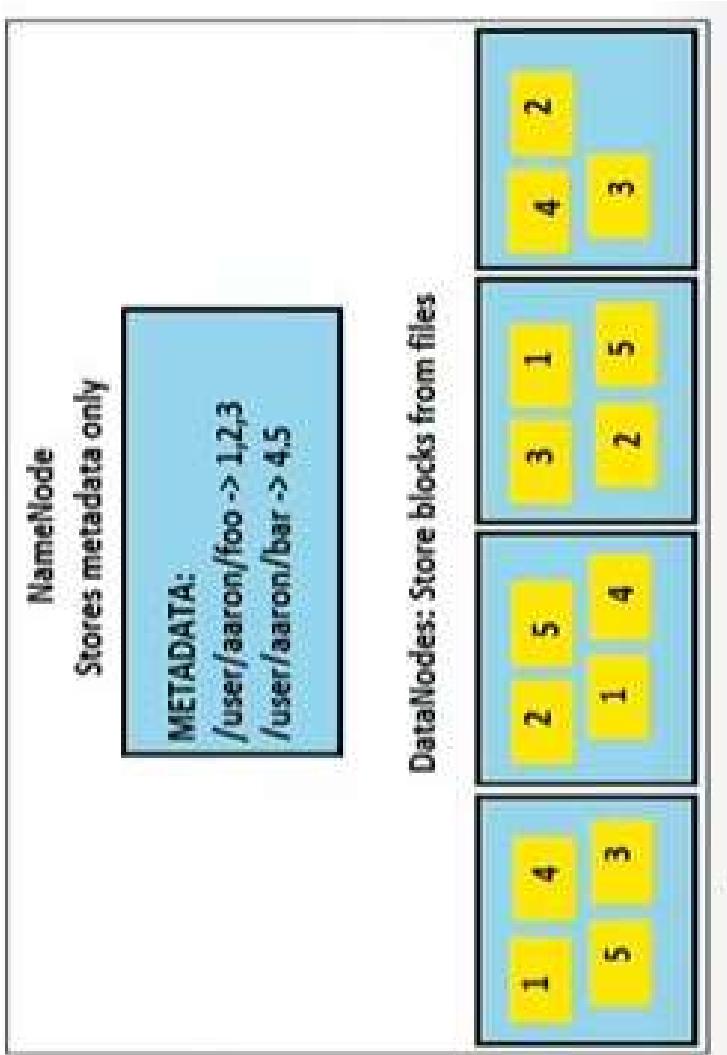


```
hdfs dfs -D dfs.replication=2 -D dfs.blocksize=300M -put sample.txt /
```

It will store 2 copies only later on it will store rest of 4 once new node is added

## HDFS Key Points

- Default Block Size: **64 MB** (Hadoop 1.x)  
**128 MB** (Hadoop 2.x)  
**256 MB** (Hadoop 3.x)
- Every file is split into *Blocks*
- Every Block is *replicated 3 times by default*
- Block is *never replicated on the same Node*
- Files are write-once and read-many
- Data can be appended to a file, but the file's existing contents cannot be changed
- data can be written to a storage medium (like a disk or optical storage) only once, and after that, it can only be read and not modified or deleted.
- Hadoop was meant Initially only for static data
- Example static and dynamic data ( transaction / transaction date ) ( mobile/address )



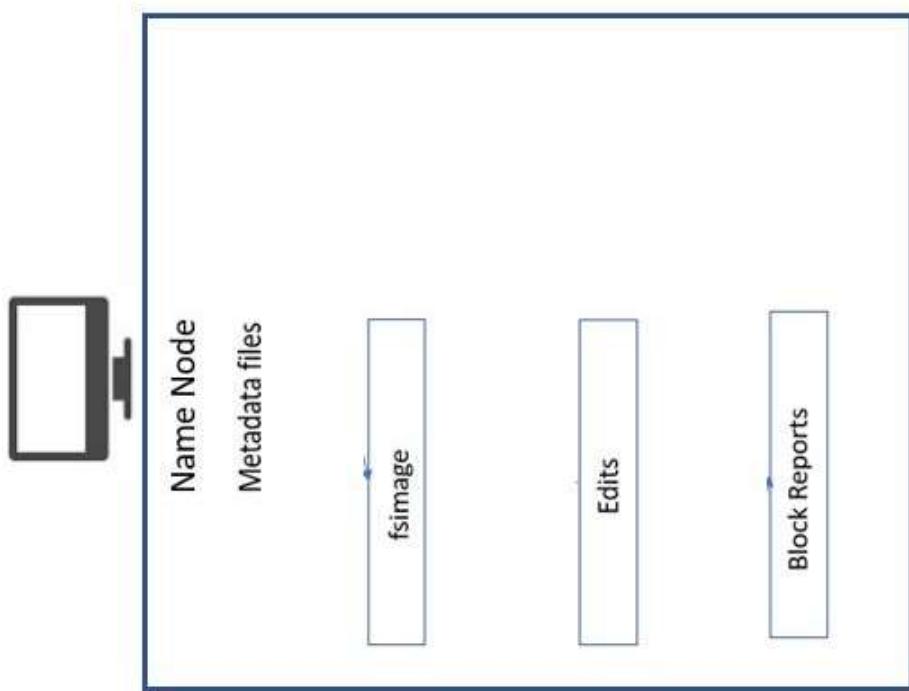
# Name Node

Hadoop Distributed File System's metadata is maintained by NameNode in various files like below

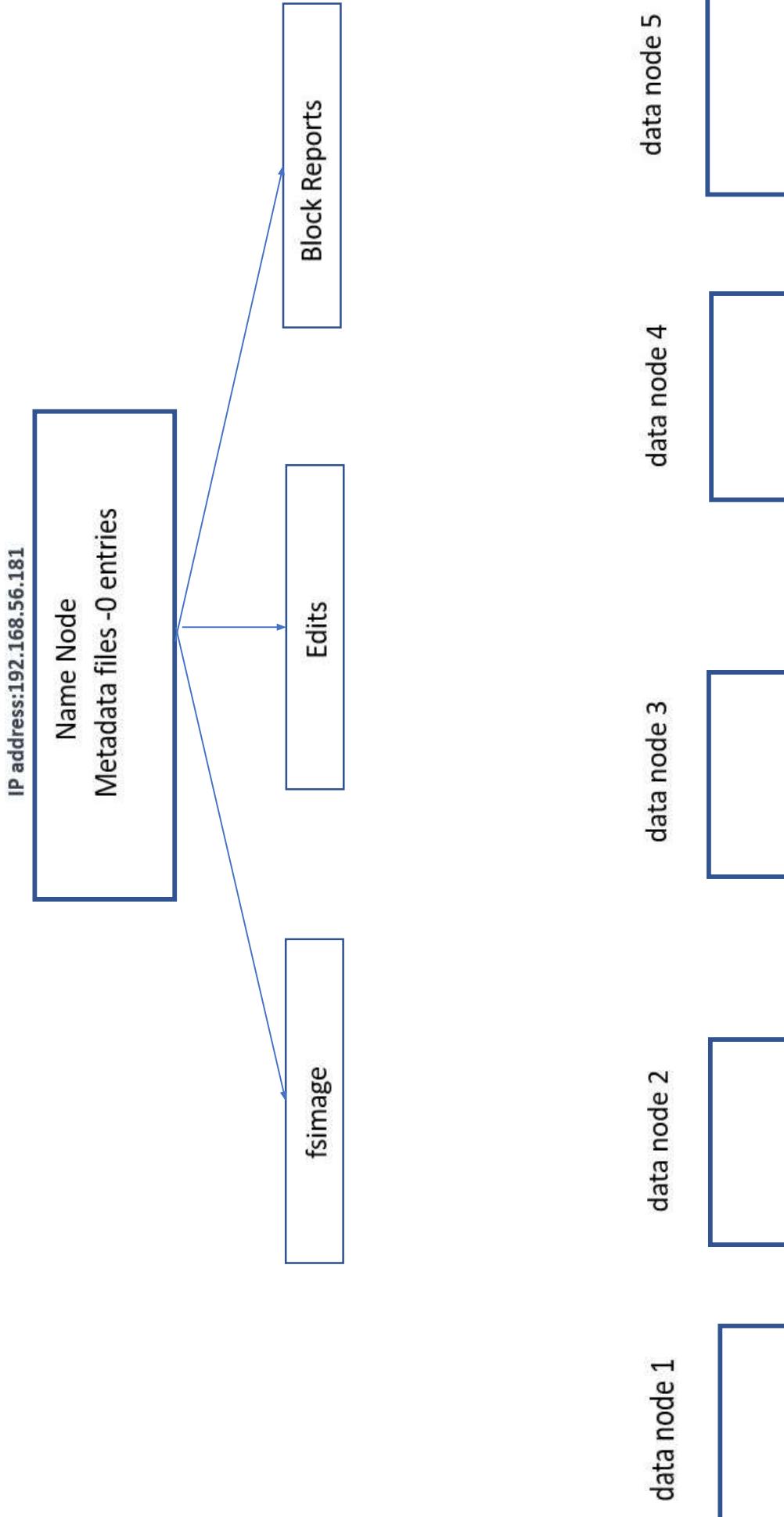
- **Fsimage:** Entire file system's metadata which includes all file properties like File Name, Ownership, Permissions, Timestamps, Size, Replication and List of Blocks is stored in a file called fsimage.
- **Edits:** Rather than modifying the existing fsimage every time for every transaction, NameNode instead records the new transactions (also called as edits) in the edit log for durability. Edit log transactions will be merged with fsimage on frequent intervals to bring fsimage content up-to-date.
- **Block Report / Bitmap:** Whenever a DataNode service is started, it sends the list of blocks it has to NameNode through Heartbeat. NameNode receives Individual Block Report from each DataNode and merge them into a single Consolidated Block Report and maintains it into its own memory. NameNode uses this Consolidated Block Report to understand where the file blocks are actually located while reading the data from HDFS.

NOTE: Along with these metadata files, NameNode also maintains a consolidated DataNode Information file which contains info on available DataNodes, allocated capacity, used & remaining capacity, No. of blocks, etc. from each DataNode

# Name Node



# Name Node



## Command executed

hadoop installed

```
dfs.replication = 3
```

```
hdfs dfs -D dfs.block.size=100M -put sample.txt /
```

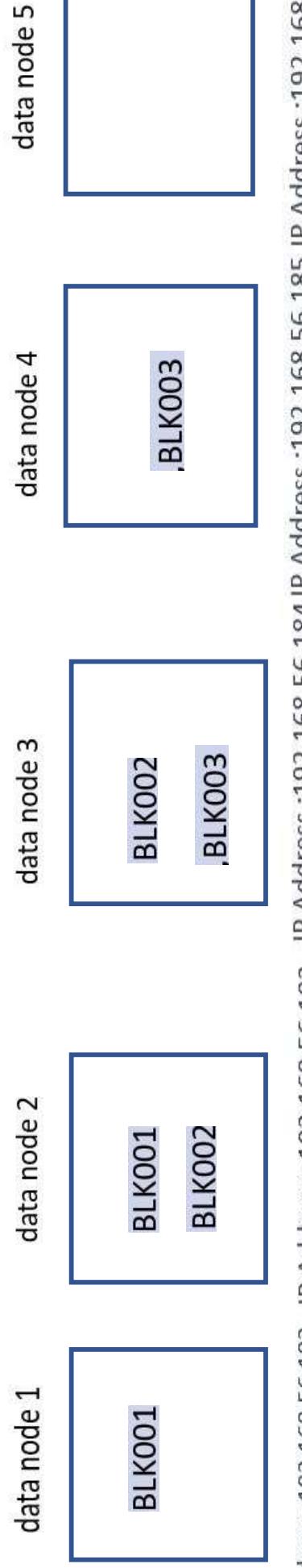
IP address:192.168.56.181

Sample.txt-256 MB	100MB	BLK001
		BLK002
	50MB	BLK003

# How file is stored in HDFS-Fsimage

fsimage

SL no	File Name	Path	Size	Replication	Block size	timestamp	Block numbers
1	Sample1.txt	/.....	256M	2	100M	12/01/2024	BLK001, BLK002, BLK003



IP Address :192.168.56.182 IP Address :192.168.56.183 IP Address :192.168.56.184 IP Address :192.168.56.185 IP Address :192.168.56.186

Command executed



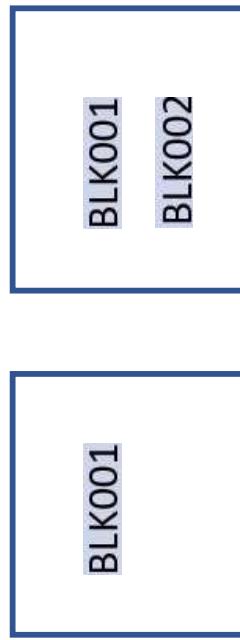
Now fsimage entry is updated hence name node metadata is update

# How file is stored in HDFS-Fsimage

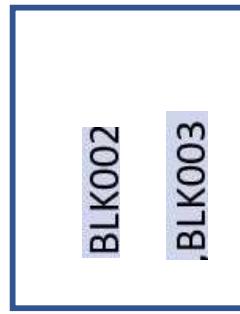
fsimage

SL no	File Name	Path	Size	Replication	Block size	timestamp	Block numbers
1	Sample1.txt	/.....	256M	2	100M	12/01/2024	BLK001, BLK002, BLK003
2	Sample2.txt	/...	20M	1	64M	12/01/2024	BLK004

data node 1



data node 2



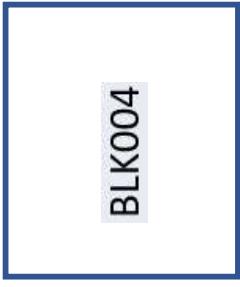
data node 3



data node 4



data node 5



IP Address :192.168.56.182 IP Address :192.168.56.183 IP Address :192.168.56.184 IP Address :192.168.56.185 IP Address :192.168.56.186

# More transactions -Fsimage

fsimage

SL no	File Name	Path	Size	Replication	Block size	timestamp	Block numbers
1	Sample1.txt	/.....	256M	2	100M	12/01/2024	BLK001, BLK002, BLK003
2	Sample2.txt	./...	20M	1	64M	12/01/2024	BLK004
---							
1M	-----	-----	-----	-----	-----	-----	-----

data node 1

BLK001  
BLK002

data node 2

BLK001  
BLK002  
BLK003

data node 3

BLK002  
BLK003

data node 4

BLK003  
BLK004

data node 5

BLK004

IP Address :192.168.56.182 IP Address :192.168.56.183 IP Address :192.168.56.184 IP Address :192.168.56.185 IP Address :192.168.56.186

# More transactions -Fsimage

## fsimage

SL no	File Name	Path	Size	Replication	Block size	timestamp	Block numbers
1	Sample1.txt	/.....	256M	2	100M	12/01/2024	BLK001,BLK002,BLK003
2	Sample2.txt	/...	20M	1	64M	12/01/2024	BLK004
---	---	---	---	---	---	---	---
1M	----	----	----	----	----	----	----

- At the beginning lets say size of fsimage is 2MB
- After 1M transactions it will become approx. 2GB say
- In Hadoop 100s of people will be writing the file in Hadoop
- Name node has to update 100 entries in to fsimage file
- For every file entry name node will open fsimage file update metadata and save it and close it.
- 100s of people parallelly writing the file in Hadoop every time namenode is opening it, also size of file is increasing lets say its now 10GB
- Every time opening such big file and closing it parallelly is very difficult will take time as well there are chances that file will get corrupted. As we are opening and closing so many times
- Entire data will loss if fsimage is gone (persons phonebook(fsimage) is loss then)  
To save this danger we they introduce edits

# To save danger-edits

## Edits

SL no	File Name	Path	Size	Replication	Block size	timestamp	Block numbers
1	Sample1.txt	/.....	256M	2	100M	12/01/2024	BLK001, BLK002, BLK003

- Name node will not write it into fsimage it will write it into edits first
- Every time you write a new file the metadata is written in edit log first

Edits after 1 hour- lets say 100 transactions

SL no	File Name	Path	Size	Replication	Block size	timestamp	Block numbers
1	Sample1.txt	/.....	256M	2	100M	12/01/2024	BLK001, BLK002, BLK003
2	Sample2.txt	/...	20M	1	64M	12/01/2024	BLK004
---	---	---	---	---	---	---	---
---	---	---	---	---	---	---	---
100	---	---	---	---	---	---	---

Will merge to fsimage and it becomes empty



# After one hour

## fsimage

SL no	File Name	Path	Size	Replication	Block size	timestamp	Block numbers
1	Sample1.txt	/.....	256M	2	100M	12/01/2024	BLK001,BLK002,BLK003
2	Sample2.txt	/...	20M	1	64M	12/01/2024	BLK004
---	---	---	---	---	---	---	---
---	---	---	---	---	---	---	---
---	---	---	---	---	---	---	---
100	---	---	---	---	---	---	---

## Edits

SL no	File Name	Path	Size	Replication	Block size	timestamp	Block numbers

- Edit file will not grow and fsimage we don't have to open fsimage often we are opening it once in hr (default time) or we can change the interval or time duration may be in 30 mins etc
- Sometimes if too many people writing file there is a chance that edit file is corrupted but we will loose last 1 hr of data

# Commands

- hadoop fs -ls
- wget [URL]
- ls
- hadoop fs -ls
- **cd /hadoop/hdfs/namenode**
- ls hadoop fs -mkdir practice
- hadoop fs -ls
- mv annual-enterprise-survey-2020-financial-year-provisional-csv.csv sample.csv
- ls
- hadoop fs -copyFromLocal sample.csv practice/sample.csv
- hadoop fs -ls
- hadoop fs -ls practice
- hadoop fs -ls practice
- hadoop fs -rm practice/sample.csv
- hadoop fs -ls practice
- hadoop fs -rmdir practice
- hadoop fs -ls
- hadoop fs
- **cd current**
- **ls**
- sudo find / -name hdfs-site.xml 2>/dev/null
- cat /etc/hadoop/2.6.5.0-292/0/hdfs-site.xml

# Process of writing a file

- Every time you write a file the meta data of the file is recorded by name node in the edit log first.
- If we have written 100 files in one hour, every one hour we are merging the content of edit log to fsimage. ( we can change 1hr time-just change config file)
- Take every thing populated in edit logs and put it into fsimage and reset edit log to empty. ( who does it?)
- Every one hour we are emptying edit log.
- also opening fsimage after 1 hour keeping it safe so no loss of data
- If too many people access the Hadoop edit file will be corrupted but we will not loose the complete data we will loose only one hour of data
- Fsimge is main file to **safeguard fsimage** ---we introduce edits

# Reading a file(searching in edits and fsimage)

fsimage						
Sl. No	FileName	Path	Size	Replication	Block#	Timestamp
1	Sample1.txt	/	256M	2	100M	4/22/2021 BLK0001, BLK0003
2	Sample2.txt	/	20M	1	64MB	4/22/2021 BLK0004

*Hit the name node*

Edits						
Sl. No	FileName	Path	Size	Replication	Block#	Timestamp
3	Sample3.txt	/	100M	2	64MB	4/22/2021 BLK0005, BLK0006

*Hit the name node*

Block Report

Block#	File Name	Size	Replication Factor	Data Node 1 IP	Data Node 1 Block#	Data Node 2 IP	Data Node 2 Block#	Data Node 3 IP	Data Node 3 Block#	Data Node 4 IP	Data Node 4 Block#	Data Node 5 IP	Data Node 5 Block#
BLK001	Sample1.txt	256M	2	192.168.56.182	BLK001	192.168.56.183	BLK001	192.168.56.184	BLK001	192.168.56.185	BLK001	192.168.56.186	BLK001

Name node report



file not found

- Now metadata is available on both the files fsimage and edits
- Name node first search file in edits as it is of less size
- Then it will search in fsimage

## Block Number

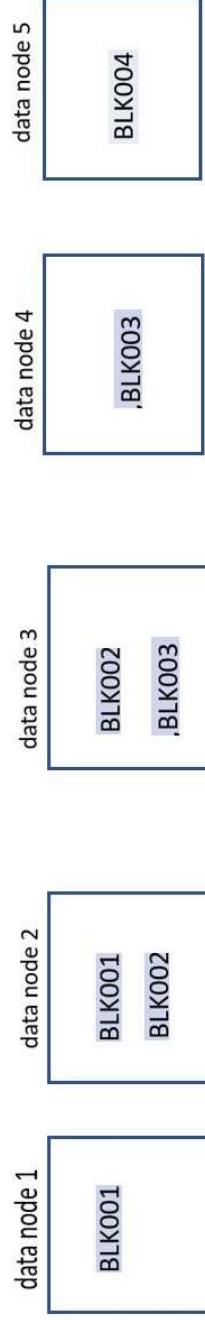
IP address:192.168.56.101  
Name Node  
Metadata files -0 entries

### fsimage

SL no	File Name	Path	Size	Replication	Block size	timestamp	Block numbers
1	Sample1.txt	/.....	256M	2	100M	12/01/2024	BLK001, BLK002, BLK003
2	Sample2.txt	/...	20M	1	64M	12/01/2024	BLK004

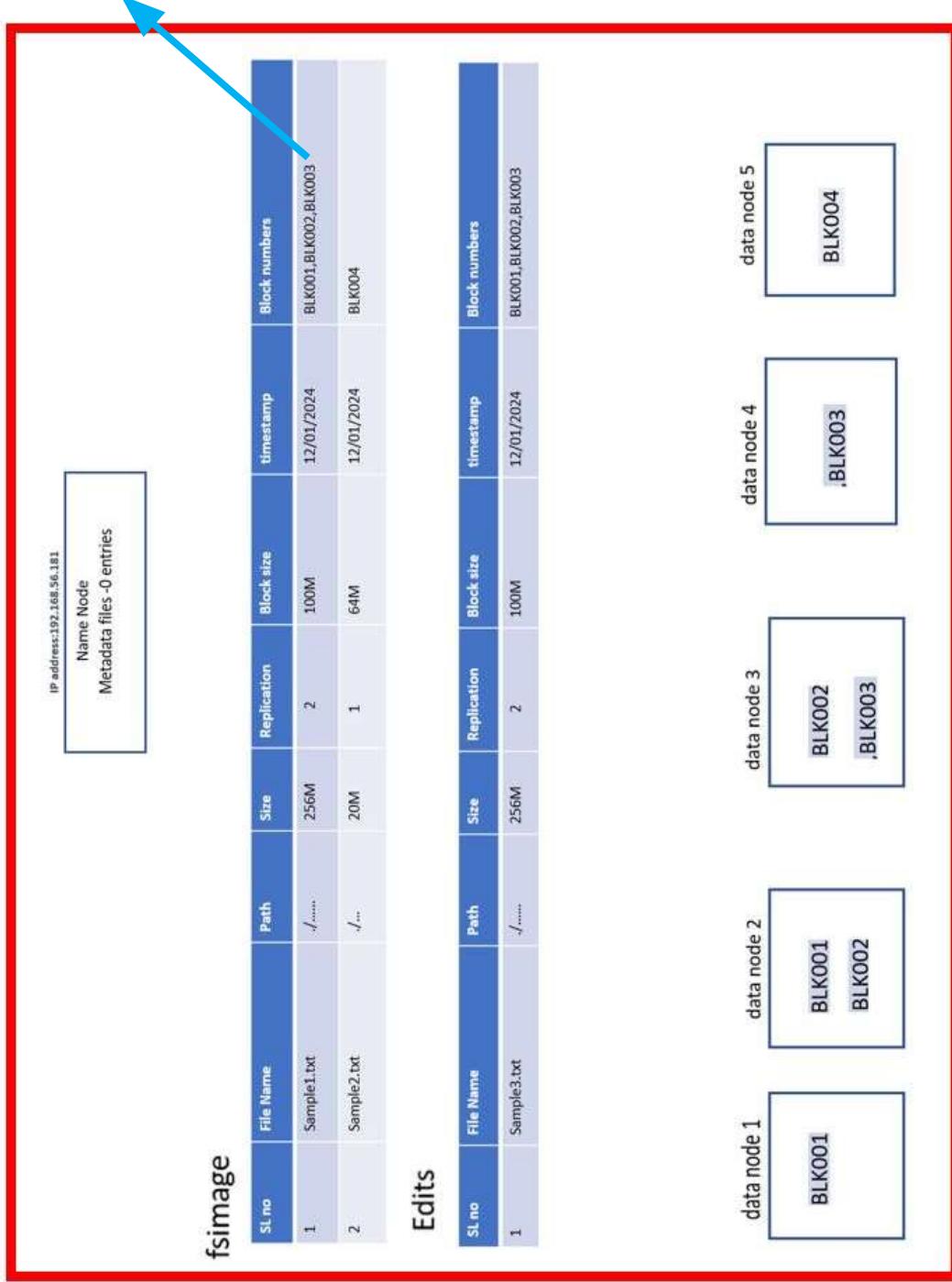
### Edits

SL no	File Name	Path	Size	Replication	Block size	timestamp	Block numbers
1	Sample3.txt	/.....	256M	2	100M	12/01/2024	BLK001, BLK002, BLK003



- We ask for sample 2.txt , four in fsimage we got all metadata
- Including its block number BLK004
- Does it understand where this 004 is available?
- No, it doesn't understand Data node number is not there.
- When it doesn't know data node number then , it has to check the places .
- It is a problem

## Block Number

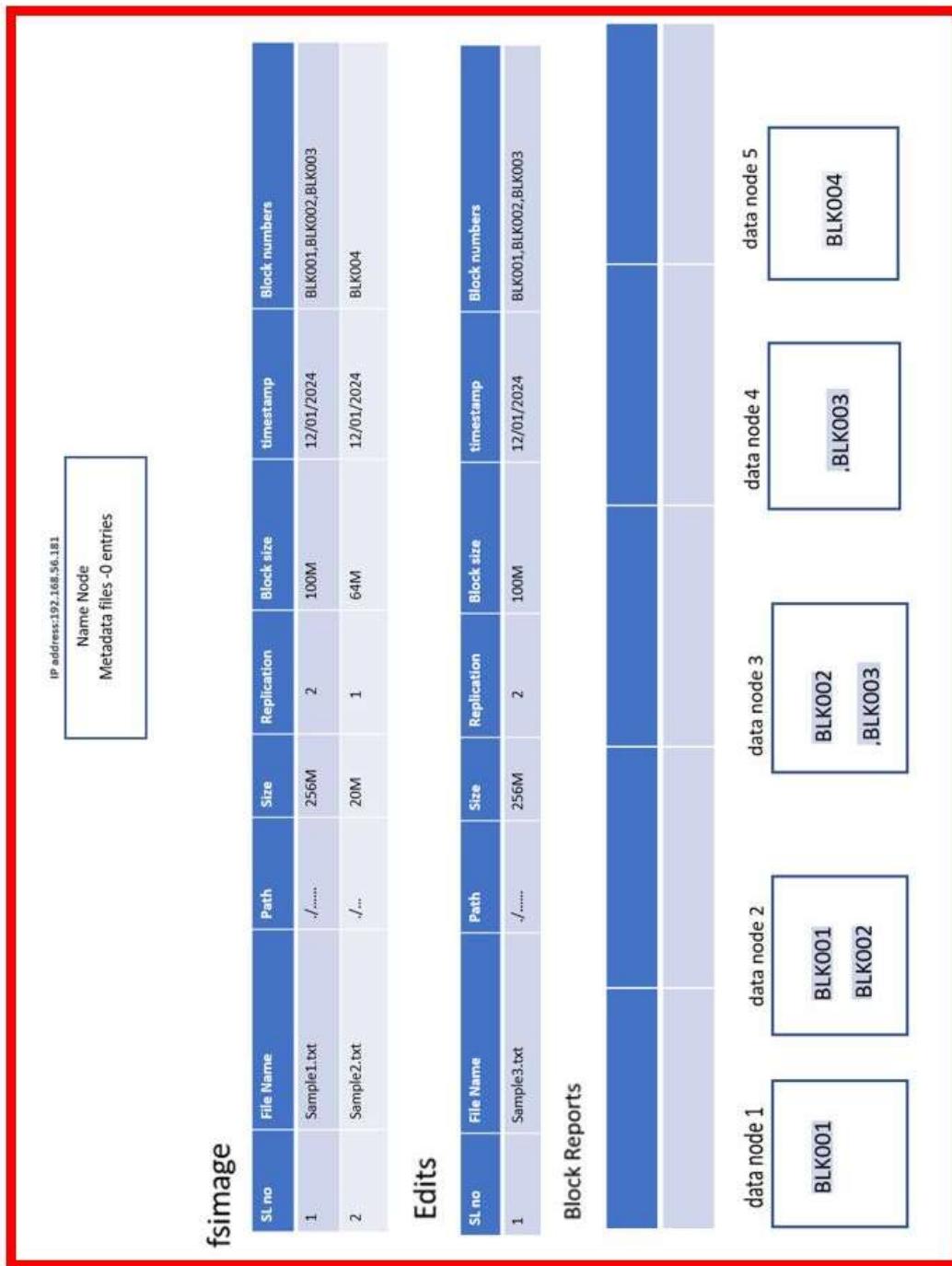


- Can we maintain data node information in fimage or edit log block numbers column(BLK001,etc.)
- **Whenever a new data node is a name node(master) will move some block from existing data node to the added data node, For the purposes. It is called **data node rebalance**.**
- So that New data nodes and old data will have equal load i.e. to make balanced cluster

- Now if suppose we put for (BLK001, DN1)
- Is there any guarantee that one will available in data node one permanently

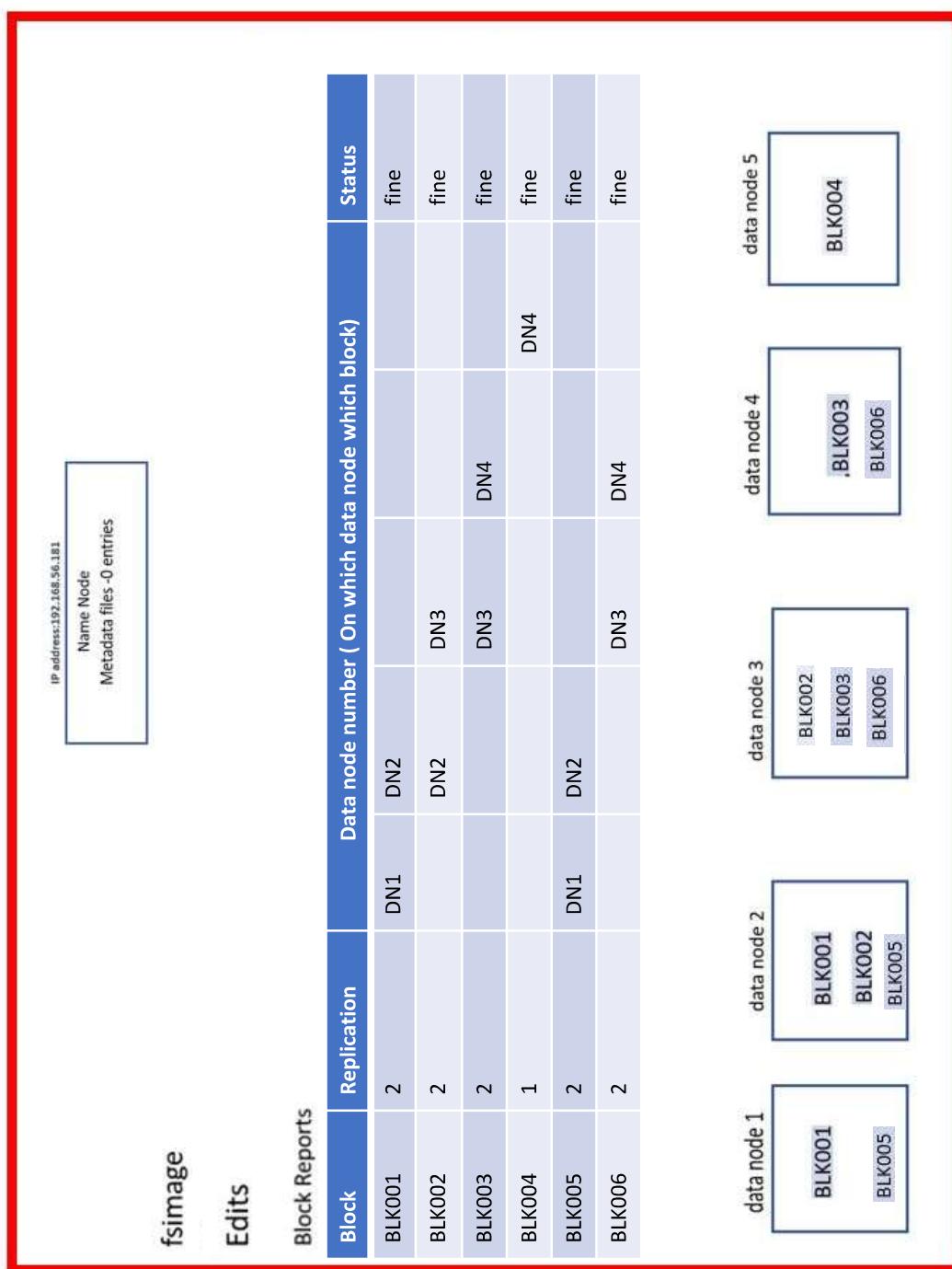
Block Number

- As there is no guarantee that BLK0 be available in datanode1 permane
  - So, we introduce Block report maintain name node 4 blocks



# Block Report

- Name Node builds metadata from Block reports
  - Status of block reports is fine
  - Status may change to uncoupled over replication valid or not valid replication

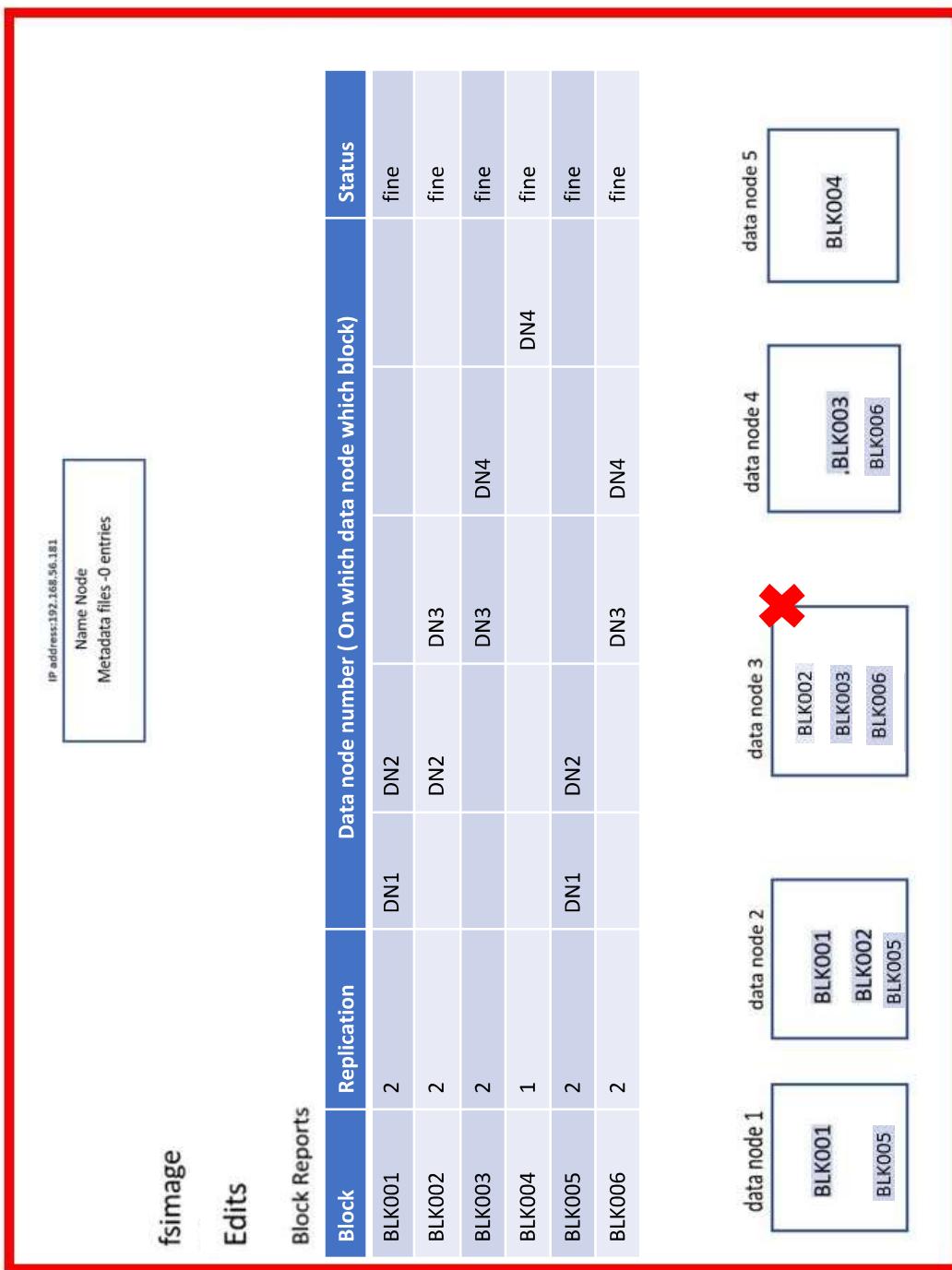


# Block Report

- Suppose some data node is down, how Name node will understand
  - Each data Node sends Heartbeats to name node after every 3 second
  - Heartbeat is proof of a data node is alive and up and running



- If any data node is not sending heart beats it will remove DN3 entries from block report



# Block Report

fsimage  
Edits  
Block Reports

IP address: 192.168.56.181

Name Node

Metadata files -0 entries

Block	Replication	Data node number ( On which data node which block)			Status
BLK001	2	DN1	DN2	✖3	fine
BLK002	2		DN2	✖3	Under replicated
BLK003	2			✖3	Under replicated
BLK004	1				DN5
BLK005	2	DN1	DN2	✖3	fine
BLK006	2			✖3	Under replicated

Diagram illustrating the state of data nodes:

- Data node 1: Contains blocks BLK001, BLK005.
- Data node 2: Contains blocks BLK001, BLK002, BLK005.
- Data node 3: Contains blocks BLK002, BLK003, BLK006. A red X marks the entry for block BLK003.
- Data node 4: Contains blocks BLK003, BLK006.
- Data node 5: Contains block BLK004.

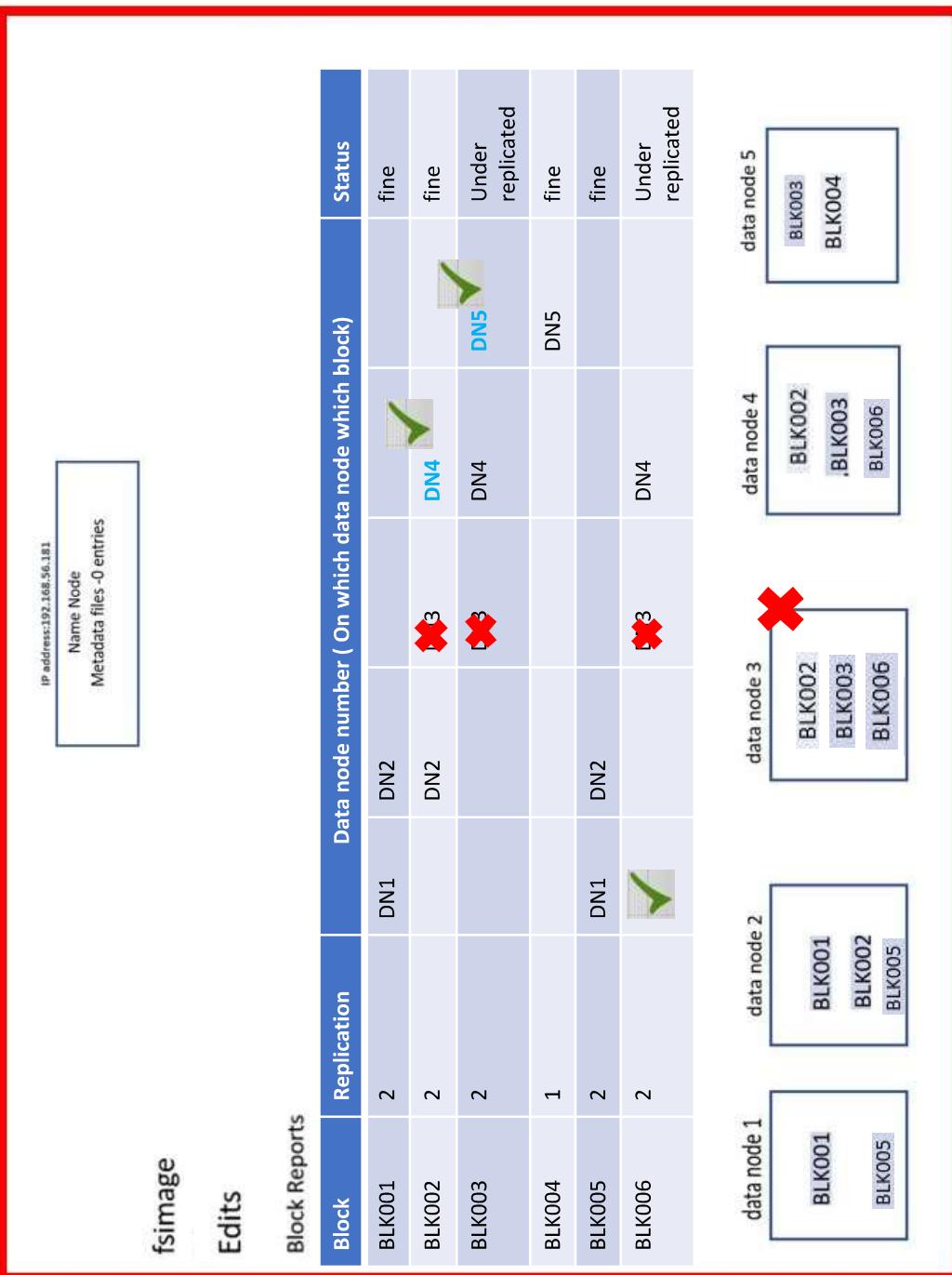
- If any data node is not sending heart beats it will remove DN3 entries from block report. And it will update block report
- Now Block 2 replication factor is 2 and after deletion of DN2 , so status will change to under replicated
- Name node updated block report
- Now Name node will tell data node 2 th you have block 2 can you send it to data node 4
- Now data node 4 will talk to name node that it has block 2,so name node will update the same in block report .
- Extra blocks will result in over replication too
- Always ensure valid replication

Block Report

- Now data node 4 will talk to name node that block 2, so name node will update the same report
  - Block 2 status will update to fine.

- Same about block 3 will get replicated to data

- Same about block 6 will get replicated to data
  - The responsibility of name node that status replication always
  - Name node will update block report status after 3 seconds
  - If data node is not sending any heartbeat to name node, then there is a property that name node will wait to receive a heartbeat from data node minits



# Block Report

fsimage

Edits

Block Reports

IP address: 192.168.56.181

Name Node

Metadata files -0 entries

- every data node is saying that I have these blocks with me.
- if there is any issue with the data node any data node and is not sending heartbeat for to name node.
- what name node will do?
- Name node will delete the entry from block report.
- Even if block report is corrupted it is to reconstruct it as it is saved in memory.

Block	Replication	DN1	DN2	DN3	DN4	DN5	Status
BLK001	2			✗	✓		fine
BLK002	2		DN2	✗	✗	✓	fine
BLK003	2			✗	DN4	DN5	Under replicated
BLK004	1					DN5	fine
BLK005	2	DN1	DN2				fine
BLK006	2	✓		✗	DN4		Under replicated

data node 1

BLK001  
BLK005

data node 2

BLK001  
BLK002  
BLK005

data node 3

BLK002  
BLK003  
BLK006

data node 4

BLK002  
.BLK003  
BLK006

data node 5

BLK003  
BLK004

```
hdfs dfs -cat /sample2.txt
```

User run the command and press

- Reading algorithm will contact name load.
  - Name node will search for sample2.txt in edit log if not found it will search in fsimage
  - Then it will search on which block it is available here it is BLK004
  - Then name node will take information from consolidated block report that where BLK00 is available
  - Block report will tell that BLK00 is available on data node 5
    - name node will return location of block to user data node 5 it available

Address: 192.168.56.181

Name Node  
Metadata files -0 entries

卷之三

fsimage Hard disk

St.no	File Name	Path	Size	Replication	Block size	timestamp	Block numbers
1	Sample1.txt	/.....	256M	2	100M	12/01/2024	BLK001,BLK002,BLK003
2	Sample2.txt	/...	20M	1	64M	12/01/2024	BLK004

Edits Hard disk

Sl.no	File Name	Path	Size	Replication	Block size	timestamp	Block numbers
1	Sample3.txt	/.....	256M	2	100M	12/01/2024	BLK001,BLK002,BLK003

Block Reports memory

Block	Replication	Data node number ( On which data node which block)				Status	
BLK001	2	DN1	DN2			fine	
BLK002	2		DN2	DN3		fine	
BLK003	2			DN3	DN4	fine	
data node 1		data node 2				data node 3	
		BLK001				BLK002	
		BLK002				BLK003	
		BLK003				BLK004	
		BLK004				BLK005	

```
hdfs dfs -cat /sample3.txt
```

Name Node  
Metadata files -0 entries

- User run the command and provide name load.
- Repeating algorithm will contact name load.

SL no	File Name	Path	Size	Replication	Block size	timestamp	Block numbers
1	Sample1.txt	/.....	256M	2	100M	12/01/2024	BLK001,BLK002,BLK003
2	Sample2.txt	/...	20M	1	64M	12/01/2024	BLK004

#### Edits Hard disk

SL no	File Name	Path	Size	Replication	Block size	timestamp	Block numbers
1	Sample3.txt	/.....	256M	2	100M	12/01/2024	BLK001,BLK002,BLK003

#### Block Reports Memory

Block	Replication	Data node number { On which data node which block}				Status
BLK001	2	DN1	DN2			fine
BLK002	2		DN2	DN3		fine
BLK003	2			DN3	DN4	fine

data node 1

BLK001

BLK002

BLK003

data node 2

BLK002

BLK003

data node 3

BLK003

data node 4

BLK004

data node 5

BLK005

#### fsimage Hard disk

SL no	File Name	Path	Size	Replication	Block size	timestamp	Block numbers
1	Sample1.txt	/.....	256M	2	100M	12/01/2024	BLK001,BLK002,BLK003

SL no	File Name	Path	Size	Replication	Block size	timestamp	Block numbers
1	Sample3.txt	/.....	256M	2	100M	12/01/2024	BLK001,BLK002,BLK003

#### Block Reports Memory

Block	Replication	Data node number { On which data node which block}				Status
BLK001	2	DN1	DN2			fine
BLK002	2		DN2	DN3		fine
BLK003	2			DN3	DN4	fine

data node 1

BLK001

BLK002

BLK003

data node 2

BLK002

BLK003

data node 3

BLK003

data node 4

BLK004

data node 5

BLK005

Data Aggregation from various blocks

IP address:192.168.56.181

Name Node  
Metadata files -0 entries

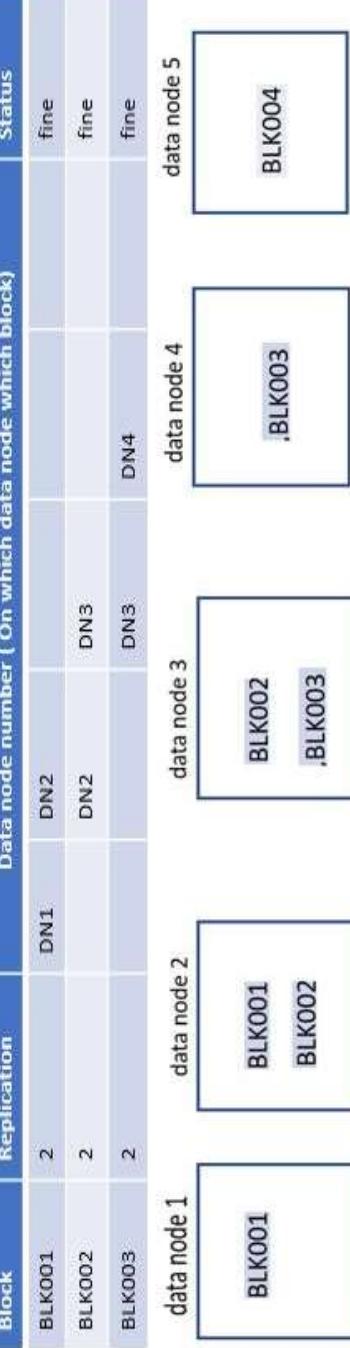
**fsmimage** Hard disk

SL no	File Name	Path	Size	Replication	Block size	timestamp	Block numbers
1	Sample1.txt	/.....	256M	2	100M	12/01/2024	BLK001,BLK002,BLK003
2	Sample2.txt	/...	20M	1	64M	12/01/2024	BLK004

Edits Hard disk

Sl.no	File Name	Path	Size	Replication	Block size	timestamp	Block numbers
1	Sample3.txt	/.....	256M	2	100M	12/01/2024	BLK001,BLK002,BLK003

Block Reports Memory



Merge

## Output



Sample2.txt

- Block report will tell that BLK001 BLK002, BLK003 are available on data node 1 data node 2 and data node 3 data node4

# Data node information Report

data node info report (in memory)						
Host Name	IP Address	Total space	Used space	Available space	No of blocks	Last heartbeat
DN1	192.168.56.182	10GB	184 M	9.9G	3	2% 3sec ago
DN2	192.168.56.183	20GB	284 M	19.98G	4	1.50% 3sec ago
DN3	192.168.56.184	300GB	156 M	299.99G	3	1% 15 sec ago
DN4	192.168.56.185	500GB	112 M	499.9G	3	0.50% 3 sec ago
DN5	192.168.56.186	1TB	96M	0.0000T	3	0.30% 3 sec

192.168.56.182  
DN1  
10GB  
BLK001(100M)  
BLK002(100M)  
BLK005(64M)  
BLK007(20M)

192.168.56.183  
DN2  
20GB  
BLK001(100M)  
BLK002(100M)  
BLK005(64M)  
BLK007(20M)

192.168.56.185  
DN3  
300GB  
BLK002(100M)  
BLK006(36M)  
BLK007(20M)

192.168.56.185  
DN4  
500GB  
BLK003(56M)  
BLK006(36M)  
BLK007(20M)

192.168.56.185  
DNS  
1 TB  
BLK003(56M)  
BLK004(20M)  
BLK007(20M)

After 630 sec it will remove that DN

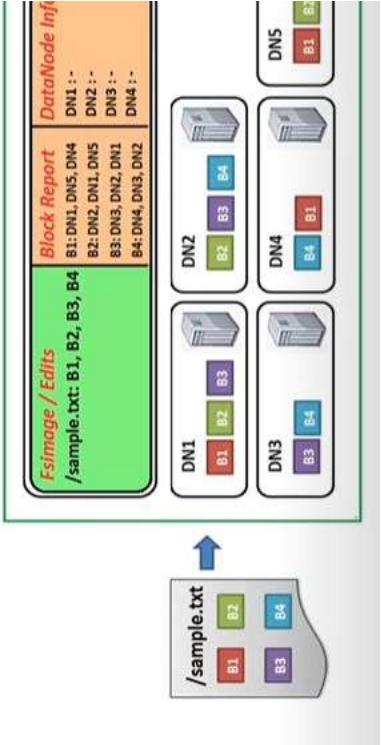
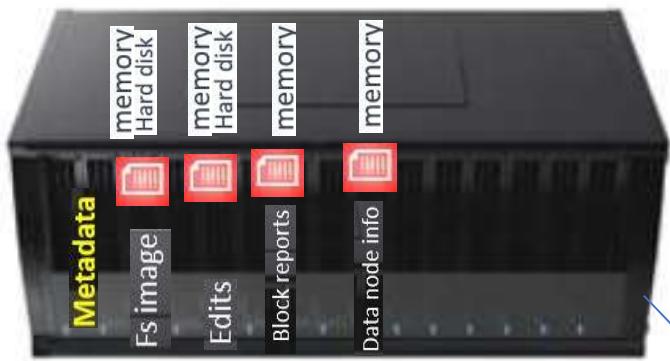
fsimage

Edits

Block Reports

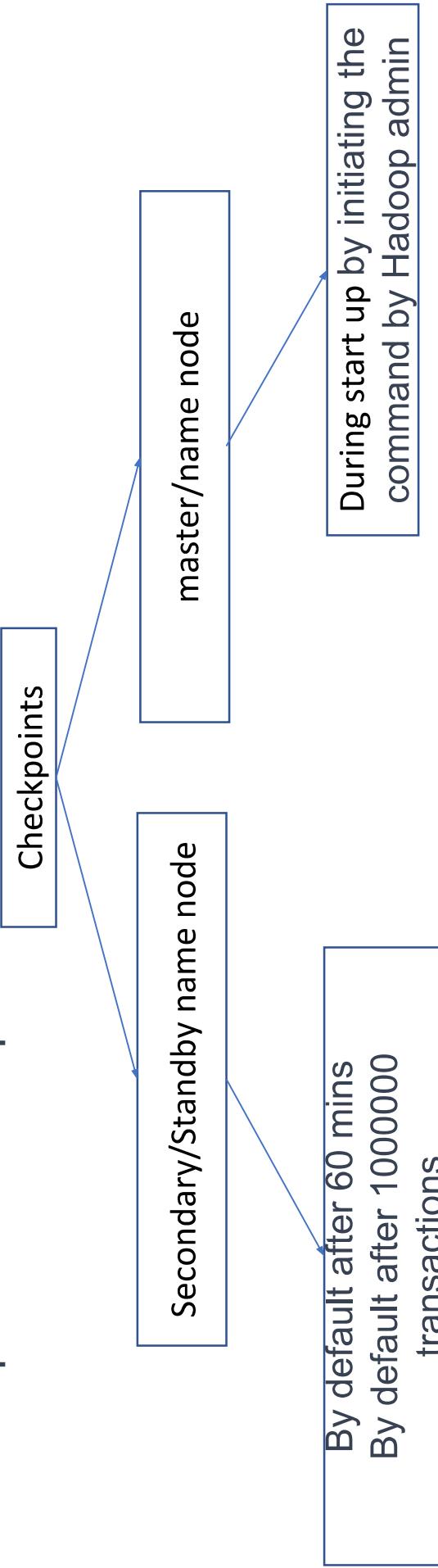
Metadata files 4 entires  
Name Node  
IP address:192.168.56.181

# Setup



# What is Checkpoint in Hadoop?

- A process that takes fsimage & edits and combines them into a new fsimage
- **Secondary NameNode or Standby NameNode** is responsible for doing Checkpoint in Hadoop Cluster.



But this operation will be performed by NameNode and NameNode should in Safemode (read-only mode) during this operation.

```
echo '<property>' >> /path/to/hdfs-site.xml
echo '  <name>dfs.namenode.checkpoint.period</name>' >> /path/to/hdfs-site.xml
echo '    <value>60</value>' >> /path/to/hdfs-site.xml
echo '</property>' >> /path/to/hdfs-site.xml

echo '<property>' >> /path/to/hdfs-site.xml
echo '  <name>dfs.namenode.checkpoint.txns</name>' >> /path/to/hdfs-site.xml
echo '    <value>1000000</value>' >> /path/to/hdfs-site.xml
echo '</property>' >> /path/to/hdfs-site.xml
```

# What is Checkpoint in Hadoop?

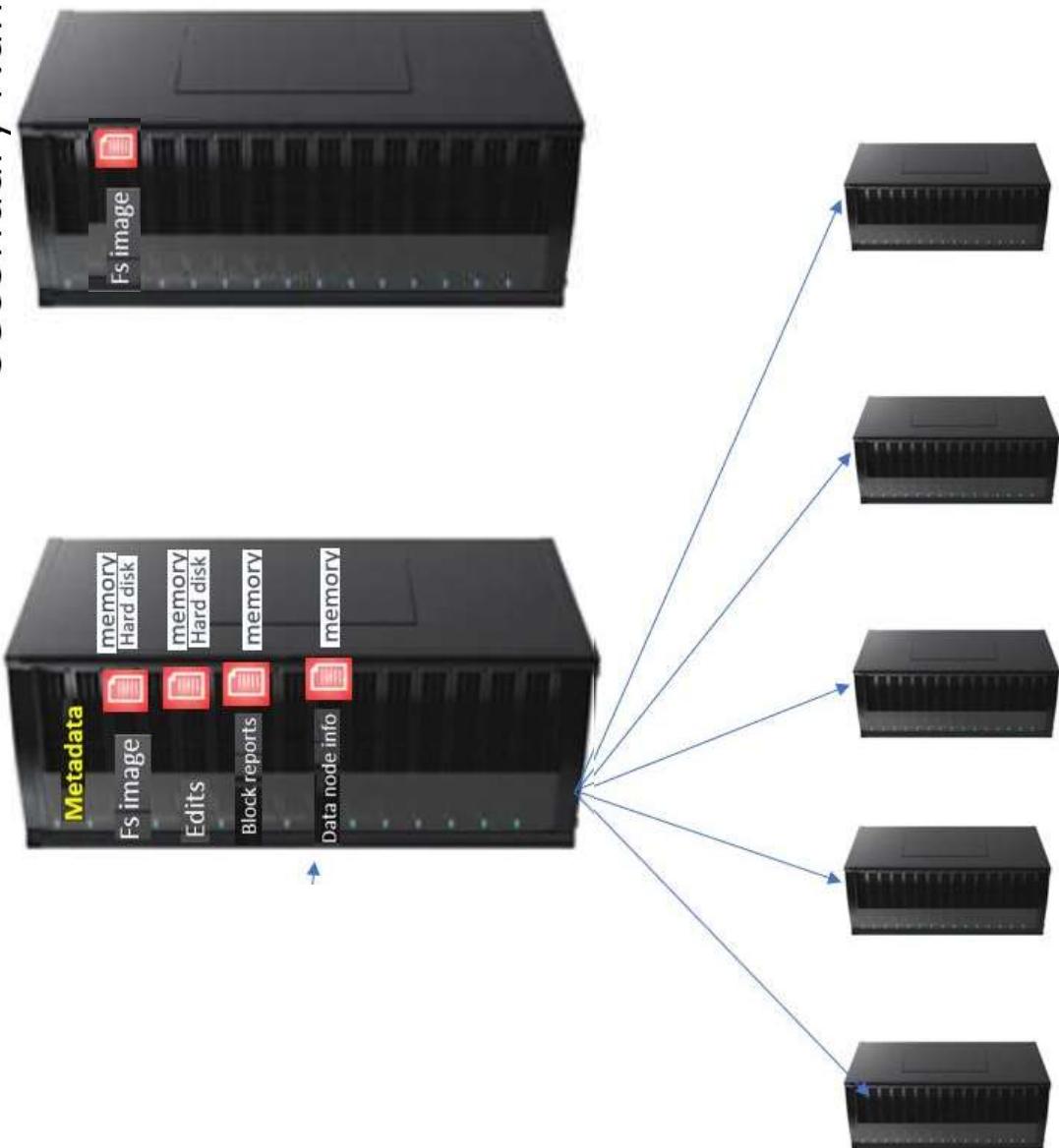
- A process that takes fsimage & edits and combines them into a new fsimage
- **Secondary NameNode or Standby NameNode** is responsible for doing Checkpoint in Hadoop Cluster.



We know after every 1 hour edit logs get merge with fsimage to form new fsimage called checkpointing but now who will do this checkpointing process' **This will be taken care by secondary/standby name node**

# Secondary Name node check point

Primary name node      Secondary Name



Secondary name node continuously  
edit logs file and merging it to fsimage  
hr or after 1M transactions happened

Time	Edits	Fs
	10	0
8.00 AM	0	10
Checkpoint	5	
	0	15
9.00 AM	0	
	50	
10.00 AM	0	6t
10:30 AM	1Million	transaction
		came suddenly
Checkpoint	0	1t
Checkpoint	11:30	

Default settings we can cha

# Checkpoint by name node

	Time	Edits	Fsimage
Checkpoint	8.00 AM	0	10
		5	
	9.00 AM	0	15
Checkpoint		50	
	10.00 AM	0	65
Checkpoint	10:30 AM	11Million transaction came suddenly	
	10:30	0	1M+65
Checkpoint	11:30		
	12:15	1M	
	12:15PM	0	2M+65
Checkpoint	12:45PM	10	
	1:00 AM		

Checkpoint by name node

Entire cluster is stopped ( Stop name node)

Name node start-requirement is 0 edits but

## Checkpoint by Hadoop admin

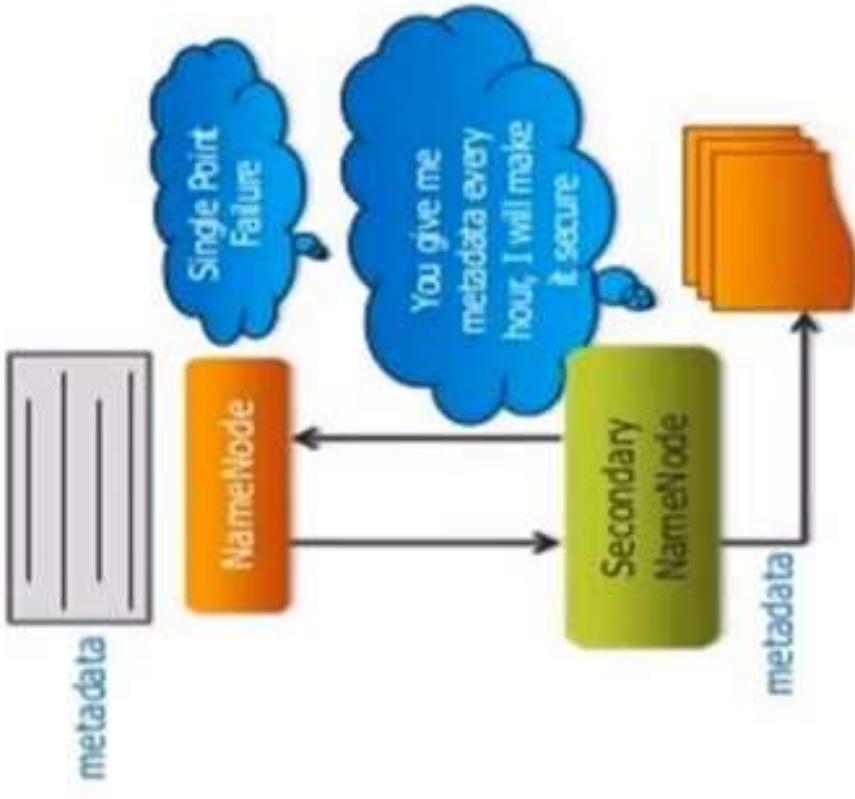
Time	Edits	Fsimage
Checkpoint	10	0
	0	10
	5	
Checkpoint	0	15
	50	
	0	65
Checkpoint	10:30 AM	1Million transaction came suddenly
	0	
	0	1M+65
Checkpoint	11:30	
	1M	
	0	2M+65
Checkpoint	12:15PM	
	10	
	0	2M+75
Checkpoint by name node	1:00 AM	
	0	
	10	
Checkpoint by Hadoop admin	1:45 AM	
	0	
	2M+85	Forcefully by giving command-Name node

# Secondary name node

- Never acts like Primary NameNode in case of Primary NameNode failure

- Retrieve fsimage and edits from Primary NameNode, perform Checkpoint on regular intervals and push the latest copy of fsimage back to Primary NameNode

- Maintains a copy of fsimage as backup of Primary NameNode's metadata



# Edge Node

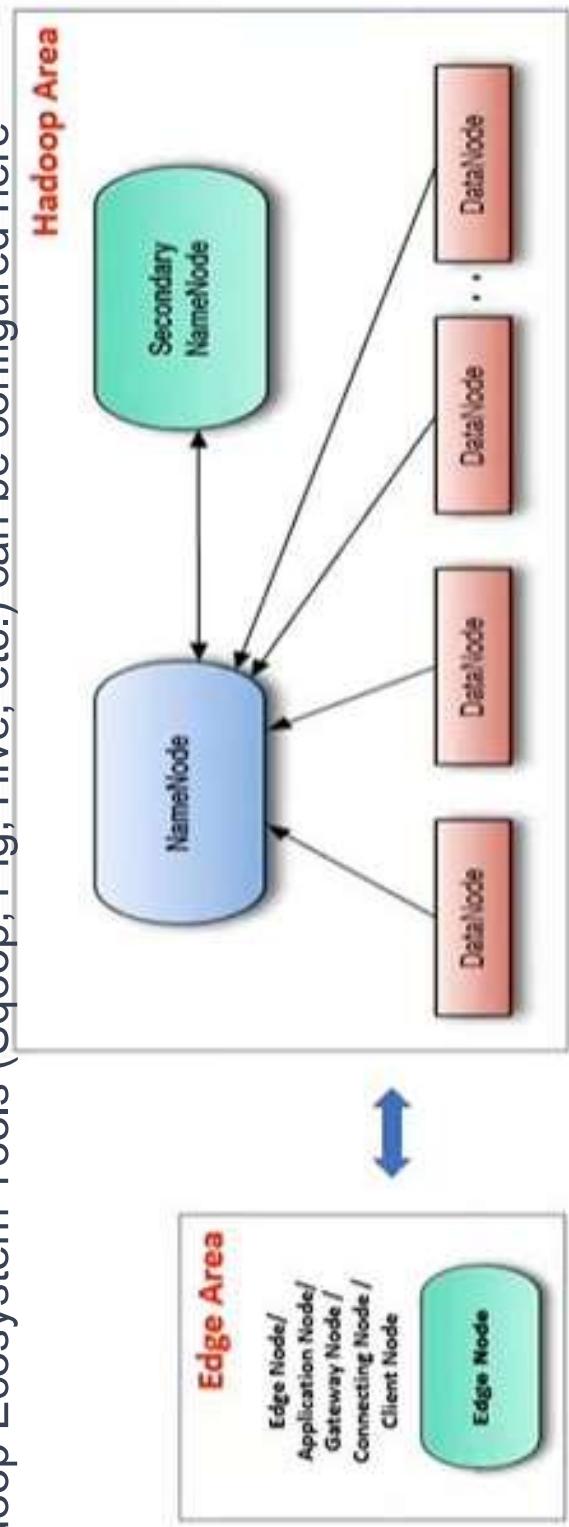
## What is Edge Node?

What is Edge Node?

- Can be referred as Application Node / Gateway Node / Connecting Node / Client Node
- Edge Node is the interface between Hadoop Cluster and Users/Client Applications
- Users/Developers will be given access to Edge Node to execute their Hadoop commands/jobs

Hadoop is installed here but no Hadoop daemons will be running

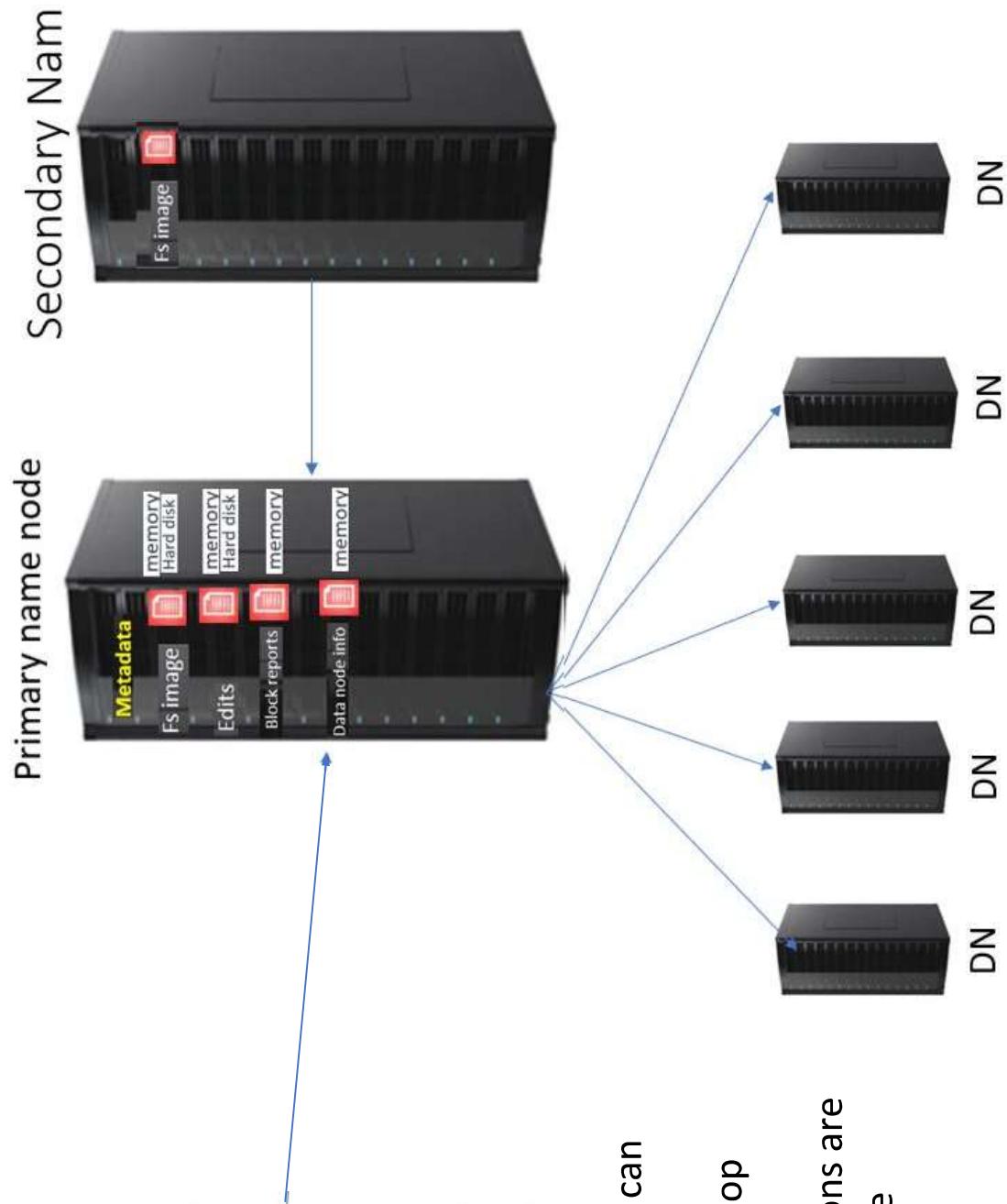
- Often used as staging area for the data being transferred into Hadoop Cluster
- Hadoop Ecosystem Tools (Sqoop, Pig, Hive, etc.) can be configured here



## Edge Node



## Primary name node



- Hadoop client software installed- can type Hadoop commands
- Interface between user and Hadoop cluster
- Hadoop is installed but no daemons are running no services like data node name node at all

# Installation

## Installation Modes of Hadoop



### Standalone Mode

All Hadoop services run in a single Java Virtual Machine (JVM) on a single machine.

### Pseudo distributed Mode

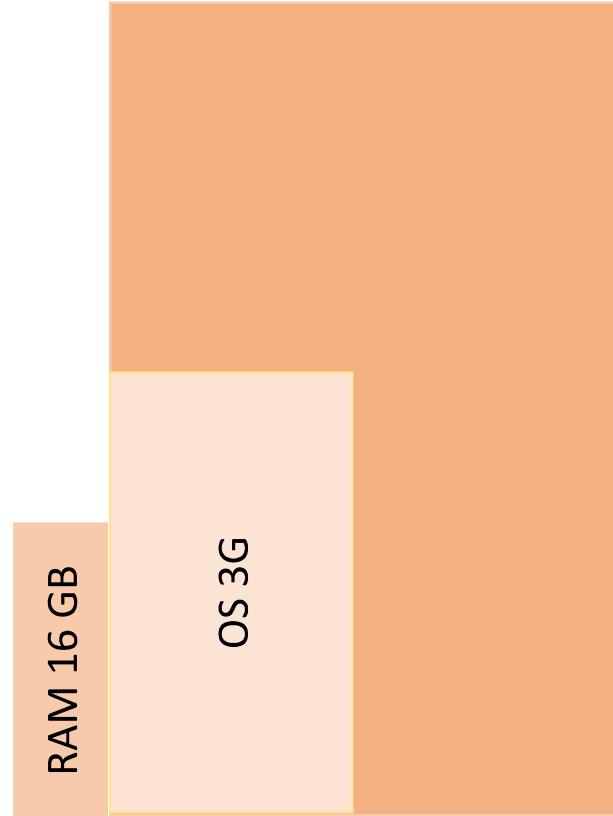
Individual Hadoop services run in an individual JVM but on a single machine.

### Fully distributed Mode

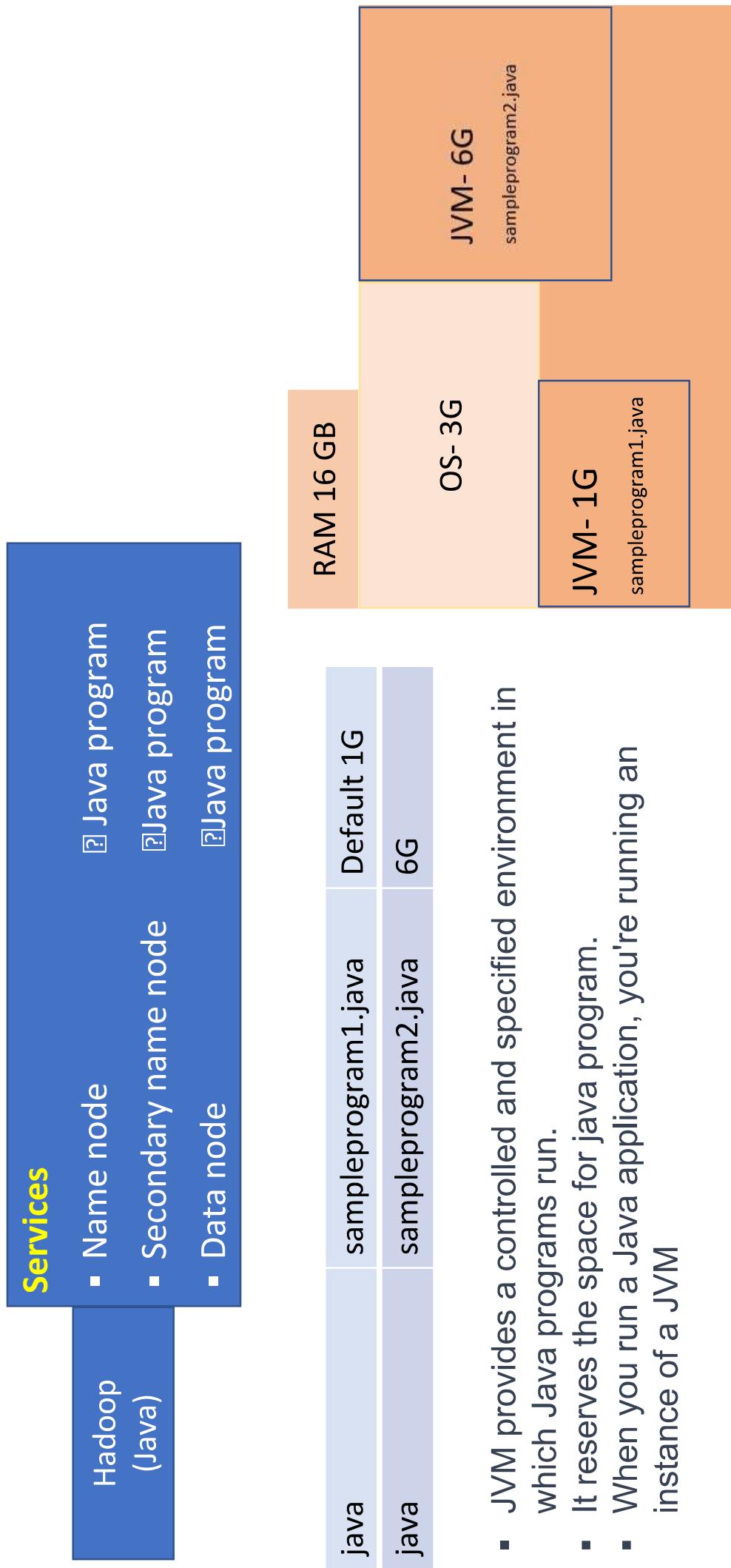
Hadoop services run in different JVMs but in a cluster

# Understanding Basics

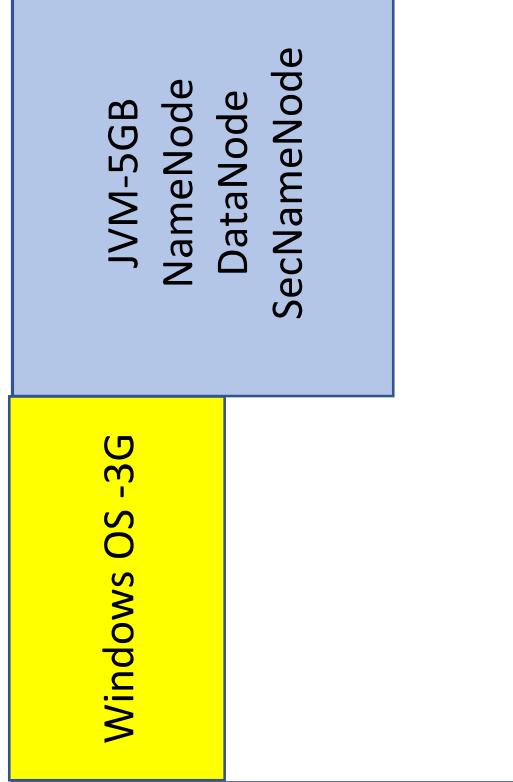
Services		
Hadoop (Java)	▪ Name node ▪ Secondary name node ▪ Data node	❑ Java program ❑ Java program ❑ Java program
c/c++	sampleprogram1.c	3G
c/c++	sampleprogram2.c	6G
c/c++	Sampleprogram3.c	10G



# Understanding Basics-JVM

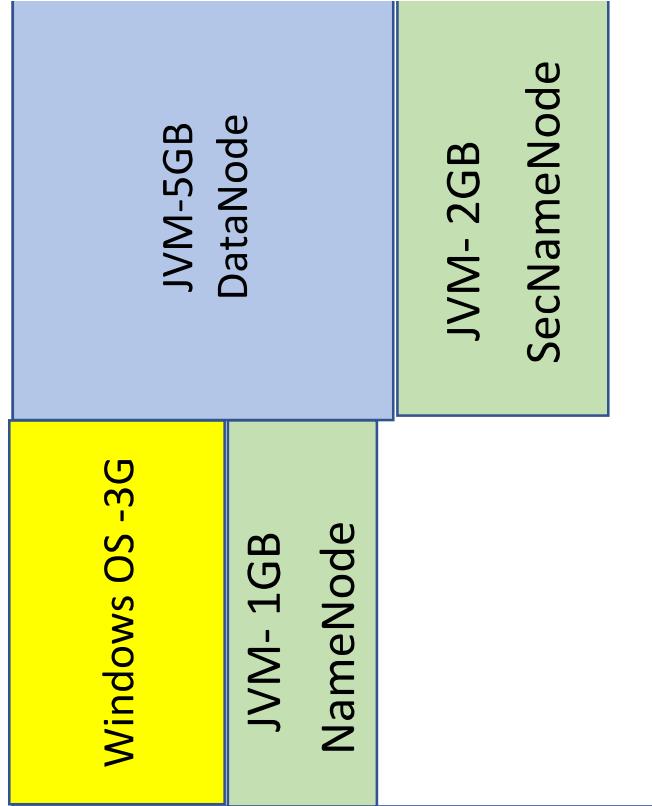


# Standalone



- In a standalone mode of Hadoop, all the components run as a single Java process within a single JVM on one machine

# Pseudo-distributed Mode

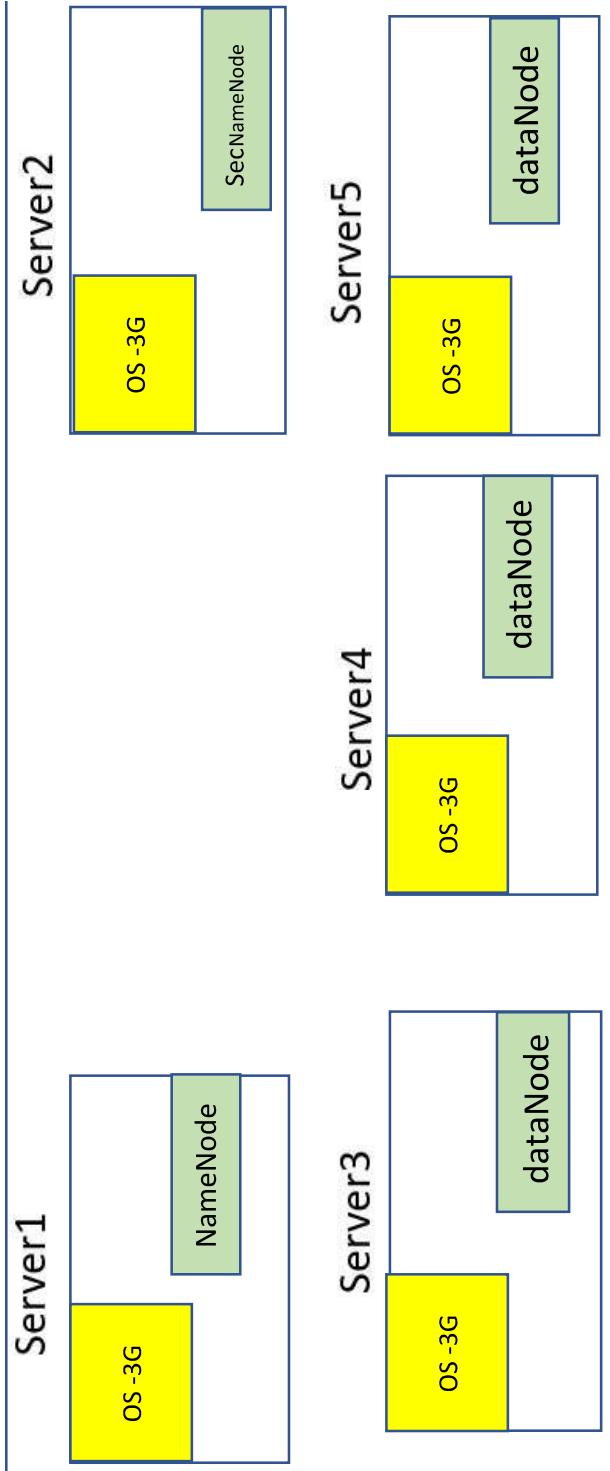


Services	
▪ Name node	❑ Java program
▪ Secondary name node	❑ Java program
▪ Data node	❑ Java program

- In pseudo-distributed mode, each Hadoop daemon (NameNode, Secondary NameNode, and DataNode) runs in its own separate JVM on a single machine.
- This mode is a simulation of the distributed environment, as it acts if having multiple nodes, but it's all localized within one physical or virtual machine

# Fully-distributed Mode

Services
▪ Name node
▪ Secondary name node
▪ Data node



- In a fully distributed mode, Hadoop runs on a cluster of machines, with each component of the Hadoop ecosystem running on its own set of dedicated hardware: (server)

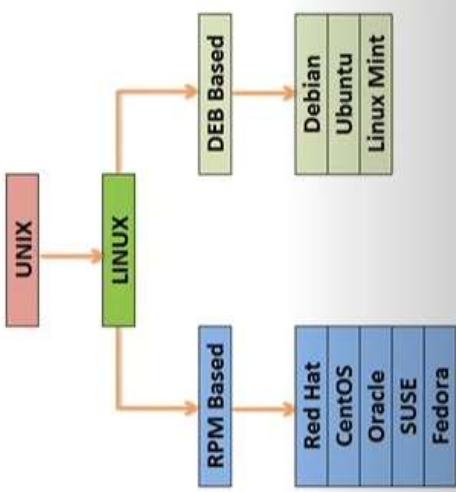
# Operating system

## Operating systems

Windows	GUI available	Command Prompt	licensed
Android	GUI available	Command Prompt	licensed
Macos	GUI available	Command Prompt	licensed
Unix		Command Prompt	Open source

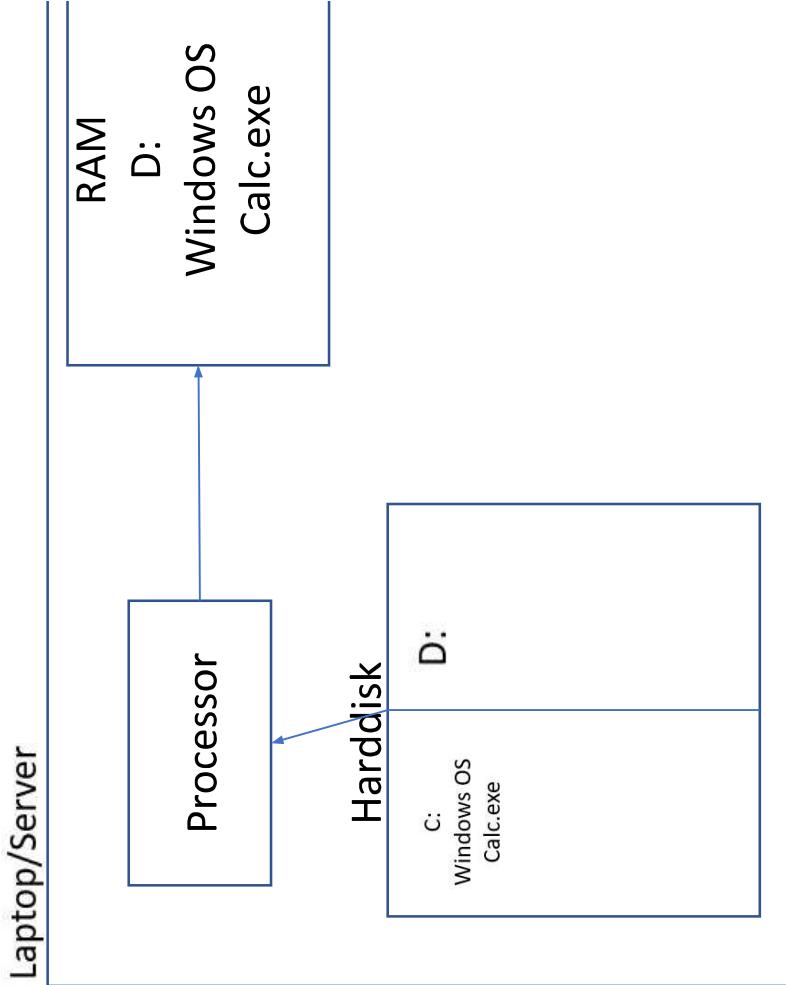
## Linux OS Flavors

RPM (Red Hat Package Manager) and DEB (Debian Package) are two types of package management systems used by different Linux distributions to install, update, and remove software.



# Processor

Pentium4  
Core2duo  
Corei3  
Corei5  
Corei7  
coreo9



Double click calc.exe operating system loads calc.exe to RAM with the help of processor you use the calculator (e.g., entering numbers, performing calculations), these inputs are processed by the processor

# Processor

Pentium 4 1 Physical Core Processor  
Core2Duo 2 Physical Core Processor  
Core i3 2 Physical Core Processor (HTE)

Core i5 4 Physical Core Processor

Core i7 4 Physical Core Processor (HTE)

Core i9 8 Physical Core Processor

PCP

PCP	LCP
6 PCP (HTE)	12 LCP
8 PCP	16 LCP
8 PCP (HTE)	16 LCP
12 PCP	24 LCP
12 PCP (HTE)	24 LCP
16 PCP	32 LCP
16 PCP (HTE)	32 LCP
24 PCP	48 LCP
24 PCP (HTE)	48 LCP
32 PCP	64 LCP
32 PCP (HTE)	64 LCP
40 PCP	80 LCP
40 PCP (HTE)	80 LCP
60 PCP	
80 PCP	
100 PCP	

- **Physical Core Processor (PCP):** This is the actual physical core inside a CPU. A physical core can independently execute computer instructions.
- **Logical Core Processor (LCP):** This typically refers to logical cores created through a technology like Intel's Hyper-Threading.

## IRCTC Server 400 tickets/sec

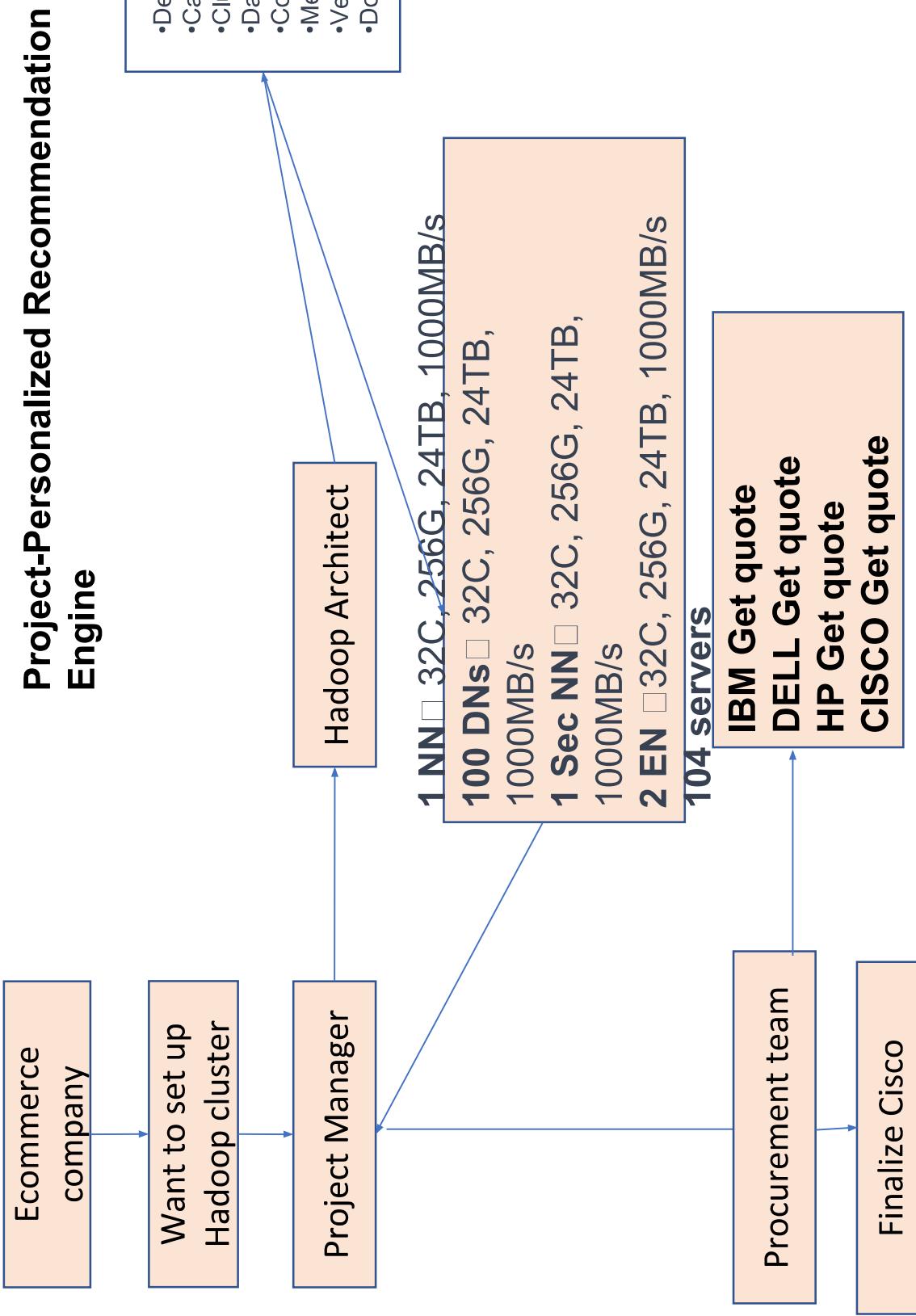
IRCTC has recently added two new high-capacity servers bringing their total server count to a higher number to handle increased demand. These new servers have improved the ticket booking capacity significantly.

**2000 tickets/sec**

# Laptop

Laptop	Total	OS	Free	VM1	VM2	VM3	VM4	VM5	VM6	Free_2
Cores	16	1	15	2	2	2	2	2	2	2
RAM (GB)	64	3	61	10	10	10	10	10	2	2
HDD (GB)	900	300	600	30	30	30	30	30	30	30
Network (MB/s)	1000	100	900	50	50	50	50	50	50	50
OS	Windows			Linux						
Hadoop	NN	SNN	DN1	DN2	DN3	EN				

# Real time scenario



- Designing and Architecting
- Capacity Planning
- Cluster Configuration
- Data Governance and Security
- Collaboration with Teams
- Mentoring
- Vendor Interaction
- Documentation

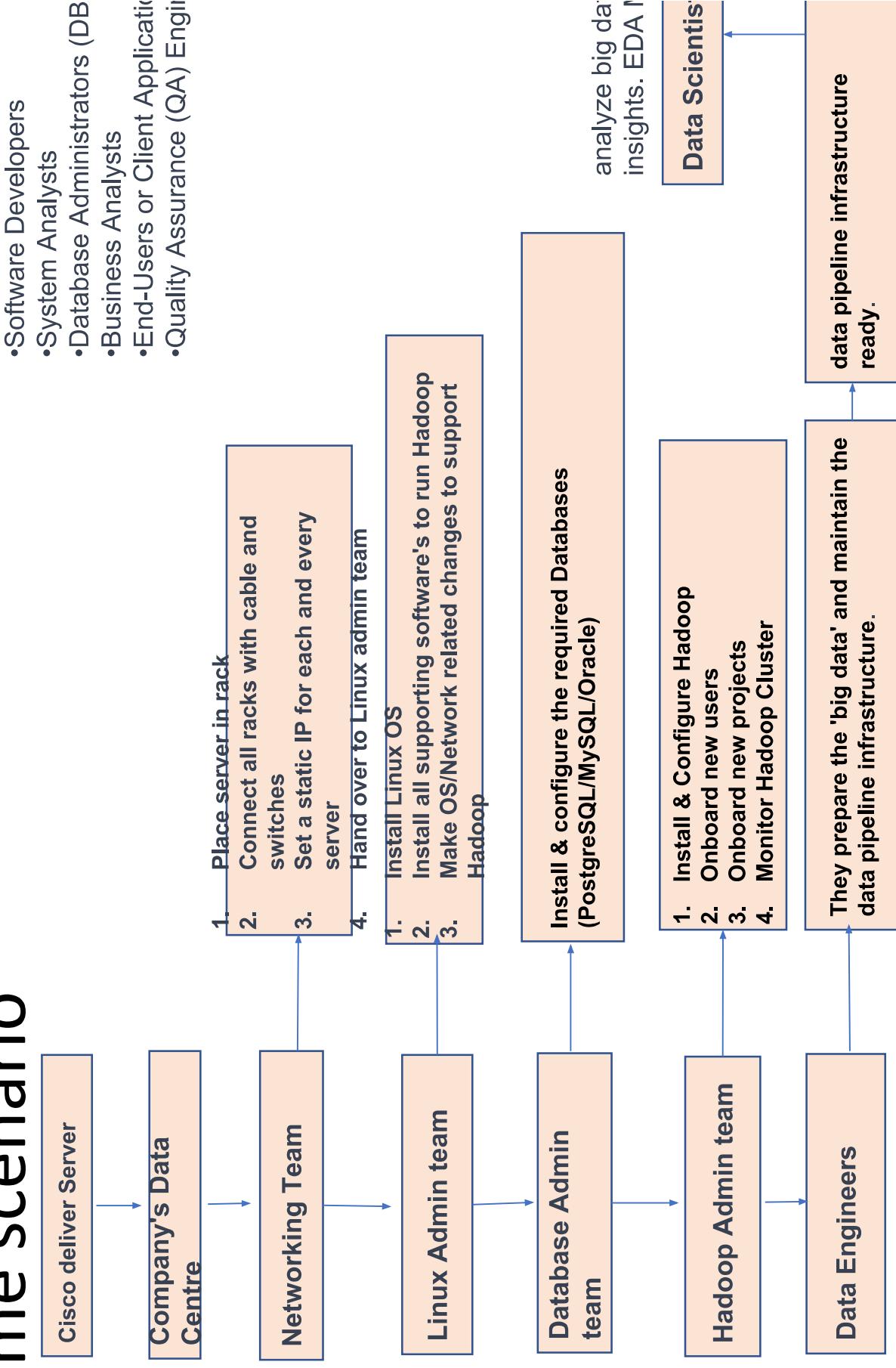
# Real time scenario



**1 NN**  32C, 256G, 24TB,  
1000MB/s  
**100 DNS**  32C, 256G, 24TB,  
1000MB/s  
**1 Sec NN**  32C, 256G, 24TB,  
1000MB/s  
**2 EN**  32C, 256G, 24TB,  
1000MB/s

**104 servers**

# Real time scenario



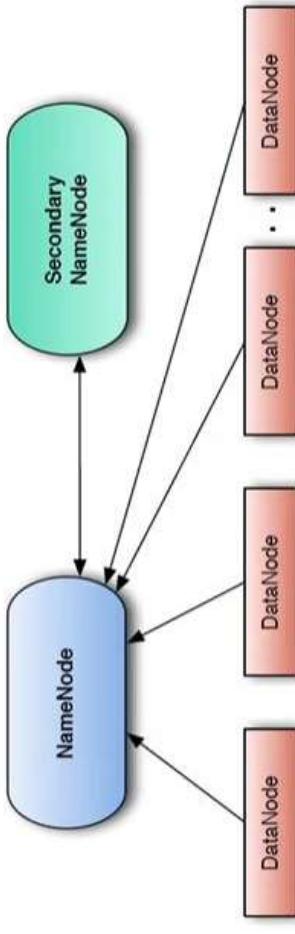
- Software Developers
- System Analysts
- Database Administrators (DBA)
- Business Analysts
- End-Users or Client Application
- Quality Assurance (QA) Engineers

# Hadoop 1.X

## Limitations of Hadoop with Single NameNode

- NameNode is Single Point of Failure as Secondary Name never acts like Primary NameNode in case of Primary NameNode failure
- Chances of losing data if Checkpoint doesn't happen
- Metadata files need to be copied manually in case of Primary NameNode crash.

Hadoop Architecture with Single NameNode



- Sec NN is just like PA and Primary NN is like PM, sec NN cannot replace PM it will only note down PMs work.
- Sec NN only has metadata file fimage stored nothing more than that

## Solution: NameNode High Availability (Active & Standby NameNodes)

# Problem

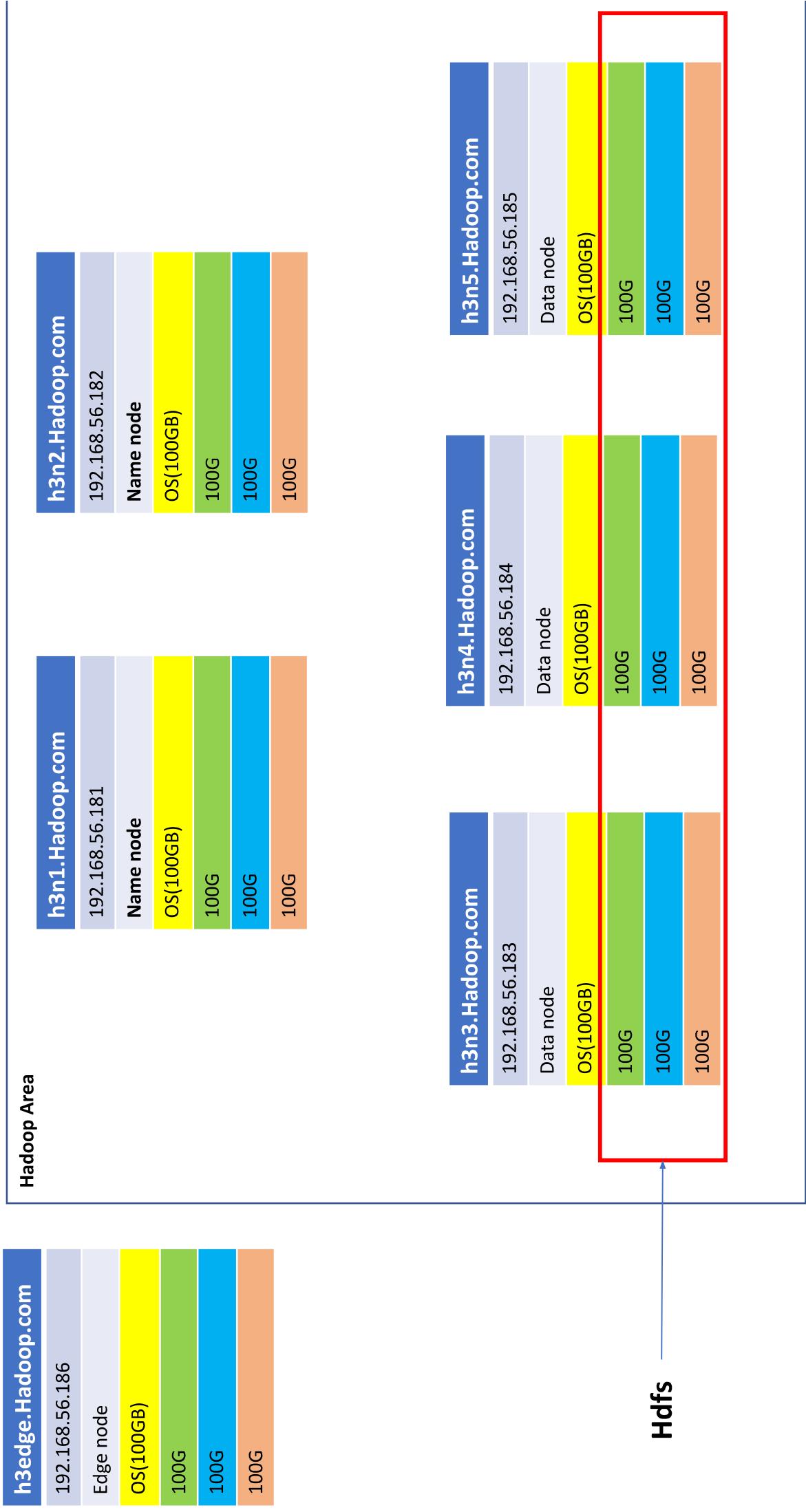
Time	Primary NN	Sec NN
	Edits	Fsimage
8:00	0	0
	40	
9:00	0	40
	20	0
10:00	0	60
	10	
10:30	10	60
11:00		

- At 10:30 Primary NN burnt out gone down completely due to electricity problem spark happen and crashed.
  - No data of last 10 transactions will be available anywhere not even in sec NN
  - What if those 10 transactions of 1 lakh Rs, the Hadoop cluster will say you have not paid
    - It is a big problem.
  - We have to manually copy fsimage from sec NN to Pri NN but there are 10 transactions.
- Hadoop is good but main drawback is Name Node / master Node/Primary name node
  - Hadoop people improved over this and came up with concept of name node with high availability
  - If one name node is not there other name node will take care of there work ([Active and standby name node](#))

## Active and standby name node

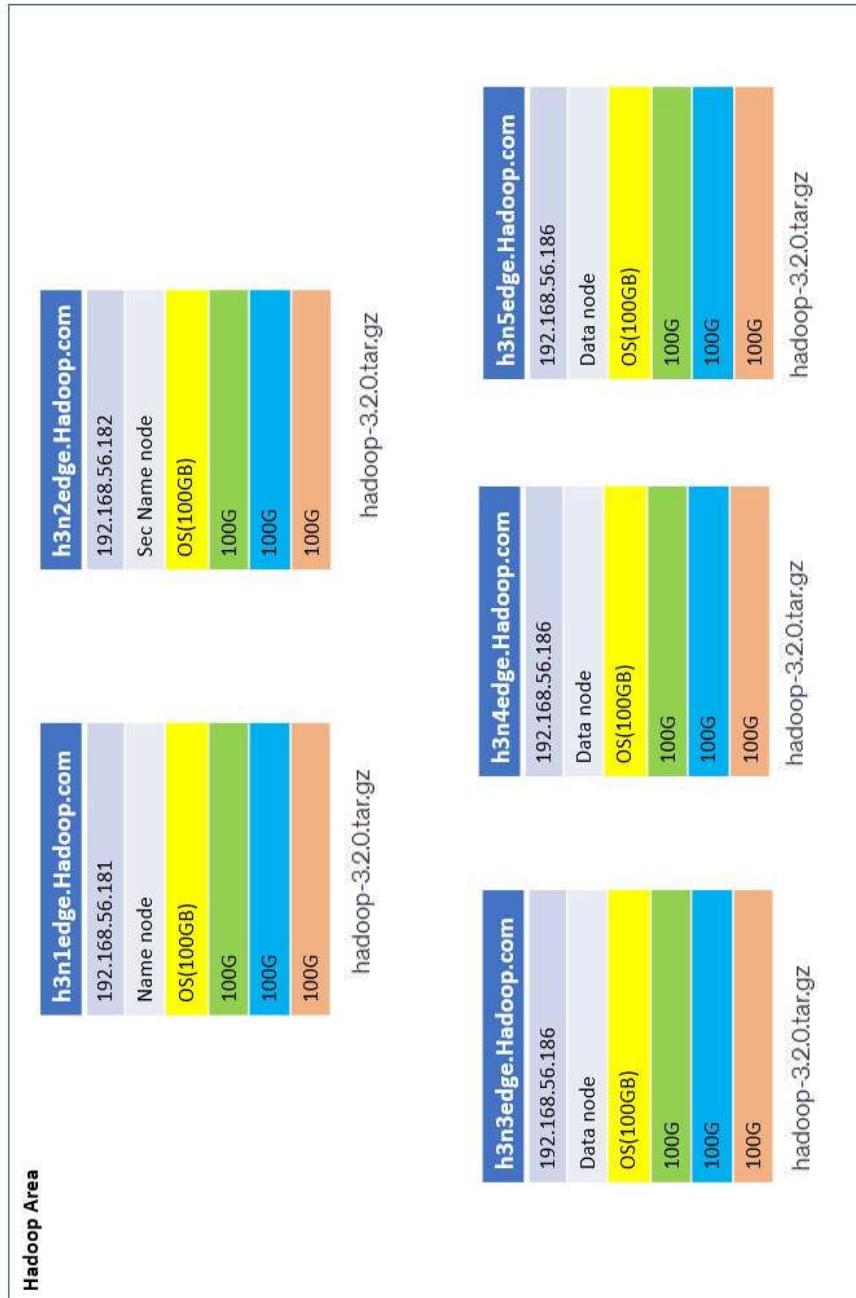
- ✓ In our clusters now we have two name nodes
- ✓ Active name node and standby name node
- ✓ If active name node is down standby name node will become active and it will take care of the work.
- ✓ manager and deputy manager

# Hadoop cluster



# Install Hadoop on every machine

TAR Ball Name	Download Location
hadoop-3.2.0.tar.gz	<a href="https://archive.apache.org/dist/hadoop/core/hadoop-3.2.0/">https://archive.apache.org/dist/hadoop/core/hadoop-3.2.0/</a>



# Hadoop installed on every machine

h3edge.Hadoop.com	
192.168.56.186	
Edge node	
OS(100GB)	Hadoop
100G	
100G	
100G	

Hadoop Area

h3n1edge.Hadoop.com	
192.168.56.181	
Name node	
OS(100GB)	Hadoop
100G	
100G	
100G	

Name node

Sec Name node

OS(100GB)

Hadoop



h3n2edge.Hadoop.com	
192.168.56.182	
Sec Name node	
OS(100GB)	Hadoop
100G	
100G	
100G	

OS(100GB)

Hadoop



h3n5edge.Hadoop.com	
192.168.56.186	
Data node	
OS(100GB)	Hadoop
100G	
100G	
100G	

h3n4edge.Hadoop.com	
192.168.56.186	
Data node	
OS(100GB)	Hadoop
100G	
100G	
100G	

h3n3edge.Hadoop.com	
192.168.56.186	
Data node	
OS(100GB)	Hadoop
100G	
100G	
100G	

h3n2edge.Hadoop.com	
192.168.56.182	
Sec Name node	
OS(100GB)	Hadoop
100G	
100G	
100G	

h3n5edge.Hadoop.com	
192.168.56.186	
Data node	
OS(100GB)	Hadoop
100G	
100G	
100G	

h3n4edge.Hadoop.com	
192.168.56.186	
Data node	
OS(100GB)	Hadoop
100G	
100G	
100G	

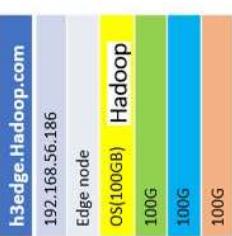
h3n3edge.Hadoop.com	
192.168.56.186	
Data node	
OS(100GB)	Hadoop
100G	
100G	
100G	

# Hadoop clusters runs On configurations file

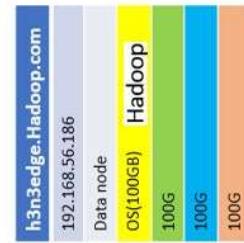
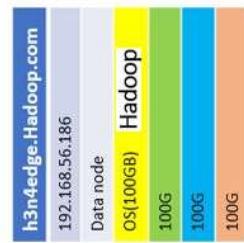
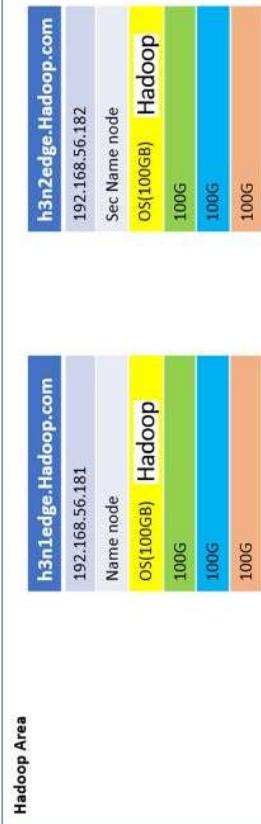
- | hadoop-env.sh | Contains Hadoop Environment Variables |
- | yarn-env.sh | Contains Hadoop Environment Variables |
- | core-site.xml | NameNode info |
- | workers | DataNode(s) info |
- | masters| SNN info|
- |hdfs-site.xml|( hdfs properties-block size replication, fsimage ,location, data block location)

# Fsshell-writing application

```
hdfs dfs -D dfs.replication=1 -put sample.txt /project1/
```



Hadoop Area



fsshell

- When you type the command here and press enter ( means you are invoking an algorithm) what is happening in the back end?
- Back end there is something called **fsshell(algorithm)** (client program) starting.
- It means you are invoking fsshell who is executing your program, fsshell is executing your program.
- fsshell will contact name node

# Fsshell-writing application

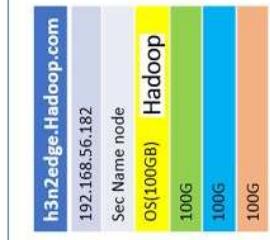
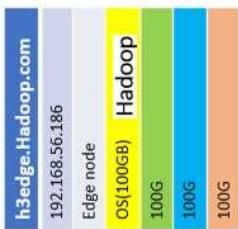
fimage

Edits

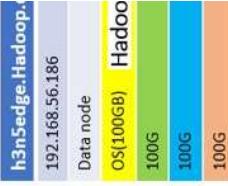
Block Reports

data node info report

```
hdfs dfs -D dfs.replication=1 -put sample.txt /project1/
```



fsshell



- Fsshell will understand which is name node through configuration file on edge node.

- Fsshell- Hey !! name node I want to write sample.txt in to project 1 directory can you help me?

- Name node will check edit log first if the same entry is available or not if it is not there then it will check it in fsimage if not there too then next step
  - Then it will check size of the file and will check if all its data nod has that much space using data node info

# Fsshell-writing application

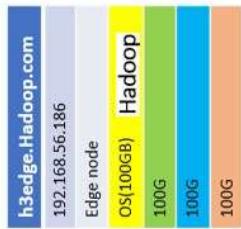
fsimage

Edits

Block Reports

data node info report

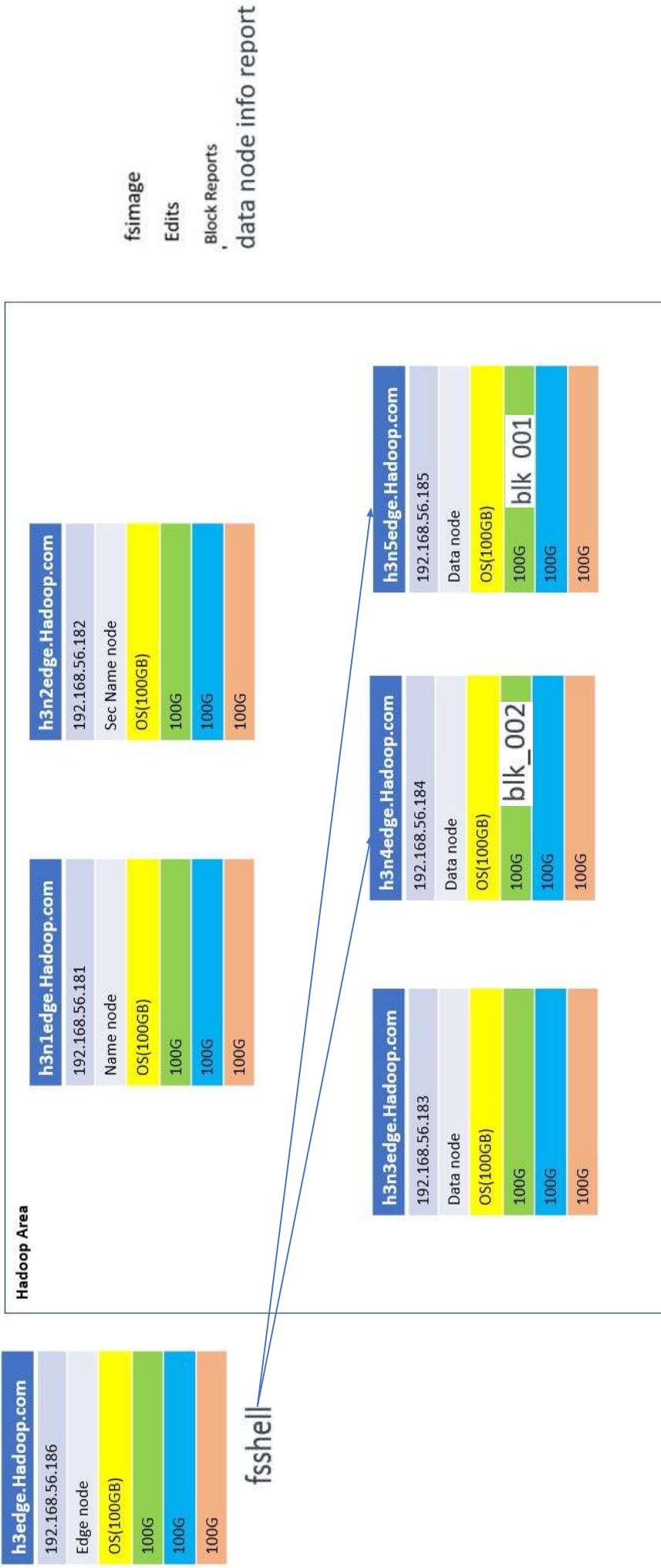
```
hdfs dfs -D dfs.replication=1 -put sample.txt /project1/
```



fsshell

- Now we have 300 GB space available on each disk, so we have space to put 150 MB file
- Name node acknowledge that it has space to put the file and same name file is not available.
- Name Node- ok fsshell accepted your request now cut your data into pieces(blocks)
- fsshell will cut the data into pieces
- Default size is of piece 128MB so 150 will cut into two pieces 128 and 22

# fsshell-writing application



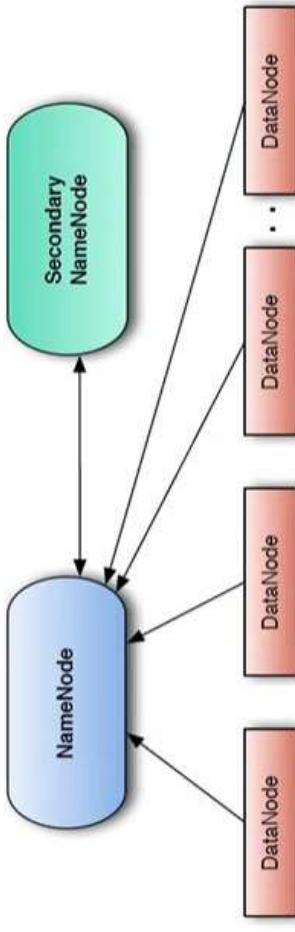
- fsshell-hey name node I cut the file into pieces as per your suggestion Name node- put first piece in 192.158.56.184 name it as blk\_001 and second piece in 192.158.56.185 with name it as blk\_002 then fsshell will contact 185 and will put it and then name node will update block report

# Hadoop 1.X

## Limitations of Hadoop with Single NameNode

- NameNode is Single Point of Failure as Secondary Name never acts like Primary NameNode in case of Primary NameNode failure
- Chances of losing data if Checkpoint doesn't happen
- Metadata files need to be copied manually in case of Primary NameNode crash.

Hadoop Architecture with Single NameNode



- Sec NN is just like PA and Primary NN is like PM, sec NN cannot replace PM it will only note down PMs work.
- Sec NN only has metadata file fimage stored nothing more than that

## Solution: NameNode High Availability (Active & Standby NameNodes)

# Active Name Node Vs Standby Name Node

- In a typical HA(High Availability) cluster, two or **more** separate machines are configured as Name Nodes.
  - At any point in time, exactly one of the Name Nodes is in an Active state, and the others are in a Standby state.
- 
- The Active Name Node is responsible for all Client operations in the Cluster.
  - The Standby Name Node is simply acting as a slave, performing checkpoint on regular intervals to provide a fast failover if necessary.

# Active Name Node Vs Standby Name Node

## Challenges with Name Node High Availability

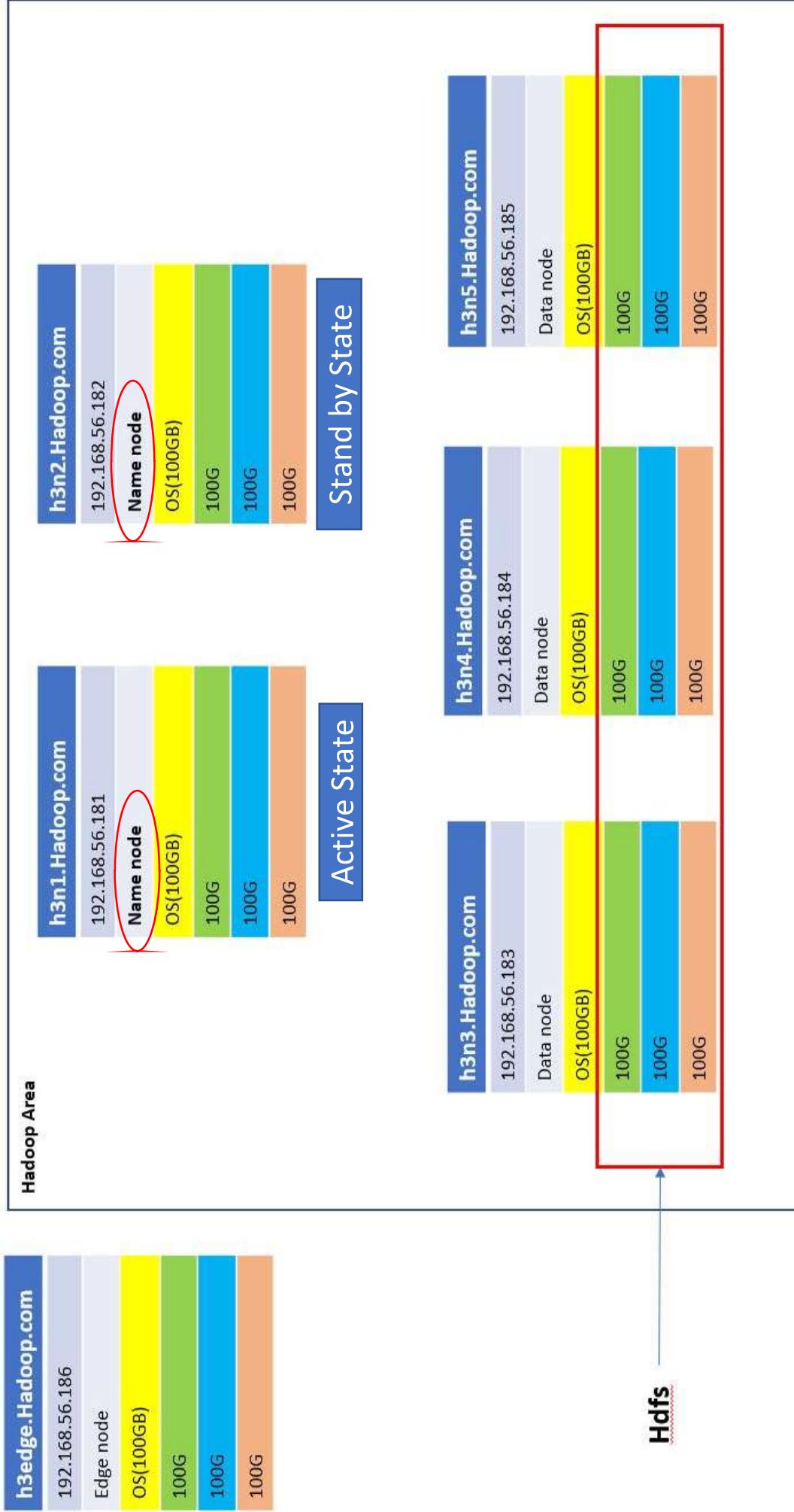
- Which NN should be Active and Which NN(s) should be standby?
- Who will change the state of Standby Name Node to Active when Active Name Node is down
- **Solution:** Additional mechanism Zookeeper is required to automate this process

# Active Name Node Vs Standby Name Node

## Challenges with Name Node High Availability

- How to maintain common copy Edits when we have multiple Name Nodes?
  - What in case if common copy of Edits is lost?
- 
- Solution: Additional mechanism Quorum Journal Manager (a group of **journal nodes**) is required to maintain multiple copies of Edits.

# Two name Nodes



# High Availability

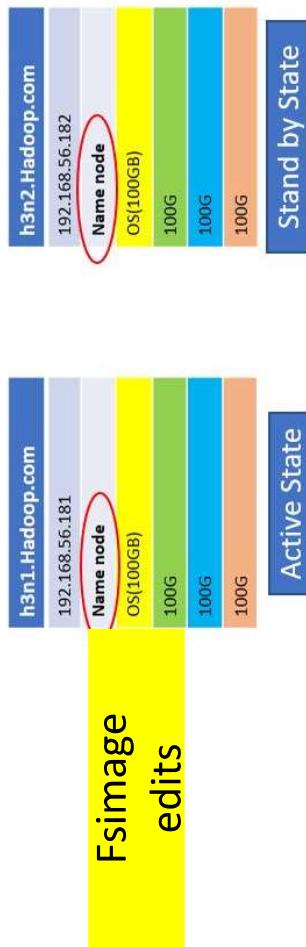
- If active is down standby will become active

- Which name node should be active, and which should be standby
- Who will make other NN active if any one of NN1 or NN2 is down .
- It is not possible for Hadoop admin to seat and watch when NNs goes down
- Requires automatic mechanism
- The answer for first challenge is zookeeper

Time	NN1	NN2
9:00	Standby	Standby
9:01	Active	Standby
9:50	Down	Active
10:00	Restarted	Active
10:01	Standby	Active

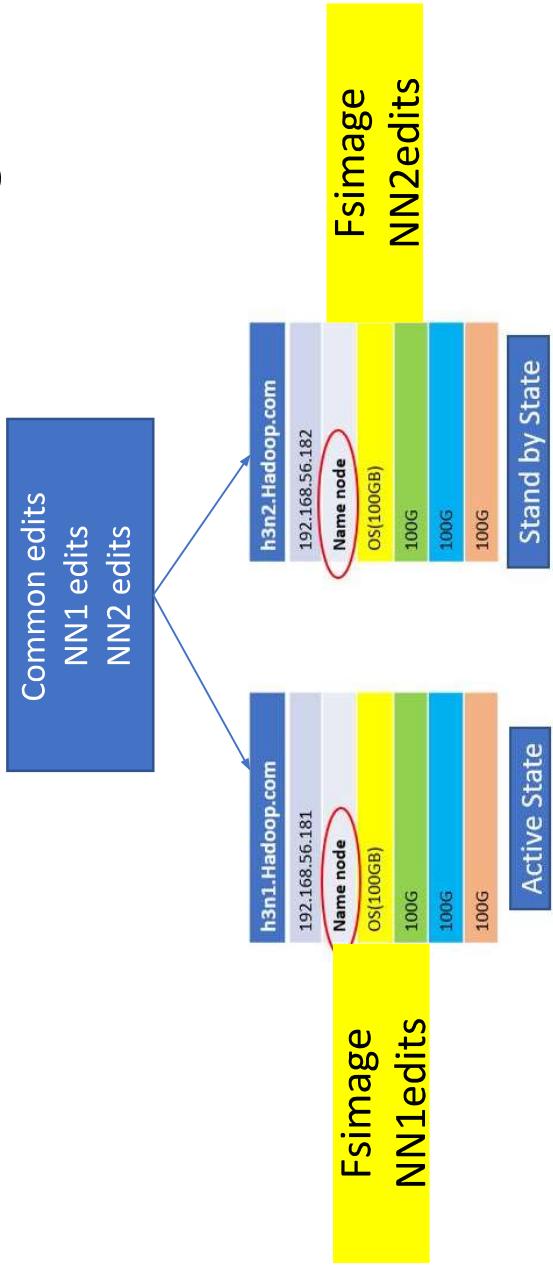
# Problem 1

- If NN1 is active it will write its metadata ( fsimage and edit logs) into its own hard disk .
- If NN2 is active it will write its meta data ( fsimage and edit logs) into its own hard disk
- Suppose NN1 is active and at 9:00 it has written 10 edits how NN2 will understand about 10 edits
- Whatever NN1 is doing that should be known to NN2 but they are not sharing each other



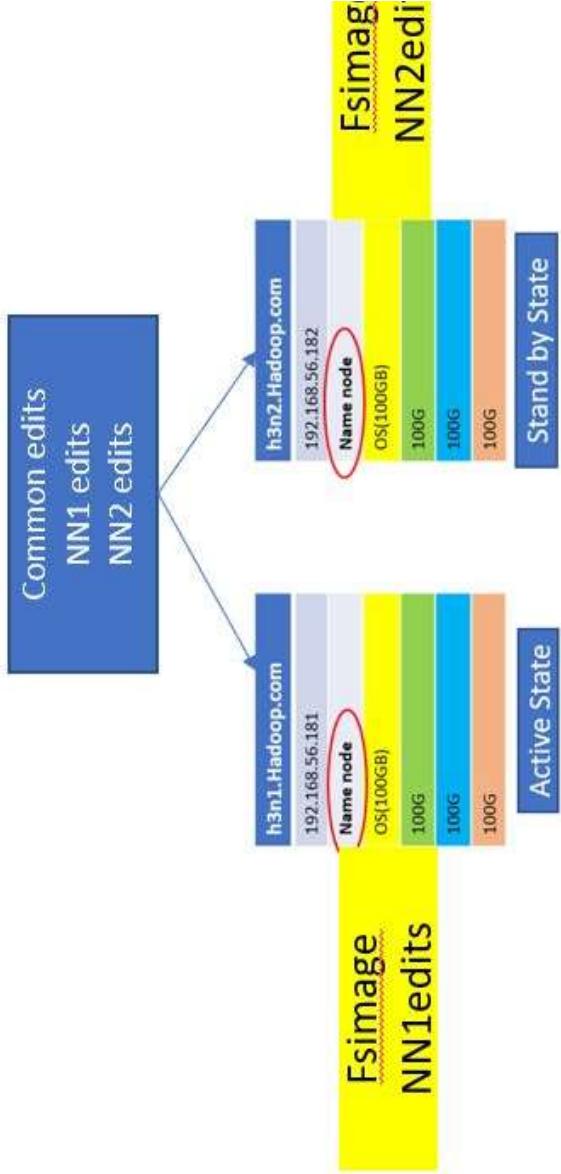
# Problem 1

- If NN1 is down or crash then all its edits and fsimage is gone.
- If NN2 is down or crash then all its edits and fsimage is gone.
- It is important for us to make sure each name node should know what other name node is doing
- So we should have a common place where we put metadata for NN1 when it is active and metadata of NN2 is doing when it is active.



# Problem solved

- NN1 when active will write edits in its hard disk as well as in common place also
- NN2 when active will write edits in its hard disk as well as in common place also
- If NN1 written edits and it goes down then NN2 will become active and play a role of name node so it will take from common area whatever edits done by NN1
- This mechanism is called as **Journal node**



## Problem 2

- When you start both NNs, both enter into Standby mode
- Challenge1: Which NN should be selected as active when we have 2 standby NNs
- Challenge2: Who will make standby as active when active NN is down
- Solution is Zookeeper we will install software called zookeeper to solve this problem

# Zookeeper

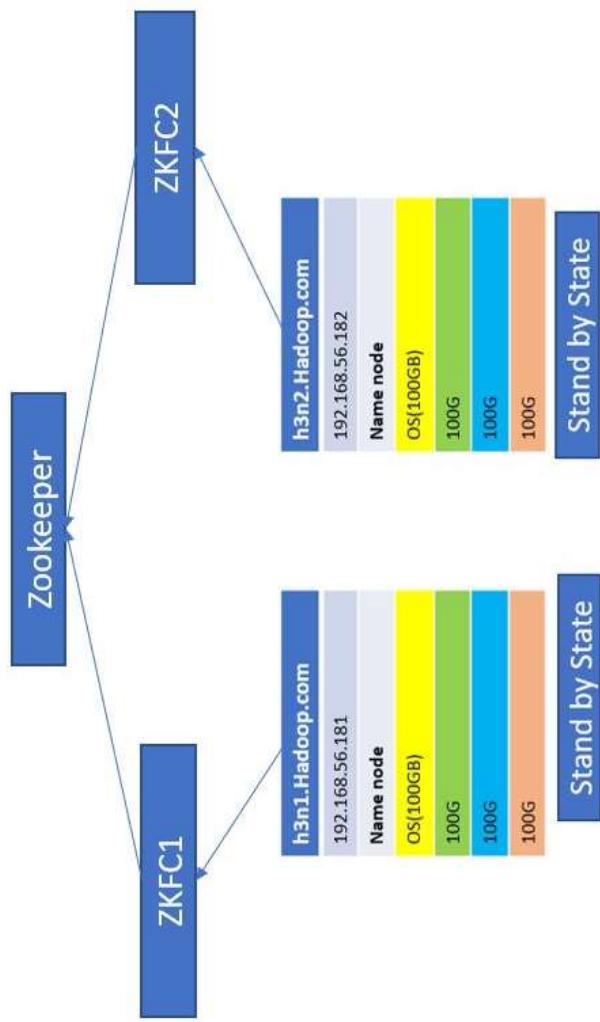
- ZooKeeper (ZK) is a highly-available, highly-reliable and fault-tolerant coordination service for distributed applications.
- **Ensemble** : A group of ZooKeeper Nodes worked as a single unit rather than individual
- **Quorum** : No. of Nodes that should be up and running in the ensemble
- Number of Nodes that should be up and running in the Ensemble to take a collective decision is calculated by using below formula.
  - $N - \left(CEIL\left(\frac{N}{2}, 0\right) - 1\right)$
  - where “N” is the number of total Nodes in Ensemble.

# Zookeeper

- What is ZNode?
- ZooKeeper maintains an active connection with all its Clients using a heartbeat mechanism.
- Furthermore, ZooKeeper keeps a session for each active Client that is connected to it and assigns a key which is called ZNode.

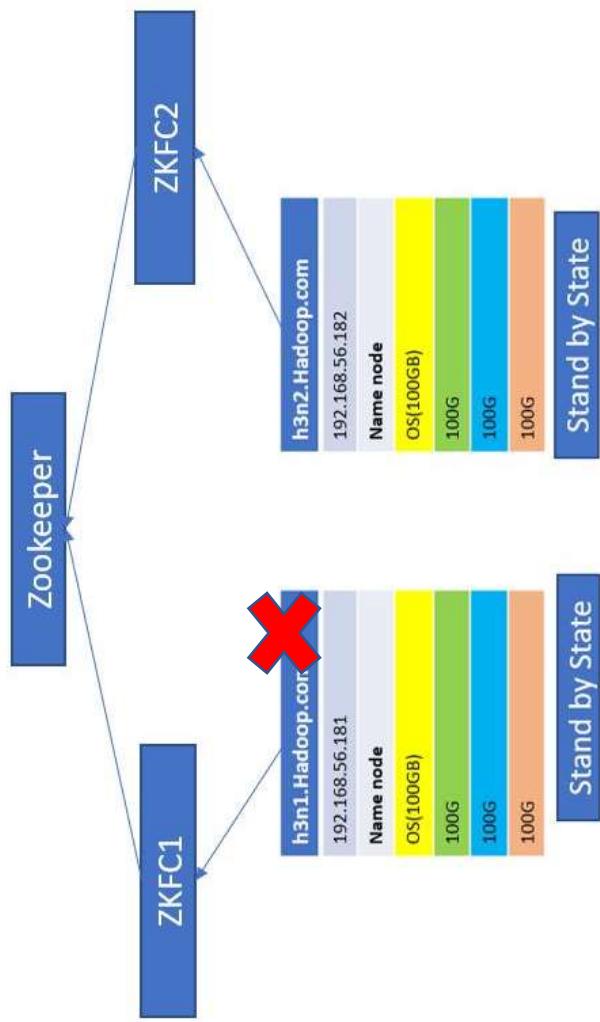
# Zookeeper failure controllers (ZKFC)

- Which NN should be selected as active when we have 2 standby NNs
- NNs will send heartbeat to zookeeper, whenever heartbeat zookeeper gets first it will make that NN as active , like first come first serve
- NN send heart beats to zookeeper through zookeeper failure controller as NN are master nodes and ZKFC acts as their PA



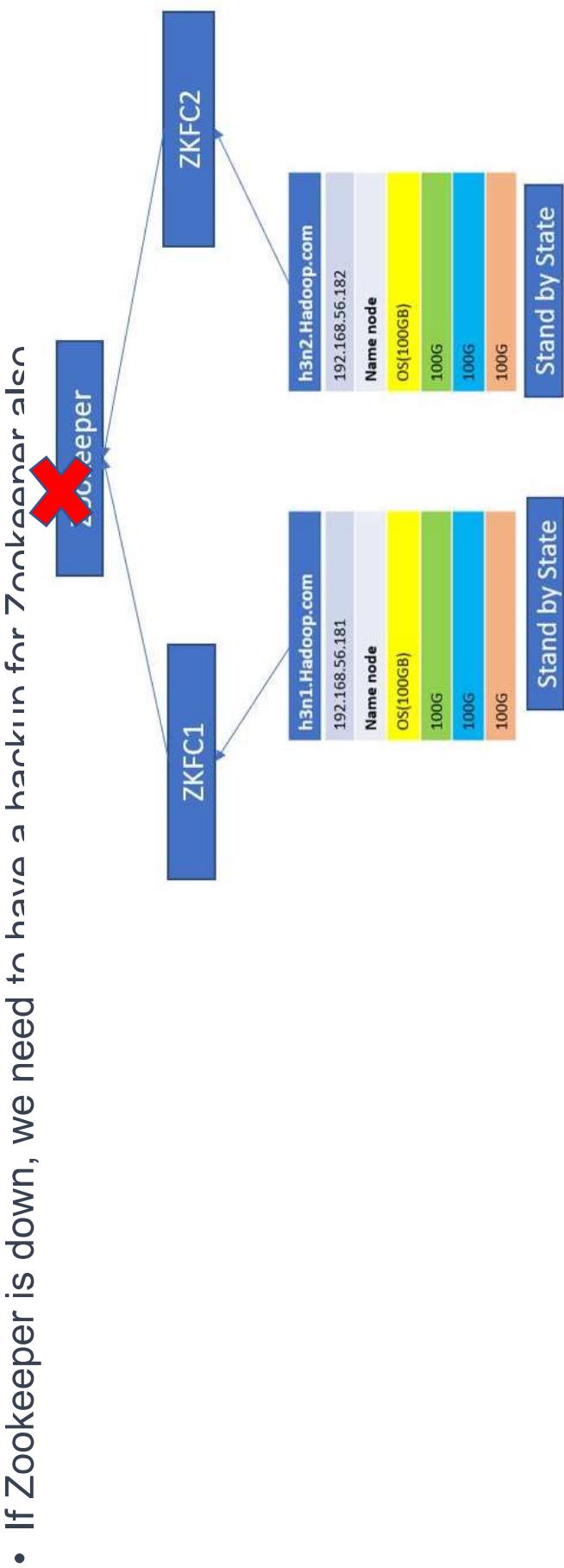
# Zookeeper failure controllers (ZKFC)

- If NN1 is down ZKFC will tell zookeeper hey my boss health is not good, please take necessary action
- Zookeeper will make NN2 active in case one NN is down
- After some time, service is started ( as we assume that system is not crashed)
- ZKFC1 will report to zookeeper that my boss is back
- Zookeeper will tell ZKFC that fine let it be in standby mode NN2 is taking care now
- Now what will happen if zookeeper is down
- If NN is down and Zookeeper is down, then NN1 will never come to active state.



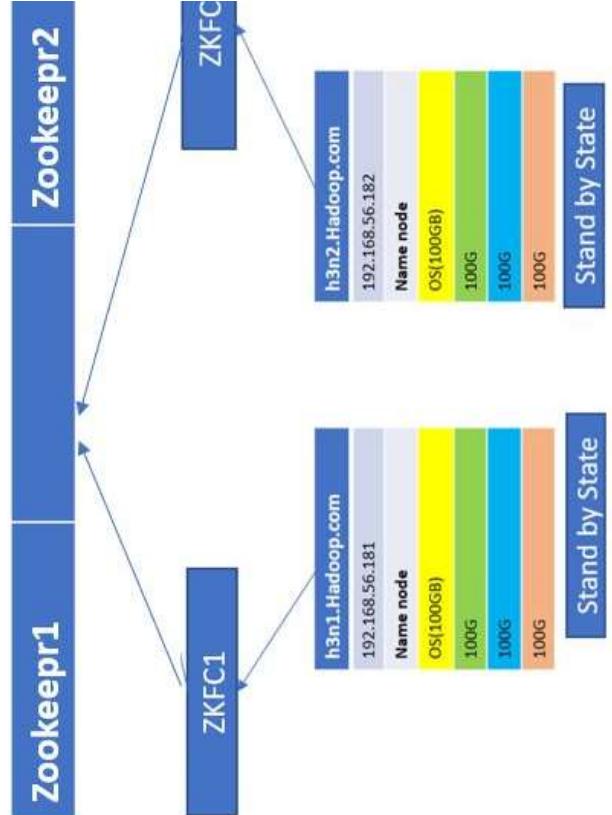
# Zookeeper failure controllers (ZKFC)

- Zookeeper is very important if its not there there won't be anyone who will monitor the health of name node if active name node is down standby name node will not become active
- First, we must start zookeeper service.
- If we start NN first, then without Zookeeper both the name node will remain in stand by mode
- We should have high availability for Zookeeper also as there are chance that any service can go down in real time



# Zookeeper Backup

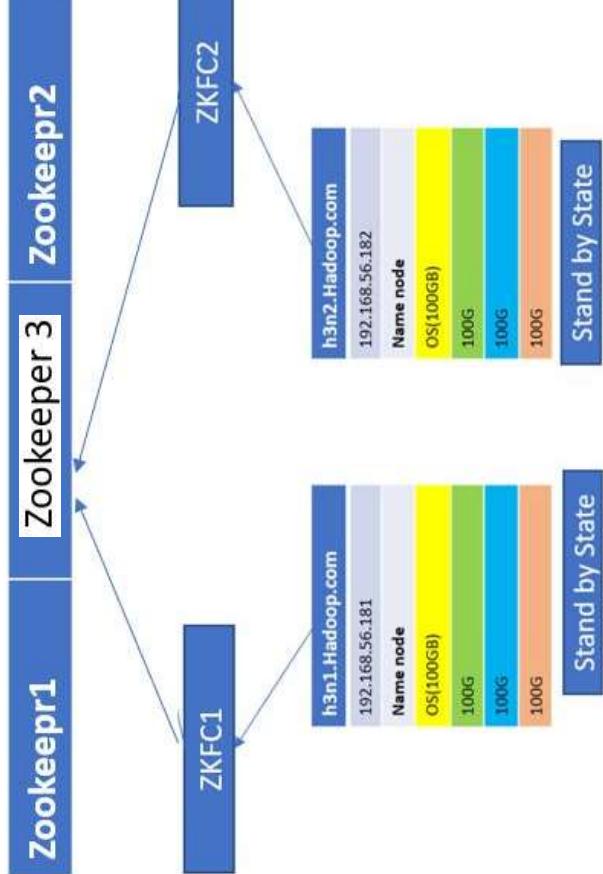
- When you have two Zookeepers or more than two Zookeepers then the concept of voting will be introduced
  - In case of single Zookeeper whichever NN sends Heartbeat first Zookeeper will make that NN active
  - In case of tie there is a problem, so we introduce 3 zookeepers
  - If ZKFC1 sends heartbeat to ZK1 and ZK2 first so NN1 got 2 votes and NN2 got 0 in this case NN1 will become active
  - If ZKFC2 sends heartbeat to ZK1 and ZK2 first so NN2 got 2 votes and NN1 got 0 in this case NN2 will become active
  - If exactly at the same point of time ZKFC1 Sends heartbeat to ZK1 and ZKFC2 sends heartbeat to ZKFC2 so there is tie
  - It seems odd number of Zookeeper is good idea, So we need



# Zookeeper Backup

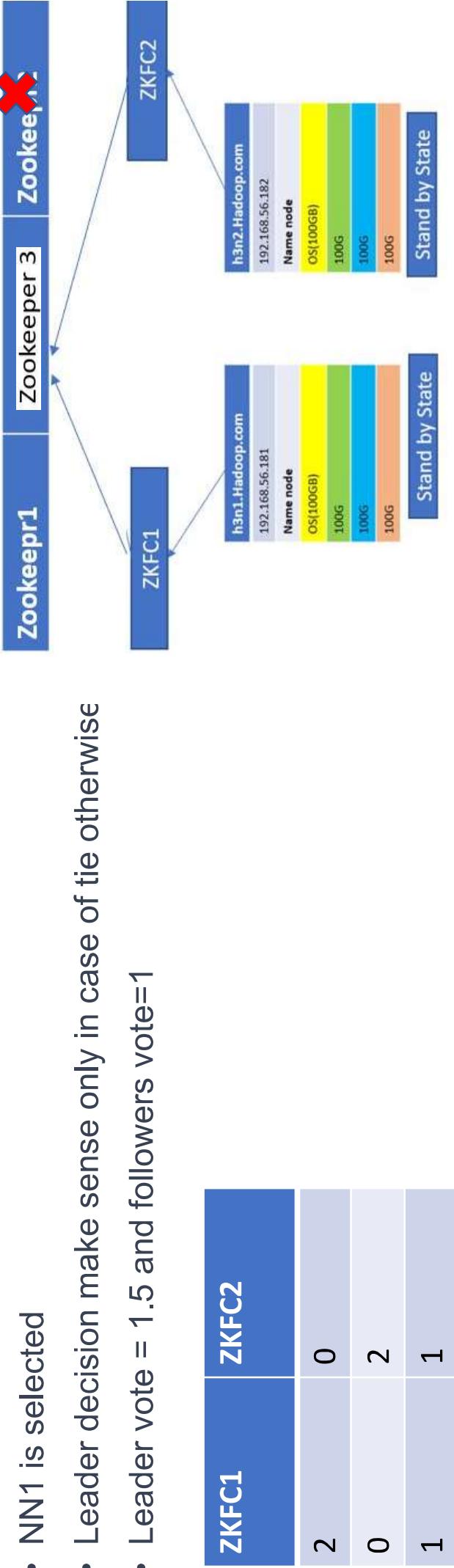
- So we need to introduce 3<sup>rd</sup> Zookeeper

ZKFC1	ZKFC2	Zookeepr1	Zookeepr2	Zookeepr3
3	0	1	2	1
2	1	0	3	2
1	2	3	0	1
0	3	2	1	0



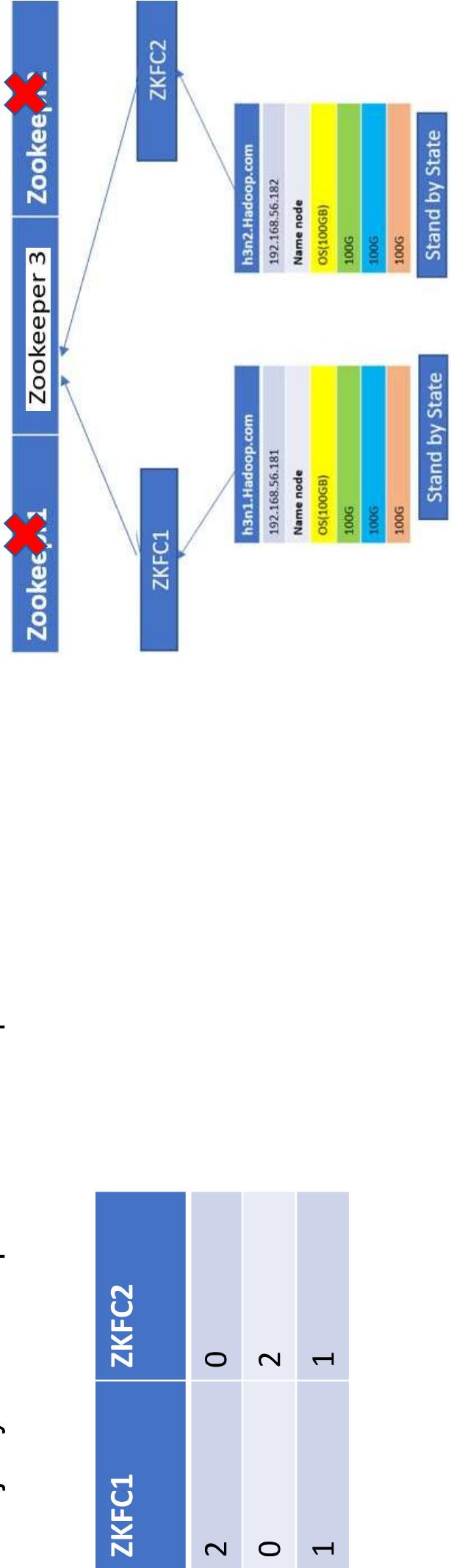
# Zookeeper Backup

- What is ZK2 is down
- ZK internally elect leader and follower before starting any process
- Leader follower game one ZK is elected to be leader and other becomes followers
- If leader goes down other follower or immediate follower will becomes leader
- In case of tie If leader say NN1 and follower says NN2
  - NN1 is selected
  - Leader decision make sense only in case of tie otherwise
  - Leader vote = 1.5 and followers vote=1



# Zookeeper Backup

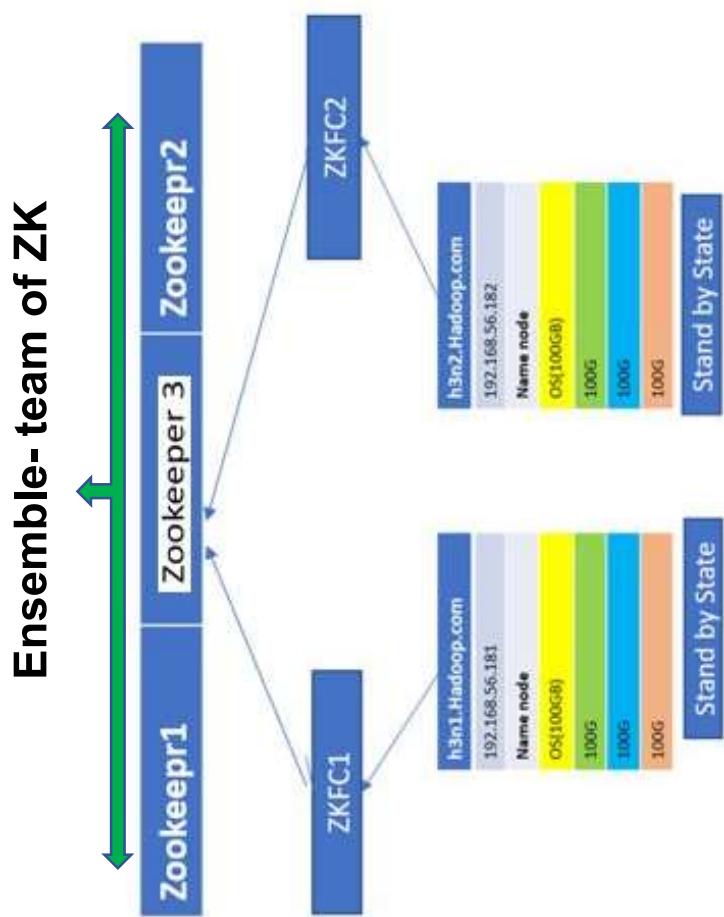
- What if two zookeepers are down
- Quorum based means majority of the zookeepers should be available.
- At least two Zookeepers should be available.
- With one zookeeper it can not do anything i.e. to decide which NN should me made active
- Majority of Zookeepers should be present



- ZooKeeper (ZK) is a highly-available, highly-reliable and fault-tolerant coordination service for distributed applications.
- **Ensemble** : A group of ZooKeeper Nodes worked as a single unit rather than individual
- **Quorum** : No. of Nodes that should be up and running in the ensemble
- Number of Nodes that should be up and running in the Ensemble to take a collective decision is calculated by using below formula.
  - $N - \left(CEIL\left(\frac{N}{2}, 0\right) - 1\right)$
  - where “N” is the number of total Nodes in Ensemble.

# Zookeeper

- **Ensemble** : A group of ZooKeeper Nodes worked as a single unit rather than individual



- **Quorum** : No. of Nodes that should be up and running in the ensemble (majority to take decision) to maintain Quorum we should have at least 3

# of ZK	Quorum ( Majority)
3	2
4	3
5	3
6	4
7	4
.	.
.	.
.	.

$$\begin{aligned}
 N = 7 \text{ then } N - (\text{CEIL}(\frac{N}{2}, 0) - 1) &= 7 - \left(\text{CEIL}\left(\frac{7}{2}, 0\right) - 1\right) \\
 &= 7 - (\text{CEIL}(3.5, 0) - 1) = 4
 \end{aligned}$$

## Zookeeper

# Zookeeper

- What is ZNode?
- ZooKeeper maintains an active connection with all its Clients (ZKFC1,ZKFC2) using a heartbeat mechanism.
- Furthermore, ZooKeeper keeps a session for each active Client that is connected to it and assigns a key(token) which is called ZNode.
- ZKFC will send heartbeat to ZK in return ZK send token number or Znode number
- If NN1 got token no 1 and NN2 has got token 2 NN1 active
  - Token number is Znode number
  - NN1 will become active

# Types of Znode

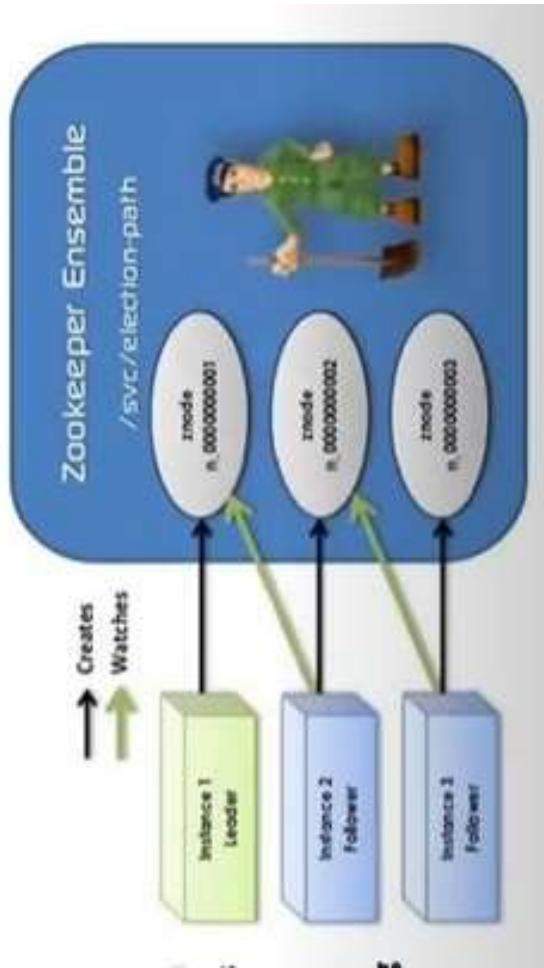
- **Persistent** – alive even after its Client session is disconnected and need to be deleted explicitly ( Token in bank will remain valid even if you go out priority will be given to you)
- **Ephemeral** – alive only for the lifetime of its Client session and deleted automatically( Inside the bank token valid as you go out come later your token is invalid)
- (ZKFC has to sent heartbeat to ZK if it does not send then new Znode will be assigned)

# Why odd number of Nodes in ZK Ensemble?

- In ZK Ensemble of 5 Nodes,  $\text{Ceil}[5/2, 0] - 1 = 2$  can go down to maintain the quorum.
- In ZK Ensemble of 6 Nodes,  $\text{Ceil}[6/2, 0] - 1 = 2$  can go down to maintain the quorum.
- In the second scenario, the extra ZooKeeper Node doesn't give any benefit to the Ensemble to maintain the quorum. So, replicating to that one extra ZooKeeper Node is just a performance overhead.

# ZooKeeper Leader Election

- During ZooKeeper Leader Election process, each ZooKeeper in the ensemble creates **ephemeral-sequential ZNode** in the election path.
- The ZooKeeper with the smallest ZNode id is the “**Leader**” and remaining ZooKeepers in the group acts like “**Follower**”.
- If Leader goes down, the next follower corresponding to the Leader will become the new Leader. (Znode is ephemeral seq hence leader will get new token)



# Journal Nodes

Common Edits Edits 1-3 Edits 4-10	
NN1	NN2
Active (has written Edits 1-3) <b>Now if Name node is down</b>	Active( has written Edits 4-10)

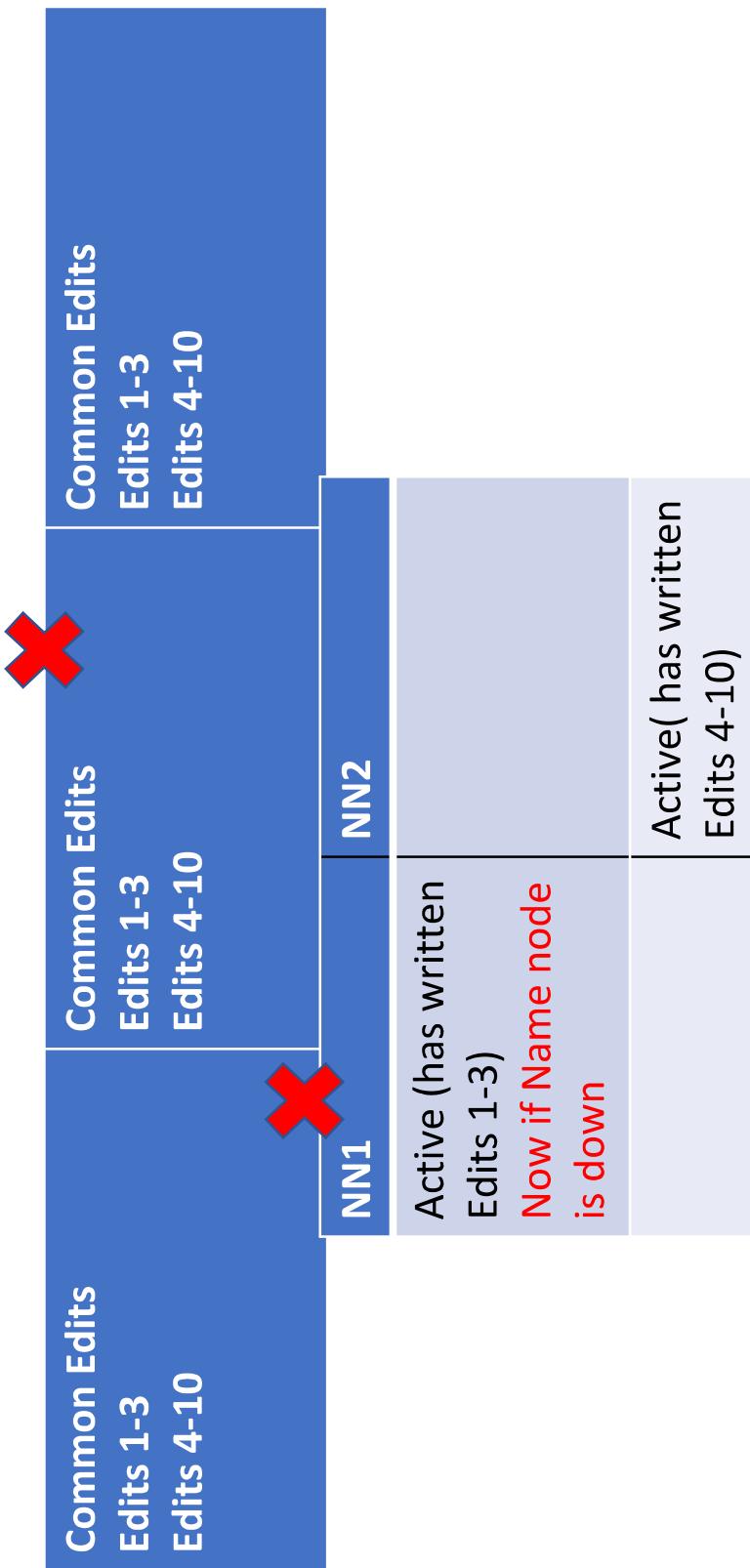
Will write one copy in common edits also

# Journal Node

Common Edits Edits 1-3 Edits 4-10		NN2
NN1	Active (has written Edits 1-3) <b>Now if Name node is down</b>	Active( has written Edits 4-10)

NN 2 will understand only edits 4-10 which is a big problem so we maintain common edits at three place

# Journal Node



If one common edit is down NN2 can place Edits at other Common Edits

# Journal node

<b>Common Edits</b> Edits 1-3 Edits 4-10	<b>Common Edits</b> Edits 1-3 Edits 4-10	<b>Common Edits</b> Edits 1-3 Edits 4-10
<b>NN1</b>	<b>NN2</b>	

**NN1**

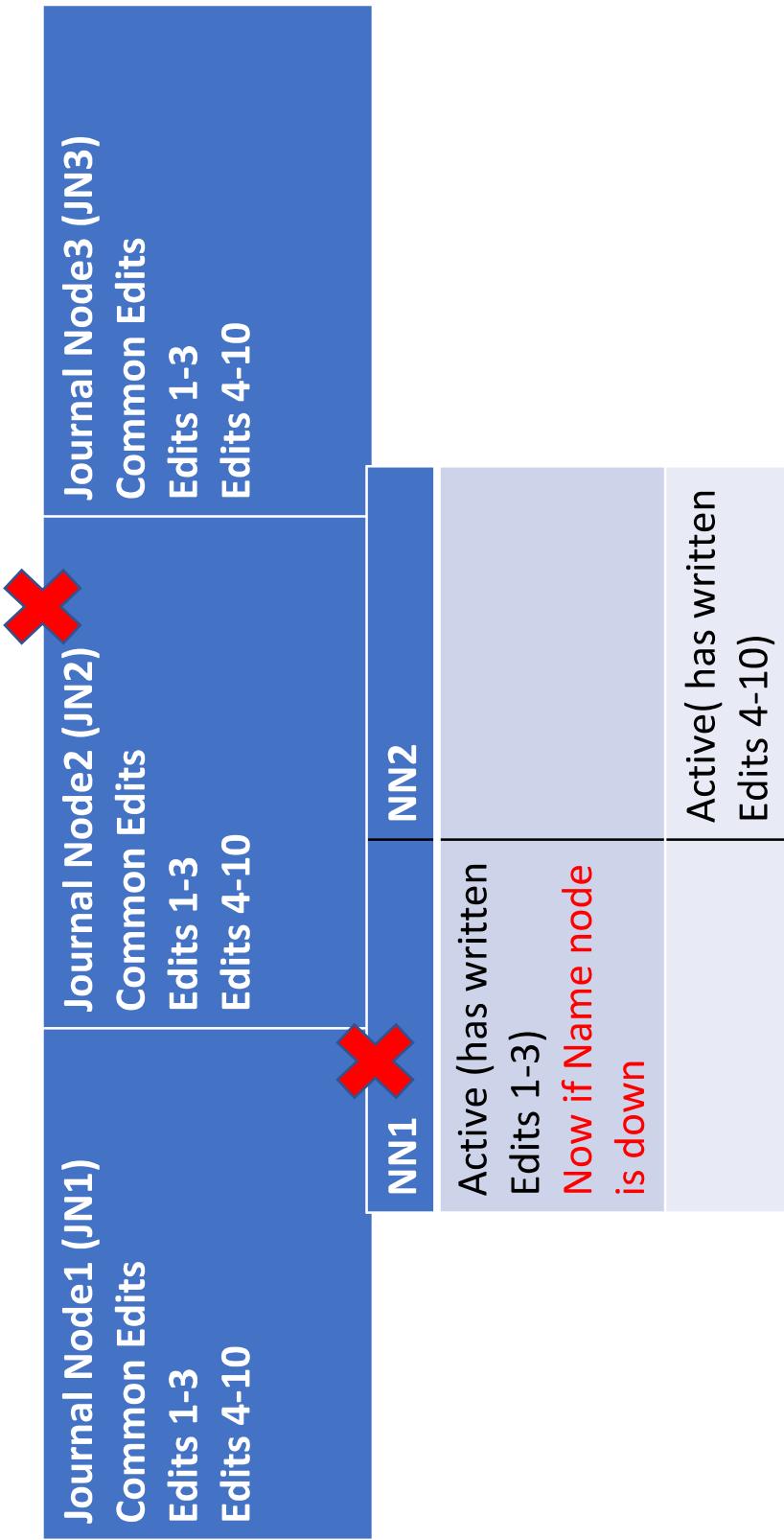
Active (has written  
Edits 1-3)  
**Now if Name node  
is down**

**NN2**

Active( has written  
Edits 4-10)

If one common edit is down NN2 can place Edits at other Common Edits

# Journal node



If one common edit is down NN2 can place Edits at other Common Edits JN will store all edits By NN1 and NN2

- **Quorum Journal Manager**

# Quorum Journal Manager (a group of Journal Nodes)

- The QJM runs as a group of journal nodes, and each new transaction (also called Edit) must be written to a majority of the journal nodes.
- Typically, there are three journal nodes, so the system can tolerate the loss of one of them.
  - In order for the Standby node to keep its state synchronized with Active NameNode, both NameNodes communicate with a group of separate daemons called Journal Nodes (JNs).
  - When a new transaction is performed by Active NameNode, it durably logs it into a majority of these Journal Nodes.
  - The Standby NameNode is capable of reading these transactions from the Journal Nodes.

# Journal node

Journal Node1 (JN1)	Journal Node2 (JN2)	Journal Node3 (JN3)
<b>Common Edits</b> Edits 1-3 Edits 4-10	<b>Common Edits</b> Edits 1-3 Edits 4-10	<b>Common Edits</b> Edits 1-3 Edits 4-10
<b>NN1</b>	<b>NN2</b>	
	Active (has written Edits 1-3) <b>Now if Name node is down</b>	Active( has written Edits 4-10)

If Two JN are down NN will not perform any Action here also concept of Quorum

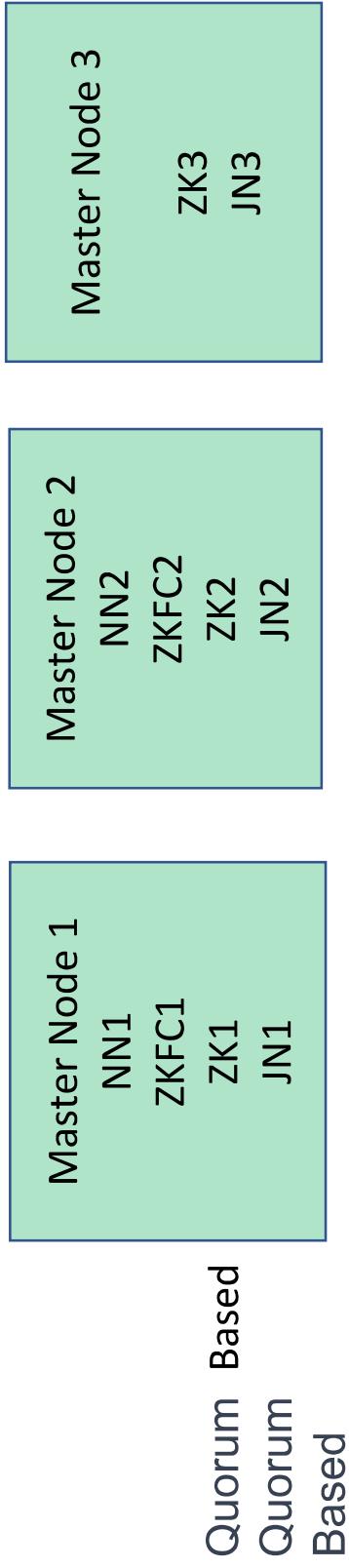
# Quorum Status

Journal Node1 (JN1) Common Edits Edits 1-3 Edits 4-10	Journal Node2 (JN2) X Common Edits Edits 1-3 Edits 4-10	Journal Node3 (JN3) Common Edits Edits 1-3 Edits 4-10
	X NN1	NN2 Active (has written Edits 1-3) <b>Now if Name node is down</b>

Quorum - **live** (if majority of JNs available)  
Quorum - **Down** (if majority of JNs not available)

When ever NN write Edits in to JN first it will check the quorum status if its alive or not  
If Quorum is Down NN will say no JN available I will not write the transaction

# Master-Worker Architecture(2.X)

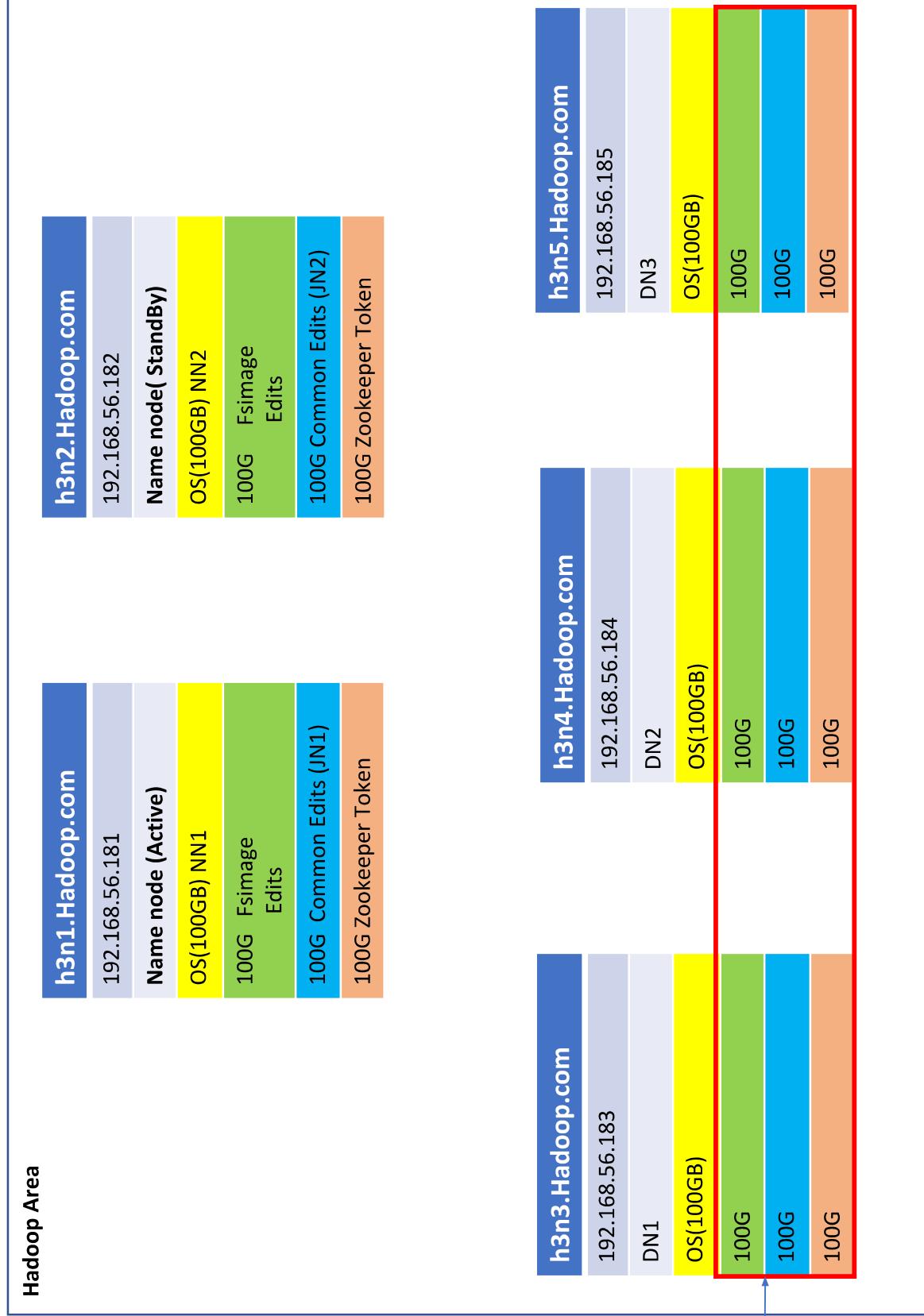


All the services Zookeeper ZKFC and JN are small Services, it does not require separate Server we install/configure them on Master Node where NN are installed We put separate hard disk on Master node to run these services

# Master-Worker Architecture(2.X)

IP Address	FQDN	Hostname	Role in Storage Layer
192.168.56.181	h3n1.hadoop.com	h3n1	NameNode, ZKFC, Journal Node
192.168.56.182	h3n2.hadoop.com	h3n2	NameNode, ZKFC, Journal Node
192.168.56.183	h3n3.hadoop.com	h3n3	Datanode, Journal Node
192.168.56.184	h3n4.hadoop.com	h3n4	Datanode
192.168.56.185	h3n5.hadoop.com	h3n5	Datanode
192.168.56.186	h3edge.hadoop.com	h3edge	Edge Node

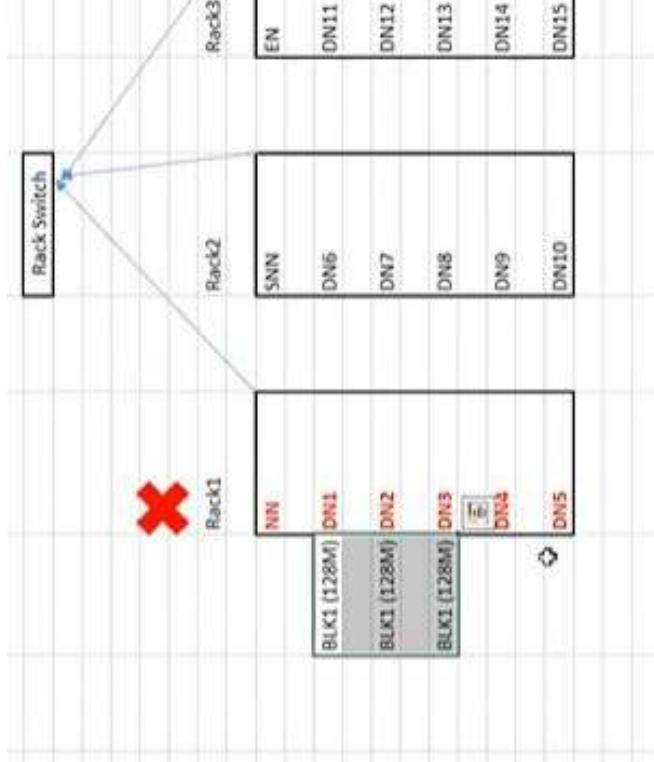
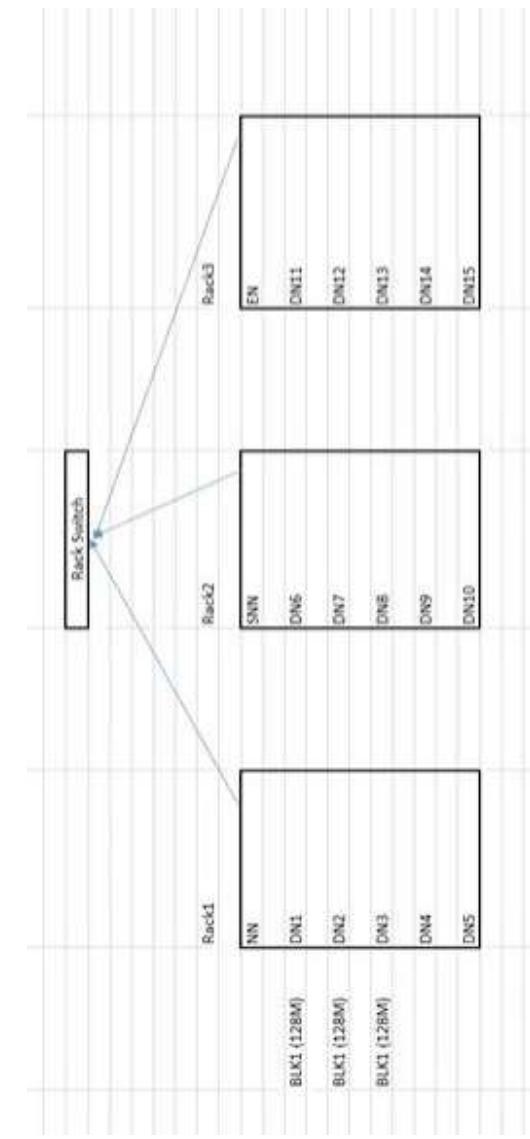
# Master-Worker Architecture(2.X)



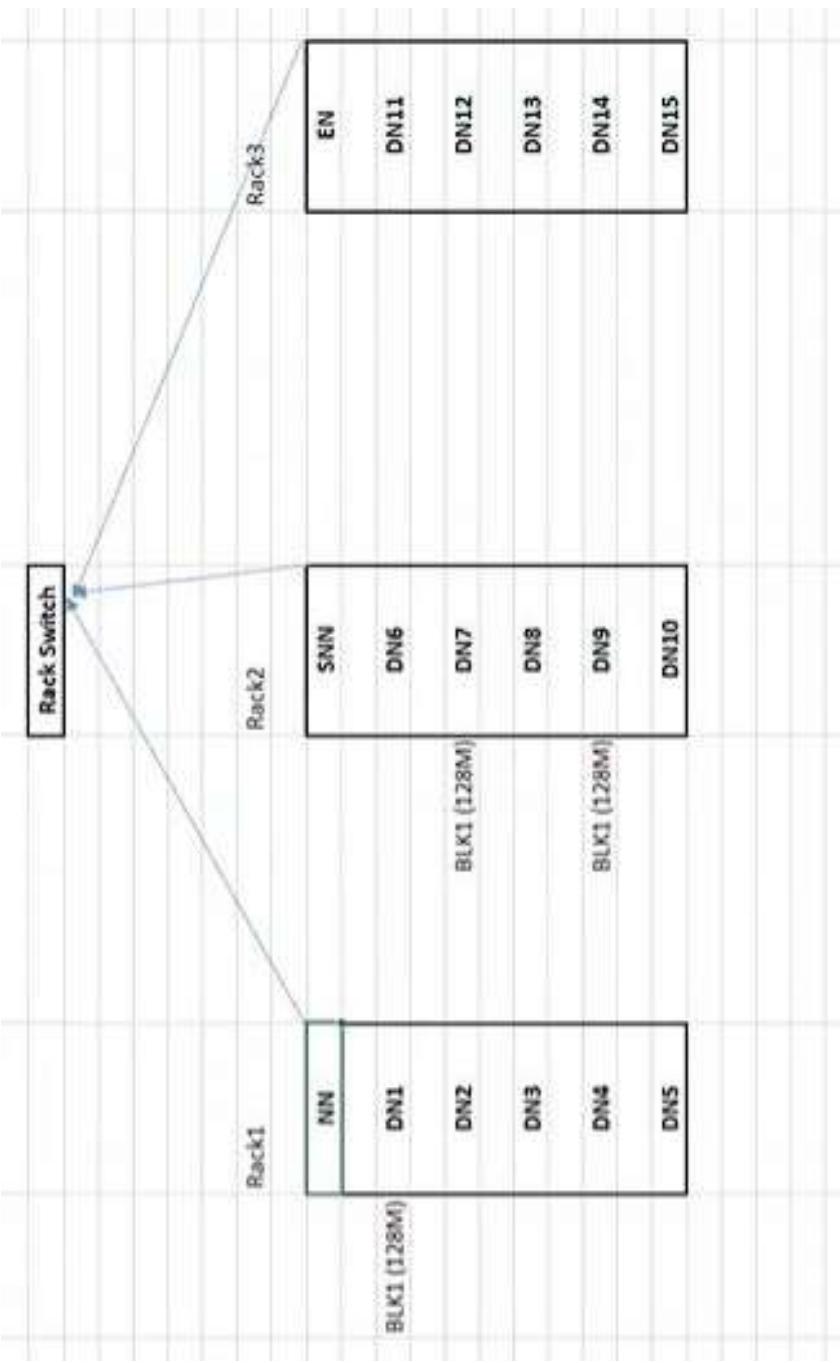
**Hdfs**

# Rack awareness

## Communication



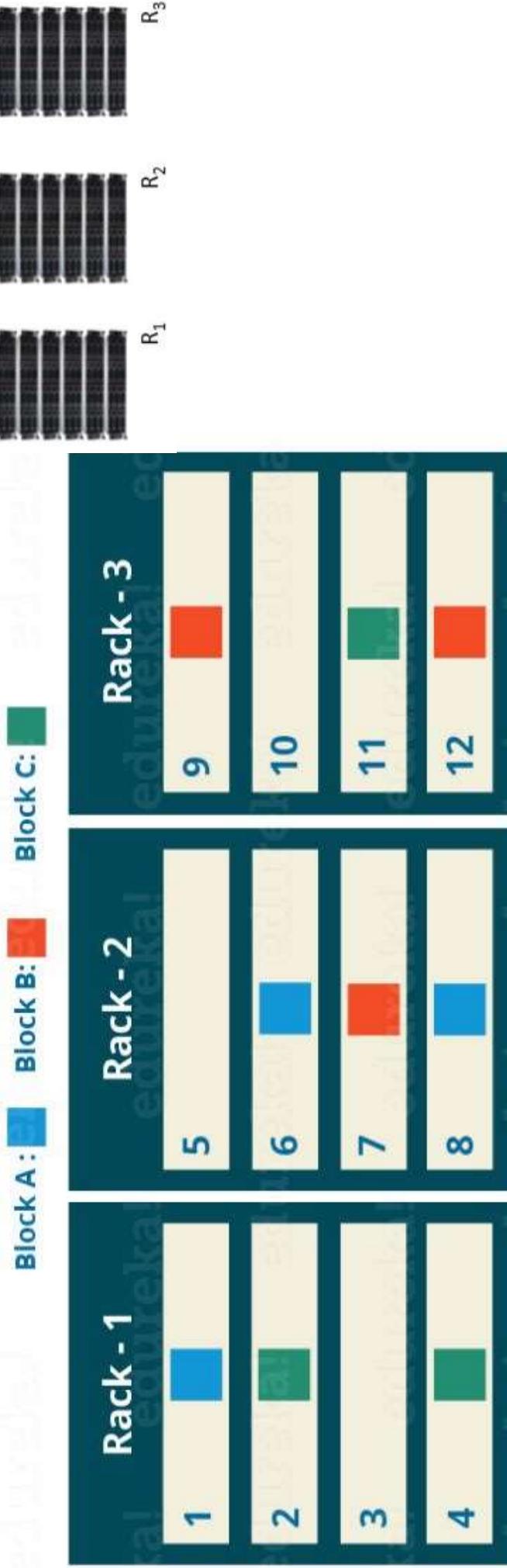
# Rack awareness



- First copy of the block is placed on random DataNode with more free space and which is not too busy
- Second copy of the block is placed on a DataNode residing on different rack (Idea is to not to lose all the data if entire rack fails)
- Third copy of the block is placed on different DataNode in the same rack as the second copy (Assumption that higher bandwidth and lower latency in Rack)
- You are reading the data from R1 suppose for some reason DN1 failed then you have to switch the rack and go to DN7 so if we put Blocks in different rack switching the rack will cause latency

## Rack Awareness:

### Rack Awareness Algorithm

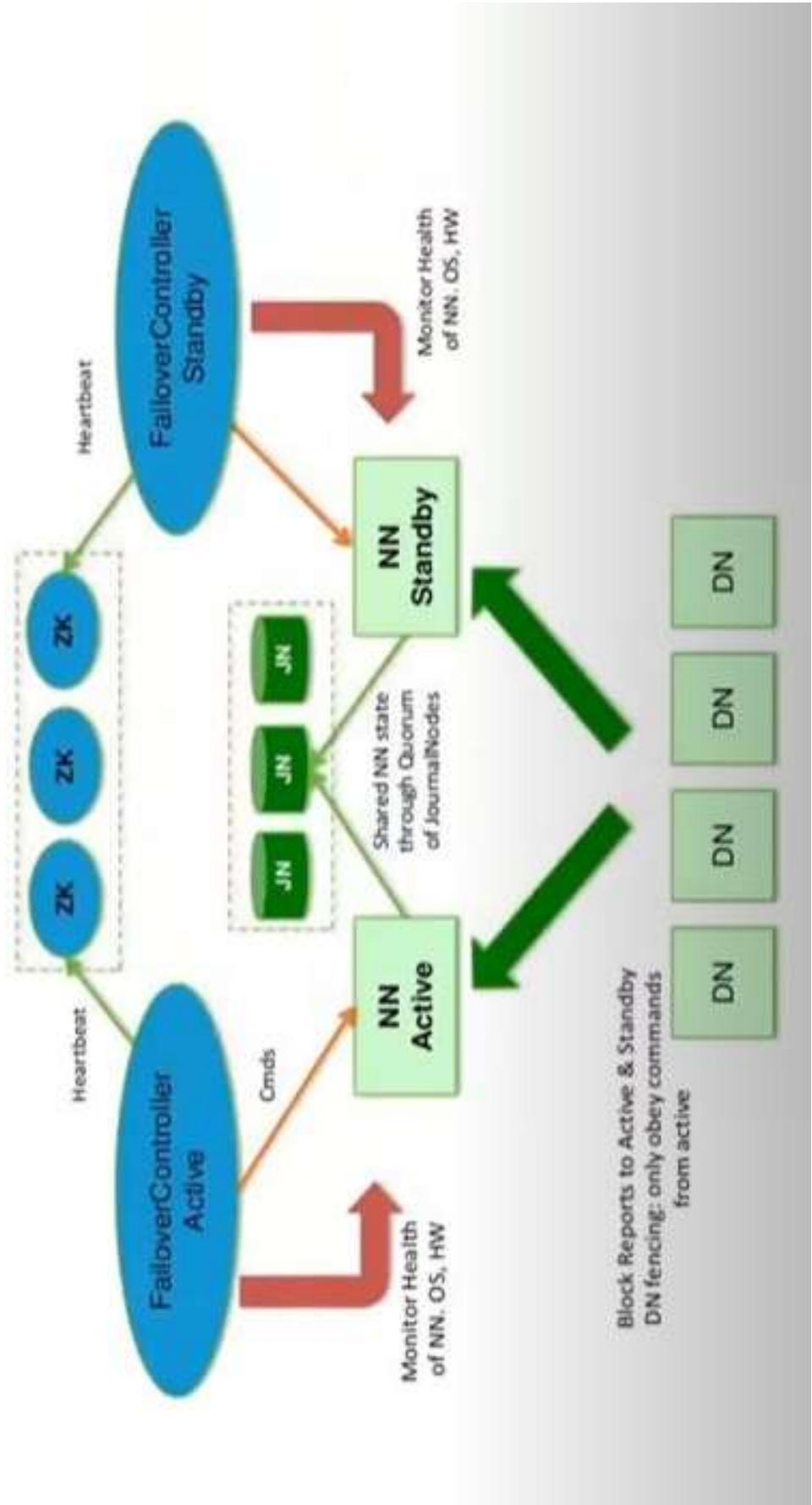


- The NameNode makes sure that copies of your data are spread out across different racks of computers. This way if one rack has a problem, your data is still safe on another rack.
- It uses a smart rule called the "Rack Awareness Algorithm".
- Suppose the replication factor configured is 3. Now rack awareness algorithm will place the first block on a local rack.
- It will keep the other two blocks on a different rack. It does not store more than two blocks in the same rack if possible.

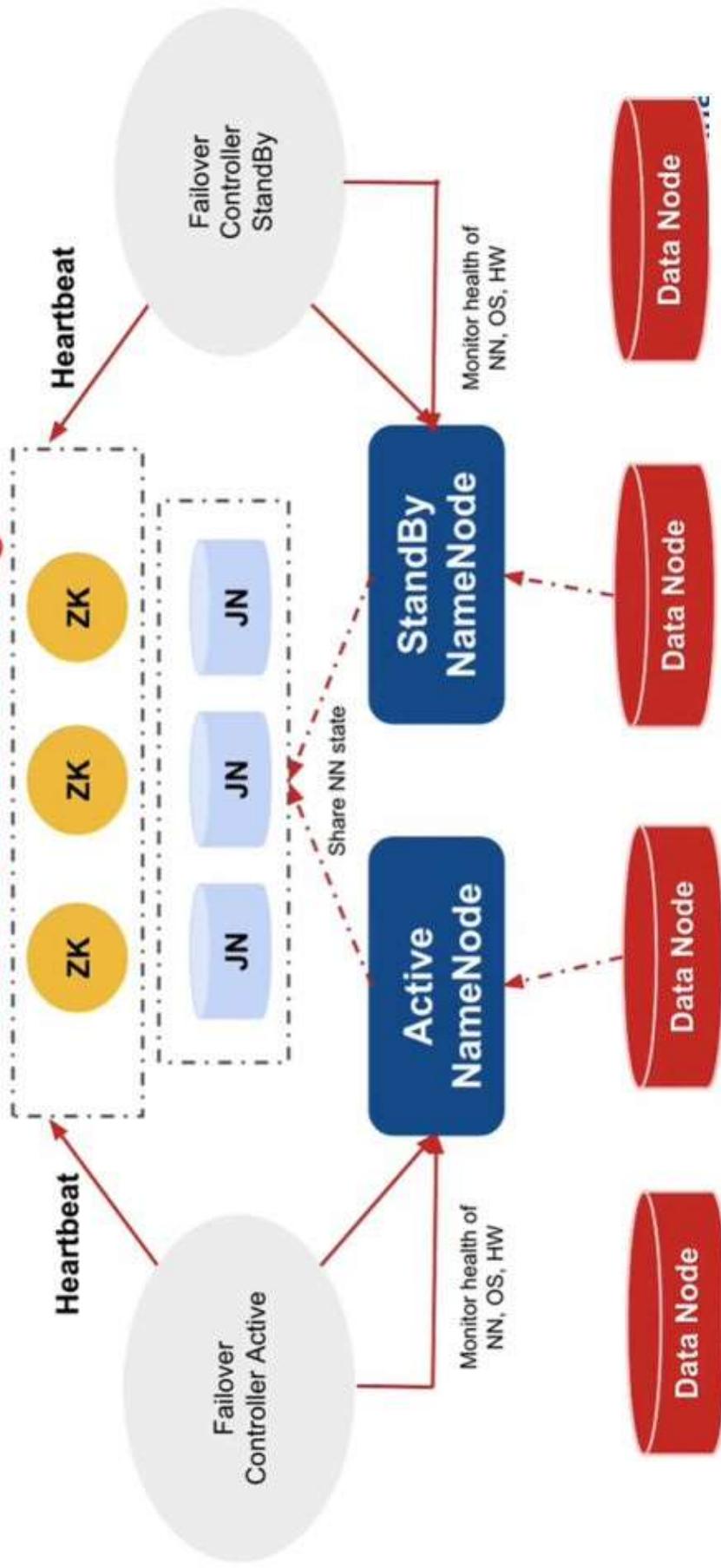
# Hadoop Federation

- Hadoop Federation enhances the scalability and robustness of HDFS by allowing multiple NameNodes to manage separate namespace volumes within the same cluster, thereby decentralizing the metadata management and removing the single point of failure and performance bottleneck associated with a single NameNode system.

# Hadoop 2.X Architecture - NameNode HA (Using Journal Nodes & ZooKeeper)



## Failover & Fencing



# Master-worker Architecture(2.x)

Services		
Master	Active NameNode	Standby NameNode
Supporting Services	<ul style="list-style-type: none"><li>• SupS: ZKFC</li><li>• SupS: JournalNode</li><li>• SupS: Zookeeper</li></ul>	<ul style="list-style-type: none"><li>• SupS: ZKFC</li><li>• SupS: JournalNode</li><li>• SupS: Zookeeper</li></ul>
Worker Service	Data Node	
Edge Node		

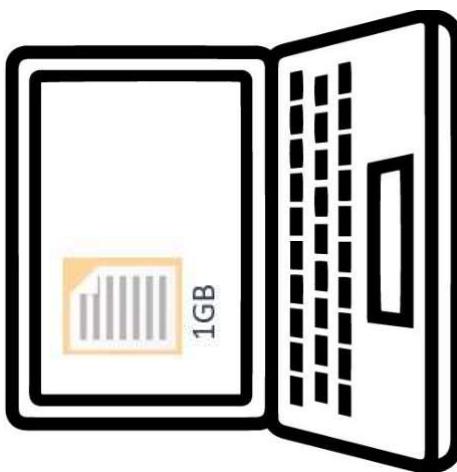
# Hadoop 3.X Setup differences in Storage Layer

Majorly,

- Minimum required Java version is **Java 1.8**
- Default Block size is **256MB**
- Support for more than **2 NameNodes** (1 Active & Multiple Standby NameNodes)
- Support for Erasure Encoding in HDFS (requires storage overhead up to 50% instead of 200%)
- Intra-DataNode Balancer
- Can be scaled for more than **10,000** nodes per Cluster
- Difference in default port numbers (Ex: NameNode port is 9870 instead of 50070)
  - `hdfs --daemon stop/start <service name>` (No `hadoop-daemon.sh`)
- Supports Microsoft Windows & Microsoft Azure Data Lake filesystem

# ErasSure Encoding in HDFS

- 1GB file with replication factor 3
- 1 GB is replicated 3 times in HDFS on data nodes
- Default block size is 256
- Suppose we have 100 nodes

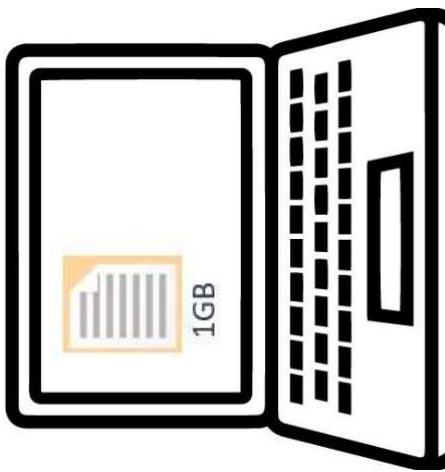


	BLK1	256	DN1(256M)	DN2(256M)	DN3(256M)	768
BLK2		256	DN4(256M)	DN5(256M)	DN6(256M)	768
BLK3		256	DN14(256M)	DN15(256M)	DN16(256M)	768
BLK4		256	DN41(256M)	DN51(256M)	DN61(256M)	768
						3072

- To save 1GB of file with replication factor 3 it is taking 3GB space
- To save 1PB of file size with replication factor 3 it will take 3PB
- if we feel that storing 3 copies is a problem Hadoop is giving option to store the block with erasure encoding algorithm

# Erasure Encoding in HDFS

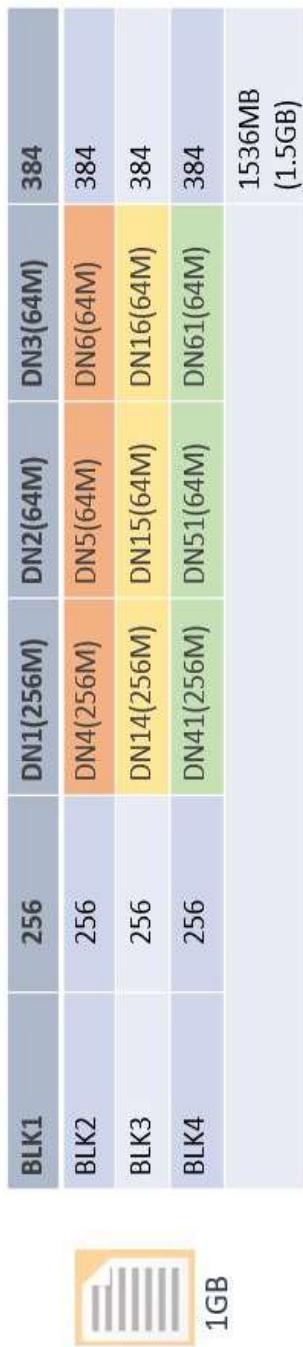
- They will keep the first copy as it is and with the help of parity they will reduce the size to 64 MB



	BLK1	256	DN1(256M)	DN2(64M)	DN3(64M)	384
	BLK2	256	DN4(256M)	DN5(64M)	DN6(64M)	384
	BLK3	256	DN14(256M)	DN15(64M)	DN16(64M)	384
	BLK4	256	DN41(256M)	DN51(64M)	DN61(64M)	384
					1536M (1.5Gi)	

Hadoop 3.x often uses the Reed-Solomon algorithm for erasure coding. This algorithm creates parity blocks that can be used to reconstruct the data in case of block loss. For example, in a (6,3) Reed-Solomon encoding scheme, there would be 6 data blocks and 3 parity blocks.

# Erasue Encoding in HDFS



Command  
executed to read  
file

Name Node will look at  
edit logs or fsimage

If found

Suppose in BLK 1 DN3 interacted  
first with Fs shell where encoded  
64 MB is stored then first it will  
decode 64 MB to 256 MB

Fs shell will take time  
there are performance  
issues/ even though it  
will save storage cost

# Intra-DataNode Balancer

Disk	<b>DN1</b>	<b>DN2</b>	<b>DN3</b>	<b>DN4</b>
/mnt/disk1	5BLK	5BL K	5BL K	5BL K
/mnt/disk2	5BL K	5BL K	5BL K	5BL K
/mnt/disk3	5BL K	5BL K	5BL K	5BL K

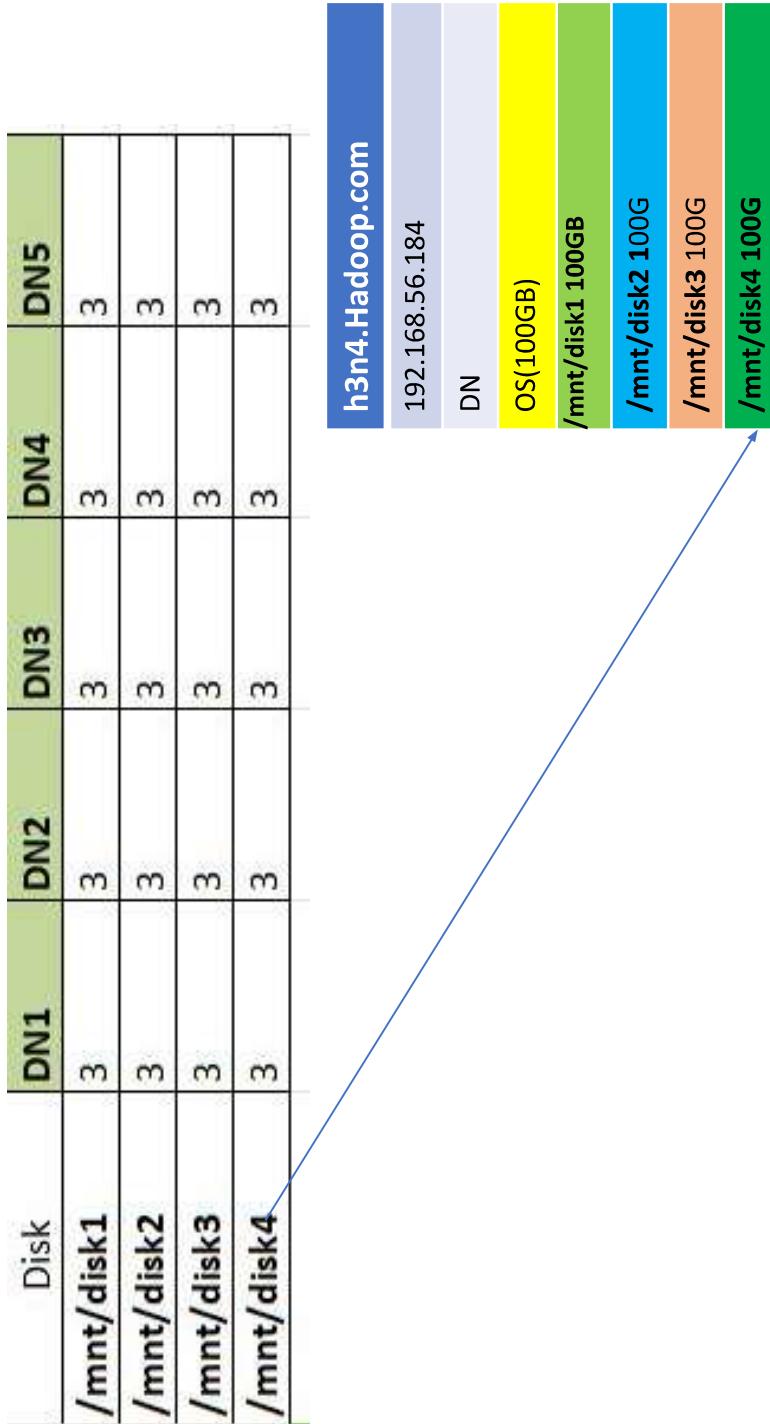
Hadoop 1.x and 2.x Add New DN and balance the data Across DNs

Disk	<b>DN1</b>	<b>DN2</b>	<b>DN3</b>	<b>DN4</b>
h3n3.Hadoop.com	192.168.56.183	192.168.56.184	192.168.56.185	192.168.56.185
DN1	OS(100GB)	OS(100GB)	OS(100GB)	OS(100GB)
DN2	/mnt/disk1 100G	/mnt/disk1 100G	/mnt/disk1 100G	/mnt/disk1 100G
DN3	/mnt/disk2 100G	/mnt/disk2 100G	/mnt/disk2 100G	/mnt/disk2 100G
	/mnt/disk3 100G	/mnt/disk3 100G	/mnt/disk3 100G	/mnt/disk3 100G

Disk	<b>DN1</b>	<b>DN2</b>	<b>DN3</b>	<b>DN4</b>	<b>DN5</b>
/mnt/disk1	4BLK	4BLK	4BLK	4BLK	4BLK
/mnt/disk2	4BLK	4BLK	4BLK	4BLK	4BLK
/mnt/disk3	4BLK	4BLK	4BLK	4BLK	4BLK

# Intra-DataNode Balancer

Hadoop 3.x Add New DN as well as Disk across data nodes- Inter data Node balancing



Most of the clusters(90%) are migrated from Hadoop 2.x to Hadoop 3.x since 2022 though Hadoop 3.x was launched in 2018 people thought of migrating in 2019 but due to covid they started migrating little late

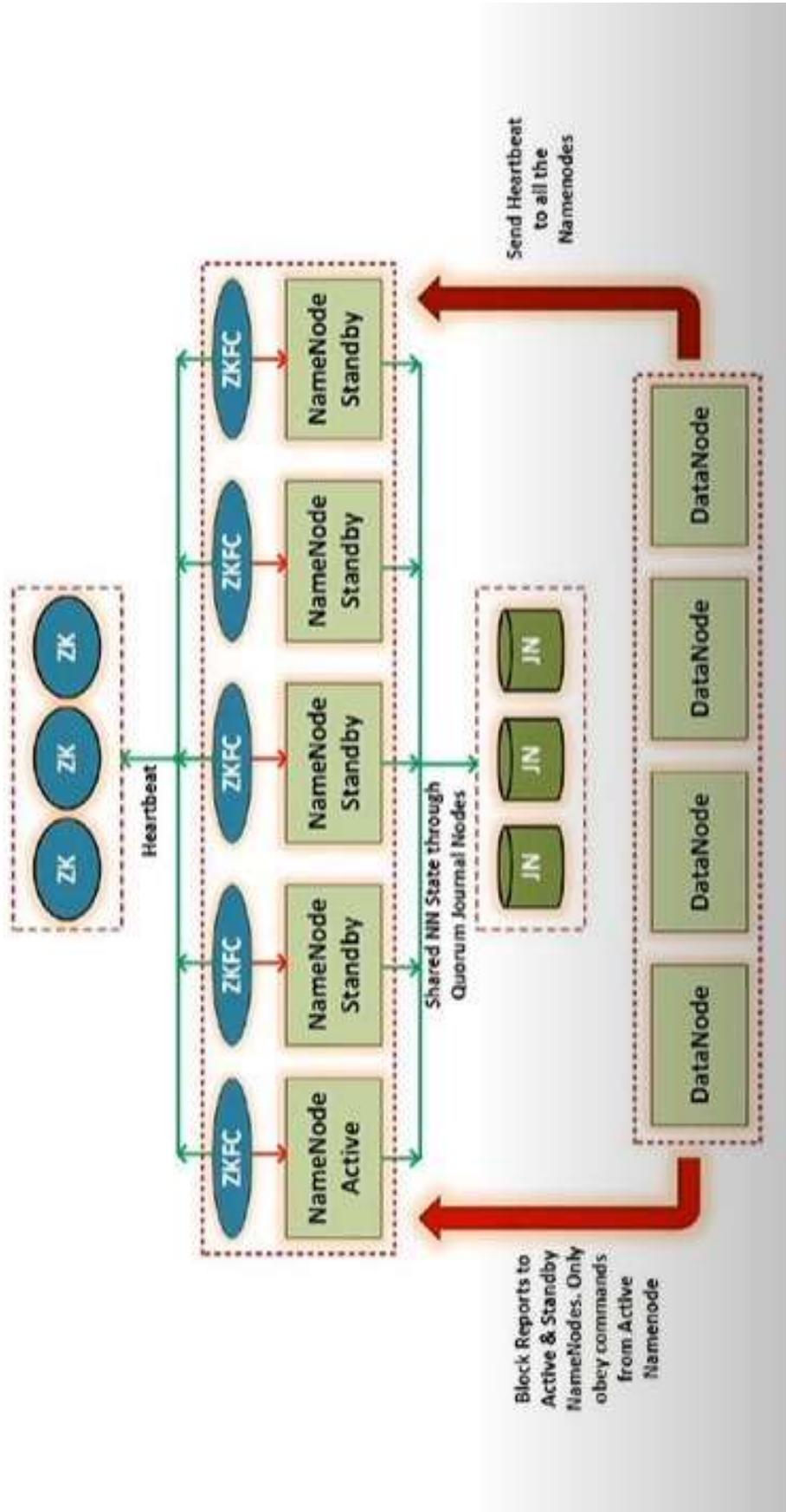
# Hadoop

- Hadoop is software-dependent because it is a software framework. It is designed to work on a cluster of machines and relies on software processes to manage its distributed storage (HDFS) and distributed processing (MapReduce). Hadoop is built and run on common software systems like Java and can be installed on various operating systems that support these software requirements.
- Hadoop is not hardware dependent— it can run on commodity hardware, which means that it does not require high-end, specialized hardware. The original intent behind Hadoop was to enable distributed processing using affordable and readily available hardware. Hadoop's architecture is designed to handle failures at the application layer, thus not relying on high-reliability or high-availability hardware features.

# Can be scaled for more than **10,000** nodes per Cluster

- Hadoop 1.x Maximum datanode you can add is 4000
- Hadoop 2.x Maximum datanode you can add is 10000
- World biggest cluster capacity is approaching 10000 nodes(linkedin), people are not even reached maximum capacity of Hadoop.
- Hadoop 3.x you can go up to 25000 data nodes

## Hadoop 3.X Architecture



# Advantages of NameNode HA with Journal Nodes & ZooKeeper

- ✓ Multiple copies of Edits for both the NameNodes by using JNs
- ✓ Automatic failover of Active and Standby NameNodes by using ZooKeeper Failover
  - The transition from NameNode's active state to the standby is managed by a new entity in the system called the Failover Controller.
  - Each NameNode runs with a lightweight ZooKeeper Failover Controller (ZKFailoverController / ZKFC), whose job is to monitor its NameNode for failures (using a simple heartbeat mechanism) and trigger a failover.
- Graceful failover: Failover may also be initiated manually by an administrator (Example: In case of routine maintenance).

# FailOver

## Types of Failover

Types of Failovers:

- **Graceful**
  - Initiated by the administrator
  - Mostly done in case of a routine maintenance
  - Failover Controller arranges an orderly transition
- **UnGraceful**
  - Not possible to make sure if NodeNode has failed
  - Slow network can trigger a Failover transition

	ZKFC1	ZK2	ZK3
Scenario 1	NN1: Active NN2: Standby	NN1 is down/Stopped ZKFC1 is running	ZKFC1 will inform Zks Zks will inform ZKFC2 ZKFC2 will make NN2 active
Scenario 2	NN1: Standby NN2: Active	NN2 is down/Stopped ZKFC2 is running	ZKFC1 will inform Zks Zks will inform ZKFC1 ZKFC2 will make NN1 active
Scenario 3	NN1: Active NN2: Standby	NN1running ZKFC1 is down	ZKFC1 will stop sending heartbeat Zks will inform ZKFC2 ZKFC2 will make NN2 Active
Scenario 4	NN1: Standby NN2: Active	NN2 is running ZKFC2 is down/stopped	ZKFC2 will stop sending heartbeat Zks will inform ZKFC1 ZKFC1 will make NN1 Active
Scenario 5	NN1: Active NN2 : Satanby	NN1 is down/Stopped ZKFC1 is down/stopped	ZKFC1 will stop sending heartbeat Zks will inform ZKFC2 ZKFC2 will make NN2 Active
Scenario 6	NN1 : Active	NN2 is down/Stopped ZKFC2 is down/stopped	ZKFC2 will stop sending heartbeat Zks will inform ZKFC1 ZKFC1 will make NN1 Active

# Fencing

- Fencing: In case of an ungraceful failover, however, it is impossible to be sure that the failed NameNode has stopped running (Example: A slow network or a network partition can trigger a failover transition).
  - In this case, the previously active NameNode will be still in Active state and try to write the Edits in Journal Nodes.
  - With the help of Fencing mechanism, HA implementation will ensure that the previously active NameNode is prevented from doing any damage and causing corruption.
- **SSH Fencing** is a method to kill the previously active NameNode's state using SSH Private Key. This requires a passwordless ssh access between both the NameNodes.
- **SHELL Fencing** is a method to kill the previously active NameNode's state using script. This will be used where the primary NameNode machine is inaccessible and the ssh method is failed.

# What if we make both the NameNodes ACTIVE?

- If both the NameNodes are active, they think that they are in control and both try to write to the same Edit Log. There are chances of corruption of metadata in this case.
- This is like “split-brain” syndrome issue.
- Fencing in ZKFC will take care of this issue by killing one of the NameNode's Active state.

Command	Description	Description
ls	Directory Listing	Output the first 10 lines of file
cd dir	Change directory to dir	
pwd	Show current working directory	
mkdir dir	Create a directory dir	
rm file	Delete file	
rm -r dir	Delete directory dir	
rm -f file	Delete file	
rm -rf dir	Force remove directory dir	
cp file1 file2	Copy file1 to file2	
cp -r dir1 dir2	Copy dir1 to dir2; Create dir2 if it doesn't exist	Change the permissions of file to octal, which can be found separately for user, group, and world by adding: 4 – read (r), 2 – write (w), 1 – execute (x)
scp	Copy from one server to another server	chmod octal file
mv file1 file2	Rename or Move file1 to file2	chmod 777
ln -s file link	Create symbolic link to file	Read, Write, Execute for all
cat file	Print file content	Change owner for the particular file
more file	Output the contents of file by page wise	Change owner for all files in the dir

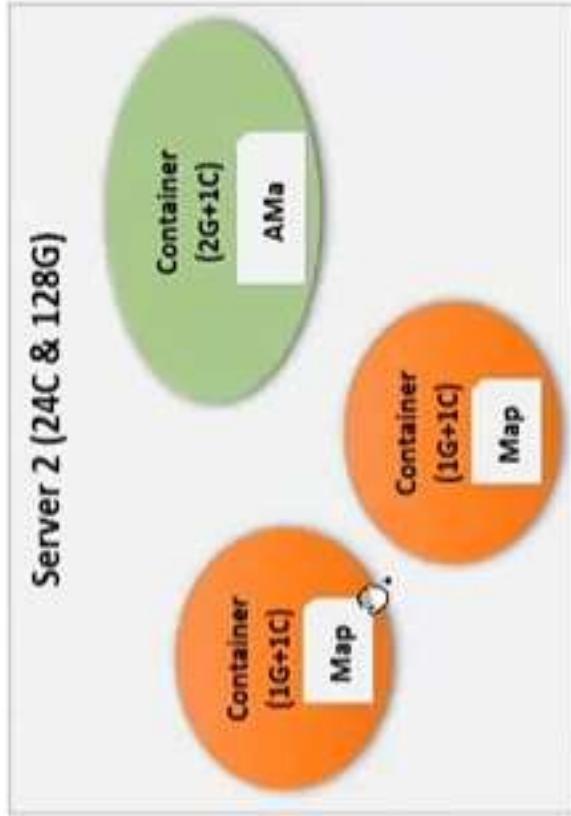
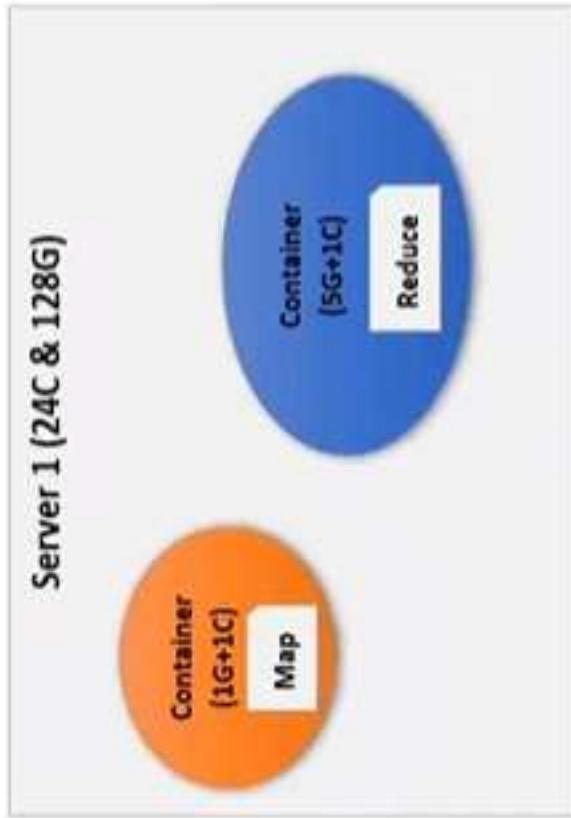
Command	Description
ls	Directory Listing
cd dir	Change directory to dir
pwd	Show current working directory
mkdir dir	Create a directory dir
rm file	Delete file
rm -r dir	Delete directory dir
rm -f file	Delete file
rm -rf dir	Force remove directory dir
cp file1 file2	Copy file1 to file2
cp -r dir1 dir2	Copy dir1 to dir2; Create dir2 if it doesn't exist
scp	Copy from one server to another server
mv file1 file2	Rename or Move file1 to file2
ln -s file link	Create symbolic link to file
cat file	Print file content
more file	Output the contents of file by page wise

# Commands

- # Lists all files and directories in the root of the HDFS
  - hdfs dfs -ls
- # Downloads a file from the given URL to the local file system of the machine where the command is executed
  - wget <https://www.stats.govt.nz/assets/Uploads/Annual-enterprise-survey/Annual-enterprise-survey-2020-financial-year-provisional-csv.csv>
- # Creates a new directory named 'practice' in the HDFS under the current user's home directory
  - hadoop fs -mkdir practice
- # Lists all files and directories in the current local directory of the Linux shell (not HDFS)
  - ls
- # Renames the file 'annual-enterprise-survey-2020-financial-year-provisional-csv.csv' to 'sample.csv' on the local file system
  - mv annual-enterprise-survey-2020-financial-year-provisional-csv.csv sample.csv
- # Copies the file 'sample.csv' from the local file system to the 'practice' directory in HDFS
  - hadoop fs -copyFromLocal sample.csv practice/sample.csv

## What is a Container?

On a specific server/host, Container is a logical construct that represents a specific amount of CPU and RAM to run the tasks from your job/application



## Resource Manager (RM)

- *Master Service in YARN Framework*
  - First point of contact for all processing layer requests
  - Keeps track of all NodeManager(s) using simple heartbeat mechanism
  - Maintains *Cluster State Store* to understand the total resources, allocated resources & free resources (COREs & RAM) that each running NodeManager has
  - Maintains *Application State Store* to understand the list of applications submitted, submission context, application-attempt level (AMA) information and credentials about each application
  - Instructs NodeManager(s) to launch Application Master container when a new application is submitted
  - Handles Application Masters' requests either to launch new containers (to run map/reduce tasks) or to release existing containers and instructs NodeManager(s) on the same
  - Determines how the resources are allocated with the help of YARN Schedulers
  - Manages Cluster level security (who can submit/kill the jobs) with the help of ACLs

RM performs multiple activities based on its role:

- App Manager (AMg) – Manage Application Masters across the Cluster
- Job Scheduler – Responsible for allocating containers to the submitted jobs

### Application Master (AMa)

- One Application Master for each job/application
- Kind of a leader container in the application
- Responsible for analyzing the application requirements & creating Job configuration XML file (contains properties used in job submission/execution)
- Responsible for negotiating the containers with RM and allocating the tasks in each container
- Monitors the progress of the tasks and tracks the status of the application in Job history JSON file (contains job, task and attempt events and their stats)
- Responsible for end to end execution of application

### Node Manager (NM)

- Slave Service in YARN Framework and runs on each DataNode
- Provides information to RM through heartbeat on total / allocated / available resources (CPU & RAM) and status of running containers
- Takes care of launching/releasing/execution of containers based on the instructions from RM
- Kills runaway containers when required