Fundamental of Big Data Analytics

lec 02

Today Content Cover

- Traditional Storage System
- Distributed Storage System
- Big Data Case study
- Hadoop

Centralized System

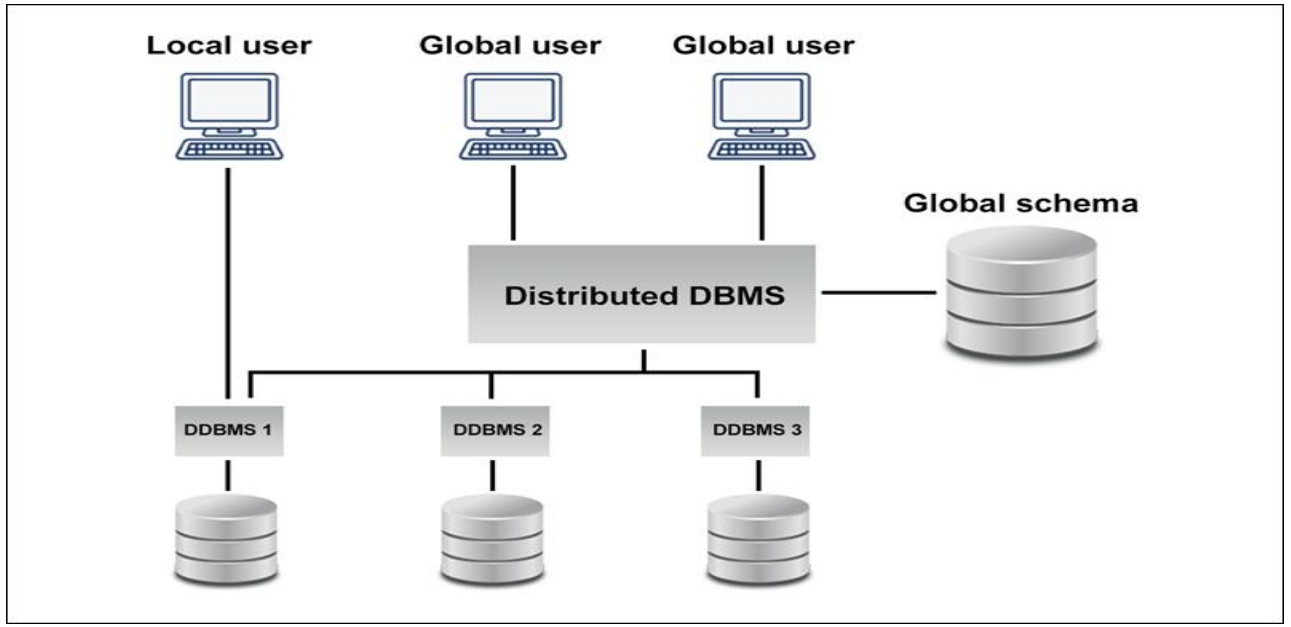
- A centralized system is a database that is located, stored and managed in one place.
- used the **client-server architecture** where one or more client nodes are directly connected to a central server.
- most commonly used in most organizations, where clients send request to corporate servers and receive a response back from them.
- **Example:** in the Wikipedia search bar you search “Big Data”, the client sends a request to the Wikipedia server and displays the relative articles.
  - A desktop or server CPU
  - A mainframe computer.

Disadvantage of Centralized System

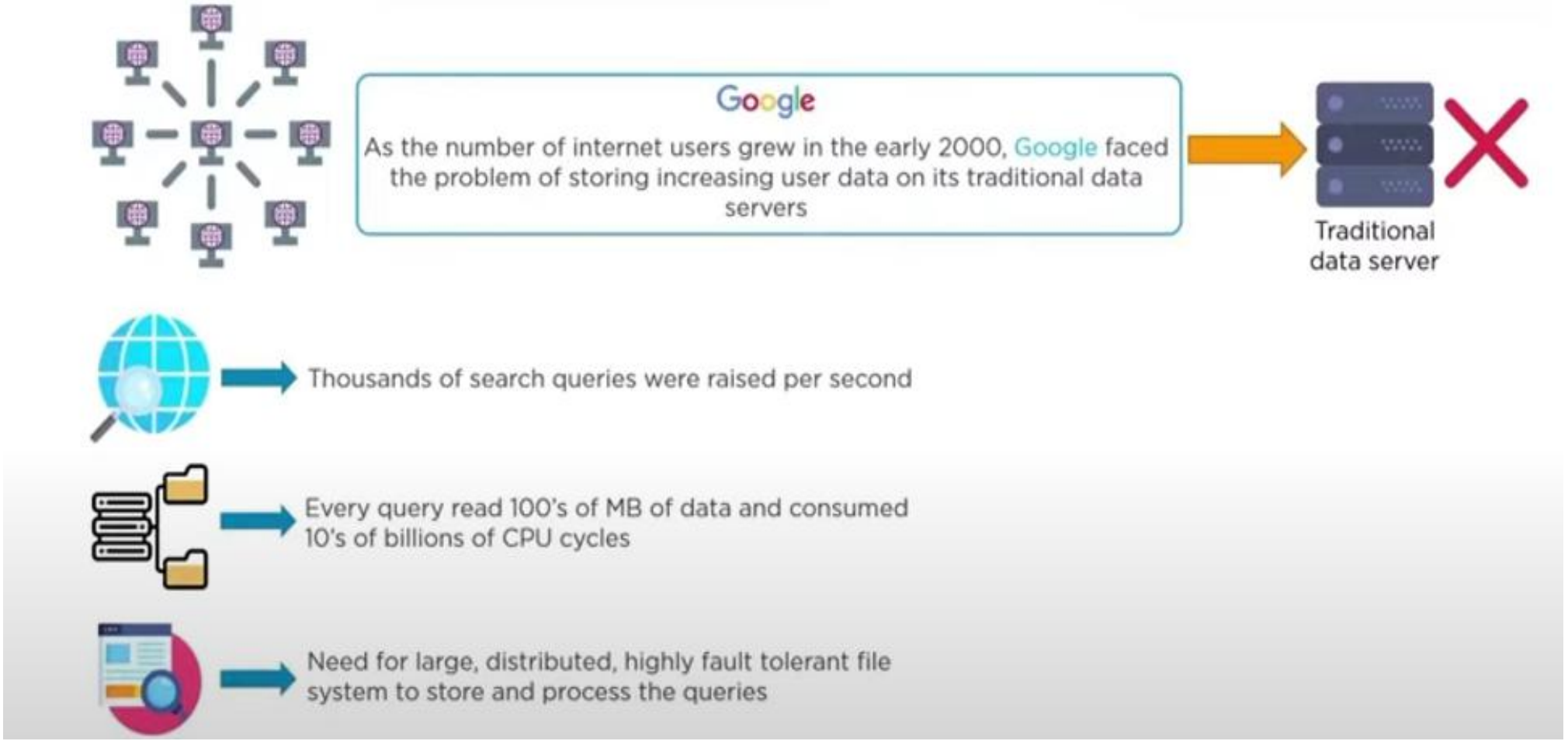
- Data searching takes time
- In case of failure of a centralized server, the whole database will be lost.
- If multiple users try to access the data at the same time then it may create issues.

Distributed Storage System

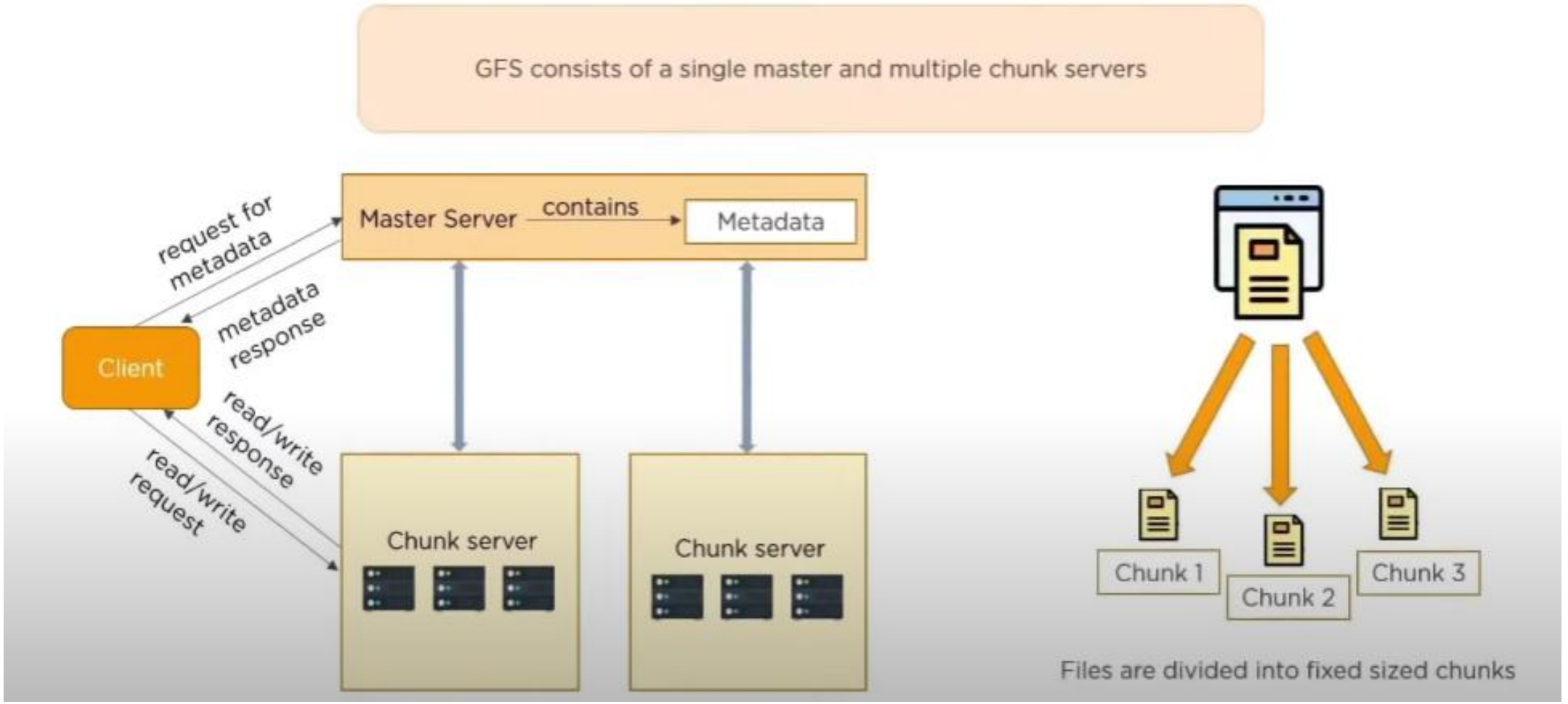
- A distributed database is basically a database that is not limited to one system, it is spread over different sites.

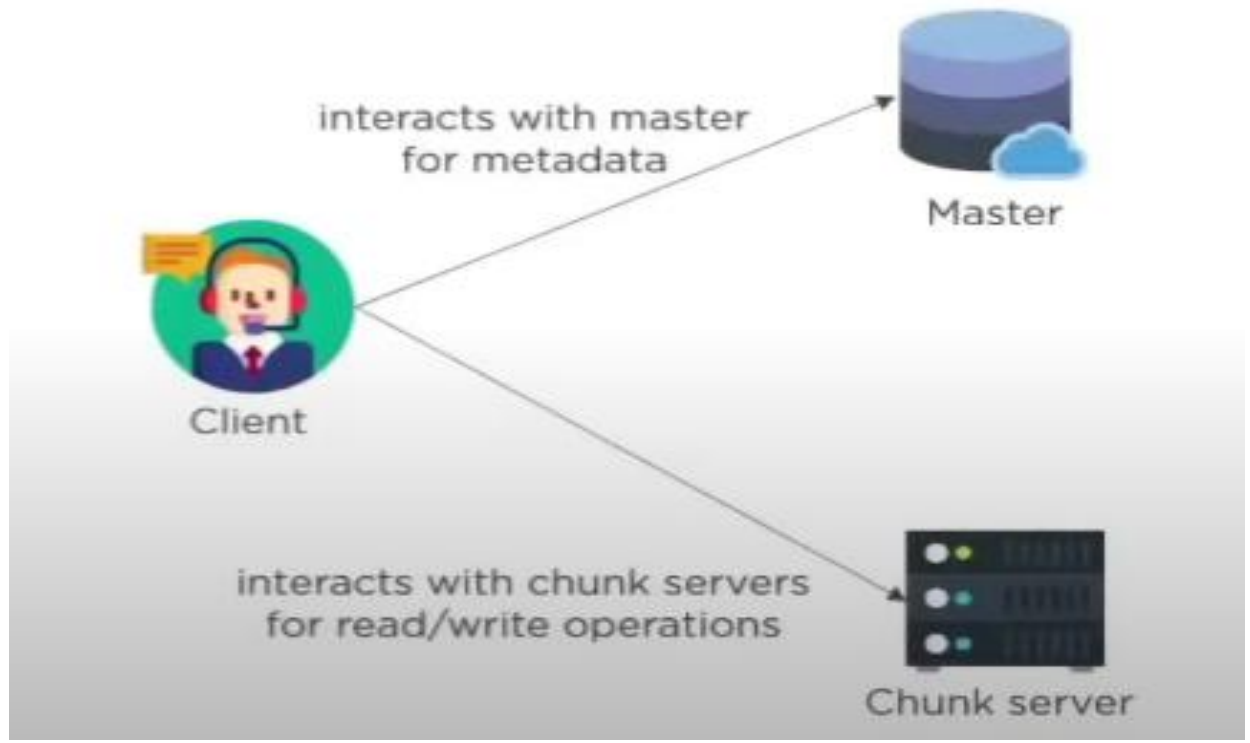


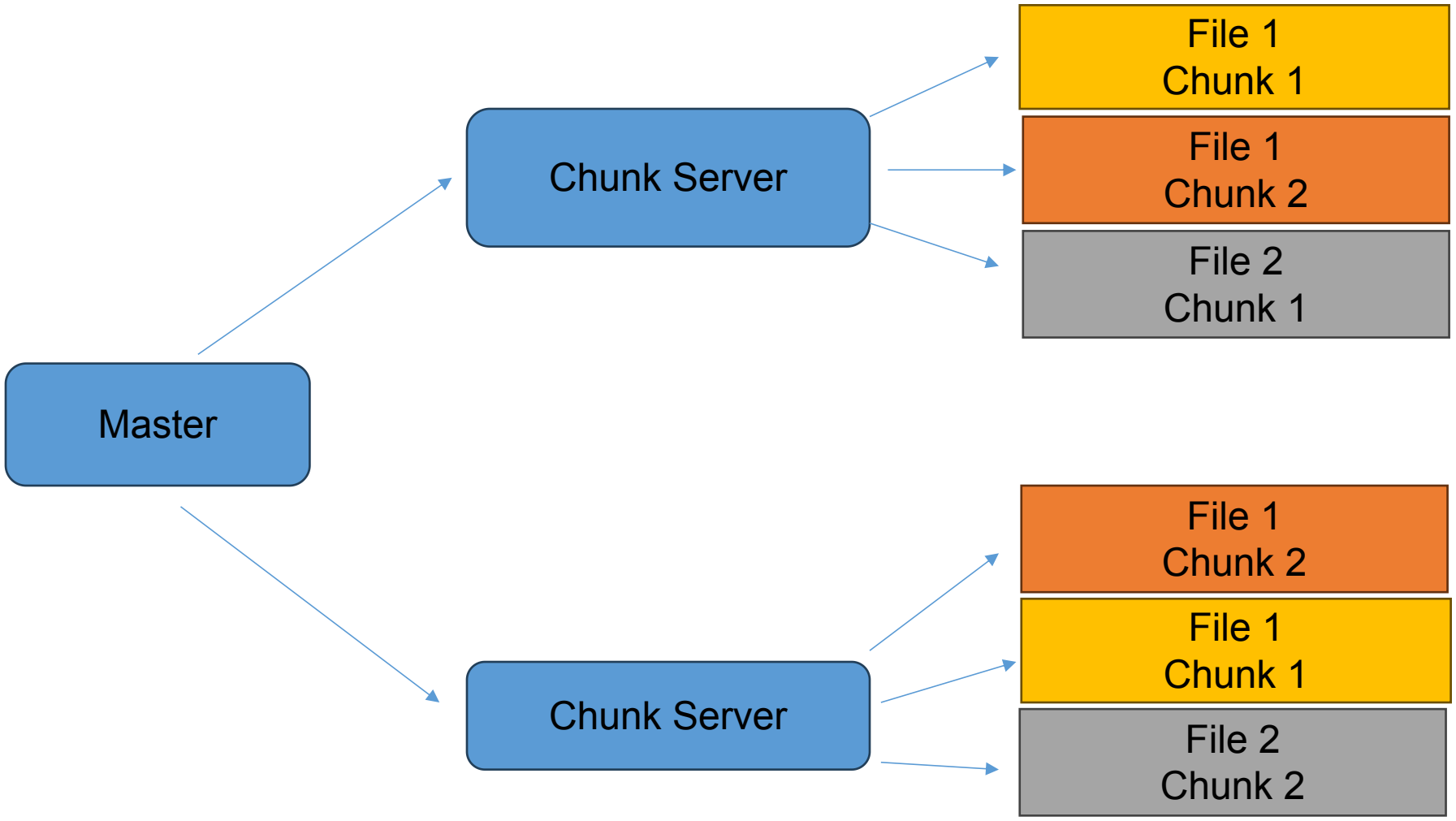
Big Data Case Study



Google file System




interacts with master
for metadata
Master
Client
interacts with chunk servers
for read/write operations
Chunk server


Master
Chunk Server
Chunk Server
File 1
Chunk 1
File 1
Chunk 2
File 2
Chunk 1
File 1
Chunk 2
File 1
Chunk 1
File 2
Chunk 2

Challenging of Big Data

- Storing huge amounts of data.
  - Enormous amount of data generated each day
  - Unstructured data cannot be stored in a traditional database.
- Processing and analysis of massive data
  - Used big data to achieve business goals.
  - Processing and extracting insights from big data takes time.
- Securing data

Hadoop as a Solution.

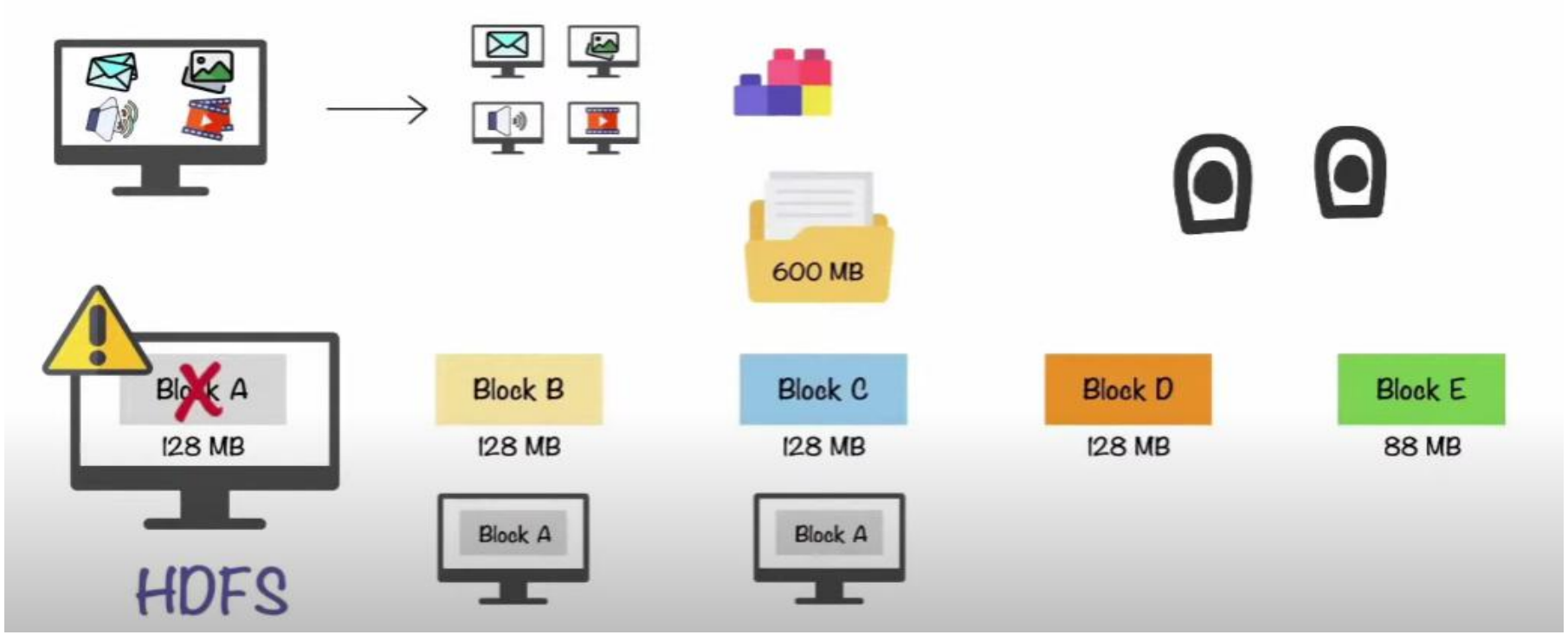
Hadoop

- Hadoop consists of three components that are specifically designed to work on Big Data.
  - Storage Unit (HDFS)
  - Process data (MapReduce)
  - YARN

Hadoop

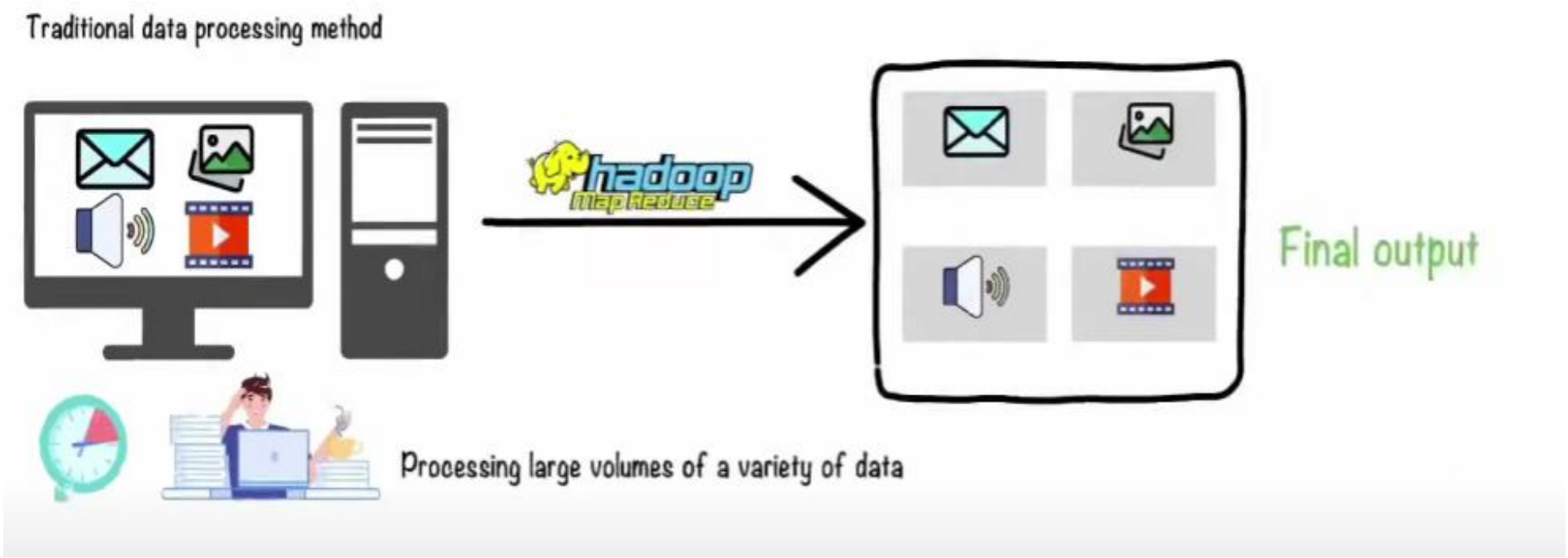
- **Apache HDFS** or **Hadoop Distributed File System** is a block-structured file system where each file is divided into blocks of a pre-determined size.
- These blocks are stored across a cluster of one or several machines.
- Apache Hadoop HDFS Architecture follows a ***Master/Slave Architecture***
  - cluster comprises of a single **NameNode (Master node)**
  - all the other nodes are **DataNodes (Slave nodes)**

HDFS

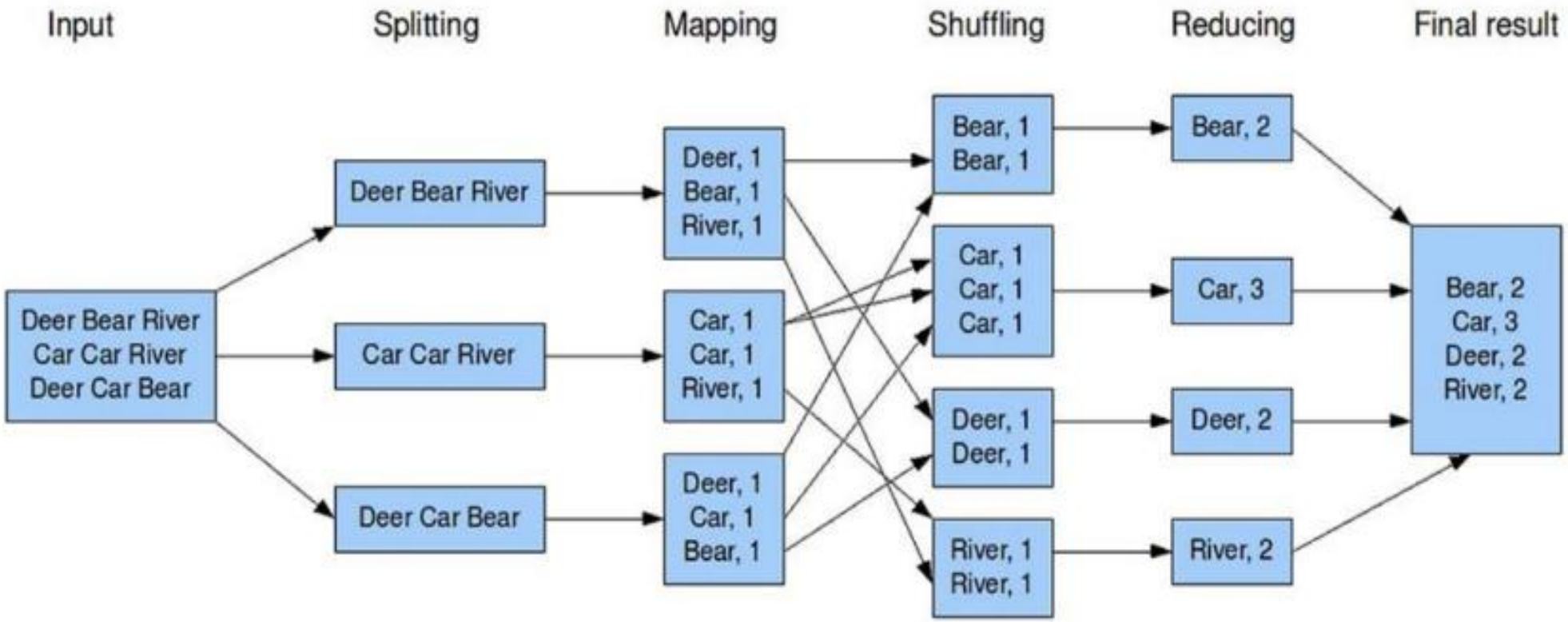


- HDFS makes copies of all data and stores it across multiple systems
- Data is not lost at any cost, even if one DataNode crashes, making HDFS fault tolerant.

MapReduce


Traditional data processing method
hadoop
MapReduce
Final output
Processing large volumes of a variety of data

Example: Word Count

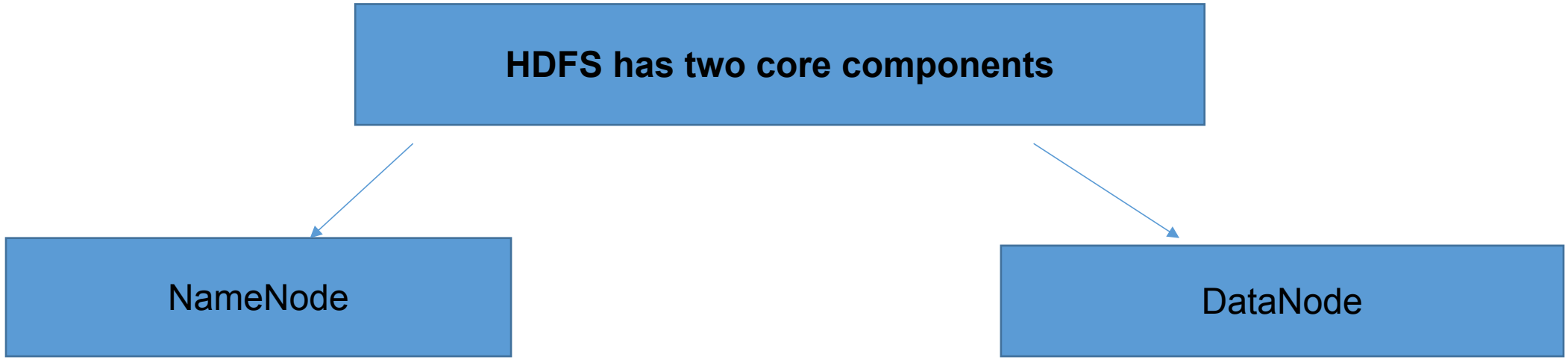


YARN


hadoop
hadoop
YARN
Resource manager
Node manager
Application master
Containers

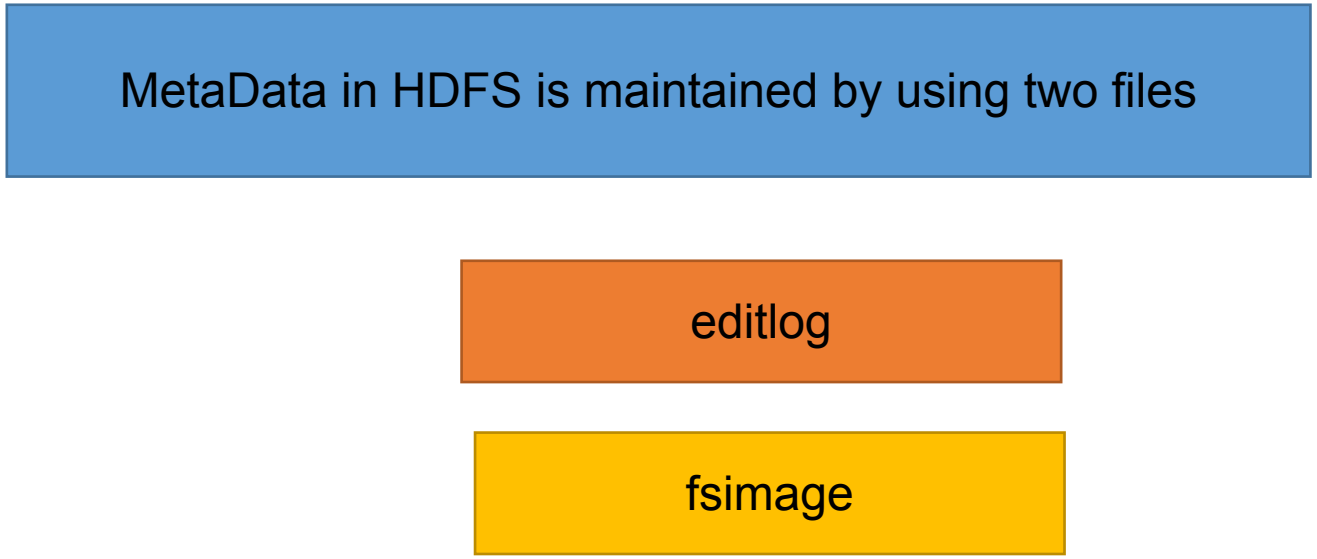
HDFS

- To store the massive data, data was divides and distributed among many individual databases.
- HDFS is a specially designed file system for storing huge dataset in commodity hardware.



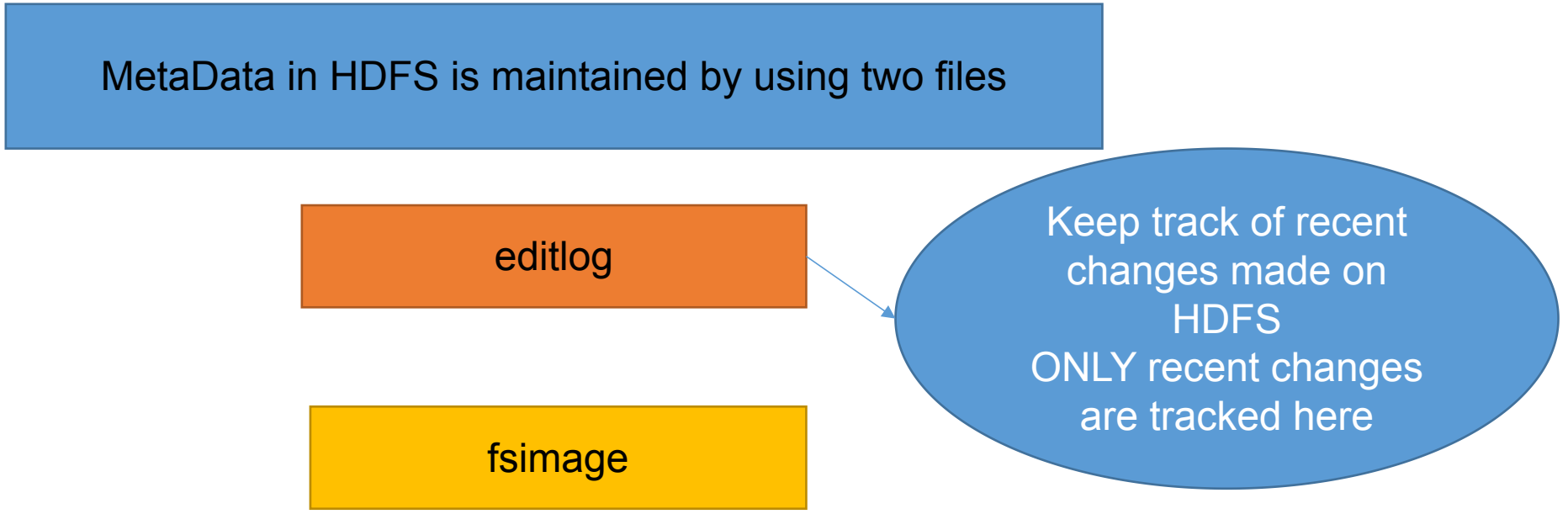
NameNode

- Is the master daemon.
- Only one active NameNode.
- Manages the DataNodes.
- Store all the metadata. MetaData gives information regarding the file location,block size and so on.



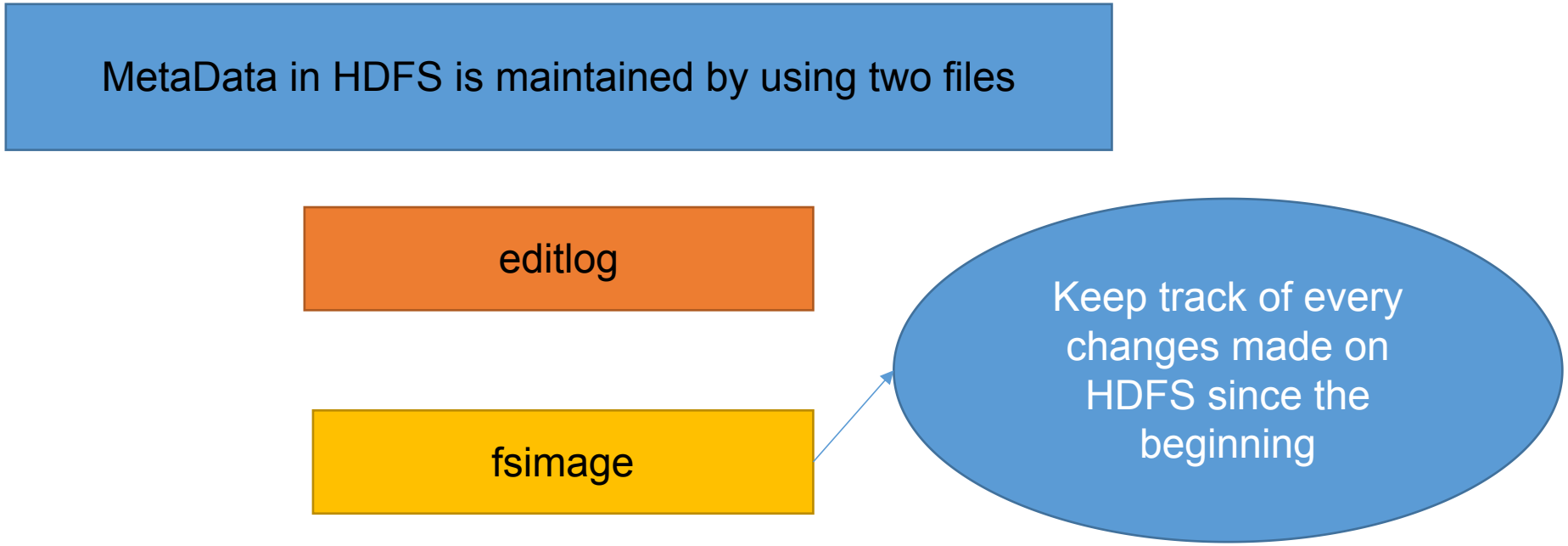
NameNode

- Is the master daemon.
- Only one active NameNode.
- Manages the DataNodes.
- Store all the metadata. MetaData gives information regarding the file location,block size and so on.



NameNode

- Is the master daemon.
- Only one active NameNode.
- Manages the DataNodes.
- Store all the metadata. MetaData gives information regarding the file location,block size and so on.



Now what happen when

- The editlog file size increases?
- The NameNode fails?

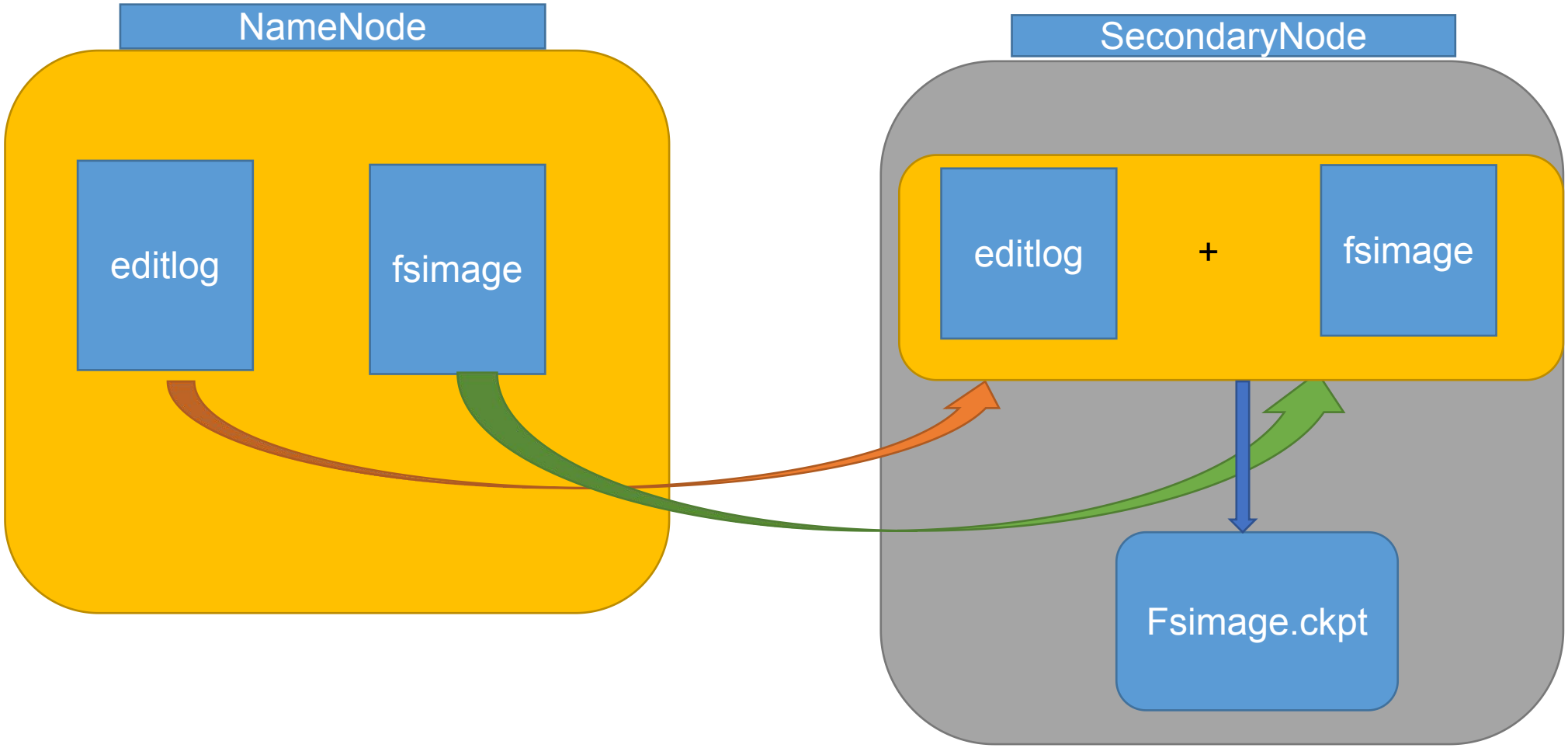
Now what happen when

- The editlog file size increases?
- The NameNode fails?

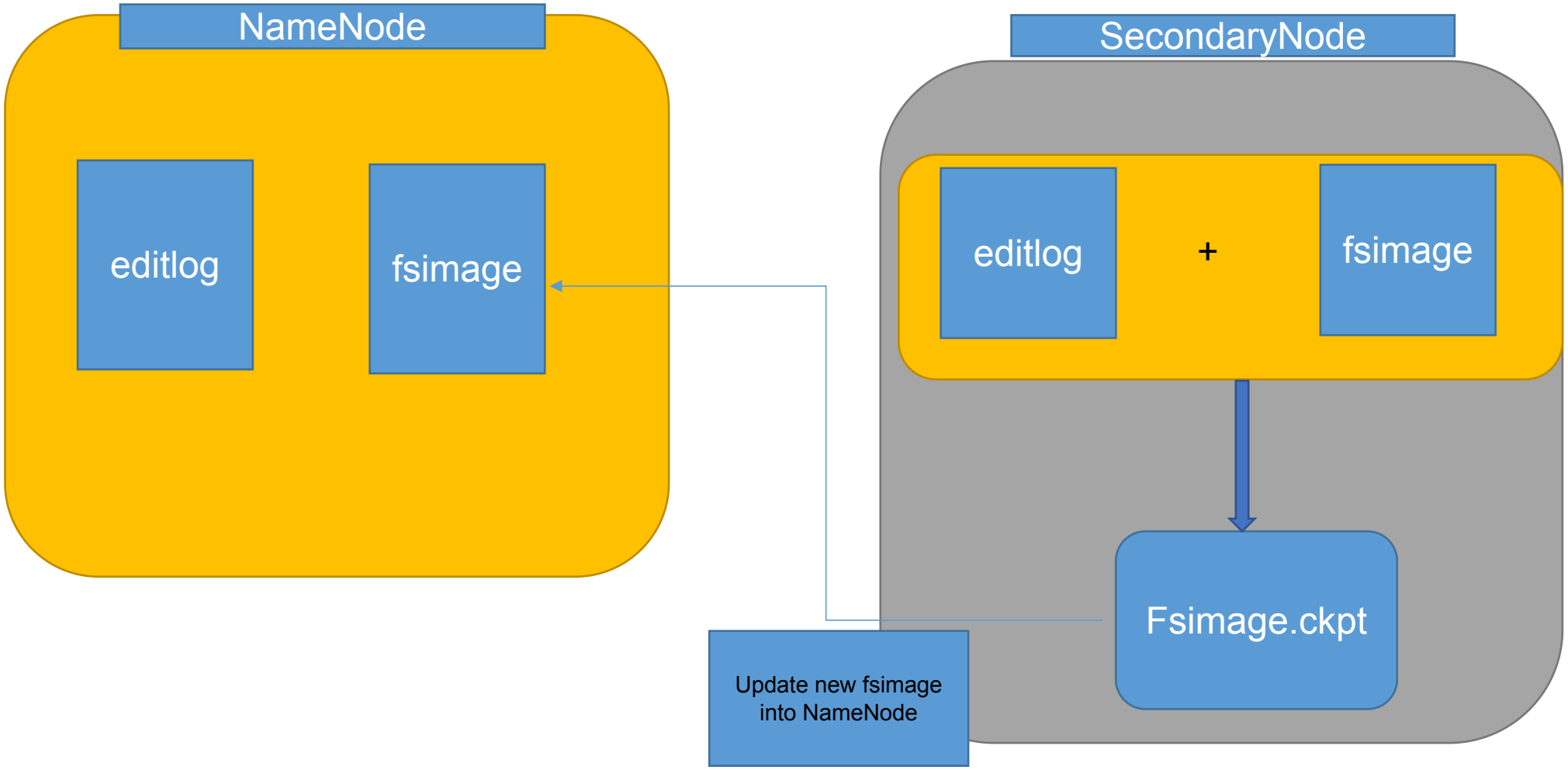
Solution

Make copies of the Editlog and Fsimage file.

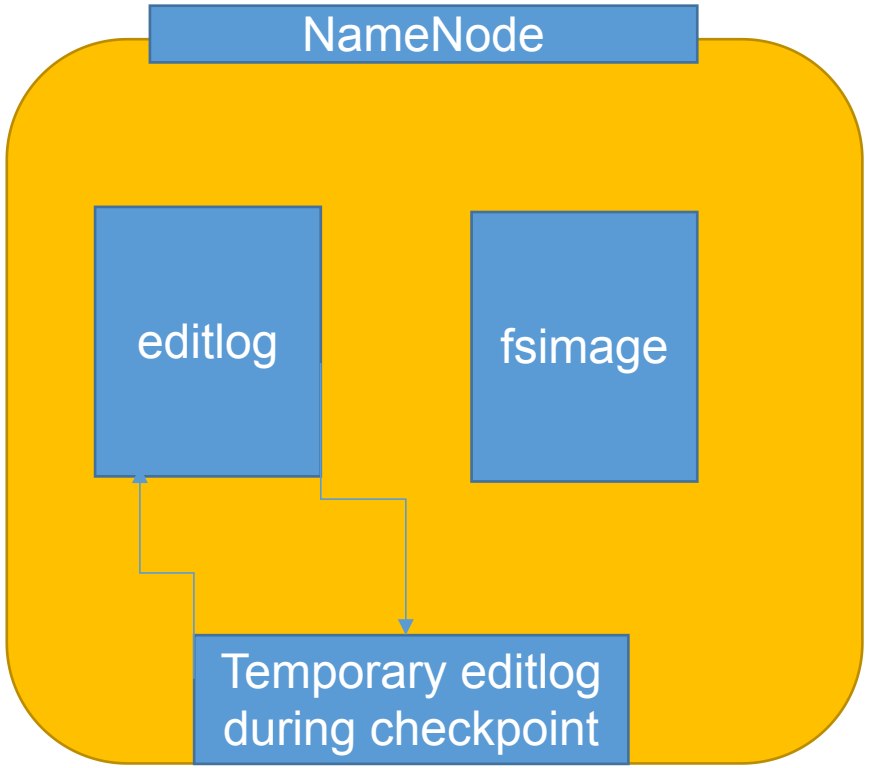
Secondary NameNode is a node that maintain the copies of editlog and fsimage.
It combine them both to get an updated version of the fsimage.

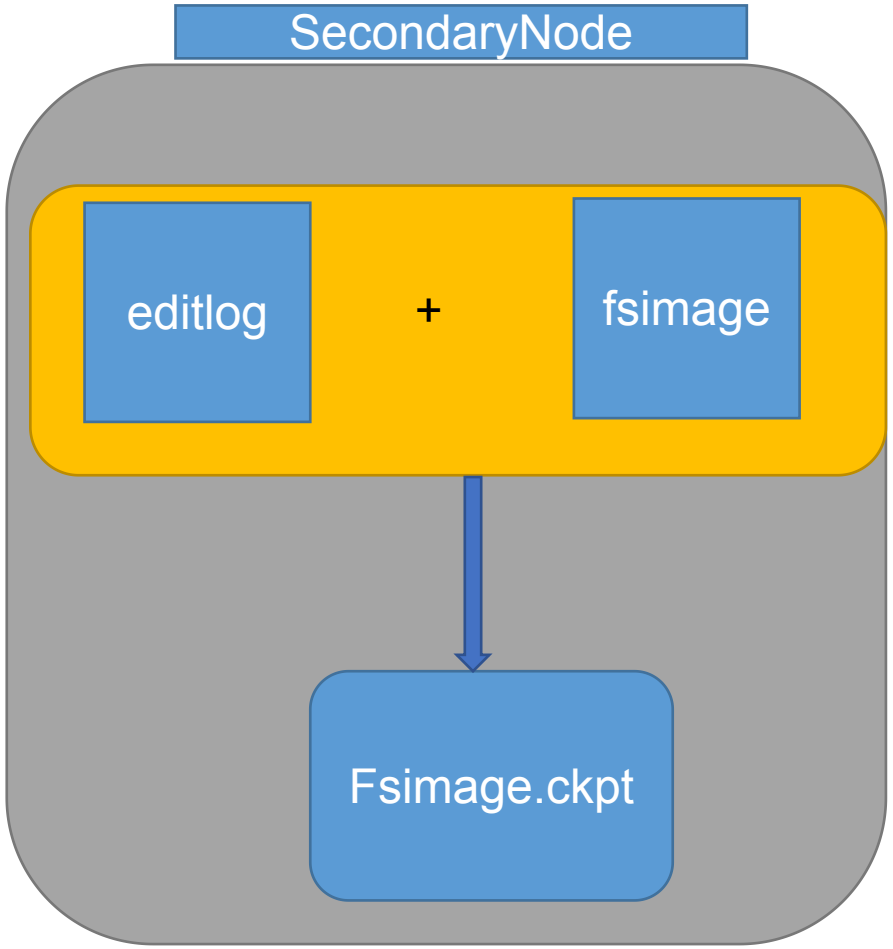


Secondary NameNode is a node that maintain the copies of editlog and fsimage. It combine them both to get an updated version of the fsimage.



Secondary NameNode is a node that maintain the copies of editlog and fsimage.
It combine them both to get an updated version of the fsimage.

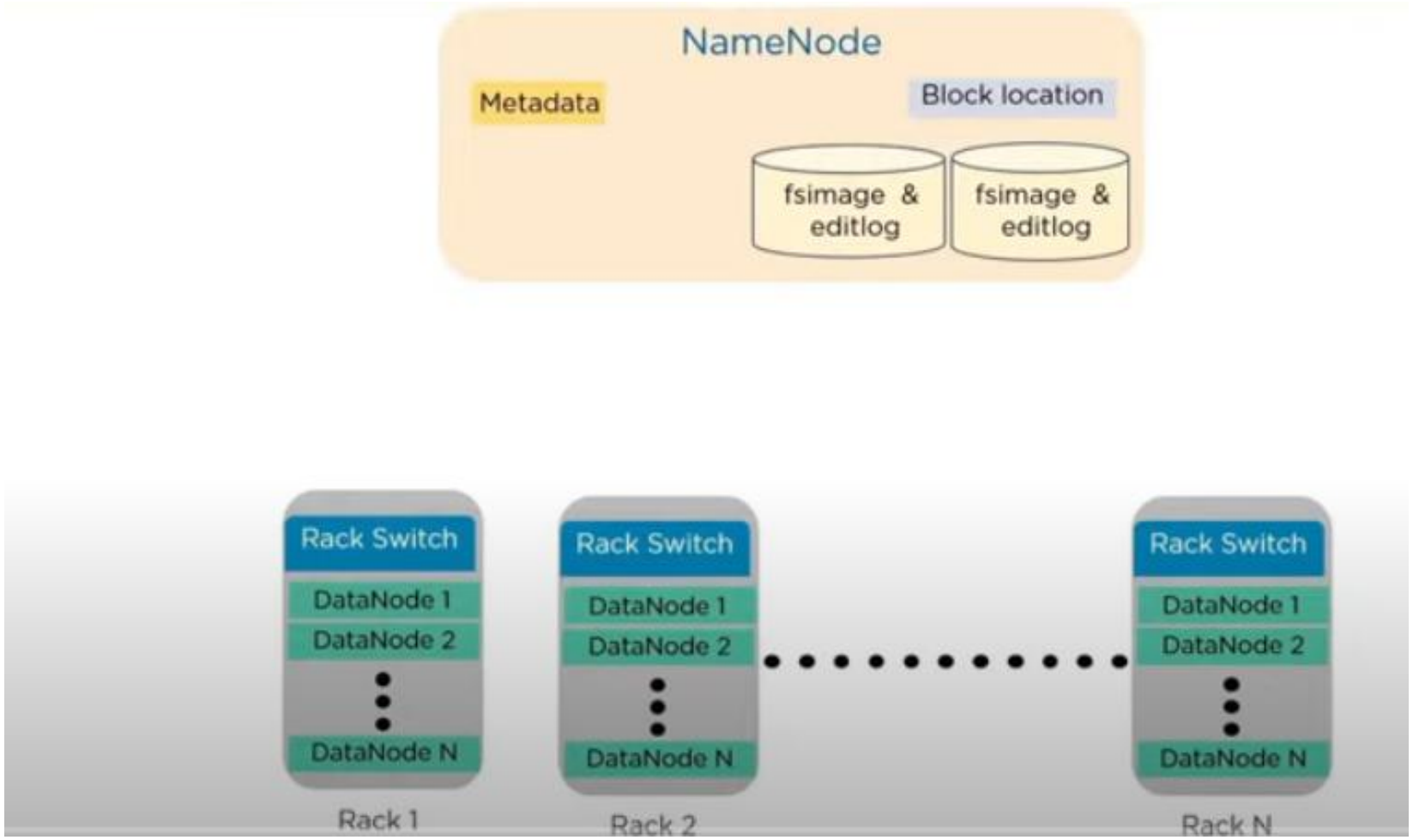




DataNode

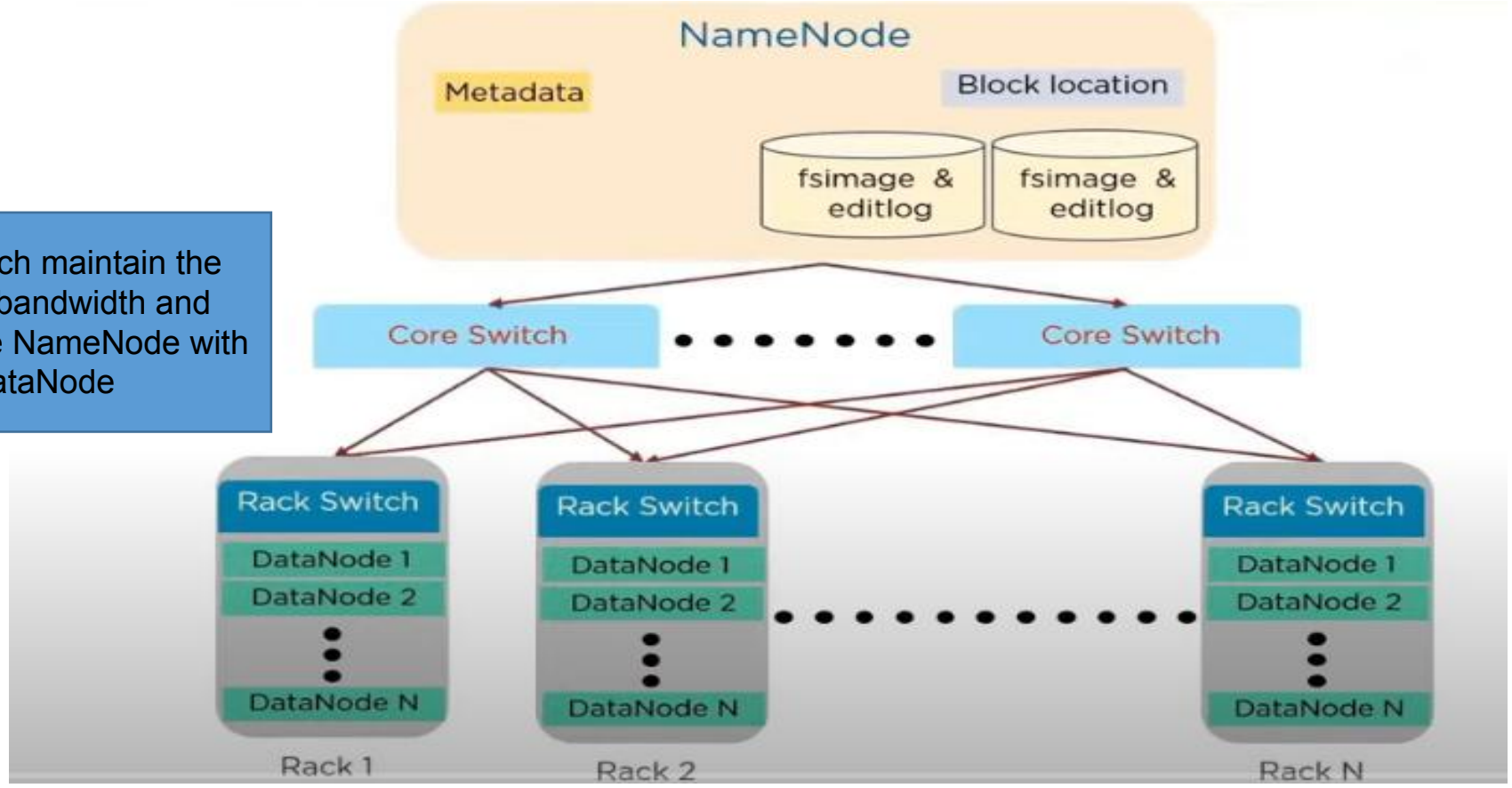
- DataNode is the slave daemon.
- There can be multiple DataNodes.
- Stores the actual data.

HDFS Cluster architecture



HDFS Cluster architecture

Core switch maintain the network bandwidth and connect the NameNode with DataNode



HDFS Data Blocks

- HDFS splits massive files into small chunks, these chunks are called **data blocks**.
- Each file in HDFS is stored as Data blocks.
  - Default size of each block is 128MB.

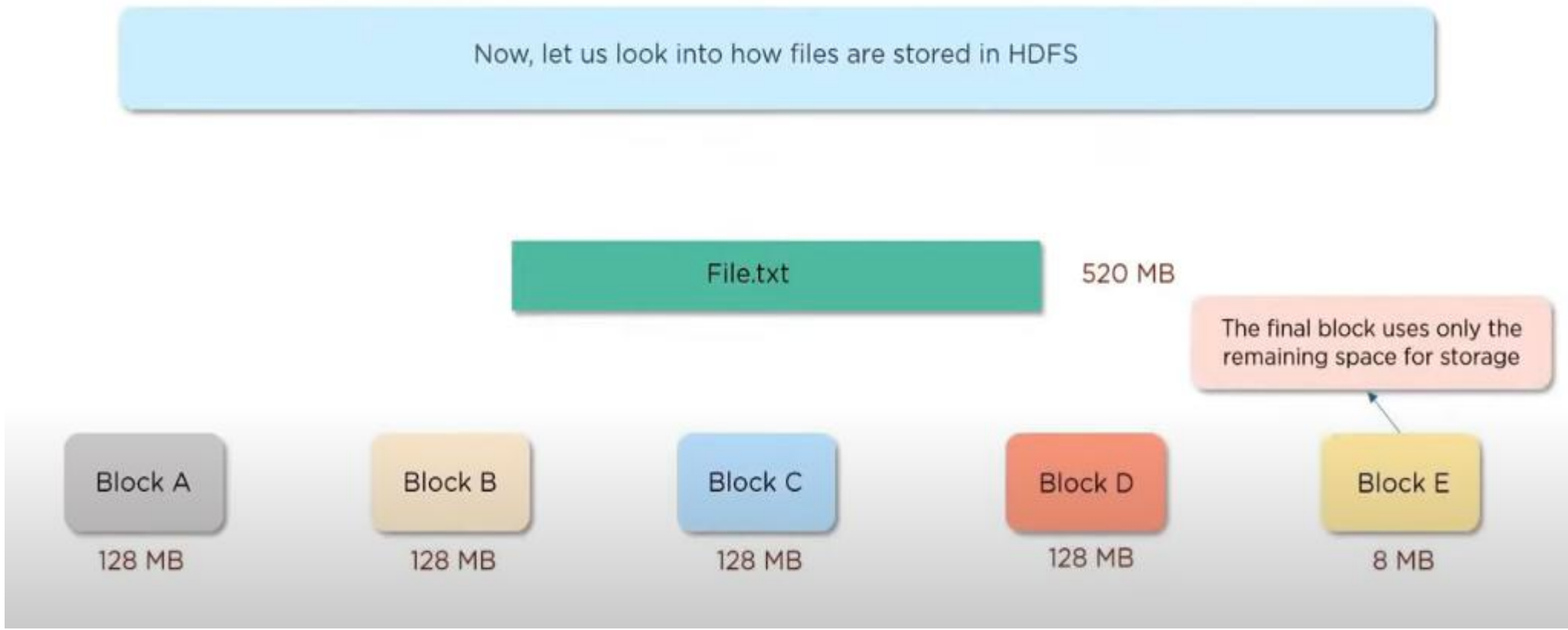
WHY 128MB?

If the block size is smaller, then there will be too many data blocks along with lots of metadata which will create overhead.

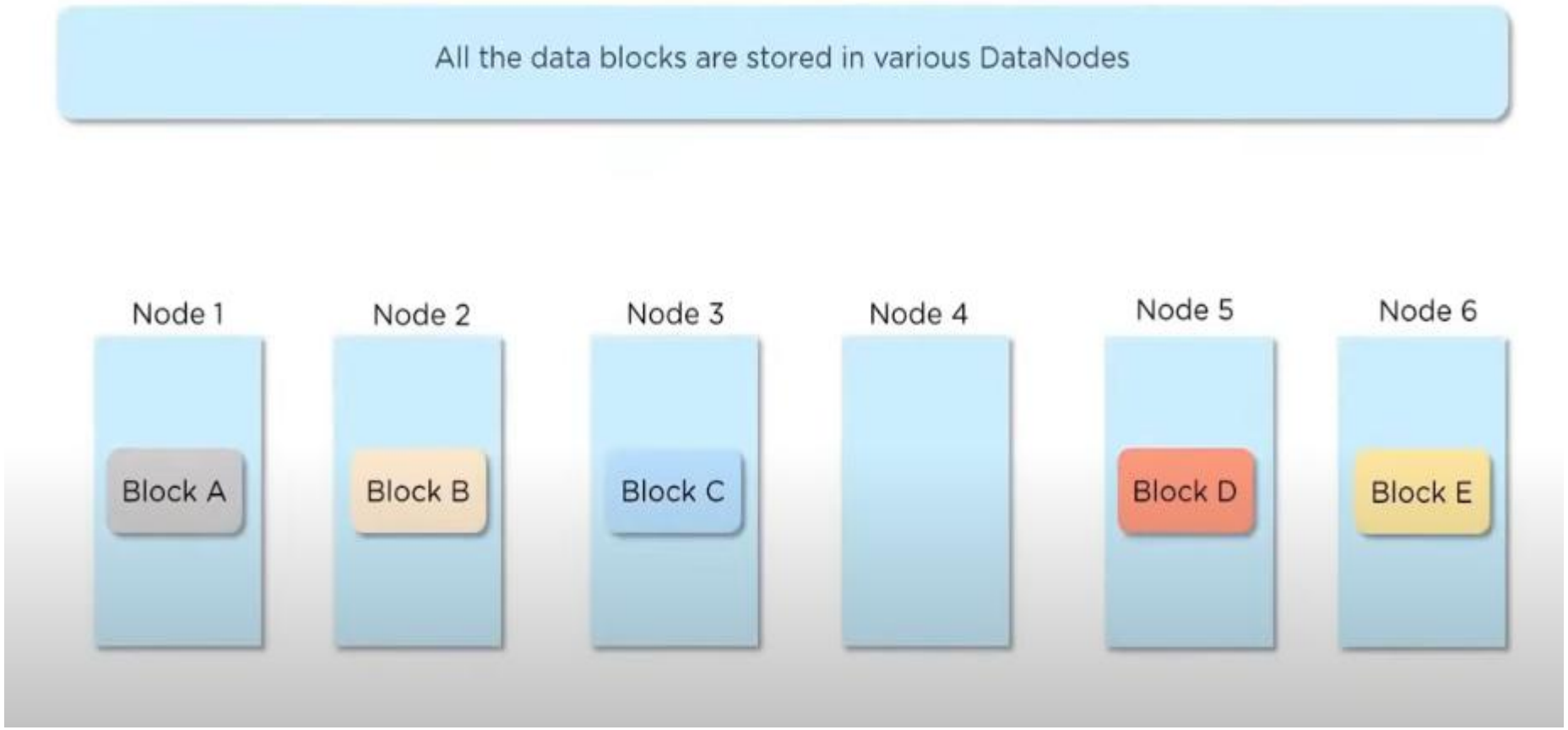
Similarly, if the block size is very large then the processing time for each block increases.

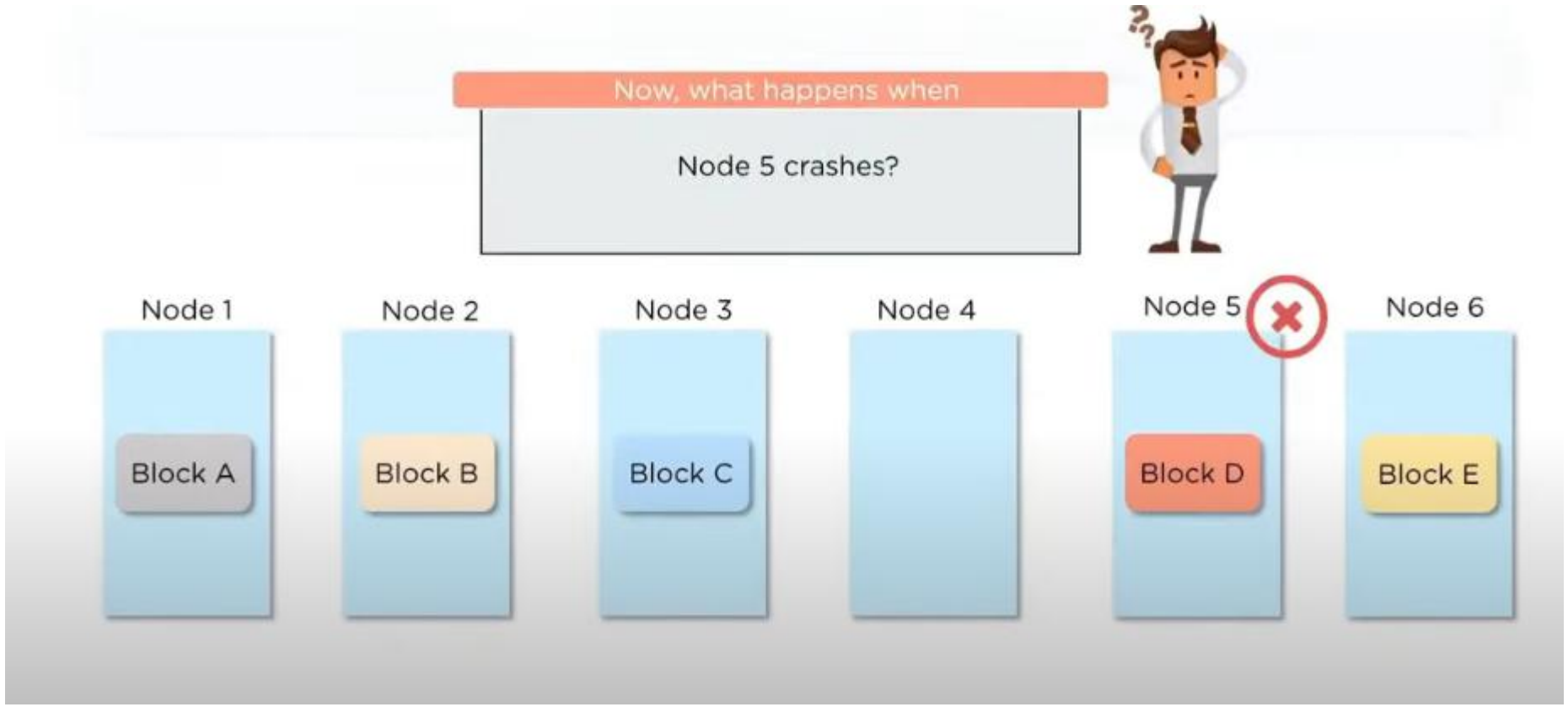
  - All block have same size but may be last data node is smaller or equal size.

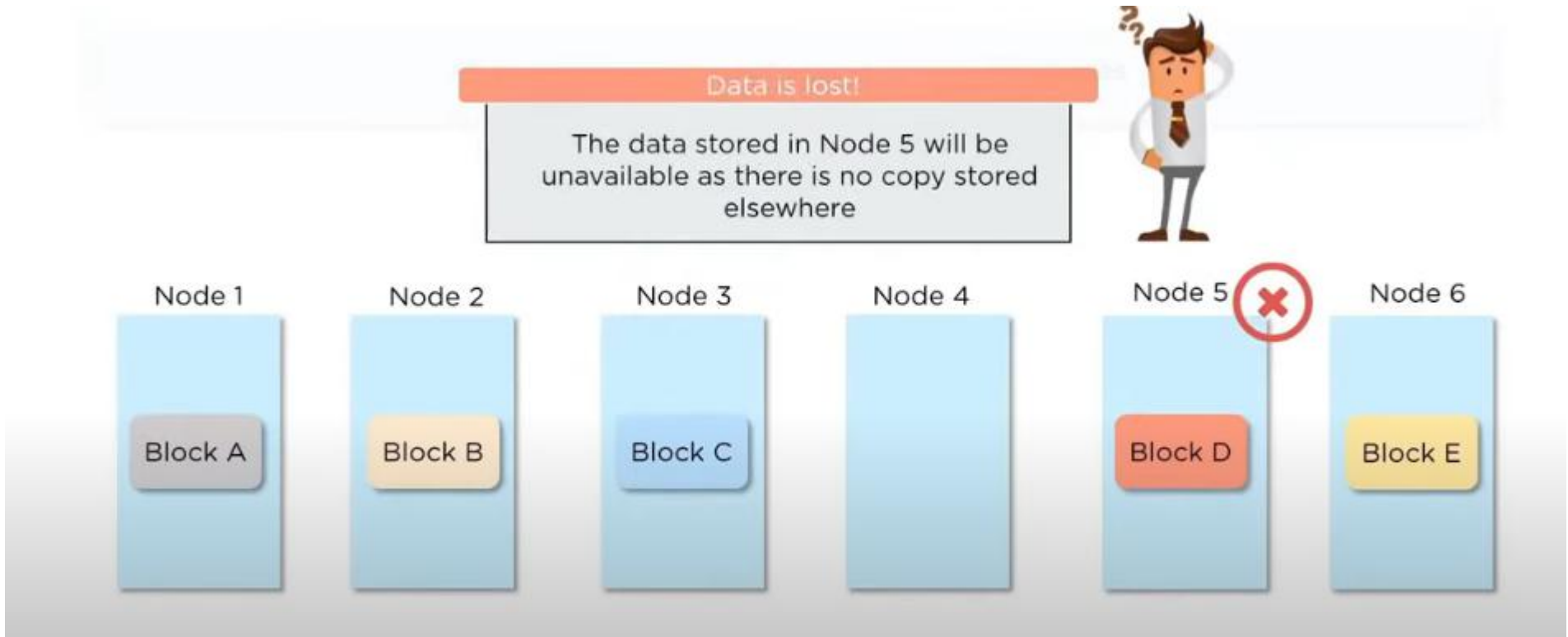
HDFS Data Blocks



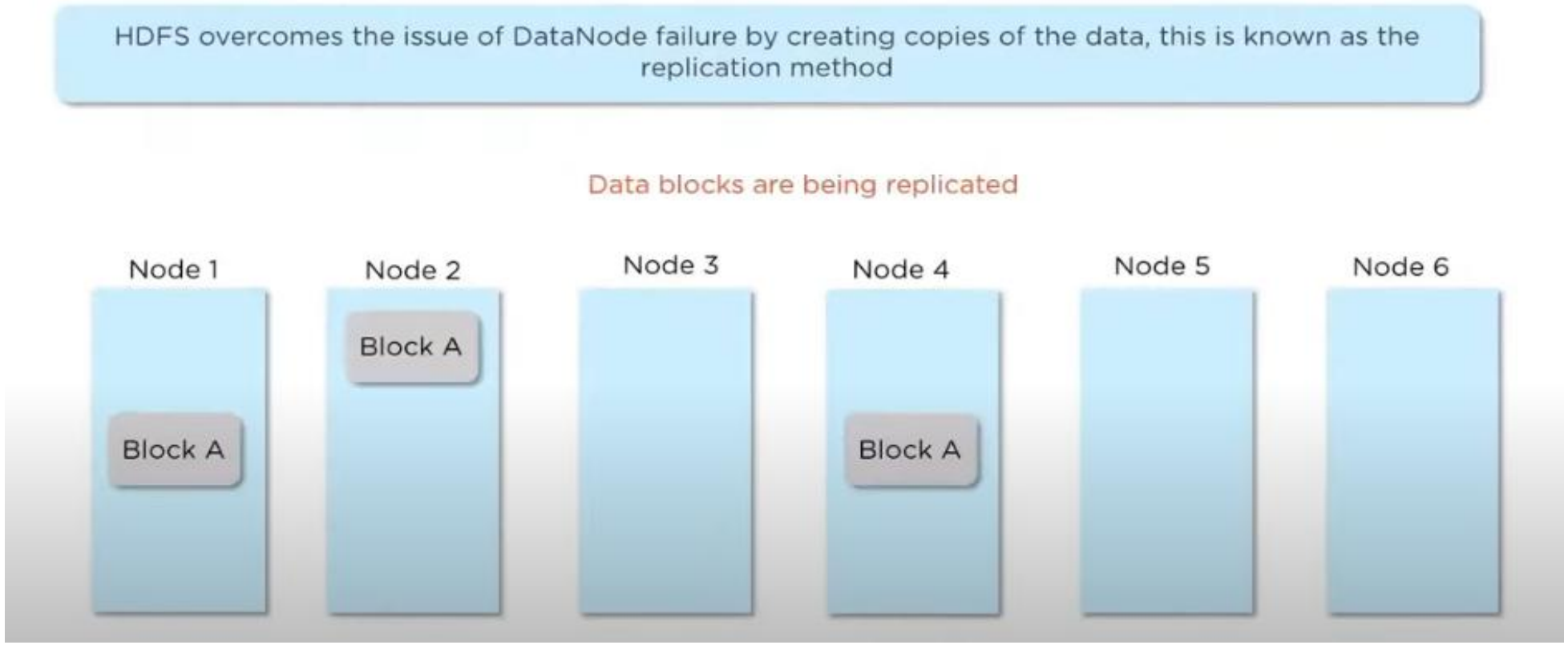
DataNode failure




Now, what happens when
Node 5 crashes?
Node 1
Node 2
Node 3
Node 4
Node 5
Node 6
Block A
Block B
Block C
Block D
Block E


Data is lost!
The data stored in Node 5 will be unavailable as there is no copy stored elsewhere
Node 1
Node 2
Node 3
Node 4
Node 5
Node 6
Block A
Block B
Block C
Block D
Block E

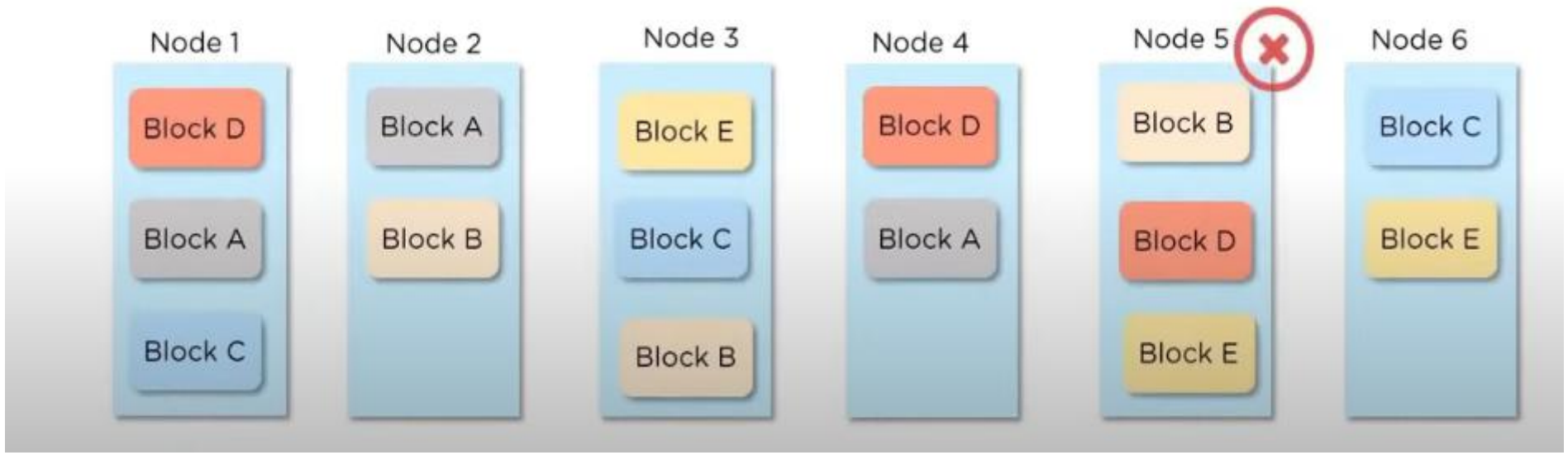
Replication



Now, what happens when

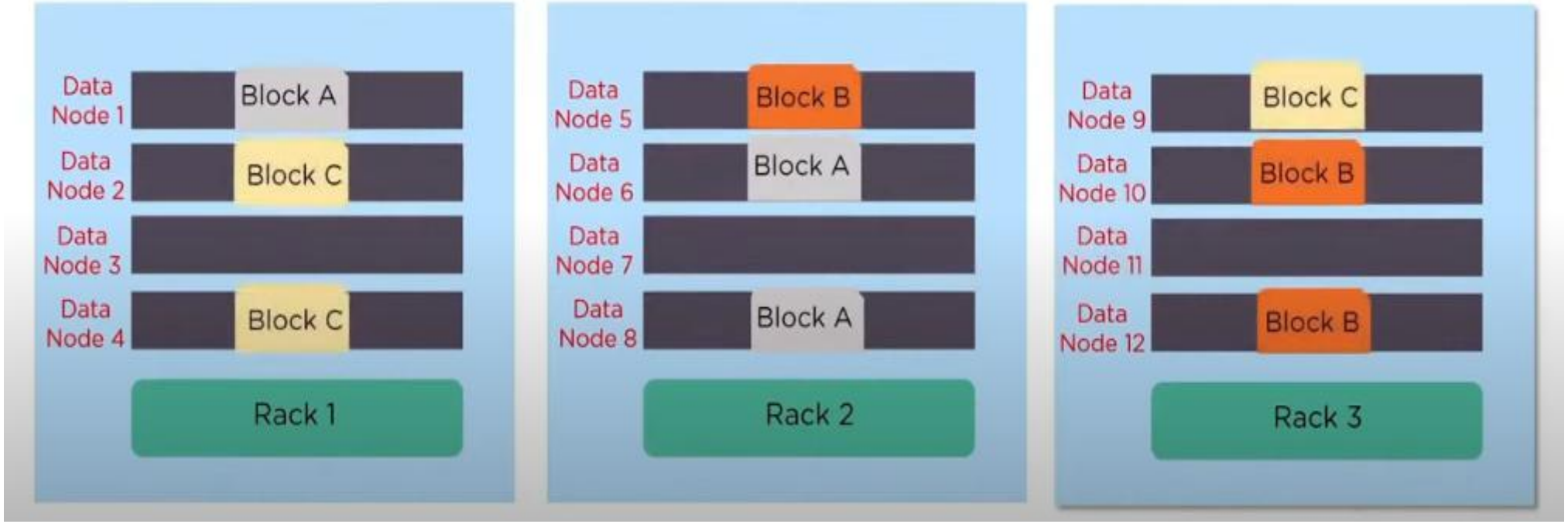
Node 5 crashes?

Will the data blocks B, D and E be lost?

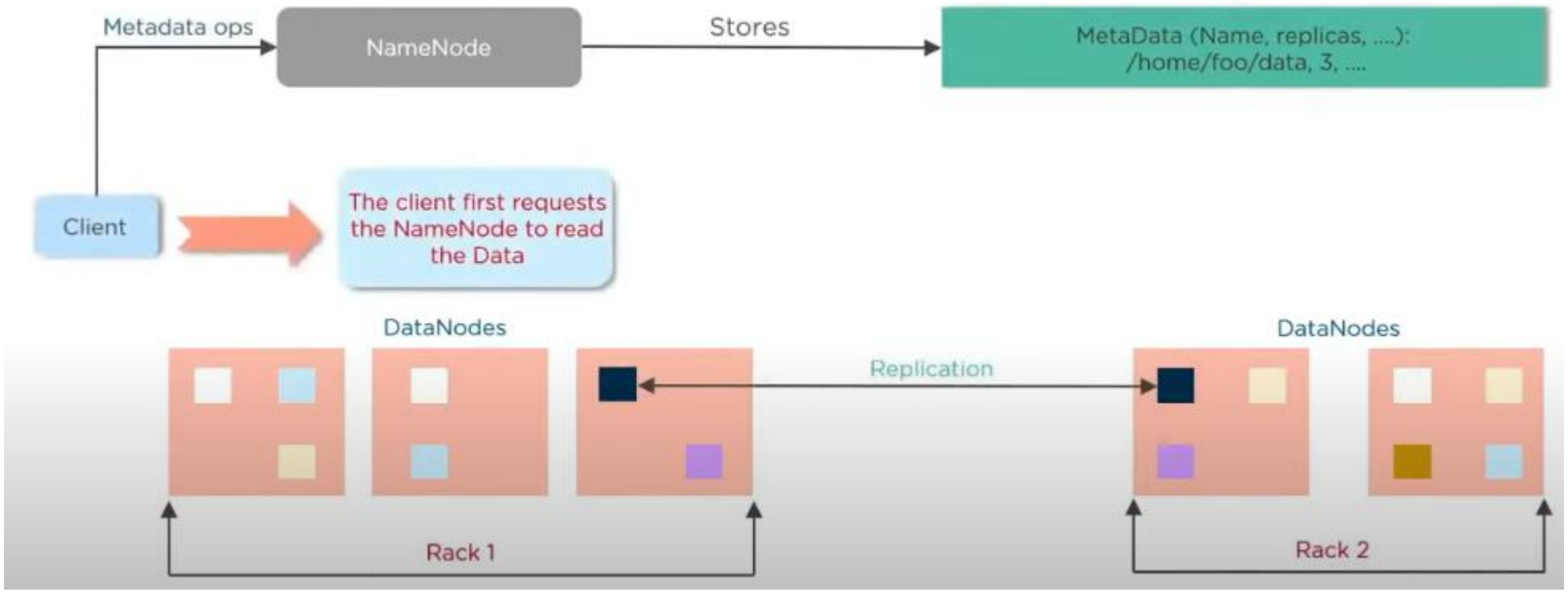


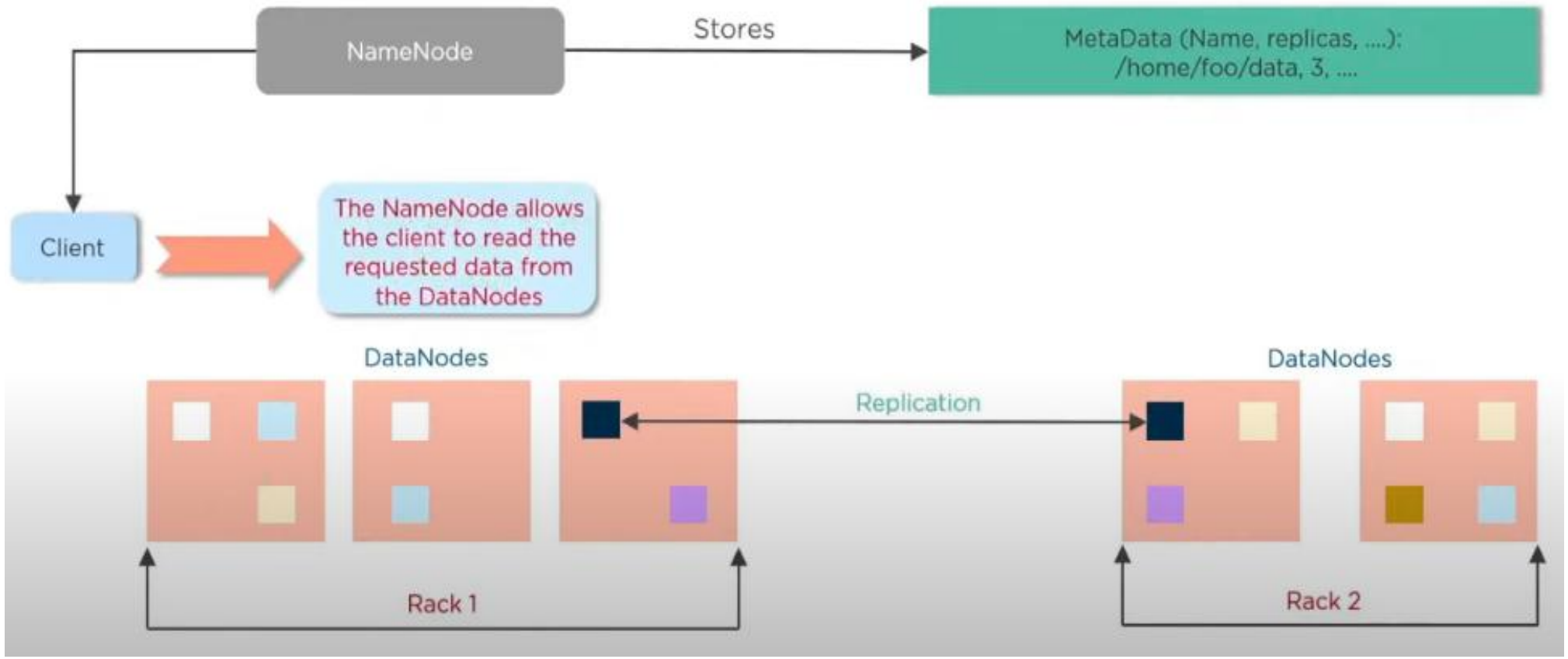
Rack Awareness in HDFS

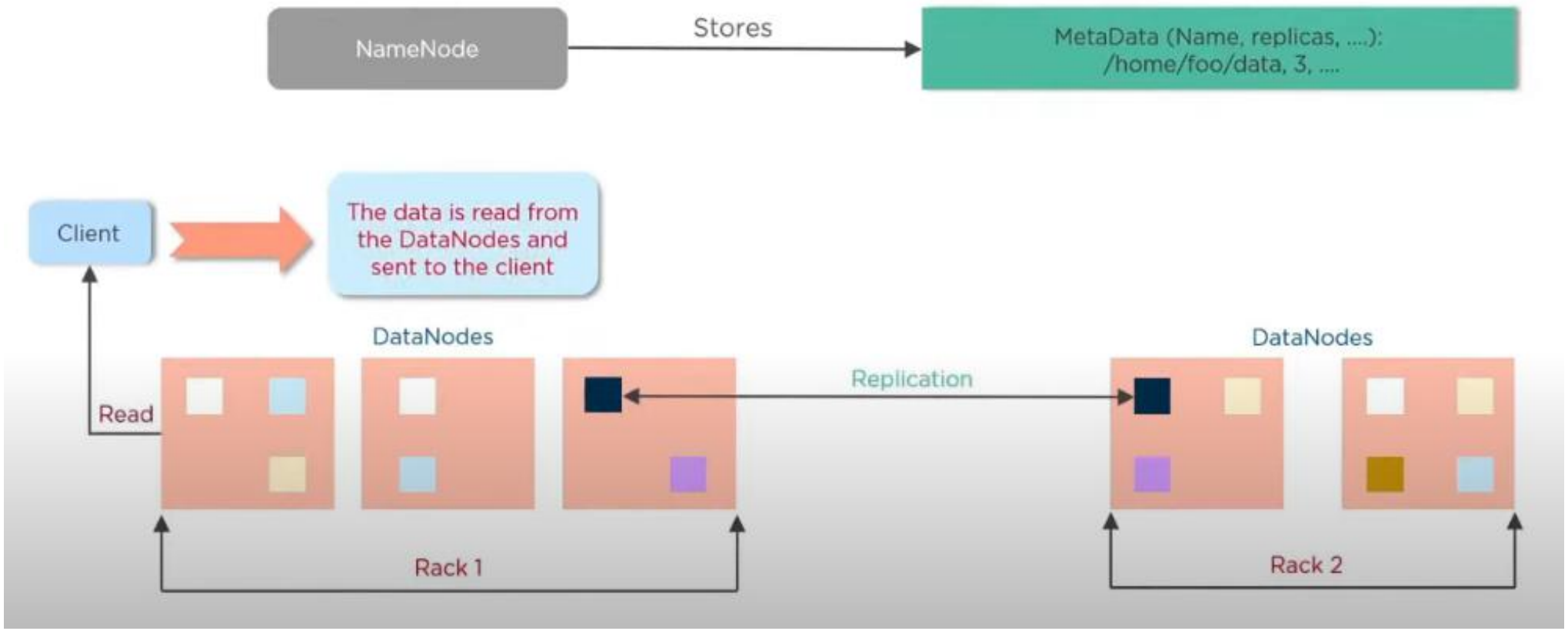
Rack is a collection of 30-40 DataNodes. Rack Awareness is a concept that helps to decide where a replica of the data block should be stored

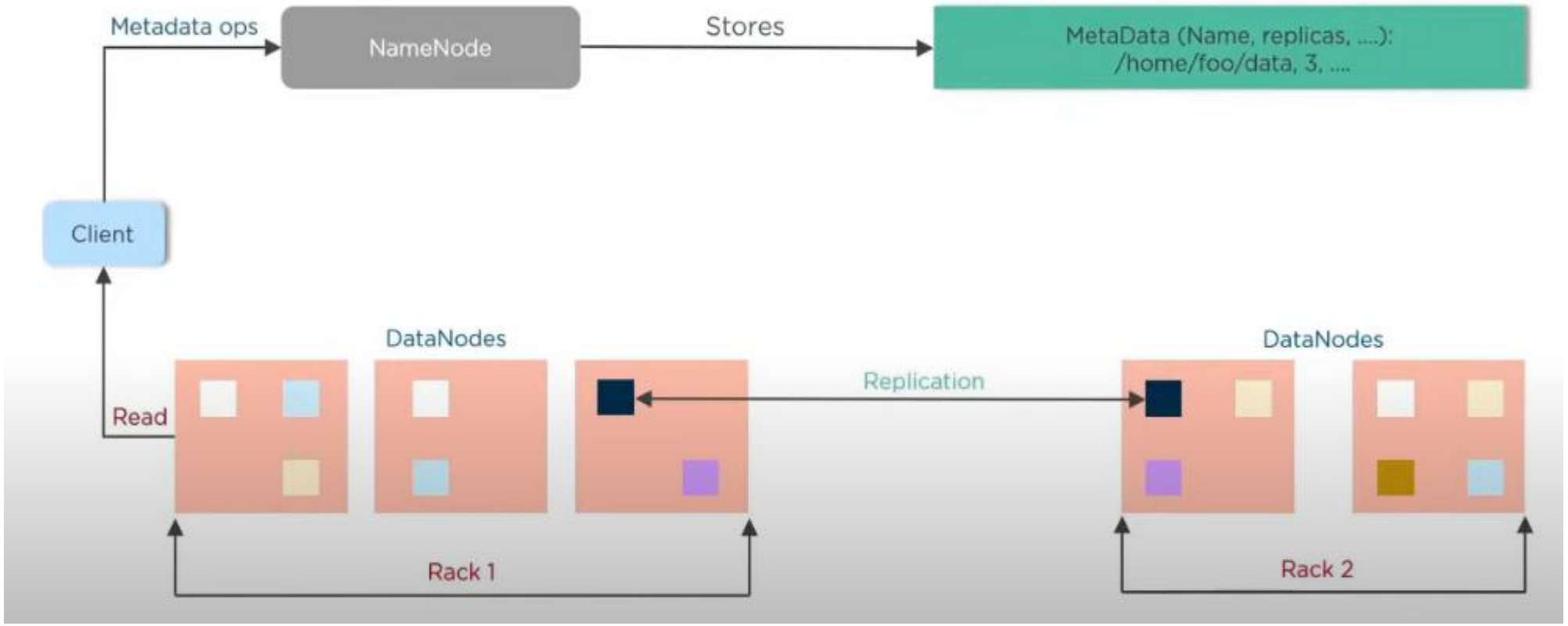


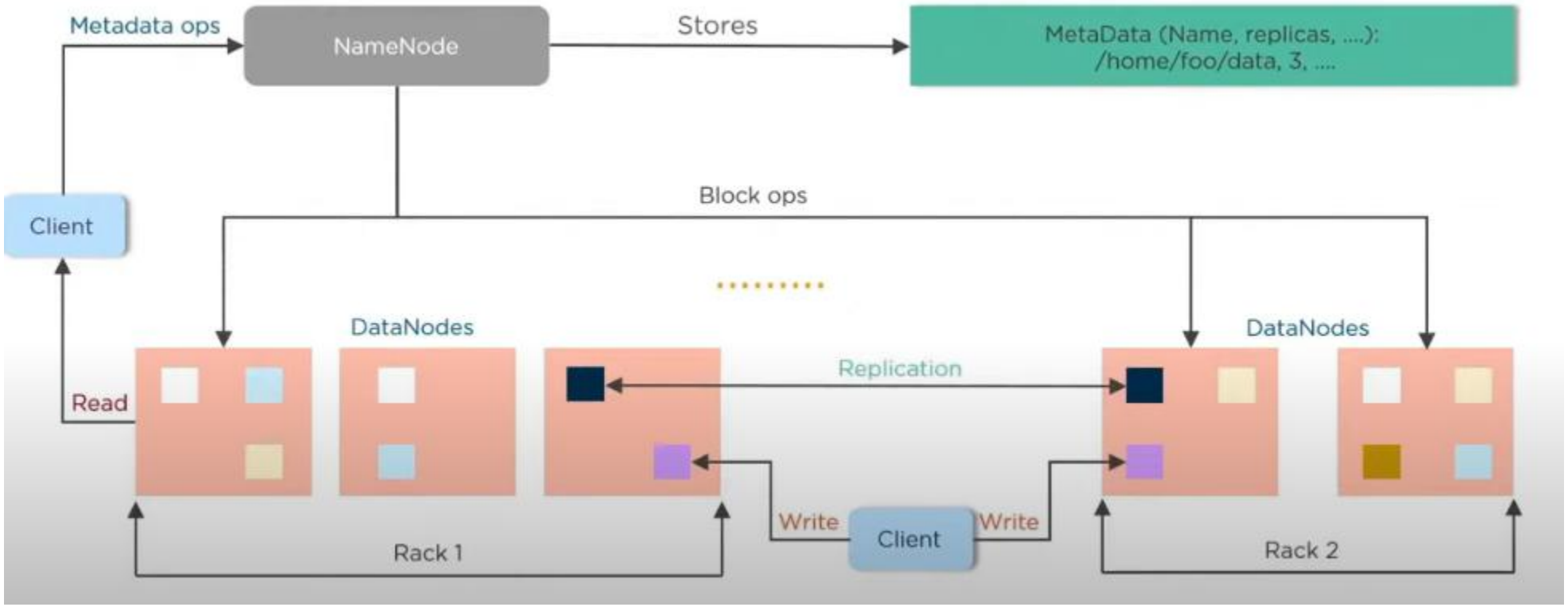
HDFS architecture


Metadata ops
NameNode
Stores
MetaData (Name, replicas, ...):
/home/foo/data, 3, ...
Client
The client first requests the NameNode to read the Data
DataNodes
Replication
DataNodes
Rack 1
Rack 2

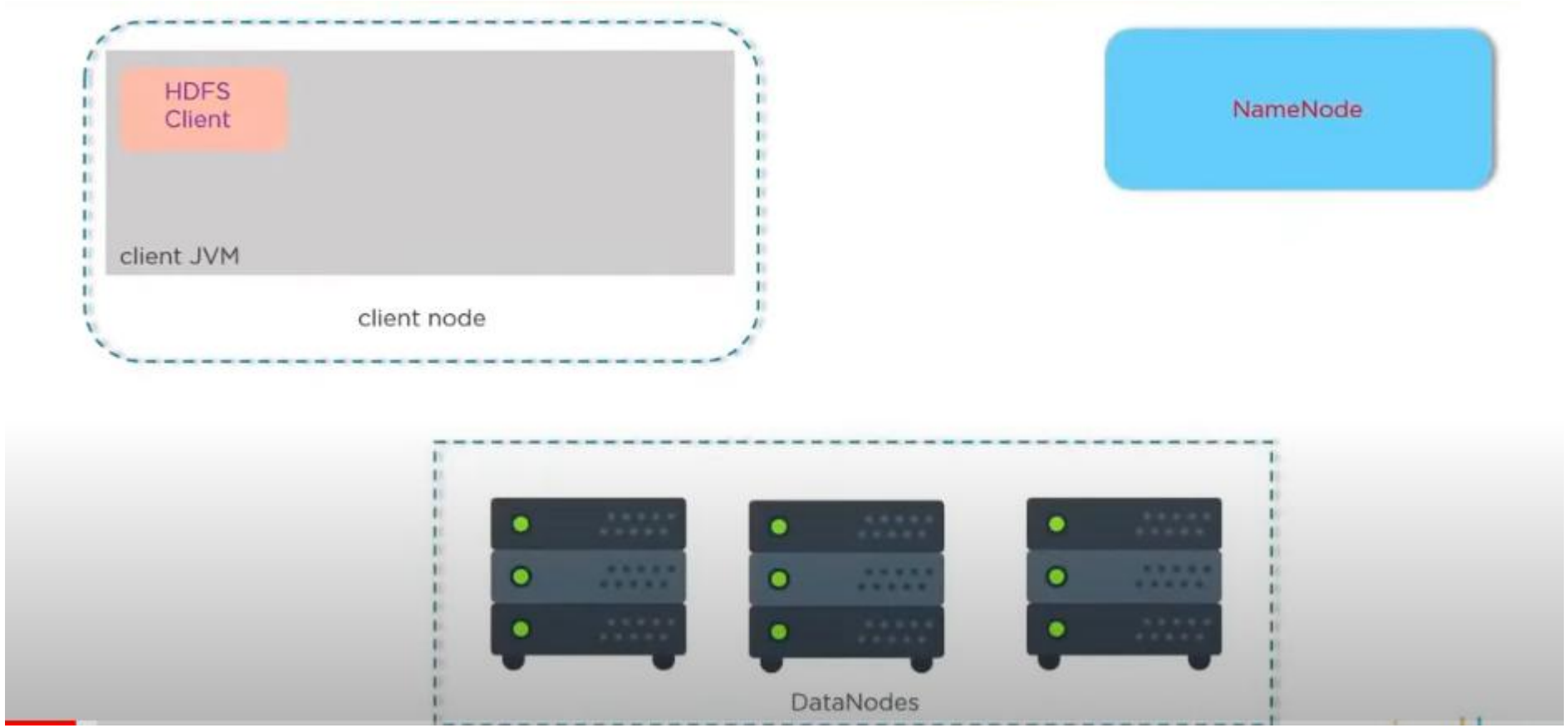

NameNode
Stores
MetaData (Name, replicas, ...):
/home/foo/data, 3, ...
Client
The NameNode allows the client to read the requested data from the DataNodes
DataNodes
DataNodes
Replication
Rack 1
Rack 2

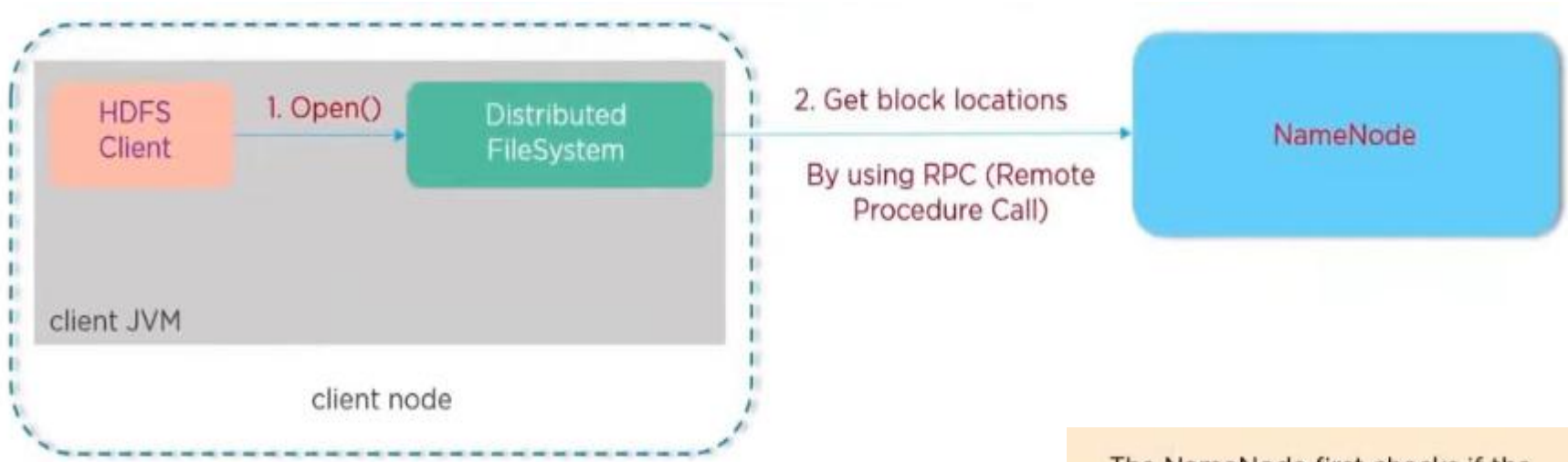

NameNode
Stores
MetaData (Name, replicas, ...):
/home/foo/data, 3, ...
Client
The data is read from the DataNodes and sent to the client
DataNodes
DataNodes
Read
Replication
Rack 1
Rack 2


Metadata ops
NameNode
Stores
MetaData (Name, replicas, ...):
/home/foo/data, 3, ...
Client
DataNodes
DataNodes
Read
Replication
Rack 1
Rack 2


Metadata ops
NameNode
Stores
MetaData (Name, replicas, ...):
/home/foo/data, 3, ...
Client
Block ops
DataNodes
DataNodes
Read
Replication
Write
Client
Write
Rack 1
Rack 2

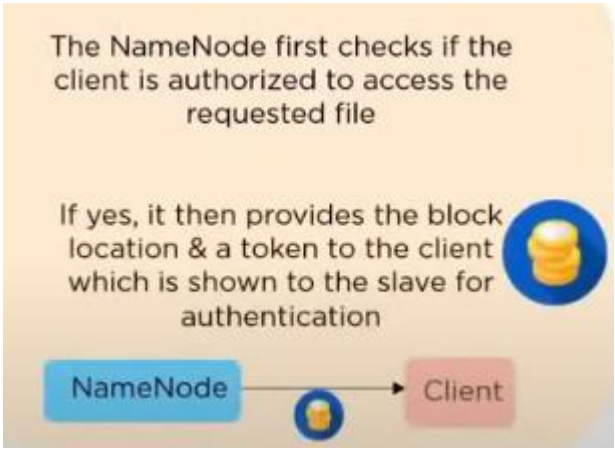
HDFS read file Mechanism

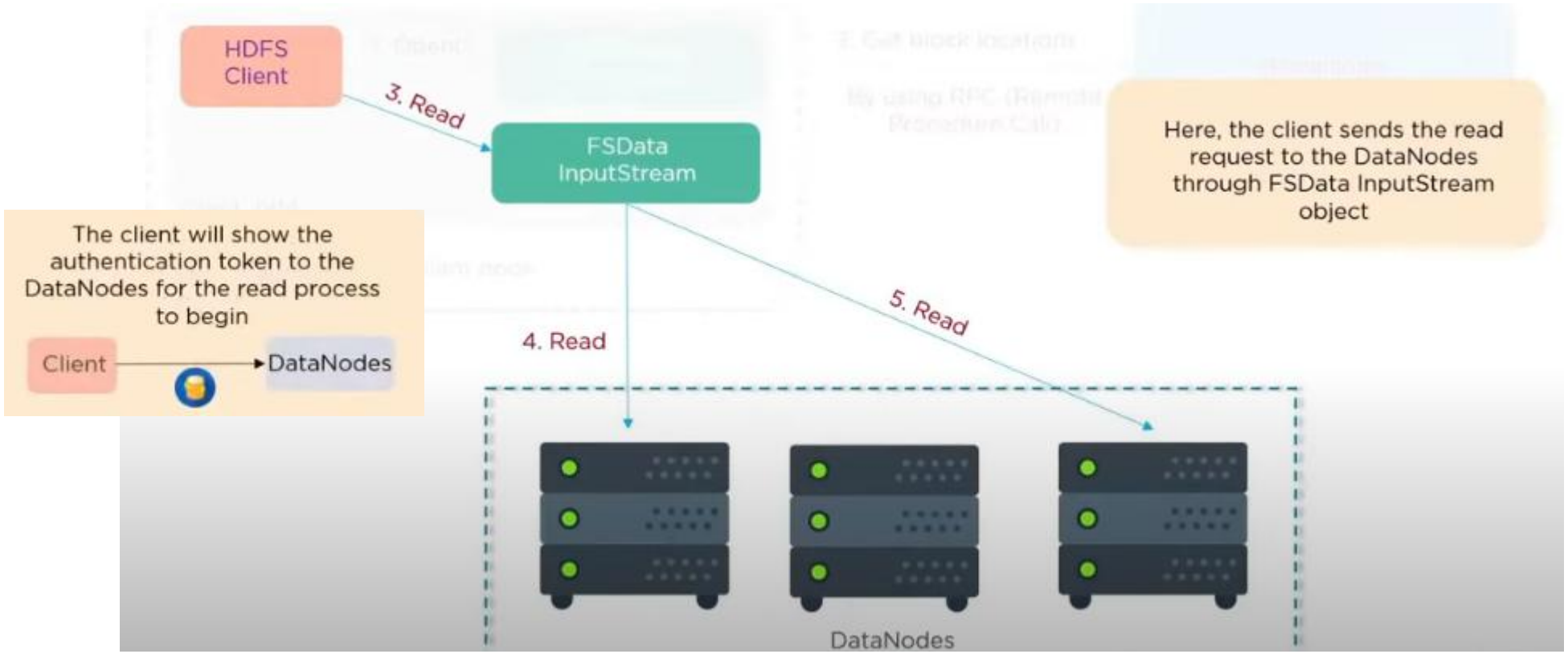


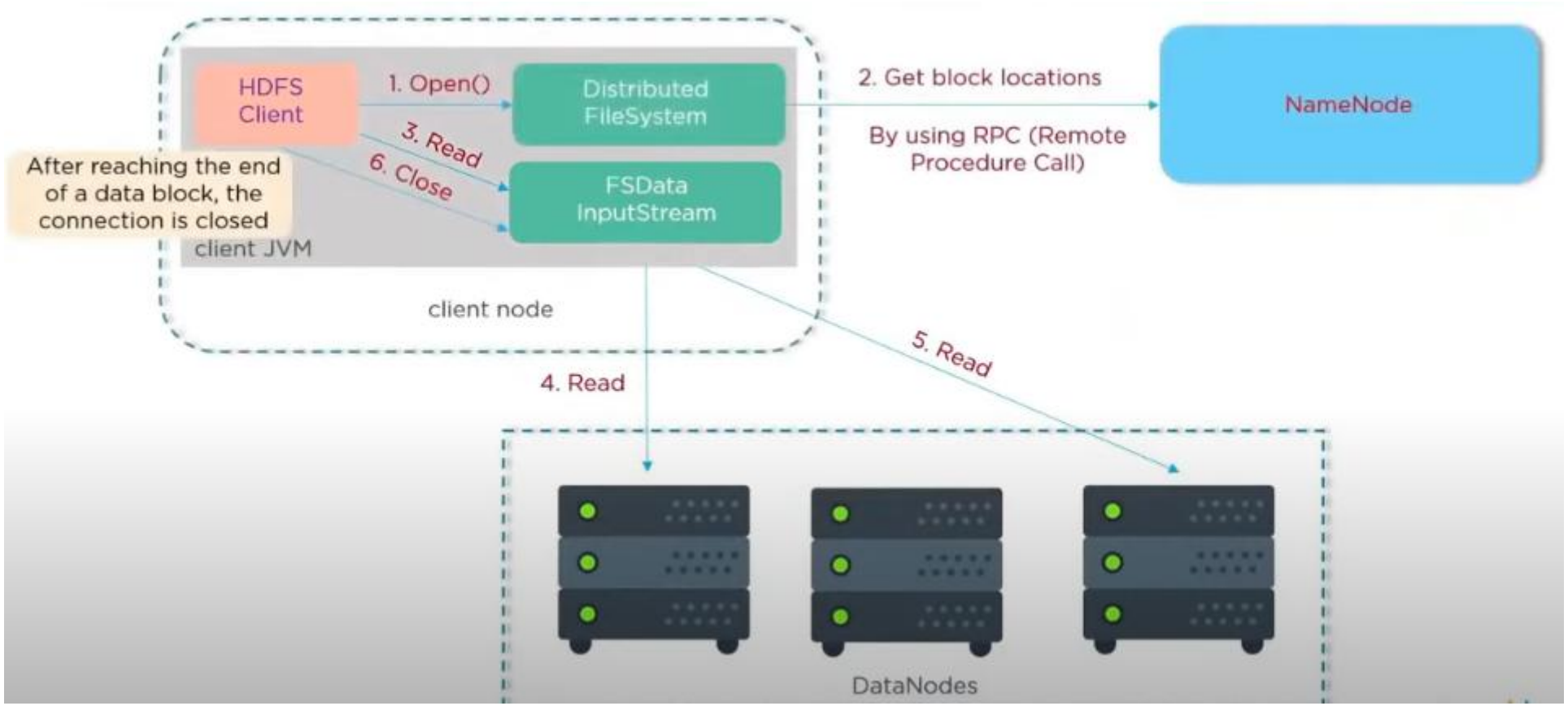


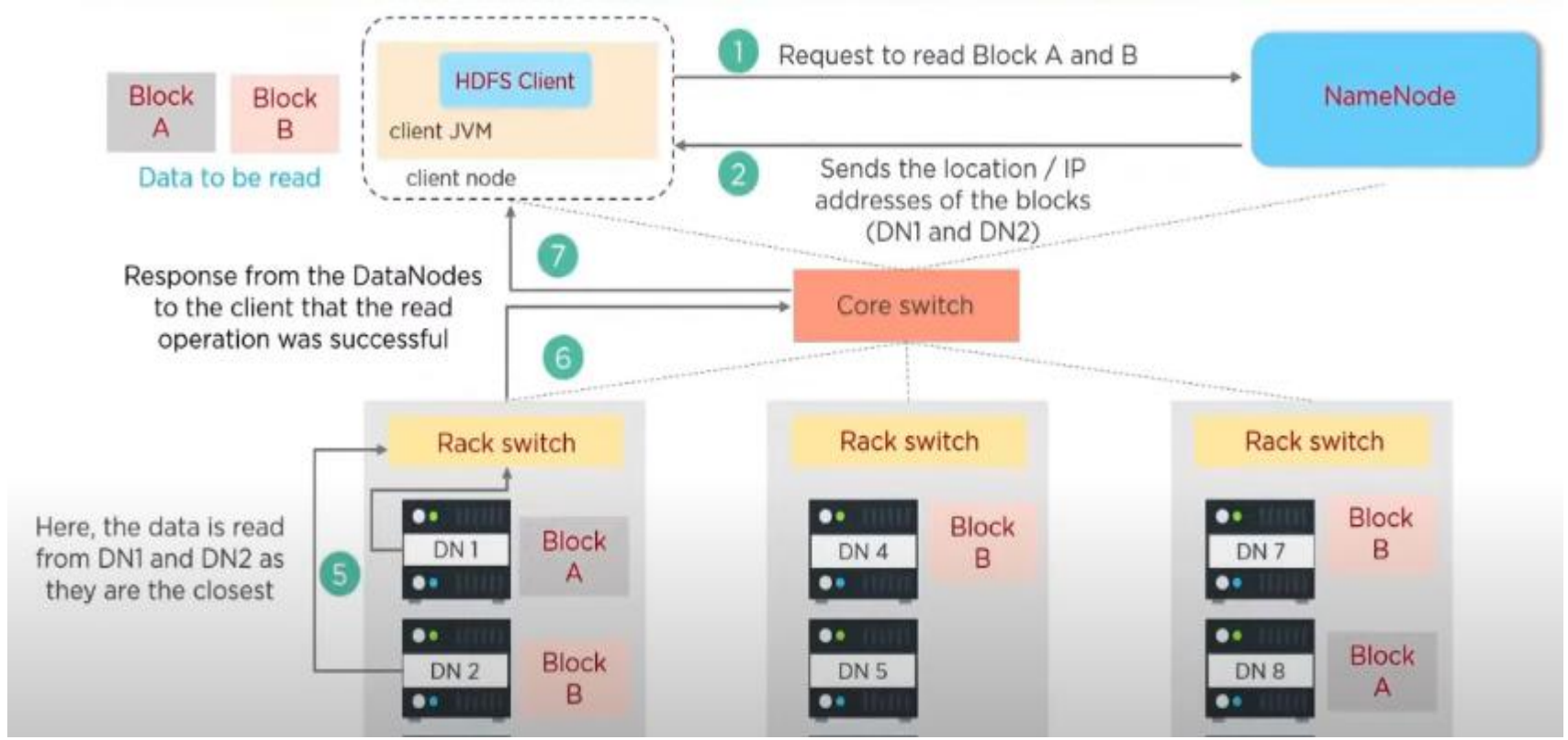
The NameNode first checks if the client is authorized to access the requested file

If yes, it then provides the block location & a token to the client which is shown to the slave for authentication

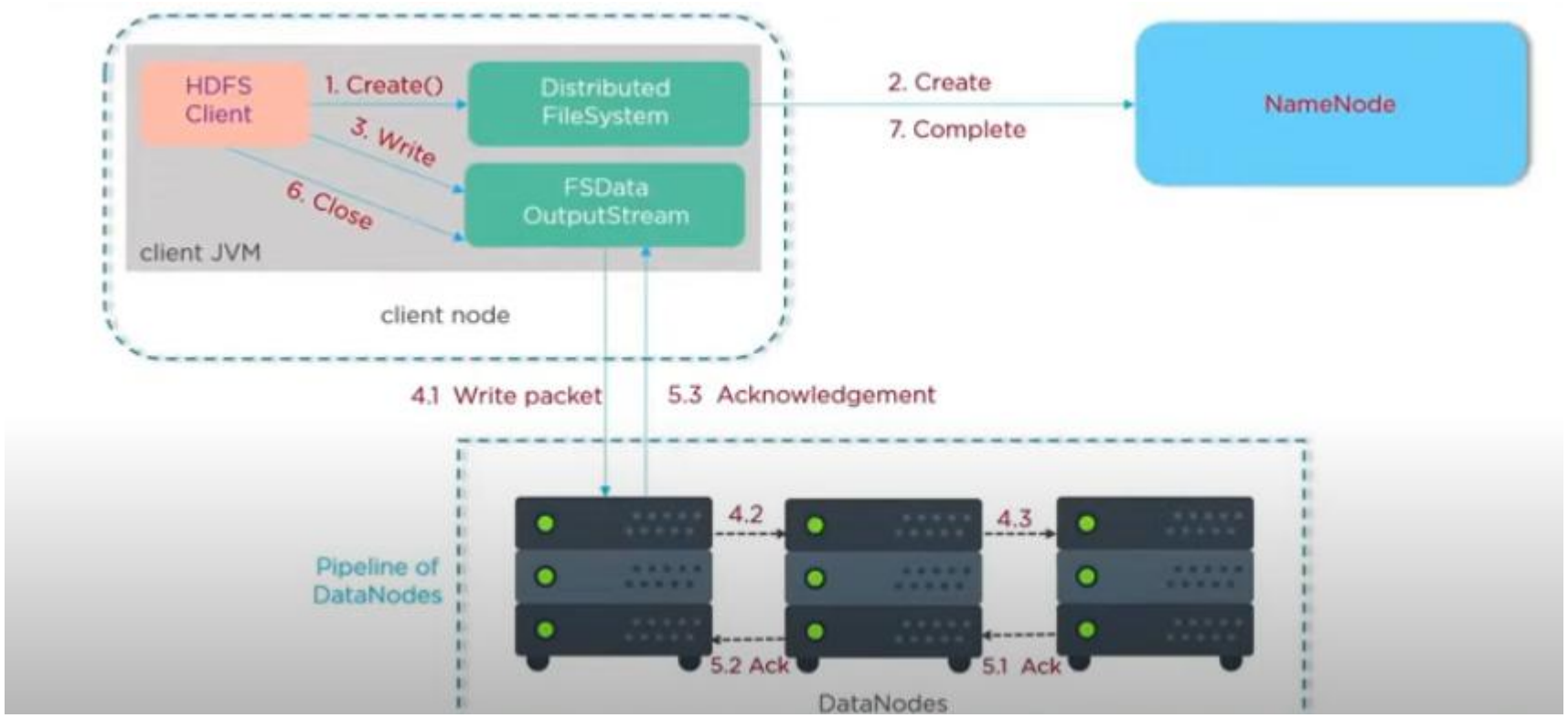


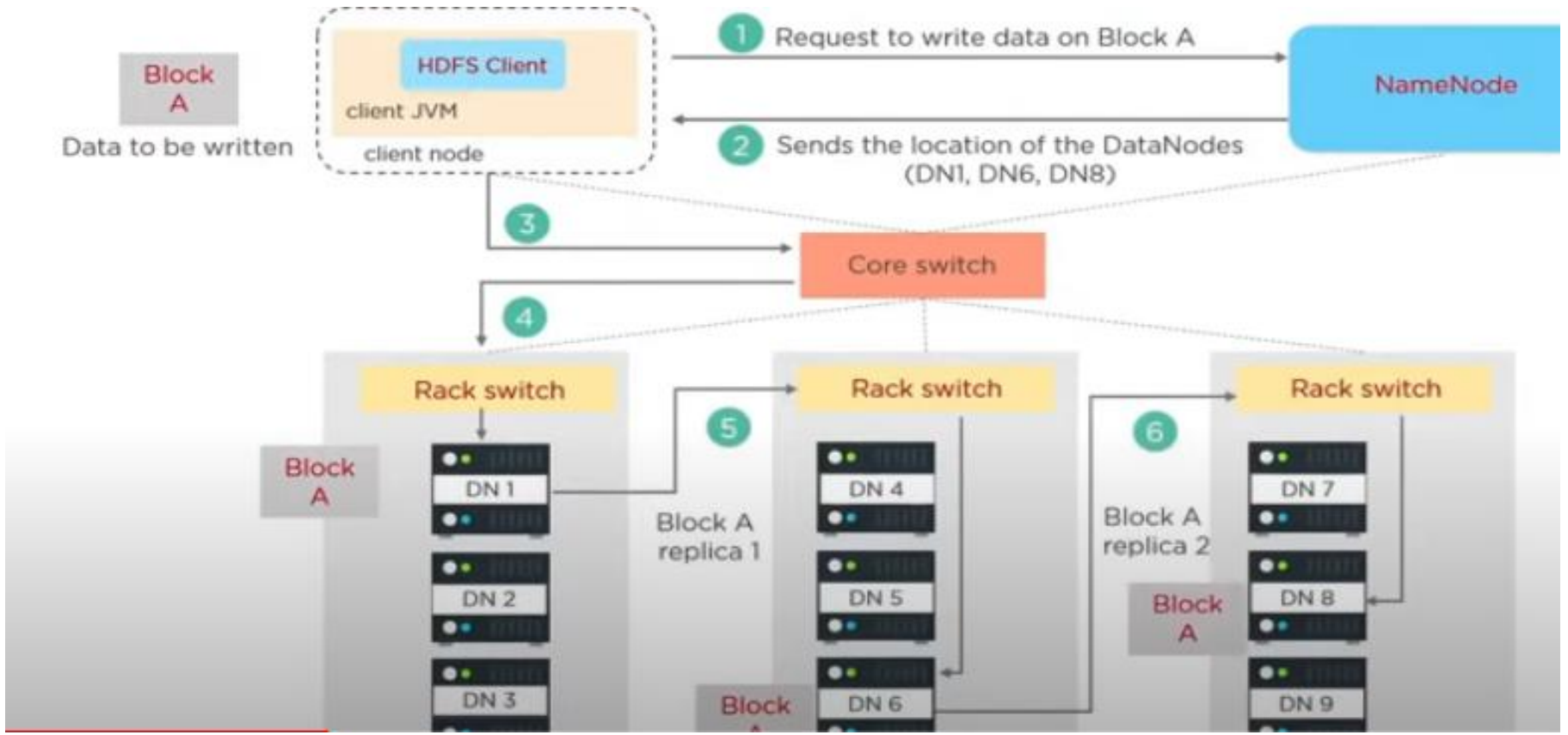

HDFS Client
3. Read
FSData InputStream
Here, the client sends the read request to the DataNodes through FSData InputStream object
The client will show the authentication token to the DataNodes for the read process to begin
Client
DataNodes
4. Read
5. Read
DataNodes

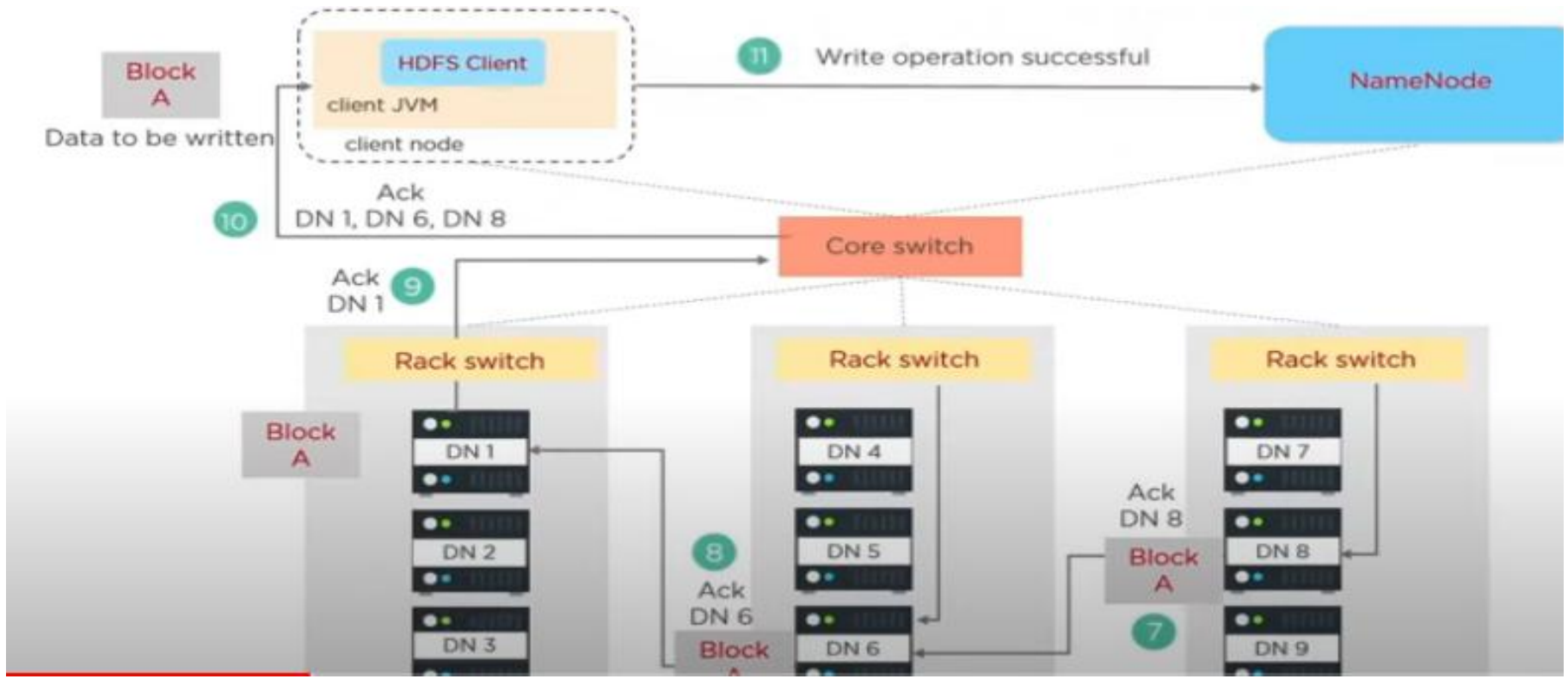

HDFS Client
1. Open()
Distributed FileSystem
2. Get block locations
By using RPC (Remote Procedure Call)
NameNode
3. Read
6. Close
FSData InputStream
After reaching the end of a data block, the connection is closed
client JVM
client node
4. Read
5. Read
DataNodes


Block A
Block B
Data to be read
HDFS Client
client JVM
client node
1
Request to read Block A and B
NameNode
2
Sends the location / IP addresses of the blocks (DN1 and DN2)
7
Response from the DataNodes to the client that the read operation was successful
Core switch
6
Rack switch
Rack switch
Rack switch
Here, the data is read from DN1 and DN2 as they are the closest
5
DN 1
Block A
DN 2
Block B
DN 4
Block B
DN 5
DN 7
Block B
DN 8
Block A

HDFS write mechanism




Block A
Data to be written
HDFS Client
client JVM
client node
1 Request to write data on Block A
NameNode
2 Sends the location of the DataNodes (DN1, DN6, DN8)
3
Core switch
4
Rack switch
Rack switch
Rack switch
5
6
Block A
DN 1
DN 2
DN 3
Block A replica 1
DN 4
DN 5
DN 6
Block A replica 2
DN 7
DN 8
DN 9
Block A


Block A
Data to be written
HDFS Client
client JVM
client node
11
Write operation successful
NameNode
10
Ack
DN 1, DN 6, DN 8
Core switch
Ack
DN 1
9
Rack switch
Rack switch
Rack switch
Block A
DN 1
DN 2
DN 3
DN 4
DN 5
DN 6
DN 7
DN 8
DN 9
8
Ack
DN 6
Block
Ack
DN 8
Block A
7

Advantage of HDFS