Hadoop

Doug Cutting

The name my kid gave a stuffed yellow elephant. Short, relatively easy to spell and pronounce, meaningless, and not used elsewhere: those are naming criteria. Kids are good at generating such. Googol is a kid's term..

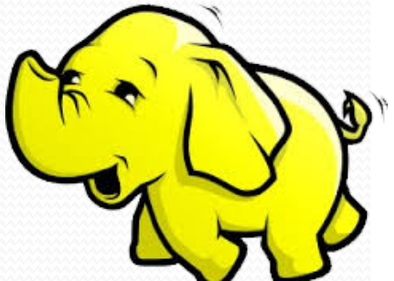
Hadoop's Developers

Doug Cutting

2005: Doug Cutting and Michael J. Cafarella developed Hadoop to support distribution for the Nutch search engine project.

The project was funded by Yahoo.

2006: Yahoo gave the project to Apache Software Foundation.



History of Hadoop

- In 2002 Apache Nutch was started and it is open-source software. In 2003 Google introduced GFS (Google File System) to get enough access to data to distributed file systems.
- In 2004 Google released a white paper on map reduces. It is a technique and program model for processing works on java based computing.
- In 2005 NDFS was introduced (Nutch distributed file system) by Doug Cutting and Mike Cafarella. It is a new file system in Hadoop. The HDFS and NDFS are the same.
- In 2006 Google joined Yahoo with Doug cutting quit. Doug cutting did a new project on Hadoop distributed file system based on Nutch distributed file system. In this same year, Hadoop's first version 0.1.0 was released.
- In 2007 yahoo started running two clusters at the same time in 1000 machines.
- In 2008 Hadoop became the fastest system.
- In 2013 Hadoop 2.2 was released.

Cluster

- Hadoop provides an efficient framework for running jobs on multiple nodes of clusters.
- Cluster means a group of systems connected via LAN. Apache Hadoop provides parallel processing of data as it works on multiple machines simultaneously.

Framework

- A software framework is a set of reusable software components that form the application development structure. It provides a predefined structure, rules, and guidelines that developers can build upon, reducing the need to start from scratch and enabling more efficient development.
- Framework is the basic structure intended to serve as a support or guide for the building of something.

Hadoop

- Hadoop is an open-source tool from the ASF — Apache Software Foundation. Open source project means it is freely available and we can even change its source code as per the requirements. If certain functionality does not fulfil your need then you can change it according to your need. Most of Hadoop code is written by Yahoo, IBM, Facebook, Cloudera.

Open Source

- It is a framework that is free to use or even distribute, and its source code can be modified to achieve specific results.

Scalable

- It can run on a single machine, and it can be scaled to run on hundreds of machines without any downtime, dynamically.

Fault-Tolerant

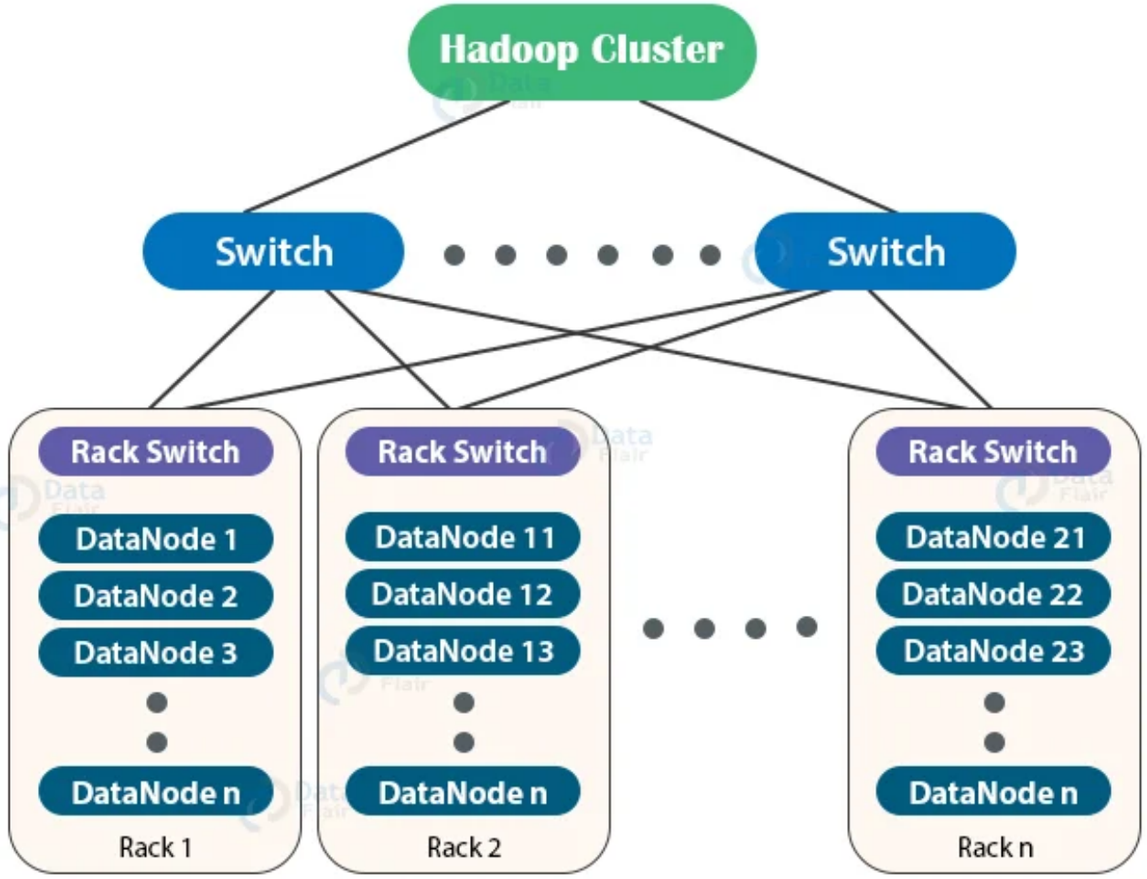
- The framework creates replicas of data on different machines in the cluster (a group of machines), which means that even if one of the machines goes down (even if it burns down to ash and smoke), Hadoop will have not lost any data and will continue working.

Runs on Commodity Hardware

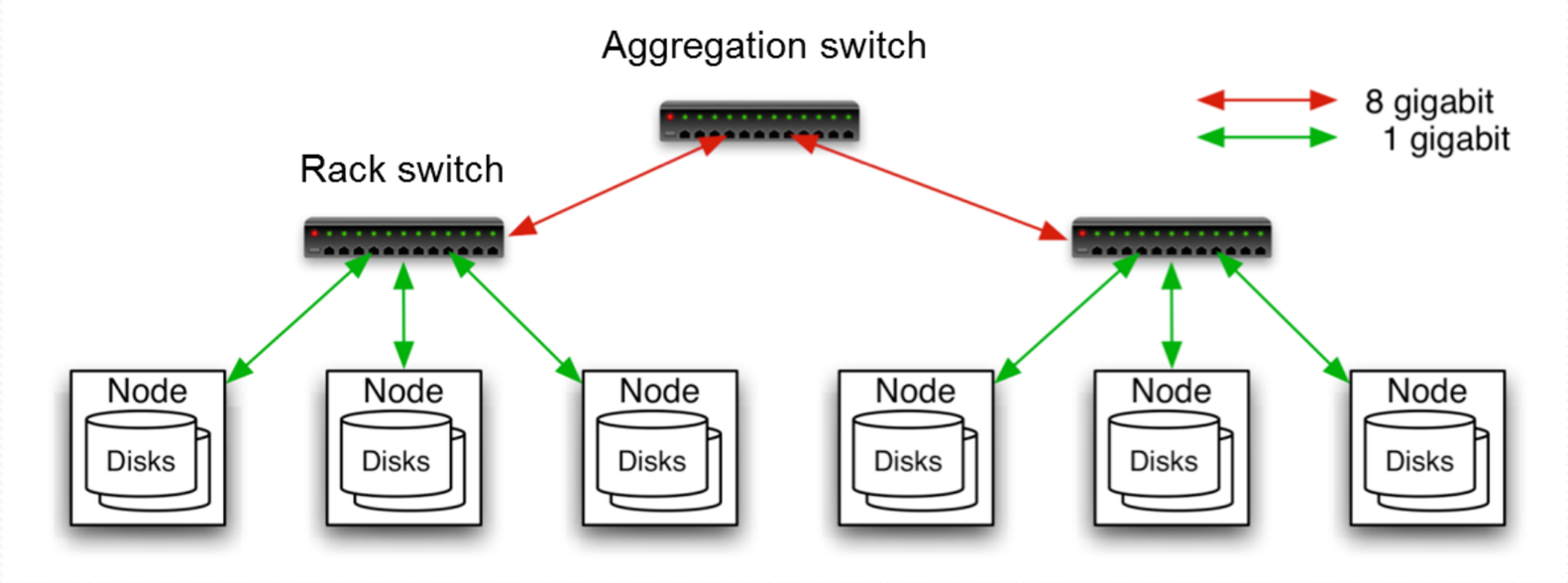
- As if it wasn’t enough for Hadoop to impress by being open-source, it turns out, it can be run on inexpensive, everyday computer systems, i.e. commodity hardware, thus proving itself highly cost-effective.

Rack

- Rack is the collection of around 40-50 machines (DataNodes) connected using the same network switch. If the network goes down, the whole rack will be unavailable.



Commodity Hardware



- Typically in 2 level architecture
- Nodes are commodity PCs
- 30-40 nodes/rack
- Uplink from rack is 3-4 gigabit
- Rack-internal is 1 gigabit

- Hadoop is a data handling framework written primarily in Java, with some secondary code in shell script and C. It uses a basic-level programming model and is able to deal with large datasets.
- Hadoop is referred to as a "framework" because it provides a comprehensive ecosystem of tools, libraries, and components that work together to solve various big data challenges. Here's why Hadoop is considered a framework:

Hadoop Framework

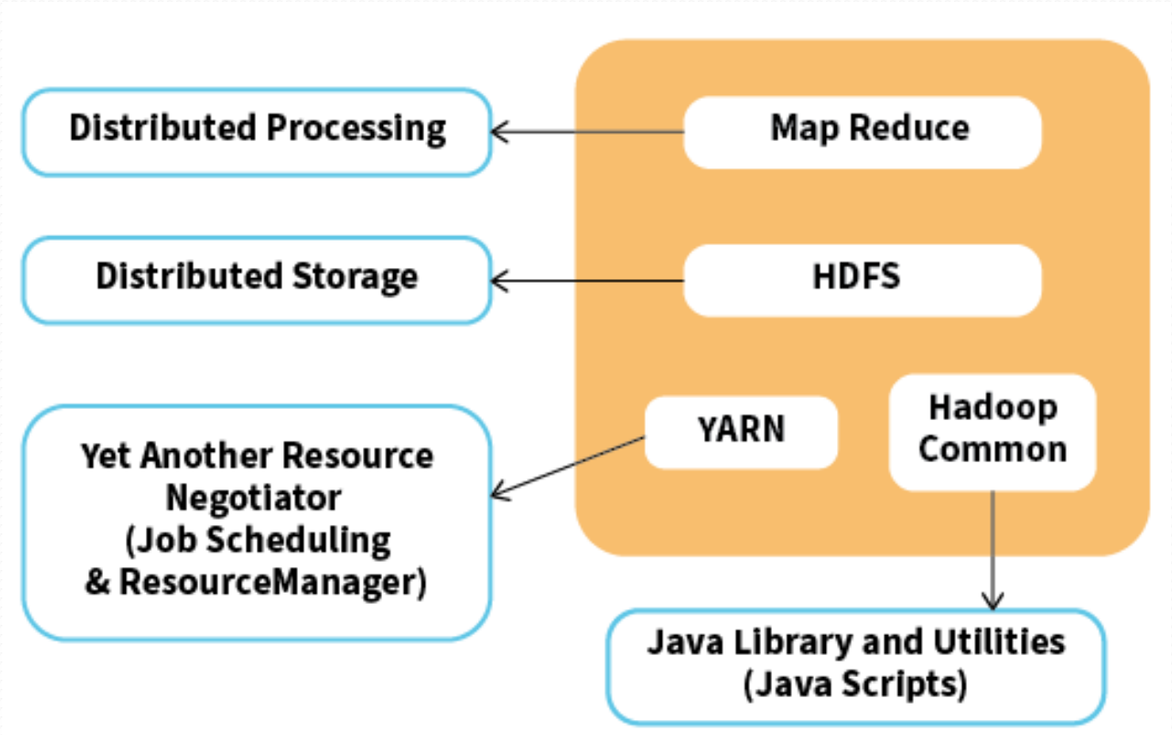
- **Abstraction of Complexity**: Hadoop abstracts away the complexities of distributed computing, making it easier for developers to work with large datasets across clusters of computers. It provides high-level abstractions like Hadoop Distributed File System (HDFS) for distributed storage and MapReduce (or other processing engines like Apache Spark) for distributed processing. This abstraction allows developers to focus on their data processing logic rather than worrying about the underlying infrastructure.
- **Modular Architecture**: Hadoop is designed with a modular architecture, allowing different components to work together seamlessly. For example, Hadoop consists of components like HDFS for storage, YARN for resource management, and MapReduce (or other processing engines) for computation. These components can be used together or independently, depending on the specific requirements of an application. This modular design makes Hadoop highly flexible and adaptable to various use cases.

- **Scalability:** Hadoop is built to scale horizontally, meaning you can easily add more commodity hardware to your Hadoop cluster to handle increasing amounts of data and workload. The framework automatically distributes data and computation across the cluster, enabling linear scalability as the size of the cluster grows. This scalability feature is essential for handling big data applications where datasets are constantly growing.
- **Fault Tolerance:** Hadoop provides built-in fault tolerance mechanisms to ensure the reliability of data processing. For example, HDFS replicates data blocks across multiple nodes in the cluster to guard against hardware failures. Similarly, processing frameworks like MapReduce or Spark have mechanisms to detect and recover from failures during computation. These fault tolerance features ensure that data processing continues uninterrupted even in the face of hardware failures or network issues.

- **Community Support and Ecosystem**: Hadoop has a large and vibrant community of developers and contributors who actively work on improving and extending the framework. The Hadoop ecosystem includes a wide range of tools and libraries for various data processing tasks, such as Apache Hive for SQL-like querying, Apache Pig for data processing, Apache Spark for real-time processing, Apache HBase for NoSQL database functionality, and many others. This rich ecosystem of tools and libraries makes Hadoop a powerful framework for building data-intensive applications.

Hadoop Basic Components

- Hadoop is built on three important core components, that take care of the data distribution in the network, resource allocation, and processing.

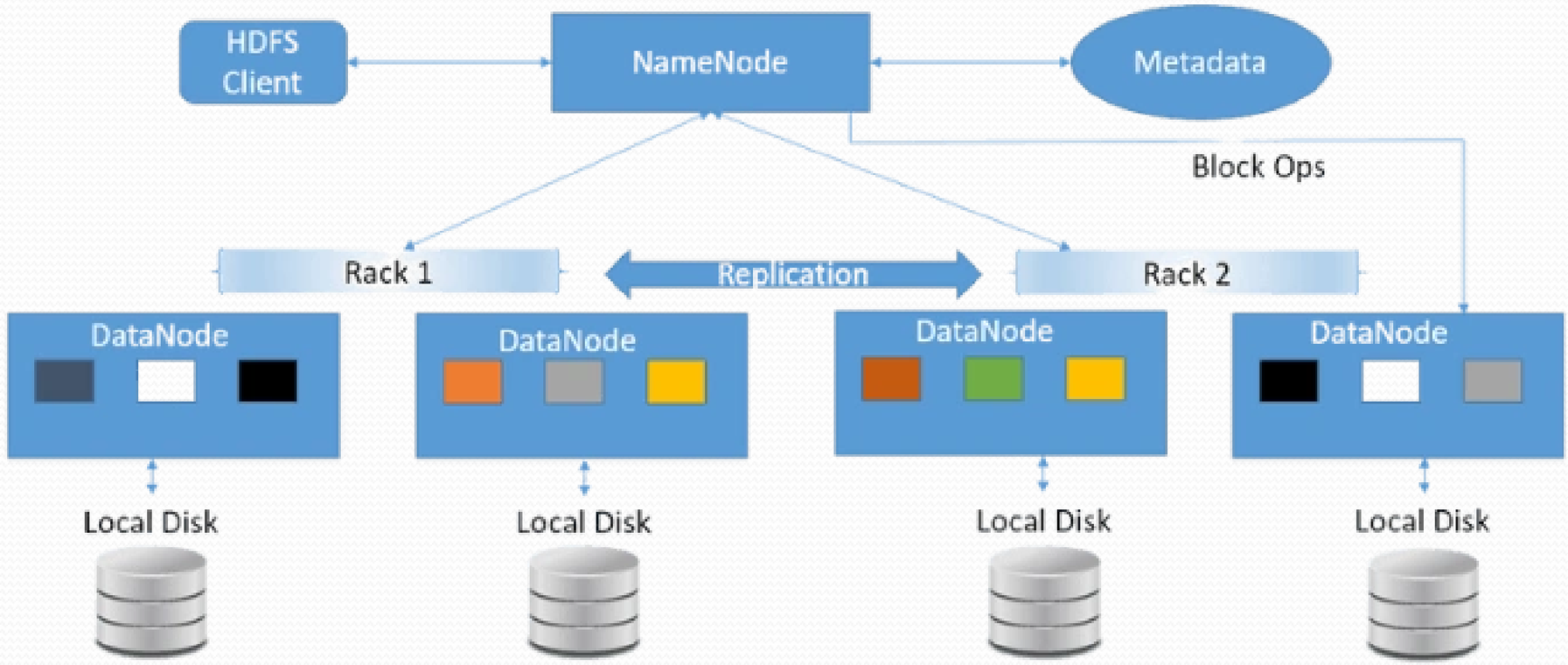


Unique features of HDFS

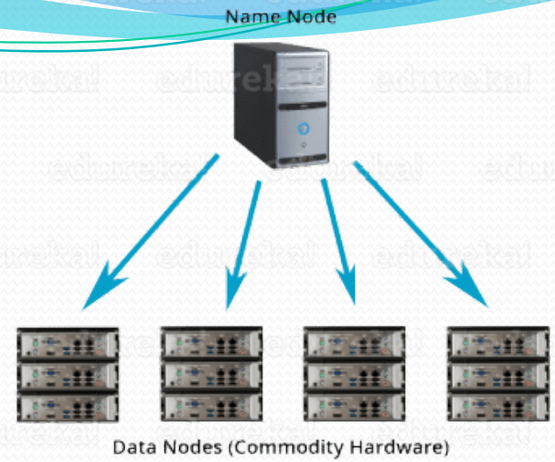
HDFS also has a bunch of unique features that make it ideal for distributed systems:

- Failure tolerant - data is duplicated across multiple DataNodes to protect against machine failures. The default is a replication factor of 3 (every block is stored on three machines).
- Scalability - data transfers happen directly with the DataNodes so your read/write capacity scales fairly well with the number of DataNodes
- Space - need more disk space? Just add more DataNodes and rebalance

HDFS Architecture



HDFS Architecture



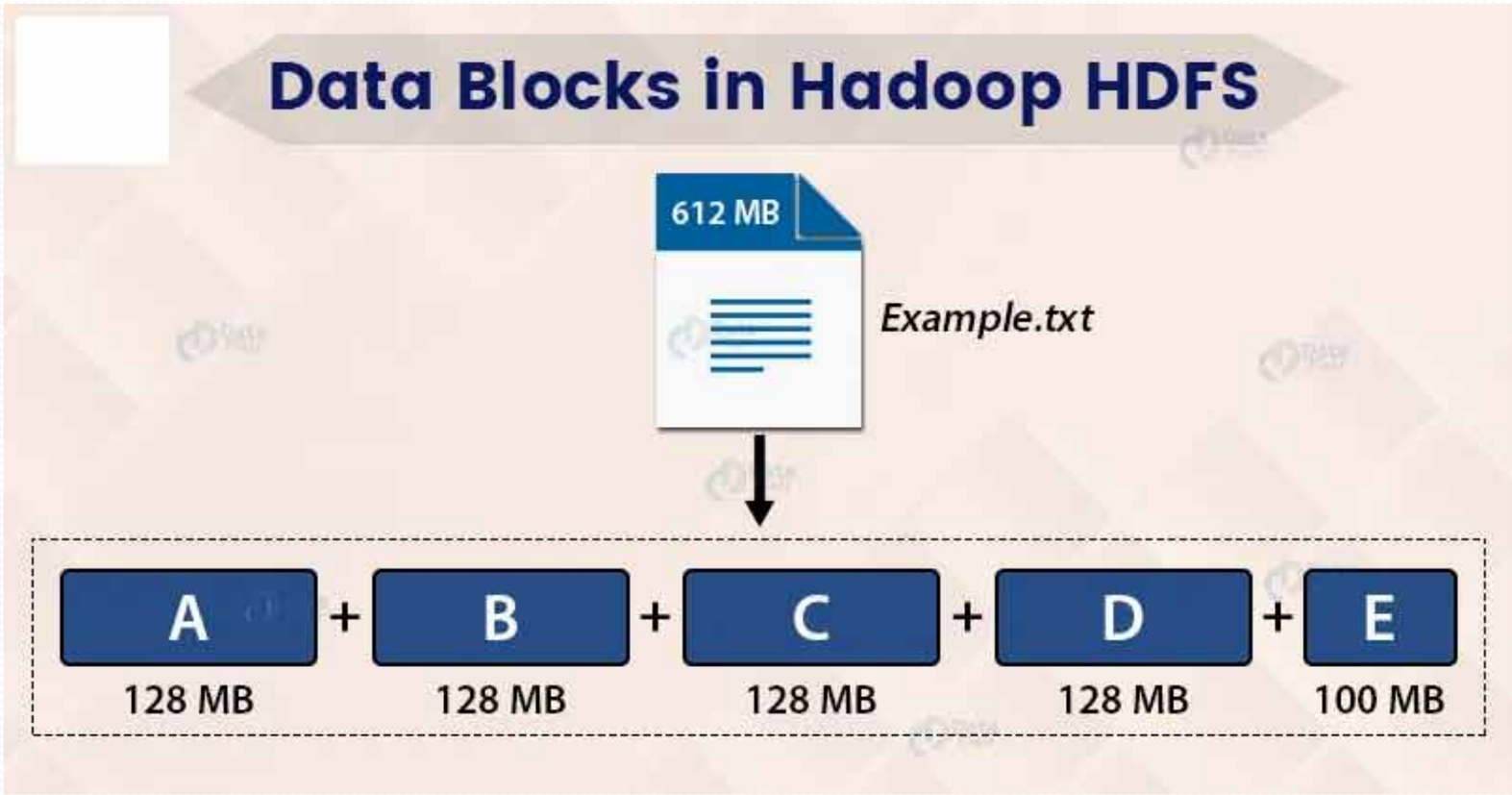
- Hadoop Distributed File System follows the **master-slave architecture**.
- Each cluster comprises a **single master node** and **multiple slave nodes**.
- Internally the files get divided into one or more **blocks**, and each block is stored on different slave machines depending on the **replication factor**
- The master node stores and manages the file system namespace, that is information about blocks of files like block locations, permissions, etc. The slave nodes store data blocks of files.
- The Master node is the NameNode and DataNodes are the slave nodes.

Blocks:

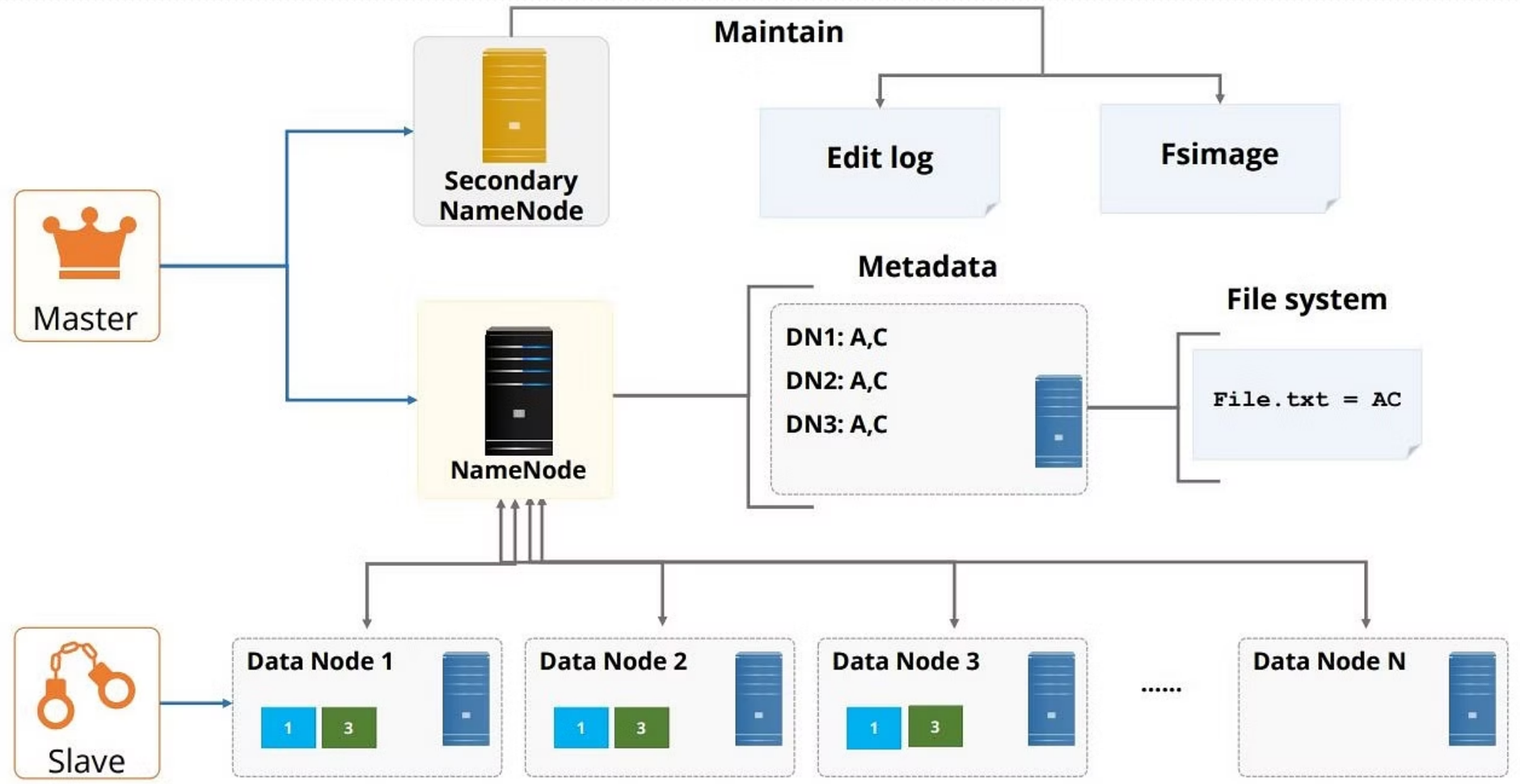
- Blocks are the smallest continuous location on your hard drive where data is stored.
- In any of the File System, you store the data as a collection of blocks.
- HDFS stores each file as blocks which are scattered throughout the Apache Hadoop cluster.
- The default size of each block is 128 MB in Apache Hadoop 2.x
- The data block approach provides:
  - Simplified replication
  - Fault-toleranceReliability.

Data Block

- Internally, HDFS split the file into block-sized chunks called a block. The size of the block is 128 Mb by default. One can configure the block size as per the requirement.



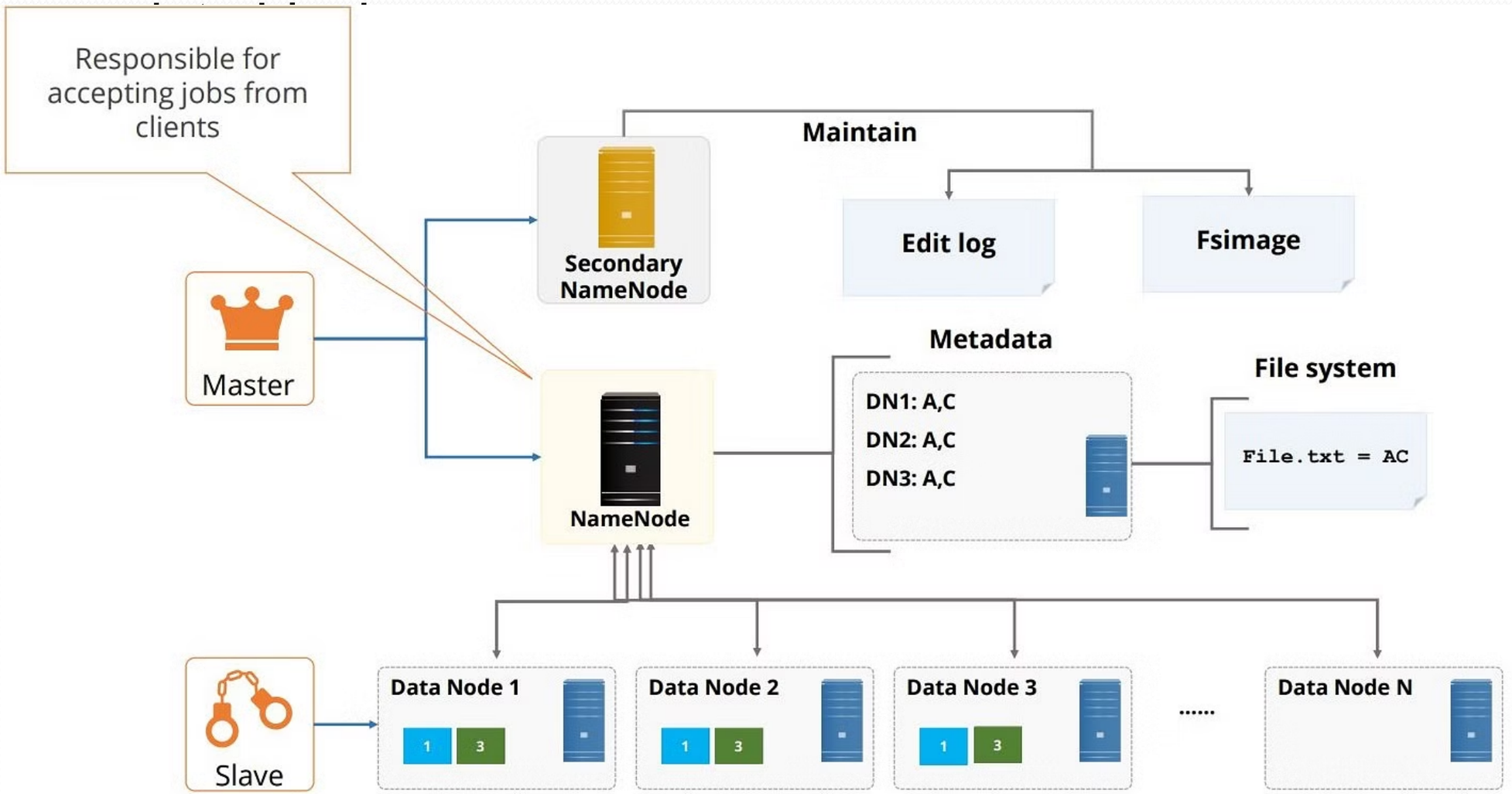
HDFS Architecture

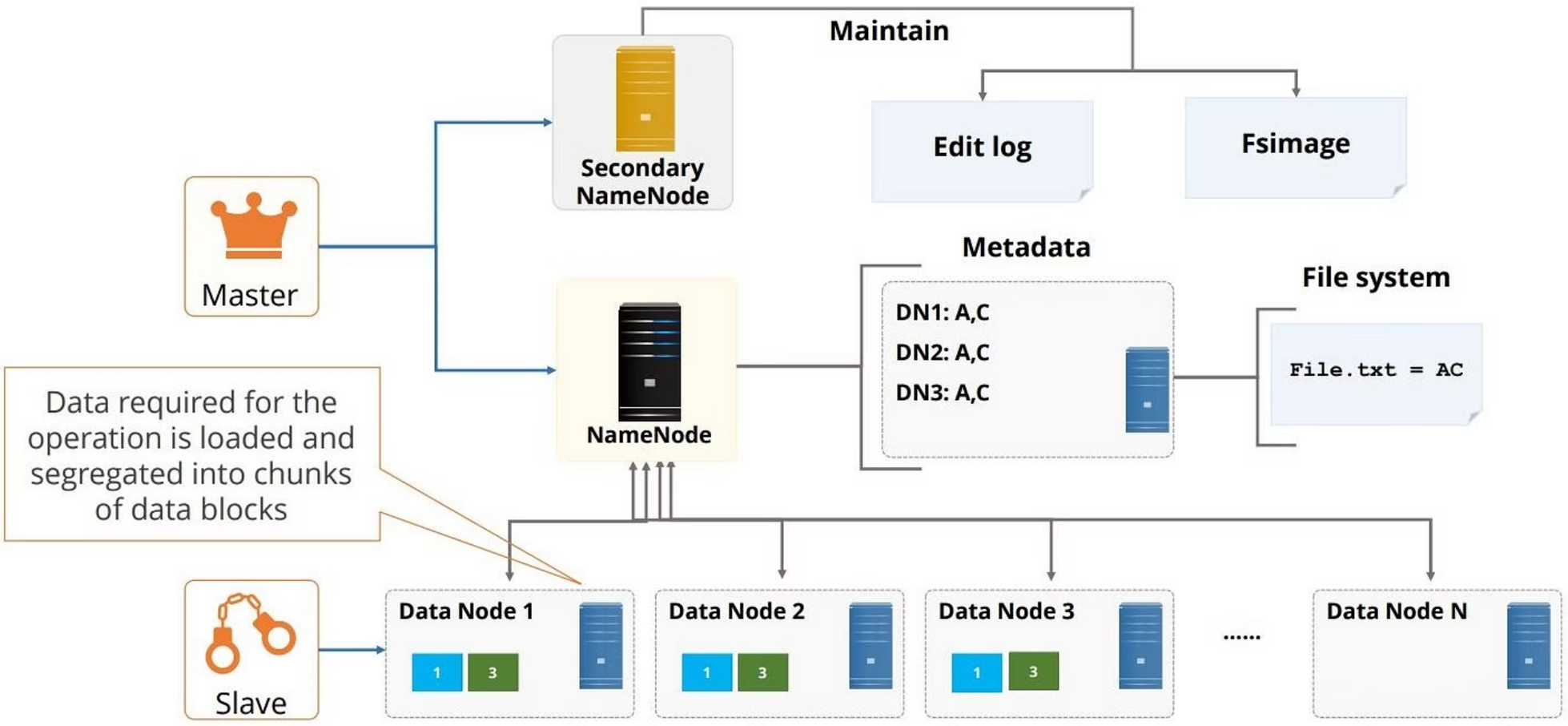


HDFS Client:

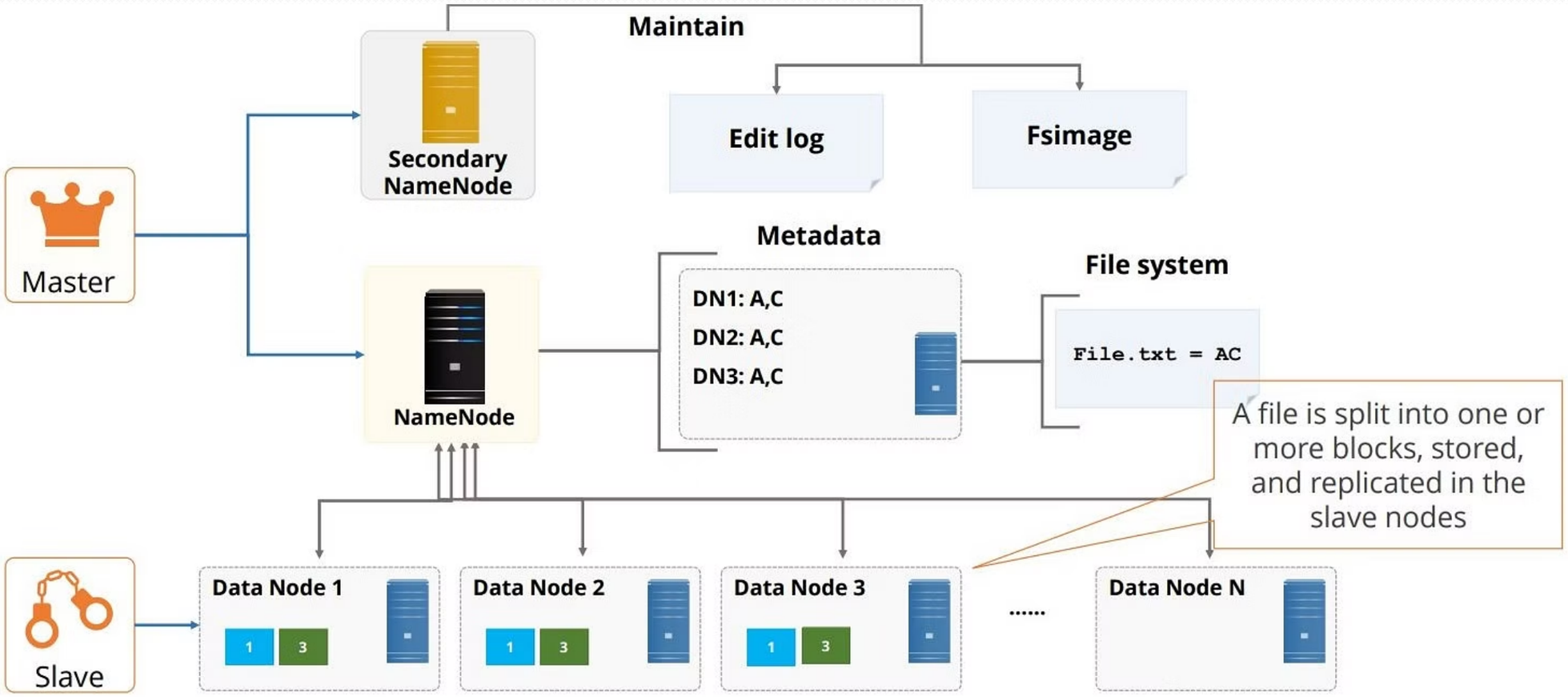
- HDFS Client is an intermediate component between HDFS and the user. It communicates with the Datanode or Namenode and fetches the essential output that the user requests.

A master node, that is the NameNode, is responsible for accepting jobs from the clients. Its task is to ensure that the data required for the operation is loaded and segregated into chunks of

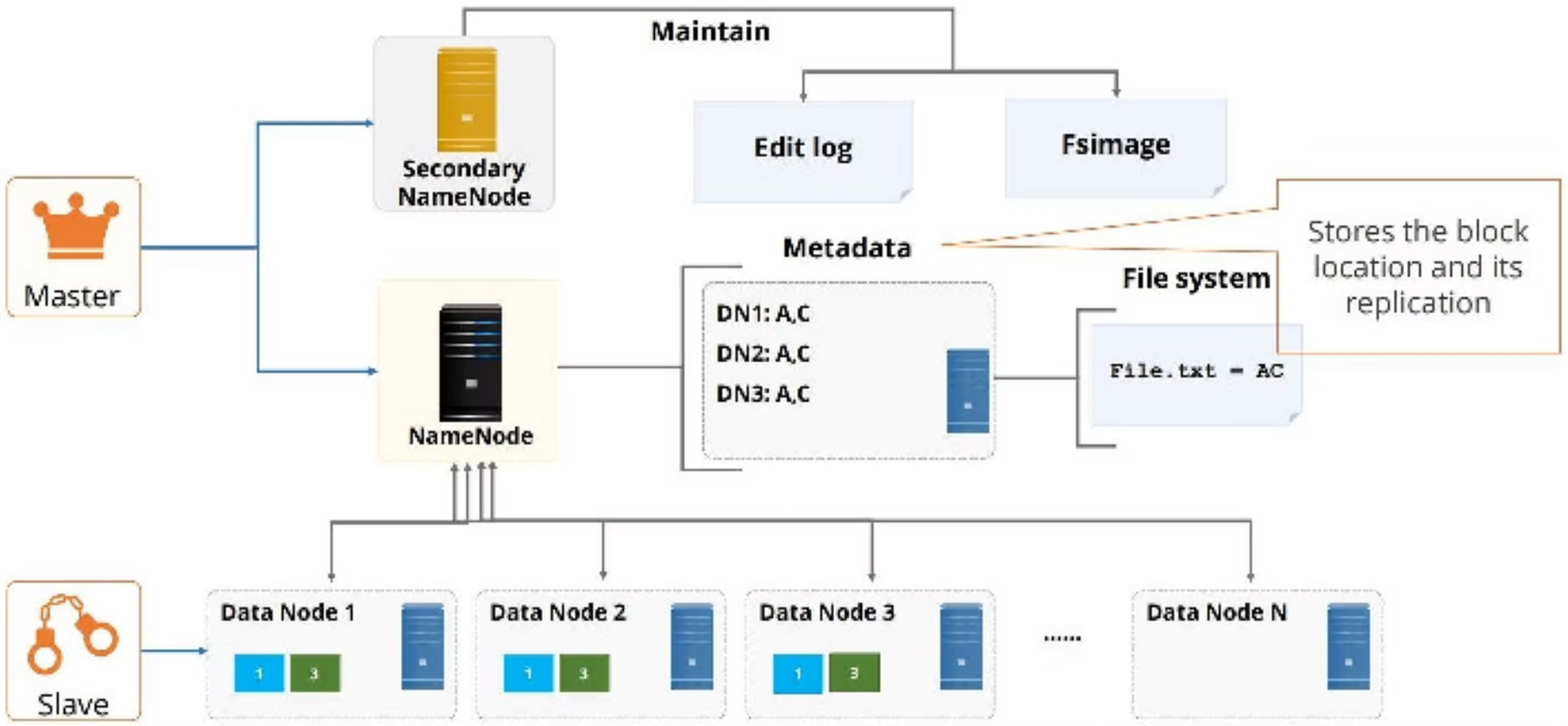



Maintain
Secondary NameNode
Edit log
Fsimage
Master
Metadata
File system
NameNode
DN1: A,C
DN2: A,C
DN3: A,C
File.txt = AC
Data required for the operation is loaded and segregated into chunks of data blocks
Slave
Data Node 1
Data Node 2
Data Node 3
.....
Data Node N
1
3
1
3
1
3

HDFS exposes a file system namespace and allows user data to be stored in files. A file is split into one or more blocks, stored, and replicated in the slave nodes known as the DataNodes as shown in the section below.



- The data blocks are then distributed to the DataNode systems within the cluster. This ensures that the replicas of the data are maintained. DataNode serves to read or write requests. It also creates, deletes, and replicates blocks on the instructions from the NameNode.



Secondary NameNode:

- Secondary NameNode works as a helper node to primary NameNode but doesn’t replace primary NameNode.

- There is a Secondary NameNode which performs tasks for NameNode and is also considered as a master node. Prior to Hadoop 2.0.0, the NameNode was a Single Point of Failure, or SPOF, in an HDFS cluster.
- Each cluster had a single NameNode. In case of an unplanned event, such as a system failure, the cluster would be unavailable until an operator restarted the NameNode.
- Also, planned maintenance events, such as software or hardware upgrades on the NameNode system, would result in cluster downtime.
- The HDFS High Availability, or HA, feature addresses these problems by providing the option of running two redundant NameNodes in the same cluster in an Active/ Passive configuration with a hot standby.

- This allows a fast failover to a new NameNode in case a system crashes or an administrator initiates a failover for the purpose of a planned maintenance.

- In an HA cluster, two separate systems are configured as NameNodes. At any instance, one of the NameNodes is in an Active state, and the other is in a Standby state.

- The Active NameNode is responsible for all client operations in the cluster, while the Standby simply acts as a slave, maintaining enough state to provide a fast failover if necessary.

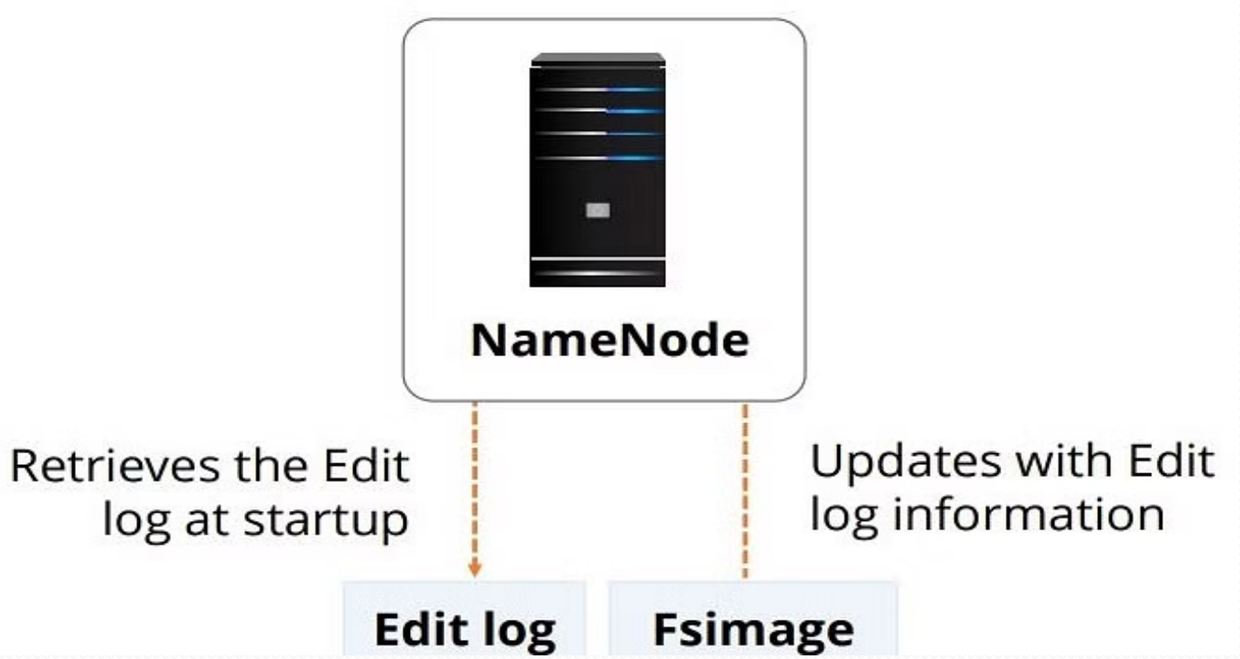
HDFS Components

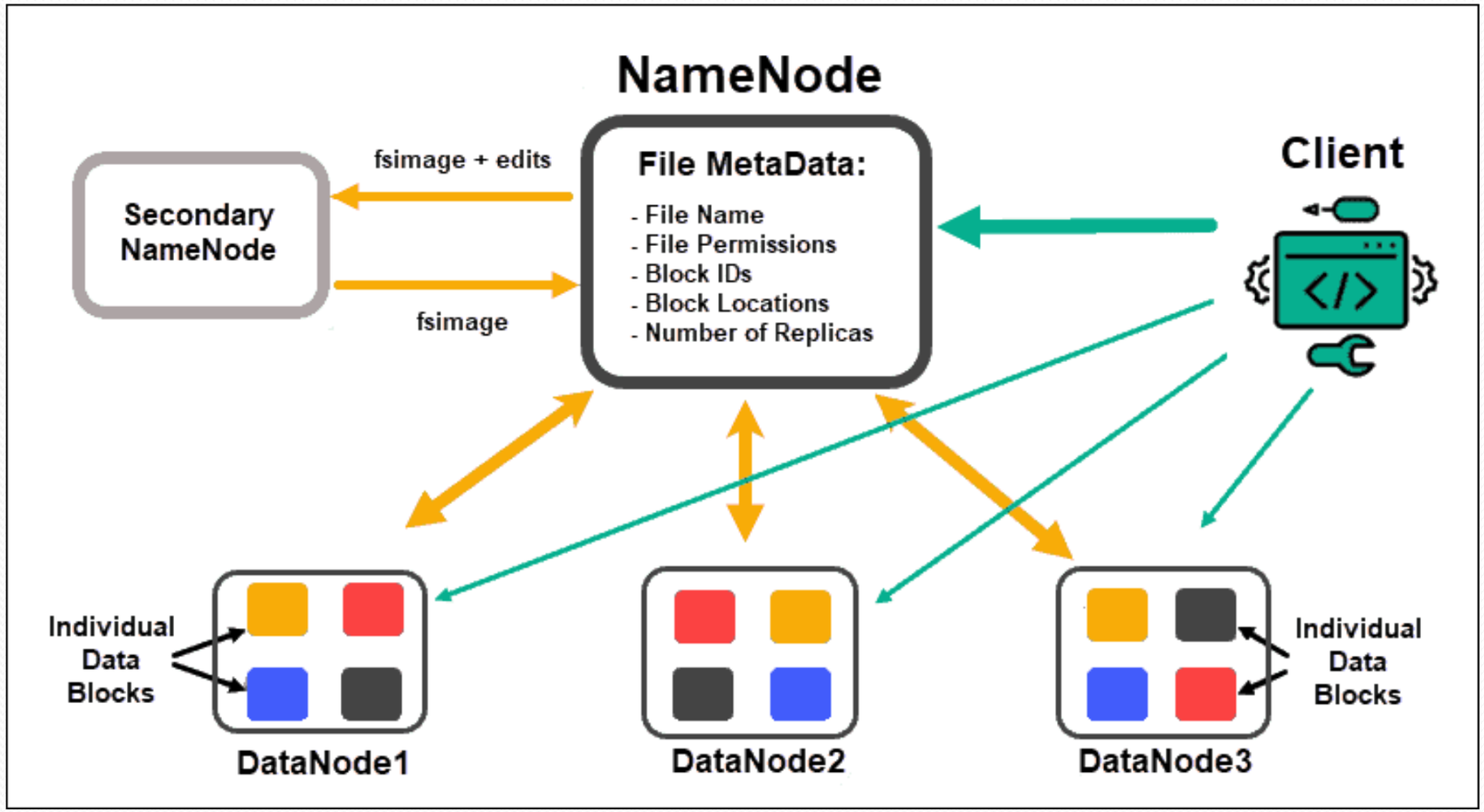
The main components of HDFS are:

- Namenode
- Secondary Namenode
- File system
- Metadata
- Datanode

Namenode

- The NameNode server is the core component of an HDFS cluster. There can be only one NameNode server in an entire cluster. Namenode maintains and executes the file system namespace operation such as opening, closing, and renaming of files and directories, which are present in HDFS.




NameNode
File MetaData:
- File Name
- File Permissions
- Block IDs
- Block Locations
- Number of Replicas
Secondary NameNode
fsimage + edits
fsimage
Client
Individual Data Blocks
DataNode1
DataNode2
DataNode3
Individual Data Blocks

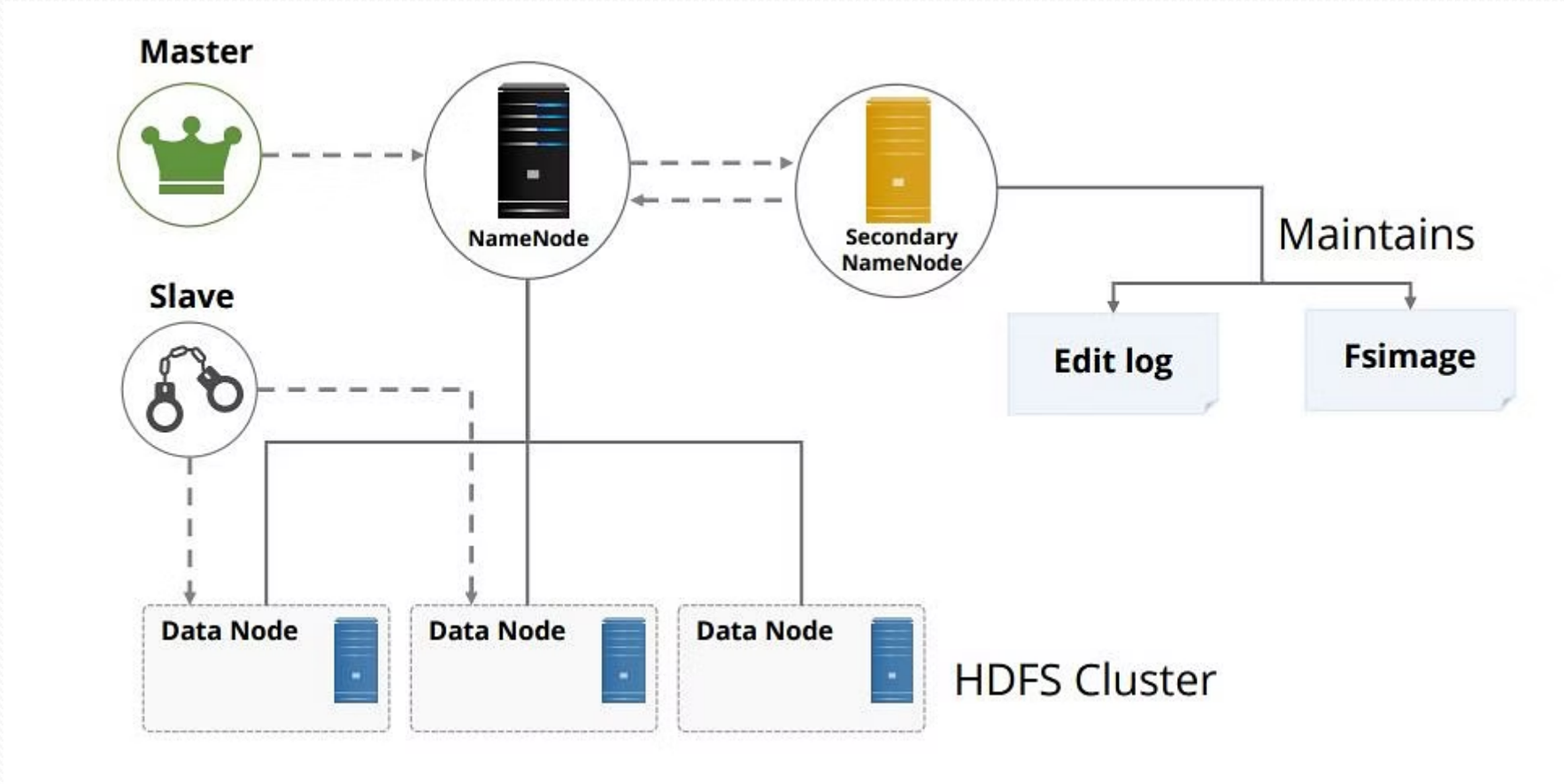
- Initially, data is broken into abstract data blocks. The file metadata for these blocks, which include the file name, file permissions, IDs, locations, and the number of replicas, are stored in a fsimage, on the NameNode local memory.

- Should a NameNode fail, HDFS would not be able to locate any of the data sets distributed throughout the DataNodes. This makes the NameNode the single point of failure for the entire cluster. This vulnerability is resolved by implementing a Secondary NameNode or a Standby NameNode.

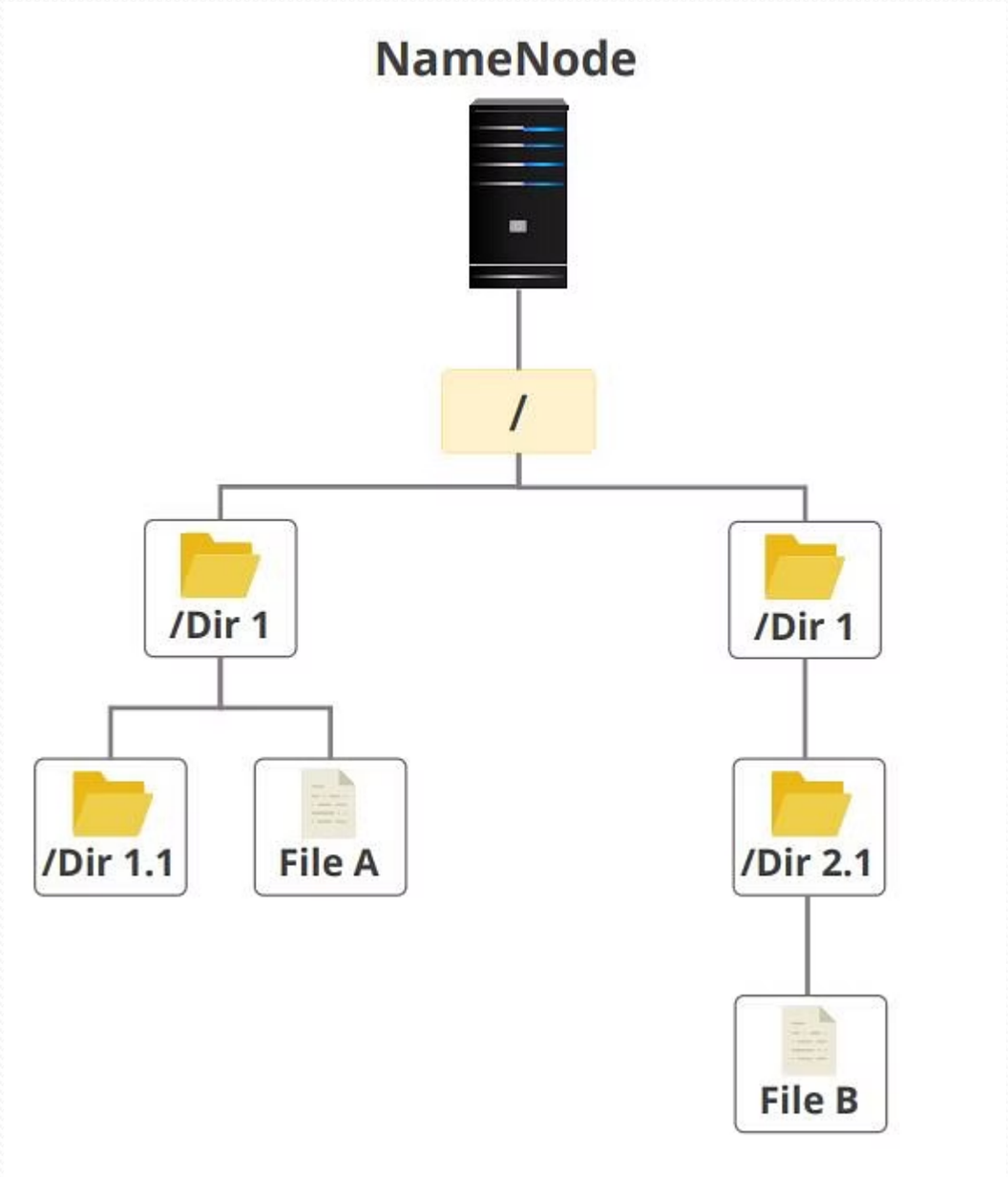
Secondary NameNode

- The Secondary NameNode served as the primary backup solution in early Hadoop versions. The Secondary NameNode, every so often, downloads the current fsimage instance and edit logs from the NameNode and merges them. The edited fsimage can then be retrieved and restored in the primary NameNode.

- The failover is not an automated process as an administrator would need to recover the data from the Secondary NameNode manually.


Master
NameNode
Secondary NameNode
Maintains
Slave
Edit log
Fsimage
Data Node
Data Node
Data Node
HDFS Cluster

File System

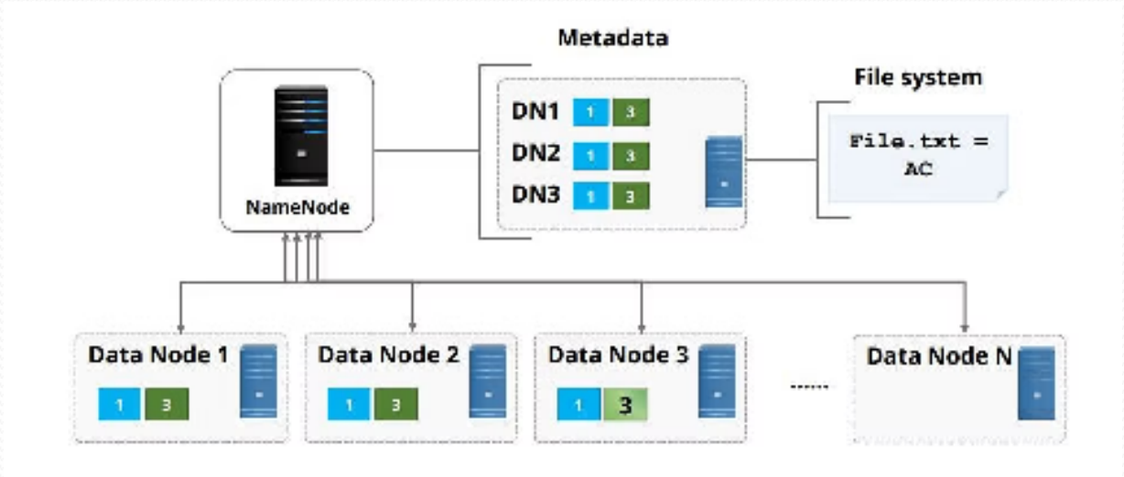


- HDFS exposes a file system namespace and allows user data to be stored in files. HDFS has a hierarchical file system with directories and files. The NameNode manages the file system namespace, allowing clients to work with files and directories.

- A file system supports operations like create, remove, move, and rename. The NameNode, apart from maintaining the file system namespace, records any change to metadata information.

Namenode: Operation

- NameNode maintains two persistent files; one a transaction log called an Edit Log and the other, a namespace image called a FsImage.
- The Edit Log records every change that occurs in the file system metadata such as creating a new file.
- The NameNode is a local filesystem that stores the Edit Log. The entire file system namespace including mapping of blocks, files, and file system properties is stored in FsImage. This is also stored in the NameNode local file system.

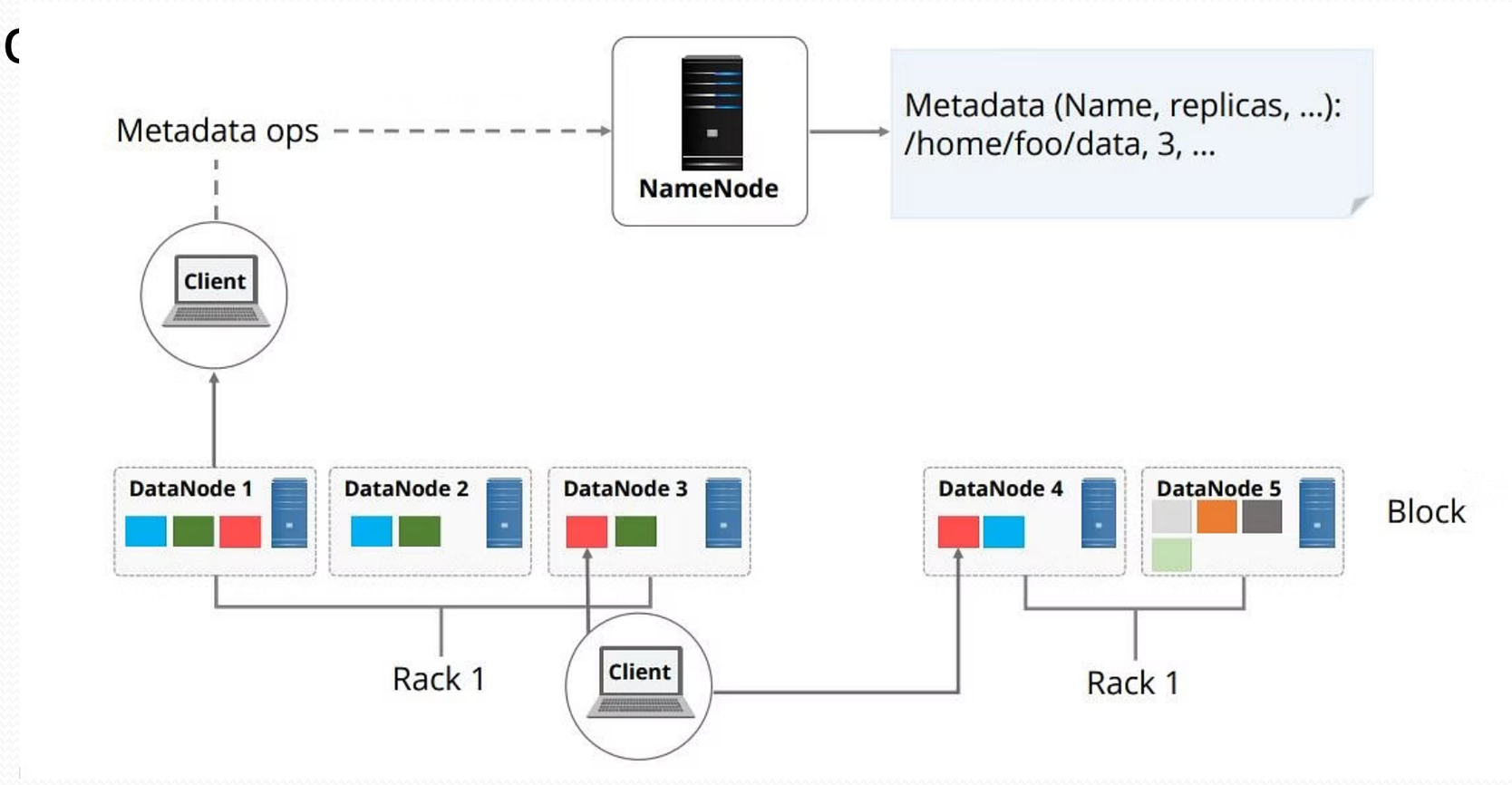


Metadata

- When new DataNodes join a cluster, metadata loads the blocks that reside on a specific DataNode into its memory at startup. Metadata then periodically loads the data at user-defined or default intervals.
- When the NameNode starts up, it retrieves the Edit Log and FsImage from its local file system. It then updates the FsImage with Edit Log information and stores a copy of the FsImage on the file system as a checkpoint.
- The metadata size is limited to the RAM available on the NameNode. A large number of small files would require more metadata than a small number of large files. Hence, the in-memory metadata management issue explains why HDFS favors a small number of large files.
- If a NameNode runs out of RAM, it will crash, and the applications will not be able to use HDFS until the NameNode is operational again.
- Data block split is an important process of HDFS architectureeach file is split into one or more blocks stored and replicated in DataNodes.

DataNode

- DataNodes manage names and locations of file blocks. By default, each file block is 128 Megabytes. However, this potentially reduces the amount of parallelism that can be achieved as the number of blocks per file dec



Functions of DataNode

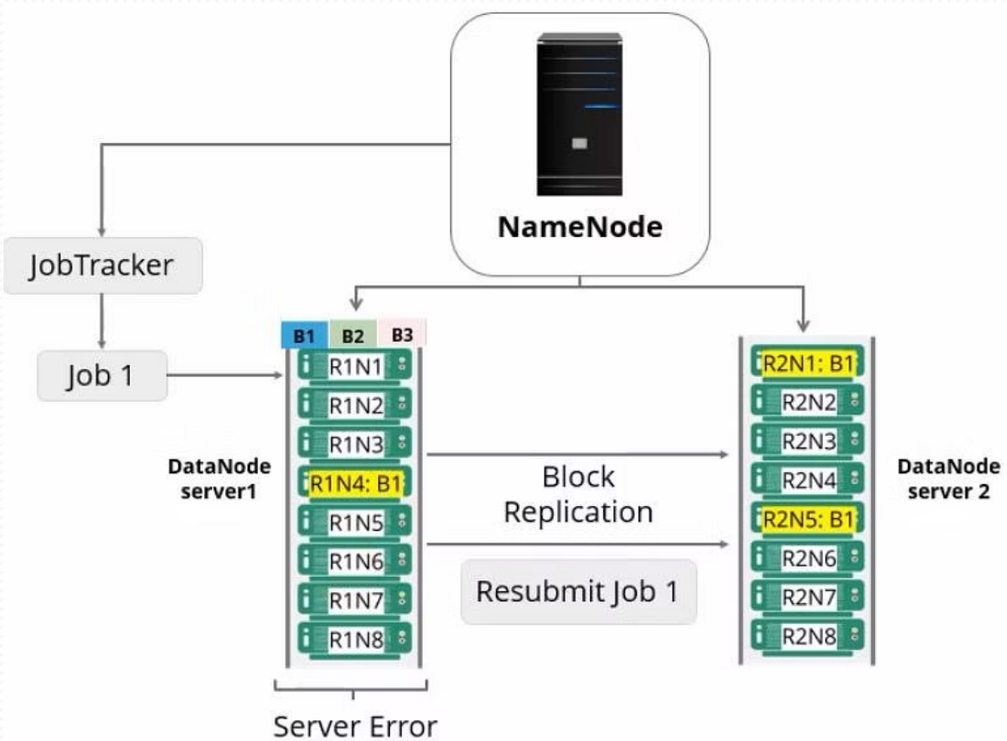
- DataNode where HDFS stores the actual data.
- DataNode is responsible for serving the client read/write requests.
- Based on the instruction from the NameNode, DataNodes performs block creation, replication, and deletion.
- DataNodes send a heartbeat to NameNode to report the health of HDFS.
- DataNodes also sends block reports to NameNode to report the list of blocks it contains.

- The data block approach provides:

- Simplified replication
- Fault-tolerance
- Reliability.
- It also helps by shielding users from storage sub-system details.

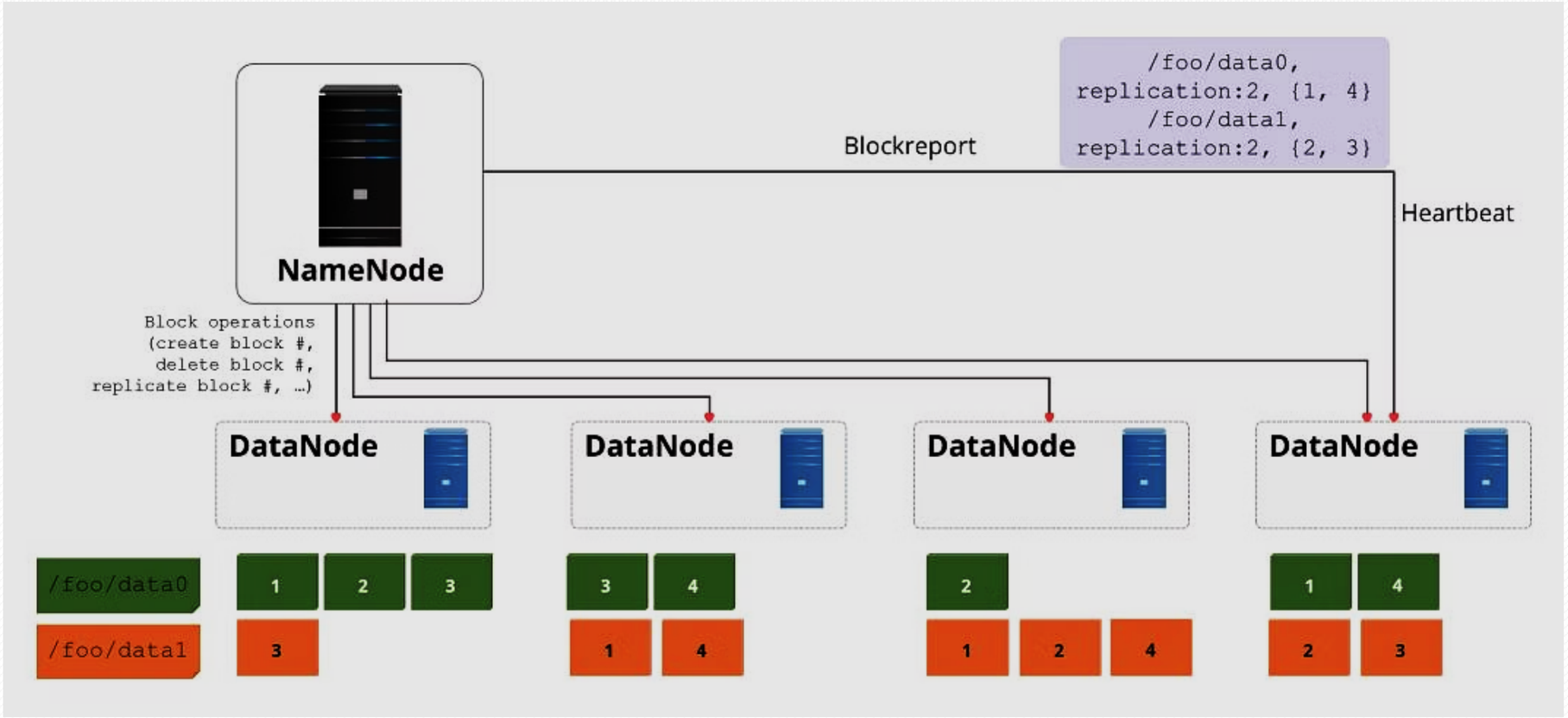
Block Replication Architecture

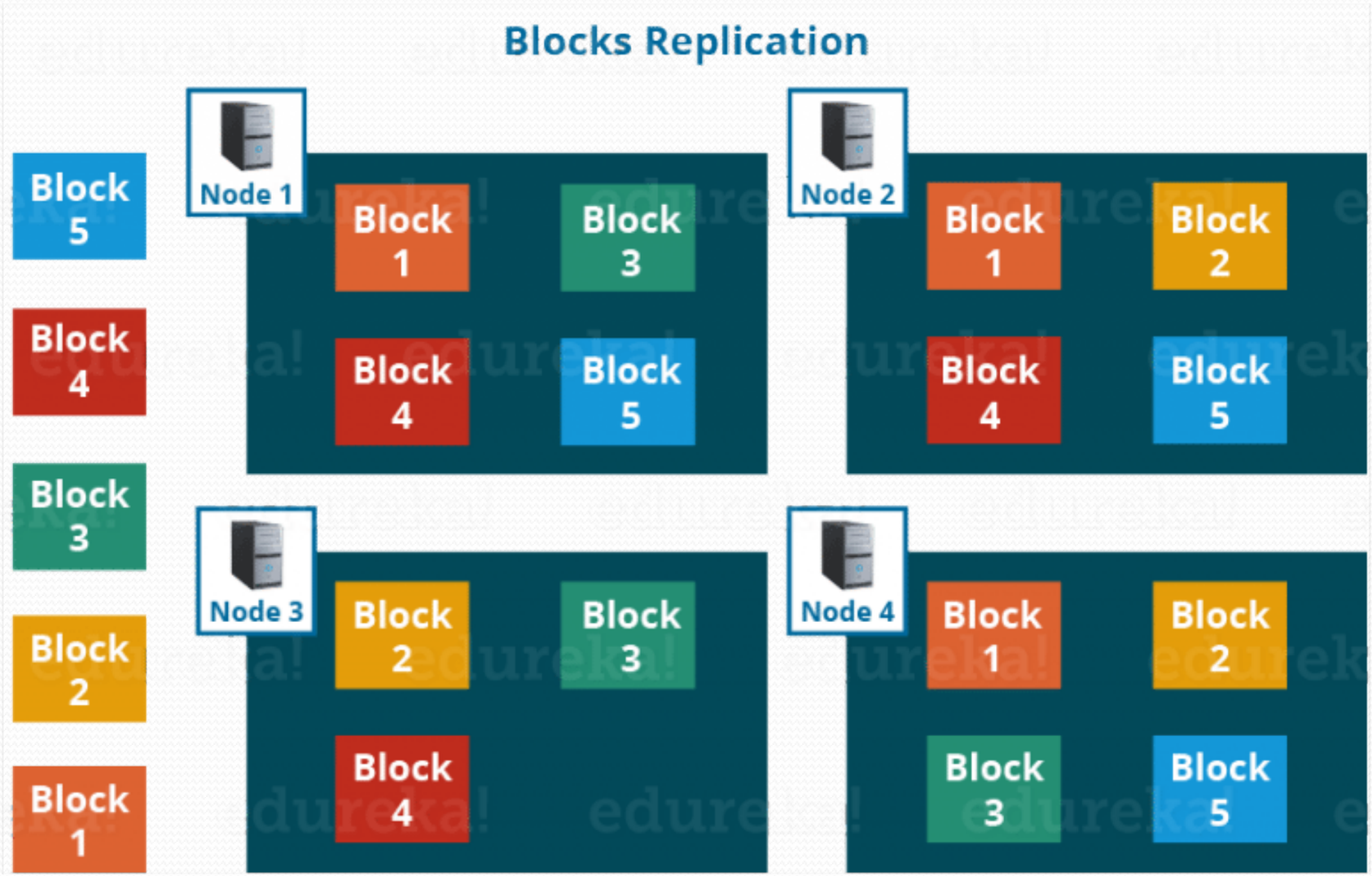
- Block replication refers to creating copies of a block in multiple data nodes. Usually, the data is split into the forms of parts such as part and part one.



Replication Method

- In the replication method, each file is split into a sequence of blocks. All blocks except the last one in the file are of the same size. Blocks are replicated for fault tolerance.





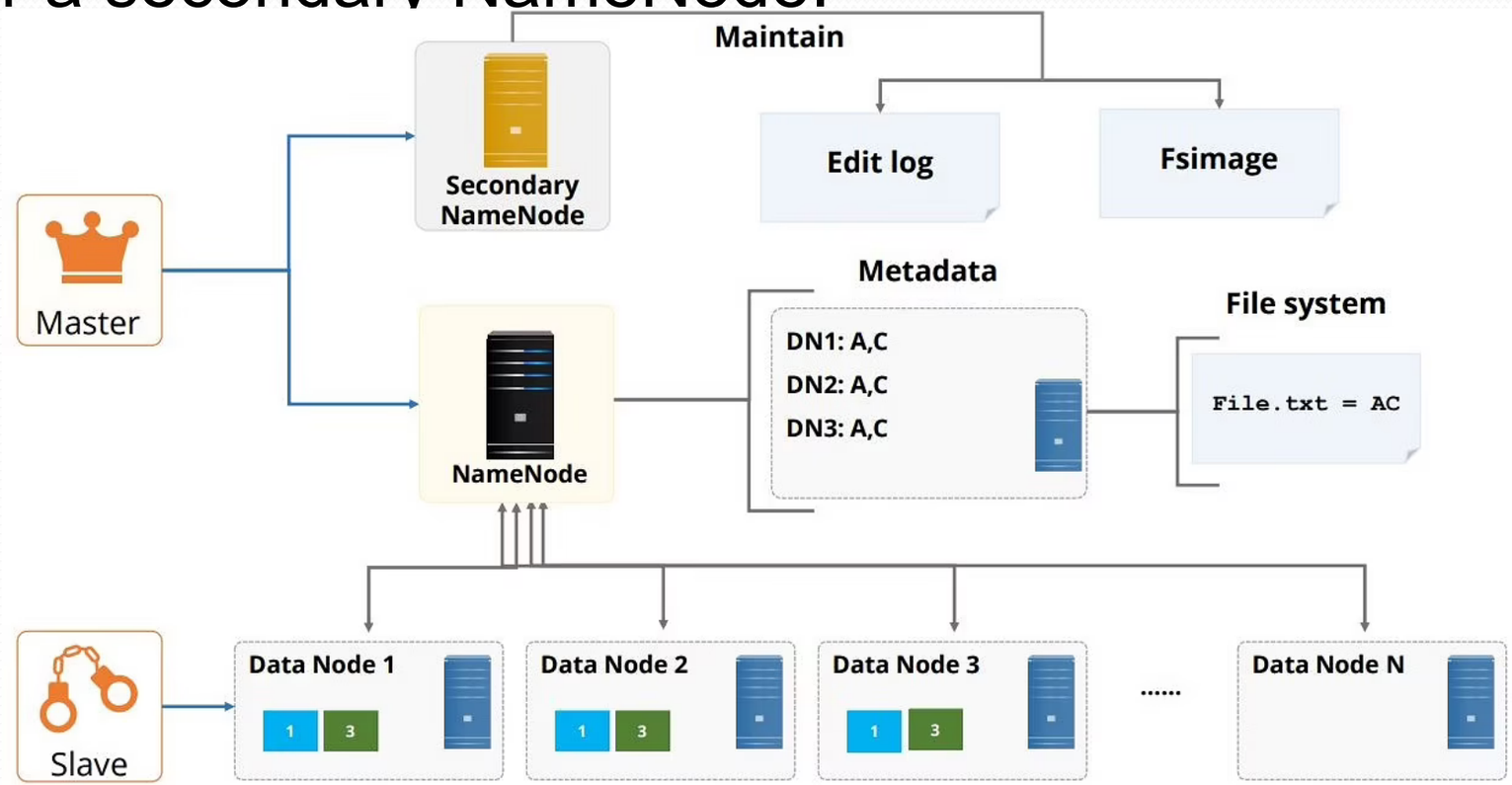
Each block is replicated three times and stored on different DataNodes

HDFS Components

- The main components of HDFS are:
- Namenode
- Secondary Namenode
- File system
- Metadata
- Datanode

HDFS Architecture and Components

- NameNode is the master server. In a non-high availability cluster, there can be only one NameNode.
- In a high availability cluster, there is a possibility of two NameNodes, and if there are two NameNodes there is no need for a secondary NameNode.



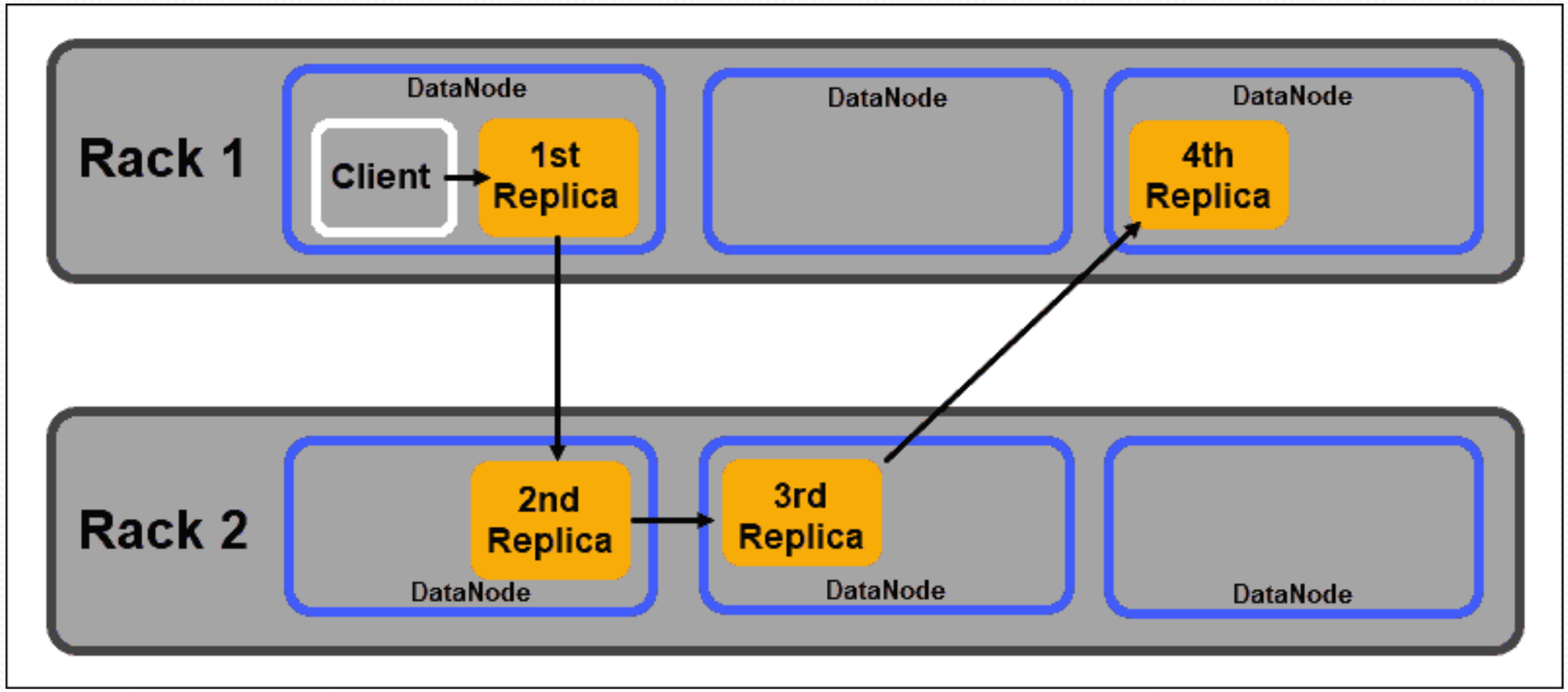
- The block replication factor is usually configured at the cluster level but it can also be configured at the file level.

- The name node receives a heartbeat and a block report from each data node in the cluster. The heartbeat denotes that the data node is functioning properly. A block report lists the blocks on a data node.

Rack Aware Placement Policy

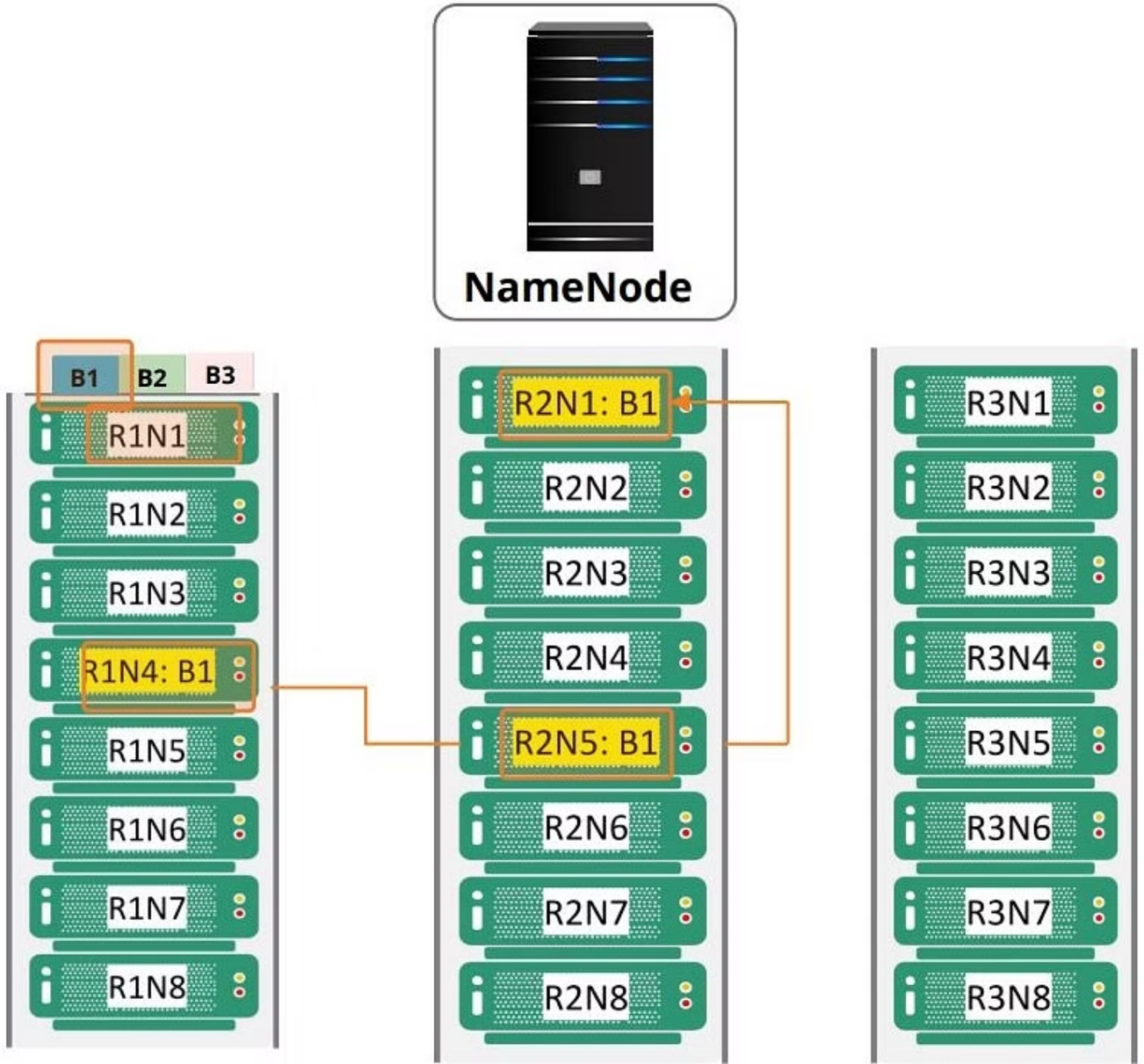
- One of the main objectives of a distributed storage system like HDFS is to maintain high availability and replication. Therefore, data blocks need to be distributed not only on different DataNodes but on nodes located on different server racks.

- This ensures that the failure of an entire rack does not terminate all data replicas. The HDFS NameNode maintains a default rack-aware replica placement policy:
  - The first data block replica is placed on the same node as the client.
  - The second replica is automatically placed on a random DataNode on a different rack.
  - The third replica is placed in a separate DataNode on the same rack as the second replica.
  - Any additional replicas are stored on random DataNodes throughout the cluster.


Rack 1
DataNode
Client
1st Replica
DataNode
DataNode
4th Replica
Rack 2
2nd Replica
DataNode
3rd Replica
DataNode
DataNode

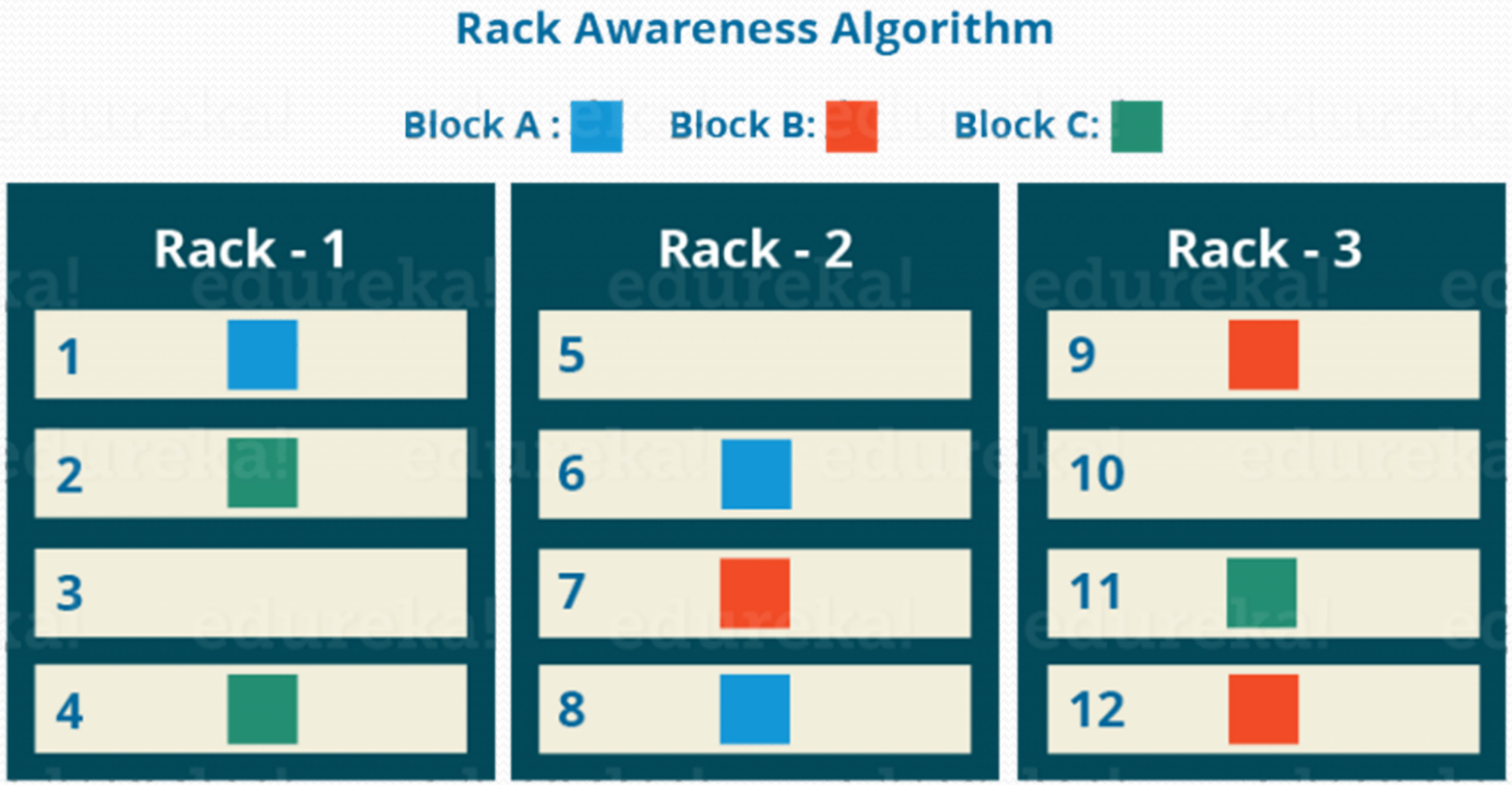
Data Replication Topology - Example

- The diagram illustrates a Hadoop cluster with three racks. A diagram for Replication and Rack Awareness in Hadoop,



- Each rack consists of multiple nodes. R1N1 represents node 1 on rack 1. Suppose each rack has eight nodes. The name node decides which data node belongs to which rack. Block 1 which is B1 is first written to node 4 on rack 1.
- A copy is then written to a different node on a different rack which is node 5 on rack 2. The third and final copy of the block is written to the same rack of the second copy but to a different node which is rack 2 node 1.

- Suppose if the replication factor is 3, then according to the rack awareness algorithm:
  - The first replica will get stored on the local rack.
  - The second replica will get stored on the other DataNode in the same rack.
  - The third replica will get stored on a different rack.



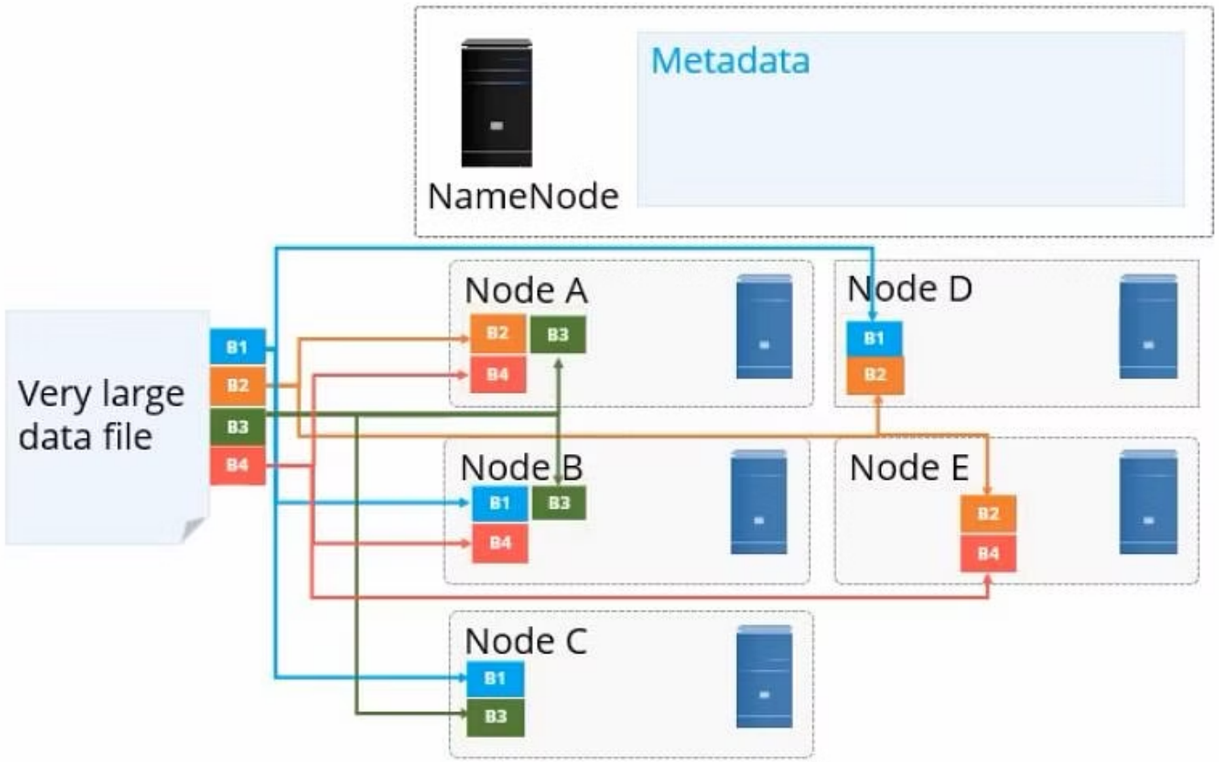
The File System Namespace

- HDFS supports a traditional hierarchical file organization. A user or an application can create directories and store files inside these directories.
- The file system namespace hierarchy is similar to most other existing file systems; one can create and remove files, move a file from one directory to another, or rename a file. HDFS does not yet implement user quotas. HDFS does not support hard links or soft links. However, the HDFS architecture does not preclude implementing these features.
- The NameNode maintains the file system namespace. Any change to the file system namespace or its properties is recorded by the NameNode. An application can specify the number of replicas of a file that should be maintained by HDFS. The number of copies of a file is called the replication factor of that file. This information is stored by the NameNode.

- HDFS is designed to reliably store very large files across machines in a large cluster.
- It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size.
- The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file.
- An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

How Are Files Stored?

- Let's say we have a large data file that is divided into four blocks. Each block is replicated three times as shown in the diagram. default size of each block is 128 megabytes. The name node then carries metadata information of all blocks and its distribution

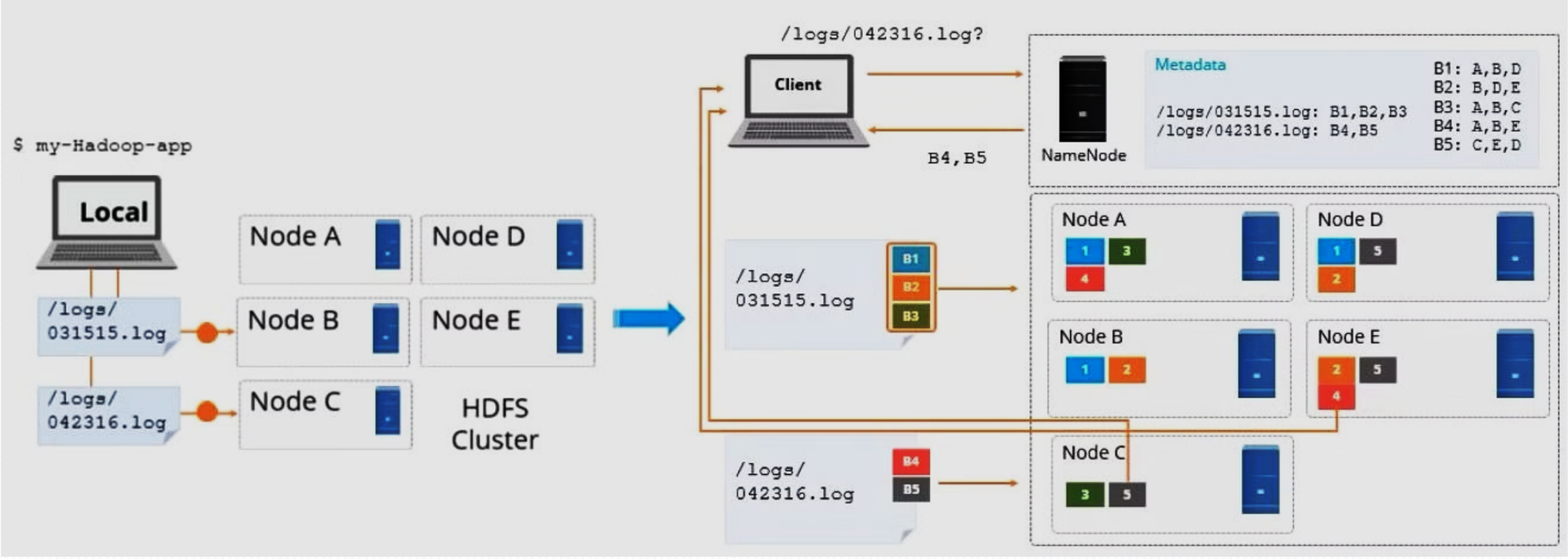


- Name node then carries metadata information of all blocks and its distribution. Let's work on an example of HDFS which gives a deep understanding of all the points discussed so far.

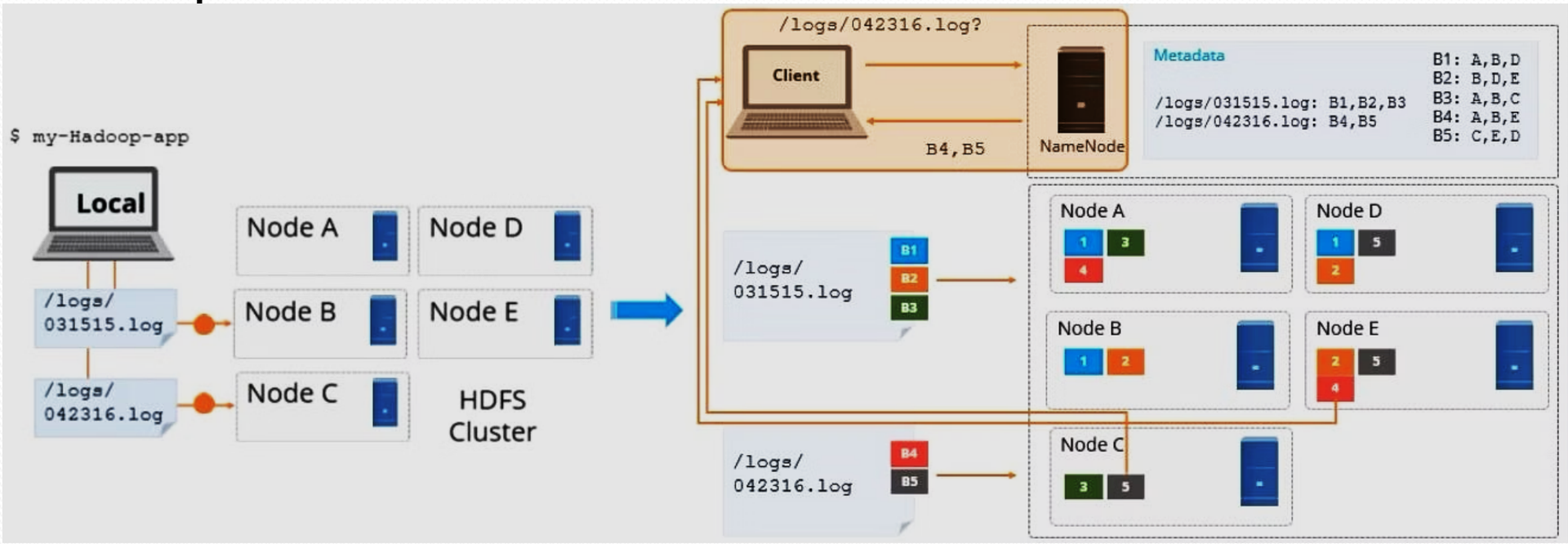
HDFS - Example

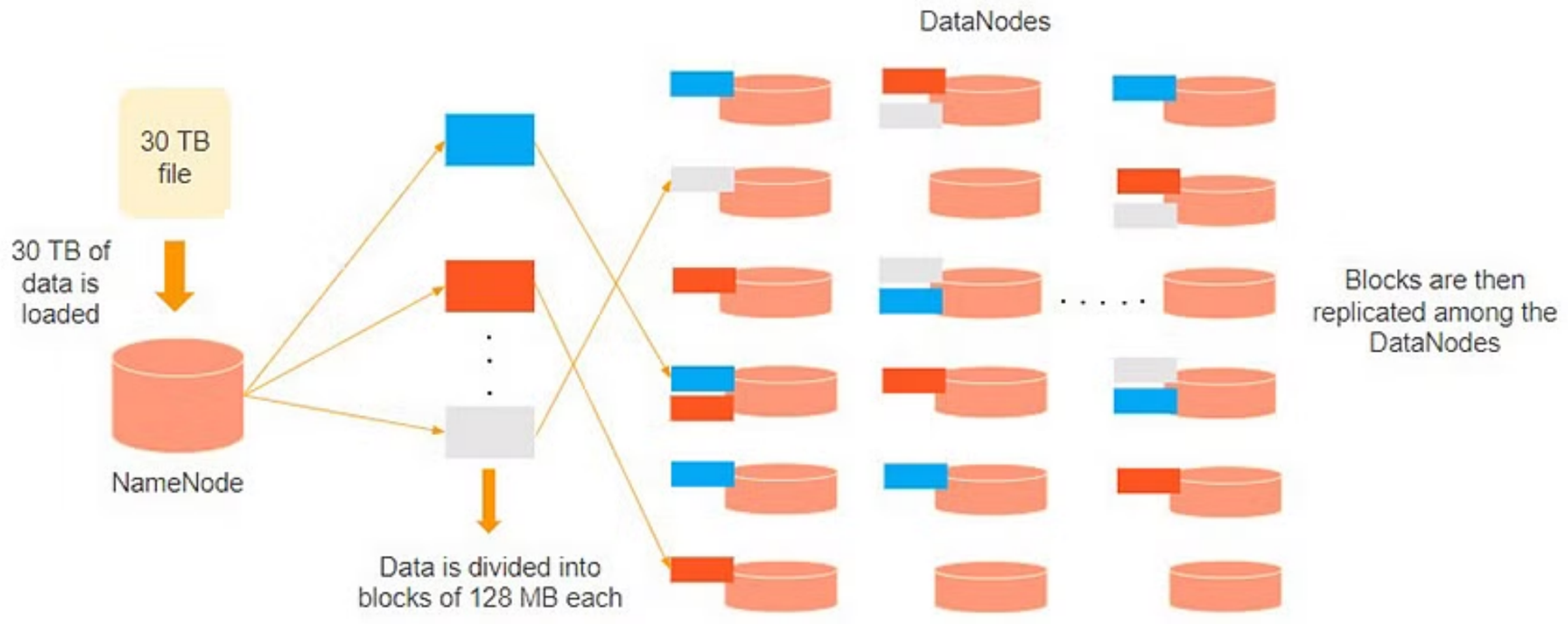
- Suppose you have 2 log files that you want to save from a local file system to the HDFS cluster.
- The cluster has 5 data nodes: node A, node B, node C, node D, and node E.
- Now the first log is divided into three blocks: b1 b2 and b3 and the other log is divided into two blocks: b4 and b5.
- Now the blocks b1 b2 b3 b4 and b5 are distributed to the node A, node B, node C, and no D respectively as shown in the diagram.

- Each block will also be replicated three times on the five data nodes. All of the information related to the list of blocks and the replication known as the metadata information of the five blocks will be stored in namenode.



- Now suppose the client asks for one log that you have stored. The inquiry goes to the namenode and the client gets the information about this log file as shown in the diagram.
- Based on the information from the namenode, the client receives the file information from the respective data nodes.





- Consider that 30TB of data is loaded into the name node. The name node distributes it across the data nodes
- This data is replicated among the data nodes.
- The blue, grey, and red data are replicated among the three data nodes.

Replication Management

- For a distributed system, the data must be redundant to multiple places so that if one machine fails, the data is accessible from other machines.
- In Hadoop, HDFS stores replicas of a block on multiple DataNodes based on the replication factor makes HDFS fault-tolerant.
- The replication factor is the number of copies to be created for blocks of a file in HDFS architecture.
- If the replication factor is 3, then three copies of a block get stored on different DataNodes. So if one DataNode containing the data block fails, then the block is accessible from the other DataNode containing a replica of the block.
- If we are storing a file of 128 Mb and the replication factor is 3, then (3*128=384) 384 Mb of disk space is occupied for a file as three copies of a block get stored.

Rack

- In a large Hadoop cluster, there are multiple racks. Each rack consists of DataNodes. Communication between the DataNodes on the same rack is more efficient as compared to the communication between DataNodes residing on different racks.
- Communication between the DataNodes on the same rack is more efficient as compared to the communication between DataNodes residing on different racks.

Rack Awareness

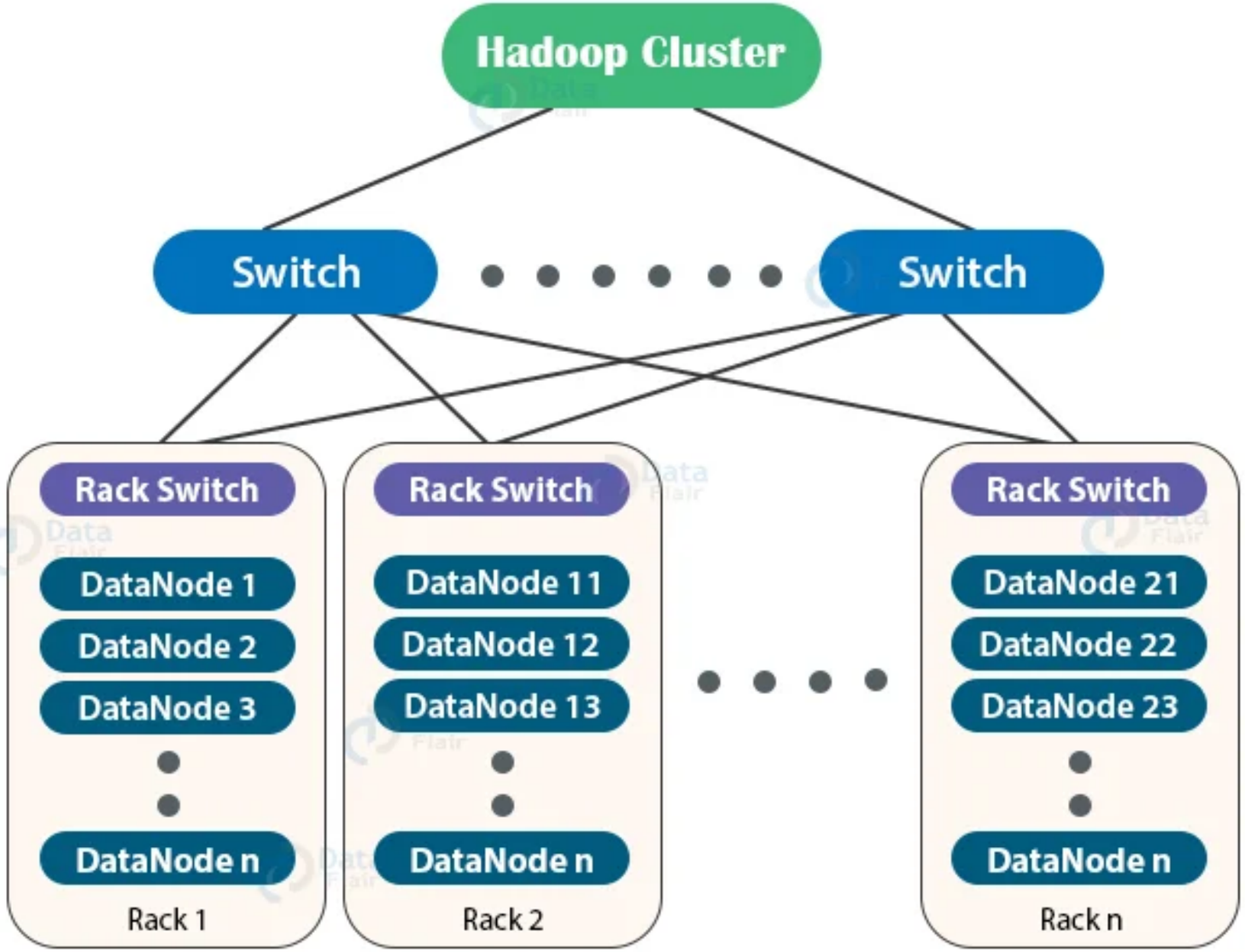
- Rack Awareness is the concept of choosing the closest node based on the rack information.
- To ensure that all the replicas of a block are not stored on the same rack or a single rack, NameNode follows a rack awareness algorithm to store replicas and provide latency and fault tolerance.
- Suppose if the replication factor is 3, then according to the rack awareness algorithm:
  - The first replica will get stored on the local rack.
  - The second replica will get stored on the other DataNode in the same rack.
  - The third replica will get stored on a different rack.

Reasons for the Rack Awareness

- To reduce the network traffic while file read/write, which improves the cluster performance.
- To achieve fault tolerance, even when the rack goes down.
- Achieve high availability of data so that data is available even in unfavorable conditions.
- To reduce the latency, that is, to make the file read/ write operations done with lower delay.

Rack Awareness

- If we store replicas on different nodes on the same rack, then it improves the network bandwidth, but if the rack fails (rarely happens), then there will be no copy of data on another rack.
- If we store replicas on unique racks, then due to the transfer of blocks to multiple racks while writes increase the cost of writes.
- Therefore, NameNode on multiple rack cluster maintains block replication by using inbuilt Rack awareness policies which says:
  - Not more than one replica be placed on one node.
  - Not more than two replicas are placed on the same rack.
  - The number of racks used for block replication should always be smaller than the number of replicas.


Hadoop Cluster
Switch
Switch
Rack Switch
DataNode 1
DataNode 2
DataNode 3
DataNode n
Rack 1
Rack Switch
DataNode 11
DataNode 12
DataNode 13
DataNode n
Rack 2
Rack Switch
DataNode 21
DataNode 22
DataNode 23
DataNode n
Rack n

Advantages of Rack Awareness:

- **To improve the network performance:** The communication between nodes residing on different racks is directed via switch. In general, you will find *greater network bandwidth* between machines in the same rack than the machines residing in different rack. So, the Rack Awareness helps you to have reduce write traffic in between different racks and thus providing a better write performance. Also, you will be gaining increased read performance because you are using the bandwidth of multiple racks.
- **To prevent loss of data:** We don't have to worry about the data even if an entire rack fails because of the switch failure or power failure.

HDFS Read operation

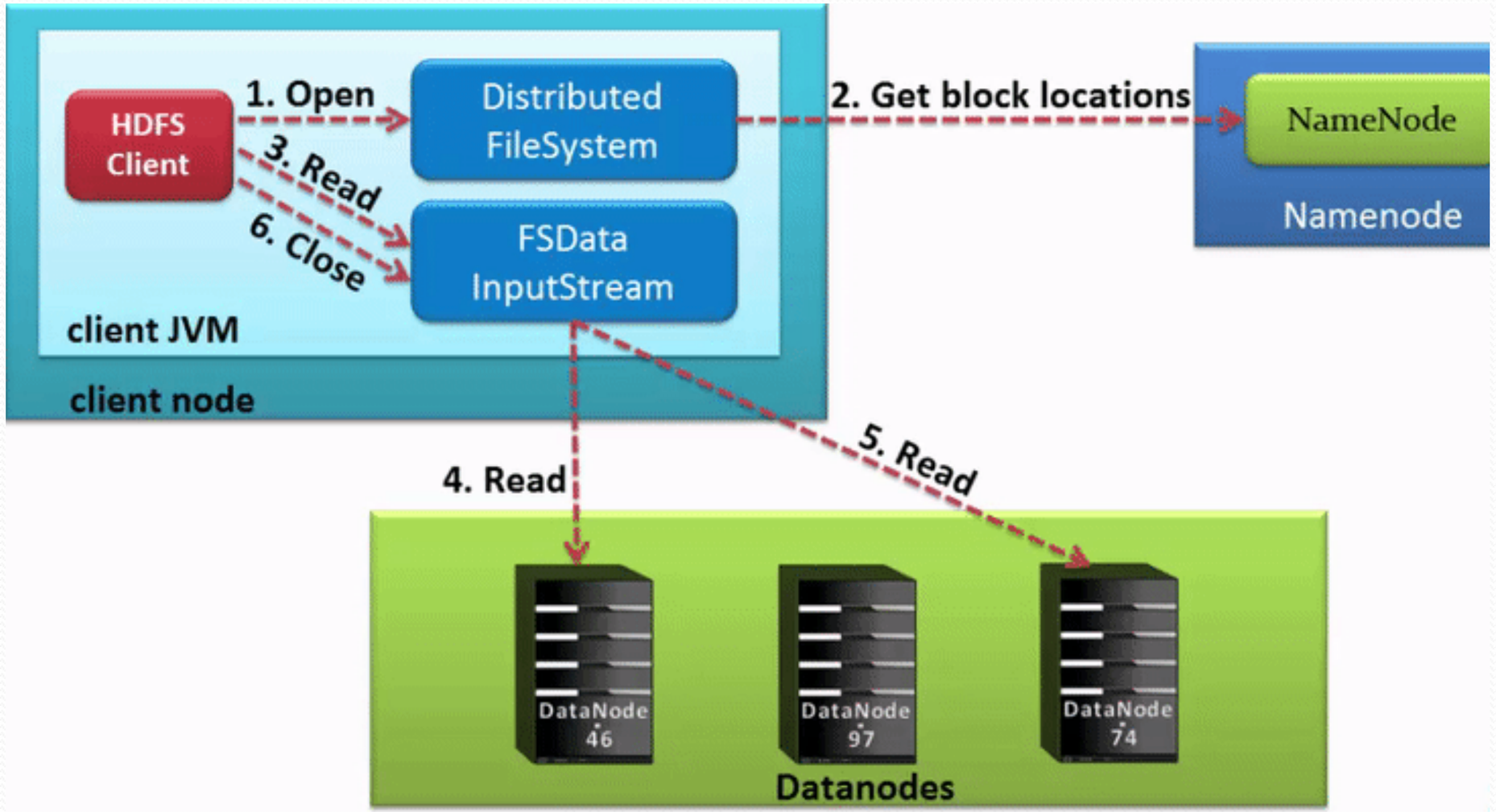
HDFS follows **Write Once Read Many** philosophies. So we cannot edit files already stored in HDFS, but we can append new data to these files by re-opening them.

To read the files stored in HDFS, the HDFS client interacts with the NameNode and DataNode.

- **HDFS Client**: On user behalf, HDFS client interacts with NameNode and Datanode to fulfill user requests.
- **NameNode:** NameNode is the master node that stores metadata about block locations, blocks of a file, etc. This metadata is used for file read and write operation.
- **DataNode:** DataNodes are the slave nodes in HDFS. They store actual data (data blocks).

Read operation in HDFS

- While reading data from HDFS, we need to get the information about the location where our respective data is stored from NameNode.
- The NameNode provides a handle to read data from data nodes.
- NameNode also provides an authentication token to the client so the only client with an authentication token can read data from the data node.


HDFS Client
1. Open
Distributed FileSystem
2. Get block locations
NameNode
Namenode
3. Read
6. Close
FSData InputStream
client JVM
client node
4. Read
5. Read
DataNode 46
DataNode 97
DataNode 74
Datanodes

Step1:

- Client Node needs to interacts with the NameNode to get the information of the data nodes. For that client will send an open request to the Distributed file system.

Step 2:

- The distributed file system will talk with the NameNode and get's the authorization token and the location of the nodes.

Step 3:

- The client will send a read request to the common JAVA input stream to read data from the data nodes.

Step 4:

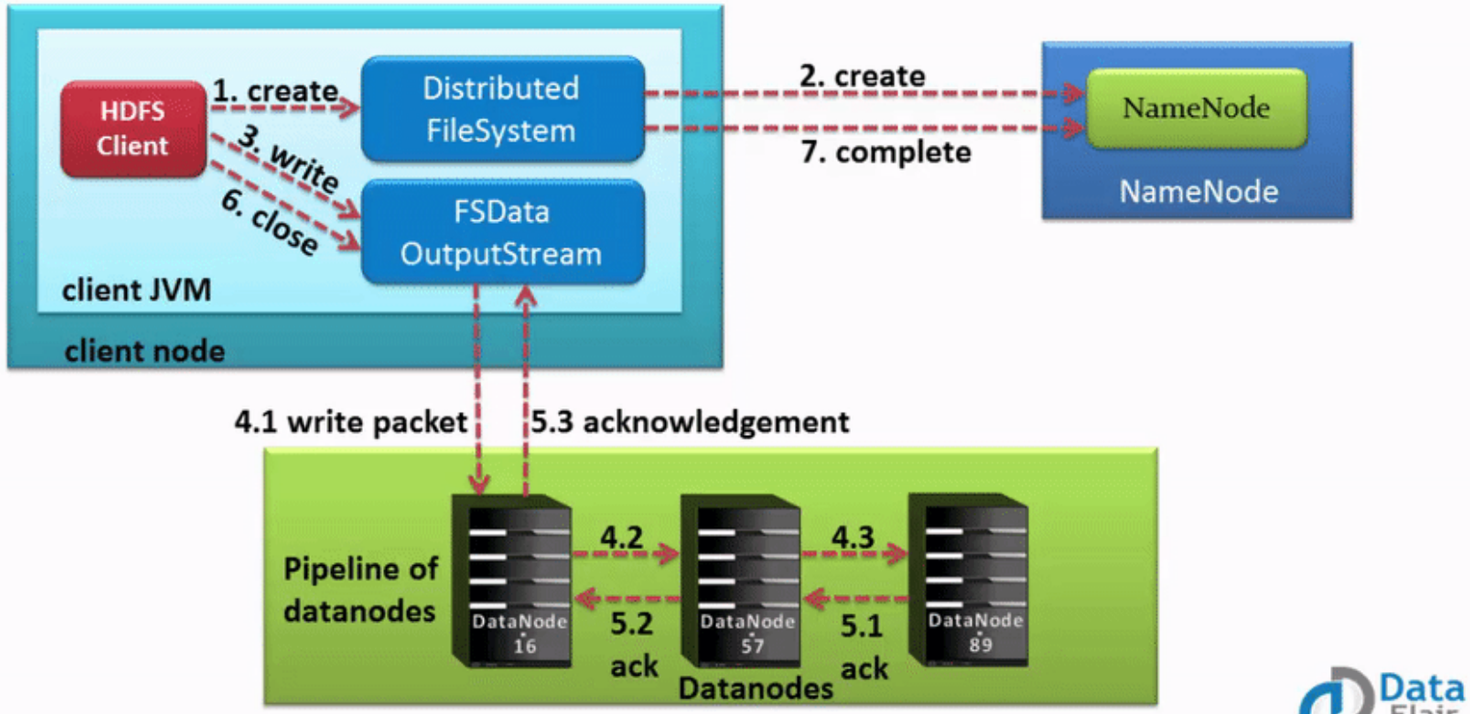
- The input stream will read the data from the data nodes.

Step 5:

- The client after getting data will send the Close command to the input stream.

HDFS Data Write Operation

- Let us assume a situation where an HDFS client, wants to write a file named “ashok.txt” of size 250 MB. Assume that the system block size is configured for 128 MB (default). So, the client will be dividing the file “ashok.txt” into 2 blocks – one of 128 MB (Block A) and the other of 122 MB (block B).


HDFS Client
1. create
Distributed FileSystem
2. create
NameNode
7. complete
NameNode
3. write
6. close
FSData OutputStream
client JVM
client node
4.1 write packet
5.3 acknowledgement
Pipeline of datanodes
DataNode 16
4.2
DataNode 57
4.3
DataNode 89
5.2 ack
5.1 ack
Datanodes

Data Flair

Now, the following protocol will be followed whenever the data is written into HDFS

1. At first, the HDFS client will reach out to the NameNode for a Write Request against the two blocks, say, Block A and Block B.

2. The NameNode will then grant the client the write permission and will provide the IP addresses of the DataNodes where the file blocks will be copied eventually.

3. The selection of IP addresses of DataNodes is purely randomized based on availability, replication factor and rack awareness.

4. Let's say the replication factor is set to default i.e. 3. Therefore, for each block the NameNode will be providing the client a list of (3) IP addresses of DataNodes. The list will be unique for each block.

- Suppose, the NameNode provided following lists of IP addresses to the client
- For Block A, list A = {IP of DataNode 1, IP of DataNode 4, IP of DataNode 6}
- For Block B, list B = {IP of DataNode 3, IP of DataNode 7, IP of DataNode 9}
- 6. Each block will be copied in three different DataNodes to maintain the replication factor consistent throughout the cluster.
- 7. Now the whole data copy process will happen