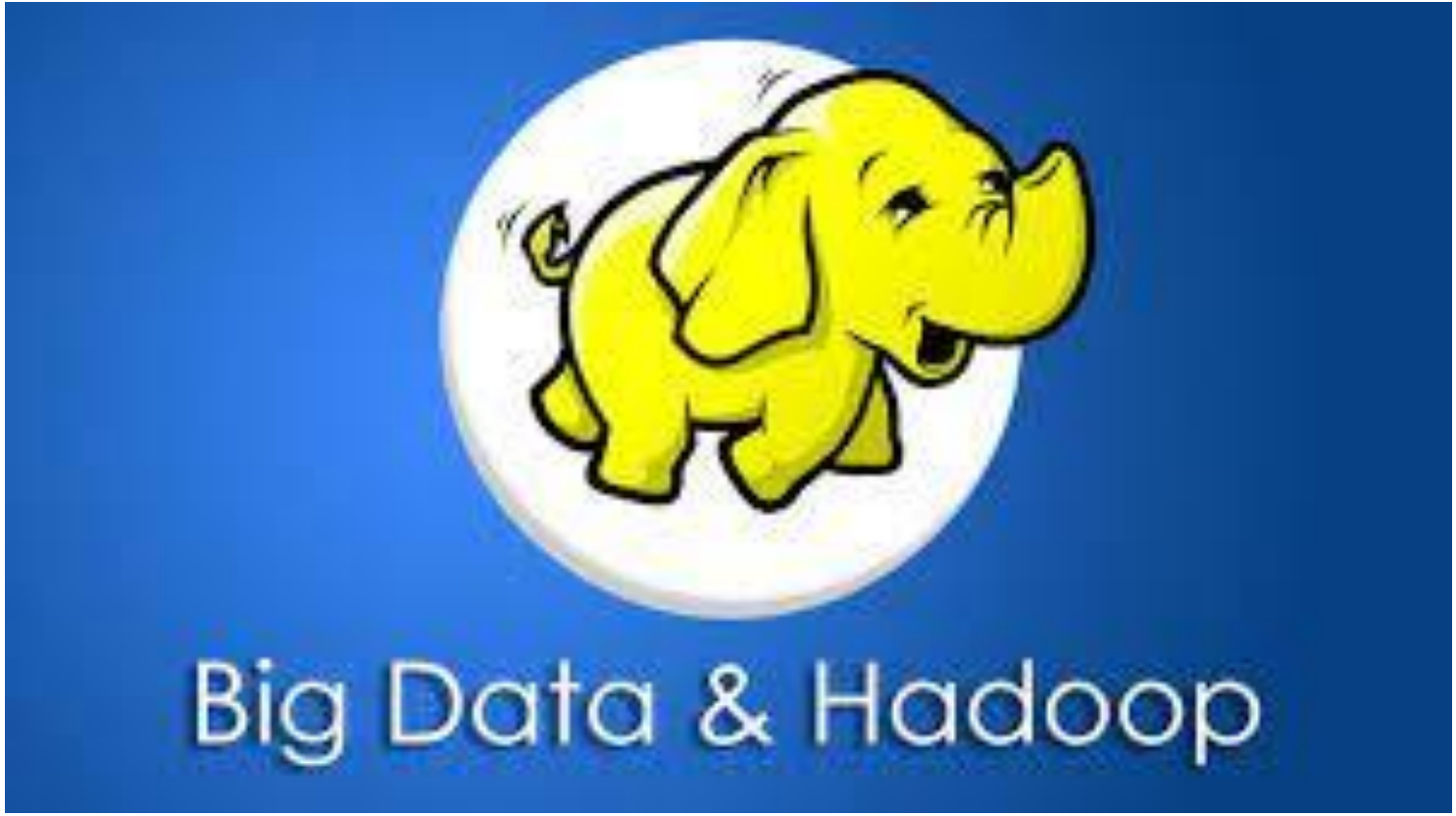

Big Data & Hadoop

What is BigData?

Big Data is a collection of data that is huge in volume, yet growing exponentially with time. It is a data with so large size and complexity that none of traditional data management tools can store it or process it efficiently.

The 5 Vs of Big Data

VELOCITY

- → Batch
- → Near time
- → Real time
- → Streams

VARIETY

- → Structured
- → Unstructured
- → Semistructured
- → All the above

VOLUME

- → Terabytes
- → Records
- → Transactions
- → Tables, files

VERACITY

- → Trustworthiness
- → Authenticity
- → Origin, reputation
- → Accountability

VALUE

- → Statistical
- → Events
- → Correlations
- → Hypothetical

5 V’s Of BigData

- **Volume**: This refers to the sheer quantity of data, which is typically enormous. It can range from terabytes to petabytes and even exabytes of data.

- **Velocity**: This refers to the speed at which new data is generated and the speed at which data moves around. With the growth of the Internet and smart devices, data is being generated continuously, in real time, from various sources.

- **Variety**: This refers to the different types of data that are now available. Big Data can include structured data (SQL databases), semi-structured data (XML, JSON), unstructured data (Word documents, JPEG images), and even complex structured data (from ERP systems or websites).

- **Veracity**: This refers to the quality of the data, which can vary greatly. Veracity allows us to deal with uncertainty or imprecision, which is often an issue with many forms of big data.

- **Value**: This is the ability to turn data into value. This is becoming the most important V of Big Data because it's important that businesses make a return on their investment in big data and data analytics.

Examples Of BigData

- **Social Media:** Data from posts, likes, shares on platforms like Facebook, Twitter, Instagram.
- **Healthcare**: Large volumes of data from electronic health records (EHRs), lab results, and patient histories.
- **Finance**: Data from financial transactions, stock exchanges, and trading systems.
- **Telecommunications**: Call detail records (CDRs) and network logs.
- **E-commerce**: Data from millions of transactions daily on platforms like Amazon, eBay.
- **IoT**: Real-time data from devices like smart homes, wearables, and industrial sensors.
- **Transport**: Data on travel times, routes, and traffic conditions from services like Uber and Google Maps.

Types of Data

- **Structured Data:** This is data that adheres to a model and is easily searchable. Examples include data stored in relational databases and spreadsheets.

An 'Employee' table in a database is an example of Structured Data

Employee_ID	Employee_Name	Gender	Department	Salary_In_lacs
2365	Rajesh Kulkarni	Male	Finance	650000
3398	Pratibha Joshi	Female	Admin	650000
7465	Shushil Roy	Male	Admin	500000
7500	Shubhojit Das	Male	Finance	500000
7699	Priya Sane	Female	Finance	550000

- **Unstructured Data:** This type of data does not have a predefined model or is not organized in a pre-defined manner. Examples include text files, images, videos, emails, web pages, and social media posts.

Types of Data

- **Semi-Structured Data:** This is a hybrid of structured and unstructured data. While it does not conform to the formal structure of data models, it contains tags or other markers to enforce hierarchy and order. Examples include JSON, XML, and email messages with both defined fields and free-form text.

```
<rec><name>Prashant Rao</name><sex>Male</sex><age>35</age></rec>
<rec><name>Seema R.</name><sex>Female</sex><age>41</age></rec>
<rec><name>Satish Mane</name><sex>Male</sex><age>29</age></rec>
<rec><name>Subrato Roy</name><sex>Male</sex><age>26</age></rec>
<rec><name>Jeremiah J.</name><sex>Male</sex><age>35</age></rec>
```

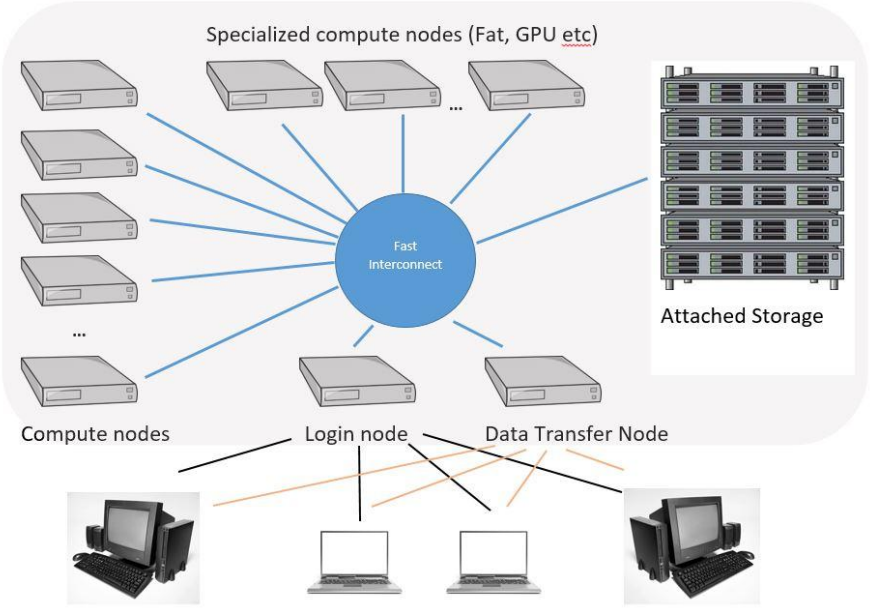
Types of Data

- **Semi-Structured Data:** This is a hybrid of structured and unstructured data. While it does not conform to the formal structure of data models, it contains tags or other markers to enforce hierarchy and order. Examples include JSON, XML, and email messages with both defined fields and free-form text.

```
<rec><name>Prashant Rao</name><sex>Male</sex><age>35</age></rec>
<rec><name>Seema R.</name><sex>Female</sex><age>41</age></rec>
<rec><name>Satish Mane</name><sex>Male</sex><age>29</age></rec>
<rec><name>Subrato Roy</name><sex>Male</sex><age>26</age></rec>
<rec><name>Jeremiah J.</name><sex>Male</sex><age>35</age></rec>
```

What is Cluster?

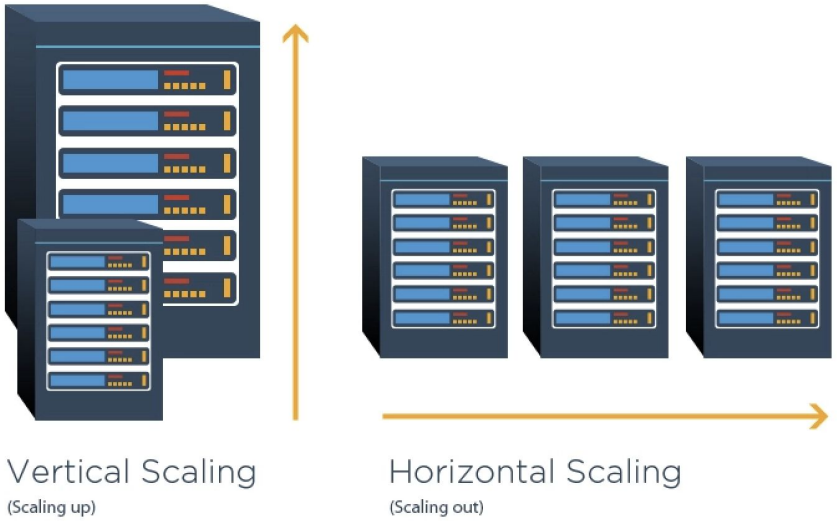
A cluster, in the context of computing, refers to a group of computers or servers that work together and can be viewed as a single system. These computers, known as nodes, interact with each other to accomplish a common goal. This setup is used to improve performance and availability over that provided by a single computer, while typically being much more cost-effective and scalable than a single computer of comparable speed or availability.



Vertical Scaling vs Horizontal Scaling

- **Vertical** Scaling, also known as scaling up, involves increasing the capacity of a single server, such as using a more powerful CPU, adding more RAM, or increasing disk space.

- **Horizontal** Scaling, also known as scaling out, involves adding more servers to a system and distributing the load across multiple servers.



Hadoop

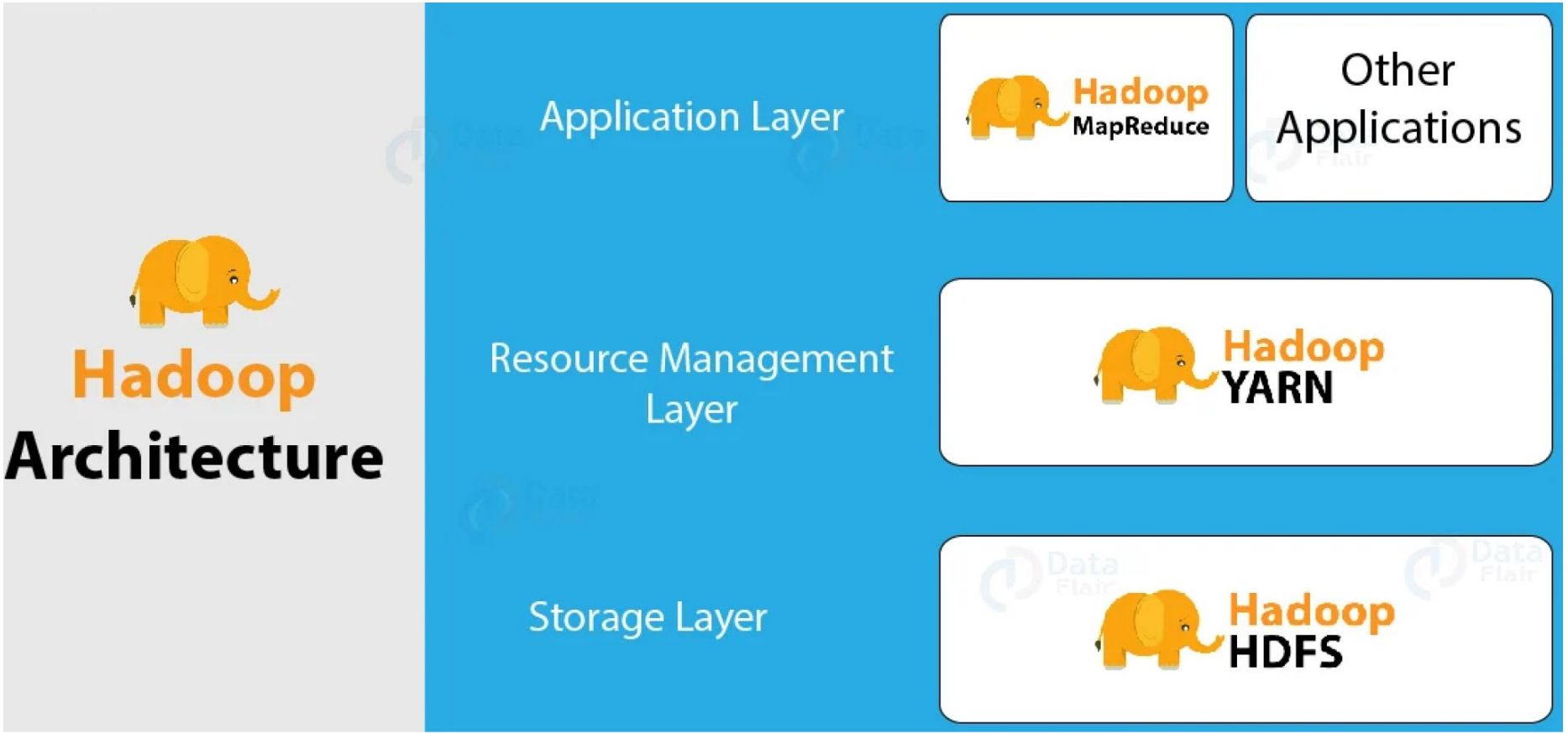
Hadoop is an open-source software framework for storing and processing big data in a distributed fashion on large clusters of commodity hardware. Essentially, it accomplishes two tasks: massive data storage and faster processing.

It was developed by the Apache Software Foundation and is based on two main components:

- **Hadoop Distributed File System (HDFS):** This is the storage component of Hadoop, designed to hold large amounts of data, potentially in the range of petabytes or even exabytes. The data is distributed across multiple nodes in the cluster, providing high availability and fault tolerance.

- **Map-Reduce:** This is the processing component of Hadoop, which provides a software framework for writing applications that process large amounts of data in parallel. MapReduce operations are divided into two stages: the **Map stage**, which sorts and filters the data, and the **Reduce stage**, which summarizes the data.

- **Yet Another Resource Negotiator (YARN):** This is the resource management layer in Hadoop. Introduced in Hadoop 2.0, YARN decouples the programming model from the resource management infrastructure, and it oversees and manages the compute resources in the clusters.



Properties of Hadoop

- **Scalability**: Can store and distribute large data sets across many servers.
- **Cost-effectiveness:** Designed to run on inexpensive, commodity hardware.
- **Flexibility**: Can handle any type of data, structured or unstructured.
- **Fault Tolerance**: Data is automatically replicated to other nodes in the cluster.
- **Data Locality**: Processes data on or near the node where it's stored, reducing network I/O.
- **Simplicity**: Provides a simple programming model (MapReduce) for processing data.
- **Open-source**: Freely available to use and modify with a large community of contributors.

HDFS

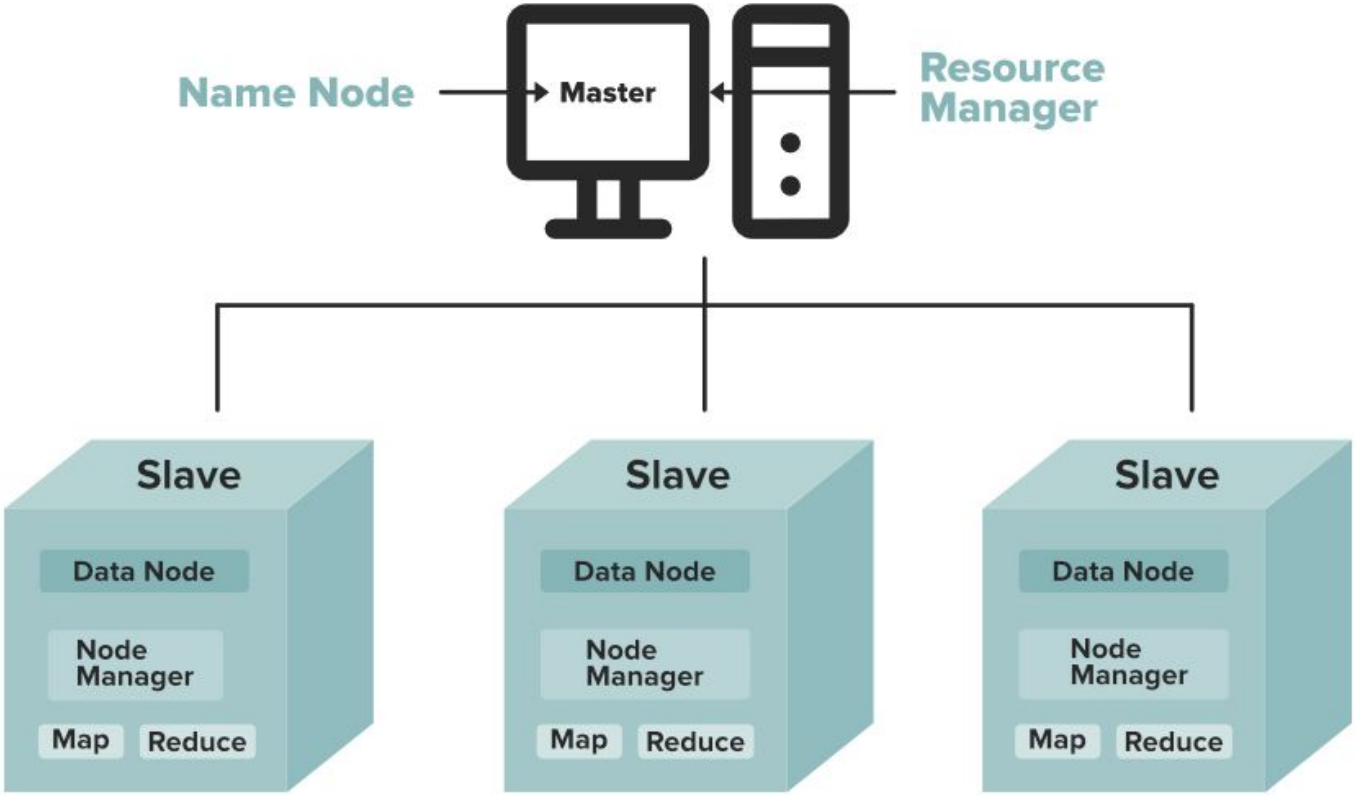
HDFS, the Hadoop Distributed File System, follows a master-slave architecture. The two main components of HDFS are the NameNode (the master) and the DataNode (the slave).

NameNode: The NameNode is the centerpiece of an HDFS file system. It keeps the directory tree of all files in the file system, and tracks where across the cluster the file data is kept. It does not store the data of these files itself.

- The NameNode manages the file system namespace. It maintains the file system tree and the metadata for all the files and directories in the tree.
- The NameNode also knows the DataNodes on which all the blocks for a given file are located.

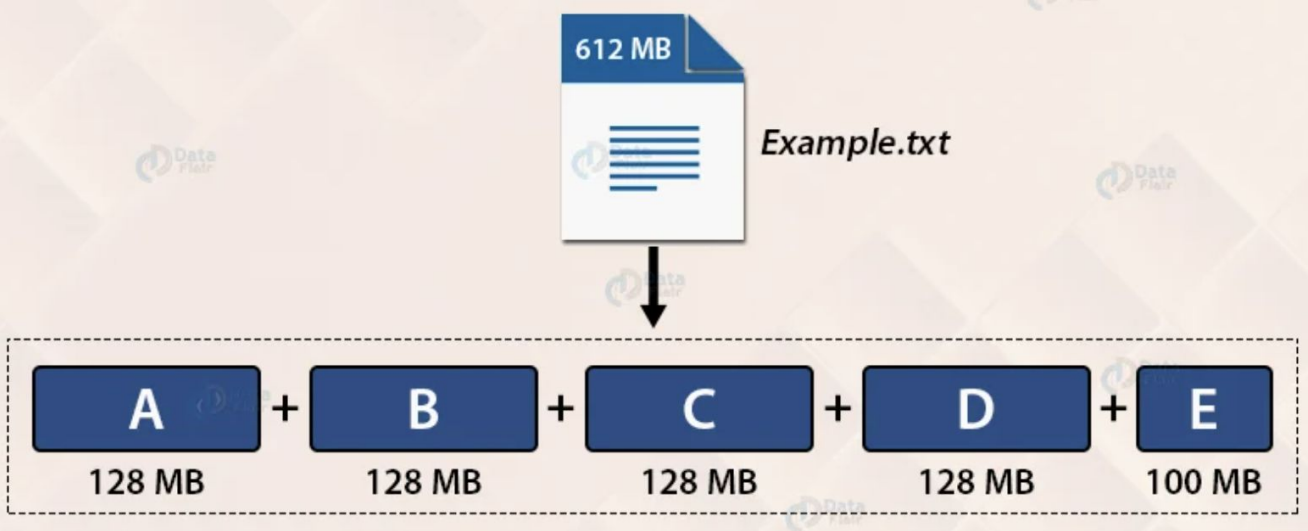
DataNode: The DataNodes are responsible for serving read and write requests from the file system's clients. They also perform **block** creation, deletion, and replication upon instruction from the NameNode.

- The DataNodes manage the storage attached to the nodes that they run on.
- DataNodes regularly report back to the NameNode with lists of blocks that they are storing.



Key terminologies and Components of HDFS

- **Block:** Block is nothing but the smallest unit of storage on a computer system. It is the smallest contiguous storage allocated to a file. In Hadoop, we have a default block size of **128MB** or **256MB**.



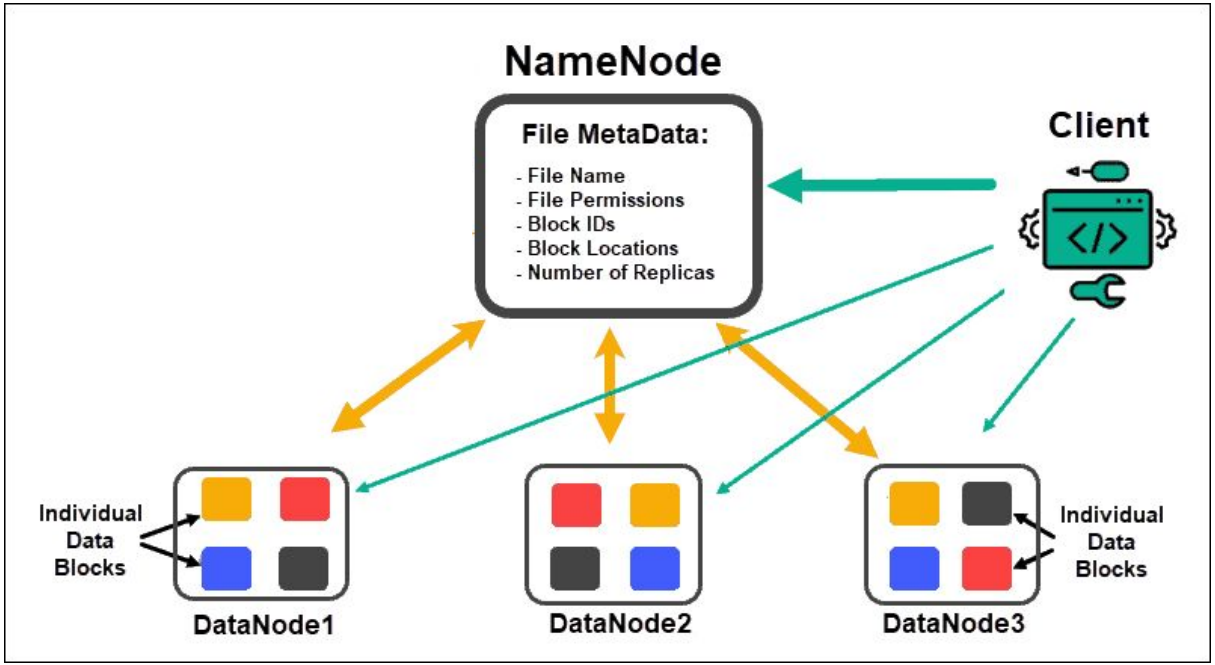
- If you have a file of 50 MB and the HDFS block size is set to 128 MB, the file will only use 50 MB of one block. The remaining 78 MB in that block will remain unused, as HDFS blocks are allocated on a per-file basis.

- It's important to note that this is one of the reasons why HDFS is not well-suited to handling a large number of small files. Since each file is allocated its own blocks, if you have a lot of files that are much smaller than the block size, then a lot of space can be wasted.

- This is also why block size in HDFS is considerably larger than it is in other file systems (default of 128 MB, as opposed to a few KBs or MBs in other systems). Larger block sizes mean fewer blocks for the same amount of data, leading to less metadata to manage, less communication between the NameNode and DataNodes, and better performance for large, streaming reads of data.

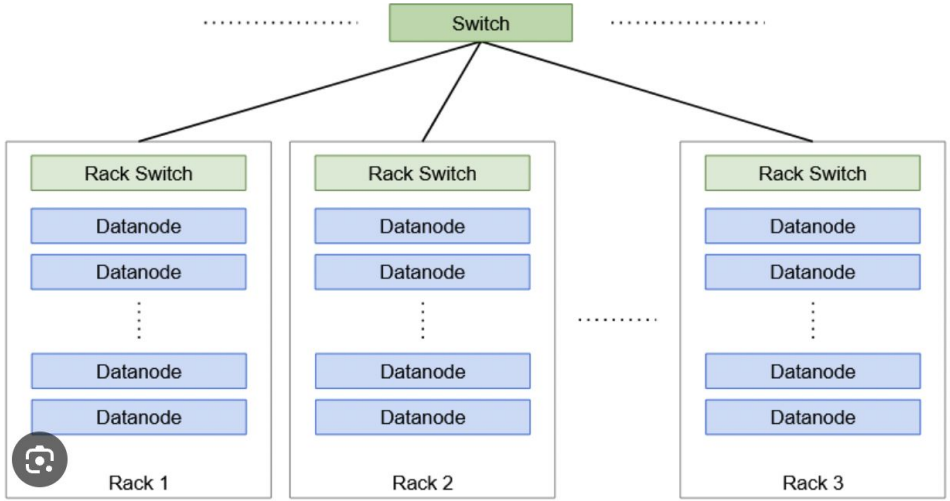
Key terminologies and Components of HDFS

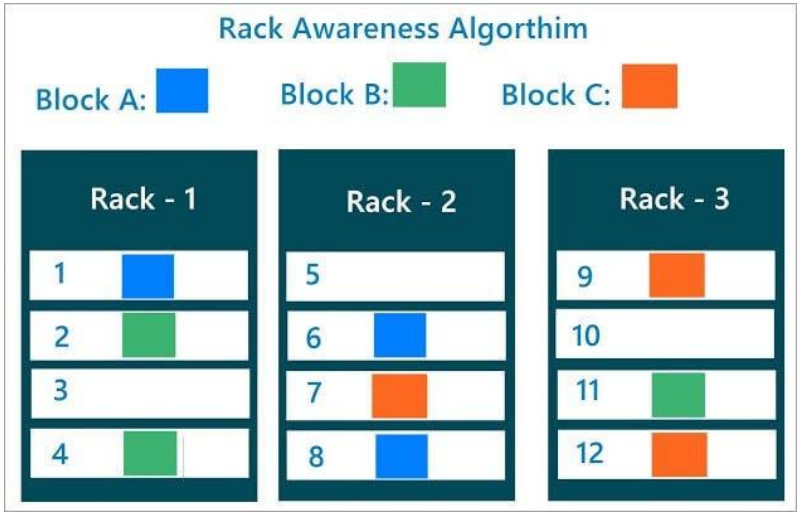
- **Replication Management:** Replication Management: To provide fault tolerance HDFS uses a replication technique. In that, it makes copies of the blocks and stores in on different DataNodes. Replication factor decides how many copies of the blocks get stored. It is 3 by default but we can configure to any value.



Key terminologies and Components of HDFS

- **Rack Awareness:** A rack contains many DataNode machines and there are several such racks in the production. HDFS follows a rack awareness algorithm to place the replicas of the blocks in a distributed fashion. This rack awareness algorithm provides for low latency and fault tolerance. 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.





Key terminologies and Components of HDFS

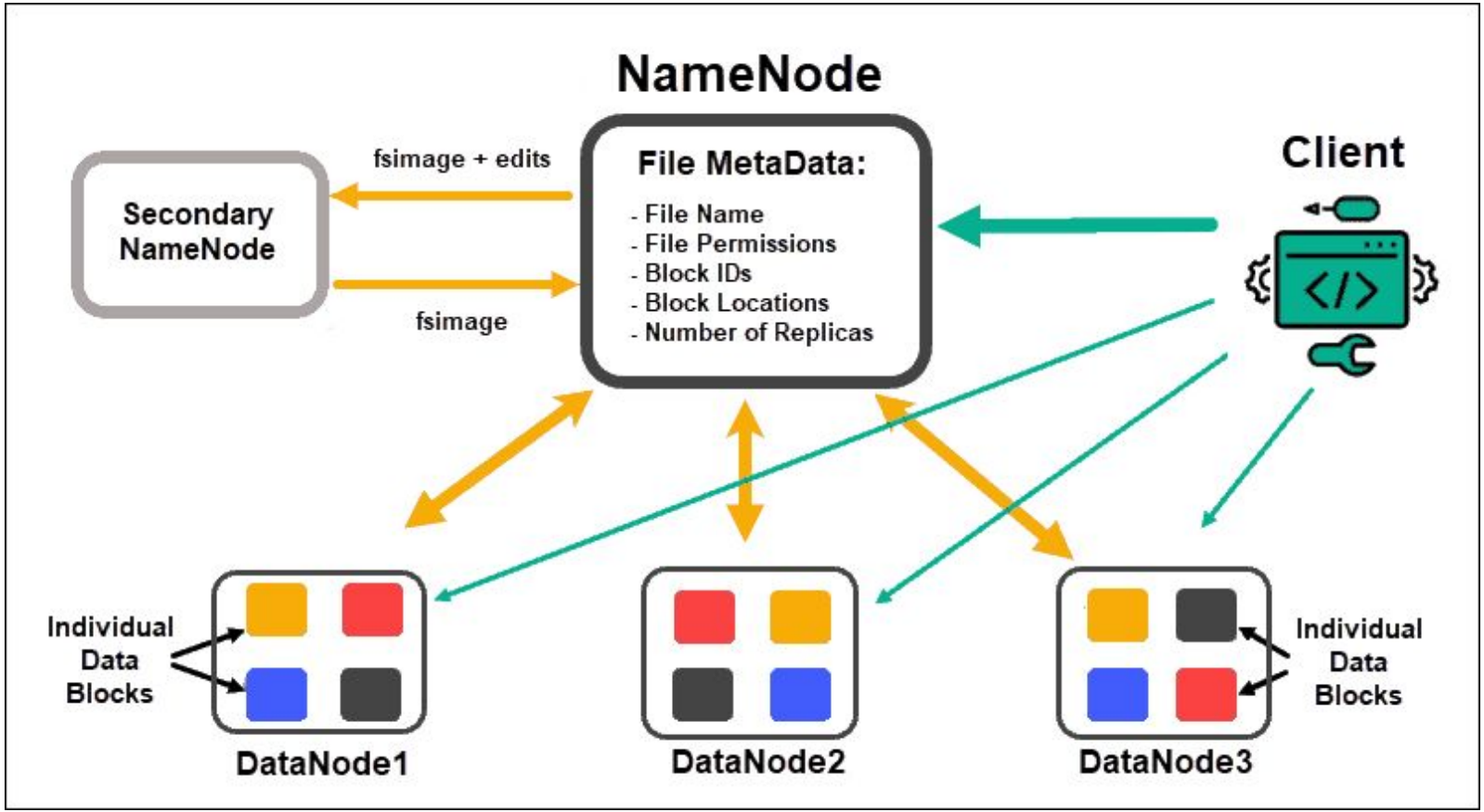
Secondary Namenode: The Secondary NameNode in Hadoop HDFS is a specially dedicated node in the Hadoop cluster that serves as a helper to the primary NameNode, but not as a standby NameNode. Its main roles are to take checkpoints of the filesystem metadata and help in keeping the filesystem metadata size within a reasonable limit.

Here is what it does:

- **Checkpointing**: The Secondary NameNode periodically creates checkpoints of the namespace by merging the fsimage file and the edits log file from the NameNode. The new fsimage file is then transferred back to the NameNode. These checkpoints help reduce startup time of the NameNode.

- **Size management**: The Secondary NameNode helps in reducing the size of the edits log file on the NameNode. By creating regular checkpoints, the edits log file can be purged occasionally, ensuring it does not grow too large.

A common misconception is that the Secondary NameNode is a failover option for the primary NameNode. However, this is not the case; the Secondary NameNode cannot substitute for the primary NameNode in the event of a failure. For that, Hadoop 2 introduces the concept of Standby NameNode.


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

Key terminologies and Components of HDFS

Standby Namenode: In Hadoop, the Standby NameNode is part of the **High Availability (HA) feature** of HDFS that was introduced with Hadoop 2.x. This feature addresses one of the main drawbacks of the earlier versions of Hadoop: the single point of failure in the system, which was the NameNode.

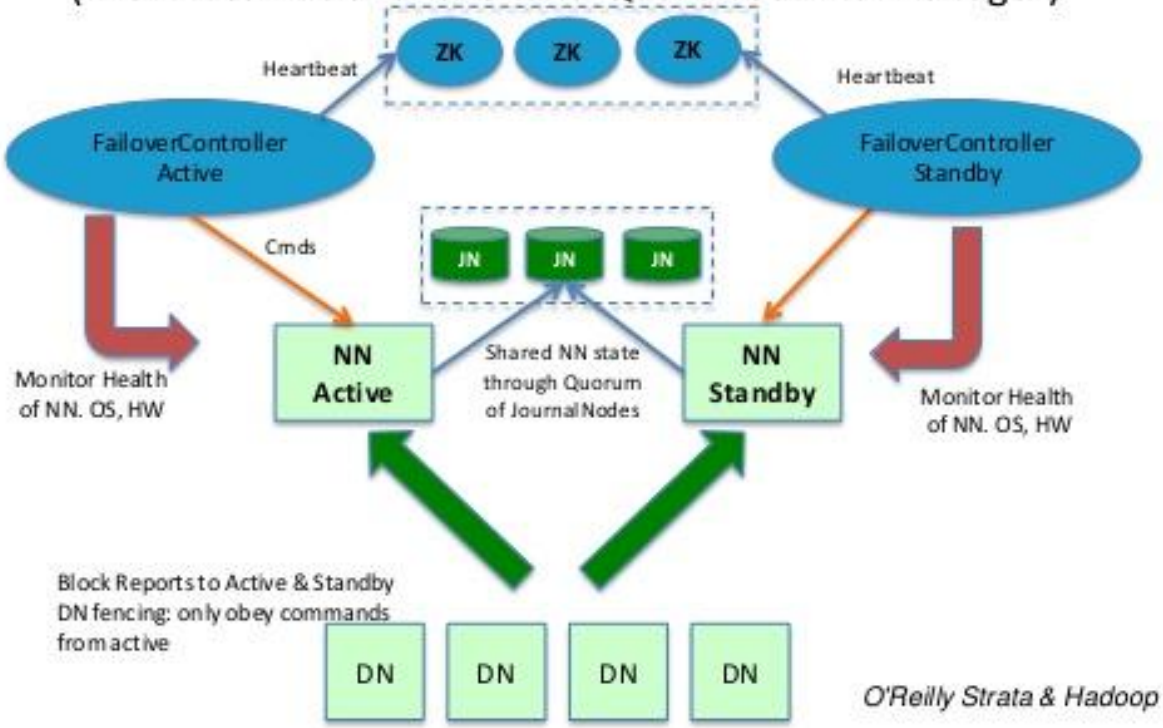
- The Standby NameNode is essentially a hot backup for the Active NameNode. The Standby NameNode and Active NameNode are in constant synchronization with each other. When the Active NameNode updates its state, it records the changes to the edit log, and the Standby NameNode applies these changes to its own state, keeping both NameNodes in sync.

- The Standby NameNode maintains a copy of the namespace image in memory, just like the Active NameNode. This means it can quickly take over the duties of the Active NameNode in case of a failure, providing minimal downtime and disruption.

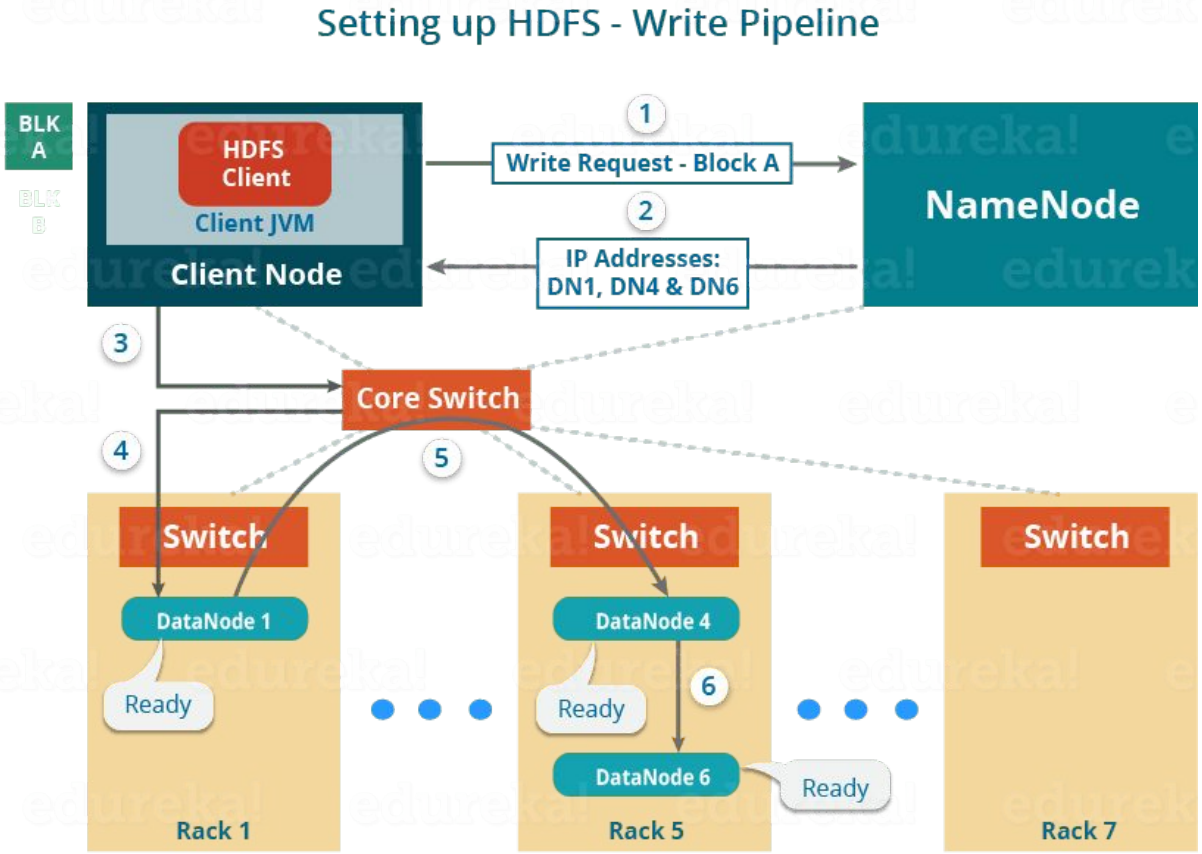
- Unlike the Secondary NameNode, the Standby NameNode is capable of taking over the role of the Active NameNode immediately without any data loss, thus ensuring the High Availability of the HDFS system.

HDFS HA Architecture

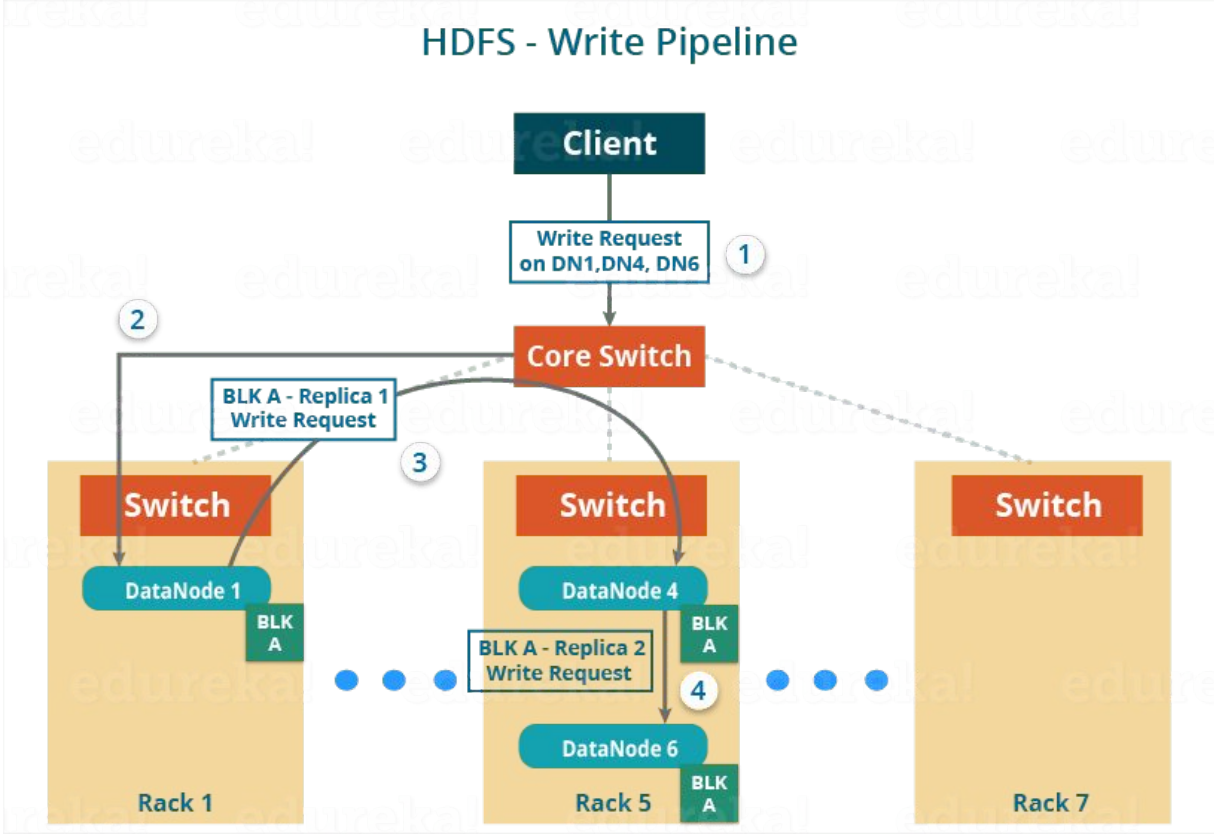
(with Automatic Failover and QuorumJournalManager)



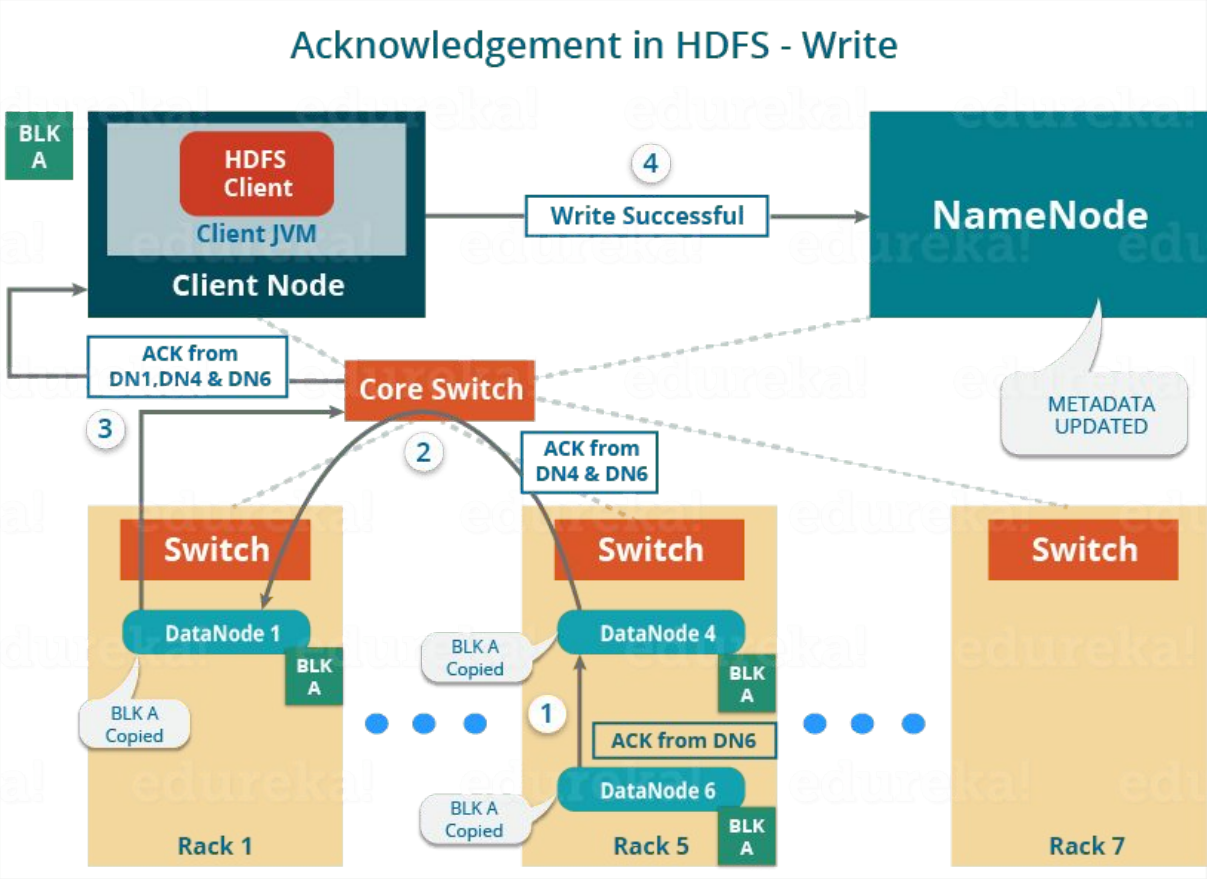
Write operation in HDFS - Step 1



Write operation in HDFS - Step 2



Write operation in HDFS - Step 3



Read Operation in HDFS