HDFS

HDFS

- Hadoop comes with a distributed file system called HDFS.
- In HDFS data is distributed over several machines and replicated to ensure their durability to failure and high availability to parallel application.
- It is cost effective as it uses commodity hardware.
- It involves the concept of
  - blocks,
  - data nodes and
  - node name.

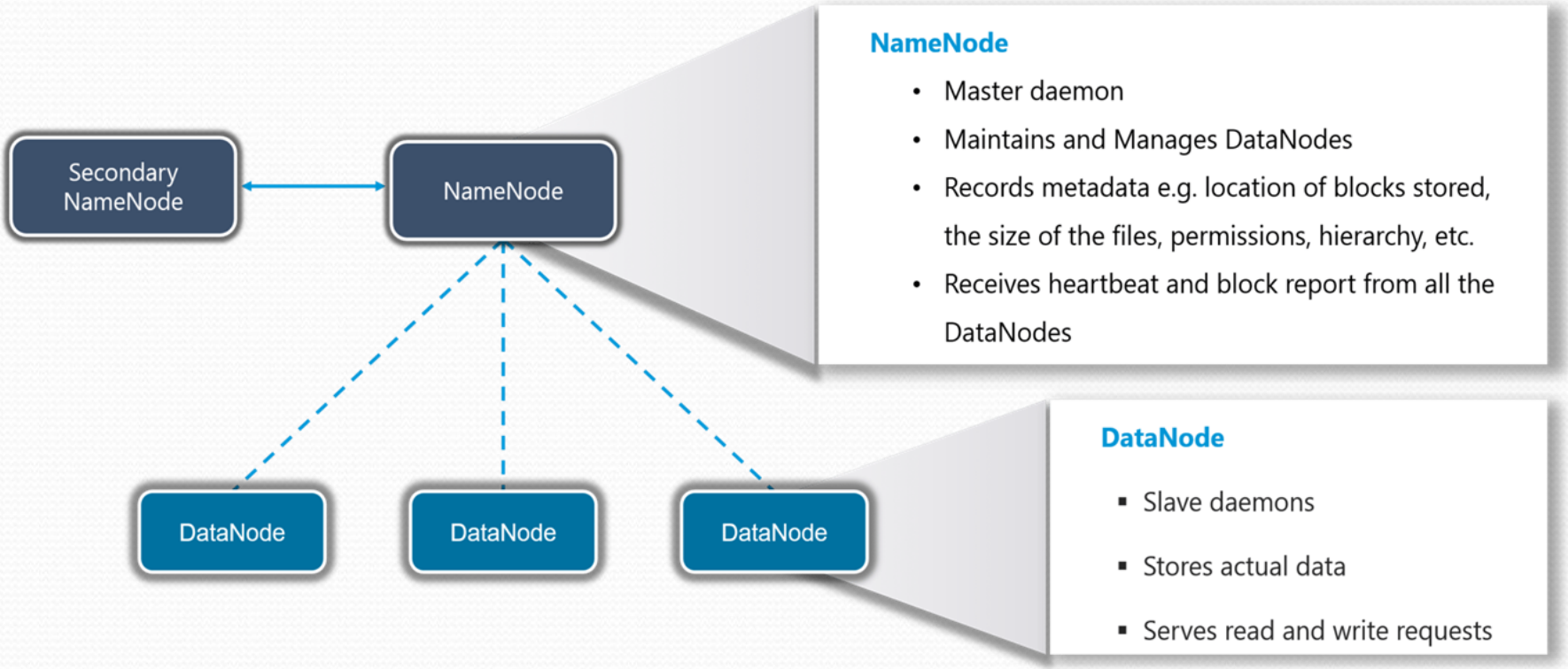
Where to use HDFS

- **Very Large Files:**
  - Files should be of hundreds of megabytes, gigabytes or more.
- **Streaming Data Access:**
  - The time to read whole data set is more important than latency in reading the first.
  - HDFS is built on write-once and read-many-times pattern.
- **Commodity Hardware:**
  - It works on low cost hardware.

Where not to use HDFS

- **Low Latency data access:**
  - Applications that require very less time to access the first data should not use HDFS as it is giving importance to whole data rather than time to fetch the first record.
- **Lots Of Small Files:**
  - The name node contains the metadata of files in memory and if the files are small in size it takes a lot of memory for name node's memory which is not feasible.
- **Multiple Writes:**
  - It should not be used when we have to write multiple times.

HDFS Concepts



HDFS Concepts

1. **Blocks:**

- A Block is the minimum amount of data that it can read or write.
- HDFS blocks are 128 MB by default and this is configurable.
- Files in HDFS are broken into block-sized chunks, which are stored as independent units.
- Unlike a file system, if the file is in HDFS is smaller than block size, then it does not occupy full block size, i.e. 5 MB of file stored in HDFS of block size 128 MB takes 5MB of space only.
- The HDFS block size is large just to minimize the cost of seek.

2. Name Node:

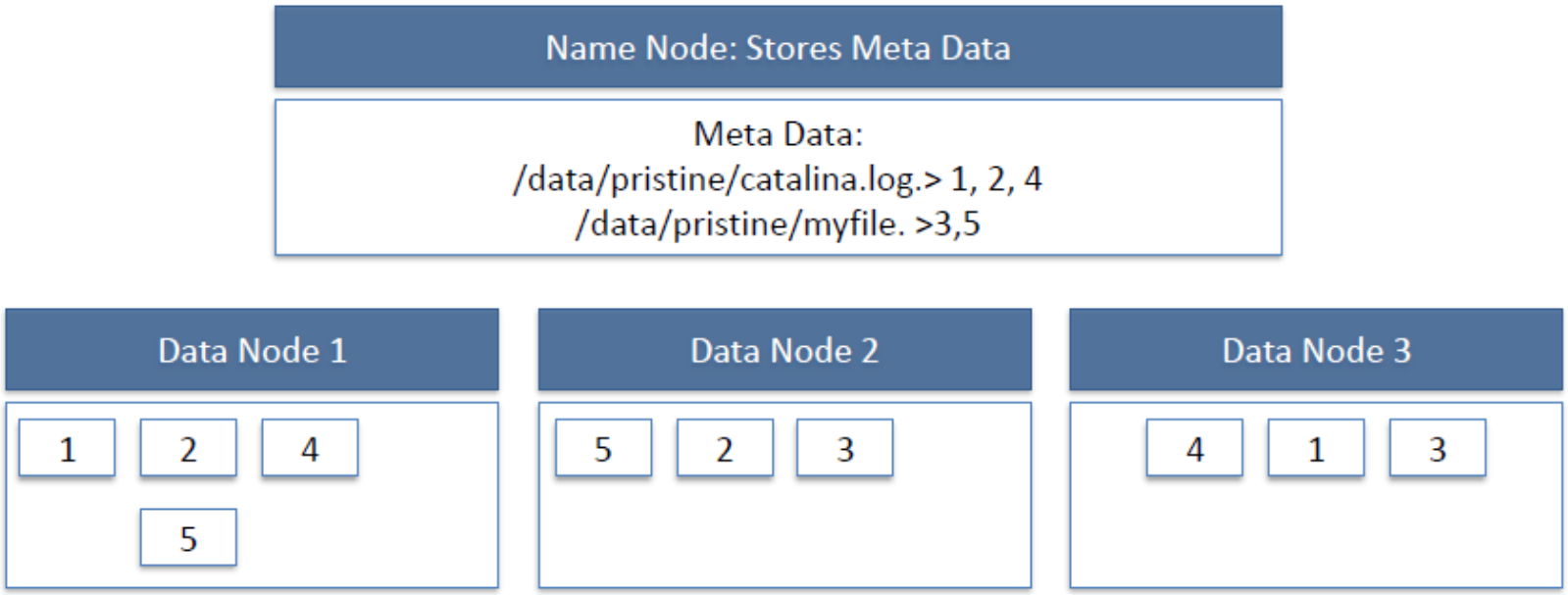
- HDFS works in a master-worker pattern where the name node acts as master.
- Name Node is controller and manager of HDFS as it knows the status and the metadata of all the files in HDFS; the metadata information being file permission, names, and location of each block.
- The metadata is small, so it is stored in the memory of the name node, allowing faster access to data.
- Moreover the HDFS cluster is accessed by multiple clients concurrently,so all this information is handled by a single machine.
- The file system operations like opening, closing, renaming etc. are executed by it.

- **Secondary Name Node:**
- It is a separate physical machine which acts as a helper of name node.
- It performs periodic check points.
- It communicates with the name node and take snapshot of meta data which helps minimize downtime and loss of data.

3. Data Node:

- They store and retrieve blocks when they are told to; by client or name node.
- They report back to name node periodically, with list of blocks that they are storing.
- The data node being a commodity hardware also does the work of block creation, deletion and replication as stated by the name node.

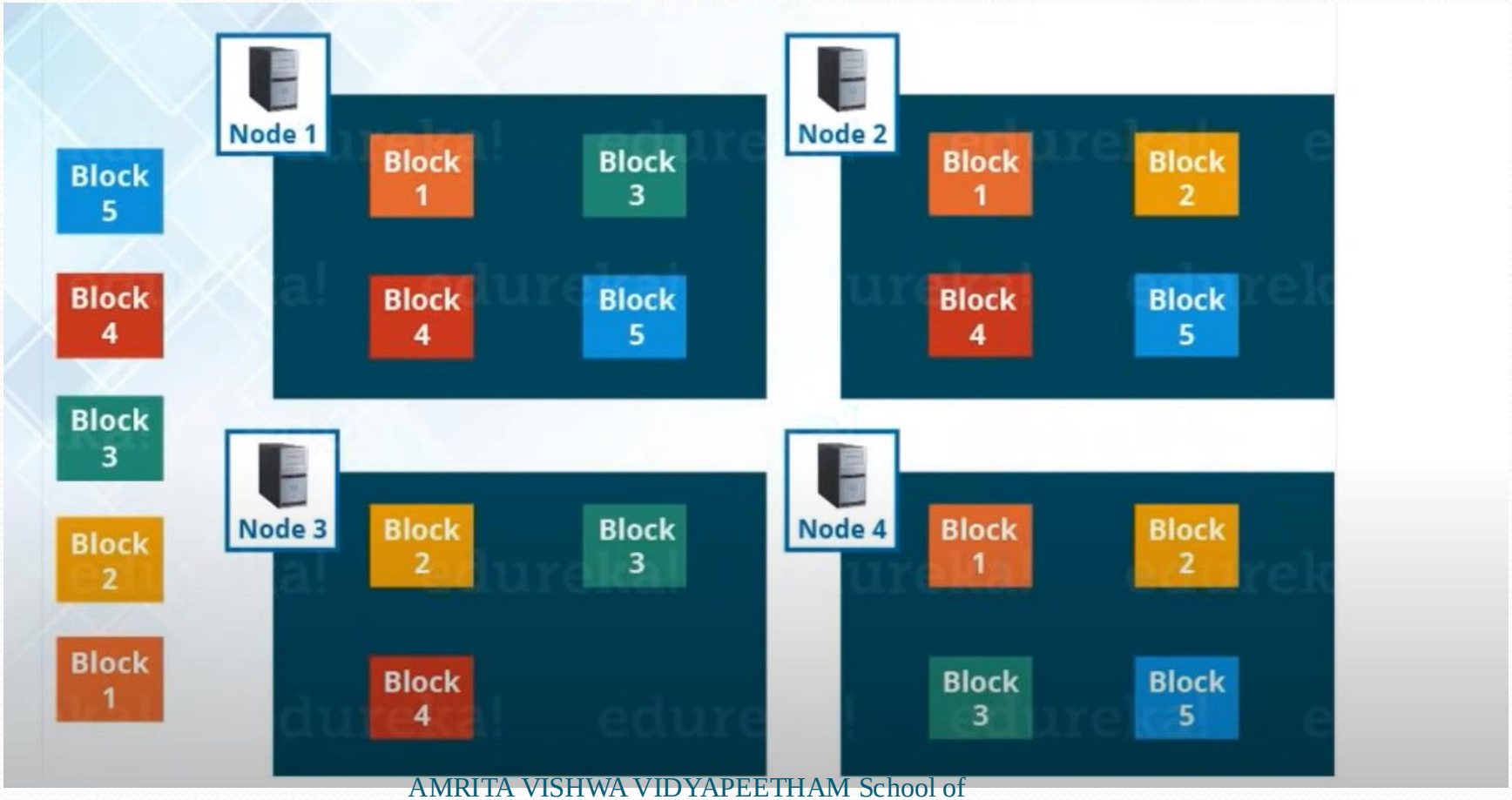
HDFS DataNode and NameNode Image:



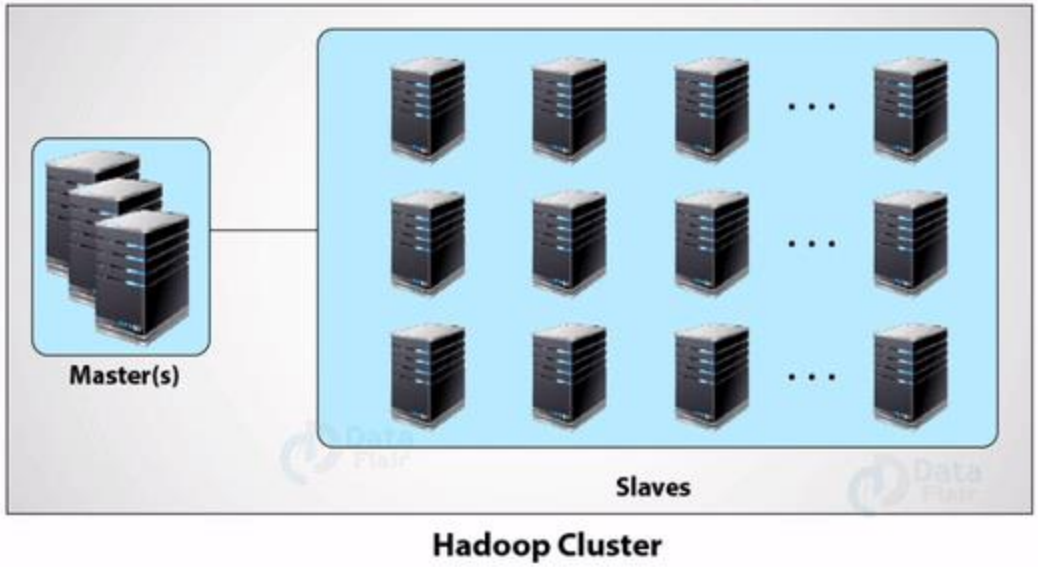
Block Replication

- Block replication is a fundamental concept in Hadoop Distributed File System (HDFS), which ensures high availability and fault tolerance of data stored in the Hadoop cluster.
- In HDFS, files are divided into fixed-size blocks, typically 128MB or 256MB, and each block is replicated across multiple data nodes in the cluster.
- By default, each block is replicated three times, which means that there are three copies of each block stored in different data nodes in the cluster.
- The block replication process is automatic and transparent to the user.
- Whenever a new file is created in HDFS, the file is divided into blocks, and each block is replicated across multiple data nodes in the cluster.
- If a data node fails, the block replicas stored on that node are automatically copied to other nodes in the cluster, ensuring that the data is still available even if one or more nodes fail.
- Block replication provides fault tolerance and high availability by ensuring that multiple copies of each block are stored in different data nodes in the cluster.
- If a node fails, HDFS can retrieve the block from one of the other replicas, ensuring that the data is still accessible.

Block Replication

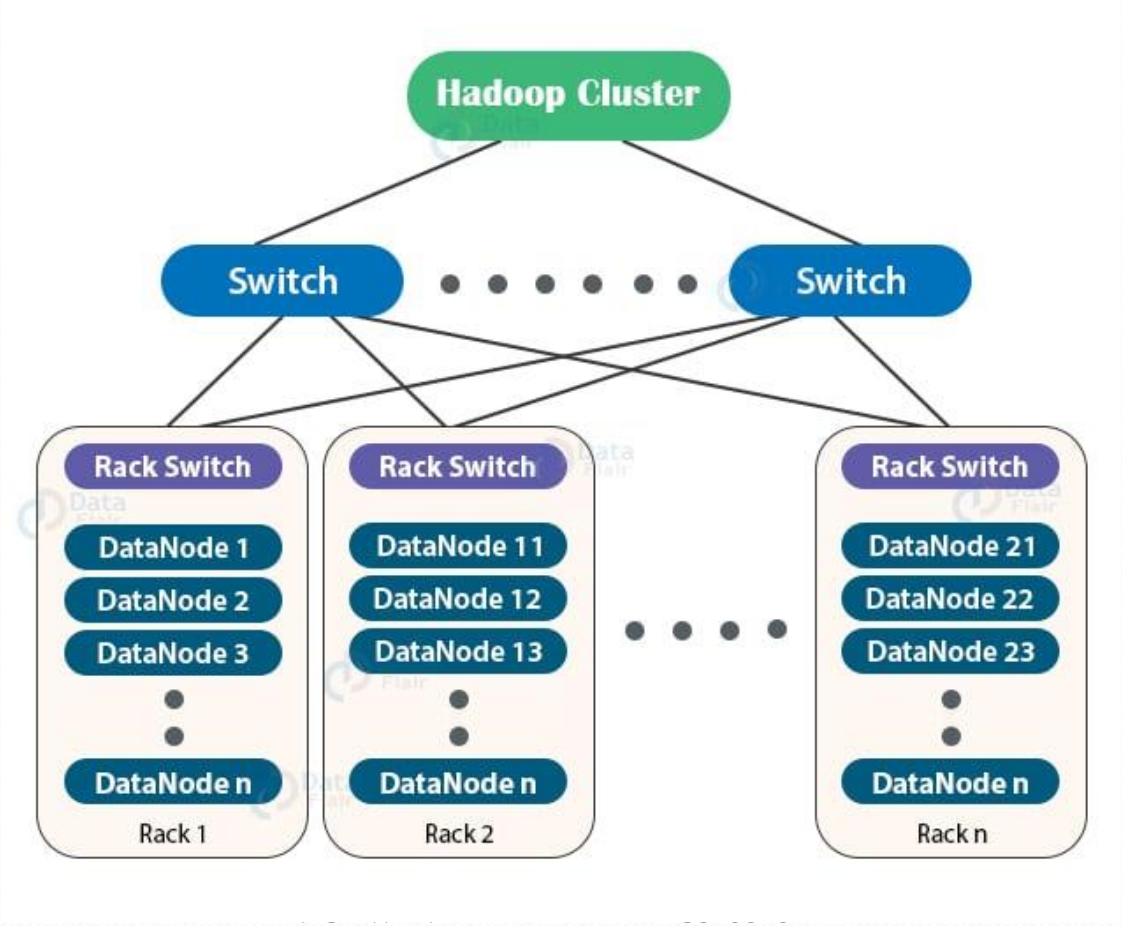


Data Storage in HDFS



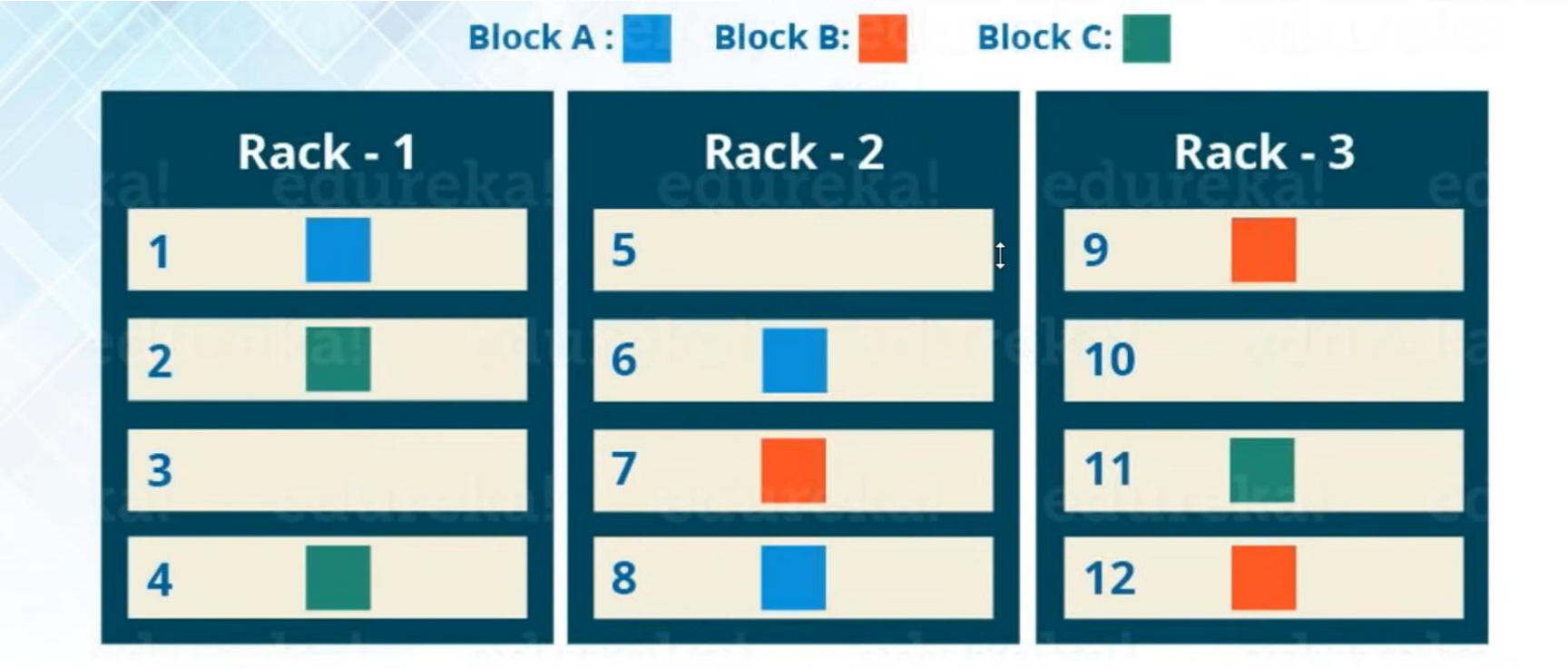
Rack Awareness

- The Rack is the collection of around 40-50 DataNodes connected using the same network switch.
- If the network goes down, the whole rack will be unavailable. A large Hadoop cluster is deployed in multiple racks.
- 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.
- To reduce the network traffic during file **read/write**, NameNode chooses the closest DataNode for serving the client read/write request.
- NameNode maintains **rack ids** of each DataNode to achieve this rack information.
- This concept of choosing the closest DataNode based on the rack information is known as **Rack Awareness**.


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

Rack Awareness

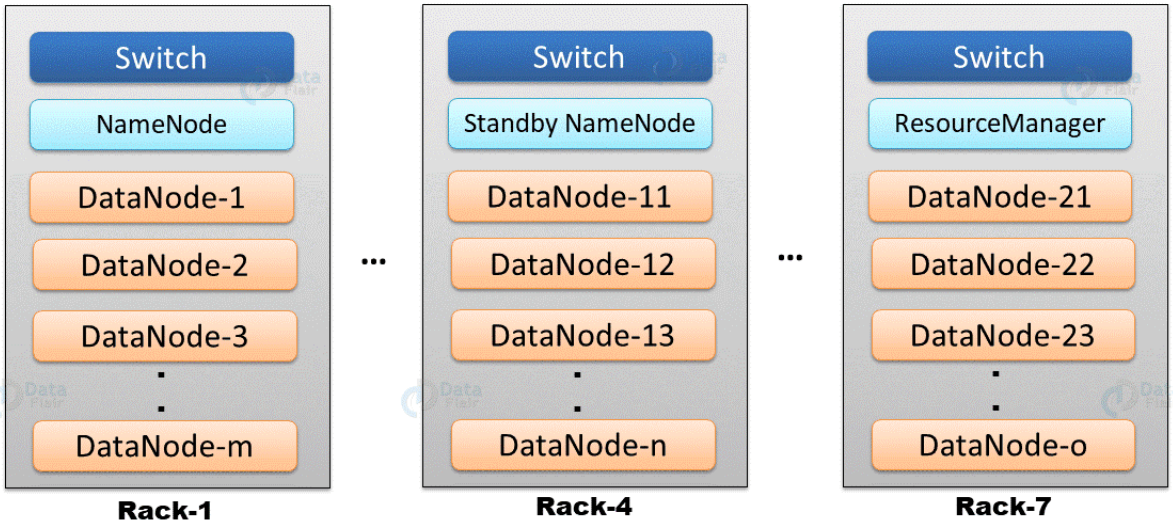
Rack Awareness



Rack Awareness

- With rack awareness enabled, HDFS tries to place the first replica of a block on a node in a different rack than the other replicas.
- This ensures that the data is distributed across different racks and reduces the risk of data loss in case a rack or a switch fails.
- Rack awareness also affects the way computation tasks are scheduled in Hadoop MapReduce.
- When a task is assigned to a node, Hadoop MapReduce scheduler tries to choose a node that is on the same rack as the data to minimize network traffic.
- This helps to reduce the data transfer time and improves the overall performance of the system.

HDFS Rack Awareness



- we are having a file “File.txt” divided into three blocks A, B, and C.
- To provide fault tolerance, HDFS creates replicas of blocks.
- NameNode places the first copy of each block on the closest DataNode, the second replica of each block on different DataNode on the same rack, and the third replica on different DataNode on a different rack.

Rack awareness policies.

- Not more than one replica be placed on one node.
- Not more than two replicas are placed on the same rack.
- Also, the number of racks used for block replication should always be smaller than the number of replicas.

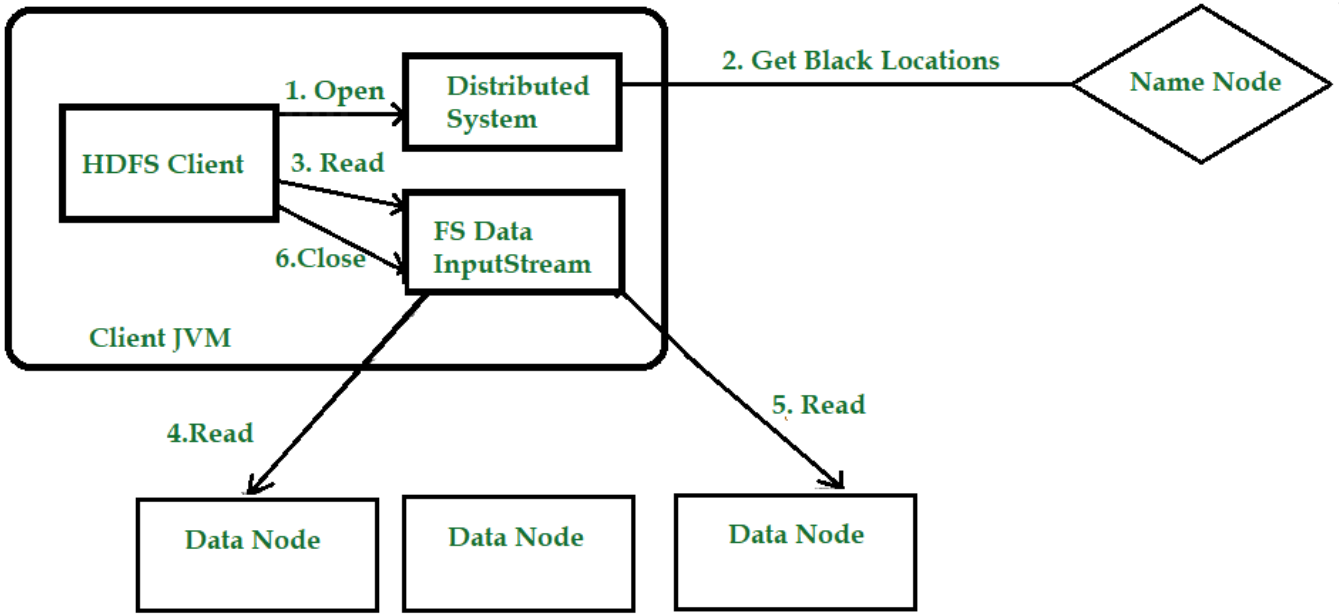
Benefits of Implementing Rack Awareness in our Hadoop Cluster:

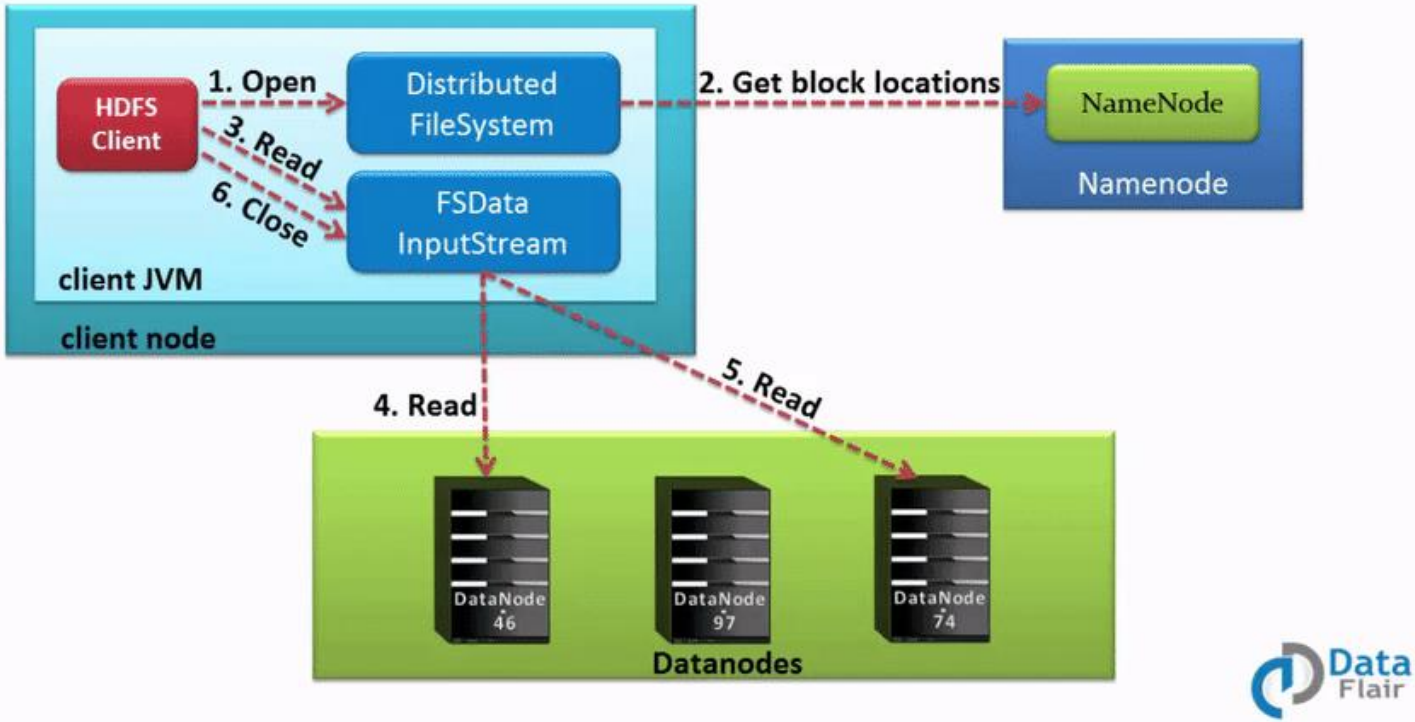
- With the rack awareness policy's we store the data in different Racks so no way to lose our data.
- Rack awareness helps to maximize the network bandwidth because the data blocks transfer within the Racks.
- It also improves the cluster performance and provides high data availability.

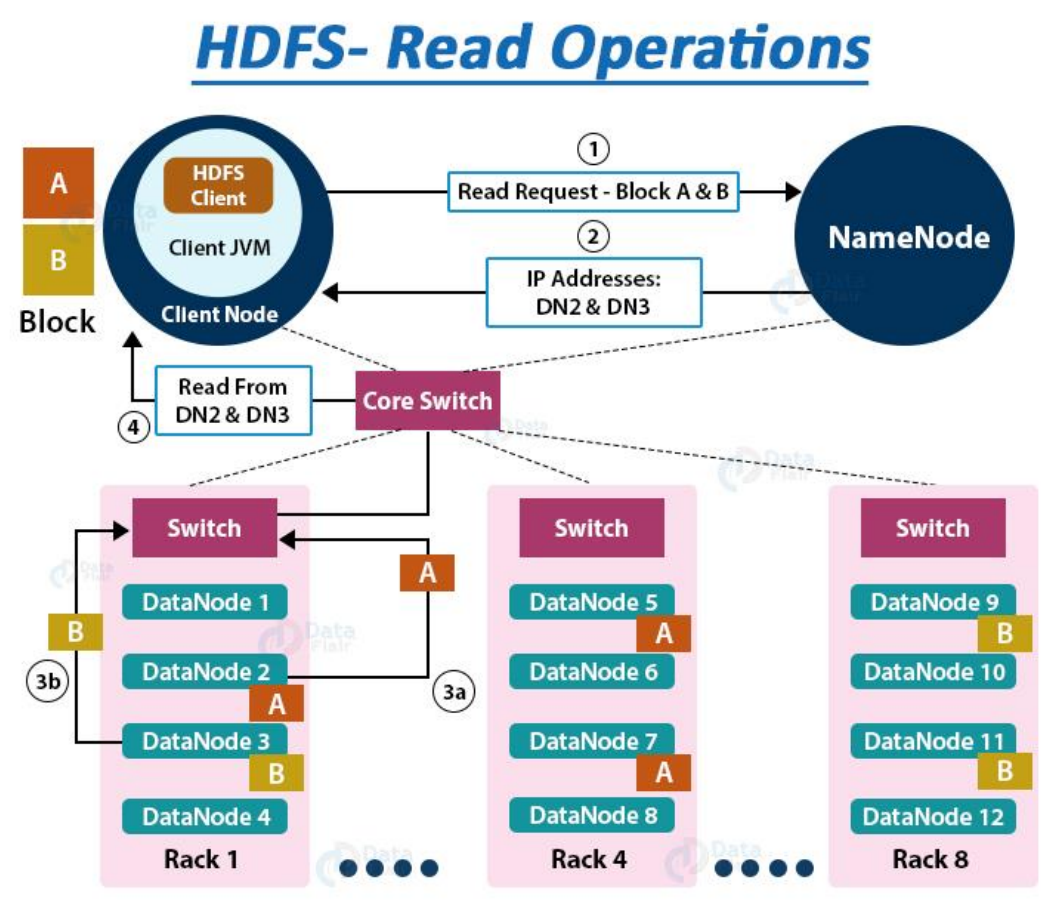
Features of HDFS

- **Highly Scalable –**
  - HDFS is highly scalable as it can scale hundreds of nodes in a single cluster.
- **Replication -**
  - Due to some unfavorable conditions, the node containing the data may be loss.
  - So, to overcome such problems, HDFS always maintains the copy of data on a different machine.
- **Fault tolerance –**
  - In HDFS, the fault tolerance signifies the robustness of the system in the event of failure.
  - The HDFS is highly fault-tolerant that if any machine fails, the other machine containing the copy of that data automatically become active.
- **Distributed data storage -**
  - This is one of the most important features of HDFS that makes Hadoop very powerful. Here, data is divided into multiple blocks and stored into nodes.
- **Portable -**
  - HDFS is designed in such a way that it can easily portable from platform to another.

HDFS Read

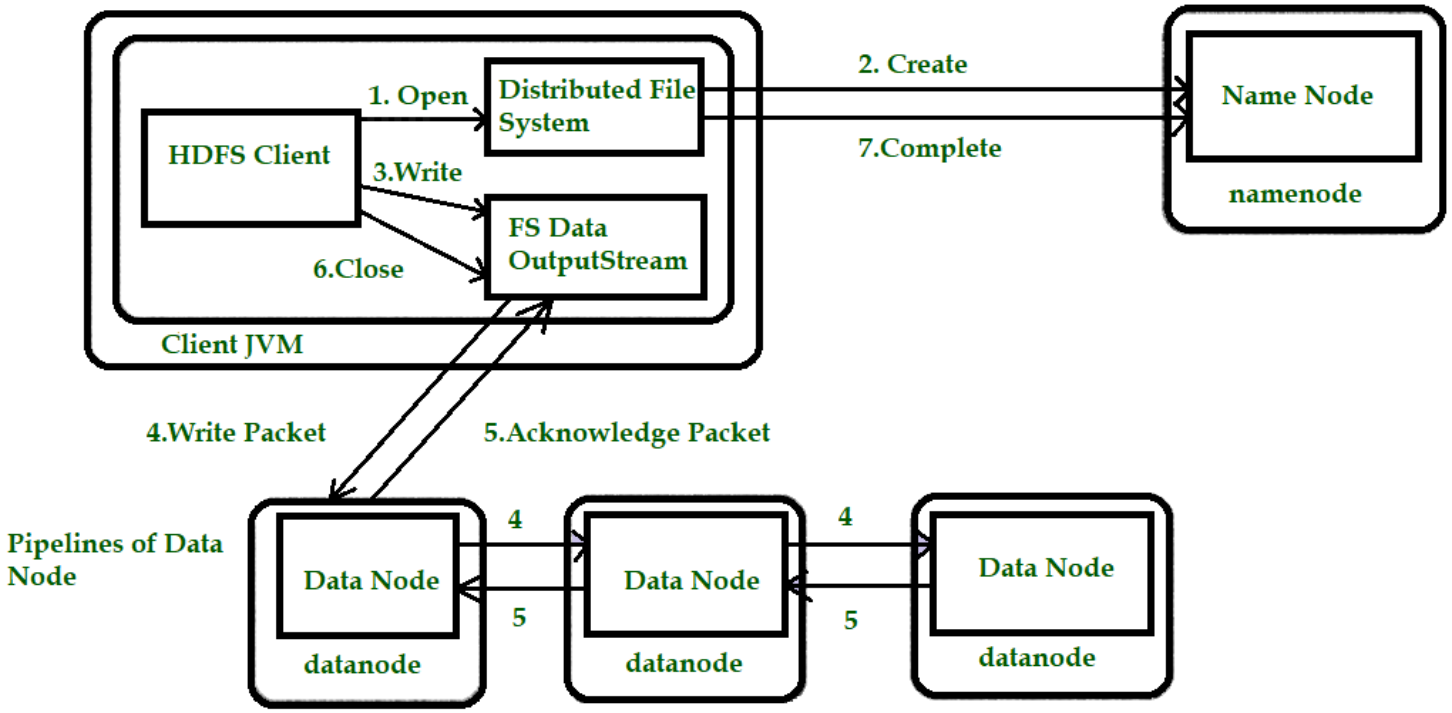

1. Open
Distributed System
2. Get Black Locations
Name Node
HDFS Client
3. Read
FS Data InputStream
6.Close
Client JVM
4.Read
5. Read
Data Node
Data Node
Data Node

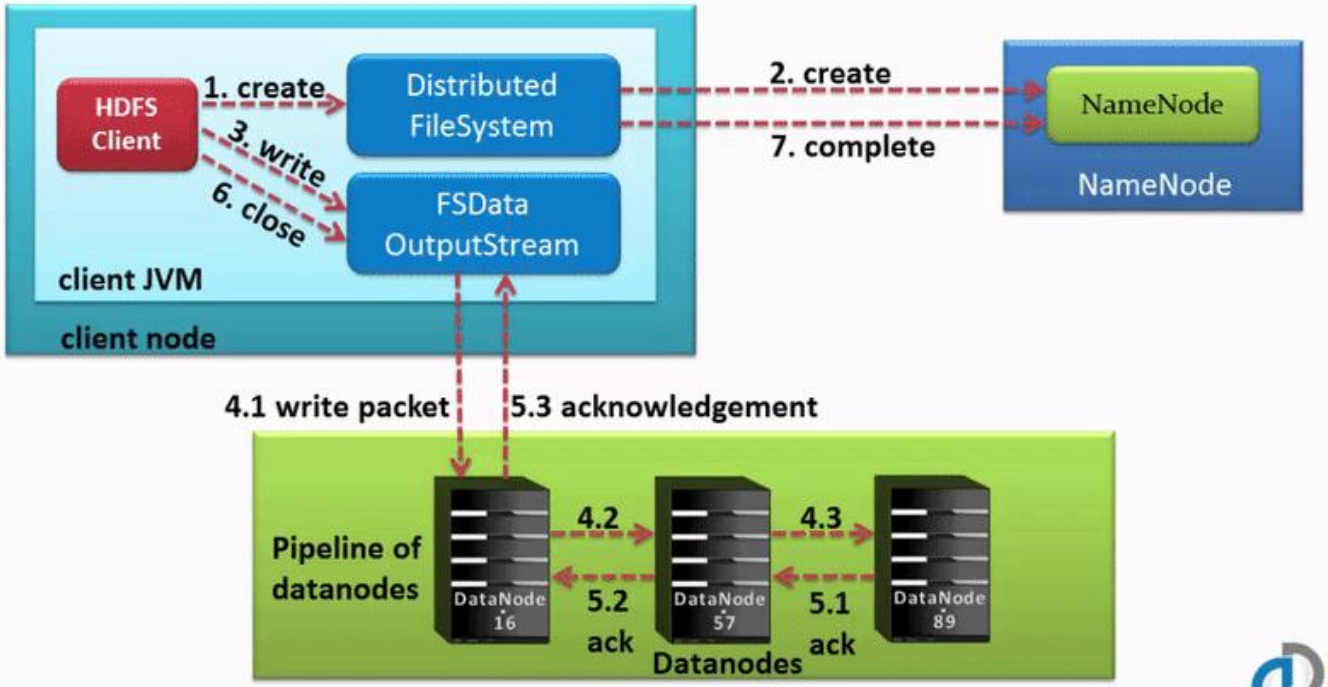

HDFS Client
1. Open
Distributed FileSystem
2. Get block locations
NameNode
Namenode
3. Read
6. Close
FSData InputStream
client JVM
client node
4. Read
5. Read
DataNode 46
DataNode 97
DataNode 74
Datanodes


HDFS- Read Operations
A
B
Block
HDFS Client
Client JVM
Client Node
1
Read Request - Block A & B
2
IP Addresses: DN2 & DN3
NameNode
Read From DN2 & DN3
4
Core Switch
Switch
DataNode 1
DataNode 2
DataNode 3
DataNode 4
Rack 1
3b
3a
Switch
DataNode 5
DataNode 6
DataNode 7
DataNode 8
Rack 4
Switch
DataNode 9
DataNode 10
DataNode 11
DataNode 12
Rack 8

- **Step 1:** The client opens the file it wishes to read by calling open() on the File System Object(which for HDFS is an instance of Distributed File System).
- **Step 2:** Distributed File System(DFS) calls the name node, using remote procedure calls (RPCs), to determine the locations of the first few blocks in the file. For each block, the name node returns the addresses of the data nodes that have a copy of that block. The DFS returns an FSDataInputStream to the client for it to read data from. FSDataInputStream in turn wraps a DFSInputStream, which manages the data node and name node I/O.
- **Step 3:** The client then calls read() on the stream. DFSInputStream, which has stored the info node addresses for the primary few blocks within the file, then connects to the primary (closest) data node for the primary block in the file.
- **Step 4:** Data is streamed from the data node back to the client, which calls read() repeatedly on the stream.
- **Step 5:** When the end of the block is reached, DFSInputStream will close the connection to the data node, then finds the best data node for the next block. This happens transparently to the client, which from its point of view is simply reading an endless stream. Blocks are read as, with the DFSInputStream opening new connections to data nodes because the client reads through the stream. It will also call the name node to retrieve the data node locations for the next batch of blocks as needed.
- **Step 6:** When the client has finished reading the file, a function is called, close() on the FSDataInputStream.

HDFS Write


HDFS Client
1. Open
Distributed File System
2. Create
Name Node
namenode
7.Complete
3.Write
FS Data OutputStream
6.Close
Client JVM
4.Write Packet
5.Acknowledge Packet
Pipelines of Data Node
Data Node
datanode
4
5
Data Node
datanode
4
5
Data Node
datanode


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

- HDFS follows the Write once Read many times model. In HDFS we cannot edit the files which are already stored in HDFS, but we can append data by reopening the files

- **Step 1:** The client creates the file by calling create() on DistributedFileSystem(DFS).
- **Step 2:** DFS makes an RPC call to the name node to create a new file in the file system's namespace, with no blocks associated with it. The name node performs various checks to make sure the file doesn't already exist and that the client has the right permissions to create the file. If these checks pass, the name node prepares a record of the new file; otherwise, the file can't be created and therefore the client is thrown an error i.e. IOException. The DFS returns an FSDataOutputStream for the client to start out writing data to.
- **Step 3:** Because the client writes data, the DFSOutputStream splits it into packets, which it writes to an indoor queue called the info queue. The data queue is consumed by the DataStreamer, which is liable for asking the name node to allocate new blocks by picking an inventory of suitable data nodes to store the replicas. The list of data nodes forms a pipeline, and here we'll assume the replication level is three, so there are three nodes in the pipeline. The DataStreamer streams the packets to the primary data node within the pipeline, which stores each packet and forwards it to the second data node within the pipeline.
- **Step 4:** Similarly, the second data node stores the packet and forwards it to the third (and last) data node in the pipeline.
- **Step 5:** The DFSOutputStream sustains an internal queue of packets that are waiting to be acknowledged by data nodes, called an "ack queue".
- **Step 6:** This action sends up all the remaining packets to the data node pipeline and waits for acknowledgments before connecting to the name node to signal whether the file is complete or not.