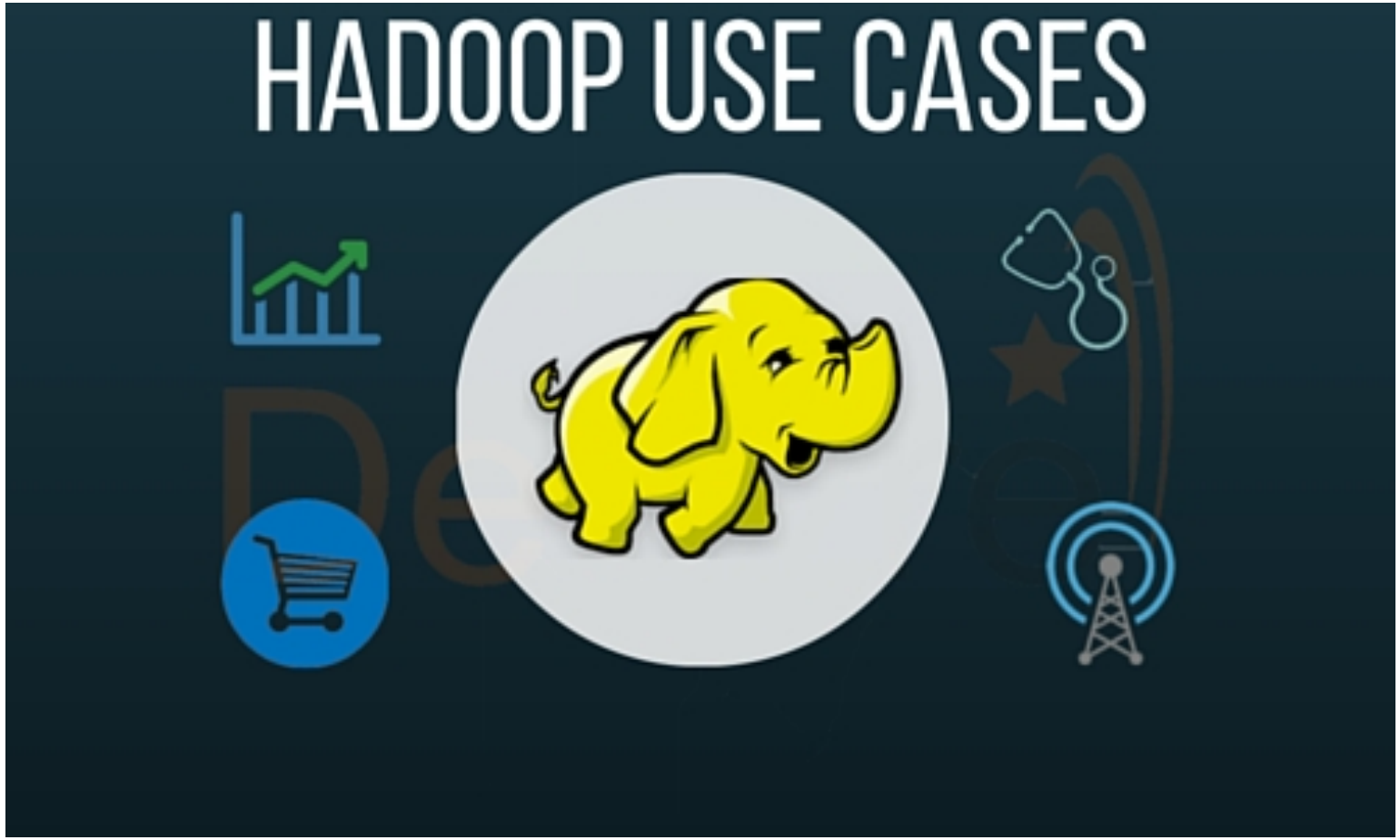
IT17701-DATA ANALYTICS
UNIT II- HDFS & Map Reduce

Contents

- Hadoop Overview
- Hadoop Use cases
- Hadoop Distributors
- HDFS
- Processing data with Hadoop
- Map Reduce
- Managing Resources and Applications with Hadoop YARN
- Interacting with Hadoop Ecosystems

HADOOP-OVERVIEW

- Hadoop is beginning to live up to its promise of being the backbone technology for Big Data storage and analytics. Companies across the globe have started to migrate their data into Hadoop to join the stalwarts who already adopted Hadoop a while ago.
- It is important to study and understand several Hadoop use cases for these simple reasons –
  - Hadoop is still a complex technology with several limitations and complications.
  - All Data is not Big Data and might not require a Hadoop solution.
  - Companies that use Hadoop, often fall into the error of not using the right Hadoop architecture and tools, thereby reducing Hadoop's efficiency.


HADOOP USE CASES

Hadoop Use Cases

- Studying Hadoop use cases will help to :

1) Understand what kind of big data problems need Hadoop

2) What sort of infrastructure should one have in order to set up and work on the Hadoop framework.

- The two obvious benefits of using Hadoop is that,
  - Provides storage for any kind of data from various sources
  - Provides a platform for proficient analytics of the data with low latency
- Hadoop is well known to be a distributed, scalable and fault-tolerant system.
- It can store petabytes with relatively low infrastructure investment. Hadoop runs on clusters of commodity servers. All such servers have local storage and CPU which can store few terabytes on its local disk.

Two Critical Components

- **Storage system** for Hadoop is known as HDFS.
  - HDFS system breaks the incoming data into multiple packets and distributes it among different servers connected in the clusters.
  - That way every server, stores a fragment of the entire data set and all such fragments are replicated on more than one server to achieve fault tolerance.
- **MapReduce** is a distributed data processing framework.
  - HDFS distributes a dataset to different servers but Hadoop MapReduce is the connecting framework responsible to distribute the work and aggregate the results obtained through data processing.

- Apache Hadoop provides solution to the problem caused by large volume of complex data. With the result of growth in data, additional servers can be used to store and analyze the data at low cost. This is complemented by processing power of the servers in a cluster by MapReduce.
- These two components define Hadoop, as it gained importance in data storage and analysis, over the legacy systems, due to its distributed processing framework.

HADOOP USE CASES

01

Morgan Stanley with assets over 350 billion is one of the world's biggest financial services organizations. It relies on the Hadoop framework to make industry critical investment decisions.

JPMorgan Chase has mentioned it on various channels that they prefer to use HDFS to support the exponentially growing size of data as well as for low latency processing of complex unstructured data.

HADOOP IN HEALTHCARE SECTOR

02

Using Hadoop, Medical insurance companies are able to create profitable policies suitable for different parts of the population by estimating general age of populace, below which individuals are not victim of a specific disease.

03

HADOOP FOR TELECOM INDUSTRY

Verticals of telecom industry that get benefitted with Hadoop Usage

- Call Data Records Management
- Servicing of Telecom Data Equipment
- Creating new products and services
- Advanced Telecom infrastructure planning
- Network traffic analytics

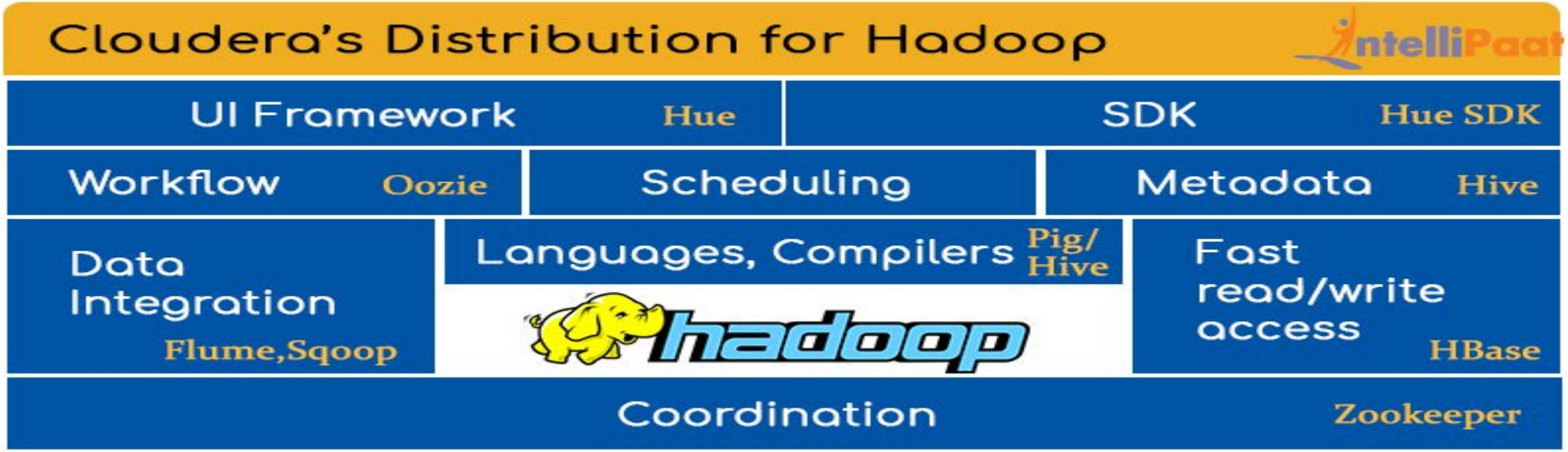
Nokia manages 100 TB of structured data along with 500+ TB of semi-structured data. Hadoop Distributed Framework System provided by Cloudera manages all variety of Nokia's data and processes it in a scale of petabytes.

https://www.dezyre.com/article/hadoop-use-cases/232

Hadoop Distributors

- **Top six vendors offering Big Data Hadoop solutions are:**

1. Cloudera
2. Hortonworks
3. Amazon Web Services Elastic MapReduce
4. Microsoft
5. MapR
6. IBM InfoSphere Insights



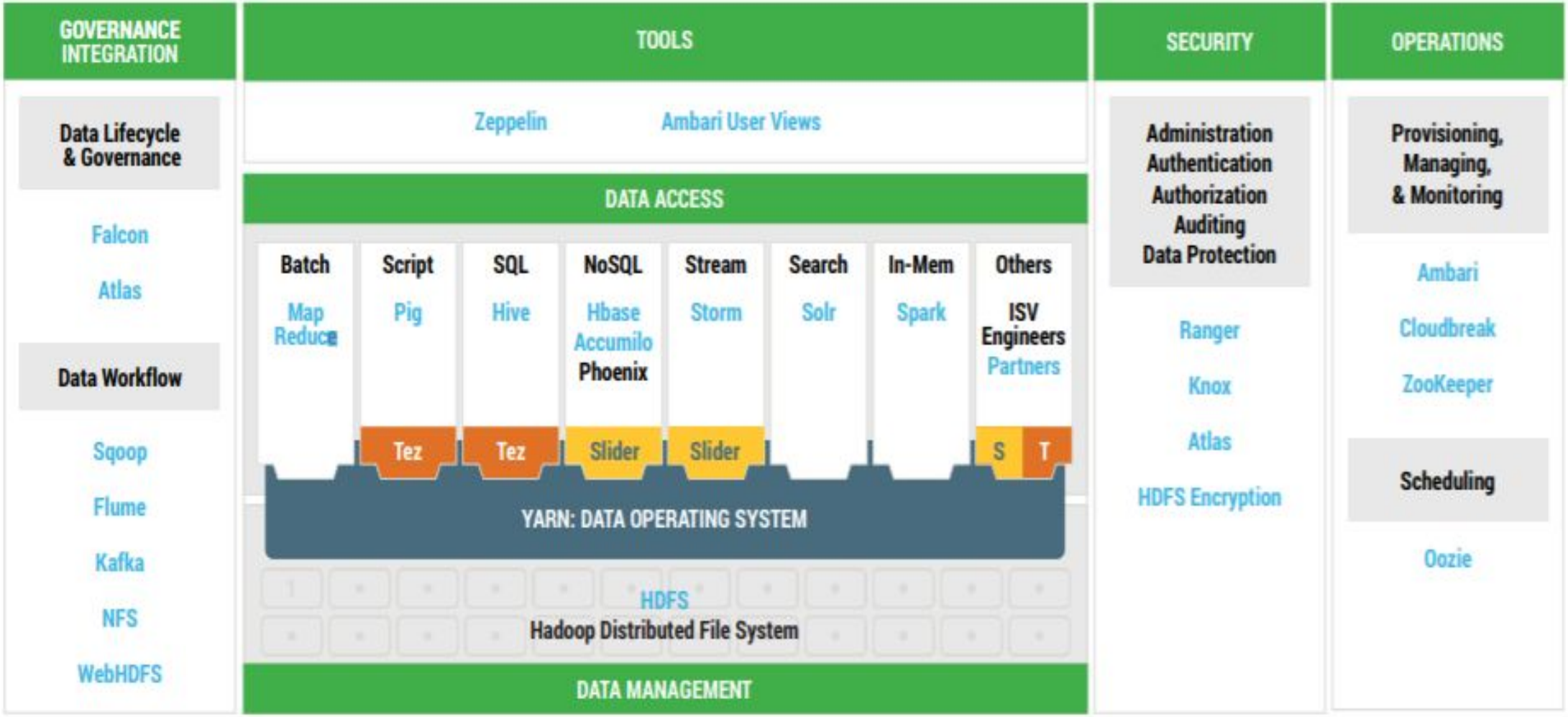
This ranks top over all the Big Data vendors for making Hadoop a reliable Big Data platform. Cloudera Hadoop vendor has around 350+ paying customers including US army, Allstate, and Monsanto.

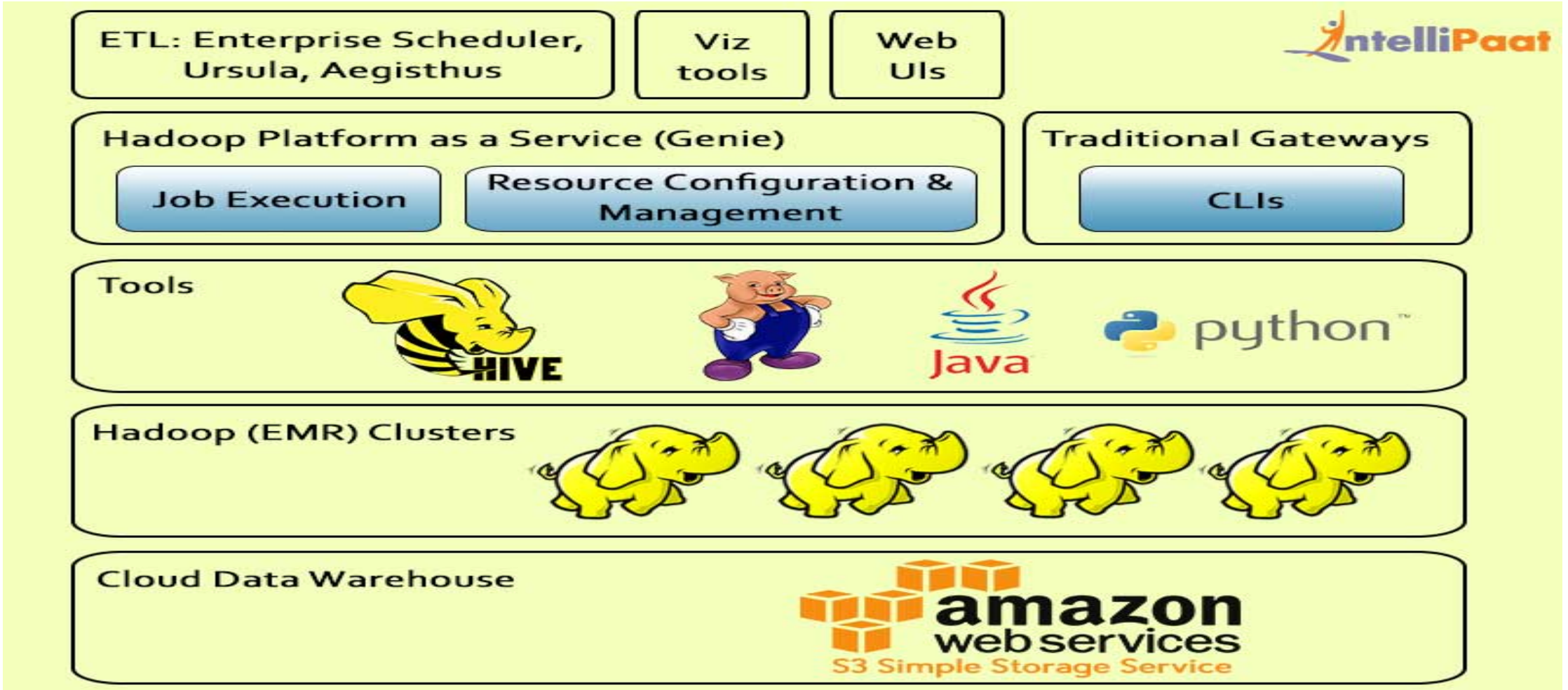
Cloudera occupies 53 percent of Hadoop market, followed by 11 percent by MapR, and 16 percent by Hortonworks. Cloudera's customers value the marketable add-on tools such as Cloudera Manager, Navigator, and Impala.

Hortonworks

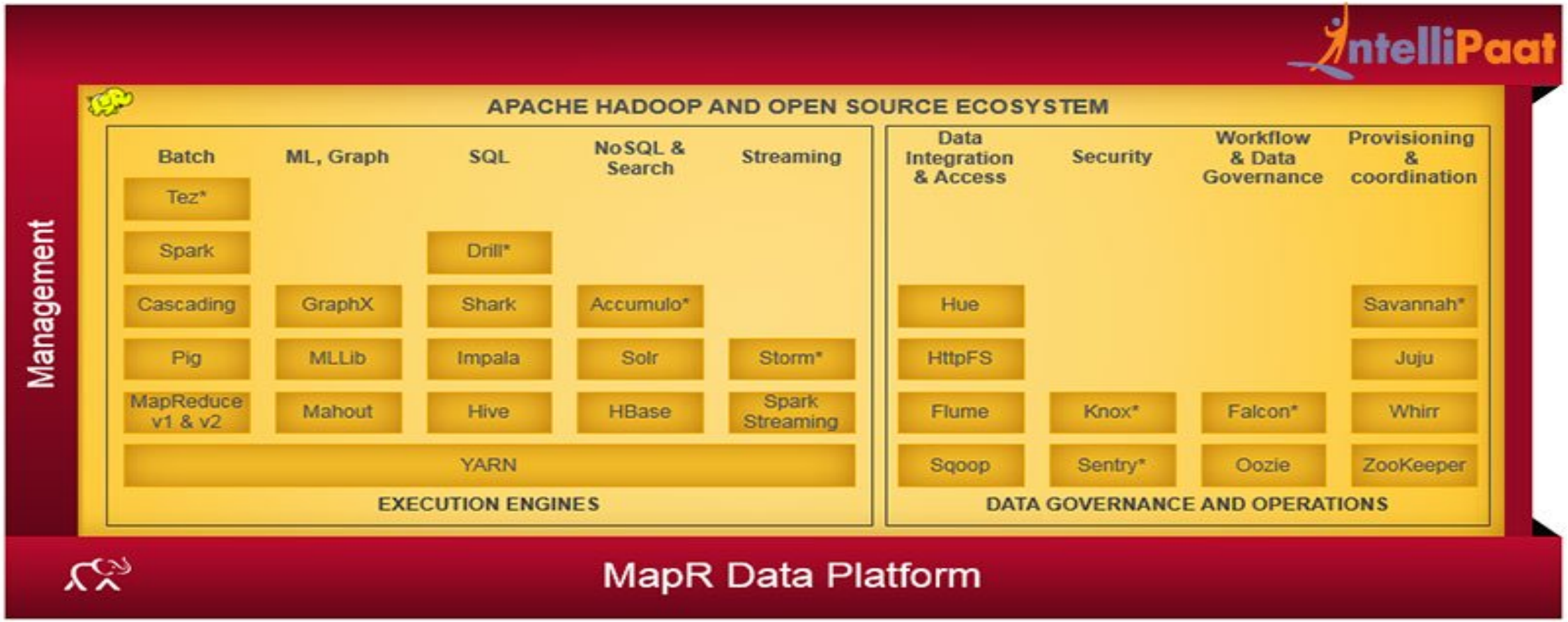
- Hortonworks is one among the top Hadoop vendors providing Big Data solutions in the Open Data Platform. It is one of the leading vendors as it promises 100 percent open-source distribution. It is also a prominent member of Open Data Platform initiative (ODPi) formed this year by IBM, Pivotal Software, and 12 other technology vendors.
- Apache Ambari is an illustration of the administration of Big Data Hadoop cluster tools developed by the vendors of Hortonworks for running, supervising, and controlling Big Data clusters. It is considered to be a focus for 60 fresh customers with massive accounts and has well-built manufacturing joint ventures with Red Hat Software, Microsoft, and Teradata.

Hortonworks

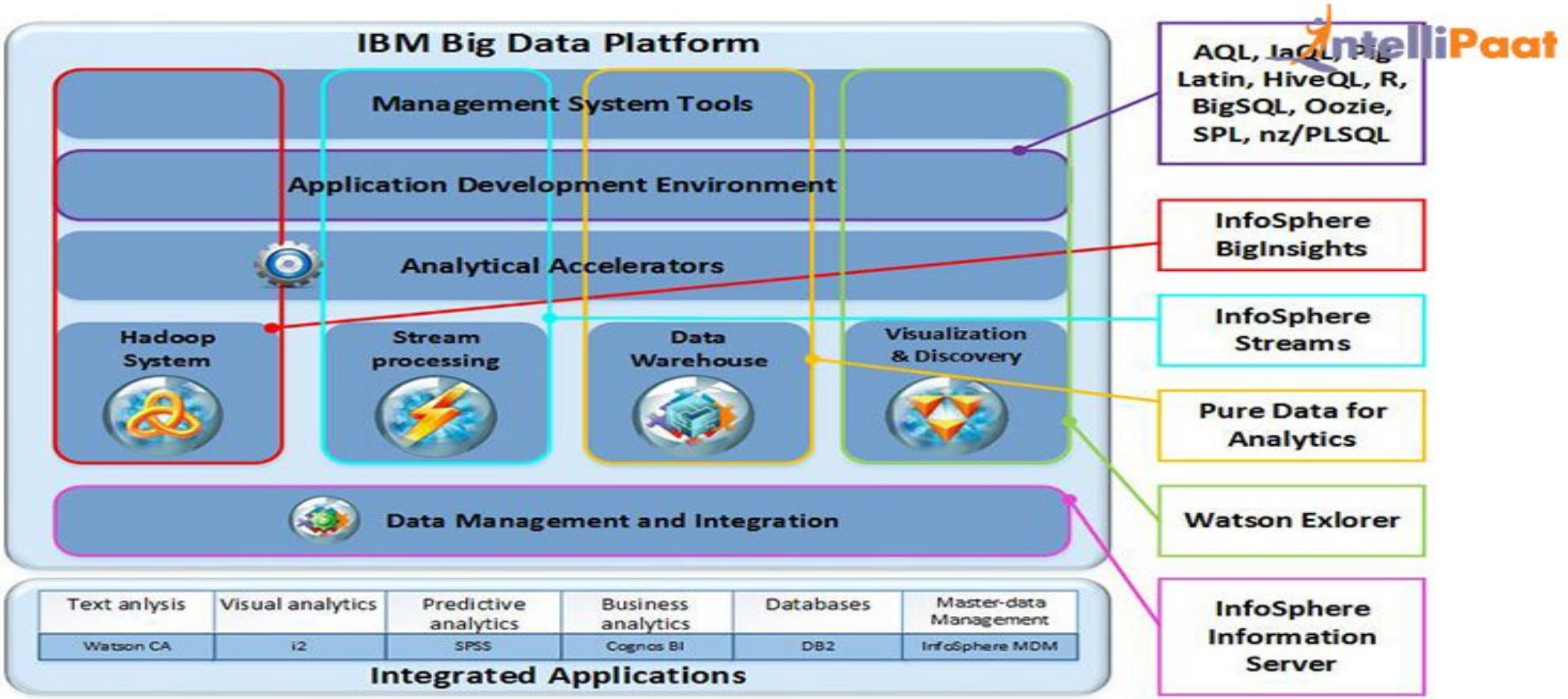




Amazon Elastic MapReduce is a part of Amazon Web Services (AWS), and it exists since the initial times of Hadoop. AWS has a simple-to-utilize and well-arranged data analytic stand built on influential HDFS structural design. It is one of the highest ranking vendors with the uppermost market distributions across the globe.



MapR technologies have been used to allow Hadoop to perform well with potential and minimal effort. Their linchpin, the MapR filesystem that inherits HDFS API, is fully read/write and can save trillions of files.
MapR has done more than any other vendor to deliver reliable and efficient distribution for huge cluster implementation.



IBM assimilates a capital of key data management parts and analytics assets into open-source distribution. The company has also launched a determined, open-source project Apache System ML for Machine Learning.
With IBM BigInsights, customers get to market in a very fast pace with their apps integrating advanced Big Data Analytics.

References

- https://intellipaat.com/blog/top-6-hadoop-vendors-providing-big-data-solutions-in-open-data-platform/
- https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
- https://www.tutorialspoint.com/hadoop/hadoop_hdfs_overview.htm

What is HDFS?

- HDFS is a distributed file system that provides access to data across Hadoop clusters.
- A cluster is a group of computers that work together.
- HDFS is a key tool that manages and supports analysis of very large volumes; petabytes and zettabytes of data.

Why HDFS?

- Before 2011, storing and retrieving petabytes or zettabytes of data had the following three major challenges: Cost, Speed, Reliability.
- Traditional file system approximately costs $10,000 to $14,000, per terabyte. Searching and analyzing data was time-consuming and expensive.
- Also, if search components were saved on different servers, fetching data was difficult.
- Cost
  - HDFS is open-source software so that it can be used with zero licensing and support costs. It is designed to run on a regular computer.
- Speed
  - Large Hadoop clusters can read or write more than a terabyte of data per second. A cluster comprises multiple systems logically interconnected in the same network.
  - HDFS can easily deliver more than two gigabytes of data per second, per computer to MapReduce, which is a data processing framework of Hadoop.

Why HDFS?

- Reliability
  - HDFS copies the data multiple times and distributes the copies to individual nodes. A node is a commodity server which is interconnected through a network device.
  - HDFS then places at least one copy of data on a different server. In case, any of the data is deleted from any of the nodes; it can be found within the cluster.
- A regular file system, like a Linux file system, is different from HDFS with respect to the size of the data. In a regular file system, each block of data is small, usually about 51 bytes. However, in **HDFS, each block is 128 Megabytes by default.**
- A regular file system provides access to large data but may suffer from disk input/output problems mainly due to multiple seek operations.
- On the other hand, HDFS can **read large quantities of data sequentially after a single seek operation**. This makes HDFS unique since all of these operations are performed in a distributed mode.

Characteristics of HDFS

- HDFS has high fault-tolerance
  - HDFS may consist of thousands of server machines. Each machine stores a part of the file system data. HDFS detects faults that can occur on any of the machines and recovers it quickly and automatically.
- HDFS has high throughput
  - HDFS is designed to store and scan millions of rows of data and to count or add some subsets of the data. The time required in this process is dependent on the complexities involved.
  - It has been designed to support large datasets in batch-style jobs. However, the emphasis is on high throughput of data access rather than low latency.

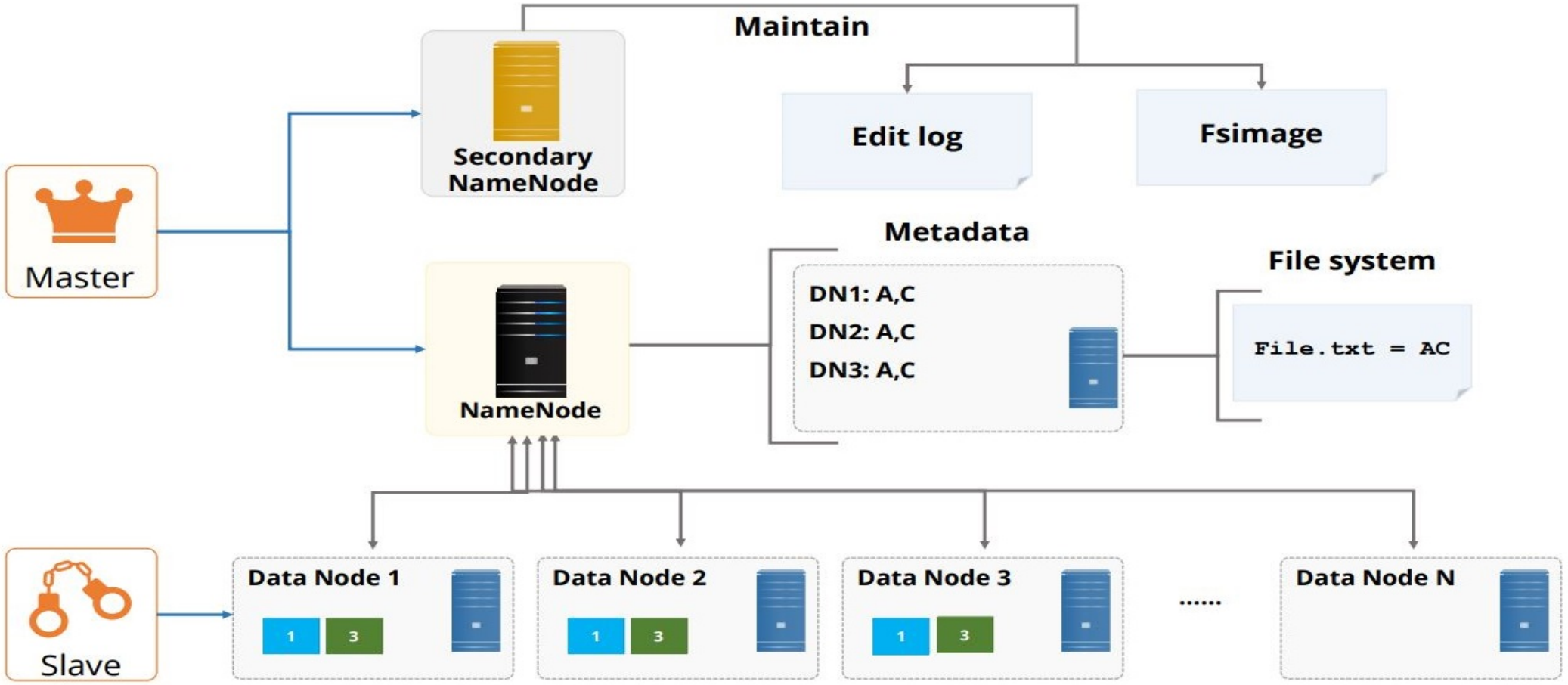
Characteristics of HDFS

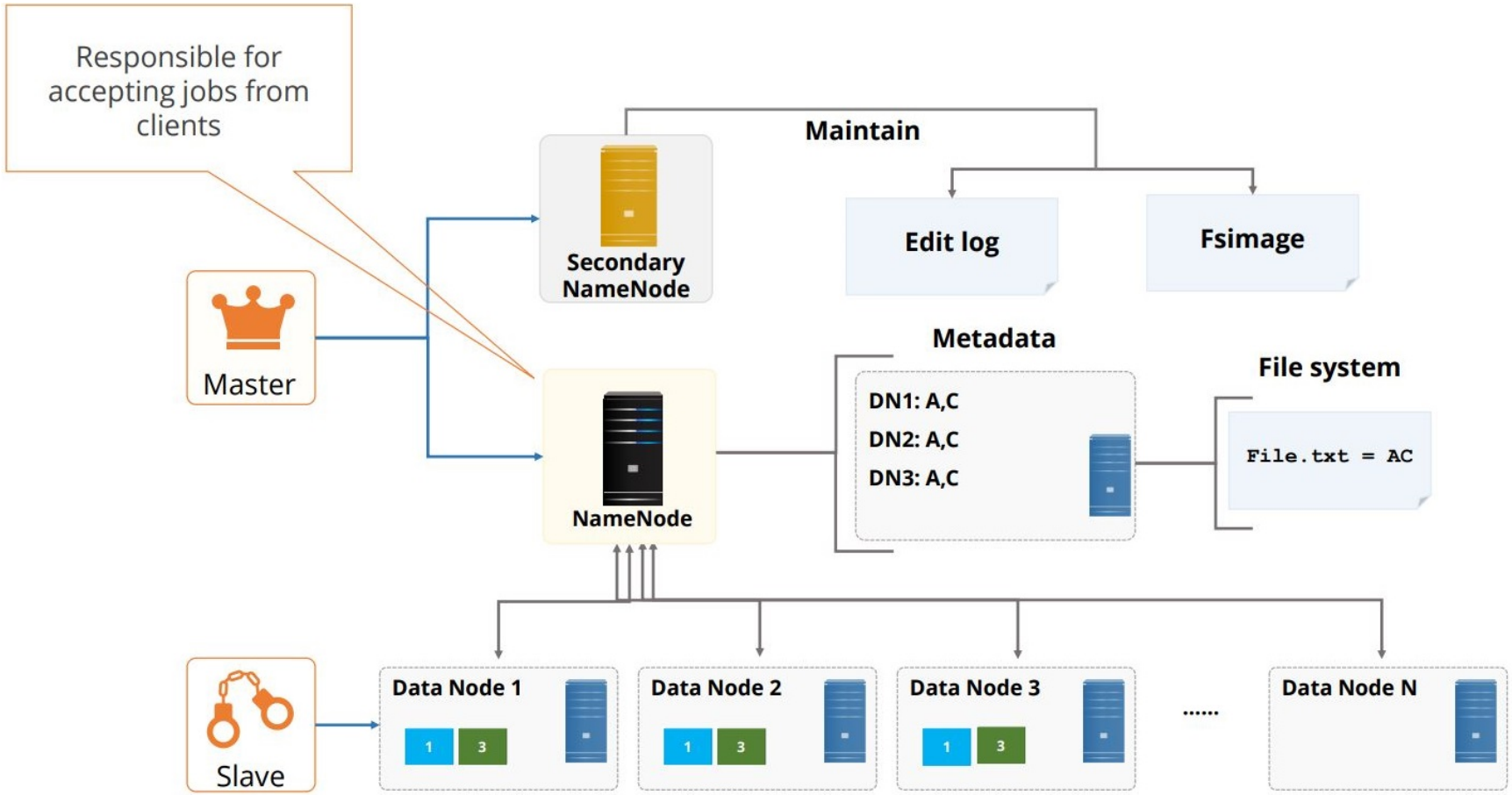
- HDFS is economical
  - HDFS is designed in such a way that it can be built on commodity hardware and heterogeneous platforms, which is low-priced and easily available.
  - HDFS stores files in a number of blocks. Each block is replicated to a few separate computers. The count of replication can be modified by the administrator. Data is divided into 128 Megabytes per block and replicated across local disks of cluster nodes. Metadata controls the physical location of a block and its replication within the cluster. It is stored in NameNode HDFS is the storage system for both input/output of MapReduce jobs. Let’s understand how HDFS stores files with an example. .

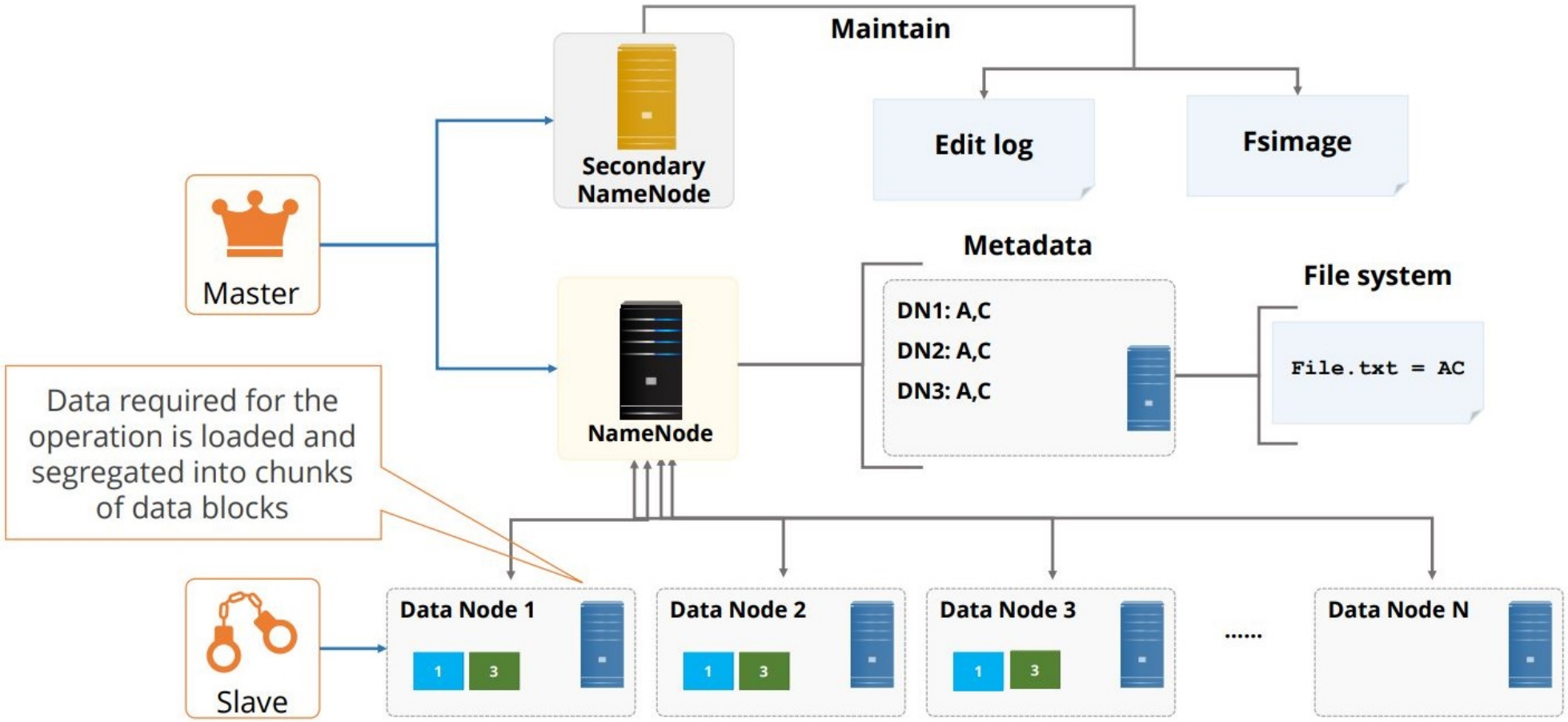
How Does HDFS Work?

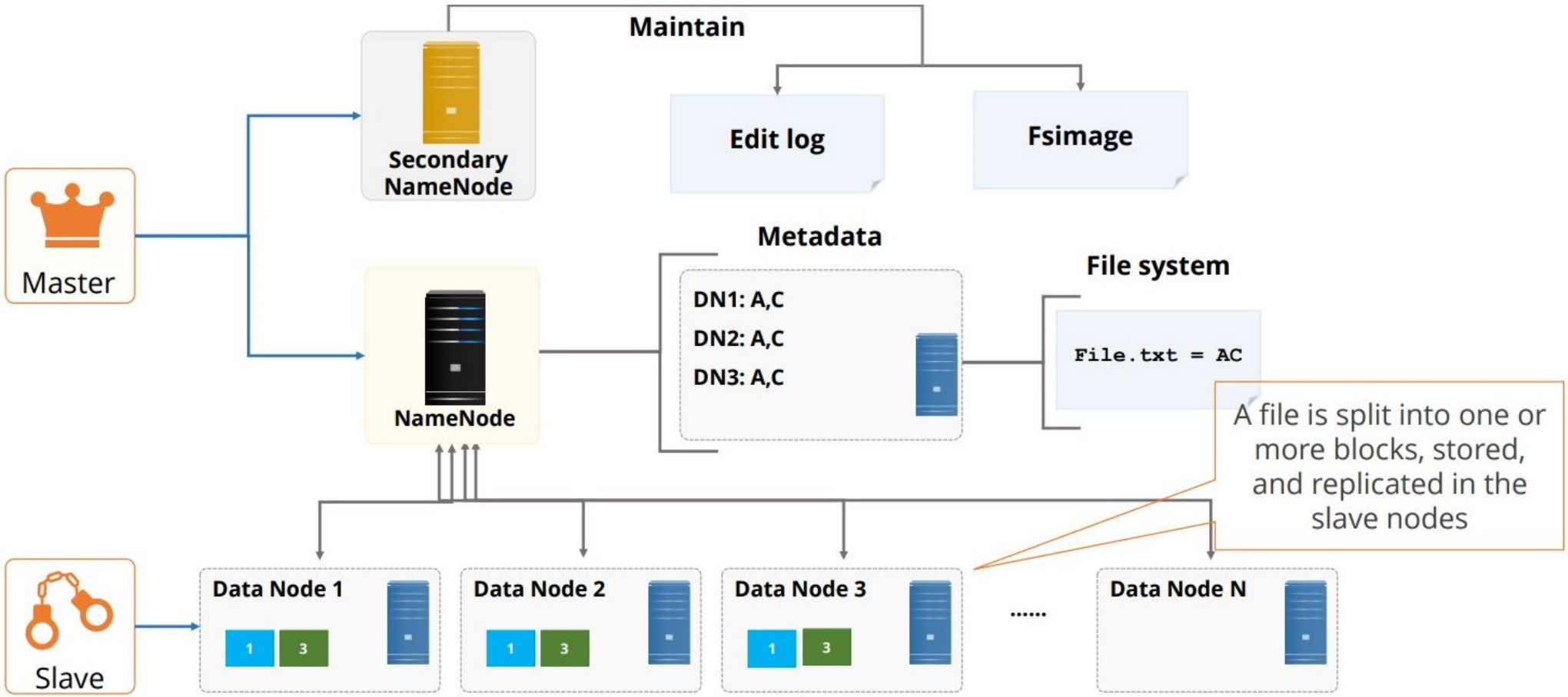
- Example - A patron gifted a collection of popular books to a college library. The librarian decided to arrange the books on a small rack and then distribute multiple copies of each book on other racks. This way the students could easily pick up a book from any of the racks.
- Similarly, HDFS creates multiple copies of a data block and keeps them in separate systems for easy access.

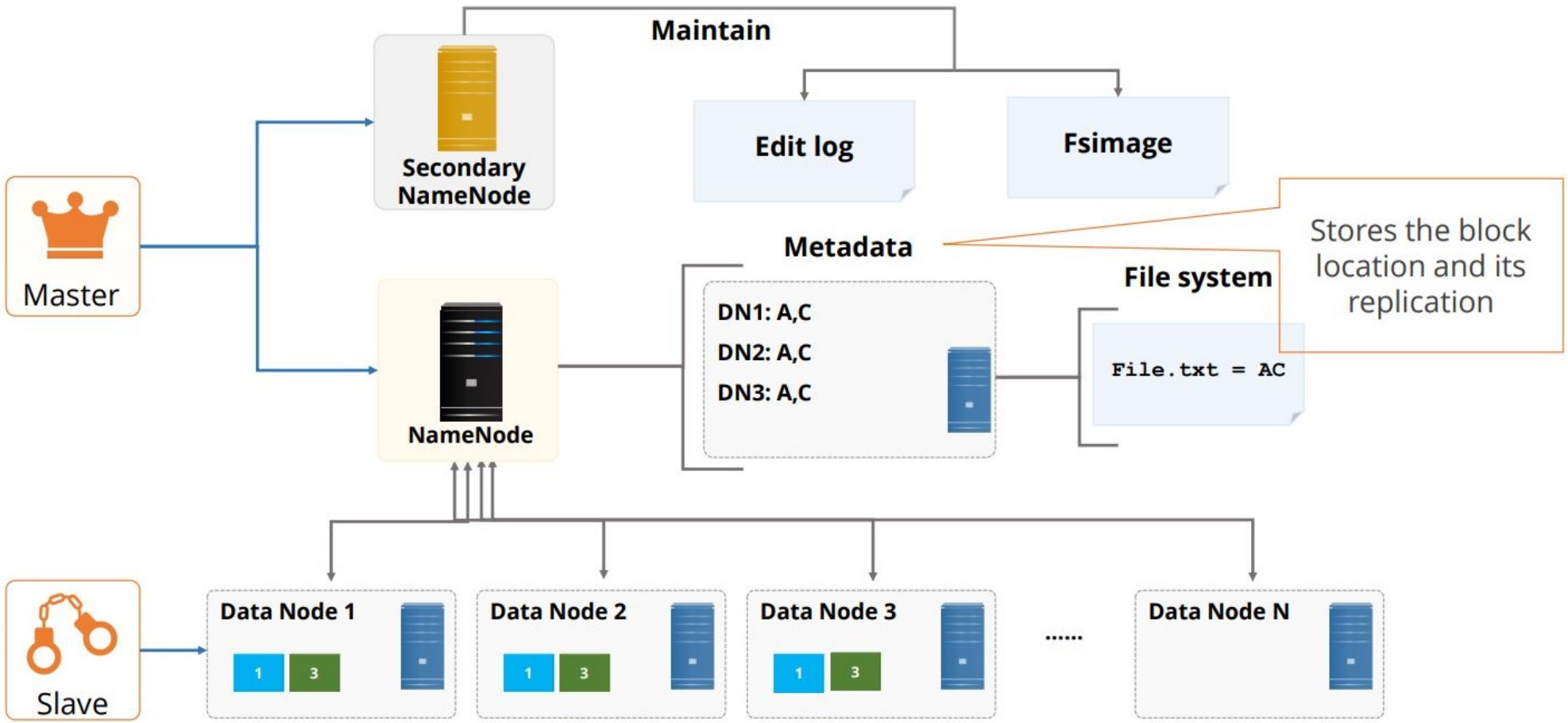
HDFS Architecture and Components




Responsible for accepting jobs from clients
Maintain
Secondary NameNode
Edit log
Fsimage
Master
Metadata
File system
DN1: A,C
DN2: A,C
DN3: A,C
File.txt = AC
NameNode
Slave
Data Node 1
Data Node 2
Data Node 3
.....
Data Node N
1
3
1
3
1
3


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


Maintain
Secondary NameNode
Edit log
Fsimage
Master
Metadata
File system
NameNode
DN1: A,C
DN2: A,C
DN3: A,C
File.txt = AC
A file is split into one or more blocks, stored, and replicated in the slave nodes
Slave
Data Node 1
Data Node 2
Data Node 3
.....
Data Node N
1
3


Maintain
Secondary NameNode
Edit log
Fsimage
Master
Metadata
Stores the block location and its replication
File system
NameNode
DN1: A,C
DN2: A,C
DN3: A,C
File.txt = AC
Slave
Data Node 1
Data Node 2
Data Node 3
.....
Data Node N
1
3
1
3
1
3

- 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.

An HDFS cluster can be managed using the following features:

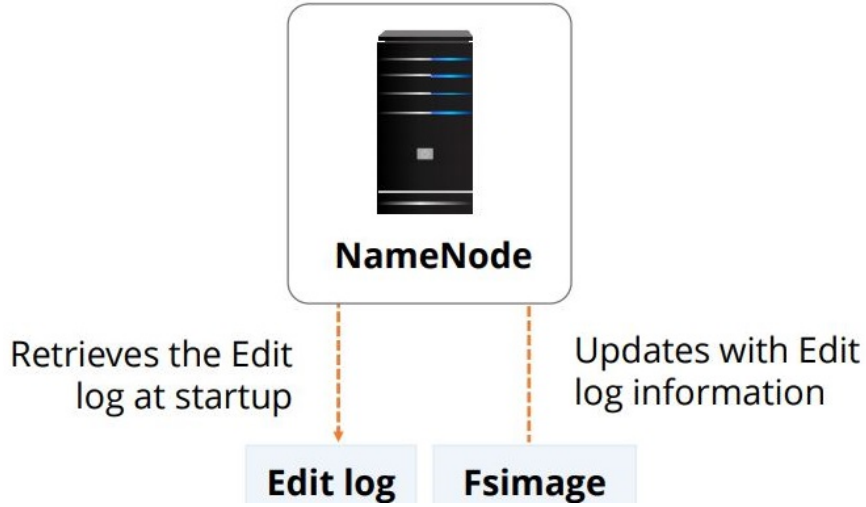
- Quorum-based storage: Quorum-based Storage refers to the HA implementation that uses Quorum Journal Manager, or QJM. During this implementation, the Standby node keeps its state synchronized with the Active node through a group of separate daemons called JournalNodes Daemons are long-running processes that typically start up with the system and listen for requests from the client processes. Each daemon runs in its own Java Virtual Machine (JVM). When any namespace modification is performed by the Active node, it durably logs a record of the modification to a majority of the JournalNodes. The Standby node reads the edits from the JournalNodes and constantly watches for changes to the edit log. As the Standby node edits, it applies them to its own namespace. In the event of a failover, the Standby ensures that it has read all the edits from the JournalNodes before it promotes itself to the active state. This ensures the namespace state is fully synchronized before a failover occurs.
- Shared storage using Network File System: In shared storage using NFS implementation, the Standby node keeps its state synchronized with the Active node through access to a directory on a shared storage device.

HDFS Components

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

Namenode

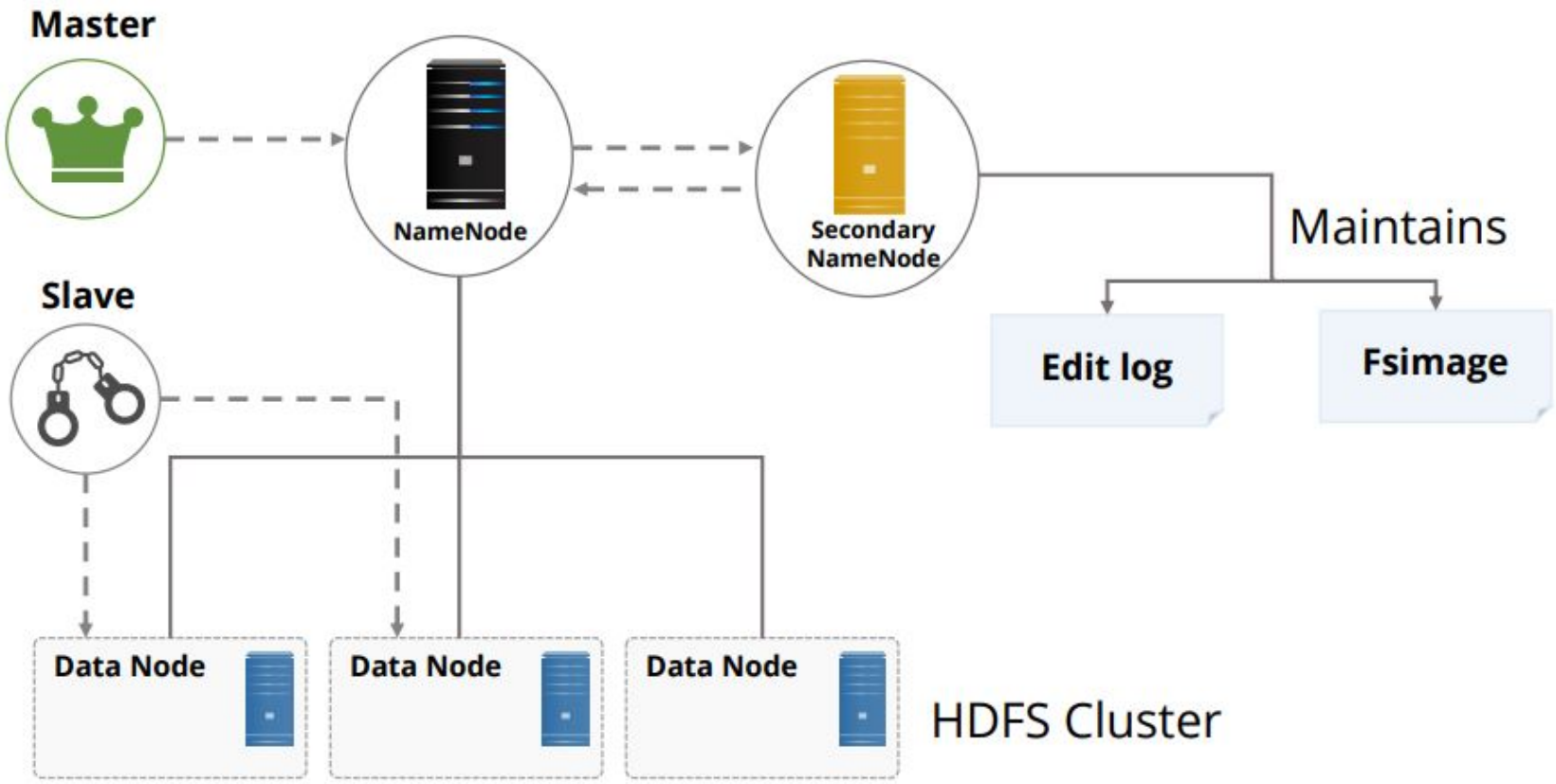
- The NameNode server is the core component of an HDFS cluster. There can be only one NameNode server in an entire cluster. Namenode maintains and executes the file system namespace operation such as opening, closing, and renaming of files and directories, which are present in HDFS.



- The namespace image and the edit log stores information of the data and the metadata. NameNode also determines the linking of blocks to DataNodes. Furthermore, the NameNode is a single point of failure.
- The DataNode is a multiple instance server. There can be several numbers of DataNode servers. The number depends on the type of network and the storage system.
- The DataNode servers, stores, and maintains the data blocks. The NameNode Server provisions the data blocks on the basis of the type of job submitted by the client.
- DataNode also stores and retrieves the blocks when asked by clients or the NameNode. Furthermore, it reads/writes requests and performs block creation, deletion, and replication of instruction from the NameNode. There can be only one Secondary NameNode server in a cluster. Note that you cannot treat the Secondary NameNode server as a disaster recovery server. However, it partially restores the NameNode server in case of a failure.

Secondary Namenode

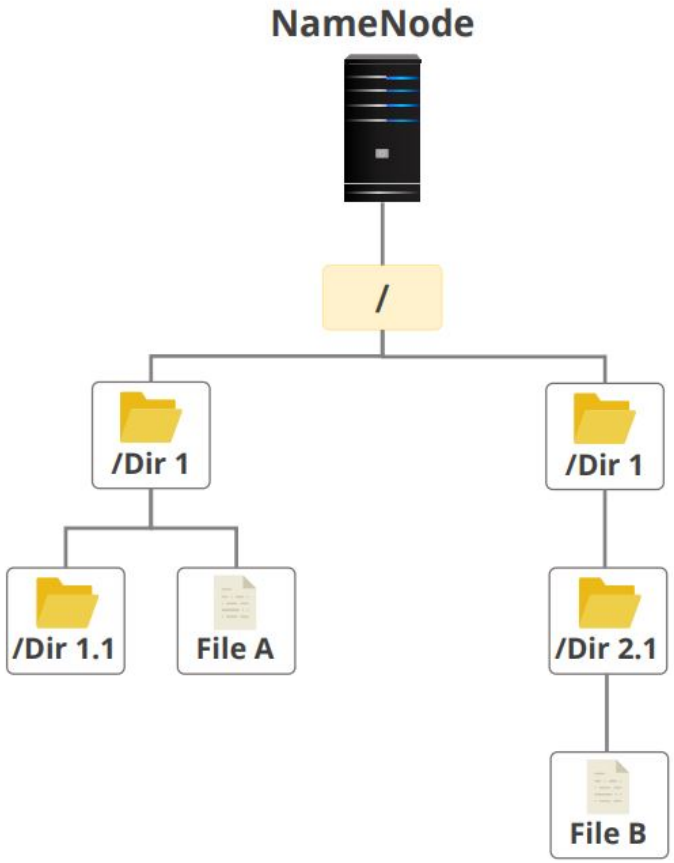
- The Secondary NameNode server maintains the edit log and namespace image information in sync with the NameNode server. At times, the namespace images from the NameNode server are not updated; therefore, you cannot totally rely on the Secondary NameNode server for the recovery process.


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.

File System

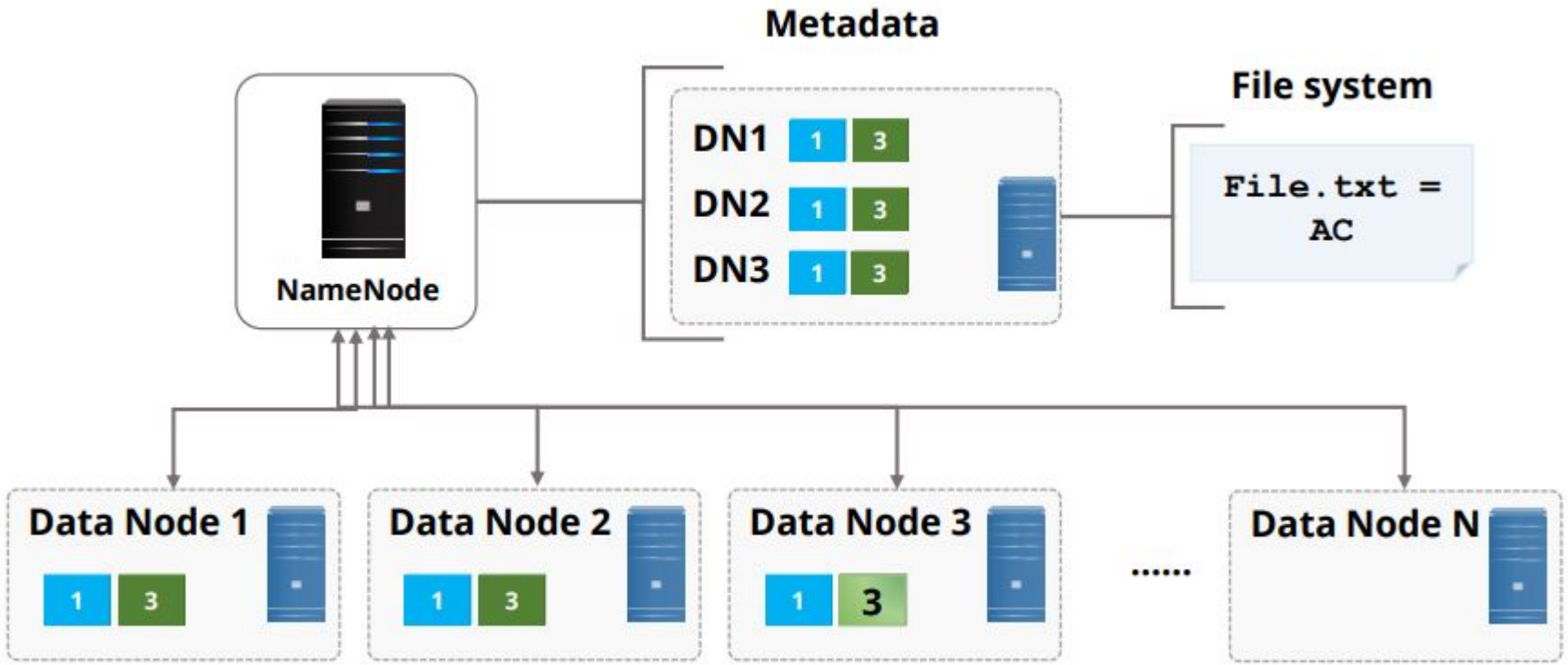


- A file system supports operations like create, remove, move, and rename. The NameNode, apart from maintaining the file system namespace, records any change to metadata information.
- Now that we have learned about HDFS components, let us see how NameNode works along with other components.

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.

Namenode: Operation



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

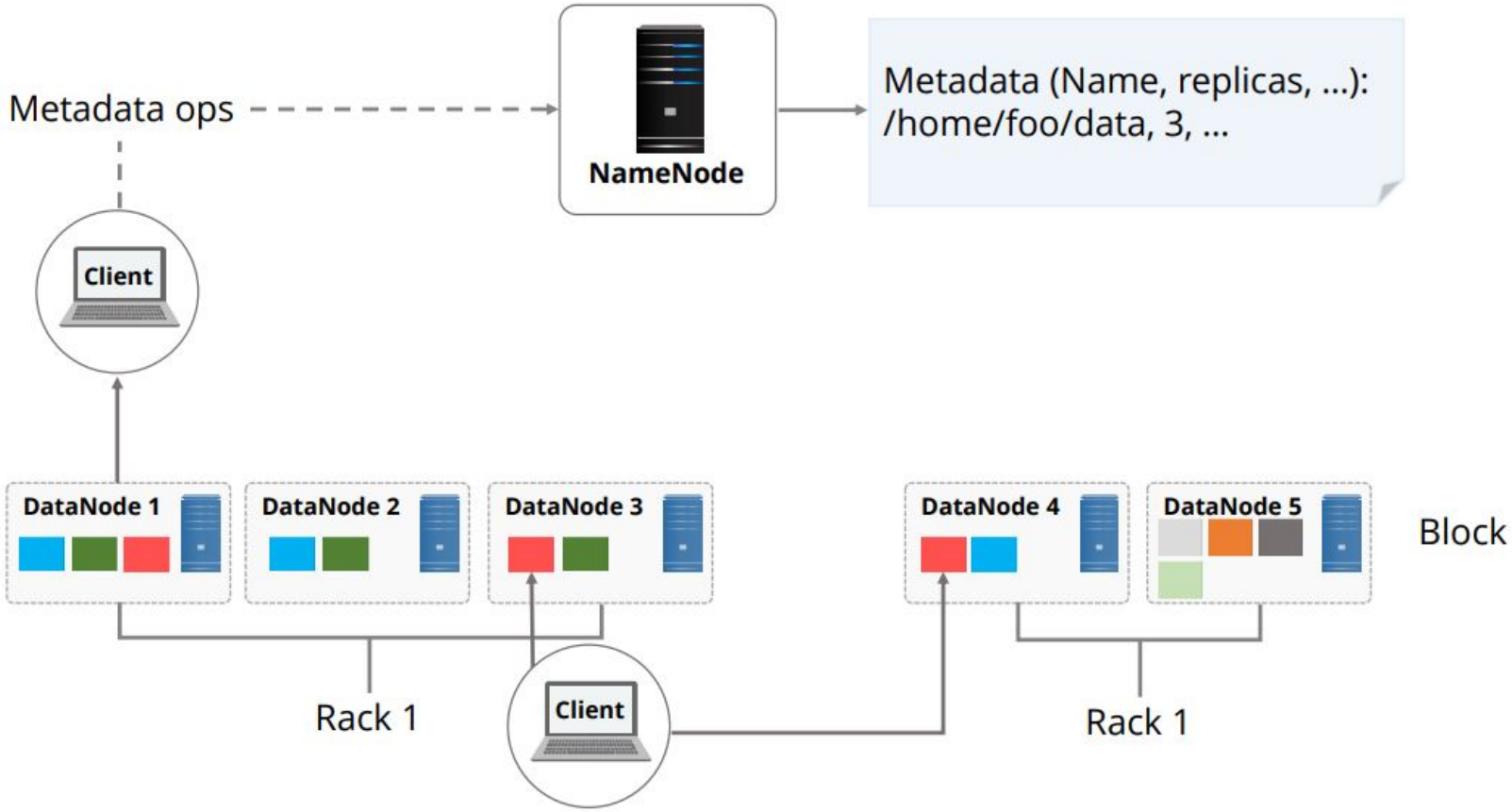
- When new DataNodes join a cluster, metadata loads the blocks that reside on a specific DataNode into its memory at startup. Metadata then periodically loads the data at user-defined or default intervals.
- When the NameNode starts up, it retrieves the Edit Log and FsImage from its local file system. 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 architecture. As discussed earlier, each 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 decreases.

DataNode

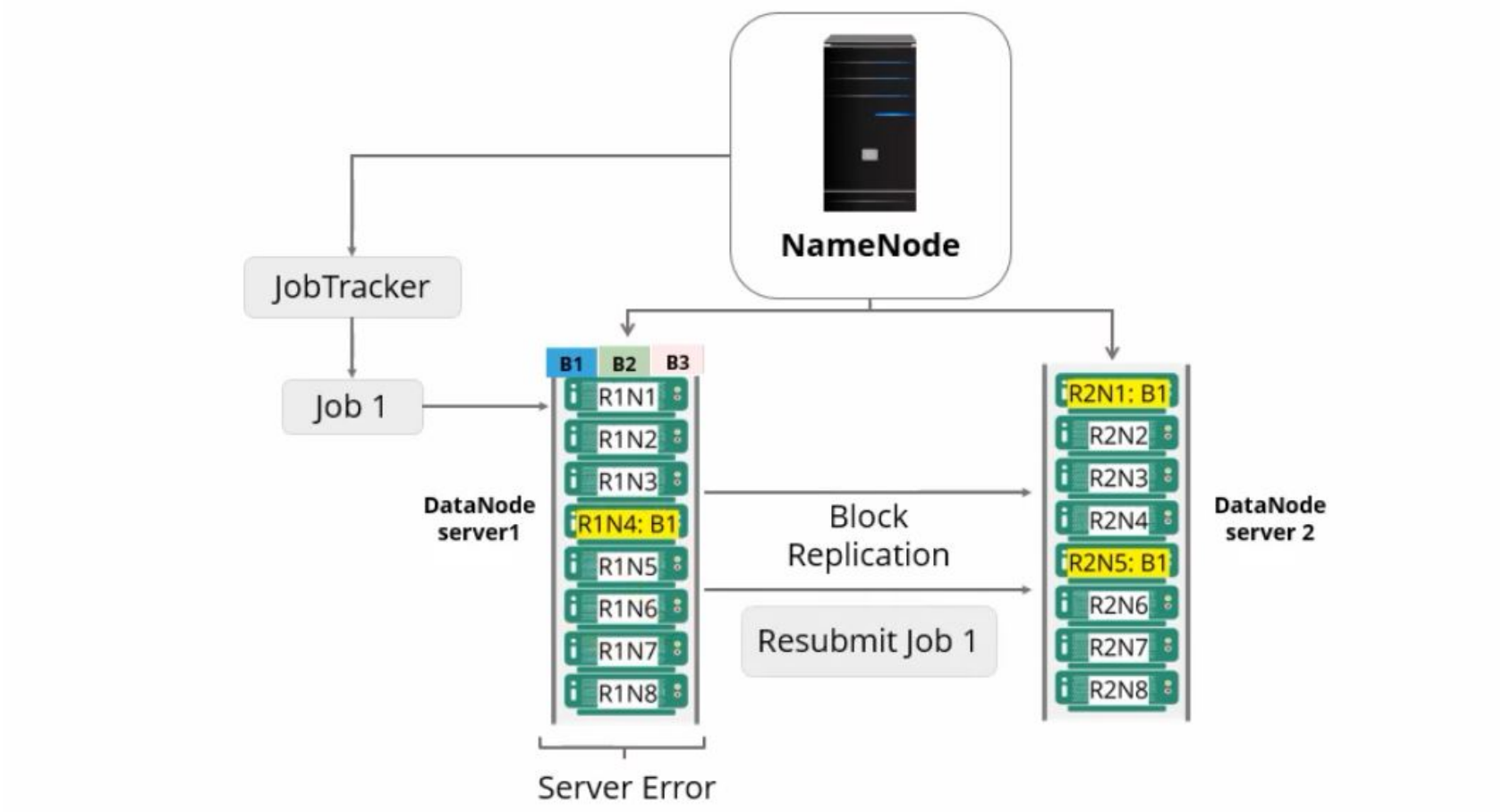

Metadata ops
NameNode
Metadata (Name, replicas, ...):
/home/foo/data, 3, ...
Client
DataNode 1
DataNode 2
DataNode 3
DataNode 4
DataNode 5
Block
Rack 1
Client
Rack 1

- The data block approach provides:
  - Simplified replication
  - Fault-tolerance
  - Reliability.
- It also helps by shielding users from storage sub-system details.

Block Replication Architecture

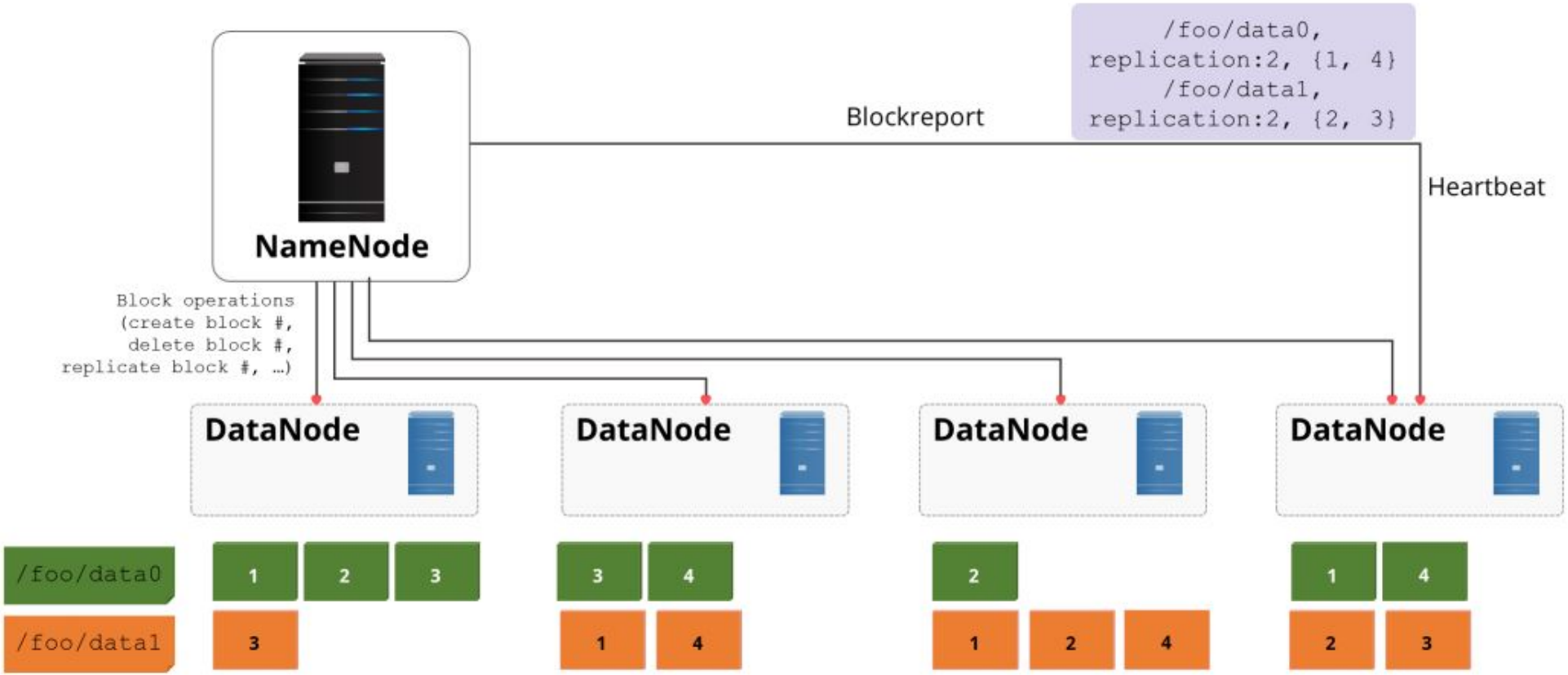
- Block replication refers to creating copies of a block in multiple data nodes. Usually, the data is split into the forms of parts such as part and part one.

Block Replication Architecture

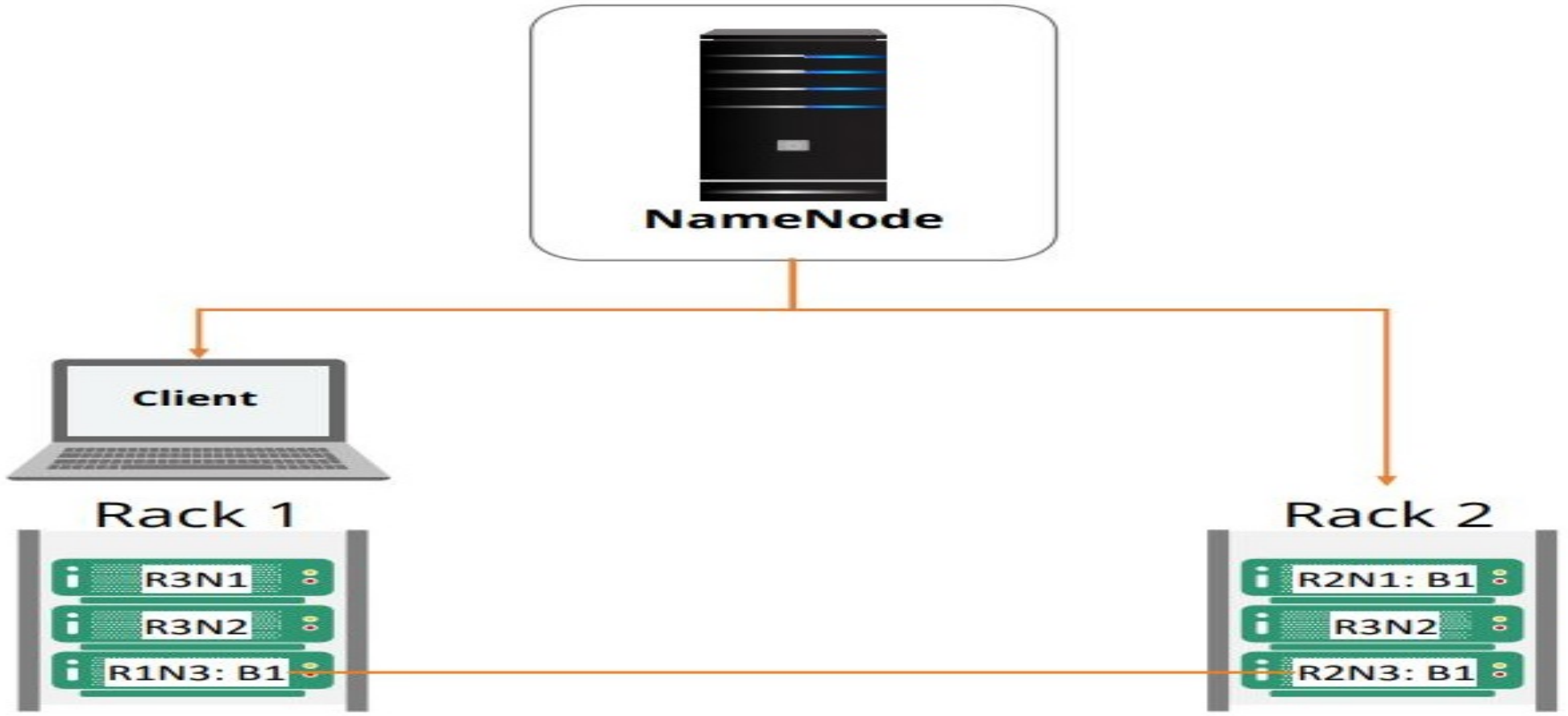


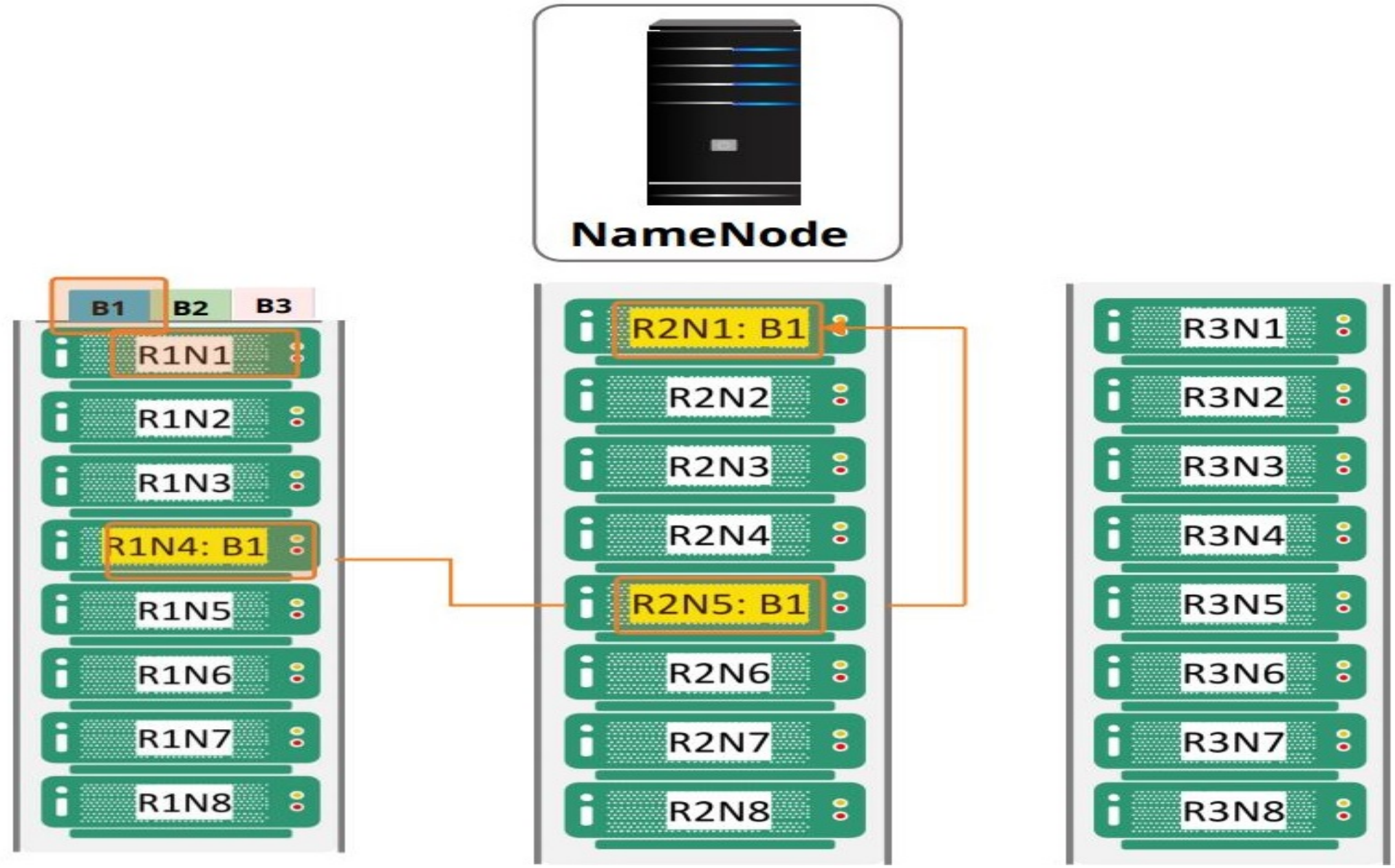
Replication Method

- 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.

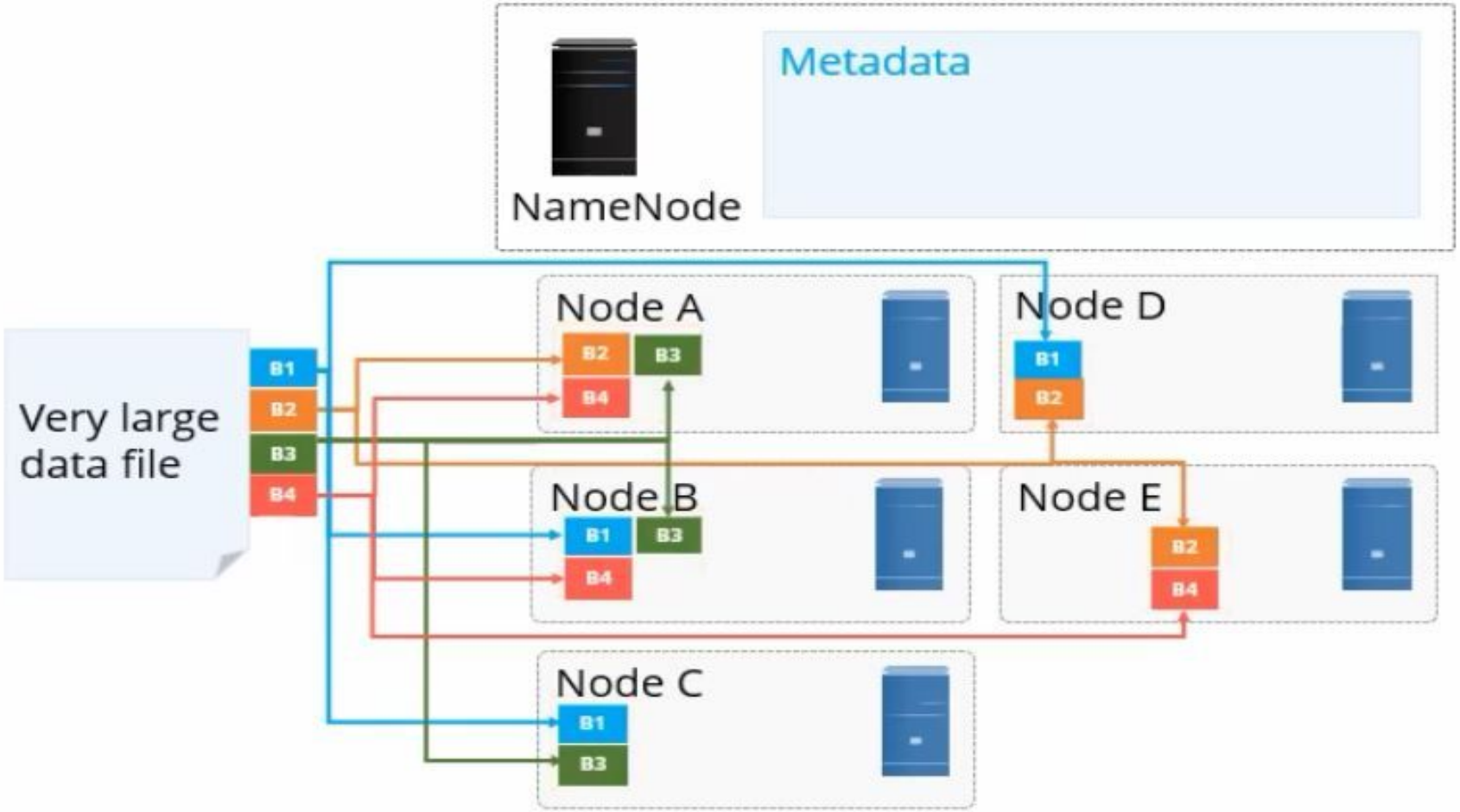

/foo/data0,
replication:2, {1, 4}
/foo/data1,
replication:2, {2, 3}
Blockreport
Heartbeat
NameNode
Block operations
(create block #,
delete block #,
replicate block #, …)
DataNode
DataNode
DataNode
DataNode
/foo/data0
/foo/data1
1
2
3
3
3
4
1
4
2
1
2
4
1
4
2
3

Data Replication Topology



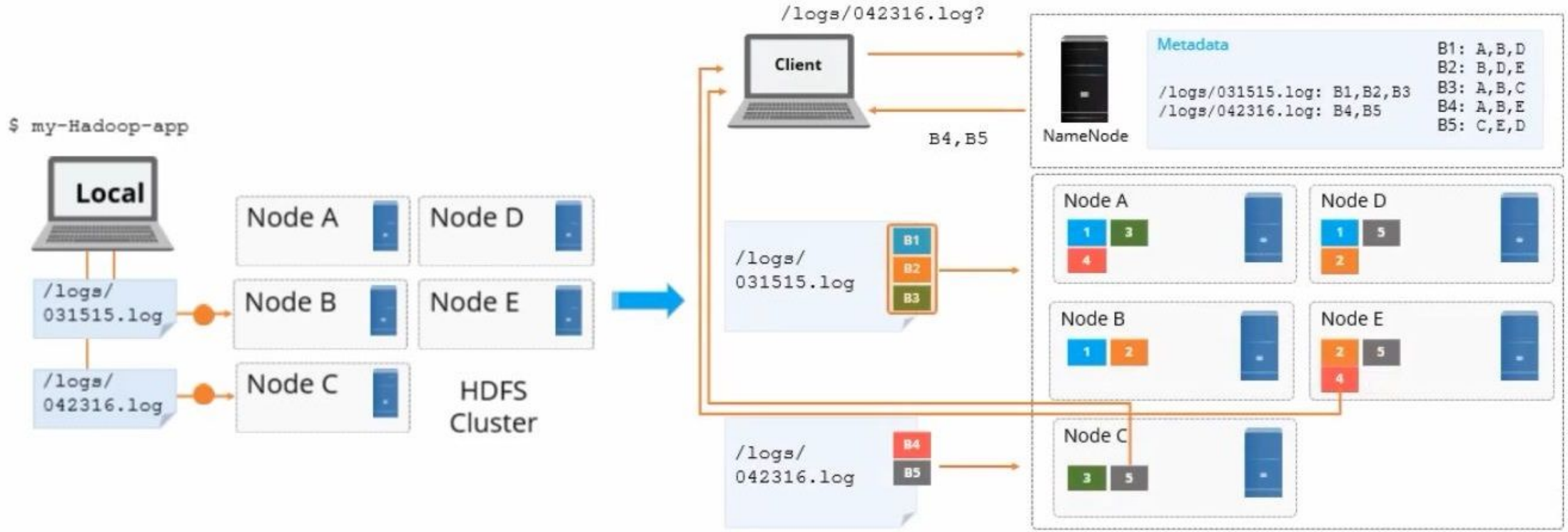

NameNode
B1
B2
B3
R1N1
R1N2
R1N3
R1N4: B1
R1N5
R1N6
R1N7
R1N8
R2N1: B1
R2N2
R2N3
R2N4
R2N5: B1
R2N6
R2N7
R2N8
R3N1
R3N2
R3N3
R3N4
R3N5
R3N6
R3N7
R3N8

How Are Files Stored?

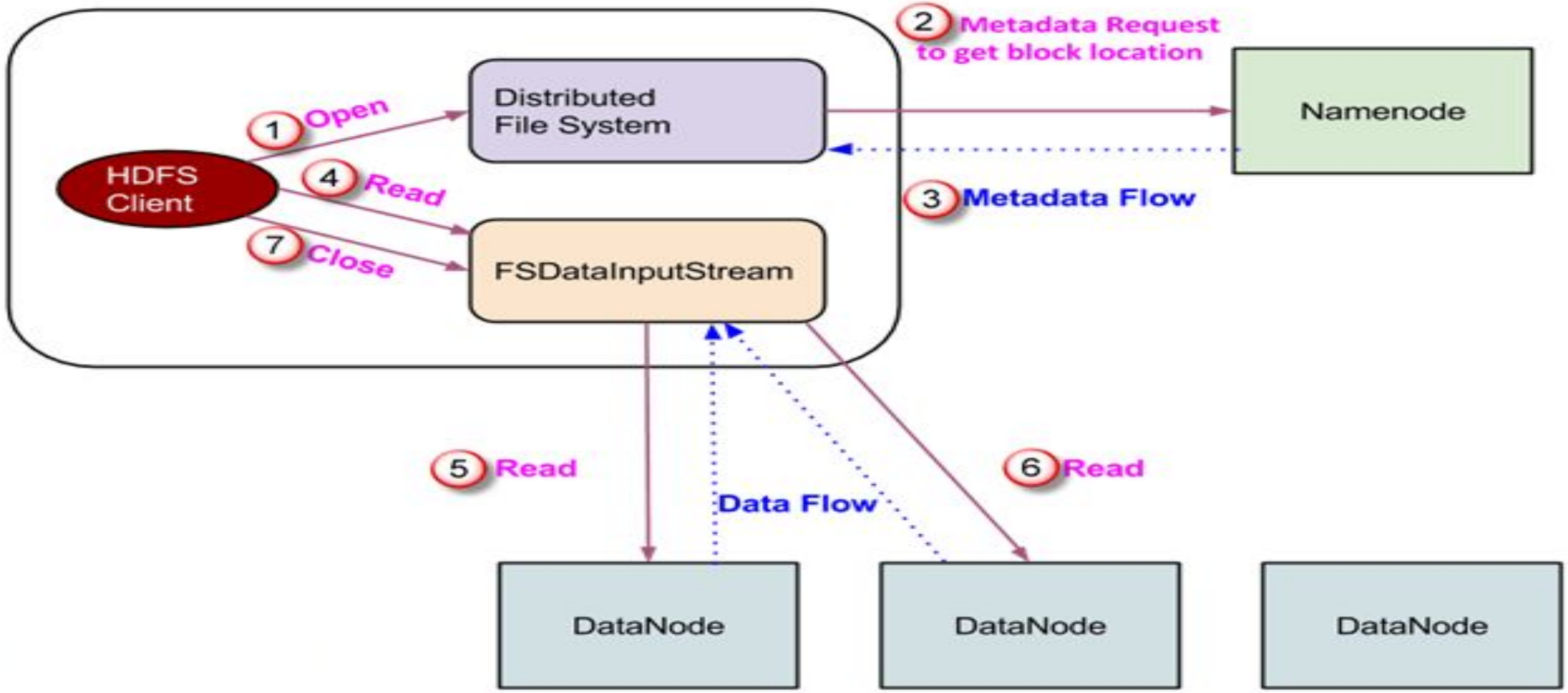


Example

- 2 log files 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.


$ my-Hadoop-app
Local
/logs/
031515.log
/logs/
042316.log
Node A
Node D
Node B
Node E
Node C
HDFS
Cluster
/logs/042316.log?
Client
B4,B5
NameNode
Metadata
/logs/031515.log: B1,B2,B3
/logs/042316.log: B4,B5
B1: A,B,D
B2: B,D,E
B3: A,B,C
B4: A,B,E
B5: C,E,D
/logs/
031515.log
B1
B2
B3
/logs/
042316.log
B4
B5
Node A
Node D
Node B
Node E
Node C

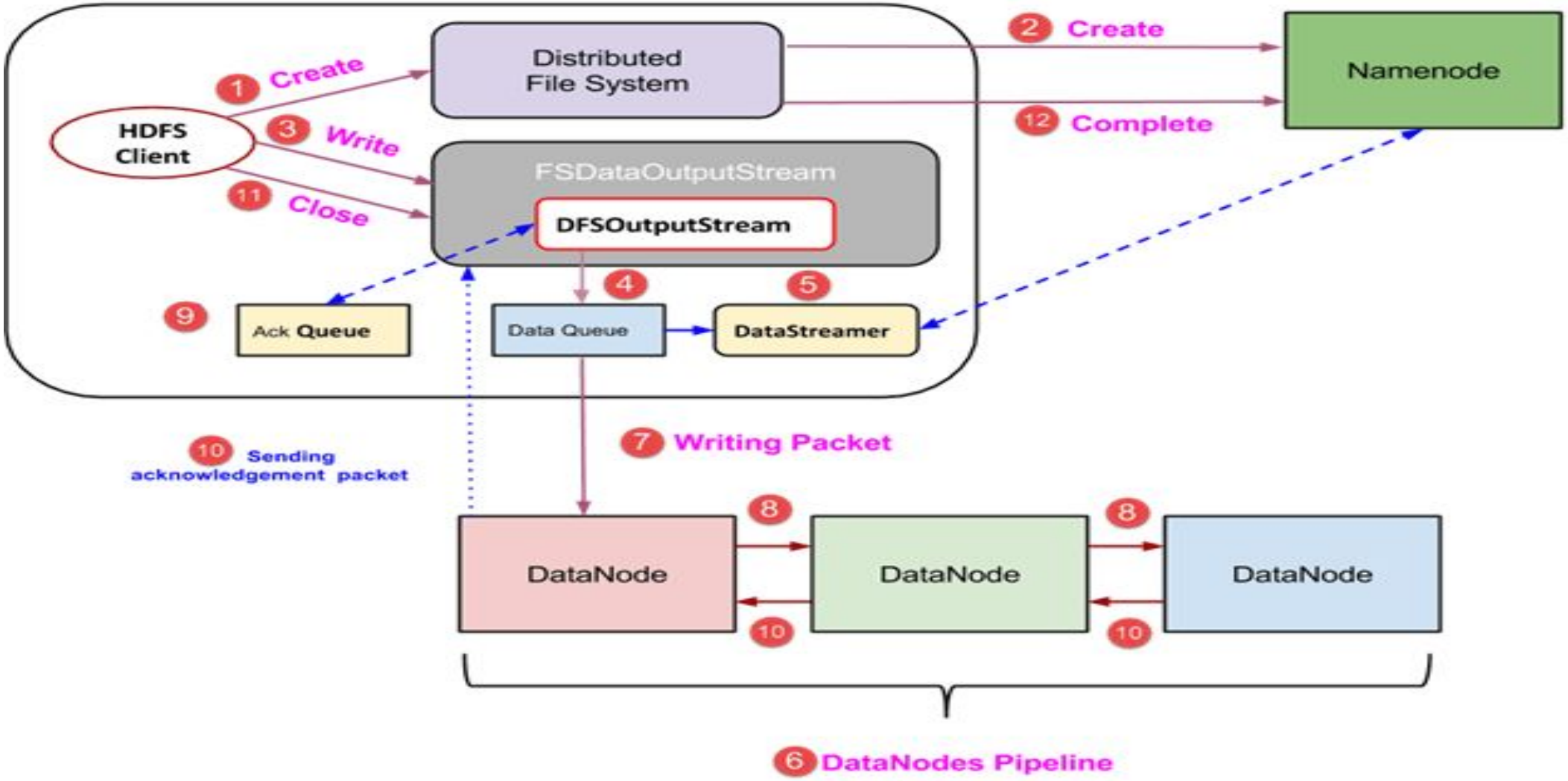
READ OPERATION



READ OPERATION

1. A client initiates read request by calling **'open()'** method of FileSystem object; it is an object of type **DistributedFileSystem**.
2. This object connects to namenode using RPC and gets metadata information such as the locations of the blocks of the file. Please note that these addresses are of first few blocks of a file.
3. In response to this metadata request, addresses of the DataNodes having a copy of that block is returned back.
4. Once addresses of DataNodes are received, an object of type **FSDataInputStream** is returned to the client. **FSDataInputStream** contains **DFSInputStream** which takes care of interactions with DataNode and NameNode. In step 4 shown in the above diagram, a client invokes **'read()'** method which causes **DFSInputStream** to establish a connection with the first DataNode with the first block of a file.
5. Data is read in the form of streams wherein client invokes **'read()'** method repeatedly. This process of **read()** operation continues till it reaches the end of block.
6. Once the end of a block is reached, DFSInputStream closes the connection and moves on to locate the next DataNode for the next block
7. Once a client has done with the reading, it calls **a close()** method.

WRITE OPERATION



WRITE OPERATION

1. A client initiates write operation by calling 'create()' method of DistributedFileSystem object which creates a new file - Step no. 1 in the above diagram.
2. DistributedFileSystem object connects to the NameNode using RPC call and initiates new file creation. However, this file creates operation does not associate any blocks with the file. It is the responsibility of NameNode to verify that the file (which is being created) does not exist already and a client has correct permissions to create a new file. If a file already exists or client does not have sufficient permission to create a new file, then **IOException** is thrown to the client. Otherwise, the operation succeeds and a new record for the file is created by the NameNode.
3. Once a new record in NameNode is created, an object of type FSDataOutputStream is returned to the client. A client uses it to write data into the HDFS. Data write method is invoked (step 3 in the diagram).
4. FSDataOutputStream contains DFSOutputStream object which looks after communication with DataNodes and NameNode. While the client continues writing data, **DFSOutputStream** continues creating packets with this data. These packets are enqueued into a queue which is called as **DataQueue**.
5. There is one more component called **DataStreamer** which consumes this **DataQueue**. DataStreamer also asks NameNode for allocation of new blocks thereby picking desirable DataNodes to be used for replication.

Replication

1. Now, the process of replication starts by creating a pipeline using DataNodes. In our case, we have chosen a replication level of 3 and hence there are 3 DataNodes in the pipeline.
2. The DataStreamer pours packets into the first DataNode in the pipeline.
3. Every DataNode in a pipeline stores packet received by it and forwards the same to the second DataNode in a pipeline.
4. Another queue, 'Ack Queue' is maintained by DFSOutputStream to store packets which are waiting for acknowledgment from DataNodes.
5. Once acknowledgment for a packet in the queue is received from all DataNodes in the pipeline, it is removed from the 'Ack Queue'. In the event of any DataNode failure, packets from this queue are used to reinitiate the operation.
6. After a client is done with the writing data, it calls a close() method (Step 9 in the diagram) Call to close(), results into flushing remaining data packets to the pipeline followed by waiting for acknowledgment.
7. Once a final acknowledgment is received, NameNode is contacted to tell it that the file write operation is complete.

MapReduce

- **MAPREDUCE** is a software framework and programming model used for processing huge amounts of data.
- **MapReduce** program work in two phases, namely, Map and Reduce.
  - Map tasks deal with splitting and mapping of data
  - Reduce tasks shuffle and reduce the data.

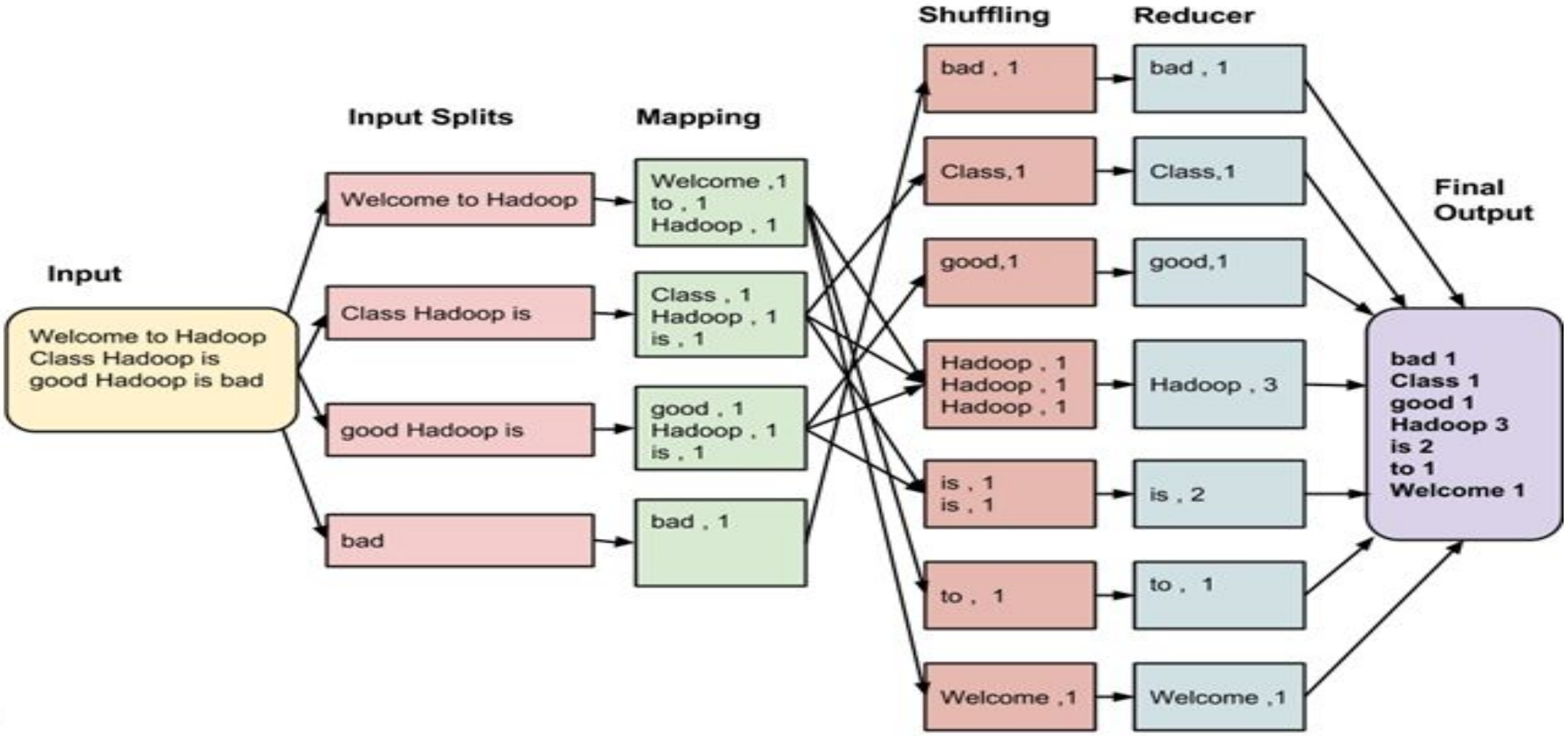
MapReduce

- Hadoop is capable of running MapReduce programs written in various languages: Java, Ruby, Python, and C++.
- MapReduce programs are parallel in nature, thus are very useful for performing large-scale data analysis using multiple machines in the cluster.
- The input to each phase is **key-value** pairs.
- In addition, every programmer needs to specify two functions: **map function** and **reduce function**.

How MapReduce Works?

- Four phases of execution :
- Phase 1: Splitting
- Phase 2: Mapping
- Phase 3: Shuffling
- Phase 4: Reducing

Working of Map Reduce



Final output of the MapReduce task

bad	1
Class	1
good	1
Hadoop	3
is	2
to	1
Welcome	1

The data goes through the following phases:

- **Input Splits:**
- An input to a MapReduce job is divided into fixed-size pieces called **input splits .**Input split is a chunk of the input that is consumed by a single map
- **Mapping:**
- This is the very first phase in the execution of map-reduce program. In this phase data in each split is passed to a mapping function to produce output values.
- Example, a job of mapping phase is to count a number of occurrences of each word from input splits and prepare a list in the form of <word, frequency>

The data goes through the following phases:

- **Shuffling**
- This phase consumes the output of Mapping phase. Its task is to consolidate the relevant records from Mapping phase output.
- In the example, the same words are clubbed together along with their respective frequency.
- **Reducing**
- In this phase, output values from the Shuffling phase are aggregated. This phase combines values from Shuffling phase and returns a single output value. In short, this phase summarizes the complete dataset.
- In the example, this phase aggregates the values from Shuffling phase i.e., calculates total occurrences of each word.

How MapReduce Organizes Work?

Hadoop divides the job into tasks. There are two types of tasks:

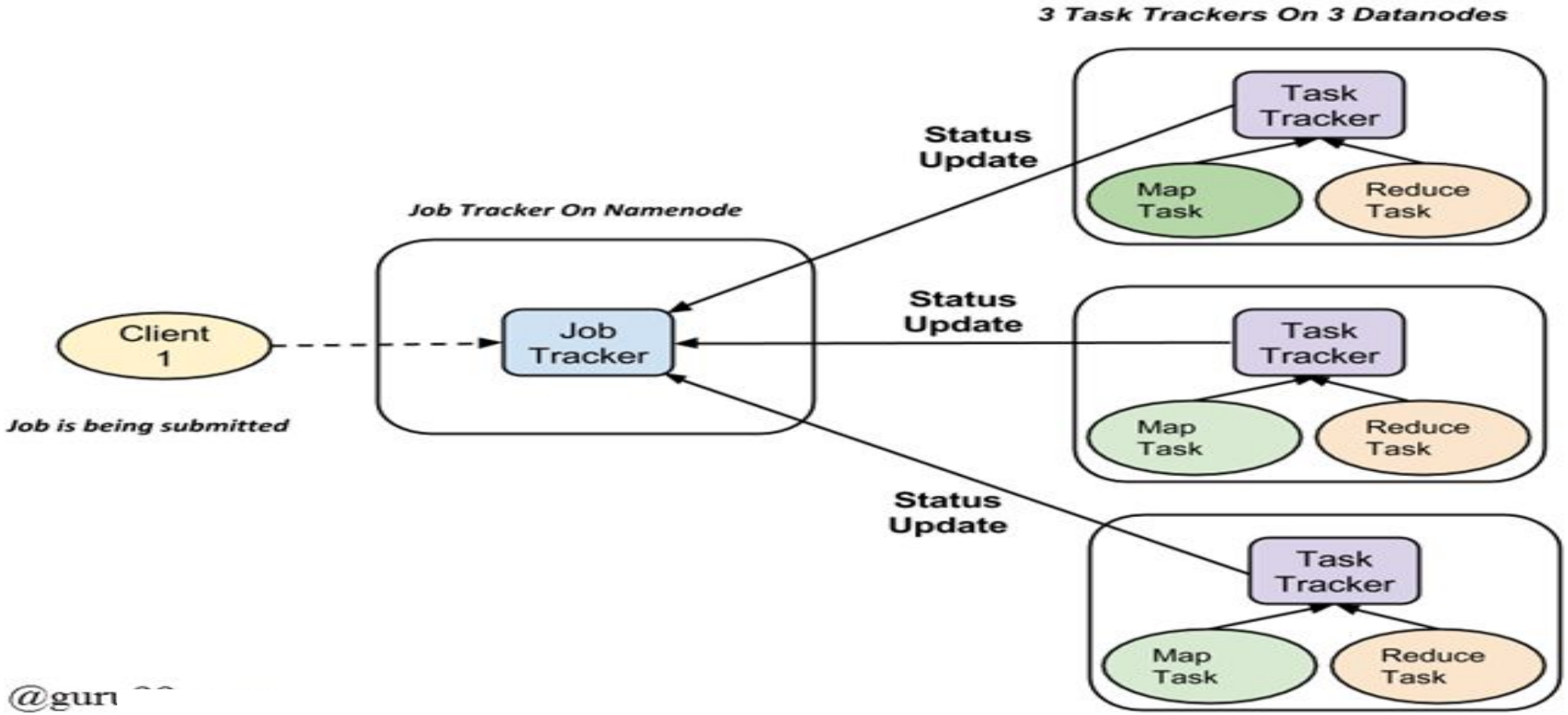
1. **Map tasks** (Splits & Mapping)
2. **Reduce tasks** (Shuffling, Reducing)

- The complete execution process (execution of Map and Reduce tasks, both) is controlled by two types of entities called

1. **Jobtracker** : Acts like a **master** (responsible for complete execution of submitted job)
2. **Multiple Task Trackers**: Acts like **slaves,** each of them performing the job

- For every job submitted for execution in the system, there is one **Jobtracker** that resides on **Namenode** and there are **multiple tasktrackers** which reside on **Datanode**.

How MapReduce Organizes Work?



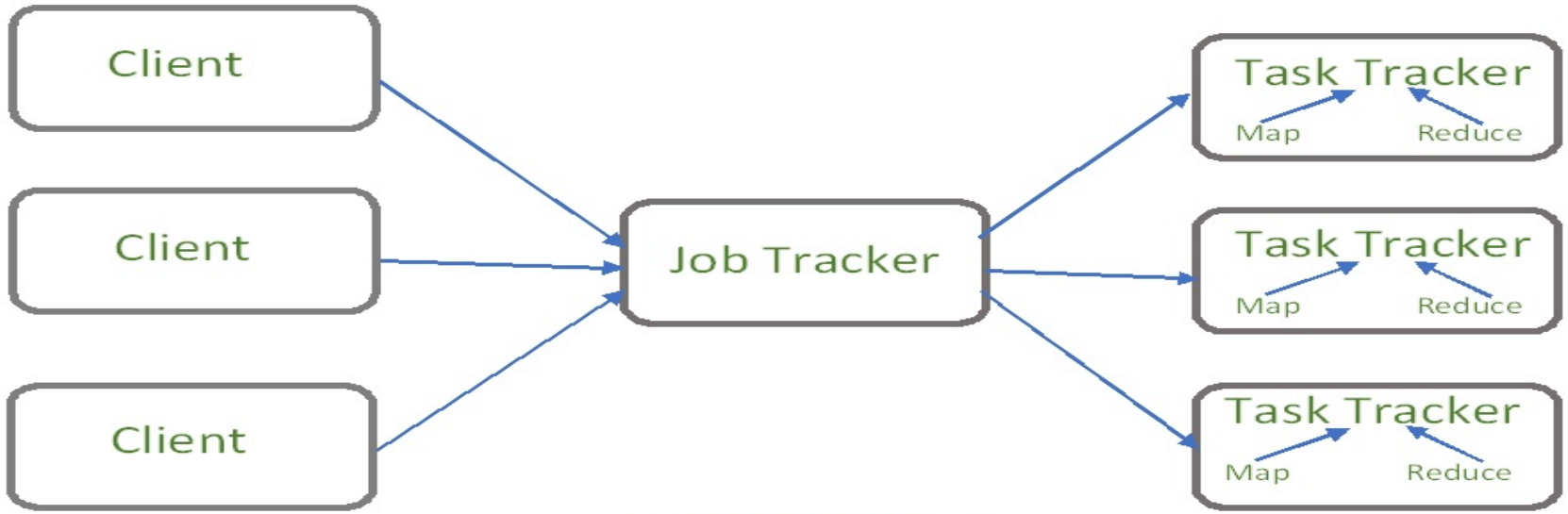
How MapReduce Organizes Work?

- A job is divided into multiple tasks which are then run onto multiple data nodes in a cluster.
- It is the responsibility of job tracker to coordinate the activity by scheduling tasks to run on different data nodes.
- Execution of individual task is then to look after by task tracker, which resides on every data node executing part of the job.
- Task tracker's responsibility is to send the progress report to the job tracker.
- In addition, task tracker periodically sends **'heartbeat'** signal to the Jobtracker so as to notify him of the current state of the system.
- Thus job tracker keeps track of the overall progress of each job. In the event of task failure, the job tracker can reschedule it on a different task tracker.

Managing Resources And Applications With Hadoop Yarn

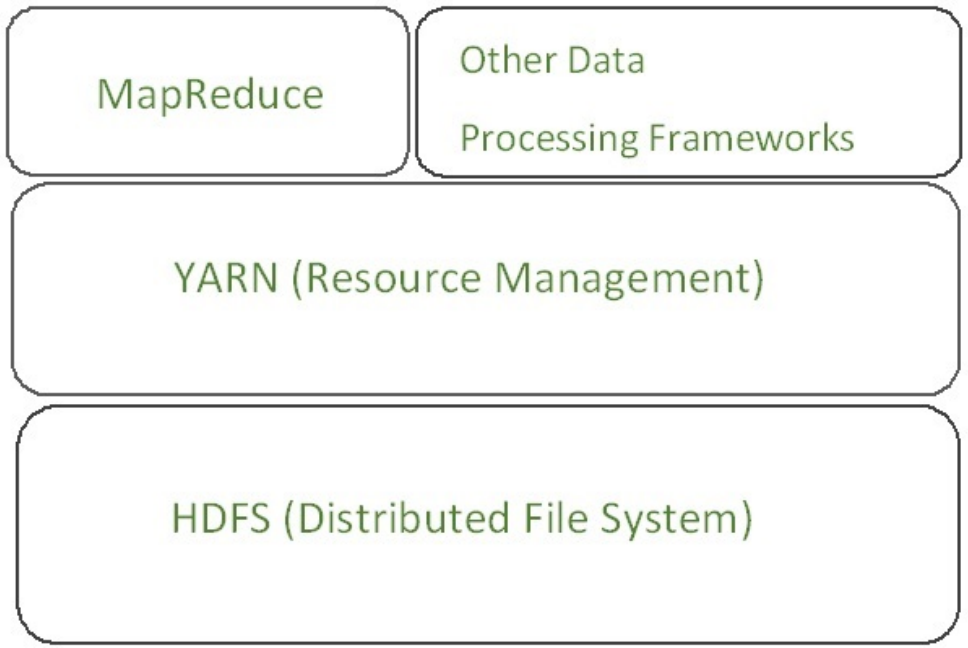
- YARN stands for "***Yet Another Resource Negotiator***".
- It was introduced in Hadoop 2.0 to remove the bottleneck on Job Tracker which was present in Hadoop 1.0.
- YARN was described as a "*Redesigned Resource Manager*" at the time of its launching, but it has now evolved to be known as large-scale distributed operating system used for Big Data processing.

Hadoop 1.0



Hadoop 1.0 architecture

Hadoop 2.0



Hadoop 2.0

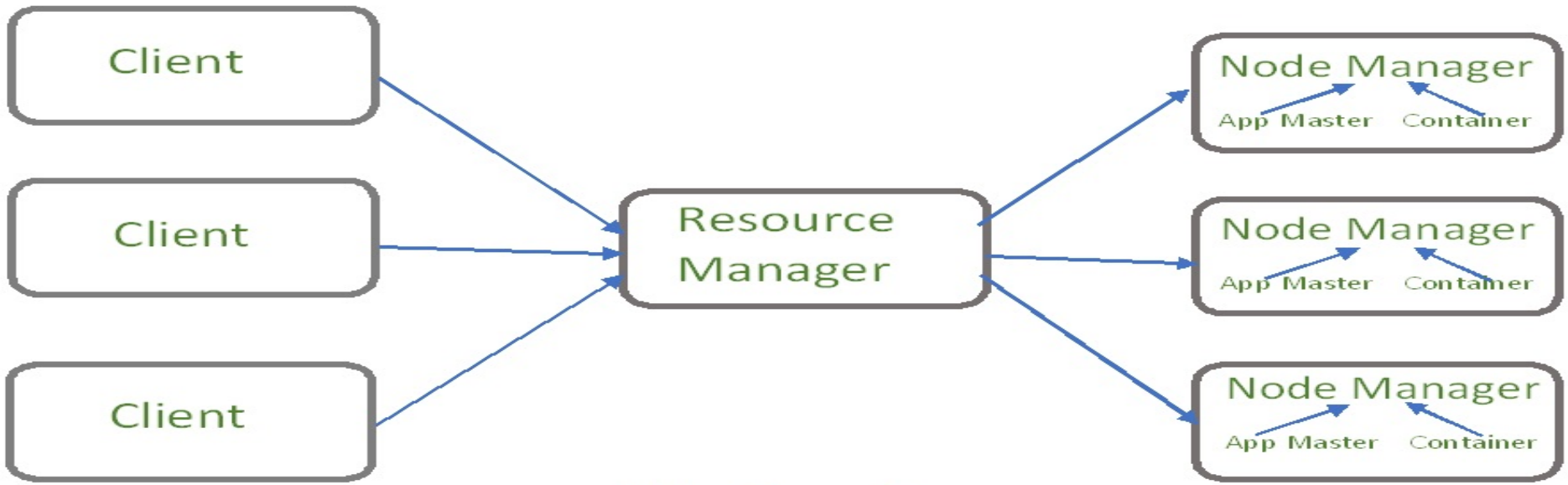
YARN

- YARN also allows different data processing engines like graph processing, interactive processing, stream processing as well as batch processing to run and process data stored in HDFS (Hadoop Distributed File System) thus making the system much more efficient.
- Through its various components, it can dynamically allocate various resources and schedule the application processing. For large volume data processing, it is quite necessary to manage the available resources properly so that every application can leverage them.

YARN Features

- **Scalability:** The scheduler in Resource manager of YARN architecture allows Hadoop to extend and manage thousands of nodes and clusters.
- **Compatibility:** YARN supports the existing map-reduce applications without disruptions thus making it compatible with Hadoop 1.0 as well.
- **Cluster Utilization :**Since YARN supports Dynamic utilization of cluster in Hadoop, which enables optimized Cluster Utilization.
- **Multi-tenancy:** It allows multiple engine access thus giving organizations a benefit of multi-tenancy.

Hadoop YARN



Hadoop Yarn architecture

Components of Yarn Architecture

- **Client**
- **Resource Manager**
  - **Scheduler**
  - **Application manager**
- **Node Manager**
- **Application Master**
- **Container**

- **Client:** It submits map-reduce jobs.

Components of Yarn Architecture

- **Resource Manager:** It is the master daemon of YARN and is responsible for resource assignment and management among all the applications. Whenever it receives a processing request, it forwards it to the corresponding node manager and allocates resources for the completion of the request accordingly. It has two major components:
  - **Scheduler:** It performs scheduling based on the allocated application and available resources. It is a pure scheduler, means it does not perform other tasks such as monitoring or tracking and does not guarantee a restart if a task fails. The YARN scheduler supports plugins such as Capacity Scheduler and Fair Scheduler to partition the cluster resources.
  - **Application manager:** It is responsible for accepting the application and negotiating the first container from the resource manager. It also restarts the Application Manager container if a task fails.

Components of Yarn Architecture

- **Node Manager:** It take care of individual node on Hadoop cluster and manages application and workflow and that particular node. Its primary job is to keep-up with the Node Manager. It monitors resource usage, performs log management and also kills a container based on directions from the resource manager. It is also responsible for creating the container process and start it on the request of Application master.

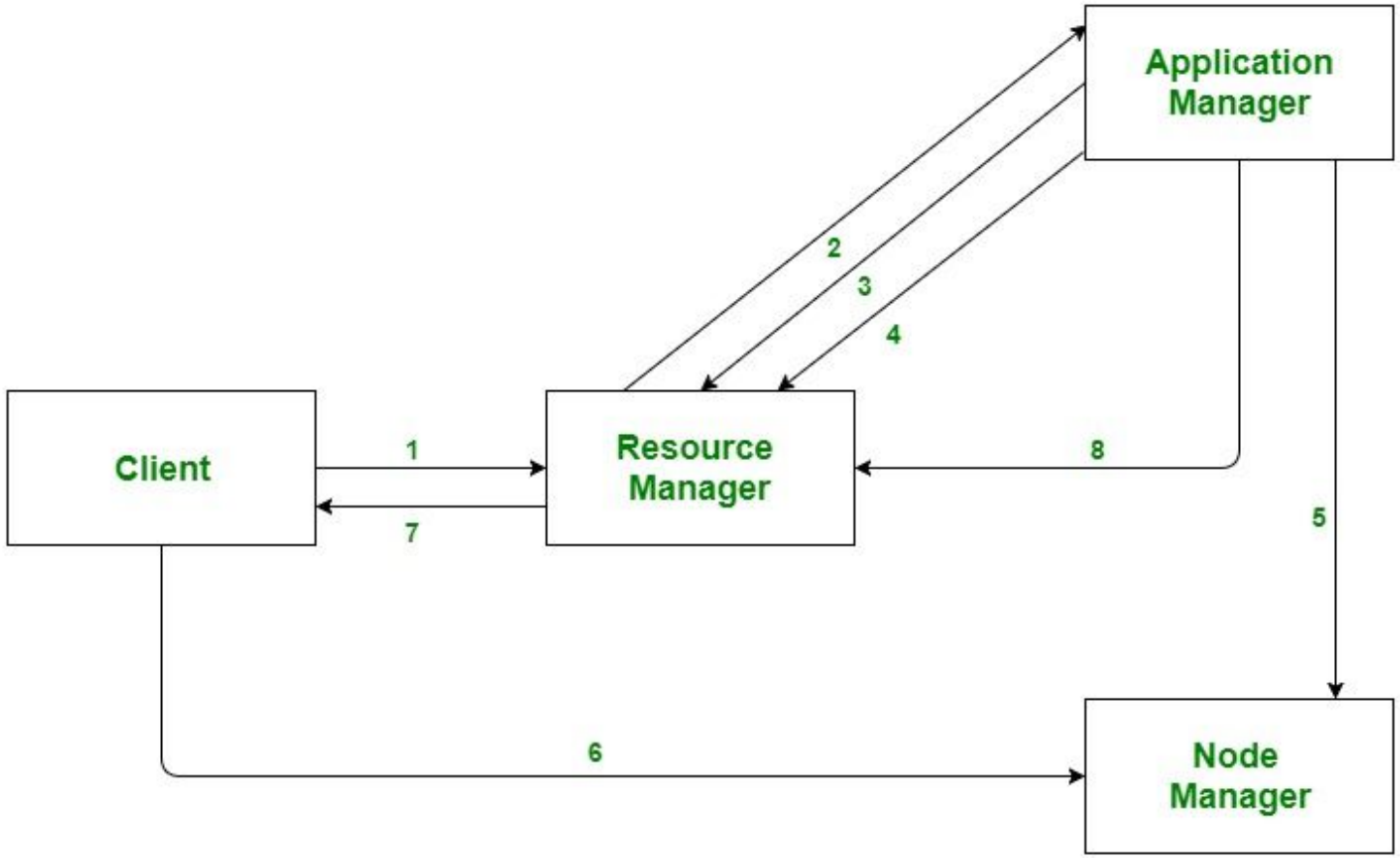
Components of Yarn Architecture

- **Application Master:** An application is a single job submitted to a framework. The application manager is responsible for negotiating resources with the resource manager, tracking the status and monitoring progress of a single application. The application master requests the container from the node manager by sending a Container Launch Context(CLC) which includes everything an application needs to run. Once the application is started, it sends the health report to the resource manager from time-to-time.

Components of Yarn Architecture

- **Container:** It is a collection of physical resources such as RAM, CPU cores and disk on a single node. The containers are invoked by Container Launch Context(CLC) which is a record that contains information such as environment variables, security tokens, dependencies etc.

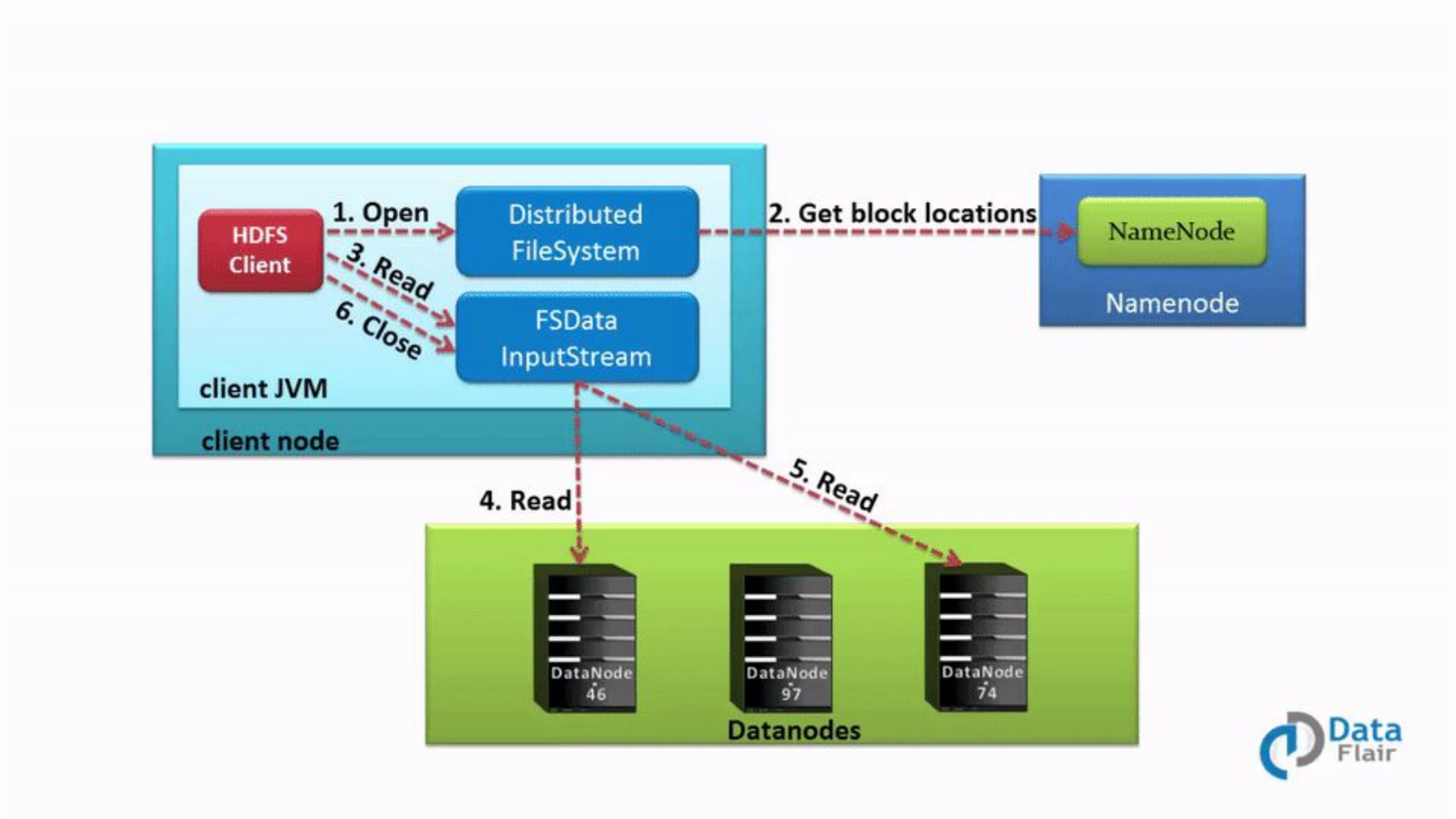
Application workflow in Hadoop YARN:



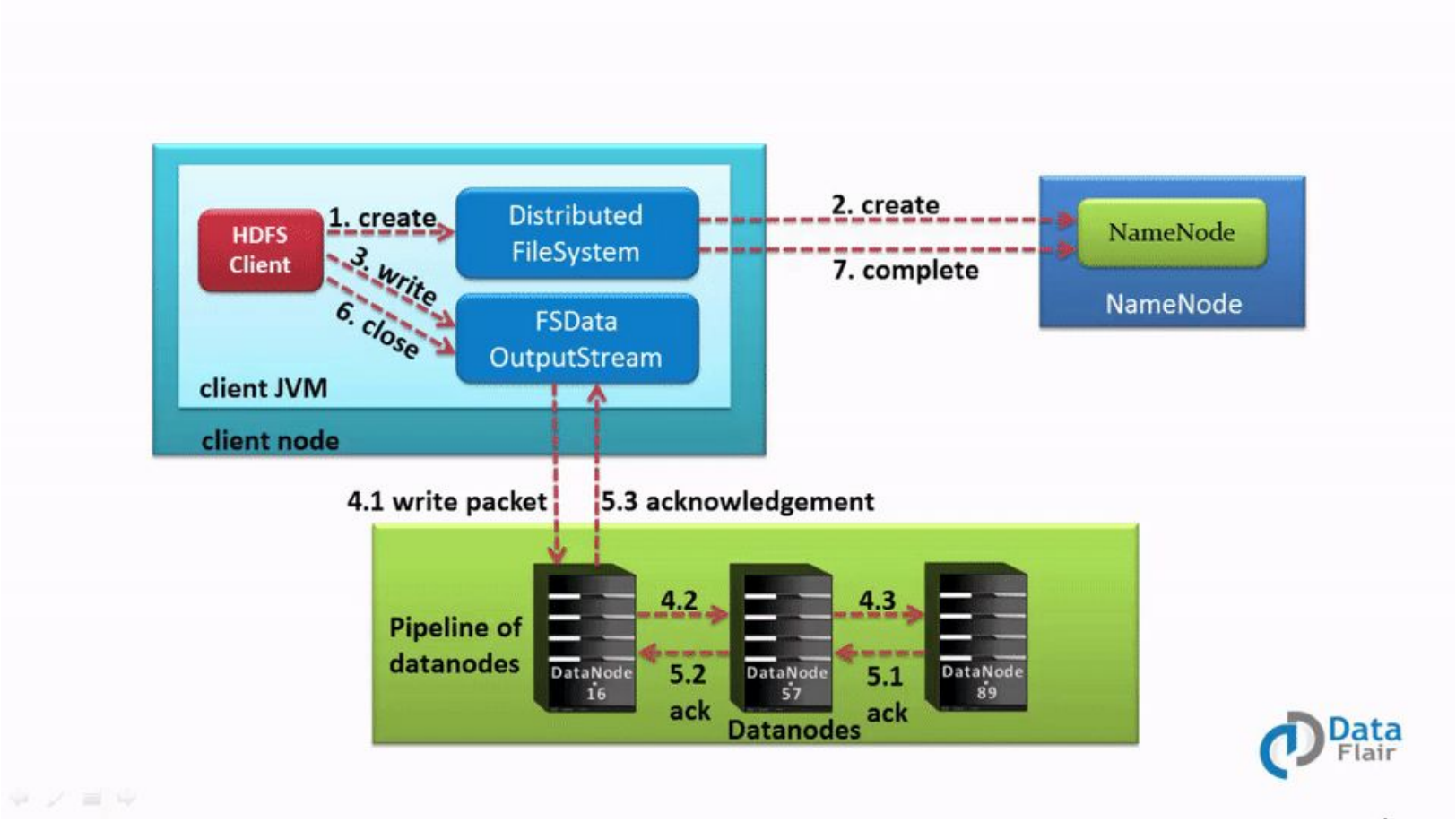
Application workflow

1. Client submits an application
2. The Resource Manager allocates a container to start the Application Manager
3. The Application Manager registers itself with the Resource Manager
4. The Application Manager negotiates containers from the Resource Manager
5. The Application Manager notifies the Node Manager to launch containers
6. Application code is executed in the container
7. Client contacts Resource Manager/Application Manager to monitor application's status
8. Once the processing is complete, the Application Manager un-registers with the Resource Manager

HDFS_READ FILE



HDFS_WRITE FILE



References

- https://www.simplilearn.com/tutorials/hadoop-tutorial/hdfs
- HDFS-READ
  - https://www.youtube.com/watch?v=Ax7EhEsVVzE
- HDFS-WRITE
  - https://www.youtube.com/watch?v=0QJKx4A4L7Y

- https://www.youtube.com/watch?v=nWqdePeOh9M
- https://techvidvan.com/tutorials/how-hadoop-works-internally/
- https://www.guru99.com/learn-hdfs-a-beginners-guide.html
- https://www.guru99.com/introduction-to-mapreduce.html
- https://data-flair.training/blogs/hdfs-data-write-operation/

THANK YOU