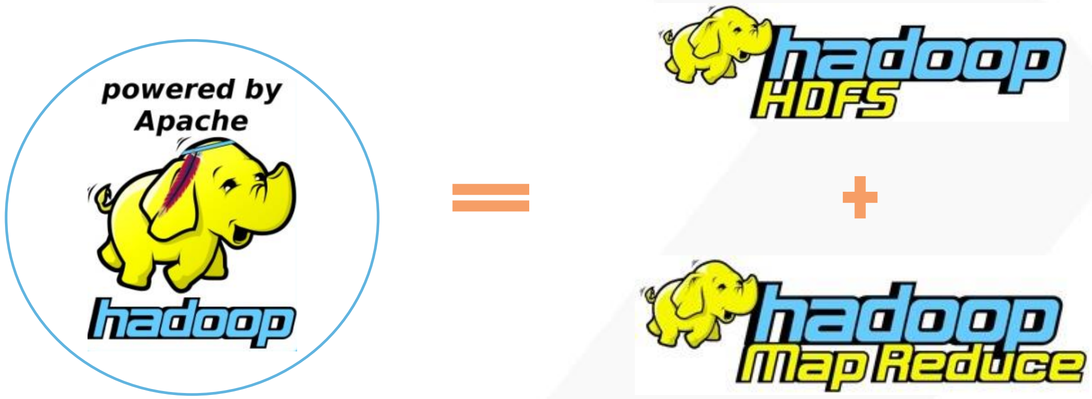
Hadoop Core Components

Disclaimer: This material is protected under copyright act AnalytixLabs ©, 2011-2016. Unauthorized use and/ or duplication of this material or any part of this material including data, in any form without explicit and written permission from AnalytixLabs is strictly prohibited. Any violation of this copyright will attract legal actions

Hadoop Core Components



Hadoop Core Components

HADOOP has two main components

- ✓ **HDFS-Hadoop Distributed File System(Storage)**
  - ✓ Distributed across nodes
  - ✓ Natively redundant
  - ✓ Name node tracks locations
- ✓ **Map Reduce (Processing)**
  - ✓ Splits a task across processors
  - ✓ "near" the data & assemble results
  - ✓ Self-healing, Hand bandwidth
  - ✓ Clustered storage
  - ✓ Job tracker manages the task tracker

HDFS(Storage) - Demons

Name Node (FSIMAGE, EDIT LOG)

Data Node

Secondary Name Node

(FSIMAGE)

Map Reduce (Processing) - Demons

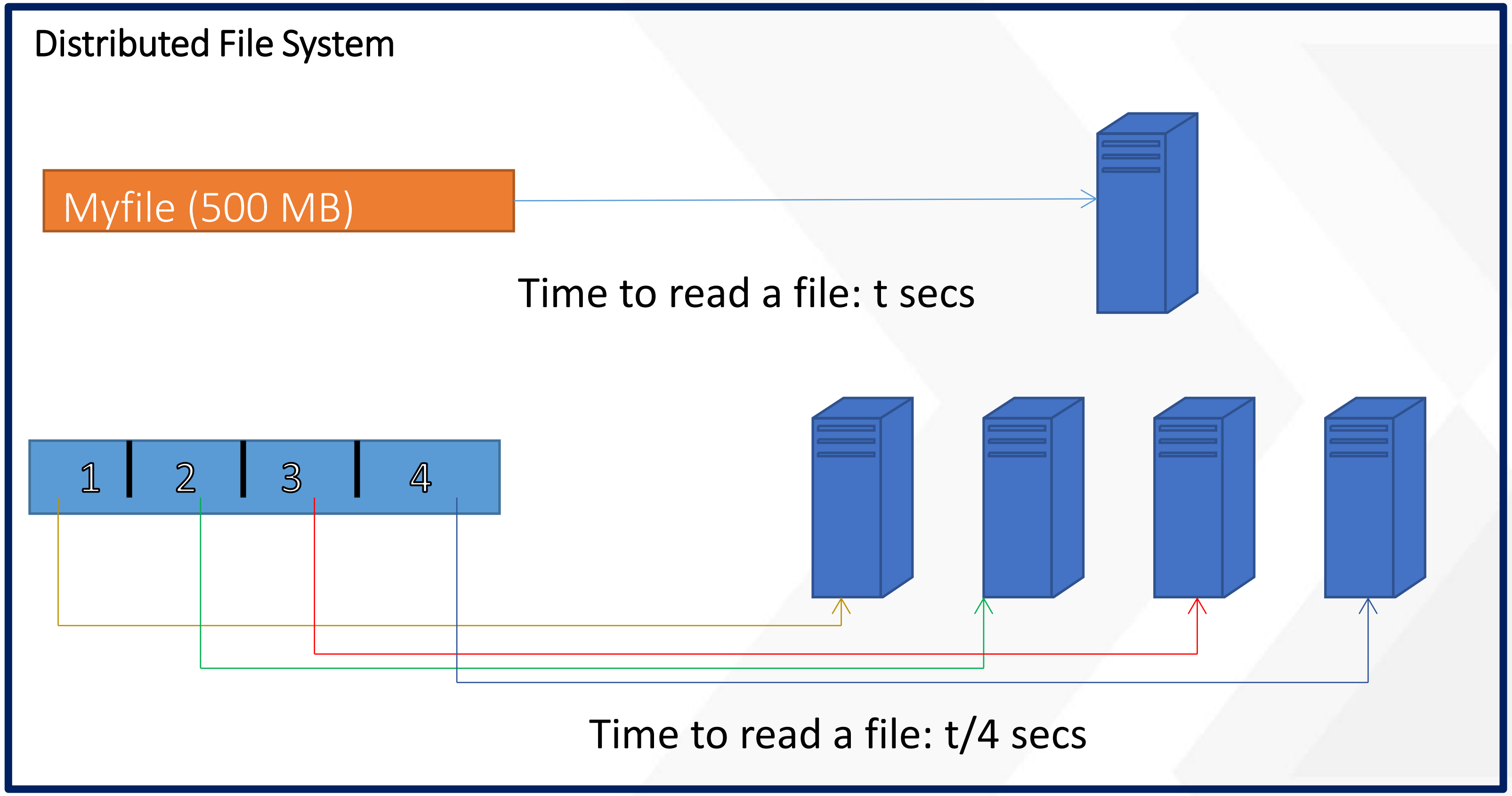
Resource Manager (job tracker)

Node Manager (task tracker)

Resource Manager, Node manager are the daemons of YARN(popularly

Hadoop - HDFS

Distributed File System



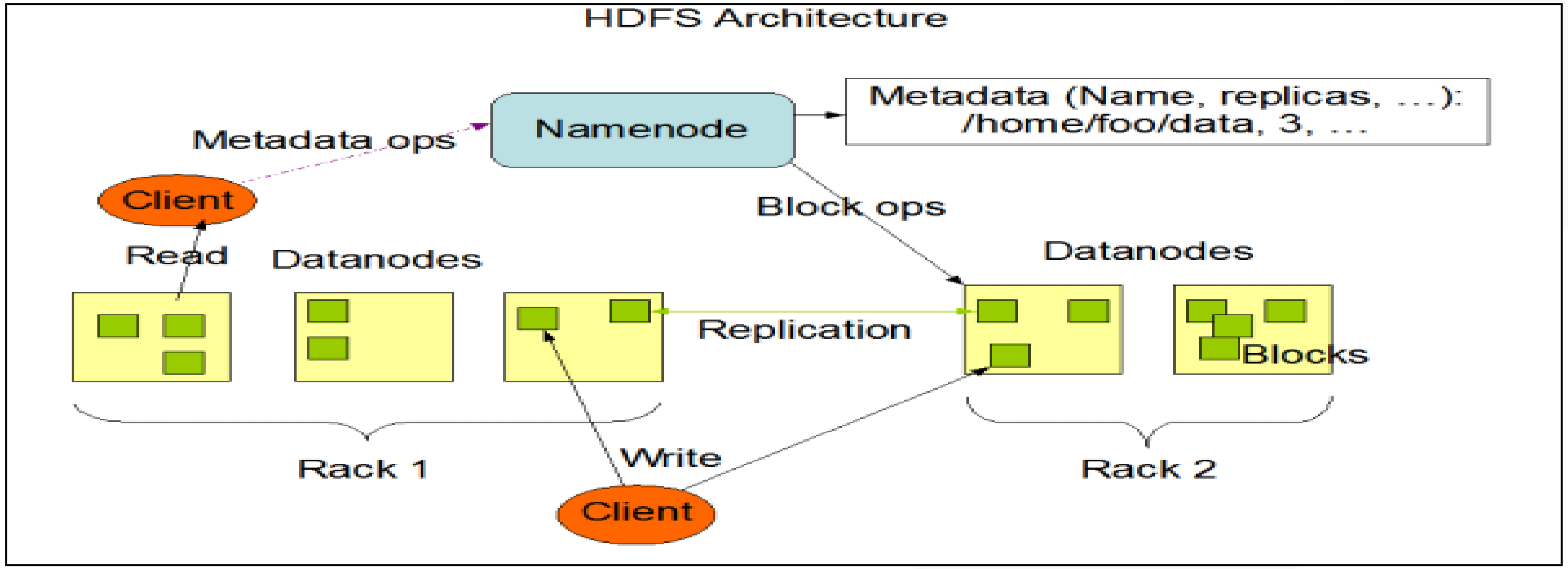
Time to read a file: t secs

Time to read a file: t/4 secs

Hadoop Distributed File System (HDFS)

What is HDFS?

- ✓ *HDFS is derived from concepts of **Google file system**.*
- ✓ *An important characteristic of Hadoop is **partitioning of data** and execution of application **computations in parallel**, close to their data.*
- ✓ *On HDFS, data files are **replicated** as sequences of blocks in the cluster*
- ✓ *A Hadoop cluster scales computation capacity and storage capacity by simply adding commodity servers*
- ✓ HDFS is the default distributed file system used by the Hadoop MapReduce computation



Key HDFS Features

- Runs with commodity hardware
- Master slave paradigm
- Distributed
- Scalable (scale out Architecture)
- Secured
- Reliable (Tunable Replication)
- Fault Tolerance
- High-throughput
- Small number of large files vs. large number of small files
- Write Once Read Many Times
- Sequential/Streaming Access

Regular File System vs. HDFS

Regular File System

- Each block of data is small in size; approximately 4K bytes.
- All the blocks of a file is stored on the local disk of a computer system.
- Storage Capacity of system should be large enough to store the file.
- Large data access suffers from disk I/O problems; mainly because of multiple seek operation.

HDFS

- Each block of data is very large in size; 64MB/128MB by default.
- Different blocks of a file are stored on different nodes.
- A large file is divided into blocks, even if storage capacity of a system is not too large, save file into cluster.
- Reads huge data sequentially after a single seek

HDFS Architecture

- HDFS is a java-based file system and is the place where all the data in the Hadoop cluster resides.
- HDFS has been restructured in the second version of Hadoop to support multiple types of data processing units.

As of now we have two versions in Hadoop;Hadoop1.x and Hadoop 2.x.

Hadoop1.x:

This is the very first version of Hadoop built to handle BigData. HDFS and MapReduce are the two steps involved in processing the data in Hadoop1.x architecture. It process data in Batches, hence the name ‘Batch processing’.

Hadoop2.x

Hadoop2.x is the enhanced version of Hadoop1.x. This version has introduced to overcome the problems and shortcomings of Hadoop1.x. A new feature called YARN(Yet Another Resource negotiator) has been introduced in Hadoop2.x. Here HDFS is used along with YARN to process the data. With YARN, Hadoop is able to process different forms of data. Along with Hadoop, we can use many other tools to process the data.

HDFS Architecture

In Hadoop1.x the components of HDFS:

- ✓NameNode
- ✓DataNode
- ✓Job Tracker
- ✓Task Tracker
- ✓Secondary NameNode

✓NameNode, DataNode, Secondary NameNode are the daemons of HDFS

✓Job tracker, Task tracker are the daemons of Map reduce

In Hadoop2.x the components of HDFS:

- ✓NameNode
- ✓DataNode
- ✓Resource Manager
- ✓Node Manager
- ✓Secondary NameNode

✓NameNode, DataNode, Secondary NameNode are the daemons of HDFS

✓Resource Manager, Node manager are the daemons of YARN(popularly known as the Map reduce 2.0)

Along with Hadoop, we can use many other tools to process the data. They are as follows:

HDFS+MapReduce- Batch processing
HDFS+Storm – Real-time processing
HDFS+Spark – Near real-time processing

HDFS Daemons

NameNode:
NameNode holds the meta data for the HDFS like Namespace information, block information etc. When in use, all this information is stored in main memory. But these information also stored in disk for persistence storage.

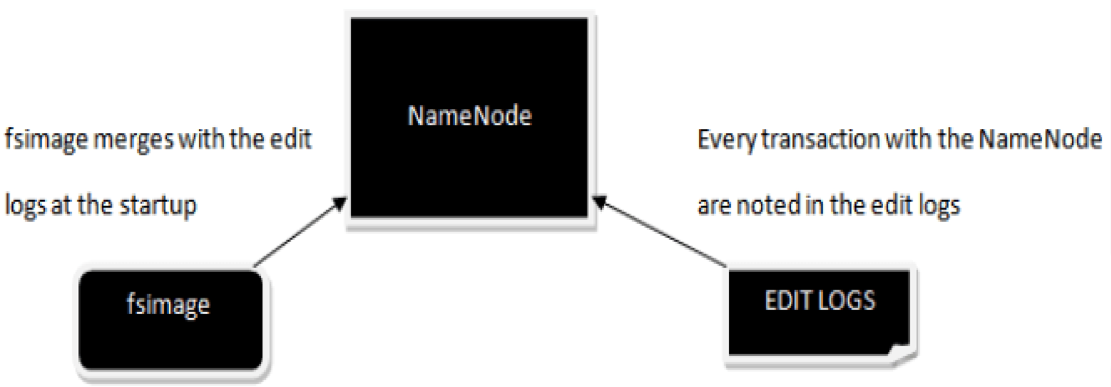
There are two different files associated with the NameNode

Fsimage – It is the snapshot of the filesystem

Edit logs – After the start of NameNode it maintains the every transaction that happened with NameNode.

The NameNode maintains the namespace tree and the mapping of blocks to DataNodes.
The Client communicates with the NameNode and provides data to the HDFS through it. The HDFS then stores data as blocks inside DataNodes.

By default, the block size in a hadoop cluster is 128MB. NameNode maintains the meta data information of all the blocks present inside the hadoop cluster like permissions, modification and access times, namespace and disk space quotas . This is the reason NameNode is also called as the Master node.


NameNode
fsimage merges with the edit logs at the startup
Every transaction with the NameNode are noted in the edit logs
fsimage
EDIT LOGS

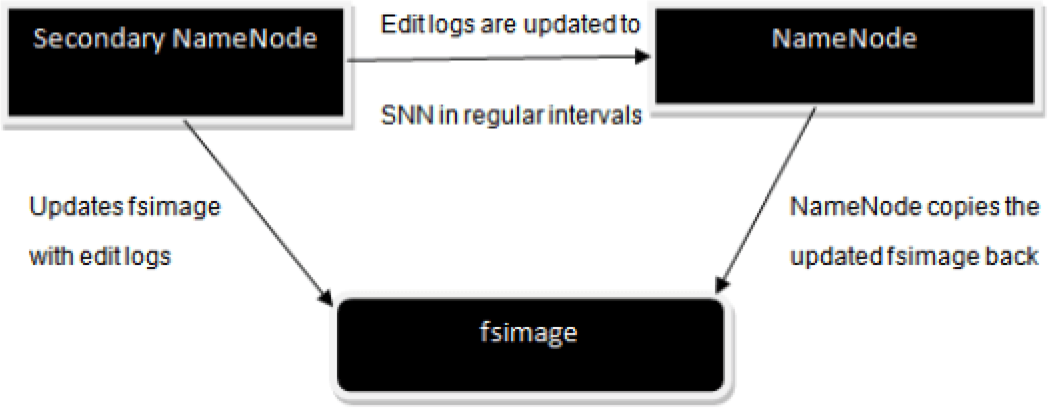
HDFS Demons

Secondary Name Node:

It asks the NameNode for its edit logs in regular intervals and copies them into the fsimage.

After updating the fsimage, NameNode copy back that fsimage. NameNode uses this fsimage when it starts, this eventually will reduce the startup time.

The main theme of secondary NameNode is to maintain a checkpoint in HDFS. When a failure occurs SNN won’t become NameNode it just helps NameNode in bringing back its data. SNN is also called as checkpoint node in hadoop’s architecture.



HDFS Demons

DataNode:

This is the place where actual data is stored in the hadoop cluster in a distributed manner.
Every DataNode will have a block scanner and it directly reports to the NameNode about the blocks which it is handling.

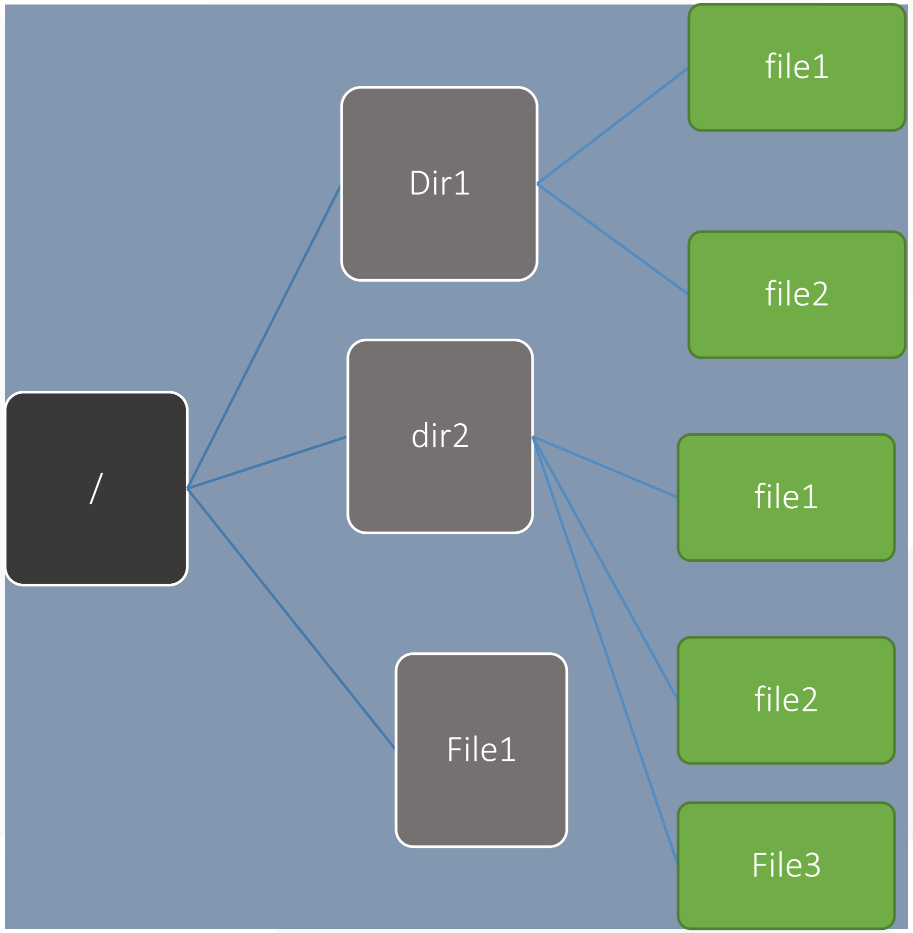
The DataNodes communicate with the NameNode by sending Heartbeats for every 3seconds and if the NameNode does not receive any Heartbeat for 10 minutes, then it treats the DataNode as a dead node and re-replicates the blocks.

During startup each DataNode connects to the NameNode and performs a handshake. The purpose of the handshake is to verify the namespace ID and the software version of the DataNode. If either does not match that of the NameNode, the DataNode automatically shuts down.

The namespace ID is assigned to the file system instance when it is formatted. The namespace ID is persistently stored on all nodes of the cluster.

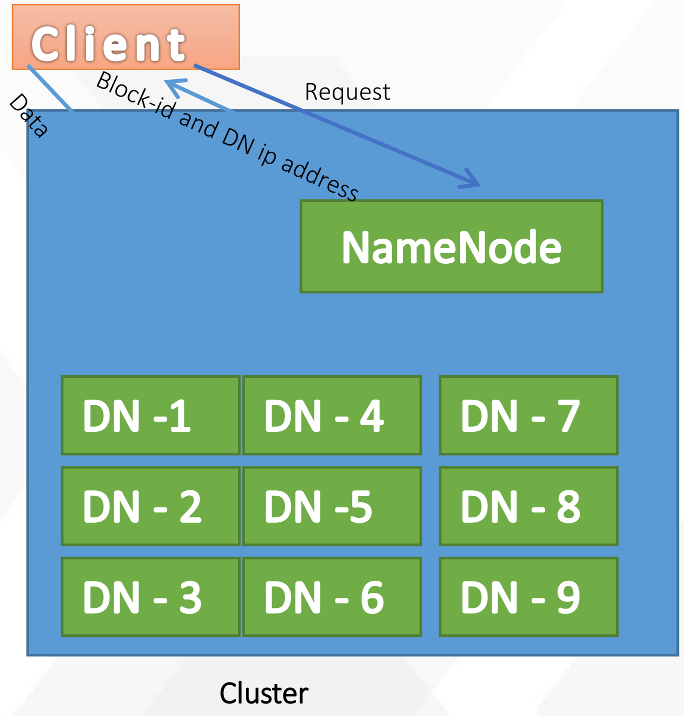
HDFS Name Space

- Hierarchical file system
- Directories and Files
- Sits on the native linux file system
- Uses same commands as linux file system
- Create, move, copy, rename or delete
- NameNode maintains the HDFS Name space.
- Any changes in HDFS Name space is recorded in the metadata stored in RAM of NameNode.
- Single Name Space for the cluster.



Client Interaction with cluster

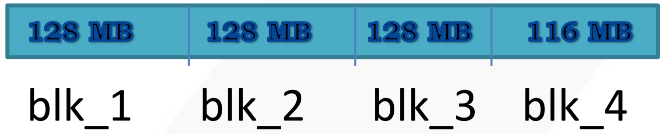
- Client requests NameNode to write a file in cluster.
- NameNode checks the authenticity of client.
- No replacement strategy, so NameNode checks if file already exists in cluster. Sends error if file exists.
- Finds the file size and divide the file into chunks based upon block-size called blocks.
- NameNode gives unique block-id to each block and decides on which DataNode to store which block.
- Sends block-id and DataNode ip address to client one by one.
- Client writes data directly to DataNode.



Data Blocks

- Default block size is 64 MB/ 128 MB, but configurable.
- Each file is divided into one or more blocks depending upon file size.
- All the blocks except the last one contains data equal to block-size.
- Each block has a unique block-id. (Integer)
- Blocks are stored on DataNodes.
- Blocks on the DataNodes are stored as complete and independent files.

Myfile (500 MB) divided into 4 blocks
(1,2,3 and 4)

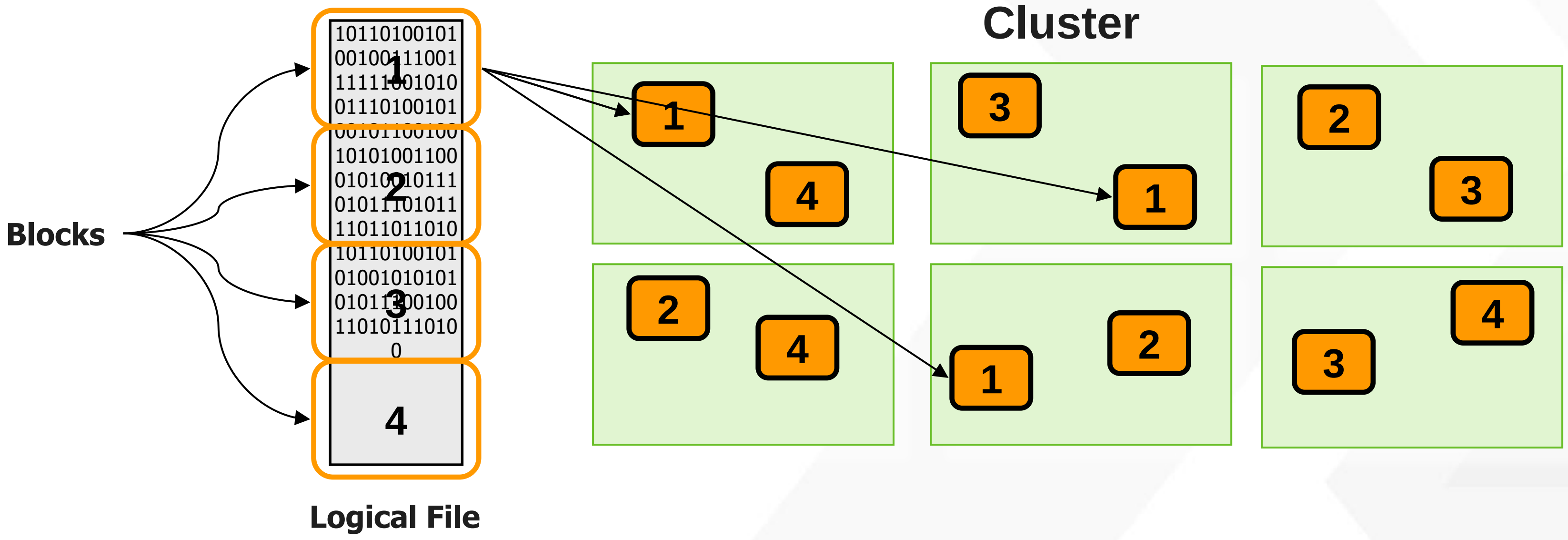


128 MB	128 MB	128 MB	116 MB
blk_1	blk_2	blk_3	blk_4

HDFS: Reliability

Fault Tolerant Distributed Storage

- Divide files into big blocks and distribute 3 copies **randomly** across the cluster
- Processing Data Locality
  - Not Just storage but computation



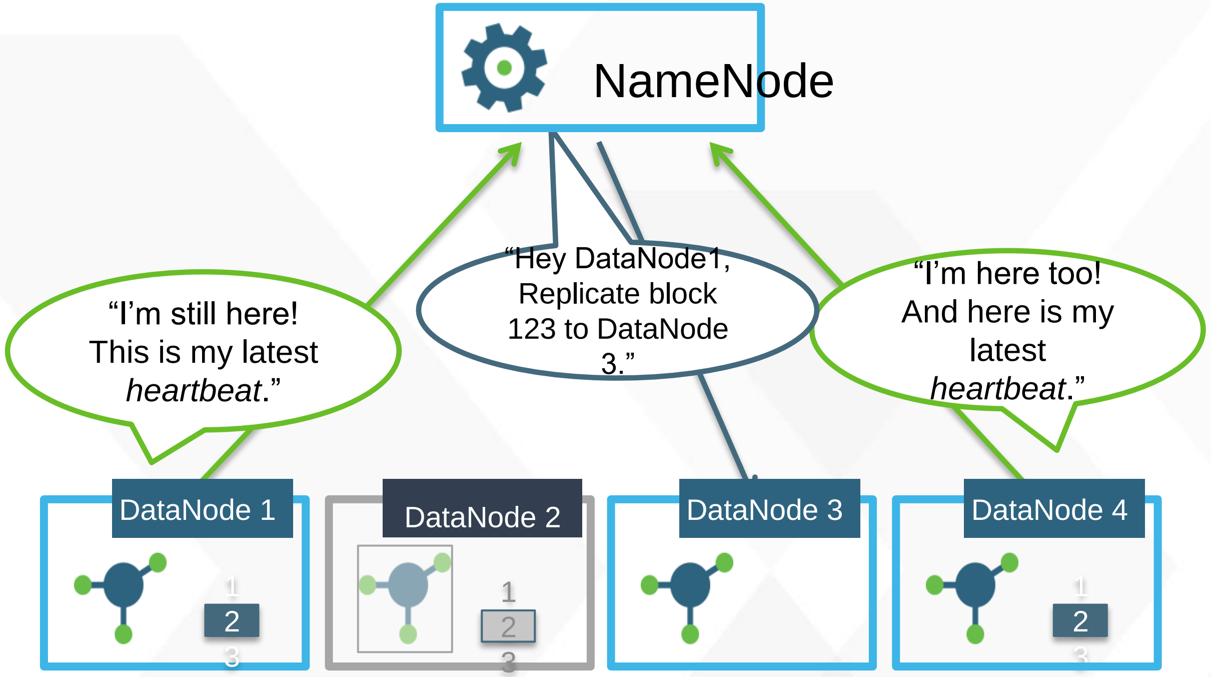
HDFS: Fault Tolerance

- Hadoop Cluster is built-up of commodity hardware.

- Frequent failure is norm rather than exception.

- All the DataNodes send heartbeat to NameNode after every minute to indicate that they are alive.

- After every 10th heartbeat, each DataNode sends block report indicating which blocks are available.


NameNode
"I'm still here! This is my latest *heartbeat*."
"Hey DataNode1, Replicate block 123 to DataNode 3."
"I'm here too! And here is my latest *heartbeat*."
DataNode 1
1
2
3
DataNode 2
1
2
3
DataNode 3
DataNode 4
1
2
3

- NameNode does not get heartbeat from a DataNode.

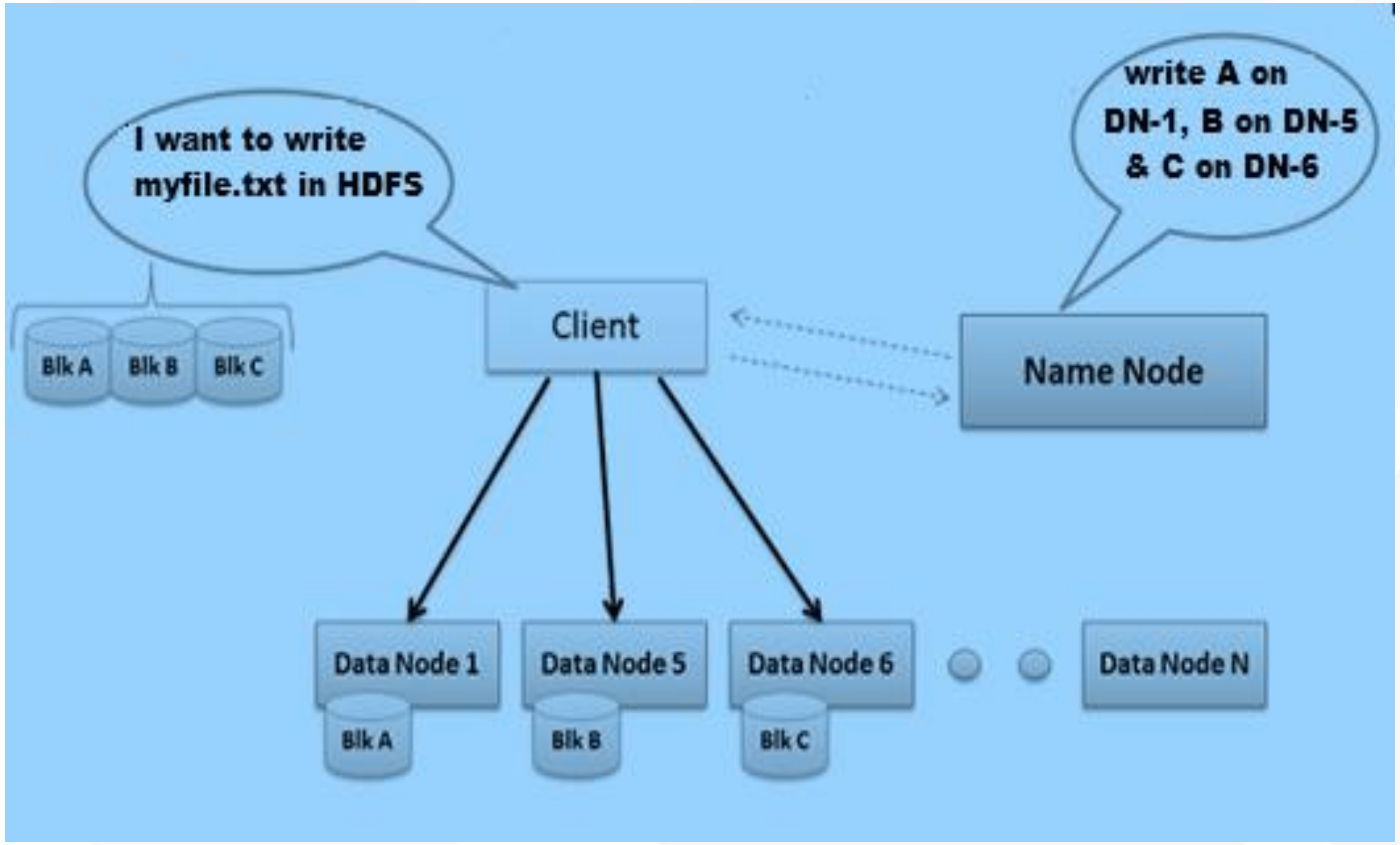
- Waits to get the block report.

- If NameNode does not get block report, assumes that DataNode is not live.

- Replicates the blocks of that DataNode to some other DataNodes.

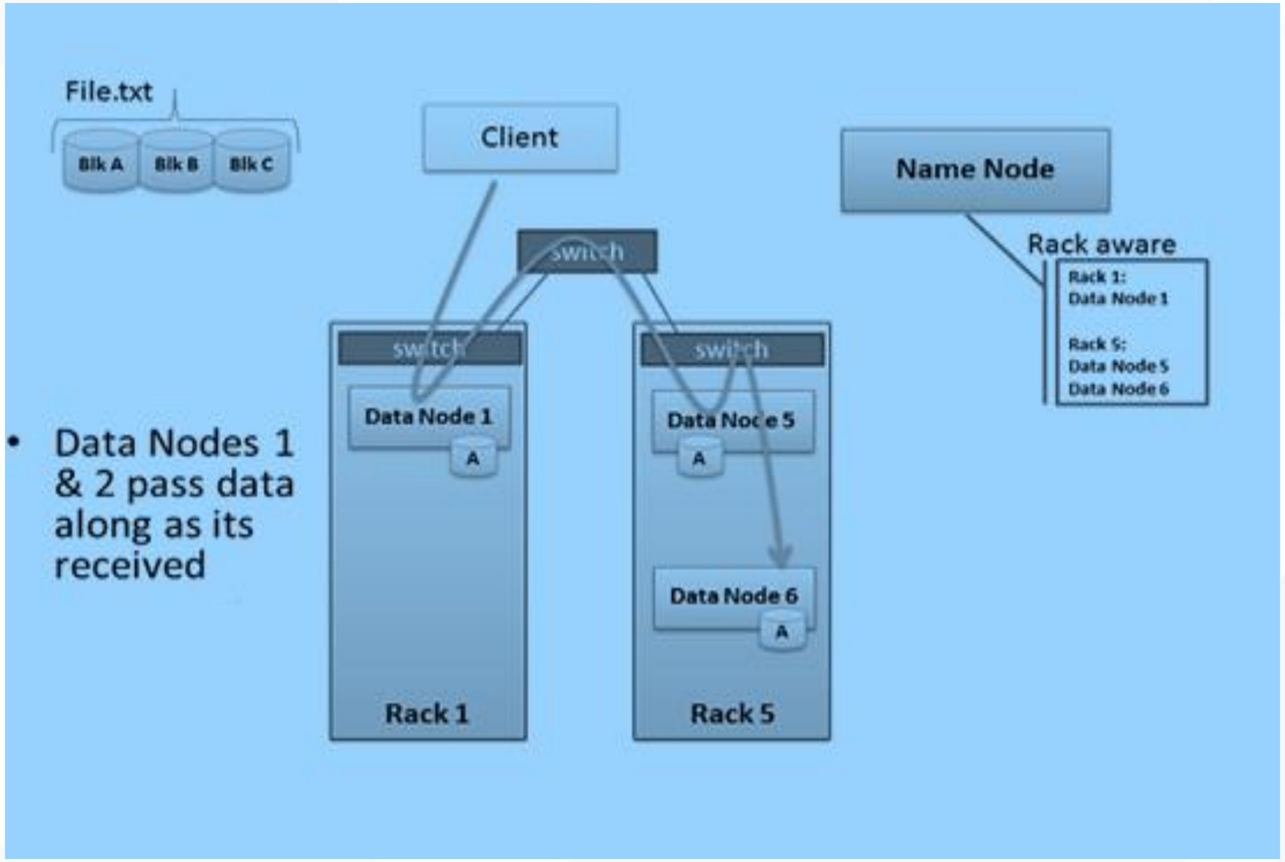
Writing File to HDFS

- Client sends a request to NameNode.
- Namenode sends Block-Id alongwith IP address of DataNode on which to write block.
- Client writes block data directly to DataNode.
- Client repeats the process for next block.



Pipelined Write to HDFS

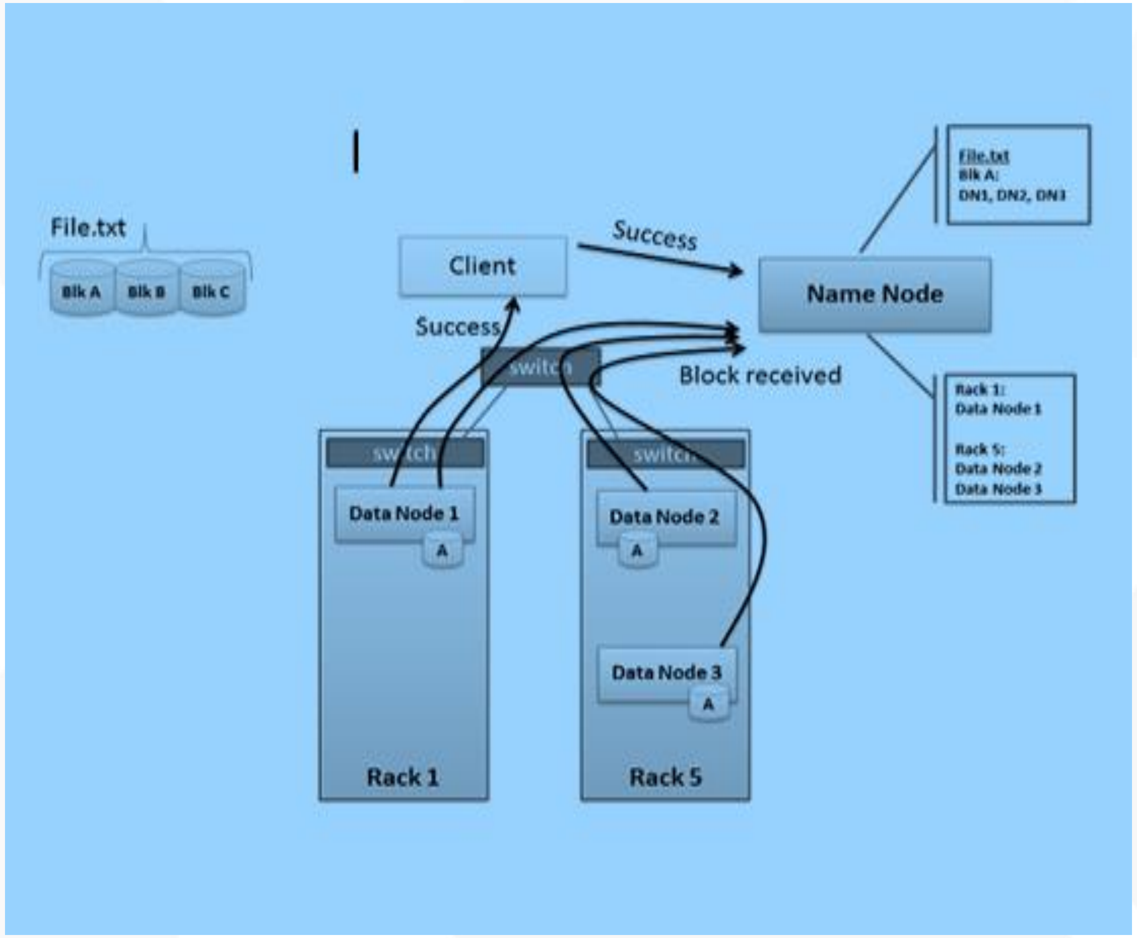
- Client sends a request to first DataNode to create a pipeline with other DataNodes.
- Once pipeline is created, client writes data to first DataNode.
- First DataNode writes and forwards data to next one and so on.
- All the DataNodes send acknowledgement to NameNode.
- Client repeats the process for next block.



- Data Nodes 1 & 2 pass data along as its received

Meta Data

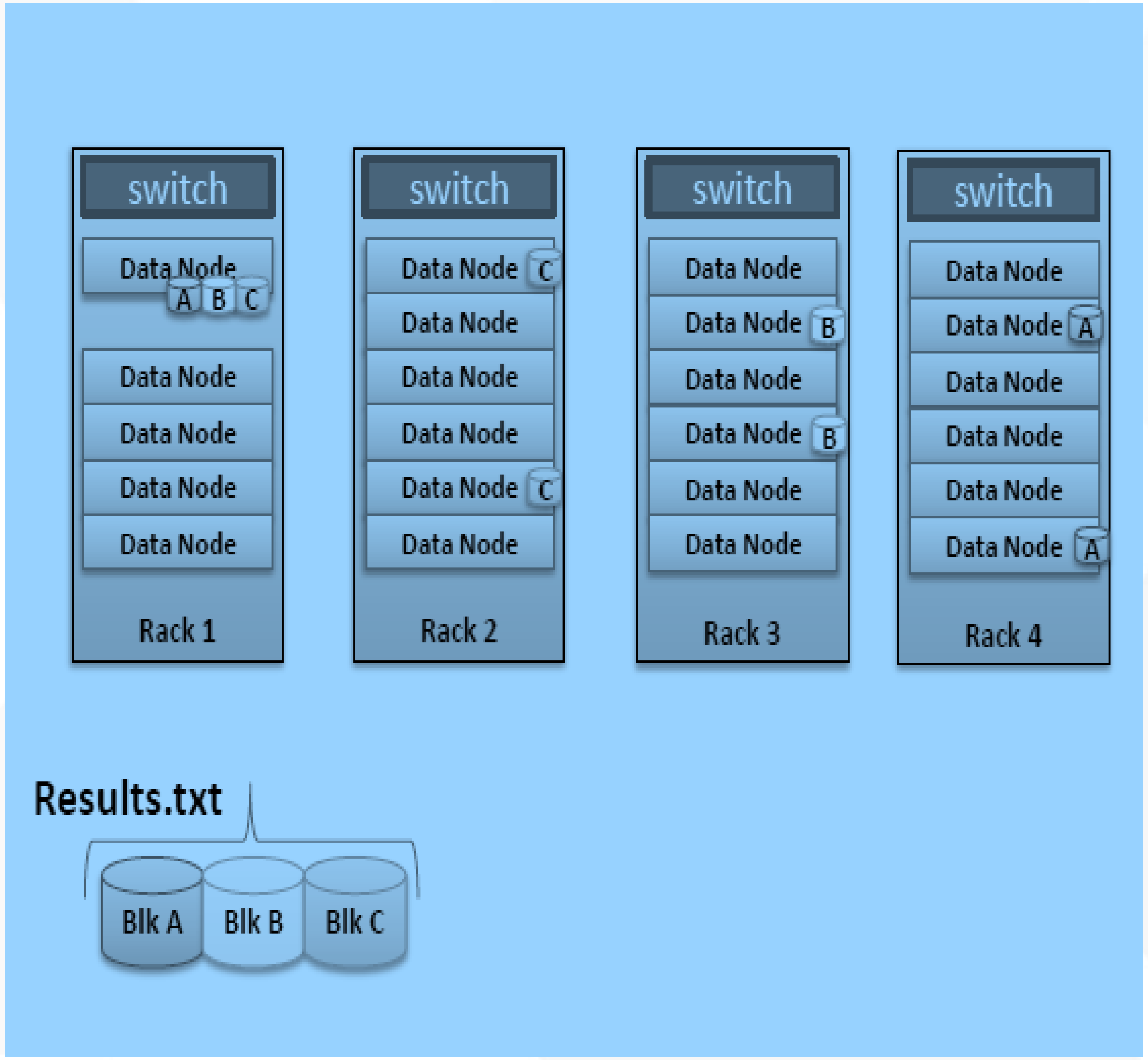
- NameNode maintains information about each file and block in metadata.
- Metadata is kept in the RAM of NameNode.
- In the metadata, information about files such as filename, owner, group, creationTime, accessTime, size of file, block size and replication factor along with block-Ids belonging to that file.
- Location of the blocks are not kept in metadata.



Data Replication Topology

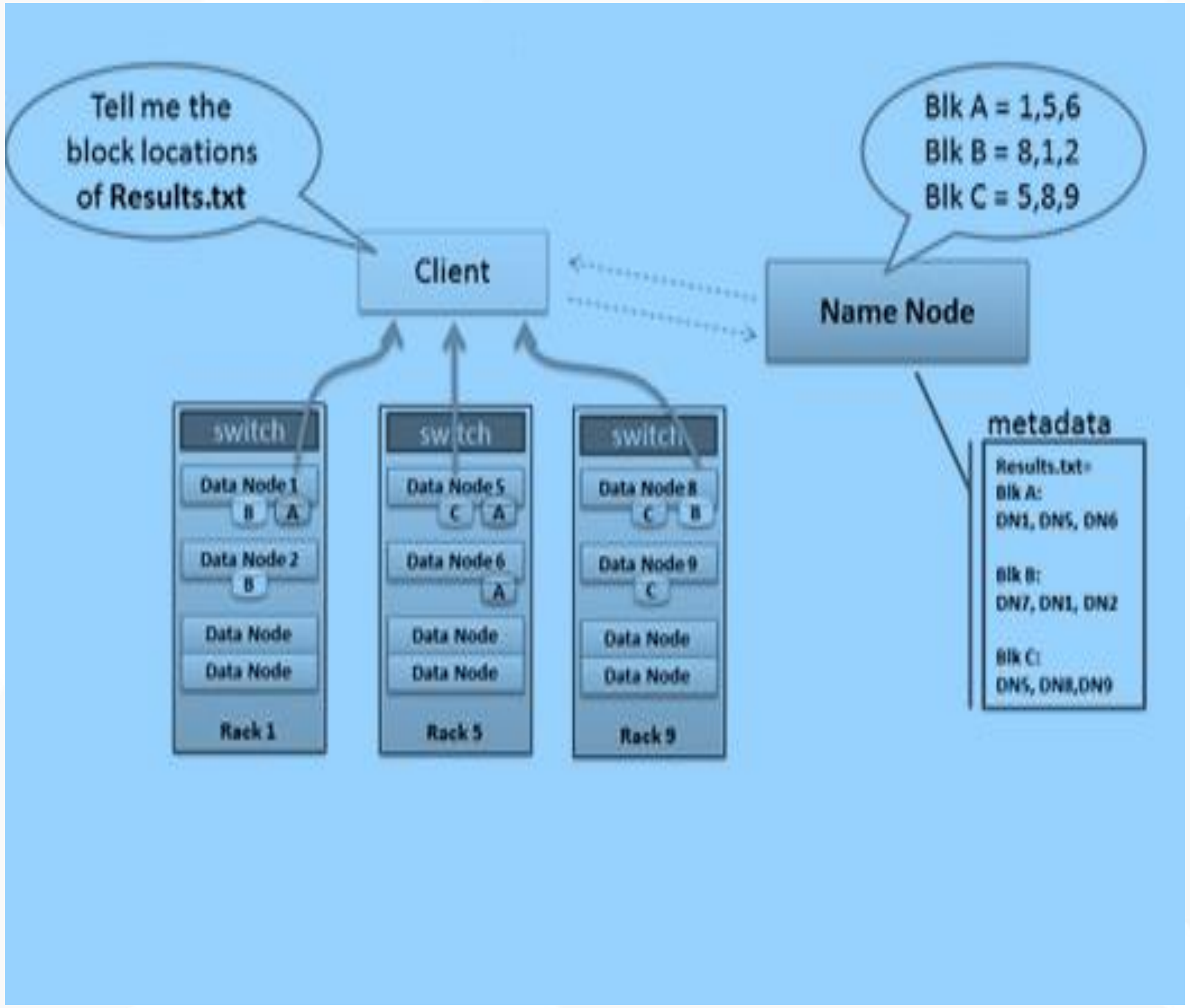
By default, Hadoop places block replicas as:

- 1st replica is placed on one node of any rack.
- 2nd replica is placed on some node of different rack.
- 3rd replica is placed on different machine on the same rack as of 2nd replica.

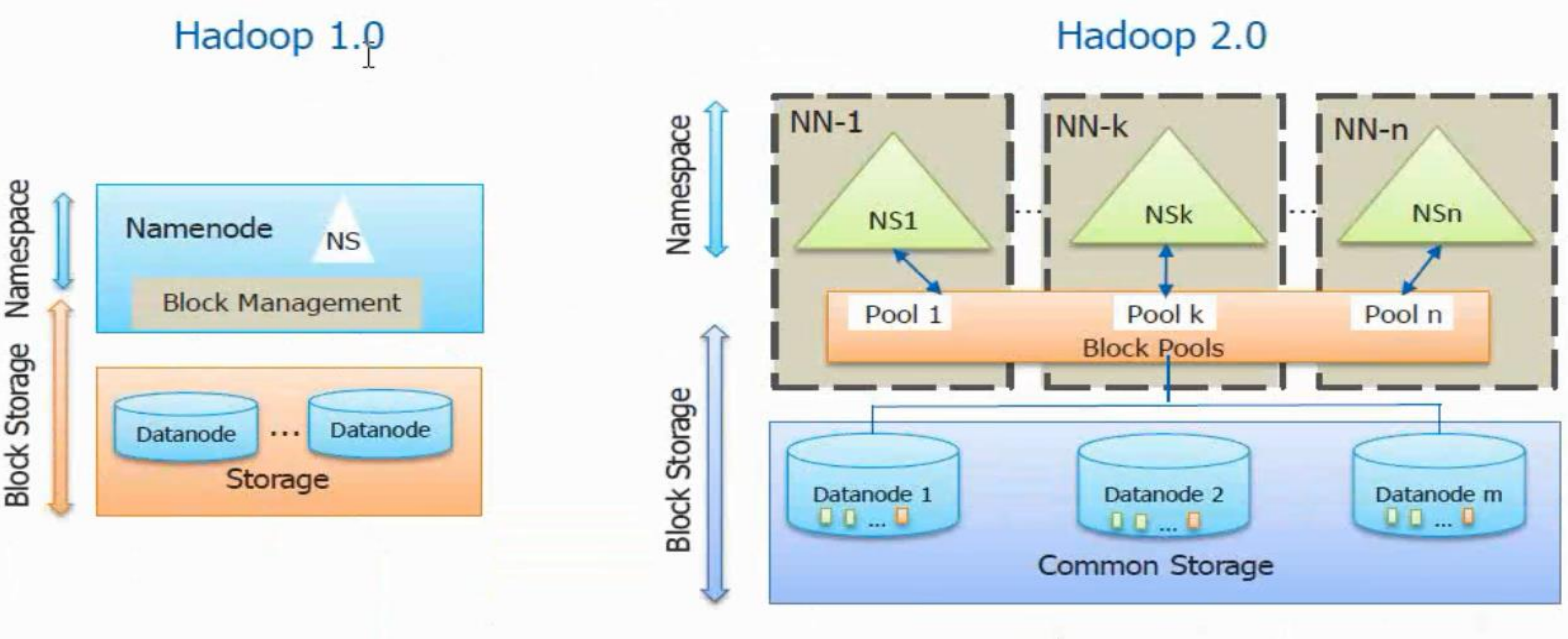


Data Reading from HDFS

- Client sends the request to NameNode.
- NameNode sends the list of IP addresses of all DataNodes on which block exists.
- Client interacts with the first DataNode in the list on which block exists.
- Client reads the data directly from the DataNode.

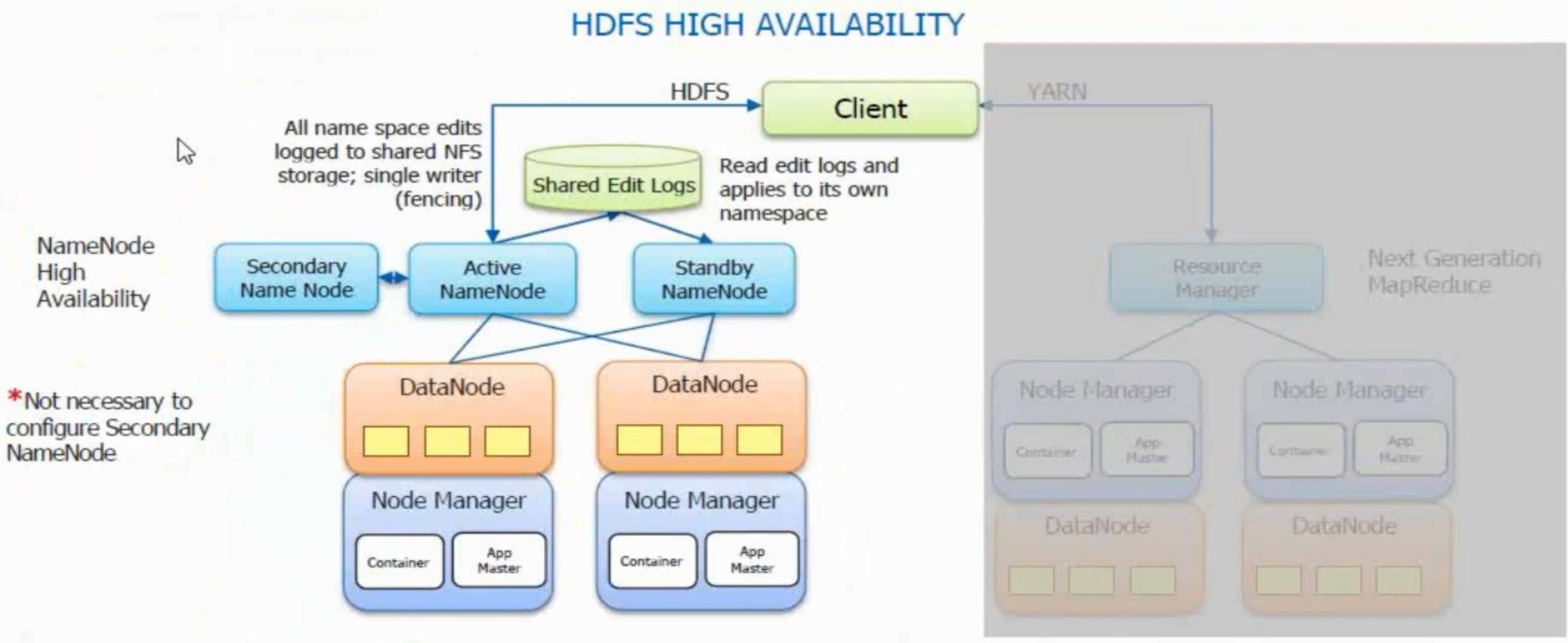


Hadoop 2.x Architecture - Federation



http://hadoop.apache.org/docs/stable2/hadoop-project-dist/hadoop-hdfs/Federation.html

HDFS High Availability feature with shared edit log



HDFS HIGH AVAILABILITY

All name space edits logged to shared NFS storage; single writer (fencing)

Read edit logs and applies to its own namespace

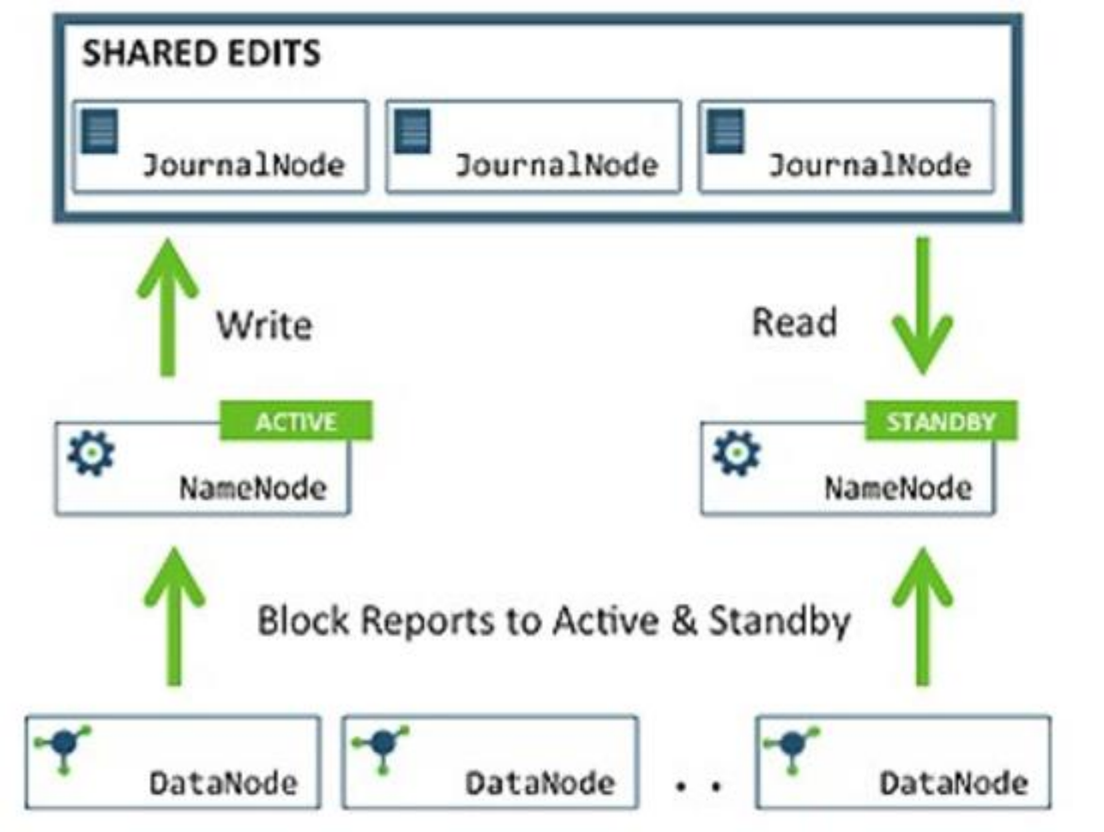
NameNode High Availability

*Not necessary to configure Secondary NameNode

http://hadoop.apache.org/docs/stable2/hadoop-yarn/hadoop-yarn-site/HDFSHighAvailabilityWithNFS.html

HDFS High Availability using Quoram Journal Manager

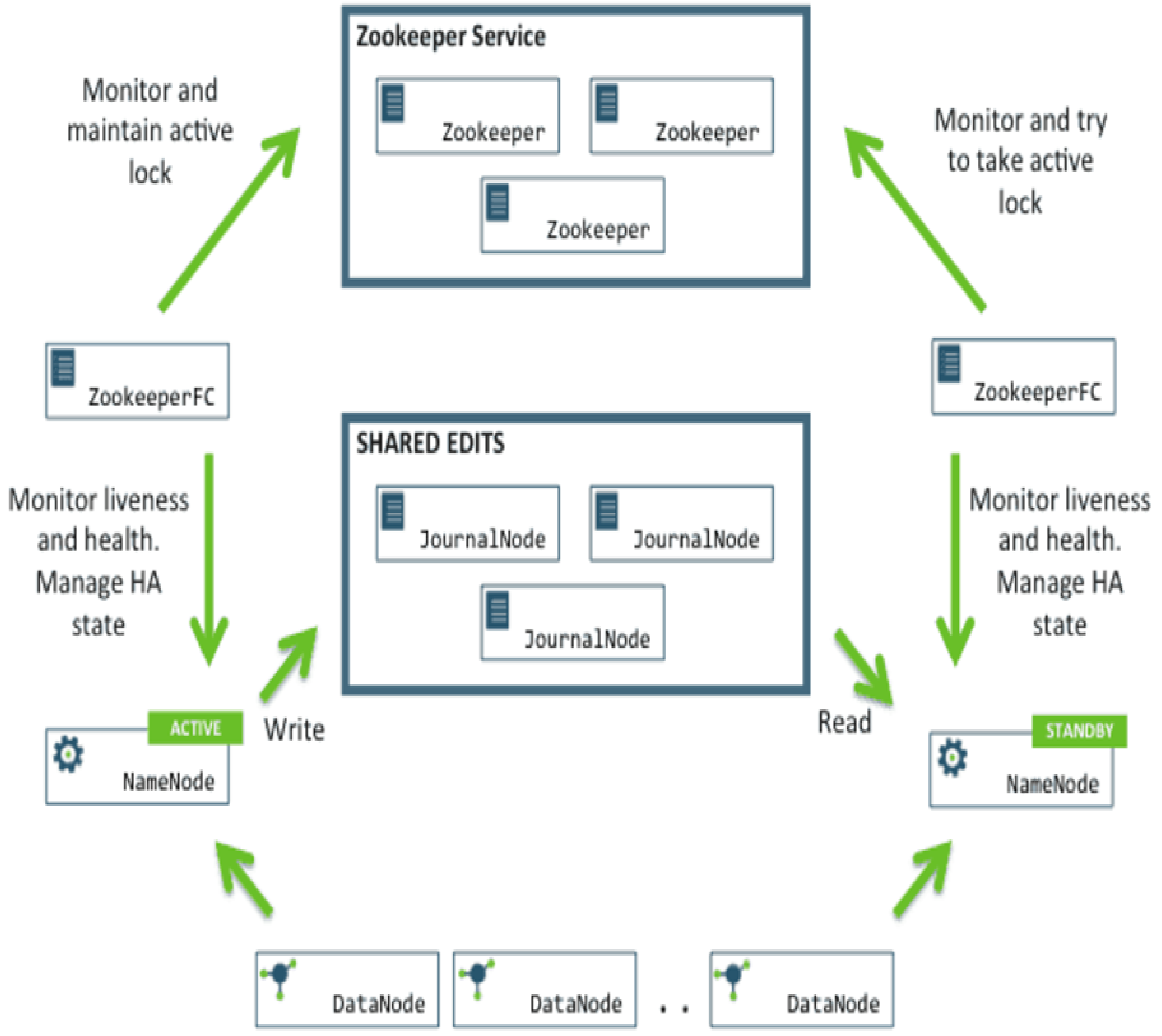
- QJM is dedicated HDFS implementation
- QJM runs as a group of Journal nodes and each edit must be written to majority of journal nodes
- QJM only allows one Name Node to write to the edit log at one time
- QJM has ssh fencing method implemented which helps to avoid the Name Node split-brain scenario



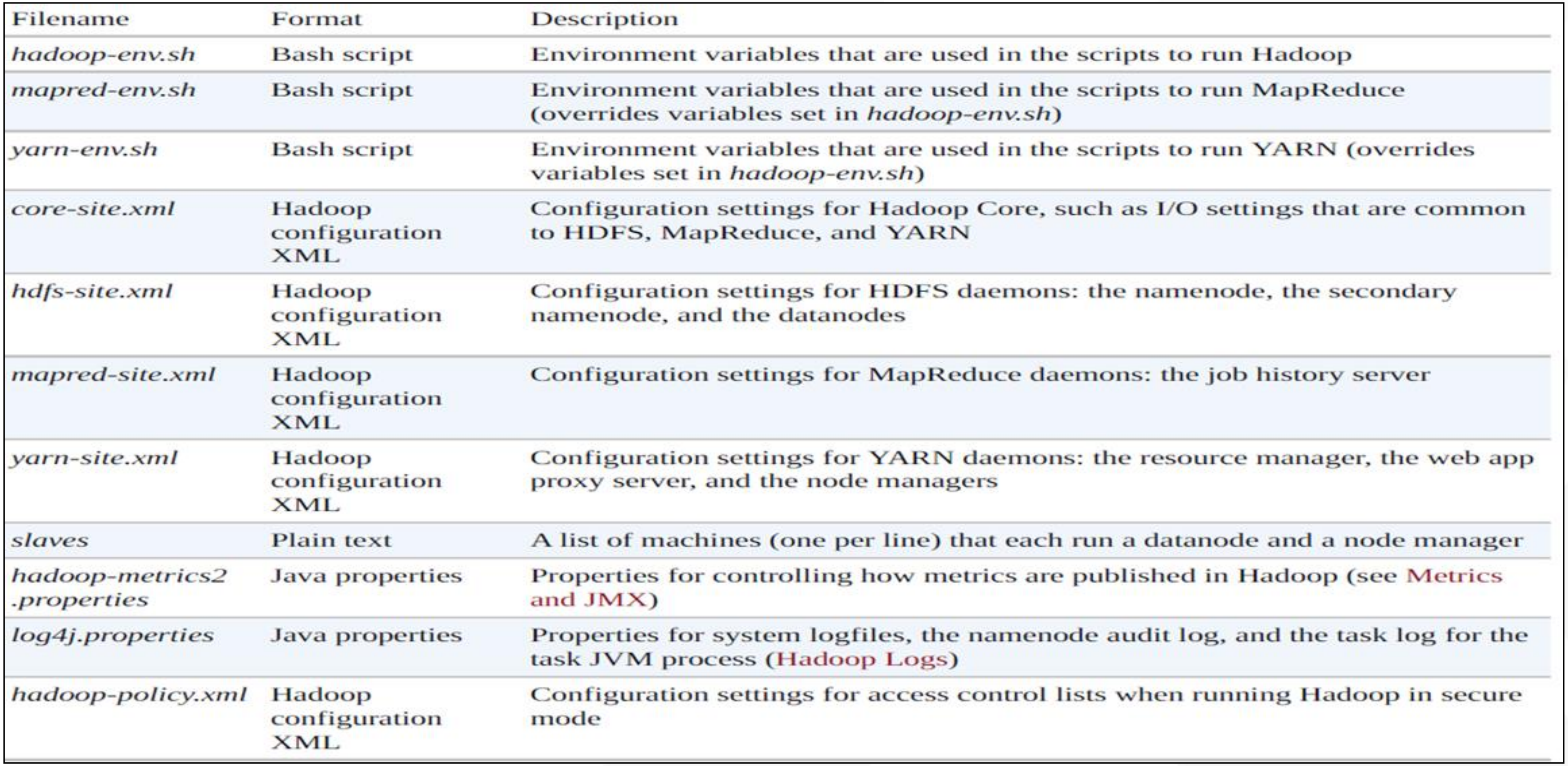
HDFS HA & Automatic failover using QJM and ZooKeeper

- QJM is dedicated HDFS implementation
- QJM runs as a group of Journal nodes and each edit must be written to majority of journal nodes
- QJM only allows one NameNode to write to the edit log at one time
- QJM has ssh fencing method implemented which helps to avoid the NameNode split-brain scenario

- ZooKeeper Failover Controller(ZKFC) is responsible for HA Monitoring for Name Node service and automatic failover
- There are two ZKFC process, one on each Name Node Machine
- ZKFC uses the ZooKeeper Service for coordination in determining which is the Active NameNode and in determining when to failover to the Standby NameNode.

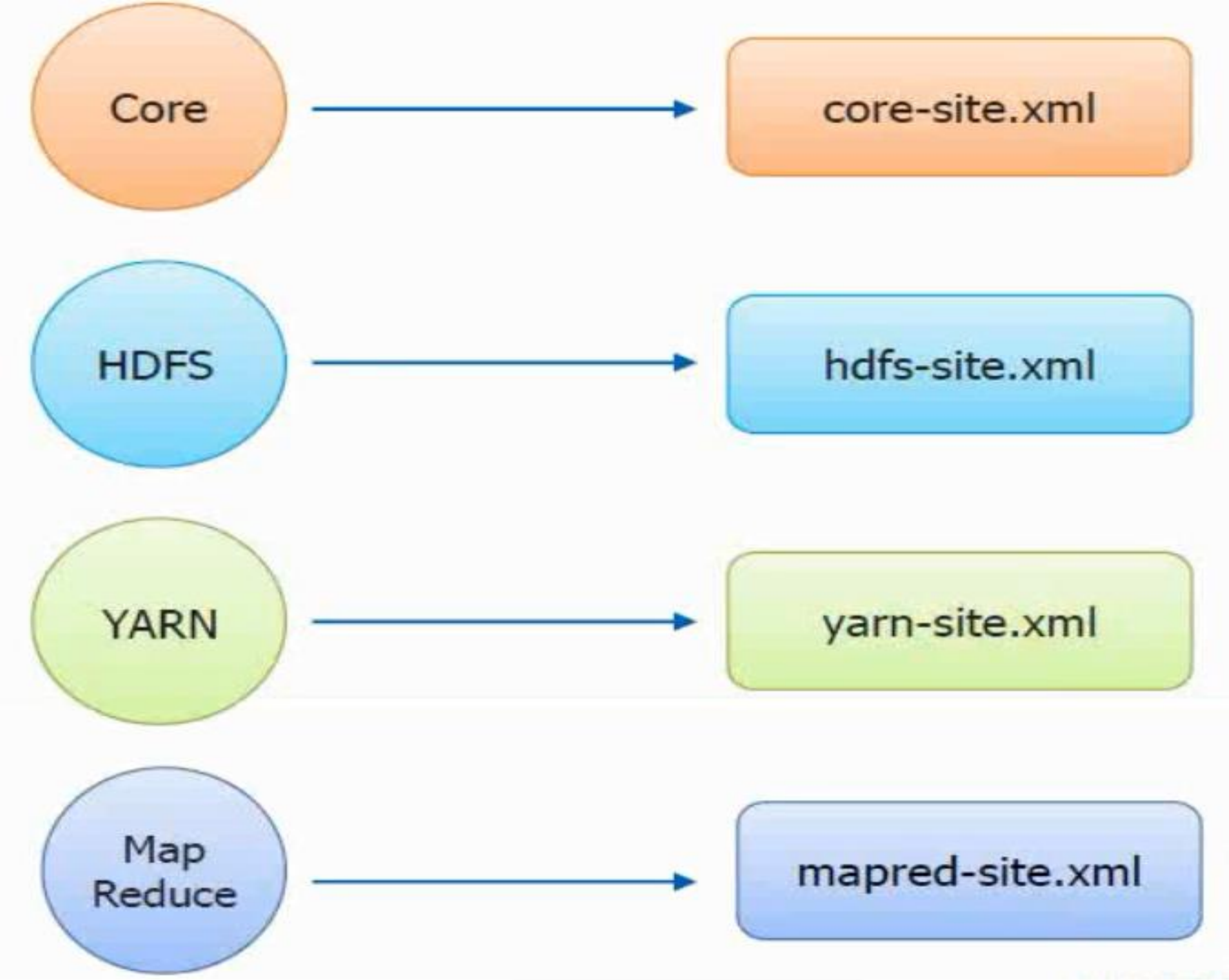


Hadoop Configuration Files

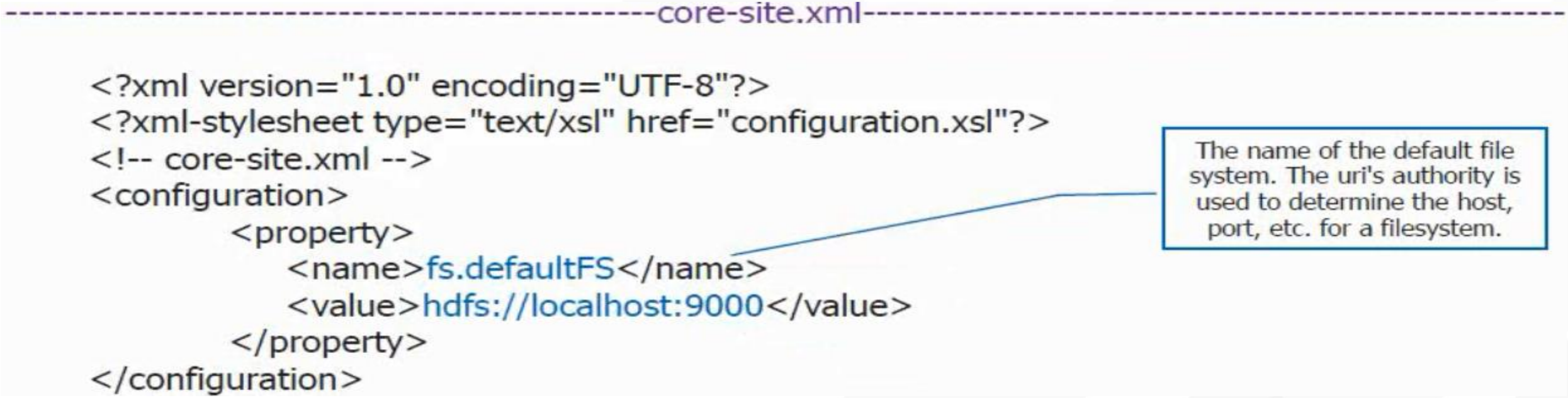


Filename	Format	Description
hadoop-env.sh	Bash script	Environment variables that are used in the scripts to run Hadoop
mapred-env.sh	Bash script	Environment variables that are used in the scripts to run MapReduce (overrides variables set in *hadoop-env.sh*)
yarn-env.sh	Bash script	Environment variables that are used in the scripts to run YARN (overrides variables set in *hadoop-env.sh*)
core-site.xml	Hadoop configuration XML	Configuration settings for Hadoop Core, such as I/O settings that are common to HDFS, MapReduce, and YARN
hdfs-site.xml	Hadoop configuration XML	Configuration settings for HDFS daemons: the namenode, the secondary namenode, and the datanodes
mapred-site.xml	Hadoop configuration XML	Configuration settings for MapReduce daemons: the job history server
yarn-site.xml	Hadoop configuration XML	Configuration settings for YARN daemons: the resource manager, the web app proxy server, and the node managers
slaves	Plain text	A list of machines (one per line) that each run a datanode and a node manager
hadoop-metrics2 .properties	Java properties	Properties for controlling how metrics are published in Hadoop (see Metrics and JMX)
log4j.properties	Java properties	Properties for system logfiles, the namenode audit log, and the task log for the task JVM process (Hadoop Logs)
hadoop-policy.xml	Hadoop configuration XML	Configuration settings for access control lists when running Hadoop in secure mode

Hadoop Core Configuration files



Hadoop Core Configuration files

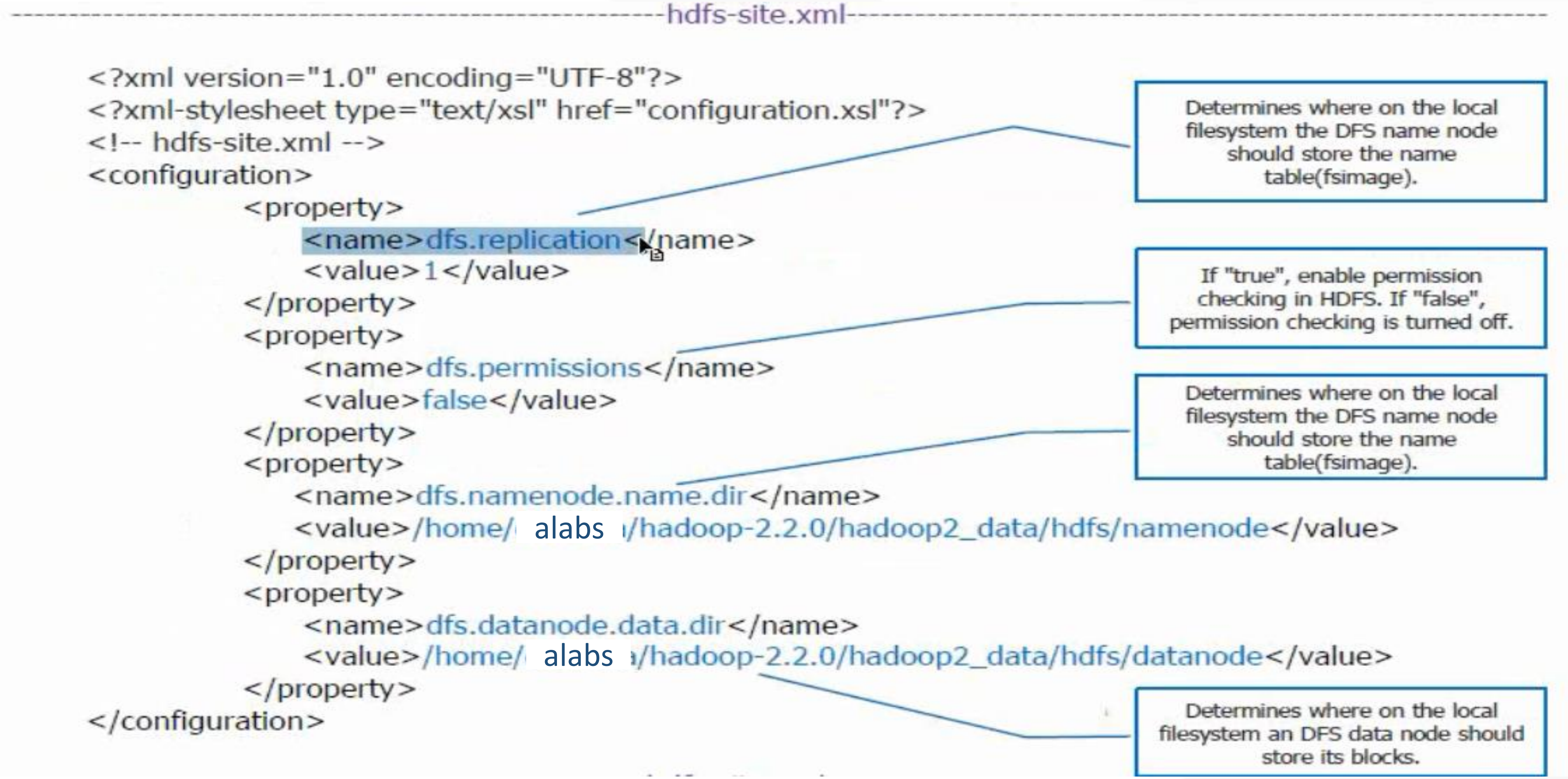


--core-site.xml--

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- core-site.xml -->
<configuration>
        <property>
            <name>fs.defaultFS</name>
            <value>hdfs://localhost:9000</value>
        </property>
</configuration>
```

The name of the default file system. The uri's authority is used to determine the host, port, etc. for a filesystem.

Hadoop Core Configuration files



---hdfs-site.xml---

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- hdfs-site.xml -->
<configuration>
          <property>
              <name>dfs.replication</name>
              <value>1</value>
          </property>
          <property>
              <name>dfs.permissions</name>
              <value>false</value>
          </property>
          <property>
             <name>dfs.namenode.name.dir</name>
             <value>/home/ alabs /hadoop-2.2.0/hadoop2_data/hdfs/namenode</value>
          </property>
          <property>
              <name>dfs.datanode.data.dir</name>
              <value>/home/ alabs /hadoop-2.2.0/hadoop2_data/hdfs/datanode</value>
          </property>
</configuration>
```

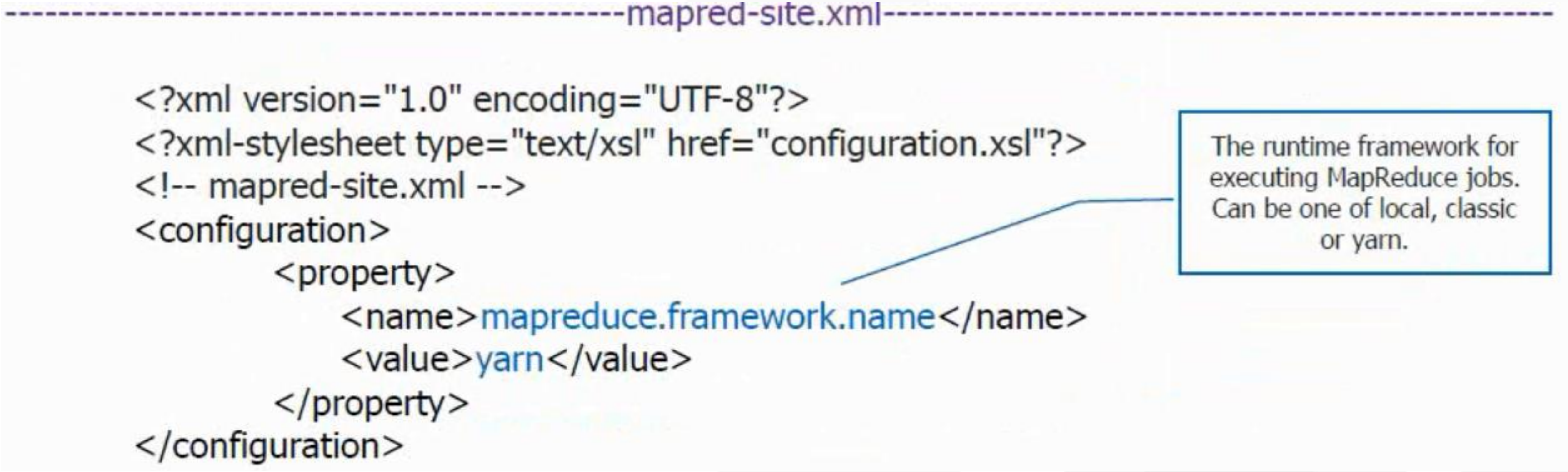
Determines where on the local filesystem the DFS name node should store the name table(fsimage).

If "true", enable permission checking in HDFS. If "false", permission checking is turned off.

Determines where on the local filesystem the DFS name node should store the name table(fsimage).

Determines where on the local filesystem an DFS data node should store its blocks.

Hadoop Core Configuration files

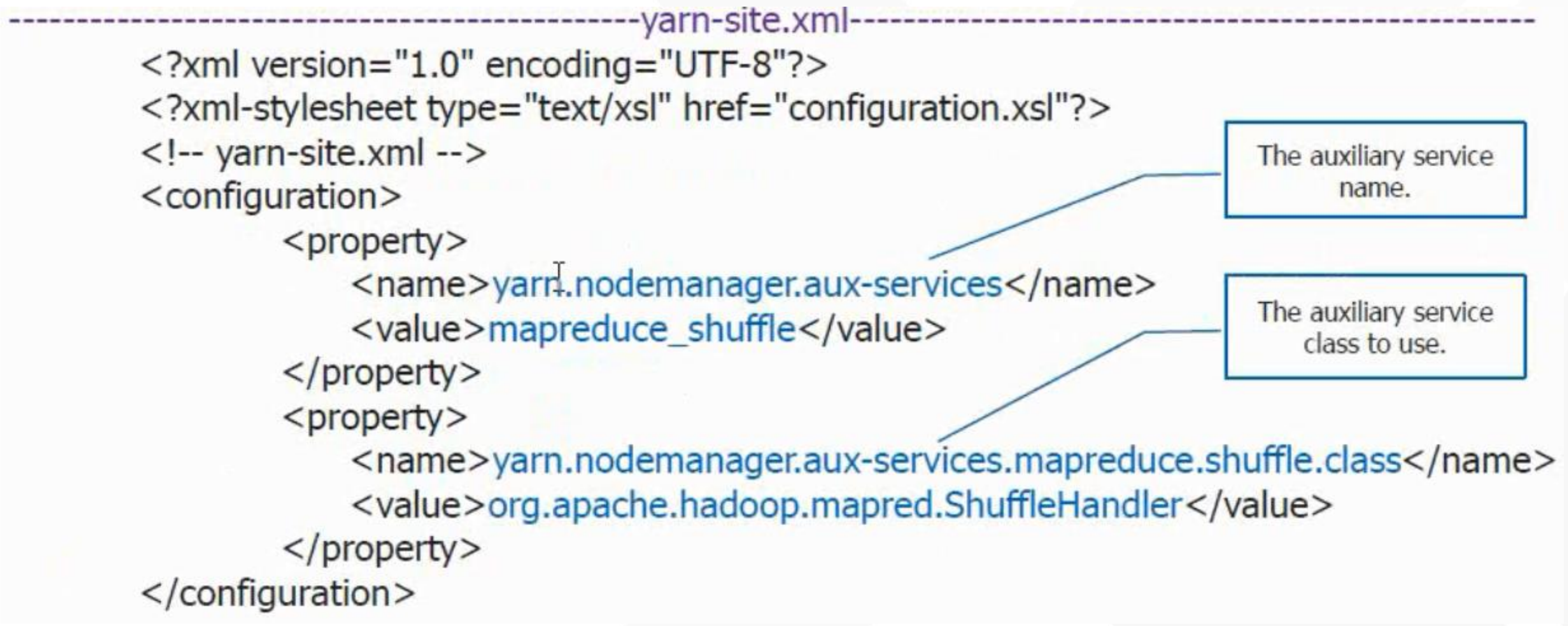


---mapred-site.xml---

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- mapred-site.xml -->
<configuration>
        <property>
            <name>mapreduce.framework.name</name>
            <value>yarn</value>
        </property>
</configuration>
```

The runtime framework for executing MapReduce jobs. Can be one of local, classic or yarn.

Hadoop Core Configuration files



```
------------------------------------yarn-site.xml------------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- yarn-site.xml -->
<configuration>
        <property>
            <name>yarn.nodemanager.aux-services</name>
            <value>mapreduce_shuffle</value>
        </property>
        <property>
            <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
            <value>org.apache.hadoop.mapred.ShuffleHandler</value>
        </property>
</configuration>
```

The auxiliary service name.

The auxiliary service class to use.

Contact Us

Visit us on: http://www.analytixlabs.in/

For more information, please contact us: http://www.analytixlabs.co.in/contact-us/

Or email: info@analytixlabs.co.in

Call us we would love to speak with you: (+91) 9910509849

Join us on:

Twitter - http://twitter.com/#!/AnalytixLabs

Facebook - http://www.facebook.com/analytixlabs

LinkedIn - http://www.linkedin.com/in/analytixlabs

Blog - http://www.analytixlabs.co.in/category/blog/