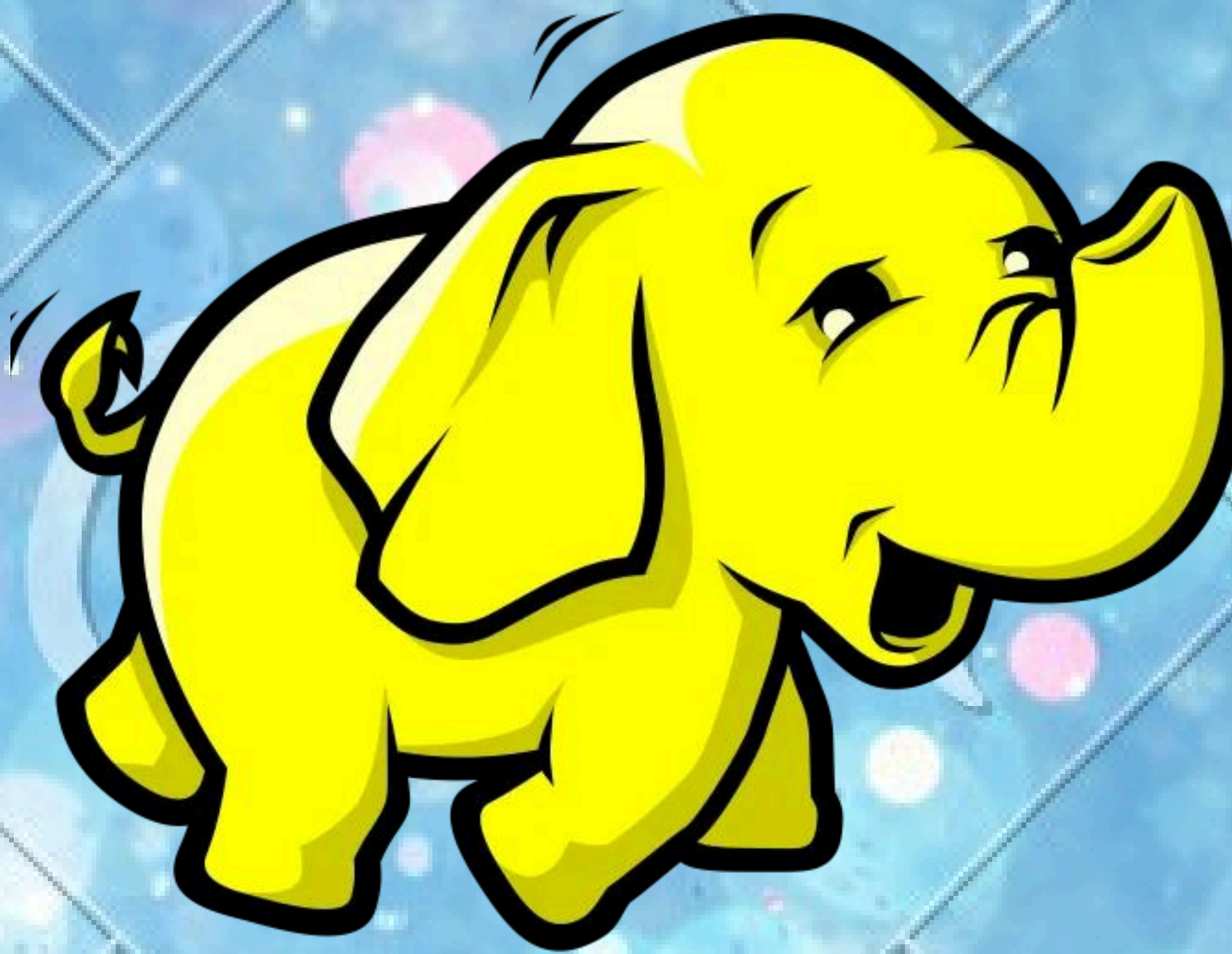


TOP 30

hadoop



Interview Questions

Asked by MAANG Companies

Disclaimer

Every learning journey is unique, and success comes with persistence and effort.

Use this document be your guide to grasp essential Hadoop concepts in Big Data, and prepare confidently for your next interview.

Section 1:

Hadoop Basics

Q1. **What is Hadoop, and why is it important for Big Data?**

Ans. Hadoop is an open-source framework that stores and processes large amounts of data across clusters of commodity hardware.

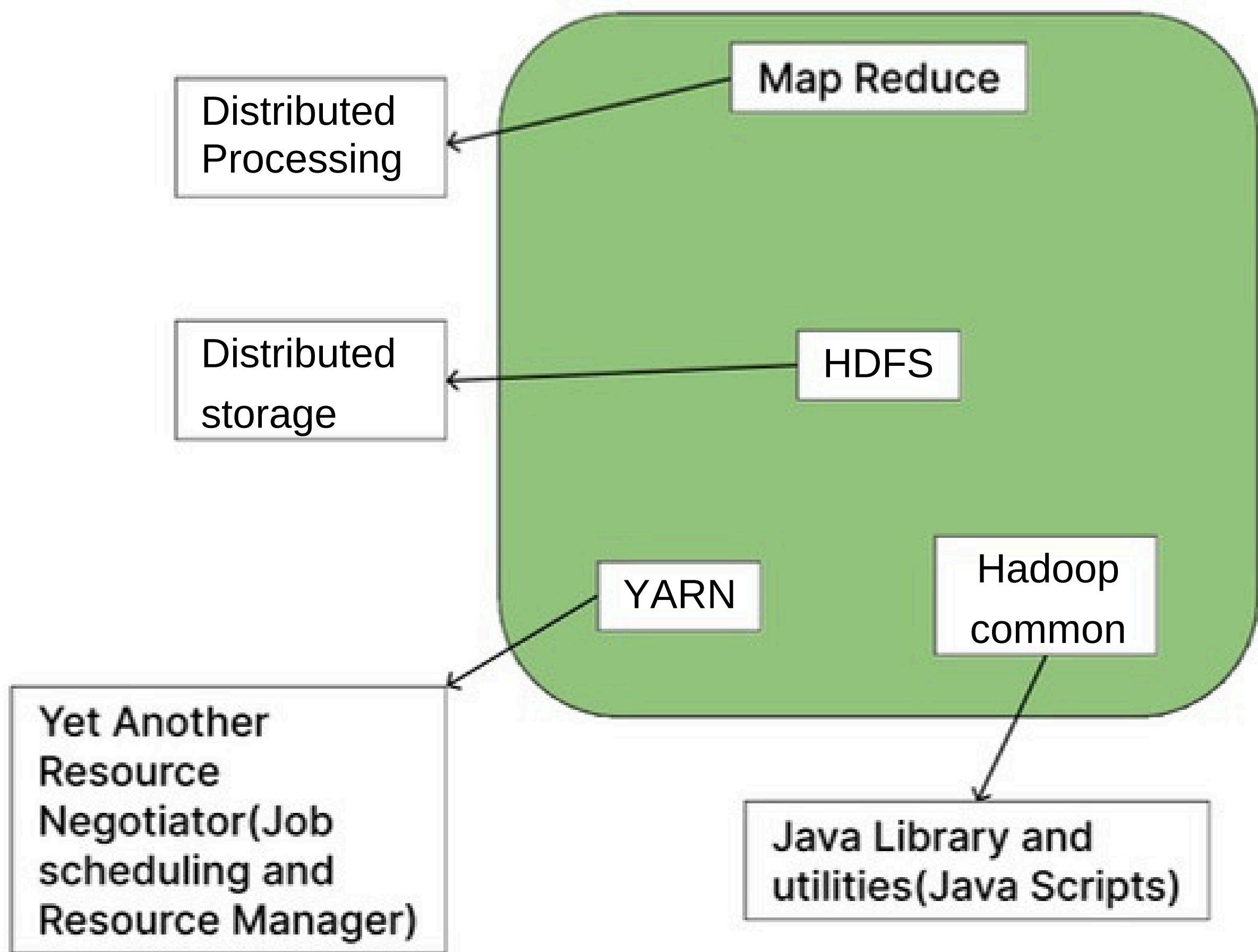
It allows parallel processing of massive datasets using MapReduce and provides distributed storage using HDFS.

Its significance lies in handling large volumes of structured, semi-structured, and unstructured data efficiently, which is crucial in Big Data analytics.

Q2. **What are the main components of Hadoop?**

Ans. Hadoop has three main components:

1. **HDFS (Hadoop Distributed File System):** A distributed file system that stores large datasets across many nodes.
2. **MapReduce:** A programming model for processing large data in parallel.
3. **YARN (Yet Another Resource Negotiator):** Manages cluster resources and job scheduling.



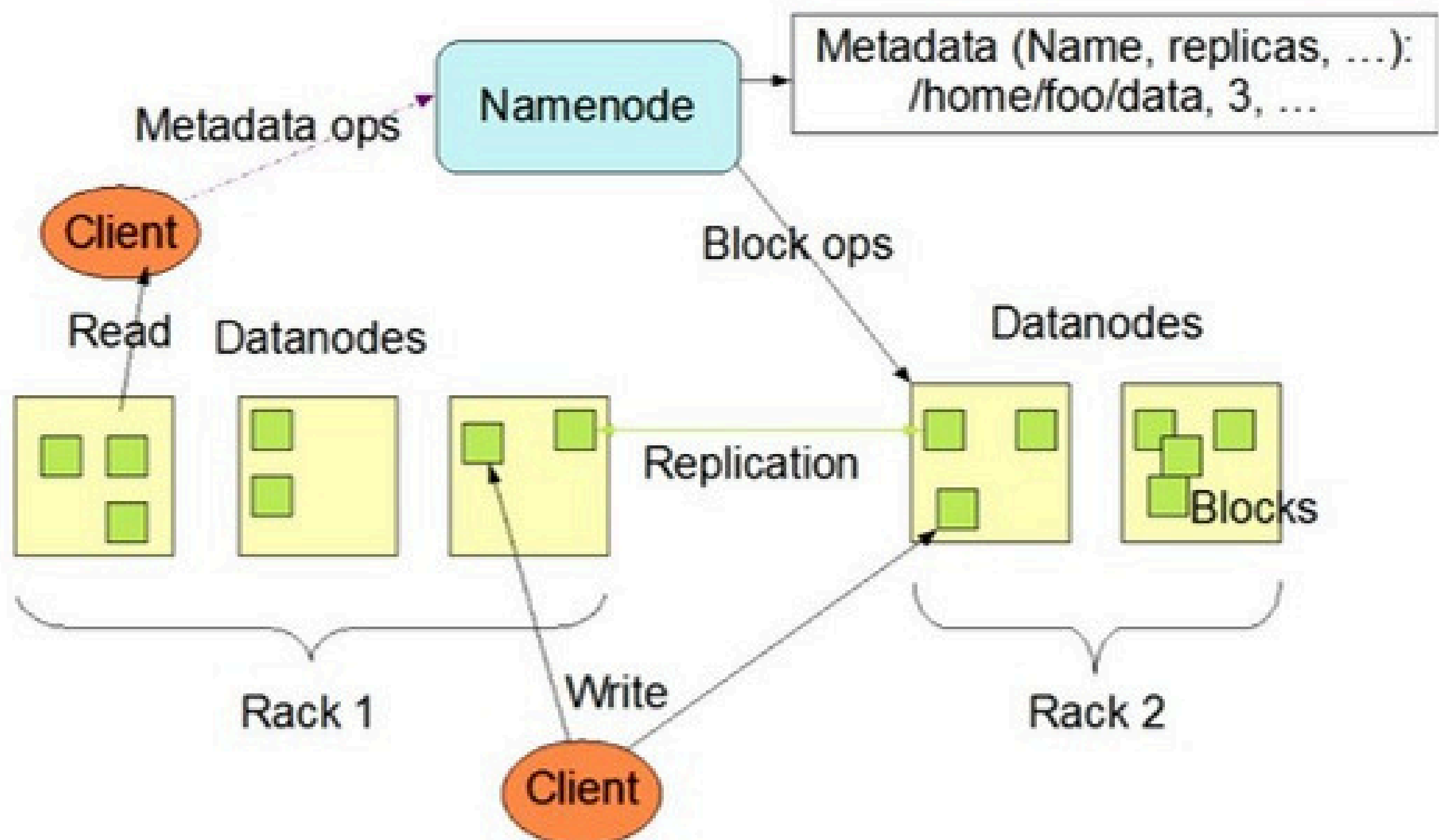
Section 2:

Hadoop Architecture

Q 3. What is HDFS, and what is its purpose?

Ans. HDFS (Hadoop Distributed File System) is designed for large-scale storage and high fault tolerance.

HDFS splits large files into smaller, fixed-size blocks (default: 128 MB) and stores them across multiple nodes, ensuring data redundancy by replicating blocks across the cluster.



Q4. What is the NameNode, and what is its role in HDFS?

Ans. The **NameNode** is the master node of HDFS that manages the metadata for files stored in the cluster, including information like file locations, replication details, and directory structures.

The NameNode does not store actual data, but it coordinates data storage across DataNodes.

```
bash
```

[Copy code](#)

```
hdfs dfsadmin —report
```

This command checks the status of the NameNode and the available nodes in the cluster.

Q5. What is a DataNode, and what does it do?

Ans. **DataNodes** are worker nodes that store actual data in HDFS. They store the blocks and send regular heartbeats to the NameNode to report their status. DataNodes also handle read and write requests from clients.

Code Example: Start DataNode Manually

```
bash
```

[Copy code](#)

```
start—dfs. sh
```

This command starts all HDFS daemons, including the DataNodes.

Section 3:

HDFS (Hadoop Distributed File System)

Q 6. What is a Block in HDFS?

Ans. A **block** is the smallest unit of data storage in HDFS, typically 128 MB in size. Blocks are replicated across nodes to ensure fault tolerance. By splitting files into blocks, Hadoop can store and process large files across multiple nodes simultaneously.

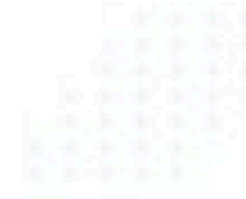
Code Example: Change Block Size

```
bash  
hdfs
```

[Copy code](#)

```
hdfs dfs -Ddfs.block.size=268435456 -put largefile.txt  
/user/hadoop/largefile
```

This command changes the block size to 256 MB when uploading a file.



Q7. Explain data replication in HDFS.

Ans. HDFS replicates each data block across multiple DataNodes to ensure fault tolerance. The default replication factor is three, meaning each block is stored on three different nodes. This redundancy allows data to be available even if some nodes fail.

Code Example: View or Set Replication Factor

bash

Copy code

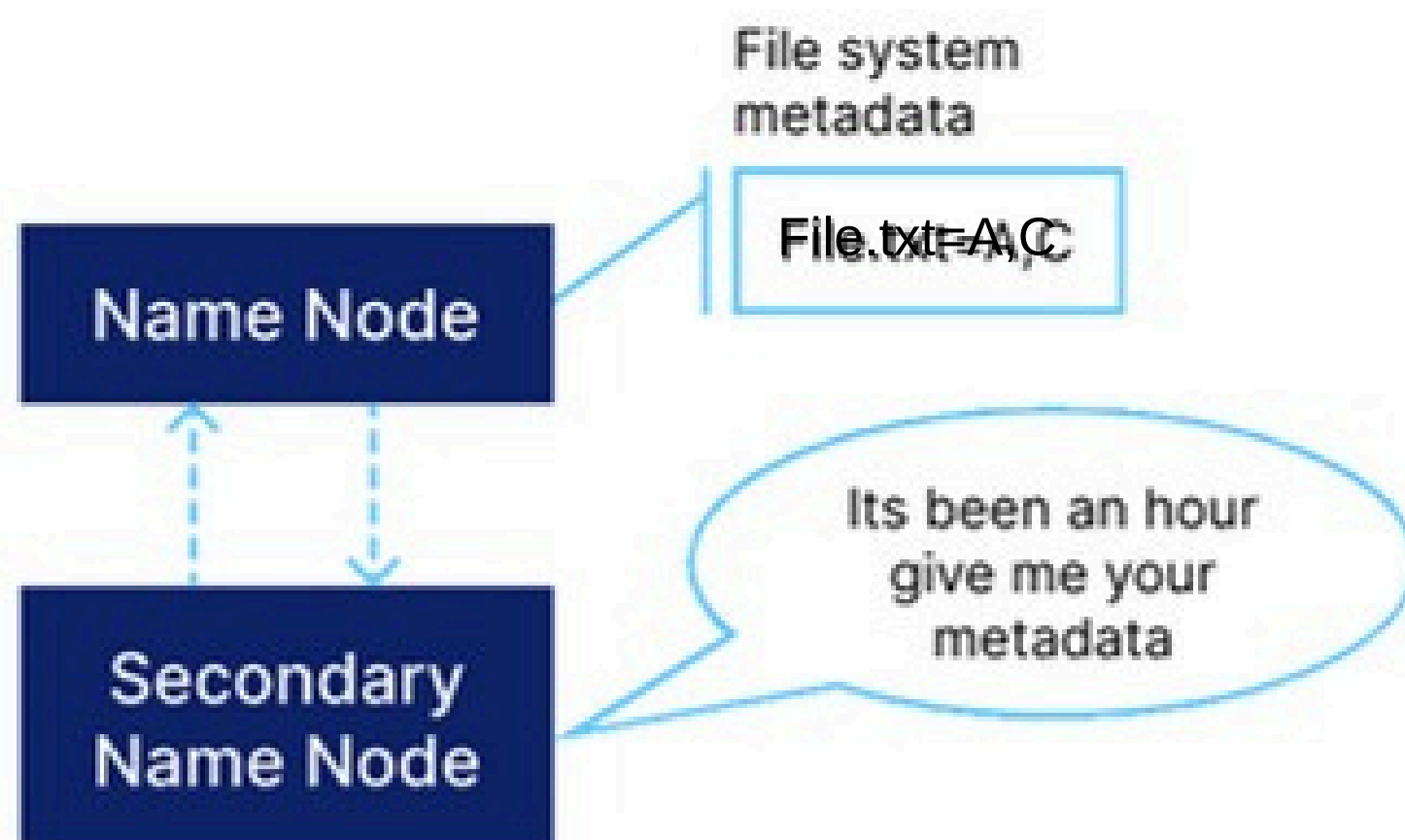
hdfs

```
hdfs dfs -setrep -w 3 /user/hadoop/myfile
```

This command sets the replication factor for myfile to three.

Q8. What is the Secondary NameNode, and why is it used?

Ans. The **Secondary NameNode** assists the NameNode by periodically merging the **FSImage** (file system image) and **edit logs** to create a new, updated FSImage. It helps reduce the load on the NameNode, but it is not a backup node.



Section 4:

MapReduce Framework

Q 6. What is MapReduce, and what are its main phases?

Ans. MapReduce is a programming model for parallel processing of large data sets. It has two main phases:

1. **Map:** Processes input data into key-value pairs.
2. **Reduce:** Aggregates and summarizes the data to produce final output.

Code Example: Writing a Simple MapReduce Job in Java

 Copy code

```
public class WordCount {  
    public static class TokenizerMapper extends  
Mapper<Object, Text, Text, IntWritable> {  
        private final static IntWritable one = new  
IntWritable(1);  
        private Text word = new Text();  
        public void map(Object key, Text value,  
Context context) throws IOException,  
InterruptedException {  
StringTokenizer itr = new
```

```
StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
    }
}
// Reducer code follows... }
```

This Java code snippet shows a simple MapReduce job that counts words.

Infographic Prompt: A flowchart showing the Map phase (splitting data into key-value pairs) and the Reduce phase (aggregating results).

Q10. What is the purpose of a combiner in MapReduce?

Ans. A **combiner** is a mini-reducer that performs local aggregation of output data from the **map** function before sending it to the reducer. It helps minimize the amount of data transferred between the map and reduce phases, thus improving performance.

Code Example: Using a Combiner

```
java
j ob. setCombinerClass ( IntSumReducer. class ) ;
```

This line sets the combiner class for a MapReduce job, reducing intermediate data.

Q11.

What is speculative execution in Hadoop MapReduce?

Ans. **Speculative execution** runs duplicate tasks on different nodes if one task appears to be running slower than expected. This helps ensure that straggling tasks do not delay the entire job.

Code Configuration: Enable Speculative Execution

```
rope rtY>  
<name>map reduce. map. speculative</name>  
<value>true</value>  
</property>
```

 Copy code

This configuration in the [mapred-site.xml](#) file enables speculative execution for mappers.

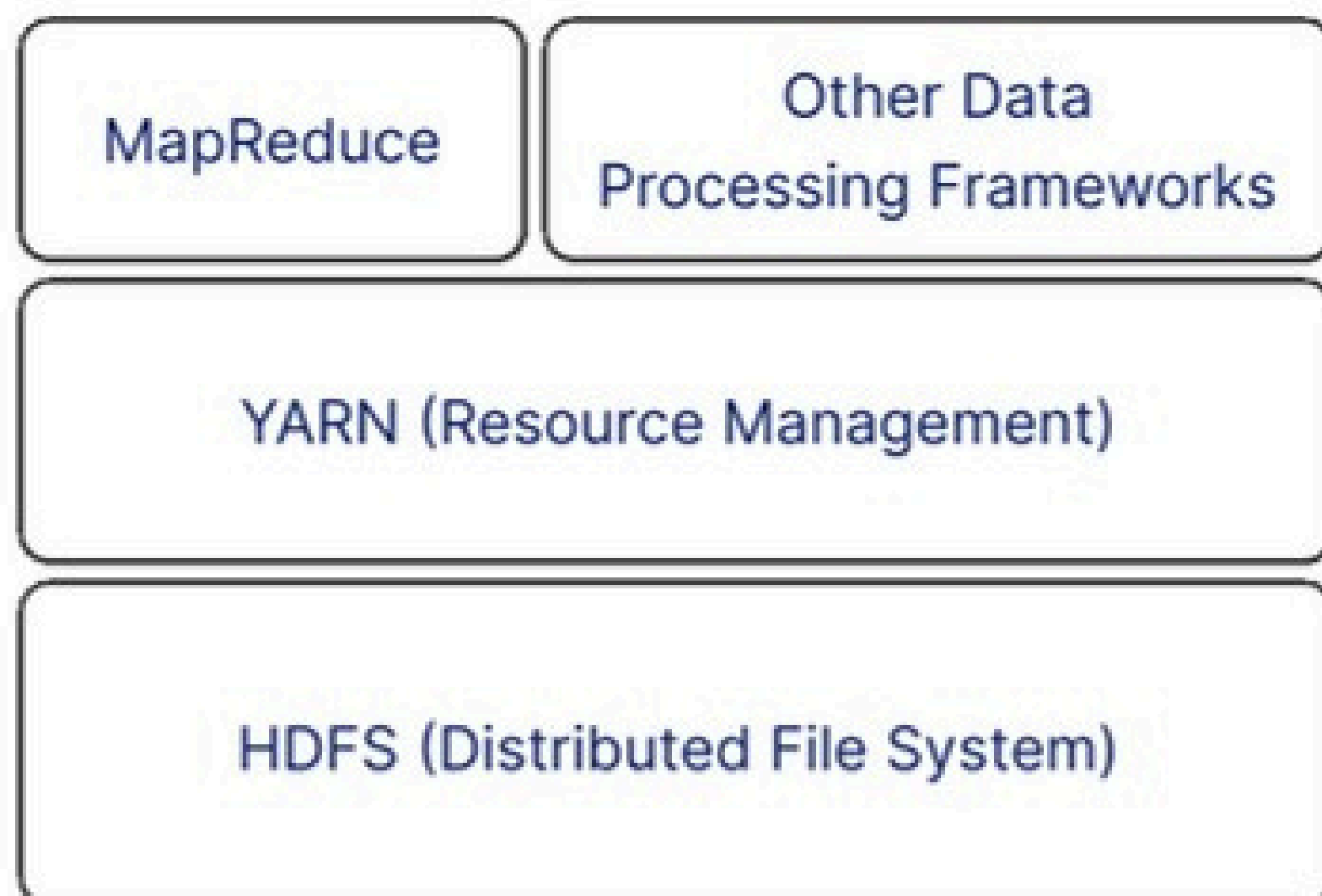
Section 5:

YARN (Yet Another Resource Negotiator)

Q12. **What is YARN, and what is its purpose?**

Ans. YARN (Yet Another Resource Negotiator) is the resource management layer of Hadoop. It separates resource management and job scheduling from data processing, making Hadoop more flexible and efficient.

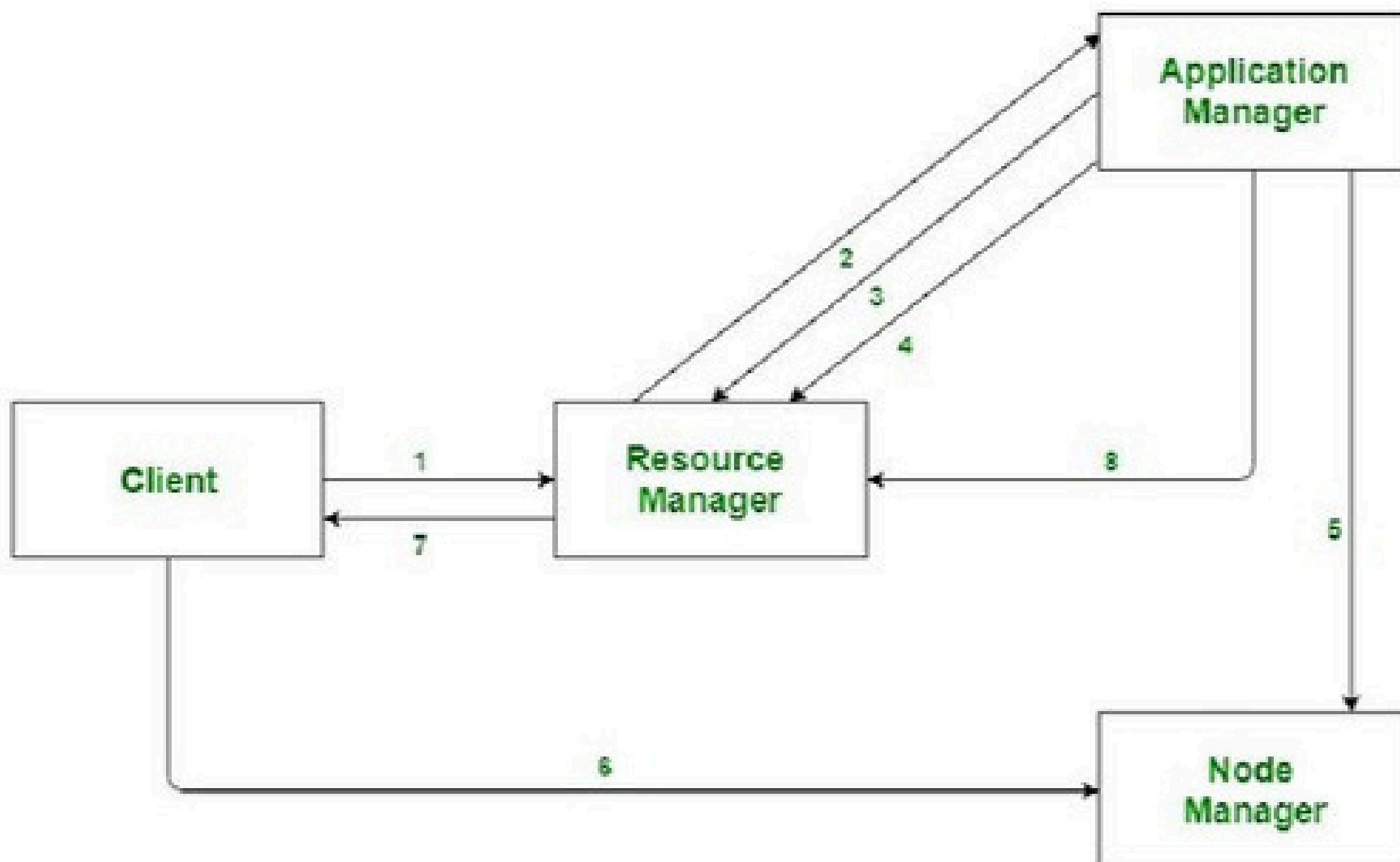
YARN allows multiple data processing engines to run on Hadoop, such as Spark, MapReduce, and Tez.



Q13. What are the main components of YARN?

Ans. The key components of YARN are:

1. **ResourceManager:** Allocates resources to different applications.
2. **NodeManager:** Monitors resources on individual nodes.
3. **ApplicationMaster:** Manages the lifecycle of applications.

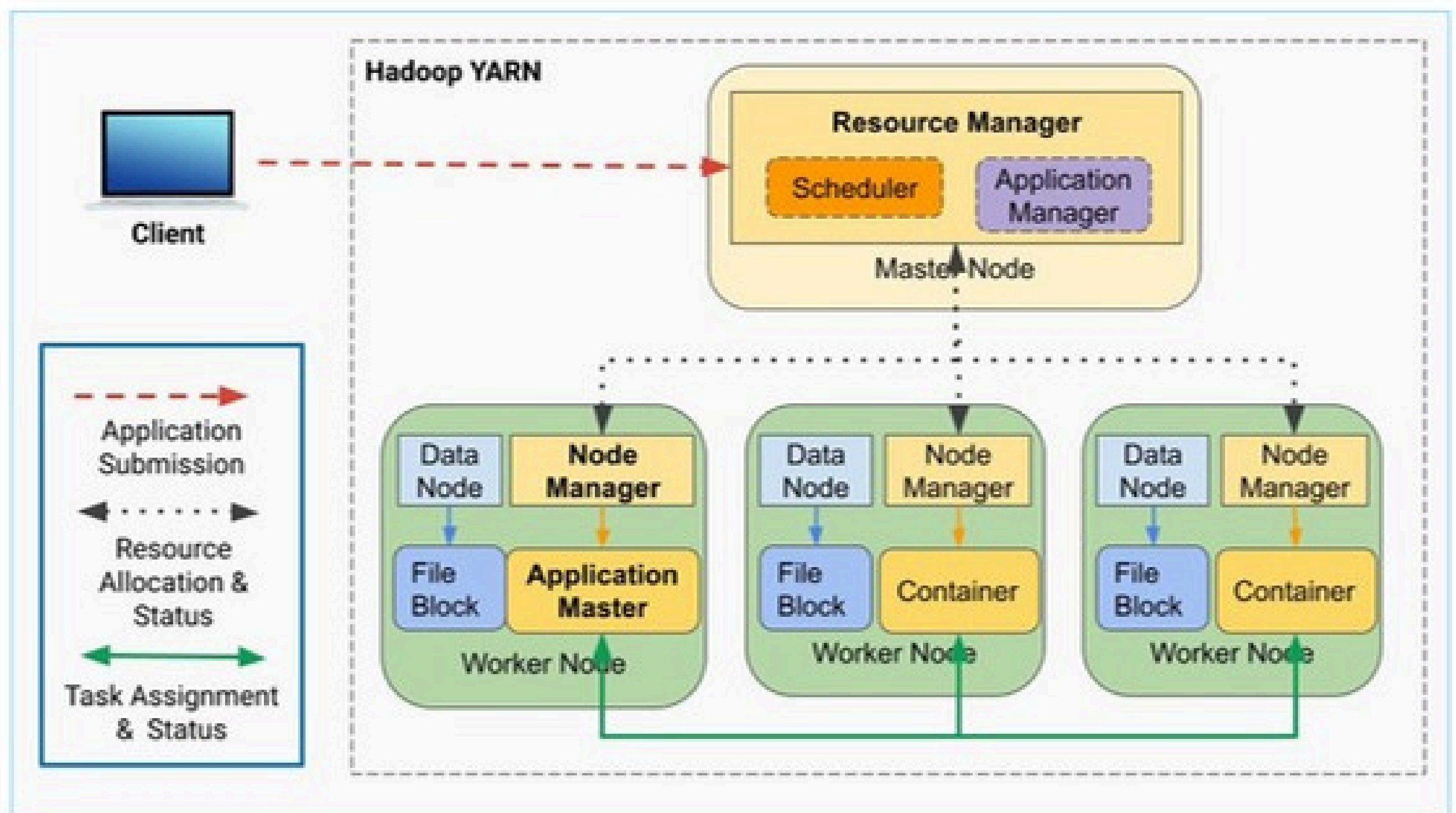


Q14.

Explain how ResourceManager works in YARN.

Ans. The **ResourceManager** is the master authority for resource management in YARN. It allocates resources to various applications running on the cluster based on availability and priority.

It consists of two parts: **Scheduler** (allocates resources to jobs) and **ApplicationManager** (manages applications).



Section 6:

Hadoop Ecosystem and Tools

Q 15. What is Apache Hive, and what is its use in Hadoop?

Ans. Hive is a data warehousing tool built on top of Hadoop that allows users to run **SQL-like queries** (using HiveQL) on data stored in HDFS.

It simplifies data analysis tasks and makes Hadoop accessible to those familiar with SQL.

Hive Query Example

```
CREATE TABLE employee (id INT, name STRING, salary  
FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED BY  
' , ' ;  
SELECT * FROM employee WHERE salary > 50000;
```

 Copy code

This Hive query creates a table and selects employees with a salary greater than 50,000.

Q16.

What is Apache Pig, and what is its role in Hadoop?

Ans. Apache Pig is a high-level platform for creating MapReduce programs using a scripting language called **Pig Latin**.

It is used for data transformation, such as loading, processing, and analyzing large data sets in HDFS. Pig simplifies the creation of MapReduce programs by abstracting the lower-level coding details.

Pig Latin Script Example

```
data = LOAD 'hdfs://path/to/data' USING
PigStorage(',') AS (name:chararray, age:int,
salary:float);
filtered_data = FILTER data BY salary > 50000;
DUMP filtered_data;
```

 Copy code

This script loads data from HDFS, filters it based on salary, and prints the results.

Q17.

What is Apache HBase, and why would you use it?

Ans. Apache HBase is a NoSQL database that runs on top of HDFS, allowing for random, real-time read and write access to data.

It is particularly useful for applications that need fast retrieval of small amounts of data and supports **large tables** with billions of rows and millions of columns.

Code Example: HBase Table Creation using HBase Shell

shell

 Copy code

```
create 'employee', 'personal _ info'
, 'professional _ info'
```

This command creates an HBase table named `employee` with two column families: `personal_info` and `professional_info`.


Q18. How does Hadoop ensure fault tolerance?

Ans. Hadoop ensures fault tolerance by replicating data blocks across multiple DataNodes.

If a DataNode fails, HDFS can still retrieve the data from replicated nodes. **MapReduce** also achieves fault tolerance by re-running failed tasks on other available nodes.

Code Example: Setting Replication Factor

xml

 Copy code

```
<property>
  <name>dfs.replication</name>
  <value>3</value>
</property>
```

This setting configures the replication factor for HDFS to 3, ensuring data redundancy.

Q19.

What are Counters in Hadoop MapReduce?

Ans. **Counters** are a mechanism to count events, such as the number of processed records or error occurrences, during the execution of a MapReduce job. They help track the job's progress and monitor the health of the MapReduce job.

Code Example: Using Counters in Java MapReduce

```
enum MyCounters {  
    RECORD_COUNT ,  
    ERROR COUNT }  
  
public void map(LongWritable key, Text value, Context  
context) {  
    context. getCounter(MyCounters. RECORD COUNT) .  
    increment (1) ;  
    // Logic here... }  
}
```

This example defines a counter to count the number of records processed by the Mapper.

Q 20.

What is the Hadoop Distributed Cache, and why is it used?

Ans. The **Distributed Cache** is a mechanism in Hadoop that allows files needed by jobs (e.g., JAR files, text files) to be cached and made available across all nodes running a MapReduce job. This reduces the need to repeatedly access HDFS for small files.

Code Example: Using Distributed Cache in Java

java

 Copy code

```
job.addCacheFile(new URI ("/user/hadoop/cache/file.  
txt#localfile"));
```

This line adds a file to the distributed cache so that each mapper or reducer can access it locally.

Section 7:

Advanced Hadoop Concepts

Q21. **What is Rack Awareness in Hadoop, and why is it important?**

Ans. Rack Awareness is a concept in Hadoop that allows the cluster to understand the physical topology of the nodes, specifically which rack a node is part of.

This knowledge is used to decide data replication to ensure that data copies are stored on different racks to protect against rack failures.

Code Configuration: Rack Awareness

xml

 Copy code

```
<property>
  <name>net.topology.script.file.name</ name>
  <value>/path/to/rack-awareness-script.sh</value>
</property>
```

This configuration sets a script that helps Hadoop determine rack topology.

Q22.

What is speculative execution in Hadoop MapReduce?

Ans. Speculative execution in Hadoop runs multiple instances of the same task on different nodes if a particular task is taking too long to complete.

The result from the first instance to complete is taken, and the others are killed, thereby ensuring faster job completion.

Configuration Example: Enabling Speculative Execution

xml

Copy code

```
<property>
<property>
  <name>mapreduce.map.speculative</name>
  <value>true</value>
</property>
<property>
  <name>mapreduce.map.speculative</name>
  <value>true</value>
</property>
```

This enables speculative execution for both map and reduce tasks.

Q23.

What is Hadoop Streaming, and why is it useful?

Ans. Hadoop Streaming is a utility that allows users to create and run MapReduce jobs with any executable or script (such as Python, Perl, etc.) as the Mapper or Reducer.

It makes Hadoop accessible to programmers who prefer scripting languages over Java.

Command Example: Running a Streaming Job

```
xml Copy code  
  
hadoop jar /path/to/hadoop-streaming.jar \  
-mapper /path/to/mapper.py \  
-reducer /path/to/reducer.py \  
-input /user/hadoop/input \  
-output /user/hadoop/output
```

This command runs a streaming job using Python scripts as mapper and reducer.

Q24. What are the primary functions of Reducer in Hadoop?

Ans. The **Reducer** in Hadoop takes the intermediate output from the **Mapper** and processes it to generate the final aggregated output.

Its main functions are **shuffle and sort** (grouping similar keys together) and **reduce** (processing these keys to produce a final summary).

Code Example: Reducer Class in Java

```
java Copy code  
  
public static class IntSumReducer extends  
Reducer<Text, IntWritable, Text, IntWritable> {
```

```
public void reduce(Text key, Iterable<IntWritable>
values, Context context) throws
Context context) throws IOException,
InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
        sum += val.get();
    }
    context.write(key, new IntWritable(sum));
}
}
```

This example sums the values associated with each key, typical in a word count program.

Infographic Prompt: A flowchart showing the shuffle, sort, and reduce phases of a Reducer, ending with the final output.

Q25. How can you handle small files in HDFS efficiently?

Ans. Handling small files efficiently in HDFS can be achieved by using:

- **HAR (Hadoop Archive)** to combine multiple small files into a single archive file.
- **SequenceFiles**, which store key-value pairs in a compressed format.
- **HBase**, stores data in a more structured format, reducing the load on HDFS.

Command Example: Creating HAR

 Copy code

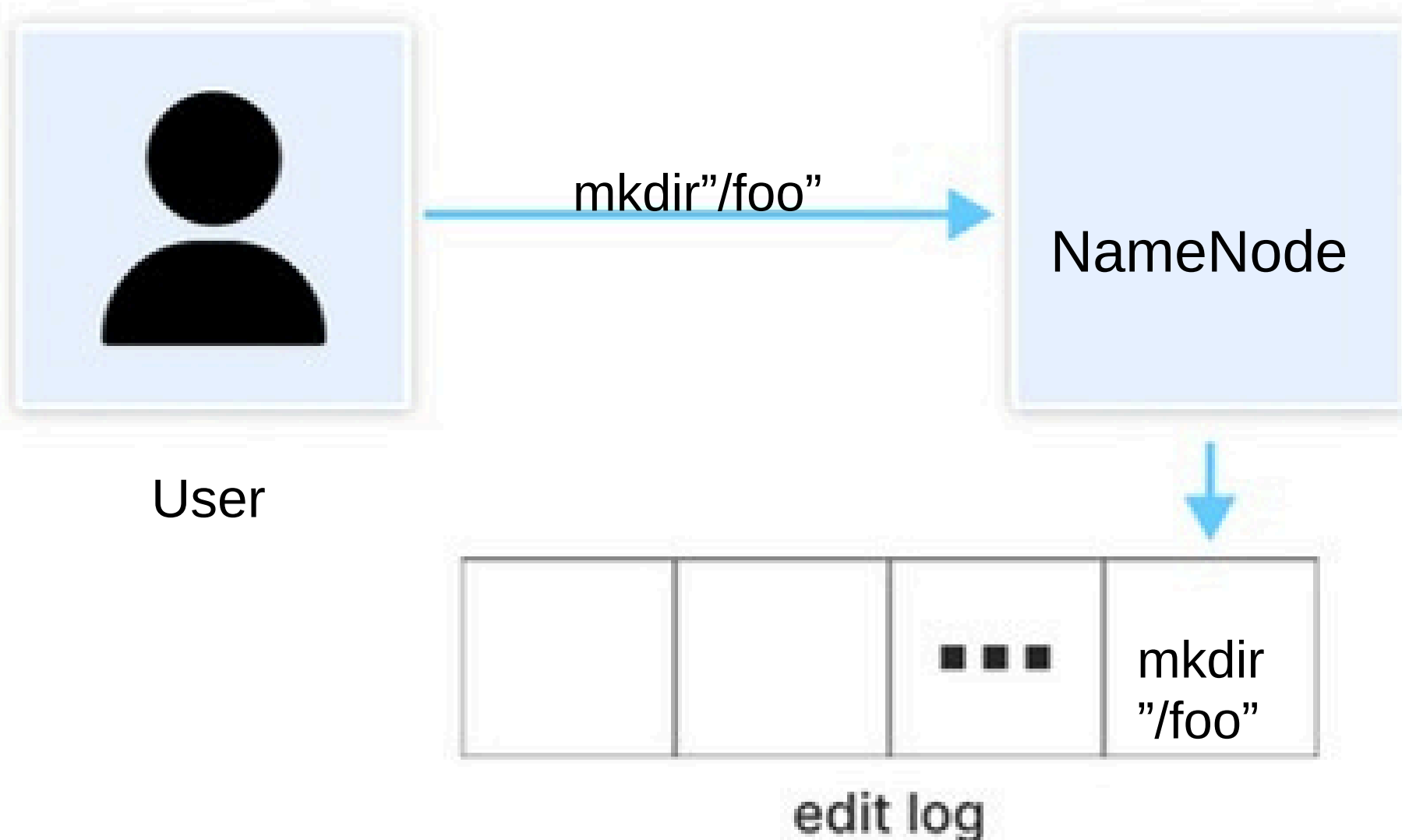
```
hadoop archive -archiveName myArchive.har -p /user/  
hadoop/input /user/hadoop/output
```

This command creates a HAR file to combine small files.

Q26.

What is FSImage in Hadoop?

Ans. **FSImage** is a file in the NameNode that contains the complete state of the file system metadata (e.g., file hierarchy, permissions). It is loaded into memory when the NameNode starts up.

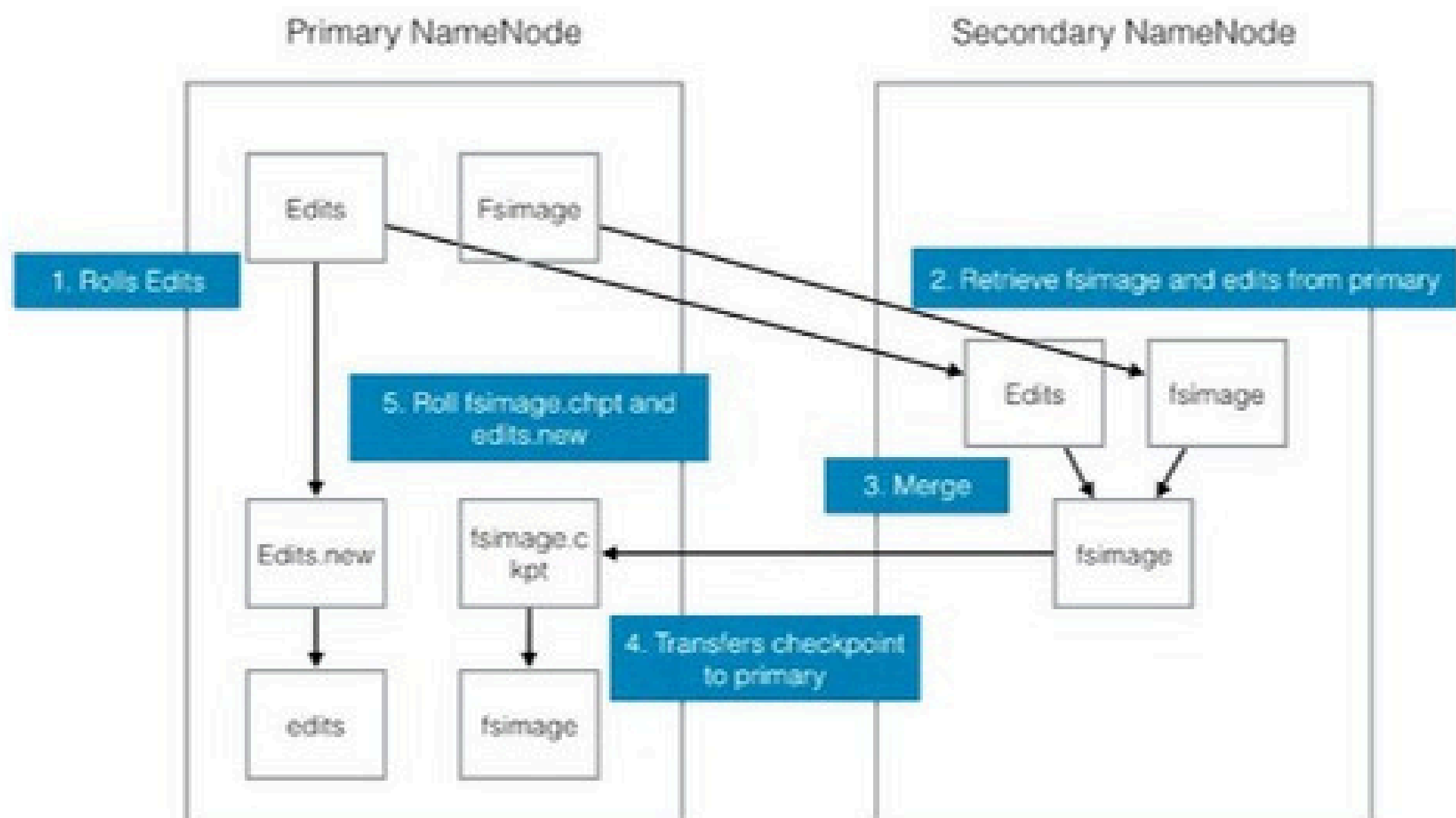


Q27.

What is Checkpointing in Hadoop?

Ans. Checkpointing is the process of merging the **edit logs** with the **FSImage** to produce an updated FSImage.

This helps the NameNode start faster and prevents the edit log from growing too large.

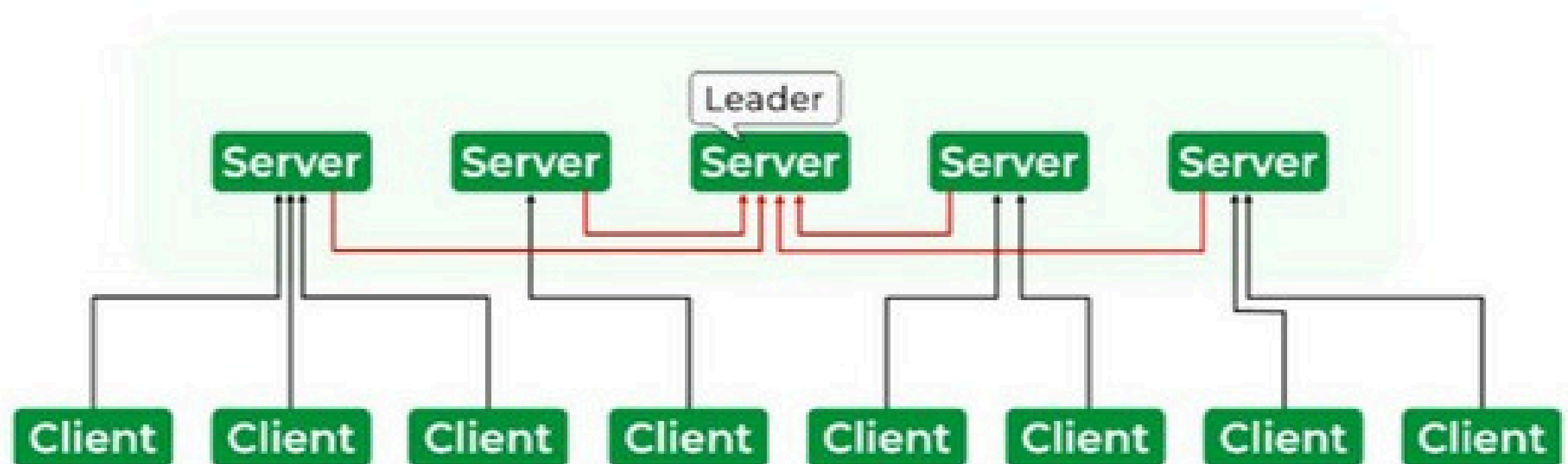


Q28.

What is ZooKeeper, and what role does it play in Hadoop?

Ans. **ZooKeeper** is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and group services.

It is often used in Hadoop for ensuring coordination and providing High Availability (HA) for the NameNode.



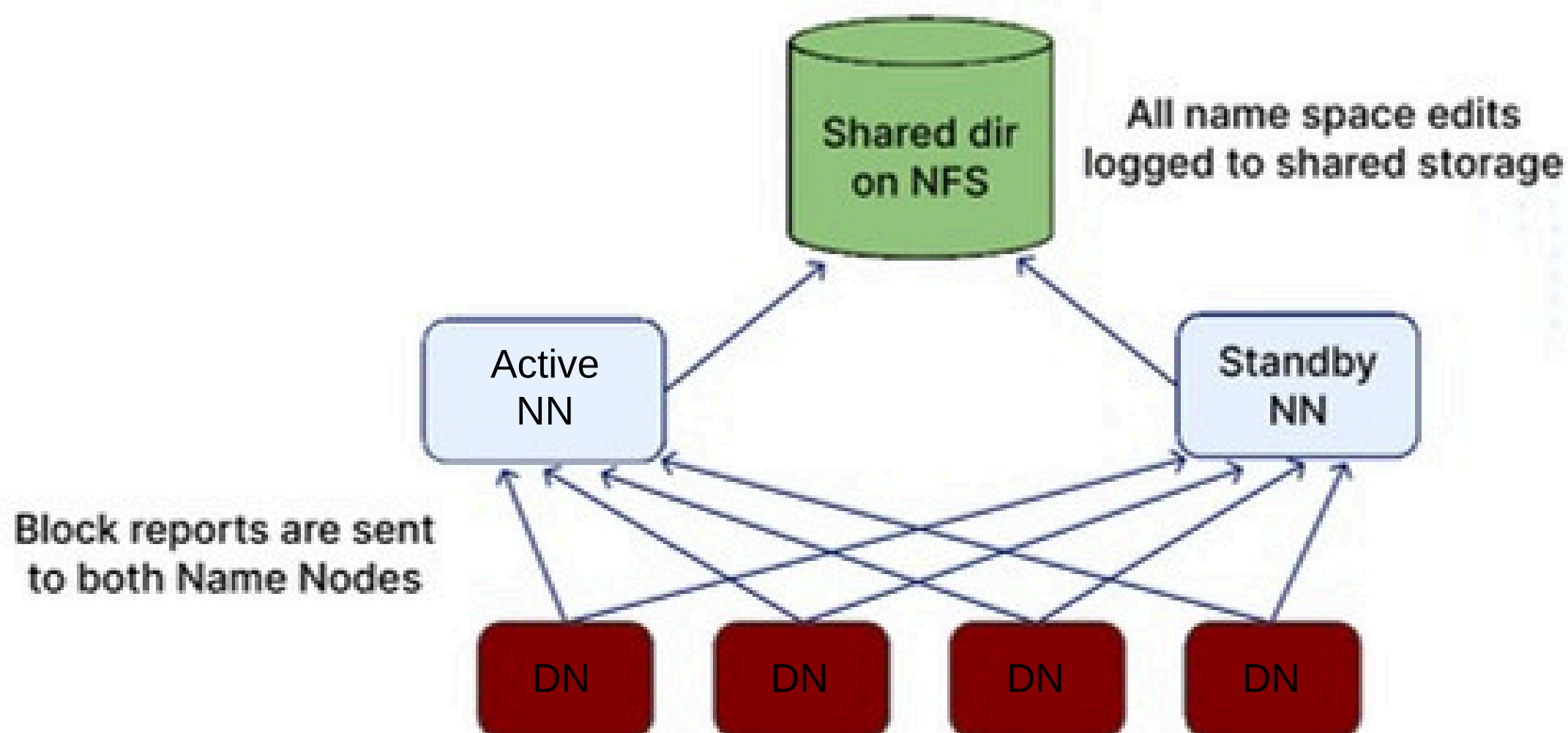
Q29.

Describe Hadoop's High Availability (HA) feature.

Ans. Hadoop's **High Availability** feature ensures that there is no single point of failure for the NameNode. It achieves this by using two NameNodes—an **active NameNode** and a **standby NameNode**—that work in tandem to ensure availability.

Configuration Example: Enabling HA

- This usually involves configuring **JournalNodes** to help synchronize the state between the active and standby NameNodes.



Q30.

What is DistCp in Hadoop, and how is it used?

Ans. DistCp (Distributed Copy) is a tool used to copy large datasets between different clusters or within a cluster, leveraging the MapReduce framework for efficient parallel copying.

Command Example: Using DistCp

```
xml
hadoop
hadoop distcp hdfs://sourceCluster/data hdfs://
targetCluster/data
```

[Copy code](#)

This command copies data from one Hadoop cluster to another.