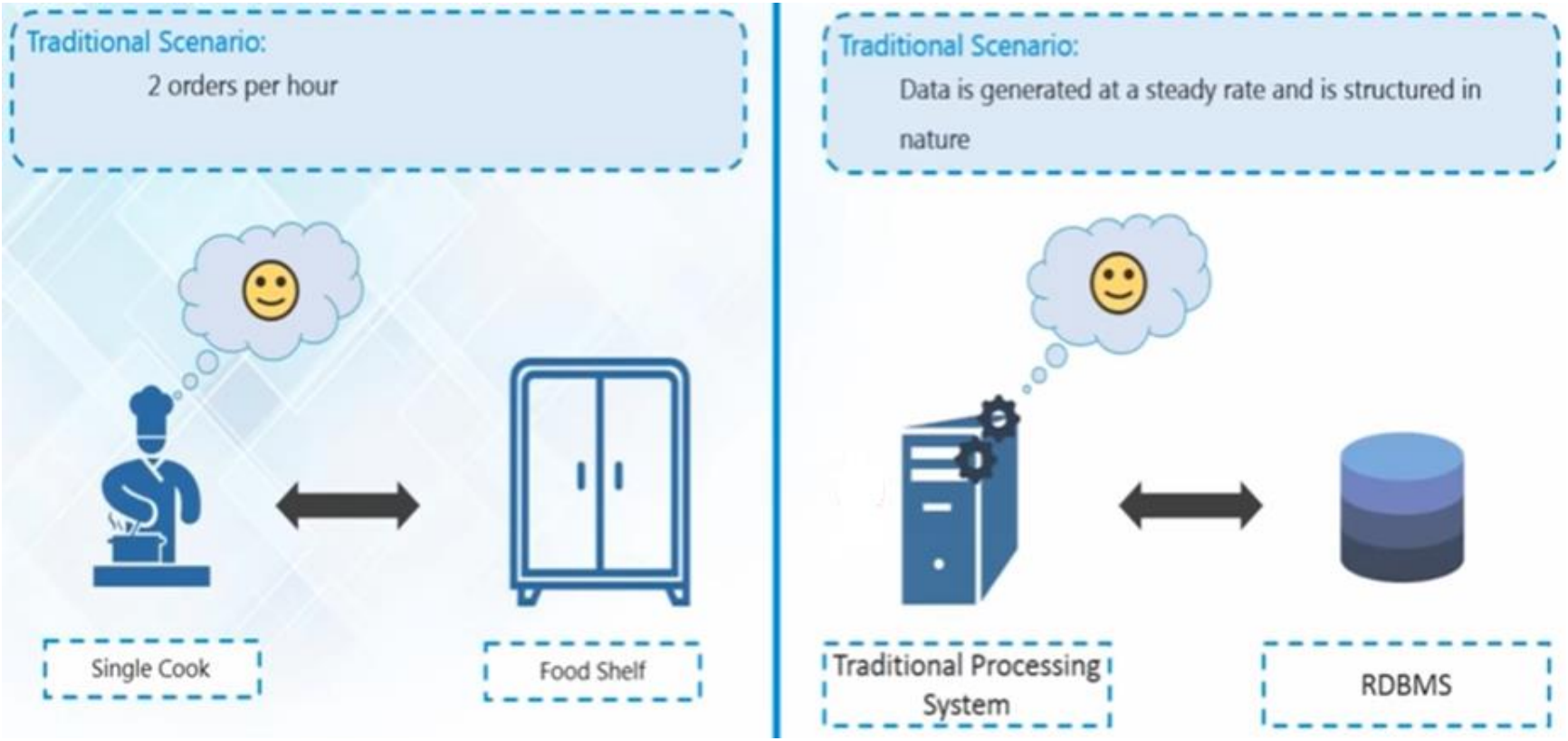
Hadoop

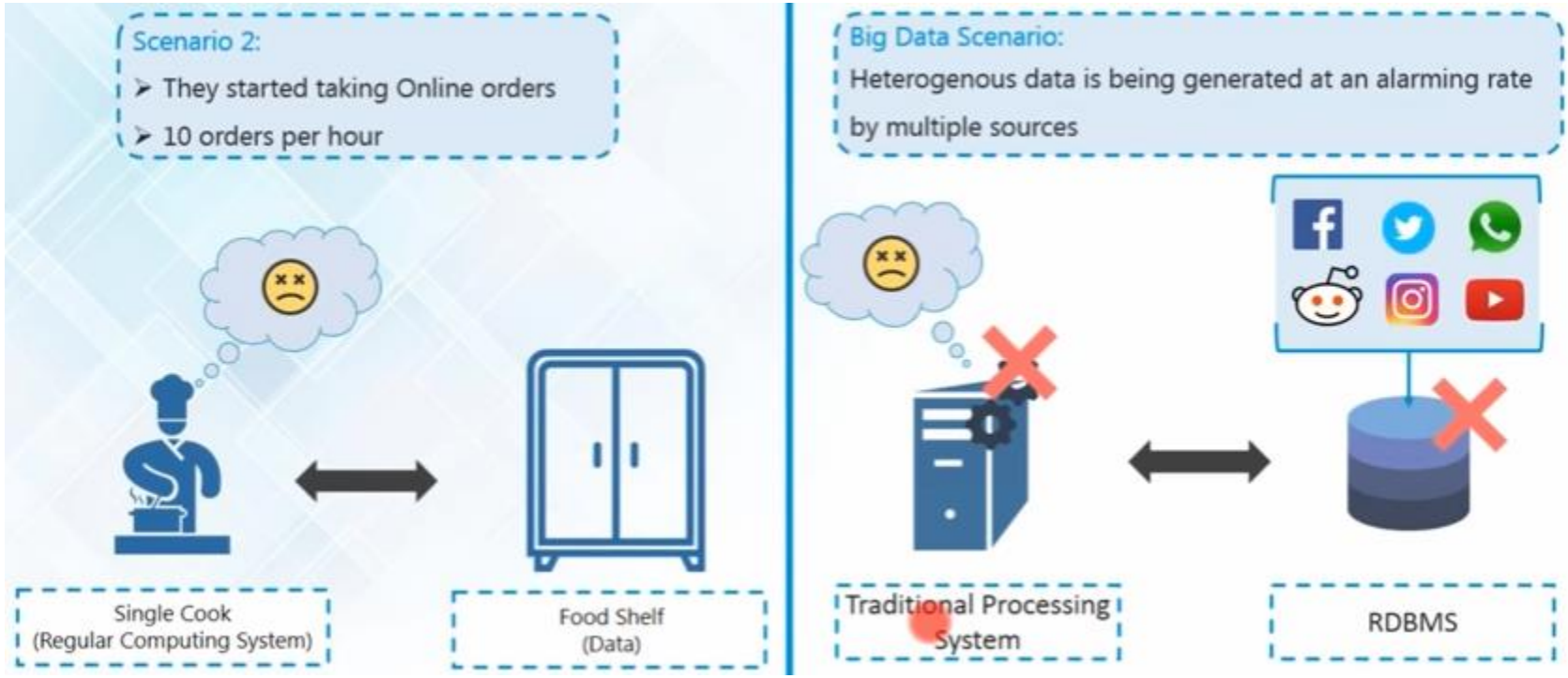
Story of Big Data and Traditional System

Traditional System

Traditional Scenario

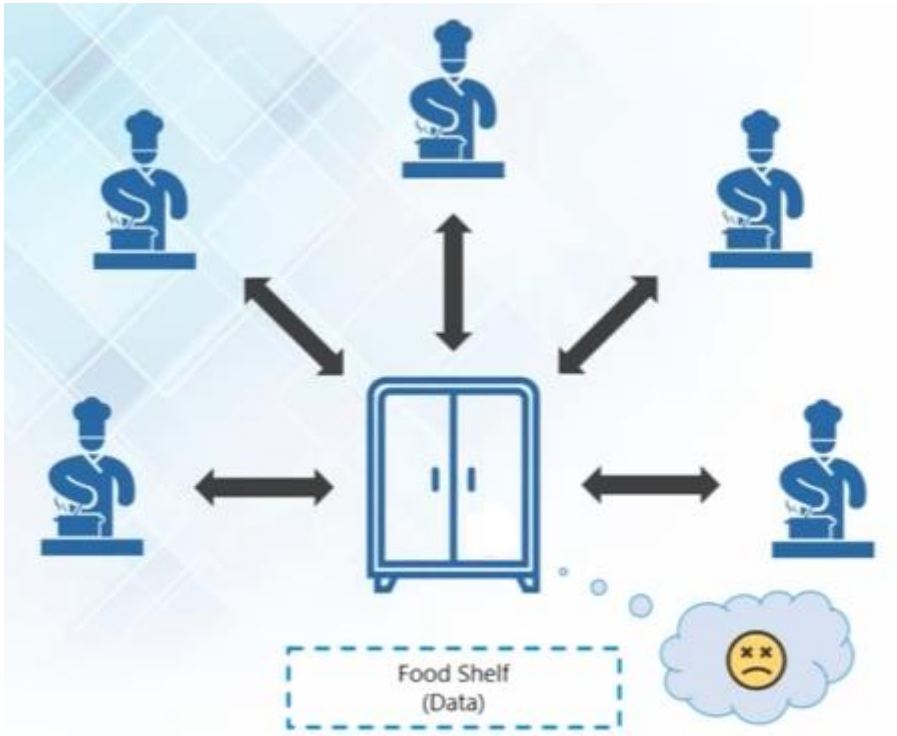


Failure of Traditional System



Issue1 : Too many orders per hour

Solution : Hire multiple cooks



Scenario:

Multiple Cook cooking food

Issue:

Food Shelf becomes the BOTTLENECK

Scenario:

Multiple Processing Unit for data processing

Issue:

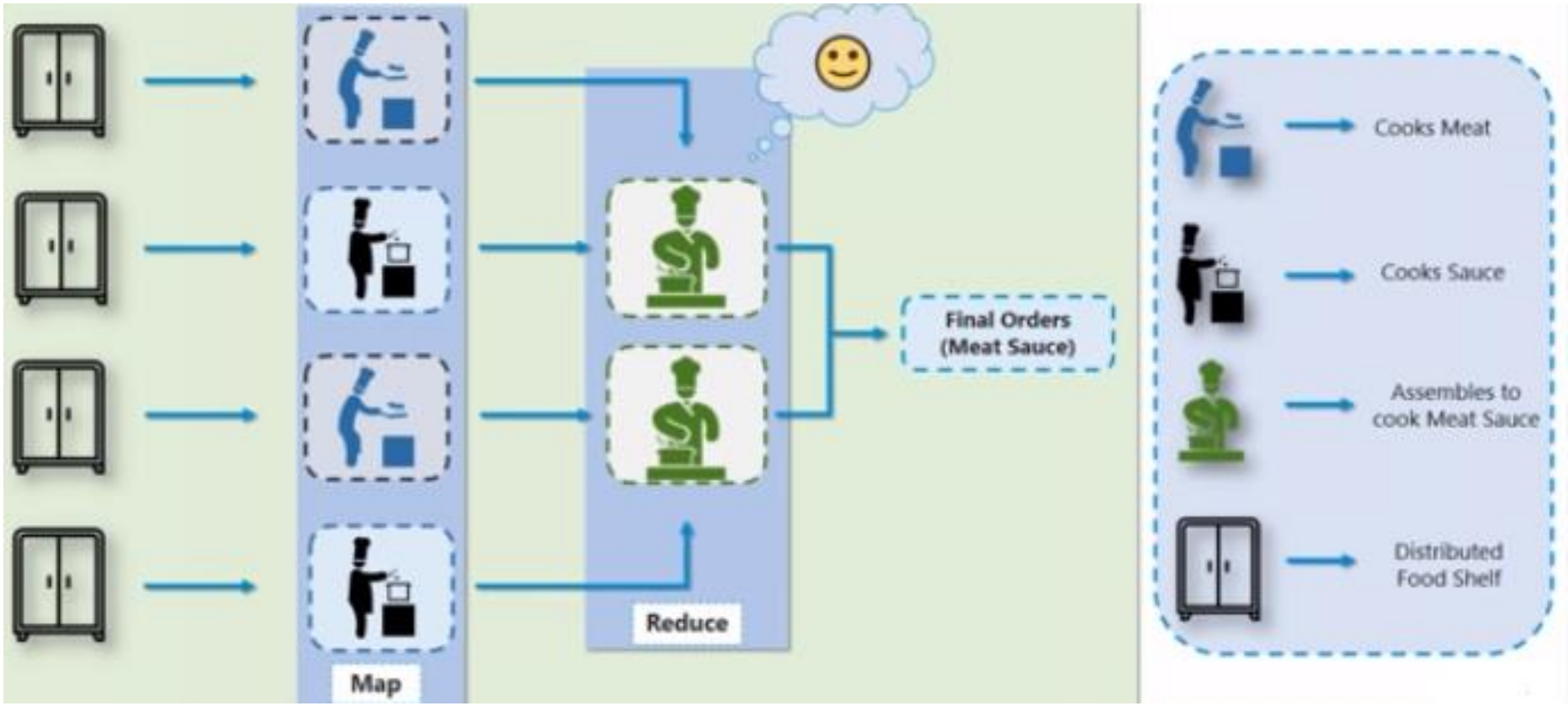
Bringing data to processing generated lots of Network overhead

Data Warehouse

Issue 2 : Food shelf becomes the bottleneck

Solution : Distributed and parallel approach

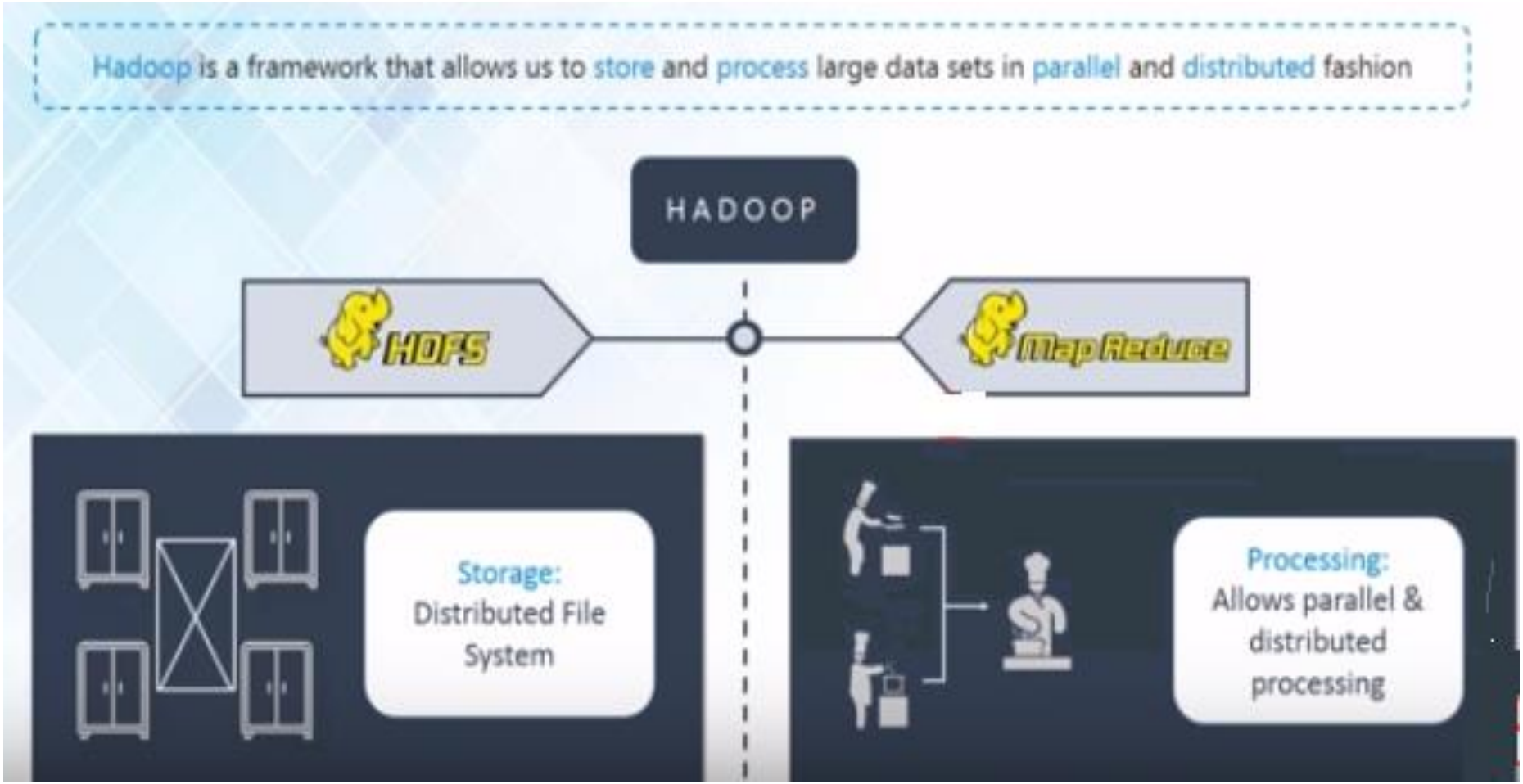
Effective Solution



Need a Framework

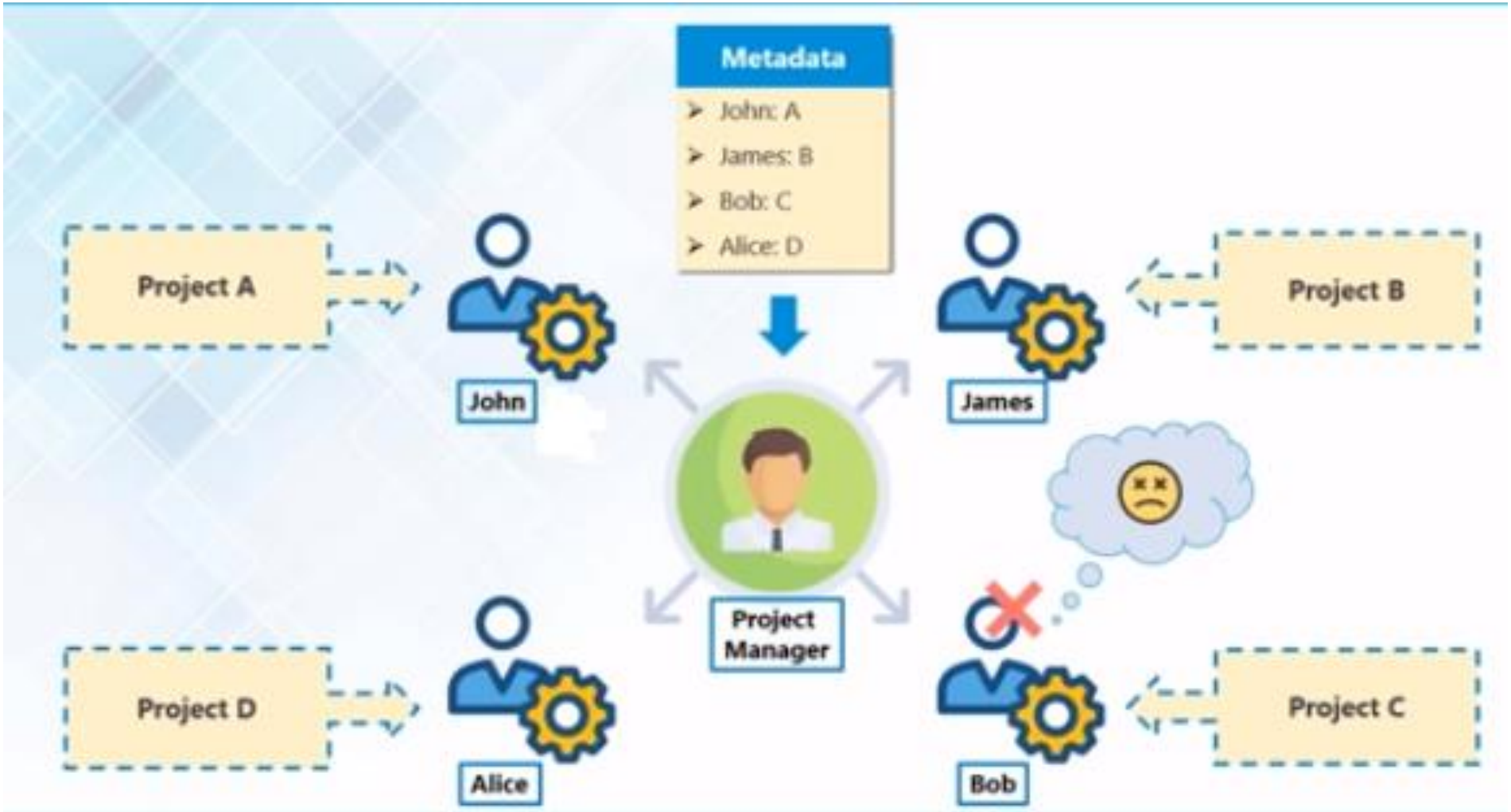
Apache Hadoop:
Framework to process Big data

Apache Hadoop:
Framework to process Big data

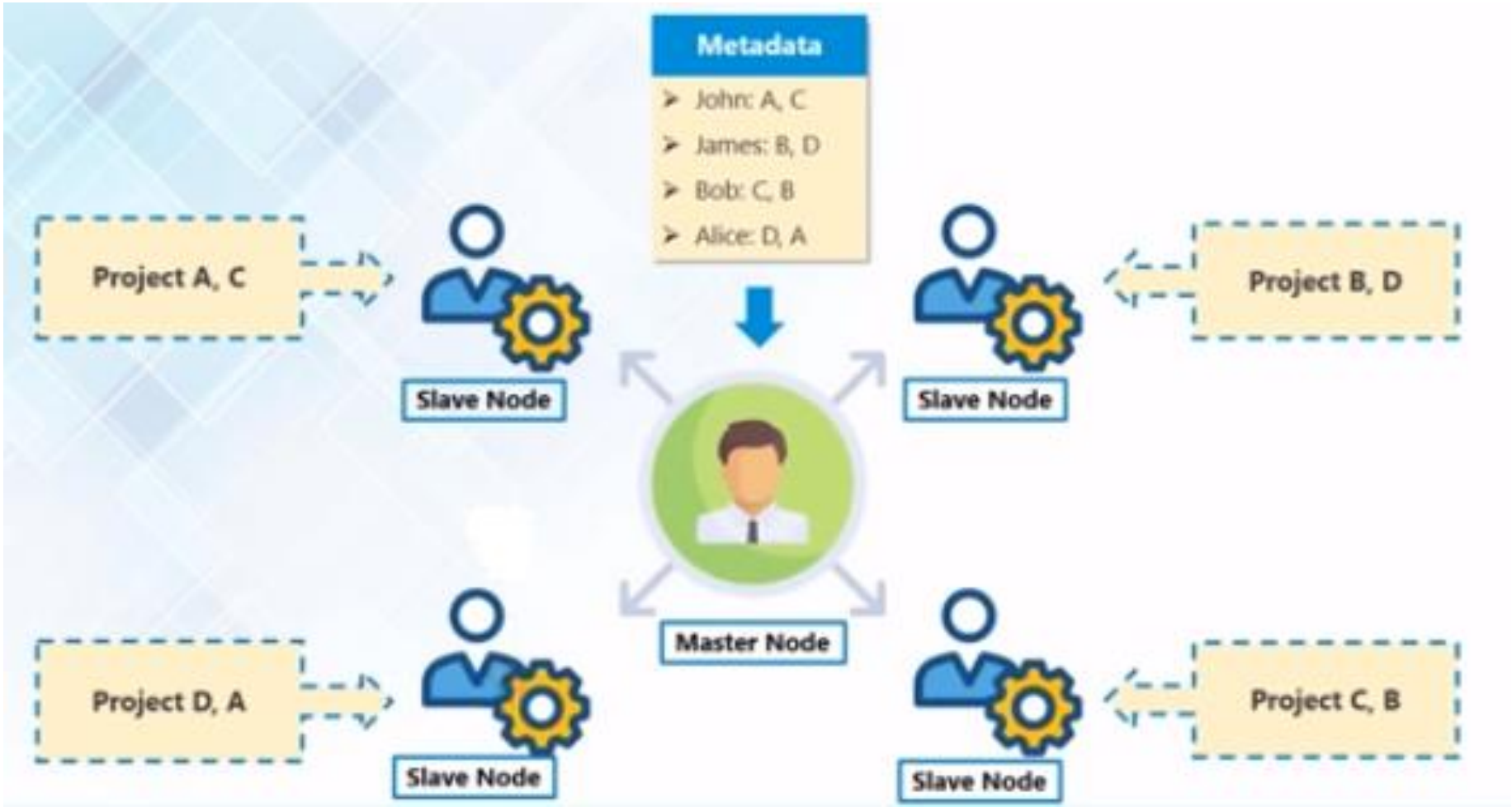


Hadoop: Master/Slave Architecture

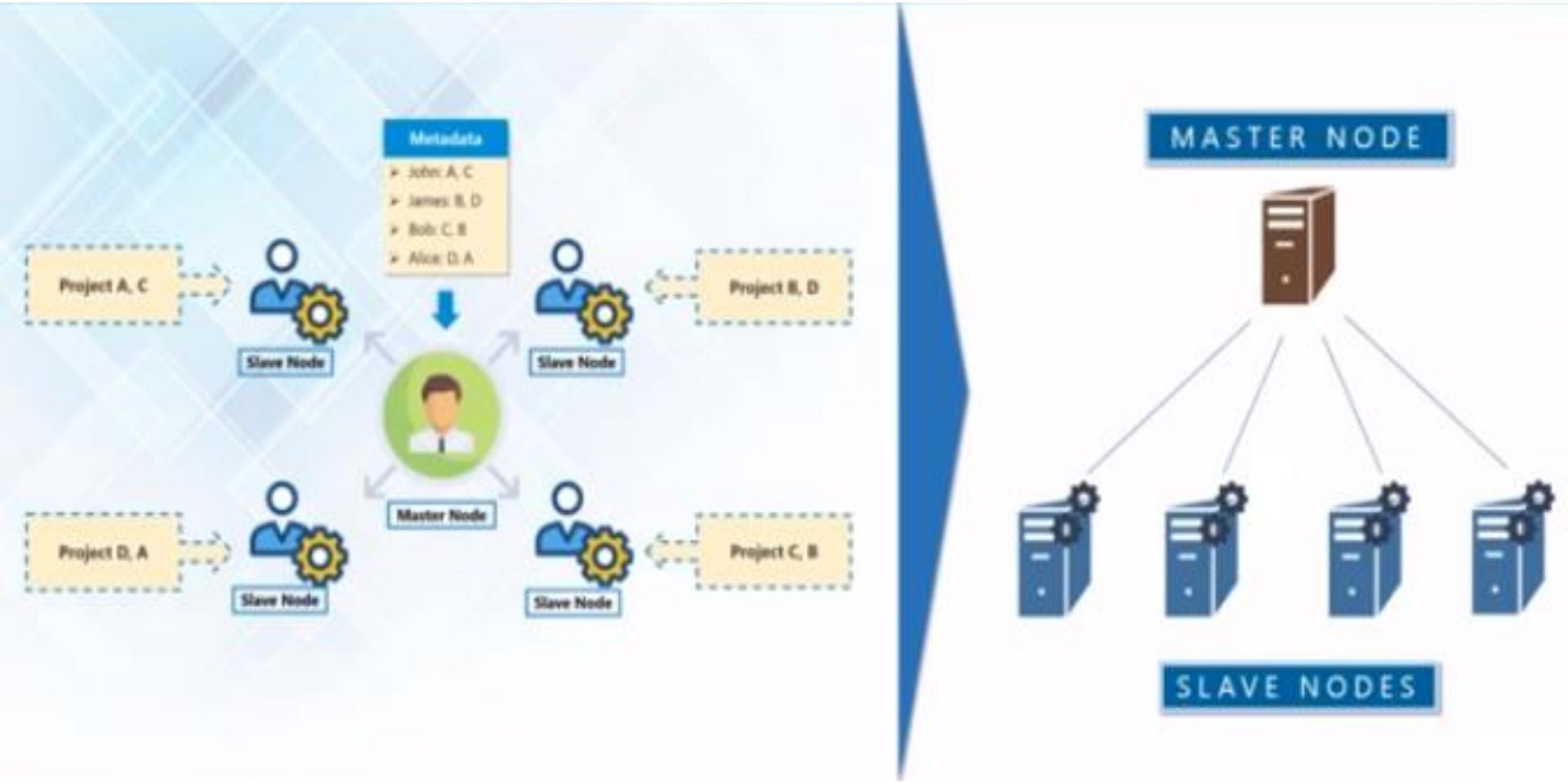
Hadoop: Master/Slave Architecture

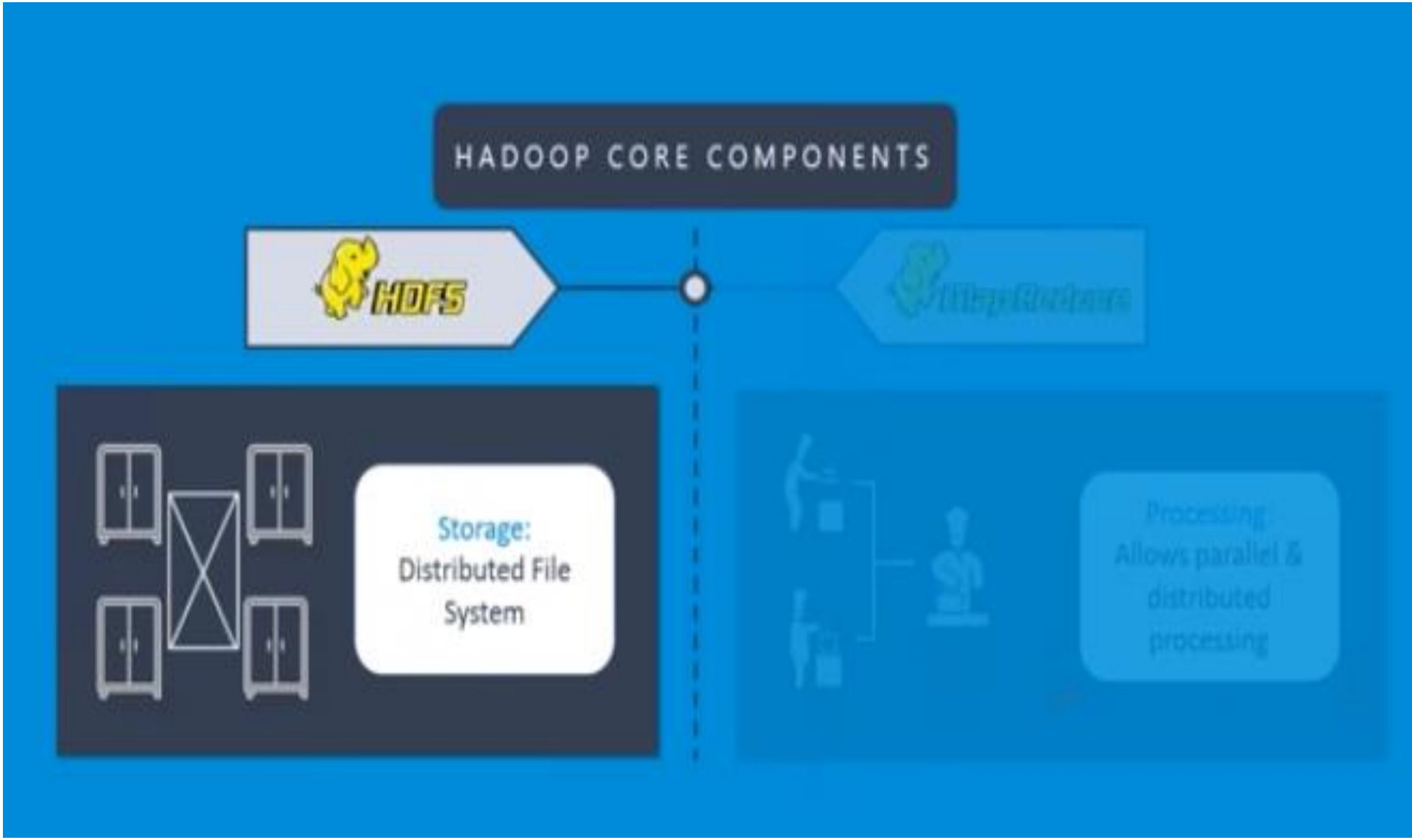


Hadoop: Master/Slave Architecture



Hadoop: Master/Slave Architecture




HADOOP CORE COMPONENTS
HDFS
Storage:
Distributed File
System
Processing:
Allows parallel &
distributed
processing

HDFS Core Components:

01

NameNode

02

DataNode

03

Secondary NameNode

Namenode and Datanode

NameNode:

- Maintains and Manages DataNodes
- Records metadata i.e. information about data blocks e.g. location of blocks stored, the size of the files, permissions, hierarchy, etc.
- Receives heartbeat and block report from all the DataNodes

DataNode:

- Slave daemons
- Stores actual data
- Serves read and write requests from the clients

HDFS Core Components:

01

NameNode

02

DataNode

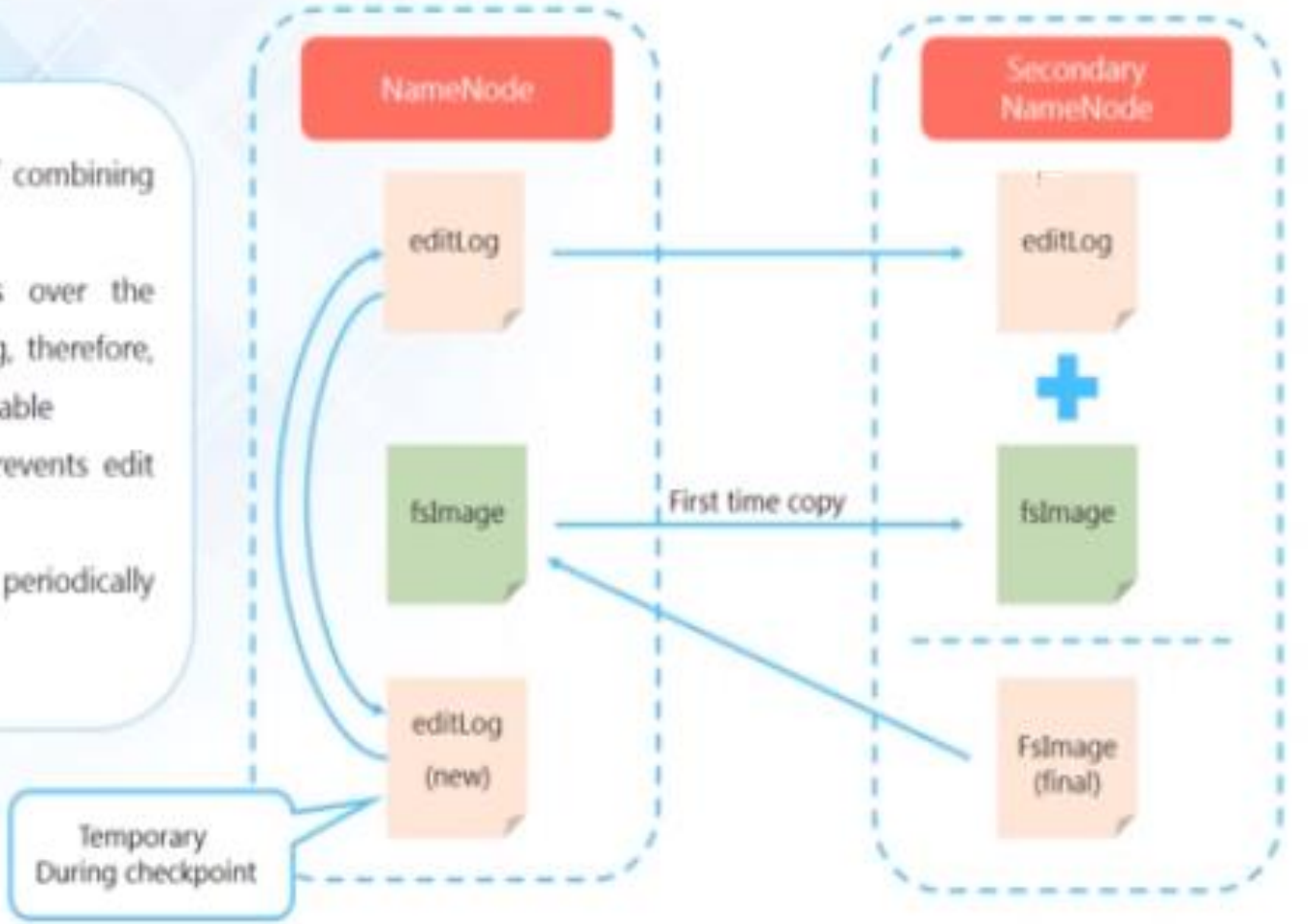
03

Secondary NameNode

Secondary Namenode & Checkpointing

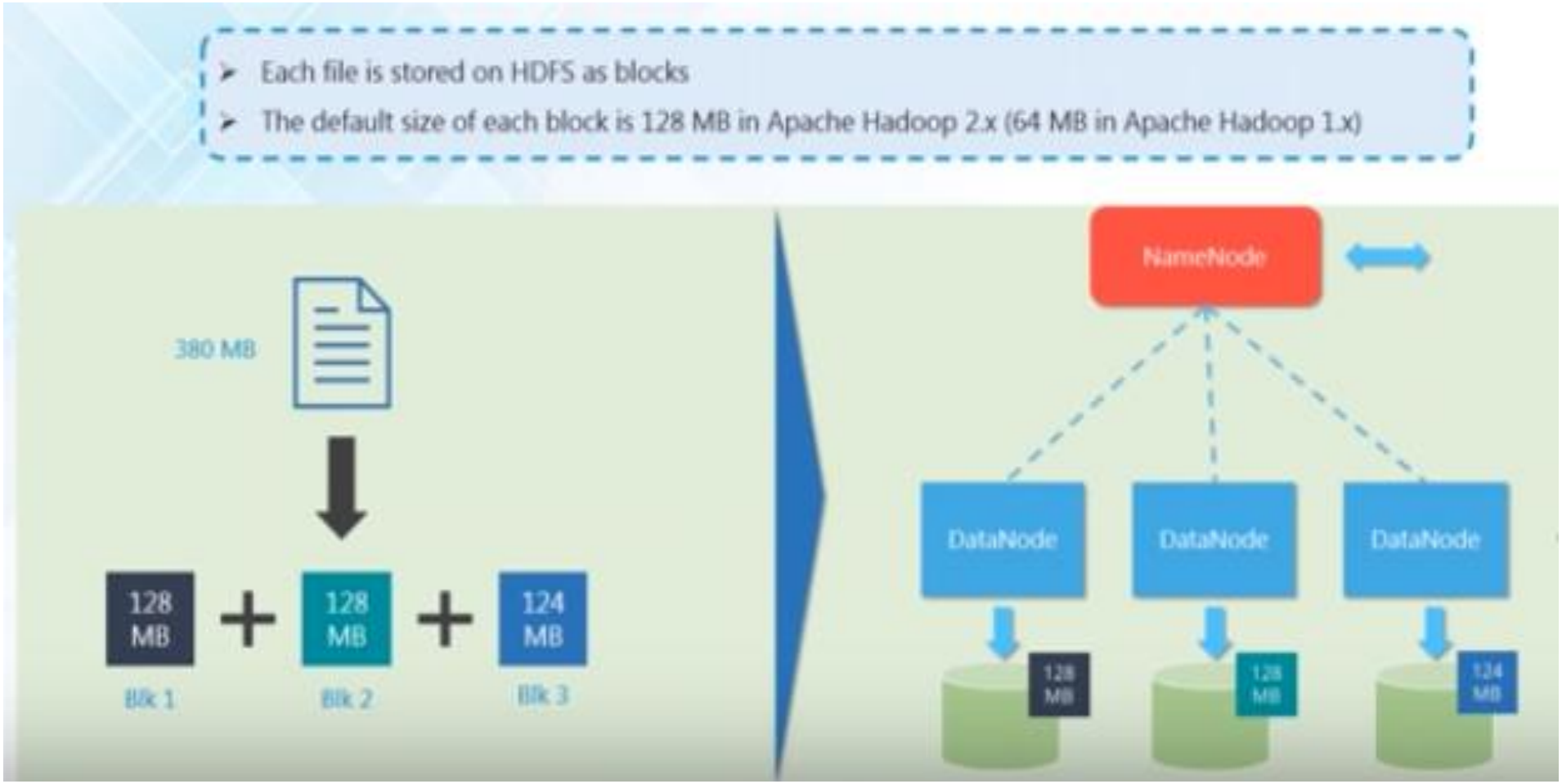
editLog(in RAM) stores recent changes in RAM, fsImage(in Disk) stores all changes in Disk

- Checkpointing is a process of combining edit logs with FsImage
- Secondary NameNode takes over the responsibility of checkpointing, therefore, making NameNode more available
- Allows faster Failover as it prevents edit logs from getting too huge
- Checkpointing happens periodically (default: 1 hour)



How the data is actually stored in Datanodes?
HDFS Data blocks

HDFS Data Blocks

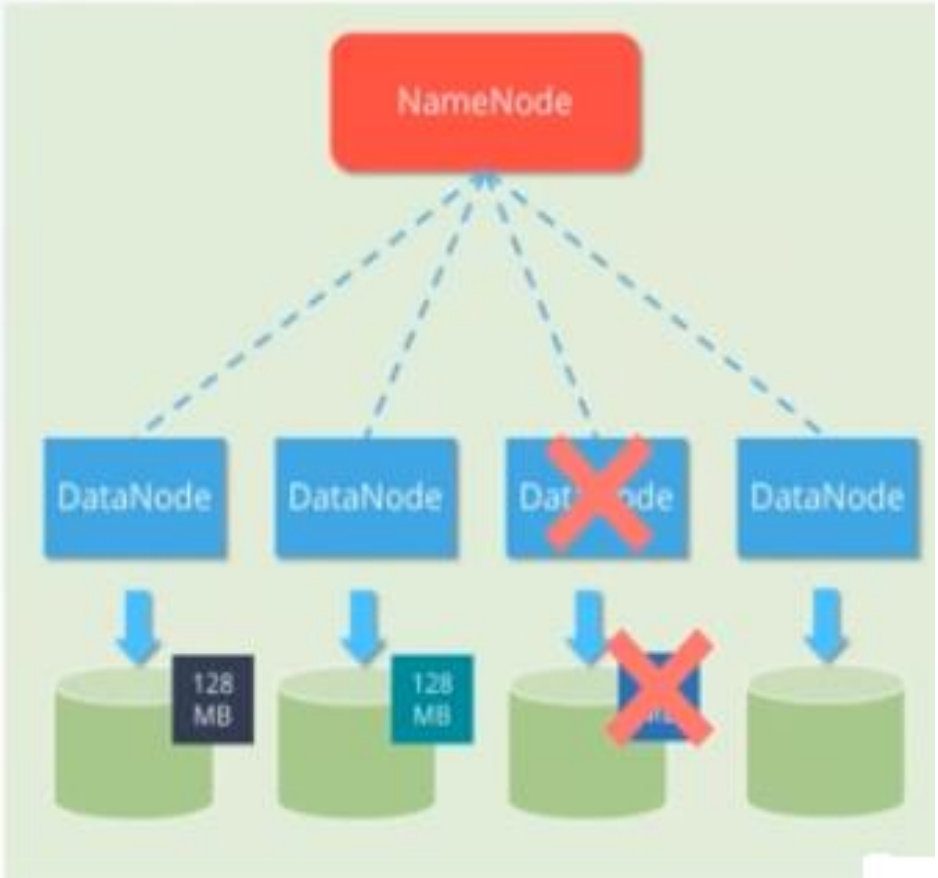


Advantages of HDFS

- If data is more than the total size of a Hadoop cluster, more computers can be added to cluster.
- If a huge data file is stored on a single machine and it takes 4 seconds to process that file, in Hadoop cluster that data file can be divided into cluster and processing time can be reduced.
- Etc.

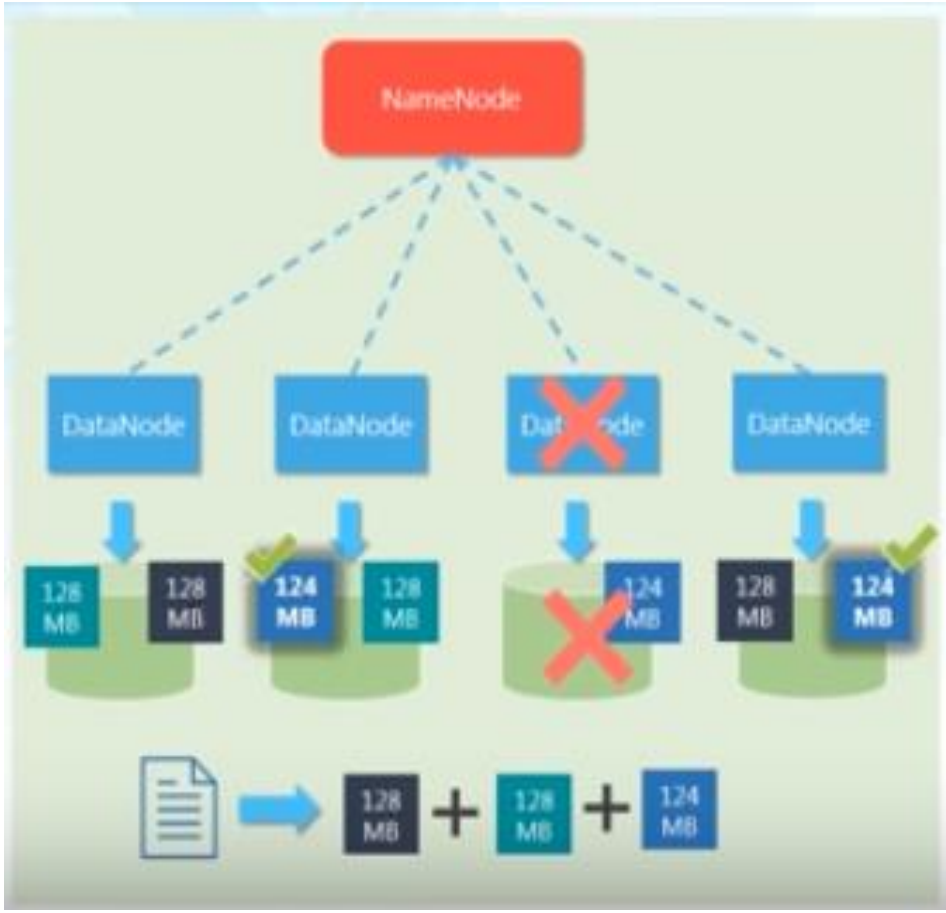
Fault Tolerance: How Hadoop cope up with Datanode failure?

Fault Tolerance



Solution : Replication factor

Fault Tolerance: Replication Factor

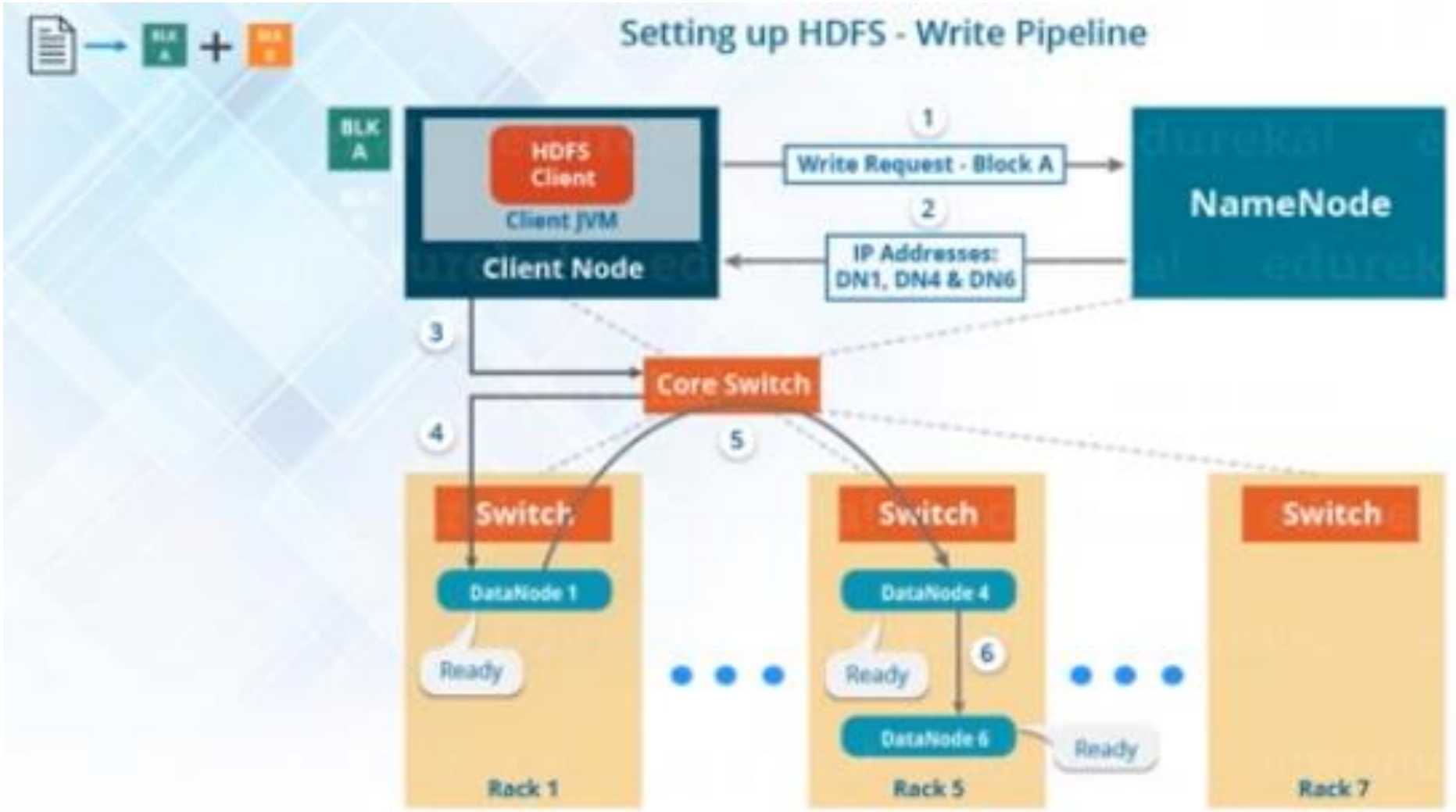


Solution:

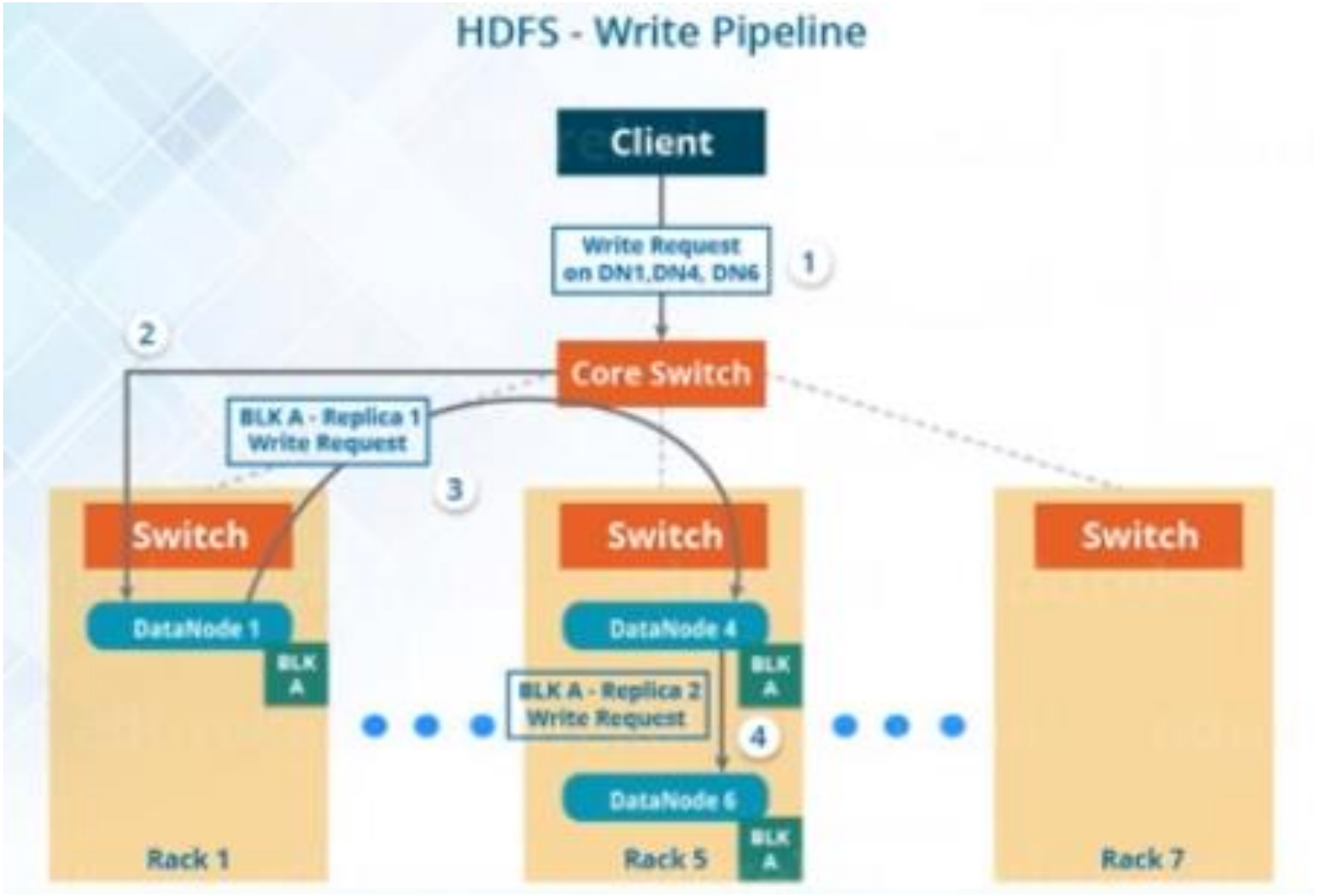
Each data blocks are replicated (thrice by default) and are distributed across different DataNodes

HDFS Write Mechanism

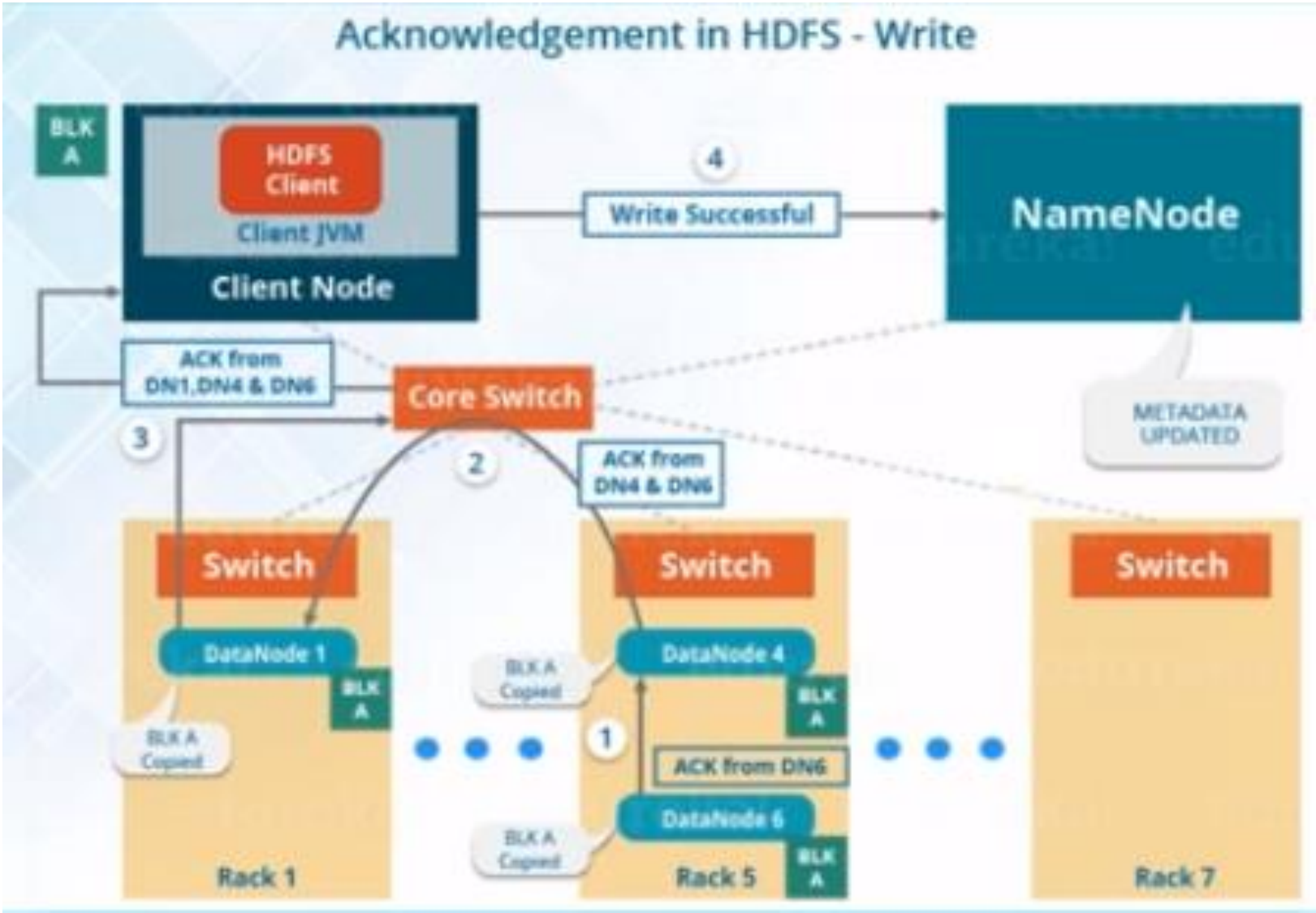
HDFS Write Mechanism – Pipeline Setup



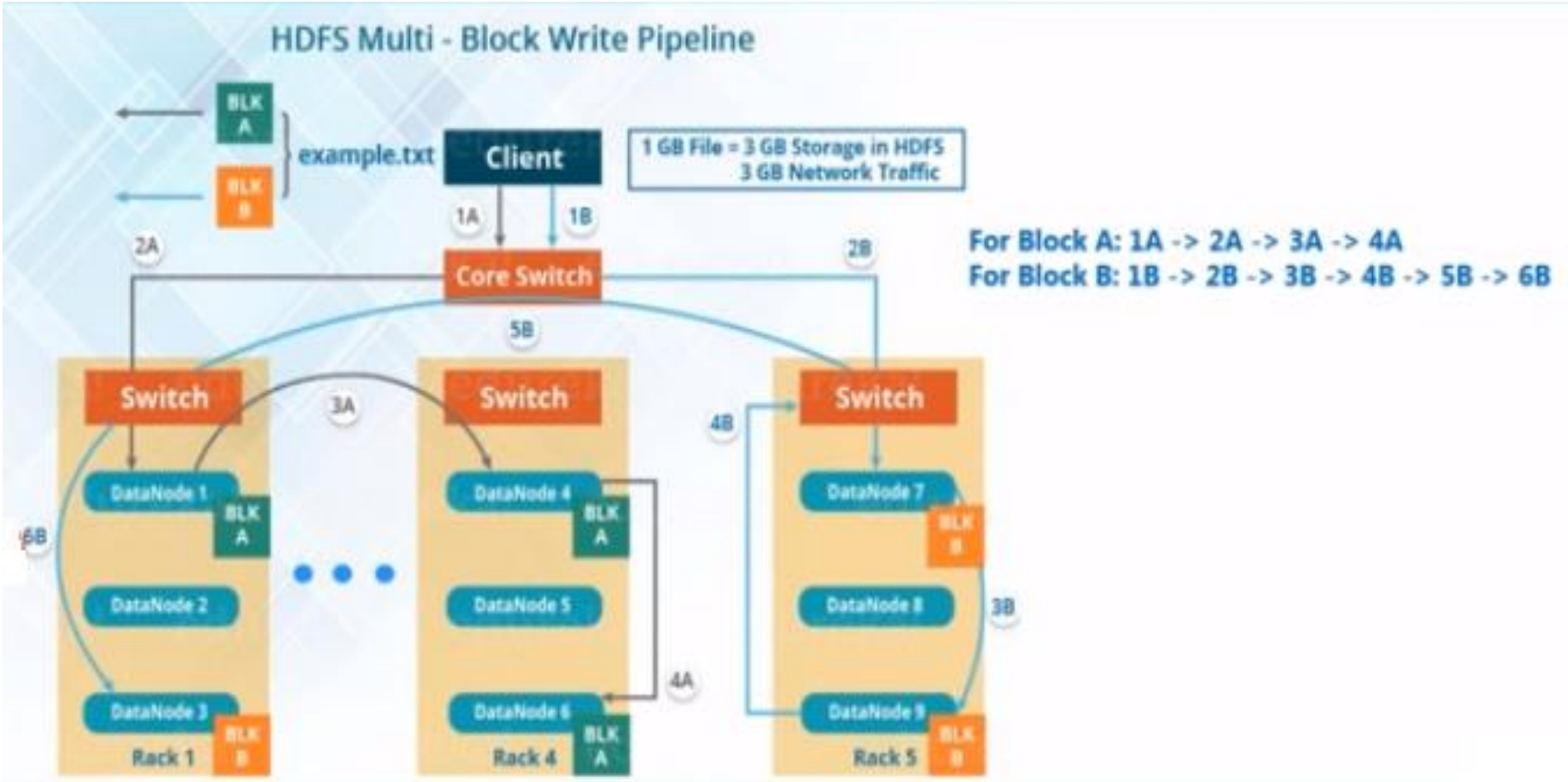
HDFS Write Mechanism – Writing Mechanism



HDFS Write Mechanism – Acknowledgment

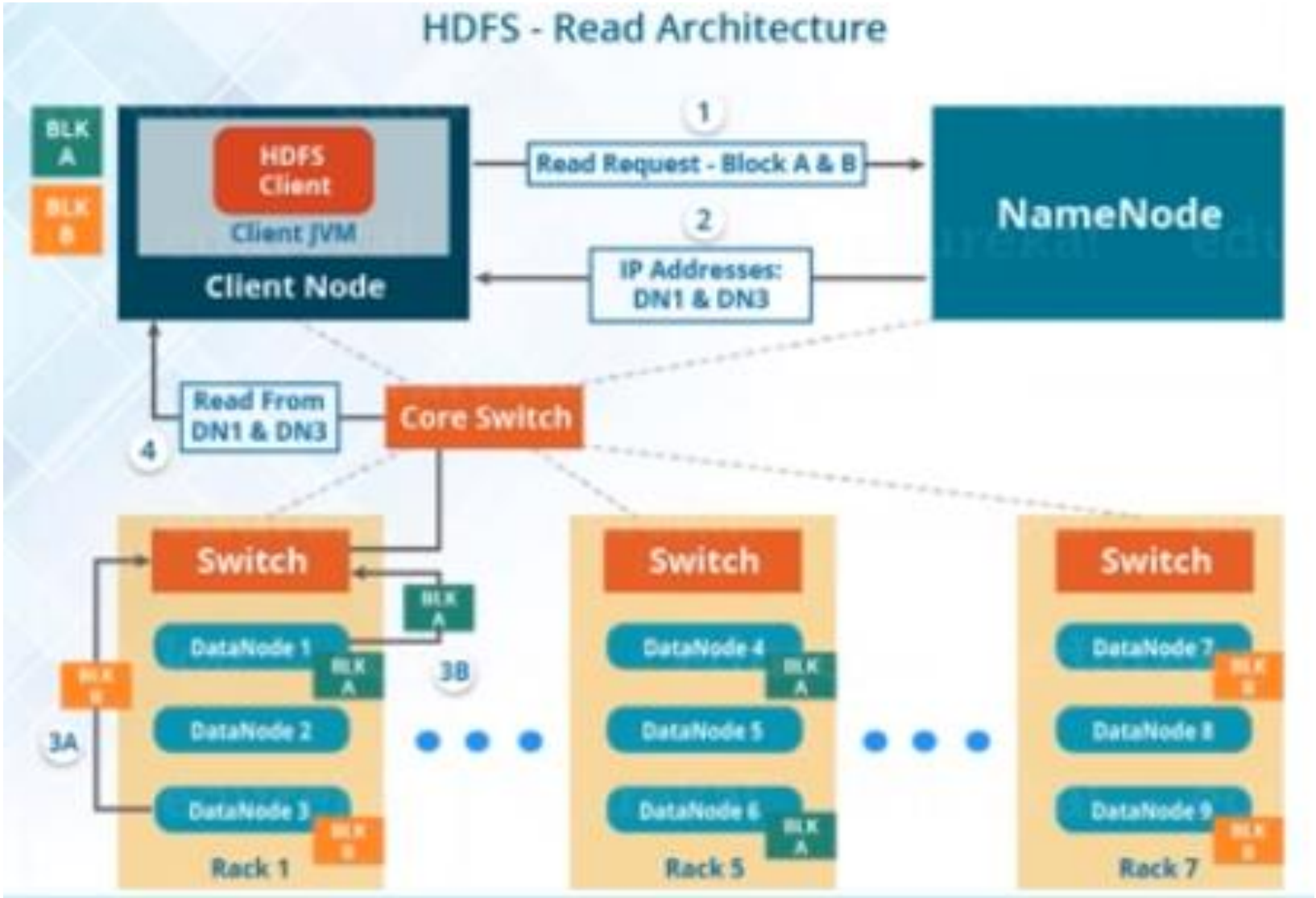


HDFS Multi-block Write Mechanism



HDFS Read Mechanism

HDFS Read Mechanism



HADOOP CORE COMPONENTS

HDFS

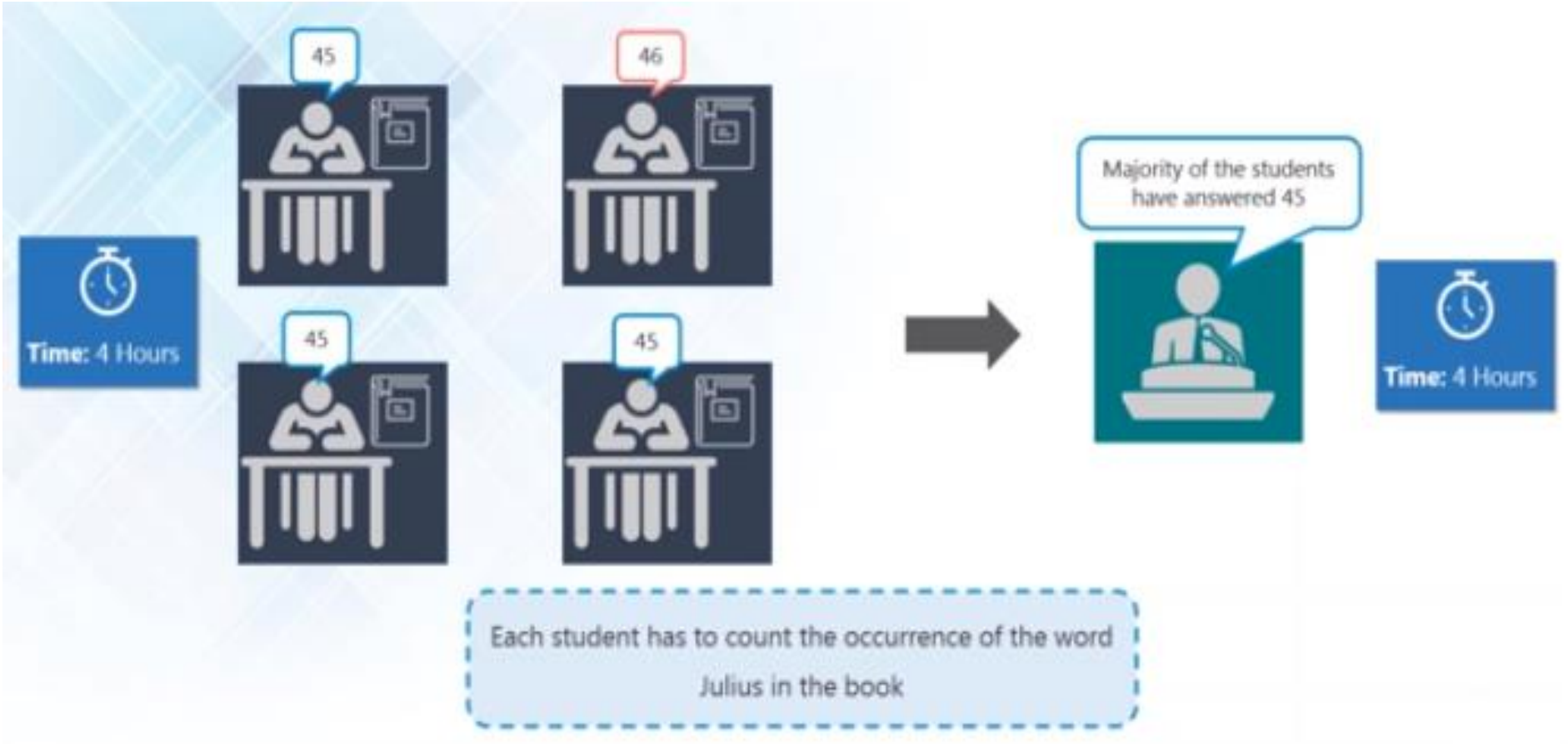
MapReduce

Storage:
Distributed File System

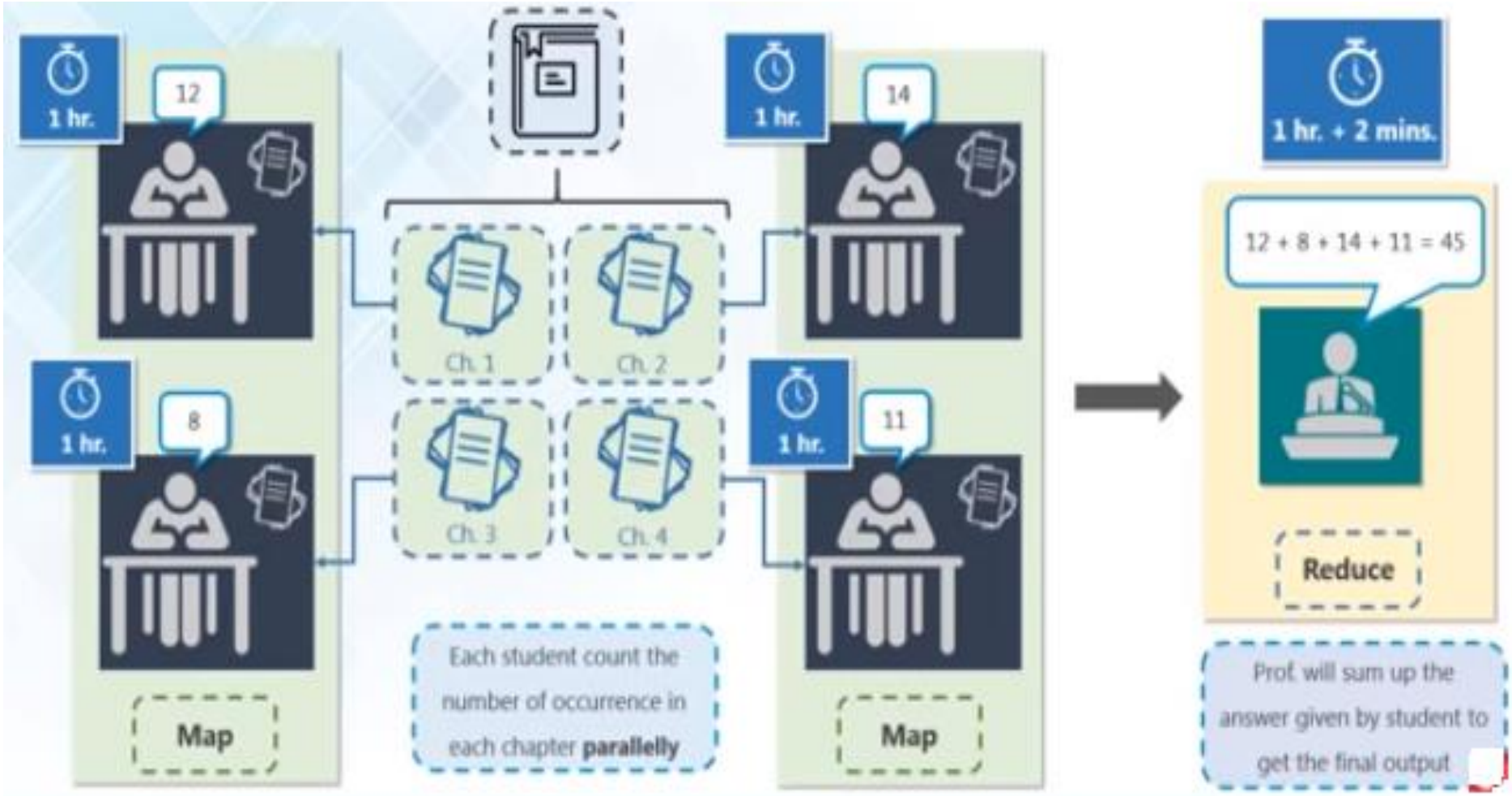
Processing:
Allows parallel & distributed processing

Another Story

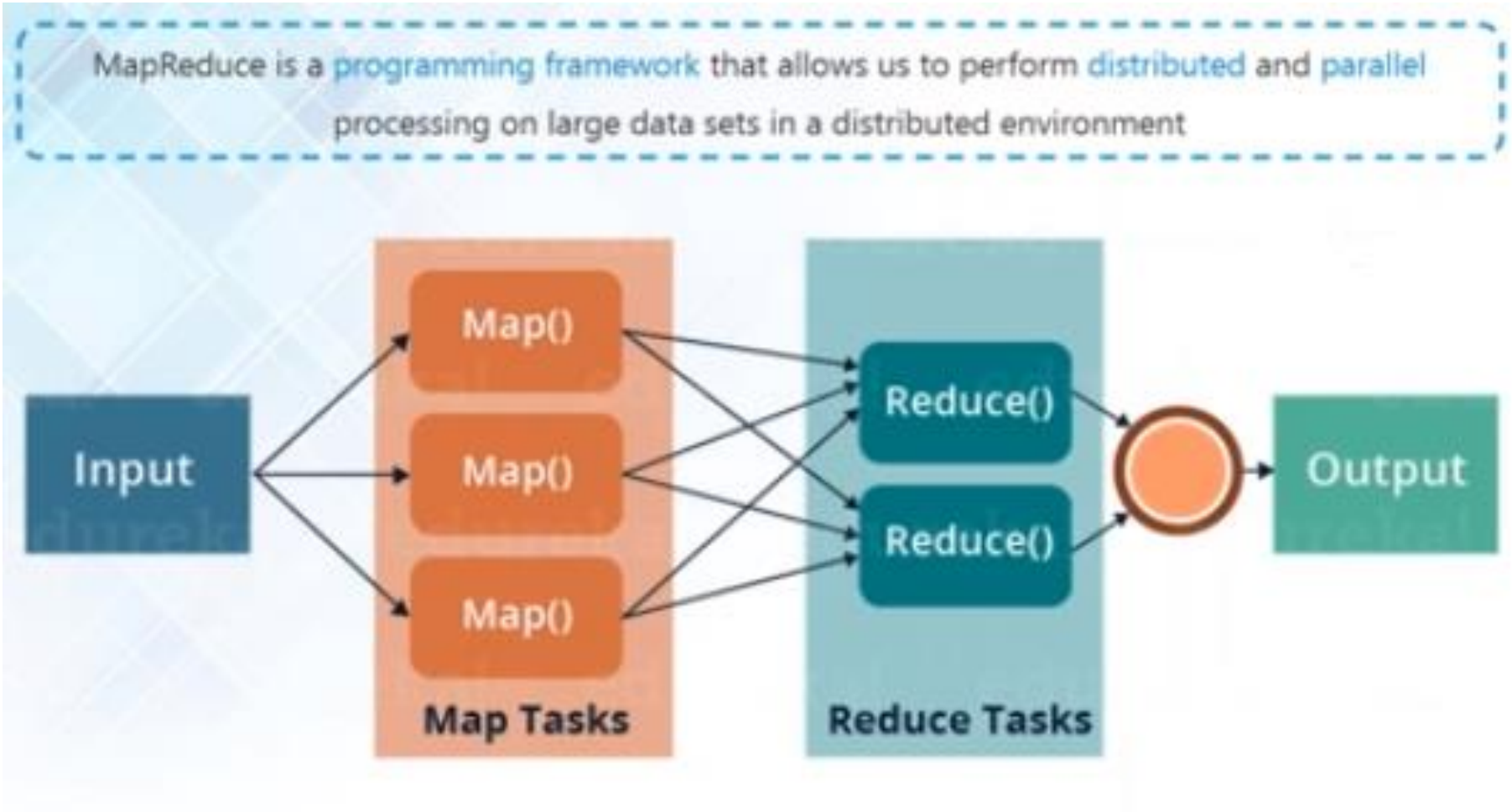
Story of MapReduce



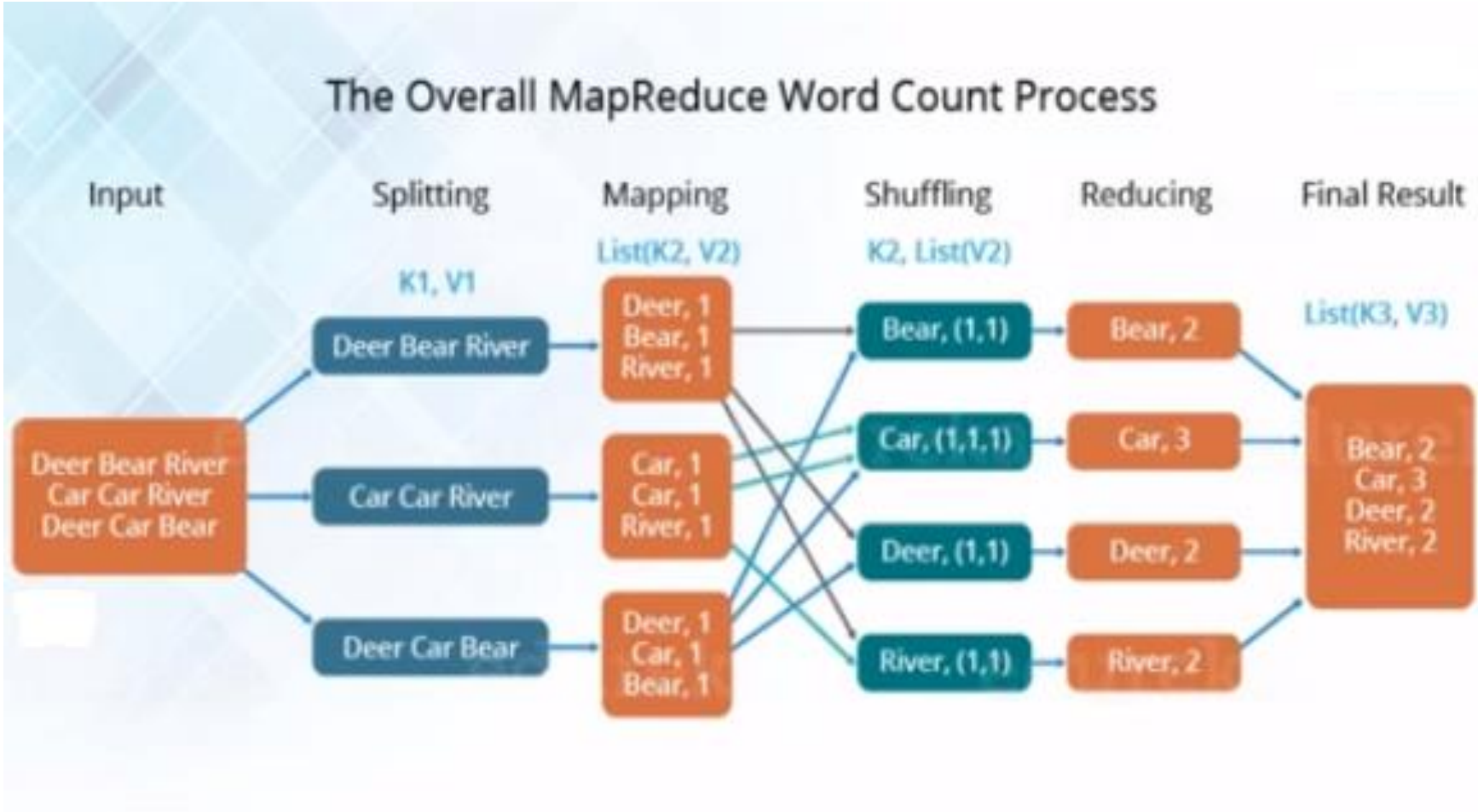
Story of MapReduce



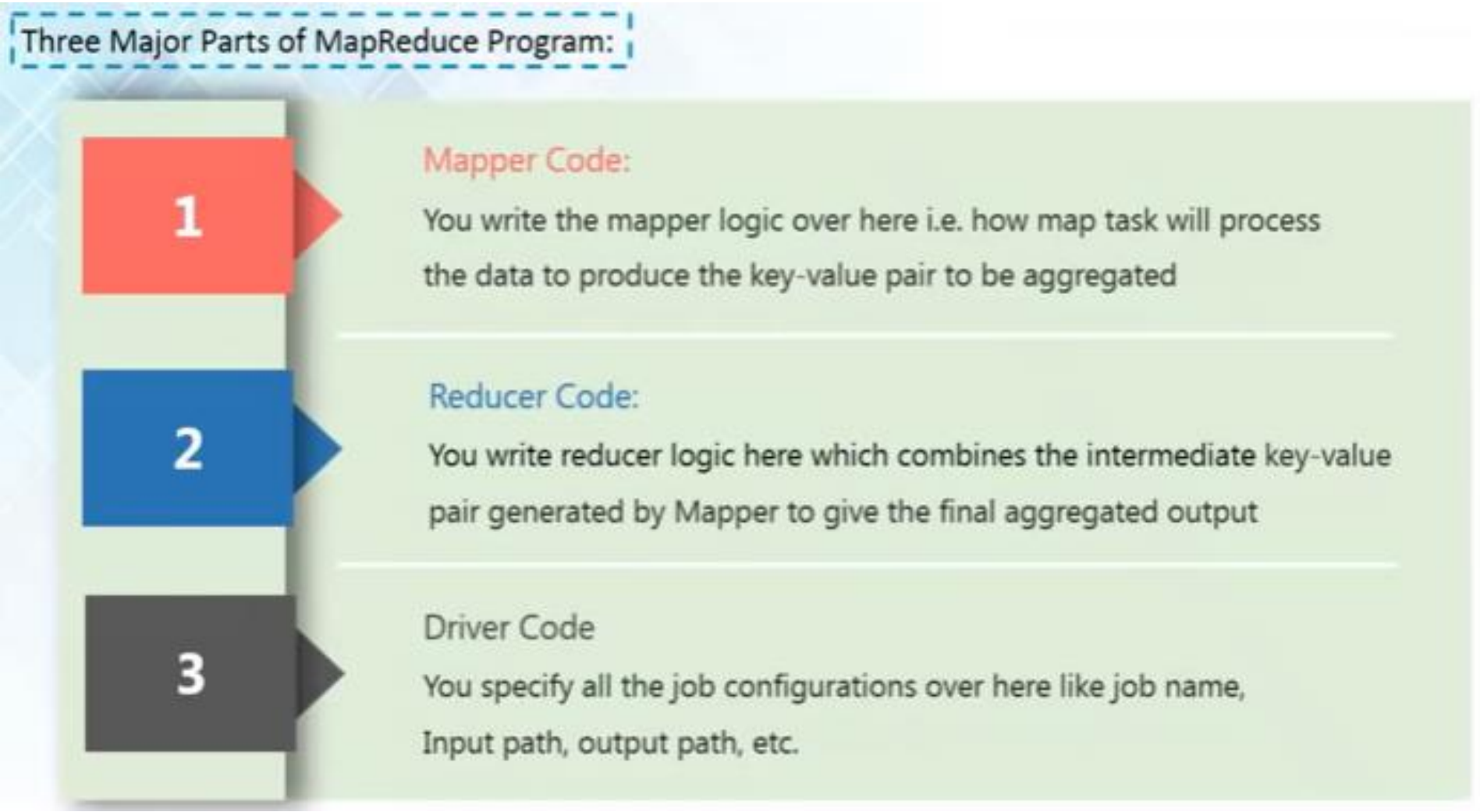
What is MapReduce?



MapReduce Word Count Example



MapReduce Wordcount Example



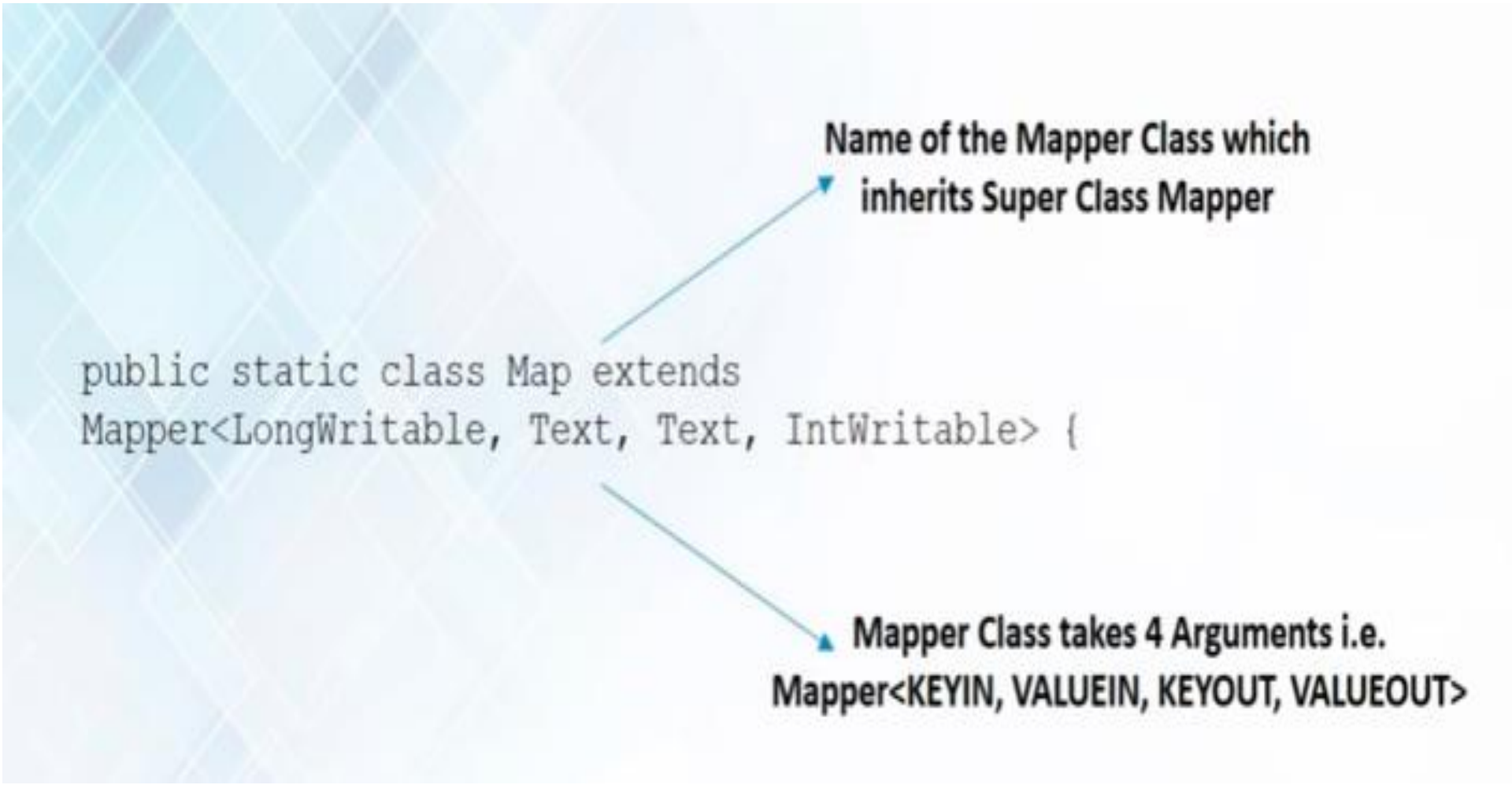
Packages and classes

```
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;

import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import
org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

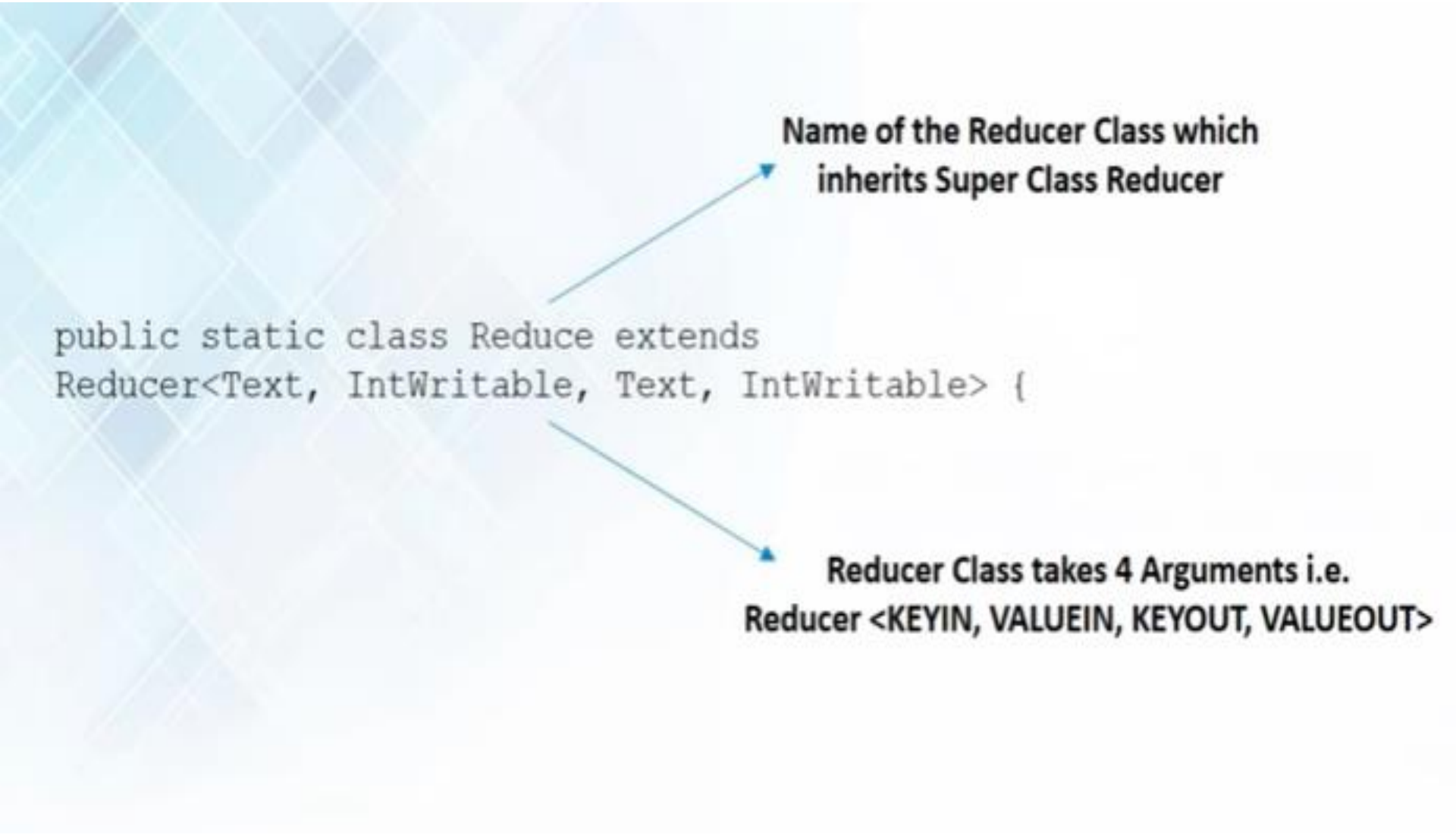
All these packages are present in hadoop-common.jar

All these packages are present in hadoop-mapreduce-client-core.jar

Mapper Class



Reducer Class



Byte Offset

- Byte offset is the number of character that exists counting from the beginning of a line.
- Byte offset is represented hexadecimal.
- E.g. this line “what is byte offset” will have a byte offset of 19. This is used as key value in hadoop
- Key = byte offset, value = line

Mapper Code

Byte Offset Type

Mapper Value Input Type

Mapper Key Output Type

Mapper Value Output Type

```
public static class Map extends Mapper<LongWritable,Text,Text,IntWritable> {

public void map(LongWritable key, Text value, Context context) throws IOException,InterruptedException {

String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
value.set(tokenizer.nextToken());
context.write(value, new IntWritable(1));
}
```

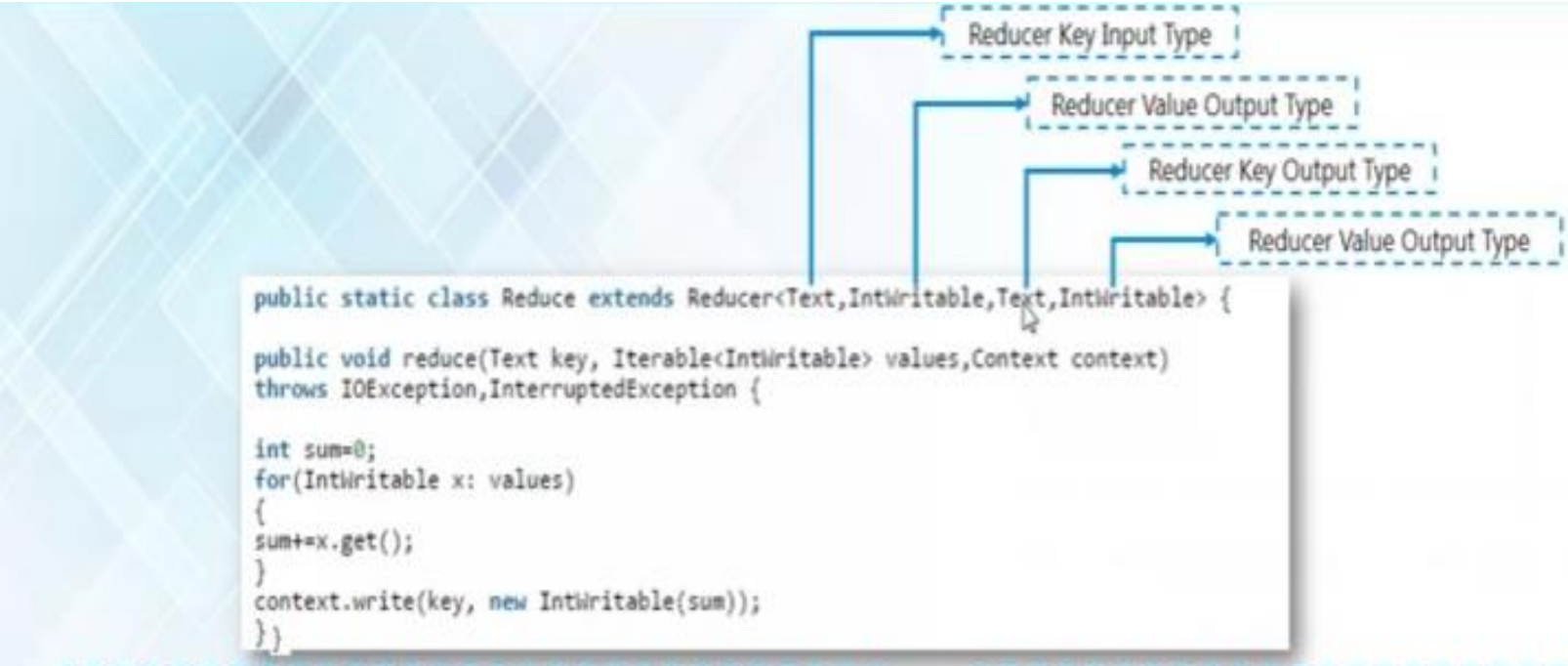
Mapper Input:

- The key is nothing but the offset of each line in the text file: *LongWritable*
- The value is each individual: *Text*

Mapper Output:

- The key is the tokenized words: Text
- We have the hardcoded value in our case which is 1: IntWritable
- Example – Dear 1, Bear 1, etc.

Reducer Code



Reducer Input:

- Keys are unique words which have been generated after the sorting and shuffling phase: Text
- The value is a list of integers corresponding to each key: IntWritable
- Example: Bear, [1, 1], etc.

Reducer Output:

- The key is all the unique words present in the input text file: Text
- The value is the number of occurrences of each of the unique words: IntWritable
- Example: Bear, 2; Car, 3, etc. .

Driver Code

In the driver class, we set the configuration of our MapReduce job to run in Hadoop

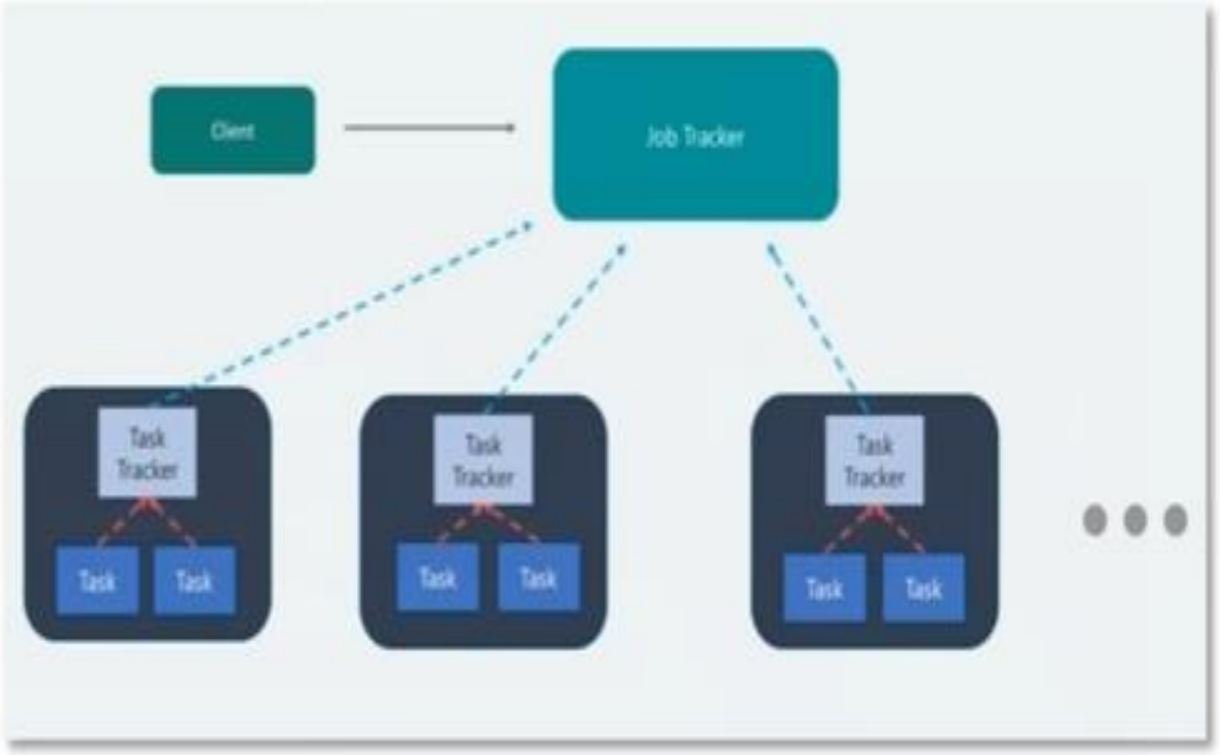
```
Configuration conf= new Configuration();
Job job = new Job(conf,"My Word Count Program");
job.setJarByClass(WordCount.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
Path outputPath = new Path(args[1]);

//Configuring the input/output path from the filesystem into the job
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

- Specify the name of the job , the data type of input/output of the mapper and reducer
- Specify the names of the mapper and reducer classes.
- Path of the input and output folder
- The method setInputFormatClass () is used for specifying the unit of work for mapper
- Main() method is the entry point for the driver

MapReduce Word Count Example

Hadoop 1.x

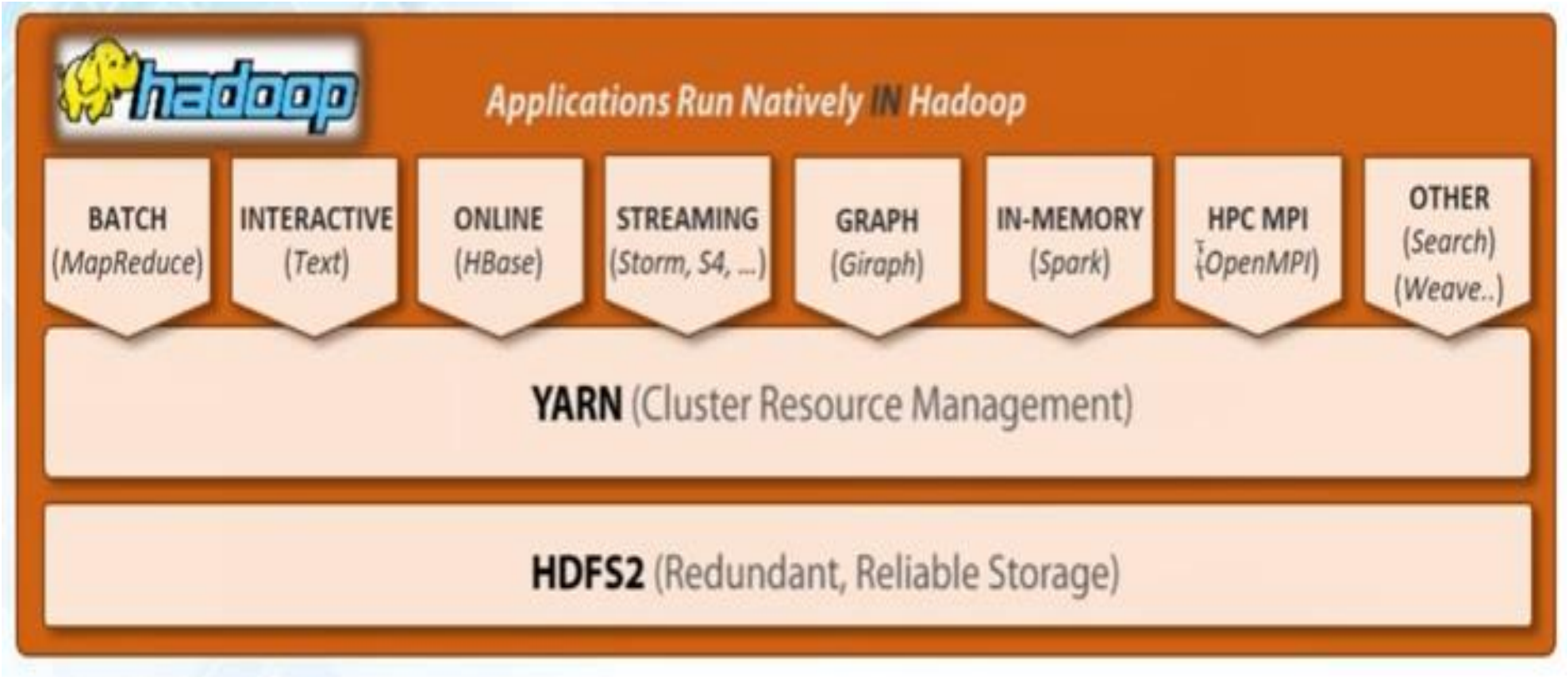

Hadoop 1.x
MapReduce
HDFS


Client
Job Tracker
Task Tracker
Task
Task
Task Tracker
Task
Task
Task Tracker
Task
Task

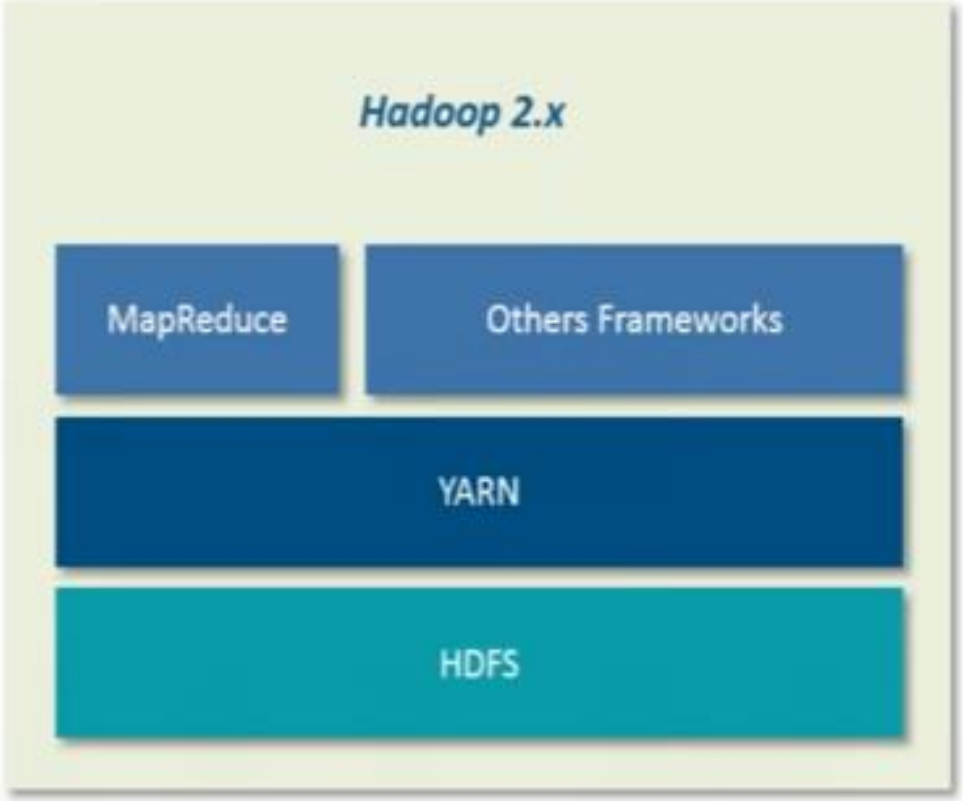
Limitations of JobTraker

- Scalability
- Need for other programming framework
- Increasing processing power, but could not utilize it fully
- Need for real-time or near real-time processing

YARN – Moving beyond MapReduce



Hadoop 1.x vs Hadoop 2.x



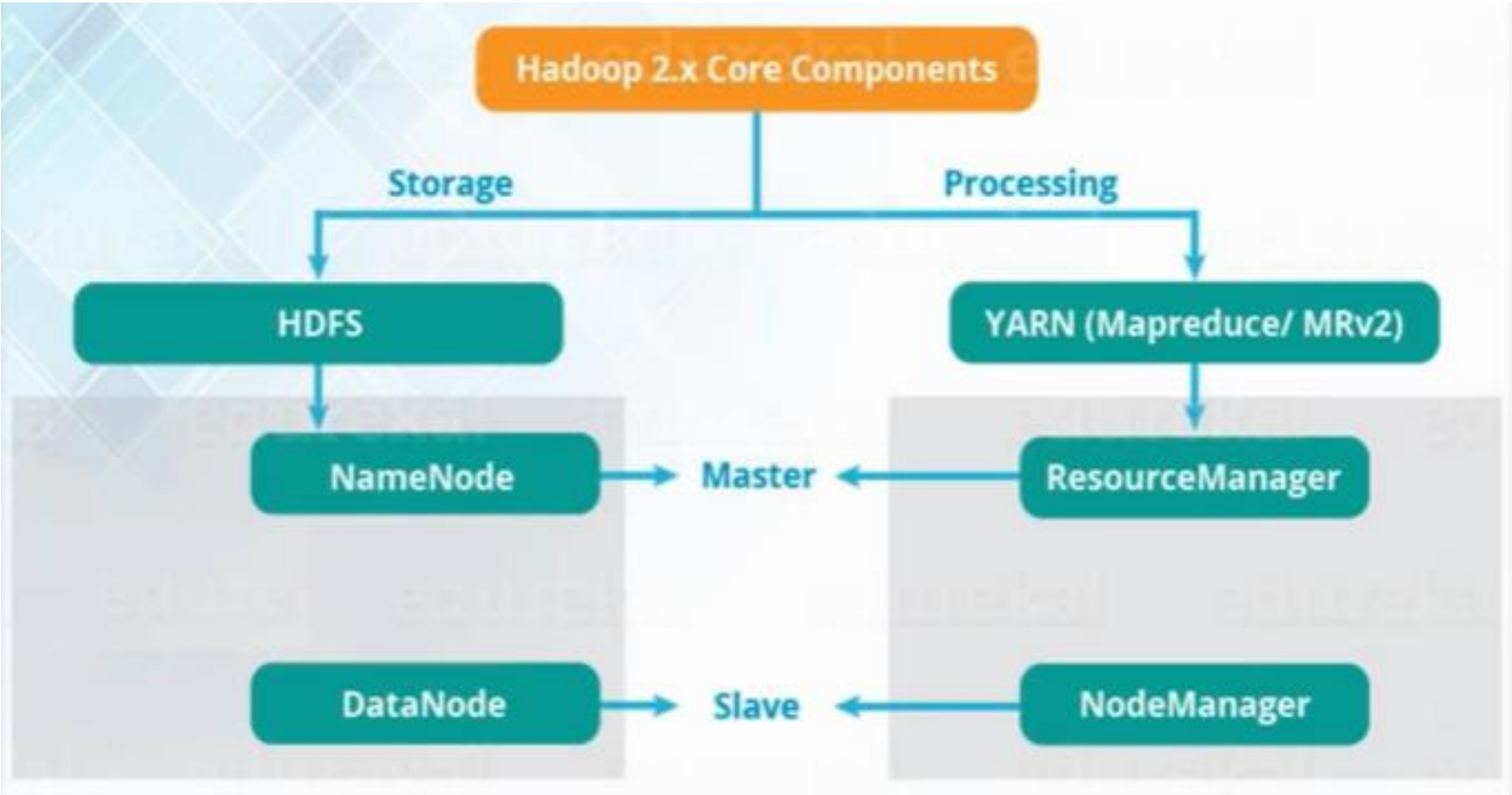
YARN

- *YARN is acronym for Yet Another Resource Negotiator*
- *In MRv1, scalability is a bottleneck when cluster size grows to 4000+*
- *Yahoo began the next generation MapReduce*
- *Programs written in MapReduce v1 work with MapReduce v2*

Main Idea was to split the Jobtackers responsibilities:

- *Resource Manager - (Job Scheduling)*
- *Application Master - (Task Monitoring)*

Hadoop 2.x Deamons

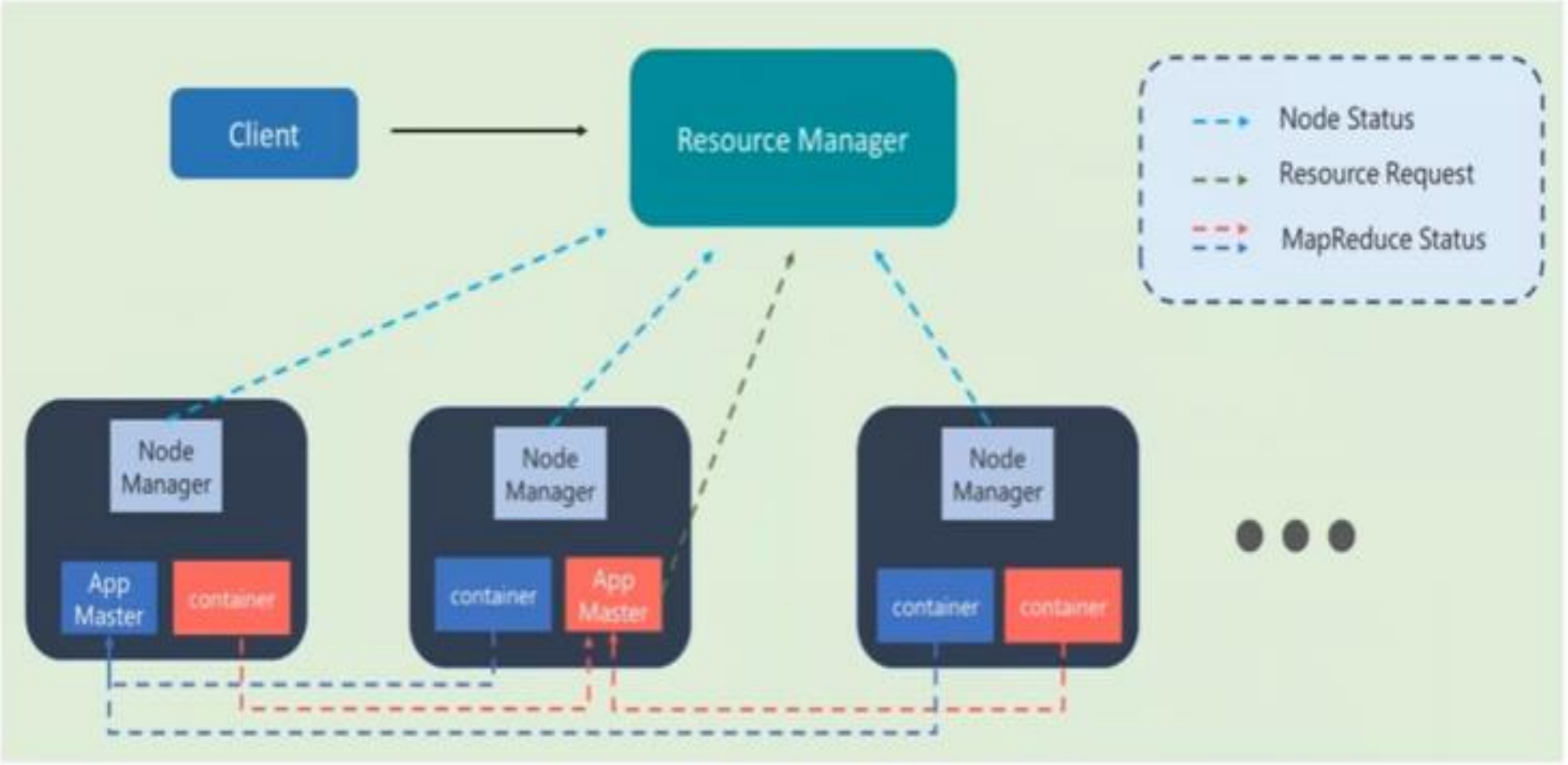


YARN Components

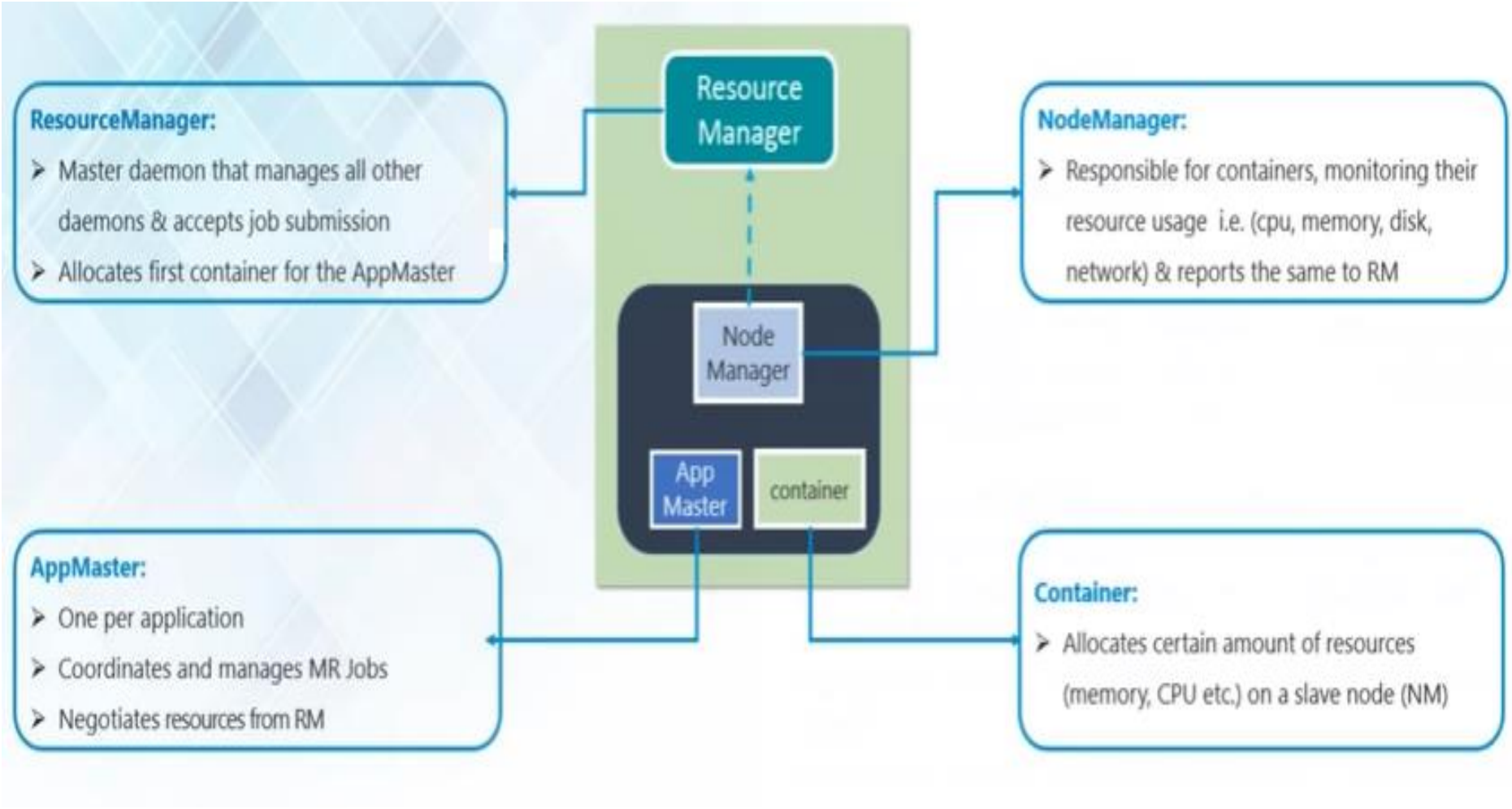
Hadoop 2.x

- Client
  - Submits a MapReduce Job
- Resource Manager
  - Cluster Level resource manager
  - Long Life, High Quality Hardware
- Node Manager
  - One per Data Node
  - Monitors resources on Data Node
- Job History Server
  - Maintains information about submitted MapReduce jobs after their ApplicationMaster terminates
- ApplicationMaster
  - One per application
  - Short life
  - Coordinates and Manages MapReduce Jobs
  - Negotiates with Resource Manager to schedule tasks
  - The tasks are started by NodeManager(s)
- Container
  - Created by NM when requested
  - Allocates certain amount of resources (memory, CPU etc.) on a slave node

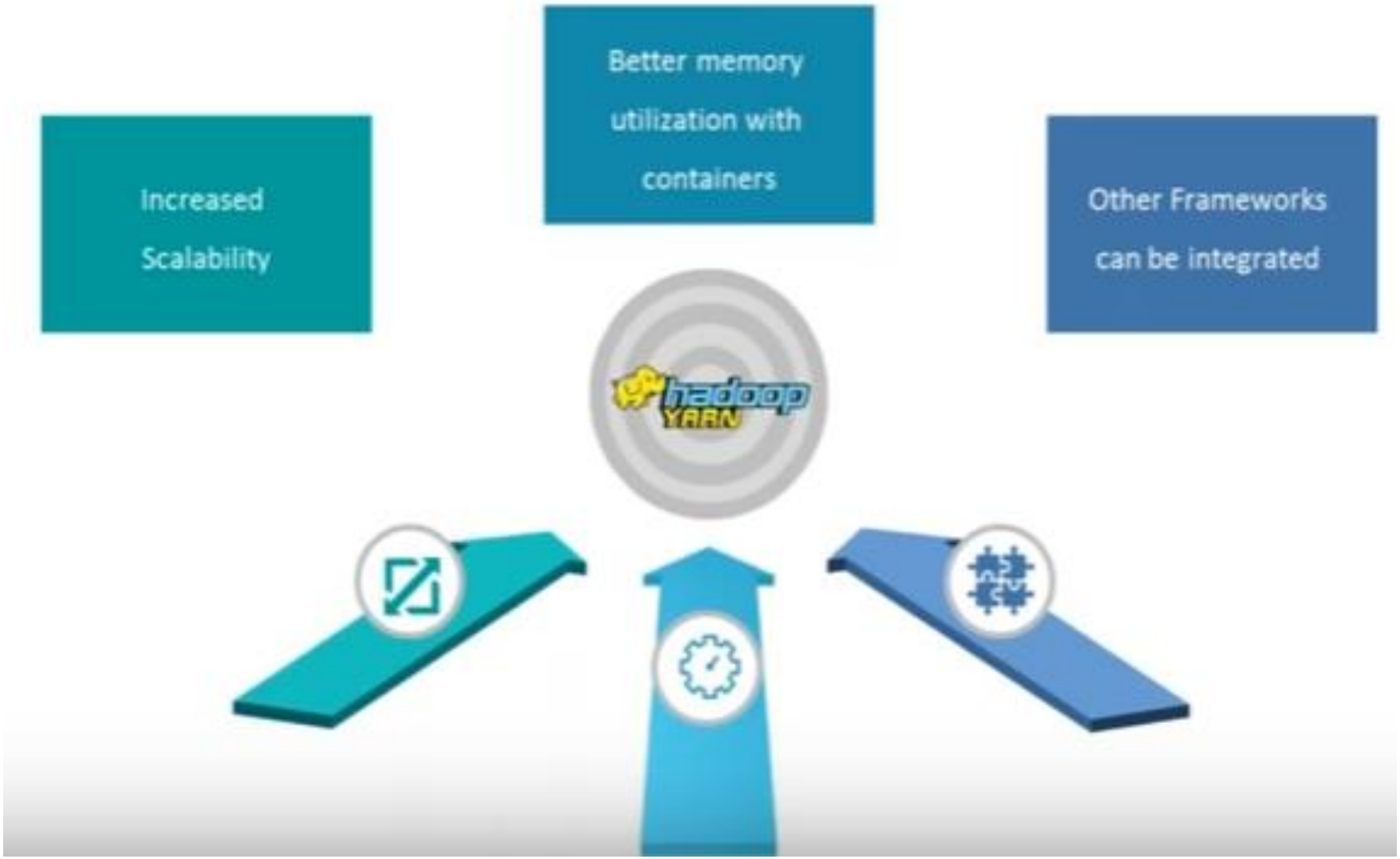
Entities in YARN



YARN Components



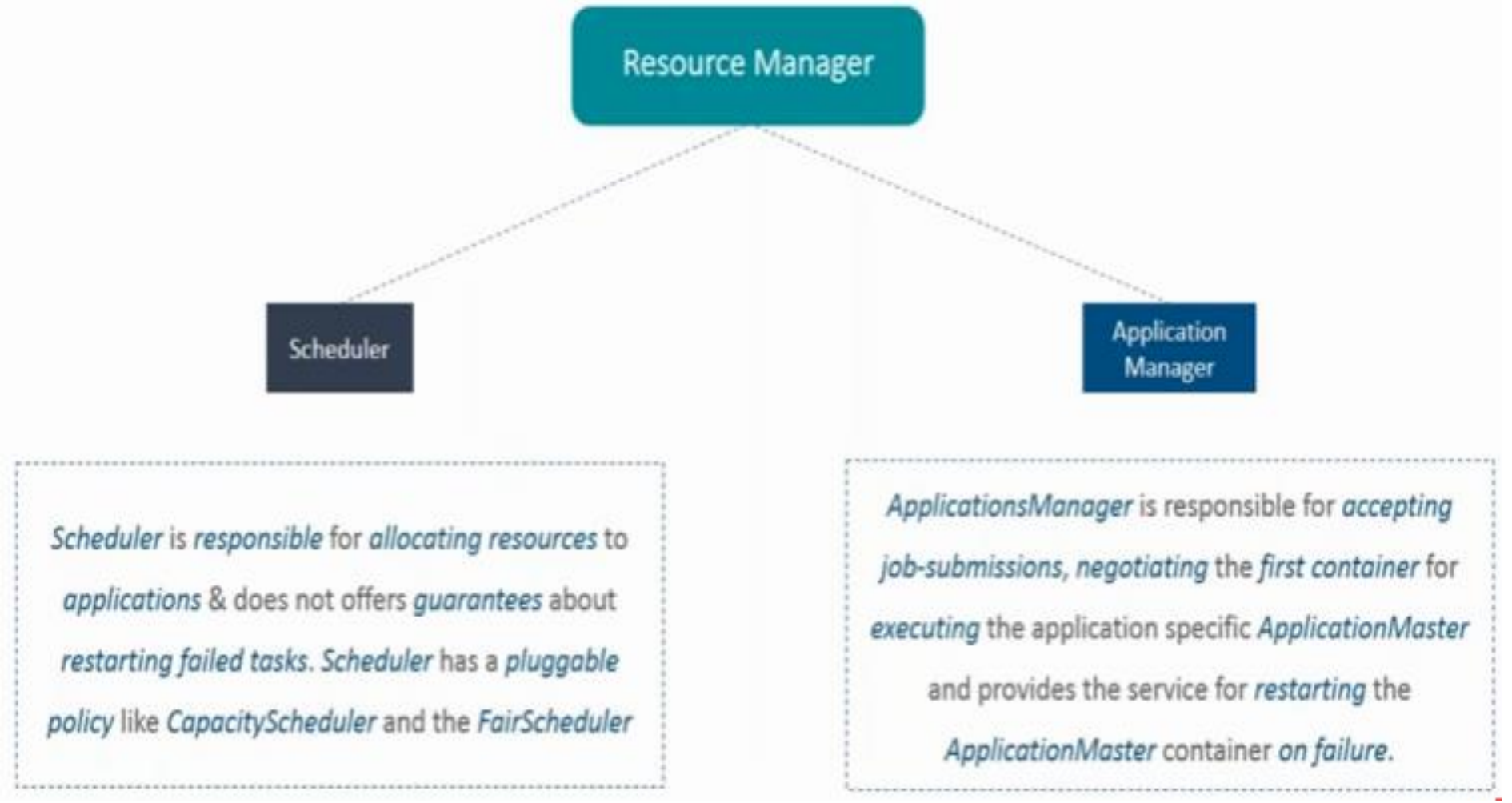
Benefits of YARN



ResourceManager

- *It has components-scheduler & application manager*
- *It is a global scheduler*
- *Manages & allocates cluster resources*

ResourceManager



NodeManager

NodeManager takes care of individual compute nodes in a Hadoop cluster

Node Manager manages following:

- *Container Lifecycle Management*
- *Container Dependencies*
- *Container Leases Node & Container Resource Usage*
- *Node Health*
- *Log Management*
- *Reporting Node & container status to RM*

ApplicationMaster

ApplicationMaster is the process that coordinates an application's execution in the cluster

Each application has its own unique ApplicationMaster, which is tasked with negotiating resources (containers) from the ResourceManager and working with the NodeManager to execute and monitor the tasks.

Container

Container is a collection of physical resources such as RAM, CPU cores, and disks on a single node.

- *There can be multiple containers on a single node*
- *Every node in the system is composed of multiple containers*
- *The ApplicationMaster can request any container so as to occupy a multiple of the minimum size*

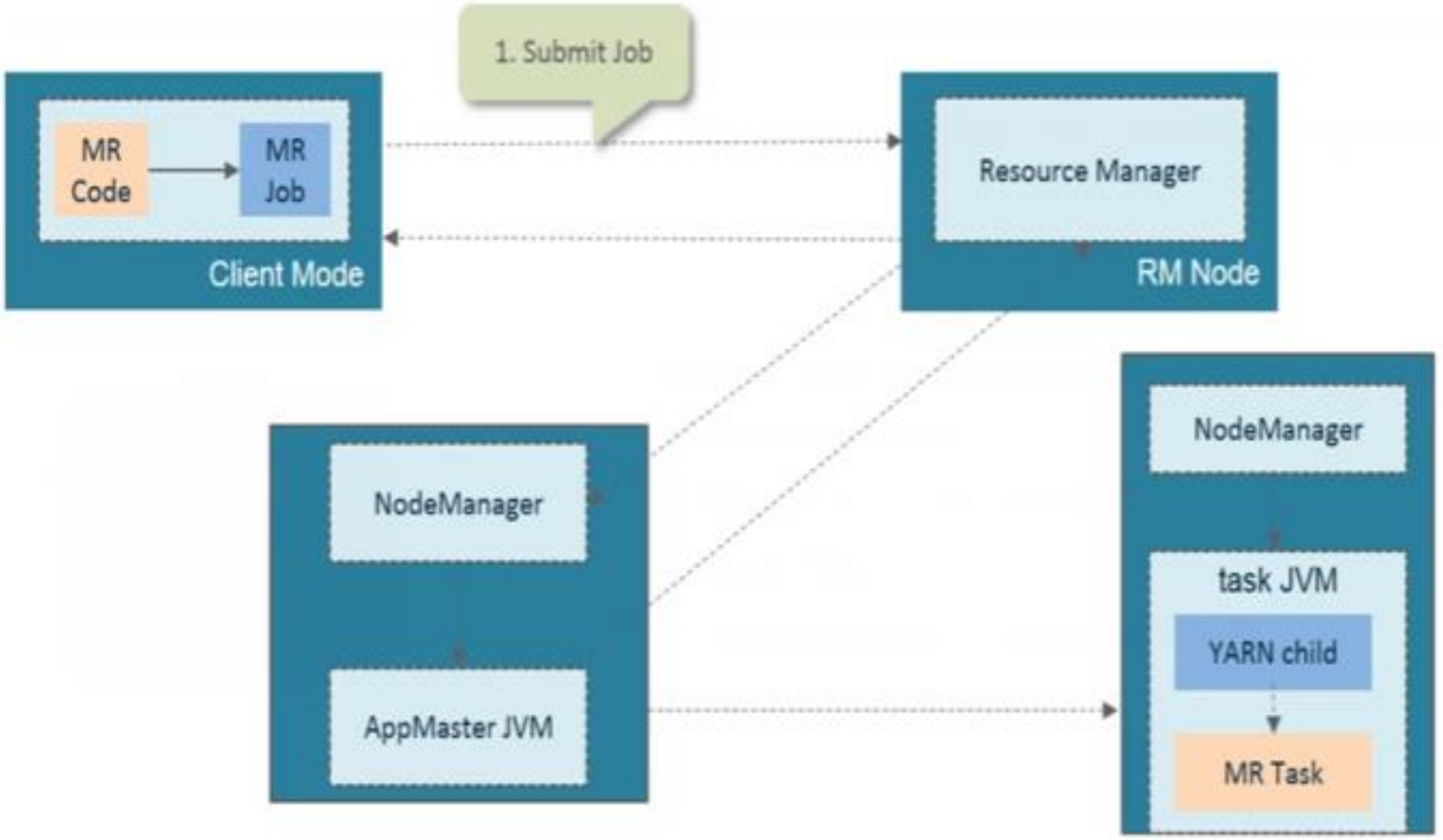
Container

Container is a collection of physical resources such as RAM, CPU cores, and disks on a single node.

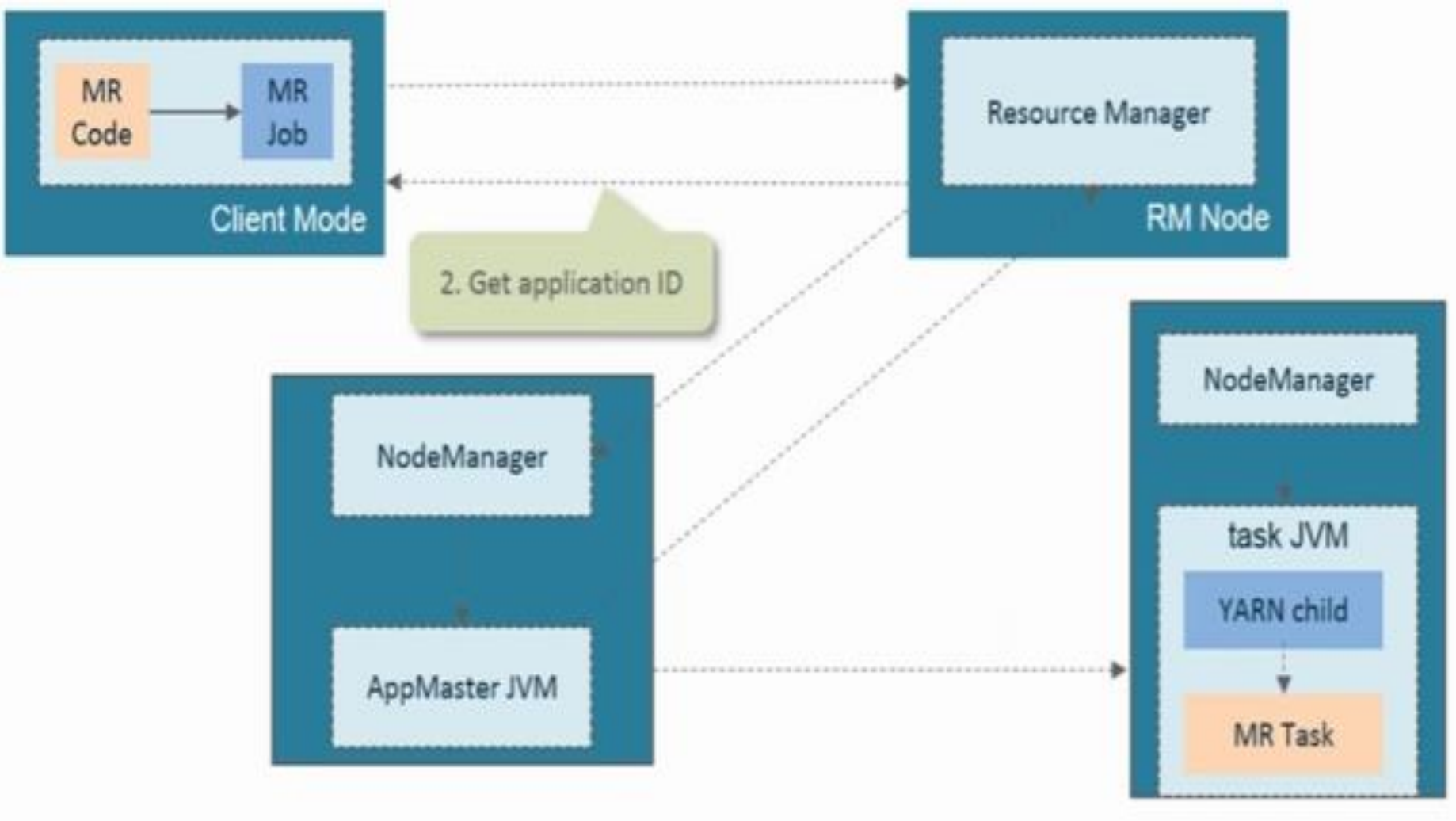
Container Launch Context (CLC) includes

- *Environment Variable*
- *Dependencies, i.e. local resources*
- *Security Tokens*
- *Command necessary to create the process, that application wants to launch*

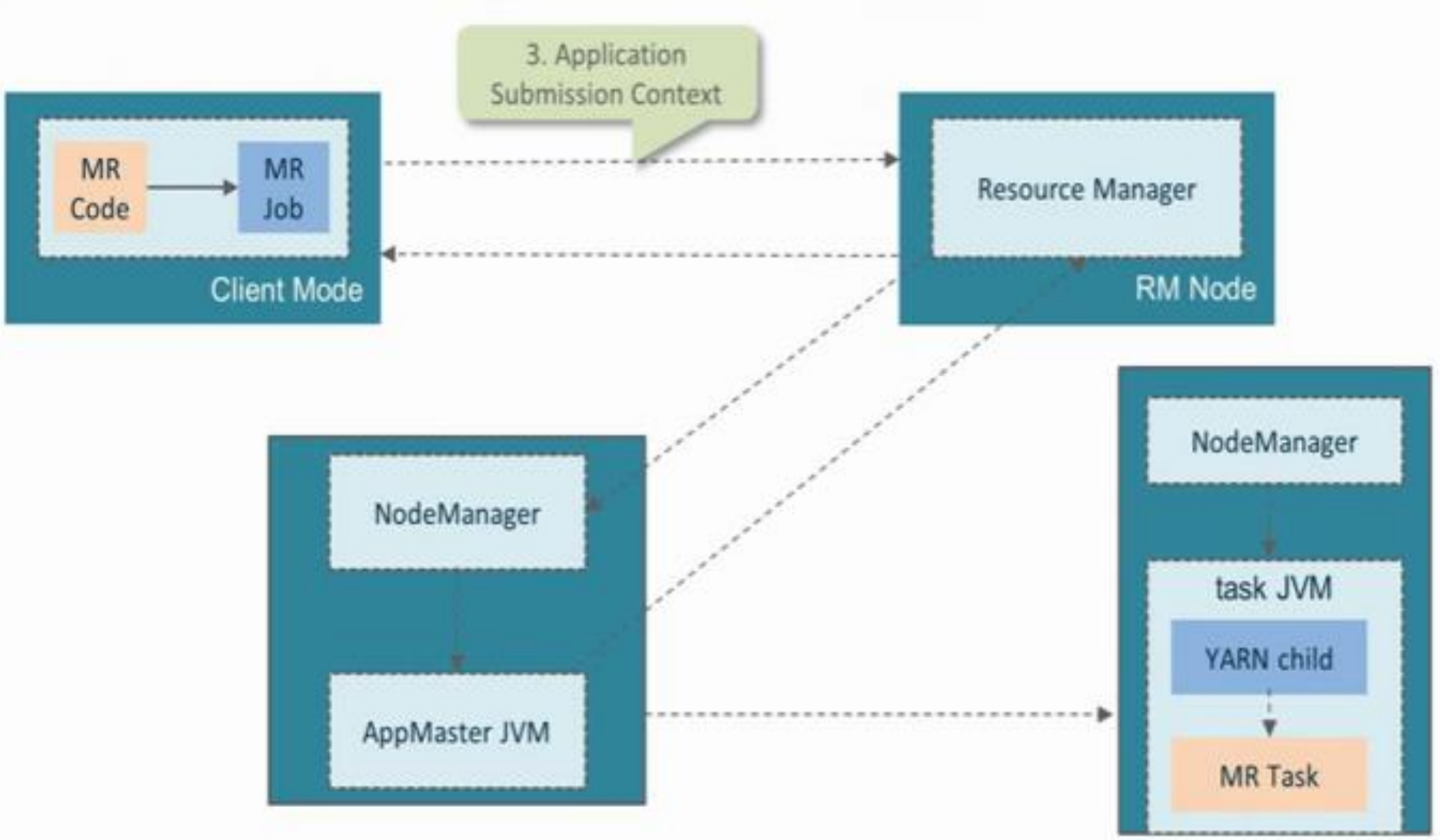
Application Submission in YARN



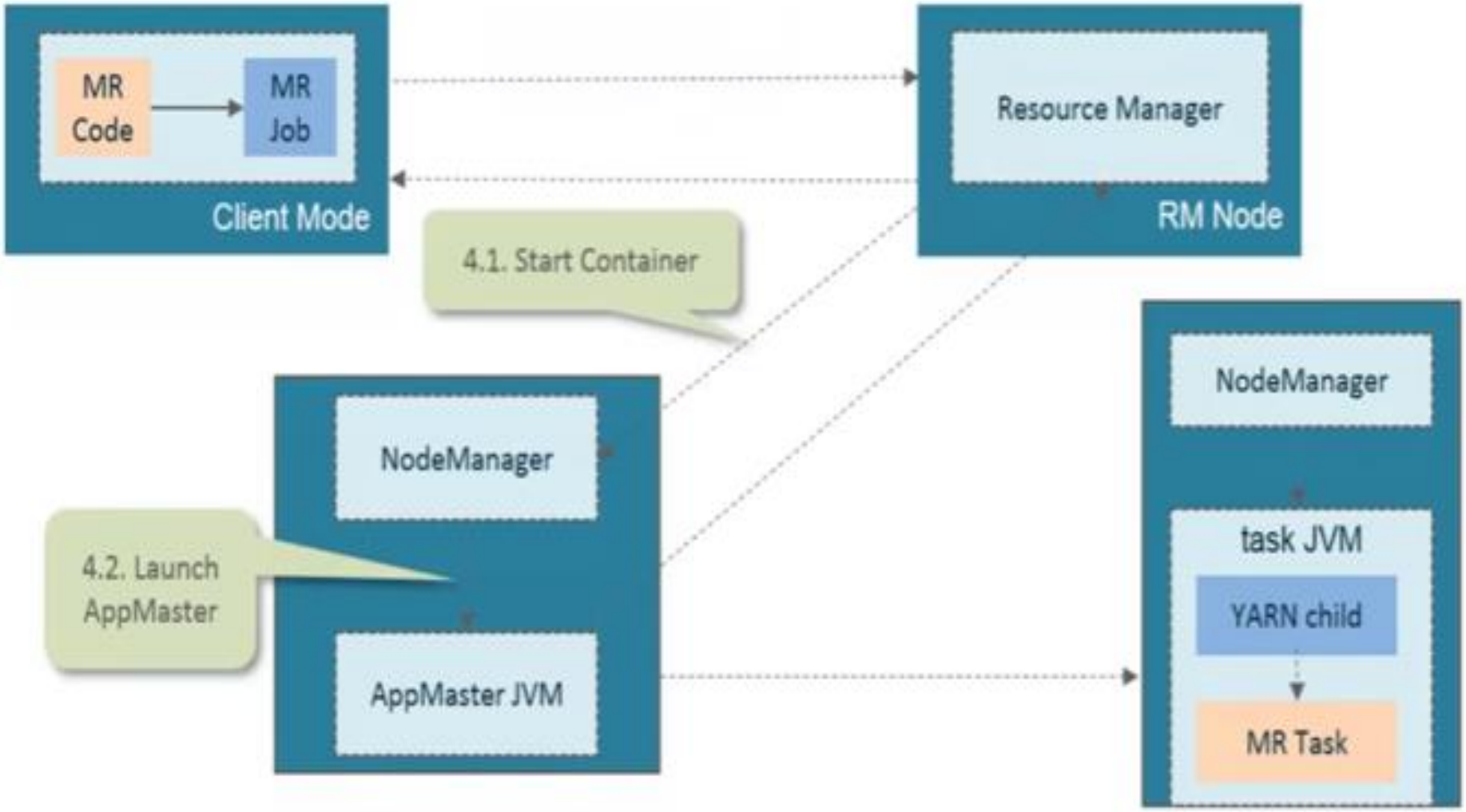
Application Submission in YARN



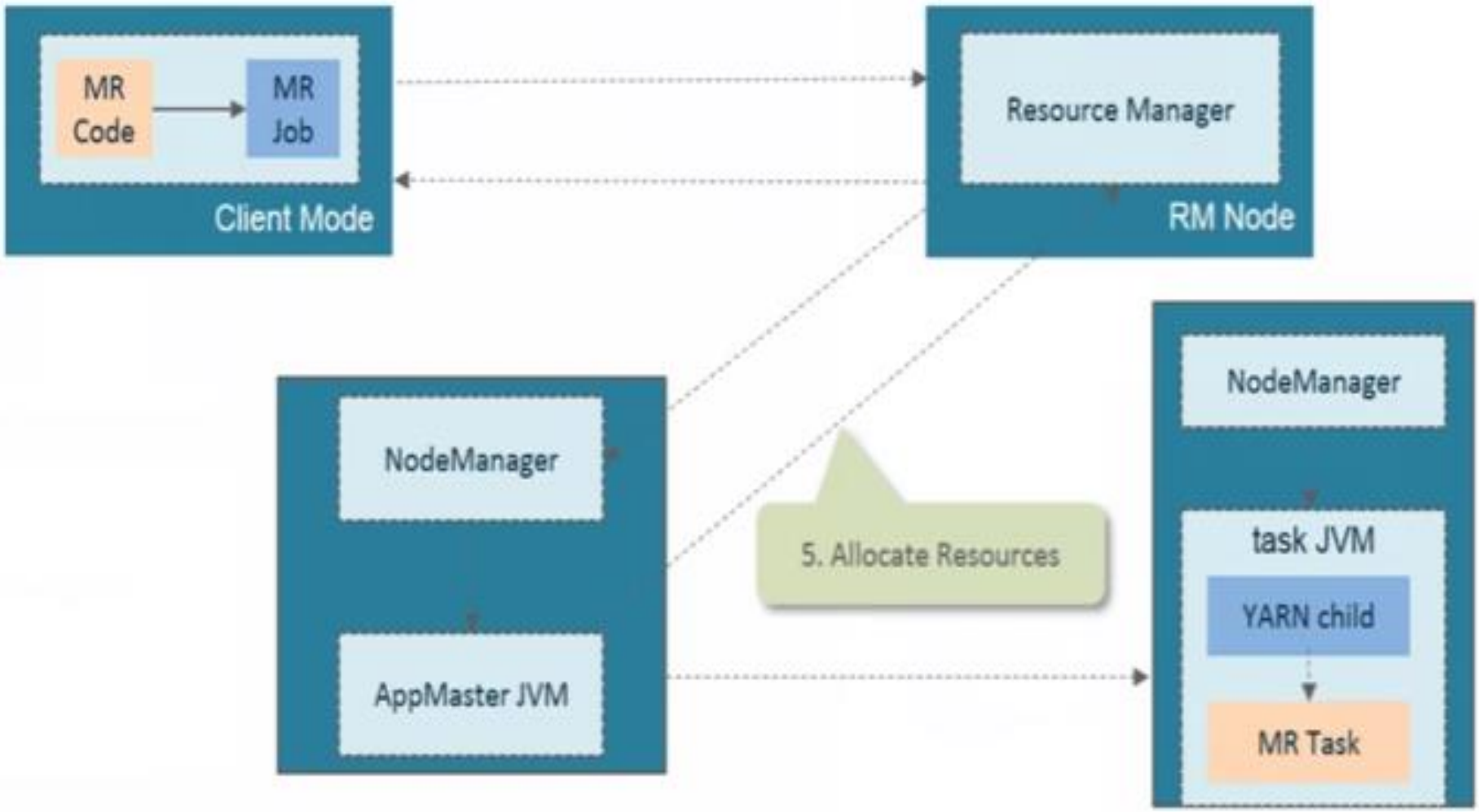
Application Submission in YARN



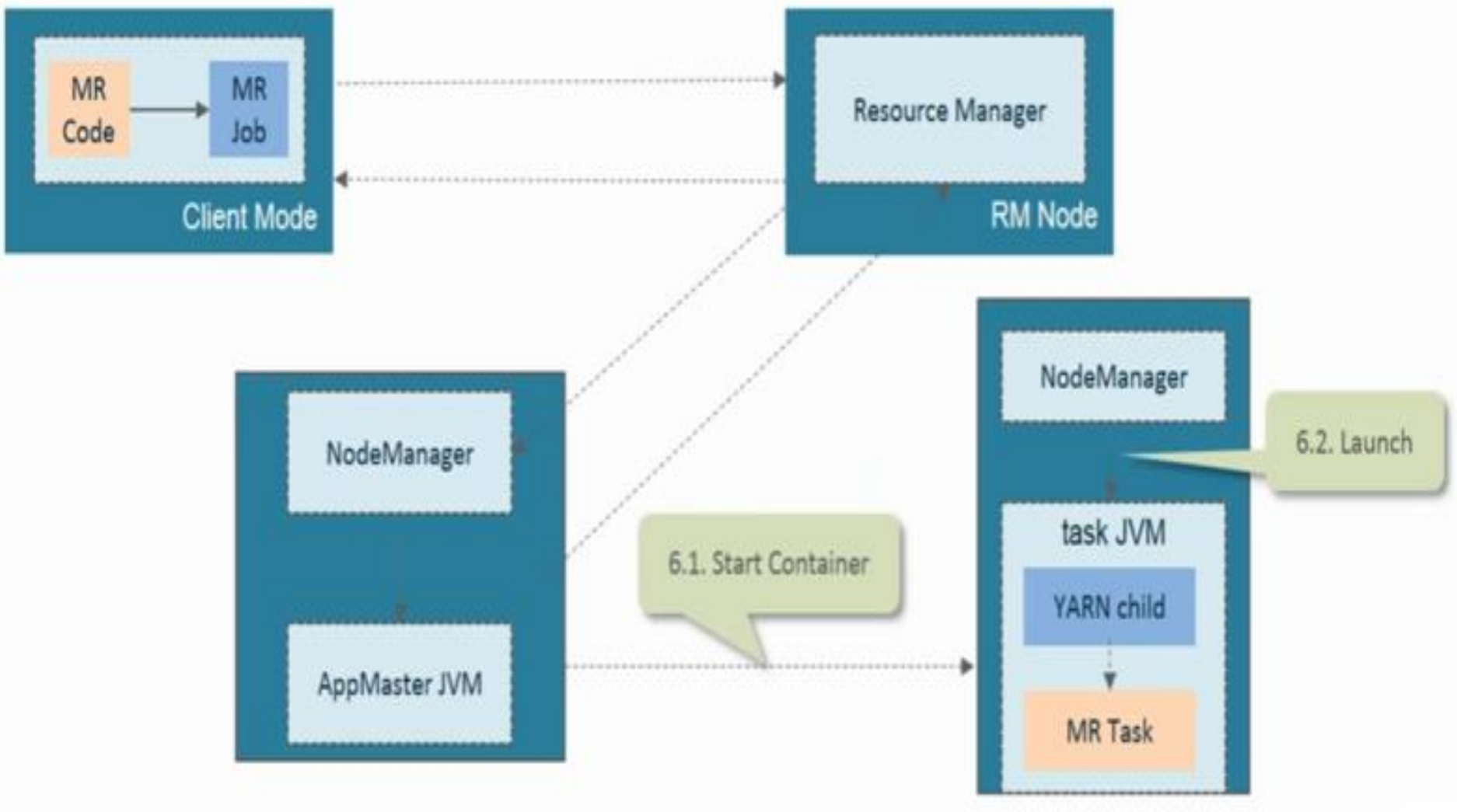
Application Submission in YARN



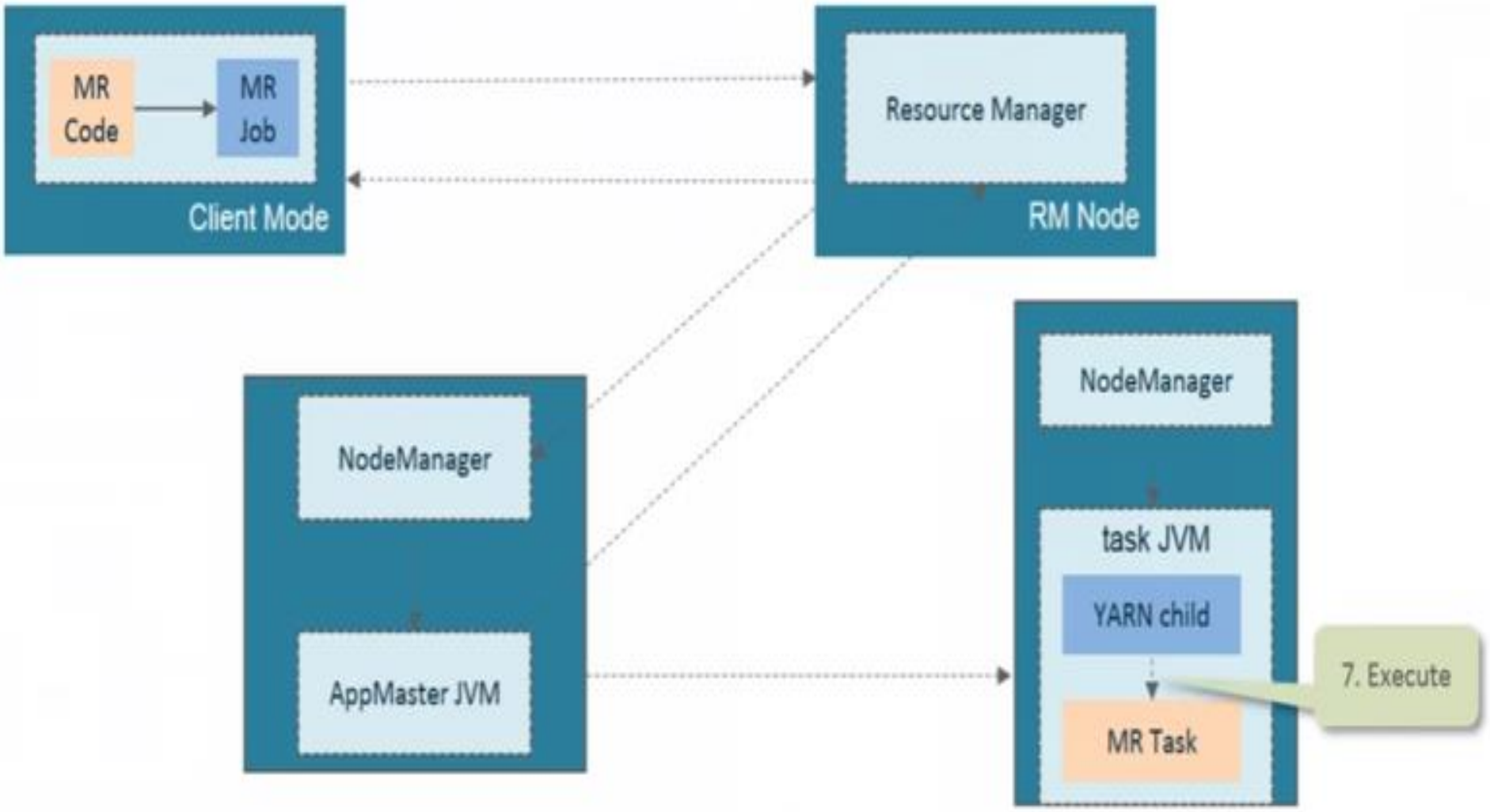
Application Submission in YARN



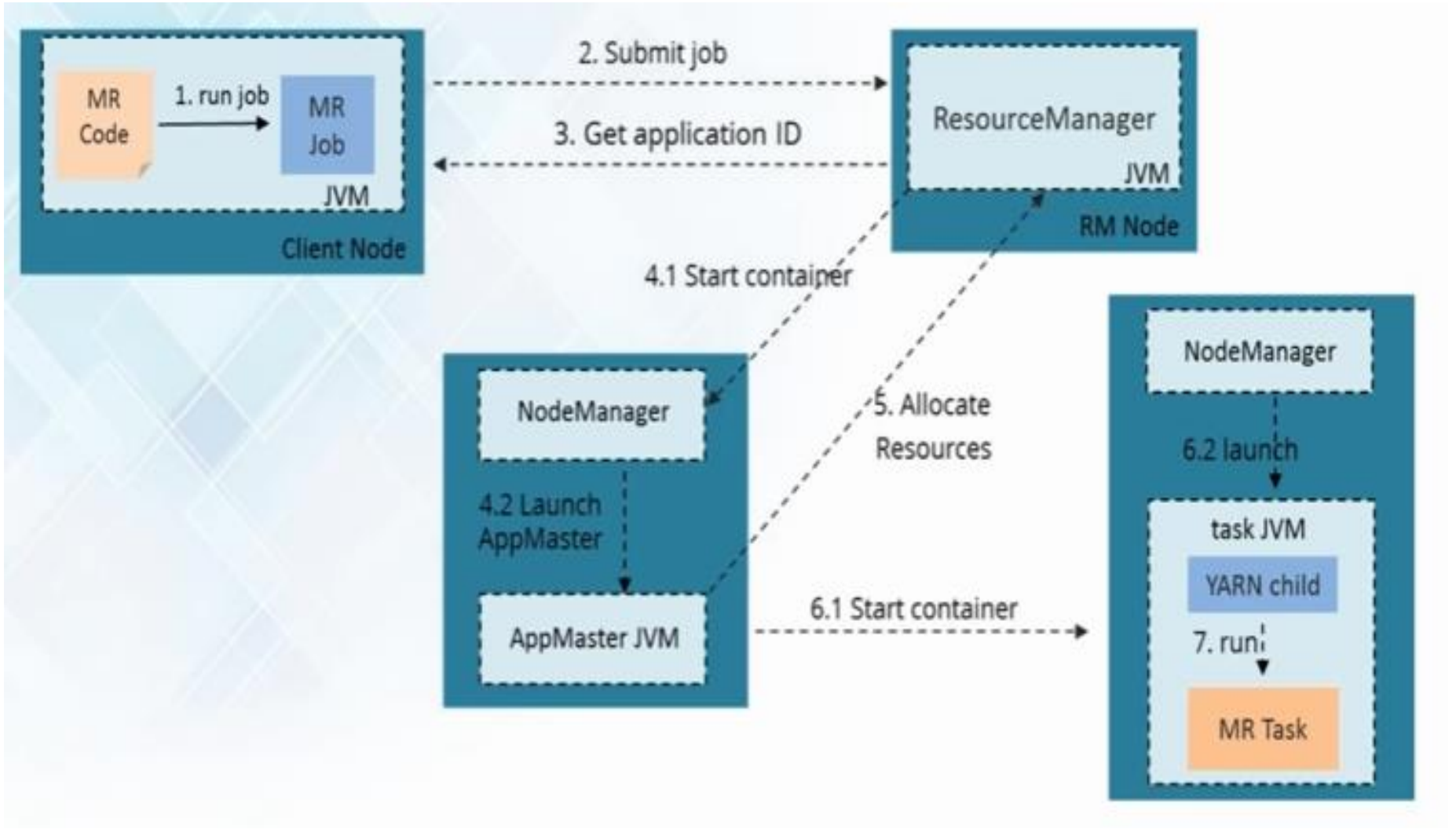
Application Submission in YARN



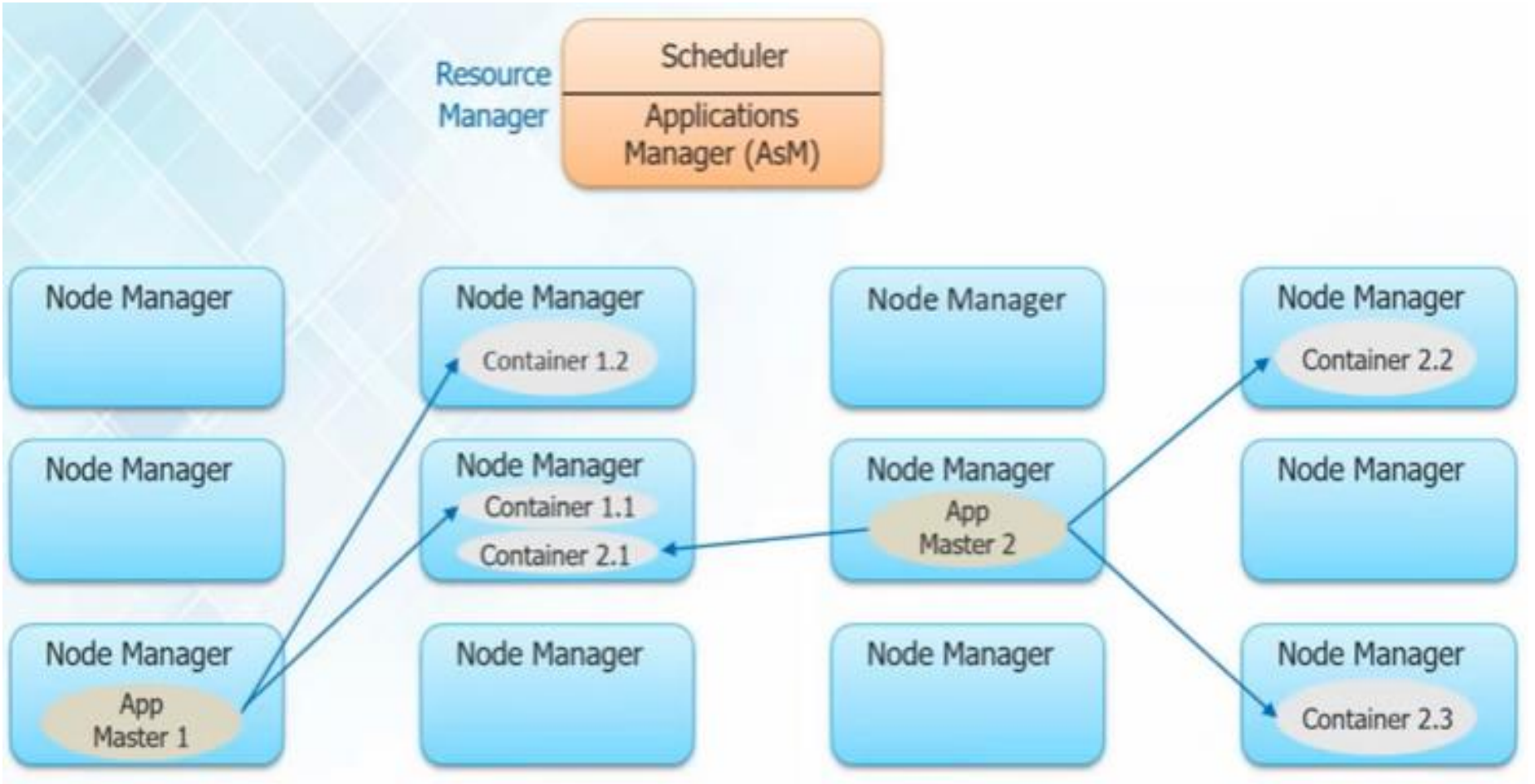
Application Submission in YARN



MapReduce job workflow



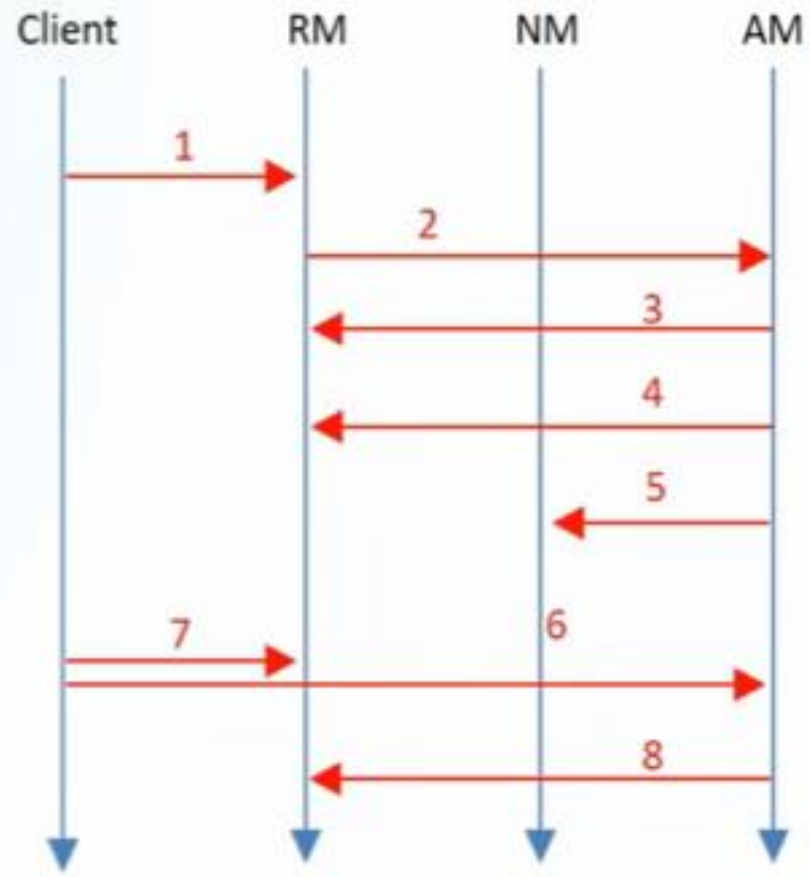
MapReduce Workflow



Application Workflow

→ Execution Sequence :

1. Client submits an application
2. RM allocates a container to start AM
3. AM registers with RM
4. AM asks containers from RM
5. AM notifies NM to launch containers
6. Application code is executed in container
7. Client contacts RM/AM to monitor application's status
8. AM unregisters with RM



References

- Book
  - Big Data and Analytics – Seema Acharya and Subhashini C – Wiley India
- Youtube Channel-edureka!
  - https://www.youtube.com/channel/UCkw4JCwteGrDHlsyIlKo4tQ