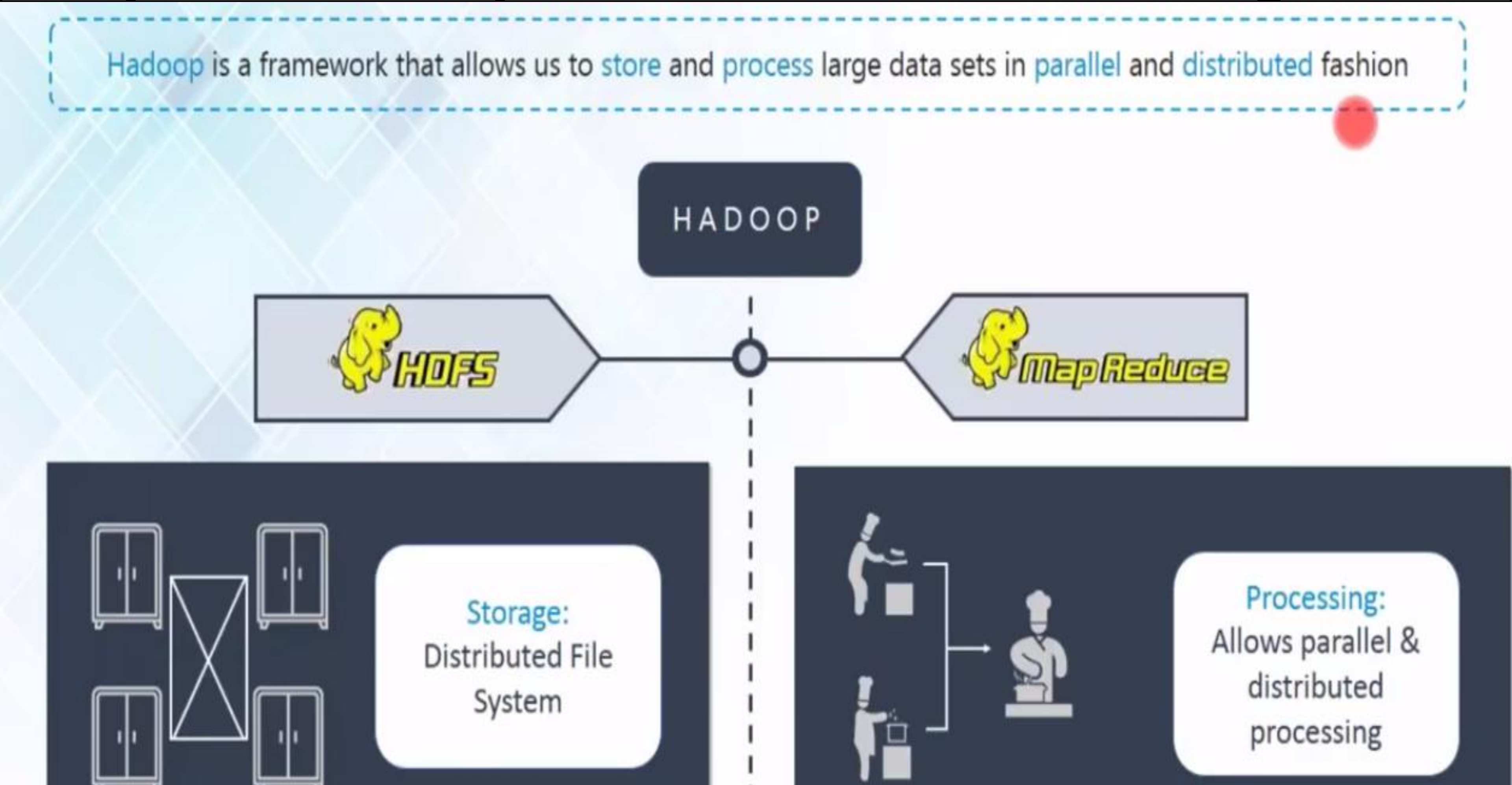
Apache Hadoop Framework to Process Big Data



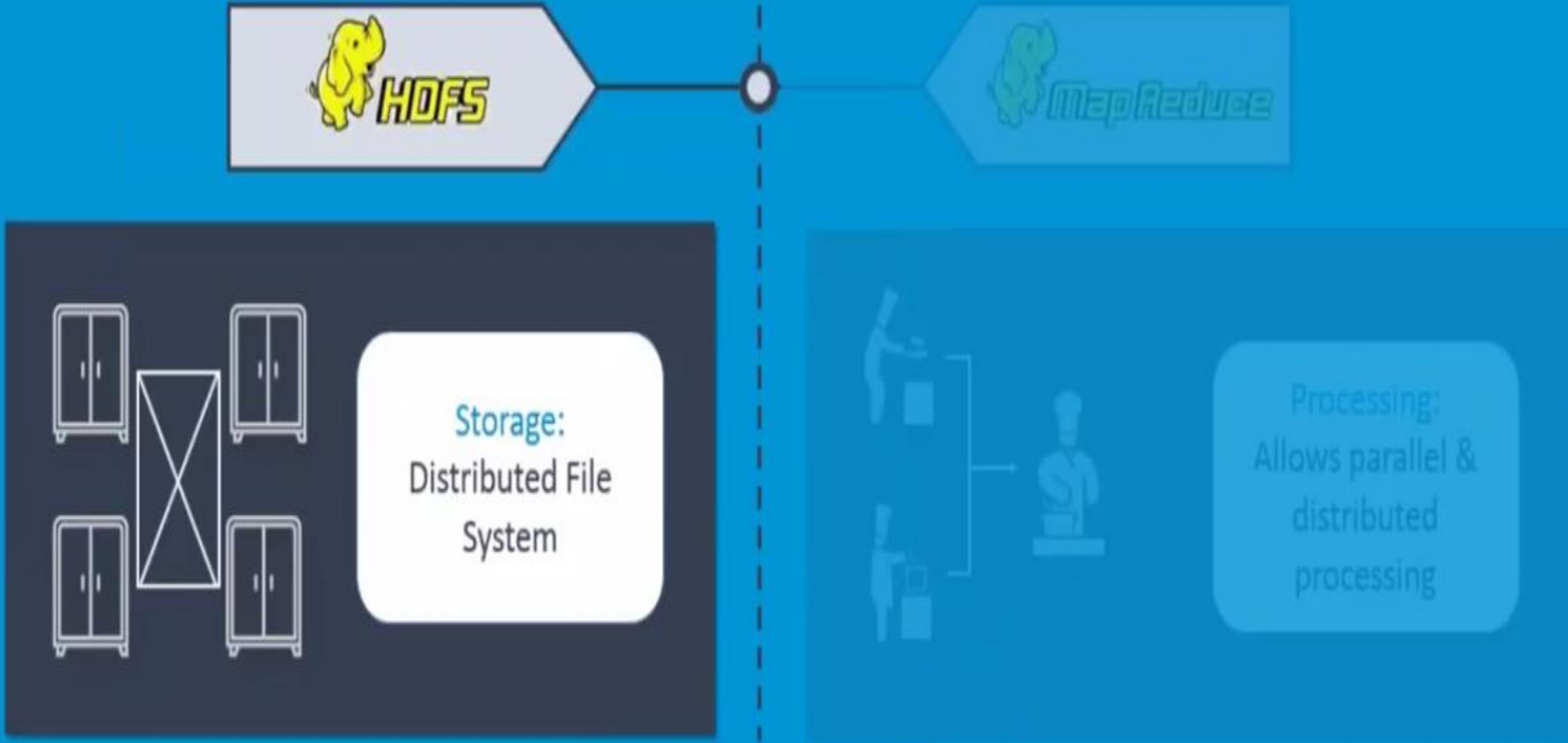
Hadoop is a framework that allows us to store and process large data sets in parallel and distributed fashion

HADOOP

Storage:
Distributed File System

Processing:
Allows parallel & distributed processing

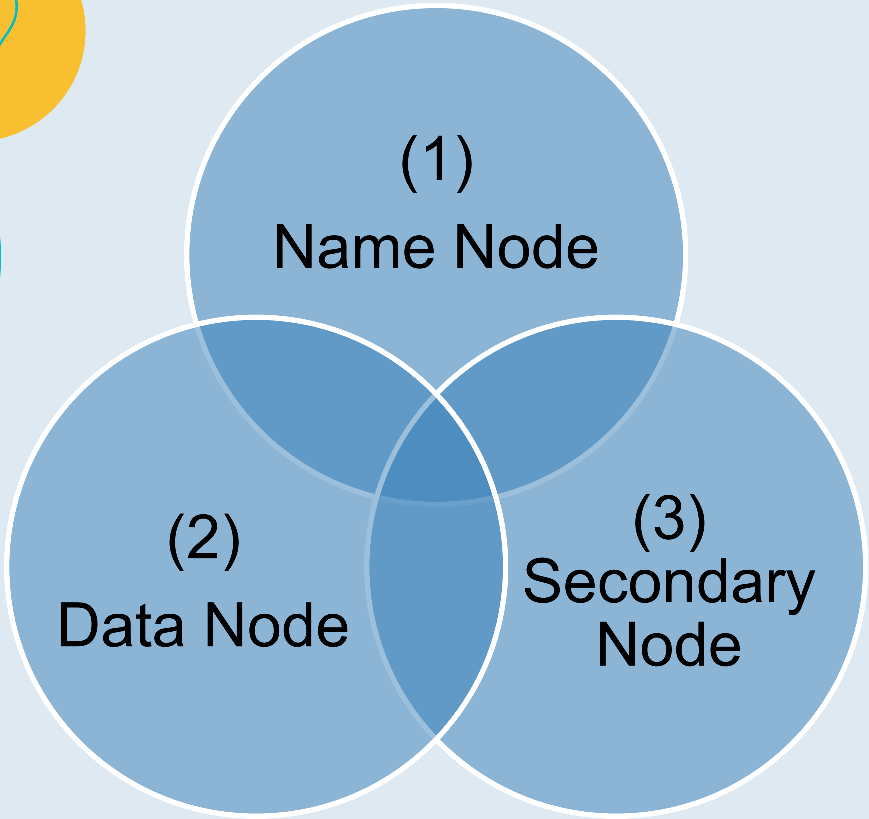
HADOOP CORE COMPONENTS



Storage: Distributed File System

Processing: Allows parallel & distributed processing

HDFS Core Components

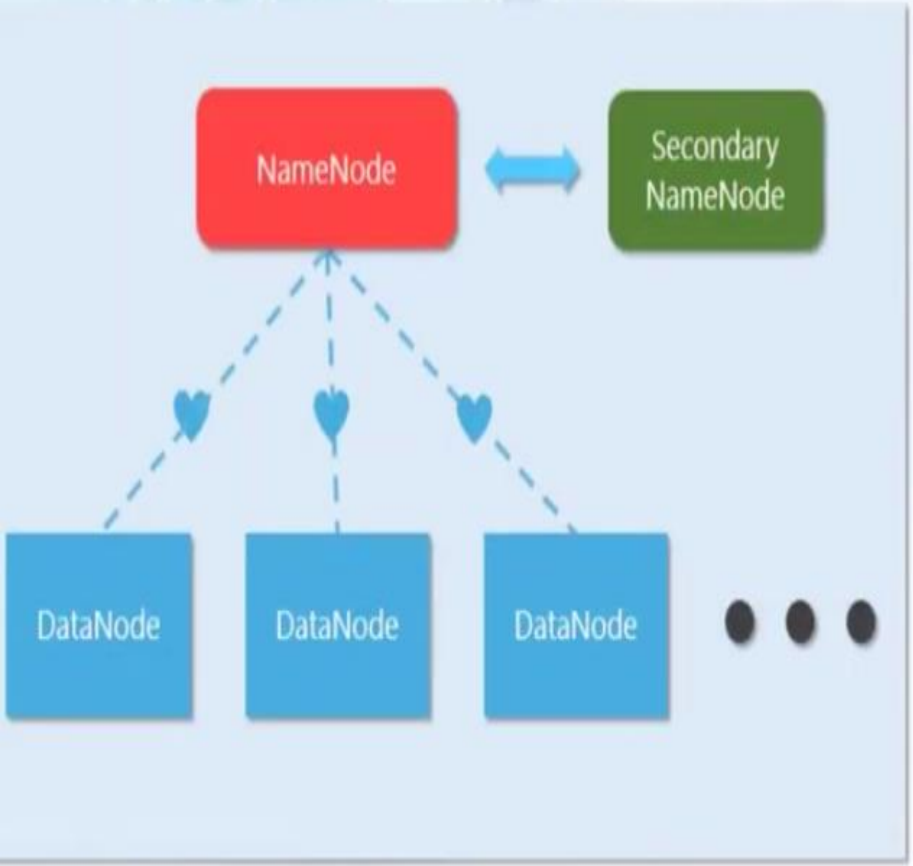


(1) Name Node

(2) Data Node

(3) Secondary Node

HDFS Core Components-Name Node & Data Node



NameNode:

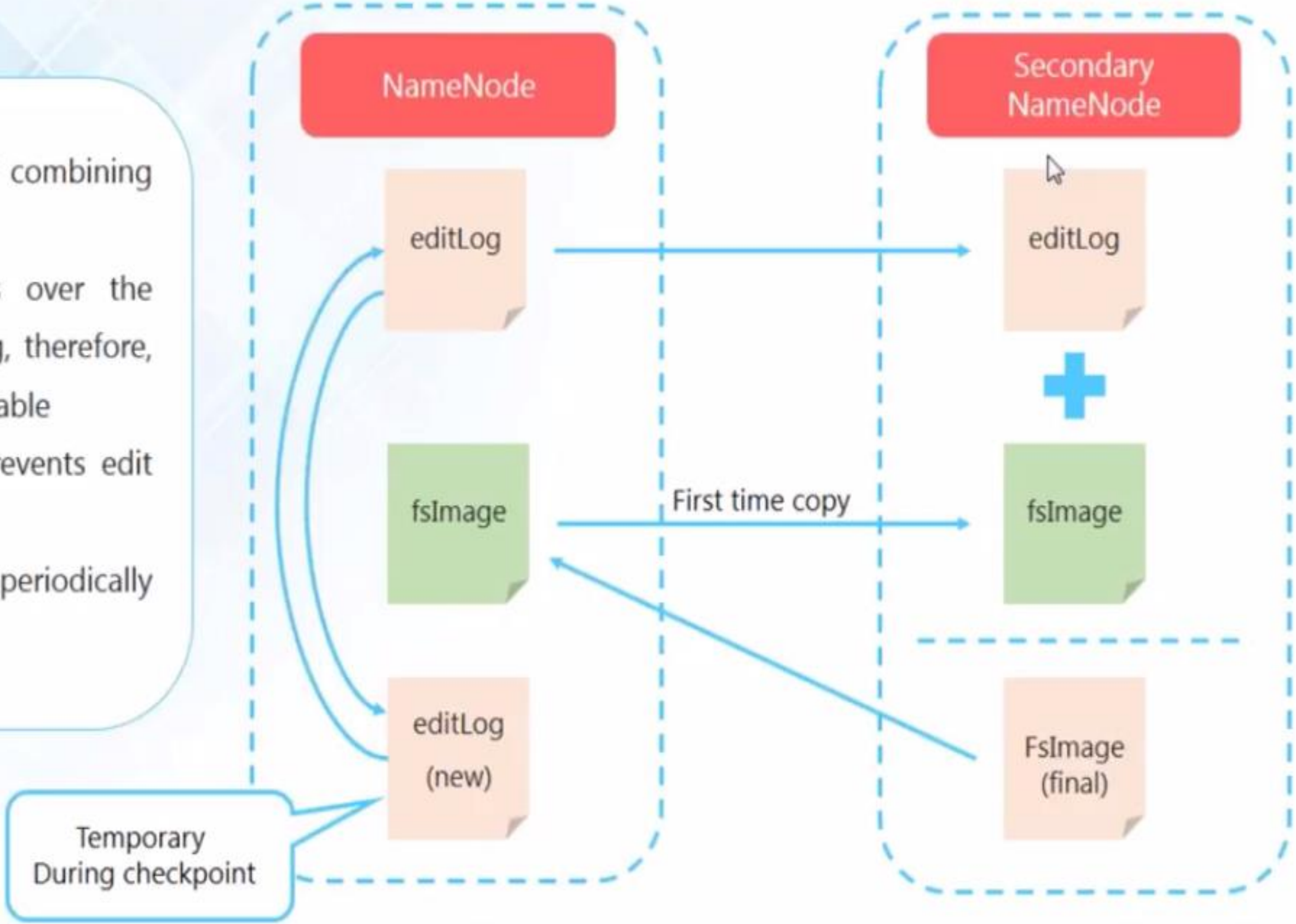
- ➤ Maintains and Manages DataNodes
- ➤ Records metadata i.e. information about data blocks e.g. location of blocks stored, the size of the files, permissions, hierarchy, etc.
- ➤ Receives heartbeat and block report from all the DataNodes

DataNode:

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

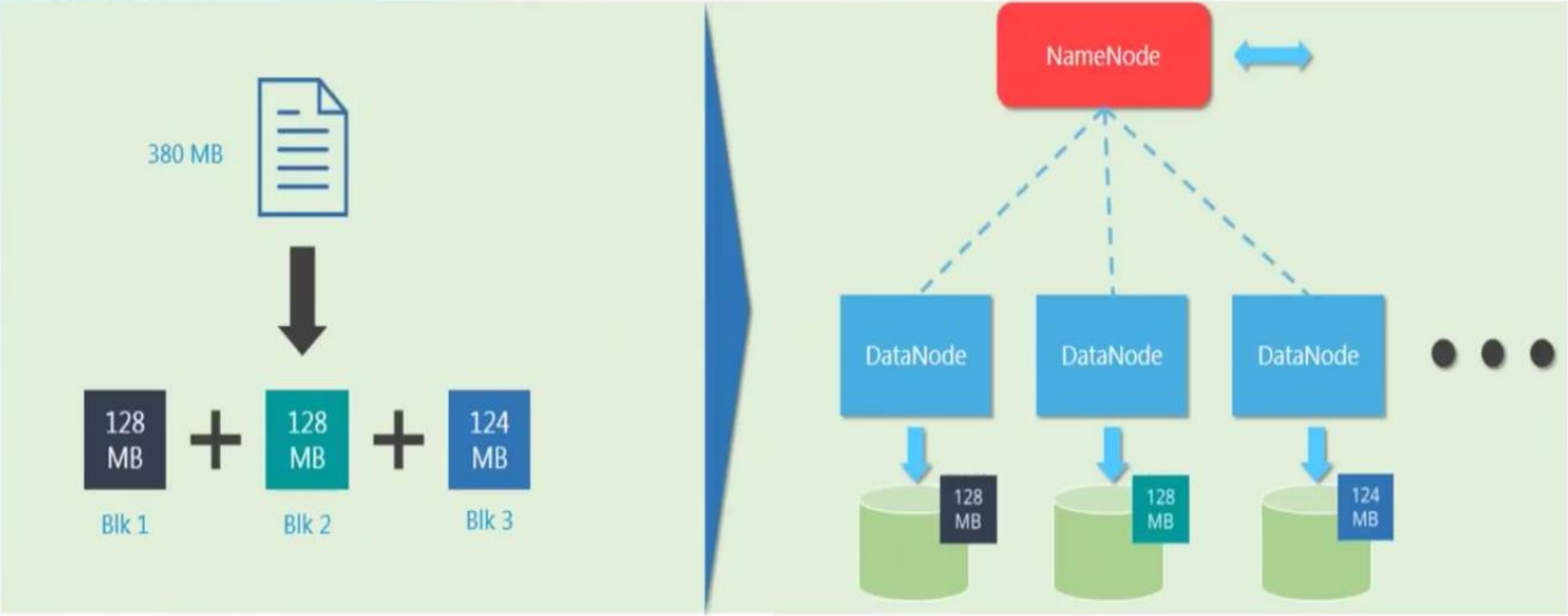
Secondary Name Node & Check pointing

- 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)



HDFS Data Blocks

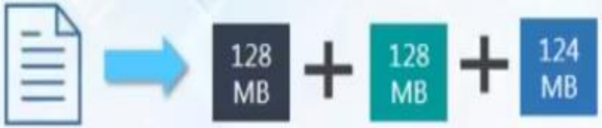
- Each file is stored on HDFS as blocks
- The default size of each block is 128 MB in Apache Hadoop 2.x (64 MB in Apache Hadoop 1.x)

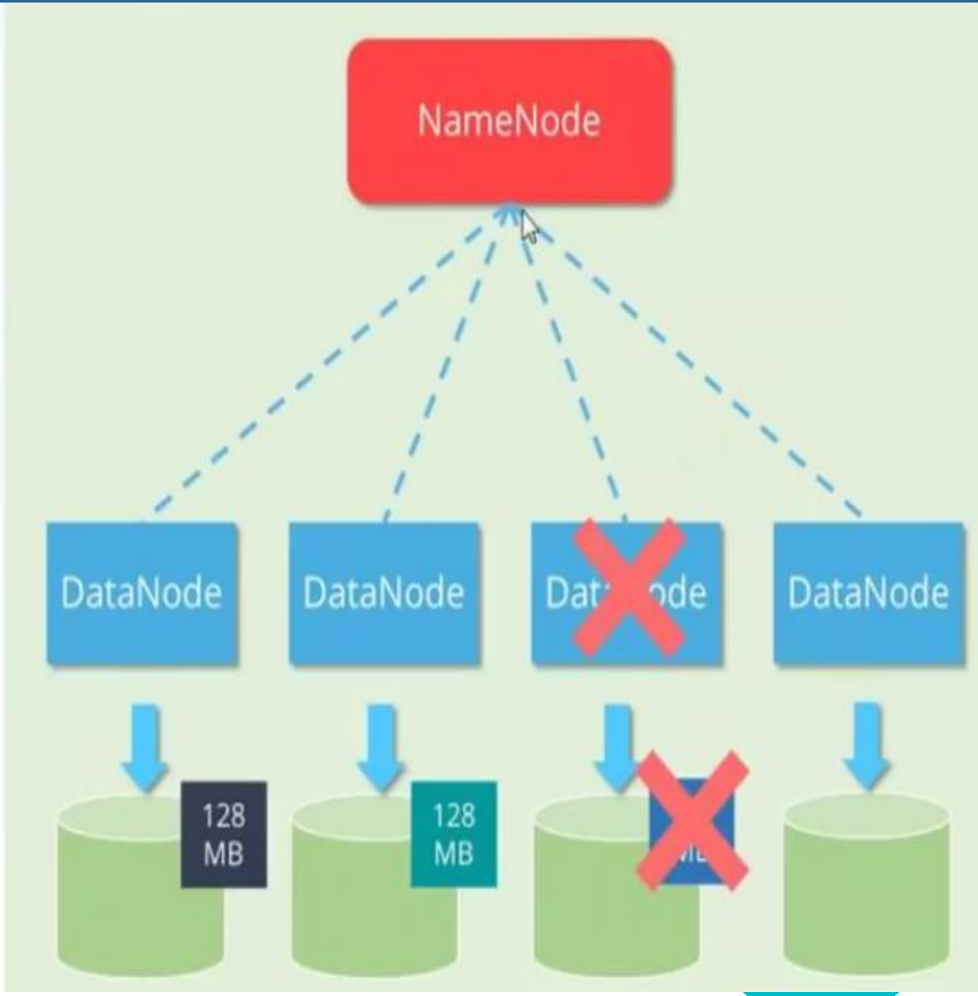


Fault Tolerance :How Hadoop cope up with DataNode Failure

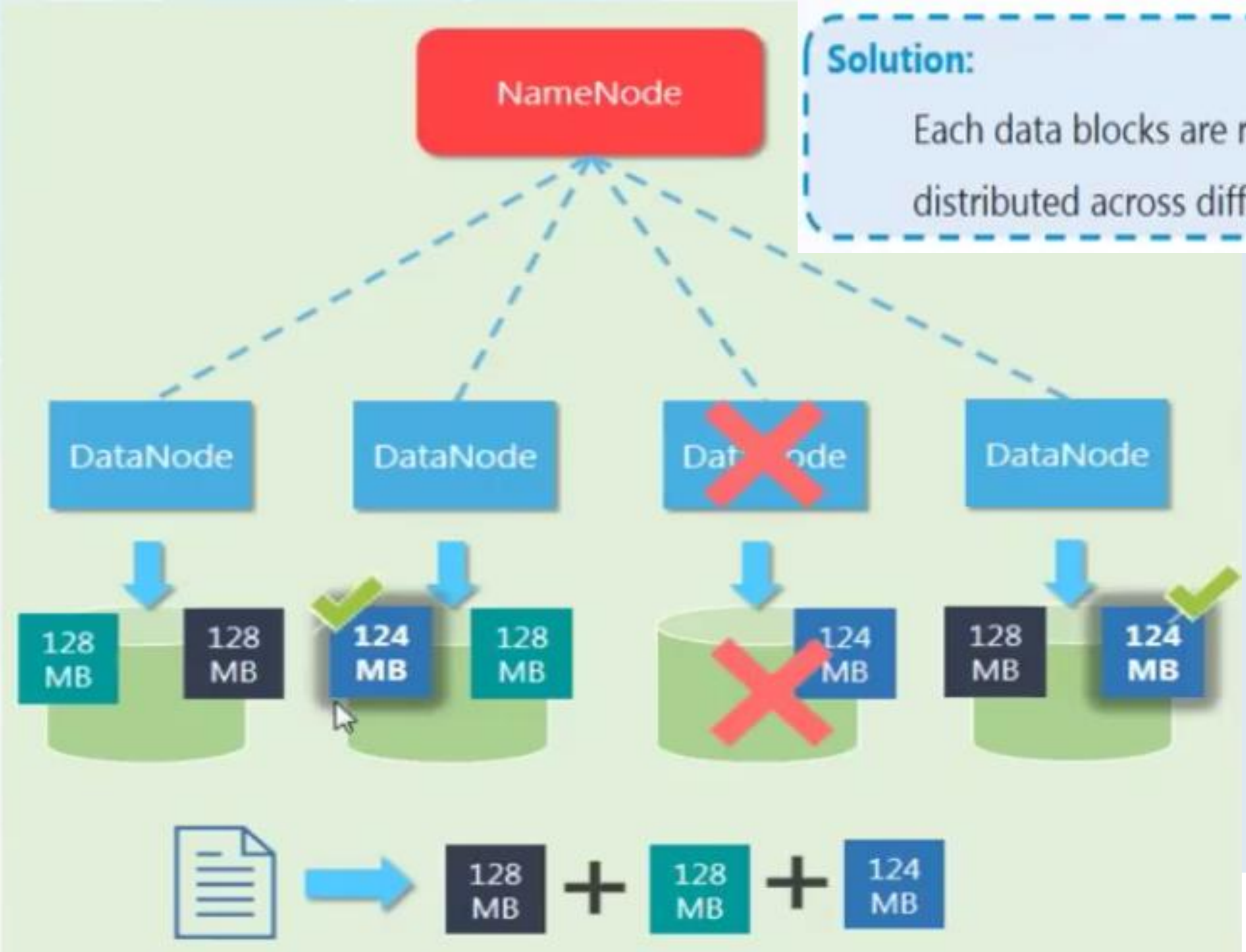
Scenario:

One of the DataNodes crashed containing the data blocks





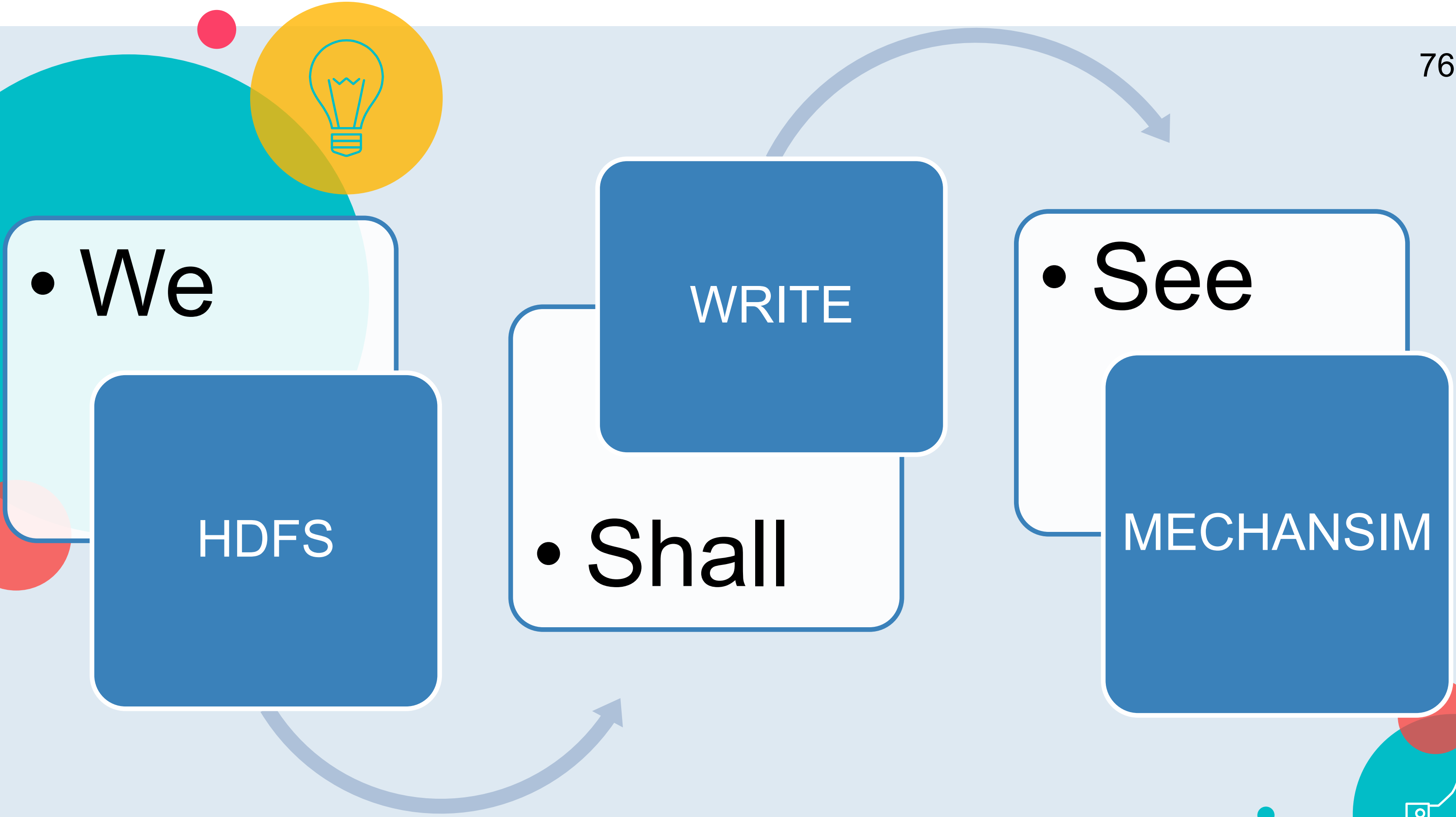
Fault Tolerance :Replication Factor



Solution:

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

As it is said Never Put All Your Eggs in the Same Basket



- We

HDFS

- Shall

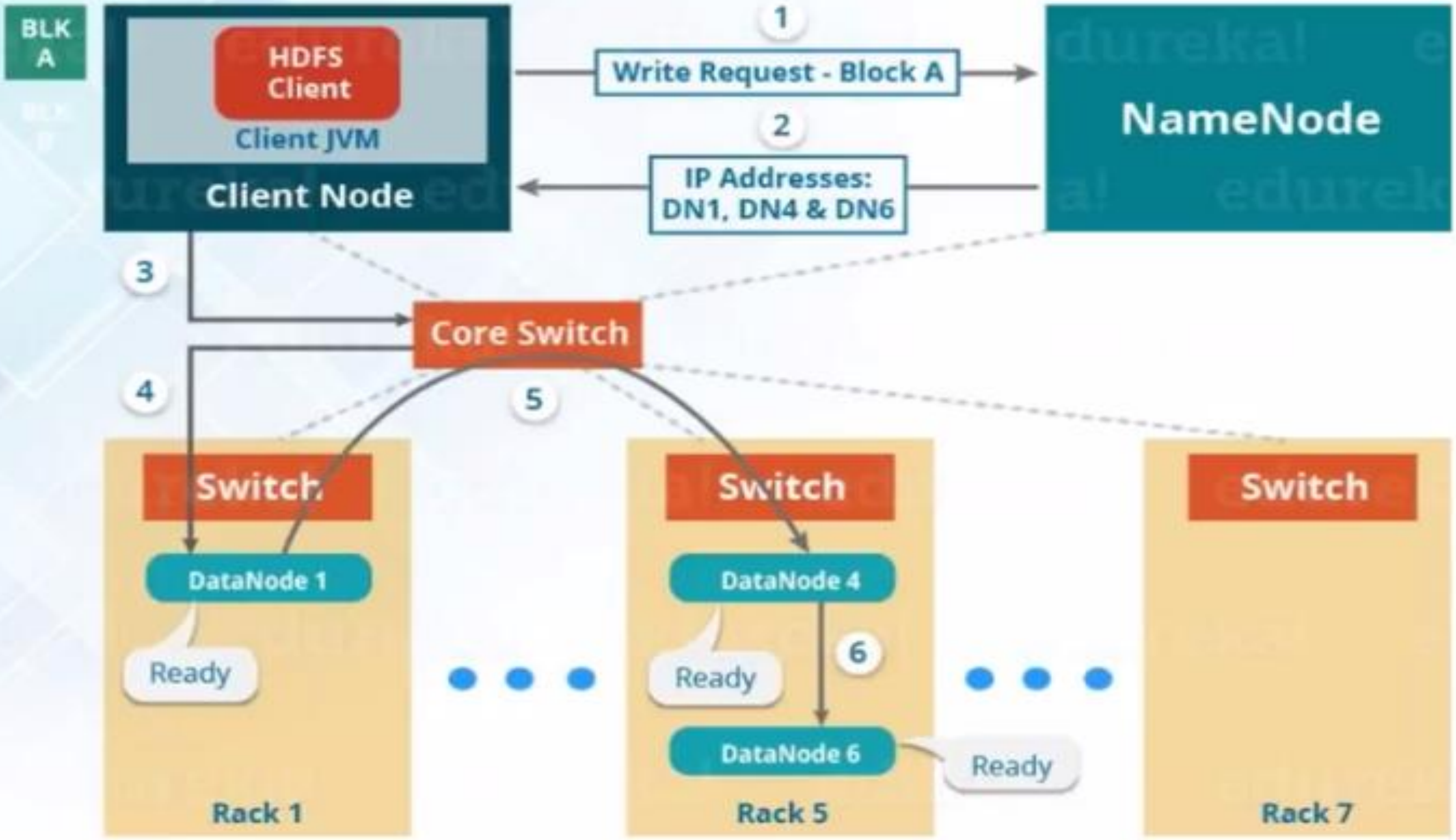
WRITE

- See

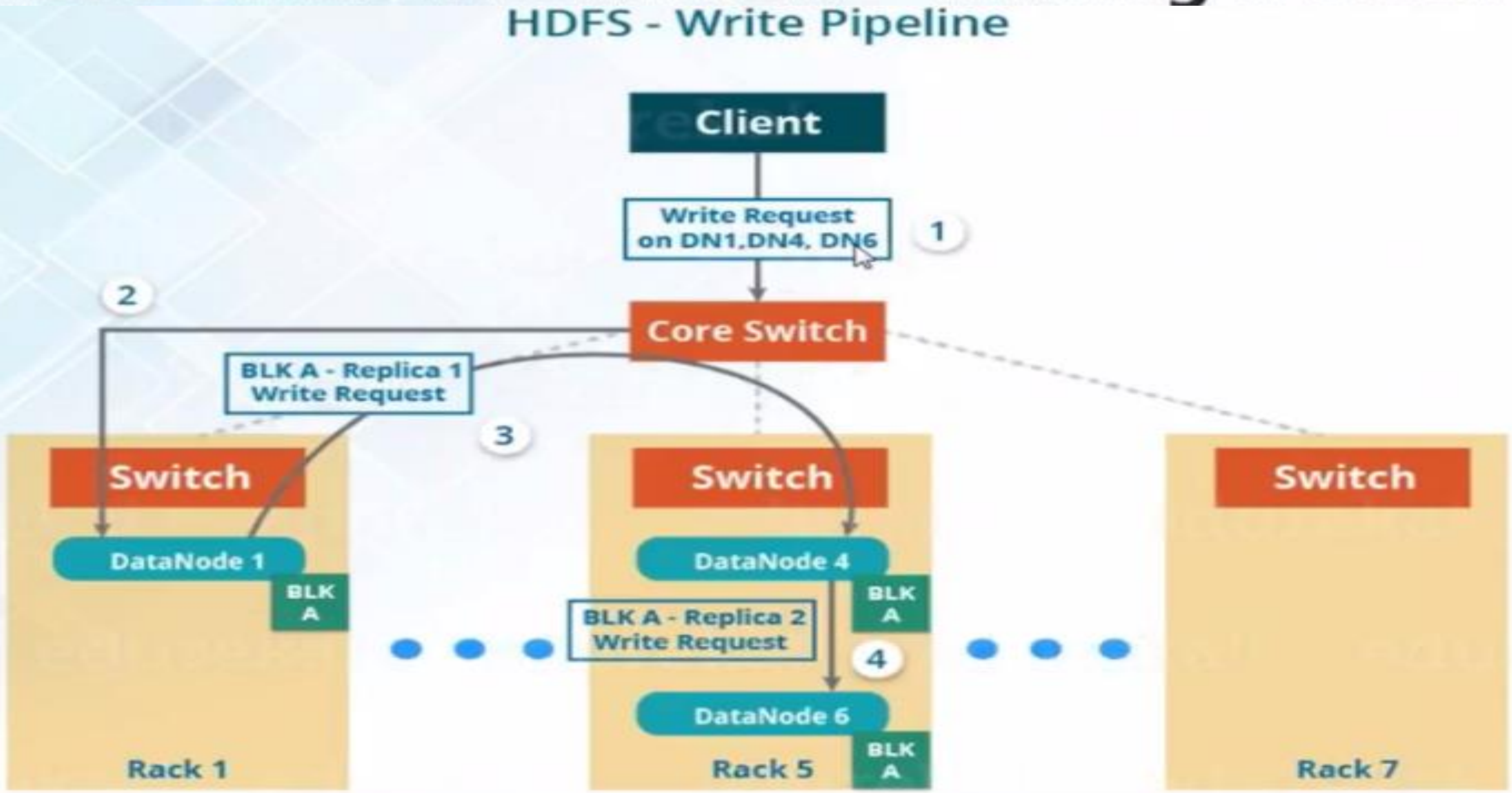
MECHANSIM

HDFS Write Mechanism – Pipeline Setup

Setting up HDFS - Write Pipeline

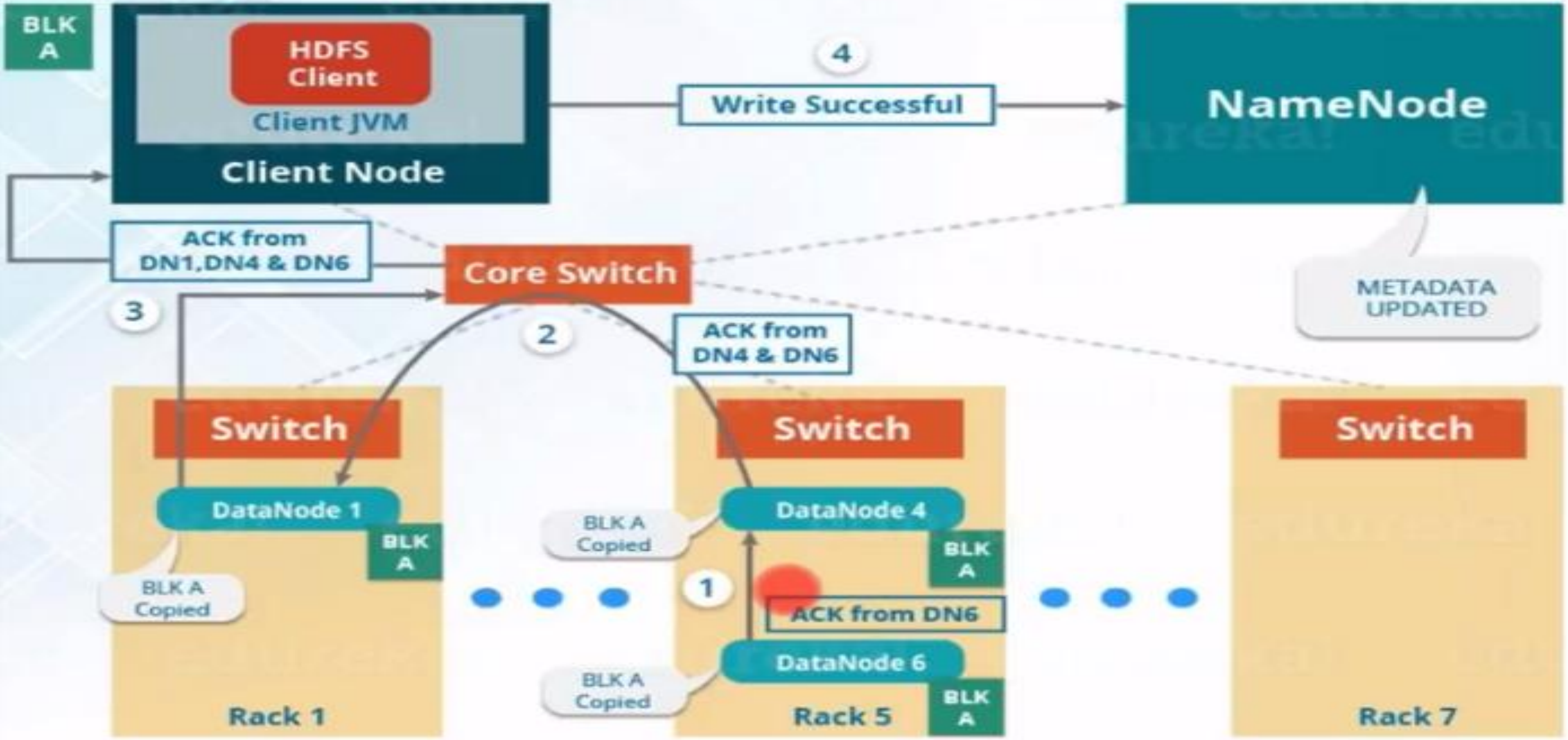


HDFS Write Mechanism – Writing a Block



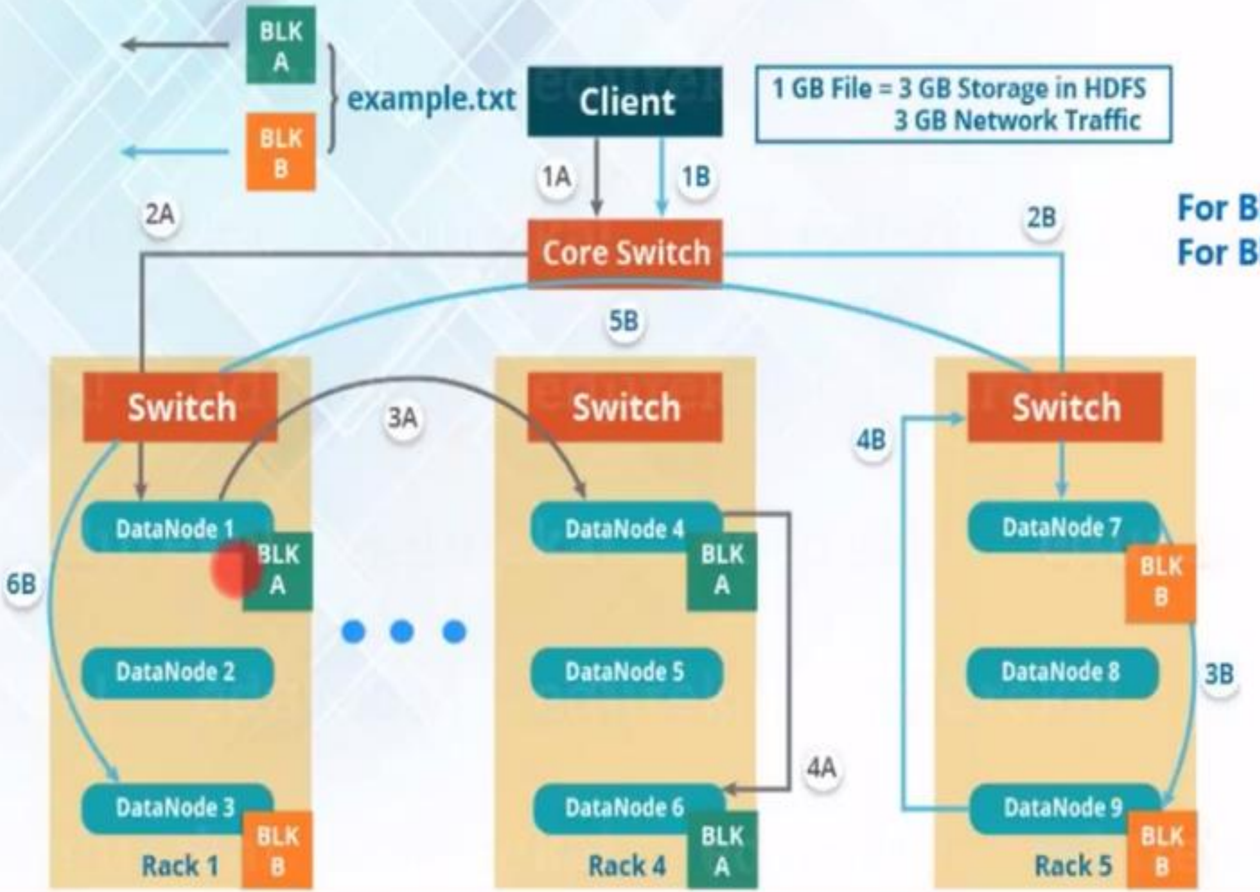
HDFS Write Mechanism - Acknowledgement

Acknowledgement in HDFS - Write



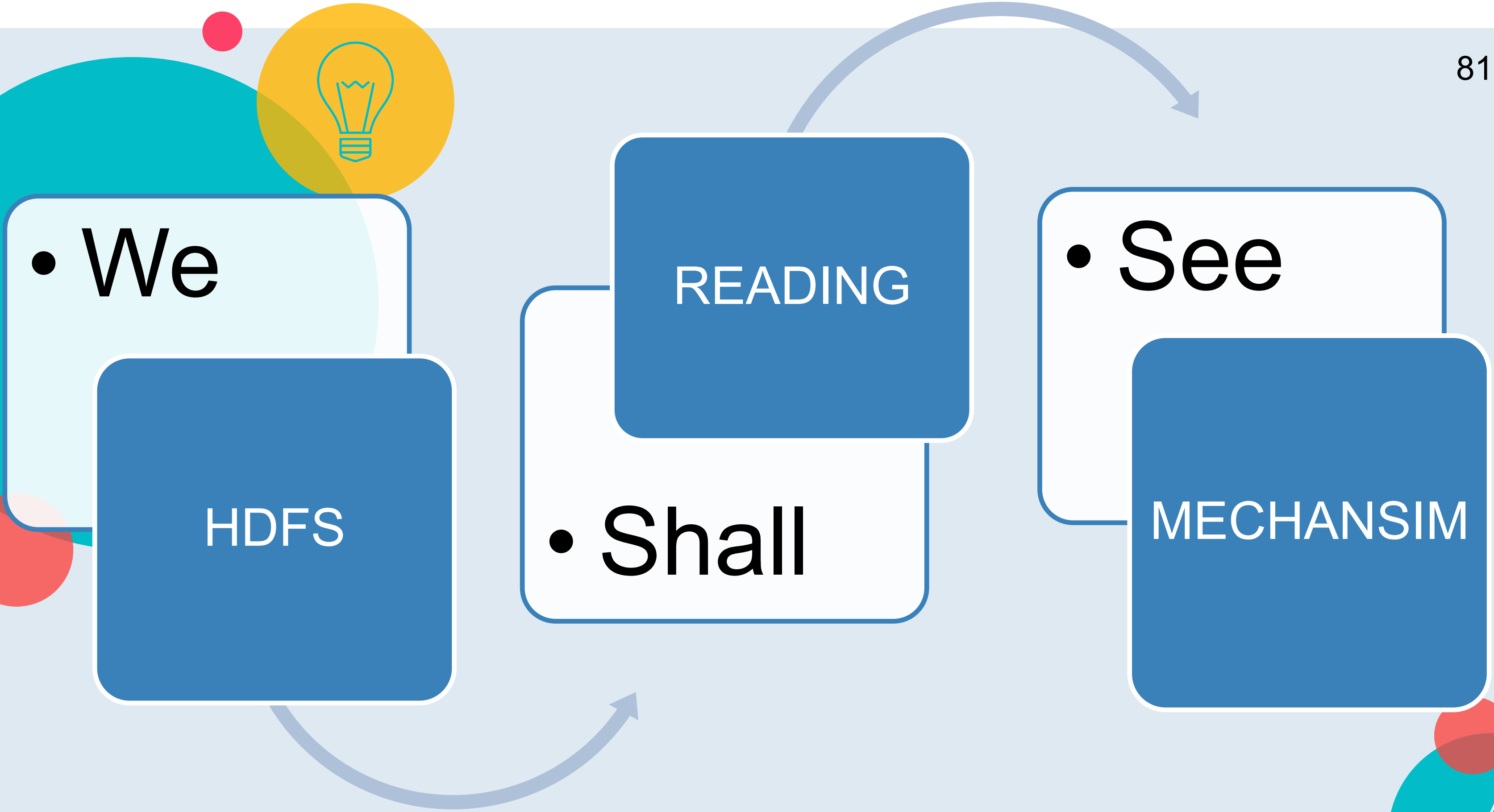
HDFS Multi-Block Write Mechanism

HDFS Multi - Block Write Pipeline



1 GB File = 3 GB Storage in HDFS
3 GB Network Traffic

For Block A: 1A -> 2A -> 3A -> 4A
For Block B: 1B -> 2B -> 3B -> 4B -> 5B -> 6B



- We

HDFS

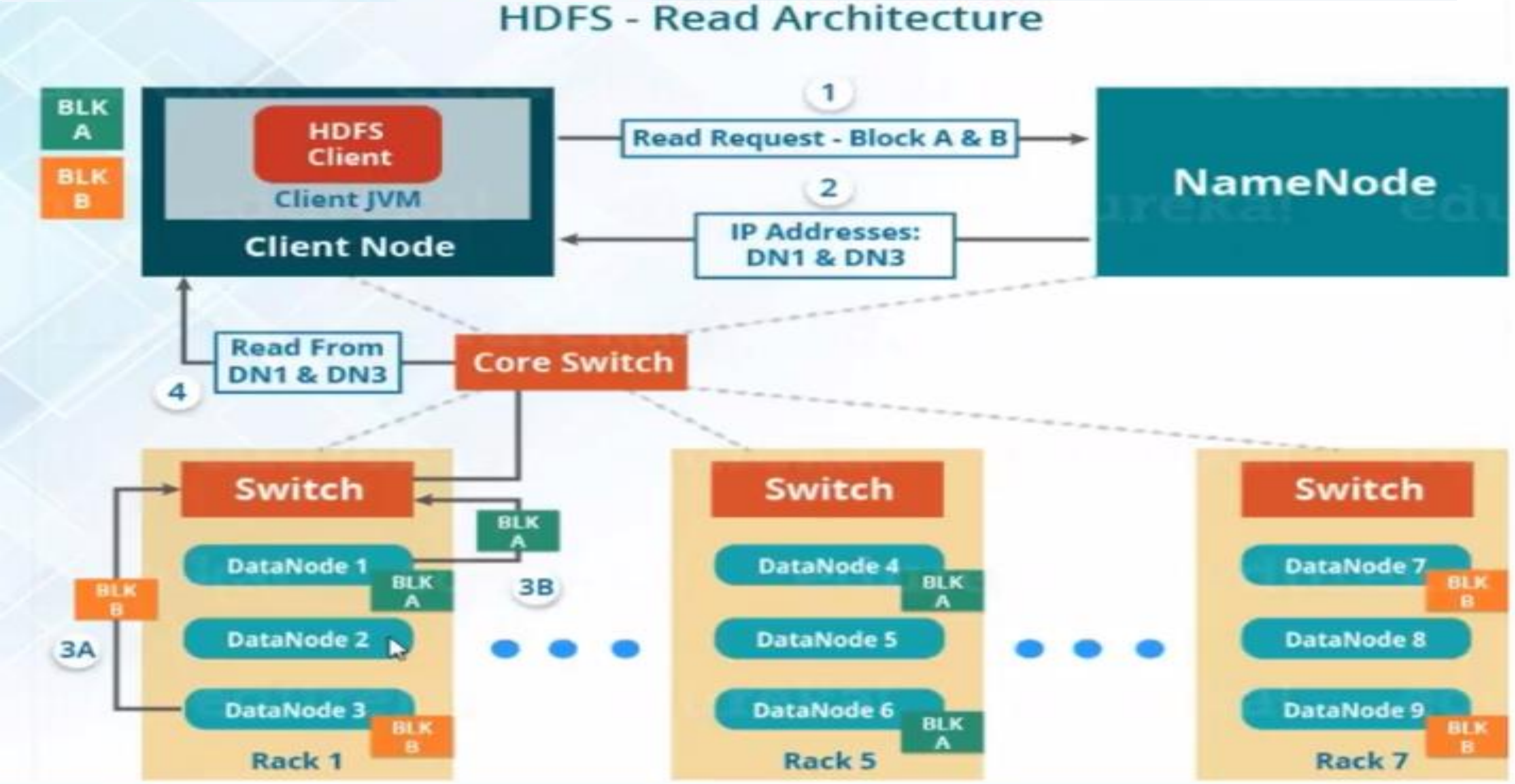
- Shall

READING

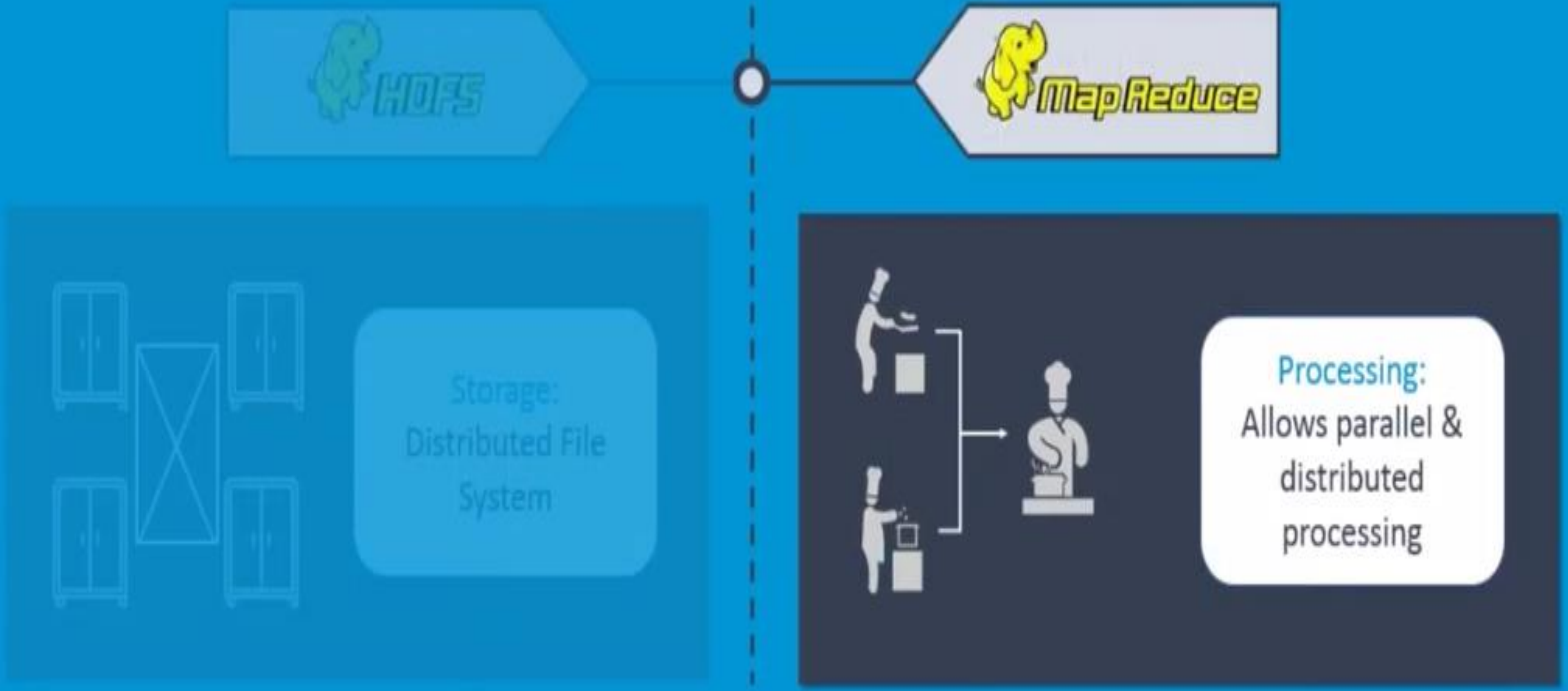
- See

MECHANSIM

HDFS Read Mechanism

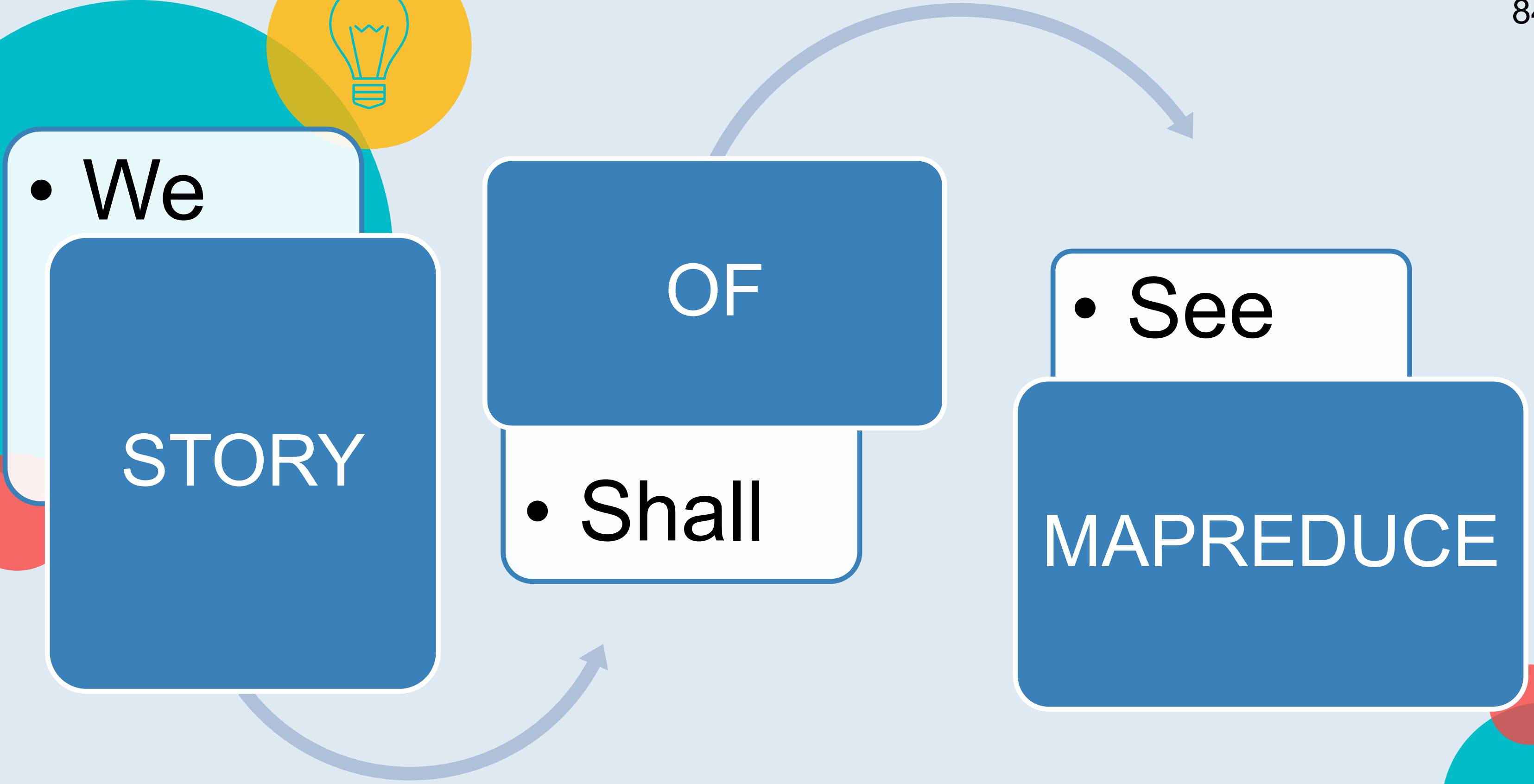


HADOOP CORE COMPONENTS



Storage: Distributed File System

Processing: Allows parallel & distributed processing



STORY

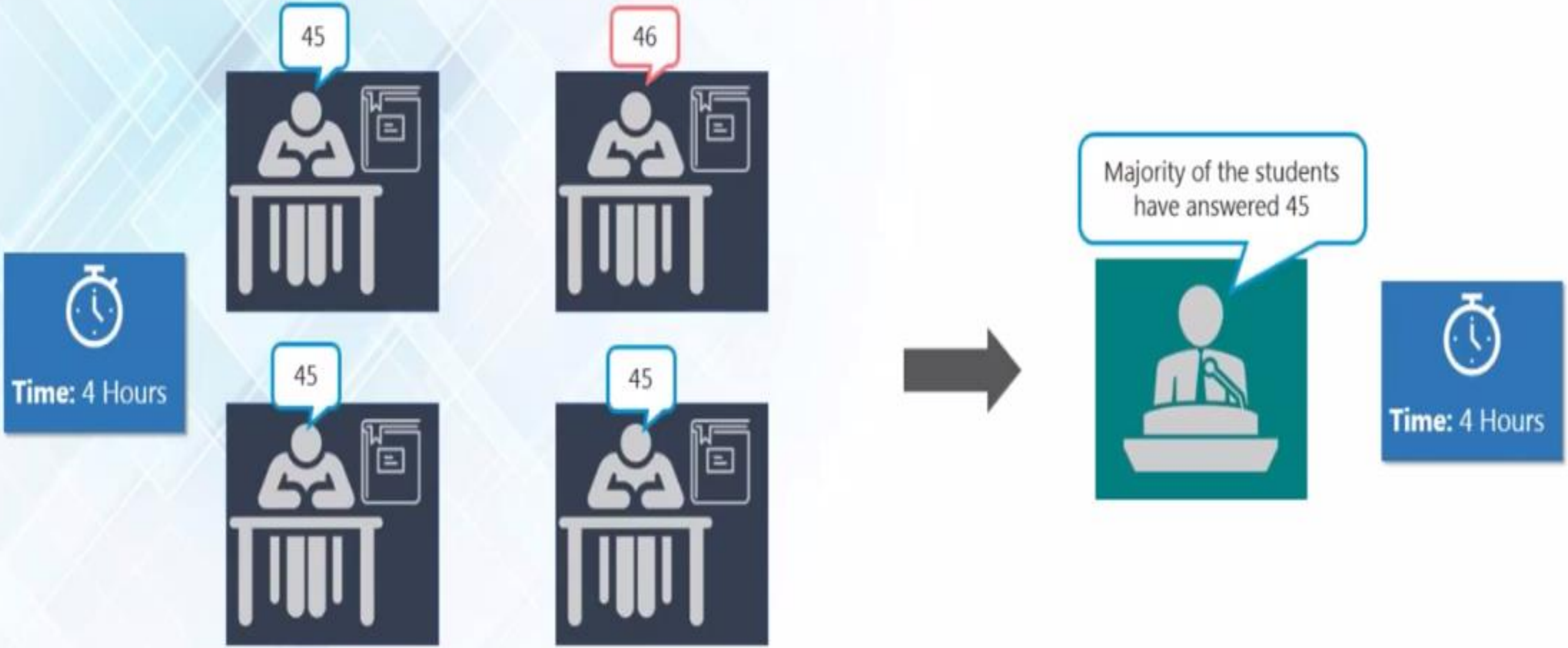
- We

OF

- Shall

MAPREDUCE

- See



Time: 4 Hours

45

46

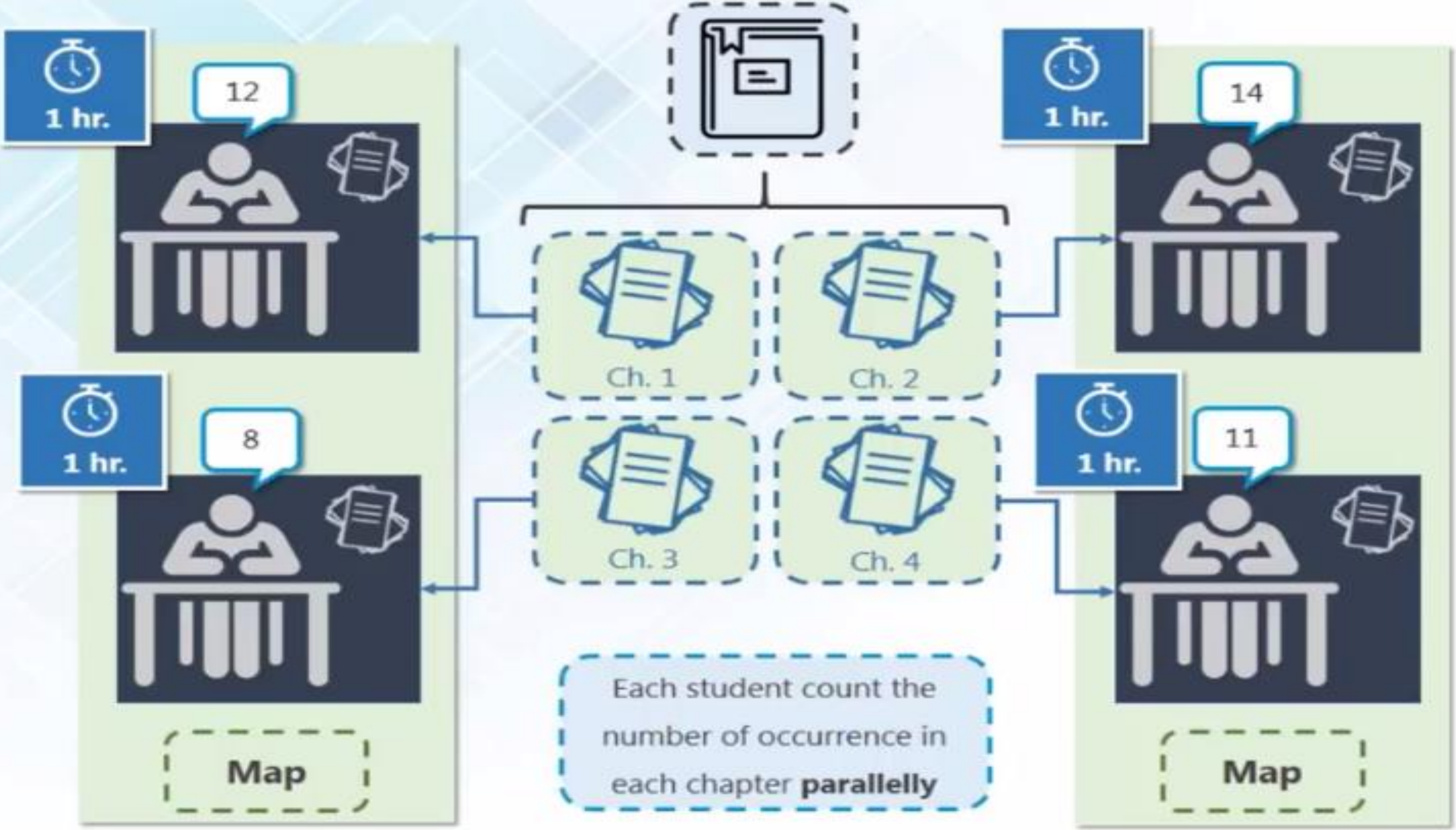
45

45

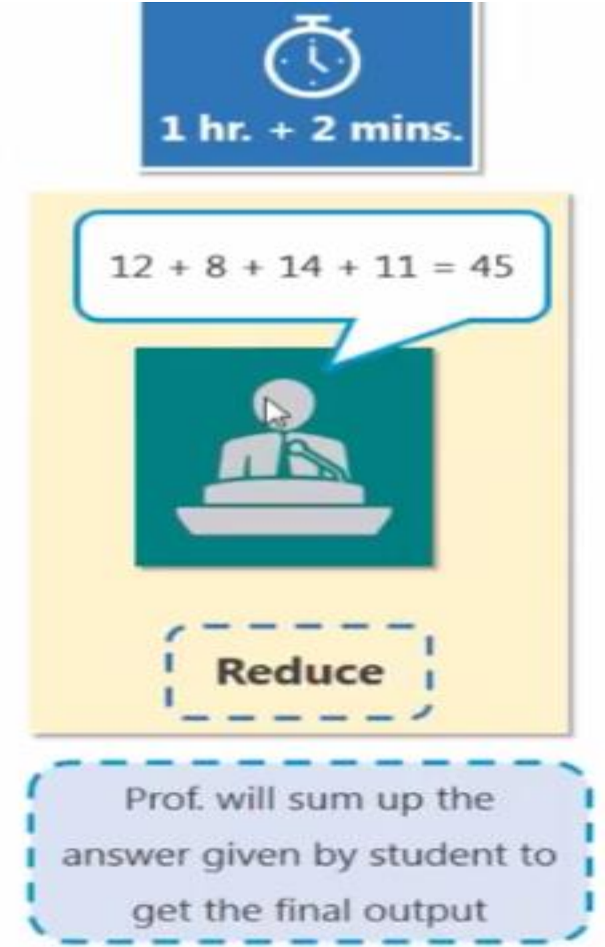
Majority of the students have answered 45

Time: 4 Hours

Each student has to count the occurrence of the word Julius in the book



Each student count the number of occurrence in each chapter **parallelly**

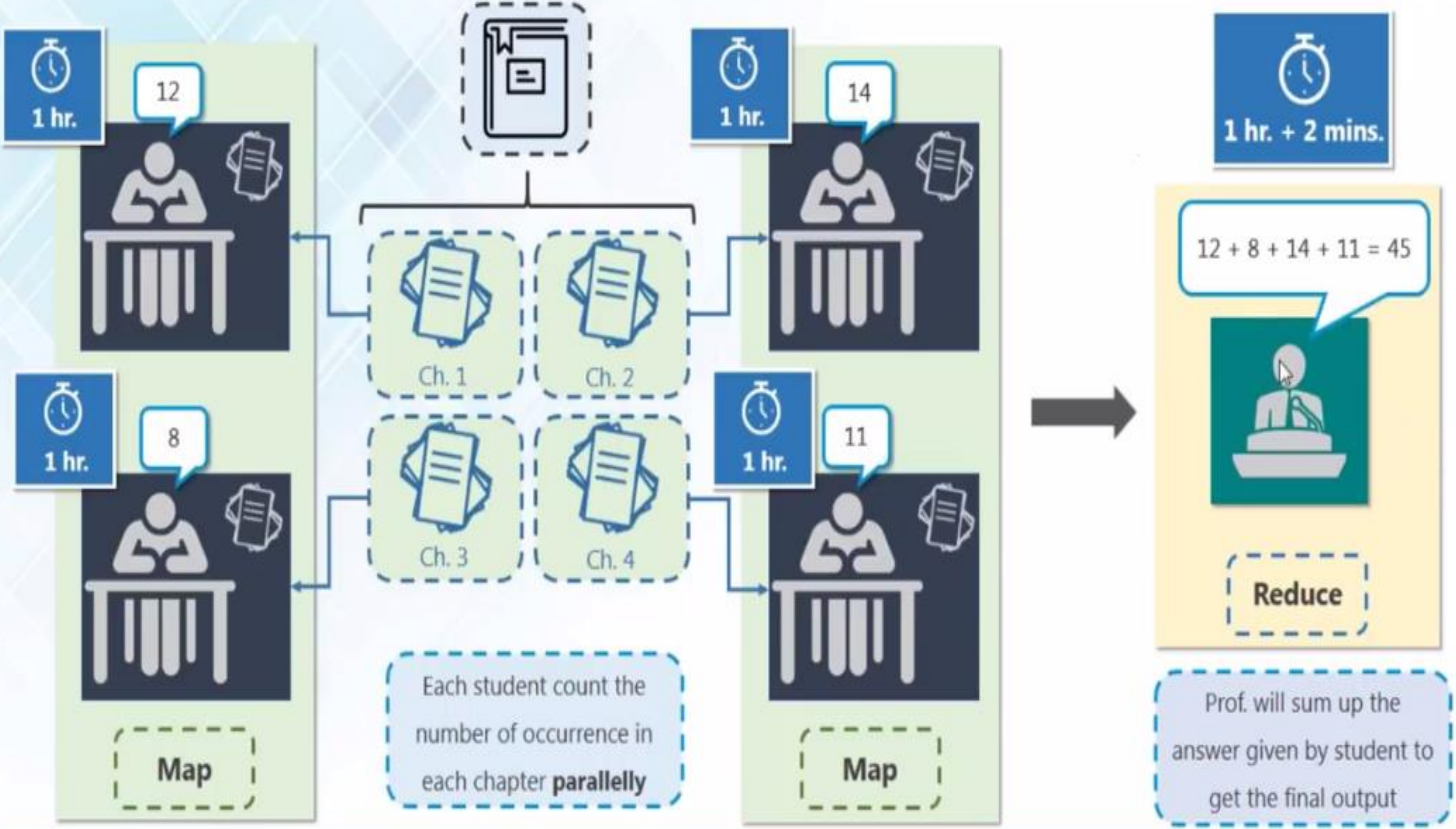


1 hr. + 2 mins.

12 + 8 + 14 + 11 = 45

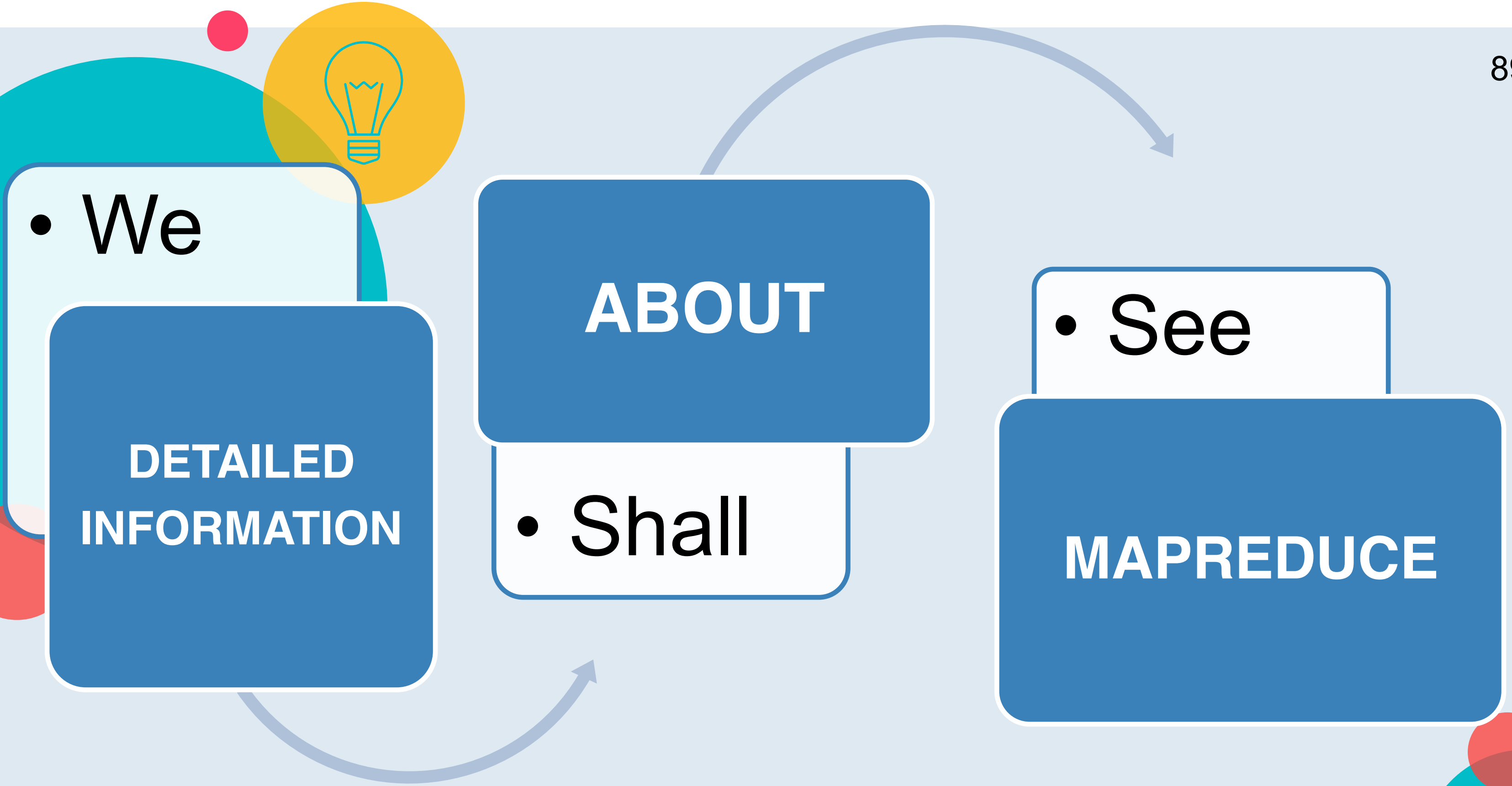
Reduce

Prof. will sum up the answer given by student to get the final output



Each student count the number of occurrence in each chapter **parallelly**

Prof. will sum up the answer given by student to get the final output



- We

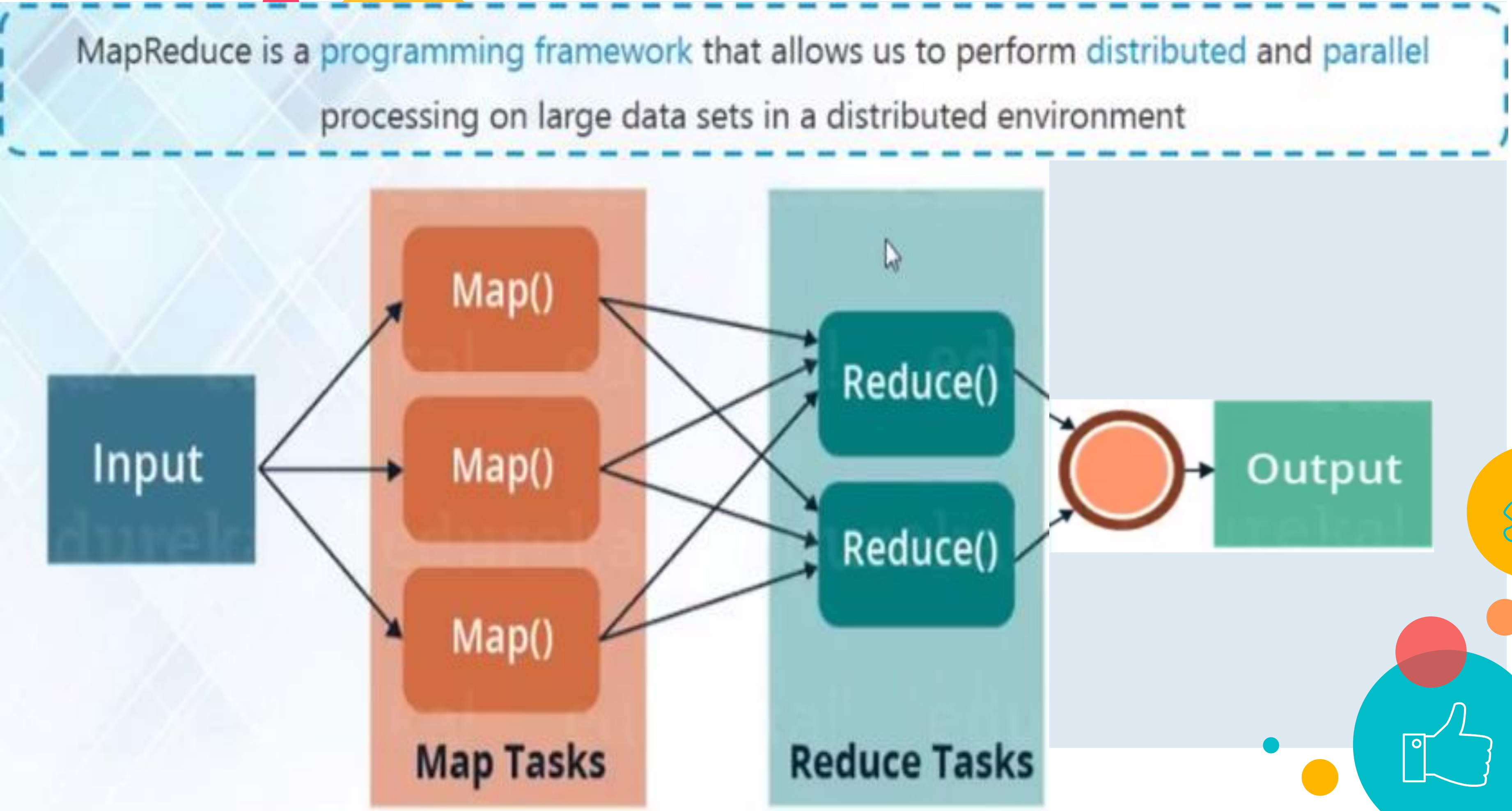
DETAILED INFORMATION

ABOUT

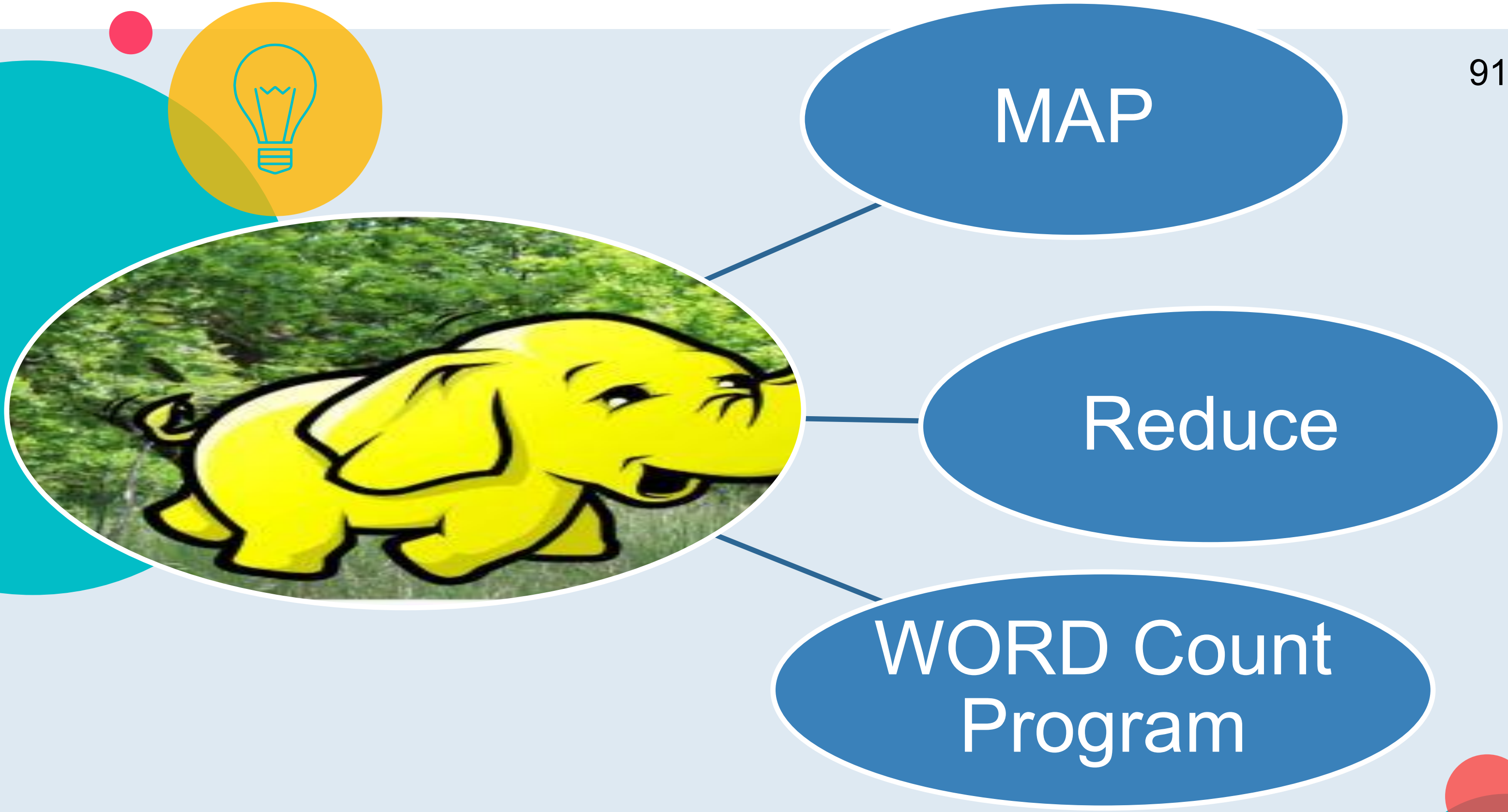
- Shall

- See

MAPREDUCE



MapReduce is a programming framework that allows us to perform distributed and parallel processing on large data sets in a distributed environment

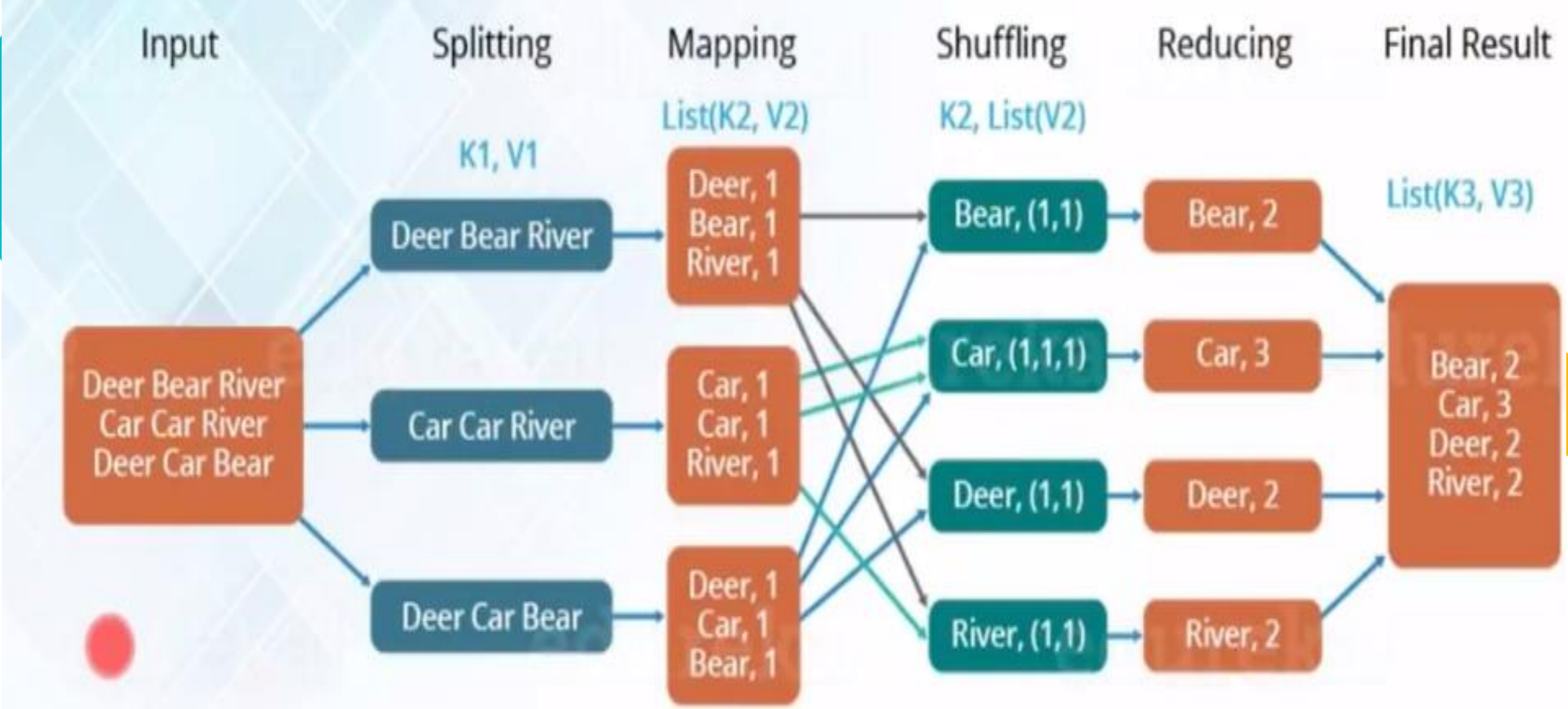


MAP

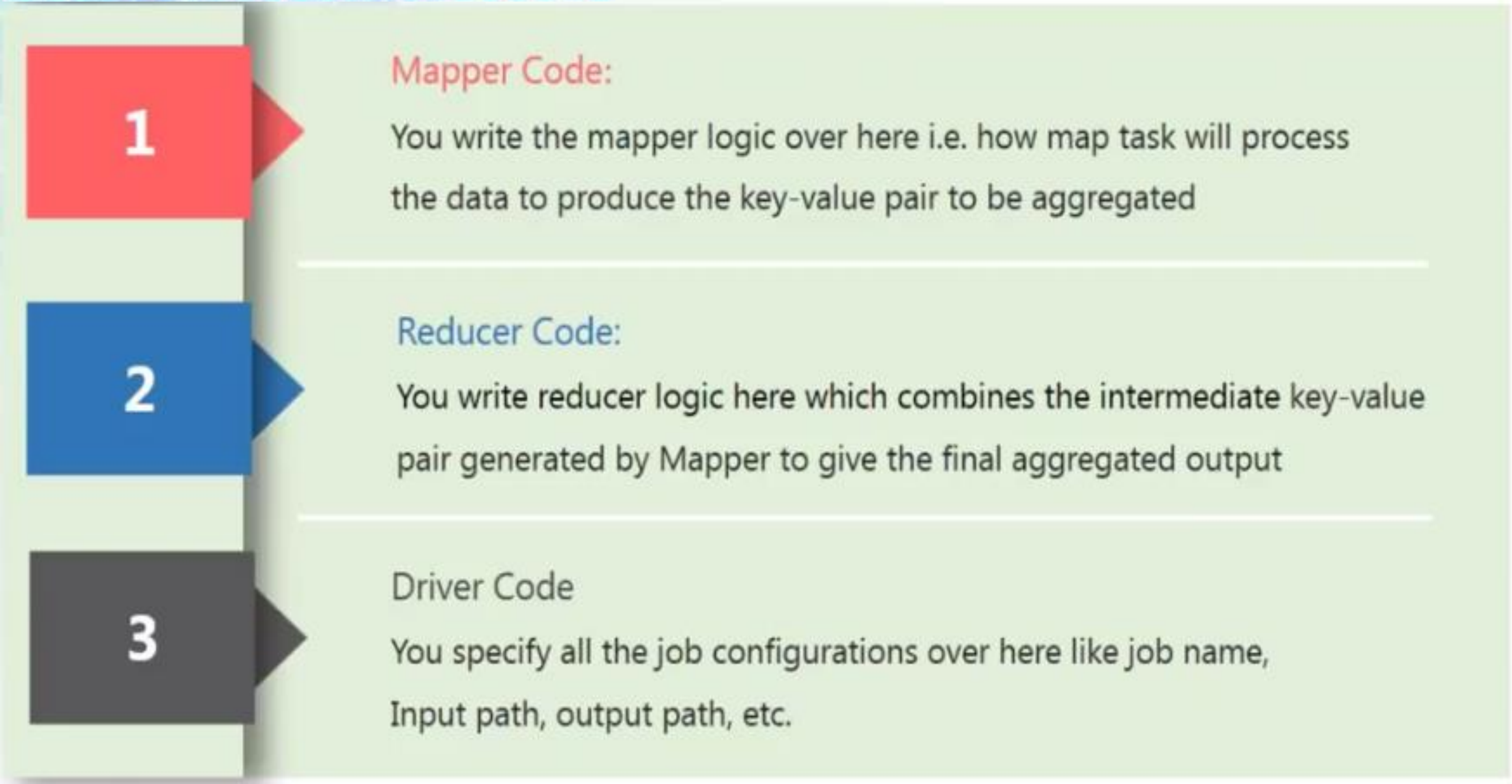
Reduce

WORD Count Program

The Overall MapReduce Word Count Process



Three Major Parts of MapReduce Program:



1

Mapper Code:

You write the mapper logic over here i.e. how map task will process the data to produce the key-value pair to be aggregated

2

Reducer Code:

You write reducer logic here which combines the intermediate key-value pair generated by Mapper to give the final aggregated output

3

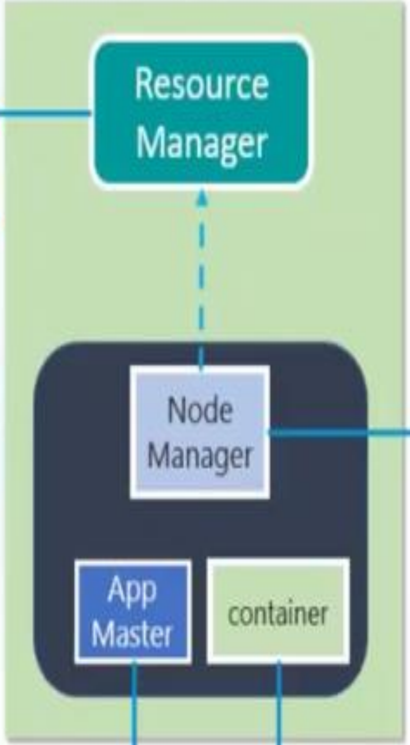
Driver Code

You specify all the job configurations over here like job name, Input path, output path, etc.

YARN Components

ResourceManager:

- Master daemon that manages all other daemons & accepts job submission
- Allocates first container for the AppMaster



NodeManager:

- Responsible for containers, monitoring their resource usage i.e. (cpu, memory, disk, network) & reports the same to RM

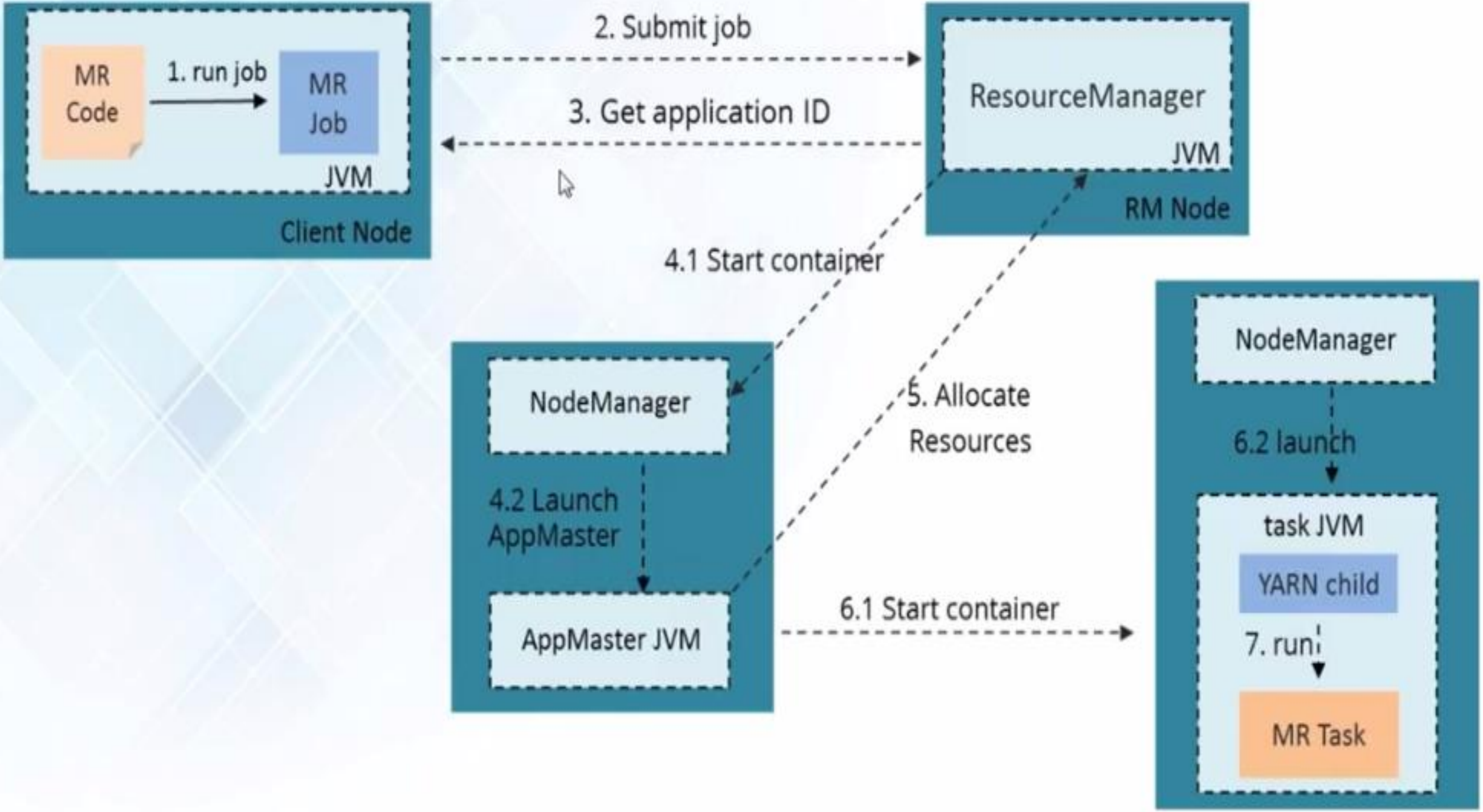
AppMaster:

- One per application
- Coordinates and manages MR Jobs
- Negotiates resources from RM

Container:

- Allocates certain amount of resources (memory, CPU etc.) on a slave node (NM)

MAPREDUCE JOB WORKFLOW



MR Code

1. run job

MR Job

JVM

Client Node

2. Submit job

3. Get application ID

ResourceManager

JVM

RM Node

4.1 Start container

NodeManager

4.2 Launch AppMaster

AppMaster JVM

5. Allocate Resources

6.1 Start container

NodeManager

6.2 launch

task JVM

YARN child

7. run

MR Task