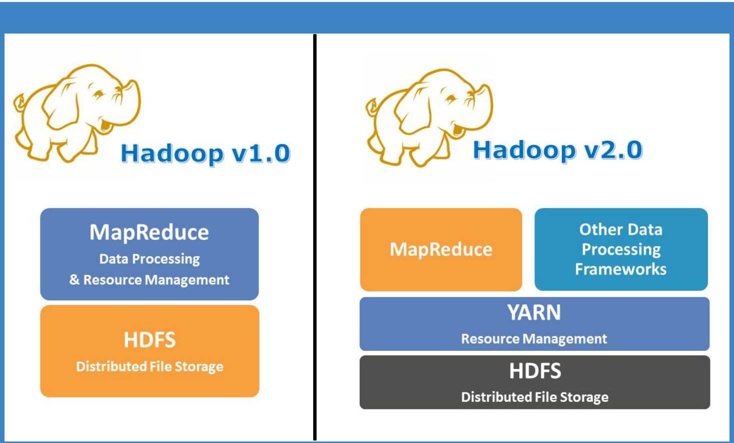


Hadoop Map Reduce and YARN
Lecture 4

Sharun Akter khushbu

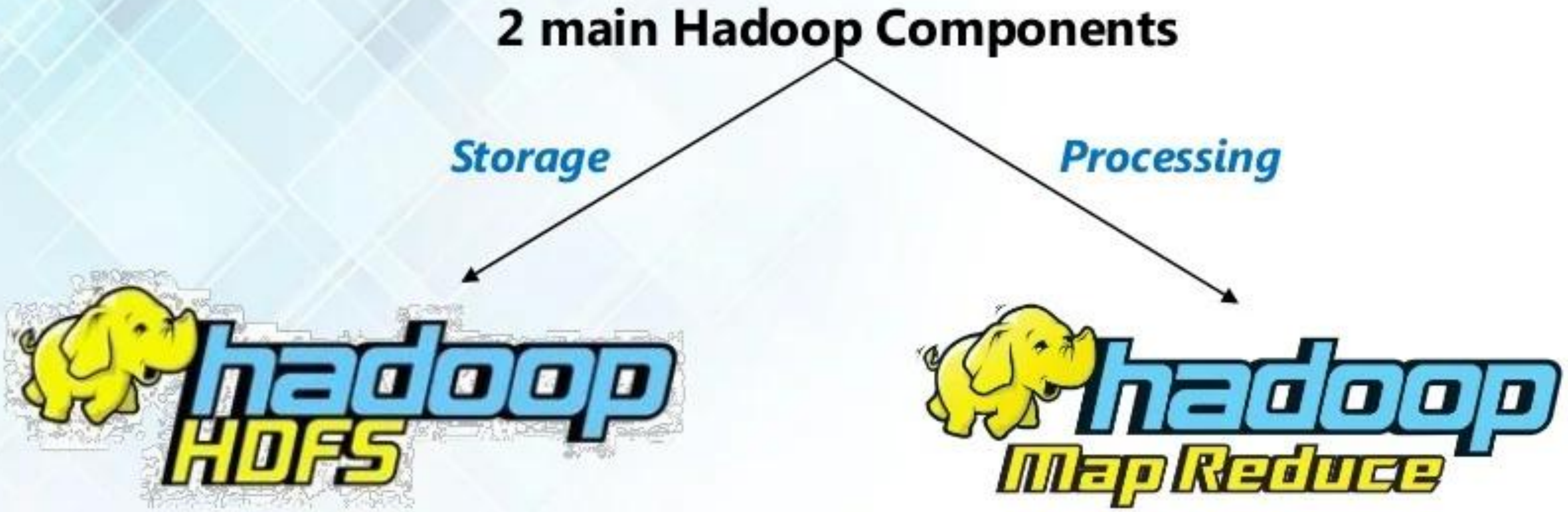
Lecturer, Dept.of CSE

Daffodil International University

Agenda for today's Session

1. What is Hadoop MapReduce?
2. MapReduce In Nutshell
3. Advantages of MapReduce
4. Hadoop MapReduce Approach with an Example
5. Hadoop MapReduce/YARN Components
6. YARN With MapReduce
7. Yarn Application Workflow
8. MapReduce Program with Hands On

Hadoop Components



2 main Hadoop Components

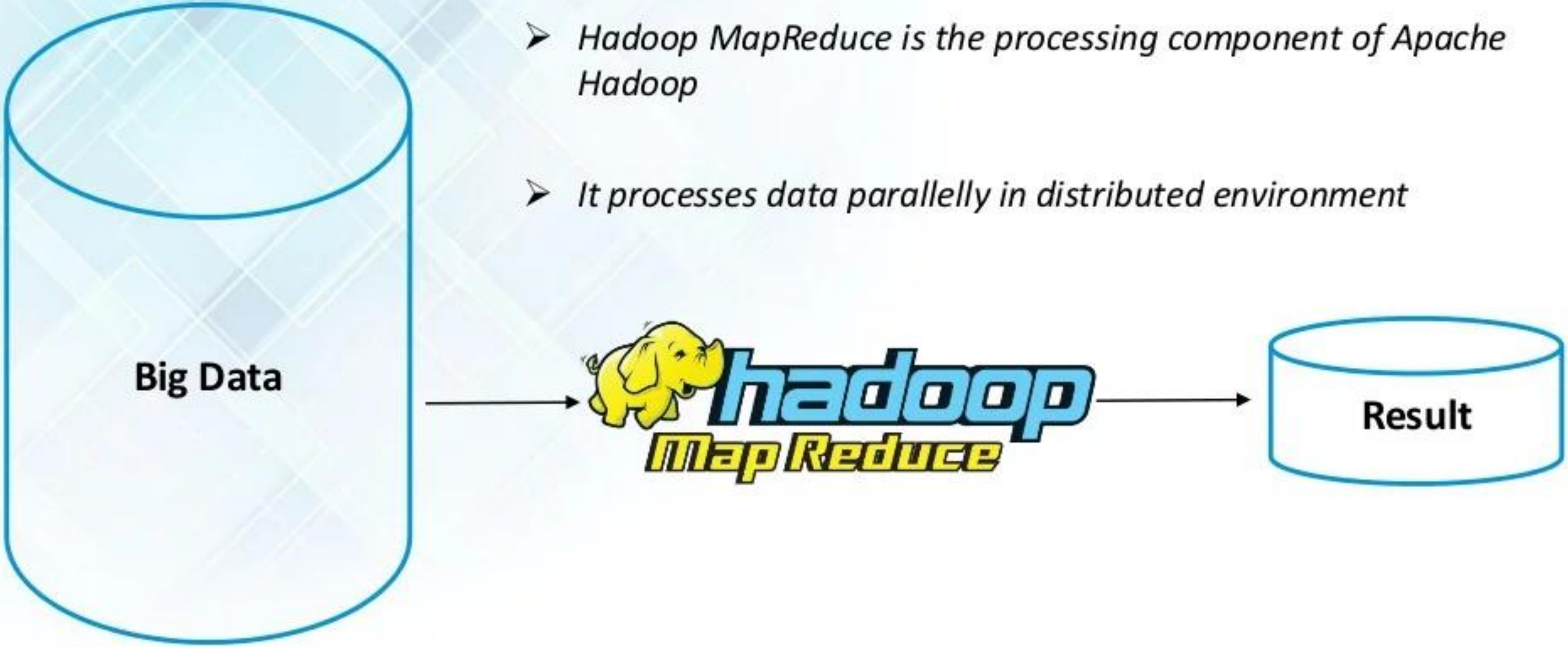
Storage

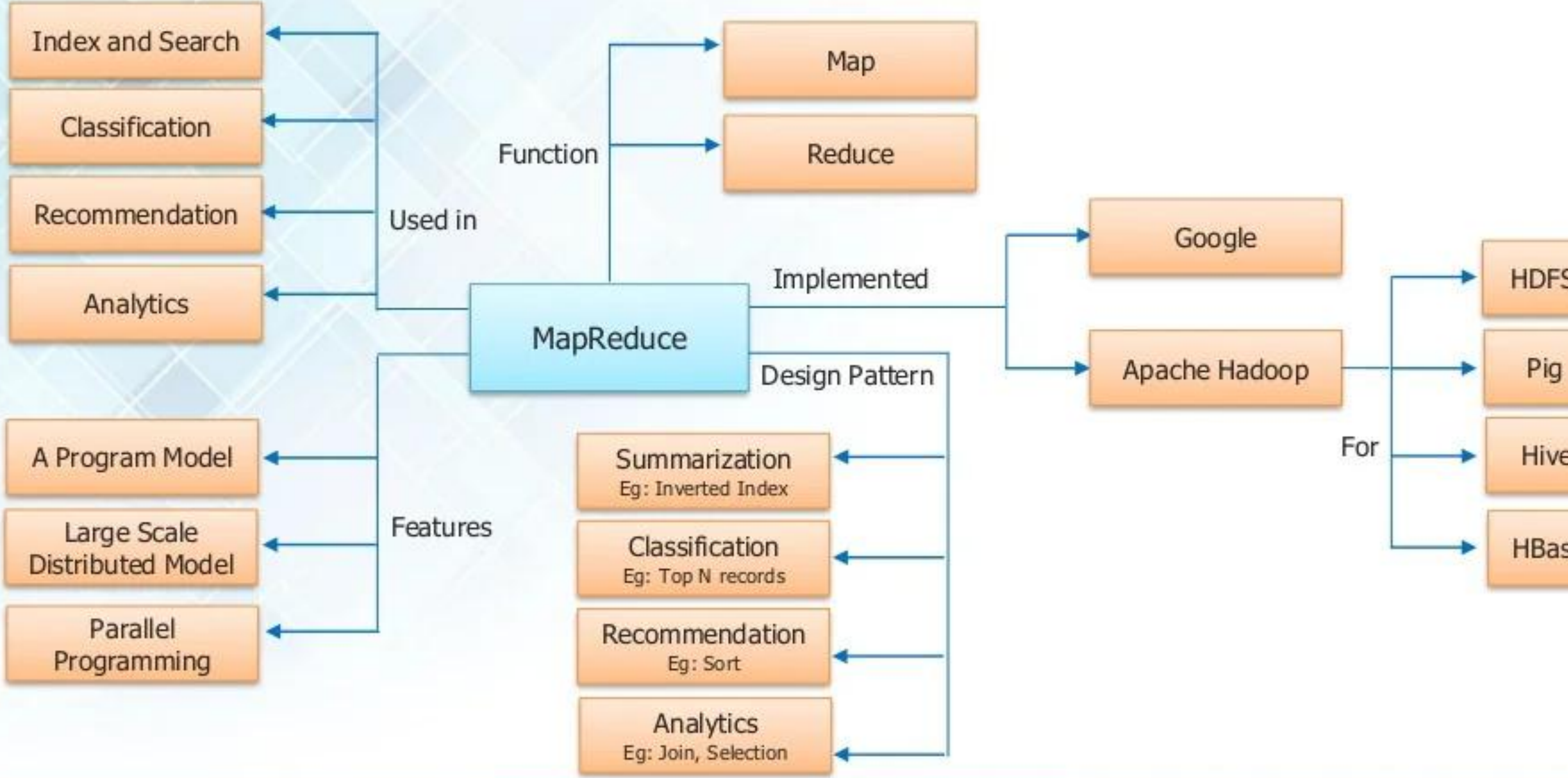
Processing

MapReduce: Data Processing Using Programming

- *Hadoop MapReduce is the processing component of Apache Hadoop*

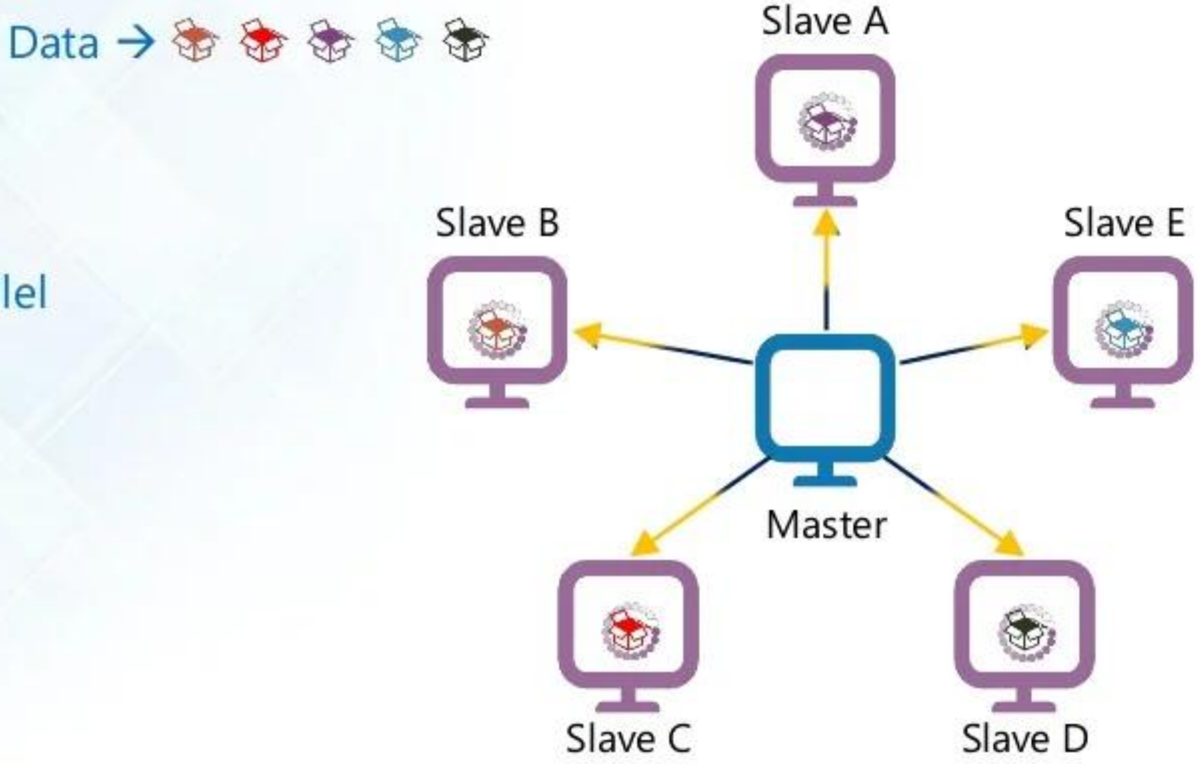
- *It processes data parallelly in distributed environment*




MapReduce
Used in
Index and Search
Classification
Recommendation
Analytics
Function
Map
Reduce
Implemented
Google
Apache Hadoop
For
HDFS
Pig
Hive
HBas
Design Pattern
Summarization
Eg: Inverted Index
Classification
Eg: Top N records
Recommendation
Eg: Sort
Analytics
Eg: Join, Selection
Features
A Program Model
Large Scale Distributed Model
Parallel Programming

2 Biggest Advantages of MapReduce

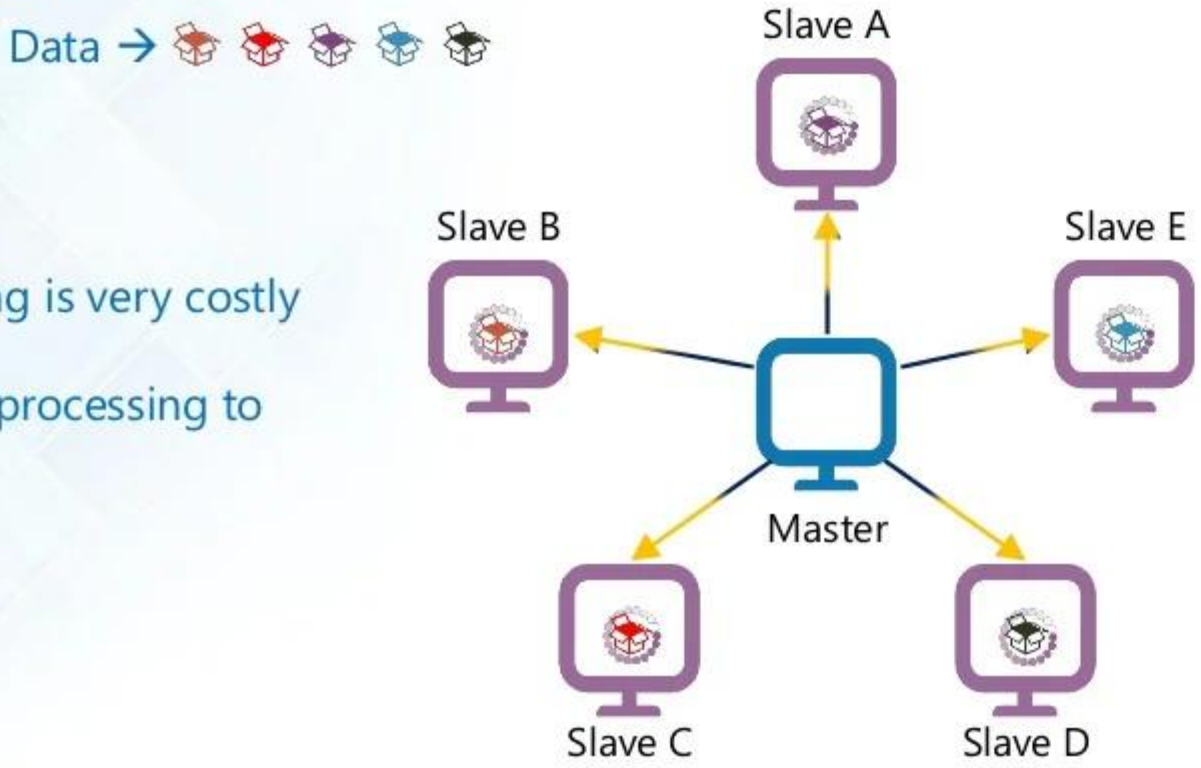
Advantage 1: Parallel Processing



- ➢ Data is processed in parallel
- ➢ Processing becomes fast

Advantage 2: Data Locality - Processing to Storage

- Moving Data to processing is very costly
- In MapReduce, we move processing to Data



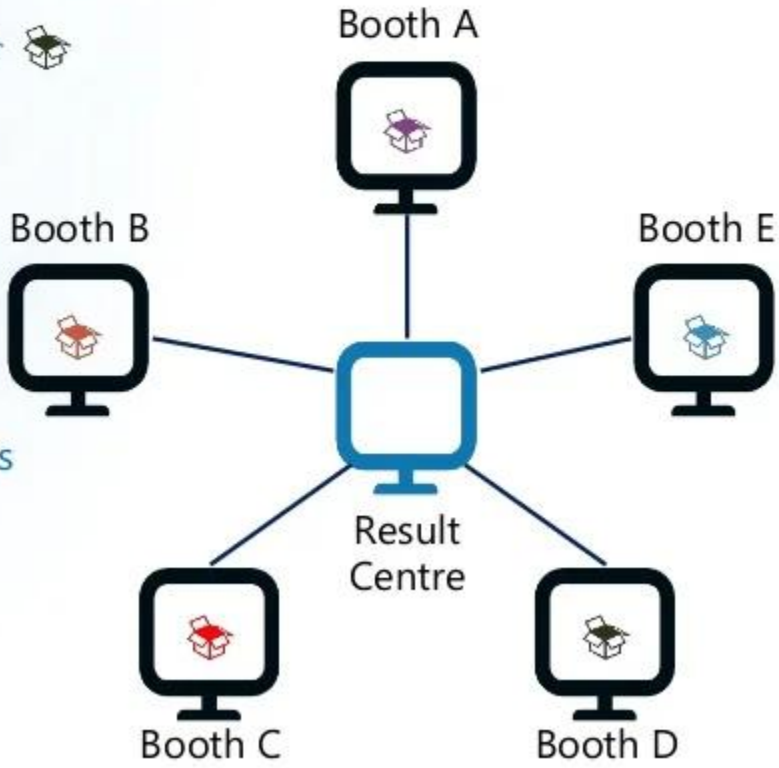
Traditional vs MapReduce Way

Election Votes Counting

Data →

Election Votes Casting

- Votes is stored at different Booths
- Result Centre has the details of all the Booths

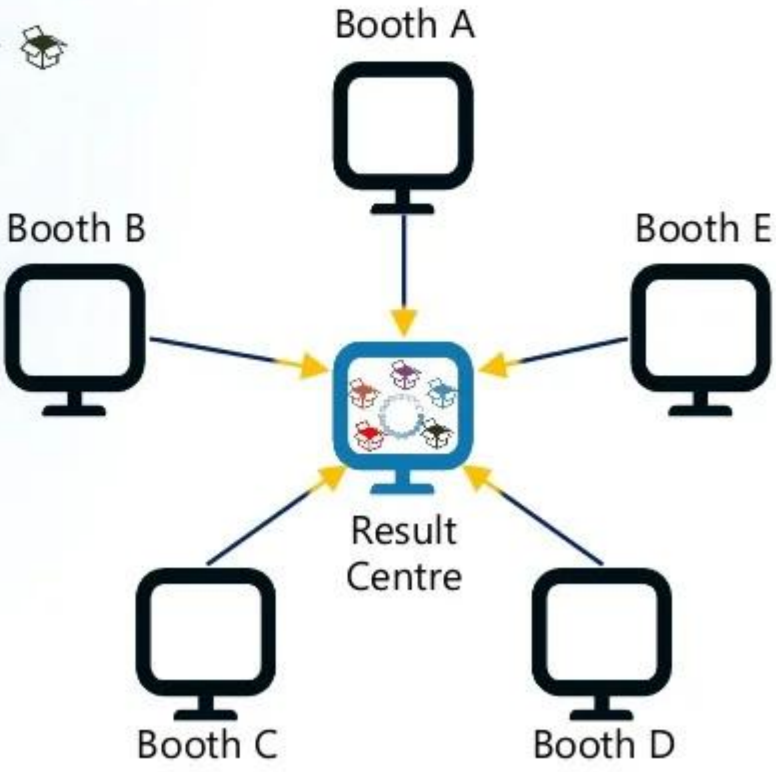


Election Votes Counting – Traditional Way

Data →

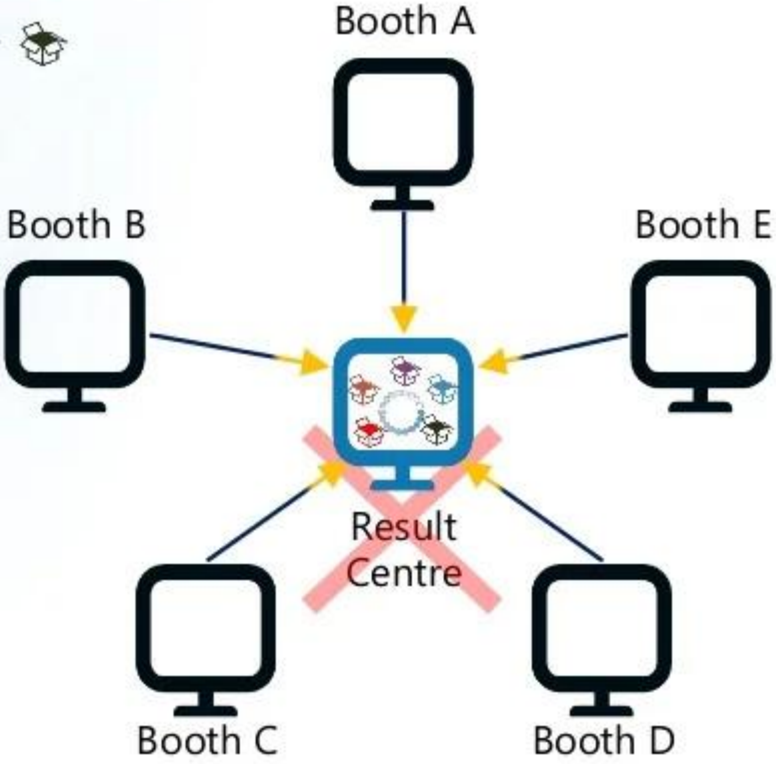
Counting – Traditional Approach

- ➢ Votes are moved to Result Centre for counting
- ➢ Moving all the votes to Centre is costly
- ➢ Result Centre is over-burdened
- ➢ Counting takes time



Hadoop MapReduce To the Rescue!

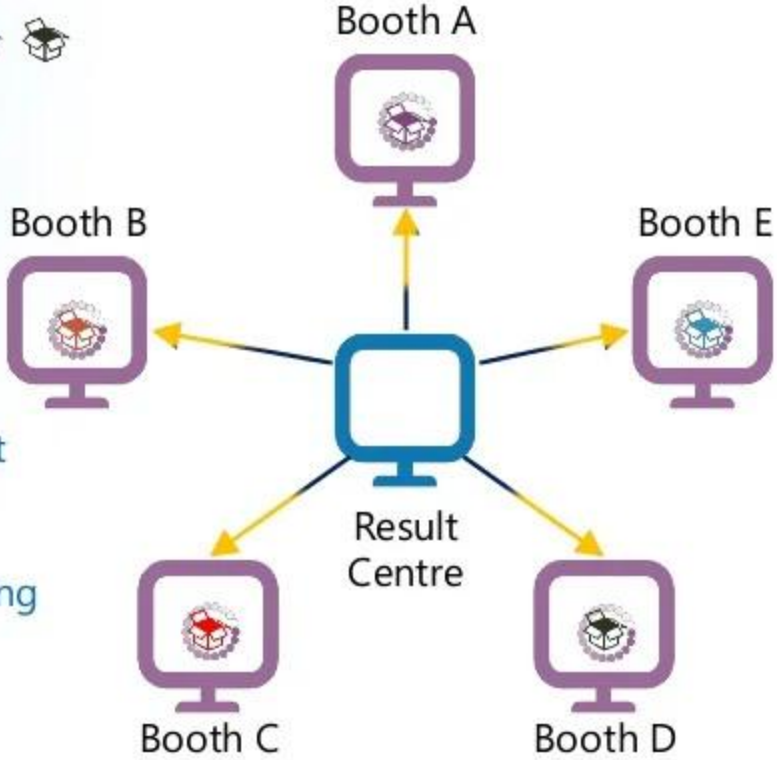
✗ Hadoop MapReduce Doesn't Follow This Approach



Election Votes Counting – MapReduce Way

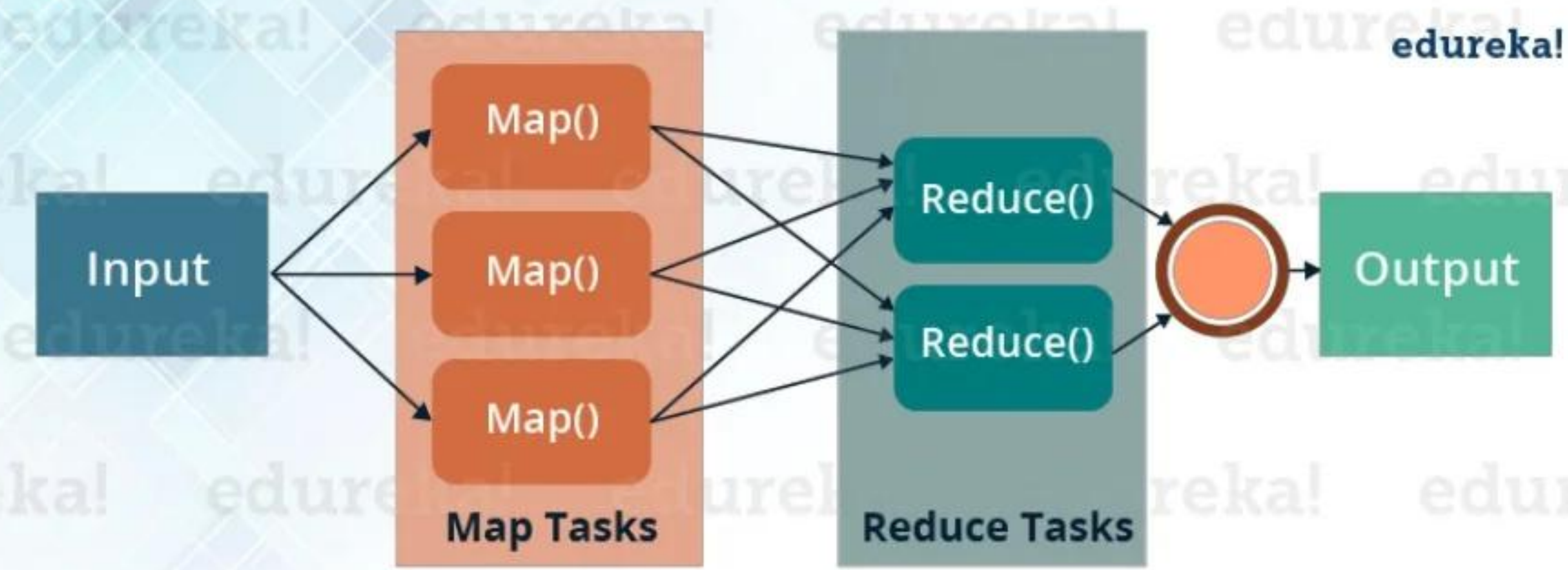
Counting – MapReduce Approach

- Votes are counted at individual booths
- Booth-wise results are sent back to the result centre
- Final Result is declared easily and quickly using this way

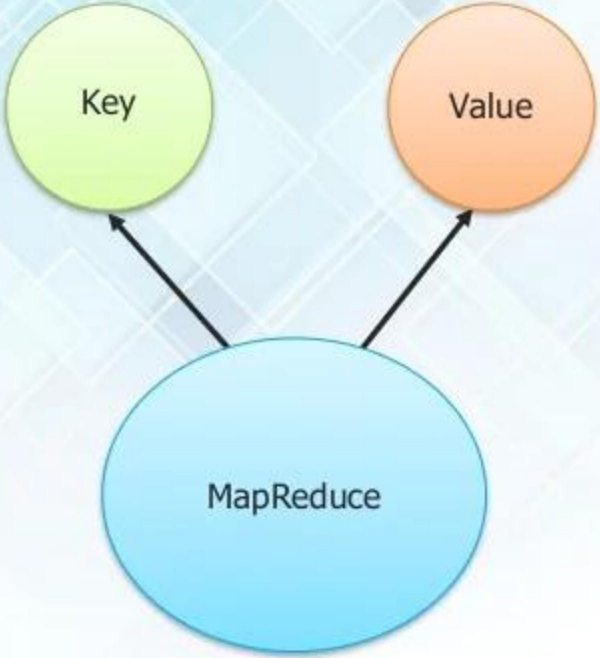


MapReduce In Detail

MapReduce Way



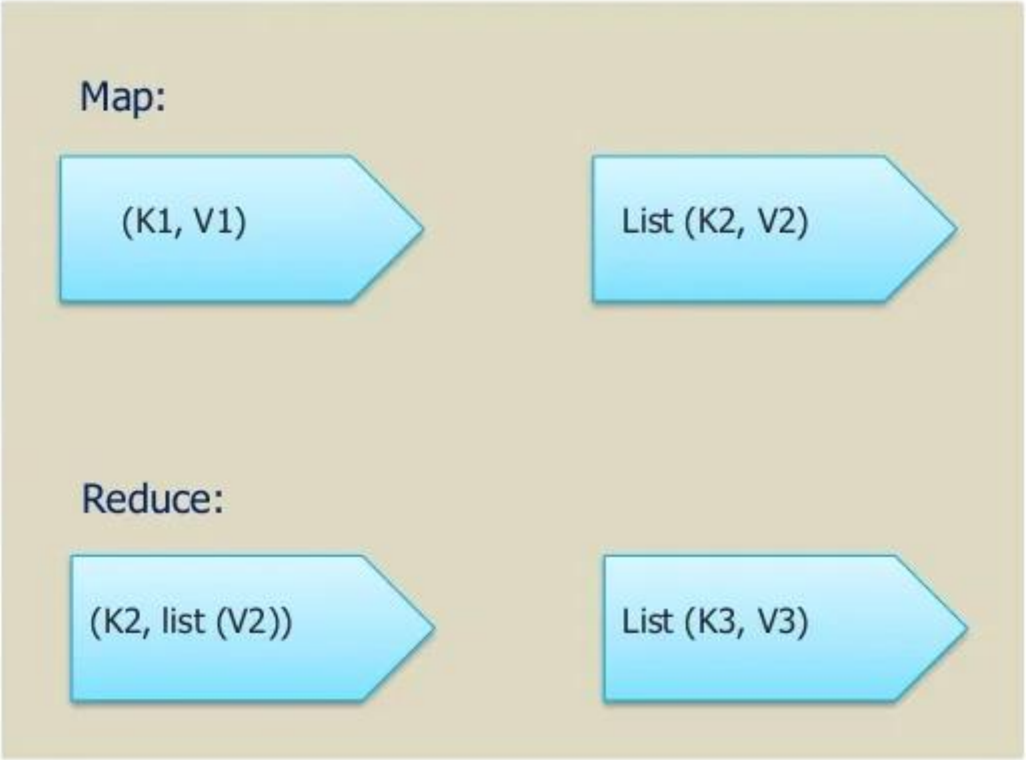
Anatomy of a MapReduce Program



Key

Value

MapReduce



Map:

(K1, V1)

List (K2, V2)

Reduce:

(K2, list (V2))

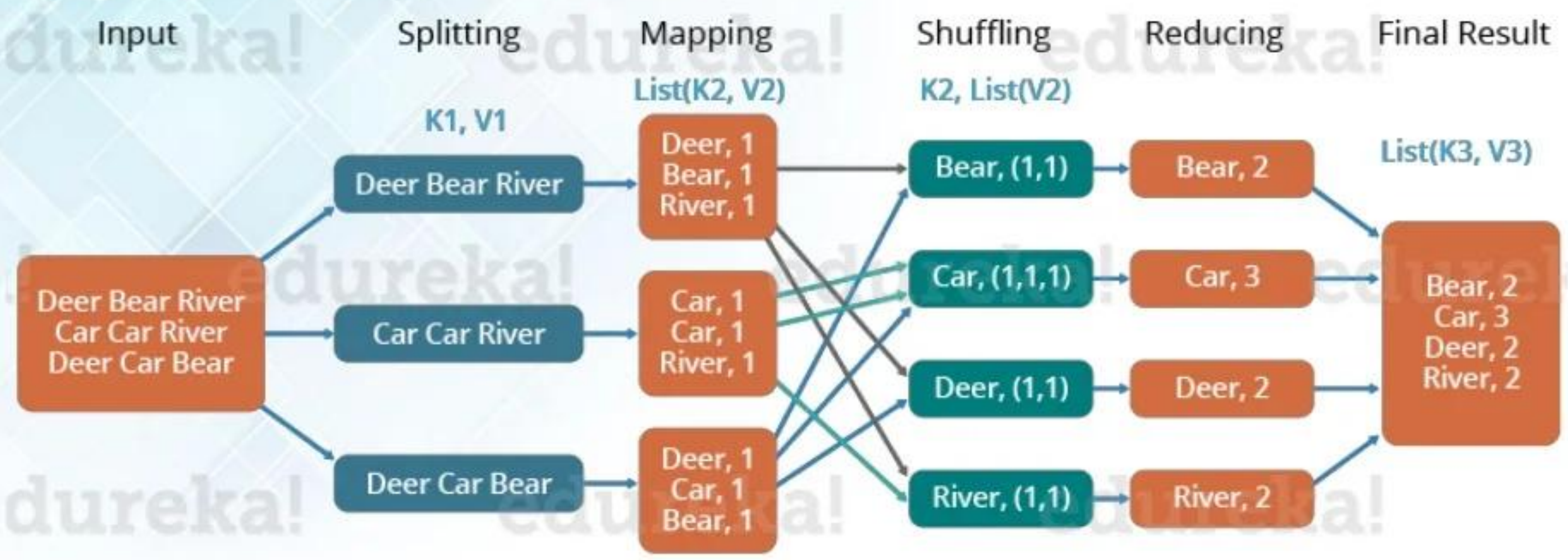
List (K3, V3)

Let us take an example to understand

MapReduce Way

lapReduce Way – Word Count Process

The Overall MapReduce Word Count Process



Input

Deer Bear River
Car Car River
Deer Car Bear

Splitting

K1, V1

Deer Bear River

Car Car River

Deer Car Bear

Mapping

List(K2, V2)

Deer, 1
Bear, 1
River, 1

Car, 1
Car, 1
River, 1

Deer, 1
Car, 1
Bear, 1

Shuffling

K2, List(V2)

Bear, (1,1)

Car, (1,1,1)

Deer, (1,1)

River, (1,1)

Reducing

Bear, 2

Car, 3

Deer, 2

River, 2

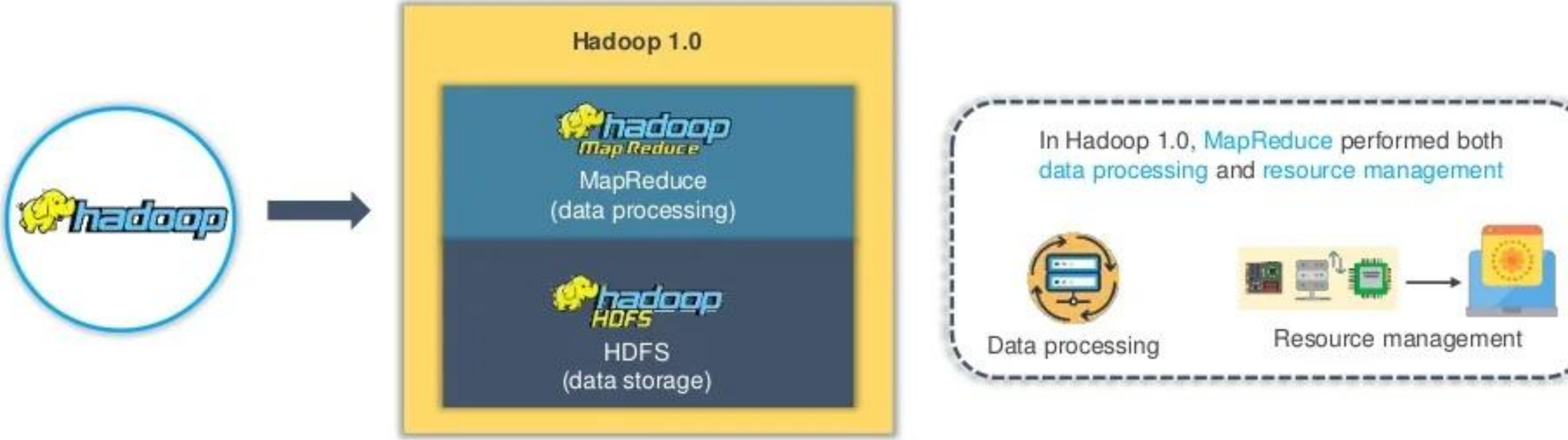
Final Result

List(K3, V3)

Bear, 2
Car, 3
Deer, 2
River, 2

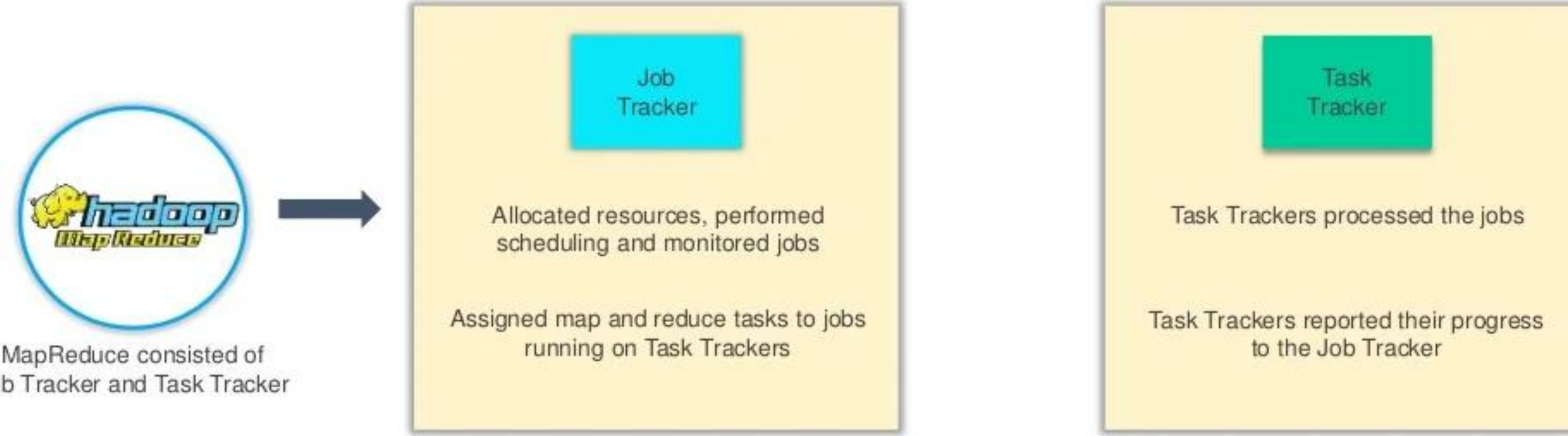
Hadoop 1.0 (MR 1)

Hadoop 1.0 (MR 1)



In Hadoop 1.0, MapReduce performed both data processing and resource management

Hadoop 1.0 (MR 1)



MapReduce consisted of Job Tracker and Task Tracker

Job Tracker

Allocated resources, performed scheduling and monitored jobs

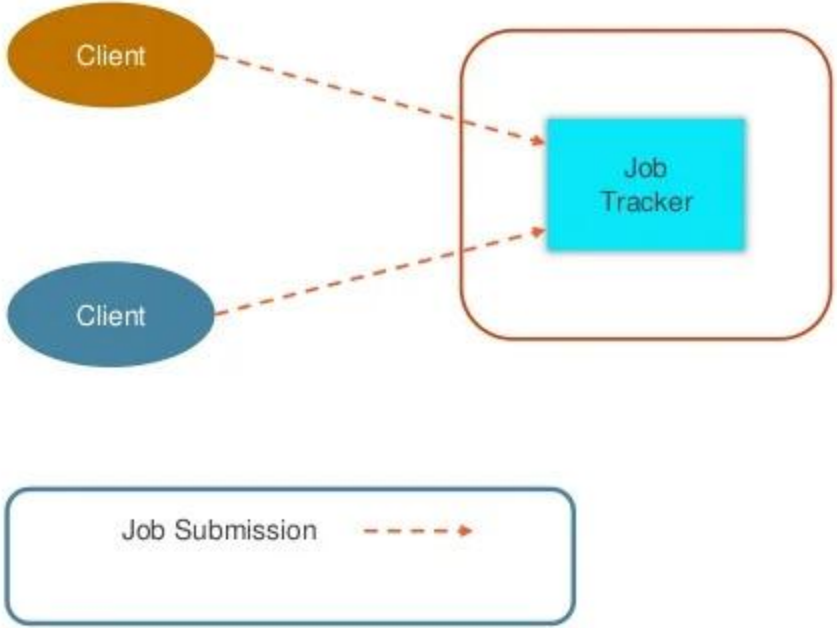
Assigned map and reduce tasks to jobs running on Task Trackers

Task Tracker

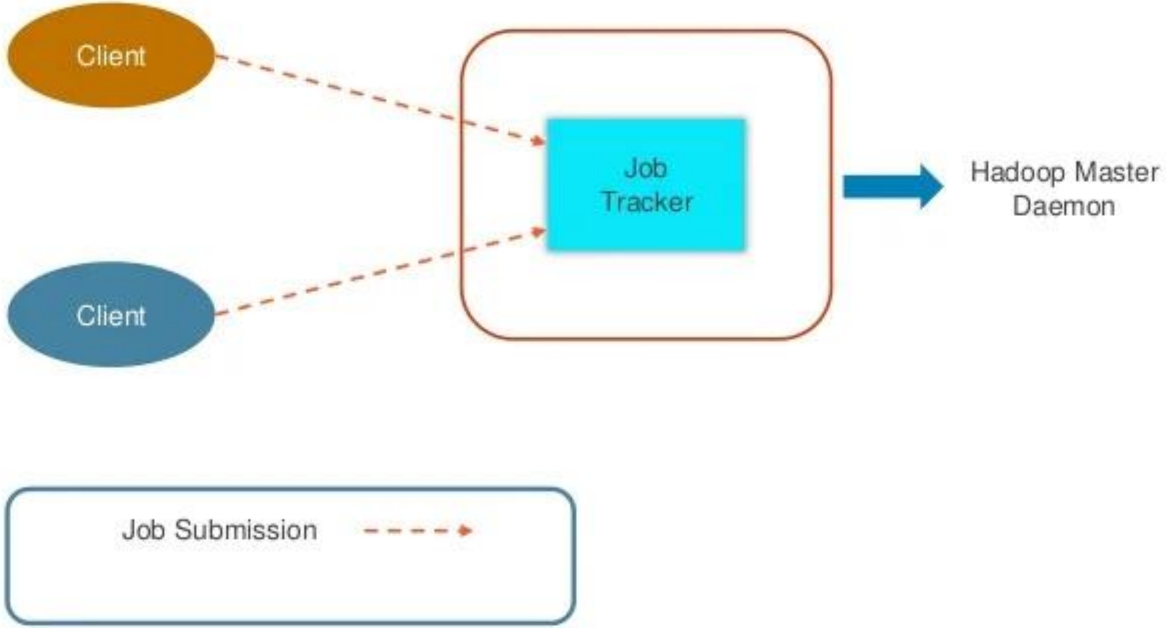
Task Trackers processed the jobs

Task Trackers reported their progress to the Job Tracker

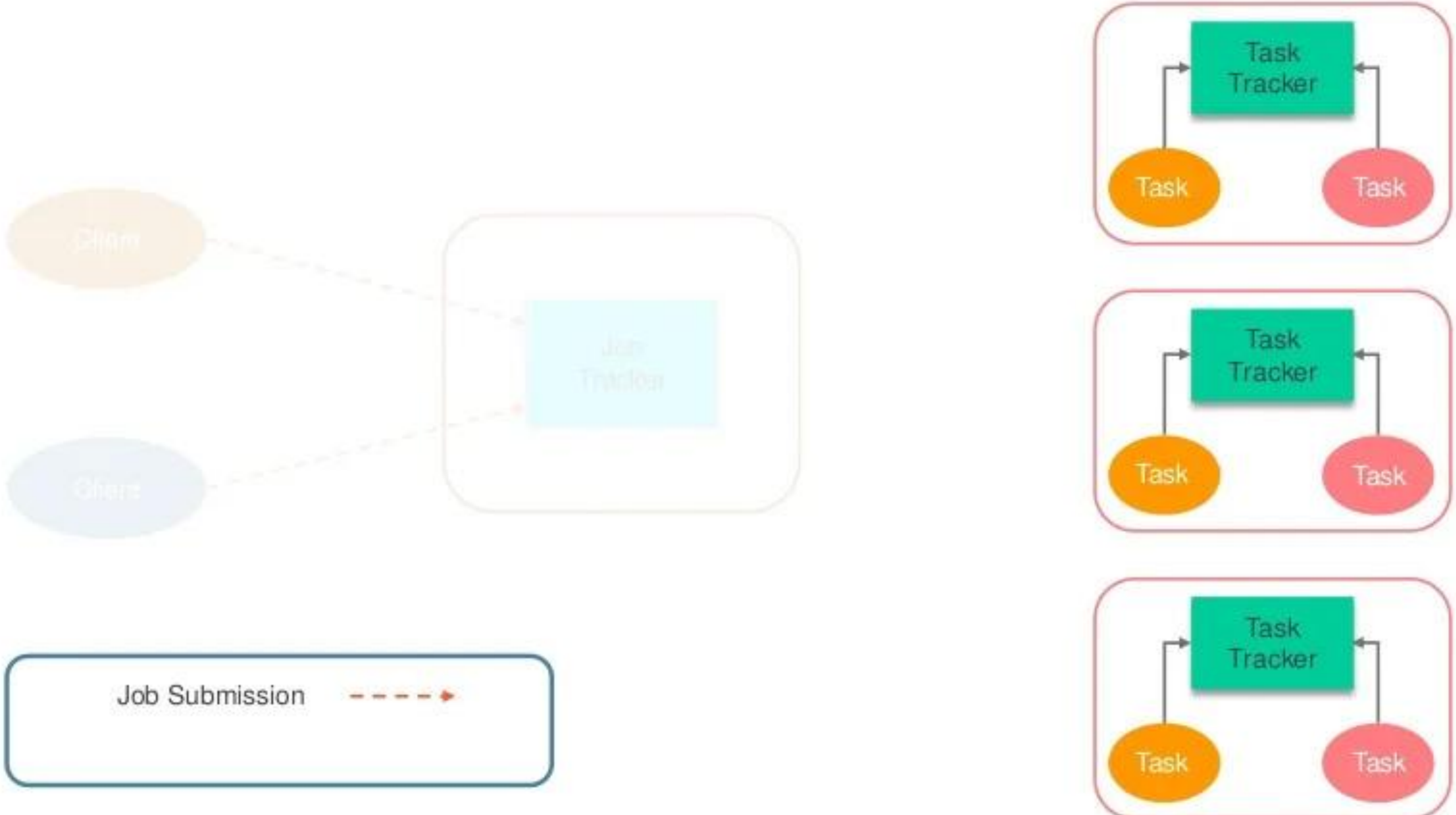
Hadoop 1.0 (MR 1)



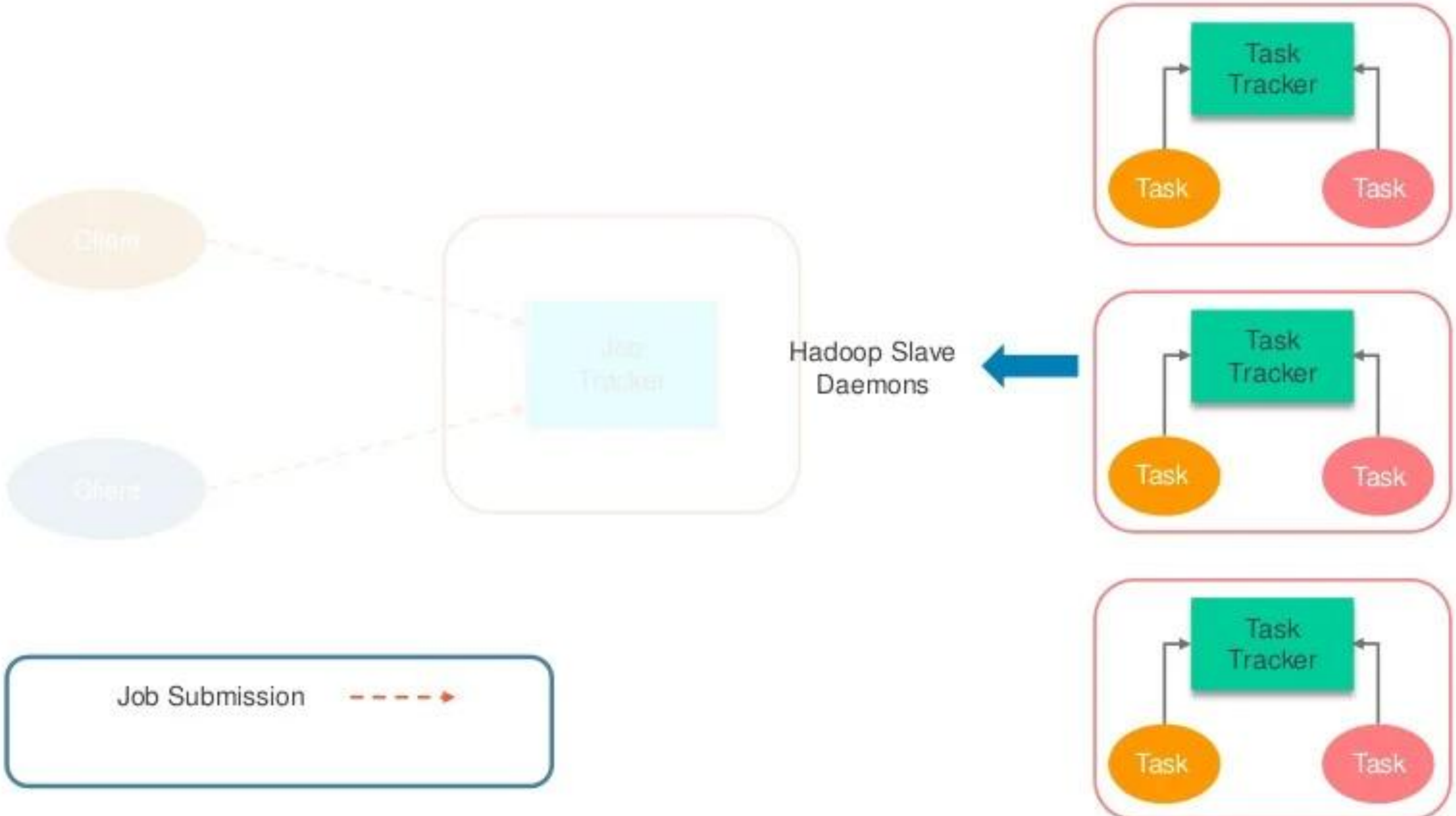
Hadoop 1.0 (MR 1)



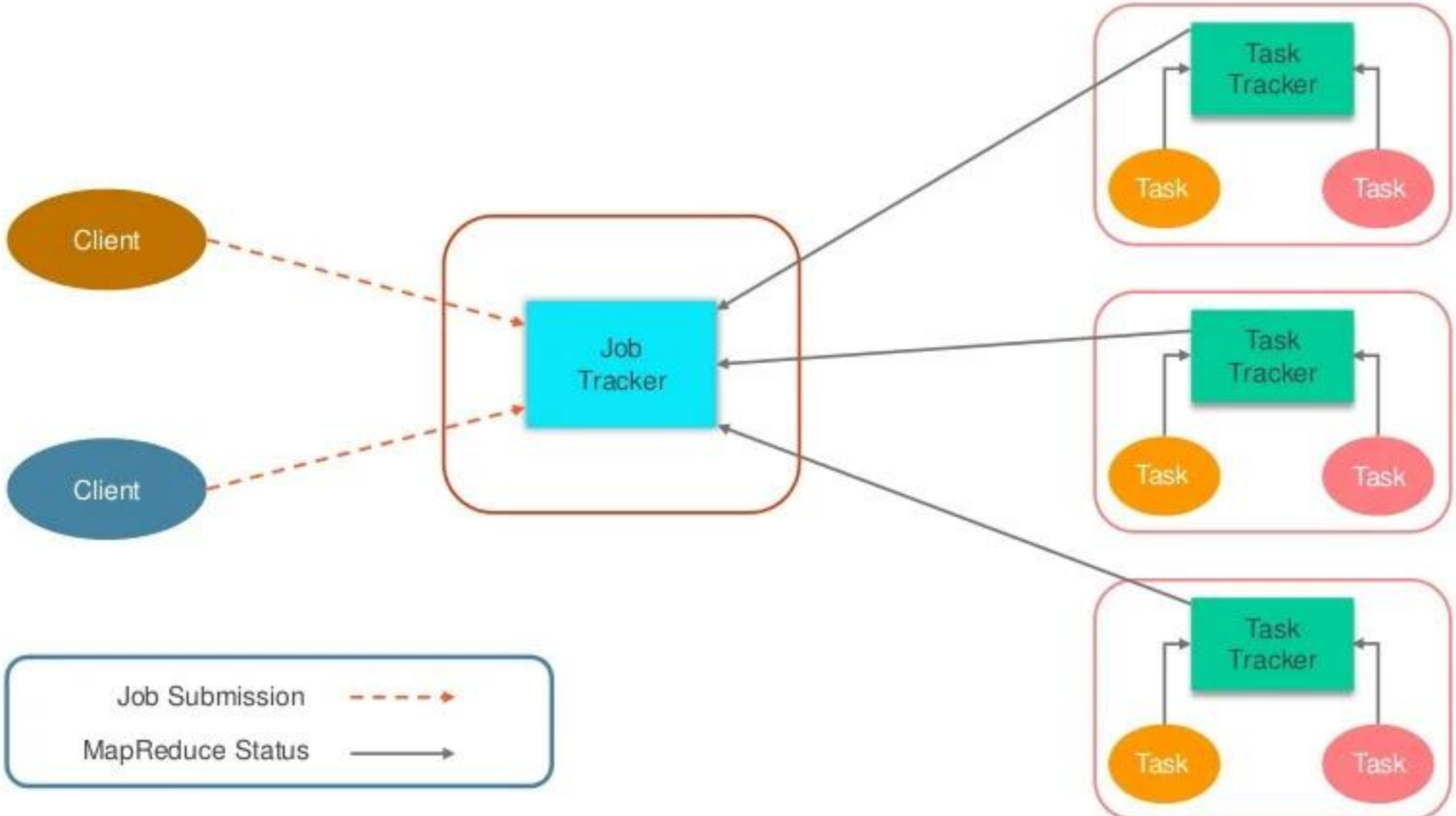
Hadoop 1.0 (MR 1)



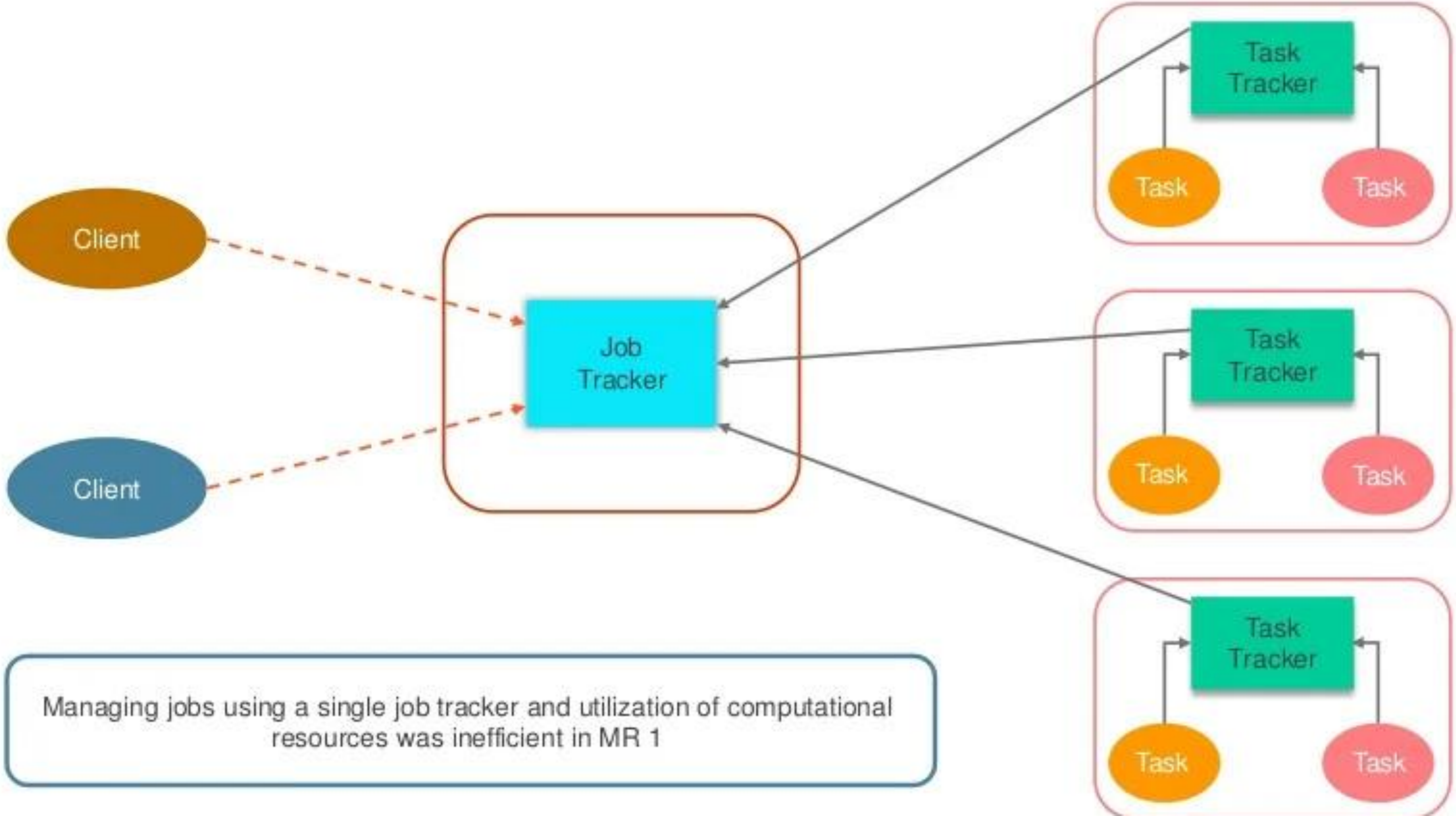
Hadoop 1.0 (MR 1)



Hadoop 1.0 (MR 1)



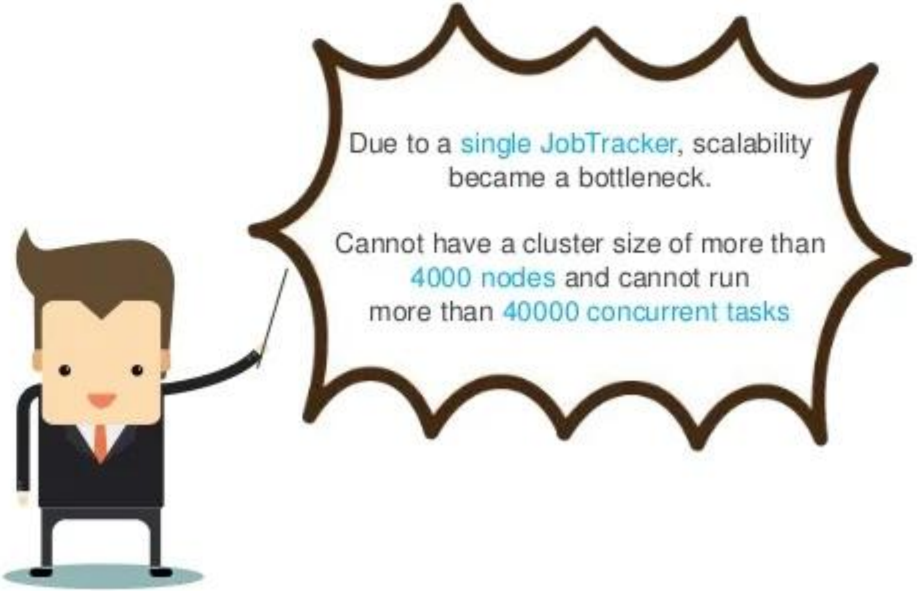
Hadoop 1.0 (MR 1)



Managing jobs using a single job tracker and utilization of computational resources was inefficient in MR 1

Limitations of Hadoop 1.0 (MR 1)

1 Scalability



Due to a single JobTracker, scalability became a bottleneck.

Cannot have a cluster size of more than 4000 nodes and cannot run more than 40000 concurrent tasks

Limitations of Hadoop 1.0 (MR 1)

1 Scalability

Due to a single JobTracker, scalability became a bottleneck.

Maximum cluster size – 4000 nodes
Maximum concurrent tasks – 40000

2 Availability issue

JobTracker is single point of failure. Any failure kills all queued and running jobs. Jobs need to be resubmitted by users

Limitations of Hadoop 1.0 (MR 1)

3 Resource Utilization

Due to predefined number of map and reduce slots for each TaskTracker, resource utilization issues occur

Limitations of Hadoop 1.0 (MR 1)

3 Resource utilization

Due to predefined number of map and reduce slots for each TaskTracker, resource utilization issues occur

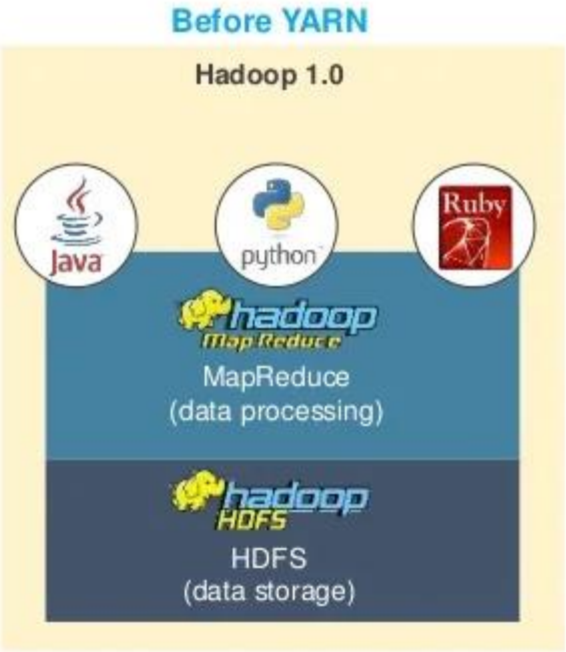
4 Limitations in running non-MapReduce applications

Problem in performing real-time analysis and running Ad-hoc query as MapReduce is batch driven

MapReduce Using Yarn

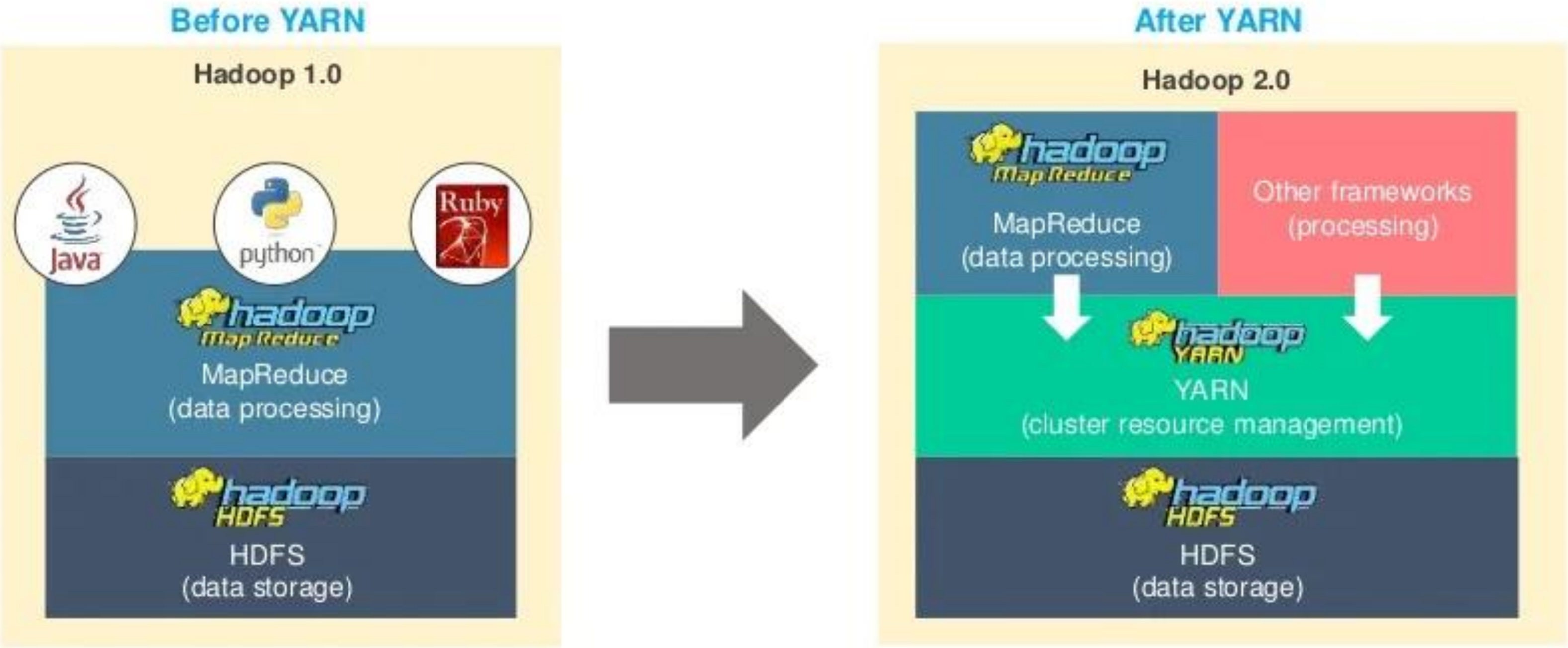
Need for YARN

Need for YARN



Designed to run MapReduce jobs only and had issues in scalability, resource utilization, etc.

Need for YARN



Designed to run MapReduce jobs only and had issues in scalability, resource utilization, etc.

YARN solved those issues and users could work on multiple processing models along with MapReduce

Solution - Hadoop 2.0 (YARN)

Scalability

Can have a cluster size of more than 10,000 nodes and can run more than 1,00,000 concurrent tasks

Solution - Hadoop 2.0 (YARN)

Scalability

Can have a cluster size of more than 10,000 nodes and can run more than 1,00,000 concurrent tasks

Compatibility

Applications developed for Hadoop 1 runs on YARN without any disruption or availability issues

Solution - Hadoop 2.0 (YARN)

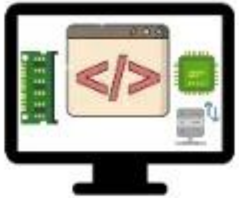
Scalability

Can have a cluster size of more than 10,000 nodes and can run
more than 1,00,000 concurrent tasks

Compatibility

Applications developed for Hadoop 1 runs on YARN without any disruption or availability issues

Resource utilization



Allows dynamic allocation of cluster resources to improve resource utilization

Solution - Hadoop 2.0 (YARN)

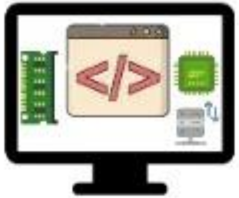
Scalability

Can have a cluster size of more than 10,000 nodes and can run more than 1,00,000 concurrent tasks

Compatibility

Applications developed for Hadoop 1 runs on YARN without any disruption or availability issues

Resource utilization

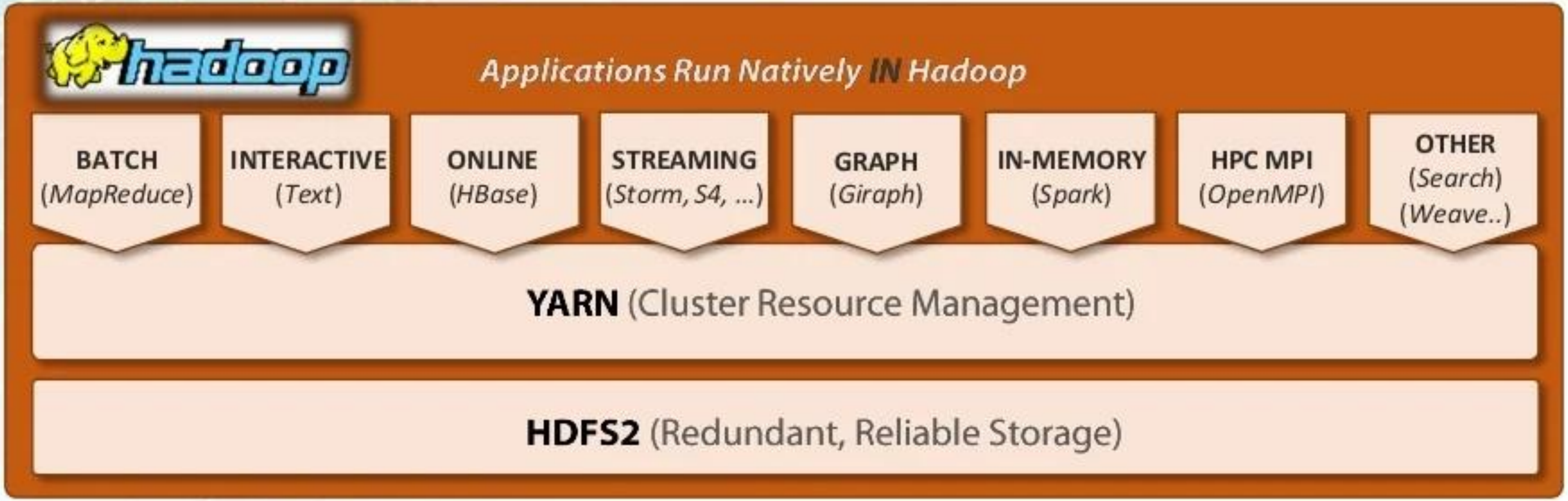


Allows dynamic allocation of cluster resources to improve resource utilization

Multitenancy

Can use open-source and propriety data access engines and perform real-time analysis and running ad-hoc query

YARN – Moving beyond MapReduce



Applications Run Natively ***IN*** *Hadoop*

BATCH (*MapReduce*)

INTERACTIVE (*Text*)

ONLINE (*HBase*)

STREAMING (*Storm, S4, ...*)

GRAPH (*Giraph*)

IN-MEMORY (*Spark*)

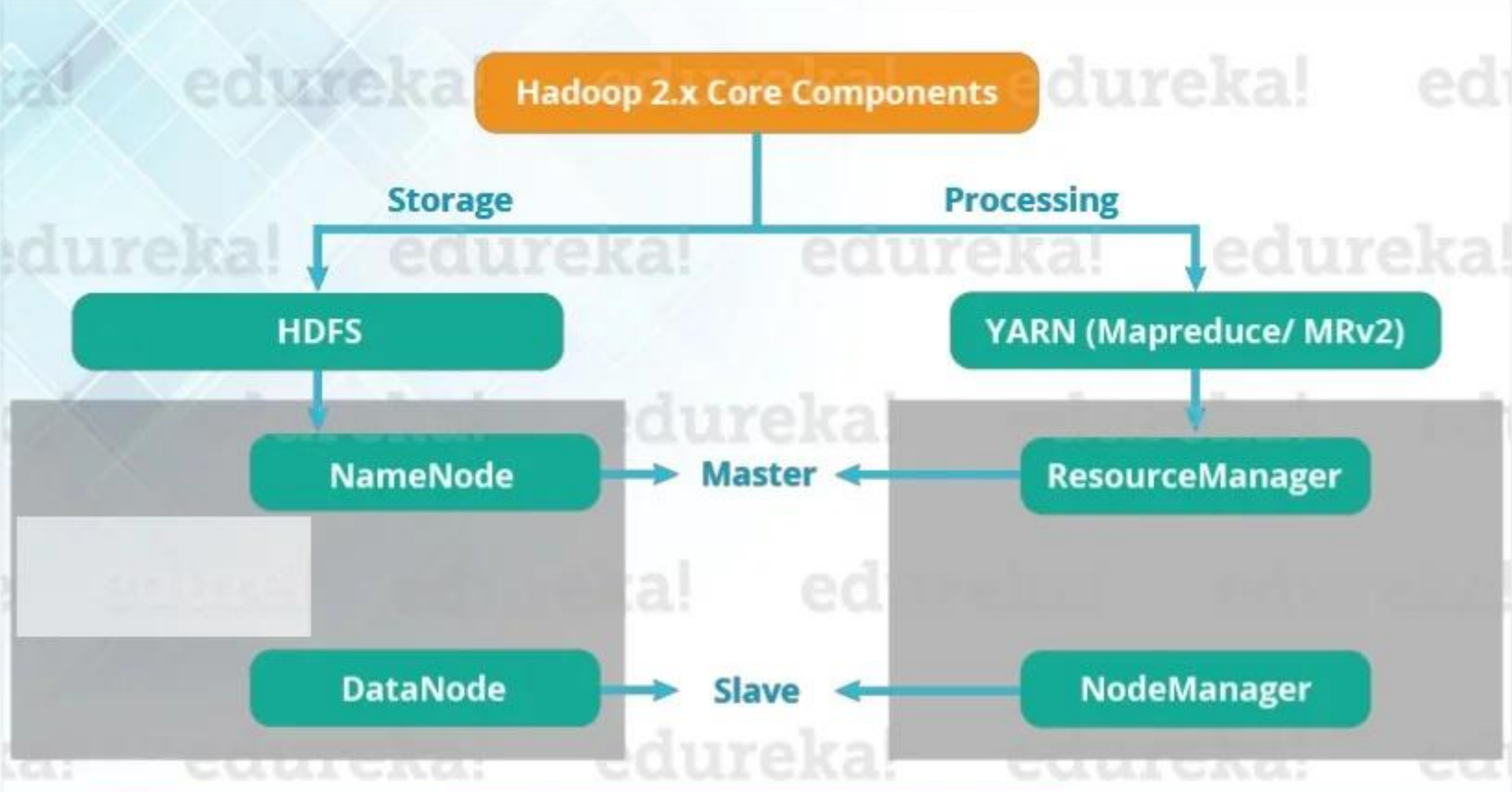
HPC MPI (*OpenMPI*)

OTHER (*Search*) (*Weave..*)

YARN (Cluster Resource Management)

HDFS2 (Redundant, Reliable Storage)

Hadoop 2.x Daemons



Hadoop 2.x Core Components

Storage

Processing

HDFS

YARN (Mapreduce/ MRv2)

NameNode

Master

ResourceManager

DataNode

Slave

NodeManager

Hadoop 2.x MapReduce Yarn Components

→ 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)

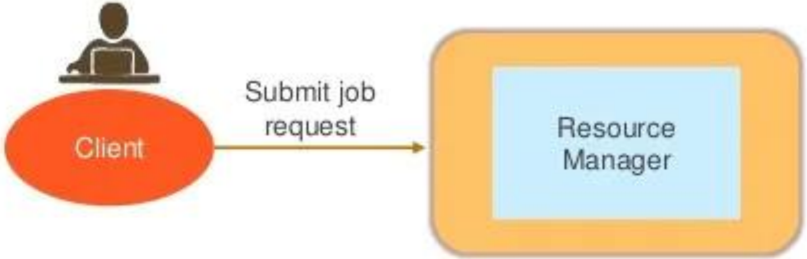
→ Container

- » Created by NM when requested
- » Allocates certain amount of resources (memory, CPU etc.) on a slave node

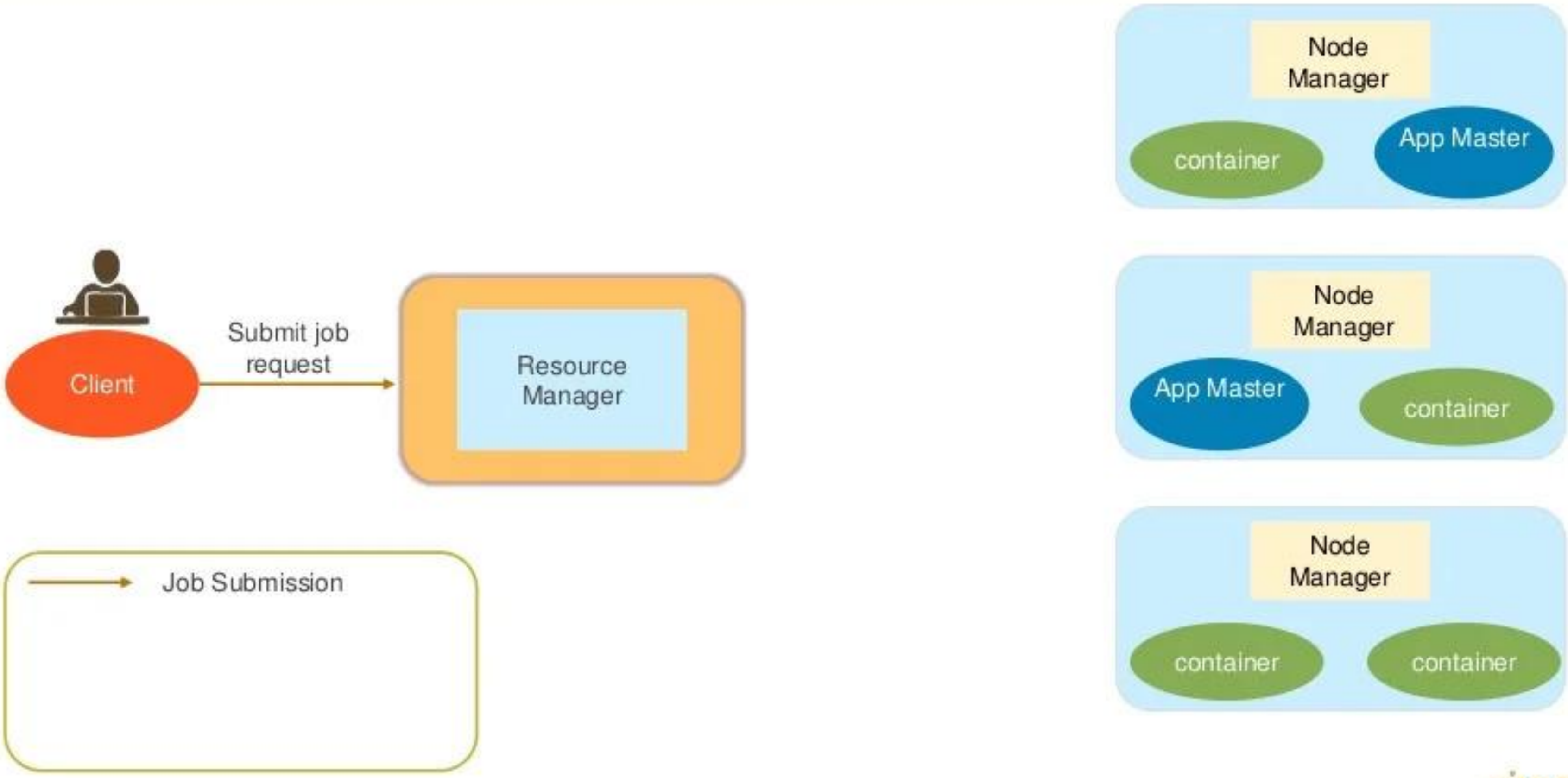
YARN Architecture

YARN Architecture

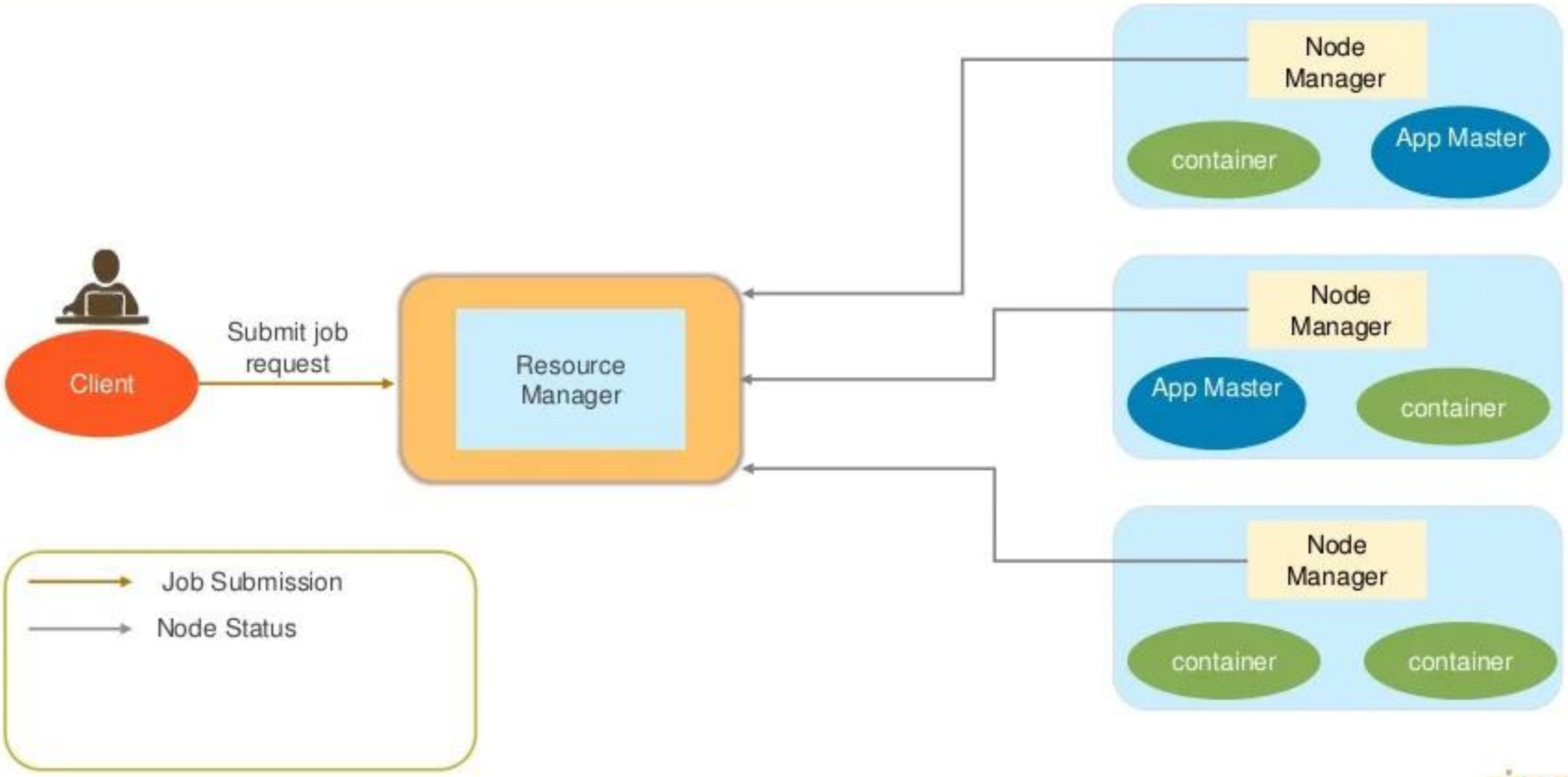
YARN Architecture



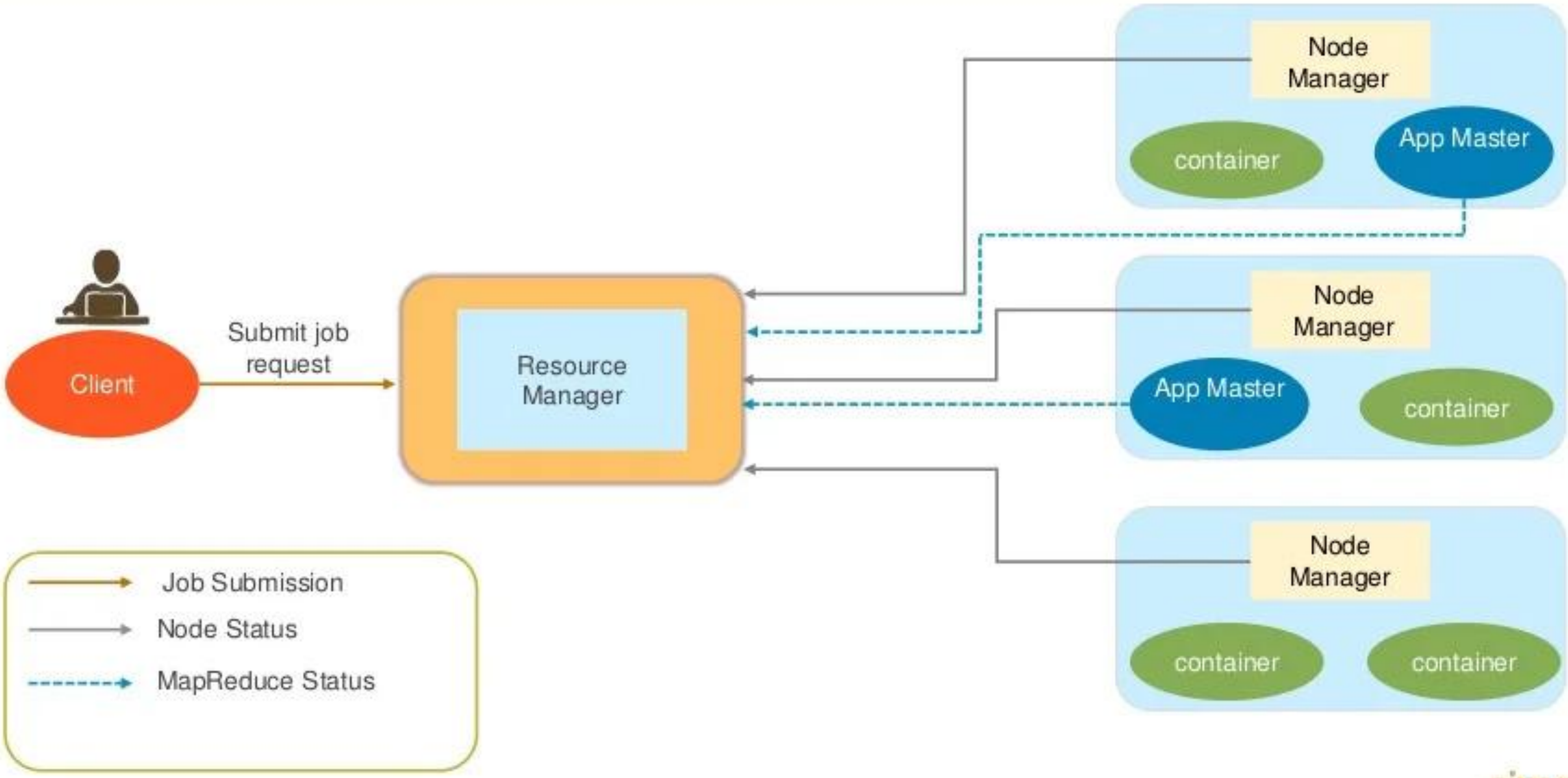
YARN Architecture



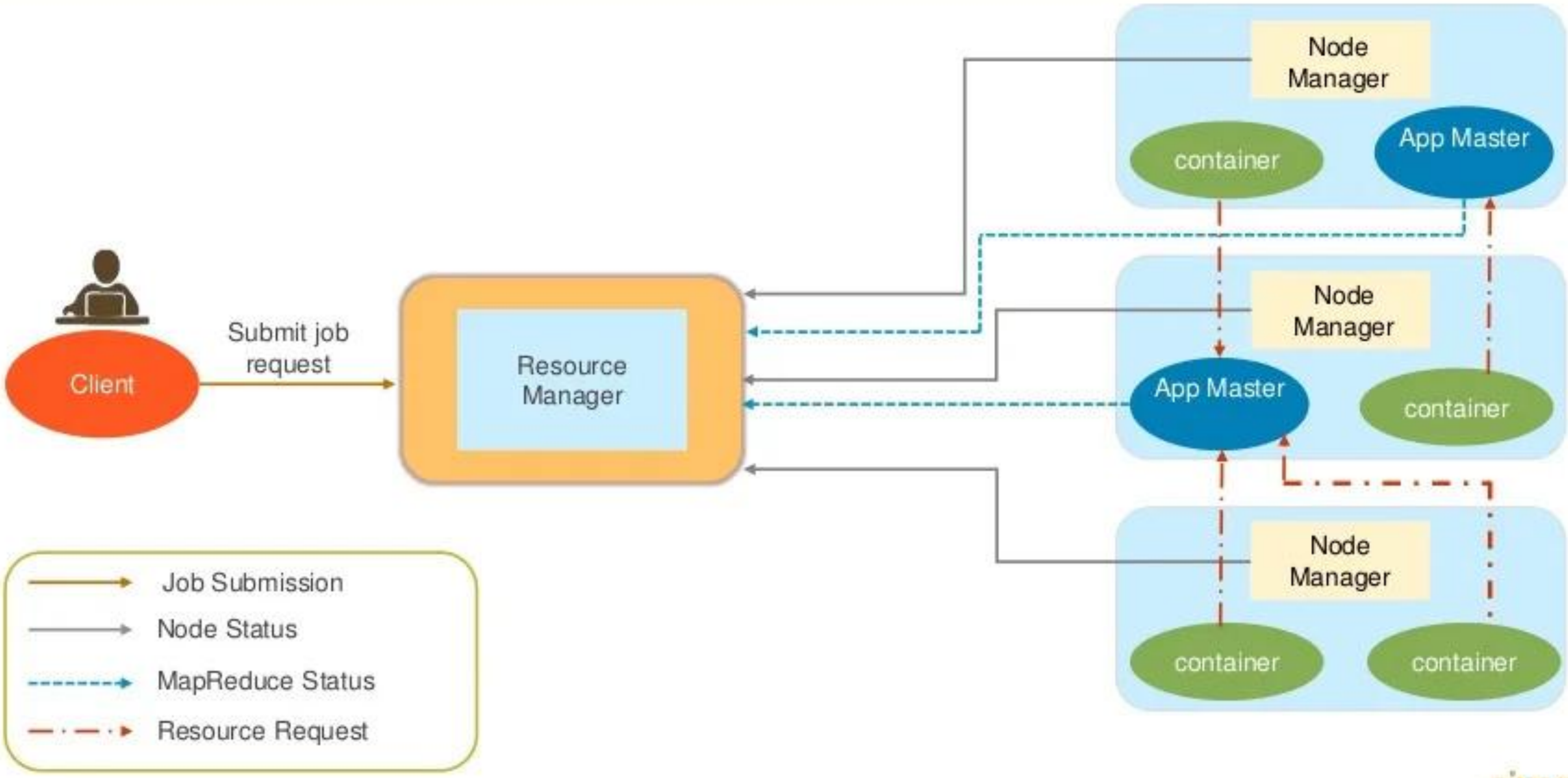
YARN Architecture



YARN Architecture

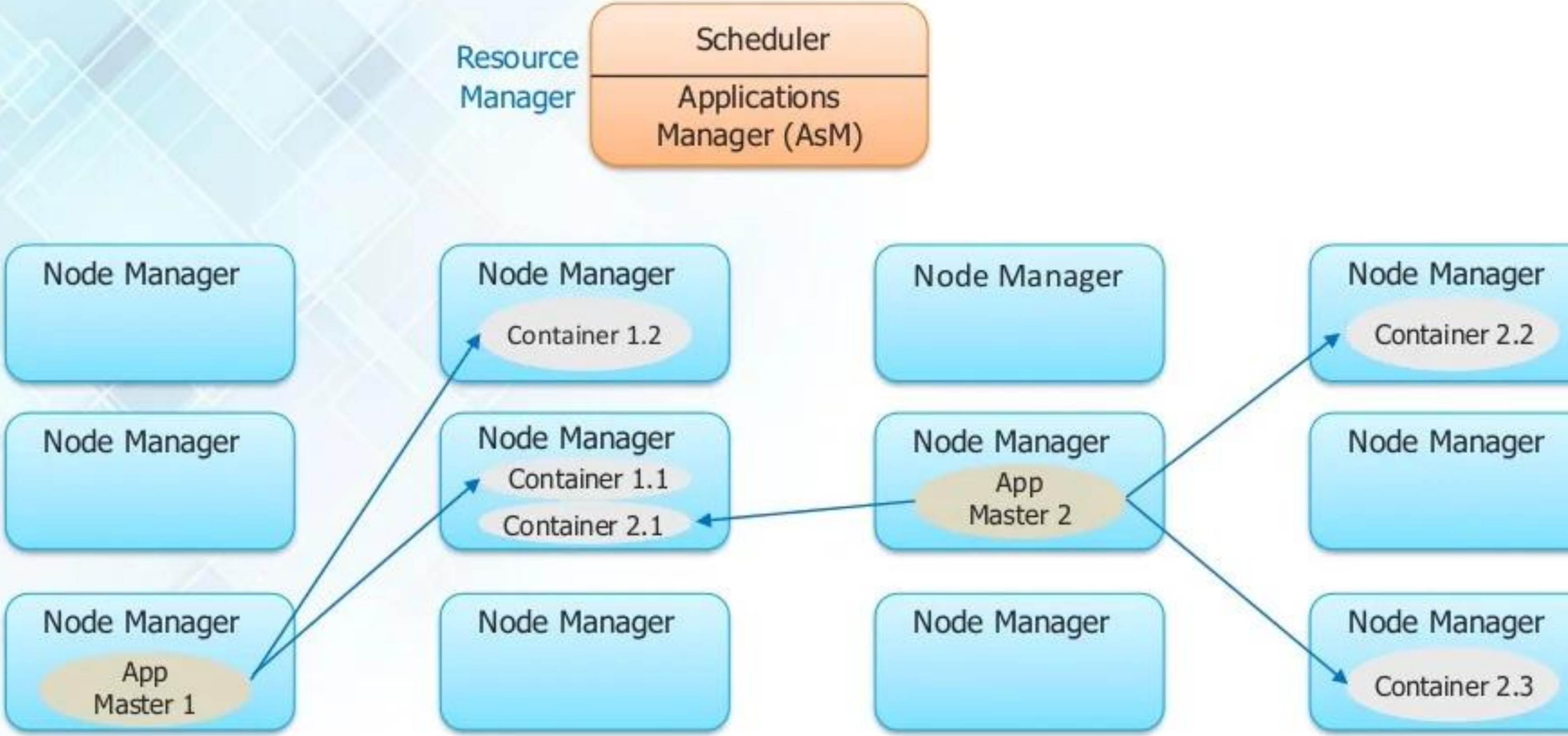


YARN Architecture



YARN Application Workflow in MapReduce

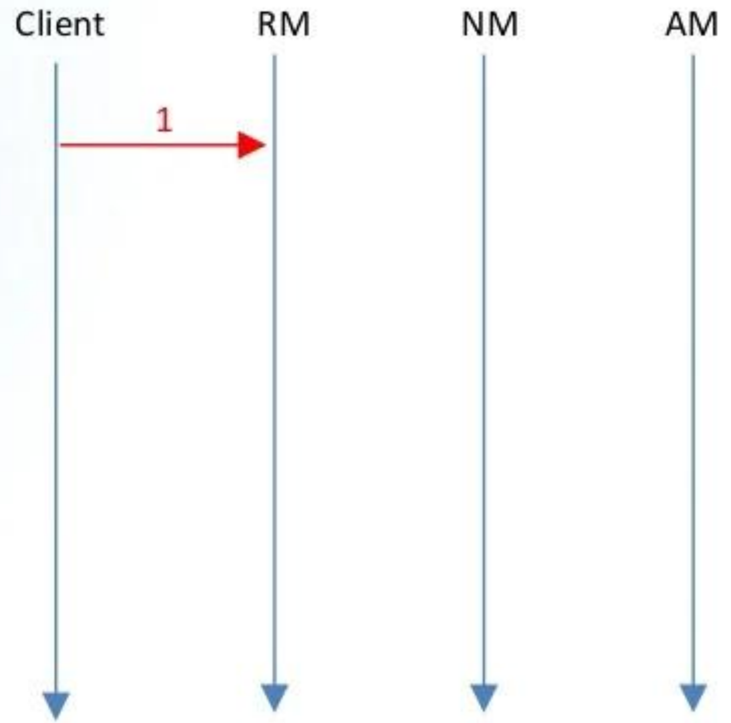
YARN Workflow



Application Workflow

→ Execution Sequence :

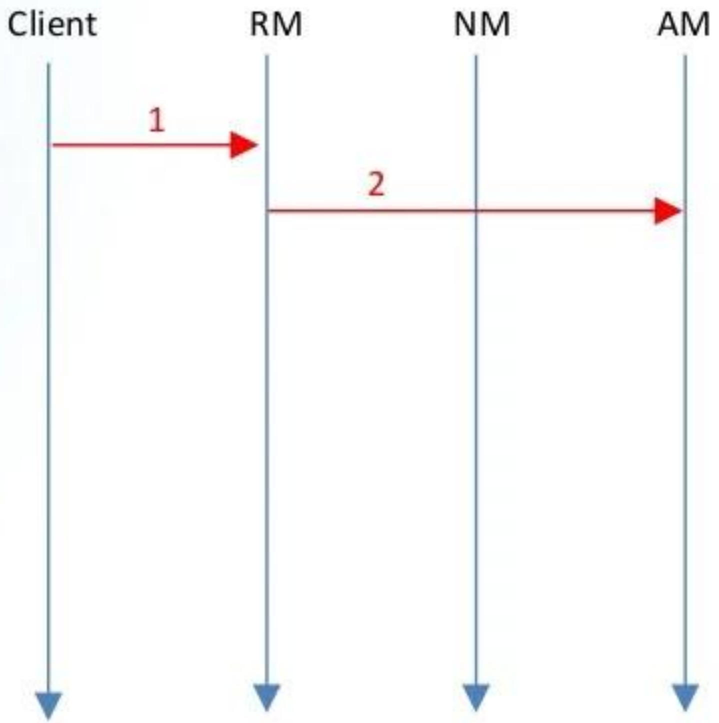
1. Client submits an application



Application Workflow

→ Execution Sequence :

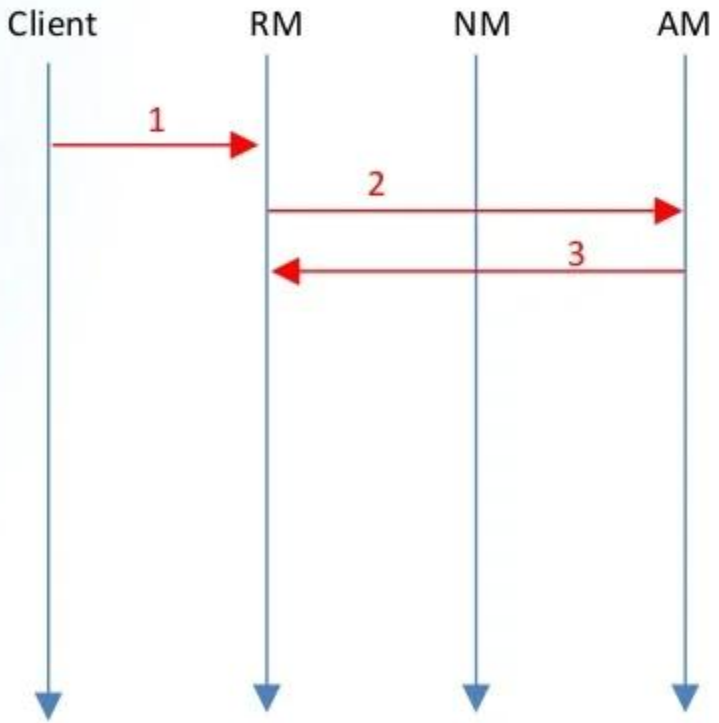
1. Client submits an application
2. RM allocates a container to start AM



Application Workflow

→ Execution Sequence :

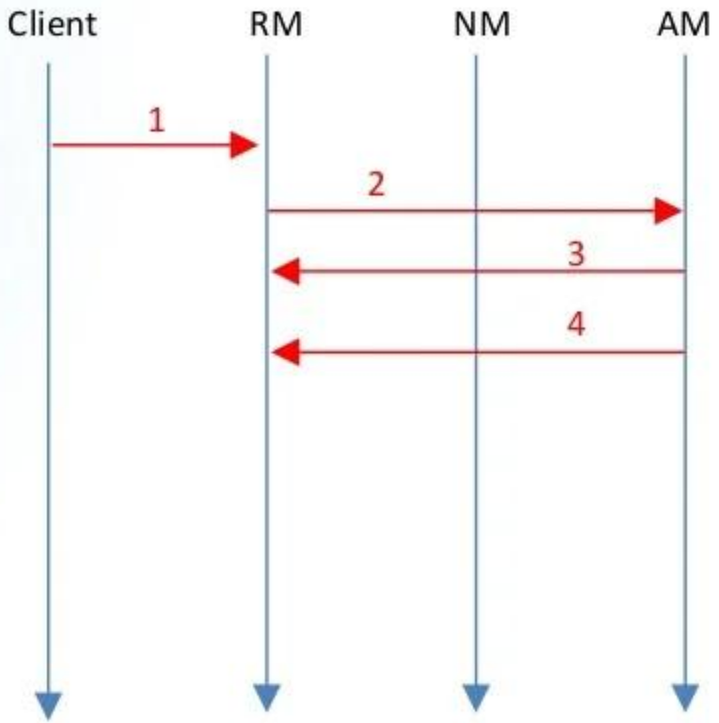
1. Client submits an application
2. RM allocates a container to start AM
3. AM registers with RM



Application Workflow

→ Execution Sequence :

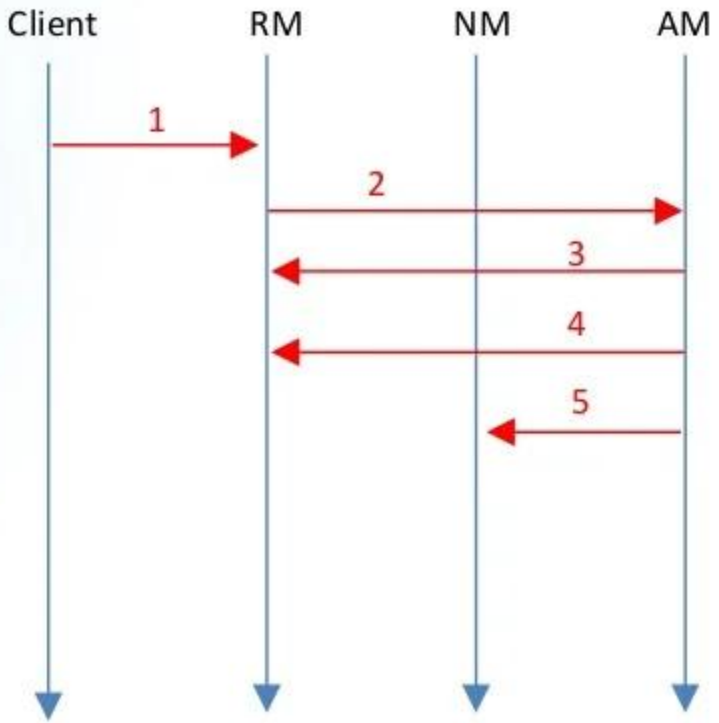
1. Client submits an application
2. RM allocates a container to start AM
3. AM registers with RM
4. AM asks containers from RM



Application Workflow

→ Execution Sequence :

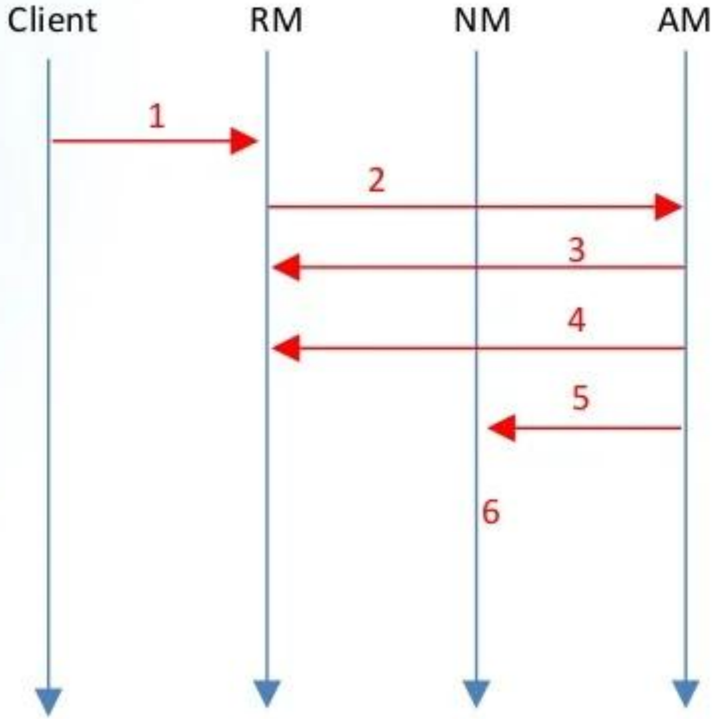
1. Client submits an application
2. RM allocates a container to start AM
3. AM registers with RM
4. AM asks containers from RM
5. AM notifies NM to launch containers



Application Workflow

→ Execution Sequence :

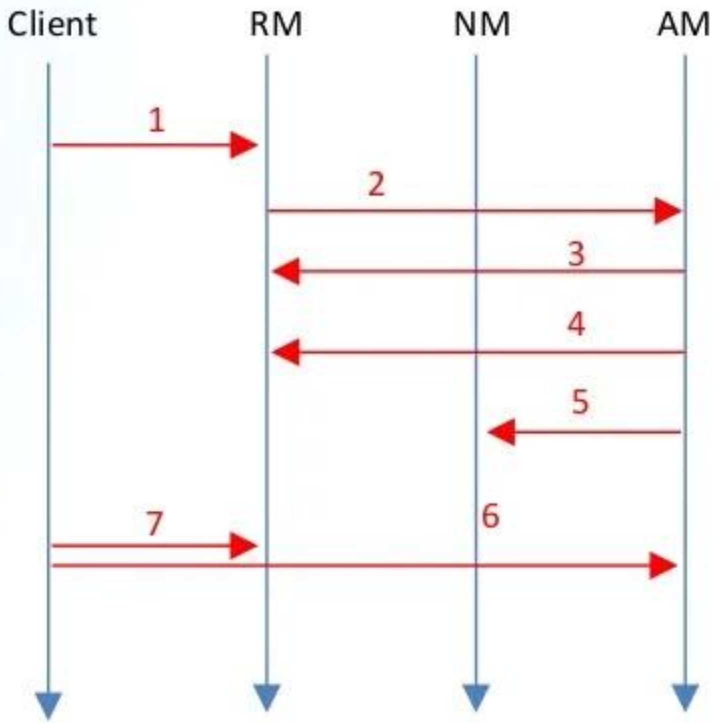
1. Client submits an application
2. RM allocates a container to start AM
3. AM registers with RM
4. AM asks containers from RM
5. AM notifies NM to launch containers
6. Application code is executed in container



Application Workflow

→ Execution Sequence :

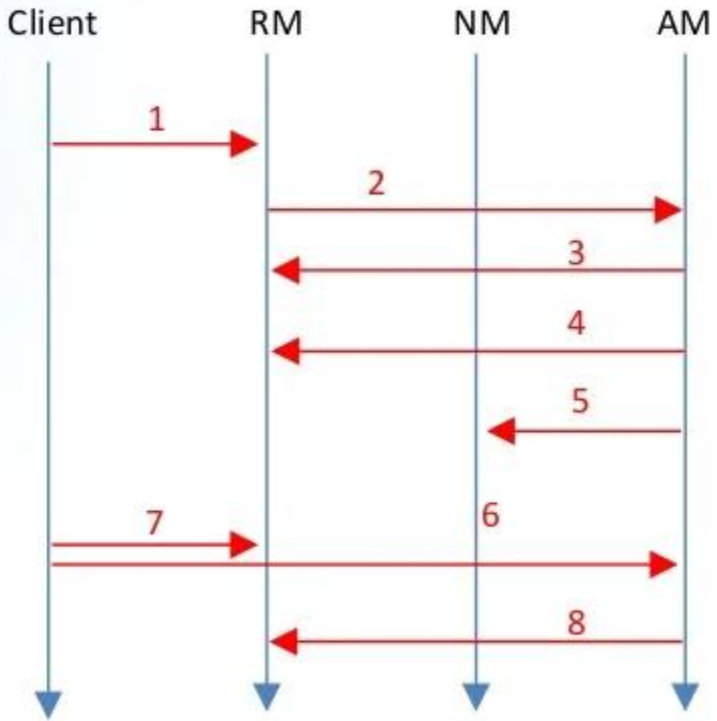
1. Client submits an application
2. RM allocates a container to start AM
3. AM registers with RM
4. AM asks containers from RM
5. AM notifies NM to launch containers
6. Application code is executed in container
7. Client contacts RM/AM to monitor application's status



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



Thank You

**Slide content from
Edureka and Simplilearn**