Distributed and Parallel Computing

Trong-Hop Do

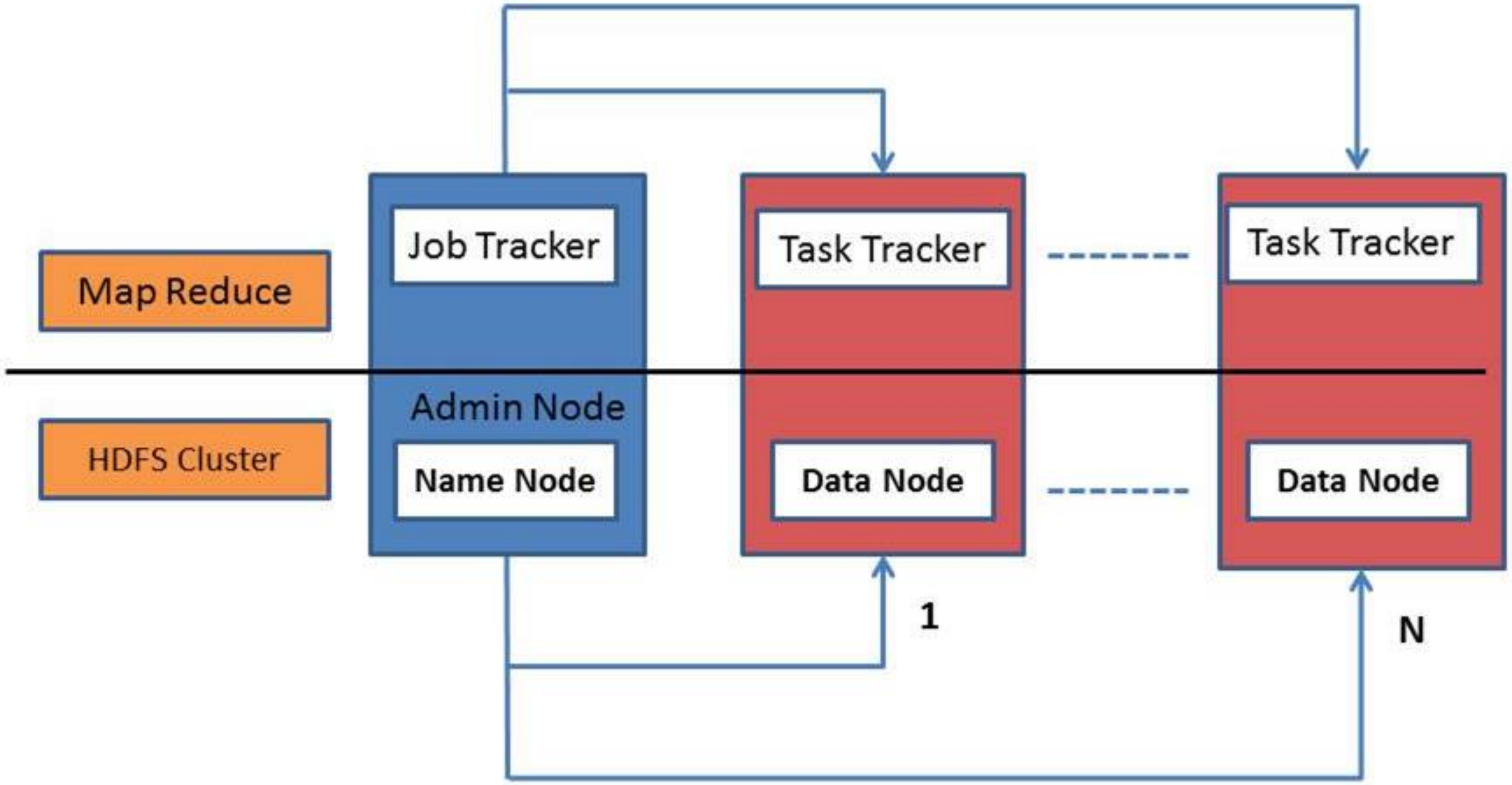
Distributed and Parallel computing with Hadoop and Spark

Introduction

- Hadoop is a software framework written in **Java** for distributed processing of large datasets (terabytes or petabytes of data) across large clusters (thousands of nodes) of computers. Included some key components as below:
  - **Hadoop Common**: common utilities
  - **Hadoop Distributed File System** (**HDFS**) (Storage Component): A distributed file system that provides high-throughput access
  - **Hadoop YARN** (Scheduling): a framework for job scheduling & cluster resource management (available from Hadoop 2.x)
  - **Hadoop MapReduce** (Processing): A yarn-based system for parallel processing of large data sets

Hadoop 1.x architecture

Core components



HDFS and MapReduce are known as “Two Pillars” of Hadoop 1.x

HDFS

- **Hadoop Distributed File System** (**HDFS**) is designed to reliably store very large files across machines in a large cluster. It is inspired by the Google File System.
- Designed to reliably store data on **commodity hardware** (crash all the time)
- Intended for Large files and Batch inserts
- Distribute large data file into **blocks**
- Each block is replicated on multiple (slave) nodes
- HDFS component is divided into two sub-components: Name node and Data node

HDFS

- **NameNode**:
  - Master of the system, daemon runs on the **master machine**
  - Maintains, monitoring and manages the **blocks** which are present on the **DataNodes**
  - records the metadata of the files like the location of blocks, file size, permission, hierarchy etc.
  - captures all the changes to the metadata like deletion, creation and renaming of the file in edit logs.
  - It regularly receives **heartbeat** and block reports from the DataNodes.

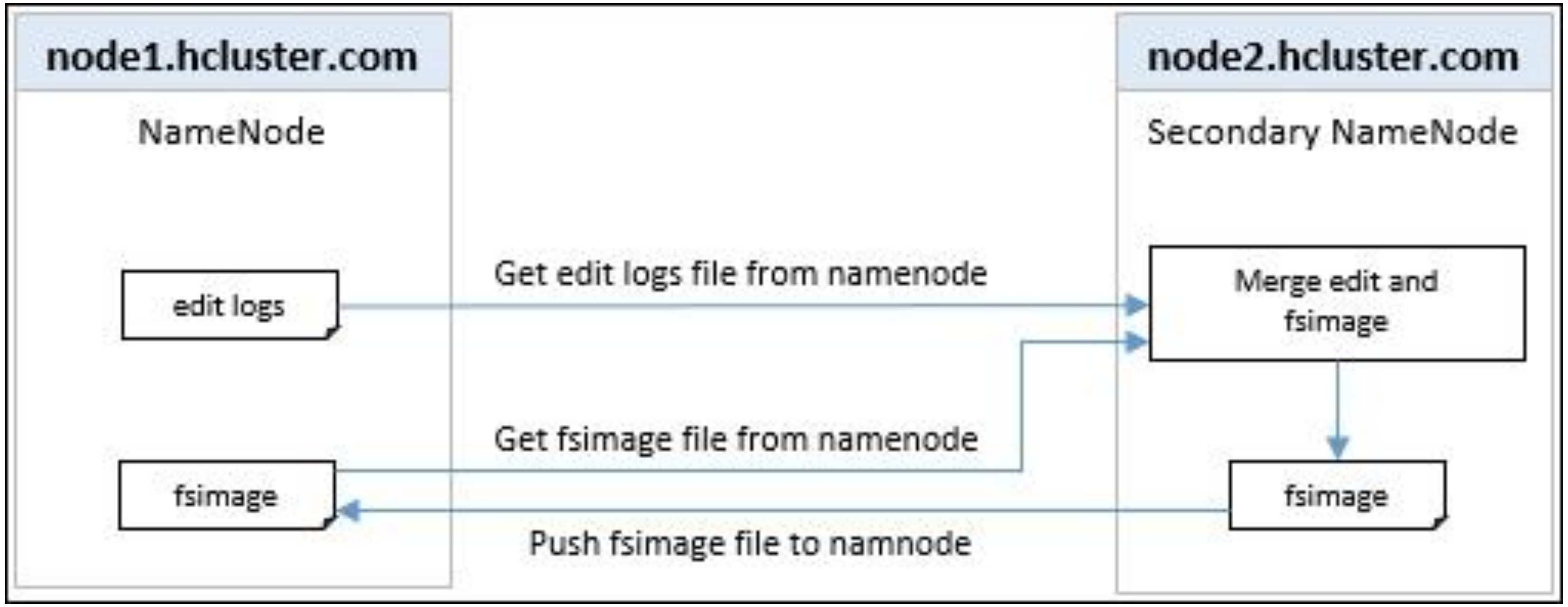
HDFS

Secondary namenode

- The secondary namenode daemon is responsible for performing periodic housekeeping functions for namenode.
- It creates checkpoints of the filesystem metadata (fsimage) present in namenode by merging the edits logfile and the fsimage file from the namenode daemon.
- In case the namenode daemon fails, this checkpoint could be used to rebuild the filesystem metadata.
- Checkpoints are done in intervals, thus checkpoint **data could be slightly outdated**. Rebuilding the fsimage file using such a checkpoint **could lead to data loss.**
- It is recommended that the secondary namenode daemon be hosted on a separate machine for large clusters.
- The checkpoints are created by merging the edits logfiles and the fsimage file from the namenode daemon.

HDFS

Secondary namenode

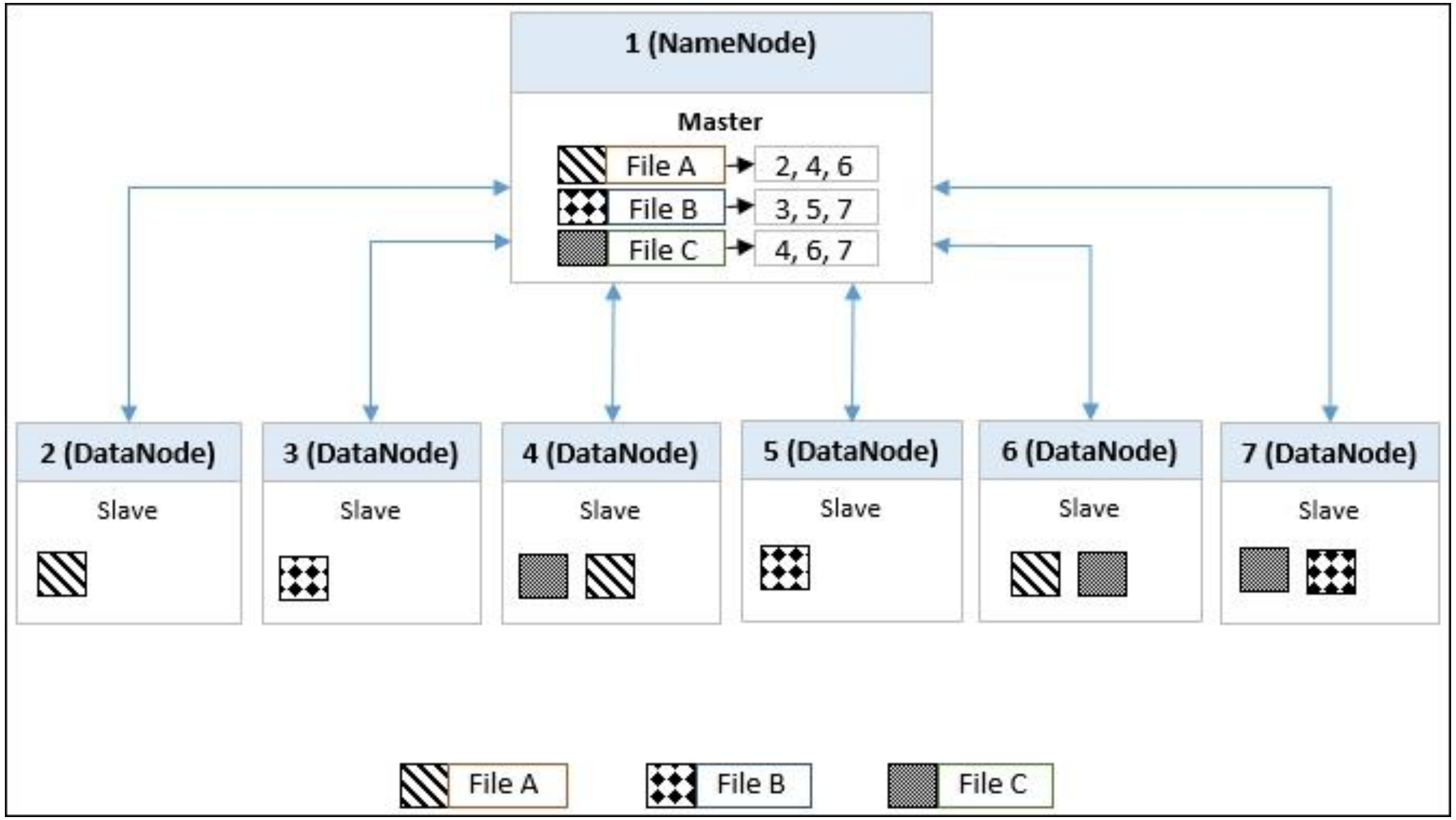


HDFS

- **DataNode**:
  - DataNode runs on the **slave machine**.
  - It stores the actual business data.
  - It serves the read-write request from the user.
  - DataNode does the ground work of creating, replicating and deleting the blocks on the command of NameNode.
  - After every **3 seconds**, by default, it sends **heartbeat** to NameNode reporting the health of HDFS.

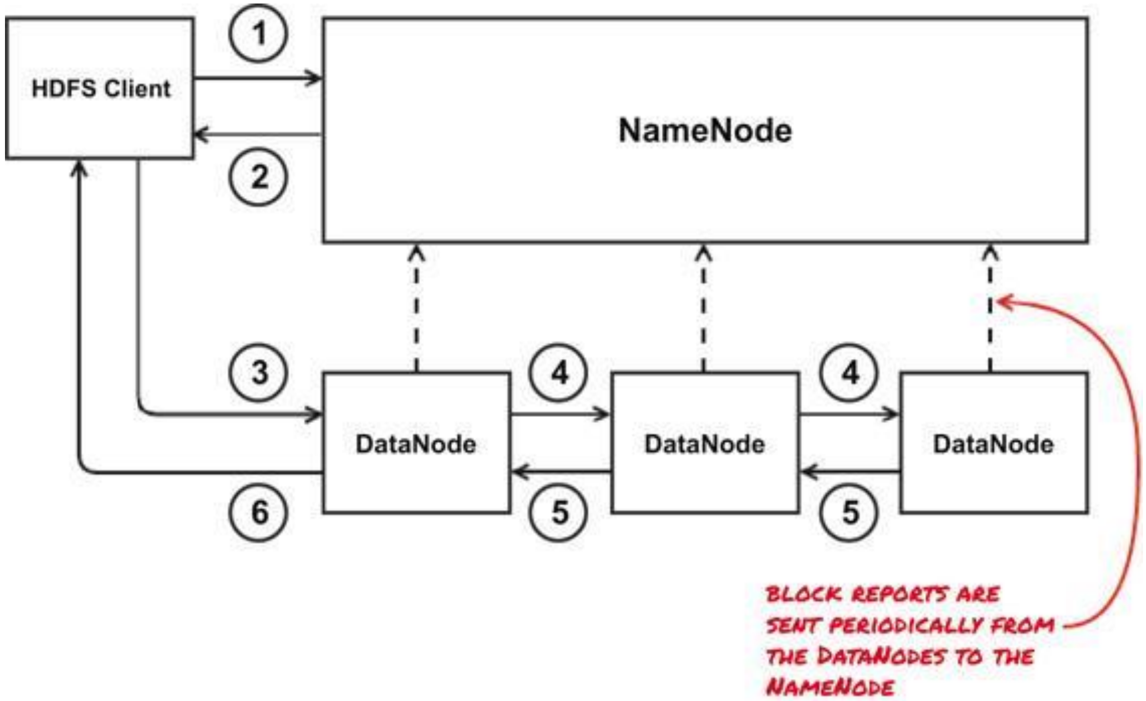
HDFS

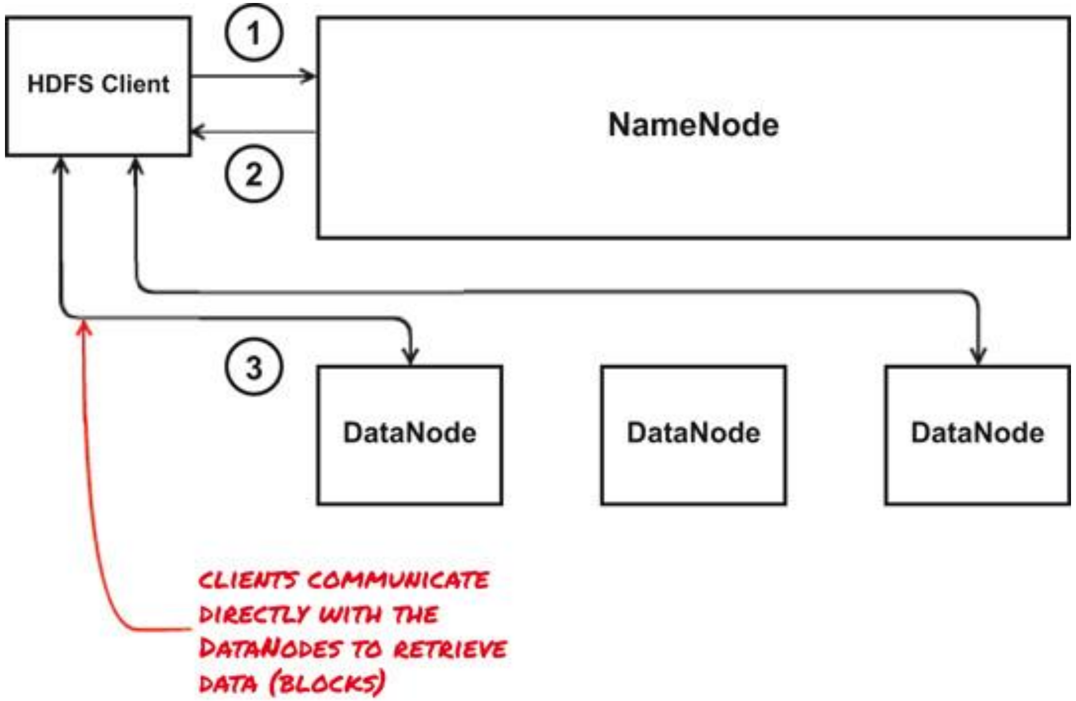
Data node



HDFS

Write & Read files





Map Reduce

- **Programming model** for **distributed computations** at a massive scale
- Execution framework for organizing and performing such computations
- **Data locality** is king
- MapReduce component is again divided into two sub-components: JobTracker and TaskTracker
- JobTracker: takes care of all the job scheduling and assign tasks to Task Trackers.
- TaskTracker: a node in the cluster that accepts tasks - **Map**, **Reduce** & **Shuffle** operations - from jobtracker

Map Reduce

JobTracker

- The **jobtracker** daemon is responsible for accepting job requests from a client and scheduling/assigning tasktrackers with tasks to be performed.
- The jobtracker daemon tries to assign tasks to the tasktracker daemon on the datanode daemon where the data to be processed is stored. This feature is called **data locality**.
- If that is not possible, it will at least try to assign tasks to tasktrackers within the same physical server rack.
- If for some reason the node hosting the datanode and tasktracker daemons fails, the jobtracker daemon assigns the task to another tasktracker daemon where the replica of the data exists. This is possible because of the replication factor configuration for HDFS where the data blocks are replicated across multiple datanodes. This ensures that the job does not fail even if a node fails within the cluster.

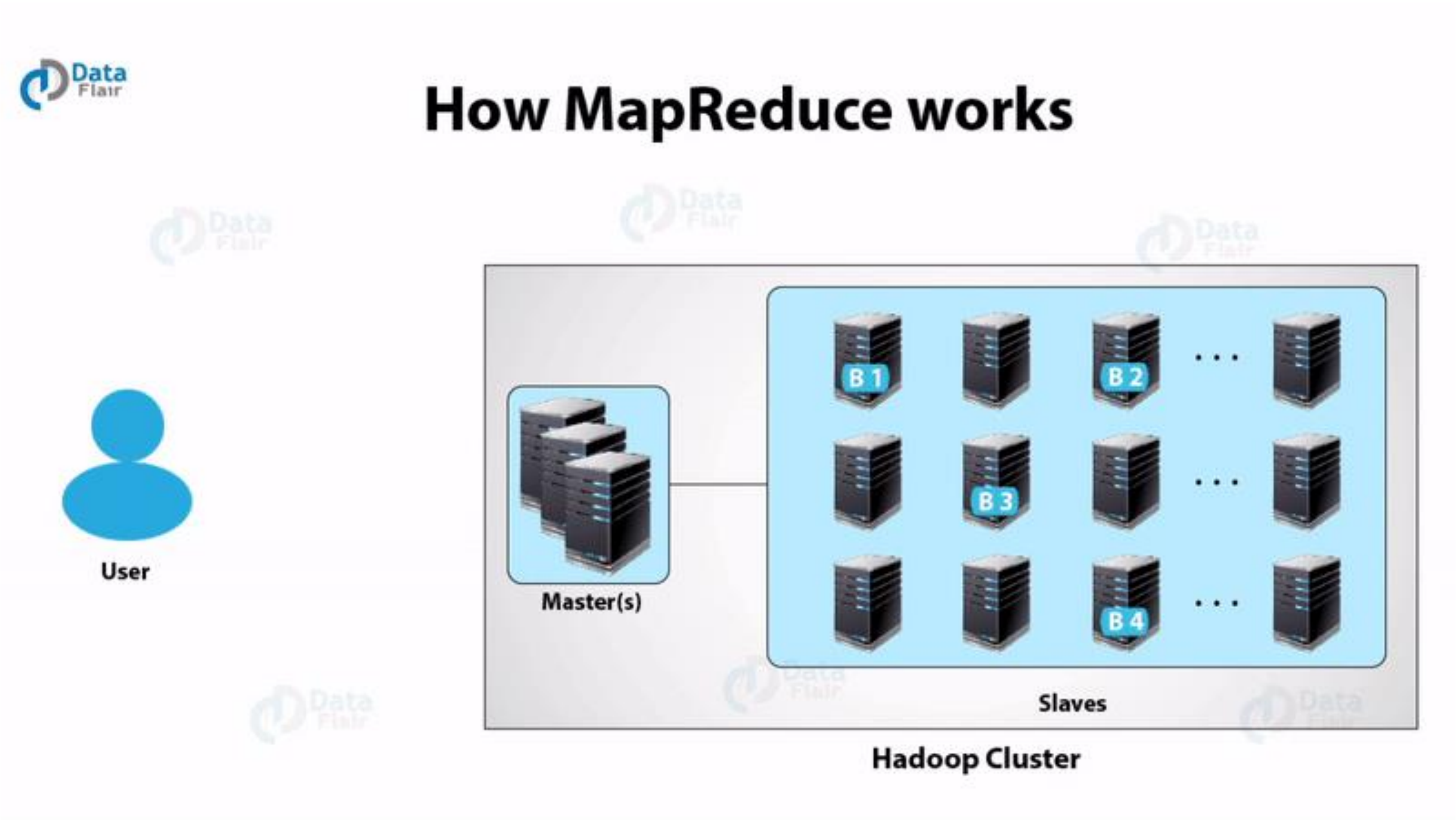
Map Reduce

TaskTracker

- The tasktracker daemon is a daemon that accepts tasks (map, reduce, and shuffle) from the jobtracker daemon.
- The tasktracker daemon is the daemon that performs the actual tasks during a MapReduce operation.
- The tasktracker daemon sends a heartbeat message to jobtracker, periodically, to notify the jobtracker daemon that it is alive.
- Along with the heartbeat, it also sends the free slots available within it, to process tasks.
- The tasktracker daemon starts and monitors the map, and reduces tasks and sends progress/status information back to the jobtracker daemon.

Map Reduce

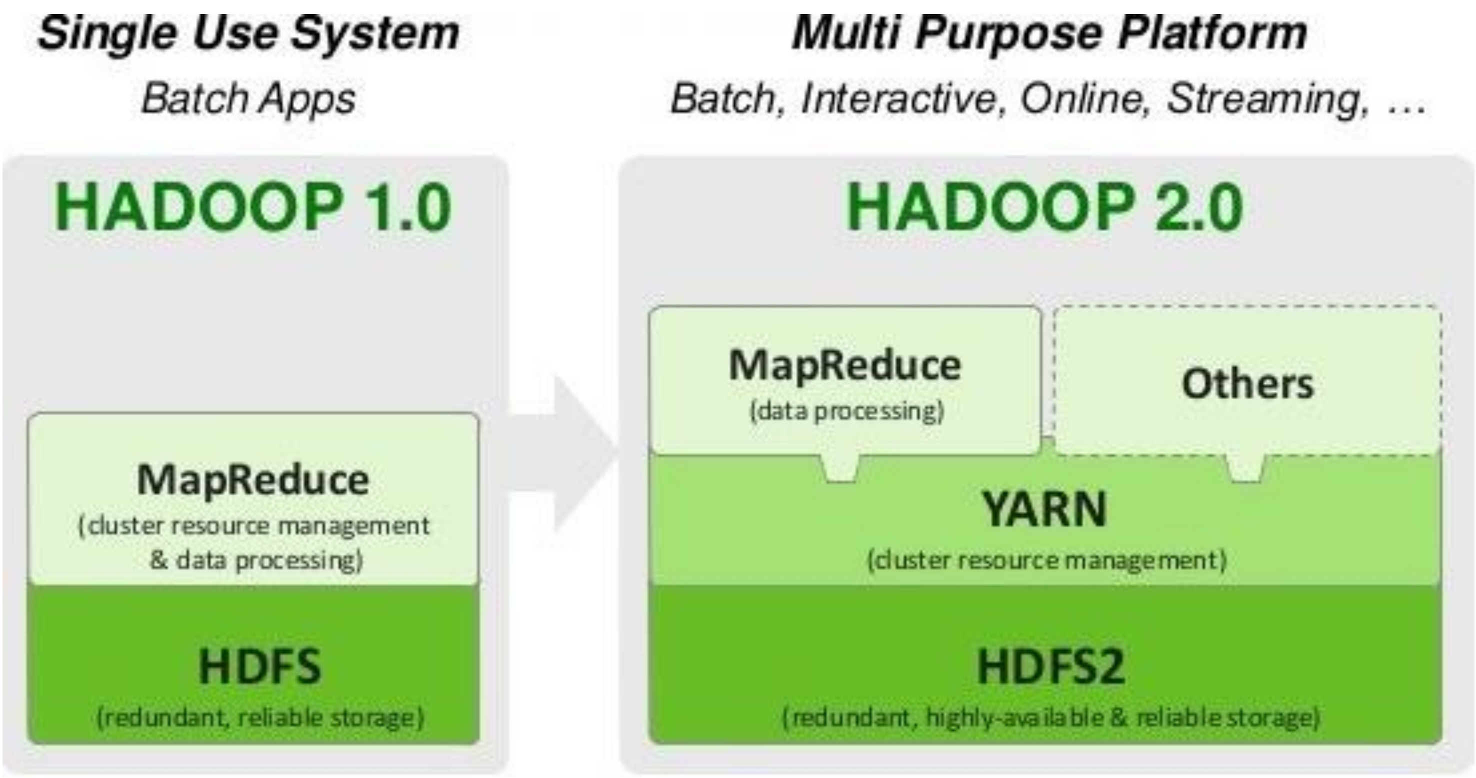
flow



Map Reduce

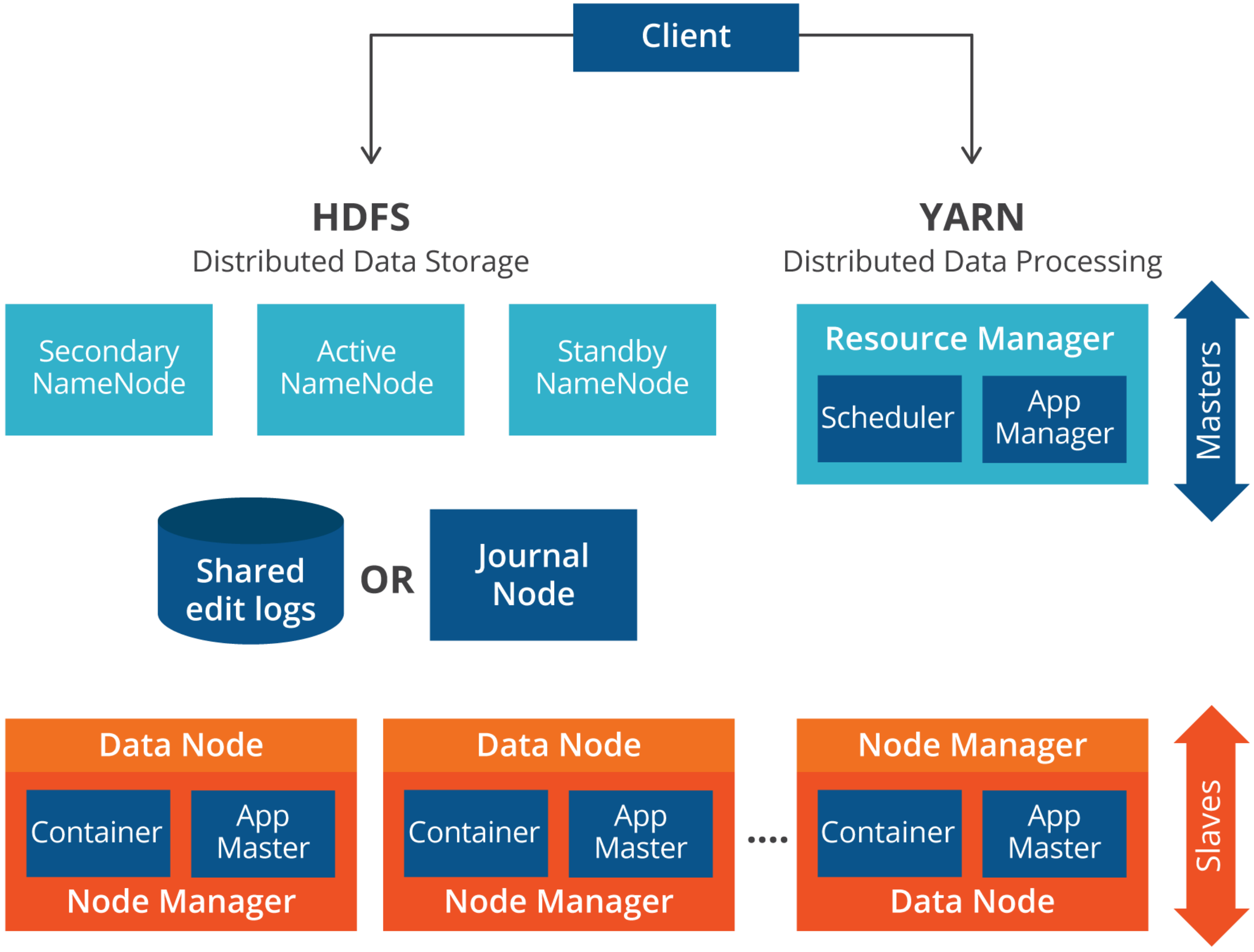
- The **Mapper**:
    - Each block is processed in isolation by a map task called mapper
    - Map task runs on the node where the block is stored
    - Iterate over a large number of records
    - Extract something of interest from each
- **Shuffle** and **sort** intermediate results
- The **Reducer**:
    - Consolidate result from different mappers
    - Aggregate intermediate results
    - Produce final output

Hadoop 2.x



HDFS2, YARN, MapReduce: Three Pillars of Hadoop 2

Apache Hadoop 2.0 and YARN



Hadoop 2.x

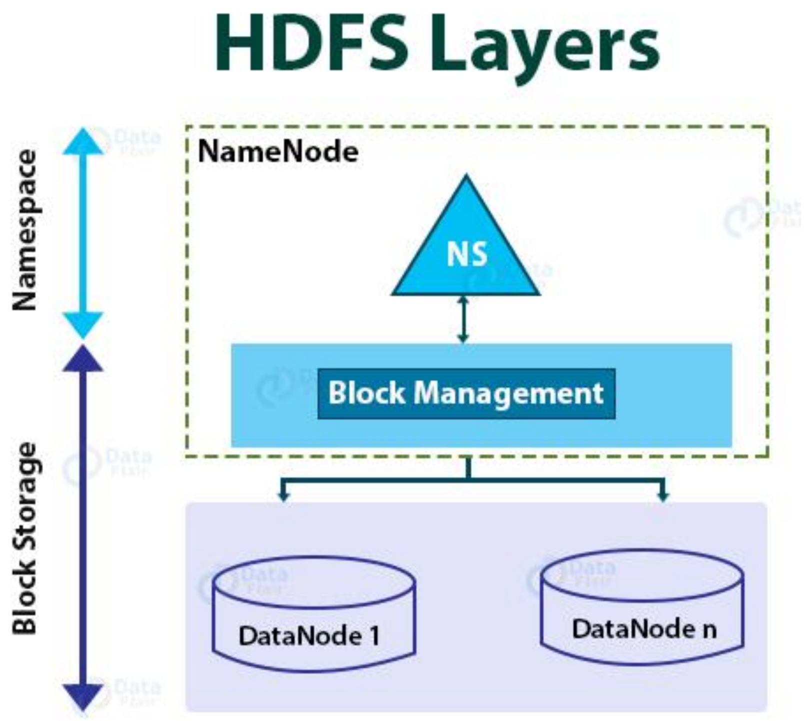
Following are the four main improvements in Hadoop 2.0 over Hadoop 1.x:

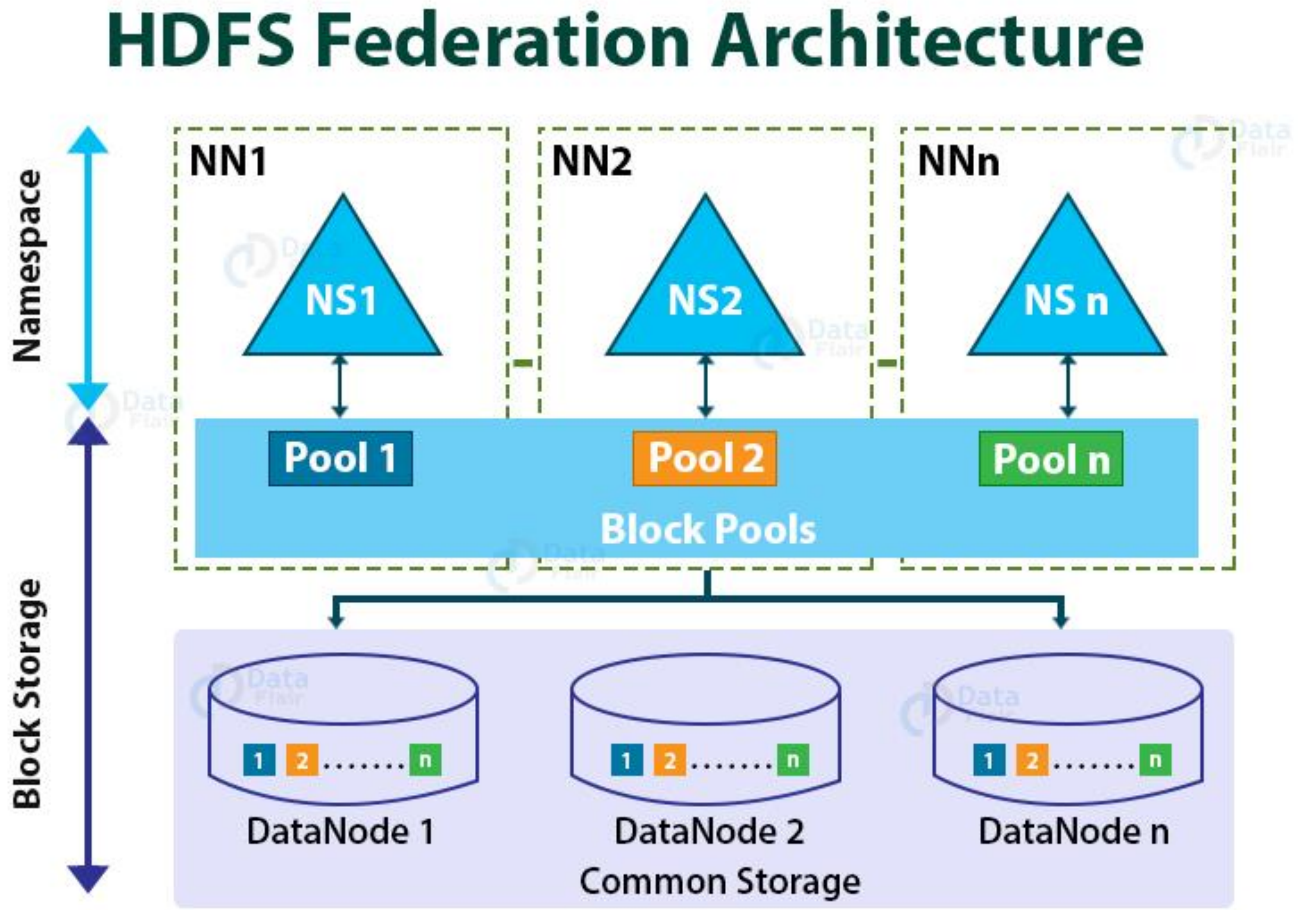
- **HDFS Federation** – horizontal scalability of NameNode
- **NameNode High Availability** – NameNode is no longer a Single Point of Failure
- **YARN** – ability to process Terabytes and Petabytes of data available in HDFS using Non-MapReduce applications such as MPI, GIRAPH
- **Resource Manager** – splits up the two major functionalities of overburdened JobTracker (resource management and job scheduling/monitoring) into two separate daemons: a global Resource Manager and per-application ApplicationMaster

There are additional features such as **Capacity Scheduler** (Enable Multi-tenancy support in Hadoop), **Data Snapshot**, **Support for Windows**, **NFS access**, enabling increased Hadoop adoption in the Industry to solve Big Data problems.

Hadoop 2.x

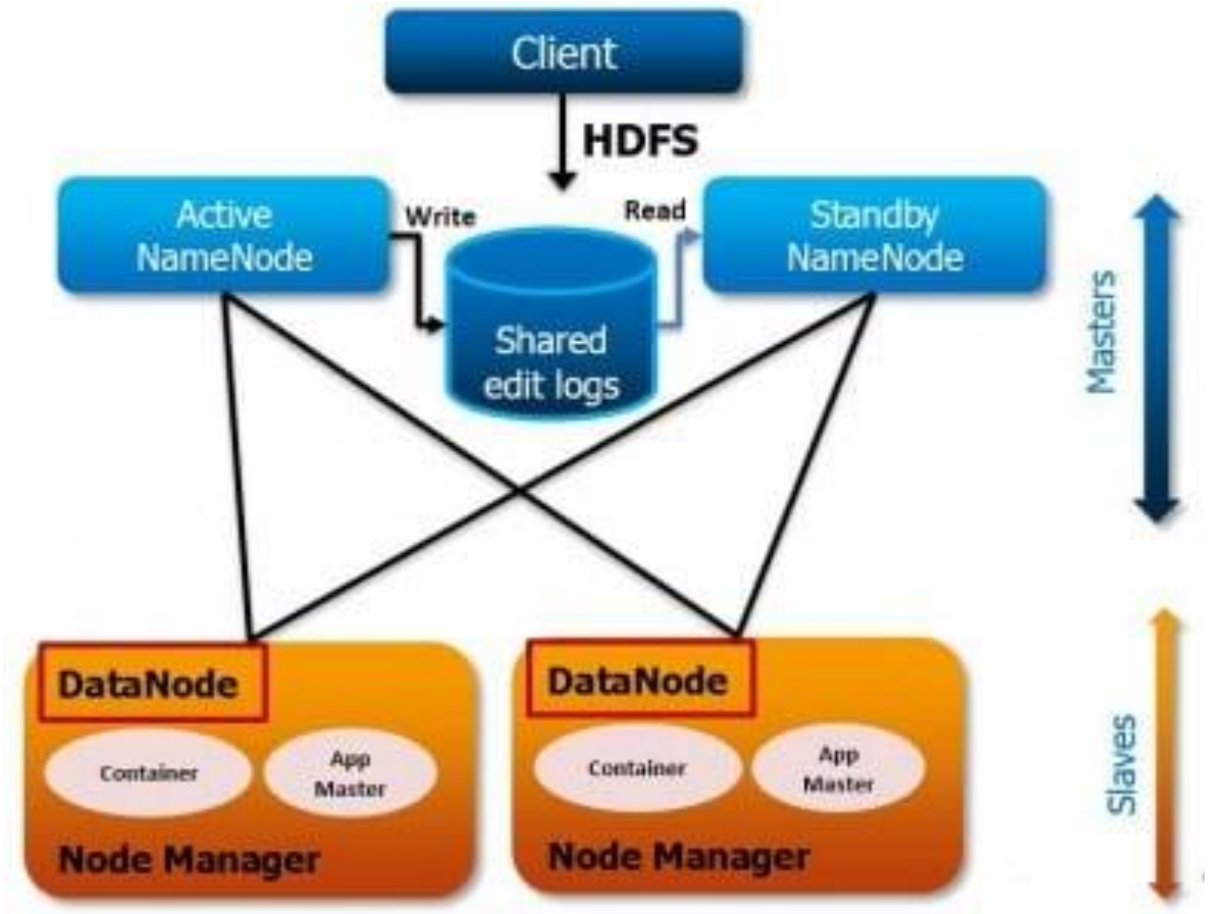
HDFS Federation





Hadoop 2.x

NameNode High Availability



What is YARN?

YARN= Yet Another Resource Negotiator

- YARN is a resource manager
- Created by separating the processing engine and the management function of MapReduce
- Monitors and manages workloads, maintains a multi-tenant environment, manages the high availability features of Hadoop, and implements security controls

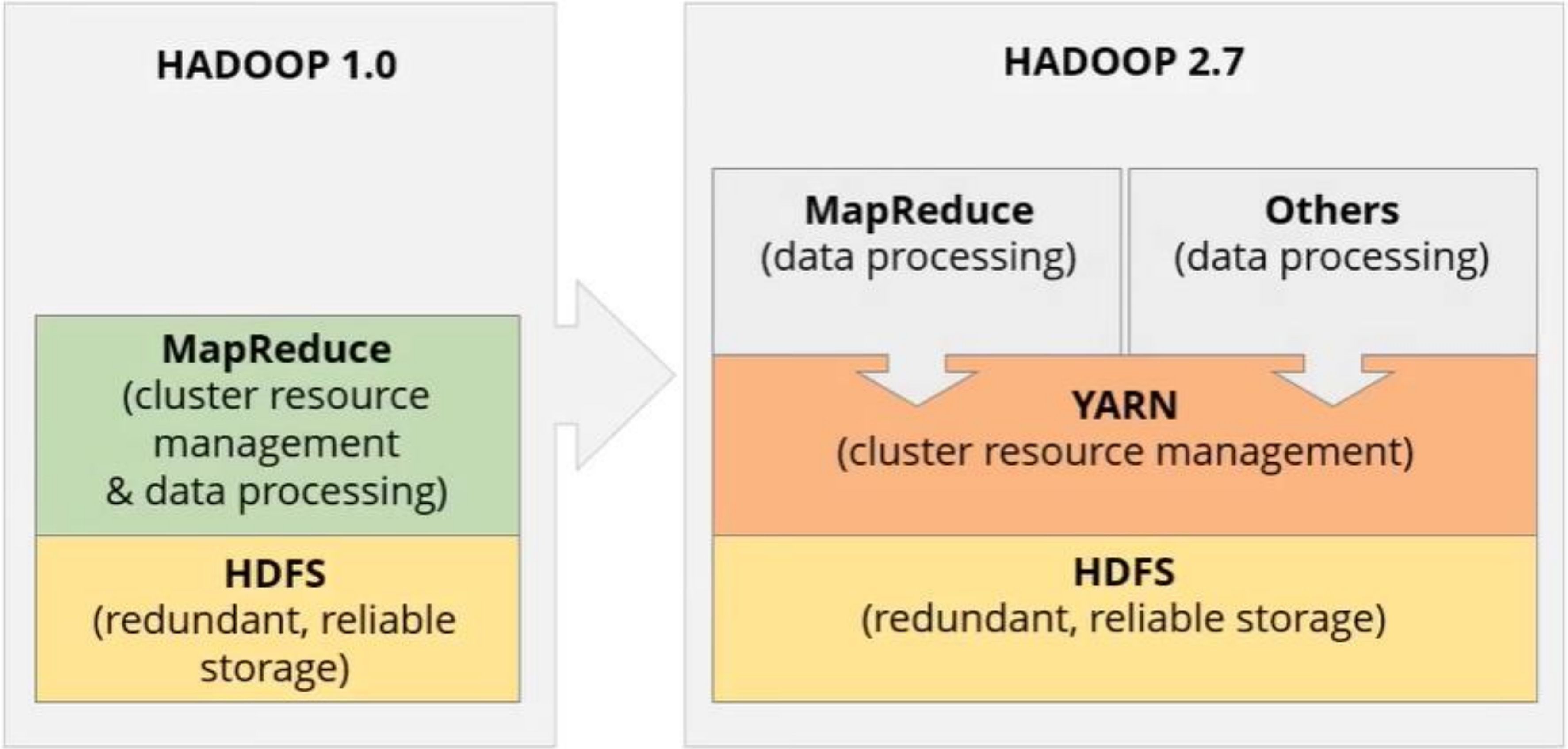
Need for YARN

Before 2012

Users could write MapReduce programs using scripting languages

Since 2012

Users could work on multiple processing models in addition to MapReduce



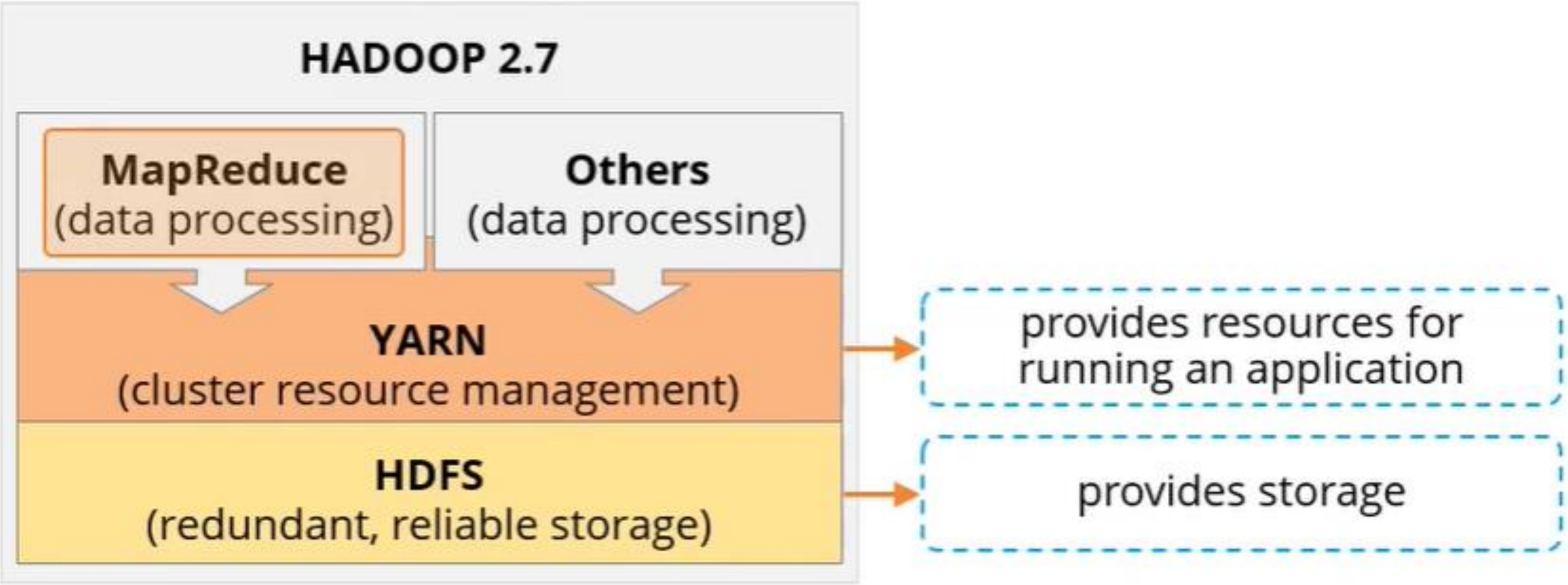
YARN—Use Case

YAHOO!

- Yahoo was the first company to embrace Hadoop in a big way, and it is a trendsetter within the Hadoop ecosystem. In late 2012, it struggled to handle iterative and stream processing of data on Hadoop infrastructure due to MapReduce limitations.
- After implementing YARN in the first quarter of 2013, Yahoo has installed more than 30,000 production nodes on
  - Spark for iterative processing
  - Storm for stream processing
  - Hadoop for batch processing
- Such a solution was possible only after YARN was introduced and multiple processing frameworks were implemented.

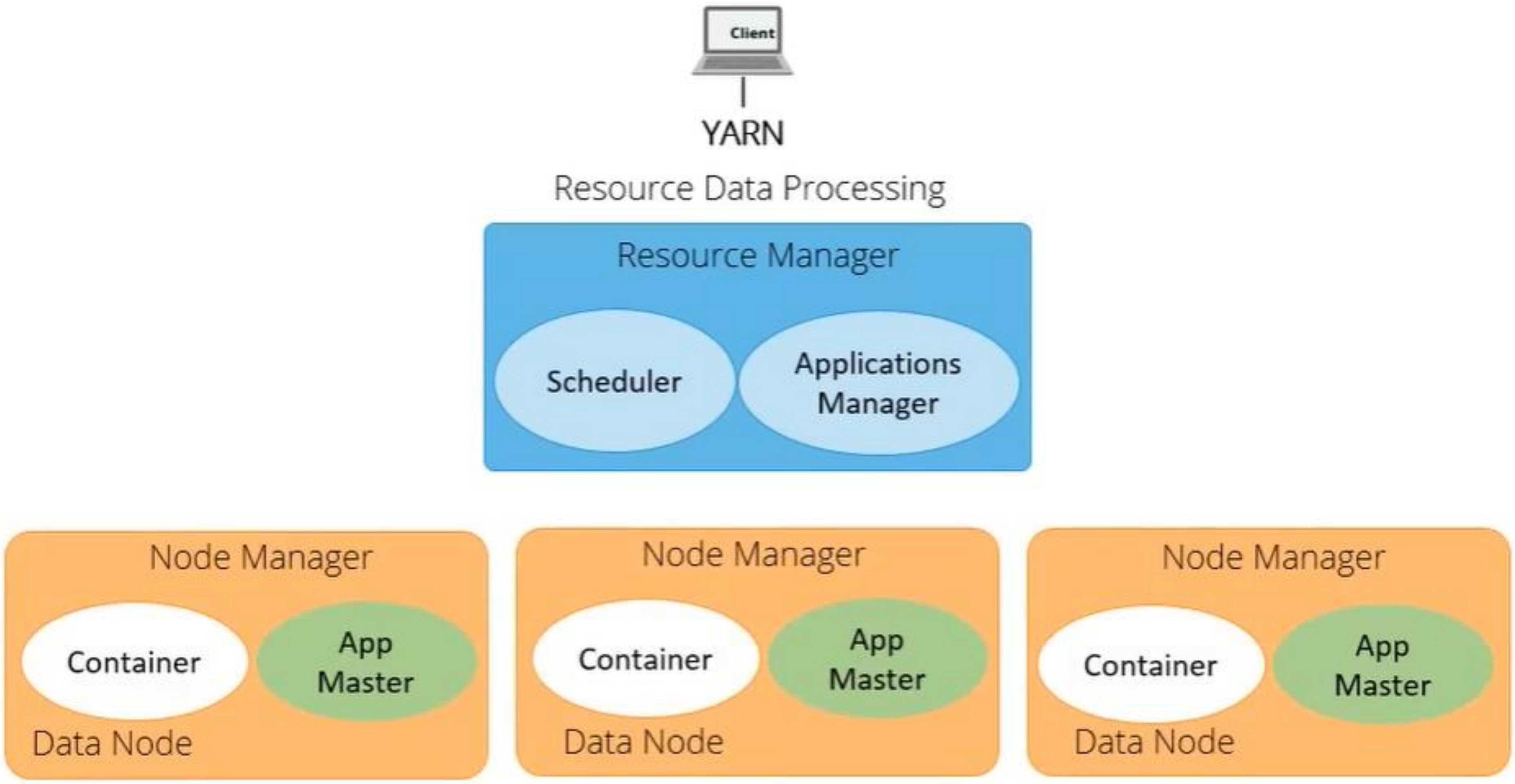
YARN Infrastructure

The YARN Infrastructure is responsible for providing computational resources for application executions.



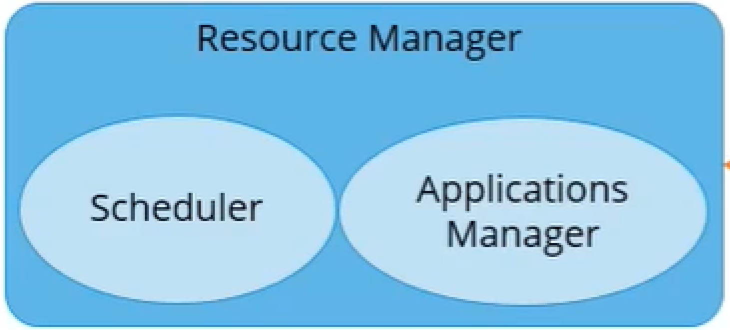
Three Elements of YARN Architecture

The three important elements of the YARN architecture are the ResourceManager, ApplicationMaster, and NodeManager.



YARN Architecture Element—ResourceManager

The RM mediates the available resources in the cluster among competing applications—to maximum cluster utilization.



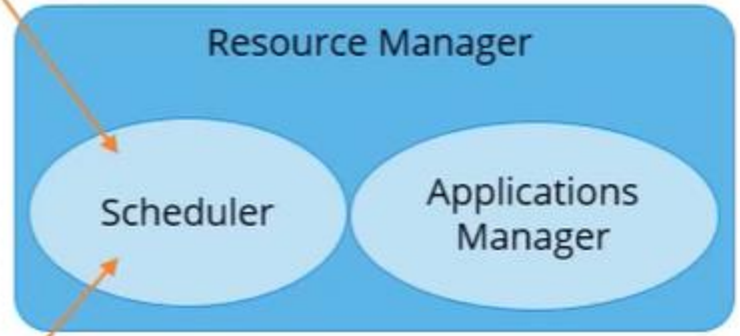
It includes a pluggable scheduler called the YarnScheduler, which allows different policies for managing constraints such as capacity, fairness, and Service Level Agreements.

ResourceManager Component—Scheduler

The Scheduler is responsible for allocating resources to various running applications.

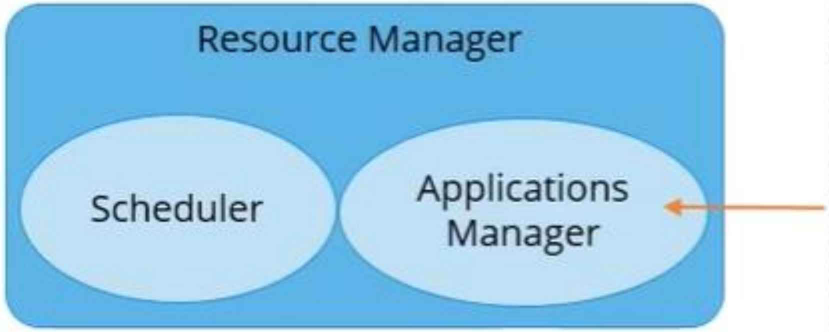
The Scheduler does not monitor or track the status of the application; nor does it restart failed tasks.

The Scheduler has a policy plug-in to partition cluster resources among various applications. Examples: CapacityScheduler, FairScheduler.



ResourceManager Component—ApplicationManager

The ApplicationsManager is an interface which maintains a list of applications that have been submitted, currently running, or completed.

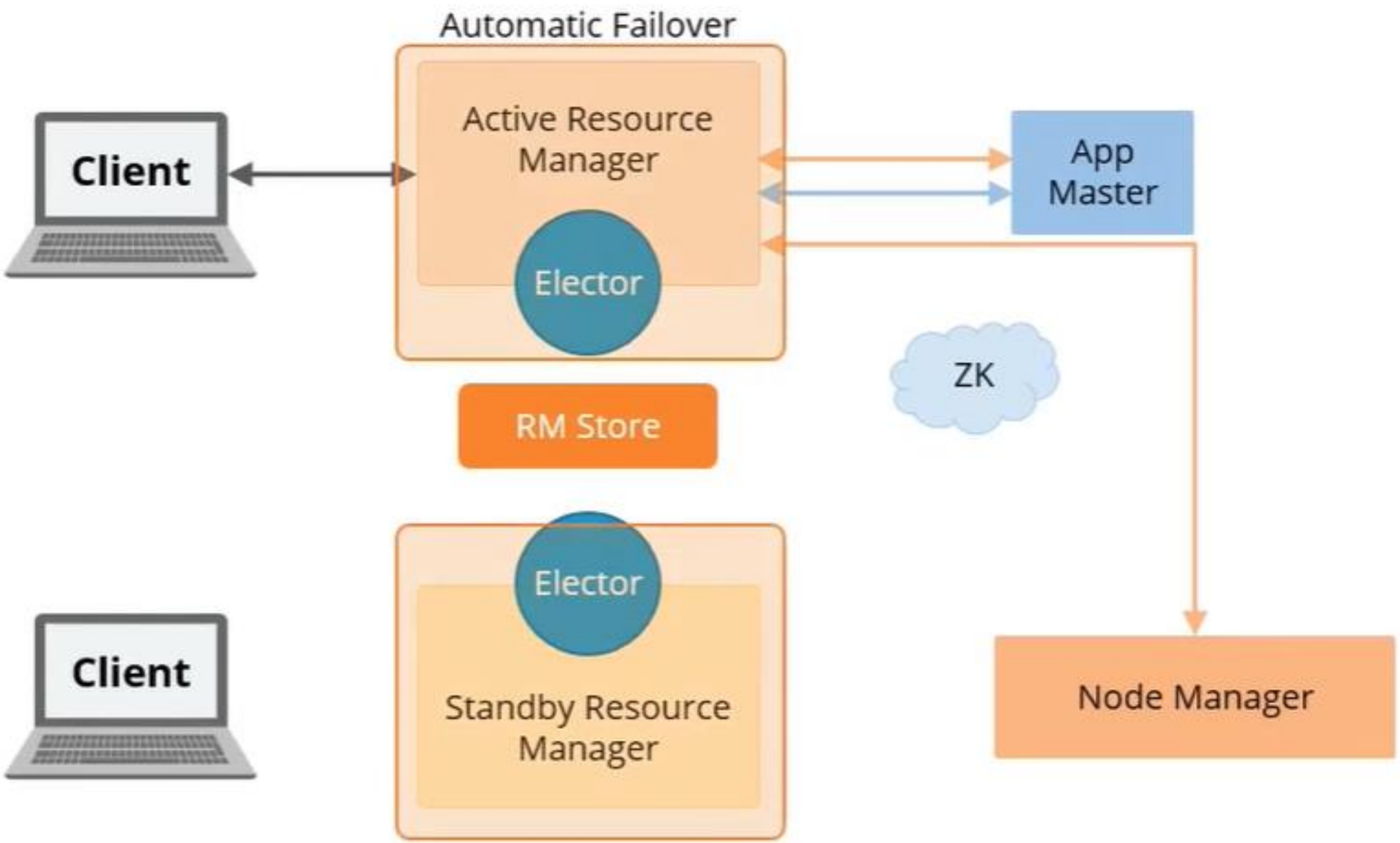


The ApplicationsManager accepts job submissions, negotiates the first container for executing the application, and restarts the ApplicationMaster container on failure.

ResourceManager in High Availability Mode

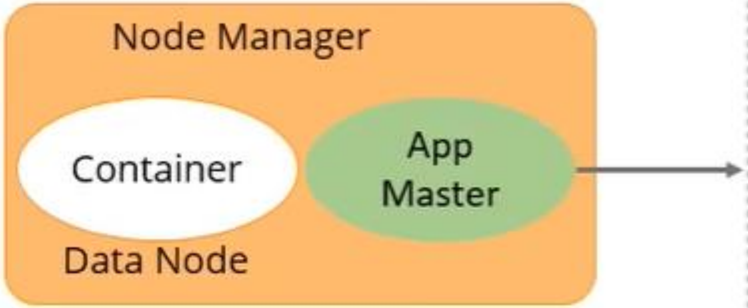
Before Hadoop 2.4, the ResourceManager was the single point of failure in a YARN cluster.

The High Availability, or HA, feature an Active/Standby ResourceManager pair to remove this single point of failure.



YARN Architecture Element—ApplicationMaster

The ApplicationMaster in YARN is a framework-specific library, which negotiates resources from the RM and works with the NodeManager or Managers to execute and monitor containers and their resource consumption.



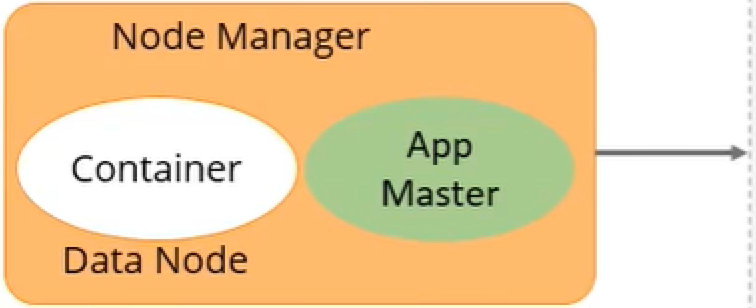
The ApplicationMaster:

- manages the application lifecycle
- makes dynamic adjustments to resource consumption
- manages execution flow
- manages faults
- provides status and metrics to the RM
- interacts with NodeManager and RM using extensible communication protocols
- Is not run as a trusted service

While every application has its own instance of an AppMaster, it is possible to implement an AppMaster for a set of applications as well.

YARN Architecture Element—NodeManager

When a container is leased to an application, the NodeManager sets up the container's environment, including the resource constraints specified in the lease and any dependencies.

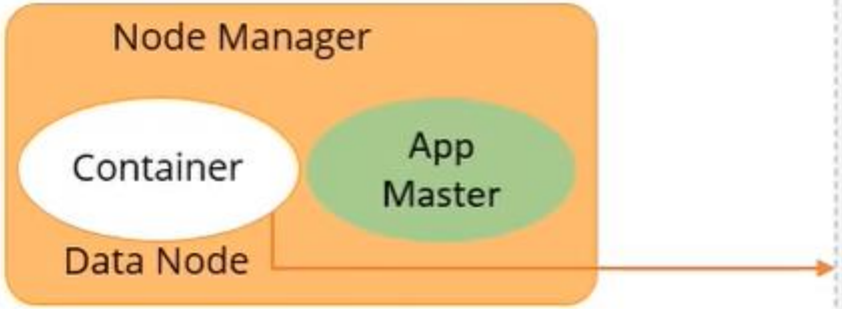


The NodeManager runs on each node and manages the following:

- Container lifecycle management
- Container dependencies
- Container leases
- Node and container resource usage
- Node health
- Log management
- Reporting node and container status to the RM

YARN Container

A YARN container is a result of a successful resource allocation, that is, the RM has granted an application a lease to use specified resources on a specific node.

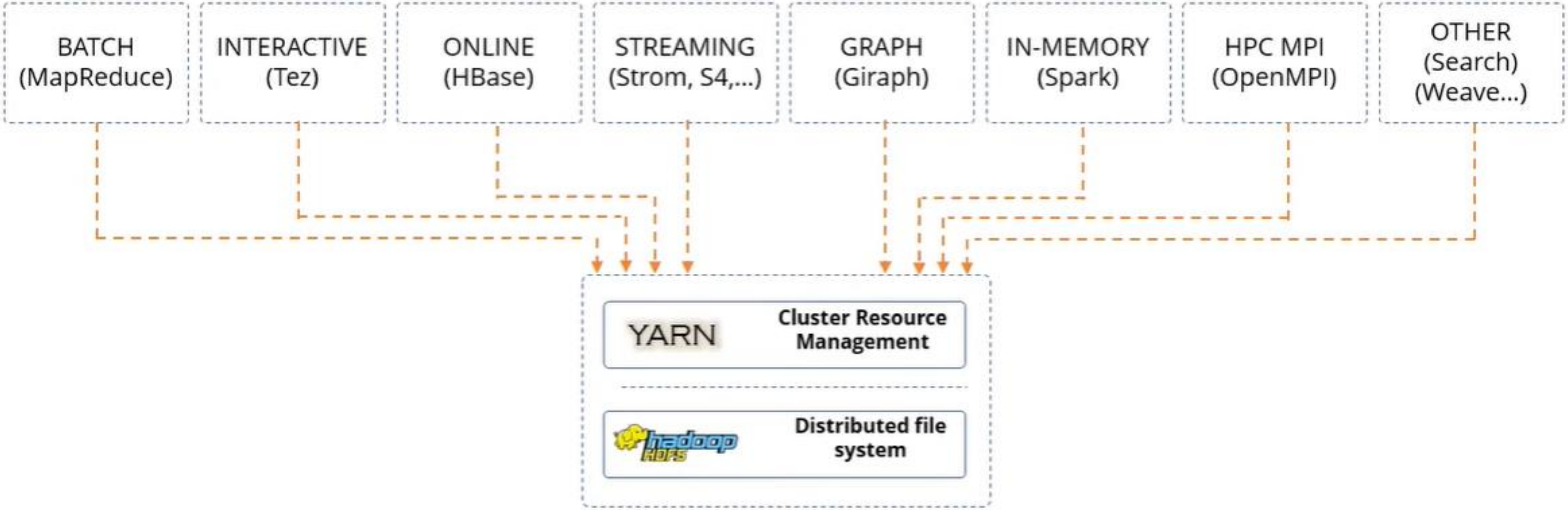


To launch the container, the ApplicationMaster must provide a container launch context (CLC) that includes the following information:

- Environment variables
- Dependencies, that is, local resources such as data files or shared objects needed prior to launch
- Security tokens
- The command necessary to create the process the application wants to launch

Applications on YARN

There can be many different workloads running on a Hadoop YARN cluster.



How YARN Runs an Application

There are five steps involved in YARN to run an application:

01 The client submits an application to the ResourceManager

02 The ResourceManager allocates a container

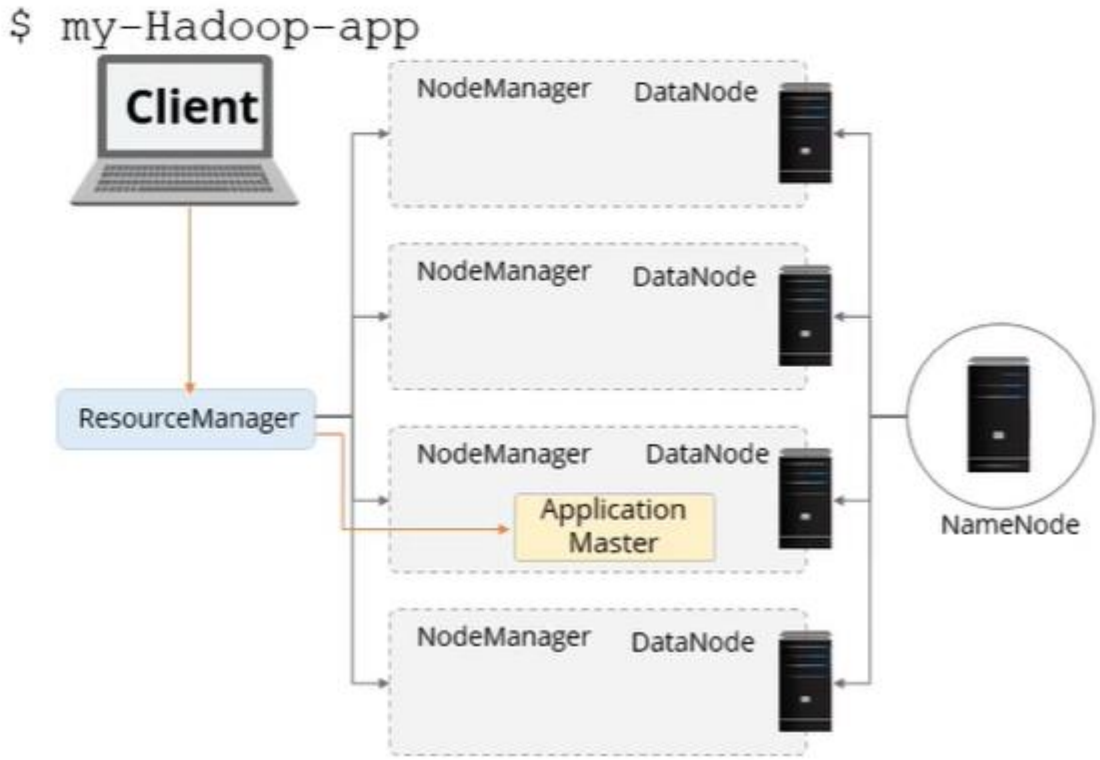
03 The ApplicationMaster contacts the related NodeManager

04 The NodeManager launches the container

05 The container executes the ApplicationMaster

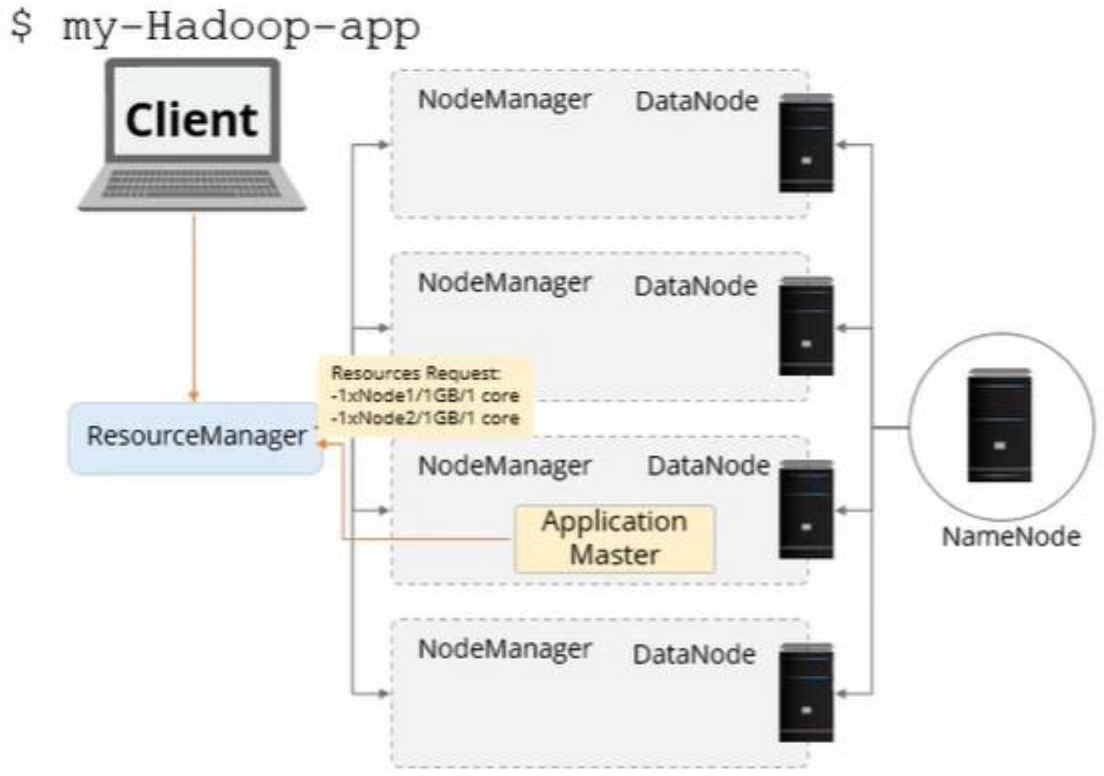
Step1—Application Submitted to ResourceManager

Users submit applications to the ResourceManager by typing the hadoop jar command.



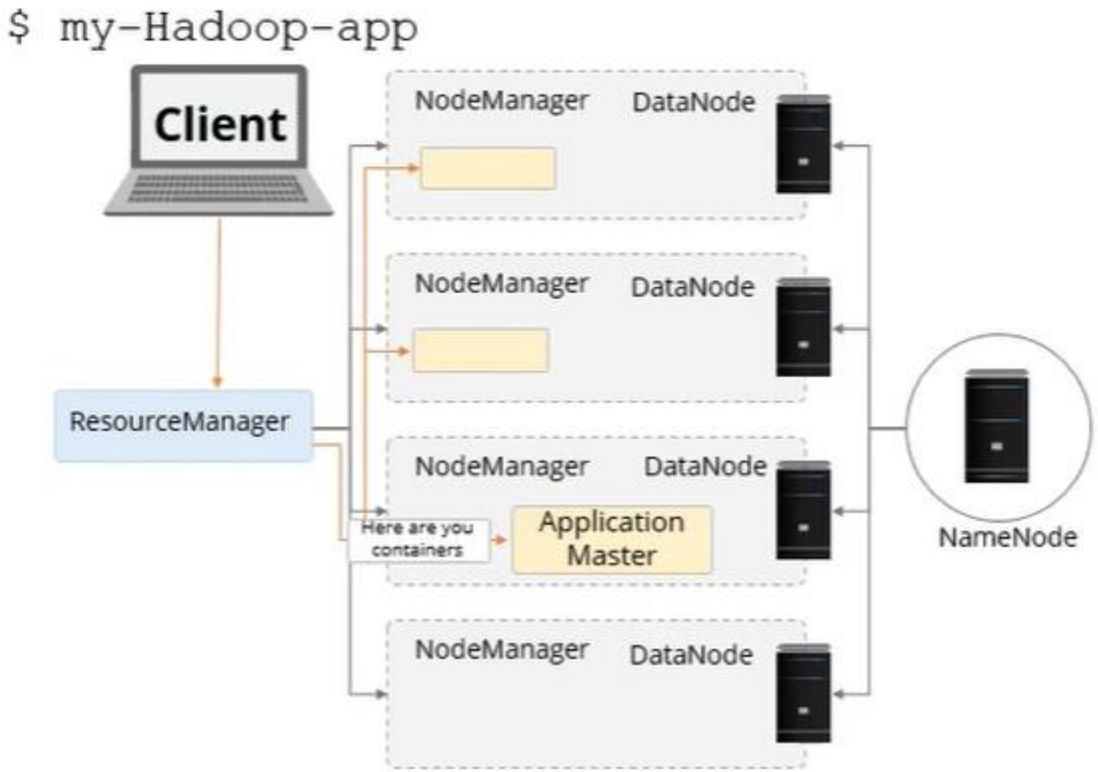
Step2—ResourceManager Allocates a Container

When the ResourceManager accepts a new application submission, one of the first decisions the Scheduler makes is selecting a container.



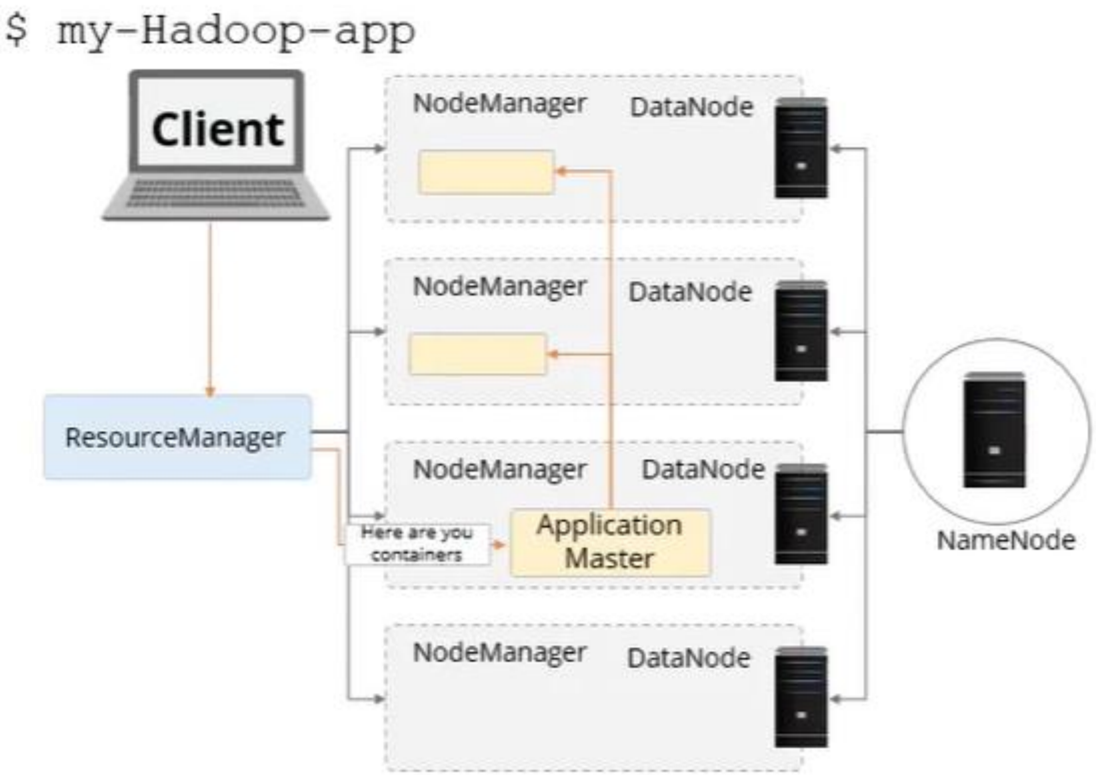
Step3 —ApplicationMaster Contacts NodeManager

After a container is allocated, the ApplicationMaster asks the NodeManager managing the host on which the container was allocated to use these resources to launch an application-specific task.



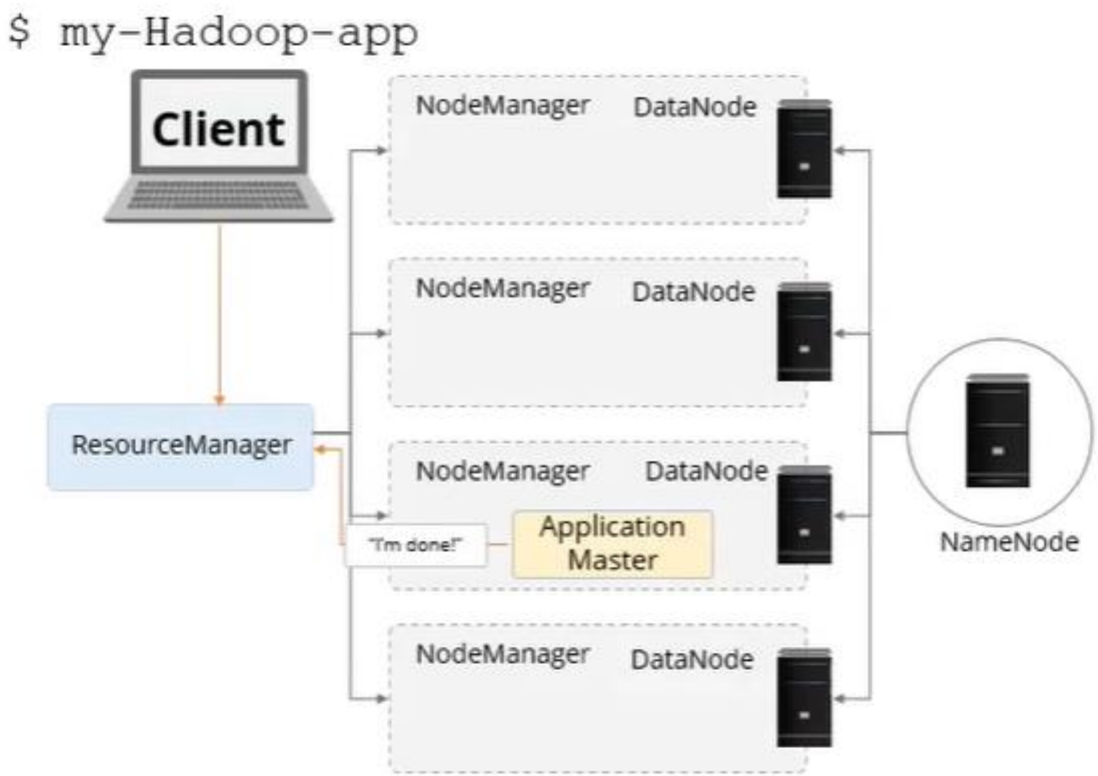
Step4—ResourceManager Launches a Container

The NodeManager does not monitor tasks; it only monitors the resource usage in the containers.

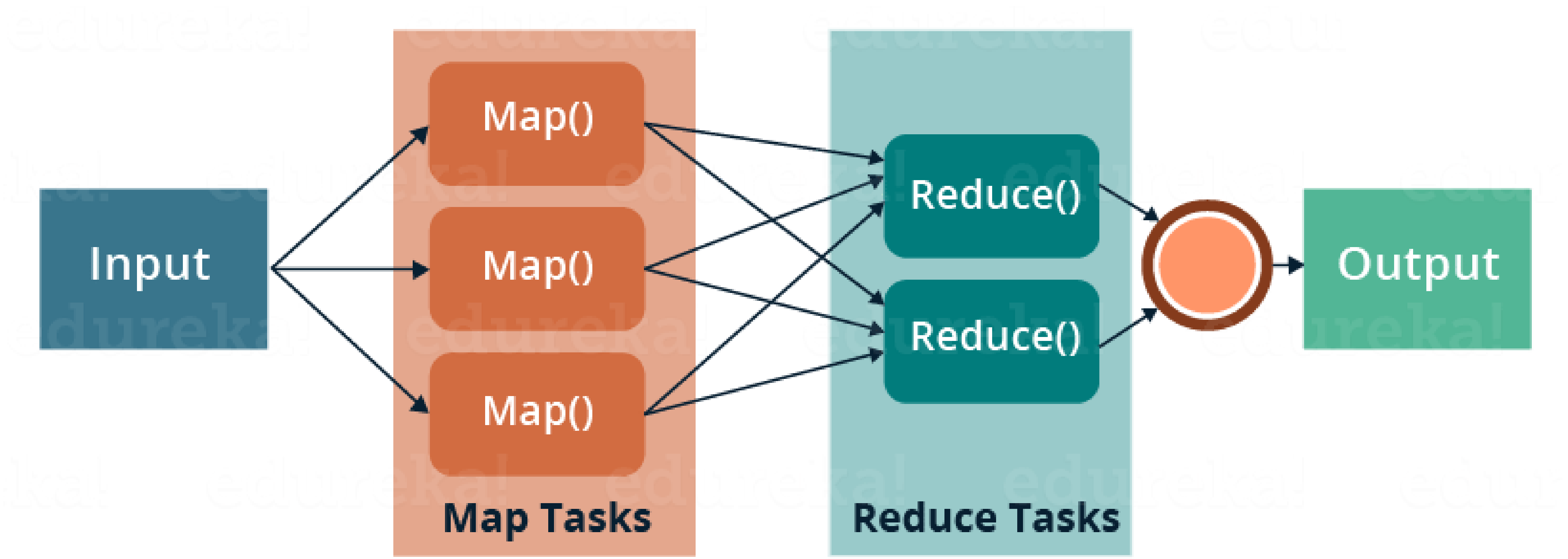


Step5—Container Executes the ApplicationMaster

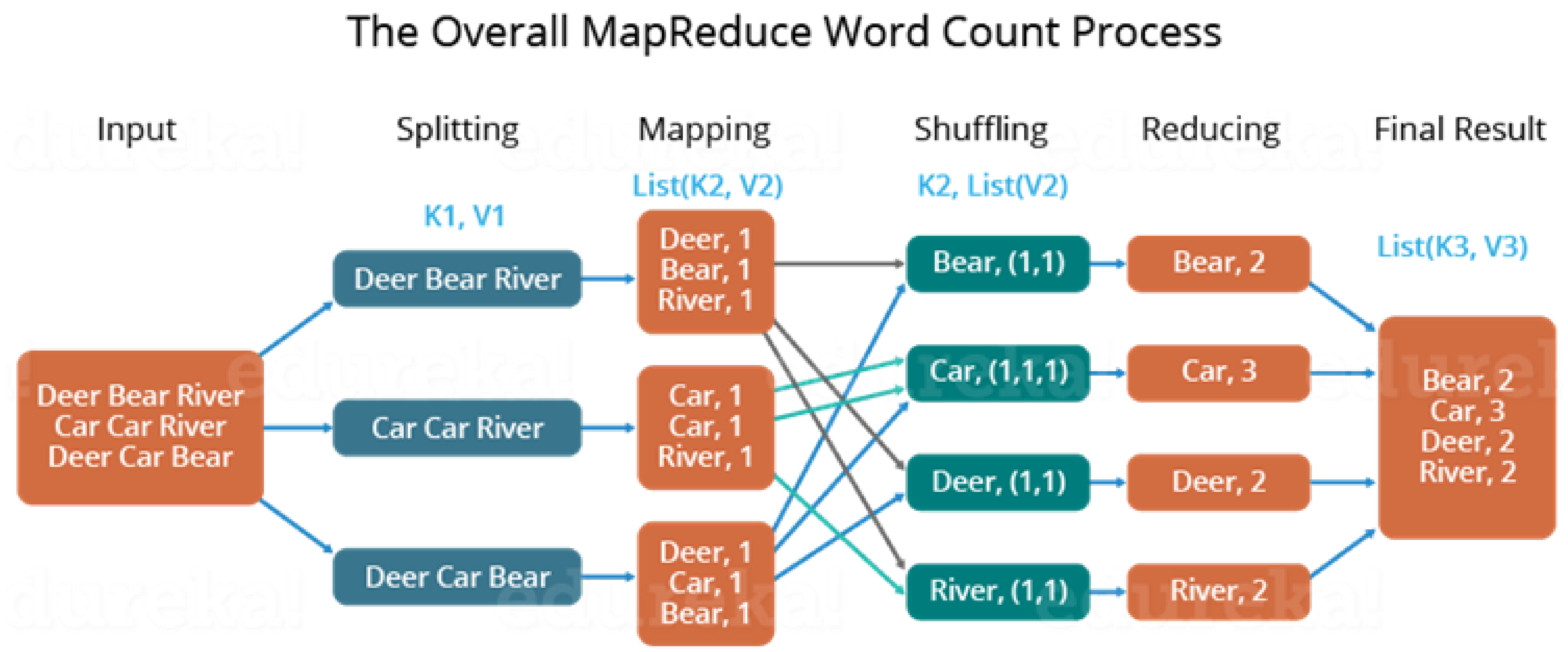
After the application is complete, the ApplicationMaster shuts itself and releases its own container.



Hadoop MapReduce



Hadoop MapReduce



Reduce side Join

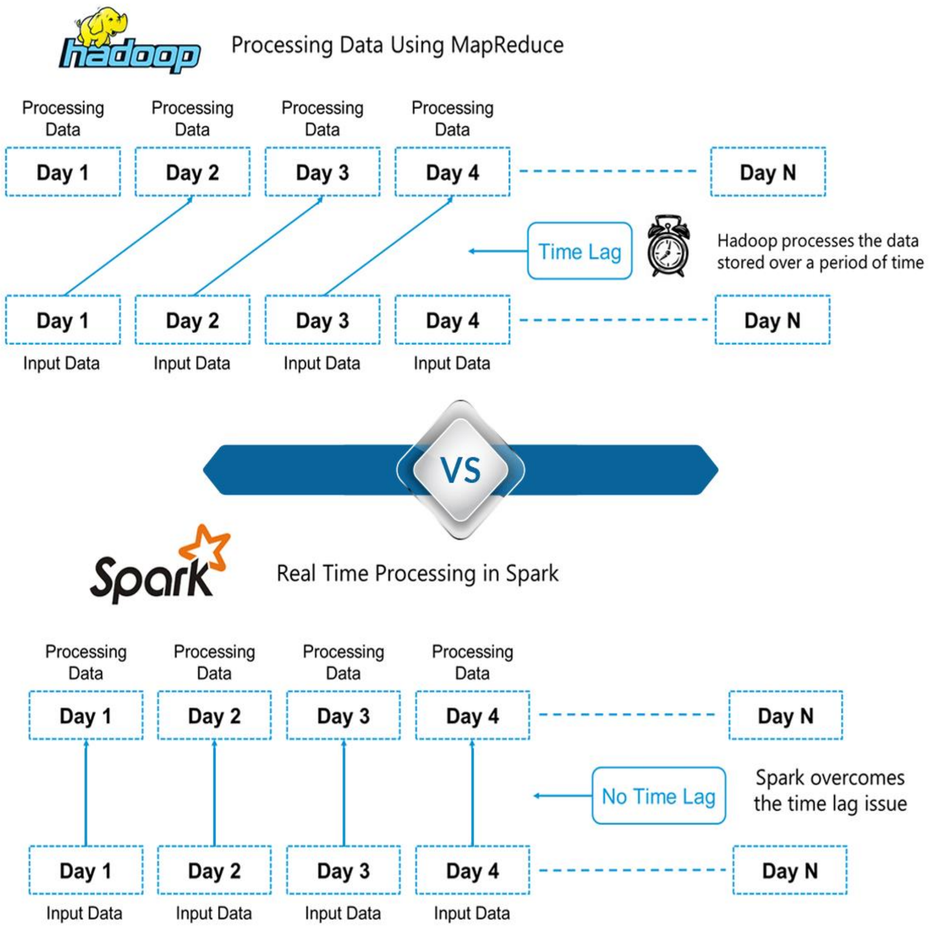
Cust ID	First Name	Last Name	Age	Profession
4000001	Kristina	Chung	55	Pilot
4000002	Paige	Chen	74	Teacher
4000003	Sherri	Melton	34	Firefighter
4000004	Gretchen	Hill	66	Engineer
………	………	………	………	………

Fig: cust_details

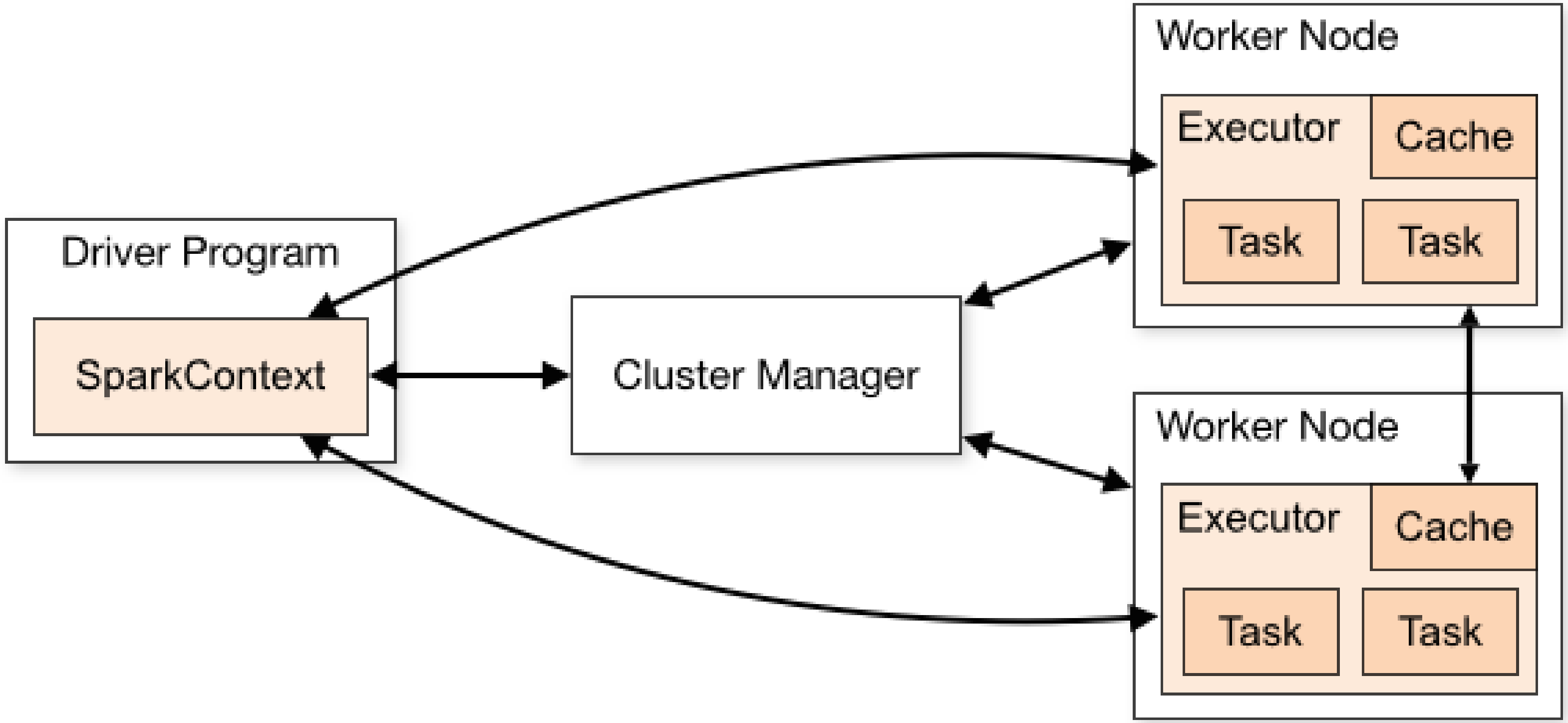
Trans ID	Date	Cust ID	Amount	Game Type	Equipment	City	State	Mode
0000000	06-26-2011	4000001	40.33	Exercise & Fitness	Cardio Machine Accessories	Clarksville	Tennessee	credit
0000001	05-05-2011	4000002	198.44	Exercise & Fitness	Weightlifting Gloves	Long Beach	California	credit
0000002	06-17-2011	4000002	5.58	Exercise & Fitness	Weightlifting Machine Accessories	Anaheim	California	credit
0000003	06-14-2011	4000003	198.19	Gymnastics	Gymnastics Rings	Milwaukee	Wisconsin	credit
0000004	12-28-2011	4000002	98.81	Team Sports	Field Hockey	Nashville	Tennessee	credit
0000005	02-14-2011	4000004	193.63	Outdoor Recreation	Camping & Backpacking & Hiking	Chicago	Illinois	credit
0000006	10-17-2011	4000005	27.89	Puzzles	Jigsaw Puzzles	Charleston	South Carolina	credit
…….	…….	…….	…….	…….	…….	…….	…….	…….

Fig: transaction_details

Apache Spark

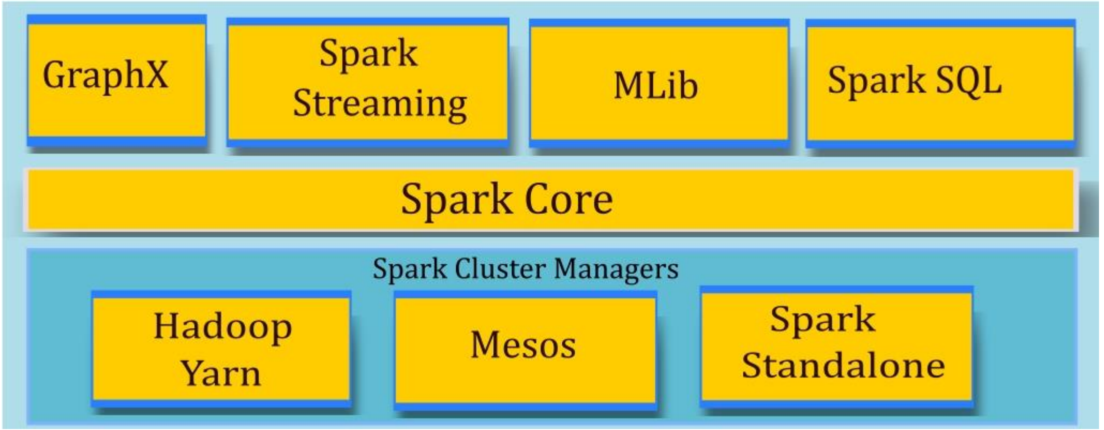


Apache Spark



Components

- Spark Core and Resilient Distributed Datasets or RDDs
- Spark SQL
- Spark Streaming
- Machine Learning Library or MLlib
- GraphX



Components


Spark
Spark SQL
Spark Streaming
Spark MLlib
Spark GraphX
Spark R
Scala
R