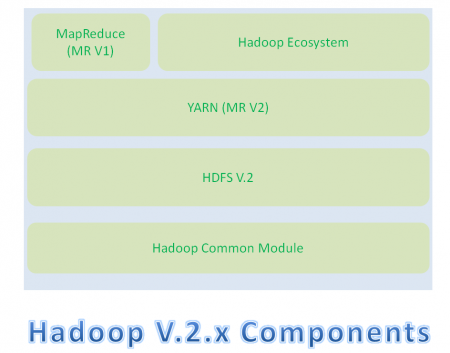
**Problem Statement**

* **List the components of Hadoop 2.X and explain each component in detail**

**Solution:**

**Hadoop 2.x Architecture**

Apache Hadoop 2.x or later versions are using the following Hadoop Architecture. It is a Hadoop 2.x High-level Architecture.



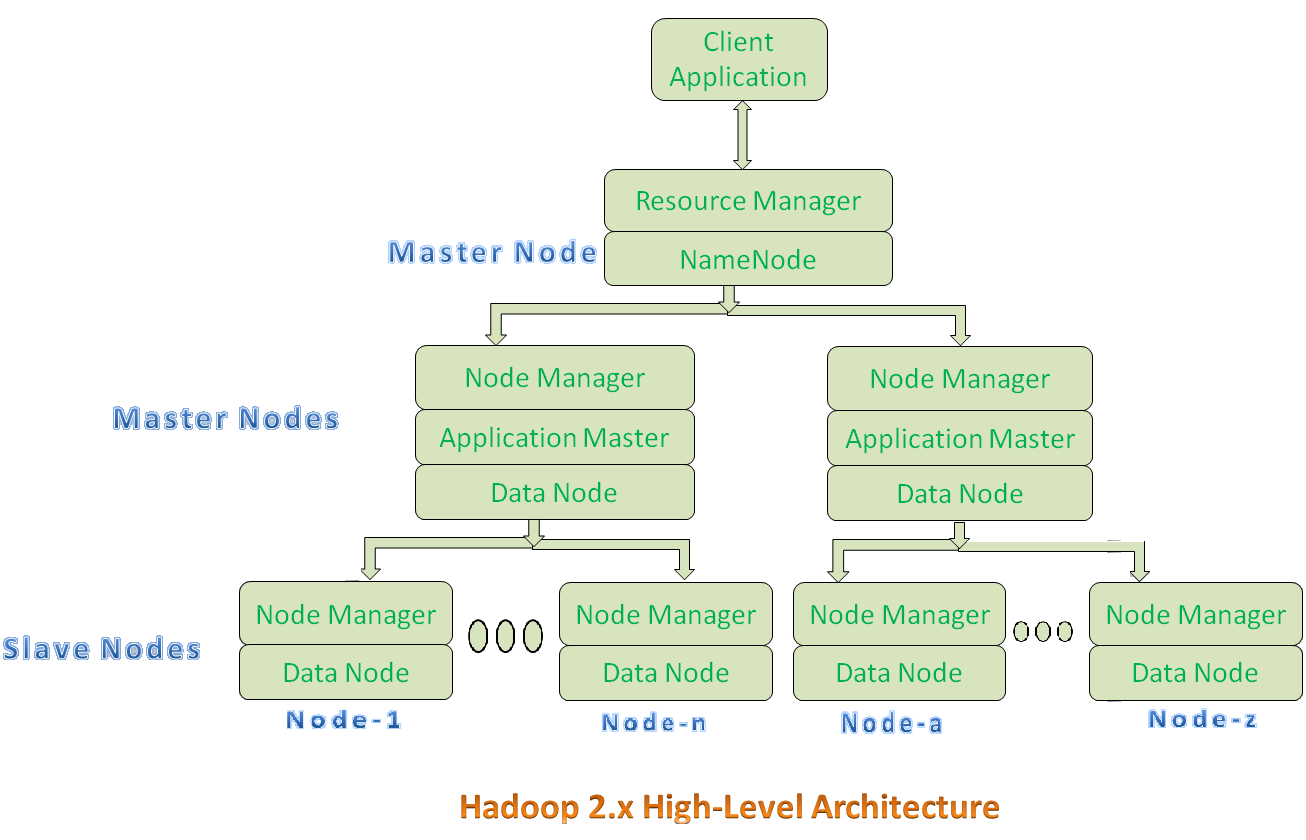
* Hadoop Common Module is a Hadoop Base API (A Jar file) for all Hadoop Components. All other components works on top of this module.
* HDFS stands for Hadoop Distributed File System. It is also known as HDFS V2 as it is part of Hadoop 2.x with some enhanced features. It is used as a Distributed Storage System in Hadoop Architecture.
* YARN stands for Yet Another Resource Negotiator. It is new Component in Hadoop 2.x Architecture. It is also known as “MR V2”.
* MapReduce is a Batch Processing or Distributed Data Processing Module. It is also known as “MR V1” as it is part of Hadoop 1.x with some updated features.
* Remaining all Hadoop Ecosystem components work on top of these three major components: HDFS, YARN and MapReduce.

### Hadoop 2.x Major Components

Hadoop 2.x has the following three Major Components:

* HDFS
* YARN
* MapReduce

These three are also known as Three Pillars of Hadoop 2.x.



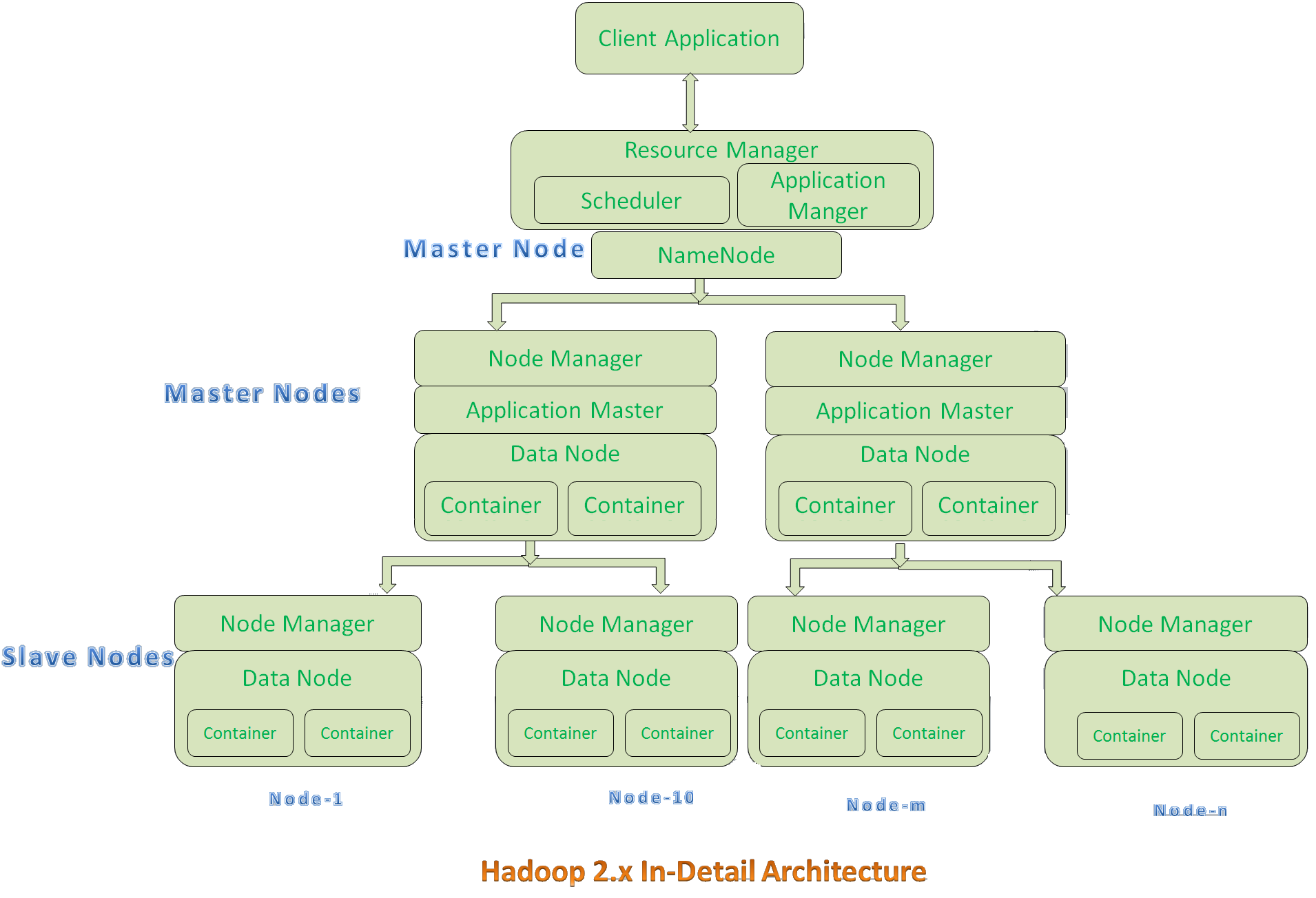
* All Master Nodes and Slave Nodes contain both MapReduce and HDFS Components.
* One Master Node has two components:
  1. Resource Manager(YARN or MapReduce v2)
  2. HDFS

HDFS component is also known as NameNode. NameNode is used to store Meta Data.

* In Hadoop 2.x, some more Nodes acts as Master Nodes as shown in the above diagram. Each this 2nd level Master Node has 3 components:
  1. Node Manager
  2. Application Master
  3. Data Node
* Each this 2nd level Master Node again contains one or more Slave Nodes as shown in the above diagram.
* These Slave Nodes have two components:
  1. Node Manager
  2. HDFS

Its HDFS component is also knows as Data Node. Its Data Node component is used to store actual our application Big Data. These nodes does not contain Application Master component.

**Hadoop 2.x Components In-detail Architecture**



**HDFS**

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on large clusters (thousands of computers) of small computer machines in a reliable, fault-tolerant manner.

HDFS uses a **Master/Slave Architecture**

Master: NameNode

Slave: {Datanode}…..{Datanode}

An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes.

* **NameNode** executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes.
* **DataNodes** are responsible for serving read and write requests from the file system’s clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

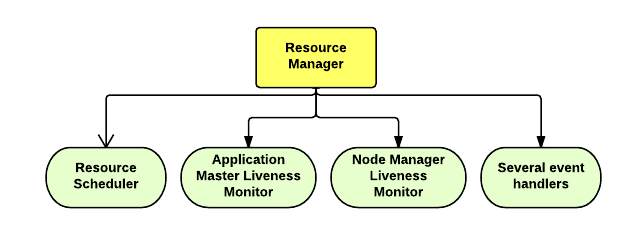


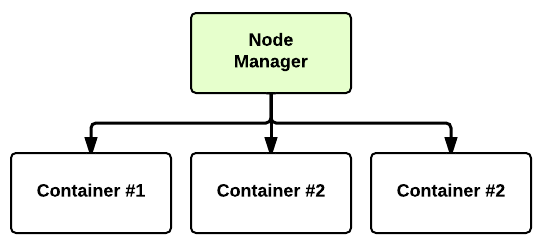
The NameNode and DataNode are pieces of software designed to run on commodity machines. These machines typically run a GNU/Linux operating system (OS). HDFS is built using the Java language; any machine that supports Java can run the NameNode or the DataNode software. Usage of the highly portable Java language means that HDFS can be deployed on a wide range of machines. A typical deployment has a dedicated machine that runs only the NameNode software. Each of the other machines in the cluster runs one instance of the DataNode software. The architecture does not preclude running multiple DataNodes on the same machine but in a real deployment that is rarely the case.

The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and repository for all HDFS metadata. The system is designed in such a way that user data never flows through the NameNode.

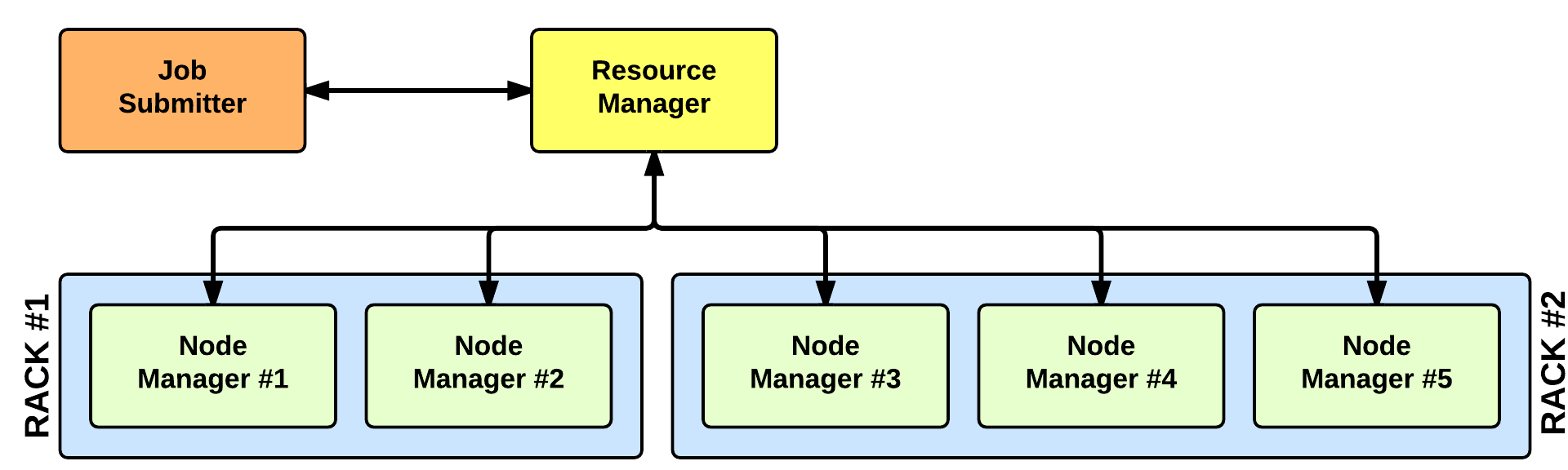
**YARN**

* **YARN is** the framework responsible for providing the computational resources (e.g., CPUs, memory, etc.) needed for application executions. Two important elements are:
  + **Resource Manager** (one per cluster) is the master. It knows where the slaves are located (Rack Awareness) and how many resources they have. It runs several services; the most important is the **Resource Scheduler** which decides how to assign the resources.



* + **Node Manager** (many per cluster) is the slave of the infrastructure. When it starts, it announces himself to the Resource Manager. Periodically, it sends a heartbeat to the Resource Manager. Each Node Manager offers some resources to the cluster. Its resource capacity is the amount of memory and the number of cores. At run-time, the Resource Scheduler will decide how to use this capacity:
  + **Container** is a fraction of the NM capacity and it is used by the client for running a program.

[**YARN: Application Startup**](http://ercoppa.github.io/HadoopInternals/HadoopArchitectureOverview.html#yarn-application-startup)

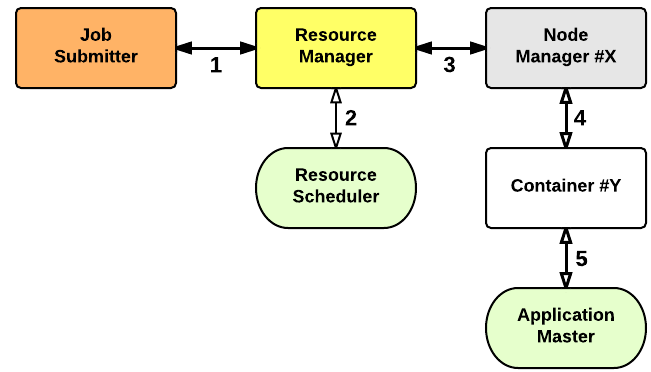


In YARN, there are at least three actors:

* the **Job Submitter** (the client)
* the **Resource Manager** (the master)
* the **Node Manager** (the slave)

The application startup process is the following:

1. a client submits an application to the Resource Manager
2. the Resource Manager allocates a container
3. the Resource Manager contacts the related Node Manager
4. the Node Manager launches the container
5. the Container executes the **Application Master**



* **Application Master** is responsible for the execution of a single application. It asks for containers to the Resource Scheduler (Resource Manager) and executes specific programs (e.g., the main of a Java class) on the obtained containers. The Application Master knows the application logic and thus it is framework-specific. The MapReduce framework provides its own implementation of an Application Master.

The Resource Manager is a single point of failure in YARN. Using Application Masters, YARN is spreading over the cluster the metadata related to running applications. This reduces the load of the Resource Manager and makes it fast recoverable.

## MapReduce Algorithm

MapReduce is a Distributed Data Processing Algorithm, introduced by **Google** in its MapReduce Tech Paper. MapReduce Algorithm is mainly inspired by Functional Programming model. MapReduce algorithm is mainly useful to process huge amount of data in parallel, reliable and efficient way in cluster environments. It uses **Divide and Conquer technique** to process large amount of data. It divides input task into smaller and manageable sub-tasks (They should be executable independently) to execute them in-parallel.

**MapReduce Algorithm Steps**

MapReduce Algorithm uses the following three main steps:

1. Map Function
2. Shuffle Function
3. Reduce Function

### Map Function

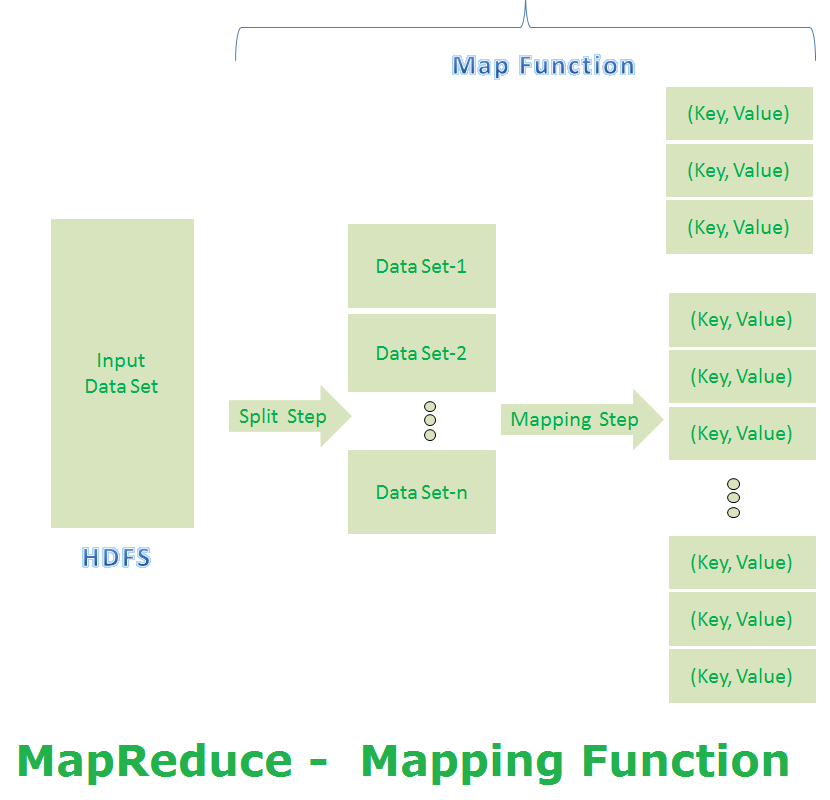
Map Function is the first step in MapReduce Algorithm. It takes input tasks (say DataSets. I have given only one DataSet in below diagram.) and divides them into smaller sub-tasks. Then perform required computation on each sub-task in parallel.

This step performs the following two sub-steps:

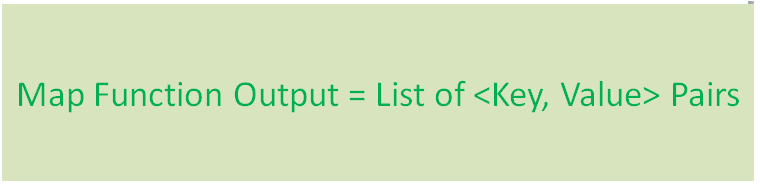
1. Splitting
2. Mapping

* Splitting step takes input DataSet from Source and divide into smaller Sub-DataSets.
* Mapping step takes those smaller Sub-DataSets and perform required action or computation on each Sub-DataSet.

The output of this Map Function is a set of key and value pairs as <Key, Value> as shown in the below diagram.



**MapReduce First Step Output:**



**Shuffle Function**

It is the second step in MapReduce Algorithm. Shuffle Function is also known as “Combine Function”.

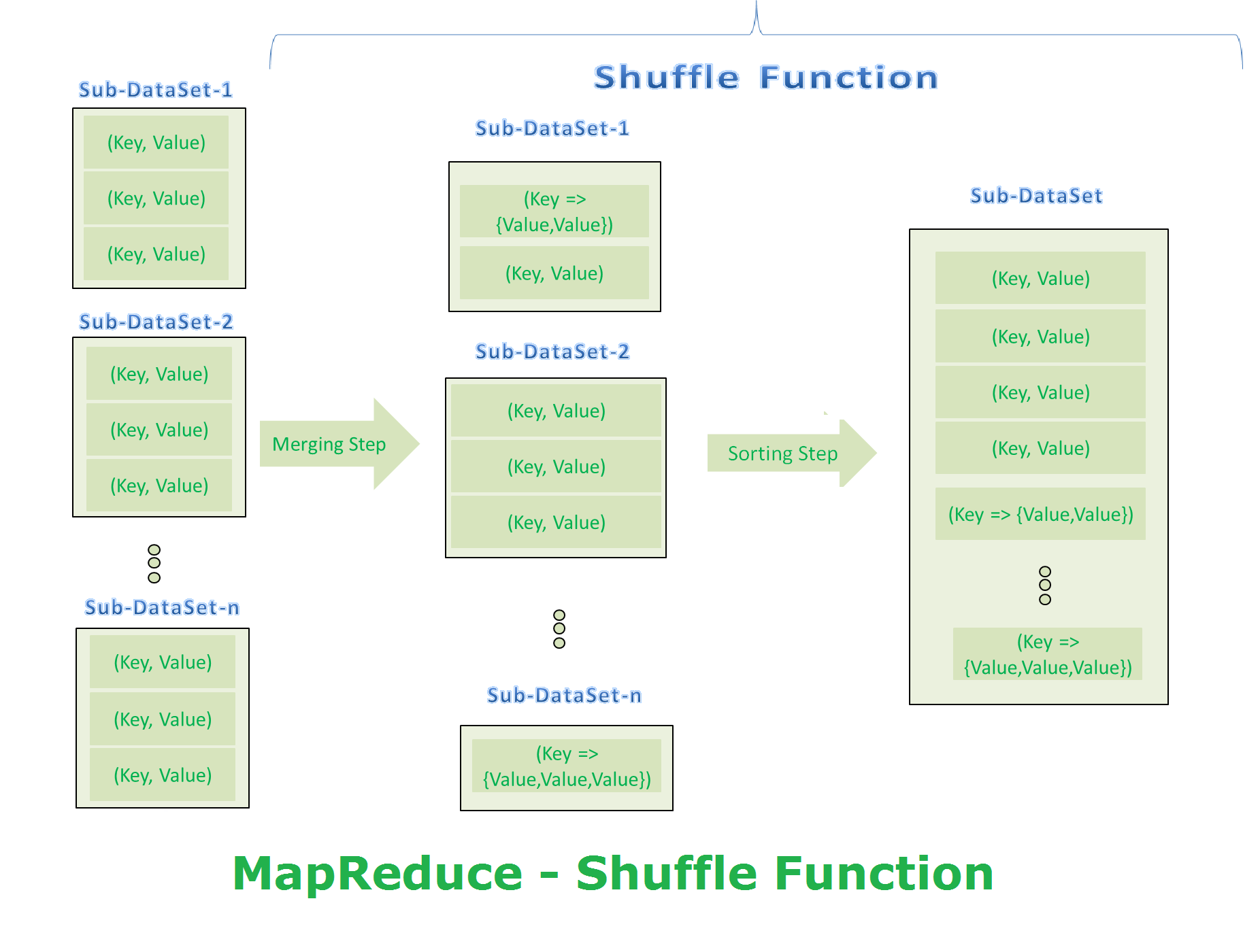
It performs the following two sub-steps:

1. Merging
2. Sorting

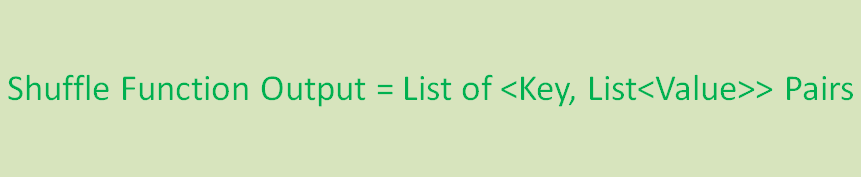
It takes a list of outputs coming from “Map Function” and performs these two sub-steps on each and every key-value pair.

* Merging step combines all key-value pairs which have same keys (that is grouping key-value pairs by comparing “Key”). This step returns <Key, List<Value>>.
* Sorting step takes input from Merging step and sorts all key-value pairs by using Keys. This step also returns <Key, List<Value>> output but with sorted key-value pairs.

Finally, Shuffle Function returns a list of <Key, List<Value>> sorted pairs to next step.



**MapReduce Second Step Output:**

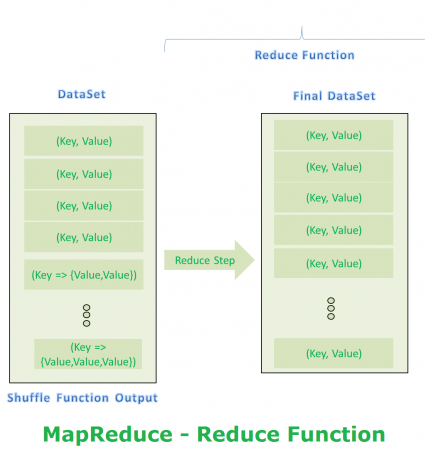


### Reduce Function

It is the final step in MapReduce Algorithm. It performs only one step:

1. Reduce step.

It takes list of <Key, List<Value>> sorted pairs from Shuffle Function and perform reduce operation as shown below.

[](https://cdn.journaldev.com/wp-content/uploads/2015/09/reduce-function.png)

**MapReduce Final Step Output:**

[](https://cdn.journaldev.com/wp-content/uploads/2015/09/reduce-function-output.png)