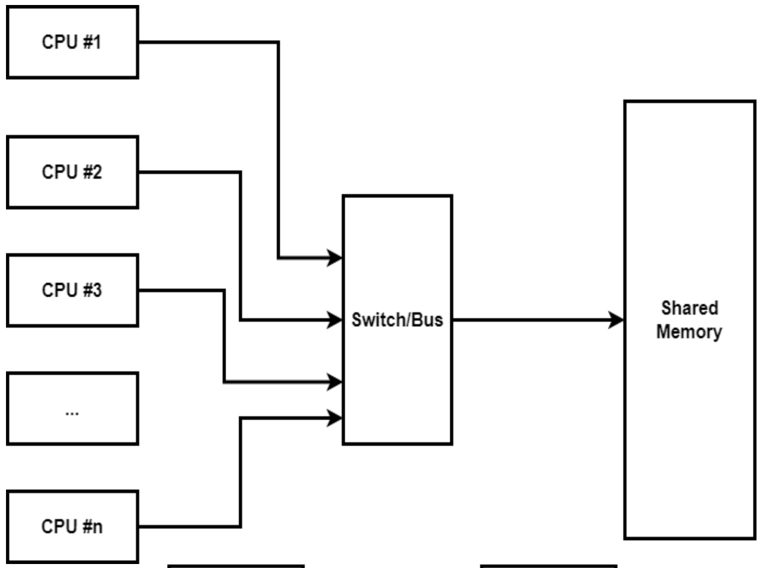
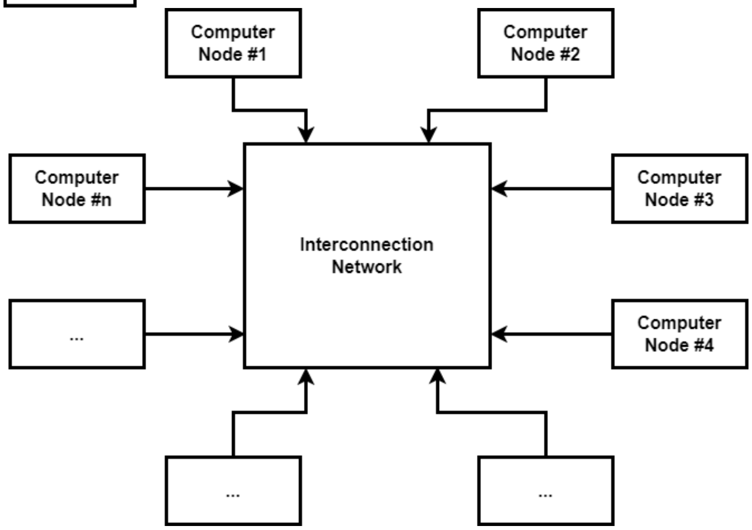
Hadoop Framework

- Overview
    - The need to process large volumes of data is not new.
    - We bring processing to the data: Data is processed in parallel and in a distributed fashion at the location in which it is stored.
    - Big data processing is a set of frameworks storing and accessing enormous amounts of information and extracting meaningful insights.
    - It is a set of processes that acquire, clean, and analyze big data.
    - There are different types of processing big data:
        - Parallel data processing
        - Distributed data processing
        - Hadoop

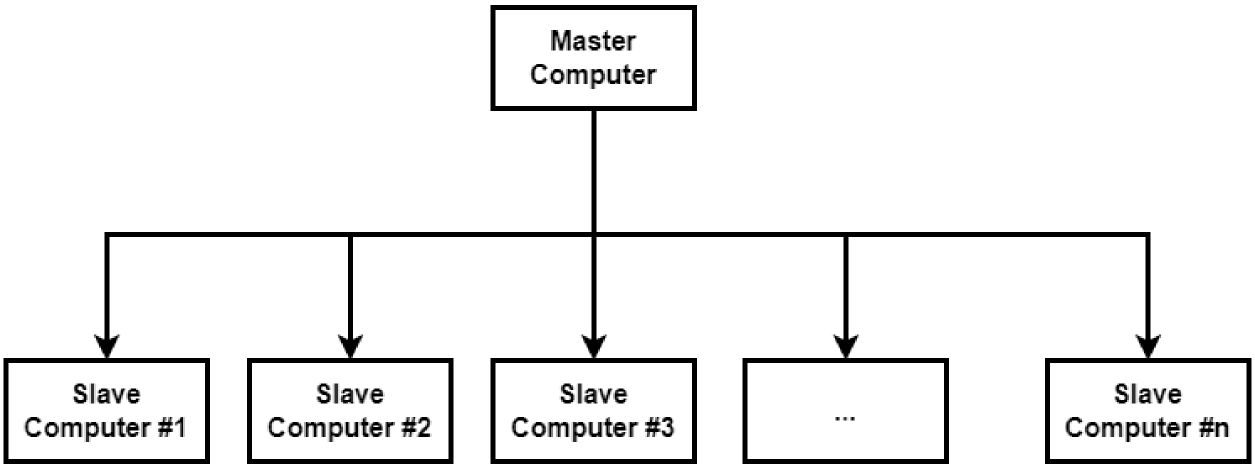
- Big Data Processing Frameworks
    - To process big data, we would need parallel and distributed data technologies:
        - Speed up the processing of applications:
            - Parallel computing
            - Cluster computing
        - A framework to handle very large dataset
            - Distributed file system
    - Parallel Computing:
        - There are two types of architecture:
            - Shared Memory vs Distributed Memory





- Cluster Computing
  - It is a process that connects multiple computers via a local network or wide area network to solve large and complex software applications.
  - It is scalable.
  - Reliability: The system is not affected if a computer goes down.
  - It is cheaper (commodity hardware) and flexible (add more computers)
  - Cluster Interconnection Topology:
    - The most commonly used interconnection is the Master/Slave topology.

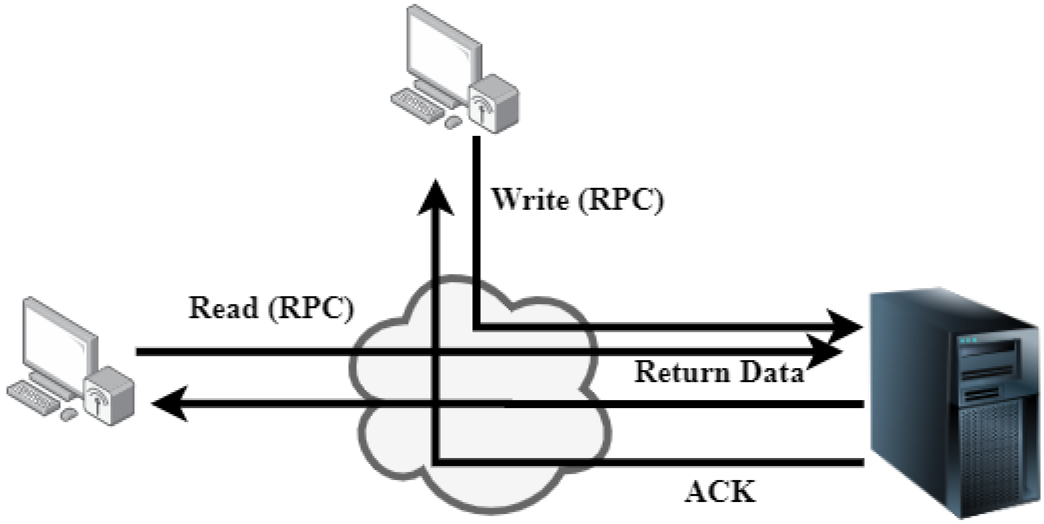
- It is a model where one computer (Master) communicates with one or more computers called Slaves.



- The master node manages all slaves and assigns them tasks.
- The slave nodes do the actual computing and store data.
- Example of cluster computer: NIH Beowulf Cluster

- Distributed File System:
  - Limitation of disk capacity
  - Handling Large File System
  - Distributed File System is similar to local File System
  - Data is distributed on computers (nodes) via network.
  - It enables programs to store and access remote files exactly as they do local ones.
  - Accessing Remote File:

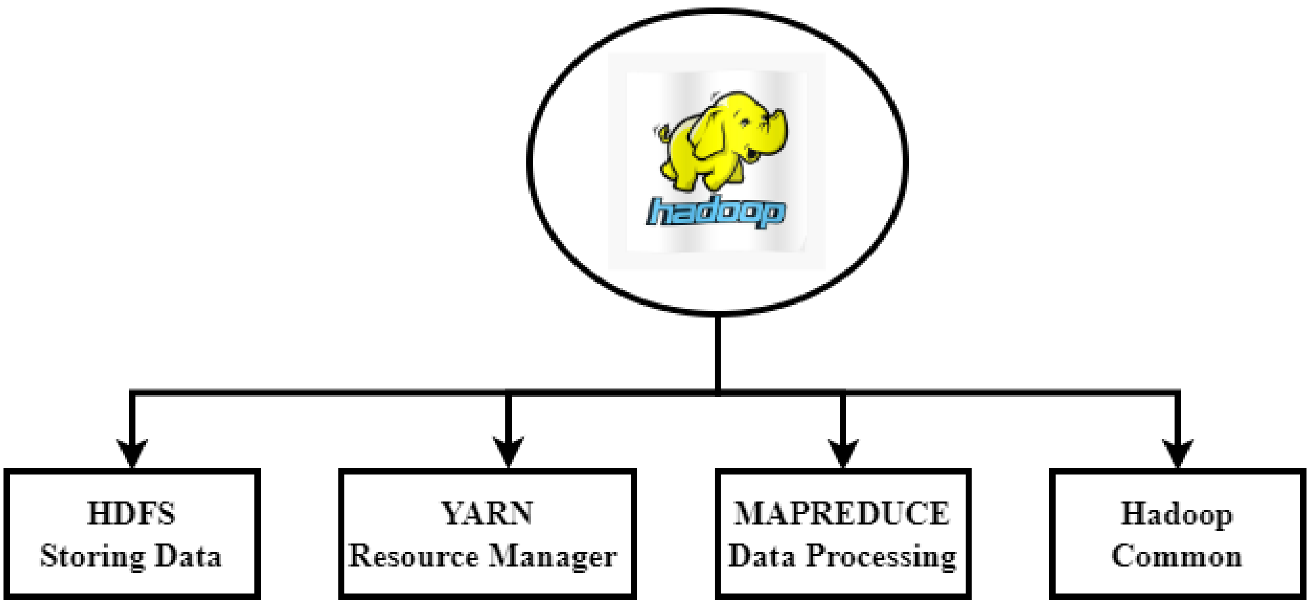
- Reads and writes remote files.
    - Use RPC (Remote Procedure Call) to translate file system calls.



- Hadoop Framework
    - It is a framework that uses cluster computing and a distributed file system to process big data with reasonable cost and time.
    - The Apache™ Hadoop® is a reliable, scalable, distributed computing open-source framework.
    - It uses a set of a master-slave cluster system using a simple programming model.
    - Hadoop Timeline:
        - 2005: Doug Cutting and Michael J. Cafarella developed Hadoop to support distribution for the Nutch search engine project.
        - Hadoop was funded by Yahoo.
        - 2006: Yahoo gave the project to Apache Software Foundation.
        - In 2008, Hadoop wins terabyte sort benchmark (sorted 1 terabyte of data in 209 seconds, compared to previous record of 297 seconds)
    - It uses cluster computing with redundancy.
    - It is designed to **horizontally** scale up from single servers to thousands of machines, each offering local computation and storage.

    - Hadoop Architecture:
        - Hadoop framework architecture uses a master-slave topology.
        - Hadoop Architecture Components:
            - Hadoop Common:
                - The common utilities that support the other Hadoop modules.
            - HDFS (Hadoop Distributed File System)

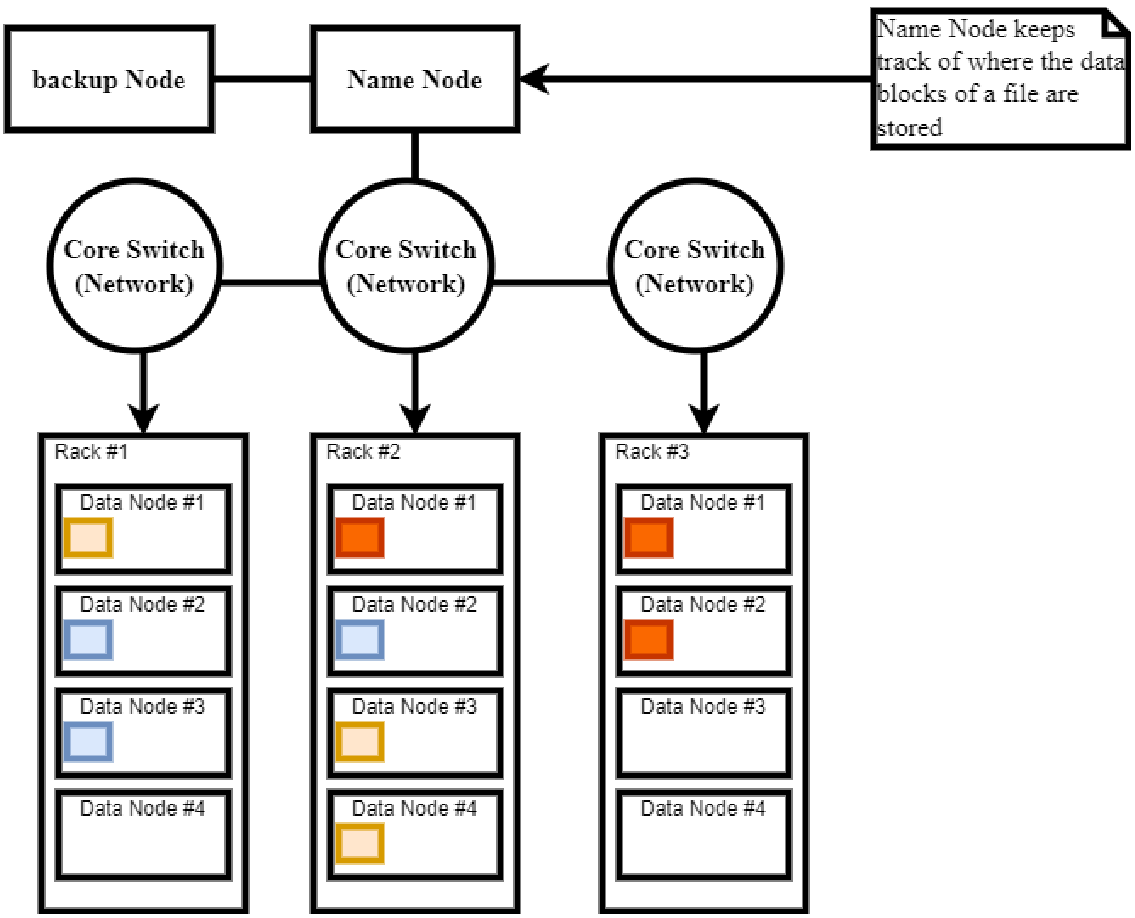
    - A distributed file system that provides high throughput access to application data.
- Hadoop YARN (Yet Another Resource Negotiator):
    - A framework for job scheduling and cluster resource management.
- Hadoop MapReduce:
    - It is one of the main components of processing data in a Hadoop framework.



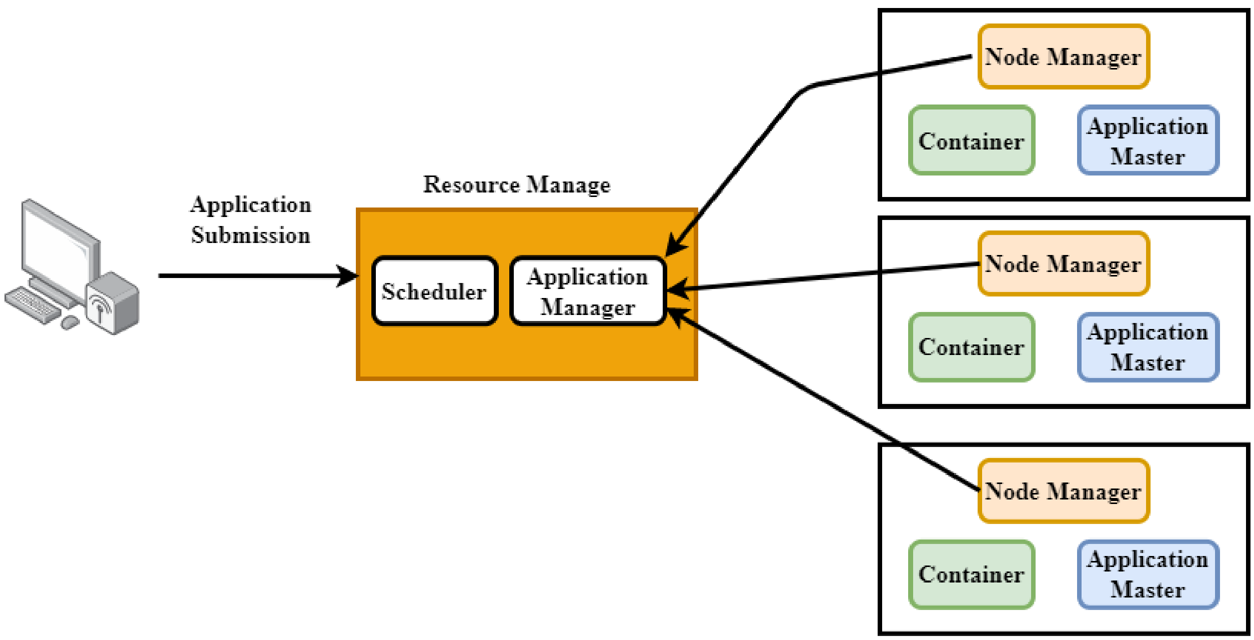
- Hadoop Common Utilities:
    - It is a set of Java libraries and utilities that support all other components in Hadoop cluster.
    - They check Hardware failure in a Hadoop cluster and provide other utilities such as (https://hadoop.apache.org/docs/r3.2.4/):
        - Rack Awareness
        - Service Level Authorization
        - HTTP Authentication
        - Hadoop KMS (It is a cryptographic key management service)
        - Etc.

- Hadoop Distributed File System (HDFS)
    - HDFS is Hadoop distributed file system.
    - The data files are divided into multiple blocks.
    - Data blocks are stored on the cluster slave nodes
    - Data Storage in HDFS:
        - Hadoop HDFFS splits the files into small pieces of data called blocks
        - Block size:
            - Hadoop 1.x: size is 64 MB
            - Hadoop 3.x size is 128 MB
    - HDFS requires two main daemons:
        - NameNode
        - DataNode
    - NameNode:
        - It resides on the master node.
        - It maintains and manages the DataNodes
        - It manages the metadata: locations of data blocks, the size of files, permissions, etc.
        - It monitors heartbeat (A signal sent between a DataNode and NameNode. If there is no signal, then there is something wrong with the DataNode) and block report from all the DataNodes.
    - DataNode:
        - It is a slave daemon that stores actual data.
        - It manages the read/write requests from clients.
    - Rack Awareness Algorithm:
        - It is an algorithm that replicates data blocks in multiple racks in HDFS.
        - It chooses closer data nodes while placing data blocks based on rack information. This information is stored when the Hadoop cluster is created.
        - Placement of replica ensures high reliability and fault tolerance of HDFS

- Replication is done using the following Rack Awareness policy:
    - One copy on one rack i: The closed to the client
    - Two copies on a different rack on different data nodes. The closet rack to rack i to minimize the bandwidth. If we duplicate each block on different racks this would increase the latency of write operations.



- Yarn (Yet Another Resource Negotiator)
    - It is a resource management layer in Hadoop framework.
    - It is responsible for resource allocation and management, job scheduling, etc.
    - Flexibility:
        - The ability to run non-MapReduce applications .
        - It also provides API to develop any generic distribution application such as Hive, Pig, etc.
    - Yarn Framework Components:
        - Yarn has two main components:
            - Resource Manager:
                - It is located on Master computer
            - Node Manager:
                - It is located on each slave machine.



    - Resource Manager:
        - Resource Manager includes two main daemons:
            - Scheduler
            - Applications Manager

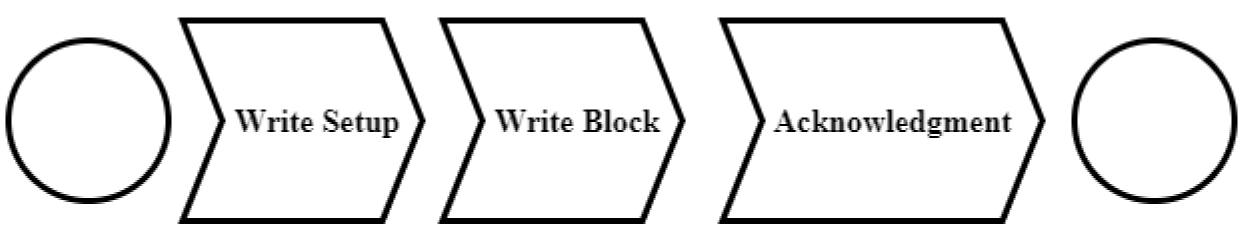
- Scheduler:
    - It is responsible for allocating resources to running applications.
        - There are three types of schedulers available in YARN:
            - FIFO (first in, first out):
                - It is the simplest scheduler and does not need any configuration.
                - It uses a queue to schedule applications.
            - Fair:
                - Fair Scheduler assigns equal amount of resource to all running jobs.
            - Capacity:
                - It maintains a separate queue for small jobs in order to start them as soon a request initiates.
                - Large applications will take more time to complete.
                - It is the default schedule in Hadoop, but it can be changed by setting Yarn.

- Applications Manager
    - It accepts application submissions, negotiating the first container for executing the application specific Application Master.
    - It also provides the service for restarting the Application Master container on failure.

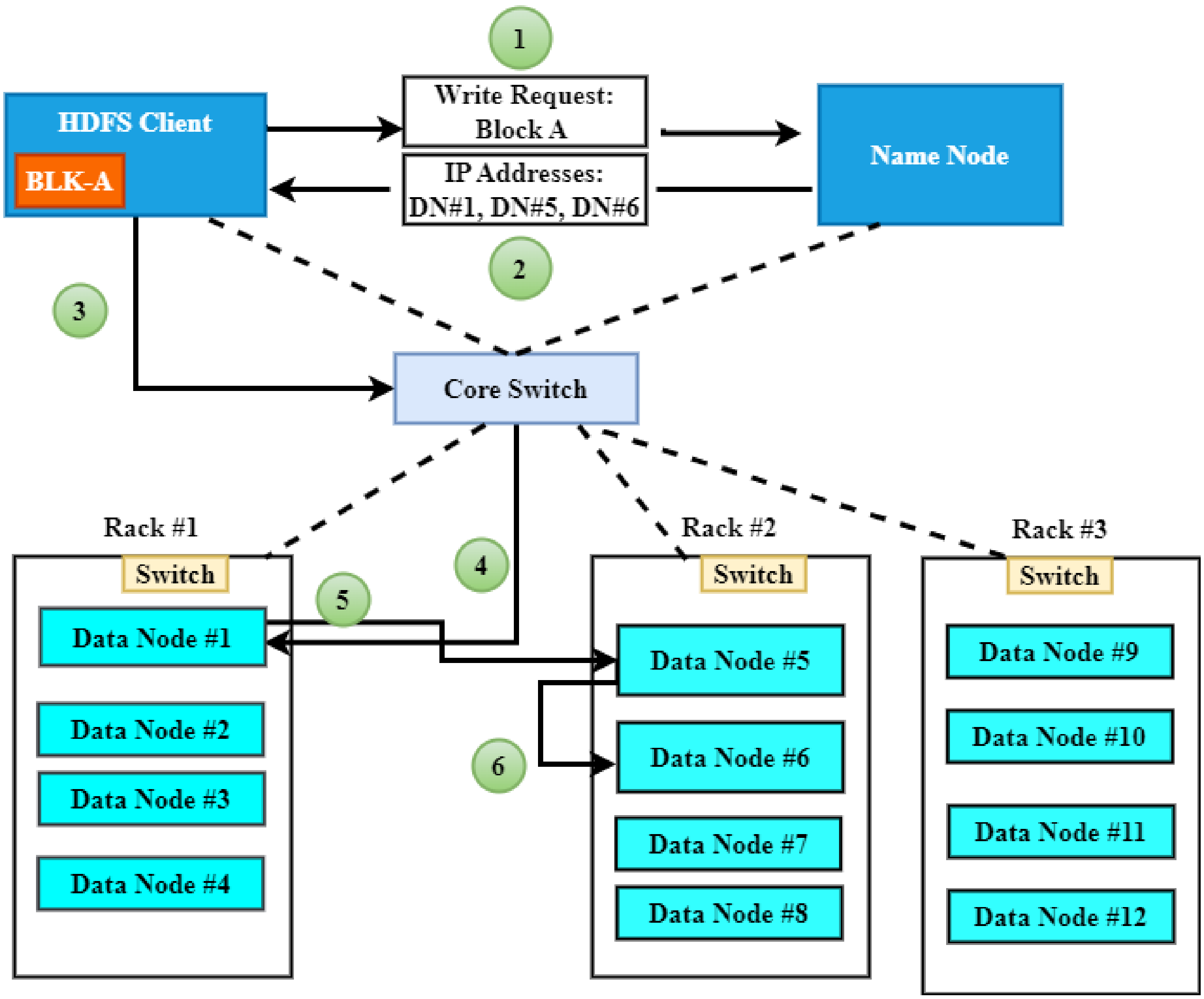
- Node Manager:
    - It is the slave daemon of YARN residing on a commodity hardware - a non-expensive system.

- Node Manager has to monitor the container’s resource usage, along with reporting it to the Resource Manager.
- It keeps the data in the Resource Manager updated.
- It is responsible for launching and managing containers on a slave. It monitors their resource usage such as CPU, memory, etc.
- It can also end the container if requested by the Resource Manager.
- It has two components:
    - Containers:
        - They are created by the node managers to execute the application such as MapReduce’s.
    - Application Master:
        - One per application
        - It is a daemon that negotiates resources with Resource Manager and coordinates the execution of an application in the cluster.

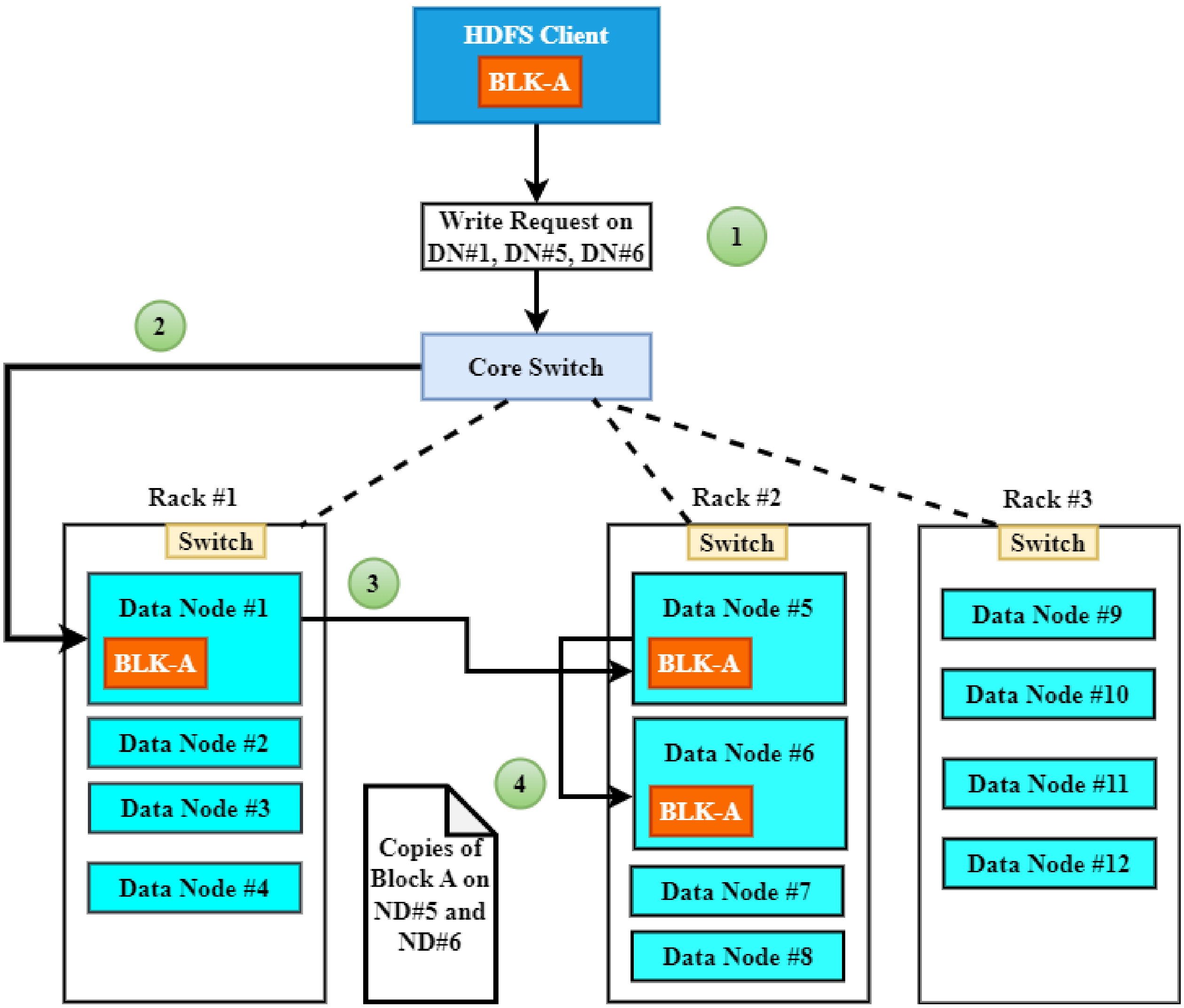
- HDS Read/Write Operations:
    - Write Operation:
        - Write operation is a pipeline that has three steps:
            - Write setup
            - Writing a block
            - Writing confirmation



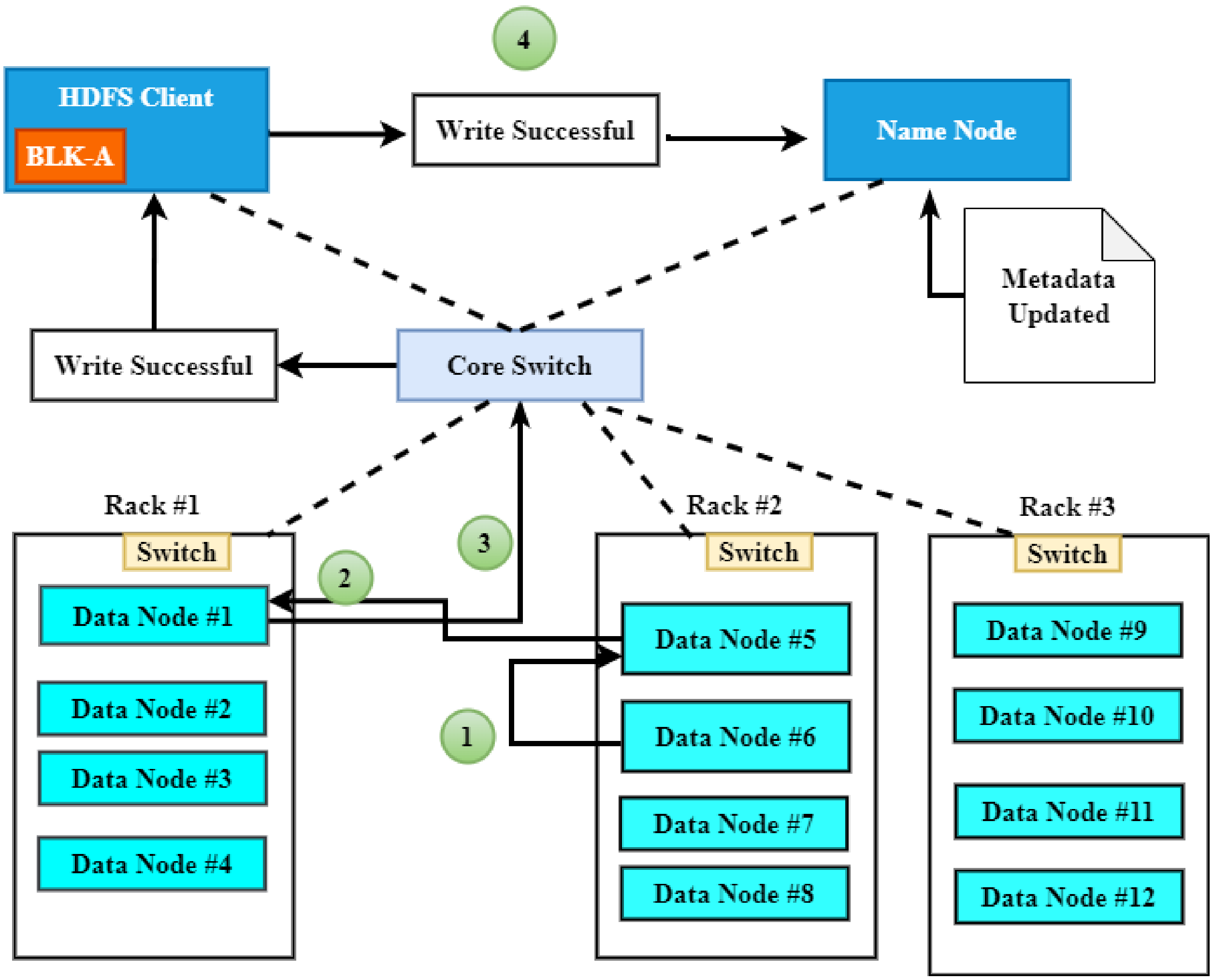
- Writing Setup:



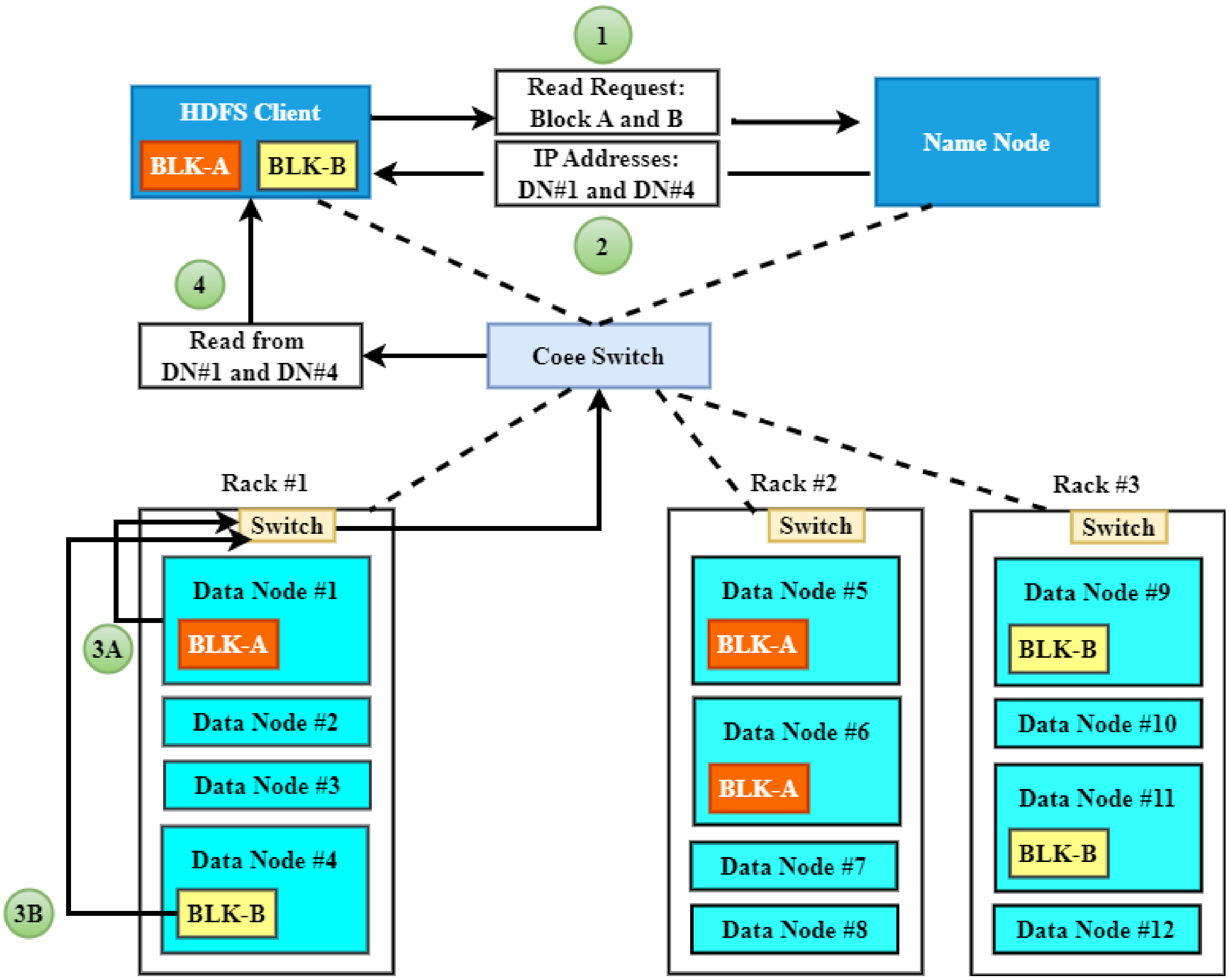
- Writing a block



- Writing acknowledgement

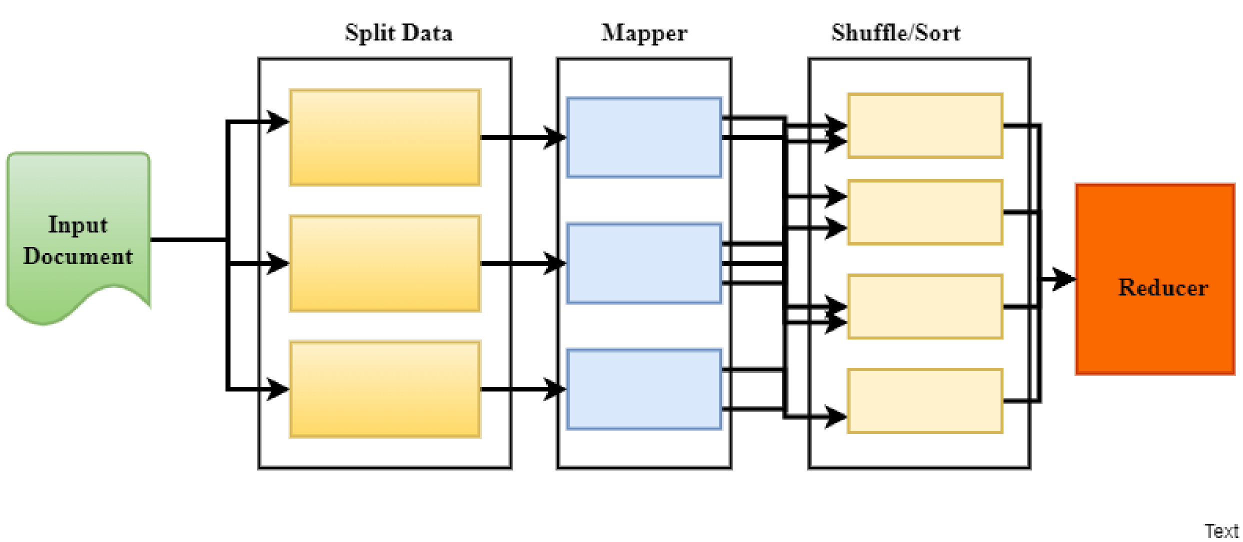


- Read Operation:



- Map Reduce Programming Model:
    - Terms are borrowed from functional programming languages (e.g., Lisp)
        - Sum of squares: (map square '(1 2 3 4)) → Output: (1 4 9 16)
        - (reduce + '(1 4 9 16)) (+ 16 (+ 9 (+ 4 1))) → Output: 30

    - Example: Wordcount:
        - Given a large dataset that cannot fit in main memory.
        - List the count for each word in the dataset:
            - This is one Unix command line if everything fits in memory.
            - For big data, If the total distinct words fit in memory:
                - Use a hash function to map each keyword and keep count.
                - If the data cannot fit in the memory and the total distinct words fits in memory
    - MapReduce is a parallel and distributed programming model used to process big data.
    - The entire MapReduce program can be fundamentally divided into three parts:



        - Mapper:

        - The code to perform the mapping function.
    - Reducer:
        - The code to perform the reducer logic.
    - Driver Code
- Shuffle/Combine/Sort:
    - Shuffle is a build in logic that transfers the map output from Mapper to a Reducer in MapReduce.
    - Data from the mapper are grouped by the key, split among reducers, and sorted by the key.
    - Every reducer obtains all values associated with the same key.

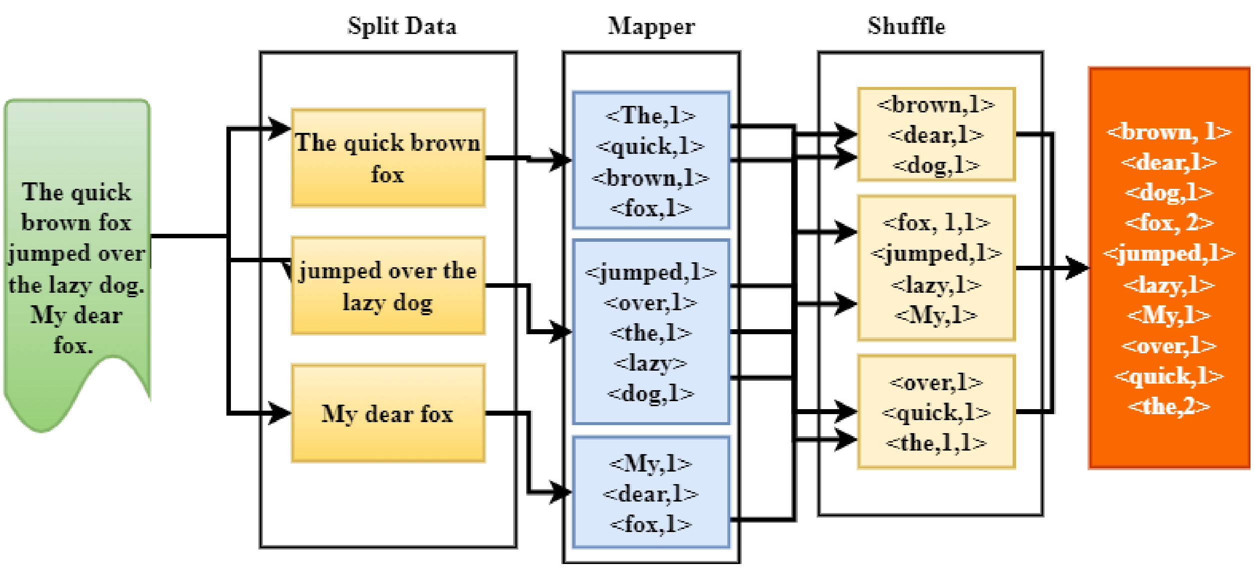
- Driver Code:
    - It is a Driver responsible for setting up a MapReduce Job to run-in Hadoop.
    - It lists the names of the Mapper and Reducer Classes, the job name, input path, output path, etc.
    - Example: gist.github.com

```
package example;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Job;
/ * MapReduce jobs are typically implemented by using a driver class,
 * which sets up the configuration and then submits the job to the
 * Hadoop cluster for execution.  Typical tasks performed in the
 * driver class include configuring the input and output data formats,
 * configuring the map and reduce classes, and specifying the types
 * of intermediate data produced by the job.*/
public class Driver {
  public static void main(String[] args) throws Exception {
     /*
      * To make our program more flexible, we'll allow the input
      * and output directory paths to be specified on the command
      * line instead of hardcoding them. The first thing our driver
      * will do is verify that we were passed these arguments (and
      * ONLY these arguments).
      */
      if (args.length != 2) {
        System.out.printf("Usage: Driver <input dir> <output dir>\n");
        System.exit(-1);
      }
      //Instantiate a Job object for our job's configuration.
        Job = new Job();
      /* Specify the paths to the input and output data based on the
      * command-line arguments.
      */
     FileInputFormat.setInputPaths(job, new Path(args[0]));
     FileOutputFormat.setOutputPath(job, new Path(args[1]));
      /* Specify the JAR (Java archive) file containing your driver, mapper,
      * and reducer.  Hadoop will transfer this JAR file to nodes in your
      * cluster that run the map and reduce tasks. This method instructs
      * Hadoop to find the JAR file based on a specific class it contains.
      */
     job.setJarByClass(Driver.class);
}
```

```
        /*
         * Explicitly setting a descripive name for the job will help us to
         * more easily identify our job in reports and log files, especially
         * on a busy cluster that runs lots of jobs from many users.
         */
        job.setJobName("Employee Salary Analysis Driver");

        /*
         * Tell Hadoop which mapper and reducer classes we'll use for
         * this job.
         */
        job.setMapperClass(EmployeeMapper.class);
        job.setReducerClass(EmployeeReducer.class);

        /*
         * Specify the job's output key and value classes.
         */
         job.setOutputKeyClass(Text.class);
         job.setOutputValueClass(IntWritable.class);

        /*
         * Start the MapReduce job and wait for it to finish.
         * If it finishes successfully, return 0. If not, return 1.
         */
        boolean success = job.waitForCompletion(true);
        System.exit(success ? 0 : 1);
    }
}
```

- Map Computation:
  - Parallelly Process individual records to generate intermediate key/value pairs.

- Reduce Computation:
  - Merge all intermediate values associated per key
- Example: MapReduce Word Count:



- Python Code: riptutorial.com
  - Map Code:

```
import sys
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        print '%s\t%s' % (word, 1)
```

- Reduce Code:

```
import sys
current_word = None
current_count = 0
word = None
for line in sys.stdin:
    # remove leading and trailing whitespaces
    line = line.strip()
    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)
    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue
    if current_word == word:
        current_count += count
    else:
        if current_word:
            print '%s\t%s' % (current_word,
current_count)
        current_count = count
        current_word = word
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

- Other Example:
  - Count of URL access frequency:
    - Input: Log of accessed URLs from a web server

    - Output: For each URL, % of total accesses for that URL

- MapReduce Application Workflow:
    1. A client program submits the application.
    2. Resource Manager allocates a specified container to start the container to start
    3. Application Master, on boot-up, registers with Resource Manager
    4. Application Master negotiates with Resource Manager for appropriate resource containers.
    5. On successful container allocations, Application Master contacts Node Manager to launch the container.
    6. Application code is executed within the container, and then Application Master is responded with the execution status
    7. During execution, the client communicates directly with Application Master or Resource Manager to get status, progress updates etc.
    8. Once the application is complete, Application Master unregisters with Resource Manager and shuts down, allowing its own container process.

- Hadoop-related Apache Projects:
    - Ambari™: A web-based tool for provisioning, managing, and monitoring Hadoop clusters. It also provides a dashboard for viewing cluster health and ability to view MapReduce, Pig and Hive applications visually.
    - Avro™: A data serialization system.
    - Cassandra™: A scalable multi-master database with no single points of failure.
    - Chukwa™: A data collection system for managing large distributed systems.
    - HBase™: A scalable, distributed database that supports structured data storage for large tables.
    - Hive™: A data warehouse infrastructure that provides data summarization and ad hoc querying.
    - Mahout™: A Scalable machine learning and data mining library.
    - Pig™: A high-level data-flow language and execution framework for parallel computation.
    - Spark™: A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL, machine learning, stream processing, and graph computation.
    - Tez™: A generalized data-flow programming framework, built on Hadoop YARN, which provides a powerful and flexible engine to execute an arbitrary DAG of tasks to process data for both batch and interactive use-cases.
    - ZooKeeper™: A high-performance coordination service for distributed applications.