



Lecturer: Narges Peyravi

What is Hadoop?

- Hadoop is an open-source framework that allows to store and process big data in a distributed environment across clusters of computers using simple programming models.
- Hadoop is written in Java.
- It is used for batch/offline processing.
- It is being used by Facebook, Yahoo, Google, Twitter, LinkedIn and many more.
- it can be scaled up just by adding nodes in the cluster.
- Applications built using HADOOP are run on large data sets distributed across clusters of commodity computers. Commodity computers are cheap and widely available. These are mainly useful for achieving greater computational power at low cost.

History of Hadoop

- In 2002, Doug Cutting and Mike Cafarella started to work on a project, **Apache Nutch**. It is an open source web crawler software project.
- While working on Apache Nutch, they were dealing with big data. To store that data they have to spend a lot of costs which becomes the consequence of that project. This problem becomes one of the important reason for the emergence of Hadoop.
- In 2003, Google introduced a file system known as GFS (Google file system). It is a proprietary distributed file system developed to provide efficient access to data.
- In 2004, Google released a white paper on Map Reduce. This technique simplifies the data processing on large clusters.
- In 2005, Doug Cutting and Mike Cafarella introduced a new file system known as NDFFS (Nutch Distributed File System). This file system also includes Map reduce.

History of Hadoop

- In 2006, Doug Cutting quit Google and joined Yahoo. On the basis of the Nutch project, Doug Cutting introduces a new project Hadoop with a file system known as HDFS (Hadoop Distributed File System). Hadoop first version 0.1.0 released in this year.
- Doug Cutting gave named his project Hadoop after his son's toy elephant.
- In 2007, Yahoo runs two clusters of 1000 machines.
- In 2008, Hadoop became the fastest system to sort 1 terabyte of data on a 900 node cluster within 209 seconds.
- In 2013, Hadoop 2.2 was released.
- In 2017, Hadoop 3.0 was released.

Components of Hadoop

- Hadoop has three components:
 - **HDFS** (Hadoop Distributed File System): a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster
 - **MapReduce**: an implementation of the MapReduce programming model for large-scale data processing.
 - **Yarn**: (introduced in 2012) a platform responsible for managing computing resources in clusters and using them for scheduling users' applications

Features of Hadoop

Reliability: When machines are working as a single unit, if one of the machines fails, another machine will take over the responsibility and work in a **reliable** and **fault-tolerant** fashion. Hadoop infrastructure has inbuilt fault tolerance features and hence, Hadoop is highly reliable.

Scalability : Hadoop has the inbuilt capability of integrating seamlessly with cloud-based services. So, if you are installing Hadoop on a cloud, you don't need to worry about the scalability factor because you can go ahead and procure more hardware and expand your set up within minutes whenever required.

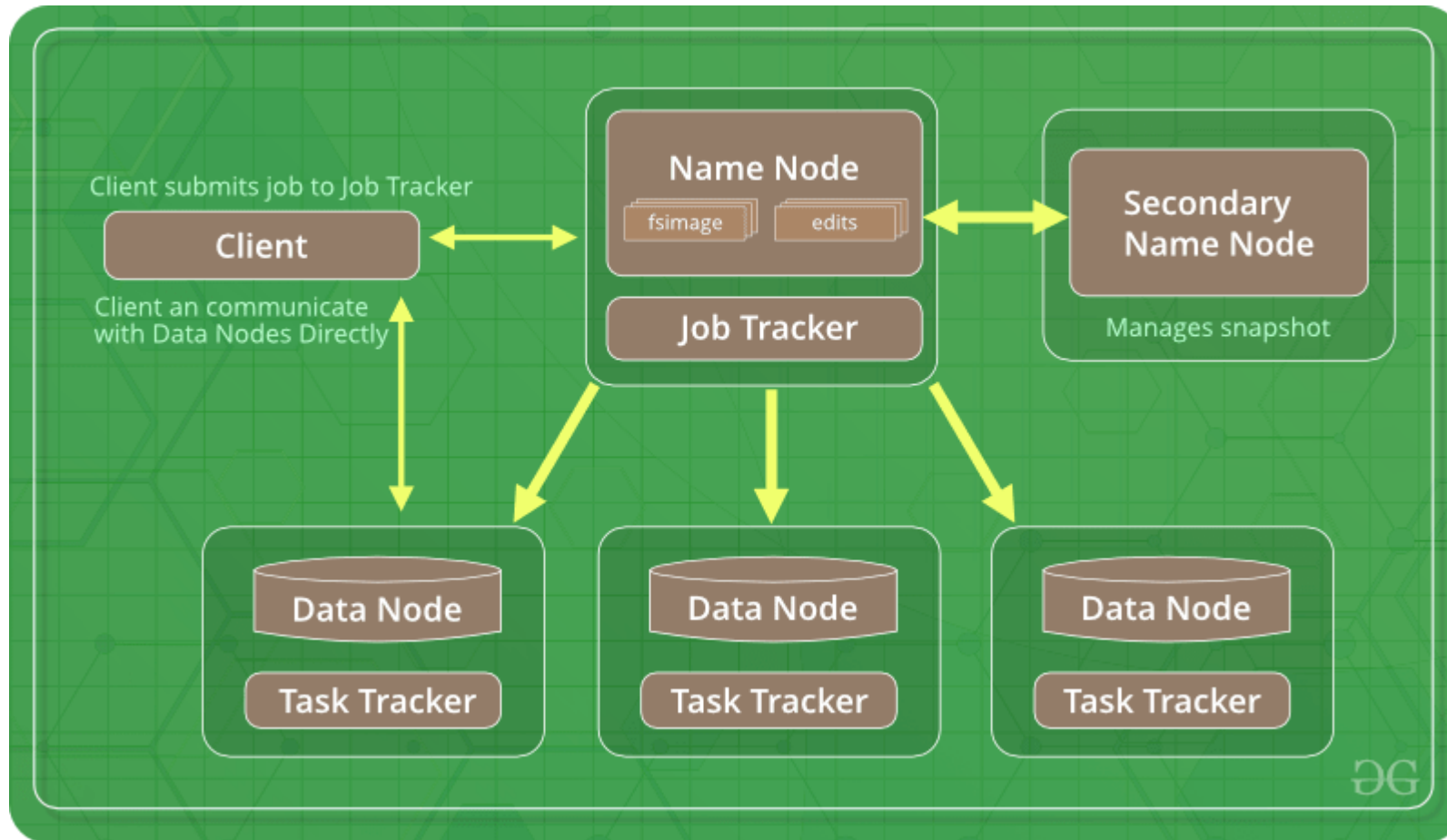


Features of Hadoop

- **Economical:** Hadoop uses commodity hardware (like your PC, laptop). For example, in a small Hadoop cluster, all your DataNodes can have normal configurations like 8-16 GB RAM with 5-10 TB hard disk and Xeon processors.
- But if I would have used hardware-based RAID with Oracle for the same purpose, I would end up spending 5x times more at least. So, the cost of ownership of a Hadoop-based project is minimized. It is easier to maintain a Hadoop environment and is economical as well. Also, Hadoop is open-source software and hence there is no licensing cost.
- **Flexibility :** Hadoop is very flexible in terms of the ability to deal with all kinds of data, where data can be of any kind and Hadoop can store and process them all.

Hadoop Versions

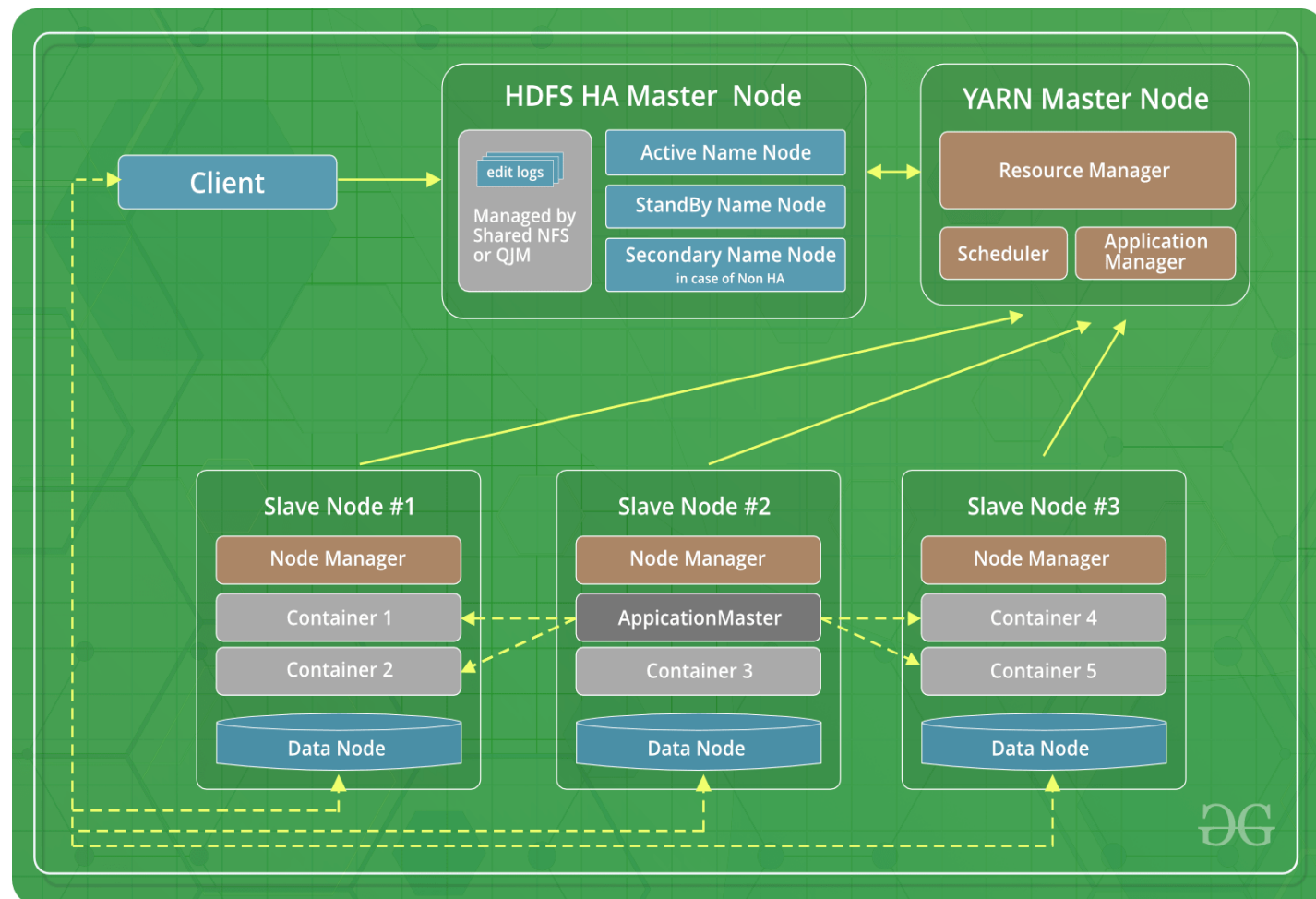
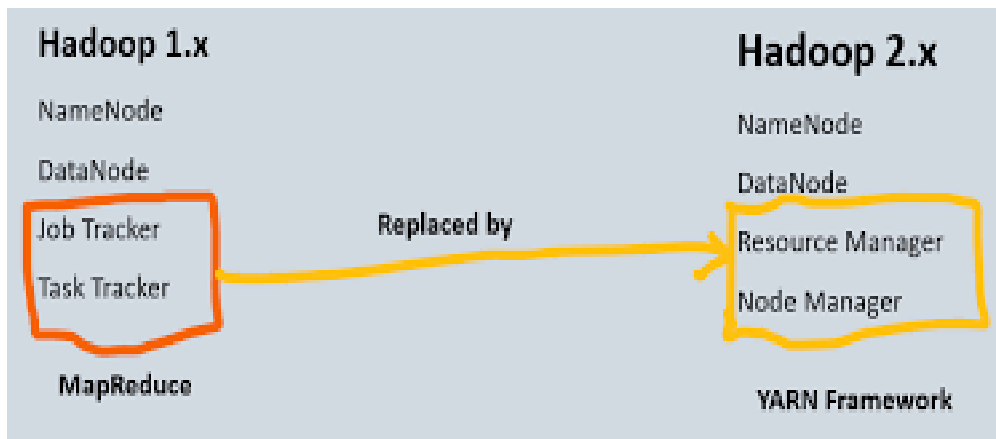
- **Hadoop 1:** This is the first and most basic version of Hadoop. It includes Hadoop Common, Hadoop Distributed File System (HDFS), and Map Reduce.



Hadoop Versions


- **Hadoop 2:** The only difference between Hadoop 1 and Hadoop 2 is that Hadoop 2 additionally contains YARN (Yet Another Resource Negotiator). YARN helps in resource management and task scheduling through its two daemons namely job tracking and progress monitoring.

In computing terms, Daemons is a process that runs in the background. Hadoop has five such daemons, namely NameNode, Secondary NameNode, DataNode, JobTracker, and TaskTracker. Each daemons runs separately in its own JVM.



Hadoop Versions

- **Hadoop 3:** This is the recent version of Hadoop. Along with the merits of the first two versions, Hadoop 3 has one most important merit. It has resolved the issue of single point failure by having multiple name nodes. Various other advantages like erasure coding, use of GPU hardware and Dockers makes it superior to the earlier versions of Hadoop.

Features	Hadoop 2.x	Hadoop 3.x 
Min Java Version Required	Java 7	Java 8
Fault Tolerance	Via replication	Via erasure coding
Storage Scheme	3x replication factor for data reliability, 200% overhead	Erasure coding for data reliability, 50% overhead
Yarn Timeline Service	Scalability issues	Highly scalable and reliable
Standby NN	Supports only 1 SBNN	Supports only 2 or more SBNN
Heap Management	We need to configure HADOOP_HEAPSIZE	Provides auto-tuning of heap

Hadoop Architecture



Hadoop Architecture

Application Layer



Other
Applications

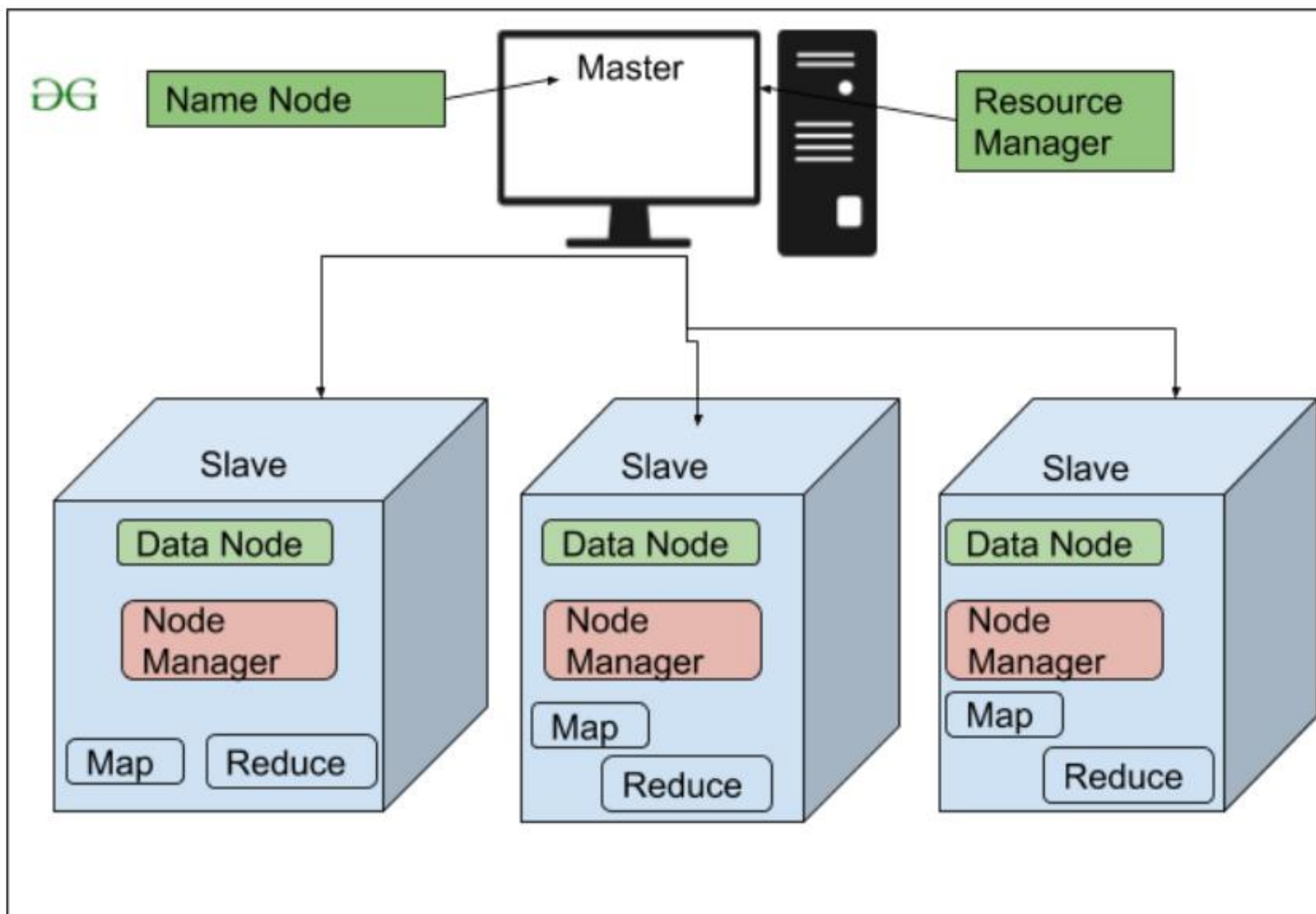
Resource Management
Layer



Storage Layer



Hadoop Architecture



HDFS

- Hadoop Distributed File System is a dedicated file system to store big data with a cluster of commodity hardware or cheaper hardware with streaming access pattern.
- It enables data to be stored at multiple nodes in the cluster which ensures data security and fault tolerance.
- It contains a master/slave architecture.
- It states that the files will be broken into blocks and stored in nodes over the distributed architecture.
- HDFS is based on the Google File System (GFS) and provides a distributed file system.

The goals of HDFS

- **Fast recovery from hardware failures:** Because one HDFS instance may consist of thousands of servers, failure of at least one server is inevitable. HDFS has been built to detect faults and automatically recover quickly.
- **Access to streaming data:** HDFS is intended more for batch processing versus interactive use, so the emphasis in the design is for high data throughput rates, which accommodate streaming access to data sets.
- **Accommodation of large data sets:** HDFS accommodates applications that have data sets typically gigabytes to terabytes in size. HDFS provides high aggregate data bandwidth and can scale to hundreds of nodes in a single cluster.
- **Portability:** To facilitate adoption, HDFS is designed to be portable across multiple hardware platforms and to be compatible with a variety of underlying operating systems.

Where to use and not to use HDFS

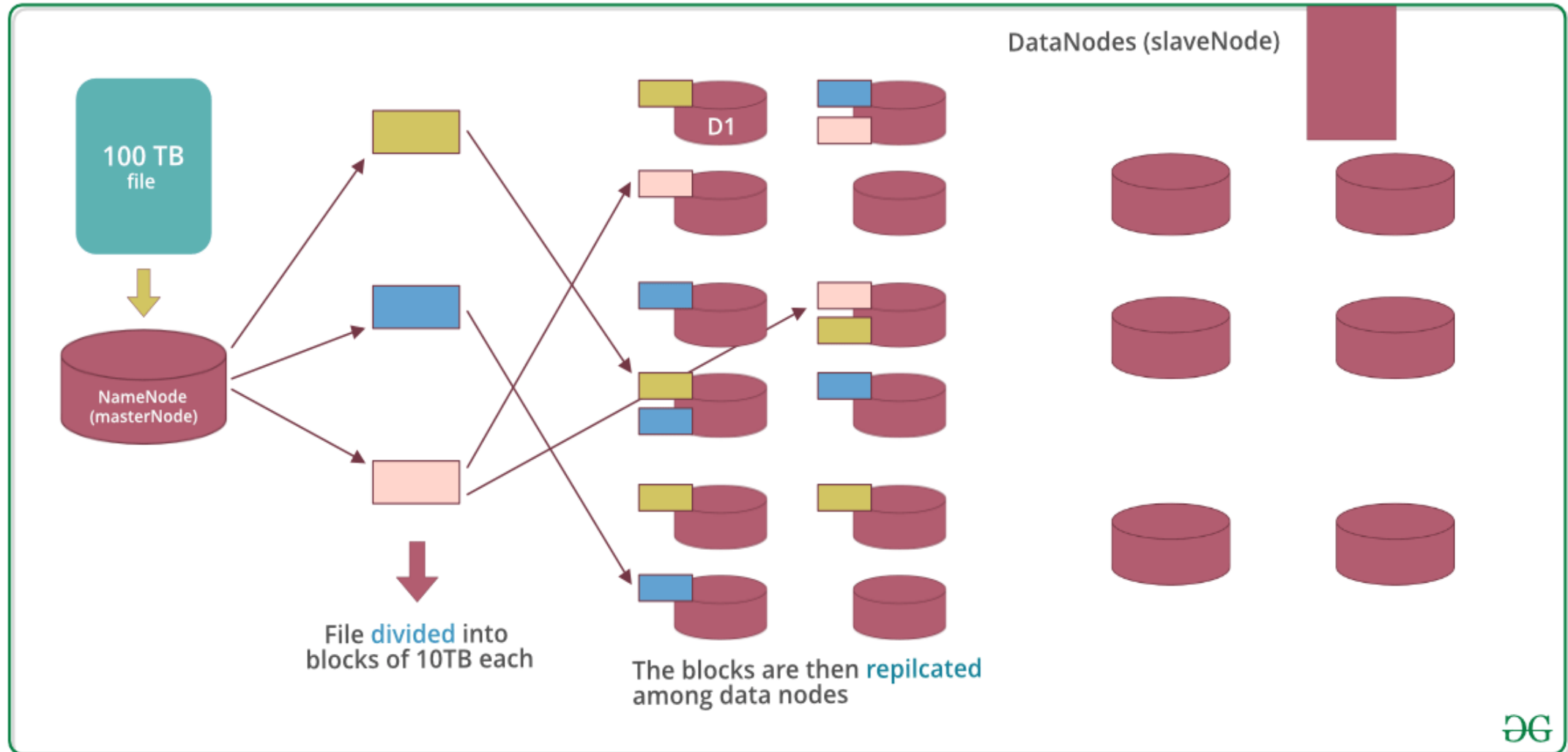
Where to use HDFS:

- **Very Large Files:** Files should be of hundreds of megabytes, gigabytes or more.
- **Streaming Data Access:** The time to read whole data set is more important than latency in reading the first. HDFS is built on write-once and read-many-times pattern.
- **Commodity Hardware:** It works on low cost hardware.

Where not to use HDFS:

- **Low Latency data access:** Applications that require very less time to access the first data should not use HDFS as it is giving importance to whole data rather than time to fetch the first record.
- **Lots Of Small Files:** The name node contains the metadata of files in memory and if the files are small in size it takes a lot of memory for name node's memory which is not feasible.
- **Multiple Writes:** It should not be used when we have to write multiple times.

Data storage in HDFS



HDFS Concepts

- **Blocks:**

- ❖ A Block is the minimum amount of data that it can read or write.
- ❖ HDFS blocks are 128 MB by default and this is configurable.
- ❖ Files in HDFS are broken into block-sized chunks, which are stored as independent units.
- ❖ Unlike a file system, if the file in HDFS is smaller than block size, then it does not occupy full block's size, i.e. 5 MB of file stored in HDFS of block size 128 MB takes 5MB of space only.
- ❖ The HDFS block size is large just to minimize the cost of seek.

HDFS Concepts

- **MasterNode:**

- ❖ Manages all the slave nodes and assign work to them.
- ❖ It executes filesystem namespace operations like opening, closing, renaming files and directories.
- ❖ It should be deployed on reliable hardware which has the high config. not on commodity hardware.
- ❖ MasterNode has the record of everything, it knows the location and info of each and every single data nodes and the blocks they contain, i.e. nothing is done without the permission of masternode.

HDFS Concepts

- **Name Node:**

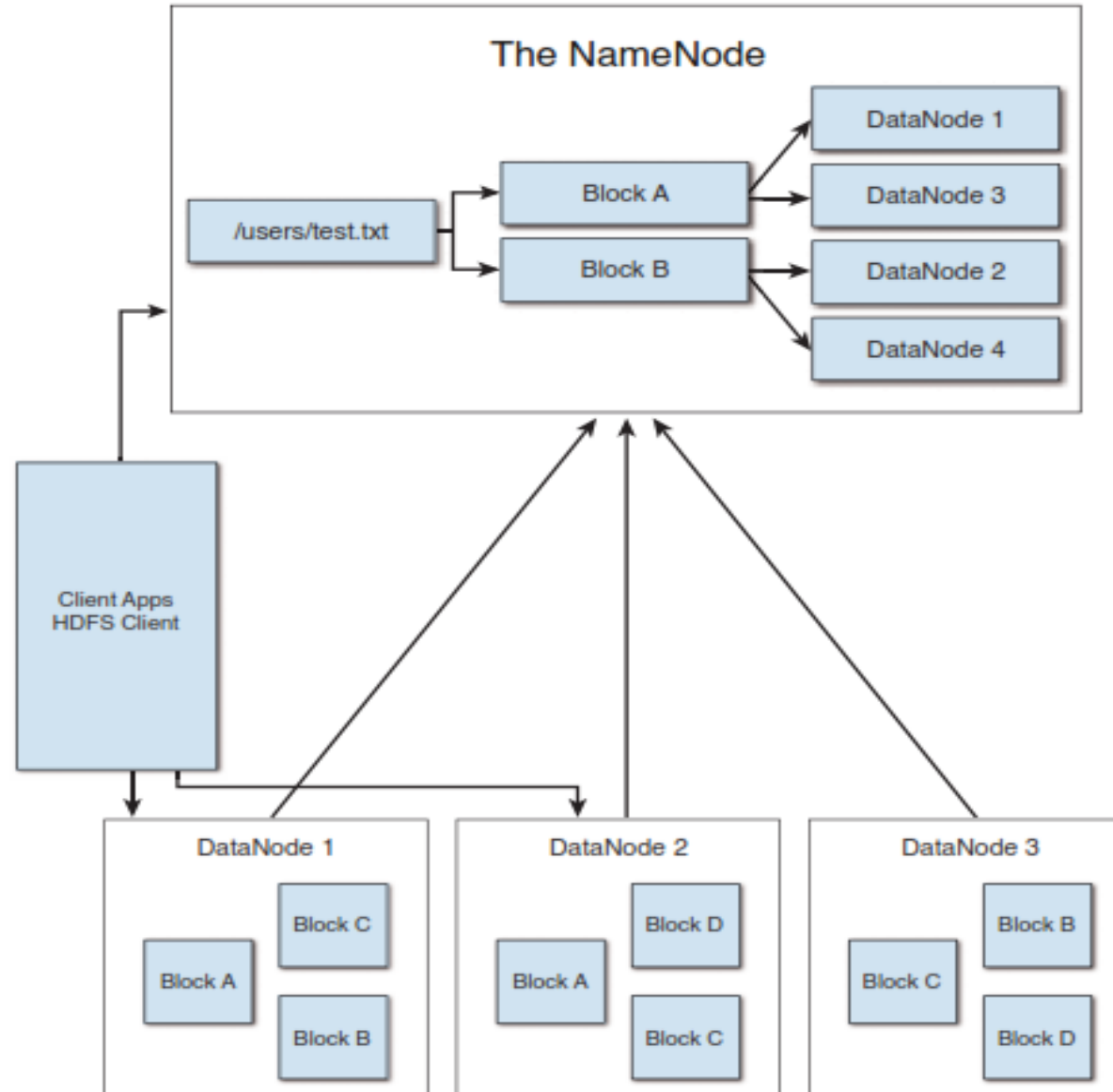
- ❖ HDFS works in master-worker pattern where the name node acts as master.
- ❖ Name Node is controller and manager of HDFS as it knows the status and the metadata of all the files in HDFS; the metadata information being file permission, names and location of each block.
- ❖ The metadata are small, so it is stored in the memory of name node, allowing faster access to data. Moreover the HDFS cluster is accessed by multiple clients concurrently, so all this information is handled by a single machine. The file system operations like opening, closing, renaming etc. are executed by it.

HDFS Concepts

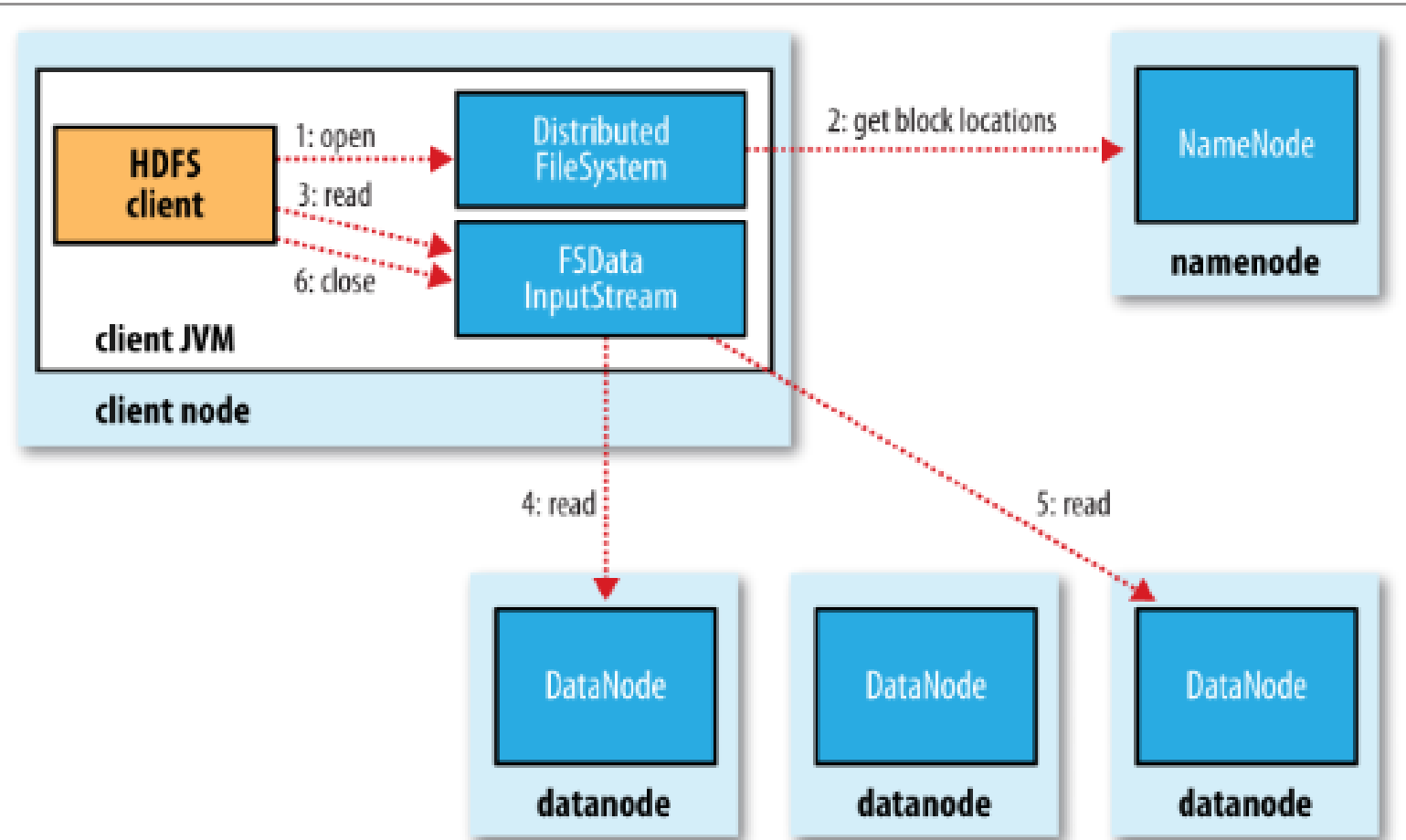
- **Data Node:**

- ❖ They store and retrieve blocks when they are told to; by client or name node.
- ❖ They report back to name node periodically, with list of blocks that they are storing.
- ❖ The data node being a commodity hardware also does the work of block creation, deletion and replication as stated by the name node.
- ❖ Run on slave nodes.
- ❖ Require high memory as data is actually stored here.

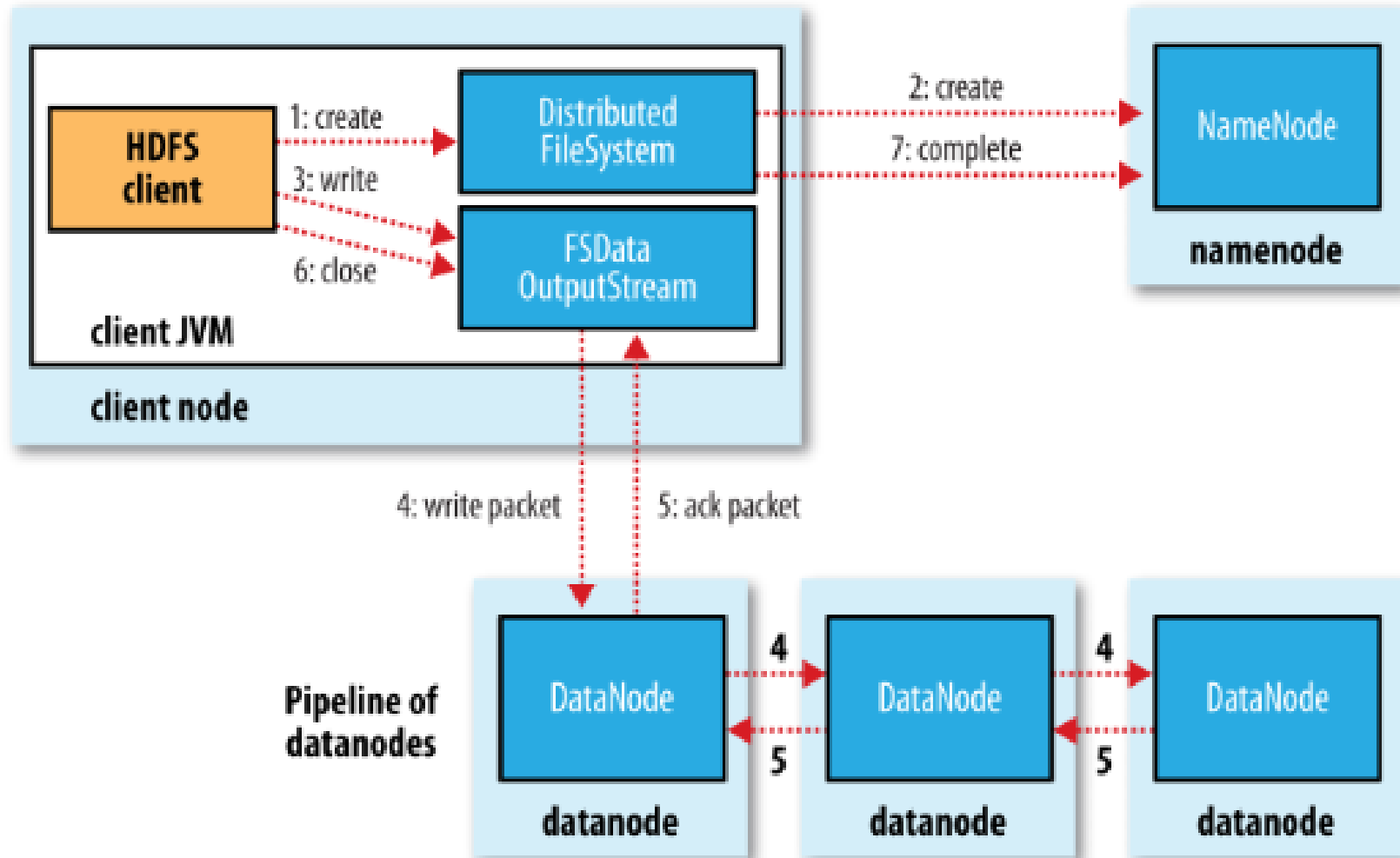
How HDFS clients communicate with the NameNode and DataNodes



How a client reading data from HDFS



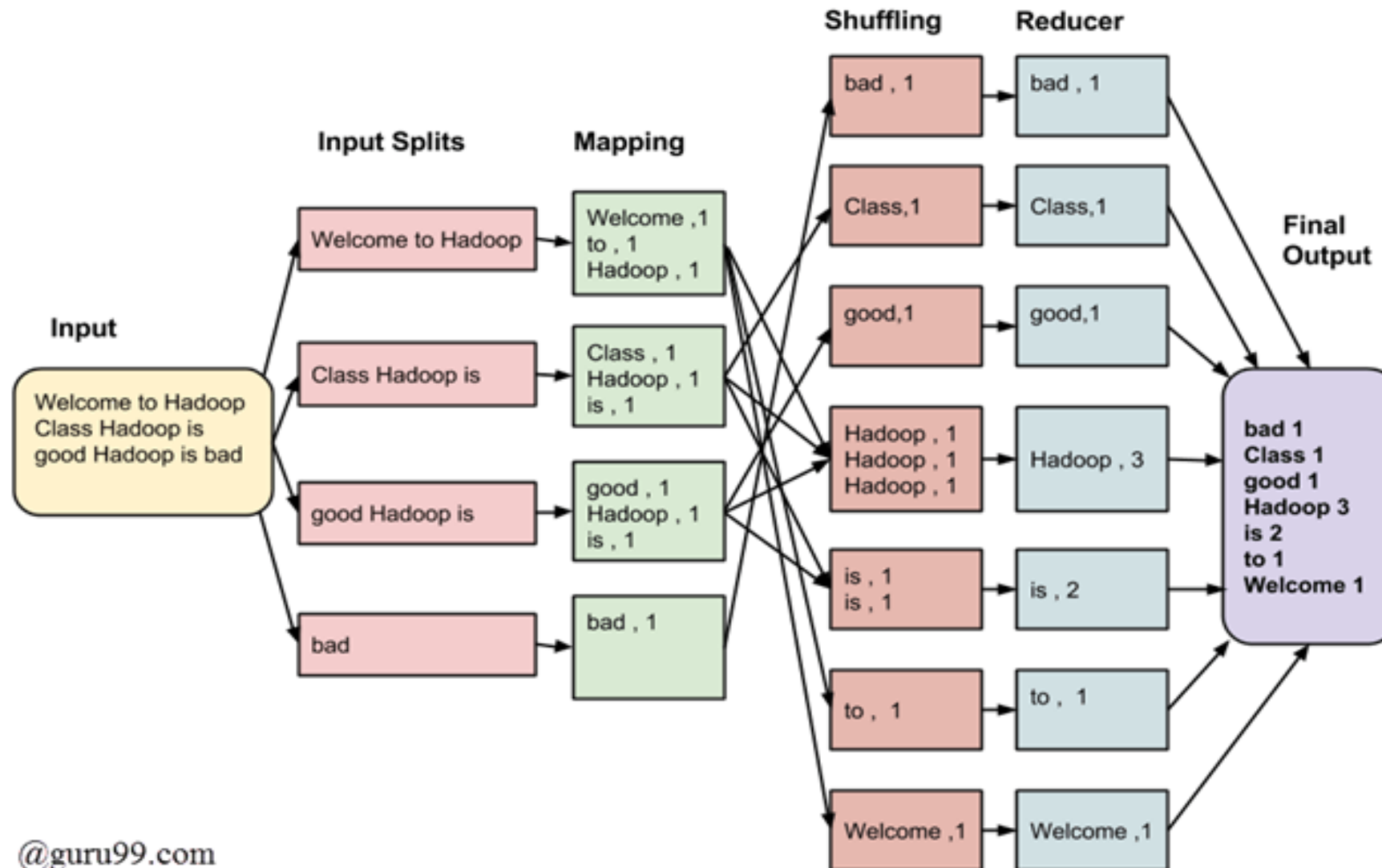
How a client writing data to HDFS



Map Reduce

- MapReduce is a data processing tool which is used to process the data parallelly in a distributed form.
- MapReduce works by breaking the processing into two phases: the map phase and the reduce phase.
- Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer.
- The programmer also specifies two functions: the map function and the reduce function.

How MapReduce works? (Example Word Count)



The MapReduce Model

A MapReduce job consists of two steps: Map and Reduce. Here's what the two processes do:

- **Map:** This step processes the original input file in a parallel fashion and transforms it into an intermediate output.
- **Reduce:** This summarization step processes all relevant records together.

The application developer provides the following four classes for enabling the MapReduce programming model:

- A class to read the input file and transform the input records into a key/value pair per record.
- A mapper class.
- A reducer class.
- A class to transform the key/value pair generated by the reducer class into the final output records.

MapReduce Classes

MapReduce majorly has the following three Classes.

They are:

- **Mapper Class:** The first stage in Data Processing using MapReduce is the **Mapper Class**. RecordReader processes each Input record and generates the respective key-value pair. Hadoop's Mapper store saves this intermediate data into the local disk.
- **Input Split:** It is the logical representation of data. It represents a block of work that contains a single map task in the MapReduce Program.
- **RecordReader:** It interacts with the Input split and converts the obtained data in the form of **Key-Value Pairs**.

MapReduce Classes

- **Reducer Class**: The Intermediate output generated from the mapper is fed to the reducer which processes it and generates the final output which is then saved in the **HDFS**.
- **Driver Class** : The major component in a MapReduce job is a **Driver Class**. It is responsible for setting up a MapReduce Job to run-in Hadoop. We specify the names of **Mapper** and **Reducer** Classes long with data types and their respective job names.



Phases of MapReduce

The MapReduce program is executed in three main phases: mapping, shuffling, and reducing. There is also an optional phase known as the combiner phase.

- **Mapping Phase:** This is the first phase of the program. There are two steps in this phase: **splitting and mapping**. A dataset is split into equal units called *chunks (input splits)* in the splitting step. Hadoop consists of a RecordReader that uses TextInputFormat to transform input splits into key-value pairs.
- The key-value pairs are then used as inputs in the mapping step. This is the only data format that a mapper can read or understand. The mapping step contains a coding logic that is applied to these data blocks. In this step, the mapper processes the key-value pairs and produces an output of the same form (key-value pairs).

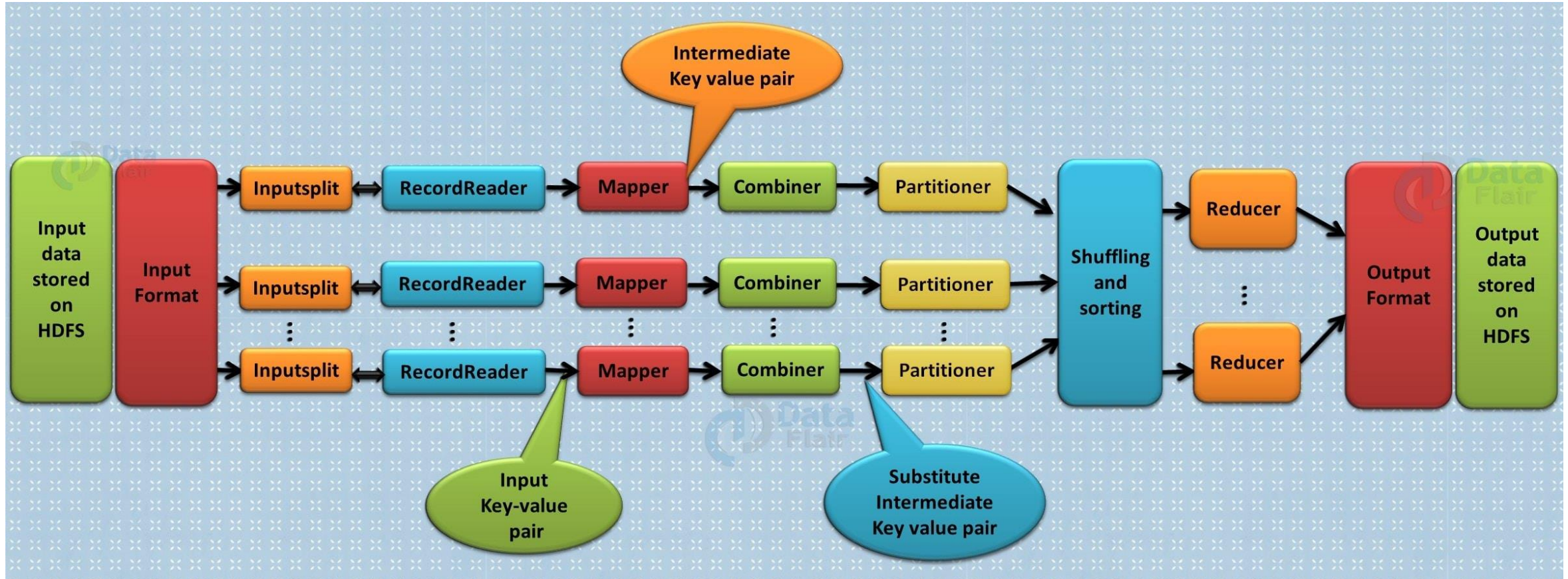
Phases of MapReduce

- **Shuffling phase**: This is the second phase that takes place after the completion of the Mapping phase. It consists of two main steps: **sorting and merging**.
- In the sorting step, the key-value pairs are sorted using the keys. Merging ensures that key-value pairs are combined.
- The shuffling phase facilitates the removal of duplicate values and the grouping of values. Different values with similar keys are grouped. The output of this phase will be keys and values, just like in the Mapping phase.

Phases of MapReduce

- **Reducer phase:** In the reducer phase, the output of the shuffling phase is used as the input. The reducer processes this input further to reduce the intermediate values into smaller values.
- It provides a summary of the entire dataset. The output from this phase is stored in the HDFS.
- **Combiner phase:** This is an optional phase that's used for optimizing the MapReduce process. It's used for reducing the outputs at the node level. In this phase, duplicate outputs from the map outputs can be combined into a single output. The combiner phase increases speed in the Shuffling phase by improving the performance of Jobs.

Phases of MapReduce



MapReduce Flow Chart

- 1. Input Files** : The data for a MapReduce task is stored in **input files**, and input files typically lives in **HDFS**. The format of these files is arbitrary, while line-based log files and binary format can also be used.
- 2. InputFormat**: **InputFormat** defines how these input files are split and read. It selects the files or other objects that are used for input. InputFormat creates InputSplit.
- 3. InputSplits**: It is created by InputFormat, logically represent the data which will be processed by an individual **Mapper** (We will understand mapper below). One map task is created for each split; thus the number of map tasks will be equal to the number of InputSplits. The split is divided into records and each record will be processed by the mapper.

MapReduce Flow Chart

4. RecordReader: It communicates with the **InputSplit** in Hadoop MapReduce and converts the data into key-value pairs suitable for reading by the mapper. By default, it uses TextInputFormat for converting data into a key-value pair. RecordReader communicates with the InputSplit until the file reading is not completed. It assigns byte offset (unique number) to each line present in the file. Further, these key-value pairs are sent to the mapper for further processing.

5. Mapper : It processes each input record (from RecordReader) and generates new key-value pair, and this key-value pair generated by Mapper is completely different from the input pair. The output of Mapper is also known as intermediate output which is written to the local disk. The output of the Mapper is not stored on HDFS as this is temporary data and writing on HDFS will create unnecessary copies. Mappers output is passed to the combiner for further process.

MapReduce Flow Chart

6. Combiner: The combiner is also known as 'Mini-reducer'. Hadoop MapReduce Combiner performs local aggregation on the mappers' output, which helps to minimize the data transfer between mapper and **reducer** . Once the combiner functionality is executed, the output is then passed to the partitioner for further work.

7. Partitioner: Hadoop MapReduce, **Partitioner** comes into the flow if we are working on more than one reducer (for one reducer partitioner is not used).

Partitioner takes the output from combiners and performs partitioning. Partitioning of output takes place on the basis of the key and then sorted. By hash function, key (or a subset of the key) is used to derive the partition.

According to the key value in MapReduce, each combiner output is partitioned, and a record having the same key value goes into the same partition, and then each partition is sent to a reducer. Partitioning allows even distribution of the map output over the reducer

MapReduce Flow Chart

8. Shuffling and Sorting: the output is Shuffled to the reduce node . The shuffling is the physical movement of the data which is done over the network. Once all the mappers are finished and their output is shuffled on the reducer nodes, then this intermediate output is merged and sorted, which is then provided as input to reduce phase.

9. Reducer: It takes the set of intermediate key-value pairs produced by the mappers as the input and then runs a reducer function on each of them to generate the output. The output of the reducer is the final output, which is stored in HDFS.

10. RecordWriter: It writes these output key-value pair from the Reducer phase to the output files.

11. OutputFormat: The way these output key-value pairs are written in output files by RecordWriter is determined by the OutputFormat. OutputFormat instances provided by the Hadoop are used to write files in HDFS or on the local disk. Thus the final output of reducer is written on HDFS by OutputFormat instances.

YARN(*Yet Another Resource Negotiator*)

- Apache YARN is a resource management layer in Hadoop.
- YARN is a large-scale, distributed operating system for big data applications.
- It was introduced in Hadoop 2.0 to remove the bottleneck on Job Tracker which was present in Hadoop 1.0.
- YARN was described as a “*Redesigned Resource Manager*” at the time of its launching, but it has now evolved to be known as large-scale distributed operating system used for Big Data processing.
- YARN is a software rewrite that is capable of decoupling MapReduce’s resource management and scheduling capabilities from the data processing component.

The main components of YARN

- **Client**: It submits map-reduce jobs.
- **Resource Manager**: It is the master daemon of YARN and is responsible for resource assignment and management among all the applications. Whenever it receives a processing request, it forwards it to the corresponding node manager and allocates resources for the completion of the request accordingly. It has two major components:
 - **Scheduler**: It performs scheduling based on the allocated application and available resources. It is a pure scheduler, means it does not perform other tasks such as monitoring or tracking and does not guarantee a restart if a task fails. The YARN scheduler supports plugins such as Capacity Scheduler and Fair Scheduler to partition the cluster resources.
 - **Application manager**: It is responsible for accepting the application and negotiating the first container from the resource manager. It also restarts the Application Manager container if a task fails.

The main components of YARN

- **Node Manager:** It take care of individual node on Hadoop cluster and manages application and workflow and that particular node. Its primary job is to keep-up with the Node Manager. It monitors resource usage, performs log management and also kills a container based on directions from the resource manager. It is also responsible for creating the container process and start it on the request of Application master.
- **Node Manager is the slave daemon of YARN. It has the following responsibilities:**
 - Node Manager has to monitor the container's resource usage, along with reporting it to the Resource Manager.
 - The health of the node on which YARN is running is tracked by the Node Manager.
 - It takes care of each node in the cluster while managing the workflow, along with user jobs on a particular node.
 - It keeps the data in the Resource Manager updated
 - Node Manager can also destroy or kill the container if it gets an order from the Resource Manager to do so.

The main components of YARN

- **Application Master:** An application is a single job submitted to a framework. The application manager is responsible for negotiating resources with the resource manager, tracking the status and monitoring progress of a single application. The application master requests the container from the node manager by sending a Container Launch Context(CLC) which includes everything an application needs to run. Once the application is started, it sends the health report to the resource manager from time-to-time.
- Every job submitted to the framework is an application, and every application has a specific Application Master associated with it. Application Master performs the following tasks:
 - It coordinates the execution of the application in the cluster, along with managing the faults.
 - It negotiates resources from the Resource Manager.
 - It works with the Node Manager for executing and monitoring other components' tasks.
 - At regular intervals, heartbeats are sent to the Resource Manager for checking its health, along with updating records according to its resource demands.

The main components of YARN

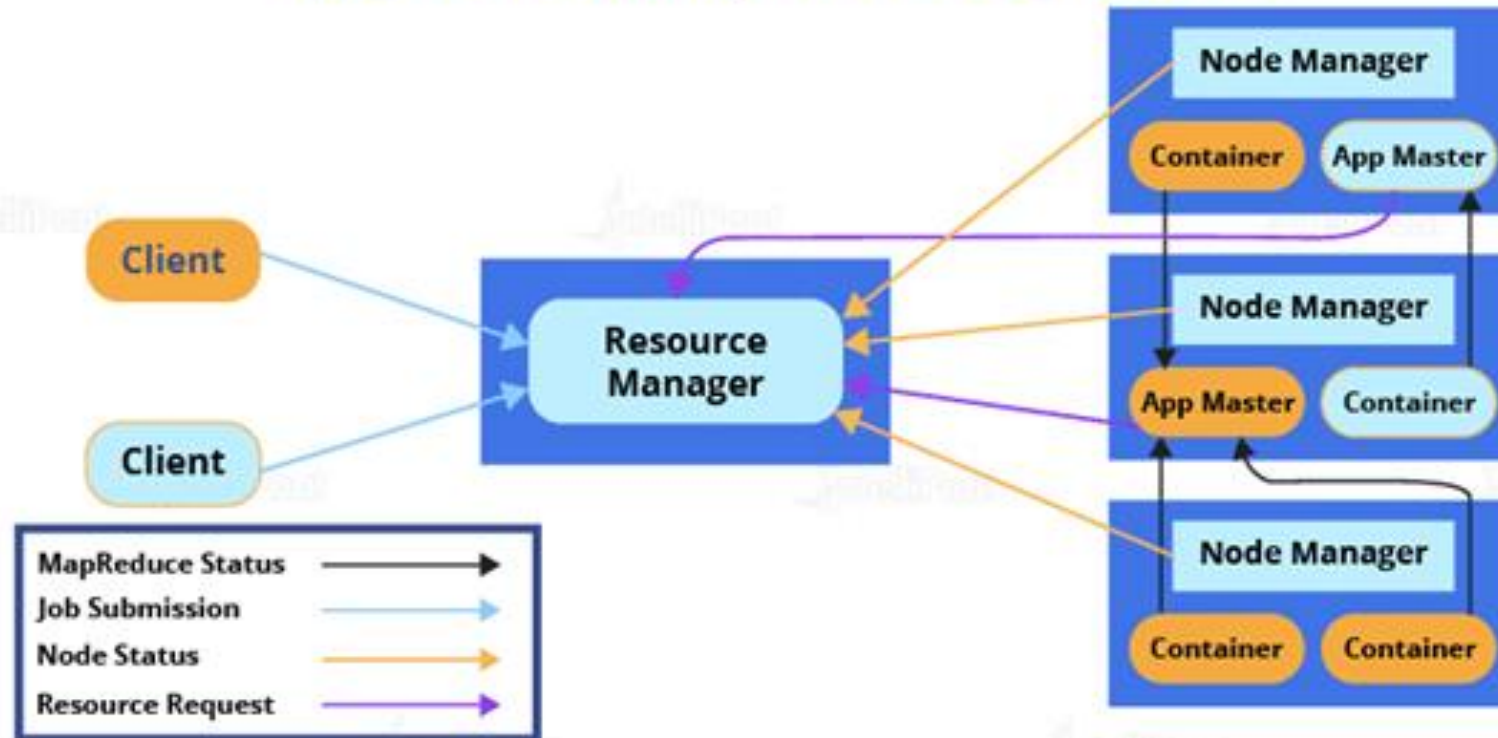
- **Container:** It is a collection of physical resources such as RAM, CPU cores and disk on a single node.
- The containers are invoked by Container Launch Context(CLC) which is a record that contains information such as environment variables, security tokens, dependencies etc.
- A container is a set of physical resources (CPU cores, RAM, disks, etc.) on a single node. The tasks of a container are listed below:
 - It grants the right to an application to use a specific amount of resources (memory, CPU, etc.) on a specific host.
 - YARN containers are particularly managed by a Container Launch context which is Container Life Cycle (CLC).
 - This record contains a map of environment variables, dependencies stored in remotely accessible storage, security tokens, payload for Node Manager services, and the command necessary to create the process.

Some features of YARN

- **Multi-tenancy:** YARN has allowed access to multiple data processing engines such as batch processing engine, stream processing engine, interactive processing engine, graph processing engine and much more. This has given the benefit of multi-tenancy to the company.
- **Cluster Utilization:** Clusters are utilized in an optimized way because clusters are used dynamically in Hadoop with the help of YARN.
- **Compatibility:** YARN is also compatible with the first version of Hadoop, i.e. Hadoop 1.0, because it uses the existing map-reduce apps. So YARN can also be used with Hadoop 1.0.
- **Scalability:** Thousands of clusters and nodes are allowed by the scheduler in Resource Manager of YARN to be managed and extended by Hadoop.

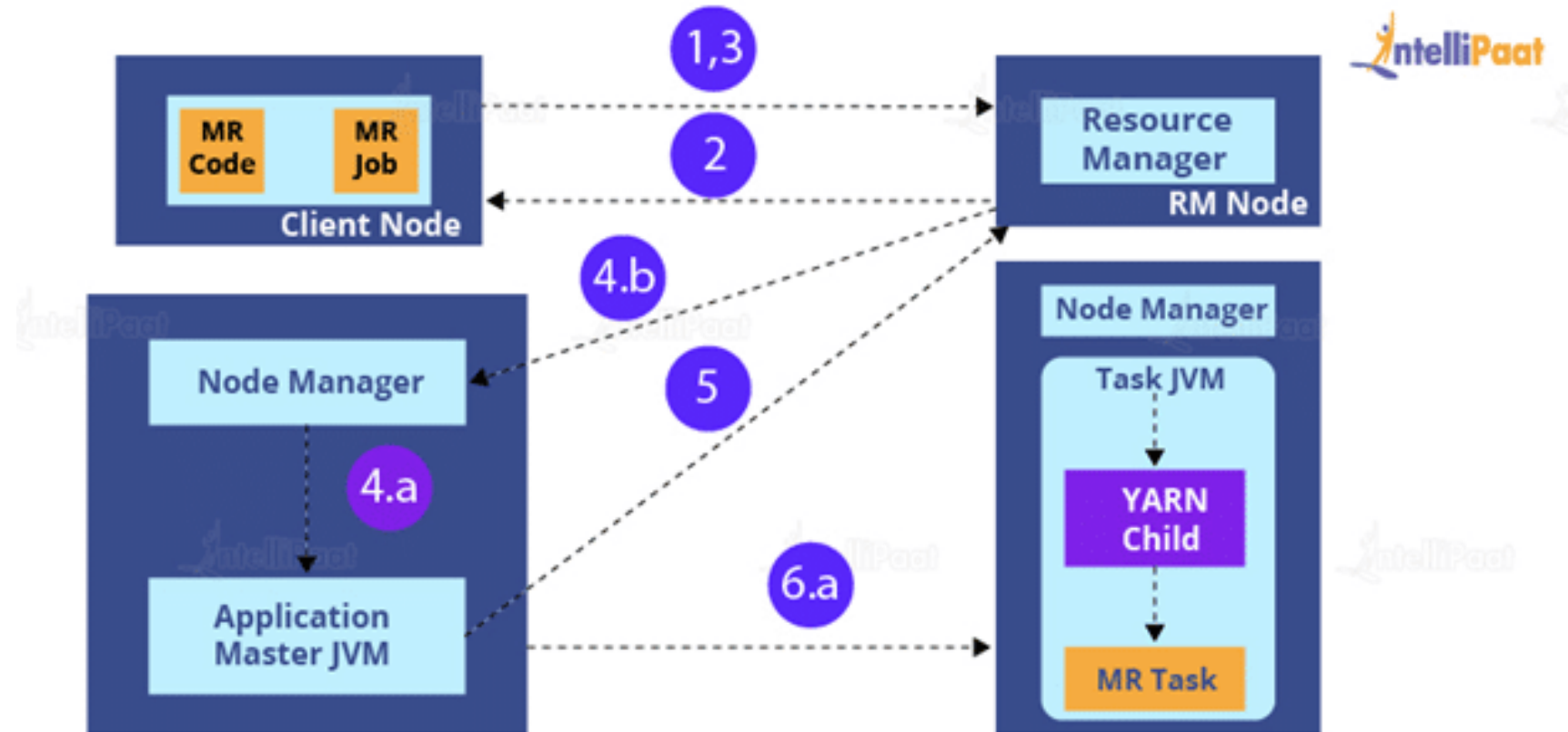
Hadoop YARN Architecture

Apache Hadoop YARN: Architecture



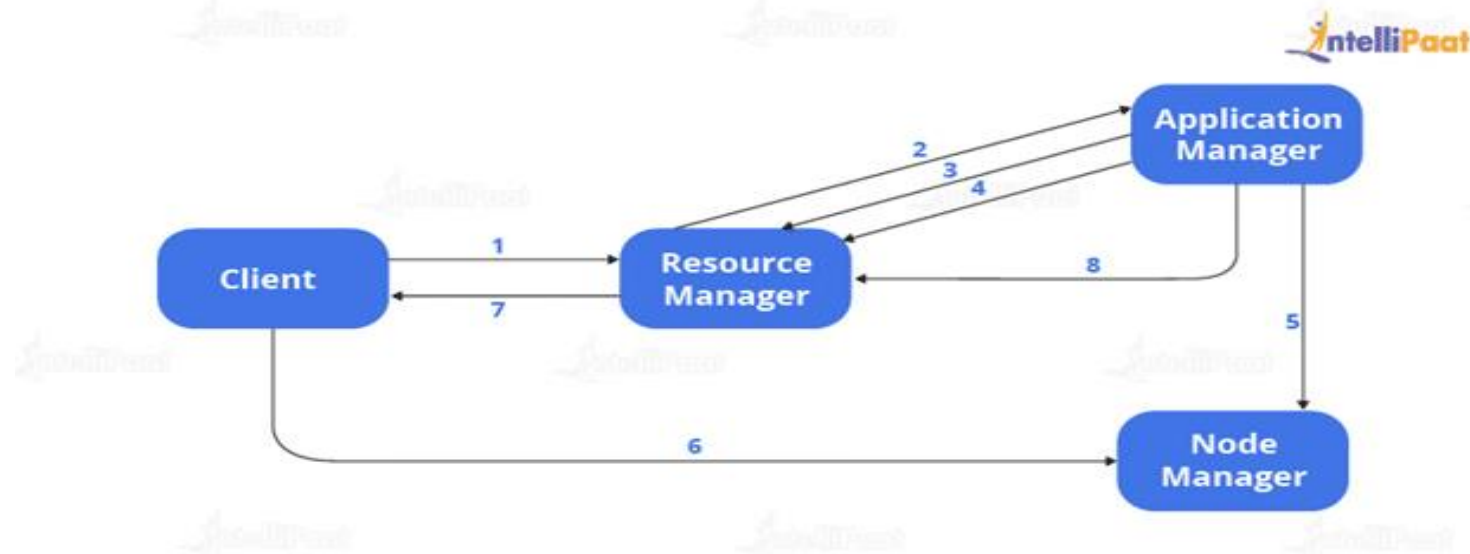
How is an application submitted in YARN?

1. Submit the job
2. Get an application ID
3. Retrieval of the context of application submission
 - Start Container Launch
 - Launch Application Master
4. Allocate Resources.
 - Container
 - Launching
5. Executing



Workflow of an Application in YARN

1. Submission of the application by Client
2. Container allocation for starting Application Manager
3. Registering the Application Manager with Resource Manager
4. Application Manager asks for containers from Resource Manager
5. Application Manager notifies Node Manager to launch containers
6. Application code gets executed in the container
7. Client contacts Resource Manager/Application Manager to monitor the status of the application
8. Application Manager gets disconnected with Resource Manager





Hadoop Ecosystem



oozie
(Work flow)

HCatalog

Table & schema
Management



Pig
(Scripting)



Hive
(Sql Query)



(Machine
Learning)



Drill
(Interactive
Analysis)



AVRO
(JSON)

Thrift

(Cross
Language
Service)

APACHE
HBASE

HBASE
(Columnar
Store)



Sqoop
(Data Collection)



Zookeeper
(Coordination)



Ambari

Apache Ambari
(Management
& Monitoring)

Mapreduce
(Data Processing)



Yarn
(Cluster Resource Management)



FLUME
Flume
(Data Collection)

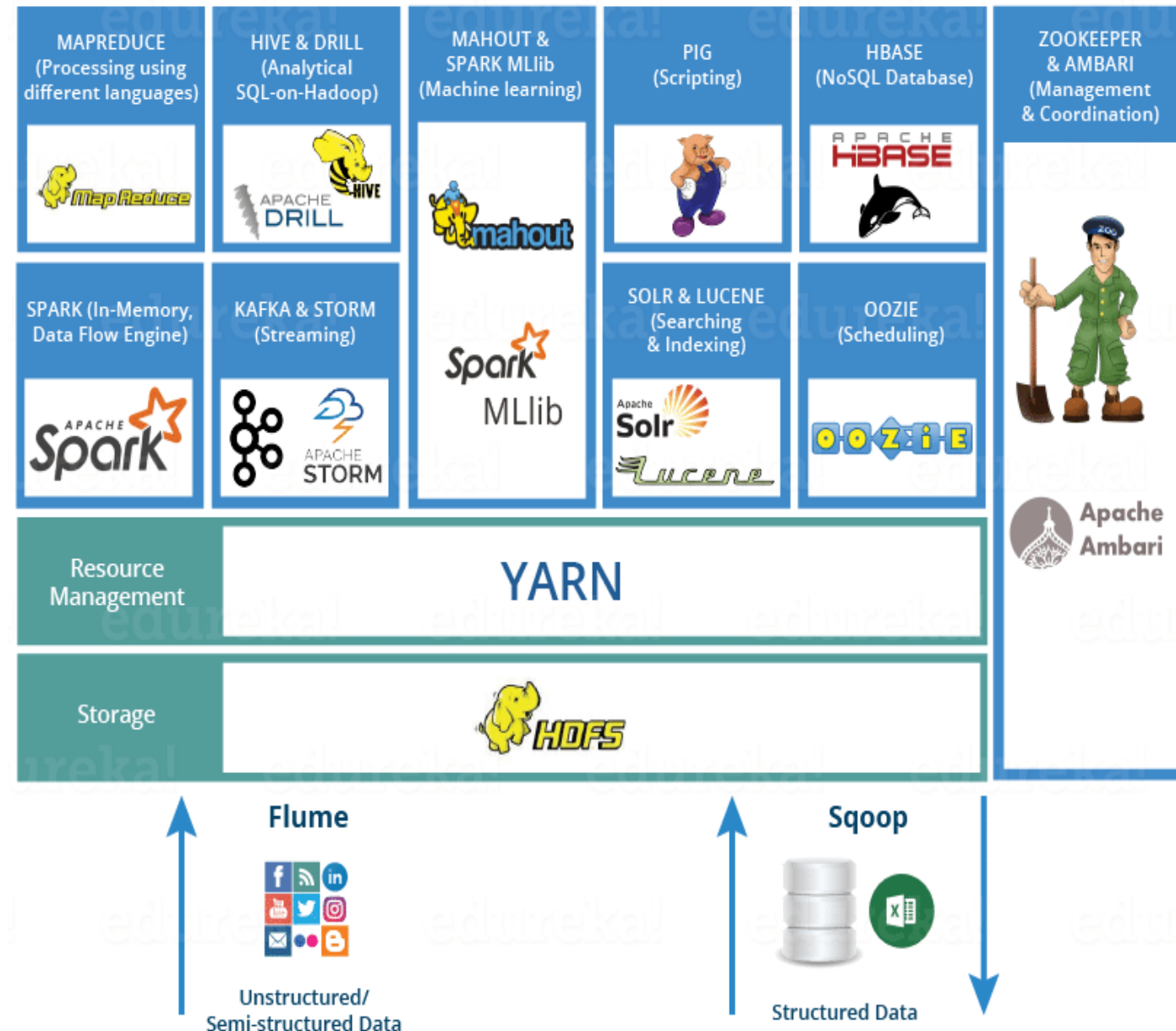
HDFS

(Hadoop Distributed File system)

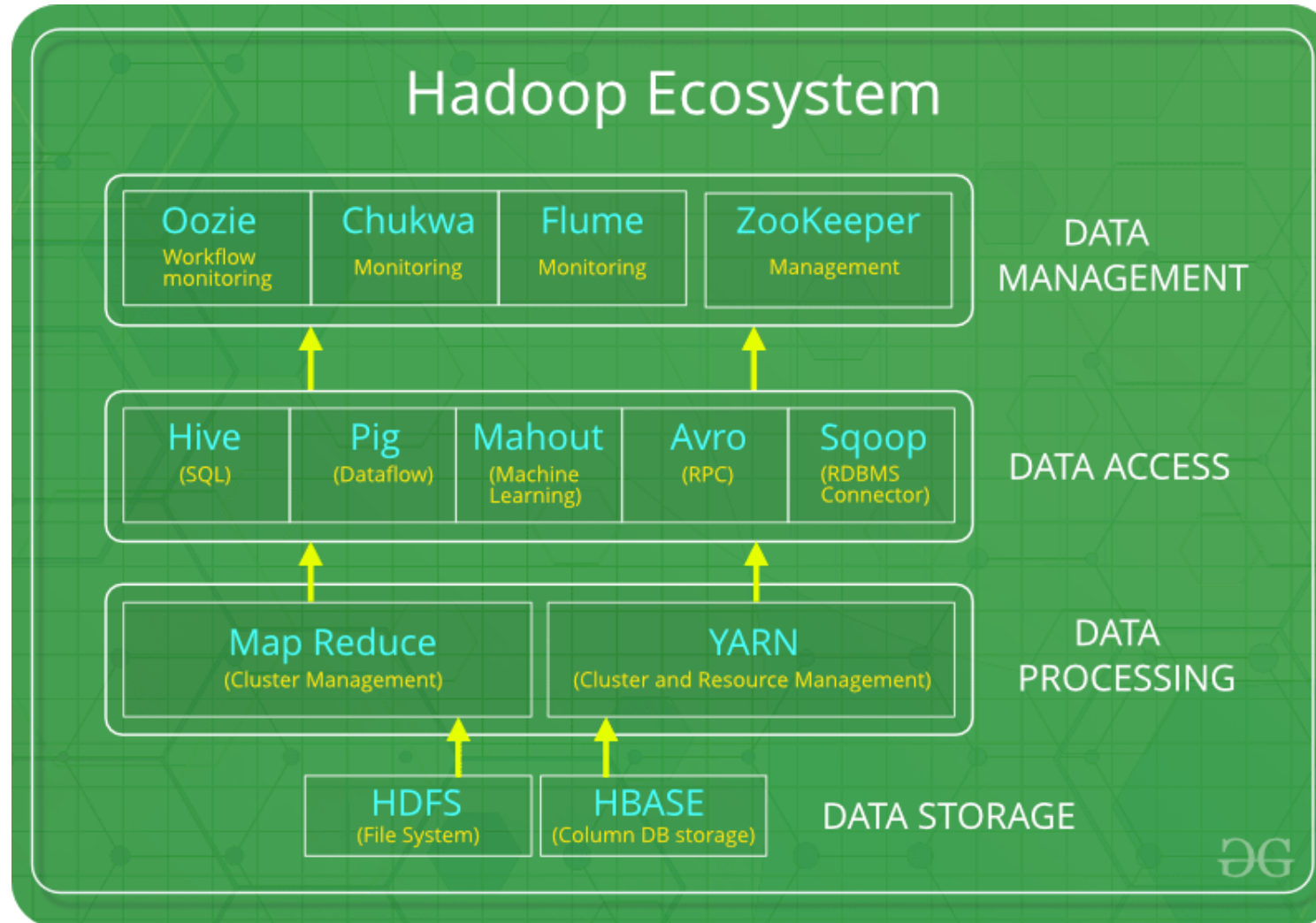


HADOOP ECOSYSTEM

- [HDFS](#) -> Hadoop Distributed File System
- [YARN](#) -> Yet Another Resource Negotiator
- [MapReduce](#) -> Data processing using programming
- [Spark](#) -> In-memory Data Processing
- [PIG](#), [HIVE](#) -> Data Processing Services using Query (SQL-like)
- [HBase](#) -> NoSQL Database
- [Mahout](#), Spark MLlib -> Machine Learning
- [Apache Drill](#) -> SQL on Hadoop
- [Zookeeper](#) -> Managing Cluster
- [Oozie](#) -> Job Scheduling
- [Flume](#), [Sqoop](#) -> Data Ingesting Services
- [Solr & Lucene](#) -> Searching & Indexing
- [Ambari](#) -> Provision, Monitor and Maintain cluster



HADOOP ECOSYSTEM





Apache Pig

- Apache Pig is a high-level language platform for analyzing and querying huge dataset that are stored in HDFS.
- Pig was developed for analyzing large datasets and overcomes the difficulty to write map and reduce functions. It consists of two components: Pig Latin and Pig Engine.
- Pig Latin is the Scripting Language that is similar to SQL. Pig Engine is the execution engine on which Pig Latin runs. Internally, the code written in Pig is converted to MapReduce functions and makes it very easy for programmers who aren't proficient in Java.

10 line of pig latin = approx. 200 lines of Map-Reduce Java code

- It gives you a platform for building data flow for ETL (Extract, Transform and Load), processing and analyzing huge data sets.
- **How Pig works?**
- In PIG, first the load command, loads the data. Then we perform various functions on it like grouping, filtering, joining, sorting, etc. At last, either you can dump the data on the screen or you can store the result back in HDFS.



- The Hadoop ecosystem component, Apache Hive, is an open source data warehouse system for querying and analyzing large datasets stored in Hadoop files. Hive do three main functions: data summarization, query, and analysis.
- Hive use language called HiveQL (HQL), which is similar to SQL. HiveQL automatically translates SQL-like queries into MapReduce jobs which will execute on Hadoop.
HIVE + SQL = HQL
- It has 2 basic components: Hive Command Line and JDBC/ODBC driver.
- Java Database Connectivity (JDBC) and Object Database Connectivity (ODBC) is used to establish connection from data storage.
- Facebook created HIVE for people who are fluent with SQL. Thus, HIVE makes them feel at home while working in a Hadoop Ecosystem.
- **Main parts of Hive are:**
 - Metastore – It stores the metadata.
 - Driver – Manage the lifecycle of a HiveQL statement.
 - Query compiler – Compiles HiveQL into Directed Acyclic Graph(DAG).
 - Hive server – Provide a thrift interface and JDBC/ODBC server.



APACHE MAHOUT

- **Mahout** is open source framework for creating scalable **machine learning** algorithm and data mining library. Once data is stored in Hadoop HDFS, mahout provides the data science tools to automatically find meaningful patterns in those big data sets.
- **Algorithms of Mahout are:**
- **Clustering** – Here it takes the item in particular class and organizes them into naturally occurring groups, such that item belonging to the same group are similar to each other.
- **Collaborative filtering** – It mines user behavior and makes product recommendations (e.g. Amazon recommendations)
- **Classifications** – It learns from existing categorization and then assigns unclassified items to the best category.
- **Frequent pattern mining** – It analyzes items in a group (e.g. items in a shopping cart or terms in query session) and then identifies which items typically appear together.



Apache HBase

- HBase is an open source, non-relational distributed database. In other words, it is a NoSQL database.
- HBase, provide real-time access to read or write data in HDFS.
- It supports all types of data and that is why, it's capable of handling anything and everything inside a Hadoop ecosystem.
- **Apache HBase** is a Hadoop ecosystem component which is a distributed database that was designed to store structured data in tables that could have billions of row and millions of columns.
- For better understanding, let us take an example. You have billions of customer emails and you need to find out the number of customers who has used the word complaint in their emails. The request needs to be processed quickly (i.e. at real time). So, here we are handling a large data set while retrieving a small amount of data. For solving these kind of problems, HBase was designed.



- Drill is an Apache open-source SQL query engine for Big Data exploration.
- Apache Drill is used to drill into any kind of data. It's an open source application which works with distributed environment to analyze large data sets.
- The main power of Apache Drill lies in combining a variety of data stores just by using a single query.

➤ **Key features of Apache Drill are:**

- Low-latency SQL queries
- Dynamic queries on self-describing data in files (such as JSON, Parquet, text) and HBase tables, without requiring metadata definitions in the Hive metastore.
- Nested data support.
- Integration with Apache Hive (queries on Hive tables and views, support for all Hive file formats and Hive UDFs)

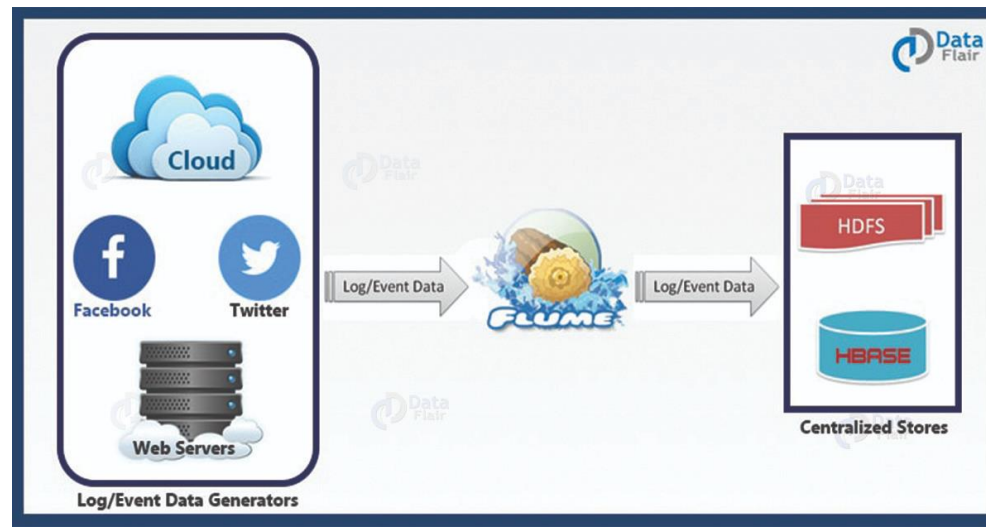


Apache Flume

Flume is an open-source, reliable, and available service used to efficiently collect, aggregate, and move large amounts of data from multiple data sources into HDFS. It can collect data in real-time as well as in batch mode. It has a flexible architecture and is fault-tolerant with multiple recovery mechanisms.

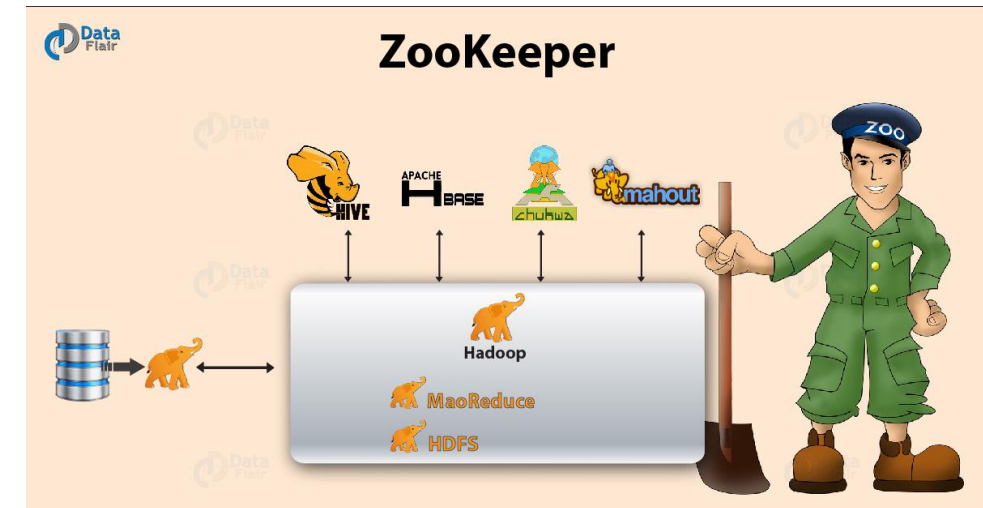
Flume efficiently collects, aggregate and moves a large amount of data from its origin and sending it back to HDFS.

Using Flume, we can get the data from multiple servers immediately into hadoop.





- **Apache Zookeeper** is a centralized service and a Hadoop Ecosystem component for maintaining configuration information, naming, providing distributed synchronization, and providing group services. Zookeeper manages and coordinates a large cluster of machines.
- **Features of Zookeeper:**
- **Fast** – Zookeeper is fast with workloads where reads to data are more common than writes. The ideal read/write ratio is 10:1.
- **Ordered** – Zookeeper maintains a record of all transactions.
- It saves a lot of time by performing synchronization, configuration maintenance, grouping and naming.





APACHE OOZIE

- Apache Oozie as a clock and alarm service inside Hadoop Ecosystem. For Apache jobs, Oozie has been just like a scheduler. It schedules Hadoop jobs and binds them together as one logical work.
 - It is a workflow scheduler system for managing apache Hadoop jobs.
 - Oozie combines multiple jobs sequentially into one logical unit of work.
 - Oozie framework is fully integrated with apache Hadoop stack, YARN as an architecture center and supports Hadoop jobs for apache MapReduce, Pig, Hive, and Sqoop.
- **There are two basic types of Oozie jobs:**
- Oozie workflow – It is to store and run workflows composed of Hadoop jobs e.g., MapReduce, pig, Hive.
 - Oozie Coordinator – It runs workflow jobs based on predefined schedules and availability of data.

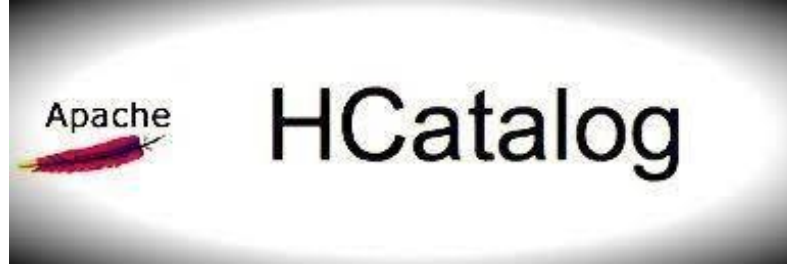


APACHE SQOOP

- **Sqoop** is a command-line interface application for transferring data between relational databases and Hadoop.
- **Sqoop** imports data from external sources into related Hadoop ecosystem components like HDFS, Hbase or Hive. It also exports data from Hadoop to other external sources. Sqoop works with relational databases such as teradata, Netezza, oracle, MySQL.

➤ Features of Apache Sqoop:

- **Import sequential datasets from mainframe** – Sqoop satisfies the growing need to move data from the mainframe to HDFS.
- **Import direct to ORC files** – Improves compression and light weight indexing and improve query performance.
- **Parallel data transfer** – For faster performance and optimal system utilization.
- **Efficient data analysis** – Improve efficiency of data analysis by combining structured data and unstructured data on a schema on reading data lake.
- **Fast data copies** – from an external system into Hadoop.



- HCatalog is a **table storage management tool for Hadoop** that exposes the tabular data of Hive metastore to other Hadoop applications.
- It enables users with different data processing tools (Pig, MapReduce) to easily write data onto a grid.
- It is a table and storage management layer for Hadoop.
- HCatalog supports different components available in Hadoop ecosystems like MapReduce, Hive, and Pig to easily read and write data from the cluster.
- HCatalog is a key component of Hive that enables the user to store their data in any format and structure. By default, HCatalog supports RCFile, CSV, JSON, sequenceFile and ORC file formats.

➤ **Benefits of HCatalog:**

- Enables notifications of data availability.
- With the table abstraction, HCatalog frees the user from overhead of data storage.
- Provide visibility for data cleaning and archiving tools.



- It is a software framework for scalable cross-language services development.
- Thrift is an interface definition language for RPC(Remote procedure call) communication. Hadoop does a lot of RPC calls so there is a possibility of using Hadoop Ecosystem component Apache Thrift for performance or other reasons.
- Apache Thrift allows you to define data types and service interfaces in a simple definition file. Taking that file as input, the compiler generates code to be used to easily build RPC clients and servers that communicate seamlessly across programming languages.
- Thrift is written in C++, but can create code for a number of languages.



- Ambari is an Apache Software Foundation Project which aims at making Hadoop ecosystem more manageable.
- It includes software for **provisioning, managing and monitoring** Apache Hadoop clusters.
- The Ambari provides:
- **Hadoop cluster provisioning:**
 - It gives us step by step process for installing Hadoop services across a number of hosts.
 - It also handles configuration of Hadoop services over a cluster.
- **Hadoop cluster management:**
 - It provides a central management service for starting, stopping and re-configuring Hadoop services across the cluster.
- **Hadoop cluster monitoring:**
 - For monitoring health and status, Ambari provides us a dashboard.
 - The **Amber Alert framework** is an alerting service which notifies the user, whenever the attention is needed. For example, if a node goes down or low disk space on a node, etc.



- Apache Spark is an open-source, distributed processing system used for big data workloads.
- It utilizes in-memory caching, and optimized query execution for fast analytic queries against data of any size.
- It provides development APIs in Java, Scala, Python and R, and supports code reuse across multiple workloads—batch processing, interactive queries, real-time analytics, machine learning, and graph processing.
- Apache Spark started in 2009 as a research project at UC Berkley's AMPLab, a collaboration involving students, researchers, and faculty, focused on data-intensive application domains.
- The goal of Spark was to create a new framework, optimized for fast iterative processing like machine learning, and interactive data analysis, while retaining the scalability, and fault tolerance of Hadoop MapReduce.

Apache Spark Vs. Hadoop MapReduce

- Hadoop MapReduce is a programming model for processing big data sets with a parallel, distributed algorithm.
- Developers can write massively parallelized operators, without having to worry about work distribution, and fault tolerance.
- However, a challenge to MapReduce is the sequential multi-step process it takes to run a job. With each step, MapReduce reads data from the cluster, performs operations, and writes the results back to HDFS.
- Because each step requires a disk read, and write, MapReduce jobs are slower due to the latency of disk I/O.

Apache Spark Vs. Hadoop MapReduce

- Spark was created to address the limitations to MapReduce, by doing processing in-memory, reducing the number of steps in a job, and by reusing data across multiple parallel operations. With Spark, only one-step is needed where data is read into memory, operations performed, and the results written back—resulting in a much faster execution.
- Spark also reuses data by using an in-memory cache to greatly speed up machine learning algorithms that repeatedly call a function on the same dataset.
- Data re-use is accomplished through the creation of DataFrames, an abstraction over Resilient Distributed Dataset (RDD), which is a collection of objects that is cached in memory, and reused in multiple Spark operations. This dramatically lowers the latency making Spark multiple times faster than MapReduce, especially when doing machine learning, and interactive analytics.

Apache Spark Advantages

- Spark is a general-purpose, **in-memory**, fault-tolerant, **distributed processing** engine that allows you to process data efficiently in a distributed fashion.
- Applications running on Spark are **100x** faster than traditional systems.
- You will get great benefits using Spark for data ingestion pipelines.
- Using Spark we can process data from Hadoop **HDFS**, **AWS S3**, **Databricks DBFS**, **Azure Blob Storage**, and many file systems.
- Spark also is used to process real-time data using **Streaming** and **Kafka**.
- Using Spark Streaming you can also stream files from the file system and also stream from the socket.
- Spark natively has **machine learning** and **graph libraries**.

Features of Apache Spark



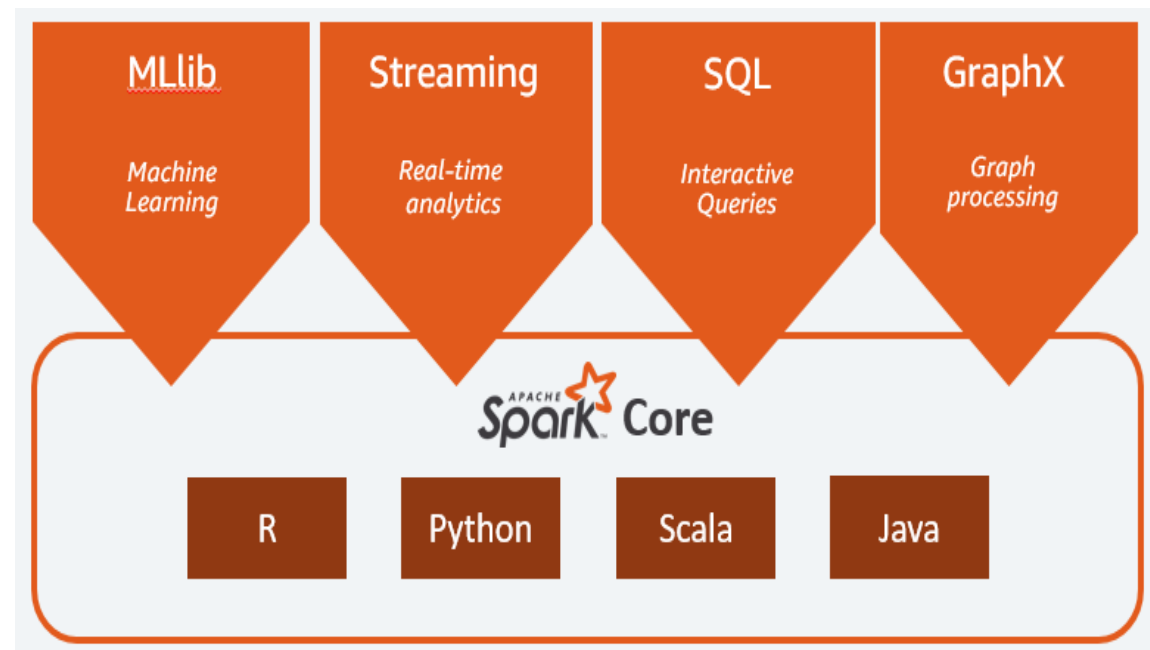
Spark Eco-System



Apache Spark Workloads

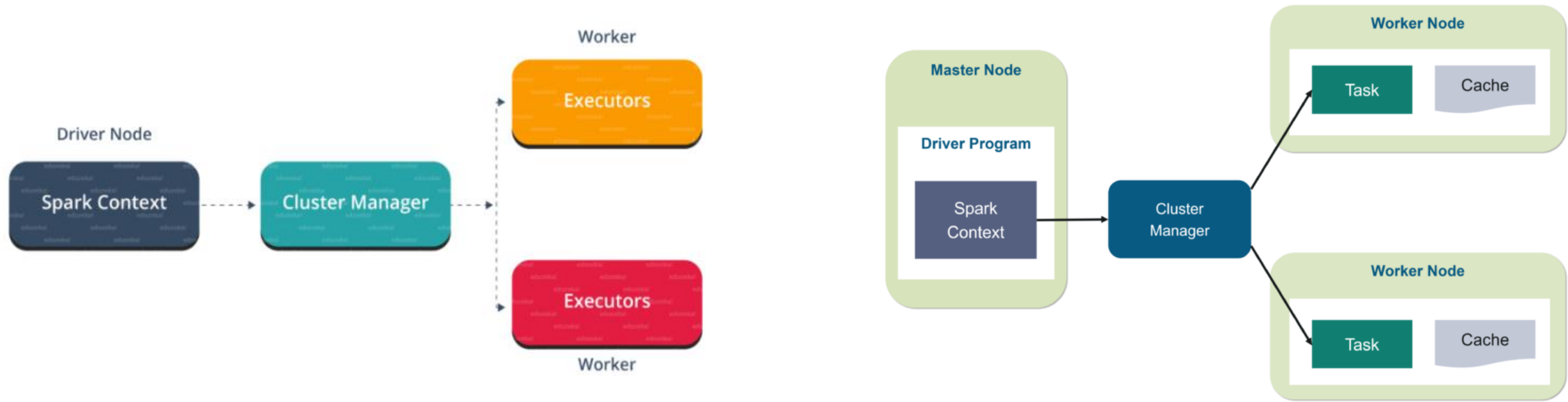
The Spark framework includes:

- Spark Core as the foundation for the platform
- Spark SQL for interactive queries
- Spark Streaming for real-time analytics
- Spark MLlib for machine learning
- Spark GraphX for graph processing



Apache Spark Architecture

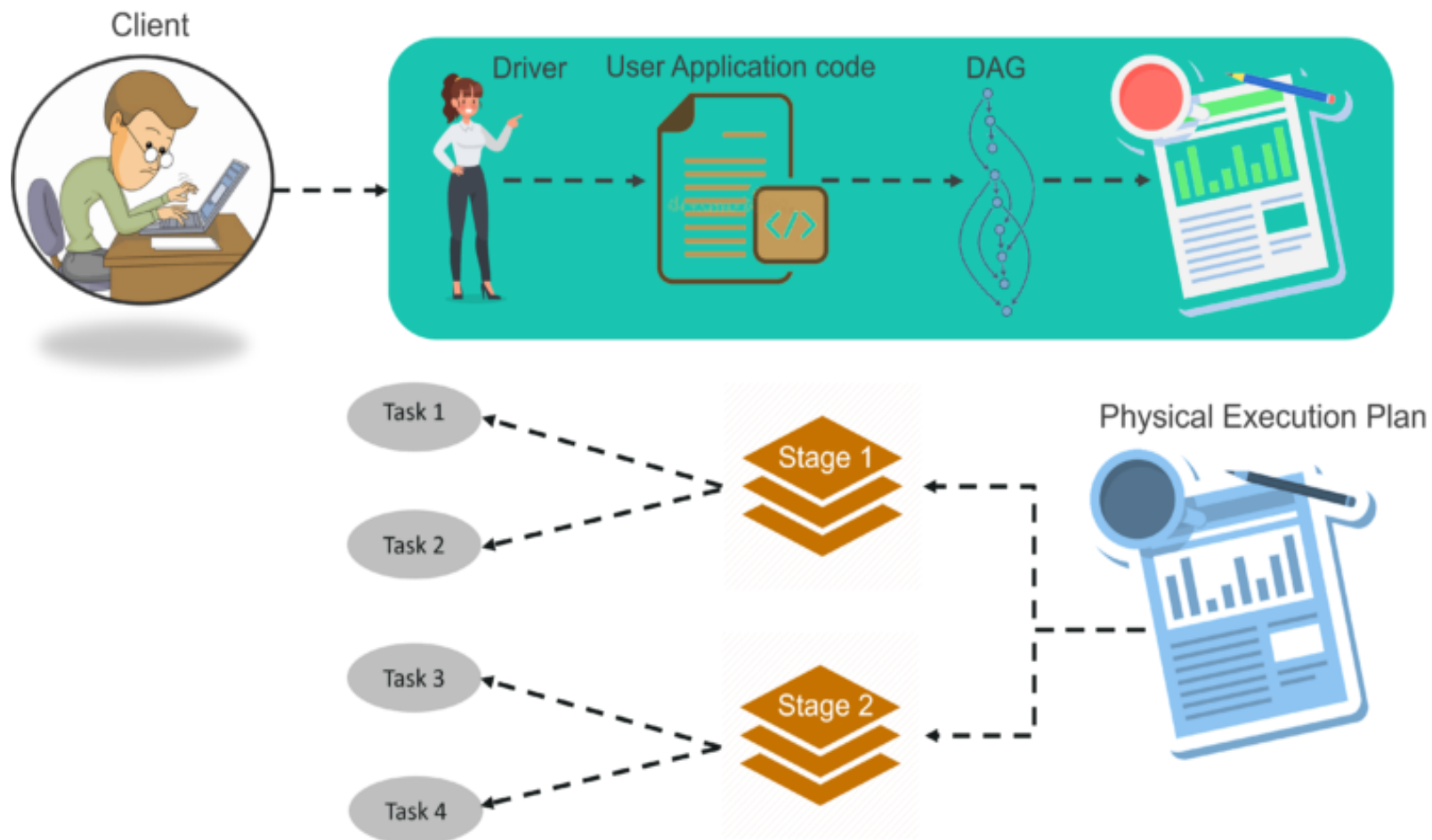
- Apache Spark works in a master-slave architecture where the master is called “Driver” and slaves are called “Workers”.
- When you run a Spark application, Spark Driver creates a context that is an entry point to your application, and all operations (transformations and actions) are executed on worker nodes, and the resources are managed by Cluster Manager.



Apache Spark Architecture

- In your **master node**, you have the *driver program*, which drives your application. The code you are writing behaves as a driver program or if you are using the interactive shell, the shell acts as the driver program.
- Inside the driver program, the first thing you do is, you *create* a ***Spark Context***.
- This Spark context works with the ***cluster manager*** to manage various jobs. The driver program & Spark context takes care of the job execution within the cluster.
- A job is split into multiple tasks which are distributed over the worker node. Anytime an RDD is created in Spark context, it can be distributed across various nodes and can be cached there.
- ***Worker nodes*** are the slave nodes whose job is to basically execute the tasks. These tasks are then executed on the partitioned RDDs in the worker node and hence returns back the result to the Spark Context.
- Spark Context takes the job, breaks the job in tasks and distribute them to the worker nodes. These tasks work on the partitioned RDD, perform operations, collect the results and return to the main Spark Context.

Spark Architecture Infographic



Resilient Distributed Dataset(RDD)

- RDDs are the building blocks of any Spark application. RDDs Stands for:
- ***Resilient***: Fault tolerant and is capable of rebuilding data on failure.
- ***Distributed***: Distributed data among the multiple nodes in a cluster.
- ***Dataset***: Collection of partitioned data with values.
- The data in an RDD is split into chunks based on a key.
- RDDs are highly resilient, i.e, they are able to recover quickly from any issues as the same data chunks are replicated across multiple executor nodes. Thus, even if one executor node fails, another will still process the data.
- once you create an RDD it becomes immutable. By immutable I mean, an object whose state cannot be modified after it is created, but they can surely be transformed.

Resilient Distributed Dataset(RDD)

- Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. Due to this, you can perform transformations or actions on the complete data parallelly.
- There are two ways to create RDDs – parallelizing an existing collection in your driver program, or by referencing a dataset in an external storage system, such as a shared file system, HDFS, HBase, etc.
- With RDDs, you can perform two types of operations:
- **Transformations:** They are the operations that are applied to create a new RDD.
- **Actions:** They are applied on an RDD to instruct Apache Spark to apply computation and pass the result back to the driver.

