**Big Data:**

* Big Data is a collection of data that is huge in volume, yet growing exponentially with time. It is a data with so large size and complexity that none of traditional data management tools can store it or process it efficiently. Big data is also a data but with huge size.
* **The 5 V's of big data (velocity, volume, value, variety and veracity) are the five main and innate characteristics of big data.**

1. Volume:

* The name ‘Big Data’ itself is related to a size which is enormous. Volume is a huge amount of data.
* To determine the value of data, size of data plays a very crucial role. If the volume of data is very large then it is actually considered as a ‘Big Data’.
* This means whether a particular data can actually be considered as a Big Data or not, is dependent upon the volume of data.

2. Velocity:

* Velocity refers to the high speed of accumulation of data.
* In Big Data velocity data flows in from sources like machines, networks, social media, mobile phones etc.
* There is a massive and continuous flow of data. This determines the potential of data that how fast the data is generated and processed to meet the demands.
* Example: There are more than 3.5 billion searches per day are made on Google. Also, FaceBook users are increasing by 22%(Approx.) year by year.

3. Variety:

* It refers to nature of data that is structured, semi-structured and unstructured data.
* It also refers to heterogeneous sources.
* Variety is basically the arrival of data from new sources that are both inside and outside of an enterprise. It can be structured, semi-structured and unstructured.
* Structured data: This data is basically an organized data. It generally refers to data that has defined the length and format of data.
* Semi- Structured data: This data is basically a semi-organised data. It is generally a form of data that do not conform to the formal structure of data. Log files are the examples of this type of data.
* Unstructured data: This data basically refers to unorganized data. It generally refers to data that doesn’t fit neatly into the traditional row and column structure of the relational database. Texts, pictures, videos etc. are the examples of unstructured data which can’t be stored in the form of rows and columns.

4. Veracity:

* It refers to inconsistencies and uncertainty in data, that is data which is available can sometimes get messy and quality and accuracy are difficult to control.
* Big Data is also variable because of the multitude of data dimensions resulting from multiple disparate data types and sources.
* Example: Data in bulk could create confusion whereas less amount of data could convey half or Incomplete Information.

5. Value:

* After having the 4 V’s into account there comes one more V which stands for Value!. The bulk of Data having no Value is of no good to the company, unless you turn it into something useful.
* Data in itself is of no use or importance but it needs to be converted into something valuable to extract Information. Hence, you can state that Value! is the most important V of all the 5V’s.
* In today’s world, there are a lot of data. Big companies utilize those data for their business growth. By analysing this data, the useful decision can be made in various cases as discussed below:

**Benefits or advantages of Big Data**

Following are the benefits or **advantages of Big Data**:  
➨Big data analysis derives innovative solutions. Big data analysis helps in understanding and targeting customers. It helps in optimizing business processes.  
➨It helps in improving science and research.  
➨It improves healthcare and public health with availability of record of patients.  
➨It helps in financial tradings, sports, polling, security/law enforcement etc.  
➨Any one can access vast information via surveys and deliver anaswer of any query.  
➨Every second additions are made.  
➨One platform carry unlimited information.

### Drawbacks or disadvantages of Big Data

Following are the drawbacks or **disadvantages of Big Data**:  
➨Traditional storage can cost lot of money to store big data.  
➨Lots of big data is unstructured.  
➨Big data analysis violates principles of privacy.  
➨It can be used for manipulation of customer records.  
➨It may increase social stratification.  
➨Big data analysis is not useful in short run. It needs to be analyzed for longer duration to leverage its benefits.  
➨Big data analysis results are misleading sometimes.  
➨Speedy updates in big data can mismatch real figures.

Applications of Big Data

[**1. Banking and Securities**](https://www.simplilearn.com/tutorials/big-data-tutorial/big-data-applications#1_banking_and_securities)

[**2. Communications, Media and Entertainment**](https://www.simplilearn.com/tutorials/big-data-tutorial/big-data-applications#2_communications_media_and_entertainment)

[**3. Healthcare Providers**](https://www.simplilearn.com/tutorials/big-data-tutorial/big-data-applications#3_healthcare_providers)

[**4. Education**](https://www.simplilearn.com/tutorials/big-data-tutorial/big-data-applications#4_education)

[**5. Manufacturing and Natural Resources**](https://www.simplilearn.com/tutorials/big-data-tutorial/big-data-applications#5_manufacturing_and_natural_resources)

[**6. Government**](https://www.simplilearn.com/tutorials/big-data-tutorial/big-data-applications#6_government)

[**7. Insurance**](https://www.simplilearn.com/tutorials/big-data-tutorial/big-data-applications#7_insurance)

[**8. Retail and Wholesale trade**](https://www.simplilearn.com/tutorials/big-data-tutorial/big-data-applications#8_retail_and_wholesale_trade)

[**9. Transportation**](https://www.simplilearn.com/tutorials/big-data-tutorial/big-data-applications#9_transportation)

[**10. Energy and Utilities**](https://www.simplilearn.com/tutorials/big-data-tutorial/big-data-applications#10_energy_and_utilities)

**BIG DATA**

* + - **Hadoop**
    - **Hive**
* Hadoop is a framework that allows you to first store Big Data in a distributed environment,

so that, you can process it parallely.

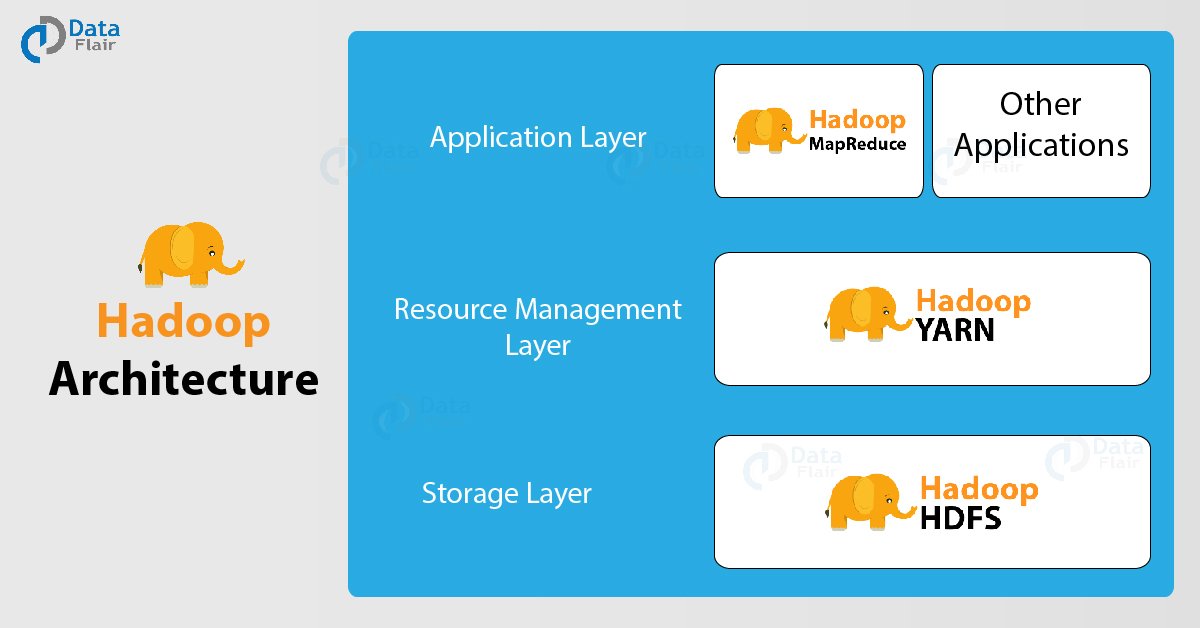
Hadoop consists of four main modules:

1. HDFS (Hadoop Distributed File System)

2. YARN (Yet Another Resource Negotiator)

3. MapReduce

4. Hadoop Common



1. HDFS (Hadoop Distributed File System)

* HDFS creates an abstraction, it Similar as virtualization,
* you can see HDFS logically as a single unit for storing Big Data, but actually you are storing your data across multiple nodes in a distributed fashion.
* HDFS follows master-slave architecture

Name Node

\* master daemon

\* maintain and manages data nodes

\* receives heartbeat and block report from all the Data nodes

Data Node

\* slave daemon

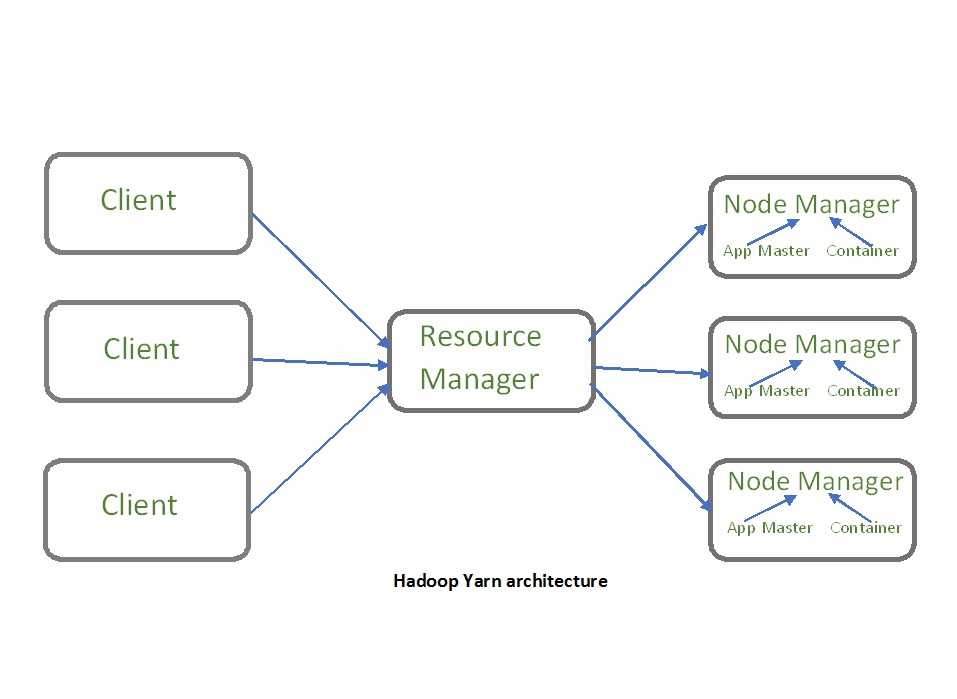
\* stores actual data

\* serves read and write requests

2. YARN (Yet Another Resource Negotiator)

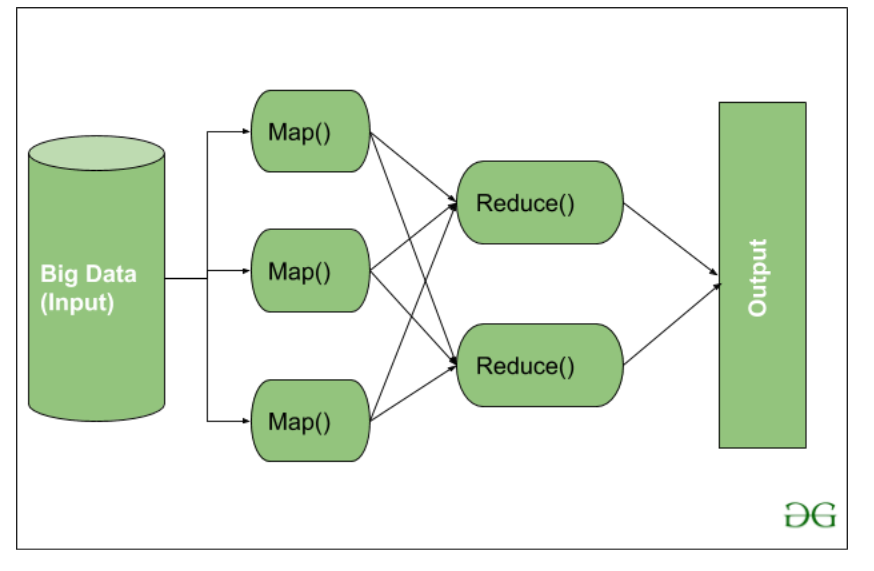
YARN performs all your processing activities by allocating resources and scheduling tasks.

It has two major components, i.e. Resource Manager and Node Manager.



3. Map Reduce

* A framework that helps programs do the parallel computation on data.
* The map task takes input data and converts it into a dataset that can be computed in key value pairs.
* The output of the map task is consumed by reduce tasks to aggregate output and provide the desired result.



4. Hadoop Common – Provides common Java libraries that can be used across all modules.

## **Advantages of Hadoop**

* **Fast:** In HDFS the data distributed over the cluster and are mapped which helps in faster retrieval. Even the tools to process the data are often on the same servers, thus reducing the processing time. It is able to process terabytes of data in minutes and Peta bytes in hours.
* **Scalable:** Hadoop cluster can be extended by just adding nodes in the cluster.
* **Cost Effective:** Hadoop is open source and uses commodity hardware to store data so it really cost effective as compared to traditional relational database management system.
* **Resilient to failure:** HDFS has the property with which it can replicate data over the network, so if one node is down or some other network failure happens, then Hadoop takes the other copy of data and use it. Normally, data are replicated thrice but the replication factor is configurable.

## **What is Hive**

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.

Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

### Hive is not

* A relational database
* A design for OnLine Transaction Processing (OLTP)
* A language for real-time queries and row-level updates

## **Features of Hive**

* It stores schema in a database and processed data into HDFS.
* It is designed for OLAP.
* It provides SQL type language for querying called HiveQL or HQL.
* It is familiar, fast, scalable, and extensible.

## **Architecture of Hive**



|  |  |
| --- | --- |
| **Unit Name** | **Operation** |
| User Interface | Hive is a data warehouse infrastructure software that can create interaction between user and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server). |
| Meta Store | Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping. |
| HiveQL Process Engine | HiveQL is similar to SQL for querying on schema info on the Metastore. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it. |
| Execution Engine | The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce. |
| HDFS or HBASE | Hadoop distributed file system or HBASE are the data storage techniques to store data into file system. |

## **Spark & its Features**

Apache Spark is an open source cluster computing framework for real-time data processing. The main feature of Apache Spark is its ***in-memory cluster computing*** that increases the processing speed of an application. Spark provides an interface for programming entire clusters with implicit **data parallelism and fault tolerance**. It is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries, and streaming.

### ****Features of Apache Spark:****

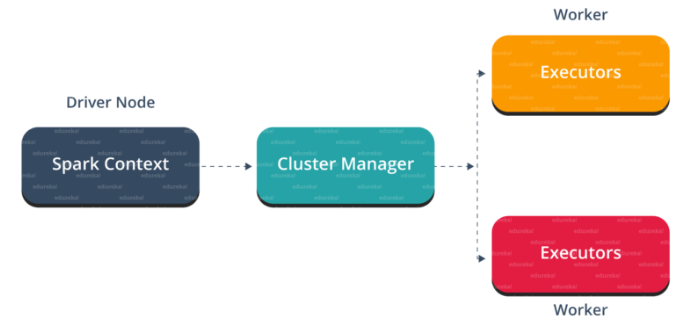
### Spark Features- Spark Architecture-Edureka

1. **Speed**Spark runs up to 100 times faster than Hadoop Map Reduce for large-scale data processing. It is also able to achieve this speed through controlled partitioning.
2. **PowerfulCaching**Simple programming layer provides powerful caching and disk persistence capabilities.
3. **Deployment**It can be deployed through ***Mesos, Hadoop via YARN, or Spark’s own cluster manager.***
4. **Real-Time**  
   It offers Real-time computation & low latency because of ***in-memory computation.***
5. **Polyglot**  
   Spark provides high-level APIs in Java, Scala, Python, and R. Spark code can be written in any of these four languages. It also provides a shell in Scala and Python.

## **Spark Architecture Overview**

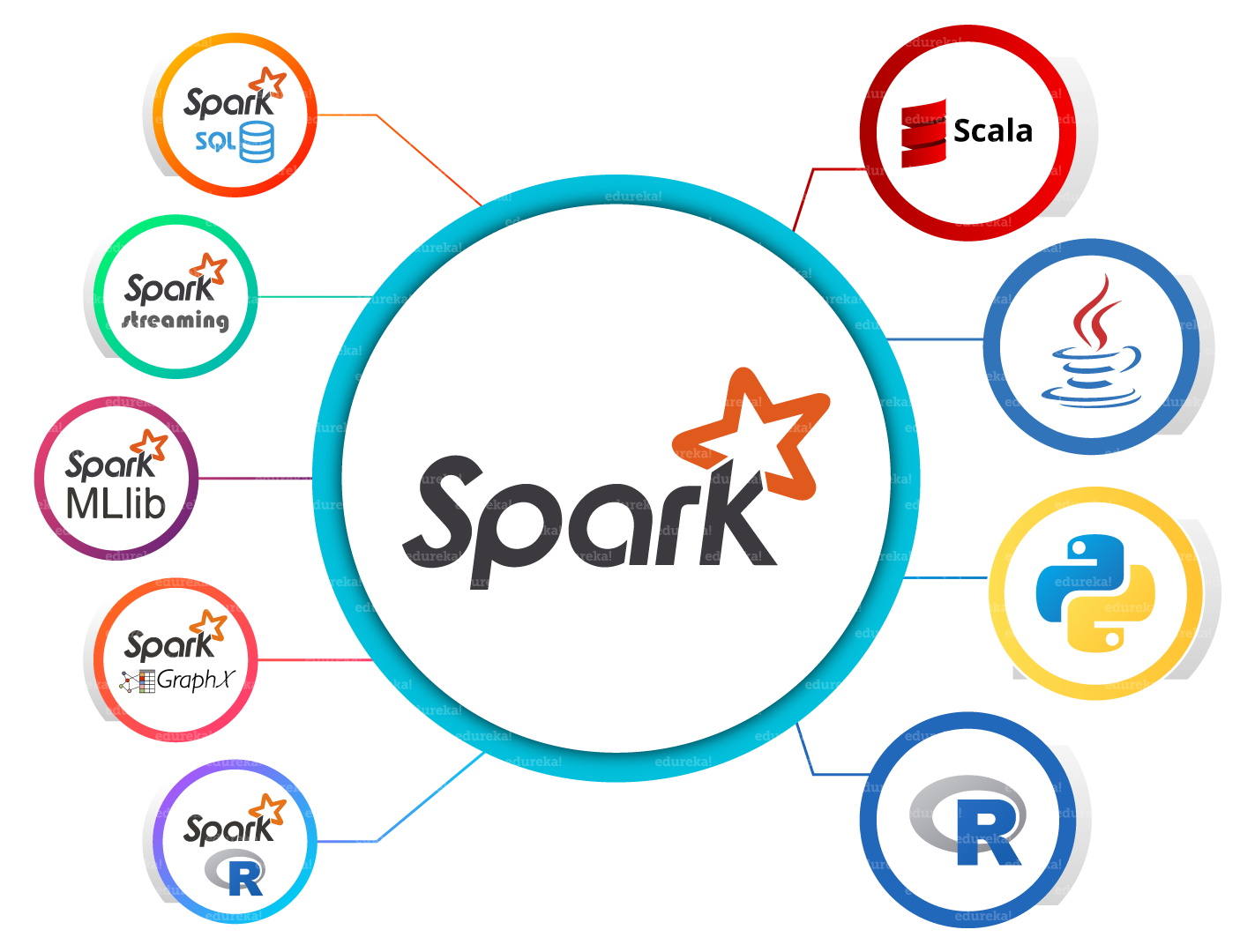
Apache Spark has a well-defined layered architecture where all the spark components and layers are loosely coupled. This architecture is further integrated with various extensions and libraries. Apache Spark Architecture is based on two main abstractions:

* Resilient Distributed Dataset (RDD)
* Directed Acyclic Graph (DAG)



## **Spark Eco-System**

As you can see from the below image, the spark ecosystem is composed of various components like Spark SQL, Spark Streaming, MLlib, GraphX, and the Core API component.

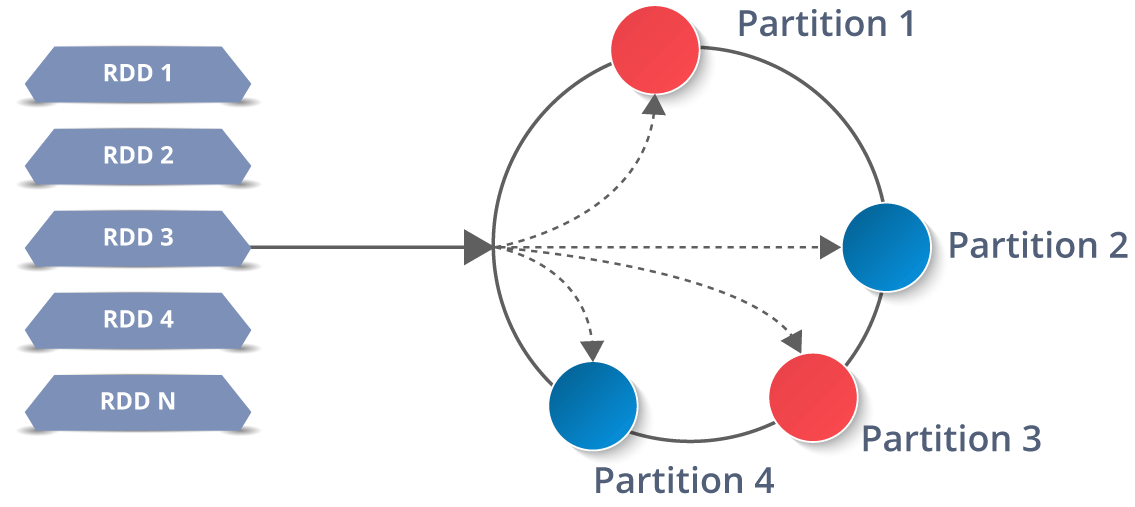


1. **Spark Core**Spark Core is the base engine for large-scale parallel and distributed data processing. Further, additional libraries which are built on the top of the core allows diverse workloads for streaming, SQL, and machine learning. It is responsible for memory management and fault recovery, scheduling, distributing and monitoring jobs on a cluster & interacting with storage systems.
2. **Spark Streaming**Spark Streaming is the component of Spark which is used to process real-time streaming data. Thus, it is a useful addition to the core Spark API. It enables high-throughput and fault-tolerant stream processing of live data streams.
3. **SparkSQL**Spark SQL is a new module in Spark which integrates relational processing with Spark’s functional programming API. It supports querying data either via SQL or via the Hive Query Language. For those of you familiar with RDBMS, Spark SQL will be an easy transition from your earlier tools where you can extend the boundaries of traditional relational data processing.
4. **GraphX**GraphX is the Spark API for graphs and graph-parallel computation. Thus, it extends the Spark RDD with a Resilient Distributed Property Graph. At a high-level, GraphX extends the Spark RDD abstraction by introducing the Resilient Distributed Property Graph (a directed multigraph with properties attached to each vertex and edge).
5. **MLlib (MachineLearning)**  
   MLlib stands for Machine Learning Library. Spark MLlib is used to perform machine learning in Apache Spark.
6. ***SparkR***It is an R package that provides a distributed data frame implementation. It also supports operations like selection, filtering, aggregation but on large data-sets.

## **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

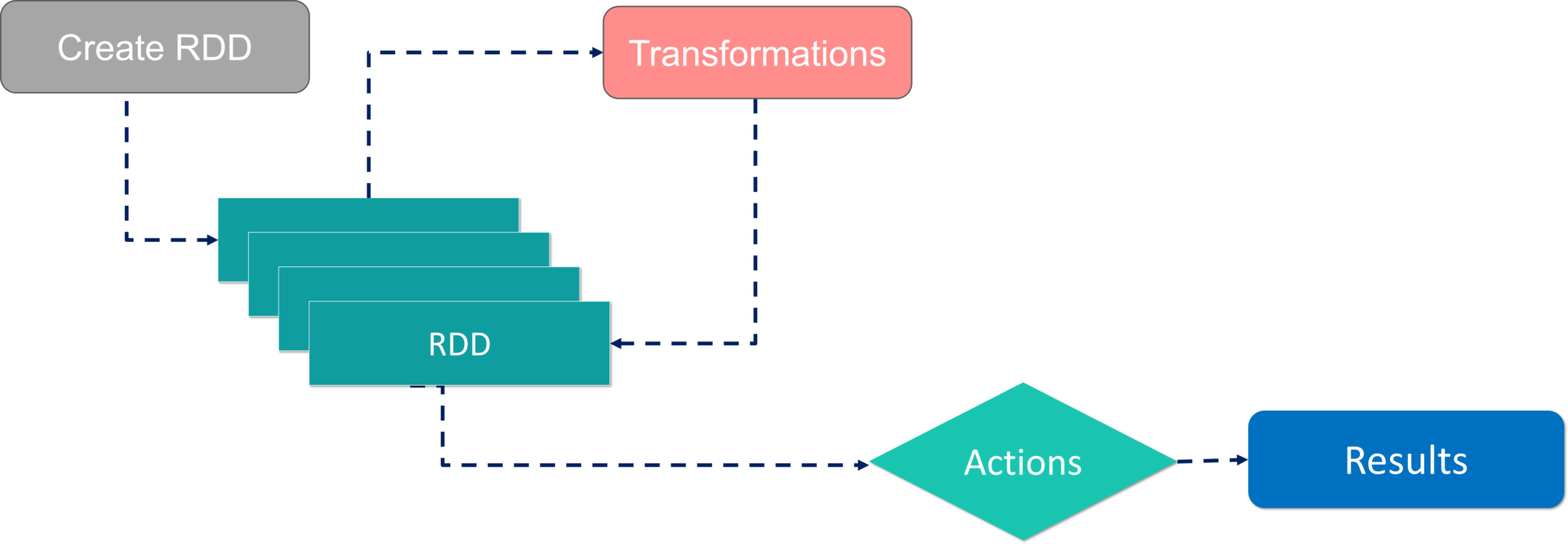


It is a layer of abstracted data over the distributed collection. It is immutable in nature and follows [*lazy transformations*](https://www.edureka.co/blog/spark-tutorial/#Spark_Features).

Now you might be wondering about its working. Well, 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. This allows you to perform your functional calculations against your dataset very quickly by harnessing the power of multiple nodes.

Moreover, 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.

Talking about the distributed environment, 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. Also, you don’t have to worry about the distribution, because Spark takes care of that.



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:

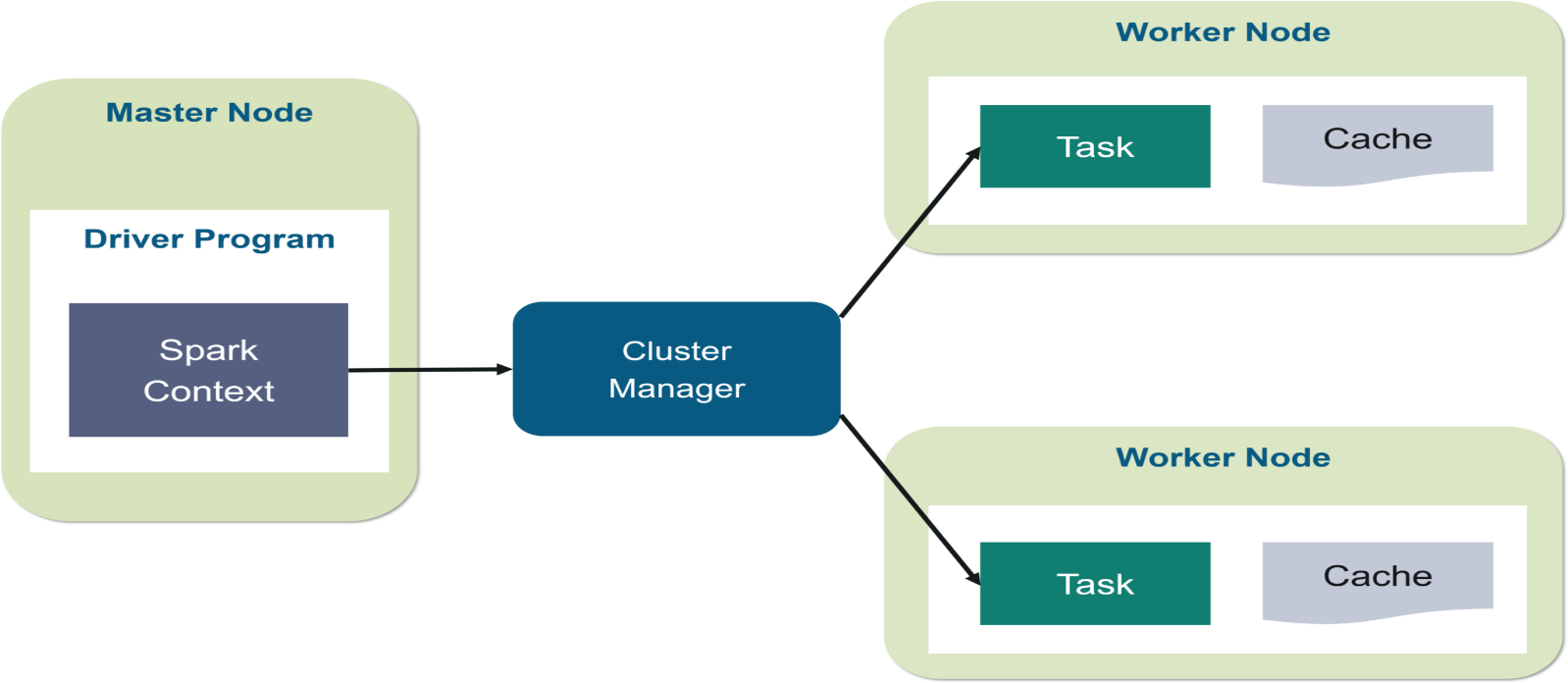
1. **Transformations:**They are the operations that are applied to create a new RDD.
2. **Actions:** They are applied on an RDD to instruct Apache Spark to apply computation and pass the result back to the driver.

I hope you got a thorough understanding of RDD concepts. Now let’s move further and see the working of Spark Architecture.

## **Working of Spark Architecture**

As you have already seen the basic architectural overview of Apache Spark, now let’s dive deeper into its working.

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.*** Assume that the Spark context is a gateway to all the Spark functionalities. It is similar to your database connection. Any command you execute in your database goes through the database connection. Likewise, anything you do on Spark goes through Spark context.

Now, 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.

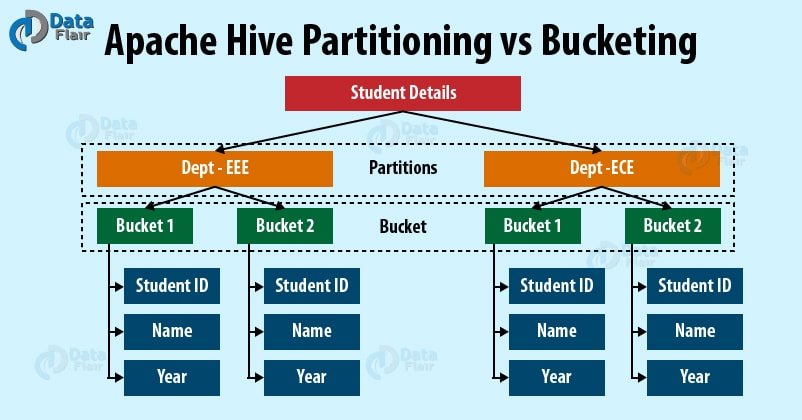
Spark Context takes the job, breaks the job in tasks and distribute them to the worker nodes. These tasks work on the partitioned RDD, perform operations, collect the results and return to the main Spark Context.

If you increase the number of workers, then you can divide jobs into more partitions and execute them parallelly over multiple systems. It will be a lot faster.

With the increase in the number of workers, memory size will also increase & you can cache the jobs to execute it faster.

|  |  |
| --- | --- |
| Repartition | coalesce |
| 1. Increase or decrease partitions. | 1. Only decrease the number of partitions. |
| 2. Repartition always involves a shuffle. | 2.Coalesce doesn’t involve a full shuffle |
| 3.Repartition works by creating new partitions | 3. if the number of partitions is reduced from 5 to 2.  Coalesce will not move data in 2 executors and move the data from the remaining 3 executors to the 2 executors. Thereby avoiding a full shuffle. and doing a full shuffle to move data around. |
| 4. Results in more or less equal sized partitions. | 4.Because of the above reason the partition size vary by a high degree |
| 5. Since a full shuffle takes place, repartition is less performant than coalesce. | 5. Since full shuffle is avoided, coalesce is more performant than repartition. |

| PARTITIONING | BUCKETING |
| --- | --- |
| Directory is created on HDFS for each partition. | File is created on HDFS for each bucket. |
| You can have one or more Partition columns | You can have only one Bucketing column |
| You can’t manage the number of partitions to create | You can manage the number of buckets to create by specifying the count |
| NA | Bucketing can be created on a partitioned table |
| Uses PARTITIONED BY | Uses CLUSTERED BY |
|  |  |



* **Partitioning –**Apache Hive organizes tables into partitions for grouping same type of data together based on a column or partition key. Each table in the hive can have one or more partition keys to identify a particular partition. Using partition we can make it faster to do queries on slices of the data.
* **Bucketing –**In Hive Tables or partition are subdivided into buckets based on the hash function of a column in the table to give extra structure to the data that may be used for more efficient queries.

Data skewness

Usually, in Apache Spark, data skewness is **caused by transformations that change data partitioning like join, groupBy, and orderBy**. For example, joining on a key that is not evenly distributed across the cluster, causing some partitions to be very large and not allowing Spark to process data in parallel.

Data shuffle

The Spark SQL shuffle is **a mechanism for redistributing or re-partitioning data so that the data is grouped differently across partitions**, based on your data size you may need to reduce or increase the number of partitions of RDD/DataFrame using spark. sql. shuffle. partitions configuration or through code.