**Spark:**

Is a unified,open source,parallel,data processing framework for big data analytics.

On a highlevel ,it is a computing framework that allows to:

Load/ingest large amount of real time data(streaming)

Transform /clean large amounts of data(sql operations )

Run ml operations on large amounts of data(mlib)

Process graph networks of large data(graph processing)

Features of spark:

Speed,powerful caching,deployment,realtime,polyglot,scalable.

Apache spark handles these huge amounts of data through an abstraction called RDDs and DAG

There are 2 types of **abstraction layers in apache spark architecture**

**Rdds:**

It is a java object at its core and has built in methods provided by spark that allows to manipulate original data we pass in.

Rdds effiectively handles bigdata through partitioning subsets of data that cn be operated on in parallel across various nodes and has replications per partition to prevent loss of data.

Lets give spark 100gb of data while It has 5 worker nodes.

Then it divides 20 to each workernode and if suddenly a node collapses its copy is present in another node so we avoid losing of data.

Resilient: Restore the data on failure.

Distributed: Data is distributed among different nodes.

Dataset: Group of data.

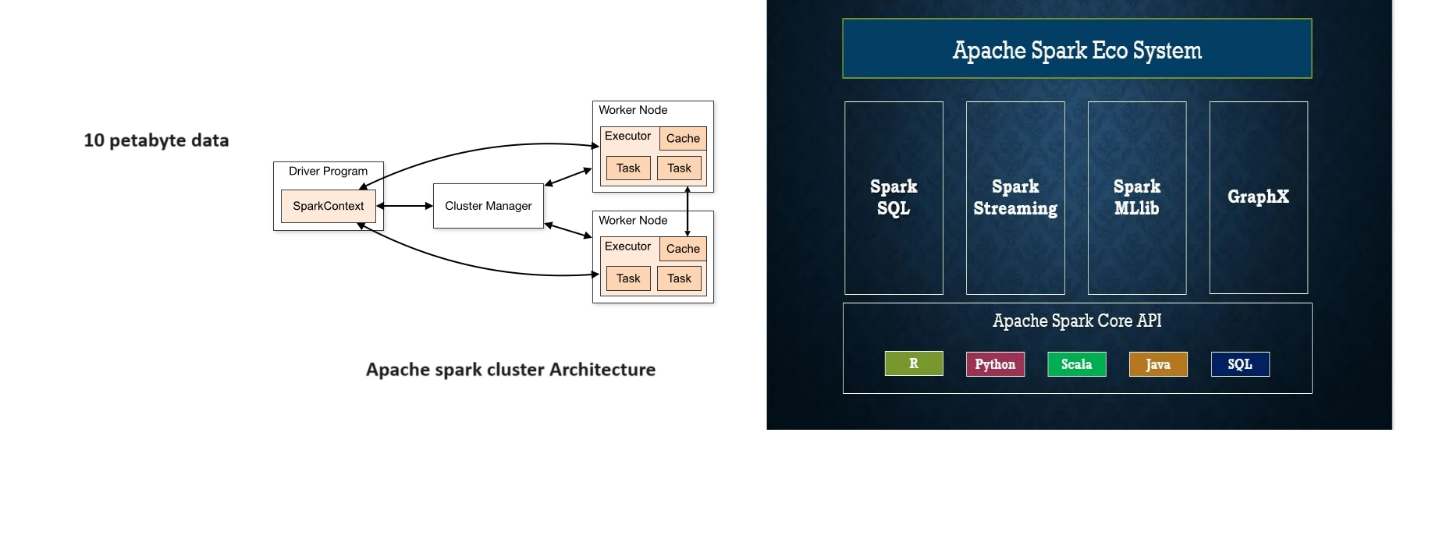
**DAG:**

Directed Acyclic Graph is a finite direct graph that performs a sequence of computations on data. Each node is an RDD partition, and the edge is a transformation on top of data.

As we know spark does lazy evaluation until we perform an action nothing will be seen by us but internally in the meantime it is creating DAG for the performed transformations.

The driver converts the program into a DAG for each job

**Sparks architecture:**



**Driver program:**

When the Driver Program in the Apache Spark architecture executes, it calls the real program of an application and creates a SparkContext. SparkContext contains all of the basic functions. The Spark Driver includes several other components, including a DAG Scheduler, Task Scheduler, Backend Scheduler, and Block Manager, all of which are responsible for translating user-written code into jobs that are actually executed on the cluster.

 A driver program monitors the executors during their performance

**Spark context:**

entry point to any spark functionality.

It represents connection to a spark cluster and Is the place where the user can configure the common properties for entire application.

Using this every spark application can communicate with datasources and performs certain operations.

It requires explicit management and can only be used once in a spark application.

**Spark session:**

Eliminates the need of creating different spark contexts.

Integrates sparkcontext and provides high level APIs .

With spark sessions developers can leverage spark capabilities without explicitly managing multiple contexts.

**Cluster manager:**

 Spark Driver works in conjunction with the Cluster Manager to control the execution of various other jobs. The cluster Manager does the task of allocating resources for the job. Once the job has been broken down into smaller jobs, which are then distributed to worker nodes, SparkDriver will control the execution.

**Worker nodes:**

The slave nodes function as executors, processing tasks, and returning the results back to the spark context. The master node issues tasks to the Spark context and the worker nodes execute them. They make the process simpler by boosting the worker nodes (1 to n) to handle as many jobs as possible in parallel by dividing the job up into sub-jobs on multiple machines.

**Executors:**

An executor is responsible for executing a job and storing data in a cache at the outset. Executors first register with the driver programme at the beginning. These executors have a number of time slots to run the application concurrently. The executor runs the task when it has loaded data and they are removed in idle mode. The executors are allocated dynamically and constantly added and removed during the execution of the tasks.

You can choose from three different **execution modes:** local, shared, and dedicated. These determine where your app’s resources are physically located when you run your app. You can decide where to store resources locally, in a shared location, or in a dedicated location.

1. Cluster mode
2. Client mode
3. Local mode

**Cluster mode:** Cluster mode is the most frequent way of running Spark Applications. In cluster mode, a user delivers a pre-compiled JAR, Python script, or R script to a cluster manager. Once the cluster manager receives the pre-compiled JAR, Python script, or R script, the driver process is launched on a worker node inside the cluster, in addition to the executor processes. This means that the cluster manager is in charge of all Spark application-related processes.

**Client mode:** In contrast to cluster mode, where the Spark driver remains on the client machine that submitted the application, the Spark driver is removed in client mode and is therefore responsible for maintaining the Spark driver process on the client machine. These machines, usually referred to as gateway machines or edge nodes, are maintained on the client machine.

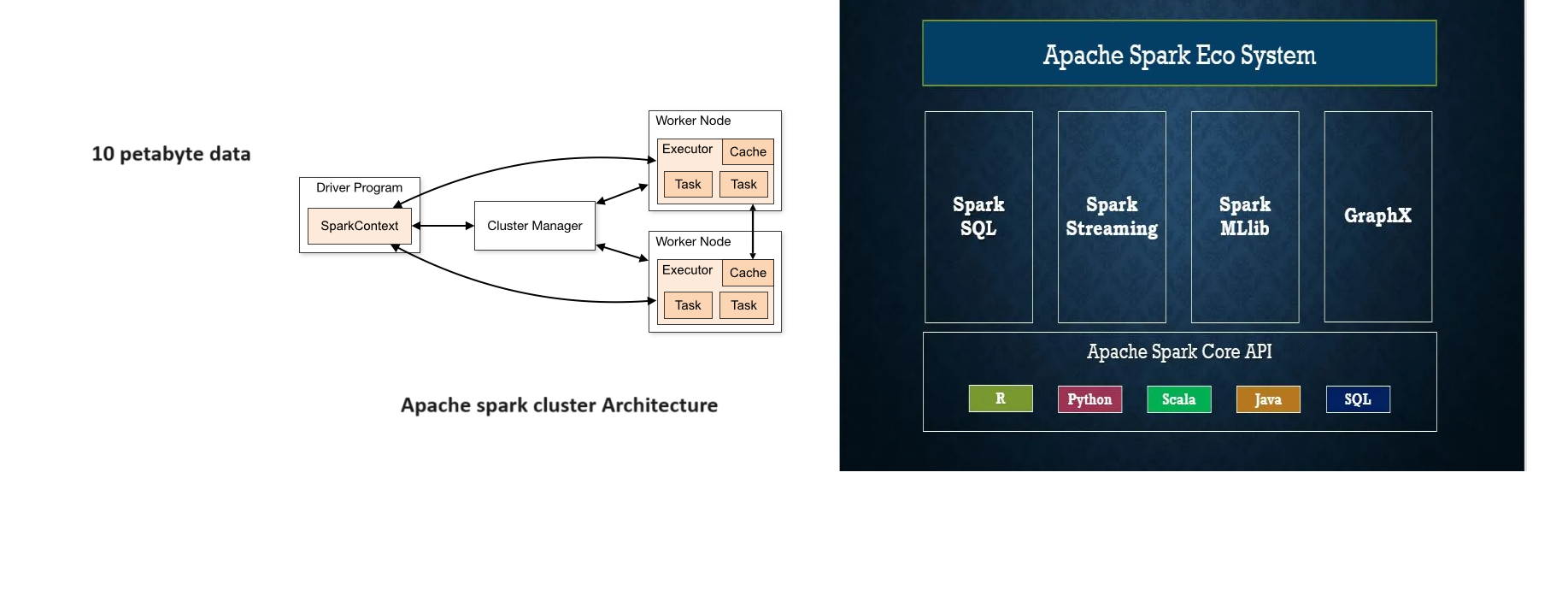
**Transformations :are** the operations performed on the rdds.

Transformations are not actually performed but instead placed into an execution map only to be activated later by an action.

**Actions:**count,collect etc.

Actual execution takes place after calling action methods.

**Spark components:**



**Spark core:**

Spark Core is the heart of the Apache Spark framework. Spark Core provides the execution engine for the Spark platform which is required and used by other components which are built on top of Spark Core as per the requirement. Spark Core provides the in-built memory computing and referencing datasets stored in external storage systems. It is Spark’s core responsibility to perform all the basic I/O functions, scheduling, monitoring, etc. Also, fault recovery and effective memory management are Spark Core’s other important functions.

**Spark sql:**

Users can perform extract, transform and load functions on data from a variety of sources in different formats like JSON, Parquet or Hive and then execute ad-hoc queries using Spark SQL.

**Spark streaming:**

 Spark Streaming is used for analyzing a continuous stream of data

Just like how Spark SQL has the concept of Dataframe/Dataset built on top of RDD, Spark streaming has something called Dstream. This is a collection of RDDs that embodies the entire stream data. The good thing about Dstream is that we can apply most of the built-in functions on RDDs also on the DStream like flatMap, map, etc. Also, the Dstream can be broken into individual RDDs and can be processed one chunk at a time. Spark developers can reuse the same code for stream and batch processing and can also integrate the streaming data with historical data.

**Mlib:**

Machine learning libraries.

**GraphX:** For graphs and graph-parallel processing Apache Spark provides another API called GraphX. The graph here does not mean charts, lines or bar graphs, but these are graphs in computer sciences like social networks which consist of vertices where each vertex consists of an individual user in the social network and there are many users connected to each other by edges. These edges represent the relationship between the users in the network.

**Lazy evaluation:**

Spark's lazy evaluation is a key optimization strategy that postpones data processing until an action is invoked. This approach allows Spark to optimize the execution plan and efficiently process large-scale distributed data while minimizing unnecessary computations and data movement.