Apache Spark for Beginners

Basic Guide

APACHE Spark™

TABLE OF CONTENT

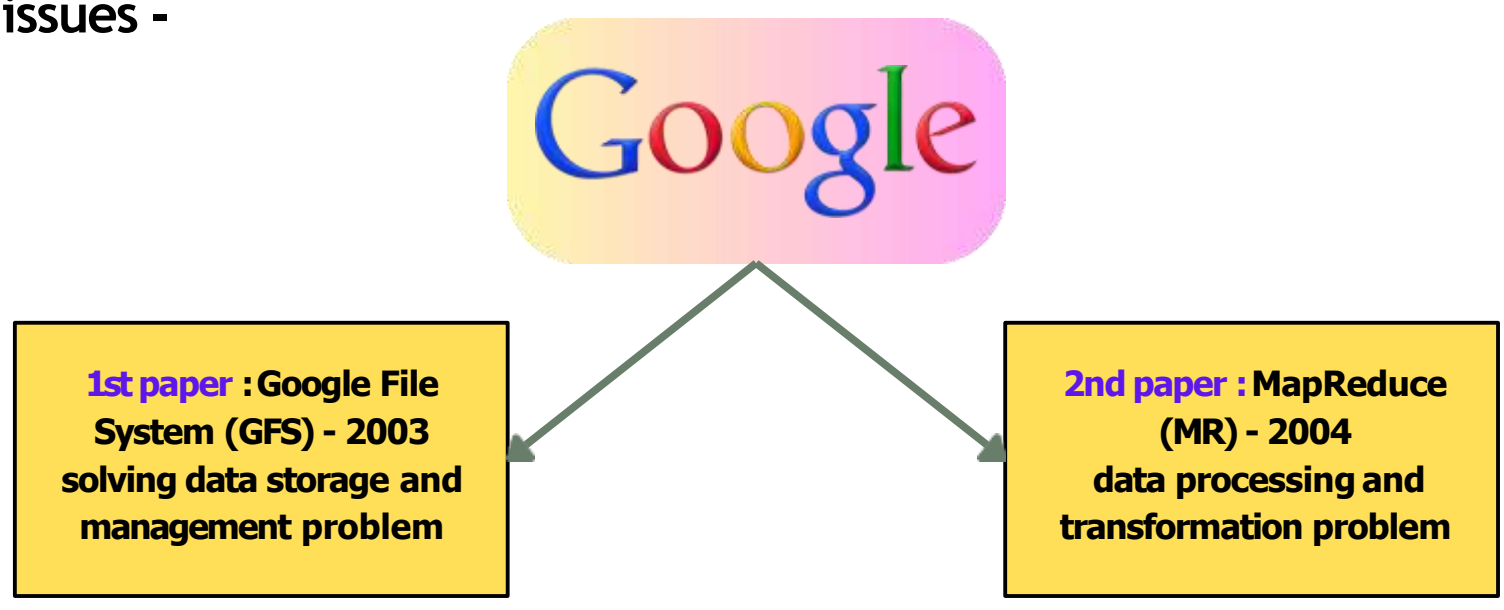
The Genesis of Spark

Increased consumer traFFic, a variety oF new Forms oF data and greater computations demanded the need For more storage and better perFormance. Traditional data storage methods including relational database management systems (RDBMSs) and imperative programming techniques were unable to handle the enormous amounts oF data and their processing.

Google is the First to overcome below problems-

- Data collection and ingestion
- Data storage and management
- Data processing and transFormation
- Data access and retrieval

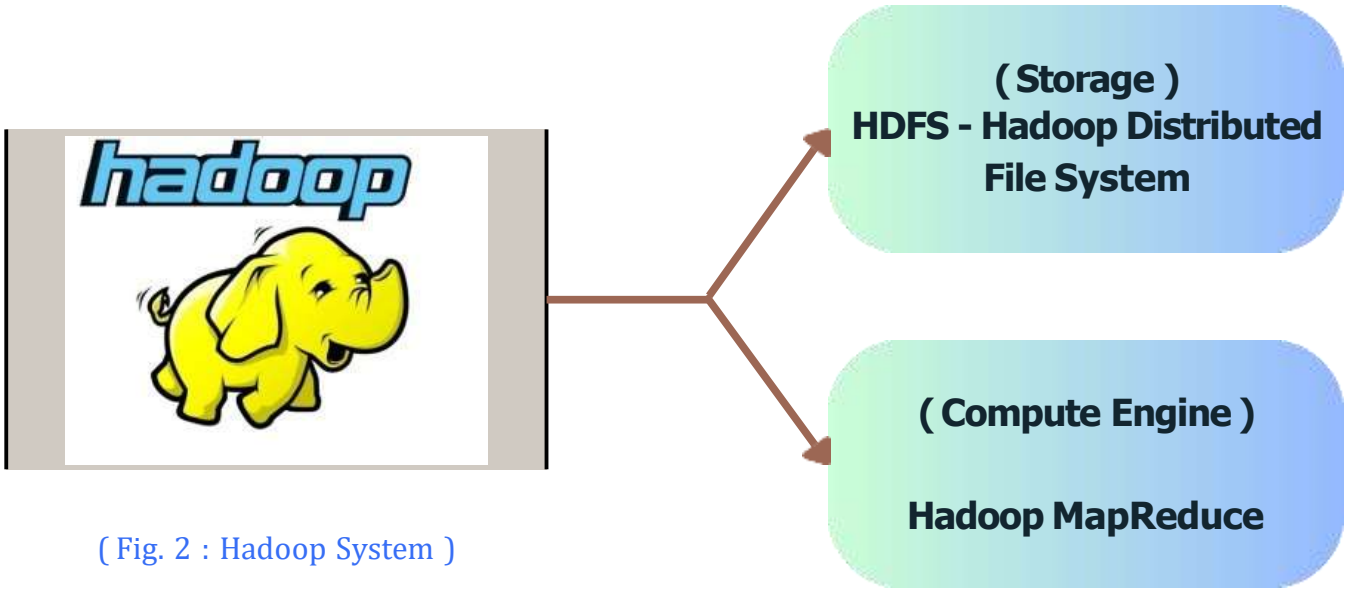
Google published the white papers in a sequence to solve these issues -



(Fig. 1 : Google WThite papers)

Hadoop

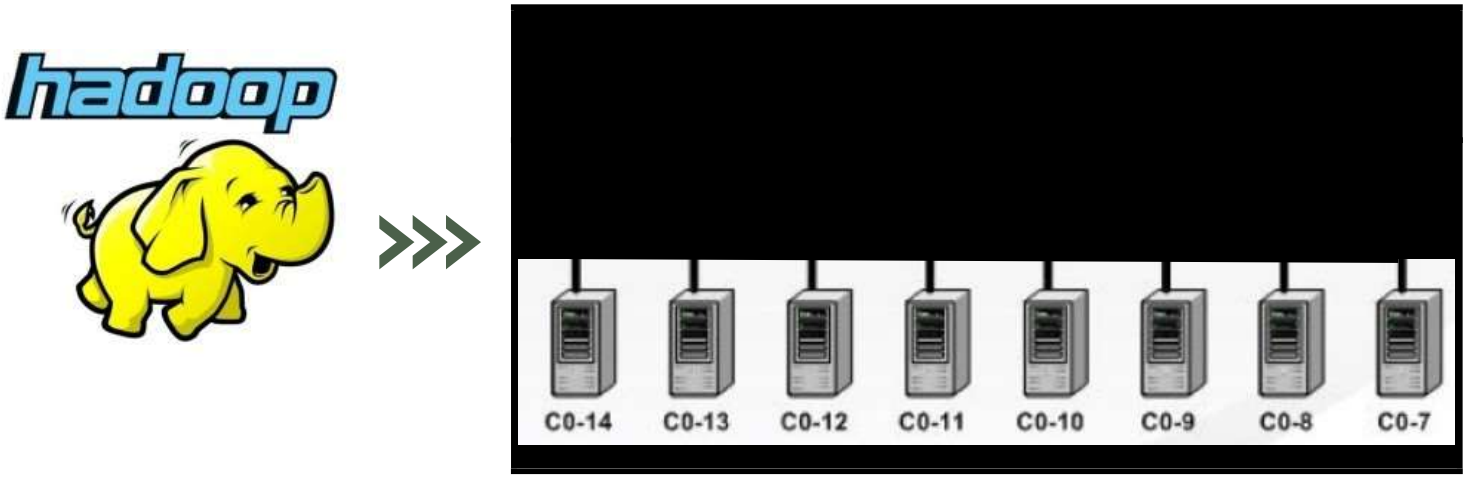
The Google white papers were highly appreciated by the open source community and served as the inspiration For the design and development oF a comparable open source implementation, called Hadoop.



(Fig. 2 : Hadoop System)

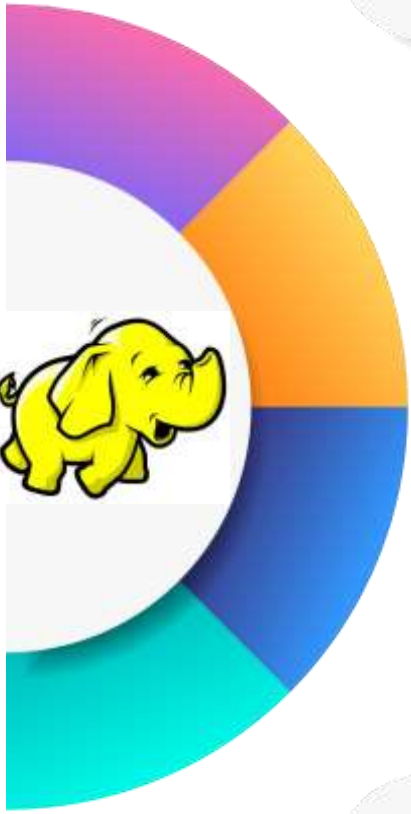
- Hadoop is an open-source soFtware Framework For storing and processing large amounts oF data in a distributed computing environment.
- It is designed to handle big data and is based on the MapReduce programming model, which allows For the parallel processing oF large datasets.
- Its Framework is based on Java programming.
- It Facilitate to start with the small clusters and expand the size as you grow.
- It allows the storage capacity oF 100's to 1000's oF computers and use it as uniFied storage system

Basic Overview :



(Fig. 3 : Hadoop basic working)

HDFS : It allowed us to Form cluster oF computers & used combined capacity For storing our data



MR: Solved compute capacity problem by implementing distributed capacity Framework over HDFS

Hadoop allowed us to break data processing jobs into smaller tasks & use the clusters to Finish the individual task.

It combines the output oF diFFerent task and produce single consolidated output.

Hive :

- Many solutions have been developed over Hadoop platForm by various organizations.
- Some oF the widely adopted systems were Hive , Pig & HBase.
- Apache Hive is the most popular adopted component oF Hadoop.

Hive oFFered Following core capabilities on Hadoop platForm -

1. Create
   - Databases
   - Tables
   - Views
2. Run SQL Queries

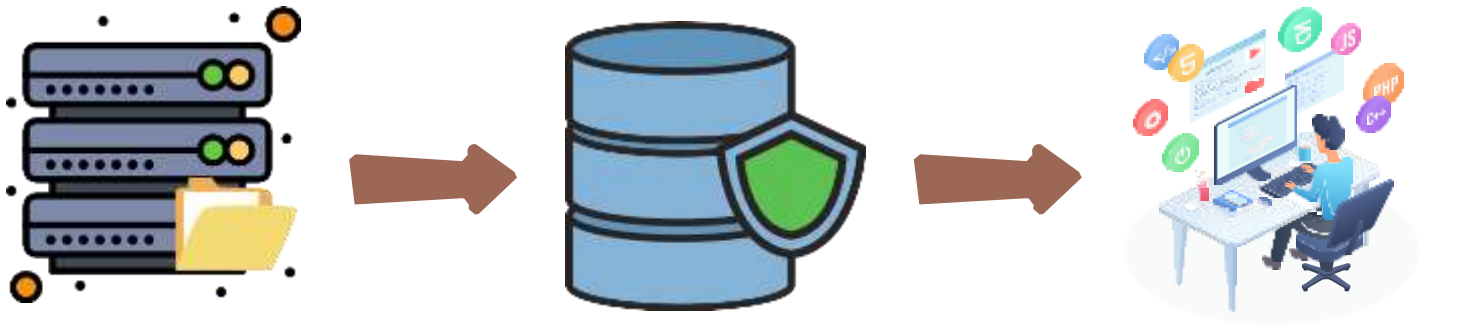
Bringing together , Hadoop as platForm and Hive as a database became very popular. But we still had other problems -

- PerFormance - Hive SQL query perForming slower than RDBMS SQL query
- Ease oF Development - writing MapReduce program was diFFicult
- Language Support - MapReduce was only available in JAVA
- Storage - expensive than cloud storage
- Resource Management - only YARN container support , unable to use other container like Mesos, Docker , Kubernetes , etc

The point is, Hadoop left a lot scope for improvement and as a result Apache Spark came into the existence...!

Understanding Data Lake Landscape

BeFore HDFS & MapReduce, we had Data warehouses (like Teradata, Exadata) where the data is brought From many OLTP/OLAP systems.



(Fig. 4 : Basic DW flow)

The challenges Faced by data warehouses are as Follows -

- Vertical Scaling - adding more DW was expensive
- Large Capital Investment
- Storage - non scalable
- Support only structured data

To overcome above challenges , Data Lake came into the picture with Following Features -

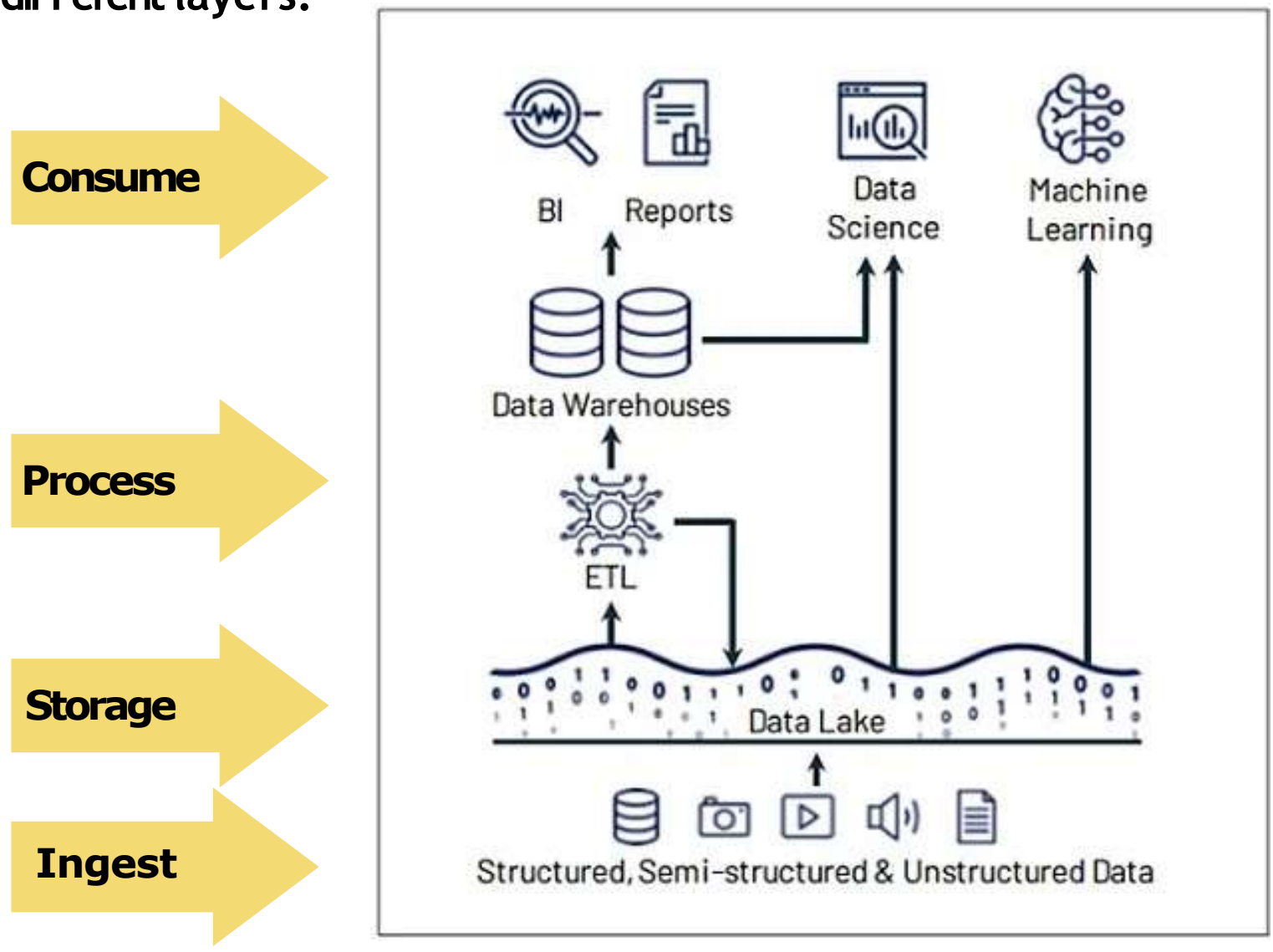
- Horizontal Scaling - adding more cheap servers to clusters
- Low Capital Investment
- Storage - scalable (cloud storage)
- Support structured , unstructured and semi-structured data

Data Lake Architecture :

The core capability oF data lake was storage but with timely advancement, it had developed 4 important capabilities -

- Data collection and ingestion (Ingest)
- Data storage and management (Storage)
- Data processing and transFormation (Process)
- Data access and retrieval (Consume)

Let's analyze the below Data Lake architecture For understanding diFFerent layers.



(Fig. 5 : Data Lake ArcThitecture)

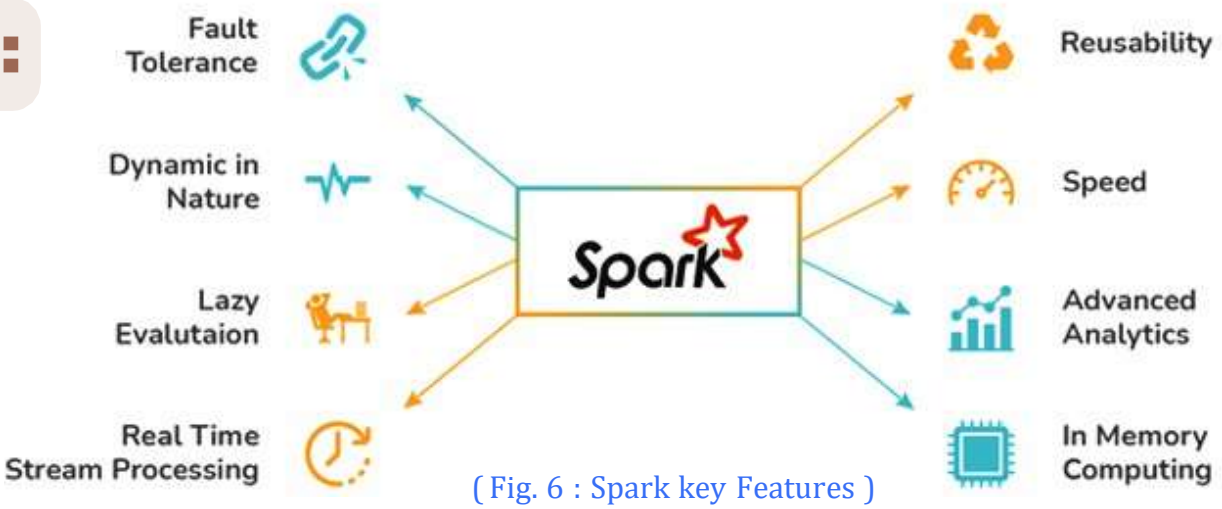
Apache Spark G it's Eco-system

- **Apache Spark is a** unified analytics engine **For large-scale distributed data processing and machine learning.**
- **It is an open-source** cluster computing framework **which handles both batch data & streaming data.**
- **Spark was built on the top oF the Hadoop MapReduce.**
- **Spark provides** in-memory storage **For intermediate computations whereas alternative approaches like Hadoop's MapReduce writes data to and From computer hard drives. So, Spark process the data much Faster than other alternatives.**

History of Spark :

The Spark was initiated by Matei Zaharia at UC Berkeley's AMPLab in 2009. It was open sourced in 2010 under a BSD license. In 2013, the project was acquired by Apache SoFtware Foundation. In 2014, the Spark emerged as a Top-Level Apache Project.

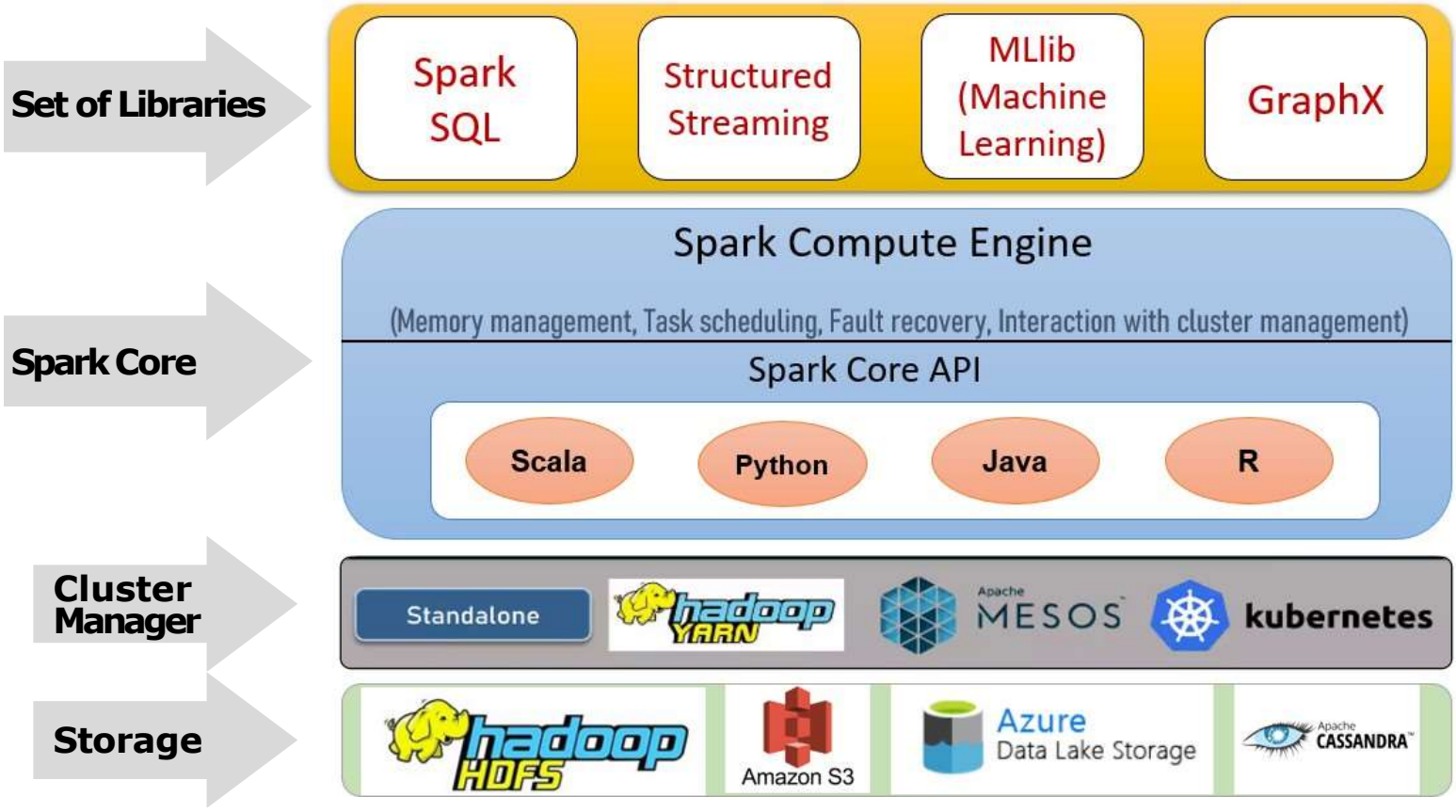
Key Features :



(Fig. 6 : Spark key Features)

Spark eco-system and it's components :

The Spark project is made up oF a variety oF closely coupled components. Spark is a computational engine at its core that can distribute, schedule, and monitor several applications.



(Fig. 7 : Spark Eco-system)

The Apache Spark ecosystem may be divided into three tiers as indicated in the above diagram.

- Storage and Cluster Manager
- Spark Core
- Set oF Libraries

1) Storage and Cluster Manager :

- Apache Spark is a distributed processing engine. However, it doesn't come with an inbuilt cluster resource manager and a distributed storage system.
- There is a good reason behind that design decision. Apache Spark tried to decouple the Functionality oF a cluster resource manager, distributed storage and a distributed computing engine From the beginning.
- This design allows us to use Apache Spark with any compatible cluster manager and storage solution. Hence, the storage and the cluster manager are part oF the ecosystem however they are not part oF Apache Spark.
- You can plugin a cluster manager and a storage system oF your choice. There are multiple alternatives. You can use Apache YARN, Mesos, and even Kubernetes as a cluster manager For Apache Spark. Similarly, For the storage system, you can use HDFS, Amazon S3, Azure Data Lake, Google Cloud storage, Cassandra File system and many others.

2) Spark Core :

Apache Spark core contains two main components -

1) Spark Compute engine
2) Spark Core APIs

Spark Compute engine -

- The Spark Core includes a computation engine For Spark. Basic Functions including memory management, job scheduling, Fault recovery, and most crucially, communication with the cluster manager and storage system, are provided by the compute engine.

- So, in order to give the user a smooth experience, the Spark compute engine manages and executes our Spark jobs. Simply submit your job to Spark, and the core oF Spark does the rest.

Spark Core APIs -

The second part oF Spark Core is core API. Spark core consists oF two types oF APIs.

a.Structured API

b.Unstructured API

- The Structured APIs consists oF data Frames and data sets. They are designed and optimized to work with structured data.
- The Unstructured APIs are the lower level APIs including RDDs, Accumulators and Broadcast variables. These core APIs are available in Scala, Python, Java, and R.

3) Set of Libraries :

- Apache Spark has diFFerent set oF libraries and packages that make it a powerFul big data processing Framework. The set oF libraries include Spark SQL, Spark Streaming, Spark MLlib, and Spark GraphX.
- These libraries provide diFFerent Functionalities For data processing, analysis, and machine learning tasks.

Working :

  - Spark SQL - Allows you to use SQL queries For structured data processing.
  - Spark Streaming - Helps you to consume and process continuous data streams.
  - MLlib - A machine learning library that delivers high-quality algorithms.
  - GraphX - Comes with a library oF typical graph algorithms.
- These libraries oFFer us APIs, DSLs, and algorithms in multiple languages. They directly depend on Spark Core APIs to achieve distributed processing.

Spark Architecture G Execution model

- Apache Spark works in a master-slave architecture where the master is called “Driver” and slaves are called “Workers".
- Master manages, maintains, and monitors the slaves while slaves are the actual workers who perForm the processing tasks. You tell the master what wants to be done and the master will take care oF the rest. It will complete the task, using its slaves.
- Apache Spark Architecture is based on two main abstractions :
  - Resilient Distributed Dataset (RDD)
  - Directed Acyclic Graph (DAG)

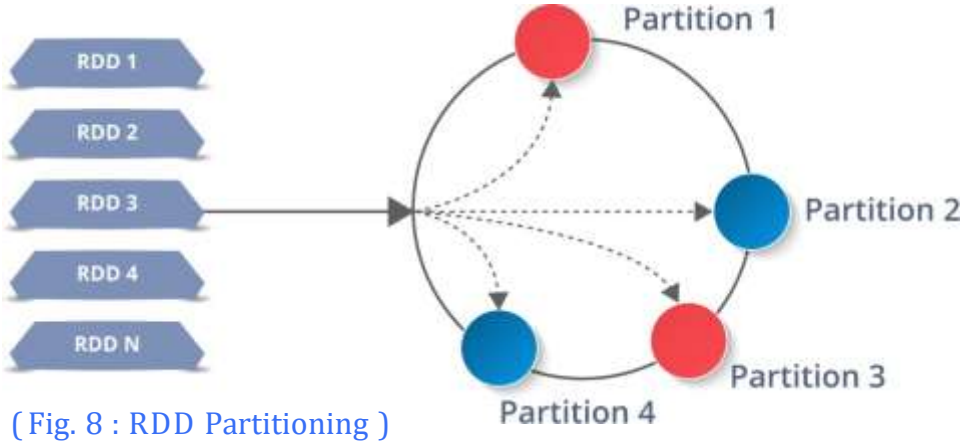
Resilient Distributed Dataset (RDD) :

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

- When you load the data into a Spark application, it creates an RDD which stores the loaded data.
- RDD is immutable, meaning that it cannot be modiFiedonce created, but it can be transFormed at any time.

- The data in an RDD is split into chunks based on a key and those chunks oF data is distributed across the cluster.



(Fig. 8 : RDD Partitioning)

- Every Dataset in RDD is divided into multiple logical partitions, which may be computed on diFFerent nodes oF the cluster. This distribution is done by Spark, so users don't have to worry about computing the right distribution.
- With RDDs, you can perForm two types oF operations :

1) Transformations :

They are the operations that are applied to create a new RDD. This Function is used to transForm the data into new RDD without altering the original data.

e.g. groupBy, sum, rename, sort,etc

2) Actions :

They are applied on an RDD to instruct Apache Spark to apply computation and send the result back to the driver.

e.g. write, collect, show,etc

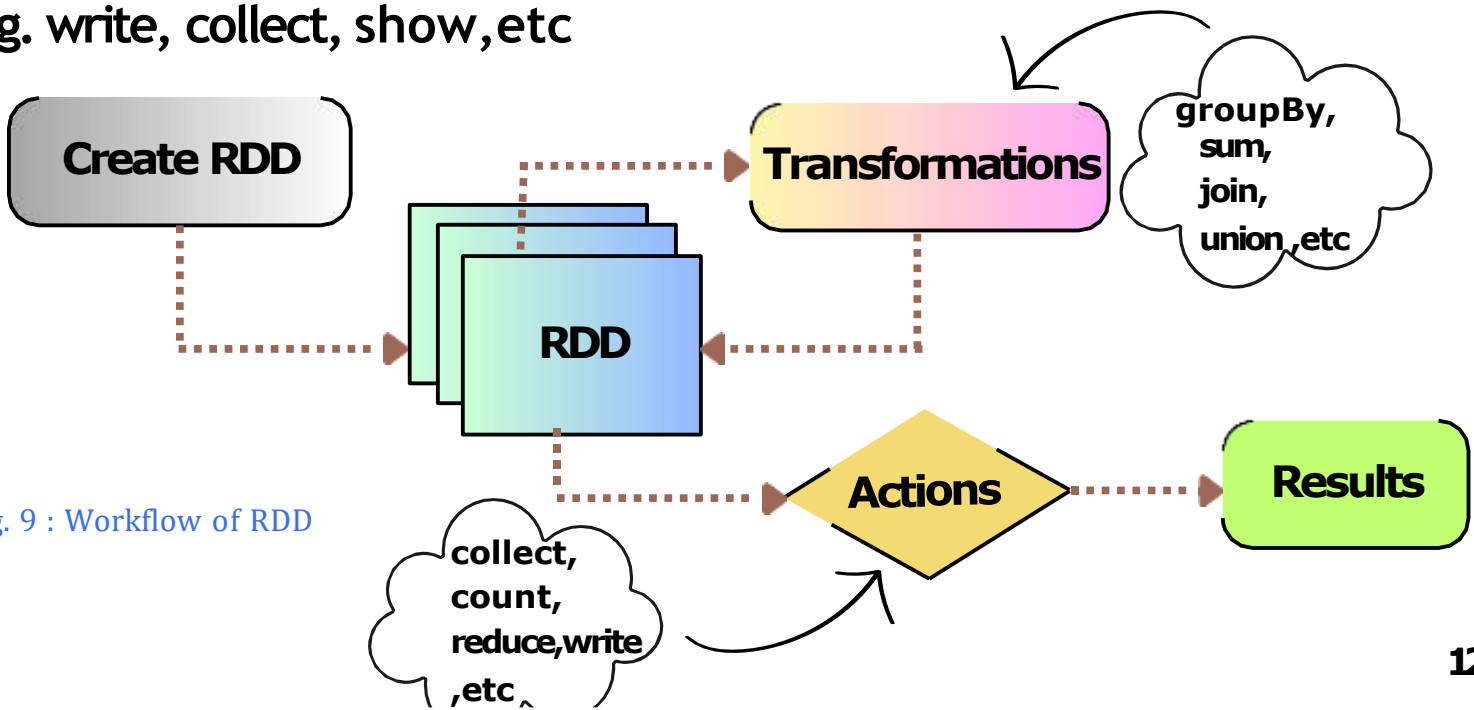


Fig. 9 : Workflow of RDD

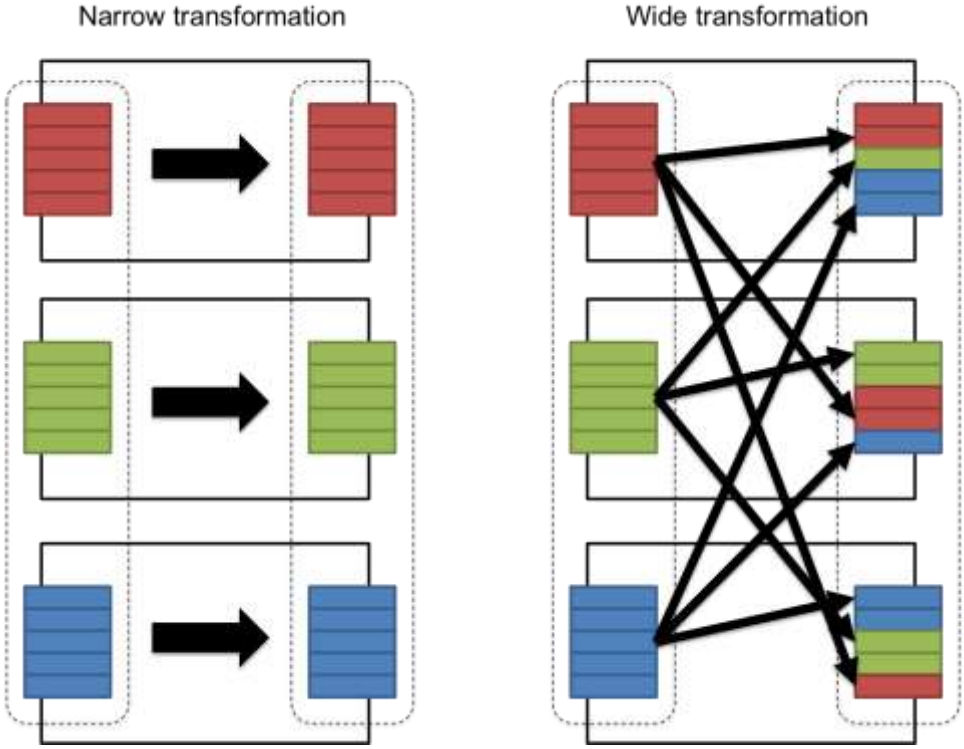
Types of transformations :

1) Narrow transformations -

- Narrow transFormations are those For which each input partition will contribute to only one output partition.
- These transFormations are perFormed on individual partition oF data in an RDD in parallel.
- Since they do not require shuFFling oF data between partitions, their perFormance is more eFFicient than wide transFormation .
- Narrow transFormations can be perFormed on various data Formats, such as unstructured, structured, and semi-structured data.

2) Wide transformations -

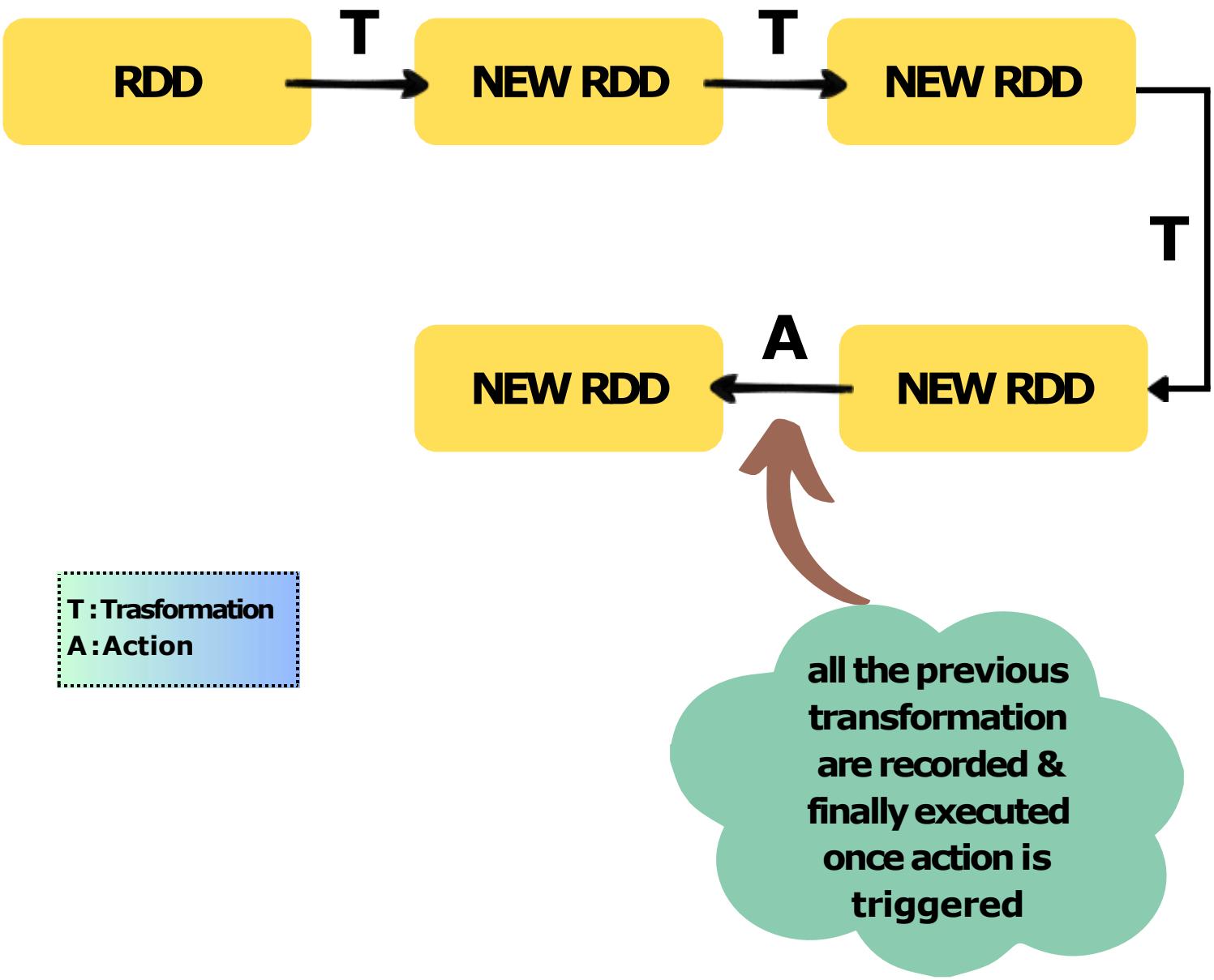
- Wide transFormation will have input partitions contributing to many output partitions.
- These transFormations require shuFFling data between partitions.
- They are typically more complex and require more resources to perForm.
- Wide transFormations, however, typically require structured data in order to perForm the necessary operations.



(Fig. 10 : Narrow & wide Transformation)

Lazy Evaluation :

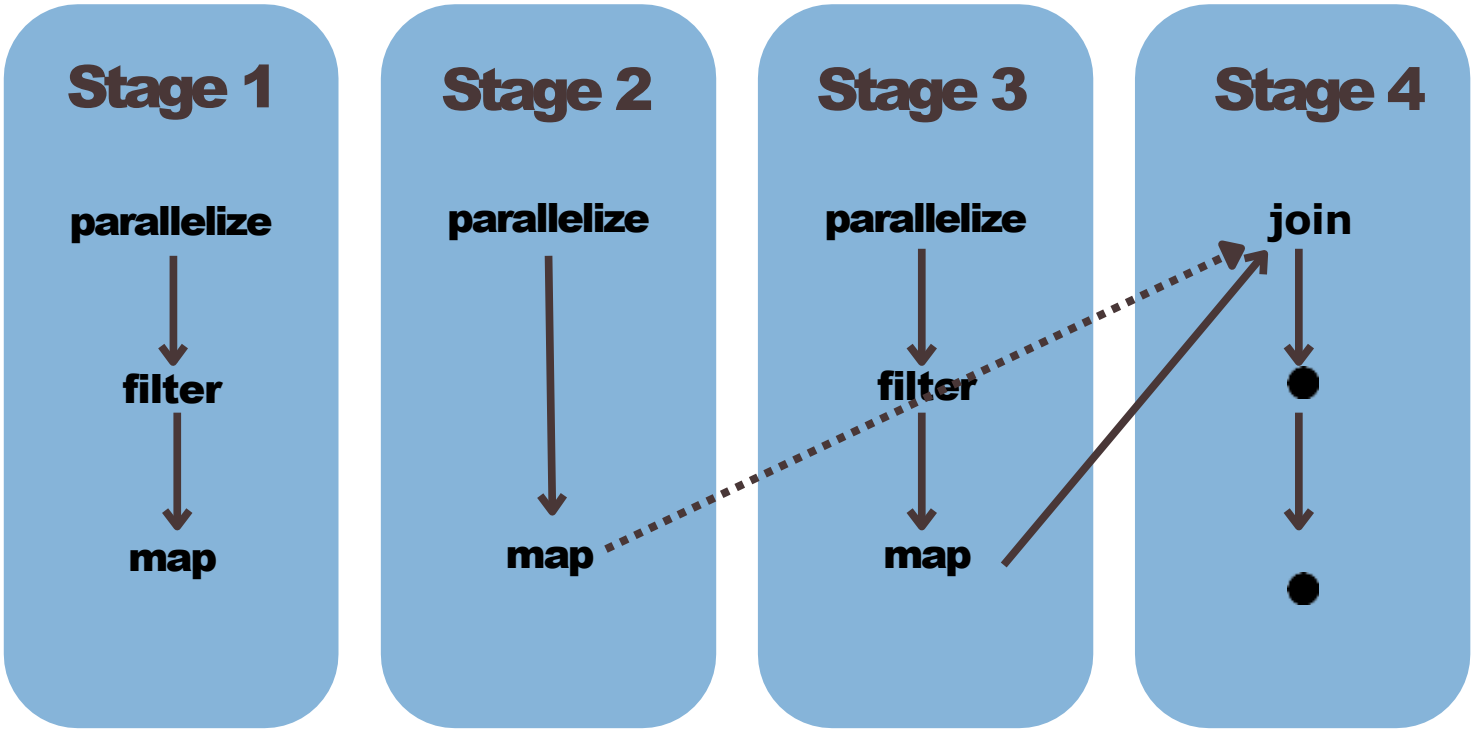
- Lazy Evaluation in Sparks means Spark will not start the execution oF the process until an ACTION is called.
- Spark is not too concernedas long as we are just perForming transFormations on the RDD, dataFrame or dataset.
- Once Spark notices an ACTION being called, it starts looking at all the transFormations, creates a DAG and execute it.



(Fig. 11 : Lazy Evaluation)

Directed Acyclic Graph (DAG) :

- Directed Acyclic Graph is a Finite direct graph that perForms a sequence oF computations on data.
- The DAG in Spark supports cyclic data Flow. Every Spark job creates a DAG oF task stages that will be executed on the cluster.
- Spark DAGs can contain many stages, unlike the Hadoop MapReduce which has only two predeFined stages.
- In a Spark DAG, there are consecutive computation stages that optimize the execution plan.
- You can achieve Fault-tolerance in Spark with DAG.



Spark DAG Visualisation

(Fig. 12 : DAG visualization)

Components of Spark Application Architecture :

Spark Application :

- Spark application is a program built with Spark APIs and runs in a Spark compatible cluster/environment. It can be a PySpark script, a Java application, a Scala application, a SparkSession started by spark-shell or spark-sql command,etc.
- It consists oF a driver container and executors.

SparkSession / SparkContext :

- An object that provides a point oF entry to interact with underlying Spark Functionality and allows programming Spark with its APIs.
- It represents the connection to the Spark cluster. This class is how you communicate with some oF Spark's lower-level APIs, such as RDDs.

Job

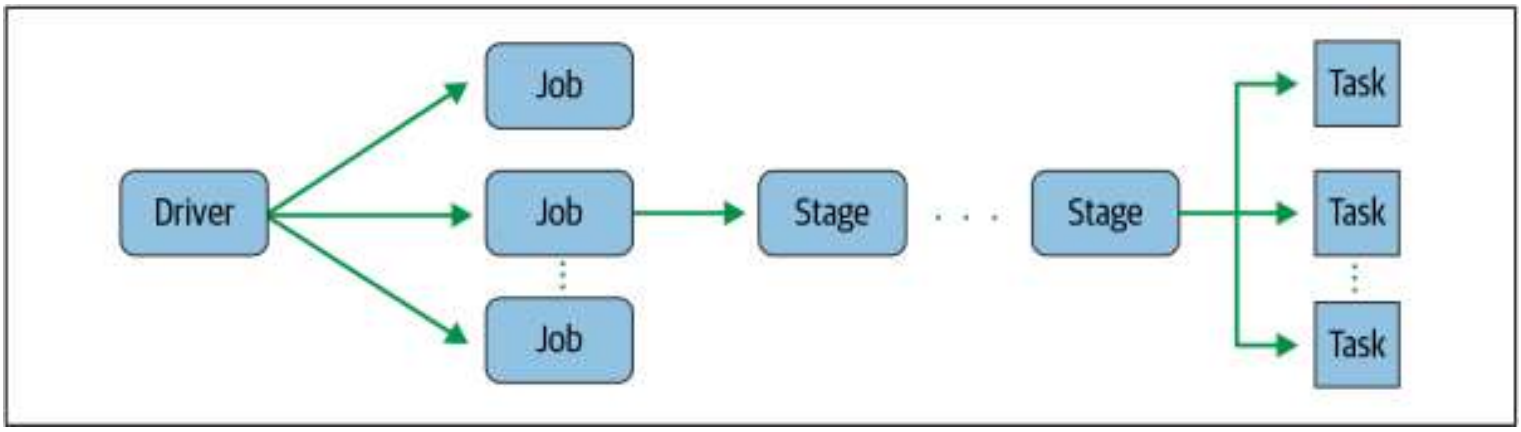
- A parallel computation consisting oF multiple tasks that gets produced in response to a Spark action (e.g., save(), collect()).

Stage

- Each job gets divided into smaller sets oF tasks called stages that depend on each other.
- Stages in Spark representgroups oF tasks that can be executed together to compute the same operation on multiple machines.

Task

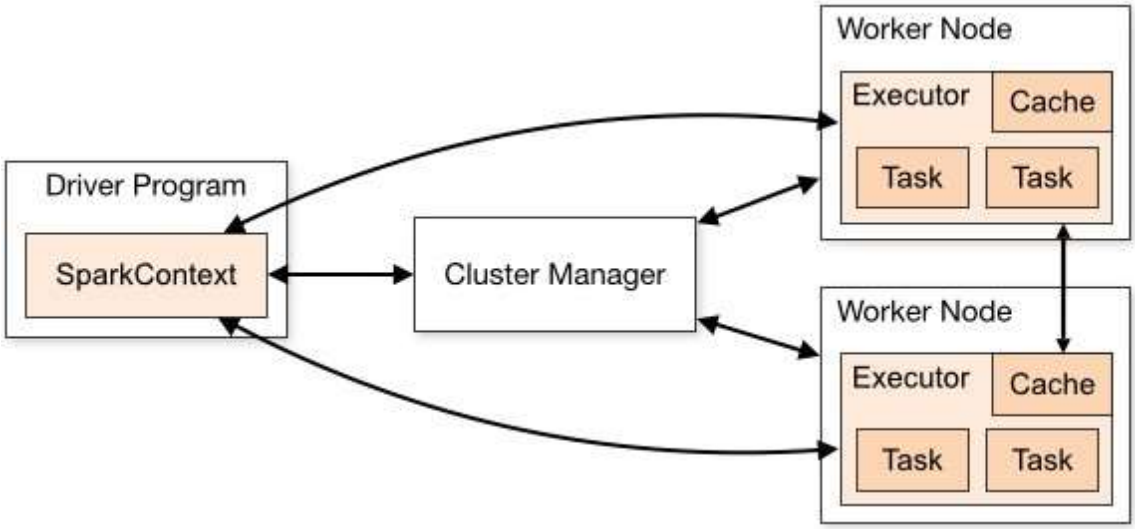
- A single unit oF work or execution that runs in a Spark executor. Each stage contains one or multiple tasks.
- Each task maps to a single core and works on a single partition oF data



(Fig. 13 : Jobs, stages & tasks distribution)

Driver :

- The Driver is the main program that runs on the master node and is responsible For coordinating the entire Spark application. The Driver is responsible For several tasks, including :
  - Managing the SparkContext: The Driver is responsible For creating and managing the SparkContext, which is the main entry point For a Spark application.
  - Breaking down the application into tasks: The Driver is responsible For breaking down the application into a set oF tasks that can be executed in parallel on the worker nodes.
  - Scheduling tasks: The Driver is responsible For scheduling tasks to worker nodes, based on the available resources and the requirements oF tasks.
  - Monitoring tasks: The Driver is responsible For monitoring the tasks and making sure that they are executing correctly. IF a task Fails, the Driver can reschedule it on a diFFerent node.
  - Gathering results: The Driver is responsible For gathering results oF the tasks and combining them to produce the Final result oF the application.
- The Driver is the central component oF a Spark application, and it plays a critical role in ensuring that the application runs correctly and eFFiciently.



(Fig. 14 : Spark Application Diagram)

Executor :

- The Executor is a program that runs on the worker nodes and is responsible For executing the tasks assigned by the Driver. The Executor is responsible For several tasks, including:
  - Running tasks: The Executor is responsible For running the tasks assigned by the Driver. Each Executor runs a set oF tasks, and it can run multiple tasks in parallel, based on the number oF cores available on the node.
  - Reporting status: The Executor is responsible For reporting the status oF the tasks to the Driver. The Driver uses this inFormation to monitor the progress oF the application and make sure that it is executing correctly.
  - Storing intermediate data: The Executor can store intermediate data in memory, which can be used by other tasks. This allows Spark to avoid shuFFling data between nodes, which can be a perFormance bottleneck.
  - Releasing resources: The Executor is responsible For releasing resources when the tasks are completed, which allows Spark to make the most eFFicient use oF the available resources.
- The Executor is a critical component oF a Spark application, and it plays a crucial role in executing the tasks and producing the Final result.

Cluster Manager :

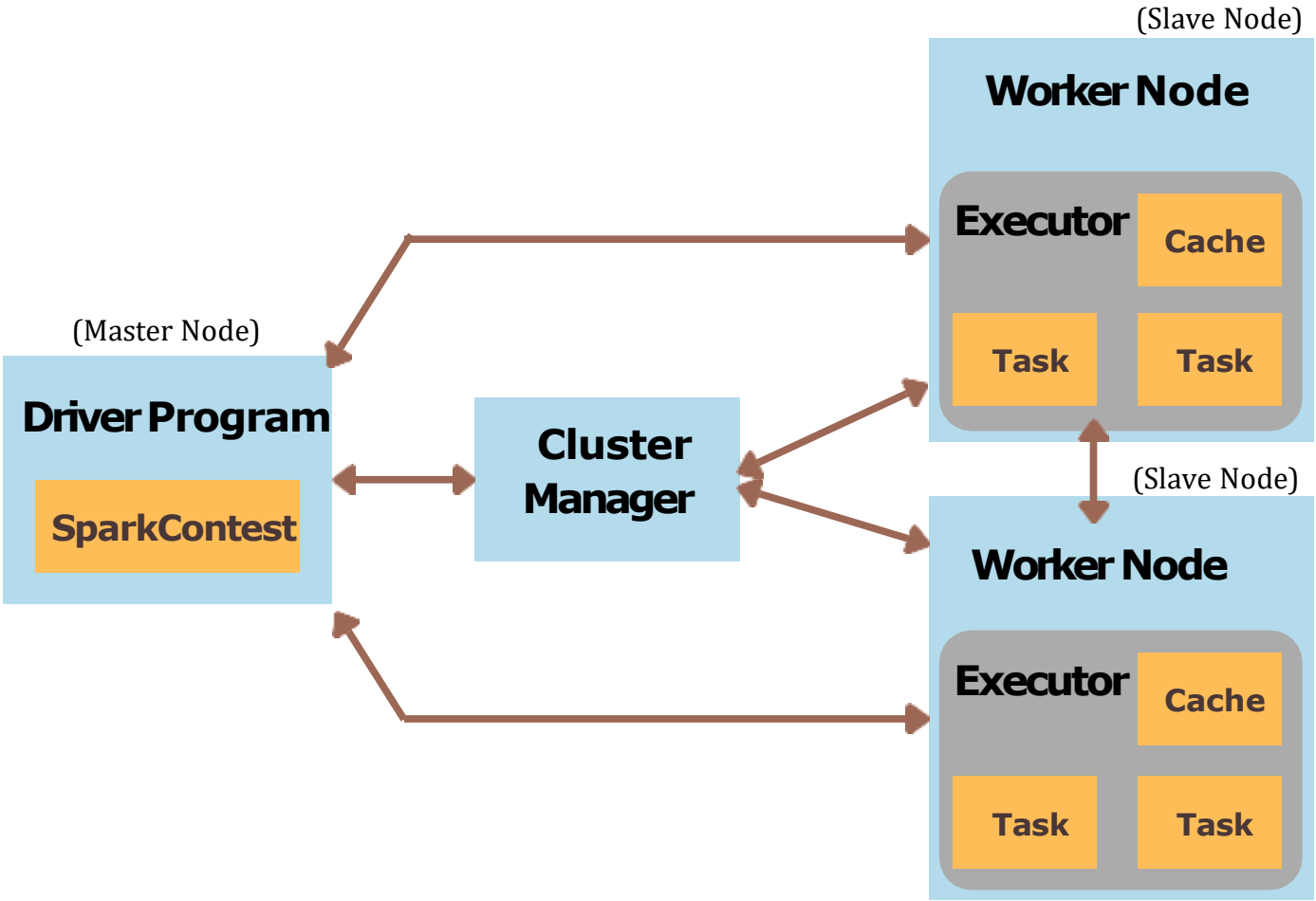
- The cluster manager is responsible For managing the resources required For a Spark application, including CPU, memory, and network resources. Its primary Functions include :
  - Resource allocation: The cluster manager receives resource requests From the Spark driver and allocates the necessary resources, such as CPU cores and memory, to the application.
  - Executor management: The cluster manager launches and manages Spark executors on worker nodes, which are responsible For executing tasks and storing data.
  - Fault tolerance: The cluster manager monitors the health oF the worker nodes and detects Failures, ensuring the smooth execution oF the application by reallocating resources and restarting Failed tasks.
  - Node management: The cluster manager keeps track oF the worker nodes' status and manages their liFecycle, handling node registration, de-registration, and decommissioning.

Types of cluster managers :

- Standalone– a simple cluster manager included with Spark that makes it easy to set up a cluster.
- Apache Mesos– a general cluster manager that can also run Hadoop MapReduce and service applications.
- Hadoop YARN– the resource manager in Hadoop 2.
- Kubernetes– an open-source system For automating deployment, scaling, and management oF containerized applications.

Execution of Spark Application :

- 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.
- The Cluster Manager manages the execution oF various jobs in the cluster. 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.



(Fig. 15 : Execution of Spark Application)

- Whenever an RDD is created in the SparkContext, it can be distributed across many worker nodes and can also be cached there. Worker nodes execute the tasks assigned by the Cluster Manager and return it back to the Spark Context.
- The executor is responsible For the execution oF these tasks. The liFespan oF executors is the same as that oF the Spark Application. We can increase the number oF workers iF we want to improve the perFormance oF the system. In this way, we can divide jobs into more coherent parts.

Spark Execution Modes :

There are 3 types oF execution modes -

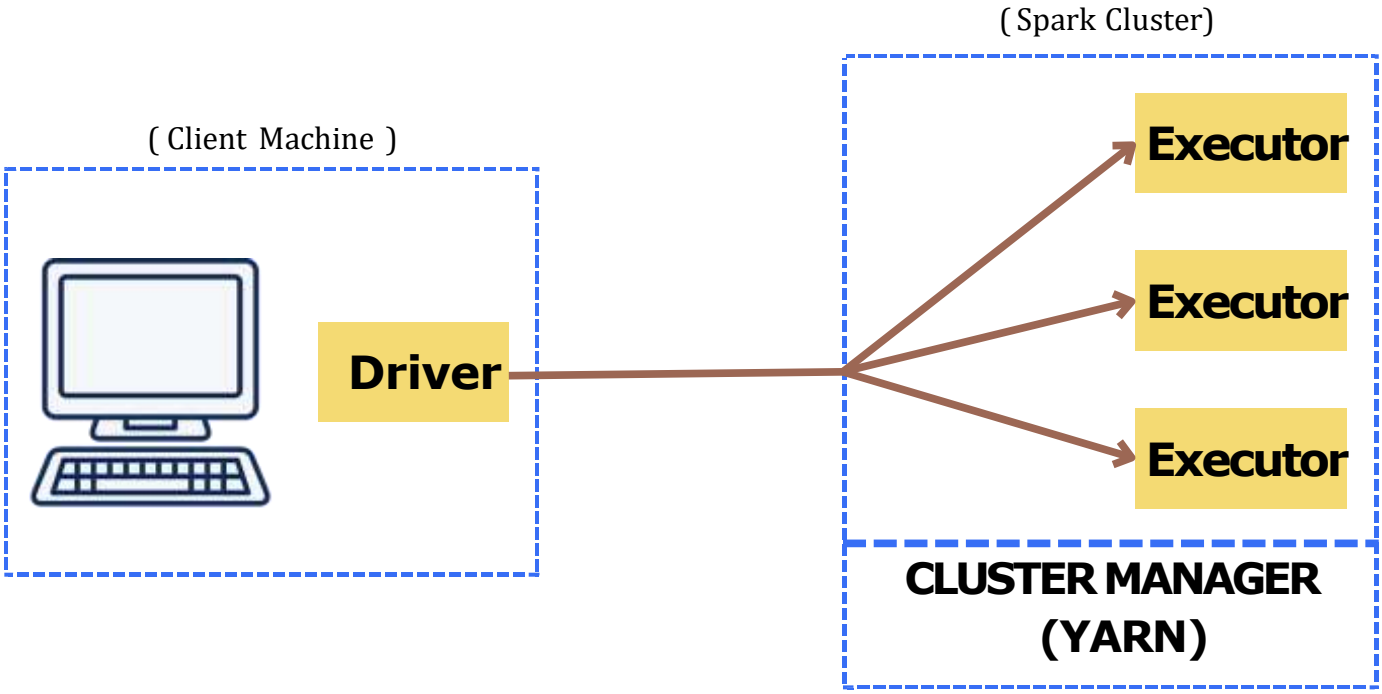
1. Local Mode
2. Client Mode
3. Cluster Mode

1) Local Mode :

- This is similar to executing a program on a single JVM on someone's laptop or desktop.
- It could be a program written in any language, such as Java, Scala or Python.
- However, you should have deFined and used a spark context object in these apps, as well as imported spark libraries and processed data From your local system Files.
- This is the local mode because everything is done locally, there is no concept oF a node, and nothing is done in a distributed manner.
- A single JVM process is used to produce both the driver and the executor.
- For example, launching a spark-shell on your laptop is an example oF a local mode oF execution.

2) Client Mode :

- In client mode, the driver is present on the client machine (laptop/ desktop). i.e., the driver is not part of the cluster. On the other hand, the executors run within the cluster.
- The driver connects to the cluster manager, starts all the executors on the clusters For interactive queries and receives the results back to the client.
- In case of a problem with the client (local) machine or you log off , the driver will go off and subsequently all executors will shut down on the cluster.

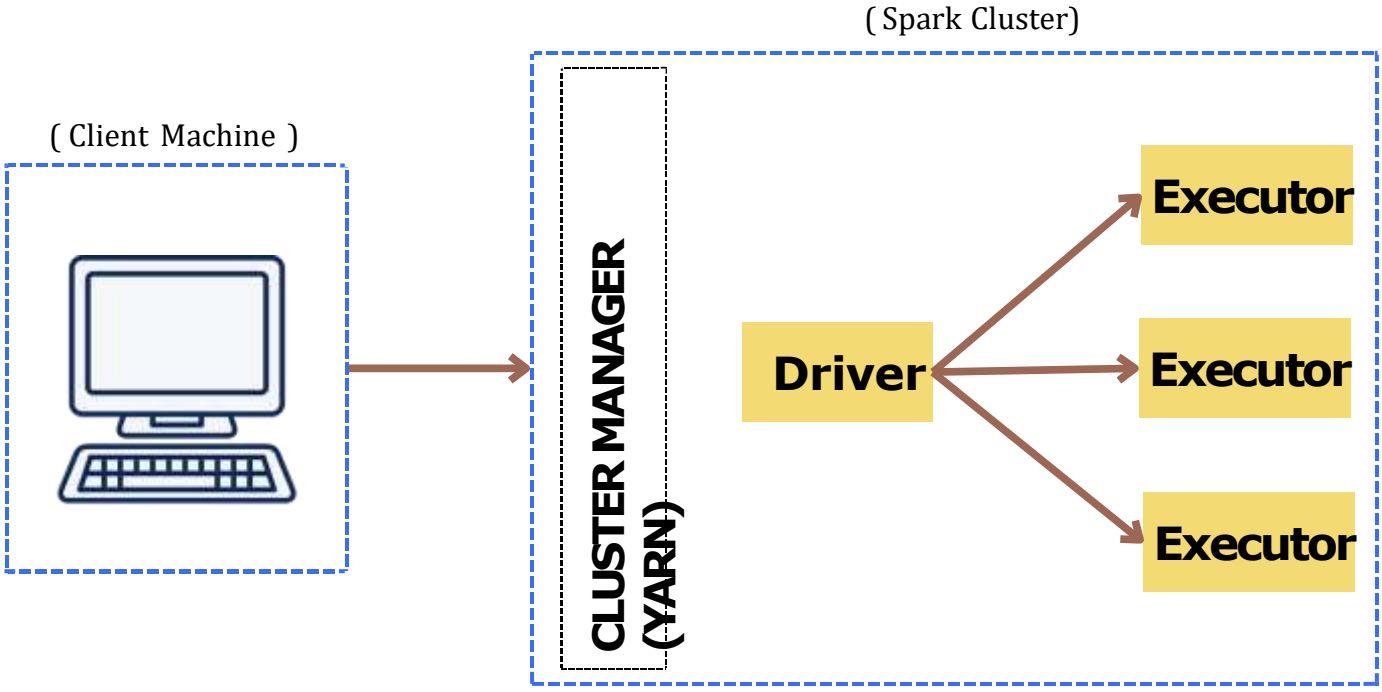


(Fig. 16 : Client Mode Execution)

- So the point is, in this mode, the entire program is dependent on the client (local) machine since the driver is located there.
- This mode is unsuitable For production environments and long running queries. It remains useFul For debugging and testing purposes.

2) Cluster Mode :

- In cluster mode, the driver and executor both run inside the cluster.
- The spark job is submitted From your local machine to a cluster machine within the cluster. Such machines are usually called edge node .
- In case oF a problem with the client (local) machine or you log oFF , the driver will not get impacted as it is running on the cluster.

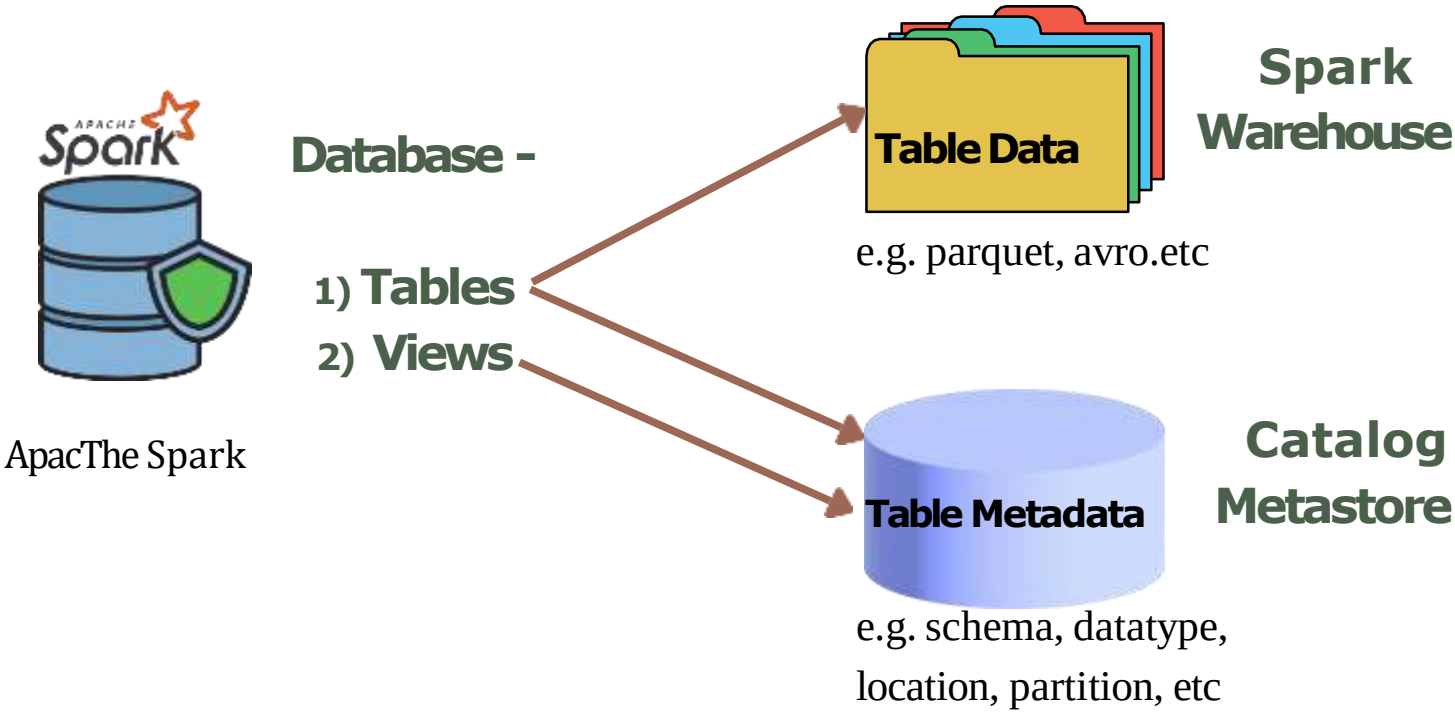


(Fig. 17: Cluster Mode Execution)

- This means that the cluster manager is responsible For maintaining all Spark Application related processes.
- This mode is useFul For long running queries and production environments.

Spark Databases, Tables G Views

- Apache Spark is not only a set oF APIs and processing engine but also it is a database in itselF.
- You can create database in spark. Once you have database, you can create tables and views.



(Fig. 18 : Spark Database)

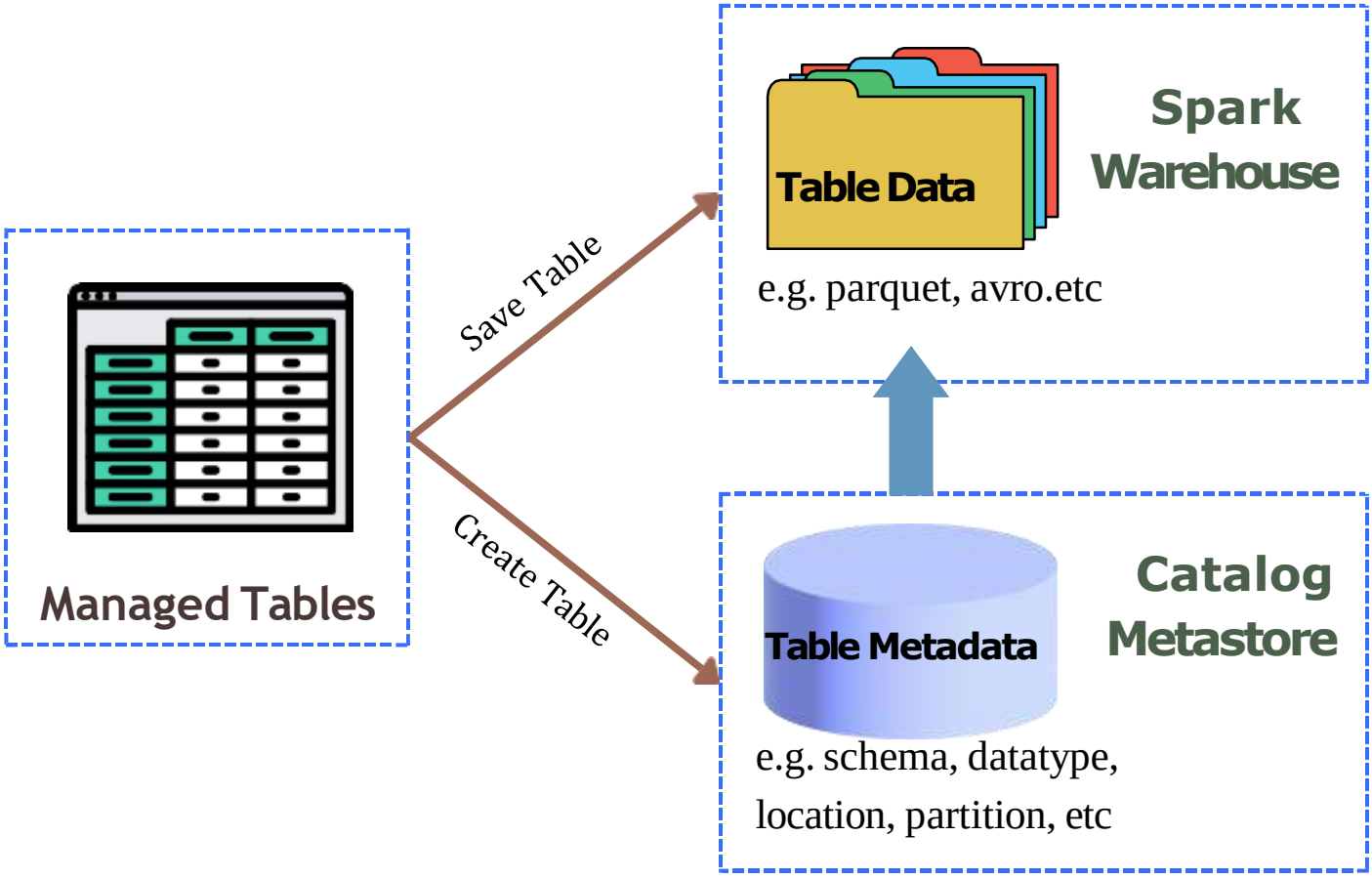
Tables in Spark :

There are 2 types oF SQL tables in Spark -

1. Managed (Internal) tables
2. Unmanaged (External) Tables

1) Managed (Internal) Tables :

- For Managed tables, Spark manages both the data and the metadata.
- The table data is stored in Spark SQL Warehouse directory which is the deFault storage For managed tables.
- Metadata gets stored in a meta-store oF relational entities (including databases, Spark tables, and temporary views).
- F we drop the managed table, Spark will delete both data as well as metadata. AFter dropping tables, we can neither query the table directly nor retrieve data From it.



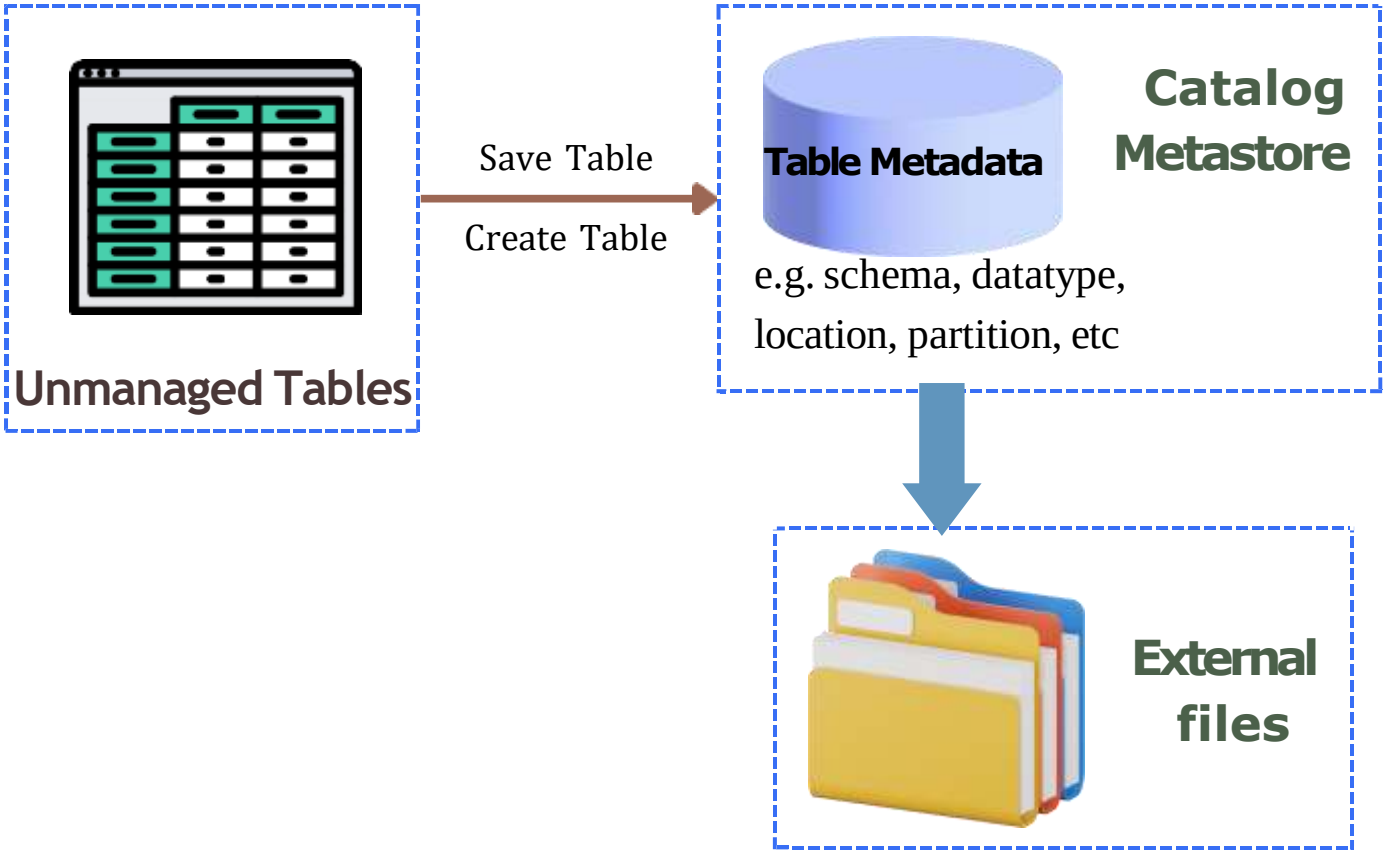
Syntax :

```
CREATE TABLE internal_table
(id INT, FirstName String, LastName String) ;
```

(Fig. 19 : Spark Managed Tables)

2) Unmanaged (External) Tables :

- For External table, Spark manages the metadata and we have Flexibility to store the table data at our preFerred location.
- We need to speciFy the exact location where you wish to store the table or, alternatively, the source directory From which data will be pulled to create a table.
- Metadata gets stored in a meta-store oF relational entities (including databases, Spark tables, and temporary views).
- IF we drop the external table, Spark will delete only metadata but the underlying data remains as it is in its directory



Syntax :

```
CREATE TABLE external_table
(id INT, FirstName String, LastName String)
LOCATION '/user/tmp/external_table';
```

Fig. 20 : Spark Unmanaged Tables

Views in Spark :

- In the context of Apache Spark, "views" typically refer to SQL views or temporary tables that allow you to organize and work with your data using SQL queries.
- Views provide a convenient way to abstract and manipulate your data without modifying the original data source. Here are common types of views in Apache Spark :

1) Global Temporary View (GlobalTempView) :

- A global temporary view is available to all Spark sessions and persists until the Spark application terminates.
- It can be accessed from different Spark sessions running on the same cluster.
- To create a global temporary view, you can use the createOrReplaceGlobalTempView method on a DataFrame or Dataset.

2) Temporary View (TempView) :

- A temporary view is specific to the Spark session that creates it and exists for the duration of that session.
- It cannot be accessed from other Spark sessions.
- To create a temporary view, you can use the createOrReplaceTempView method on a DataFrame or Dataset.

LET'S TAKE A PAUSE...!

By now, you have -

- Learned The Genesis of Spark
- Gained some knowledge on hadoop
- Understood data lake concept
- Learned Apache Spark basics and eco-system
- Grasped the Spark Architecture & Execution model.
- Captured some details about Spark Databases, Tables & Views

Let's get masters on all of the above subjects, and soon we'll meet in the next workbook, "Apache Spark for Intermediate."

Till then... "Keep Learning and keep Exploring...!"