Disclaimer: These slides are copyrighted and strictly for personal use only

- This document is reserved for people enrolled into the Ultimate Big Data Masters Program (Cloud Focused) by Sumit Sir

- **Please do not share this document,** it is intended for personal use and exam preparation only, thank you.

- If you've obtained these slides for free on a website that is not the course's website, please reach out to legal@trendytech.in . Thanks!

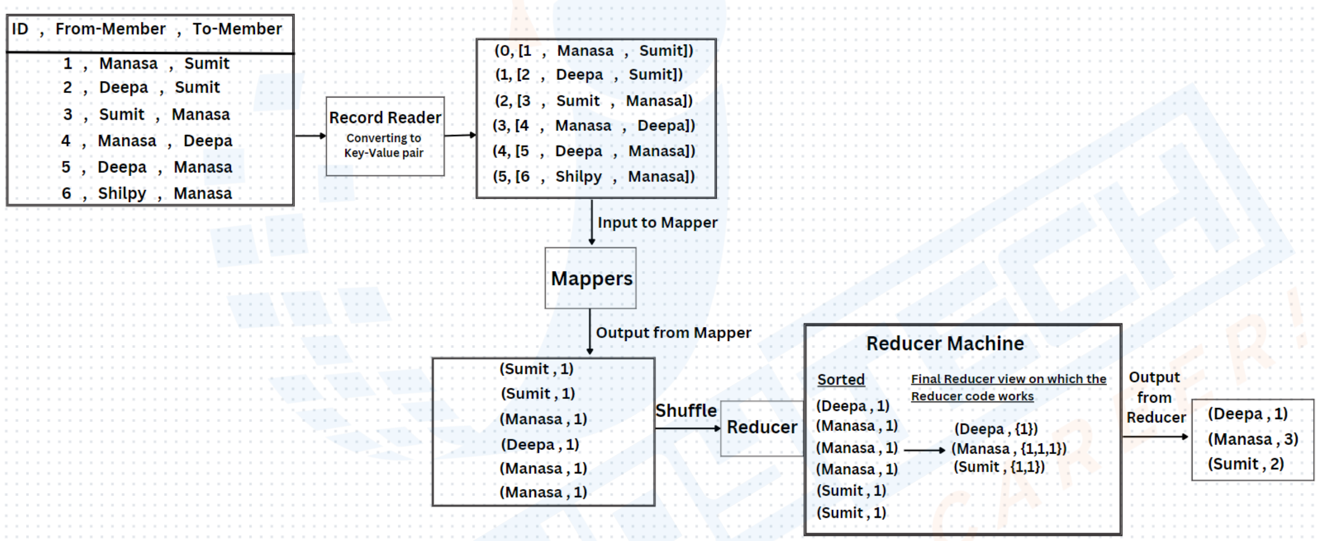
- All the Best and Happy Learning!

Distributed Processing

MapReduce is used to process large amounts of data that is distributed across the cluster

-Mapper gives parallelism
-Reduce is for aggregation

Example :
Problem statement - Calculate the number of views for each of the LinkedIn profiles



- **Number of Mappers = Number of blocks**

- **Number of Mappers Running in Parallel = Number of Data Nodes in the Cluster**

- **No.of Reducers Launched = Can be Configured to a desired number (Default = 1 Reducer)**

Case 1 : When to increase the number of Reducers - To avoid the bottleneck due to a single reducer.

If the reducer has to do a lot of aggregation, then a single reducer might become a bottleneck and we would increase the number of reducers.

Case 2 : When to decrease the number of Reducers to Zero - For the jobs that don't require any Aggregation and Mapper output is the final output. Ex - Filter

When the no.of Reducers is increased to more than one Reducer-

Partitions come into picture!

No.of Partitions = No.of Reducers

The output [Key,Value] from the Mapper with a specific key always goes to a specific Reducer based on a logic as follow -

1. **Default System Defined Hash Function - Ex : Mod (%) function**
2. **Custom Function**

Note : A function is said to be consistent if a given specific key always goes to the same reducer.

The Workflow :

PARTITION - Is for distributing the Data that is in the form of (Key,Value) pairs based on some logic across different Reducers.

SHUFFLE - Is the process of sending the intermediate output from the Mapper machine to the Reducer machine for further processing/aggregation.

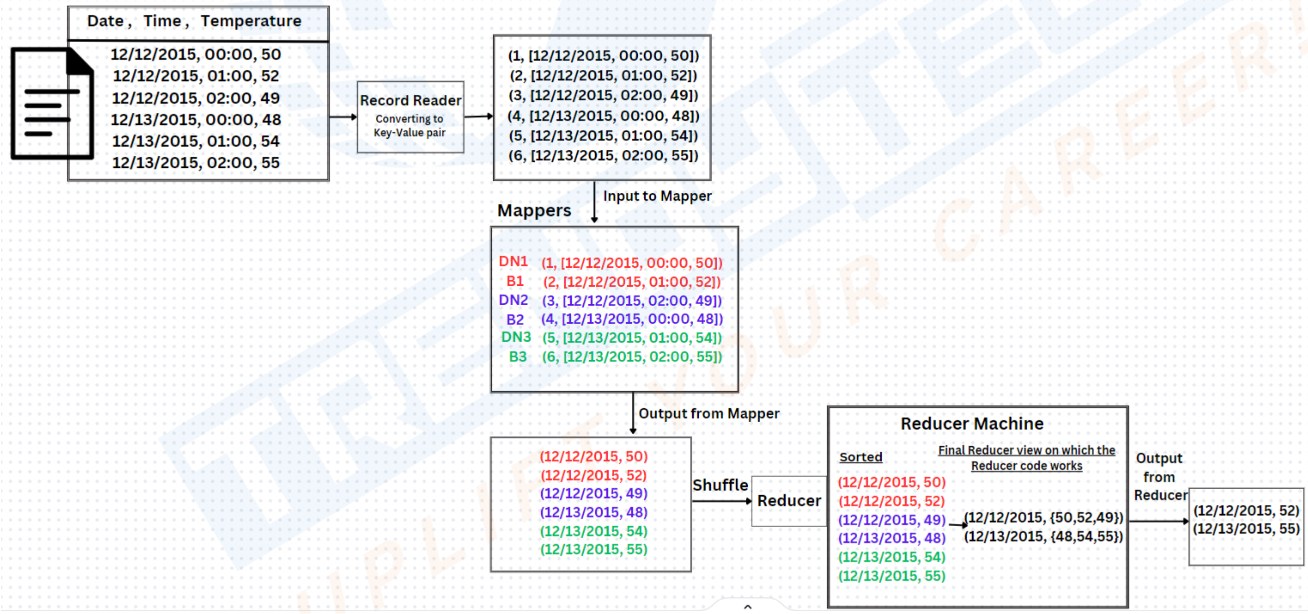
SORT - Is the process of bringing all the same keys together into a group on a Reducer machine.

Example : Sensor Data

Consider a 300MB file of Sensor data that contains temperatures recorded every 1 hour.

With 128 MB as default block Size -> no.of block for 300 MB file = 300 / 128 = 3 Blocks

Problem statement : Is to find the Maximum temperature recorded everyday.

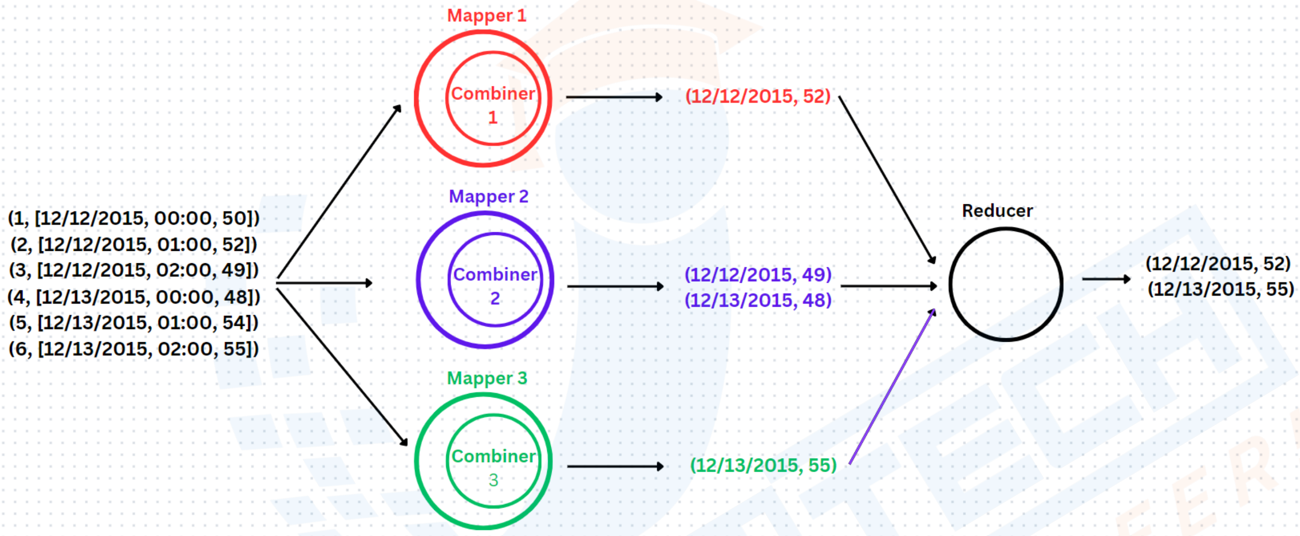


-In the above Scenario, Mapper is doing the bare minimum operations of just removing a column.

-However, in order to improve the performance, we need to have more operations to be performed in parallel i.e., at the Mapper end.

Advantage of introducing Local Aggregation at Mapper - Combiner

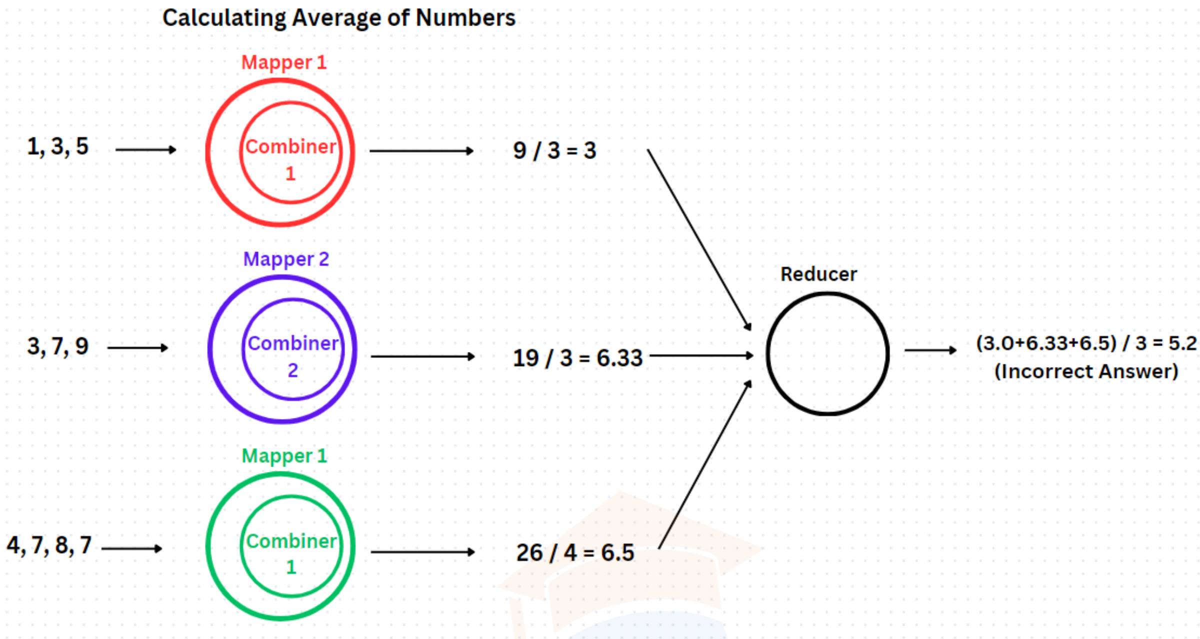
1. Improve Parallelism
2. Reduce Data Transfer



Use Combiners with Caution - When not to use Combiners?

In cases of calculating Maximum, Minimum and Sum - The results will remain the same with or without the use of Combiner - These are the safe scenarios to use combiner as it doesn't change the results. Ideally, combiners are used only to improve the performance.

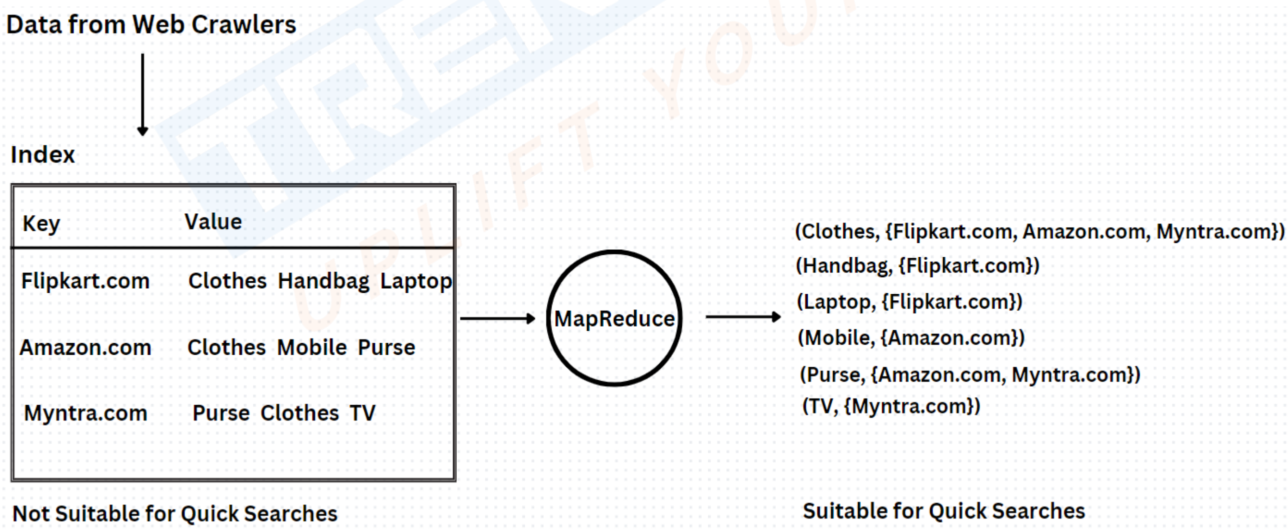
In a case where Average has to be calculated, then usage of Combiners leads to incorrect results. Therefore, we cannot use a combiner in cases where the results are affected (incorrect results)



However, Combiner can still be used in the Average calculation when the output of combiner is in a different format like (sum of numbers, count of numbers) in place of directly providing the average.

Classical Industry Use Case of MapReduce : Google used MapReduce for their Web Search!

A Crawler application crawls the web to build an Index.



MAP - Gives Parallelism

COMBINER - Combine and Aggregate the Mapper output (Local aggregation at Mapper end)

SHUFFLE - Sending the Mapper output to the Reducer for further aggregation.

SORT - Sorting done on the reducer machine, so it appears as a collection for further aggregation.

REDUCER - Produces the Final output after aggregating the Mapper's outputs. (There can be 1 or more Reducers)

Example Programs :

- **mapreduce_prog.jar**

Functionality packaged - Count the frequency of each word

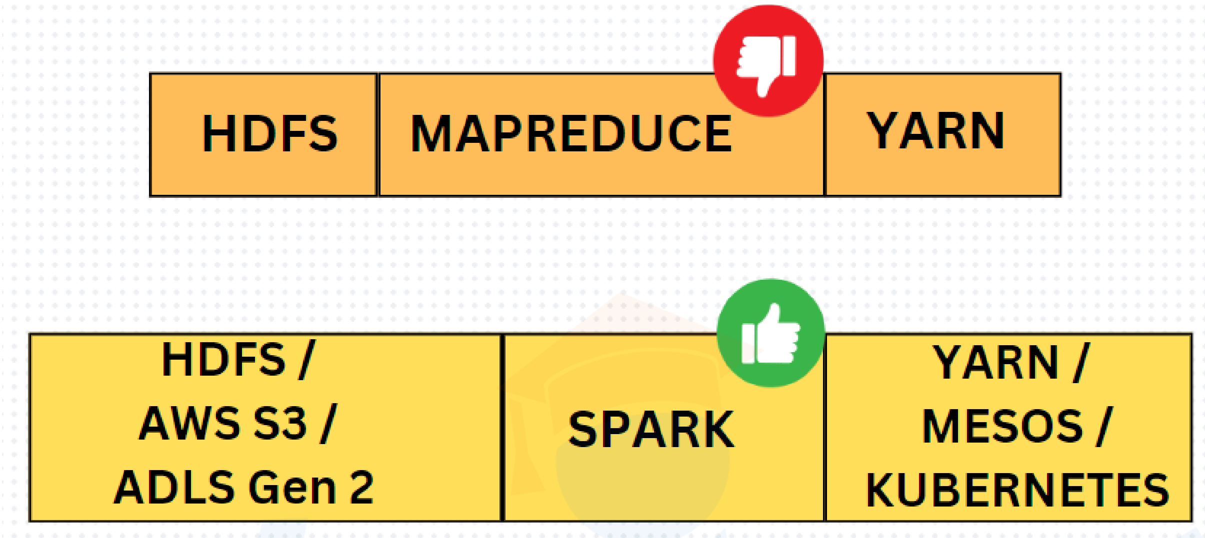
Command to execute the Jar:

> **hadoop jar <jar_name> <input_file_path_in_hdfs> <output_directory_in_hdfs>**
>
> **No.of Splits (Split Size) = No.of Blocks**
>
> After Execution, the output Directory contains 2 types of files -
>
> 1. _SUCCESS file that the job is successful
> 2. part-r-00000 - there could be 1 or more such files, that depends on the no.of reducers. This file contains the actual results.

Other Example Use Cases Considered -

- **With Zero Reducers**
- **With 2 Reducers**
- **With Custom Partitioner**
- **With Combiner**

Apache Spark



Challenges of MapReduce

1. Less Performant due to many IO disk seeks.
2. Need to write many lines of Code to accomplish even a simple task.
3. MapReduce Supports only Batch Processing
4. Learning curve is high
5. Constrained to always think in a Map-Reduce perspective.
6. No Interactive mode

Spark is a Plug and Play compute engine used for Distributed processing.

Spark needs -

1.**Storage** (Could be HDFS or any Data Lake like - ADLS Gen2 | Amazon S3 | Google Cloud Storage)

2.**Resource Manager** (Could be YARN | Mesos | Kubernetes)

Formal Definition :

Apache Spark is a multi language engine for executing data engineering, data science and machine learning on a single node or cluster.

Spark with Python - Pyspark

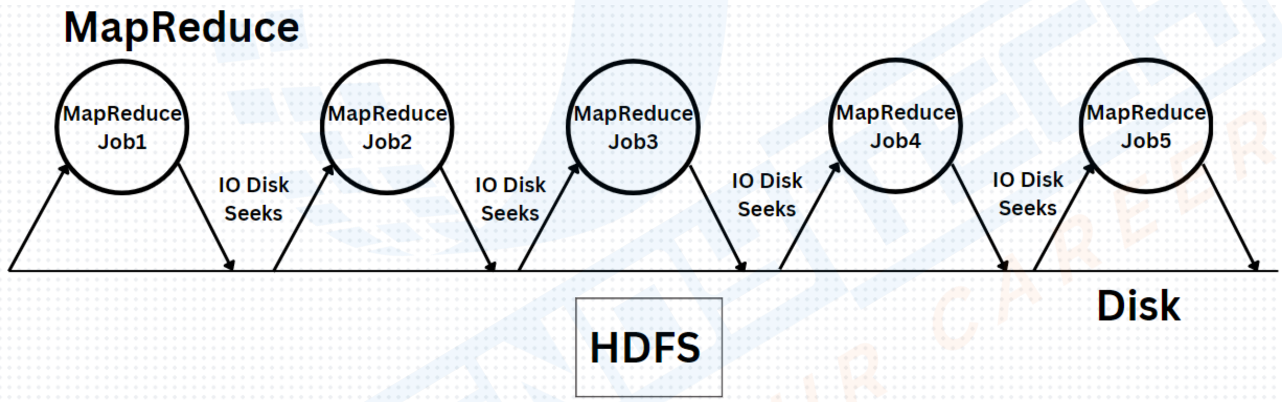
Another Definition of Spark :

Spark is a

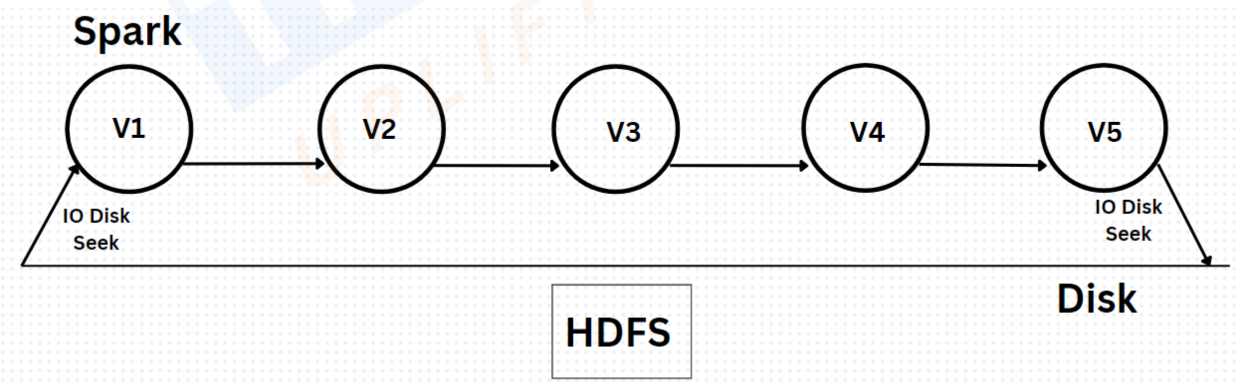
General Purpose | In-memory | Compute Engine

What is meant by In-memory and why Spark has high performance?

In case of MapReduce - Many IO Disk Seeks leads to poor performance.



In case of Spark - Very few IO Disk Seeks as it is an in-memory compute engine, leading to performance gains.



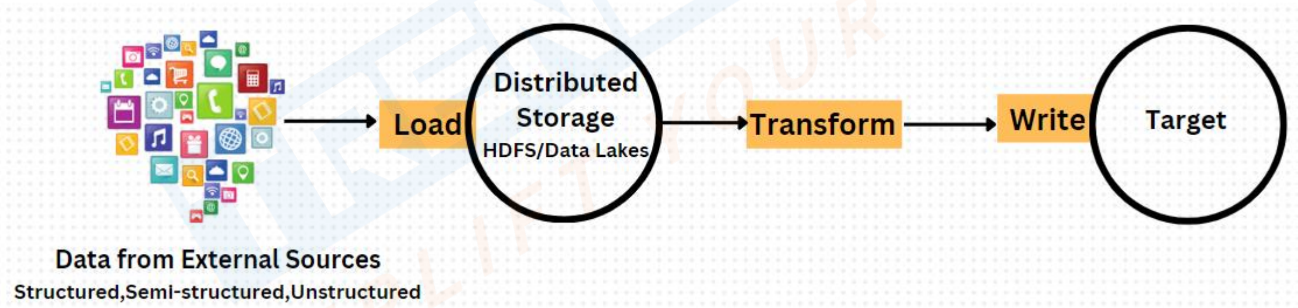
Apache Spark Vs Databricks

- Apache Spark is a Open Source distributed processing framework.
- Databricks is Spark on cloud with additional features.

Spark has 2 layers

Layer 2 of Spark	HIGHER LEVEL APIs			
	Spark SQL, Data Frames	Structured Streaming	MLlib	GraphX
Layer 1 of Spark	SPARK CORE APIs			
Working at RDD level	Scala	Python	Java	R

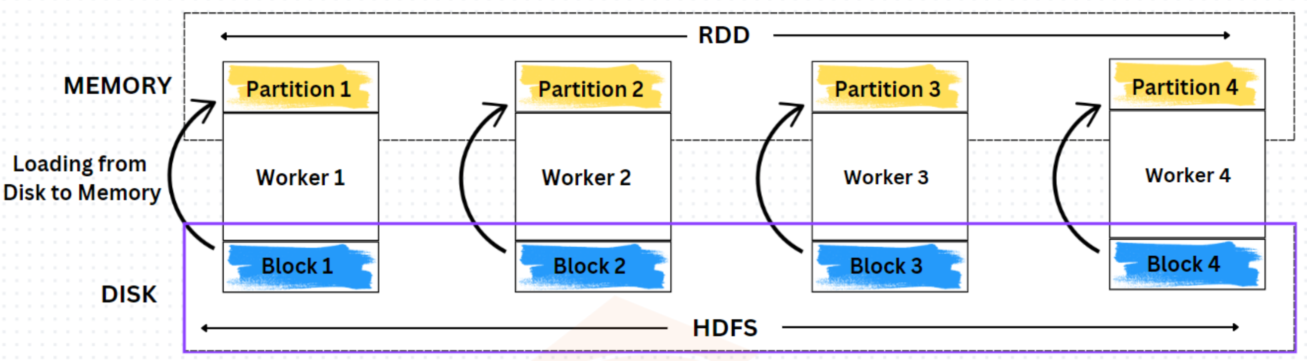
Spark performs the following 3 steps on data -



Note - Resilient Distributed Dataset (RDD) is a basic unit in Apache Spark that holds the Data.

Visualization of RDD -

Consider a 4 node Cluster. A file of size 512 MB is divided into 4 Blocks in HDFS (512 MB / 128 MB) and distributed across the cluster.



Working of Apache Spark - Driver and Worker nodes

Step 1 : Load file from Data Source (Datalake - HDFS or Cloud Datalake) to Memory (This creates an RDD)

Ex :

rdd1 = code to load file from datalake

Step 2 : Apply Transformations

Ex :

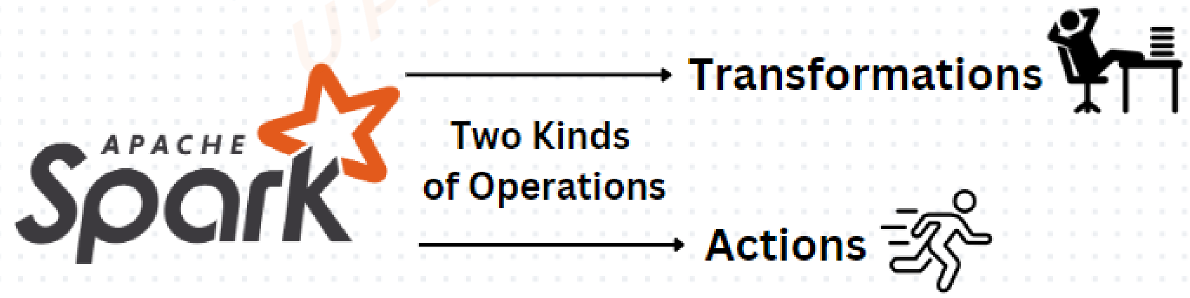
rdd2 = rdd1.map

rdd3 = rdd2.filter

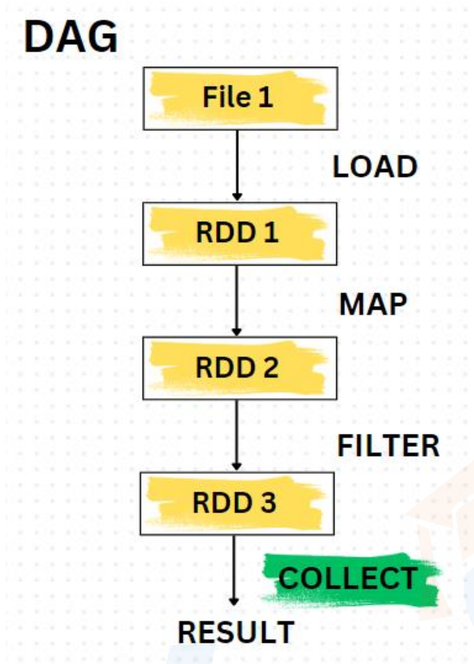
Step 3 : Action operation to get the results.

Ex :

rdd3.collect()



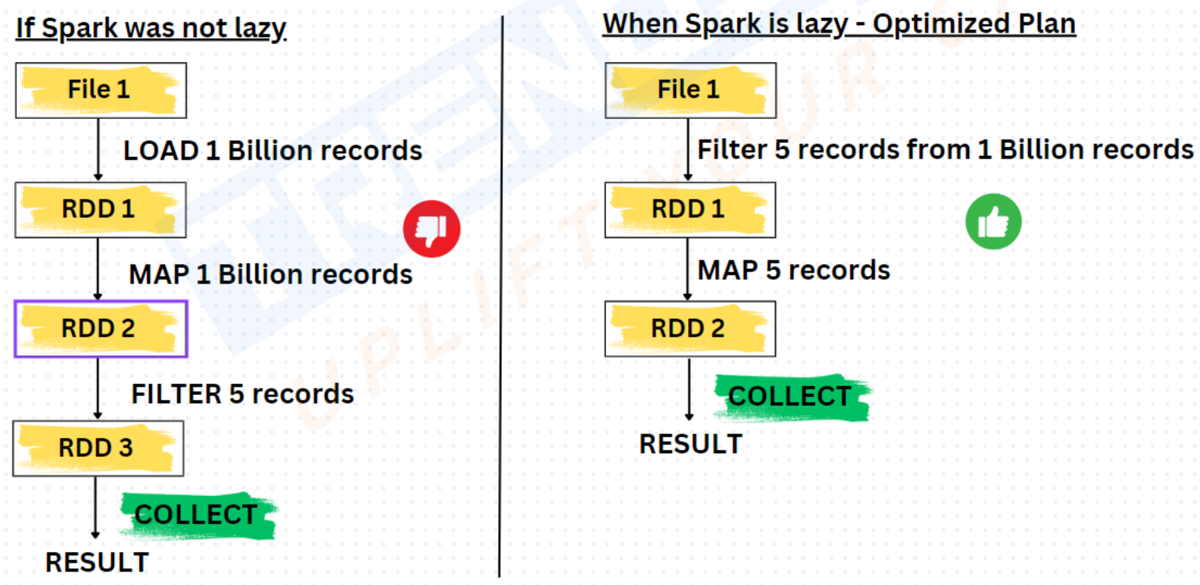
Directed Acyclic Graph (DAG) : Execution Plan for the Spark Job



Note : RDDs are Resilient to Failures because of -

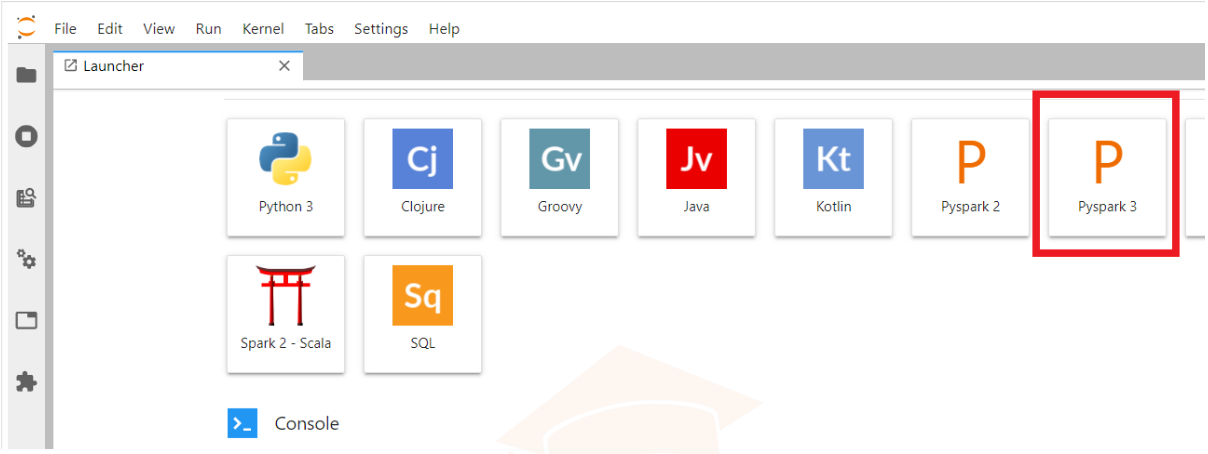
1. **Immutability (we cannot change the existing RDDs)**
2. **Lineage (lost RDD can be easily recovered by applying the transformation on the parent RDD as per the DAG)**

Advantage of Spark Transformations being Lazy :



Word Count Program on Spark :

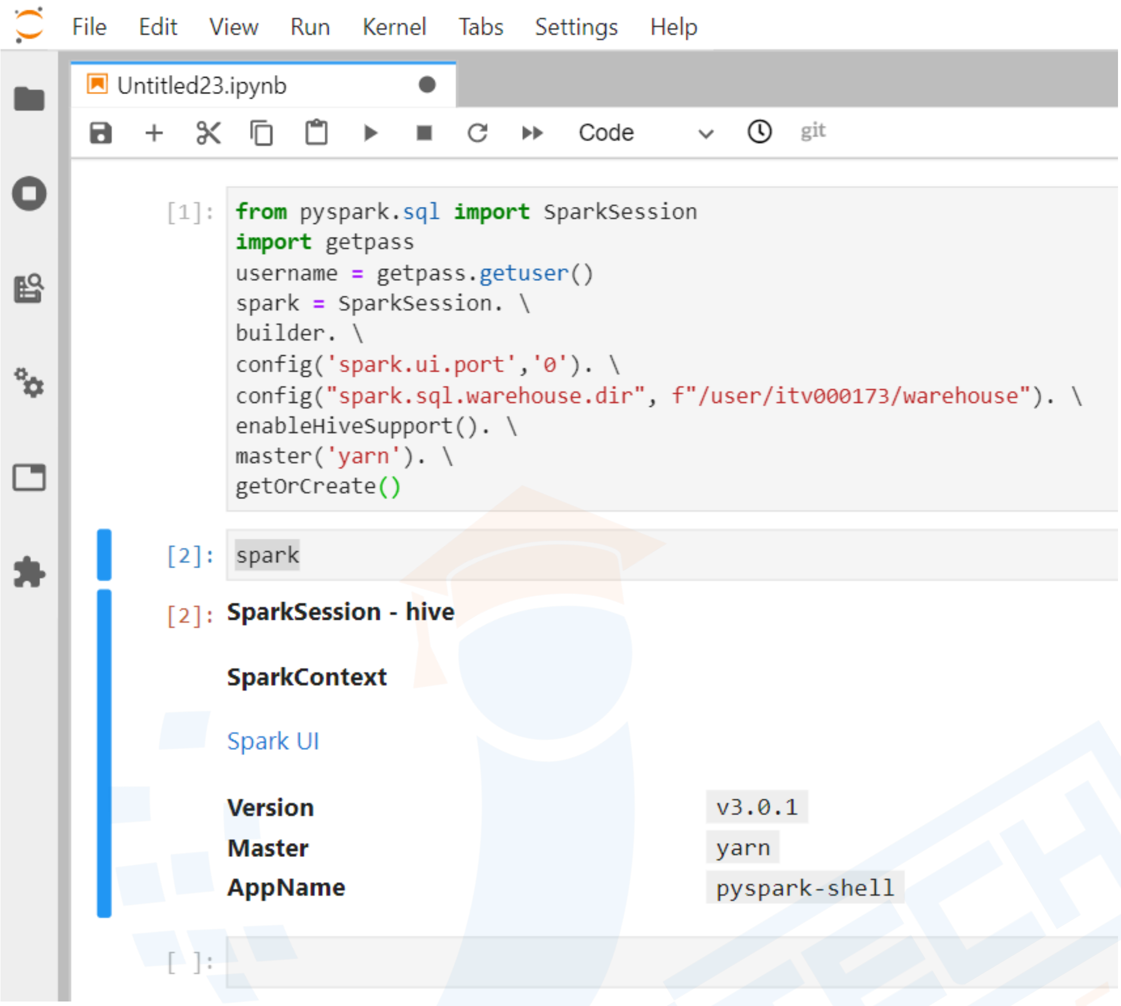
How to launch a spark execution environment in Lab?



Click on Pyspark3 in the application panel, a Jupyter Notebook launches where all the spark code can be executed.

1. **Create a Spark Session - Which is an entry point to the Spark Cluster holding all the context (Like Spark, Hive, SQL)**

```
from pyspark.sql import SparkSession

import getpass

username = getpass.getuser()

spark = SparkSession. \

builder. \

config('spark.ui.port','0'). \

config("spark.sql.warehouse.dir", f"/user/itv000173/warehouse"). \

enableHiveSupport(). \

master('yarn'). \

getOrCreate()
```



2. Load the file from source into memory as RDD

    rdd1 = spark.sparkContext.textFile("<file-path>")

3. Apply Transformations

    rdd2 = rdd1.flatMap(lambda x : x.split(" "))

    rdd3 = rdd2.map(lambda word : (word,1))

    rdd4 = rdd3.reduceByKey(lambda x,y : x+y)

4. Action operation to run the DAG and get the final output

    rdd4.collect() - can lead to out of memory error

    Or

    rdd4.saveAsTextFile("<output-file-path>") - to avoid out of memory error, save it as a file in HDFS