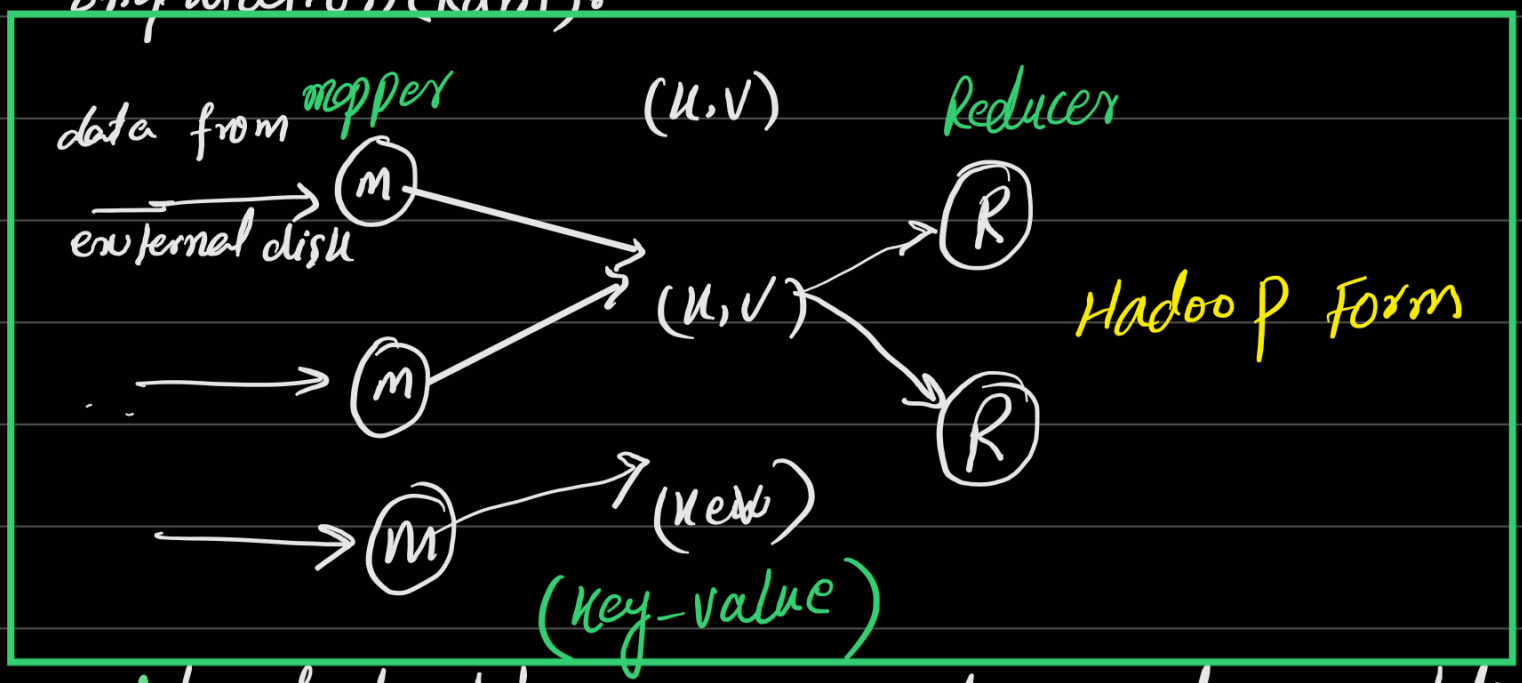
Apache Spark:-

Hadoop Vs Spark:

Hadoop Drawback:-

- It was only made for batch data processing.
- Slow processing because not in memory computation (Ram).



- Not fault_tolerance in term of computation.
  ∴ fault_tolerance = not affected when some node fail.
- NO mechanism for cashing and persist.

Hadoop → Map_Reduce
Hadoop → HDFS
Hadoop → Yarn (Resource Manager)

Map_Reduce

HDFS → split → map → Shuffle → Reduce → output

(1) Apache Spark.

It is an distributed computation framework used for large scale data processing.
It does not have any separate storage or file system.

Feature :-

- Speed : spark is 

times faster then hadoop, because it does in memory processing.
It hold data in memory for processing.

- Powerful_Caching :
capability to persist data in memory

- Deployment.
spark can work with different resource manages. / cluster manger.

1 → YARN : reource M_ Design specially for Hadoop.

2 → mesos

3 → Kubernetes

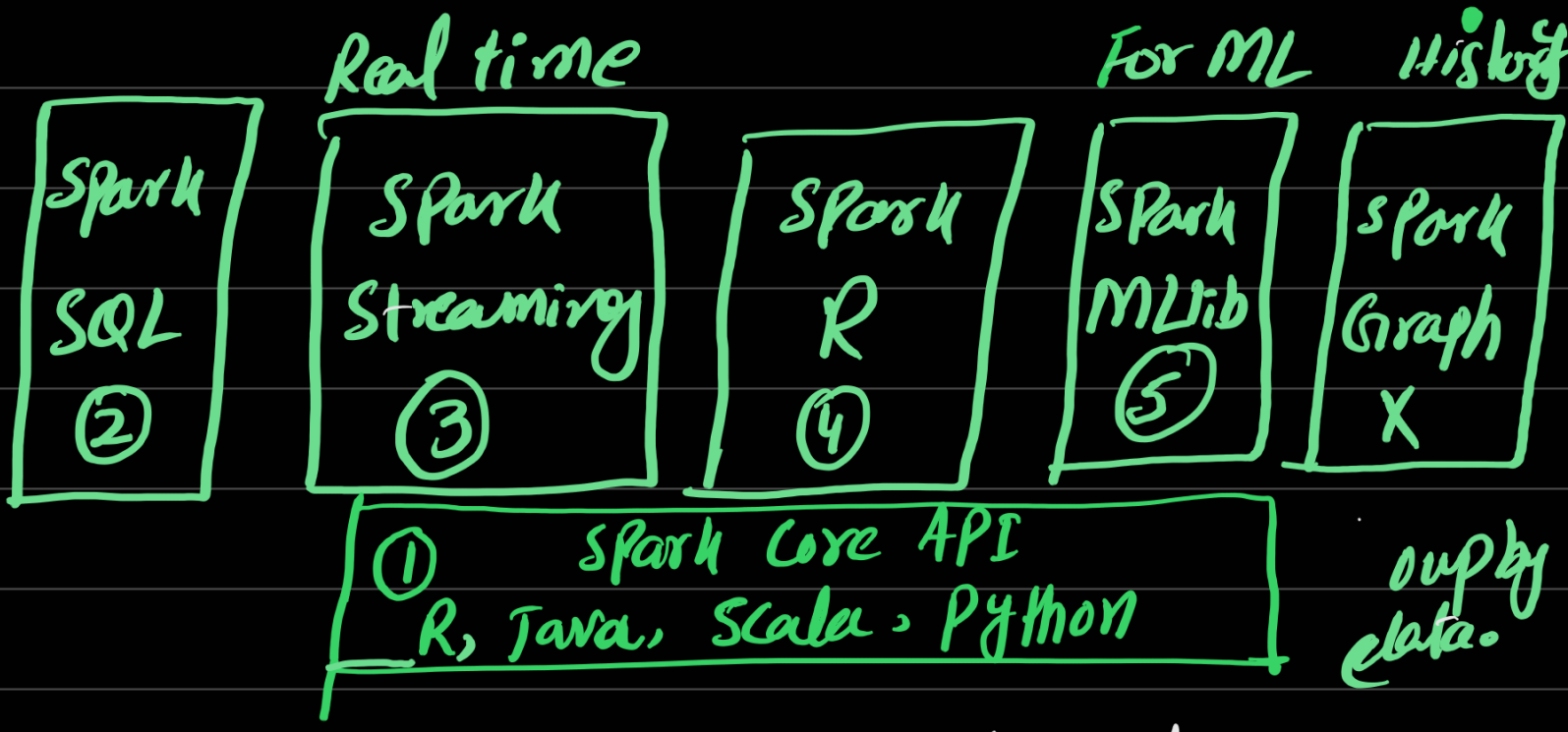
4 → Spark own cluster. Stand Alone

Resource Manager: ✓

- Good fit for batch and Real time data processing.

- Polyglot :-
spark provide high level API or libraries for different Language like Python, Java, Scala, R.

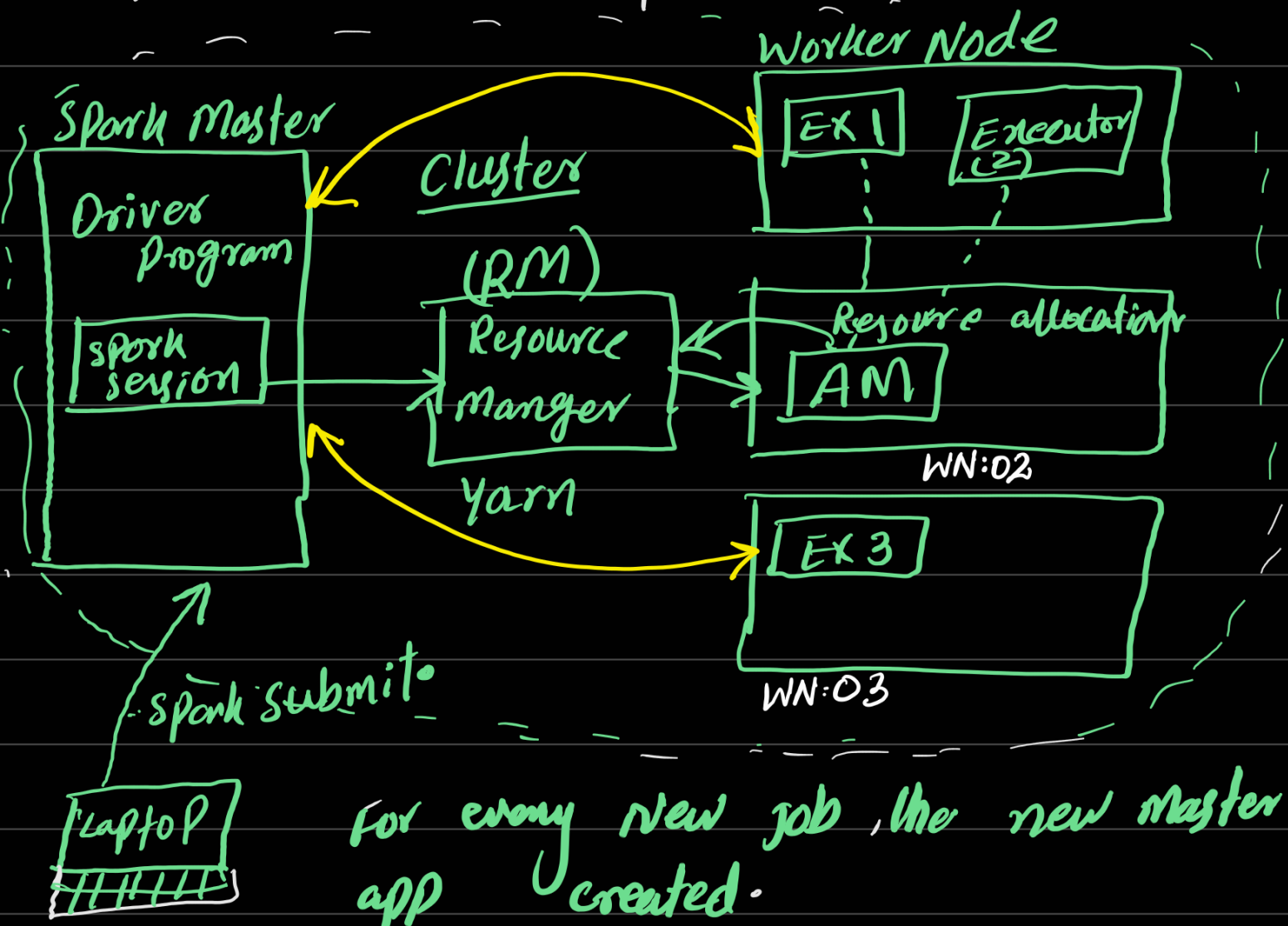
Apache Spark Ecosystem ✓



Yarn :- work by manging allocation of resources (CPU, memory and disk) accross the cluster/scheduling tash.

(3) Spark Architecture :- ✓

- Spark also fellows Master-slave Architure. — Do Parallel Processing (Data)
- Spark Master, Spark Worker node



For every new job, the new master app created.

✓ Driver Program.

- It act like a main method which will create Spark context.

The first method which will be called.

It does all mangment related things for Job like, lineage preparation scheduling the tasks, data distribution.

✓ **Spark Context as Spark Session:** Provide
- entry point in Spark cluster.

Spark Contex used working for (SQL & DataFrame)
- It will Connect spark application with execution cluster.
- Before spark 2.0 it was spark content (S) after 2.0 it is session.

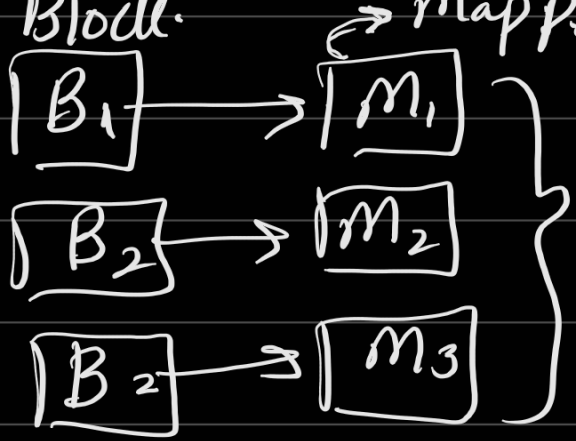
Resource Mangor:-
- It will Create the very first Container and start application master Service in it.
- Application master will request to RM for complete resource allocation or to start required numbers of executor.

Executor:- ✓

It does the Actual computation. They are like virtual containers on worker→

node which has CPU core, RAM, & Network Bandwidth.

(RM) will create required number of erector and offers that. Executor will start intrecting with (AM) to send the update for Job progress.



✔ (mapper task = Number of Blocks)

RM :-

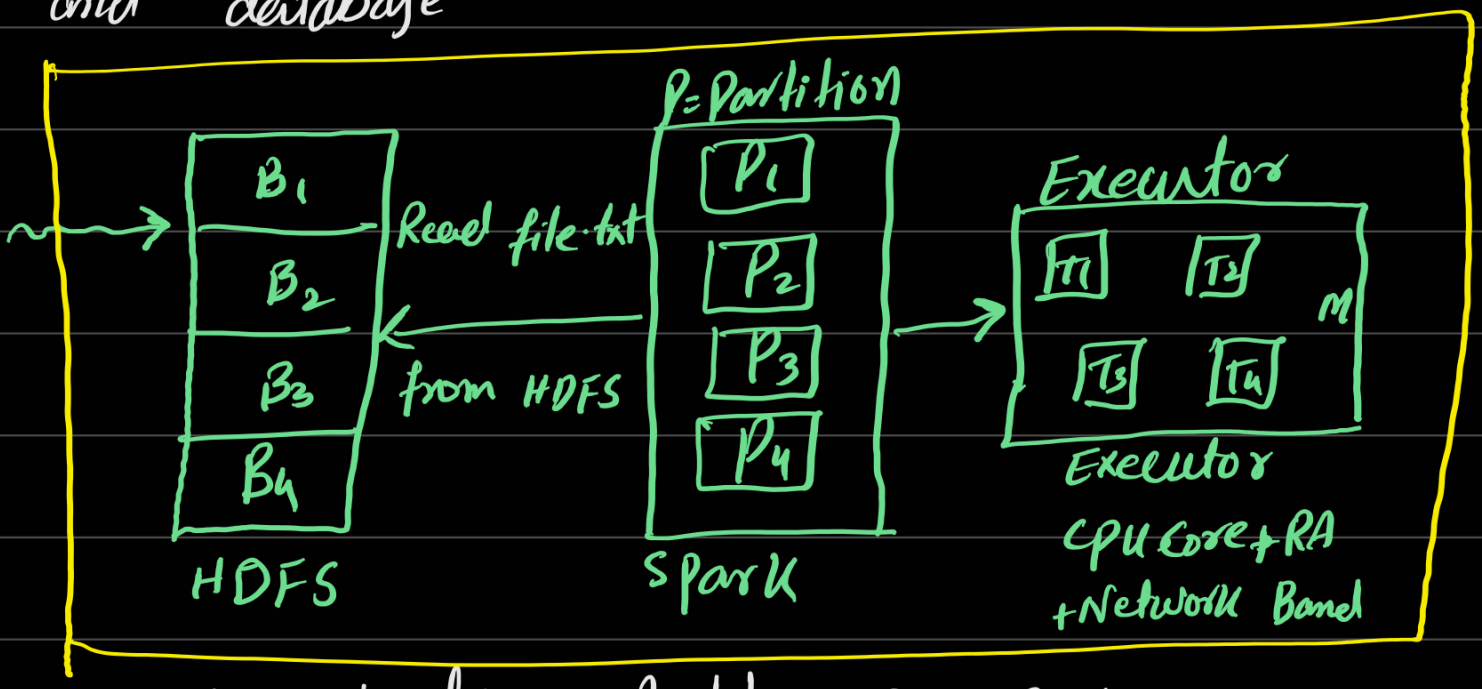
RM will create required number of executor.

Executor will start interacting with application master to send the update for Job progress

(4) Distributed Computation In Spark.

- In spark, data will be devided in number of partition. (NO block)
- How spark does parallel processing?

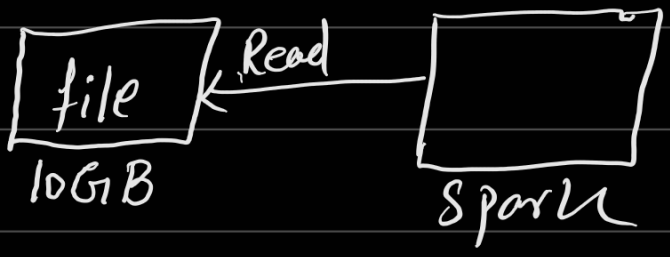
spark read data from distributed file system and database.



Block and like Partition in spark.

In Spark we will have controle to decide number of executor, ts CPU core.

(5) Partition Explicity in Spark.



Task:-

- Smallest unit of executon is known as task.
- Each task will Process one data partition.
- One task will get executed on 1 cpu core.
- One executor can run many tasks and equal to number of cpu core.

Parameter in Spark Submit Command.

- number of executor
- deriver memory (main Program to get memory for executon)
- executor memory
- Executor core.

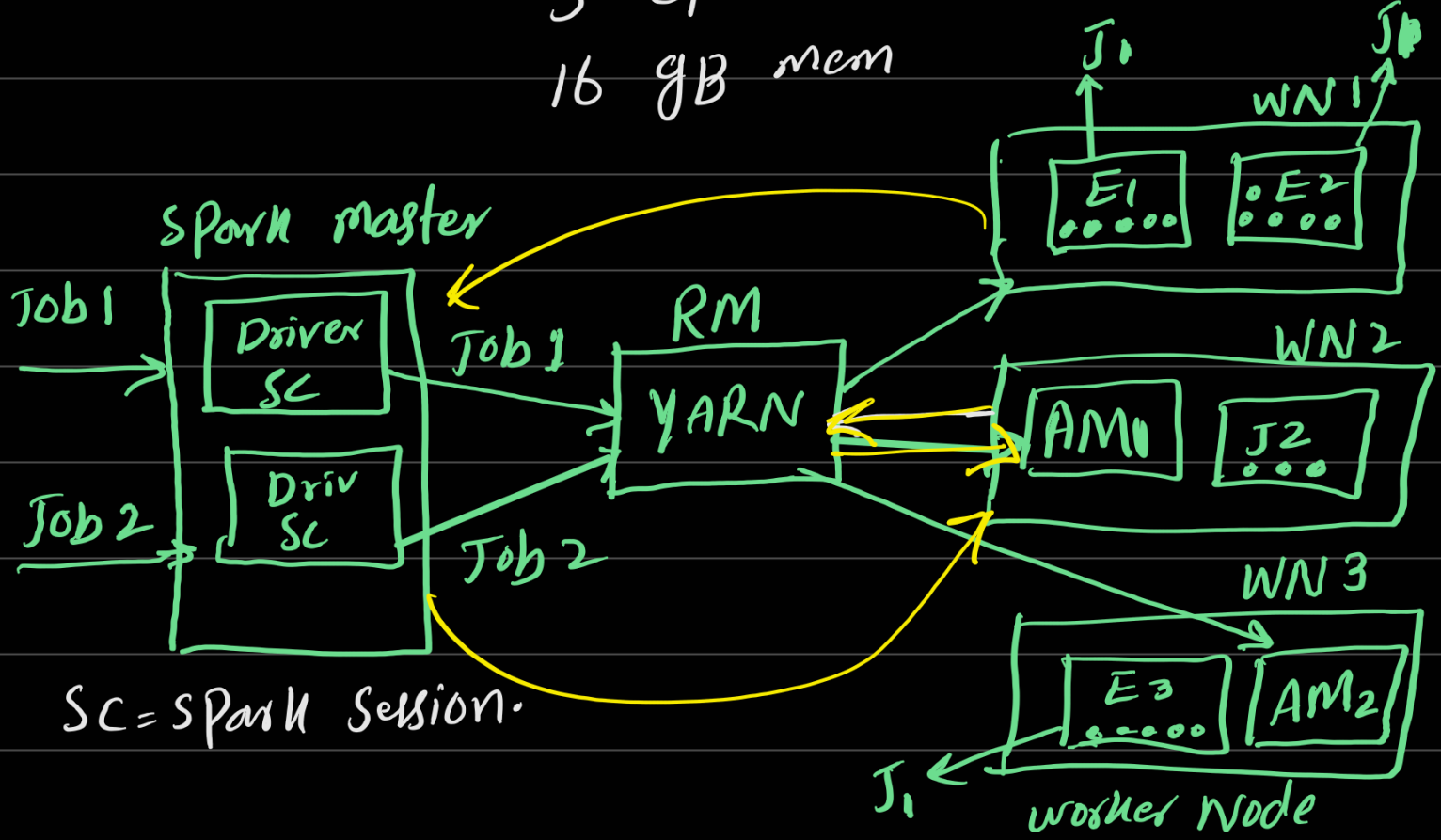
(6) Spark with Yarn as RM.

Ex worker_node => 256GB, 32 cpu core

we want to execute two JOBS (Job1, Job2)

Job 1 => 3 executor
5 cpu
32 Gb memory

Job 2 => 1 executor
3 cpu core
16 gB mem



- AM request to Yarn, to allocate resources like cores, executor.
- worker node create based on cluster capacity, (How much) they need to process. in Company.

(7) Deployment Mode in SPARK.

- client mode
- cluster mode

In client when you shutdown the machine task are ended / Job Cenceled.

In cluster whe you shutdown the machine Job running in background and excuted.

[Part = 2]

(8) RDD in Spark

RDD → Resilient Distributed dataset.

RDD are the main logical data units in the spark. They are distributed collection of objects. which stored in memory or disks of different machines on cluster.

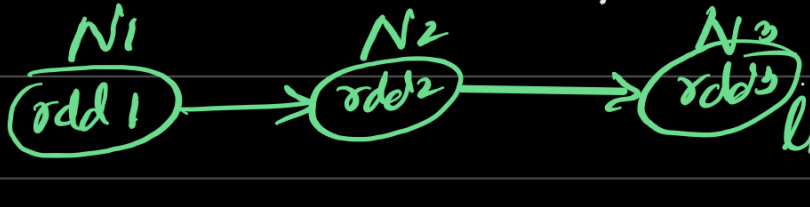
Feature of RDD (Data structre)

RDD track data lineage information to recover from fiald stat automaticaly.

It also known as fault Tolerance.

EX:-

```
RDD = Spark.read_csv("EX.csv")
RDD2 = RDD.filter("salary >1000")
```



lineage (sequence of data)

if N2 feril (RDD) automatically check the failed rdd.

- **Distributed:-**
  data present in a RDD resides on multiple nodes.
- ☆ **Lazy Evaluation.**
  data does not get loaded in RDD even if you define it. Do Action
- ☆ **Immutability:**
  Data stored in an RDD is in the read only mode. we cannot change the value during run time.

RDD —Trasform:→ RDD1 —Tranform→ RDD2

- **In memory Computation:-**
  An RDD stores any intermediate data

that is generated in the RAM so that faster access can be provided.

RDD Vs Dataframe:

RDD: It can have any type of data (unstructre, semi and Structre)

```
RDD 1 = sparK.read("data.txt")
df 1 =sparK.read("data.csv")
```

Data frome: allow users to create and manupulate data in tabuler format, excel, SQL

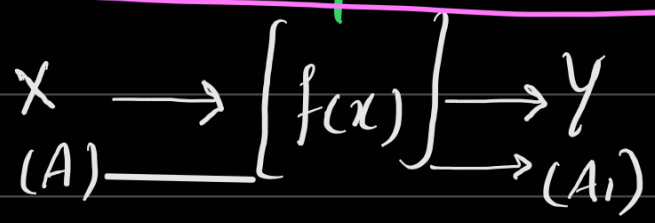
(9) Action and Tranformation in SPARK.

Tranformation:-

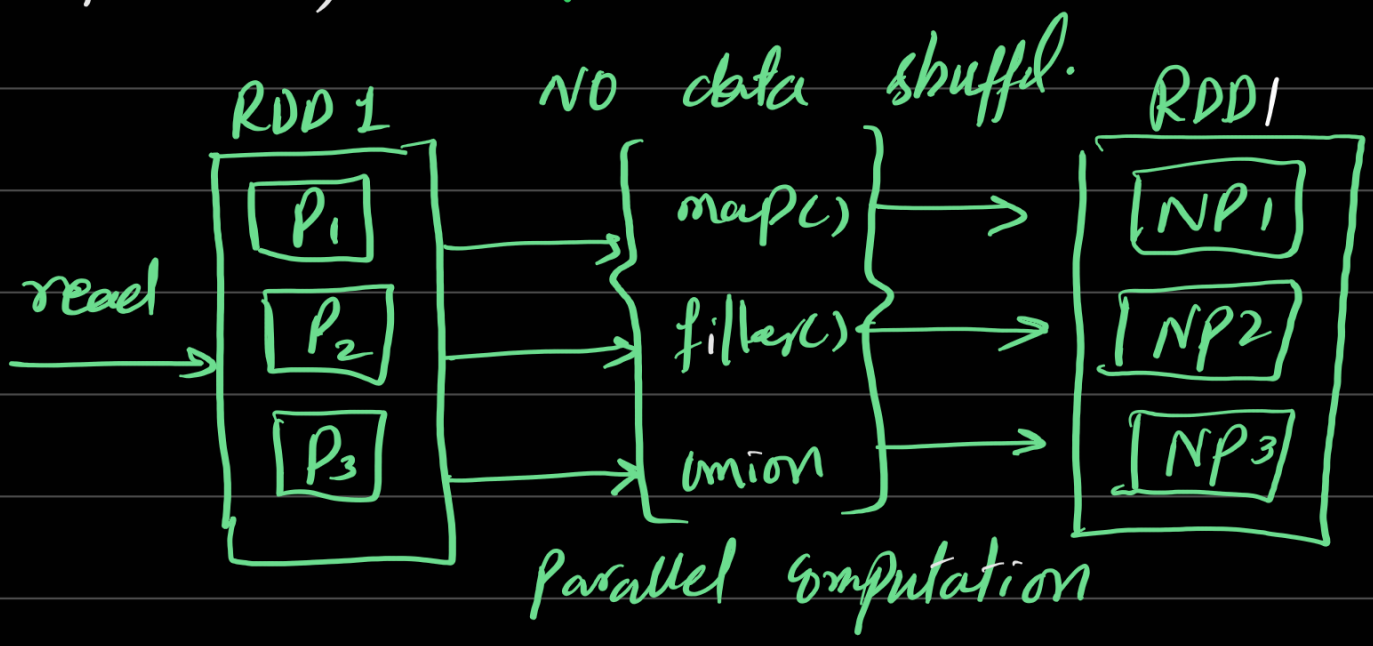
it is a function that produces a new RDD from existing RDD.
It takes RDD as input & generate new RDD as output.

- Two types:-
  - Narrows Tranformation (no data shuffle)
  - wide Tranformation (data shuffling)

Narrow Transformation:-



map(), flat map(), union()
filter(). Select.

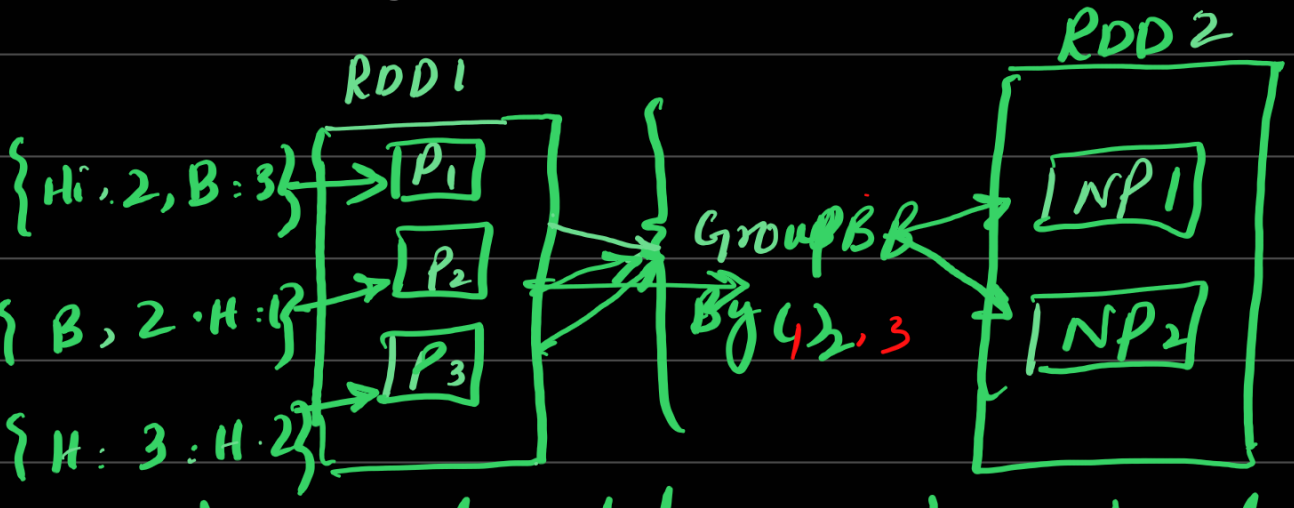


a = [1, 2, 3, 4] result = [1, 4, 9, 16]
fun = lambda x: x**2

Wider Transformation: Shuffle:

In wide Transformation all the element that are required to compute the records in a single partition, may live in many partition. That' why we need to shuffle the data.

Join() , groupby() , Repartition()
ReduceByKey() , GroupByKey()
get data from many executor/partion.
Example of word Count:-



+ Same type of data maybe stored in other partition like: Numbers, Int, float, Char.

(9)
+ Action In Spark:-
Action is a way of sending data from executor to the driver.

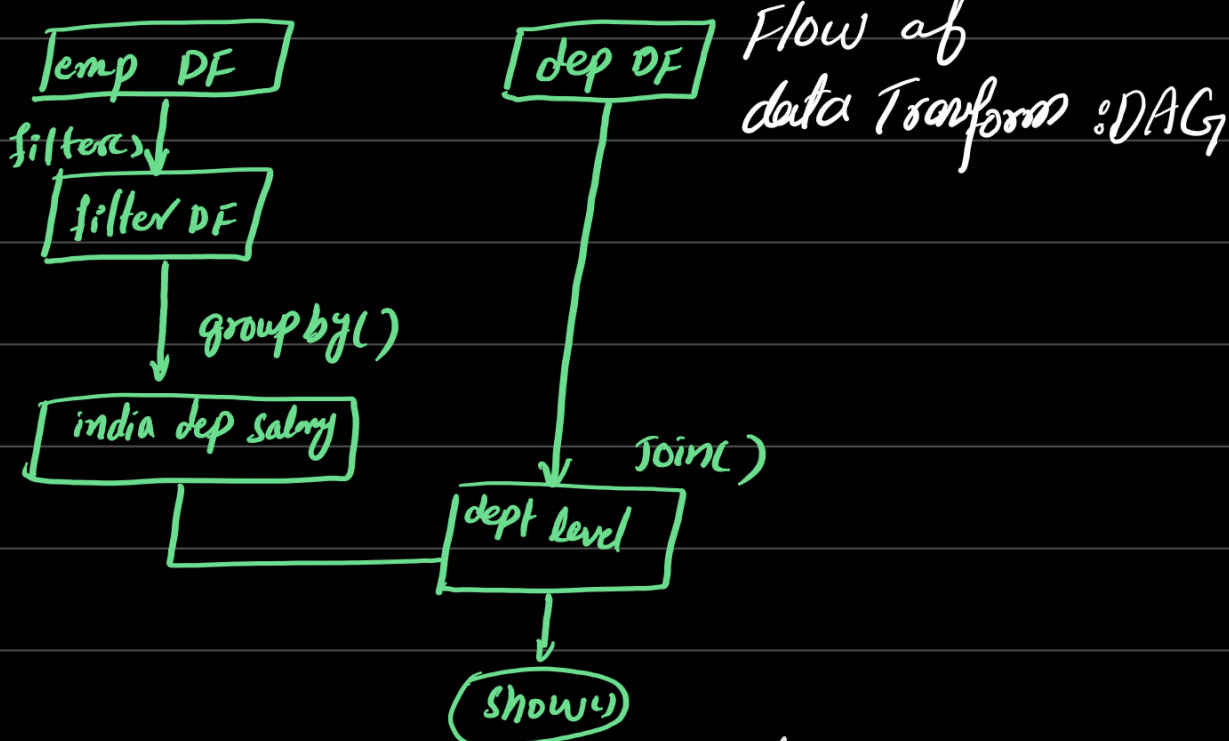
Executor → Driver, collect(), take(), top()
Driver is manage the thing.

+ Demo Pyspark application (first.py)
spark = sparkSession.getOrCreate()

```
empDF= spark.read_csv("empby.csv")
deptDF= spark.read_csv("dep.csv")

filterDF = empDf.filter("Country = "INDIA")
```

Lineage graph(DAG) :- ✓

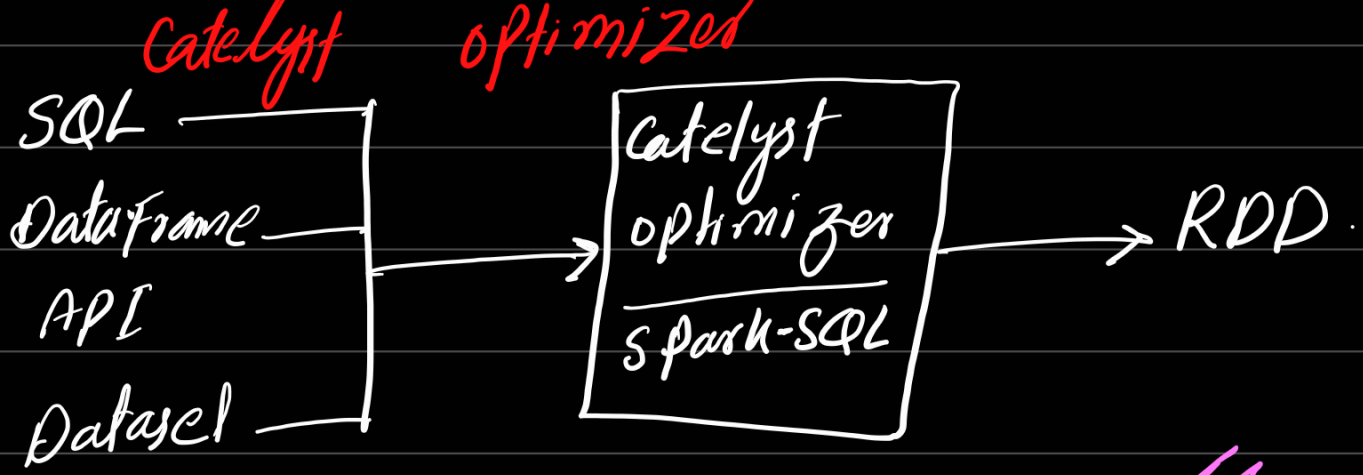


Flow of data Tranform : DAG

Two time data shuffling happen groupby and join().

- DAG create for each Job. Job created when action run.

when we do Tranformation it optimize the code in Lazy Tranformation.



Phases Spark-SQL engine:-

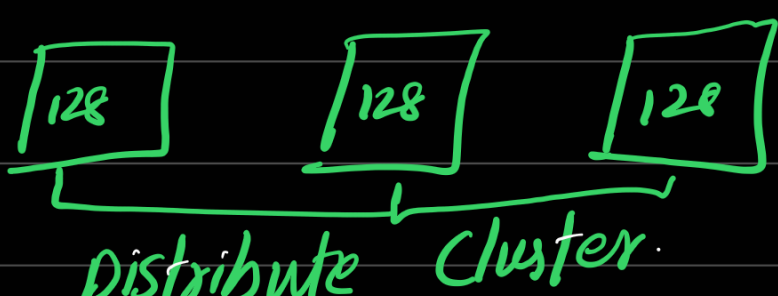
1. Analysis
2. Physical
3. logical
4. Code generate.

Catelog:- meta-data of our data:
{Size of file, Name, when upload.}
all information about orignal File.

RDD :-

Resilient distributed dataset.

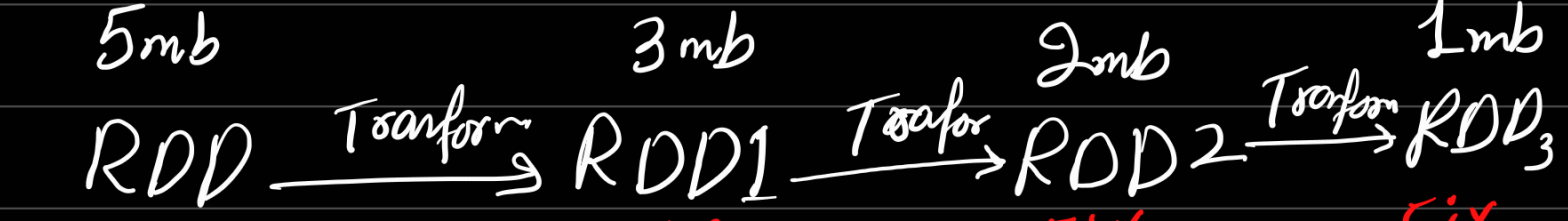
RDD is type of Data structure. It work on our cluster. suppose = $\frac{500}{128}$ = 4-partion.



(Resilient) is fault-tolerant.

RDD immutable:-

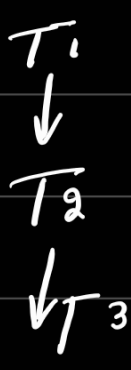
we cannot change RDD if it create.
Its have only reeel operate not right one:-



we create copy form 1RDD to 2nd RDD.

⟹

It creat DAG.



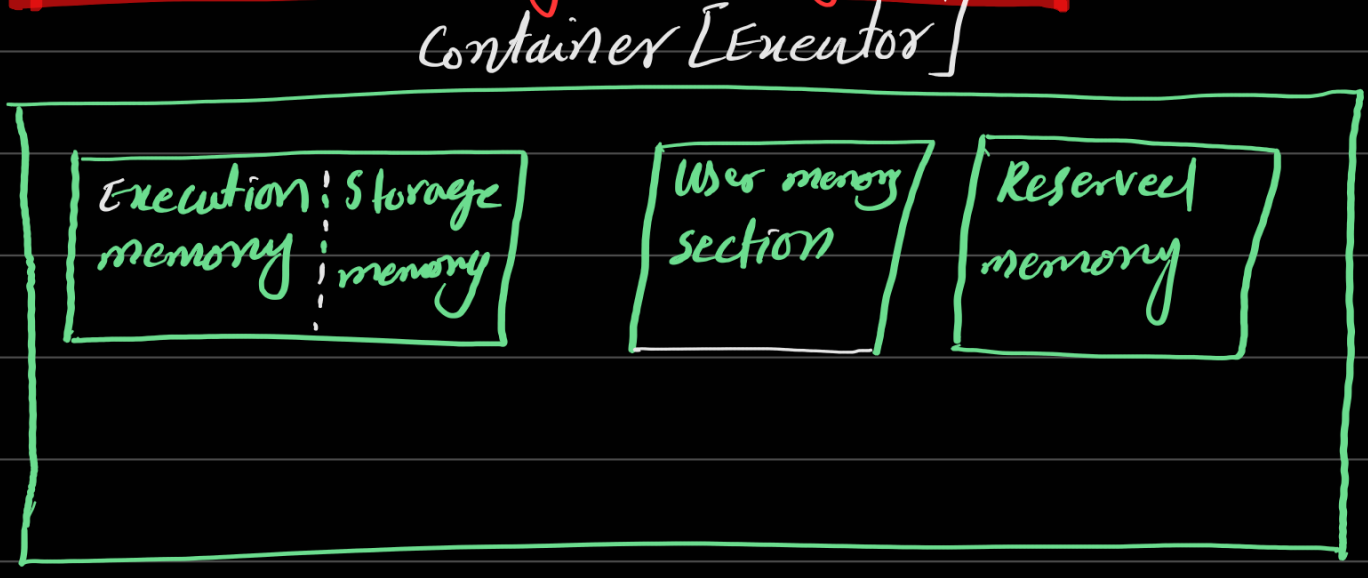
- RDD : No optimize

if ~~RDD1~~ Fail. then it creat agoin creat.

- benefit → work with unstructure data.

when to use RDD? when we need full control our data.

Spark Memory Managment.



Container (Executor)

- CPU , core
- Ram
- Network Bandwidth

Spark_Submit Command.

Spark_submit

--master <master-url>

--deploy-mode <Cluster or Client>

--Config <key:value>

--driver_memory

--executor_memory

--num-executor (N)

we have 1 node with 32 GB ram and each executor can have 4GB memory ?

Total executor = $\frac{32}{4}$ = 8 executor.

Reserved memory:-

This is storage for running executor. it will store the data of internal processing which is required to execute the container.

fixed 300mb

User memory:-

It store the data structure, meta data and user defined data structre. (struct)

Storage memory:

It stores cash data, data of broad cost variable.

Execution memory:

It is a storage for the data required for executions, shuffle, join

aggregation etc.

Example of memory distribution.

Let say we have given 4GB memory to each executor.

Executor-memory (Java heap) = 4GB

4GB = 4096 MB

Two parameter impact on memory component.

- Spark.memory.fraction = 75% (0.75)
- Spark.memory.Storage fraction = 50% (0.50)

Calculation:

- **Reserved memory:-**

It is fixed 300MB.

- **User memory:-**

formula = (Java heap - Reserved memory) x (1 - Spark.memory.fraction)

= (4096 - 300) x (1 - 0.75) = 949 MB

- **Storage memory:-**

formula = (Java heap - Reserved mem) * (Spar.mem.frac * Spark.mem.Stor.fraction)

= (4096 - 300) * (.75 x 0.50) = 1423.5 MB

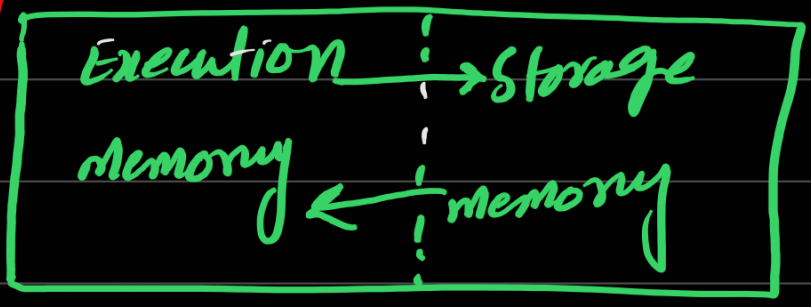
Execution Mem :-

formula = (Java heap − Reserved mem) * (spark.mem.frac) * (1 − spark.m.storagefraction)

= (4096 − 300)(0.75)(1 − 0.25) = 1423.5 MB = 1.4 GB

Driver memory error
Execution memory error

(ii) Rules for execution and storage memory.



- storage mem can borrow space from execution mem only if memory blocks of execution mem are not in use.
- Execution memory can also borrow space from storage memory if blocks are not in use.
- If block from execution memory is used by storage memory and execution memory need more memory it can forcefully evict the excess blocks occupied by storage memory.

If block from storage memory is used by execution memory and storage memory needs more memory.
It cannot forcefully evict the excess block occupied by execution memory.
It will end up having less memory
It will wait untill spark realses the excess block stored by execution memory and then occupies.

(12) Ex Resource Allocation for Spark app:-

Case 1: Hardware Config → 6 Nodes in cluster each node is having 16 cores & 64GB RAM.

☆ From Recommendation 5 CPU core for executor.

1 GB & 1 core of each Node will be required by for operating system. ✗

15 cores & 63GB

How many executor in our cluster?

∵ 5 cores = 1 executor ?

15 cores = 15/5 = 3

1 Node = 3 executor

Total executor on cluster = 6 × 3 = 18 executor.

1 executor will be created by YARN as AM.
so final executors which we should mention in spark submit.

num-exect = 18 - 1 = 17

Memory for each executor?

1 Node = 3 executor

1 Node = 63 GB

1 executor = 63/3 = 21 GB

spark.memory.overhead = 10% of exe. memory.

= max(384 MB, 0.07 * executor_mem)

overhead memory = max(384 MB, 0.07 * 21)

= max(384 MB, 1.47 G)

= 1.47 GB

Final executor mem = 21 GB - 1.47 GB

memory for each executor is = 19 GB

find values of Spark-submit parameter based on (Case-1) resources.

5 CPU core per executor
17 executor
19 GB for each executor

(13) How to process 1 TB of data in SPARK.

Here 1 tb data store in HDFS

in HDFS the block size is 128 mb.

$1\,TB = (1 * 1024 * 1024)/128 = 8192$ block.

Let us take :-

=> 20 executor = each executor have 5 cores
=> 5 cores - in each executor
=> 6 gb ram in each executor.

In parallel $20 \times 5 = 100$ task will be running.

100 task = 100 data block.

$100 \times 128 = 12800\,mb = \frac{12800}{1024} = 12.5\,GB$

12.5GB data will get process in first batch.

RAM

$\Longrightarrow 6 \times 20 \text{ Exec} = 120 \text{ GB}$ RAM

So at the time 12GB of Ram will be occupied in cluster.

$\frac{20 \text{ Node}}{120\text{GB}} = 1.6 \text{ GB}$ RAM from each executor will be used.

Now Available RAM in each executor (6GB − 1.6GB = 4.4GB) will be free to use by other user Job.

$1\text{TB} = \frac{1024\text{GB}}{12\text{GB}} = 85$ Batches.

whole data will Process covered 85 batches.

Failure Scenarios in Spark:

1) out of memory error.

- avoid unnecessary data shuffle
- Check Broadcast big in size

(2) Network Error

It happen when **spark driver** & **executor** are not able to **communicate with** each other.

Solution = { spark.network.timeout, spark.executor.heartbeatInterval }

(3) Data Input / output.

- Make sure data is consistant as per data contract.
- Always put read/write related operation in try_except block for easy debugging.

(4) Serilization Error:-

Data is not according to the serilized schema.

(5) Disk Space Error:

spark job spills intermediat data

into the local machine Disk. if no mem then job fil. Ex (spark.local.dir)

Heap memory:- Part of memory which allocate to (JVM) to store object. JVM used heap memory when spark app running to store object, data (RDD). Driver/executor memory.

(6) Memory Configration.

If we are facing heap space relatced issue and memory overhead issues then try to configure mem component

=> Analyse the data volume

=> Define driver & executor memory properly

=> Adjust memory configration parameter properly.

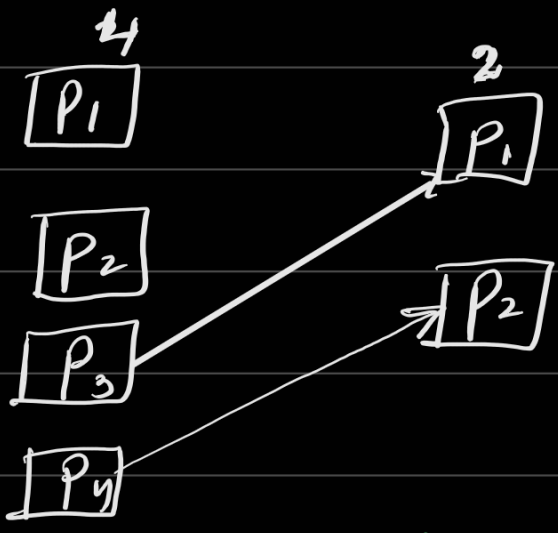
Frequently used optimization in SPARK.

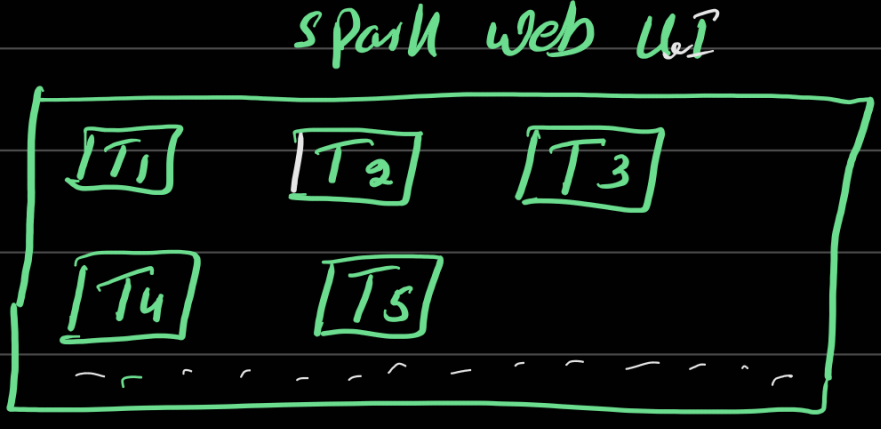
(1) Partition:-

- Repartition() increase
- Coalesce decrease

```
emp DF = spark.read.csv("emp.csv")
emp DF.partition(10)

Join DF = emp DF.join(dept DF)
```





(2) optimization in the case of skewness we can use key-sorting technique.

(3) Broadcast variables:-
It will help to broad cost dataset on multiple executor.

(4) Persist() and Cashing():-
Cash() only in memory save.

Persist() → memory + Disk both.

(5) Try to minimize number of stages in the spark Job. That means unoptimized uses of wide Tranformation

Difference between Job vs Stage vs Task (interview)

Job:-
when Action hit it creat Job.
N.Action = N.Job.

Stage: wherever data shuffling takes Place in the spark app. a new stage get Created.

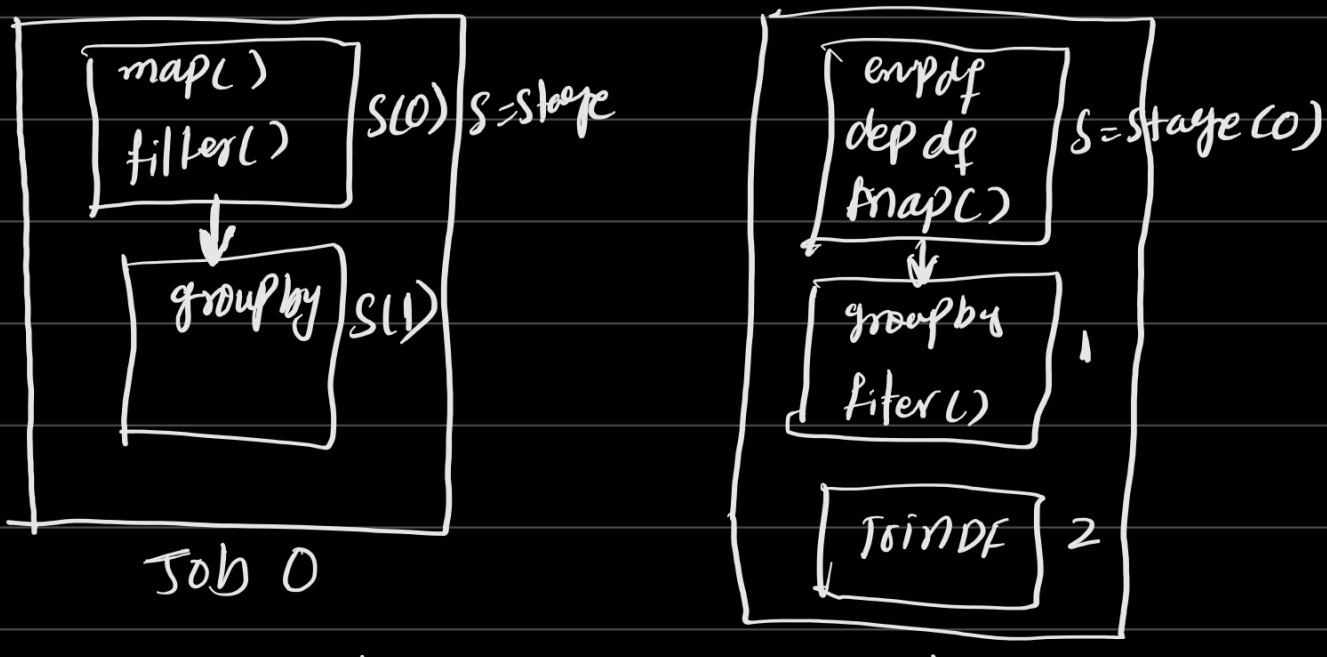
Stage 1 → Stage 2 → Stage 3

Task: small unit of execution. called Task

empDF,
deptDF,

EX: empDF.filter('INDIA')
EX: empDF.groupBy('dept_id.Sum(salary)

Transaction
```
JoinDF = empDF.join(deptDF)
JoinDF.filter(deptDF.dept_id NotNull)
JoinDF.collect(10)
```
Action



Number of action to number of Job. and shuffling occure.

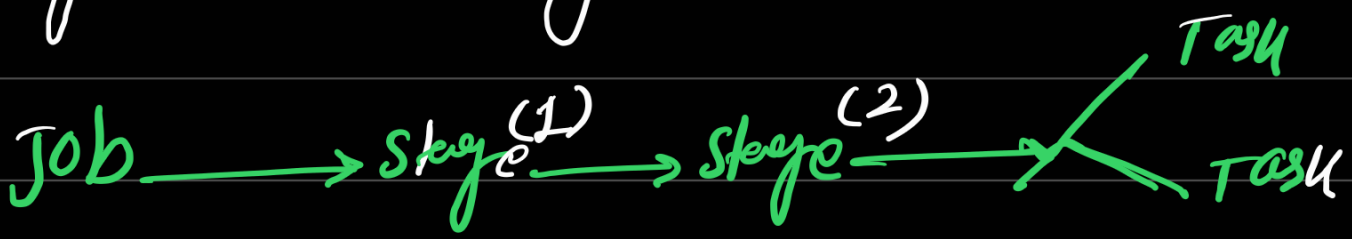
1 Action = 1 Job.

//////////////////////////

How many Job created for 100 GB of data.

Jobs based on type and Transformation of data.

Each Job have many Stage, each Stage have many Task



P1

Id	Name
1	
2	
3	

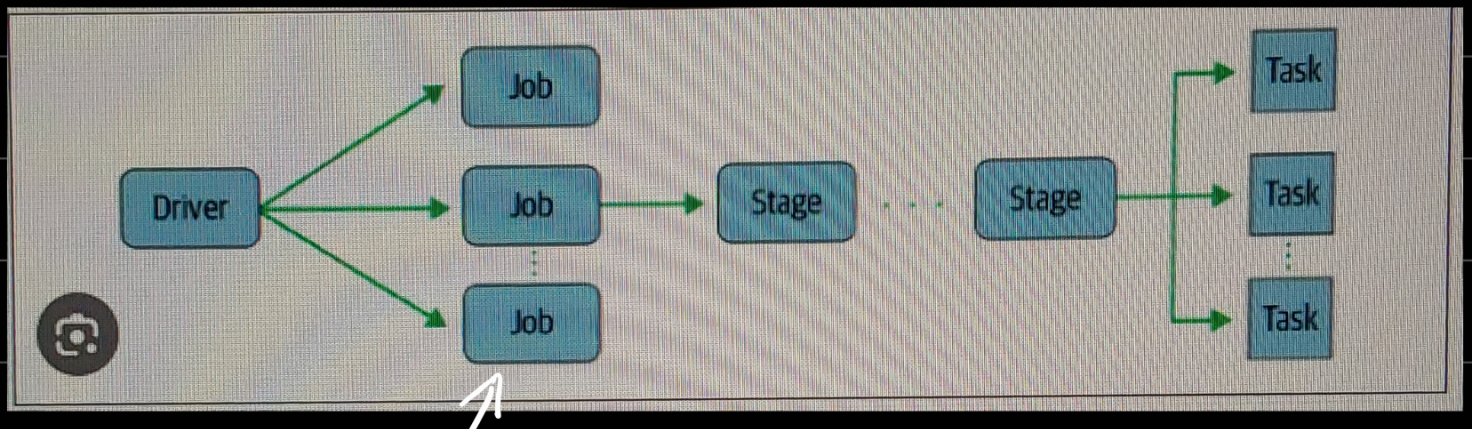
P2

Id	Name
4	
5	
6	

Print where Id is even: (Data Shuffel)

(Point):-

Exact Number of Jobs and Stages based on Spark app, data and Tranformation.



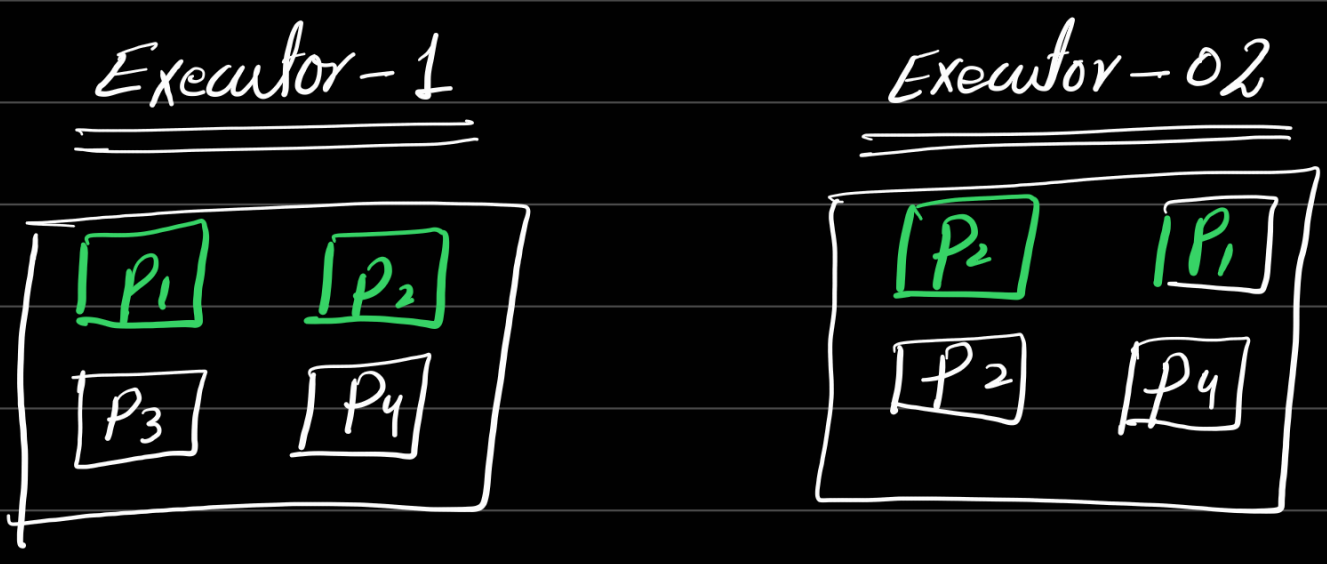
Action HiT

Repartition vs Coalesce:

- Coalesce use to decrease portion.
- partition used to increase or decrease Partition.

Spark Join:

Shuffle data between worker-Node



Strategies in Spark:-

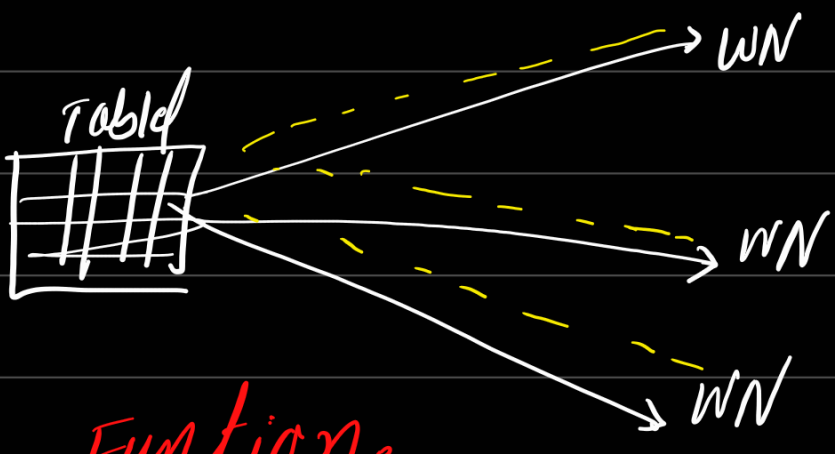
- shuff Sort Join
- shuff hash Join
- Broad Cast hash Join
- Cartesion Join
- Nested Join

Broad cast Join in Spark:

- Broad cast hash Join:

Broad cast data, during Join. which eliminate shuffle.
It increase the speed of data.

☆ Broad Cast Join sends copy of small table to all worker node.
Ensure that all worker node have memory.



Hash Funtion:-

- meta data of real data.
  real data store in hash data.
- 

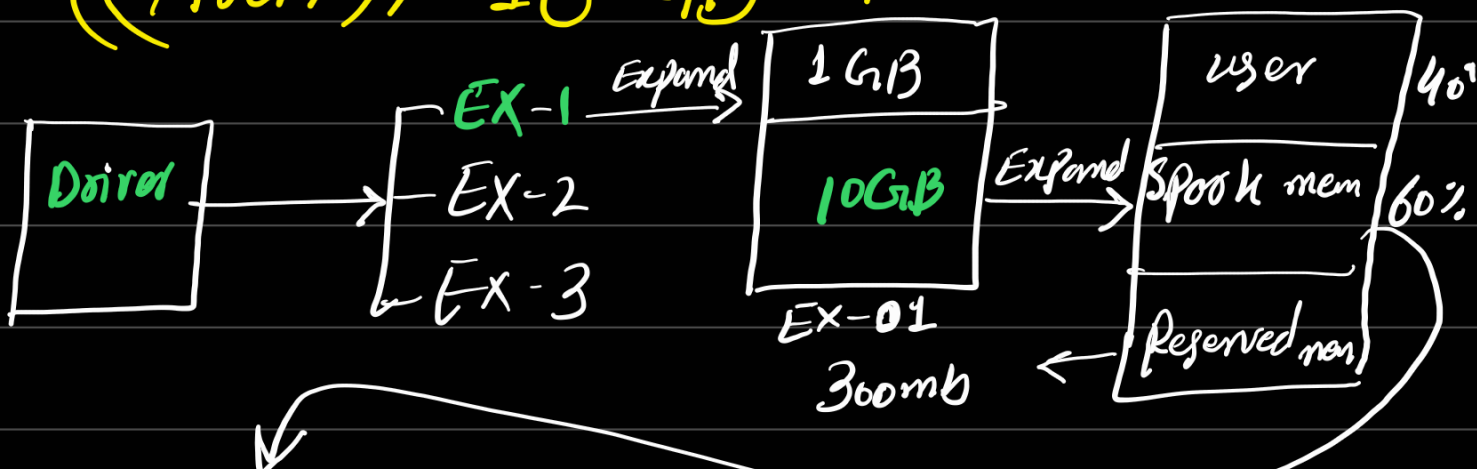
Execution memory error:-

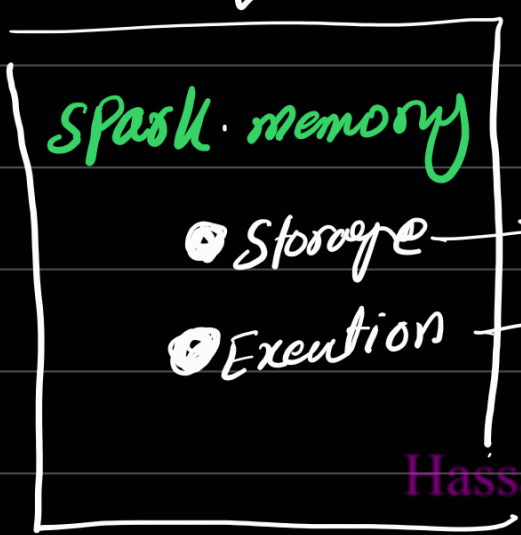
- out-of-memory error:-

- Optimize memory. need memory.
- Caching
- Partition correctly
- monitoring
- data Filter

Avoid of error.

((Arch)) 10 GB Data





Unified memory Management.

4-core , 4-task.

Spark-submit