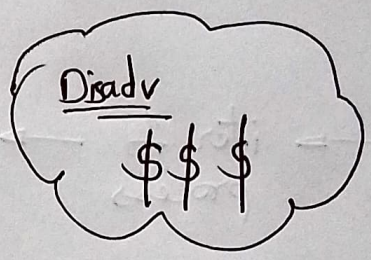
Spark

1. Disk Computing Framework
2. It is independent from Hadoop
3. It does not have any File System.
4. Read + Write externally like read from RDBMS, HDFS, Local files or any kind of Local file system/ or Cloud.
5. Faster than Map Reduce
6. In memory storage.



Spark is not extension of Hadoop.

Spark can run on Standalone sys.

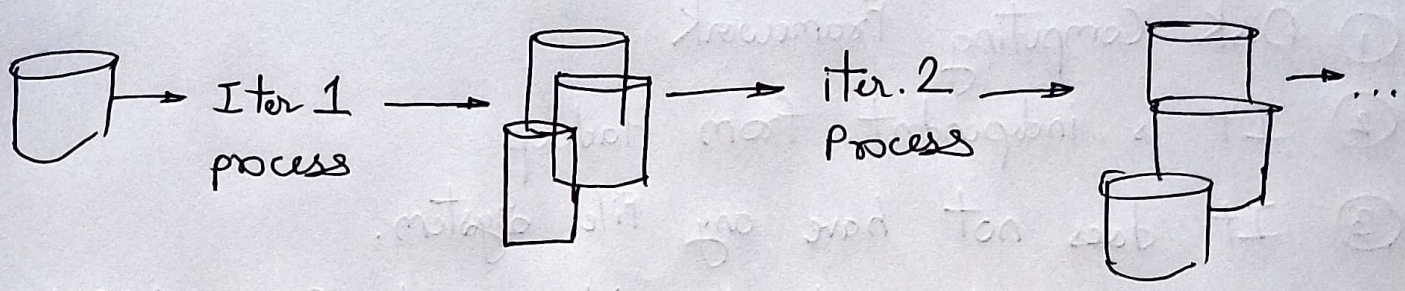
It's a low latency computing sys.

Separate, fast, MapReduce-like Engine

Earlier we Studied,

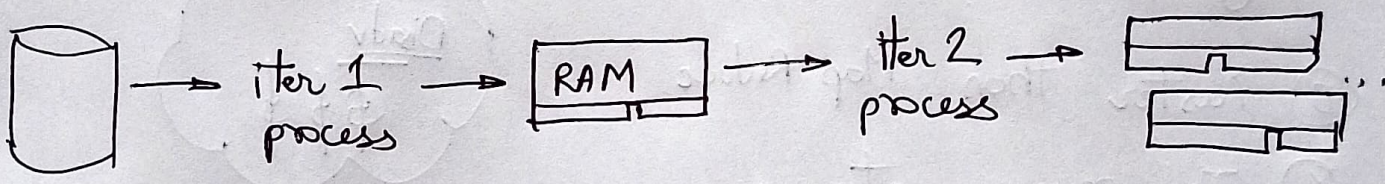
Map → Local Disk → Reducer → Local Disk

MapReduce

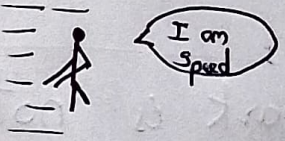


Spark

Every task data will be stored in memory



And the final o/p will be stored at disk i.e maybe Oracle DB or any external disk



40x faster / 100x faster than Hadoop

→ than MapReduce for iterative algorithm.

But it is Compatible with Hadoop's Storage APIs

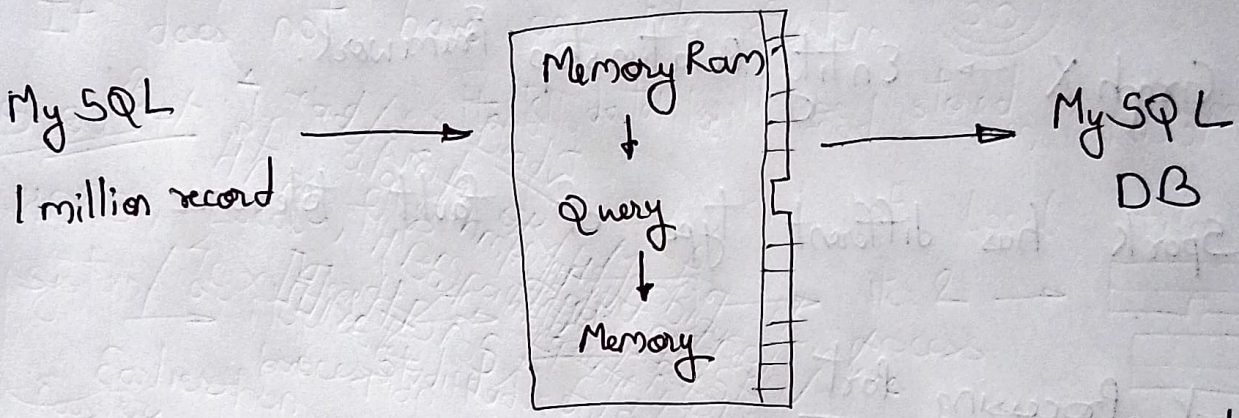
→ read/write to any Hadoop-Supported system, HDFS, HBase, SequenceFiles etc

---

Hive = HDFS + Table Support

Hadoop = Processing + Storage

Spark = Only Processing (Saw Pratishat)



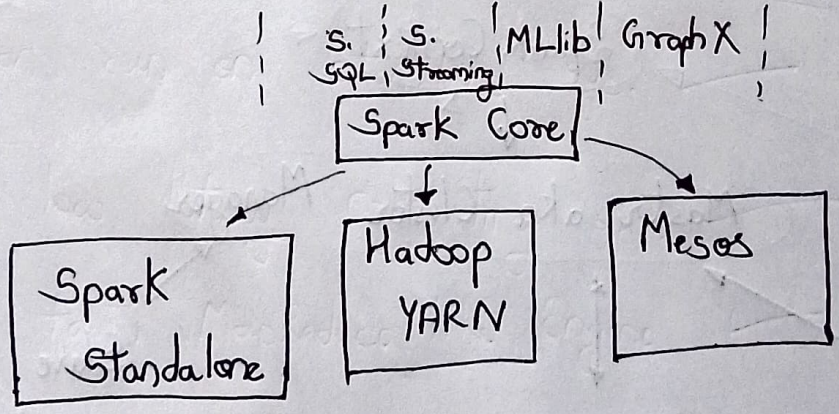
Spark Architec

Core Engine language (Scala)

⇓

JVM

(To run Scala)



- Scala mai likho
- Java mai likho
- Python mai likho

} To write code in Spark

To write SQL query in Spark we use Spark SQL.

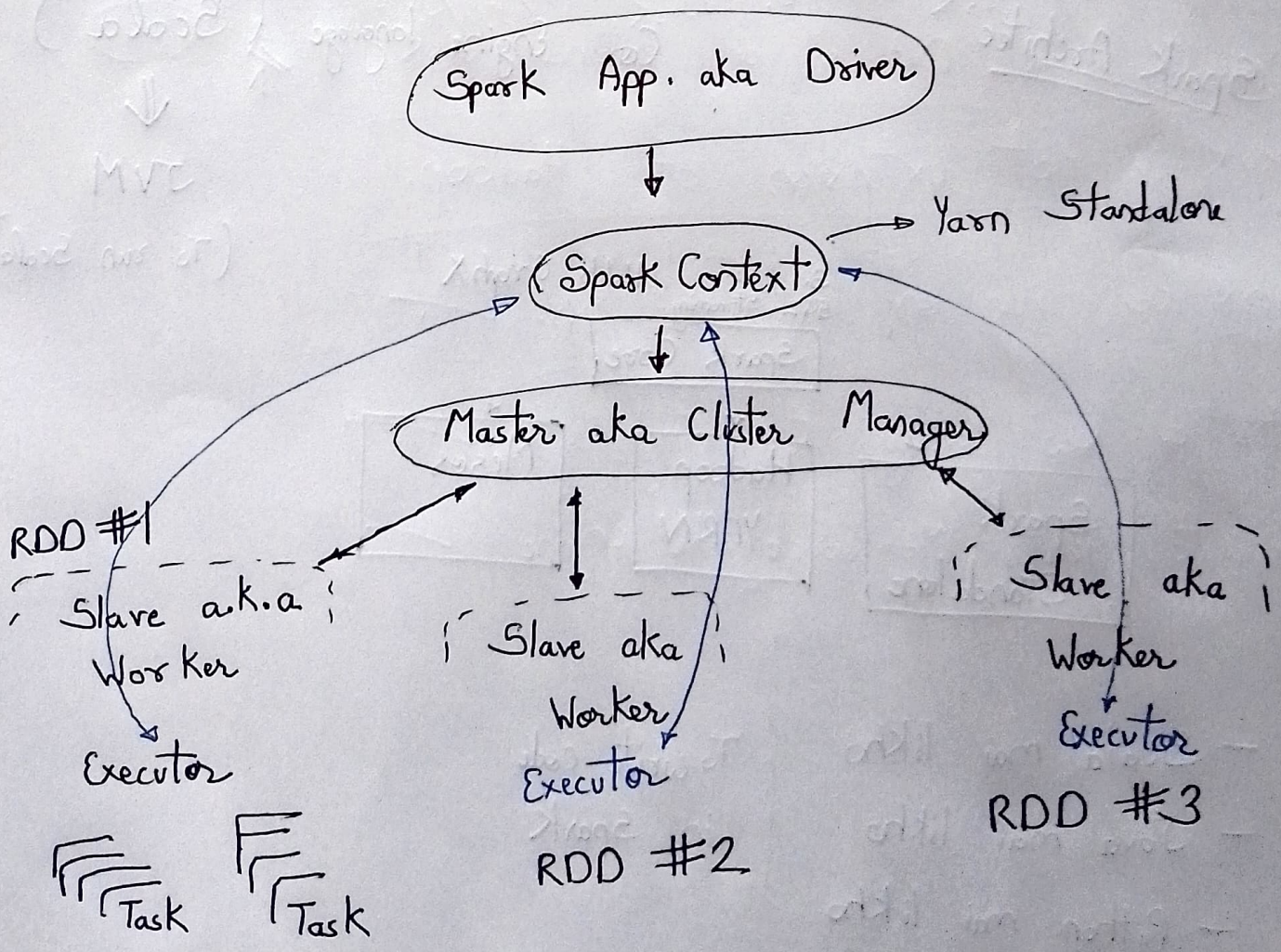
For real time we using Spark Streaming

For ML we use MLlib

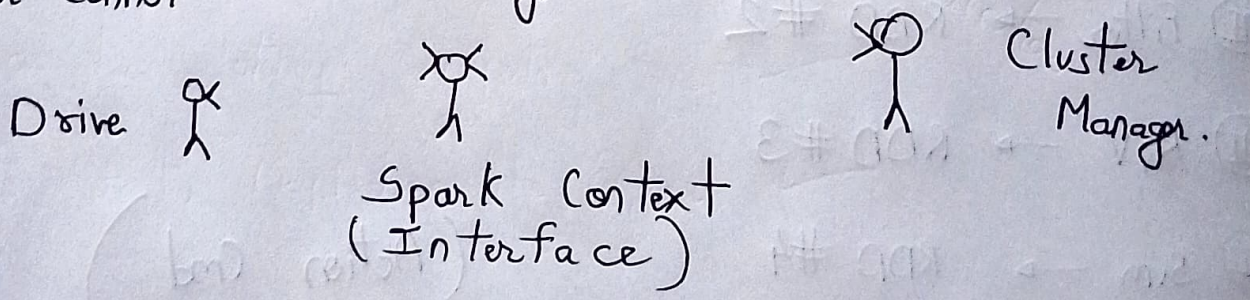
Graph X → Entity Relationship Framework.

Spark has different type of fault tolerance.

Spark Processing Part



- Jitne jyada Worker utna jaldi data process hoga.
- Cluster Manager provides resources to Worker
- Coordinator node manage the whole thing.
- (Coordinator node) Driver node reads data from disk and loads into memory of Worker node.
- Now before loading data ⇒ launch Executor (Container in Hadoop)
- Processing of data occurs in Executor
- Who will be launching Executor ?
  → Cluster Manager as told by Driver.
- You cannot talk directly.



- Cores decides no. of partitions or else you can manually decide it at time of loading
  So data is logically distributed not like HDFS.

So why logical data Partitioning ?

→ Cause data hum load Kar rahe hai toh 1/2 ya 1/4 data kyu load Kare.

- Range Partitioning } Type of
- Hash Partitioning } Partitioning

Shuru mai jab data load hota hai it is range P. but you process data using Hash P. as well

RDD (storage obj. in memory)

Resilent Distributed Dataset

Sara data jo memory mai store hota hai is in RDD

① Load → RDD #1

② Filter → RDD #2

③ K,V → RDD #3

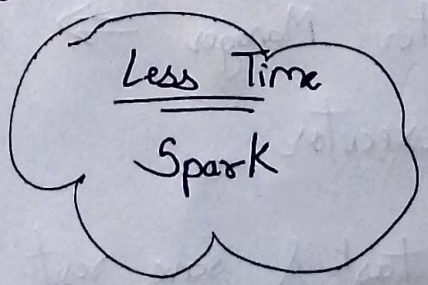
④ Sum → RDD #4 (Action Cmd)

⋮ ⋮

Jitna Task → Utna RDD ⇒ Display / Save

Action Cmd. ⇒ invokes driver ⇒ which can either display or store on ext. DBs

- Driver node is coordinator node.
- After O/P all the RDDs are deleted.
- Worker node is like blank machine where executor is launched.
- Spark uses Lazy Evaluation so only Action cmd invokes load → filter → 3rd process etc.
- Spark has 2 operation Transformation and Action Ek RDD se dusra RDD banana
- Idea is to consume memory for the shortest duration of time. but in that time memory consumption is very high.
- In MP everything occurs in stages.



or we can go for hybrid processing as well.

Fault Tolerance

- Either Worker node could fail or it could become slow.
- Immediately that Task & data will be launched on another worker node following DAG
- This is called as Speculative Copy.
- And the whole process is called Fault Tolerance.
- This is done cause data has to be collected collectively from all worker nodes.
- Driver stores all data like Lineage information.
- RDDs maintain lineage information that be based to reconstruct lost partition.

  (Resilient Distributed Datasets.
  (has to capacity to bounce back)

(I) Initiate Spark Context

(II) Driver ask Cluster Manager ⟹ launch executors

(III) Task send to executor

(IV) Executor own the task & save result.

↳ Spark Automatically deals with failure and takes o/p from either original or copy jo bhi aaya first.

↳ "Spark is lazy programming and RDDs are the primary abstraction which are resilient and fault tolerance in nature".

↳ If we don't do Hash Partitioning then we might have to do tons of reshuffling.

Mining Console logs

Load error messages from a log into memory, then interactively search for patterns

(RDD) (spark context) (function)

> lines = spark.textFile("hdfs://...") Base RDD

> errors = lines.filter(lambda s: ~~"Error" in s~~)
"Error" in s

Transformed RDD

All the errors are tab separated.

> messages = errors.map(lambda s: s.split('\t')[2])

> messages.cache()

> messages.filter(lambda s: "Recived Sgnl 15" in s).count()

Action Cmd

> Now all the data will be loaded and process will start after action command. is invoked.

> messages.filter(lambda s: "foo" in s).count()

Now this won't restart almost everything we cache those data and there's no need to load data again.

> Last mai you have to close the connection

> Toh idhar 3 RDD bane

lines, errors, messages