Spark

1. Most important topic
2. Restart
3. Interview ✓
   + work

~1 month work + 1 year

(Storage) HDFS → distributed storage, though cloud based gaining property

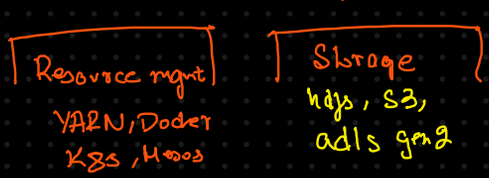
(resource manager) YARN → resource manager (Docker, Kubern, Mesos).

(Processing) MR → writing & performance ⟶ Spark

Some Common Questions.

Q:- Do I need to know hadoop before spark? → No

Q:- Do I need to install hadoop first? → No, Standalone



Q: Prerequisites before learning spark

→ Python, Java, Scala
→ SQL
→ hadoop essentials ⇒ distr. storage & processing

Q: Can you run spark on your laptop? ⇒ Yes

Q: Programming language?

Q: How is spark diff. from other databases? ⇒ Spark is not a dB

Q: What storage can you use with spark? ⇒ hdfs, s3, adls gen2, Kafka

Q: is spark free? ⇒ Yes

Android ⟶ OneUI

Q: What is diff b/w spark & pyspark?

Q: Can you use spark on cloud?

MR and its limitation

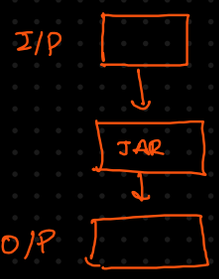
1. Performance → high latency
   disk i/o

2. Complex & hard to write :- hard to write boilerplate code
   verbose

3. Only batch processing is supported :- not stream processing

4. Rigid Sequence :- Everything has to be thought of interms of MR

5. No interactive mode/way to monitor jobs or run in ports?

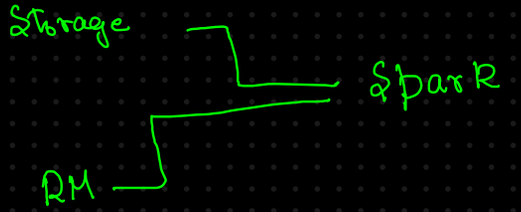


Spark

Spark 3

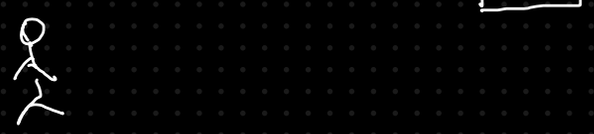
Apache Spark is an open-source distributed computing system designed to process large datasets quickly.

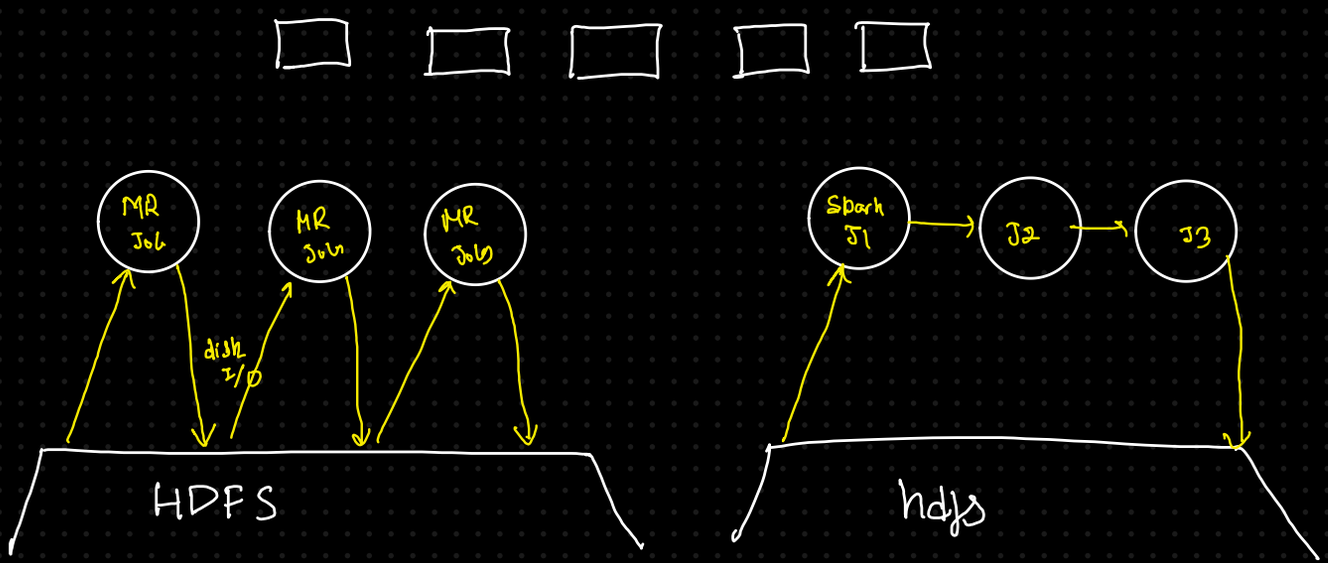
fast, general purpose & support batch & real time processing



Characteristic

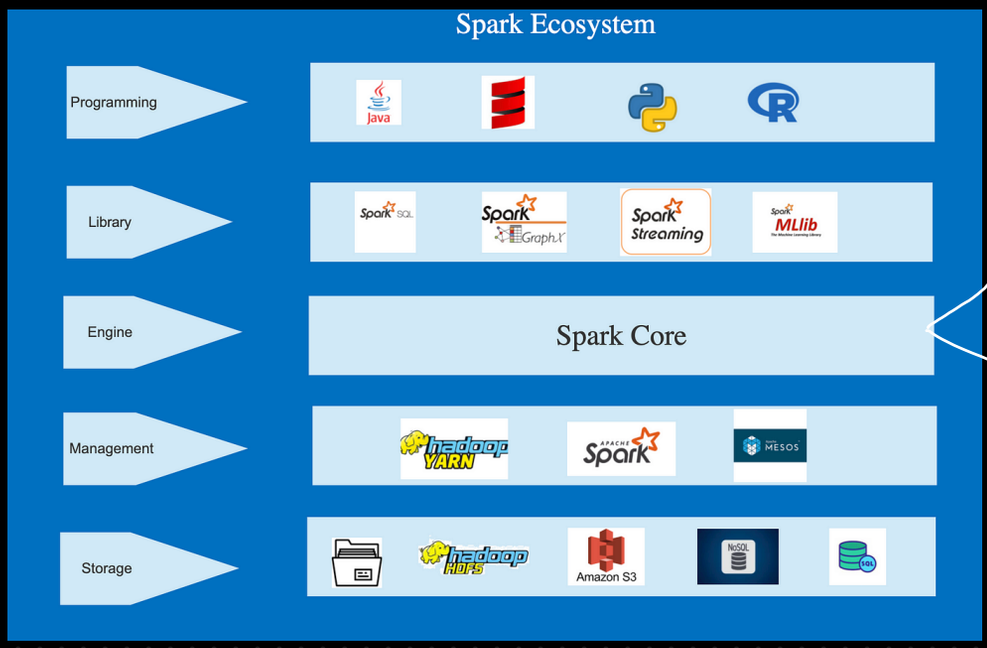
1. In-memory Processing: in-memory processing faster than traditional disk-based system





2. Ease of use → lang. support + lib

3. Unified Framework →



developed to make life easy (least flexible)

↑

Spark higher level

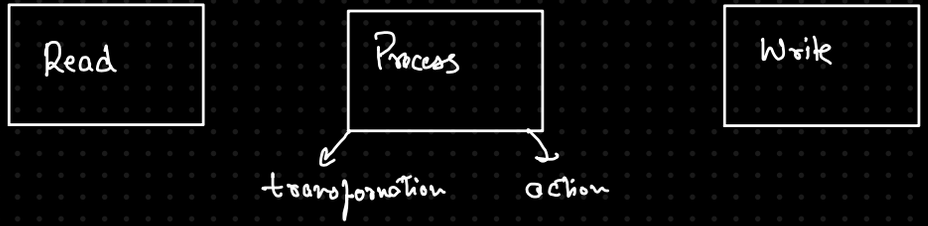
Spark core / low level

RDD (most flexible)

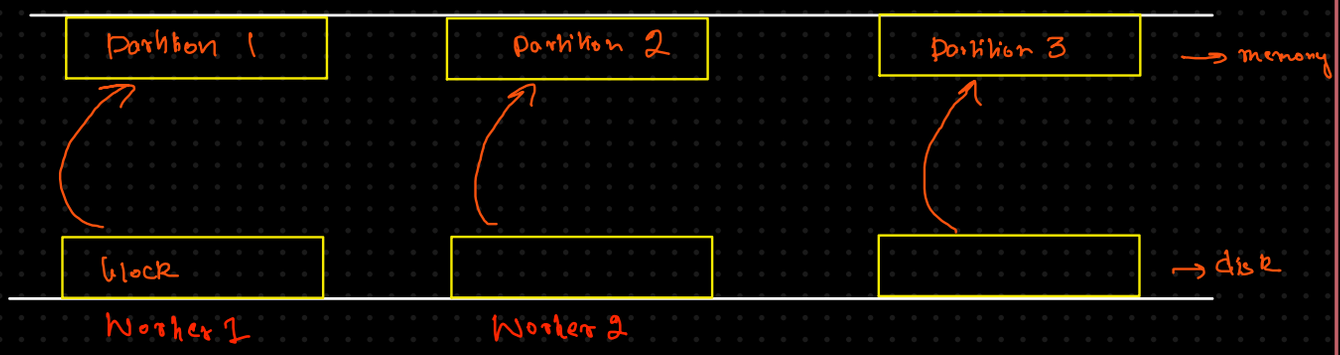
↓

Features of Apache spark

1. Speed : in memory processing, DAG,
2. Scalable : one m/c → 1000 m/c
3. Fault tolerant : automatically recovers the data in case of failure
4. Multi-language / Polyglot : multi lang support
5. Unified framework
6. Integration with existing systems : hdfs, yarn, Kafka



RDD & execution plan in Spark



data loaded from disk → memory

300mb ⇒ 3 blocks

block on disk
&
partition on memory

hdfs file → read in spark

↓

processing → Send it back to disk

rdd from some local data

rdd1 → → → →

filter, map...

↙ ↘

initialized data | hdfs

Nothing happened in our code unless we ran collect()

Lazy evaluation

Transformation

Opⁿ that create a new RDD from existing one.

Lazy

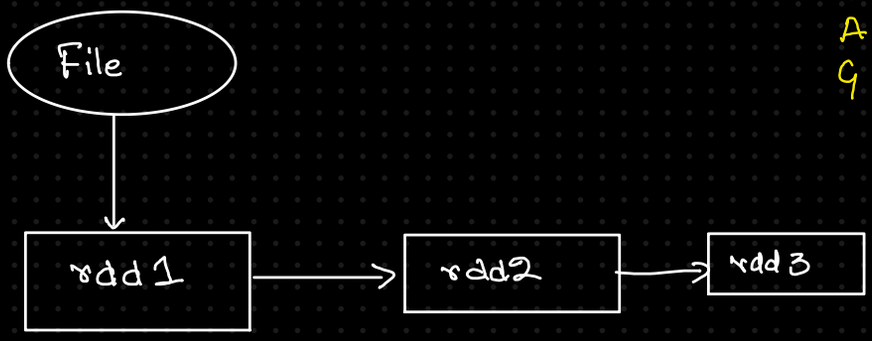
Action

Opⁿ that triggers the execution of all transformations & return result.

Eager

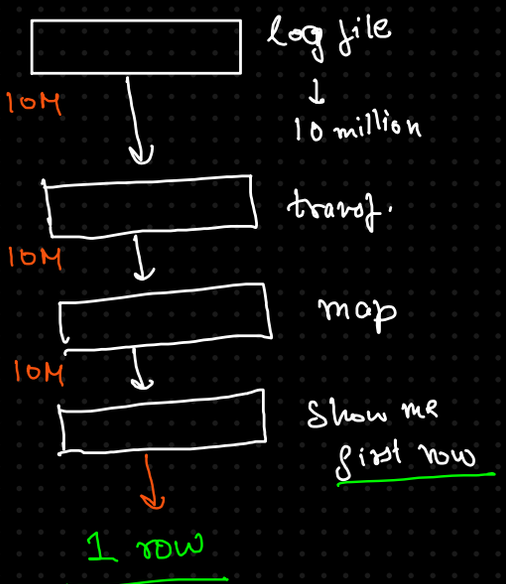
Cake

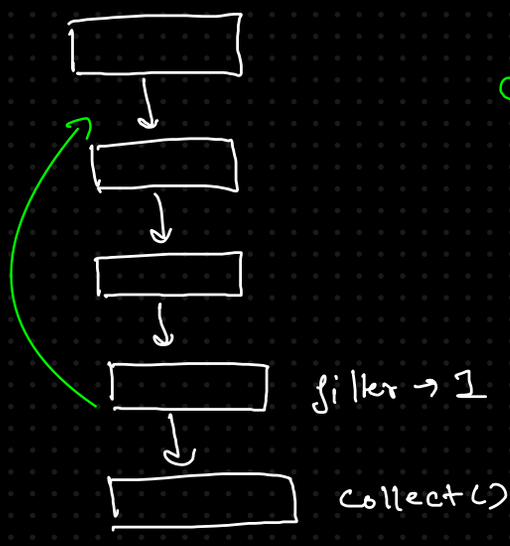
DAG / execution plan



D → Direct
A → Acyclic
G → Graph

Q: Why are transformation is lazy





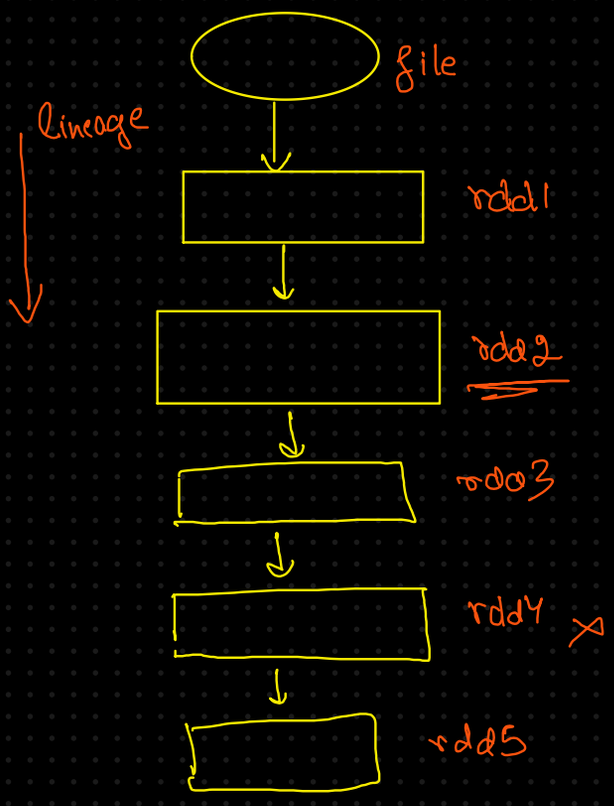
Transformations				
map	join	union	distinct	repartition
mapPartitions	flatMap	intersection	pipe	coalesce
cartesian	cogroup	filter	sample	
sortByKey	groupByKey	reduceByKey	aggregateByKey	
mapPartitionswithIndex		repartitionAndSortWithinPartitions		
Actions				
reduce	take	collect	takeSample	count
takeOrdered	countByKey	first	foreach	saveAsTextFile
saveAsSequenceFile		saveAsObjectFile		

R - resilient

D - distributed

D - dataset

Resilient will mean that we can quickly recover



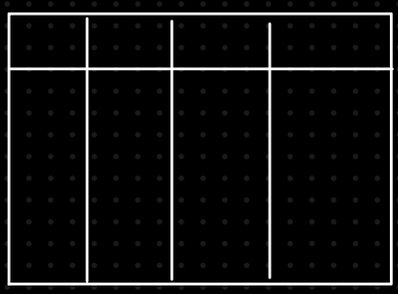
→ rdd are immutable

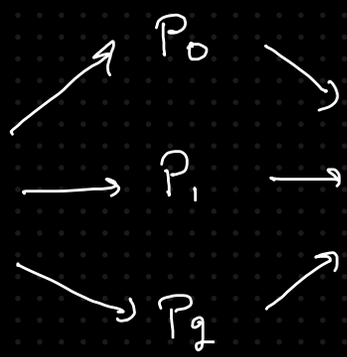
Properties of rdd

1. Immutable
2. Lazy evaluated
3. Partitioned

Characteristic

1. Foundamental data structure
2. Resilient
3. Fault Tolerance
4. Lazily evaluated





1. **SparkContext (sc)**:
   - It's the older entry point for Spark functionality
   - Primarily works with RDDs (Resilient Distributed Datasets)
   - Each JVM can only have one active SparkContext
   - More low-level control over Spark configurations
2. **SparkSession (spark)**:
   - Introduced in Spark 2.0 as the unified entry point
   - Encapsulates SparkContext, SQLContext, and HiveContext
   - Provides access to DataFrame and Dataset APIs
   - Can have multiple SparkSessions in the same application
   - Recommended for modern Spark applications
3. **"Normal" Spark**:
   - This usually refers to running Spark without specialized contexts
   - Basic functionality without SQL or Hive support
   - Limited compared to using SparkSession

Spark 2.0

Python & Spark ⟶ Pyspark

1. GCP Dataproc
2. Terminal of dataproc
3. Google Colab
4. Databricks

Python map reduce higher order fn

Reading & Partioning Data in Spark

Data is read and divided into partition

What is partitioning in Spark?

> Logical chunks of data

Processing

large dataset is divided into smaller chunks
called partition.

1. when already divided into blocks → hdfs, S3, lake gen 2

2. read data & create partition based on the configure.

execution of one partition of data = one task
= 1 cpu core required for execution

≈ Mapper task

(executor) performs a task & it can have more than 1 core as well

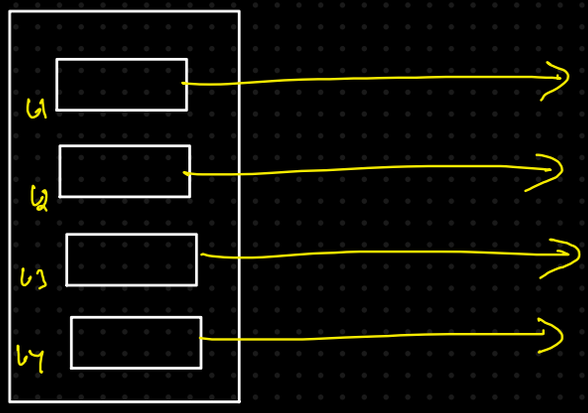
8 blocks (1gb) → 8 partition → 8 tasks

- inc. → i will get more ||ism
- dec → less parallelism

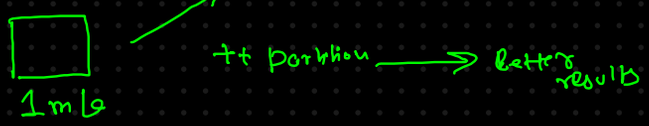
Spark parallelize

Subset of data

we can create an rdd using parallelize & then work on it



< 128 mb
↓
2 partition ⇒ 2 cores



→ Read the data → Parallelize & partition, count key value

→ Transformations → map, reduce, filter, reduceByKey

Reduce By Key

transformation

Count key value

action

Notebook → functions in pyspark

reducebyKey 5 keys → 5 values

map 100 rows → 100

filter 100 rows → 0 — 100

This is still the most diff. way of writing code in spark
⇒ more easy way

. Save as text file (hdfs)

Transformation in Spark : Narrow vs wide

transformations are operation that create a new rdd from existing one

Narrow

data is not shuffled across partition

faster → no shuffle

map, filter, flatmap

dependent on one parent partition

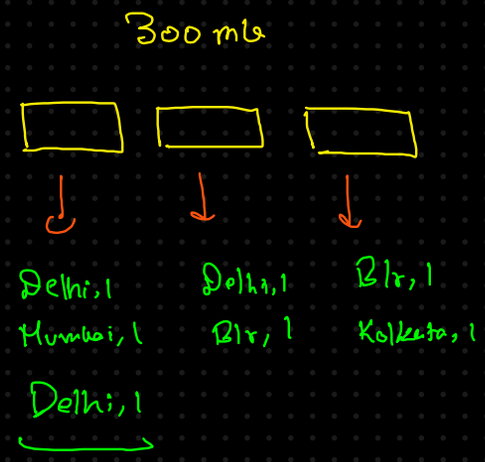
Wide

Data is shuffled across our partitions

Slower – as shuffling ⇒ I/O network

groupbyKey
reduceByK
SortbyKey

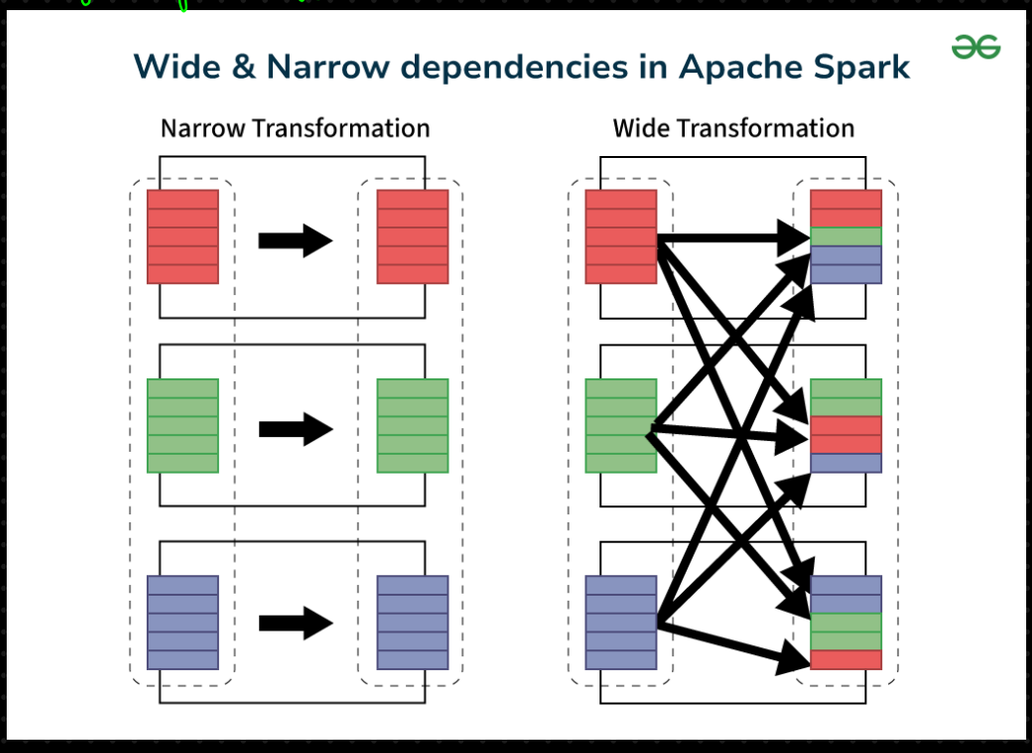
dependent on multiple partition



either

→ try to have very less wide transformation

→ narrow down the data so that data getting shuffled is less.



Job ⟶ Stage ⟶ tasks

||

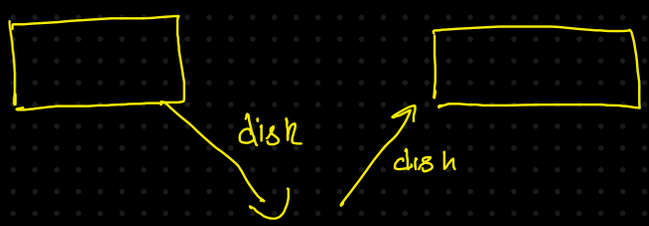
||

action ⟶ wide transformation ⟶ partitions

+

1

When we do a wide transformation ⇒ a stage is created



→ homework

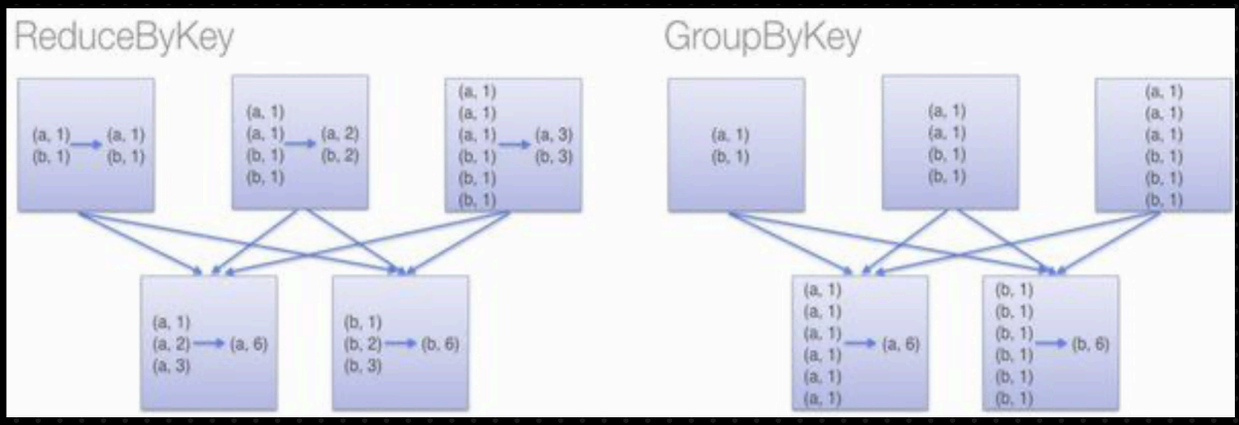
→ just run what I did & understand spark UI

→ be comfortable in pyspark

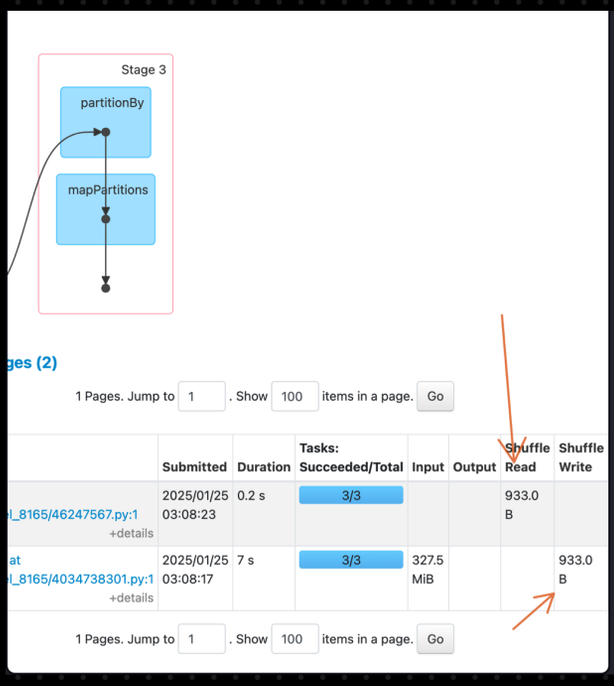
from next class, lots of code without major break

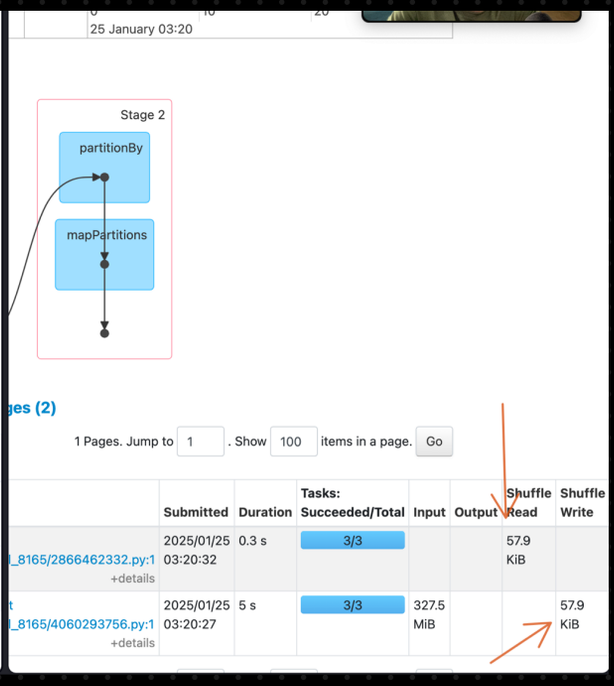
Reduce By Key & Group by Key

Definition part	Reduce by Key	Group by Key
	Aggregating the value of keys during shuffle phase	Group all the key values and we can aggregate on top of them



For say median, reduceByKey will give wrong answer
avg





Increasing and Decreasing No. of partition

↳ Repartition vs coalesce

Why would we need to increase or decrease no. of partition?

Increase

1gb data → 8

but say we have 12 nodes

We can increase the partitions here to use all nodes & get better ilism

Cluster should be able to handle

Decrease

Say we have 1tb file → 8192

& say we have 50 nodes cluster

⇒ decreas its better

⇓

8192 partitions

⇓ filter

Say 5000 partition of 1mb each

Sparse partition

Repartition

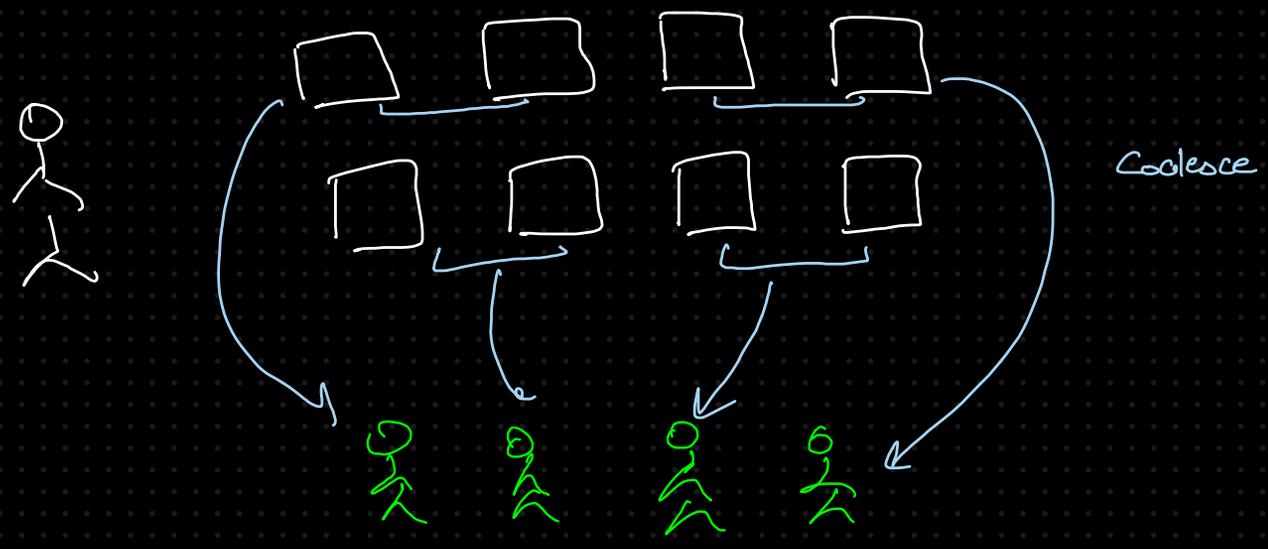
Can be used to increase or decrease no. of partition

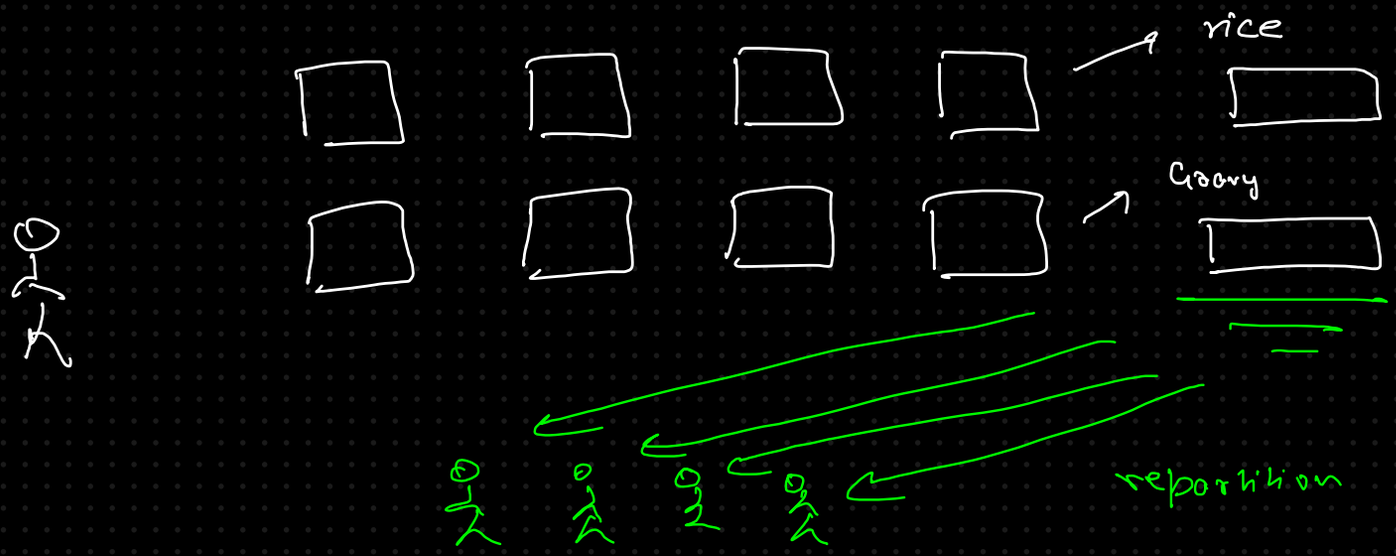
It does a full shuffle & then repartitions the data

Coalesce

decrease the number of partitions.

Tries to avoid shuffling while reducing no. of partition.





Aspect	Repartition	Coalesce
Definition	Shuffles data across nodes to **increase or decrease** partitions.	Combines partitions **without shuffle**, typically reducing their number.
Use Case	To balance load or increase parallelism.	To optimize performance by reducing partitions.
Shuffling	Yes, full shuffle across the cluster.	No shuffle; only adjacent partitions are merged.
Performance	Slower due to shuffling overhead.	Faster as it avoids shuffling.
Scalability	Used to increase partitions or even redistribute them.	Used only for decreasing partitions.
When to Use	After reading unbalanced data (e.g., uneven partitions).	When shrinking partitions after a filter operation.

Spark Higher level APIs

RDDs not used majorly.

much more than required

RDDs are core abstraction in Spark but they lack schema & metadata

1. Verbose
2. Error-Prone
3. Performance-based :

very imp for basics & interview prep.

Not persistent

schema metadata

2. What are higher level APIs

Dataframes & Spark SQL

no dataset in pyspark

& provide schema aware abstraction for data processing

Schema awareness ⟶ optimize query execution

SQL like programming

Integrate with other data sources (CSV, parquet, orm)

Dataframe

df is a distributed collection of data organized into named columns, similar to a table in a dB

rdd with some structure (schema) & metadata) data

not persistent

Feature	RDDs	DataFrames
Schema Support	No schema	Schema-aware
Ease of Use	Low	High
Performance	Less optimized	Catalyst-optimized

Spark SQL

module for structured data processing.

provides you an interface for running SQL queries on df & table.

SQL Queries

Spark table

- data (distributed)
- metadata/schema (dB)

Spark → process

Spark dataframe

spark.read. ⟶ format

header

infer schema ⟶ avoid this

Provide our own schema

load ⟶ path in hdfs

spark.read

action or transformation ?

Scenario	Behavior	Explanation
Without `inferSchema`, with `header=True`	**Eager Evaluation**	Reads only the first line to determine column names. Triggers a lightweight job (`collect` with `limit 1`).
With `inferSchema=True`	**Eager Evaluation**	Scans the entire dataset to infer schema. Triggers two jobs (1 for column names, 1 for schema inference).
With Explicit Schema Definition	**Lazy Evaluation**	No upfront job is triggered. Schema validation occurs when an action is performed on the DataFrame.

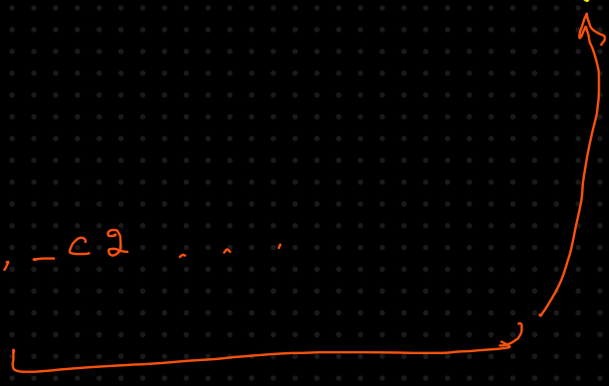
Schema enforcement in Spark df

Challenges with infer schema

- Incorrect inference
- Performance issues
- Bad for Production

1. headers → _c1, _c2 . . .
2. infer schema →

1. takes time ⟵
2. wrong



Schema Enforcement Techniques

1. Struct Type

Avoids incorrect inference
construct schema
Stricter checks → null

2. DDL string