Spark python Exam

- https://files.training.databricks.com/assessments/practice-exams/PracticeExam-DCADAS3-Python.pdf?_gl=1*14x1olh*_gcl_au*MTczODMwNzU0OS4xNjg3NDE1NTQy

spark preparation - https://www.databricks.com/learn/certification/apache-spark-developer-associate

exam registration - http://webassessor.com/databricks

Coalesce and repartition wrt partitions - shuffle

intro to spark Architecture - https://partner-academy.databricks.com/learn/course/112/introduction-to-apache-spark-archite[...]=299698&hash=7663f2e1b01db8ba7afea3ebe022d2609d8d323d
quick reference to spark architecture - https://partner-academy.databricks.com/learn/course/127/quick-reference-spark-architecture?g[...]=299698&hash=4b123c10e9c28529a8884ade004a6b093725d644

spark practice exams - https://files.training.databricks.com/assessments/practice-exams/PracticeExam-DCADAS3-Pytho[...].245159045.1632594692.1667888687-1188414711.1665672476

databricks workspace - https://portal.azure.com/#@manideepgrandhi2gmail.onmicrosoft.com/resource/subscriptio[...]ks/providers/Microsoft.Databricks/workspaces/dbcert/overview

adaptive query documentation - https://spark.apache.org/docs/3.4.1/sql-performance-tuning.html#adaptive-query-execution (edited)

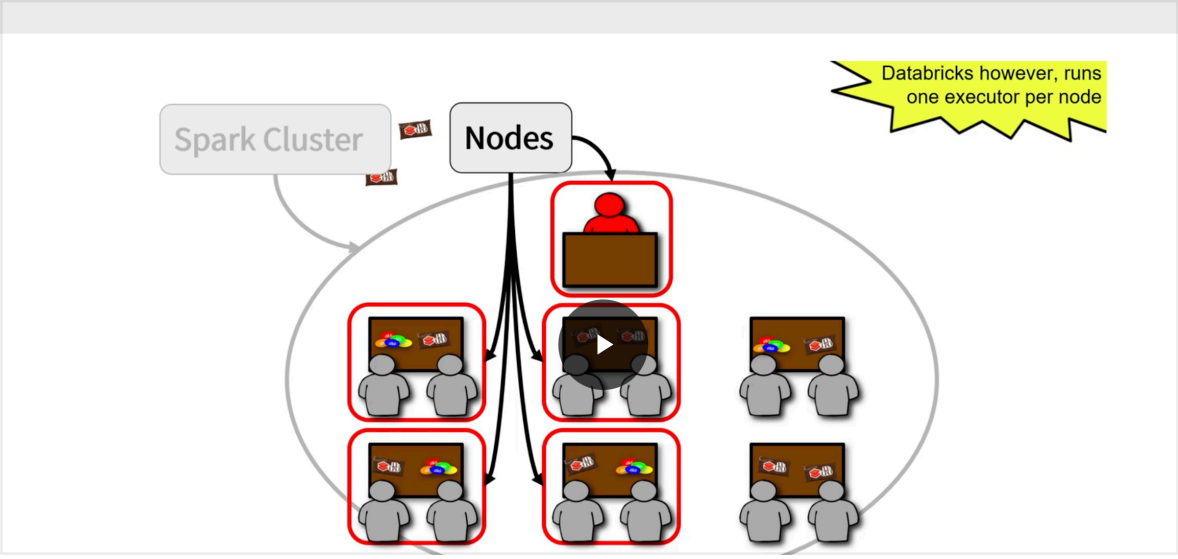
Introduction to Spark's Architecturerview

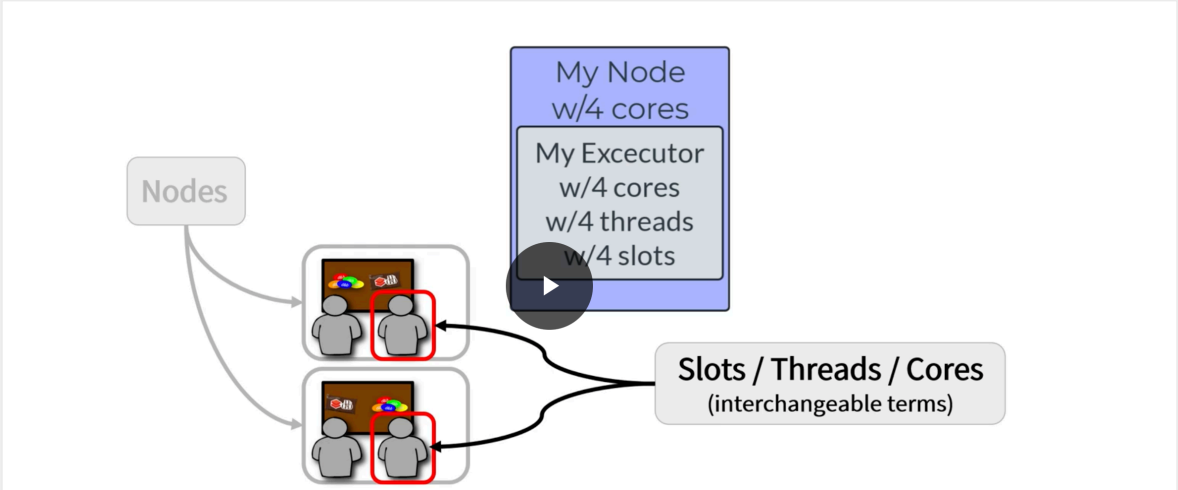
Key Concepts & Terms

This course also introduces the following concepts and terms:

- Shared resources
- Parallelization
- Partitions
- Jobs, Stages, and Tasks
- Drivers
- Executors
- Clusters & Nodes
- Slots/Cores/Threads

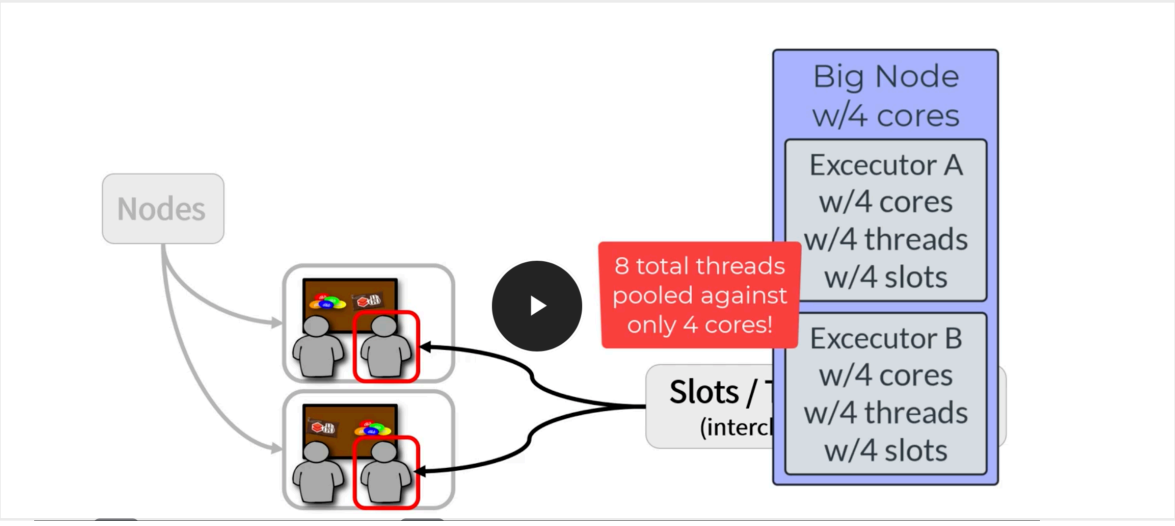
Driver	The instructor
Executor	The table at which students are working
Cluster	Collectively, the instructor, students, and their respective resources
Slots/Threads/Cores	The students
Partitions	The small bags of candies
Dataset	The larger bag of candies
Job	The request to eliminate all the brown candies & place the remaining candies on the table
Tasks	Specific instructions assigned to a single student to process a specific bag of candies



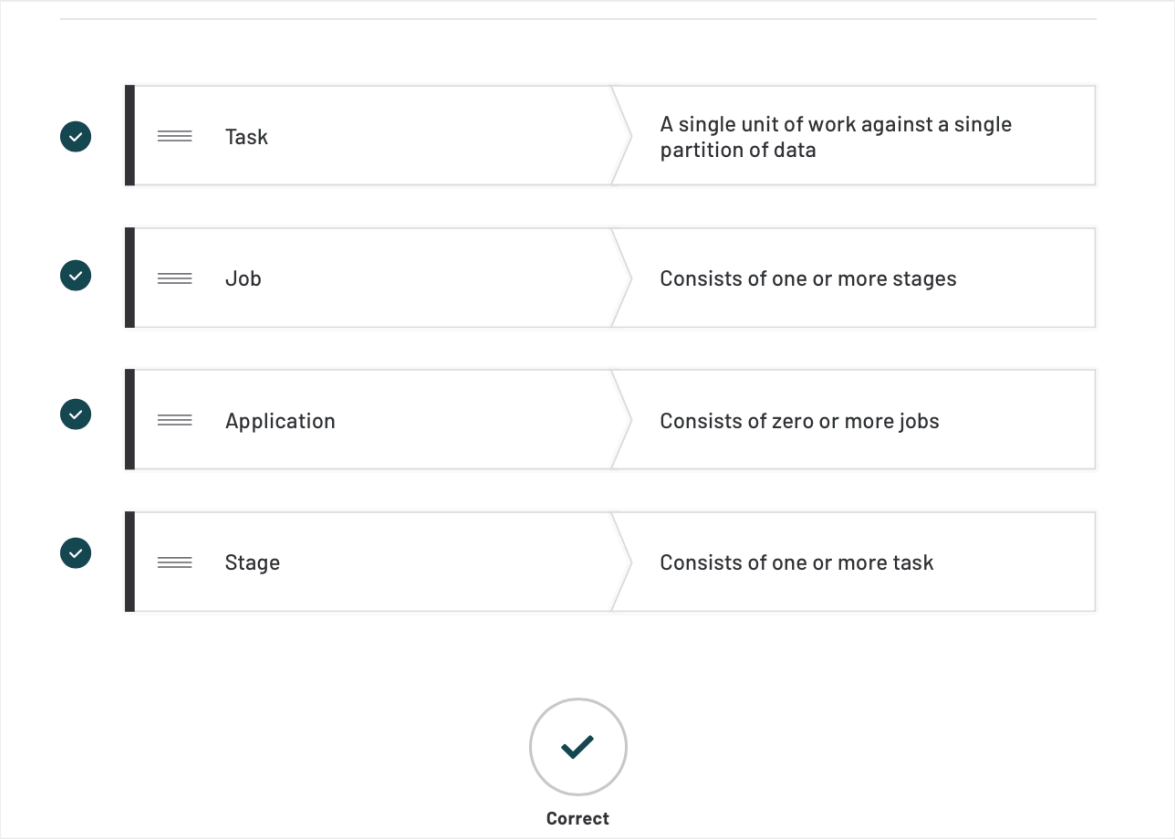


we are considering one executor in one node as it prevails
node has 4 cores . that means executor can run 4 threads . that means executor can utilise them as 4 slots.
Those threads in same executor share the resources(ram,disk,network).
The table (executor) can be considered as JVM space that thread(people) work on.

if we have 2 executors in big node as below, if it subscribe for 4 cores, we end up in 4 threads/slots for each executor.



The data is divided into partitions wrt to disk and a task is created for each partitions . Once partitions are set, they are never get divided .
The application we submit by spark submit - it reaches the spark driver..
Then it decides how many jobs to do .



Lesson Recap

An Introduction to Spark's Architecture

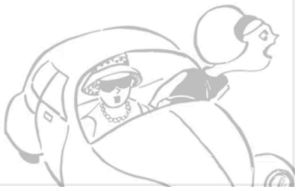
Spark Architecture Concepts: Recap

Clusters and Nodes

- A Cluster is a group of nodes
- Nodes are the individual machines within a cluster (generally a VM)
- With Databricks, the driver (a JVM) and each executor (each a JVM) all run in their own nodes

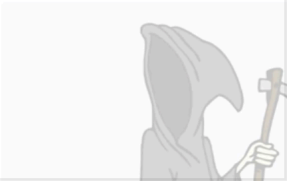
Driver

- Runs the Spark application
- Assigns tasks to slots in an executor
- Coordinates the work between tasks
- Receives the results, if any



Executors

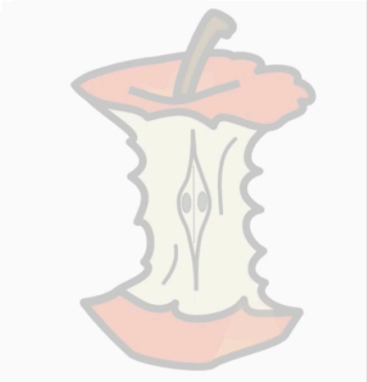
- Provides an environment in which tasks can be run
- Leverages the JVM to execute many threads



Executor is in a Jvm which is designed to take workload from driver.

Slots/Cores/Threads

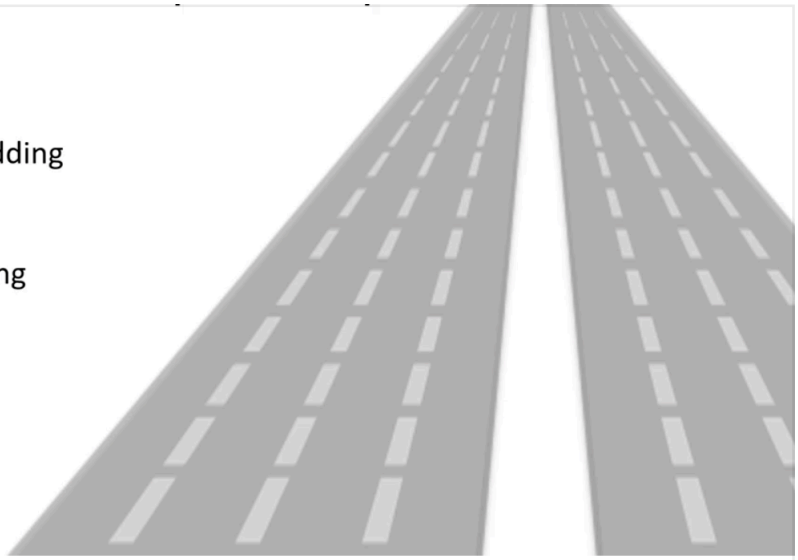
- The lowest unit of parallelization
- Generally interchangeable terms, but “slot” is the most accurate term
- Executes a set of transformations against a partition as directed to by the driver

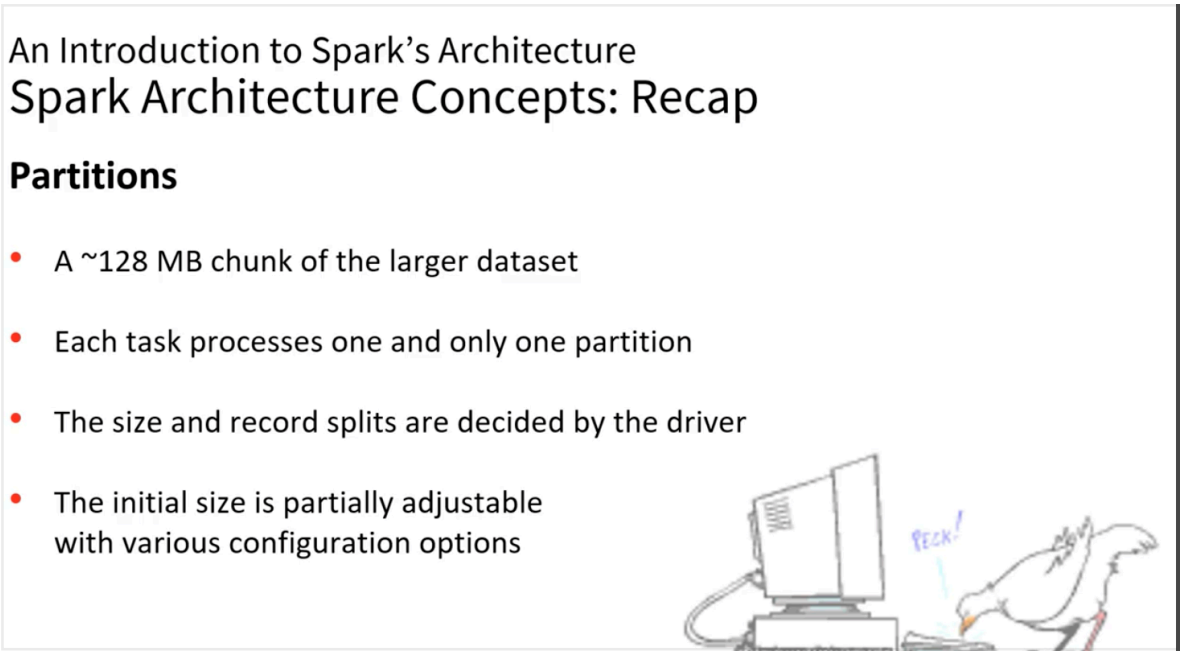


Driver sends a partition (data is divided into partition which is decided by driver) to a task. The task is assigned to a slot (core/ thread) in executor . The transformations sent by driver are applied on that partition.

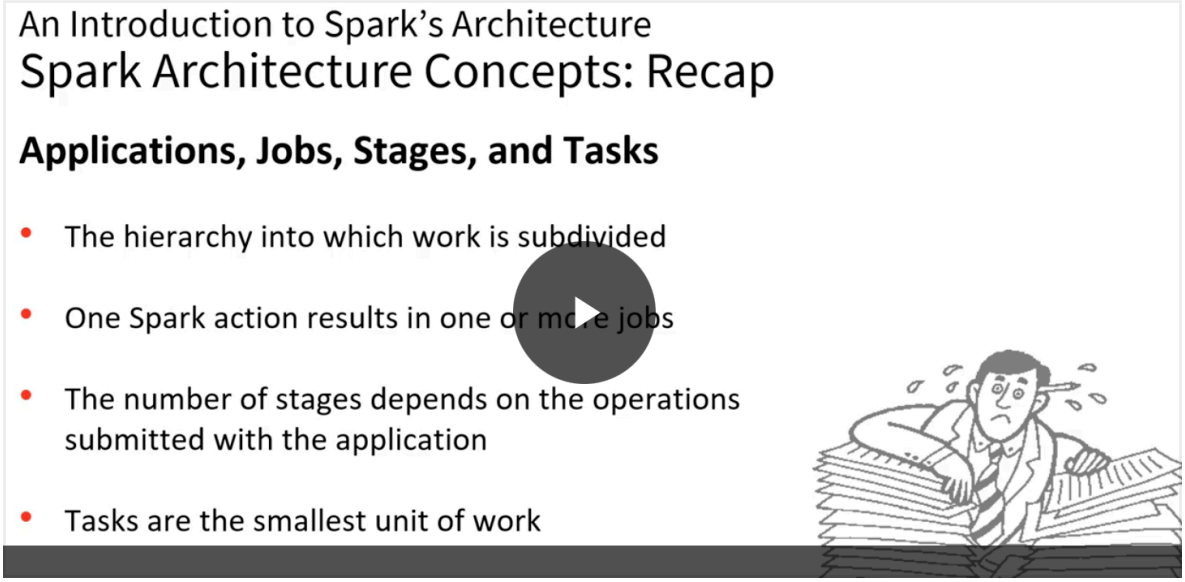
Parallelization

- Scale horizontally by adding more executors
- Scale vertically by adding cores to each executor





The data split into partitions are decisioned by Driver . There is 1-1 relation b/w the task and partition.



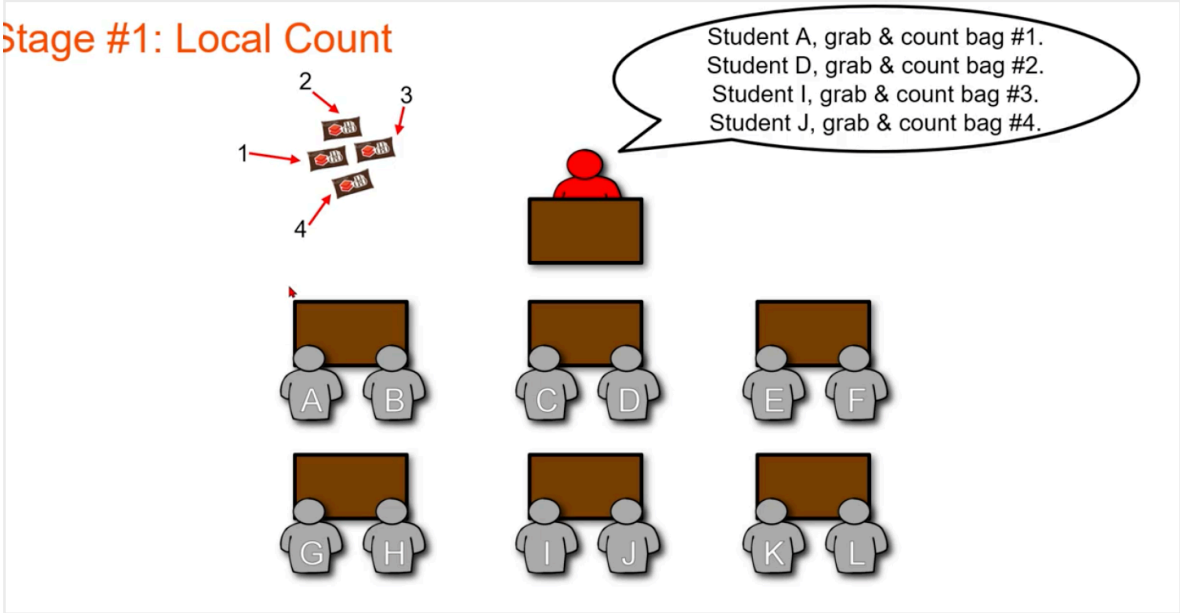
when the application is submitted to driver (via spark-sbumit or normal way), the driver identifies the no of jobs .

STAGES

Counting

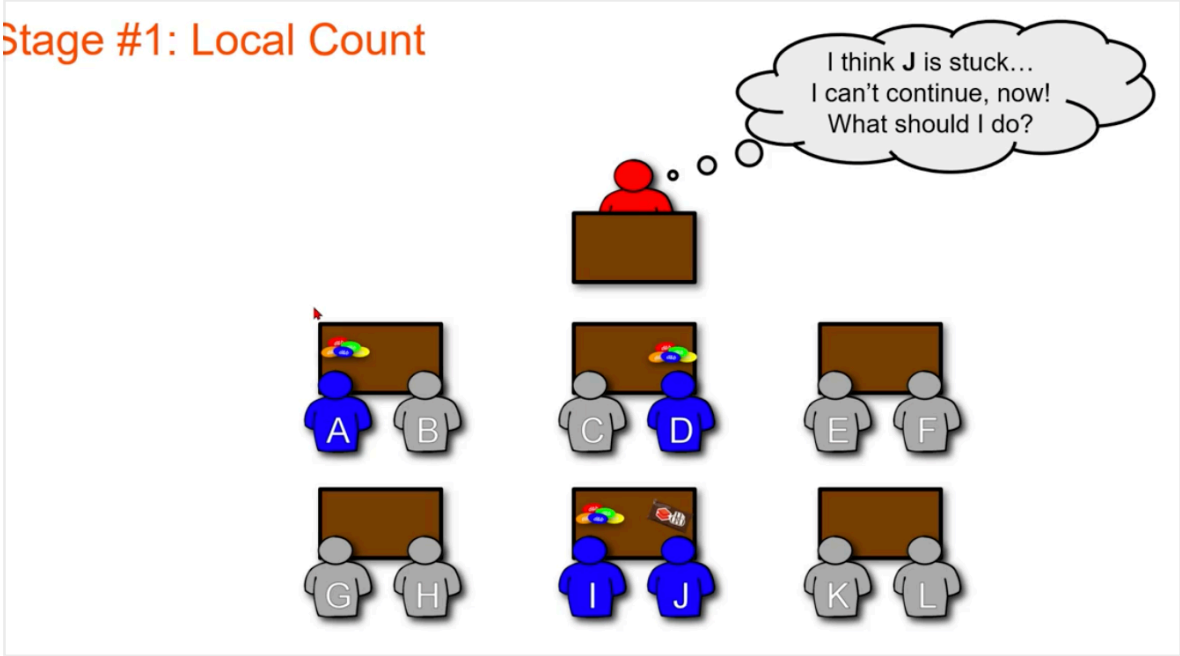
In this lesson we take a look at how Apache Spark breaks up a single job into multiple stages. In doing so we see, by example, the basic patterns used to solve increasingly more complex problems.

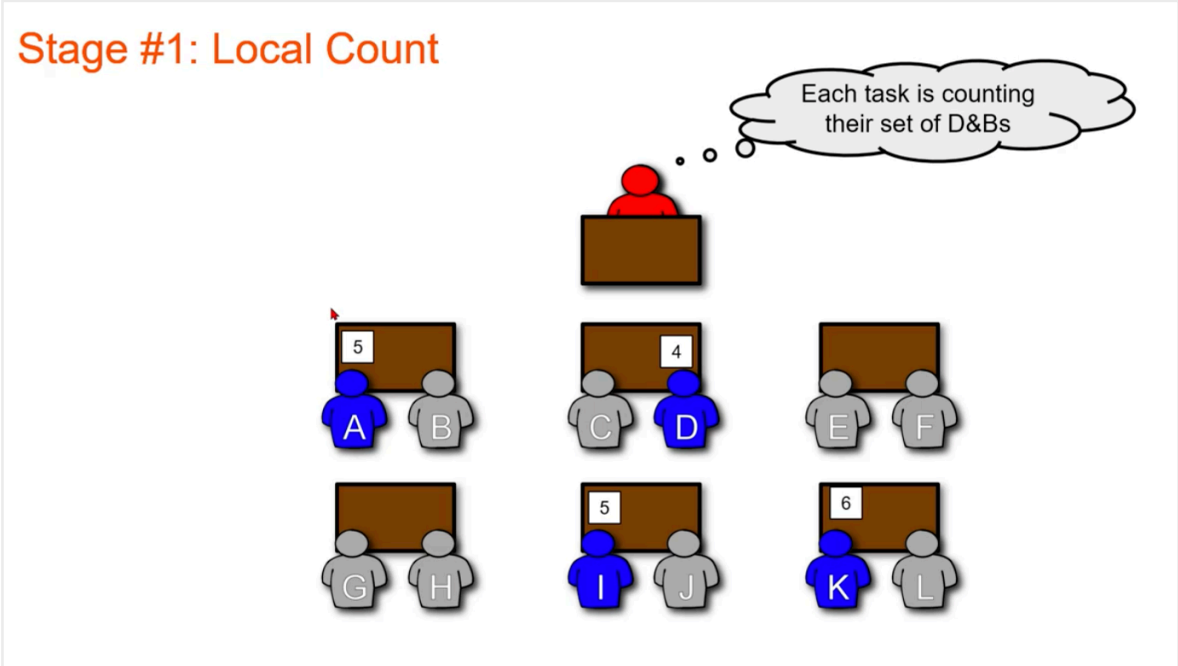
In below, Driver identfies the slots/threads/cores and gave them the tasks to count



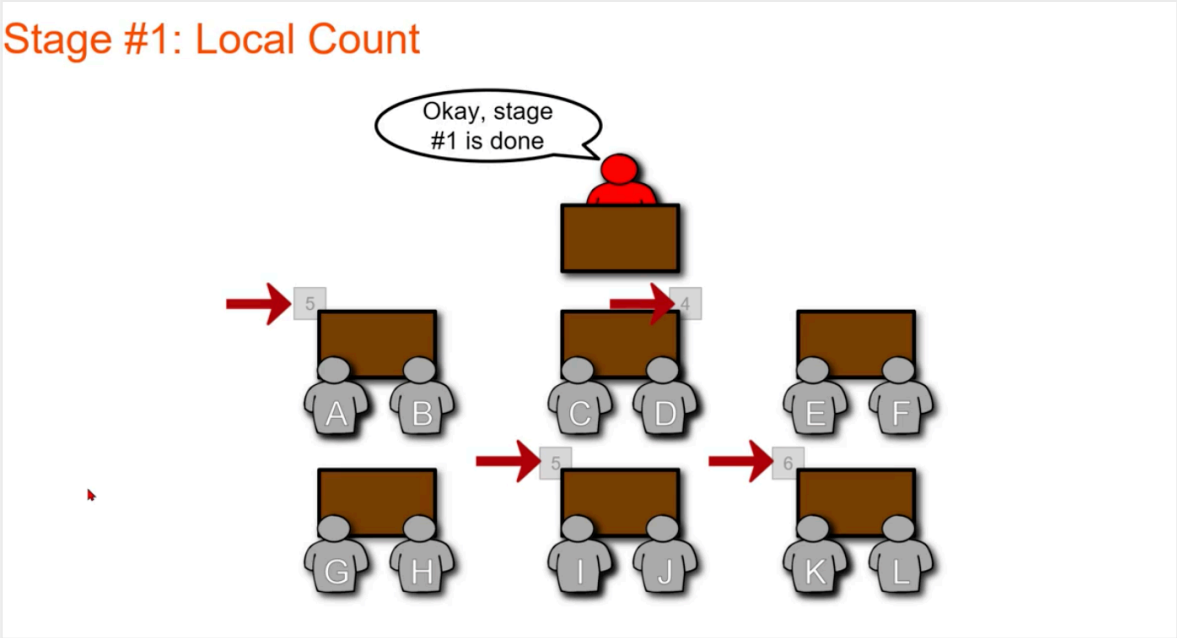
In below, there may be heartbeat connection b/w driver and J / J is not able to work. so Driver cancels J and reassigns the task to k (for example) .

The task can be executed in two slots by two threads, the driver takes the result whoever completes it first (this is called speculation execution)

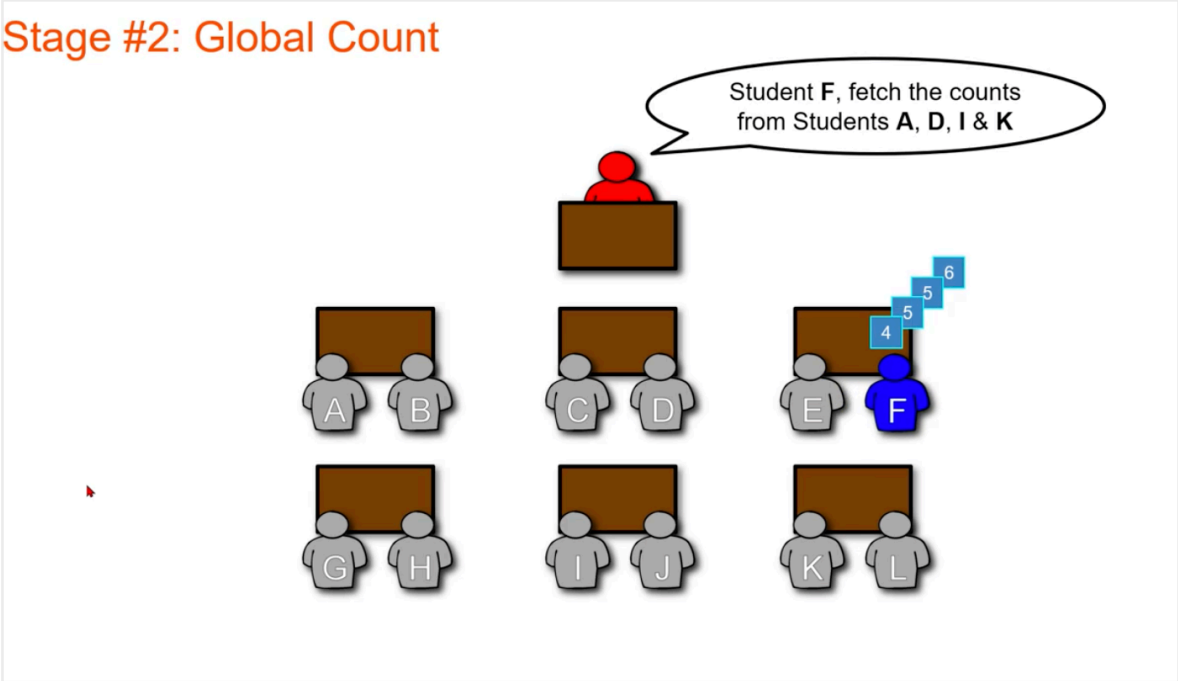




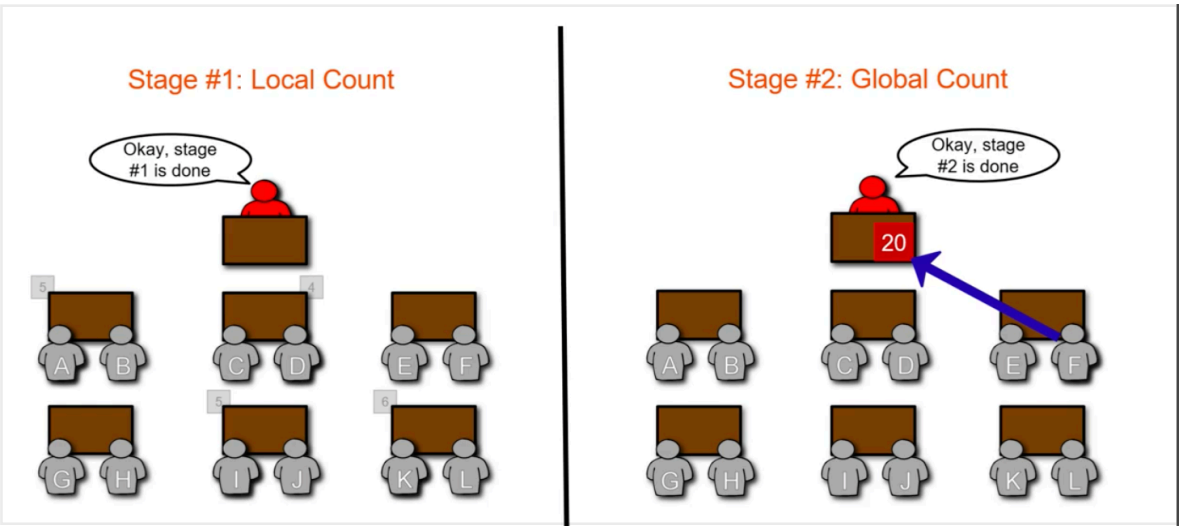
In below, the count values are saved in shuffle files for other stage as the driver is not allowed to touch the data .



Now the driver gives the a tasks to F, to fetch the data from remaining slots/cores/threads.The F sums up the total and returns the result to driver.



Stage 1 and stage 2

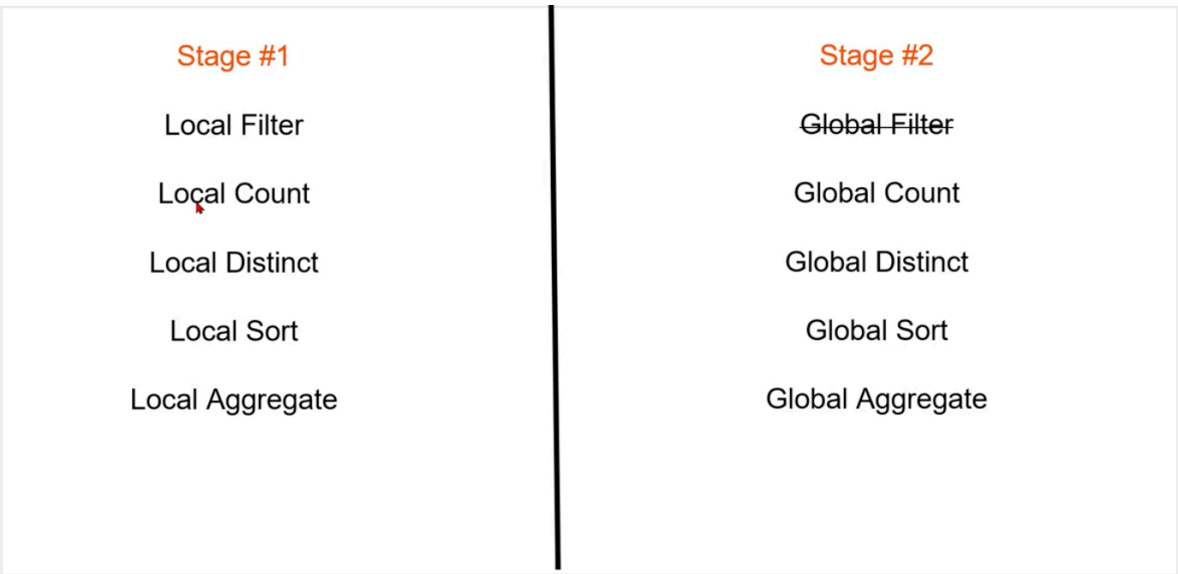


There is stage boundary between two stages. If any task is failed or running continuously, the thread F is waiting for the data from the stage .

In below,

local filter in one stage doesnt require global filter as local filter removes all the occurences in every task, so the filter at next stage is not needed.

where as for other funcs like count,Distinct, - the count or distinct is needed at later stage respectively as the results from individual threads/cores are needed to be combined to get overall result.



In our example of counting our D&B candies, which of the following operations (on the left) best describes the event (on the right)?

Count all the D&Bs	Job
Produce the local count	Stage 1
Produce the global count	Stage 2
Count the number of D&Bs in a single partition	Stage 1's four tasks
Sum all of the local counts to produce the global count	Stage 2's only task

Which transformations must Apache Spark execute with multiple stages?

- ✗ None, no transformation "requires" multiple stages, it is simply an optimization strategy
- ✗ All transformations must be executed in multiple stages to achieve distribution of the compute across the entire cluster.
- ✗ Only those transformations that can be executed with disregard to any of the other data in the global dataset
- ✓ Only those transformations that ultimately require knowledge of the entire dataset

Which transformations must Apache Spark execute in only one stage?

- ✗ None - there is no transformation that can be parallelized across a cluster in a single stage
- ✗ All transformations must be executed in a single stage, otherwise, the processing cannot be parallelized across the cluster.
- ✓ Only those transformations that can be executed with disregard to any of the other data in the global dataset
- ✗ Only those transformations that ultimately require knowledge of the entire dataset

✓

Correct

Transformations that can be executed in a single stage are generally referred to as “wide” transformations

- ✗ True
- ✓ False

Correct

TAKE AGAIN

Transformations that must be executed across multiple stages are generally referred to as “narrow” transformations

- ✗ True
- ✓ False

In a job with multiple stages, subsequent stages can be started if the majority of the tasks in the previous stage have completed

- ✗ True
- ✓ **False**

✓ Correct

In a job with multiple stages, the execution of the secondary stages can be delayed by a single, slow-running task in the previous stage?

- ✓ **True**
- ✗ False

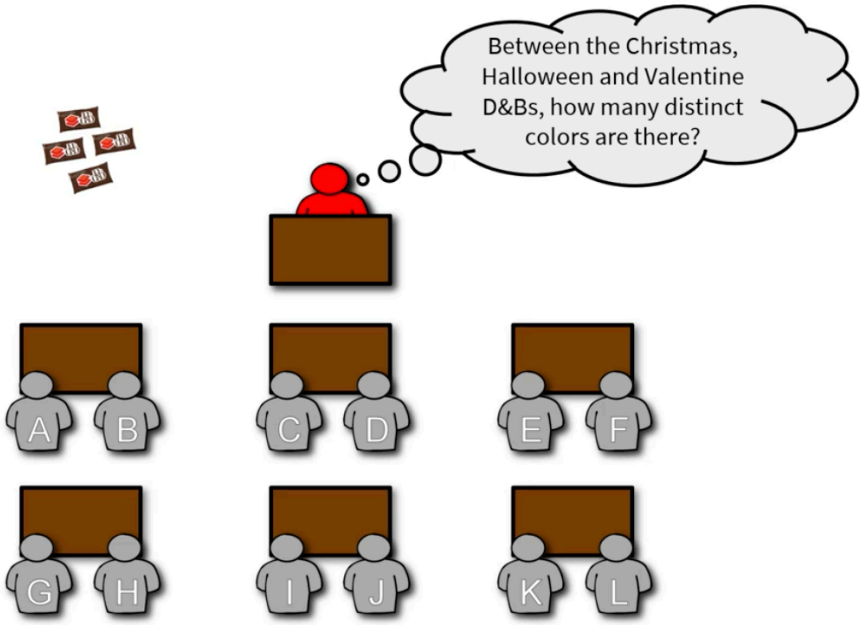
✓ Correct

Recap

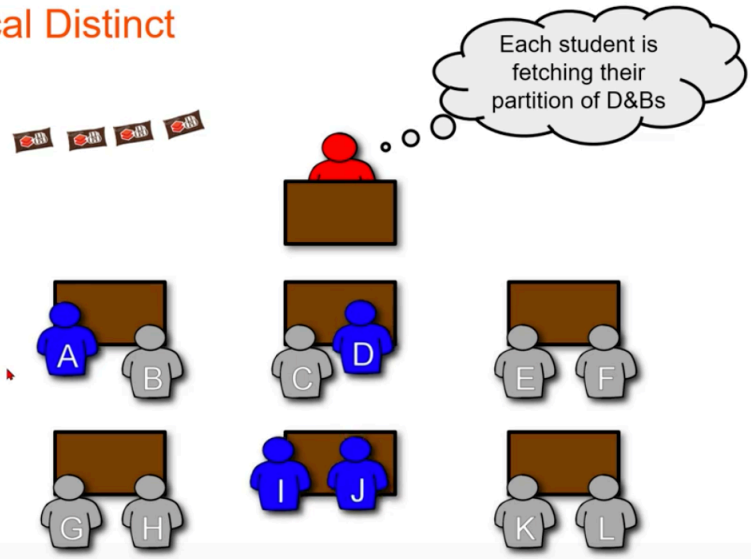
spark job can avoid shuffle-join and do broadcast join as we see in adaptive query execution.

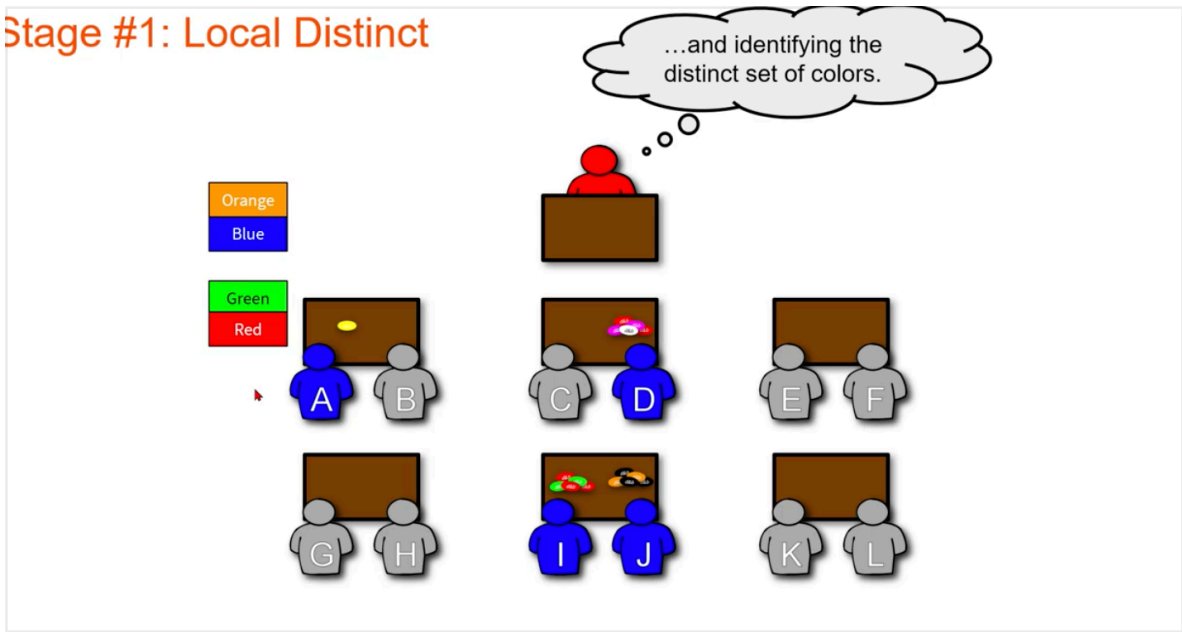
Shuffling - we learn abt wide transformations

Stage 1 - Local Distinct

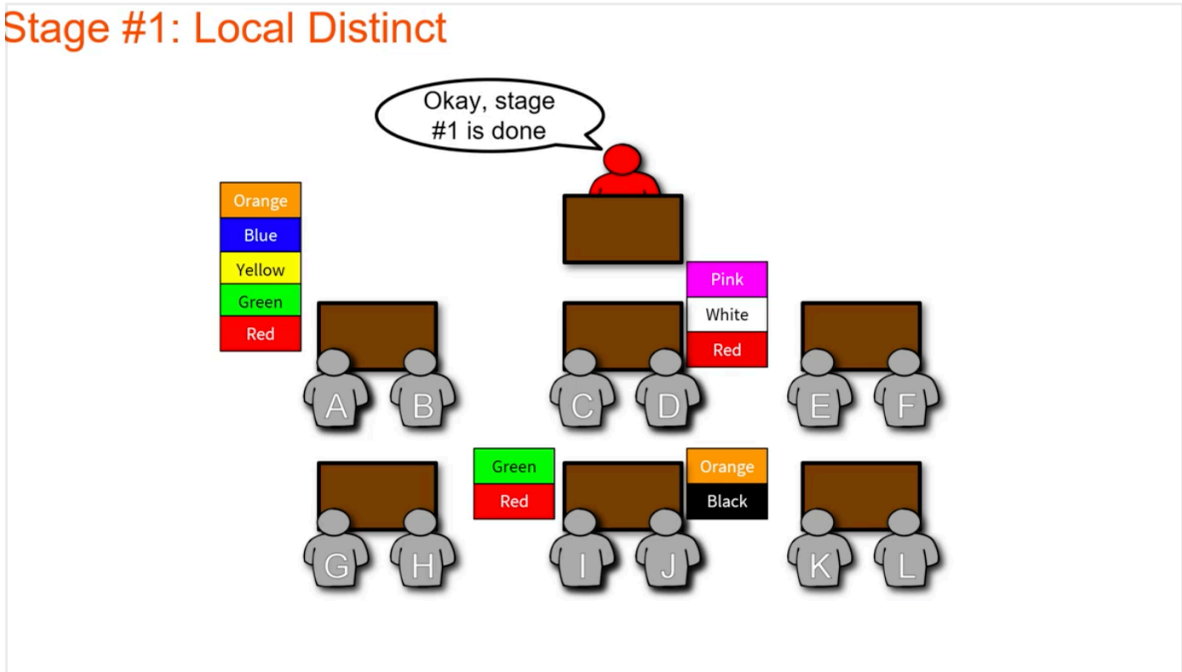


Stage #1: Local Distinct

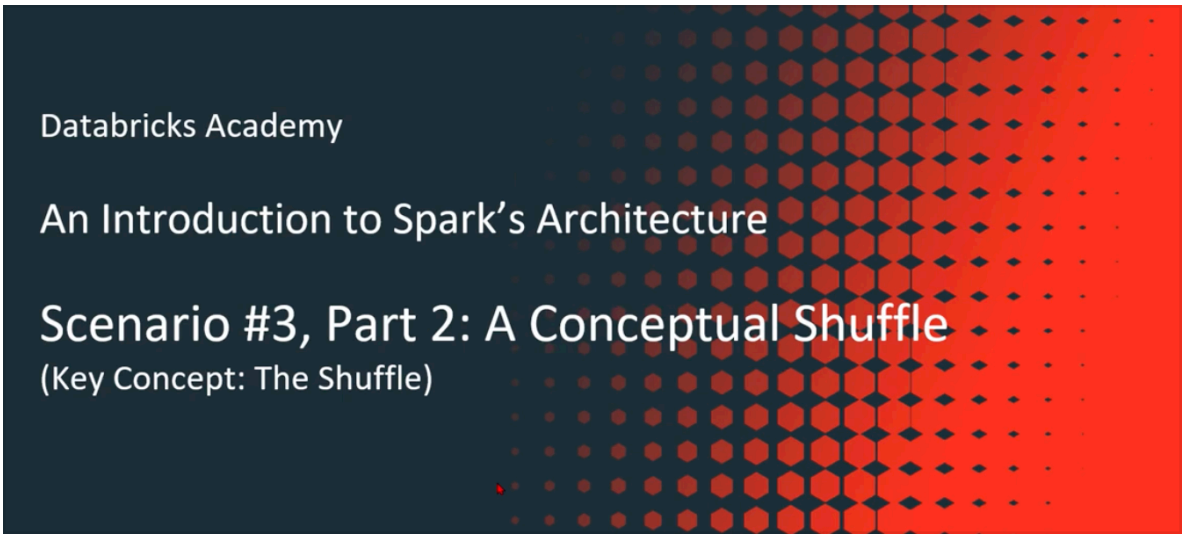




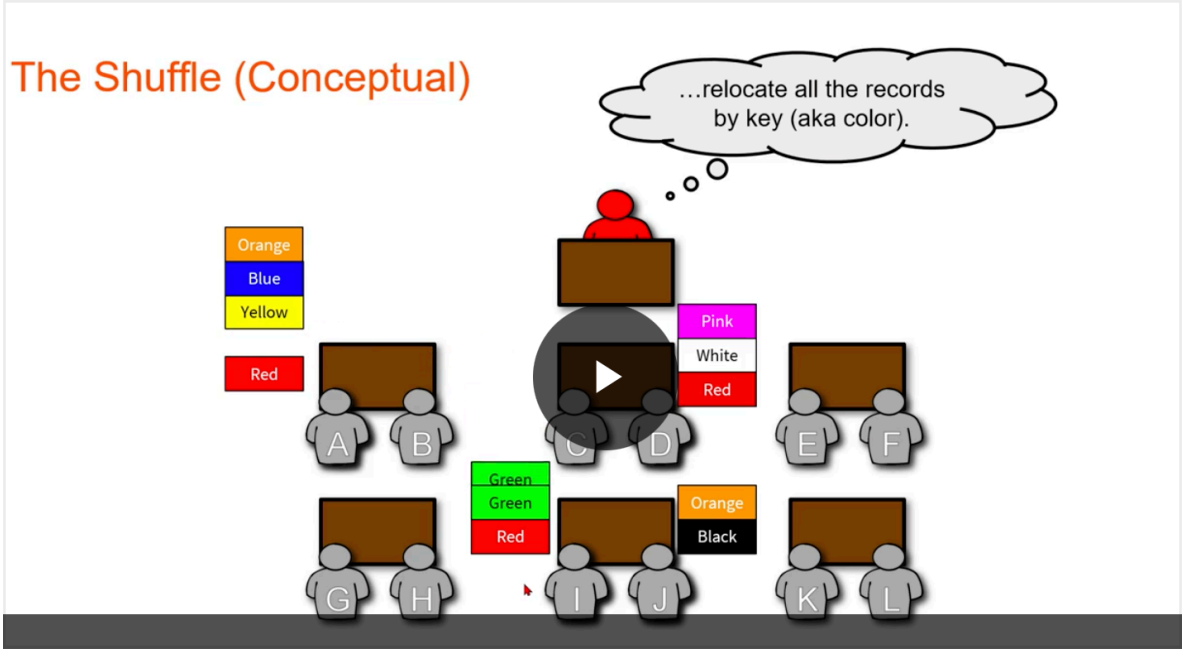
AND each task determines the different colors in its partition. which is stage one
The data from stage 1 is filled in shuffle files (as below with color stack)



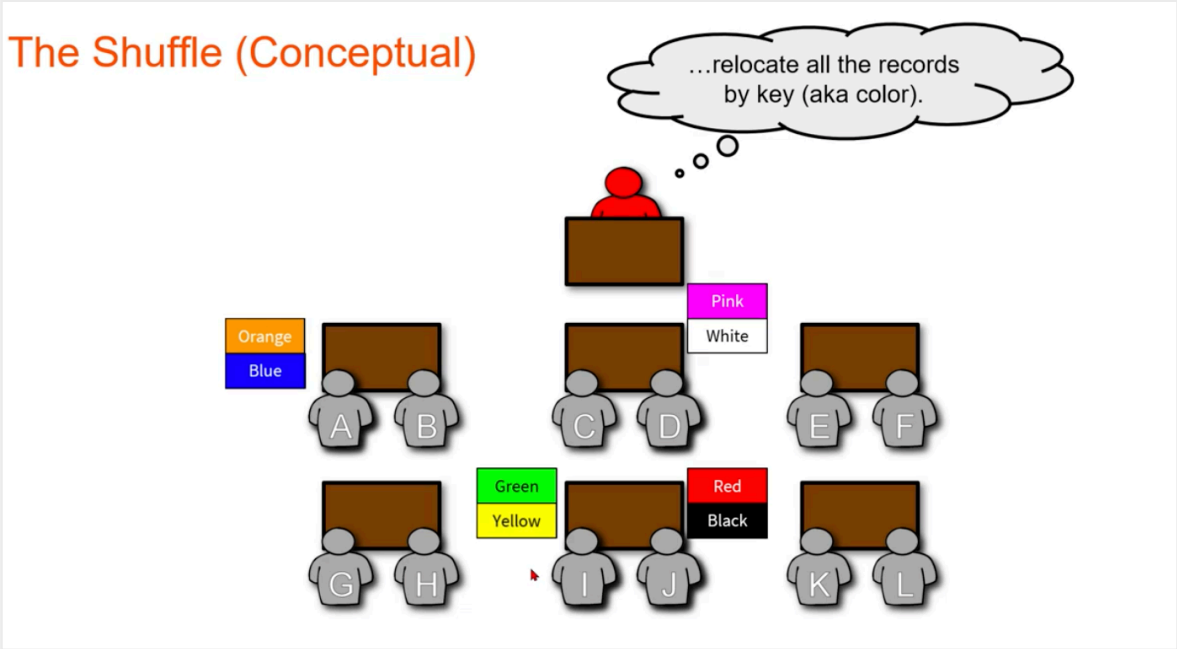
SHUFFLE PART 2



In stage 2, relocate all the colors so that same colors are combined. (data is moving in the network)
In below, the colors are moving (not completed) to get combined .



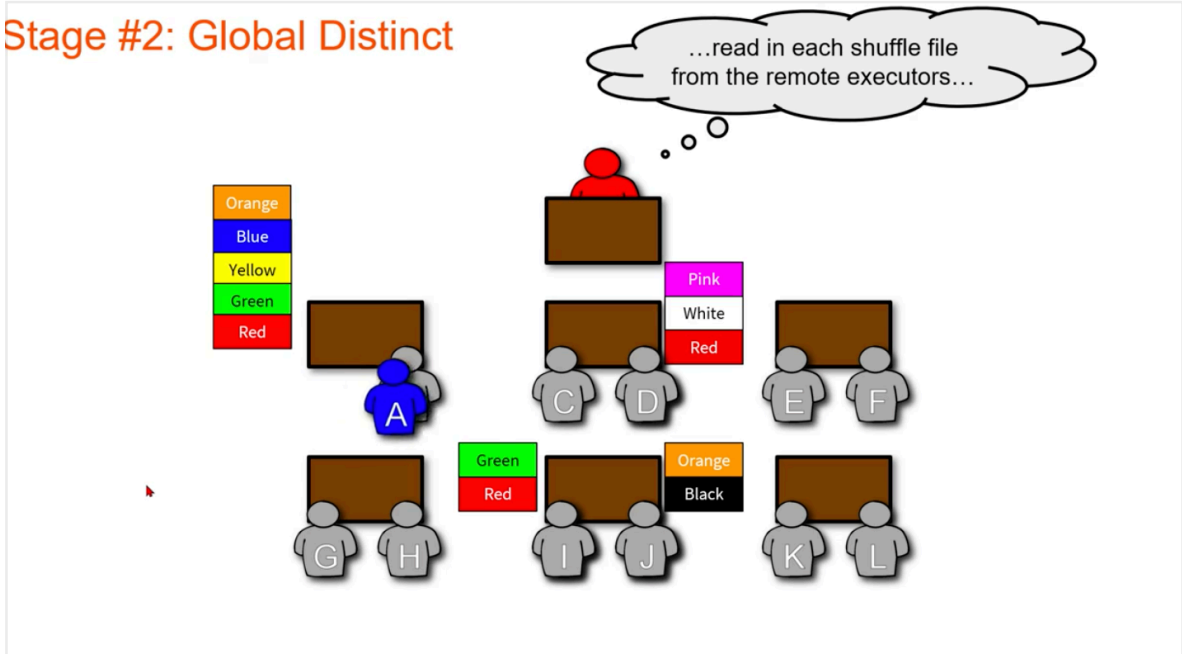
The Resultant shuffle :-



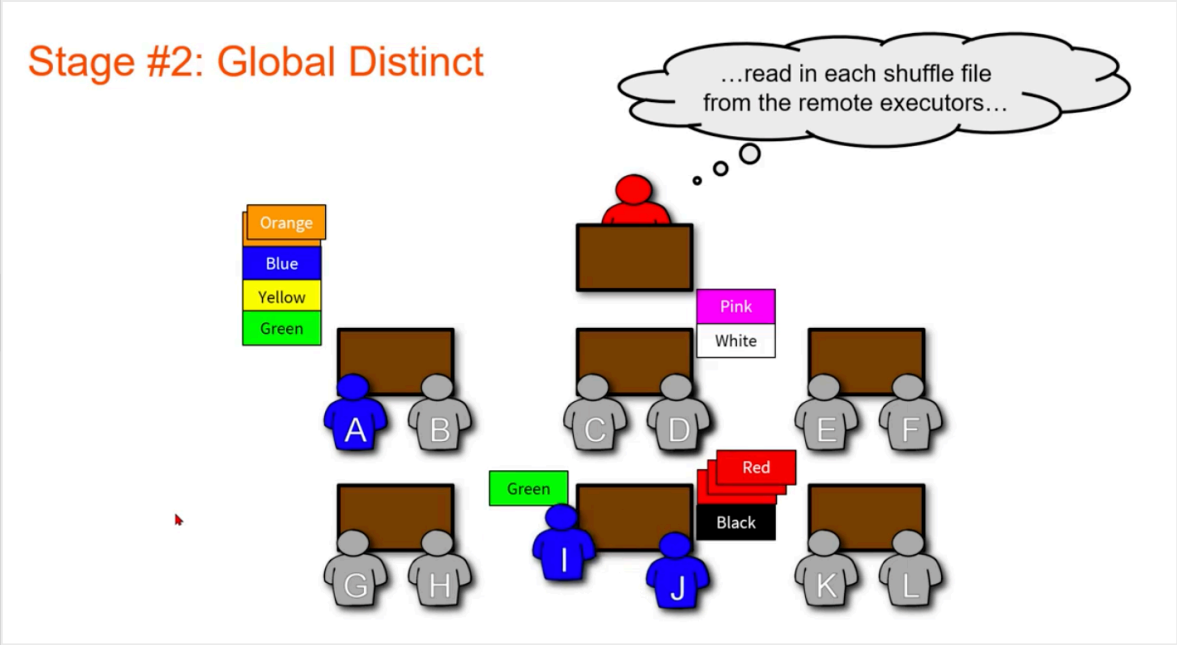
Global Distinct (stage 2)

In this stage 2, data is on different executors.The tasks pick up the distinct colrs from other executors and finally send to the driver.

from below pic, the A goes to J to pick orange from J.
J goes to D ,A, and I to pick red



the result will be as below

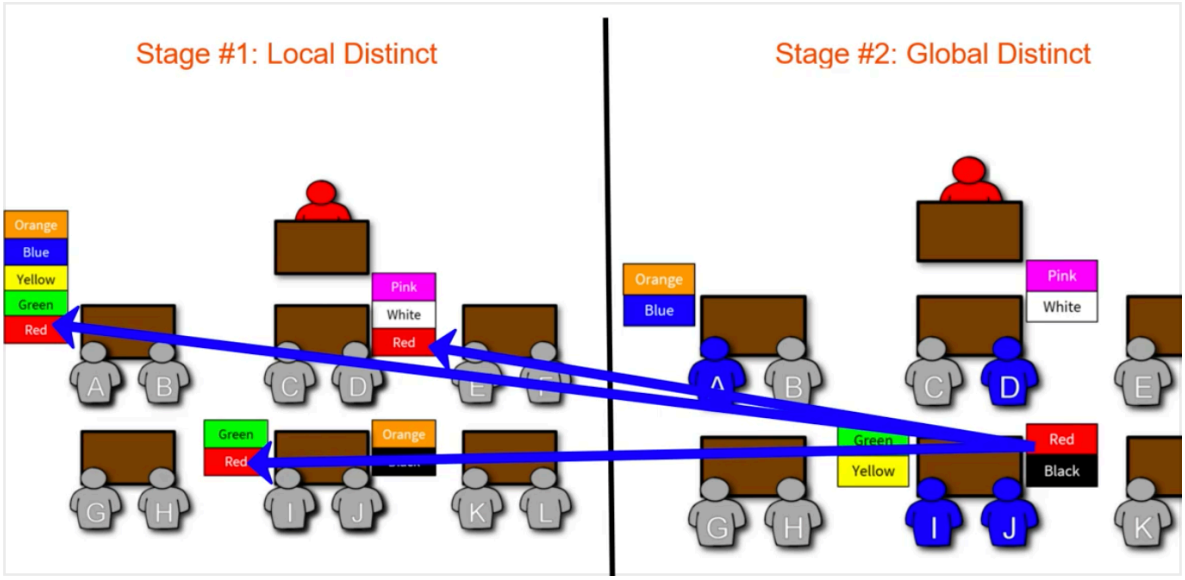


and I goes to A , pick yellow and green to itself.
D doesnt have to go anywhere as it has all the distinct colors

##

The stages for distinct : After stage 1, the J has to read data from A,D,I for red from other executors through network I/o and DISK I/O (write and read operantions happens b/w executors) .

There are optimisations which decreases the amount of work.
eg: bucketing
broadcast join



In scenario #3, what was the last action taken in the conclusion of stage #1?

- ✓ The new dataset was written to disk as shuffle files
- ✗ Each partition of data (bag of D&B candies) was reduced to a distinct set of colors
- ✗ Each partition of data was read into each executor by their corresponding tasks.
- ✗ Each task executed a push operation to move their respective partitions into the next stage
- ✗ None of the above

✗

Incorrect

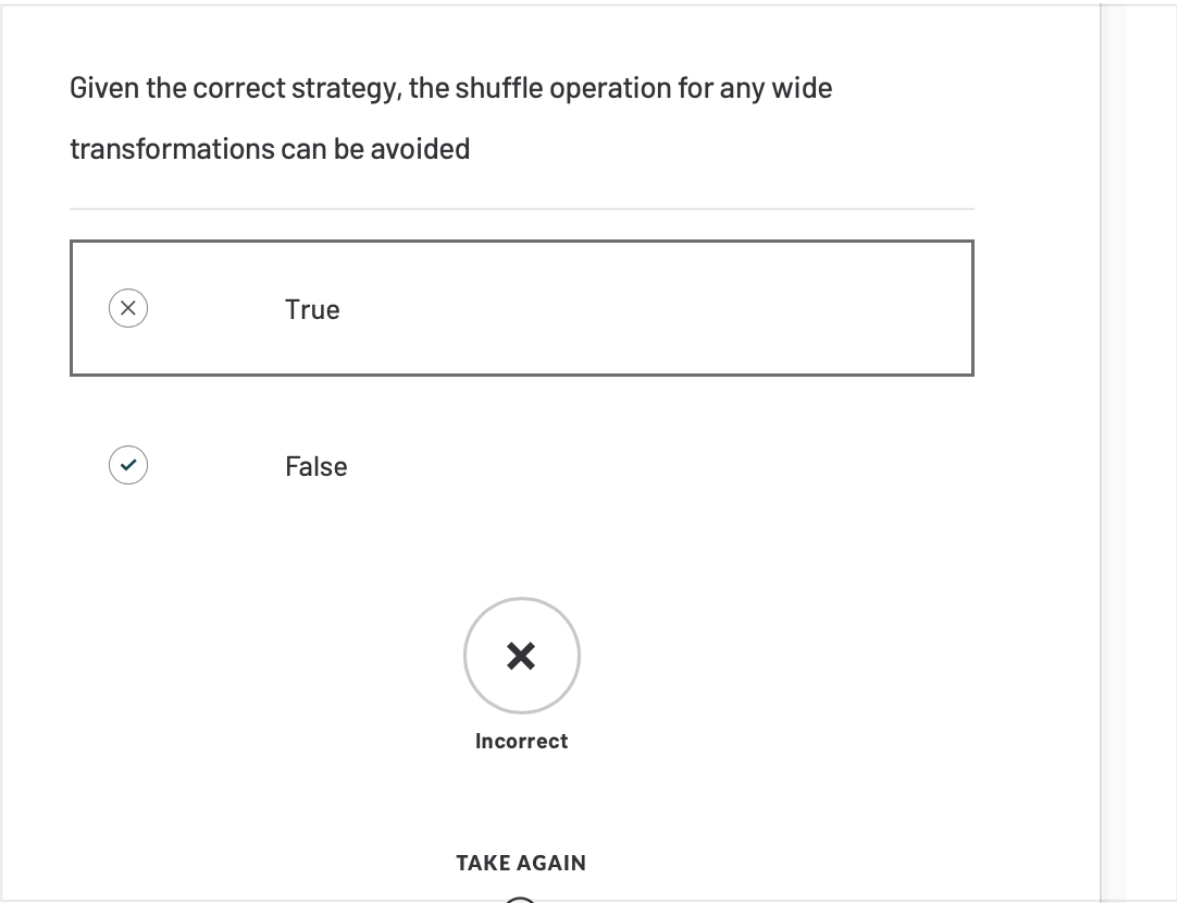
Which key factors relate to the performance hit associated with wide transformations?

- ✗ The pre-processing of each partition to conform to a data structure suitable to the corresponding wide transformation.
- ✗ The writing of shuffle data to disk
- ✗ The reading of shuffle data from disk
- ✗ The reading of shuffle data from one executor into another
- ✗ The post-processing of each partition to conclude the shuffle operation
- ✓ All of the above

Narrow - WIDE

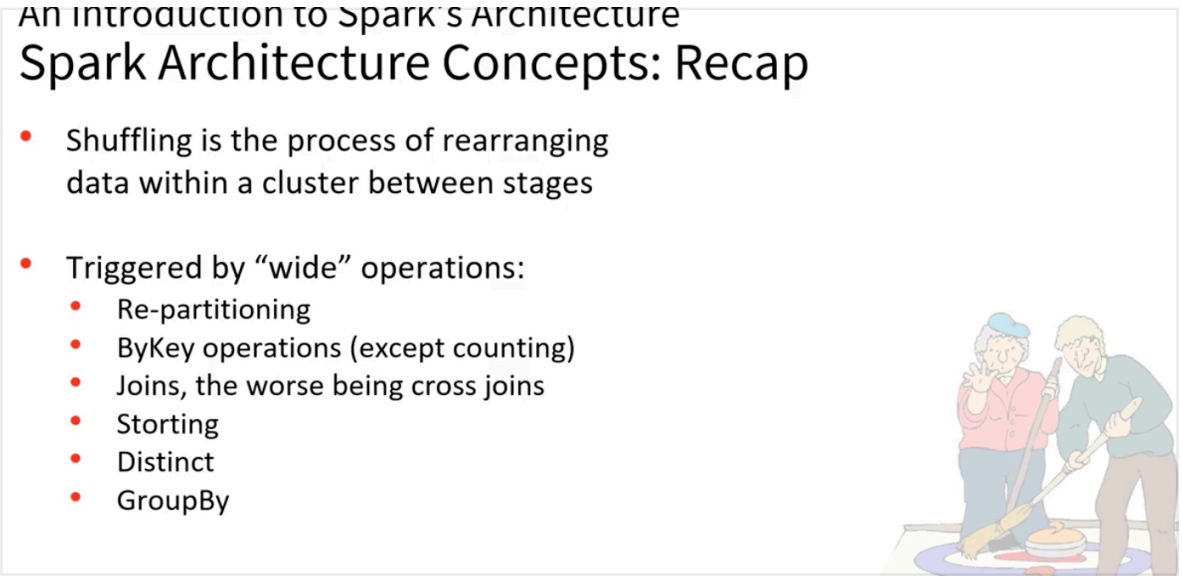
coalesce, sample, filter(), map(), flatMap(), filter(), union() are some examples of narrow transformation - Narrow

dISTINCT, sort() , , groupByKey(), aggregate(), aggregateByKey(), join(), repartition() are some examples of a wide transformations.



Catalyst optimiser to optimise the operations - https://www.databricks.com/glossary/catalyst-optimizer

In below, the wide opn happens by below opns:



As below, for best optimisation, we can group wide transformations togetther. eg: in below image

- Try to group wide transformations together for the best automatic optimization
- Aim for: narrow, narrow, wide, wide, wide, narrow
- Avoid: narrow, wide, narrow, wide, narrow, wide
- One of the most significant, yet often unavoidable, cause of performance degradation
- Just because it's slow, it doesn't mean that it's bad (don't polish the cannon ball)