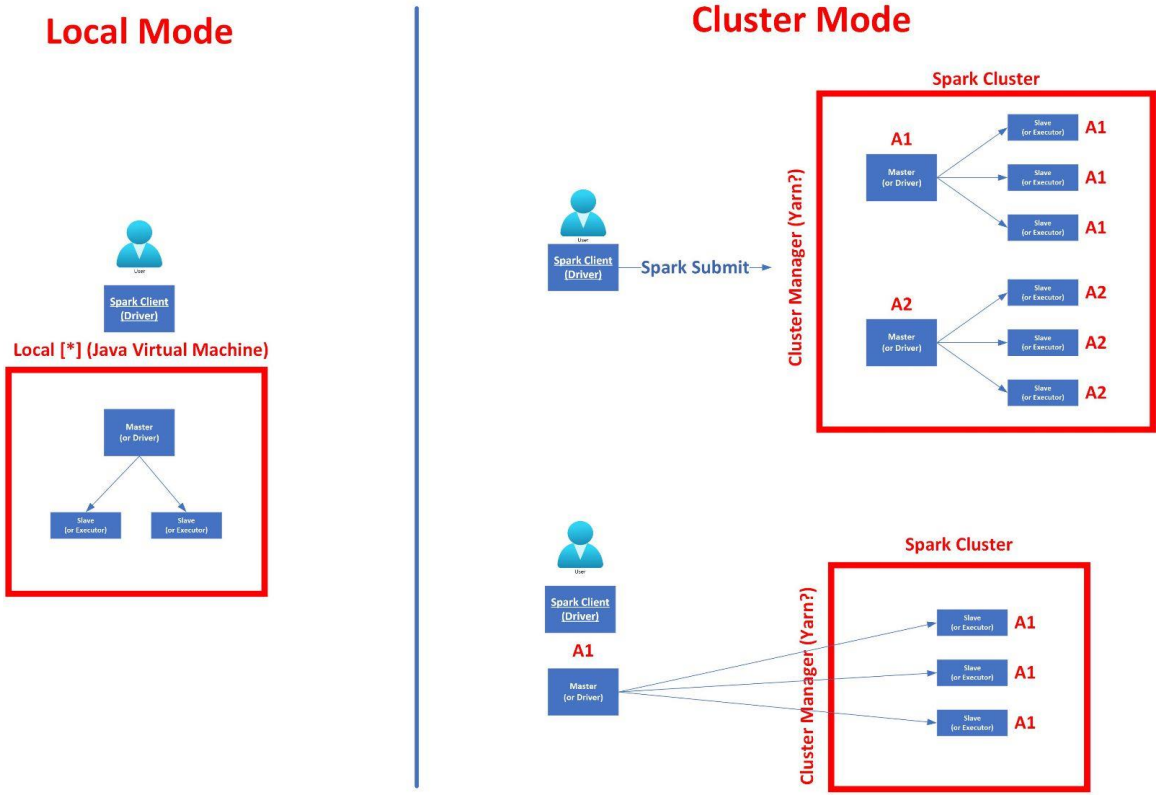
APACHE SPARK ARCHITECTURE

DR. ALVIN ANG

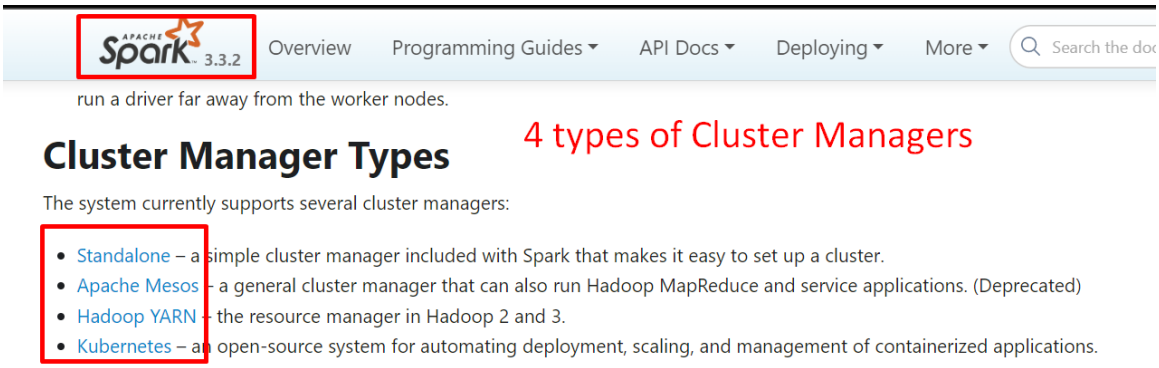
CONTENTS

I. SIMPLE ARCHITECTURE

- Apache Spark is meant for distributing big data to do parallel processing workloads.
- Basically, we have 2 modes:
    - Local vs Cluster



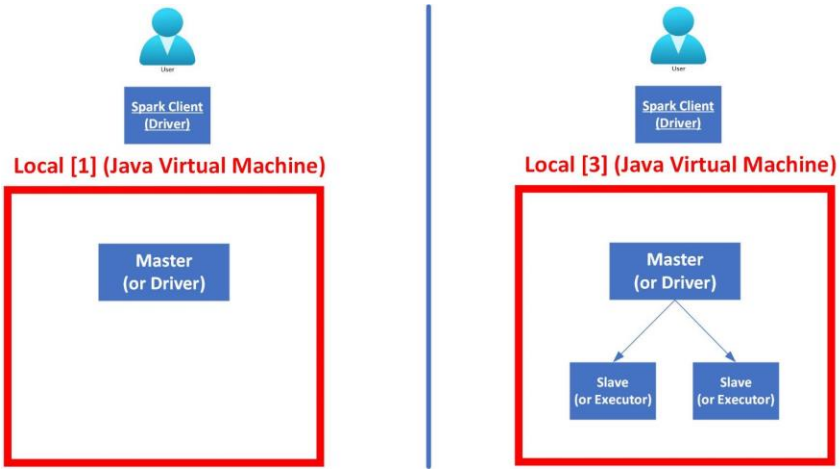
- Spark Client refers to your IDE (like Jupyter Notebook or Colab)… the place where you code
- Your Code is also known as an "Application". It can either sit on your laptop running (like Jupyter NB), or you can submit it to a Cluster Manager, once you are done with your code.
- The red boxes represent the Cluster Managers.



- There are 4 official types of Cluster Managers.
- Standalone refers to Local[*]
- Yarn is the most popular Cluster manager.
- Its supposed to take care of the automatic scheduling of your resources backend.

A. LOCAL

- Local mode represents running everything on 1 laptop.
- It doesn't make sense since Apache Spark was meant to be performed for big data running on huge clusters.
- Thus, Local mode is just for testing and learning purposes.
- You can either have just the Master running… or Master and Slaves…

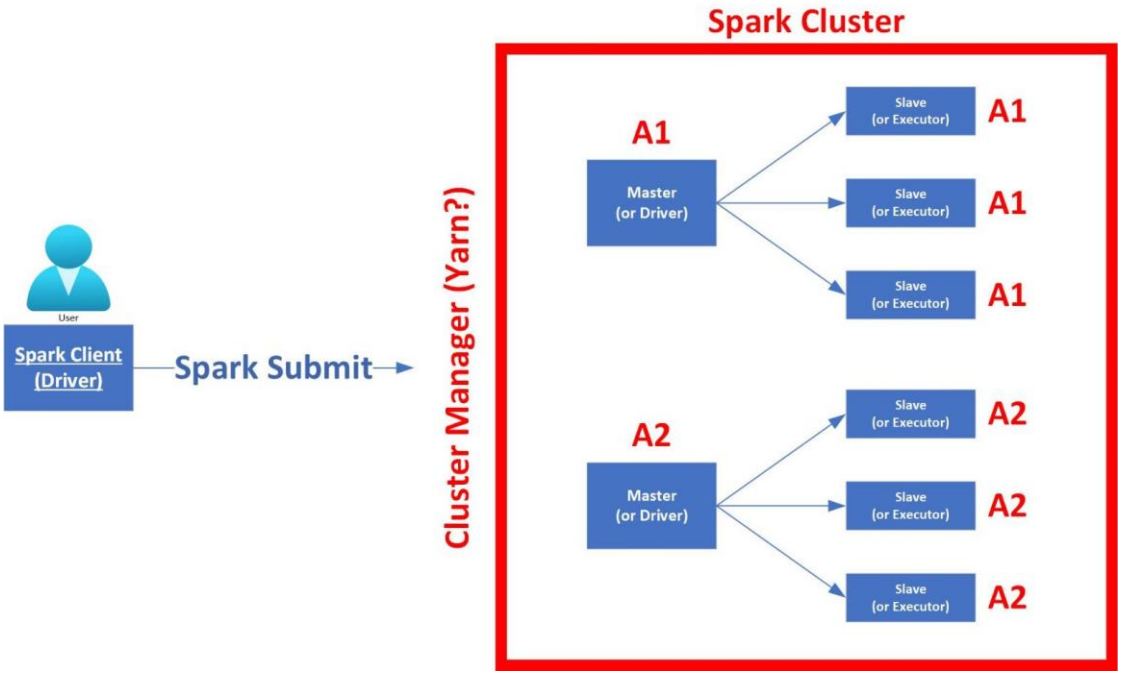


- Local [*] … the * represents the number of threads (running inside the Java Virtual Machine JVM).
- Local [1] means only 1 thread… which refers to only the Master… and with no other threads, the Master has to do all the work himself….
- Local [3] means 3 threads… which refers to the Master and 2 others Slaves… which means both Master and 2 Slaves work together….
- However, note that all these still happen only on 1 laptop, meaning its just using all the resources on 1 laptop.
- An example would be: https://www.alvinang.sg/s/Installing-Spark-on-Colab-by-Dr-Alvin-Ang.pdf
- This article represents running Spark on 'Local'…. Because Colab can't do multiple Clusters (its not connected anywhere else…)
- Thus the entire processing is using the Resources on 1 local Colab runtime only…

B. CLUSTER MODE

- You either have "Submit" mode or "Real Time" mode

1. SUBMIT MODE

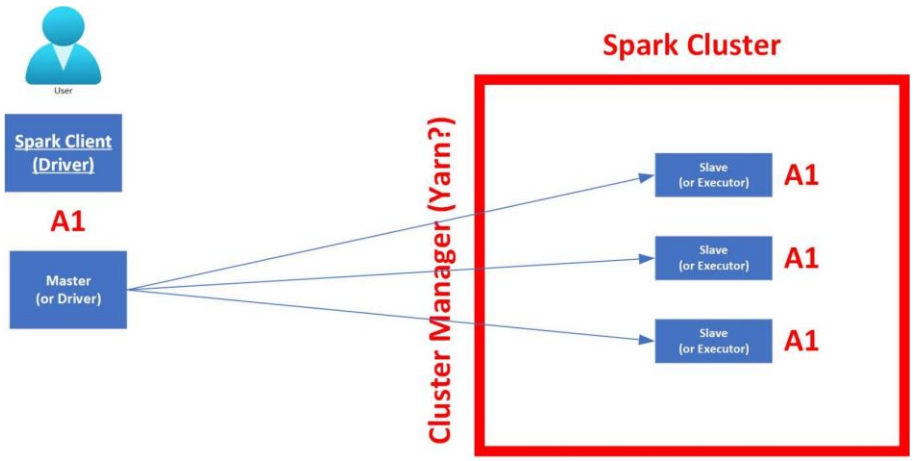


- You basically Code on your IDE or your laptop (also known as your Application) …
- And once done, you ***submit*** your Application (A1 and/or A2 if you have 2 applications) to the Cluster Manager (like Yarn) who will take care of managing the Masters and Slaves.
- When you switch off your laptop (or IDE), the Cluster Manager continues to work.
- An example is: https://www.alvinang.sg/s/Setting-Up-Apache-Spark-Cluster-in-Google-Cloud-by-Dr-Alvin-Ang.pdf

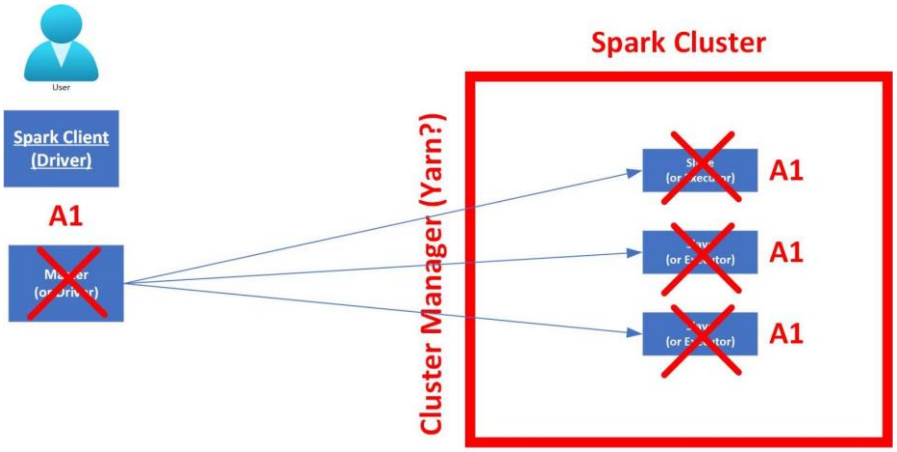
2. "REAL TIME" MODE

- The difference between "Real Time" vs "Submit" mode is that here, the Master sits on your laptop.

- The Master is continuously supervising the Slaves on the Cluster in "Real Time".



- But once you turn off your laptop, the Master dies and all job dies because all Slaves are turned off as well.



- If you are using your own laptops as Slaves, like this article:

    - https://www.alvinang.sg/s/Building-a-Apache-Spark-Local-Cluster-on-Windows-by-Dr-Alvin-Ang.pdf

    - Then this is a "Real Time" mode.

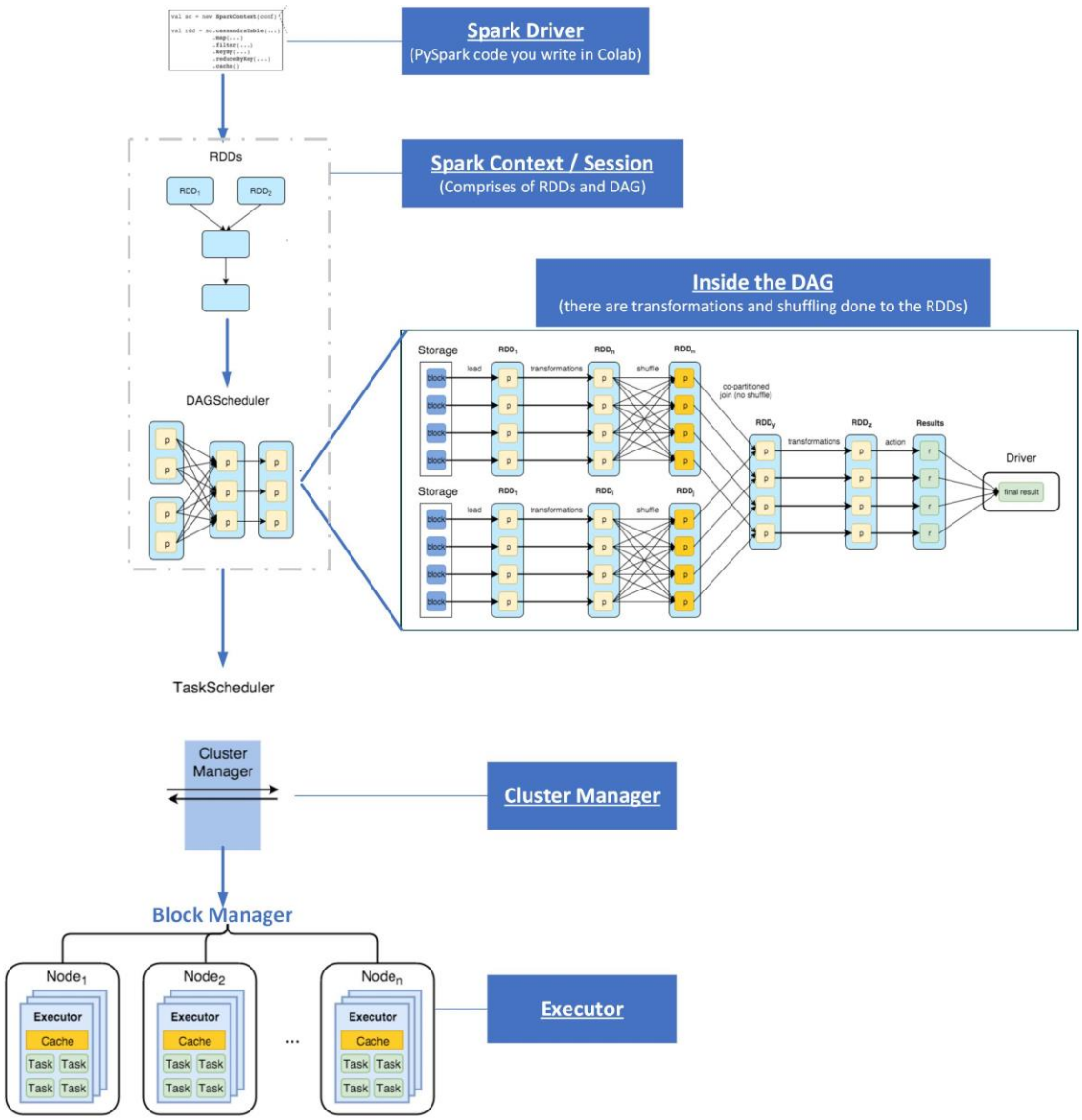
    - But run locally.

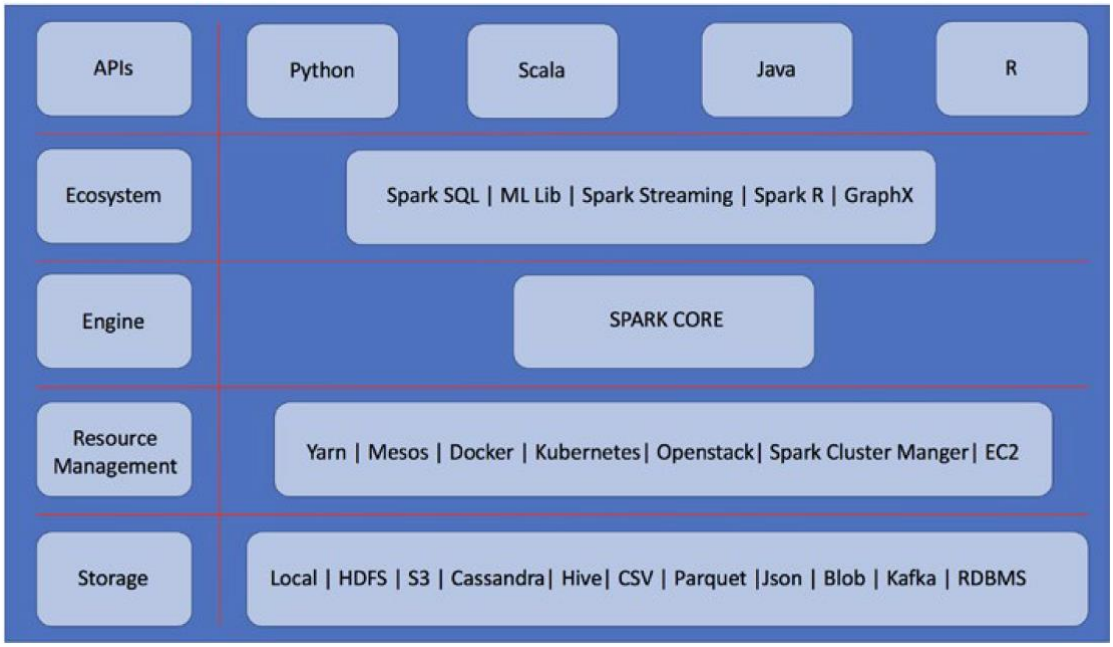
- But if you are using the Cloud like AWS / Databricks…
    - https://www.alvinang.sg/s/Using-Apache-Spark-in-AWS-and-Databricks-by-Dr-Alvin-Ang.pdf

II. DETAILED ARCHITECTURE

For details, you may refer here:

- https://datastrophic.io/core-concepts-architecture-and-internals-of-apache-spark/
- https://towardsdatascience.com/a-neanderthals-guide-to-apache-spark-in-python-9ef1f156d427





A. HOW THIS WORKS (IN A NUTSHELL…)

- At the start, there is a Spark Driver.
    - You can call this "coding"… where you code the Spark interface either using Python / Scala / Java or R.
    - You may also call this "driver" the Application Programming Interface (API).
    - You code the "driver" to give it multiple "tasks".
- But which library / environment / ecosystem are you coding in?
    - You then pick the library you want to use to start to code in.
    - Spark SQL / Spark ML Lib / Spark Streaming / Graph X.
    - Example: You pick the ML Lib and start coding in it to do Machine Learning Tasks.

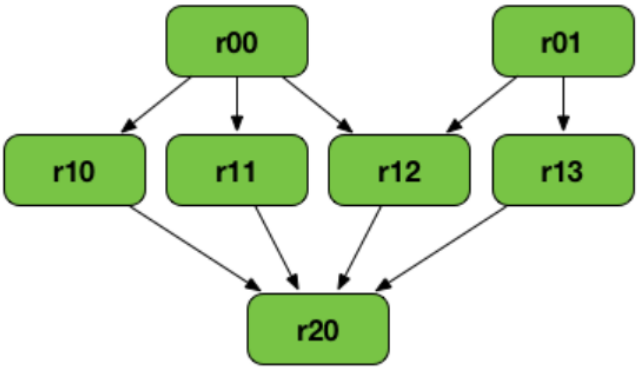
- Subsequently, you need to store your dataset some where right?
    - S3 / CSV / Kafka / RDMS (Relational Database Management Systems) are places you can store your big data.
    - Spark calls / labels storage of your datasets as RDDs (Resilient Distributed Database).
- Once you have setup your RDD, you have entered the Spark "Context"
    - Spark Context / Session distributes the RDDs (Datasets) into multiple chunks.
    - This distributing process is called DAG (Direct Acyclic Graph).
    - The purpose of breaking down the RDD is akin to breaking down a huge dataset (which is a lot of work) to be redistributed later to other "workers" to handle later on.
- Now, it's the Cluster Manager's turn to take over.
    - He will group multiple nodes into "blocks".
    - That's why he is also called the "block" manager.
    - Each node is known as an executor / worker / just simply a machine (like one laptop).
    - The Cluster Manager will take the chunks of RDDs (smaller chunks of datasets) and hand them over to his "workers" / nodes to do processing – each labelled as a "task".
    - The Cluster Manager will use Resource Management tools such as Yarn / Mesos / Docker / Kubernetes / OpenStack / Spark Cluster Manager / EC2 to help with distribution.

B. RDD

- RDDs = Resilient Distributed Datasets

1. R FOR RESILIENT

- RDDs are fault tolerant.
- Meaning they work properly even when a failure occurs.
- A failure could be a node bursting into flames for example, or just a communication breakdown between nodes.



- The graph above shows the Lineage Graph of a RDD.
- Its what the DAG does – transformation and actions (more about this later) that distributes the RDDS.
- This distribution of RDD makes it fault tolerant.
- In other words, its like replication of the same dataset and placed around other places.
- It makes the RDD independent of other RDDs.
- Thus, if a node / "worker" fails for some reason, all the information about what that node was supposed to be doing is stored in the lineage graph, which can be replicated elsewhere.

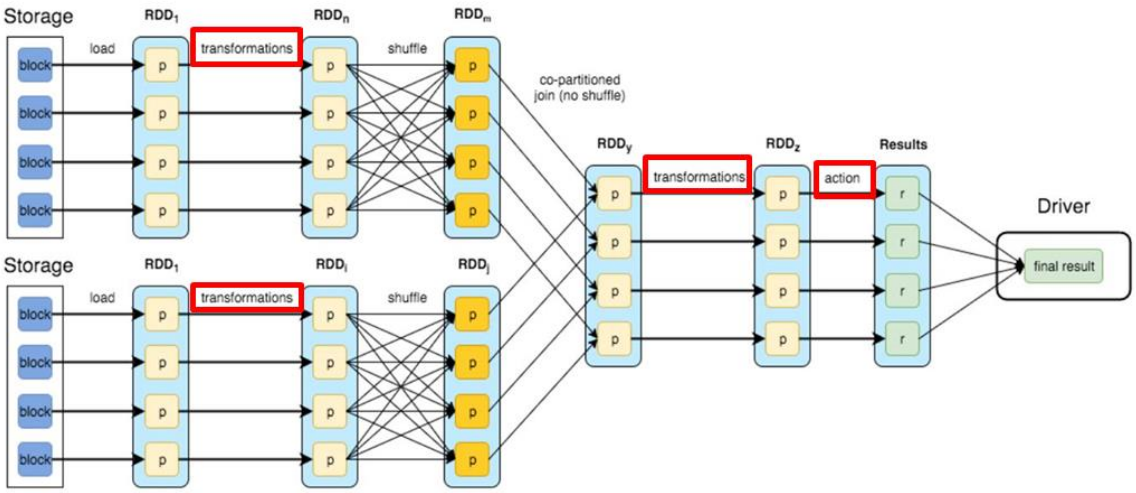
2. D FOR DISTRIBUTED

- RDDS are distributed.
- Which means they can be repartitioned / shuffled (for sharing of computing resources if the dataset is very huge).
- Since the processing of data will be divided across multiple nodes, the data is also divided across multiple nodes.
- Partitioned data refers to data that has been optimized to be able to be processed on multiple nodes.

3. D FOR DATASETS

- RDDs are a collection of datasets.
- RDDs are immutable: Which means once an RDD has been made, it is impossible to alter it.
- RDD is similar to Pandas DataFrame.
- But RDD do not have a schema, which means that they do not have a columnar structure.
- Records are just recorded row-by-row, and are displayed similar to a list.
- While Pandas DataFrames are organized into columnar structure.
- However, we have Spark DataFrames.
- Spark DataFrame have all of the features of RDDs but also have a schema.
- This will make them our data structure of choice for getting started with PySpark.

C. DAG



- DAG stands for Direct Acyclic Graph.
- Two things happen in the DAG:

1. TRANSFORMATIONS

- Transformations create new RDDs (something like creating multiple mirror images of the dataset, without cloning them actually).
- Transformations cannot alter RDDs because they are immutable once created.
- So if you look at the Lineage Graph earlier (in green), the RDD at the top is the "parent" while those at the bottom are "child".
- The "child" is just a mirror image of the "parent". They are just hypothetical RDDs.
- They aren't really existent until an "Action" is called upon it.
- Thus, even if one RDD is down, they can "mirror" back to other RDDS to be recreated.

2. ACTIONS

- An Action does NOT produce an RDD as an output.
- An Action is the cue to the compiler to evaluate the Lineage Graph and return the value specified by the Action.
- Some examples of common Actions are:
    - doing a count of the data,
    - finding the max or min,
    - returning the first element of an RDD, etc.

D. TASK SCHEDULER / CLUSTER MANAGER / EXECUTOR

a) Task Scheduler = Cluster Manager = BOSS

- Job Scope to Check:
    - Status of worker node (busy/available)
    - Location of worker node
    - Memory of worker node
    - Total CPU cores of worker node
    - Cluster manager keep check on the availability of nodes for task allocation.
- The two most widely used Resource Managers by Cluster Managers are YARN and Mesos.

b) Executor = Node = SLAVE

- Job Scope:
    - To work on any task given by the Cluster Manager.

Dr. Alvin Ang earned his Ph.D., Masters and Bachelor degrees from NTU, Singapore. He is a scientist, entrepreneur, as well as a personal/business advisor. More about him at www.AlvinAng.sg.