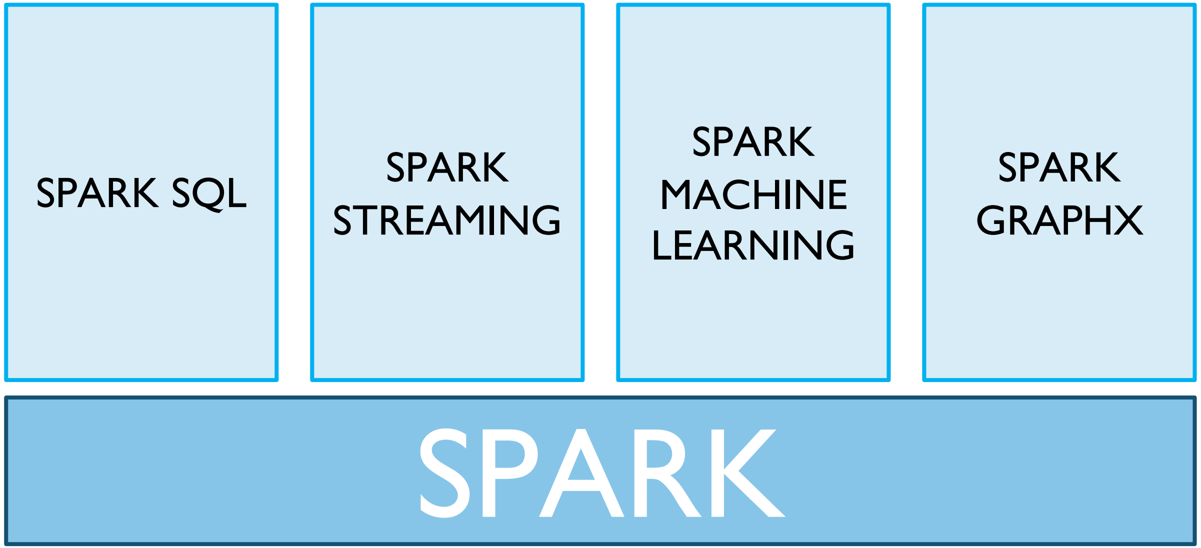
METIS

INTRODUCTION TO SPARK

What is Spark?

- Spark looks at Hadoop and says, “hey, that’s neat... but what if we made everything way faster?”
- It combines some of the principles we discussed in DASK and the storage mechanisms of HDFS

What is Spark?



Why is it fast?

- Roughly speaking, Spark does two fast things:
  - It employs a REALLY good project manager that coordinates all tasks efficiently
  - It uses RAM wherever possible instead of hard drive I/O
- These two components make Spark between 10x (worst case) and 100x (best case) faster than Hadoop.

Directed Acyclic Graphs (DAGs)

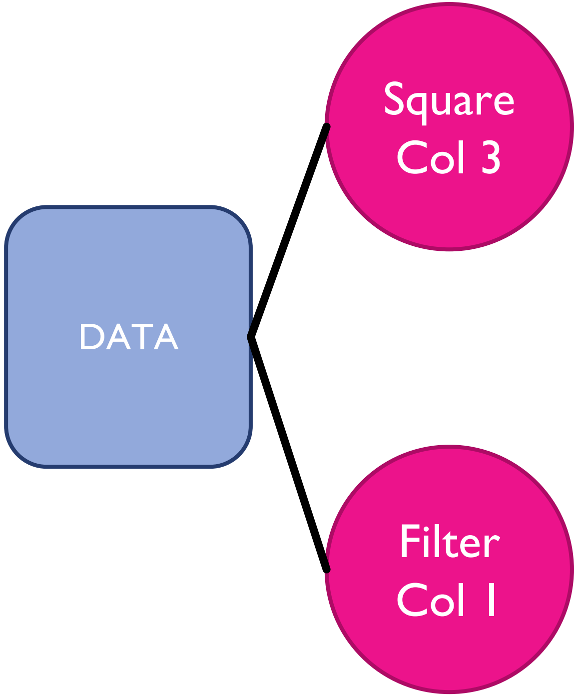
- DAGs are the project managers
  - If given a task to complete that has 60 steps, Spark doesn't start until a DAG is created that maps out the most efficient way to do all the things.

Directed Acyclic Graphs (DAGs)

Me: Hey Spark. I’d like you to filter on column 1 to only rows with values greater than 10, square column 3, then multiply columns 3&4 together, then get the mean of that product.

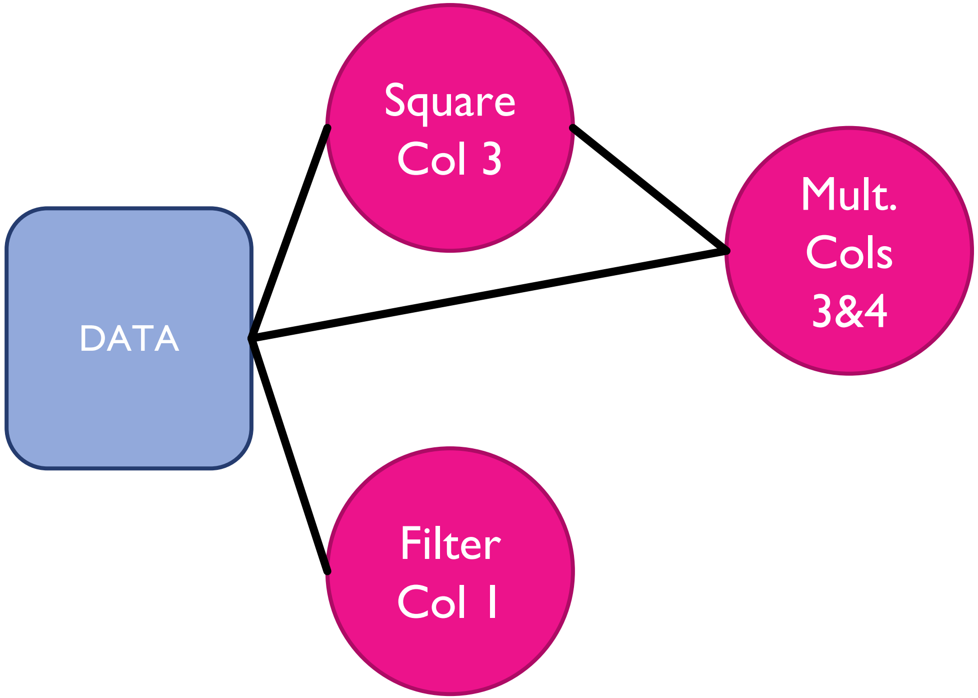
Spark: Sure. Let me look up how many nodes can I use, how much RAM on each node, and how many CPUs on each node. Got it. Just a sec.

Directed Acyclic Graphs (DAGs)



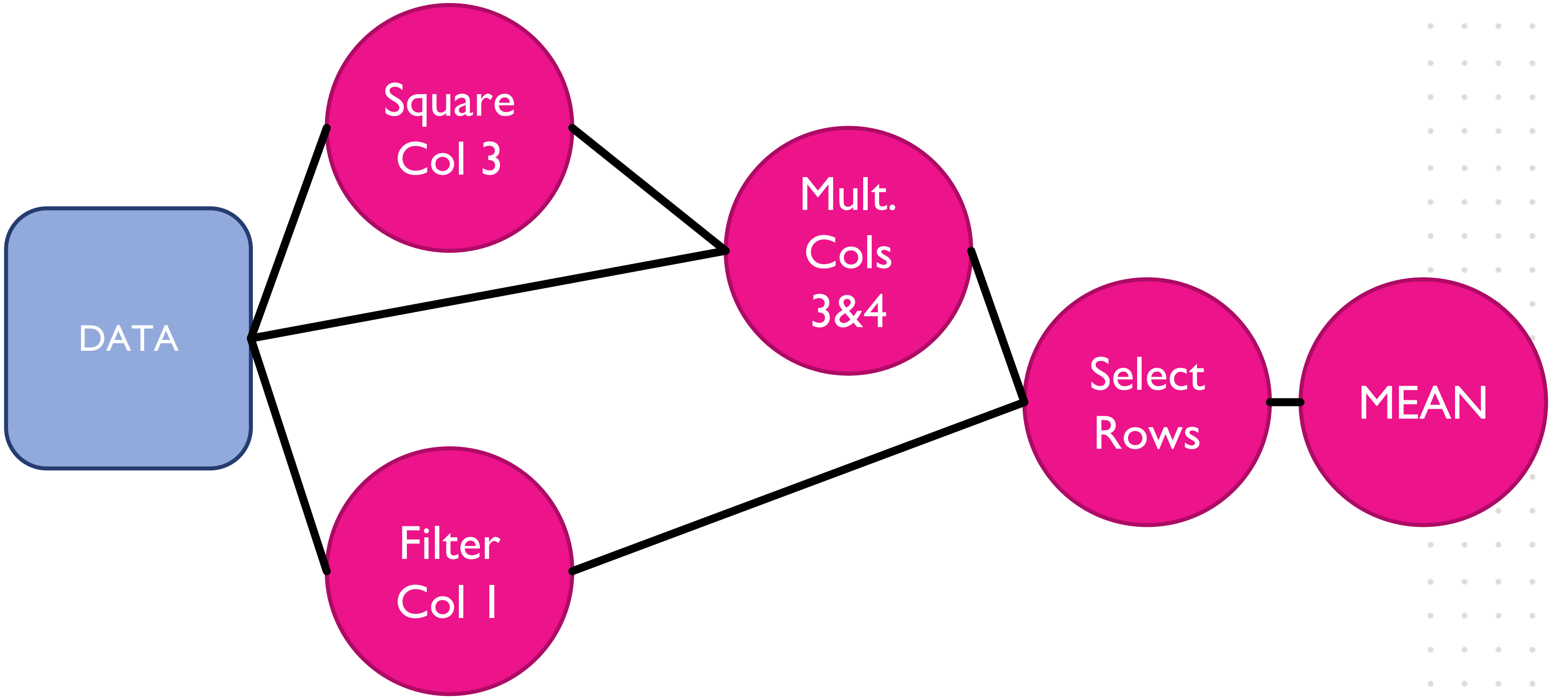
This stage can happen simultaneously, since I don't need column 1 to change column 3... so let's do those in parallel first.

Directed Acyclic Graphs (DAGs)



The next stage has a requirement from a previous stage, so we have to wait on stage 1 to do stage 2. There's nothing to be done with column 1 during stage 2. So we wait.

Directed Acyclic Graphs (DAGs)



Directed Acyclic Graphs (DAGs)

- The DAG can also consider the size of the data and adapt accordingly. For example, it may have been better to filter columns 3 and 4 before the multiplication if the parallelization gain wasn't big enough. It estimates that.

DAGs and RAM

- Not only do DAGs allow us to parallelize, they allow us to maximize our efficiency loading data.
- We know we have big data, but if we can do everything possible to the data while it's in RAM before loading the next chunk, we can be faster.

DAGs and RAM

- To demonstrate how much RAM can speed us up, let's imagine a scenario where every time the computer thinks (a single CPU cycle) it takes 1 whole second, instead of 0.4 ns.
  - This is just to give us a more "human" sense of scale to see why RAM matters.

DAGs and RAM

System Event	Actual Latency	Scaled Latency
One CPU cycle	0.4 ns	**1 s**
Level 1 cache access	0.9 ns	**2 s**
Level 2 cache access	2.8 ns	**7 s**
Level 3 cache access	28 ns	**1 min**
Main memory access (DDR DIMM)	~100 ns	**4 min**
Intel Optane memory access	<10 μs	**7 hrs**
NVMe SSD I/O	~25 μs	**17 hrs**
SSD I/O	50–150 μs	**1.5–4 days**
Rotational disk I/O	1–10 ms	**1–9 months**
Internet call: San Francisco to New York City	65 ms	**5 years**
Internet call: San Francisco to Hong Kong	141 ms	**11 years**

SCALE CHANGE

Credit: https://www.prowesscorp.com/computer-latency-at-a-human-scale/

DAGs and RAM

System Event	Actual Latency	Scaled Latency
One CPU cycle	0.4 ns	**1 s**
Level 1 cache access	0.9 ns	**2 s**
Level 2 cache access	2.8 ns	**7 s**
Level 3 cache access	28 ns	**1 min**
Main memory access (DDR DIMM)	~100 ns	**4 min**
Intel Optane memory access	<10 μs	**7 hrs**
NVMe SSD I/O	~25 μs	**17 hrs**
SSD I/O	50–150 μs	**1.5–4 days**
Rotational disk I/O	1–10 ms	**1–9 months**
Internet call: San Francisco to New York City	65 ms	**5 years**
Internet call: San Francisco to Hong Kong	141 ms	**11 years**

RAM takes 4 minutes to read.

Credit: https://www.prowesscorp.com/computer-latency-at-a-human-scale/

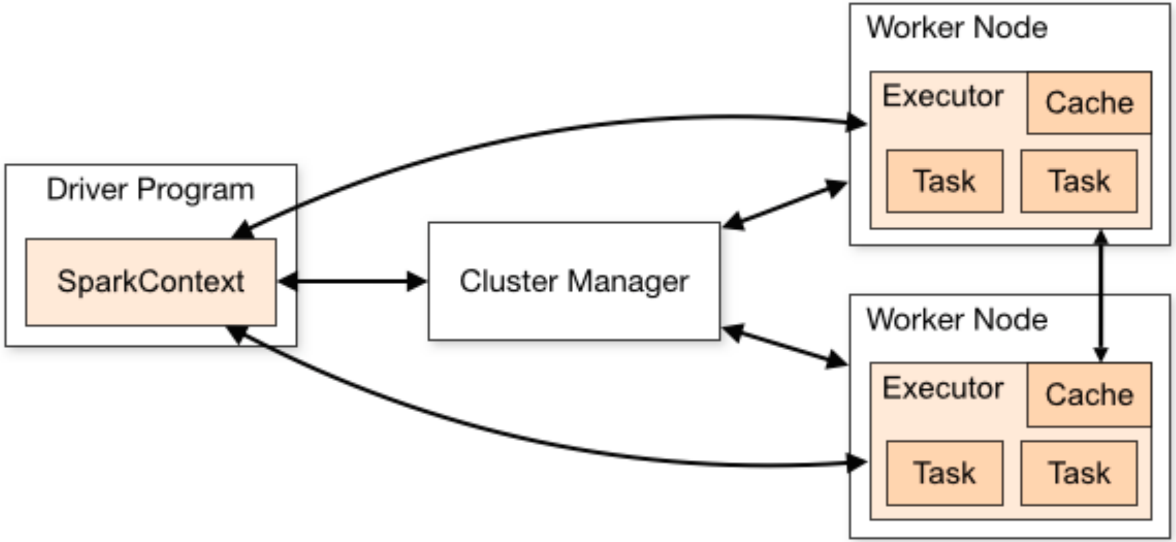
DAGs and RAM

System Event	Actual Latency	Scaled Latency
One CPU cycle	0.4 ns	**1 s**
Level 1 cache access	0.9 ns	**2 s**
Level 2 cache access	2.8 ns	**7 s**
Level 3 cache access	28 ns	**1 min**
Main memory access (DDR DIMM)	~100 ns	**4 min**
Intel Optane memory access	<10 μs	**7 hrs**
NVMe SSD I/O	~25 μs	**17 hrs**
SSD I/O	50–150 μs	**1.5–4 days**
Rotational disk I/O	1–10 ms	**1–9 months**
Internet call: San Francisco to New York City	65 ms	**5 years**
Internet call: San Francisco to Hong Kong	141 ms	**11 years**

Disk takes 1-9 months to read.

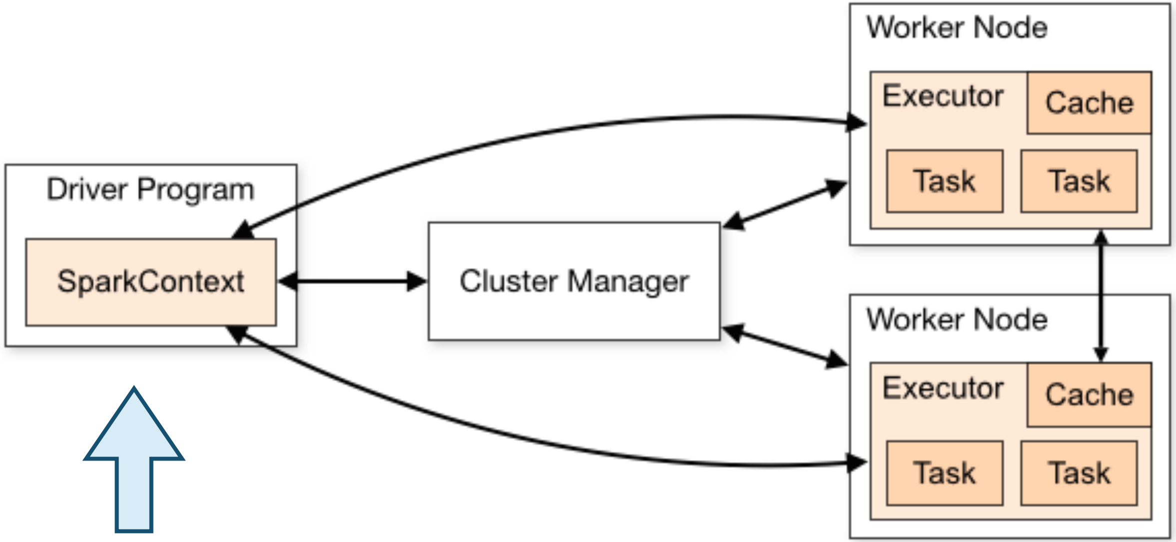
Credit: https://www.prowesscorp.com/computer-latency-at-a-human-scale/

Who manages all of this?



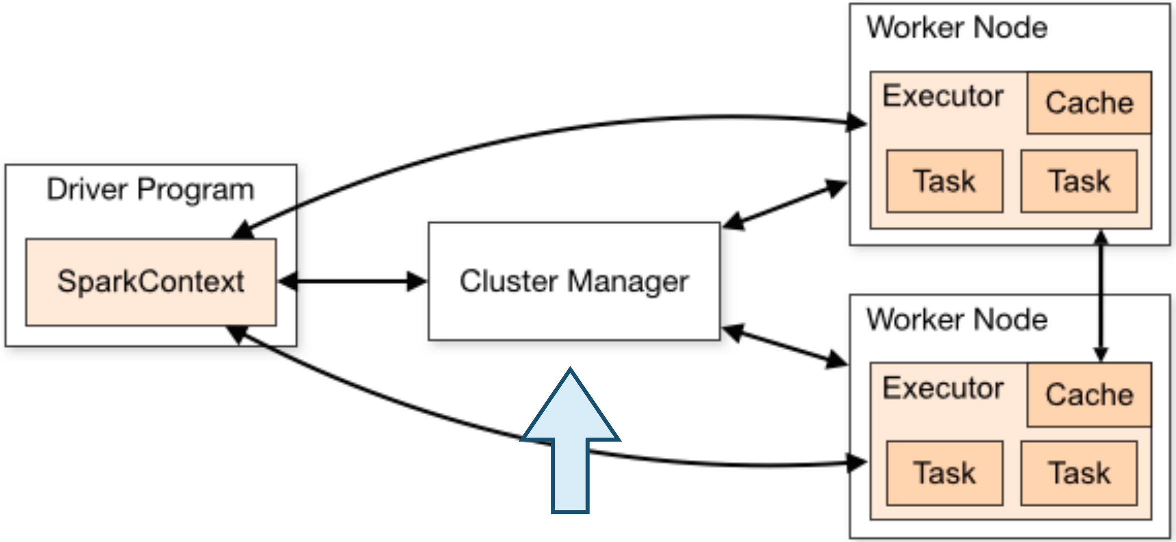
Credit: http://spark.apache.org/docs/latest/cluster-overview.html

Who manages all of this?



User submits jobs and tasks by interfacing with the Spark Context. Spark Context creates an instance of the Spark "run me" app. Creates DAGs.

Who manages all of this?



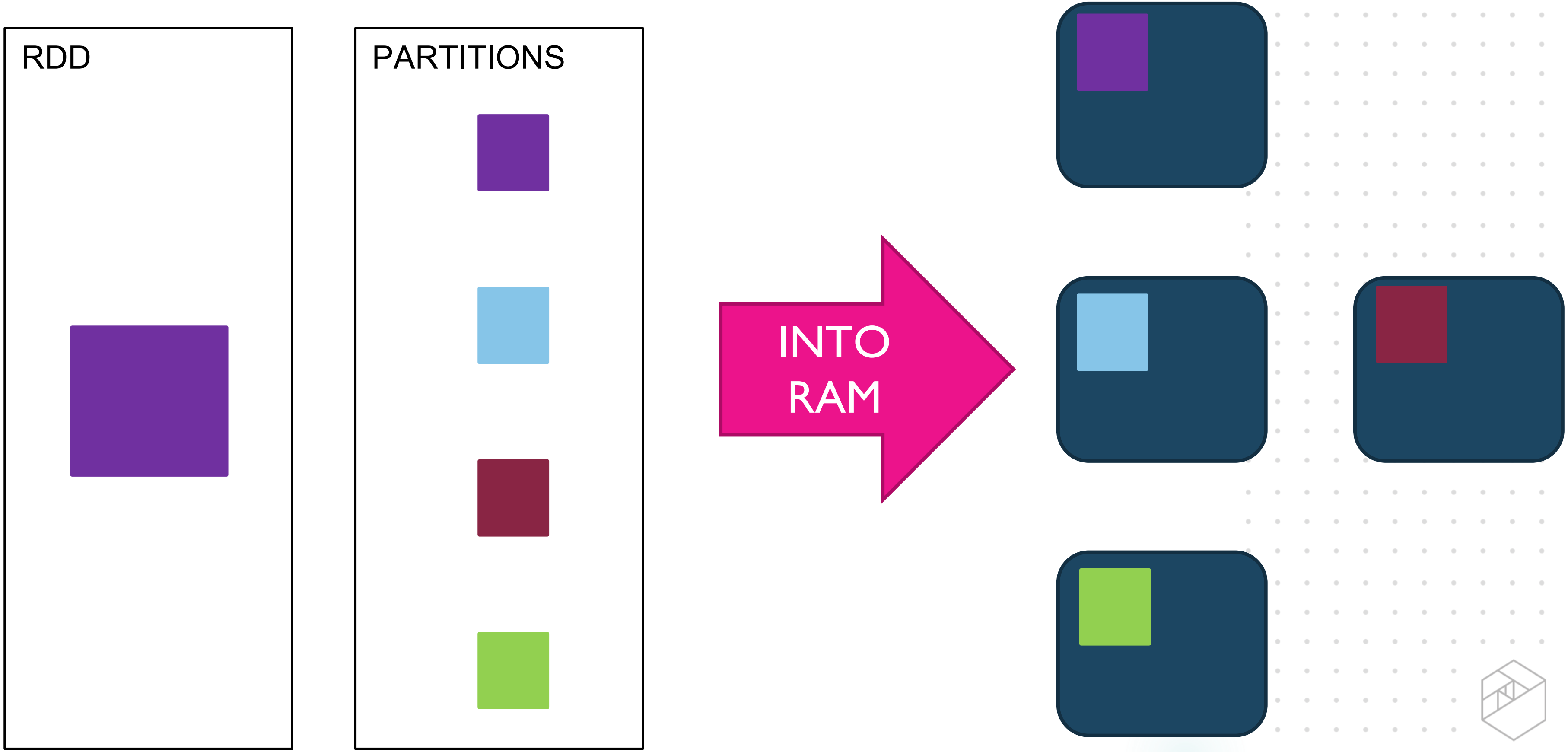
Figures out where/how the data is distributed, tracks tasks, executes DAG instructions, looks for node failures, etc. Hadoop YARN is a common cluster manager.

RDDs

Resilient Distributed Datasets (RDDs)

- The base unit in Spark. We'll rarely work with them directly, but we need to understand how they work.
- It's a method of taking a dataset and putting it onto several different computers RAM in a nice way.
- Hadoop distributed to disk, the RDD distributes to RAM

SPARK partitions data to fit it into the RAM of worker nodes



Resilient Distributed Datasets (RDDs)

- RDDs serialize the dataset, partition it, and then send it to multiple machines.
- We have to manage the number of partitions.
- If you have 1,000 computers, but only allow for 4 partitions, you aren’t using all of your hardware.
- RDDs are immutable to make sure we don’t break things while they’re split up.

Getting to know Spark

Spark 2.0

- We’ll be using Spark 2.0, which hides the RDDs behind DataFrames.
- So RDDs are the backbone, but we’ll usually interact with them through a DataFrame or SQL API.

Spark API

- Sparks API also uses lazy evaluation like Dask.
- There are two types of behaviors:
  - Transformations (don’t happen right away)
  - Actions (cause the whole DAG to execute)

Spark API – A few examples

Transformations: ***lazy***	**Actions:** ***executing***
map(func): pass each element of source through func	**reduce(func):** aggregate elements with func
filter(func): select elements of the source for which func returns true	**take(n):** copy top n elements to driver
distinct(): return duplicate-free	**collect():** copy all elements to driver
sample(withReplacement, fraction [seed]): sample with or without replacement	**foreach(func):** apply provided func to each element of RDD

Spark API – A few examples

Transformations: ***lazy***	**Actions:** ***executing***
map(func): pass each element of source through func	**reduce(func):** aggregate elements with func
filter(func): select elements of the source for which func returns true	**take(n):** copy top n elements to driver
distinct(): return duplicate-free	**collect():** copy all elements to driver
sample(withReplacement, fraction [seed]): sample with or without replacement	**foreach(func):** apply provided func to each element of RDD

Prepares Data

Does a Calculation

Spark API

- Spark uses this lazy evaluation to properly build DAGs
- We'll dive into the API more in a notebook with some hands on examples

Spark API

- Spark is actually written in Scala.
- To use Python we have to use a wrapper called "PySpark"
- Behind the scenes, all the code becomes DAGs anyway, so PySpark doesn't make the performance any worse*

* Except when you use User-Defined Functions that actually execute in Python

General Spark Flow for Users

- Create the Spark Context
- Choose interactive mode or job submissions
- Load data into a DataFrame (it creates RDDs)
- Do analysis with lazy evaluations/SparkSQL

General Spark Flow for Users

- (Optional) Create a Spark Cluster and Manager
- Create the Spark Context
- Choose interactive mode or job submissions
- Load data into a DataFrame (it creates RDDs)
- Do analysis with lazy evaluations/SparkSQL

QUESTIONS?