Apache Spark

Data and task distribution over a cluster

Summary from White, T. *Hadoop: The definitive guide*. "O'Reilly Media, Inc.", 2014.

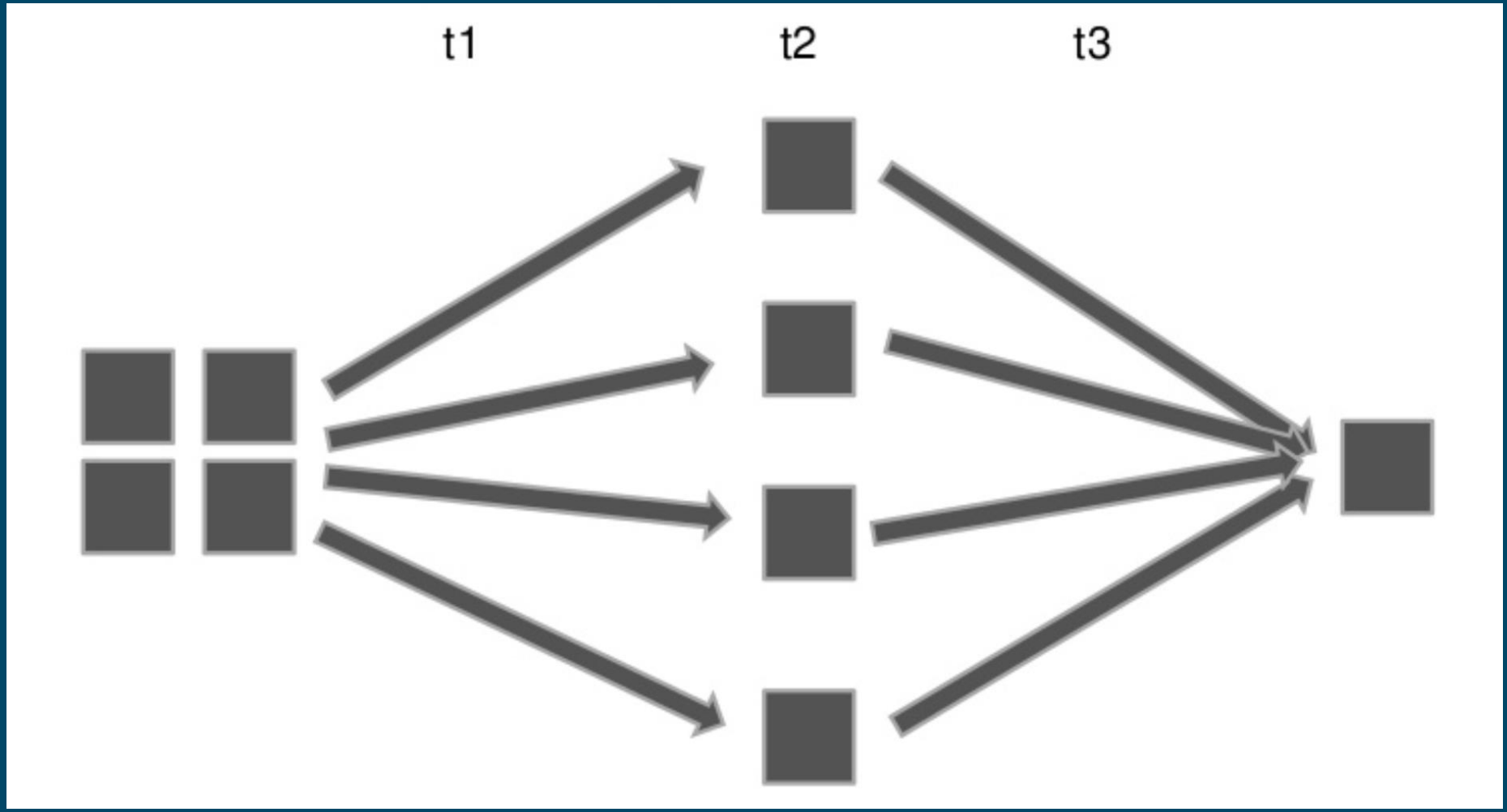
The Spark Cluster

A bit of review

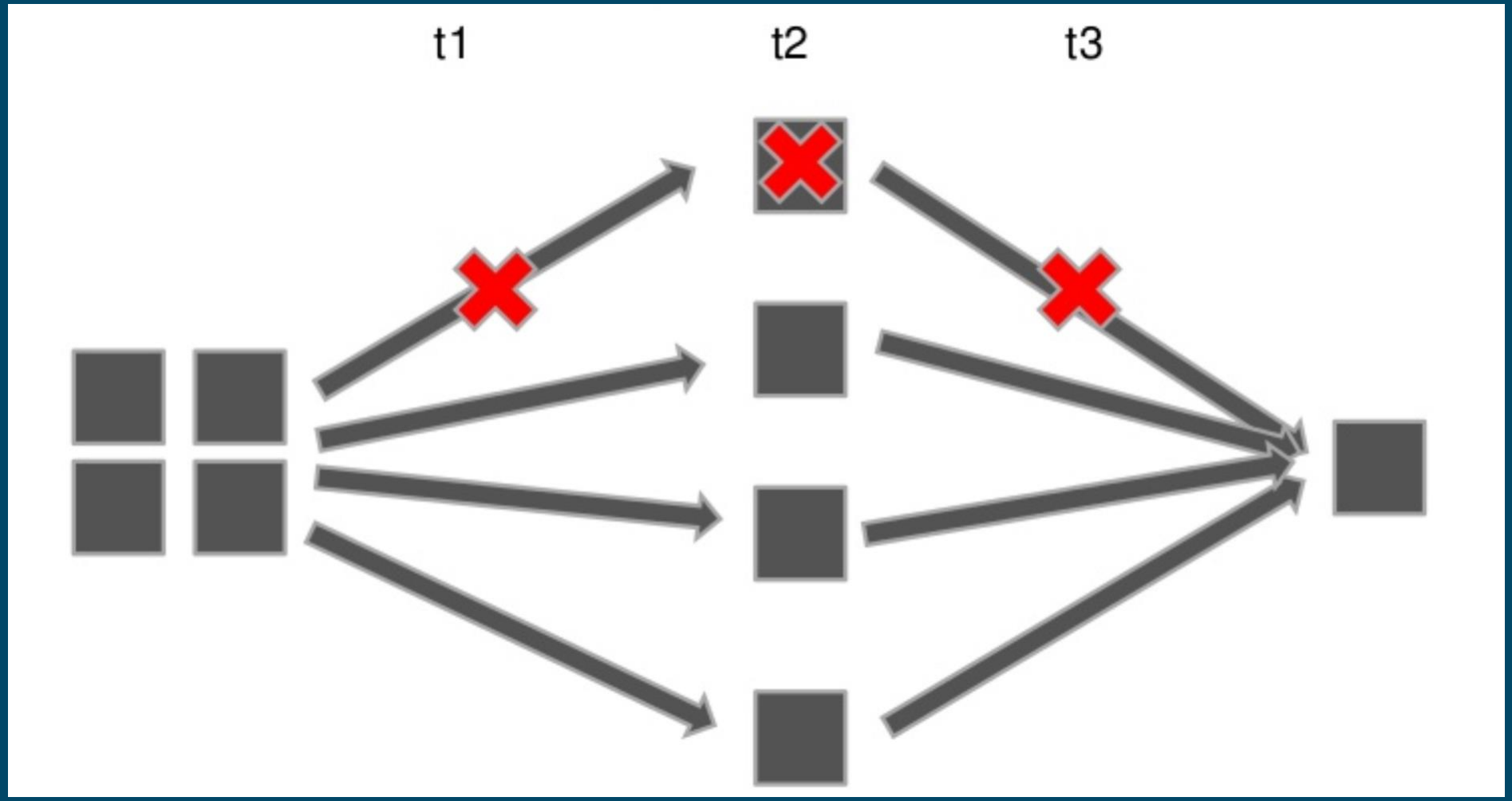
Data and code are distributed over cluster

Cluster

Data processed in parallel



Data processing can fail!



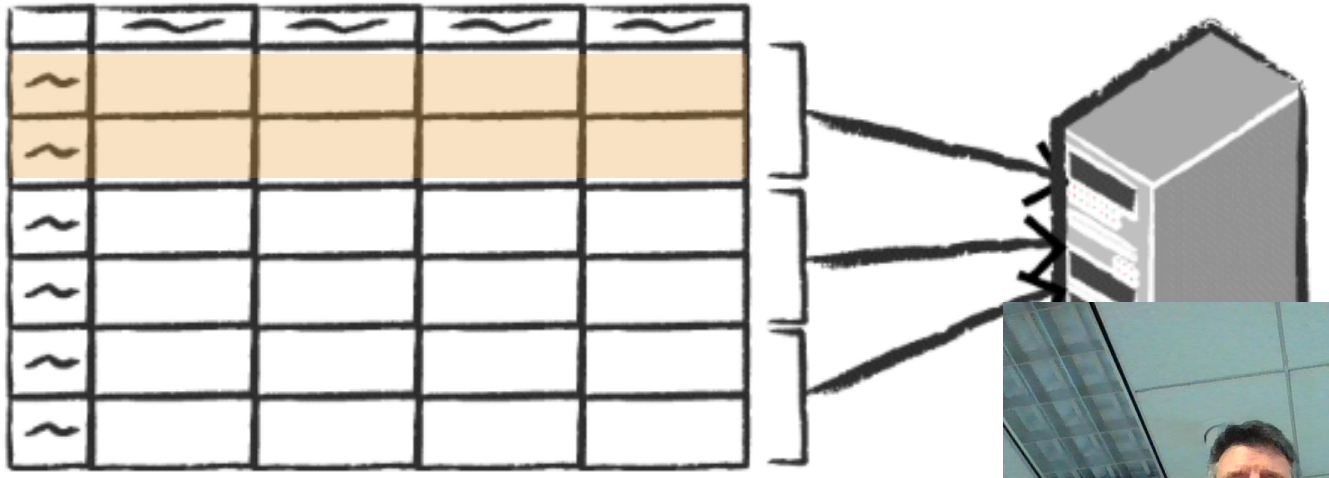
Data is distributed over the cluster

Tabular data is distributed over cluster

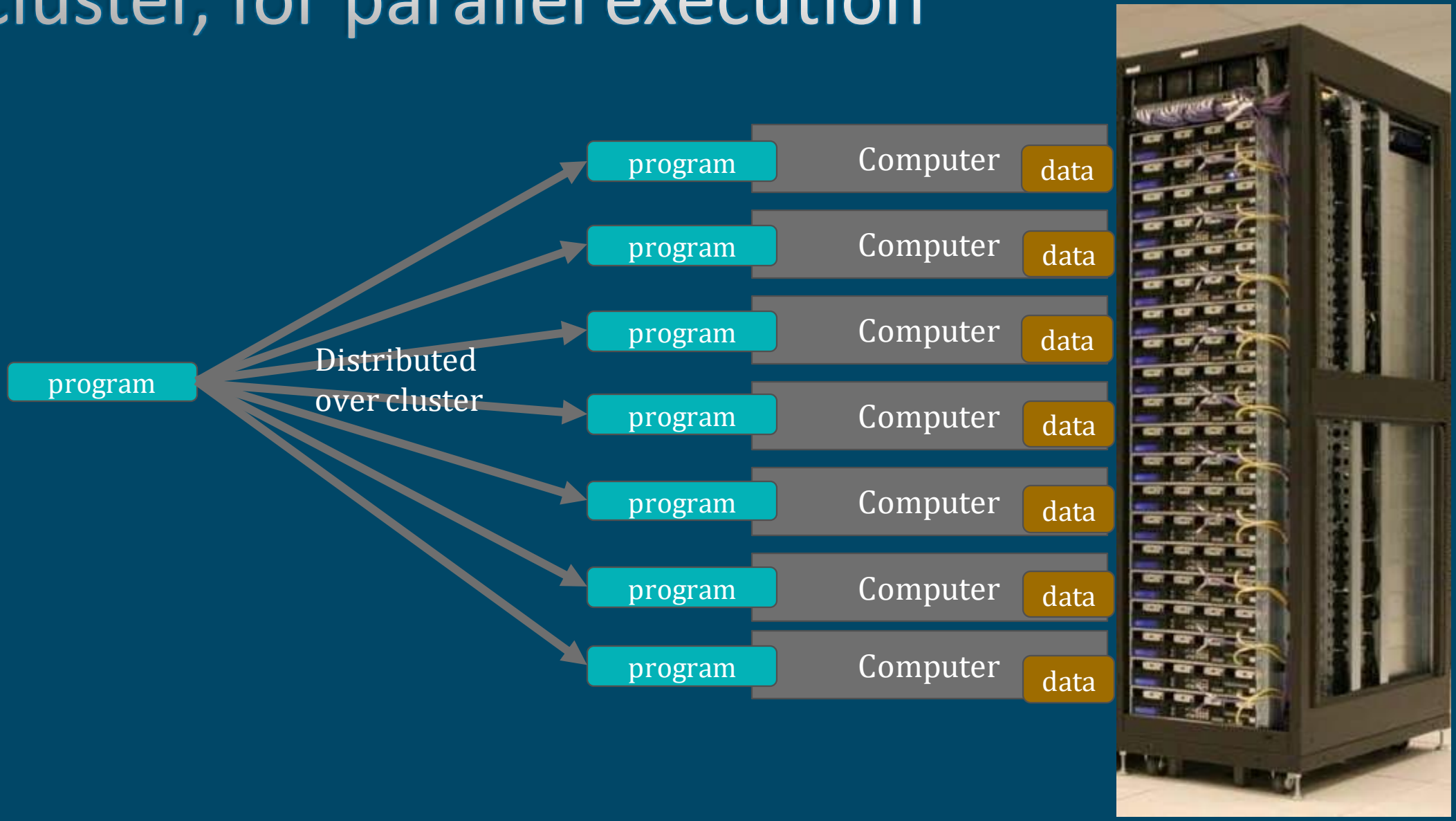
row subsets are partitions

Spreadsheet on a single machine

Table or Data Frame partitioned across servers in a data center



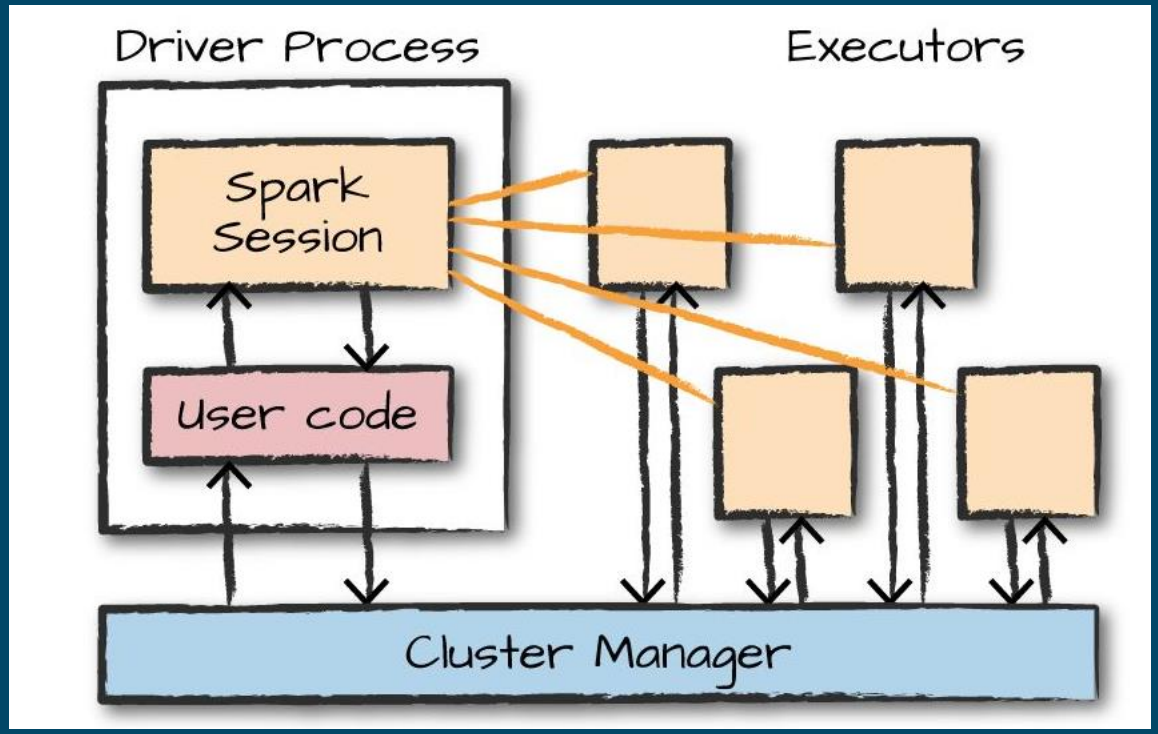
Spark copies a program to nodes in the cluster, for parallel execution



Task execution in Spark

Driver sends code, via manager, to cluster which contains the data

- Spark employs a cluster manager that keeps track of the resources available.
- The driver process is responsible for executing the driver program's commands across the executors to complete a given task.

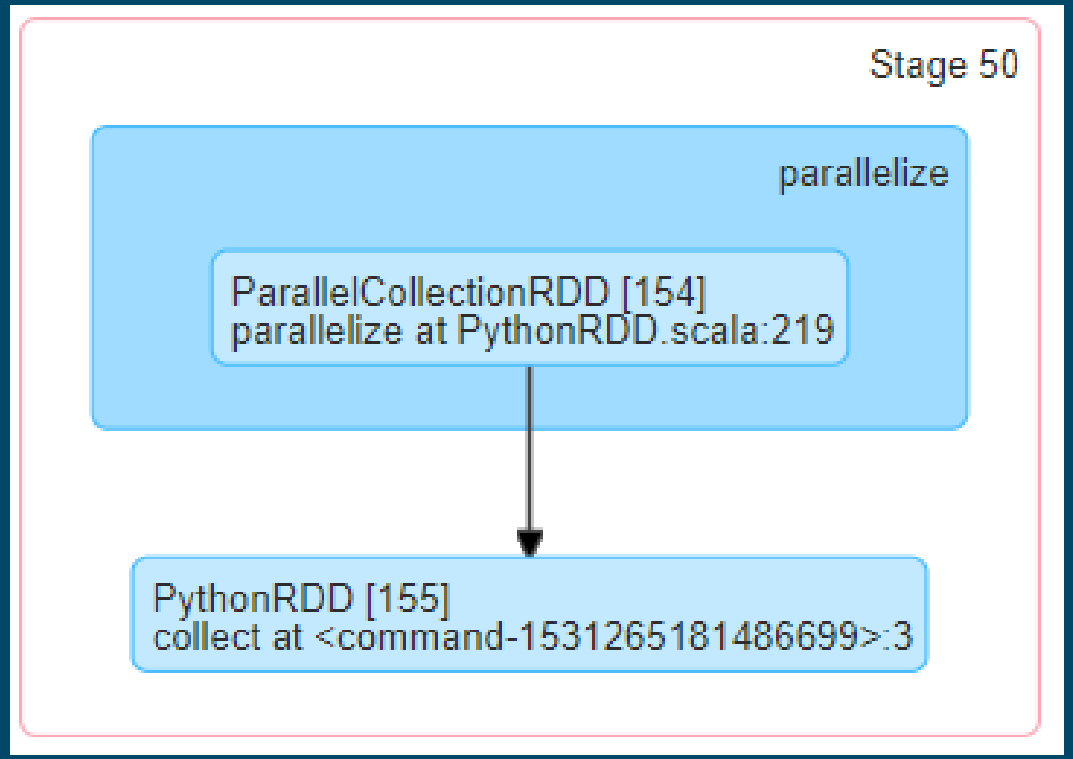


5 partitions -> 5 tasks

```
1  data = range(1,100)
2  rdd = sc.parallelize(data,5)
3  rdd.map(lambda x: x * x).collect()
4
```

▼ (1) Spark Jobs

▶ Job 37 View (Stages: 1/1)



Tasks (5)

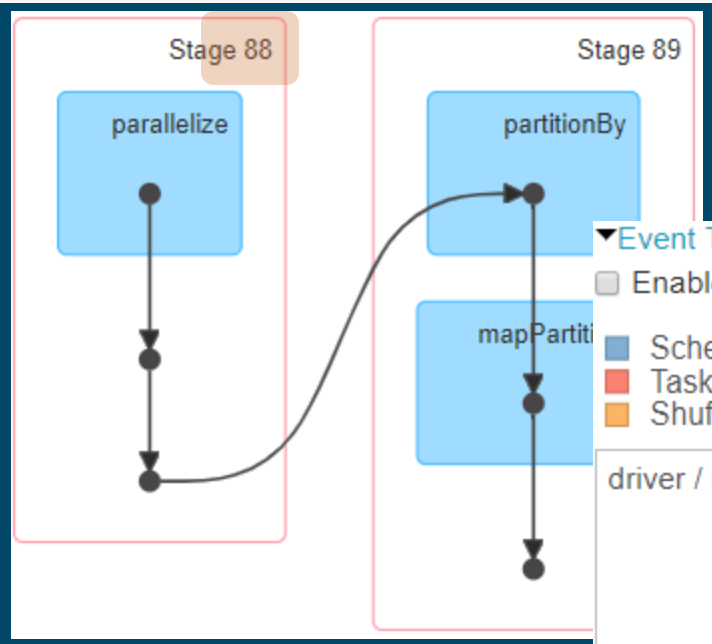
Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration
0	208	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2018/11/02 16:58:40	0.2 s
1	209	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2018/11/02 16:58:40	0.2 s
2	210	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2018/11/02 16:58:40	0.3 s
3	211	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2018/11/02 16:58:40	0.2 s
4	212	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2018/11/02 16:58:40	0.2 s

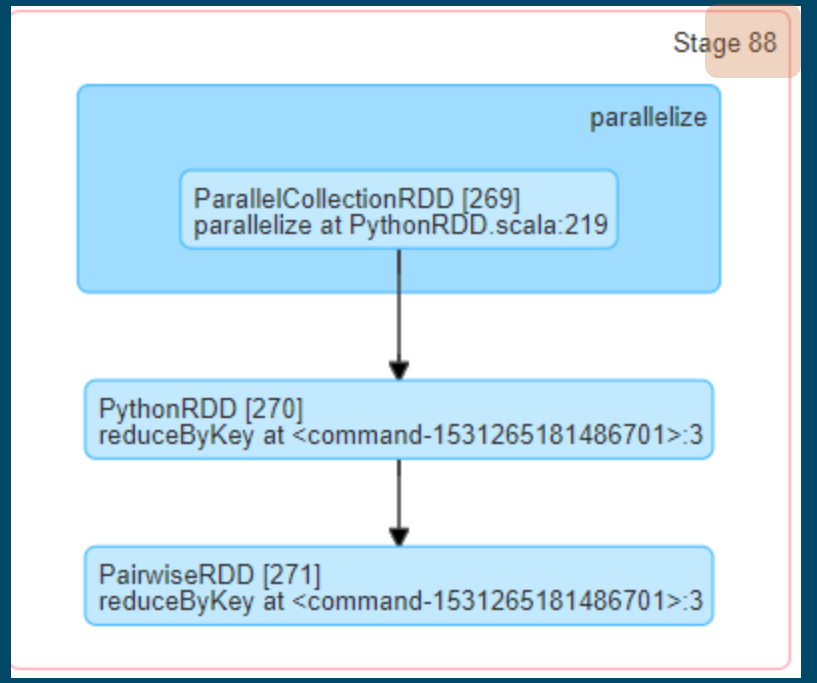
Map Reduce with 5 tasks

```
1 data = range(1,100)
2 rdd = sc.parallelize(data,5)
3 rdd.map(lambda x: (x%5, x)).reduceByKey(lambda x,y: x + y).collect()
```

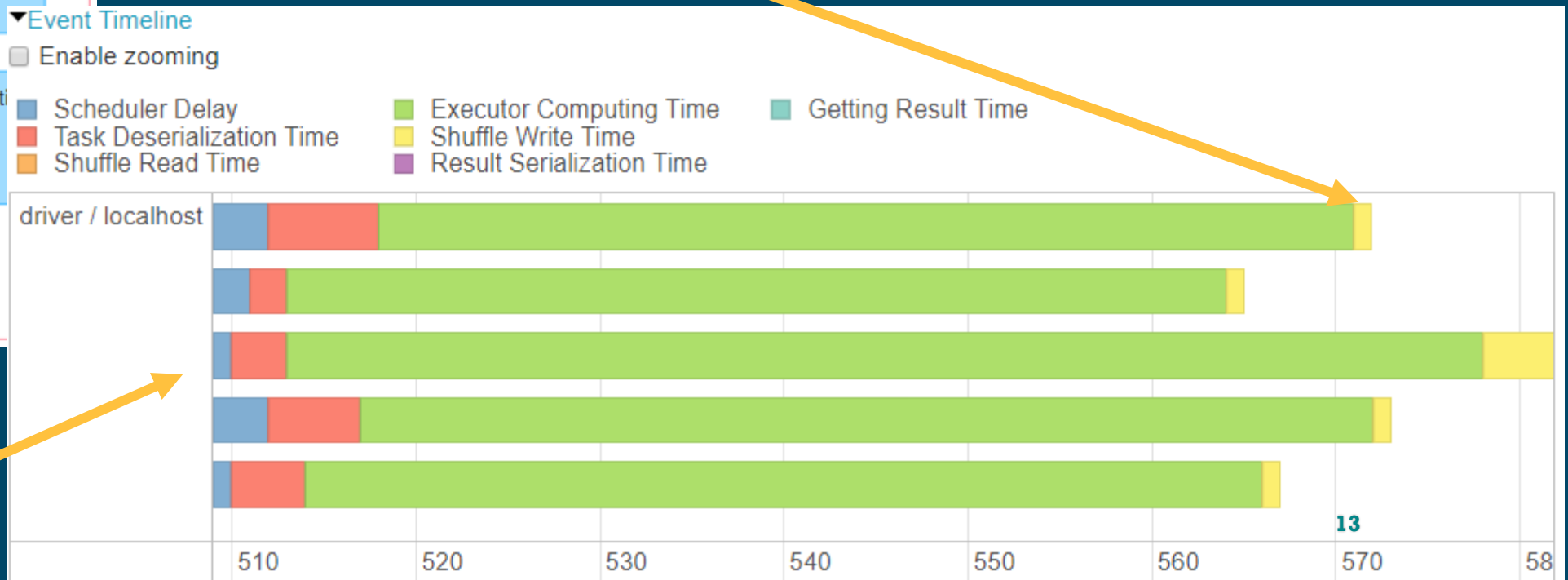
▾ (1) Spark Jobs

▸ Job 59 View (Stages: 2/2)





Shuffle shown @ end

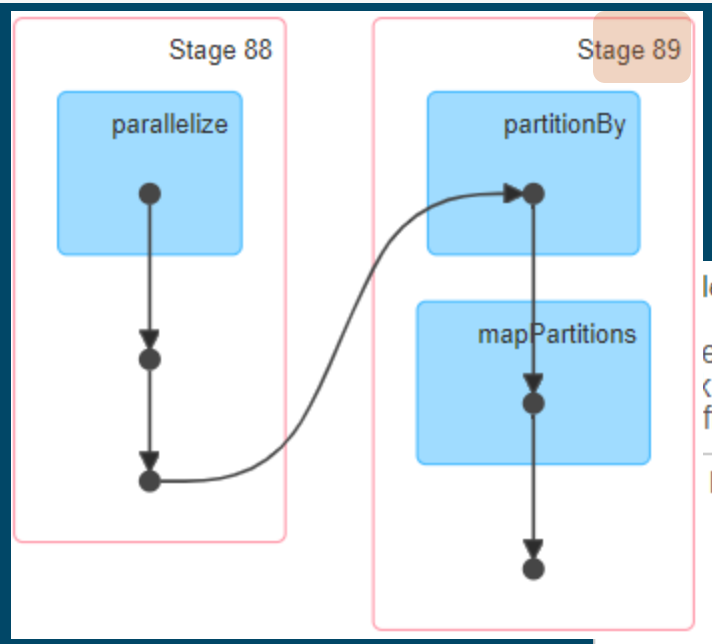


5 Tasks

Map Reduce with 5 tasks

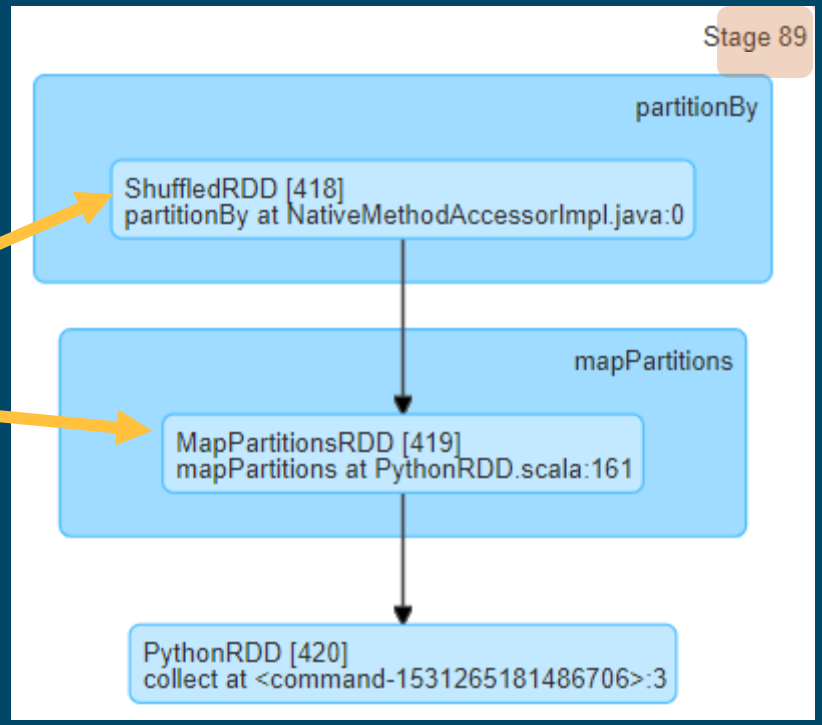
```
1 data = range(1,100)
2 rdd = sc.parallelize(data,5)
3 rdd.map(lambda x: (x%5, x)).reduceByKey(lambda x,y: x + y).collect()
```

▾ (1) Spark Jobs
▸ Job 59 View (Stages: 2/2)

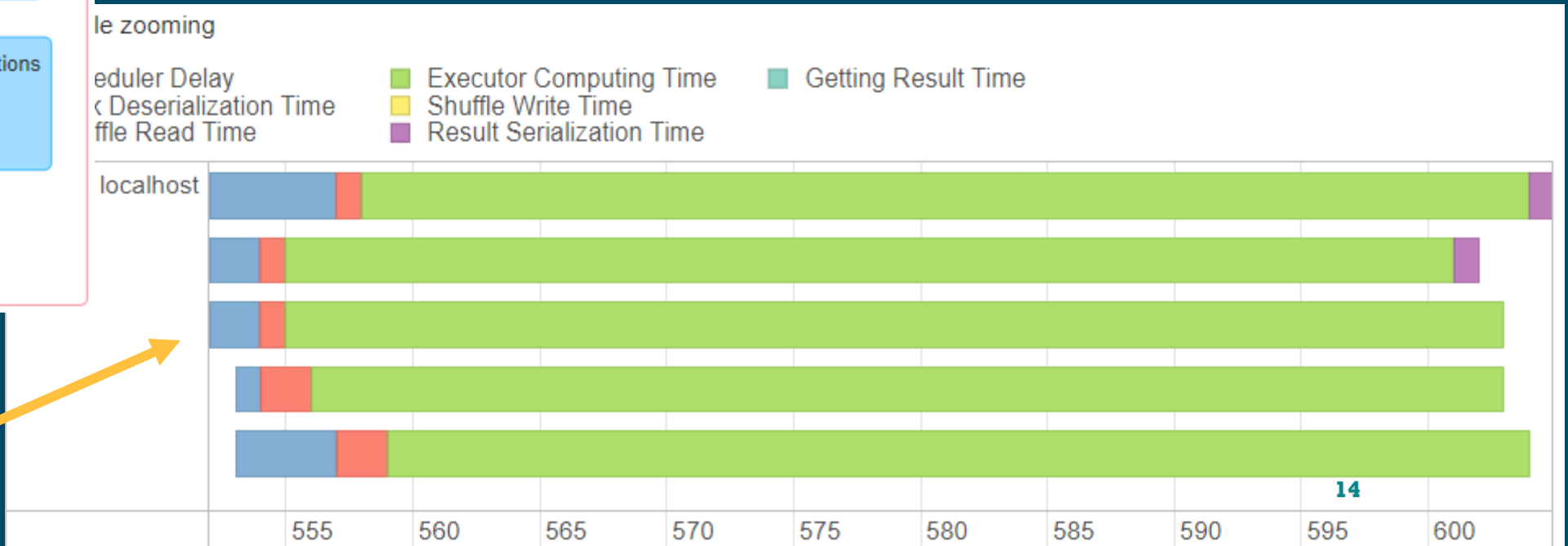


Shuffled

Reduce



5 Tasks



Map Reduce with 5 tasks

Stage Id ▾	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total
156	82615525489265433 78	data = range(1,100) rdd = sc.parallelize(data,5... collect at <command-1531265181486706>:3 +details	2018/11/02 21:09:34	50 ms	5/5
155	82615525489265433 78	data = range(1,100) rdd = sc.parallelize(data,5... reduceByKey at <command-1531265181486706>:3 +details	2018/11/02 21:09:33	0.3 s	5/5

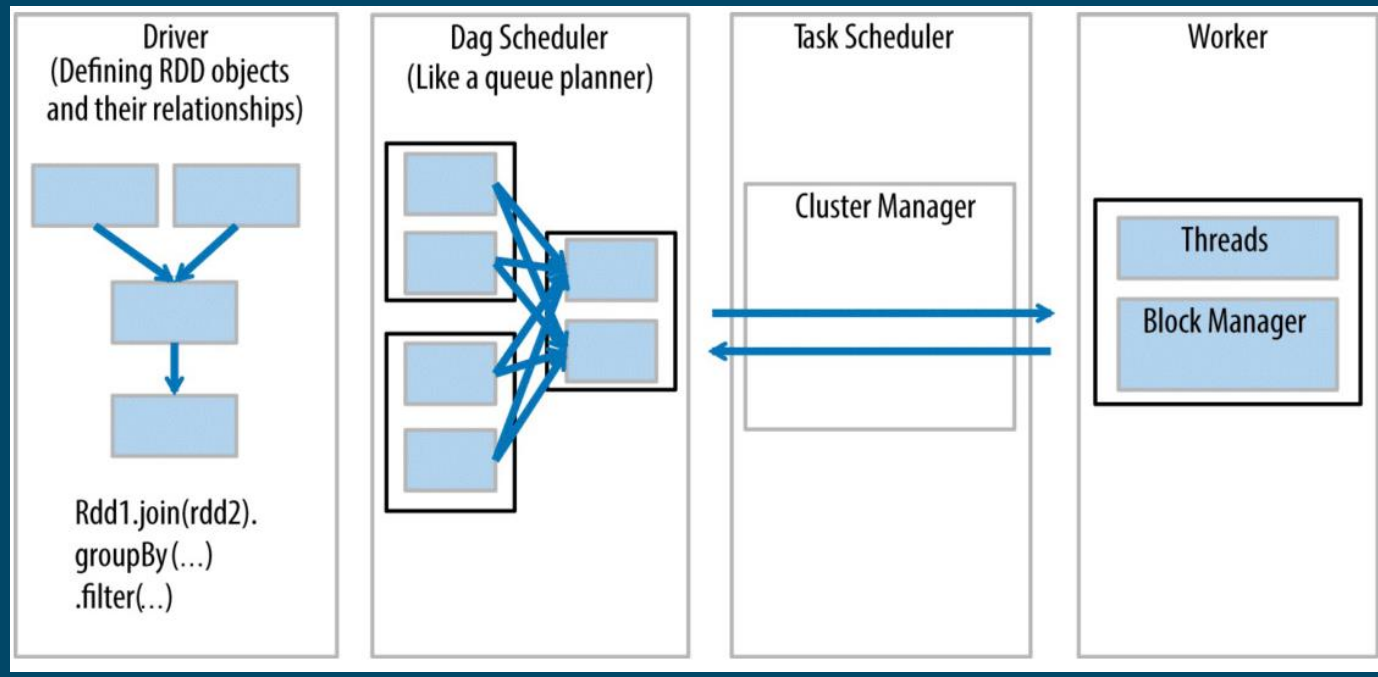
Spark’s Cluster Manager

Tasks are distributed over the cluster

Ideally, a task is placed on a node that has the data it needs

Spark Components

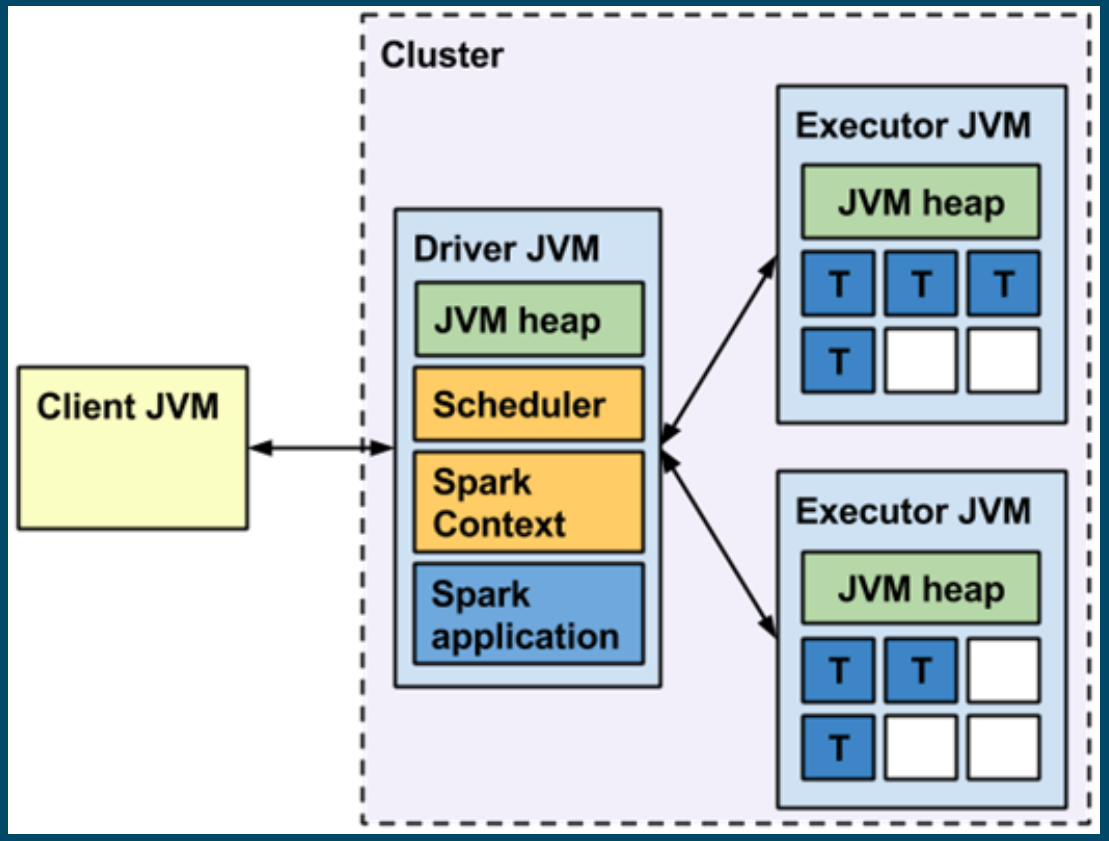
- Driver
  - Main program
- DAG scheduler
  - Plans the DAG execution
  - Includes memory caching
- Scheduler
  - Yarn, Mesos, etc., which manages the execution
- Worker
  - Execution on a node



Distributed task execution (Spark, Yarn, MR)

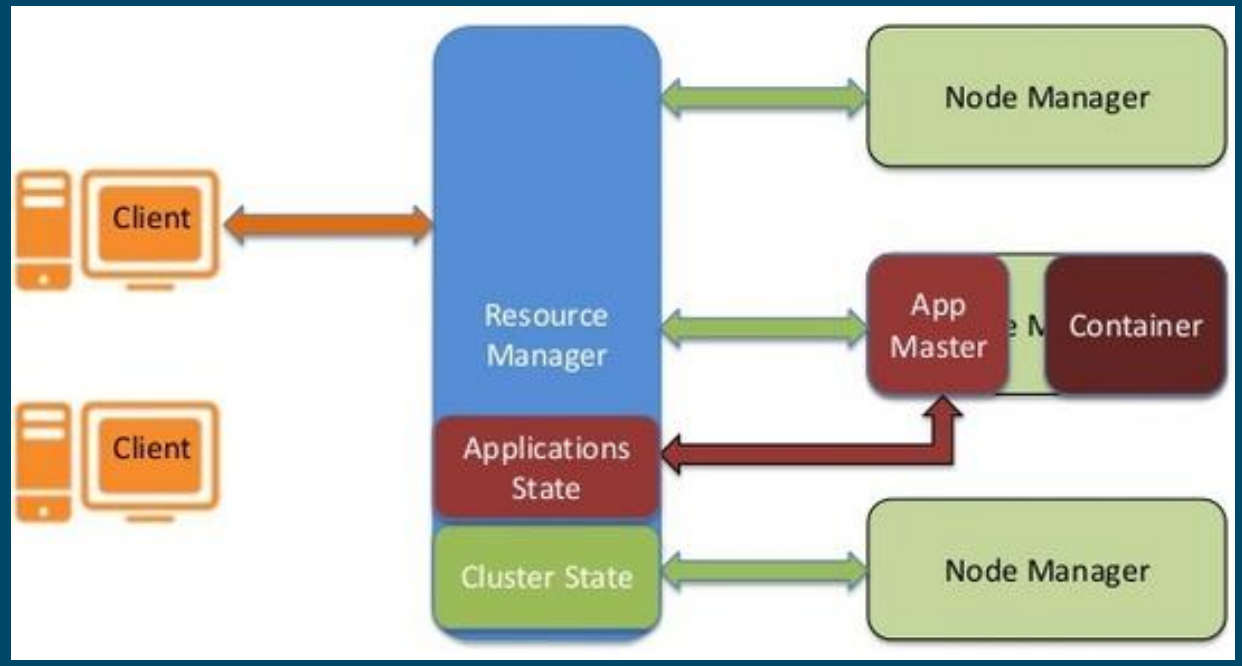
- Executor
  - JVM runs on a node
    - Can have >1 on a node
  - Executes multiple tasks
  - A task is thread in a JVM (executor)
- Task
  - Running (PySpark) code
  - Processes one data partition
- Scheduler
  - Specifies partitions to execute on nodes
  - Sends task (PySpark) code to nodes

Spark cluster mode



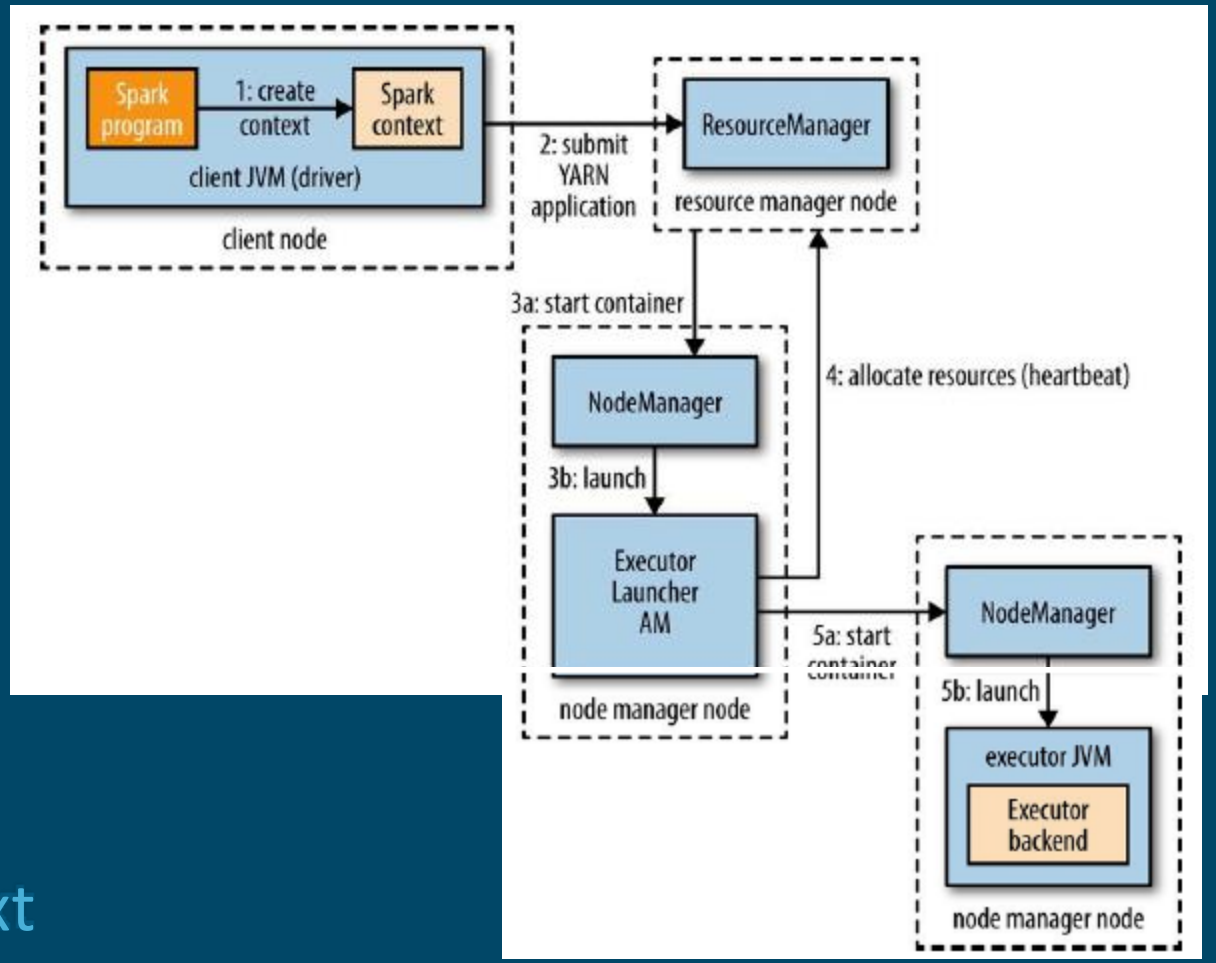
YARN Architecture

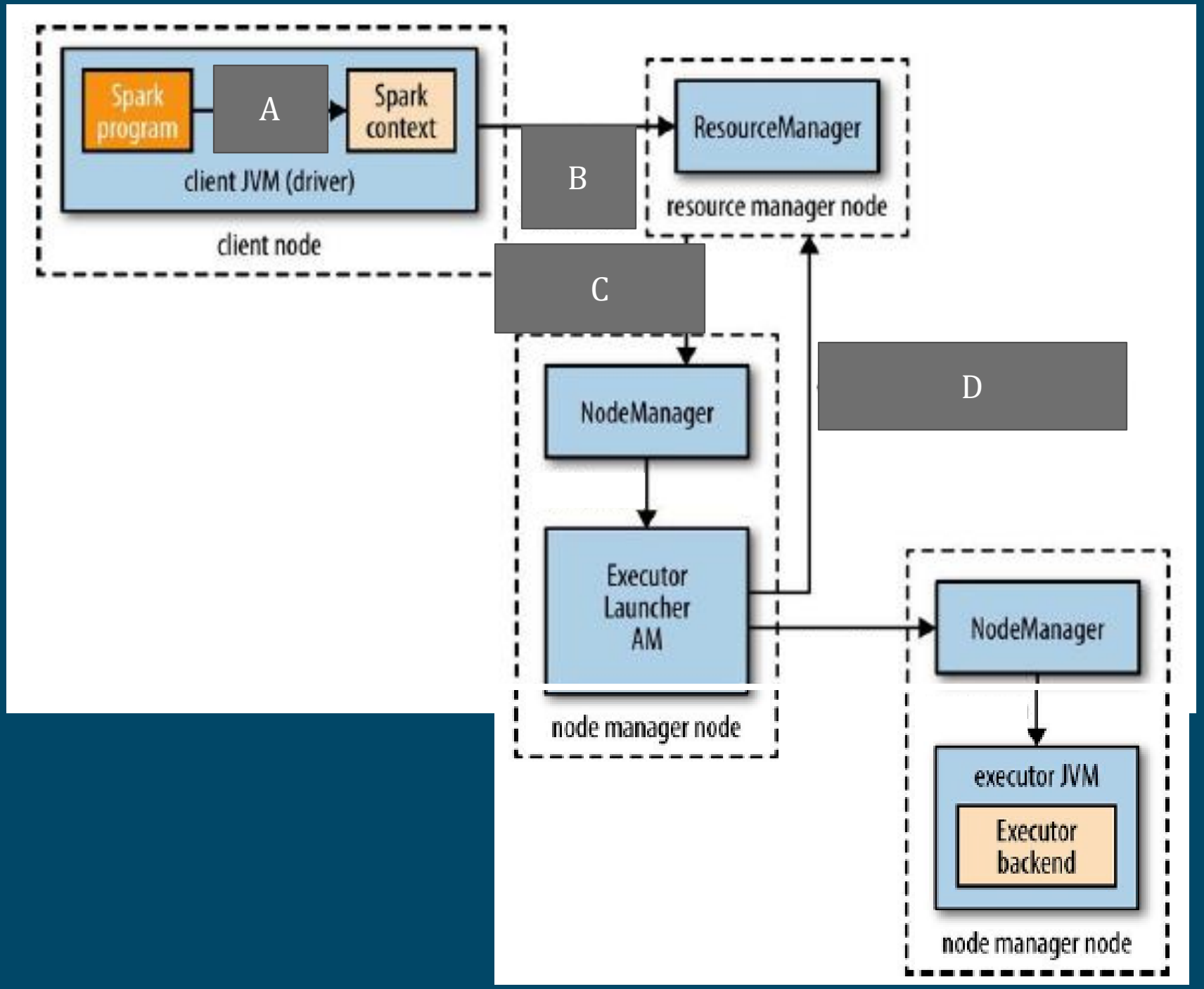
- Spark runs on a cluster
- Cluster managed by
  - YARN
  - Kubernetes
  - Etc.
- Spark works with YARN to schedule spark jobs and their tasks



Spark job submission: Client mode
(notebook or command line submission)

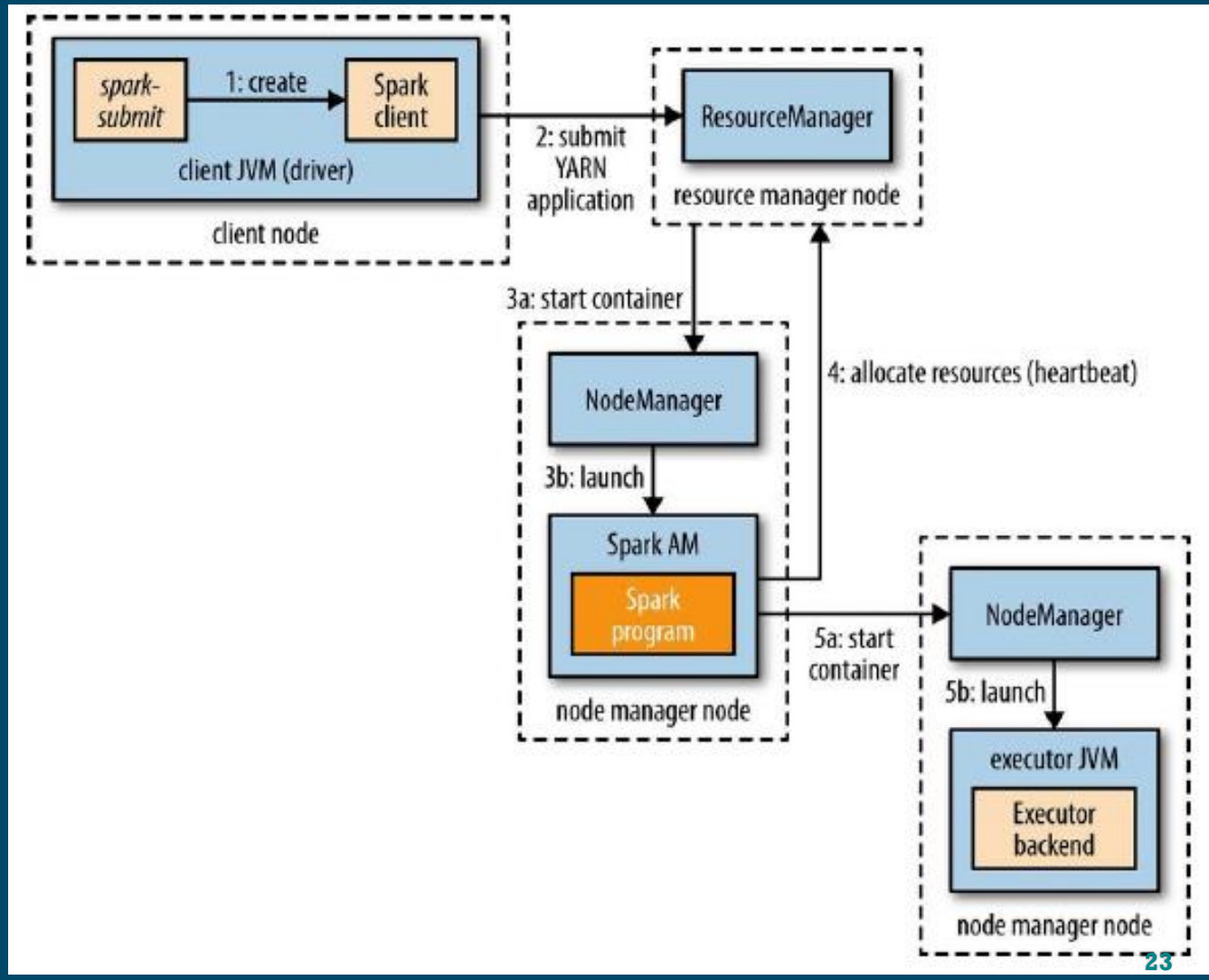
1. Driver creates SparkContext
2. SparkContext submits job to RM
3. RM starts container with AM
4. AM requests resources (files, CPUs, memory)
5. AM starts executors
   - A spark task runs in executor JVM
   - Executor registers with SparkContext




Spark program
A
Spark context
client JVM (driver)
client node
B
ResourceManager
resource manager node
C
D
NodeManager
Executor Launcher AM
node manager node
NodeManager
executor JVM
Executor backend
node manager node

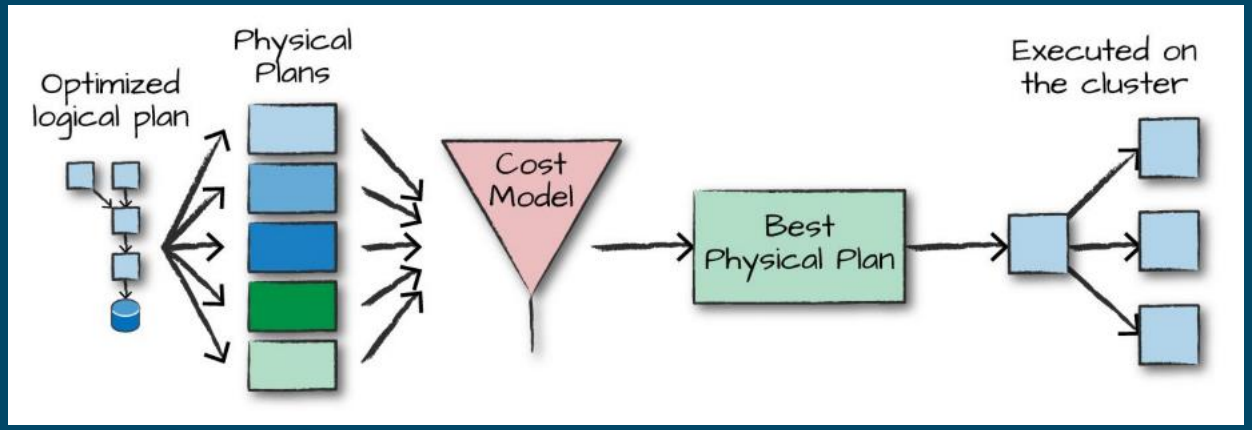
Spark job submission: Cluster mode
(spark-submit job)

1. Driver creates Spark client
2. Spark client submits job to RM
3. RM starts container with AM
4. AM requests resources (files, CPUs, memory)
5. AM starts the driver program and then the executors
   - A spark task runs in executor JVM
   - Executor registers with SparkContext
- Key difference is that driver program is on the cluster and not the client



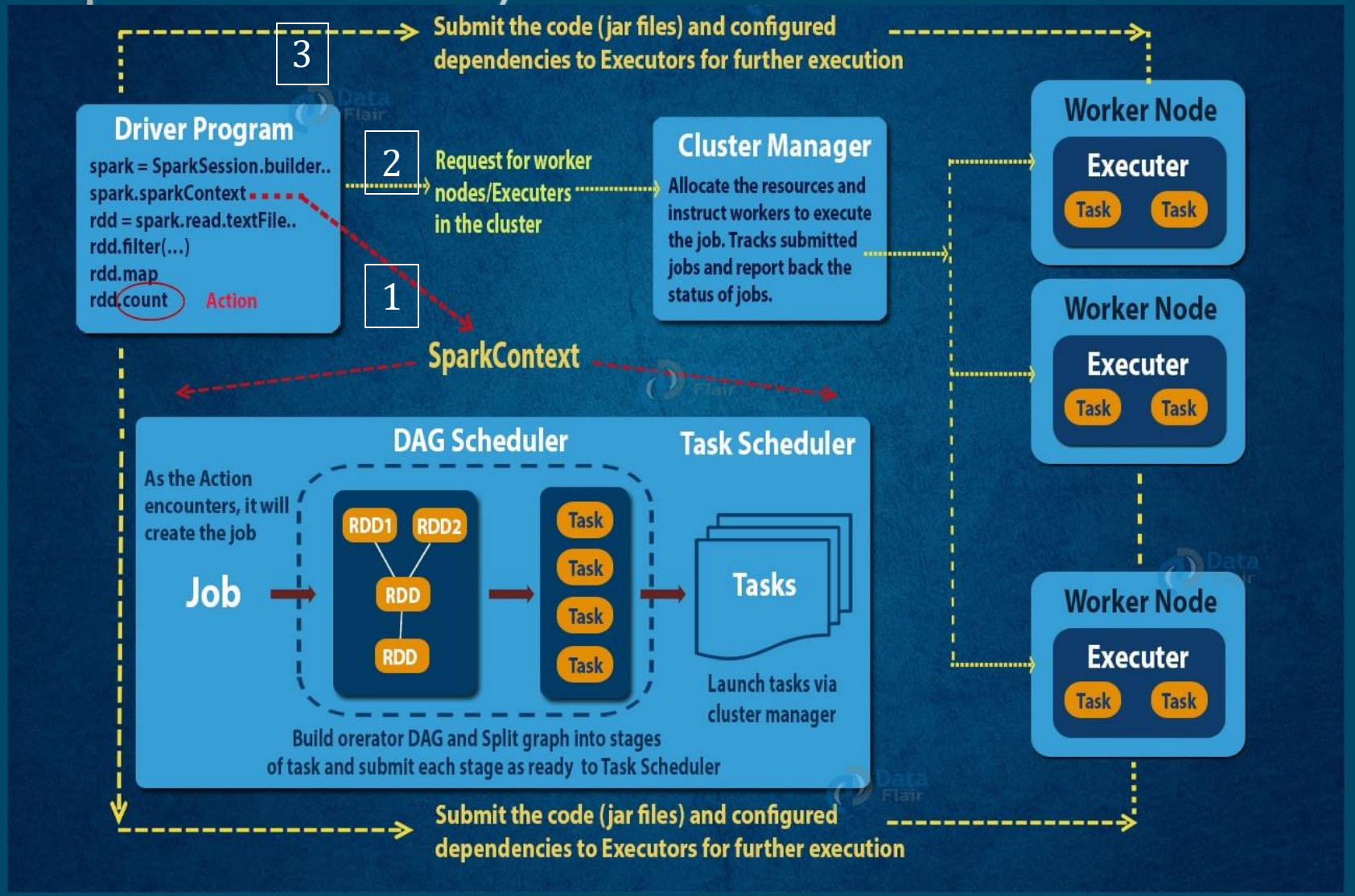
Program and data are analyzed to determine how to schedule tasks to data nodes

Lazy evaluation



- Spark waits until the last moment to execute the graph of instructions
- Transformations create a logical transformation plan
- Different languages (Scala, PySpark) can have the same plan, which compiles to the same code (mostly)
- An action triggers the computation.
  - Spark then compute a result from the series of transformations

Spark job execution
(conceptual overview)

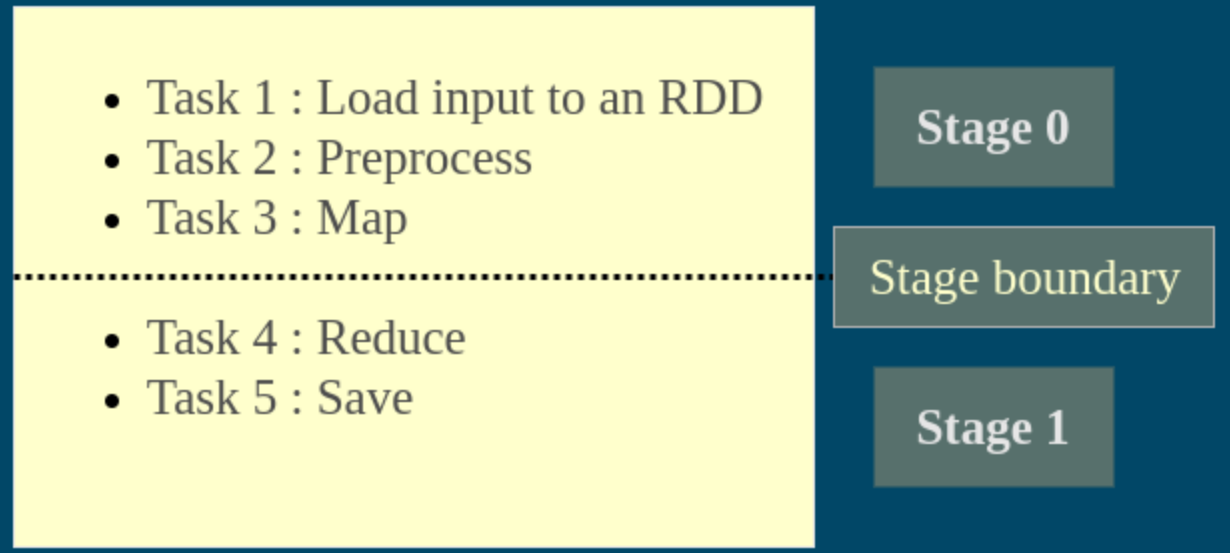


https://data-flair.training/blogs/dag-in-apache-spark/

Execution plan from code

- Task 1 : Load input to an RDD
- Task 2 : Preprocess
- Task 3 : Map
- Task 4 : Reduce
- Task 5 : Save

```
counts = sc.textFile("/path/to/input/")
           .flatMap(lambda line: line.split(" "))
           .map(lambda word: (word, 1))
           .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("/path/to/output/")
```



Execution plan visualized

```
counts = sc.textFile("/path/to/input/")
           .flatMap(lambda line: line.split(" "))
           .map(lambda word: (word, 1))
           .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("/path/to/output/")
```

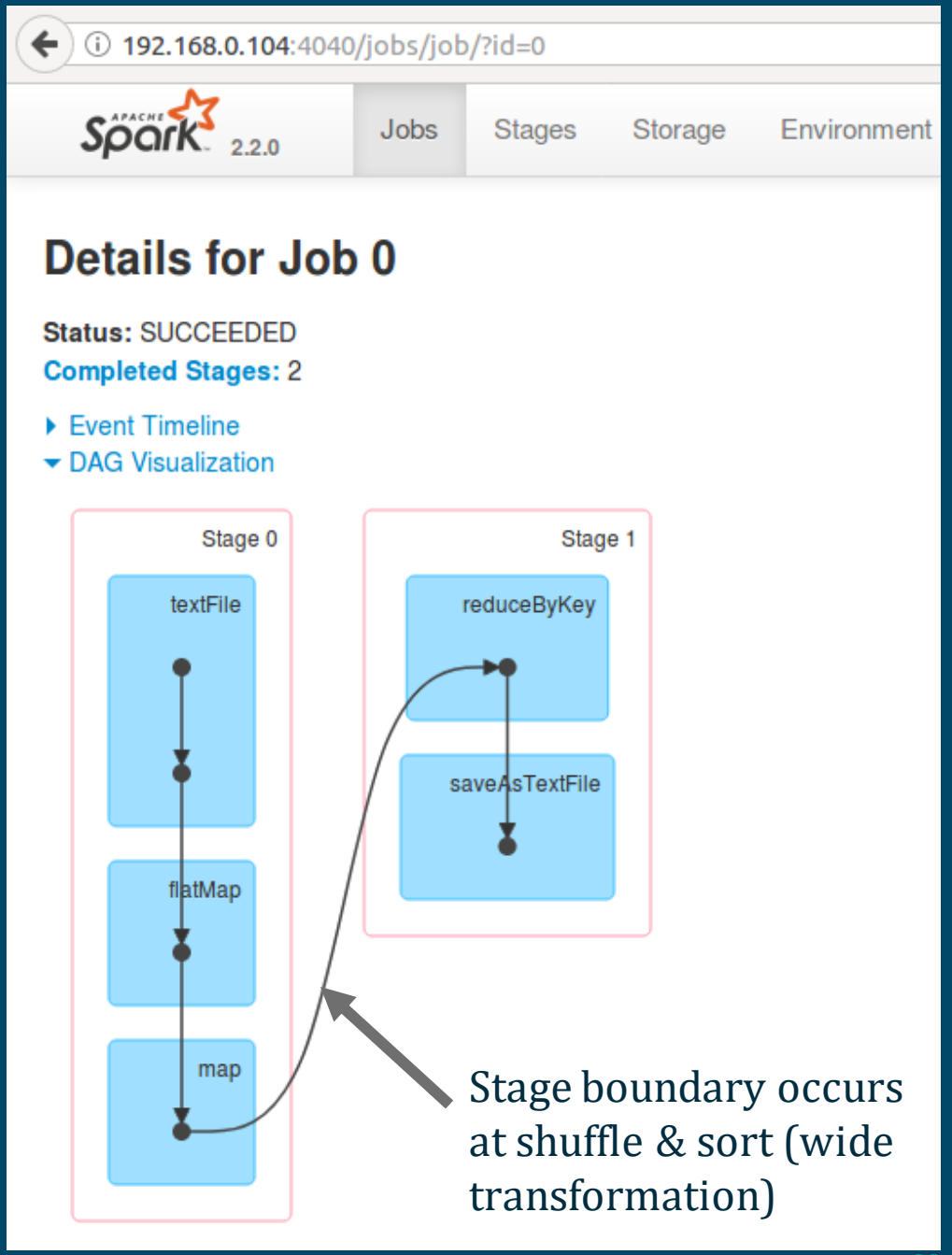
- Task 1 : Load input to an RDD
- Task 2 : Preprocess
- Task 3 : Map

Stage 0

Stage boundary

- Task 4 : Reduce
- Task 5 : Save

Stage 1



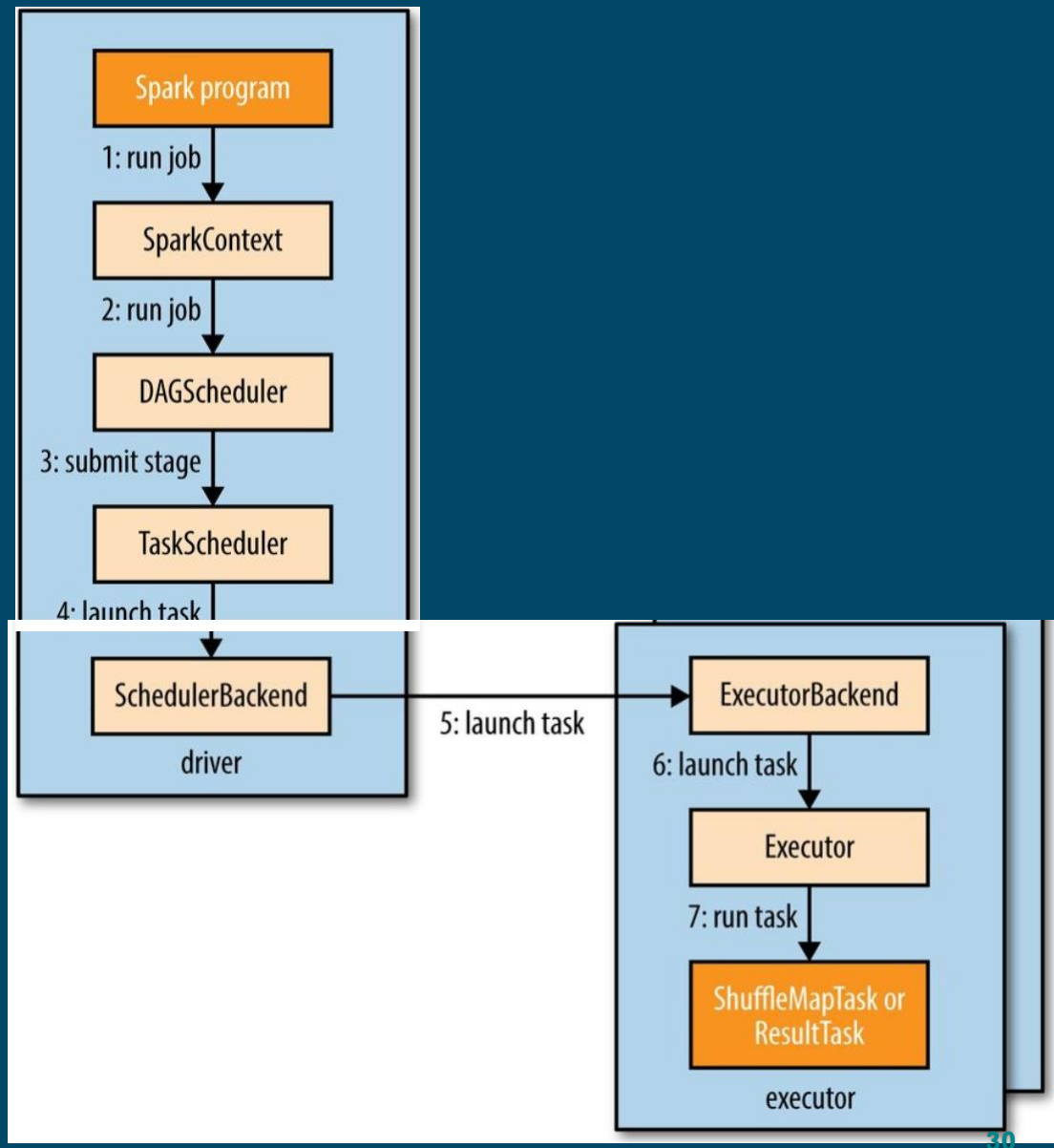
How Apache Spark builds a DAG and Physical Execution Plan ?

- Driver identifies transformations and actions present in the Spark application
  - Based on the flow of program, these tasks are arranged in a graph with directed flow of execution from task to task forming no loops in the graph (called DAG)
- DAG Scheduler creates a Physical Execution Plan from the DAG
  - Physical Execution Plan contains tasks and are bundled to be sent to nodes of cluster
  - Transformation that requires data shuffling between partitions, i.e., a wide transformation, results in a stage boundary
  - Optimization may combine stages

https://www.tutorialkart.com/apache-spark/dag-and-physical-execution-plan/

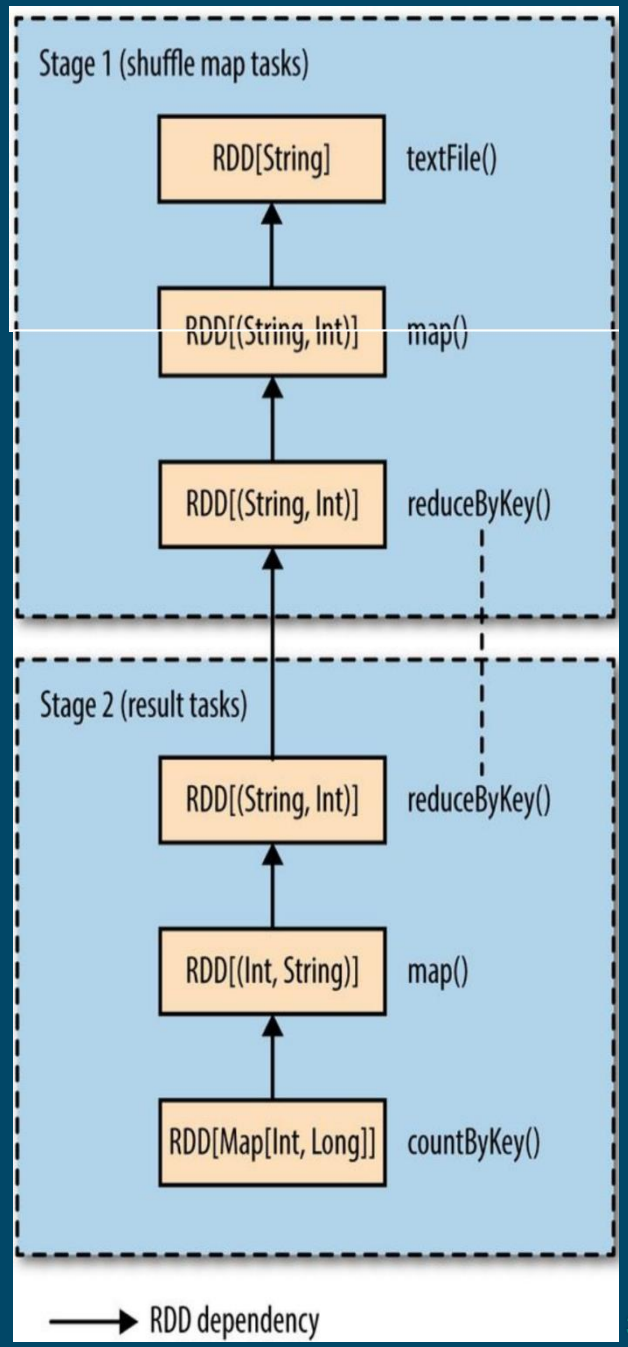
DAG scheduler & Spark Job Run

- DAG scheduler:
  - Shuffle map tasks
    - the map-side part of the shuffle in MapReduce.
    - Each shuffle map task **runs a computation on one RDD partition** and, based on a partitioning function, writes its output to a new set of partitions
  - Result tasks
    - run in the final stage that returns the result to the user's program
      - such as the result of a count()
    - runs a computation on its RDD partition, then **sends the result back to the driver**, and the driver assembles the results from each partition into a final result



Spark applies optimizations…

- DAG determines RDD dependencies
- Notice that the reduceByKey() transformation spans two stages
- this is because it is implemented using a shuffle, and the reduce function runs as a **combiner** on the map side (stage 1) and as a reducer on the reduce side (stage 2) — just like in MapReduce.

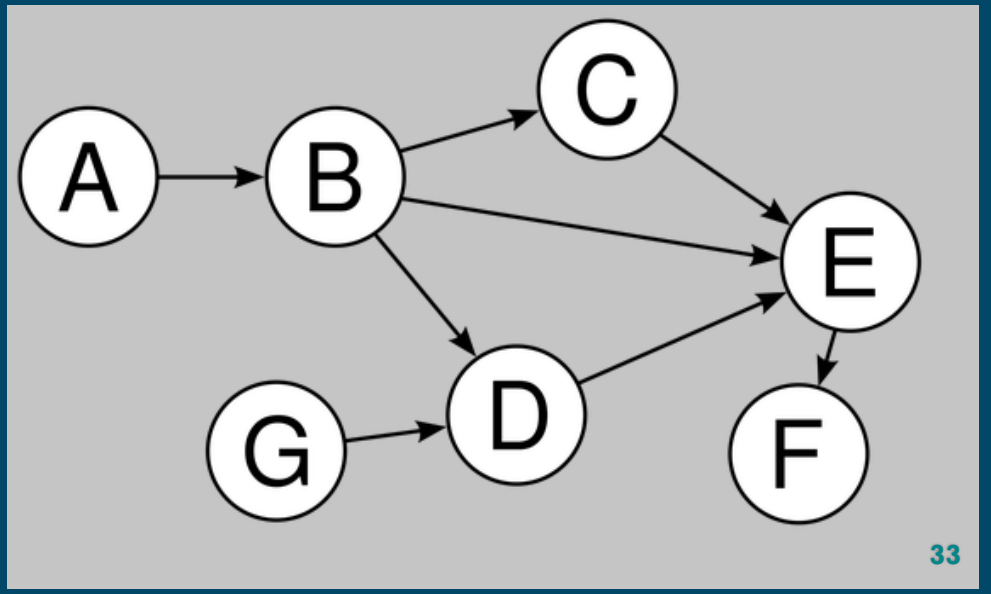


Spark DAG

Graph of computation used to optimize & track computation

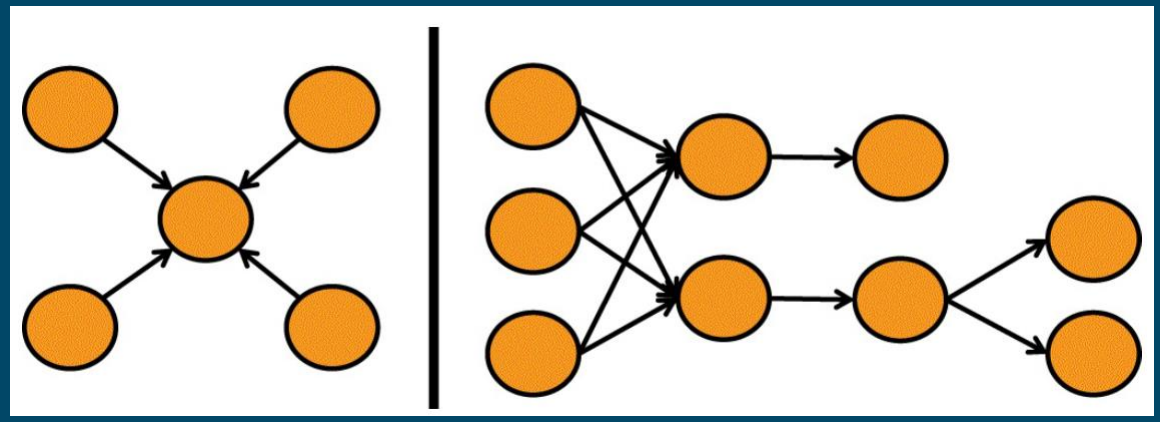
Spark is like Map Reduce, but…

- A Spark job is more general than a MapReduce job
  - Because it is made up of an arbitrary directed acyclic graph (DAG) of stages,
  - each of which is roughly equivalent to a map or reduce phase in MapReduce
    - Thus, more than 2 (map & reduce)
- A DAG
  - Has no loops



Spark DAG is better than MR workflow

- Complex chains of tasks are known as directed acyclic graphs, or DAGs
- Sequences of tasks
  - Repeated jobs in Map Reduce
  - Workflows in Ooozie (on MR)
  - Now, inferred and optimized by Spark
- Spark
  - the Spark engine itself creates those complex chains of steps from the application's logic, rather than the DAG being an abstraction added externally to the model.
  - This allows developers to express complex algorithms and data processing pipelines within the same job and allows the framework to optimize the job as a whole, leading to improved performance



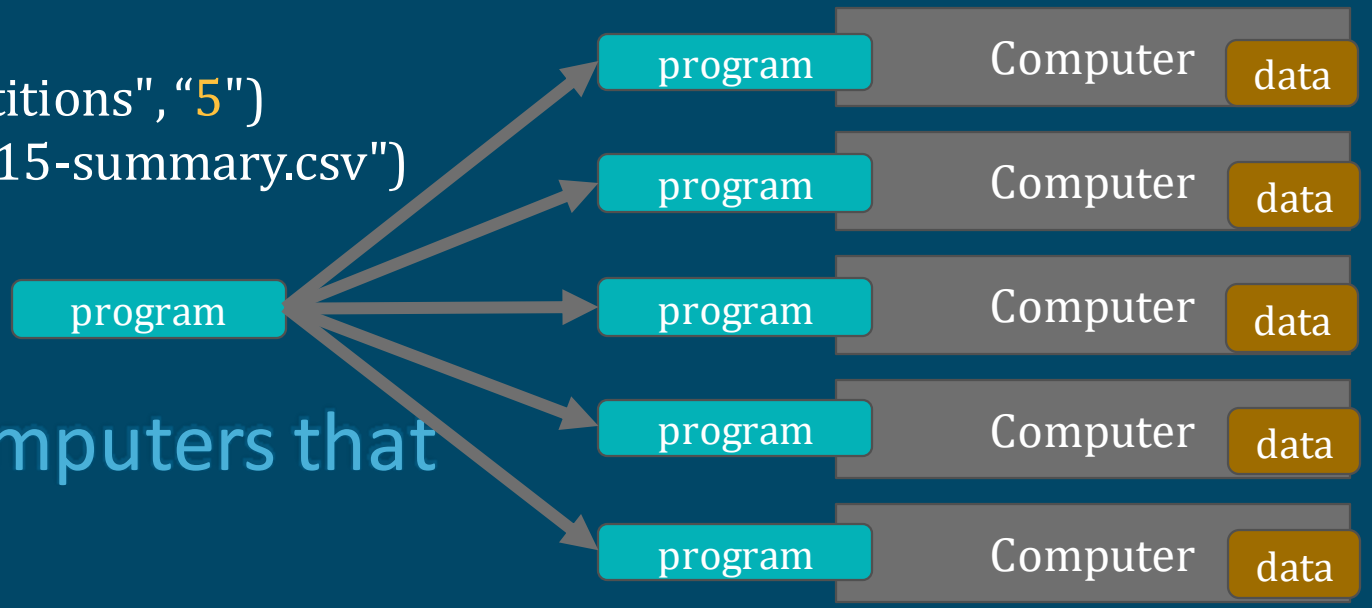
Data locality
place the tasks near the data to be processed

Program copied to computers that have the data

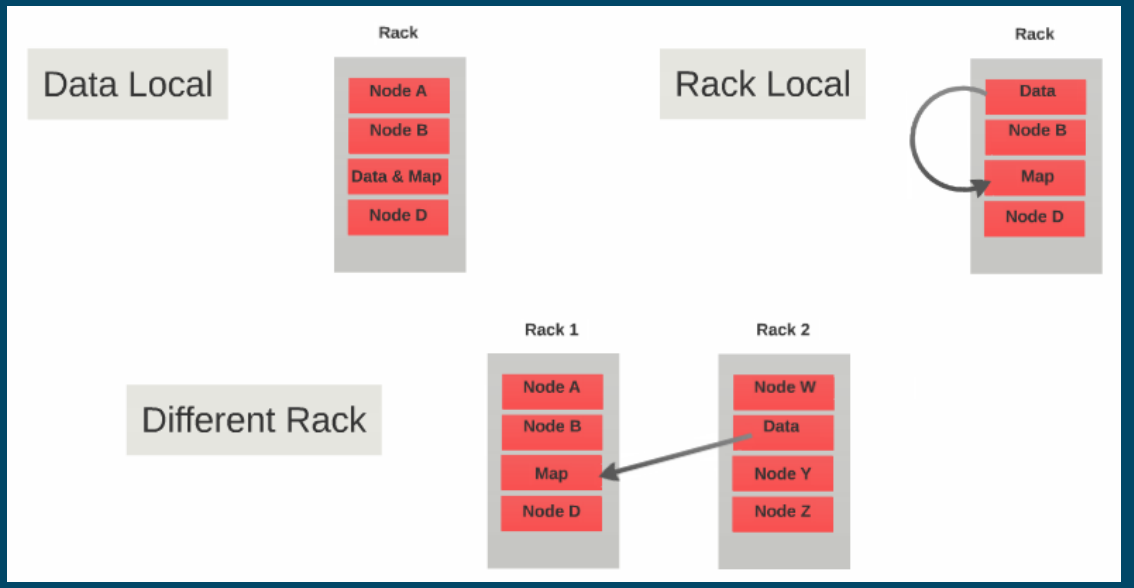
- PySpark

```
spark.conf.set("spark.sql.shuffle.partitions", "5")
flightData2015 = spark.read.csv("2015-summary.csv")
```

- Data locality
  - Program copied to computers that have the data
  - Run program on computers each accessing local data partitions



Spark data locality



- Execute tasks as close to where the data lives as possible, to minimize data transfer
- Kinds of locality
  - PROCESS_LOCAL
    - data is in the same JVM as the running code. This is the best locality possible
  - NODE_LOCAL
    - data is on the same node. Examples might be in HDFS on the same node, or in another executor on the same node. This is a little slower than PROCESS_LOCAL because the data has to travel between processes
  - NO_PREF
    - data is accessed equally quickly from anywhere and has no locality preference
  - RACK_LOCAL
    - data is on the same rack of servers. Data is on a different server on the same rack so needs to be sent over the network, typically through a single switch
  - ANY
    - data is elsewhere on the network and not in the same rack

https://stackoverflow.com/questions/31006632/spark-hadoop-data-locality

Data locality

Tasks

Index	ID	Attempt	Status	Locality Level	Executor	Launch Time	Duration
2	2748	0	SUCCESS	PROCESS_LOCAL	2.compute.internal	2014/09/18 00:09:56	2 ms
1	2747	0	SUCCESS	PROCESS_LOCAL	2.compute.internal	2014/09/18 00:09:56	2 ms

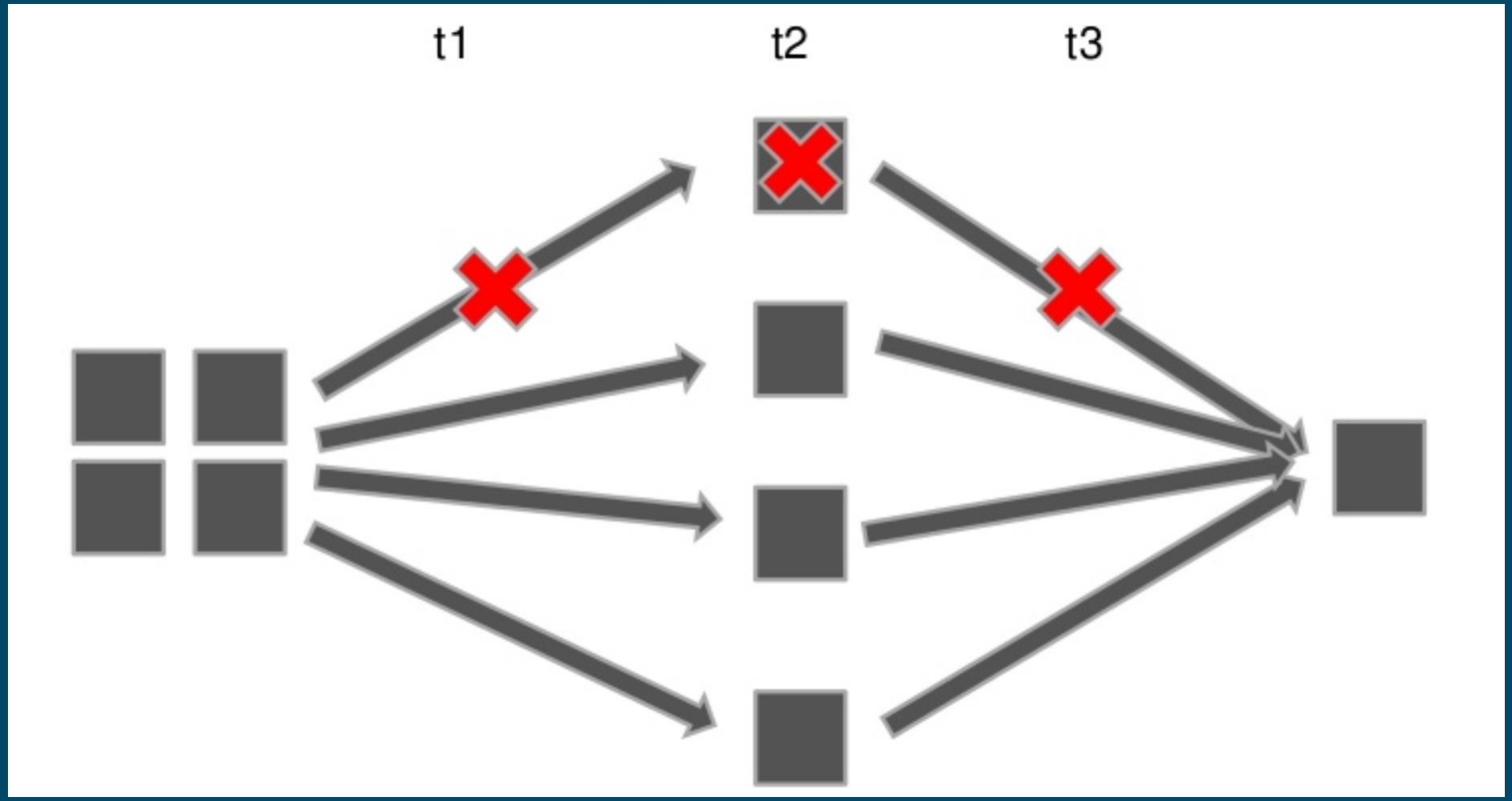
- Spark on YARN & HDFS
  - YARN will determine data locality
  - Spawn executors close to data
- SparkContext locality hints
  - Without YARN & HDFS, need to provide location hints

```
val preferredLocations = InputFormatInfo.computePreferredLocations(
    Seq(new InputFormatInfo(new Configuration(), classOf[TextInputFormat],
    inputPath)))
val sc = new SparkContext(conf, preferredLocations)
```

Spark RDD lineage and task recovery

What spark does when tasks fail

Data processing can fail!



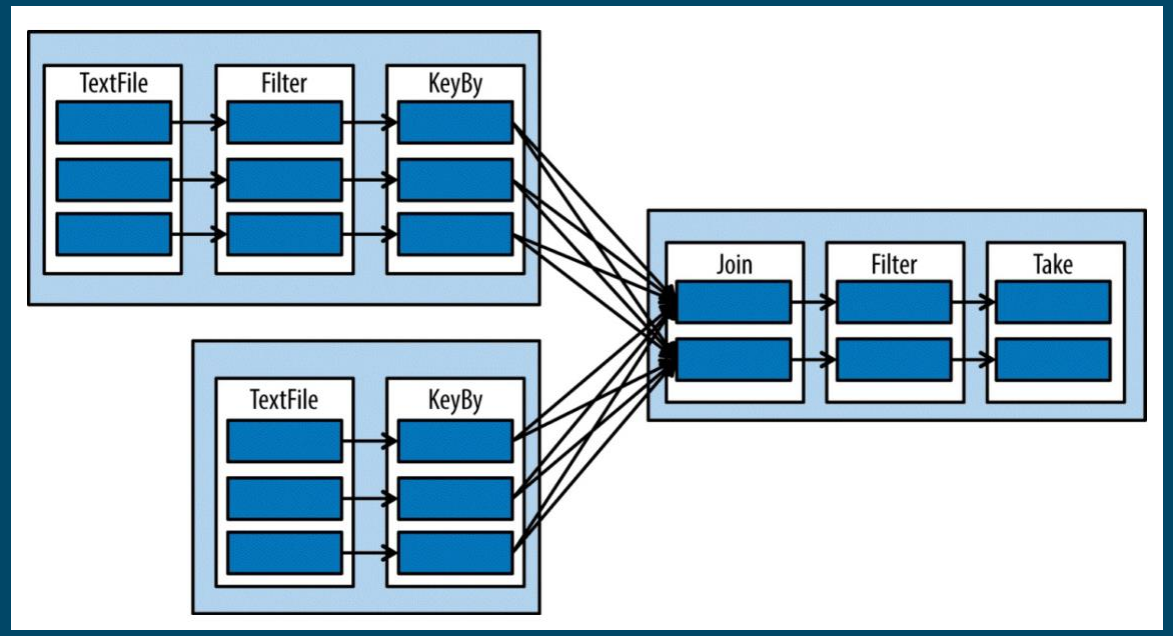
Resilient Distributed Datasets

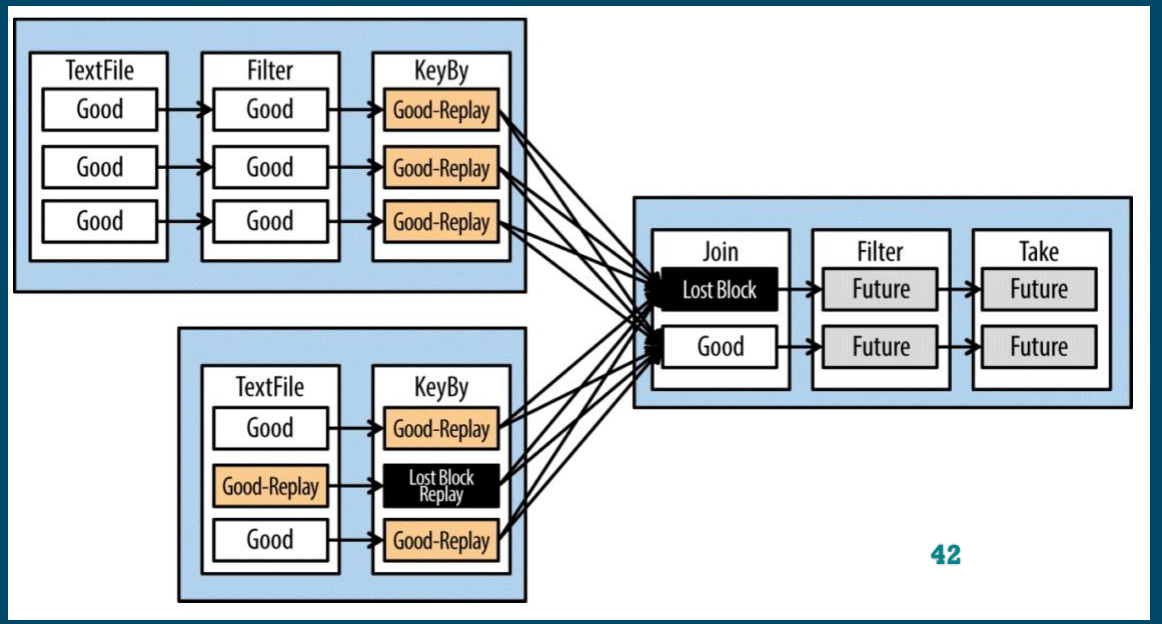
- Collections of serializable data
- RDDs store their **lineage** —
  - the set of transformations that was used to create the current state, starting from the first input format that was used to create the RDD.
  - If the data is lost, Spark will replay the lineage to rebuild the lost RDDs so the job can continue.
  - Can be very long, consuming lots of resources
    - Can do checkpoint in case of crashes

Lineage is an exam question

Spark RDD Recovery

- Replay task as necessary





Important to remember

- Spark distributes tasks over cluster and manages parallel execution
  - Data is already (1) partitioned, and (2) distributed over cluster
  - For each partition, one Spark task is created for its processing
    - Data is processed in memory, unless its too large and must be spilled to disk
  - Driver creates a DAG, which is a logical execution plan based on transformations over the partitions
  - Scheduler creates a physical execution plan from the DAG
  - If a task fails, then RDD lineage is replayed to recover from failure