Stelios Sotiriadis

9. Big data analytics with Apache Spark

Plan for today

- Lecture part:
  - Big data analytics
  - In-memory analytics with Apache Spark
- Lab session:
  - Deploy Spark on Docker
  - Using Python to submit Spark tasks

Why do you need to learn Spark?

- Spark is a big data ecosystem to run tasks in parallel
  - Unified Analytics Engine for Big Data
- Very active community of contributors
  - Key skill for data scientists/data engineers/big data analysts
- Tasks to master:
  - Concept of in-memory processing
  - Being able to write a spark job
  - Transform a serial execution to a parallel processing program

Is it difficult to write Spark programs?

- No and Yes...
  - It is much easier than Hadoop MapReduce!
- Spark uses less code than other programming tools (e.g. Java in Hadoop), but still it is challenging
  - Spark written in Scala, but you can use Python & R for data analysis
- When I started learning Spark...
  - my main problem was the huge amount of new terms
  - after this, it is all about getting used with how Spark works (and keep programming)

Basic terminology

- What is a Batch job?
- What is a Streaming job?
- What is a Master client approach?
- What is a DataFrame?
- What is a Resilient Distributed Datasets (RDD)?
- What is a Lazy evaluation?
  - What is a Transformation?
  - What is an Action?

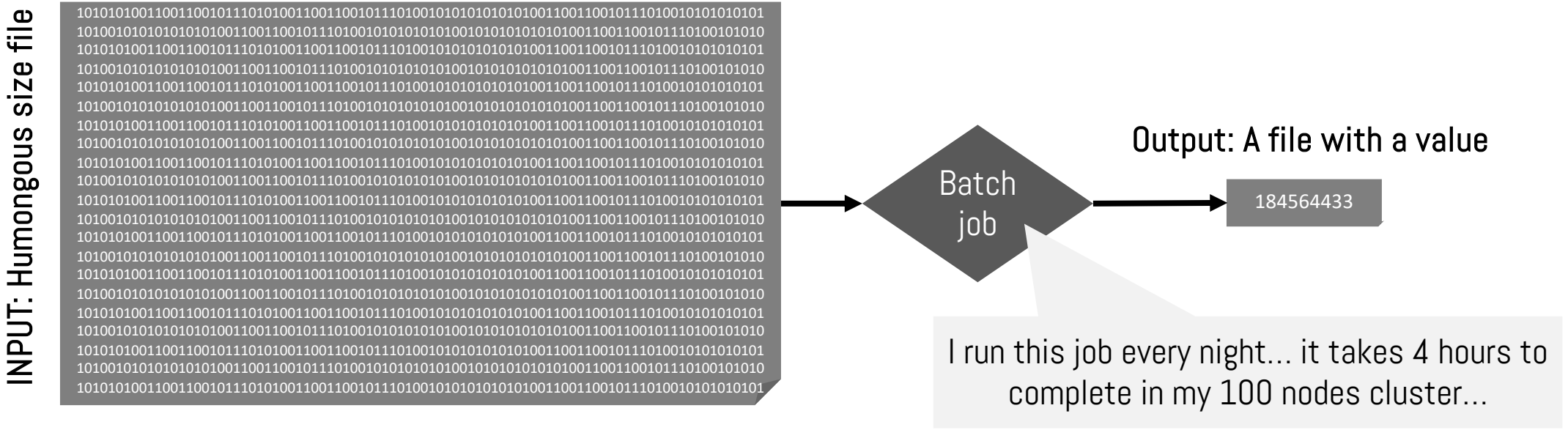
Questions about big data!

1. Name **10** Python data structures!
2. What data structure is ideal for parallel processing?

1. Lists
2. Dictionaries
3. Sets
4. Tuples
5. Stacks
6. Queues
7. Linked Lists
8. Binary Trees
9. Heaps
10. Dataframes

Batch job

- In YouTube...,
  - Every 24 hours a batch job reads all the data collected during the 24-hour window and computes the watch time minutes for that period
  - At the end, we have just a number (we store this to a file)



What is the biggest challenge?

- This is an example of a streaming application:
    1. The app collects blood pressure from a Bluetooth device every 10 minutes
    2. The app streams the data to a database system (for storage)
    3. Then, it applies a processing method
        - Classification problem: Take a pill/Don't take a pill
    4. The app completes the analysis before the next analysis starts
    5. The app sends recommendations to the user…
        - For example: Take a pill!

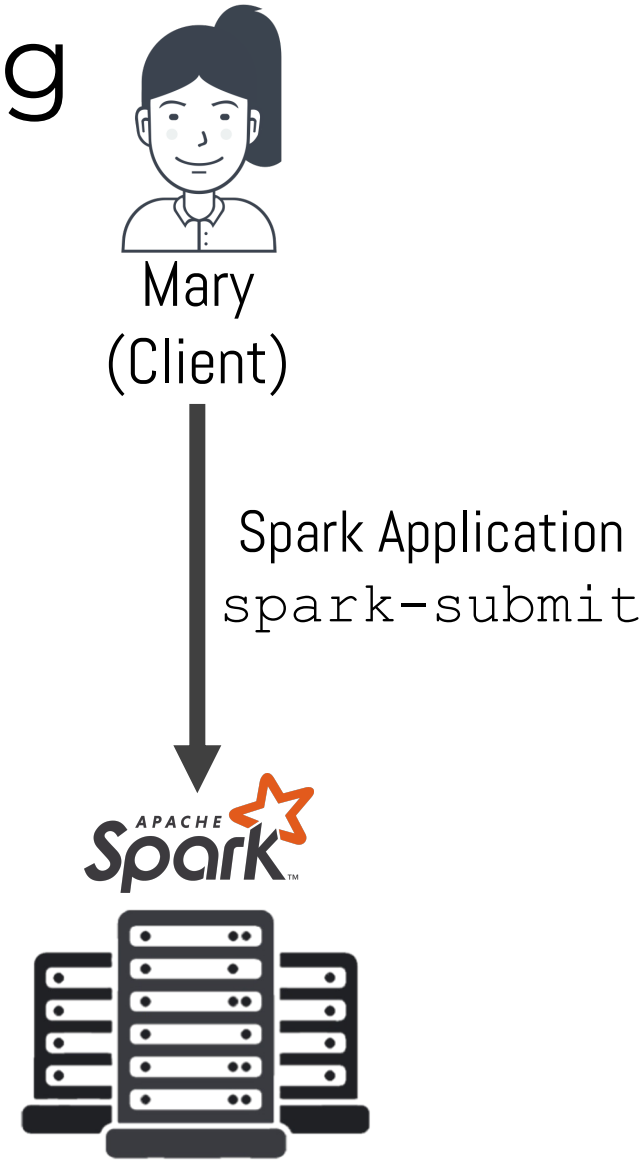
Analysis could take more than 10 minutes (we lost our deadline…)

Streaming applications

- What happens when a scheduled job runs longer than the scheduled interval?
  - For example, if a job is scheduled to be executed every 10 minutes and takes 35 minutes to complete, does the job execute again while running?
- Big data streaming is **a process in which big data is quickly processed to extract real-time insights.**
- The data on which processing is done is the data in motion
- Big data streaming is ideally a speed-focused approach wherein a continuous data stream is processed.

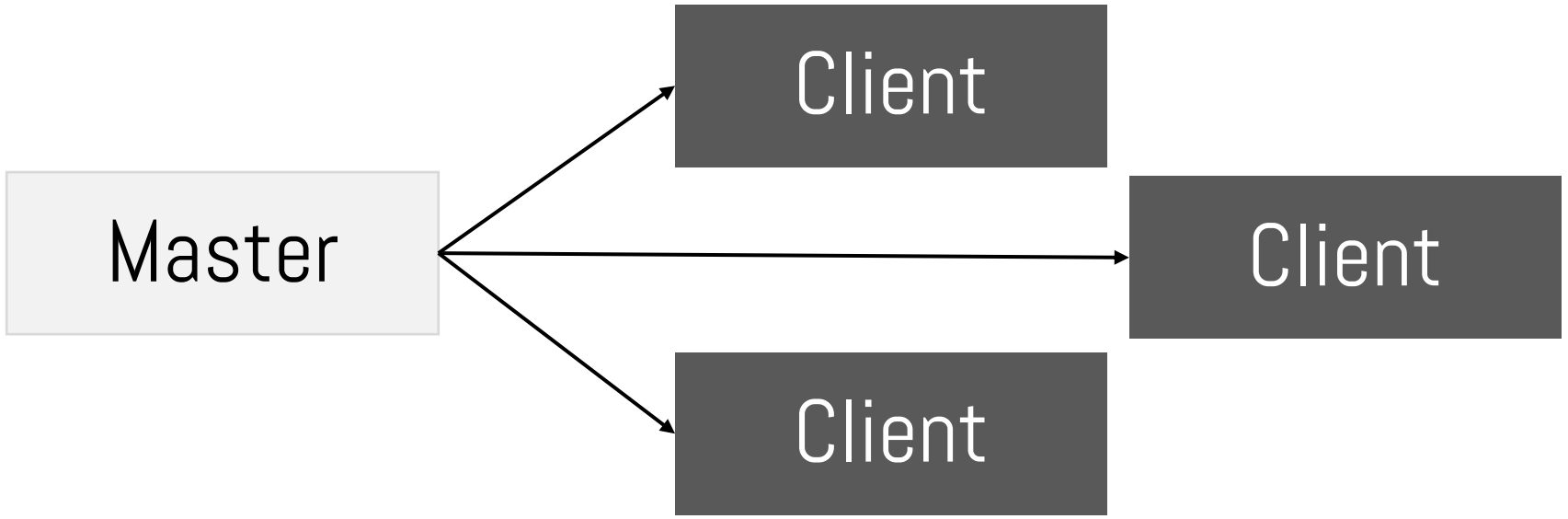
Spark is for batch and streaming

- A periodic running batch job
  - Ideal for Hadoop MapReduce
  - You can run it in Spark (as in Hadoop)
- A long running streaming job
- To run jobs, we can use an interactive client:
  - Run a Python notebook or in shell
    - Ideal for testing
  - Submit a job to a cluster for execution
    - Ideal for production (using the spark-submit)



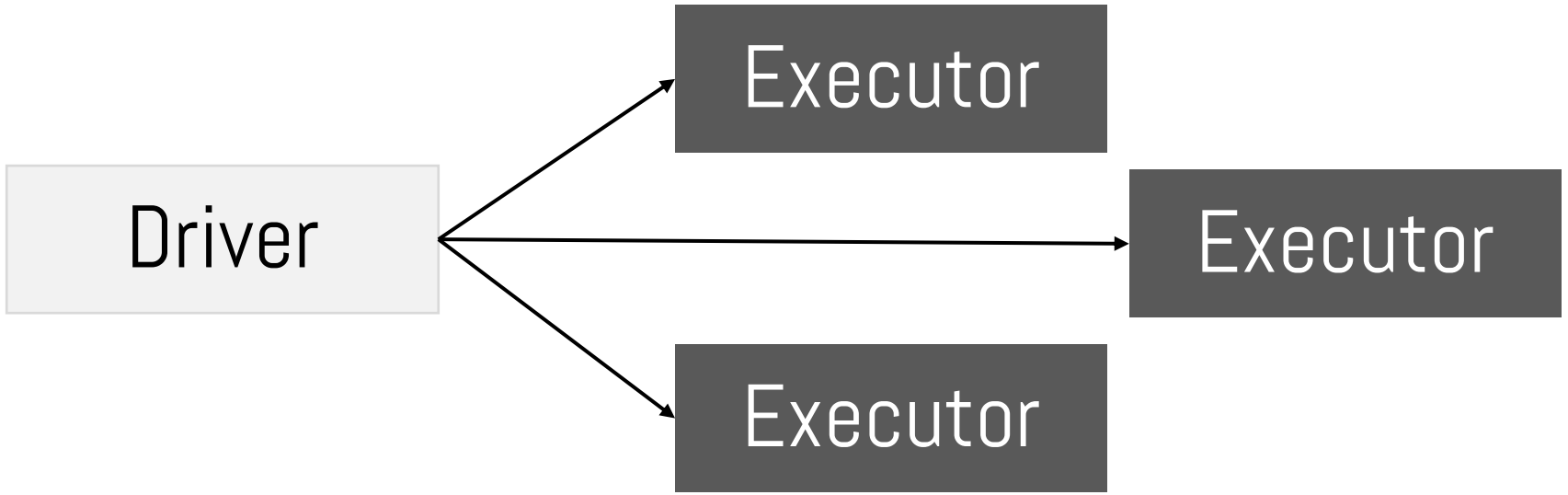
Master – Client approach

- A model for a communication protocol in which one device or process (known as the **master**) controls one or more other devices or processes (known as **client**).



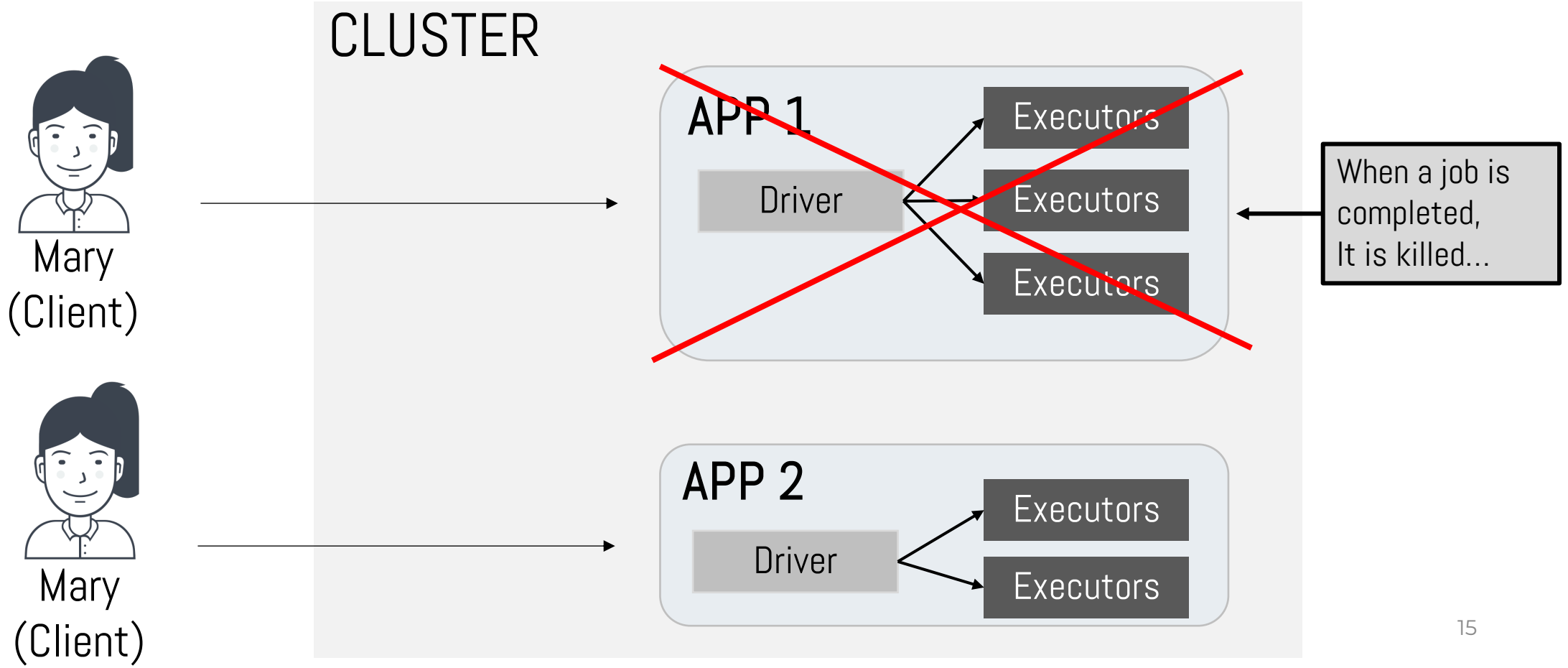
Spark: Master → Driver & Client → Executor

- A model for a communication protocol in which one device or process (known as the **master**) controls one or more other devices or processes (known as **clients**).



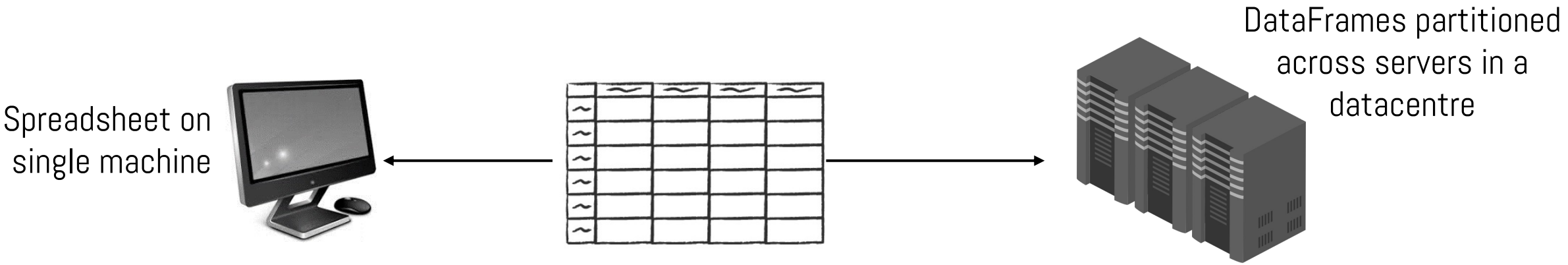
Submitting applications in Spark

- Each application creates a driver and a set of executors



What is a dataframe?

- **DataFrame** is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labelled axes (rows and columns).



RDD: a distributed dataset which is tolerant to failure

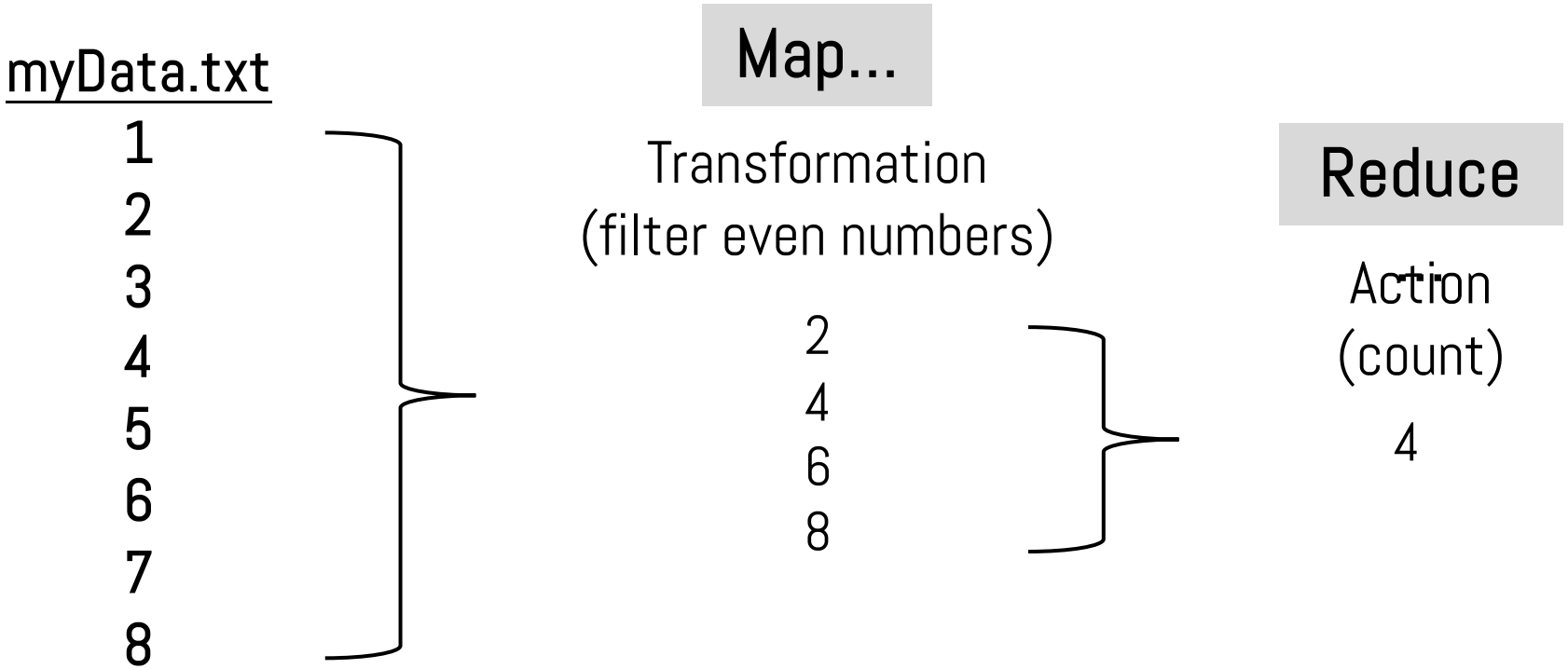
- **Resilient Distributed Datasets (RDD)** is a fundamental data structure of Spark
- It is an immutable distributed collection of objects
- Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster
- RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes
- RDD is a read-only, partitioned collection of records

Why is spark Resilient Distributed Datasets (RDDs) are immutable?

- Immutability rules out a big set of potential problems due to updates from multiple threads at once
- Both **Iterative** and **Interactive** applications require faster data sharing across parallel jobs.
  - **Iterative**: store intermediate results in a distributed memory
  - **Interactive**: keep data in memory for multiple usage
- The key idea of spark is RDDs, that support in-memory processing computation
  - It stores the state of memory as an object across the jobs, and the object is sharable between those jobs
  - Data sharing in memory is 10 to 100 times faster than network and Disk

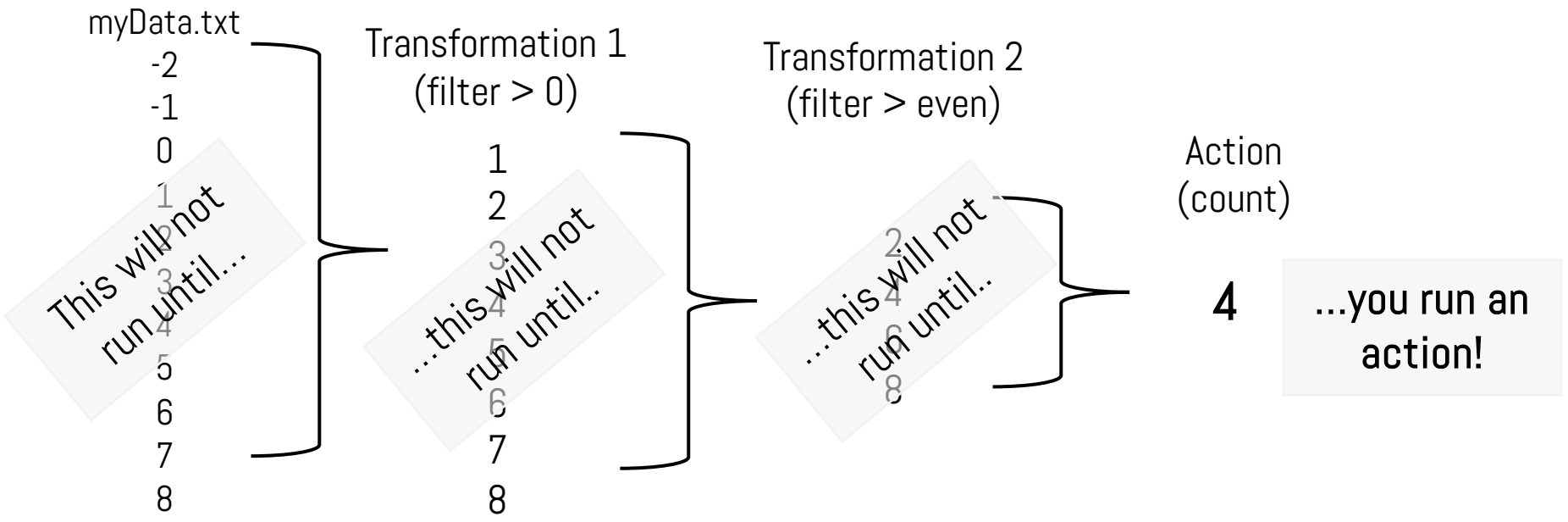
RDDs: Types of operations

- Transformations:
  - Transform data from one dataset to another
- Actions:
  - When transformation is done, we apply a computation!



Don’t run until…

- Spark delays the evaluation of an expression until its value is needed
  - It is done lazily (when I need it…)
- All transformations are lazily evaluated until you apply an action on an RDD

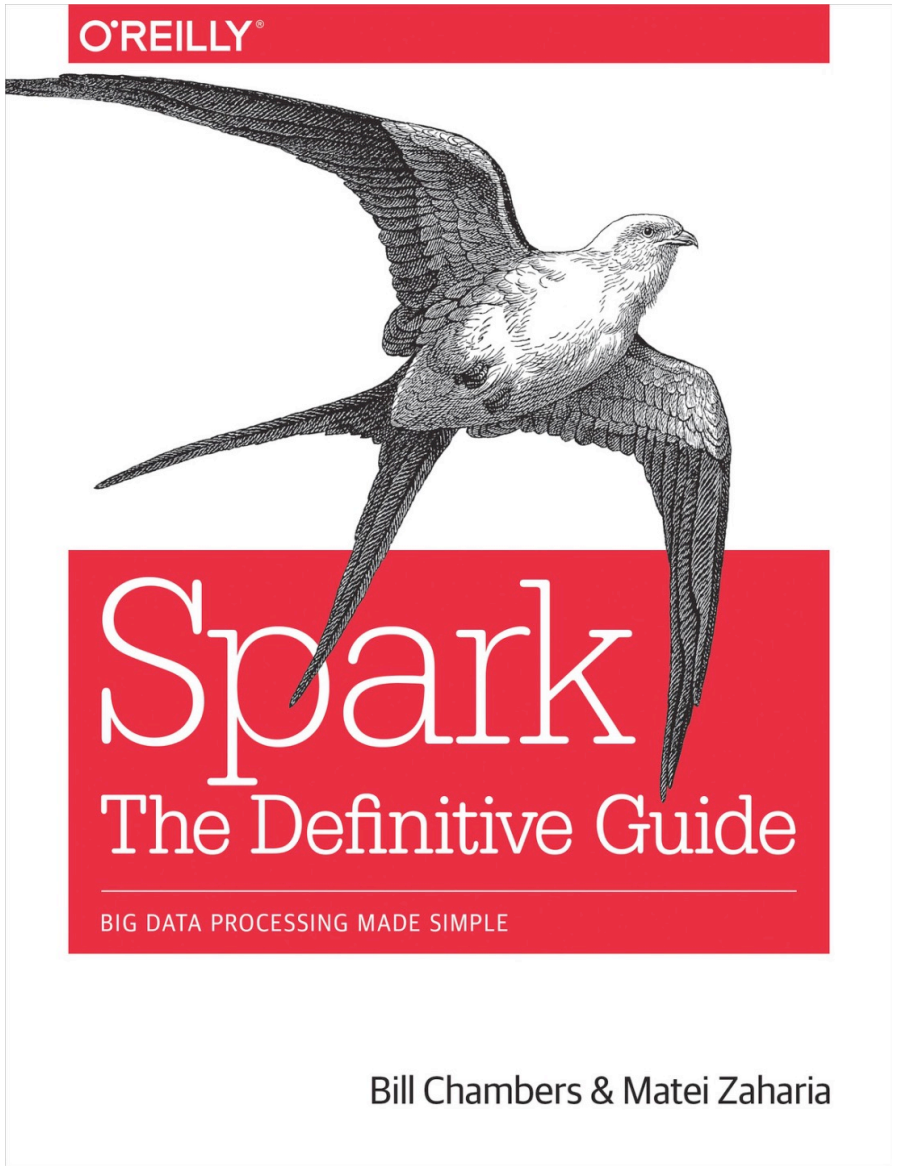


Lazy evaluation

- It reduces the time complexity of an algorithm by discarding the temporary computations and conditionals
- It reduces the running time of certain functions
- We run when we are ready!

Today’s class is on Spark

- Today we will talk about:
  - Big Data analytics with Apache Spark
- The lecture is based on:
  - Spark, The Definitive Guide, Big data processing made simple



What is Apache Spark?

- A unified computing engine and a set of libraries for parallel data processing on computer clusters
- It is a Big Data open source engine that supports various programming languages: Python, Java, Scala and R

Some history...

- Apache Spark began at University of California, Berkeley in 2009
- Presented in the famous paper (2010):
  - "Spark: Cluster Computing with Working Sets" by Matei Zaharia, Mosharaf Chowdhury, Michael Franklin, Scott Shenker, and Ion Stoica of the UC Berkeley AMPlab.

Spark: cluster computing with working sets

Authors: Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica Authors Info & Claims

HotCloud'10: Proceedings of the 2nd USENIX conference on Hot topics in cloud computing • June 2010 • Pages 10

Online: 22 June 2010 Publication History

ABSTRACT

MapReduce and its variants have been highly successful in implementing large-scale data-intensive applications on commodity clusters. However, most of these systems are built around an acyclic data flow model that is not suitable for other popular applications. This paper focuses on one such class of applications: those that reuse a working set of data across multiple parallel operations. This includes many iterative machine learning algorithms, as well as interactive data analysis tools. We propose a new framework called Spark that supports these applications while retaining the scalability and fault tolerance of MapReduce. To achieve these goals, Spark introduces an abstraction called resilient distributed datasets (RDDs). An RDD is a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Spark can outperform Hadoop by 10x in iterative machine learning jobs, and can be used to interactively query a 39 GB dataset with sub-second response time.

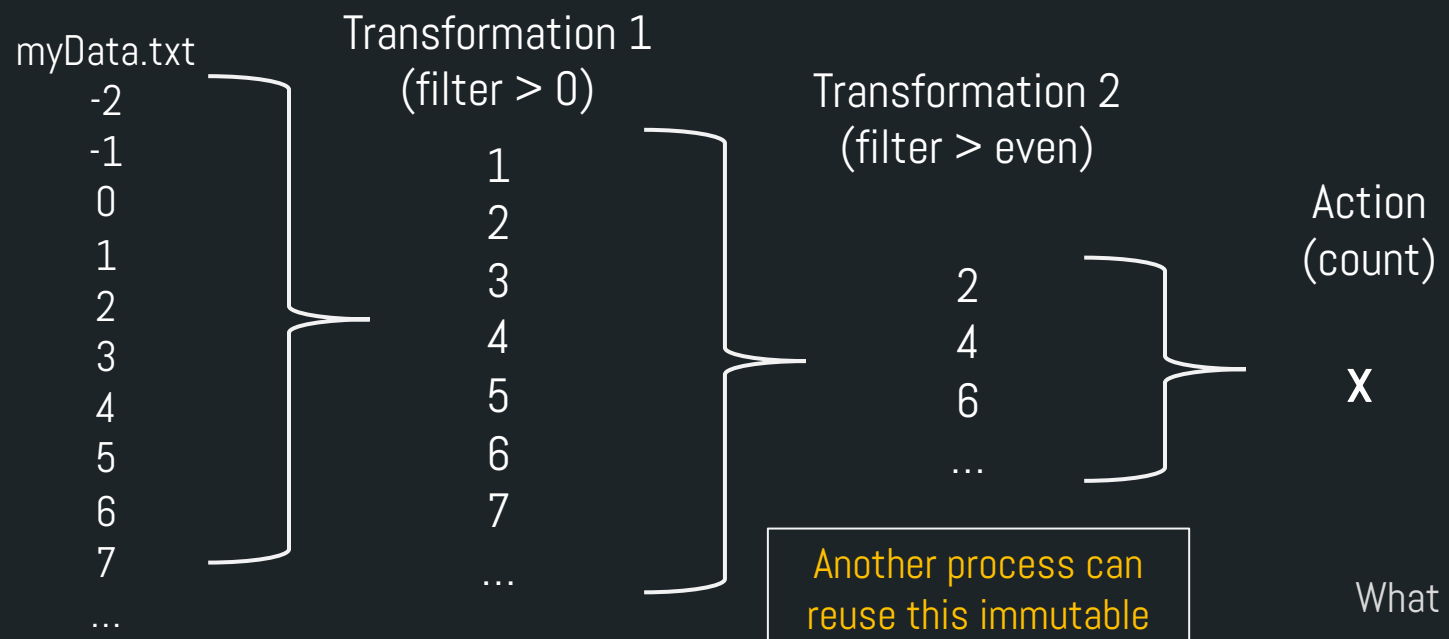
Some history…

- Three important points:
    1. Cluster computing held tremendous potential
    2. The MapReduce engine made it both challenging and inefficient to build large applications
    3. Applications that reuse same data in multiple operations does not perform well on Hadoop

In-memory vs disk-based processing

Considering the following example, how in-memory processing can help us?

Note myData.txt is a 4 GB file.

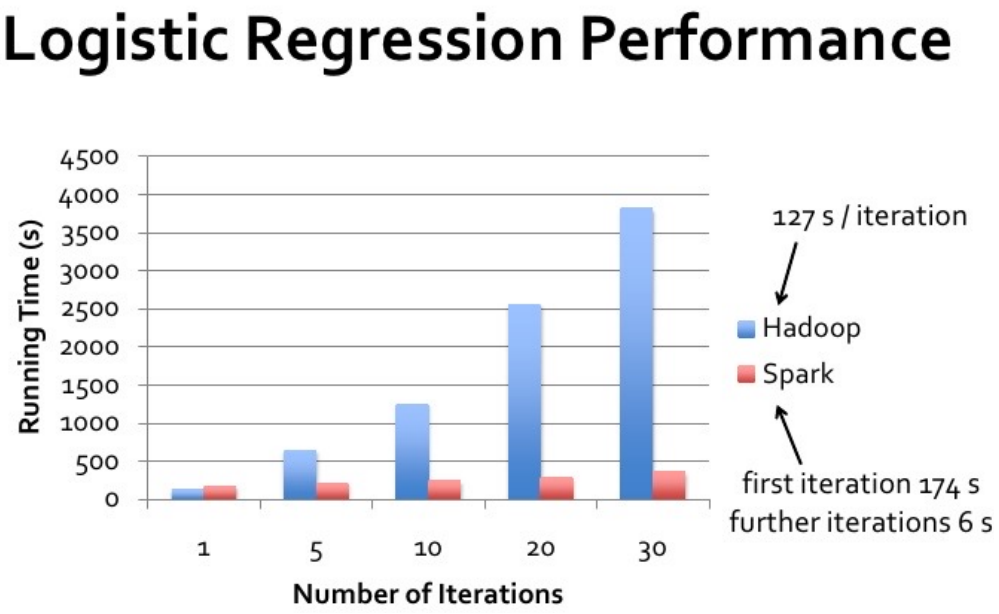


What happens if there is no available memory?

Data saved to disk...

What is the problem with Hadoop MapReduce?

- The typical machine-learning algorithm might need to make 10 or 20 passes over the data
- In MapReduce, each pass had to be written as a separate MapReduce job
- Which had to be launched separately on the cluster and load the data from scratch.
- **This is not a problem it is a different use case!**



MapReduce is disk-based while Apache **Spark** uses memory and disk for processing.

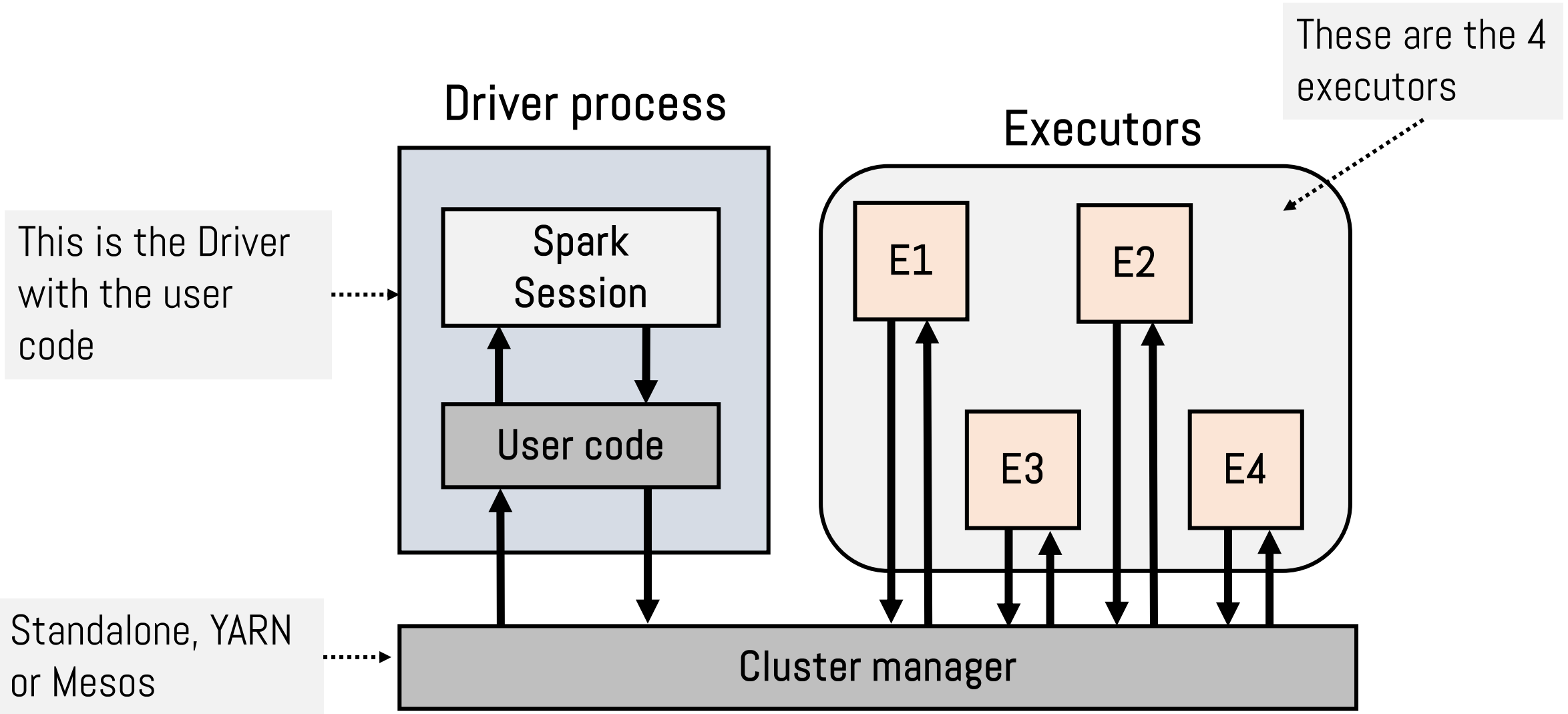
Spark’s basic architecture

- Single machines do not have enough power and resources to perform computations on huge amounts of information
- **Spark manages the execution of tasks on data across a cluster of computers.**
- We submit Spark Applications to a cluster manager, which will grant resources to our application to complete our work.

Spark applications

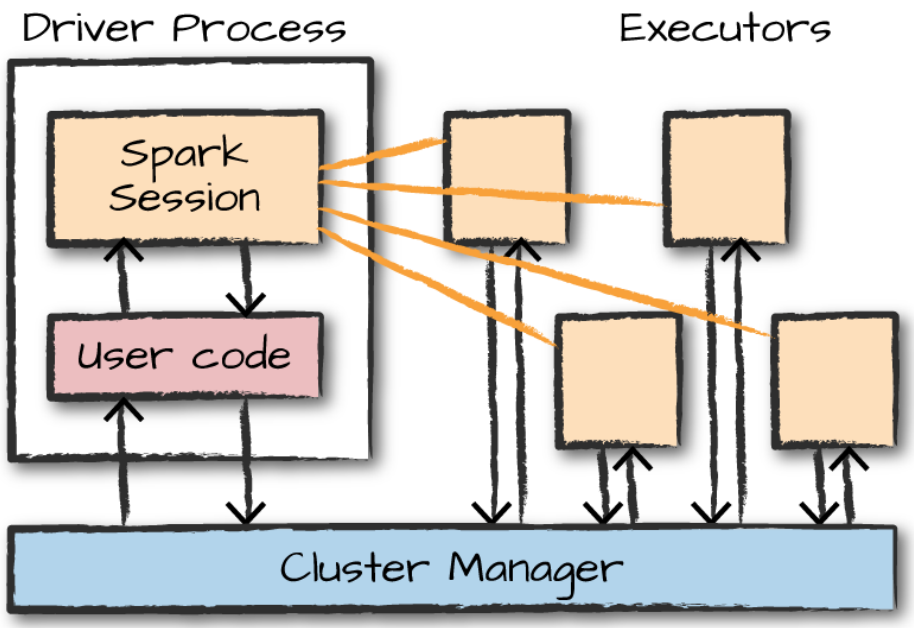
- Spark Applications consist of a **driver** process and a set of **executor** processes.
- The **driver** process sits on a node in the cluster and is responsible for three things:
  - Maintaining information about the Spark Application
  - Responding to a user's program or input
  - Analyse, distribute, and schedule work across the executors
- The **executors** are responsible for carrying out the work that the driver assigns them, including:
  - Executing code assigned to it by the driver
  - Reporting the state of the computation on that executor back to the driver node

Spark Applications cont.



How to run a Big Data job on Spark?

- To send user commands and data to the Spark cluster we create a **SparkSession**
  - It is the way Spark executes user-defined manipulations across the cluster
  - The way to interact with various spark's functionality with a lesser number of constructs.



Spark uses, dataframes that can be partitioned...

- Spark breaks up the data into chunks called partitions
  - An executor performs work in parallel on partitions
- If you have one partition, Spark will have a parallelism of only one, even if you have thousands of executors
- If you have many partitions but only one executor, Spark will still have a parallelism of only one because there is only one computation resource
- With DataFrames, you do not manipulate partitions manually or individually
  - Spark determines how this work will execute on the cluster.

We have RDDs, Dataframes and Datasets...

- An **RDD** is a black box of data (cannot be optimized)
  - Yet, you can go from a Dataframe to an RDD via its **rdd** method, and you can go from an RDD to a Dataframe (if the RDD is in a tabular format) via the **toDF** method
- A **Dataframe** is a table, or a two-dimensional array-like structure
  - A Dataframe has additional metadata so allows Spark to run optimizations on queries
- **DataSets**:
  - It allows to perform an operation on serialized data and improves memory usage
- **It is recommended to use a DataFrame where possible due to the built-in query optimization**

Transformations, is about transforming data...

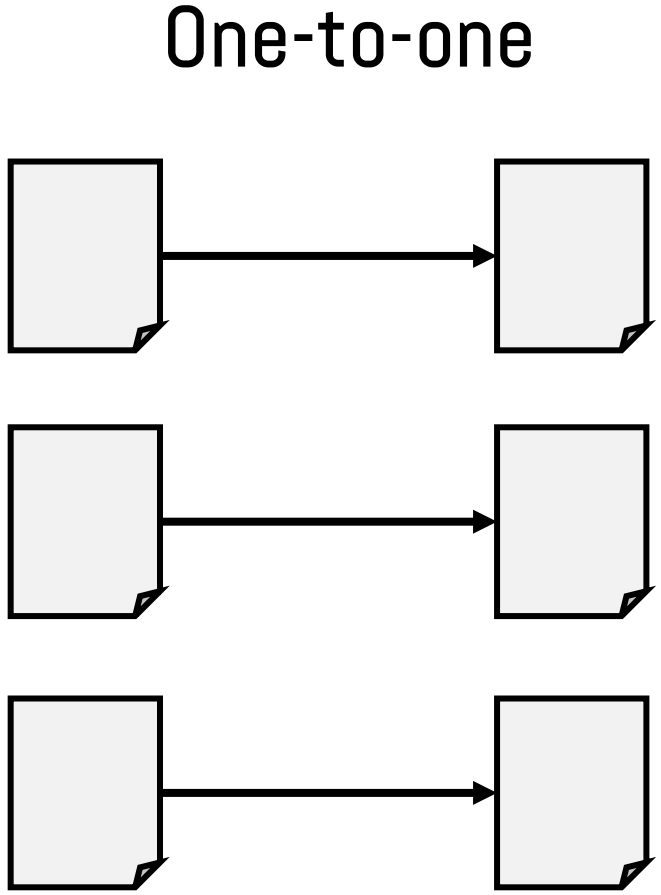
- In Spark, the core data structures are immutable
  - Cannot be changed after they're created.
- But... if you cannot change it, how are you supposed to use it?
  - To "change" a DataFrame, you need to instruct Spark how you would like to modify it to do what you want.
- **These instructions are called transformations**

Spark will act only when we ask for it…

- We specified a transformation, and Spark will not act until we call an action
- There are two types of transformations that:
  - Specify Narrow dependencies (also known as narrow transformations)
  - Specify Wide dependencies (also known as wide transformations)

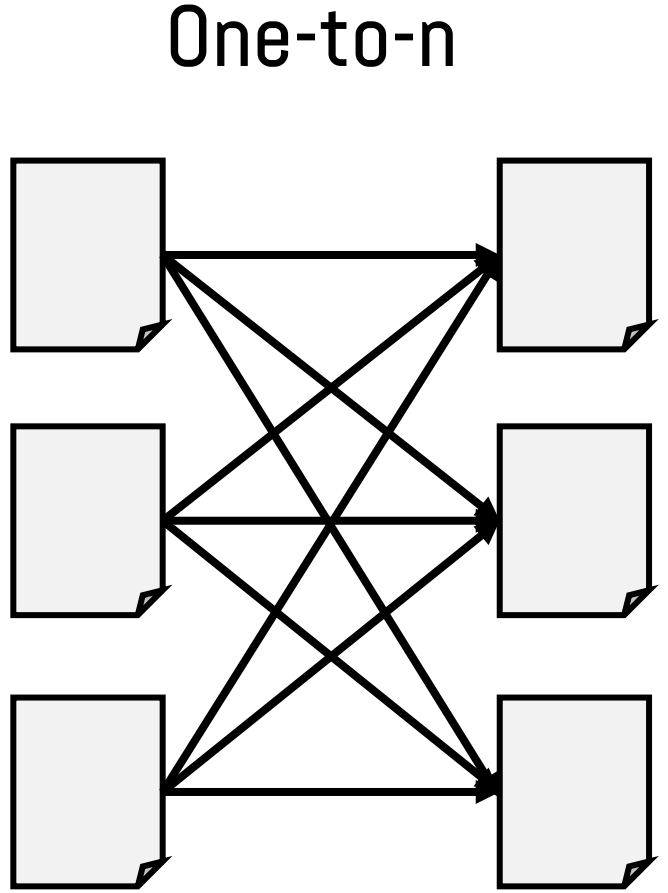
Narrow transformations

- Narrow transformations are the result of map(), filter()
- Only one partition contributes to at most one output partition
- All the elements that are required to compute the records in single partition live in the single partition

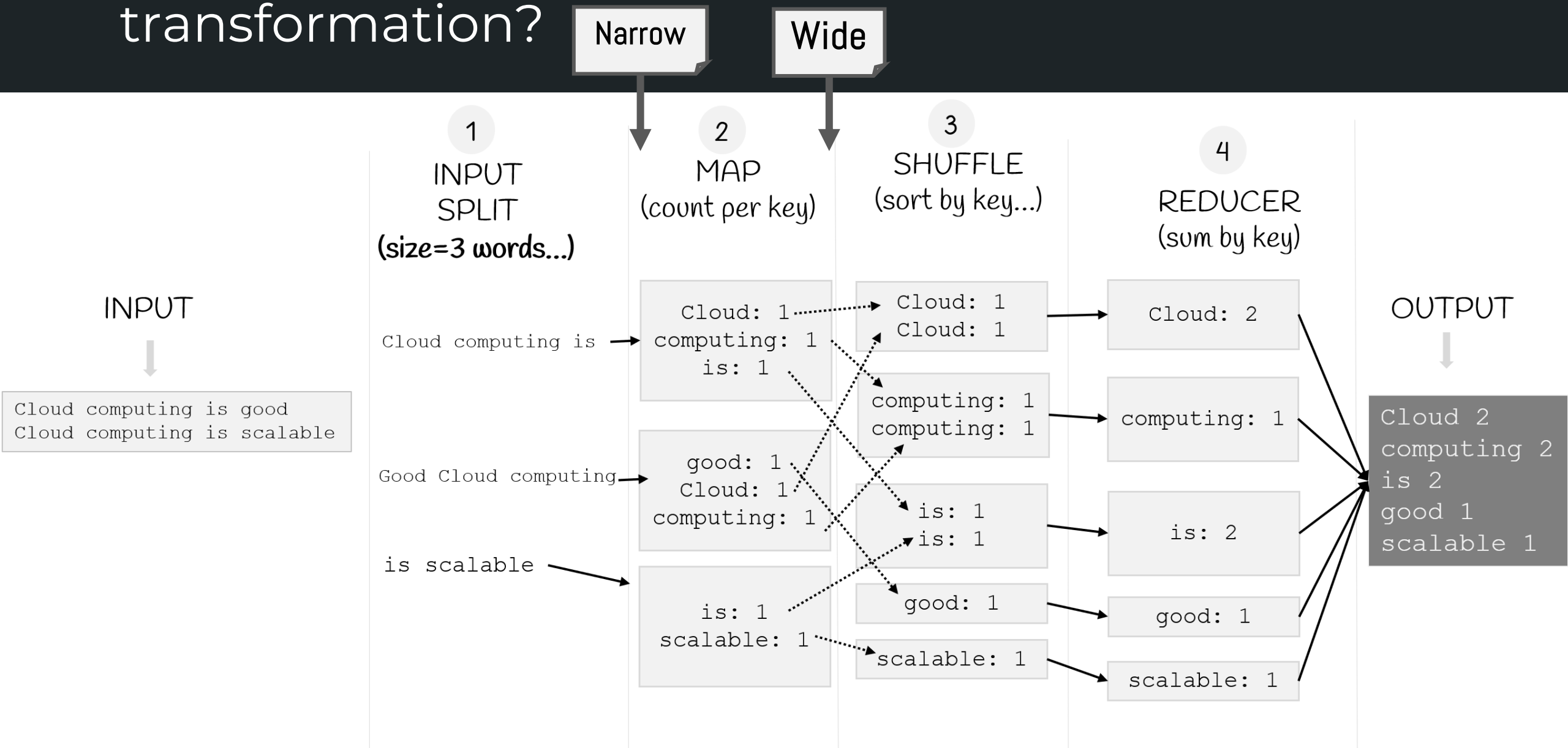


Wide transformation

- A wide dependency transformation (or wide transformation) will have input partitions contributing to many output partitions
  - You will often hear this referred to as a **shuffle**
    - Wide transformations are the result of **groupbyKey** and **reducebyKey**



Can you identify a Narrow and a Wide transformation?



Time to act...(Lazy evaluation)

- In Spark, instead of modifying the data immediately, you build up a plan of transformations that you would like to apply to your source data.
- Spark will wait until the very last moment to execute the computation instructions.

Let’s run it: Actions

- Transformations allow us to build up our logical transformation plan.
  - To trigger the computation, we run an action.
- An action instructs Spark to compute a result from a series of transformations.
- Count example:
  - Count the total number of records in the DataFrame

An End-to-End Example

- Use Spark to analyse flight data from the United States Bureau of Transportation statistics
  - What we have is a set of CSV files...
- Each file has a number of rows within it.

```
$ head /data/flight-data/csv/2015-summary.csv

DEST_COUNTRY_NAME,ORIGIN_COUNTRY_NAME,count
United States,Romania,15
United States,Croatia,1
United States,Ireland,344
```

- What are the top two countries that flight less to United States?
  - How to approach this query from the perspective of map/reduce or wide/narrow transformations?

How to read data with Spark?

- To read the data, we will use a **DataFrameReader** that is associated with our **SparkSession**.
- In our case, we want a **schema inference**:
  - We instruct Spark to take **a best guess** at what the schema of our DataFrame should be.
- We also want to specify that the first row is the header in the file...

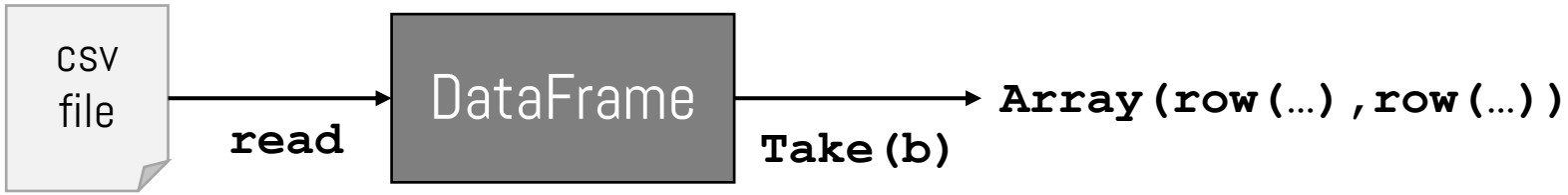
Schema inference

- To get the schema information, Spark reads in a little bit of the data
- Then attempts to parse **the types** in those rows according to the types available in Spark.

```
# in Python
flightData2015 = spark\
  .read\
  .option("inferSchema", "true")\
  .option("header", "true")\
  .csv("/data/flight-data/csv/2015-summary.csv")
```

Reading data is a lazy operation…

- Each of these DataFrames (in Scala or Python) have a set of columns with an unspecified number of rows.
- Spark peeked at only a couple of rows of data to try to guess what types each column should be.
  - The CSV file being read into a DataFrame and then being converted into a local array or list of rows.



Sorting data

- Let’s sort our data according to the count column (that is an integer)
- Nothing happens to the data when we call sort because it’s just a transformation
  - Remember, sort is not an action

Spark is building up a plan…

- Spark is building up a plan for how it will execute this across the cluster

```
flightData2015.sort("count").explain()

== Physical Plan ==
*Sort [count#195 ASC NULLS FIRST], true, 0
+- Exchange rangepartitioning(count#195 ASC NULLS FIRST, 200)
   +- *FileScan csv [DEST_COUNTRY_NAME#193,ORIGIN_COUNTRY_NAME#194,count#195] ...
```

Apply an action

- We can specify an action to kick off this plan!

```
flightData2015.sort("count").take(2)

... Array([United States,Singapore,1], [Moldova,United States,1])
```

- Here, you can find a list of actions you can use for your analytics, such as:
  - Count
  - Reduce
  - ...

Spark and SQL

- We can query our data in SQL
- We can use the **spark.sql** function that conveniently returns a new DataFrame
  - SQL query against a DataFrame returns another DataFrame, this is actually quite powerful

```
# in Python
sqlWay = spark.sql("""
SELECT DEST_COUNTRY_NAME, count(1)
FROM flight_data_2015
GROUP BY DEST_COUNTRY_NAME
""")

dataFrameWay = flightData2015\
  .groupBy("DEST_COUNTRY_NAME")\
  .count()

sqlWay.explain()
dataFrameWay.explain()
```

```
== Physical Plan ==
*HashAggregate(keys=[DEST_COUNTRY_NAME#182], functions=[count(1)])
+- Exchange hashpartitioning(DEST_COUNTRY_NAME#182, 5)
   +- *HashAggregate(keys=[DEST_COUNTRY_NAME#182], functions=[partial_count(1)])
      +- *FileScan csv [DEST_COUNTRY_NAME#182] ...
```

What about complex queries?

- Let's perform a more complicated query
  - Find the top five destination countries in the data
- This is a multi-transformation query

Spark SQL

```
maxSql = spark.sql("""
SELECT DEST_COUNTRY_NAME, sum(count) as destination_total
FROM flight_data_2015
GROUP BY DEST_COUNTRY_NAME
ORDER BY sum(count) DESC
LIMIT 5
""")
```

Spark DF

```
from pyspark.sql.functions import desc

flightData2015\
  .groupBy("DEST_COUNTRY_NAME")\
  .sum("count")\
  .withColumnRenamed("sum(count)", "destination_total")\
  .sort(desc("destination_total"))\
  .limit(5)\
  .show()
```

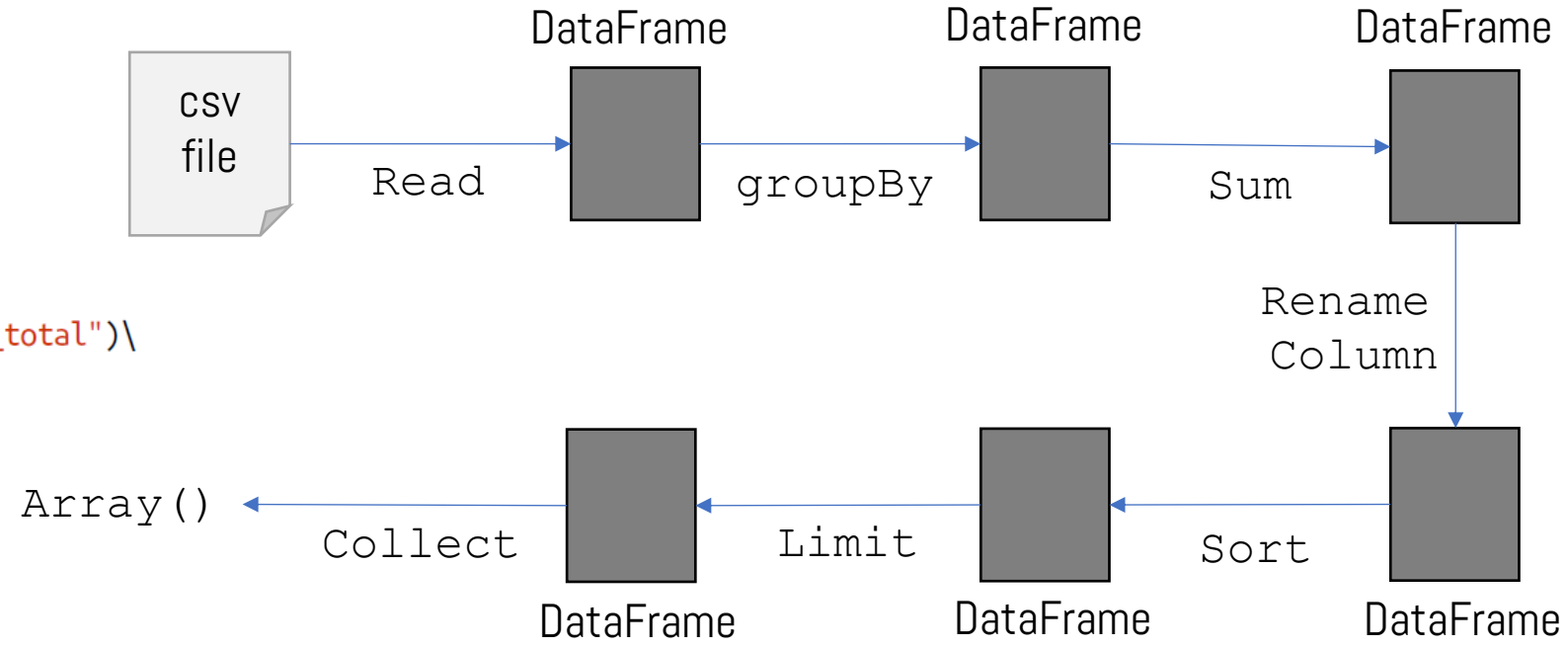
```
+-----------------+-----------------+
|DEST_COUNTRY_NAME|destination_total|
+-----------------+-----------------+
|    United States|           411352|
|           Canada|             8399|
|           Mexico|             7140|
|   United Kingdom|             2025|
|            Japan|             1548|
+-----------------+-----------------+
```

There are **seven** steps in our example

- This execution plan is a **directed acyclic graph (DAG)** of transformations, each resulting in a new immutable DataFrame, on which we call an action to generate a result.

```
# in Python
from pyspark.sql.functions import desc

flightData2015\
  .groupBy("DEST_COUNTRY_NAME")\
  .sum("count")\
  .withColumnRenamed("sum(count)", "destination_total")\
  .sort(desc("destination_total"))\
  .limit(5)\
  .show()
```



Let us summarise

- Transformations applied on RDDs to access, modify and filter RDD to generate a new one
- We build up a lazy evaluation plan…
- We complete our query with an action
  - Count, collect, first, take

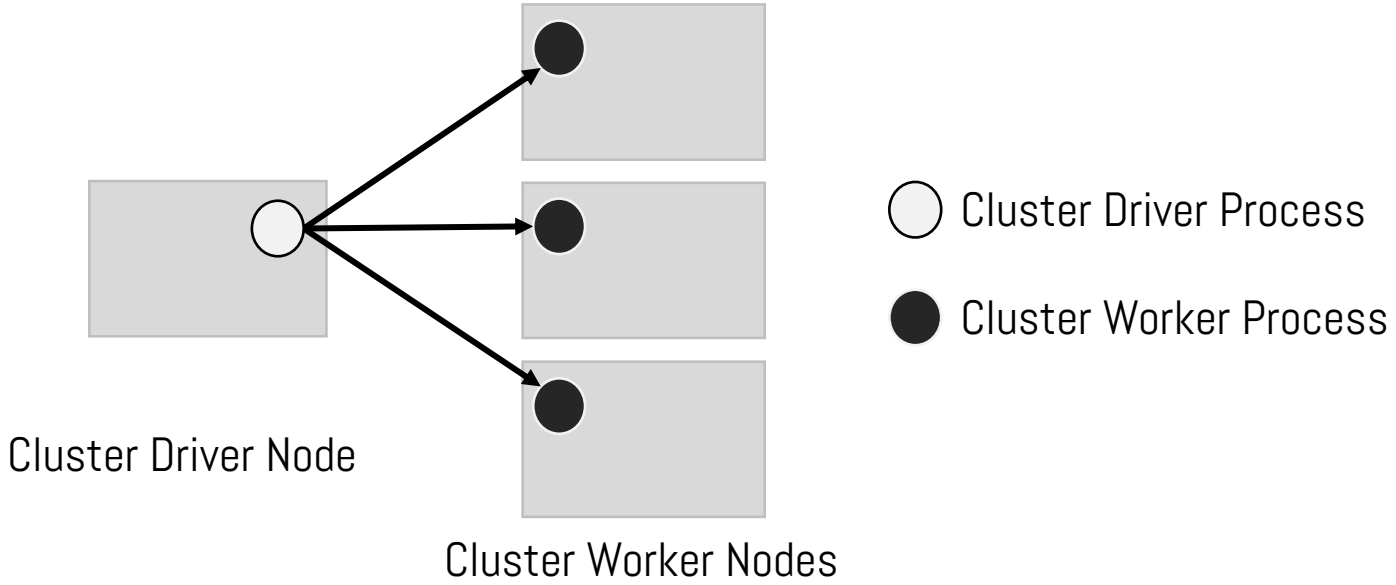
Spark, under the hood…

Spark, under the hood

- The Architecture of a Spark Application:
  - The Spark driver:
    - It is the controller of the execution of a Spark Application and maintains all of the state of the Spark cluster.
    - It communicates with the cluster manager in order to actually get physical resources and launch executors.
  - The Spark executors:
    - Processes that perform the tasks assigned by the Spark driver.
  - The cluster manager (for example YARN):
    - The cluster manager is responsible for maintaining a cluster of machines that will run your Spark Application(s).

Let us see the details...

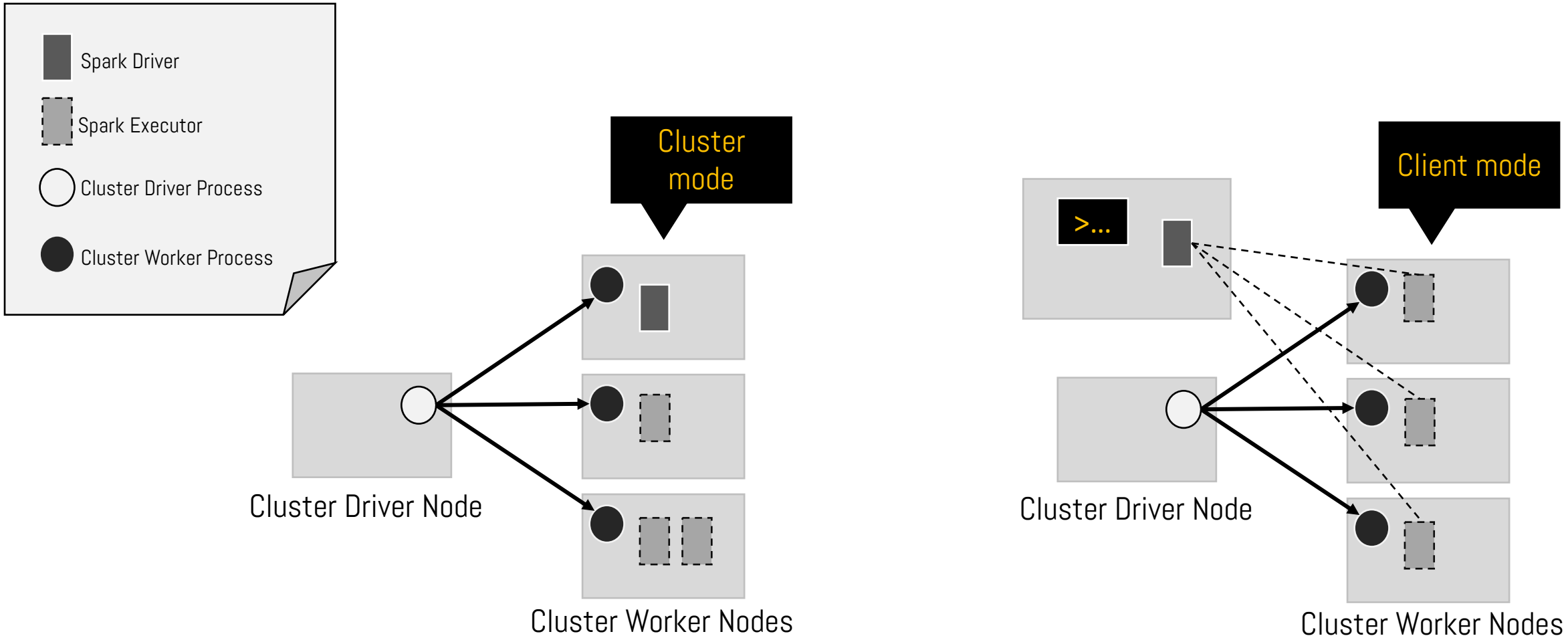
- A cluster manager will have its own “driver” (the master)
  - Drivers and workers are tied to physical machines (rather than processes)
- The circles represent daemon processes running on nodes
  - These are just the processes from the cluster manager



Three Spark running modes...

- Local mode:
  - Runs the entire Spark Application on a single machine e.g. your laptop.
- Cluster mode:
  - The most common way of running Spark Applications.
  - The **cluster manager launches the driver process on a worker node** inside the cluster, in addition to the executor processes.
- Client mode:
  - The same as cluster mode except that the Spark driver remains on the client machine that submitted the application.

Cluster vs Client mode



The Life Cycle of a Spark Application

- We assume that a cluster is already running with four nodes, a driver (a cluster manager driver) and three worker nodes.
- There are four steps to follow:

  Step 1: Client request

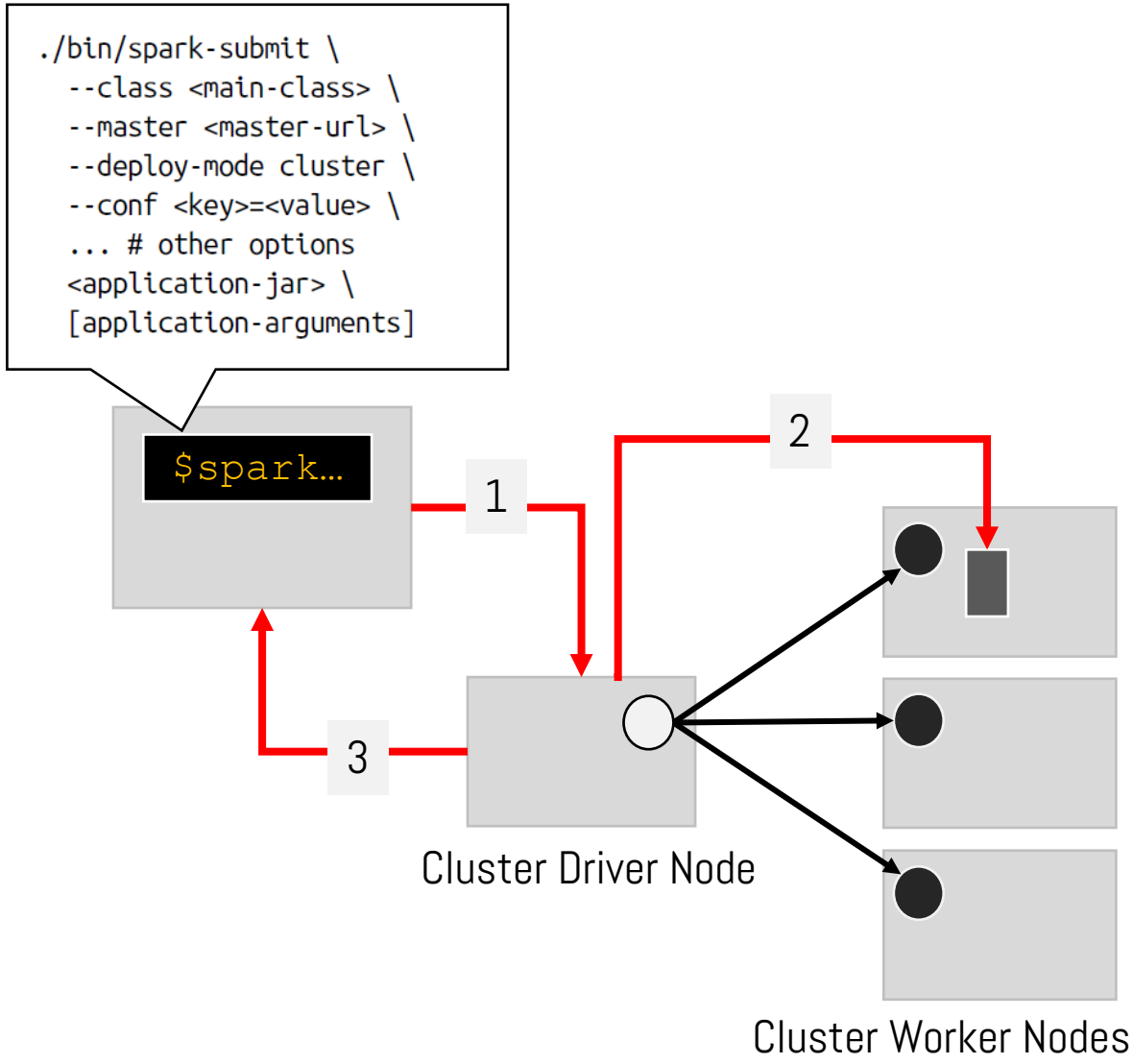
  Step 2: Launch

  Step 3: Execution

  Step 4: Completion

Step 1: Client request

- A request to submit an application.
- At this point, you are executing code on your local machine and you're going to make a request to the cluster manager driver node.
- Here, we are explicitly asking for resources for the Spark driver process only.
- We assume that the cluster manager accepts this offer and places the driver onto a node in the cluster



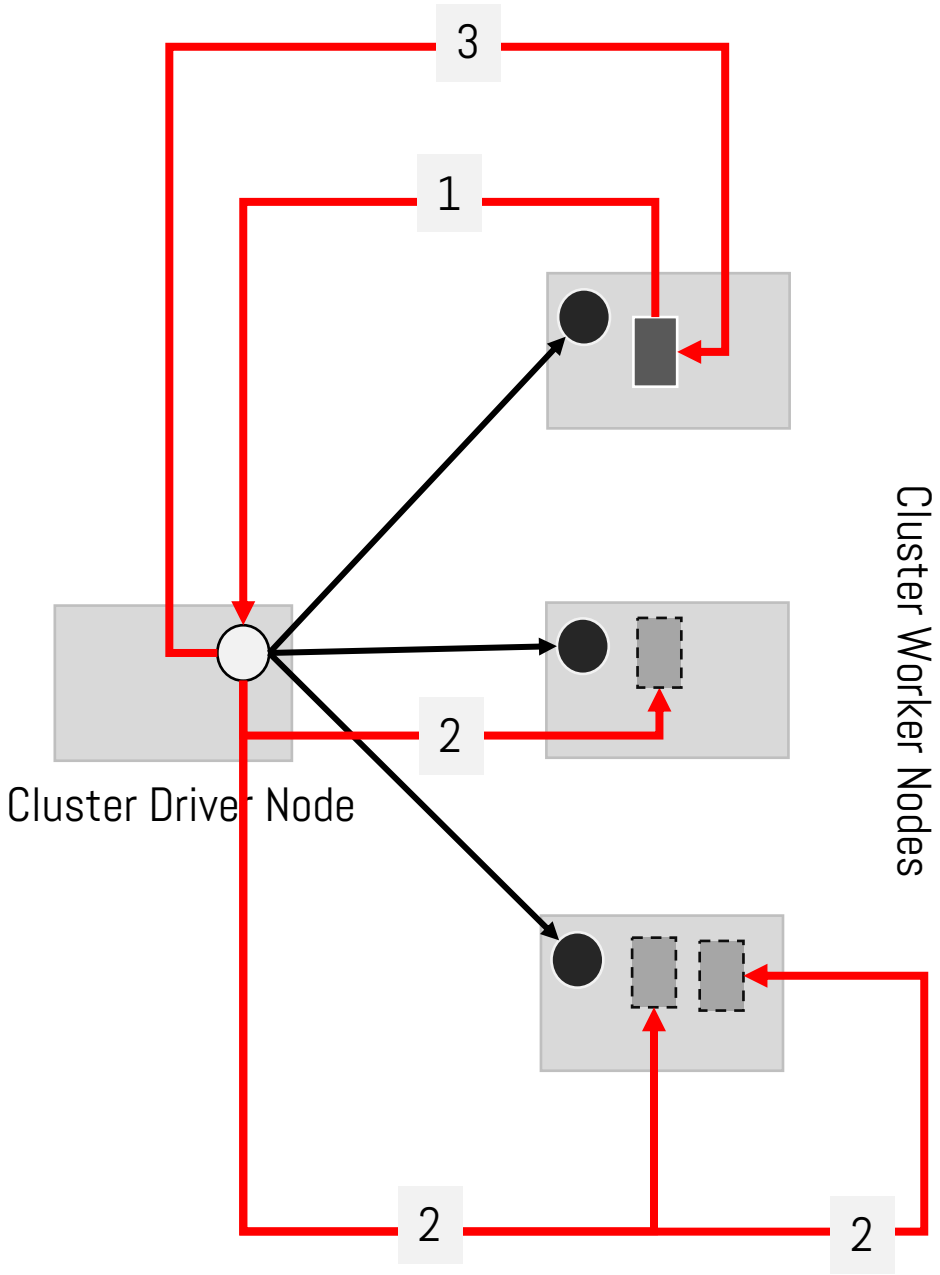
1: Submit the job to the cluster driver

2: Access the Spark driver

3: The app is running!

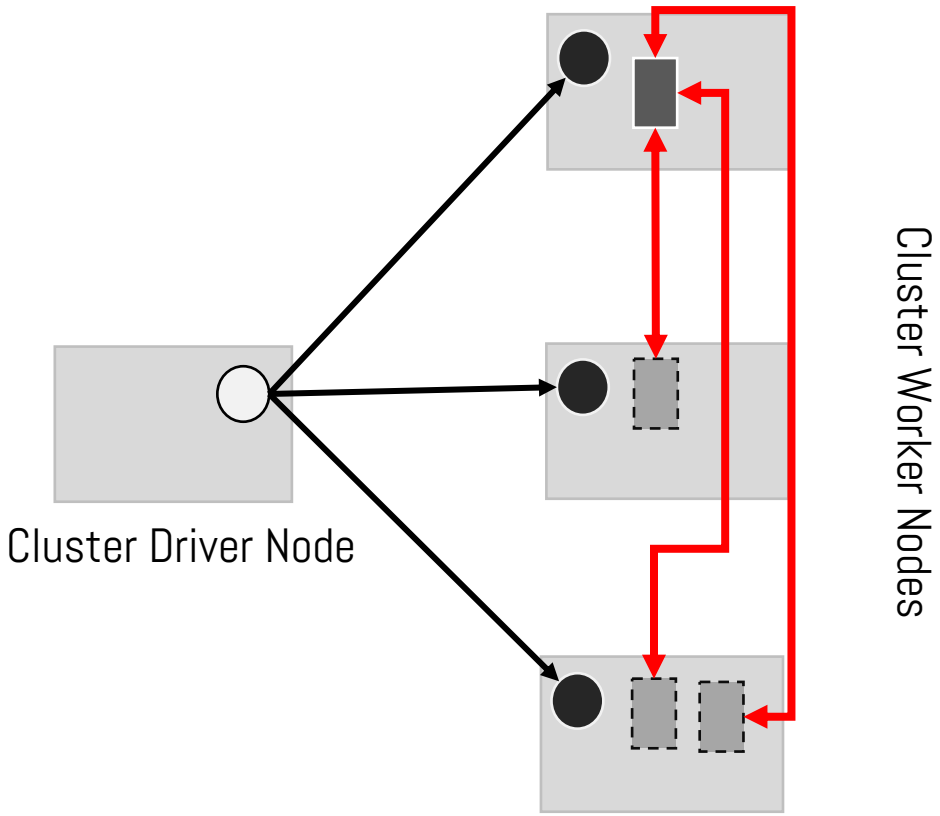
Step 2: Launch

- Now that the driver process has been placed on the cluster, it begins running user code
- This code must include a SparkSession that initializes a Spark cluster
  - For example: Driver + executors
- The SparkSession will subsequently communicate with the cluster manager (1), asking it to launch Spark executor processes across the cluster (2).
- The cluster manager launches the executor processes and informs the driver process about their location (3).



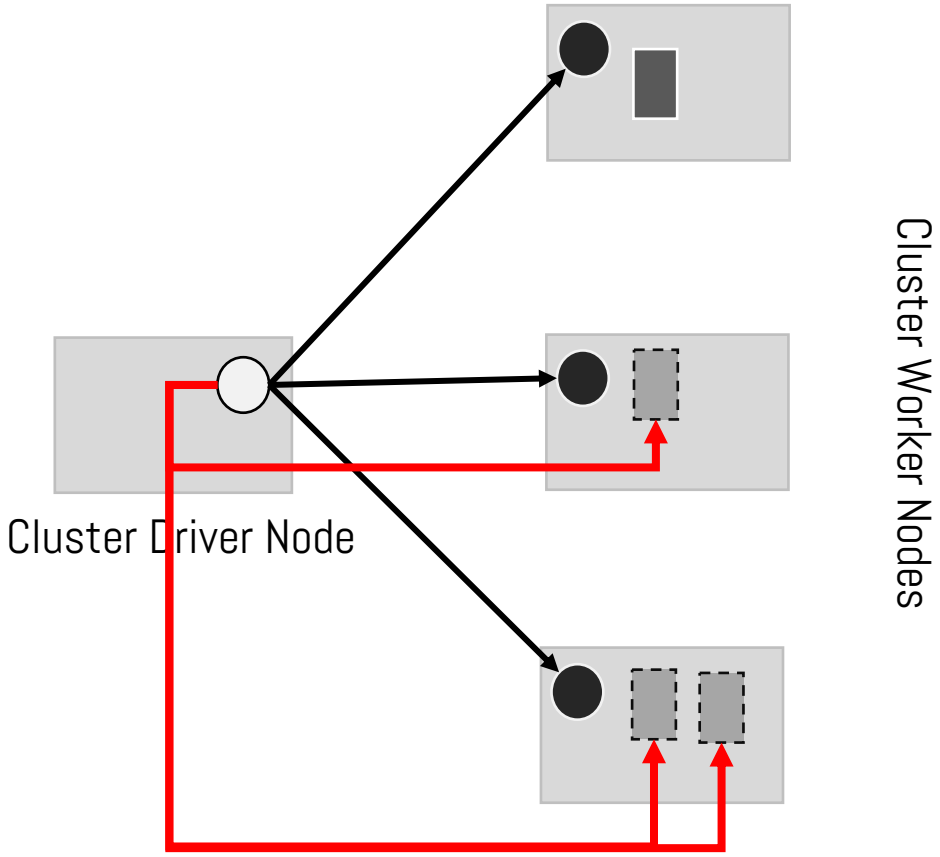
Step 3: Execution

- Spark goes and executes the code
- The driver and the workers communicate among themselves, executing code and moving data around.
- **The driver schedules the tasks in each worker, and each worker responds with the status of those tasks and success or failure.**

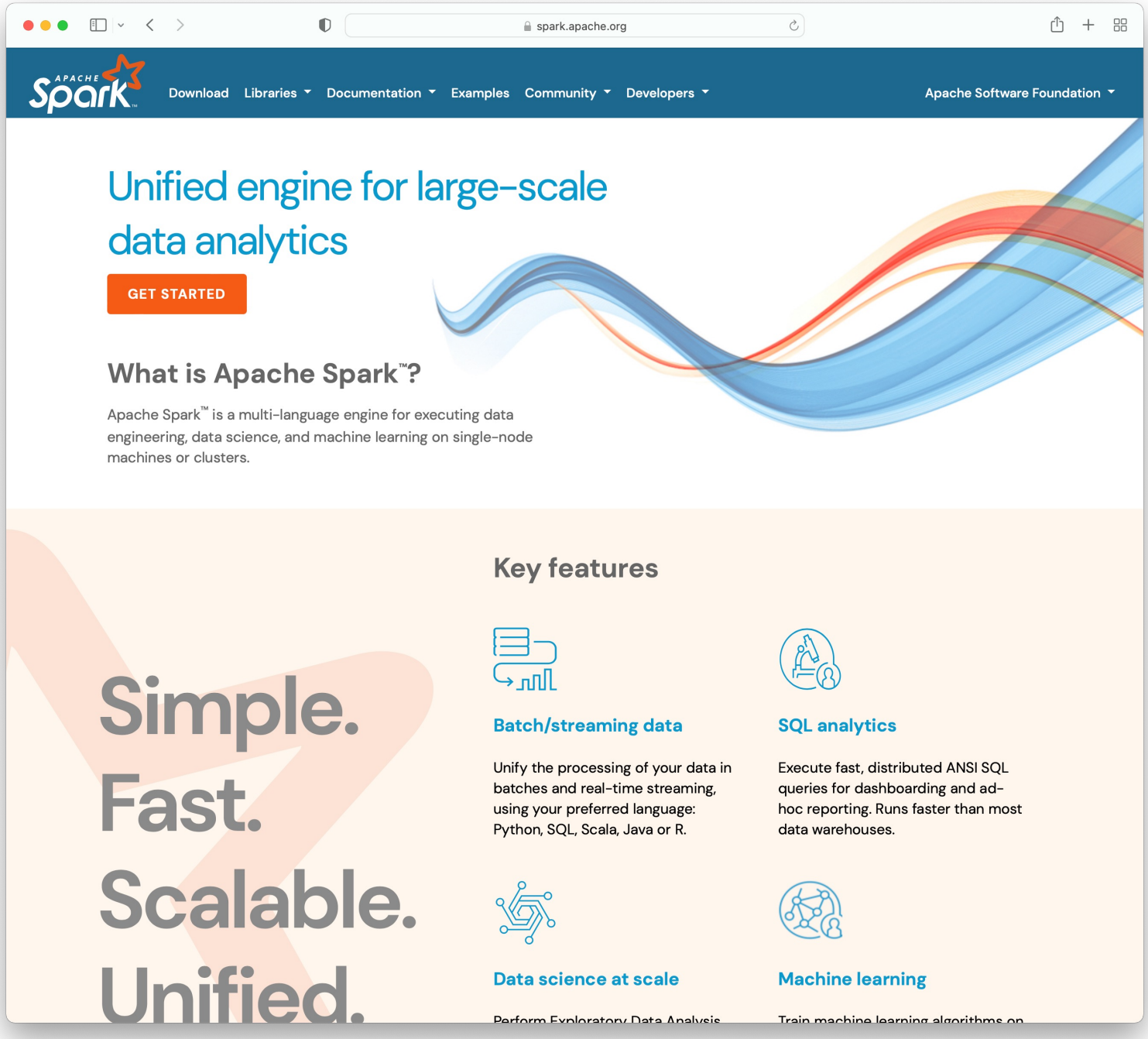


Step 4: Completion

- After a Spark Application completes, the driver process exits with either success or failure.
- The cluster manager then shuts down the executors in that Spark cluster for the driver.
- At this point, you can see the success or failure of the Spark Application by asking the cluster manager for this information.



How to start?


spark.apache.org
Download
Libraries
Documentation
Examples
Community
Developers
Apache Software Foundation
Unified engine for large-scale data analytics
GET STARTED
What is Apache Spark™?
Apache Spark™ is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters.
Simple.
Fast.
Scalable.
Unified.
Key features
Batch/streaming data
Unify the processing of your data in batches and real-time streaming, using your preferred language: Python, SQL, Scala, Java or R.
SQL analytics
Execute fast, distributed ANSI SQL queries for dashboarding and ad-hoc reporting. Runs faster than most data warehouses.
Data science at scale
Machine learning

That’s all for our module!

- **Thank you!**
- Quote of the day:

  "Why, sometimes I've believed as many as six impossible things before breakfast."

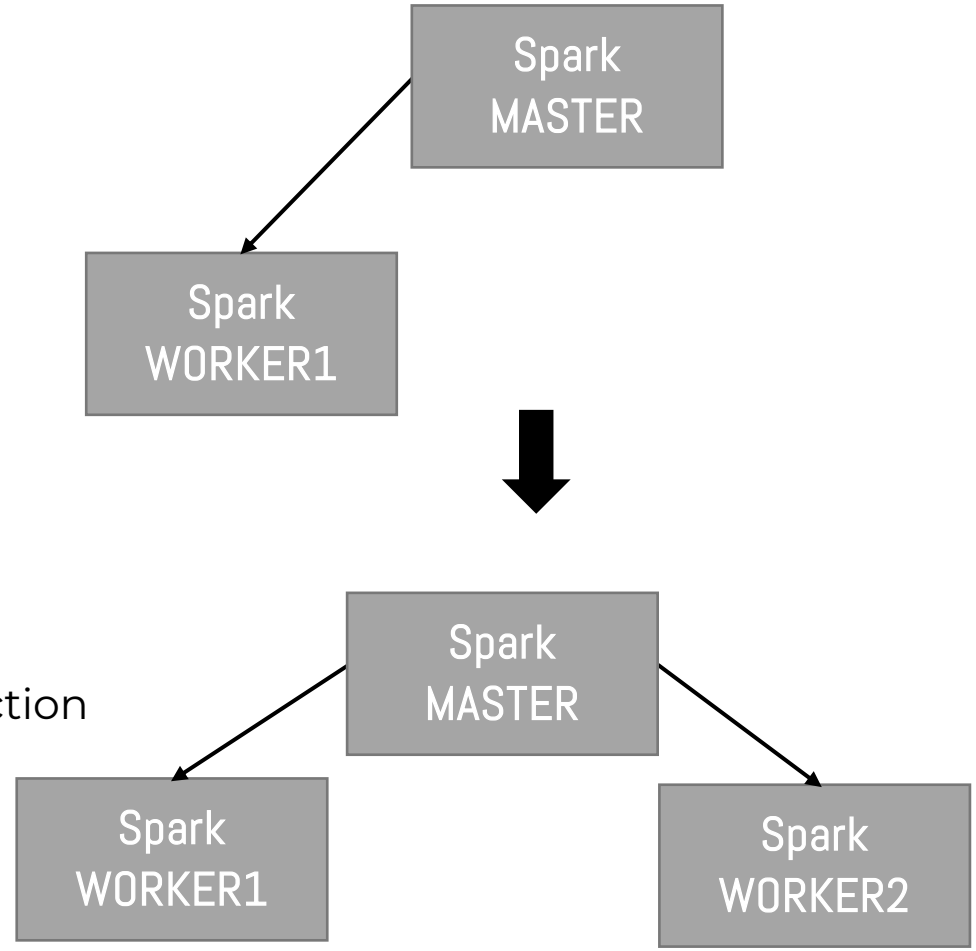
  Alice in Wonderland, Lewis Carroll

- Lab starts soon!

Lab 9: Running Apache Spark on Docker

What we will run today?

- We will deploy a cluster of two nodes (Master-Worker)
  - Driver and executor
- We will deploy a three nodes cluster
  - Driver and two executors
- We will run Spark programs
  - A word count on Spark
  - A Spark program with two transformations and an action
  - A Spark program to calculate the pi
  - A Spark SQL example



Take home

- What is Spark?
- What is a data frame?
- What is an RDD?
- What is a transformation?
- What is an action?
- How does Spark work?
- How can I run a simple Spark job?