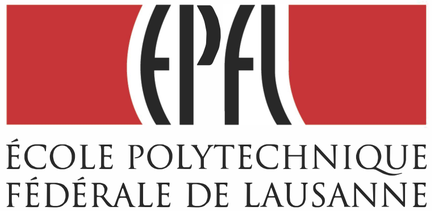


Cluster Toplogy Matters!

Big Data Analysis with Scala and Spark

Heather Miller

Example 1: A Simple `println`

Let's start with an example. Assume we have an RDD populated with `Person` objects:

```
case class Person(name: String, age: Int)
```

What does the following code snippet do?

```
val people: RDD[Person] = ...
people.foreach(println)
```

What happens?

Example 2: A Simple `take`

What about here? Assume we have an RDD populated with the same definition of `Person` objects:

```
case class Person(name: String, age: Int)
```

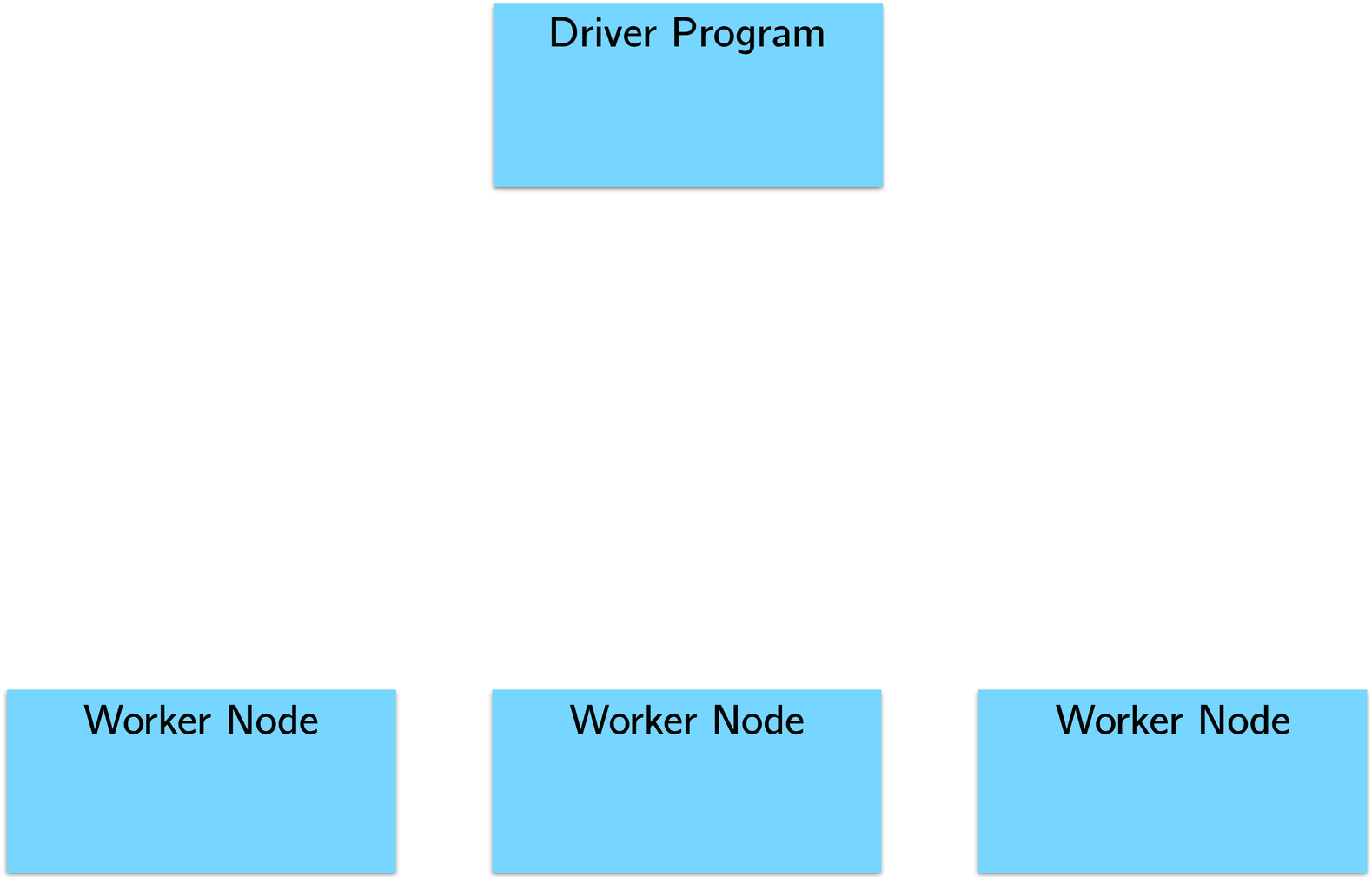
What does the following code snippet do?

```
val people: RDD[Person] = ...
val first10 = people.take(10)
```

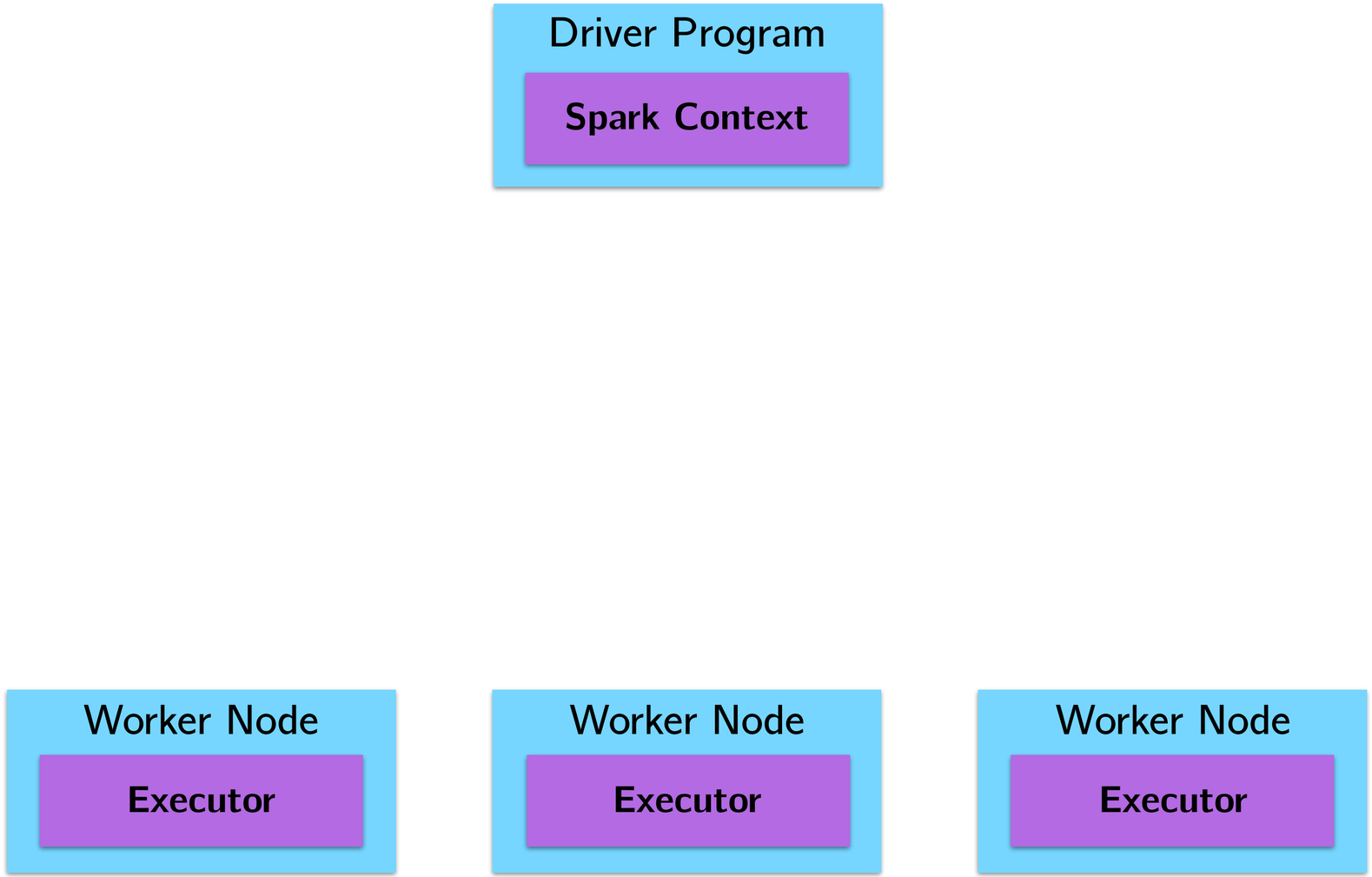
Where will the `Array[Person]` representing `first10` end up?

How Spark Jobs are Executed

How Spark Jobs are Executed

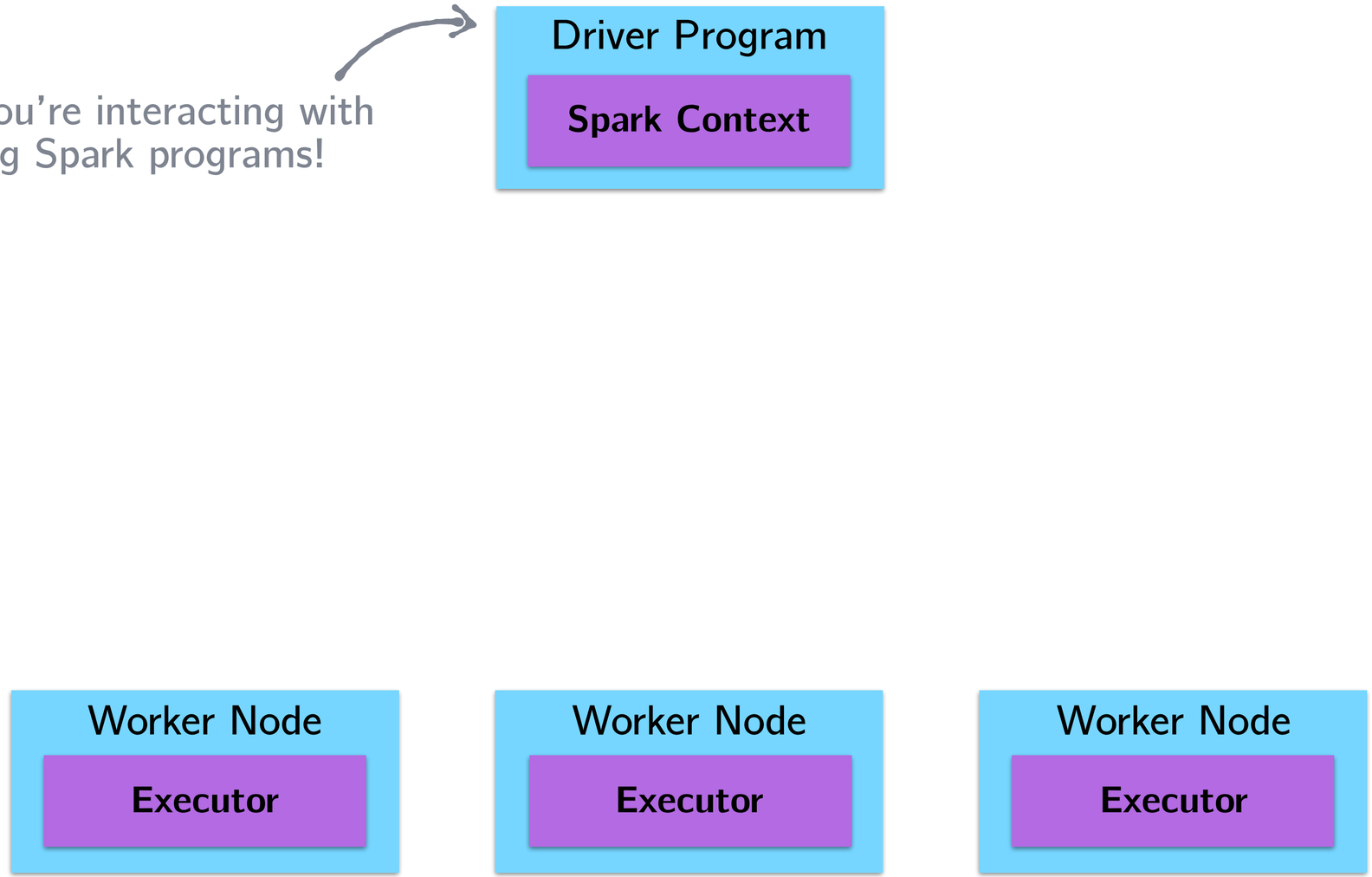


How Spark Jobs are Executed



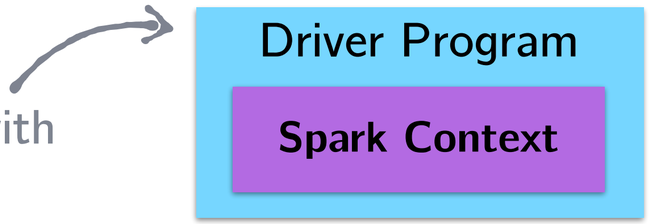
How Spark Jobs are Executed

This is the node you're interacting with when you're writing Spark programs!

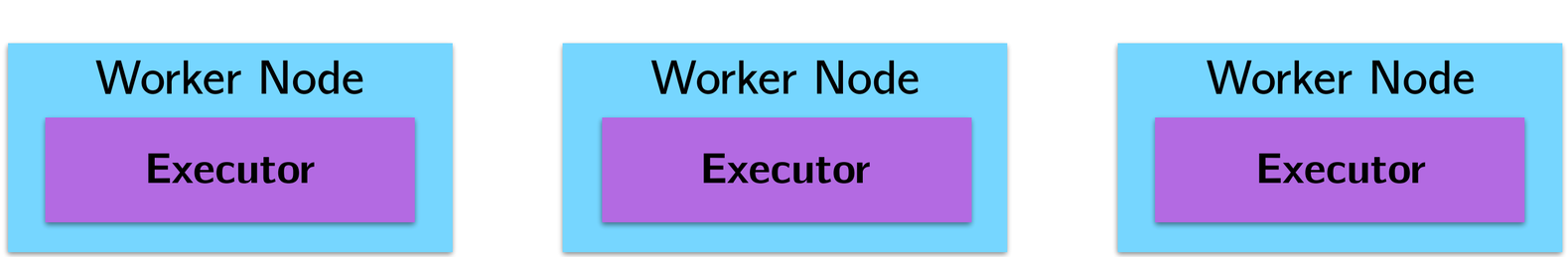


How Spark Jobs are Executed

This is the node you're interacting with when you're writing Spark programs!

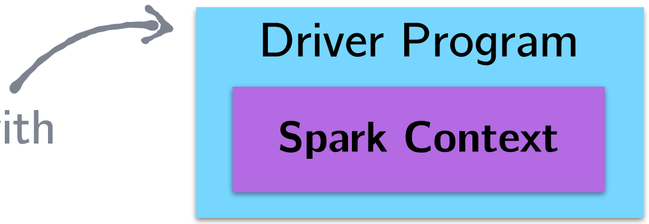


These are the nodes actually executing the jobs!



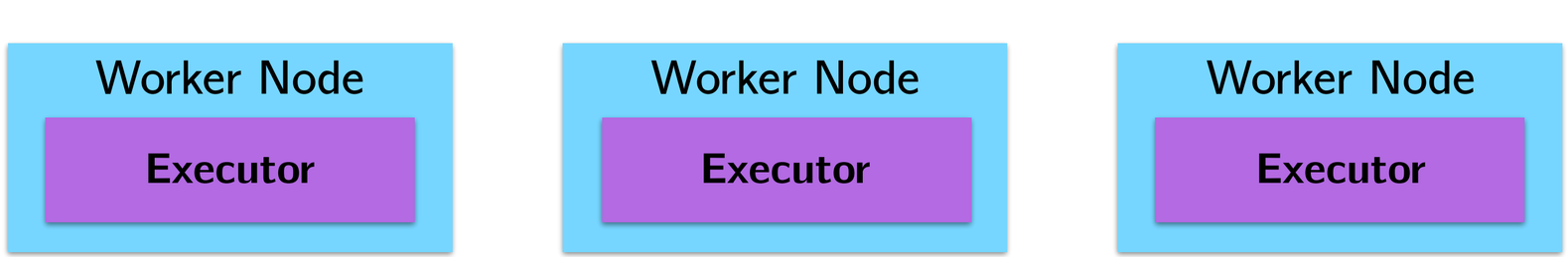
How Spark Jobs are Executed

This is the node you're interacting with when you're writing Spark programs!

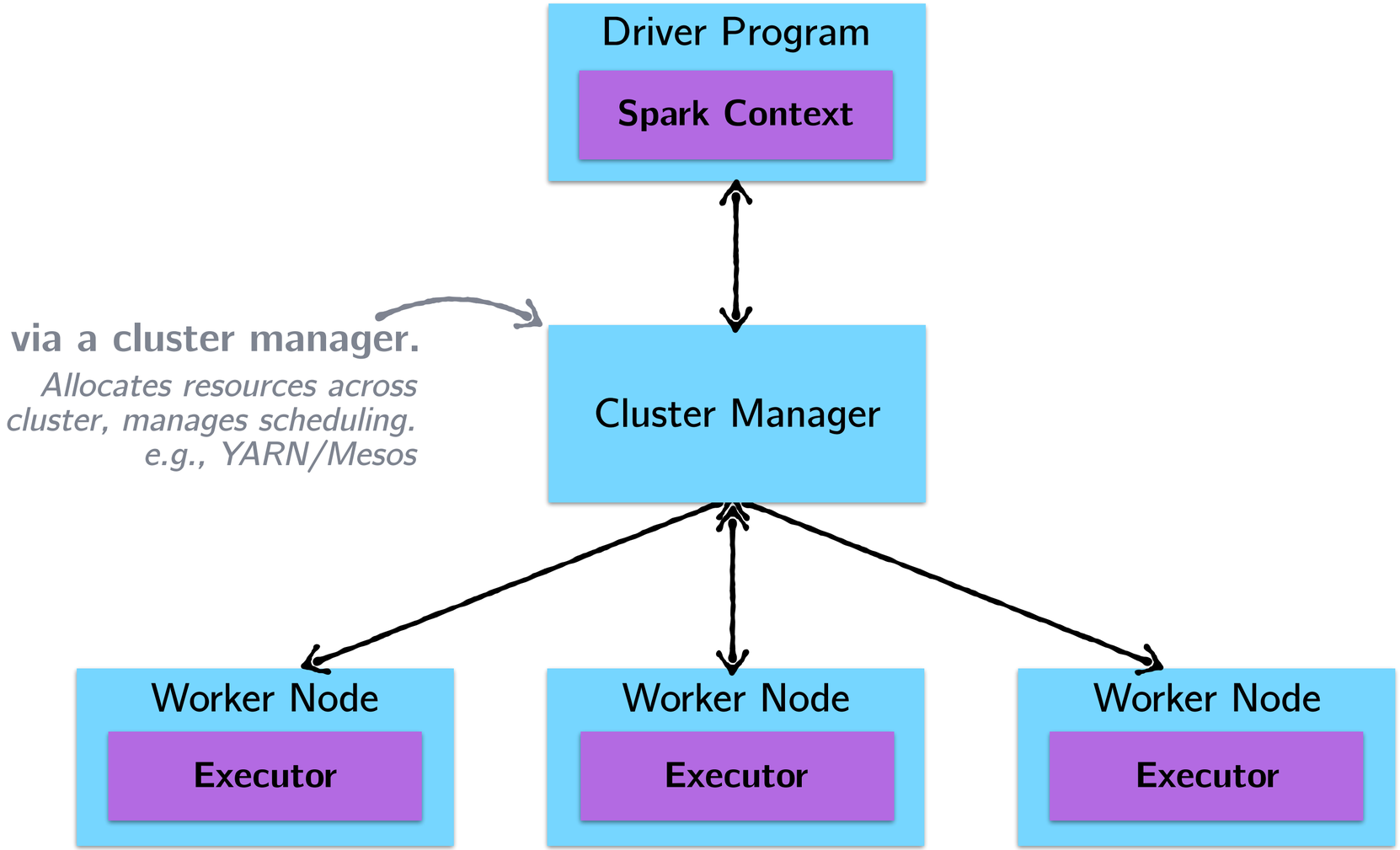


But how do they all communicate?

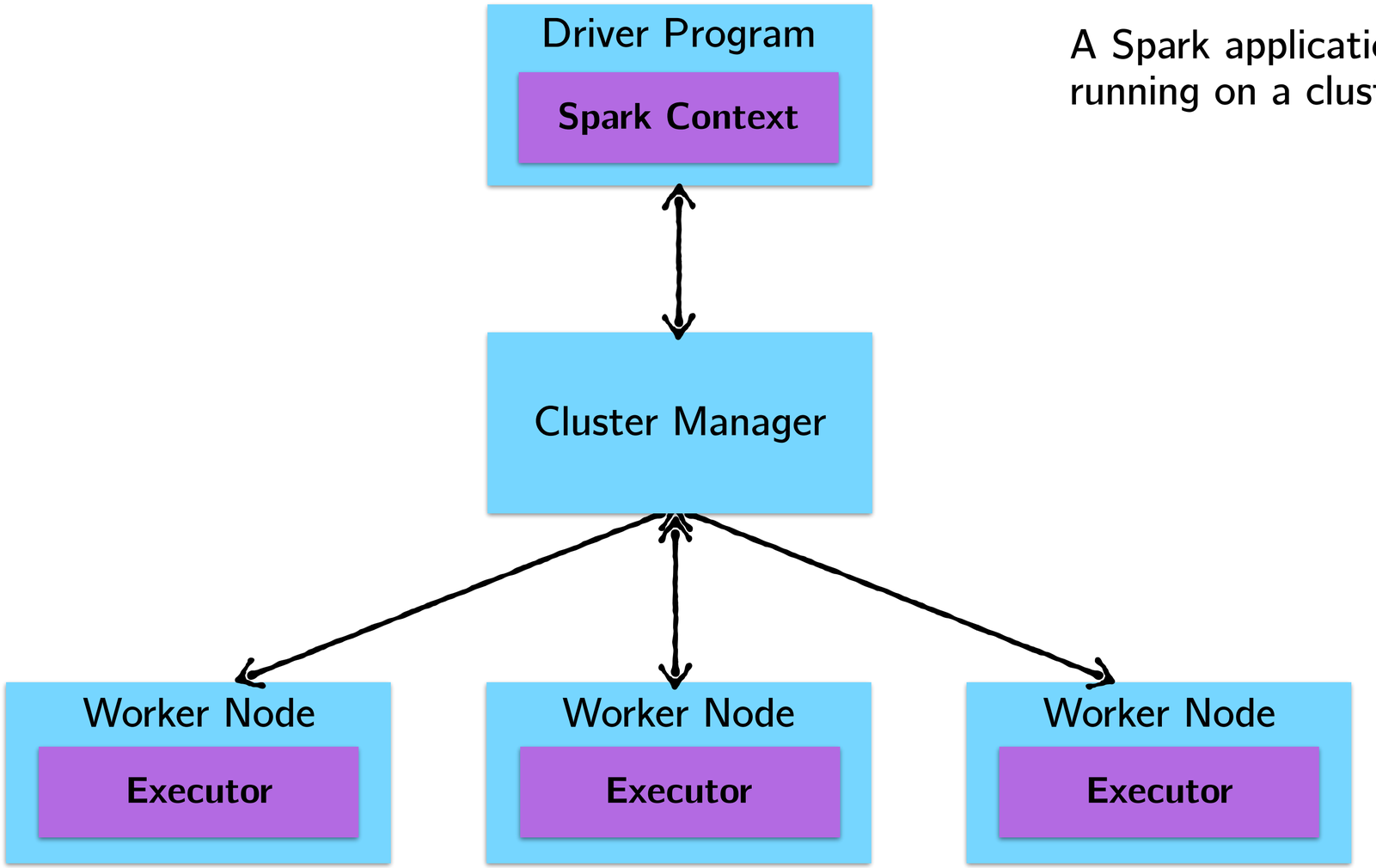
These are the nodes actually executing the jobs!



How Spark Jobs are Executed

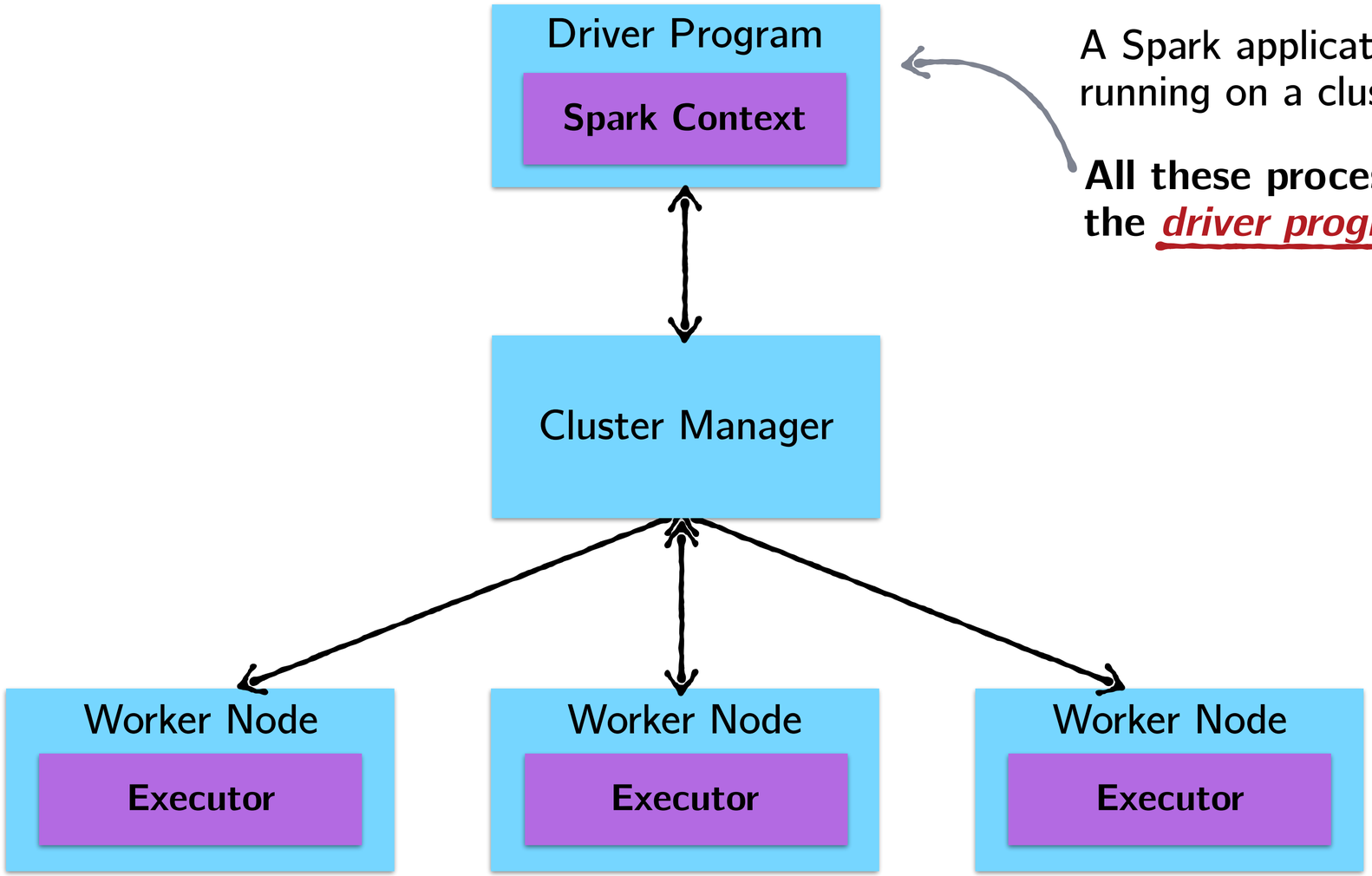


How Spark Jobs are Executed



A Spark application is a set of processes running on a cluster.

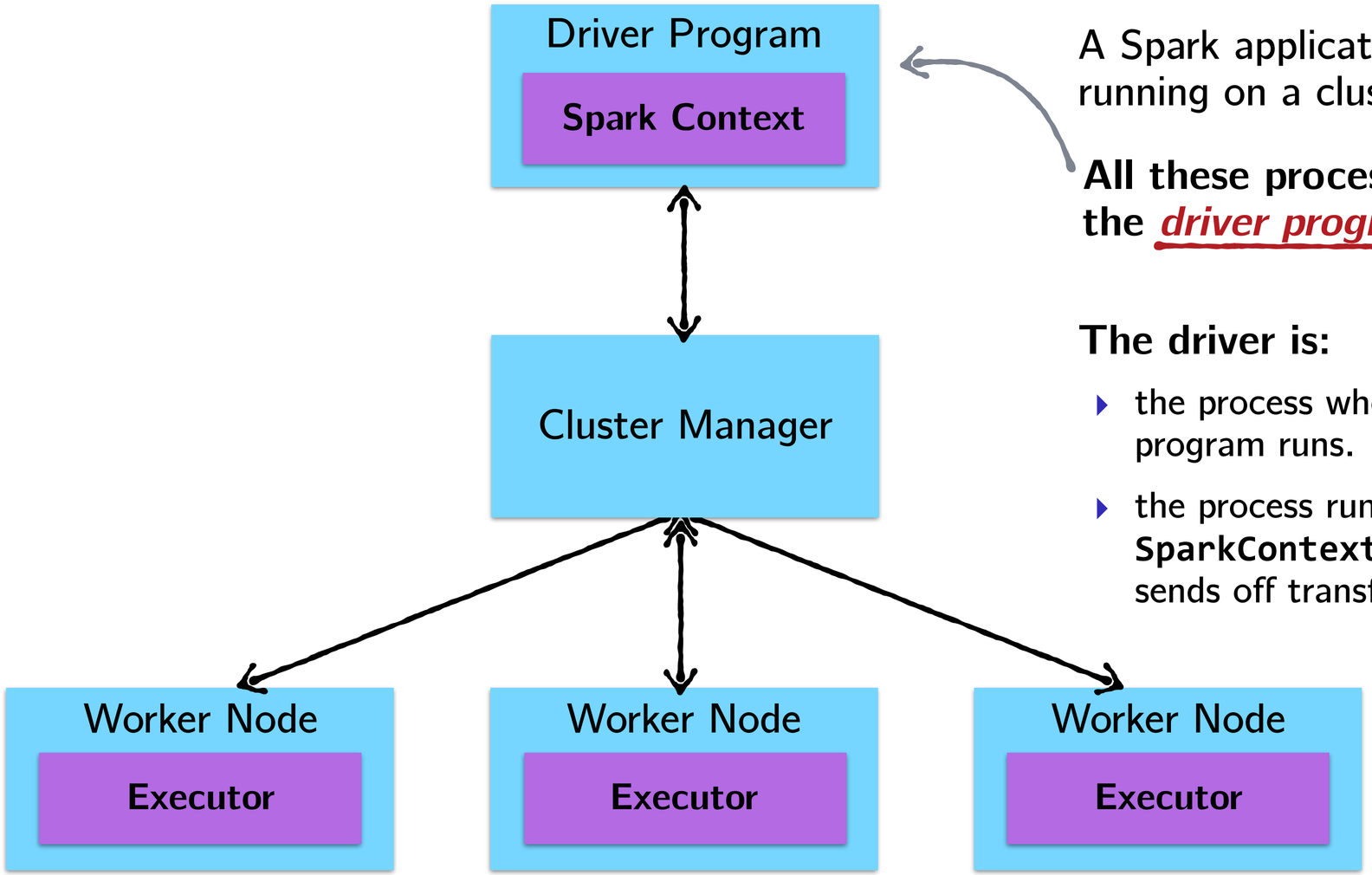
How Spark Jobs are Executed



A Spark application is a set of processes running on a cluster.

All these processes are coordinated by the *driver program*.

How Spark Jobs are Executed



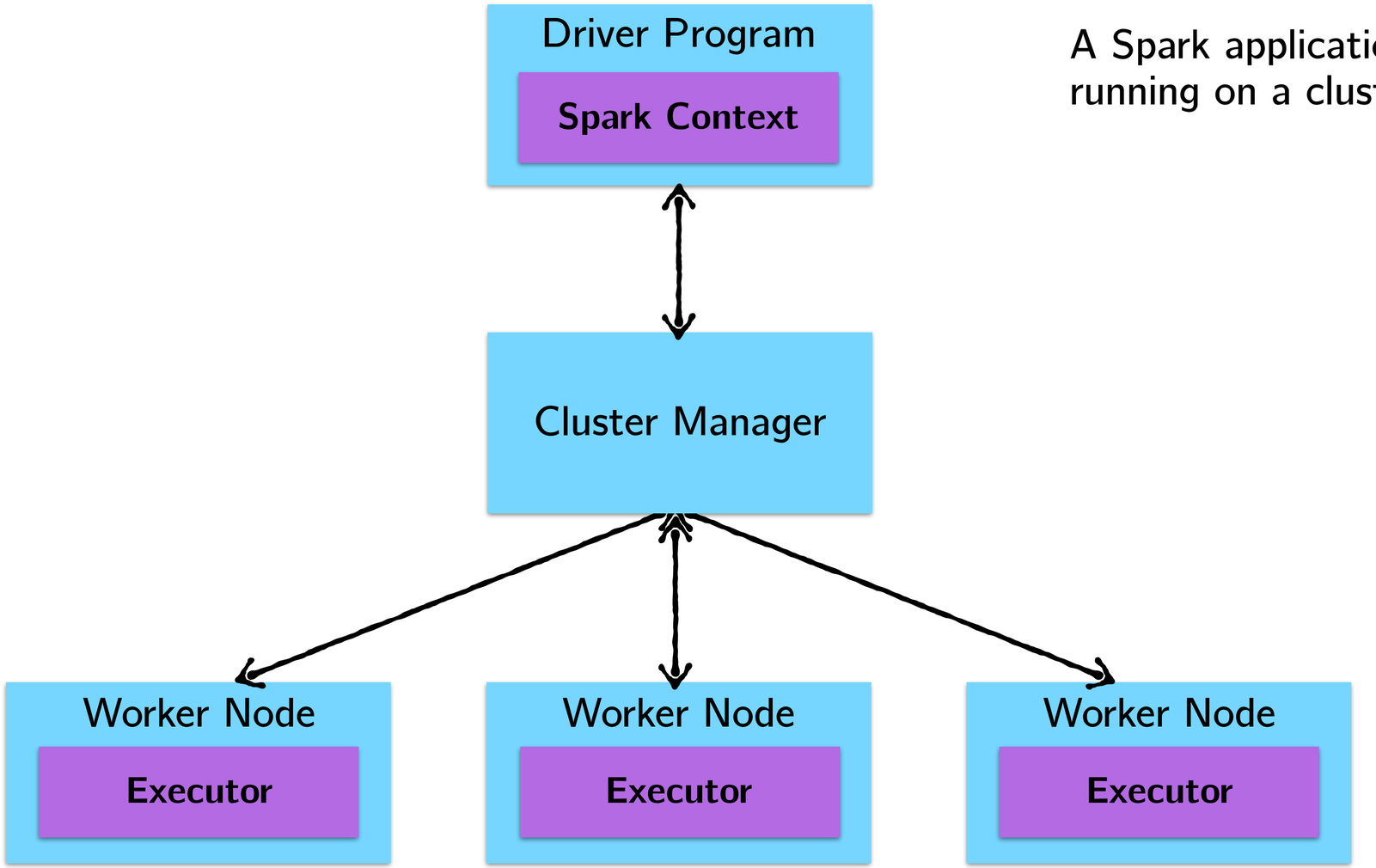
A Spark application is a set of processes running on a cluster.

All these processes are coordinated by the *driver program*.

The driver is:

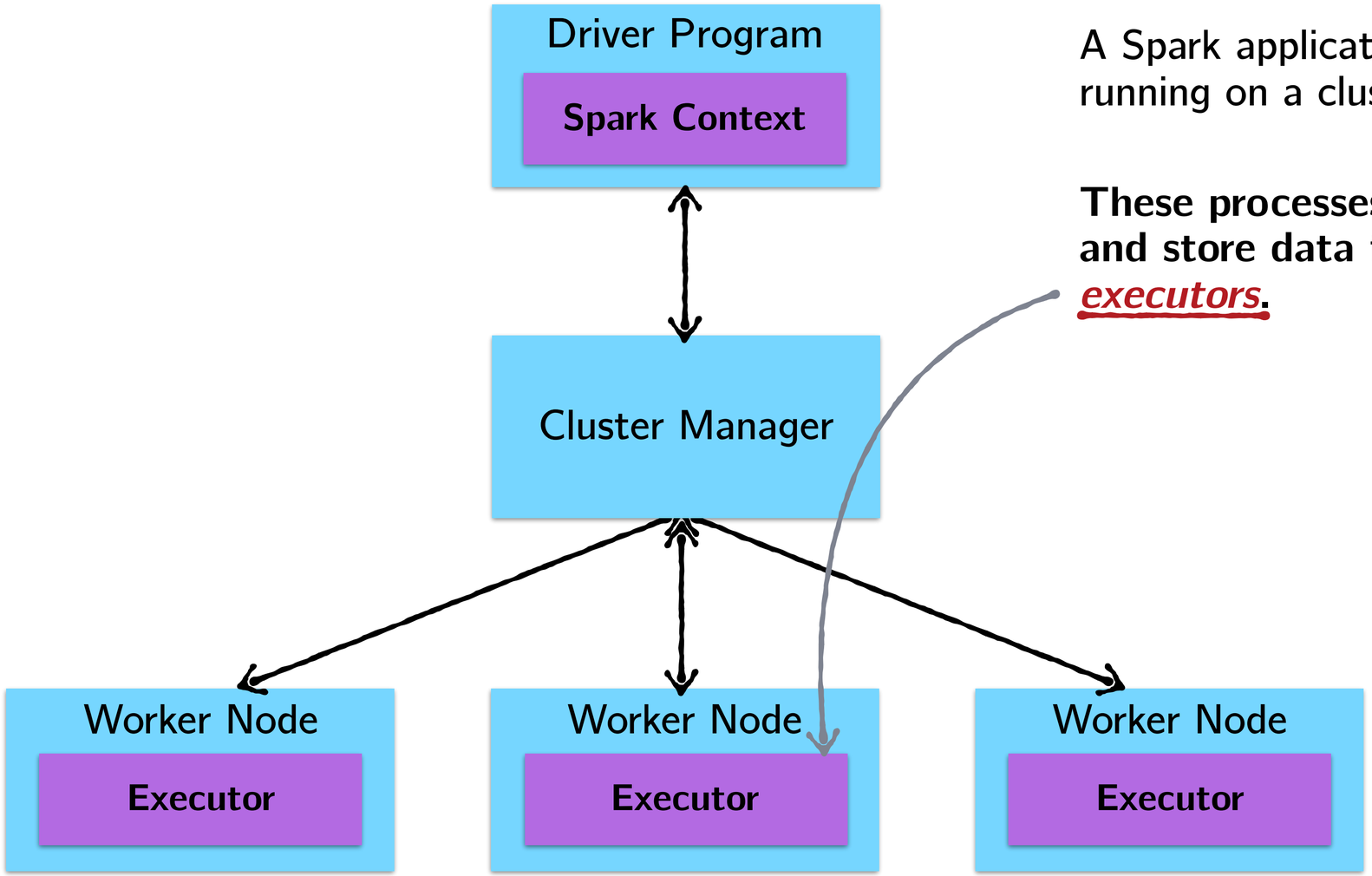
- the process where the `main()` method of your program runs.
- the process running the code that creates a `SparkContext`, creates `RDDs`, and stages up or sends off transformations and actions.

How Spark Jobs are Executed



A Spark application is a set of processes running on a cluster.

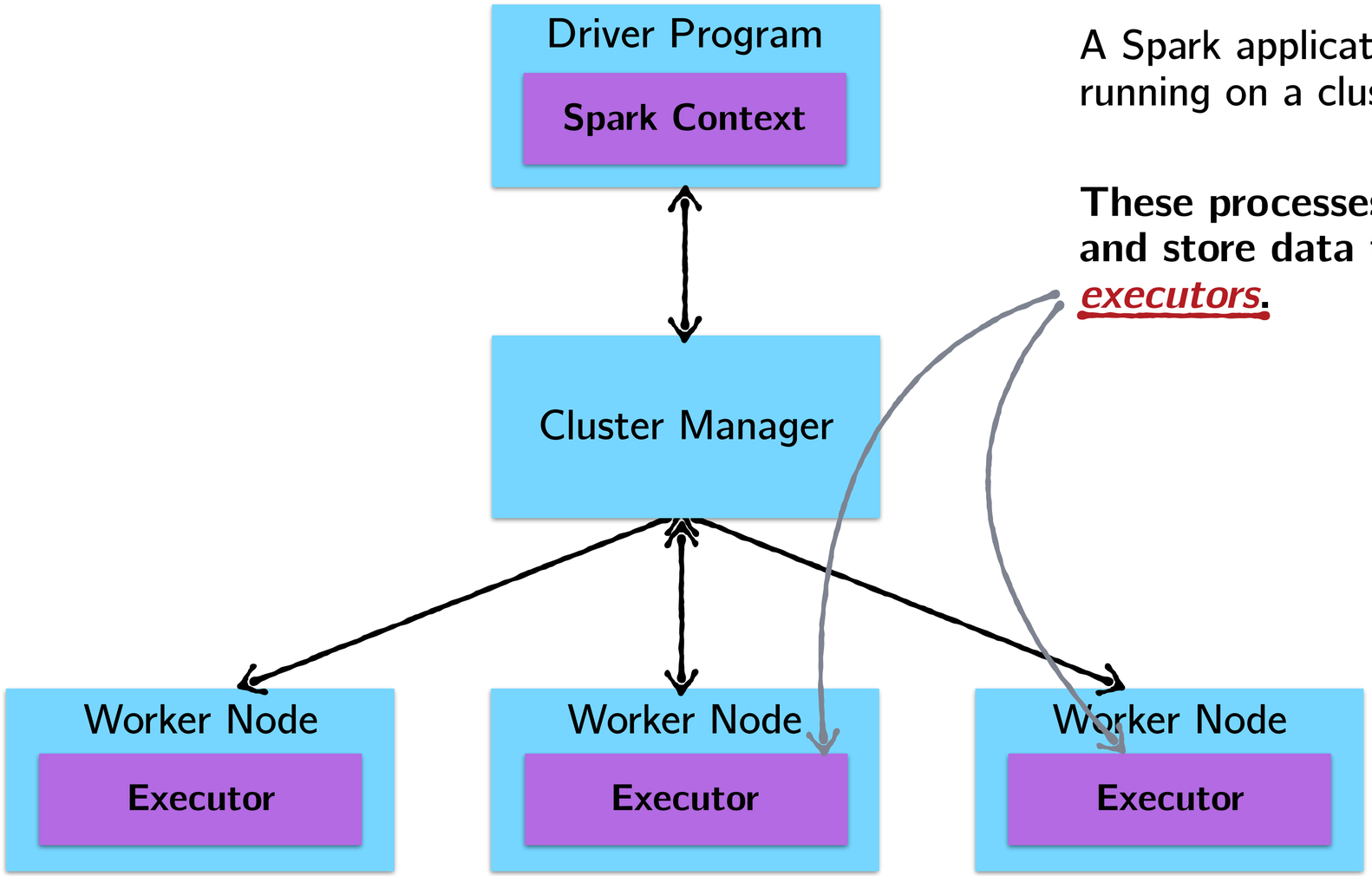
How Spark Jobs are Executed



A Spark application is a set of processes running on a cluster.

These processes that run computations and store data for your application are *executors*.

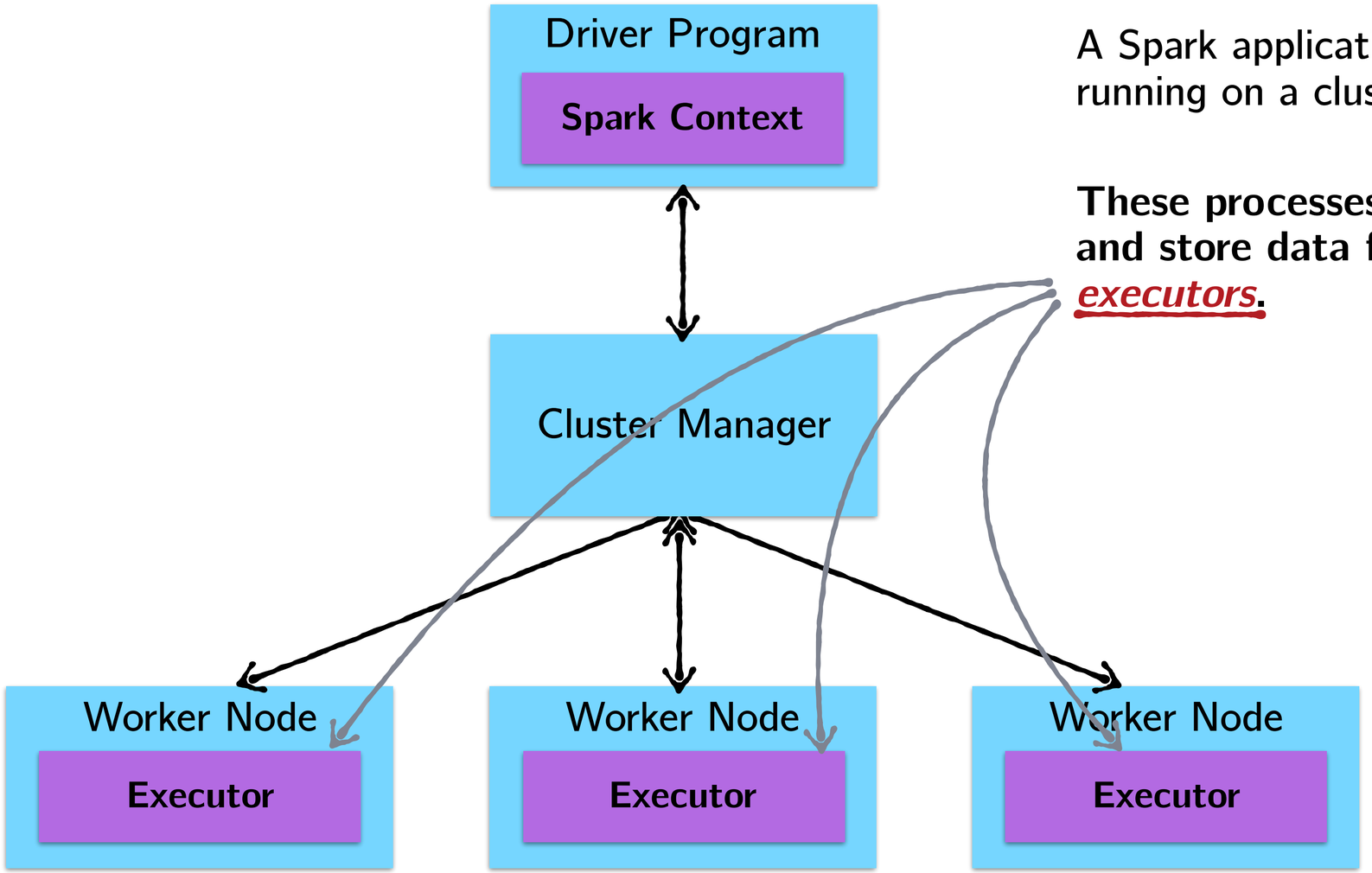
How Spark Jobs are Executed



A Spark application is a set of processes running on a cluster.

These processes that run computations and store data for your application are *executors*.

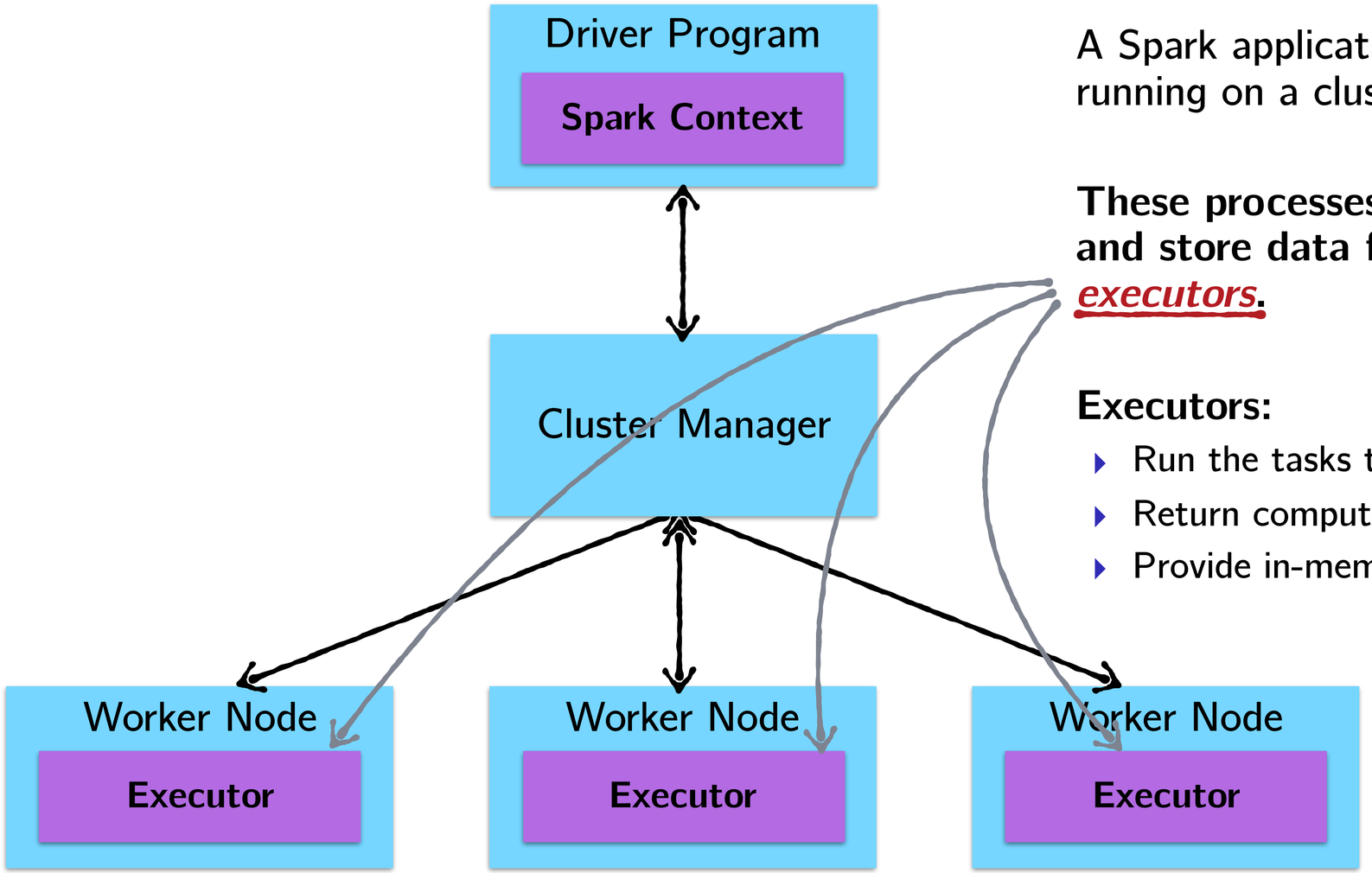
How Spark Jobs are Executed



A Spark application is a set of processes running on a cluster.

These processes that run computations and store data for your application are *executors*.

How Spark Jobs are Executed



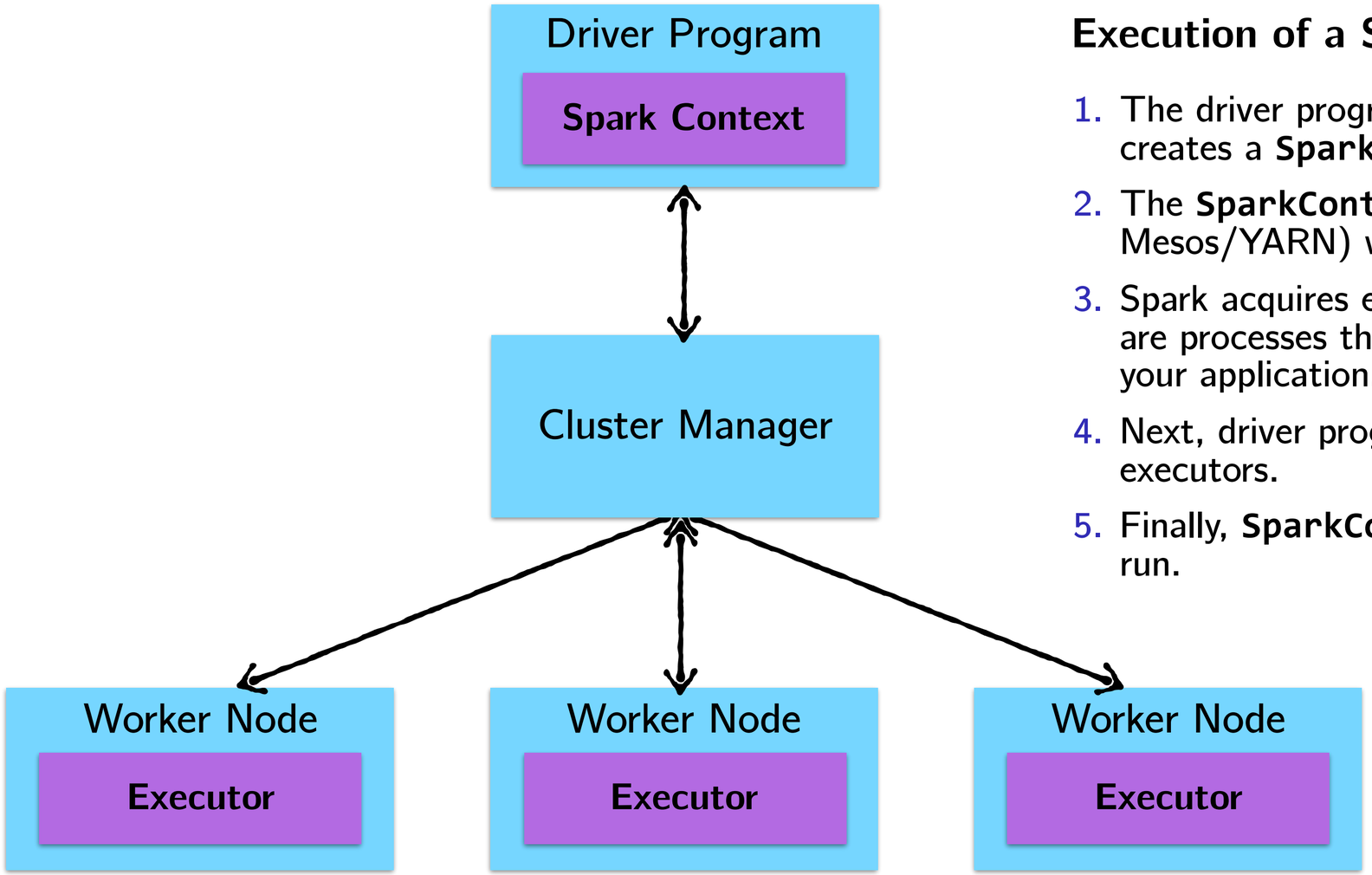
A Spark application is a set of processes running on a cluster.

These processes that run computations and store data for your application are *executors*.

Executors:

- Run the tasks that represent the application.
- Return computed results to the driver.
- Provide in-memory storage for cached RDDs.

How Spark Jobs are Executed



Execution of a Spark program:

1. The driver program runs the Spark application, which creates a **`SparkContext`** upon start-up.
2. The **`SparkContext`** connects to a cluster manager (e.g., Mesos/YARN) which allocates resources.
3. Spark acquires executors on nodes in the cluster, which are processes that run computations and store data for your application.
4. Next, driver program sends your application code to the executors.
5. Finally, **`SparkContext`** sends tasks for the executors to run.

Back to Example 1: A Simple `println`

Let's start with an example. Assume we have an RDD populated with `Person` objects:

```
case class Person(name: String, age: Int)
```

What does the following code snippet do?

```
val people: RDD[Person] = ...
people.foreach(println)
```

Back to Example 1: A Simple `println`

Let's start with an example. Assume we have an RDD populated with `Person` objects:

```
case class Person(name: String, age: Int)
```

What does the following code snippet do?

```
val people: RDD[Person] = ...
people.foreach(println)
```

On the driver: Nothing. Why?

Back to Example 1: A Simple `println`

Let's start with an example. Assume we have an RDD populated with `Person` objects:

```
case class Person(name: String, age: Int)
```

What does the following code snippet do?

```
val people: RDD[Person] = ...
people.foreach(println)
```

On the driver: Nothing. Why?

Recall that `foreach` is an action, with return type `Unit`. Therefore, it is eagerly executed on the executors, not the driver. Therefore, any calls to `println` are happening on the stdout of worker nodes and are thus not visible in the stdout of the driver node.

Back to Example 2: A Simple take

What about here? Assume we have an RDD populated with the same definition of `Person` objects:

```
case class Person(name: String, age: Int)
```

What does the following code snippet do?

```
val people: RDD[Person] = ...
val first10 = people.take(10)
```

Where will the `Array[Person]` representing `first10` end up?

Back to Example 2: A Simple take

What about here? Assume we have an RDD populated with the same definition of `Person` objects:

```
case class Person(name: String, age: Int)
```

What does the following code snippet do?

```
val people: RDD[Person] = ...
val first10 = people.take(10)
```

Where will the `Array[Person]` representing `first10` end up?

The driver program.

In general, executing an action involves communication between worker nodes and the node running the driver program.

Cluster Toplogy Matters!

Moral of the story:

To make effective use of RDDs, you have to understand a little bit about how Spark works under the hood.

Due an API which is mixed eager/lazy, it's not always immediately obvious upon first glance on what part of the cluster a line of code might run on.
It's on you to know where your code is executing!

Even though RDDs look like regular Scala collections upon first glance, unlike collections, RDDs require you to have a good grasp of the underlying infrastructure they are running on.