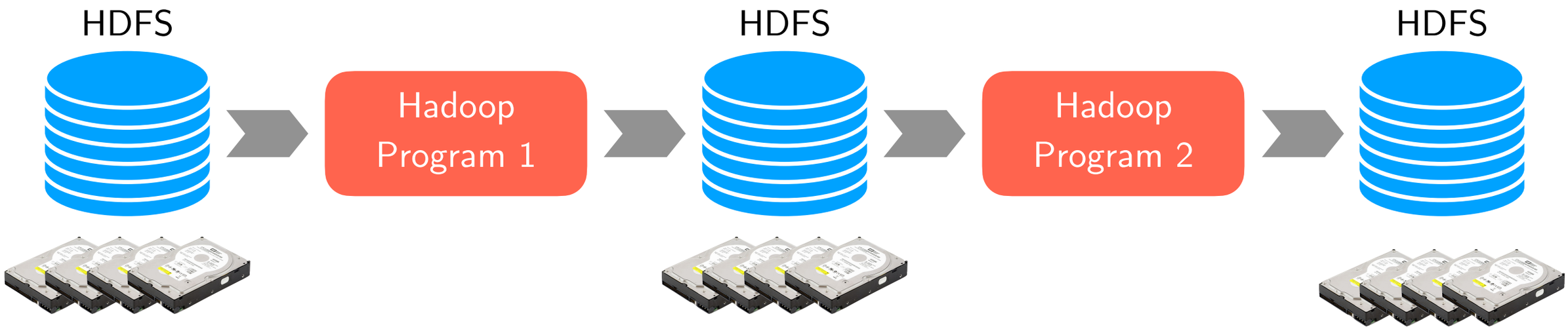
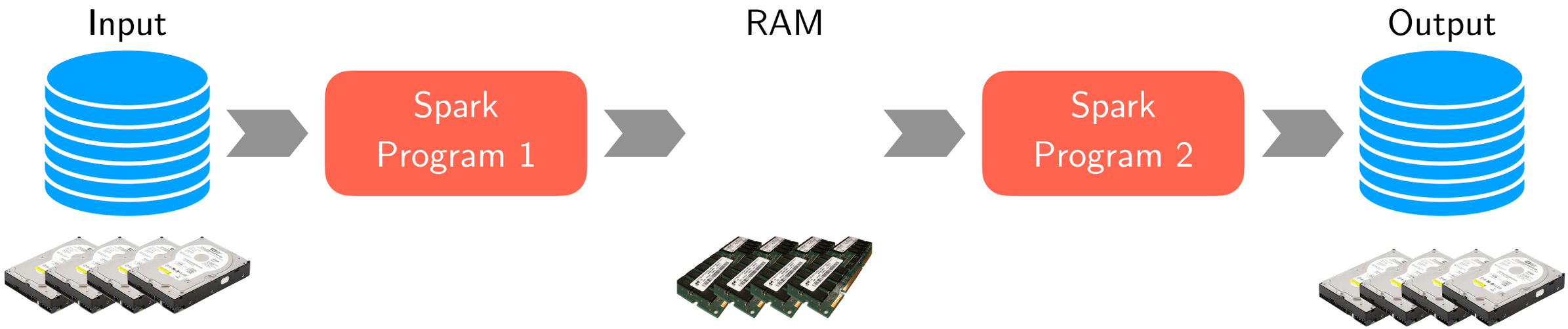
Spark

Hadoop vs Spark





Data Flow Models

- **Restrict** the programming interface so that the system can do more **automatically**
- Express jobs as **graphs** of high-level operators
  - System chooses how to **split** each operator into tasks and **where** to run each task
  - Run parts multiple times for **fault** recovery
- Biggest example: **MapReduce**

Limitations of Map Reduce

- MapReduce is great at **one-pass** computation
- Inefficient for **multi-pass** algorithms
- No efficient primitives for **data sharing**
  - State between steps goes to distributed file system
  - Slow due to replication & disk storage
- Example: **PageRank**
  - **Repeatedly** multiply sparse matrix and vector
  - Requires **repeatedly** hashing together page adjacency lists and rank vector
- While MapReduce is simple, it can require **asymptotically more communication or I/O**
- MapReduce algorithms research doesn't go to waste: still useful to study as an **algorithmic framework**, silly to **use directly**

Spark Stack

Spark SQL
structured data

Spark Streaming
realtime

MLlib
machine learning

GraphX
graph processing

Spark Core

Standalone
Scheduler

MESOS

YARN

Spark Core

- Provides **basic functionalities**, including:
  - task scheduling,
  - memory management,
  - fault recovery,
  - interacting with storage systems
- Provides a data abstraction called **resilient distributed dataset** (**RDD**), a collection of items distributed across many compute nodes that can be manipulated in parallel
  - Spark Core provides many APIs for building and manipulating these collections
- Written in **Scala** but APIs for **Java**, **Python** and **R**

Spark Modules

- **Spark SQL**
  - To work with structured data
  - Allows querying data via SQL
  - Extends the Spark RDD API
- **Spark Streaming**
  - To process live streams of data
  - Extends the Spark RDD API
- **MLlib**
  - Scalable machine learning (ML) Library
  - Many distributed algorithms: feature extraction, classification, regression, clustering, recommendation, ...
- **GraphX**
  - API for manipulating graphs and performing graph-parallel computations
  - Includes also common graph algorithms (e.g., PageRank)
  - Extends the Spark RDD API

RDD (I)

- A **resilient distributed dataset** (RDD) is a distributed memory abstraction
- Immutable collection of objects spread across the cluster

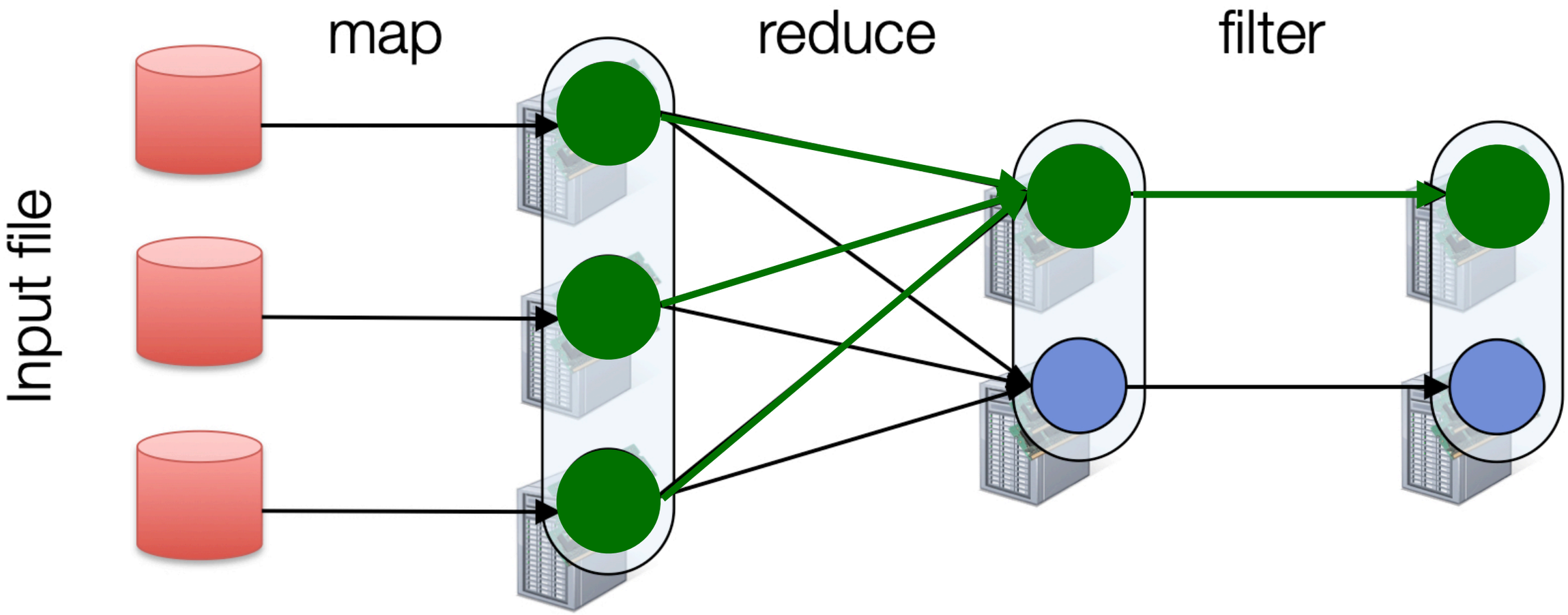
- An RDD is divided into a number of **partitions**, which are atomic pieces of information
- Partitions of an RDD can be stored on different nodes of a cluster

RDD (II)

- Collections of objects across a cluster with **user controlled partitioning & storage** (memory, disk, ...)
- Built via **parallel transformations** (`map`, `filter`, ...)
- The world only lets you make make RDDs such that **they can be automatically rebuilt on failure**

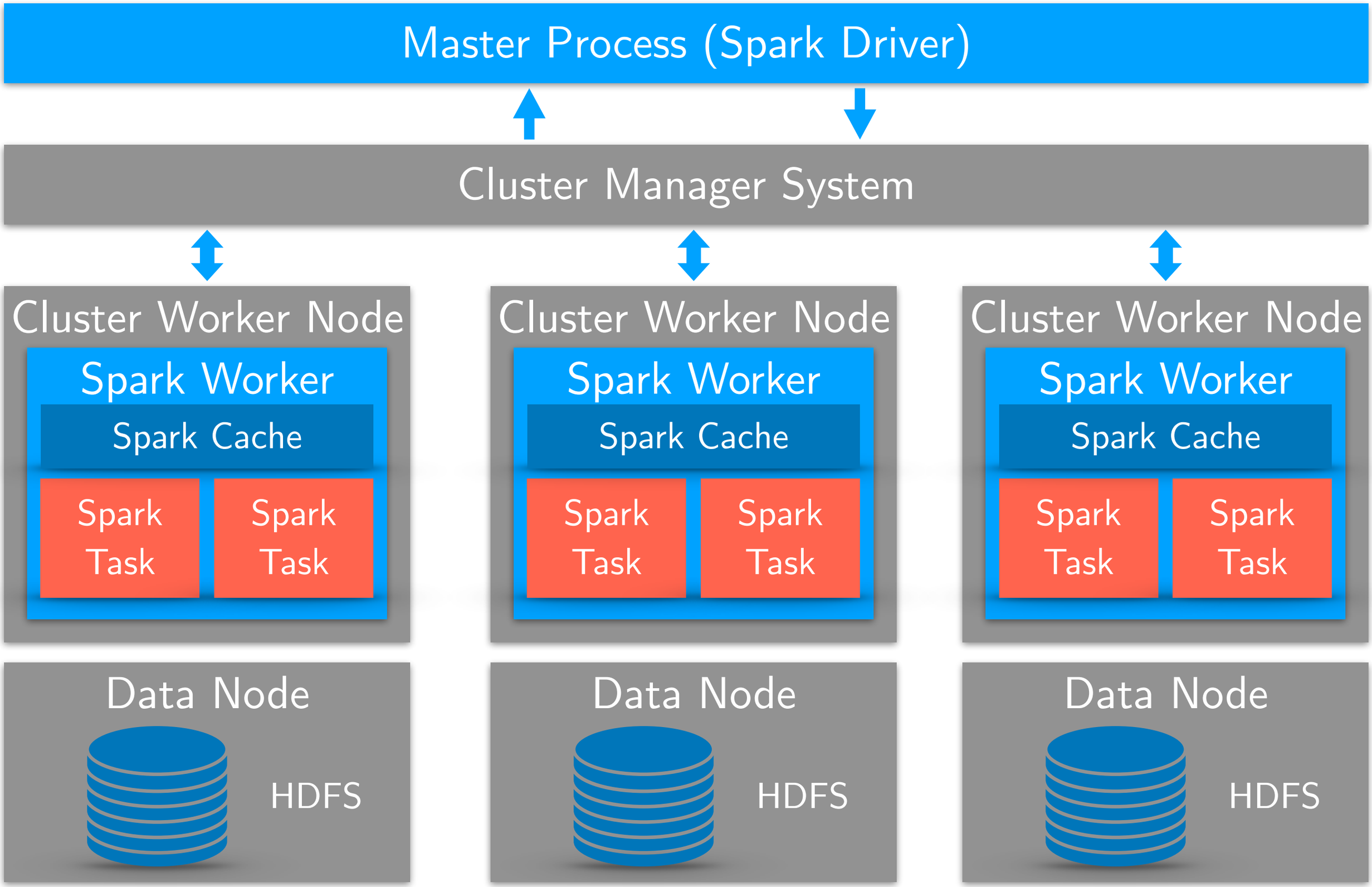
Lineage

```
file.map(lambda rec: (rec.type, 1))
    .reduceByKey(lambda x, y: x + y)
    .filter(lambda (type, count): count > 10)
```



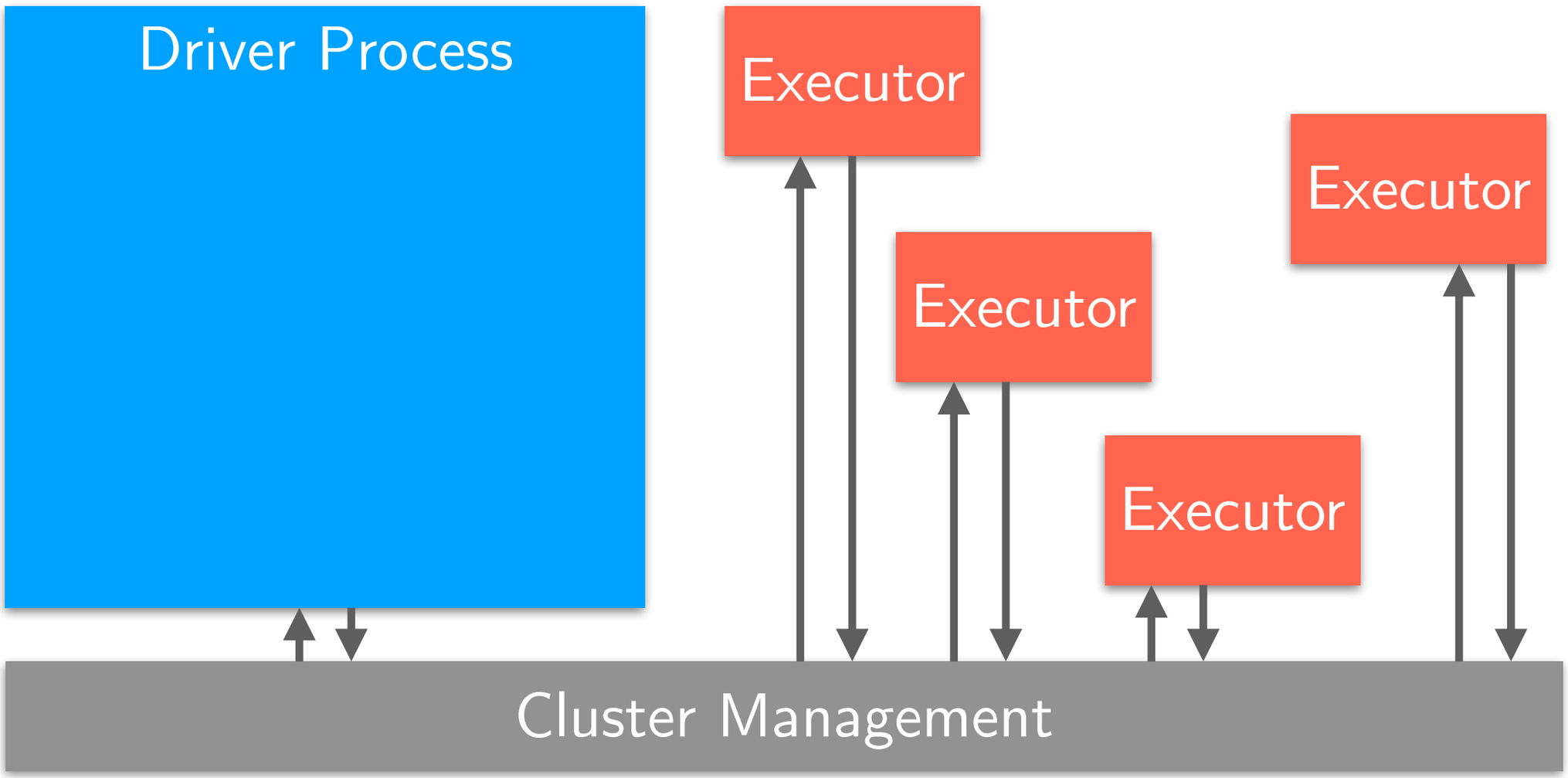
- RDDs track **lineage** info to rebuild lost data

Spark Architecture



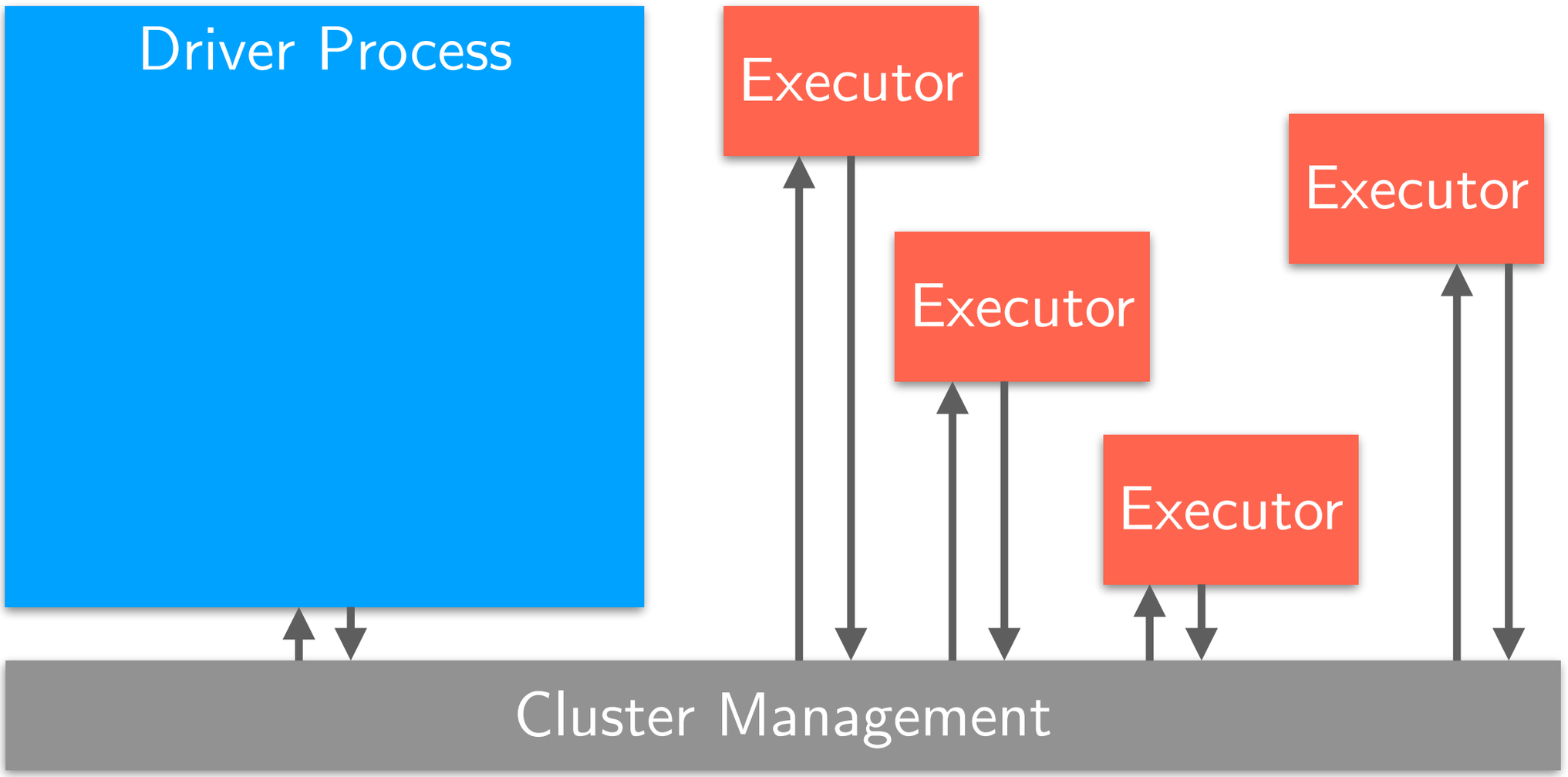
Spark Applications Architecture

- A **Spark application** consists of
  - a **driver** process
  - a **set of executor** processes



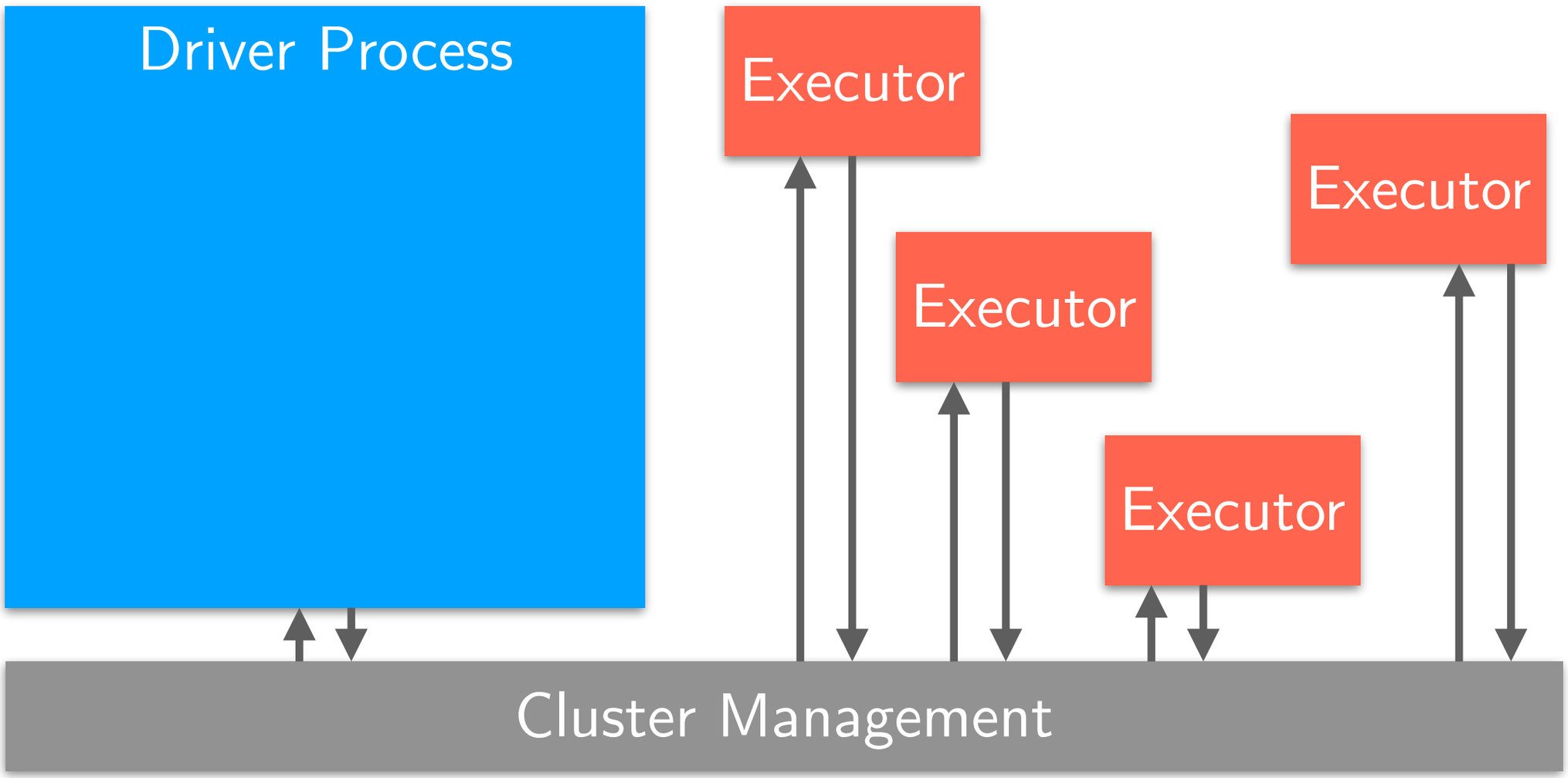
Spark Driver

- The driver process is
  - the heart of a Spark application
  - runs in a node of the cluster
  - runs the `main()` function



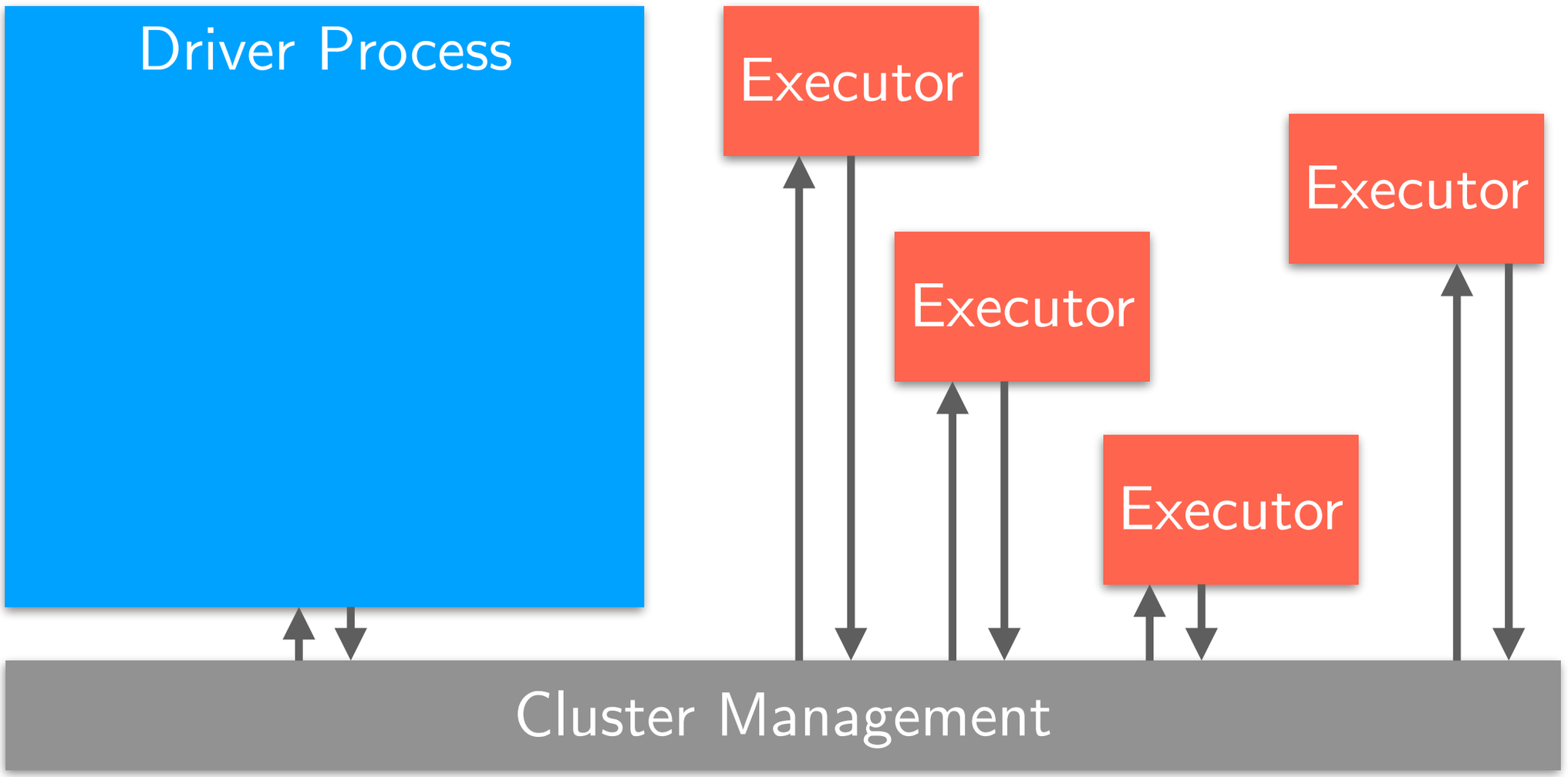
Spark Driver

- Responsible for three things:
  1. **Maintaining information** about the Spark application
  2. **Interacting** with the user
  3. **Analyzing, distributing and scheduling** work across the executors



Spark Executors

- Responsible for two things:
  1. **Executing code** assigned to it by the driver
  2. **Reporting the state** of the computation on that executor back to the driver



Spark Context

- The driver process is composed by:
  - A **spark context**
  - A **user code**