Ciencia de Datos a través del Big Data

Diego García (djgarcia@ugr.es)
Isaac Triguero (isaaktriguero@ugr.es)

Large-Scale Data Analytics with Python and Spark
A Hands-on Guide to Implementing Machine Learning Solutions

Chapter 4

Spark I

© Isaac Triguero and Mikel Galar

Learning Outcomes

CAMBRIDGE UNIVERSITY PRESS

- The importance of caching for efficient Big Data processing [KU]
- Extending Map Reduce: The concept of distributed datasets (RDDs) for Big Data [KU]
- How to operate with RDDs: lazy transformation vs. actions [KU, IS, PPS]
- How a Spark application is distributed guaranteeing fault-tolerance and transparency [KU]
- How to use Spark in practice for problem solving [IS, PPS]

Objective for today

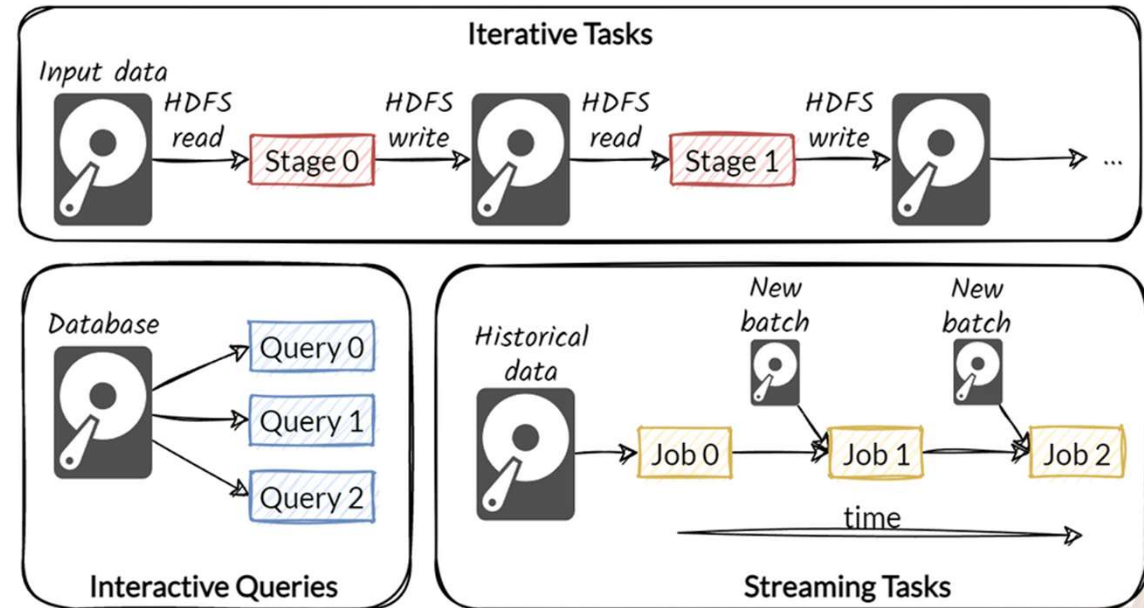
CAMBRIDGE UNIVERSITY PRESS

- Recap and introduction to Apache Spark
- How to work with Apache Spark
  - Basic concepts
  - RDD
  - (**Basic**) Operations with Spark

Introduction and Motivation

CAMBRIDGE UNIVERSITY PRESS

- Executing MapReduce for:
  - Iterative Tasks
  - Interactive queries
  - Streaming
- **Would require multiple accesses to disk**
  - I/O is SLOW!
- Can we keep the data in main memory?



Note: Even between map and reduce phases, there are many I/O operations

What is Spark?

CAMBRIDGE UNIVERSITY PRESS

- General execution graphs: it is a **data processing engine only**
- **In-memory storage**

Fast and **Expressive** Cluster Computing
Engine Compatible with Apache Hadoop

Up to **10×** faster on disk,
100× in memory

- Efficient

2-5× less code

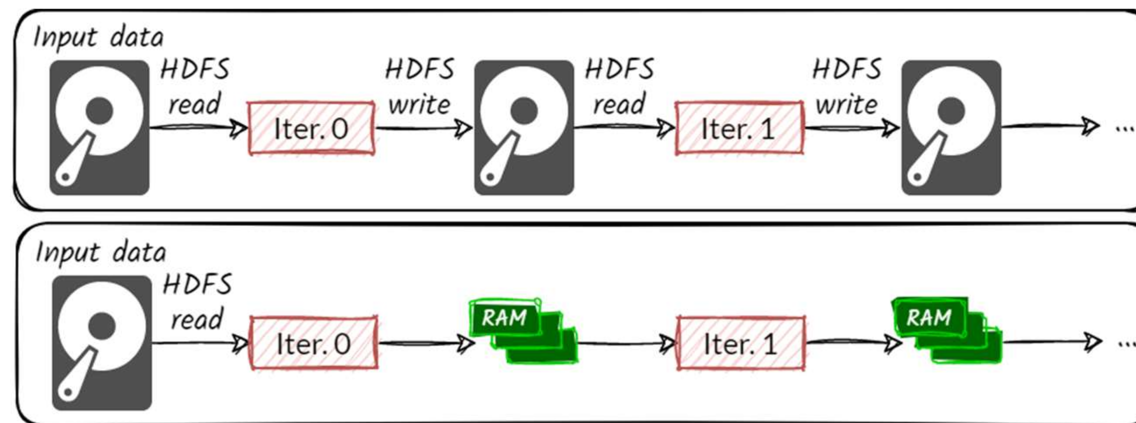
- Usable
- Rich APIs in Java, Scala, Python, R
- Interactive shell

What’s new in Apache Spark?

CAMBRIDGE UNIVERSITY PRESS

- **Store data in main memory**
  - Much faster than serialising every stage on a drive
  - More efficient when storing on a drive

- **A new Data Processing Engine (only) for Big Data**
  - Extends the MapReduce paradigm to *Interactive*, *iterative* and *stream* processing



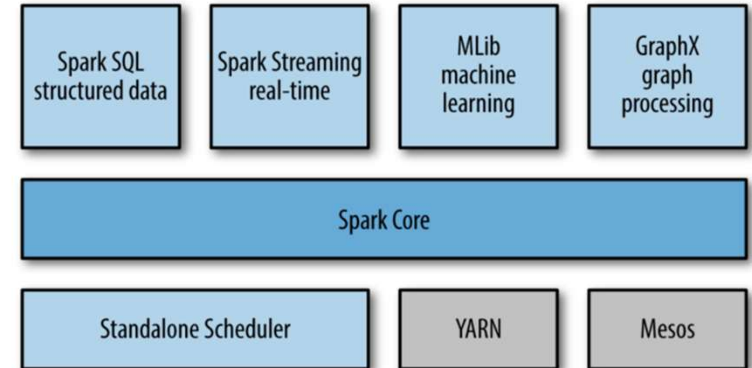
Zaharia, M., Chowdhury, M., Franklin, M. J, Shenker, S., and Stoica, I. 2010. Spark: Cluster computing with working sets. In: *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*. Link

What is Spark?

CAMBRIDGE UNIVERSITY PRESS

As a Data processing engine only it comes:

- Without a distributed file system
  - Uses other existing DFS
    - HDFS, NoSQL...
    - Hadoop is not a prerequisite
- Works with different cluster management tools
  - Hadoop (YARN)
  - Mesos
  - Standalone mode (included in Spark)



Source: Spark documentation

- Provides different libraries (e.g., Streaming, Machine Learning)

Spark Goal

CAMBRIDGE UNIVERSITY PRESS

- Provide **distributed memory abstractions** for clusters of computers
- **Retain the attractive properties of MapReduce:**
  - Fault tolerance (for crashes & stragglers)
  - Data locality
  - Scalability

Solution: augment data flow model with "resilient distributed datasets" (RDDs)

Zaharia, M. *et al*. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for in-Memory Cluster Computing. Page 2 of: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. NSDI'12. USA: USENIX Association. Link

Resilient Distributed Datasets (RDDs)

CAMBRIDGE UNIVERSITY PRESS

- An **RDD** *is a fault-tolerant collection of elements that can be operated on in parallel and can be cached for future reuse*
- Two types of parallel operations:
  - **Transformations** (e.g., `map`, `filter`, `groupBy`, `join`);
    - Lazy operations to build RDDs from other RDDs
  - **Actions** (e.g., `count`, `collect`, `save`)
    - Return a result or write it to storage
- RDDs are reconstructed automatically in case of failure

Spark vs Hadoop MapReduce

CAMBRIDGE UNIVERSITY PRESS

	Hadoop MapReduce	Spark
Storage	Disk only	In-memory or on disk
Operation	Map and Reduce	Map, Reduce, Join, Sample, etc…
Execution mode	Batch	Batch, interactive, streaming
Programming languages	Java	Scala, Java, R, and Python

Spark Birth

CAMBRIDGE UNIVERSITY PRESS

VS.

	Daytona
Gray	2013, 1.42 TB/min **Hadoop** 102.5 TB in 4,328 seconds 2100 nodes x (2 2.3Ghz hexcore Xeon E5-2630, 64 GB memory, 12x3TB disks) Thomas Graves Yahoo! Inc.

	Daytona
Gray	2-way tie: 2014, 4.35 TB/min **TritonSort** 100 TB in 1,378 seconds 186 Amazon EC2 i2.8xlarge nodes x (32 vCores - 2.50Ghz Intel Xeon E5-2670 v2, 244GB memory, 8x800 GB SSD) Michael Conley, Amin Vahdat, George Porter University of California, San Diego 2014, 4.27 TB/min **Apache Spark** 100 TB in 1,406 seconds 207 Amazon EC2 i2.8xlarge nodes x (32 vCores - 2.5Ghz Intel Xeon E5-2670 v2, 244GB memory, 8x800 GB SSD) Reynold Xin, Parviz Deyhim, Xiangrui Meng, Ali Ghodsi, Matei Zaharia Databricks

http://sortbenchmark.org/

Basic Concepts

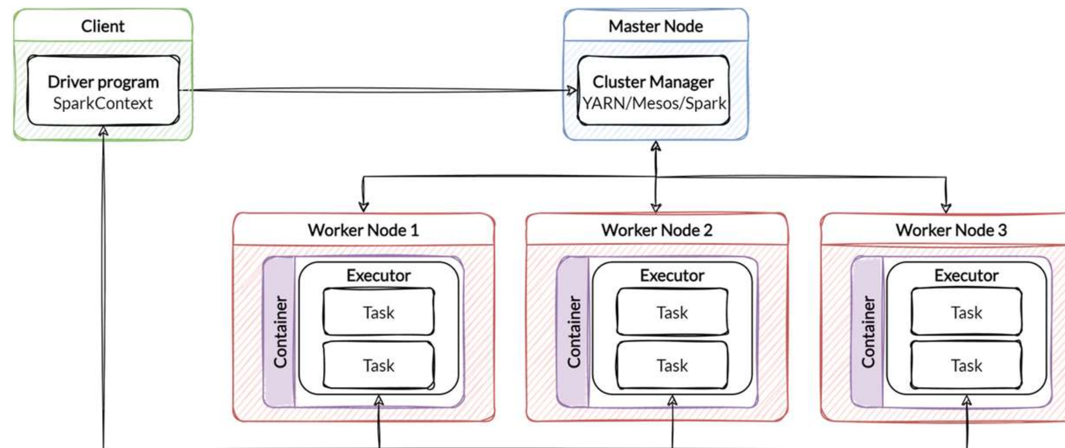
CAMBRIDGE UNIVERSITY PRESS

- *How do we use Spark?*
  - There are APIs for Scala, Java, R, and Python
  - We will use **pyspark**, the Python Interface for Spark
    - Interactive application – **this course** (with Jupyter Notebooks)
    - Standalone application

Basic Concepts

CAMBRIDGE UNIVERSITY PRESS

- A **Spark program is composed of two programs**
  - **Driver** – Runs anything sequential
  - **Workers** – Parallel operations
    - Executed in the computing nodes
    - Or in local threads
  - RDDs are distributed across the whole cluster



Basic Concepts

CAMBRIDGE UNIVERSITY PRESS

- The `SparkContext` is the main entry point to Spark functionality
  - This tells Spark how and where to access a cluster
  - **pyspark** creates it for you automatically
  - Any Python program (or Jupyter notebook) should use a `SparkContext` constructure to work
  - This allows us to create new RDDs

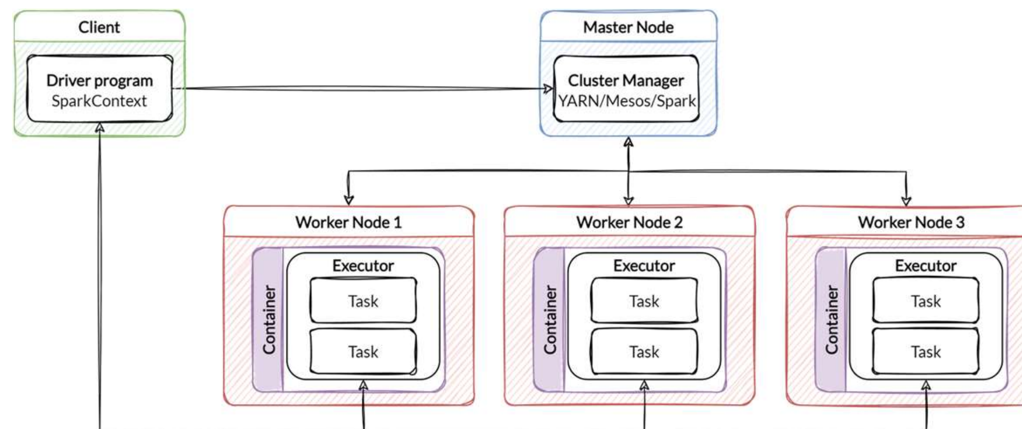
```
conf = SparkConf().setMaster(master).setAppName("My App")
```

where `master` is a Spark, Mesos or YARN cluster URL, or a special `"local[*]"` string to run in local mode.

`*` means to use all available threads as executors. Could be the number to use

Basic Concepts

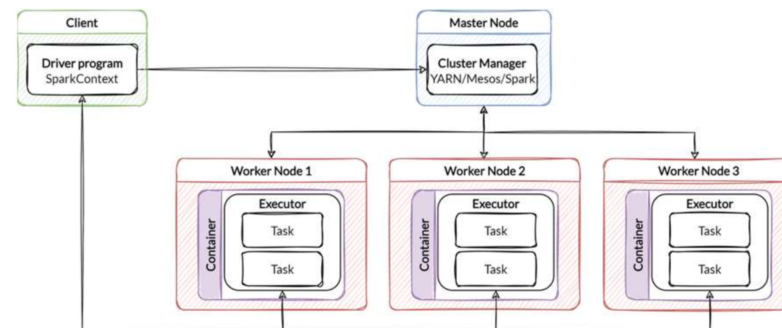
- `SparkContext` (sc) is created in the driver
  - Using the sc, **a connection with the cluster manager is established**
  - Once connected, executors are requested
  - An executor is a process that performs the computation and stores the data
  - The driver sends the code and tasks to the executors



Basic Concepts

CAMBRIDGE UNIVERSITY PRESS

- Some remarks:
  - **Each program has its own executors**
    - **Advantage**: Isolation
    - **Disadvantage**: Data sharing is not possible
  - **A worker could have more than one executor**
    - From different programs/applications
    - Depending on the number of cores and RAM memory needed
  - When using **YARN**, an **executor** runs in a **container**



RDDs

CAMBRIDGE UNIVERSITY PRESS

- **A distributed collection of objects**
  - **Immutable** when created
  - **Lineage** of RDDs is kept to ensure fault tolerance
  - It can contain Python, Java or Scala objects! Including your own classes
- There are three ways to create RDDs:
  - Parallelizing an existing collection in your driver program
  - Referencing a dataset in an external storage system, such as a shared filesystem, HDFS, Hbase
  - Transforming an existing RDD

RDDs

CAMBRIDGE UNIVERSITY PRESS

- Creating RDDs from Python collections:

```
>>> data = [1, 2, 3, 4, 5]
>>> data
[1, 2, 3, 4, 5]

>>> rDD = sc.parallelize(data, 4)
>>> rDD
ParallelCollectionRDD[0] at parallelize at
PythonRDD.scala:229
```

`sc.parallelize()` is a lazy operation. There isn't any kind of computation. Spark simply saves how to create the RDD with 4 partitions

RDDs

CAMBRIDGE UNIVERSITY PRESS

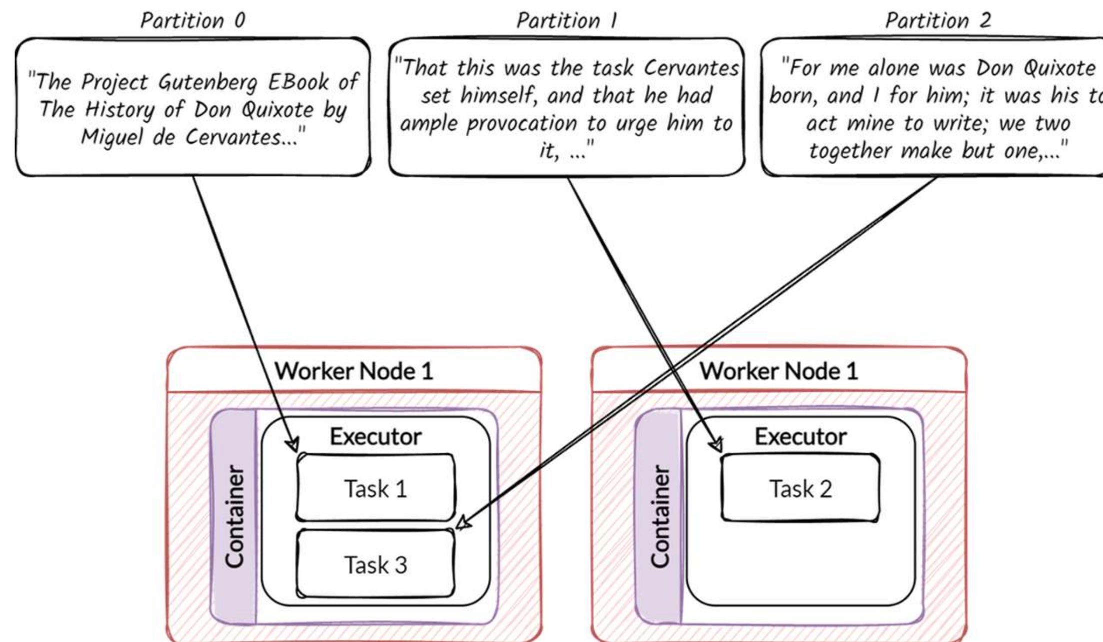
- Creating RDDs from an external storage system:
  - You can reference HDFS, Amazon S3, Hbase,...
  - If you use '*', you can load all files in a folder

```
>>> distFile = sc.textFile("README.md", 4)
>>> distFile
MappedRDD[2] at textFile at
NativeMethodAccessorImpl.java:-2
```

`sc.textFile()` is also a lazy operation

RDDs

- Spark partitions your file or collection automatically
  - This is the maximum degree of parallelism we can achieve
  - You can use `rdd.getNumPartitions()`
    - Is this an action or a transformation?



Cambridge University Press

- **RDDs – Simple example**
  - Most Spark operators use high-order functions
  - For example, to filter out the lines from an RDD, we can use anonymous/lambda functions:

```
>>> lines = sc.textFile("README.md")
>>> pythonLines = lines.filter(lambda line: "Python" in line)
>>> pythonLines.first()
u'## Interactive Python Shell'
```

All operations to RDDs are executed in parallel. So, a `line.contains("Python")` instruction is sent to all workers.

  - **Without lambda functions:**

```
def hasPython(line):
    return "Python" in line
pythonLines = lines.filter(hasPython)
```

RDDs – Basic Transformations

CAMBRIDGE UNIVERSITY PRESS

- **Transformations**: Create a new RDD from an existing one

Transformation	**Meaning**
`map(`*func*`)`	Return a new distributed dataset formed by passing each element of the source through a function *func*.
`filter(`*func*`)`	Return a new dataset formed by selecting those elements of the source on which *func* returns true.
`flatMap(`*func*`)`	Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item).

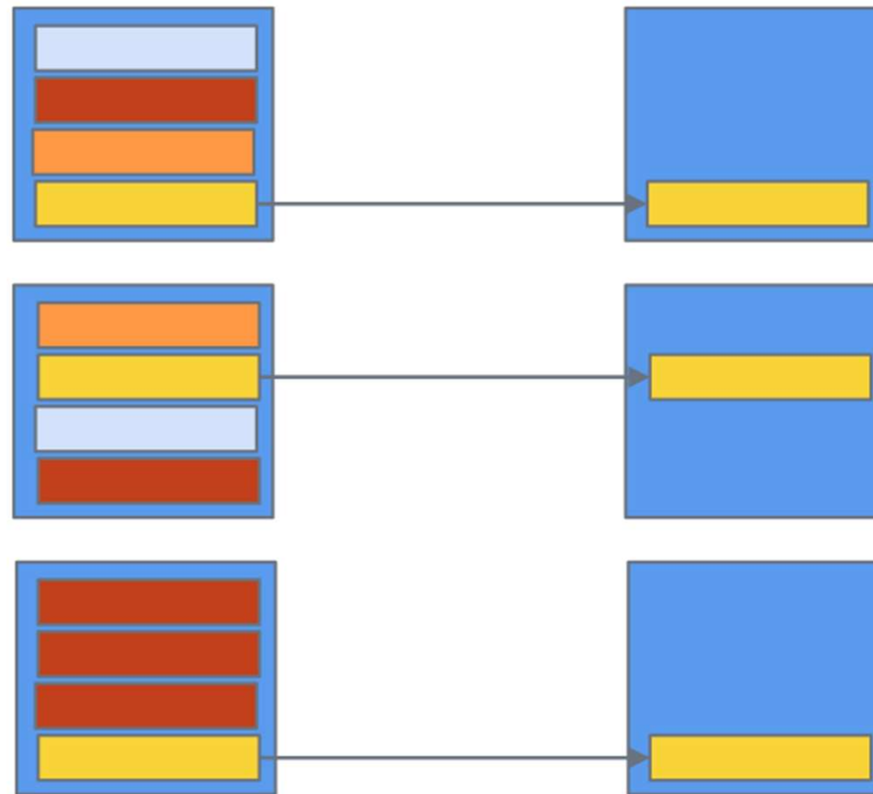
Basic Transformations: `map(lambda x: x+2)`

CAMBRIDGE UNIVERSITY PRESS

Source: Dirk Van den Poel. Spark: The new kid on the block (2014)

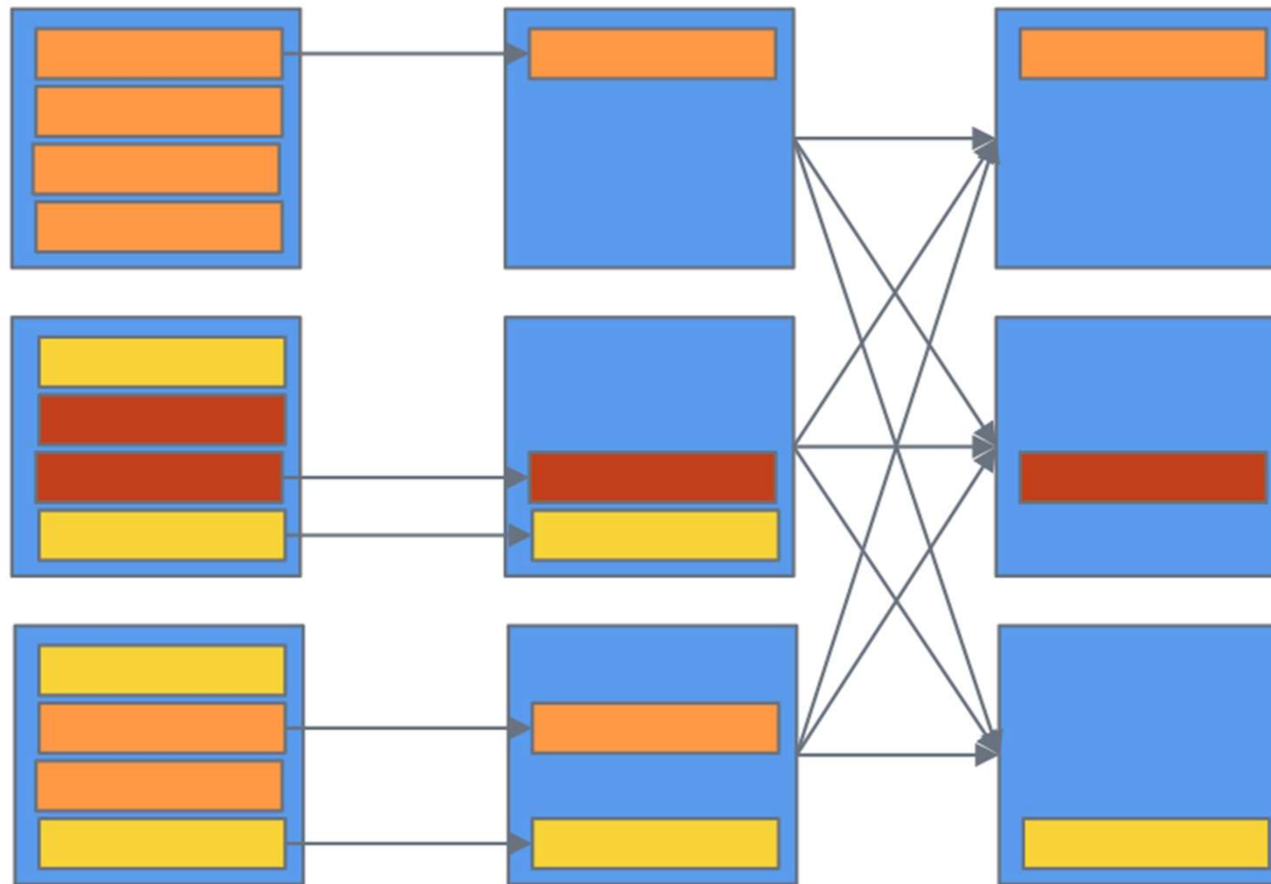
Basic Transformations: `filter(only yellow)`

CAMBRIDGE UNIVERSITY PRESS



Basic Transformations : `distinct()`

CAMBRIDGE UNIVERSITY PRESS



RDDs – Basic Transformations

CAMBRIDGE UNIVERSITY PRESS

- **Transformations**

```
>>> rdd = sc.parallelize([1, 2, 3, 4])
>>> rdd.map(lambda x: x * 2)
RDD: [1, 2, 3, 4] → [2, 4, 6, 8]

>>> rdd.filter(lambda x: x != 1)
RDD: [1, 2, 3, 4] → [2, 3, 4]


>>> rdd.filter(lambda x: x % 2 == 0)
RDD: [1, 2, 3, 4] → [2, 4]

>>> rdd2 = sc.parallelize([1, 4, 2, 2, 3])
>>> rdd2.distinct()
RDD: [1, 4, 2, 2, 3] → [1, 4, 2, 3]
```

The workers use the input function on the input RDD

RDDs – Basic Transformations

CAMBRIDGE UNIVERSITY PRESS

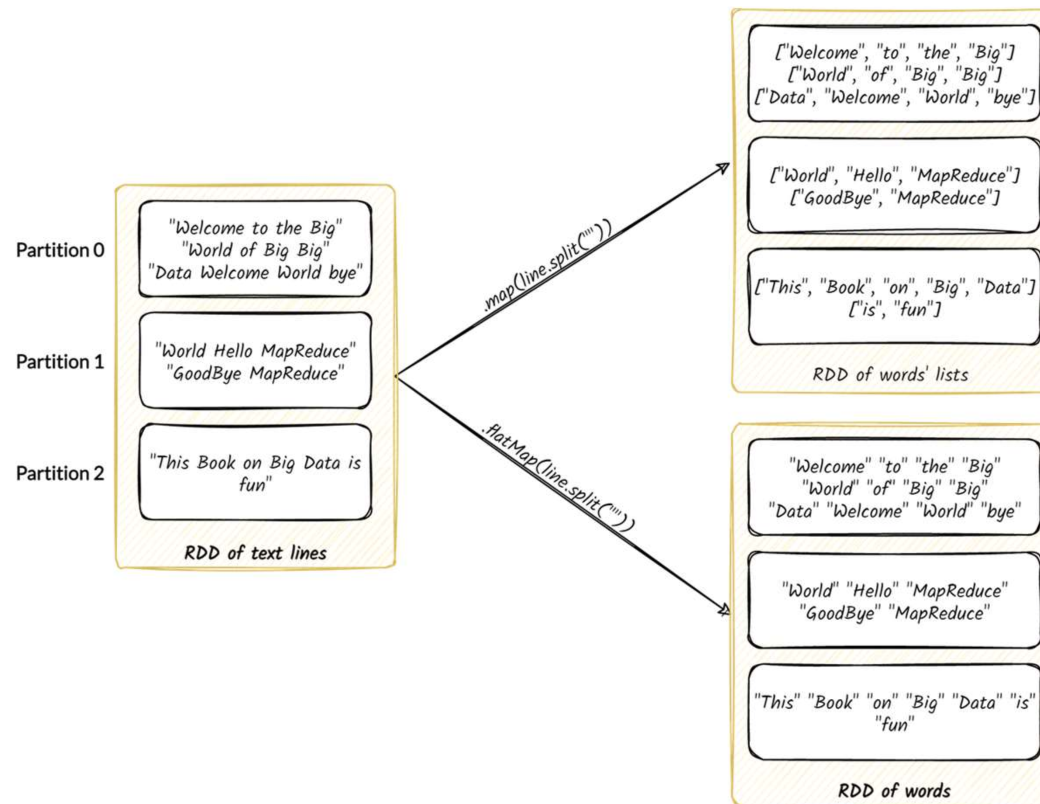
- **Transformations**

```
>>> rdd = sc.parallelize([1, 2, 3])
>>> rdd.Map(lambda x: [x, x+5])
RDD: [1, 2, 3] → [[1, 6], [2, 7], [3, 8]]

>>> rdd.flatMap(lambda x: [x, x+5])
RDD: [1, 2, 3] → [1, 6, 2, 7, 3, 8]



>>> lines = sc.parallelize(["hello world", "hi"])
>>> words = lines.flatMap(lambda line: line.split(" "))
RDD: ["hello world", "hi"] → [“hello”, “world”, “hi]
```

RDDs – Basic Transformations

CAMBRIDGE UNIVERSITY PRESS

- **Transformations**
  - Difference between `map()` and `flatMap()`



RDDs – Pseudo-sets

CAMBRIDGE UNIVERSITY PRESS

- **Transformations**
  - You can use set-like operations on RDDs
    - **RDDs are not really set structures**
    - All involved RDDs must be of the **same type**

RDD1
{water, wine, beer, water, water, wine}

RDD2
{beer, beer, water, water, wine, coca-cola, lemonade}

RDD1.distinct()
{wáter, beer, wine}

RDD1.union(RDD2)
{water, wine, beer, water, water, wine, beer, beer, water, water, wine, coca-cola, lemonade}

RDD1.intersection(RDD2)
{water, beer, wine}

RDD1.subsctract(RDD2)
{ }

RDDs – Actions

CAMBRIDGE UNIVERSITY PRESS

- **Actions**: Return a result to the driver node

Action	Meaning
`reduce(func)`	Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
`collect()`	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
`count()`	Return the number of elements in the dataset.
`first()`	Return the first element of the dataset (similar to take(1)).

RDDs – Actions

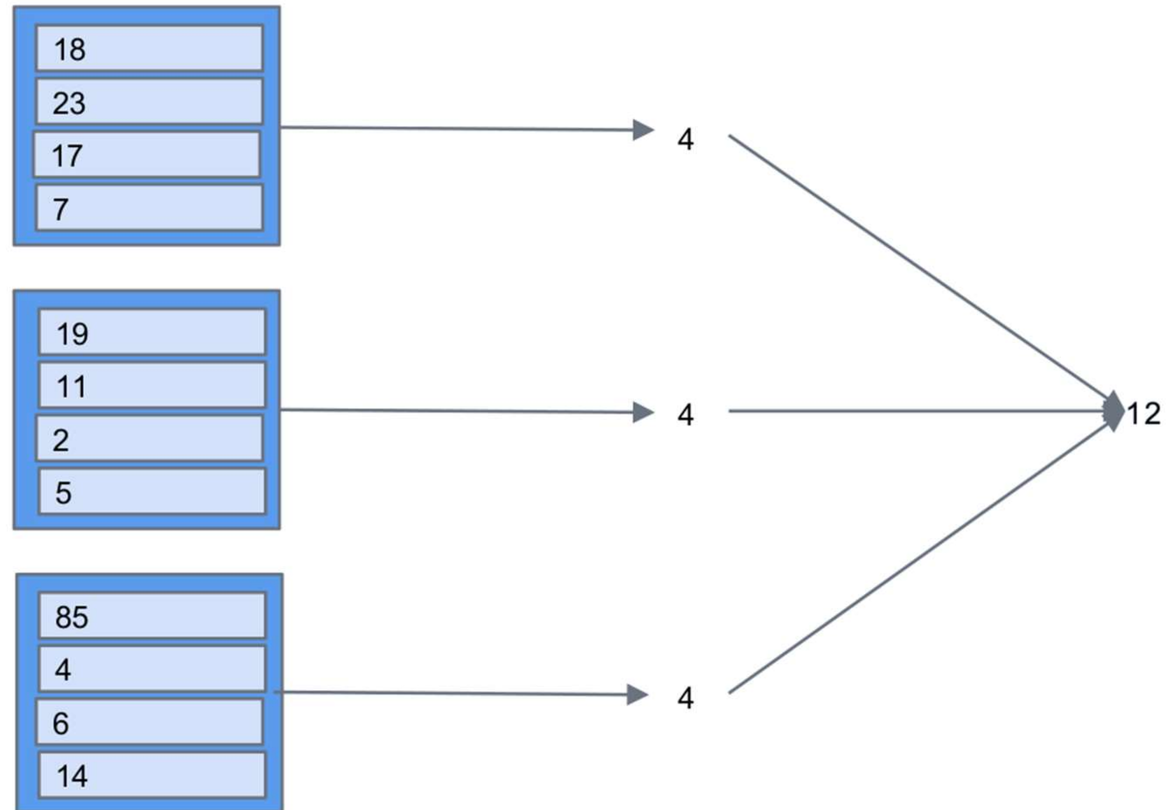
CAMBRIDGE UNIVERSITY PRESS

- **Actions**: Return a result to the driver node

Action	Meaning
`take(n)`	Returns a list of the first *n* elements of an RDD.
`takeOrdered(n, key=func)`	Returns *n* elements in ascending order or in the order determined by the optional function `func`.
`foreach(func)`	Applies the function `func` to each element of the RDD. It doesn't return anything. It could be useful to insert stuff in a database

RDDs – Actions: count()

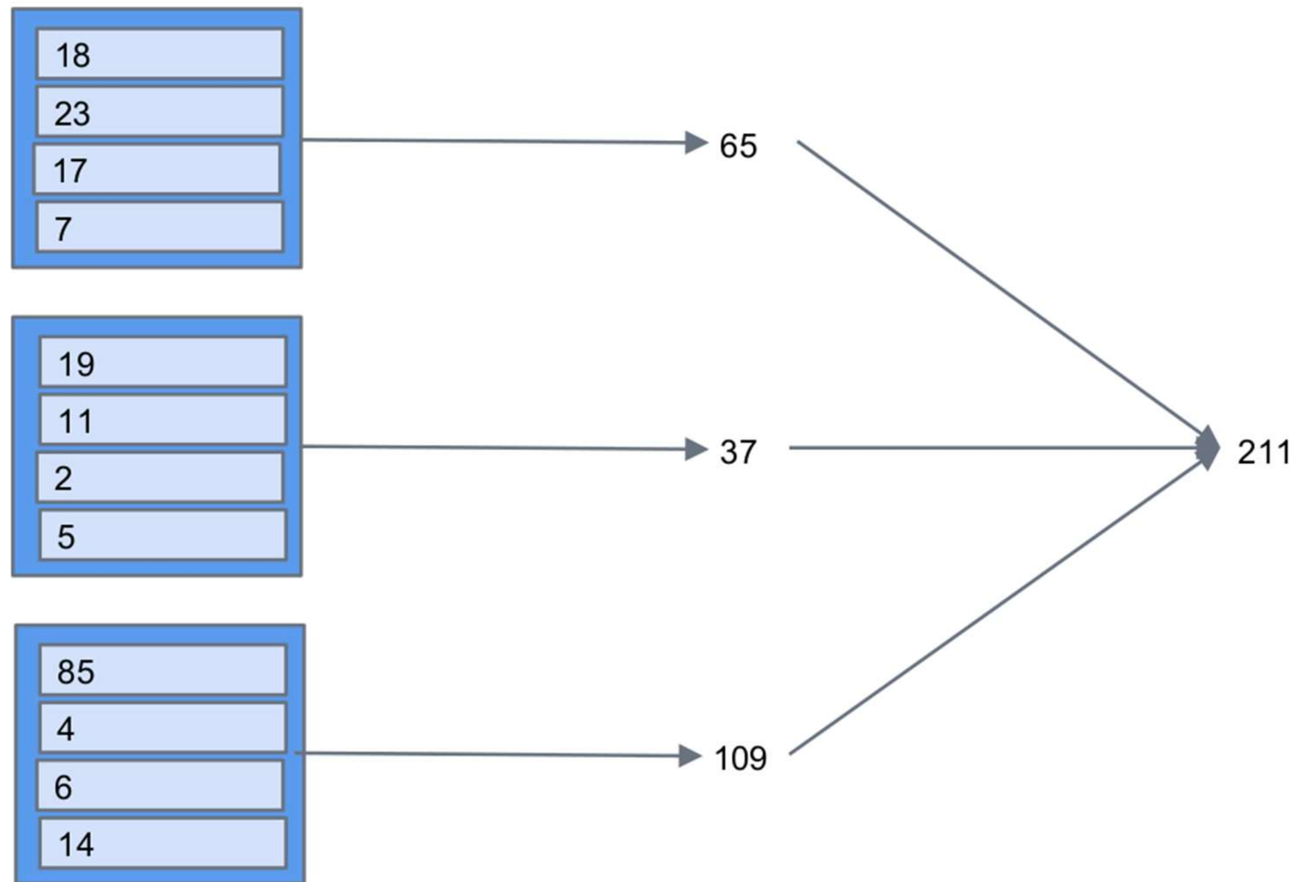
CAMBRIDGE UNIVERSITY PRESS



Source: Dirk Van den Poel. Spark: The new kid on the block (2014)

Actions: reduce(add)

CAMBRIDGE UNIVERSITY PRESS



RDDs – Actions

CAMBRIDGE UNIVERSITY PRESS

- **Actions**

```
>>> rdd = sc.parallelize([1, 2, 3])
>>> rdd.reduce(lambda a, b: a * b)
Value: 6

>>> rdd = sc.parallelize([1, 2, 3])
>>> rdd.take(2)
Value: [1, 2] # as a list

>>> rdd.collect()
Value: [1, 2, 3] # as a list

>>> rdd = sc.parallelize([5, 3, 1, 2])
>>> rdd.takeOrdered(3, lambda s: -1 * s)
Value: [5, 3, 2] #  as a list
```

RDDs – Actions

CAMBRIDGE UNIVERSITY PRESS

- **Actions**

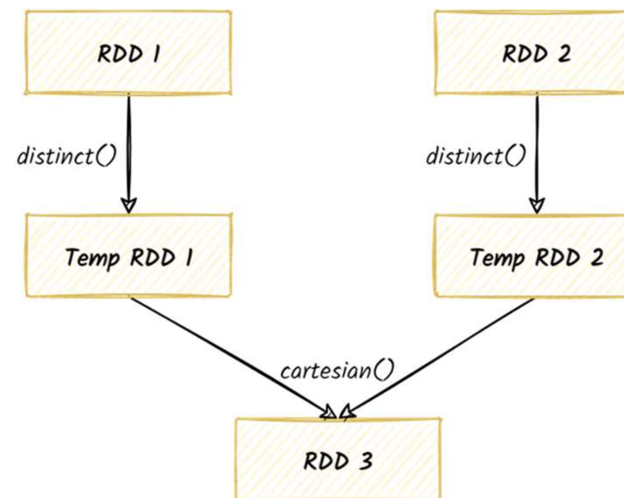
```
badLinesRDD = inputRDD.filter(lambda x: ["error", "warning"] in x)
print(f"This file has {badLinesRDD.count()} lines with problems")

print("First 10 lines: ")
for line in badLinesRDD.take(10):
    print(line)
```

RDDs - Lineage

CAMBRIDGE UNIVERSITY PRESS

- **RDD Lineage**: keeps track of all used transformations to provide tolerance to errors

```
rdd3 = rdd1.distinct().cartesian(rdd2.distinct())
```



Lineage Graph

Take-home Message

CAMBRIDGE UNIVERSITY PRESS

- Spark is an efficient data processing engine
  - Entry point: `SparkContext`
- RDDs is a distributed data abstraction
  - Immutable
  - Fault-tolerant
- Operations with RDDs: Basic Transformations and Actions

What's next?

  - More about Spark

Chapter 4

Spark I

© Isaac Triguero and Mikel Galar