MC855 - Projeto em Sistemas de Computação

Apache Spark

Islene Calciolari Garcia

Instituto de Computação - Unicamp

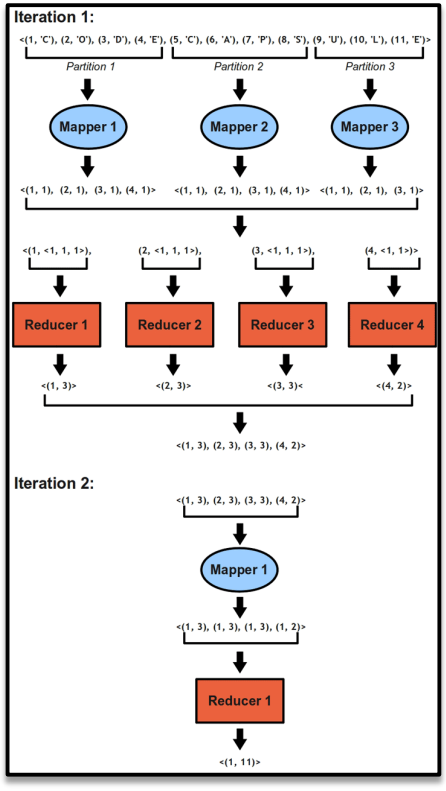
Primeiro Semestre de 2016

Sumário

MapReduce: A Programming Model

- **MapReduce: Simplified Data Processing on Large Clusters** *(published 2004)*

- **Parallel and Distributed Algorithm:**
- Data Locality
- Fault Tolerance
- Linear Scalability



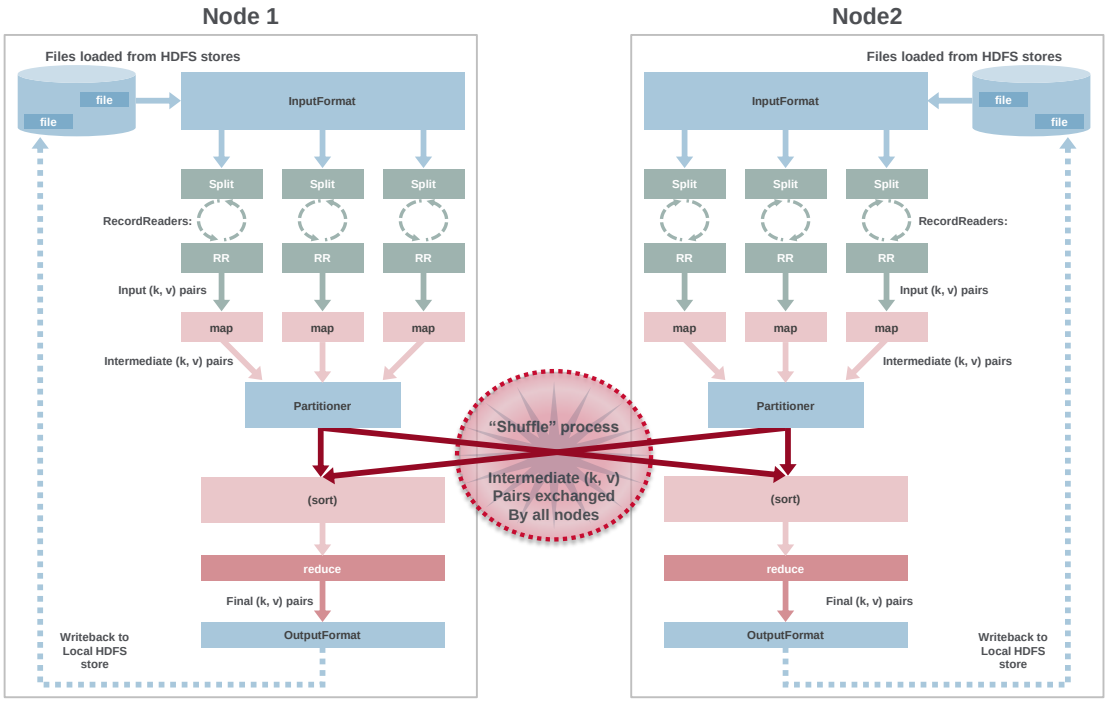
Fonte: Carol McDonald: An Overview of Apache Spark

MapReduce Basics

- Foundational model is based on a distributed file system
  - Scalability and fault-tolerance
- Map
  - Loading of the data and defining a set of keys
    - Many use cases do not utilize a reduce task
- Reduce
  - Collects the organized key-based data to process and output
- Performance can be tweaked based on known details of your source files and cluster shape (size, total number)

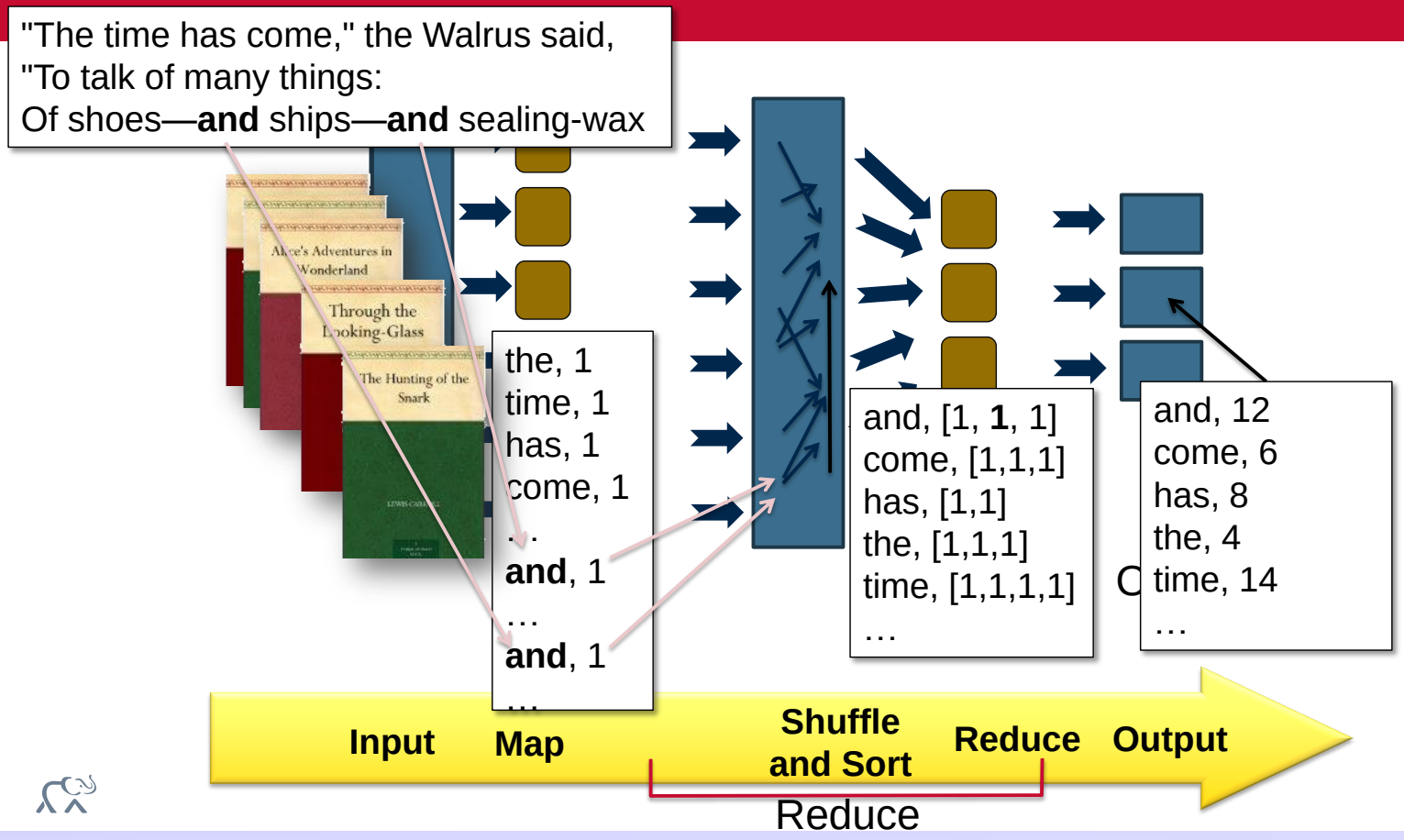
Fonte: Carol McDonald: An Overview of Apache Spark

MapReduce Execution and Data Flow



Fonte: Carol McDonald: An Overview of Apache Spark

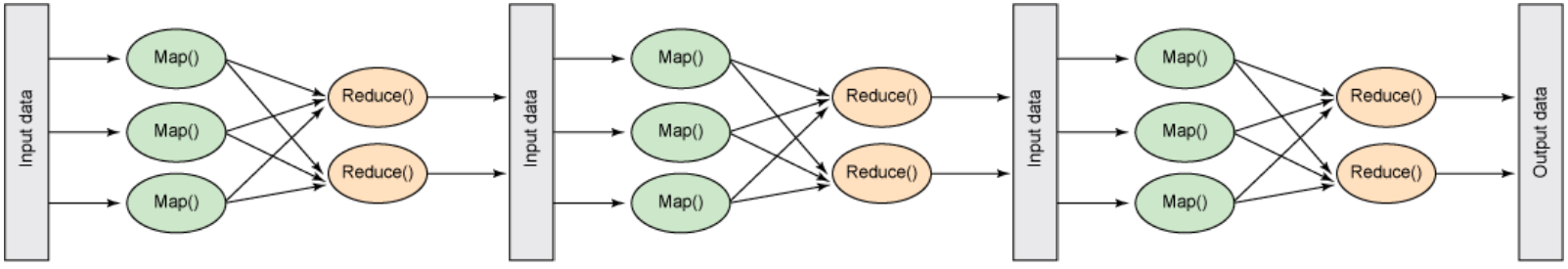
MapReduce Example: Word Count



Fonte: Carol McDonald: An Overview of Apache Spark

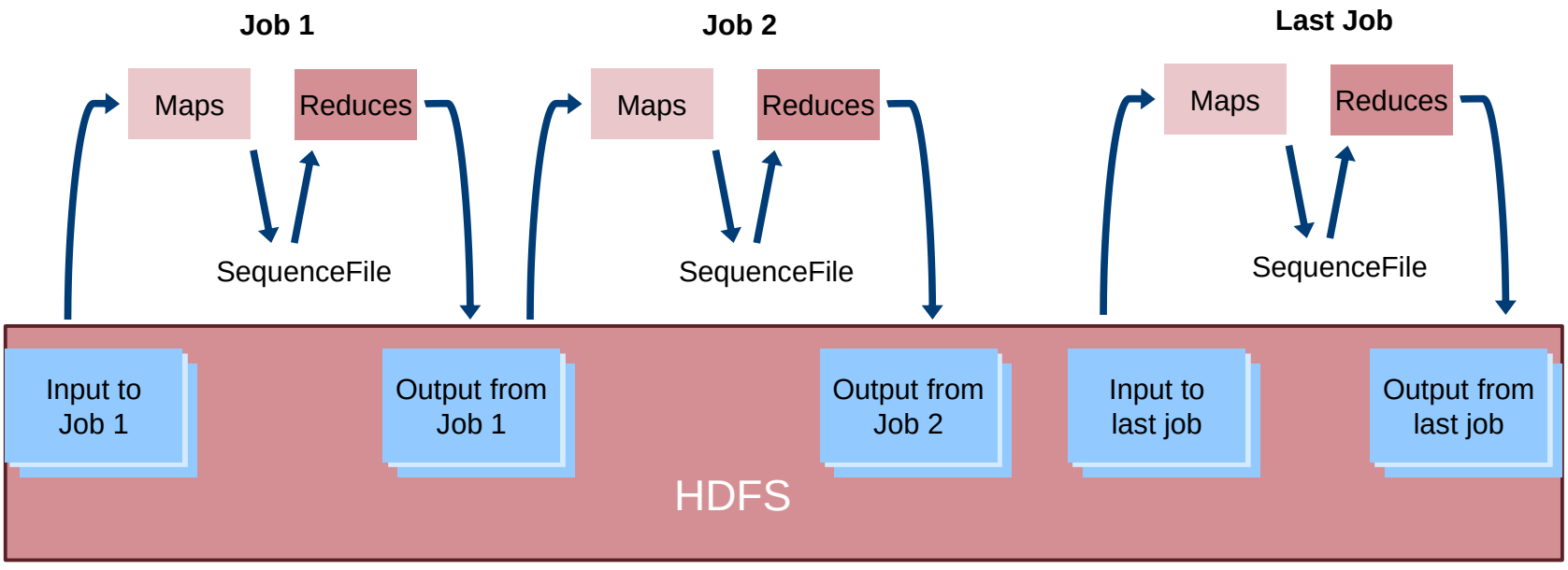
MapReduce Processing Model

- Define mappers
- Shuffling is automatic
- Define reducers
- For complex work, chain jobs together
  - Use a higher level language or DSL that does this for you



Fonte: Carol McDonald: An Overview of Apache Spark

Typical MapReduce Workflows



Fonte: Carol McDonald: An Overview of Apache Spark

MapReduce: The Good

- Built in fault tolerance
- Optimized IO path
- Scalable
- Developer focuses on Map/Reduce, not infrastructure
- simple? API

Fonte: Carol McDonald: An Overview of Apache Spark

MapReduce: The Bad

- Optimized for disk IO
  - Doesn’t leverage memory well
  - **Iterative** algorithms go through disk IO path again and again
- Primitive API
  - simple abstraction
  - Key/Value in/out
  - basic things like join
    - require extensive code
- Result often many files that need to be combined appropriately

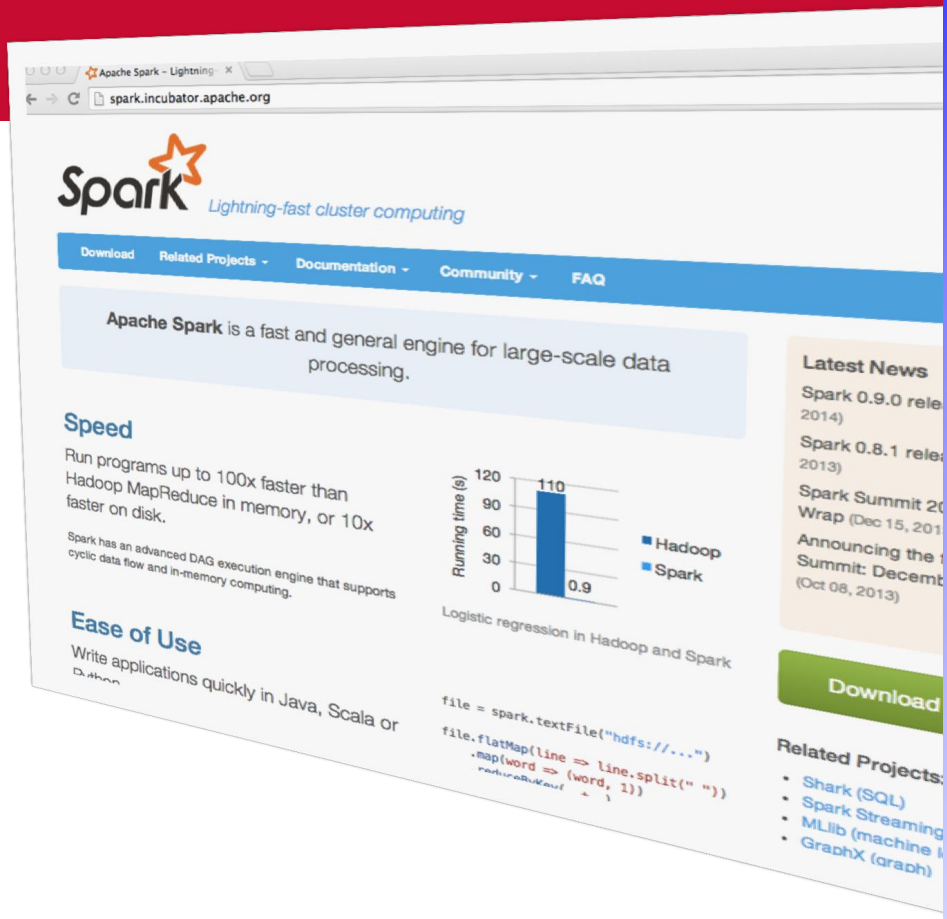
Fonte: Carol McDonald: An Overview of Apache Spark

Apache Spark

Fonte: Carol McDonald: An Overview of Apache Spark

Apache Spark

- Originally developed in 2009 in UC Berkeley's AMP Lab
- Fully open sourced in 2010 – now a Top Level Project at the Apache Software Foundation



spark.apache.org
github.com/apache/spark
user@spark.apache.org

Fonte: Carol McDonald: An Overview of Apache Spark

Spark: Fast Big Data

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

2-5× less code

- **Fast to Run**
  - General execution graphs
  - In-memory storage

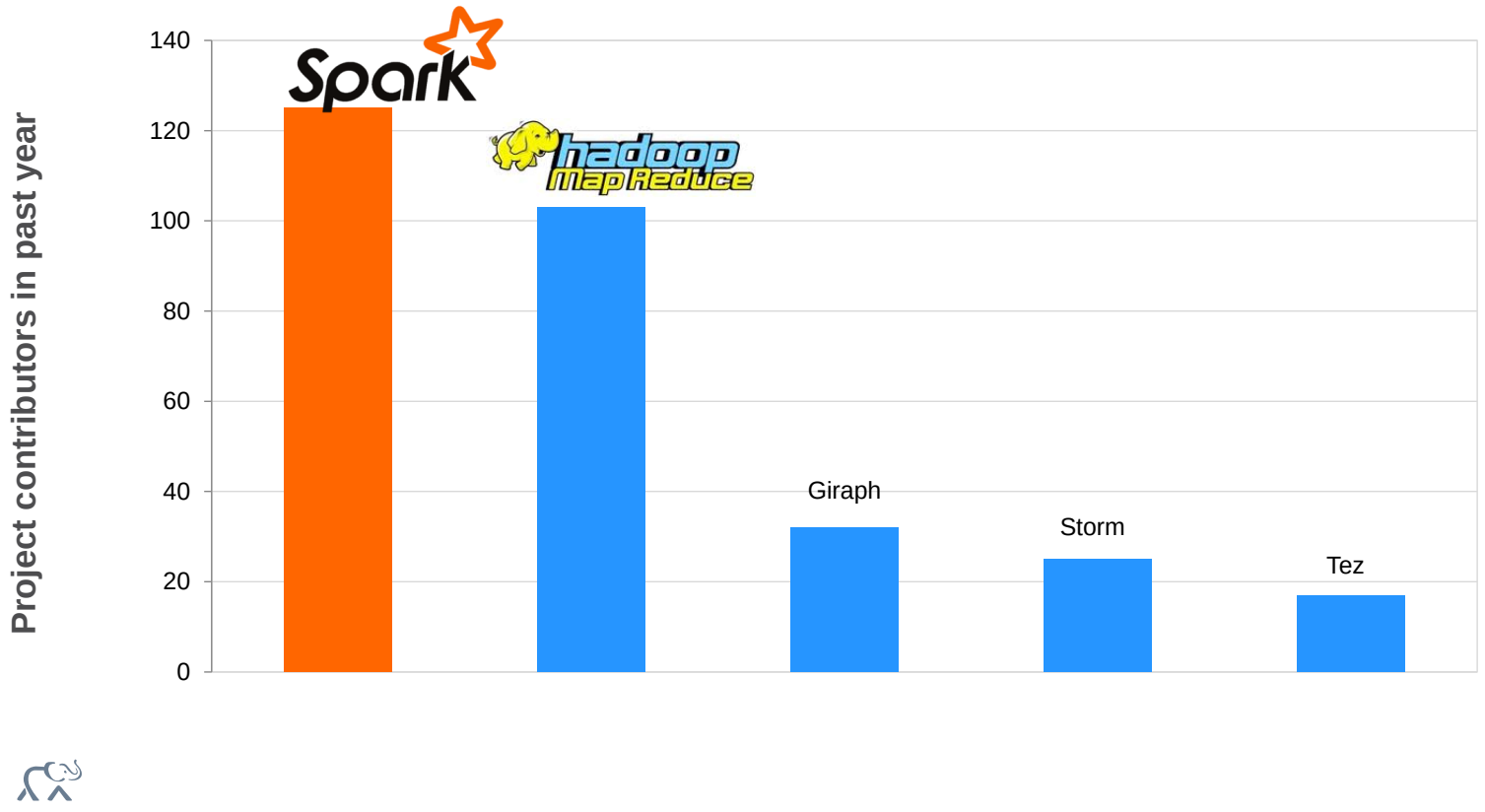
Fonte: Carol McDonald: An Overview of Apache Spark

The Spark Community

amplab UC BERKELEY
DATABRICKS
YAHOO!
amazon web services
wandisco
intel
MAPR
阿里巴巴 Alibaba.com
Hortonworks
CLOUD PHYSICS
guavus Go Decisively
AdMobius
ClearStory Data Now You See It
IBM
TAGGED
4B
stratio
CONVIVA
Adobe
bizo
webtrends
ADATAO BIG DATA INTELLIGENCE
sharethrough
HHMI janelia farm research campus
OOYALA
quantiFind
tuplejump
cloudera

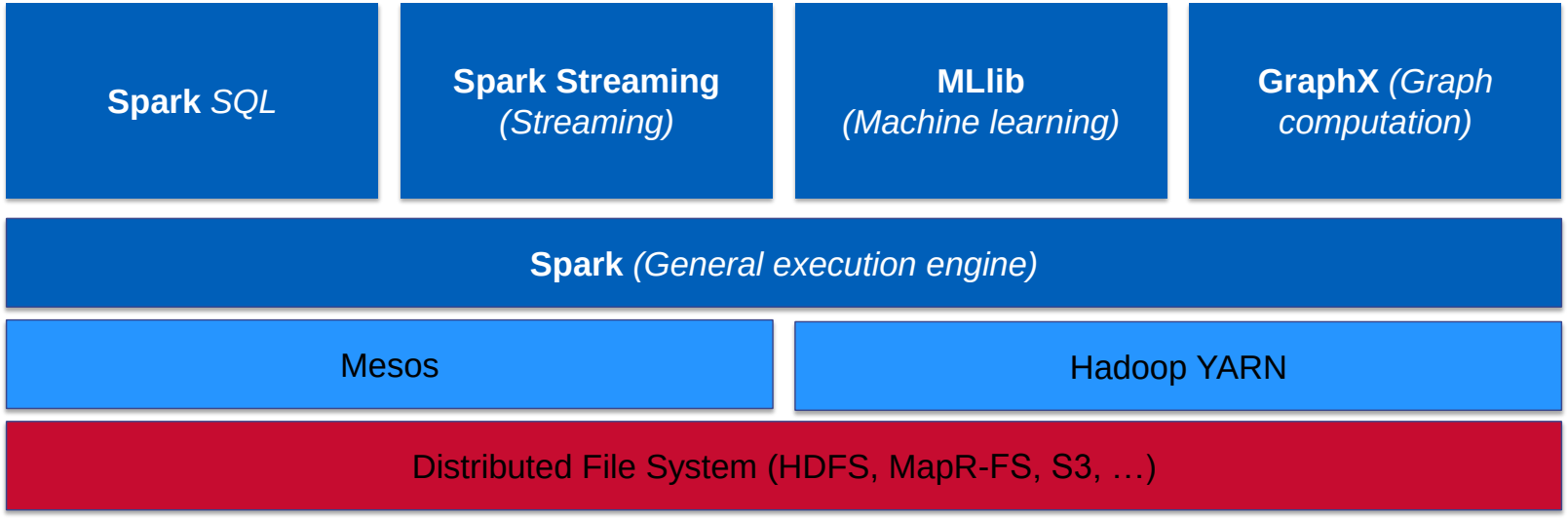
Fonte: Carol McDonald: An Overview of Apache Spark

Spark is the Most Active Open Source Project in Big Data



Fonte: Carol McDonald: An Overview of Apache Spark

Unified Platform



Fonte: Carol McDonald: An Overview of Apache Spark

Spark Use Cases

- Iterative Algorithms on on large amounts of data
- Anomaly detection
- Classification
- Predictions
- Recommendations

Fonte: Carol McDonald: An Overview of Apache Spark

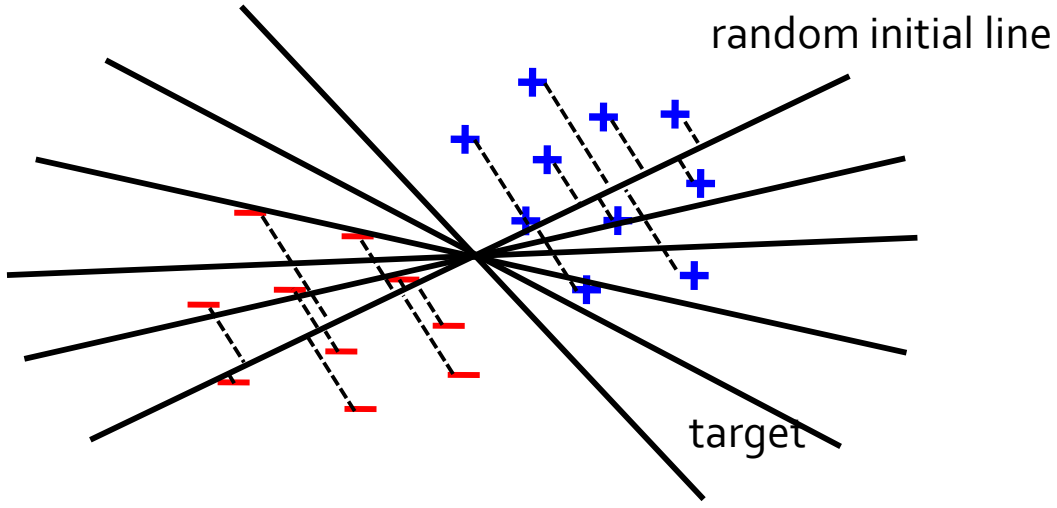
Why Iterative Algorithms

- Algorithms that need iterations
  - Clustering (K-Means, Canopy, …)
  - Gradient descent (e.g., Logistic Regression, Matrix Factorization)
  - Graph Algorithms (e.g., PageRank, Line-Rank, components, paths, reachability, centrality,)
  - Alternating Least Squares ALS
  - Graph communities / dense sub-components
  - Inference (believe propagation)
  - ...

Fonte: Carol McDonald: An Overview of Apache Spark

Example: Logistic Regression

- Goal: find best line separating two sets of points



Fonte: Carol McDonald: An Overview of Apache Spark

Logistic Regression

```
data = spark.textFile(...).map(readPoint).cache()

w = numpy.random.rand(D)

for i in range(iterations):
    gradient = data
     .map(lambda p: (1 / (1 + exp(-p.y * w.dot(p.x))))
          * p.y * p.x)
     .reduce(lambda x, y: x + y)
    w -= gradient

print “Final w: %s” % w
```

Iteration!

Fonte: Carol McDonald: An Overview of Apache Spark

Data Sources

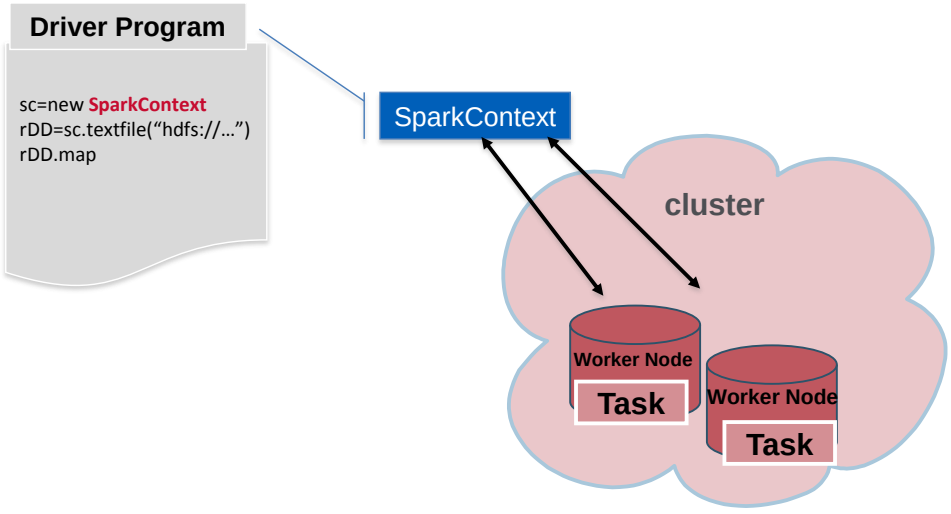
- Local Files
  - file:///opt/httpd/logs/access_log
- S3
- Hadoop Distributed Filesystem
  - Regular files, sequence files, any other Hadoop InputFormat
- HBase
- other NoSQL data stores

Fonte: Carol McDonald: An Overview of Apache Spark

How Spark Works

Fonte: Carol McDonald: An Overview of Apache Spark

Spark Programming Model



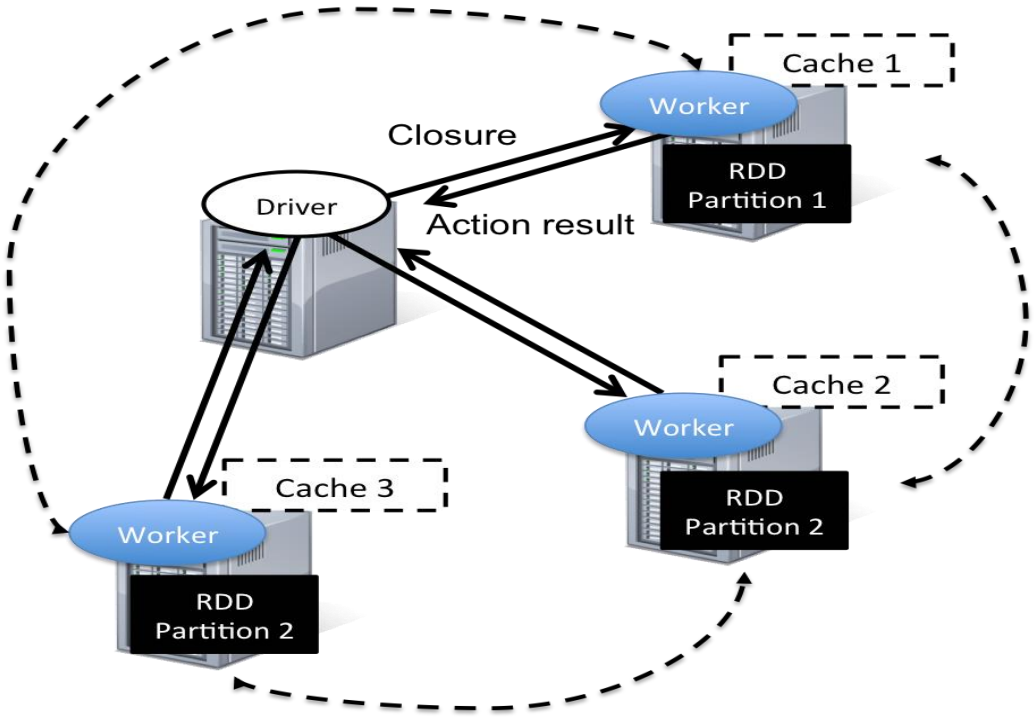
Fonte: Carol McDonald: An Overview of Apache Spark

Resilient Distributed Datasets (RDD)

Spark revolves around RDDs

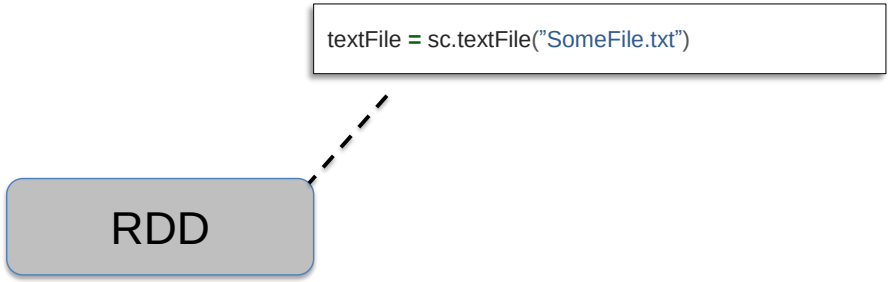
- Fault-tolerant
- read only collection of elements
- operated on in parallel
- Cached in memory
- Or on disk

http://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf



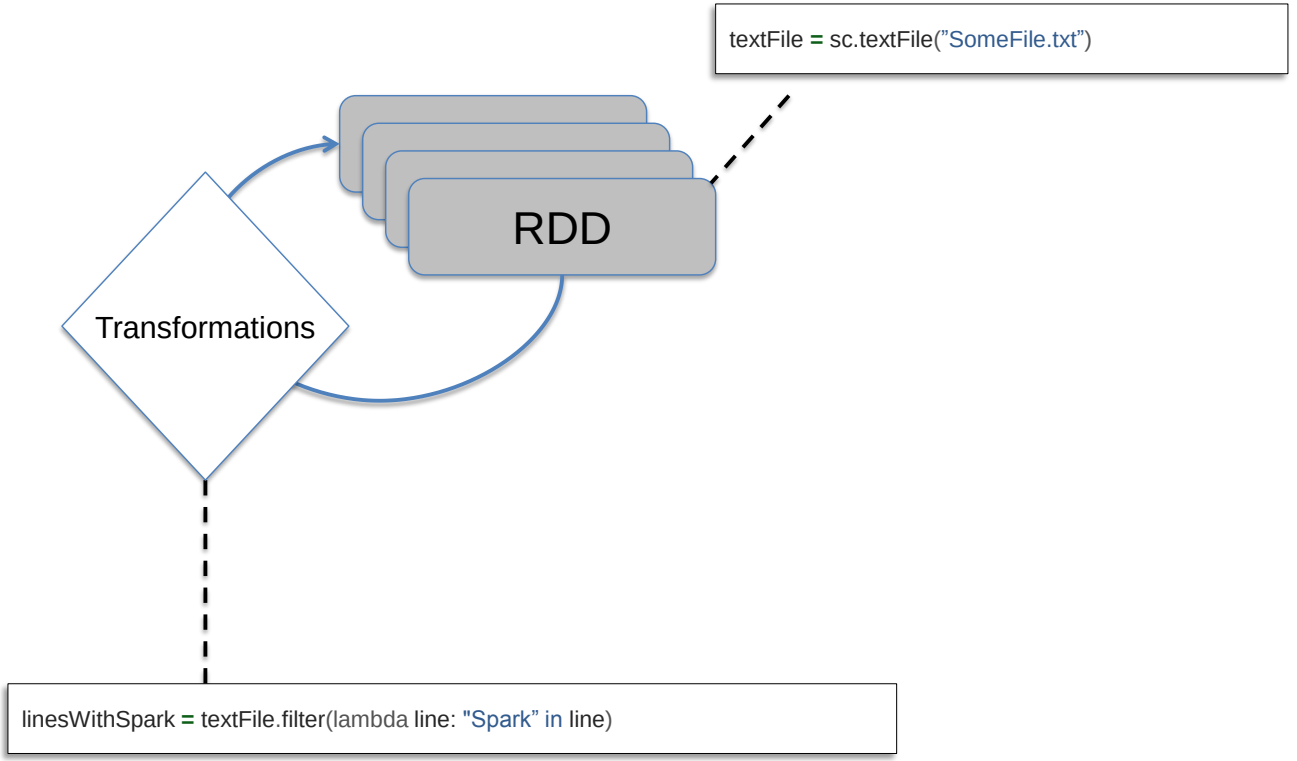
Fonte: Carol McDonald: An Overview of Apache Spark

Working With RDDs



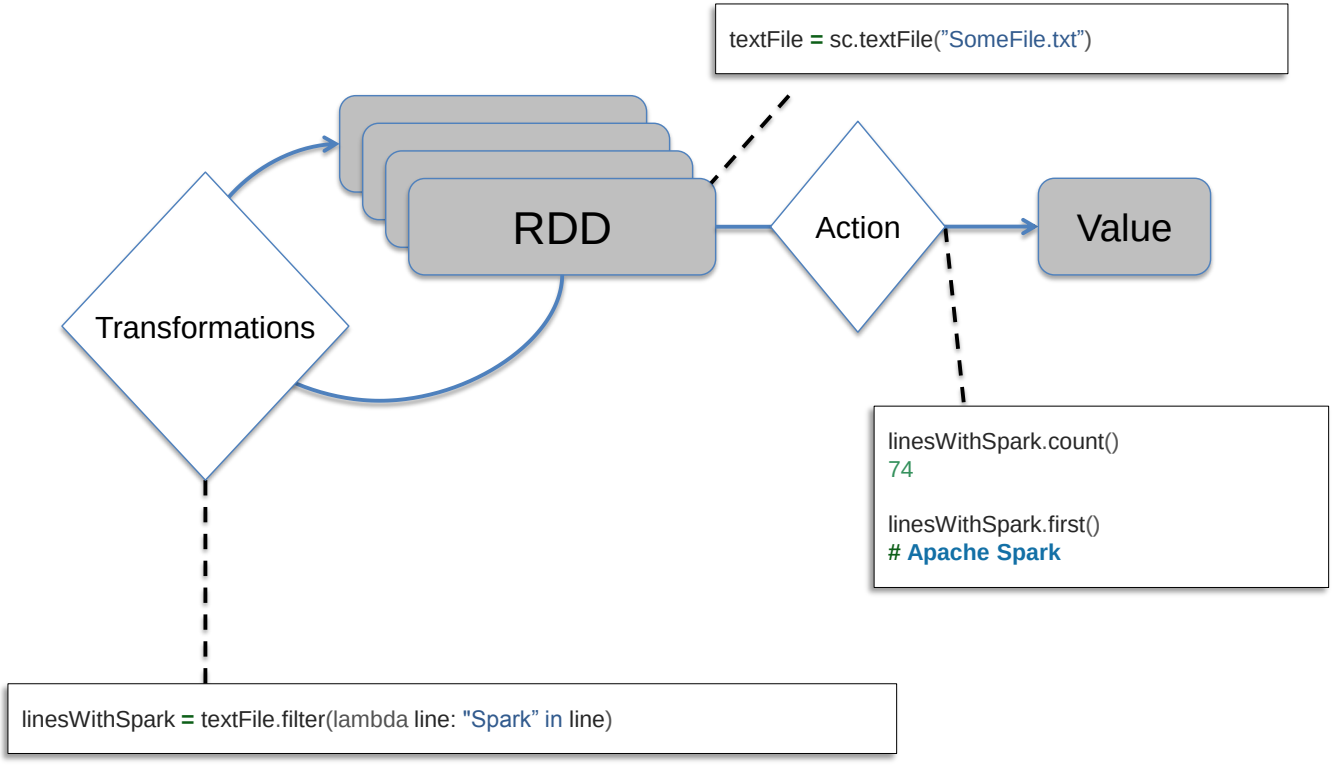
Fonte: Carol McDonald: An Overview of Apache Spark

Working With RDDs



Fonte: Carol McDonald: An Overview of Apache Spark

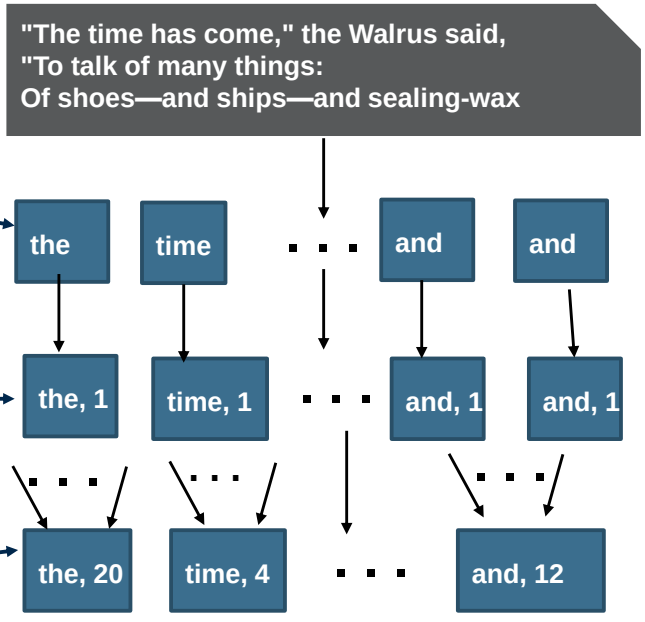
Working With RDDs



Fonte: Carol McDonald: An Overview of Apache Spark

Example Spark Word Count in Java

```
JavaRDD<String> input = sc.textFile(inputFile);
// Split each line into words
JavaRDD<String> words = input.flatMap(
  new FlatMapFunction<String, String>() {
    public Iterable<String> call(String x) {
      return Arrays.asList(x.split(" "));
   }});
// Turn the words into (word, 1) pairs
JavaPairRDD<String, Integer> word1s== words.mapToPair(
  new PairFunction<String, String, Integer>(){
    public Tuple2<String, Integer> call(String x){
      return new Tuple2(x, 1);
   }});
// reduce add the pairs by key to produce counts
JavaPairRDD<String, Integer> counts =word1s.reduceByKey(
  new Function2<Integer, Integer, Integer>(){
    public Integer call(Integer x, Integer y){
      return x + y;
   }});
```



Fonte: Carol McDonald: An Overview of Apache Spark

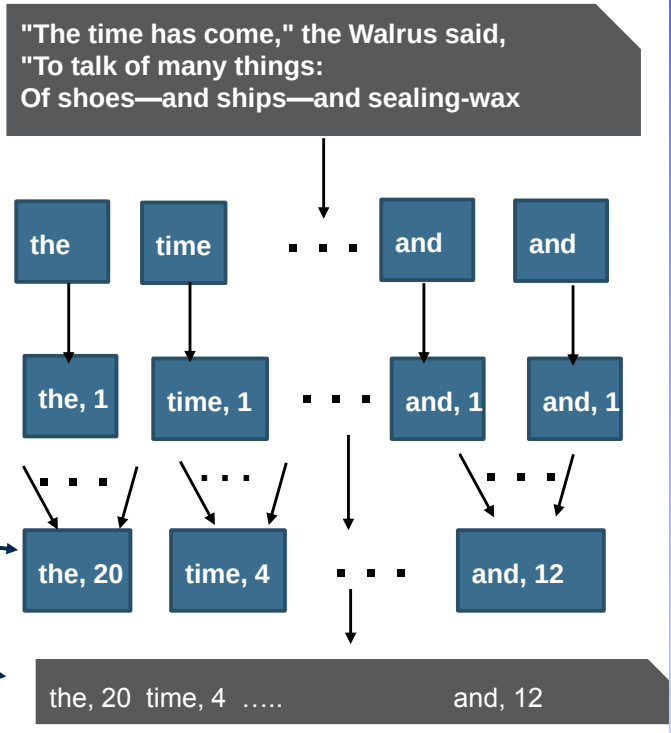
Example Spark Word Count in Scala

```
// Load our input data.
val input =  sc.textFile(inputFile)

// Split it up into words.
val words = input.flatMap(line => line.split(" "))

// Transform into pairs and count.
val counts = words
      .map(word => (word, 1))
      .reduceByKey{case (x, y) => x + y}

// Save the word count back out to a text file,
counts.saveAsTextFile(outputFile)
```



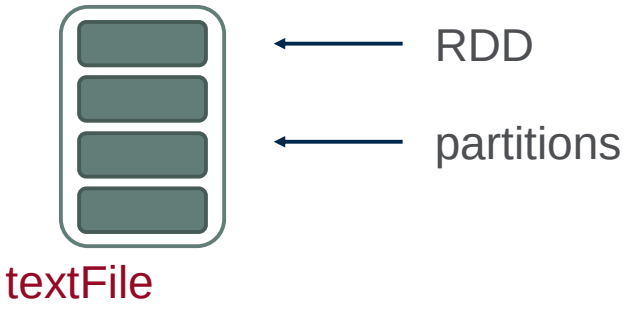
Fonte: Carol McDonald: An Overview of Apache Spark

Example Spark Word Count in Scala

```
// Load  input data.
val input =  sc.textFile(inputFile)
```

HadoopRDD

MapPartitionsRDD



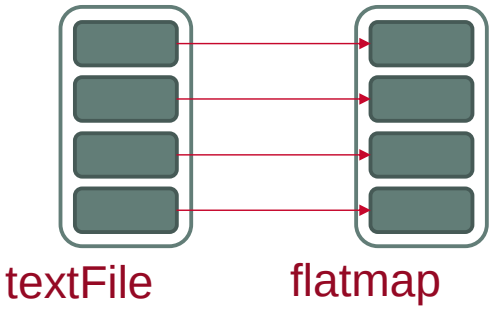
Fonte: Carol McDonald: An Overview of Apache Spark

Example Spark Word Count in Scala

```
// Load our input data.
val input =  sc.textFile(inputFile)
// Split it up into words.
val words = input.flatMap(line => line.split(" "))
```

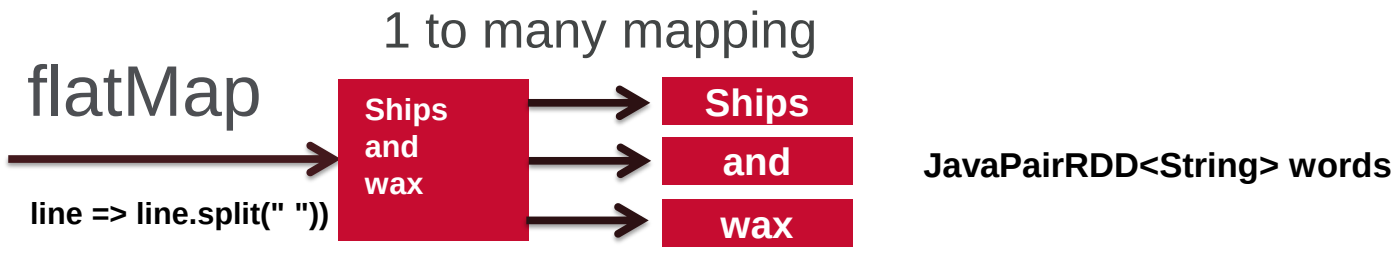
HadoopRDD
MapPartitionsRDD

MapPartitionsRDD



Fonte: Carol McDonald: An Overview of Apache Spark

FlatMap



Fonte: Carol McDonald: An Overview of Apache Spark

Example Spark Word Count in Scala

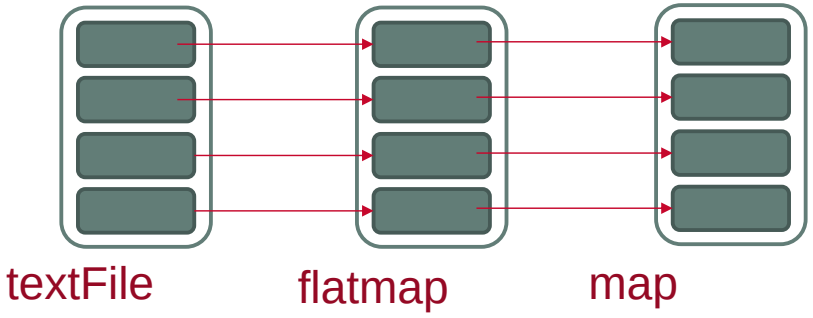
```
val input =  sc.textFile(inputFile)
val words = input.flatMap(line => line.split(" "))
// Transform into pairs
val counts = words.map(word => (word, 1))
```

HadoopRDD

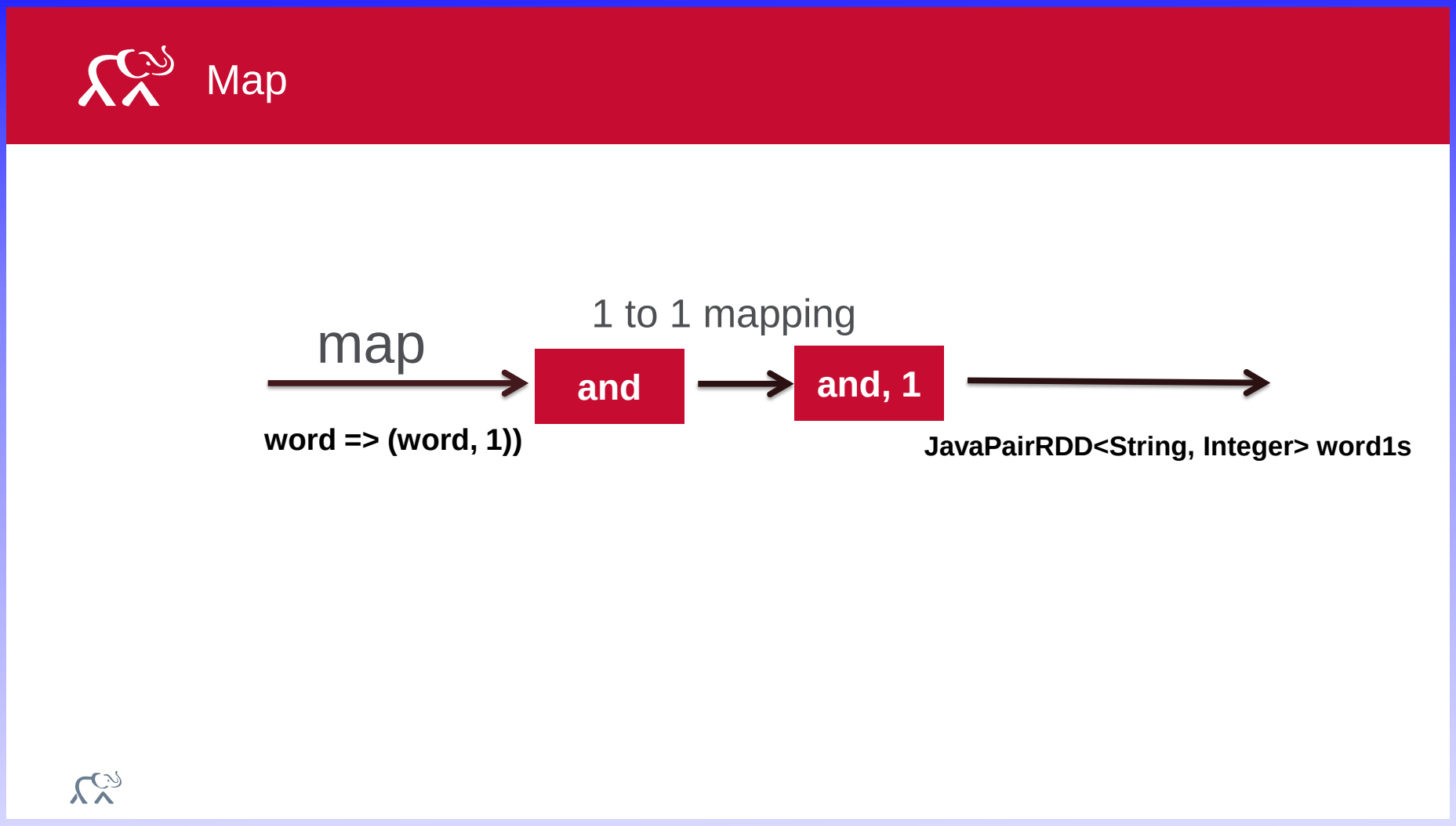
MapPartitionsRDD

MapPartitionsRDD

MapPartitionsRDD



Fonte: Carol McDonald: An Overview of Apache Spark



Fonte: Carol McDonald: An Overview of Apache Spark

Example Spark Word Count in Scala

```
val input =  sc.textFile(inputFile)
val words = input.flatMap(line => line.split(" "))
val counts = words
      .map(word => (word, 1))
      .reduceByKey{case (x, y) => x + y}
```

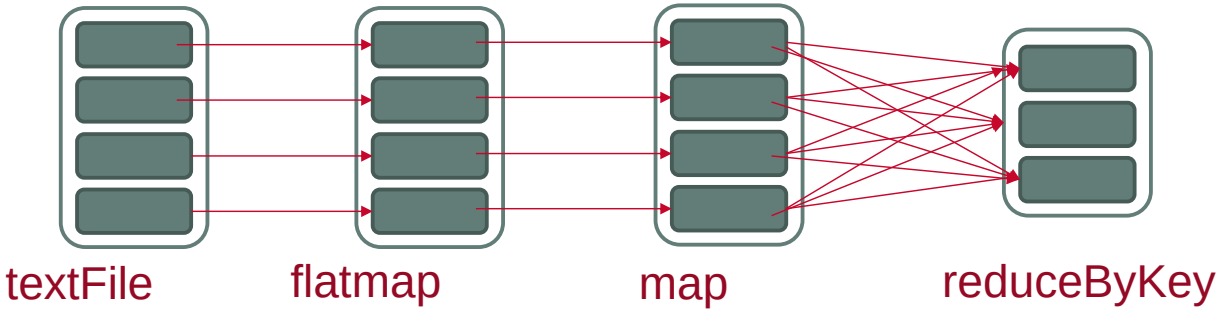
HadoopRDD

MapPartitionsRDD

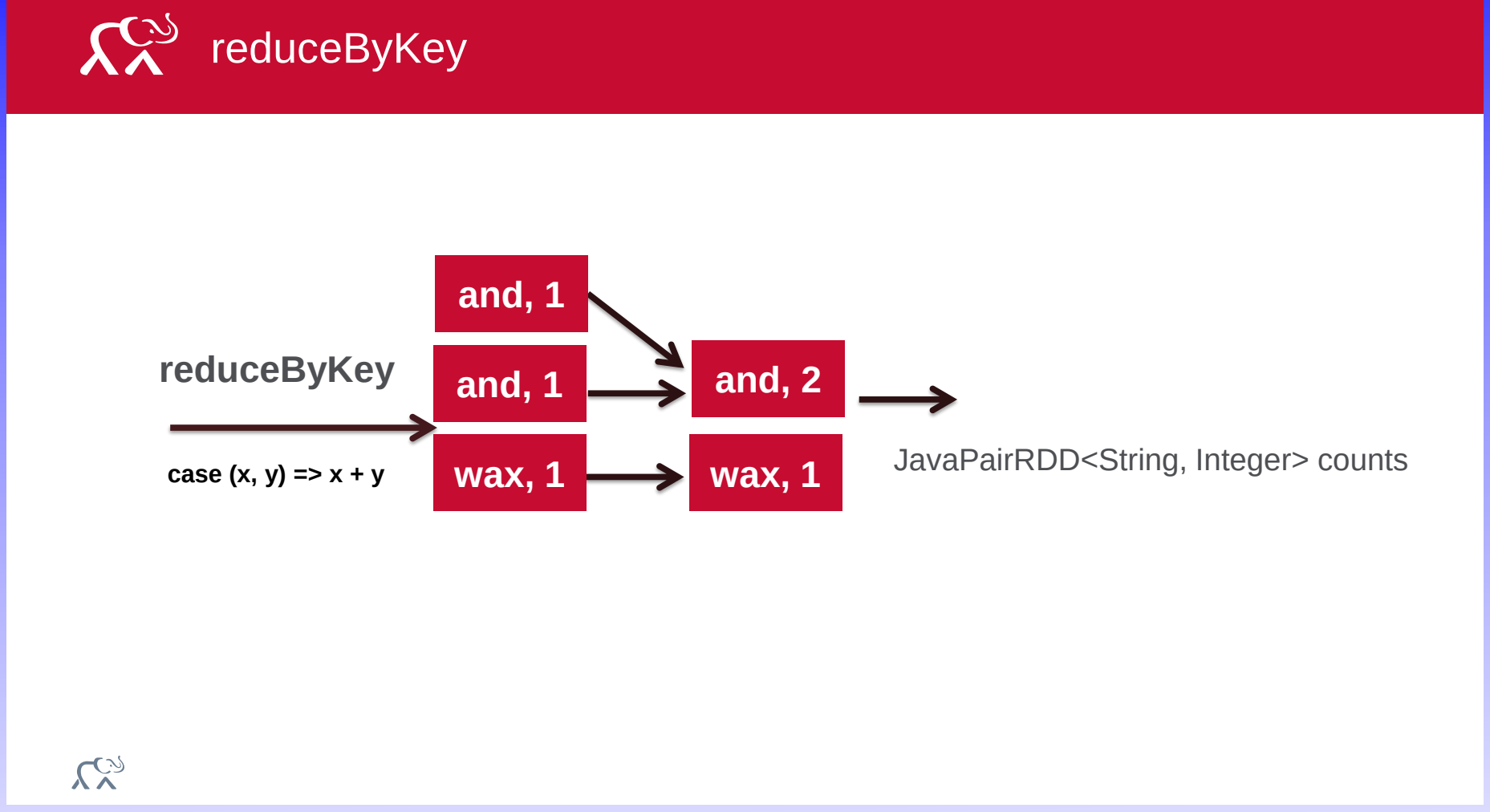
MapPartitionsRDD

MapPartitionsRDD

ShuffledRDD



Fonte: Carol McDonald: An Overview of Apache Spark



Fonte: Carol McDonald: An Overview of Apache Spark

Example Spark Word Count in Scala

```
val input =  sc.textFile(inputFile)
val words = input.flatMap(line => line.split(" "))
val counts = words
      .map(word => (word, 1))
      .reduceByKey{case (x, y) => x + y}
val countArray = counts.collect()
```

HadoopRDD

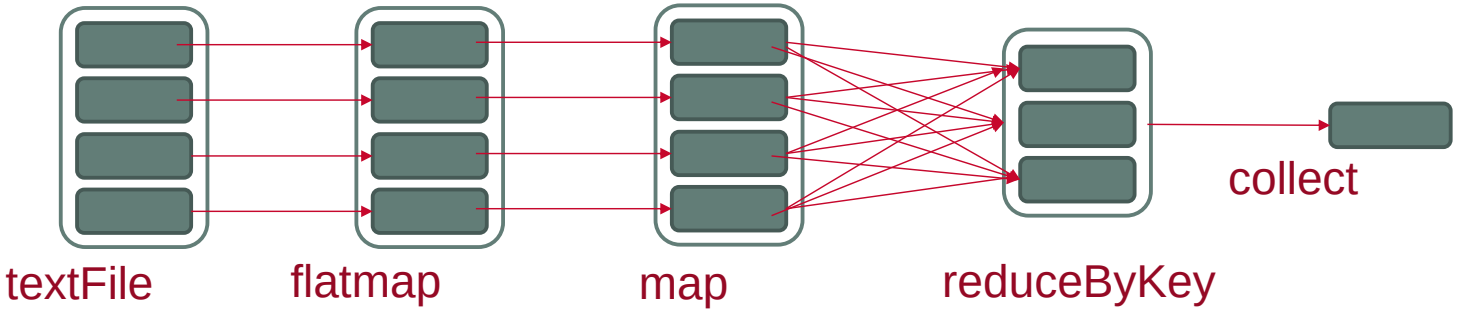
MapPartitionsRDD

MapPartitionsRDD

MapPartitionsRDD

ShuffledRDD

Array

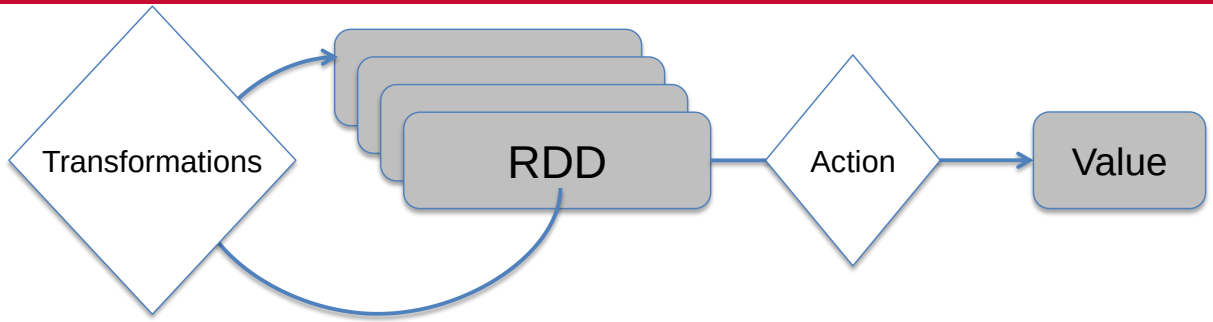


Fonte: Carol McDonald: An Overview of Apache Spark

Transformations and Actions

Fonte: Carol McDonald: An Overview of Apache Spark

RDD Transformations and Actions



Transformations
(define a new RDD)

map
filter
sample
union
groupByKey
reduceByKey
join
cache
...

Actions
(return a value)

reduce
collect
count
save
lookupKey
...

Fonte: Carol McDonald: An Overview of Apache Spark

Basic Transformations

```
>  nums = sc.parallelize([1, 2, 3])

# Pass each element through a function
>  squares = nums.map(lambda x: x*x)   // {1, 4, 9}

# Keep elements passing a predicate
>  even = squares.filter(lambda x: x % 2 == 0) // {4}

# Map each element to zero or more others
>  nums.flatMap(lambda x: => range(x))
    >  # => {0, 0, 1, 0, 1, 2}
```

Range object (sequence of numbers 0, 1, …, x-1)

DATABRICKS

Fonte: Carol McDonald: An Overview of Apache Spark

Basic Actions

```
> nums = sc.parallelize([1, 2, 3])

# Retrieve RDD contents as a local collection
> nums.collect() # => [1, 2, 3]

# Return first K elements
> nums.take(2)   # => [1, 2]

# Count number of elements
> nums.count()   # => 3

# Merge elements with an associative function
> nums.reduce(lambda x, y: x + y)  # => 6

# Write elements to a text file
> nums.saveAsTextFile("hdfs://file.txt")
```

DATABRICKS

Fonte: Carol McDonald: An Overview of Apache Spark

There’s a lot more !

Fonte: Carol McDonald: An Overview of Apache Spark

Projeto 2

Neste projeto vamos comparar o uso do Hadoop MapReduce com o Apache Spark.

- Escolha uma aplicação
- Estude o código e proponha uma pequena alteração
- O modelo de transformações e ações foi mais adequado?

Referências

- http://spark.apache.org/
- An Overview of Apache Spark, Carol McDonald
- Clash of Titans: MapReduce vs. Spark for Large Scale Data Analytics, Juwei Shi e outros, IBM Research, China