INTRODUCTION TO SPARK WITH SCALA

Apache Spark Programing Model

Agenda

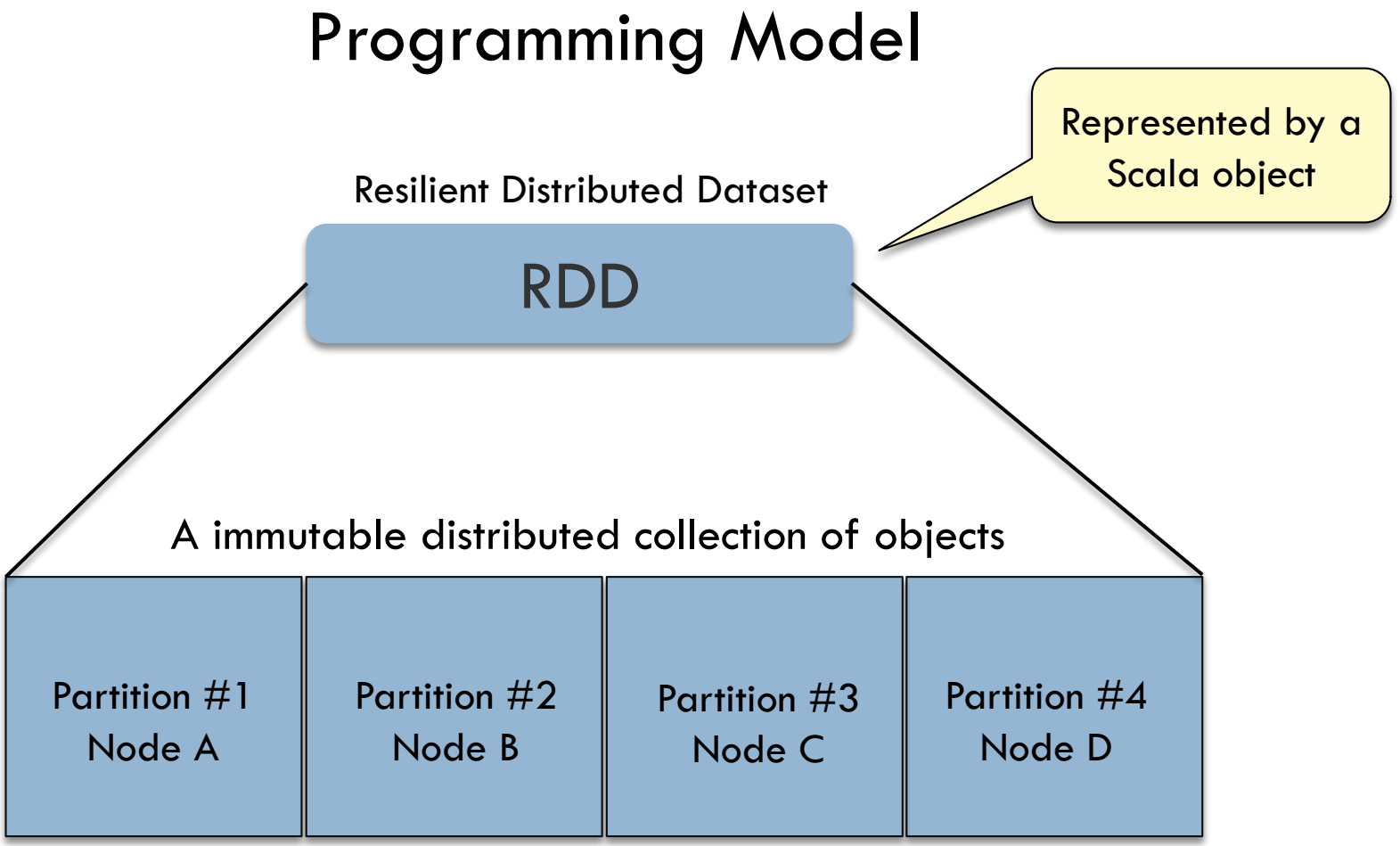
- Programming with RDD
- Working with RDD
- Working with Pair RDD
- In class exercises

Programming with RDD

Process distributed collections with functional operators, the same way you can for local ones

- Matie Zaharia

Programming with RDD

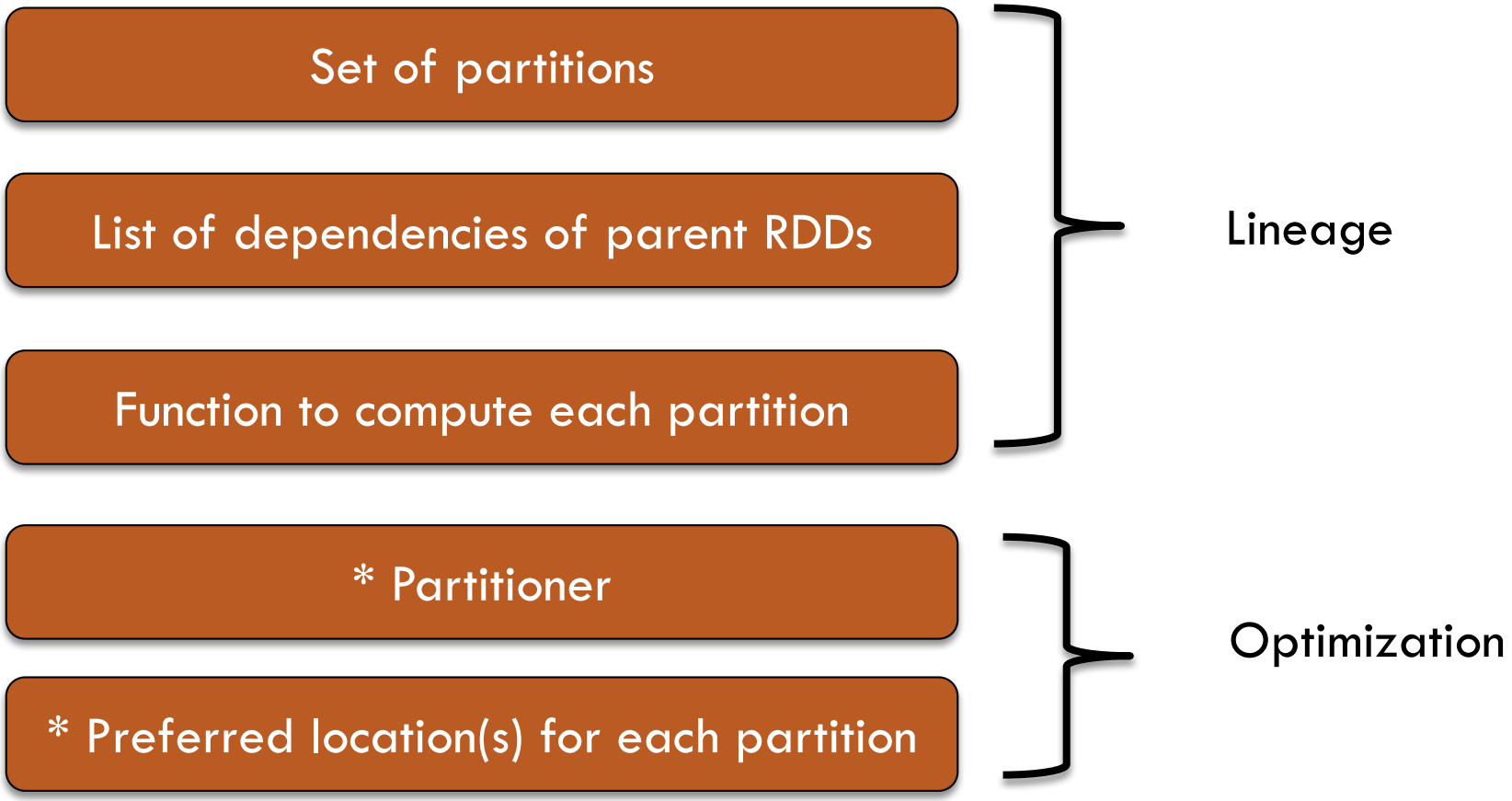


Programming with RDD

- What is an RDD?
  - A distributed collection of objects on disk
  - A distributed collection of objects in memory
  - A distributed collection of objects in Cassandra

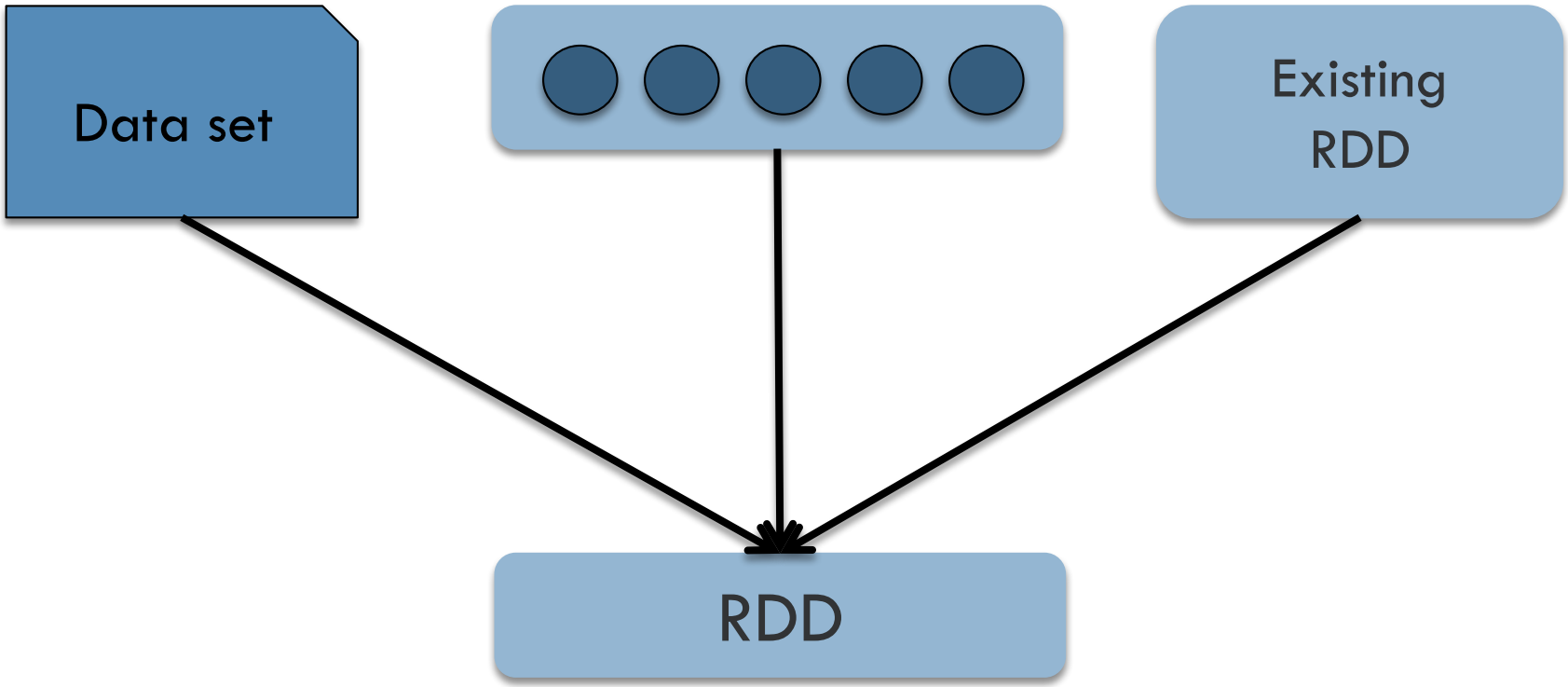
Programming with RDD

RDD is an interface (scientific answer)



Programming with RDD

Creating RDDs



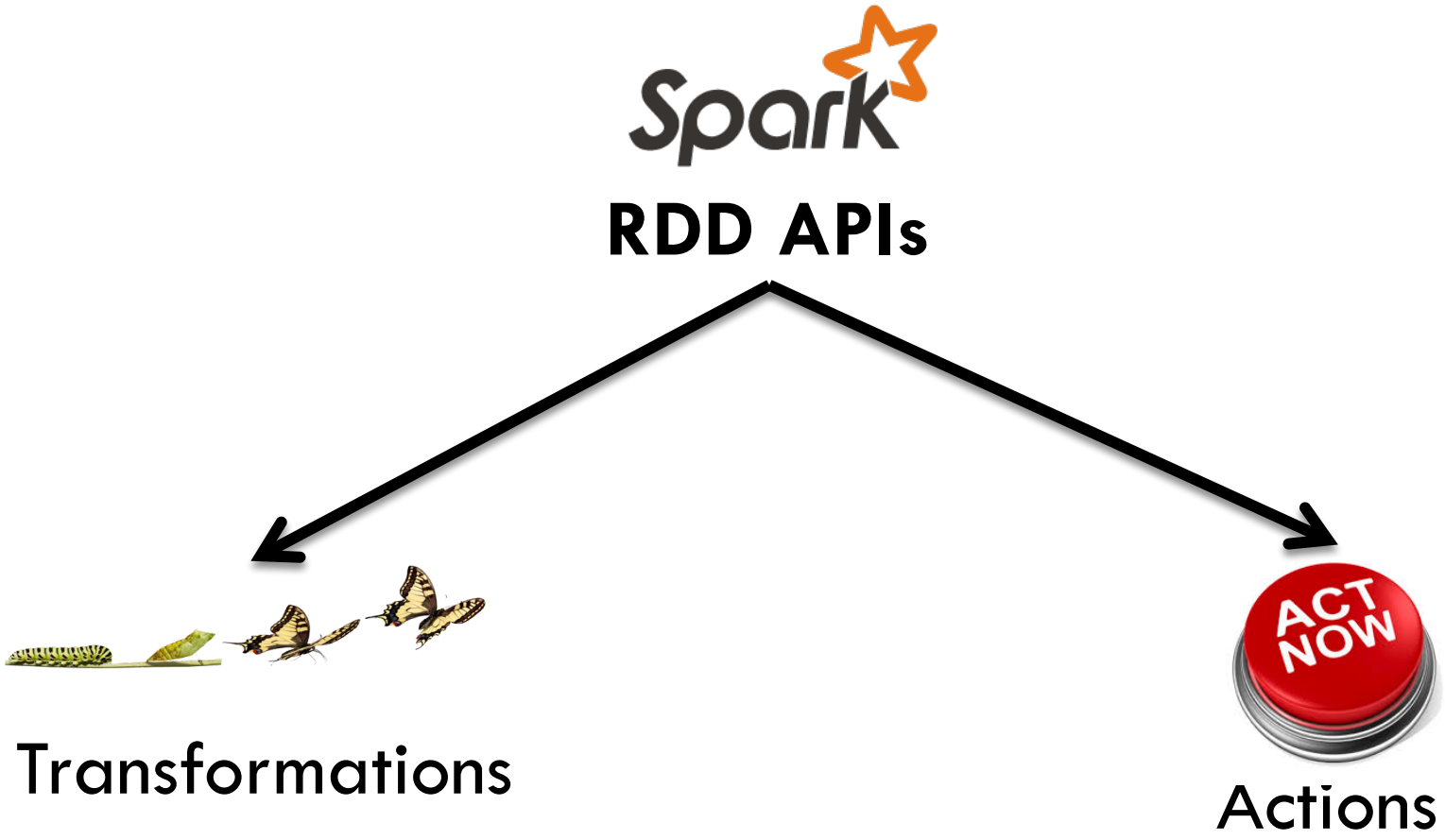
Programming with RDD


```
// from a specific file
val wordsRDD = sc.textFile("/path/to/wordlist.txt")

// all the files in the given directory
val wordsRDD = sc.textFile("/path/words")
```

```
// from in-memory collection of objects
// good for prototyping and learning purposes

val wordsRDD = sc.parallelize (List("Spark","is","cool"))
```

```
// from an existing RDD

val goodWordsRDD = wordsRDD.filter( .. )
```

Programming with RDD



Programming with RDD

- Transformation
  - Return a new RDD
  - Lazy evaluation
    - Record metadata about the request
    - Instructions for how to compute data
  - Most are on element-wise
- Actions
  - Return the result to the driver
  - Write to storage (HDFS, S3, local file system)
  - Kick off a computation (job)

Programming with RDD

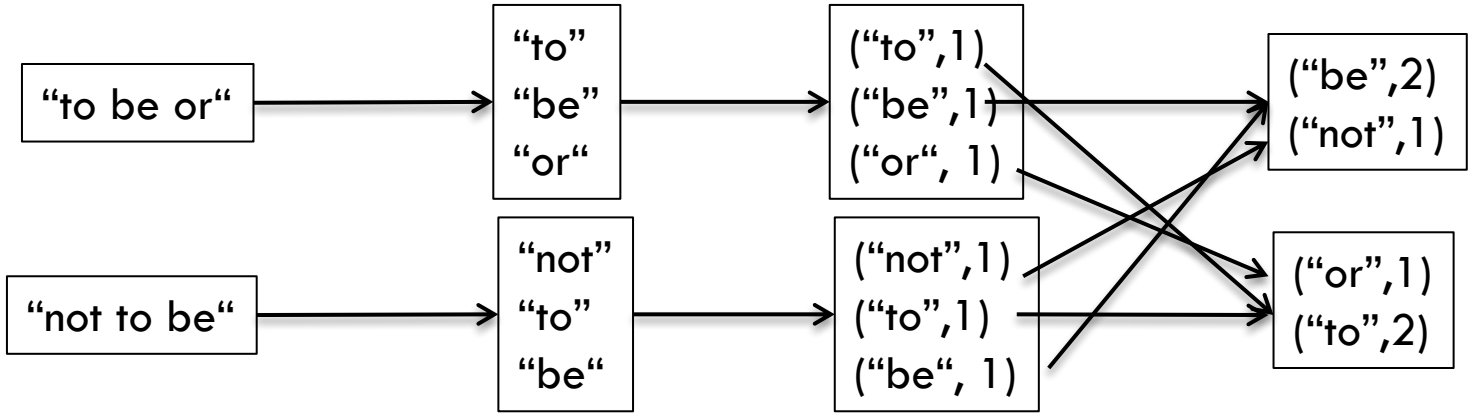
- RDD transformations
  - Data transformation, filtering,
  - Create a new RDD from existing one
    - Immutability
  - Lazy execution
    - Remember previous transformations
    - Builder pattern
    - Execute when an action is invoked

http://spark.apache.org/docs/latest/programming-guide.html#transformations

Programming with RDD

Word count example

```
// loading data file
val file = sc.textFile("README.md”)

val words = file.flatMap(l => l.split(" "))
val wordCountPairs = words.map(word => (word, 1))
val wordCountSum = wordCountPairs.reduceByKey(_ + _)

// store output
wordCountSum.saveAsTextFile(”/output")
```



Programming with RDD

Common Data Processing Operations

- Transformation
- Grouping
- Joining
- Filtering
- Aggregation
- Sorting
- Sampling
- Distinct
- Counting

Programming with RDD

Transformation	Description
map(func)	Passing each element through given function
filter(func)	Select those elements which func returns true
flatMap(func)	Each input item can be mapped to 0 or more output items
sample(repl, fraction, seed)	Sample a fraction of data
union(dataSet)	Create a union of two data sets
intersection(dataSet)	Create an intersection between two data sets
distinct([numTasks])	Distinct elements
subtract(dataSet)	Remove the contents of one RDD

http://spark.apache.org/docs/latest/programming-guide.html#transformations

Working with RDD

Actions compute a result or save to storage system

Action	Description
reduce(func)	Aggregate the elements using give function
collect()	Return all elements of dataset as an array
count()	Return number of elements in the dataset
first()	Return the first element of the dataset (take(1))
take(n)	Return an array of the first n elements of dataset
takeOrdered(n)	Return first n elements using either natural order or custom comparator
countByKey()	Return a hash map of (K, count) pairs for each key
saveAsTextFile(path)	Write elements in dataset to a text file

http://spark.apache.org/docs/latest/programming-guide.html#actions

Working with RDD

map(func) - transformation

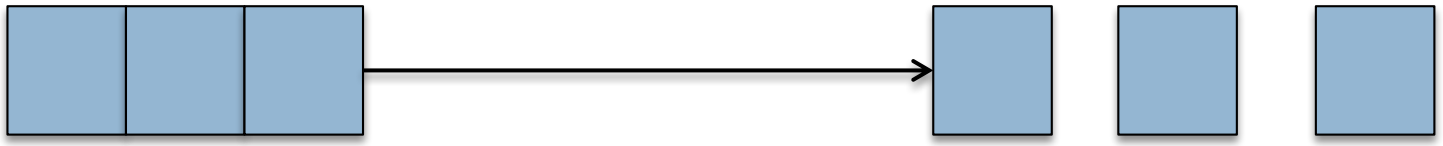
- Pass each element through the given function
- Each input item is mapped to only one output item

```
val wordsRDD = sc.parallelize(List("Spark","is","cool"))

val lowerCase = wordsRDD.map (word => word.toLowerCase)
lowerCase.collect()

def toLower(w : String) : String = { w.toLowerCase() }

val lowerCase2 = wordsRDD.map (word => toLower(word))
lowerCase2.collect()

val wordCount = wordsRDD.map (word => (word, 1))
wordCount.collect()
```

Working with RDD

flatMap(func) – transformation

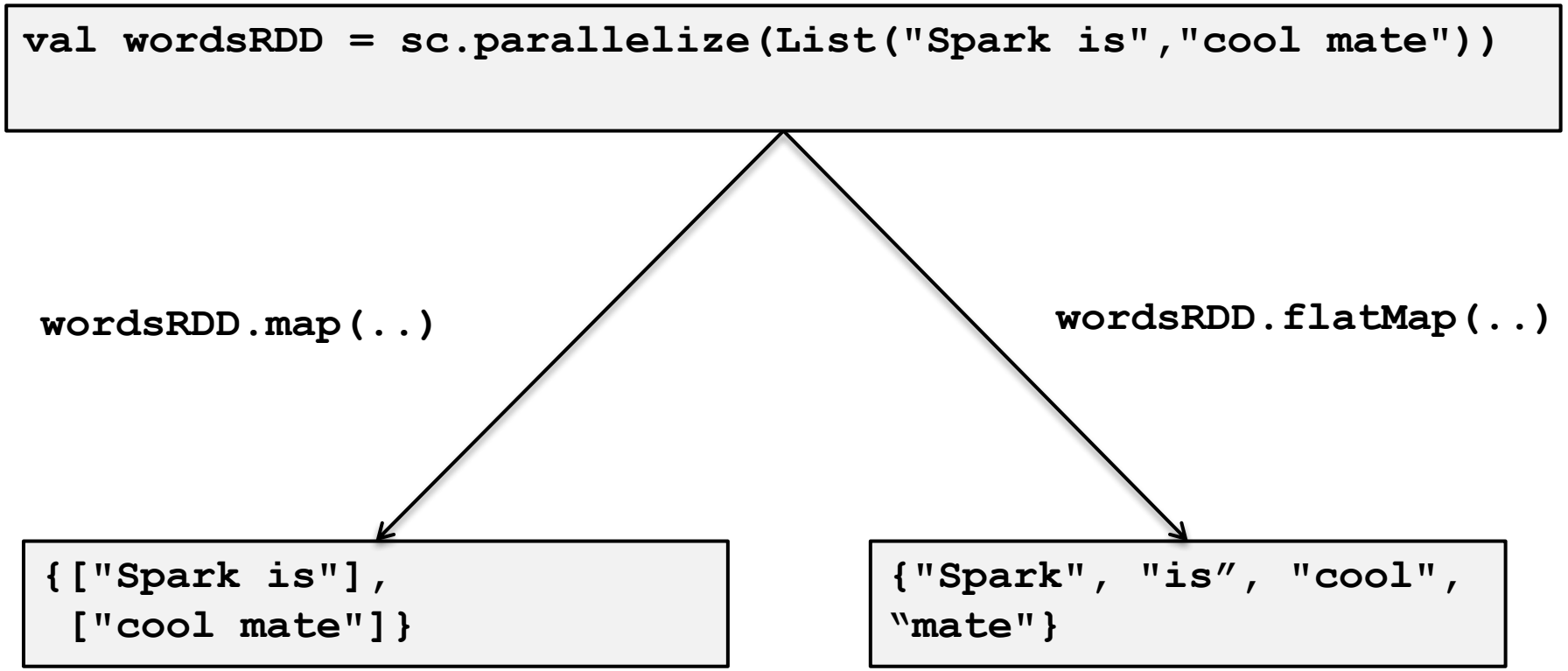
- Similar to map, each input item can be mapped to 0 or more output items



```
val wordsRDD = sc.parallelize(List("Spark is","cool mate"))

// split return an array of words
val words = wordsRDD.flatMap(l => l.split(" "))

words.first()   // Spark
words.collect() // Spark, is, cool, mate
```

Working with RDD

map(func) vs flatMap(func)



Working with RDD

filter(func) – transformation

- Return elements that *func* returns true

```
val numbersRDD = sc.parallelize(List(1,2,3,4,5))

val evens = numbersRDD.filter(n => n % 2 == 0)
evens.collect()

def odd(n : Int) : Boolean = { n % 2 == 1}
val odds = numbersRDD.filter(n => odd(n))

odds.collect().mkString(", ")
```

Working with RDD

sample(withReplacement, fraction, seed=none) –

- Sample a fraction of the data
- With replacement – an element may appear more than once
- Without replacement: avoid choosing any element more than once

```
val numbersRDD = sc.parallelize(1 to 10000, 3)
val randInts = numbersRDD.sample(true, 0.1, 0)

randInts.count

randInts.take(10)

// exercise, find elements that appear more than once
```

Working with RDD

distinct() –

- Remove duplicates
- Require shuffling all data over the network

```
val numbersRDD = sc.parallelize(Array(1,1,2,3,3,4,5,5))
val uniqueInts = numbersRDD.distinct()

uniqueInts.collect() // 1,2,3,4,5
```

Working with RDD

***union(otherRDD)* —**

- New RDD containing elements from both RDDs
- Useful for combining data sets
- Will contain duplicates

```
val sc = spark.sparkContext
val rdd1 = sc.parallelize(Array(1,2,3,4,5))
val rdd2 = sc.parallelize(Array(1,6,7,8))

val rdd3 = rdd1.union(rdd2)
// 1,2,3,4,5,1,6,7,8
```

Working with RDD

intersection(otherRDD) —

- Return only elements in both RDDs
- Will remove duplicates
- Require shuffle data over the network

```
val rdd1 = sc.parallelize(Array(1,1,2,3,4,5))
val rdd2 = sc.parallelize(Array(1,5,6,7,8))

val rdd3 = rdd1.intersection(rdd2)

rdd3.collect()  // 1,5
```

Working with RDD

subtract(otherRDD) —

- Elements present in first RDD and not in second RDD
- Require shuffle data over the network

```
val rdd1 = sc.parallelize(Array(1,1,2,3,4,5))
val rdd2 = sc.parallelize(Array(1,5,6,7,8))

val rdd3 = rdd1.subtract(rdd2)

rdd3.collect()  // 2,3,4

// good for removing stop words
```

Working with RDD

cartesian(otherRDD) –

- Return all possible pairs (a,b)
- Useful for finding similarity between all possible pairs
- Very expensive on large RDD

```
val rdd1 = sc.parallelize(Array(1,2,3))
val rdd2 = sc.parallelize(Array("A","B","C"))

val rdd3 = rdd1.cartesian(rdd2)

rdd3.collect()

// (1,A), (1,B), (1,C), (2,A), (2,B), (2,C), (3,A), (3,B), (3,C)
```

Working with RDD

Actions compute a result or save to storage system

Action	Description
reduce(func)	Aggregate the elements using give function
collect()	Return all elements of dataset as an array
count()	Return number of elements in the dataset
first()	Return the first element of the dataset (take(1))
take(n)	Return an array of the first n elements of dataset
takeOrdered(n)	Return first n elements using either natural order or custom comparator
countByKey()	Return a hash map of (K, count) pairs for each key
saveAsTextFile(path)	Write elements in dataset to a text file

http://spark.apache.org/docs/latest/programming-guide.html#actions

Working with RDD

reduce(func) –

- Aggregate the elements using given func
- Return type must be the same type as the input elements
- Examples - addition, multiplication

```
val rdd1 = sc.parallelize(Array(1,2,3,4,5))

val result = rdd1.reduce((a,b) => (a+b))

println("Spark is easy: " + result)

// Spark is easy: 15

// How would one compute average?
```

Working with RDD

collect() –

- Bring all the elements from RDD back to driver
- Make sure RDD is small, otherwise OOM

```
val rdd1 = sc.parallelize(Array(1,2,3,4,5))
rdd1.collect()  // 1,2,3,4,5
```

count() – return # of elements

```
val rdd1 = sc.parallelize(Array(1,2,3,4,5))

rdd1.count()  // 5
```

Working with RDD

first() – first element

```
val rdd1 = sc.parallelize(Array(1,2,3,4,5))
rdd1.first()  // 1
```

take(n) – an array of n elements

```
val rdd1 = sc.parallelize(Array(1,2,3,4,5))

rdd1.take(2)  // (1,2)
```

takeOrdered(n) – n element with ordering

```
val rdd1 = sc.parallelize(Array(5,3,1,2,4))
rdd1.takeOrdered(2)  // (1,2)
rdd1.takeOrdered(2)(Ordering[Int].reverse)  // (5,4)
```

Working with RDD

first() – first element

```
val rdd1 = sc.parallelize(Array(1,2,3,4,5))
rdd1.first()  // 1
```

take(n) – an array of n elements

```
val rdd1 = sc.parallelize(Array(1,2,3,4,5))

rdd1.take(2)  // (1,2)
```

top(n) – top n elements implicit ordering

```
val rdd1 = sc.parallelize(Array(1,2,3,4,5))

rdd1.top(2)  // (5,4)
```

Working with RDD

countByValue() –

- Count number of times each element occurs
- Return a map [value, occurrence]
- One way of counting words

```
val rdd1 = sc.parallelize(Array(1,2,3,4,5,5))
rdd1.countByValue()

// Map(5 -> 2, 1 -> 1, 2 -> 1, 3 -> 1, 4 -> 1)
```

Working with RDD

aggregate(zeroValue, combiner, merger) —

- Can return different type than original RDD
- Flexible and can use to compute average
- Combiner is applied within each partition
- Merger is applied to results of all partitions

```
val rdd1 = sc.parallelize(Array(1,2,3,4,5))

val result = rdd1.aggregate((0,0))(
    (acc, value) => (acc._1 + value, acc._2 + 1),
    (acc1, acc2) => (acc1._1 + acc2._1, acc1._2 + acc2._2))

val avg = result._1 / result._2.toDouble
```

Working with Pair RDD

- Working with key/value pairs
  - Each element must be a key-value pair
    - A two-element tuple
  - Key and value can be of any type
  - Good for performing aggregation, counting, join, etc
  - Understand partitioning to reduce shuffling
  - Create pair RDD using tuple

```
val rdd1 = sc.parallelize(Array(1,2,3,4,5))

val pairs = rdd1.map(n => (n,1))

// Array((1,1), (2,1), (3,1), (4,1), (5,1))
```

Working with Pair RDD

Transformation on RDDs of key-value pairs

Transformation	Description
groupByKey()	Group together values with same key. (K,V) => (K, Iterable<V>)
cogroup(otherDataset)	(K,V), (K,W) => (K, Iterable<V>, Iterable<W>)
reduceByKey(func)	Combine values with same key together
mapValues(func)	Apply func to each value of a Pair w/o changing key
flatMapValues(func)	Apply func to each value and run flatMap on ressult
keys()	Return an RDD of just keys
values()	Return an RDD of just values
join(otherDataset)	Return an RDD of the join. (K,V) and (K,W) => (K, (V,W))
sortByKey([ascending])	Return an RDD sorted by key

Working with Pair RDD

Transformation on two key-value pairs RDDs

Transformation	Description
rightOuterJoin(other)	Key must present on other RDD
leftOuterJoin(other)	Key must present on first RD
cogroup(other)	Group data from both RDDs

Working with Pair RDD

groupByKey() –

- Group data by key
- Usually follow by some kind of aggregation

```
val rdd1  = sc.parallelize(Array(("Spark",10), ("Tez",8),
("Pig",5), ("Spark",2), ("Pig",3)))

val groups = rdd1.groupByKey()
groups.collect()

// Array((Tez,(8)), (Spark,(10, 2)), (Pig,(5, 3)))

val sum = groups.map(p => (p._1, p._2.sum))
// Array((Tez,8), (Spark,12), (Pig,8))
```

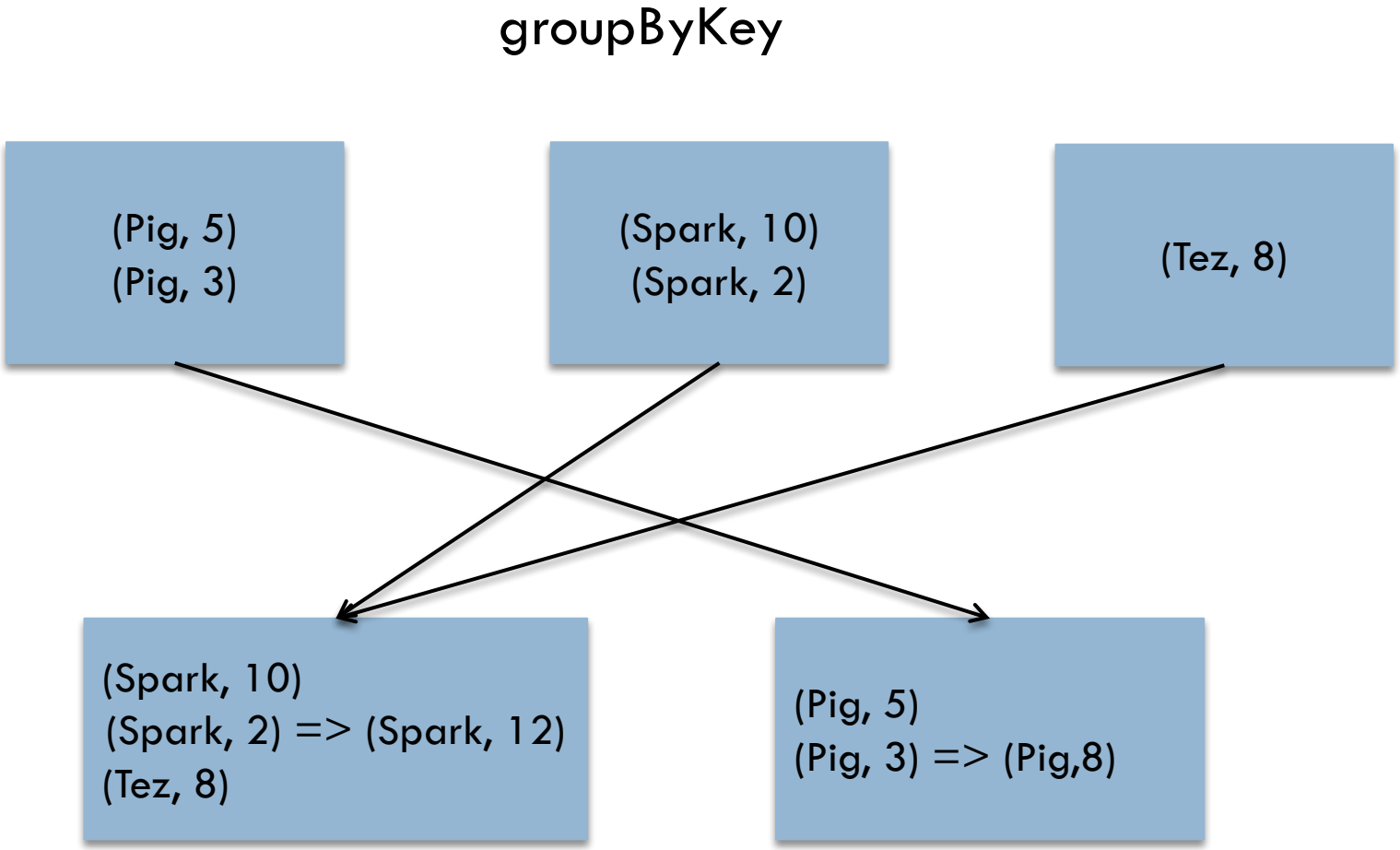
Working with Pair RDD

reduceByKey() –

- Perform aggregate on values
- Combine common keys on each partition before shuffling

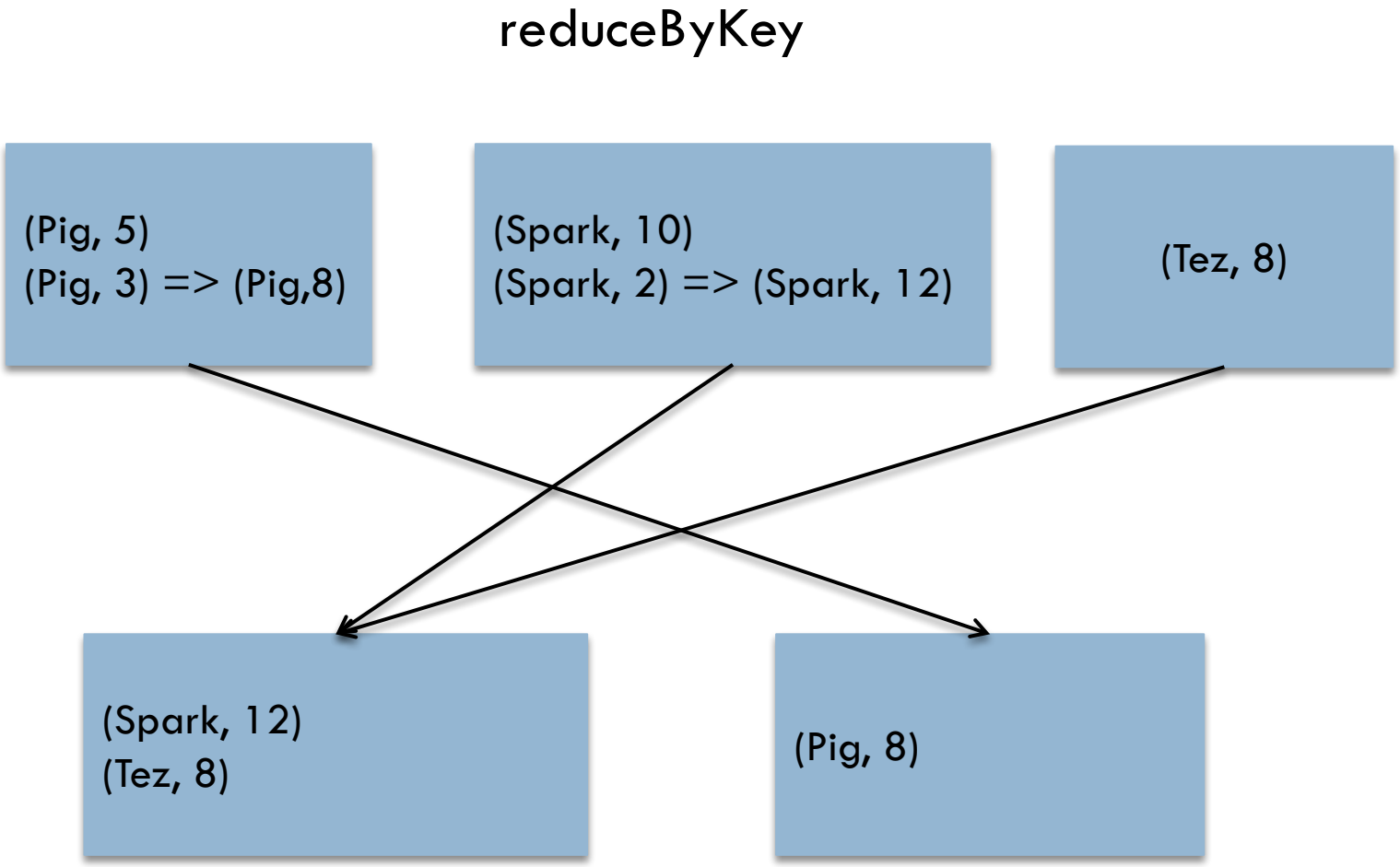
```
val rdd1  = sc.parallelize(Array(("Spark",10), ("Tez",8),
("Pig",5), ("Spark",2), ("Pig",3)))

val groups = rdd1.reduceByKey((x,y) => x + y)
groups.collect()

// Array((Tez,(8)), (Spark,(12)), (Pig,(8)))
```

Working with Pair RDD



A lot of unnecessary data to being transferred over the network

Working with Pair RDD



Working with Pair RDD

sortByKey([ascending]) –

- Return a RDD sorted by key by ascending or descending

```
val rdd1  = sc.parallelize(Array(("Spark",10), ("Tez",8),
("Pig",5), ("Spark",2), ("Pig",3)))

val sortedRDD = rdd1.sortByKey()
sortedRDD.collect()
// Array((Pig,5), (Pig,3), (Spark,10), (Spark,2), (Tez,8))

rdd1.sortByKey(false).collect()
// Array((Tez,8), (Spark,10), (Spark,2), (Pig,5), (Pig,3))
```

Working with Pair RDD

mapValues(func) –

- Apply func to each pair w/o changing the key

```
val rdd1  = sc.parallelize(Array(("Spark",10), ("Tez",8),
("Pig",5), ("Spark",2), ("Pig",3)))

val valueRDD = rdd1.mapValues(v => v + 5)
valueRDD.collect()

// Array((Spark,15), (Tez,13), (Pig,10), (Spark,7),
(Pig,8))
```

Working with Pair RDD

keys(), values()

- Return RDD with just keys or values

```
val rdd1  = sc.parallelize(Array(("Spark",10), ("Tez",8),
("Pig",5), ("Spark",2), ("Pig",3)))

rdd1.keys.collect()

// Array(Spark, Tez, Pig, Spark, Pig)

rdd1.values.collect()
//Array(10, 8, 5, 2, 3)

rdd1.values.sum
```

Working with Pair RDD

join(otherRDD)

- Inner join between two RDDs
- One of the most commonly used transformations

```
val rdd1  = sc.parallelize(Array(("Spark",10), ("Tez",8),
("Pig",5), ("Spark",2), ("Pig",3)))
val rdd2  = sc.parallelize(Array(("Spark","Awesome")))

val joinedRdd = rdd1.join(rdd2)
joinedRdd.collect

//Array((Spark,(10,Awesome)), (Spark,(2,Awesome)))
```

Working with Pair RDD

cogroup(otherRDD)

- Grouping of elements from two RDDs
- (K,V) and (K,W) => (K, Iterable<V>, Iterable<W>)

```
val rdd1  = sc.parallelize(Array(("Spark",10), ("Tez",8),
("Pig",5), ("Spark",2), ("Pig",3)))
val rdd2  = sc.parallelize(Array(("Spark","Awesome")))

val cogroup = rdd1.cogroup(rdd2)
cogroup.collect

//Array((Tez,((8),())), (Spark,((10, 2),(Awesome))),
(Pig,((5, 3),())))
```

Working with Pair RDD

K-V pair transformation examples

```
{("Spark",10), ("Tez”,8), ("Pig”,5), ("Spark”,2), ("Pig”,3)}
```

Transformation	Example
reduceByKey(func)	`reduceByKey((x,y) => (x+y))` `{(“Spark”,12), (“Tez”,8”), (“Pig”, 8)}`
groupByKey()	`{(“Spark”,[10,2]), (“Tez”,[8]), (“Pig”, [5,3])}`
sortByKey()	`{("Pig”,5), ("Pig”,3), ("Spark",10), ("Spark”,2), ("Tez”,8)}`
countByKey()	`{("Spark”,2), ("Pig”,2), ("Tez”,1)}`

Working with Pair RDD

RDDs of key-value pairs actions

Transformation	Description
countByKey()	Count the number of elements for each key
lookup(key)	Return values associated with given key

Working with Pair RDD

countByKey() –

- Return a hashmap (K,Long) pairs with count for each key

```
val rdd1  = sc.parallelize(Array(("Spark",10), ("Tez",8),
("Pig",5), ("Spark",2), ("Pig",3)))

rdd1.countByKey()

// Map(Tez -> 1, Spark -> 2, Pig -> 2)
```

Working with Pair RDD

Time for hands on exercise

Programming with RDD

- Spark shell
  - Interactive REPL (Scala or Python)
  - spark-shell, pyspark
- Spark application
  - Python, Scala, Java

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.4.1
      /_/

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_25)
```