UNIVERSITY of WASHINGTON

BIGDATA 210: Introduction to Data Engineering

Autumn 2017

Module 3: Data Processing Using Spark Part I

Jason Kolter

jkolter@uw.edu

Week 3 Agenda

- Assignment/Week 2 Review
- Programming in Spark Part I
  - What is Spark
  - Spark architecture
  - Programming in Spark
    - Functional Programming Concepts
    - Using RDDs
- Week 2 Catch-up

Assignment 2

Week 2

- Sandbox “Learning the Ropes” tutorial
  - https://hortonworks.com/tutorial/learning-the-ropes-of-the-hortonworks-sandbox
  - Tour of the sandbox and Ambari
    - SSH into Docker
    - SCP files into Docker
      - Also should see your /data folder mounted into Docker container
  - Has additional Hadoop/Hive/Pig tutorials/examples

What is Spark?

- Apache Spark is a fast and general engine for large-scale data processing

What is Spark?

- Problems with Hadoop
  - Programming Model
    - Strictly map -> reduce with key/value pairs
  - Slow
    - HDFS dependent
    - Doesn’t support multiple transformation stages

What is Spark?

- Spark addresses these issues (and more)
  - Programming Model
    - ~~Strictly map -> reduce with key/value pairs~~
    - Supports any Scala/Java/Python/R/SQL constructs
    - Higher level API to perform most functionality
  - Slow
    - ~~HDFS Dependent~~
    - ~~Doesn't support multiple stages of transformation~~
    - In-memory execution model
      - Algorithms that can reuse/share data run exponentially faster
    - Supports other storage medium

What is Spark?

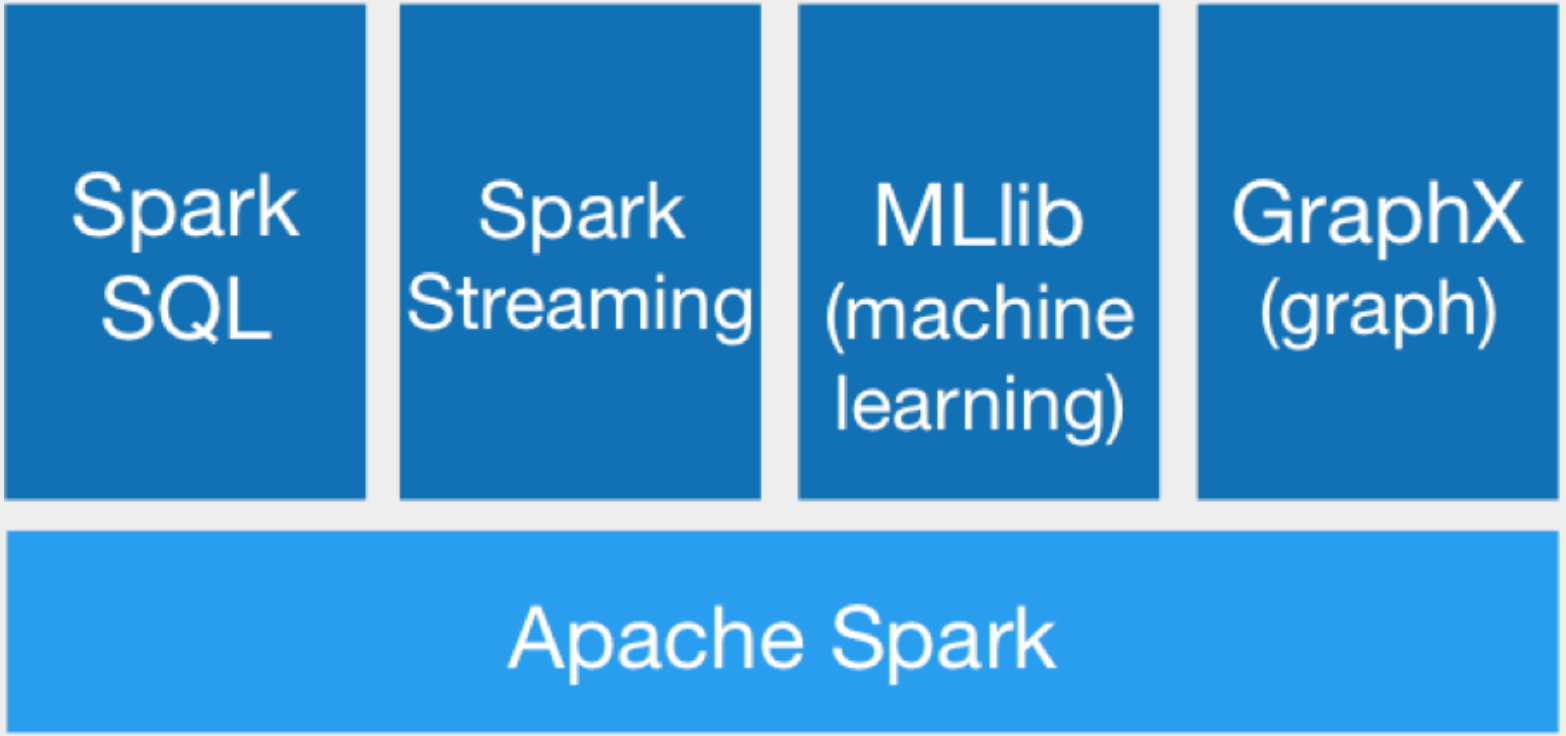
- Spark addresses these issues (and more)
  - Programming Model
    - ~~Strictly map -> reduce with key/value pairs~~
    - Supports any Scala/Java/Python/R/SQL constructs
    - Higher level API to perform most functionality
  - Slow
    - ~~HDFS Dependent~~
    - ~~Doesn't support multiple stages of transformation~~
    - In-memory execution model
      - Algorithms that can reuse/share data run exponentially faster
    - Supports other storage medium

What is Spark?

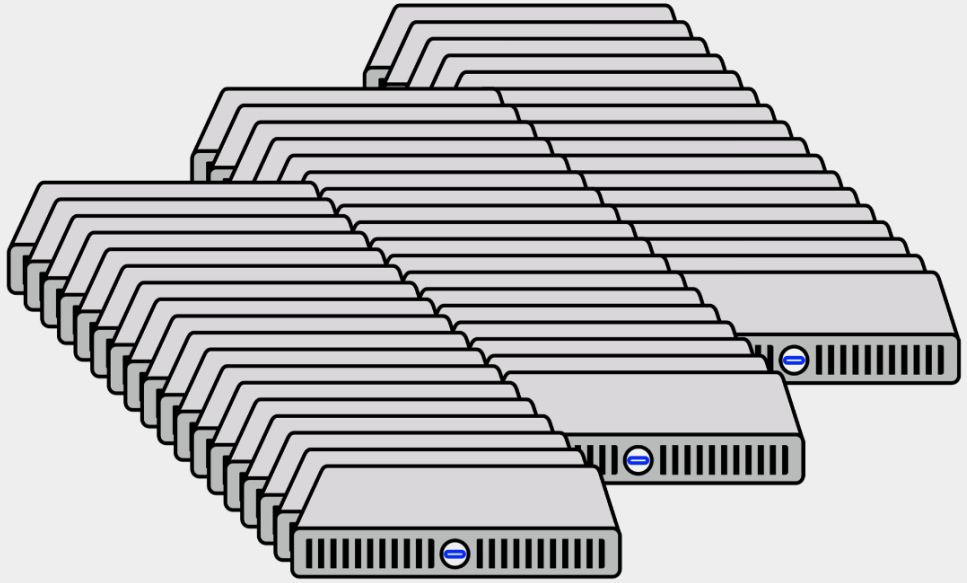
```
// mapper
private IntWritable one = new IntWritable(1);
private IntWritable output = new IntWritable();
proctected void map(LongWritable key, Text value, Context context) {
    String[] fields = value.split("\t");
    output.set(Integer.parseInt(fields[1]));
    context.write(one, output);
}

// reducer
IntWritable one = new IntWritable(1);
DoubleWritable average = new DoubleWritable();
protected void reduce(IntWritable key, Iterable<IntWrtiable> values, Context context) {
    int sum = 0;
    int count = 0;
    for(IntWritable value : values) {
        sum += value.get();
        count++;
    }
    average.set(sum / (double) count);
    context.write(key, average);
}
```

```
val input = sc.textFile(“/path/to/input”)
val sums = numbers.flatMap(line => line.split(“\t)).reduce(_+_)
val count = numbers.count
val avg = sum / (count * 1.0)
```

What is Spark?

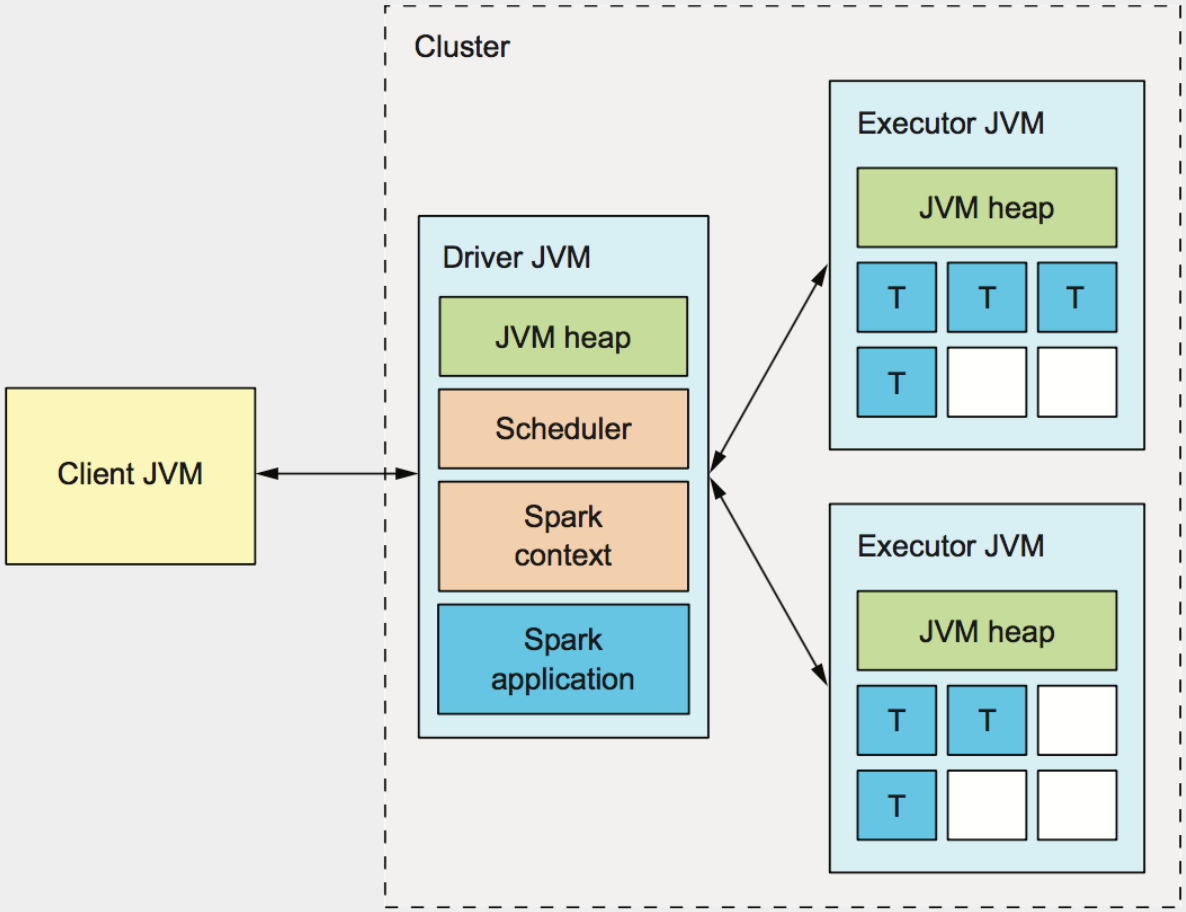


Spark Application Architecture



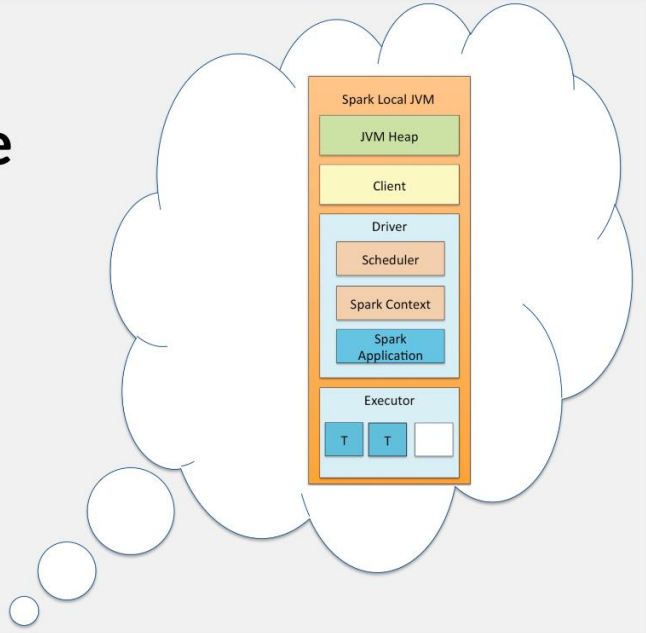
Spark Application Architecture

Spark Runtime Architecture



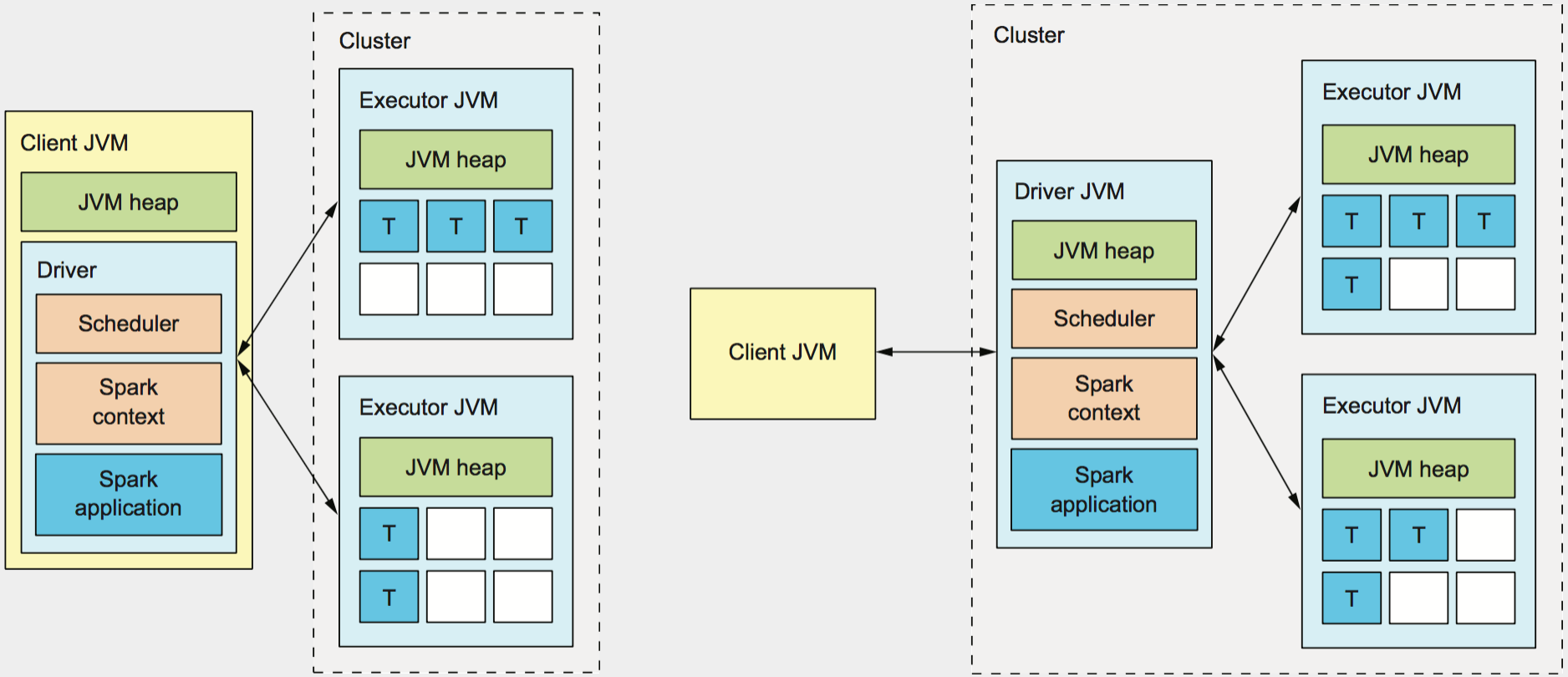
Spark Application Architecture

Local Mode



Spark Application Architecture

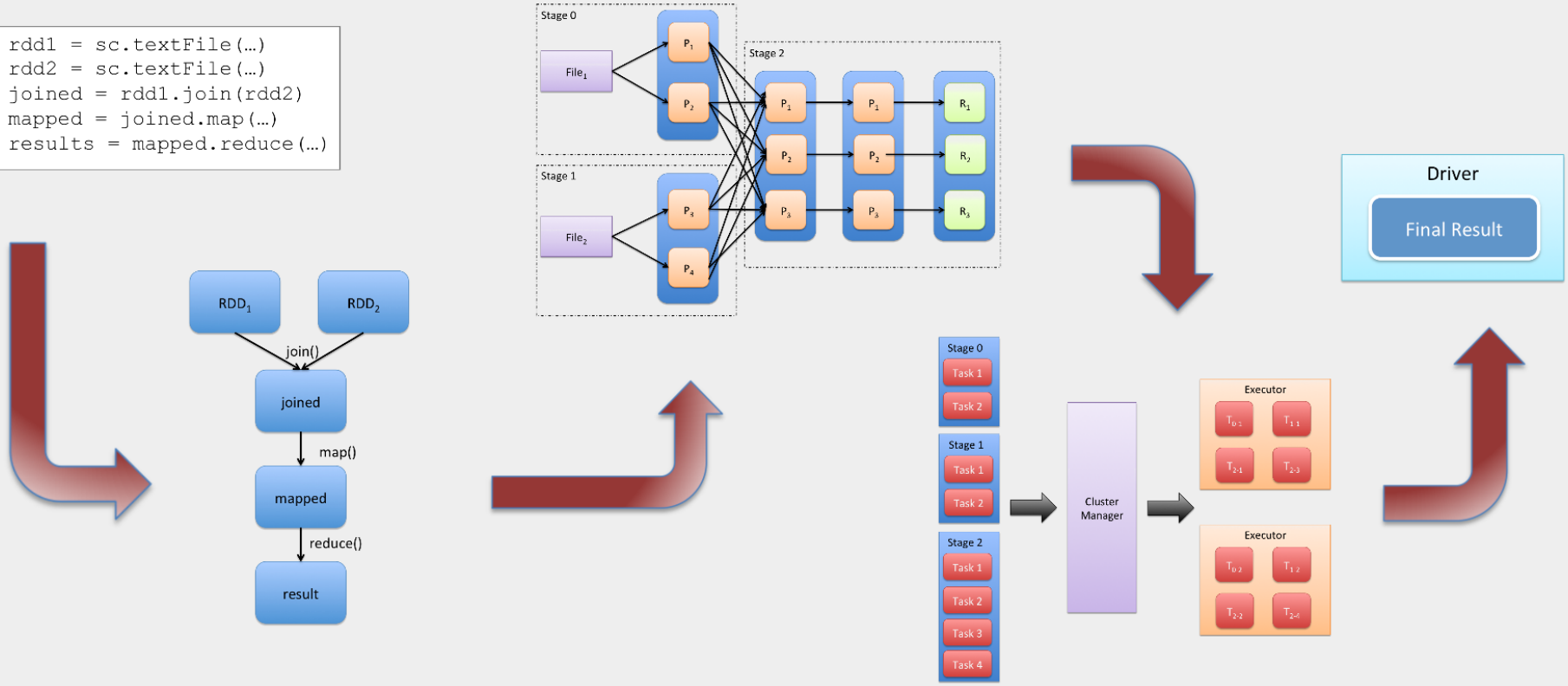
Cluster Mode



Spark Application Architecture

Spark Application Lifecycle

```
val rdd1 = sc.textFile(…)
val rdd2 = sc.textFile(…)
val joined = rdd1.join(rdd2)
val mapped = joined.map(…)
val results = mapped.reduce(…)
```



Programming Spark

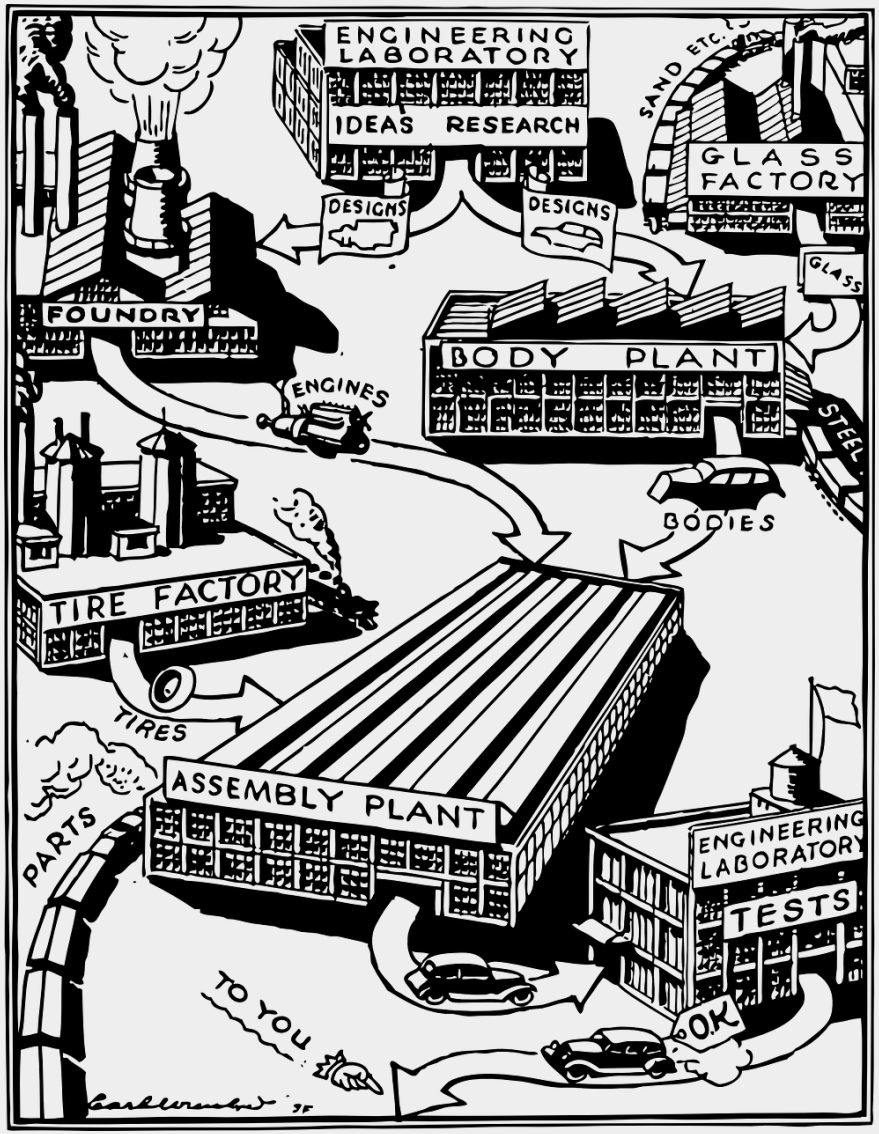
- Functional Programming
  - Applications defined in terms of mathematical functions
  - Declarative vs imperative paradigm

```
int[] array = new int[] {1, 2, 3, 4, 5};
for (int i = 0; i < array.length; i++) {
    array[i] = array[i] * 2;
}
```

```
val array = Array(1, 2, 3, 4, 5)
val doubled = array.map(x => x * 2)
```

Programming Spark

Programming Spark



Programming Spark

- Executing Spark Applications
  - Static compiled programs
  - REPL Shell
    - Read, Eval, Print, Loop

Programming Spark

- Resilient Distributed Dataset (RDD)
  - Built-in fault tolerance
  - Partitioned across cluster
  - Can represent different kinds of data
    - Text, Key/Value pairs, custom

–IMMUTABLE

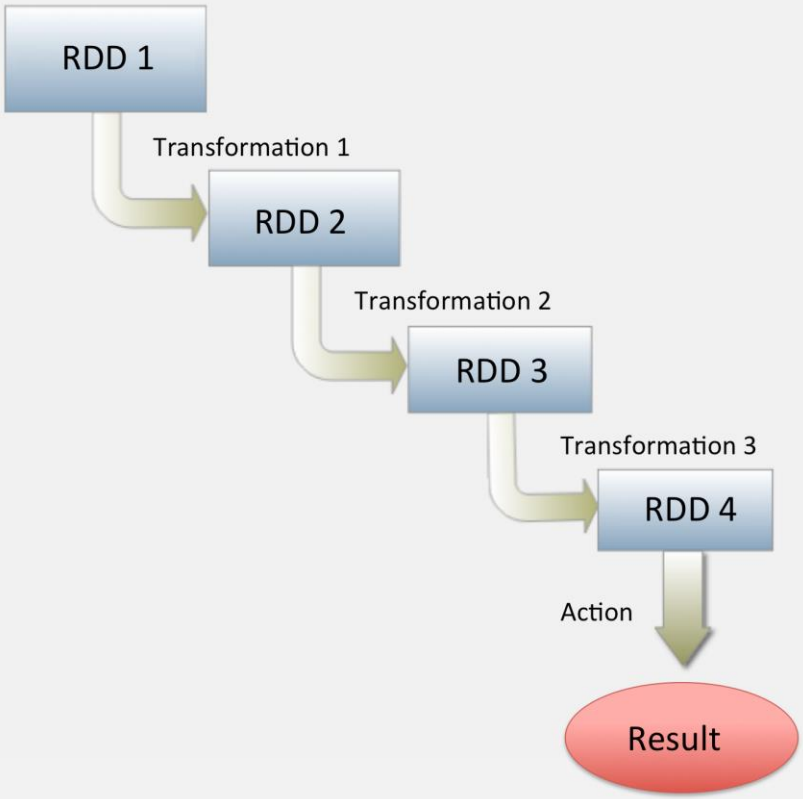
Programming Spark

- Resilient Distributed Dataset (RDD)
  - LAZY EXECUTION

Programming Spark

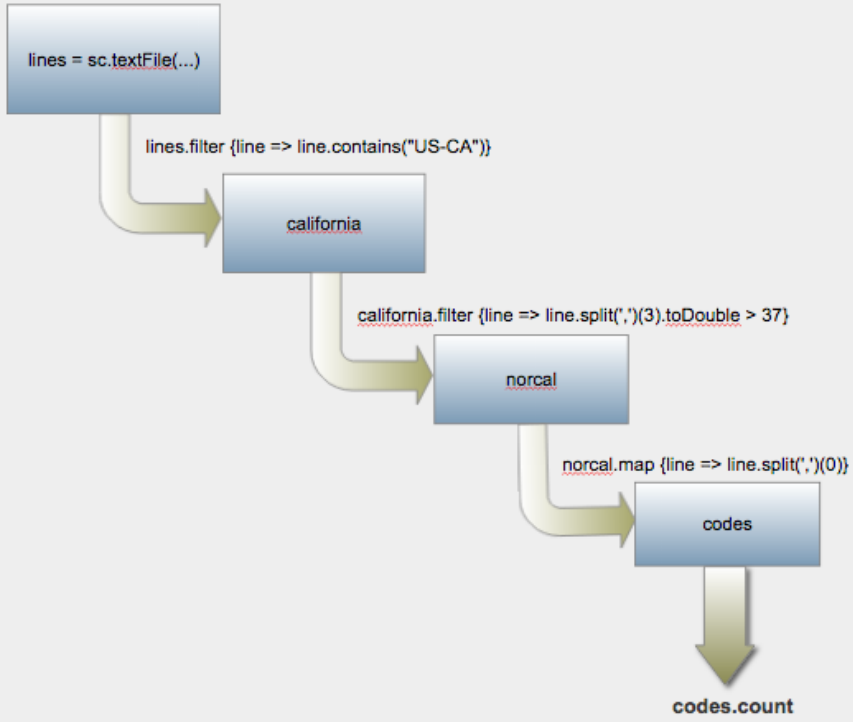
- Resilient Distributed Dataset (RDD)
  - Transformation
    - Create a new data set from an existing one
  - Action
    - Return a result to the driver
    - Does not need to be a “final” result

Programming Spark



Programming Spark

```
val lines = sc.textFile("file:///data/airport_codes.csv")
val california = lines.filter {line => line.contains("US-CA")}
val norcal = california.filter {line => line.split(',')(3).toDouble > 37}
val codes = norcal.map {line => line.split(',')(0)}
codes.count
```



Programming Spark

- Creating/loading RDDs
  - Existing Scala collection

```
import scala.util._
val randoms = Seq.fill(1000)(Random.nextInt)
print(randoms.take(10))
```

  - Text File

```
val lines = sc.textFile("file:///data/home_data.csv")
```

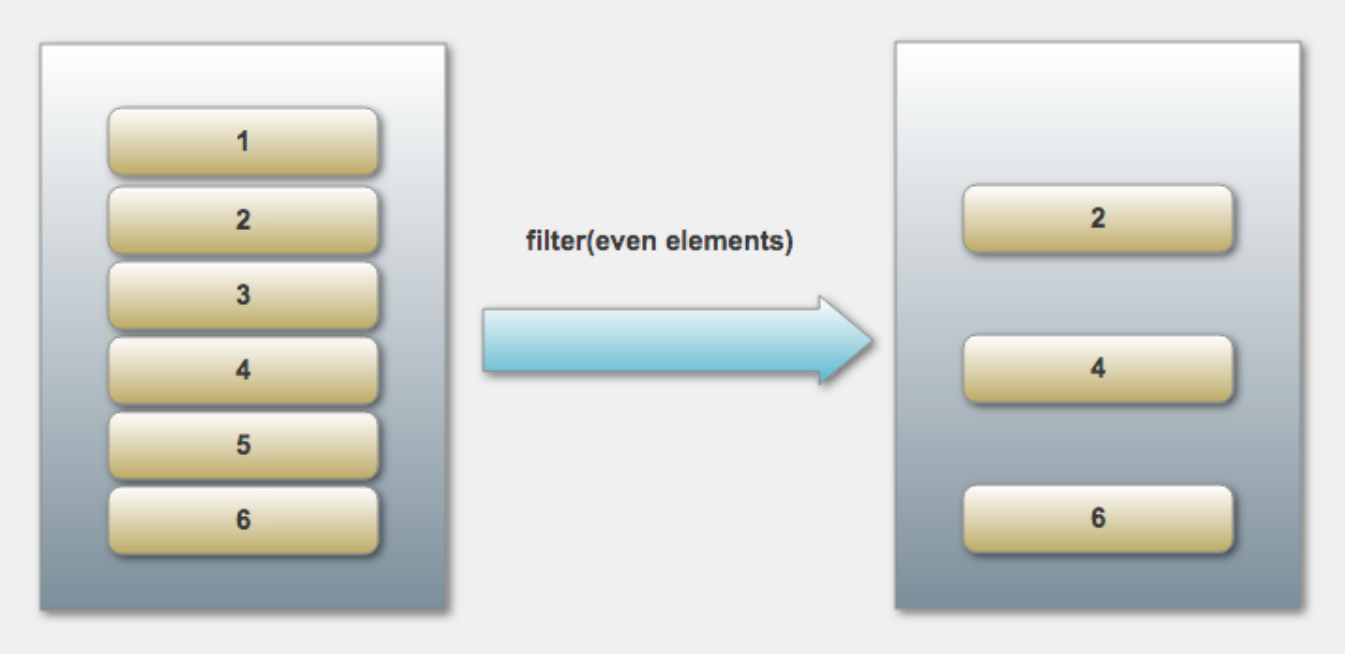
```
val lines = sc.textFile("hdfs://sandbox.hortonworks.com:8020/tmp/home_data.csv")
```

  - Whole Directory of Files

```
val files = sc.wholeTextFiles("file:///data/*.csv")
```

RDD Transformations

Filter



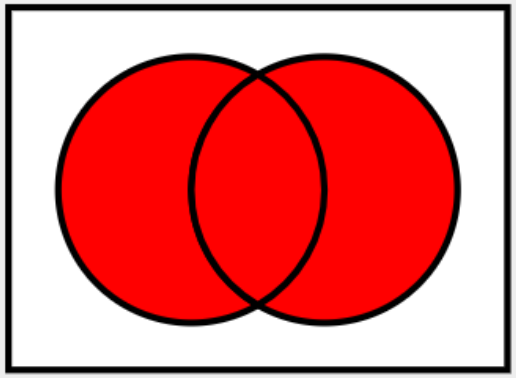
def filter(f: (T) ⇒ Boolean): RDD[T]

Return a new RDD containing only the elements that satisfy a predicate.

```
val exampleRDD = sc.parallelize(1 to 20)
val filteredRDD = exampleRDD.filter(value => value % 2 == 0)
```

RDD Transformations

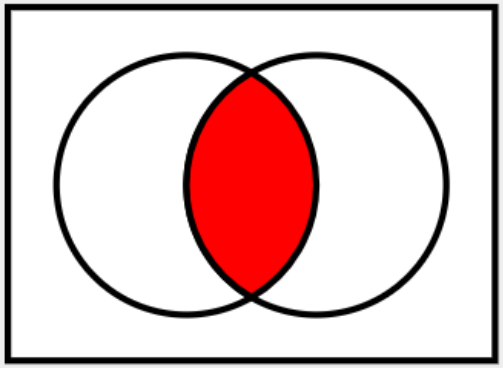
Union



def union(other: RDD[T]): RDD[T]

Return the union of this RDD and another one.

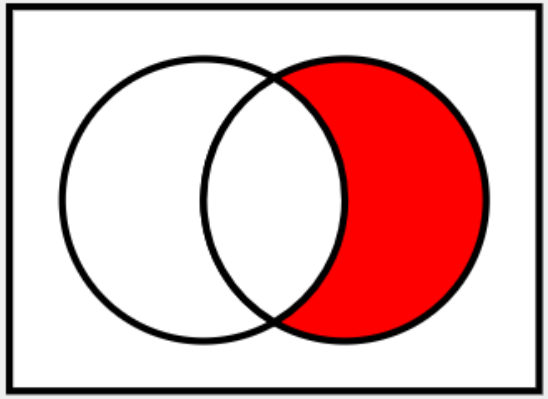
Intersection



def intersection(other: RDD[T]): RDD[T]

Return the intersection of this RDD and another one.

Subtraction



def subtract(other: RDD[T]): RDD[T]

Return an RDD with the elements from this that are not in other.

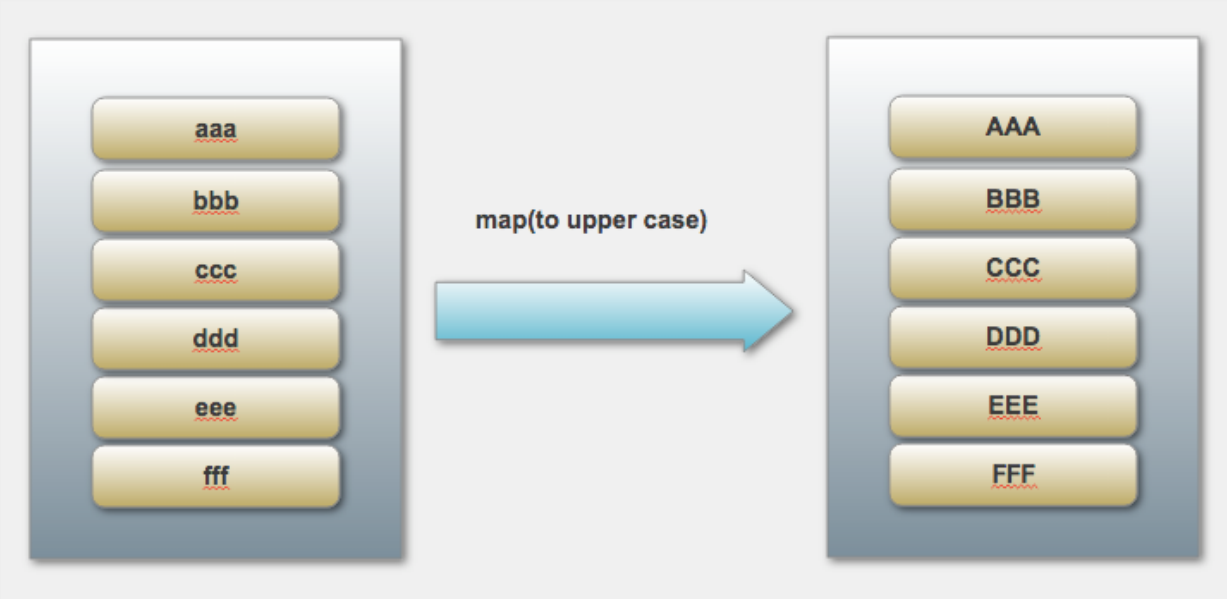
RDD Transformations

Set Operations

```
val a = sc.parallelize(1 to 10)
val b = sc.parallelize(11 to 20)


val aUnionB = a.union(b)
// Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20)

val c = sc.parallelize(1 to 10 by 2
val aIntersectC = a.intersection(c)
// Array(1, 9, 5, 3, 7)

val aUnionBSub.subtract(a)
// Array(16, 20, 12, 17, 13, 18, 14, 19, 11, 15)
```

RDD Transformations

Map

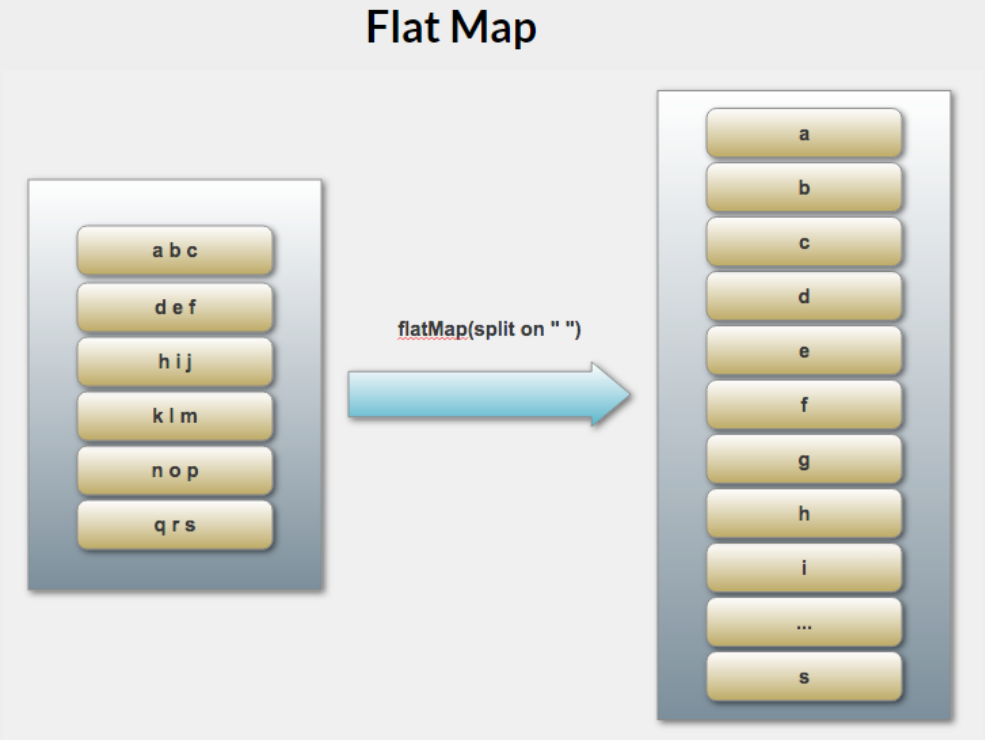


def map[U](f: (T) ⇒ U)(implicit arg0: ClassTag[U]): RDD[U]

Return a new RDD by applying a function to all elements of this RDD.

```
val text = sc.textFile("file:///data/war_and_peace.txt")
val upperText = text.map(line => line.toUpperCase())
```

RDD Transformations

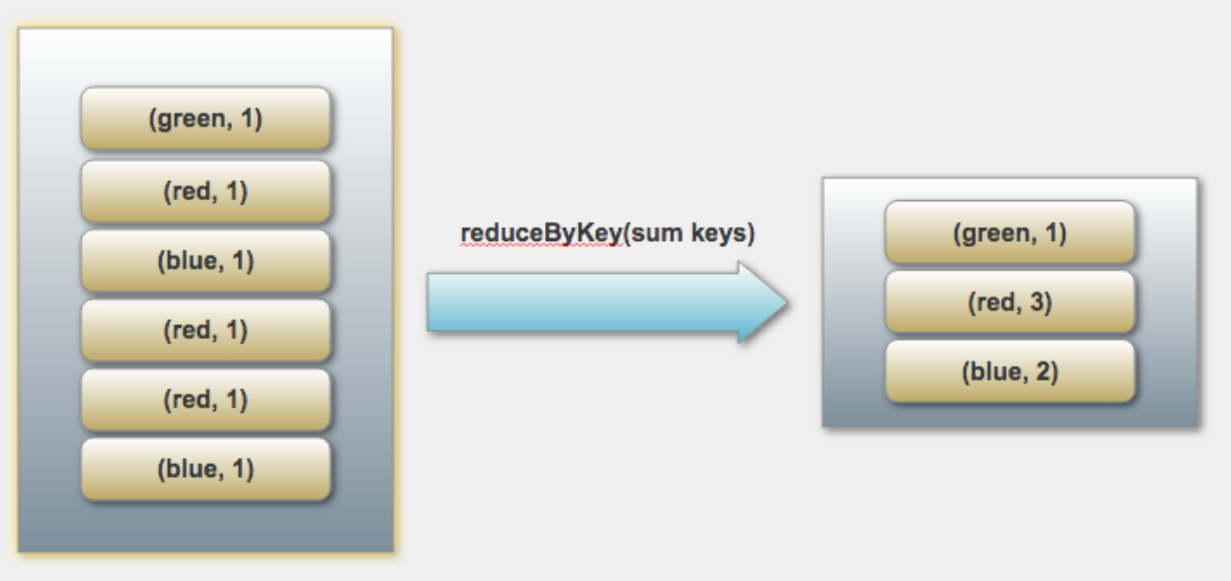


def flatMap[U](f: (T) ⇒ TraversableOnce[U])(implicit arg0: ClassTag[U]): RDD[U]

```
val text = sc.textFile("file:///data/war_and_peace.txt")
val words = text.flatMap(line => line.split(" "))
```

RDD Transformations

Reduce By Key



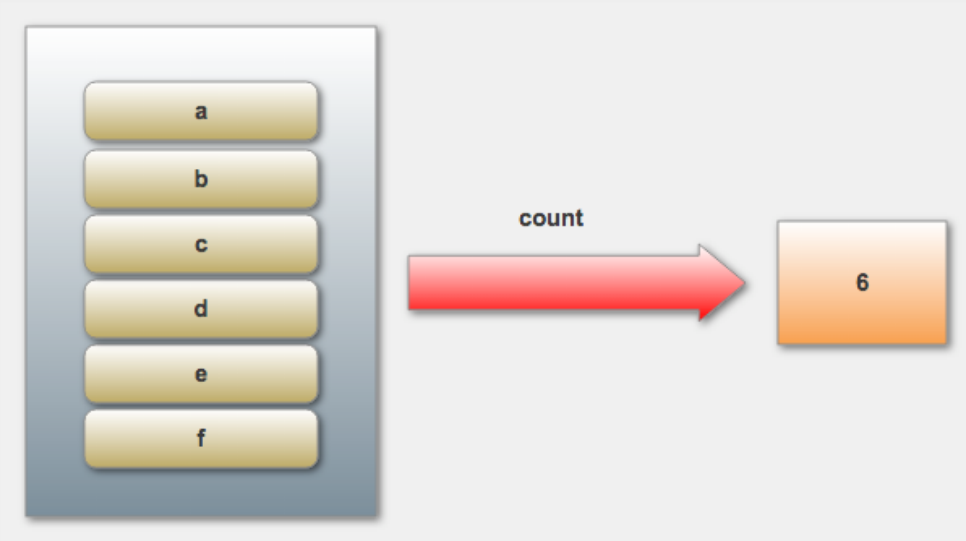
def reduceByKey(func: (V, V) ⇒ V): RDD[(K, V)]

Merge the values for each key using an associative and commutative reduce function.

```
val text = sc.textFile("file:///data/war_and_peace.txt")
val words = text.flatMap(line => line.split(" "))
val wordsPairs = words.map(word => (word,1))
val counts = wordsPairs.reduceByKey((value1,value2) => value1 + value2)
```

RDD Actions

Count



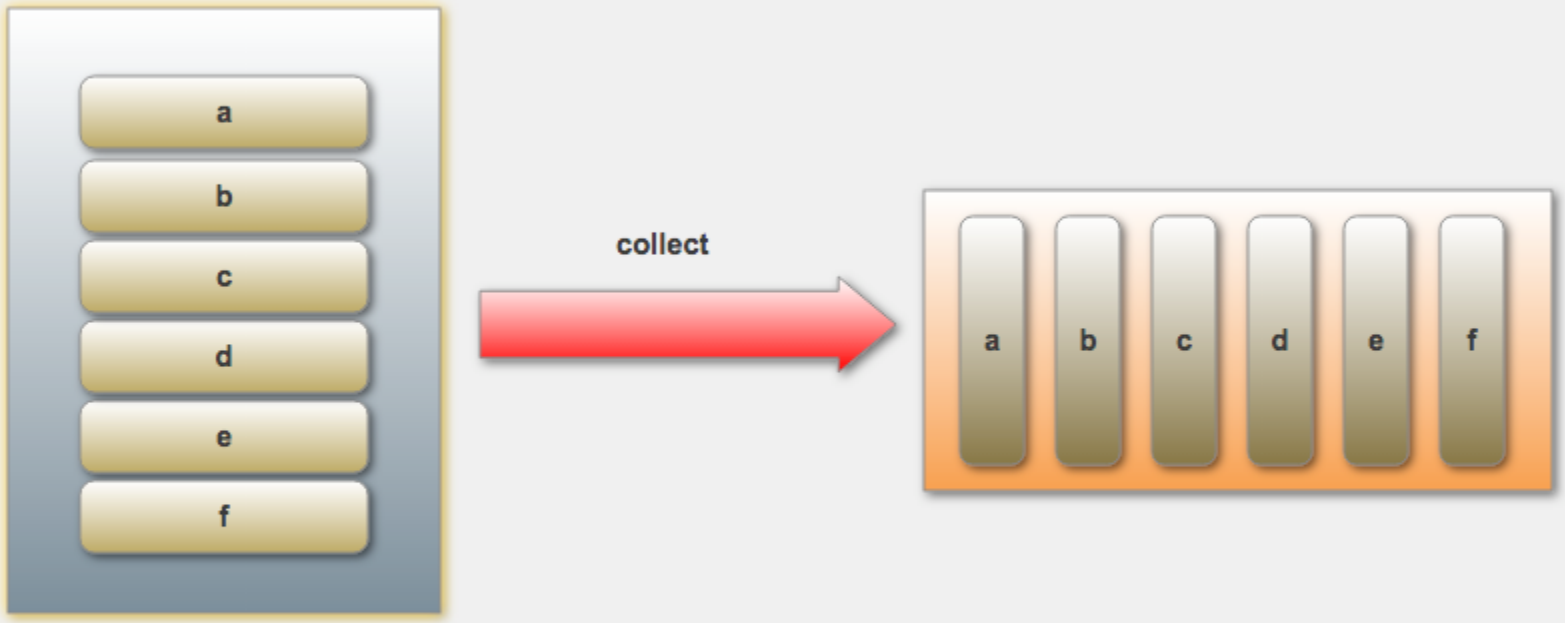
def count(): Long

Return the number of elements in the RDD.

```
val a = sc.parallelize(1 to 10)
a.count

// 10
```

RDD Actions

Collect



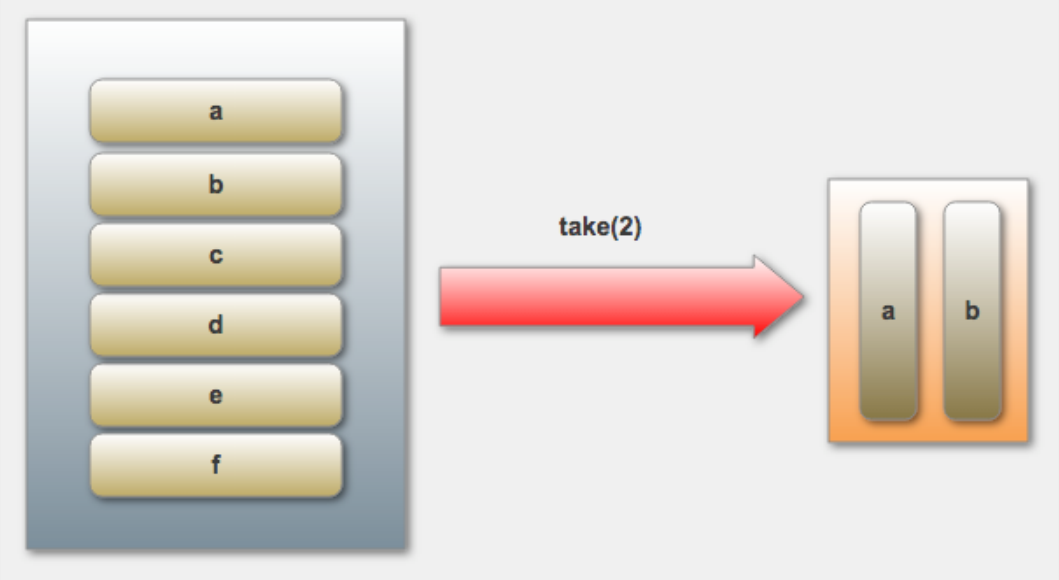
def collect(): Array[T]

Return an array that contains all of the elements in this RDD.

```
val a = sc.parallelize(1 to 10)
a.collect
// Array(1,2,3,4,5,6,7,8,9,10)
```

RDD Actions

Take



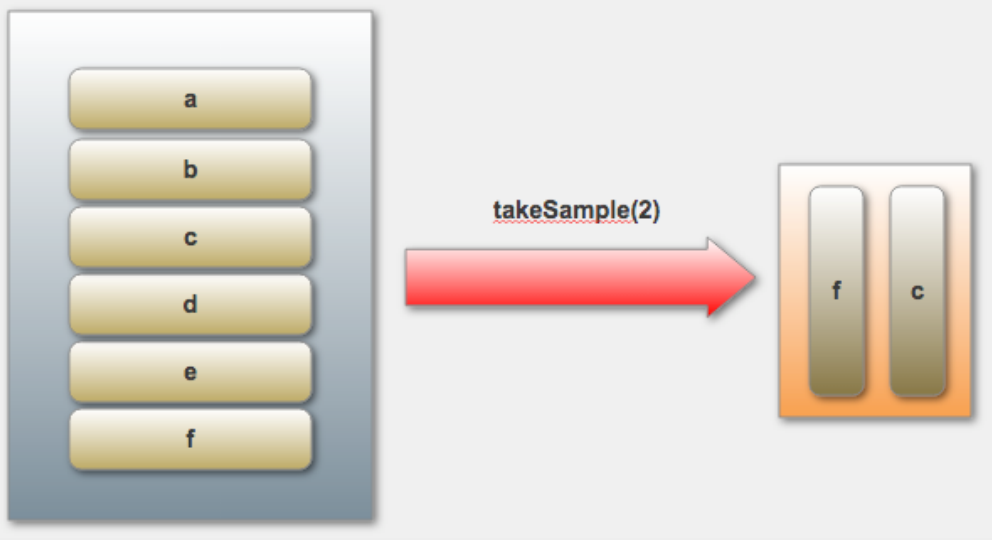
def take(num: Int): Array[T]

Take the first num elements of the RDD.

```
val a = sc.parallelize(1 to 10)
a.take(2)
// Array(1,2)
```

RDD Actions

Take Sample



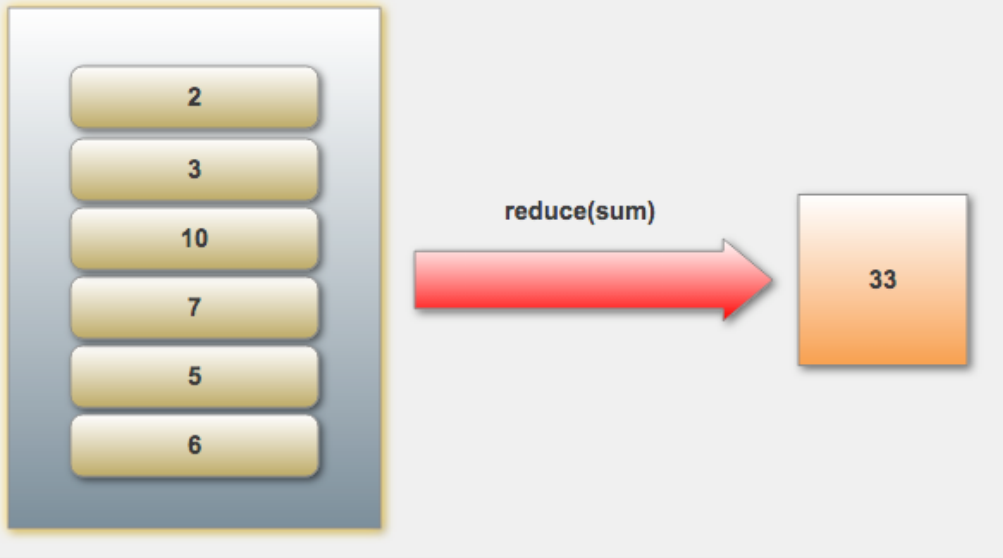
def takeSample(withReplacement: Boolean, num: Int, seed: Long = Utils.random.nextLong): Array[T]

Return a fixed-size sampled subset of this RDD in an array

```
val a = sc.parallelize(1 to 10)
a.takeSample(false, 2)
// Array(6, 8)
```

RDD Actions

Reduce



def reduce(f: (T, T) ⇒ T): T

Reduces the elements of this RDD using the specified commutative and associative binary operator.

```
val a = sc.parallelize(1 to 100)
a.reduce((v1,v2) => v1 + v2)
// 5050
```

RDD Save Actions

Save an RDD to disk as a text file, one element per line. Note that is this an output directory, not a file name. Each individual partition will be written to a separate file

```
counts.saveAsTextFile("file:///data/word_counts")
```

Save an RDD to disk using Java serialization. Also specifies an output directory, with individual files for each partition

```
counts.saveAsObjectFile("file:///data/word_counts_binary")
```

Spark Resources

API documentation

http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.rdd.RDD>

http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD

http://spark.apache.org/docs/latest/api/java/index.html?org/apache/spark/api/java/JavaRDD.html