intro to spark

compiled by waleed ammar, based on the following materials:

- Paco Nathan, Reza Zadeh and Jimmy Lin @ http://lintool.github.io/SparkTutorial
- Official guide @ http://spark.apache.org/docs/latest/programming-guide.html
- Fernando Rodriguez @ http://www.slideshare.net/frodriguezolivera/apache-spark-streaming
- Nicole Hemsoth @ hadoop_vs_spark
- MLlib docs @ http://spark.apache.org/docs/latest/mllib-feature-extraction.html
-

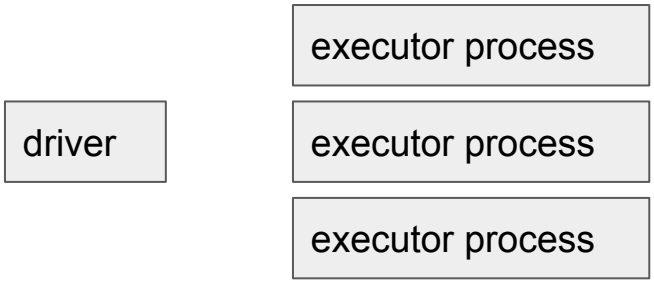
hands-on slide: open a spark shell on allegro

```
ssh allegro

cd /usr3/home/wammar/corpora/spark-training/        # or download here

./spark/bin/spark-shell  --master local[3]          # master urls


scala> val x = 10

scala> x + 5
```

spark on allegro

```
ssh allegro

cd /usr3/home/wammar/corpora/spark-training/        // or download here

./spark/bin/spark-shell  --master local[3]          // master urls
```

behind the scenes:



hands-on slide: first program

```
login@allegro:/usr3/home/wammar/corpora/spark-training$ ./spark/bin/spark-shell  --master local[3]
```

```
val data = 1 to 10000                          // scala land (scala cheatsheet)
val distData = sc.parallelize(data)            // How to obtain an RDD? (1st answer)
                                               // sc = Spark Context
                                               // distData is an RDD
val filtered = distData.filter(_ < 10)         // filter an RDD into another RDD?
val scalaArray = filtered.collect()            // back to scala land
```

where did **sc** come from?

Scala

```scala
val spark = new SparkContext()

val lines    = spark.textFile("hdfs://docs/")    // RDD[String]
val nonEmpty = lines.filter(l => l.nonEmpty())   // RDD[String]

val count = nonEmpty.count
```

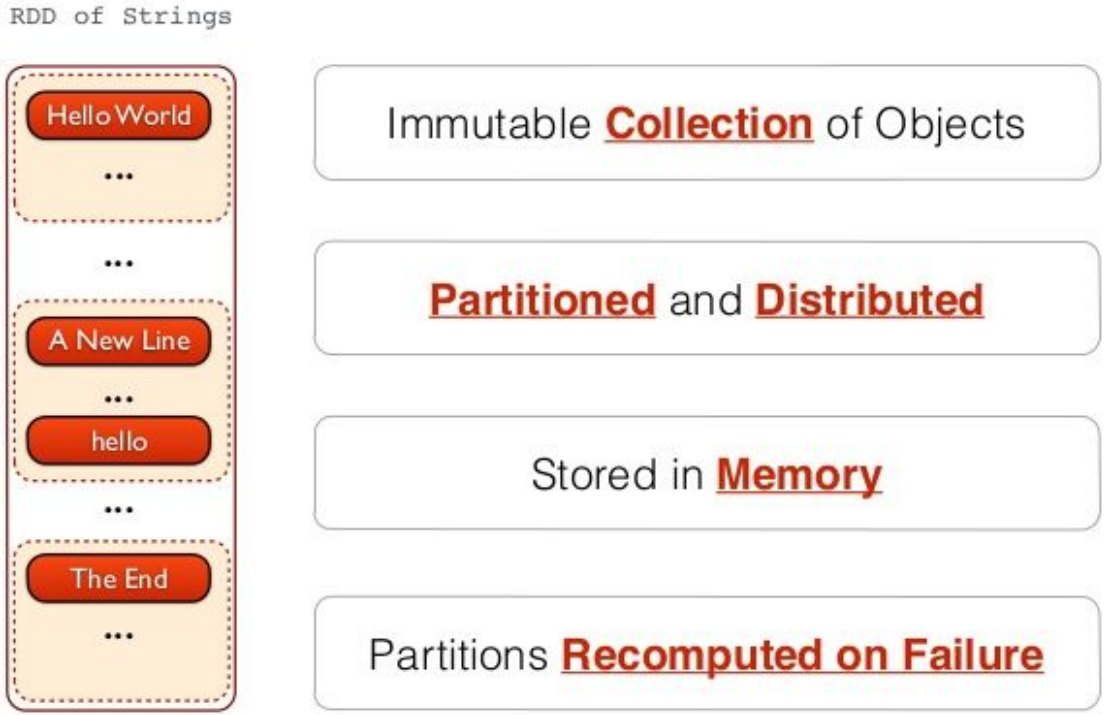
Java 8

```java
SparkContext spark = new SparkContext();

JavaRDD<String> lines    = spark.textFile("hdfs://docs/")
JavaRDD<String> nonEmpty = lines.filter(l -> l.length() > 0);

long count = nonEmpty.count();
```

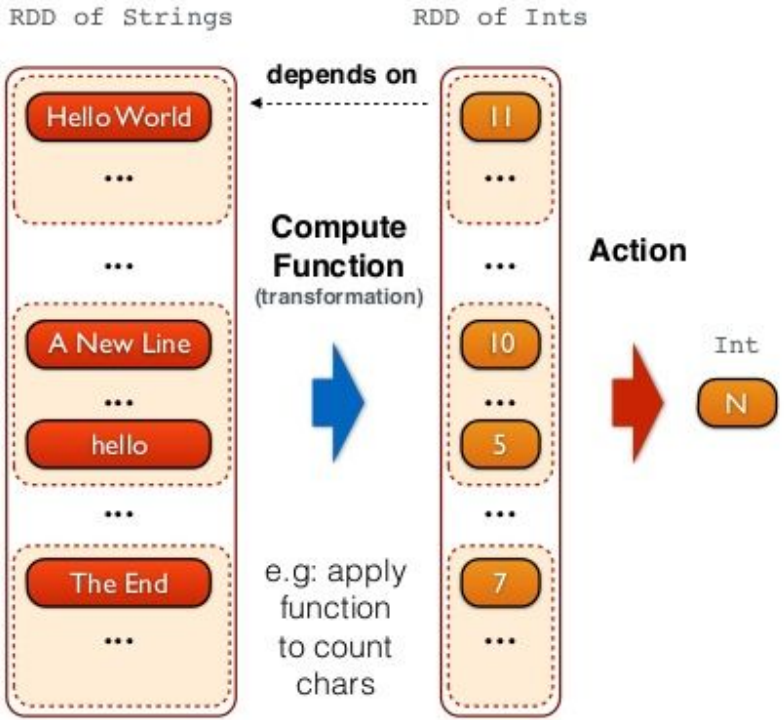
Python

```python
spark = SparkContext()

lines = spark.textFile("hdfs://docs/")
nonEmpty = lines.filter(lambda line: len(line) > 0)

count = nonEmpty.count()
```

what is an RDD anyway?



2 kinds of RDD operations: transformations & actions



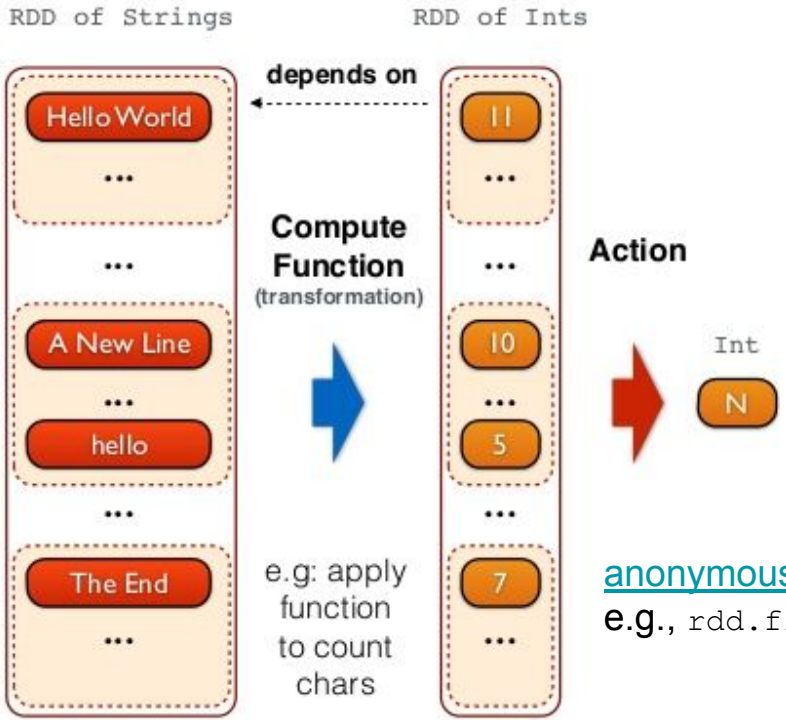
[transformations](#)

```
map(func)
flatMap(func)
filter(func)
groupByKey()
reduceByKey(func)
mapValues(func)

…
```

[actions](#)

```
take(N)
count()
collect()
reduce(func)
takeOrdered(N)
top(N)

…
```

2 kinds of RDD operations: transformations & actions



transformations

```
map(func)
flatMap(func)
filter(func)
groupByKey()
reduceByKey(func)
mapValues(func)
...
```

actions

```
take(N)
count()
collect()
reduce(func)
takeOrdered(N)
top(N)
...
```

anonymous functions

e.g., `rdd.filter(_ < 10)`

static methods of singleton objects

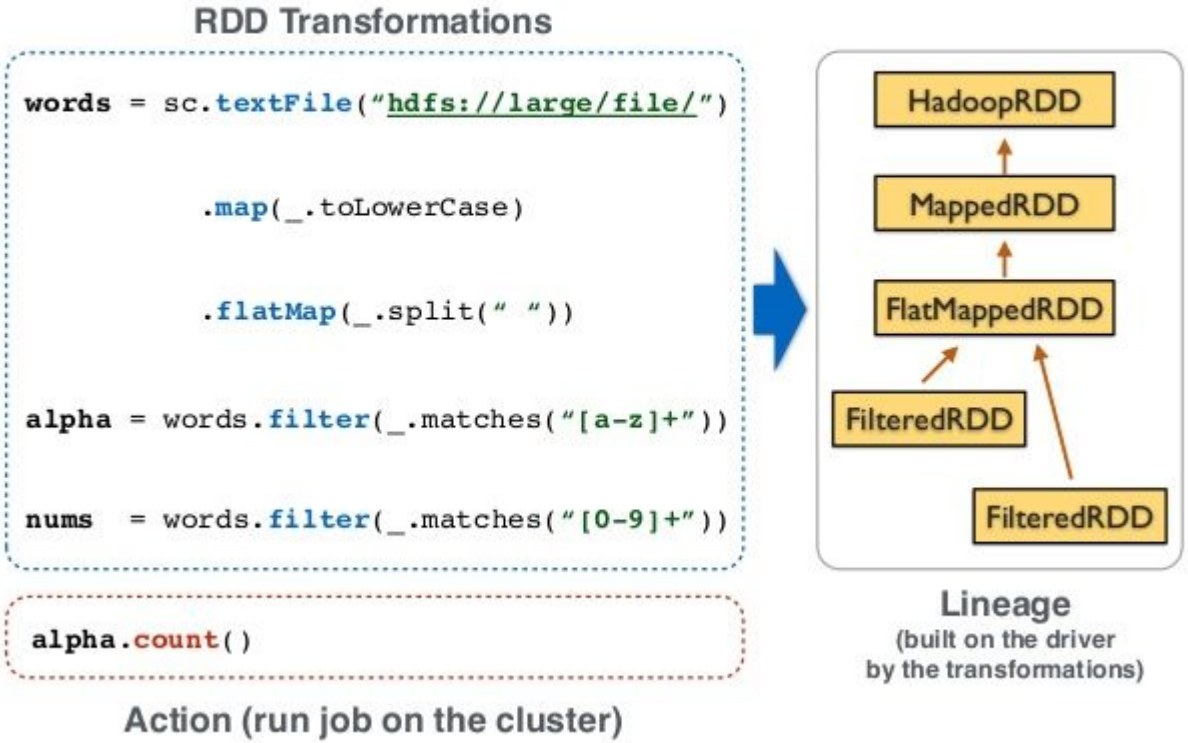
e.g.,

```
object Functors {
  def less_than_10(x: Int) = {x < 10};
}
rdd.filter(Functors.less_than_10)
```

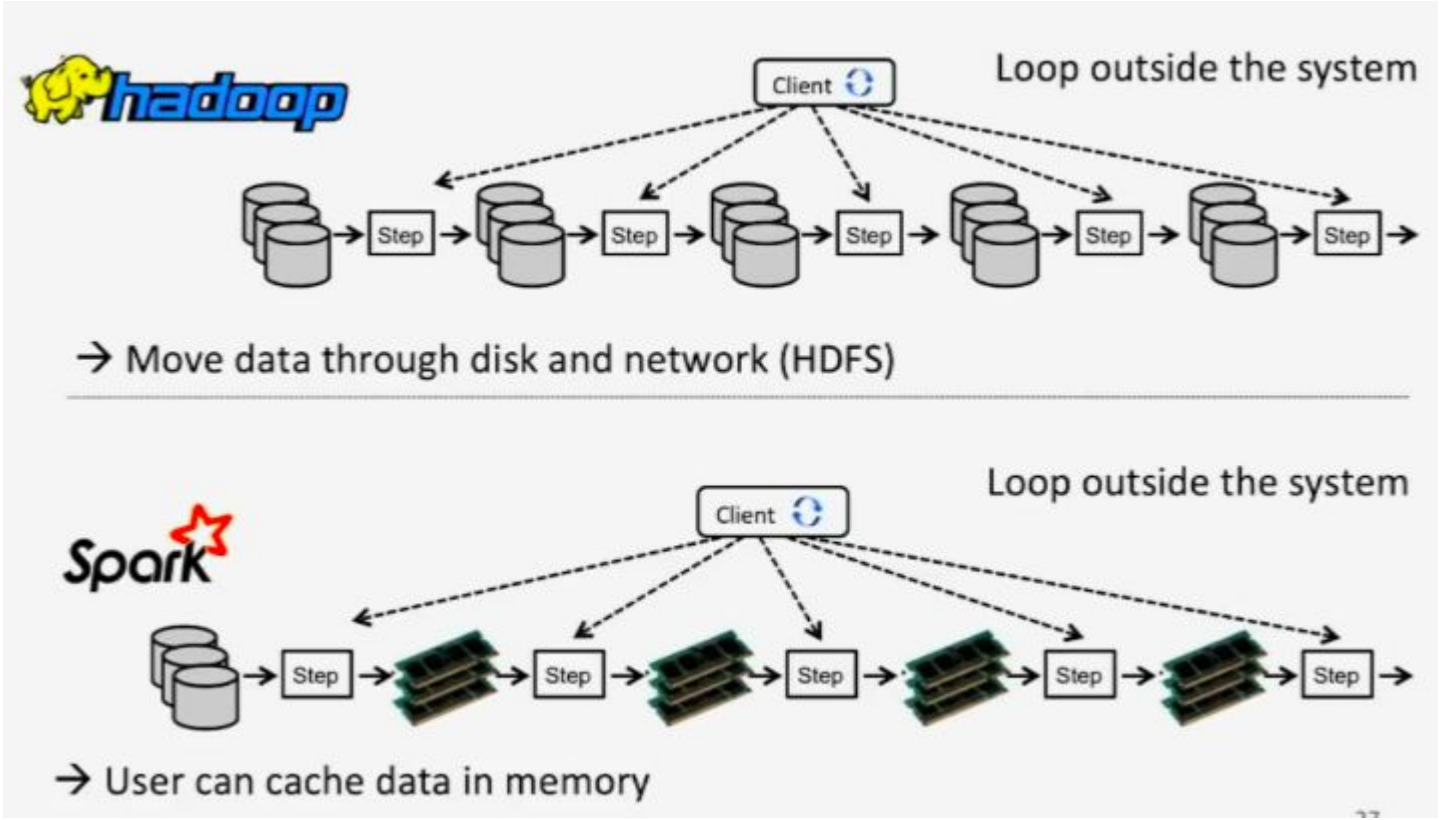
transformations are lazy, actions are not

```
val data = 1 to 10000                          // scala land


val distData = sc.parallelize(data)

                                               // consider it done, sir!

val filtered = distData.filter(_ < 10)

                                               // consider it done, sir!

val scalaArray = filtered.collect()

                                               // done, sir!
```

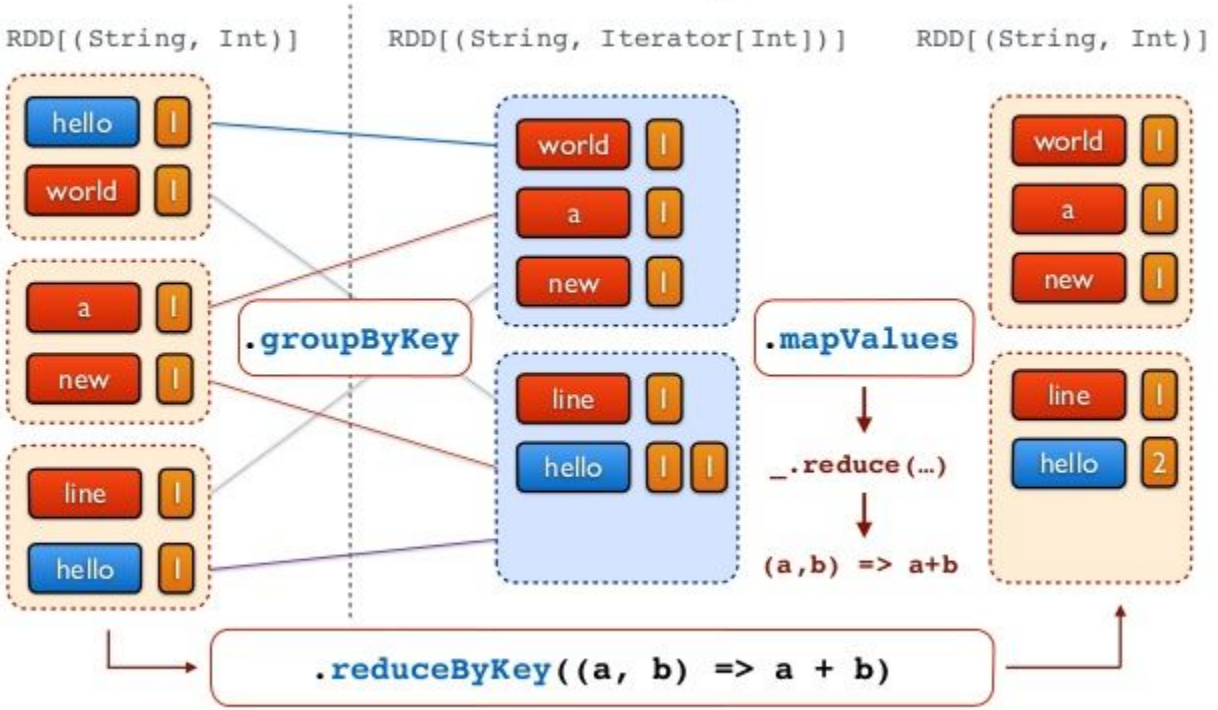
RDD “lineage”


RDD Transformations
words = sc.textFile(“hdfs://large/file/”)
.map(_.toLowerCase)
.flatMap(_.split(“ “))
alpha = words.filter(_.matches(“[a-z]+”))
nums = words.filter(_.matches(“[0-9]+”))
alpha.count()
Action (run job on the cluster)
HadoopRDD
MappedRDD
FlatMappedRDD
FilteredRDD
FilteredRDD
Lineage
(built on the driver
by the transformations)

main advantage over hadoop



caveat: beware of the shuffles!



naughty operations:

- reduceByKey
- repartition
- coalesce
- groupByKey
- cogroup
- join
- (among others)

mind your memory

- broadcast()
- accumulator()
- persist()

hands-on slide: word2vec

```
login@allegro:/usr3/home/wammar/corpora/spark-training$ ./spark/bin/spark-shell  --master local[3]

import org.apache.spark._
import org.apache.spark.rdd._
import org.apache.spark.SparkContext._
import org.apache.spark.mllib.feature.Word2Vec

val input = sc.textFile("data/english-text").map(line => line.toLowerCase().split(" ").toSeq)

val word2vec = new Word2Vec()

val model = word2vec.fit(input)

val synonyms = model.findSynonyms("china", 40)

for((synonym, cosineSimilarity) <- synonyms) {
  println(s"$synonym $cosineSimilarity")
}
```

hands-on slide: character-based langid

```
login@allegro:/usr3/home/wammar/corpora/spark-training$ ./spark/bin/spark-shell  --master local[3]

// read data. Each line looks like: en:perisher ||| clusterid508675

val lines = sc.textFile("data/multilingual_words").sample(false, 0.01, 0)

val wordToLang = lines.map(line => line.split(" ")(0))

                      .map(qualified => (qualified.substring(3), qualified.substring(0,2)))

wordToLang.cache()

val broadcastLangs = sc.broadcast(

                     wordToLang.map( wordLangPair => wordLangPair._2 )

                               .distinct()

                               .collect())
```

```
// create a labeled data set for MLlib

import org.apache.spark.mllib.classification.NaiveBayes
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint

import org.apache.spark.mllib.feature.HashingTF

val broadcastTransformer = sc.broadcast(new HashingTF(10000))

val labeledData = wordToLang.map(

                        wordLangPair => LabeledPoint(

                              1.0 * broadcastLangs.value.indexOf(wordLangPair._2),

                              broadcastTransformer.value.transform(wordLangPair._1.sliding(4).toSeq))
                        )
```

```
val model = NaiveBayes.train(labeledData, lambda = 1.0)

broadcastLangs.value(model.predict(broadcastTransformer.value.transform("travail".sliding(4).toSeq)
).toInt)
```