



Introduction

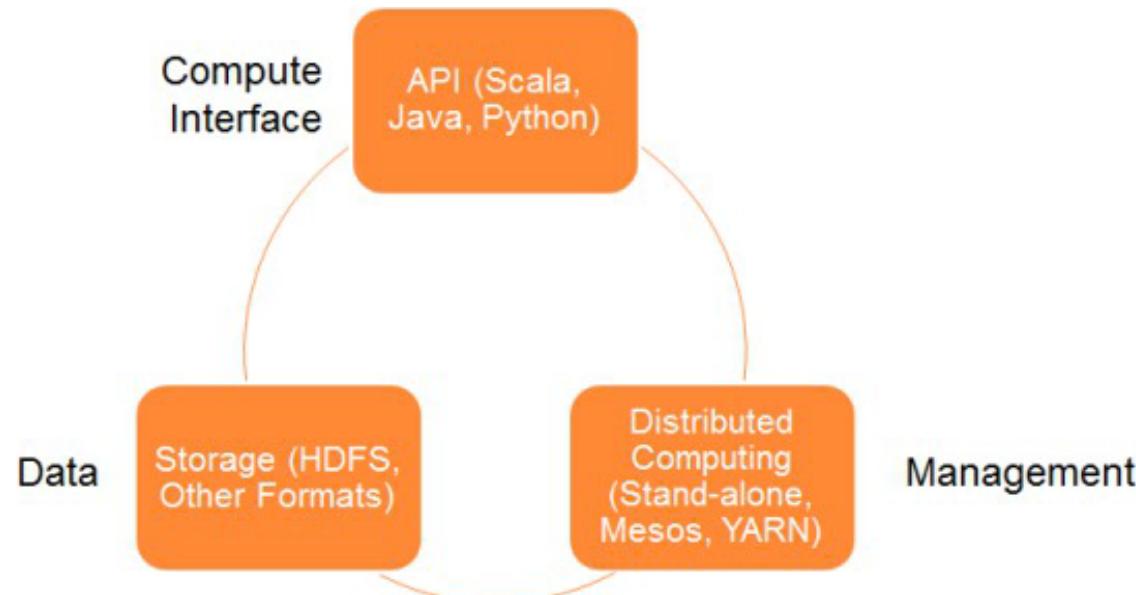
A fast and general engine for large scale data processing



An open source big data processing framework built around speed, ease of use, and sophisticated analytics. Spark enables applications in Hadoop clusters to run up to 100 times faster in memory and 10 times faster even when running on disk.

What is Spark ?

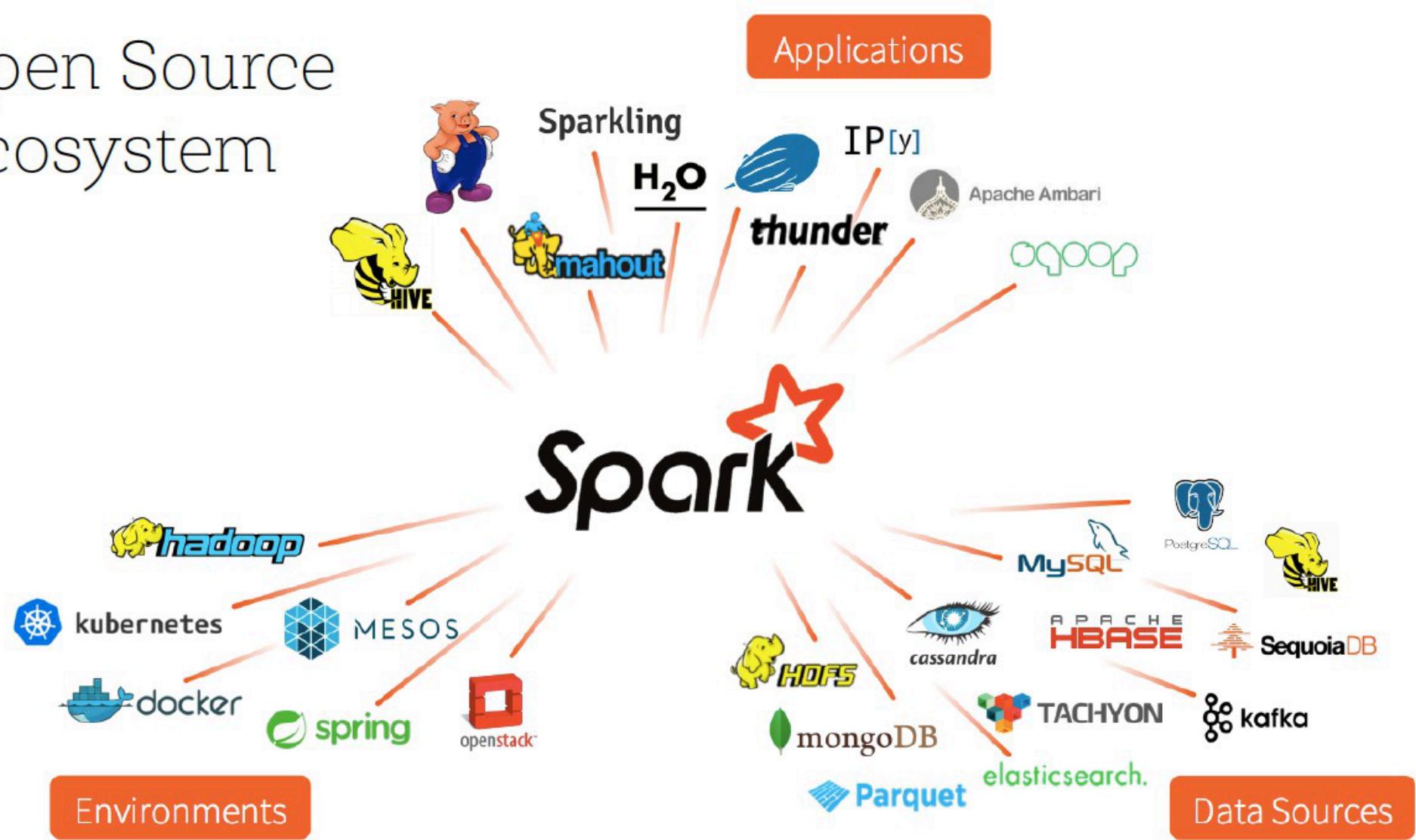
Framework for distributed processing.
In-memory, fault tolerant data structures
Flexible APIs in Scala, Java, Python, SQL, R
Open source



Why Spark ?

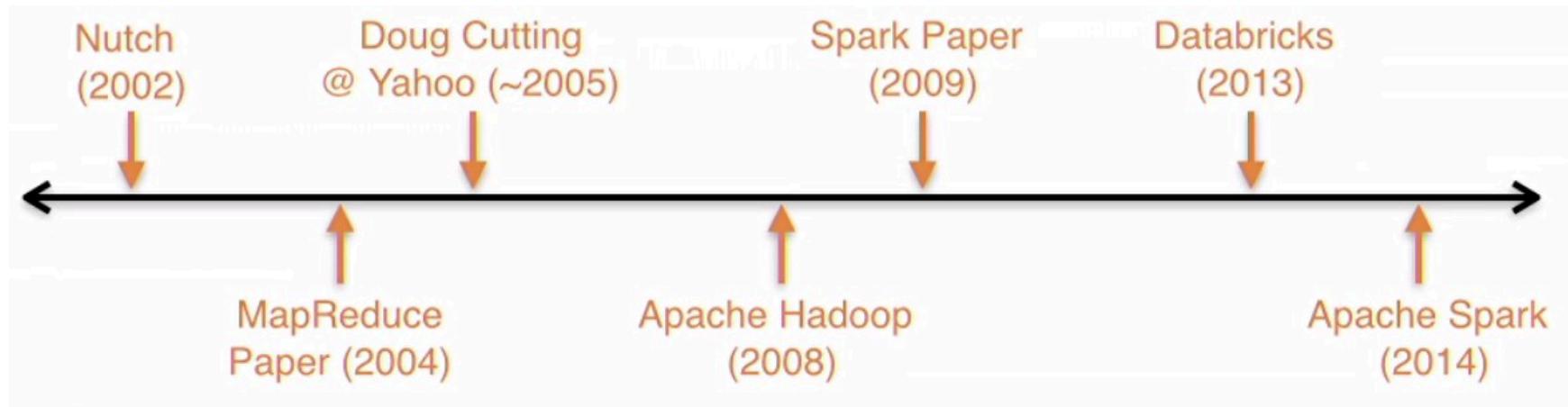
- Handle Petabytes of data
- Significant faster than MapReduce
- Simple and intuitive APIs
- General framework
 - Runs anywhere
 - Handles (most) any I/O
 - Interoperable libraries for specific use-cases

Open Source Ecosystem



Spark: History

- Founded by AMPIlab, UC Berkeley
- Created by Matei Zaharia (PhD Thesis)
- Maintained by Apache Software Foundation
- Commercial support by Databricks





PRODUCT SPARK SOLUTIONS CUSTOMERS COMPANY BLOG RESOURCES

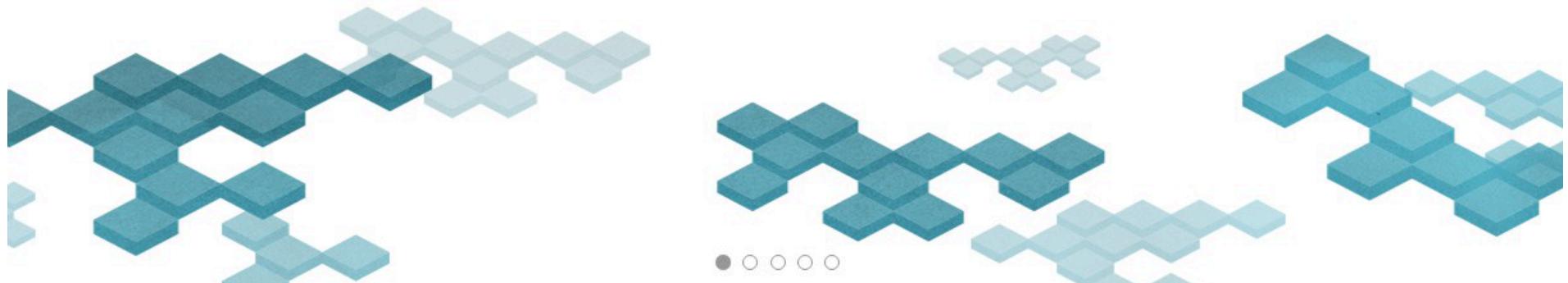
Partners Training [Sign Up](#) 

databricks™

making big data simple

Data Science made easy, from ingest to production. Powered by Apache Spark™.

[SIGN UP FOR A 14-DAY FREE TRIAL](#)

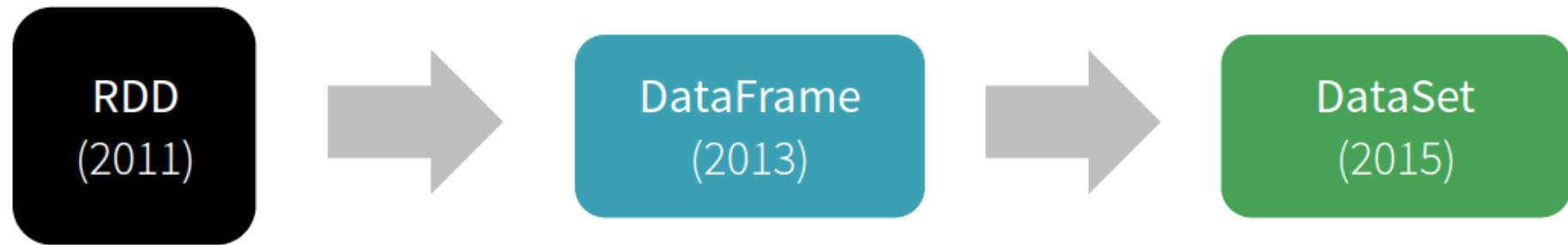


● ○ ○ ○ ○

[LEARN SPARK](#)
Join the Community Edition Beta waitlist >

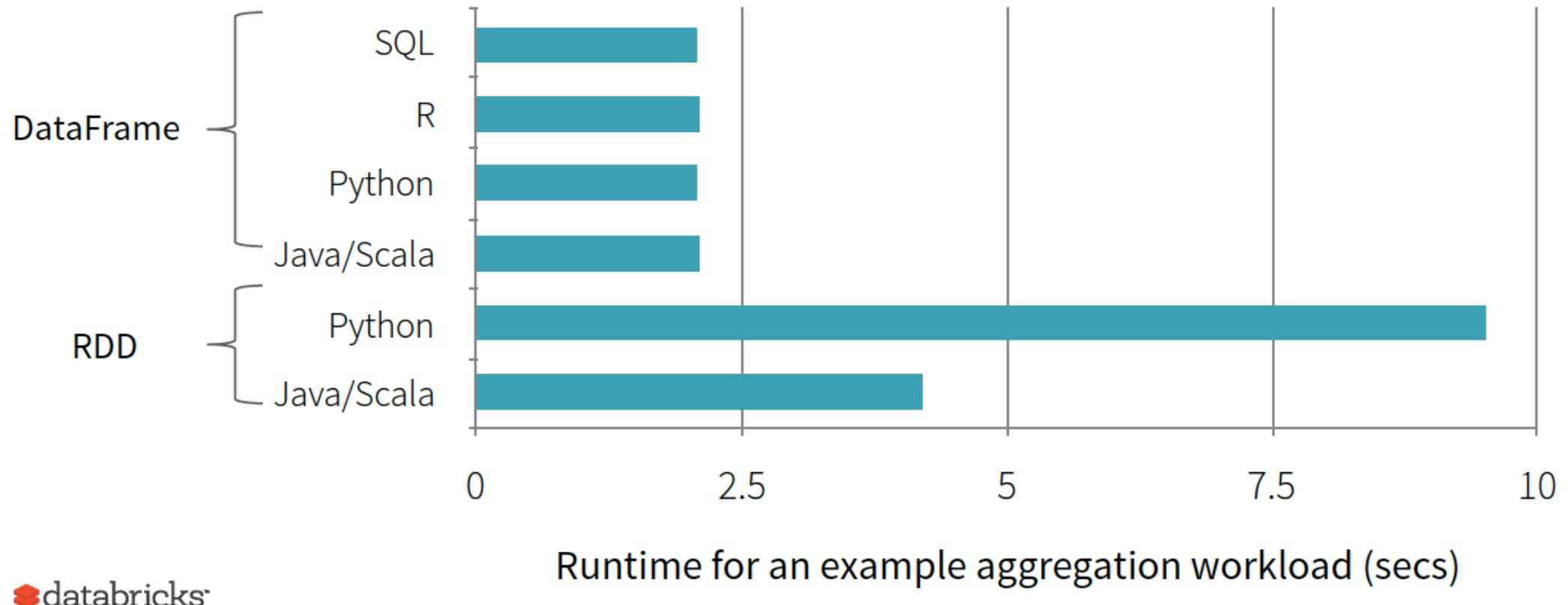
[Start a free trial](#) 

History of Spark APIs



- Distribute collection of JVM objects
 - Functional Operators (map, filter, etc.)
- Distribute collection of Row objects
 - Expression-based operations and UDFs
 - Logical plans and optimizer
 - Fast/efficient internal representations
- Internally rows, externally JVM objects
 - “Best of both worlds”
type safe + fast

Benefit of Logical Plan: Performance Parity Across Languages



 databricks

What is a RDD ?

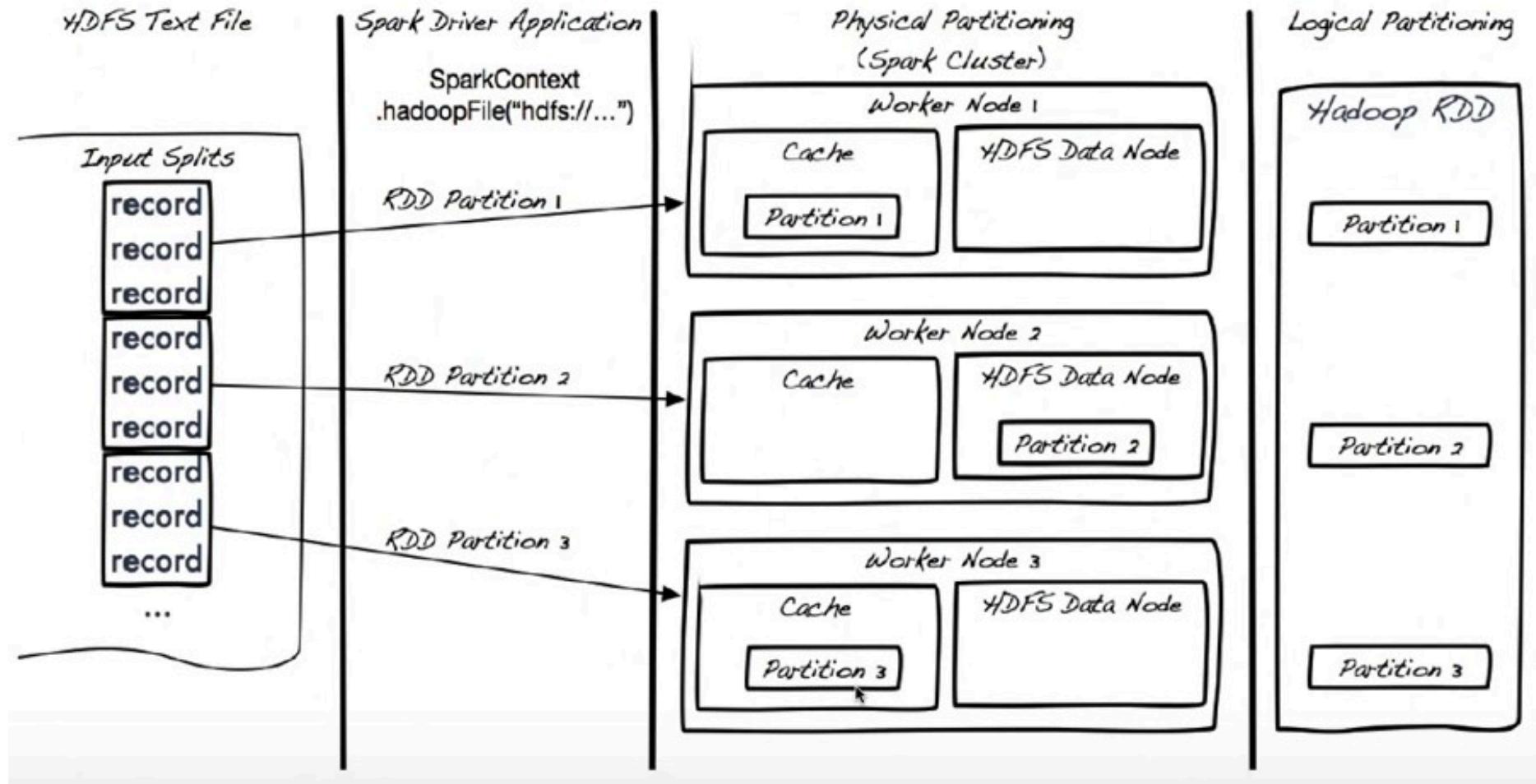
Resilient: if the data in memory (or on a node) is lost, it can be recreated.

Distributed: data is chunked into partitions and stored in memory across the cluster.

Dataset: initial data can come from a table or be created programmatically

RDD Creation

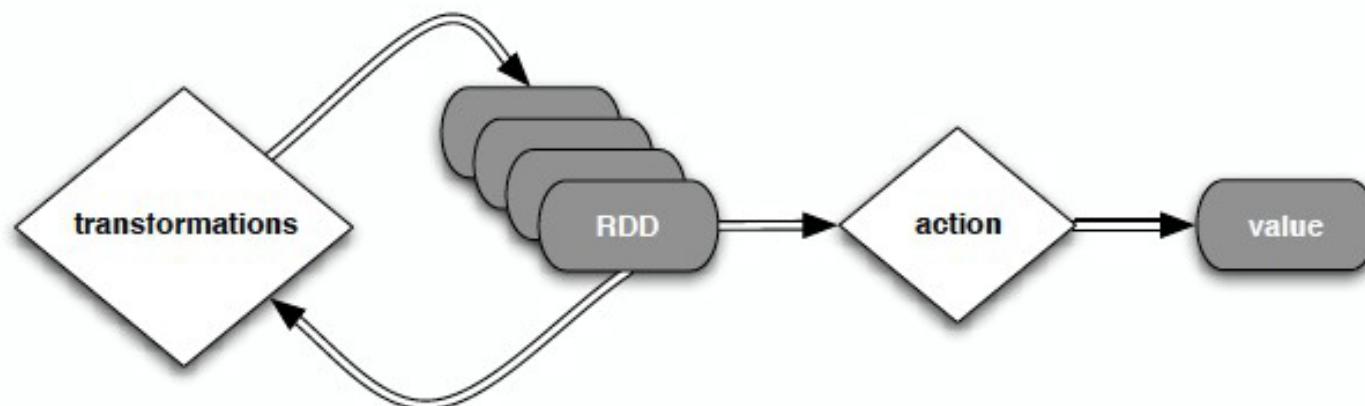
`hdfsData = sc.textFile("hdfs://data.txt")`



RDD: Operations

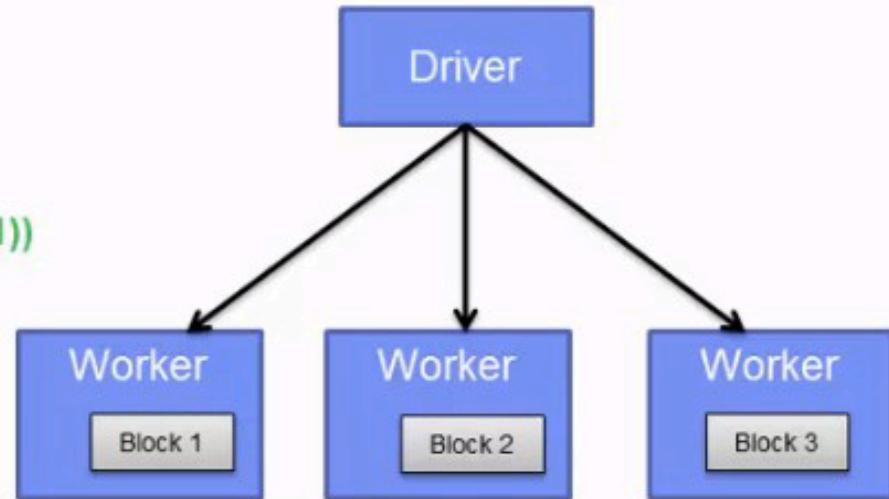
Transformations: transformations are lazy (not computed immediately)

Actions: the transformed RDD gets recomputed when an action is run on it (default)



What happens when an action is executed

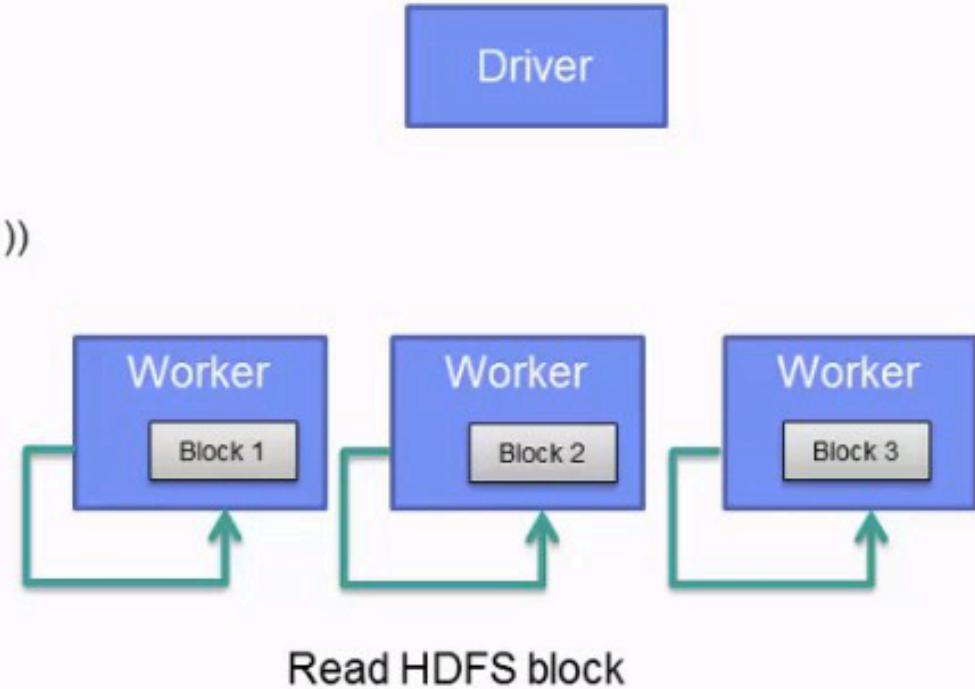
```
// Creating the RDD  
  
val logFile = sc.textFile("hdfs://...")  
  
// Transformations  
  
val errors = logFile.filter(_.startsWith("ERROR"))  
val messages = errors.map(_.split("\t")).map(r => r(1))  
  
//Cache  
messages.cache()  
  
// Actions  
messages.filter(_.contains("mysql")).count()  
messages.filter(_.contains("php")).count()
```



Driver sends the code to be
executed on each block

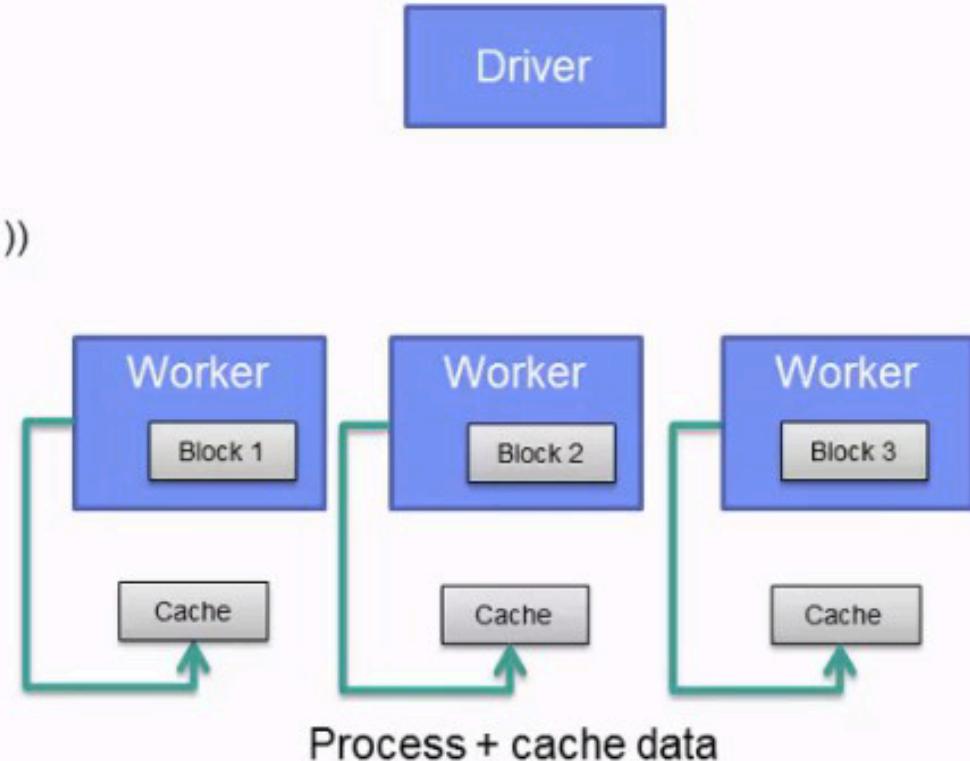
What happens when an action is executed

```
// Creating the RDD  
  
val logFile = sc.textFile("hdfs://...")  
  
// Transformations  
  
val errors = logFile.filter(_.startsWith("ERROR"))  
  
val messages = errors.map(_.split("\t")).map(r => r(1))  
  
//Caching  
  
messages.cache()  
  
// Actions  
  
messages.filter(_.contains("mysql")).count()  
messages.filter(_.contains("php")).count()
```



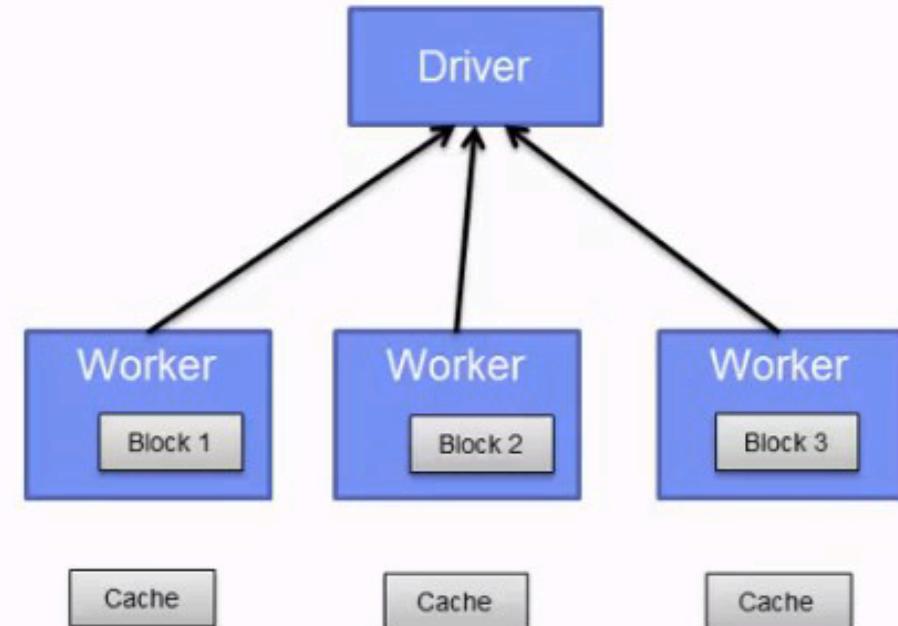
What happens when an action is executed

```
// Creating the RDD  
  
val logFile = sc.textFile("hdfs://...")  
  
// Transformations  
  
val errors = logFile.filter(_.startsWith("ERROR"))  
  
val messages = errors.map(_.split("\t")).map(r => r(1))  
  
//Caching  
  
messages.cache()  
  
// Actions  
  
messages.filter(_.contains("mysql")).count()  
  
messages.filter(_.contains("php")).count()
```



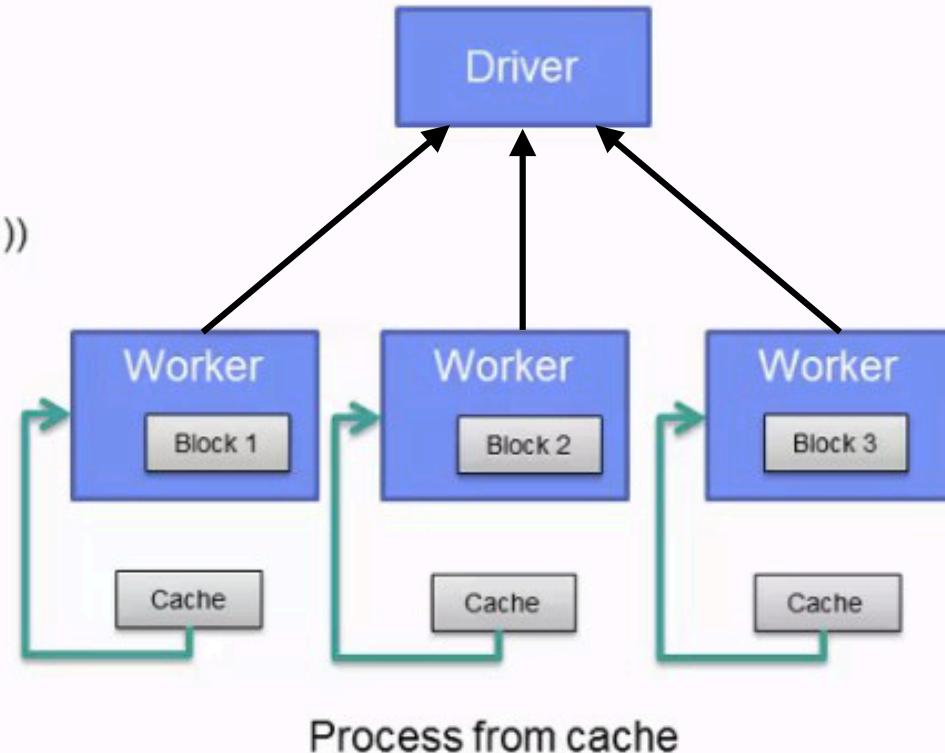
What happens when an action is executed

```
// Creating the RDD
val logFile = sc.textFile("hdfs://...")
// Transformations
val errors = logFile.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
//Caching
messages.cache()
// Actions
messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```



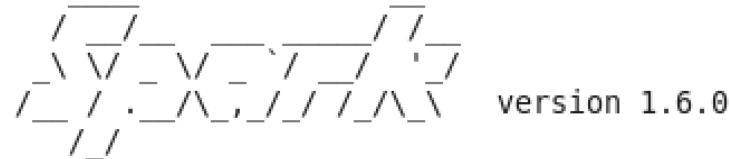
What happens when an action is executed

```
// Creating the RDD  
  
val logFile = sc.textFile("hdfs://...")  
  
// Transformations  
  
val errors = logFile.filter(_.startsWith("ERROR"))  
val messages = errors.map(_.split("\t")).map(r => r(1))  
  
//Caching  
  
messages.cache()  
  
// Actions  
  
messages.filter(_.contains("mysql")).count()  
  
messages.filter(_.contains("php")).count()
```



What happens when an action is executed

```
16/11/01 01:03:54 INFO storage.BlockManagerMaster: Registered BlockManager
Welcome to
```



version 1.6.0

```
Using Python version 2.6.6 (r266:84292, Jul 23 2015 15:22:56)
SparkContext available as sc, HiveContext available as sqlContext.
>>> rdd1 = sc.parallelize(range(1, 11))                                [1,2,3,4,5,6,7,8,9,10]
>>> rdd1 = rdd1.filter(lambda n : n % 2 == 0)                          [2,4,6,8,10]
>>> rdd1 = rdd1.map(lambda n : n ** 2)                                 [4, 16, 36, 64, 100]
>>> rdd1 = rdd1.map(lambda n : n / 2)                                 [2, 8, 18, 32, 50]
>>> ans = rdd1.reduce(lambda a, b : a + b)
16/11/01 01:04:52 INFO scheduler.DAGScheduler: ResultStage 0 (reduce at <stdin>:
1) finished in 0.360 s
16/11/01 01:04:52 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 0.0
(TID 0) in 329 ms on localhost (1/1)
16/11/01 01:04:52 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 0.0, whose t
asks have all completed, from pool
16/11/01 01:04:52 INFO scheduler.DAGScheduler: Job 0 finished: reduce at <stdin>
:1, took 0.990547 s
>>> print(ans)
110
```

Spark: Transformation



<i>transformation</i>	<i>description</i>
map(<i>func</i>)	return a new distributed dataset formed by passing each element of the source through a function <i>func</i>
filter(<i>func</i>)	return a new dataset formed by selecting those elements of the source on which <i>func</i> returns true
flatMap(<i>func</i>)	similar to map, but each input item can be mapped to 0 or more output items (so <i>func</i> should return a Seq rather than a single item)
sample(<i>withReplacement</i>, <i>fraction</i>, <i>seed</i>)	sample a fraction <i>fraction</i> of the data, with or without replacement, using a given random number generator <i>seed</i>
union(<i>otherDataset</i>)	return a new dataset that contains the union of the elements in the source dataset and the argument
distinct([<i>numTasks</i>]))	return a new dataset that contains the distinct elements of the source dataset

Spark: Transformation



transformation	description
groupByKey([numTasks])	when called on a dataset of (K, V) pairs, returns a dataset of (K, Seq[V]) pairs
reduceByKey(func, [numTasks])	when called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function
sortByKey([ascending], [numTasks])	when called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean ascending argument
join(otherDataset, [numTasks])	when called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key
cogroup(otherDataset, [numTasks])	when called on datasets of type (K, V) and (K, W), returns a dataset of (K, Seq[V], Seq[W]) tuples – also called groupWith
cartesian(otherDataset)	when called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements)

Single RDD Transformation

filter females to analyze female buying patterns

male1, male2, female1 -> female1

map squared values

2, 5, 6 -> 4, 25, 36

flatMap to break up a sentence into words

my name is ray -> my, name, is, ray

find the **distinct** values in a dataset

apple, apple, banana -> apple, banana

sample two values at random

apple, banana, guava -> banana, apple

Multiple RDD Transformation

union

apple, orange, banana, guava,
banana, pear

intersection

banana

subtract anything shown in Dataset B
from Dataset A

apple, orange

cartesian (every possible pair combo)

(apple, guava), (apple, banana), ...

Dataset A

apple
orange
banana

Dataset B

guava
banana
pear

Pair RDD Transformation

- reduceByKey
- groupByKey
- combineByKey
- mapValues
- flatMapValues
- keys
- values
- subtractByKey
- join
- rightOuterJoin
- leftOuterJoin
- cogroup
- sortByKey

Spark:Actions



action	description
reduce(func)	aggregate the elements of the dataset using a function <code>func</code> (which takes two arguments and returns one), and should also 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 – 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 <code>take(1)</code>
take(n)	return an array with the first <code>n</code> elements of the dataset – currently not executed in parallel, instead the driver program computes all the elements
takeSample(withReplacement, fraction, seed)	return an array with a random sample of <code>num</code> elements of the dataset, with or without replacement, using the given random number generator <code>seed</code>

Spark:Actions



action	description
saveAsTextFile(path)	write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call <code>toString</code> on each element to convert it to a line of text in the file
saveAsSequenceFile(path)	write the elements of the dataset as a Hadoop SequenceFile in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. Only available on RDDs of key-value pairs that either implement Hadoop's Writable interface or are implicitly convertible to Writable (Spark includes conversions for basic types like Int, Double, String, etc).
countByKey()	only available on RDDs of type <code>(K, V)</code> . Returns a 'Map' of <code>(K, Int)</code> pairs with the count of each key
foreach(func)	run a function <code>func</code> on each element of the dataset – usually done for side effects such as updating an accumulator variable or interacting with external storage systems

Transformations

```
>>> nums = sc.parallelize([1,2,3])  
>>> squared = nums.map(lambda x : x*x)  
>>> even = squared.filter(lambda x: x%2 == 0)  
>>> evens = nums.flatMap(lambda x: range(x))
```

Actions

```
>>> nums = sc.parallelize([1,2,3])  
>>> nums.collect()  
>>> nums.take(2)  
>>> nums.count()  
>>> nums.reduce(lambda:x, y:x+y)  
>>> nums.saveAsTextFile("hdfs://user/cloudera/output/test")
```

Spark Programming

Functional tools in Python

map

filter

reduce

lambda

IterTools

Chain, flatmap

map

```
>>> a= [1,2,3]
```

```
>>> def add1(x) : return x+1
```

```
>>> map(add1, a)
```

Result: [2,3,4]

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

```
val result = input.map(x => x*x)
```

```
println(result.collect().mkString(","))
```

Result: [1,4,9,16]

Filter

```
>>> a= [1,2,3,4]  
>>> def isOdd(x) : return x%2==1  
>>> filter(isOdd, a)
```

Result: [1,3]

Reduce

```
>>> a= [1,2,3,4]  
>>> def add(x,y) : return x+y  
>>> reduce(add, a)
```

Result: 10

lambda

```
>>> (lambda x: x + 1)(3)
```

Result: 4

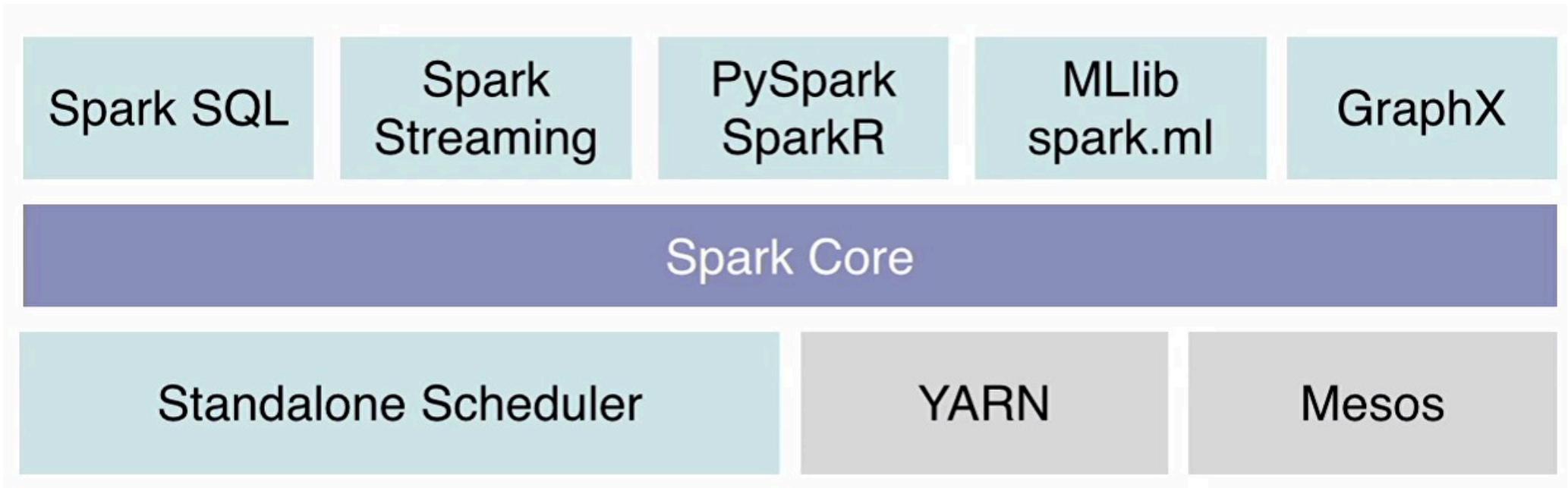
```
>>> map((lambda x: x + 1), [1,2,3])
```

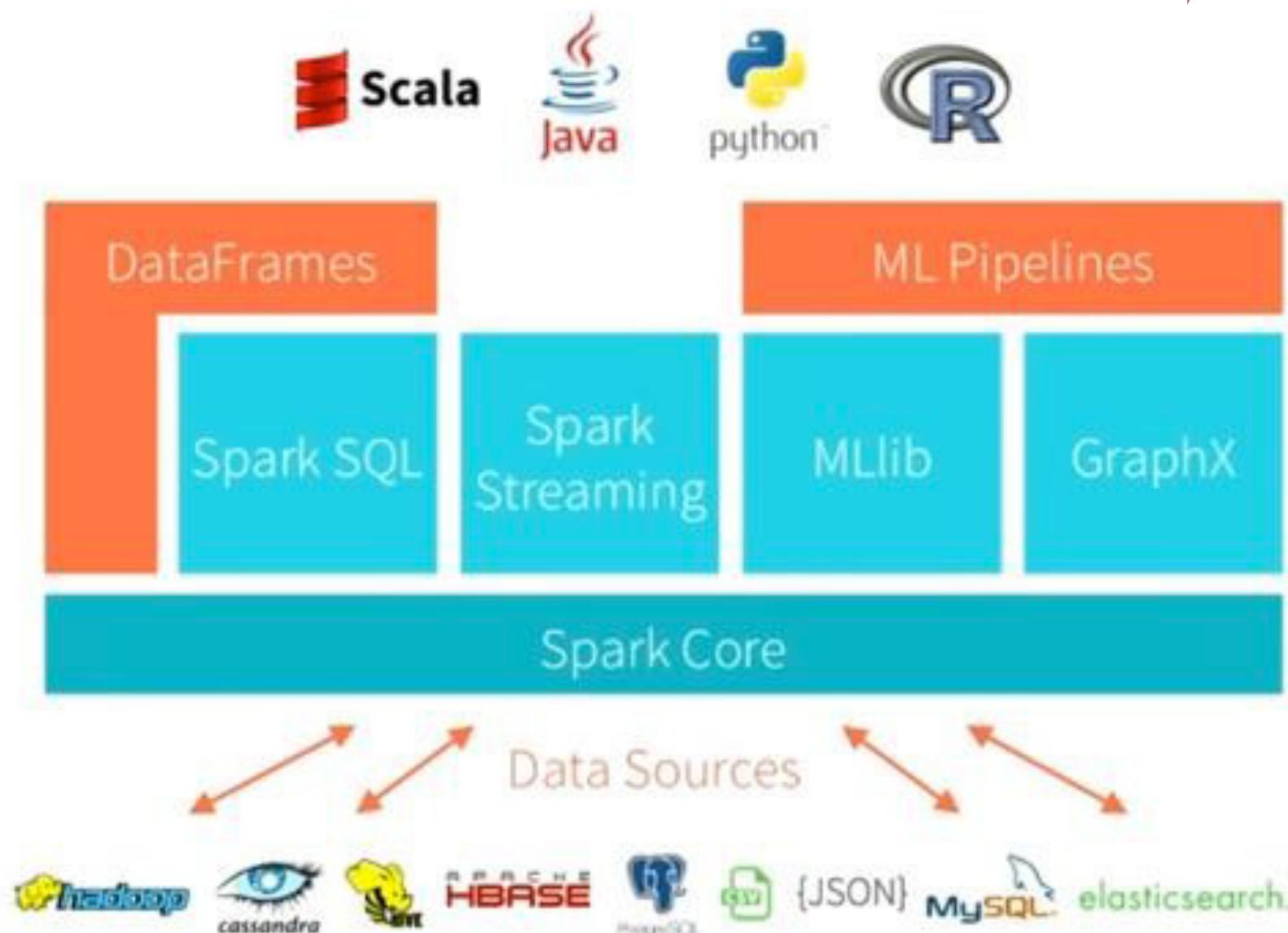
Result: [2,3,4]

Exercises

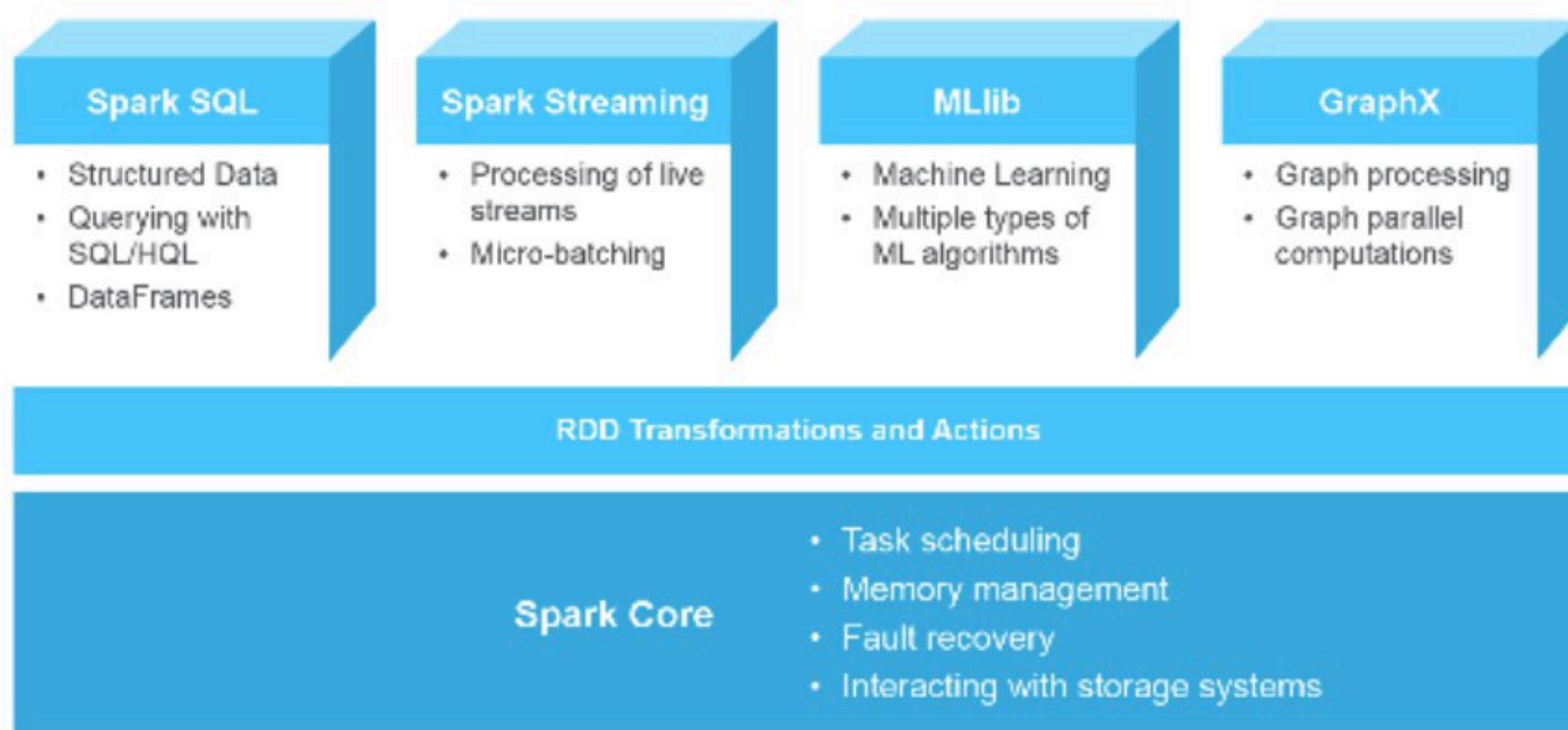
- (lambda x: 2*x)(3) => ?
- map(lambda x: 2*x, [1,2,3]) =>
- map(lambda t: t[0], [(1,2), (3,4), (5,6)]) =>
- reduce(lambda x,y: x+y, [1,2,3]) =>
- reduce(lambda x,y: x+y, map(lambda t: t[0], [(1,2), (3,4), (5,6)]))=>

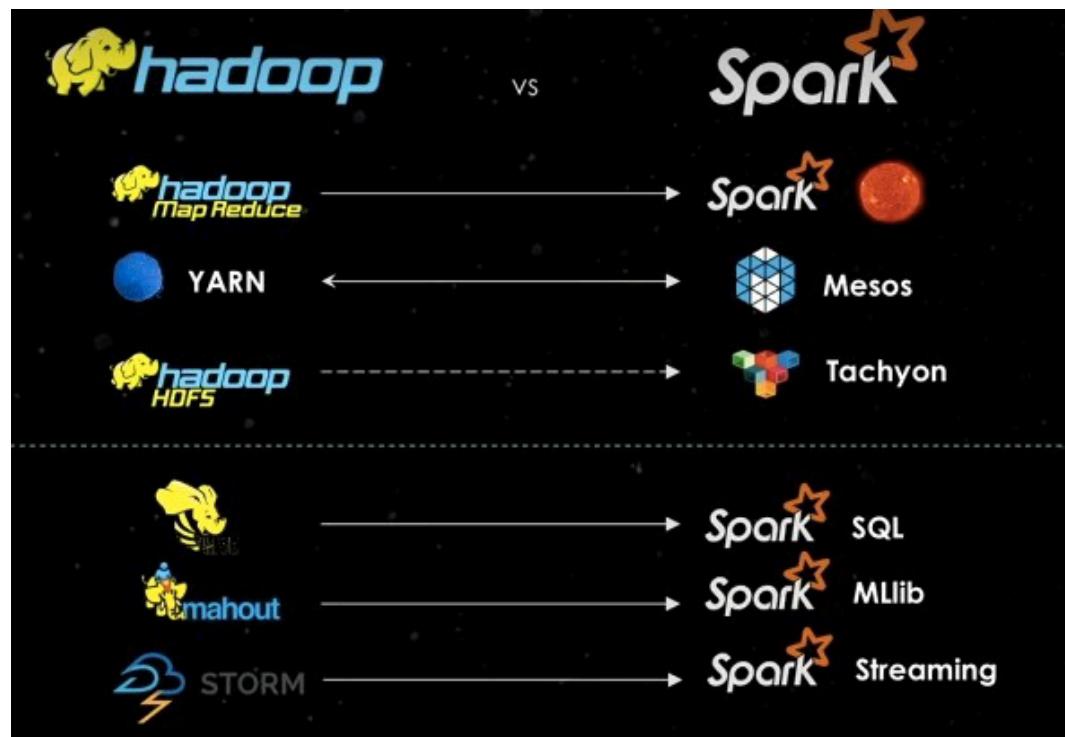
Spark Platform



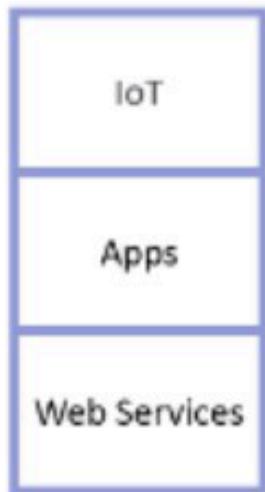


Spark Platform

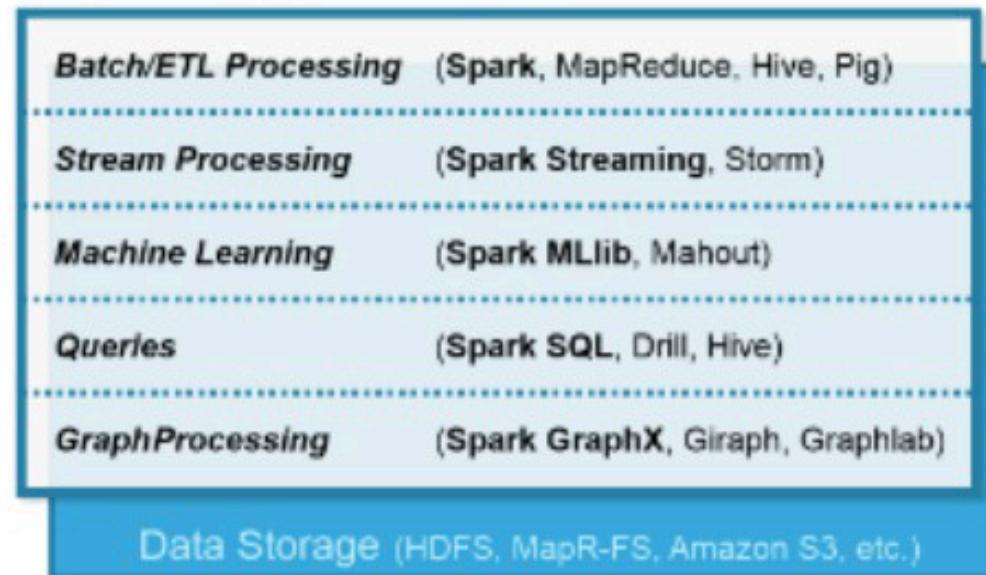




Data Sources



Big Data Application Stack



User



Do we still need Hadoop?

Yes, why Hadoop?

- HDFS
- YARN
- MapReduce is mature and still be appropriate for certain workloads
- Other services: Sqoop, Flume, etc.

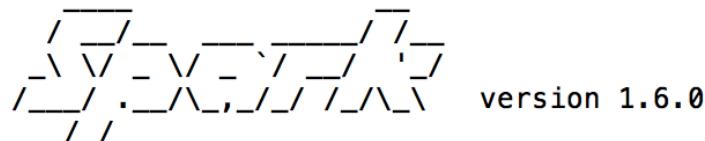
But you can still use other resource management, storages

- Spark Standalone
- Amazon S3
- Mesos

Start Spark-shell

\$spark-shell

```
[root@quickstart 201402_babs_open_data]# spark-shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
Welcome to
```



```
Using Scala version 2.10.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_67)
Type in expressions to have them evaluated.
Type 'help' for more information.
```

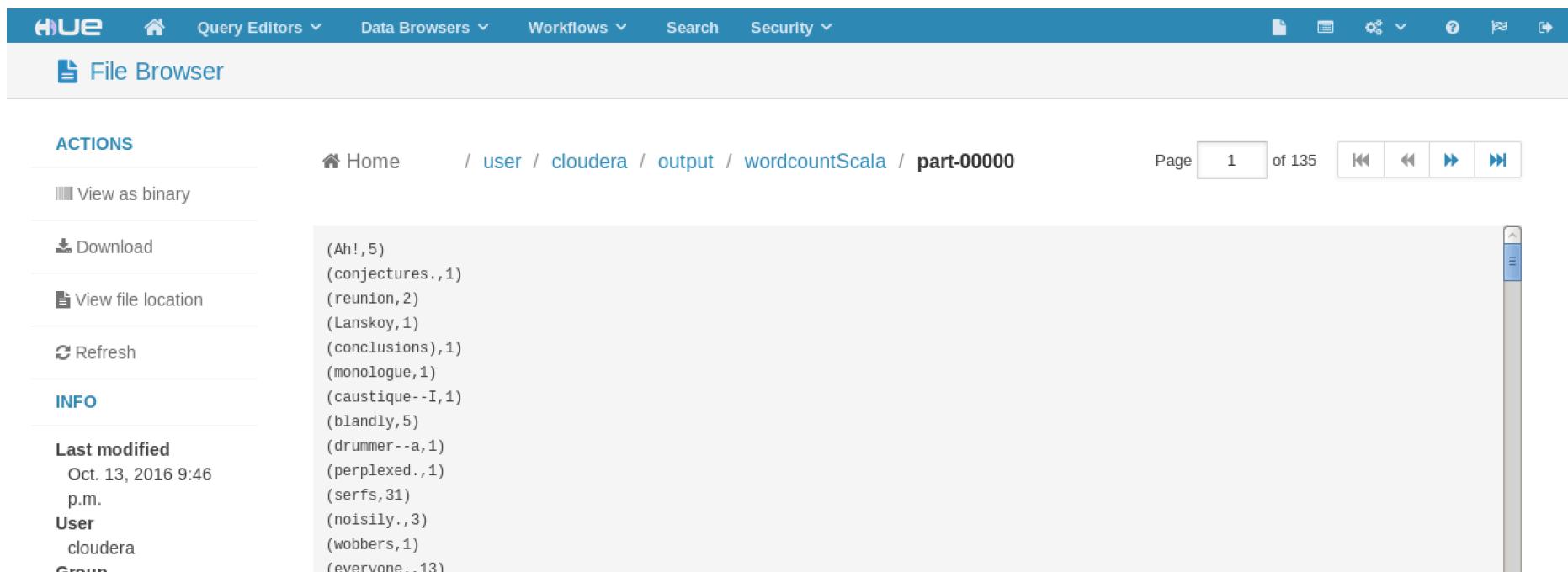
Testing SparkContext

scala> sc

```
scala> sc
res0: org.apache.spark.SparkContext = org.apache.spark.SparkContext@18c07e25
```

Spark Program in Scala: WordCount

```
scala> val file = sc.textFile("hdfs://user/cloudera/input/PG2600.txt")
scala> val wc = file.flatMap(l => l.split(" ")).map(word =>(word,
1)).reduceByKey(_ + _)
scala> wc.saveAsTextFile("hdfs://user/cloudera/output/wordcountScala")
```



The screenshot shows the Hue File Browser interface. The top navigation bar includes links for HUE, Home, Query Editors, Data Browsers, Workflows, Search, and Security. Below the navigation is a toolbar with icons for file operations like upload, download, and search. The main area is titled "File Browser" and displays a list of files. On the left, there's a sidebar with "ACTIONS" and "INFO" sections. The "ACTIONS" section contains links for "View as binary", "Download", "View file location", and "Refresh". The "INFO" section shows the file was last modified on Oct. 13, 2016 at 9:46 p.m. by user cloudera. The main content area shows the output of the WordCount program as a list of word counts:

Word	Count
(Ah!,	5
(conjectures.,	1
(reunion,	2
(Lanskoy,	1
(conclusions,	1
(monologue,	1
(caustique--I,	1
(blandly,	5
(drummer--a,	1
(perplexed.,	1
(serfs,	31
(noisily.,	3
(wobbers,	1
(everyone..	13

Spark Program in Python: WordCount



```
$ pyspark
```

```
>>> from operator import add  
>>> file = sc.textFile("hdfs://user/cloudera/input/PG2600.txt")  
>>> wc = file.flatMap(lambda x: x.split(' ')).map(lambda x:(x,  
1)).reduceByKey(add)  
>>> wc.saveAsTextFile("hdfs://user/cloudera/output/  
wordcountPython")
```

The screenshot shows the Hue File Browser interface. The top navigation bar includes links for Home, Query Editors, Data Browsers, Workflows, Search, and Security. Below the navigation is a toolbar with various icons. The main area is titled "File Browser" and shows a list of files under the path "/user/cloudera/output/wordcountPython/part-00000". The "ACTIONS" column contains links for "View as binary", "Download", "View file location", and "Refresh". The "INFO" column displays the last modified date as "Oct. 13, 2016 11:41 p.m." and the user as "User". The right side of the screen displays the contents of the file, which is a list of word counts:

(u'', 13598)
(u'considered,', 3)
(u'considered.', 6)
(u'grenadier.', 3)
(u'S--', 2)
(u'grenadier,', 1)
(u'"Fool,', 1)
(u'hanging', 31)
(u'"Fool!', 1)
(u'untold.', 1)
(u'"Peace', 1)
(u'disobeying', 2)

Loading data from MySQL

Download MySQL driver & Start Spark-shell

Open New Terminal

```
$ mkdir spark
```

```
$ cd spark
```

```
$ wget https://github.com/bobbylovelove/movie/trainbigdata/raw/master/  
Spark/mysql-connector-java-5.1.23.jar
```

Running Spark-shell

```
$ spark-shell --jars mysql-connector-java-5.1.23.jar
```

```
...  
16/06/28 15:23:35 WARN shortcircuit.DomainSocketFactory: The short-circuit local rea  
ds feature cannot be used because libhadoop cannot be loaded.  
SQL context available as sqlContext.
```

```
scala> █
```

```
$ scala> :paste
val url="jdbc:mysql://localhost:3306/test_mysql_db"
val username = "root"
val password = "cloudera"
import org.apache.spark.rdd.JdbcRDD
import java.sql.{Connection, DriverManager, ResultSet}
Class.forName("com.mysql.jdbc.Driver").newInstance
val myRDD = new JdbcRDD( sc, () =>
  DriverManager.getConnection(url,username,password) ,
  "SELECT * FROM country_tbl LIMIT ?,?", 0, 5, 2, r =>r.getString("id") + "," +
  r.getString("country"))
myRDD.count
myRDD.foreach(println)
```

Output

```
// Exiting paste mode, now interpreting.  
  
1, USA  
2, CANADA  
4, Brazil  
61, Japan  
65, Singapore  
66, Thailand  
url: String = jdbc:mysql://localhost:3306/test_mysql_db  
username: String = root  
password: String = cloudera  
import org.apache.spark.rdd.JdbcRDD  
import java.sql.{Connection, DriverManager, ResultSet}  
myRDD: org.apache.spark.rdd.JdbcRDD[String] = JdbcRDD[0] at JdbcRDD at <console>  
:39
```

**scala> myRDD.saveAsTextFile("hdfs:///user/cloudera/output/
mysqlFromSpark")**