

L06: Introducing Spark

ANLY 502: Massive Data Fundamentals

Simson Garfinkel

February 27, 2017



GEORGETOWN UNIVERSITY

Outline for today's class

Background:

- Midterm Assessment
- A03 Redux — it should be graded!
- A04 Questions
- Reading Release Notes

Spark 1

- Introducing Spark
- RDDs and Datasets
- Writing Spark programs in Python

Lab 1

Spark 2:

- The State of Spark
- Internet sources for information about Spark.
- Using Spark with jupyter on EMR

Lab 2

We've done a lot in the past six weeks!

What we've done:

- Unix command line
- Amazon Web Services
- Spending real money — \$\$\$
- MapReduce
- mrjob
- Debugging
- Gigabyte-sized data sets
- Data wrangling

Where we're going:

- Spark
- SparkSQL
- Terabyte-sized data sets
- Class Projects

Survey...

Please fill out: <http://bit.ly/21VLQTj>

Responses: <http://bit.ly/2mu8LT1>

Create Cluster - Advanced Options [Go to quick options](#)

Step 1: Software and Steps
Step 2: Hardware
Step 3: General Cluster Settings
Step 4: Security

Software Configuration

Vendor ☒ Amazon ☐ MapR

Release

<input checked="" type="checkbox"/> Hadoop 2.7.3	<input type="checkbox"/> Zeppelin 0.6.2	<input type="checkbox"/> Tez 0.8.4
<input type="checkbox"/> Flink 1.1.4	<input type="checkbox"/> Ganglia 3.7.2	<input type="checkbox"/> HBase 1.2.3
<input checked="" type="checkbox"/> Pig 0.16.0	<input checked="" type="checkbox"/> Hive 2.1.1	<input type="checkbox"/> Presto 0.157.1
<input type="checkbox"/> ZooKeeper 3.4.9	<input type="checkbox"/> Sqoop 1.4.6	<input type="checkbox"/> Mahout 0.12.2
<input checked="" type="checkbox"/> Hue 3.11.0	<input type="checkbox"/> Phoenix 4.7.0	<input type="checkbox"/> Oozie 4.3.0
<input checked="" type="checkbox"/> Spark 2.1.0	<input type="checkbox"/> HCatalog 2.1.1	

Edit software settings (optional) ⓘ

☒ Enter configuration ☐ Load JSON from S3

`classification=configuration-file-name,properties={myKey1=myValue1,myKey2=myValue2}`

Add steps (optional) ⓘ

Step type **Configure**

☐ Auto-terminate cluster after the last step is completed

[Cancel](#) [Next](#)

Please log in to Amazon and
start a 1-node EMR cluster!

emr-5.3.1
Spark 2.1.0

A3 - Any Questions?



<https://pixabay.com/en/student-typing-keyboard-text>

A4

A4 — You have a month for this problem set

Mon Feb 13 — A4 released!

Mon Feb 20 — Holiday (President's Day): Reading and online homework

Mon, Feb 27 — L06: Intro to Apache Spark

Mon, Mar 6 — Holiday (Spring Break, Fri Mar 3 — Sun Mar 12)

Mon, Mar 13 — L07: HBase and Spark SQL

Fri., Mar 17 — A4 Due Today!

Start thinking about your final projects!

- Mon Mar 20 — L08: Hive & Pig (probably will change)
- Tue Mar 22 — Final Project Individual Proposals
- Mon Mar 27 — Scalable Machine Learning with Spark
- Tue Mar 28 — Final Project Group Proposals
- Mon April 3 — L10: Streaming Databases & Graph Databases
- Mon April 10 — No class — Passover & Italy
- Mon April 10 – Fri April 14 — Final Project Online Clinic (it's graded!)
- Mon April 17 — No class Easter Break
- Mon April 24 — L11: NoSQL
- Mon May 1 — L12: Final Project Presentations (last class!!!)
- Wed May 10 — Final Projects Due
- Mon May 15 — Grades due for graduating students (anybody graduating?)

Reading the release notes

Read the documentation! Read the release notes.

Python Release History

- Python 1.0 — January 1994
- Python 2.0 — October 2000
 - Python 2.1* — April 2001
 - Python 2.2* — Dec. 2001
 - Python 2.3* — July 2003
 - Python 2.4* — Nov. 2004
 - Python 2.5* — Sept. 2006
 - Python 2.6* — Oct. 2008
 - Python 2.7* — July 2010
- Python 3.0 — Dec. 2008
 - Python 3.1* — June 2009
 - Python 3.2* — Feb. 2011
 - Python 3.3* — Sept. 2012
 - Python 3.4* — March 2014
 - Python 3.5* — Sept. 2015
 - Python 3.6* — Dec. 2016

Many Python users are still using 2.7! (6.5 years old!)

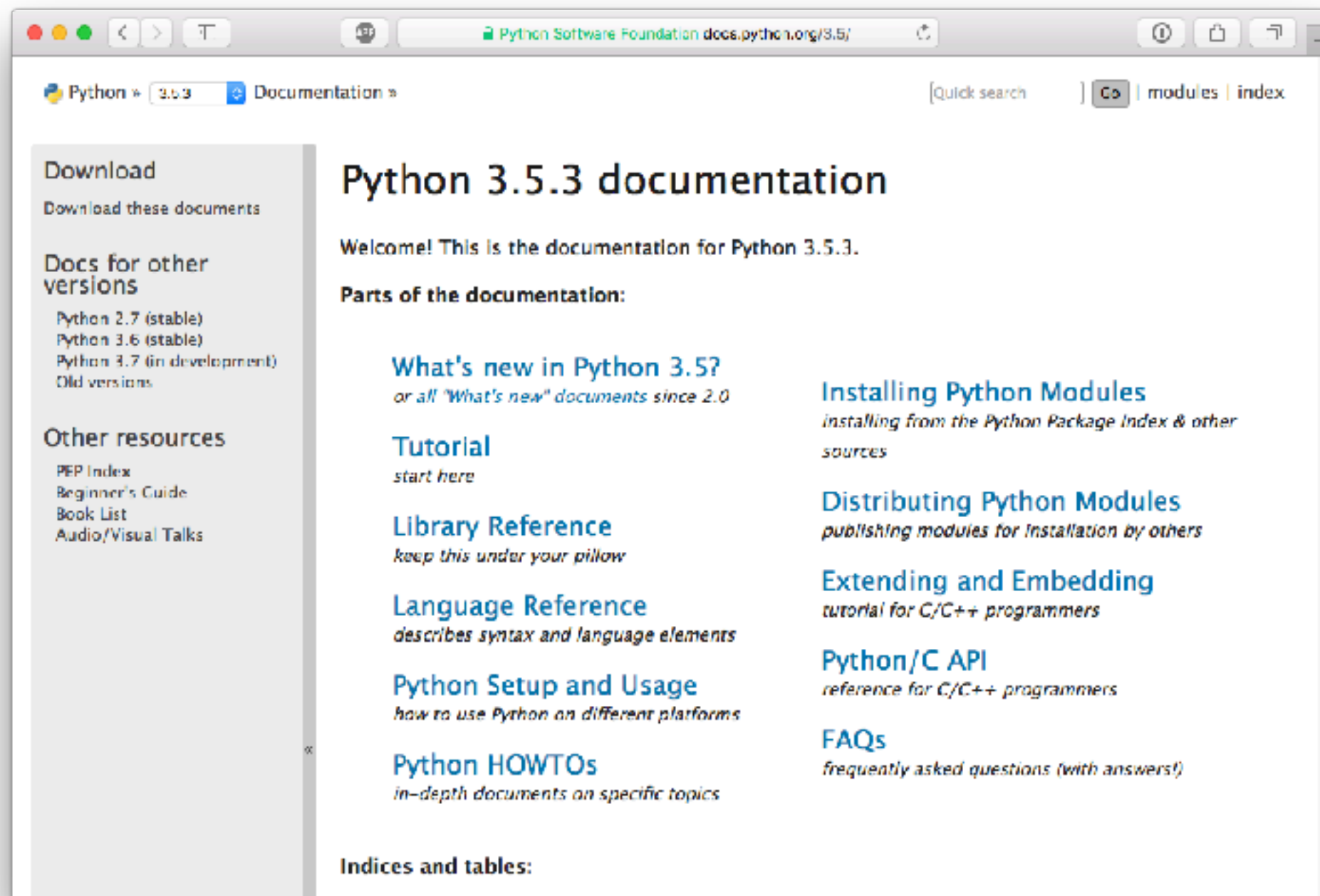
We are using Python 3.4 on EMR! (3 years old!)

I just got Python3.6 at work...

What's new in Python3.6?

What's new in Python3.5?

- Check the release notes!
- <https://docs.python.org/3.5/>



Key things new in Python 3.5

You should read "What's New In Python 3.5"

Here's what I think is neat:

- `bytes.hex()`
- `collections.OrderedDict()` — 4 to 100 times faster (implemented in C)
- `os.scandir()`
- `subprocess.run()`
- `@` infix operator for matrix multiplication (with numpy)
- typing module and Type hints

```
def greeting(name:str) -> str:  
    return 'Hello ' + name
```

Key things in Python 3.6

New modules:

- secrets — holds cryptographic secrets

Faster modules:

- dict — takes 20% to 25% less memory compared to Python 3.5

Better modules

- datetime — support for Local Time Disambiguation (for daylight savings time)
- typing — improvements
- hash lib — support for BLAKE2, SHA-3, and SHAKE

Language improvements

```
>>> name = "Fred"
>>> f"He said his name is {name}."
'He said his name is Fred.'
```

```
>>> 1_000_000
1000000
```

Lots more cool stuff

When should you use new language features?

Use a new feature if:

- It simplifies your program
- You can specify which version of Python your users will use.
- The version of Python is supported on *every computer you need to use*.

Don't use a new feature if:

- You want your code to run on as many systems as possible.
- Your code needs to run on legacy or un-patched systems.

Use Python 2 if:

- You need to use a module that isn't supported on Python 3
- Only support Python 2.7
- After reading <https://wiki.python.org/moin/Python2orPython3>

Be aware of the Python HOWTOs

The screenshot shows a web browser window displaying the Python Software Foundation documentation page for Python HOWTOs. The browser's address bar shows the URL `docs.python.org/3.5/howto/index`. The page header includes the Python logo, version `3.5.3`, and the word `Documentation`. A search bar and navigation links (`Go`, `previous`, `next`, `modules`, `index`) are also present.

The main content area is titled **Python HOWTOs**. It explains that Python HOWTOs are documents covering specific topics and are modeled on the Linux Documentation Project's HOWTO collection. It states that the collection aims to provide more detailed documentation than the Python Library Reference.

Under the heading "Currently, the HOWTOs are:", a list of 15 topics is provided:

- [Porting Python 2 Code to Python 3](#)
- [Porting Extension Modules to Python 3](#)
- [Curses Programming with Python](#)
- [Descriptor HowTo Guide](#)
- [Functional Programming HOWTO](#)
- [Logging HOWTO](#)
- [Logging Cookbook](#)
- [Regular Expression HOWTO](#)
- [Socket Programming HOWTO](#)
- [Sorting HOW TO](#)
- [Unicode HOWTO](#)
- [HOWTO Fetch Internet Resources Using The urllib Package](#)
- [Argparse Tutorial](#)
- [An introduction to the ipaddress module](#)
- [Argument Clinic How-To](#)

The left sidebar contains navigation links: "Previous topic" (Installing Python Modules), "Next topic" (Porting Python 2 Code to Python 3), and "This Page" (Report a Bug, Show Source).

The footer of the page includes copyright information: "© Copyright 2001–2017, Python Software Foundation. The Python Software Foundation is a non-profit corporation. [Please donate.](#) Last updated on Feb 20, 2017. [Found a bug?](#) Created using [Sphinx 1.3.3](#)."



Introducing Spark

Spark — Not another layer on MapReduce

What's wrong with MapReduce?

- Slow! All data written to disk at each stage.
- Awkward — Lots of calculations can't be easily described as a "map" and a "reduce"
- Wasteful — Many programs (machine learning) require multiple passes over same data.

UC Berkeley by the AMP Lab — Developed Spark

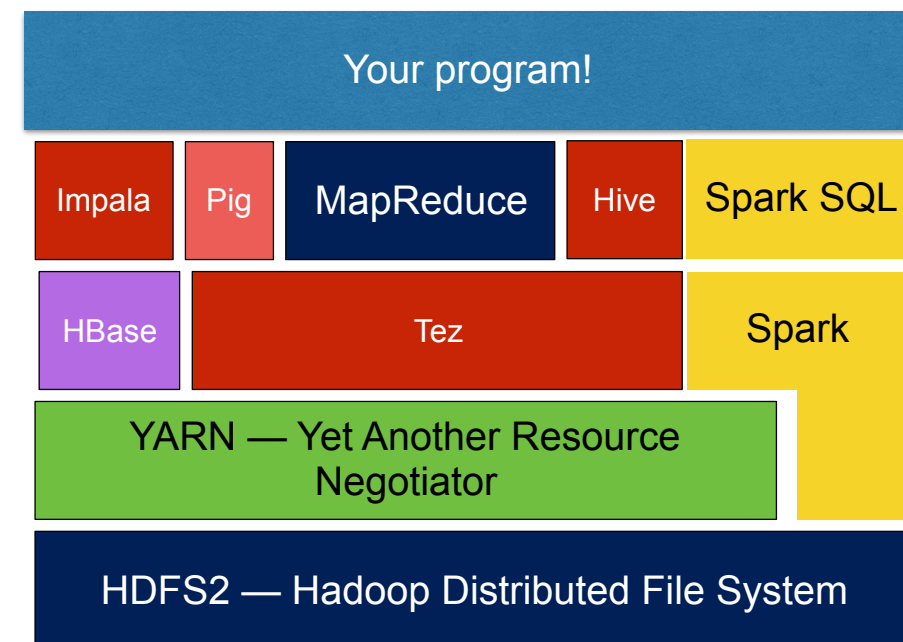
- <http://spark-project.org/>

Not another layer on MapReduce

- Run on top of YARN
- Run directly on the cluster

Compatible with Hadoop:

- Read/write any Hadoop data source
 - HDFS (Text Files, Sequence Files, etc)
 - S3 (on EMR)
 - HBase, Hive, etc.
- Run legacy MapReduce jobs



Spark — improves on core Hadoop ideas

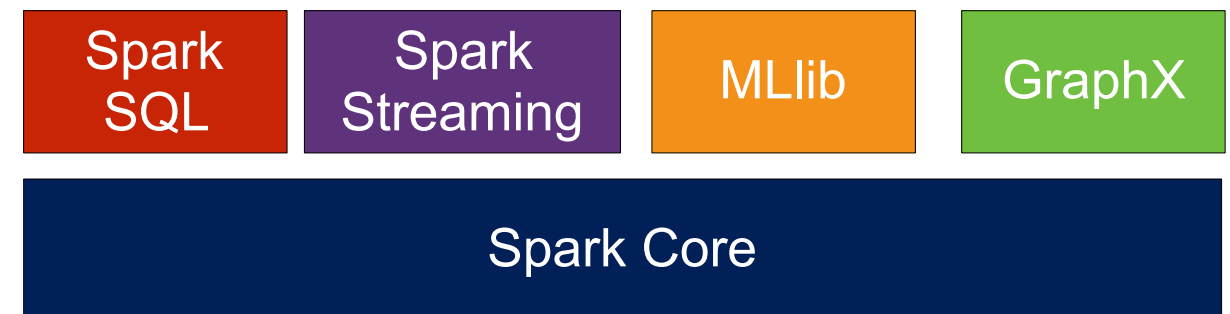
Hadoop MapReduce:

- Designed for data flow, not for iterative calculations — e.g. Machine Learning
- Designed for batch processing — high overhead, slow.
- Java/JVM is "first class" citizen
 - *other language require "streaming" interface*

Spark:

- Designed for data sharing between steps — Iterative processing
- Support interactive development.
- Supports multiple languages (Scala, Python, R)
- Flexible, expressive programming model.

Key Spark Parts:



Four ways to run:

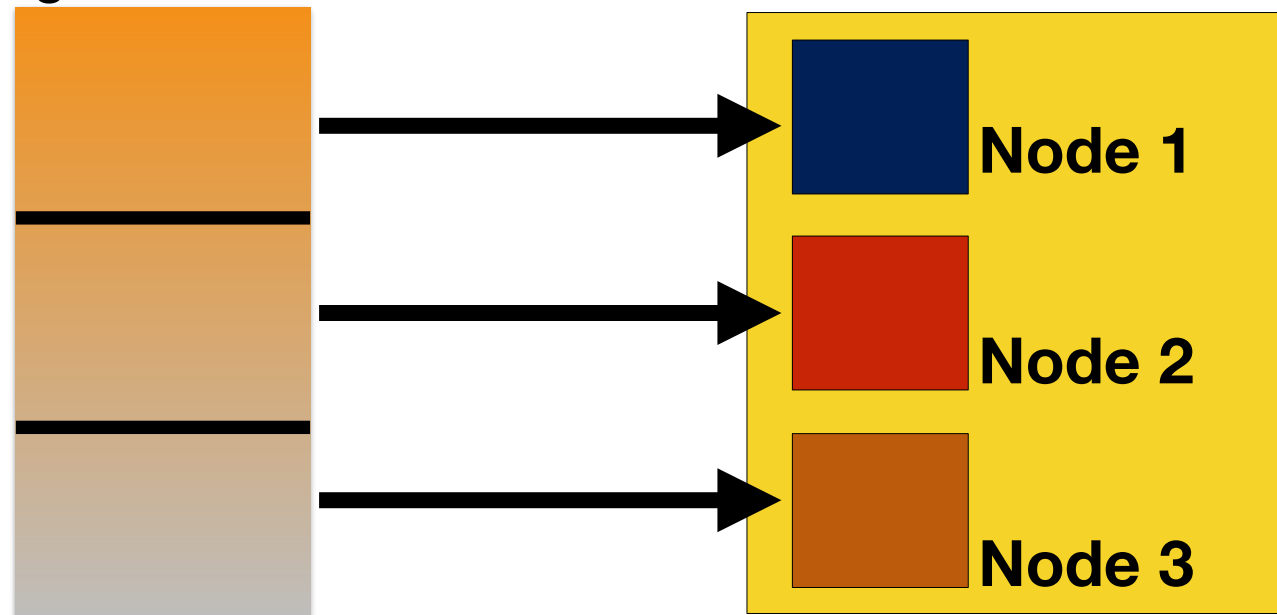
- Local Mode
- EC2 — multiple systems, native OS
- Apache Mesos (cluster management software)*
- Apache YARN — Run alongside MapReduce

*[https://aws.amazon.com/blogs/compute/cluster-management-with-](https://aws.amazon.com/blogs/compute/cluster-management-with-ec2/)

Spark — A fundamentally different computation model

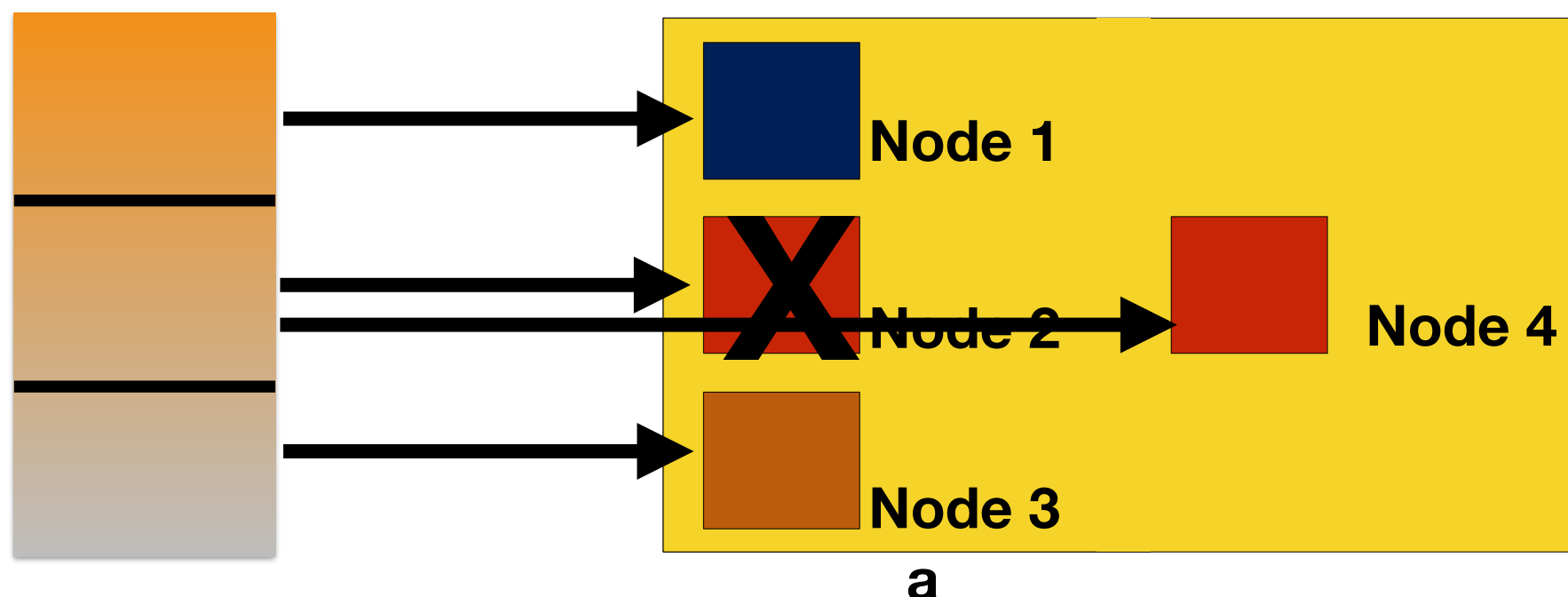
Resilient Dynamic Datasets (RDDs)

- RDD stores a single "dataset" as a set of buffers on different computers:

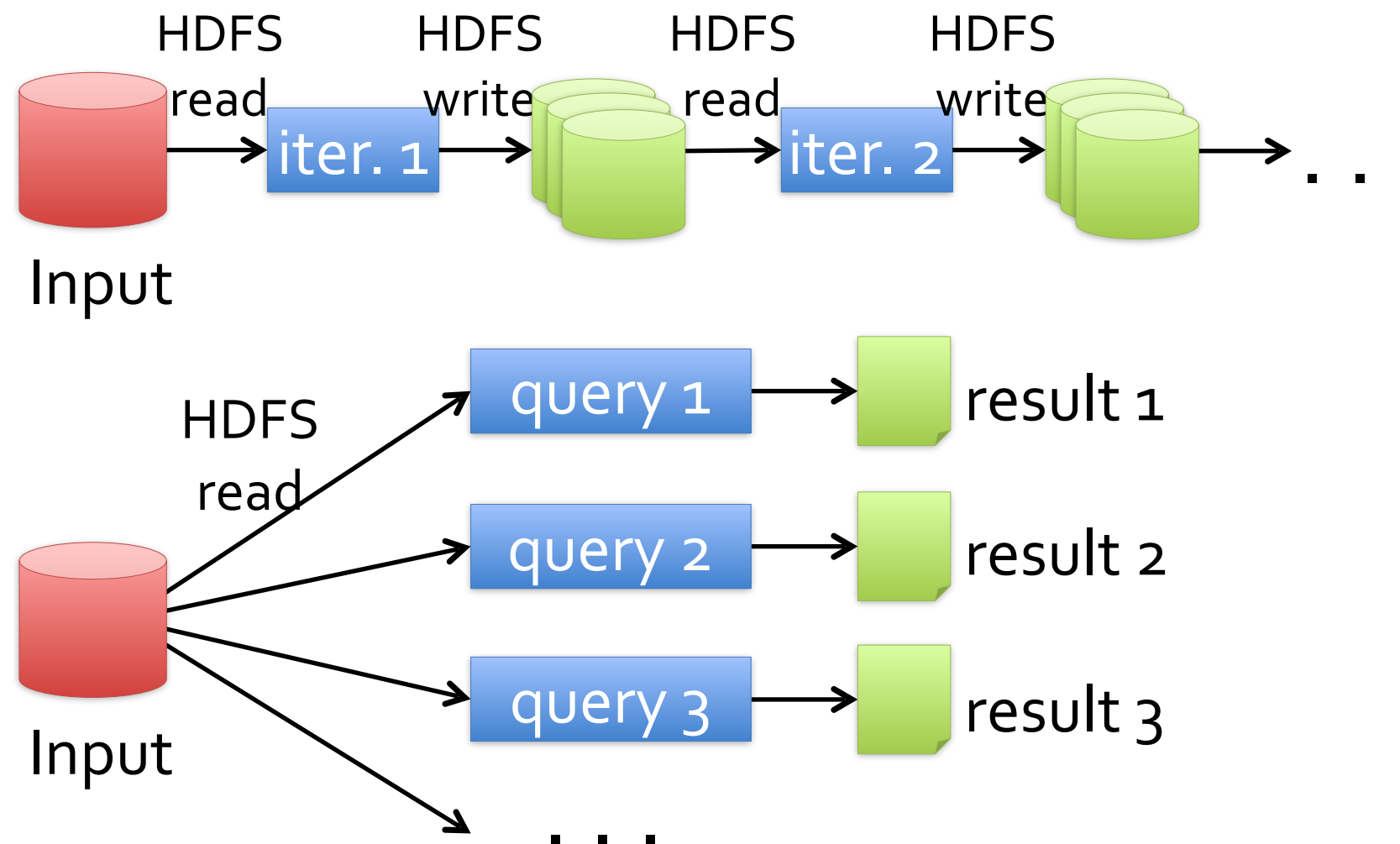


```
a = sc.textFile("s3://gu-anly502/logs/forensicswiki.2012.txt")
```

If a node fails, Spark re-generates the data as necessary.

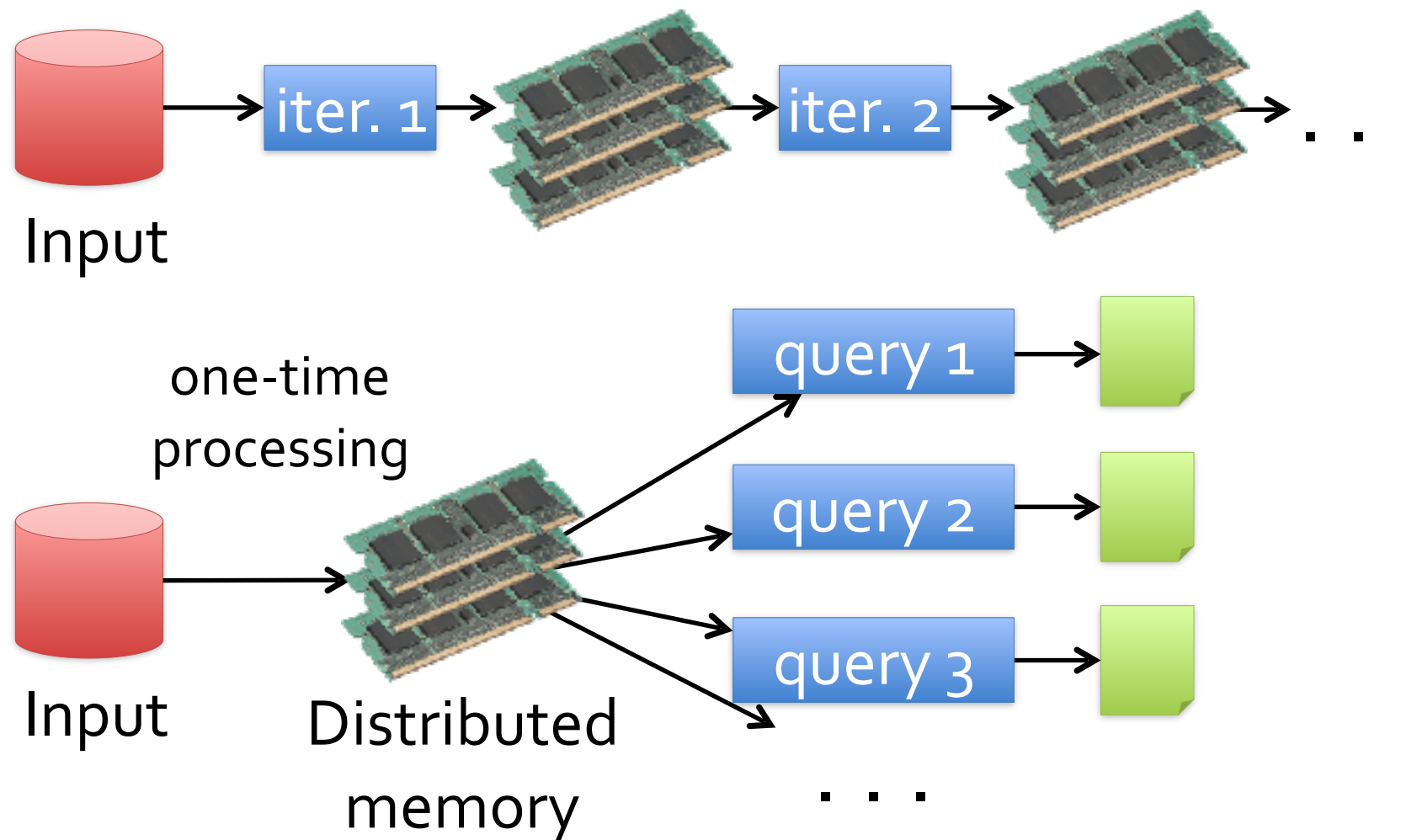


Data Sharing in MapReduce



Slow due to replication, serialization, and disk IO

Data Sharing in Spark



10-100× faster than network and disk

Resilient Distributed Datasets (RDDs)

Hold structured data.

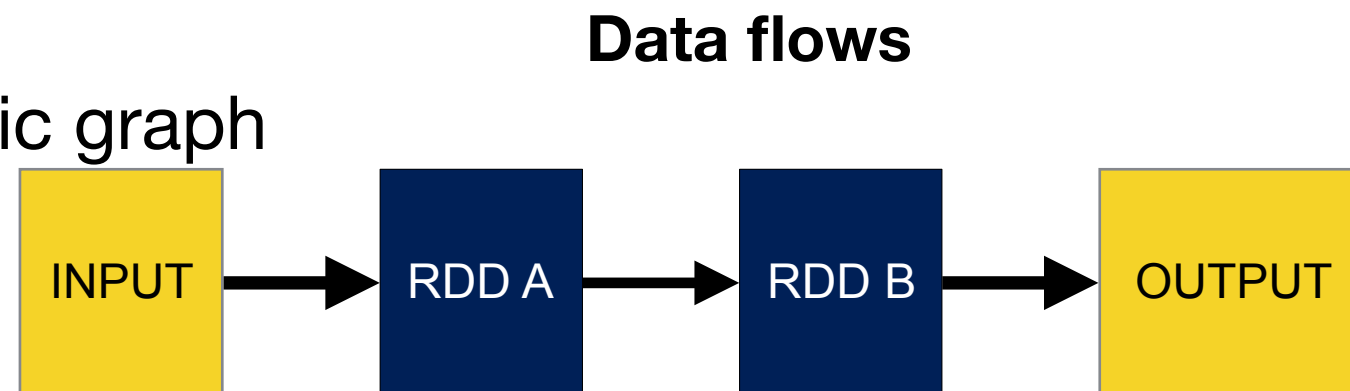
Distributed across 1 or more nodes.

Connected together in a directed acyclic graph

- Graph is created by the Spark program

Automatically rebuilt on failure

Immutable / Read-Only (do not change)



RDDs can hold:

- "Rows" — Untyped rows of data
- Dataframes — Typed records

Spark requires:

- **Cluster manager**
- **Distributed storage system**

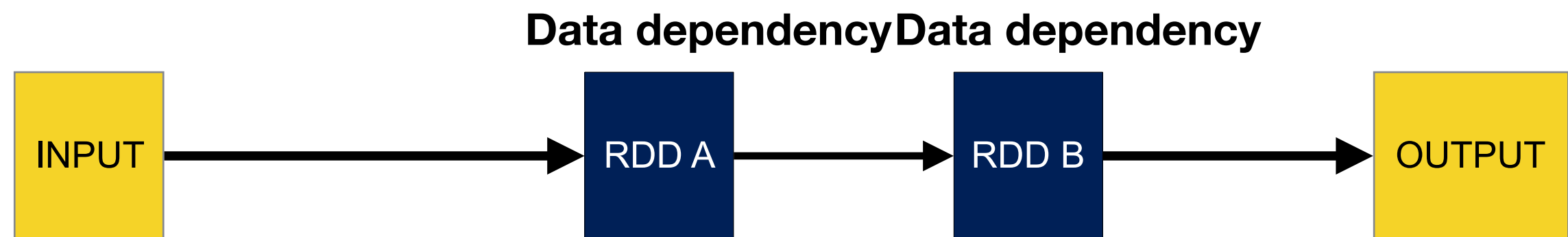
Spark also offers:

- Broadcast variables — copied to every node (read-only)
- Accumulators — Can only be "added" to (similar to Hadoop Counters)

You write code in Python that performs a computation.

Sample Question:

- How many lines in Shakespeare contain the word "Hamlet" ?



<s3://gu-anly502/ps04/Shakespeare.txt>

**1. Read the
file from S3
line-by-line**

**2. Filter the
lines for
"Hamlet"**

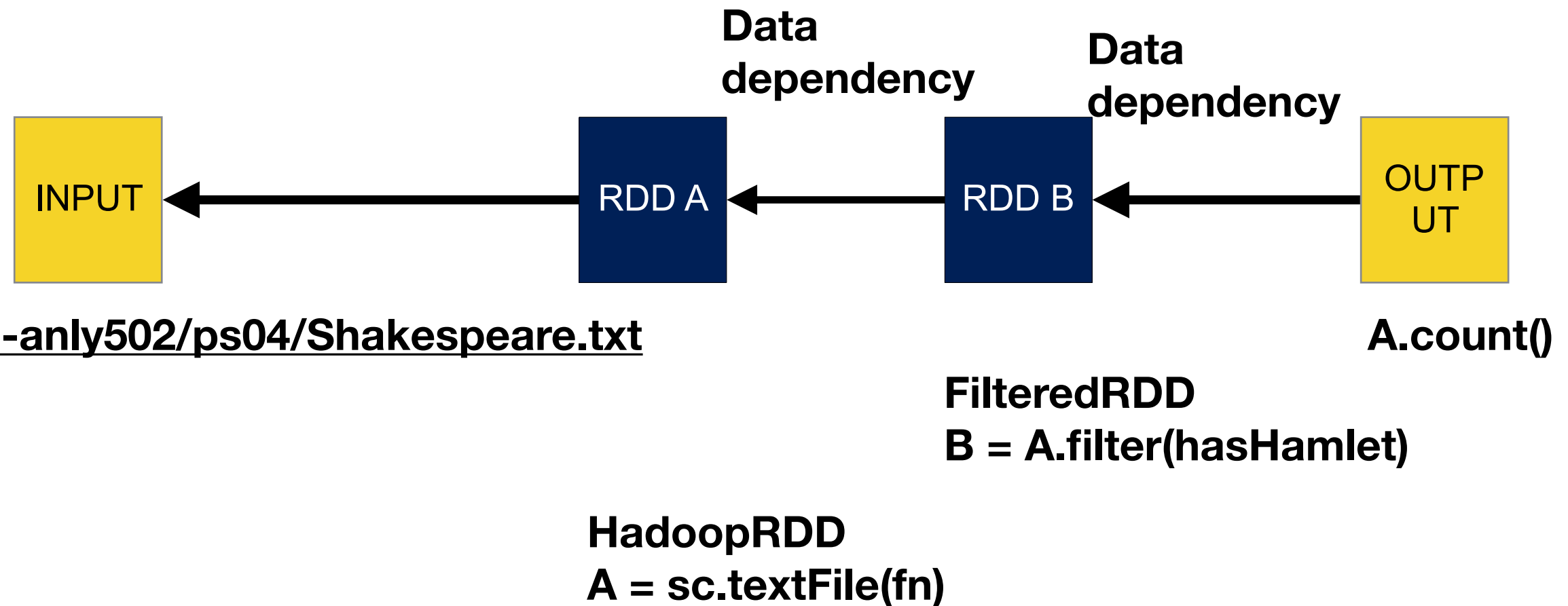
**3. Print Line
Count**

```
A = sc.textFile("s3://gu-anly502/ps04/Shakespeare.txt")

def hasHamlet( s ):
    return "Hamlet" in s

B = A.filter( hasHamlet )
print( B.count() )
```

Behind the scenes, Spark builds a dependency tree



When the output is requested, the RDDs are created as necessary.

1. Start Spark

```
$ ipyspark
Python 3.4.3 (default, Sep 1 2016, 23:33:38)
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?                -> Introduction and overview of IPython's features.
%quickref        -> Quick reference.
help             -> Python's own help system.
object?         -> Details about 'object', use 'object??' for extra details.
Ivy Default Cache set to: /home/hadoop/.ivy2/cache
The jars for the packages stored in: /home/hadoop/.ivy2/jars
:: loading settings :: url = jar:file:/usr/lib/spark/jars/ivy-2.4.0.jar!/org/apache/ivy/
core/settings/ivysettings.xml
com.databricks#spark-csv_2.11 added as a dependency
com.databricks#spark-avro_2.11 added as a dependency
org.elasticsearch#elasticsearch-spark_2.11 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent;1.0
  confs: [default]
  found com.databricks#spark-csv_2.11;1.5.0 in central
  found org.apache.commons#commons-csv;1.1 in central
  found com.univocity#univocity-parsers;1.5.1 in central
  found com.databricks#spark-avro_2.11;3.0.0 in central
  found org.slf4j#slf4j-api;1.7.5 in central
  found org.apache.avro#avro;1.7.6 in central
  found org.codehaus.jackson#jackson-core-asl;1.9.13 in central
  found org.codehaus.jackson#jackson-mapper-asl;1.9.13 in central
  found com.thoughtworks.paranamer#paranamer;2.3 in central
  found org.xerial.snappy#snappy-java;1.0.5 in central
  found org.apache.commons#commons-compress;1.4.1 in central
  found org.tukaani#xz;1.0 in central
  found org.elasticsearch#elasticsearch-spark_2.11;2.4.0 in central
:: resolution report :: resolve 601ms :: artifacts dl 19ms
...
```

```
17/02/26 15:43:58 WARN Client: Same path resource file:/home/hadoop/.ivy2/jars/
org.apache.commons_commons-compress-1.4.1.jar added multiple times to distributed cache.
17/02/26 15:43:58 WARN Client: Same path resource file:/home/hadoop/.ivy2/jars/
org.tukaani_xz-1.0.jar added multiple times to distributed cache.
Welcome to
```

[illegible]

```
Using Python version 3.4.3 (default, Sep 1 2016 23:33:38)
SparkSession available as 'spark'.
```

```
In [1]: A = sc.textFile("s3://gu-anly502/ps04/Shakespeare.txt")
```

```
In [2]: def hasHamlet( s ):
...:     return "Hamlet" in s
...:
```

```
In [3]: B = A.filter( hasHamlet )
```

```
In [4]: print( B.count() )
        ... magic ...
```

108

```
In [5]: print( B.count() )
```

With Spark, you can look at your data interactively!

```
In [6]: A.first()
```

```
Out[6]: '<<THIS ELECTRONIC VERSION OF THE COMPLETE WORKS OF
```

`.first()` takes first line

```
In [7]: B.take(5)
```

```
Out[7]:
```

```
[' Hamlet, son to the former, and nephew to the present king.',  
' Horatio, friend to Hamlet.',  
' Getrude, Queen of Denmark, mother to Hamlet.',  
" Ghost of Hamlet's Father.",  
" Dar'd to the combat; in which our valiant Hamlet"]
```

`.take(n)` takes first n

```
In [8]: B.takeSample(False, 5)
```

```
Out[8]:
```

```
[' Mar. Lord Hamlet!',  
' Oph. So please you, something touching the Lord Hamlet',  
' [Laertes wounds Hamlet; then] in scuffle with Laertes',  
" [The King puts Laertes' hand into Hamlet's bosom].",  
' King. From Hamlet? Who brought them?']
```

`.takeSample(False,3)` takes 3 random lines

```
In [9]: B.collect()
```

```
Out[9]:
```

```
[' Hamlet, son to the former, and nephew to the present king.',  
' Horatio, friend to Hamlet.',  
' Getrude, Queen of Denmark, mother to Hamlet.',  
" Ghost of Hamlet's Father.",  
" Dar'd to the combat; in which our valiant Hamlet",  
' His fell to Hamlet. Now, sir, young Fortinbras,',  
' Unto young Hamlet; for, upon my life,',  
' Flourish. [Enter Claudius, King of Denmark]',  
" King. Though yet of Hamlet our dear brother's death",  
' But now, my cousin Hamlet, and my son—',  
...]
```

`.collect()` returns *all the lines*

```
In [1]: A = sc.textFile("s3://gu-anly502/ps04/Shakespeare.txt")  
In [2]: def hasHamlet( s ):  
...:     return "Hamlet" in s  
...:  
In [3]: B = A.filter( hasHamlet )
```


Use completion to see all of the methods...

In [10]: A.

A.aggregate A.aggregateByKey A.cache A.cartesian A.checkpoint	A.coalesce A.cogroup A.collect A.collectAsMap A.combineByKey	A.context A.count A.countApprox A.countApproxDistinct A.countByKey	A.countByValue A.ctx A.distinct A.filter A.first
A.flatMap A.flatMapValues < A.fold A.foldByKey A.foreach	A.foreachPartition A.fullOuterJoin A.getCheckpointFile A.getNumPartitions A.getStorageLevel	A.glom A.groupBy A.groupByKey A.groupWith A.histogram	A.id A.intersection A.is_cached A.is_checkpointed A.isCheckpointed
A.isEmpty A.isLocallyCheckpointed < A.join A.keyBy A.keys	A.leftOuterJoin A.localCheckpoint A.lookup A.map A.mapPartitions	A.mapPartitionsWithIndex A.mapPartitionsWithSplit A.mapValues A.max A.mean	A.meanApprox A.min A.name A.partitionBy A.partitioner
A.persist A.pipe < A.randomSplit A.reduce A.reduceByKey	A.reduceByKeyLocally A.repartition A.repartitionAndSortWithinPartitions A.rightOuterJoin A.sample	A.sampleByKey A.sampleStdev A.sampleVariance A.saveAsHadoopDataset A.saveAsHadoopFile	A.saveAsNewAPIHadoopDataset A.saveAsNewAPIHadoopFile A.saveAsPickleFile A.saveAsSequenceFile A.saveAsTextFile
A.saveAsNewAPIHadoopDataset A.saveAsNewAPIHadoopFile < A.saveAsPickleFile A.saveAsSequenceFile A.saveAsTextFile	A.setName A.sortBy A.sortByKey A.stats A.stdev	A.subtract A.subtractByKey A.sum A.sumApprox A.take	A.takeOrdered A.takeSample A.toDebugString A.toDF A.toLocalIterator
A.saveAsNewAPIHadoopDataset A.saveAsNewAPIHadoopFile < A.saveAsPickleFile A.saveAsSequenceFile A.saveAsTextFile	A.setName A.sortBy A.sortByKey A.stats A.stdev	A.subtract A.subtractByKey A.sum A.sumApprox A.take	A.takeOrdered A.takeSample A.toDebugString A.toDF A.toLocalIterator
A.subtract A.subtractByKey < A.sum A.sumApprox A.take	A.takeOrdered A.takeSample A.toDebugString A.toDF A.toLocalIterator	A.top A.treeAggregate A.treeReduce A.union A.unpersist	A.values A.variance A.zip A.zipWithIndex A.zipWithUniqueId

sample() — performs sampling into an RDD

takeSample() — Combines sampling and .collect()

sample(self, withReplacement, fraction, seed=None) method of pyspark.rdd.PipelinedRDD instance

Return a sampled subset of this RDD.

takeSample(self, withReplacement, num, seed=None) method of pyspark.rdd.PipelinedRDD instance

Return a fixed-size sampled subset of this RDD.

```
In [10]: B.sample(False,.10)
Out[10]: PythonRDD[7] at RDD at PythonRDD.sc
```

.sample() returns an RDD

```
In [11]: B.sample(False,.10).collect()
...
```

```
Out[11]:
```

```
[u'      Unto young Hamlet; for, upon my life,',
u'      Queen. Good Hamlet, cast thy nighted colour off,',
u'      Oph. So please you, something touching the Lord Hamlet.
u'      Than a command to parley. For Lord Hamlet,',
u"      Lord Hamlet, with his doublet all unbrac'd,",
u"      Of Hamlet's transformation. So I call it,",
u'      [Exit the Queen. Then] Exit Hamlet, tugging in',
u'      Enter Hamlet and Guildenstern [with Attendants].',
u'      King. Hamlet, this deed, for thine especial safety,-',
u'      The present death of Hamlet. Do it, England;
u'      [Exeunt all but Hamlet.]',
u"      And that in Hamlet's hearing, for a quality",
u'      King. Stay, give me drink. Hamlet, this pearl is thine;']
```

.collect() transfers the data to the driver

```
In [12]:
```

Execution time: 2 seconds

Same example, bigger dataset:

Question: How many times does Main_Page appear in the forensicswiki logs?

```
In [13]: s = sc.textFile("s3://gu-anly502/logs/forensicswiki.2012.txt")
```

```
In [14]: smain = s.filter( lambda line:"Main_Page" in line )
```

```
In [15]: %time smain.count()
```

```
[Stage 9:>
```

```
[Stage 9:=====>
```

```
[Stage 9:=====>
```

```
[Stage 9:=====>
```

```
[Stage 9:=====>
```

(0 + 4) / 64]

(20 + 4) / 64]

(28 + 4) / 64]

(35 + 4) / 64]

(53 + 4) / 64]

lambda notation!

```
348820
```

```
CPU times: user 32 ms, sys: 4 ms, total: 36 ms
```

```
Wall time: 1min 15s
```

75 sec

```
In [16]:
```

Spark does not cache results by default

If we type `main.count()` again, it has to recompute:

```
In [16]: smain.count()
[Stage 10:=====> (32 + 4) / 64]
[Stage 10:=====> (36 + 4) / 64]
[Stage 10:=====> (51 + 4) / 64]
[Stage 10:=====> (56 + 4) / 64]
[Stage 10:=====> (62 + 2) / 64]
Out[16]: 348820
```

To cache `smain`, we need to explicitly tell it to cache:

`cache()` method of `pyspark.rdd.PipelinedRDD` instance
Persist this RDD with the default storage level (`C{MEMORY_ONLY}`).

```
In [18]: smain.cache()
Out[18]: PythonRDD[16] at RDD at PythonRDD.scala:48
```

```
In [19]: smain.count()
[Stage 11:> (0 + 0) / 64]
[Stage 11:=====> (8 + 4) / 64]
...
Out[19]: 348820
```

```
In [20]: smain.count()
Out[20]: 348820
```

```
In [21]: smain.count()
Out[21]: 348820
```

With Spark, it's easy to check your work.

How do we trust this number:

```
In [16]: smain.count()
```

```
Out[16]: 348820
```

```
In [17]:
```

Try looking at some of the output:

```
In [17]: smain.sample(False,.001).collect()
```

```
Out[17]:
```

```
[u'193.105.210.94 - - [01/Jan/2012:08:35:04 -0800] "GET //Talk:Main_Page HTTP/1.0" 404 518 "http://  
www.forensicswiki.org//Talk:Main_Page" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; APC; .NET  
CLR 1.0.3705; .NET CLR 1.1.4322; .NET CLR 2.0.50215; InfoPath.1)" ',  
u'32.178.74.29 - - [02/Jan/2012:12:18:16 -0800] "HEAD /index.php?title=Main_Page HTTP/1.1" 200 321  
"- "Mozilla/4.0 (compatible; Powermarks/3.5; Windows 95/98/2000/NT)" ',  
u'91.117.143.86 - - [02/Jan/2012:15:57:09 -0800] "GET /w/extensions/BibTex/bibtex.js HTTP/1.1" 200  
1384 "http://www.forensicswiki.org/wiki/Main_Page" "Mozilla/5.0 (X11; U; i686 Linux; es, gl, en_GB,  
en_US) AppleWebKit/533.3 (KHTML, like Gecko) Chrome/5.0.358.0 Safari/533.3" ',  
u'61.68.18.158 - - [03/Jan/2012:14:44:07 -0800] "GET /w/skins/monobook/main.css?270 HTTP/1.1" 304 174  
"http://www.forensicswiki.org/wiki/Main_Page" "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:8.0.1) Gecko/  
20100101 Firefox/8.0.1" ',  
u'198.234.82.254 - - [04/Jan/2012:06:48:42 -0800] "GET /wiki/Tools HTTP/1.1" 200 13302 "http://  
www.forensicswiki.org/wiki/Main_Page" "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0;  
.NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30; .NET CLR 3.0.4506.2152; .NET CLR  
3.5.30729; MDDR; InfoPath.2)" ',  
...
```

Whoops! — We got the answer to our question, but it was the wrong question...

More about RDDs

A RDD may be on multiple nodes.

The control program is called the "Driver."



```
A = sc.textFile("s3://gu-anly502/ps04/Shakespeare.txt")
```

```
print( B.count() )
```

```
def hasHamlet( s ):  
    return "Hamlet" in s
```

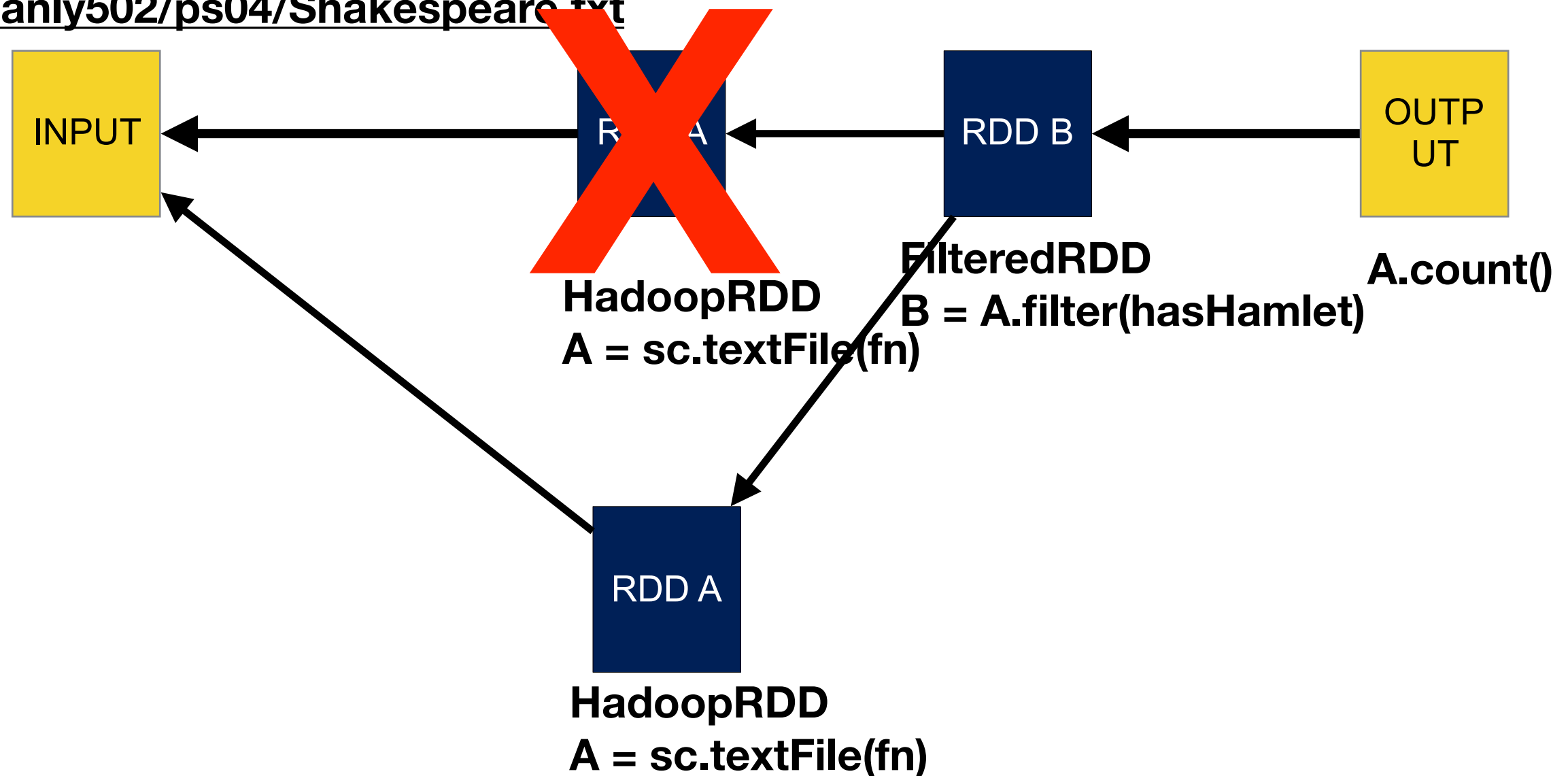
```
B = A.filter( hasHamlet )
```

HadoopRDD
A = sc.textFile(fn)

RDDs are deterministic.

If an RDD fails, it is rebuilt automatically

s3://gu-only502/ps04/Shakespeare.txt



Example: Log Mining

NOTE: SCALA!

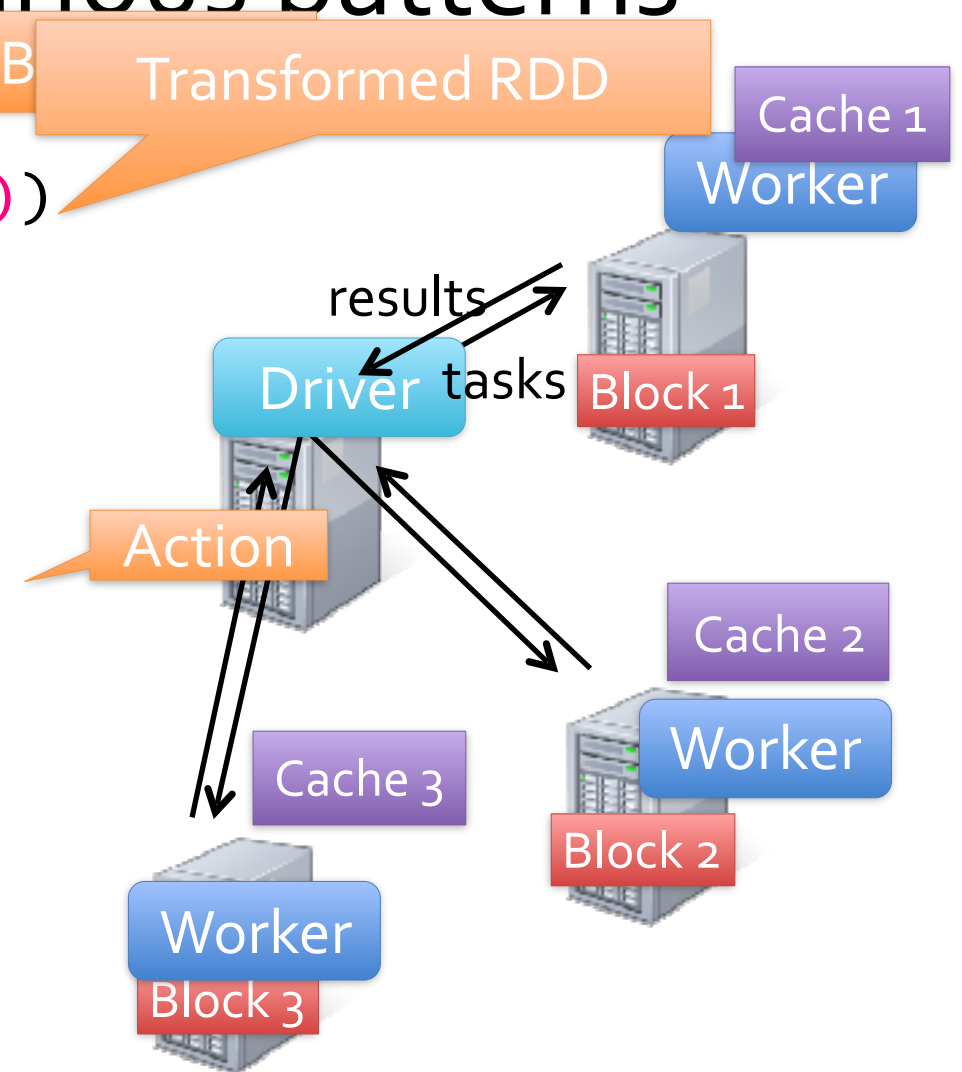
Load error messages from a log into memory,
then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
. . .
```

Result: full-text search of Wikipedia
in <1 sec (vs 20 sec for on-disk data)

Result: scaled to 1 TB data in 5-7 sec
(vs 170 sec for on-disk data)



Example: Logistic Regression

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

```
var w = Vector.random(D)
```

Initial parameter vector

Load data in memory once

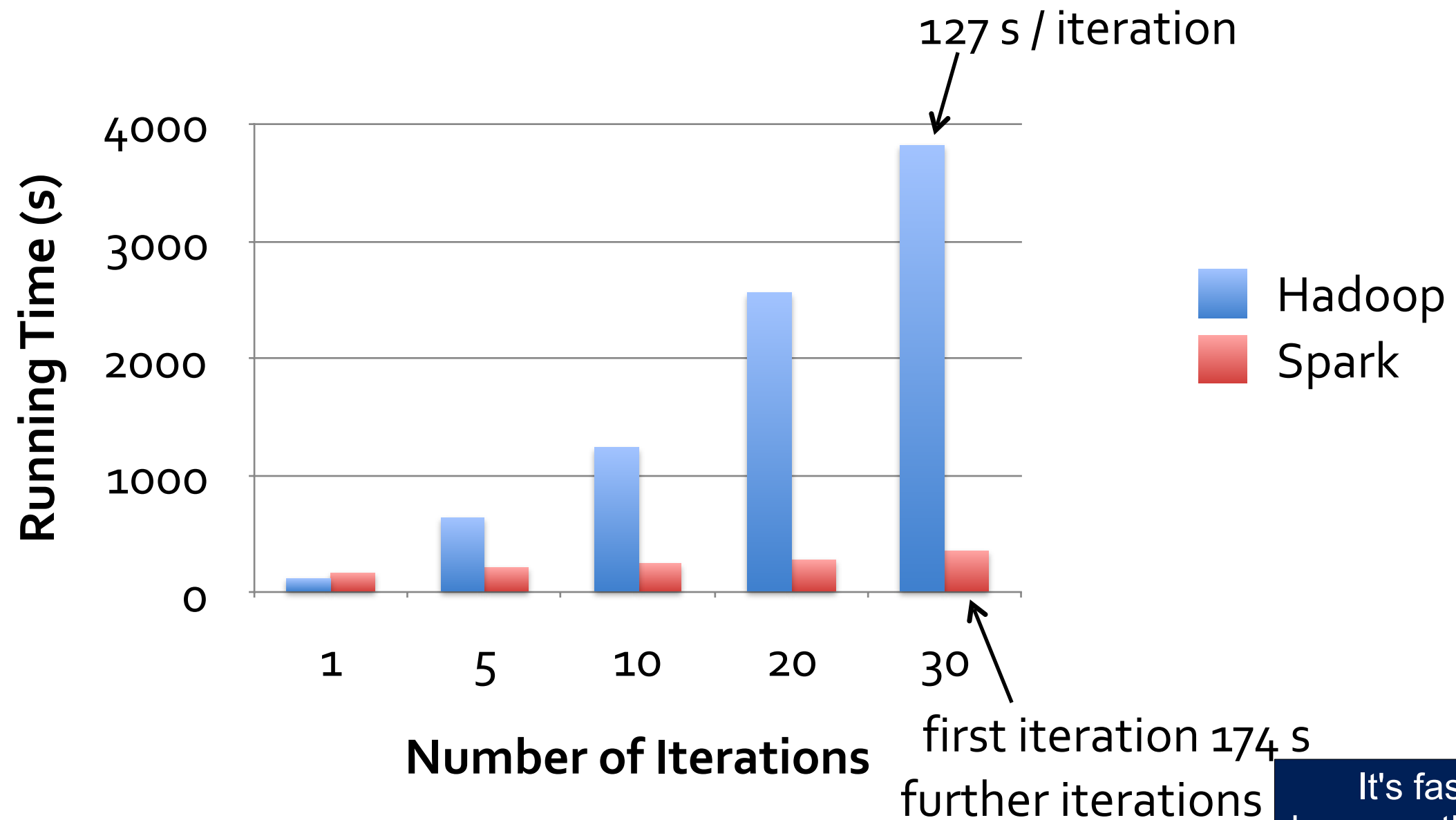
```
for (i <- 1 to ITERATIONS) {  
  val gradient = data.map(p =>  
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x  
  ).reduce(_ + _)  
  w -= gradient  
}
```

Repeated MapReduce steps
to do gradient descent

```
println("Final w: " + w)
```

Logistic Regression Performance

AMP Lab Slide



first iteration 174 s
further iterations

It's faster
because the data
are read into
memory ONCE.

Scala vs. Python

Spark was developed in Scala

Good news: Python and scala look a lot alike.

Expect to see code like this:

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()
```

Translate it like this:

```
lines = sc.textFile("hdfs://...")
errors = lines.filter( lambda a:a.startswith("ERROR"))
messages = errors.map( lambda a:a.split("\t", 1)
cachedMsgs = messages.cache()
```

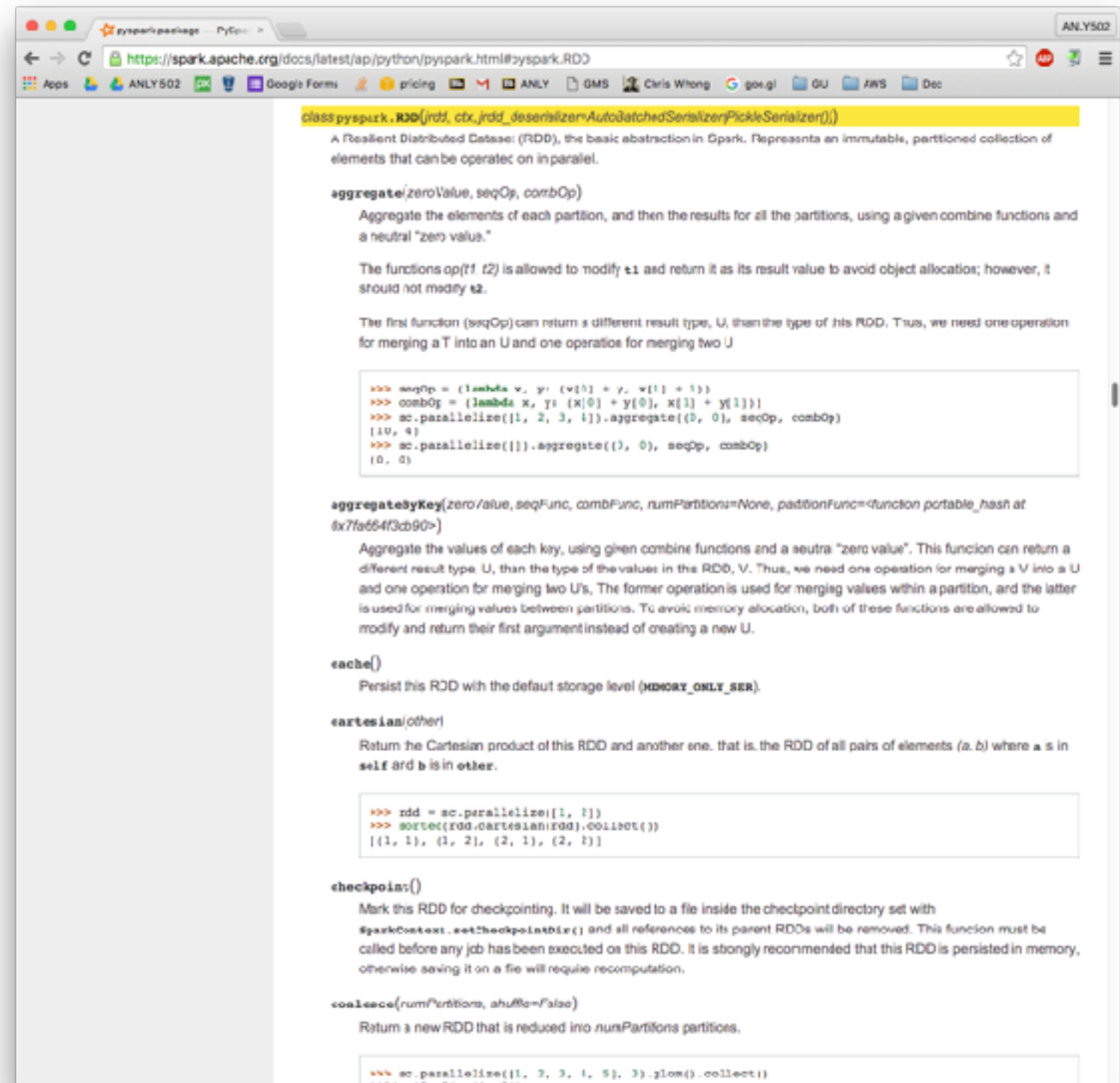
pyspark.RDD — basic class for RDD

<https://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD>

About 100 different methods

- map, reduce, reduceByKey
- filter
- count
- cogroup, groupBy
- partitionBy
- join, leftOuterJoin, rightOuterJoin, cross
- sample
- save
- pipe

More...



Lab #1

Start up EMR and Spark and Log in

Run pyspark (or ipyspark)

Work the Shakespeare example:

```
A = sc.textFile("s3://gu-anly502/ps04/Shakespeare.txt")
def hasHamlet( s ):
    return "Hamlet" in s
B = A.filter( hasHamlet )
```

or

```
B = A.filter( lambda s: "Hamlet" in s)
```

```
print( B.count() )
B.cache()
B.takeSample(False, 10)
B.take(10)
B.first()
B.collect()
```

Redo problem sets with Spark!

A1 - Find the malformed entries?

```
A = sc.textFile("s3://gu-anly502/A1/quazyilx1.txt")
B = A.filter(lambda bad:"fnard:-1 fnok:-1 cark:-1 gnuck:-1" in bad)
B.cache()
B.count()
```

A2 - Forensicswiki hit analysis

```
'[28/Dec/2012:06:43:35 -0800] "GET /w/load.php?
debug=false&lang=en&modules=jquery.checkboxShiftClick%2Ccookie%2CmakeCollapsible%2CmessageBox%2Cmw
Prototypes%2Cplaceholder%7Cmediawiki.language%2Cuser%2Cutil%7Cmediawiki.legacy.ajax%2Cwikibits%7Cm
ediawiki.page.ready&skin=monobook&version=20120730T153329Z&* HTTP/1.1" 200 11293 "http://
www.forensicswiki.org/wiki/Main_Page" "Mozilla/5.0 (X11; Linux i686) AppleWebKit/536.11 (KHTML,
like Gecko) Ubuntu/12.04 Chromium/20.0.1132.47 Chrome/20.0.1132.47 Safari/536.11"'
```

```
date_re = re.compile("\[(\d\d/[a-zA-Z]+\d\d\d\d)\]")
def extract(line):
    m = date_re.search(line)
    if m:
        d = datetime.datetime.strptime(m.group(1), "%d/%b/%Y")
        return "{:04}-{:02}".format(d.year, d.month)
W = sc.textFile("s3://gu-anly502/logs/forensicswiki.2012.txt")
W.cache()
dates = W.map( lambda line: [ extract( line ), 1 ])
dates.cache()
dates.countByKey()
```

7:45 Start

Spark history and community

2009 — Developed at UC Berkeley AMPLab

2010 — First open source release

2012 — Spark 0.5

2013 — Donated to Apache Foundation

— Databricks founded by Ali Ghodsi, Andy Konwinski, Ion Stoica, Patrick Wendell, Reynold Xin, Matei Zaharia

2014 — Spark becomes a "Top-Level Apache Project"

— **Spark 1.0**

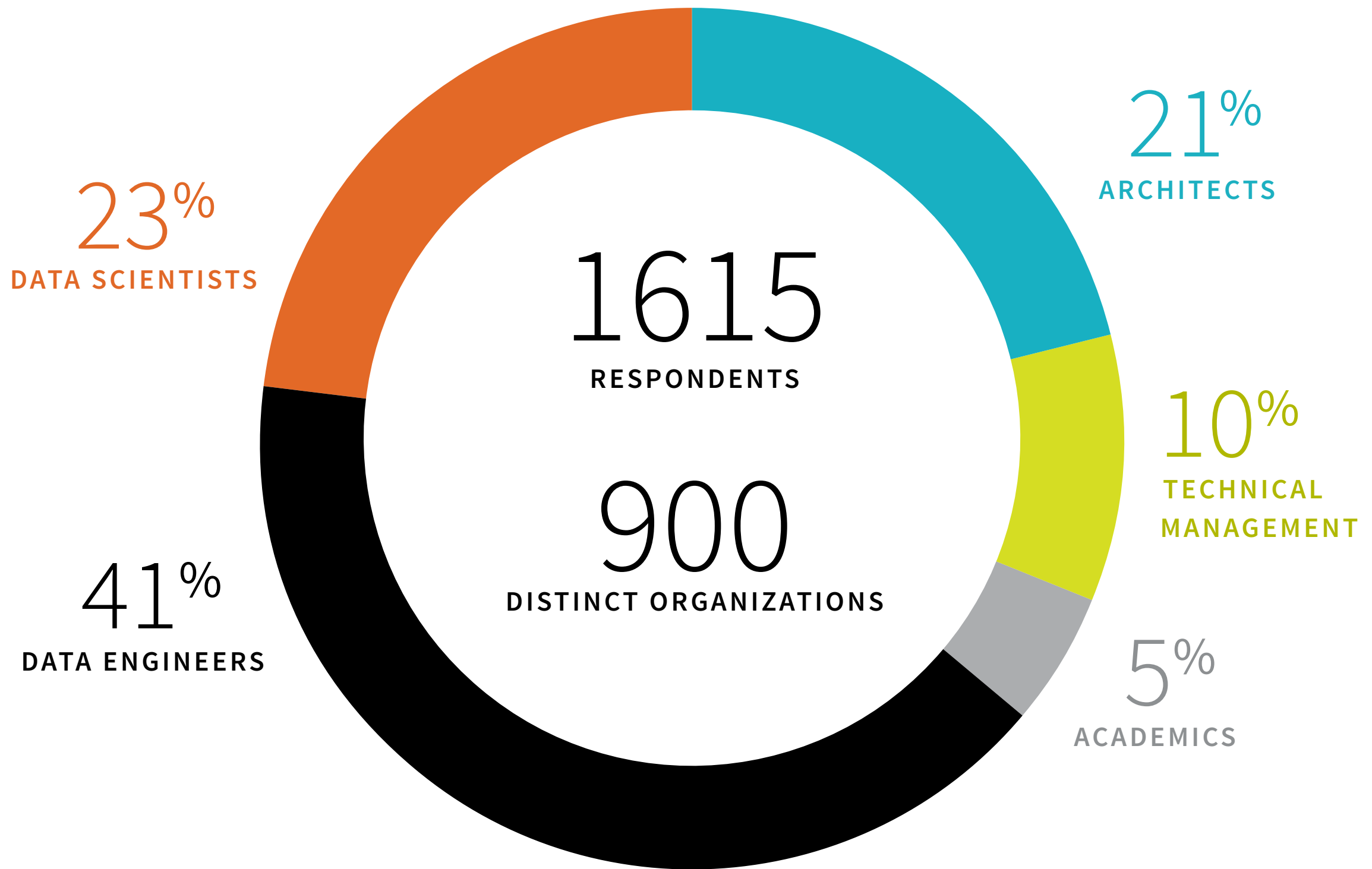
2015 — Spark has more than 1000 contributors.

2016 — Spark has more than 2000 contributors

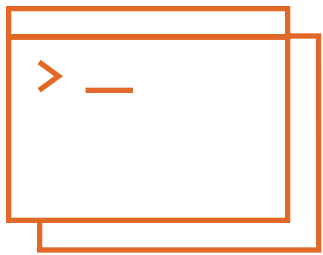
— **Spark 2.0**



Who is using Spark?



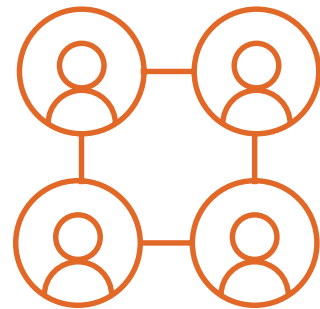
Spark is the most active open source Big Data project.



CODE CONTRIBUTORS

+67%

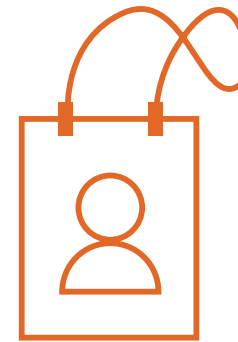
2015	2016
600	1000



SPARK MEETUP MEMBERS

+240%

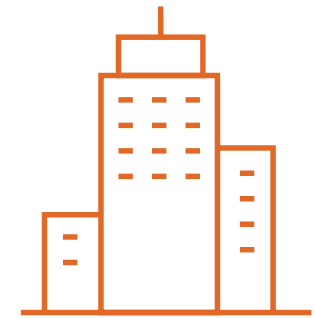
2015	2016
66,000	225,000



SPARK SUMMIT ATTENDEES

+30%

2015	2016
3912	5100



COMPANIES REPRESENTED AT SUMMITS

+57%

2015	2016
1144	1800

Spark components

Spark SQL

Spark
Streaming

MLib
machine
learning

GraphX
graph
processing

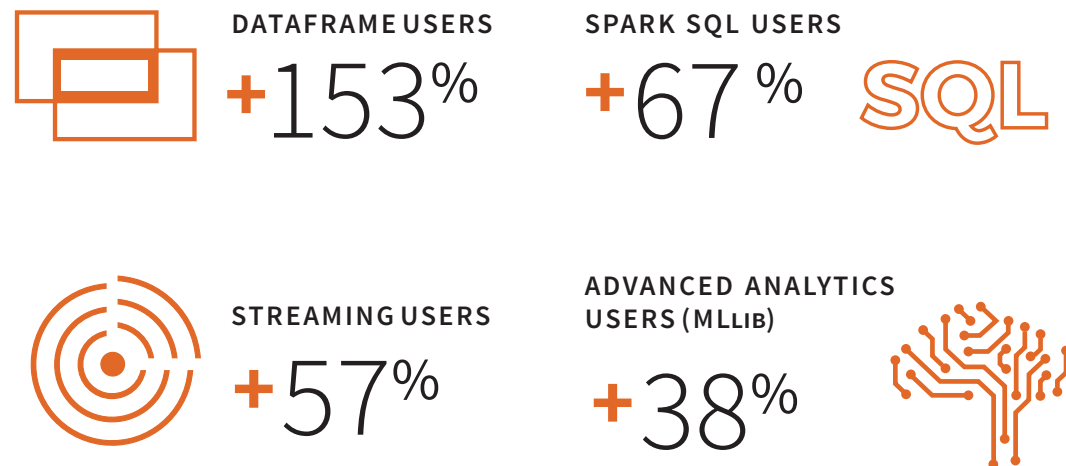
Spark Core

Standalone Scheduler

YARN

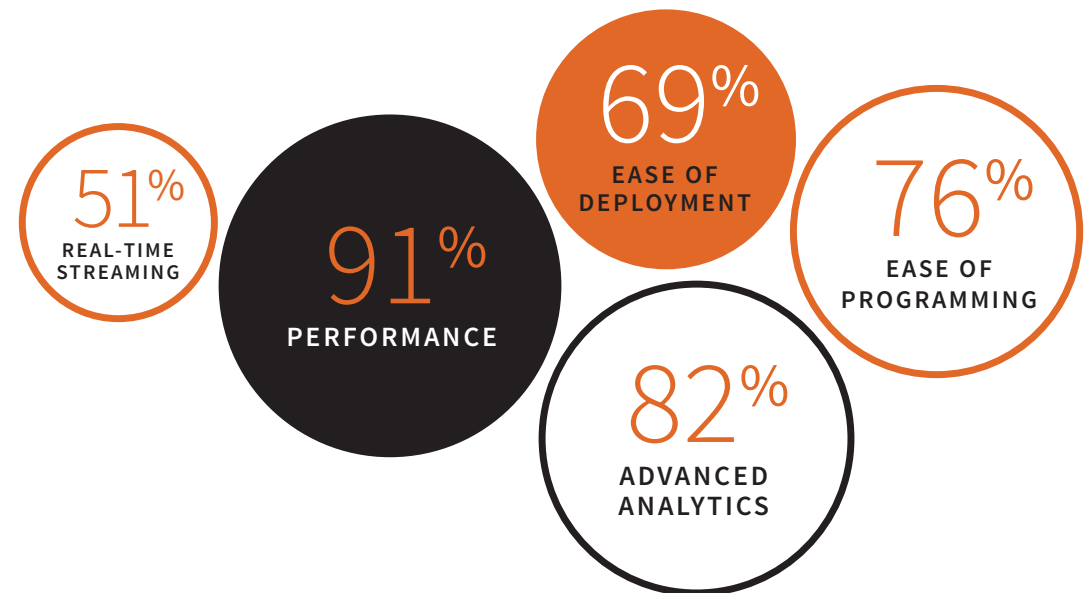
Mesos

APACHE SPARK'S FASTEST GROWING AREAS IN 2016



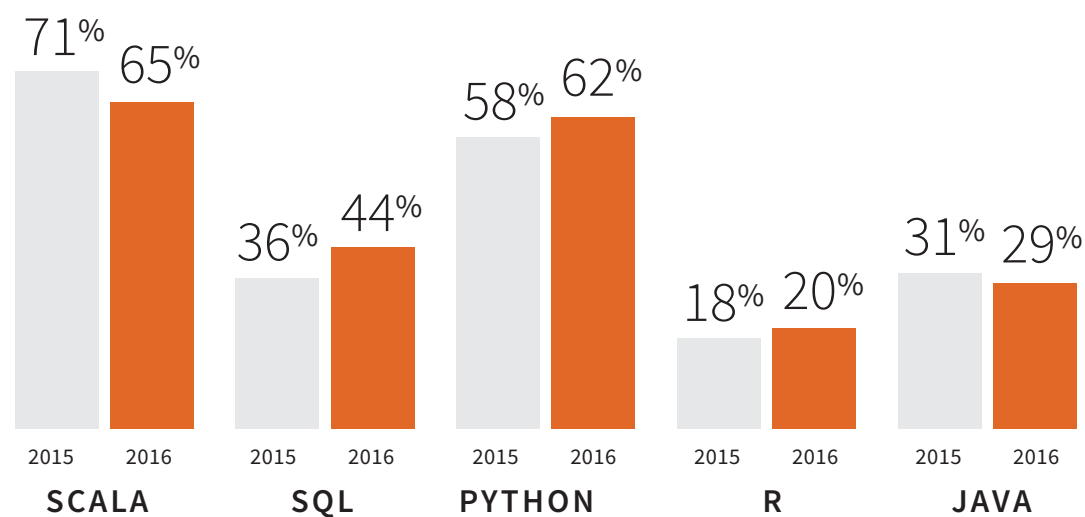
FEATURES USERS CONSIDER IMPORTANT

Respondents were allowed to select more than one feature.



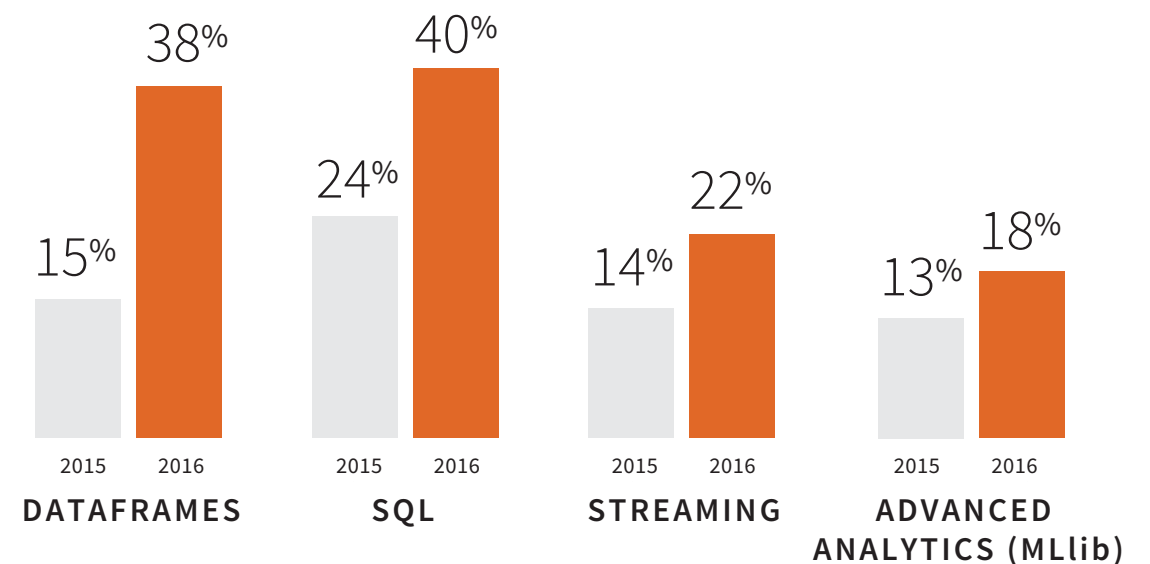
LANGUAGES USED IN APACHE SPARK

Respondents were allowed to select more than one language.



SPARK COMPONENTS USED IN PRODUCTION

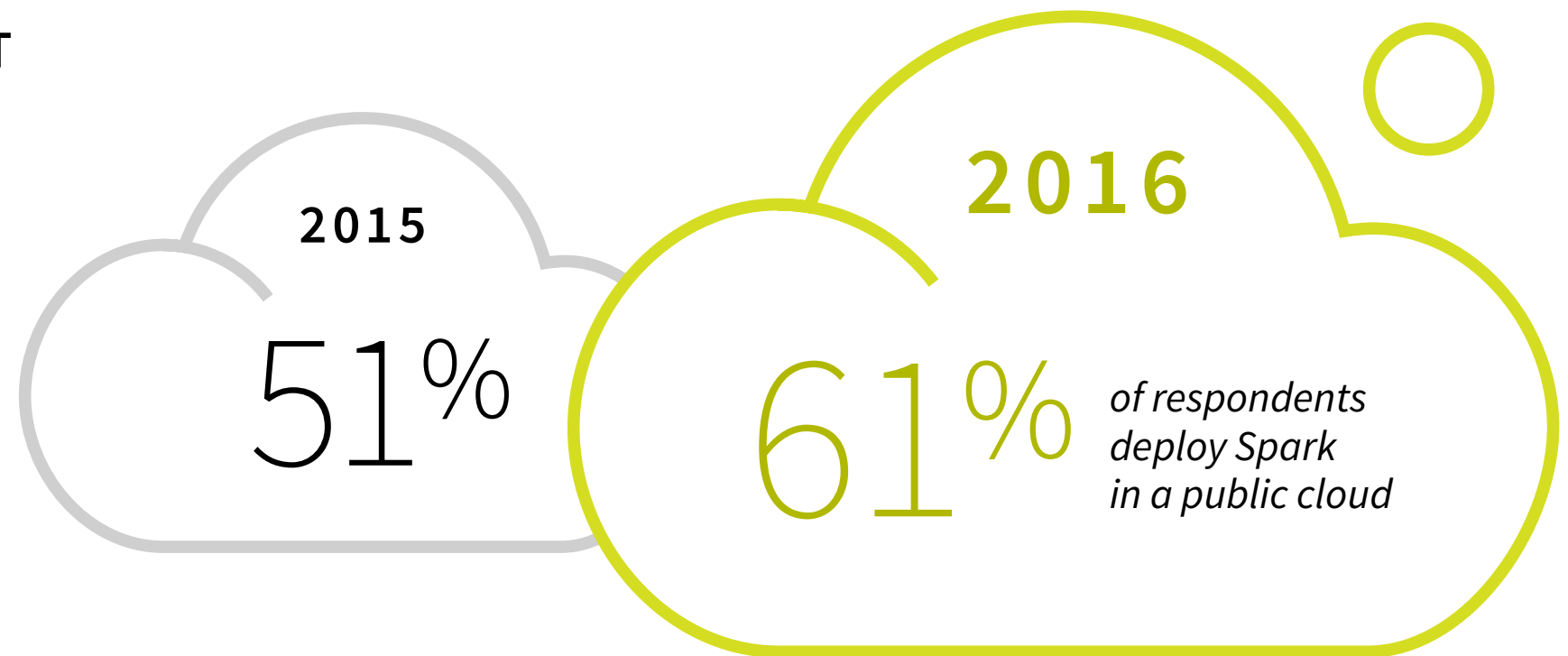
Respondents were allowed to select more than one component.



Most respondents deploy Spark in the cloud!

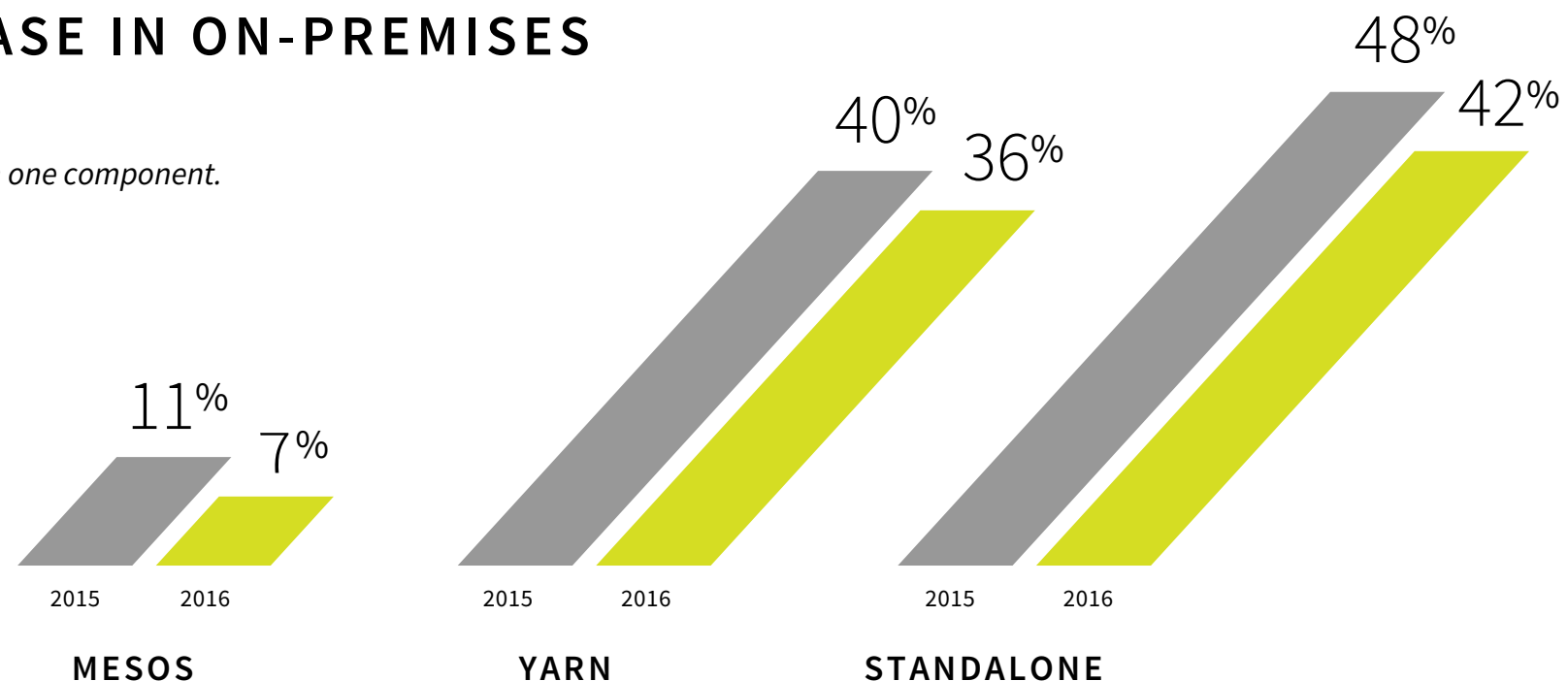
Apache Spark deployments in the public cloud increased in 2016. In contrast, the percentage of Spark deployments on-premises decreased in the past year.

**APACHE SPARK DEPLOYMENT
IN PUBLIC CLOUDS
HAS INCREASED BY 10%
SINCE 2015.**



PERCENTAGE DECREASE IN ON-PREMISES DEPLOYMENTS

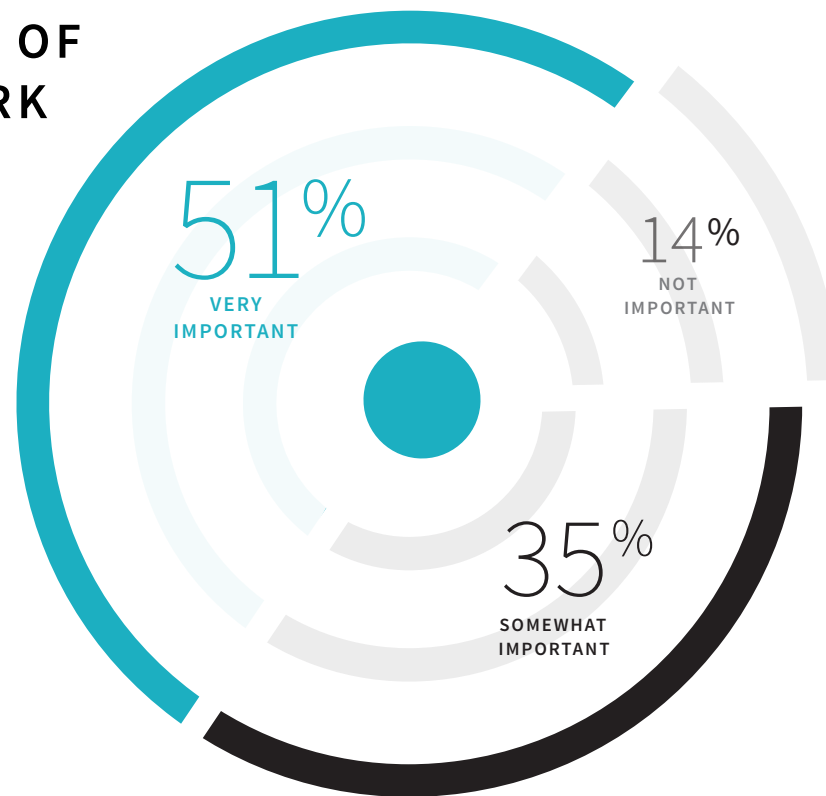
Respondents were allowed to select more than one component.



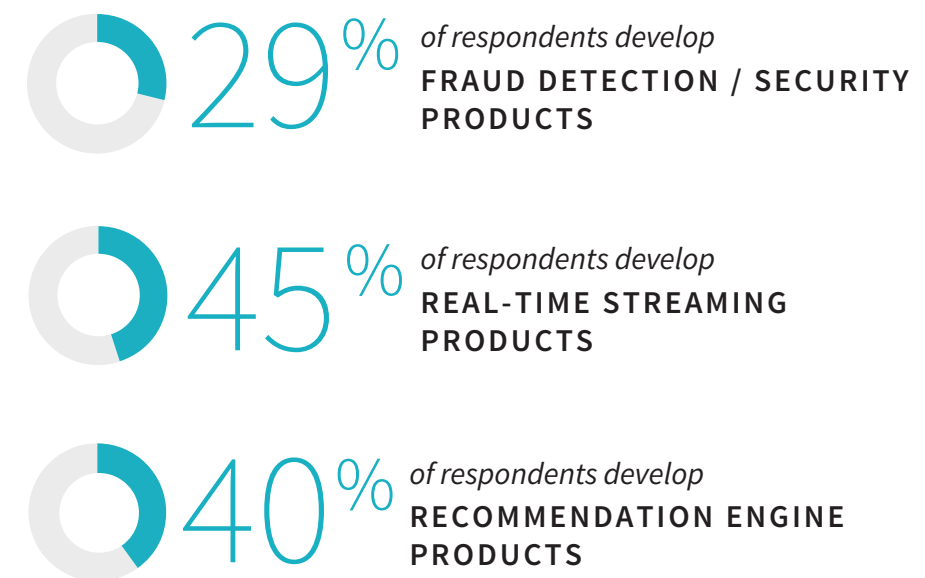
Streaming and Machine Learning are very important

Apache Spark Streaming is growing. Since its release, **Spark Streaming has become one of the most widely used distributed streaming engines**. Interest in developing real-time applications and advanced analytics is on the rise.

IMPORTANCE OF APACHE SPARK STREAMING TO USERS



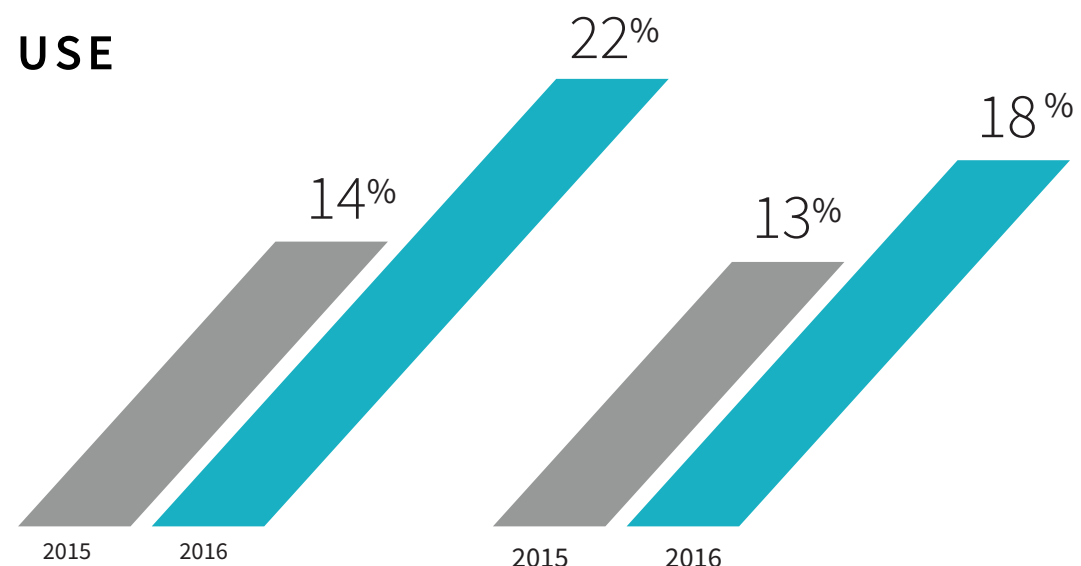
TYPES OF PRODUCTS DEVELOPED USING APACHE SPARK



APACHE SPARK STREAMING AND MLlib USE IN PRODUCTION

+57%
STREAMING
PRODUCTION CASES

+38%
ADVANCED ANALYTICS (MLlib)
PRODUCTION CASES



Apache Spark Philosophy

Streaming SQL ML Graph

① Unified engine
Support end-to-end applications

② High-level APIs
Easy to use, rich optimizations

③ Integrate broadly
Storage systems, libraries, etc



Apache
MESOS™



Mateiza Hariakey Keynote, 2016 Spark Summit, "Spark 2.0"

Spark 1 vs. Spark 2

Spark 1 — Speedup over MapReduce comes from RDDs

- Keep data in memory!
- Explicit "caching" — Programmer tells Spark when to cache.

Spark 2 — Speedup comes from *better optimizing*

- Strong typing of Dataframes — Avoid unnecessary storing/moving/computing
- "Structured APIs"
- Whole-stage code generation

"Type" — Python is not a strongly typed language

An array of integers in Python:

```
In [1]: A = [1,2,3,4]
```

```
In [2]: A[3] = "whoops"
```

```
In [3]: A
```

```
Out[3]: [1, 2, 3, 'whoops']
```

Each element is an "object"

Invalid assignments aren't caught until runtime.

Scala is a strongly typed language.

```
$ scala
Welcome to Scala 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_40).
Type in expressions for evaluation. Or try :help.
```

```
scala> val A = Array(1,2,3,4)
A: Array[Int] = Array(1, 2, 3, 4)
```

```
scala> A(2)
res0: Int = 3
```

```
scala> A(2) = 10
```

```
scala> A
res2: Array[Int] = Array(1, 2, 10, 4)
```

```
scala> A(2) = "Whoops"
<console>:13: error: type mismatch;
 found   : String("Whoops")
 required: Int
    A(2) = "Whoops"
           ^
```

```
scala>
```

Spark 2 tracks type and uses the most efficient representation possible. Dataframes can be 75% smaller



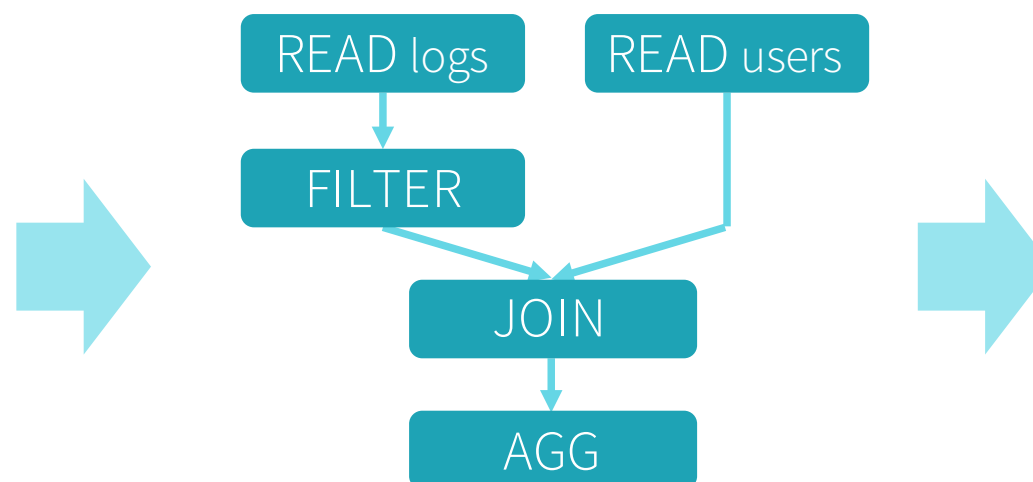
Scala runs on
top of the
JVM!

Spark 2 — Structured APIs

Spark 2 analyzes and refactors code and analyzes when it runs.

```
events =  
  sc.read.json("/logs")  
  
stats =  
  events.join(users)  
    .groupBy("loc", "status")  
    .avg("duration")  
  
errors = stats.where(  
  stats.status == "ERR")
```

DataFrame API



Optimized Plan

```
while(logs.hasNext) {  
  e = logs.next  
  if(e.status == "ERR") {  
    u = users.get(e.uid)  
    key = (u.loc, e.status)  
    sum(key) += e.duration  
    count(key) += 1  
  }  
}  
...
```

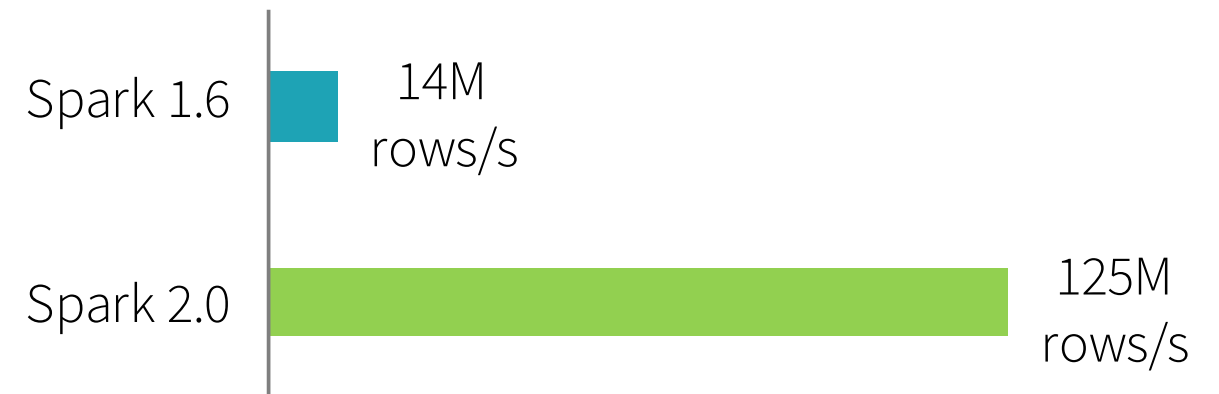
Specialized Code



Spark 2.0 - "Whole-stage code generation"

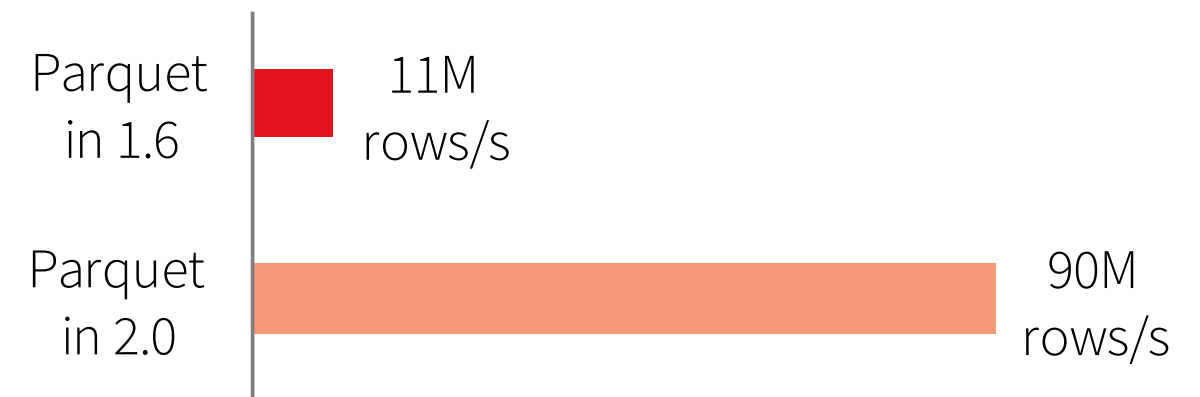
Whole-stage code generation

- Fuse across multiple operators



Optimized input / output

- Apache Parquet + built-in cache



Mateiza Hariakey Keynote, 2016 Spark Summit, "Spark 2.0"

Scala — Spark is written in Scala

Scala is a high-performance, compiled language.

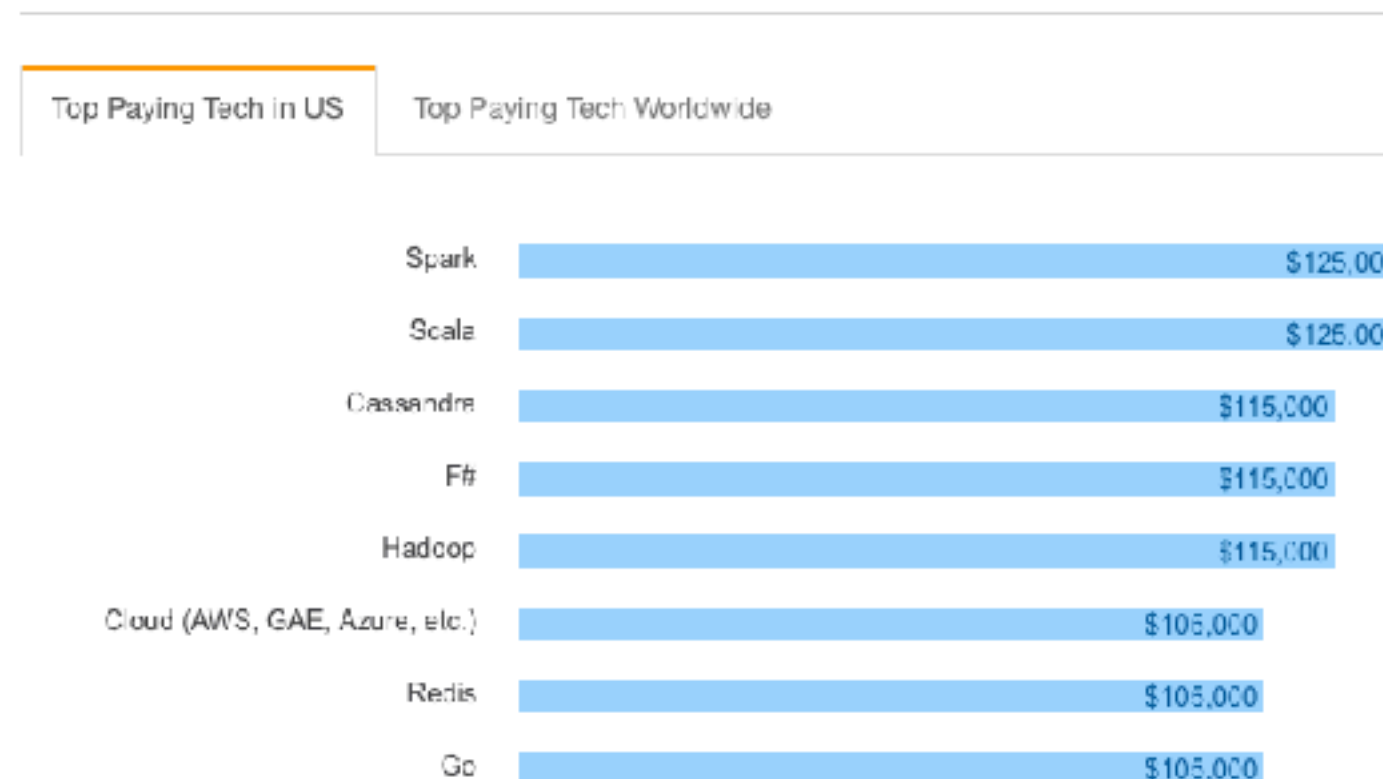
- Supports interactive code development with "read-eval-print" (REP) loop.
- Can freely call Java code.

Scala runs *much faster* and *more portable* than Python

- Scala is compiled and optimized.
- JVM has billions of dollars worth of performance tuning over 20+ years

Scala programmers are in higher demand

V. Top Paying Tech



<http://stackoverflow.com/research/developer-survey-2016>

Being more effective
with iPython

Recall the date example....

```
'[28/Dec/2012:06:43:35 -0800] "GET /w/load.php?
debug=false&lang=en&modules=jquery.checkboxShiftClick%2Ccookie%2CmakeCollapsible%2CmessageBox%2CmwPrototypes%2Cpl
aceholder%7Cmediawiki.language%2Cuser%2Cutil%7Cmediawiki.legacy.ajax%2Cwikibits%7Cmediawiki.page.ready&skin=monob
ook&version=20120730T153329Z&* HTTP/1.1" 200 11293 "http://www.forensicswiki.org/wiki/Main_Page" "Mozilla/5.0
(X11; Linux i686) AppleWebKit/536.11 (KHTML, like Gecko) Ubuntu/12.04 Chromium/20.0.1132.47 Chrome/20.0.1132.47
Safari/536.11"'
```

```
date_re = re.compile("\[(\d\d/[a-zA-Z]+\d\d\d\d)\]")
def extract(line):
    m = date_re.search(line)
    if m:
        d = datetime.datetime.strptime(m.group(1), "%d/%b/%Y")
        return "{:04}-{:02}".format(d.year, d.month)
W = sc.textFile("s3://gu-anly502/logs/forensicswiki.2012.txt")
dates = W.map( lambda line: [ extract( line ), 1 ])
dates.countByKey()

defaultdict(int,
    {'2012-01': 1544100,
     '2012-02': 1325030,
     '2012-03': 1274061,
     '2012-04': 1016456,
     '2012-05': 1173380,
     '2012-06': 1300250,
     '2012-07': 1287187,
     '2012-08': 1450426,
     '2012-09': 1284945,
     '2012-10': 1498895,
     '2012-11': 1397343,
     '2012-12': 1396198,
     '2013-01': 1283})
```


Here's what I actually did...

```
In [33]: W = sc.textFile("s3://gu-anly502/logs/forensicswiki.2012.txt")
```

```
In [34]: import re
```

```
In [35]: pat = re.compile("(\\d\\d)/([A-Za-z]+)/\\d\\d\\d\\d)")
```

```
In [36]: import datetime
```

```
In [37]: datetime.strptime("%d/%M/%y", "28/Dec/2012")
```

```
-----  
AttributeError                                Traceback (most recent call last)  
<ipython-input-37-0110e8ced25e> in <module>()  
----> 1 datetime.strptime("%d/%M/%y", "28/Dec/2012")
```

```
AttributeError: 'module' object has no attribute 'strptime'
```

```
In [38]: datetime.datetime.strptime("%d/%M/%y", "28/Dec/2012")
```

```
-----  
ValueError                                Traceback (most recent call last)  
<ipython-input-38-3156935281bf> in <module>()  
----> 1 datetime.datetime.strptime("%d/%M/%y", "28/Dec/2012")
```

```
/usr/lib64/python3.4/_strptime.py in _strptime_datetime(cls, data_string, format)  
    498     """Return a class cls instance based on the input string and the  
    499     format string."""  
--> 500     tt, fraction = _strptime(data_string, format)  
    501     tzname, gmtoff = tt[-2:]  
    502     args = tt[:6] + (fraction,)
```

```
/usr/lib64/python3.4/_strptime.py in _strptime(data_string, format)  
    335     if not found:  
    336         raise ValueError("time data %r does not match format %r" %  
--> 337             (data_string, format))  
    338     if len(data_string) != found.end():  
    339         raise ValueError("unconverted data remains: %s" %
```

```
ValueError: time data '%d/%M/%y' does not match format '28/Dec/2012'
```

```

In [40]: datetime.datetime.strptime("28/Dec/2012", "%d/%m/%y")
-----
ValueError                                Traceback (most recent call last)
<ipython-input-40-c0db5093c8da> in <module>()
----> 1 datetime.datetime.strptime("28/Dec/2012", "%d/%m/%y")

/usr/lib64/python3.4/_strptime.py in _strptime_datetime(cls, data_string, format)
    498     """Return a class cls instance based on the input string and the
    499     format string."""
--> 500     tt, fraction = _strptime(data_string, format)
    501     tzname, gmtoff = tt[-2:]
    502     args = tt[:6] + (fraction,)

/usr/lib64/python3.4/_strptime.py in _strptime(data_string, format)
    335     if not found:
    336         raise ValueError("time data %r does not match format %r" %
--> 337                        (data_string, format))
    338     if len(data_string) != found.end():
    339         raise ValueError("unconverted data remains: %s" %

ValueError: time data '28/Dec/2012' does not match format '%d/%m/%y'

```

Hm... Perhaps I should check the documentation?

```
$ man strptime
```

```
$ man strftime
```

```
STRFTIME(3)
```

```
BSD Library Functions Manual
```

```
STRFTIME(3)
```

```
NAME
```

```
    strftime, strftime_l -- format date and time
```

```
...
```

```
The conversion specifications are copied to the buffer after expansion as follows:—
```

```
%A    is replaced by national representation of the full weekday name.
```

```
%a    is replaced by national representation of the abbreviated weekday name.
```

```
%B    is replaced by national representation of the full month name.
```

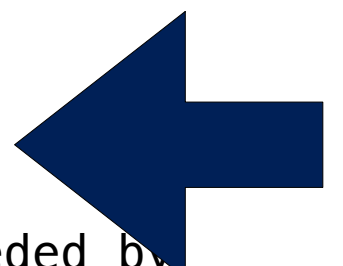
```
%b    is replaced by national representation of the abbreviated month name.
```

```
%C    is replaced by (year / 100) as decimal number; single digits are preceded by a zero.
```

```
%c    is replaced by national representation of time and date.
```

```
%D    is equivalent to ``%m/%d/%y''.
```

```
%d    is replaced by the day of the month as a decimal number (01–31).
```



Apparently my extraction didn't work well..

```
In [44]: datetime.datetime.strptime("28/Dec/2012", "%d/%b/%Y")
Out[44]: datetime.datetime(2012, 12, 28, 0, 0)
```

```
In [45]: def extract(line):
...:     bracket = line.find('[')
...:     return datetime.datetime.strptime(line[bracket:], "%d/%b/%Y")
...:
```

```
In [47]: l = '180.241.12.227 - - [28/Dec/2012:06:43:35 -0800] "GET /w/load.php?
debug=false&lang=en&modules=jquery.checkboxShiftClick%2Ccookie%2CmakeCollapsible%2CmessageBox%2C1.1" 200 11293 "http://
www.forensicswiki.org/wiki/Main_Page" "Mozilla/5.0 (X11; Linux i686) AppleWebKit/536.11 (KHTML, like Gecko) Ubuntu/12.04 Chromium/
20.0.1132.47 Chrome/20.0.1132.47 Safari/536.11"'
```

```
In [48]: extract(l)
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-48-8b69f0a350bf> in <module>()
----> 1 extract(l)
```

```
<ipython-input-45-1bf51943eae1> in extract(line)
      1 def extract(line):
      2     bracket = line.find('[')
----> 3     return datetime.datetime.strptime(line[bracket:], "%d/%b/%Y")
```

```
/usr/lib64/python3.4/_strptime.py in _strptime_datetime(cls, data_string, format)
    498     """Return a class cls instance based on the input string and the
    499     format string."""
--> 500     tt, fraction = _strptime(data_string, format)
    501     tzname, gmtoff = tt[-2:]
    502     args = tt[:6] + (fraction,)
```

```
/usr/lib64/python3.4/_strptime.py in _strptime(data_string, format)
    335     if not found:
    336         raise ValueError("time data %r does not match format %r" %
--> 337             (data_string, format))
    338     if len(data_string) != found.end():
    339         raise ValueError("unconverted data remains: %s" %
```

```
ValueError: time data '[28/Dec/2012:06:43:35 -0800] "GET /w/load.php?
debug=false&lang=en&modules=jquery.checkboxShiftClick%2Ccookie%2CmakeCollapsible%2CmessageBox%2CmwPrototypes%2Cplaceholder%7Cmediawik
i.language%2Cuser%2Cutil%7Cmediawiki.legacy.ajax%2Cwikibits%7Cmediawiki.page.ready&skin=monobook&version=20120730T153329Z&* HTTP/1.1"
200 11293 "http://www.forensicswiki.org/wiki/Main_Page" "Mozilla/5.0 (X11; Linux i686) AppleWebKit/536.11 (KHTML, like Gecko) Ubuntu/
12.04 Chromium/20.0.1132.47 Chrome/20.0.1132.47 Safari/536.11"' does not match format '%d/%b/%Y'
```

Use a proper regular expression, date time parser, and return a string...

```
In [51]: def extract(line):
...:     bracket = line.find('[')
...:     colon = line.find(':')
...:     return datetime.datetime.strptime(line[bracket+1:colon], "%d/%b/%Y")
...:
...:
...:
```

```
In [52]: extract(l)
Out[52]: datetime.datetime(2012, 12, 28, 0, 0)
```

```
In [53]: date_re = re.compile("\[(\d\d/[a-zA-Z]+/\d\d\d\d)\]")
```

```
In [56]: m = date_re.search(l)
```

```
In [57]: m
```

```
In [58]:
```

```
In [59]: date_re = re.compile("(\d\d/[a-zA-Z]+/\d\d\d\d)")
```

```
In [60]: m = date_re.search(l)
```

```
In [61]: m
Out[61]: <_sre.SRE_Match object; span=(20, 31), match='28/Dec/2012'>
```

```
In [62]: def extract(line):
...:     m = date_re.search(line)
...:     if m:
...:         d = datetime.datetime.strptime(m.group(1))
...:         return str(d.year) + "-" + (d.month)
...:
```

```
In [63]: Out[52].year
Out[63]: 2012
```

```
In [64]: Out[52].month
Out[64]: 12
```

Okay... we finally got it all to work!

```
In [67]: extract(l)
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-67-8b69f0a350bf> in <module>()  
----> 1 extract(l)  
  
<ipython-input-66-a0231b148705> in extract(line)  
      3     if m:  
      4         d = datetime.datetime.strptime(m.group(1), "%d/%b/%Y")  
----> 5         return str(d.year) + "-" + (d.month)  
      6
```

TypeError: Can't convert 'int' object to str implicitly

```
In [68]: def extract(line):  
...:     m = date_re.search(line)  
...:     if m:  
...:         d = datetime.datetime.strptime(m.group(1), "%d/%b/%Y")  
...:         return str(d.year) + "-" + str(d.month)  
...:  
...:
```

```
In [69]: extract(l)
```

```
Out[69]: '2012-12'
```

```
In [70]: dates = W.map(extract)
```

```
In [71]: dates.cache()
```

```
Out[71]: PythonRDD[30] at RDD at PythonRDD.scala:48
```

```
In [72]: W.cache()
```

```
Out[72]: s3://gu-anly502/logs/forensicswiki.2012.txt MapPartitionsRDD[29] at textFile at  
NativeMethodAccessorImpl.java:0
```

```
In [73]: dates.take(5)
```

```
Out[73]: ['2012-1', '2012-1', '2012-1', '2012-1', '2012-1']
```

```
In [74]: dates.countByKey()
```

```
Out[74]: defaultdict(int, {'2': 15949554})
```

.countsByKey() expect each row to be a (key,value) pair! It is counting row[0]... which is a '2'

We need to create (date,'1') pairs for each row. .countByKey() will ignore the '1'.

```
In [75]: dates = W.map(lambda line:[extract(line),1])
```

```
In [76]: dates.cache()
```

```
Out[76]: PythonRDD[33] at RDD at PythonRDD.scala:48
```

```
In [77]: dates.take(5)
```

```
Out[77]: [['2012-1', 1], ['2012-1', 1], ['2012-1', 1], ['2012-1', 1], ['2012-1', 1]]
```

```
In [78]: dates.countByKey()
```

```
Out[78]: defaultdict(int,
    {'2012-1': 1544100,
     '2012-10': 1498895,
     '2012-11': 1397343,
     '2012-12': 1396198,
     '2012-2': 1325030,
     '2012-3': 1274061,
     '2012-4': 1016456,
     '2012-5': 1173380,
     '2012-6': 1300250,
     '2012-7': 1287187,
     '2012-8': 1450426,
     '2012-9': 1284945,
     '2013-1': 1283})
```

```
In [79]: def extract(line):
```

```
    ...:     m = date_re.search(line)
```

```
    ...:     if m:
```

```
    ...:         d = datetime.datetime.strptime(m.group(1), "%d/%b/%Y")
```

```
    ...:         return "{:04}-{:02}".format(d.year,d.month)
```

```
    ...:
```

```
In [80]: extract(l)
```

```
Out[80]: '2012-12'
```

Okay, this seems to work

So now put it all together!

```
In [81]: dates = W.map(lambda line:[extract(line),1])
```

```
In [82]: dates.cache()
```

```
Out[82]: PythonRDD[36] at RDD at PythonRDD.scala:48
```

```
In [83]: dates.take(5)
```

```
Out[83]:
```

```
[['2012-01', 1],  
 ['2012-01', 1],  
 ['2012-01', 1],  
 ['2012-01', 1],  
 ['2012-01', 1]]
```

```
In [84]: dates.countByKey()
```

```
Out[84]:
```

```
defaultdict(int,  
             {'2012-01': 1544100,  
              '2012-02': 1325030,  
              '2012-03': 1274061,  
              '2012-04': 1016456,  
              '2012-05': 1173380,  
              '2012-06': 1300250,  
              '2012-07': 1287187,  
              '2012-08': 1450426,  
              '2012-09': 1284945,  
              '2012-10': 1498895,  
              '2012-11': 1397343,  
              '2012-12': 1396198,  
              '2013-01': 1283})
```

```
In [85]:
```




Spark Resources

<http://spark.apache.org/> — Your primary resource

The screenshot shows the Apache Spark website homepage. At the top is the Spark logo with the tagline "Lightning-fast cluster computing". Below the logo is a navigation bar with links: Download, Libraries, Documentation, Examples, Community, FAQ, and Apache Software Foundation. A central banner states: "Apache Spark™ is a fast and general engine for large-scale data processing."

Speed
Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.
Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.

Ease of Use
Write applications quickly in Java, Scala, Python, R.
Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it interactively from the Scala, Python and R shells.

Generality
Combine SQL, streaming, and complex analytics.
Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application.

A bar chart titled "Lightning regression in Hadoop and Spark" compares running times. Hadoop takes 110 seconds, while Spark takes 0.0 seconds.

Word count in Spark's Python API

```
text_file = spark.textFile("hdfs://...")
text_file.flatMap(lambda line: line.split())
.map(lambda word: word)
.reduceByKey(lambda x, y: x+y)
```

Latest News
Submission is open for spark summit San Francisco (Feb 11, 2016)
Spark Summit East (Feb 16, 2016, New York) agenda posted (Jan 14, 2016)
Spark 1.6.0 released (Jan 04, 2016)
CFP for Spark Summit East 2016 is closing soon! (Nov 19, 2015)

Download Spark

Built-in Libraries:
SQL and DataFrames
Spark Streaming
MLlib (machine learning)
GraphX (graph)
Third-Party Packages

Spark SQL, Spark Streaming, MLlib (machine learning), GraphX (graph), Apache Spark

Documentation

- <https://spark.apache.org/docs/latest/>

Examples:

- <https://spark.apache.org/examples.html>

Word Count

In this example, we use a few transformations to build a dataset of (String, Int) pairs called counts and then save it to a file.

Python Scala Java

```
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
                   .map(lambda word: (word, 1)) \
                   .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

Python Scala Java

```
val textFile = sc.textFile("hdfs://...")
val counts = textFile.flatMap(line => line.split(" "))
                    .map(word => (word, 1))
                    .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

Python Scala Java

```
JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaRDD<String> words = textFile.flatMap(new FlatMapFunction<String, String>() {
    public Iterable<String> call(String s) { return Arrays.asList(s.split(" ")); }
});
JavaPairRDD<String, Integer> pairs = words.mapToPair(new PairFunction<String, String, Integer>() {
    public Tuple2<String, Integer> call(String s) { return new Tuple2<String, Integer>(s, 1); }
});
JavaPairRDD<String, Integer> counts = pairs.reduceByKey(new Function2<Integer, Integer, Integer>() {
    public Integer call(Integer a, Integer b) { return a + b; }
});
counts.saveAsTextFile("hdfs://...");
```

API:

- <https://spark.apache.org/docs/latest/api/python/index.html>

Spark Summit — Annual conference with information about Spark

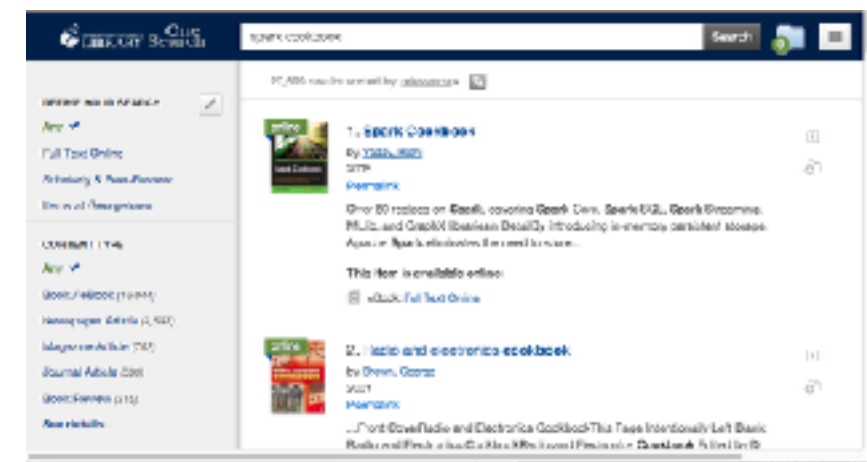
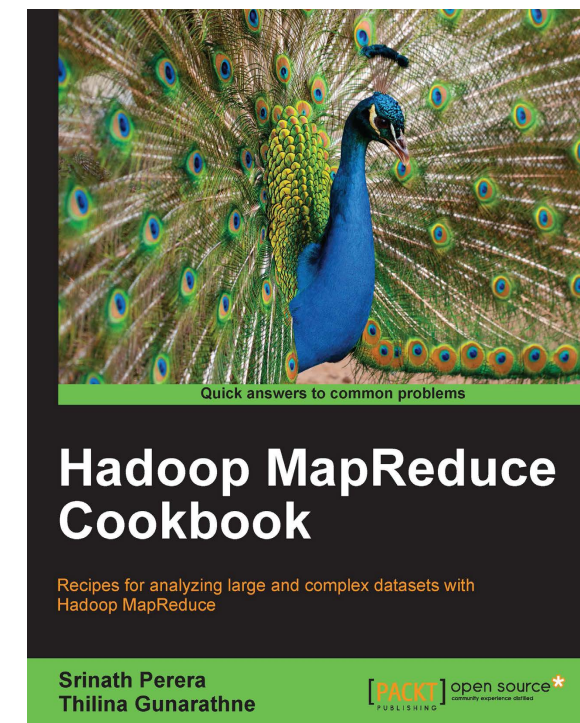
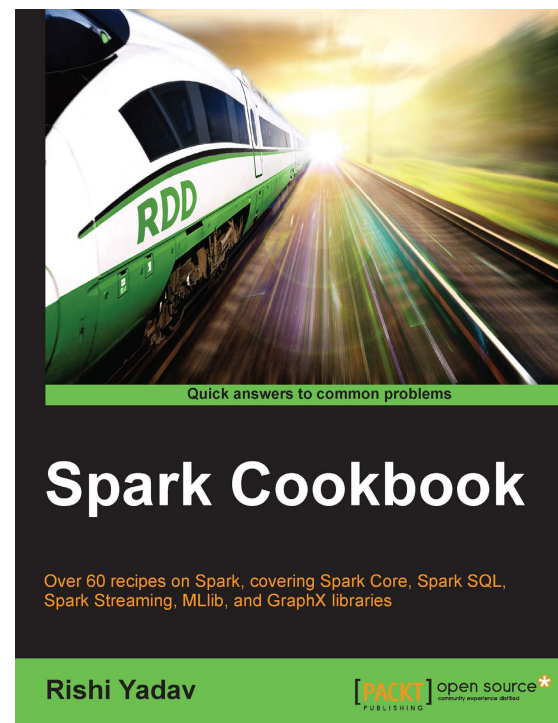
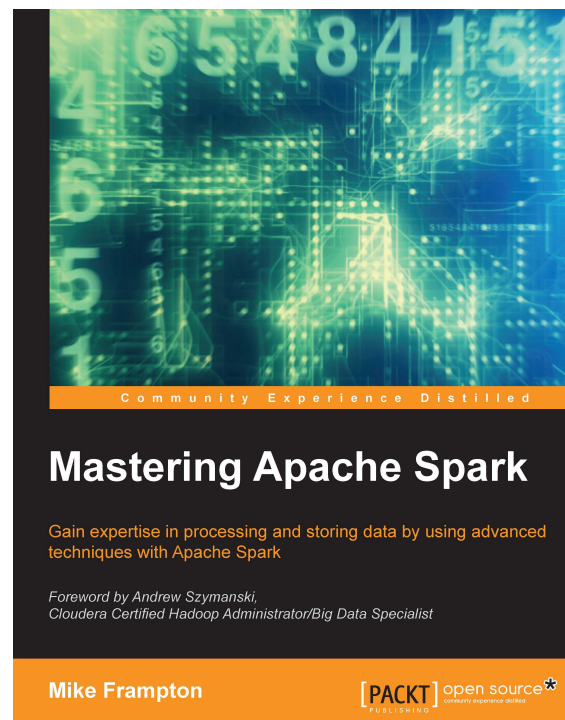
<https://spark-summit.org/2016/>

Slides, code & videos — Great source for what's new.

The image shows two overlapping browser windows. The background window is the Spark Summit 2016 website, which features a green and black header with the text "SPARK SUMMIT 2016 DATA SCIENCE AND ENGINEERING AT SCALE JUNE 6 - 8, 2016 | SAN FRANCISCO". Below the header is a navigation bar with links to Home, Schedule, Speakers, Spark Training, Venue, Job Board, and Sponsors. The main content area is titled "2016 KEYNOTE SPEAKERS" and displays a row of six speaker portraits with their names and titles: Matei Zaharia (CTO & Co-Founder, Databricks), Jeff Dean (Senior Fellow, Google), Doug Cutting (Chief Architect, Cloudera / Co-founder, Apache Hadoop, Cloudera), Andrew Ng (Chief Scientist, Baidu), Ali Ghodsi (CEO & Co-Founder, Databricks), and Marvin Thompson (Distinguished Engineer, Amazon). A "SEE THE FULL SCHEDULE" button is located at the bottom of the speaker list. The foreground window is a YouTube video player showing a video titled "Day 1 Keynotes - Spark Summit East 2016 - #SparkSummit" by the channel "SiliconANGLE". The video player shows a progress bar at 0:02 / 1:12:31 and a view count of 3,666 views. The video content shows the Spark Summit logo and the text "SPARK SUMMIT EAST 2016".

<https://www.youtube.com/watch?v=QLfjJXB5I7U>

PACKT — I got these on sale for \$5 each.
You can get them for free at library.georgetown.edu



But be careful — most of these books are for Spark 1.x, not 2.x

Some web resources

Dean Wampler's Spark Workshop

- <https://github.com/deanwampler/spark-workshop>
- 12 demos and exercises in Scala:
 - *Intro, WordCount (2), Matrix math, Web Crawler, Inverted Index, n-gram calculations, Joins, SparkSQL, Hive, and SparkStreaming.*

IPython and Qt Console for Jupyter

- <http://ipython.readthedocs.io/en/stable/>
- <https://qtconsole.readthedocs.io/en/latest/>

Old, but still useful:

- <https://ipython.org/ipython-doc/1/interactive/notebook.html>
- <https://ipython.org/ipython-doc/3/interactive/magics.html>

Jupyter Notebook tricks:

- <https://www.dataquest.io/blog/jupyter-notebook-tips-tricks-shortcuts/>

Databricks Reference Apps

- Log Analysis Application — monitor Apache access logs in batch & streaming
- Twitter Streaming Language Classifier — Spark MLlib, classifying , filtering & clustering
- Weather TimeSeries Data Application with Cassandra
 - <https://github.com/databricks/reference-apps>
 - <https://www.gitbook.com/book/databricks/databricks-spark-reference-applications/details>

Databricks Community Edition — Free cloud-version of Spark (no s3://)

— <https://databricks.com/ce>

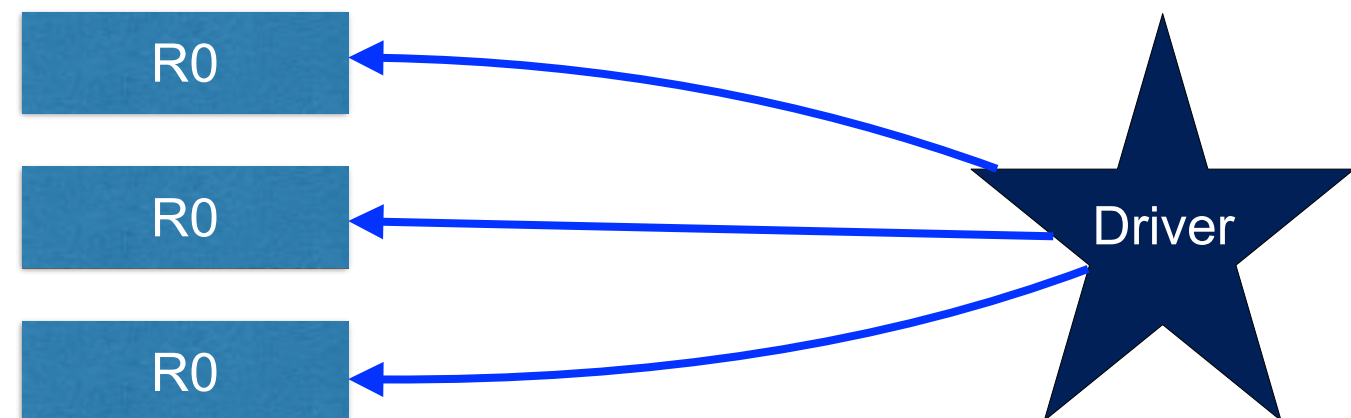
Spark programming and mini tutorial

RDD methods can be *actions* or *transformations*

Transformations create new RDDs within the worker nodes

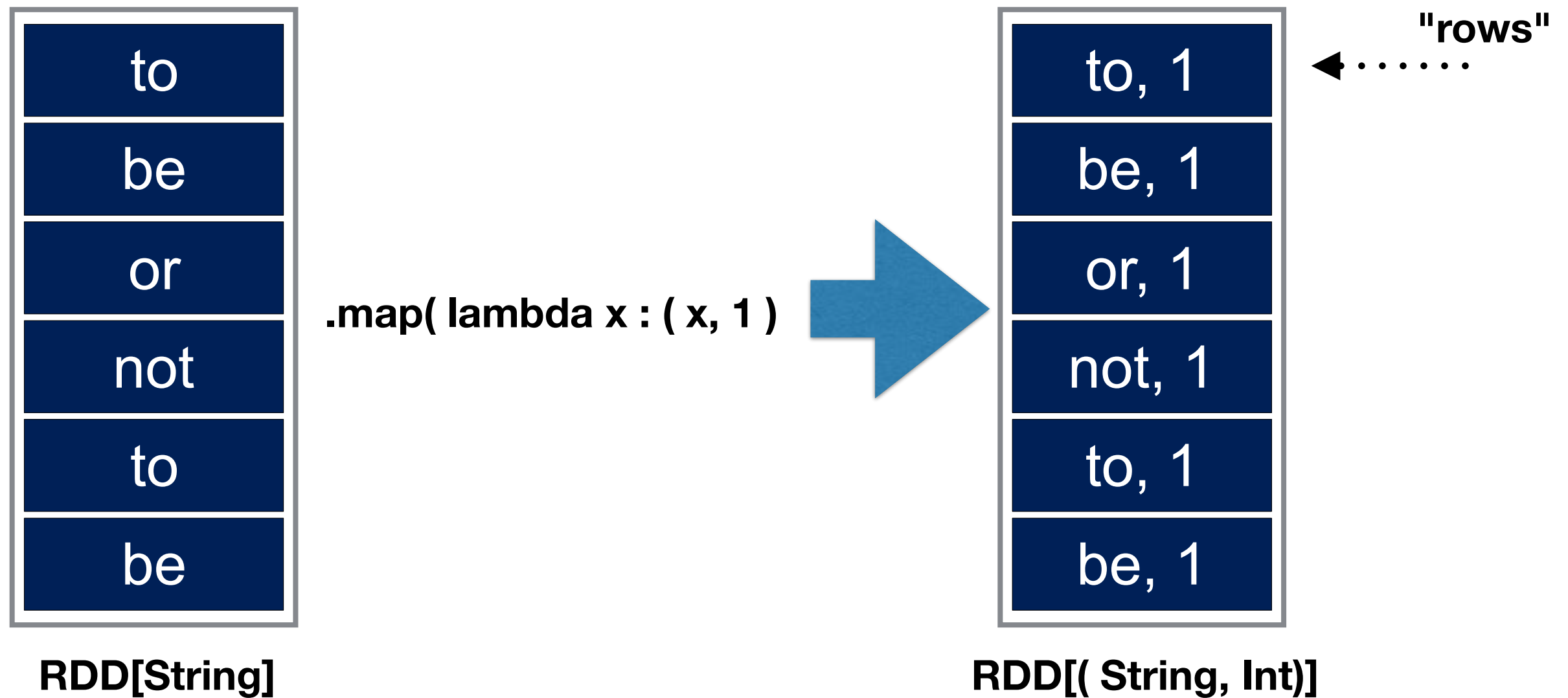


Actions move data between the worker nodes and the Driver:



Spark *transformations* produce RDDs (sometimes from other RDDs)

`.map()` is the same as mrjob's map function:



There are many transformations:

Operate on any RDD:

- `S = R.map(f)` *f should return a value*
- `S = R.flatMap(f)` *same, but f should return a Seq*
- `S = R.filter(f)` *f returns boolean True/False*
- `S = R.mapPartitions(f)`
- `S = R.mapPartitionsWithIndex(f)` *f is called with (index,val)*
- `S = R.sample(withReplacement, fraction, seed)` *Produces a random sampling*
- `S = R.union(R2)` *Produces union of two RDDs*
- `S = R.intersection(R2)` *Intersection of two RDD2*

Operate on RDDs of (k,v) pairs:

- `S = R.groupByKey([numTasks])`
- `S = R.reduceByKey(func, [numTasks])` *f must take (v1,v2)*
- `S = R.aggregateByKey(zeroValue)`
- `S = R.sortByKey([ascending])`
- `S = R.join(R2, [numTasks])`
- <https://spark.apache.org/docs/1.3.0/programming-guide.html>

Spark *actions* return values to the driver program

Operate on any RDD:

- `R.reduce(f)` = `f(f(f(f(a,b), c), d), e)...` => val
- `R.collect()` = RDD values
- `R.count()` = # of elements in RDD
- `R.first()` = first element in RDD; same as `.take(1)`
- `R.take(n)` = first n elements
- `R.takeSample(withReplacement, n [, seed])` = random sampling
— *Use `withReplacement=False` to avoid repeats.*
- `RDD.saveAsTextFile(dirname)` = Writes elements to HDFS

Operate on RDDs of (k,v) pairs:

- `RDD.countByKey()` = Returns a histogram (e.g. (k, count of k))

• <https://spark.apache.org/docs/1.3.0/programming-guide.html>

Spark commands for making RDDs and controlling the Spark Application

sc is the SparkContext.

- `R = sc.parallelize(alist)` *Turns alist into an RDD*
- `R = sc.range(start, end=None, step=1, numSlices=None)` *Creates an RDD of ints*
- `R = sc.sequenceFile(path, ...)` *Open a sequenceFile*
- `R = sc.textFile(path)` *Read a text file from HDFS, local file system, S3, etc...*
- `R = wholeTextFiles(path, minPartitions=None, use_unicode=True)` *Get a directory of text files.*

- `R = sc.union(R1, R2, R3, ...)` *Combine RDDs*

Coordinating the nodes:

- `sc.addFile(path)` *Add a file to every Spark node*
- `pyspark.SparkFiles.get()` *Gets the files that were added*

Other features

Caching — Memory/CPU trade-off.

Broadcast variables — Send data to all nodes

Accumulators — Similar to Hadoop *counters*

Tuning Spark — <http://spark.apache.org/docs/latest/tuning.html>

- Data Serialization
- Memory Tuning
- Tuning Data Structures
- Tuning Garbage Collection
- Parallelism — number of partitions — numPartitions=None uses the *default number of partitions*
- Data Locality

Caching keep the RDD after it has been evaluated.

Caching is good for Iterative algorithms & debugging

```
In [21]: A = sc.textFile("s3://gu-anly502/ps03/forensicswiki.2012.txt")
In [22]: B = A.filter(lambda line:"01/Jul/2012" in line)
In [23]: B.count()
...
16/02/28 20:03:53 INFO DAGScheduler: Job 16 finished: count at <ipython-input-23-9628edc95825>:1,
took 68.834135 s
Out[23]: 35039

In [24]: B.count()
...
16/02/28 20:05:45 INFO DAGScheduler: Job 17 finished: count at <ipython-input-24-9628edc95825>:1,
took 75.764237 s
Out[24]: 35039

In [25]: B.count()
...
16/02/28 20:16:25 INFO DAGScheduler: Job 18 finished: count at <ipython-input-25-9628edc95825>:1,
took 72.602255 s
Out[25]: 35039

In [26]: B.cache()
...
Out[26]: PythonRDD[24] at RDD at PythonRDD.scala:43

In [27]: B.count()
...
16/02/28 20:17:34 INFO DAGScheduler: Job 19 finished: count at <ipython-input-27-9628edc95825>:1,
took 69.611443 s
Out[27]: 35039

In [28]: B.count()
...
16/02/28 20:17:36 INFO DAGScheduler: Job 20 finished: count at <ipython-input-28-9628edc95825>:1,
took 1.639699 s
Out[28]: 35039
```

This was done last year; today we would use %time

Spark applications — standalone scripts

Run python program that use Spark with spark-submit:

```
#!/usr/bin/spark-submit
#
# wordcount as a pyspark

import sys
from operator import add
from pyspark import SparkContext
```

You must create the SparkContext:

```
from pyspark import SparkContext
sc = SparkContext( appName="PythonWordCount" )
```

Beware — syntax errors don't always appear on EMR

```
$ ./wordcount.py s3://gu-anly502/ps03/forensicswiki/access.log.2012-01-13.gz
$ ./wordcount.py s3://gu-anly502/ps03/forensicswiki/access.log.2012-01-13.gz
```

```
$ python wordcount.py
File "wordcount.py", line 26
    counts = lines.map(lambda line: filter(str.isalpha,line))) \
                                                                ^
SyntaxError: invalid syntax
```

```
$
```

Python2 wordcount...

sc.textFile() returns *unicode* text.

- Unicode text and strings are different in Python2, but not Python3
- Under Python2, our filtering, top-10 wordcount looks like this:


```
##
## Run WordCount on Spark
##

sc      = SparkContext( appName="PythonWordCount" )
lines   = sc.textFile( filename, 1 )
counts  = lines.flatMap(lambda line: line.split(' ')) \
              .map(lambda word: filter(unicode.isalpha,word)) \
              .map(lambda x: (x, 1)) \
              .reduceByKey(add)

top20counts = counts.sortBy(lambda x: x[1], ascending=False) \
                    .take(20)
for (word, count) in top20counts:
    print "%-10s: %i" % (word, count)

##
## Terminate the Spark job
##

sc.stop()
```



Notice
unicode.isalpha instead of
str.isalpha



old-style print

Python3 wordcount:

sc.textFile() returns *unicode* text.

```
##
## Run WordCount on Spark
##

sc      = SparkContext( appName="PythonWordCount" )
lines   = sc.textFile( filename, 1 )
counts  = lines.flatMap(lambda line: line.split(' ')) \
              .map(lambda word: filter(str.isalpha,word)) \
              .map(lambda x: (x, 1)) \
              .reduceByKey(add)

top20counts = counts.sortBy(lambda x: x[1], ascending=False) \
                    .take(20)
for (word, count) in top20counts:
    print("{:-10}: {}".format(word, count))

##
## Terminate the Spark job
##

sc.stop()
```

A module to parse Apache web logs (with Spark):

```
#!/usr/bin/env python2
#
# https://databricks.com/blog/2015/04/21/analyzing-apache-access-logs-with-databricks-cloud.html

import re                # bring in regular expression package
import dateutil, dateutil.parser
from pyspark.sql import Row
APPACHE_COMBINED_LOG_REGEX = '([(\d\.)]+) [^ ]+ [^ ]+ \[(.*)\] "(.*)" (\d+) [^ ]+ ("(.*)")? ("(.*)")?'
WIKIPAGE_PATTERN = "(index.php\?title=|/wiki/)([^\&]*)"

apache_re = re.compile(APPACHE_COMBINED_LOG_REGEX)
wikipage_re = re.compile(WIKIPAGE_PATTERN)

def parse_apache_log_line(logline):
    m = apache_re.match(logline)
    if m==None:
        raise Error("Invalid logline: {}".format(logline))

    timestamp = m.group(2)
    request    = m.group(3)
    agent      = m.group(7).replace('"','') if m.group(7) else ''

    request_fields = request.split(" ")
    url            = request_fields[1] if len(request_fields)>2 else ""
    datetime       = dateutil.parser.parse(timestamp.replace(":", " ", 1)).isoformat()
    (date,time) = (datetime[0:10],datetime[11:])

    n = wikipage_re.search(url)
    wikipage = n.group(2) if n else ""

    return Row(
        ipaddr = m.group(1),
        timestamp = timestamp,
        request = request,
        result = int(m.group(4)),
        user = m.group(5),
        referrer = m.group(6),
        agent = agent,
        url = url,
        datetime = datetime,
        date = date,
        time = time,
        wikipage = wikipage)
```

A Row object is return

Using the Apache module:

Need to get module to every python instance:

```
$ ipyspark --py-files sweblog.py
```



--py-files distributes sweblog.py

```
...
```

```
In [2]: import sweblog
```

```
In [3]: lines = sc.textFile("s3://gu-anly502/logs/forensicswiki.2012.txt")
```

```
In [4]: plines = lines.map(sweblog.parse_apache_log_line).cache()
```

```
In [5]: plines.count()
```

```
Out[5]: 52677
```

```
In [6]: plines.take(1)
```

```
Out[6]: [Row(agent='', date='2012-01-13', datetime='2012-01-13T00:05:10-08:00',  
ipaddr=u'80.243.191.178', referrer=u'http://www.forensicswiki.org/" "Mozilla/4.0  
(compatible; MSIE 7.0b; Windows NT 6.0)', request=u'GET / HTTP/1.0', result=301,  
time='00:05:10-08:00', timestamp=u'13/Jan/2012:00:05:10 -0800', url=u'/',  
user=u'"http://www.forensicswiki.org/" "Mozilla/4.0 (compatible; MSIE 7.0b; Windows  
NT 6.0)"', wikipage='')]
```

```
In [7]: plines.map(lambda x: ( x.result, 1 )).reduceByKey(operator.add).collect()
```

```
Out[7]:
```

```
[(200, 47381),  
(301, 330),  
(302, 383),  
(304, 4275),  
(403, 2),  
(404, 284),  
(206, 17),  
(503, 4),  
(500, 1)]
```

Sort on the driver node:

```
sorted(plines.map(lambda x: ( x.result, 1 ))\  
        .reduceByKey(operator.add).collect())
```

Sort with .takeOrdered():

```
plines.map(lambda x: ( x.result, 1 ))\  
        .reduceByKey(operator.add).takeOrdered(100)
```

Sort with .sortBy():

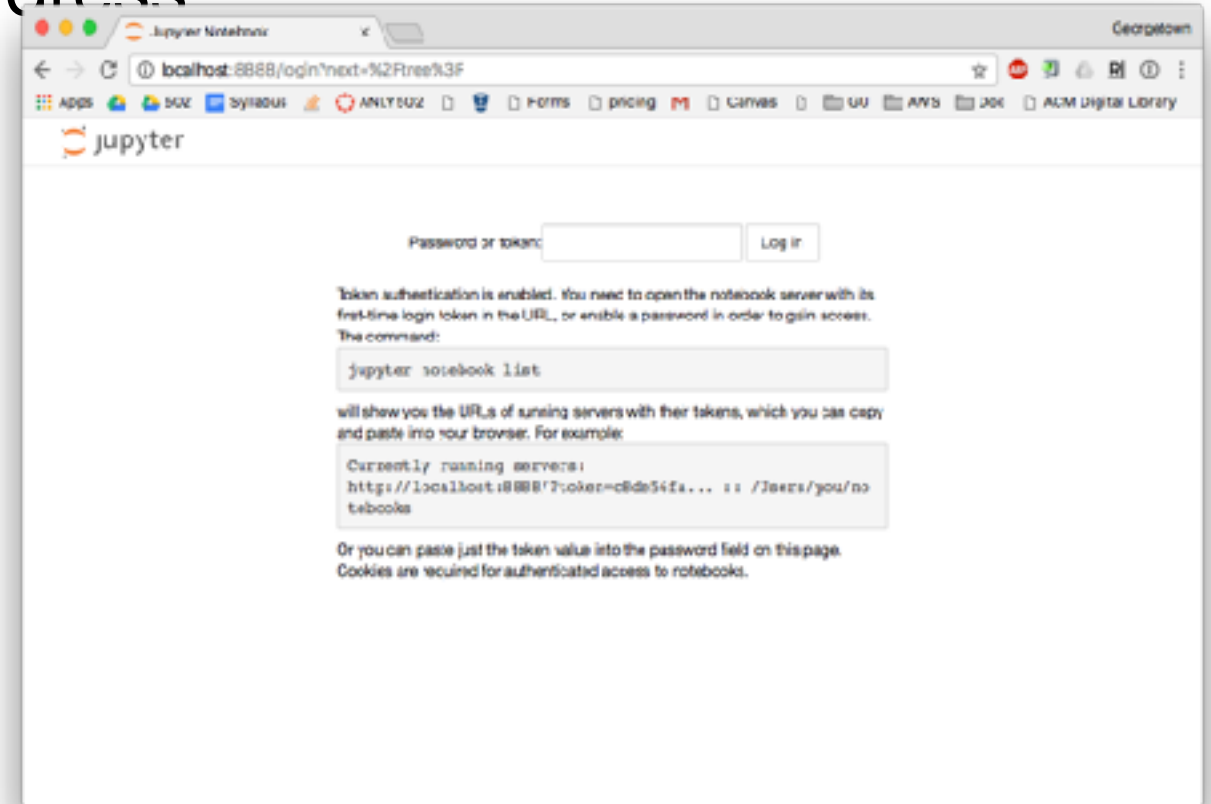
```
plines.map(lambda x: ( x.result, 1 ))\  
        .reduceByKey(operator.add)\  
        .sortBy(lambda x:x[0]).collect()
```

Jupyter notebook on EMR

Use `s3://gu-anly502/bootstrap-spark.sh`

`ssh -L8888:localhost:8888 hadoop@ipaddress`

`http://localhost:8888/`



```
$ cat .jupyter/jupyter_notebook_config.py
c.NotebookApp.open_browser = False
c.NotebookApp.ip = '*'
c.NotebookApp.port = 8888
c.NotebookApp.token = u'foo'
c.Authenticator.admin_users = {'hadoop'}
c.LocalAuthenticator.create_system_users = True
$
```



<https://www.pexels.com/photo/people-apple-iphone-writing-154/>

Homework and L07 Preview

Homework: A04

Part 1 - Tidy Data

- Turn TXT output from MTR into CSV output

Part 2 - iPython with Spark - Try it out

Part 3 - Alexa top 1M websites

- Count the .com domains
- Generate a histogram
- Find the websites that use Google Analytics

Extra Credit — Under Development....

- Mon Feb 20 — Holiday (President's Day): Reading and online homework
- Mon, Feb 27 — L06: Intro to Apache Spark
- Mon, Mar 6 — Holiday (Spring Break, Fri Mar 3 — Sun Mar 12)
- Mon, Mar 13 — L07: HBase and Spark SQL
- Fri., Mar 17 — A4 Due Today!

Lo7 — Preview

Next week:

- SQL
- Spark SQL

How many people know SQL? What do you know?

Reading!

Read this Apache Spark Documentation:

- <http://spark.apache.org/docs/latest/quick-start.html>
- <http://spark.apache.org/docs/latest/programming-guide.html>
- <http://spark.apache.org/docs/latest/submitting-applications.html>
- <http://spark.apache.org/docs/latest/cluster-overview.html>
- Databricks ["Performance of Spark 2.0's Tungsten engine."](#)

Skim the API

- <http://spark.apache.org/docs/latest/api/python/pyspark.html>
- <http://spark.apache.org/docs/latest/monitoring.html>

Scala:

- <http://www.scala-lang.org/>

Recommended blog posts:

- <http://blog.appliedinformaticsinc.com/how-to-write-spark-applications-in-python/>
- <http://www.mccarroll.net/blog/pyspark2/>