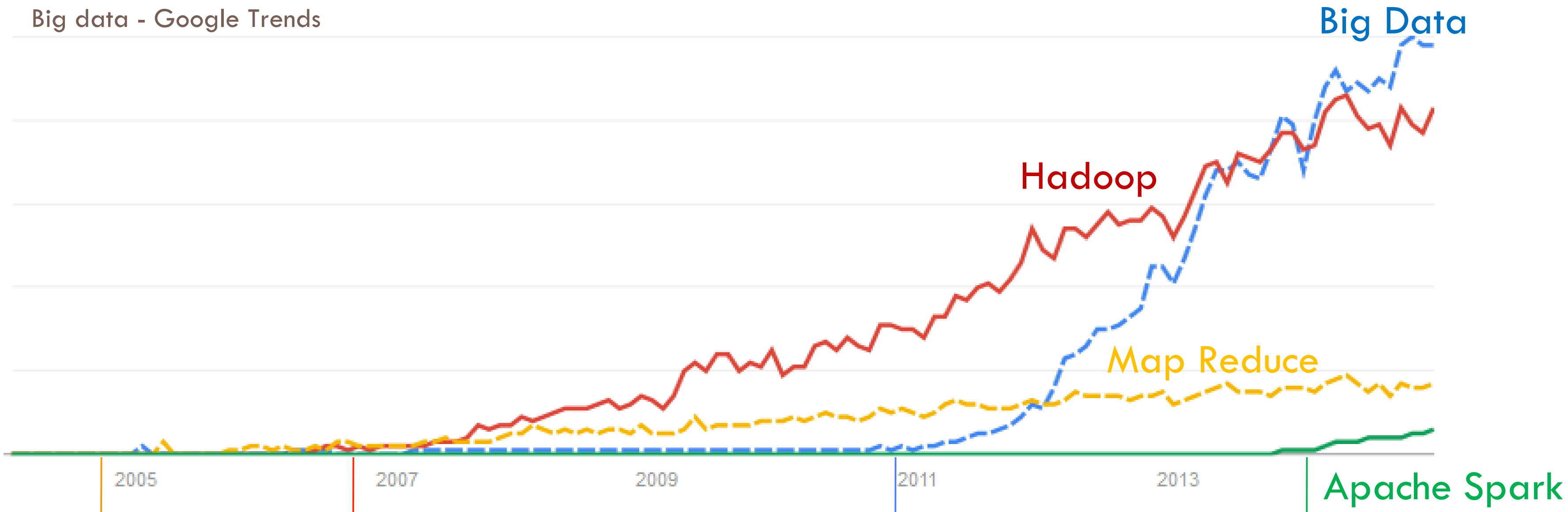
APACHE SPARK

DS8003 – MGT OF BIG DATA AND TOOLS

Ryerson University

Instructor: Kanchana Padmanabhan

Big Data History



Big data - Google Trends

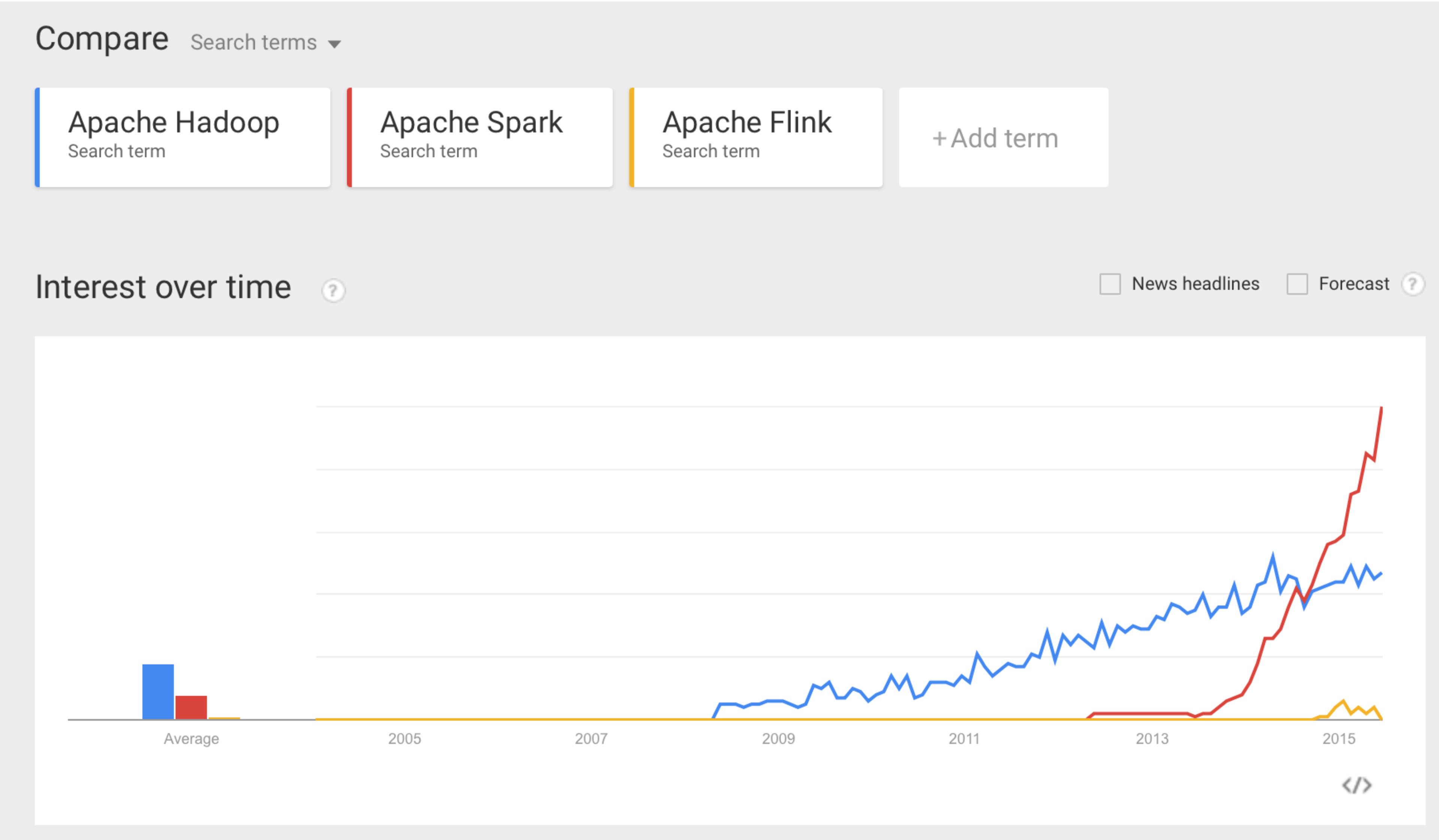
Google MapReduce Paper

Doug Cutting got hired by Yahoo! to work on Hadoop

McKinsey report, HBR coverage… Big data started to hit mainstream

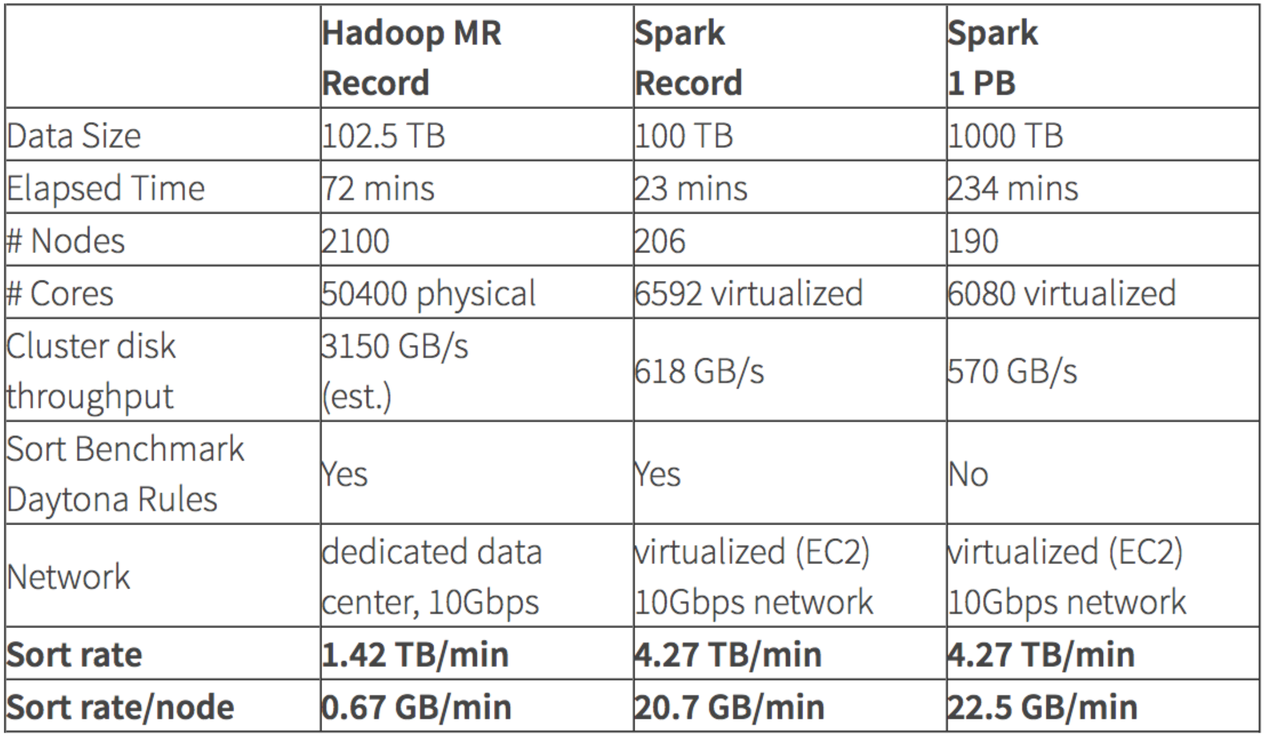
Spark took off

Big Data – New Trend

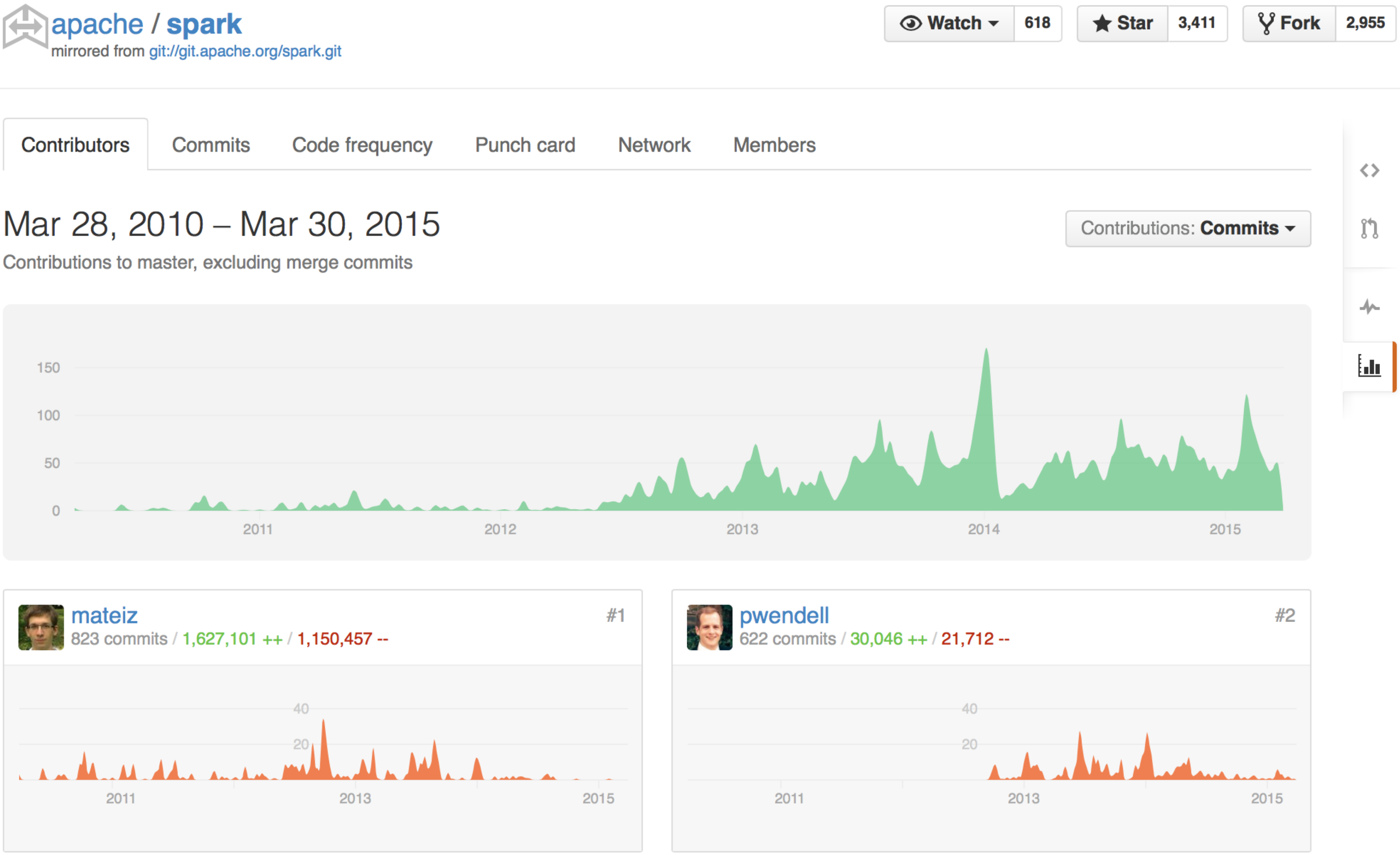


New TeraSort World Record!

- The previous world record was 72 minutes, set by Yahoo using a Hadoop MapReduce cluster of 2100 nodes.
- New world record set by Spark using Spark on 206 EC2 nodes, basically with **3X** faster using **10X** fewer machines.



	Hadoop MR Record	**Spark Record**	**Spark 1 PB**
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**	**22.5 GB/min**



apache / spark
mirrored from git://git.apache.org/spark.git

Mar 28, 2010 – Mar 30, 2015

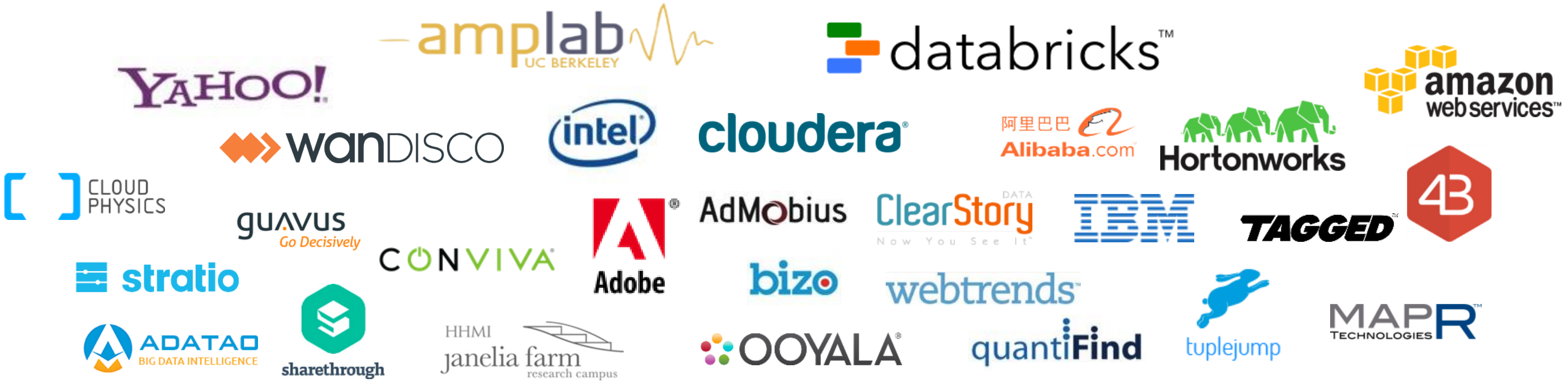
Contributions to master, excluding merge commits

Contributions: Commits

mateiz #1
823 commits / 1,627,101 ++ / 1,150,457 --

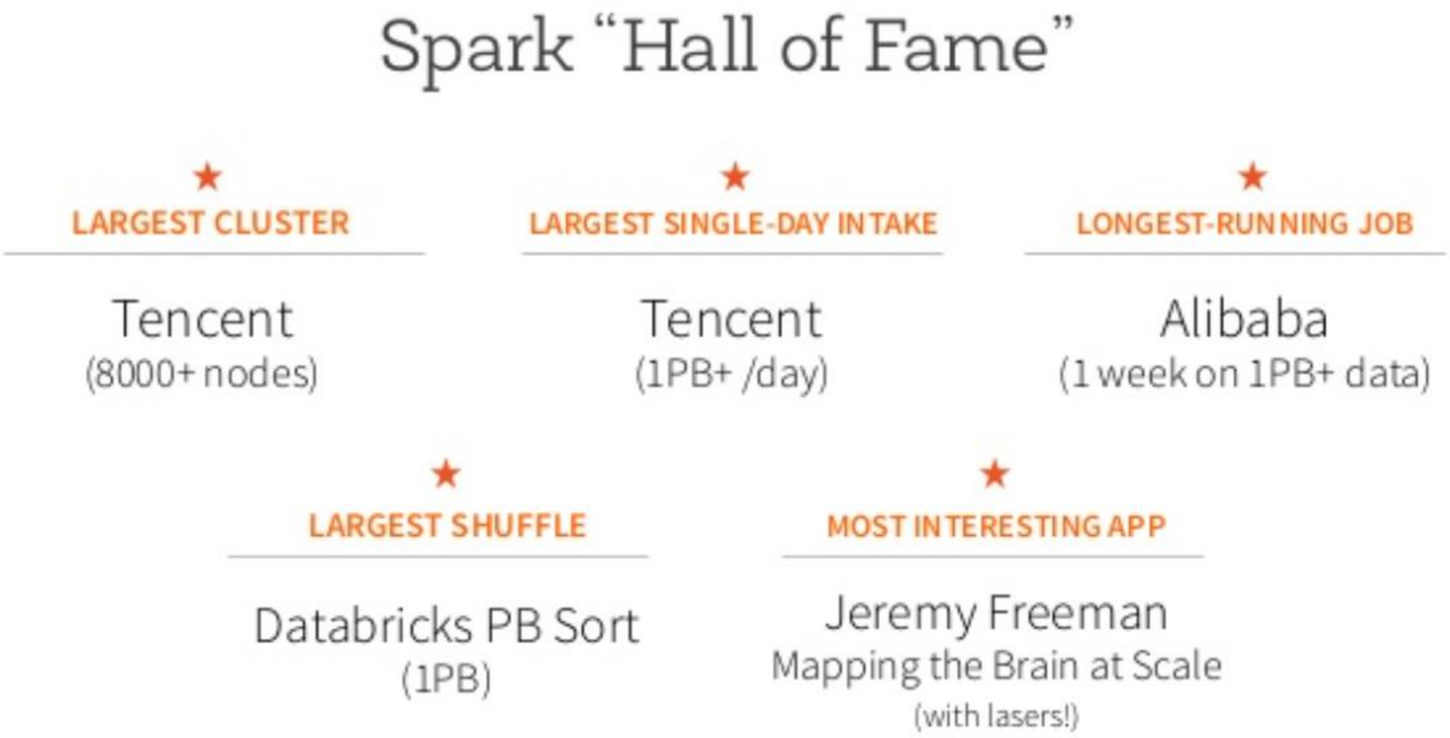
pwendell #2
622 commits / 30,046 ++ / 21,712 --

Spark Adoptions



- □ Yahoo! – Personalization and ad analytics
- □ Conviva – Real-time video stream optimization
- □ Ooyala – Cross-device personalized video experience
- □ Groupon, Shopify, Alibaba, Taobao, Tencent, etc…

Spark Hall of Fame



Spark “Hall of Fame”

★ LARGEST CLUSTER

Tencent
(8000+ nodes)

★ LARGEST SINGLE-DAY INTAKE

Tencent
(1PB+ /day)

★ LONGEST-RUNNING JOB

Alibaba
(1 week on 1PB+ data)

★ LARGEST SHUFFLE

Databricks PB Sort
(1PB)

★ MOST INTERESTING APP

Jeremy Freeman
Mapping the Brain at Scale
(with lasers!)

Apache Spark History

- Spark was initially started by Matei Zaharia at UC Berkeley AMPLab in 2009
- Open sourced in 2010
- Donated to Apache Foundation in 2013
- Became an Apache Top-Level Project in Feb 2014

The Berkeley AMPLab

Governmental and industrial funding:

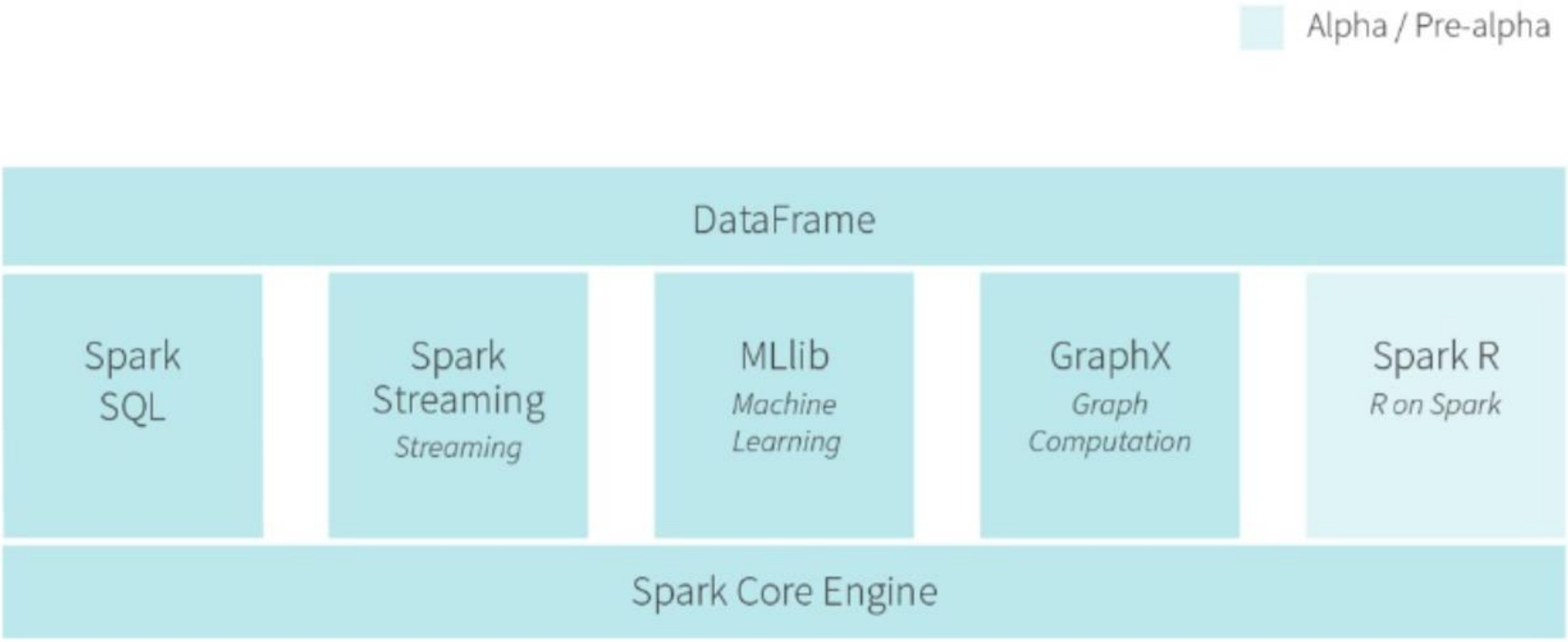
NSF
DARPA
amazon.com web services
Google
SAP
YAHOO!
intel
facebook
Microsoft
GE imagination at w
CISCO
ORACLE
SAMSUNG
HUAWEI
ERICSSON
NetApp
Go further, faster
vmware
ClearStory DATA
Now You See It
cloudera
FIT wave
Hortonworks
splunk>
wanDISCO

Goal: Next generation of open source data analytics stack for industry & academia: Berkeley Data Analytics Stack (BDAS)

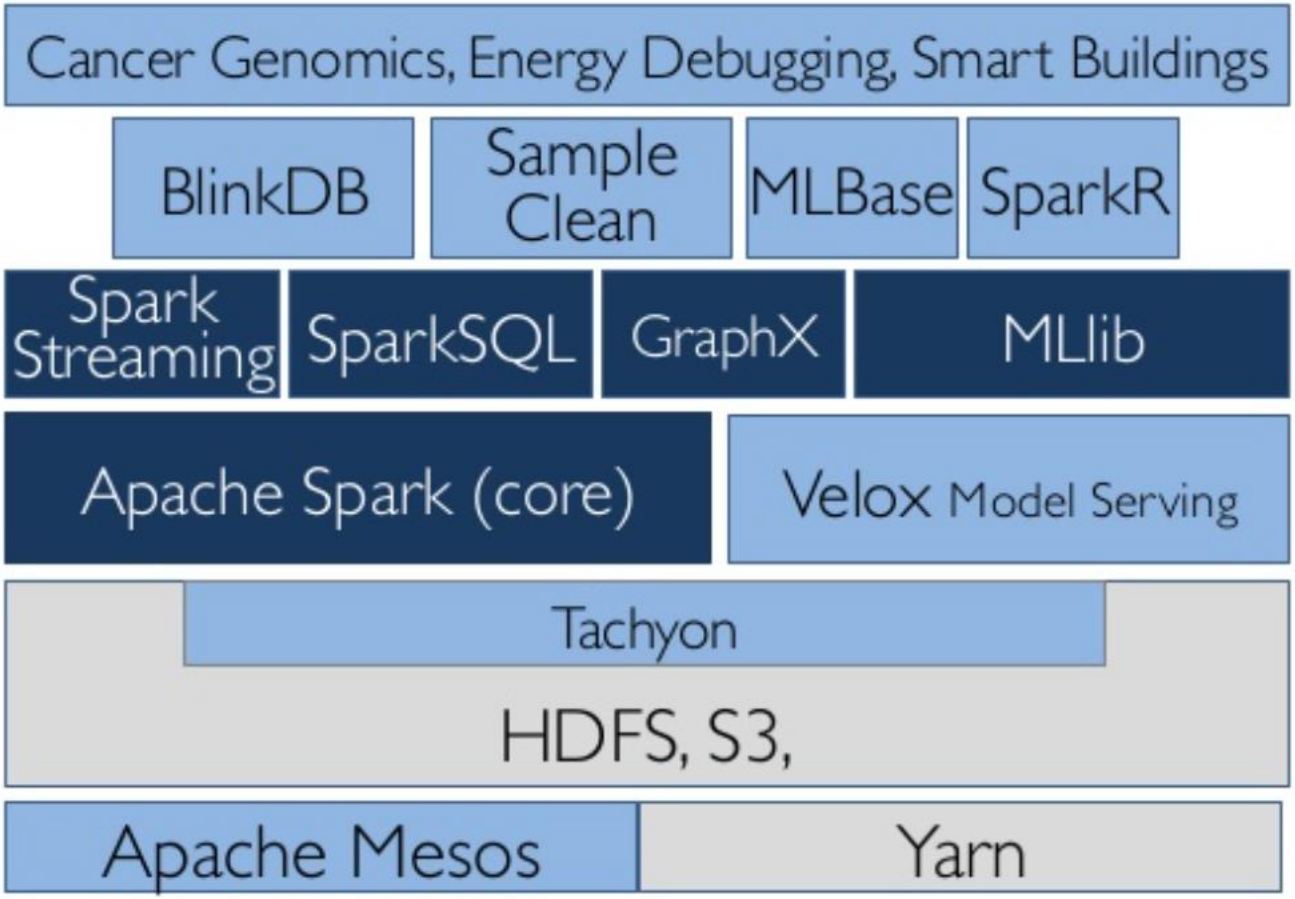
DataDricks

databricks™

- Databricks is a company founded by the creators of Apache Spark, that aims to help clients with cloud-based big data processing using Spark



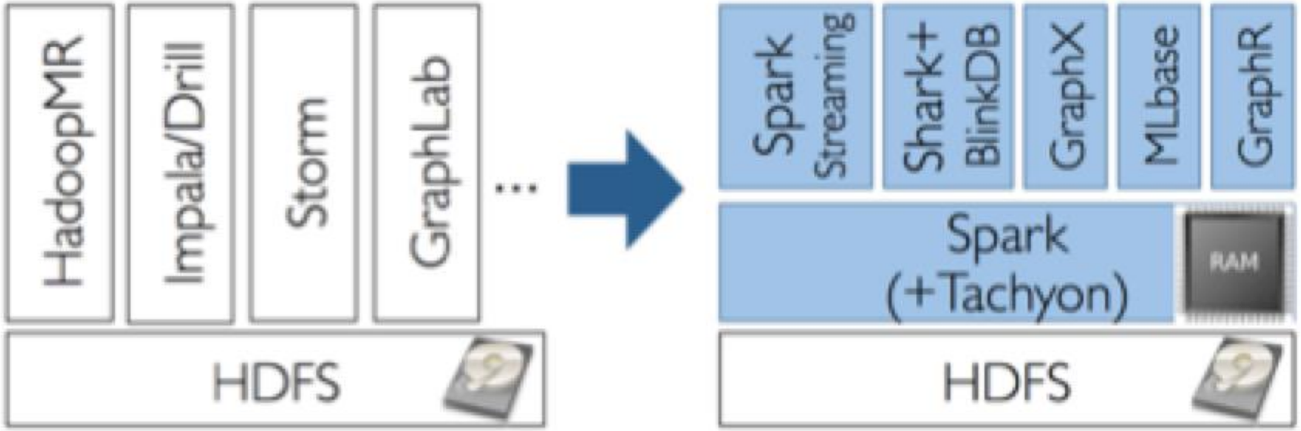
The BDAS – Berkeley Data Analytics Stack



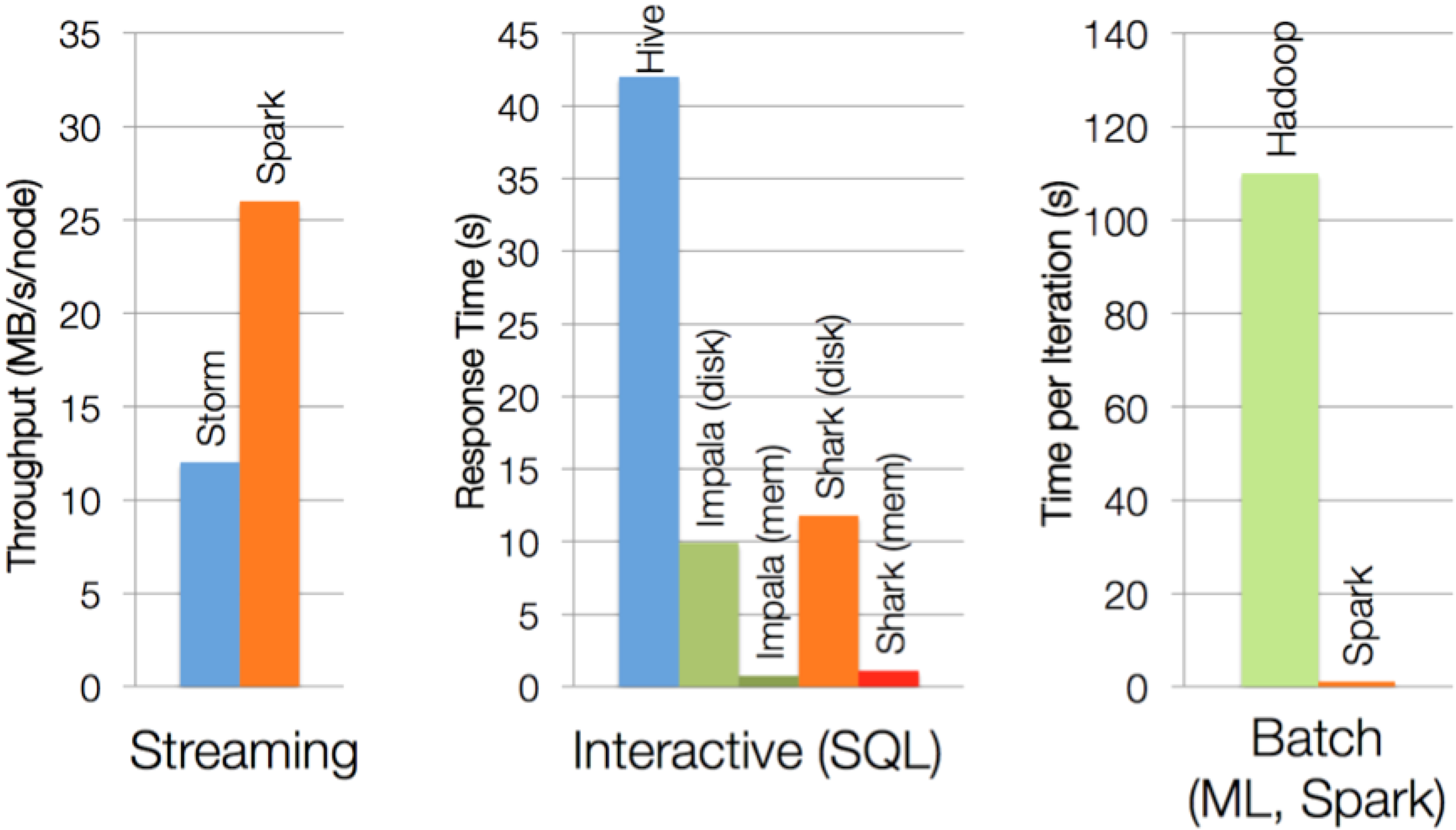
Unified Data Platform

- A unified platform that supports many data processing needs including
  - Batch processing (Spark)
  - Stream processing (SparkX)
  - Interactive (Spark SQL)
  - Iterative (MLlib, GraphX)

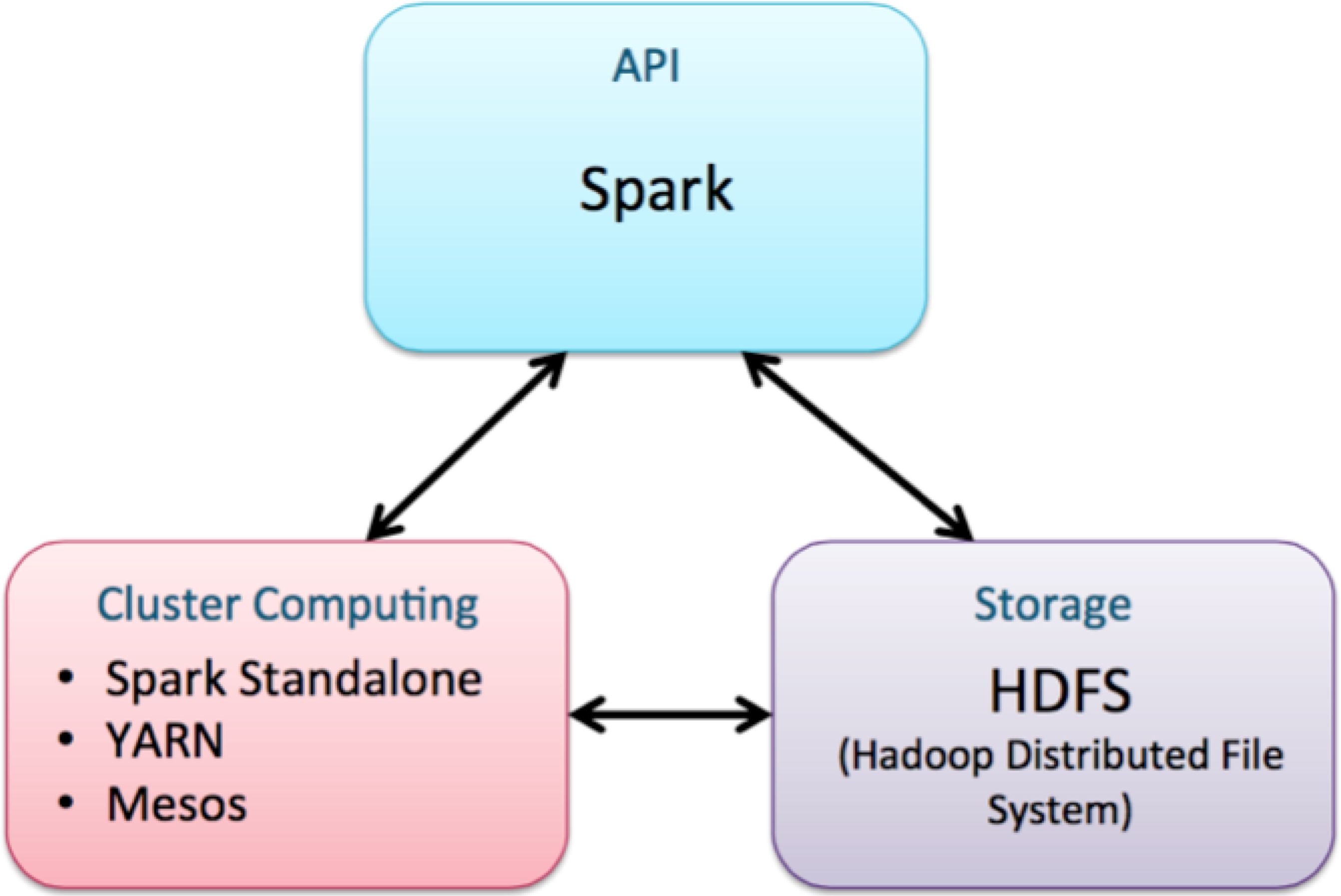
One size fits many!



Performance Benchmarks



Distributed Computing with Spark



API

Spark

Cluster Computing

- Spark Standalone
- YARN
- Mesos

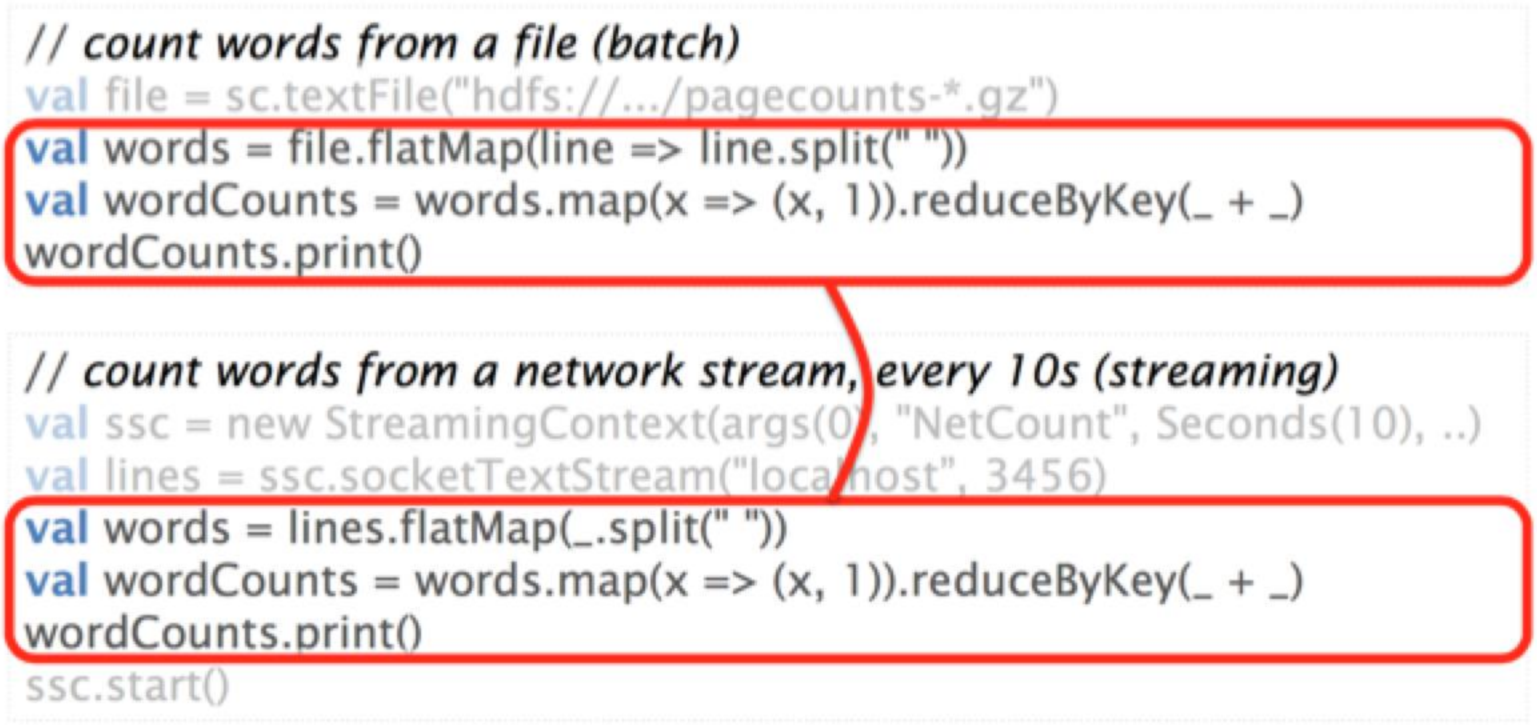
Storage

HDFS

(Hadoop Distributed File System)

Unify Real-Time and Historical Analytics

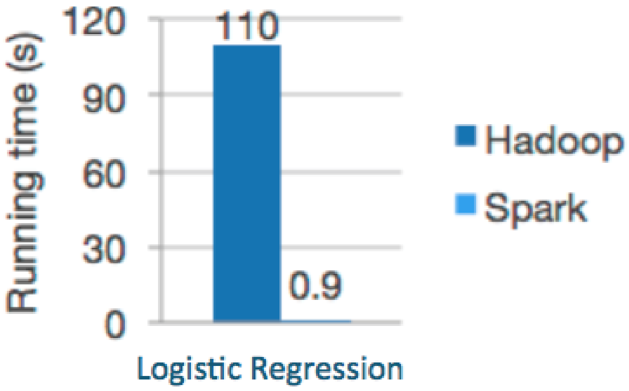
- Spark allows one to write virtually the same batch and streaming codes
  - Easy to develop and maintain consistency

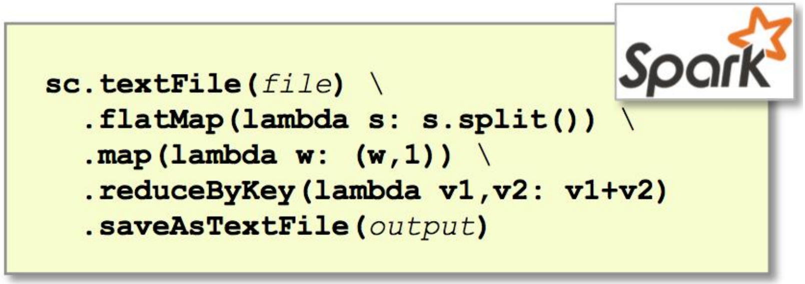


```
// count words from a file (batch)
val file = sc.textFile("hdfs://.../pagecounts-*.gz")
val words = file.flatMap(line => line.split(" "))
val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
wordCounts.print()

// count words from a network stream, every 10s (streaming)
val ssc = new StreamingContext(args(0), "NetCount", Seconds(10), ..)
val lines = ssc.socketTextStream("localhost", 3456)
val words = lines.flatMap(_.split(" "))
val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
wordCounts.print()
ssc.start()
```

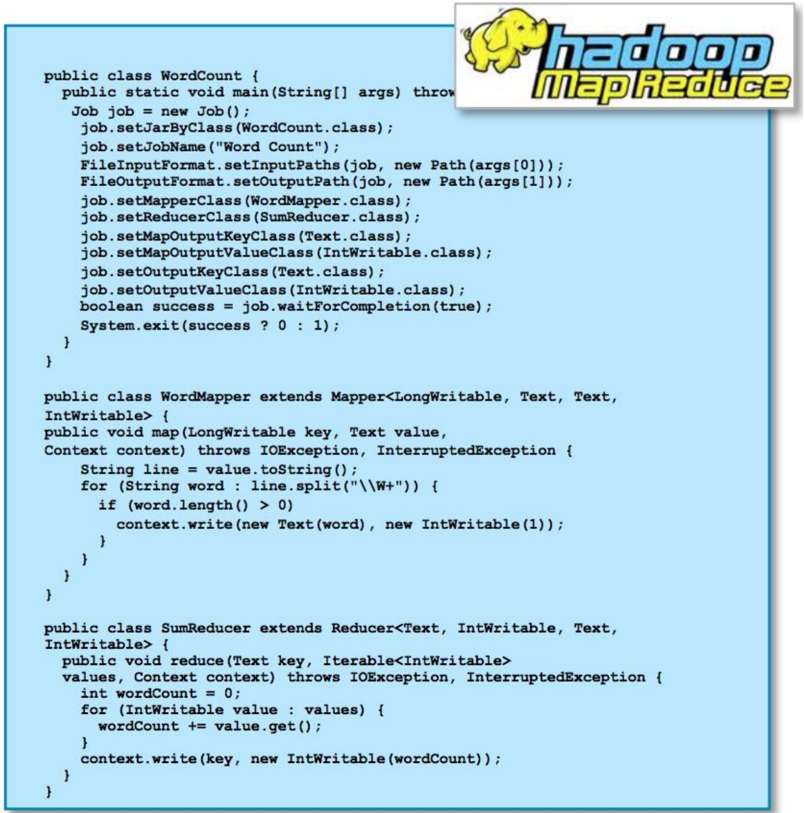
Spark v. Hadoop MapReduce

- Spark takes the concepts of MapReduce to the next level!
  - Higher-level API = faster, easier development
  - Low latency = near real-time processing
  - In-memory data storage = up to 100x performance improvement





```
sc.textFile(file) \
   .flatMap(lambda s: s.split()) \
   .map(lambda w: (w,1)) \
   .reduceByKey(lambda v1,v2: v1+v2)
   .saveAsTextFile(output)
```



```
public class WordCount {
  public static void main(String[] args) throw
   Job job = new Job();
    job.setJarByClass(WordCount.class);
    job.setJobName("Word Count");
    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.setMapperClass(WordMapper.class);
    job.setReducerClass(SumReducer.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    boolean success = job.waitForCompletion(true);
    System.exit(success ? 0 : 1);
  }
}

public class WordMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {
public void map(LongWritable key, Text value,
Context context) throws IOException, InterruptedException {
    String line = value.toString();
    for (String word : line.split("\\W+")) {
      if (word.length() > 0)
        context.write(new Text(word), new IntWritable(1));
      }
    }
  }
}

public class SumReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {
  public void reduce(Text key, Iterable<IntWritable>
  values, Context context) throws IOException, InterruptedException {
    int wordCount = 0;
    for (IntWritable value : values) {
      wordCount += value.get();
    }
    context.write(key, new IntWritable(wordCount));
  }
}
```

What is Apache Spark

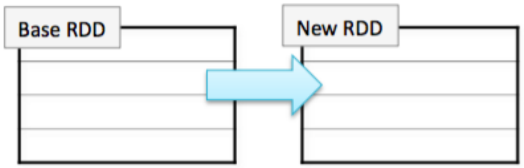
- Apache Spark is a fast and general engine for large-scale data processing
- Written in Scala
  - Functional programming language that runs in a JVM
- Spark Shell
  - Interactive – for learning or data exploration
  - Python or Scala
- Spark Application
  - For large scale data processing
  - Python, Scala, or Java

RDD (Resilient Distributed Dataset)

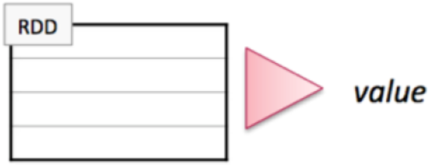
- **Resilient Distributed Datasets**
  - Collections of objects spread across a cluster, stored in RAM or on Disk
    - Analogous to HDFS but in memory
    - Still works efficiently on disks
  - Resilient – if data in memory is lost, it can be recreated
  - Distributed – stored in memory across the cluster
  - Dataset – initial data can come from a file or be created programmatically
- RDDs are the fundamental unit of data in Spark
- Most Spark programming consists of performing operations on RDDs
- Automatically rebuilt on failure

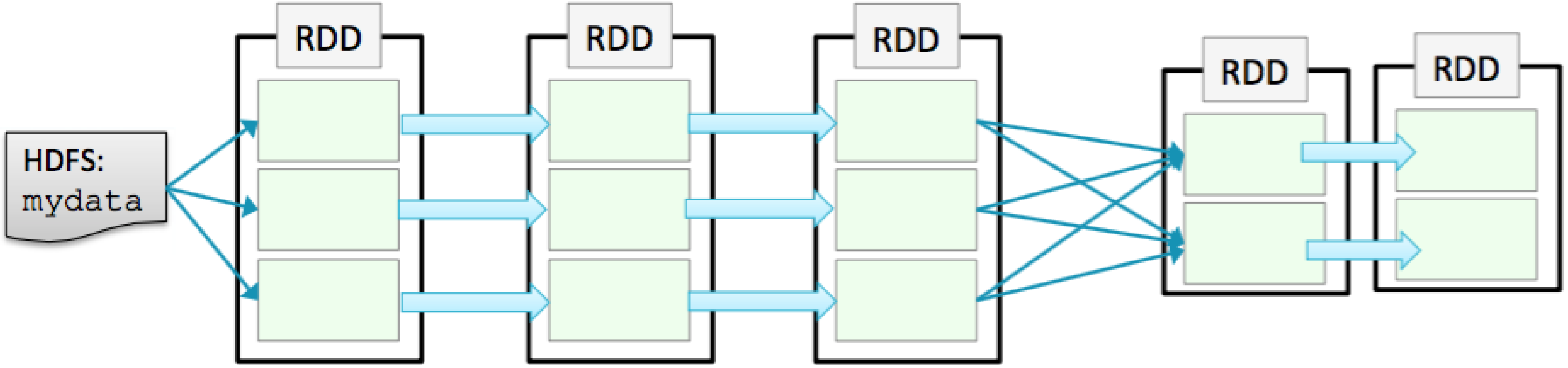
RDD Operations

- Transformations → create new RDDs
  - (e.g. map, filter, groupBy)



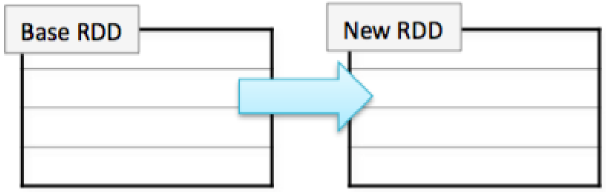
- Actions → returns value
  - (e.g. count, collect, save)





RDD Operations: Transformations

- Transformations create a new RDD from an existing one
- RDDs are immutable
  - Data in an RDD is never changed
  - Transform in sequence to modify the data as needed



- Some common transformations
  - map(function) – creates a new RDD by performing a function on each record in the base RDD
  - filter(function) – creates a new RDD by including or excluding each record in the base RDD according to a boolean function

RDD Operations

- map
- filter
- groupBy
- sort
- union
- join
- leftOuterJoin
- rightOuterJoin

- reduce
- count
- fold
- reduceByKey
- groupByKey
- cogroup
- cross
- zip

sample

take

first

partitionBy

mapWith

pipe

save ...

Good Explanations:
https://trongkhoanguyenblog.wordpress.com/2014/11/27/understand-rdd-operations-transformations-and-actions/

RDD Operations Explained

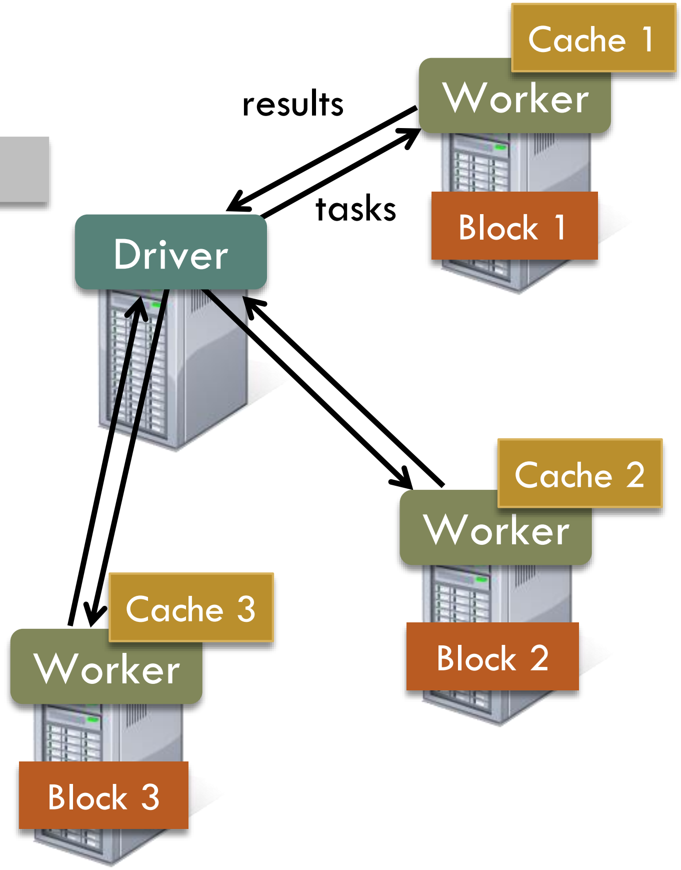
Loading messages from a log into memory and search for various patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()

messages.filter(lambda s: “mysql” in s).count()

messages.filter(lambda s: “php” in s).count()

. . .
```

Base RDD

Transformed RDD

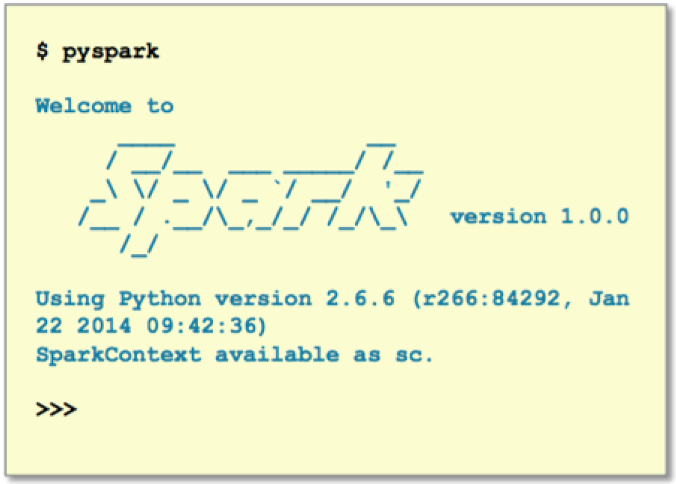
Action



Starting Spark Shell

- The Spark Shell provides interactive data exploration
- Similar to the “hive” or “pig” command
- Set environment variable in linux
  - export SPARK_HOME=/usr/hdp/2.3.2.0-2950/spark/
  - export PATH=/usr/hdp/2.3.2.0-2950/spark/bin:$PATH

Python Shell: **pyspark**



```
$ pyspark

Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.0.0
      /_/

Using Python version 2.6.6 (r266:84292, Jan
22 2014 09:42:36)
SparkContext available as sc.

>>>
```

REPL: Read/Evaluate/Print Loop

Creating RDDs

```
# Turn a Python collection into an RDD
sc.parallelize([1, 2, 3])

# Load text file from local FS
sc.textFile("file:///root/lab/full_text.txt")
# Load text file from  HDFS
sc.textFile(“/user/root/shakespeare_100.txt”)
sc.textFile(“/user/root/genre/*”)
sc.textFile(“hdfs://sandbox.hortonworks.com:8020/user/root/shakespeare_100.txt”)
```

Basic Transformations

```
    nums = sc.parallelize([1, 2, 3])
    text = sc.textFile(“/user/root/shakespeare_100.txt”)

# Map each element to zero or more others
    nums.flatMap(lambda x:  range(x))
    words  =  text.map(lambda line: line.split())

# Pass each element through a function
    squares = nums.map(lambda x: x*x)
    wordWithCount = words.map(lambda word: (word, 1))

# Keep elements passing a predicate
    even = squares.filter(lambda x: x % 2 == 0)
```

Basic Actions

```
nums = sc.parallelize([1, 2, 3])

# Retrieve RDD contents as a local collection
nums.collect() # => [1, 2, 3]

# Return first K elements
nums.take(2)  # => [1, 2]

# Count number of elements
nums.count()  # => 3

# Merge elements with an associative function
nums.reduce(lambda x, y: x + y)  # => 6

# Write elements to a text file in HDFS
nums.saveAsTextFile(“file.txt”)
# To save to local file system
X = nums.collect()
Then save using standard write operations
```

Using spark-submit

- Submit some spark commands as a python file

sparkTemplate.py

```
from pyspark import SparkConf, SparkContext

def main(sc):
    textFile = sc.textFile("shakespeare_100.txt")
    words = textFile.flatMap(lambda line: line.split())
    wordWithCount = words.map(lambda word: (word, 1))
    wordWithCount.saveAsTextFile("word_count.txt")
    collectedWords = wordWithCount.collect()
    print(collectedWords)

if __name__  == "__main__":
    conf = SparkConf().setAppName("Testing Spark Commands")
    sc = SparkContext(conf = conf)
    main(sc)
    sc.stop()
```

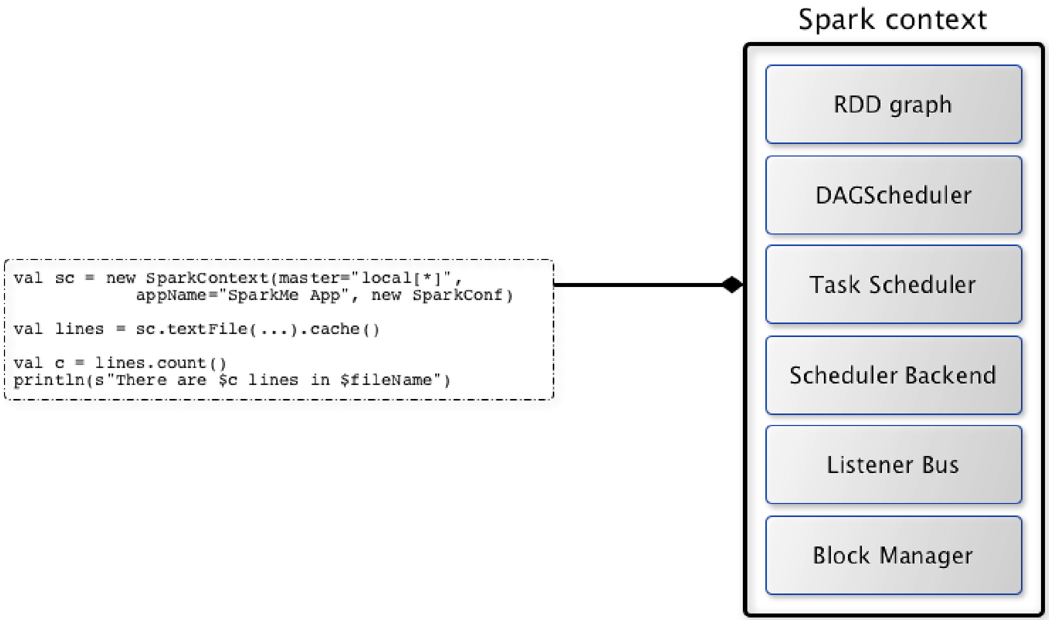
- Submit job using
  - *spark-submit --master yarn-client --executor-memory 512m --num-executors 3 --executor-cores 1 --driver-memory 512m sparkTemplate.py*

http://spark.apache.org/docs/latest/submitting-applications.html

http://hortonworks.com/hadoop-tutorial/a-lap-around-apache-spark/

Spark Context

- □ SparkContext (aka Spark context) represents the connection to a Spark execution environment
- □ You have to create a Spark context before using Spark features and services in your application.
- □ A Spark context can be used to create RDDs, accumulators and broadcast variables, access Spark services andrun jobs.

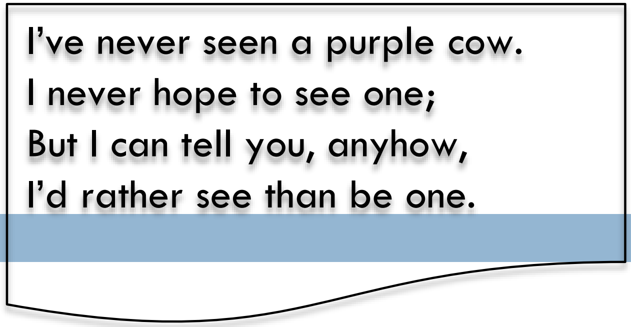


```
val sc = new SparkContext(master="local[*]",
            appName="SparkMe App", new SparkConf)

val lines = sc.textFile(...).cache()

val c = lines.count()
println(s"There are $c lines in $fileName")
```

https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-sparkcontext.html

RDD Operations

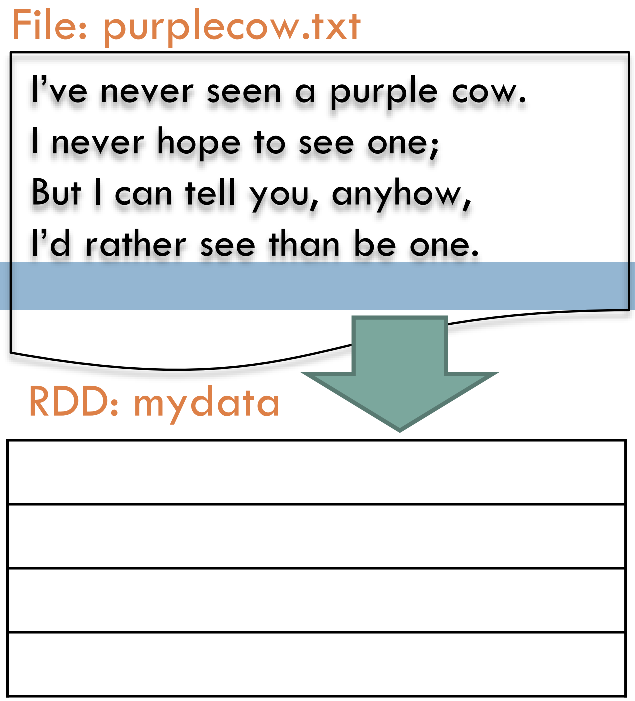
File: purplecow.txt



I’ve never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I’d rather see than be one.

RDD Operations

- mydata = sc.textFile(“shakespeare_100.txt”)



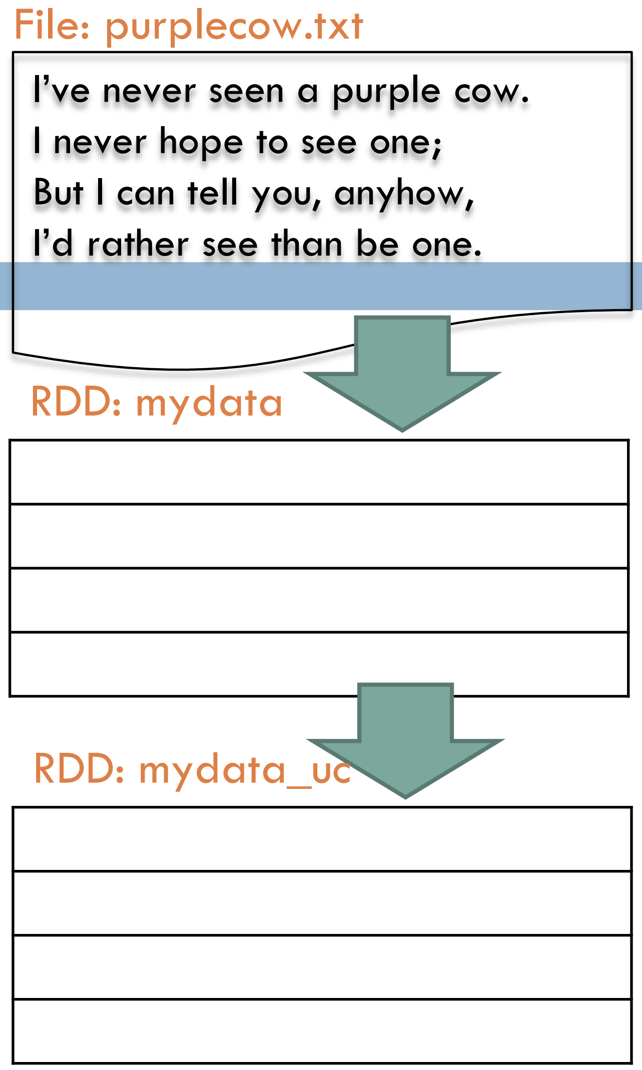
File: purplecow.txt

I’ve never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I’d rather see than be one.

RDD: mydata

RDD Operations

- mydata = sc.textFile(“shakespeare_100.txt”)
- mydata_uc = mydata.map(lambda line: line.upper())



File: purplecow.txt

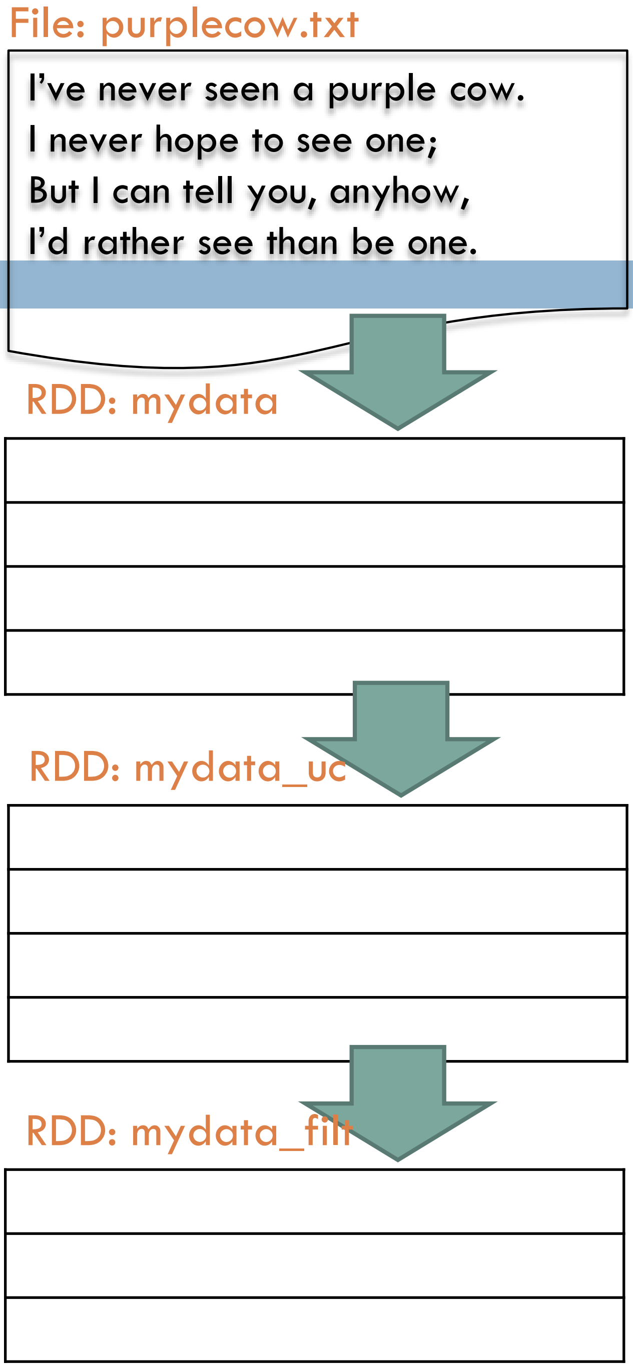
I’ve never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I’d rather see than be one.

RDD: mydata

RDD: mydata_uc

RDD Operations

- mydata = sc.textFile(“shakespeare_100.txt”)
- mydata_uc = mydata.map(lambda line: line.upper())
- mydata_filt = mydata_uc.filter(lambda line: line.startswith(‘I’))



File: purplecow.txt

I’ve never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I’d rather see than be one.

RDD: mydata

RDD: mydata_uc

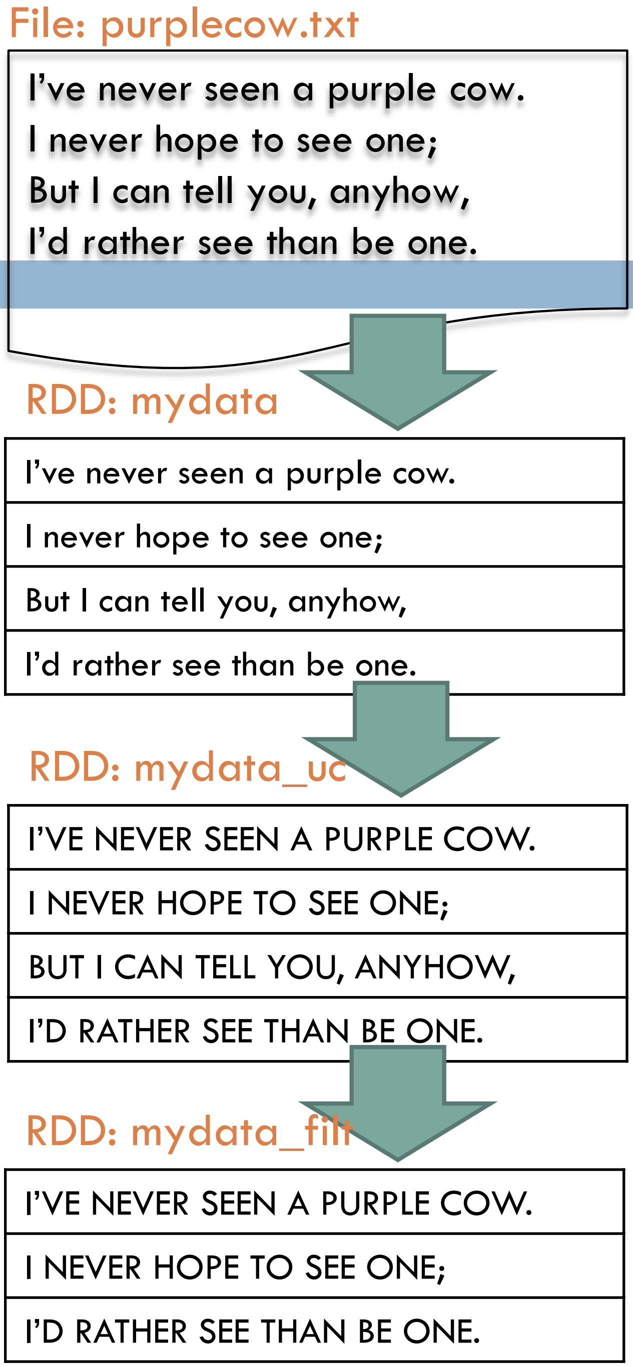
RDD: mydata_filt

RDD Operations

- mydata = sc.textFile(“shakespeare_100.txt”)
- mydata_uc = mydata.map(lambda line: line.upper())
- mydata_filt = mydata_uc.filter(lambda line: line.startswith(‘I’))
- mydata_filt.count()

Action

3



File: purplecow.txt

I’ve never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I’d rather see than be one.

RDD: mydata

I’ve never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I’d rather see than be one.

RDD: mydata_uc

I’VE NEVER SEEN A PURPLE COW.
I NEVER HOPE TO SEE ONE;
BUT I CAN TELL YOU, ANYHOW,
I’D RATHER SEE THAN BE ONE.

RDD: mydata_filt

I’VE NEVER SEEN A PURPLE COW.
I NEVER HOPE TO SEE ONE;
I’D RATHER SEE THAN BE ONE.

RDD Map/Reduce Operations

- □ MapReduce in Spark works on Pair RDDs
- □ Spark's "distributed reduce" transformations operate on RDDs of key-value pairs

Python:

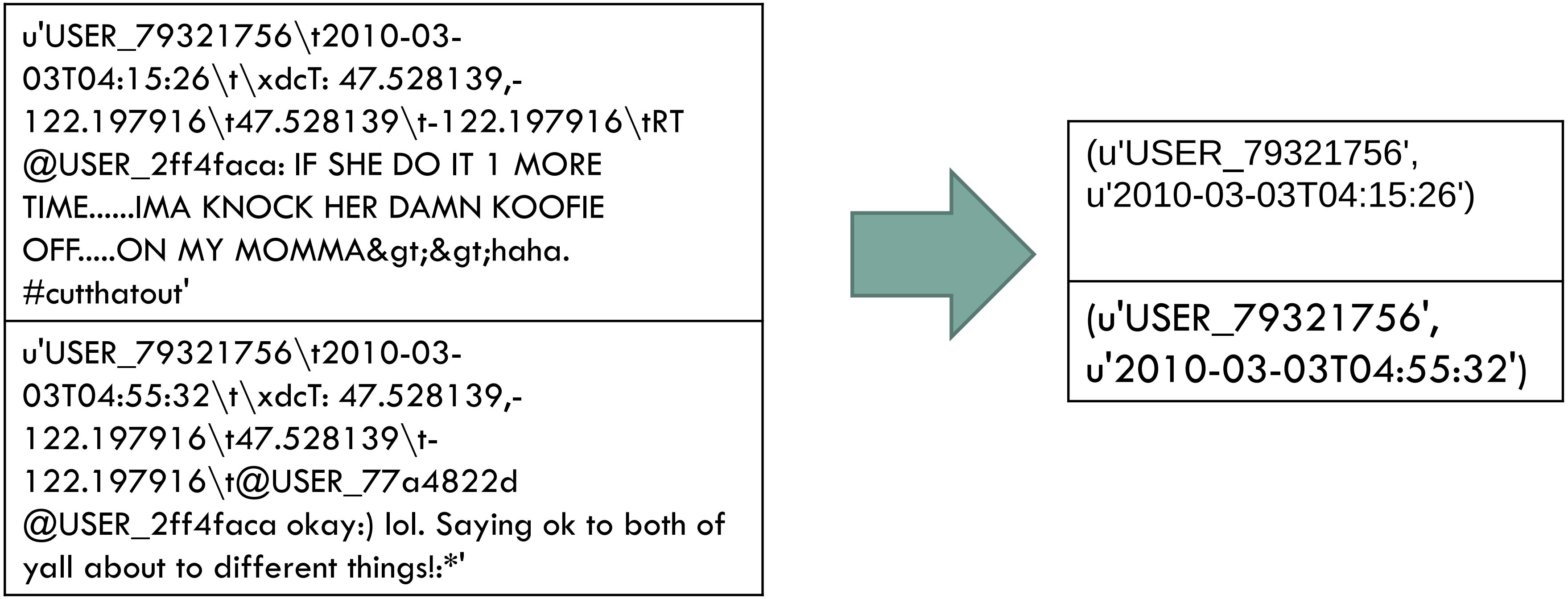
```
pair = (a, b)
pair[0] # => a
pair[1] # => b
```

Creating Pair RDDs

- The first step in most workflows is to get the data into key/value form
  - What should the RDD be keyed on?
  - What is the value?
- Commonly used functions to create Pair RDDs
  - map
  - flatMap / flatMapValues
  - keyBy

Example: A Simple Pair RDD

```
text = sc.textFile(“full_text.txt”) \
        .map(lambda line: line.split(‘\t’)) \
        .map(lambda fields: (fields[0], fields[1]))
```



u'USER_79321756\t2010-03-03T04:15:26\t\xdcT: 47.528139,-122.197916\t47.528139\t-122.197916\tRT @USER_2ff4faca: IF SHE DO IT 1 MORE TIME......IMA KNOCK HER DAMN KOOFIE OFF.....ON MY MOMMA&gt;&gt;haha. #cutthatout'

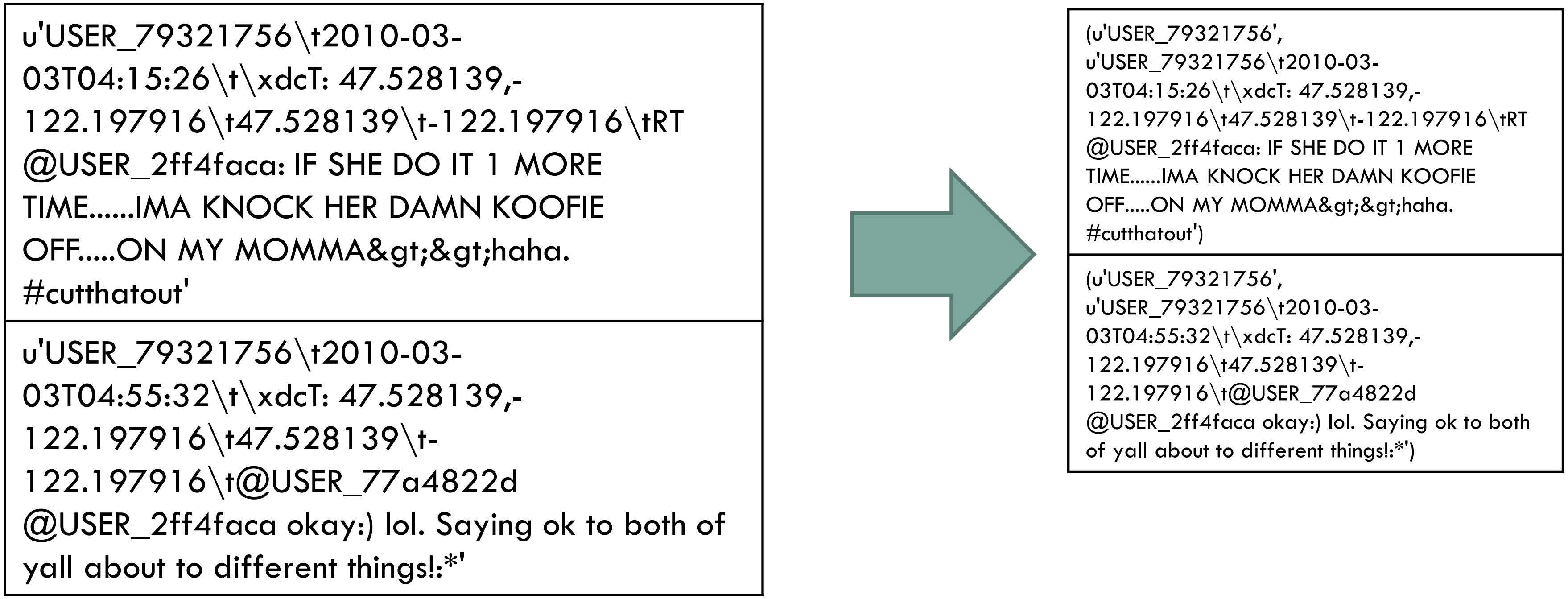
u'USER_79321756\t2010-03-03T04:55:32\t\xdcT: 47.528139,-122.197916\t47.528139\t-122.197916\t@USER_77a4822d @USER_2ff4faca okay:) lol. Saying ok to both of yall about to different things!:*'

(u'USER_79321756', u'2010-03-03T04:15:26')

(u'USER_79321756', u'2010-03-03T04:55:32')

Example: Keying Friend Pairs by Friend ID

```
text = sc.textFile("full_text.txt") \
.keyBy(lambda line: line.split("\t")[0])
#Adding a key as userID
```



```
u'USER_79321756\t2010-03-03T04:15:26\t\xdcT: 47.528139,-122.197916\t47.528139\t-122.197916\tRT @USER_2ff4faca: IF SHE DO IT 1 MORE TIME......IMA KNOCK HER DAMN KOOFIE OFF.....ON MY MOMMA&gt;&gt;haha. #cutthatout'
```

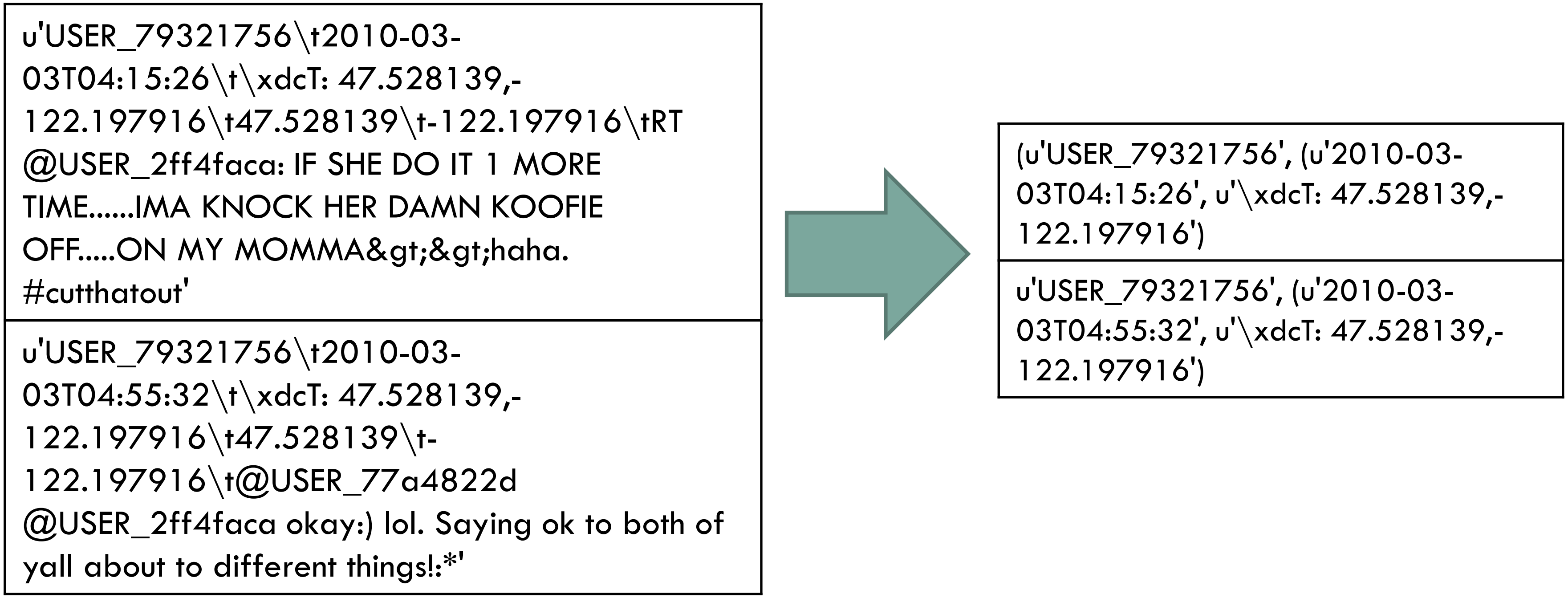
```
u'USER_79321756\t2010-03-03T04:55:32\t\xdcT: 47.528139,-122.197916\t47.528139\t-122.197916\t@USER_77a4822d @USER_2ff4faca okay:) lol. Saying ok to both of yall about to different things!:*'
```

```
(u'USER_79321756', u'USER_79321756\t2010-03-03T04:15:26\t\xdcT: 47.528139,-122.197916\t47.528139\t-122.197916\tRT @USER_2ff4faca: IF SHE DO IT 1 MORE TIME......IMA KNOCK HER DAMN KOOFIE OFF.....ON MY MOMMA&gt;&gt;haha. #cutthatout')
```

```
(u'USER_79321756', u'USER_79321756\t2010-03-03T04:55:32\t\xdcT: 47.528139,-122.197916\t47.528139\t-122.197916\t@USER_77a4822d @USER_2ff4faca okay:) lol. Saying ok to both of yall about to different things!:*')
```

Example: Pairs with Complex Values

```
text = sc.textFile(("full_text.txt") \
            .map(lambda line: line.split(‘\t’)) \
            .map(lambda fields: (fields[0], (fields[1], fields[2])))
```



```
u'USER_79321756\t2010-03-
03T04:15:26\t\xdcT: 47.528139,-
122.197916\t47.528139\t-122.197916\tRT
@USER_2ff4faca: IF SHE DO IT 1 MORE
TIME......IMA KNOCK HER DAMN KOOFIE
OFF.....ON MY MOMMA&gt;&gt;haha.
#cutthatout'
```

```
u'USER_79321756\t2010-03-
03T04:55:32\t\xdcT: 47.528139,-
122.197916\t47.528139\t-
122.197916\t@USER_77a4822d
@USER_2ff4faca okay:) lol. Saying ok to both of
yall about to different things!:*'
```

```
(u'USER_79321756', (u'2010-03-
03T04:15:26', u'\xdcT: 47.528139,-
122.197916')
```

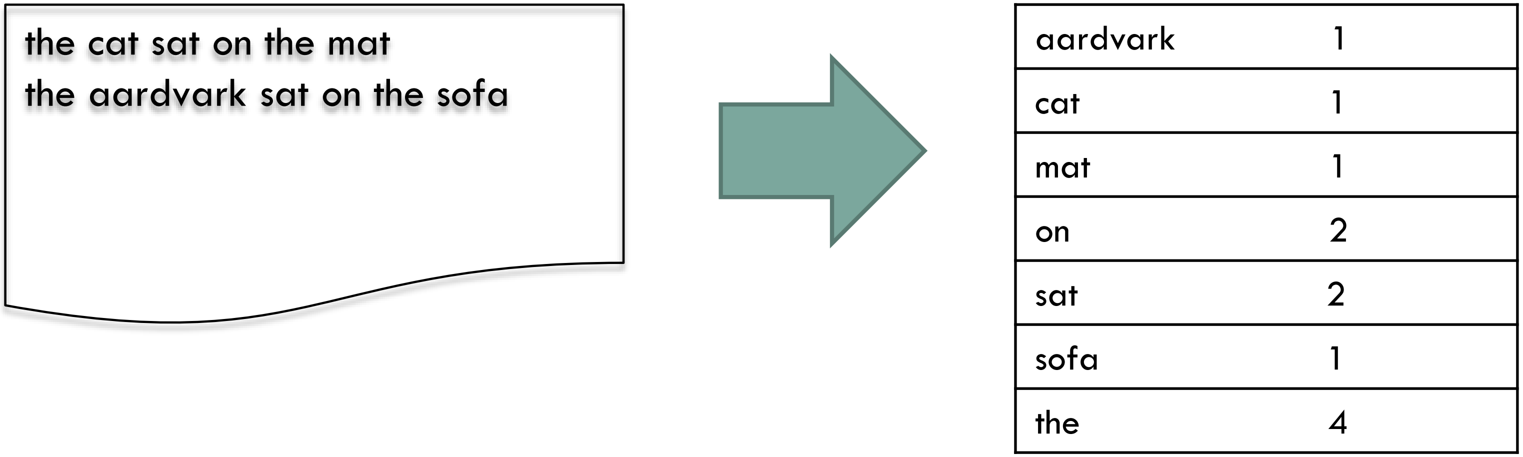
```
u'USER_79321756', (u'2010-03-
03T04:55:32', u'\xdcT: 47.528139,-
122.197916')
```

Some Key-Value Operations

```
pets = sc.parallelize([(“cat”, 1), (“dog”, 1), (“cat”, 2)])
pets.reduceByKey(lambda x, y: x + y) # => {(cat, 3), (dog, 1)}
pets.groupByKey() # => {(cat, [1, 2]), (dog, [1])}
pets.sortByKey()  # => {(cat, 1), (cat, 2), (dog, 1)}
```

reduceByKey also automatically implements combiners on the map side

Spark RDD: WordCount Example



the cat sat on the mat
the aardvark sat on the sofa

aardvark	1
cat	1
mat	1
on	2
sat	2
sofa	1
the	4

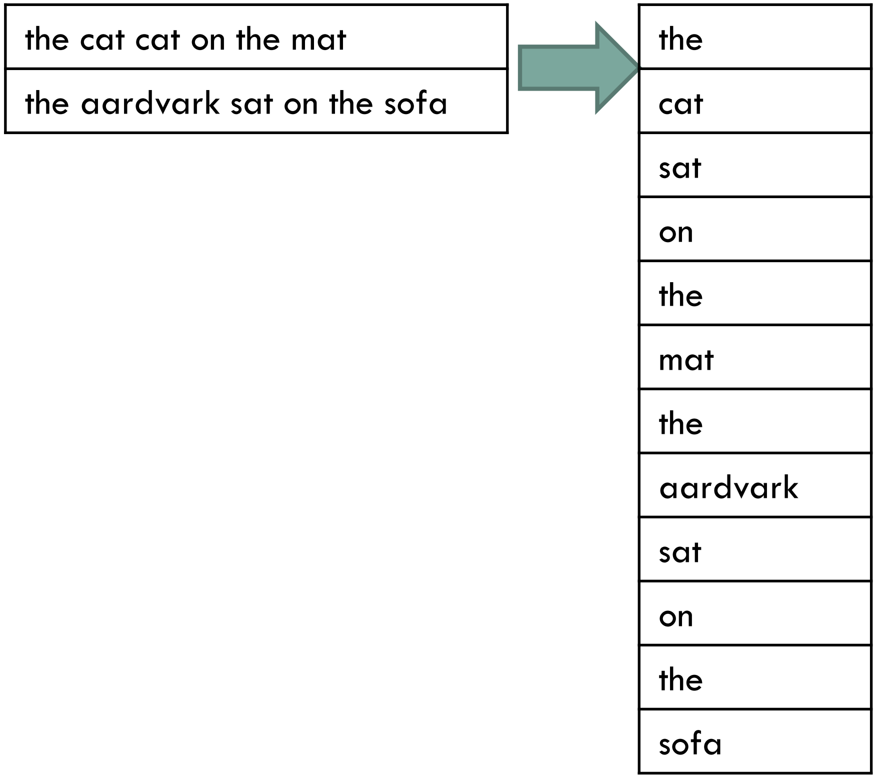
Spark RDD: WordCount Example

- counts = sc.textFile(“shakespeare_100.txt”)

the cat cat on the mat
the aardvark sat on the sofa

Spark RDD: WordCount Example

```
counts = sc.textFile(text)
           .flatMap(lambda line: line.split() )
```



the cat cat on the mat

the aardvark sat on the sofa

the

cat

sat

on

the

mat

the

aardvark

sat

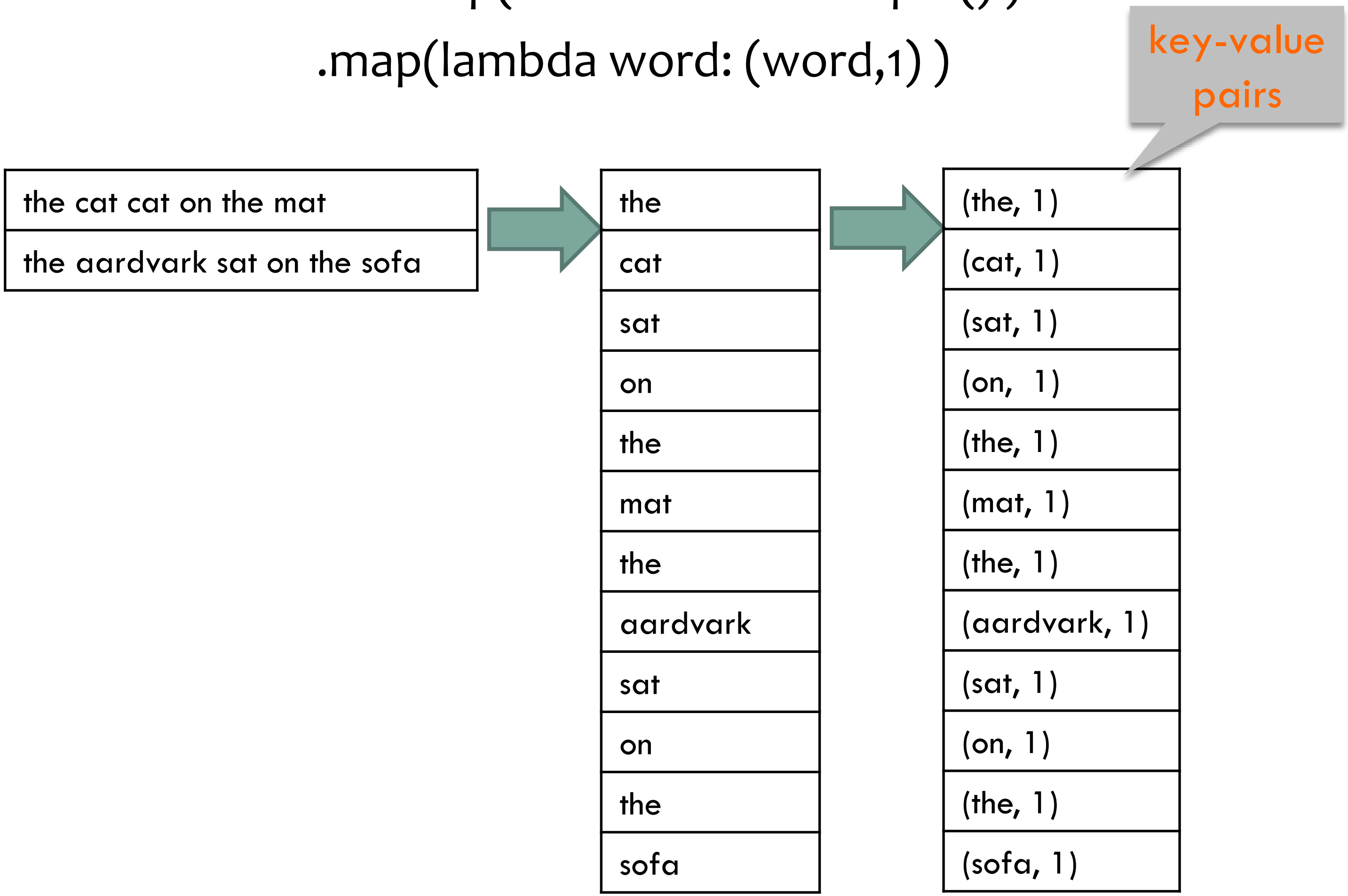
on

the

sofa

Spark RDD: WordCount Example

-
  ```
  counts = sc.textFile(text)
          .flatMap(lambda line: line.split() )
          .map(lambda word: (word,1) )
  ```



key-value pairs

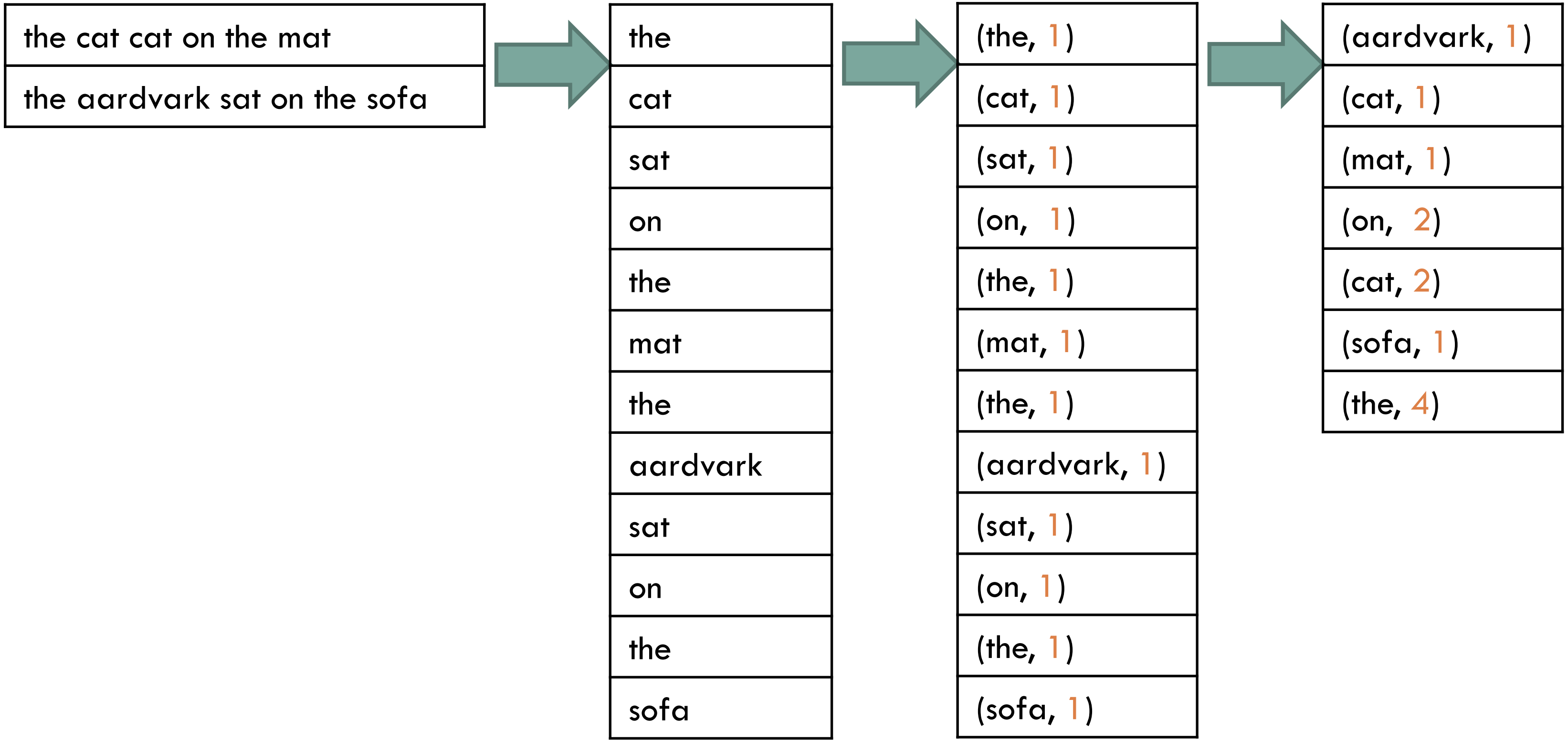
the cat cat on the mat
the aardvark sat on the sofa

the
cat
sat
on
the
mat
the
aardvark
sat
on
the
sofa

(the, 1)
(cat, 1)
(sat, 1)
(on, 1)
(the, 1)
(mat, 1)
(the, 1)
(aardvark, 1)
(sat, 1)
(on, 1)
(the, 1)
(sofa, 1)

Spark RDD: WordCount Example

```
counts = sc.textFile(text)
          .flatMap(lambda line: line.split() )
          .map(lambda word: (word,1) )
          .reduceByKey(lambda v1,v2: v1+v2)
```



the cat cat on the mat
the aardvark sat on the sofa

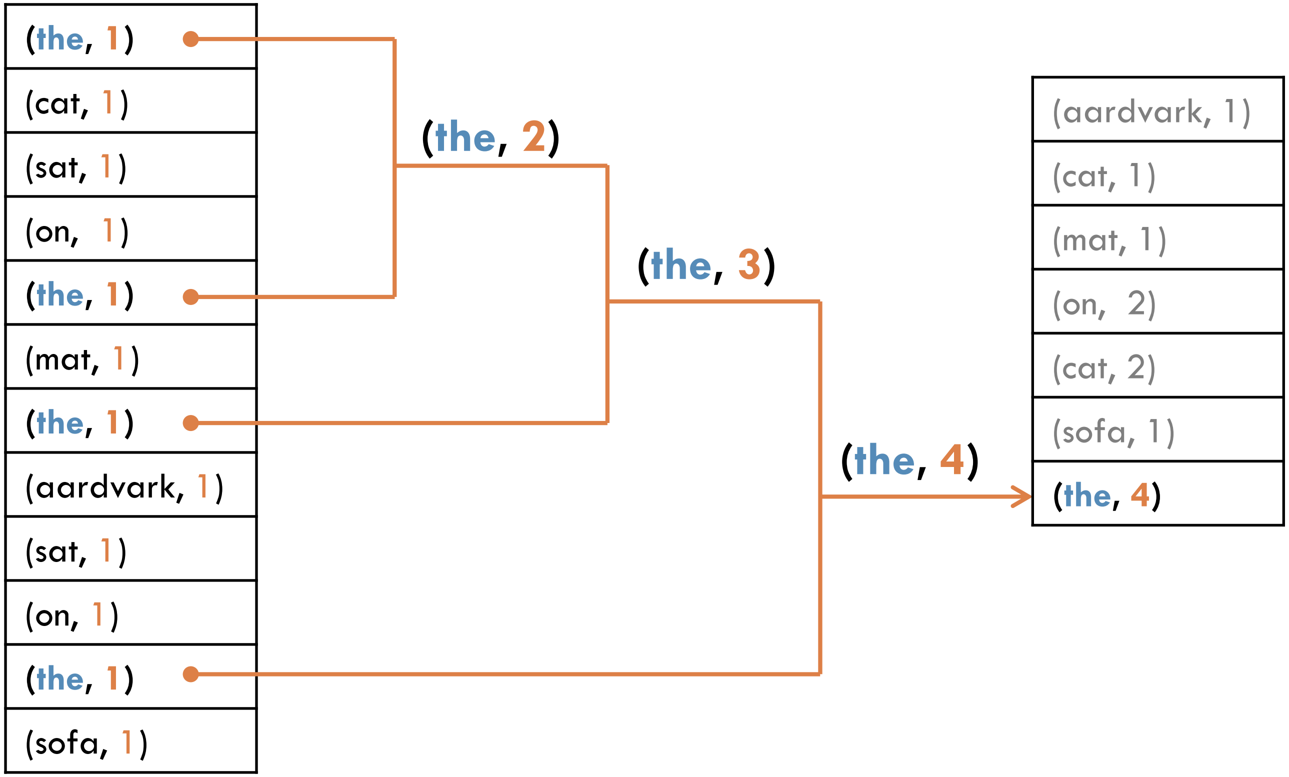
the
cat
sat
on
the
mat
the
aardvark
sat
on
the
sofa

(the, 1)
(cat, 1)
(sat, 1)
(on, 1)
(the, 1)
(mat, 1)
(the, 1)
(aardvark, 1)
(sat, 1)
(on, 1)
(the, 1)
(sofa, 1)

(aardvark, 1)
(cat, 1)
(mat, 1)
(on, 2)
(cat, 2)
(sofa, 1)
(the, 4)

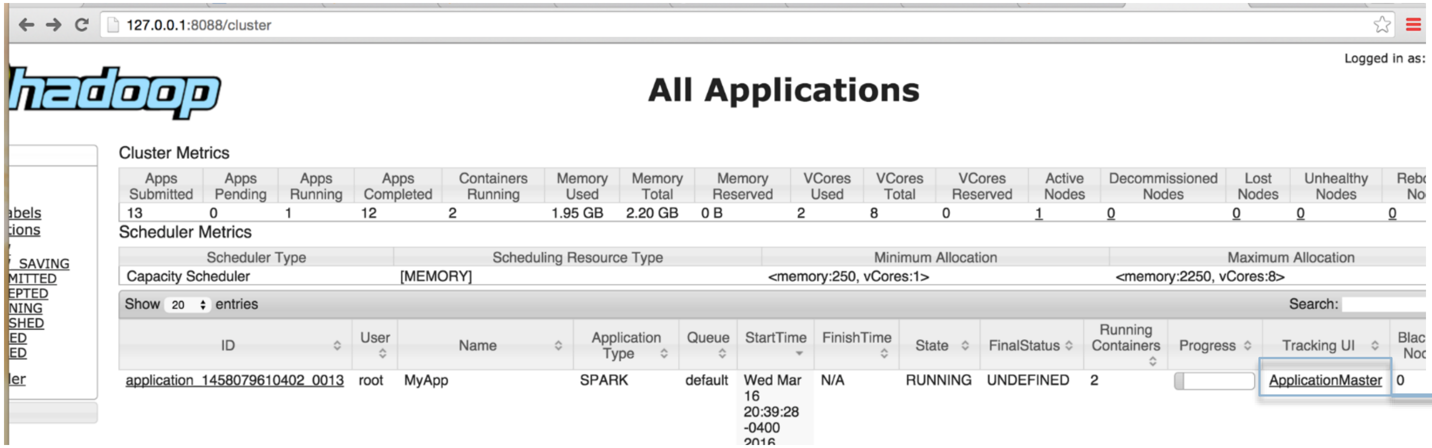
Spark RDD: WordCount Example

- ReduceByKey functions must be
  - Binary – combines values from two keys
  - Commutative → x+y = y+x
  - Associative → (x+y)+z = x+(y+z)



Monitor Status of Spark Job

- All jobs run on the VM can be monitored using http://127.0.0.1:8088/cluster



127.0.0.1:8088/cluster

Logged in as:

hadoop

All Applications

abels
tions
SAVING
MITTED
EPTED
NING
SHED
ED
ED
ler

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebo No
13	0	1	12	2	1.95 GB	2.20 GB	0 B	2	8	0	1	0	0	0	0

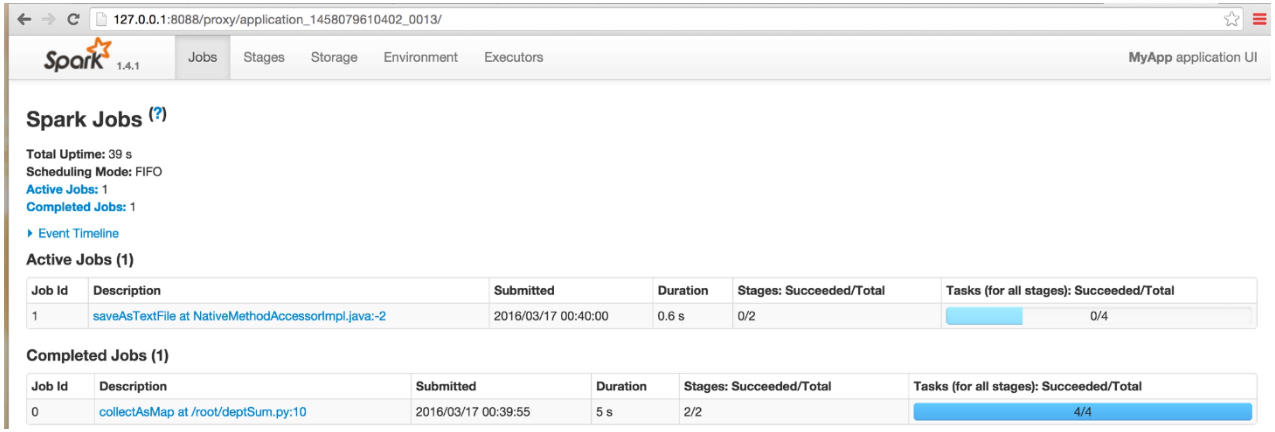
Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:250, vCores:1>	<memory:2250, vCores:8>

Show 20 entries Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Running Containers	Progress	Tracking UI	Blac Noc
application_1458079610402_0013	root	MyApp	SPARK	default	Wed Mar 16 20:39:28 -0400 2016	N/A	RUNNING	UNDEFINED	2		ApplicationMaster	0

Click to get to the page below



127.0.0.1:8088/proxy/application_1458079610402_0013/

Spark Jobs (?)

Total Uptime: 39 s
Scheduling Mode: FIFO
Active Jobs: 1
Completed Jobs: 1

▸ Event Timeline

Active Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1	saveAsTextFile at NativeMethodAccessorImpl.java:-2	2016/03/17 00:40:00	0.6 s	0/2	0/4

Completed Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	collectAsMap at /root/deptSum.py:10	2016/03/17 00:39:55	5 s	2/2	4/4

Getting Started with Spark

- Install Spark Standalone on Linux
- Cloudera/HDP Distributions
  - http://www.cloudera.com/content/cloudera/en/downloads/quickstart_vms/cdh-5-3-x.html

Spark Documentation

Spark 1.0.0 Documentation: http://spark.apache.org/docs/latest/
Spark Programming Guide (Scala, Python): http://spark.apache.org/docs/latest/programming-guide.html#overview
Spark Cassandra Connector - DataStax (github)
Spark HBase - lighting Spark with HBase (link)
Spark MLlib Documentation (link)
Spark GraphX Documentation (link)
(Spark) Spark Cluster Mode Overview (link)
(Spark) Running Spark on EC2 (link)
(Cloudera) Pig is Flying: Apache Pig on Apache Spark (link)