CHICAGO
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE 2015

goto; conference

Intro to Apache Spark

Paco Nathan

follow us @gotochgo

Conference: May 11-13 / Workshops: 13-14

Intro to Apache Spark

GOTO Chicago
2015-05-14

Lecture Outline:

- login/quick start on **Databricks Cloud**
- Spark theory of operation in a cluster
- a brief history, context for Apache Spark
- how to "think notebooks"
- progressive coding exercises
- overview of SQL, Streaming, MLlib, GraphX, DataFrames
- case studies
- demos – (as time permits)
- lots of Q&A!
- resources: certification, events, community, etc.

Welcome + Getting Started

databricks

Getting Started: *Step 1*

Everyone will receive a username/password for one of the Databricks Cloud *shards*. Use your laptop and browser to login there:

- **https://class01.cloud.databricks.com/**

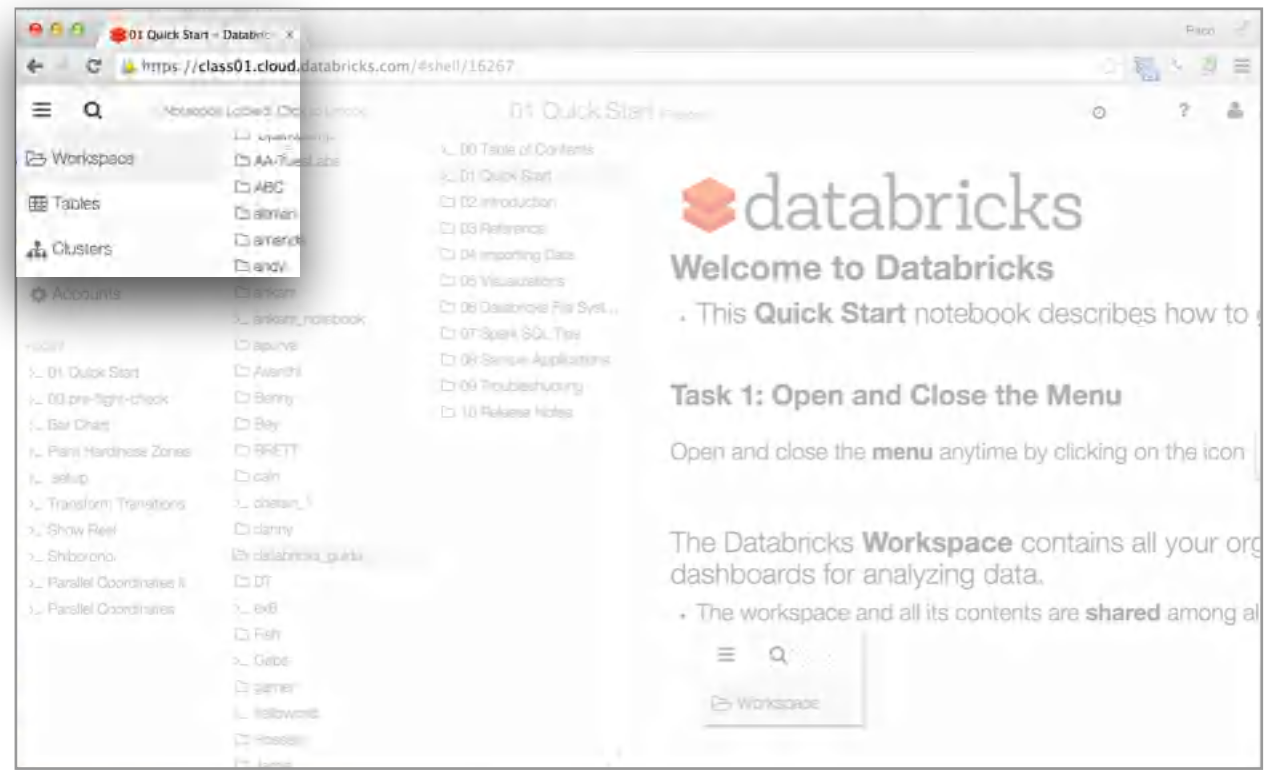
user: **NN@QConSP.com**
pass: **NN@QConSP.com**

We find that cloud-based notebooks are a simple way to get started using **Apache Spark** – as the motto "Making Big Data Simple" states.

Please create and run a variety of notebooks on your account throughout the tutorial. These accounts will remain open long enough for you to export your work.

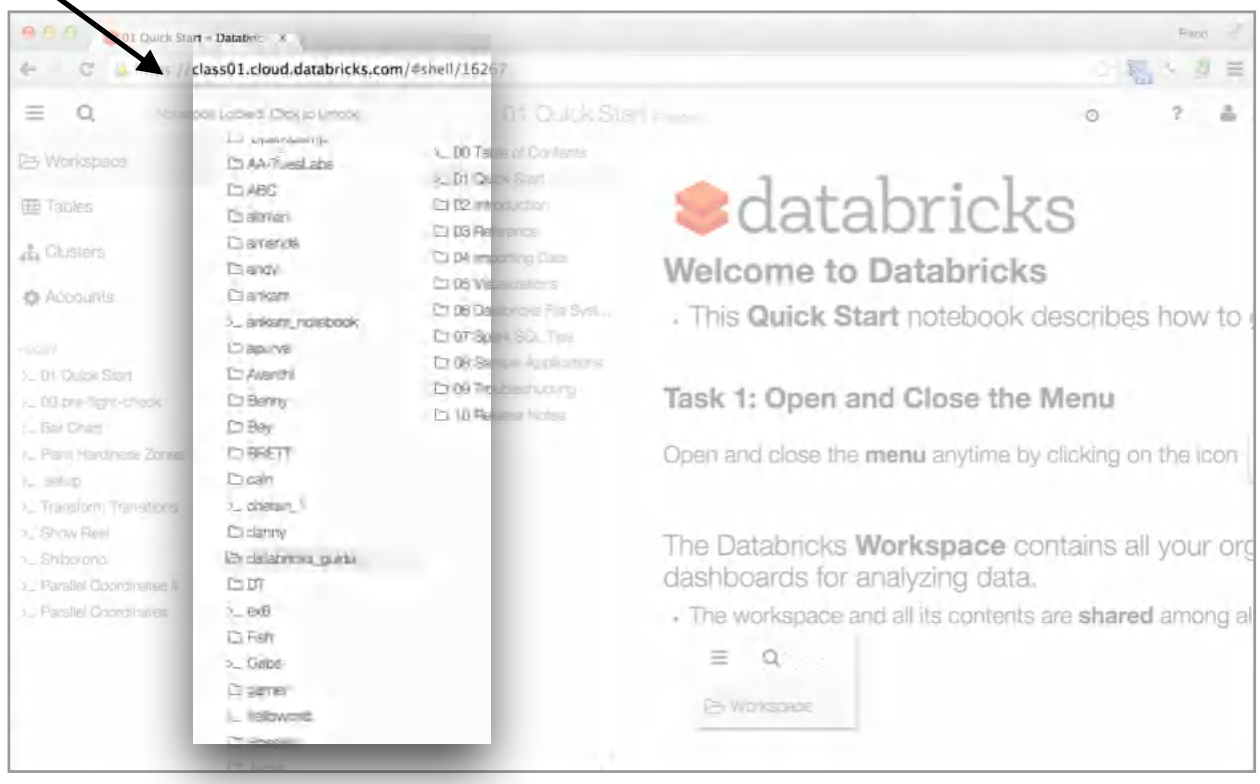
Getting Started: *Step 2*

Open in a browser window, then click on the *navigation* menu in the top/left corner:



Getting Started: *Step 3*

The next columns to the right show *folders*,
and scroll down to click on `databricks_guide`



Getting Started: *Step 4*

Scroll to open the `01 Quick Start` notebook, then follow the discussion about using key features:

Getting Started: *Step 5*

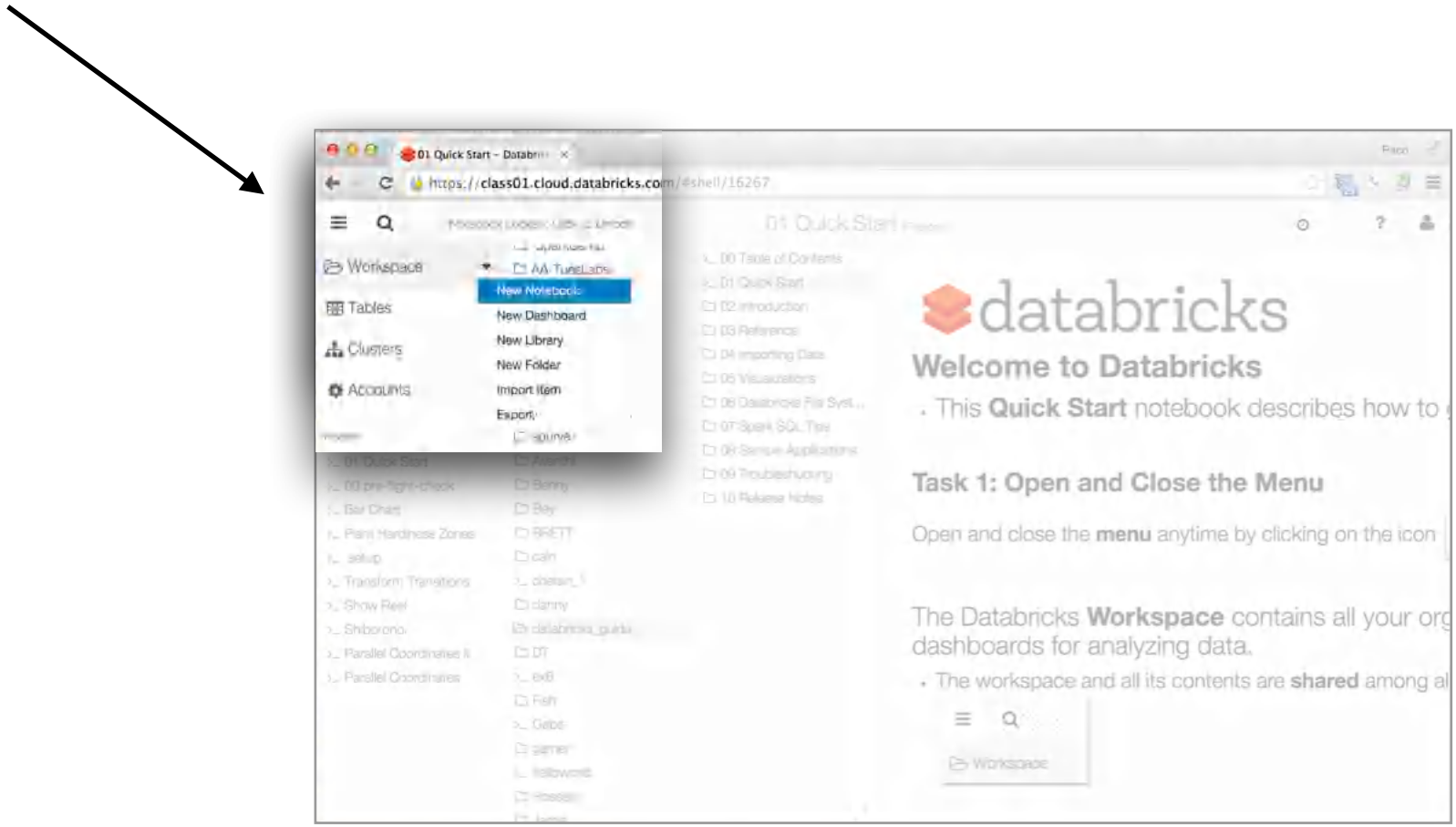
See `/databricks-guide/01 Quick Start`

Key Features:

- Workspace / Folder / Notebook
- Code Cells, run/edit/move/comment
- **Markdown**
- Results
- Import/Export

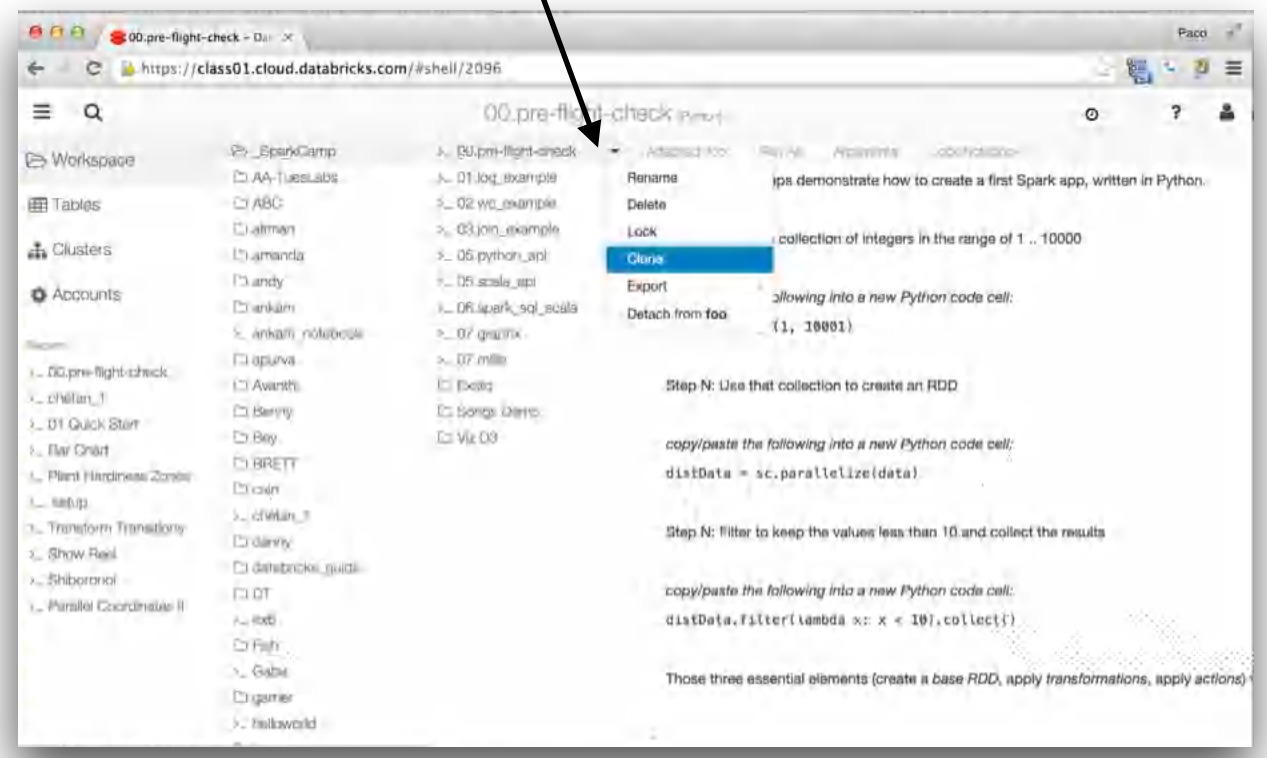
Getting Started: *Step 6*

Click on the *Workspace* menu and create your own folder (pick a name):



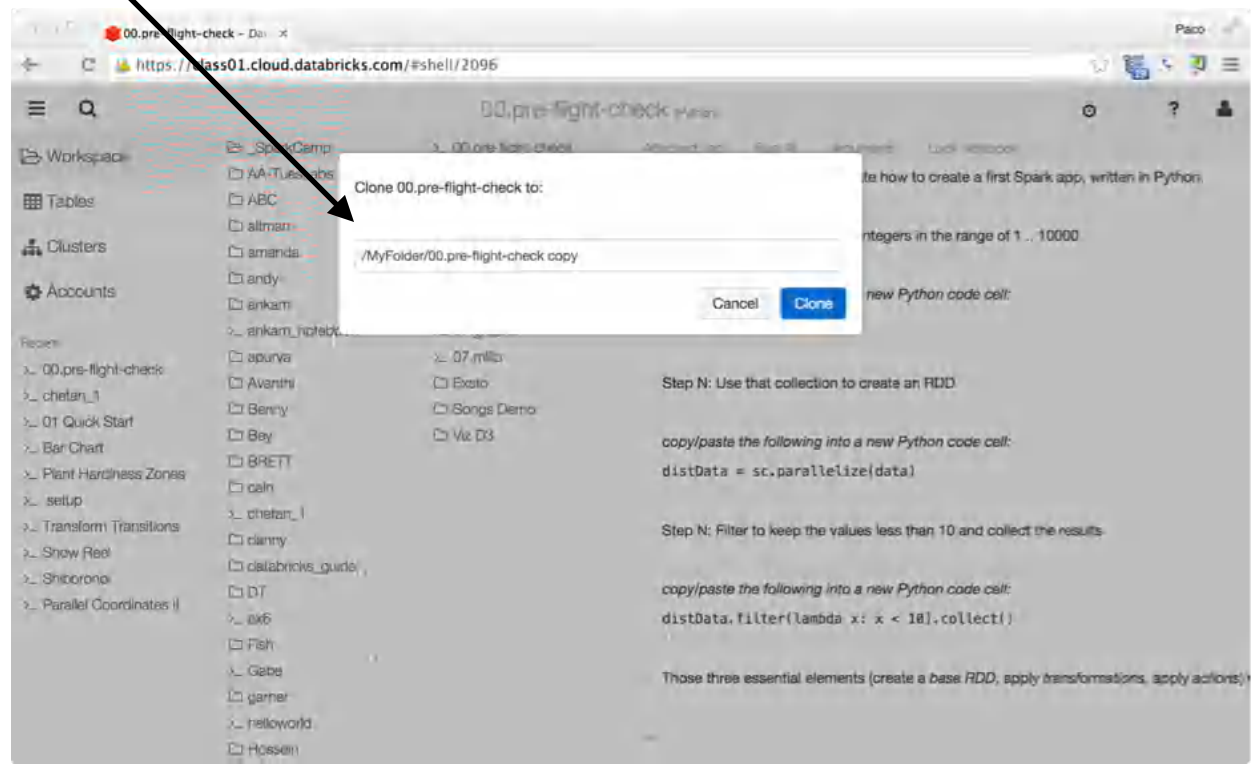
Getting Started: *Step 7*

Navigate to `/_SparkCamp/00.pre-flight-check`
hover on its drop-down menu, on the right side:



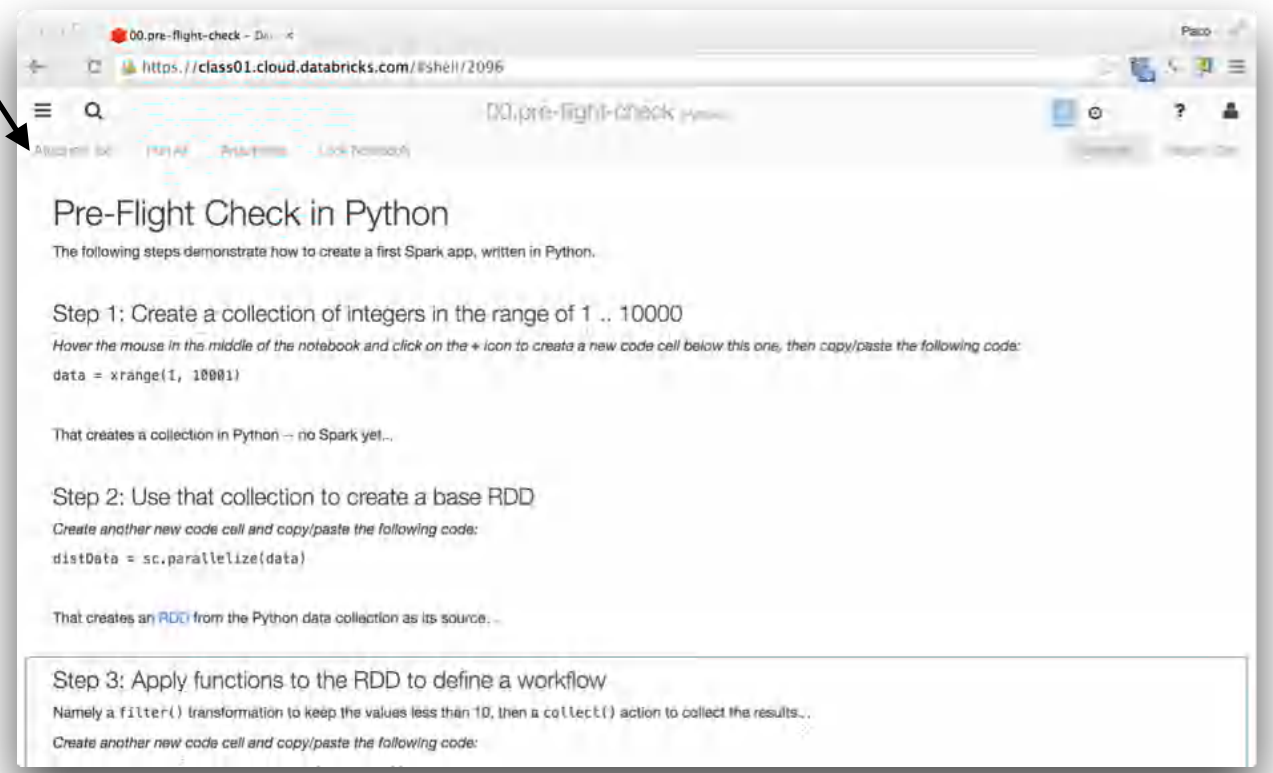
Getting Started: *Step 8*

Then create a *clone* of this notebook in the folder that you just created:



Getting Started: *Step 9*

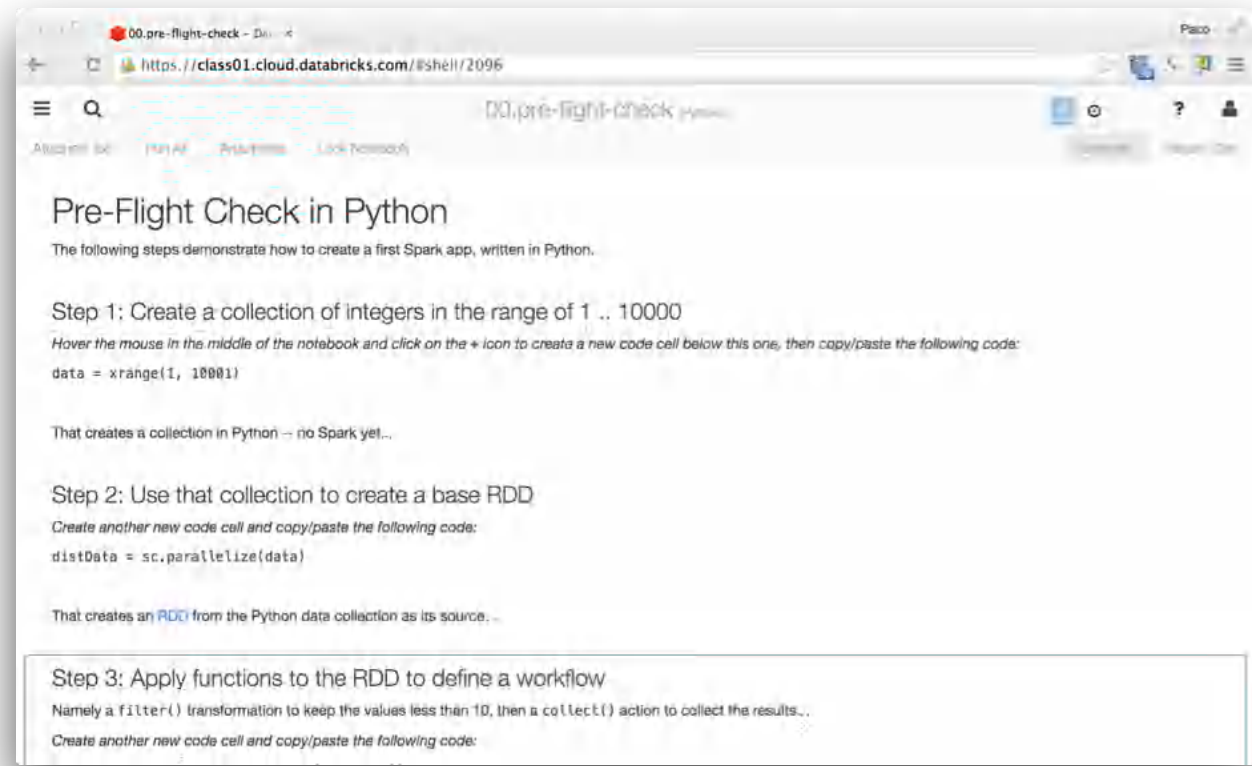
Attach your *cluster* – same as your *username*:



Getting Started: *Coding Exercise*

Now let's get started with the coding exercise!

We'll define an initial Spark app in three lines of code:



Getting Started: *Bonus!*

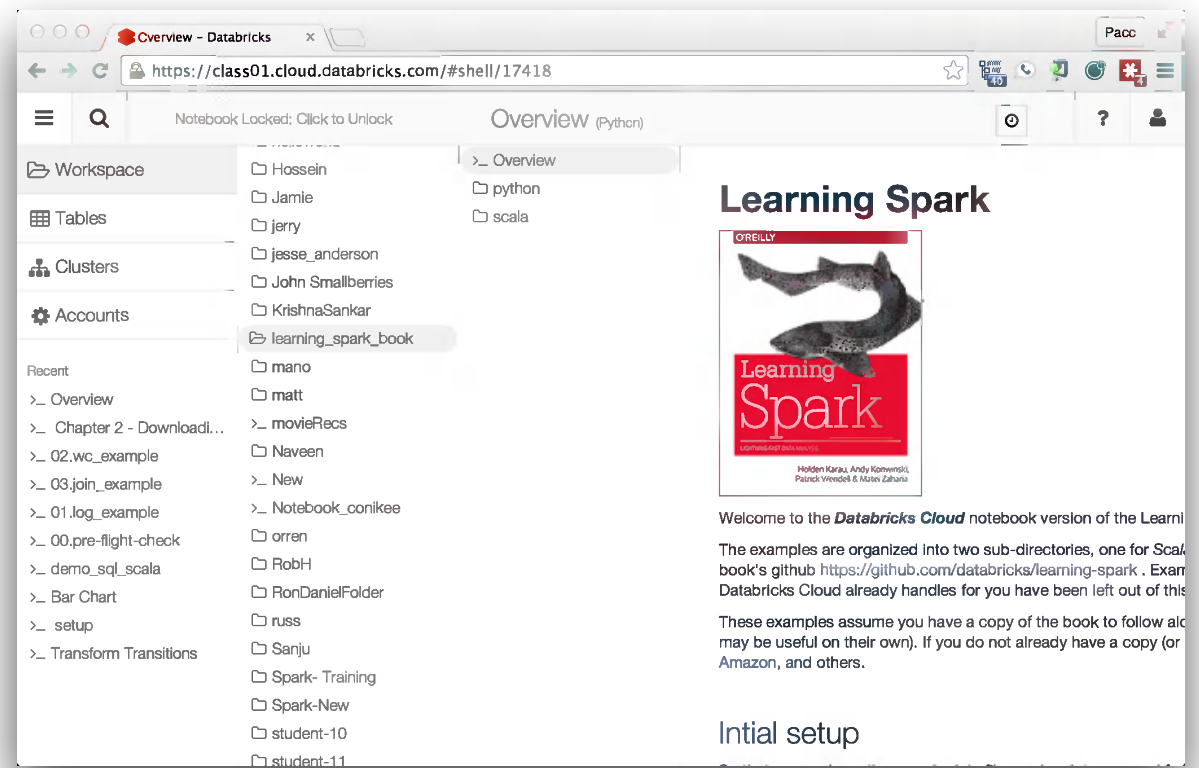
If you're new to this **Scala** thing and want to spend a few minutes on the basics…

Scala Crash Course
Holden Karau
lintool.github.io/SparkTutorial/slides/day1_Scala_crash_course.pdf

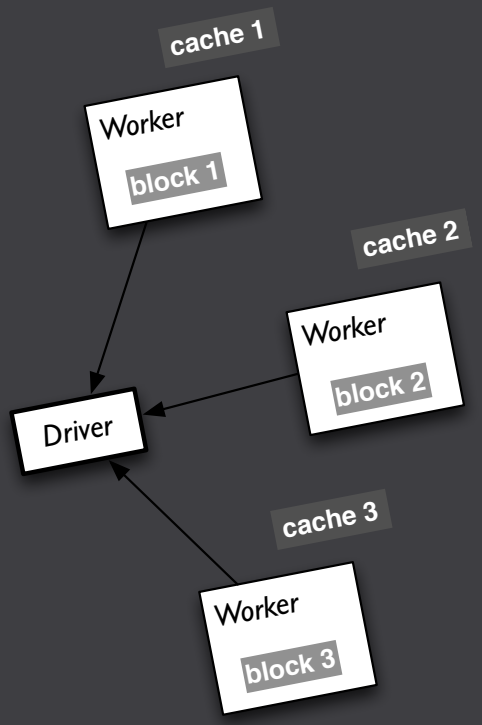
Getting Started: *Extra Bonus!!*

See also the `/learning_spark_book`

for all of its code examples in notebooks:

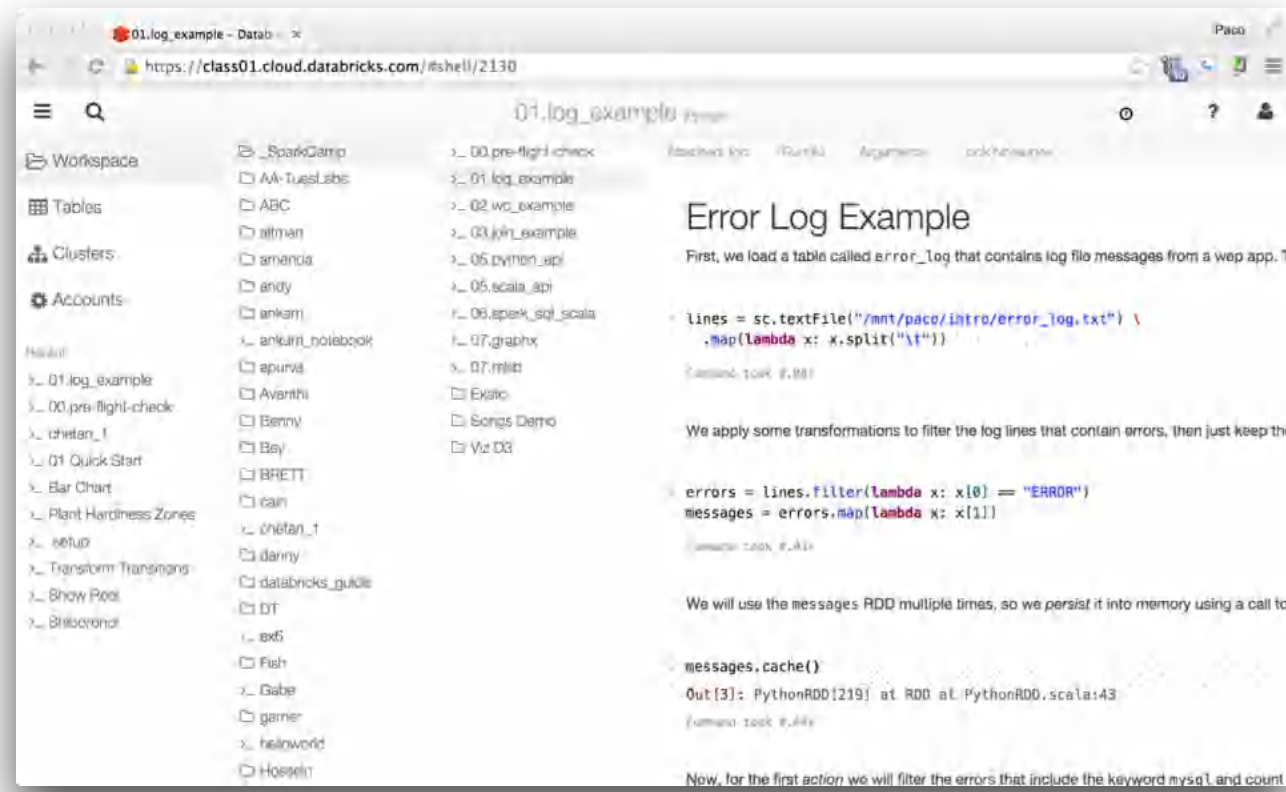


How Spark runs on a Cluster



Spark Deconstructed: *Log Mining Example*

Clone and run `/_SparkCamp/01.log_example` in your folder:



Spark Deconstructed: *Log Mining Example*

```
# load error messages from a log into memory
# then interactively search for patterns

# base RDD
lines = sc.textFile("/mnt/paco/intro/error_log.txt") \
 .map(lambda x: x.split("\t"))

# transformed RDDs
errors = lines.filter(lambda x: x[0] == "ERROR")
messages = errors.map(lambda x: x[1])

# persistence
messages.cache()

# action 1
messages.filter(lambda x: x.find("mysql") > -1).count()

# action 2
messages.filter(lambda x: x.find("php") > -1).count()
```

Spark Deconstructed: *Log Mining Example*

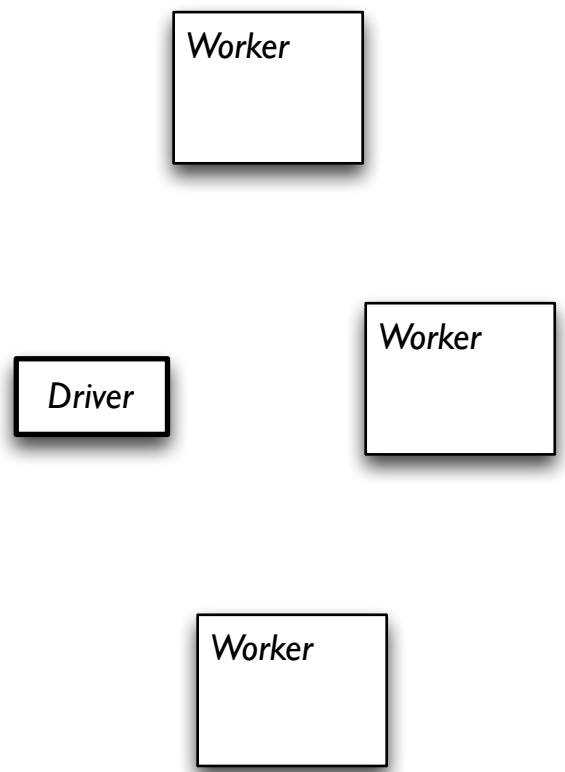
Note that we can examine the *operator graph* for a transformed RDD, for example:

```
x = messages.filter(lambda x: x.find("mysql") > -1)
print(x.toDebugString())
```

```
(2) PythonRDD[772] at RDD at PythonRDD.scala:43 []
 |  PythonRDD[219] at RDD at PythonRDD.scala:43 []
 |  error_log.txt MappedRDD[218] at NativeMethodAccessorImpl.java:-2 []
 |  error_log.txt HadoopRDD[217] at NativeMethodAccessorImpl.java:-2 []
```

Spark Deconstructed: *Log Mining Example*

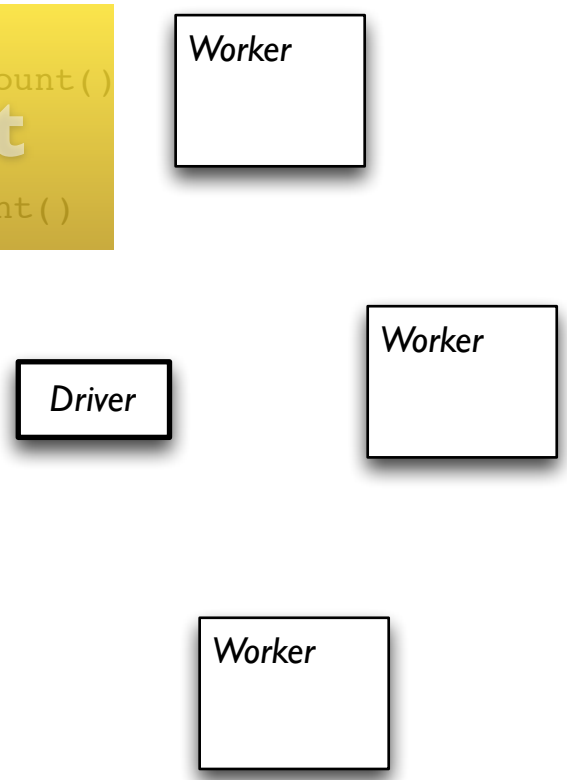
We start with Spark running on a cluster…
submitting code to be evaluated on it:



Spark Deconstructed: *Log Mining Example*

```
# base RDD
lines = sc.textFile("/mnt/paco/intro/error_log.txt") \
 .map(lambda x: x.split("\t"))

# transformed RDDs
errors = lines.filter(lambda x: x[0] == "ERROR")
messages = errors.map(lambda x: x[1])

# persistence
messages.cache()

# action 1
messages.filter(lambda x: x.find("mysql") > -1).count()

# action 2
messages.filter(lambda x: x.find("php") > -1).count()
```

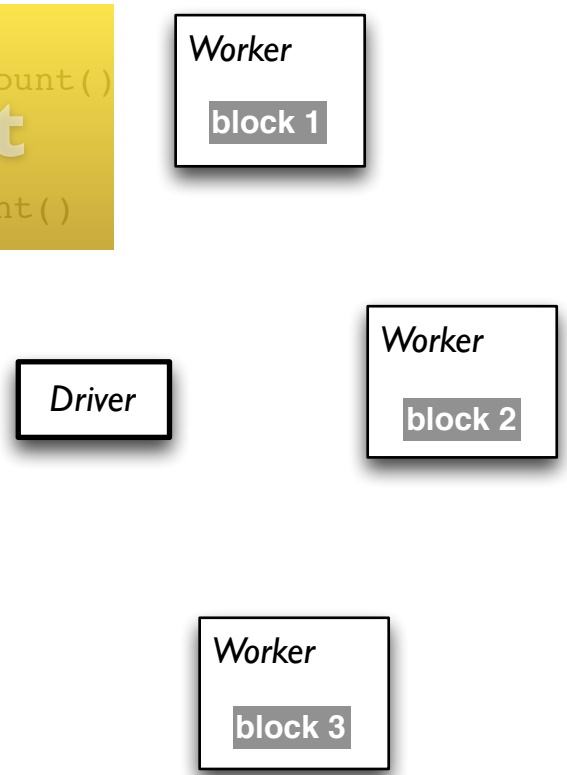
discussing the other part



Spark Deconstructed: *Log Mining Example*

```
# base RDD
lines = sc.textFile("/mnt/paco/intro/error_log.txt") \
 .map(lambda x: x.split("\t"))

# transformed RDDs
errors = lines.filter(lambda x: x[0] == "ERROR")
messages = errors.map(lambda x: x[1])

# persistence
messages.cache()

# action 1
messages.filter(lambda x: x.find("mysql") > -1).count()

# action 2
messages.filter(lambda x: x.find("php") > -1).count()
```

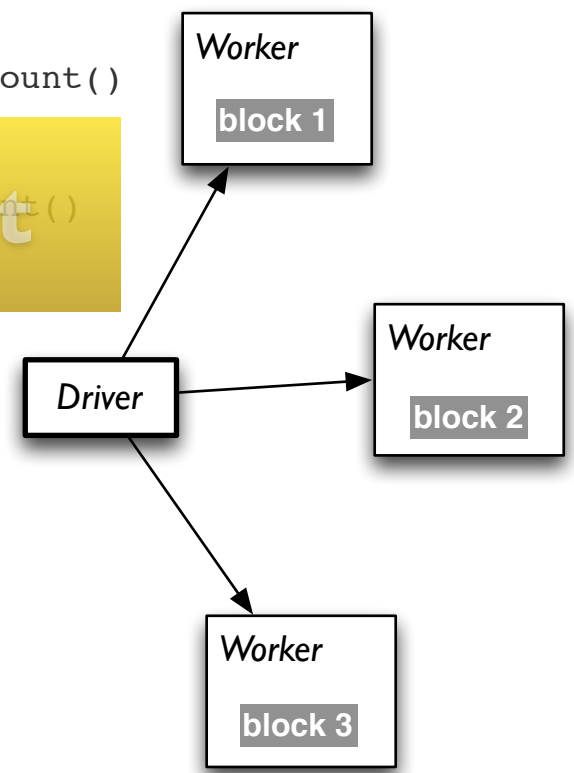
discussing the other part



Spark Deconstructed: *Log Mining Example*

```
# base RDD
lines = sc.textFile("/mnt/paco/intro/error_log.txt") \
 .map(lambda x: x.split("\t"))

# transformed RDDs
errors = lines.filter(lambda x: x[0] == "ERROR")
messages = errors.map(lambda x: x[1])

# persistence
messages.cache()

# action 1
messages.filter(lambda x: x.find("mysql") > -1).count()

# action 2
messages.filter(lambda x: x.find("php") > -1).count()
```

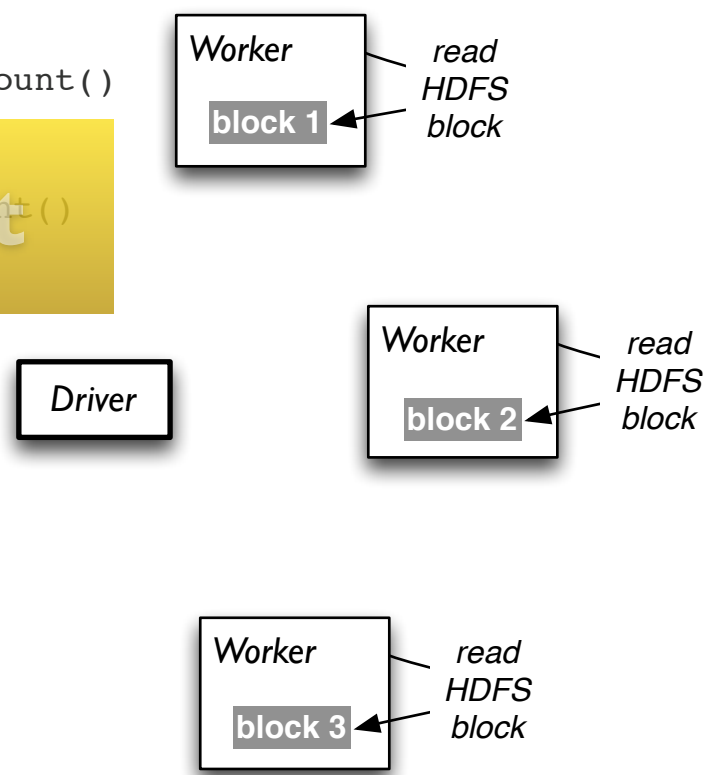
discussing the other part



Spark Deconstructed: *Log Mining Example*

```
# base RDD
lines = sc.textFile("/mnt/paco/intro/error_log.txt") \
 .map(lambda x: x.split("\t"))

# transformed RDDs
errors = lines.filter(lambda x: x[0] == "ERROR")
messages = errors.map(lambda x: x[1])

# persistence
messages.cache()

# action 1
messages.filter(lambda x: x.find("mysql") > -1).count()

# action 2
messages.filter(lambda x: x.find("php") > -1).count()
```

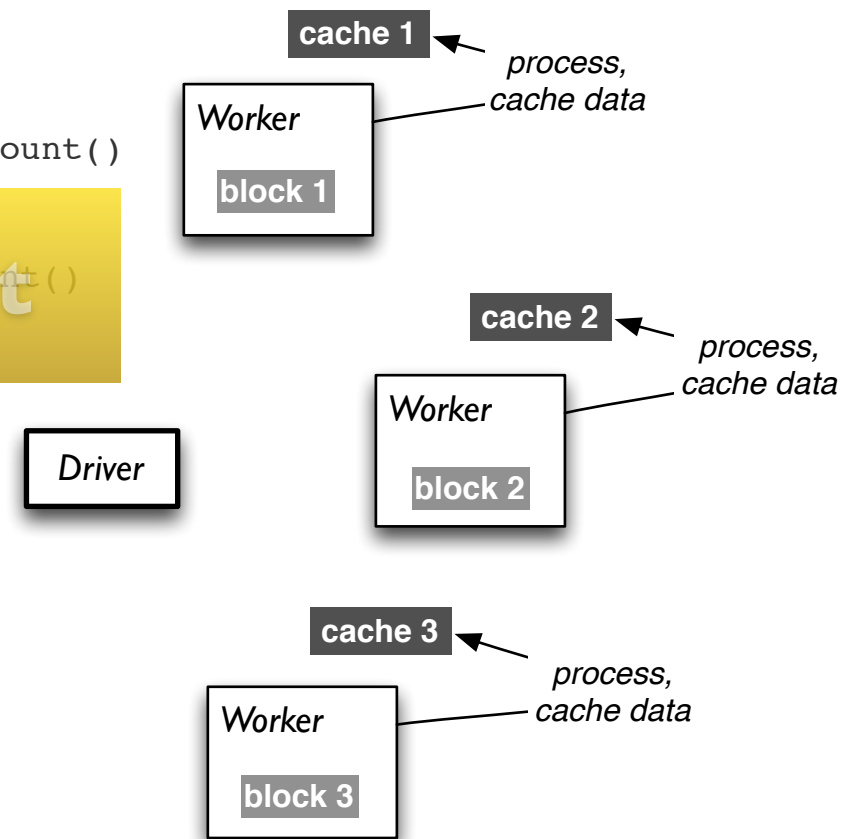
discussing the other part



Spark Deconstructed: *Log Mining Example*

```
# base RDD
lines = sc.textFile("/mnt/paco/intro/error_log.txt") \
 .map(lambda x: x.split("\t"))

# transformed RDDs
errors = lines.filter(lambda x: x[0] == "ERROR")
messages = errors.map(lambda x: x[1])

# persistence
messages.cache()

# action 1
messages.filter(lambda x: x.find("mysql") > -1).count()

# action 2
messages.filter(lambda x: x.find("php") > -1).count()
```

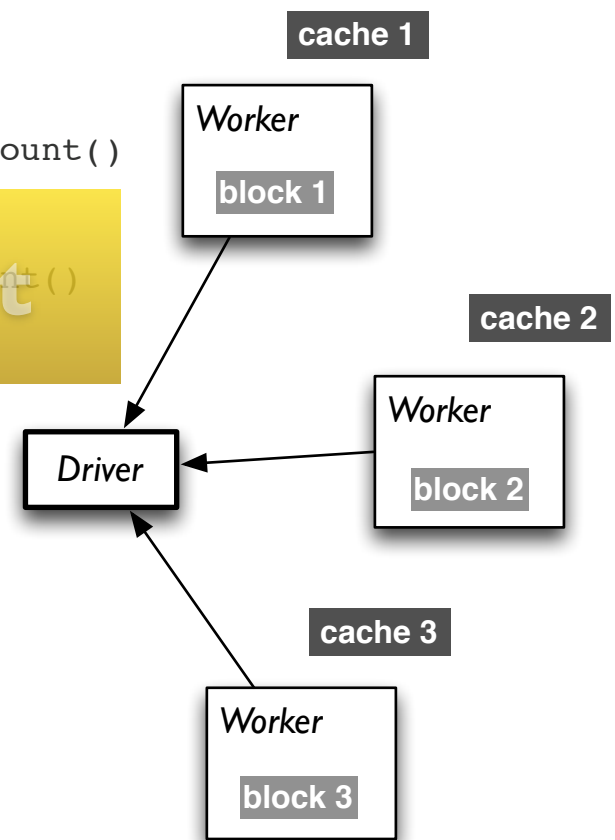
discussing the other part



Spark Deconstructed: *Log Mining Example*

```
# base RDD
lines = sc.textFile("/mnt/paco/intro/error_log.txt") \
 .map(lambda x: x.split("\t"))

# transformed RDDs
errors = lines.filter(lambda x: x[0] == "ERROR")
messages = errors.map(lambda x: x[1])

# persistence
messages.cache()

# action 1
messages.filter(lambda x: x.find("mysql") > -1).count()

# action 2
messages.filter(lambda x: x.find("php") > -1).count()
```

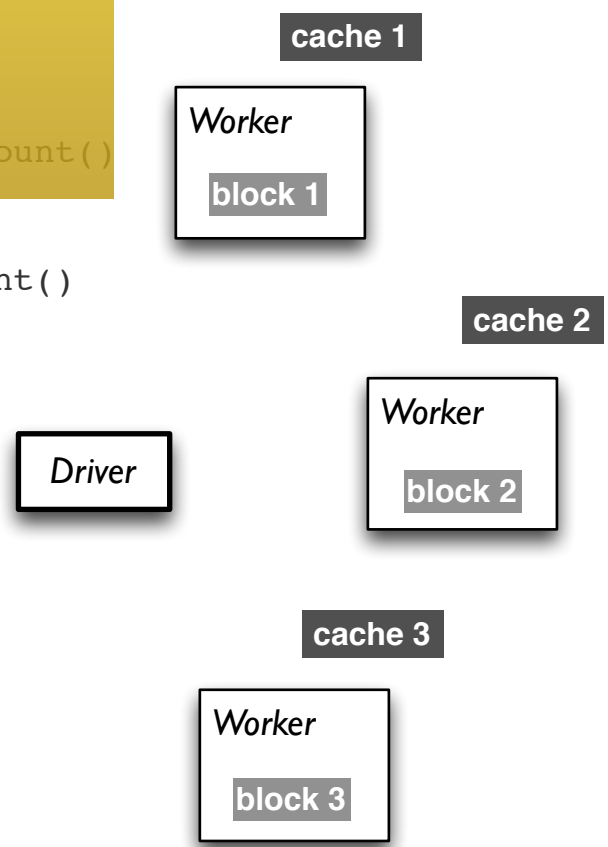
discussing the other part



Spark Deconstructed: *Log Mining Example*

```
# base RDD
lines = sc.textFile("/mnt/paco/intro/error_log.txt") \
  .map(lambda x: x.split("\t"))

# transformed RDDs
errors = lines.filter(lambda x: x[0] == "ERROR")
messages = errors.map(lambda x: x[1])

# persistence
messages.cache()

# action 1
messages.filter(lambda x: x.find("mysql") > -1).count()

# action 2
messages.filter(lambda x: x.find("php") > -1).count()
```

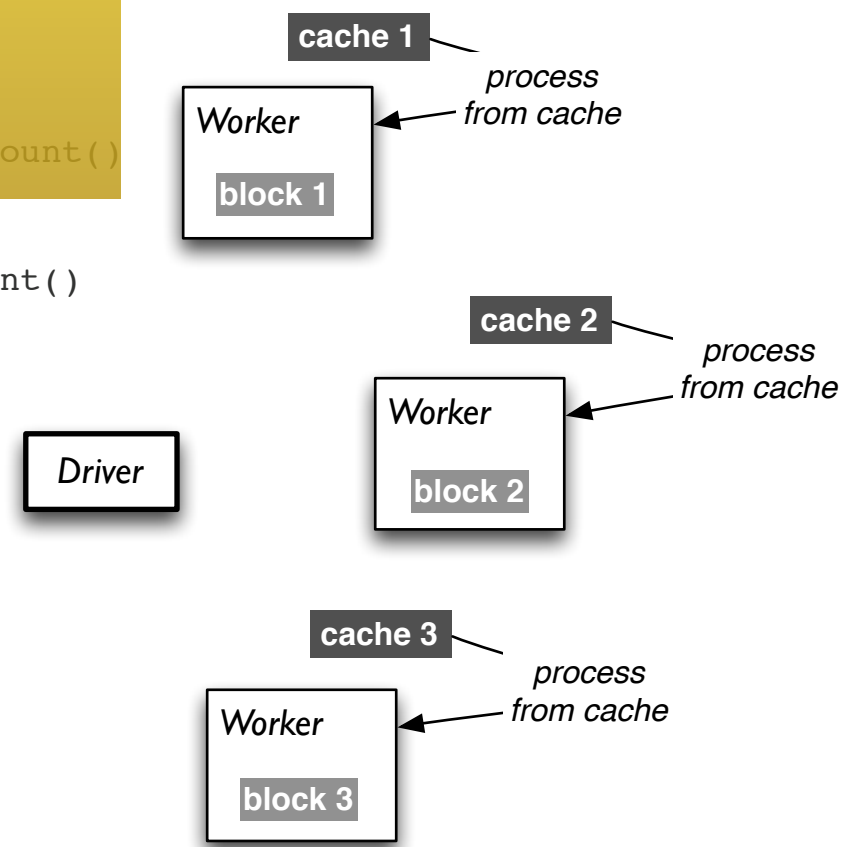
discussing the other part



Spark Deconstructed: *Log Mining Example*

```
# base RDD
lines = sc.textFile("/mnt/paco/intro/error_log.txt") \
  .map(lambda x: x.split("\t"))

# transformed RDDs
errors = lines.filter(lambda x: x[0] == "ERROR")
messages = errors.map(lambda x: x[1])

# persistence
messages.cache()

# action 1
messages.filter(lambda x: x.find("mysql") > -1).count()

# action 2
messages.filter(lambda x: x.find("php") > -1).count()
```

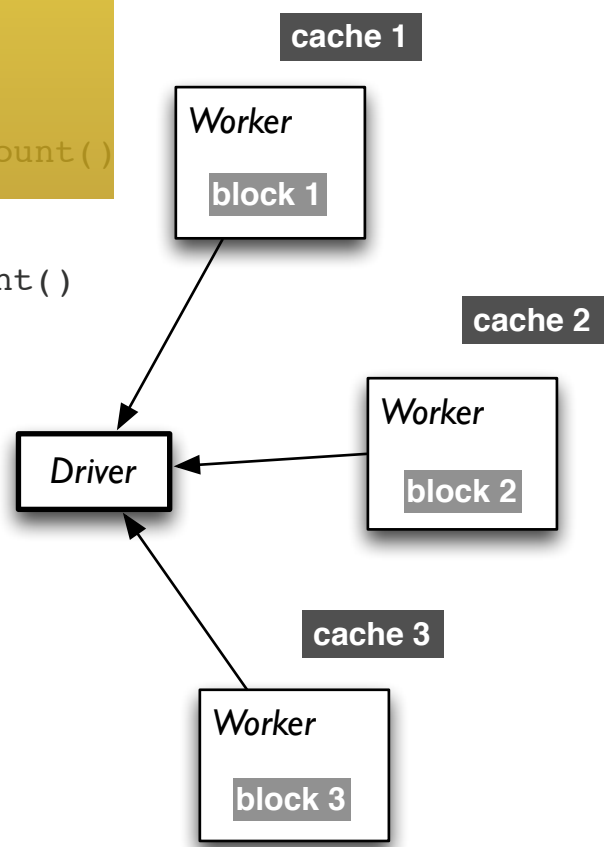
discussing the other part



Spark Deconstructed: *Log Mining Example*

```
# base RDD
lines = sc.textFile("/mnt/paco/intro/error_log.txt") \
  .map(lambda x: x.split("\t"))

# transformed RDDs
errors = lines.filter(lambda x: x[0] == "ERROR")
messages = errors.map(lambda x: x[1])

# persistence
messages.cache()

# action 1
messages.filter(lambda x: x.find("mysql") > -1).count()

# action 2
messages.filter(lambda x: x.find("php") > -1).count()
```

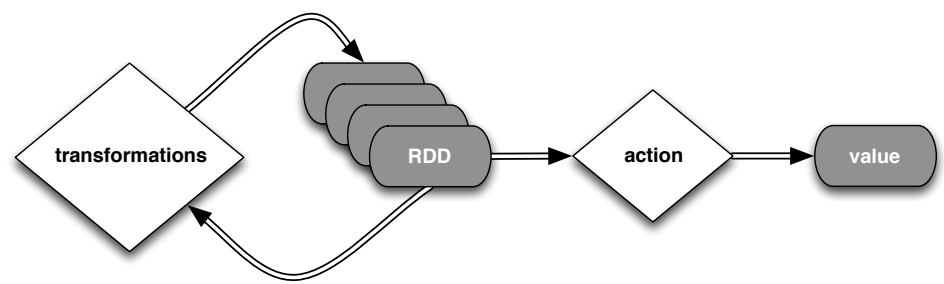
discussing the other part



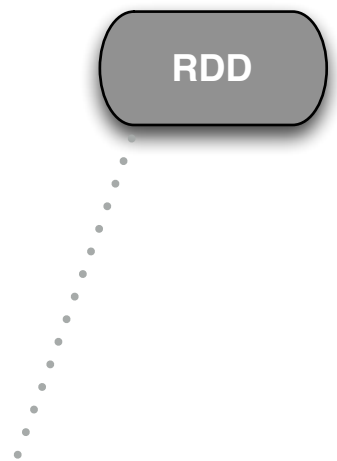
Spark Deconstructed: *Log Mining Example*

Looking at the RDD transformations and actions from another perspective…

```
# base RDD
lines = sc.textFile("/mnt/paco/intro/error_log.txt") \
 .map(lambda x: x.split("\t"))

# transformed RDDs
errors = lines.filter(lambda x: x[0] == "ERROR")
messages = errors.map(lambda x: x[1])

# persistence
messages.cache()

# action 1
messages.filter(lambda x: x.find("mysql") > -1).count()

# action 2
messages.filter(lambda x: x.find("php") > -1).count()
```

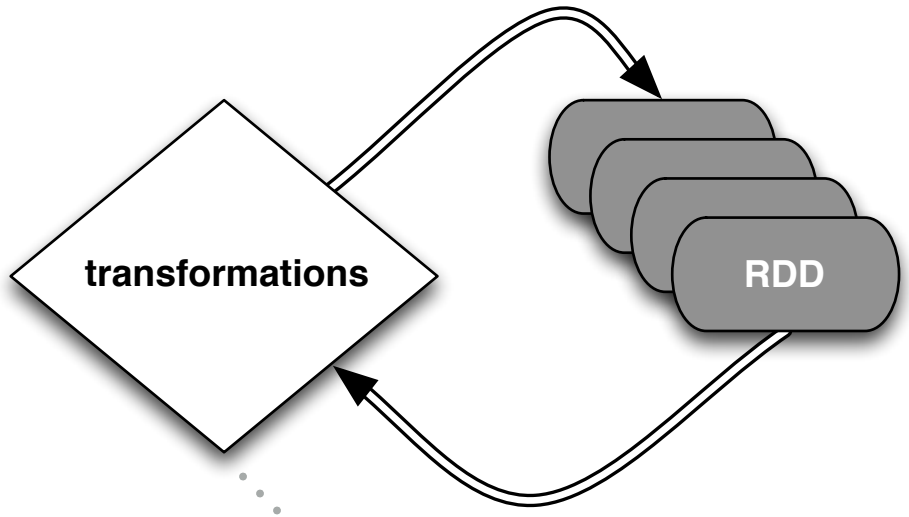


Spark Deconstructed: *Log Mining Example*



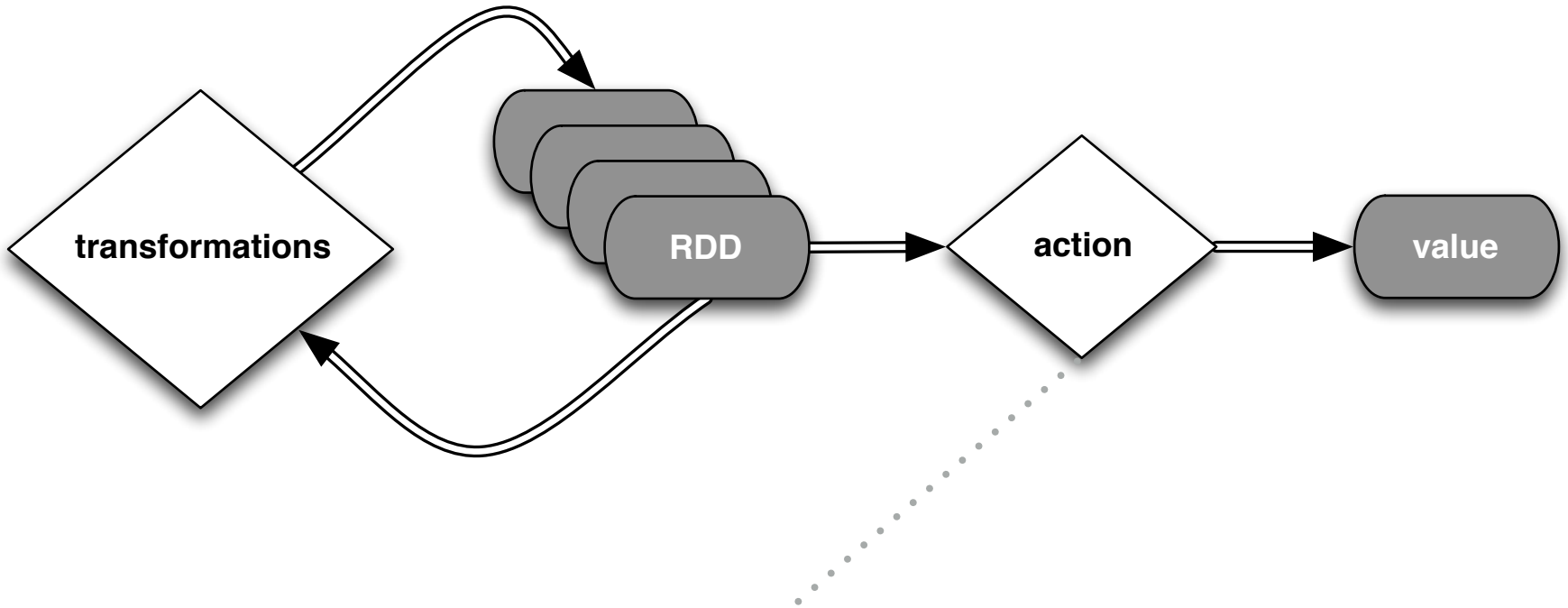

```
# base RDD
lines = sc.textFile("/mnt/paco/intro/error_log.txt") \
 .map(lambda x: x.split("\t"))
```

Spark Deconstructed: *Log Mining Example*




```
# transformed RDDs
errors = lines.filter(lambda x: x[0] == "ERROR")
messages = errors.map(lambda x: x[1])

# persistence
messages.cache()
```

Spark Deconstructed: *Log Mining Example*




```
# action 1
messages.filter(lambda x: x.find("mysql") > -1).count()
```

A Brief History

	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**

circa late 1990s:
explosive growth e-commerce and machine data implied that workloads could not fit on a single computer anymore…

notable firms led the shift to *horizontal scale-out* on clusters of commodity hardware, especially for machine learning use cases at scale

A Brief History: *MapReduce*

circa 2002:
mitigate risk of large distributed workloads lost due to disk failures on commodity hardware…

Google File System
Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung
research.google.com/archive/gfs.html

MapReduce: Simplified Data Processing on Large Clusters
Jeffrey Dean, Sanjay Ghemawat
research.google.com/archive/mapreduce.html

A Brief History: *MapReduce*

circa 1979 – **Stanford, MIT, CMU**, etc.
set/list operations in LISP, Prolog, etc., for parallel processing
www-formal.stanford.edu/jmc/history/lisp/lisp.htm

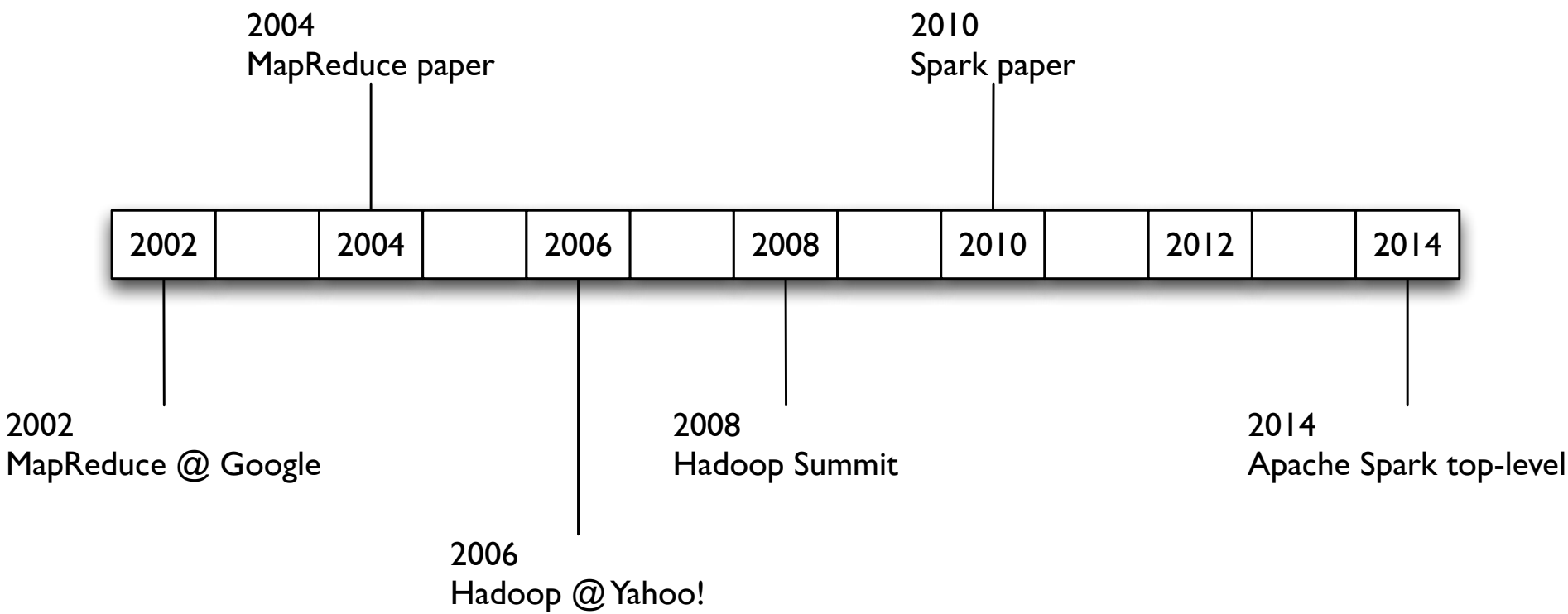
circa 2004 – **Google**
MapReduce: Simplified Data Processing on Large Clusters
Jeffrey Dean and Sanjay Ghemawat
research.google.com/archive/mapreduce.html

circa 2006 – **Apache**
Hadoop, originating from the Nutch Project
Doug Cutting
research.yahoo.com/files/cutting.pdf

circa 2008 – **Yahoo**
web scale search indexing
Hadoop Summit, HUG, etc.
developer.yahoo.com/hadoop/

circa 2009 – **Amazon AWS**
Elastic MapReduce
Hadoop modified for EC2/S3, plus support for Hive, Pig, Cascading, etc.
aws.amazon.com/elasticmapreduce/

A Brief History: *Functional Programming for Big Data*

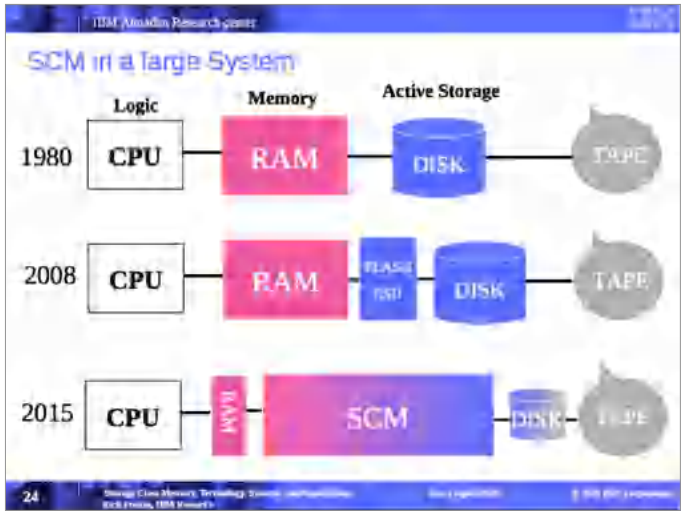


A Brief History: *MapReduce*

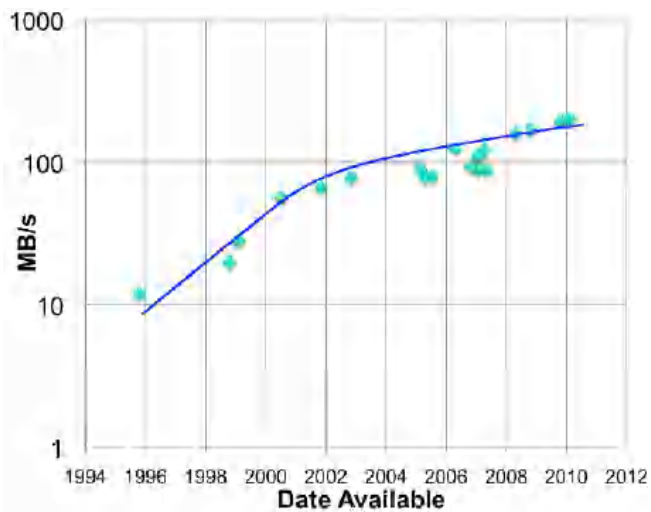
Open Discussion:

Enumerate several changes in data center technologies since 2002…

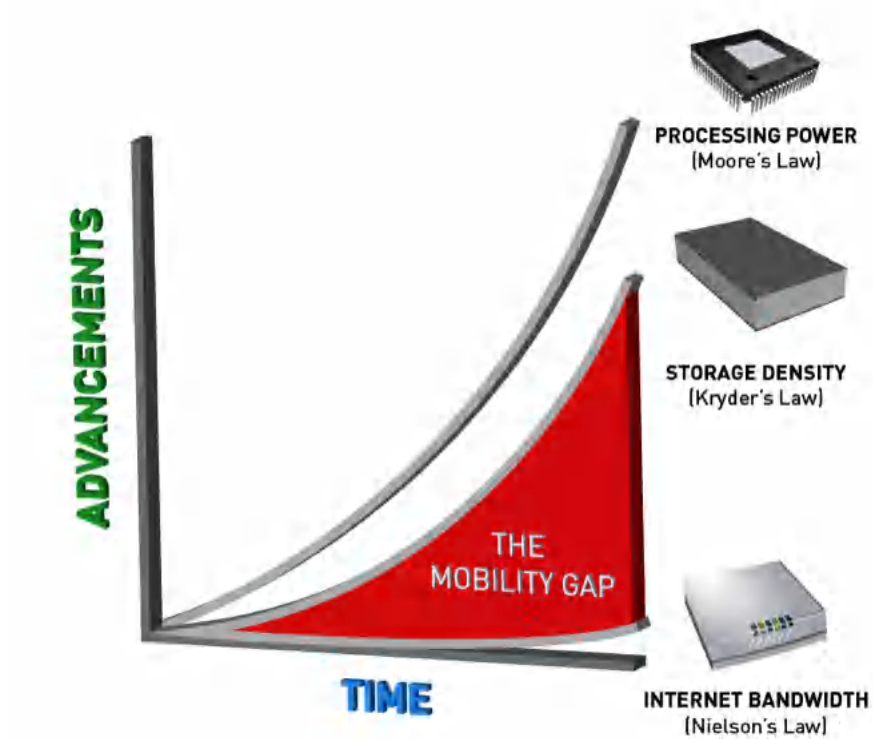
A Brief History: *MapReduce*



Rich Freitas, **IBM Research**



storagenewsletter.com/rubriques/hard-disk-drives/hdd-technology-trends-ibm/



pistoncloud.com/2013/04/storage-and-the-mobility-gap/

meanwhile, spinny disks haven't changed all that much…

A Brief History: *MapReduce*

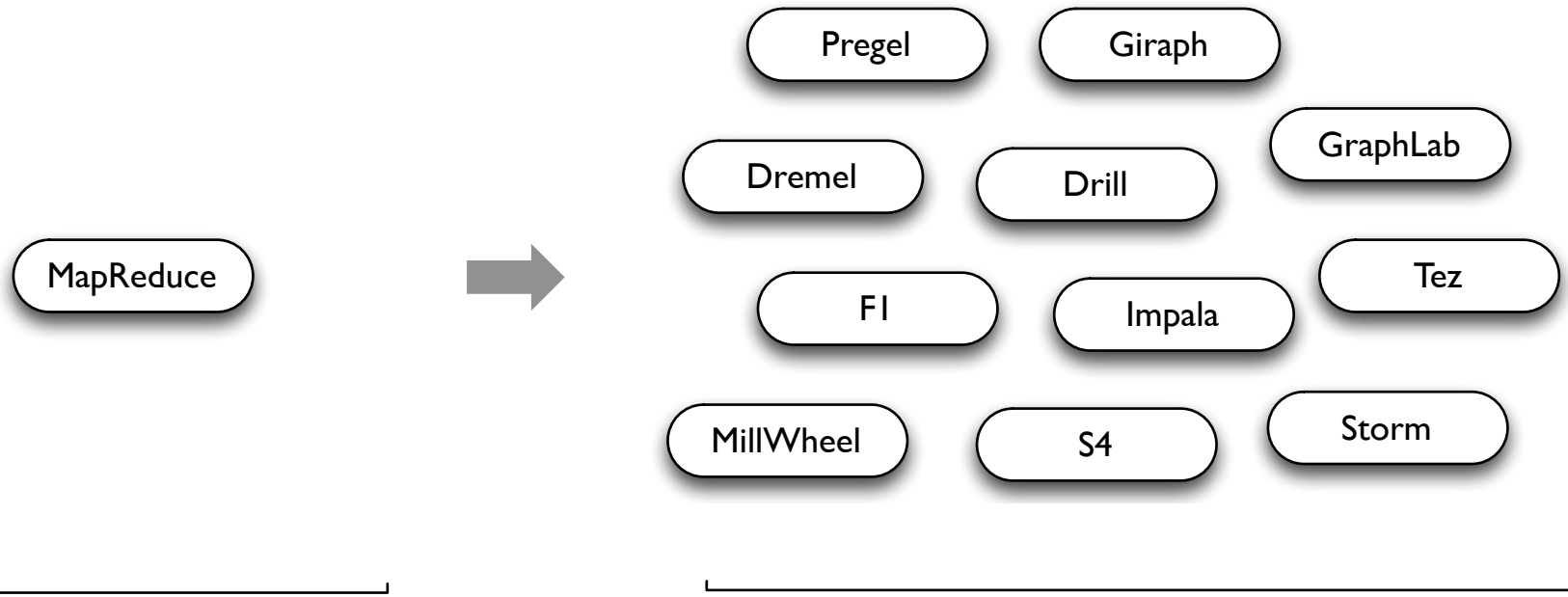
MapReduce use cases showed two major limitations:

1. difficultly of programming directly in MR
2. performance bottlenecks, or batch not fitting the use cases

In short, MR doesn't compose well for large applications

Therefore, people built *specialized systems* as workarounds…

A Brief History: *MapReduce*



General Batch Processing

Specialized Systems:
iterative, interactive, streaming, graph, etc.

MR doesn't compose well for large applications, and so *specialized systems* emerged as workarounds

A Brief History: *Spark*

Developed in 2009 at UC Berkeley AMPLab, then open sourced in 2010, Spark has since become one of the largest OSS communities in big data, with over 200 contributors in 50+ organizations

"Organizations that are looking at big data challenges – including collection, ETL, storage, exploration and analytics – should consider Spark for its in-memory performance and the breadth of its model. It supports advanced analytics solutions on Hadoop clusters, including the iterative model required for machine learning and graph analysis."

Gartner, *Advanced Analytics and Data Science* (2014)

spark.apache.org

A Brief History: *Spark*

circa 2010:
a unified engine for enterprise data workflows,
based on commodity hardware a decade later…

Spark: Cluster Computing with Working Sets
Matei Zaharia, Mosharaf Chowdhury,
Michael Franklin, Scott Shenker, Ion Stoica
people.csail.mit.edu/matei/papers/2010/hotcloud_spark.pdf

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing
Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave,
Justin Ma, Murphy McCauley, Michael Franklin, Scott Shenker, Ion Stoica
usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf

A Brief History: *Spark*

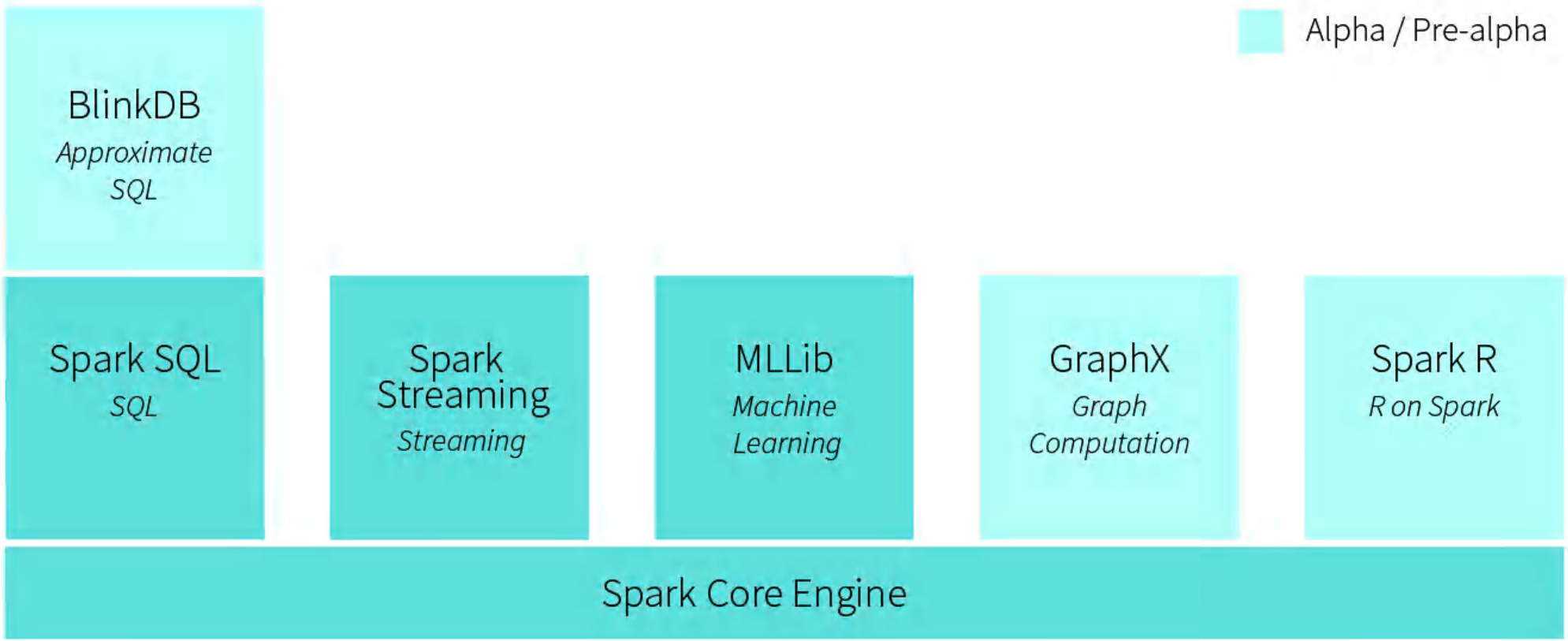
Unlike the various specialized systems, Spark's goal was to *generalize* MapReduce to support new apps within same engine

Two reasonably small additions are enough to express the previous models:

- *fast data sharing*
- *general DAGs*

This allows for an approach which is more efficient for the engine, and much simpler for the end users

A Brief History: *Spark*



A Brief History: *Spark*

Some key points about Spark:

- handles batch, interactive, and real-time within a single framework
- native integration with Java, Python, Scala
- programming at a higher level of abstraction
- more general: map/reduce is just one set of supported constructs

A Brief History: *Key distinctions for Spark vs. MapReduce*

- generalized patterns
  ⇒ unified engine for many use cases
- lazy evaluation of the lineage graph
  ⇒ reduces wait states, better pipelining
- generational differences in hardware
  ⇒ off-heap use of large memory spaces
- functional programming / ease of use
  ⇒ reduction in cost to maintain large apps
- lower overhead for starting jobs
- less expensive shuffles

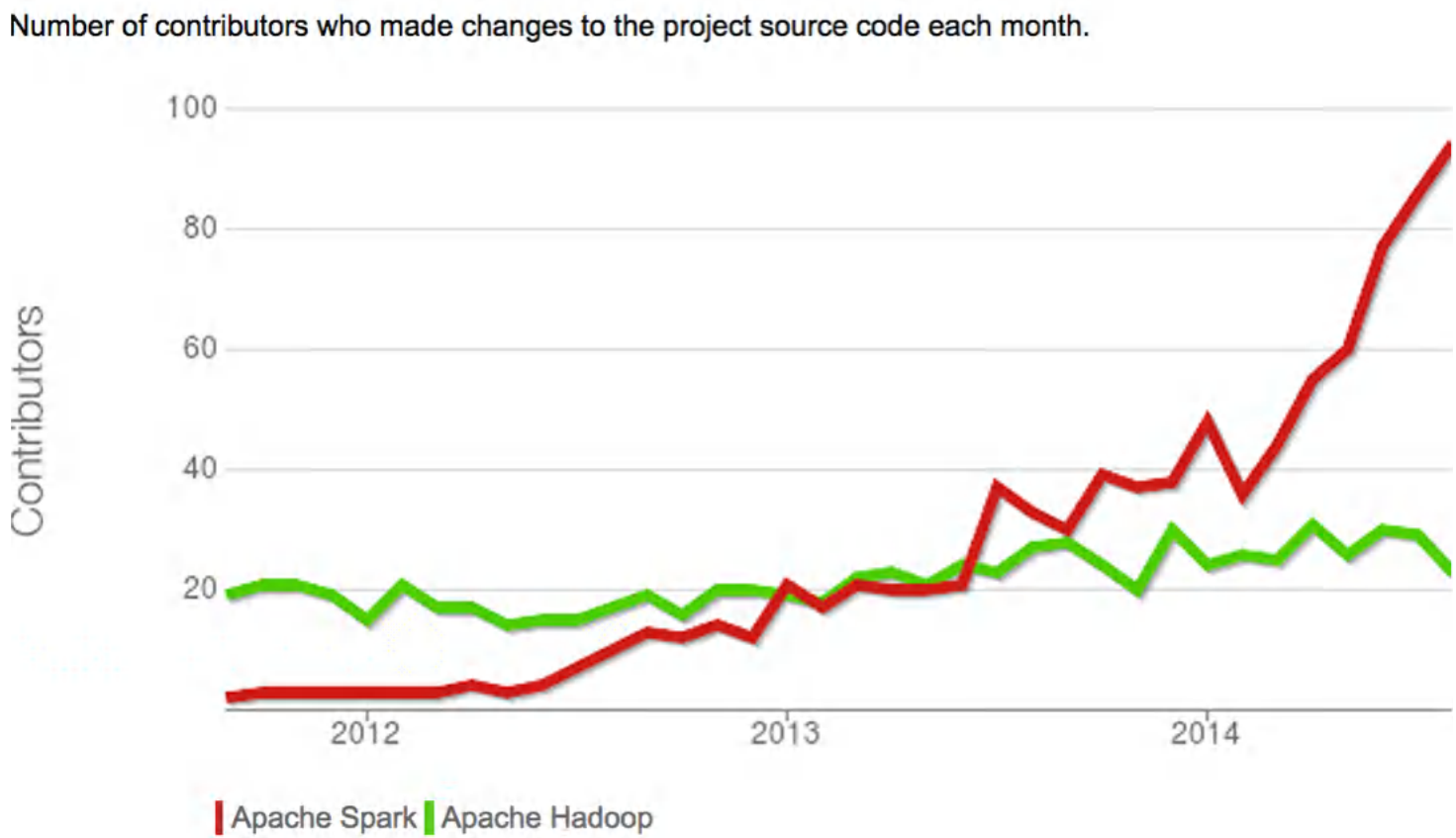
TL;DR: *Smashing The Previous Petabyte Sort Record*

databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

	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**

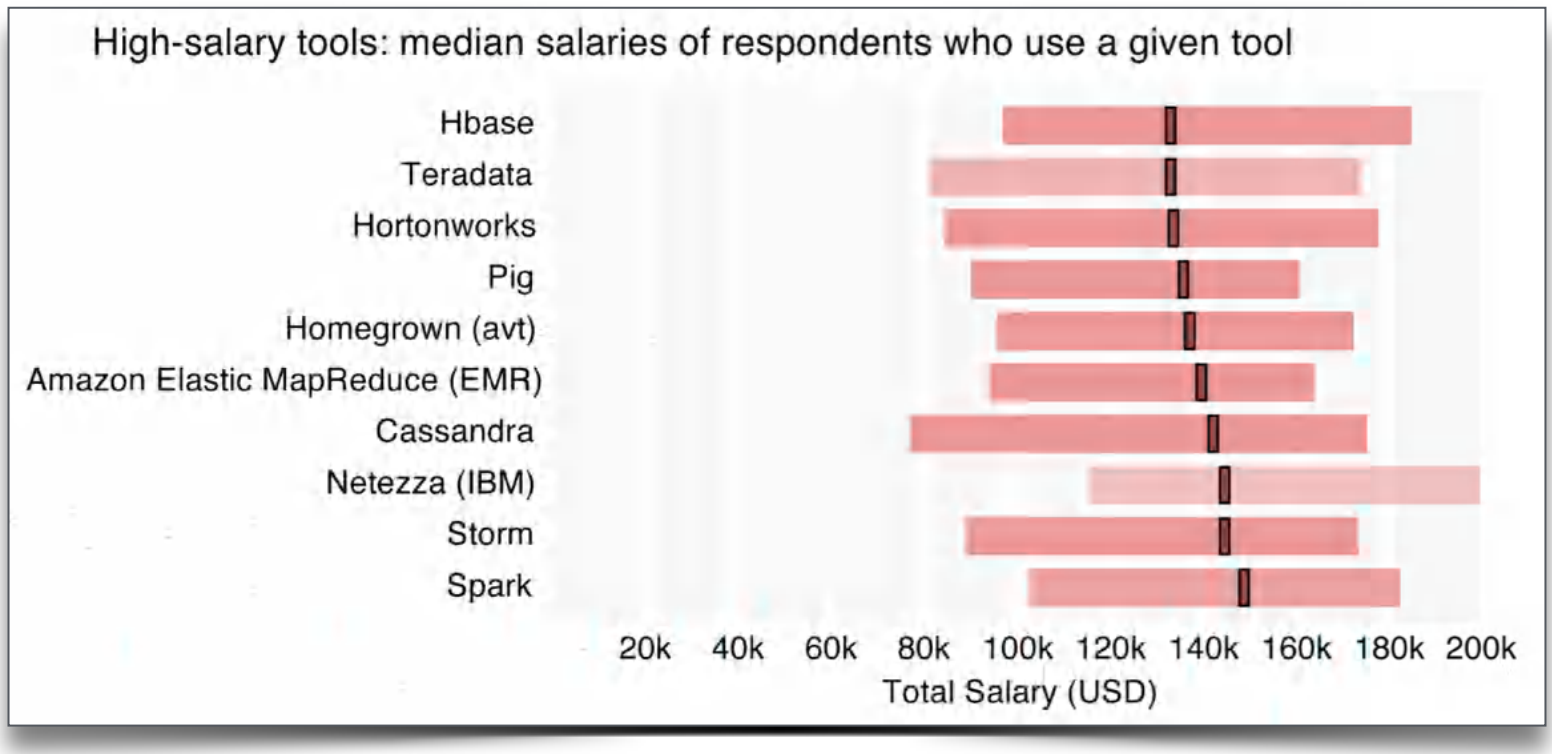
TL;DR: *Sustained Exponential Growth*

Spark is one of the most active Apache projects
ohloh.net/orgs/apache



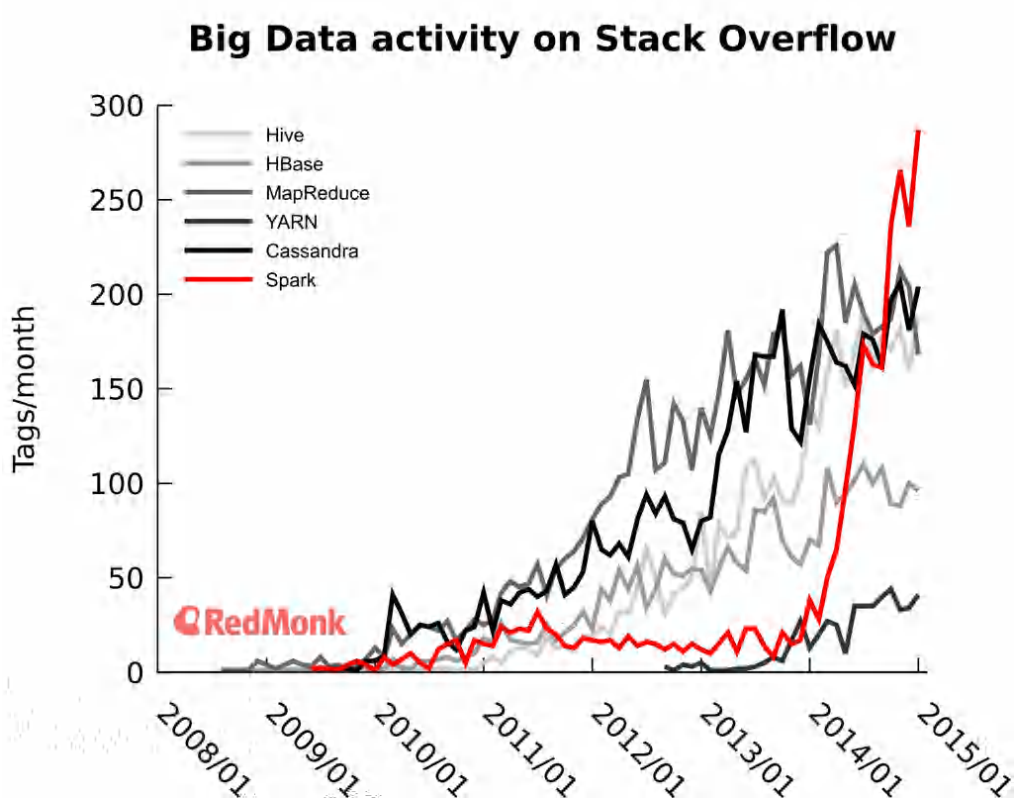
TL;DR: *Spark Expertise Tops Median Salaries within Big Data*

oreilly.com/data/free/2014-data-science-salary-survey.csp



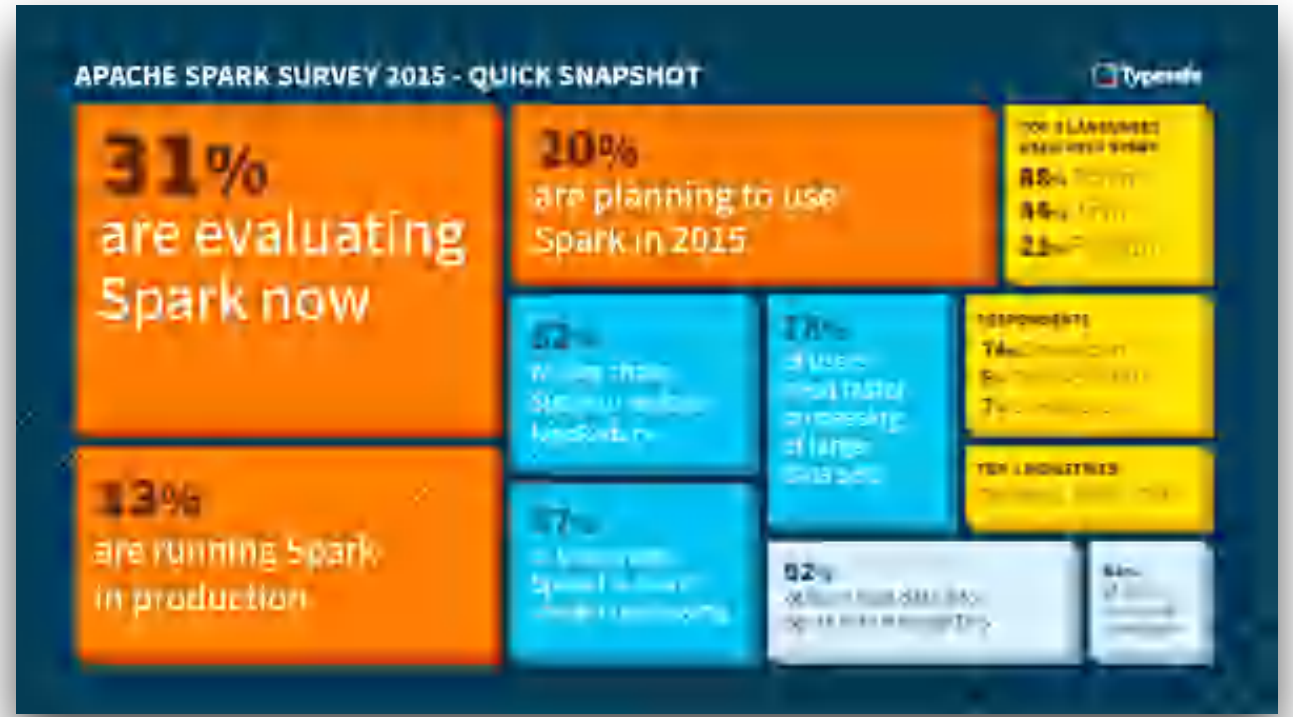
TL;DR: *Spark on StackOverflow*

twitter.com/dberkholz/status/568561792751771648

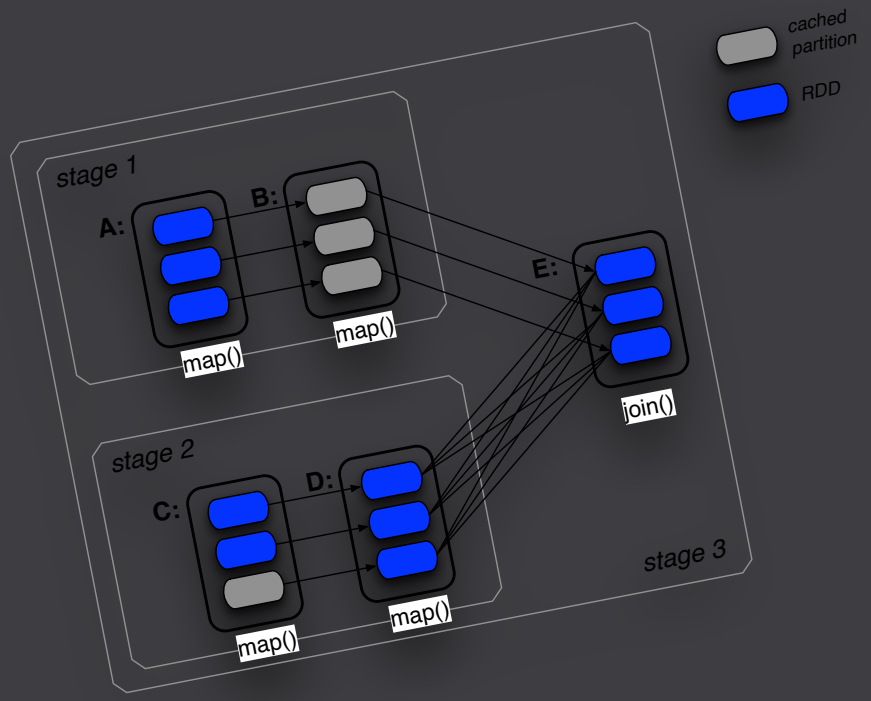


TL;DR: *Spark Survey 2015 by Databricks + Typesafe*

databricks.com/blog/2015/01/27/big-data-projects-are-hungry-for-simpler-and-more-powerful-tools-survey-validates-apache-spark-is-gaining-developer-traction.html



Ex #3:
WC, Joins, Shuffles



Coding Exercise: *WordCount*

Definition:

count how often each word appears in a collection of text documents

This simple program provides a good test case for parallel processing, since it:

- requires a minimal amount of code
- demonstrates use of both symbolic and numeric values
- isn't many steps away from search indexing
- serves as a "Hello World" for Big Data apps

```
void map (String doc_id, String text):
  for each word w in segment(text):
    emit(w, "1");


void reduce (String word, Iterator group):
  int count = 0;

  for each pc in group:
    count += Int(pc);

  emit(word, String(count));
```

A distributed computing framework that can run WordCount **efficiently in parallel at scale** can likely handle much larger and more interesting compute problems

Coding Exercise: *WordCount*

```
public class WordCount {
  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length < 2) {
      System.err.println("Usage: wordcount <in> [<in>,...] <out>");
      System.exit(2);
    }
    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    for (int i = 0; i < otherArgs.length - 1; ++i) {
      FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
    }
    FileOutputFormat.setOutputPath(job,
      new Path(otherArgs[otherArgs.length - 1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

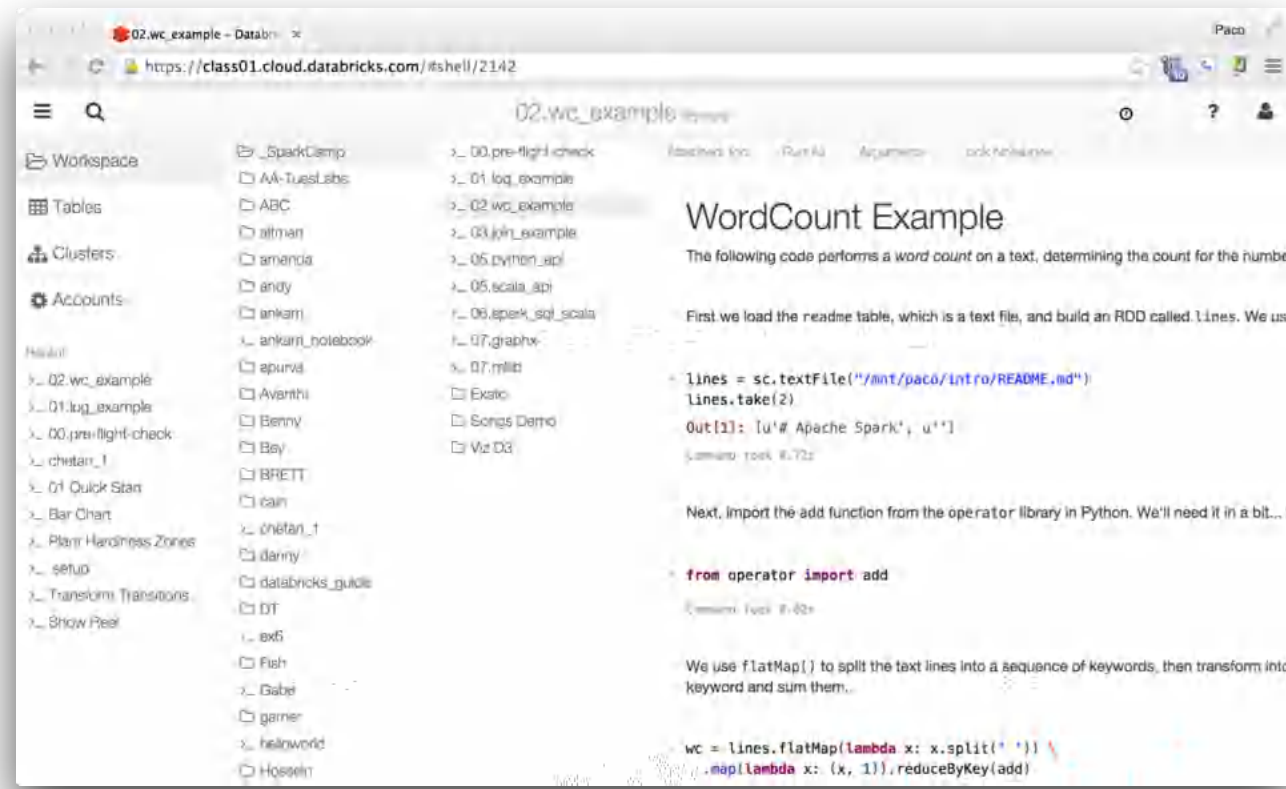
```
1  val f = sc.textFile(inputPath)
2  val w = f.flatMap(l => l.split(" ")).map(word => (word, 1)).cache()
3  w.reduceByKey(_ + _).saveAsText(outputPath)
```

WordCount in 3 lines of Spark

WordCount in 50+ lines of Java MR

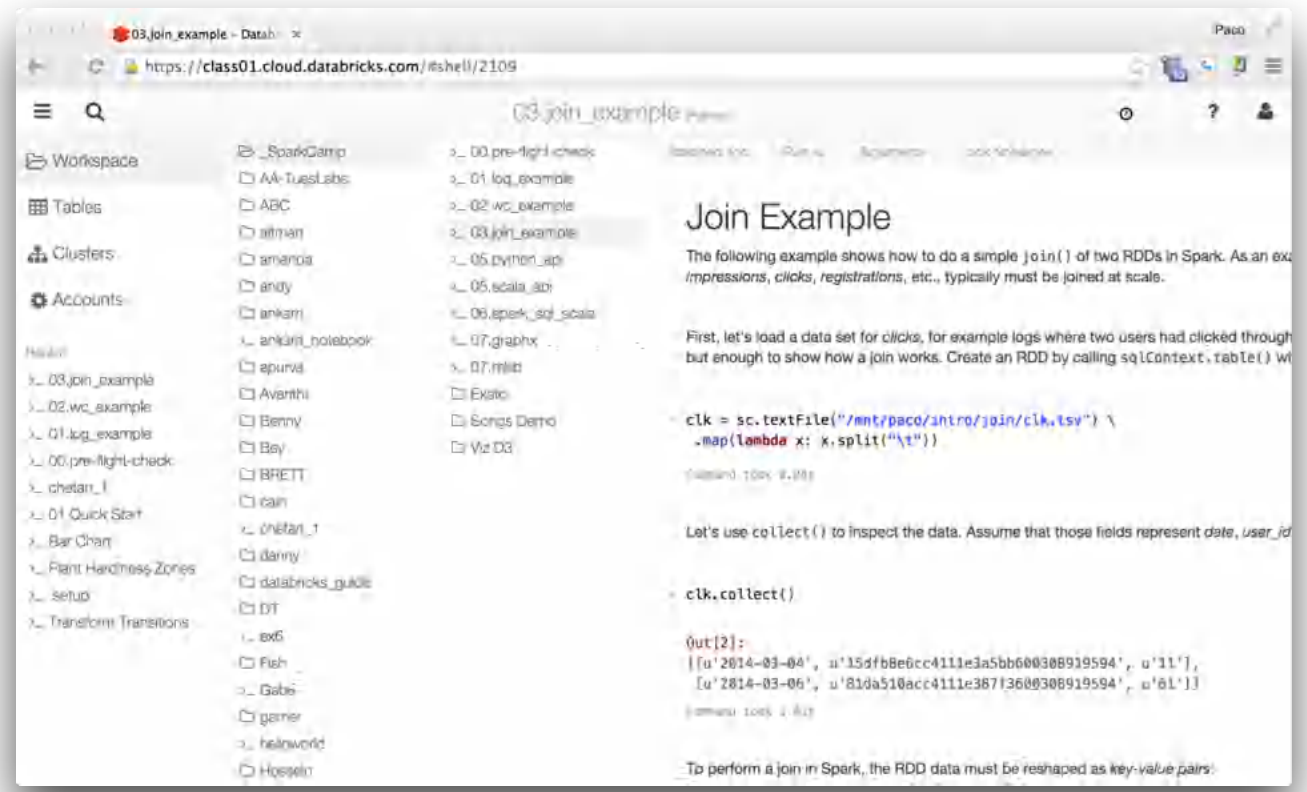
Coding Exercise: *WordCount*

Clone and run `/_SparkCamp/02.wc_example` in your folder:

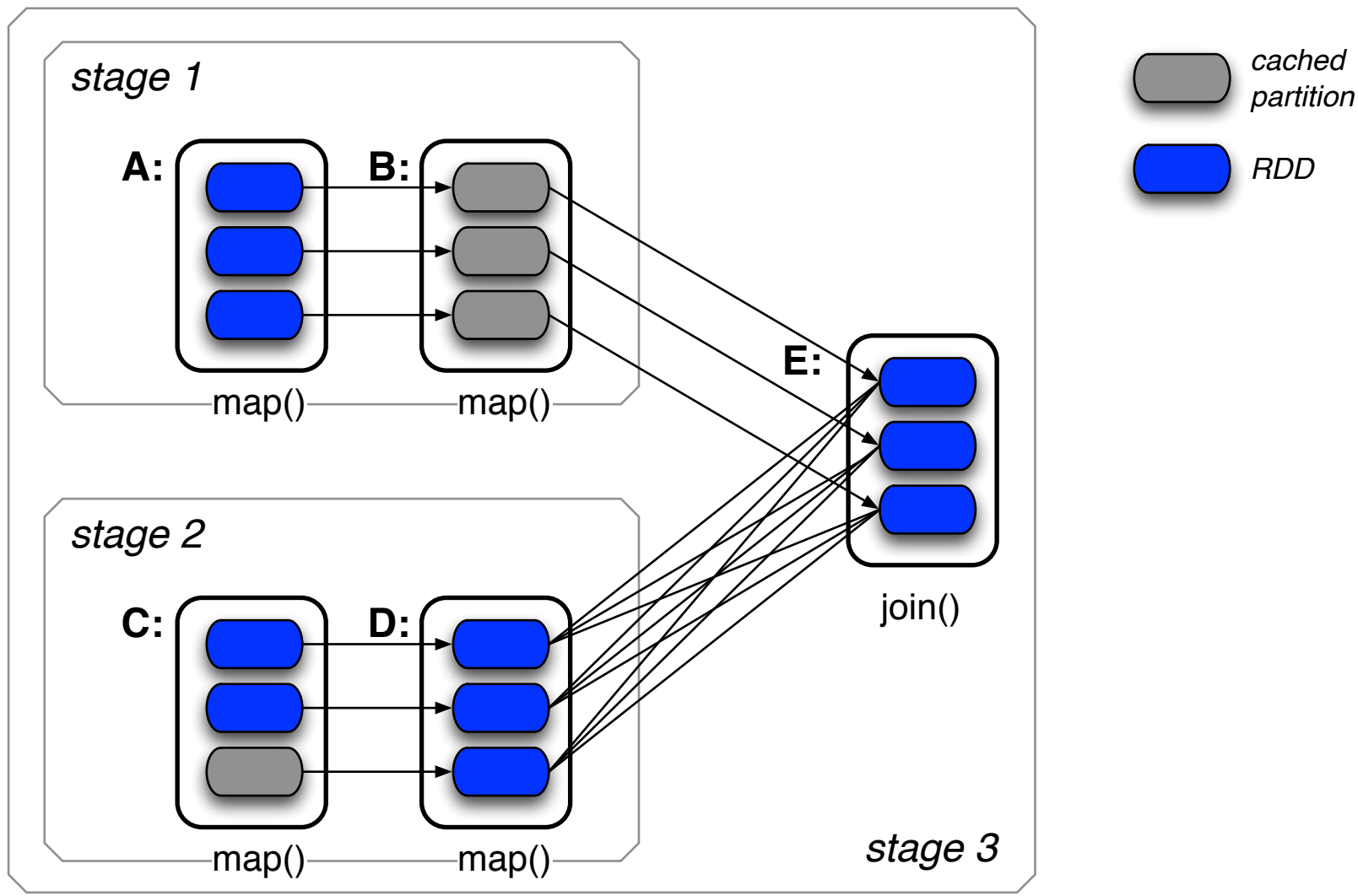


Coding Exercise: *Join*

Clone and run `/_SparkCamp/03.join_example` in your folder:

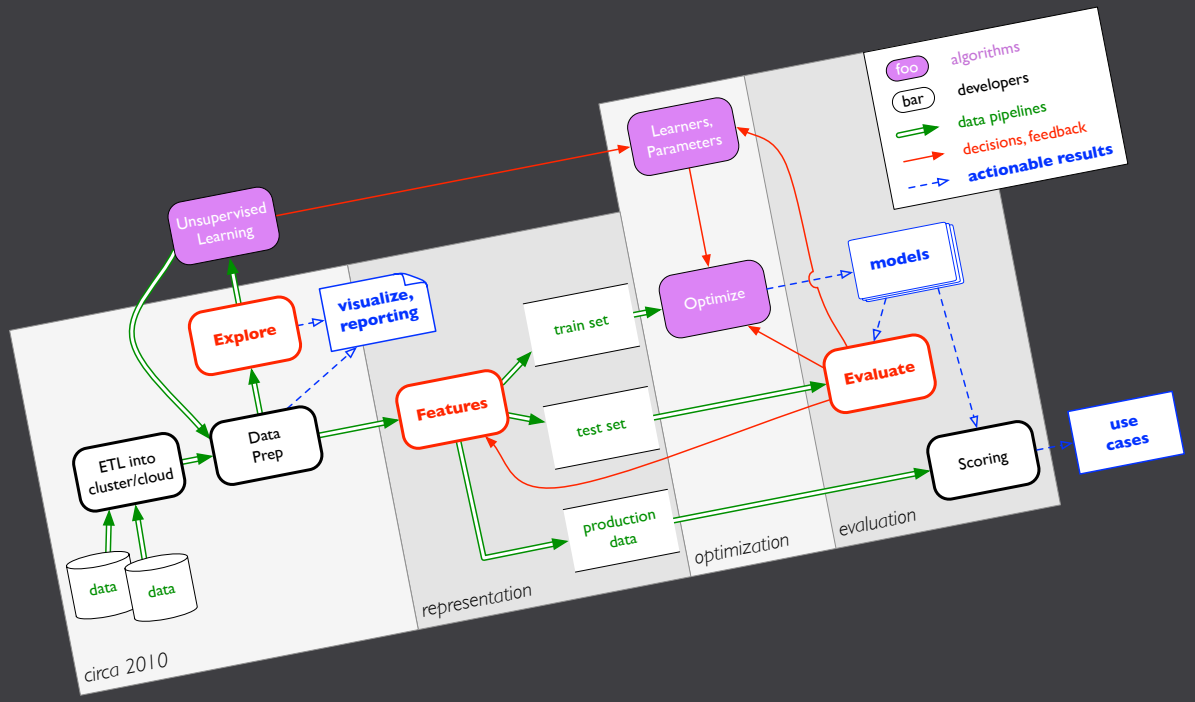


Coding Exercise: *Join and its Operator Graph*



15 min break:

DBC Essentials



DBC Essentials: *What is Databricks Cloud?*

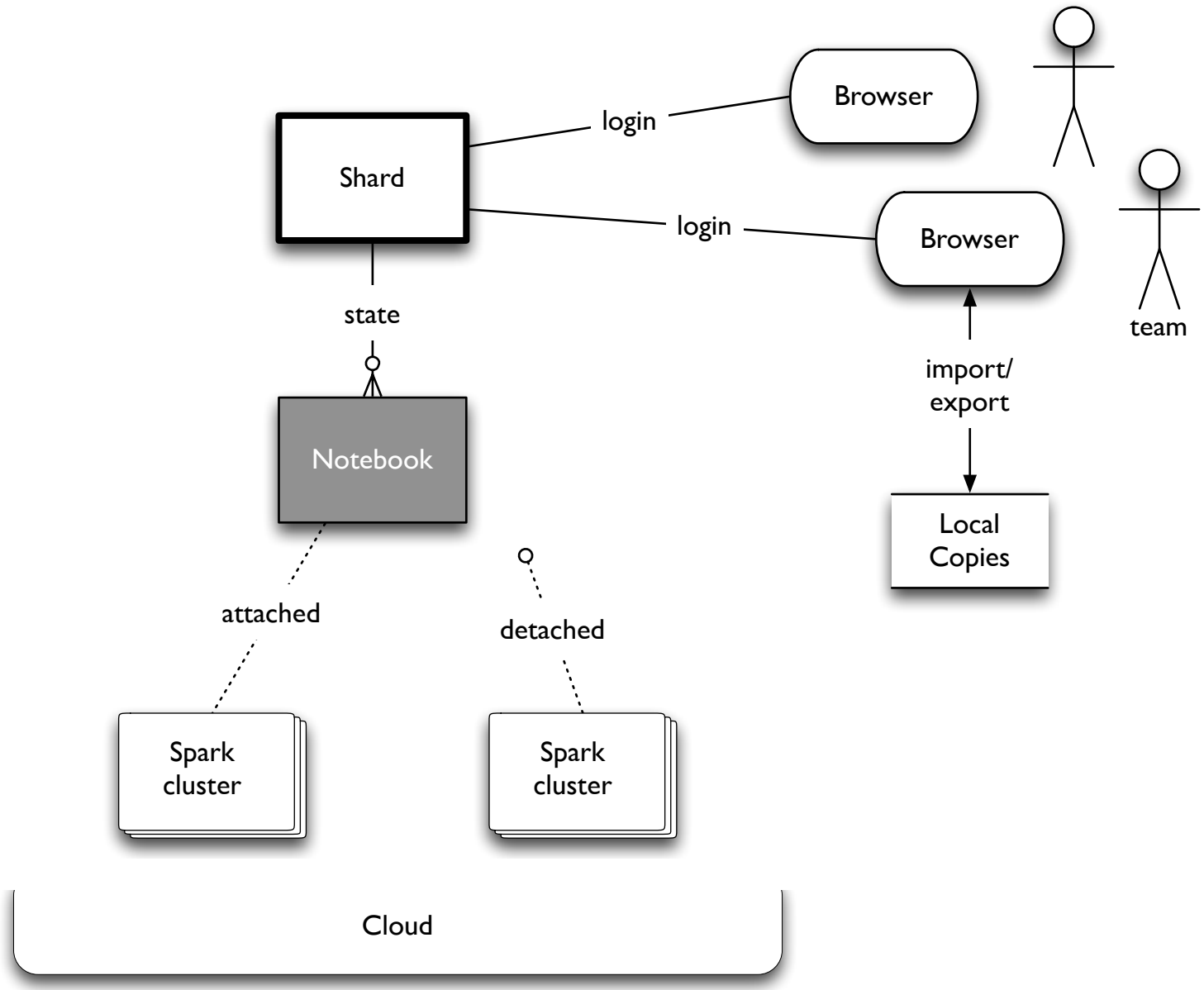
Also see **FAQ** for more details…

DBC Essentials: *What is Databricks Cloud?*

Also see **FAQ** for more details…

	key concepts
Shard	*an instance of Databricks Workspace*
Cluster	*a Spark cluster (multiple per shard)*
Notebook	*a list of markdown, executable commands, and results*
Dashboard	*a flexible space to create operational visualizations*

DBC Essentials: *Team, State, Collaboration, Elastic Resources*



DBC Essentials: *Team, State, Collaboration, Elastic Resources*

Excellent collaboration properties, based on the use of:

- *comments*
- *cloning*
- *decoupled state* of notebooks vs. clusters
- relative *independence* of code blocks within a notebook

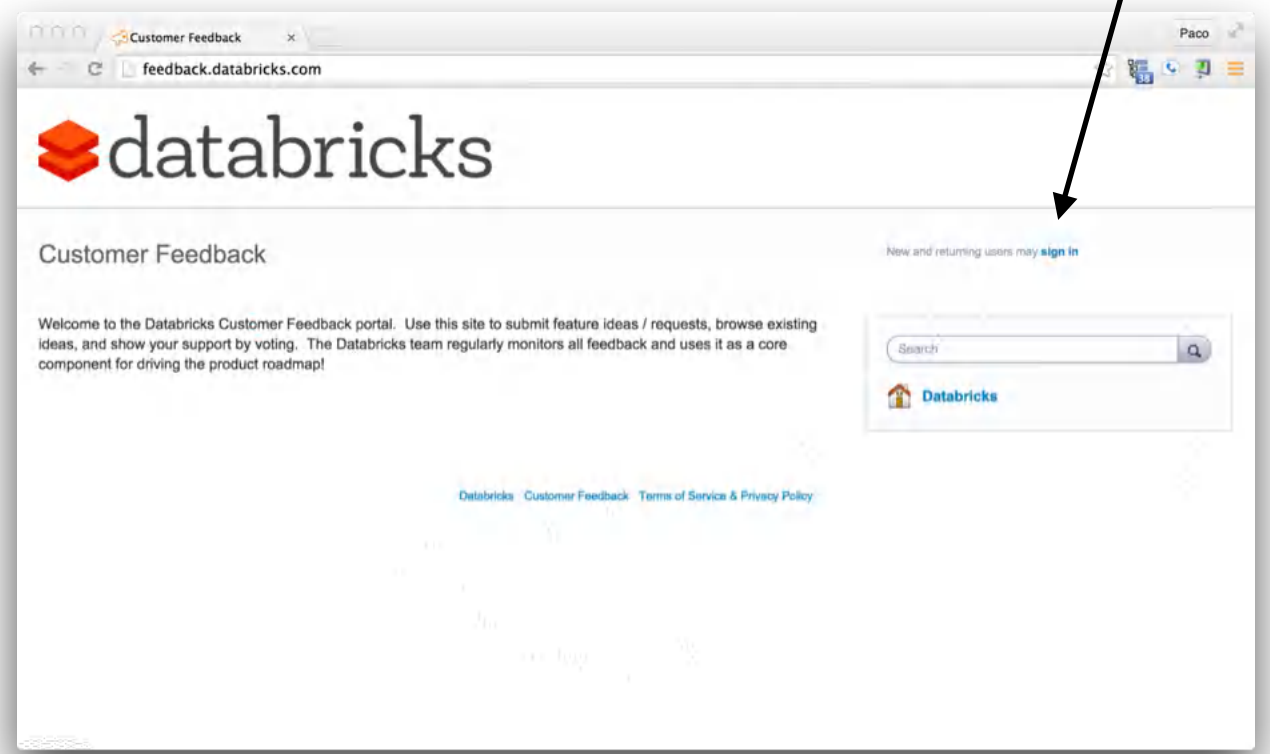
DBC Essentials: *Feedback*

Other feedback, suggestions, etc.?

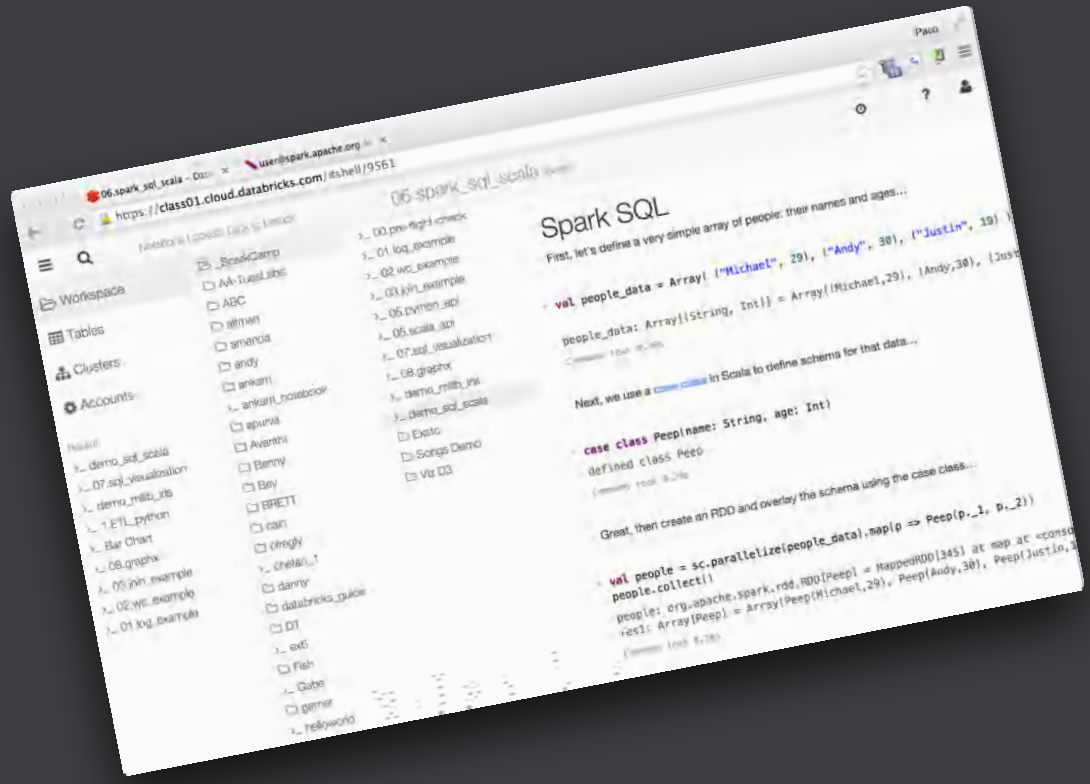
http://feedback.databricks.com/

http://forums.databricks.com/

UserVoice login in top/right corner…



How to “Think Notebooks”



databricks

Think Notebooks:

How to "think" in terms of leveraging notebooks, based on **Computational Thinking**:

"The way we depict space has a great deal to do with how we behave in it."
– **David Hockney**

Think Notebooks: *Computational Thinking*

"The impact of computing extends far beyond science… affecting all aspects of our lives. To flourish in today's world, everyone needs computational thinking." – CMU

Computing now ranks alongside the proverbial Reading, Writing, and Arithmetic…

Center for Computational Thinking @ CMU
http://www.cs.cmu.edu/~CompThink/

Exploring Computational Thinking @ Google
https://www.google.com/edu/computational-thinking/

Think Notebooks: *Computational Thinking*

Computational Thinking provides a structured way of conceptualizing the problem…

In effect, developing notes for yourself and your team

These in turn can become the basis for team process, software requirements, etc.,

In other words, conceptualize how to leverage computing resources at scale to build high-ROI apps for Big Data

Think Notebooks: *Computational Thinking*

The general approach, in four parts:

- Decomposition: *decompose a complex problem into smaller solvable problems*
- Pattern Recognition: *identify when a known approach can be leveraged*
- Abstraction: *abstract from those patterns into generalizations as strategies*
- Algorithm Design: *articulate strategies as algorithms, i.e. as general recipes for how to handle complex problems*

Think Notebooks:

How to "think" in terms of leveraging notebooks, by the numbers:

1. create a new notebook
2. copy the assignment description as markdown
3. split it into separate code cells
4. for each step, write your code under the markdown
5. run each step and verify your results

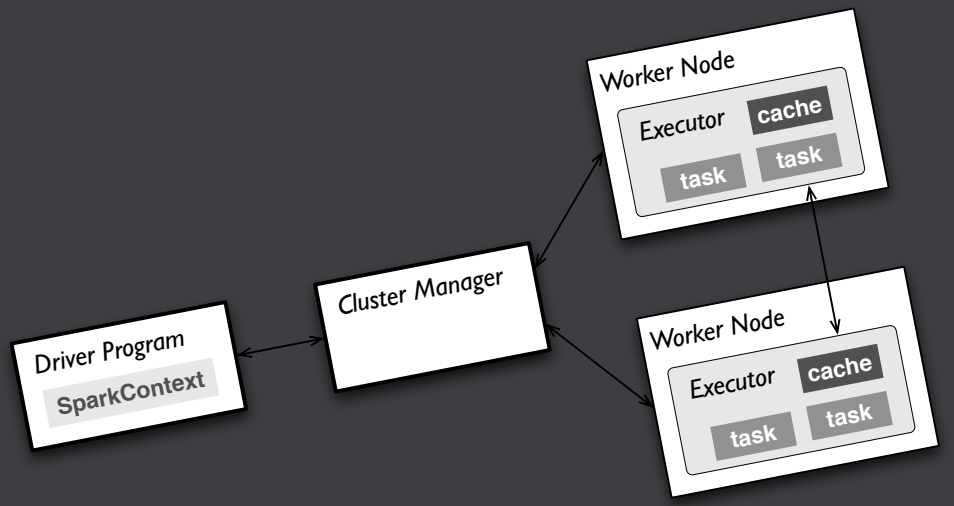
Coding Exercises: *Workflow assignment*

Let's assemble the pieces of the previous few code examples, using two files:

```
/mnt/paco/intro/CHANGES.txt
/mnt/paco/intro/README.md
```

1. create RDDs to filter each line for the keyword `Spark`
2. perform a WordCount on each, i.e., so the results are (K,V) pairs of (keyword, count)
3. join the two RDDs
4. how many instances of `Spark` are there in each file?

Tour of Spark API



Spark Essentials:

The essentials of the Spark API in both Scala and Python…

```
/_SparkCamp/05.scala_api
/_SparkCamp/05.python_api
```

Let's start with the basic concepts, which are covered in much more detail in the docs:

spark.apache.org/docs/latest/scala-programming-guide.html

Spark Essentials: *SparkContext*

First thing that a Spark program does is create a `SparkContext` object, which tells Spark how to access a cluster

In the shell for either Scala or Python, this is the `sc` variable, which is created automatically

Other programs must use a constructor to instantiate a new `SparkContext`

Then in turn `SparkContext` gets used to create other variables

Spark Essentials: *SparkContext*

Scala:

```
sc
res0: org.apache.spark.SparkContext = org.apache.spark.SparkContext@6ad51e9c
```

Python:

```
sc
Out[1]: <__main__.RemoteContext at 0x7ff0bfb18a10>
```

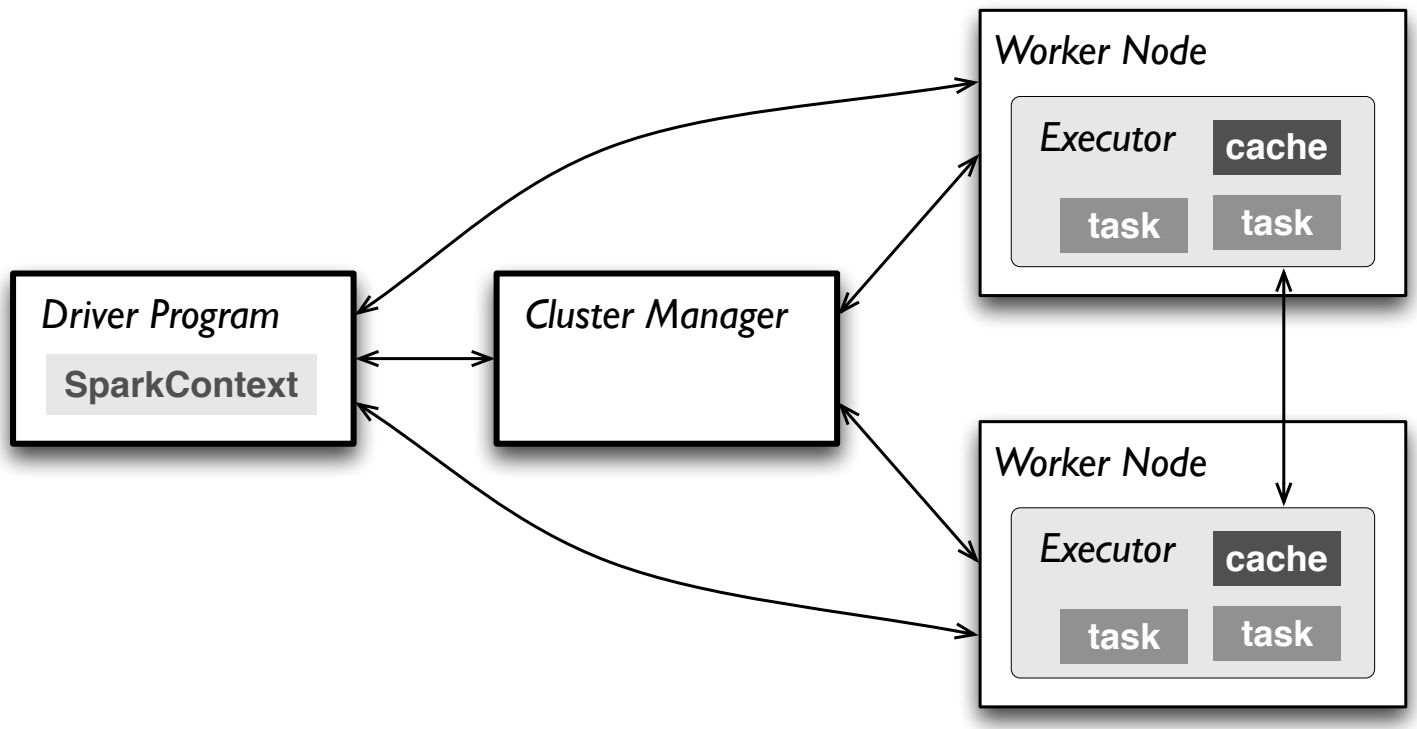
Spark Essentials: *Master*

The `master` parameter for a `SparkContext` determines which cluster to use

master	*description*
`local`	run Spark locally with one worker thread (no parallelism)
`local[K]`	run Spark locally with K worker threads (ideally set to # cores)
`spark://HOST:PORT`	connect to a Spark standalone cluster; PORT depends on config (7077 by default)
`mesos://HOST:PORT`	connect to a Mesos cluster; PORT depends on config (5050 by default)

Spark Essentials: *Master*

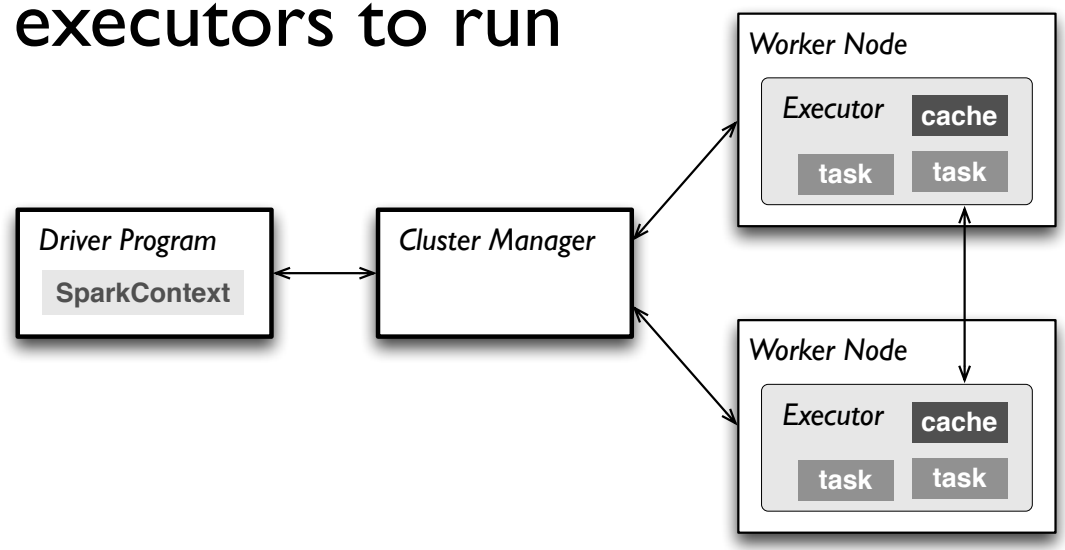
spark.apache.org/docs/latest/cluster-overview.html



Spark Essentials: *Clusters*

The *driver* performs the following:

1. connects to a *cluster manager* to allocate resources across applications
2. acquires *executors* on cluster nodes – processes run compute tasks, cache data
3. sends *app code* to the executors
4. sends *tasks* for the executors to run



Spark Essentials: *RDD*

Resilient **D**istributed **D**atasets (RDD) are the primary abstraction in Spark – a fault-tolerant collection of elements that can be operated on in parallel

There are currently two types:

- *parallelized collections* – take an existing Scala collection and run functions on it in parallel
- *Hadoop datasets* – run functions on each record of a file in Hadoop distributed file system or any other storage system supported by Hadoop

Spark Essentials: *RDD*

- two types of operations on RDDs: *transformations* and *actions*
- transformations are lazy (not computed immediately)
- the transformed RDD gets recomputed when an action is run on it (default)
- however, an RDD can be *persisted* into storage in memory or disk

Spark Essentials: *RDD*

Scala:

```
val data = Array(1, 2, 3, 4, 5)
data: Array[Int] = Array(1, 2, 3, 4, 5)

val distData = sc.parallelize(data)
distData: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[24970]
```

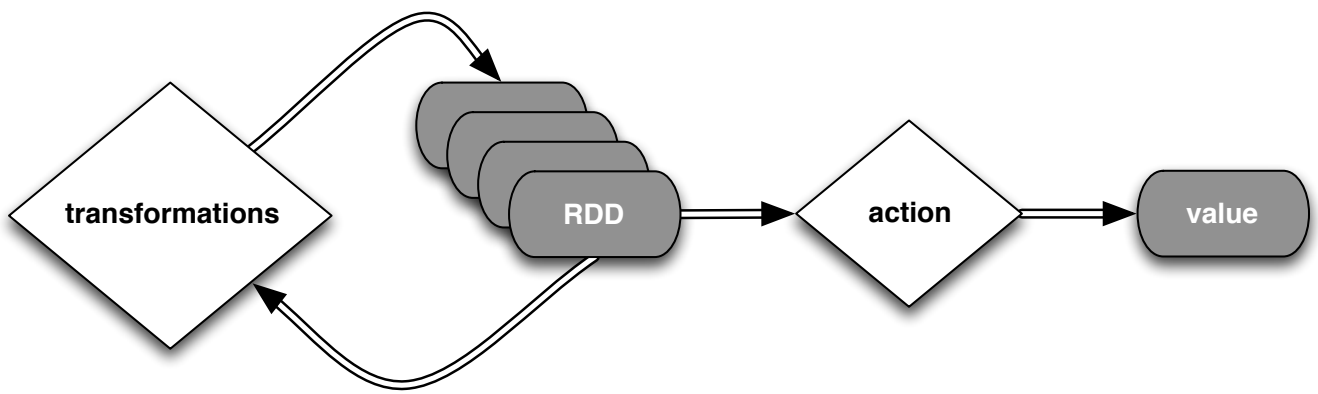
Python:

```
data = [1, 2, 3, 4, 5]
data
Out[2]: [1, 2, 3, 4, 5]

distData = sc.parallelize(data)
distData
Out[3]: ParallelCollectionRDD[24864] at parallelize at PythonRDD.scala:364
```

Spark Essentials: *RDD*

Spark can create RDDs from any file stored in HDFS or other storage systems supported by Hadoop, e.g., local file system, Amazon S3, Hypertable, HBase, etc.

Spark supports text files, SequenceFiles, and any other Hadoop `InputFormat`, and can also take a directory or a glob (e.g. `/data/201404*`)



Spark Essentials: *RDD*

Scala:

```
val distFile = sqlContext.table("readme")
distFile: org.apache.spark.sql.SchemaRDD =
SchemaRDD[24971] at RDD at SchemaRDD.scala:108
```

Python:

```
distFile = sqlContext.table("readme").map(lambda x: x[0])
distFile
Out[11]: PythonRDD[24920] at RDD at PythonRDD.scala:43
```

Spark Essentials: *Transformations*

Transformations create a new dataset from an existing one

All transformations in Spark are *lazy*: they do not compute their results right away – instead they remember the transformations applied to some base dataset

- optimize the required calculations
- recover from lost data partitions

Spark Essentials: *Transformations*

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

Spark Essentials: *Transformations*

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

Spark Essentials: *Transformations*

Scala:

```
val distFile = sqlContext.table("readme").map(_(0).asInstanceOf[String])
distFile.map(l => l.split(" ")).collect()
distFile.flatMap(l => l.split(" ")).collect()
```

distFile is a collection of lines

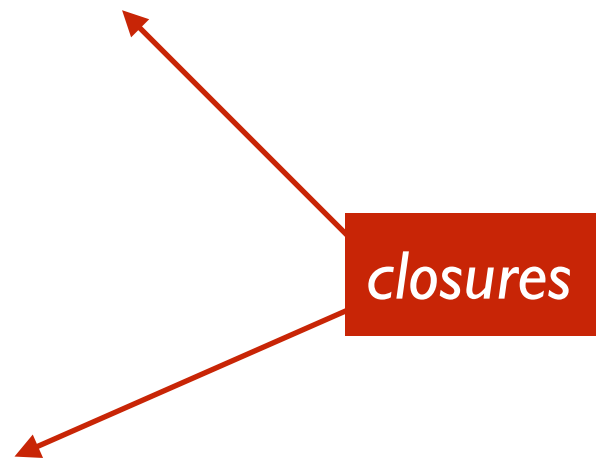
Python:

```
distFile = sqlContext.table("readme").map(lambda x: x[0])
distFile.map(lambda x: x.split(' ')).collect()
distFile.flatMap(lambda x: x.split(' ')).collect()
```

Spark Essentials: *Transformations*

Scala:

```
val distFile = sqlContext.table("readme").map(_(0).asInstanceOf[String])
distFile.map(l => l.split(" ")).collect()
distFile.flatMap(l => l.split(" ")).collect()
```



Python:

```
distFile = sqlContext.table("readme").map(lambda x: x[0])
distFile.map(lambda x: x.split(' ')).collect()
distFile.flatMap(lambda x: x.split(' ')).collect()
```

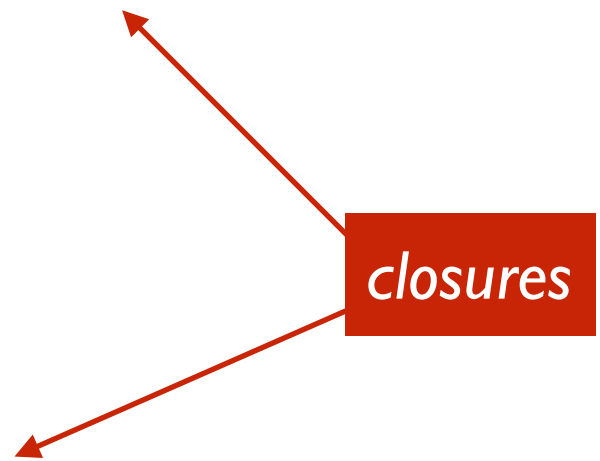
Spark Essentials: *Transformations*

Scala:

```
val distFile = sqlContext.table("readme").map(_(0).asInstanceOf[String])
distFile.map(l => l.split(" ")).collect()
distFile.flatMap(l => l.split(" ")).collect()
```

Python:

```
distFile = sqlContext.table("readme").map(lambda x: x[0])
distFile.map(lambda x: x.split(' ')).collect()
distFile.flatMap(lambda x: x.split(' ')).collect()
```



looking at the output, how would you compare results for map() vs. flatMap() ?

Spark Essentials: *Actions*

action	*description*
reduce(*func***)**	aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one), and should also be commutative and associative so that it can be computed correctly in parallel
collect()	return all the elements of the dataset as an array at the driver program – usually useful after a filter or other operation that returns a sufficiently small subset of the data
count()	return the number of elements in the dataset
first()	return the first element of the dataset – similar to *take(1)*
take(*n***)**	return an array with the first *n* elements of the dataset – currently not executed in parallel, instead the driver program computes all the elements
takeSample(*withReplacement, fraction, seed***)**	return an array with a random sample of *num* elements of the dataset, with or without replacement, using the given random number generator seed

Spark Essentials: *Actions*

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

Spark Essentials: *Actions*

Scala:

```
val distFile = sqlContext.table("readme").map(_(0).asInstanceOf[String])
val words = f.flatMap(l => l.split(" ")).map(word => (word, 1))
words.reduceByKey(_ + _).collect.foreach(println)
```

Python:

```
from operator import add
f = sqlContext.table("readme").map(lambda x: x[0])
words = f.flatMap(lambda x: x.split(' ')).map(lambda x: (x, 1))
words.reduceByKey(add).collect()
```

Spark Essentials: *Persistence*

Spark can *persist* (or cache) a dataset in memory across operations

spark.apache.org/docs/latest/programming-guide.html#rdd-persistence

Each node stores in memory any slices of it that it computes and reuses them in other actions on that dataset – often making future actions more than 10x faster

The cache is *fault-tolerant*: if any partition of an RDD is lost, it will automatically be recomputed using the transformations that originally created it

Spark Essentials: *Persistence*

transformation	*description*
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
MEMORY_ONLY_SER	Store RDD as serialized Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.
MEMORY_AND_DISK_SER	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.
MEMORY_ONLY_2, **MEMORY_AND_DISK_2,** etc	Same as the levels above, but replicate each partition on two cluster nodes.
OFF_HEAP (experimental)	Store RDD in serialized format in Tachyon.

Spark Essentials: *Persistence*

Scala:

```
val distFile = sqlContext.table("readme").map(_(0).asInstanceOf[String])
val words = f.flatMap(l => l.split(" ")).map(word => (word, 1)).cache()
words.reduceByKey(_ + _).collect.foreach(println)
```

Python:

```
from operator import add
f = sqlContext.table("readme").map(lambda x: x[0])
w = f.flatMap(lambda x: x.split(' ')).map(lambda x: (x, 1)).cache()
w.reduceByKey(add).collect()
```

Spark Essentials: *Broadcast Variables*

Broadcast variables let programmer keep a read-only variable cached on each machine rather than shipping a copy of it with tasks

For example, to give every node a copy of a large input dataset efficiently

Spark also attempts to distribute broadcast variables using efficient broadcast algorithms to reduce communication cost

Spark Essentials: *Broadcast Variables*

Scala:

```
val broadcastVar = sc.broadcast(Array(1, 2, 3))
broadcastVar.value
res10: Array[Int] = Array(1, 2, 3)
```

Python:

```
broadcastVar = sc.broadcast(list(range(1, 4)))
broadcastVar.value
Out[15]: [1, 2, 3]
```

Spark Essentials: *Accumulators*

Accumulators are variables that can only be "added" to through an *associative* operation

Used to implement counters and sums, efficiently in parallel

Spark natively supports accumulators of numeric value types and standard mutable collections, and programmers can extend for new types

Only the driver program can read an accumulator's value, not the tasks

Spark Essentials: *Accumulators*

Scala:

```
val accum = sc.accumulator(0)
sc.parallelize(Array(1, 2, 3, 4)).foreach(x => accum += x)

accum.value
res11: Int = 10
```

Python:

```
accum = sc.accumulator(0)
rdd = sc.parallelize([1, 2, 3, 4])

def f(x):
   global accum
   accum += x

rdd.foreach(f)

accum.value
Out[16]: 10
```

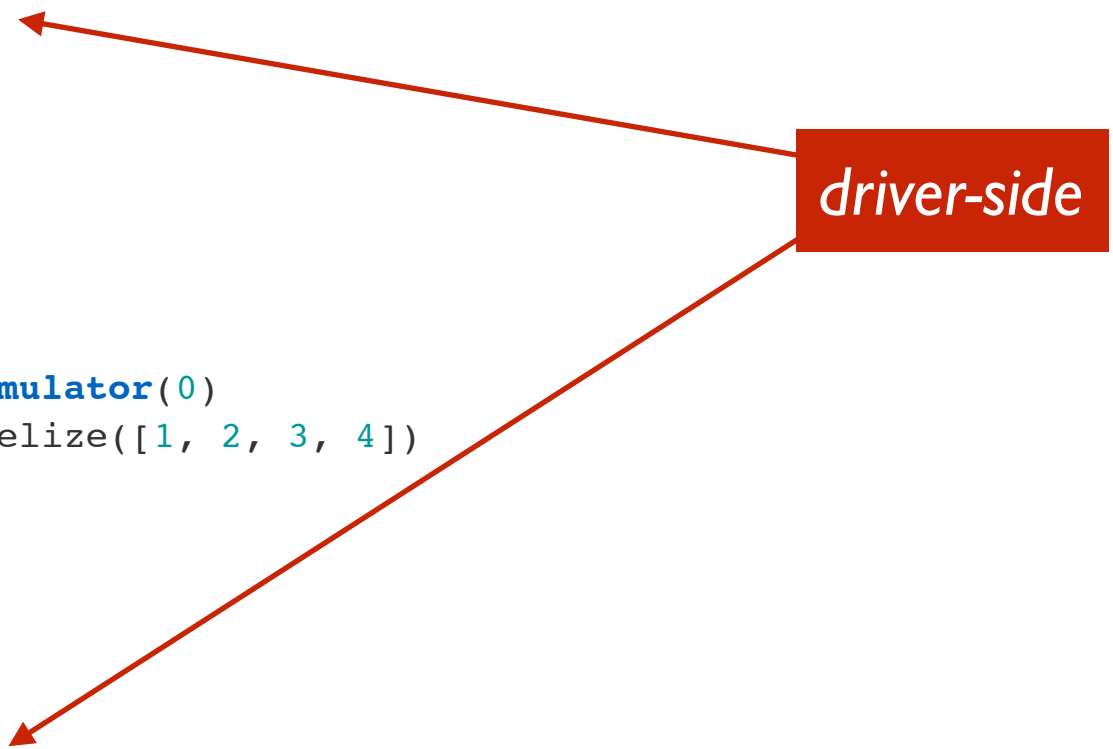
Spark Essentials: *Accumulators*

Scala:

```
val accum = sc.accumulator(0)
sc.parallelize(Array(1, 2, 3, 4)).foreach(x => accum += x)

accum.value
res11: Int = 10
```

Python:

```
accum = sc.accumulator(0)
rdd = sc.parallelize([1, 2, 3, 4])
def f(x):
    global accum
    accum += x

rdd.foreach(f)

accum.value
Out[16]: 10
```



For a deep-dive about broadcast variables and accumulator usage in Spark, see also:

Advanced Spark Features

Matei Zaharia, Jun 2012

ampcamp.berkeley.edu/wp-content/uploads/2012/06/matei-zaharia-amp-camp-2012-advanced-spark.pdf

Spark Essentials: *(K,V) pairs*

Scala:

```
val pair = (a, b)

    pair._1 // => a
    pair._2 // => b
```

Python:

```
pair = (a, b)

    pair[0] # => a
    pair[1] # => b
```

Spark Essentials: *API Details*

For more details about the Scala API:

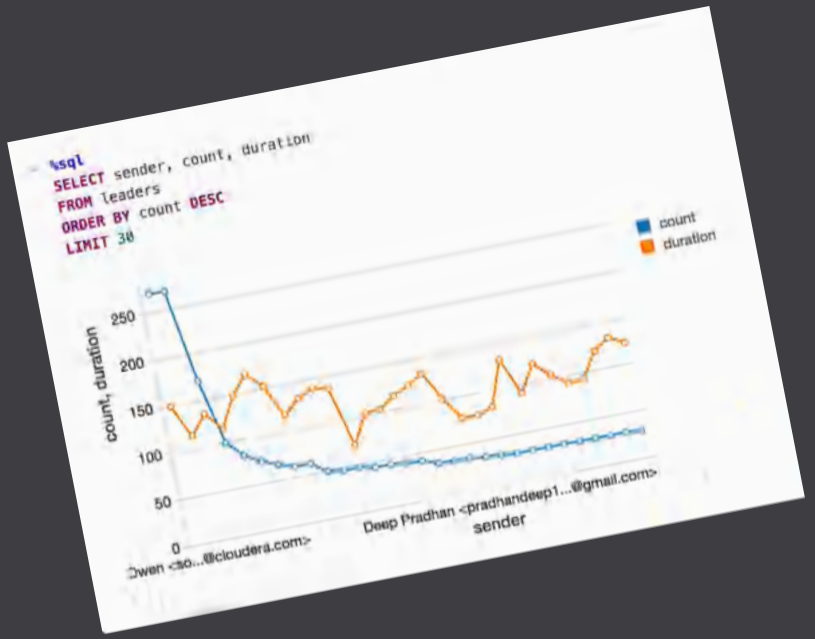
spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.package

For more details about the Python API:

spark.apache.org/docs/latest/api/python/

60 min lunch:

Spark SQL



Spark SQL: *Data Workflows*

blurs the lines between RDDs and relational tables

spark.apache.org/docs/latest/sql-programming-guide.html

intermix SQL commands to query external data, along with complex analytics, in a single app:

- allows SQL extensions based on MLlib
- provides the "heavy lifting" for ETL in DBC

Spark SQL: *Data Workflows*

Spark SQL: Manipulating Structured Data Using Spark
Michael Armbrust, Reynold Xin
databricks.com/blog/2014/03/26/Spark-SQL-manipulating-structured-data-using-Spark.html

The Spark SQL Optimizer and External Data Sources API
Michael Armbrust
youtu.be/GQSNJAzxOr8

What's coming for Spark in 2015
Patrick Wendell
youtu.be/YWppYPWznSQ

Introducing DataFrames in Spark for Large Scale Data Science
Reynold Xin, Michael Armbrust, Davies Liu
databricks.com/blog/2015/02/17/introducing-dataframes-in-spark-for-large-scale-data-science.html

Parquet is a columnar format, supported by many different Big Data frameworks

http://parquet.io/

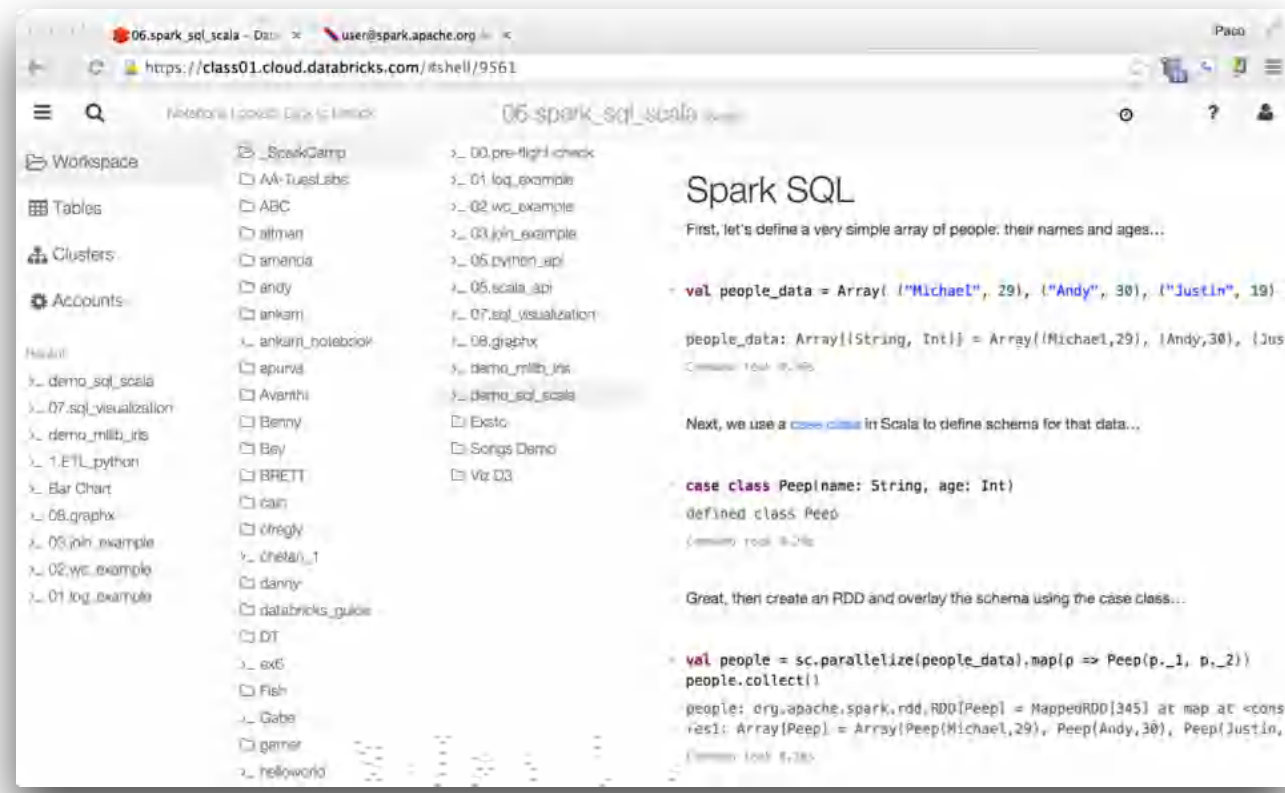
Spark SQL supports read/write of parquet files, automatically preserving the schema of the original data (HUGE benefits)

See also:

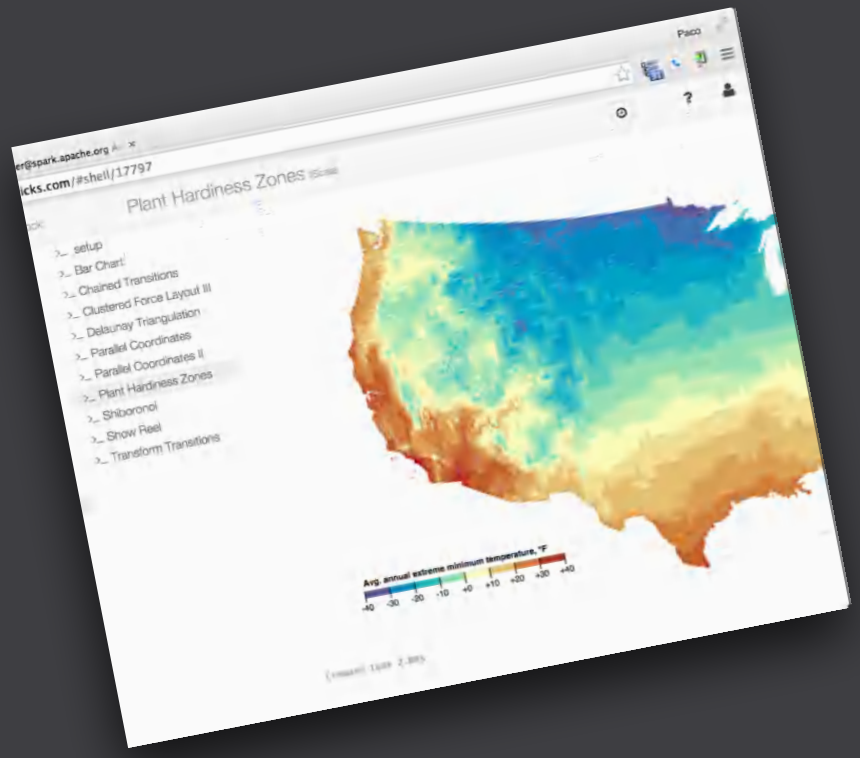
Efficient Data Storage for Analytics with Parquet 2.0
Julien Le Dem @Twitter
slideshare.net/julienledem/th-210pledem

Spark SQL: *SQL Demo*

Demo of `/_SparkCamp/demo_sql_scala`
by the instructor:

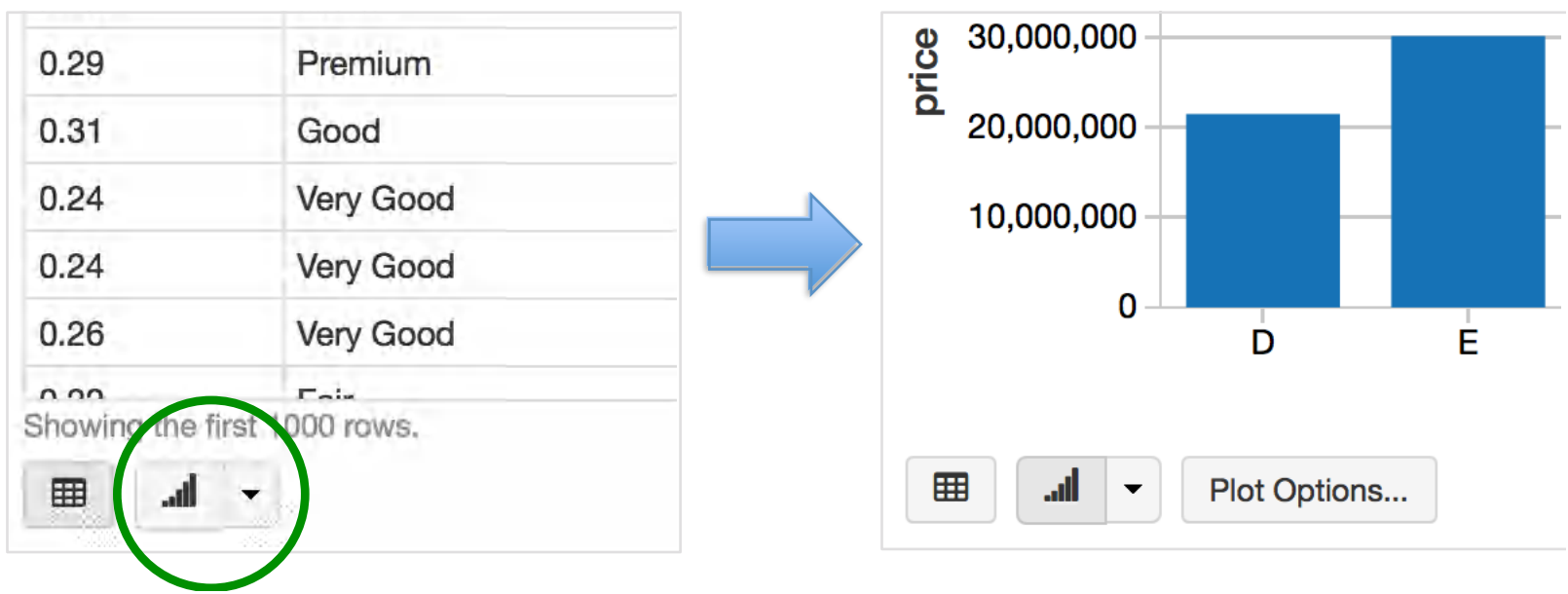


Visualization



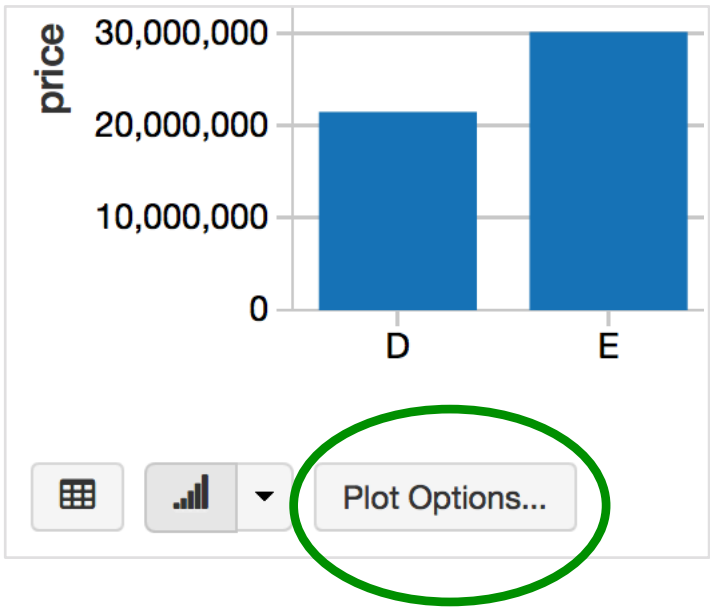
Visualization: *Built-in Plots*

For any SQL query, you can show the results as a table, or generate a plot from with a single click…



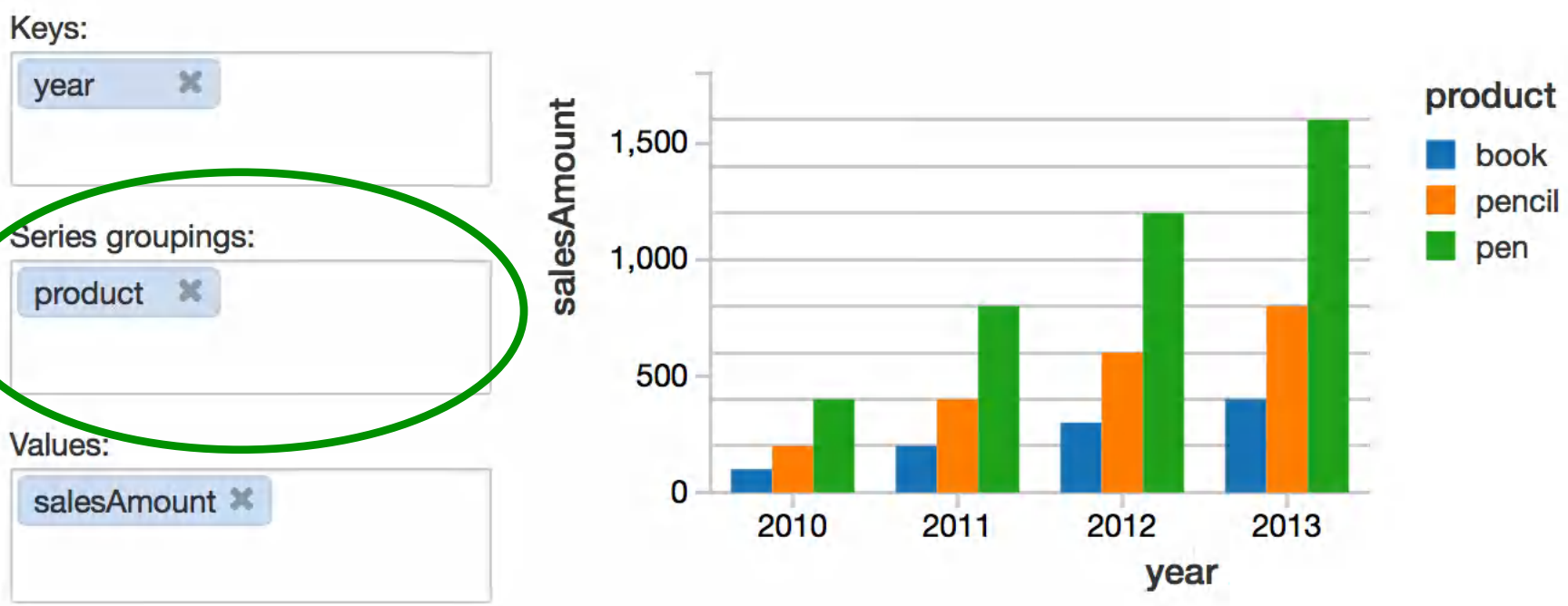
Visualization: *Plot Options*

Several of the plot types have additional options to customize the graphs they generate…



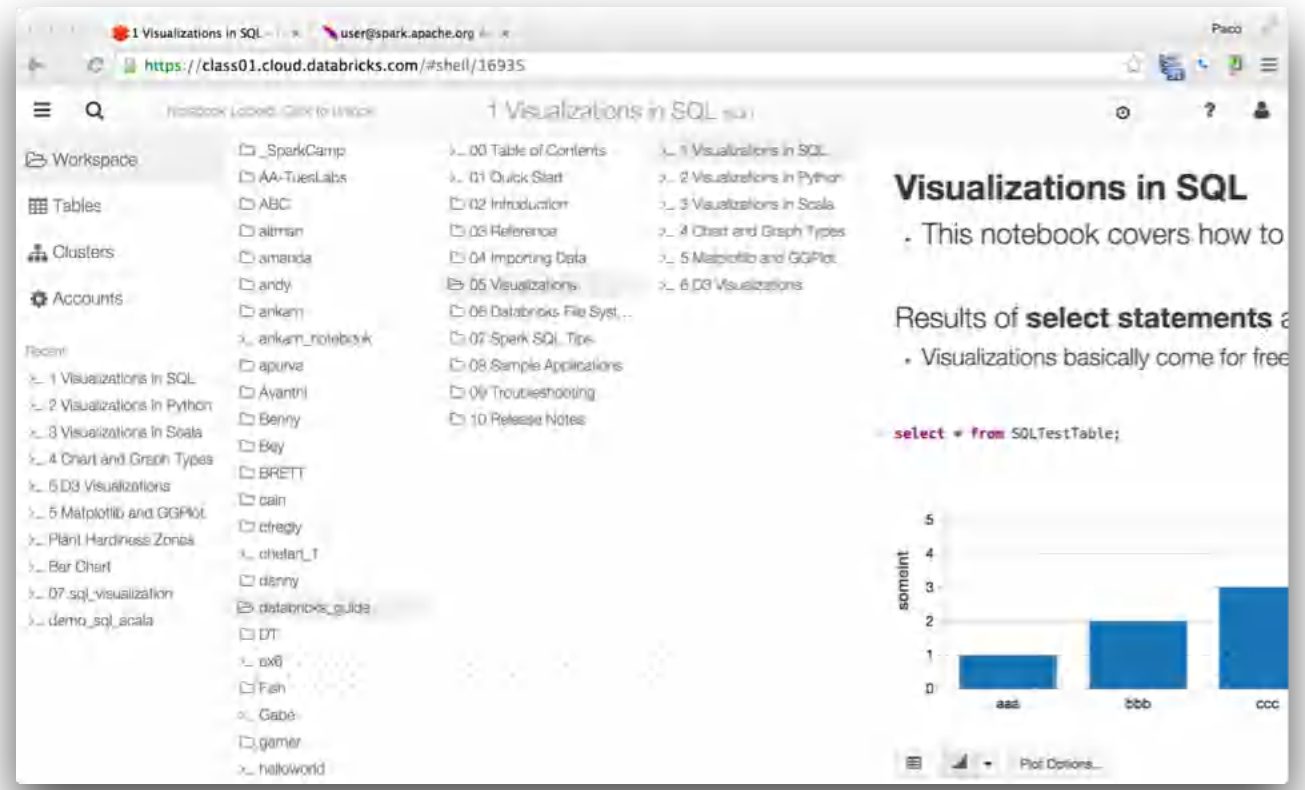
Visualization: *Series Groupings*

For example, *series groupings* can be used to help organize bar charts…



Visualization: *Reference Guide*

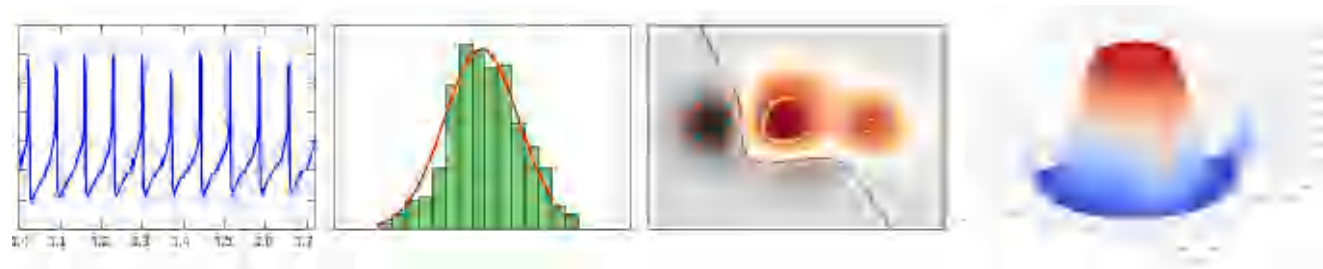
See `/databricks-guide/05 Visualizations` for details about built-in visualizations and extensions…



Visualization: *Using display()*

The `display()` command:

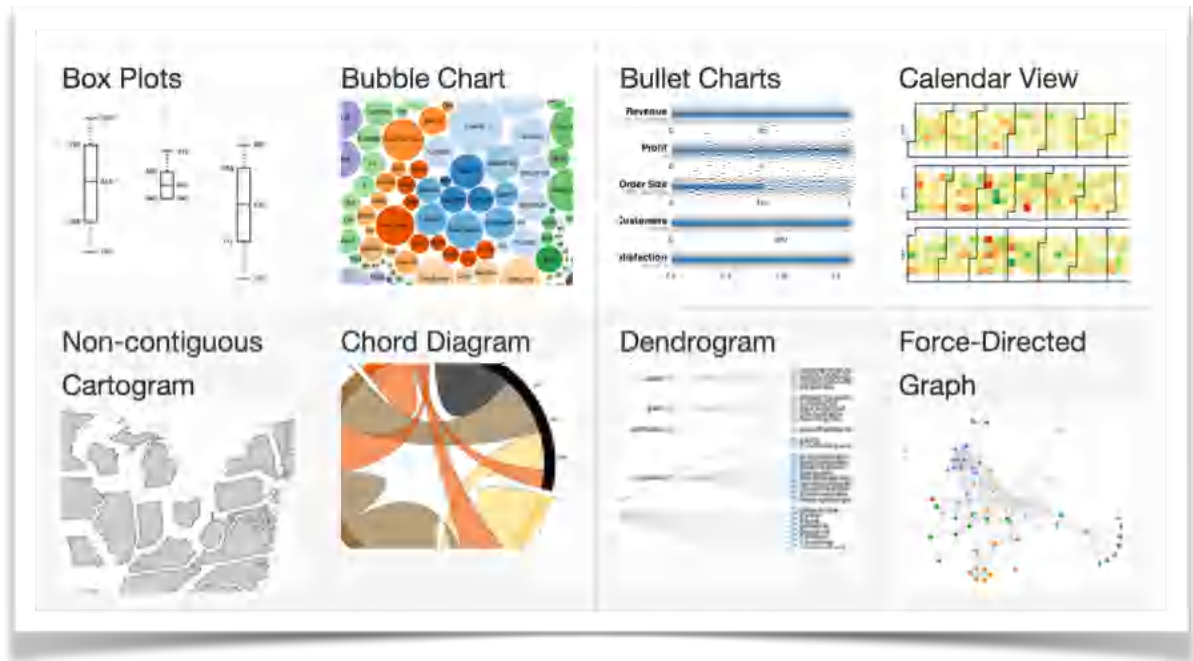
- programmatic access to visualizations
- pass a SchemaRDD to print as an HTML table
- pass a Scala list to print as an HTML table
- call without arguments to display **matplotlib** figures



Visualization: *Using displayHTML()*

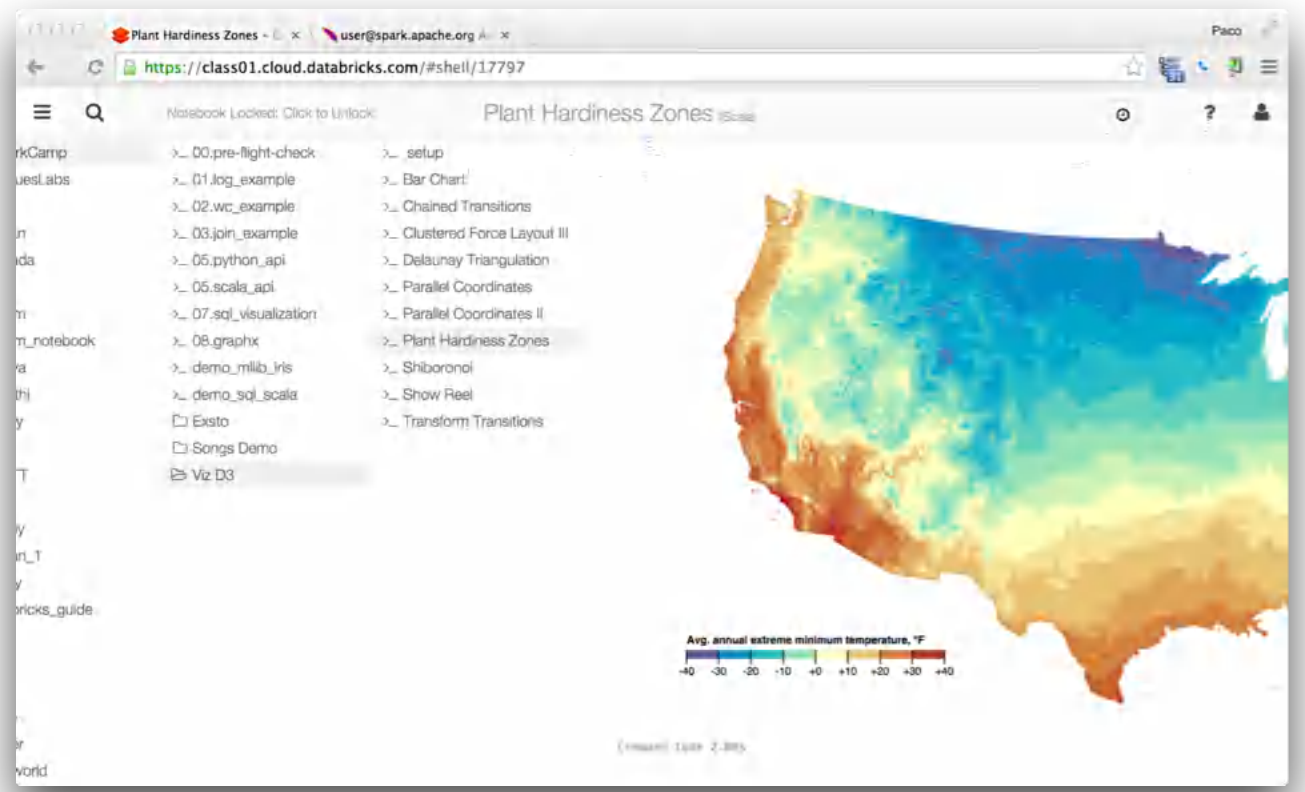
The `displayHTML()` command:

- render any arbitrary HTML/JavaScript
- include JavaScript libraries (advanced feature)
- paste in **D3** examples to get a sense for this…



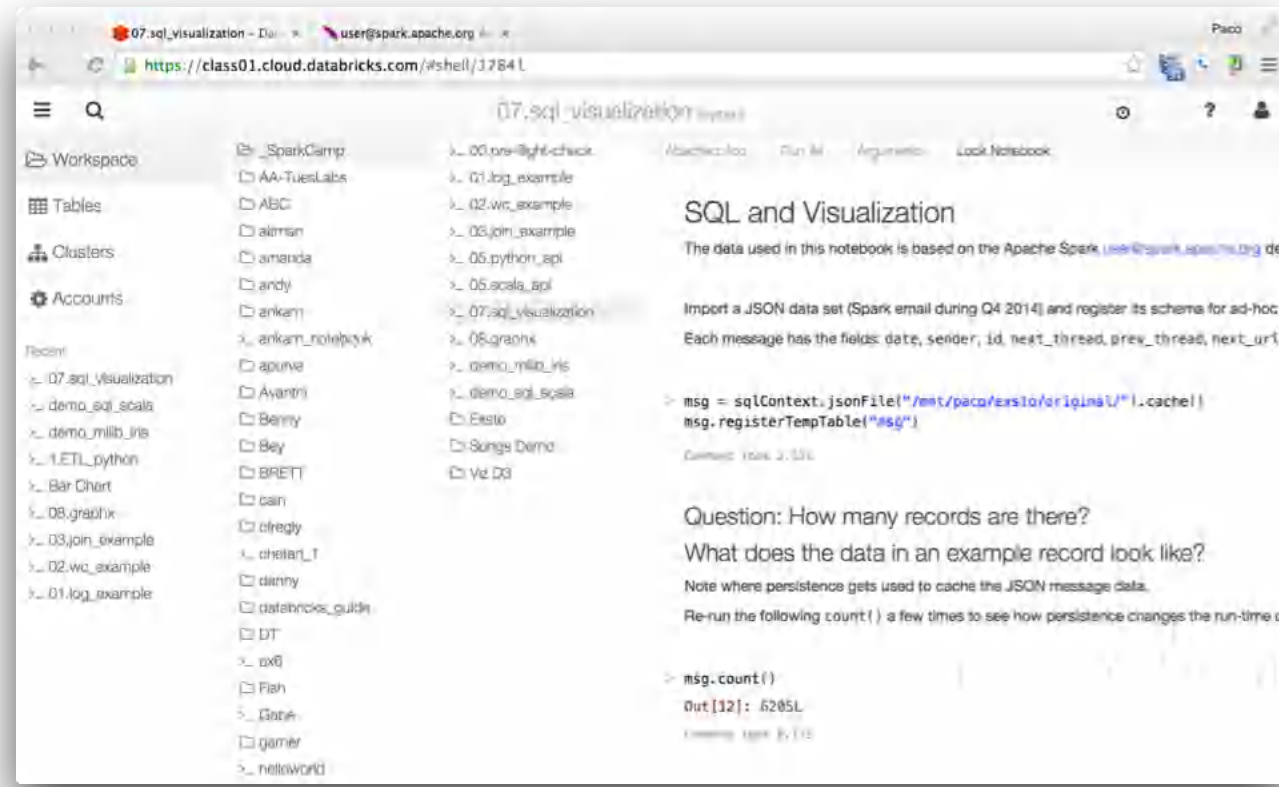
Demo: *D3 Visualization*

Clone the entire folder `/_SparkCamp/Viz D3` into your folder and run its notebooks:

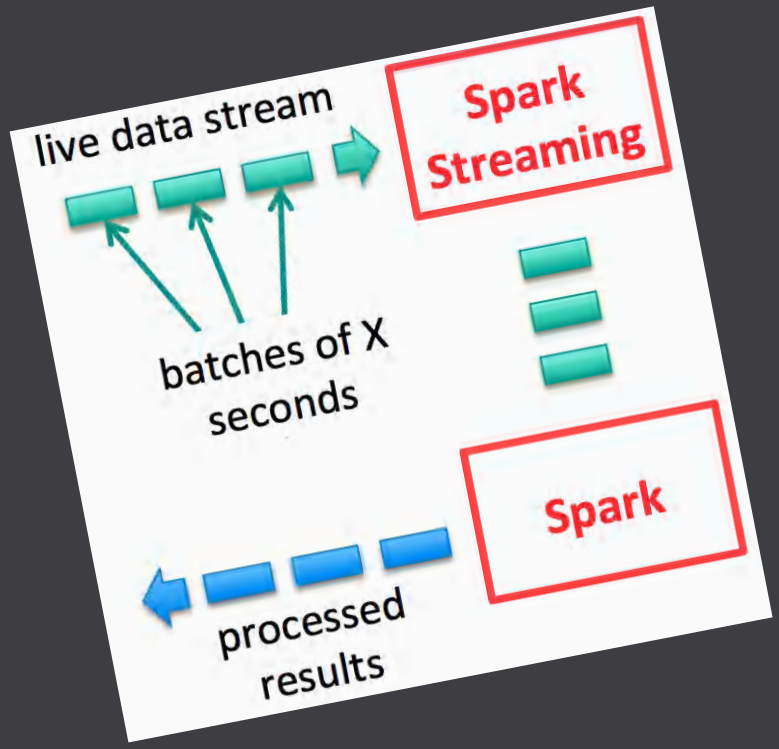


Coding Exercise: *SQL + Visualization*

Clone and run `/_SparkCamp/07.sql_visualization` in your folder:



Spark Streaming



Spark Streaming: *Requirements*

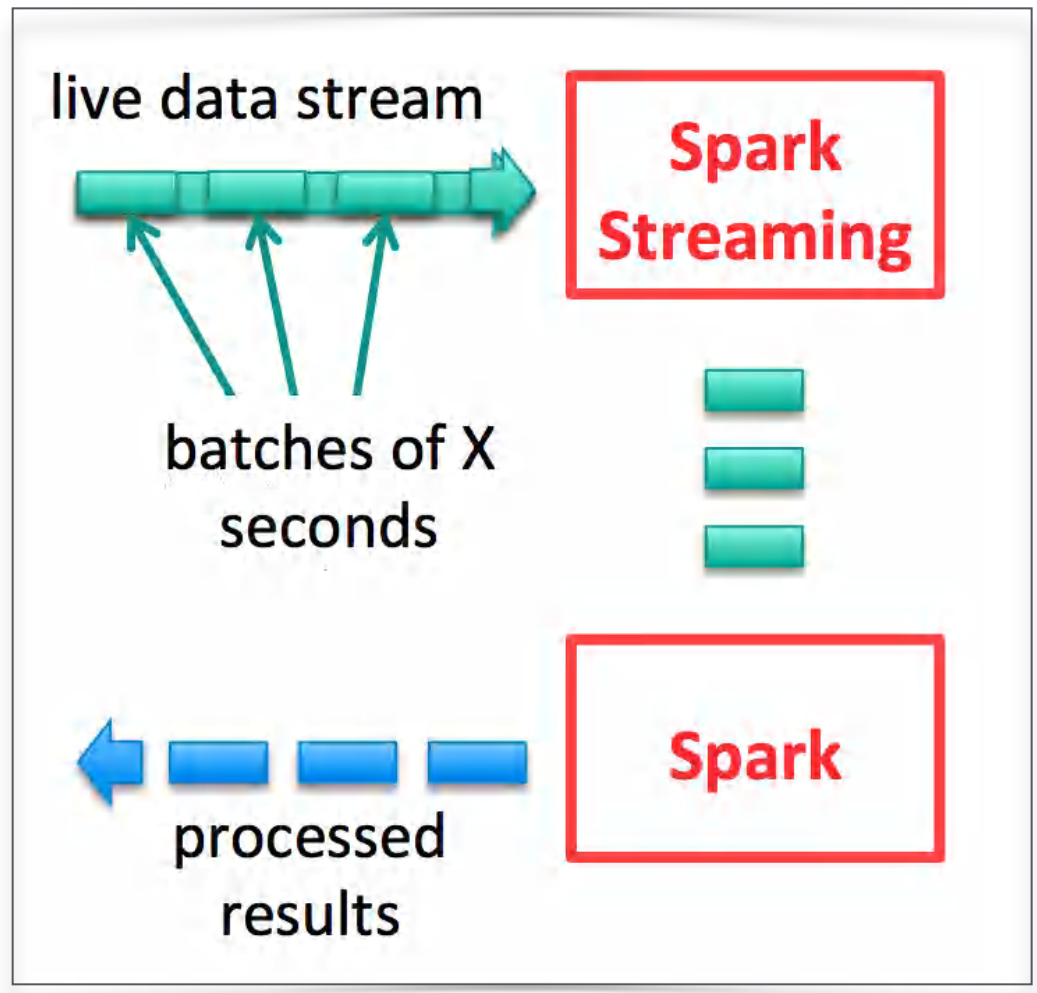
Let's consider the top-level requirements for a streaming framework:

- clusters scalable to 100's of nodes
- low-latency, in the range of seconds (meets 90% of use case needs)
- efficient recovery from failures (which is a hard problem in CS)
- integrates with batch: many co's run the same business logic both online+offline

Spark Streaming: *Requirements*

Therefore, run a streaming computation as:
a series of very small, deterministic batch jobs

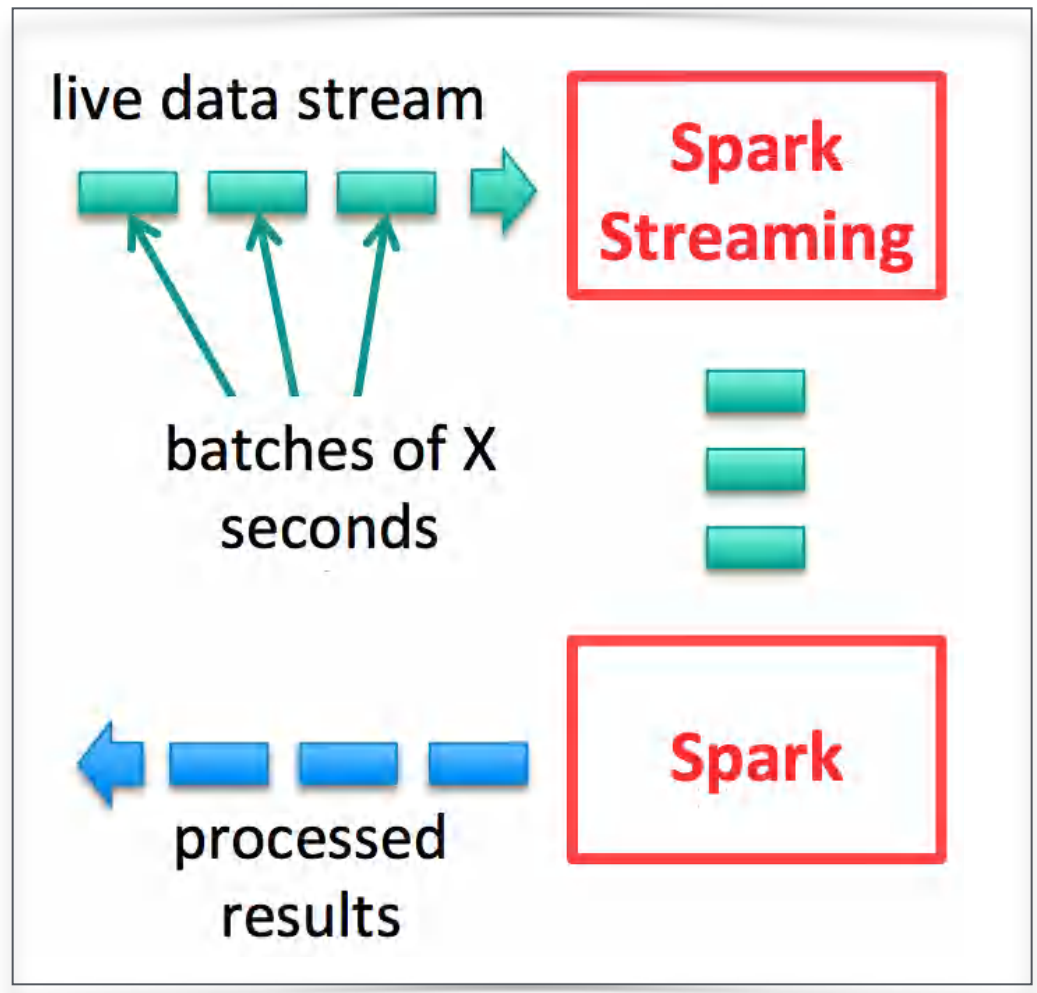
- *Chop up the live stream into batches of X seconds*
- *Spark treats each batch of data as RDDs and processes them using RDD operations*
- *Finally, the processed results of the RDD operations are returned in batches*



Spark Streaming: *Requirements*

Therefore, run a streaming computation as:
a series of very small, deterministic batch jobs

- *Batch sizes as low as ½ sec, latency of about 1 sec*
- *Potential for combining batch processing and streaming processing in the same system*



Spark Streaming: *Integration*

Data can be ingested from many sources:
Kafka, **Flume**, **Twitter**, **ZeroMQ**, TCP sockets, etc.

Results can be pushed out to filesystems, databases, live dashboards, etc.

Spark's built-in machine learning algorithms and graph processing algorithms can be applied to data streams

Spark Streaming: *Timeline*

2012 project started

2013 alpha release (Spark 0.7)

2014 graduated (Spark 0.9)

Discretized Streams: A Fault-Tolerant Model for Scalable Stream Processing
Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, Ion Stoica
Berkeley EECS (2012-12-14)
www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-259.pdf

project lead:
Tathagata Das **@tathadas**

Spark Streaming: *Use Cases*

Typical kinds of applications:

- *datacenter operations*
- *web app funnel metrics*
- *ad optimization*
- *anti-fraud*
- *telecom*
- *video analytics*
- *various telematics*

and much much more!

Spark Streaming: *Some Excellent Resources*

Programming Guide
spark.apache.org/docs/latest/streaming-programming-guide.html

TD @ Spark Summit 2014
youtu.be/o-NXwFrNAWQ?list=PLTPXxbhUt-YWGNTaDj6HSjnHMxiTD1HCR

"Deep Dive into Spark Streaming"
slideshare.net/spark-project/deep-divewithsparkstreaming-tathagatadassparkmeetup20130617

Spark Reference Applications
databricks.gitbooks.io/databricks-spark-reference-applications/

Quiz: *name the bits and pieces…*

```
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._

// create a StreamingContext with a SparkConf configuration
val ssc = new StreamingContext(sparkConf, Seconds(10))

// create a DStream that will connect to serverIP:serverPort
val lines = ssc.socketTextStream(serverIP, serverPort)

// split each line into words
val words = lines.flatMap(_.split(" "))

// count each word in each batch
val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)

// print a few of the counts to the console
wordCounts.print()

ssc.start()
ssc.awaitTermination()
```

Demo: *PySpark Streaming Network Word Count*

```
import sys
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

sc = SparkContext(appName="PyStreamNWC", master="local[*]")
ssc = StreamingContext(sc, Seconds(5))

lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))

counts = lines.flatMap(lambda line: line.split(" ")) \
              .map(lambda word: (word, 1)) \
              .reduceByKey(lambda a, b: a+b)

counts.pprint()

ssc.start()
ssc.awaitTermination()
```

Demo: *PySpark Streaming Network Word Count - Stateful*

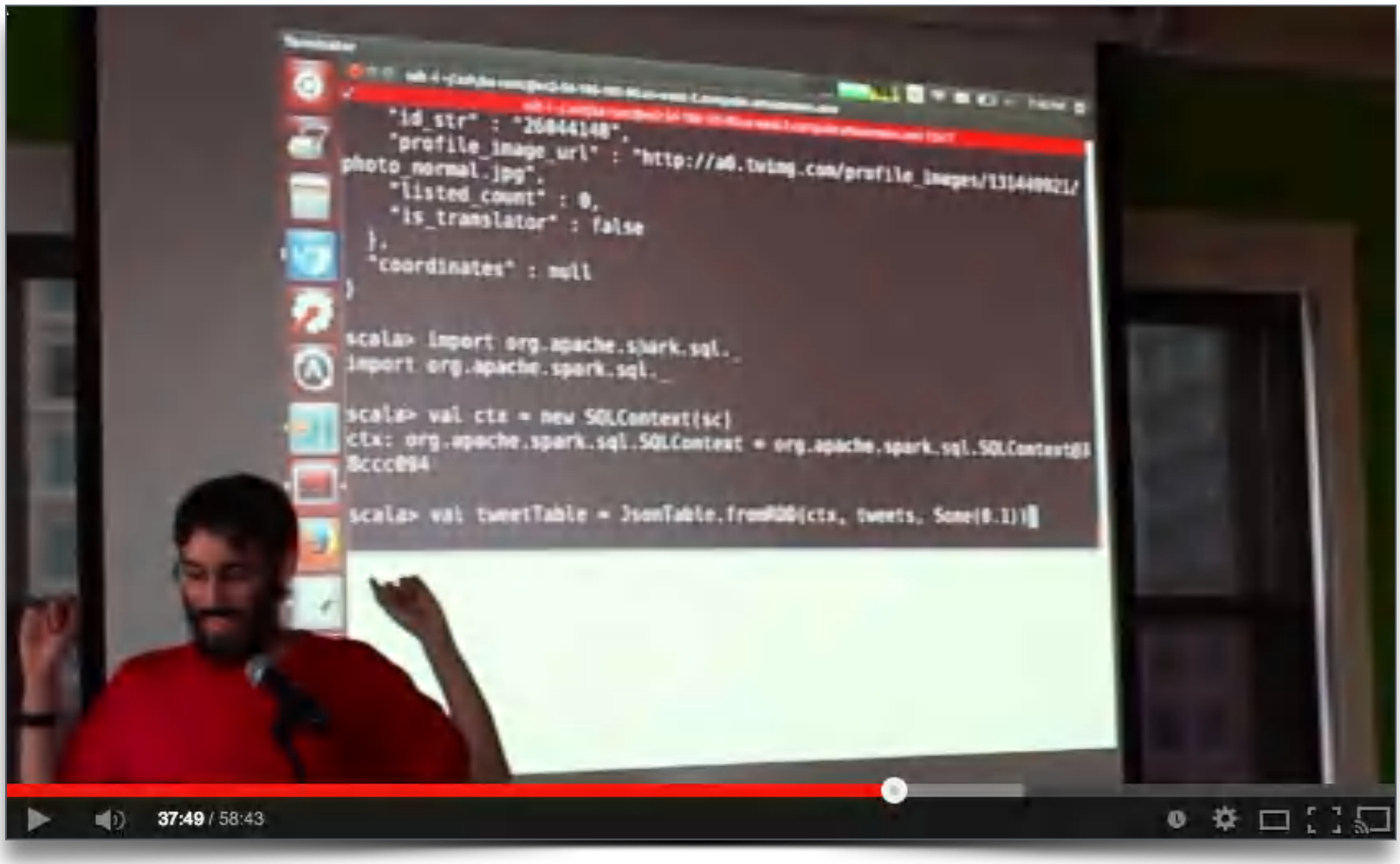
```
import sys
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

def updateFunc (new_values, last_sum):
    return sum(new_values) + (last_sum or 0)

sc = SparkContext(appName="PyStreamNWC", master="local[*]")
ssc = StreamingContext(sc, Seconds(5))
ssc.checkpoint("checkpoint")

lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))

counts = lines.flatMap(lambda line: line.split(" ")) \
              .map(lambda word: (word, 1)) \
              .updateStateByKey(updateFunc) \
              .transform(lambda x: x.sortByKey())

counts.pprint()

ssc.start()
ssc.awaitTermination()
```

Ref Apps:

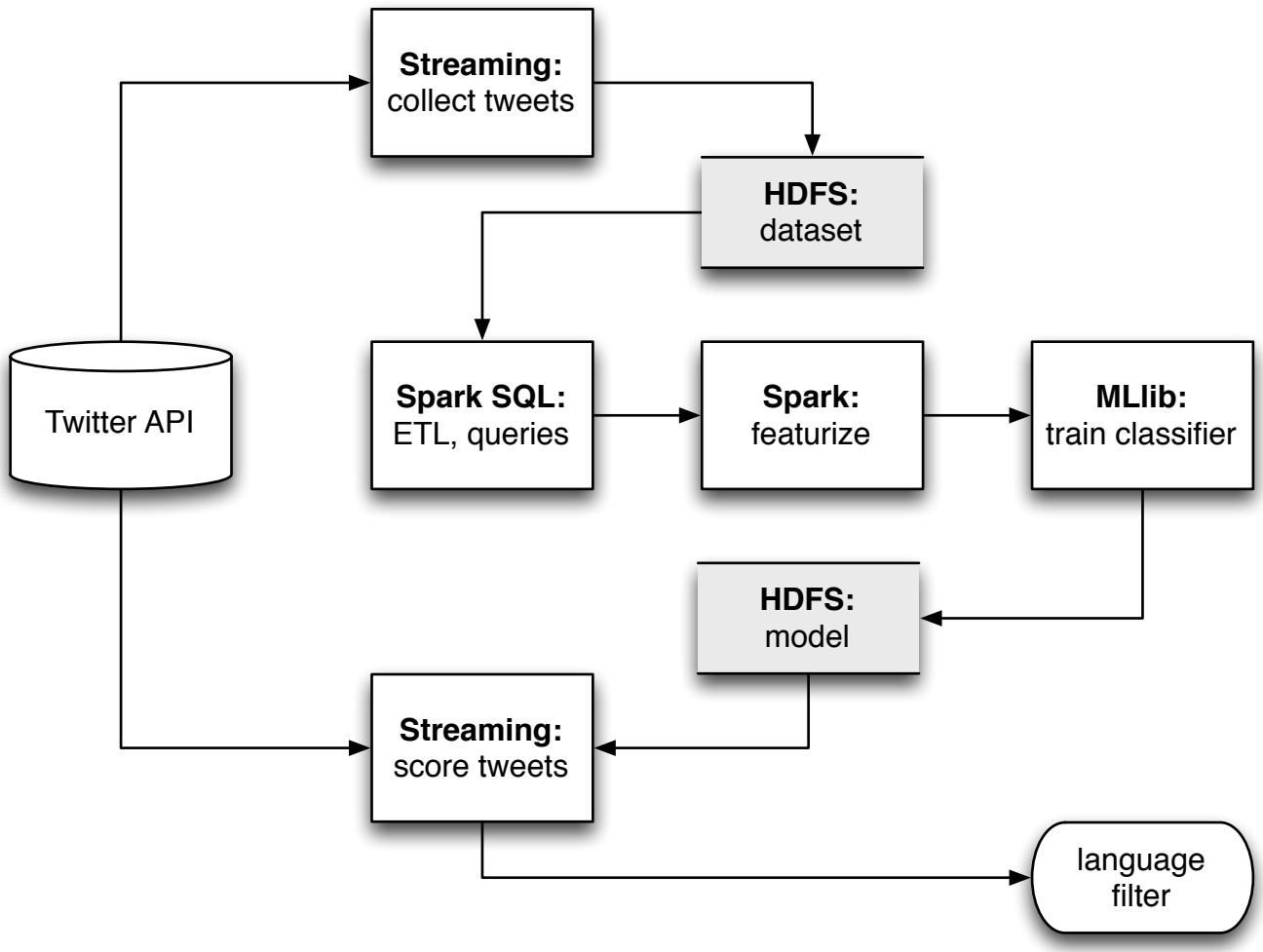
Twitter Streaming Demo

Demo: *Twitter Streaming Language Classifier*

databricks.gitbooks.io/databricks-spark-reference-applications/content/twitter_classifier/README.html



Demo: *Twitter Streaming Language Classifier*



Demo: *Twitter Streaming Language Classifier*

From tweets to ML features, approximated as sparse vectors:

1. extract text from the tweet	`https://twitter.com/andy_bf/status/16222269370011648`	`"Ceci n'est pas un tweet"`
2. sequence text as bigrams	`tweet.sliding(2).toSeq`	`("Ce", "ec", "ci", …, )`
3. convert bigrams into numbers	`seq.map(_.hashCode())`	`(2178, 3230, 3174, …, )`
4. index into sparse tf vector	`seq.map(_.hashCode() % 1000)`	`(178, 230, 174, …, )`
5. increment feature count	`Vector.sparse(1000, …)`	`(1000, [102, 104, …], [0.0455, 0.0455, …])`

Spark in Production: *Monitor*

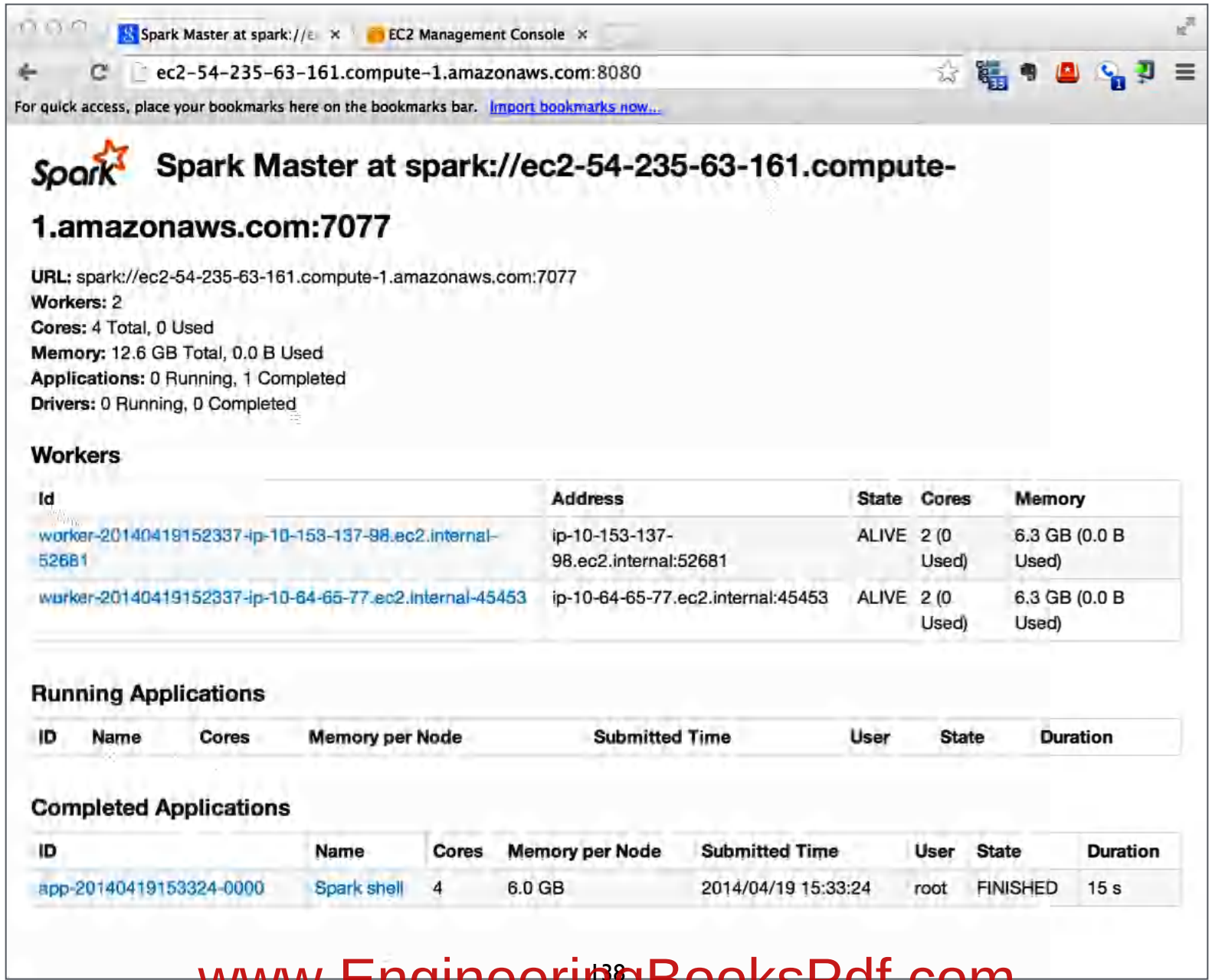
review UI features

spark.apache.org/docs/latest/monitoring.html

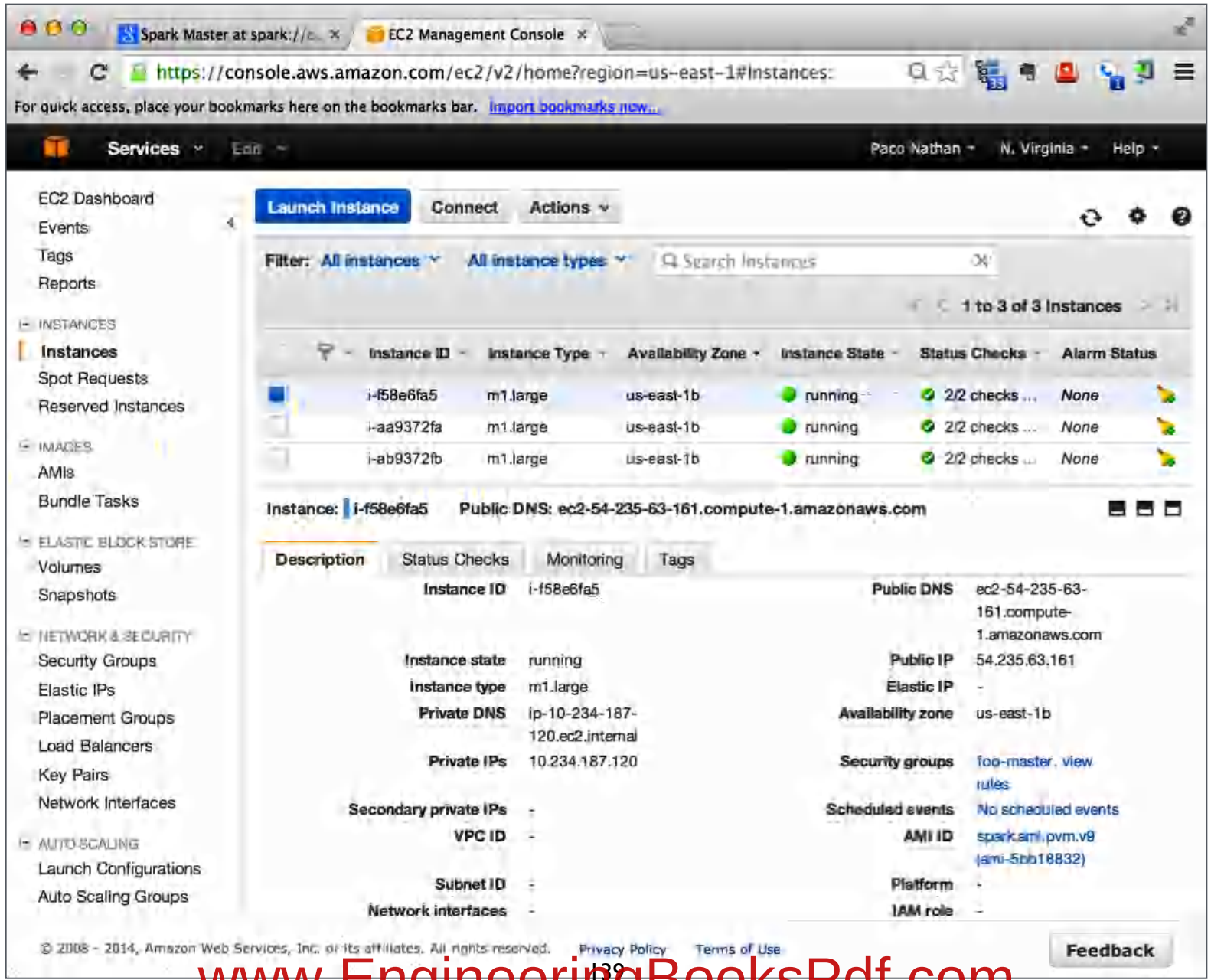
http://<master>:8080/
http://<master>:50070/

- verify: is my job still running?
- drill-down into *workers* and *stages*
- examine *stdout* and *stderr*
- discuss how to diagnose / troubleshoot

Spark in Production: *Monitor – Spark Console*



Spark in Production: *Monitor – AWS Console*



15 min break:

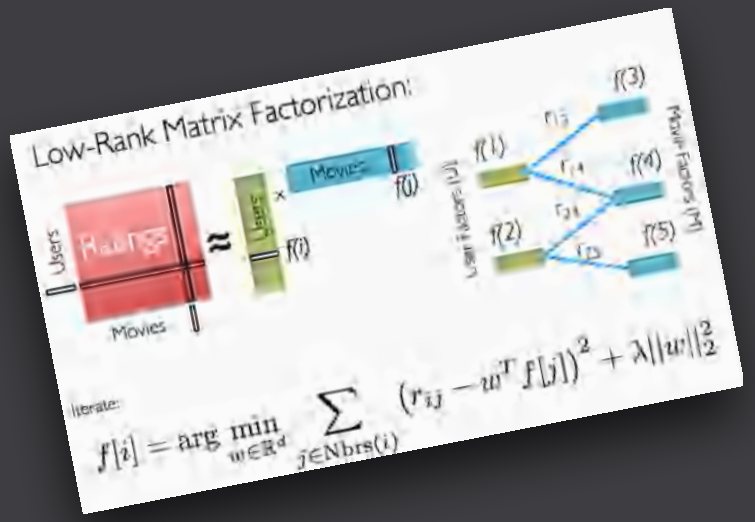
Post-Training Survey

We appreciate your feedback about this workshop. Please let us know how best to improve the material:

http://goo.gl/forms/pbRnPci7RZ

Also, if you'd like to sign-up for our monthly newsletter:
go.databricks.com/newsletter-sign-up

Intro to MLlib



MLlib: *Background…*

Distributing Matrix Computations with Spark MLlib
Reza Zadeh, Databricks
lintool.github.io/SparkTutorial/slides/day3_mllib.pdf

MLlib: Spark's Machine Learning Library
Ameet Talwalkar, Databricks
databricks-training.s3.amazonaws.com/slides/Spark_Summit_MLlib_070214_v2.pdf

Common Patterns and Pitfalls for Implementing Algorithms in Spark
Hossein Falaki, Databricks
lintool.github.io/SparkTutorial/slides/day1_patterns.pdf

Advanced Exercises: MLlib
databricks-training.s3.amazonaws.com/movie-recommendation-with-mllib.html

MLlib: *Background…*

spark.apache.org/docs/latest/mllib-guide.html

Key Points:

- framework vs. library
- *scale*, *parallelism*, *sparsity*
- building blocks for long-term approach
- see also: **Spark.ML**

MLlib: *Background…*

Components:

- scalable statistics
- classifiers, regression
- collab filters
- clustering
- matrix factorization
- feature extraction, normalizer
- optimization

MLlib: *Background…*

An excellent overview of ML definitions (up to this point) is given in:

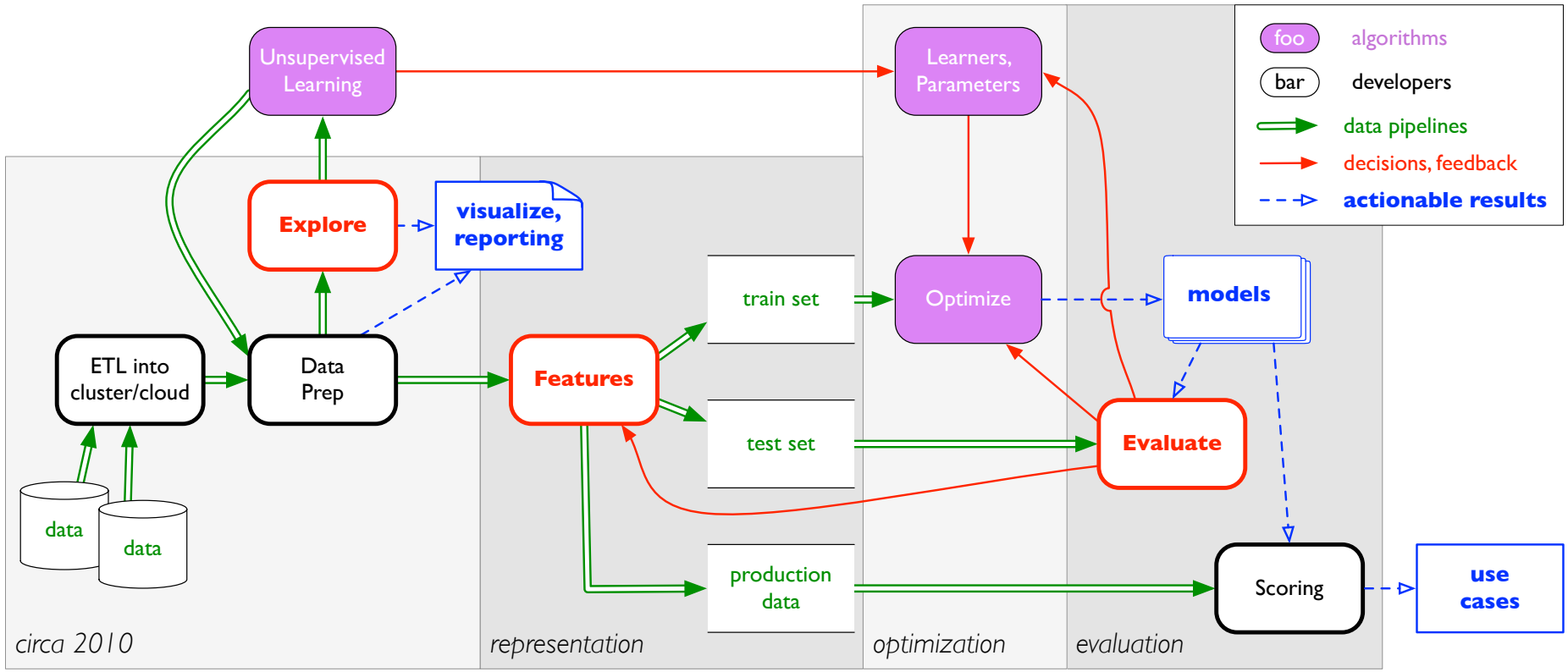
> *A Few Useful Things to Know about Machine Learning*
> **Pedro Domingos**
> CACM 55:10 (Oct 2012)
> http://dl.acm.org/citation.cfm?id=2347755

To wit:

Generalization = Representation + Optimization + Evaluation

MLlib: *Workflows*

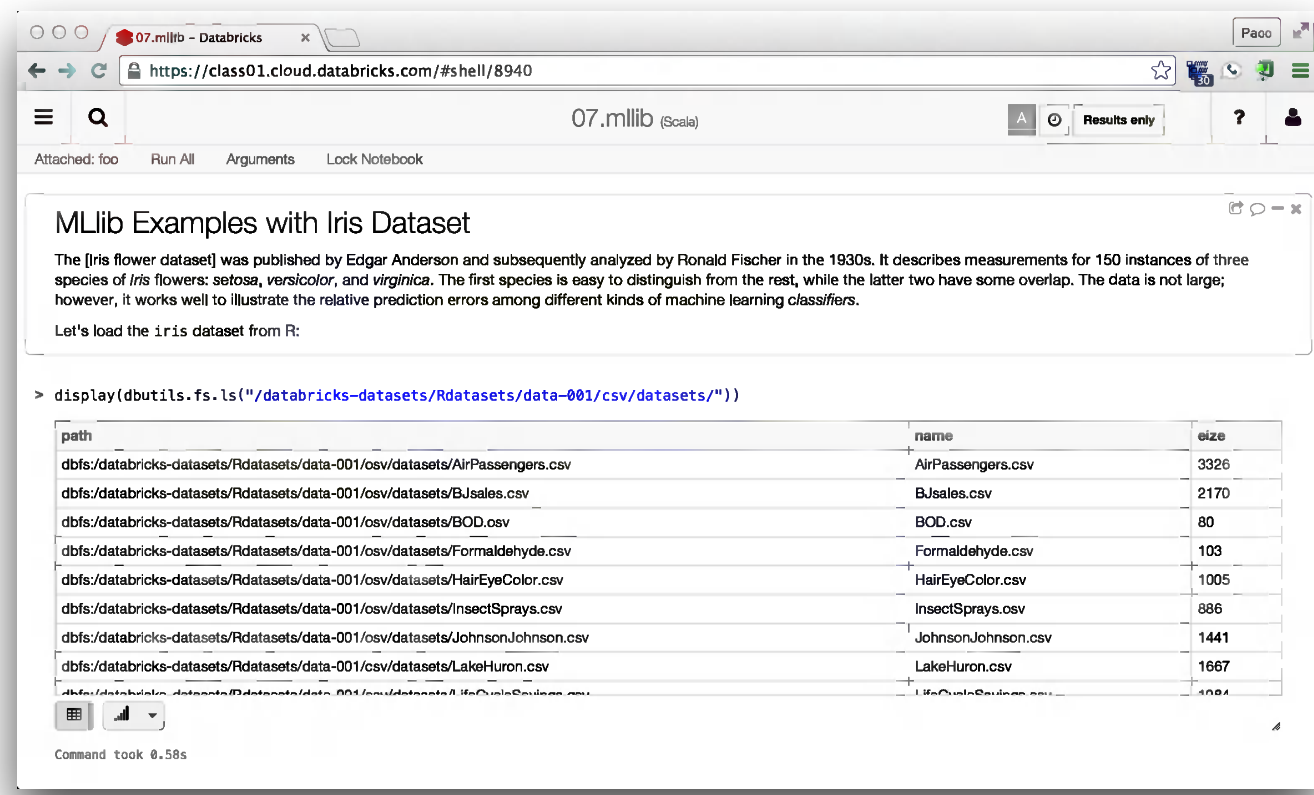
A generalized ML workflow looks like this…



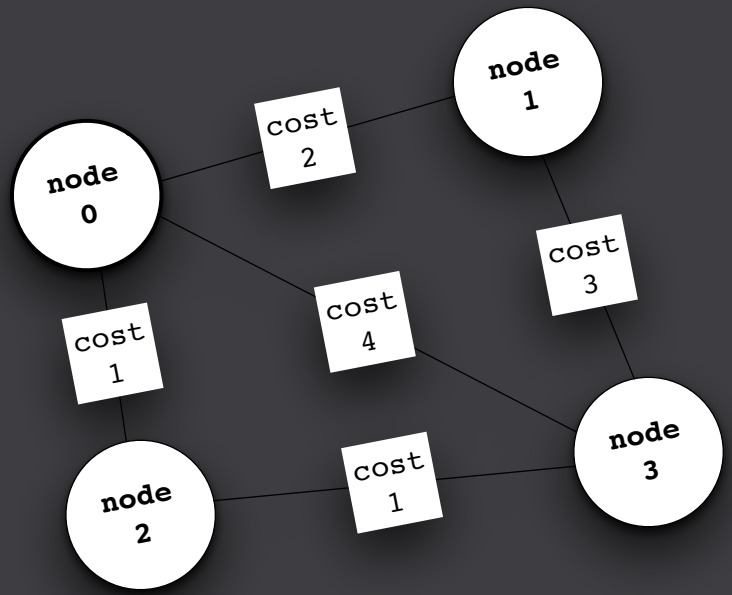
With results shown in **blue**, and the harder parts of this work highlighted in **red**

MLlib: *Code Exercise #7*

Clone and run `/_SparkCamp/demo_iris_mllib_1` in your folder:



GraphX examples



GraphX:

spark.apache.org/docs/latest/graphx-programming-guide.html

Key Points:

- graph-parallel systems
- importance of workflows
- optimizations

GraphX: *Further Reading…*

PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs
J. Gonzalez, Y. Low, H. Gu, D. Bickson, C. Guestrin
graphlab.org/files/osdi2012-gonzalez-low-gu-bickson-guestrin.pdf

Pregel: Large-scale graph computing at Google
Grzegorz Czajkowski, et al.
googleresearch.blogspot.com/2009/06/large-scale-graph-computing-at-google.html

GraphX: Unified Graph Analytics on Spark
Ankur Dave, Databricks
databricks-training.s3.amazonaws.com/slides/graphx@sparksummit_2014-07.pdf

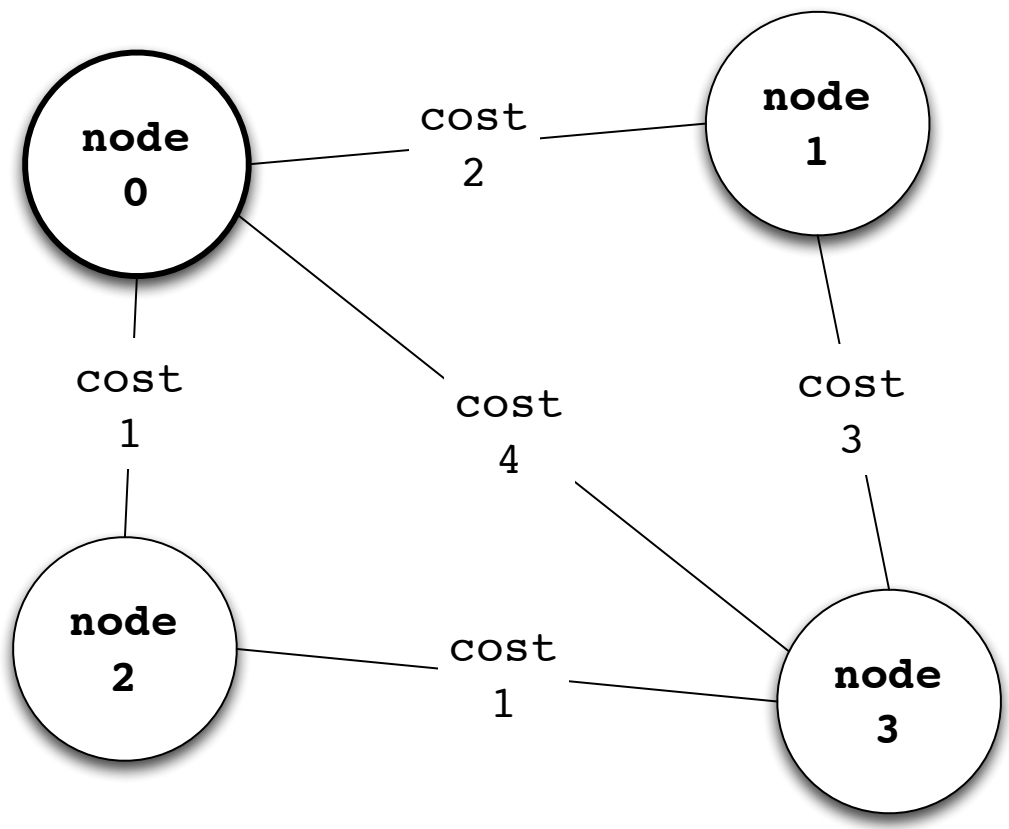
Advanced Exercises: GraphX
databricks-training.s3.amazonaws.com/graph-analytics-with-graphx.html

GraphX: *Example – simple traversals*

```
// http://spark.apache.org/docs/latest/graphx-programming-guide.html

import org.apache.spark.graphx._
import org.apache.spark.rdd.RDD

case class Peep(name: String, age: Int)

val nodeArray = Array(
  (1L, Peep("Kim", 23)), (2L, Peep("Pat", 31)),
  (3L, Peep("Chris", 52)), (4L, Peep("Kelly", 39)),
  (5L, Peep("Leslie", 45))
  )
val edgeArray = Array(
  Edge(2L, 1L, 7), Edge(2L, 4L, 2),
  Edge(3L, 2L, 4), Edge(3L, 5L, 3),
  Edge(4L, 1L, 1), Edge(5L, 3L, 9)
  )

val nodeRDD: RDD[(Long, Peep)] = sc.parallelize(nodeArray)
val edgeRDD: RDD[Edge[Int]] = sc.parallelize(edgeArray)
val g: Graph[Peep, Int] = Graph(nodeRDD, edgeRDD)

val results = g.triplets.filter(t => t.attr > 7)

for (triplet <- results.collect) {
  println(s"${triplet.srcAttr.name} loves ${triplet.dstAttr.name}")
}
```

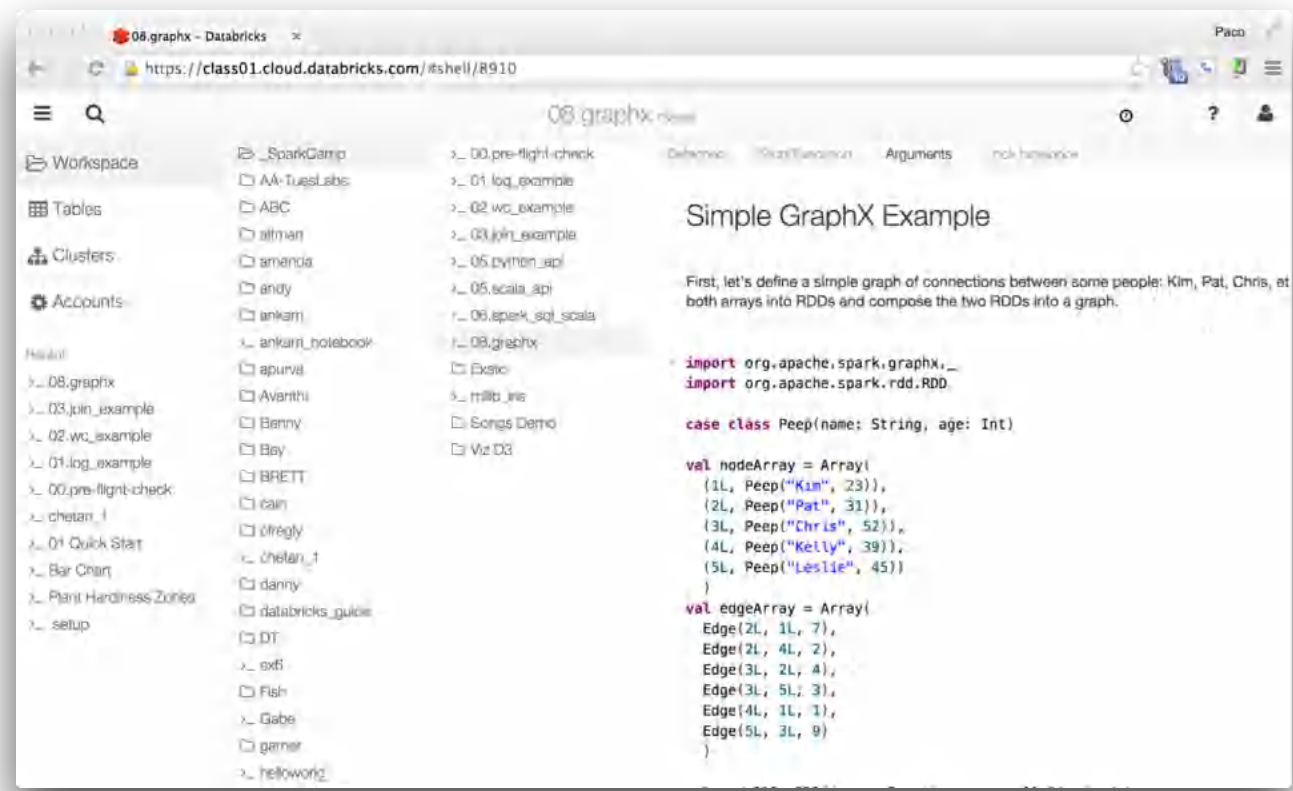
GraphX: *Example – routing problems*

What is the cost to reach **node 0** from any other node in the graph? This is a common use case for graph algorithms, e.g., **Djikstra**



GraphX: *Coding Exercise*

Clone and run `/_SparkCamp/08.graphx` in your folder:



Case Studies

Case Studies: *Apache Spark, DBC, etc.*

Additional details about production deployments for Apache Spark can be found at:

http://go.databricks.com/customer-case-studies

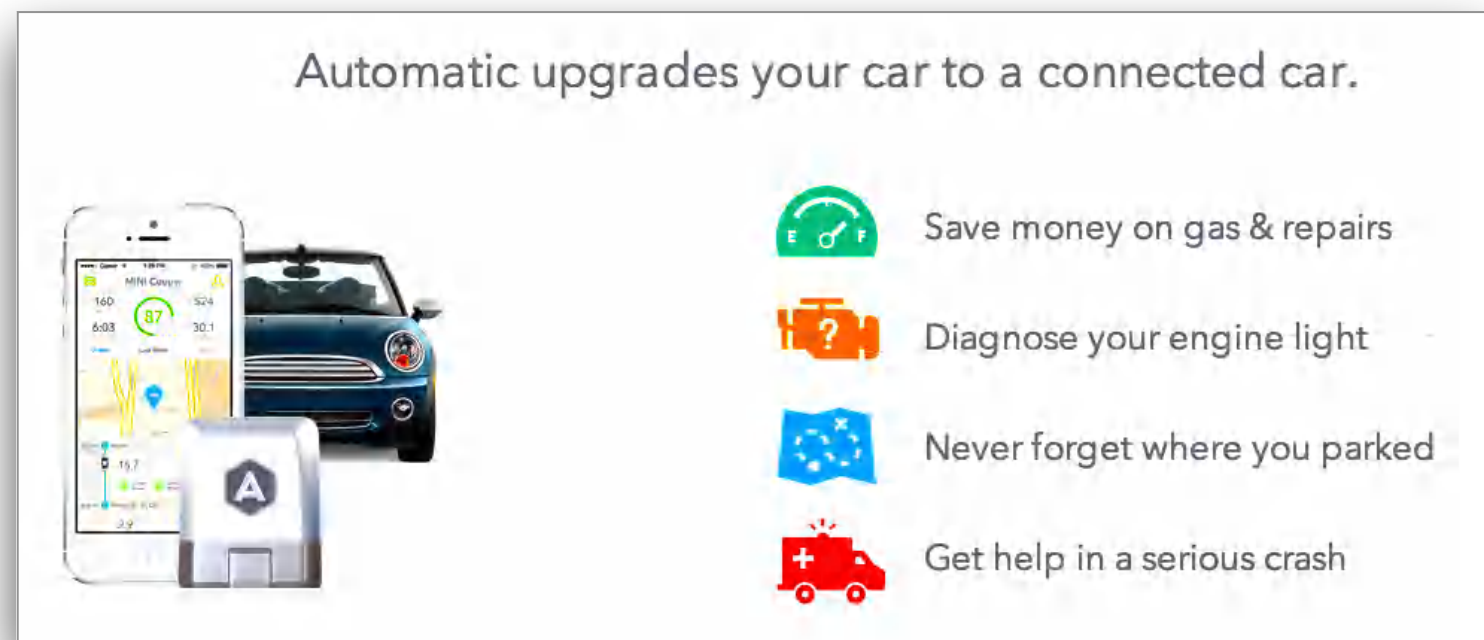
https://databricks.com/blog/category/company/partners

https://cwiki.apache.org/confluence/display/SPARK/Powered+By+Spark

Case Studies: *Automatic Labs*

http://automatic.com/

Detects events like hard braking, acceleration – uploaded in real-time with geolocation to a Spark Streaming pipeline … data trends indicate road hazards, blind intersections, bad signal placement, and other input to improve traffic planning. Also detects inefficient vehicle operation, under-inflated tires, poor driving behaviors, aggressive acceleration, etc.



Case Studies: *Automatic Labs*

Databricks Use Case:
Creates personalized driving habit dashboards

Business Overview:
Offering a dashboard for customers based on their driving habits, with information collected from GPS and other sensors built into the car.

Potentially for a wide array of data products:

- incentivizing good driving patterns
- performing preventative maintenance
- identifying patterns of failure

Case Studies: *Automatic Labs*

Challenges Encountered:

- *Infrastructure management difficulties:* wanted to use Spark while minimizing investment in DevOps
- *Lack of data accessibility:* needed to provide data access to non-technical analysts via SQL

Case Studies: *Automatic Labs*

AUTOMATIC

Databricks Impact:

- *Reduce complexity:*
  replaced Redshift and disparate ML tools with a single platform
- *Facilitate BI:*
  leveraged built-in visualization capabilities in Notebooks to generate dashboards easily and quickly
- *Jumpstart advanced analytics:*
  used MLlib on Spark for the needed functionality out of the box

Case Studies: *Automatic Labs*

AUTOMATIC

Talks:

Spark Plugs Into Your Car

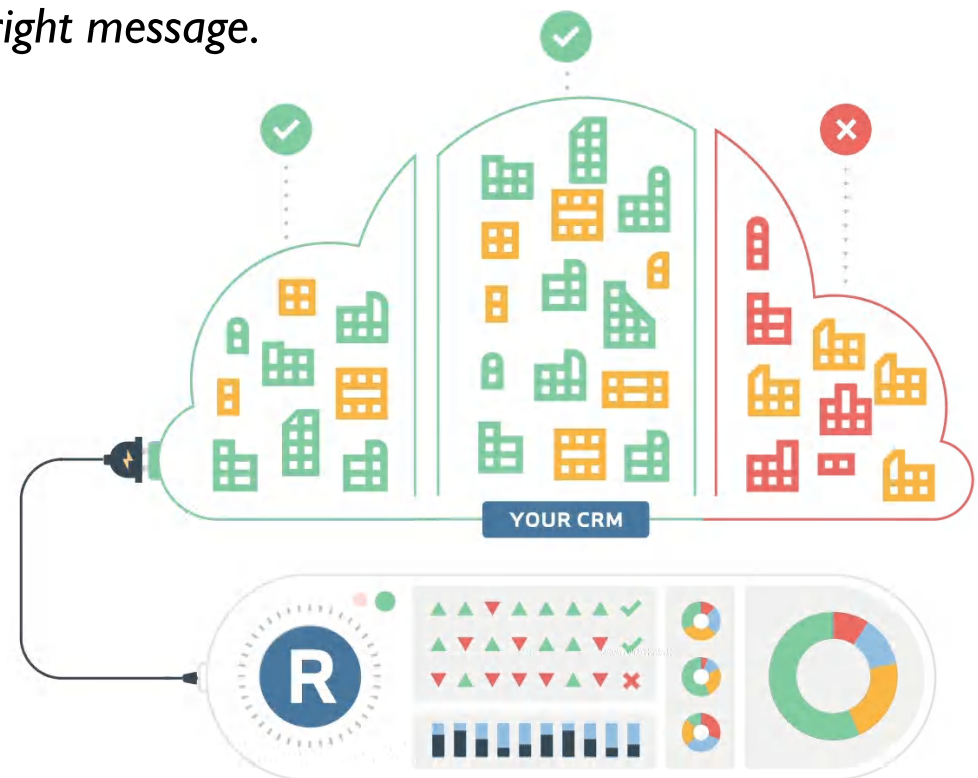
Rob Ferguson

spark-summit.org/east/2015/talk/spark-plugs-into-your-car

Case Studies: *Radius Intelligence*

http://radius.com/

By connecting your CRM to Radius, you can instantly visualize where you've historically had success and identify the top market segments to pursue next. Radius surfaces in-depth signals that go beyond industry category with social media activity, web presence, advertising spend and more, so that you can reach the right audience with the right message.



Case Studies: *Radius Intelligence*

Databricks Use Case:
Combs the web for business intelligence

Business Overview:
Provides SMB market intelligence by collecting and synthesizing information automatically from many different data sources ranging from news outlets to social networks

Data Science expertise creates competitive advantage against entrenched competitors who rely on less efficient or manual processes

Challenges Encountered:

- *Data pipeline too slow:*
  building a full SMB index takes 12+ hours using Hadoop and Cascading
- *Pipeline complexity:*
  difficult to change or enhance existing data pipeline
- *Lack of data accessibility:*
  non-technical personnel cannot make use of Big Data without writing code

Case Studies: *Radius Intelligence*

Databricks Impact:

- *Improve pipeline performance:*
  Spark increased pipeline performance 10x

- *Higher productivity:*
  interactive shell and notebooks enabled data scientists to experiment and develop code faster

- *Increase Big Data accessibility:*
  PMs and business development staff can use SQL to query large data sets

Case Studies: *Radius Intelligence*

Talks:

From Hadoop to Spark in 4 months, Lessons Learned

Alexis Roos

http://youtu.be/o3-lokUFqvA

Case Studies: *MyFitnessPal*

http://www.myfitnesspal.com/

We believe – and medical studies prove – that the best way to lose weight and keep it off is to simply keep track of the foods you eat. Gimmicky machines and fad diets don't work, so we designed a free website and mobile apps that make calorie counting and food tracking easy.

Case Studies: *MyFitnessPal*

Databricks Use Case:
Builds integrated offline analysis platform

Business Overview:
Mobile app and website for people to track, learn, communicate and improve their health – also one of the largest ad placement venues for fitness related ads

Data Science expertise increases user engagement and monetizes traffic:

- show people how they rank within categories such as geographic region and age
- recommend diets and exercises

Case Studies: *MyFitnessPal*

Challenges Encountered:

- *High pipeline complexity:*
  data pipeline had many ad hoc scripts and required many different software packages
- *Long data product development cycle:*
  needed a single tool to help create machine learning data products more rapidly

Case Studies: *MyFitnessPal*

Databricks Impact:

- *Reduced complexity:*
  provide the capabilities of three disparate technologies (LIBNEAR, Redshift, Tableau) within a single platform

- *Jumpstart advanced analytics:*
  used MLlib on Spark for the needed functionality out of the box

Case Studies: *MyFitnessPal*

Talks:

SILK: A Spark Based Data Pipeline to Construct a Reliable and Accurate Food Dataset

Hesamoddin Salehian

spark-summit.org/east/2015/talk/silk-a-spark-based-data-pipeline-to-construct-a-reliable-and-accurate-food-dataset

Case Studies: *Twitter*

Spark at Twitter: Evaluation & Lessons Learnt
Sriram Krishnan
slideshare.net/krishflix/seattle-spark-meetup-spark-at-twitter

- Spark can be more interactive, efficient than MR
  - *support for iterative algorithms and caching*
  - *more generic than traditional MapReduce*
- Why is Spark faster than Hadoop MapReduce?
  - *fewer I/O synchronization barriers*
  - *less expensive shuffle*
  - *the more complex the DAG, the greater the performance improvement*

Case Studies: *Spotify*

Collaborative Filtering with Spark

Chris Johnson

slideshare.net/MrChrisJohnson/collaborative-filtering-with-spark

- *collab filter (ALS) for music recommendation*
- *Hadoop suffers from I/O overhead*
- *show a progression of code rewrites, converting a Hadoop-based app into efficient use of Spark*

Case Studies: *Stratio*

Stratio Streaming: a new approach to Spark Streaming
David Morales, **Oscar Mendez**
2014-06-30
spark-summit.org/2014/talk/stratio-streaming-a-new-approach-to-spark-streaming

- Stratio Streaming is the union of a real-time messaging bus with a complex event processing engine using Spark Streaming
- allows the creation of streams and queries on the fly
- paired with Siddhi CEP engine and Apache Kafka
- added global features to the engine such as auditing and statistics

Case Studies: *Pearson*

Pearson uses Spark Streaming for next generation adaptive learning platform
Dibyendu Bhattacharya
2014-12-08
databricks.com/blog/2014/12/08/pearson-uses-spark-streaming-for-next-generation-adaptive-learning-platform.html

- Kafka + Spark + Cassandra + Blur, on AWS on a YARN cluster
- single platform/common API was a key reason to replace Storm with Spark Streaming
- custom Kafka Consumer for Spark Streaming, using Low Level Kafka Consumer APIs
- handles: Kafka node failures, receiver failures, leader changes, committed offset in ZK, tunable data rate throughput

Case Studies: *Guavus*

Guavus Embeds Apache Spark
into its Operational Intelligence Platform
Deployed at the World's Largest Telcos
Eric Carr
2014-09-25
databricks.com/blog/2014/09/25/guavus-embeds-apache-spark-into-its-operational-intelligence-platform-deployed-at-the-worlds-largest-telcos.html

- 4 of 5 top mobile network operators, 3 of 5 top Internet backbone providers, 80% MSOs in NorAm
- analyzing 50% of US mobile data traffic, +2.5 PB/day
- latency is critical for resolving operational issues before they cascade: 2.5 MM transactions per second
- "analyze first" not "store first ask questions later"

Case Studies: *Sharethrough*

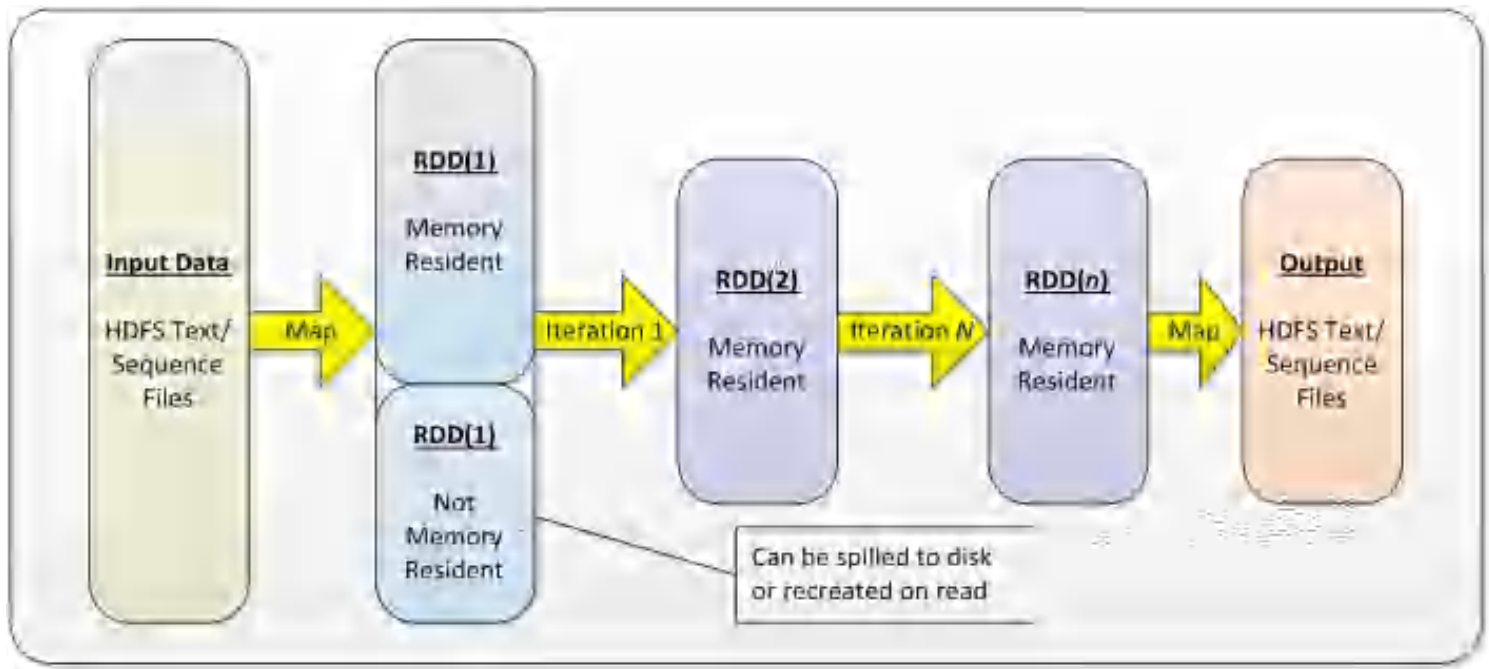
Sharethrough Uses Spark Streaming to Optimize Bidding in Real Time
Russell Cardullo, **Michael Ruggier**
2014-03-25
databricks.com/blog/2014/03/25/sharethrough-and-spark-streaming.html

- the profile of a 24 x 7 streaming app is different than an hourly batch job…
- take time to validate output against the input…
- confirm that supporting objects are being serialized…
- the output of your Spark Streaming job is only as reliable as the queue that feeds Spark…
- monoids…

Case Studies: *eBay*

Using Spark to Ignite Data Analytics

ebaytechblog.com/2014/05/28/using-spark-to-ignite-data-analytics/



Further Resources + Q&A

Spark Developer Certification

- go.databricks.com/spark-certified-developer
- defined by Spark experts @Databricks
- assessed by O'Reilly Media
- establishes the bar for Spark expertise

Developer Certification: *Overview*

- 40 multiple-choice questions, 90 minutes
- mostly structured as choices among code blocks
- expect some Python, Java, Scala, SQL
- understand theory of operation
- identify best practices
- recognize code that is more parallel, less memory constrained

Overall, you need to write Spark apps in practice

community:

spark.apache.org/community.html

events worldwide: goo.gl/2YqJZK

YouTube channel: goo.gl/N5Hx3h

video+preso archives: spark-summit.org

resources: databricks.com/spark/developer-resources

workshops: databricks.com/spark/training

MOOCs:

Anthony Joseph
UC Berkeley
begins Jun 2015
edx.org/course/uc-berkeleyx/uc-berkeleyx-cs100-1x-introduction-big-6181

Introduction to Big Data with Apache Spark

Learn how to apply data science techniques using parallel programming in Apache Spark to explore big (and small) data.

Scalable Machine Learning

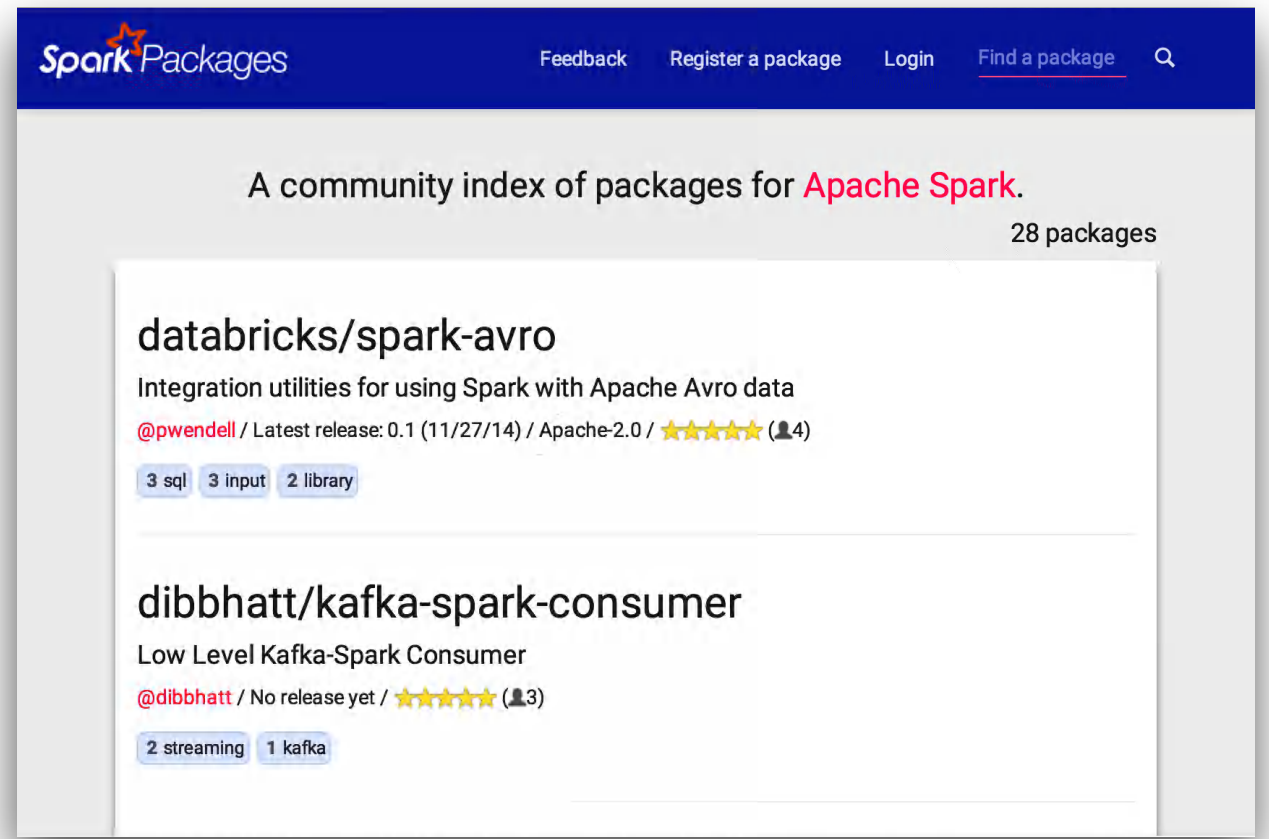
Learn the underlying principles required to develop scalable machine learning pipelines and gain hands-on experience using Apache Spark.

Ameet Talwalkar
UCLA
begins Jun 2015
edx.org/course/uc-berkeleyx/uc-berkeleyx-cs190-1x-scalable-machine-6066

Resources: *Spark Packages*

Looking for other libraries and features? There are a variety of third-party packages available at:

http://spark-packages.org/



confs:

Scala Days
Amsterdam, Jun 8
scaladays.org/

Spark Summit SF
SF, Jun 15
spark-summit.org/2015/

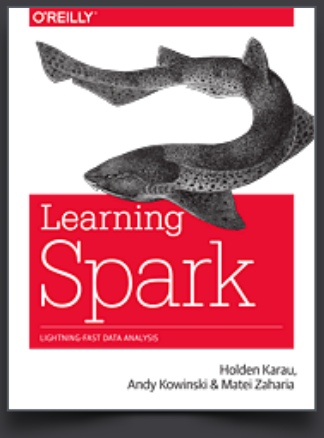
Strata NY
NYC, Sep 29
strataconf.com/big-data-conference-ny-2015

Spark Summit EU
Amsterdam, Oct 27
spark-summit.org

Strata SG
Singapore, Dec 2
strataconf.com/big-data-conference-sg-2015

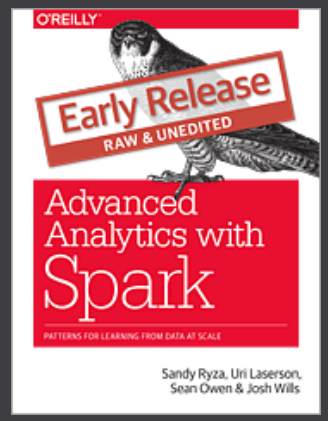
books+videos:

Learning Spark
Holden Karau, Andy Konwinski, Parick Wendell, Matei Zaharia
O'Reilly (2015)
shop.oreilly.com/product/06369200285l2.do



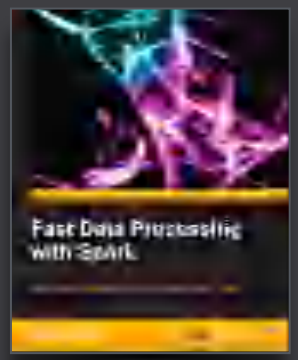
Intro to Apache Spark
Paco Nathan
O'Reilly (2015)
shop.oreilly.com/product/0636920036807.do

Advanced Analytics with Spark
Sandy Ryza, Uri Laserson, Sean Owen, Josh Wills
O'Reilly (2014)
shop.oreilly.com/product/0636920035091.do



Spark in Action
Chris Fregly
Manning (2015)
sparkinaction.com/

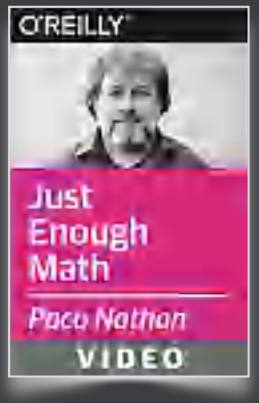
Fast Data Processing with Spark
Holden Karau
Packt (2013)
shop.oreilly.com/product/9781782167068.do



presenter:

monthly newsletter for updates, events, conf summaries, etc.:

liber118.com/pxn/



Just Enough Math
O'Reilly, 2014

justenoughmath.com
preview: youtu.be/TQ58cWgdCpA

Enterprise Data Workflows with Cascading
O'Reilly, 2013

shop.oreilly.com/product/0636920028536.do

CHICAGO
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE 2015

goto; conference

Intro to Apache Spark

Paco Nathan

follow us @gotochgo

Conference: May 11-13 / Workshops: 13-14