Spark Tutorial @ DAO

download slides:
training.databricks.com/workshop/su_dao.pdf

Welcome + Getting Started

Getting Started: *Step 1*

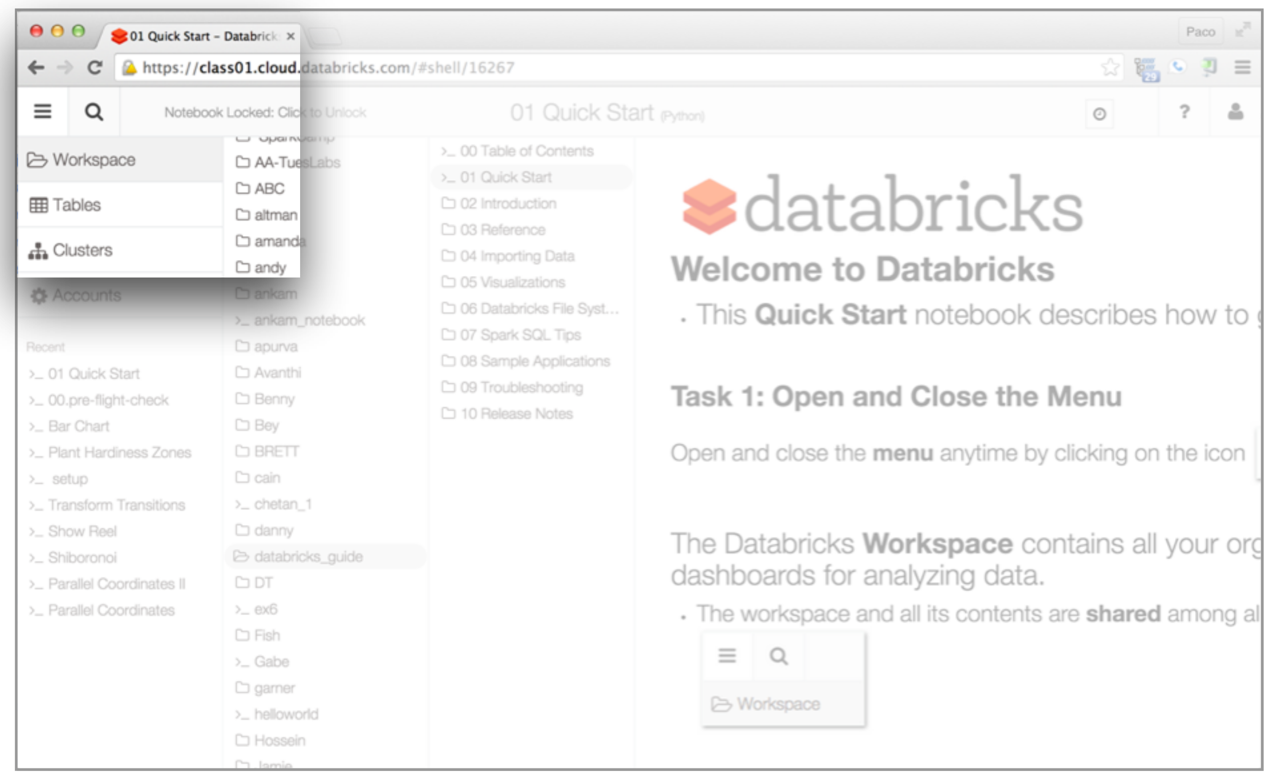
Everyone will receive a username/password for one of the Databricks Cloud *shards*. Use your laptop and browser to login there.

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.

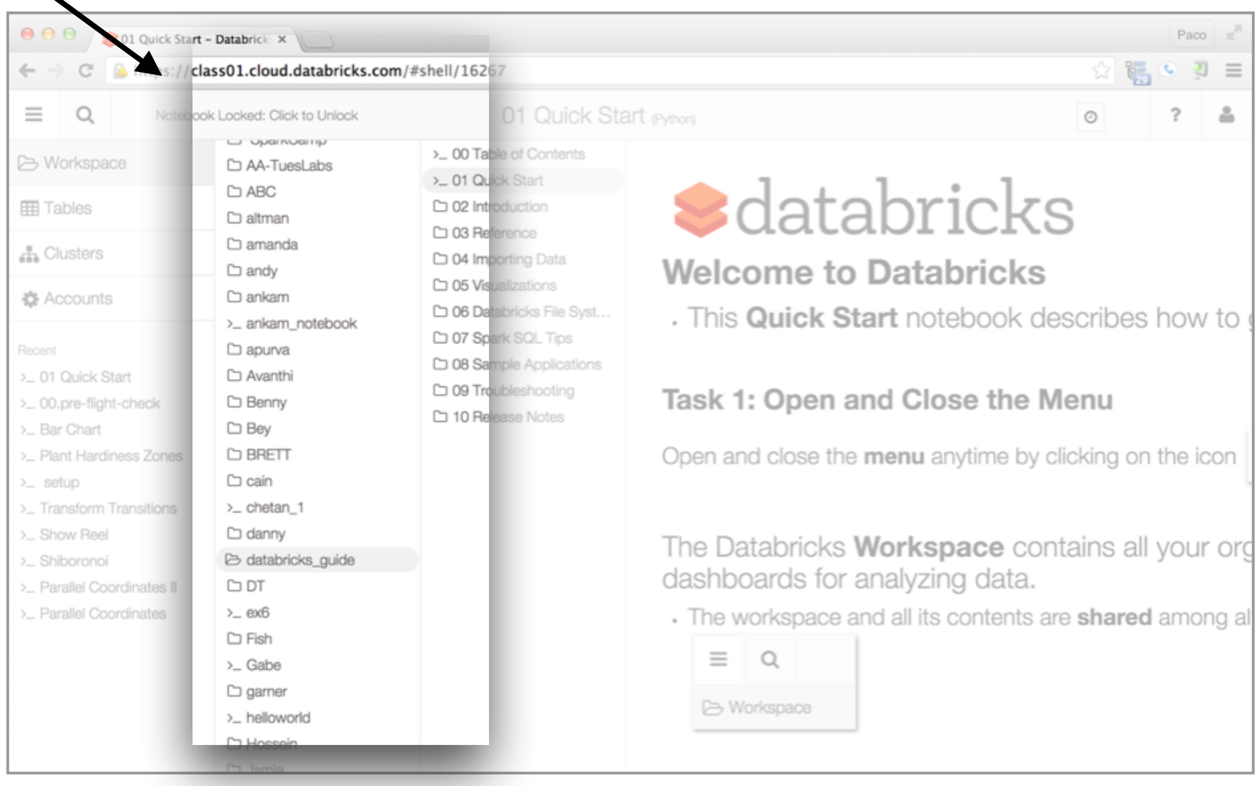
Getting Started: *Step 2*

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



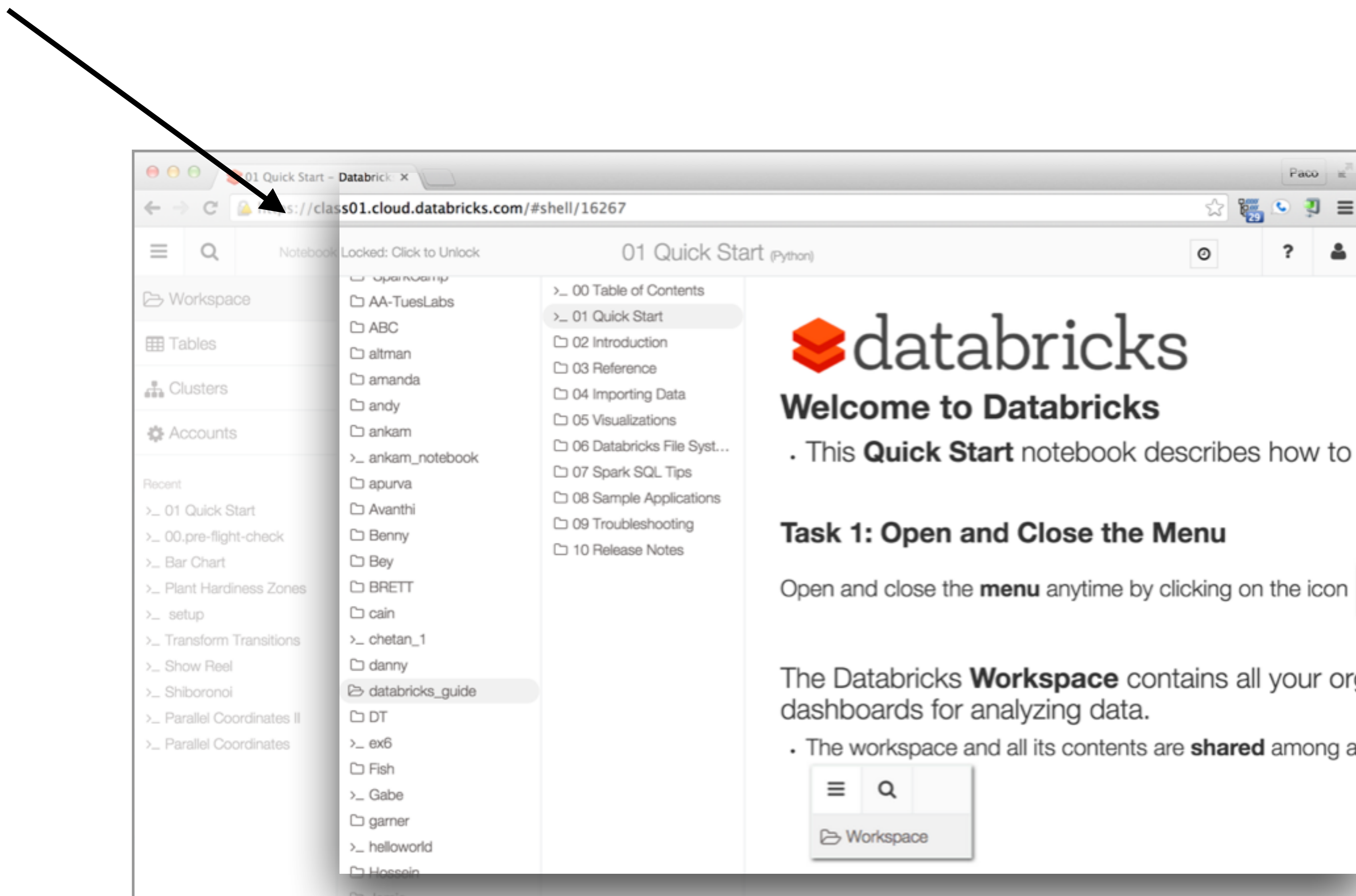
Getting Started: *Step 3*

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:

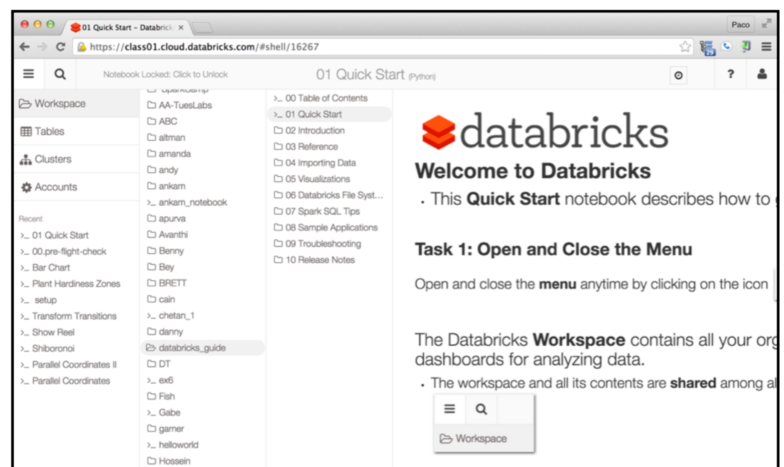


Getting Started: *Step 5*

See `/databricks-guide/01 Quick Start`

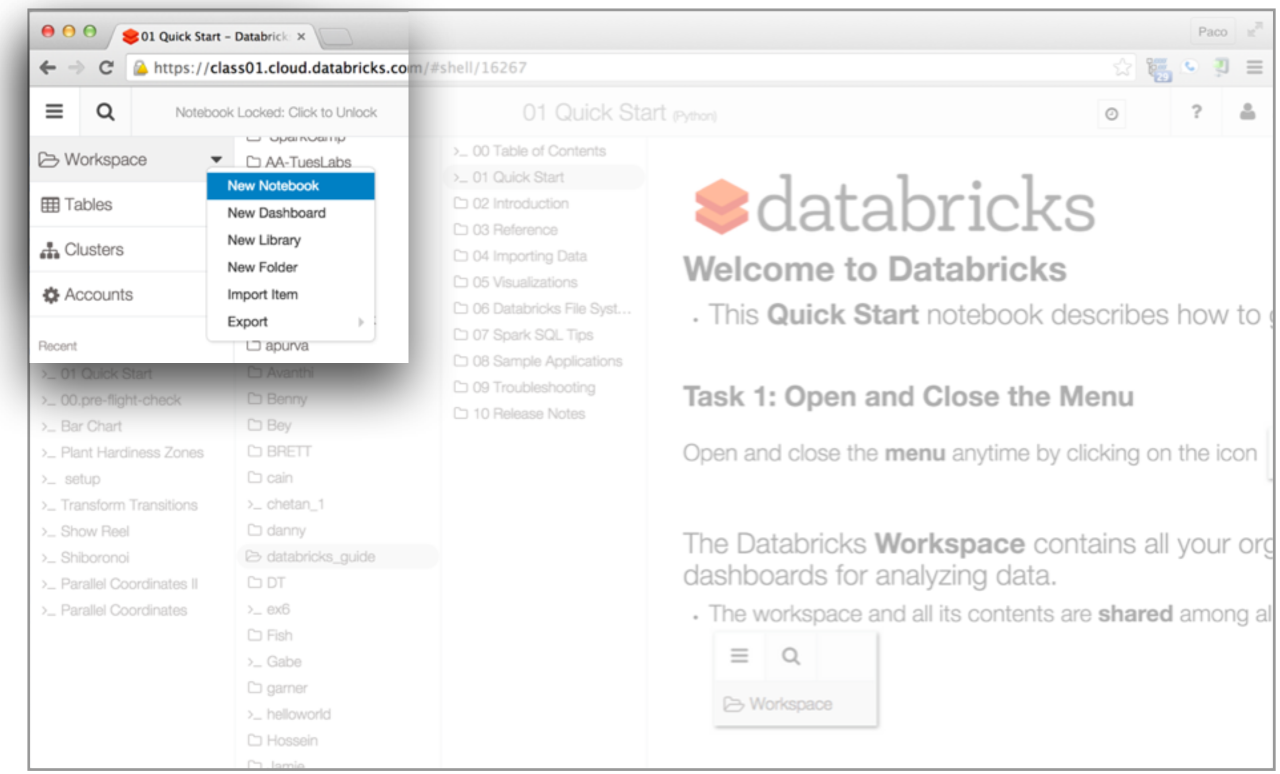
Key Features:

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



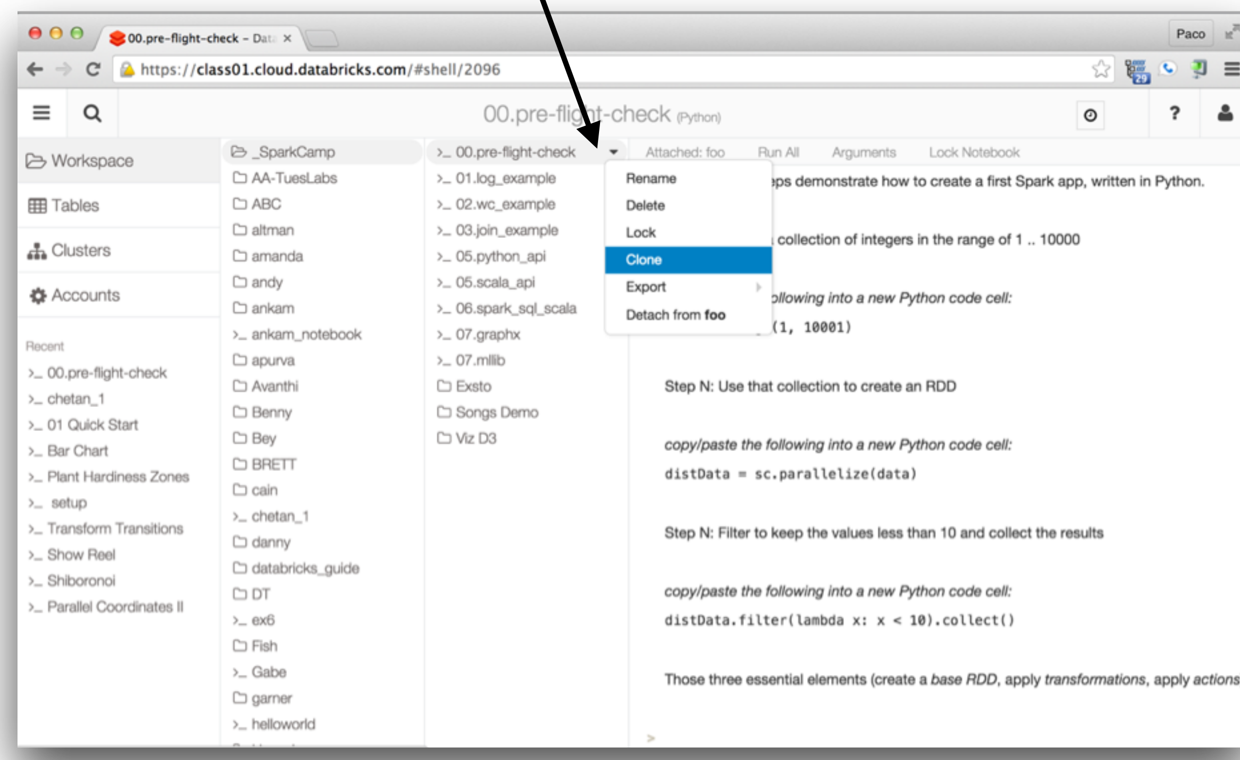
Getting Started: *Step 6*

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



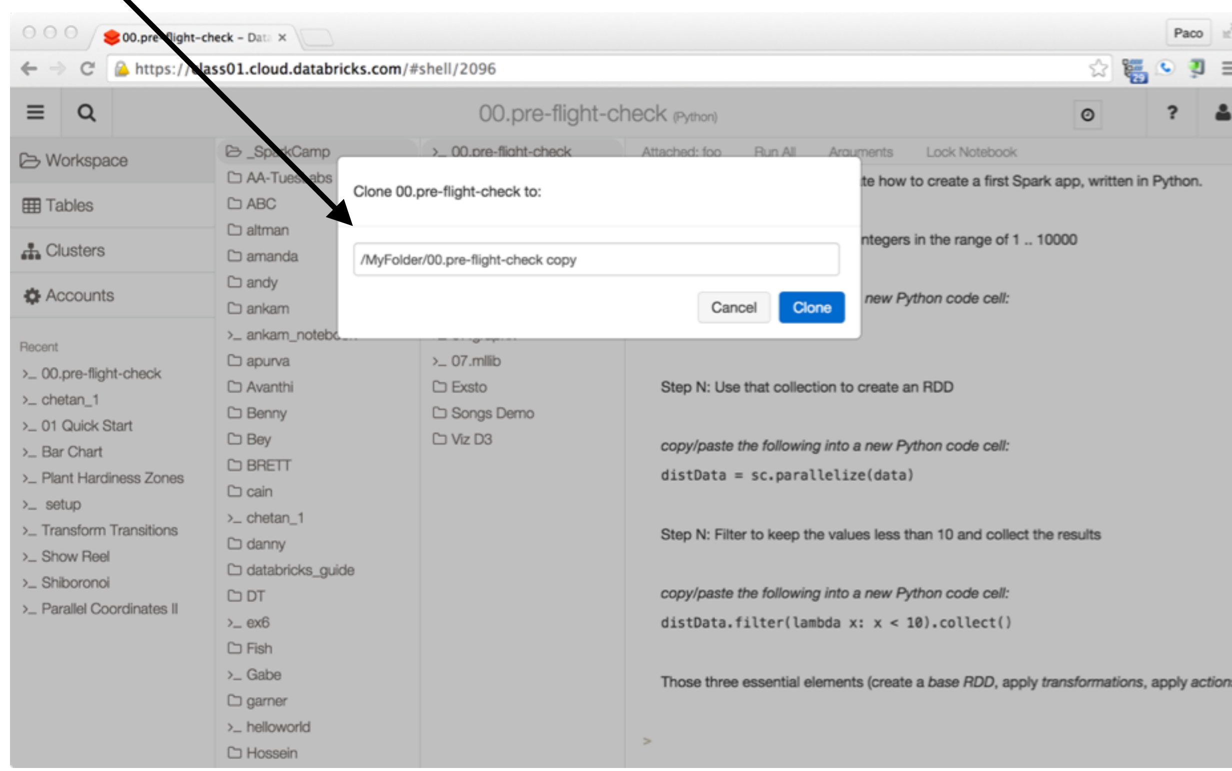
Getting Started: *Step 7*

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



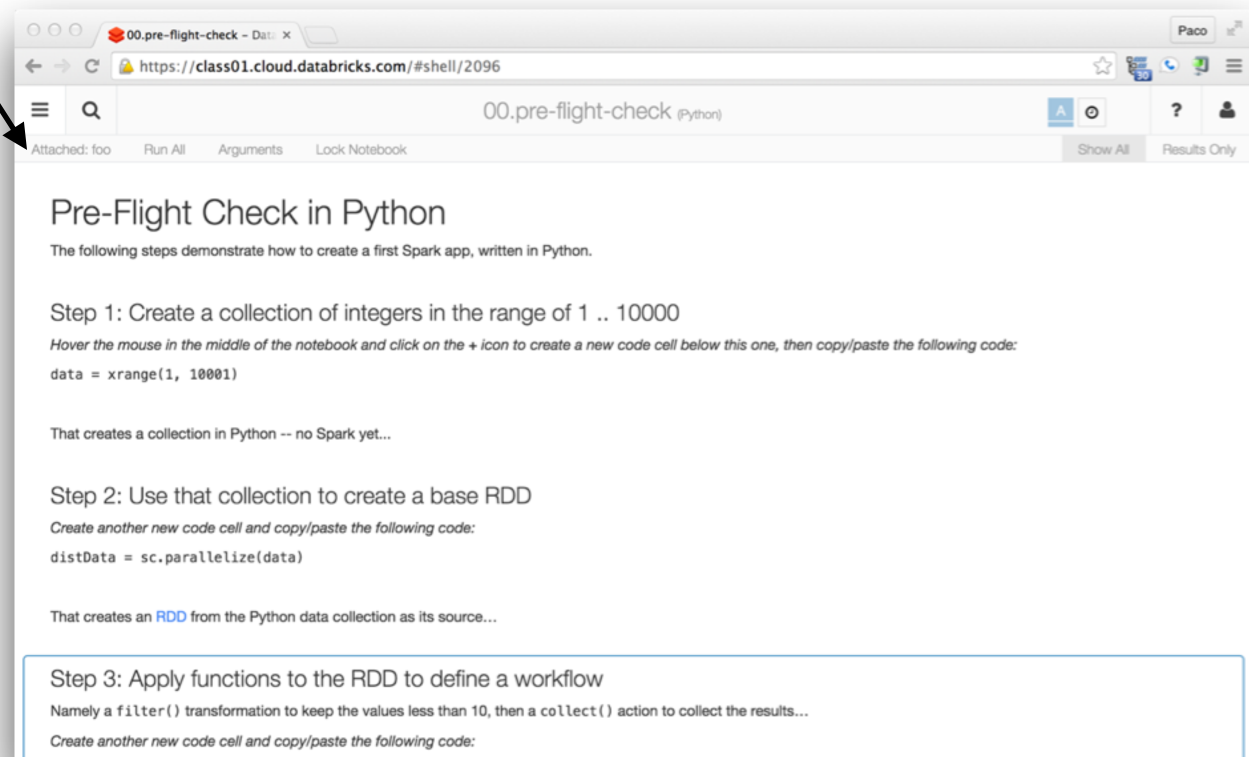
Getting Started: *Step 8*

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



Getting Started: *Step 9*

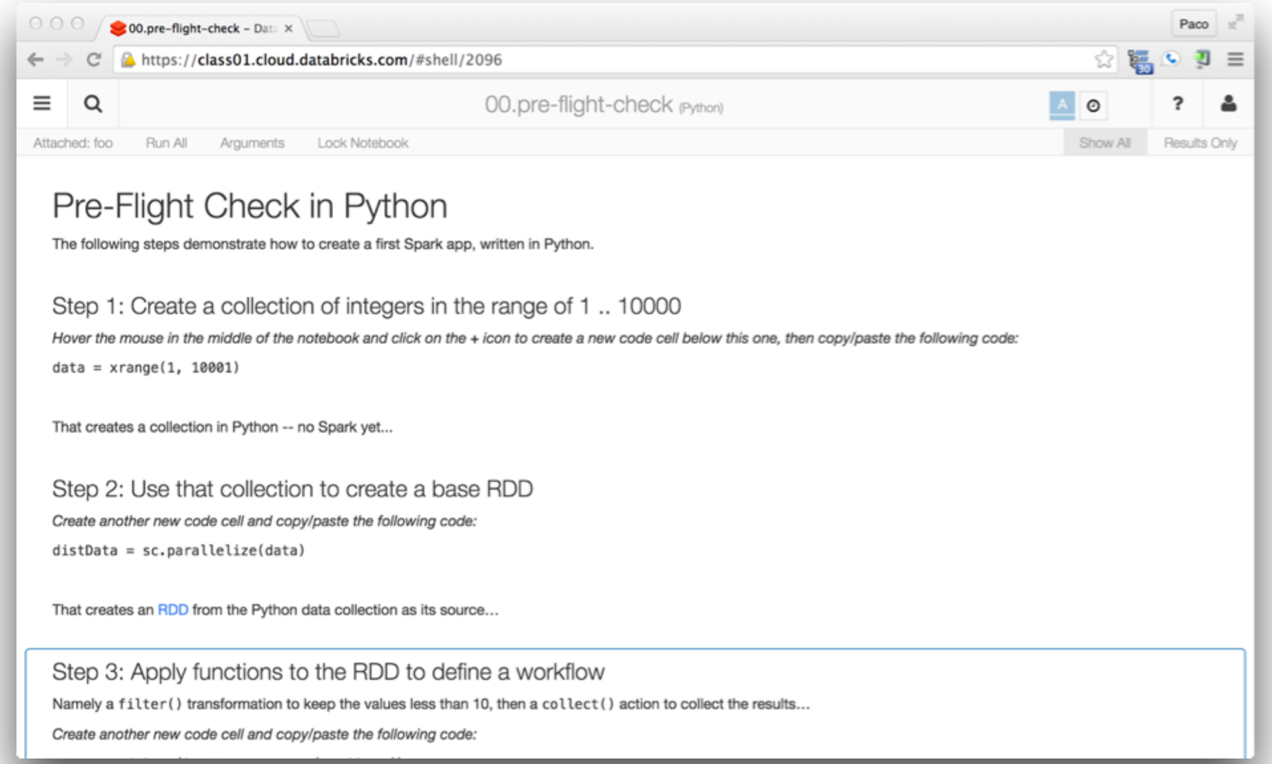
Attach your *cluster* – same as your *username*:



Getting Started: *Coding Exercise*

Now let's get started with the coding exercise!

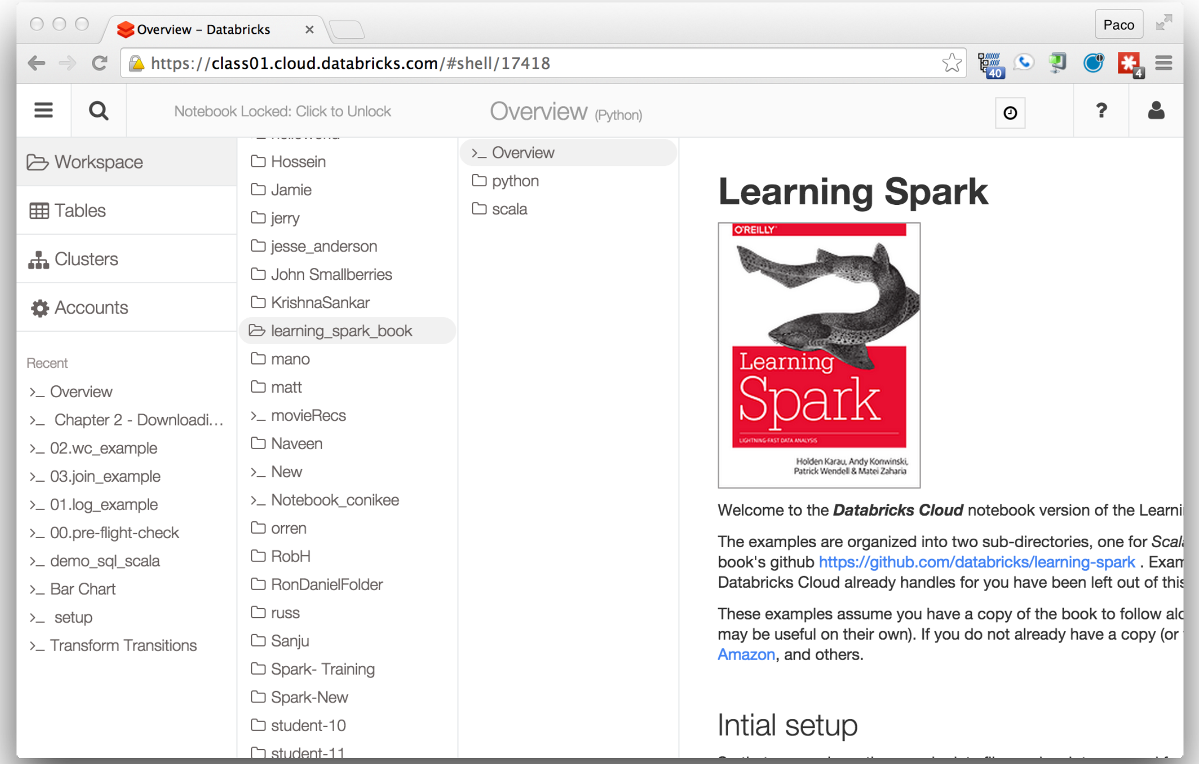
We'll define an initial Spark app in three lines of code:



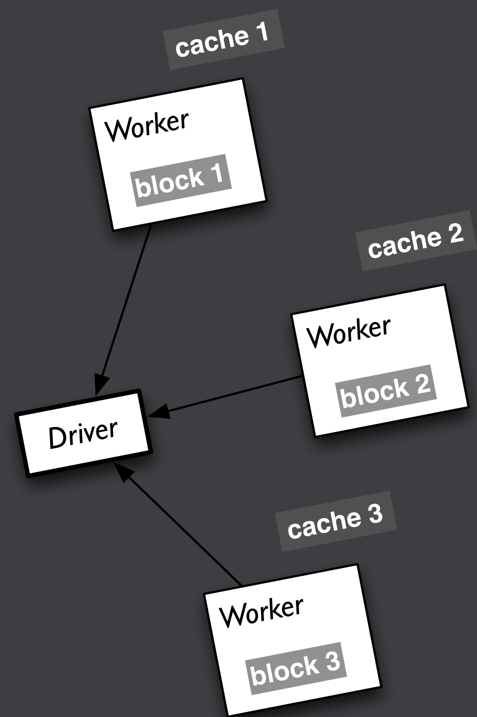
Getting Started: *Extra Bonus!!*

See also the `/learning_spark_book`

for all of its code examples in notebooks:



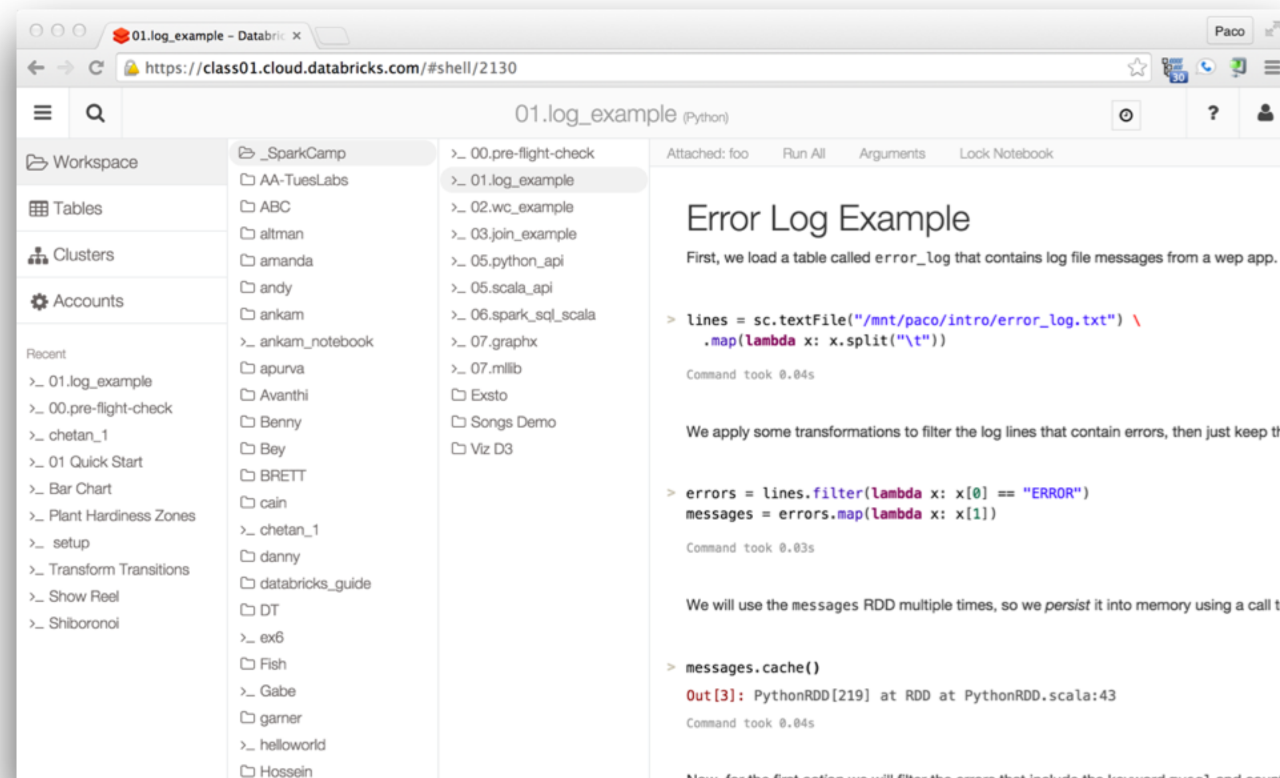
How Spark runs on a Cluster



databricks

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*

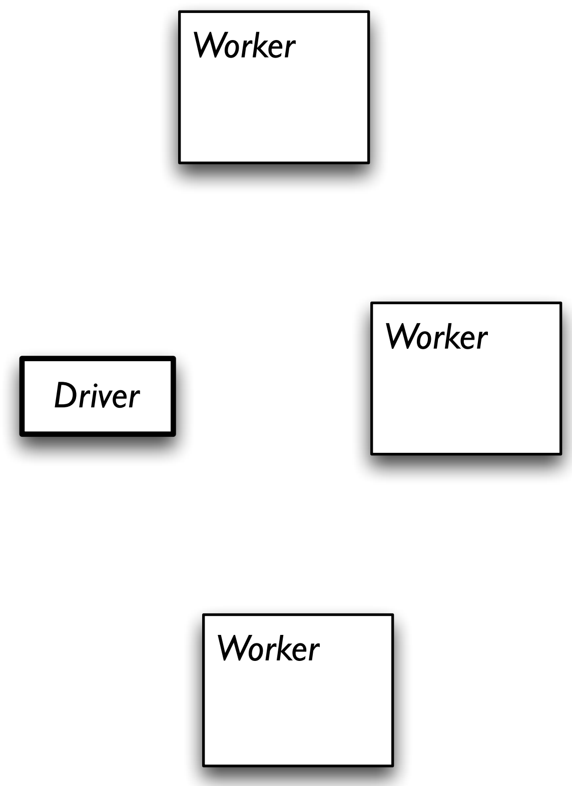
Note that we can examine the *operator graph* for a transformed RDD, for example:

```
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*

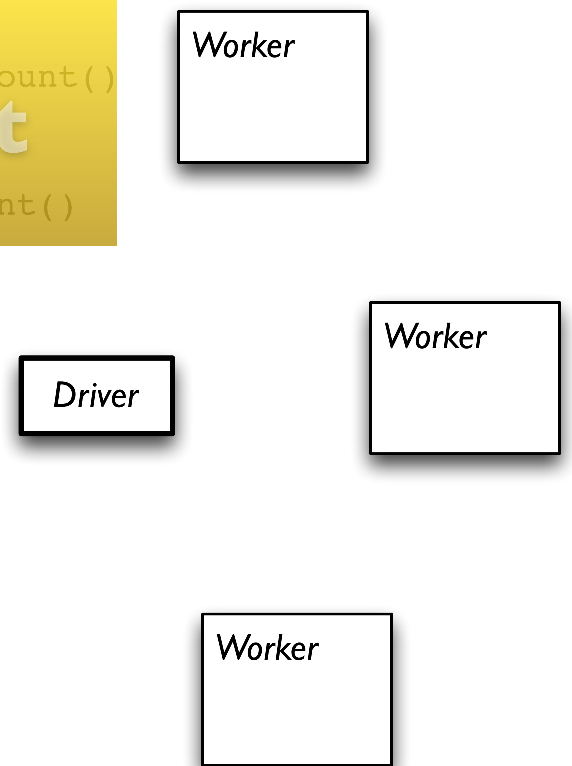
We start with Spark running on a cluster… submitting code to be evaluated on it:



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()
```

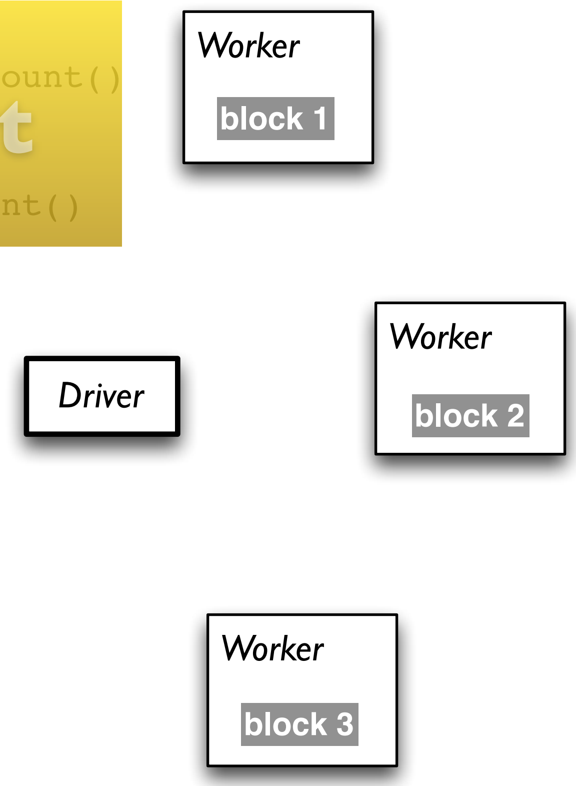
discussing the other part



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()
```

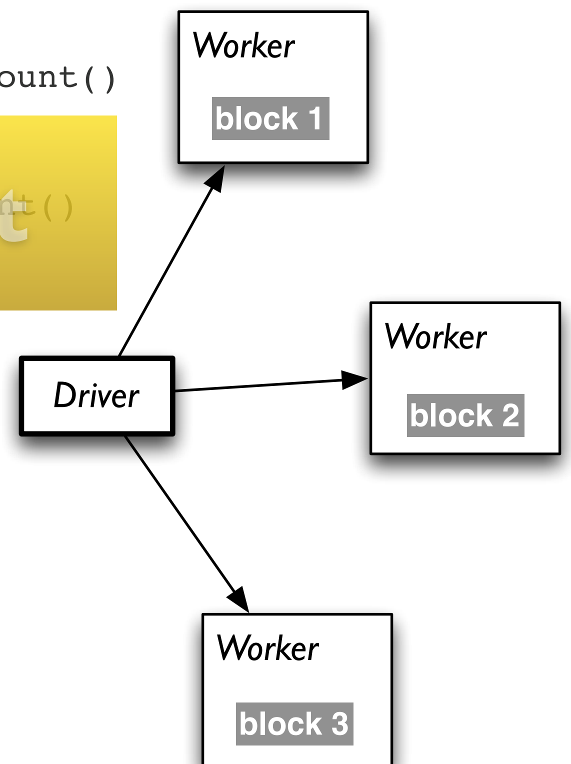
discussing the other part



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()
```

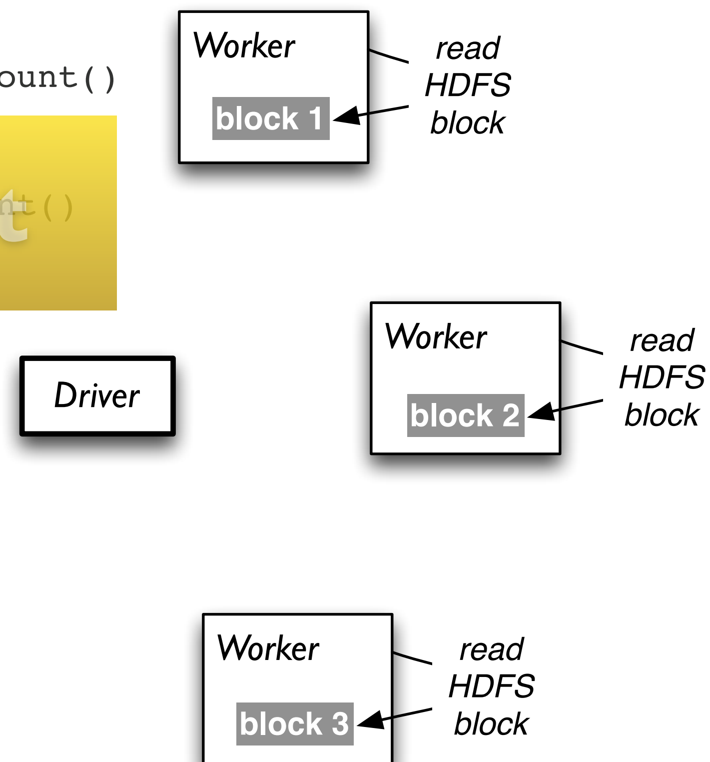
discussing the other part



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()
```

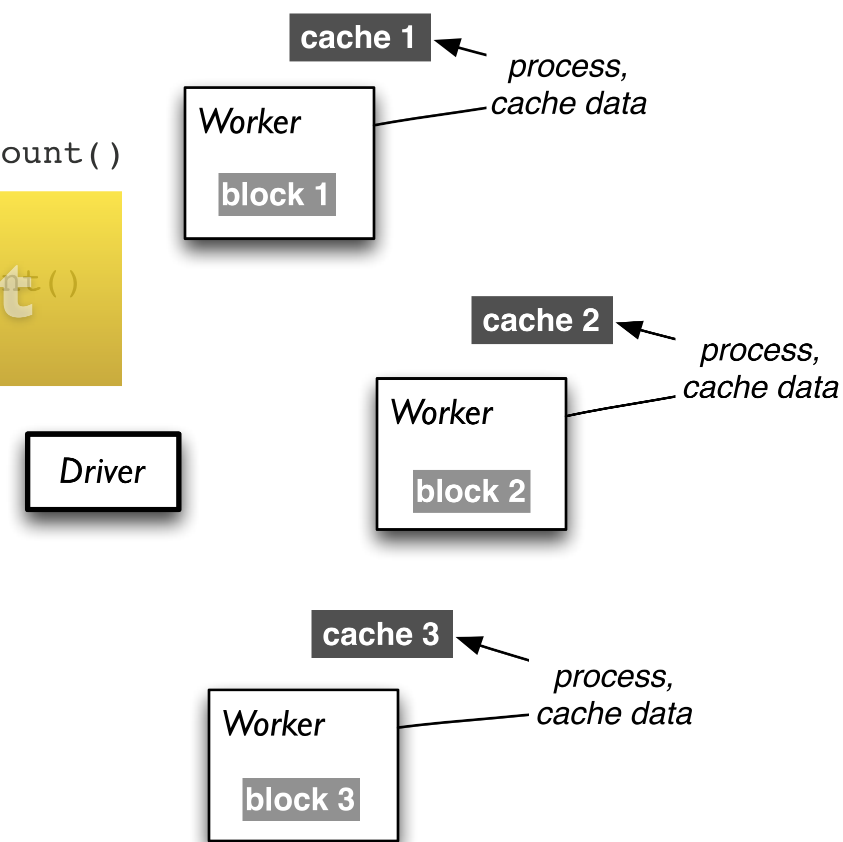
discussing the other part



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()
```

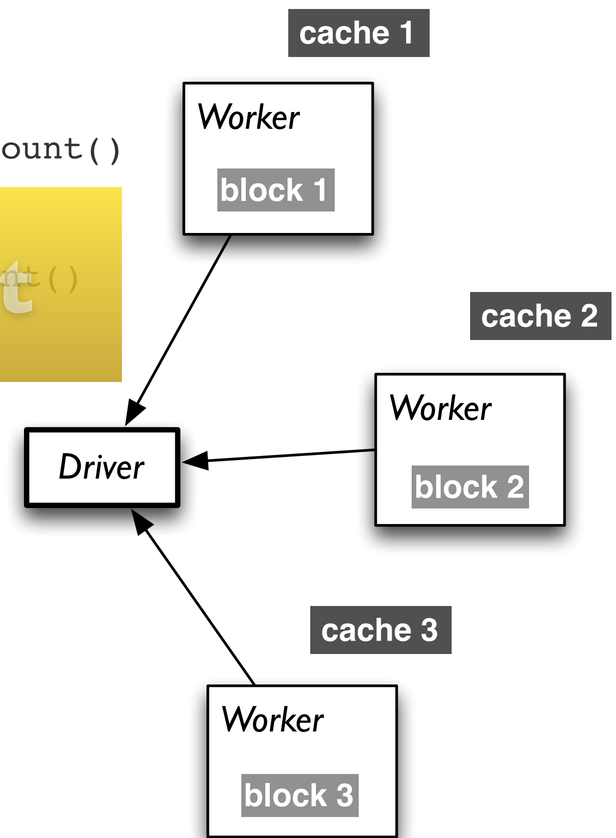
discussing the other part



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

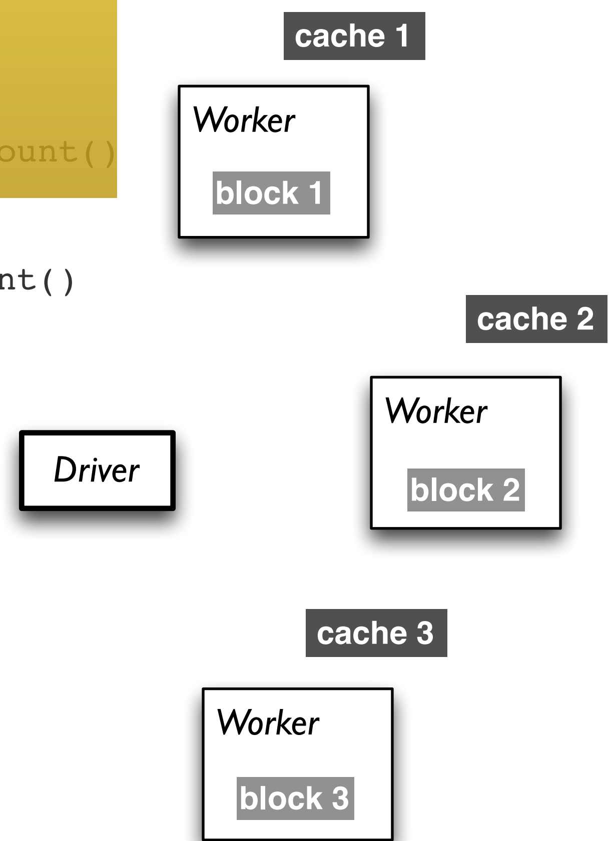


Spark Deconstructed: *Log Mining Example*

```
# 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()
```

discussing the other part

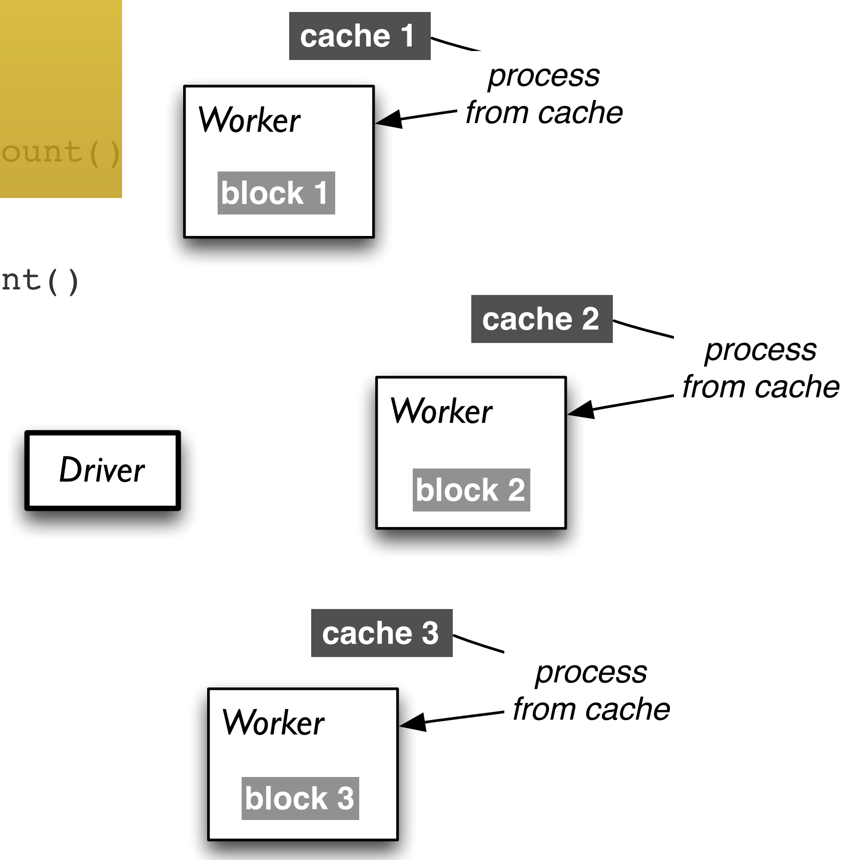
```
# action 2
messages.filter(lambda x: x.find("php") > -1).count()
```



Spark Deconstructed: *Log Mining Example*

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

# 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()
```

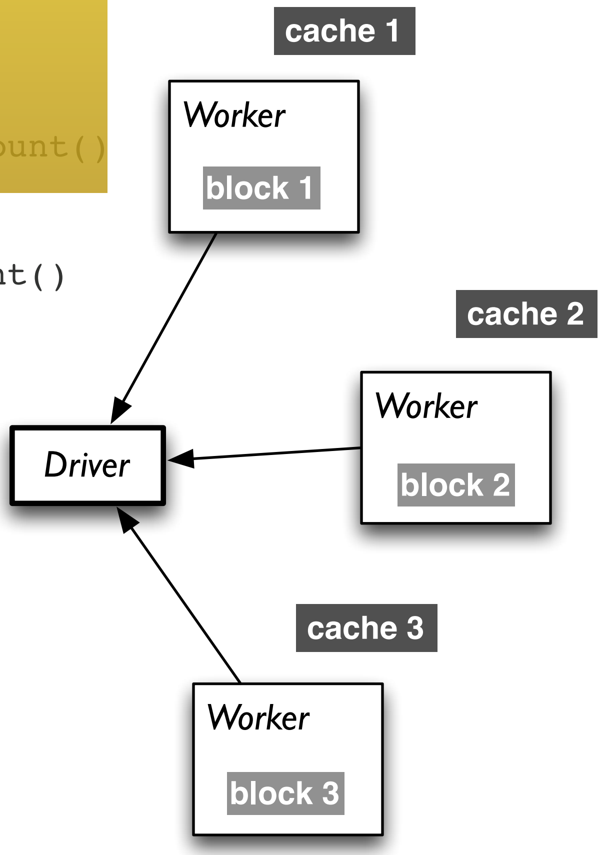
discussing the other part



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()
```

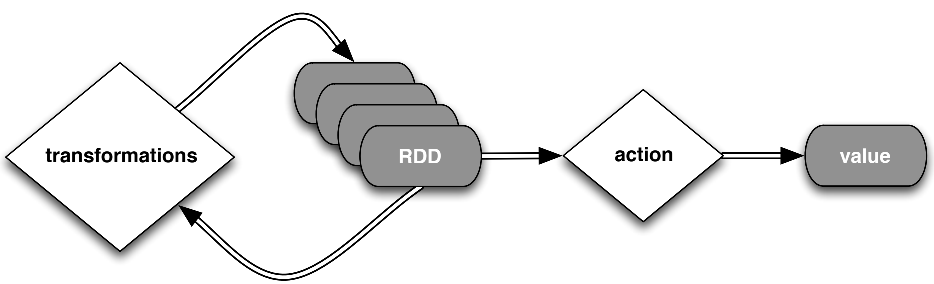
discussing the other part



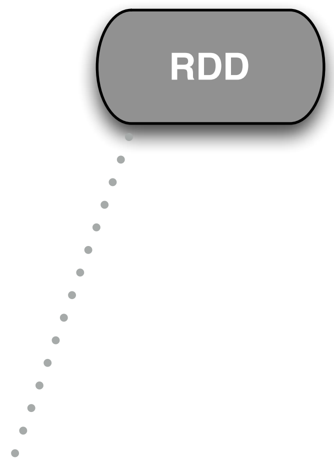
Spark Deconstructed: *Log Mining Example*

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()
```

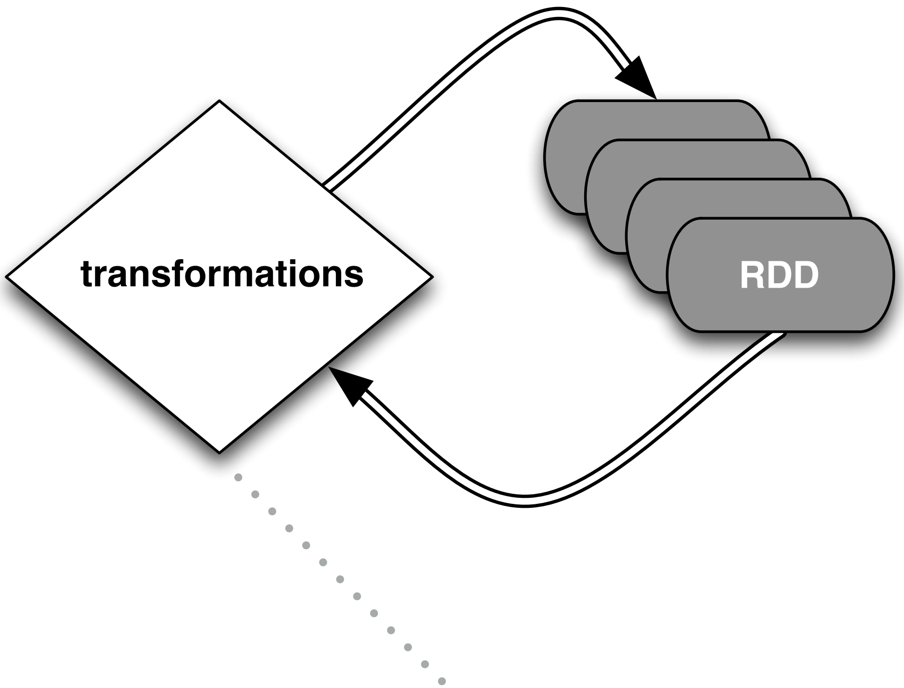


Spark Deconstructed: *Log Mining Example*



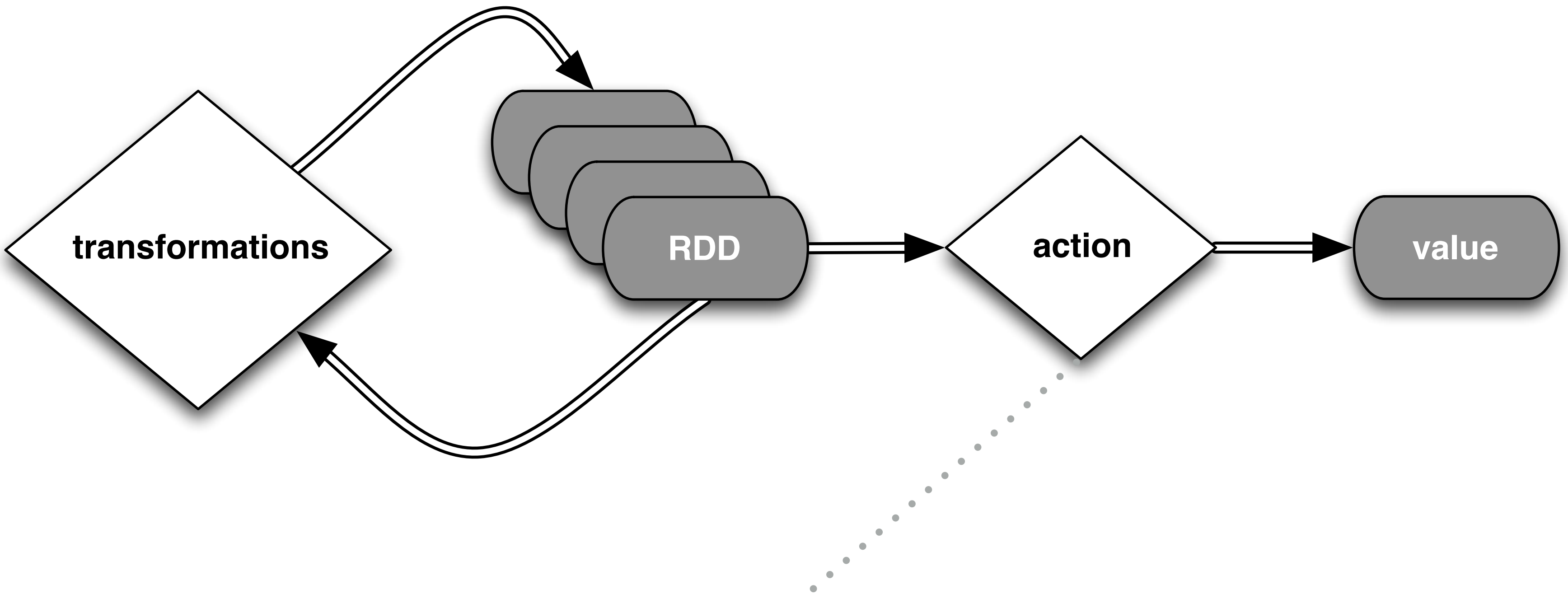

```
# base RDD
lines = sc.textFile("/mnt/paco/intro/error_log.txt") \
 .map(lambda x: x.split("\t"))
```

Spark Deconstructed: *Log Mining Example*



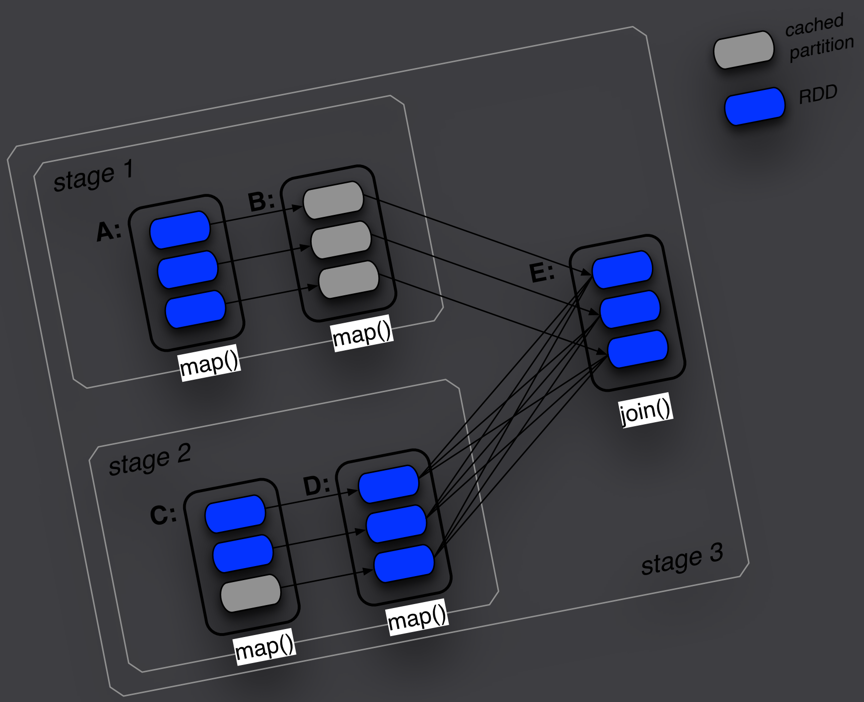

```
# transformed RDDs
errors = lines.filter(lambda x: x[0] == "ERROR")
messages = errors.map(lambda x: x[1])

# persistence
messages.cache()
```

Spark Deconstructed: *Log Mining Example*




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

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

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

```
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));
```

Coding Exercise: *WordCount*

```java
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);
  }
}
```

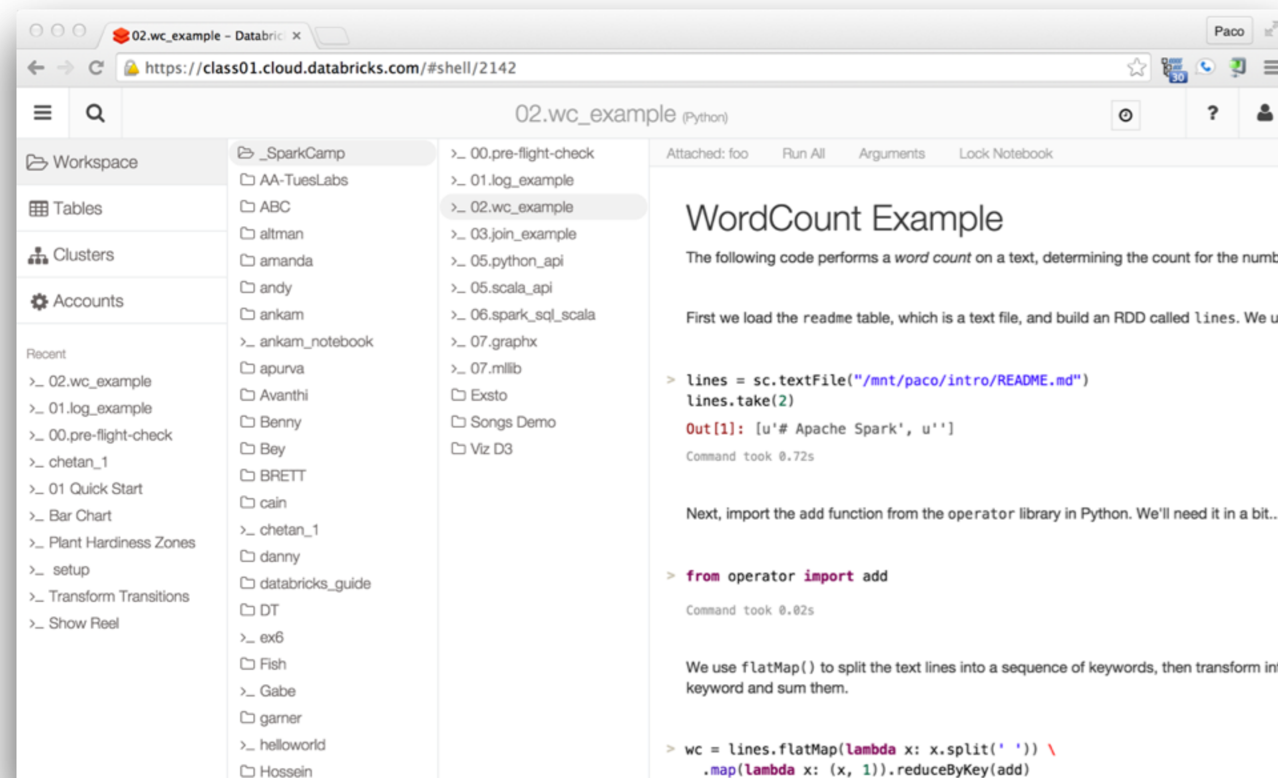
```scala
val f = sc.textFile(inputPath)
val w = f.flatMap(l => l.split(" ")).map(word => (word, 1)).cache()
w.reduceByKey(_ + _).saveAsText(outputPath)
```

WordCount in 3 lines of Spark

WordCount in 50+ lines of Java MR

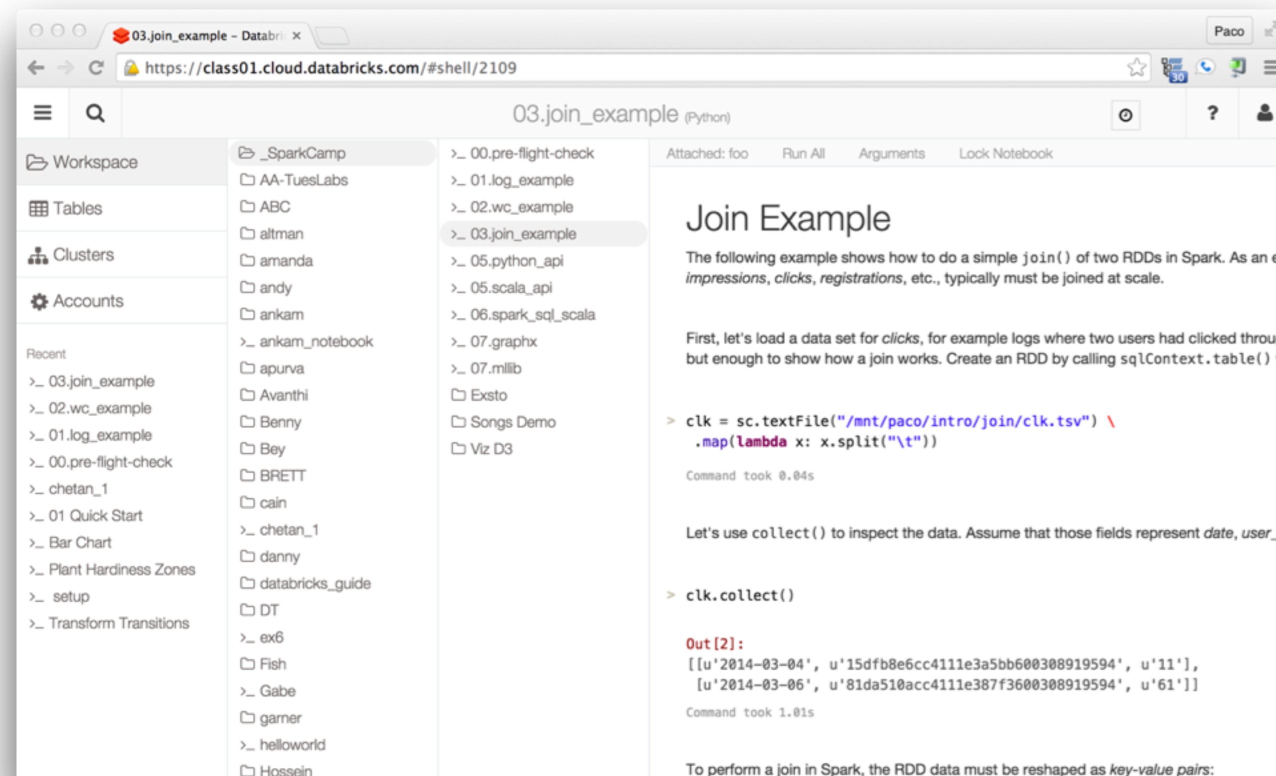
Coding Exercise: *WordCount*

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

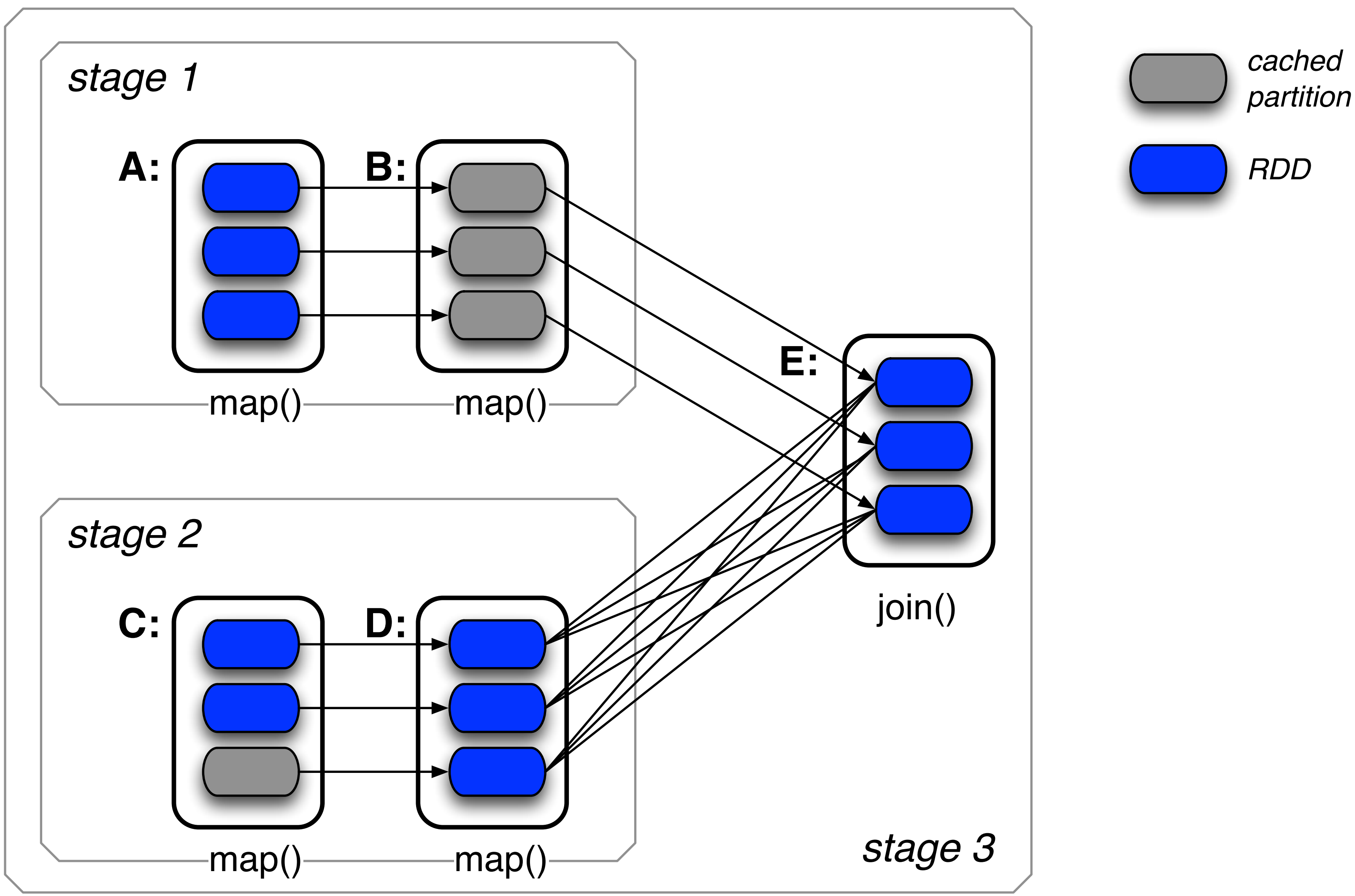


Coding Exercise: *Join*

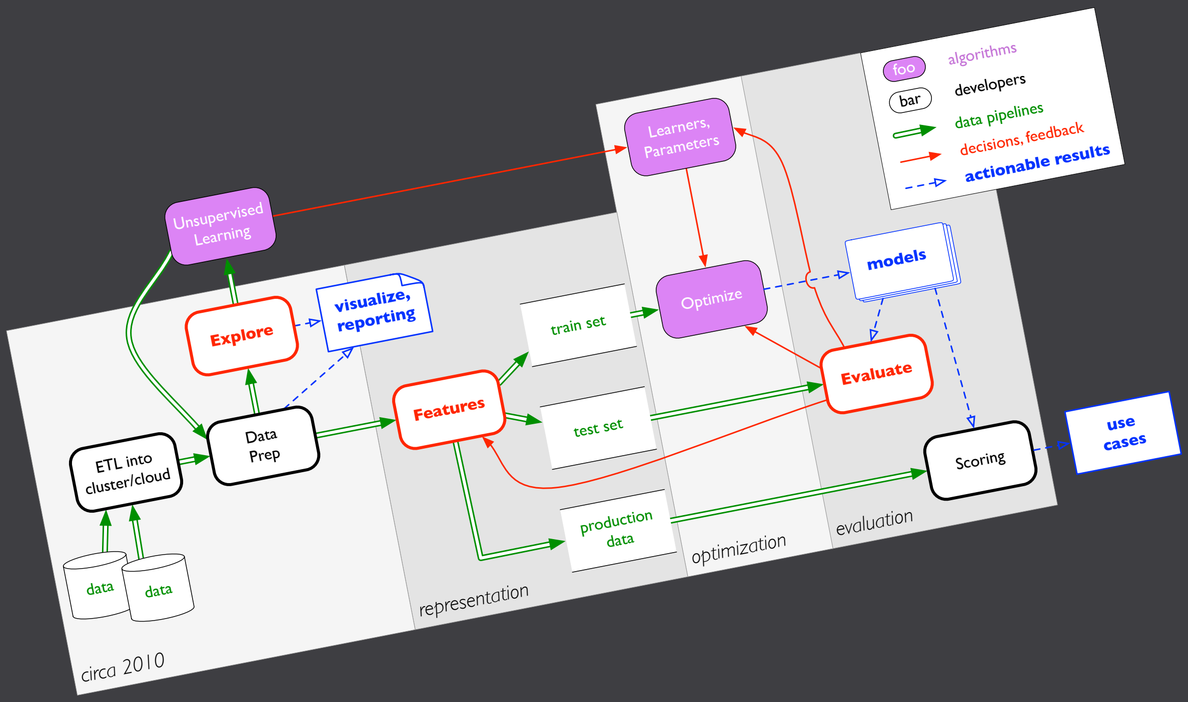
Clone and run `/_SparkCamp/03.join_example` in your folder:



Coding Exercise: *Join and its Operator Graph*



DBC Essentials



databricks

DBC Essentials: *What is Databricks Cloud?*

Also see **FAQ** for more details…

Databricks Workspace

Databricks Platform

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: *Notebooks*

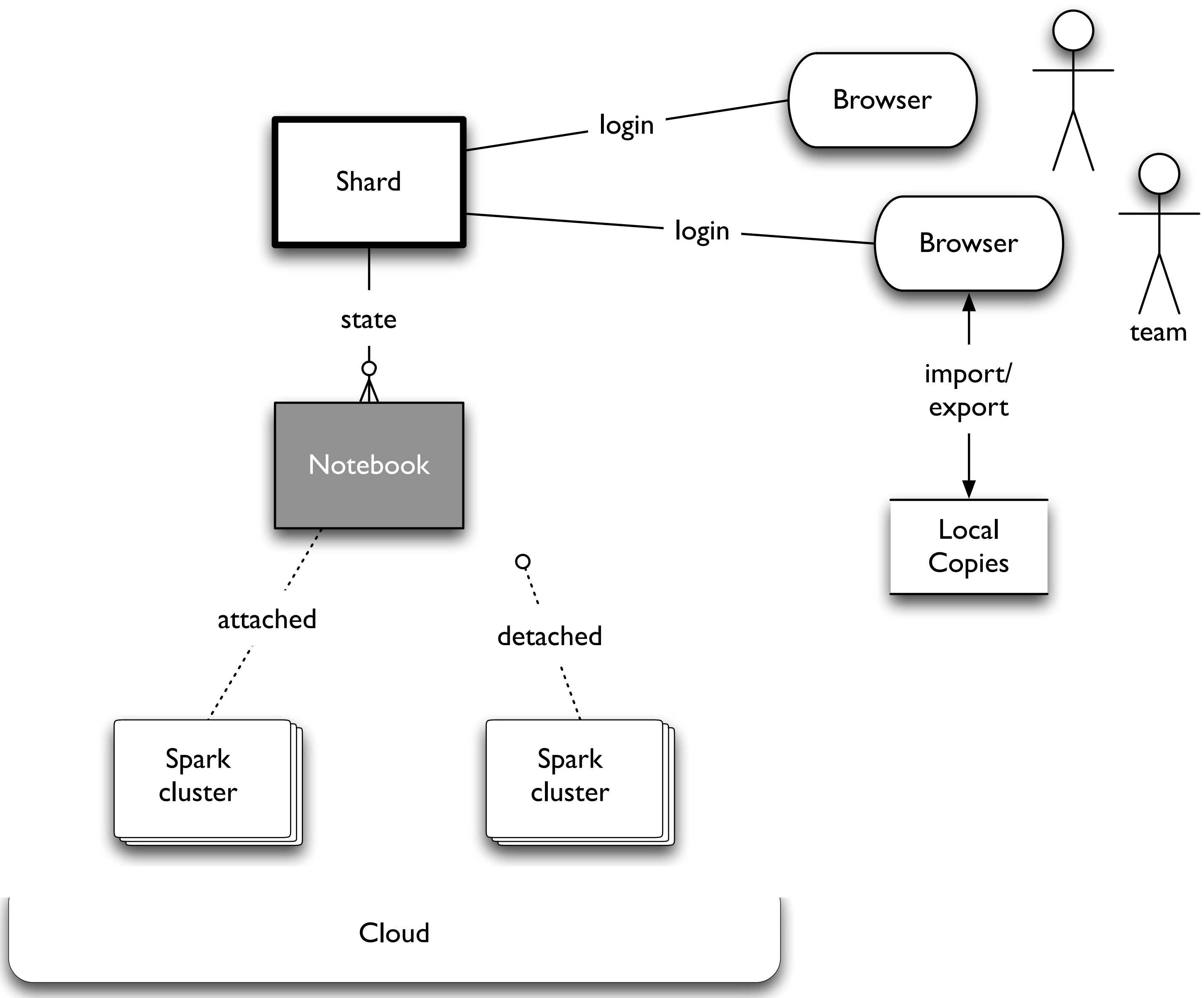
- Series of commands (think shell++)
- Each notebook has a language type, chosen at notebook creation:
  - Python + SQL
  - Scala + SQL
  - SQL only
- Command output captured in notebook
- Commands can be…
  - edited, reordered, rerun, exported, cloned, imported, etc.

DBC Essentials: *Clusters*

- Open source Spark clusters hosted in the cloud
- Access the Spark UI
- Attach and Detach notebooks to clusters

NB: our training shards use 7 GB cluster configurations

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

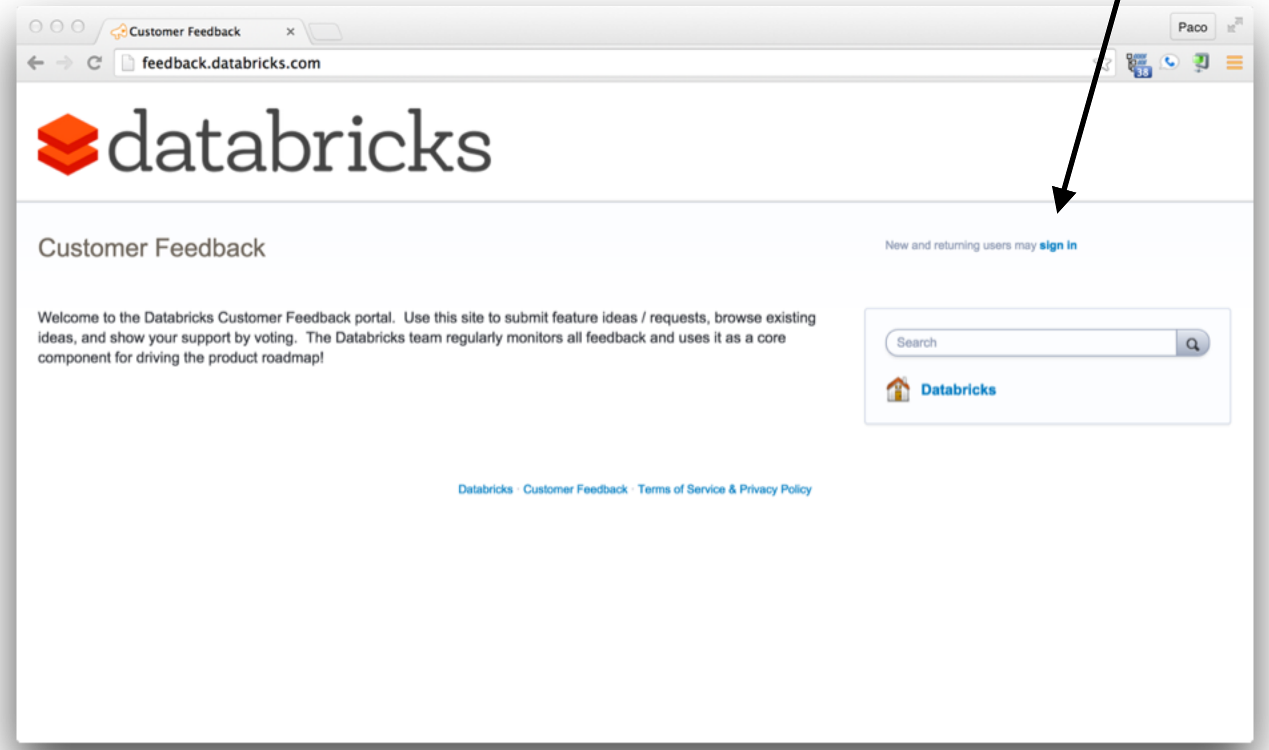
DBC Essentials: *Feedback*

Other feedback, suggestions, etc.?

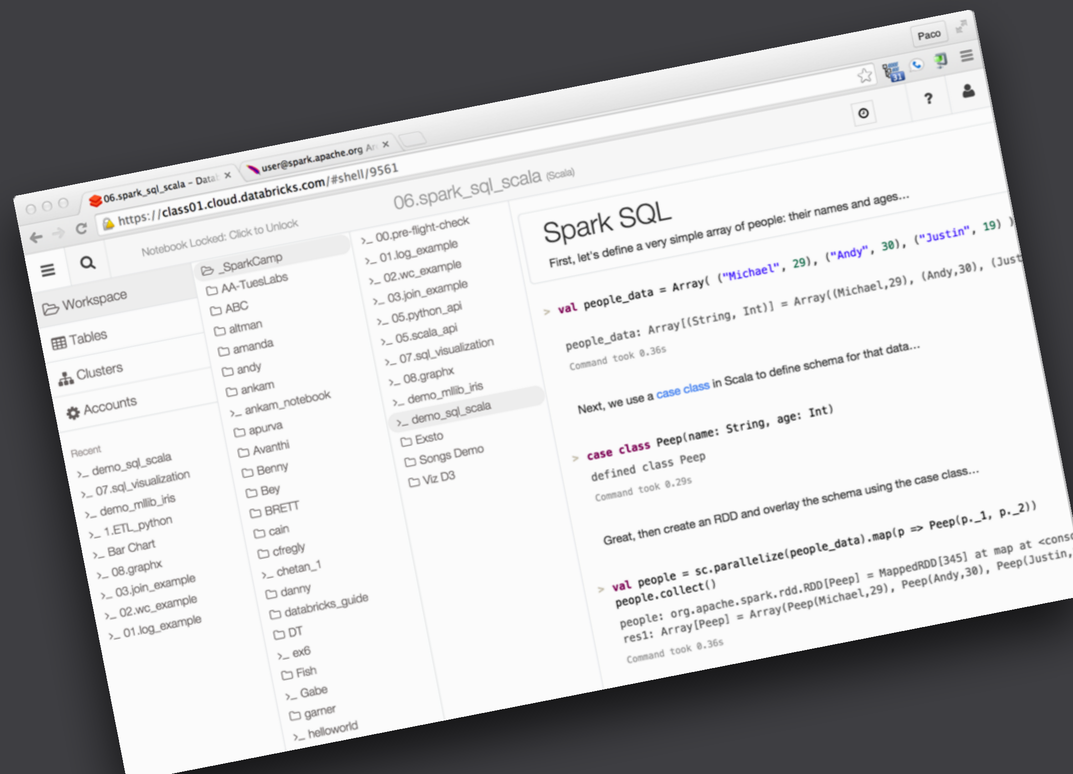
http://feedback.databricks.com/

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

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

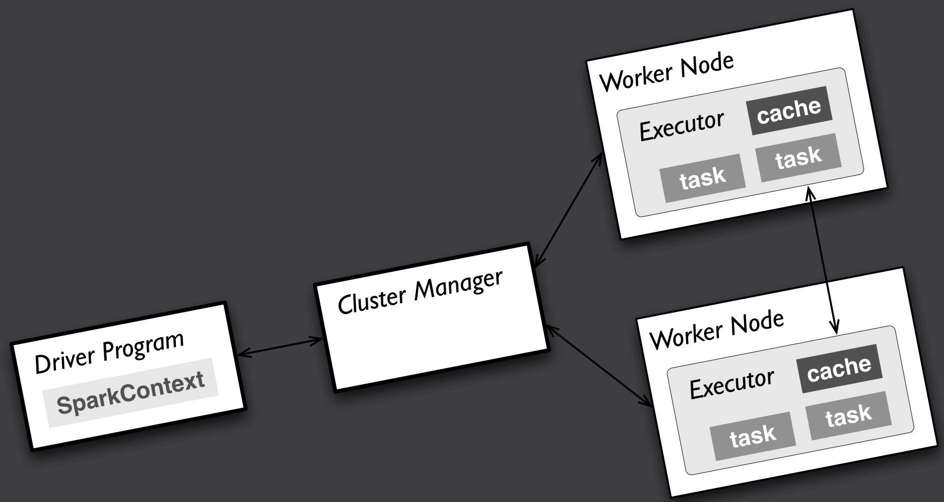
Coding Exercises: *Workflow assignment*

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>
```

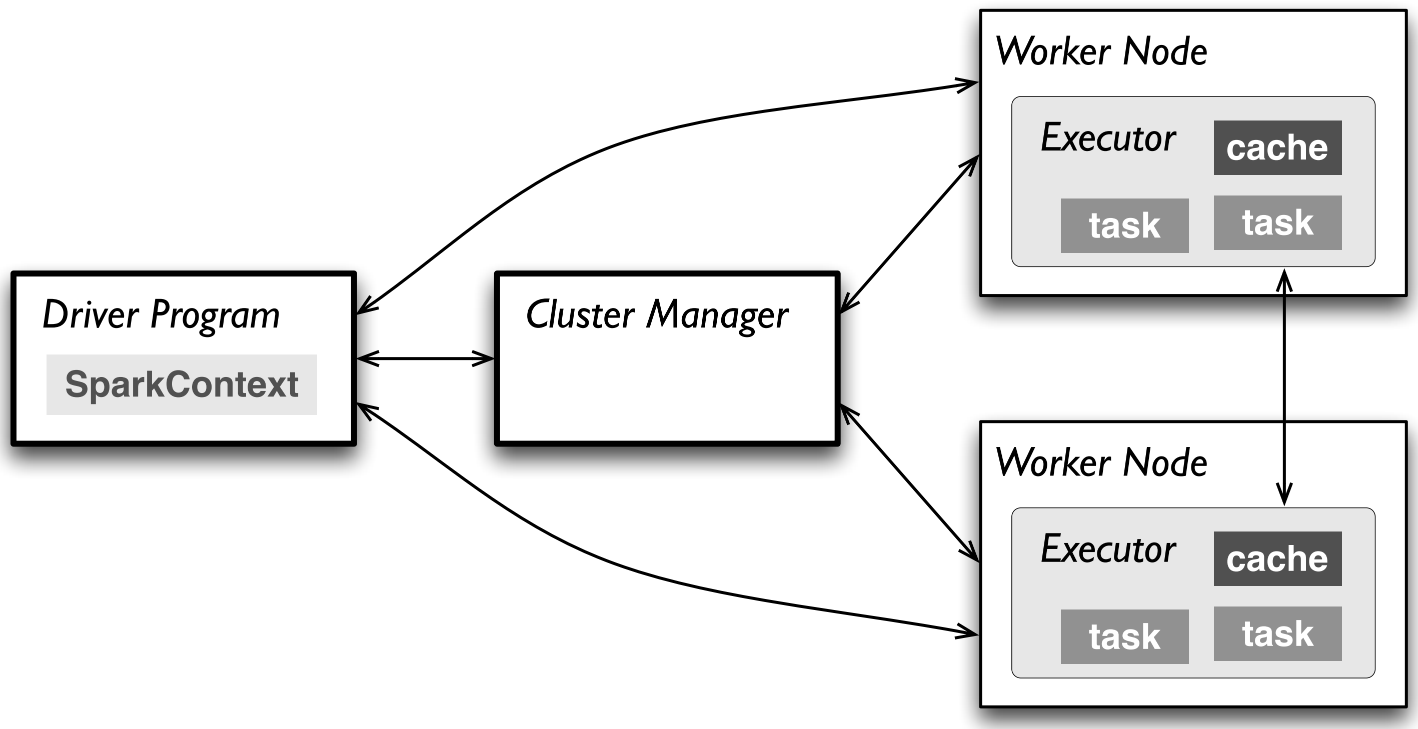
Spark Essentials: *Master*

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*

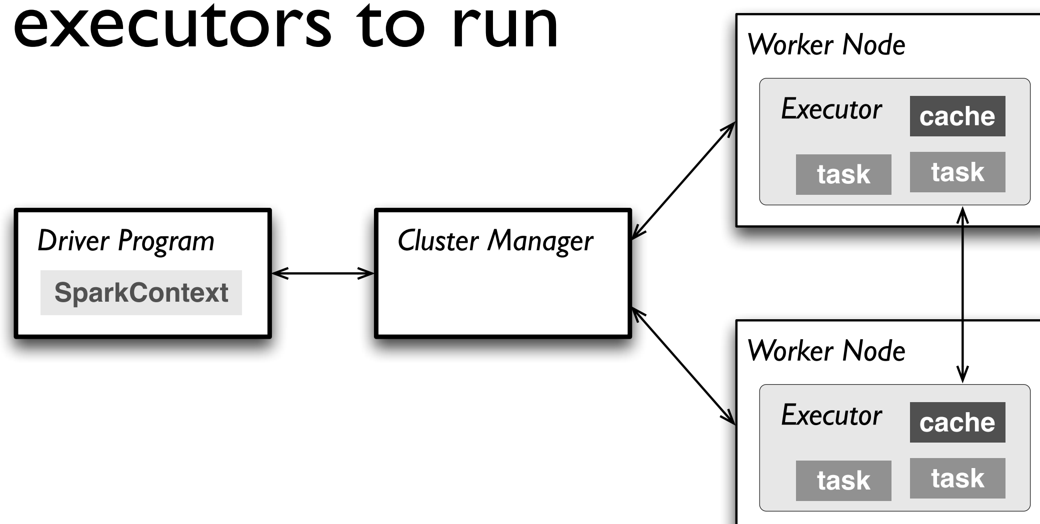
spark.apache.org/docs/latest/cluster-overview.html



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]
```

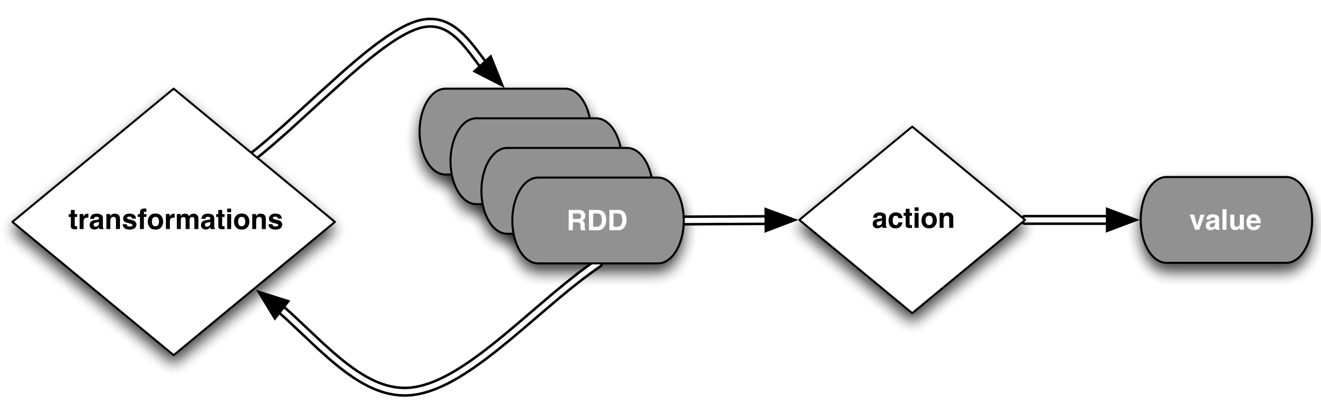
Python:

```
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()
```

closures

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()
```

closures

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 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
```

driver-side

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*

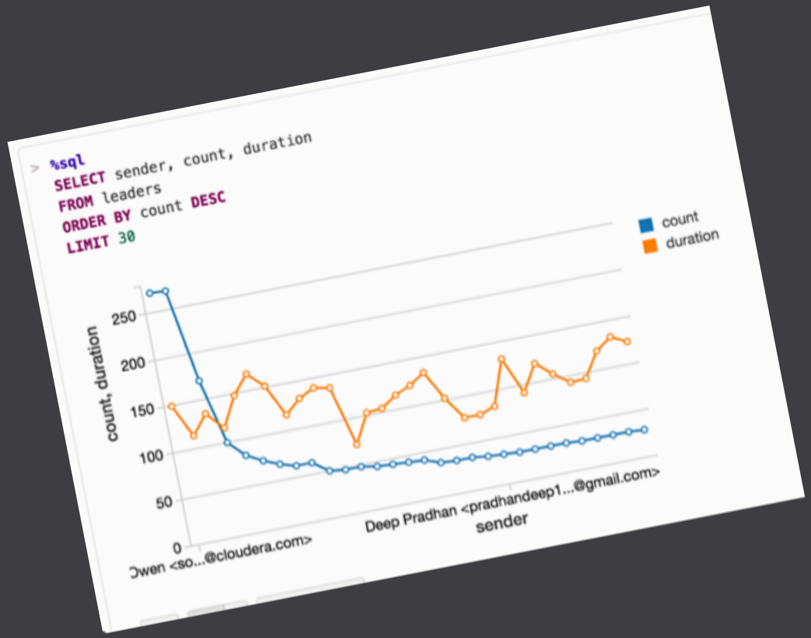
For more details about the Scala API:

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/

Spark SQL



databricks

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/

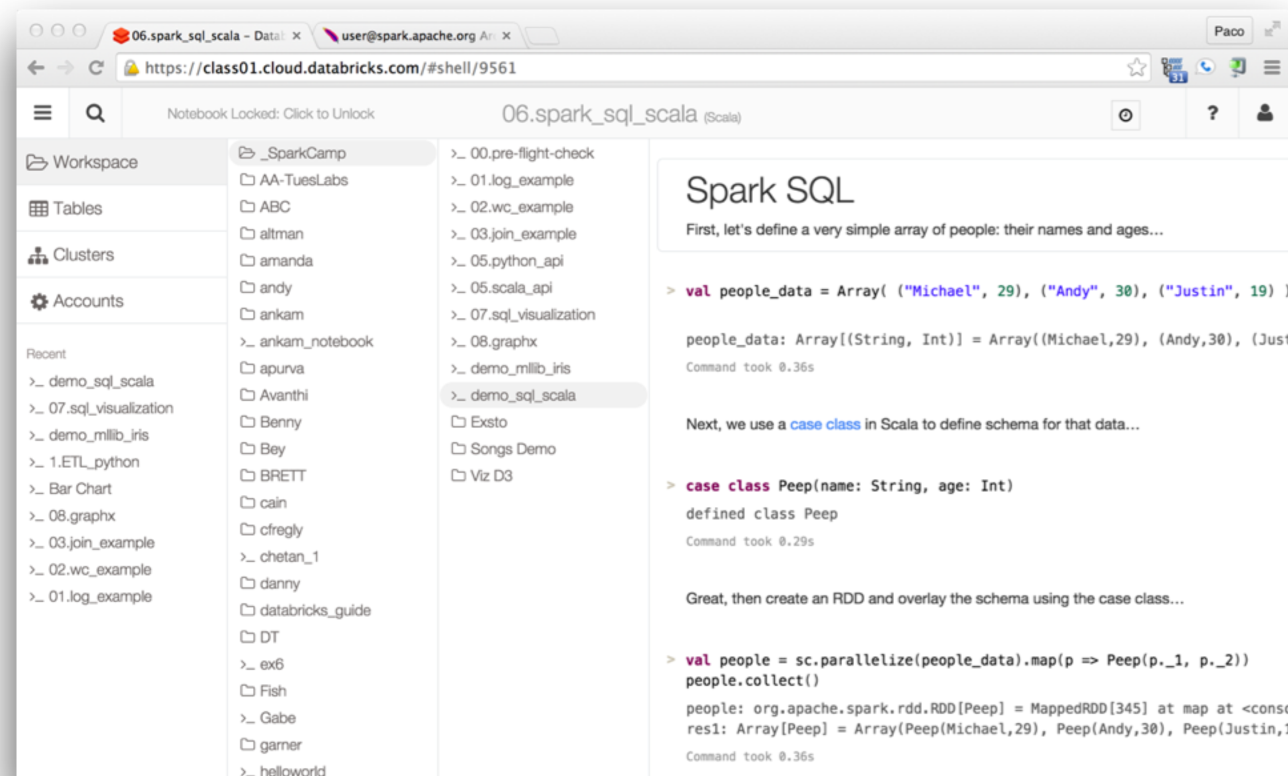
Spark SQL supports read/write of parquet files, automatically preserving the schema of the original data (HUGE benefits)

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:



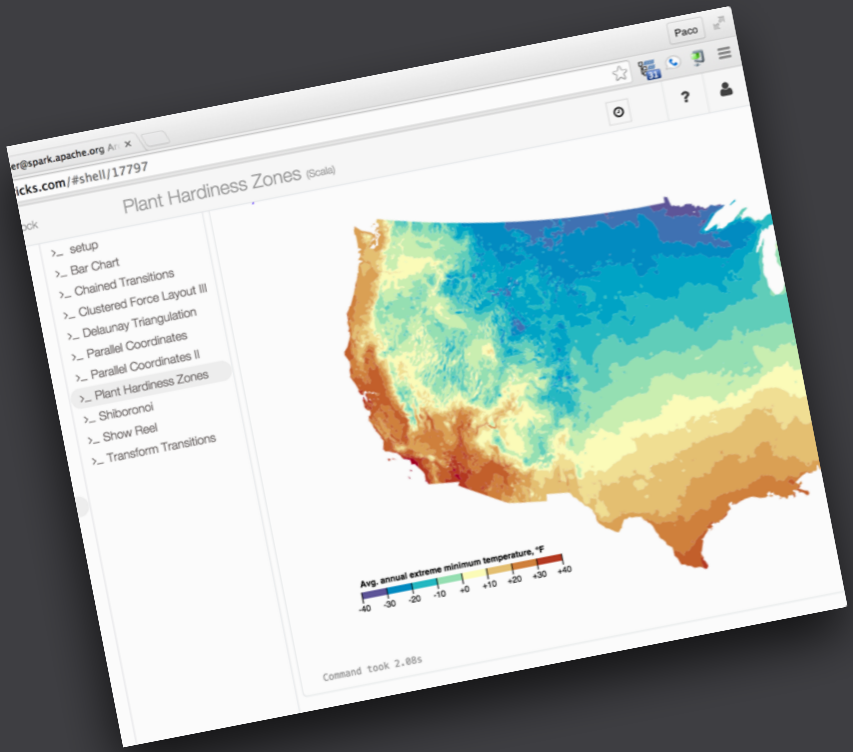
Next, we'll review the following sections in the *Databricks Guide*:

```
/databricks_guide/Importing Data
/databricks_guide/Databricks File System
```

Key Topics:

- JSON, CSV, Parquet
- S3, Hive, Redshift
- DBFS, dbutils

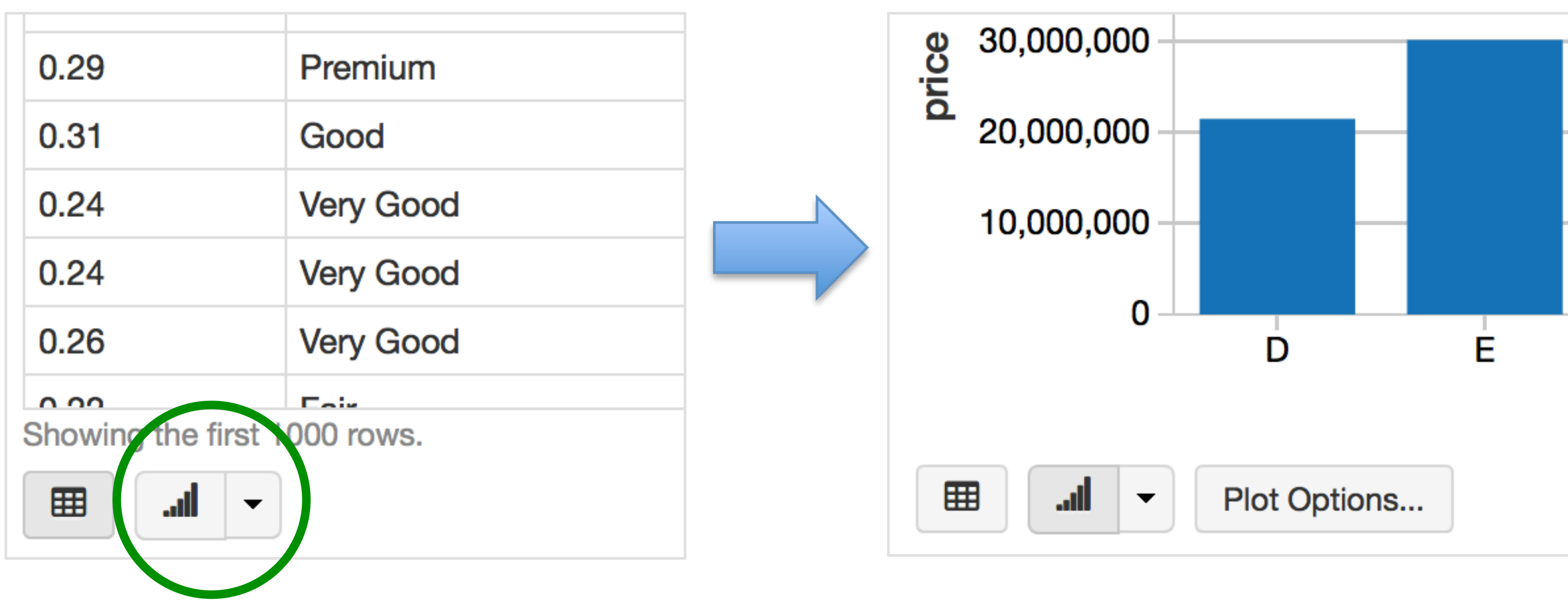
Visualization



databricks

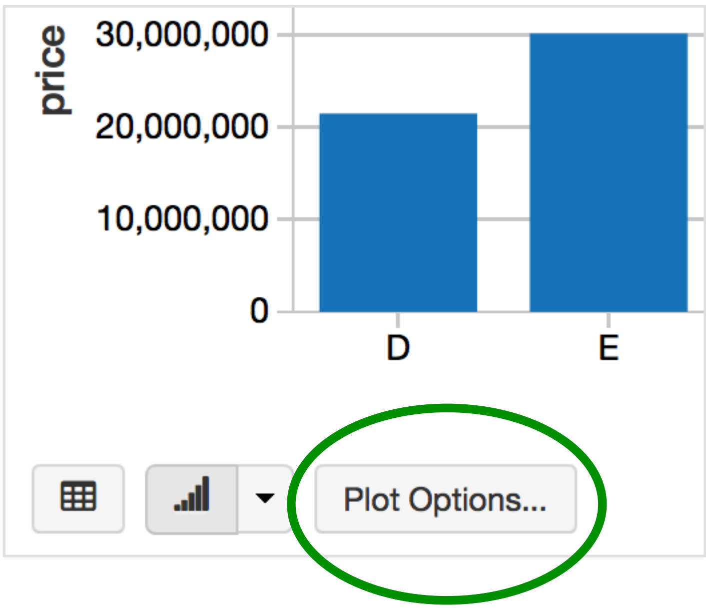
Visualization: *Built-in Plots*

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



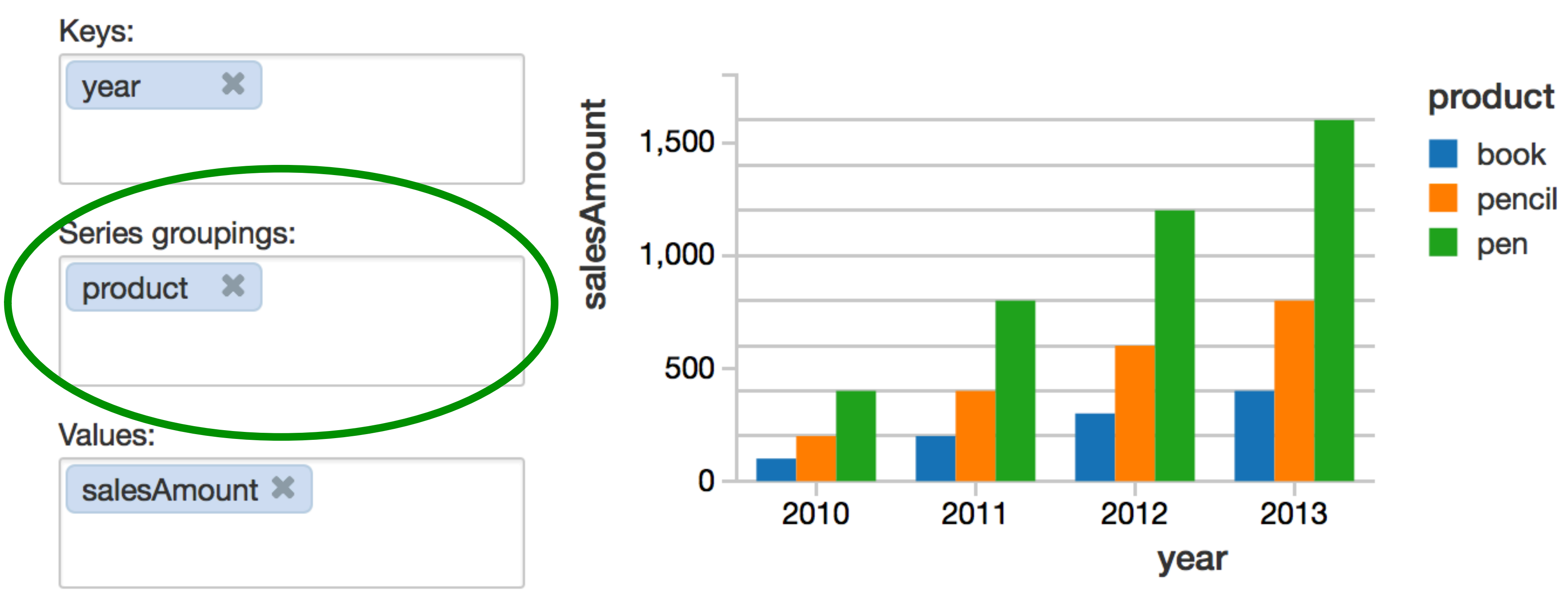
Visualization: *Plot Options*

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



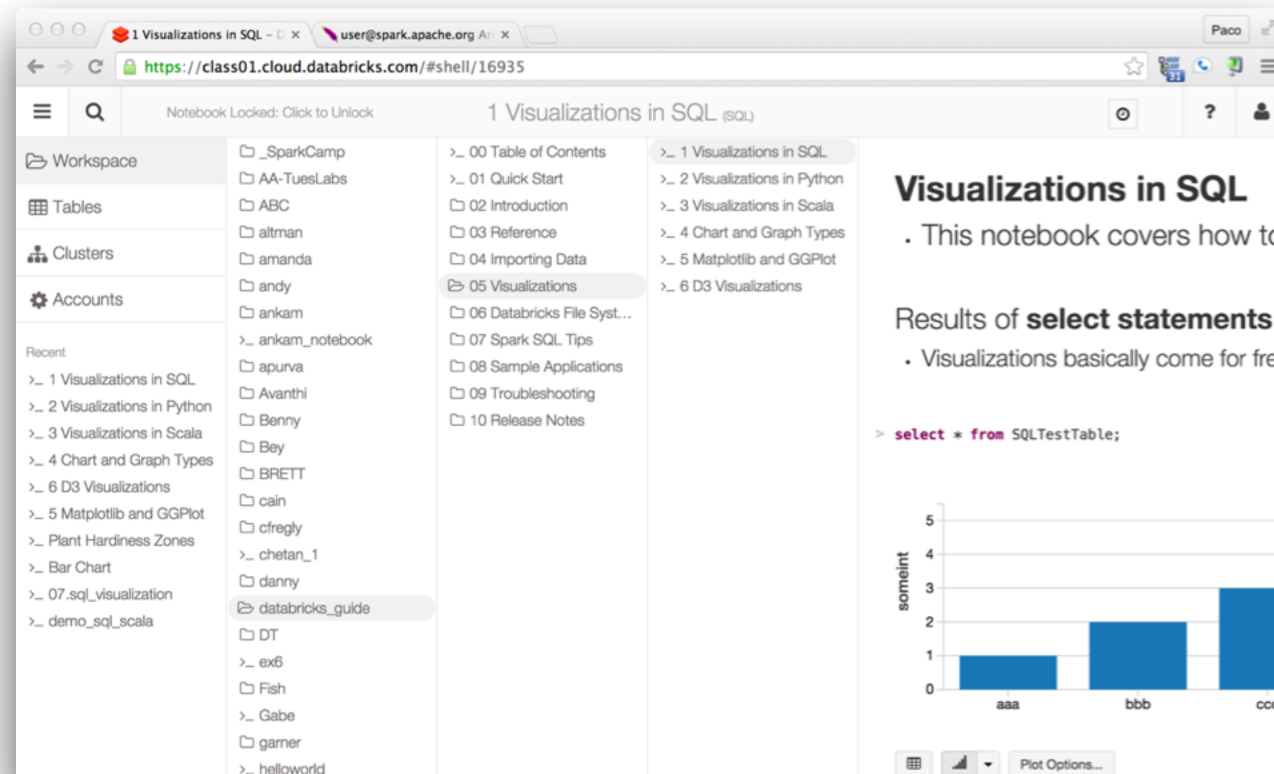
Visualization: *Series Groupings*

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



Visualization: *Reference Guide*

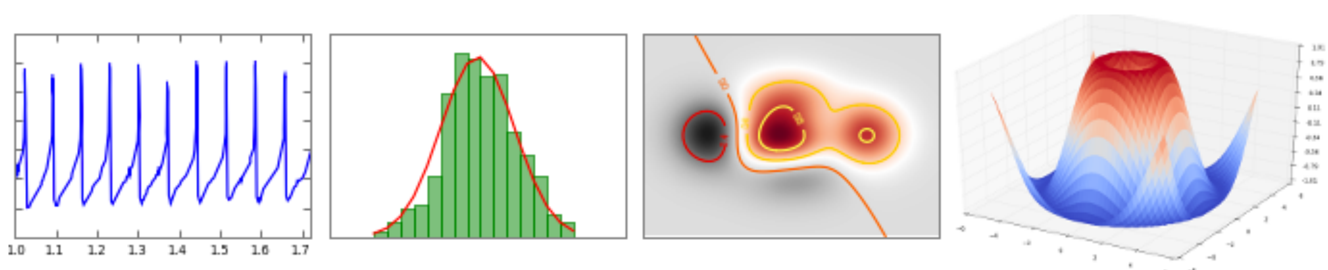
See `/databricks-guide/05 Visualizations` for details about built-in visualizations and extensions…



Visualization: *Using display()*

The `display()` command:

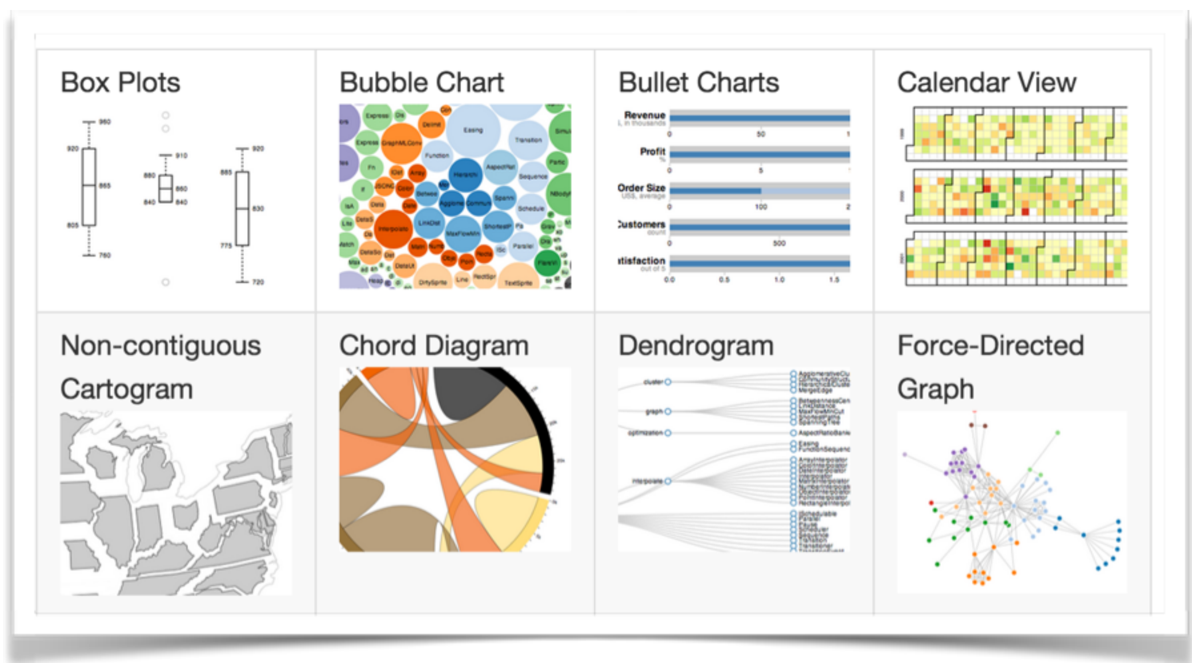
- programmatic access to visualizations
- pass a SchemaRDD to print as an HTML table
- pass a Scala list to print as an HTML table
- call without arguments to display **matplotlib** figures



Visualization: *Using displayHTML()*

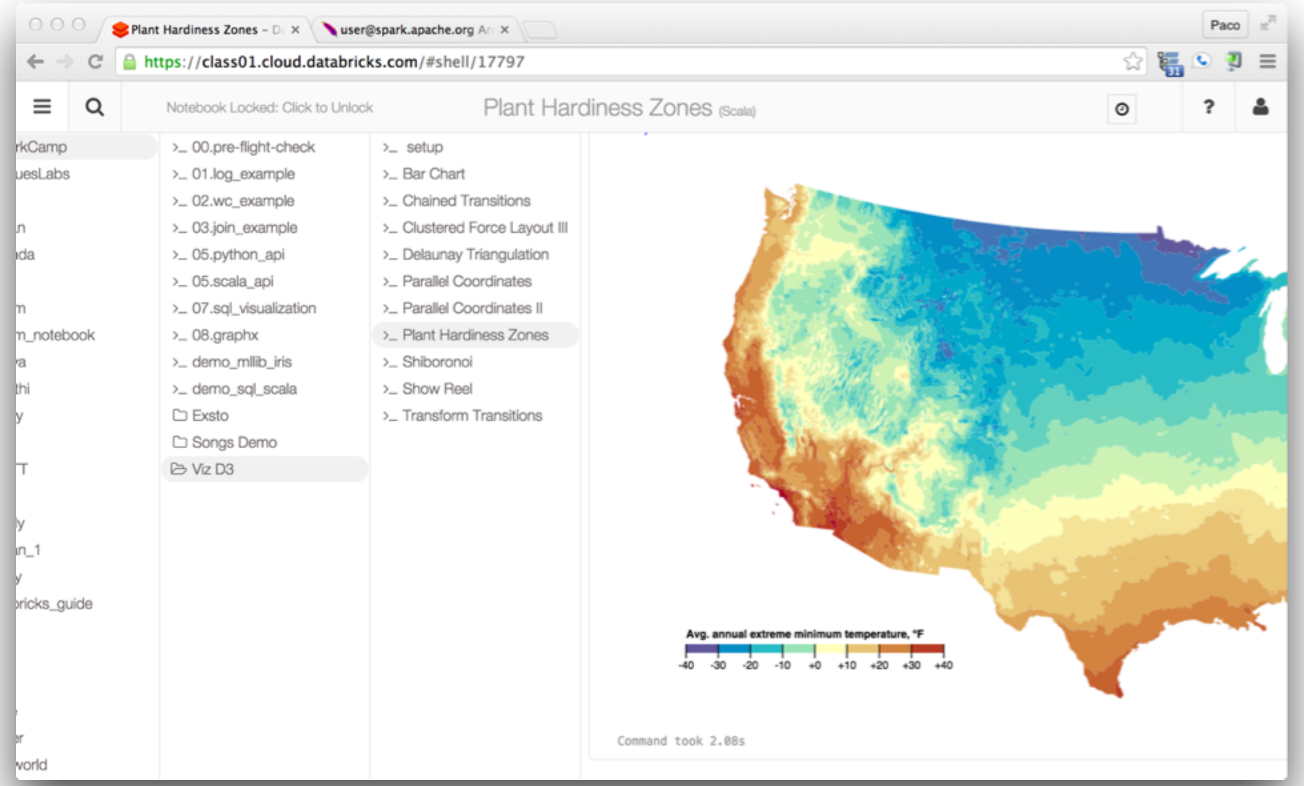
The `displayHTML()` command:

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



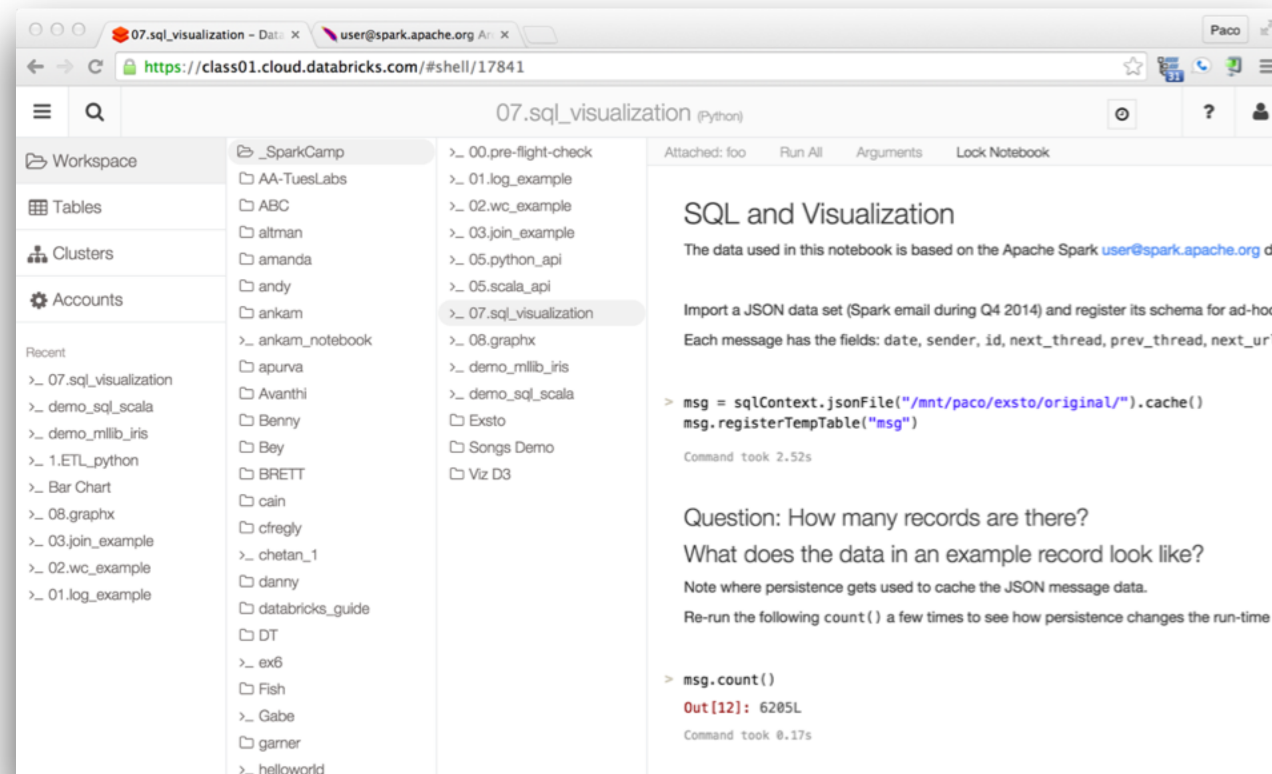
Demo: *D3 Visualization*

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:



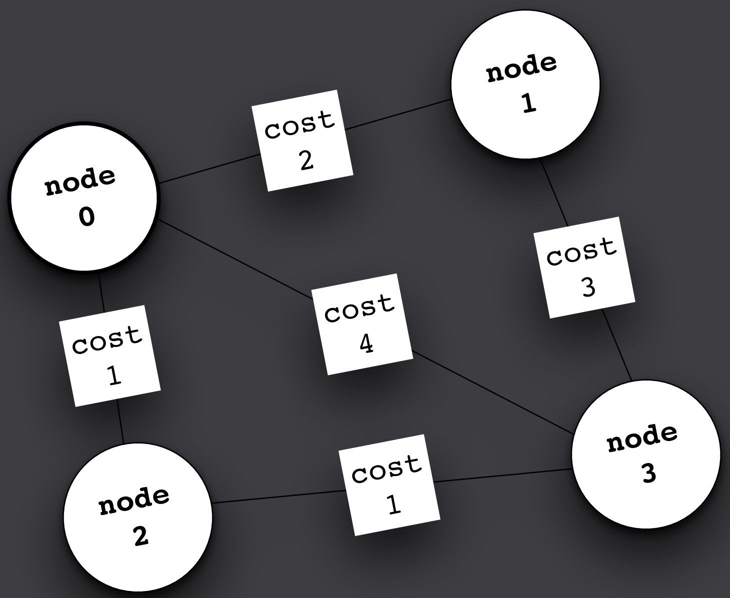
Training Survey

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

http://goo.gl/forms/oiA7YeO7VH

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

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

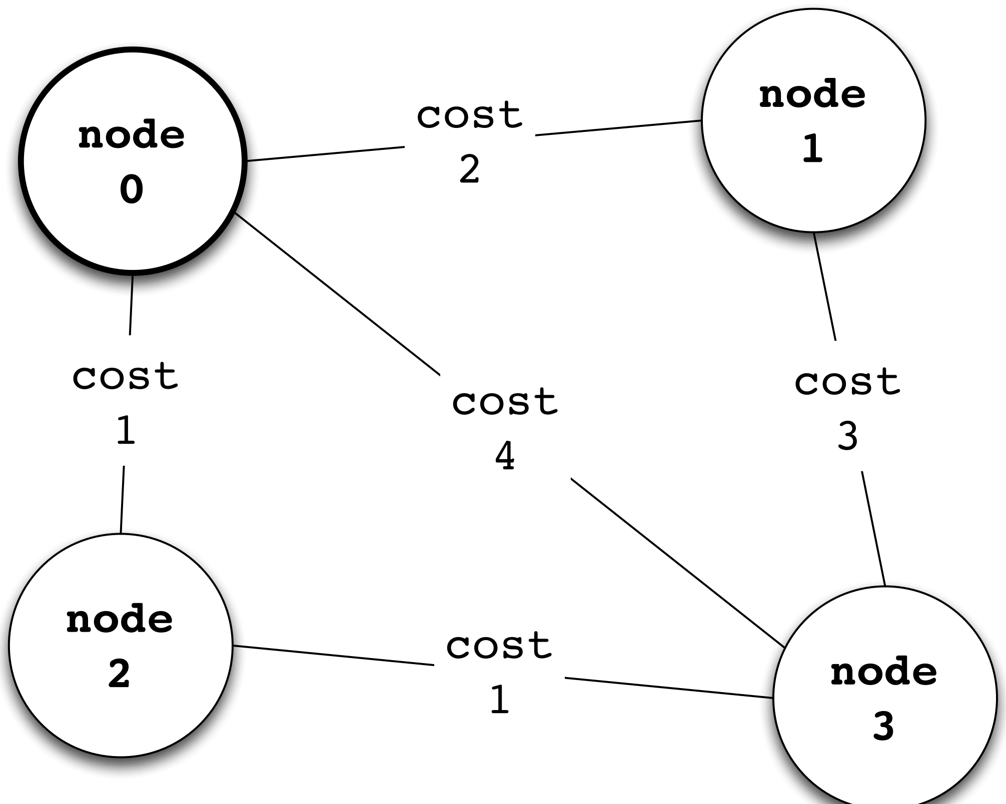
Advanced Exercises: GraphX
databricks-training.s3.amazonaws.com/graph-analytics-with-graphx.html

GraphX: *Example – simple traversals*

```scala
// 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}")
}
```

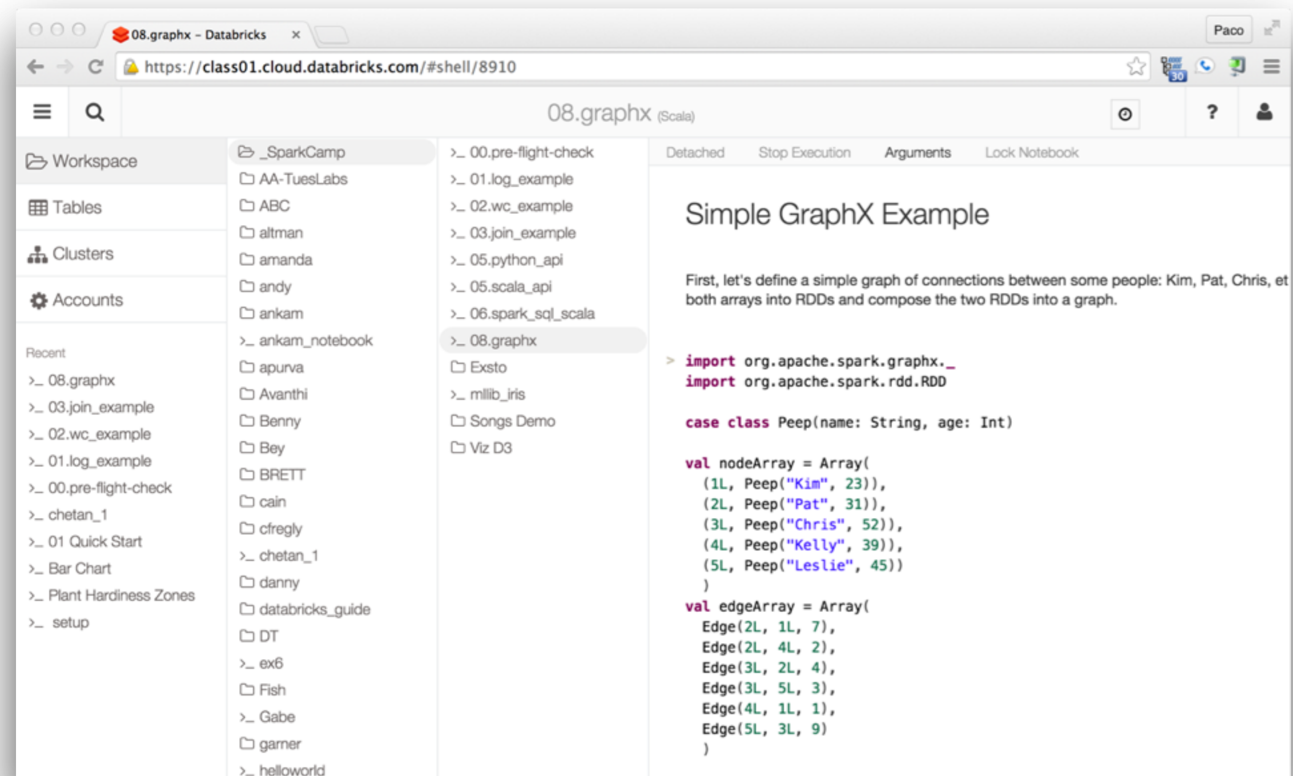
GraphX: *Example – routing problems*

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:



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

O'REILLY

databricks

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

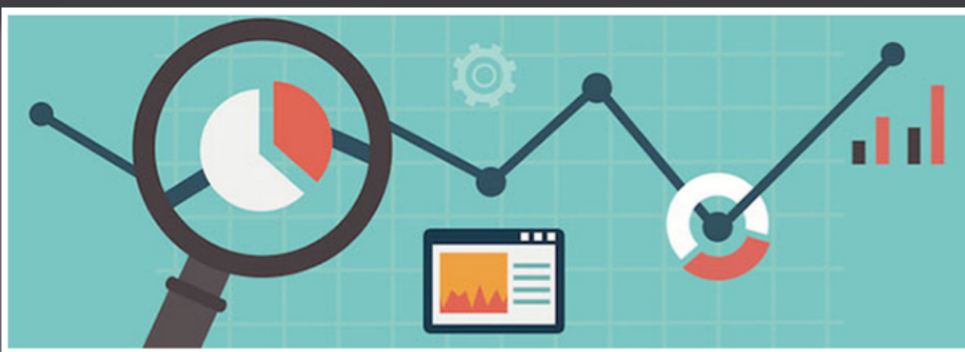
workshops: databricks.com/spark/training

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

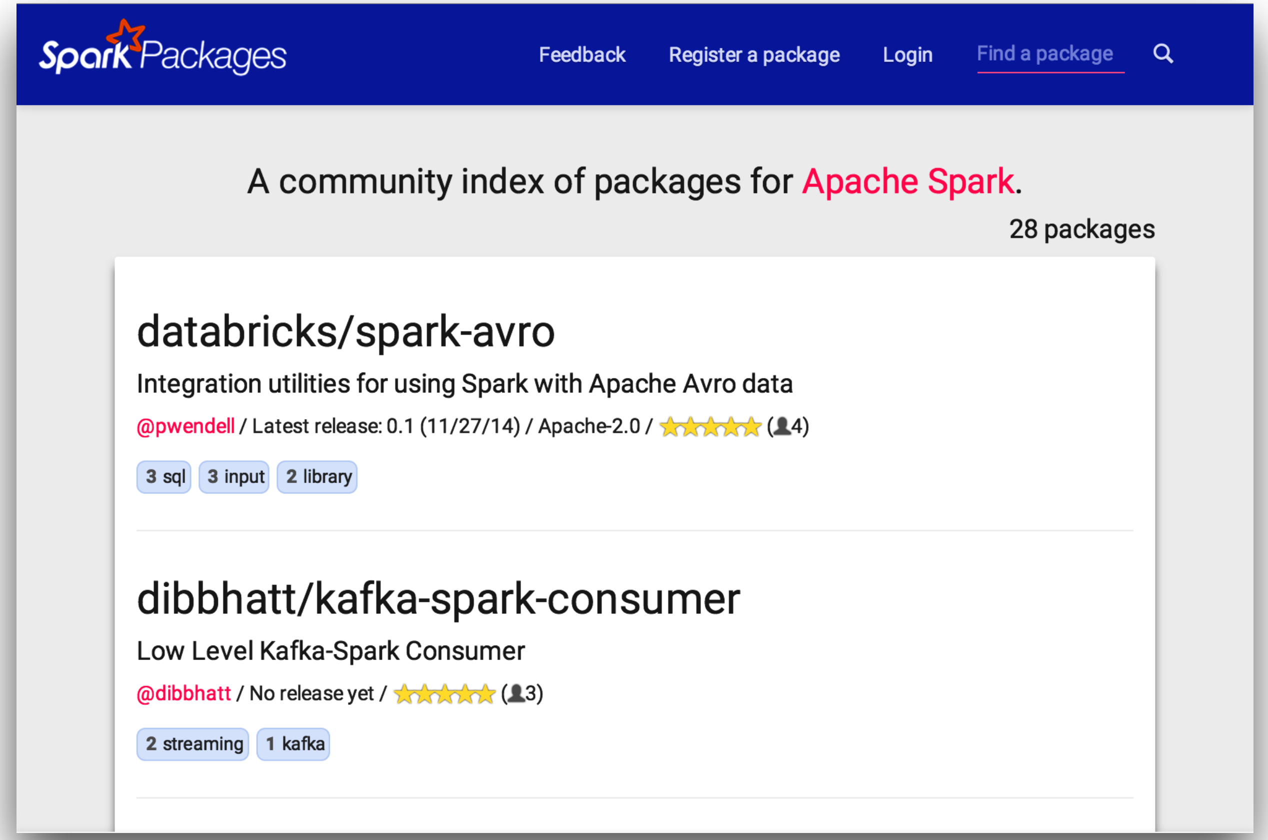
Learn the underlying principles required to develop scalable machine learning pipelines and gain hands-on experience using Apache Spark.

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:

Big Data Tech Con
Boston, Apr 26-28
bigdatatechcon.com

Strata EU
London, May 5-7
strataconf.com/big-data-conference-uk-2015

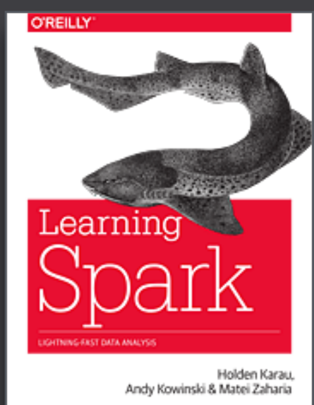
GOTO Chicago
Chicago, May 11-14
gotocon.com/chicago-2015

Spark Summit 2015
SF, Jun 15-17
spark-summit.org

http://spark-summit.org/

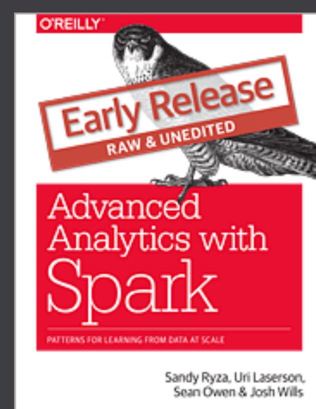
books+videos:

Learning Spark
Holden Karau,
Andy Konwinski,
Parick Wendell,
Matei Zaharia
O'Reilly (2015)
shop.oreilly.com/product/0636920028512.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