

Spark Tutorial @ DAO

download slides:

training.databricks.com/workshop/su_dao.pdf



Licensed under a Creative Commons Attribution-
NonCommercial-NoDerivatives 4.0 International License



Welcome + Getting Started



Getting Started: Step 1

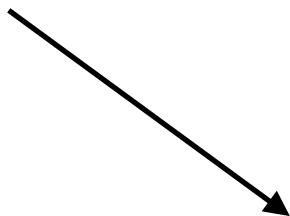
Everyone will receive a username/password for one of the Databricks Cloud shards. Use your laptop and browser to login there.

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

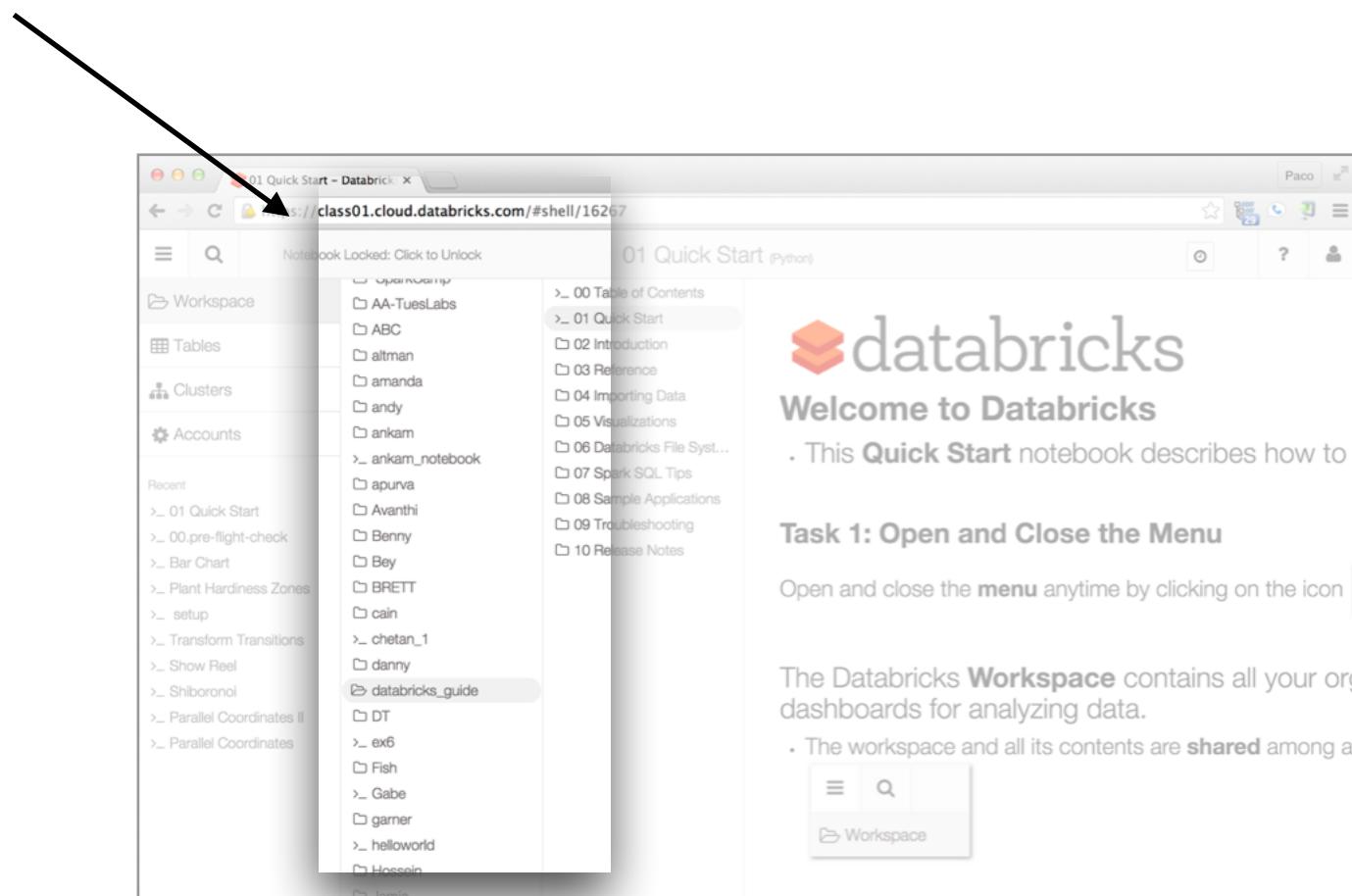
Open in a browser window, then click on the navigation menu in the top/left corner:



The screenshot shows a browser window titled "01 Quick Start - Databrick". The URL is <https://class01.cloud.databricks.com/#shell/16267>. The page content is the "01 Quick Start (Python)" notebook. On the left, there is a navigation sidebar with sections for Workspace, Tables, Clusters, and Accounts. The Accounts section is expanded, showing a list of users including AA-TuesLabs, ABC, alman, amanda, andandy, anikam, anikam_notebook, apurva, Avanthi, Benny, Bey, BRETT, cain, chetan_1, danny, databricks_guide, DT, ex6, Fish, Gabe, garner, helloworld, Hossein, and Jamie. A dropdown menu is open over the "01 Quick Start" item in the main content area, listing options like "Table of Contents", "Quick Start", "Introduction", etc. The main content area displays the "Welcome to Databricks" message and the "Task 1: Open and Close the Menu" section.

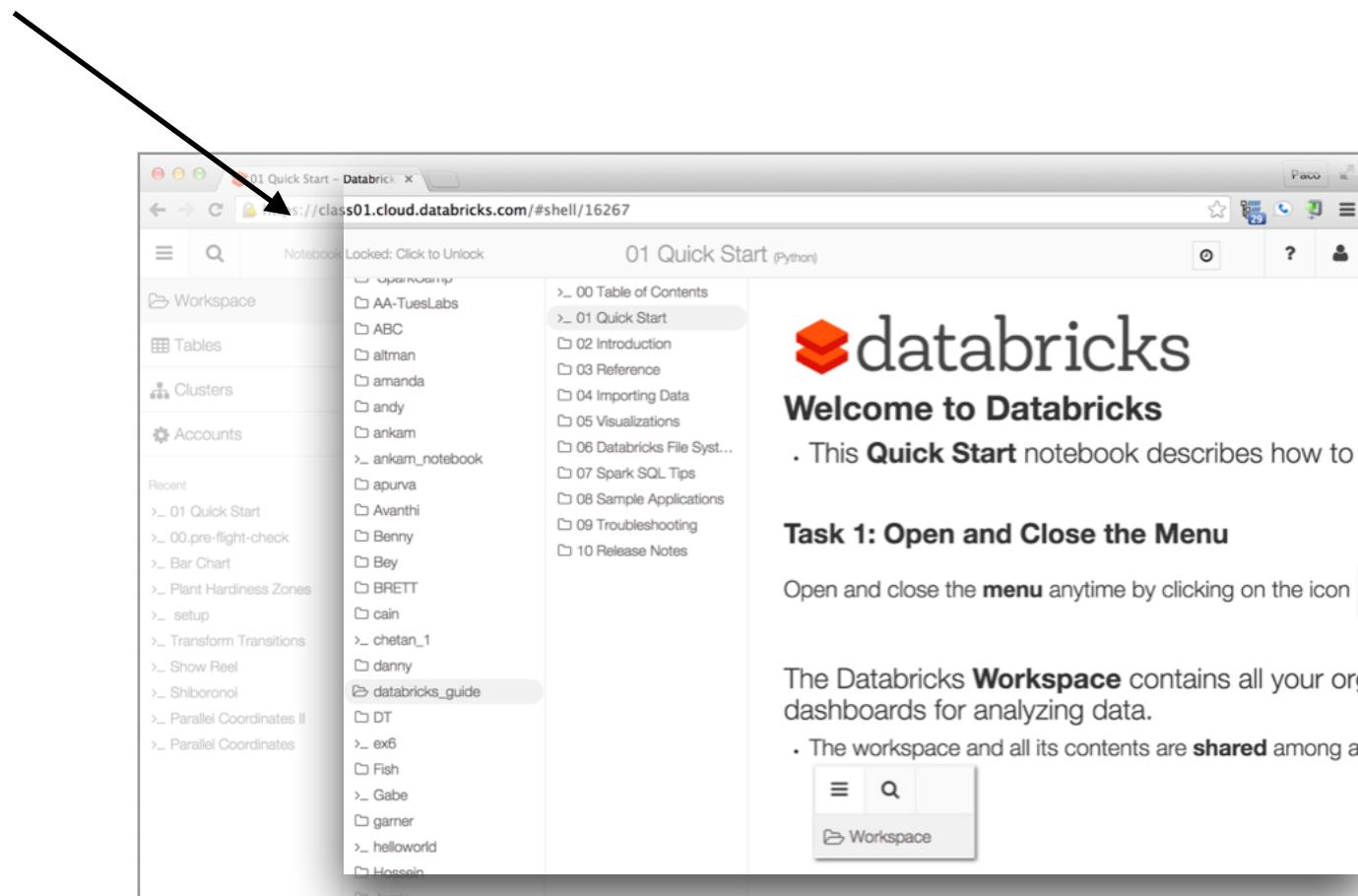
Getting Started: Step 3

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:

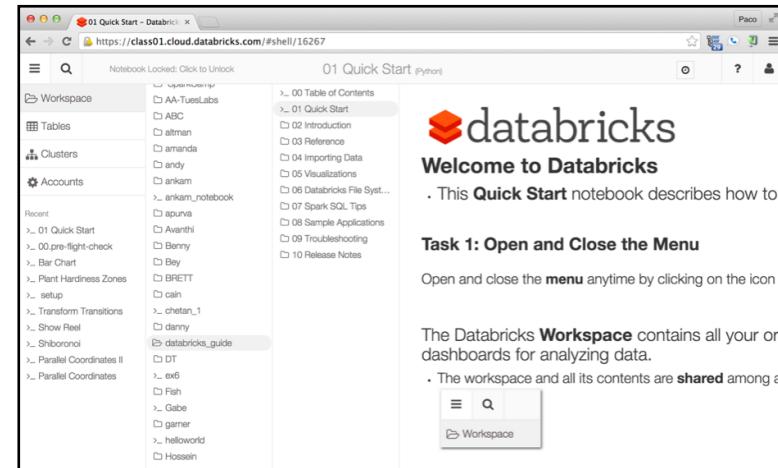


Getting Started: Step 5

See /databricks-guide/01 Quick Start

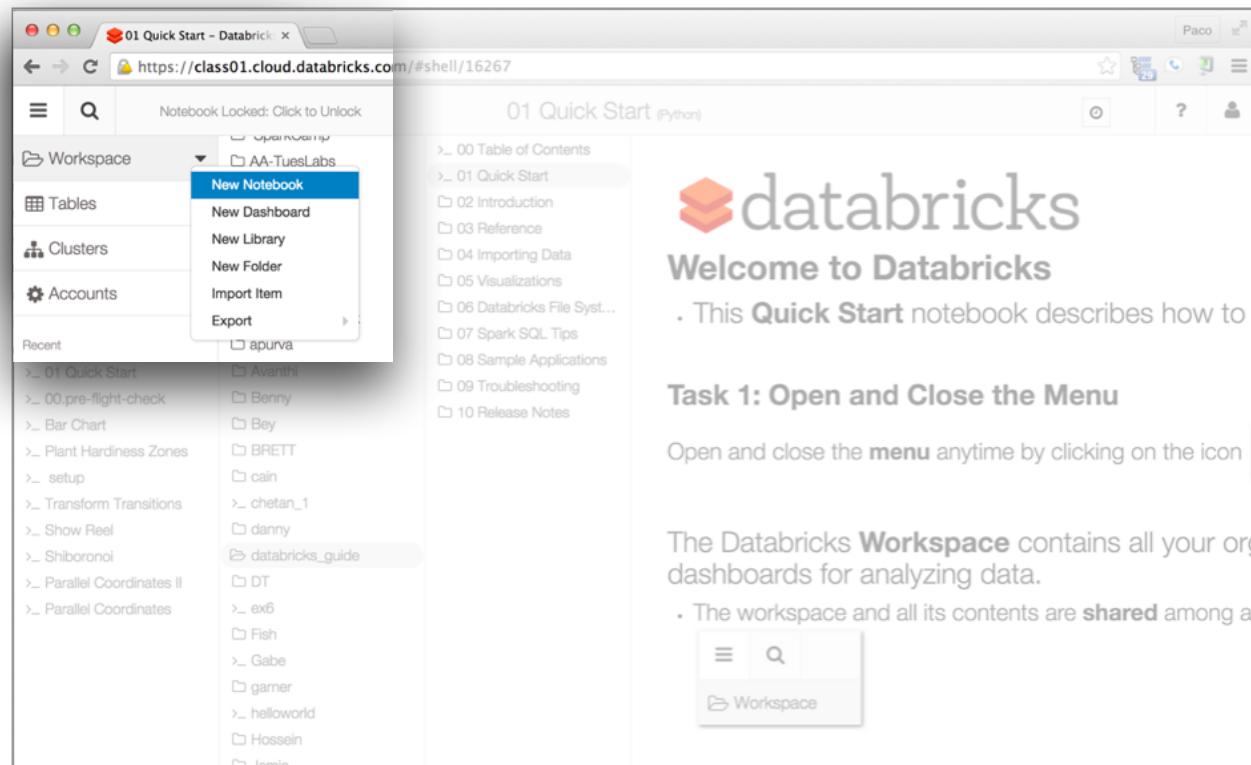
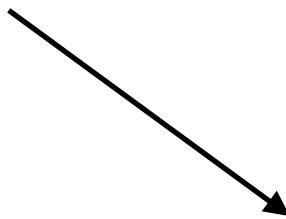
Key Features:

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



Getting Started: Step 6

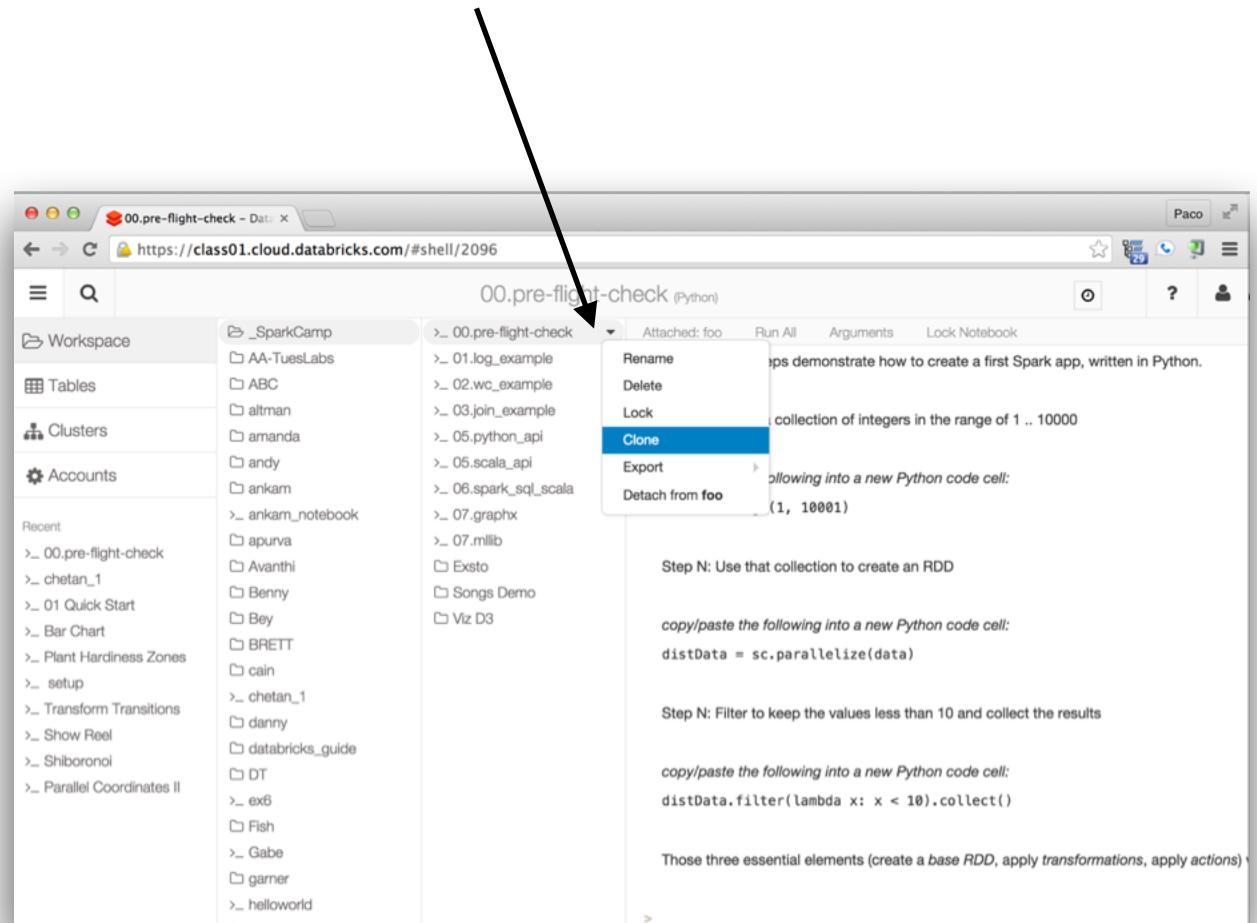
Click on the Workspace menu and create your own folder (pick a name):



The screenshot shows the Databricks workspace interface. On the left, there's a sidebar with 'Workspace', 'Tables', 'Clusters', and 'Accounts'. Below that is a 'Recent' section with various notebook names like '01 Quick Start', '00.pre-flight-check', etc. A context menu is open over the 'Workspace' item, with 'New Folder' highlighted. The main content area shows a '01 Quick Start (Python)' notebook with sections for Table of Contents, Introduction, Reference, Importing Data, Visualizations, Databricks File System, Spark SQL Tips, Sample Applications, Troubleshooting, and Release Notes. At the bottom, there's a 'Task 1: Open and Close the Menu' section with instructions and a note about workspace sharing.

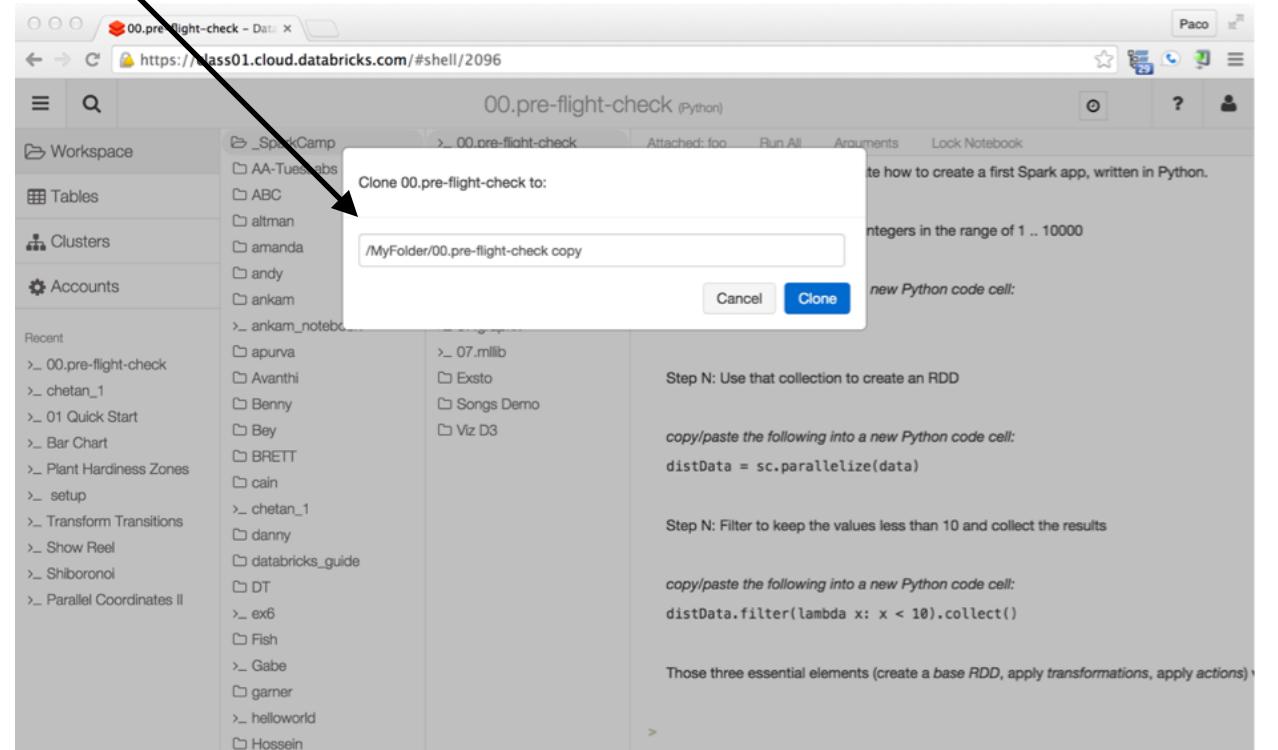
Getting Started: Step 7

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



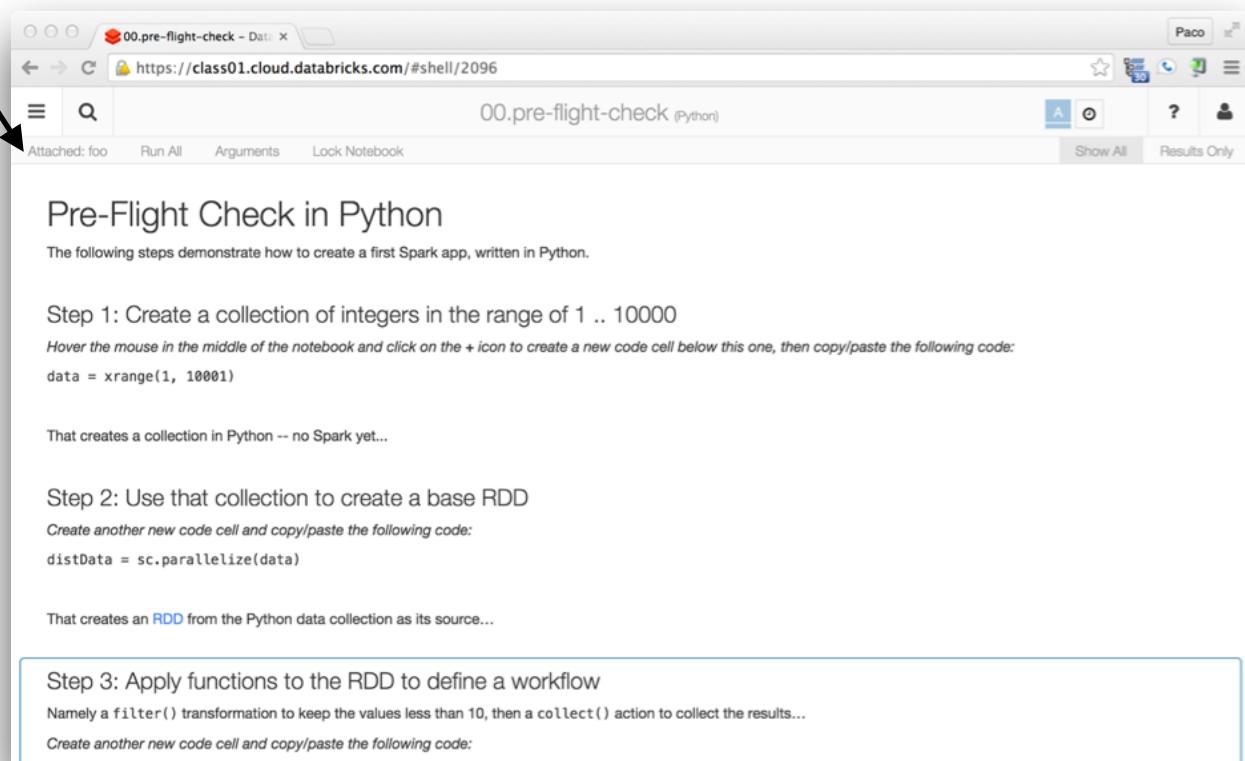
Getting Started: Step 8

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*:



Attached: foo Run All Arguments Lock Notebook

Pre-Flight Check in Python

The following steps demonstrate how to create a first Spark app, written in Python.

Step 1: Create a collection of integers in the range of 1 .. 10000

Hover the mouse in the middle of the notebook and click on the + icon to create a new code cell below this one, then copy/paste the following code:

```
data = xrange(1, 10001)
```

That creates a collection in Python -- no Spark yet...

Step 2: Use that collection to create a base RDD

Create another new code cell and copy/paste the following code:

```
distData = sc.parallelize(data)
```

That creates an RDD from the Python data collection as its source...

Step 3: Apply functions to the RDD to define a workflow

Namely a filter() transformation to keep the values less than 10, then a collect() action to collect the results...

Create another new code cell and copy/paste the following code:

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:

The screenshot shows a Databricks notebook interface with the title "00.pre-flight-check" and a sub-page header "00.pre-flight-check (Python)". The notebook content is a "Pre-Flight Check in Python" section. It provides instructions for creating a first Spark app in Python, starting with generating a collection of integers from 1 to 10,000. The code cell for this step is shown as:

```
data = xrange(1, 10001)
```

Below the code, it says "That creates a collection in Python -- no Spark yet...". The next step involves using this collection to create a base RDD, with the corresponding code being:

```
distData = sc.parallelize(data)
```

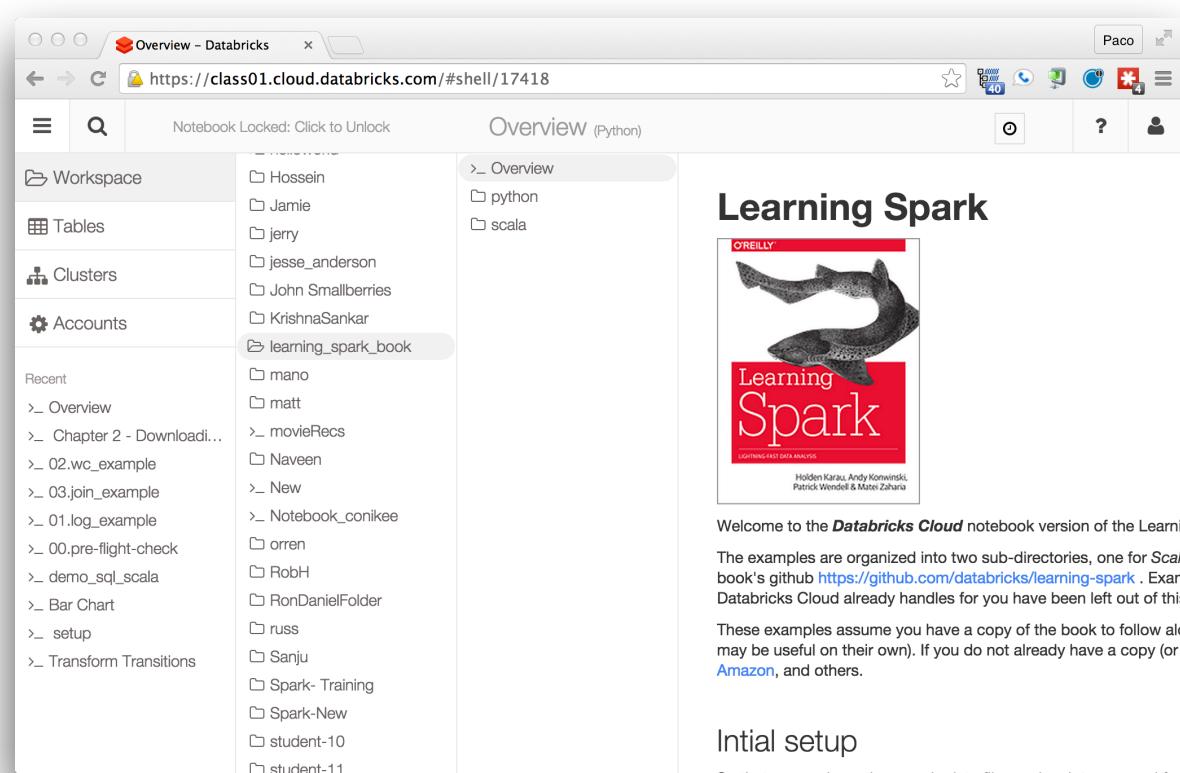
It notes that "That creates an RDD from the Python data collection as its source...". The final step, highlighted with a blue border, is to apply functions to the RDD to define a workflow, with the code being:

```
distData.filter(lambda x: x < 10).collect()
```

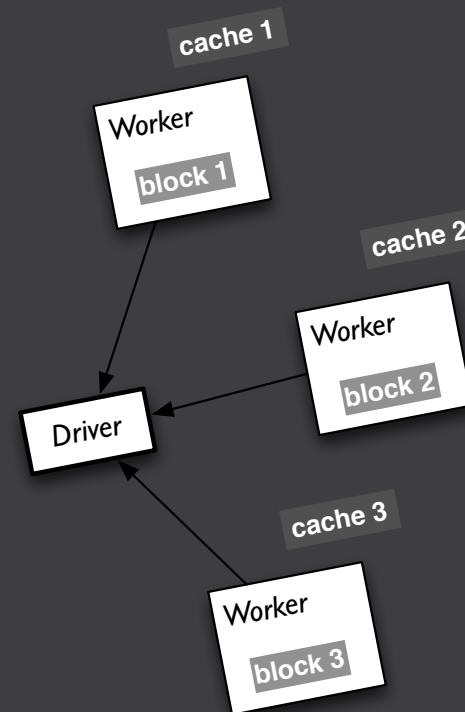
Instructions for creating a new code cell and copying/pasting the code are provided throughout the steps.

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:

The screenshot shows a Databricks notebook interface with the title "01.log_example (Python)". The left sidebar displays a file tree with various notebooks and log files. The main area contains Python code for reading a log file, filtering errors, and persisting the results.

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

Command took 0.04s

We apply some transformations to filter the log lines that contain errors, then just keep the

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

Command took 0.03s

We will use the messages RDD multiple times, so we persist it into memory using a call to

> messages.cache()

Out[3]: PythonRDD[219] at RDD at PythonRDD.scala:43

Command took 0.04s

Now, for the first action we will filter the errors that include the keyword mysql and count t
```

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

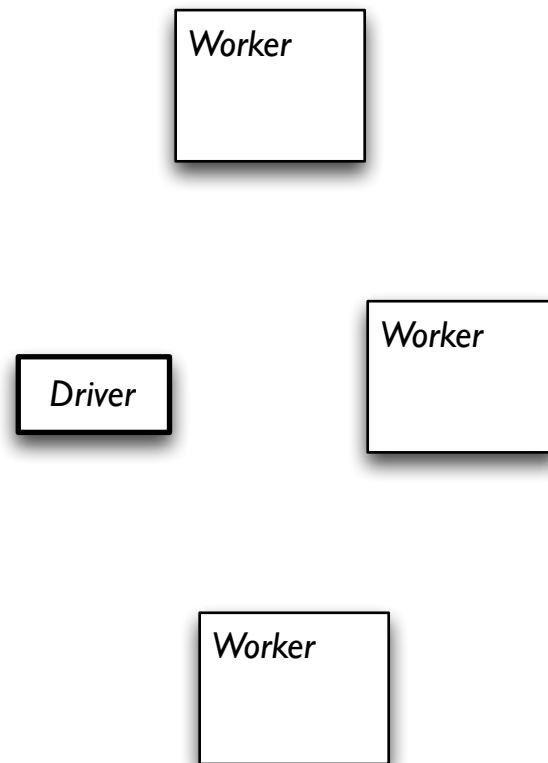
Note that we can examine the *operator graph* for a transformed RDD, for example:

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

Worker

Worker

Driver

Worker

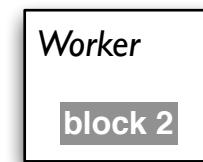
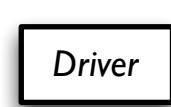
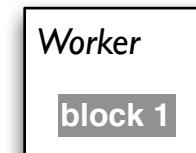
Spark Deconstructed: Log Mining Example

```
# 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"))

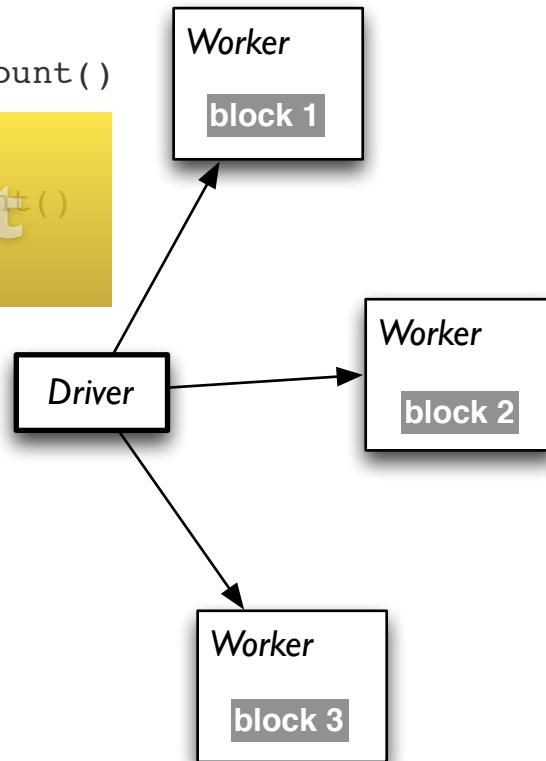
# transformed RDDs
errors = lines.filter(lambda x: x[0] == "ERROR")
messages = errors.map(lambda x: x[1])

# 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"))

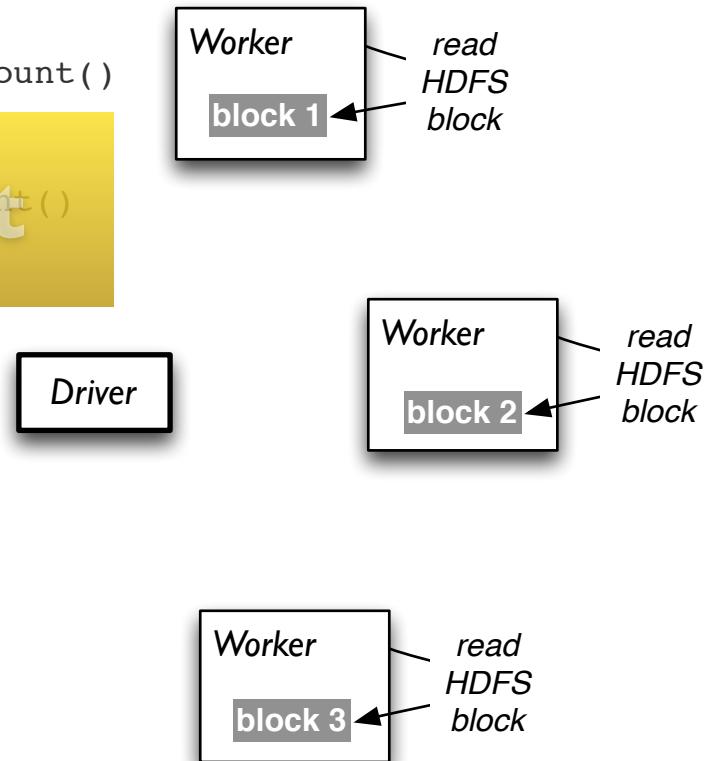
# transformed RDDs
errors = lines.filter(lambda x: x[0] == "ERROR")
messages = errors.map(lambda x: x[1])

# 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

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

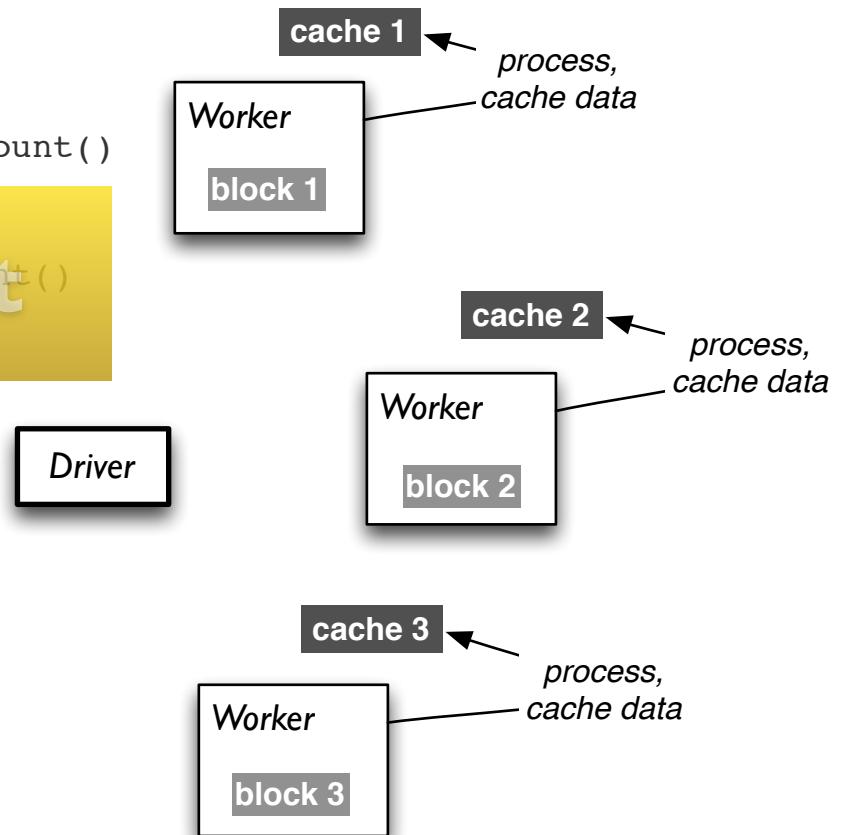
# transformed RDDs
errors = lines.filter(lambda x: x[0] == "ERROR")
messages = errors.map(lambda x: x[1])

# 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"))

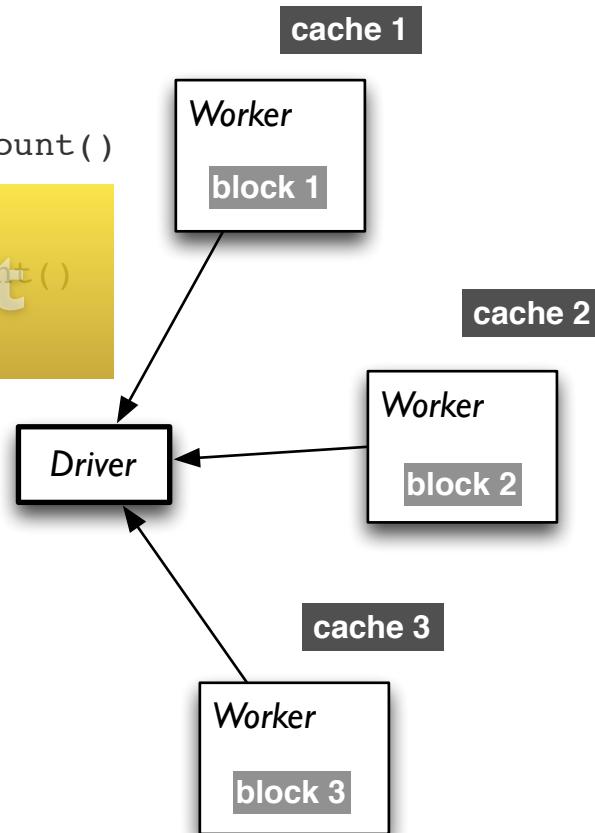
# transformed RDDs
errors = lines.filter(lambda x: x[0] == "ERROR")
messages = errors.map(lambda x: x[1])

# 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

cache 1



cache 2



cache 3



Spark Deconstructed: Log Mining Example

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

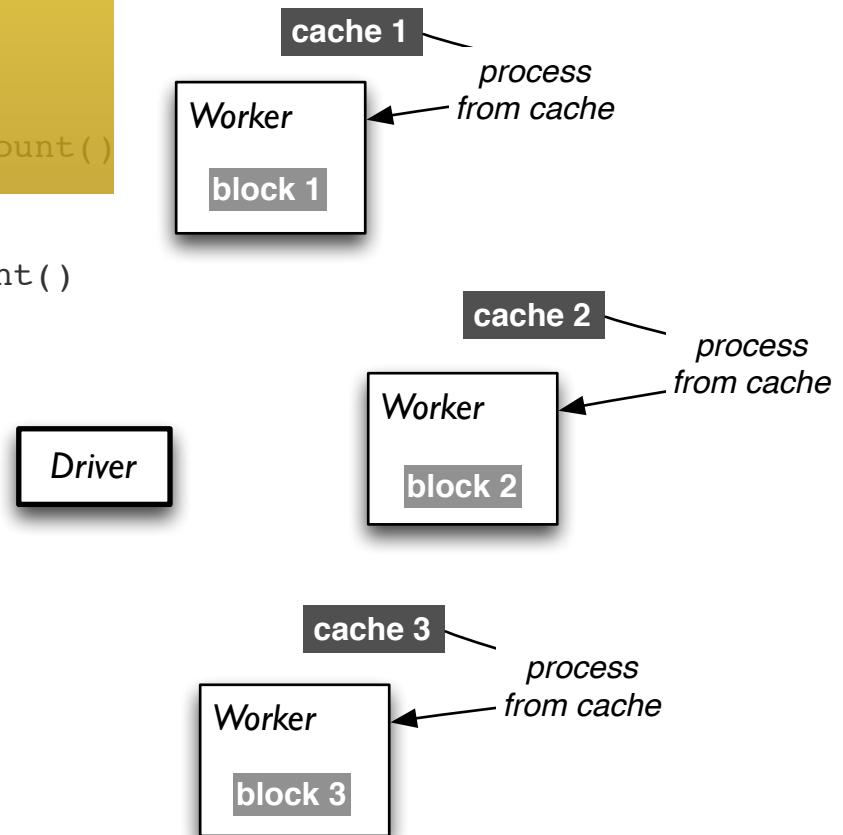
# transformed RDDs
errors = lines.filter(lambda x: x[0] == "ERROR")
messages = errors.map(lambda x: x[1])

# 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"))

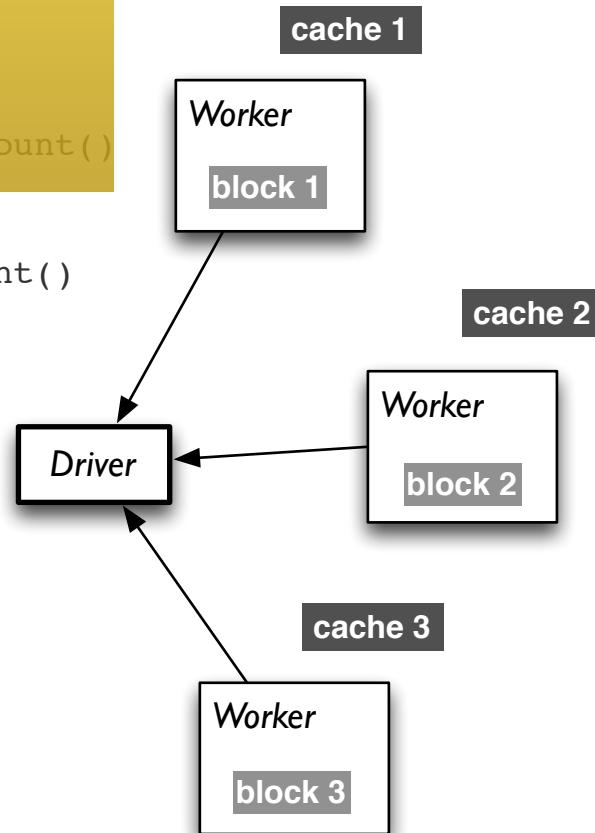
# transformed RDDs
errors = lines.filter(lambda x: x[0] == "ERROR")
messages = errors.map(lambda x: x[1])

# 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...

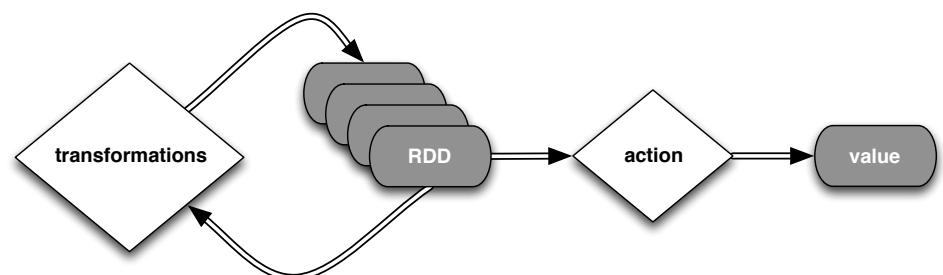
```
# base RDD
lines = sc.textFile("/mnt/paco/intro/error_log.txt") \
    .map(lambda x: x.split("\t"))

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

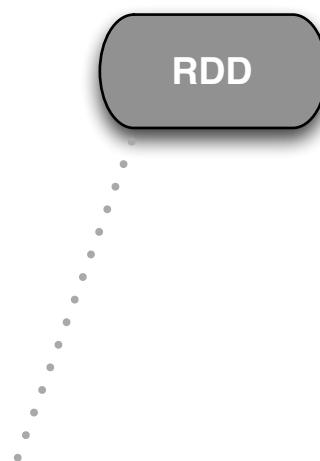
# persistence
messages.cache()

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

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

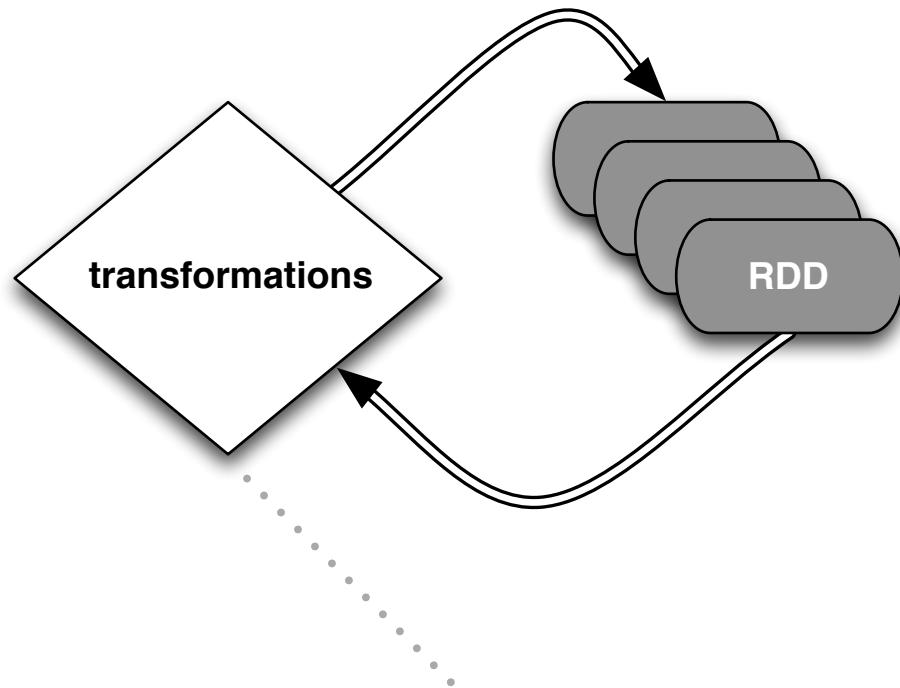


Spark Deconstructed: Log Mining Example



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

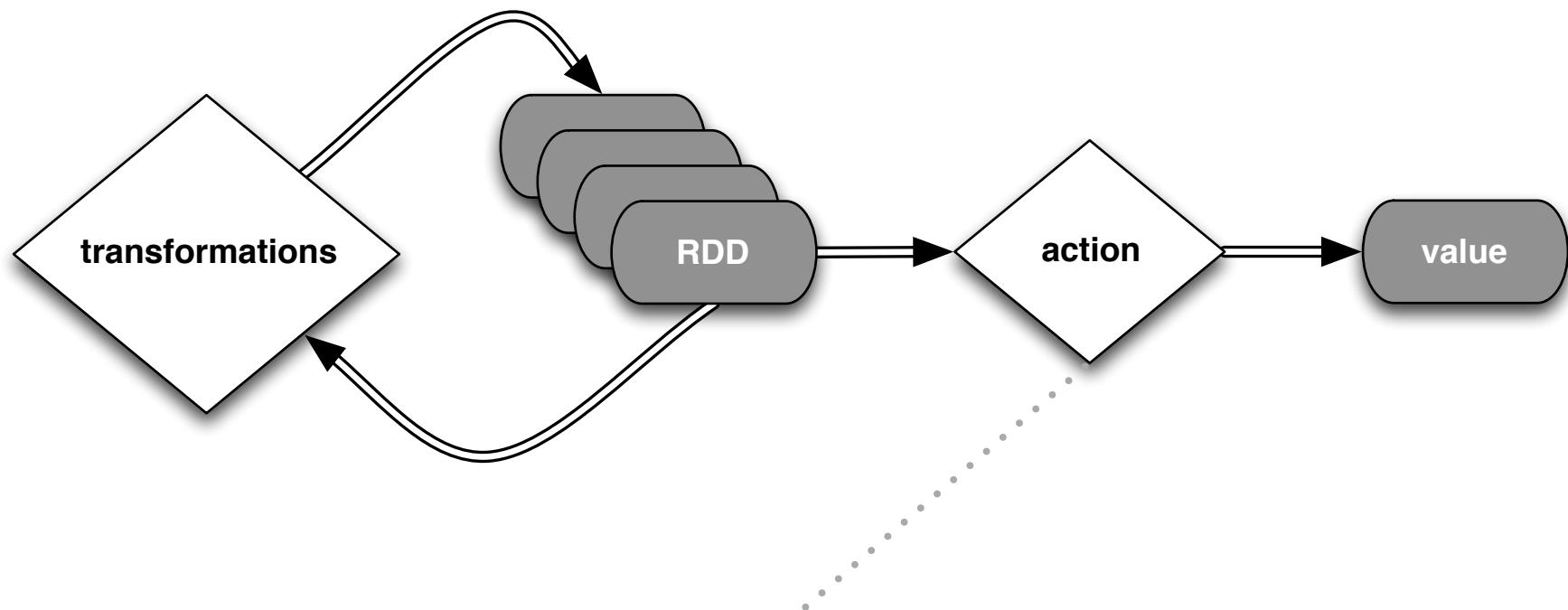
Spark Deconstructed: Log Mining Example



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

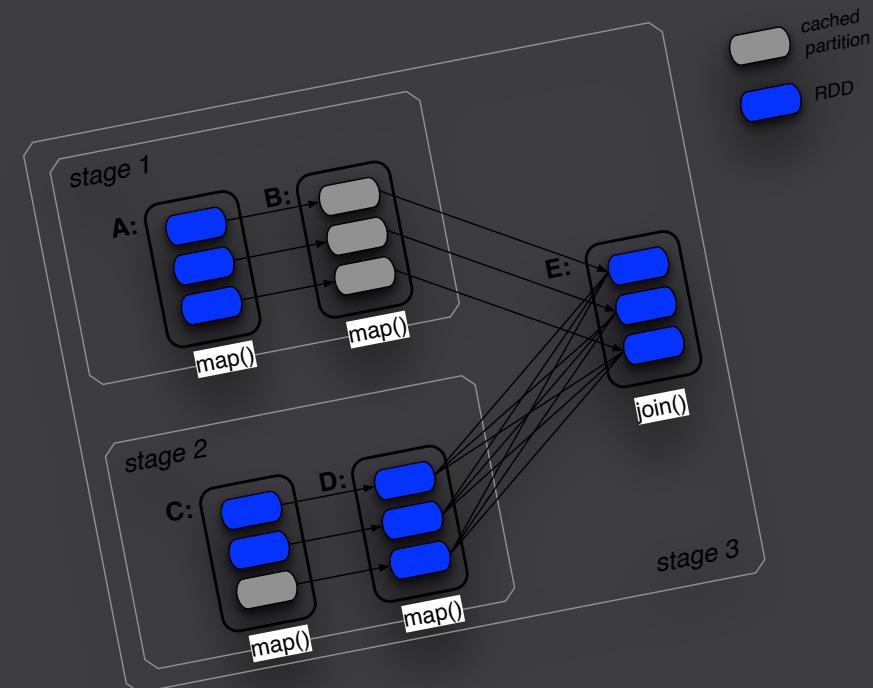
# persistence
messages.cache()
```

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

```
1 public class WordCount {
2     public static class TokenizerMapper
3         extends Mapper<Object, Text, Text, IntWritable> {
4
5     private final static IntWritable one = new IntWritable(1);
6     private Text word = new Text();
7
8     public void map(Object key, Text value, Context context
9                     ) throws IOException, InterruptedException {
10        StringTokenizer itr = new StringTokenizer(value.toString());
11        while (itr.hasMoreTokens()) {
12            word.set(itr.nextToken());
13            context.write(word, one);
14        }
15    }
16
17
18    public static class IntSumReducer
19        extends Reducer<Text,IntWritable,Text,IntWritable> {
20        private IntWritable result = new IntWritable();
21
22        public void reduce(Text key, Iterable<IntWritable> values,
23                           Context context
24                           ) throws IOException, InterruptedException {
25            int sum = 0;
26            for (IntWritable val : values) {
27                sum += val.get();
28            }
29            result.set(sum);
30            context.write(key, result);
31        }
32    }
33
34    public static void main(String[] args) throws Exception {
35        Configuration conf = new Configuration();
36        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
37        if (otherArgs.length < 2) {
38            System.err.println("Usage: wordcount <in> [<in>...] <out>");
39            System.exit(2);
40        }
41        Job job = new Job(conf, "word count");
42        job.setJarByClass(WordCount.class);
43        job.setMapperClass(TokenizerMapper.class);
44        job.setCombinerClass(IntSumReducer.class);
45        job.setReducerClass(IntSumReducer.class);
46        job.setOutputKeyClass(Text.class);
47        job.setOutputValueClass(IntWritable.class);
48        for (int i = 0; i < otherArgs.length - 1; ++i) {
49            FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
50        }
51        FileOutputFormat.setOutputPath(job,
52            new Path(otherArgs[otherArgs.length - 1]));
53        System.exit(job.waitForCompletion(true) ? 0 : 1);
54    }
55 }
```

```
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:

The screenshot shows a Databricks workspace interface. On the left, there's a sidebar with sections for Workspace, Tables, Clusters, and Accounts. Under Recent, several notebooks are listed, including `02.wc_example`. The main area is titled "02.wc_example (Python)". At the top right, there are buttons for Attached: foo, Run All, Arguments, and Lock Notebook. Below the title, the text "WordCount Example" is displayed. The notebook content starts with a code cell:

```
> lines = sc.textFile("/mnt/paco/intro/README.md")
lines.take(2)
```

The output of this cell is:

```
Out[1]: [u'Apache Spark', u'']
```

Followed by a note: "Command took 0.72s". Then, another code cell is shown:

```
> from operator import add
```

The output of this cell is: "Command took 0.02s". A note explains: "We use flatMap() to split the text lines into a sequence of keywords, then transform into keyword and sum them." Finally, the last part of the code is:

```
> wc = lines.flatMap(lambda x: x.split(' '))
    .map(lambda x: (x, 1)).reduceByKey(add)
```

Coding Exercise: *Join*

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

The screenshot shows a Databricks workspace interface. On the left, there's a sidebar with sections for Workspace, Tables, Clusters, and Accounts. Under Recent, several notebooks are listed, including `03.join_example`. The main area is titled "03.join_example (Python)". At the top right, there are buttons for Attached: foo, Run All, Arguments, and Lock Notebook. Below the title, a section titled "Join Example" contains the following text: "The following example shows how to do a simple `join()` of two RDDs in Spark. As an example, we'll join two datasets of user impressions, clicks, registrations, etc., typically must be joined at scale." A code snippet follows:

```
> clk = sc.textFile("/mnt/paco/intro/join/clk.tsv") \
    .map(lambda x: x.split("\t"))

Command took 0.04s

Let's use collect() to inspect the data. Assume that those fields represent date, user_id, clicks, and impressions.
```

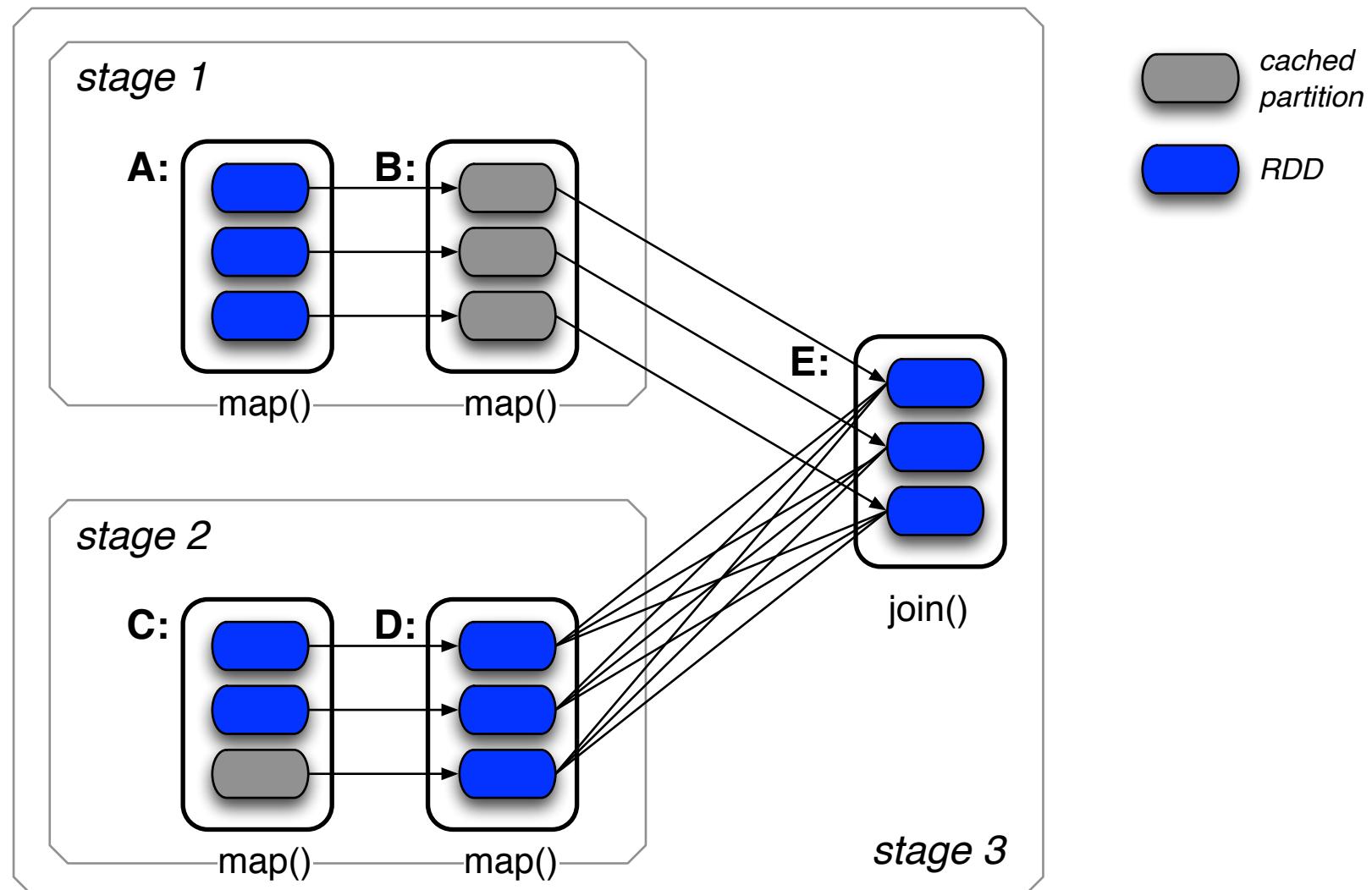
Output:

```
Out[2]:
[[u'2014-03-04', u'15dfb8e6cc4111e3a5bb600308919594', u'11'],
 [u'2014-03-06', u'81da510acc4111e387f3600308919594', u'61']]
```

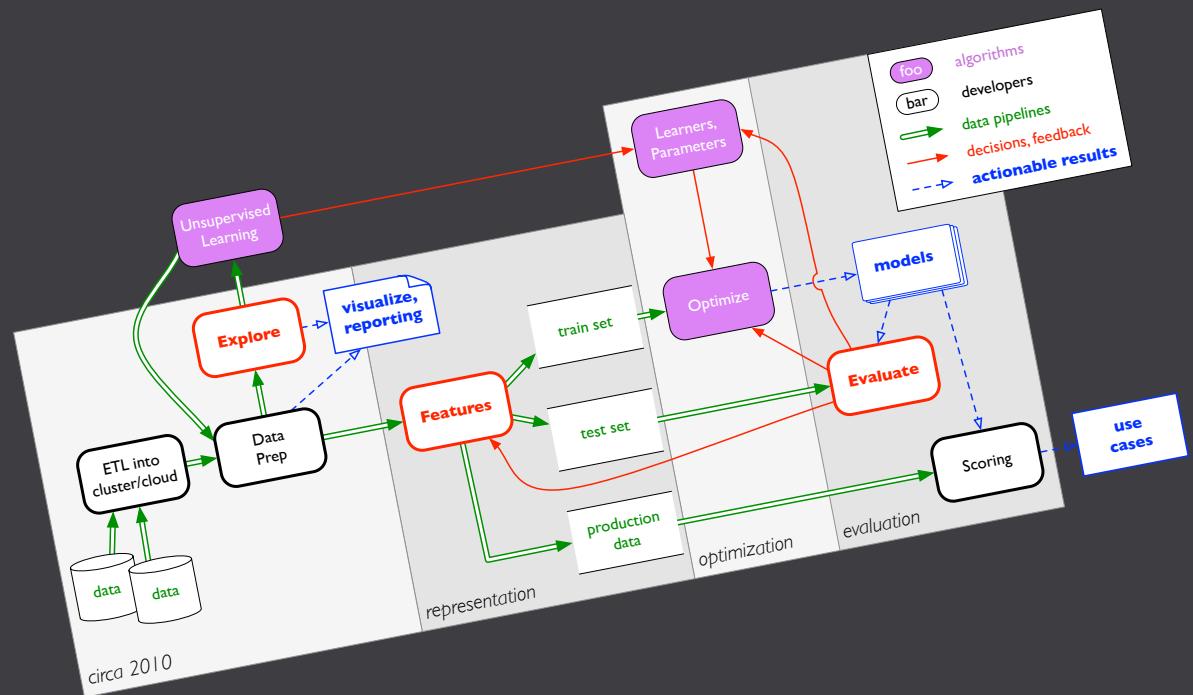
Command took 1.01s

To perform a join in Spark, the RDD data must be reshaped as key-value pairs:

Coding Exercise: Join and its Operator Graph



DBC Essentials



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	<i>an instance of Databricks Workspace</i>
Cluster	<i>a Spark cluster (multiple per shard)</i>
Notebook	<i>a list of markdown, executable commands, and results</i>
Dashboard	<i>a flexible space to create operational visualizations</i>

DBC Essentials: Notebooks

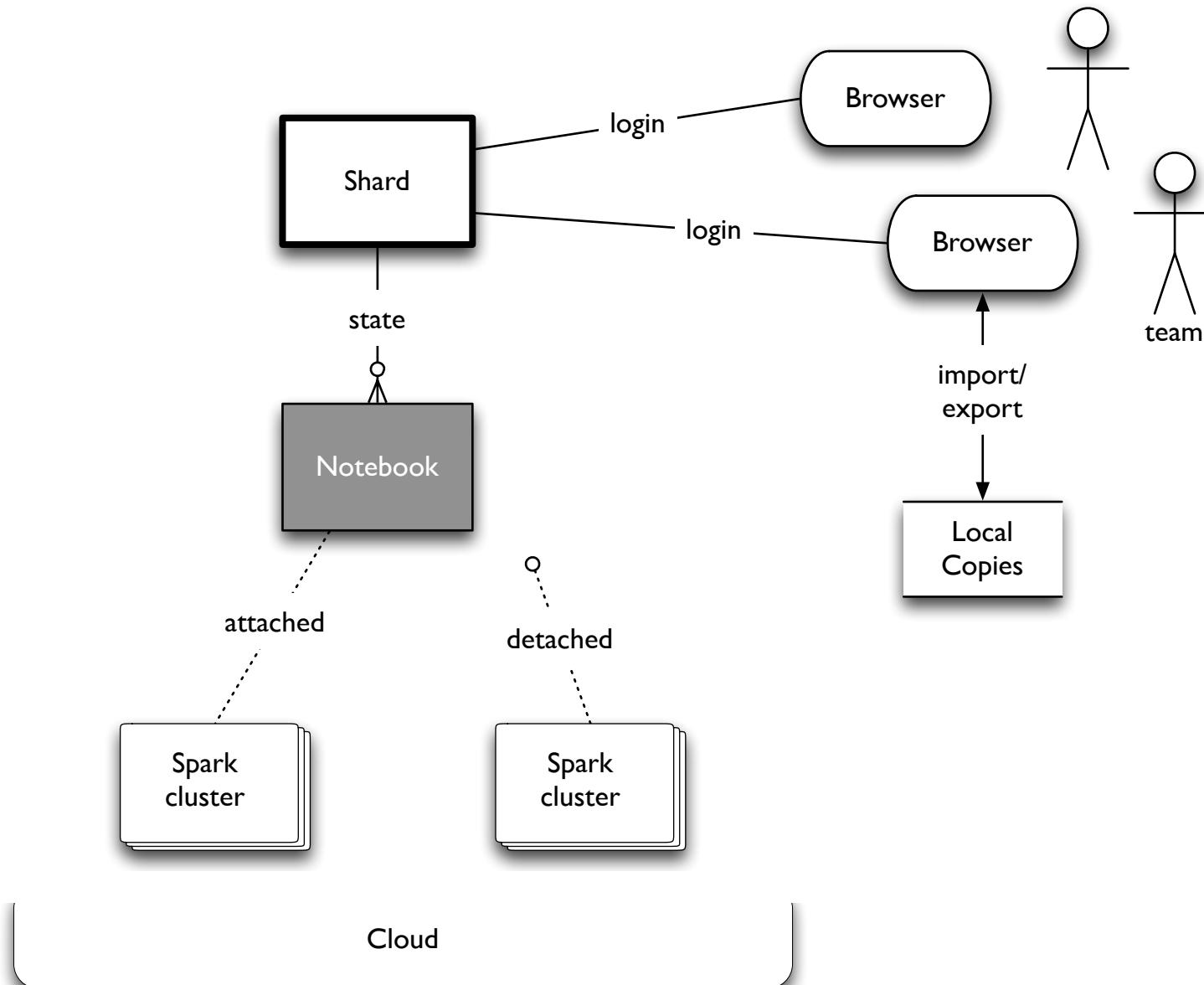
- Series of commands (think shell++)
- Each notebook has a language type, chosen at notebook creation:
 - Python + SQL
 - Scala + SQL
 - SQL only
- Command output captured in notebook
- Commands can be...
 - edited, reordered, rerun, exported, cloned, imported, etc.

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



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

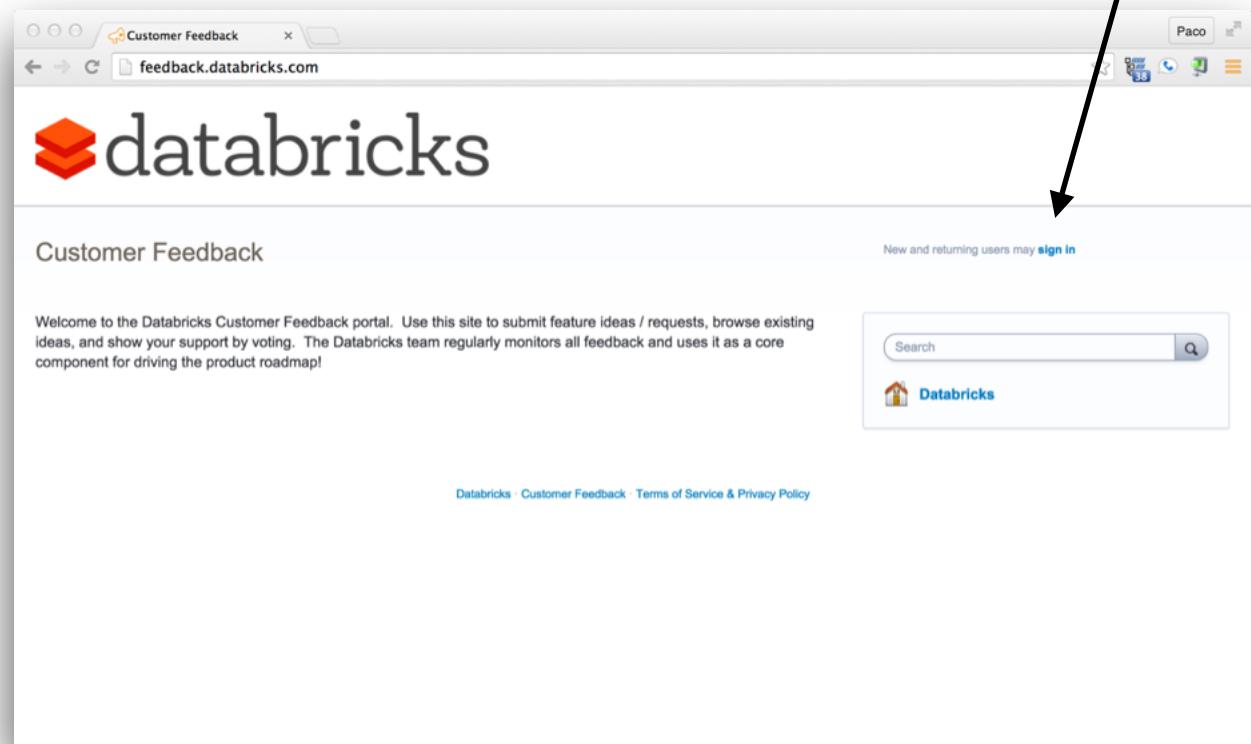
DBC Essentials: Feedback

Other feedback, suggestions, etc.?

<http://feedback.databricks.com/>

<http://forums.databricks.com/>

UserVoice login in
top/right corner...



How to “Think Notebooks”

The screenshot shows a Databricks notebook interface with the following details:

- Left Sidebar:** Shows the workspace structure with sections like Workspace, Tables, Clusters, Accounts, Recent, and a list of notebooks.
- Current Notebook:** Titled "06.spark_sql_scala" (Scala), it displays a Scala code snippet for defining a simple array of people and creating an RDD from it.
- Code Snippet:**

```
> val people_data = Array( ("Michael", 29), ("Andy", 30), ("Justin", 19) )
people_data: Array[(String, Int)] = Array((Michael,29), (Andy,30), (Justi
Command took 0.36s

First, let's define a very simple array of people: their names and ages...
> case class Peep(name: String, age: Int)
defined class Peep
Command took 0.29s

Next, we use a case class in Scala to define schema for that data...
> val people = sc.parallelize(people_data).map(p => Peep(p._1, p._2))
people.collect()
people: org.apache.spark.rdd.RDD[Peep] = MappedRDD@345 at map at <com
res1: Array[Peep] = Array(Peep(Michael,29), Peep(Andy,30), Peep(Justin,1
Command took 0.36s
```

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

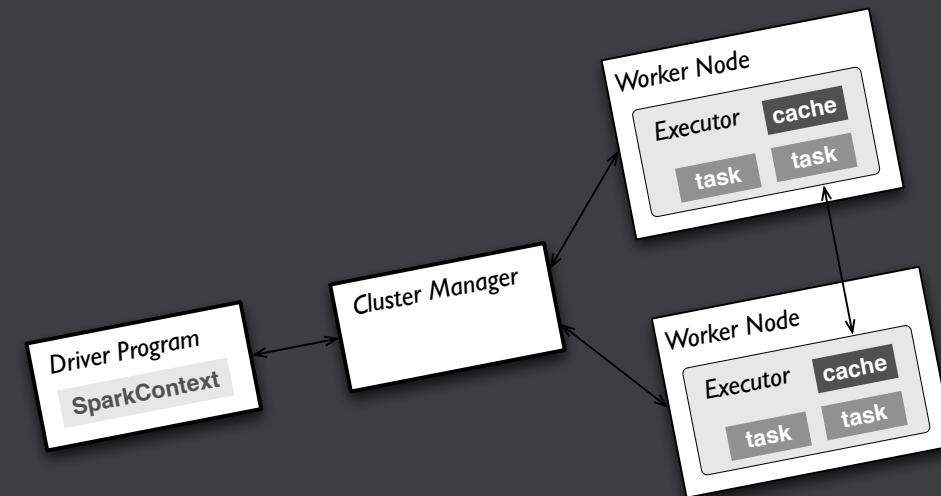
Let's assemble the pieces of the previous few code examples, using two files:

/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 0x7ff0bf818a10>
```

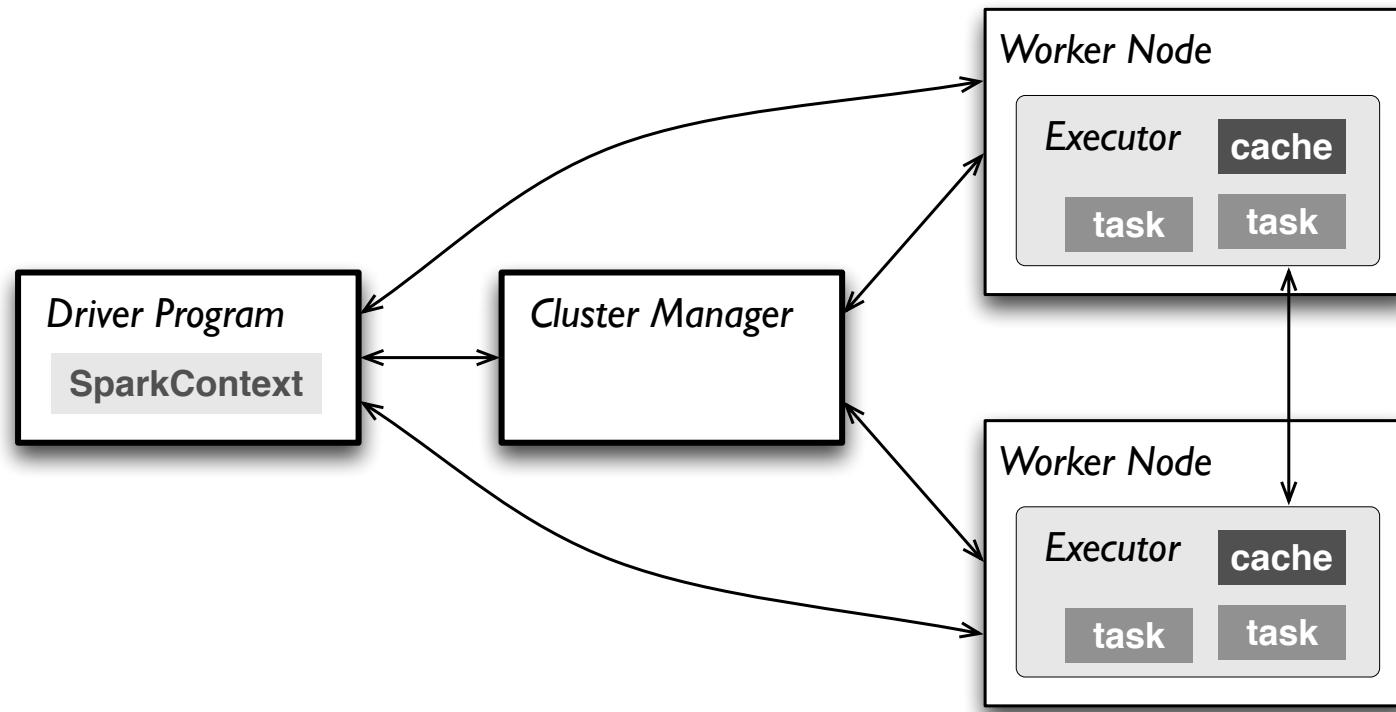
Spark Essentials: Master

The master parameter for a SparkContext determines which cluster to use

<i>master</i>	<i>description</i>
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

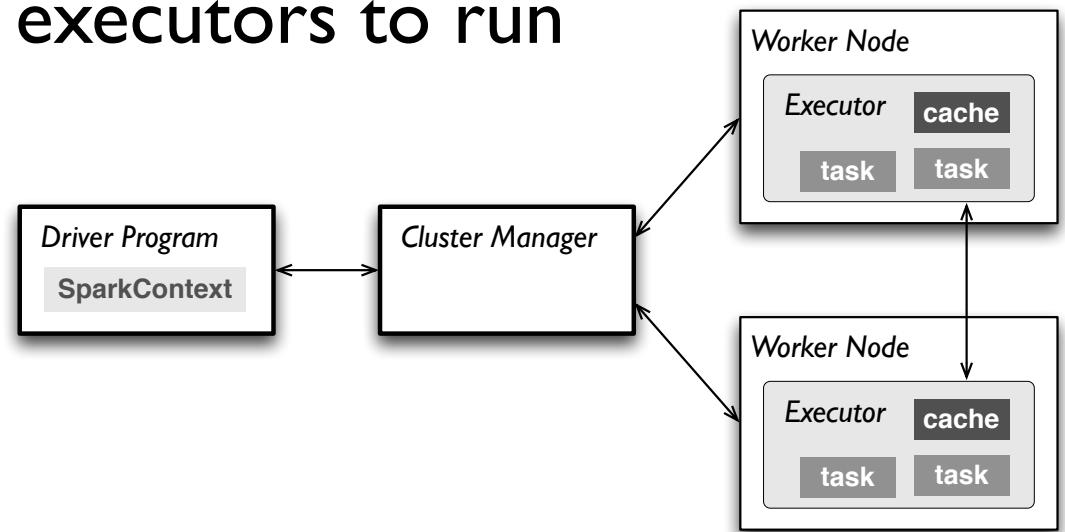
spark.apache.org/docs/latest/cluster-overview.html



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:

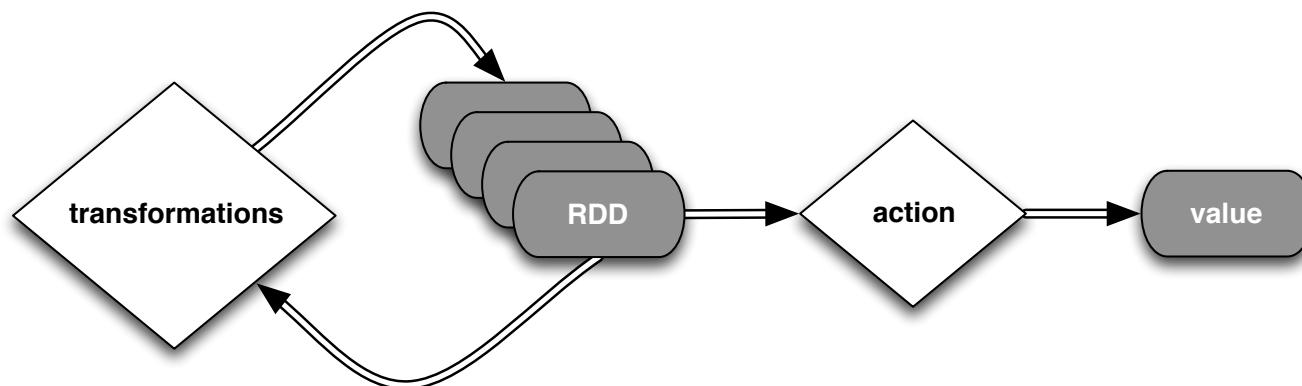
```
data = [1, 2, 3, 4, 5]
data
Out[2]: [1, 2, 3, 4, 5]

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

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

Spark Essentials: Transformations

<i>transformation</i>	<i>description</i>
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()
```

Python:

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

distFile is a collection of lines



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(<i>func</i>)	aggregate the elements of the dataset using a function <i>func</i> (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 <i>take(1)</i>
take(<i>n</i>)	return an array with the first <i>n</i> elements of the dataset – currently not executed in parallel, instead the driver program computes all the elements
takeSample(<i>withReplacement</i>, <i>fraction</i>, <i>seed</i>)	return an array with a random sample of <i>num</i> 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 <code>toString</code> on each element to convert it to a line of text in the file
saveAsSequenceFile(path)	write the elements of the dataset as a Hadoop SequenceFile in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. Only available on RDDs of key-value pairs that either implement Hadoop's Writable interface or are implicitly convertible to Writable (Spark includes conversions for basic types like Int, Double, String, etc).
countByKey()	only available on RDDs of type (K, V). Returns a `Map` of (K, Int) pairs with the count of each key
foreach(func)	run a function <code>func</code> on each element of the dataset – usually done for side effects such as updating an accumulator variable or interacting with external storage systems

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

<i>transformation</i>	<i>description</i>
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
```

Spark Essentials: *Broadcast Variables and Accumulators*

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

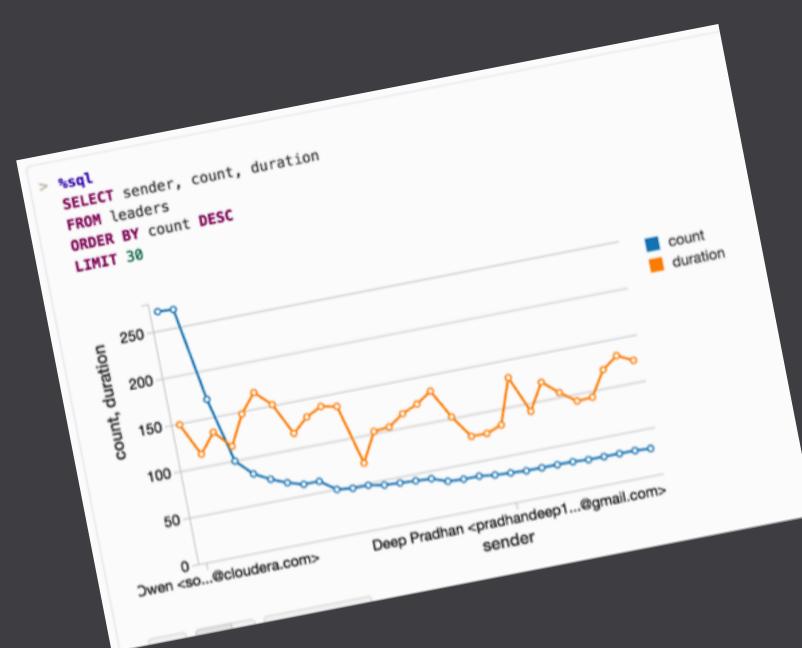
For more details about the Scala API:

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



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

Spark SQL: *Data Workflows – Parquet*

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/the-210pledem](https://www.slideshare.net/julienledem/the-210pledem)



Spark SQL: SQL Demo

Demo of /_SparkCamp/demo_sql_scala
by the instructor:

The screenshot shows a Databricks notebook interface. On the left is a sidebar with sections for Workspace, Tables, Clusters, and Accounts. The workspace contains several notebooks and examples. The main area is titled "06.spark_sql_scala (Scala)". It contains the following Scala code and output:

```
Spark SQL
First, let's define a very simple array of people: their names and ages...

> val people_data = Array( ("Michael", 29), ("Andy", 30), ("Justin", 19) )
people_data: Array[(String, Int)] = Array((Michael,29), (Andy,30), (Just
Command took 0.36s

Next, we use a case class in Scala to define schema for that data...

> case class Peep(name: String, age: Int)
defined class Peep
Command took 0.29s

Great, then create an RDD and overlay the schema using the case class...

> val people = sc.parallelize(people_data).map(p => Peep(p._1, p._2))
people: org.apache.spark.rdd.RDD[Peep] = MappedRDD[345] at map at <cons
res1: Array[Peep] = Array(Peep(Michael,29), Peep(Andy,30), Peep(Justin,1
Command took 0.36s
```

Spark SQL: Using DBFS

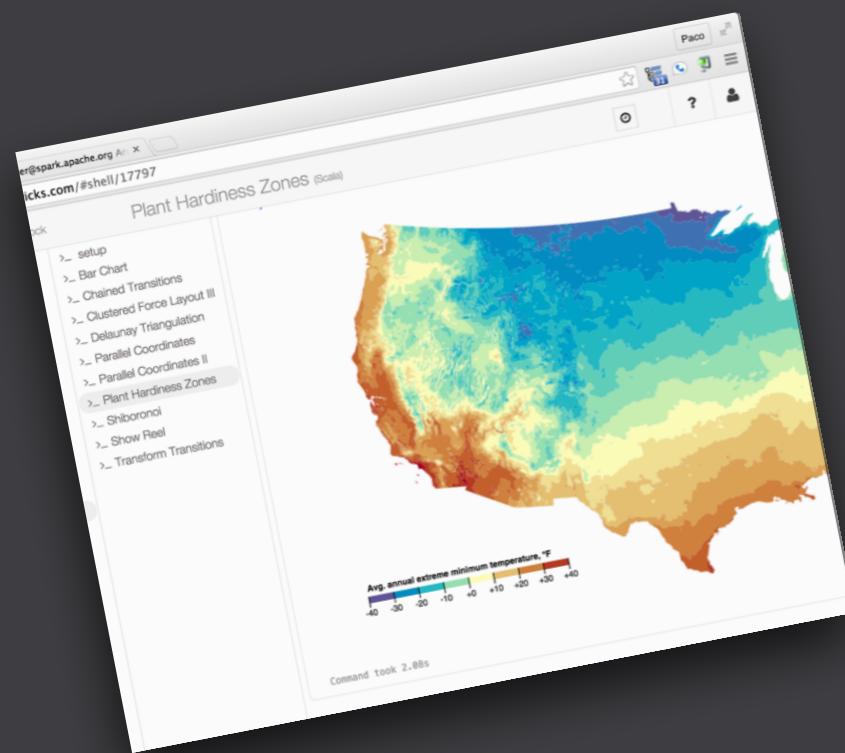
Next, we'll review the following sections in the *Databricks Guide*:

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

Key Topics:

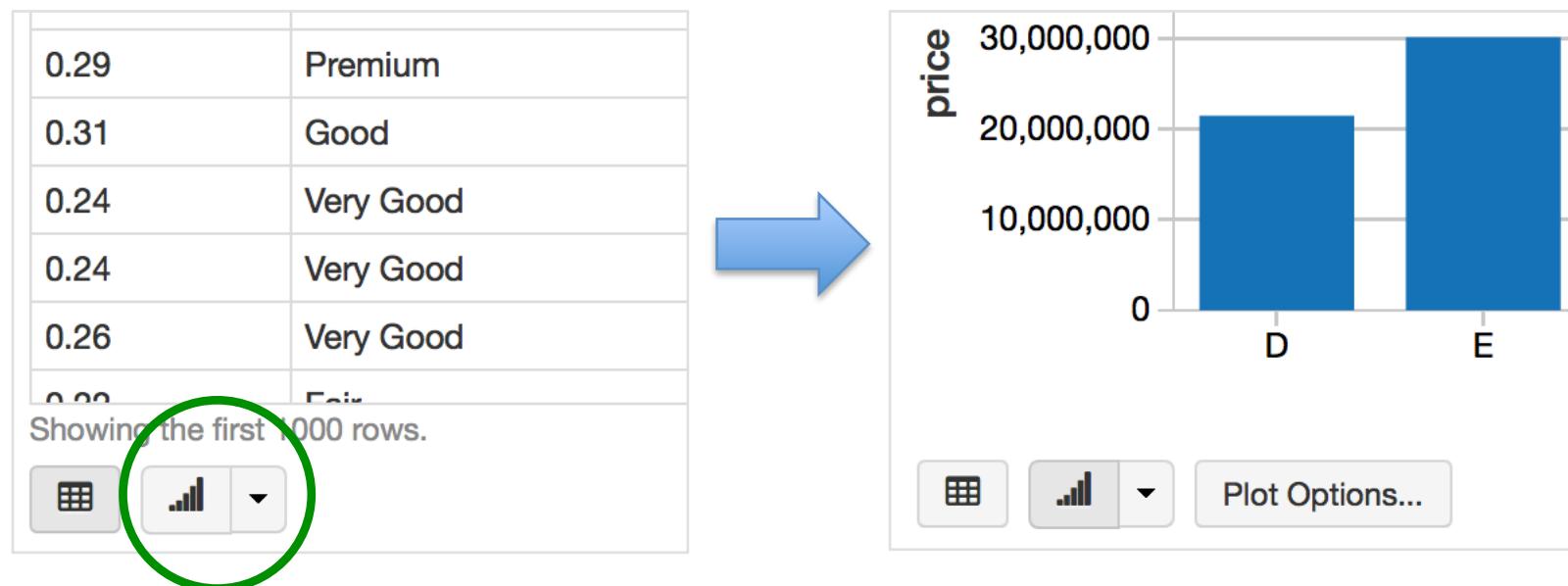
- JSON, CSV, Parquet
- S3, Hive, Redshift
- DBFS, dbutils

Visualization



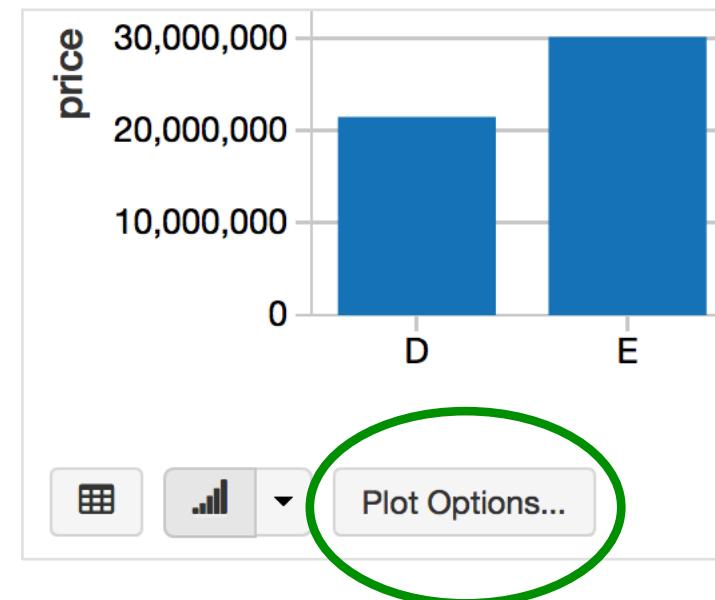
Visualization: Built-in Plots

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...

Keys:

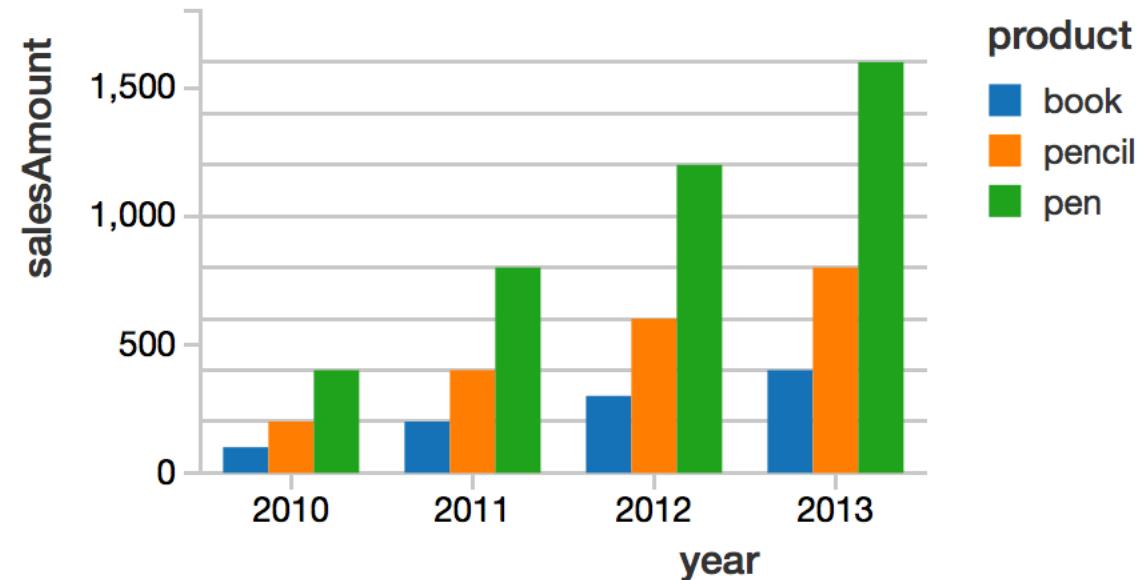
- year x

Series groupings:

- product x

Values:

- salesAmount x



Visualization: Reference Guide

See /databricks-guide/05 visualizations for details about built-in visualizations and extensions...

The screenshot shows a Databricks notebook interface. The left sidebar contains sections for Workspace, Tables, Clusters, and Accounts, along with a Recent section listing various notebooks. The main content area is titled "1 Visualizations in SQL (SQL)". The sidebar navigation includes:

- 00 Table of Contents
- 01 Quick Start
- 02 Introduction
- 03 Reference
- 04 Importing Data
- 05 Visualizations
- 06 Databricks File System
- 07 Spark SQL Tips
- 08 Sample Applications
- 09 Troubleshooting
- 10 Release Notes

The "05 Visualizations" item is highlighted. To the right of the sidebar, the main content area displays the title "Visualizations in SQL" and a bulleted list:

- This notebook covers how to

Below this, under "Results of select statements", there is another bulleted list:

- Visualizations basically come for free

A code snippet is shown:

```
> select * from SQLTestTable;
```

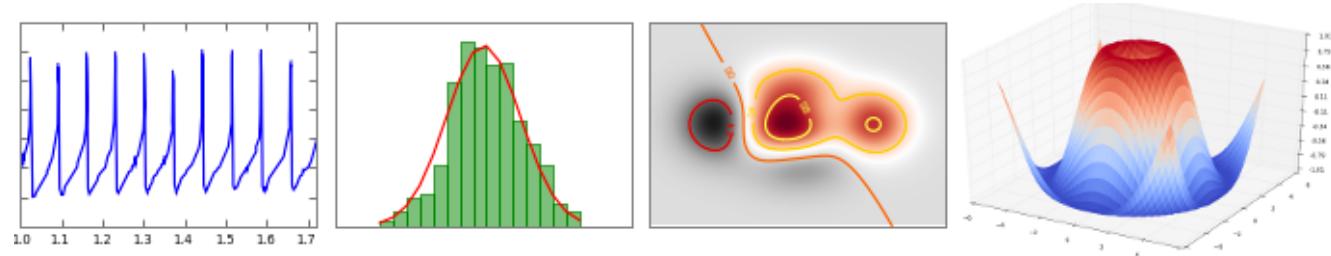
At the bottom, a bar chart is displayed with the following data:

Category	Value
aaa	1
bbb	2
ccc	3

Visualization: Using `display()`

The `display()` command:

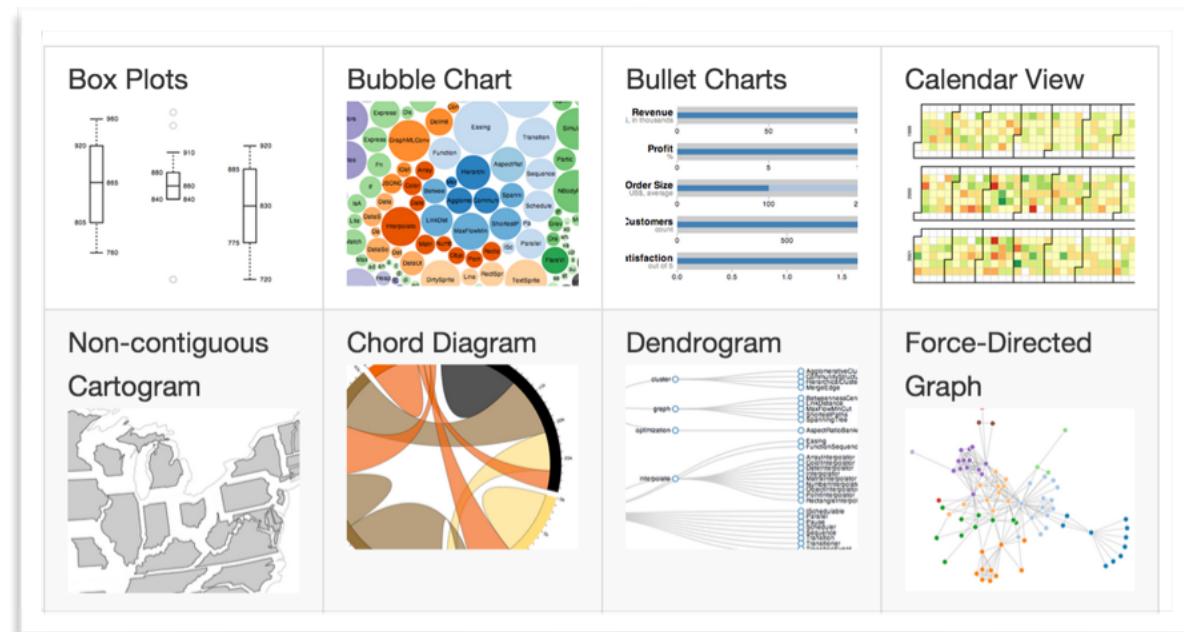
- programmatic access to visualizations
- pass a SchemaRDD to print as an HTML table
- pass a Scala list to print as an HTML table
- call without arguments to display **matplotlib** figures



Visualization: Using `displayHTML()`

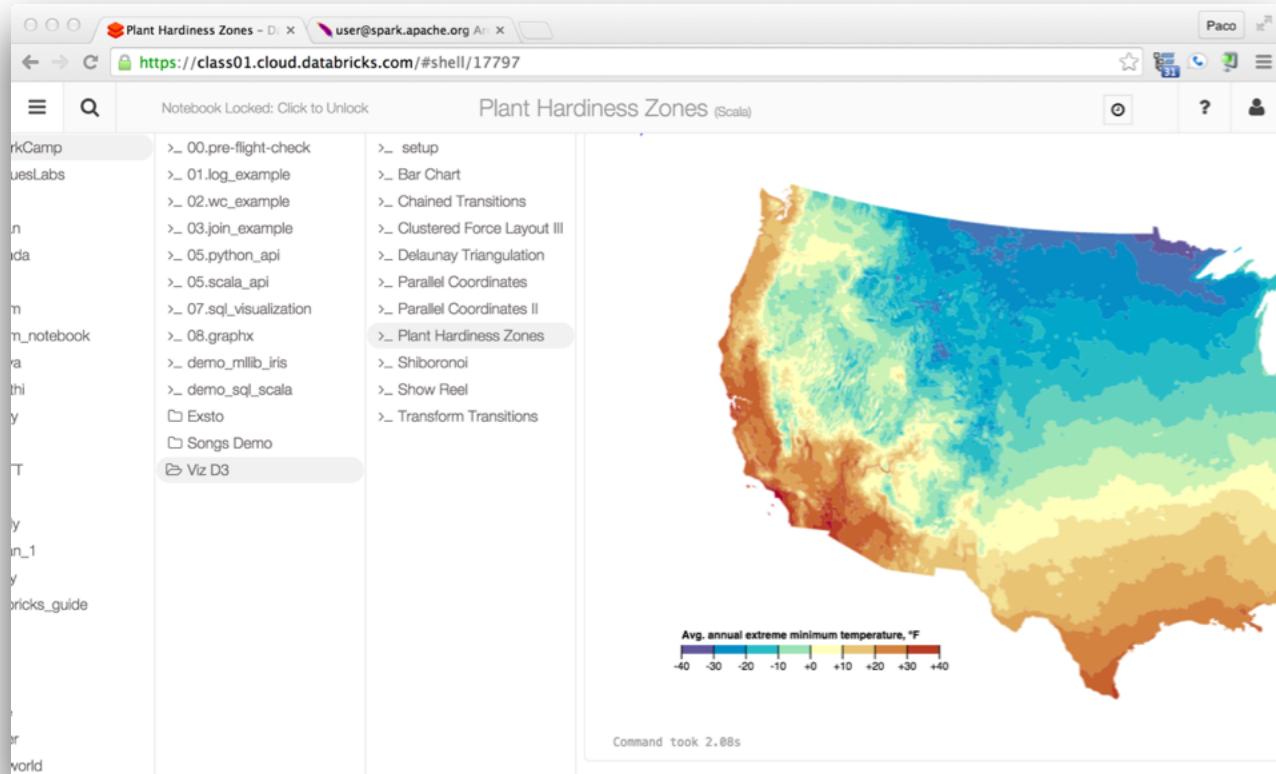
The `displayHTML()` command:

- 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:

The screenshot shows a Databricks notebook interface. The left sidebar contains navigation links for Workspace, Tables, Clusters, and Accounts. The main area displays a notebook titled "07.sql_visualization (Python)". The notebook content includes the following code:

```
> msg = sqlContext.jsonFile("/mnt/paco/exsto/original/").cache()
> msg.registerTempTable("msg")
Command took 2.52s

> msg.count()
Out[12]: 6205L
Command took 0.17s
```

Below the code, there are several explanatory text blocks:

- SQL and Visualization**
The data used in this notebook is based on the Apache Spark user@spark.apache.org dev
- Import a JSON data set (Spark email during Q4 2014) and register its schema for ad-hoc S
- Each message has the fields: date, sender, id, next_thread, prev_thread, next_url,
- Question: How many records are there?
- What does the data in an example record look like?
- Note where persistence gets used to cache the JSON message data.
- Re-run the following count() a few times to see how persistence changes the run-time co

Training Survey

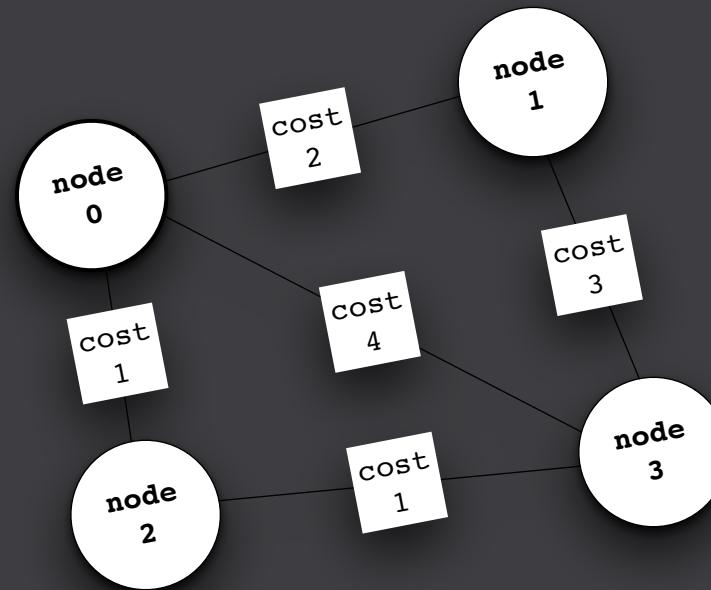
We appreciate your feedback about the DBC workshop. Please let us know how best to improve this material:

<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

```
// 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)
)

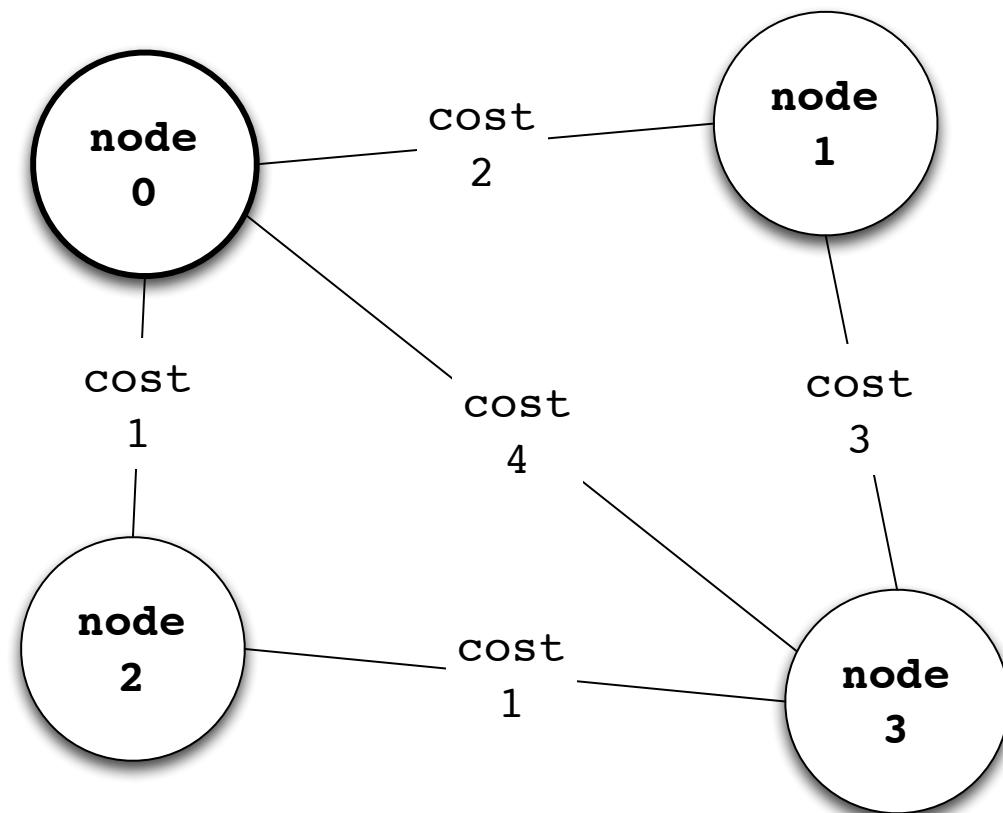
val nodeRDD: RDD[(Long, Peep)] = sc.parallelize(nodeArray)
val edgeRDD: RDD[Edge[Int]] = sc.parallelize(edgeArray)
val g: Graph[Peep, Int] = Graph(nodeRDD, edgeRDD)

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., **Dijkstra**



GraphX: Coding Exercise

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

The screenshot shows a Databricks notebook interface with the title "08.graphx (Scala)". The left sidebar displays a file tree with various notebooks and examples, including "08.graphx". The main workspace contains a section titled "Simple GraphX Example" with the following Scala code:

```
> 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)  
)
```

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](https://www.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](https://www.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/>

The screenshot shows the Spark Packages website interface. At the top, there is a dark blue header bar with the "Spark Packages" logo on the left and navigation links for "Feedback", "Register a package", "Login", "Find a package" (which is underlined), and a search icon on the right. Below the header, a main content area has a light gray background. It displays a message "A community index of packages for Apache Spark." followed by "28 packages". Two package entries are listed: "databricks/spark-avro" and "dibbhatt/kafka-spark-consumer". Each entry includes the package name, a brief description, the author's GitHub handle, the latest release information, a star rating, and the number of releases. Under each entry, there are small blue buttons labeled with the package's categories.

A community index of packages for **Apache Spark**.

28 packages

databricks/spark-avro

Integration utilities for using Spark with Apache Avro data

@pwendell / Latest release: 0.1 (11/27/14) / Apache-2.0 / ★★★★★ (14)

3 sql | 3 input | 2 library

dibbhatt/kafka-spark-consumer

Low Level Kafka-Spark Consumer

@dibbhatt / No release yet / ★★★★★ (3)

2 streaming | 1 kafka

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



The logo for Spark Summit 2015 features the word "Spark" in a bold, black, sans-serif font. A stylized orange star with five points is positioned above the letter "k". Below "Spark", the words "Summit 2015" are written in a smaller, blue, sans-serif font.

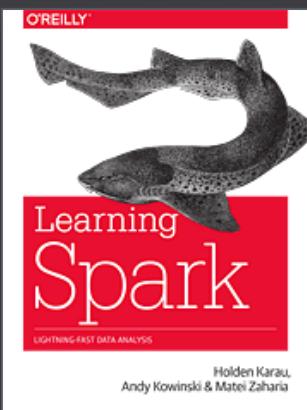


San Francisco June 15-17, 2015

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

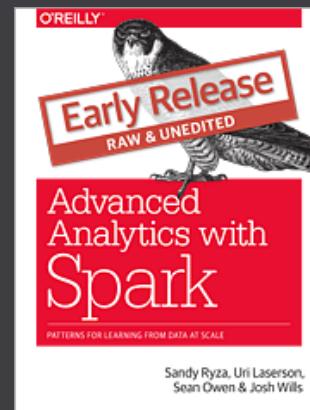
books+videos:

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



Intro to Apache Spark
Paco Nathan
O'Reilly (2015)
[shop.oreilly.com/
product/
0636920036807.do](http://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](http://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](http://shop.oreilly.com/product/9781782167068.do)

