Apache Spark Tutorial

Future Cloud Summer School

Paco Nathan @pacoid

2015-08-06

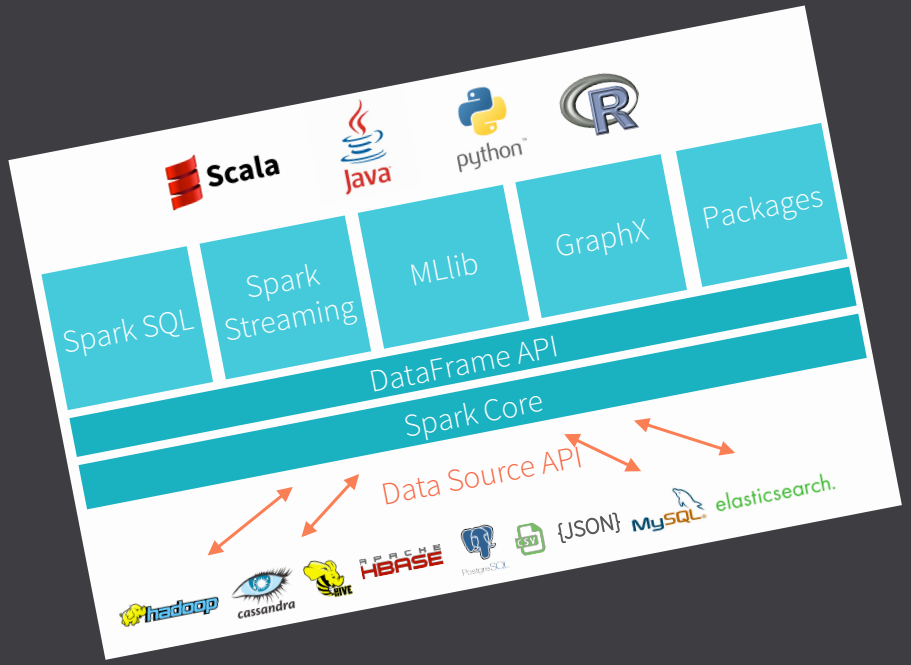
http://cdn.liber118.com/workshop/fcss_spark.pdf

EIT ICT Labs Summer School on Cloud and Big Data

In Conjunction with EU Marie Curie Initial Training Network "iSocial"

Stockholm, August 3-14, 2015

Getting Started



Getting Started: *Step 1*

Everyone will receive a username/password for one of the Databricks Cloud *shards*.

- required: laptop with wifi, browser
- everyone should have a password slip for the **Databricks** login – if not, please ask

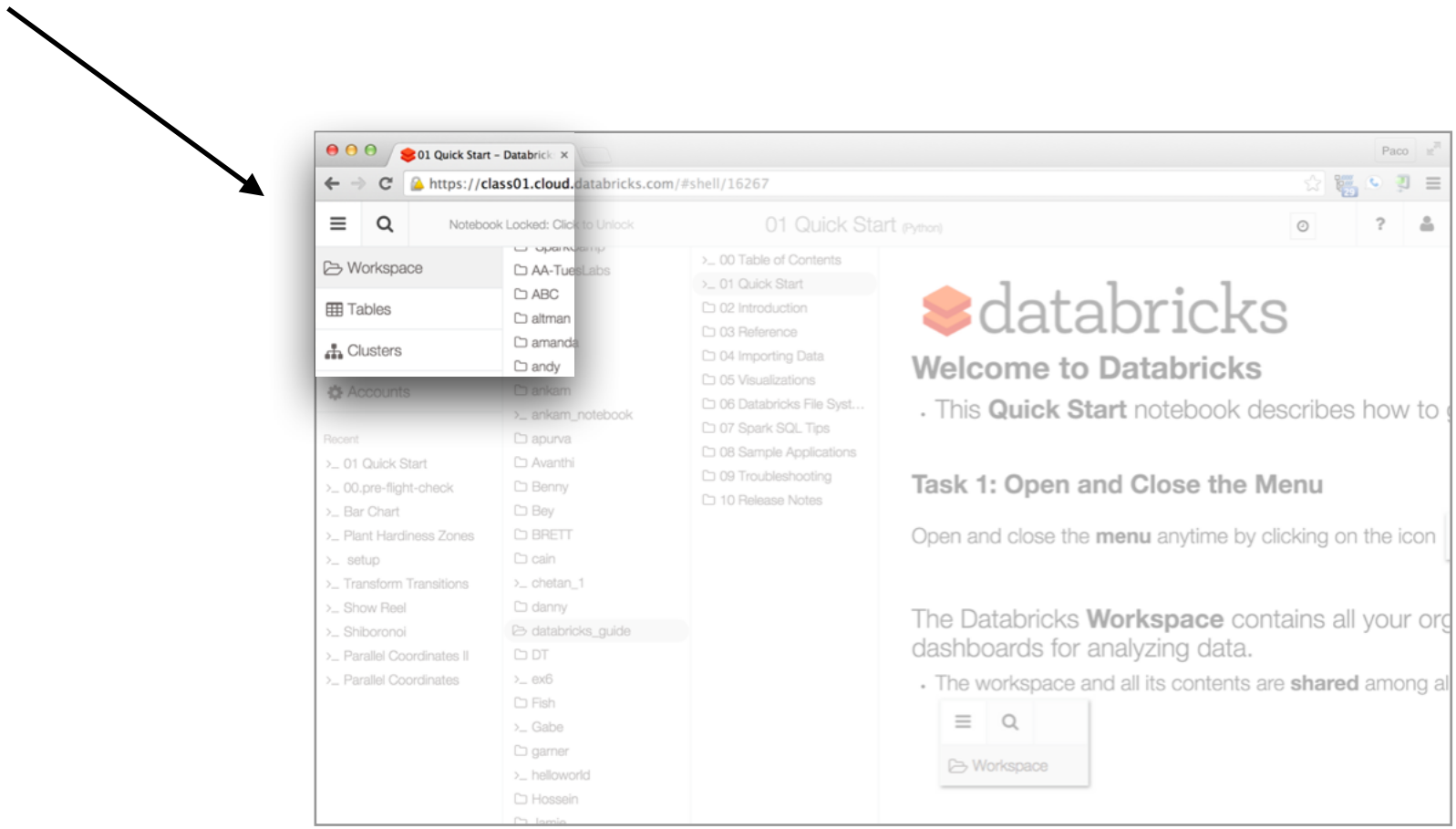
Please create and run a variety of notebooks on your account throughout the tutorial. You can export your work as source modules in Python, Scala, SQL, or R.

These accounts will remain open for at least one week.

For more details, see:
databricks.com/blog/2015/08/05/databricks-2-0-leading-the-charge-to-democratize-data.html

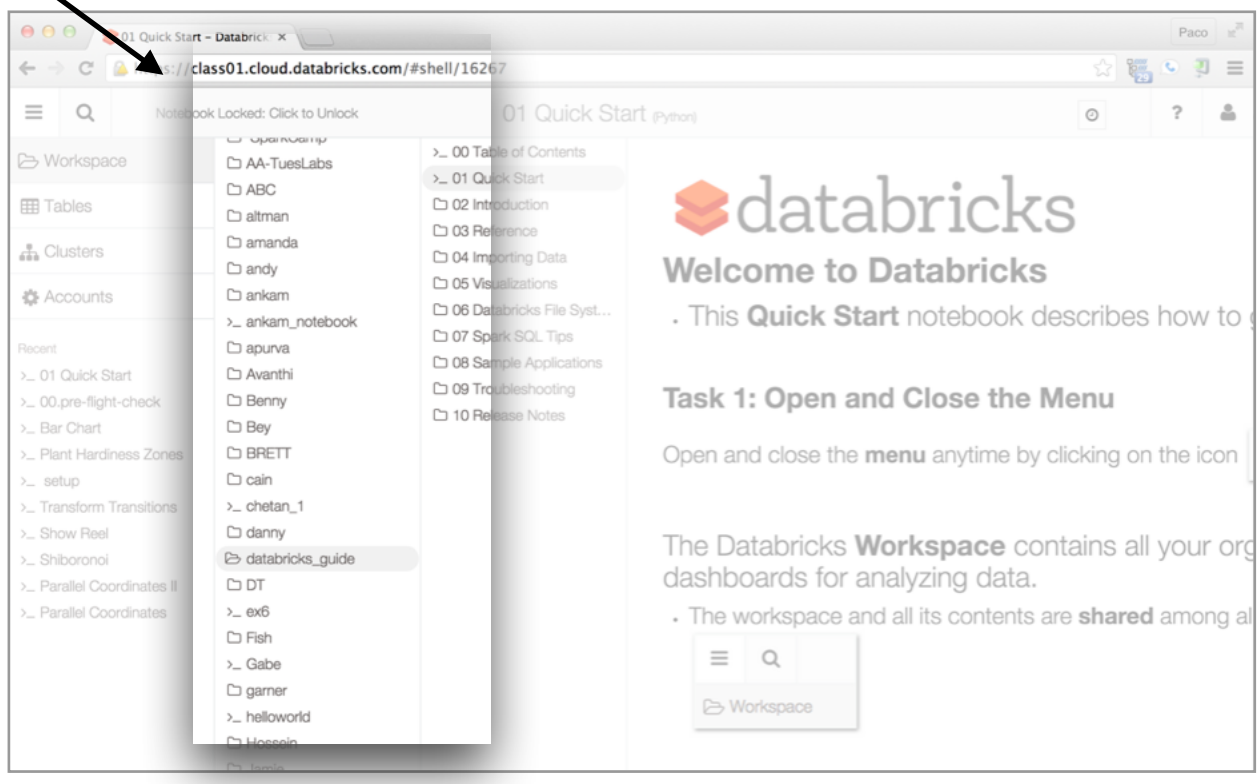
Getting Started: *Step 2*

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



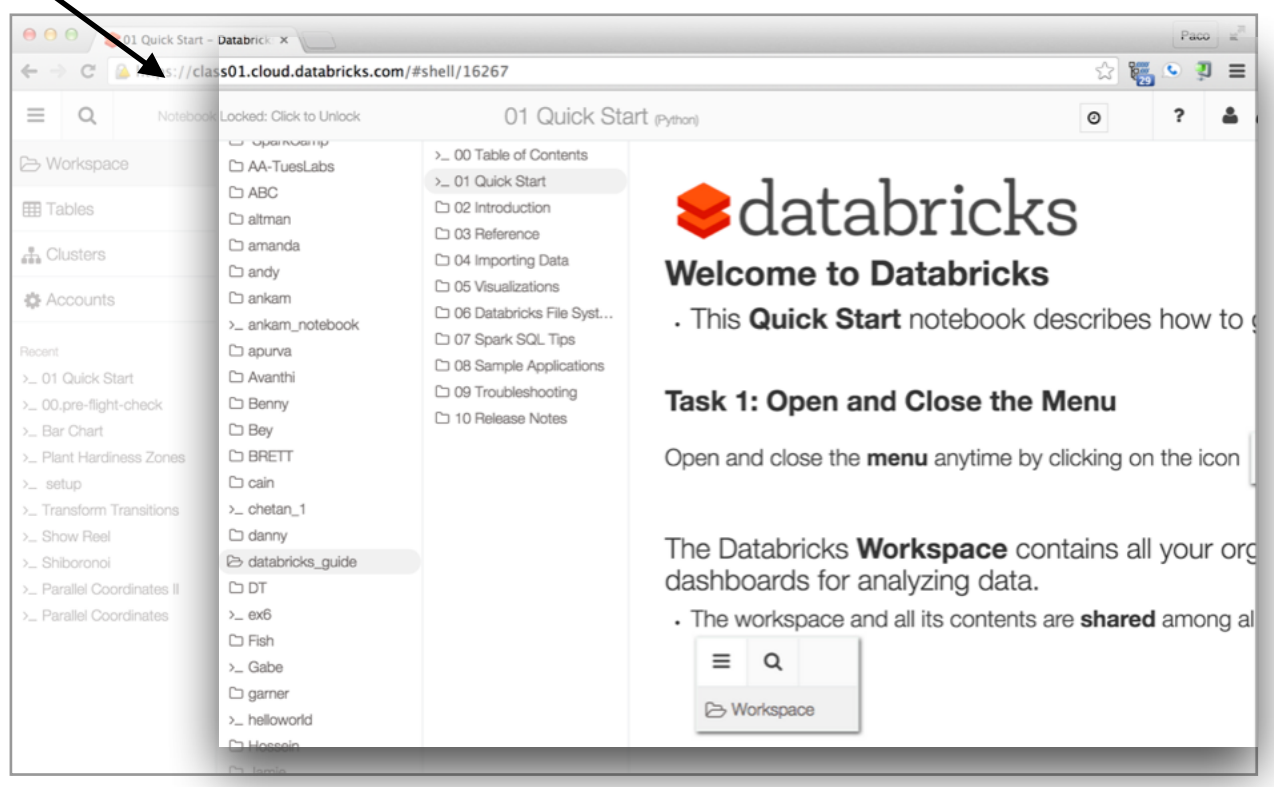
Getting Started: *Step 3*

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:

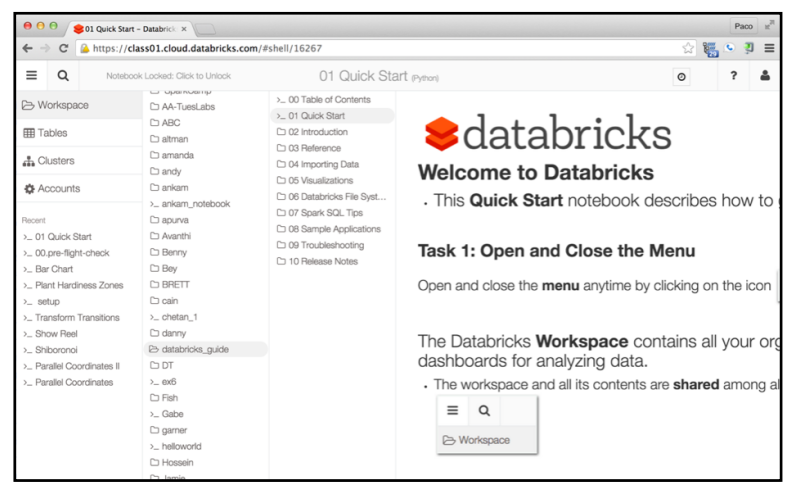


Getting Started: *Step 5*

See `/databricks-guide/01 Quick Start`

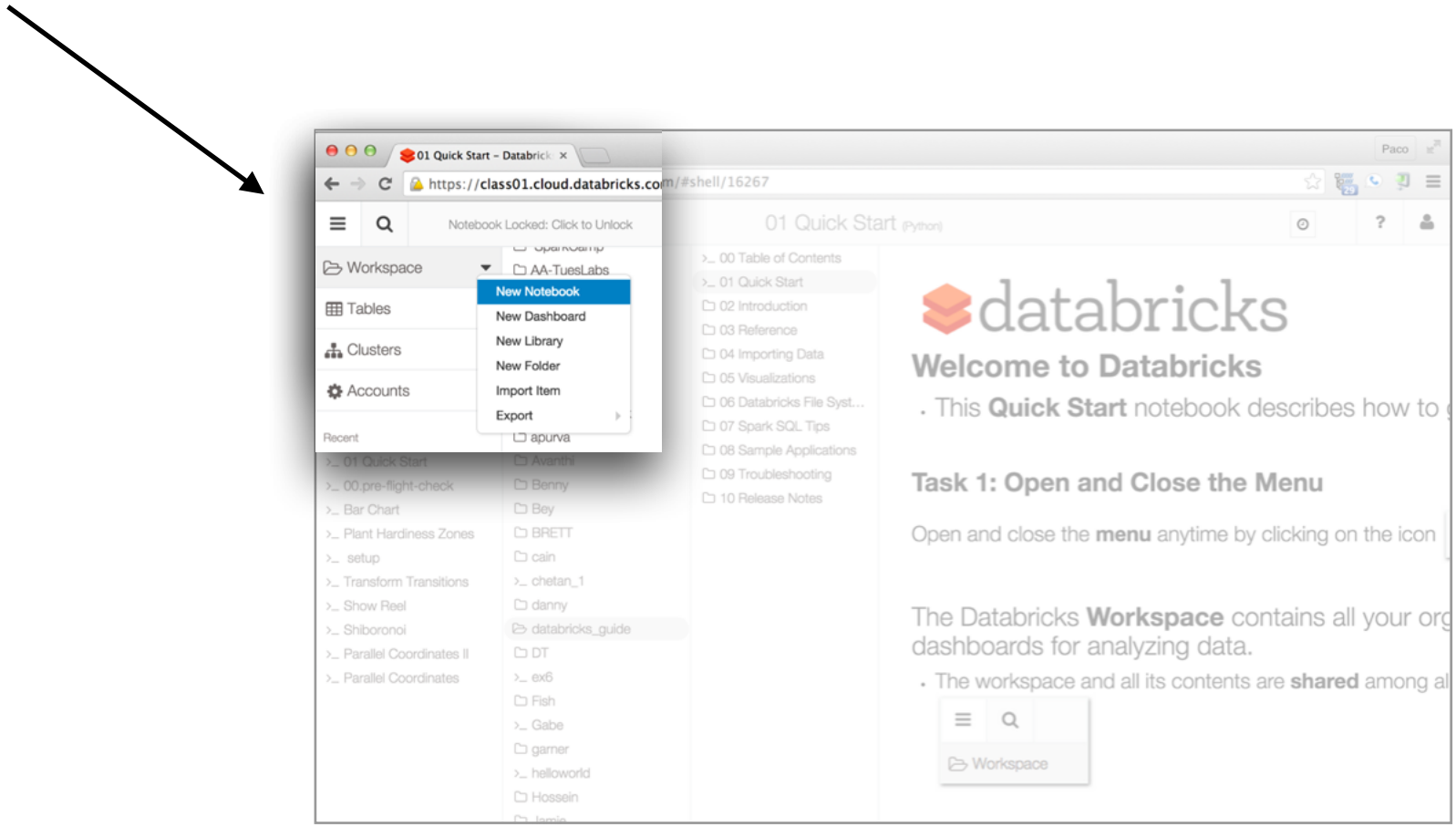
Key Features:

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



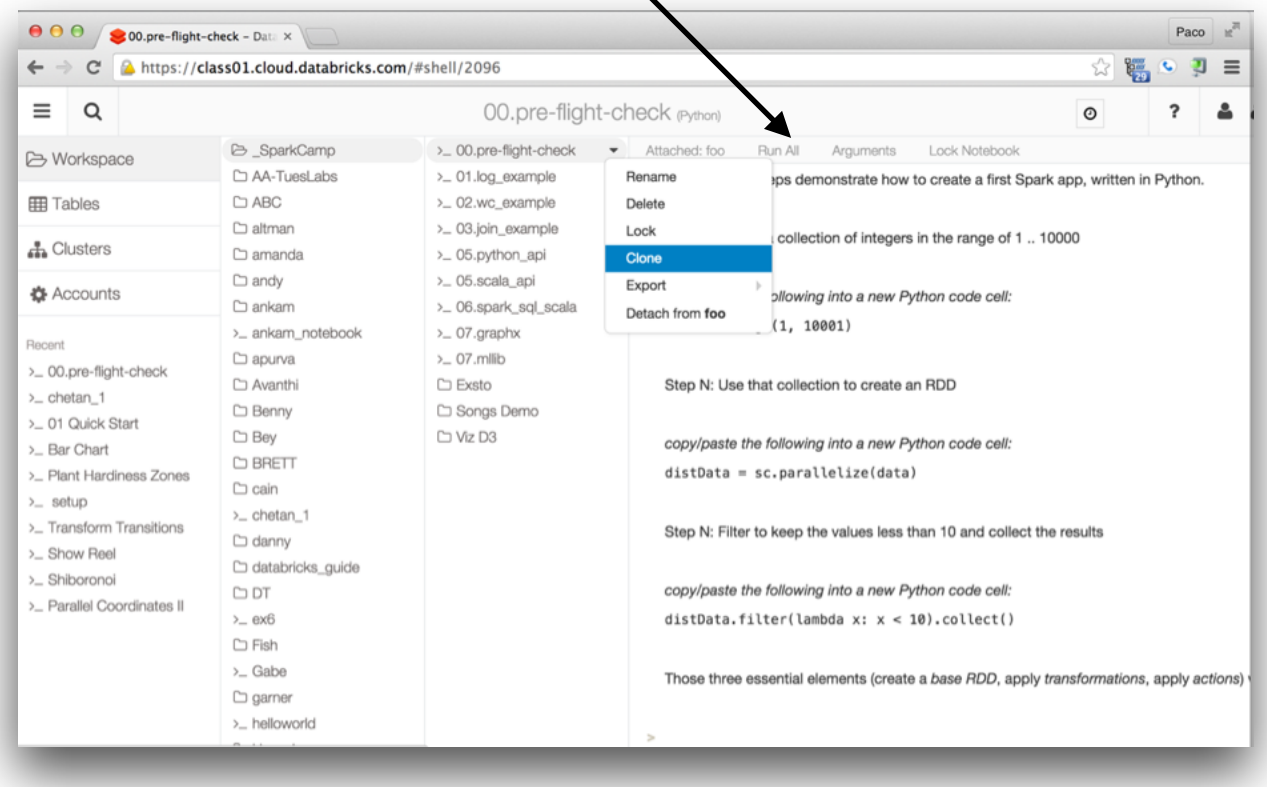
Getting Started: *Step 6*

Click on the *Workspace* menu and find your *folder* shown on the password slip:



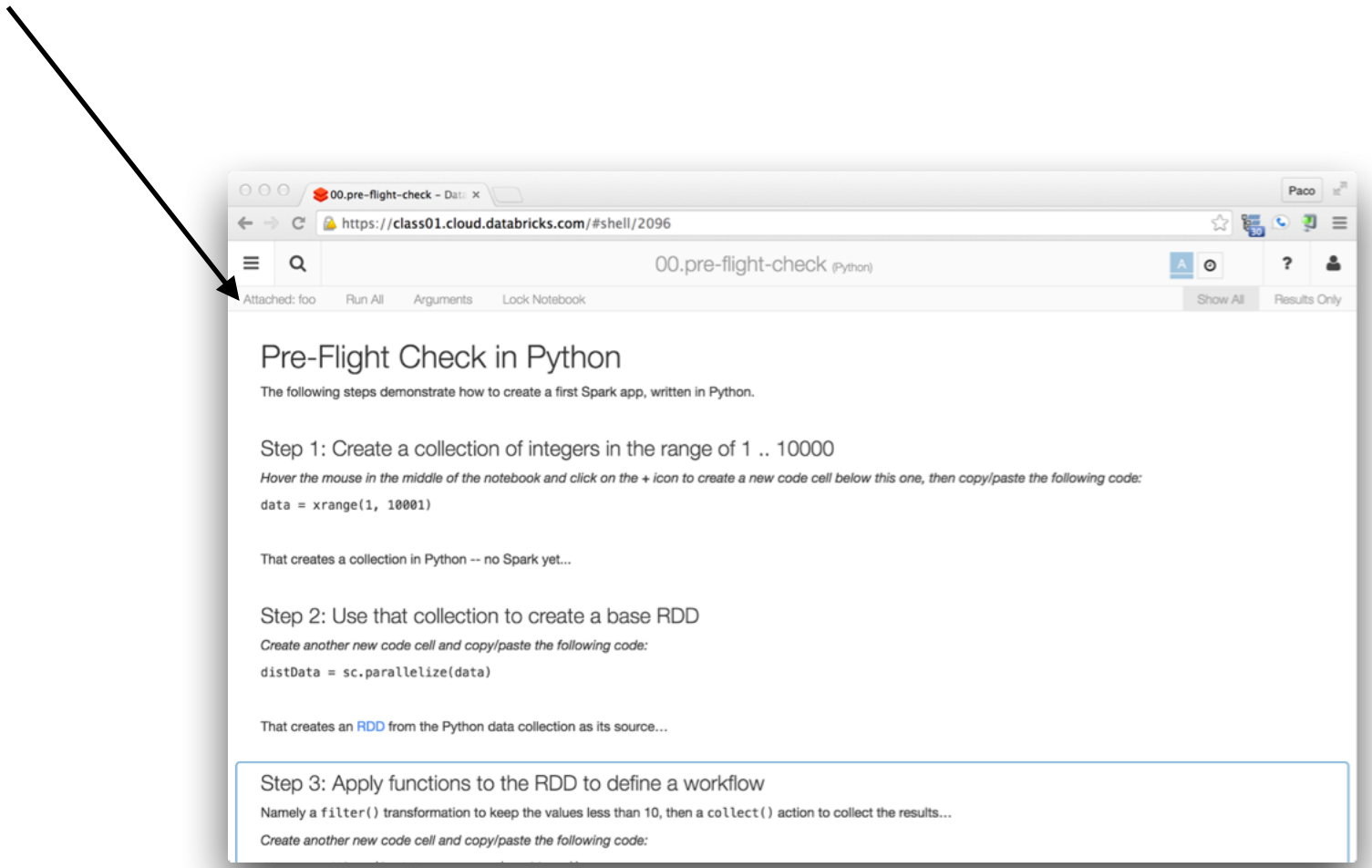
Getting Started: *Step 7*

Navigate to `/_SparkCamp/00.pre-flight-check` then click the *clone* button at the top/middle and copy to your home folder:



Getting Started: *Step 8*

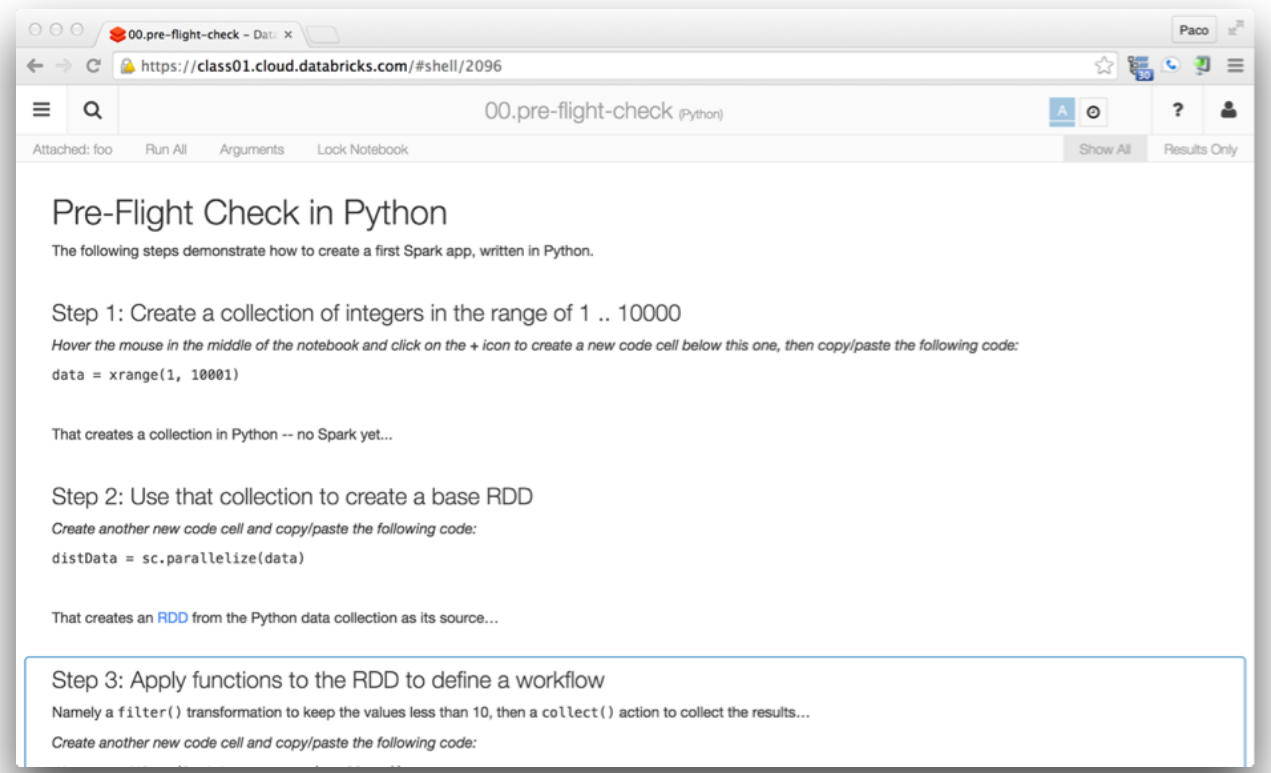
When you first open a cloned notebook, attach it to an available *cluster*:



Getting Started: *Coding Exercise*

Now let's get started with the coding exercise!

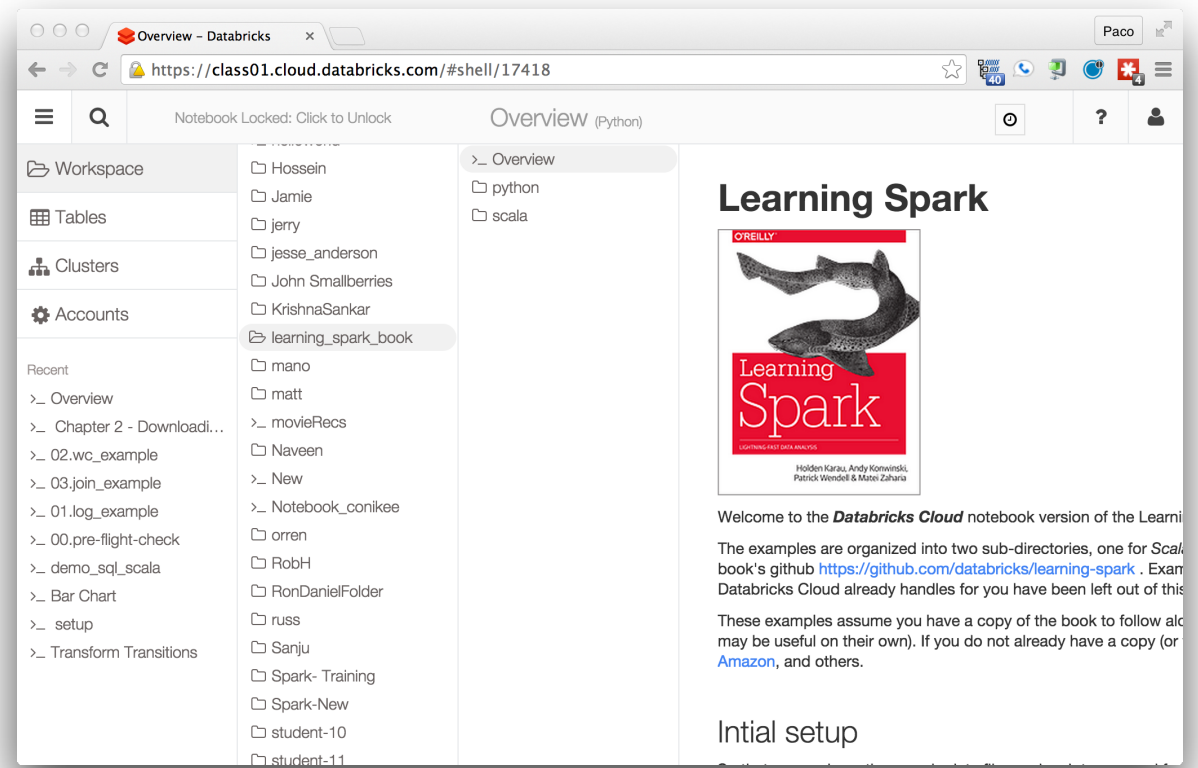
We'll define an initial Spark app in three lines of code:



Getting Started: *Extra Bonus!!*

See also the `/learning_spark_book`

for all of its code examples in notebooks:



Spark Overview

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

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

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

Spark Overview: *MapReduce*

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

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

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

Spark Overview: *MapReduce*

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

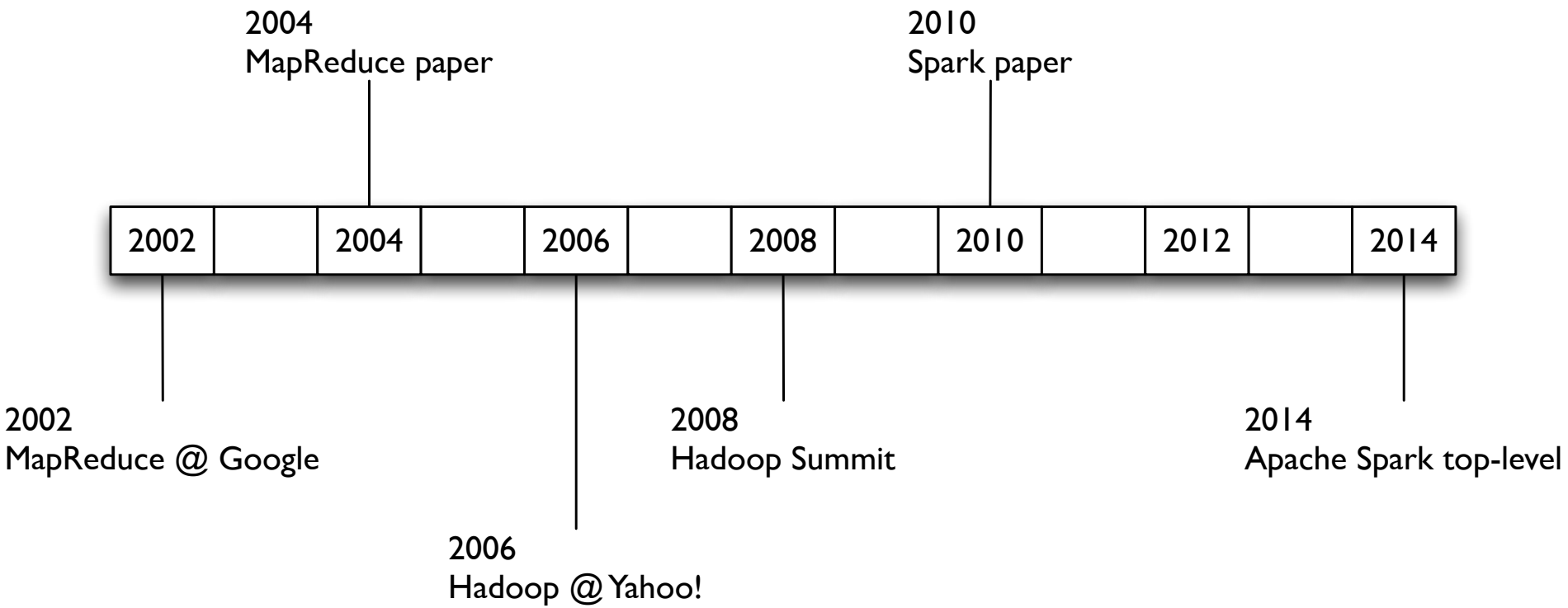
circa 2004 – **Google**
MapReduce: Simplified Data Processing on Large Clusters
Jeffrey Dean and Sanjay Ghemawat
research.google.com/archive/mapreduce.html

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

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

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

Spark Overview: *Functional Programming for Big Data*

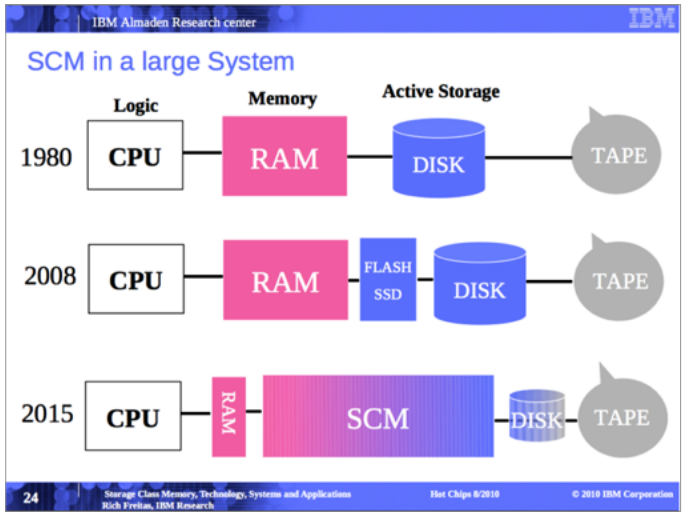


Spark Overview: *MapReduce*

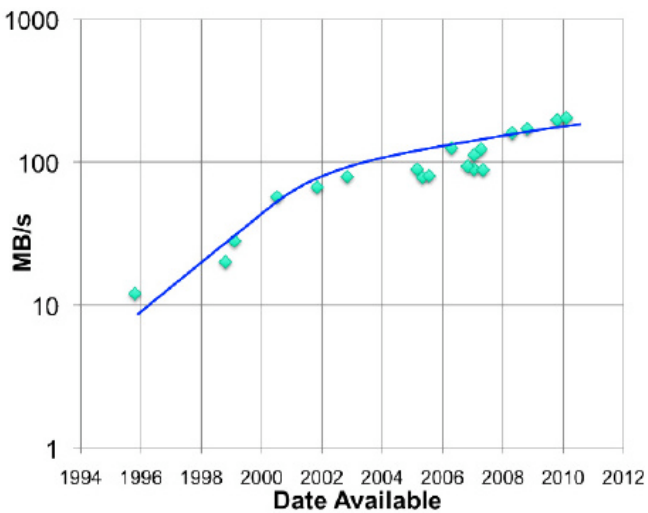
Open Discussion:

Enumerate several changes in data center technologies since 2002…

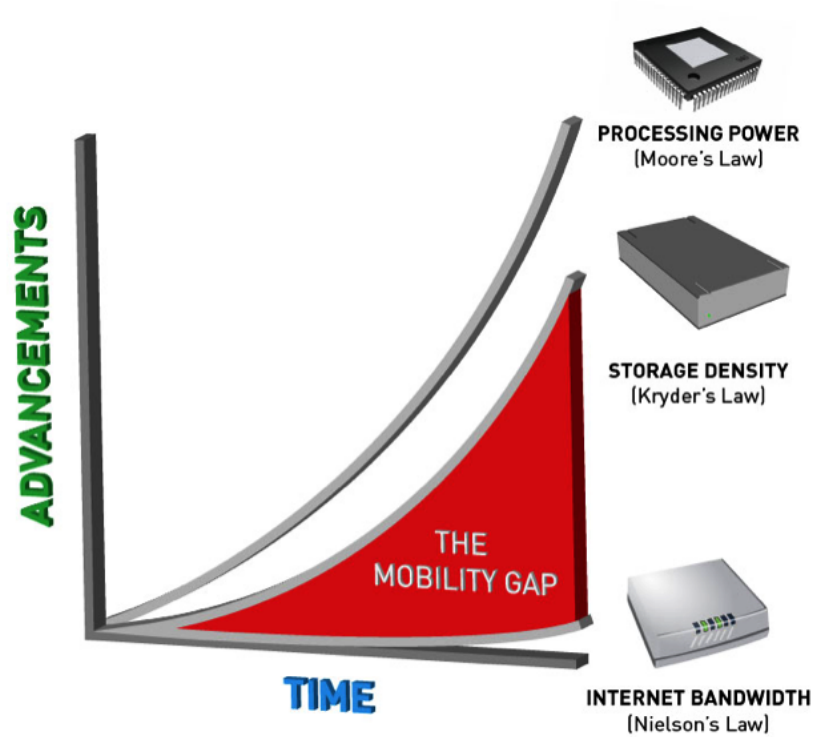
Spark Overview: *MapReduce*



Rich Freitas, **IBM Research**



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



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

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

Spark Overview: *MapReduce*

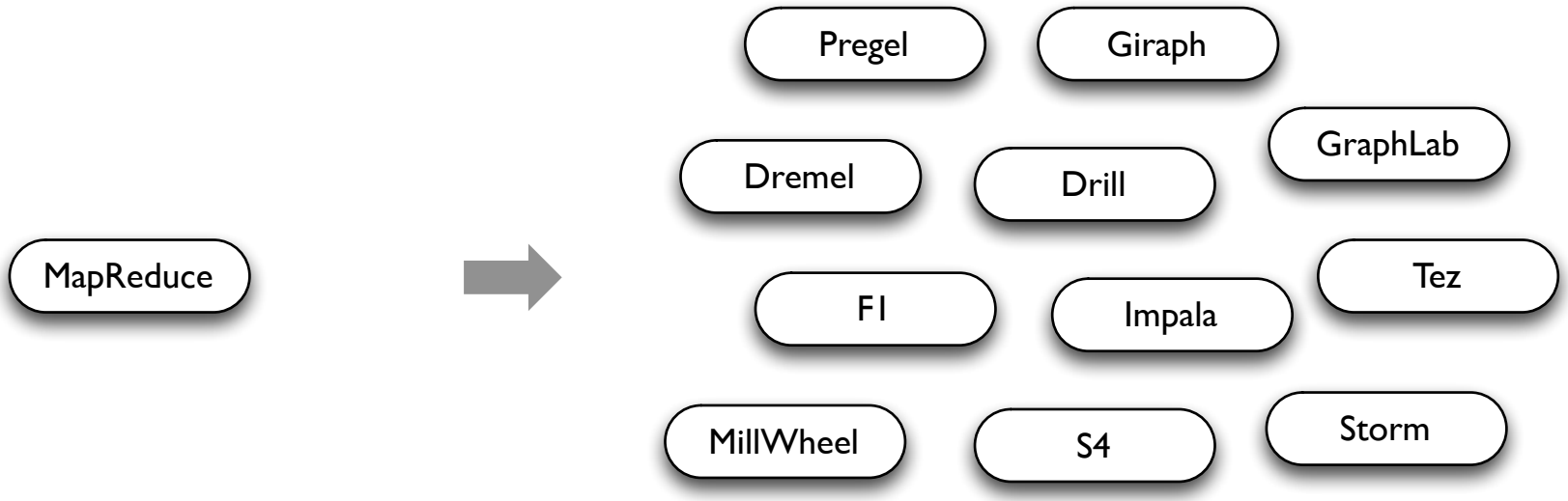
MapReduce use cases showed two major limitations:

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

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

Therefore, people built *specialized systems* as workarounds…

Spark Overview: *MapReduce and Motivations*



General Batch Processing

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

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

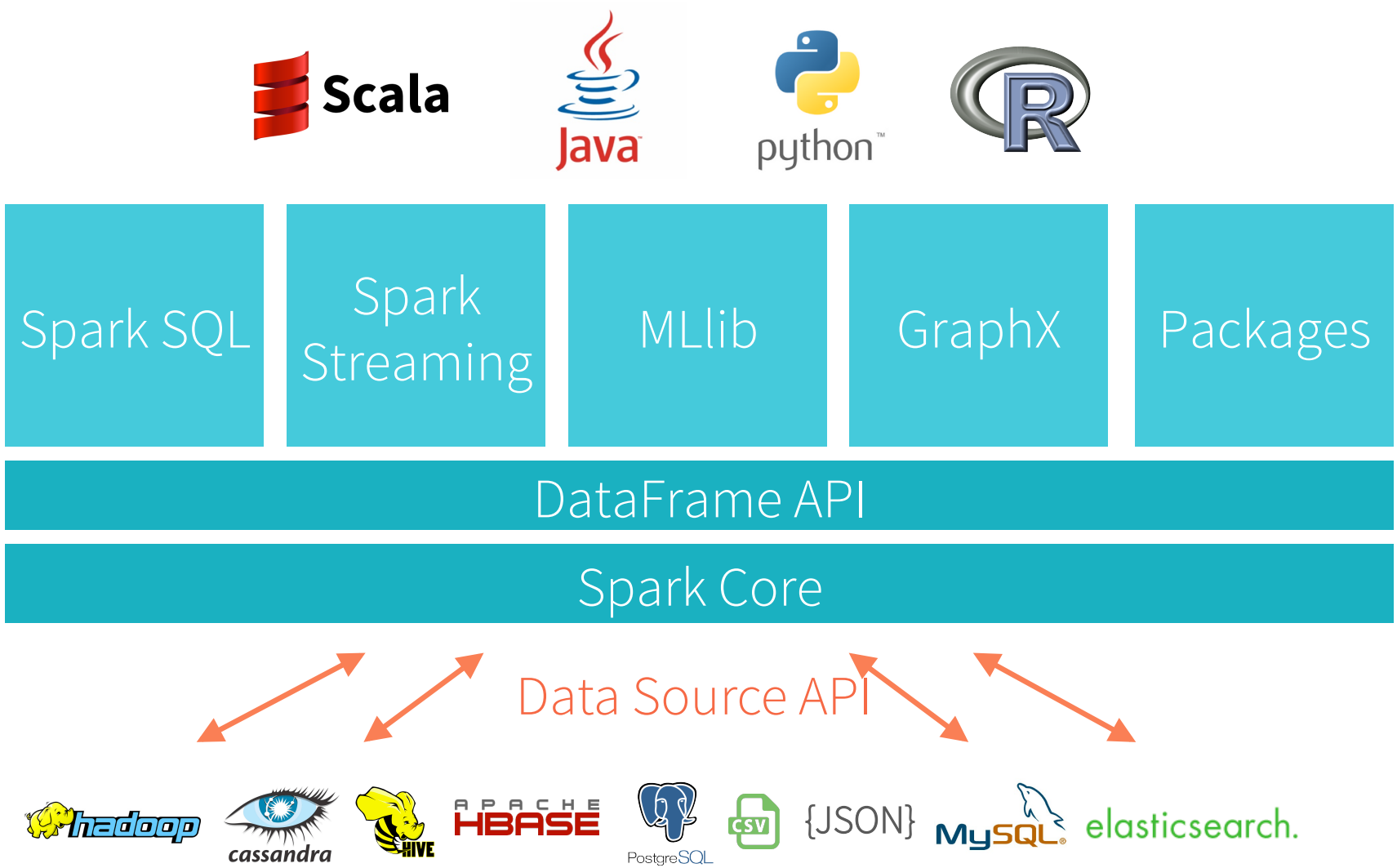
Spark Overview: *Origins at UC Berkeley*

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

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

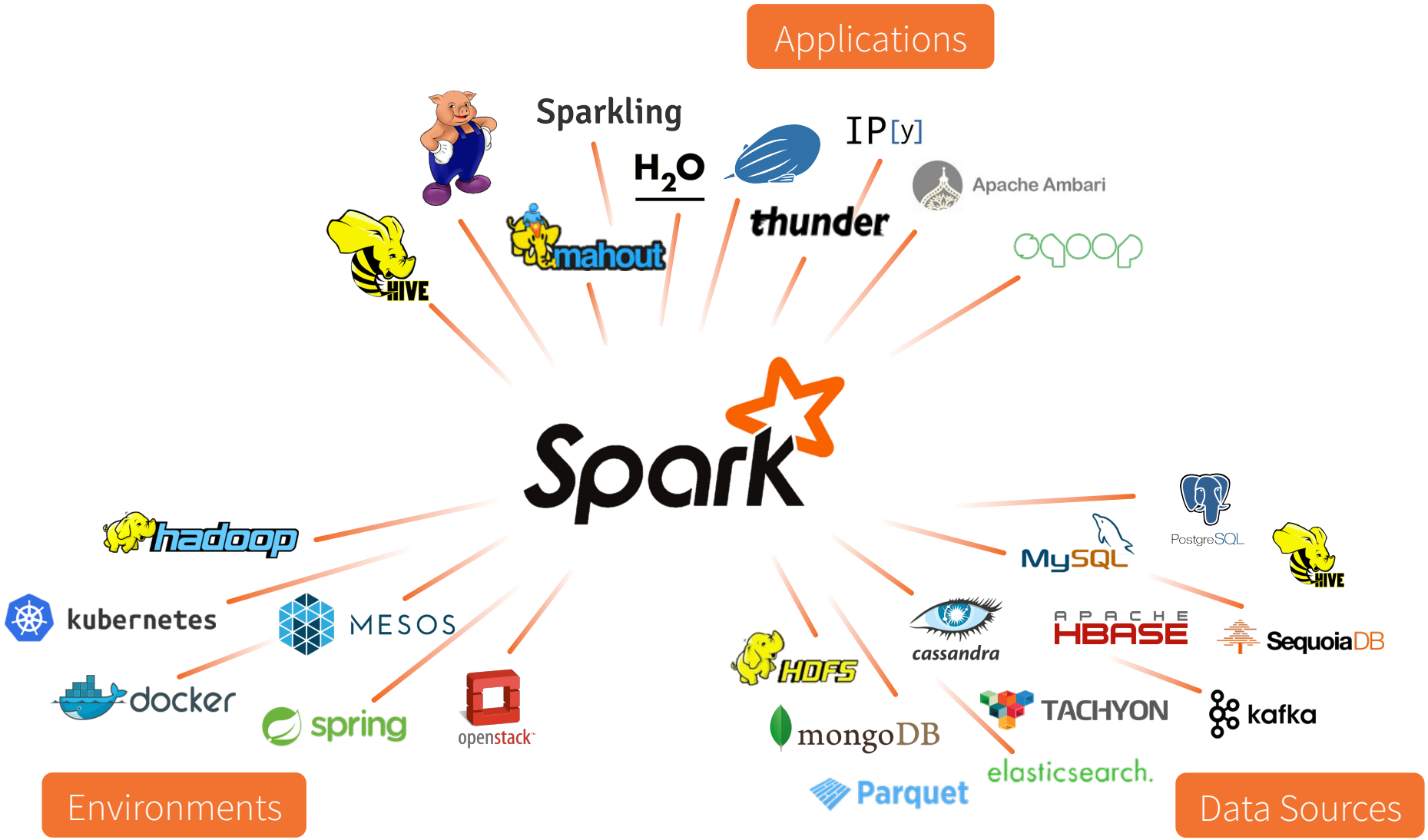
Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing
Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave,
Justin Ma, Murphy McCauley, Michael Franklin, Scott Shenker, Ion Stoica
usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf

Spark Overview: *Spark Components*



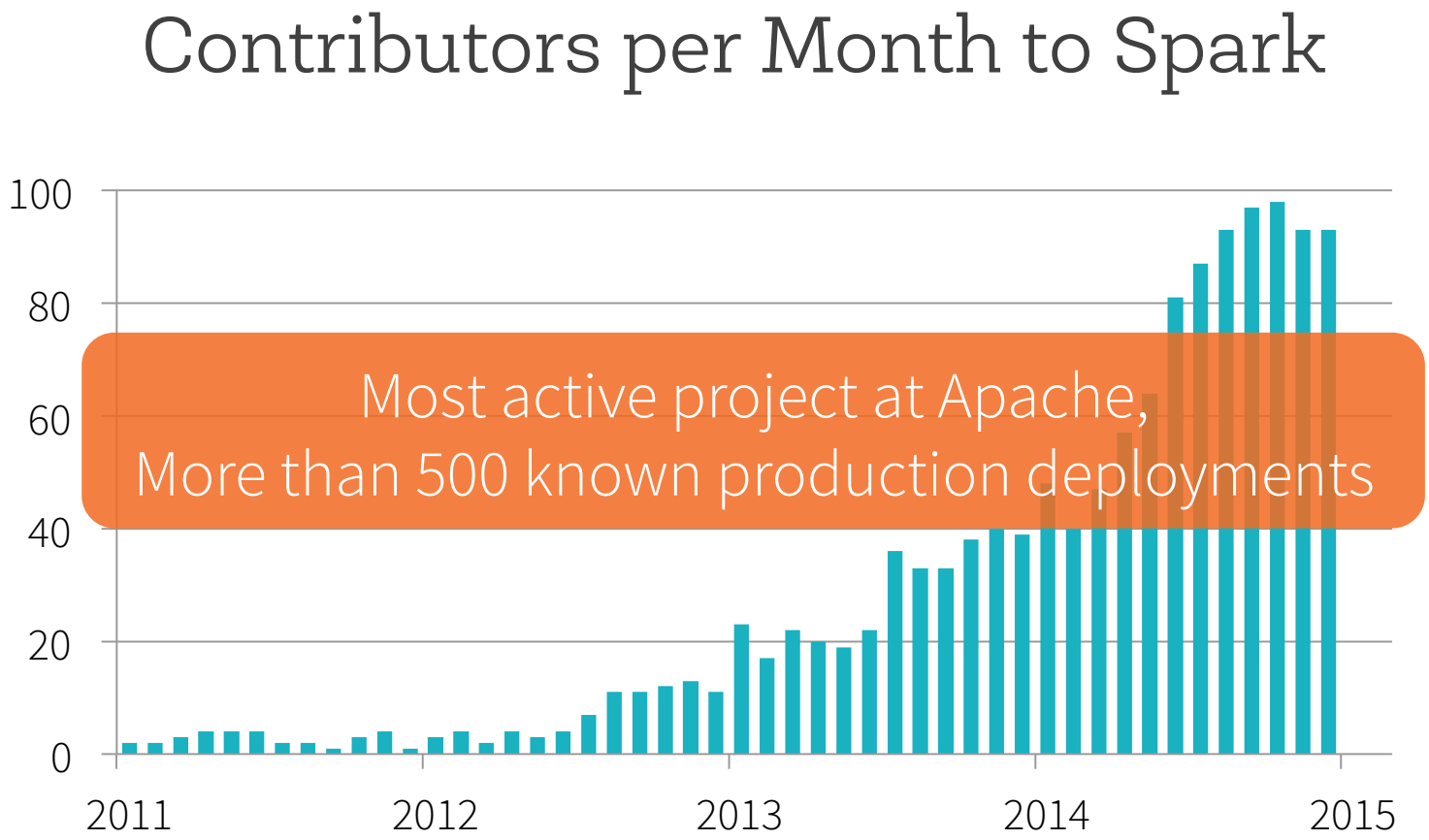
from Databricks

Spark Overview: *Open Source Ecosystem*



from Databricks

TL;DR: *Exponential Growth*



from Databricks

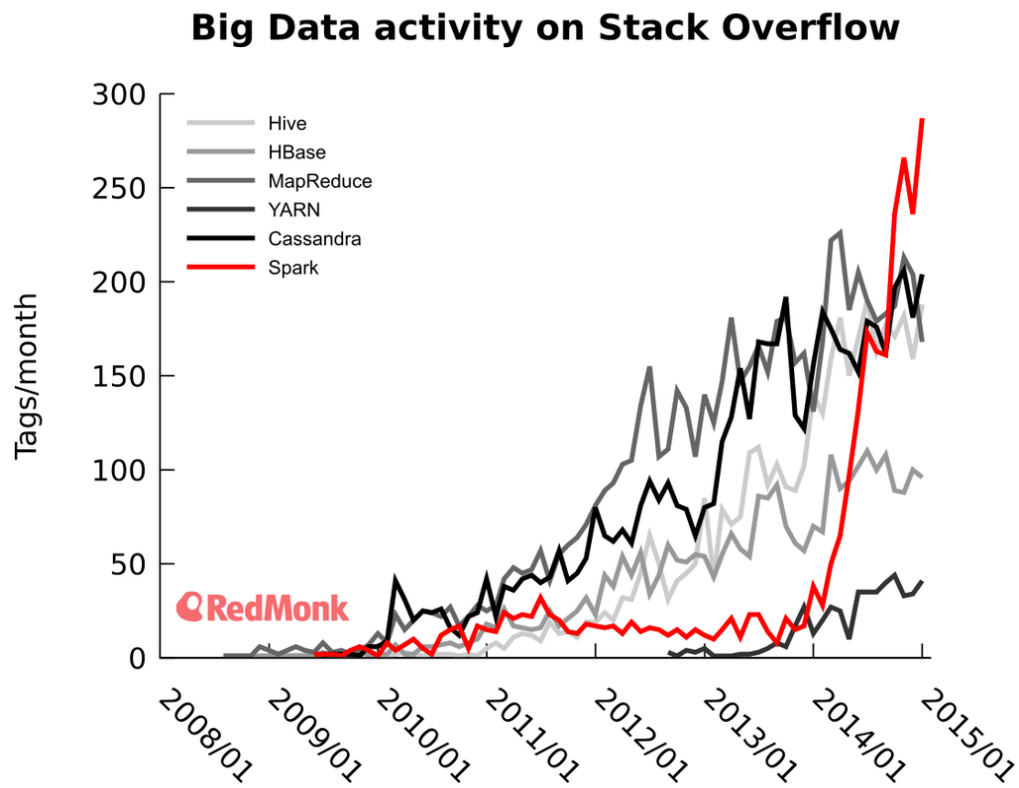
TL;DR: *Smashing Previous Sort Record*

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

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

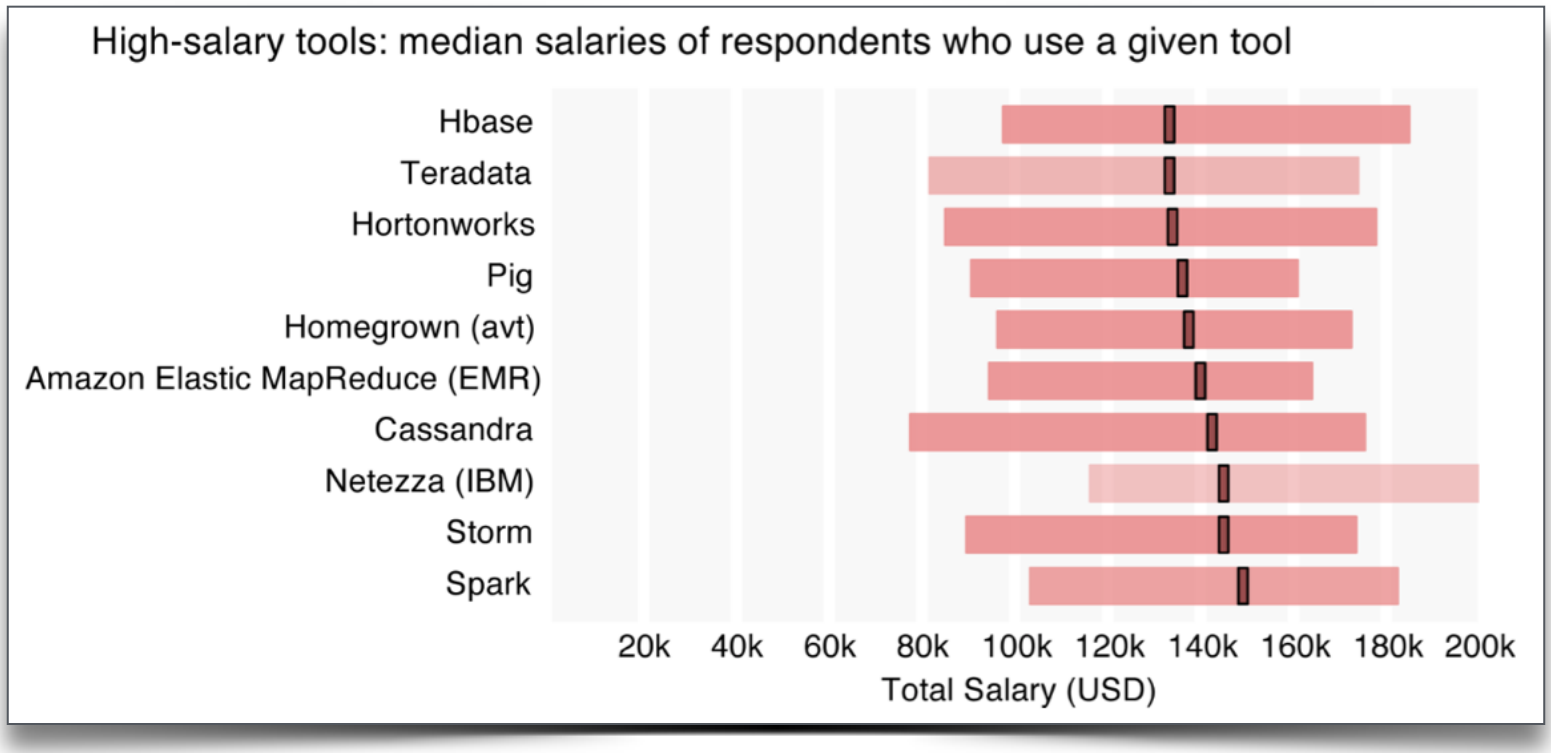
TL;DR: *Spark on StackOverflow*

twitter.com/dberkholz/status/568561792751771648



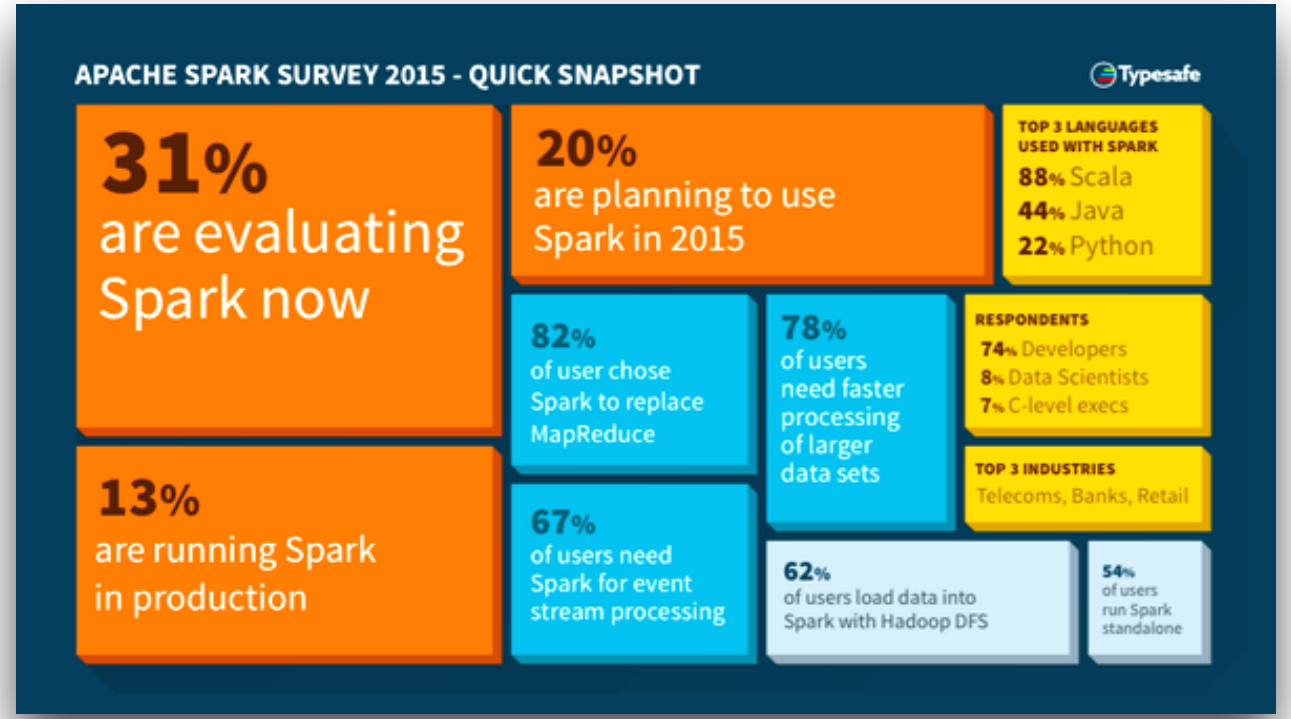
TL;DR: *Spark Expertise Tops Median Salaries for Big Data*

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

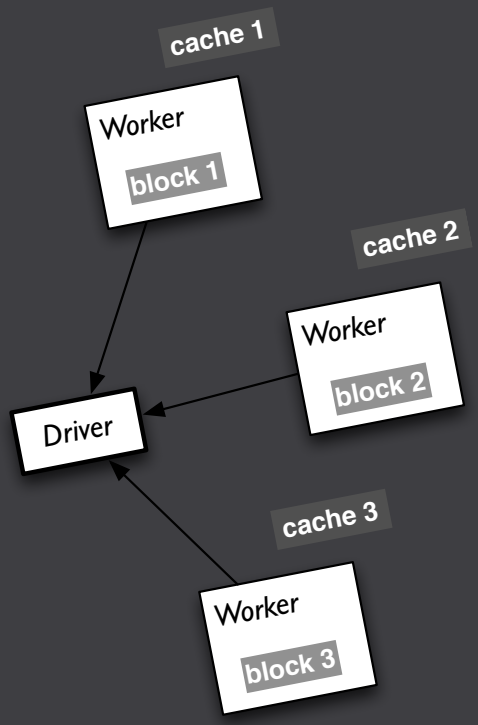


TL;DR: *Spark Survey 2015 by Databricks + Typesafe*

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

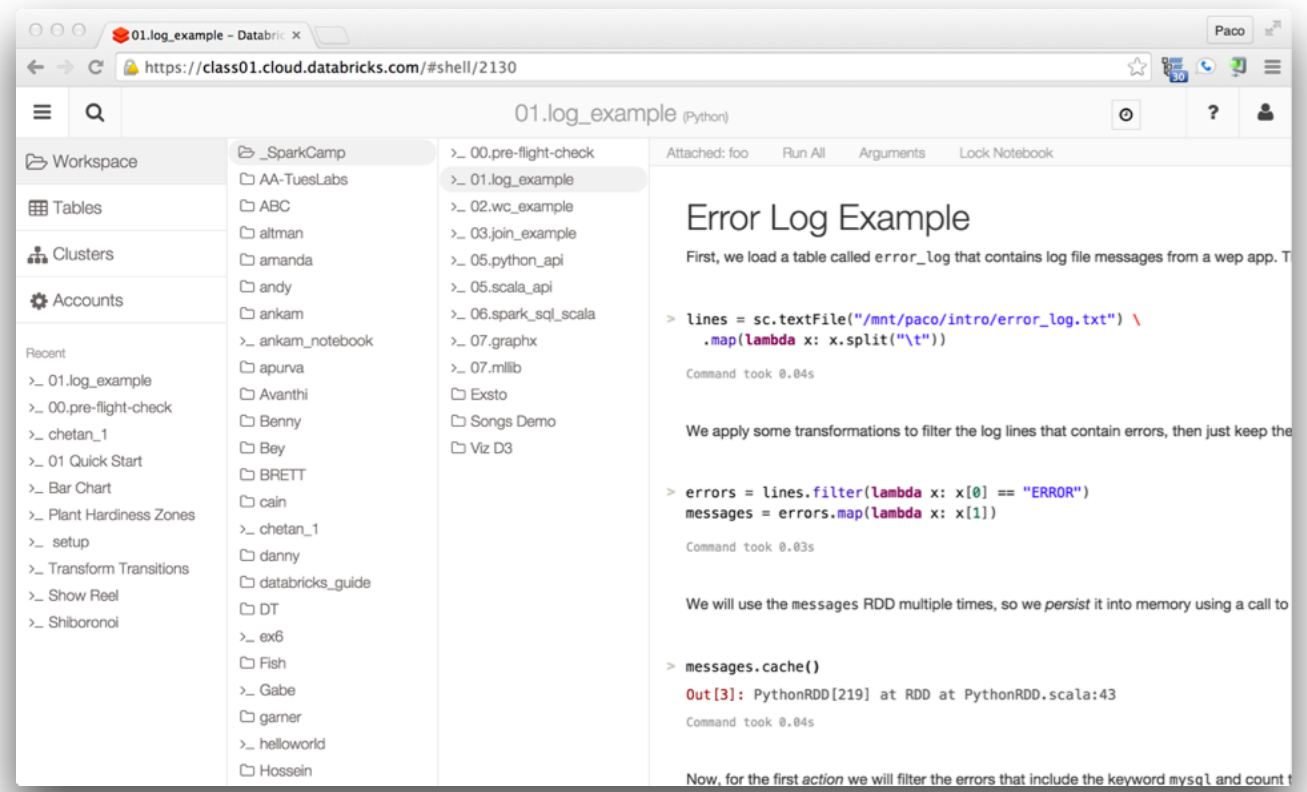


How Spark runs on a Cluster



Spark Deconstructed: *Log Mining Example*

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

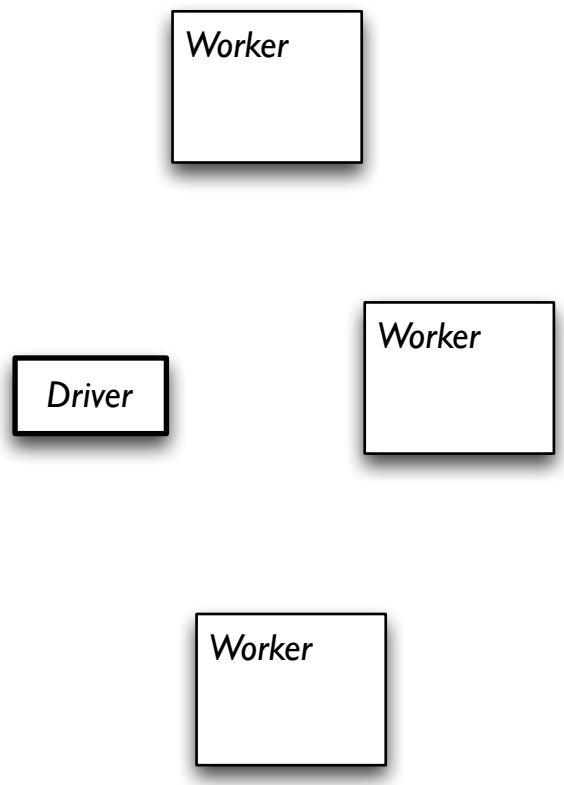


Spark Deconstructed: *Log Mining Example*

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

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

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

# persistence
messages.cache()

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

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

Spark Deconstructed: *Log Mining Example*

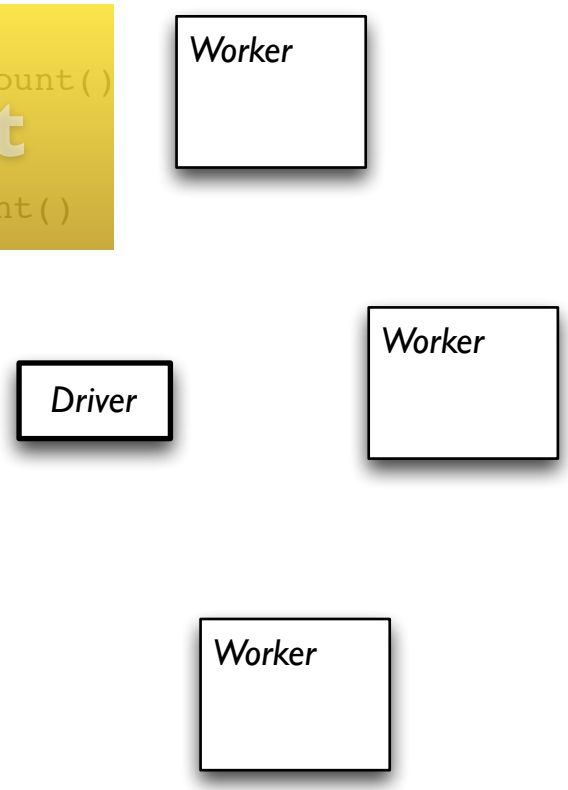
We start with Spark running on a cluster…
submitting code to be evaluated on it:



Spark Deconstructed: *Log Mining Example*

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

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

# persistence
messages.cache()

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

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

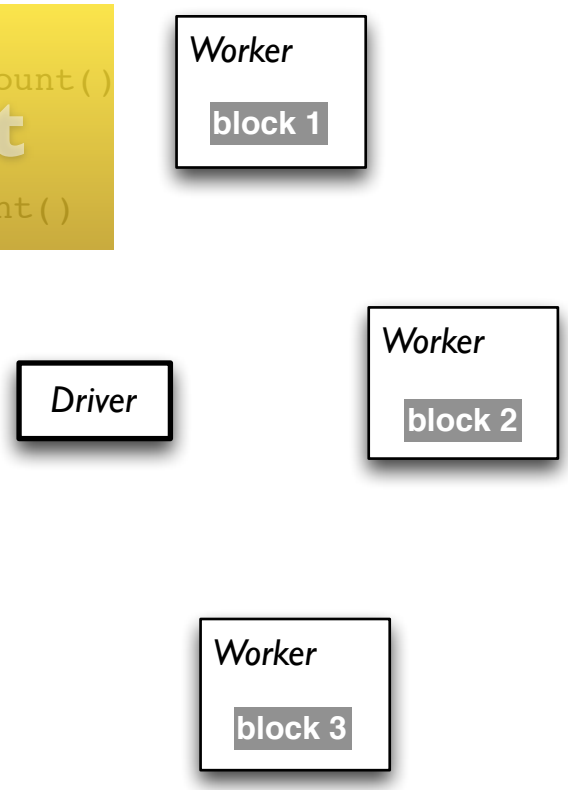
discussing the other part



Spark Deconstructed: *Log Mining Example*

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

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

# persistence
messages.cache()

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

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

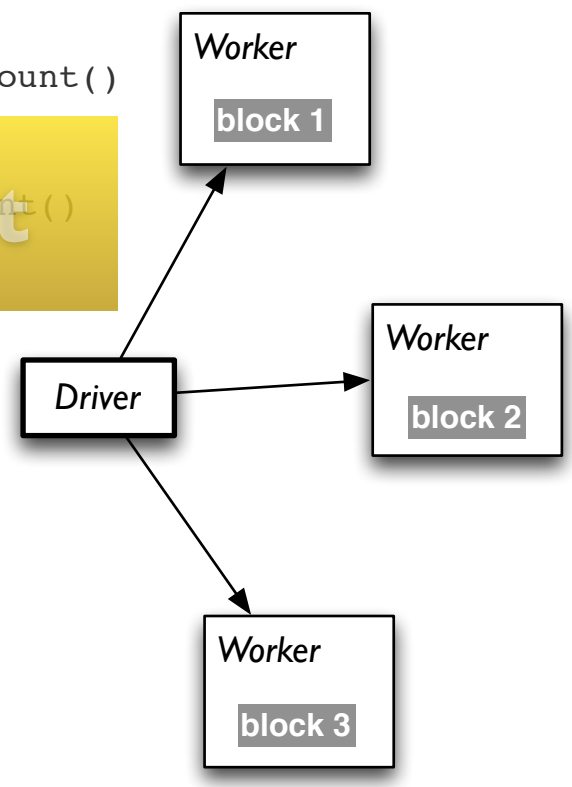
discussing the other part



Spark Deconstructed: *Log Mining Example*

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

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

# persistence
messages.cache()

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

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

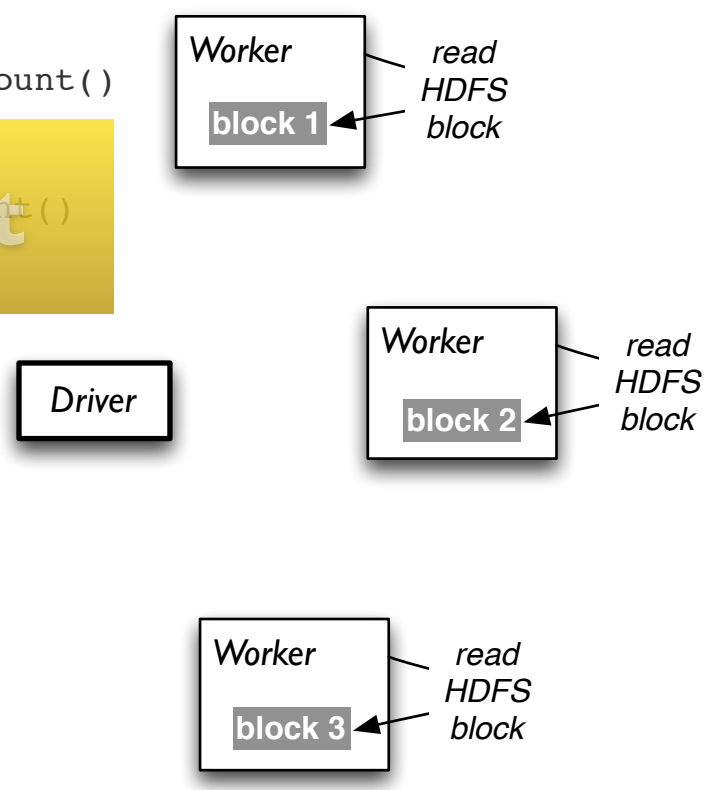
discussing the other part



Spark Deconstructed: *Log Mining Example*

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

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

# persistence
messages.cache()

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

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

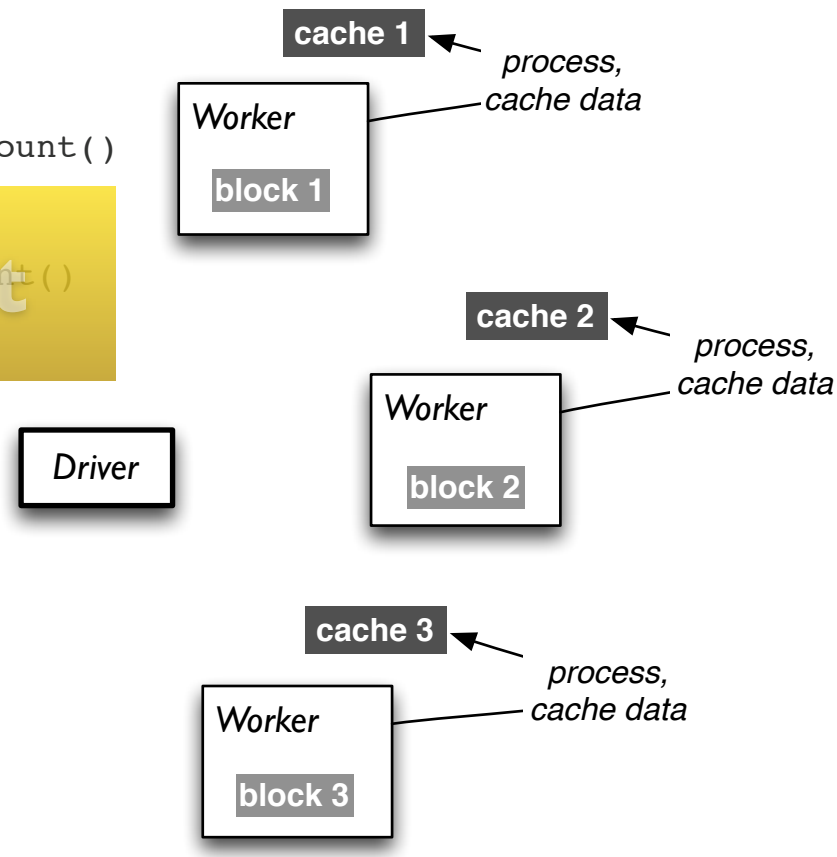
discussing the other part



Spark Deconstructed: *Log Mining Example*

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

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

# persistence
messages.cache()

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

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

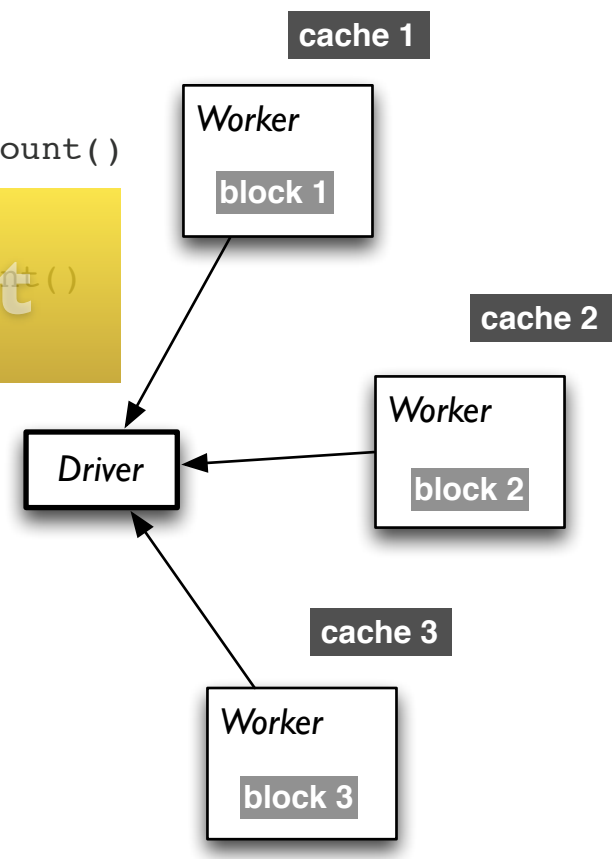
discussing the other part



Spark Deconstructed: *Log Mining Example*

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

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

# persistence
messages.cache()

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

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

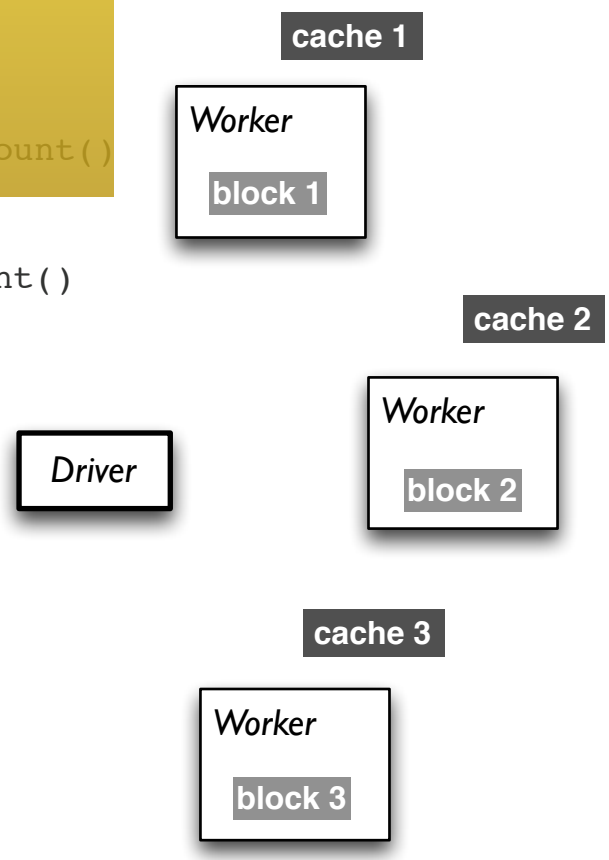
discussing the other part



Spark Deconstructed: *Log Mining Example*

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

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

# persistence
messages.cache()

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

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

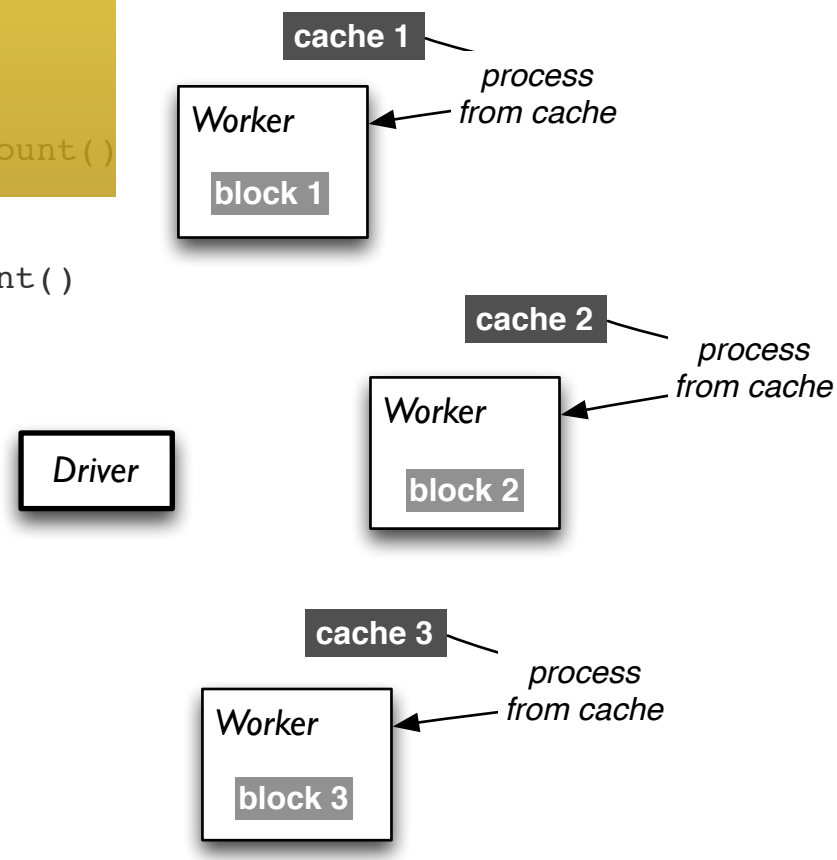
discussing the other part



Spark Deconstructed: *Log Mining Example*

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

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

# persistence
messages.cache()

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

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

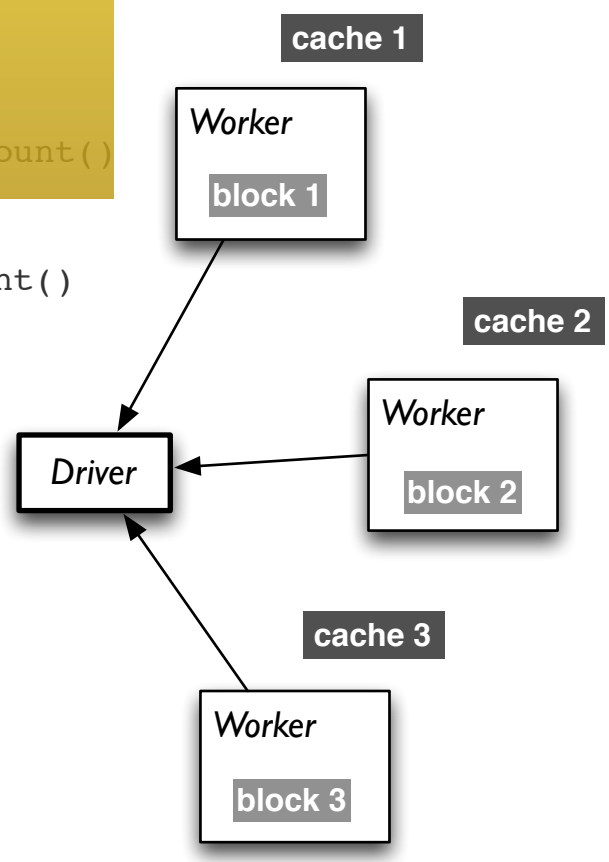
discussing the other part



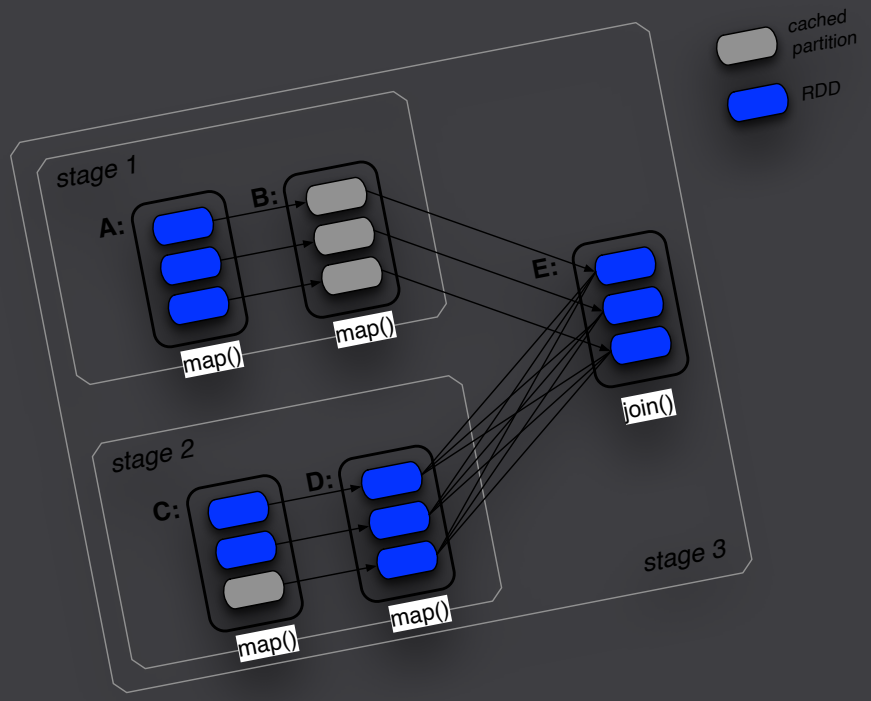
Spark Deconstructed: *Log Mining Example*

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

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

# persistence
messages.cache()

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

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

discussing the other part



WC, Joins, Shuffles



Coding Exercise: *WordCount*

Definition:

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

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

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

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


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

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

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

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

Coding Exercise: *WordCount*

```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);
  }
}
```

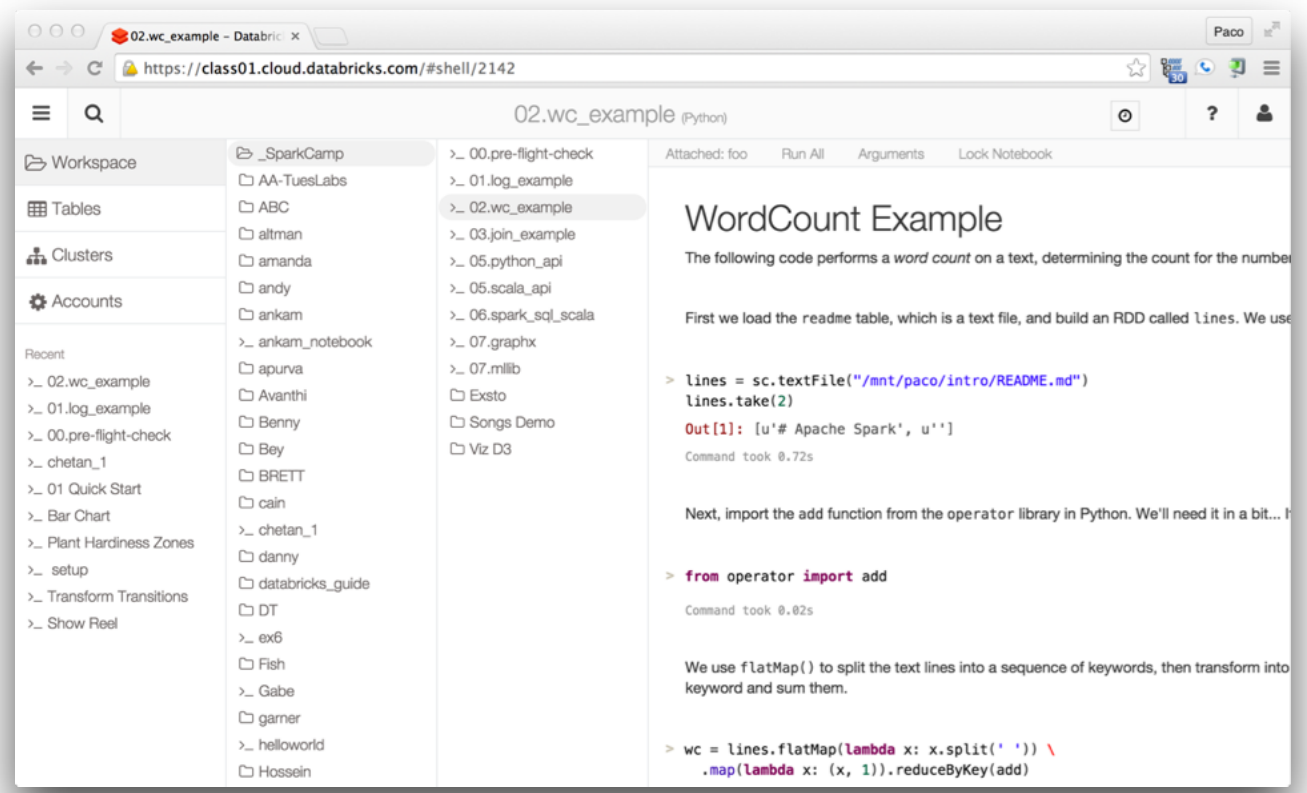
WordCount in 50+ lines of Java MR

```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

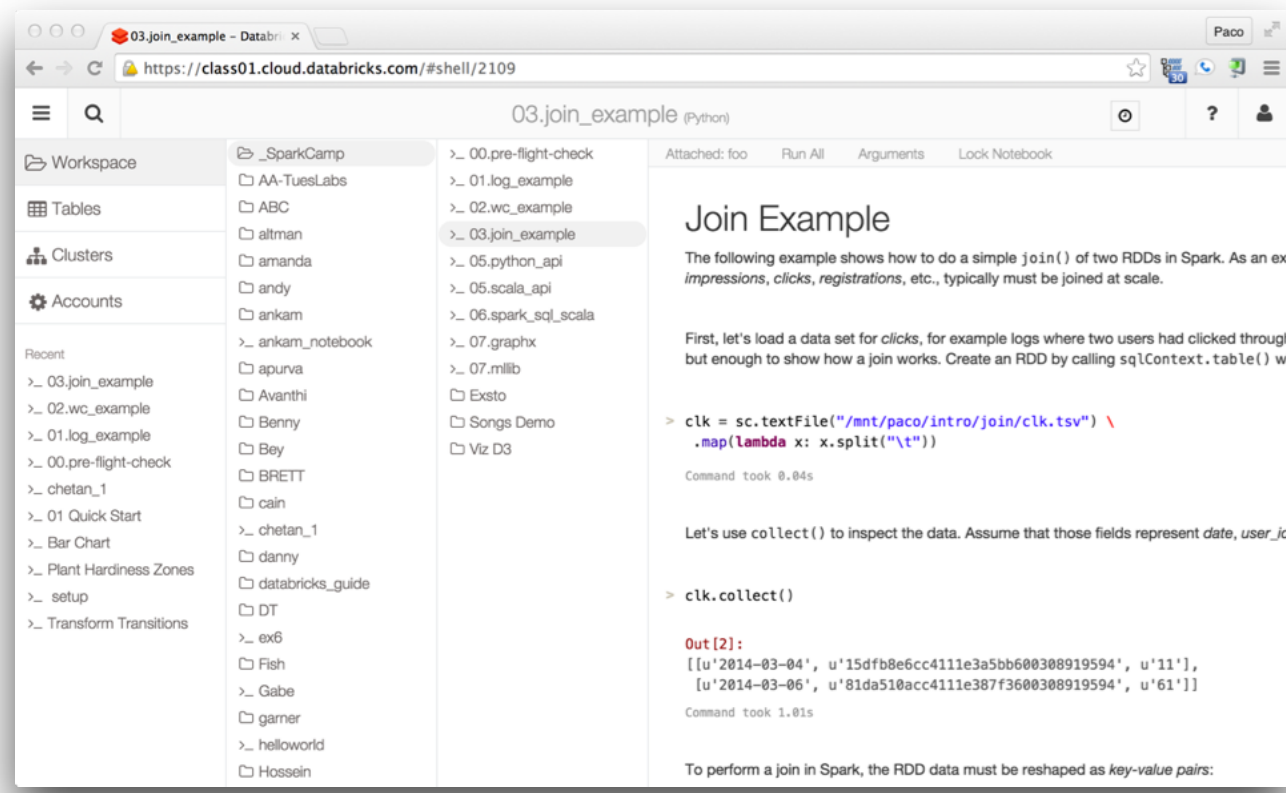
Coding Exercise: *WordCount*

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

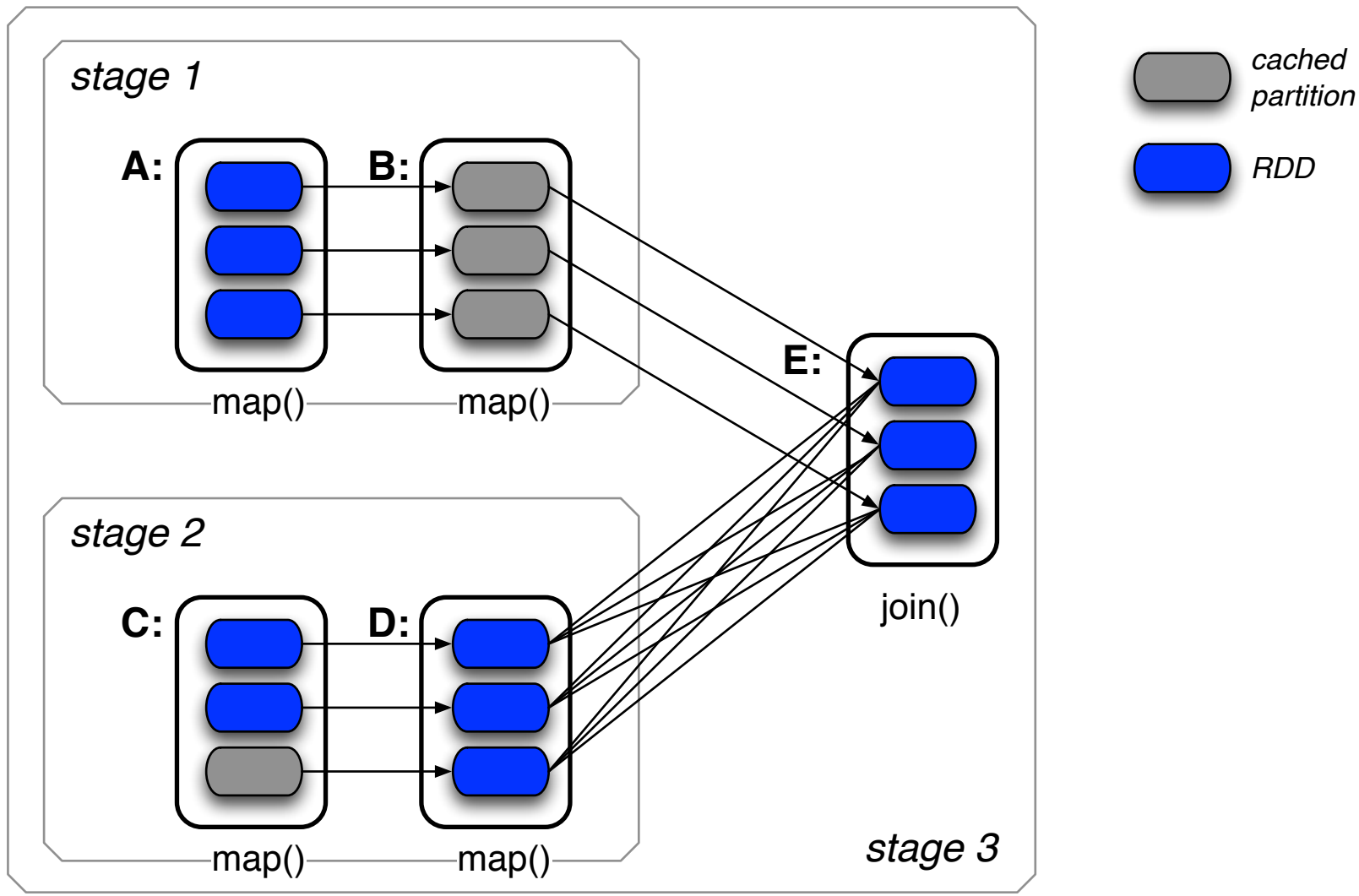


Coding Exercise: *Join*

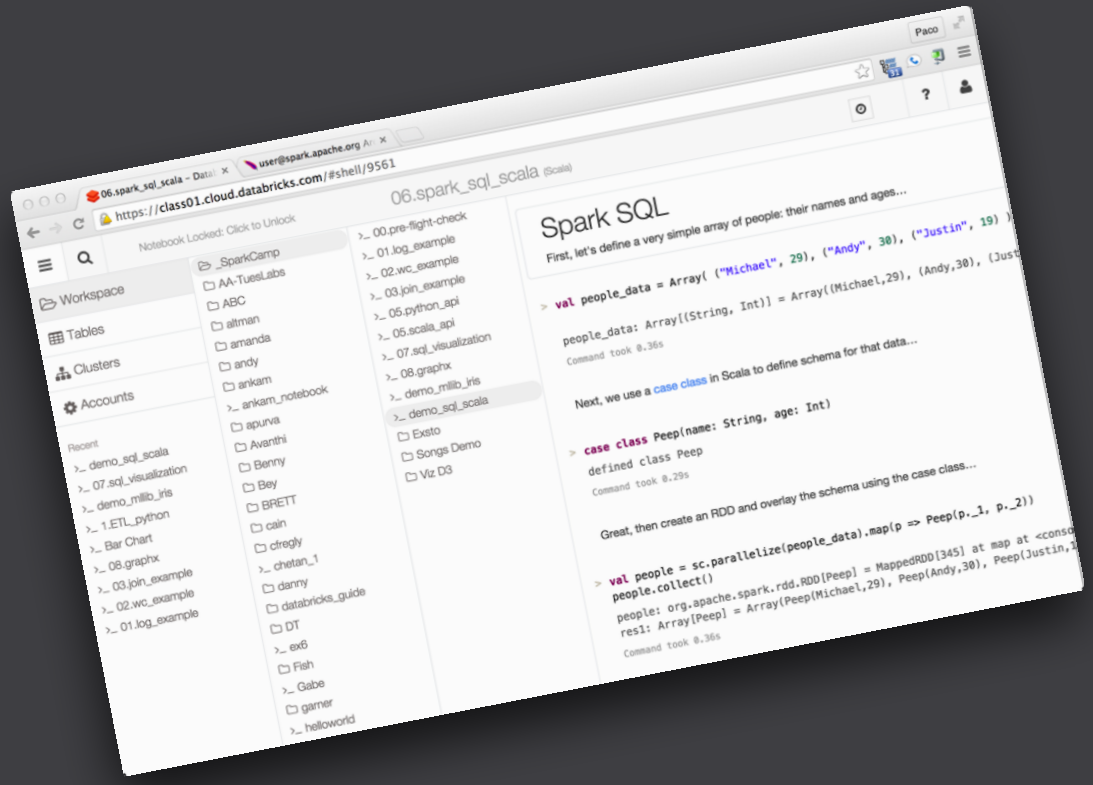
Clone and run `/_SparkCamp/03.join_example` in your folder:



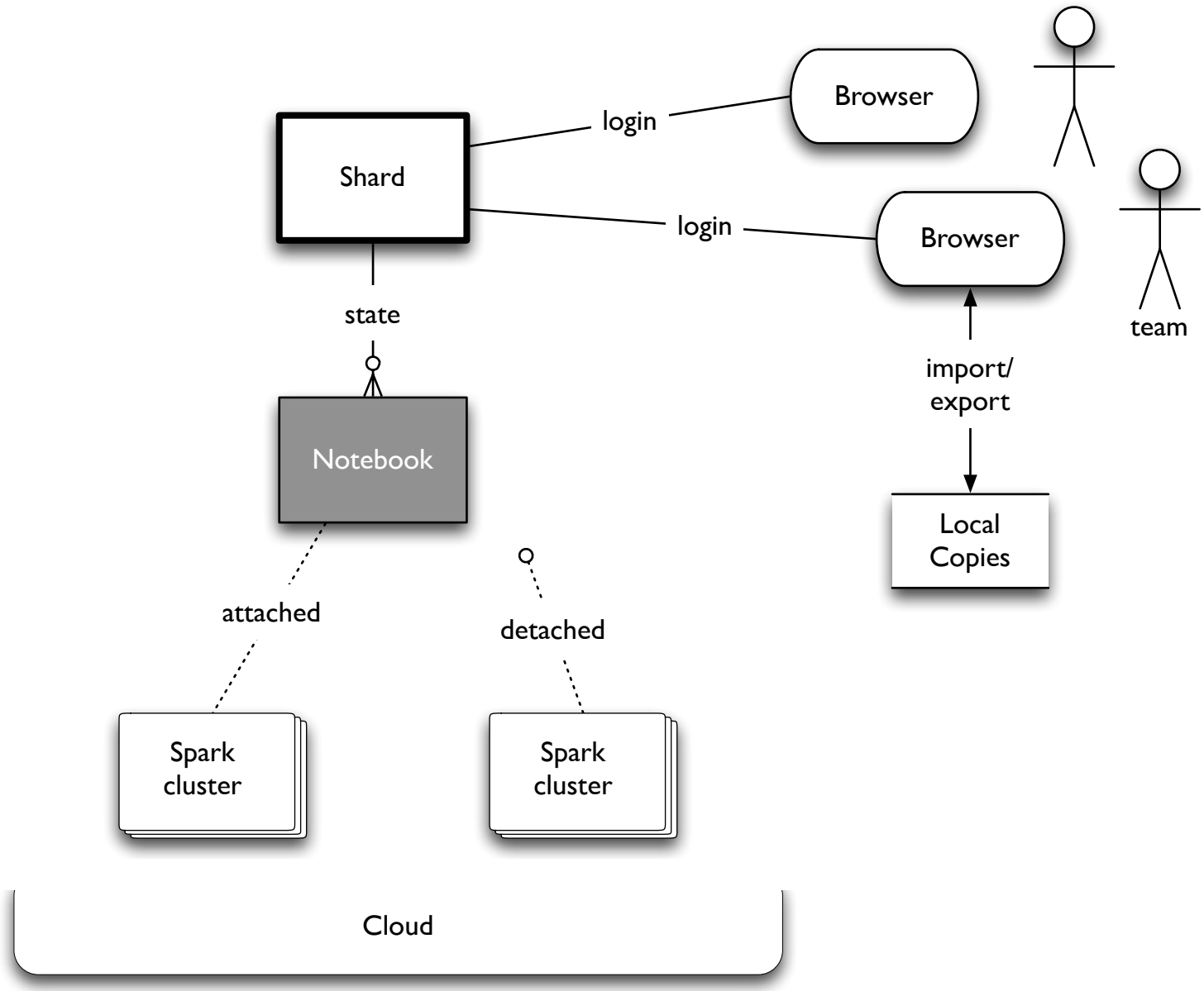
Coding Exercise: *Join and its Operator Graph*



How to “Think Notebooks”



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

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

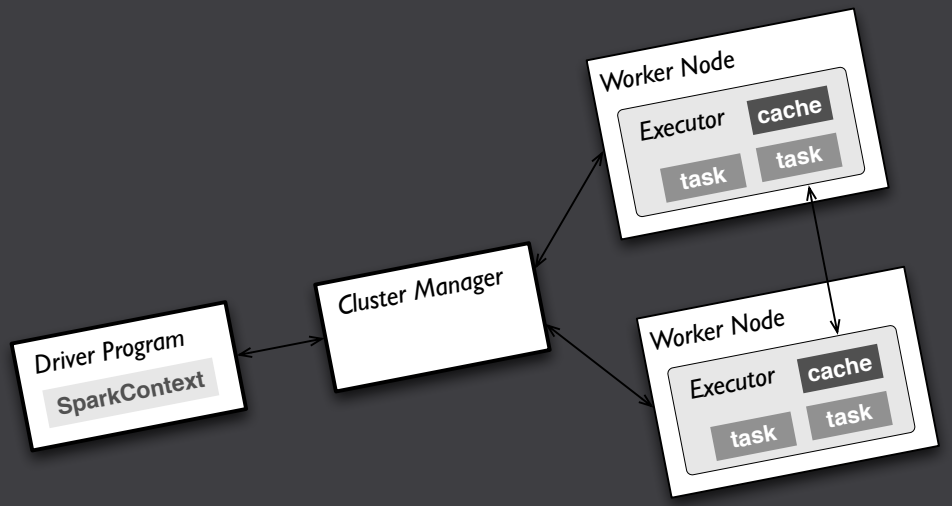
Coding Exercises: *Workflow assignment*

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: *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

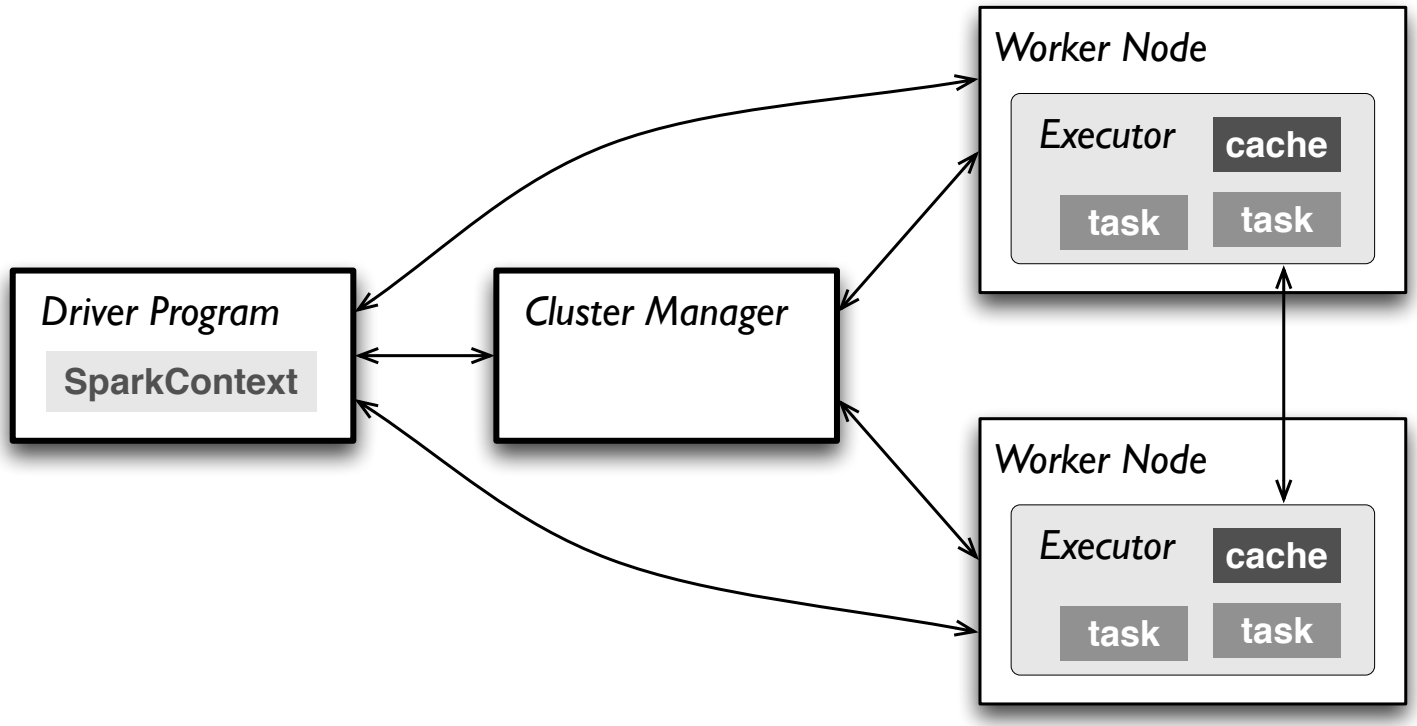
Spark Essentials: *Master*

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*

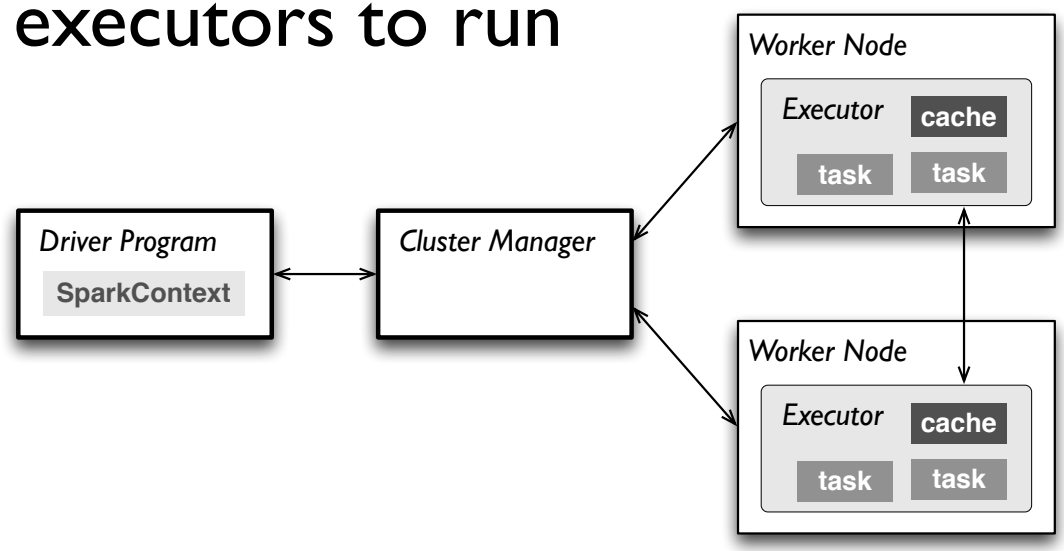
spark.apache.org/docs/latest/cluster-overview.html



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*

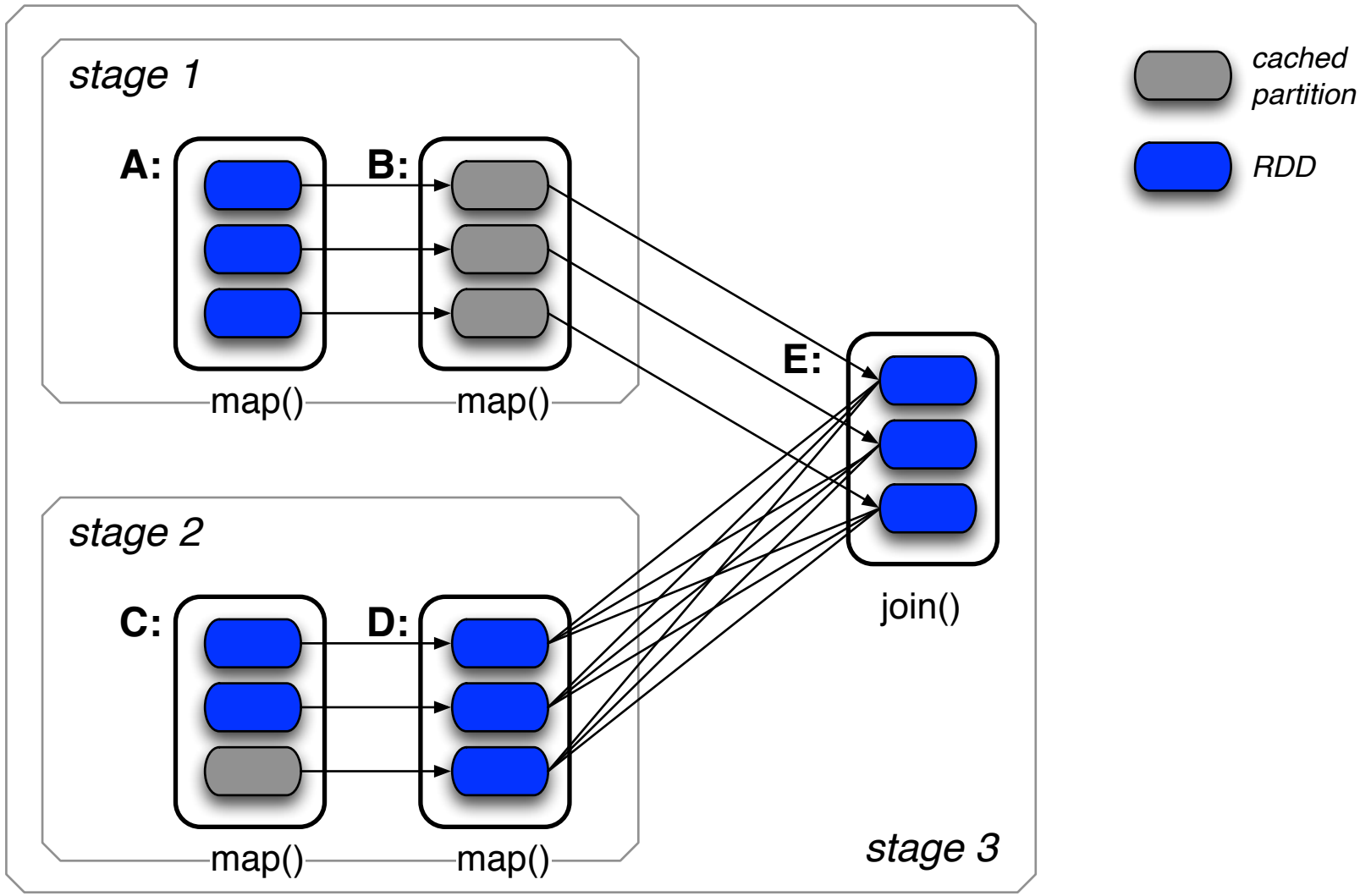
Scala:

```
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 and shuffles*



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: *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: *Persistence*

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

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

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

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

Spark Essentials: *Persistence*

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

Spark Essentials: *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
```

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*

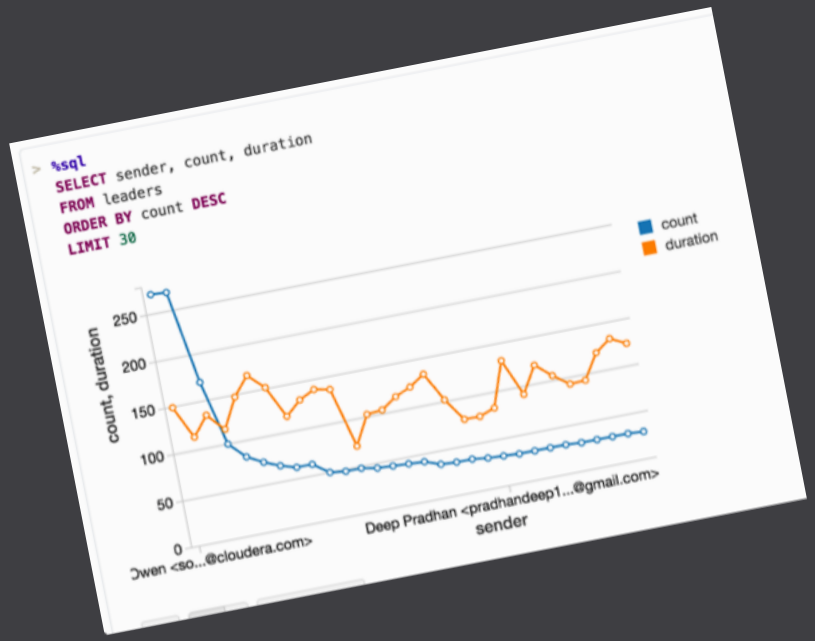
Scala:

```
val pair = (a, b)

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

Python:

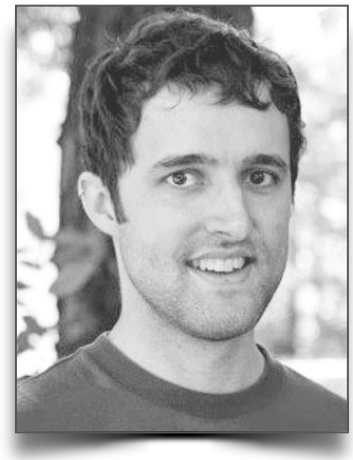
```
pair = (a, b)

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

Spark SQL + DataFrames



Spark SQL + DataFrames: *Suggested References*

Spark DataFrames:
Simple and Fast Analysis of Structured Data
Michael Armbrust
spark-summit.org/2015/events/spark-dataframes-simple-and-fast-analysis-of-structured-data/



For docs, see:

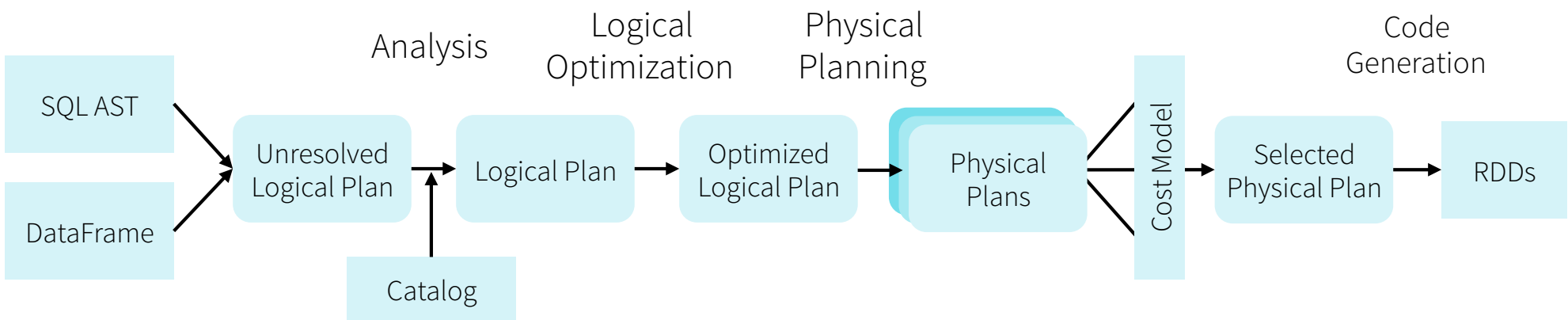
spark.apache.org/docs/latest/sql-programming-guide.html

Spark SQL + DataFrames: *Rationale*

- **DataFrame** model – allows expressive and concise programs, akin to Pandas, R, etc.
- pluggable **Data Source API** – reading and writing data frames while minimizing I/O
- **Catalyst** logical optimizer – optimization happens late, includes pushdown predicate, code gen, etc.
- columnar formats, e.g., **Parquet** – can skip fields
- **Project Tungsten** – optimizes physical execution throughout Spark

Spark SQL + DataFrames: *Optimization*

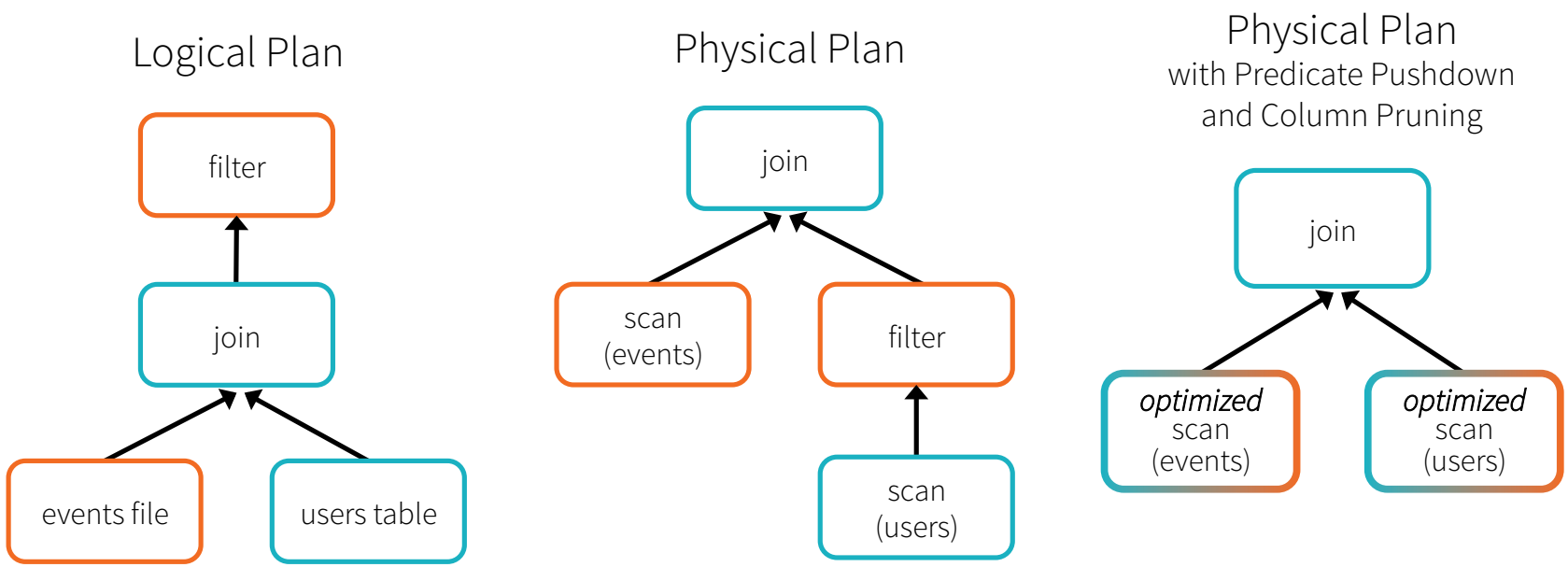
Plan Optimization & Execution



from Databricks

Spark SQL + DataFrames: *Optimization*

```
def add_demographics(events):
    u = sqlCtx.table("users")                    # Load partitioned Hive table
    events \
      .join(u, events.user_id == u.user_id) \    # Join on user_id
      .withColumn("city", zipToCity(u.zip))      # Run udf to add city column

events = add_demographics(sqlCtx.load("/data/events", "parquet"))
training_data = events.where(events.city == "New York").select(events.timestamp).collect()
```



Spark SQL + DataFrames: *Using 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 schema of original data

See also:

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

Spark SQL + DataFrames: *Code Example*

Identify the people who sent more than thirty messages on the **user@spark.apache.org** email list during January 2015…

on Databricks:

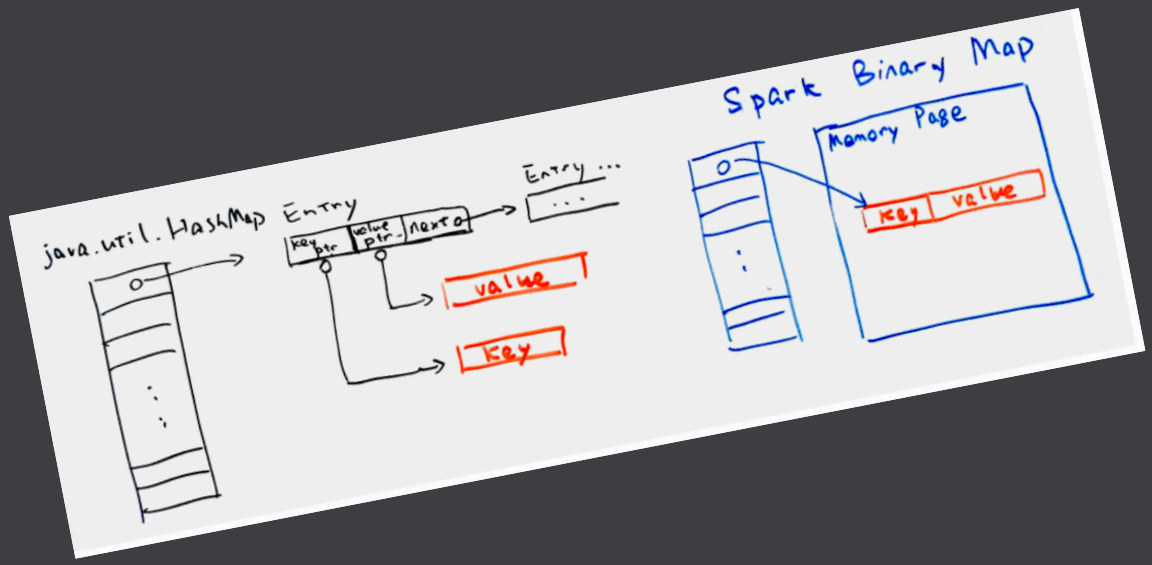
- `/mnt/paco/exsto/original/2015_01.json`

otherwise:

- download directly from **S3**

For more details, see: `/_SparkCamp/Exsto/`

Tungsten


java.util.HashMap
Entry
key ptr
value ptr
next
Entry ...
value
Key
Spark Binary Map
Memory Page
key
value

Tungsten: *Suggested References*

Deep Dive into Project Tungsten:
Bringing Spark Closer to Bare Metal

Josh Rosen

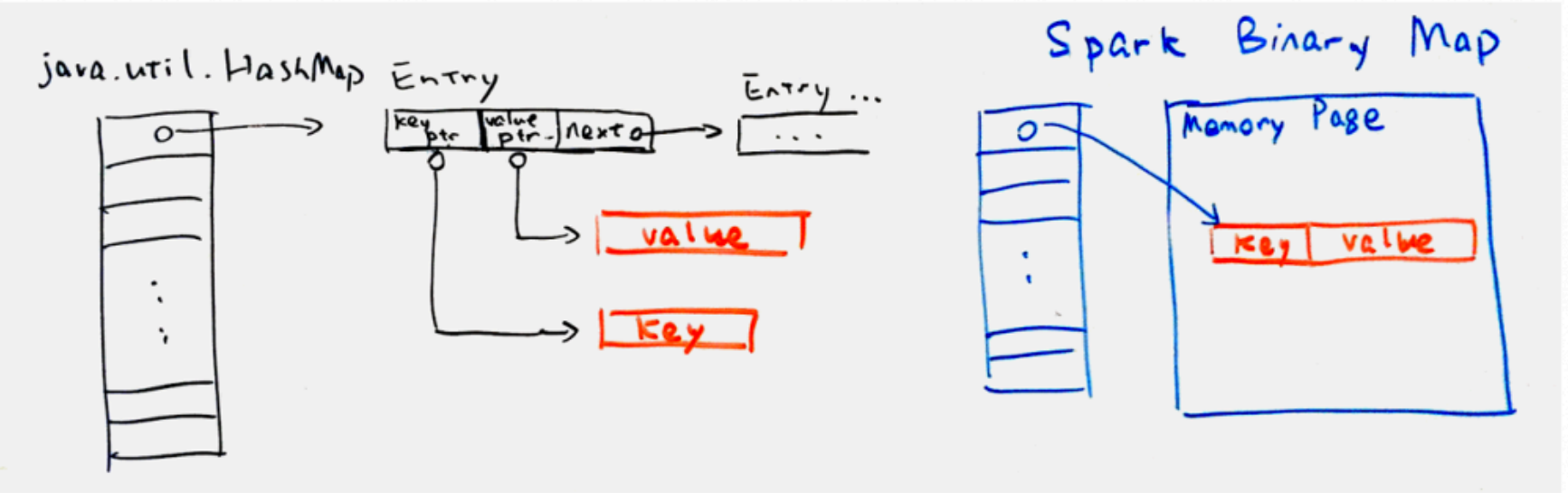
spark-summit.org/2015/events/deep-dive-into-project-tungsten-bringing-spark-closer-to-bare-metal/

Tungsten: *Roadmap*

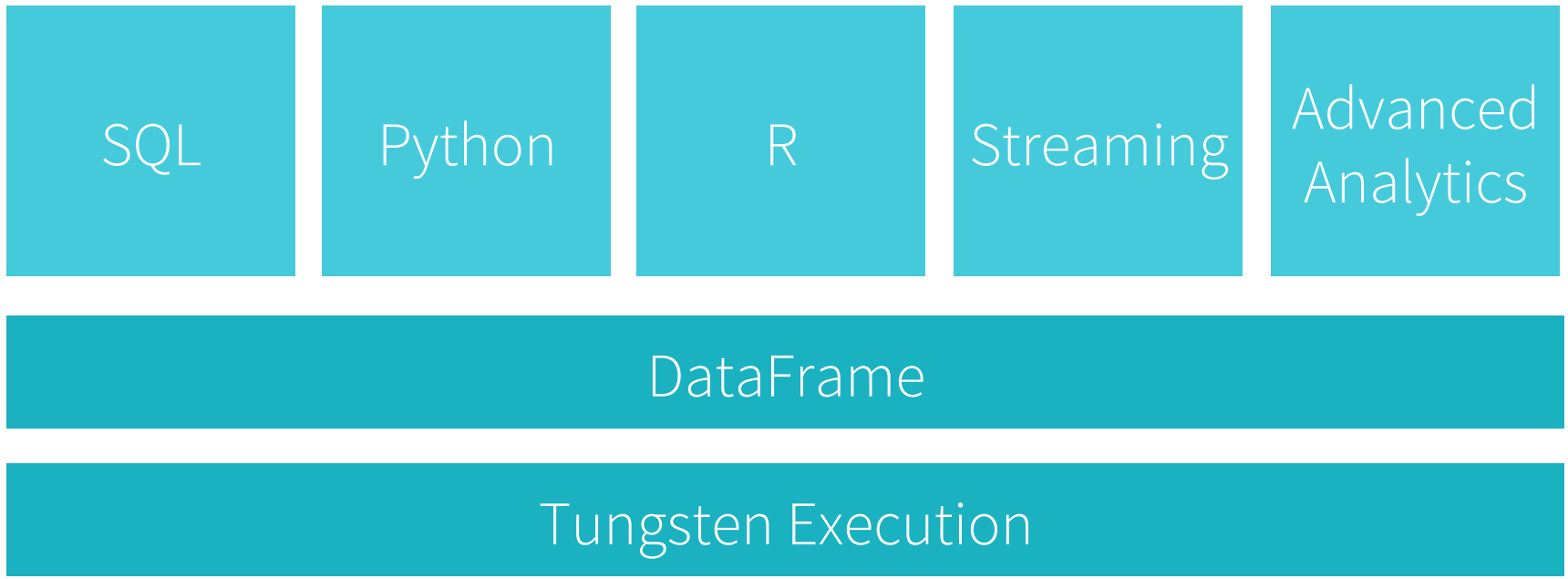
- early features are experimental in Spark 1.4
- new shuffle managers
- compression and serialization optimizations
- custom binary format and off-heap managed memory – faster and "GC-free"
- expanded use of code generation
- vectorized record processing
- exploiting cache locality

Physical Execution: CPU Efficient Data Structures

Keep data closure to CPU cache



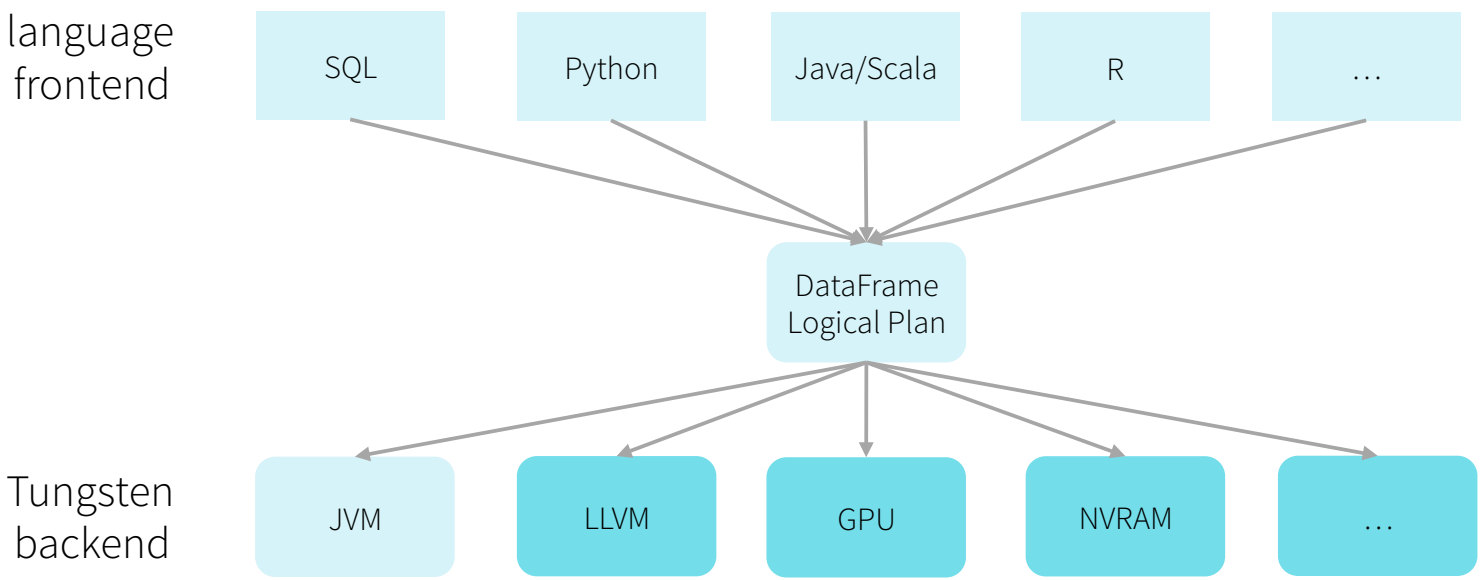
Tungsten: *Optimization*



from Databricks

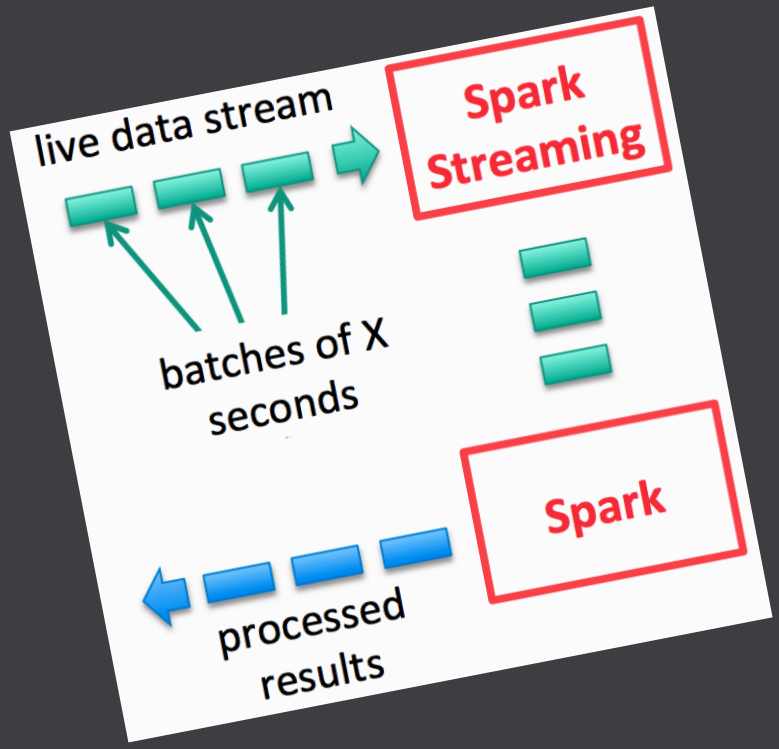
Tungsten: *Optimization*

Unified API, One Engine, Automatically Optimized



from Databricks

Spark Streaming



Spark Streaming: *Requirements*

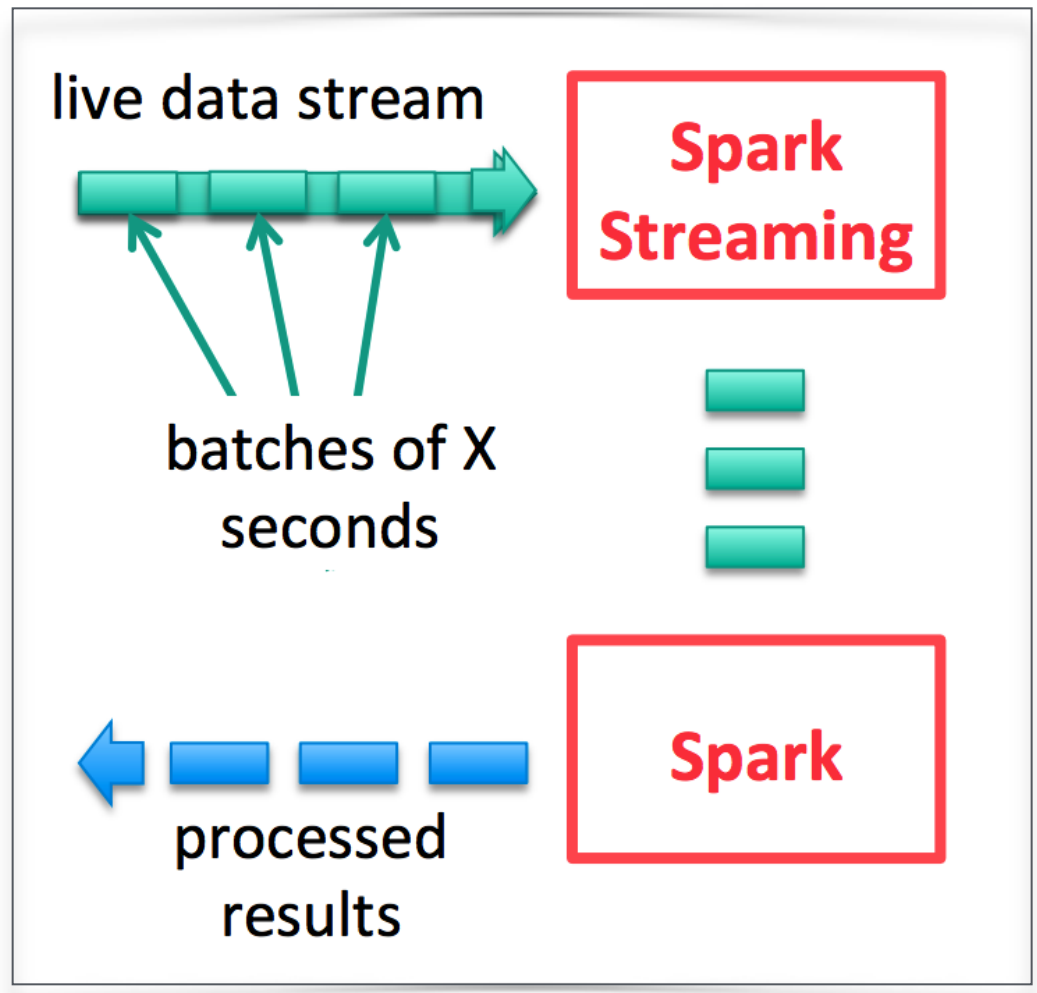
Let's consider the top-level requirements for a streaming framework:

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

Spark Streaming: *Requirements*

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

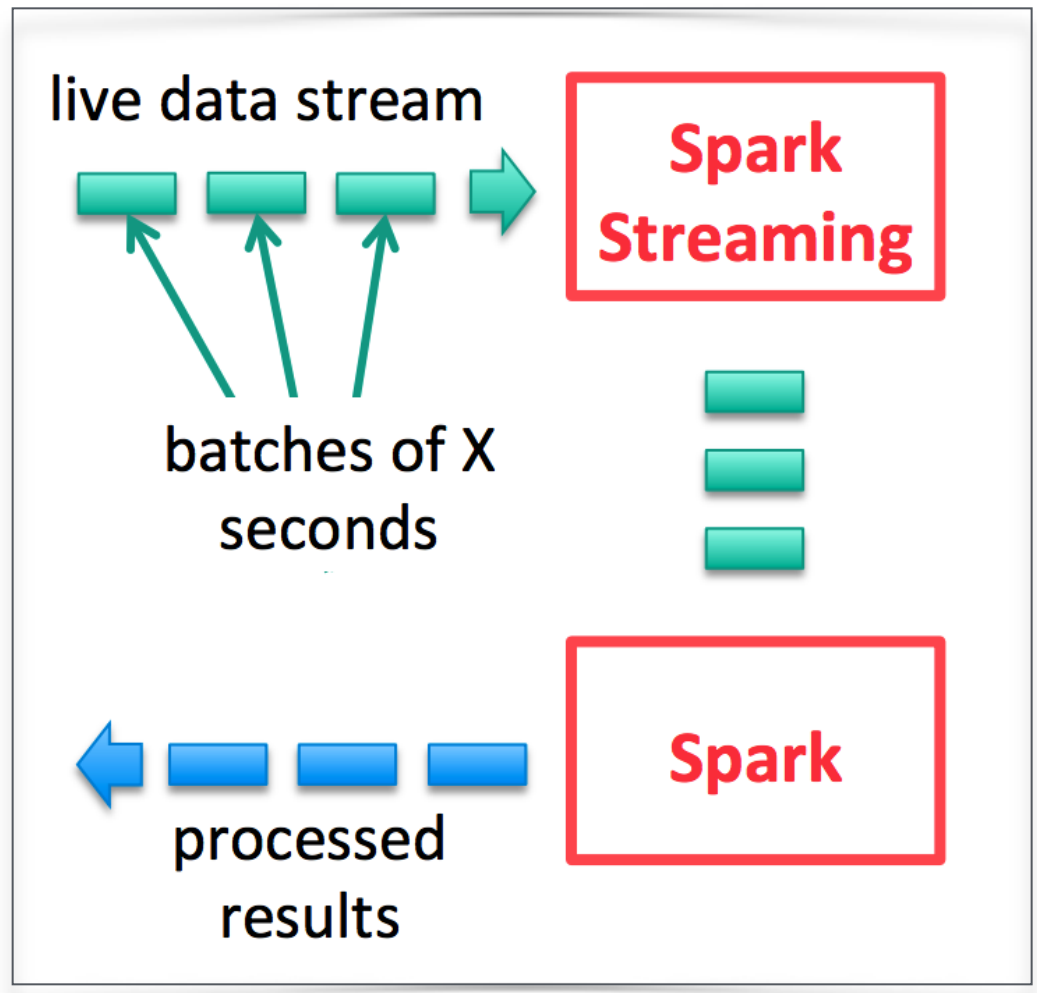
- *Chop up the live stream into batches of X seconds*
- *Spark treats each batch of data as RDDs and processes them using RDD operations*
- *Finally, the processed results of the RDD operations are returned in batches*



Spark Streaming: *Requirements*

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

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



Spark Streaming: *Integration*

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

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

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

Spark Streaming: *Micro Batch*

Because Google!

MillWheel: Fault-Tolerant Stream Processing at Internet Scale
Tyler Akidau, **Alex Balikov**, **Kaya Bekiroglu**, **Slava Chernyak**, **Josh Haberman**, **Reuven Lax**, **Sam McVeety**, **Daniel Mills**, **Paul Nordstrom**, **Sam Whittle**
Very Large Data Bases (2013)
research.google.com/pubs/pub41378.html

Spark Streaming: *Timeline*

2012 project started

2013 alpha release (Spark 0.7)

2014 graduated (Spark 0.9)

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

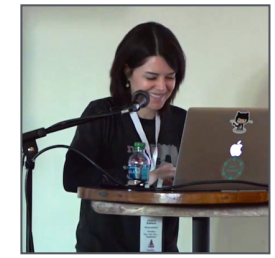
project lead:
Tathagata Das **@tathadas**

Spark Streaming: *Community – A Selection of Thought Leaders*

David Morales
Stratio
@dmoralesdf

Krishna Gade
Pinterest
@krishnagade

Claudiu Barbura
Atigeo
@claudiubarbura



Helena Edelson
DataStax
@helenaedelson

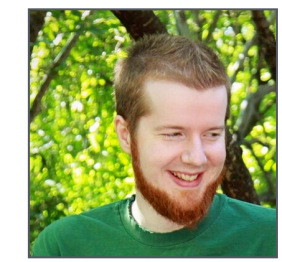
Eric Carr
Guavus
@guavus

Gerard Maas
Virdata
@maasg

Mayur Rustagi
Sigmoid Analytics
@mayur_rustagi

Russell Cardullo
Sharethrough
@russellcardullo

Jeremy Freeman
HHMI Janelia
@thefreemanlab



Cody Koeninger
Kixer
@CodyKoeninger

Antony Arokiasamy
Netflix
@aasamy

Dibyendu Bhattacharya
Pearson
@maasg

Mansour Raad
ESRI
@mraad

Spark Streaming: *Some Excellent Resources*

Diving into Spark Streaming's Execution Model
Tathagata Das, Matei Zaharia, Patrick Wendell
databricks.com/blog/2015/07/30/diving-into-spark-streamings-execution-model.html

Spark Streaming @Strata CA 2015
slideshare.net/databricks/spark-streaming-state-of-the-union-strata-san-jose-2015

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

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

Spark Streaming: *Use Cases*

Typical kinds of applications:

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

and much much more!

Spark Streaming: *Example Code*

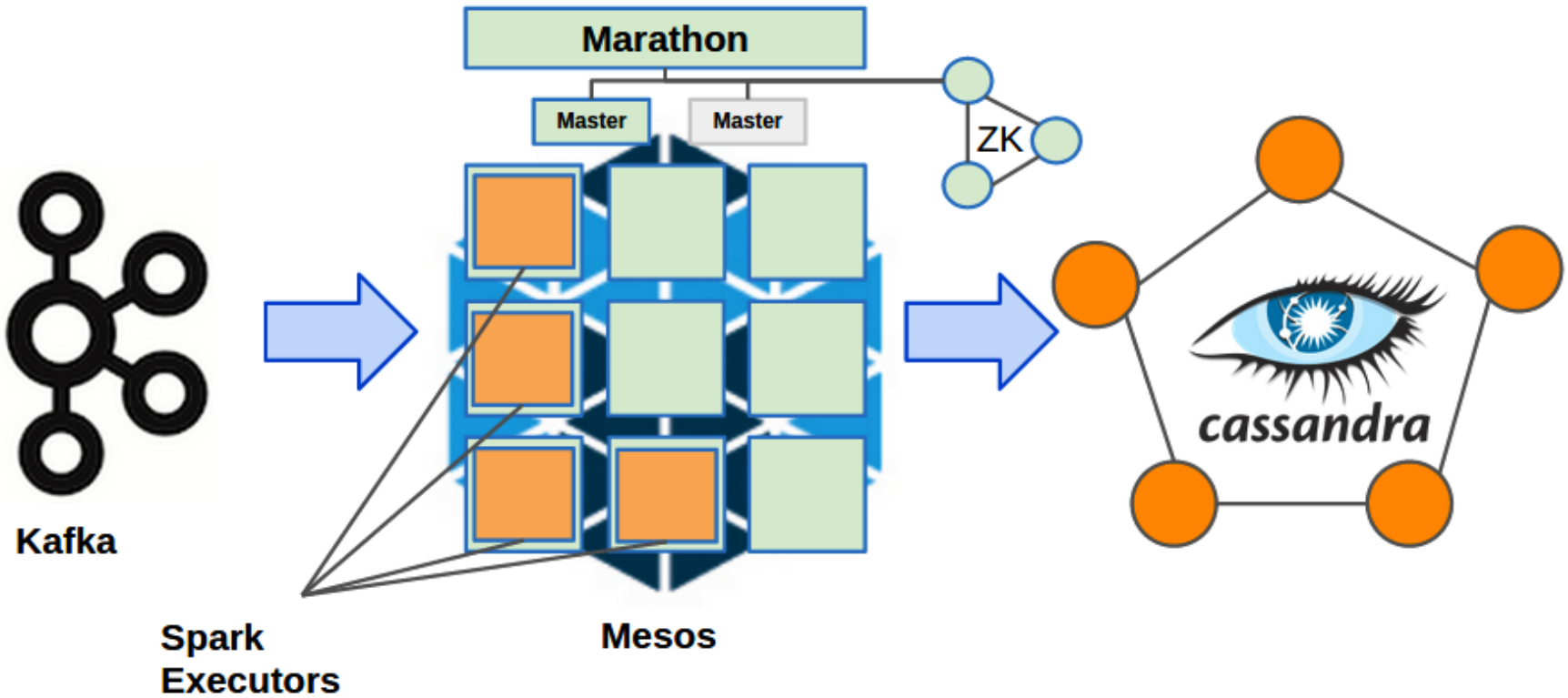
```
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._

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

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

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

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

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

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

Tuning: *Virdata tutorial*

Tuning Spark Streaming for Throughput

Gerard Maas, 2014-12-22

virdata.com/tuning-spark/

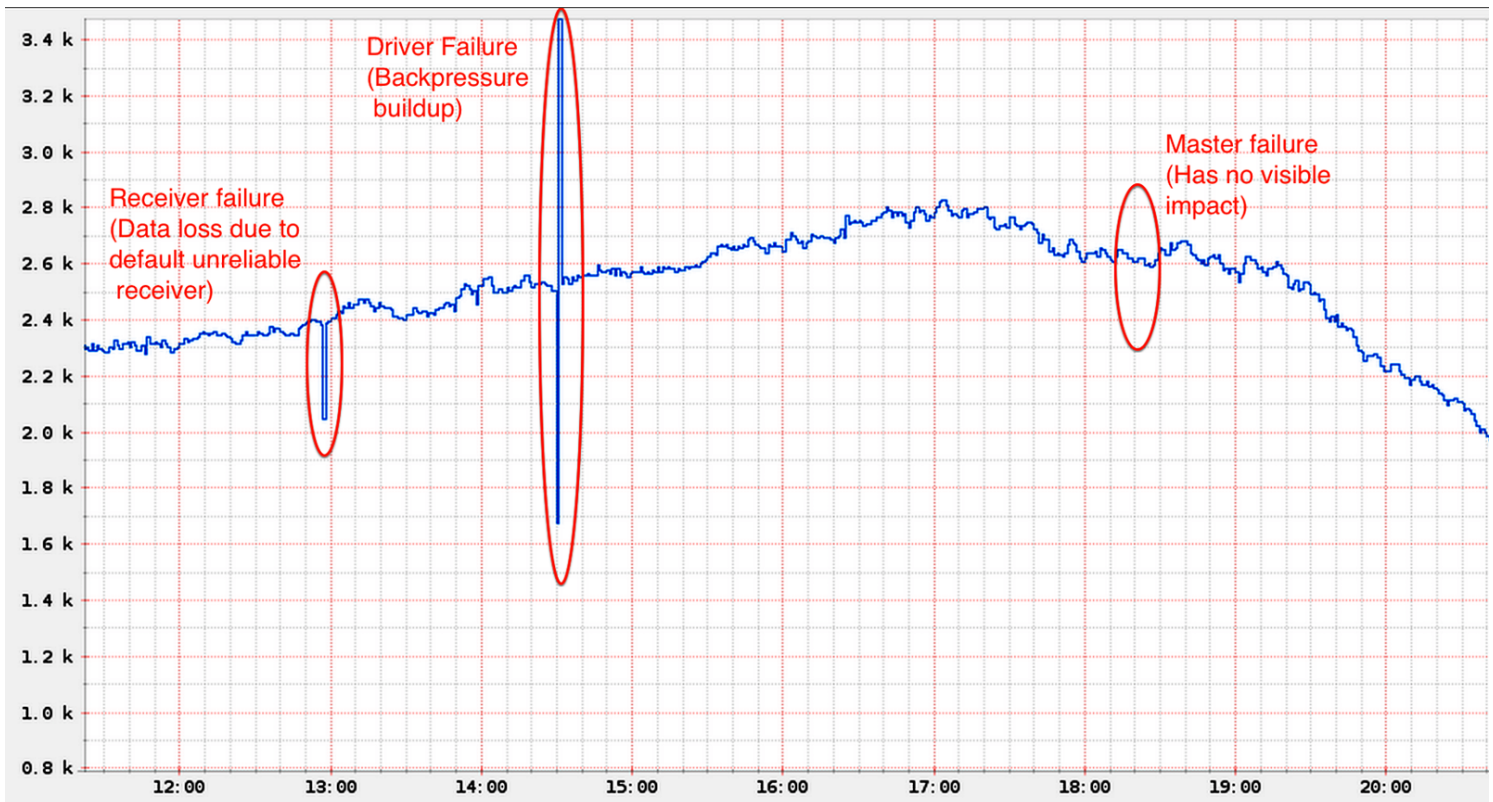


Resiliency: *Netflix tutorial*

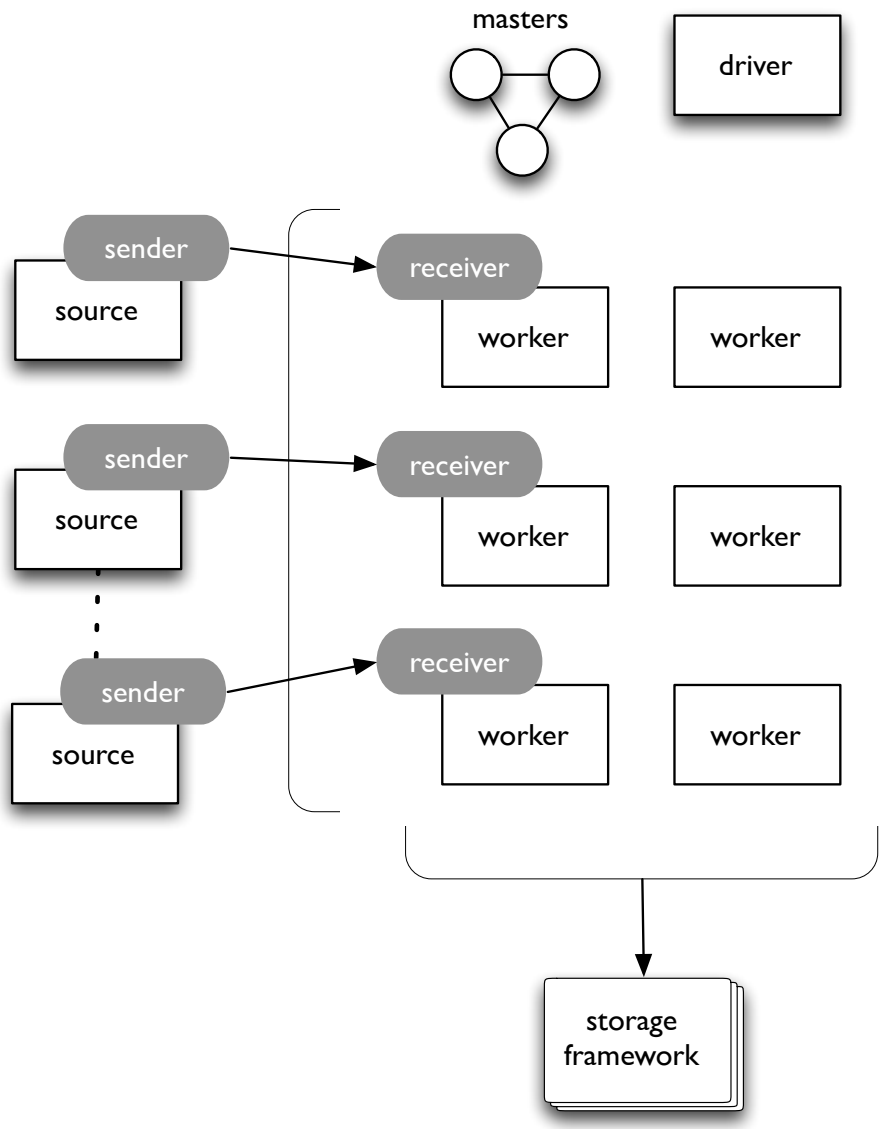
Can Spark Streaming survive Chaos Monkey?

Bharat Venkat, Prasanna Padmanabhan, Antony Arokiasamy, Raju Uppalapati

techblog.netflix.com/2015/03/can-spark-streaming-survive-chaos-monkey.html

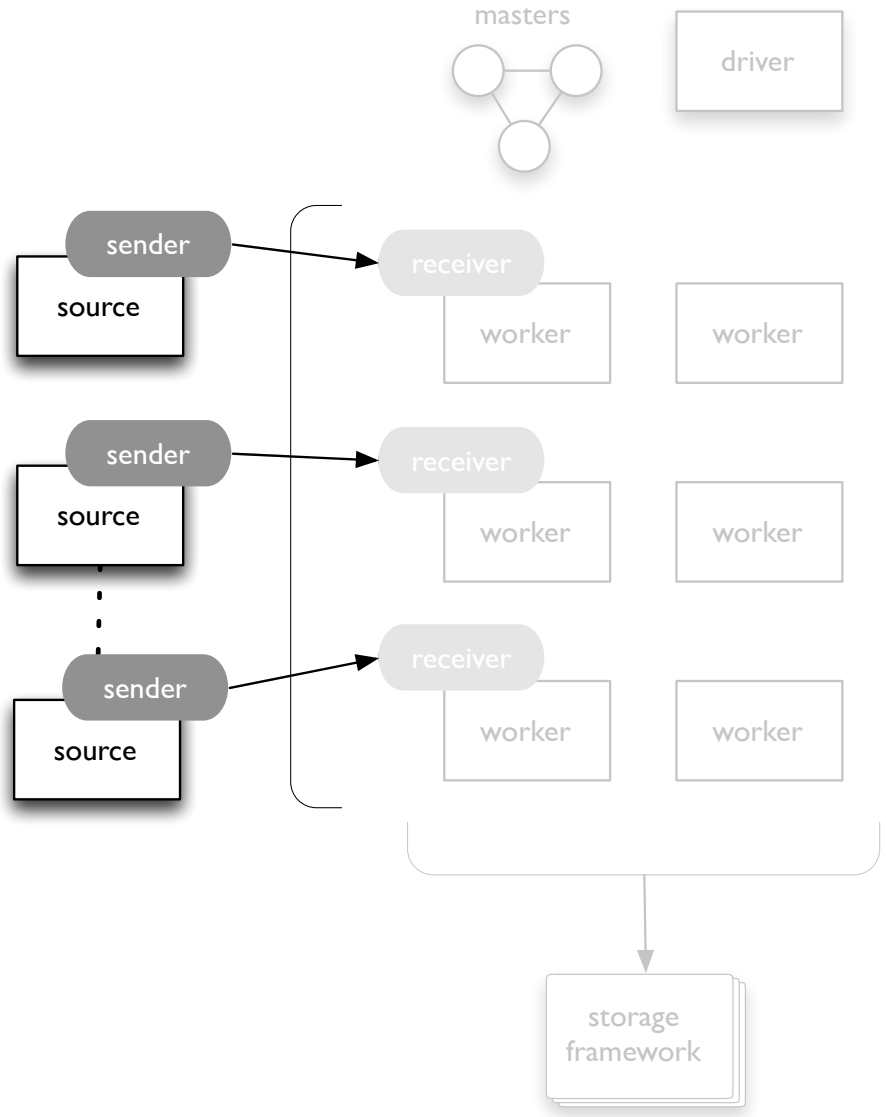


Resiliency: *illustrated*



(resiliency features)

Resiliency: *illustrated*



backpressure
(flow control is a hard problem)

reliable receiver

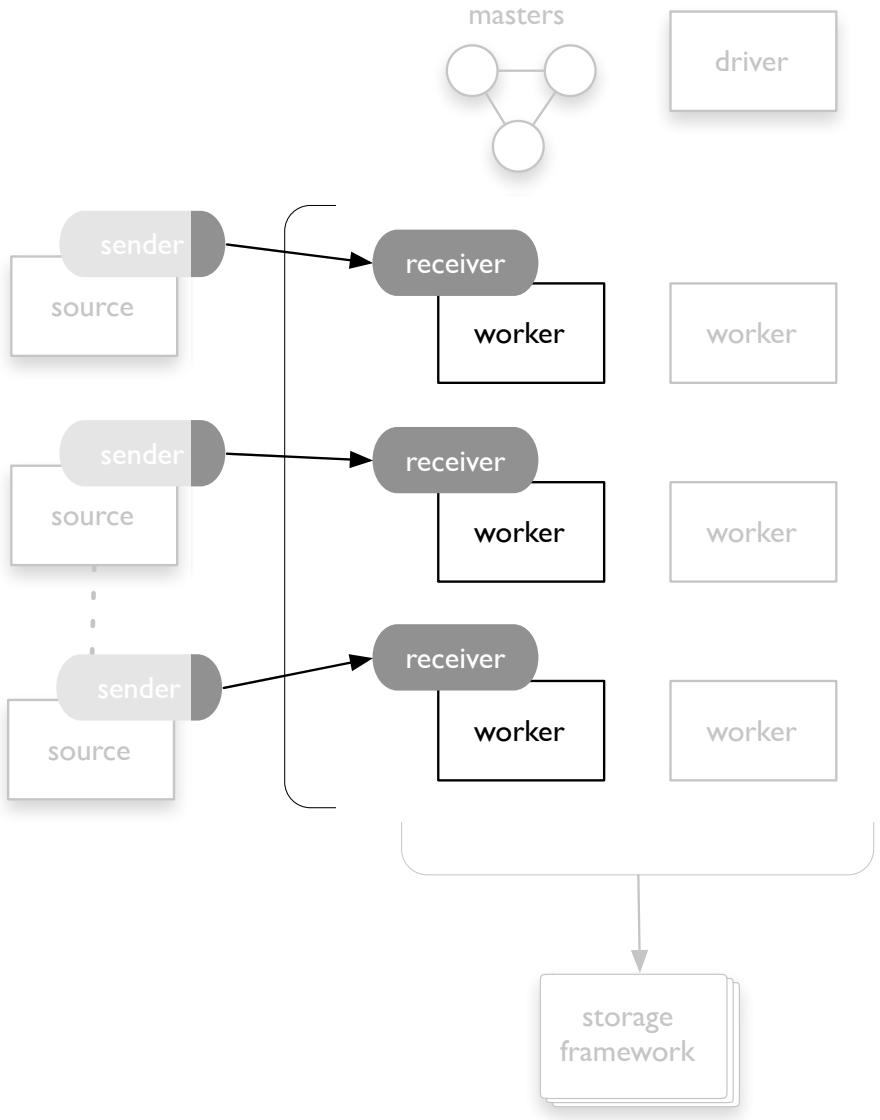
in-memory replication
write ahead log (data)

driver restart
checkpoint (metadata)

multiple masters

worker relaunch
executor relaunch

Resiliency: *illustrated*



backpressure
(flow control is a hard problem)

reliable receiver

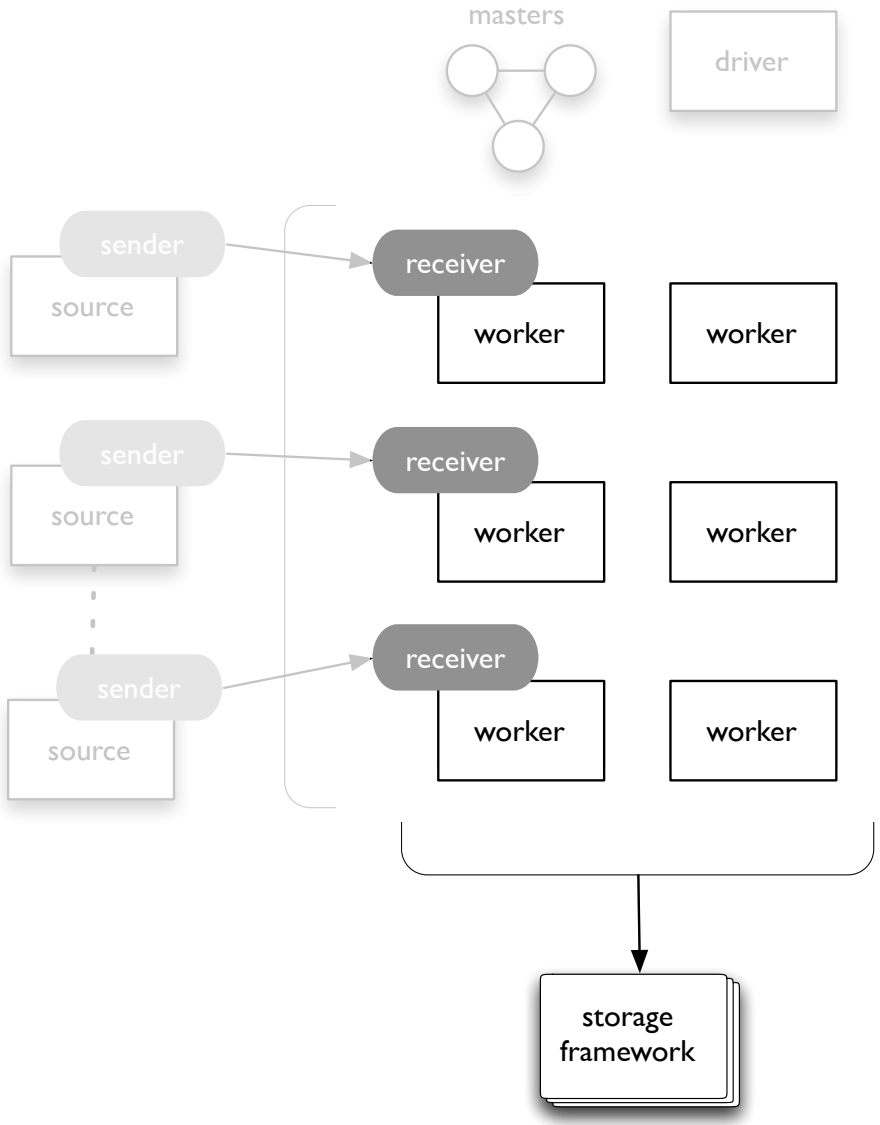
in-memory replication
write ahead log (data)

driver restart
checkpoint (metadata)

multiple masters

worker relaunch
executor relaunch

Resiliency: *illustrated*



backpressure
(flow control is a hard problem)

reliable receiver

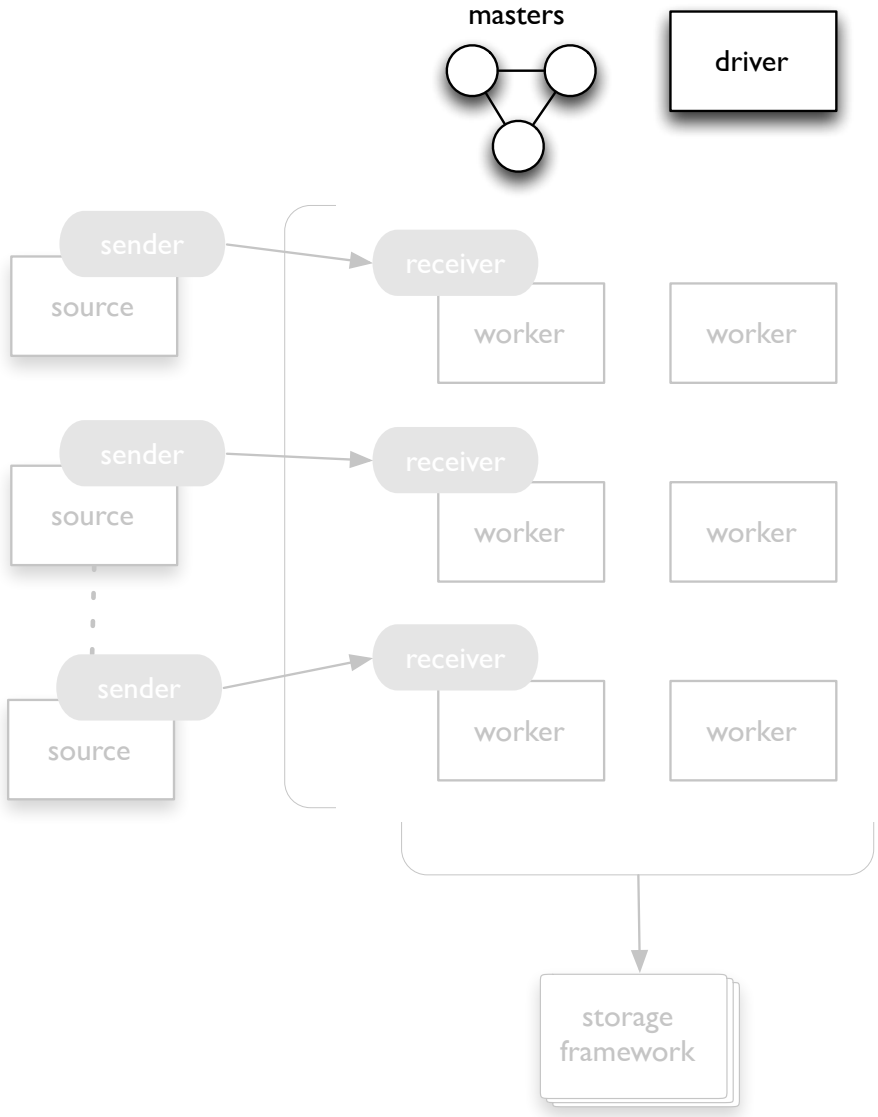
in-memory replication
write ahead log (data)

driver restart
checkpoint (metadata)

multiple masters

worker relaunch
executor relaunch

Resiliency: *illustrated*



backpressure
(flow control is a hard problem)

reliable receiver

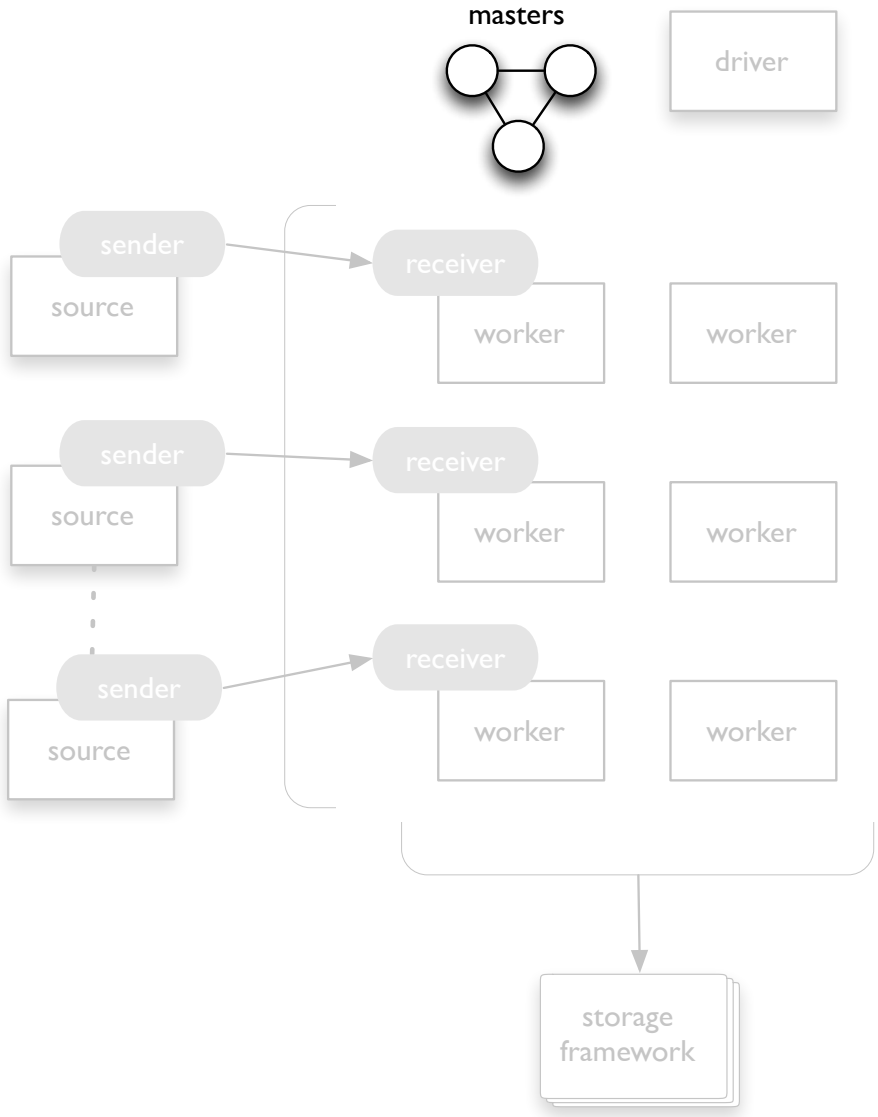
in-memory replication
write ahead log (data)

driver restart
checkpoint (metadata)

multiple masters

worker relaunch
executor relaunch

Resiliency: *illustrated*



backpressure
(flow control is a hard problem)

reliable receiver

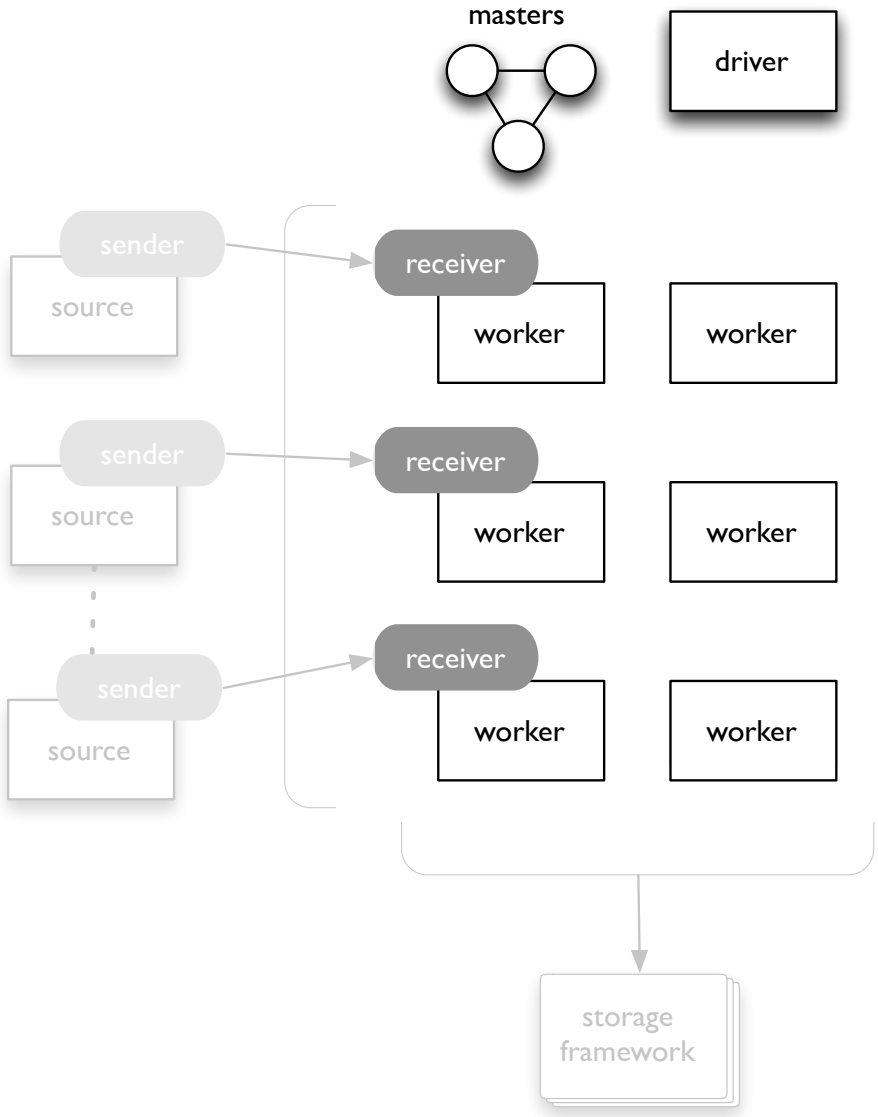
in-memory replication
write ahead log (data)

driver restart
checkpoint (metadata)

multiple masters

worker relaunch
executor relaunch

Resiliency: *illustrated*



backpressure
(flow control is a hard problem)

reliable receiver

in-memory replication
write ahead log (data)

driver restart
checkpoint (metadata)

multiple masters

worker relaunch
executor relaunch

Integrations: *architectural pattern deployed frequently in the field…*

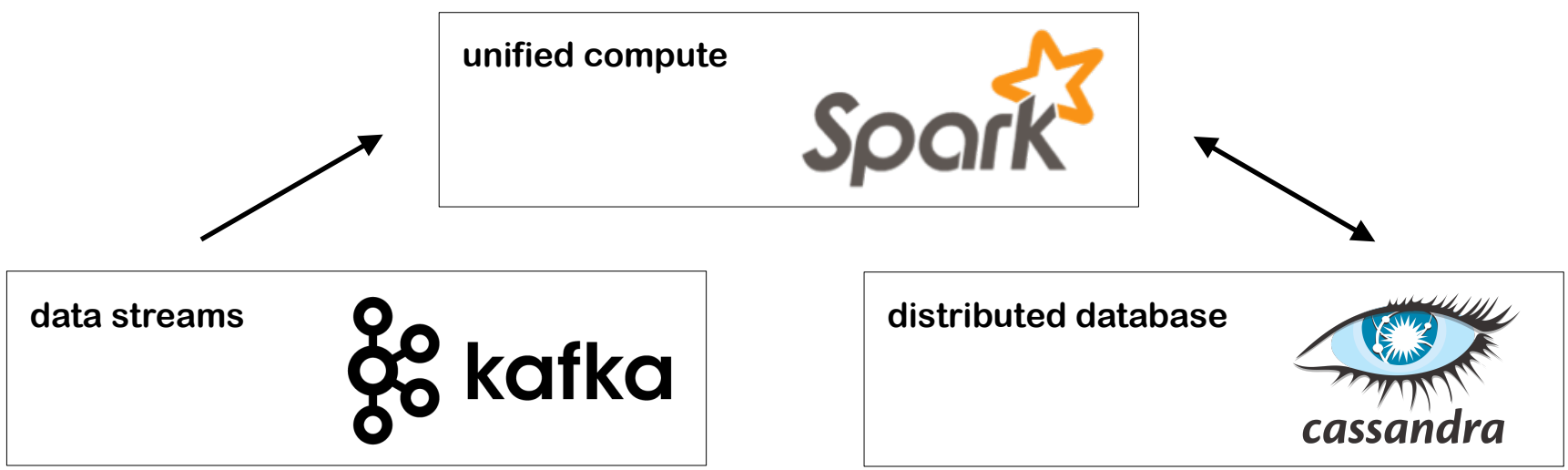
Kafka + Spark + Cassandra

datastax.com/documentation/datastax_enterprise/4.7/datastax_enterprise/spark/sparkIntro.html

http://helenaedelson.com/?p=991

github.com/datastax/spark-cassandra-connector

github.com/dibbhatt/kafka-spark-consumer



Integrations: *rich search, immediate insights*

Spark + ElasticSearch

databricks.com/blog/2014/06/27/application-spotlight-elasticsearch.html

elasticsearch.org/guide/en/elasticsearch/hadoop/current/spark.html

spark-summit.org/2014/talk/streamlining-search-indexing-using-elastic-search-and-spark

Because Use Cases: *+80 known production use cases*

Because Use Cases: *Stratio*

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

- Stratio Streaming is the union of a real-time messaging bus with a complex event processing engine using Spark Streaming
- allows the creation of streams and queries on the fly
- paired with Siddhi CEP engine and Apache Kafka
- added global features to the engine such as auditing and statistics
- use cases: large banks, retail, travel, etc.
- using Apache Mesos

Because Use Cases: *Pearson*

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

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

Because Use Cases: *Guavus*

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

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

Because Use Cases: *Sharethrough*

sharethrough

Spark Streaming for Realtime Auctions
Russell Cardullo
2014-06-30
slideshare.net/RussellCardullo/russell-cardullo-spark-summit-2014-36491156

- the profile of a 24 x 7 streaming app is different than an hourly batch job…
- data sources from RabbitMQ, Kinesis
- ingest ~0.5 TB daily, mainly click stream and application logs, 5 sec micro-batch
- feedback based on click stream events into auction system for model correction
- monoids… using **Algebird**
- using Apache Mesos on AWS

Because Use Cases: *Freeman Lab, Janelia*

hhmi | janelia Research Campus

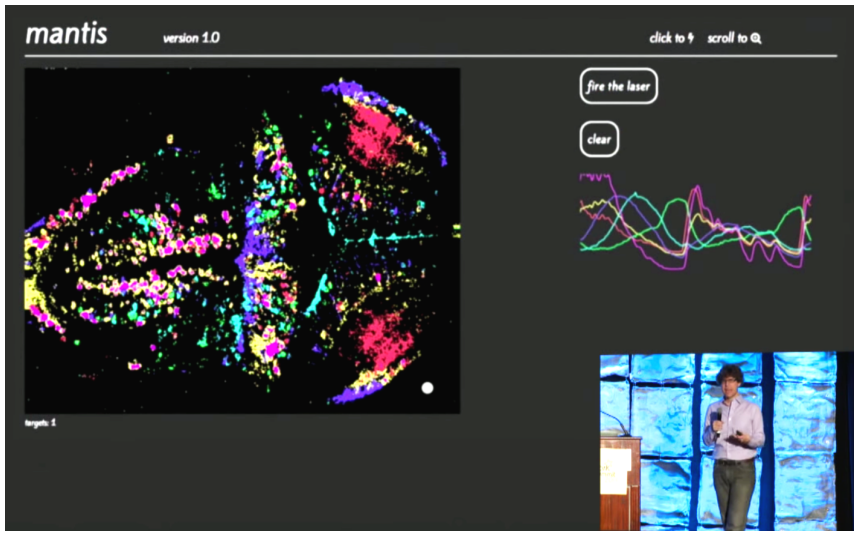
Analytics + Visualization for Neuroscience: Spark, Thunder, Lightning

Jeremy Freeman

2015-01-29

youtu.be/cBQm4LhHn9g?t=28m55s

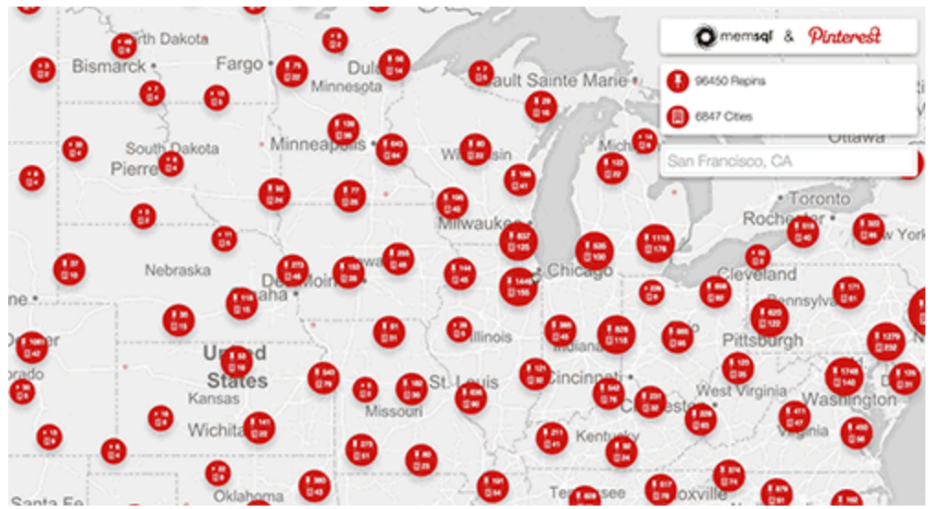
- genomics research – zebrafish neuroscience studies
- real-time ML for laser control
- 2 TB/hour per fish
- 80 HPC nodes



Because Use Cases: *Pinterest*

Real-time analytics at Pinterest
Krishna Gade
2015-02-18
engineering.pinterest.com/post/111380432054/real-time-analytics-at-pinterest

- higher performance event logging
- reliable log transport and storage
- faster query execution on real-time data
- integrated with MemSQL



Because Use Cases: *Ooyala*

Productionizing a 24/7 Spark Streaming service on YARN

Issac Buenrostro, Arup Malakar

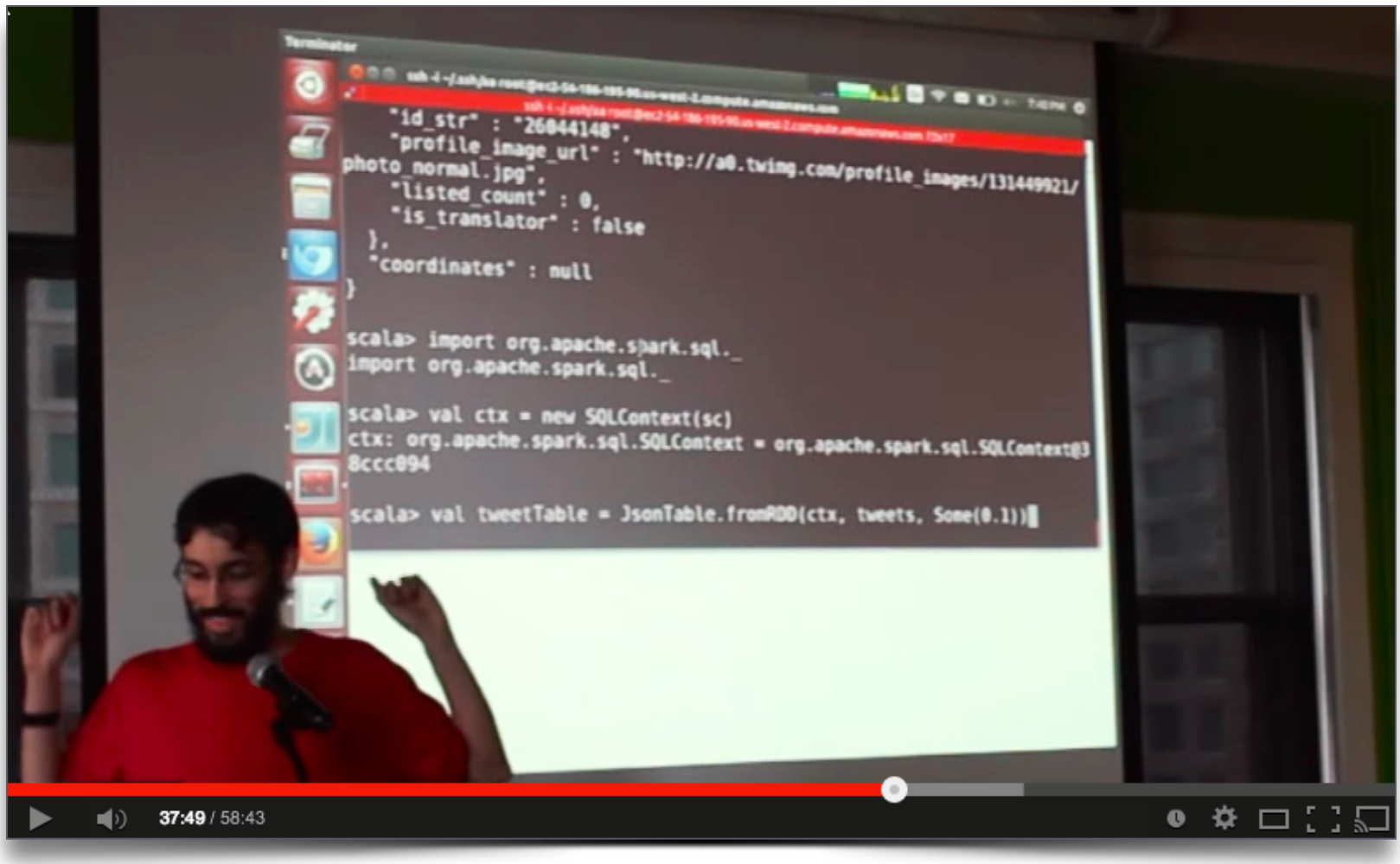
2014-06-30

spark-summit.org/2014/talk/productionizing-a-247-spark-streaming-service-on-yarn

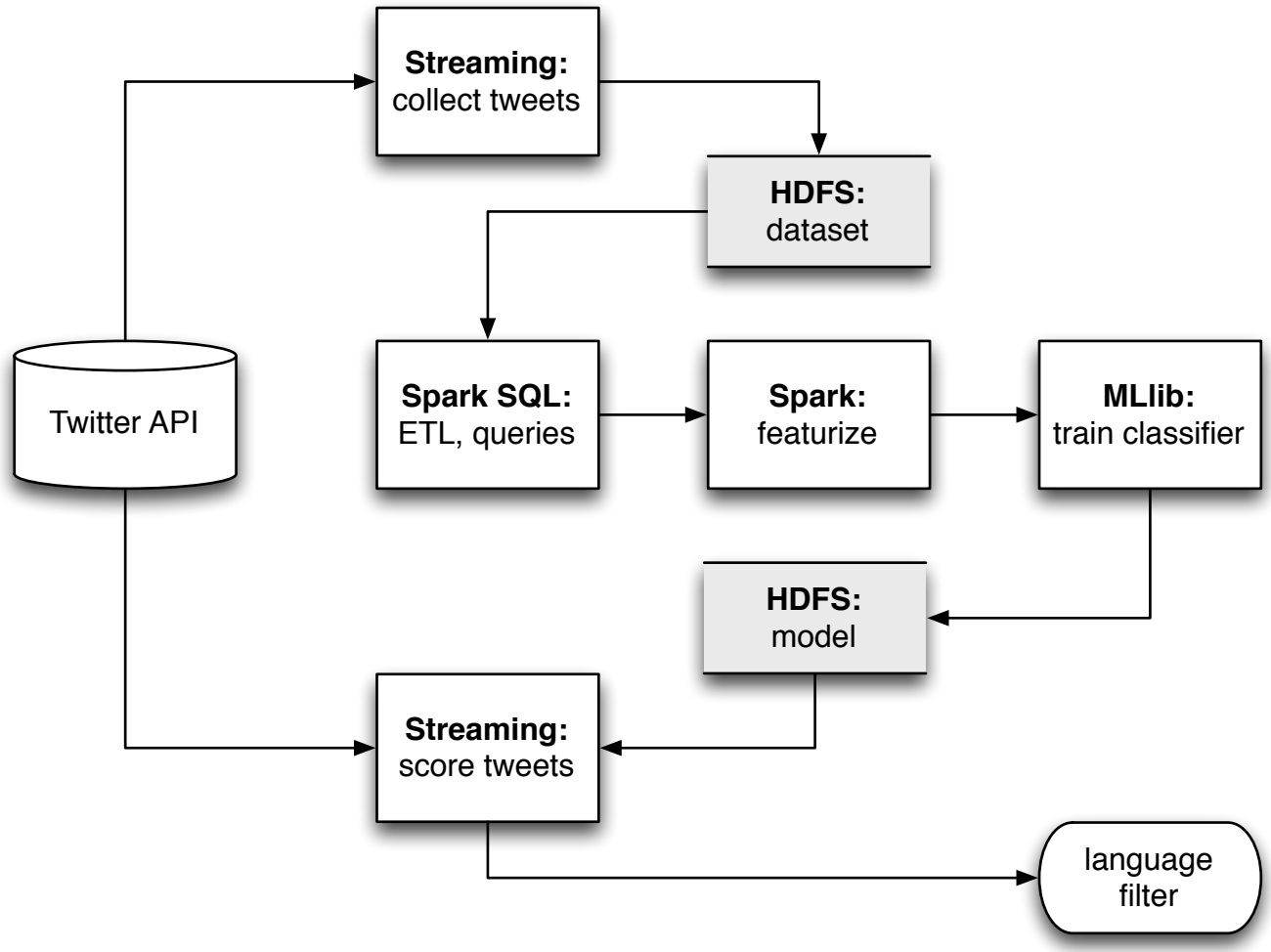
- state-of-the-art ingestion pipeline, processing over two billion video events a day
- how do you ensure 24/7 availability and fault tolerance?
- what are the best practices for Spark Streaming and its integration with Kafka and YARN?
- how do you monitor and instrument the various stages of the pipeline?

Demo: *Twitter Streaming Language Classifier*

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



Demo: *Twitter Streaming Language Classifier*



Demo: *Twitter Streaming Language Classifier*

From tweets to ML features, approximated as sparse vectors:

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

Demo: *Twitter Streaming Language Classifier*

Sample Code + Output:

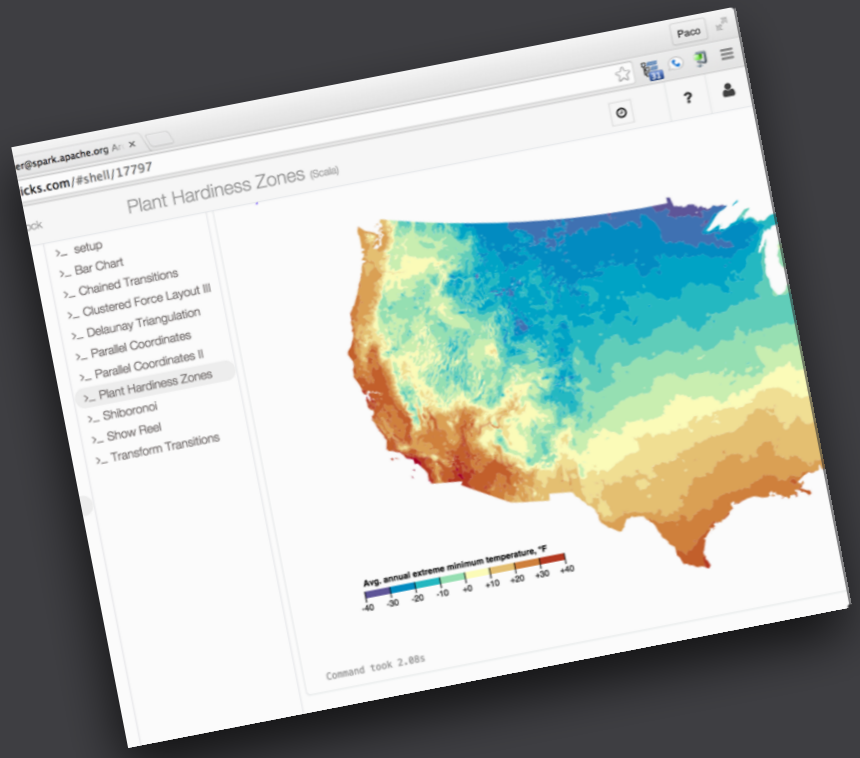
gist.github.com/ceteri/835565935da932cb59a2

```
val sc = new SparkContext(new SparkConf())
val ssc = new StreamingContext(conf, Seconds(5))

val tweets = TwitterUtils.createStream(ssc, Utils.getAuth)
val statuses = tweets.map(_.getText)

val model = new KMeansModel(ssc.sparkContext.objectFile[Vector]
(modelFile.toString).collect())

val filteredTweets = statuses
  .filter(t =>
    model.predict(Utils.featurize(t)) == clust)
filteredTweets.print()

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

CLUSTER 1:
TLあんまり見ないけど
@くれたっら
いつでもくっるよ٩(δωδ)۶
そういえばディスガイアも今日か

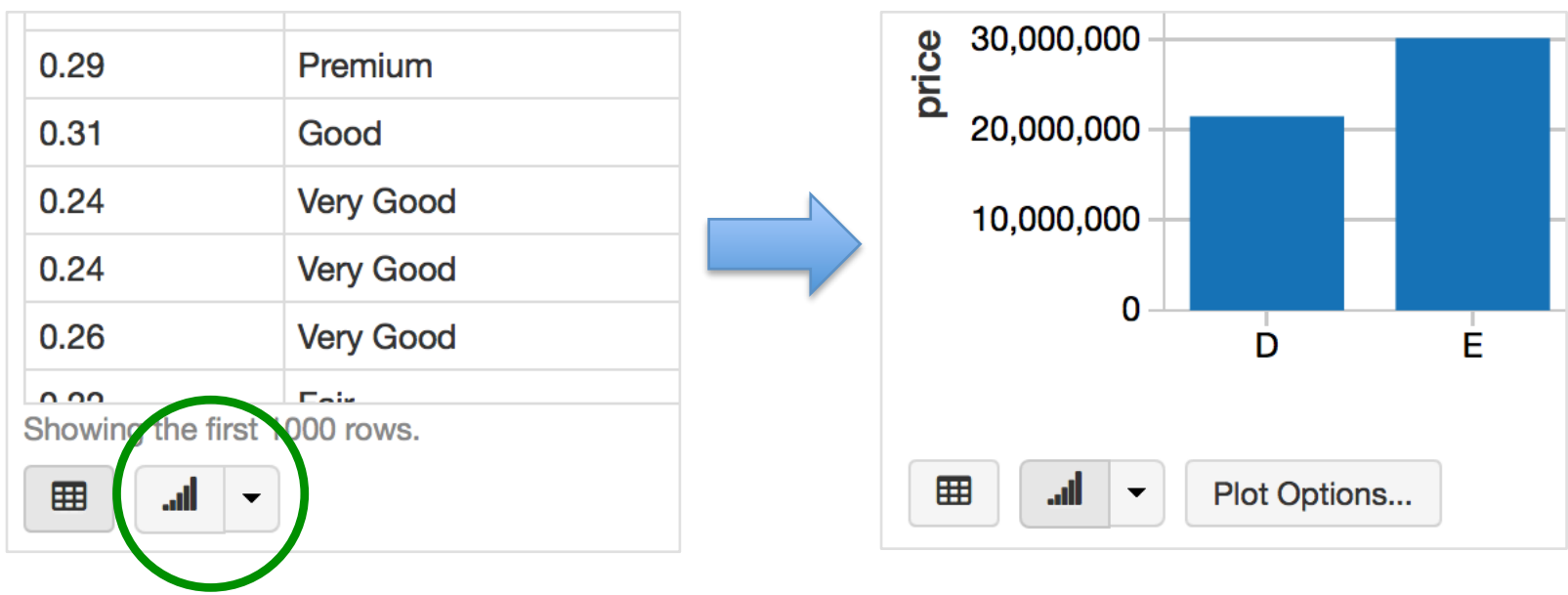
CLUSTER 4:
قالوا العروبه روحت بعد صدام
واقول مع سلمان تحيى العروبه
RT @vip588: √ فولو مي \ زيادة متابعين \ للمتواجدين الان
فولو باك \ رتويت للتغريدة \ فولو للي عمل رتويت \ √ vip588
اللي ما يلتزم ما بيستفيد ...
ن سورة

Visualization



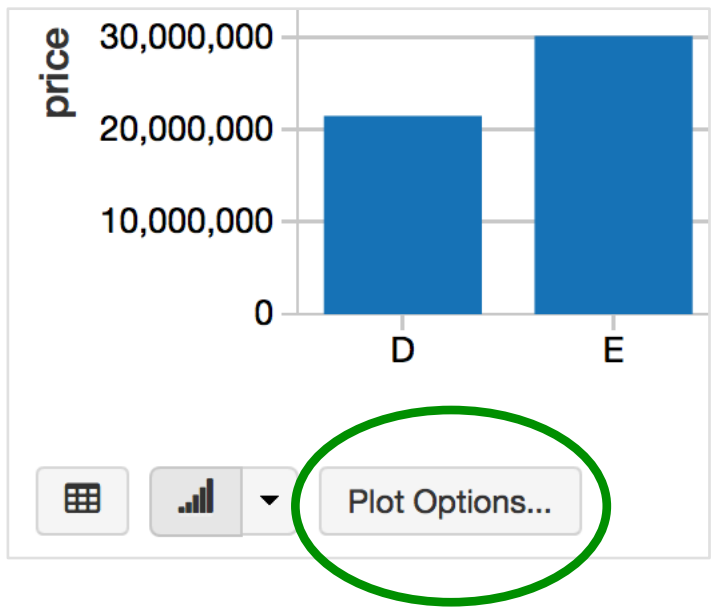
Visualization: *Built-in Plots*

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



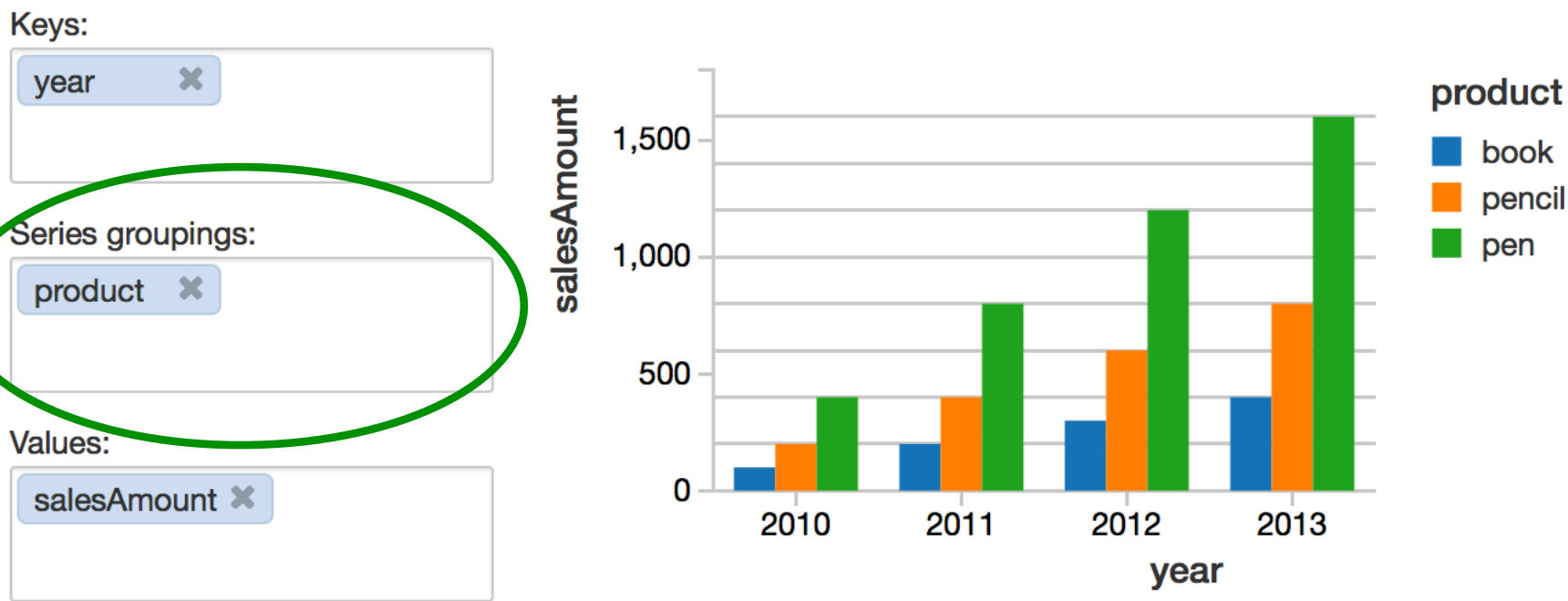
Visualization: *Plot Options*

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



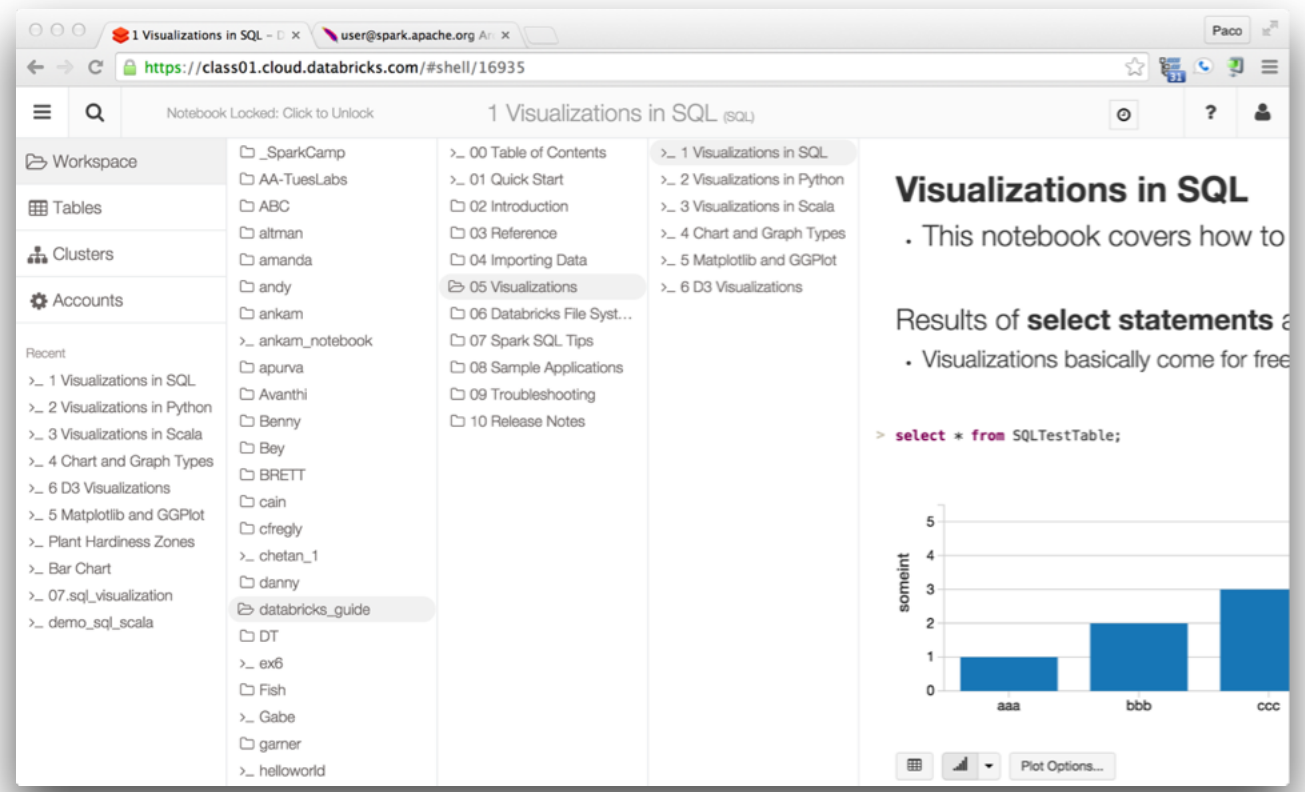
Visualization: *Series Groupings*

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



Visualization: *Reference Guide*

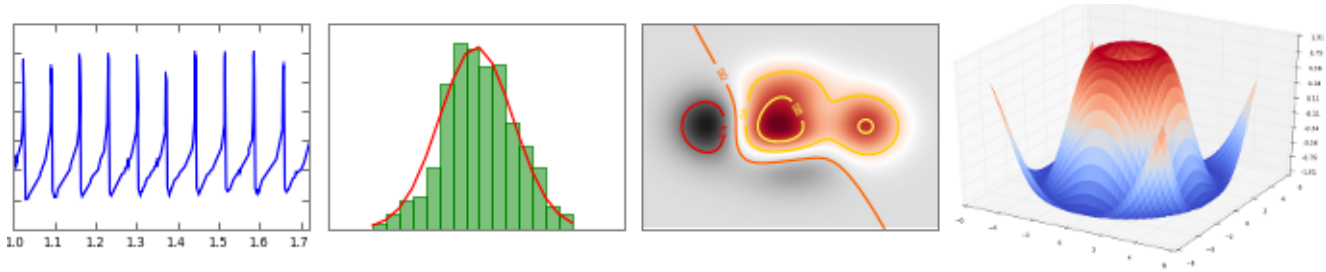
See `/databricks-guide/05 Visualizations` for details about built-in visualizations and extensions…



Visualization: *Using display()*

The `display()` command:

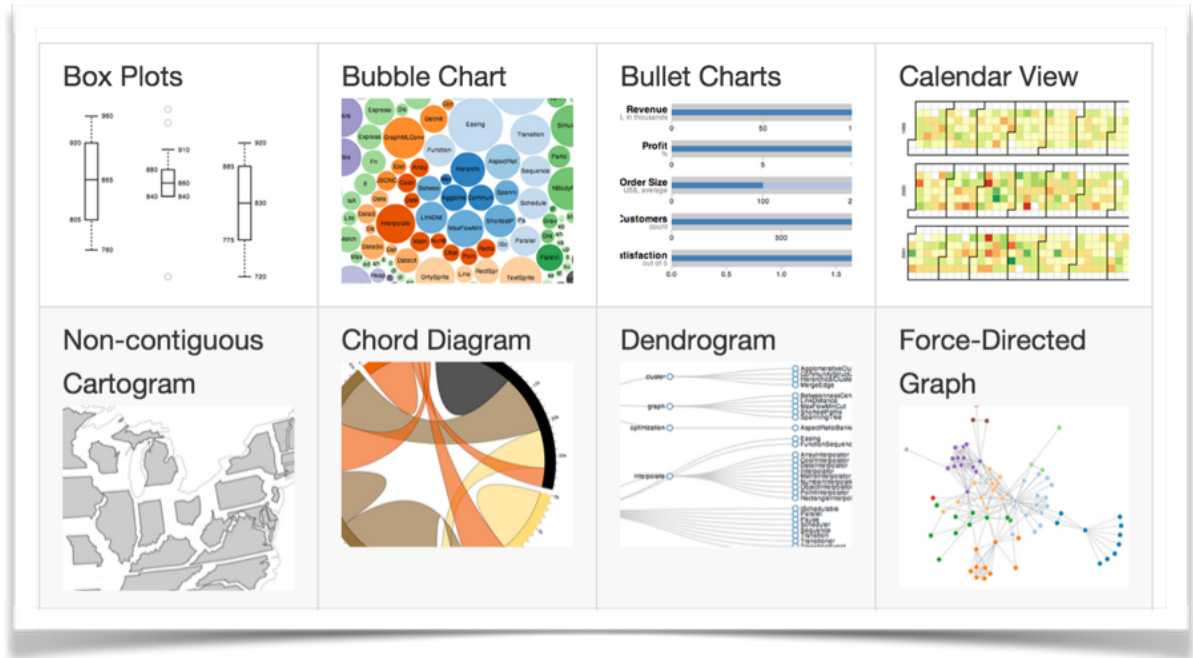
- programmatic access to visualizations
- pass a SchemaRDD to print as an HTML table
- pass a Scala list to print as an HTML table
- call without arguments to display **matplotlib** figures



Visualization: *Using displayHTML()*

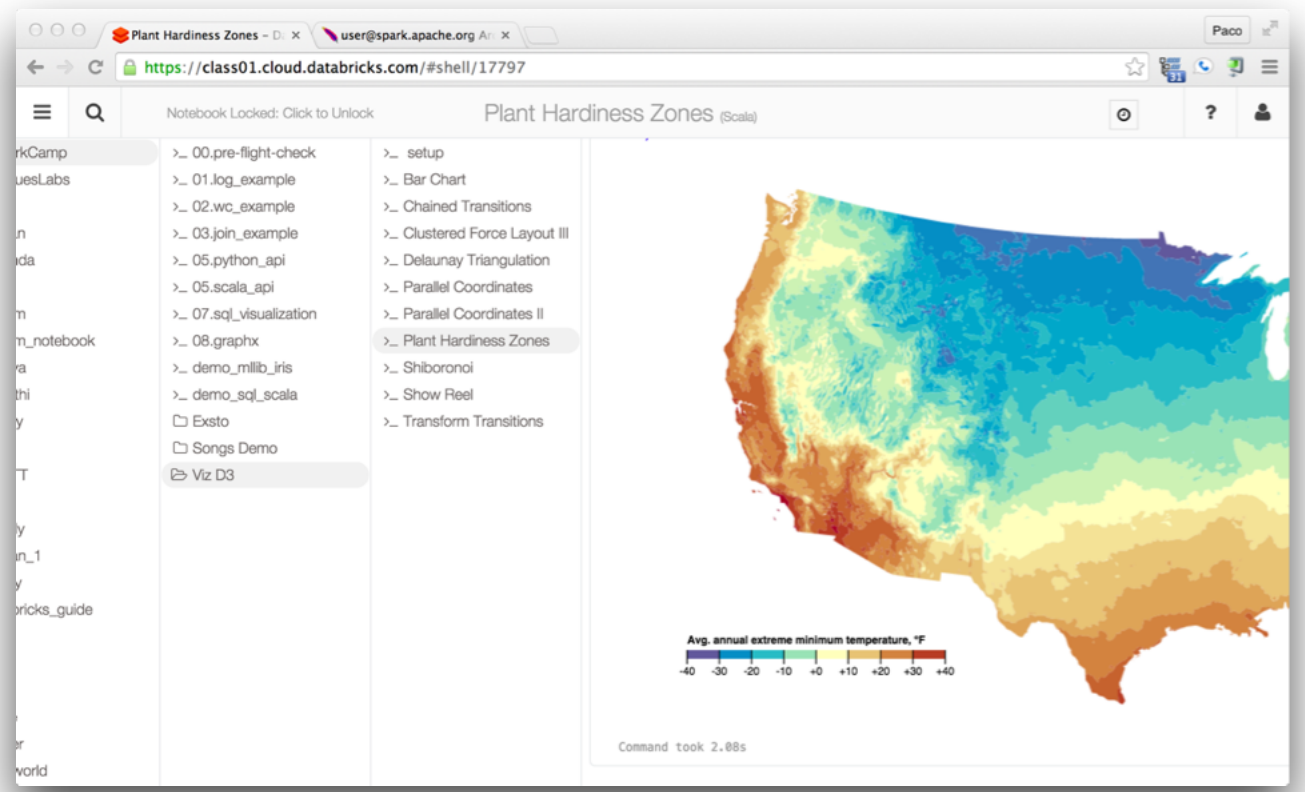
The `displayHTML()` command:

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



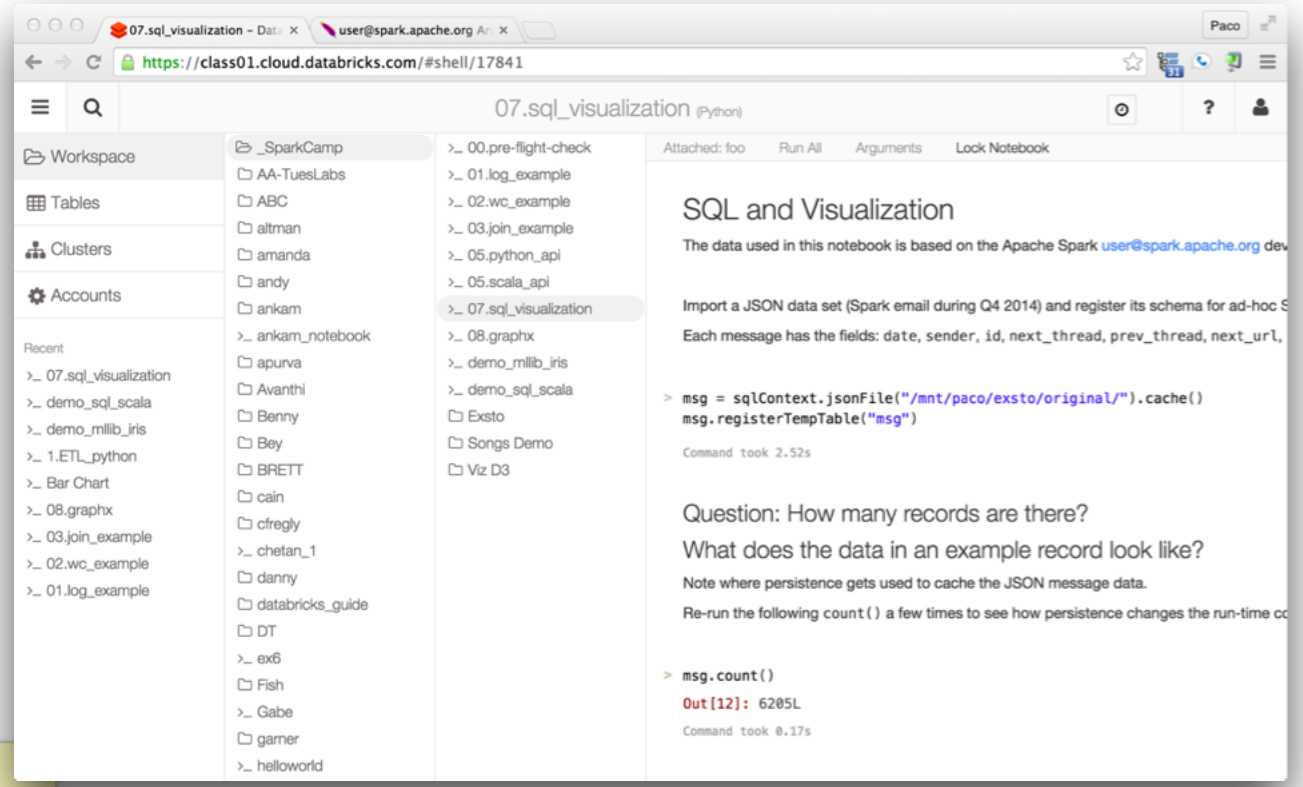
Demo: *D3 Visualization*

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



Coding Exercise: *SQL + Visualization*

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



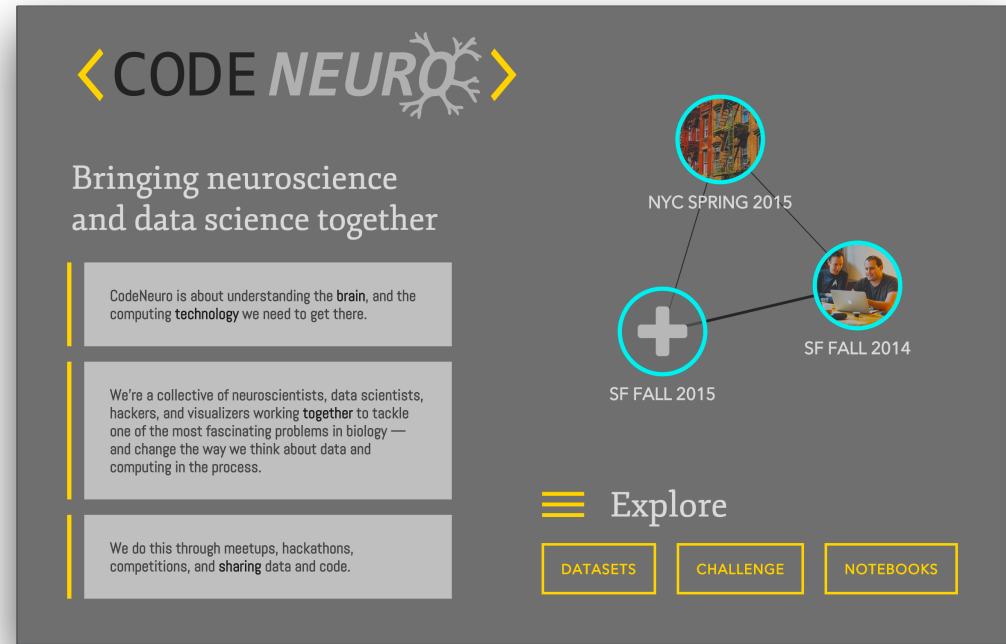

```
%sql
SELECT sender, day, COUNT(id) AS
num_msgs, SUM(chars) AS sum_chars,
```

Great Examples:

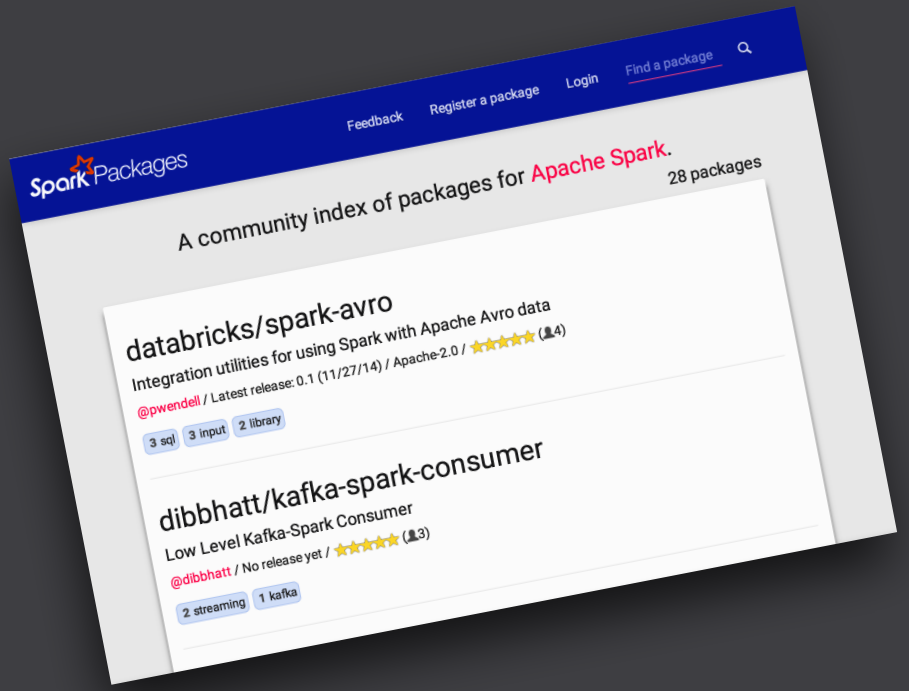
Most definitely check out **CodeNeuro**, both online and the conf/hackathon… and the related **Lightning** project:

Jeremey Freeman, HHMI Janelia Farm
http://notebooks.codeneuro.org/

Matthew Conlen, NY Data Company
http://lightning-viz.org/



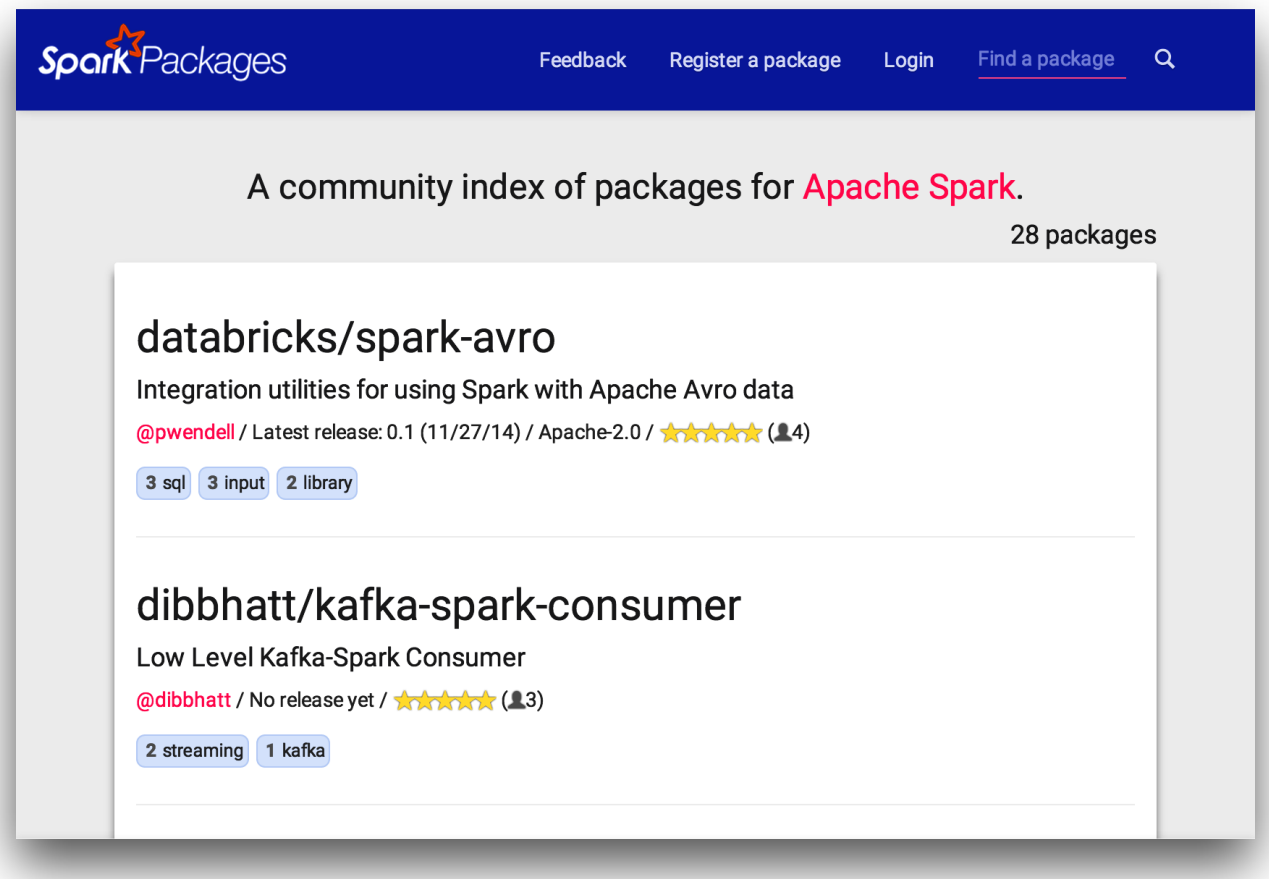
Spark Packages



Resources: *Spark Packages*

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

http://spark-packages.org/



Resources: *Spark Packages*

spark-packages.org

API Extensions

Clojure API

Spark Kernel

Zepplin Notebook

Indexed RDD

Data Sources

Avro

CSV

Elastic Search

MongoDB

Deployment Utilities

Google Compute

Microsoft Azure

Spark Jobserver

```
> ./bin/spark-shell --packages databricks/spark-avro:0.2
```

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

Developer Certification: *Great Prep…*

Find and study the Spark Summit and Strata + HW talks by:

Vida Ha

Exam prep materials are in production at O'Reilly Media by:

Olivier Girardot

community:

spark.apache.org/community.html

events worldwide: goo.gl/2YqJZK

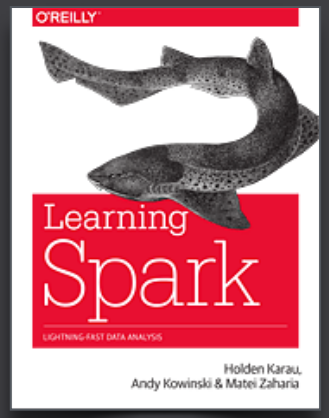
YouTube channel: goo.gl/N5Hx3h

video+preso archives: spark-summit.org

http://spark-summit.org/

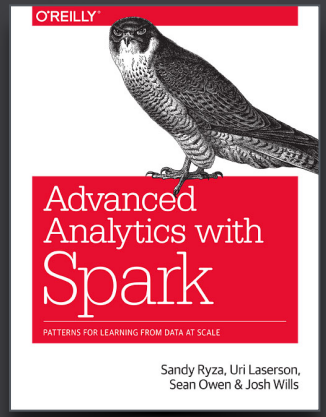
books+videos:

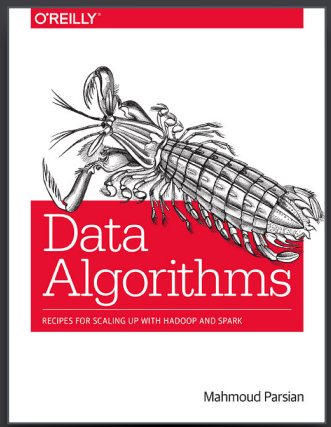
Learning Spark
Holden Karau, Andy Konwinski, Parick Wendell, Matei Zaharia
O'Reilly (2015)
shop.oreilly.com/product/06369200285 12.do



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



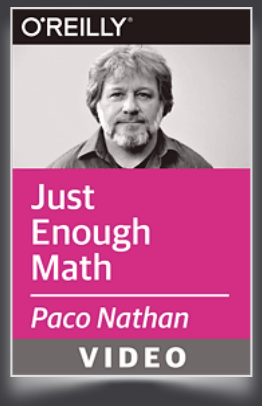


Data Algorithms
Mahmoud Parsian
O'Reilly (2015)
shop.oreilly.com/product/0636920033950.do

presenter:

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

liber118.com/pxn/



Just Enough Math
O'Reilly (2014)

justenoughmath.com
preview: youtu.be/TQ58cWgdCpA