Intro to Apache Spark

http://databricks.com/

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

00: Getting Started

Introduction

installs + intros, while people arrive: 20 min

Intro: *Online Course Materials*

Resources for the course are available at:
databricks.com/spark-training-resources#itas

Download slides+code+data to your laptop:

training.databricks.com/workshop/itas_workshop.pdf

training.databricks.com/workshop/usb.zip

(should have been provided on USB sticks)

Intro: *Success Criteria*

By end of day, participants will be comfortable with the following:

- open a Spark Shell
- develop Spark apps for typical use cases
- tour of the Spark API
- explore data sets loaded from HDFS, etc.
- review of Spark SQL, Spark Streaming, MLlib
- follow-up courses and certification
- developer community resources, events, etc.
- return to workplace and demo use of Spark!

01: Getting Started

Installation

hands-on lab: 20 min

Installation:

Let's get started using Apache Spark,
in just four easy steps…

databricks.com/spark-training-resources#itas

for class, copy from the USB sticks

NB: please do not install/run Spark using:

- *Homebrew* on MacOSX
- *Cygwin* on Windows

Step 1: *Install Java JDK 6/7 on MacOSX or Windows*

oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

- follow the license agreement instructions
- then click the download for your OS
- need JDK instead of JRE (for Maven, etc.)

Step 2: *Download Spark*

we will use Spark 1.1.0

1. copy from the USB sticks
2. double click the archive file to open it
3. connect into the newly created directory

for a fallback: **spark.apache.org/downloads.html**

Step 3: *Run Spark Shell*

we'll run Spark's interactive shell…

within the "spark" directory, run:

```
./bin/spark-shell
```

then from the "scala>" REPL prompt,
let's create some data…

```
val data = 1 to 10000
```

Step 4: *Create an RDD*

create an **RDD** based on that data…

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

then use a filter to select values less than 10…

```
distData.filter(_ < 10).collect()
```

Step 4: *Create an RDD*

create an

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

then use a filter to select values less than 10...

```
d
```

Checkpoint:
what do you get for results?

gist.github.com/ceteri/f2c3486062c9610eac1d#file-01-repl-txt

Installation: *Optional Downloads: Python*

For Python 2.7, check out *Anaconda* by Continuum Analytics for a full-featured platform:

store.continuum.io/cshop/anaconda/

Installation: *Optional Downloads: Maven*

Java builds later also require Maven, which you can download at:

maven.apache.org/download.cgi

02: Getting Started

Spark Deconstructed

lecture: 20 min

Spark Deconstructed:

Let’s spend a few minutes on this Scala thing…

scala-lang.org/

Scala Crash Course

Holden Karau

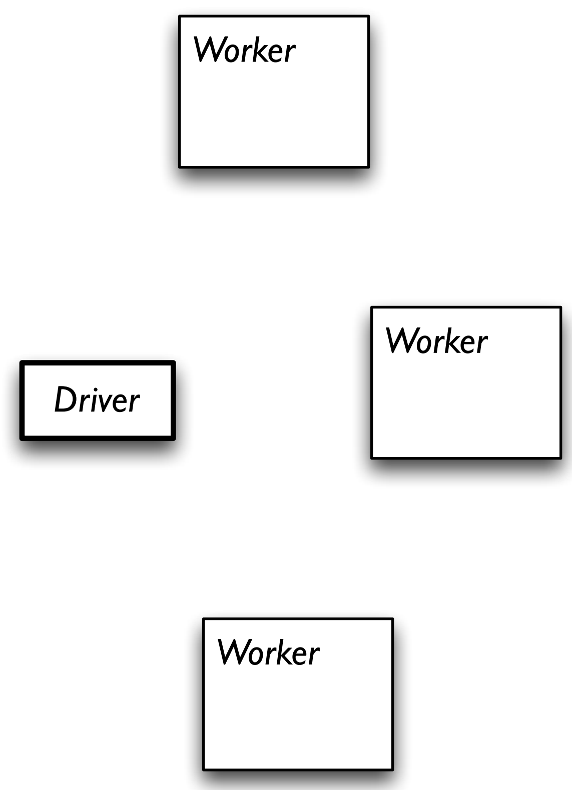
lintool.github.io/SparkTutorial/slides/day1_Scala_crash_course.pdf

Spark Deconstructed: *Log Mining Example*

```
// load error messages from a log into memory
// then interactively search for various patterns
// https://gist.github.com/ceteri/8ae5b9509a08c08a1132

// base RDD
val lines = sc.textFile("hdfs://...")

// transformed RDDs
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
messages.cache()

// action 1
messages.filter(_.contains("mysql")).count()

// action 2
messages.filter(_.contains("php")).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
val lines = sc.textFile("hdfs://...")

// transformed RDDs
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
messages.cache()

// action 1
messages.filter(_.contains("mysql")).count()

// action 2
messages.filter(_.contains("php")).count()
```

discussing the other part

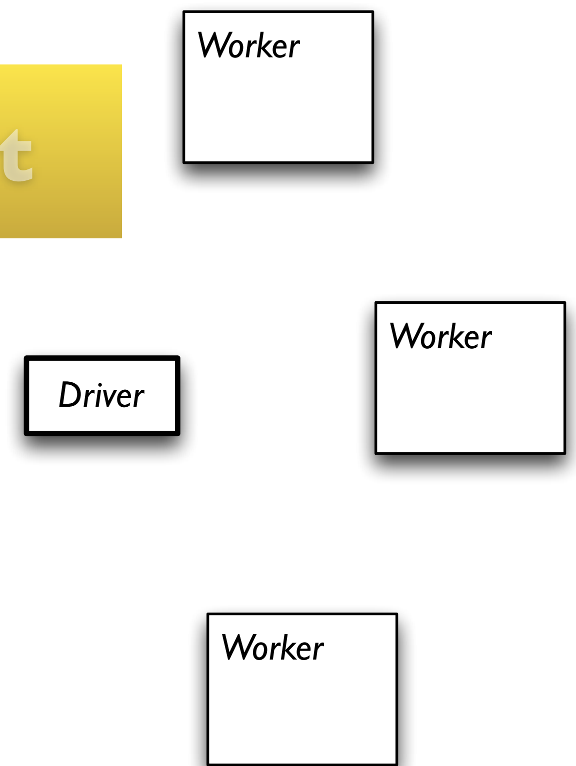
At this point, take a look at the transformed RDD *operator graph*:

```
scala> messages.toDebugString
res5: String =
MappedRDD[4] at map at <console>:16 (3 partitions)
  MappedRDD[3] at map at <console>:16 (3 partitions)
    FilteredRDD[2] at filter at <console>:14 (3 partitions)
      MappedRDD[1] at textFile at <console>:12 (3 partitions)
        HadoopRDD[0] at textFile at <console>:12 (3 partitions)
```

Spark Deconstructed: *Log Mining Example*

```
// base RDD
val lines = sc.textFile("hdfs://...")

// transformed RDDs
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
messages.cache()

// action 1
messages.filter(_.contains("mysql")).count()

// action 2
messages.filter(_.contains("php")).count()
```

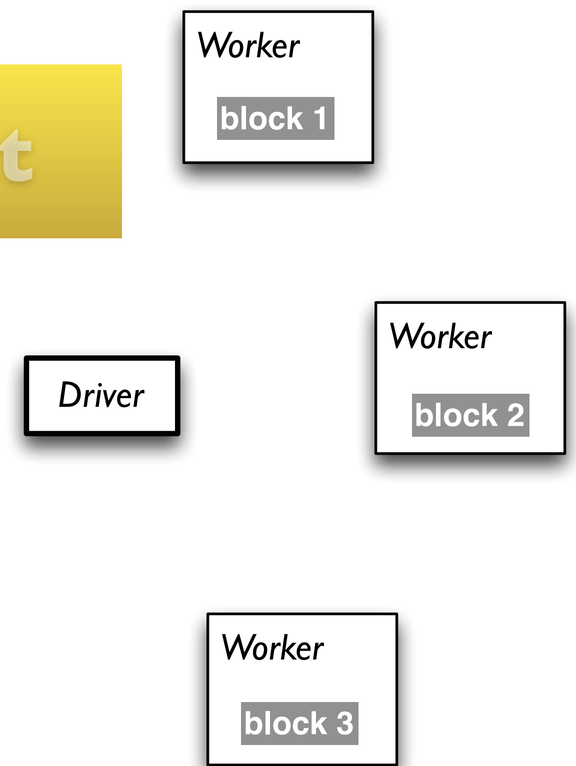
discussing the other part



Spark Deconstructed: *Log Mining Example*

```
// base RDD
val lines = sc.textFile("hdfs://...")

// transformed RDDs
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
messages.cache()

// action 1
messages.filter(_.contains("mysql")).count()

// action 2
messages.filter(_.contains("php")).count()
```

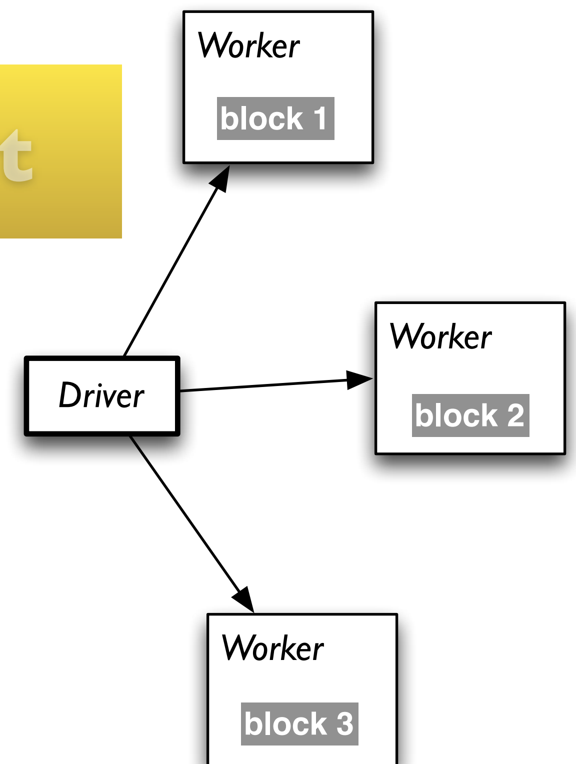
discussing the other part



Spark Deconstructed: *Log Mining Example*

```
// base RDD
val lines = sc.textFile("hdfs://...")

// transformed RDDs
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
messages.cache()

// action 1
messages.filter(_.contains("mysql")).count()

// action 2
messages.filter(_.contains("php")).count()
```

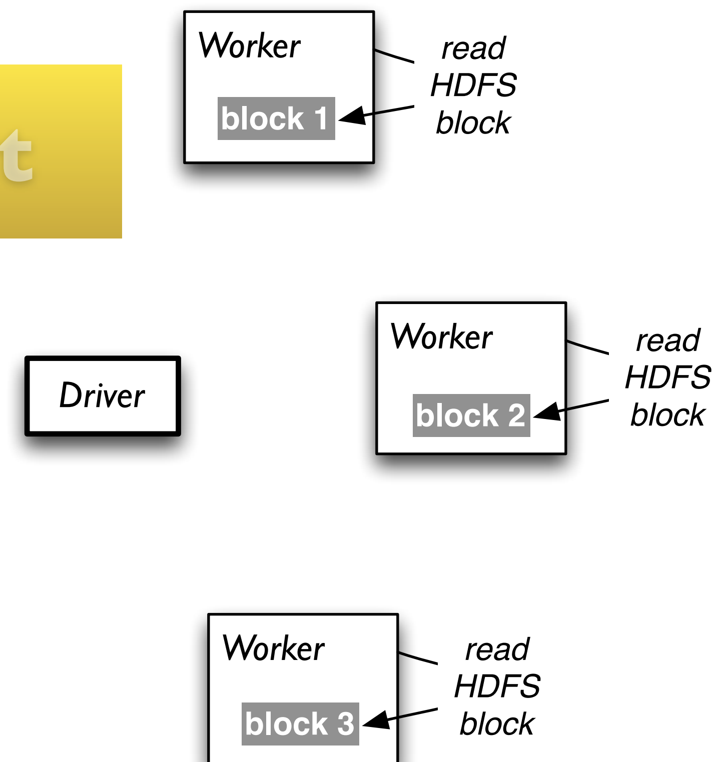
discussing the other part



Spark Deconstructed: *Log Mining Example*

```
// base RDD
val lines = sc.textFile("hdfs://...")

// transformed RDDs
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
messages.cache()

// action 1
messages.filter(_.contains("mysql")).count()

// action 2
messages.filter(_.contains("php")).count()
```

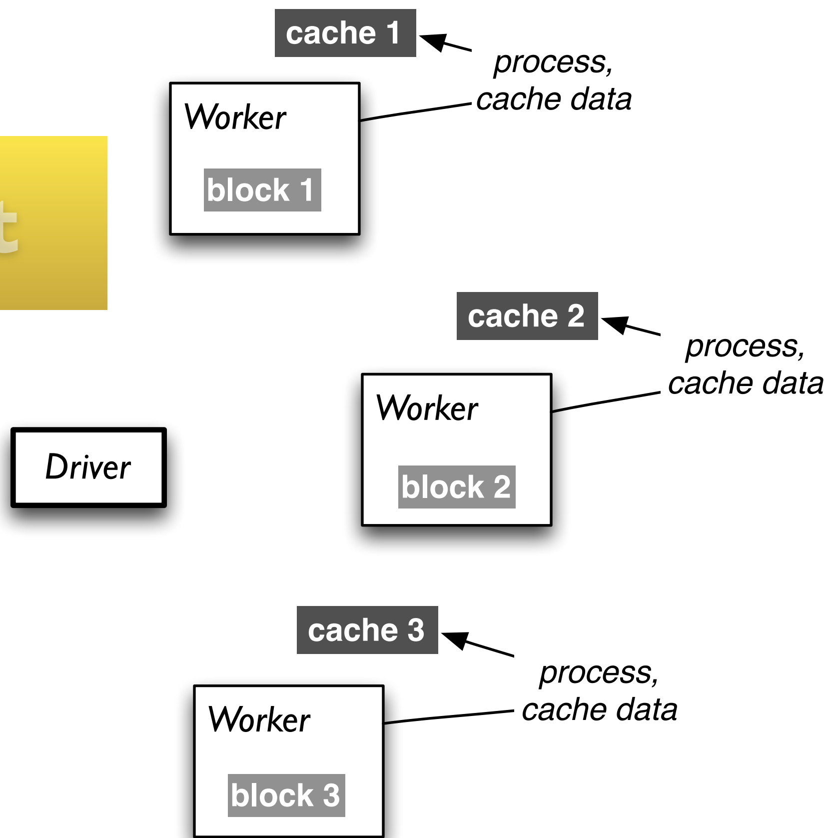
discussing the other part



Spark Deconstructed: *Log Mining Example*

```
// base RDD
val lines = sc.textFile("hdfs://...")

// transformed RDDs
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
messages.cache()

// action 1
messages.filter(_.contains("mysql")).count()

// action 2
messages.filter(_.contains("php")).count()
```

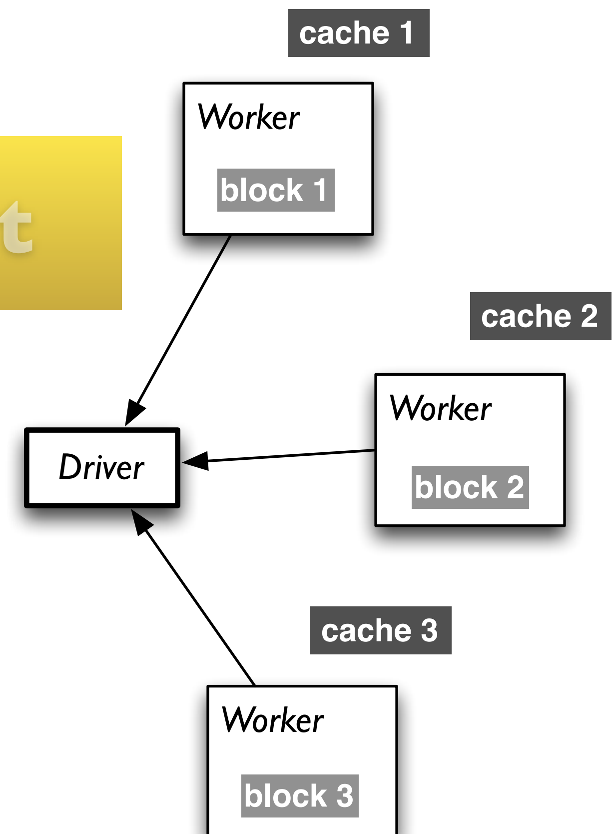
discussing the other part



Spark Deconstructed: *Log Mining Example*

```
// base RDD
val lines = sc.textFile("hdfs://...")

// transformed RDDs
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
messages.cache()

// action 1
messages.filter(_.contains("mysql")).count()

// action 2
messages.filter(_.contains("php")).count()
```

discussing the other part

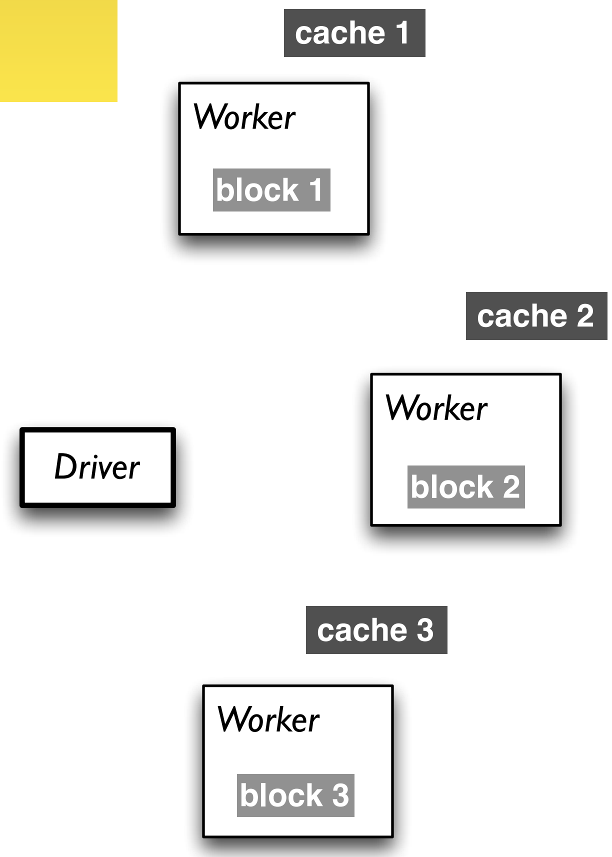


Spark Deconstructed: *Log Mining Example*

```
// base RDD
val lines = sc.textFile("hdfs://...")

// transformed RDDs
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
messages.cache()

// action 1
messages.filter(_.contains("mysql")).count()
```

discussing the other part

```
// action 2
messages.filter(_.contains("php")).count()
```

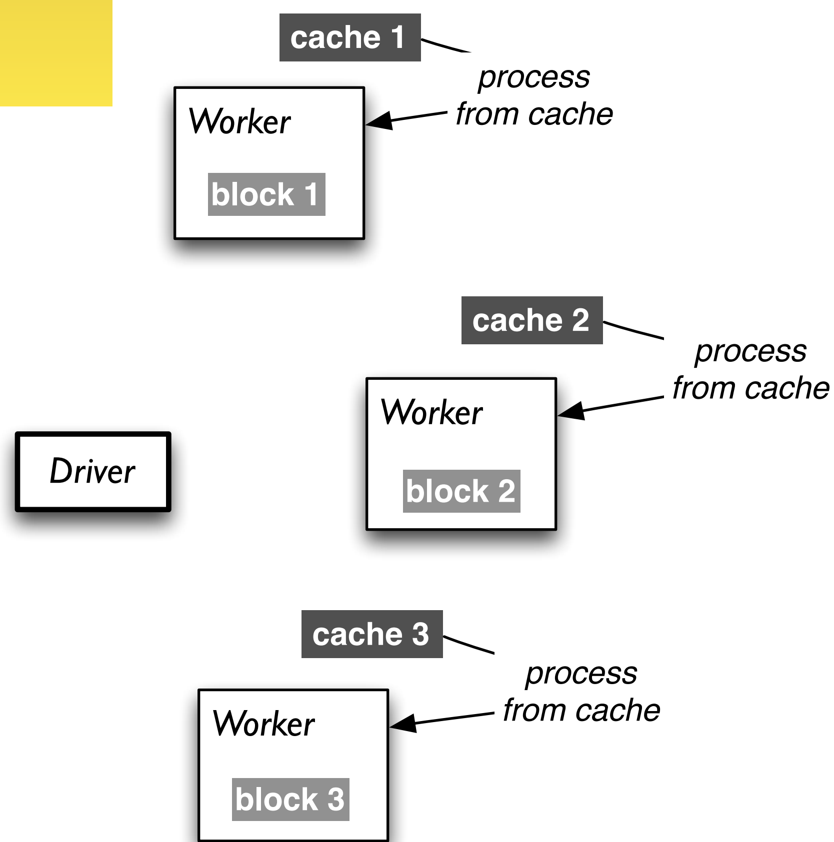


Spark Deconstructed: *Log Mining Example*

```
// base RDD
val lines = sc.textFile("hdfs://...")

// transformed RDDs
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
messages.cache()

// action 1
messages.filter(_.contains("mysql")).count()
```

discussing the other part

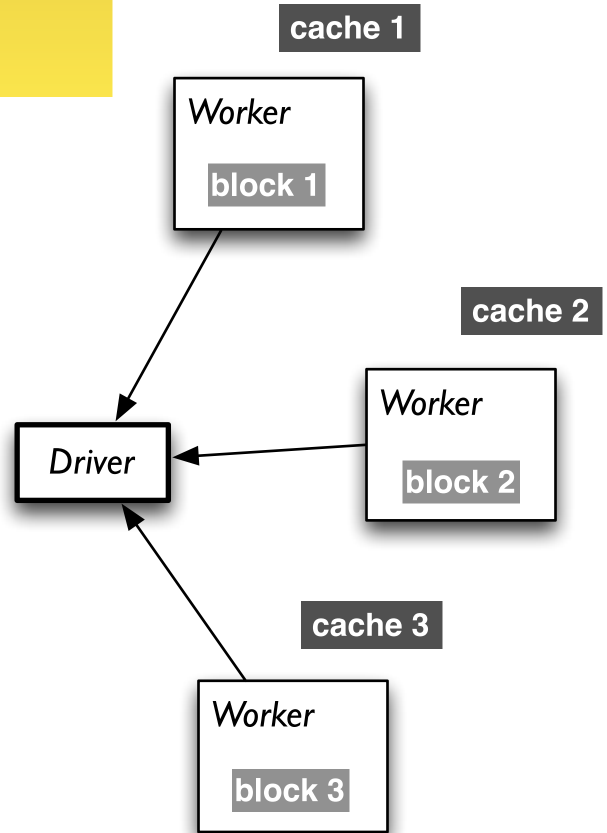
```
// action 2
messages.filter(_.contains("php")).count()
```



Spark Deconstructed: *Log Mining Example*

discussing the other part

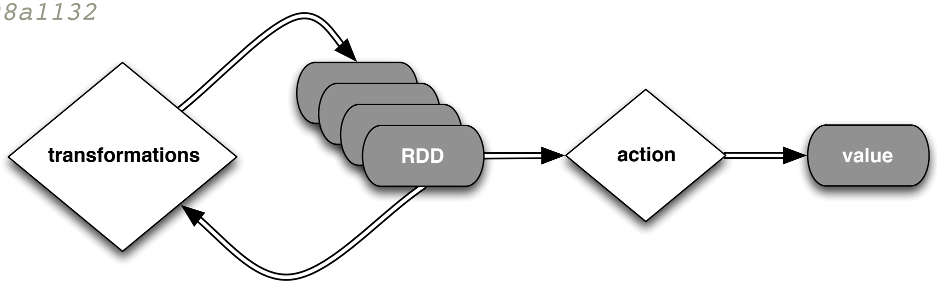
```
// base RDD
val lines = sc.textFile("hdfs://...")

// transformed RDDs
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
messages.cache()

// action 1
messages.filter(_.contains("mysql")).count()

// action 2
messages.filter(_.contains("php")).count()
```



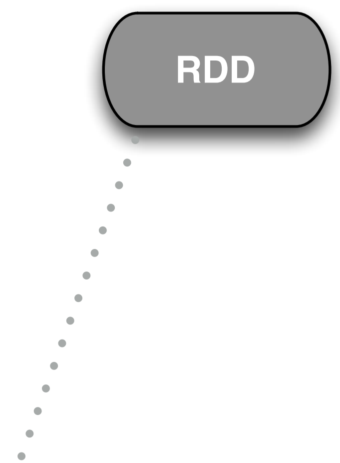
Spark Deconstructed:

Looking at the RDD transformations and actions from another perspective…

```
// load error messages from a log into memory
// then interactively search for various patterns
// https://gist.github.com/ceteri/8ae5b9509a08c08a1132

// base RDD
val lines = sc.textFile("hdfs://...")

// transformed RDDs
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
messages.cache()

// action 1
messages.filter(_.contains("mysql")).count()

// action 2
messages.filter(_.contains("php")).count()
```

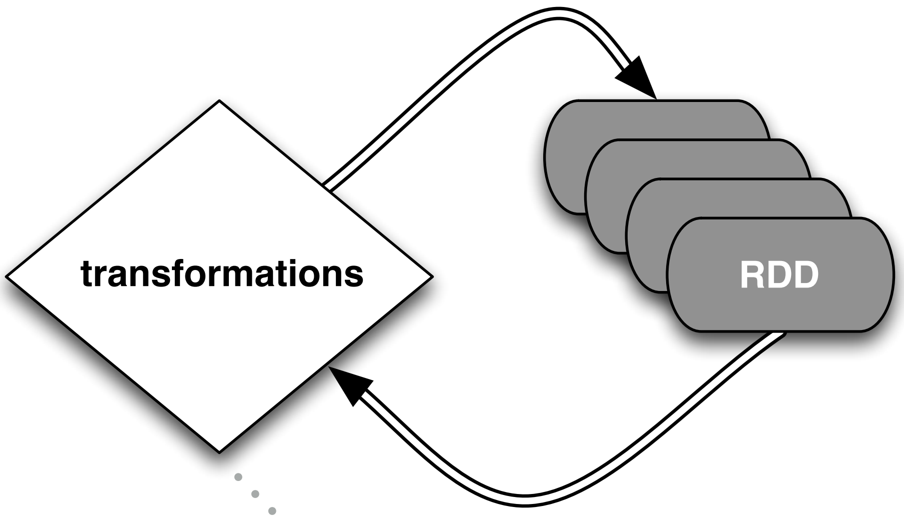


Spark Deconstructed:



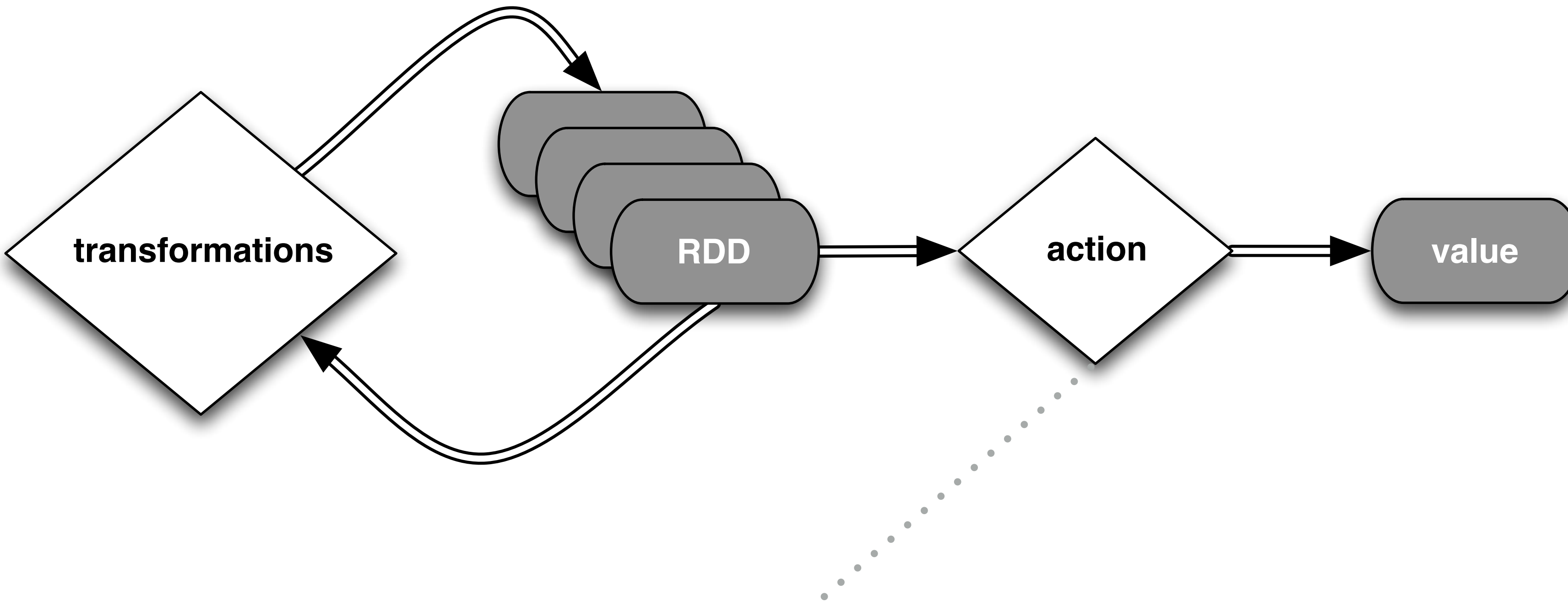

```
// base RDD
val lines = sc.textFile("hdfs://...")
```

Spark Deconstructed:




```
// transformed RDDs
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
messages.cache()
```

Spark Deconstructed:



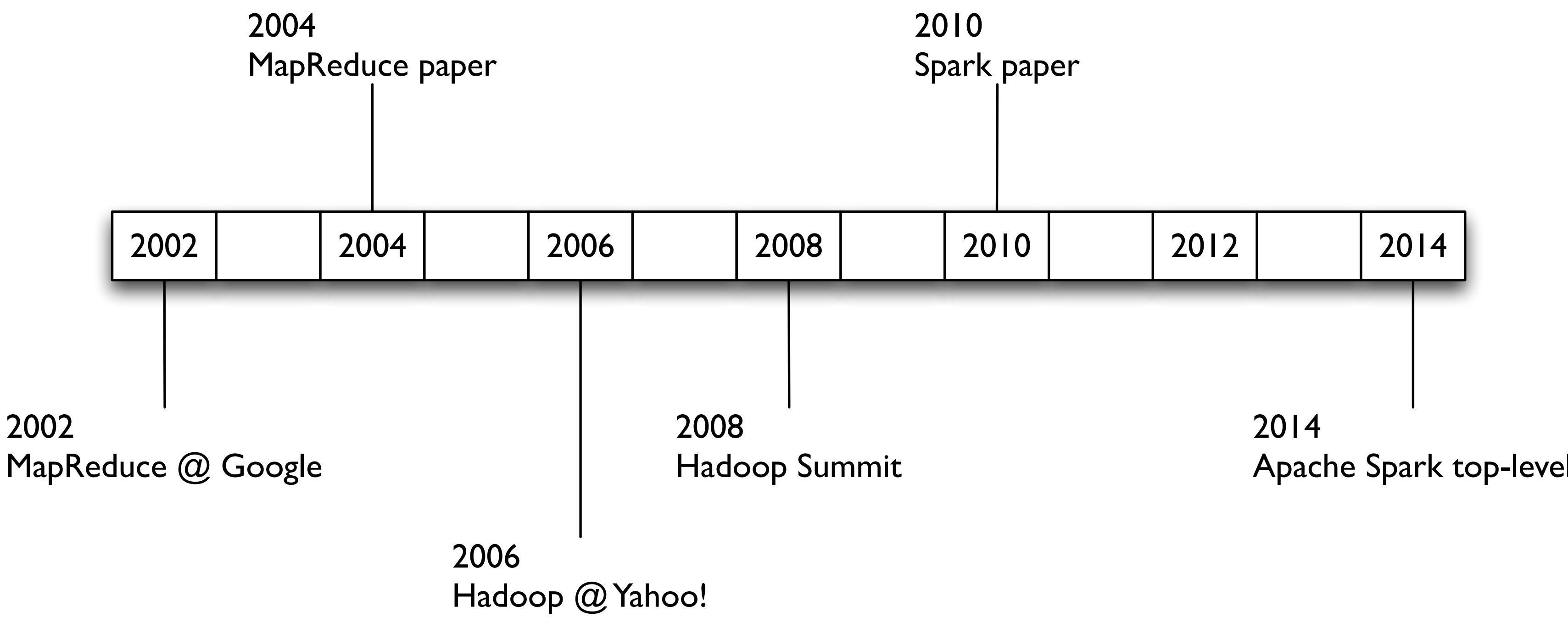

```
// action 1
messages.filter(_.contains("mysql")).count()
```

03: Getting Started

A Brief History

lecture: 35 min

A Brief History:



A Brief History: *MapReduce*

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

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

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

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

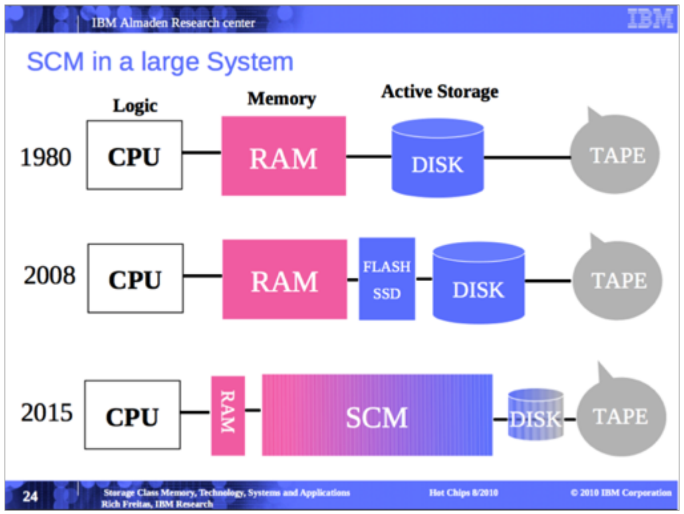
circa 2009 – **Amazon AWS**
Elastic MapReduce
Hadoop modified for EC2/S3, plus support for Hive, Pig, Cascading, etc.
aws.amazon.com/elasticmapreduce/

A Brief History: *MapReduce*

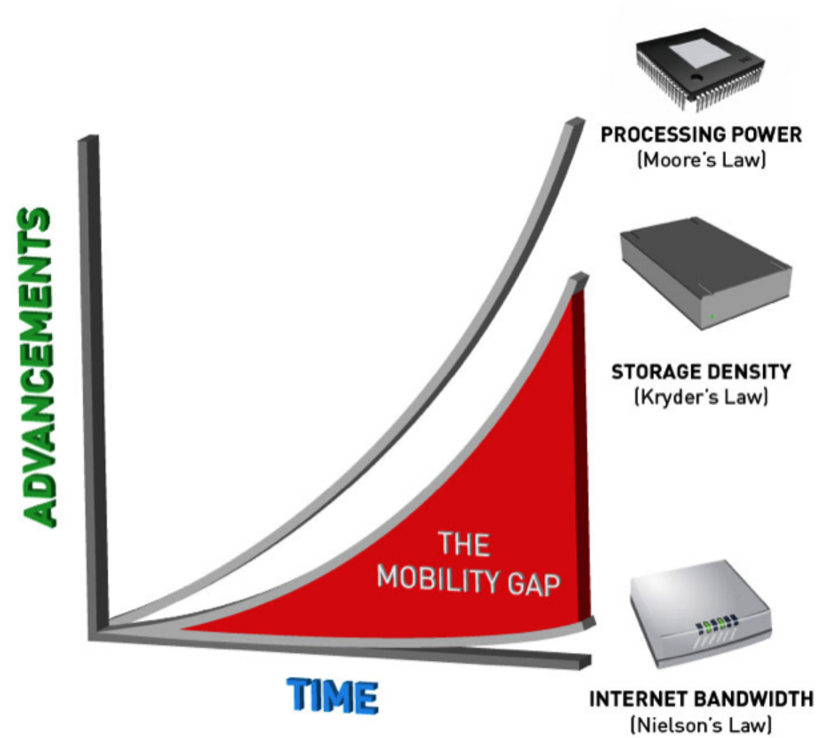
Open Discussion:

Enumerate several changes in data center technologies since 2002…

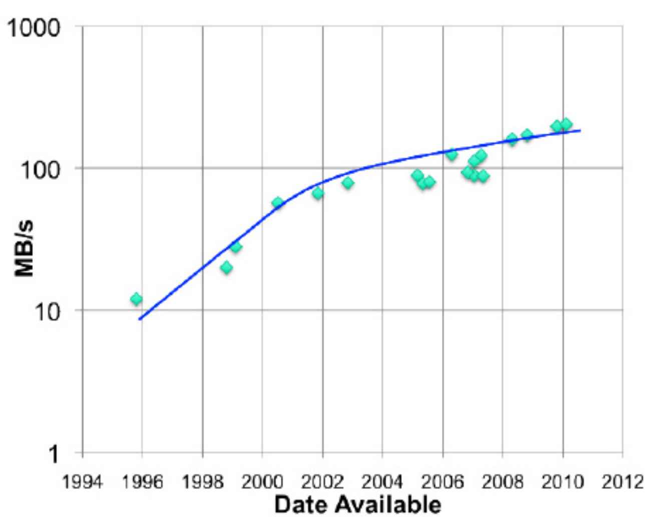
A Brief History: *MapReduce*



Rich Freitas, **IBM Research**



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



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

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

A Brief History: *MapReduce*

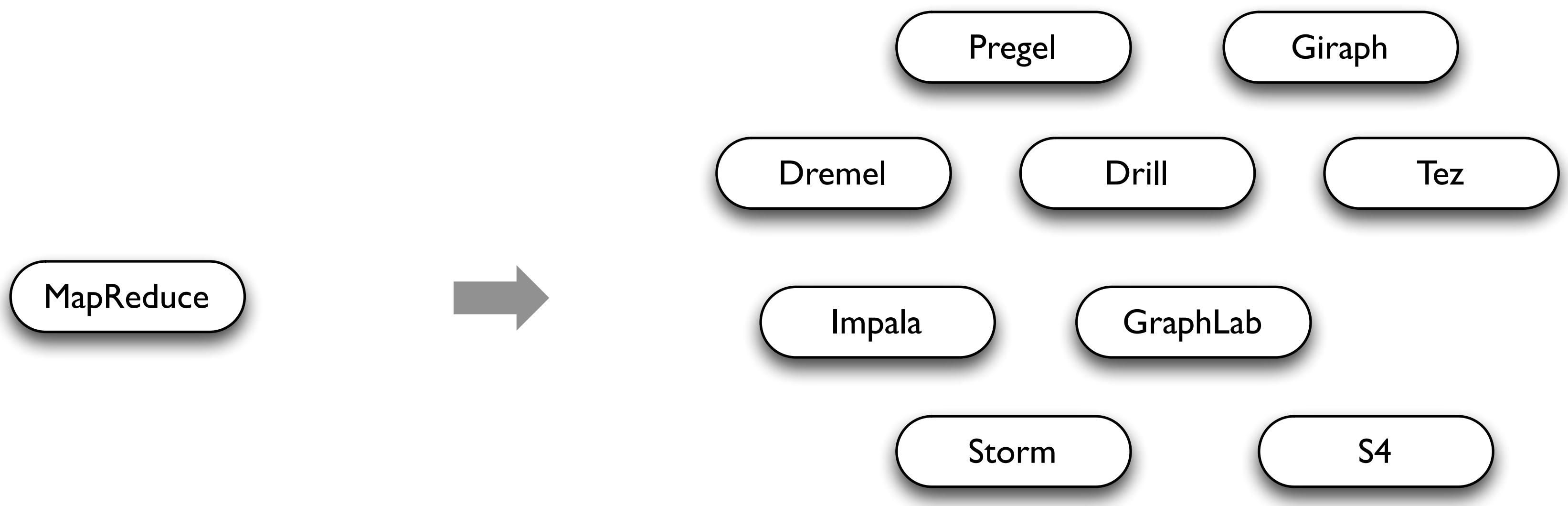
MapReduce use cases showed two major limitations:

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

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

Therefore, people built *specialized systems* as workarounds…

A Brief History: *MapReduce*



General Batch Processing

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

The State of Spark, and Where We're Going Next
Matei Zaharia
Spark Summit (2013)
youtu.be/nU6vO2EJAb4

A Brief History: *Spark*

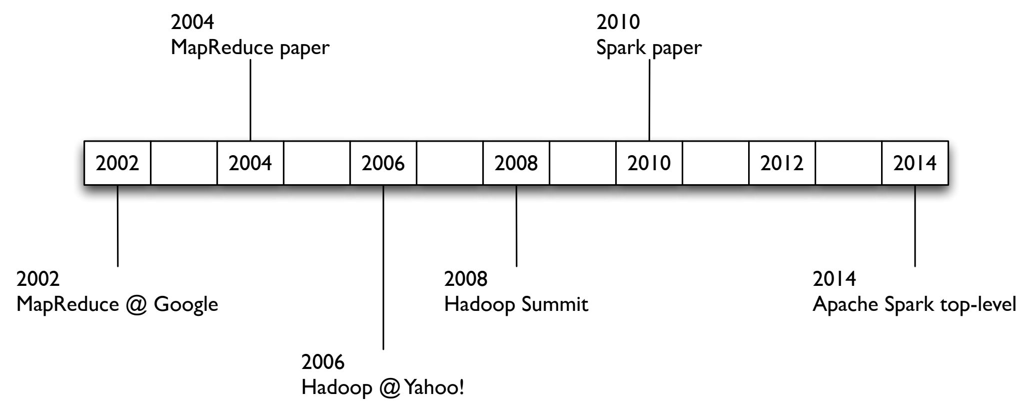
Developed in 2009 at UC Berkeley AMPLab, then open sourced in 2010, Spark has since become one of the largest OSS communities in big data, with over 200 contributors in 50+ organizations

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

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

spark.apache.org

A Brief History: *Spark*



Spark: Cluster Computing with Working Sets
Matei Zaharia, Mosharaf Chowdhury,
Michael J. Franklin, Scott Shenker, Ion Stoica
USENIX HotCloud (2010)
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 J. Franklin, Scott Shenker, Ion Stoica
NSDI (2012)
usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf

A Brief History: *Spark*

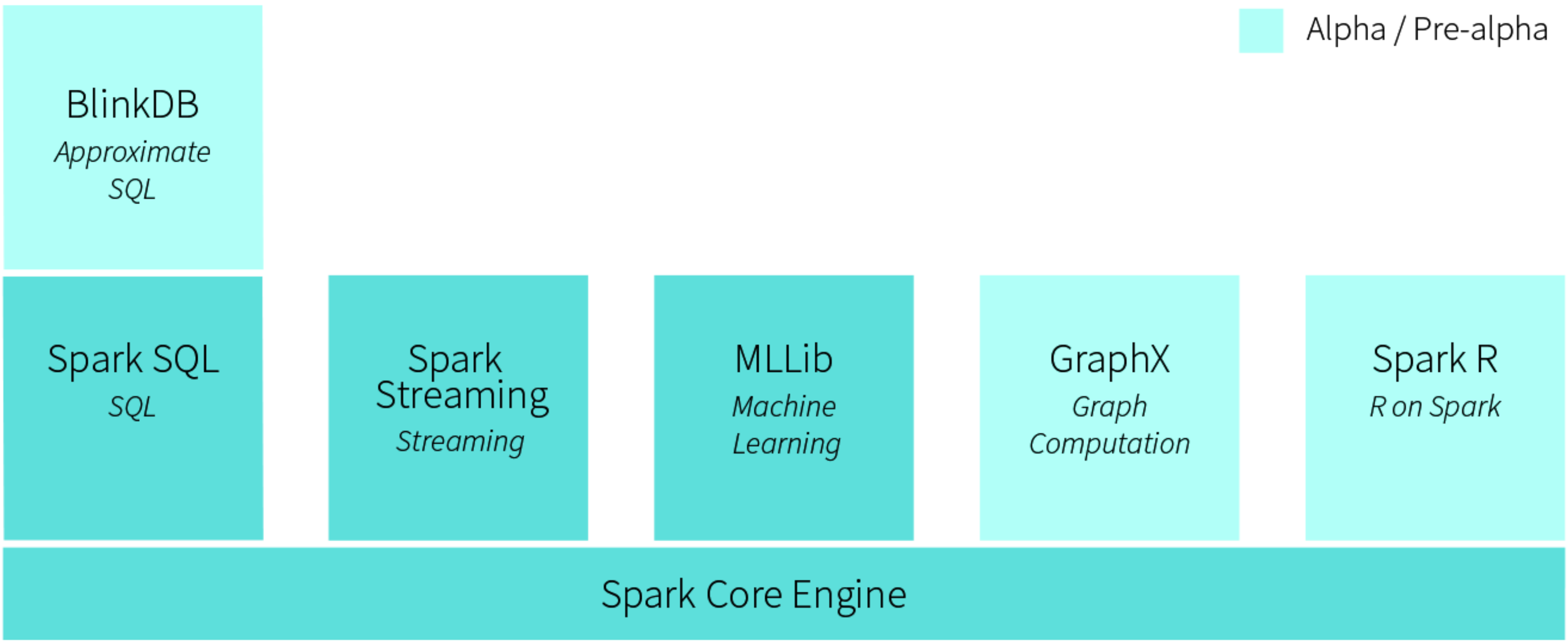
Unlike the various specialized systems, Spark's goal was to *generalize* MapReduce to support new apps within same engine

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

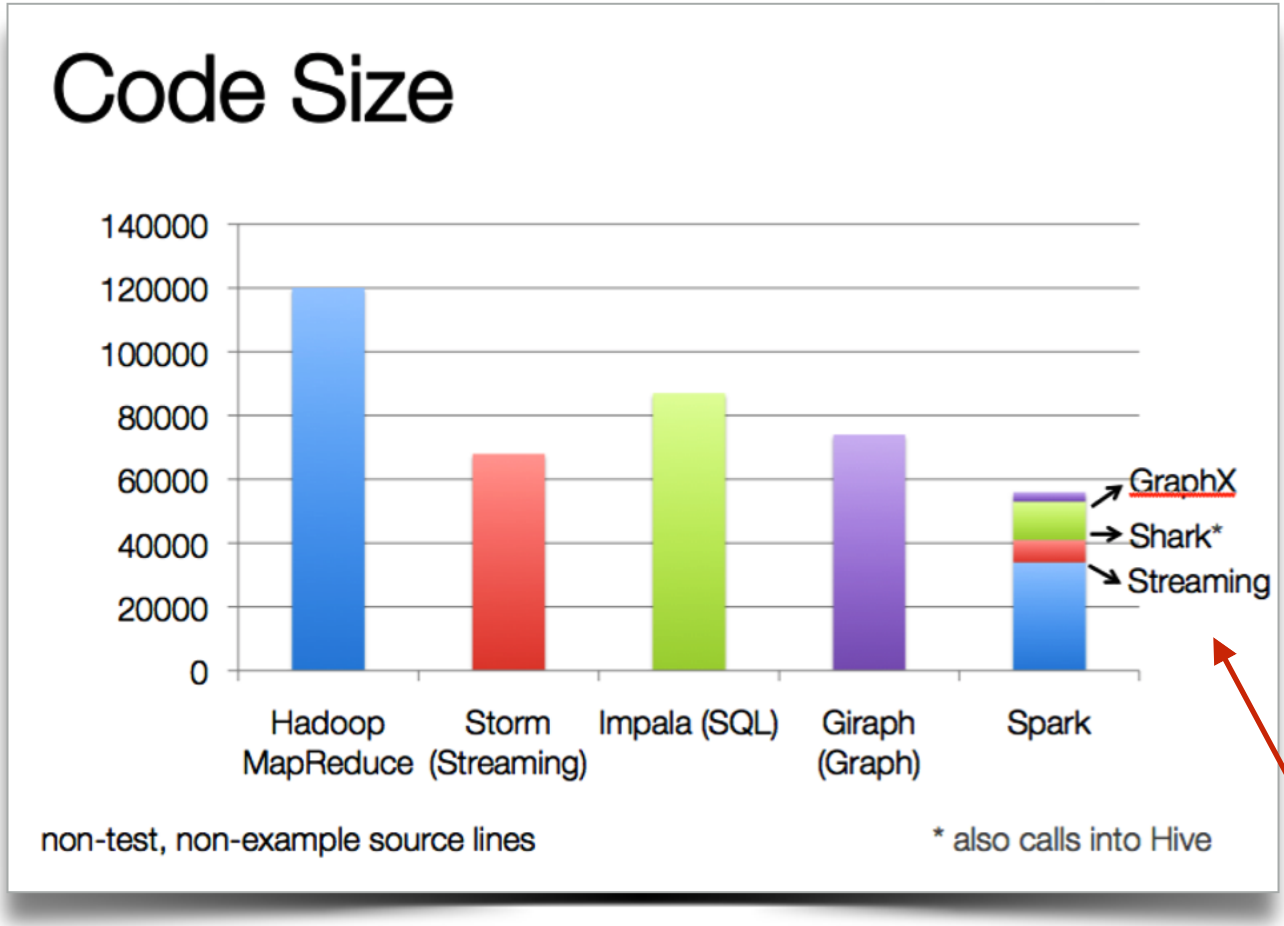
- *fast data sharing*
- *general DAGs*

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

A Brief History: *Spark*



A Brief History: *Spark*



used as libs, instead of specialized systems

A Brief History: *Spark*

Some key points about Spark:

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

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

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

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

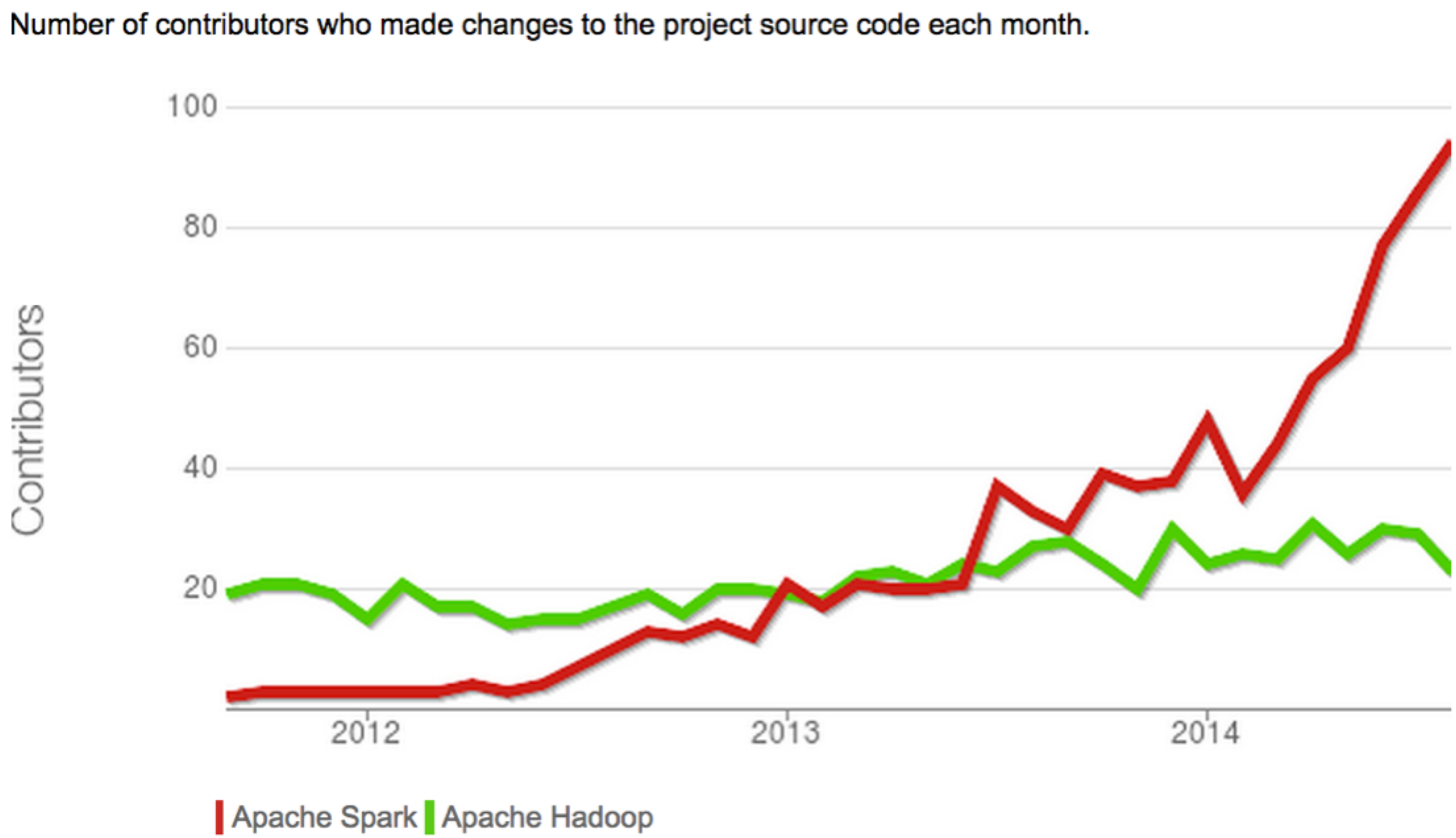
databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

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

TL;DR: *Sustained Exponential Growth*

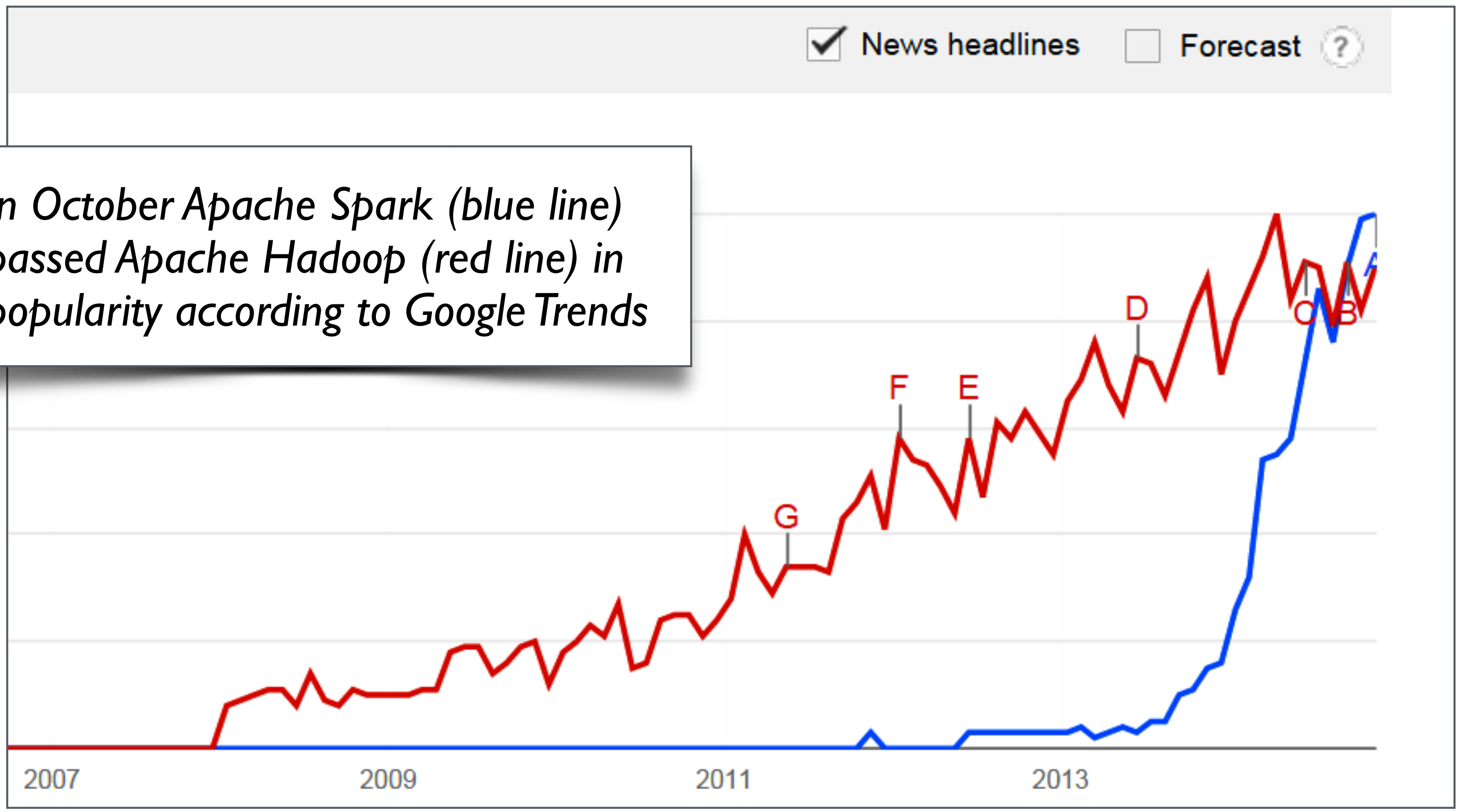
Spark is one of the most active Apache projects

ohloh.net/orgs/apache



TL;DR: *Spark Just Passed Hadoop in Popularity on Web*

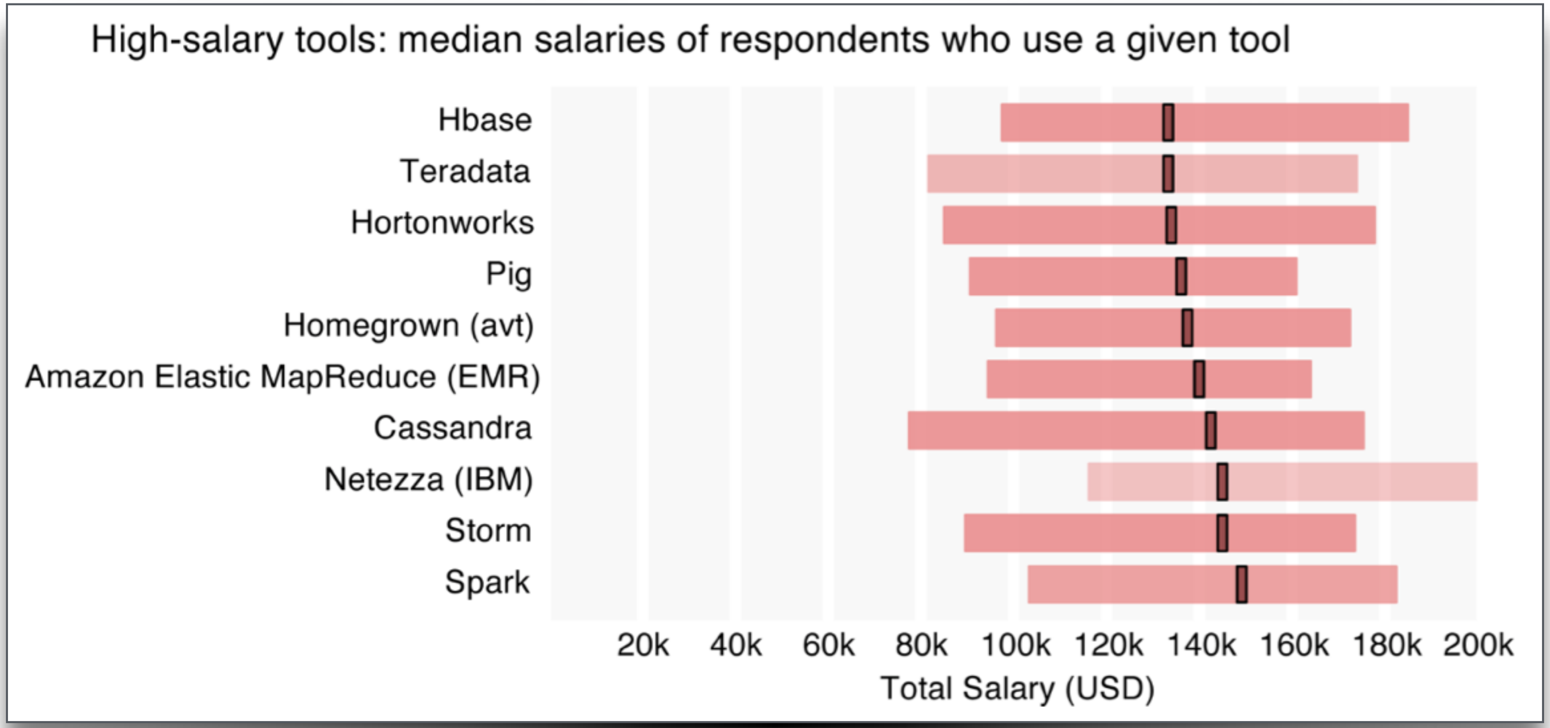
datanami.com/2014/11/21/spark-just-passed-hadoop-popularity-web-heres/



In October Apache Spark (blue line) passed Apache Hadoop (red line) in popularity according to Google Trends

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

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



04: Getting Started

Simple Spark Apps

lab: 20 min

Simple Spark Apps: *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));
```

Simple Spark Apps: *WordCount*

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

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

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

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

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

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

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

WordCount in 3 lines of Spark

WordCount in 50+ lines of Java MR

Simple Spark Apps: *WordCount*

Scala:

```
val f = sc.textFile("README.md")
val wc = f.flatMap(l => l.split(" ")).map(word => (word, 1)).reduceByKey(_ + _)
wc.saveAsTextFile("wc_out")
```

Python:

```
from operator import add
f = sc.textFile("README.md")
wc = f.flatMap(lambda x: x.split(' ')).map(lambda x: (x, 1)).reduceByKey(add)
wc.saveAsTextFile("wc_out")
```

Simple Spark Apps: *WordCount*

Scala:

```
val f = sc.textFile(
val wc
wc.saveAsTextFile(
```

Python

```
from operator
f = sc
wc = f
wc.saveAsTextFile(
```

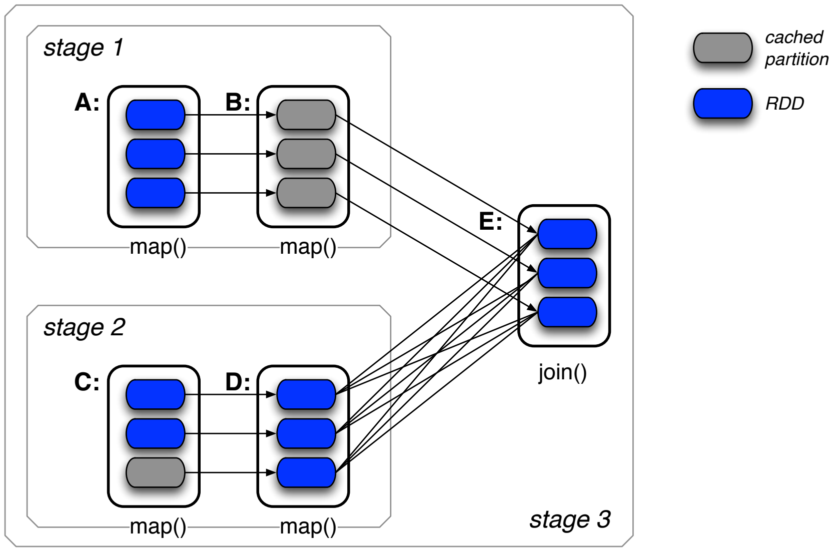
Checkpoint:
how many "Spark" keywords?

Simple Spark Apps: *Source Code*

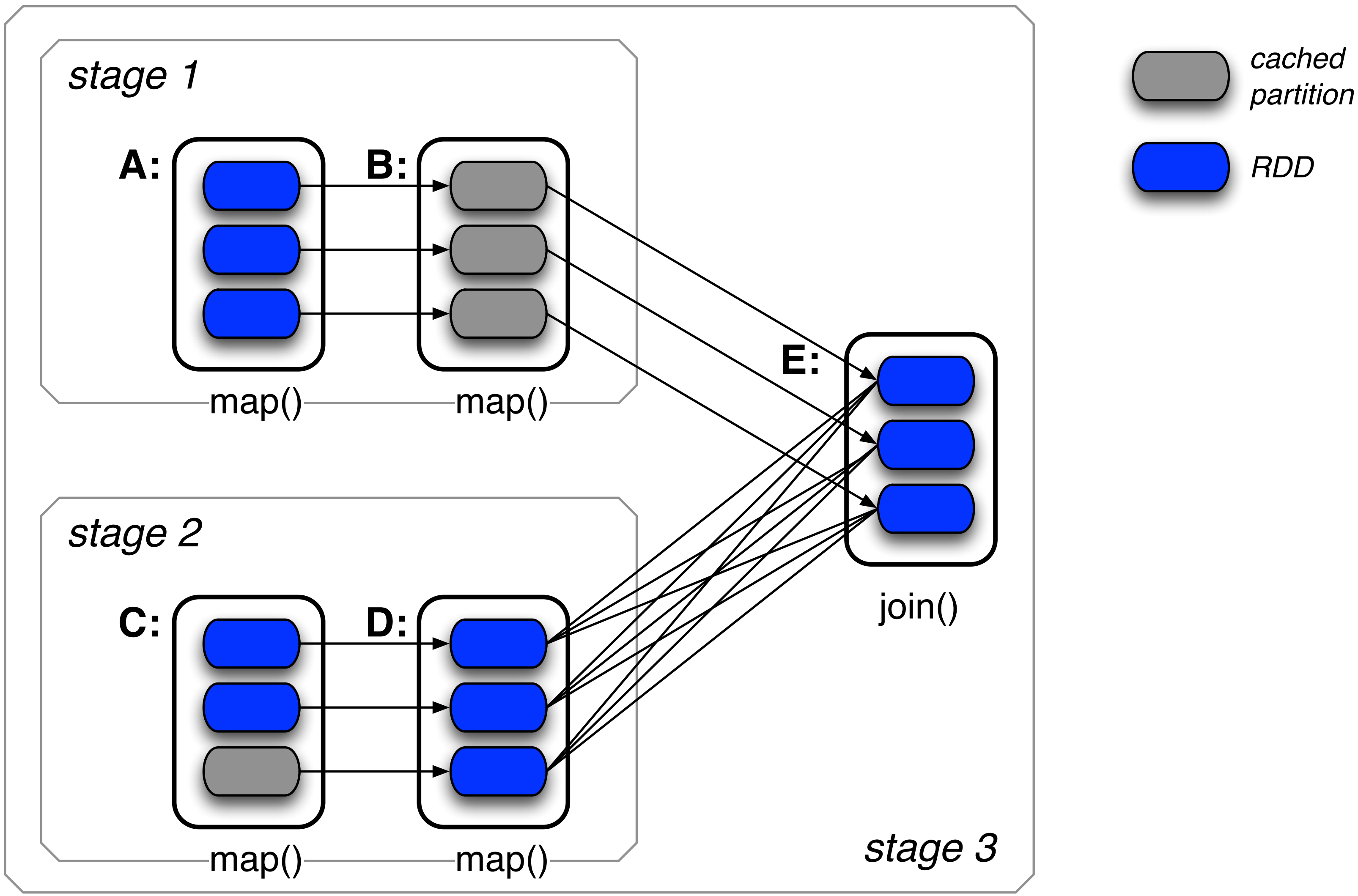
```
val format = new java.text.SimpleDateFormat("yyyy-MM-dd")

case class Register (d: java.util.Date, uuid: String, cust_id: String, lat: Float,
lng: Float)

case class Click (d: java.util.Date, uuid: String, landing_page: Int)

val reg = sc.textFile("reg.tsv").map(_.split("\t")).map(
 r => (r(1), Register(format.parse(r(0)), r(1), r(2), r(3).toFloat, r(4).toFloat))
)

val clk = sc.textFile("clk.tsv").map(_.split("\t")).map(
 c => (c(1), Click(format.parse(c(0)), c(1), c(2).trim.toInt))
)

reg.join(clk).collect()
```

Simple Spark Apps: *Operator Graph*

```
scala> reg.join(clk).toDebugString
res5: String =
FlatMappedValuesRDD[46] at join at <console>:23 (1 partitions)
  MappedValuesRDD[45] at join at <console>:23 (1 partitions)
    CoGroupedRDD[44] at join at <console>:23 (1 partitions)
      MappedRDD[36] at map at <console>:16 (1 partitions)
        MappedRDD[35] at map at <console>:16 (1 partitions)
          MappedRDD[34] at textFile at <console>:16 (1 partitions)
            HadoopRDD[33] at textFile at <console>:16 (1 partitions)
      MappedRDD[40] at map at <console>:16 (1 partitions)
        MappedRDD[39] at map at <console>:16 (1 partitions)
          MappedRDD[38] at textFile at <console>:16 (1 partitions)
            HadoopRDD[37] at textFile at <console>:16 (1 partitions)
```



Simple Spark Apps: *Operator Graph*



Simple Spark Apps: *Assignment*

Using the `README.md` and `CONTRIBUTING.md` files in the Spark directory:

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 (word, count)
3. join the two RDDs

Simple Spark Apps: *Assignment*

Using the
in the Spark directory:

1. create RDDs to filter each line for the key

2. per
resu

3. join the two RDDs

Checkpoint:
how many "Spark" keywords?

(break)

break: 15 min

05: Intro Spark Apps

Spark Essentials

lecture/lab: 45 min

Spark Essentials:

Intro apps, showing examples in both Scala and Python…

Let's start with the basic concepts in:

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

using, respectively:

```
./bin/spark-shell
```

```
./bin/pyspark
```

alternatively, with IPython Notebook:

```
IPYTHON_OPTS="notebook --pylab inline" ./bin/pyspark
```

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:

```
scala> sc
res: spark.SparkContext = spark.SparkContext@470d1f30
```

Python:

```
>>> sc
<pyspark.context.SparkContext object at 0x7f7570783350>
```

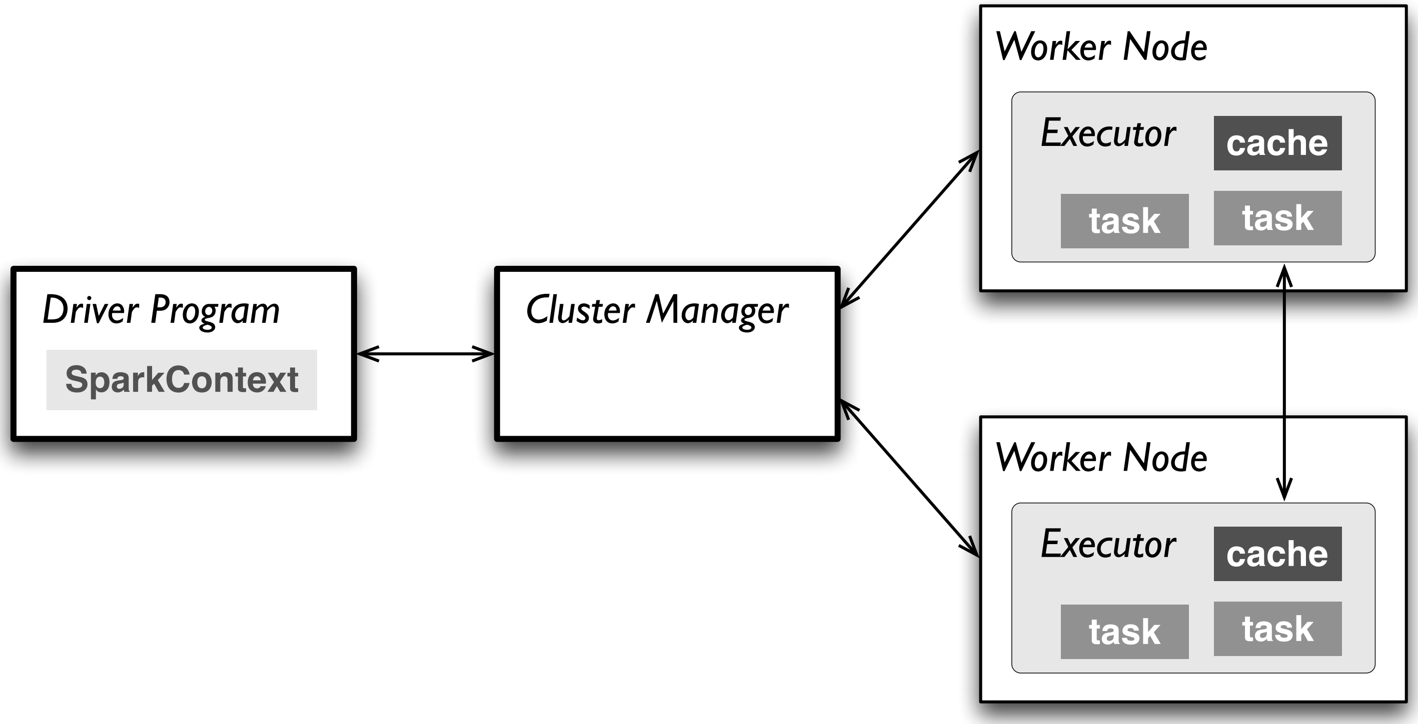
Spark Essentials: *Master*

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)

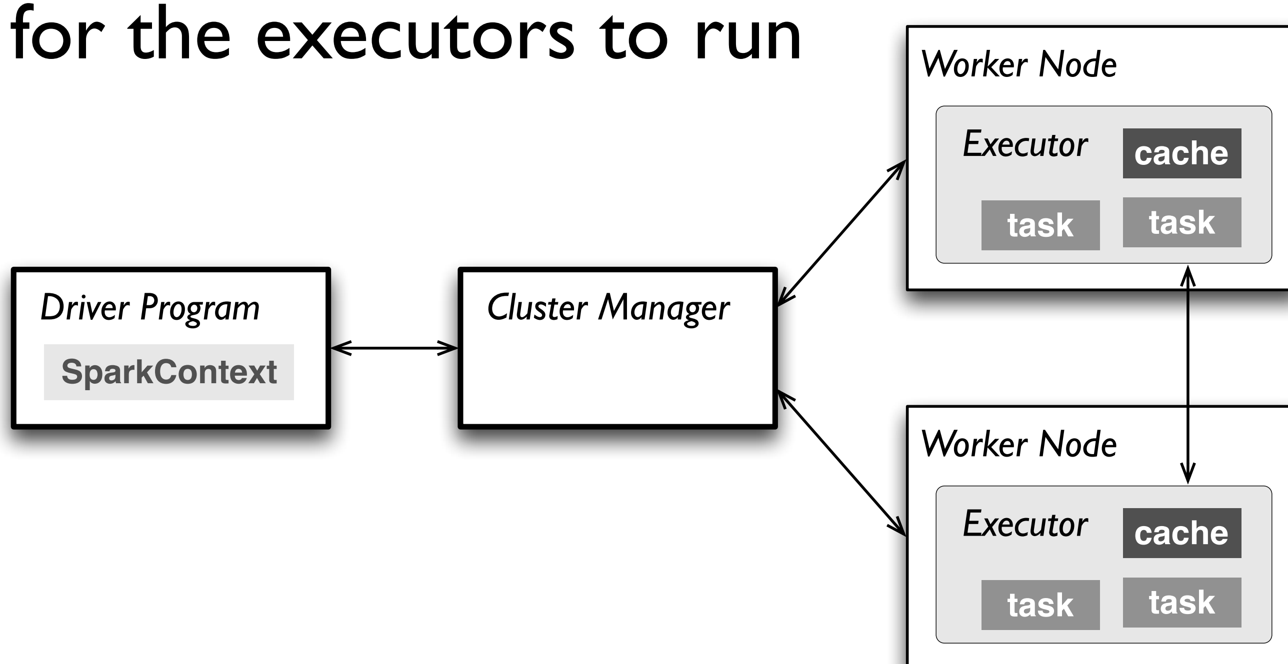
Spark Essentials: *Master*

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



Spark Essentials: *Clusters*

1. master 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



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:

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

scala> val distData = sc.parallelize(data)
distData: spark.RDD[Int] = spark.ParallelCollection@10d13e3e
```

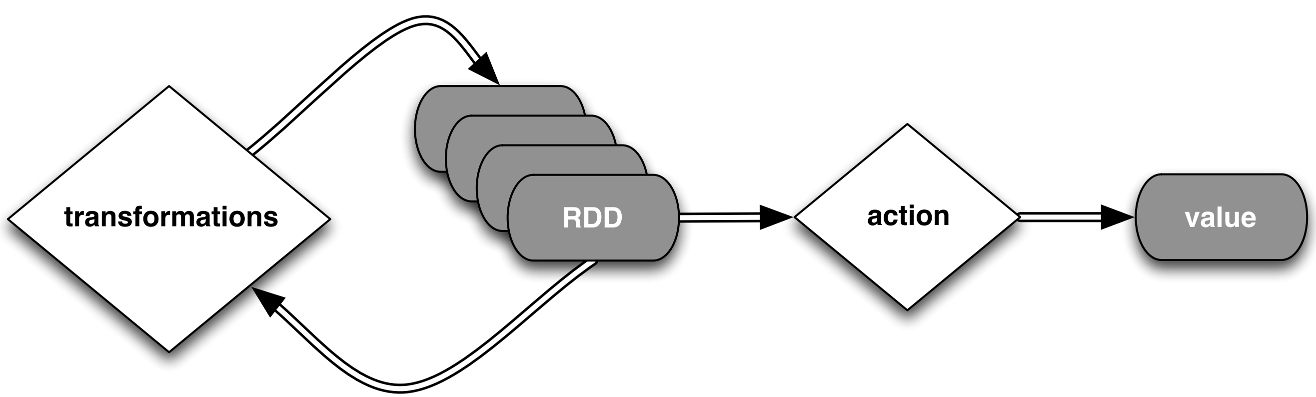
Python:

```
>>> data = [1, 2, 3, 4, 5]
>>> data
[1, 2, 3, 4, 5]

>>> distData = sc.parallelize(data)
>>> distData
ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:229
```

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:

```
scala> val distFile = sc.textFile("README.md")
distFile: spark.RDD[String] = spark.HadoopRDD@1d4cee08
```

Python:

```
>>> distFile = sc.textFile("README.md")
14/04/19 23:42:40 INFO storage.MemoryStore: ensureFreeSpace(36827) called
with curMem=0, maxMem=318111744
14/04/19 23:42:40 INFO storage.MemoryStore: Block broadcast_0 stored as
values to memory (estimated size 36.0 KB, free 303.3 MB)
>>> distFile
MappedRDD[2] at textFile at NativeMethodAccessorImpl.java:-2
```

Spark Essentials: *Transformations*

Transformations create a new dataset from an existing one

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

- optimize the required calculations
- recover from lost data partitions

Spark Essentials: *Transformations*

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

Spark Essentials: *Transformations*

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

Spark Essentials: *Transformations*

Scala:

```
val distFile = sc.textFile("README.md")
distFile.map(l => l.split(" ")).collect()
distFile.flatMap(l => l.split(" ")).collect()
```

distFile is a collection of lines

Python:

```
distFile = sc.textFile("README.md")
distFile.map(lambda x: x.split(' ')).collect()
distFile.flatMap(lambda x: x.split(' ')).collect()
```

Spark Essentials: *Transformations*

Scala:

```
val distFile = sc.textFile("README.md")
distFile.map(l => l.split(" ")).collect()
distFile.flatMap(l => l.split(" ")).collect()
```

closures

Python:

```
distFile = sc.textFile("README.md")
distFile.map(lambda x: x.split(' ')).collect()
distFile.flatMap(lambda x: x.split(' ')).collect()
```

Spark Essentials: *Transformations*

Scala:

```
val distFile = sc.textFile("README.md")
distFile.map(l => l.split(" ")).collect()
distFile.flatMap(l => l.split(" ")).collect()
```

closures

Python:

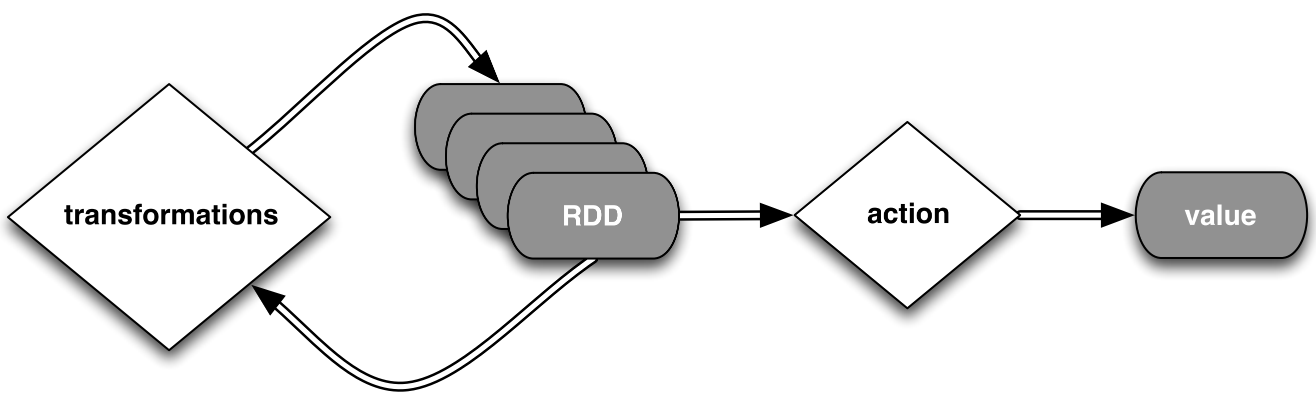
```
distFile = sc.textFile("README.md")
distFile.map(lambda x: x.split(' ')).collect()
distFile.flatMap(lambda x: x.split(' ')).collect()
```

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

Spark Essentials: *Transformations*

Using closures is now possible in Java 8 with *lambda expressions* support, see the tutorial:

databricks.com/blog/2014/04/14/Spark-with-Java-8.html



Spark Essentials: *Transformations*

Java 7:

```
JavaRDD<String> distFile = sc.textFile("README.md");

// Map each line to multiple words
JavaRDD<String> words = distFile.flatMap(
  new FlatMapFunction<String, String>() {
    public Iterable<String> call(String line) {
      return Arrays.asList(line.split(" "));
    }
});
```

Java 8:

```
JavaRDD<String> distFile = sc.textFile("README.md");
JavaRDD<String> words =
    distFile.flatMap(line -> Arrays.asList(line.split(" ")));
```

Spark Essentials: *Actions*

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

Spark Essentials: *Actions*

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

Spark Essentials: *Actions*

Scala:

```scala
val f = sc.textFile("README.md")
val words = f.flatMap(l => l.split(" ")).map(word => (word, 1))
words.reduceByKey(_ + _).collect.foreach(println)
```

Python:

```python
from operator import add
f = sc.textFile("README.md")
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

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.

See:

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

Spark Essentials: *Persistence*

Scala:

```
val f = sc.textFile("README.md")
val w = f.flatMap(l => l.split(" ")).map(word => (word, 1)).cache()
w.reduceByKey(_ + _).collect.foreach(println)
```

Python:

```
from operator import add
f = sc.textFile("README.md")
w = f.flatMap(lambda x: x.split(' ')).map(lambda x: (x, 1)).cache()
w.reduceByKey(add).collect()
```

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
```

Python:

```
broadcastVar = sc.broadcast(list(range(1, 4)))
broadcastVar.value
```

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
```

Python:

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

rdd.foreach(f)

accum.value
```

Spark Essentials: *Accumulators*

Scala:

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

accum.value
```

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
```

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
```

Java:

```
Tuple2 pair = new Tuple2(a, b);

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

For more details about the Scala/Java 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/

06: Intro Spark Apps

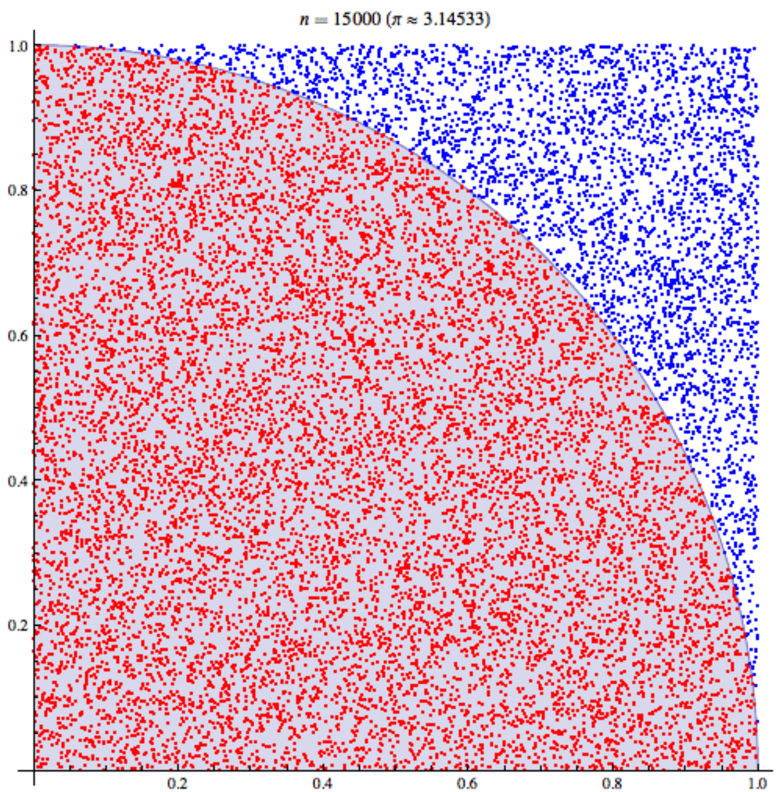
Spark Examples

lecture/lab: 10 min

Spark Examples: *Estimate Pi*

Next, try using a **Monte Carlo method** to estimate the value of Pi

```
./bin/run-example SparkPi 2 local
```



wikipedia.org/wiki/Monte_Carlo_method

Spark Examples: *Estimate Pi*

```scala
import scala.math.random
import org.apache.spark._

/** Computes an approximation to pi */
object SparkPi {
  def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("Spark Pi")
    val spark = new SparkContext(conf)

    val slices = if (args.length > 0) args(0).toInt else 2
    val n = 100000 * slices

    val count = spark.parallelize(1 to n, slices).map { i =>
      val x = random * 2 - 1
      val y = random * 2 - 1
      if (x*x + y*y < 1) 1 else 0
    }.reduce(_ + _)

    println("Pi is roughly " + 4.0 * count / n)
    spark.stop()
  }
}
```

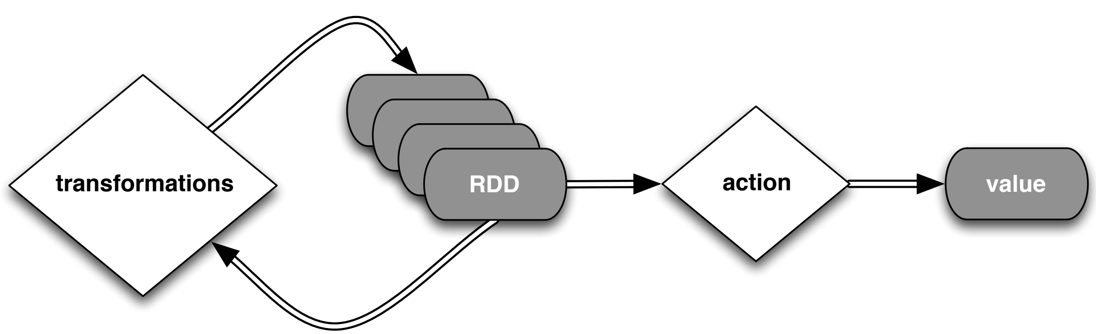
Spark Examples: *Estimate Pi*

```
val count = sc.parallelize(1 to n, slices)

  .map { i =>
   val x = random * 2 - 1
   val y = random * 2 - 1
   if (x*x + y*y < 1) 1 else 0
  }

  .reduce(_ + _)
```

base RDD

transformed RDD

action



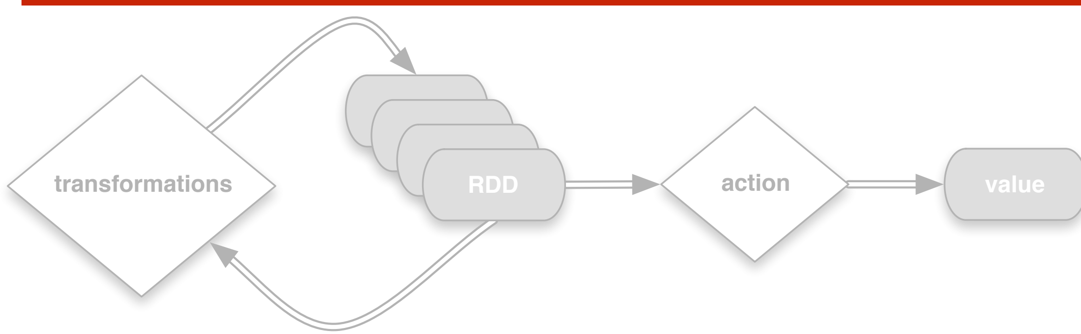
Spark Examples: *Estimate Pi*

```
val count

 .map
  val
  val
  if
 }

 .reduc
```

base RDD

transformed RDD

Checkpoint:
what estimate do you get for Pi?



Spark Examples: *K-Means*

Next, try using **K-Means** to cluster a set of vector values:

```
cp ../data/examples-data/kmeans_data.txt .
./bin/run-example SparkKMeans kmeans_data.txt 3 0.01 local
```

Based on the data set:

```
0.0 0.0 0.0
0.1 0.1 0.1
0.2 0.2 0.2
9.0 9.0 9.0
9.1 9.1 9.1
9.2 9.2 9.2
```

Please refer to the source code in:

```
examples/src/main/scala/org/apache/spark/examples/SparkKMeans.scala
```

Spark Examples: *PageRank*

Next, try using **PageRank** to rank the relationships in a graph:

```
cp ../data/examples-data/pagerank_data.txt .
./bin/run-example SparkPageRank pagerank_data.txt 10 local
```

Based on the data set:

```
1 2
1 3
1 4
2 1
3 1
4 1
```

Please refer to the source code in:

```
examples/src/main/scala/org/apache/spark/examples/SparkPageRank.scala
```

(lunch)

lunch: 60 min -ish

Lunch:

Depending on the venue:

- *if not catered, we're off to find food!*
- *we'll lock the room to secure valuables*

Let's take an hour or so…

Networking is some of the best part of these workshops!

07: Data Workflows

Unifying the Pieces

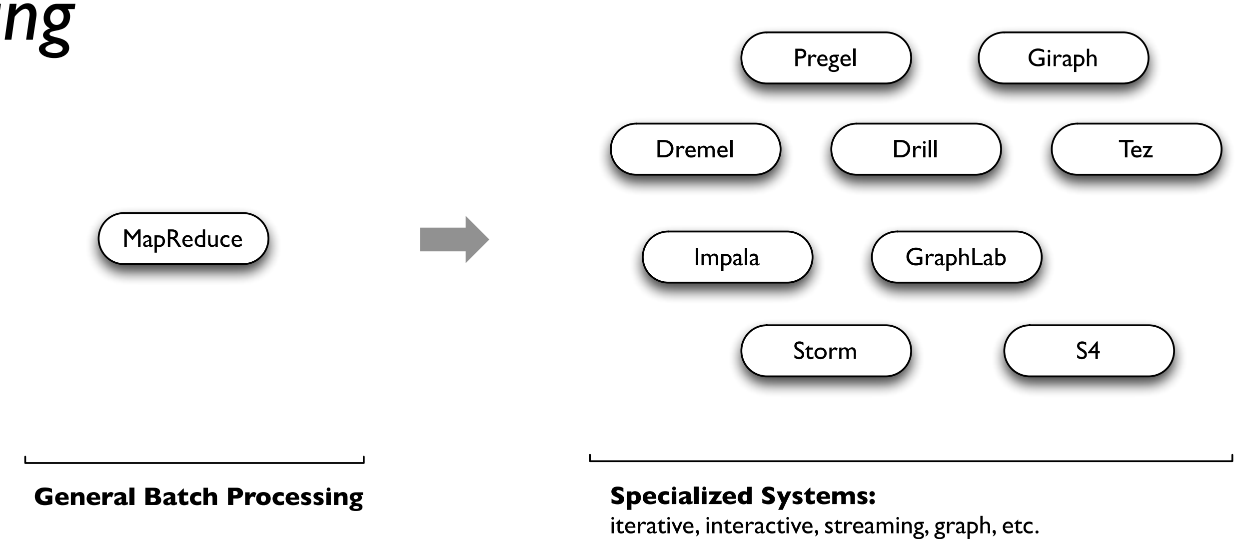
lecture/demo: 45 min

Data Workflows:

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

Two reasonably small additions allowed the previous specialized models to be expressed within Spark:

- *fast data sharing*
- *general DAGs*



Data Workflows:

Unifying the pieces into a single app:
Spark SQL, Streaming, MLlib, GraphX, etc.

- discuss how the same business logic can be deployed across multiple topologies
- demo Spark SQL, Spark Streaming
- discuss MLlib, GraphX

Data Workflows: *Spark SQL*

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
- Shark is being migrated to Spark SQL

Spark SQL: Manipulating Structured Data Using Spark
Michael Armbrust, Reynold Xin (2014-03-24)
databricks.com/blog/2014/03/26/Spark-SQL-manipulating-structured-data-using-Spark.html

Data Workflows: *Spark SQL*

```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
import sqlContext._

// Define the schema using a case class.
case class Person(name: String, age: Int)

// Create an RDD of Person objects and register it as a table.
val people = sc.textFile("examples/src/main/resources/
people.txt").map(_.split(",")).map(p => Person(p(0), p(1).trim.toInt))

people.registerTempTable("people")

// SQL statements can be run by using the sql methods provided by sqlContext.
val teenagers = sql("SELECT name FROM people WHERE age >= 13 AND age <= 19")

// The results of SQL queries are SchemaRDDs and support all the
// normal RDD operations.
// The columns of a row in the result can be accessed by ordinal.
teenagers.map(t => "Name: " + t(0)).collect().foreach(println)
```

Data Workflows: *Spark SQL*

```
val sqlContext
import

// Define the schema using a case class.
case class

// Create an RDD of Person objects and register it as a table.
val people
people.txt"

people

// SQL stat                                              ovided by
sqlContext.
val teenagers

// The results of SQL queries are SchemaRDDs and support all the
// normal RDD operations.
// The columns of a row in the result can be accessed by ordinal.
teenagers
```

Checkpoint:
what name do you get?

Data Workflows: *Spark SQL: queries in HiveQL*

```
//val sc: SparkContext // An existing SparkContext.
//NB: example on laptop lacks a Hive MetaStore
val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc)

// Importing the SQL context gives access to all the
// public SQL functions and implicit conversions.
import hiveContext._

hql("CREATE TABLE IF NOT EXISTS src (key INT, value STRING)")
hql("LOAD DATA LOCAL INPATH 'examples/src/main/resources/kv1.txt' INTO TABLE src")

// Queries are expressed in HiveQL
hql("FROM src SELECT key, value").collect().foreach(println)
```

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)

Modifying the previous example…

Data Workflows: *Spark SQL: Parquet*

```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
import sqlContext._

// Define the schema using a case class.
case class Person(name: String, age: Int)

// Create an RDD of Person objects and register it as a table.
val people = sc.textFile("examples/src/main/resources/
people.txt").map(_.split(",")).map(p => Person(p(0), p(1).trim.toInt))
people.registerTempTable("people")

// The RDD is implicitly converted to a SchemaRDD
## allowing it to be stored using parquet.
people.saveAsParquetFile("people.parquet")

// Read in the parquet file created above.  Parquet files are
// self-describing so the schema is preserved.
// The result of loading a parquet file is also a JavaSchemaRDD.
val parquetFile = sqlContext.parquetFile("people.parquet")

//Parquet files can also be registered as tables and then used in
// SQL statements.
parquetFile.registerTempTable("parquetFile")
val teenagers =
  sql("SELECT name FROM parquetFile WHERE age >= 13 AND age <= 19")
teenagers.collect().foreach(println)
```

Data Workflows: *Spark SQL: Parquet*

In particular, check out the *query plan* in the console output:

```
== Query Plan ==
Project [name#4:0]
 Filter ((age#5:1 >= 13) && (age#5:1 <= 19))
  ParquetTableScan [name#4,age#5], (ParquetRelation people.parquet), None
```

generated from the SQL query:

```
  SELECT name FROM parquetFile WHERE age >= 13 AND age <= 19
```

An output directory get created for each Parquet "file":

```
$ ls people.parquet/
._SUCCESS.crc       .part-r-1.parquet.crc   _SUCCESS        part-r-1.parquet
._metadata.crc      .part-r-2.parquet.crc   _metadata       part-r-2.parquet

$ file people.parquet/part-r-1.parquet
people.parquet/part-r-1.parquet: Par archive data
```

gist.github.com/ceteri/f2c3486062c9610eac1d#file-05-spark-sql-parquet-txt

Spark SQL also provides a DSL for queries

Scala symbols represent columns in the underlying table, which are identifiers prefixed with a tick (')

For a full list of the functions supported, see:

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

…again, modifying the previous example

For a comparison, check out LINQ:
linqpad.net/WhyLINQBeatsSQL.aspx

Data Workflows: *Spark SQL: DSL*

```scala
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
import sqlContext._

// Define the schema using a case class.
case class Person(name: String, age: Int)

// Create an RDD of Person objects and register it as a table.
val people = sc.textFile("examples/src/main/resources/
people.txt").map(_.split(",")).map(p => Person(p(0), p(1).trim.toInt))

// The following is the same as
// 'SELECT name FROM people WHERE age >= 13 AND age <= 19'
val teenagers = people.where('age >= 13).where('age <= 19).select('name)

// The results of SQL queries are SchemaRDDs and support all the
// normal RDD operations.
// The columns of a row in the result can be accessed by ordinal.
teenagers.map(t => "Name: " + t(0)).collect().foreach(println)
```

Let's also take a look at Spark SQL in PySpark, using **IPython Notebook**…

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

To launch:

```
IPYTHON_OPTS="notebook --pylab inline" ./bin/pyspark
```

IP[y]: IPython
Interactive Computing

Data Workflows: *Spark SQL: PySpark*

```python
from pyspark.sql import SQLContext, Row
sqlCtx = SQLContext(sc)

# Load a text file and convert each line to a dictionary
lines = sc.textFile("examples/src/main/resources/people.txt")
parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))

# Infer the schema, and register the SchemaRDD as a table.
# In future versions of PySpark we would like to add support
# for registering RDDs with other datatypes as tables
peopleTable = sqlCtx.inferSchema(people)
peopleTable.registerTempTable("people")

# SQL can be run over SchemaRDDs that have been registered as a table
teenagers = sqlCtx.sql("SELECT name FROM people WHERE age >= 13 AND age <= 19")

teenNames = teenagers.map(lambda p: "Name: " + p.name)
teenNames.collect()
```

Data Workflows: *Spark Streaming*

Spark Streaming extends the core API to allow high-throughput, fault-tolerant stream processing of live data streams

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

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

Data Workflows: *Spark Streaming*

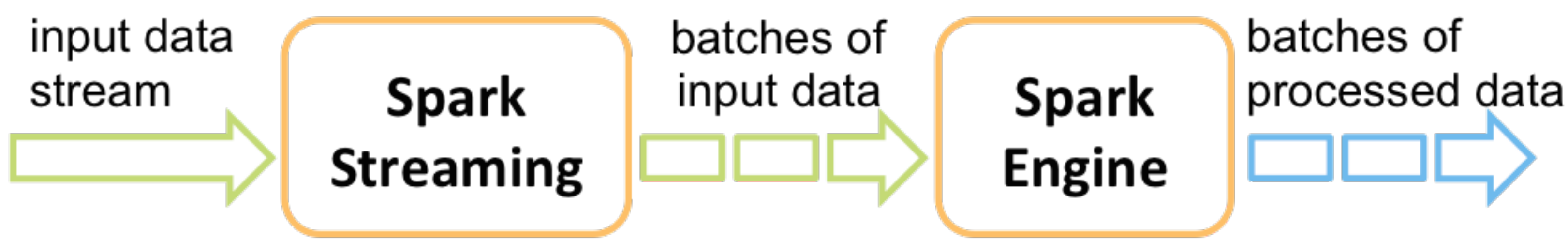
Data can be ingested from many sources: **Kafka**, **Flume**, **Twitter**, **ZeroMQ**, TCP sockets, etc.

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

Comparisons:

- Twitter **Storm**
- Yahoo! **S4**
- Google **MillWheel**



Data Workflows: *Spark Streaming*

```
# in one terminal run the NetworkWordCount example in Spark Streaming
# expecting a data stream on the localhost:9999 TCP socket
./bin/run-example org.apache.spark.examples.streaming.NetworkWordCount
localhost 9999
```

```
# in another terminal use Netcat http://nc110.sourceforge.net/
# to generate a data stream on the localhost:9999 TCP socket
$ nc -lk 9999
hello world
hi there fred
what a nice world there
```

Data Workflows: *Spark Streaming*

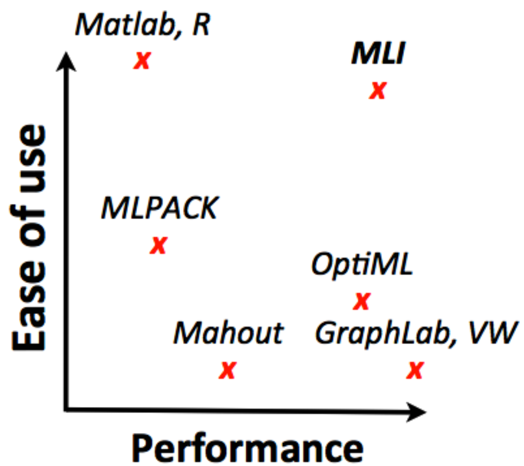
```
// http://spark.apache.org/docs/latest/streaming-programming-guide.html

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

// create a StreamingContext
val ssc = new StreamingContext(sc, 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()              // start the computation
ssc.awaitTermination()   // wait for the computation to terminate
```

What the stream analysis produced:

```
14/04/19 13:41:28 INFO scheduler.TaskSetManager: Finished TID 3 in 17 ms on localhost
(progress: 1/1)
14/04/19 13:41:28 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 3.0, whose tasks
have all completed, from pool
14/04/19 13:41:28 INFO scheduler.DAGScheduler: Completed ResultTask(3, 1)
14/04/19 13:41:28 INFO scheduler.DAGScheduler: Stage 3 (take at DStream.scala:583)
finished in 0.019 s
14/04/19 13:41:28 INFO spark.SparkContext: Job finished: take at DStream.scala:583,
took 0.034258 s
-------------------------------------------
Time: 1397940088000 ms
-------------------------------------------
(hello,1)
(what,1)
(world,2)
(there,2)
(fred,1)
(hi,1)
(a,1)
(nice,1)
```

Data Workflows: *MLlib*

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

val train_data = // RDD of Vector
val model = KMeans.train(train_data, k=10)

// evaluate the model
val test_data = // RDD of Vector
test_data.map(t => model.predict(t)).collect().foreach(println)
```



MLI: An API for Distributed Machine Learning
Evan Sparks, **Ameet Talwalkar**, et al.
International Conference on Data Mining (2013)
http://arxiv.org/abs/1310.5426

demo:

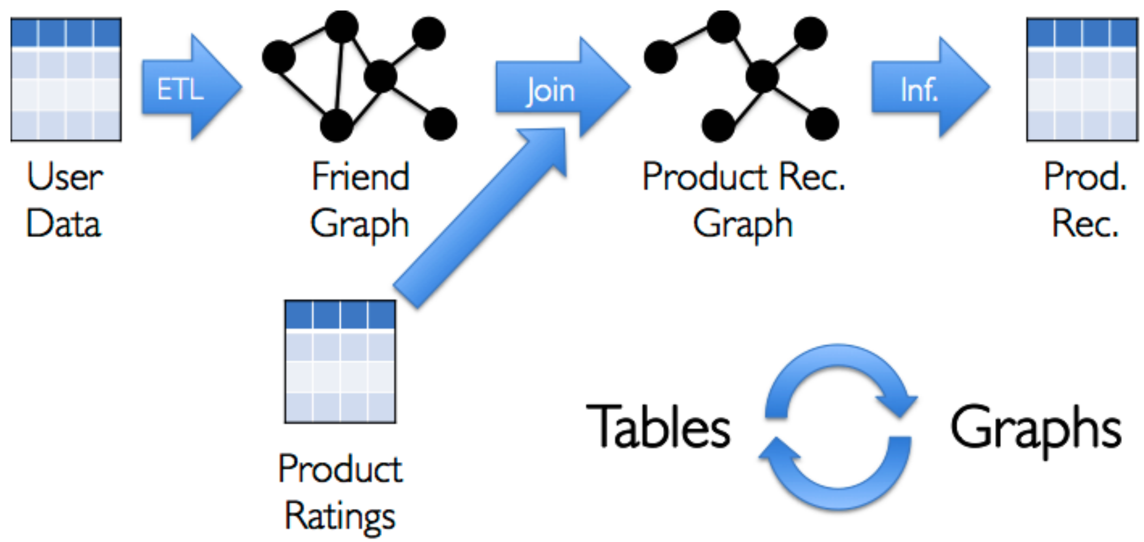
Twitter Streaming Language Classifier
databricks.gitbooks.io/databricks-spark-reference-applications/twitter_classifier/README.html

GraphX **amplab.github.io/graphx/**

extends the distributed fault-tolerant collections API and interactive console of Spark with a new graph API which leverages recent advances in graph systems (e.g., GraphLab) to enable users to easily and interactively build, transform, and reason about graph structured data at scale

Data Workflows: *GraphX*

unifying graphs and tables



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

ampcamp.berkeley.edu/big-data-mini-course/graph-analytics-with-graphx.html

Data Workflows: *GraphX*

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

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

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

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

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

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

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

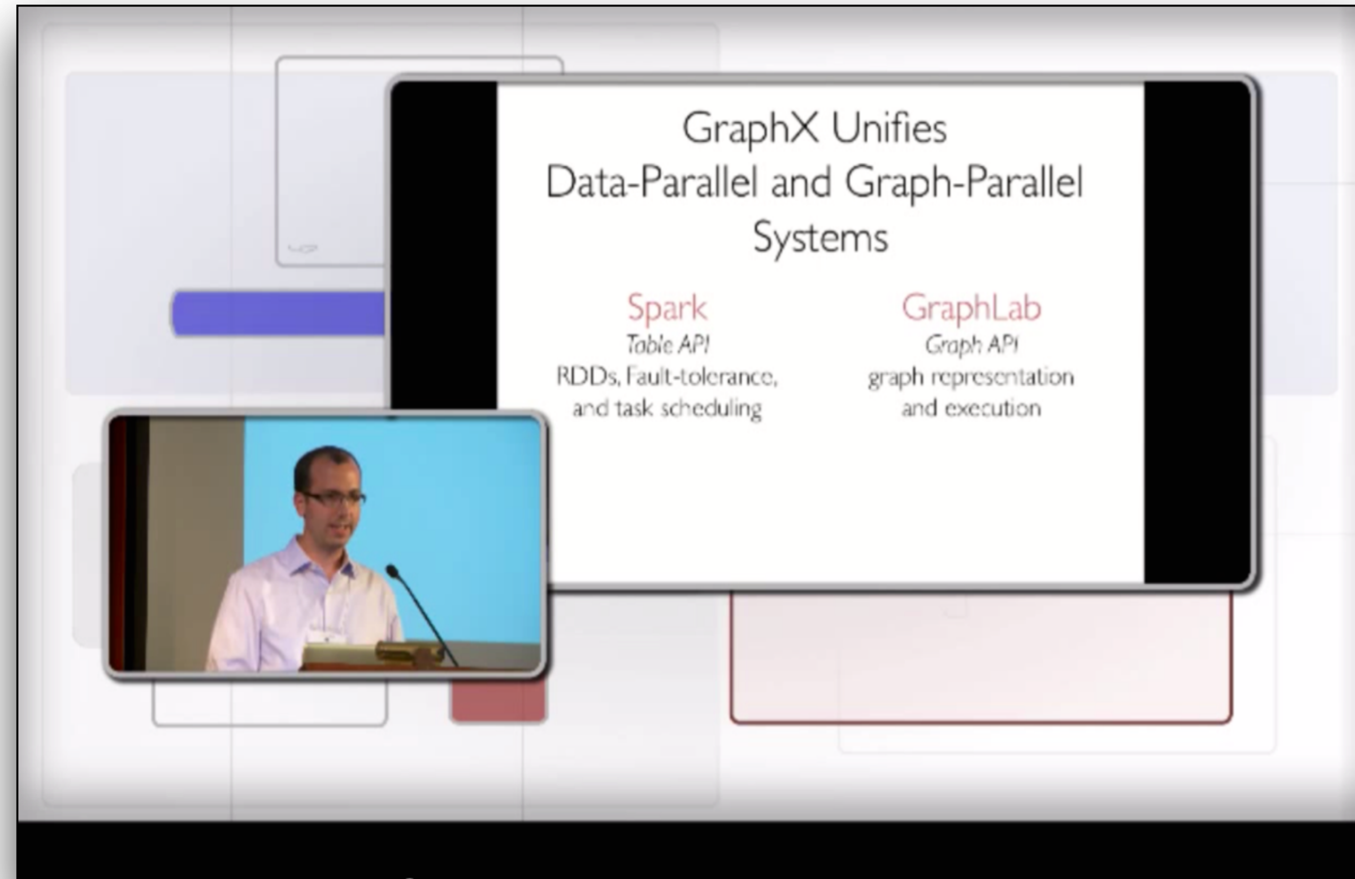
Data Workflows: *GraphX*

demo:

Simple Graph Query
gist.github.com/ceteri/c2a692b5161b23d92ed1

Data Workflows: *GraphX*

Introduction to GraphX
Joseph Gonzalez, Reynold Xin
youtu.be/mKEn9C5bRck



(break)

break: 15 min

08: Spark in Production

The Full SDLC

lecture/lab: 75 min

Spark in Production:

In the following, let's consider the progression through a full software development lifecycle, step by step:

1. **build**
2. **deploy**
3. **monitor**

builds:

- build/run a JAR using Java + Maven
- SBT primer
- build/run a JAR using Scala + SBT

The following sequence shows how to build a JAR file from a Java app, using Maven

maven.apache.org/guides/introduction/introduction-to-the-pom.html

- First, connect into a *different* directory where you have space to create several files
- Then run the following commands…

Spark in Production: *Build: Java*

```
# Java source (cut&paste 1st following slide)
mkdir -p src/main/java
cat > src/main/java/SimpleApp.java

# project model (cut&paste 2nd following slide)
cat > pom.xml

# copy a file to use for data
cp $SPARK_HOME/README.md .

# build the JAR
mvn clean package

# run the JAR
mvn exec:java -Dexec.mainClass="SimpleApp"
```

Spark in Production: *Build: Java*

```java
/*** SimpleApp.java ***/
import org.apache.spark.api.java.*;
import org.apache.spark.api.java.function.Function;

public class SimpleApp {
  public static void main(String[] args) {
    String logFile = "README.md";
    JavaSparkContext sc = new JavaSparkContext("local", "Simple App",
      "$SPARK_HOME", new String[]{"target/simple-project-1.0.jar"});
    JavaRDD<String> logData = sc.textFile(logFile).cache();

    long numAs = logData.filter(new Function<String, Boolean>() {
      public Boolean call(String s) { return s.contains("a"); }
    }).count();

    long numBs = logData.filter(new Function<String, Boolean>() {
      public Boolean call(String s) { return s.contains("b"); }
    }).count();

    System.out.println("Lines with a: " + numAs + ", lines with b: " + numBs);
  }
}
```

Spark in Production: *Build: Java*

```xml
<project>
  <groupId>edu.berkeley</groupId>
  <artifactId>simple-project</artifactId>
  <modelVersion>4.0.0</modelVersion>
  <name>Simple Project</name>
  <packaging>jar</packaging>
  <version>1.0</version>
  <repositories>
    <repository>
      <id>Akka repository</id>
      <url>http://repo.akka.io/releases</url>
    </repository>
  </repositories>
  <dependencies>
    <dependency> <!-- Spark dependency -->
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-core_2.10</artifactId>
      <version>1.2.0</version>
    </dependency>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-client</artifactId>
      <version>2.2.0</version>
    </dependency>
  </dependencies>
</project>
```

Source files, commands, and expected output are shown in this gist:

gist.github.com/ceteri/f2c3486062c9610eac1d#file-04-java-maven-txt

…and the JAR file that we just used:

```
ls target/simple-project-1.0.jar
```

builds:

- build/run a JAR using Java + Maven
- SBT primer
- build/run a JAR using Scala + SBT

SBT is the **S**imple **B**uild **T**ool for Scala:

www.scala-sbt.org/

This is included with the Spark download, and does not need to be installed separately.

Similar to Maven, however it provides for *incremental compilation* and an *interactive shell*, among other innovations.

SBT project uses *StackOverflow* for Q&A, that's a good resource to study further:

stackoverflow.com/tags/sbt

Spark in Production: *Build: SBT*

command	*description*
`clean`	delete all generated files (in the *target* directory)
`package`	create a JAR file
`run`	run the JAR (or main class, if named)
`compile`	compile the main sources (in *src/main/scala* and *src/main/java* directories)
`test`	compile and run all tests
`console`	launch a Scala interpreter
`help`	display detailed help for specified commands

builds:

- build/run a JAR using Java + Maven
- SBT primer
- build/run a JAR using Scala + SBT

The following sequence shows how to build a JAR file from a Scala app, using SBT

- First, this requires the “source” download, not the “binary”
- Connect into the `SPARK_HOME` directory
- Then run the following commands…

Spark in Production: *Build: Scala*

```
# Scala source + SBT build script on following slides

cd simple-app

../sbt/sbt -Dsbt.ivy.home=../sbt/ivy package

../spark/bin/spark-submit \
  --class "SimpleApp" \
  --master local[*] \
  target/scala-2.10/simple-project_2.10-1.0.jar
```

Spark in Production: *Build: Scala*

```scala
/*** SimpleApp.scala ***/
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._

object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "README.md" // Should be some file on your system
    val sc = new SparkContext("local", "Simple App", "SPARK_HOME",
      List("target/scala-2.10/simple-project_2.10-1.0.jar"))
    val logData = sc.textFile(logFile, 2).cache()

    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()

    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))
  }
}
```

Spark in Production: *Build: Scala*

```
name := "Simple Project"

version := "1.0"

scalaVersion := "2.10.4"

libraryDependencies += "org.apache.spark" % "spark-core_2.10" % "1.2.0"

resolvers += "Akka Repository" at "http://repo.akka.io/releases/"
```

Spark in Production: *Deploy*

deploy JAR to Hadoop cluster, using these alternatives:

- discuss how to run atop Apache Mesos
- discuss how to install on CM
- discuss how to run on HDP
- discuss how to run on MapR
- discuss how to run on EC2
- discuss using **SIMR** (run shell within MR job)
- …or, simply run the JAR on YARN

Spark in Production: *Deploy: Mesos*

deploy JAR to Hadoop cluster, using these alternatives:

- discuss how to run atop Apache Mesos
- discuss how to install on CM
- discuss how to run on HDP
- discuss how to run on MapR
- discuss how to run on EC2
- discuss using **SIMR** (run shell within MR job)
- …or, simply run the JAR on YARN

Apache Mesos, from which Apache Spark originated…

Running Spark on Mesos

spark.apache.org/docs/latest/running-on-mesos.html

Run Apache Spark on Apache Mesos
tutorial based on Mesosphere + Google Cloud
ceteri.blogspot.com/2014/09/spark-atop-mesos-on-google-cloud.html

Getting Started Running Apache Spark on Apache Mesos
O'Reilly Media webcast
oreilly.com/pub/e/2986

Spark in Production: *Deploy: CM*

deploy JAR to Hadoop cluster, using these alternatives:

- discuss how to run atop Apache Mesos
- discuss how to install on CM
- discuss how to run on HDP
- discuss how to run on MapR
- discuss how to run on EC2
- discuss using **SIMR** (run shell within MR job)
- …or, simply run the JAR on YARN

Cloudera Manager 4.8.x:

cloudera.com/content/cloudera-content/cloudera-docs/CM4Ent/latest/Cloudera-Manager-Installation-Guide/cmig_spark_installation_standalone.html

- 5 steps to install the Spark parcel
- 5 steps to configure and start the Spark service

Also check out Cloudera Live:

cloudera.com/content/cloudera/en/products-and-services/cloudera-live.html

deploy JAR to Hadoop cluster, using these alternatives:

- discuss how to run atop Apache Mesos
- discuss how to install on CM
- discuss how to run on HDP
- discuss how to run on MapR
- discuss how to run on EC2
- discuss using **SIMR** (run shell within MR job)
- …or, simply run the JAR on YARN

Spark in Production: *Deploy: HDP*

Hortonworks provides support for running Spark on HDP:

spark.apache.org/docs/latest/hadoop-third-party-distributions.html

hortonworks.com/blog/announcing-hdp-2-1-tech-preview-component-apache-spark/

Spark in Production: *Deploy: MapR*

deploy JAR to Hadoop cluster, using these alternatives:

- discuss how to run atop Apache Mesos
- discuss how to install on CM
- discuss how to run on HDP
- discuss how to run on MapR
- discuss how to run on EC2
- discuss using **SIMR** (run shell within MR job)
- …or, simply run the JAR on YARN

MapR Technologies provides support for running Spark on the MapR distros:

mapr.com/products/apache-spark

slideshare.net/MapRTechnologies/map-r-databricks-webinar-4x3

deploy JAR to Hadoop cluster, using these alternatives:

- discuss how to run atop Apache Mesos
- discuss how to install on CM
- discuss how to run on HDP
- discuss how to run on MapR
- discuss how to run on EC2
- discuss using **SIMR** (run shell within MR job)
- …or, simply run the JAR on YARN

Running Spark on Amazon AWS **EC2**:

blogs.aws.amazon.com/bigdata/post/Tx15AY5C50K70RV/Installing-Apache-Spark-on-an-Amazon-EMR-Cluster

deploy JAR to Hadoop cluster, using these alternatives:

- discuss how to run atop Apache Mesos
- discuss how to install on CM
- discuss how to run on HDP
- discuss how to run on MapR
- discuss how to run on EC2
- discuss using **SIMR** (run shell within MR job)
- …or, simply run the JAR on YARN

Spark in Production: *Deploy: SIMR*

Spark in MapReduce (SIMR) – quick way for Hadoop MR1 users to deploy Spark:

databricks.github.io/simr/

spark-summit.org/talk/reddy-simr-let-your-spark-jobs-simmer-inside-hadoop-clusters/

- Sparks run on Hadoop clusters without any install or required admin rights
- SIMR launches a Hadoop job that only contains mappers, includes Scala+Spark

```
./simr jar_file main_class parameters
   [—outdir=] [—slots=N] [—unique]
```

Spark in Production: *Deploy:YARN*

deploy JAR to Hadoop cluster, using these alternatives:

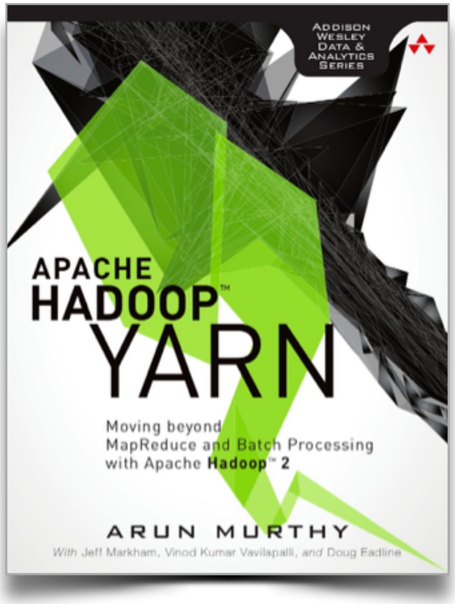
- discuss how to run atop Apache Mesos
- discuss how to install on CM
- discuss how to run on HDP
- discuss how to run on MapR
- discuss how to rum on EMR
- discuss using **SIMR** (run shell within MR job)
- …or, simply run the JAR on YARN

Spark in Production: *Deploy:YARN*

spark.apache.org/docs/latest/running-on-yarn.html

- Simplest way to deploy Spark apps in production
- Does not require admin, just deploy apps to your Hadoop cluster

Apache Hadoop YARN
Arun Murthy, et al.
amazon.com/dp/0321934504



Exploring data sets loaded from HDFS…

1. launch a Spark cluster using EC2 script
2. load data files into HDFS
3. run Spark shell to perform *WordCount*

NB: be sure to use *internal* IP addresses on AWS for the "hdfs://…" URLs

Spark in Production: *Deploy: HDFS examples*

```
# http://spark.apache.org/docs/latest/ec2-scripts.html
cd $SPARK_HOME/ec2

export AWS_ACCESS_KEY_ID=$AWS_ACCESS_KEY
export AWS_SECRET_ACCESS_KEY=$AWS_SECRET_KEY
./spark-ec2 -k spark -i ~/spark.pem -s 2 -z us-east-1b launch foo

# can review EC2 instances and their security groups to identify master
# ssh into master
./spark-ec2 -k spark -i ~/spark.pem -s 2 -z us-east-1b login foo

# use ./ephemeral-hdfs/bin/hadoop to access HDFS
/root/ephemeral-hdfs/bin/hadoop fs -mkdir /tmp
/root/ephemeral-hdfs/bin/hadoop fs -put CHANGES.txt /tmp

# now is the time when we Spark
cd /root/spark
export SPARK_HOME=$(pwd)

SPARK_HADOOP_VERSION=1.0.4 sbt/sbt assembly

/root/ephemeral-hdfs/bin/hadoop fs -put CHANGES.txt /tmp
./bin/spark-shell
```

Spark in Production: *Deploy: HDFS examples*

```
/** NB: replace host IP with EC2 internal IP address **/

val f = sc.textFile("hdfs://10.72.61.192:9000/foo/CHANGES.txt")
val counts =
 f.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(_ + _)

counts.collect().foreach(println)
counts.saveAsTextFile("hdfs://10.72.61.192:9000/foo/wc")
```

Spark in Production: *Deploy: HDFS examples*

Let's check the results in HDFS…

```
root/ephemeral-hdfs/bin/hadoop fs -cat /tmp/wc/part-*
```

```
(Adds,1)
(alpha,2)
(ssh,1)
(graphite,1)
(canonical,2)
(ASF,3)
(display,4)
(synchronization,2)
(instead,7)
(javadoc,1)
(hsaputra/update-pom-asf,1)
```

…

review UI features

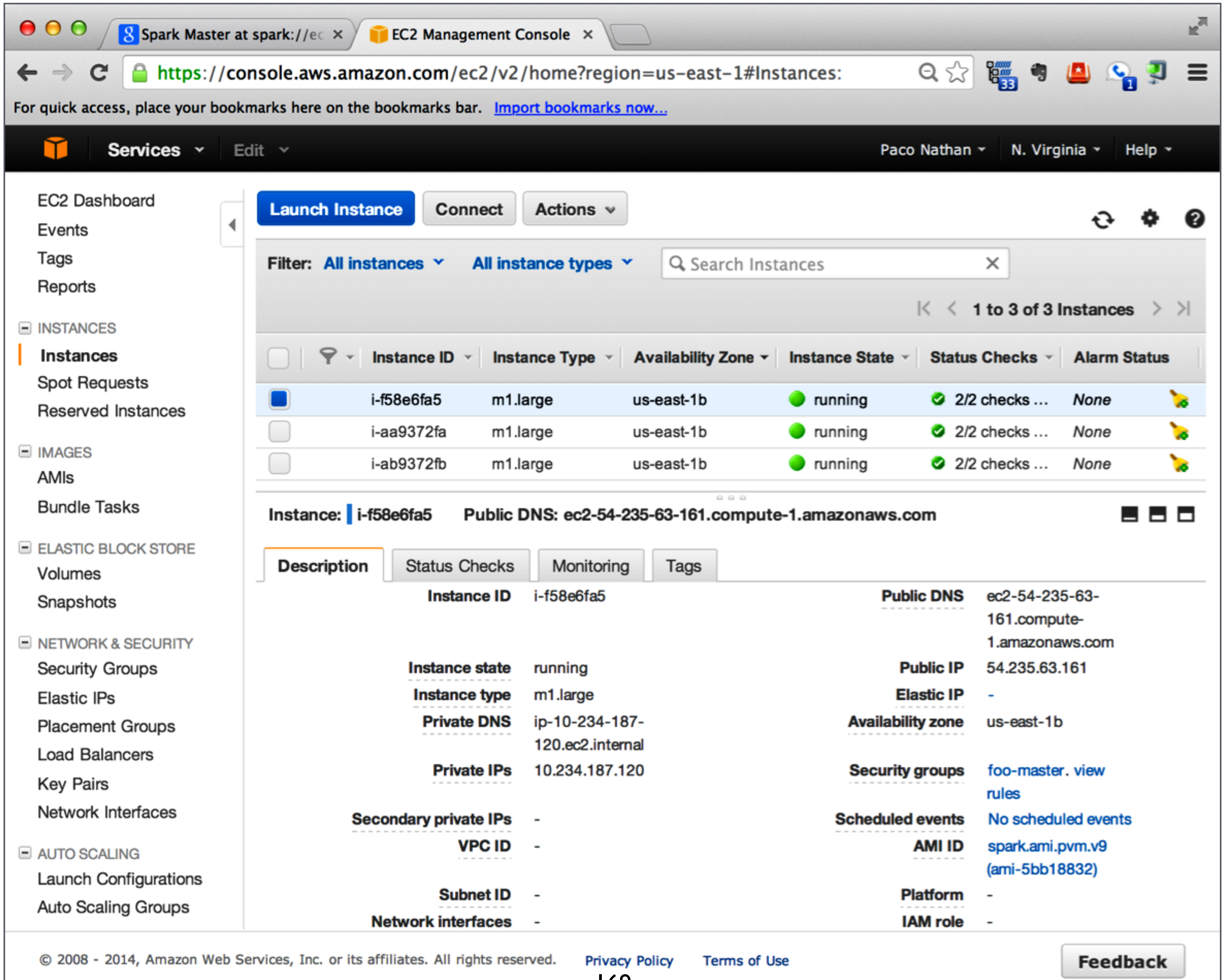
spark.apache.org/docs/latest/monitoring.html

http://<master>:8080/

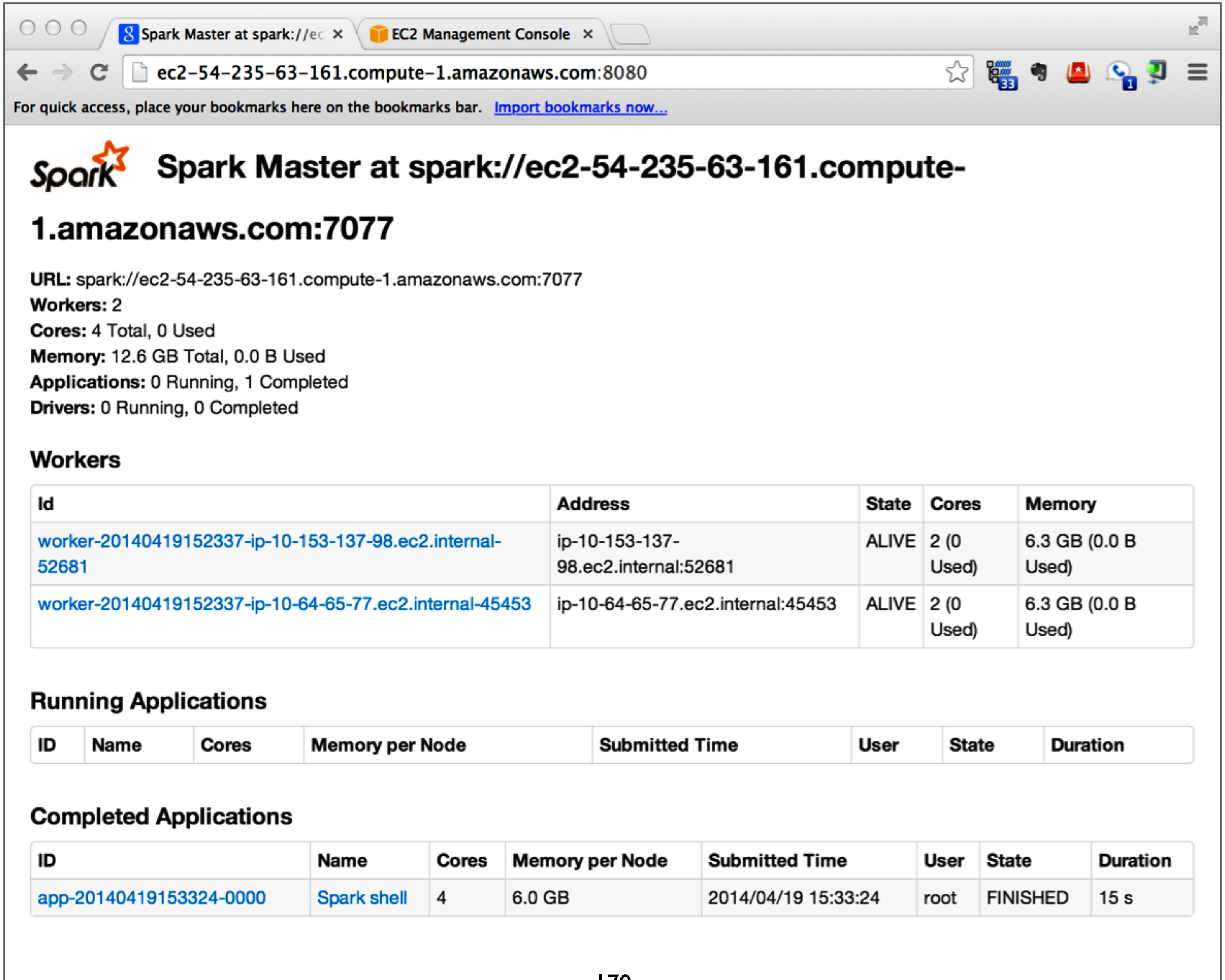
http://<master>:50070/

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

Spark in Production: *Monitor: AWS Console*



Spark in Production: *Monitor: Spark Console*


Spark Master at spark://ec2-54-235-63-161.compute-1.amazonaws.com:7077

URL: spark://ec2-54-235-63-161.compute-1.amazonaws.com:7077
Workers: 2
Cores: 4 Total, 0 Used
Memory: 12.6 GB Total, 0.0 B Used
Applications: 0 Running, 1 Completed
Drivers: 0 Running, 0 Completed

Workers

Id	Address	State	Cores	Memory
worker-20140419152337-ip-10-153-137-98.ec2.internal-52681	ip-10-153-137-98.ec2.internal:52681	ALIVE	2 (0 Used)	6.3 GB (0.0 B Used)
worker-20140419152337-ip-10-64-65-77.ec2.internal-45453	ip-10-64-65-77.ec2.internal:45453	ALIVE	2 (0 Used)	6.3 GB (0.0 B Used)

Running Applications

ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

Completed Applications

ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20140419153324-0000	Spark shell	4	6.0 GB	2014/04/19 15:33:24	root	FINISHED	15 s

Case Studies

discussion: 30 min

http://databricks.com/certified-on-spark

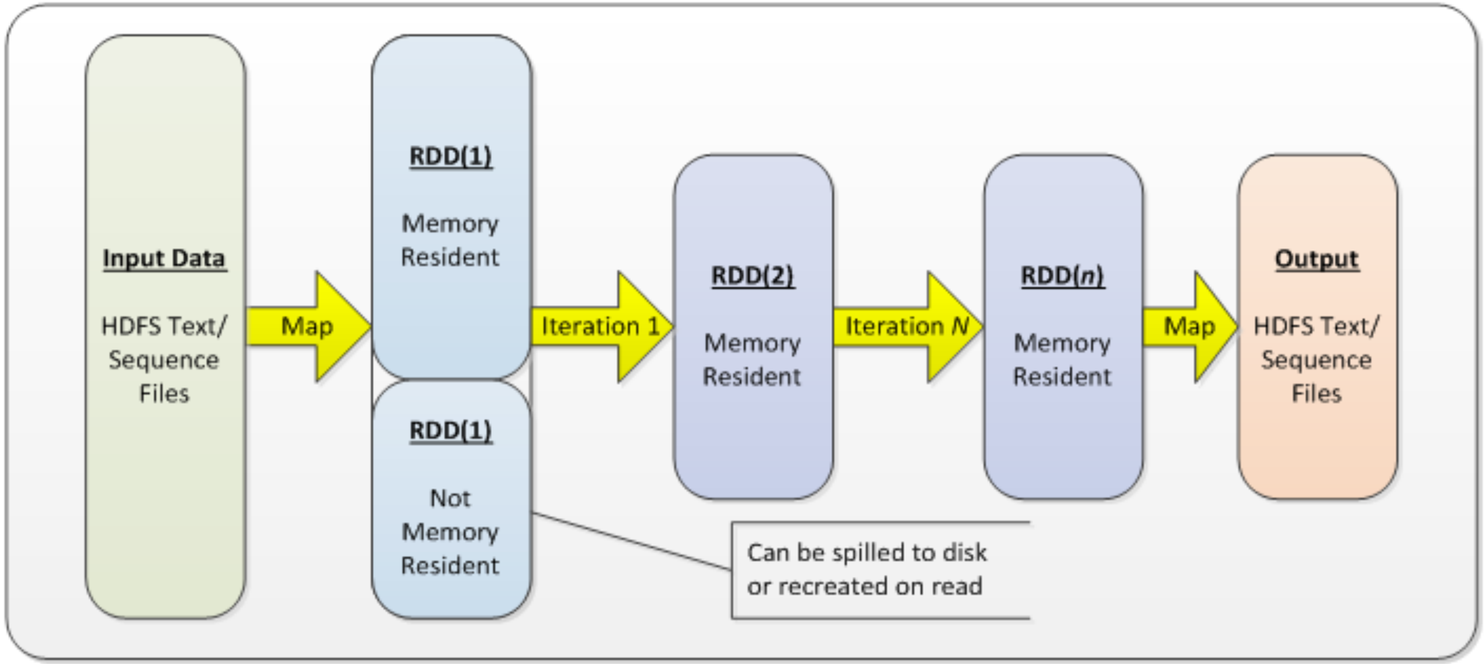
Spark at Twitter: Evaluation & Lessons Learnt
Sriram Krishnan
slideshare.net/krishflix/seattle-spark-meetup-spark-at-twitter

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

Summary: *Case Studies*

Using Spark to Ignite Data Analytics

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

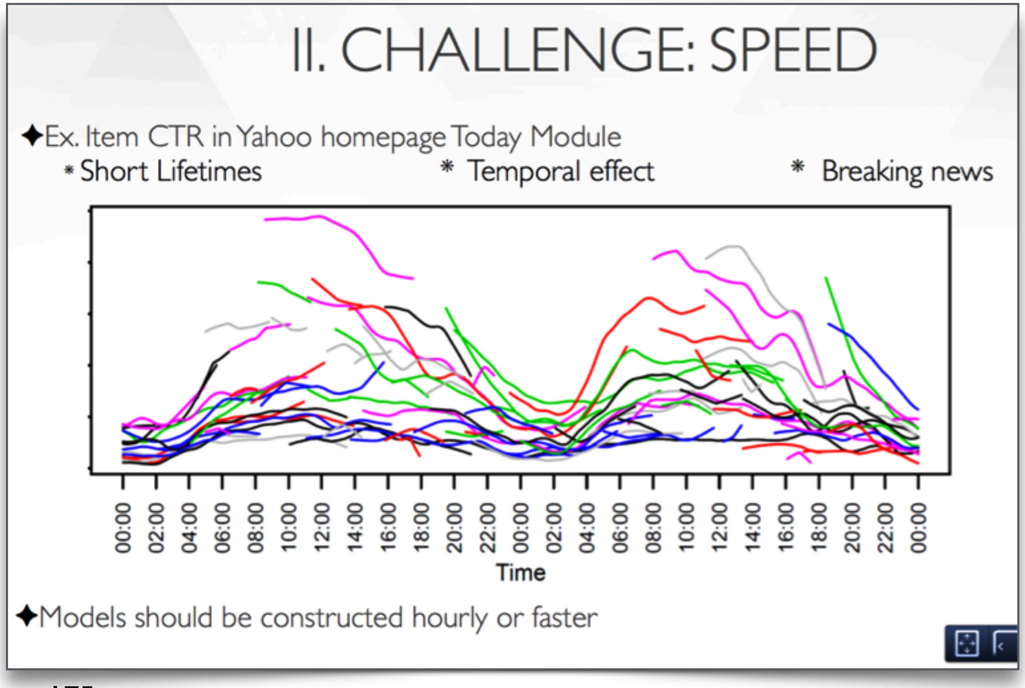


Summary: *Case Studies*

Hadoop and Spark Join Forces in Yahoo

Andy Feng

spark-summit.org/talk/feng-hadoop-and-spark-join-forces-at-yahoo/



Collaborative Filtering with Spark

Chris Johnson

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

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

Summary: *Case Studies*

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

Open Sourcing Our Spark Job Server

Evan Chan

engineering.ooyala.com/blog/open-sourcing-our-spark-job-server

- **github.com/ooyala/spark-jobserver**
- REST server for submitting, running, managing Spark jobs and contexts
- company vision for Spark is as a multi-team big data service
- shares Spark RDDs in one SparkContext among multiple jobs

Summary: *Case Studies*

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

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

Summary: *Case Studies*

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

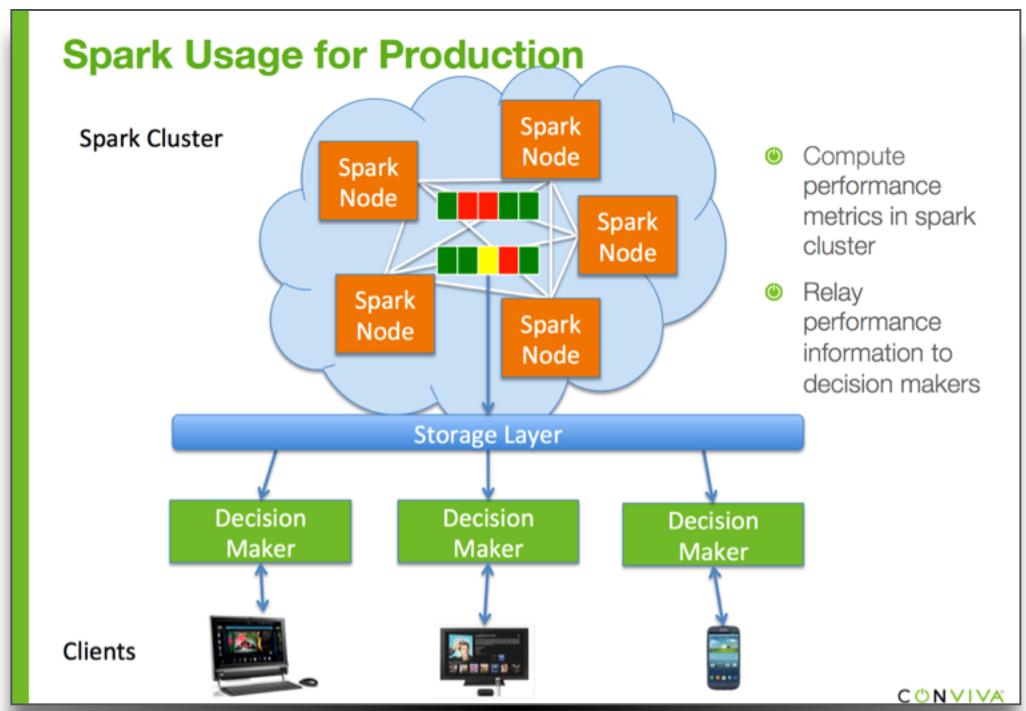
- 4 of 5 top mobile network operators, 3 of 5 top Internet backbone providers, 80% MSOs in NorAm
- analyzing 50% of US mobile data traffic, +2.5 PB/day
- latency is critical for resolving operational issues before they cascade: 2.5 MM transactions per second
- "analyze first" not "store first ask questions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

Summary: *Case Studies*

One platform for all: real-time, near-real-time, and offline video analytics on Spark

Davis Shepherd, Xi Liu

spark-summit.org/talk/one-platform-for-all-real-time-near-real-time-and-offline-video-analytics-on-spark



10: Summary

Follow-Up

discussion: 20 min

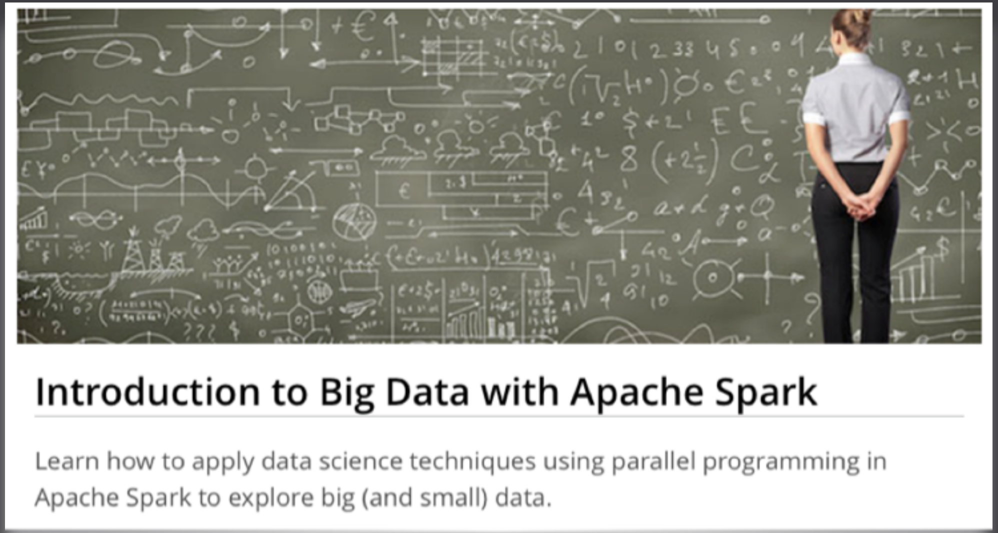
certification:

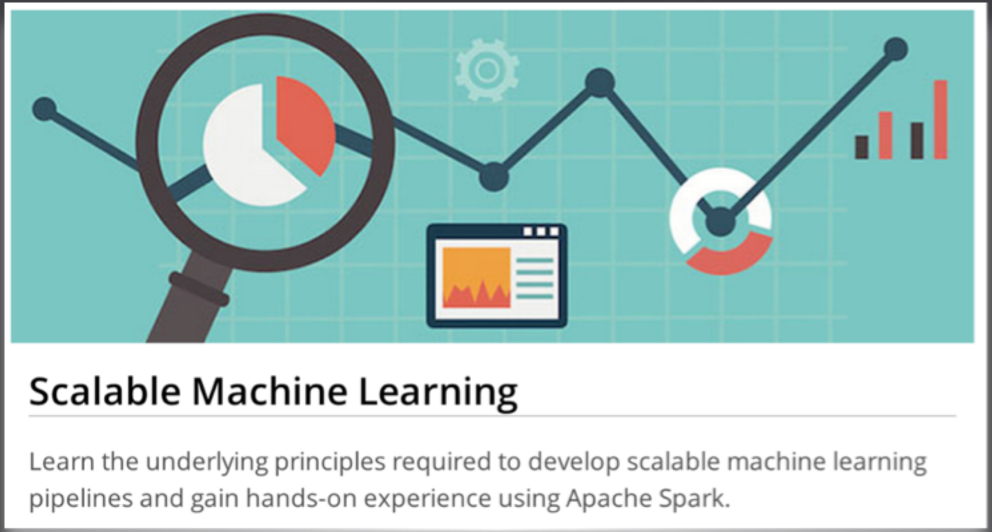
Apache Spark developer certificate program

- http://oreilly.com/go/sparkcert
- defined by Spark experts @Databricks
- assessed by O'Reilly Media
- establishes the bar for Spark expertise

MOOCs:

Anthony Joseph
UC Berkeley
begins 2015-02-23
edx.org/course/uc-berkeleyx/uc-berkeleyx-cs100-1x-introduction-big-6181





Ameet Talwalkar
UCLA
begins 2015-04-14
edx.org/course/uc-berkeleyx/uc-berkeleyx-cs190-1x-scalable-machine-6066

community:

spark.apache.org/community.html

events worldwide: goo.gl/2YqJZK

video+preso archives: spark-summit.org

resources: databricks.com/spark-training-resources

workshops: databricks.com/spark-training

books:

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

events:

Strata CA
San Jose, Feb 18-20
strataconf.com/strata2015

Spark Summit East
NYC, Mar 18-19
spark-summit.org/east

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

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