OVERVIEW

Vadim Bichutskiy
@vybstat

Interface Symposium
June 11, 2015

WHO AM I

- Computational and Data Sciences, PhD Candidate, George Mason
- Lead Data Scientist, Echelon Insights
- Independent Data Science Consultant
- MS/BS Computer Science, MS Statistics
- NOT a Spark expert (yet!)

ACKNOWLEDGEMENTS

- Much of this talk is inspired by *SparkCamp* at *Strata Hadoop World*, San Jose, CA, February 2015 licensed under: Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License

- Taught by Paco Nathan

BIG NEWS TODAY!

Announcing Apache Spark 1.4

June 11, 2015 | by Patrick Wendell

Join us at the Spark Summit to learn more about Spark 1.4. Use the code ***Databricks20*** to receive a 20% discount!

Today I'm excited to announce the general availability of Spark 1.4! Spark 1.4 introduces SparkR, an R API targeted towards data scientists. It also evolves Spark's DataFrame API with a large number of new features. Spark's ML pipelines API first introduced in Spark 1.3 graduates from an alpha component. Finally, Spark Streaming and Core add visualization and monitoring to aid in production debugging. We'll be publishing in-depth posts covering Spark's new features over the coming weeks. Here I'll briefly outline some of the major themes and features in this release.

SparkR ships in Spark

Spark 1.4 introduces SparkR, an R API for Spark and Spark's first new language API since PySpark was added in 2012. SparkR is based on Spark's parallel DataFrame abstraction. Users can create SparkR DataFrames from "local" R data frames, or from any Spark data source such as Hive, HDFS, Parquet or JSON. SparkR DataFrames support all Spark DataFrame operations including aggregation, filtering, grouping, summary statistics, and other analytical functions. They also supports mixing-in SQL queries, and converting query results to and from DataFrames. Because SparkR uses the Spark's parallel engine underneath, operations take advantage of multiple cores or multiple machines, and can scale to data sizes much larger than standalone R programs.

```
print 'hello world!'people <- read.df(sqlContext, "./examples/src/main/resources/
people.json", "json")
head(people)
```

databricks

databricks.com/blog/2015/06/11/announcing-apache-spark-1-4.html

Spark

SPARK HELPS YOU BUILD NEW TOOLS

Josh Wills
@josh_wills

Following

There's no tool that makes you into a data scientist. The ability to push beyond the limit of your tools is what makes you a data scientist.

THIS TALK…

- Part I: Big Data: A Brief History
- Part II: A Tour of Spark
- Part III: Spark Concepts

PART I:

BIG DATA: A BRIEF HISTORY

DOT COM BUBBLE: 1994-2001

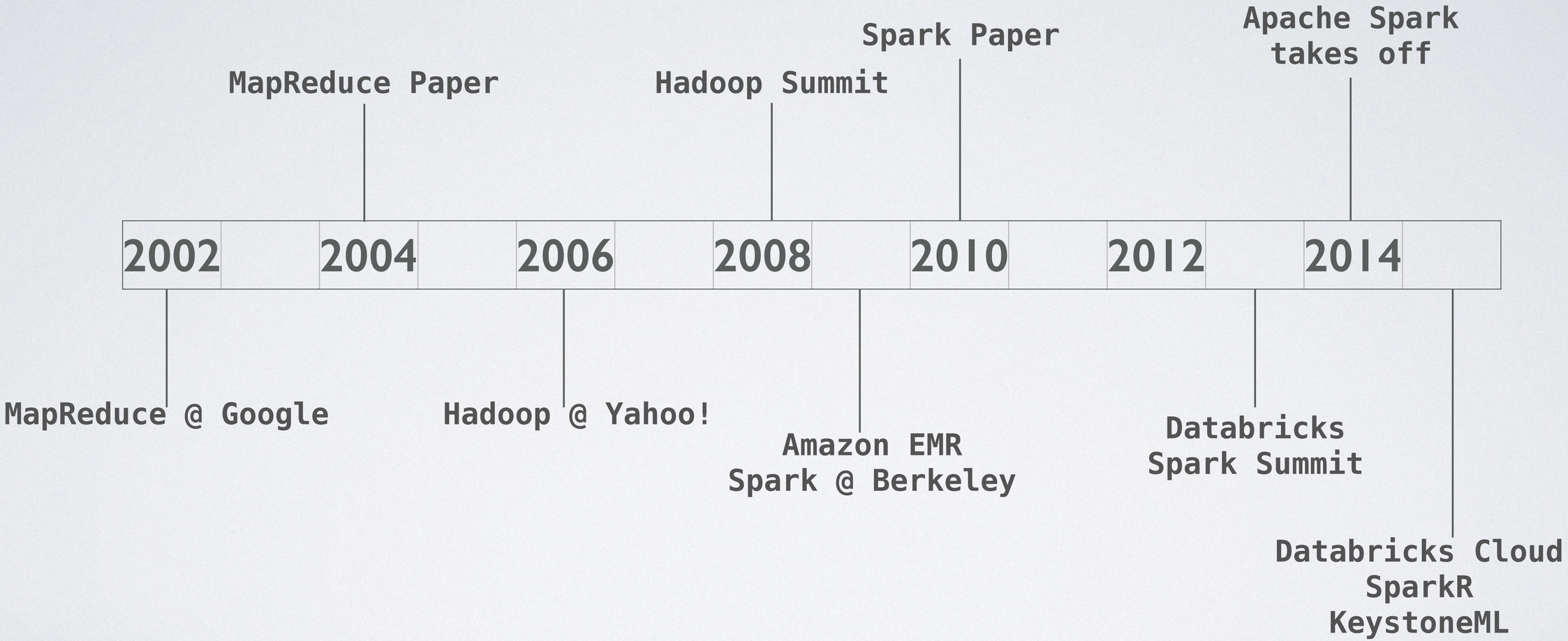
- Web, e-commerce, marketing, other data explosion
- Work no longer fits on a single machine
- Move to *horizontal scale-out* on clusters of commodity hardware
- Machine learning, indexing, graph processing use cases at scale

GAME CHANGE: C. 2002-2004

Google File System
research.google.com/archive/gfs.html

MapReduce: Simplified Data Processing on Large Clusters
research.google.com/archive/mapreduce.html

HISTORY: FUNCTIONAL PROGRAMMING FOR BIG DATA



c. 1979 - MIT, CMU, Stanford, etc.

LISP, Prolog, etc. operations: map, reduce, etc.

Slide adapted from *SparkCamp,* Strata Hadoop World, San Jose, CA, Feb 2015

MapReduce Limitations

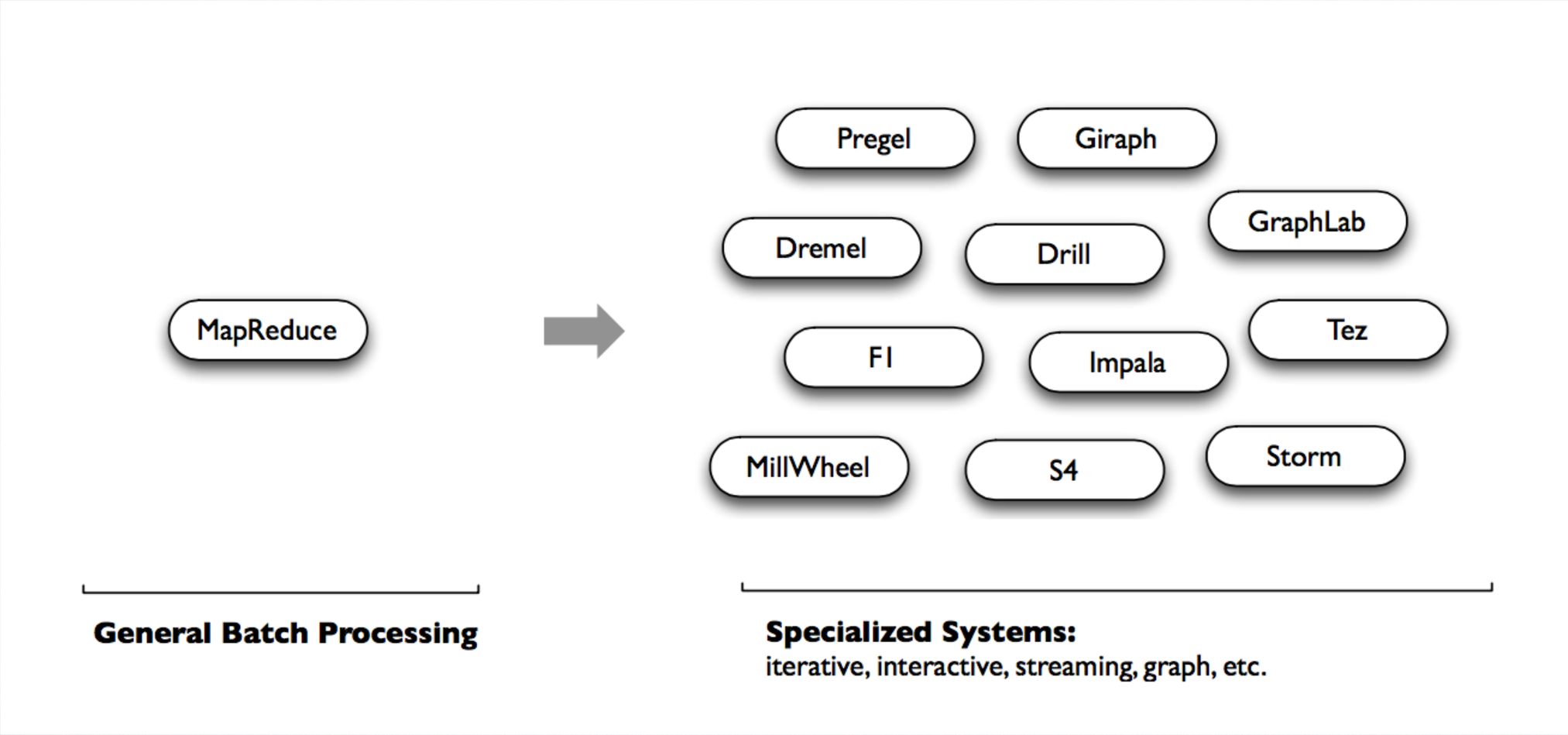
- Difficult to program directly in MR
- Performance bottlenecks, batch processing only
- Streaming, iterative, interactive, graph processing,…

MR doesn't fit modern use cases
Specialized systems developed as workarounds…

MapReduce Limitations

MR doesn't fit modern use cases

Specialized systems developed as workarounds



Slide adapted from *SparkCamp*, Strata Hadoop World, San Jose, CA, Feb 2015

PART II:

APACHE SPARK TO THE RESCUE…

Apache Spark

- Fast, unified, large-scale data processing engine for modern workflows
- Batch, streaming, iterative, interactive
- SQL, ML, graph processing

- Developed in ’09 at UC Berkeley AMPLab, open sourced in ’10
- Spark is one of the largest Big Data OSS projects

> *“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)

Apache Spark

Spark's goal was to *generalize* MapReduce, supporting modern use cases within *same engine*!

Spark Research

Spark: Cluster Computer with Working Sets

http://people.csail.mit.edu/matei/papers/2010/hotcloud_spark.pdf

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf

Spark: Key Points

- Same engine for batch, streaming and interactive workloads
- Scala, Java, Python, and (soon) R APIs
- Programming at a higher level of abstraction
- More general than MR

WordCount: "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));
```

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

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

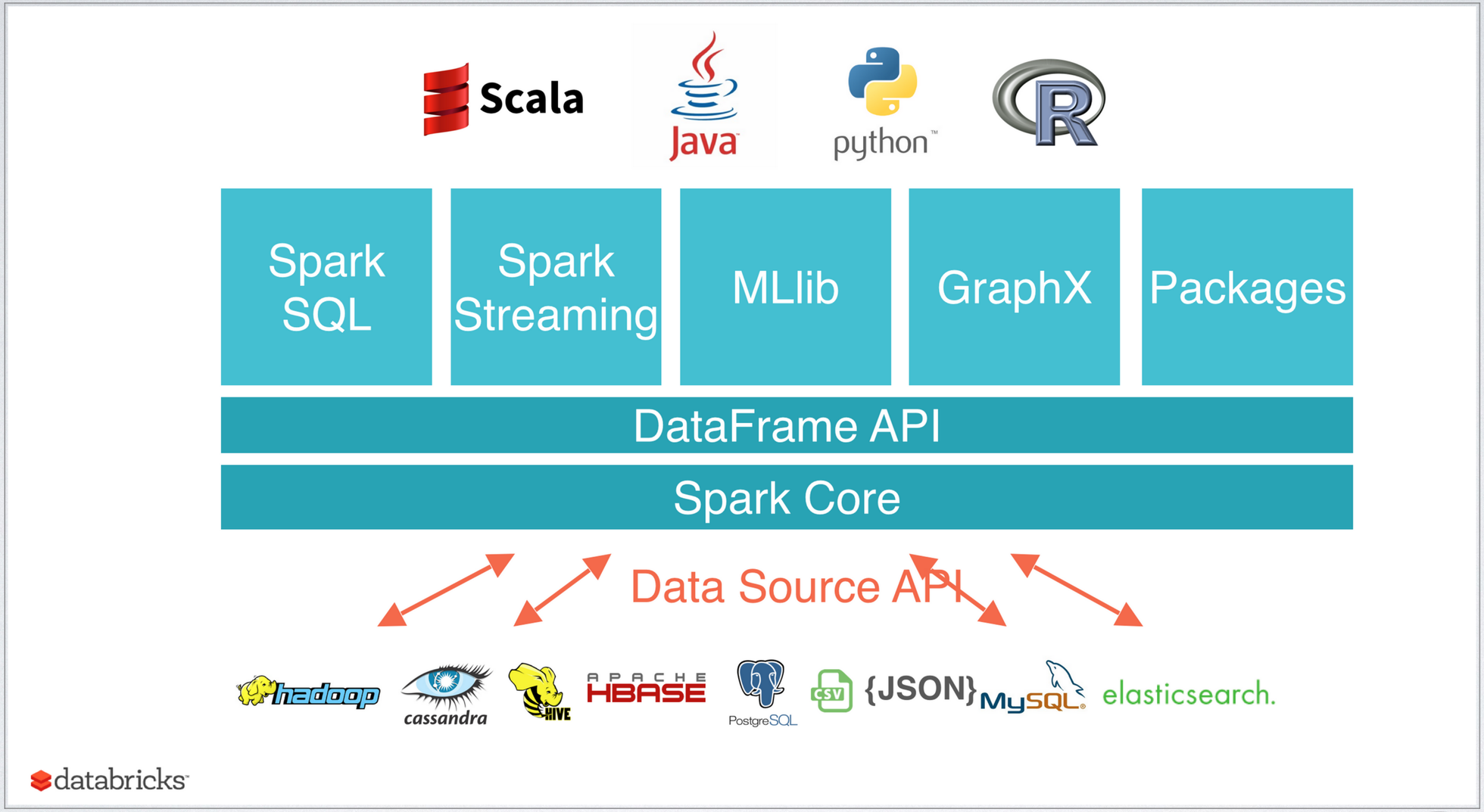
WordCount in 3 lines of Spark

WordCount in 50+ lines of Java MR

Spark vs. MapReduce

- Unified engine for modern workloads
- Lazy evaluation of the operator graph
- Optimized for modern hardware
- Functional programming / ease of use
- Reduction in cost to build/maintain enterprise apps
- Lower start up overhead
- More efficient shuffles

Spark in a nutshell…



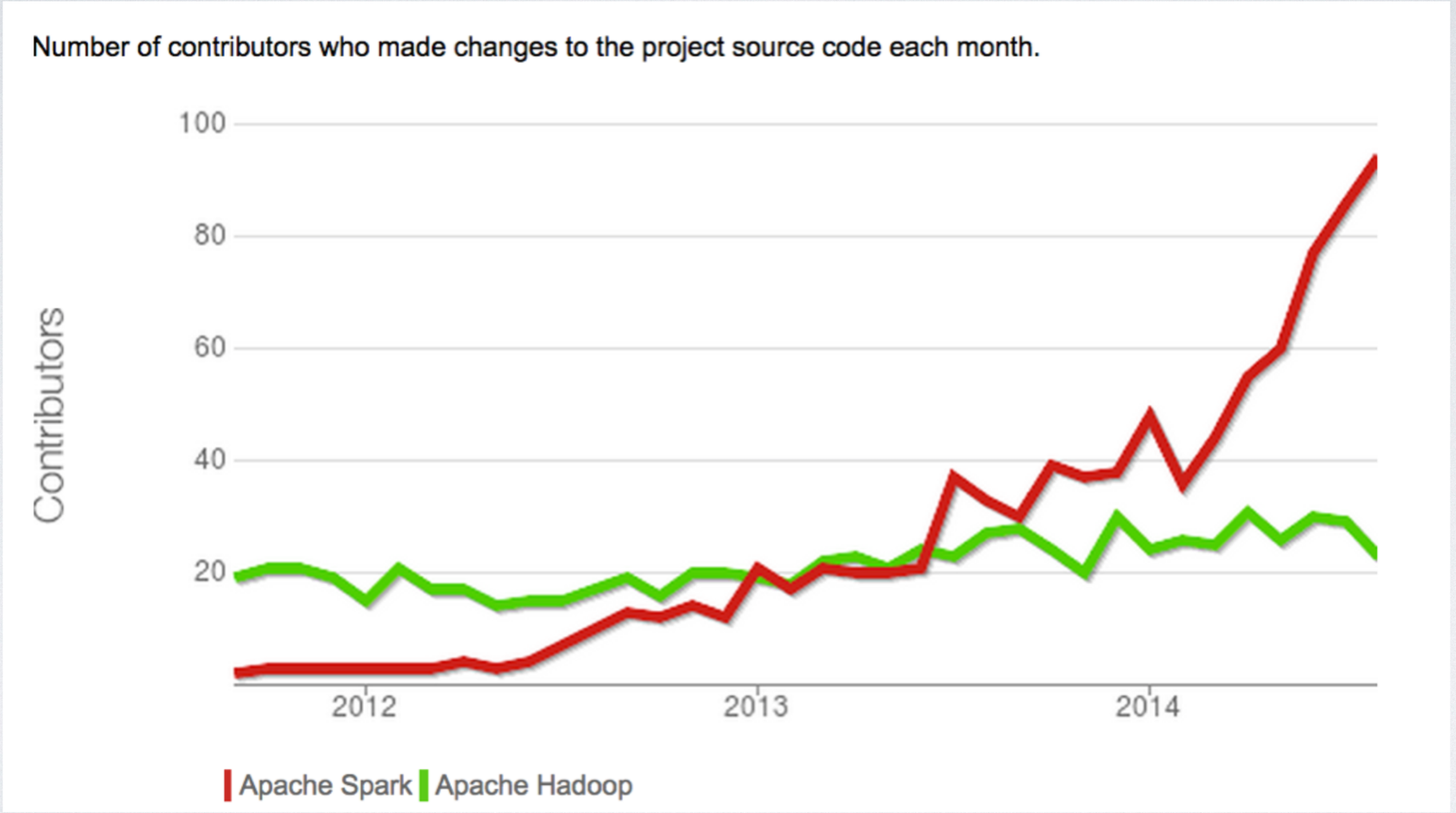
Spark Destroys Previous Sort Record

Spark: 3x faster with 10x fewer nodes

	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**

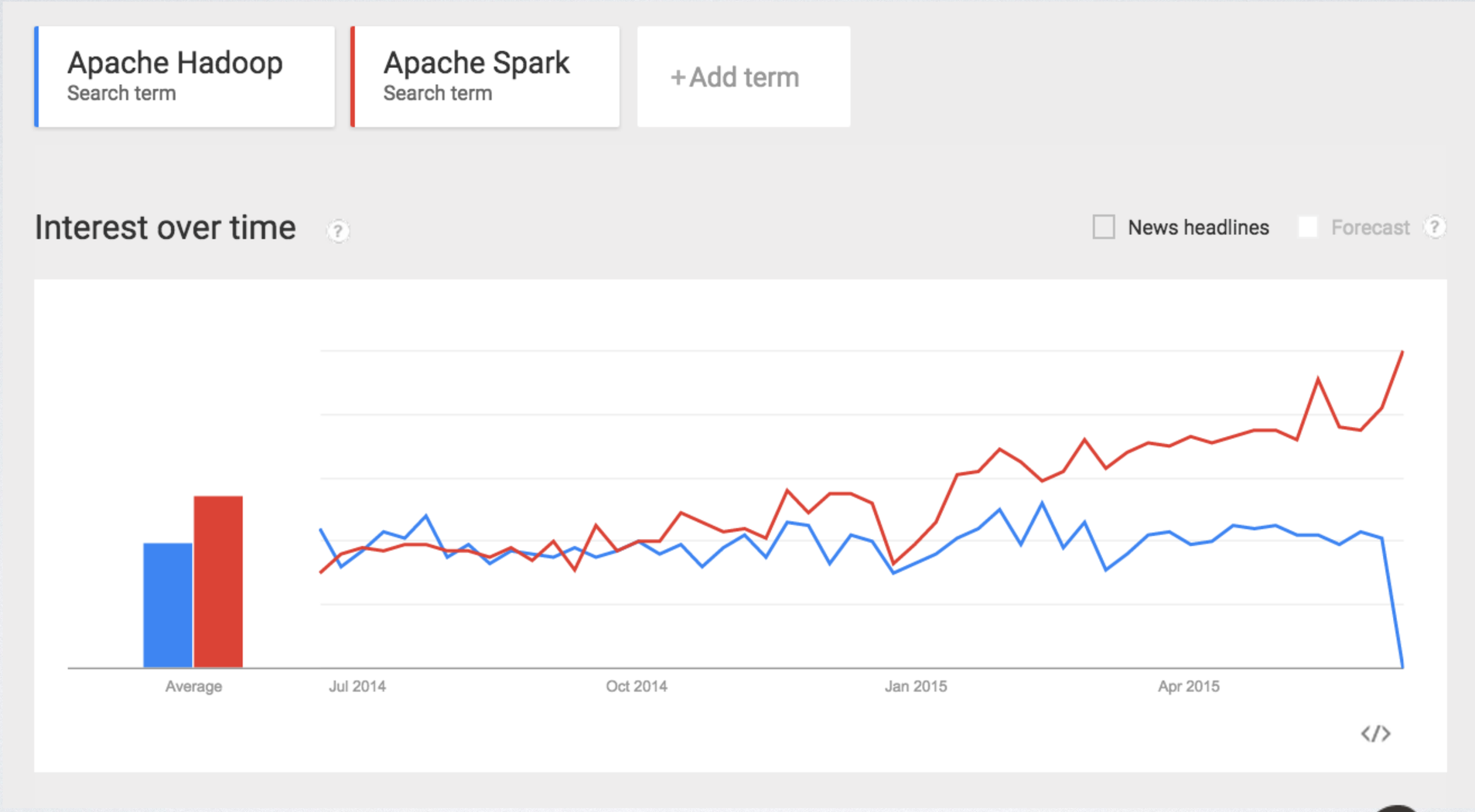
databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

Spark is one of the most active Big Data projects…

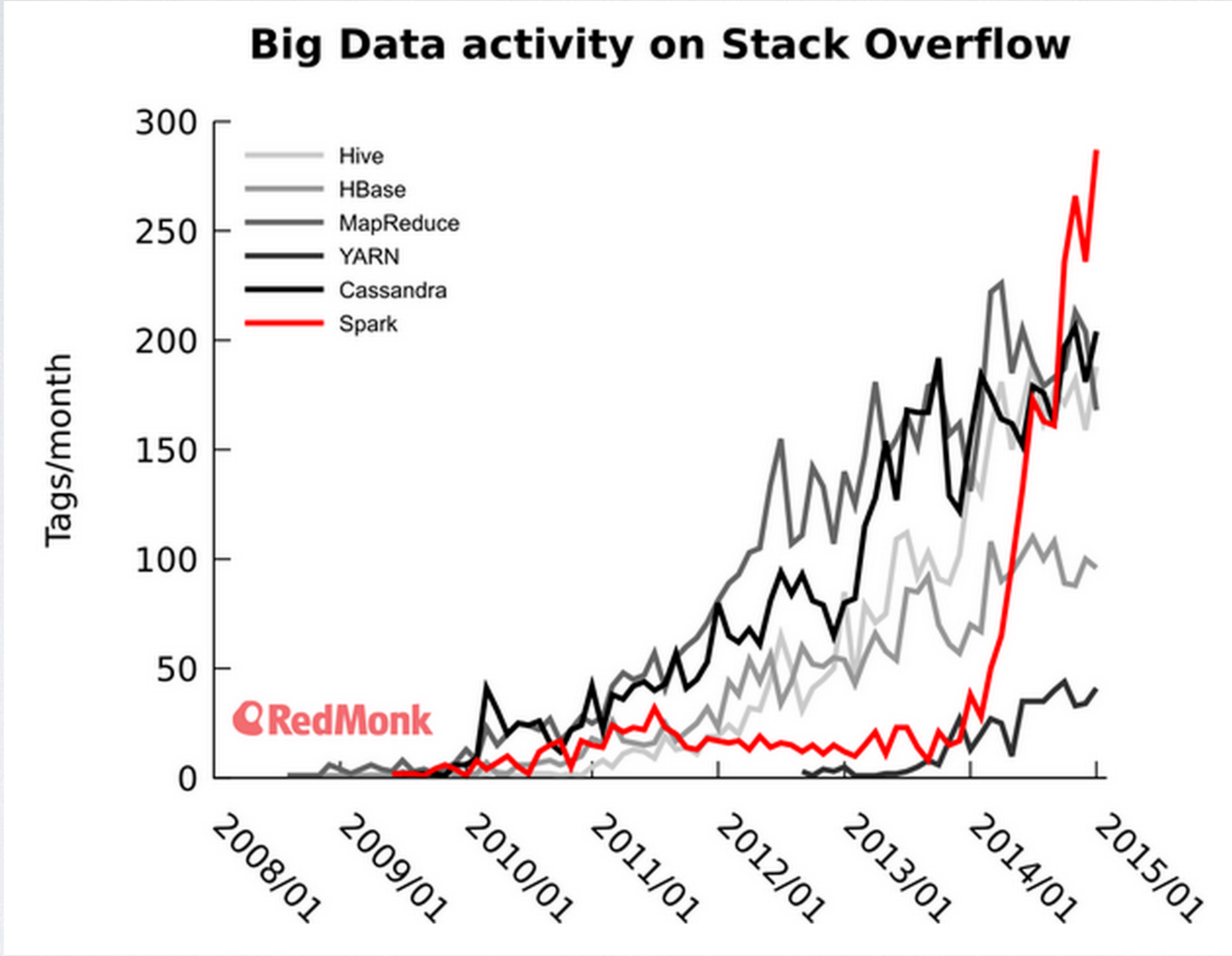


openhub.net/orgs/apache

What does Google say…

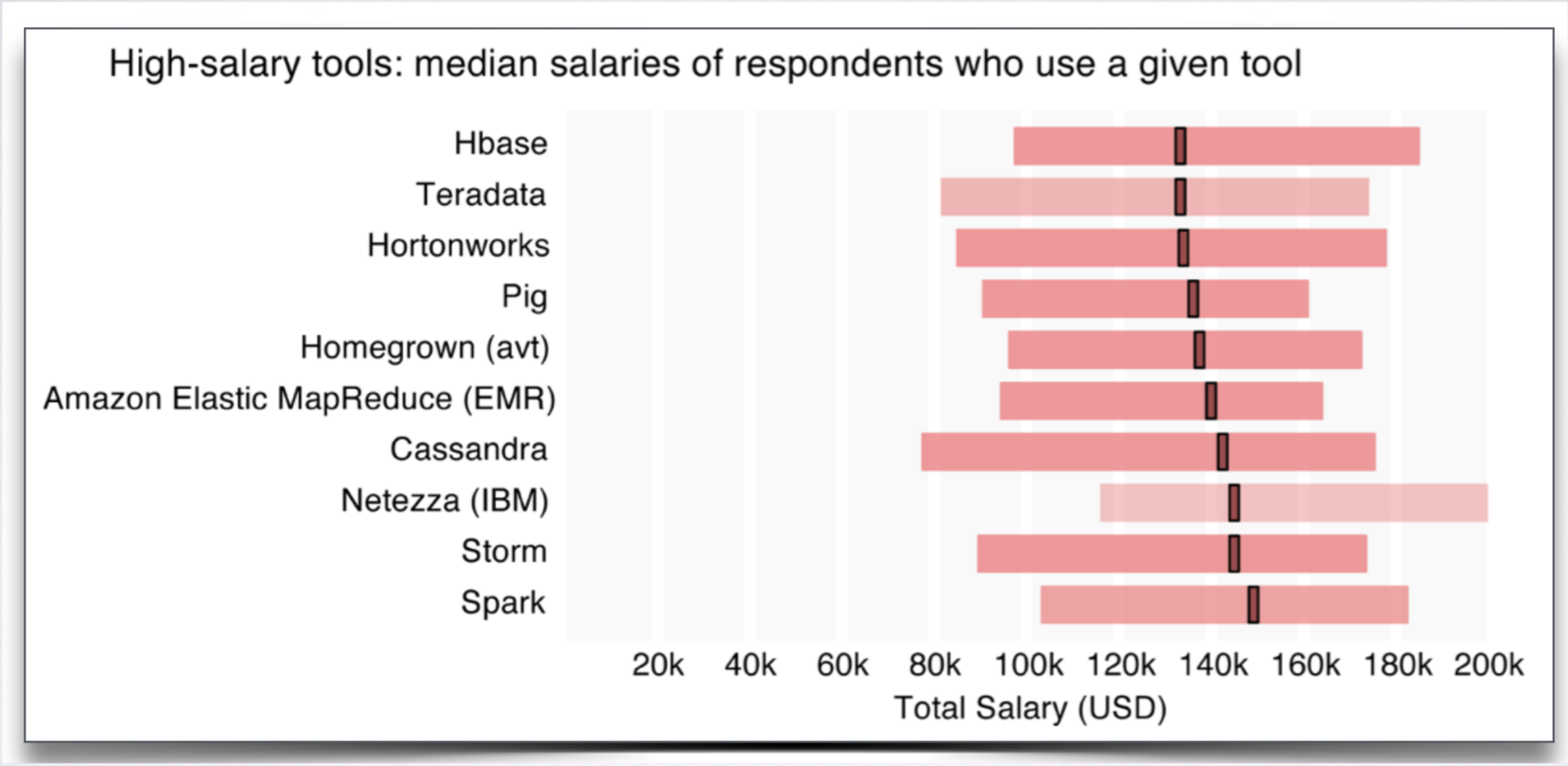


Spark on Stack Overflow



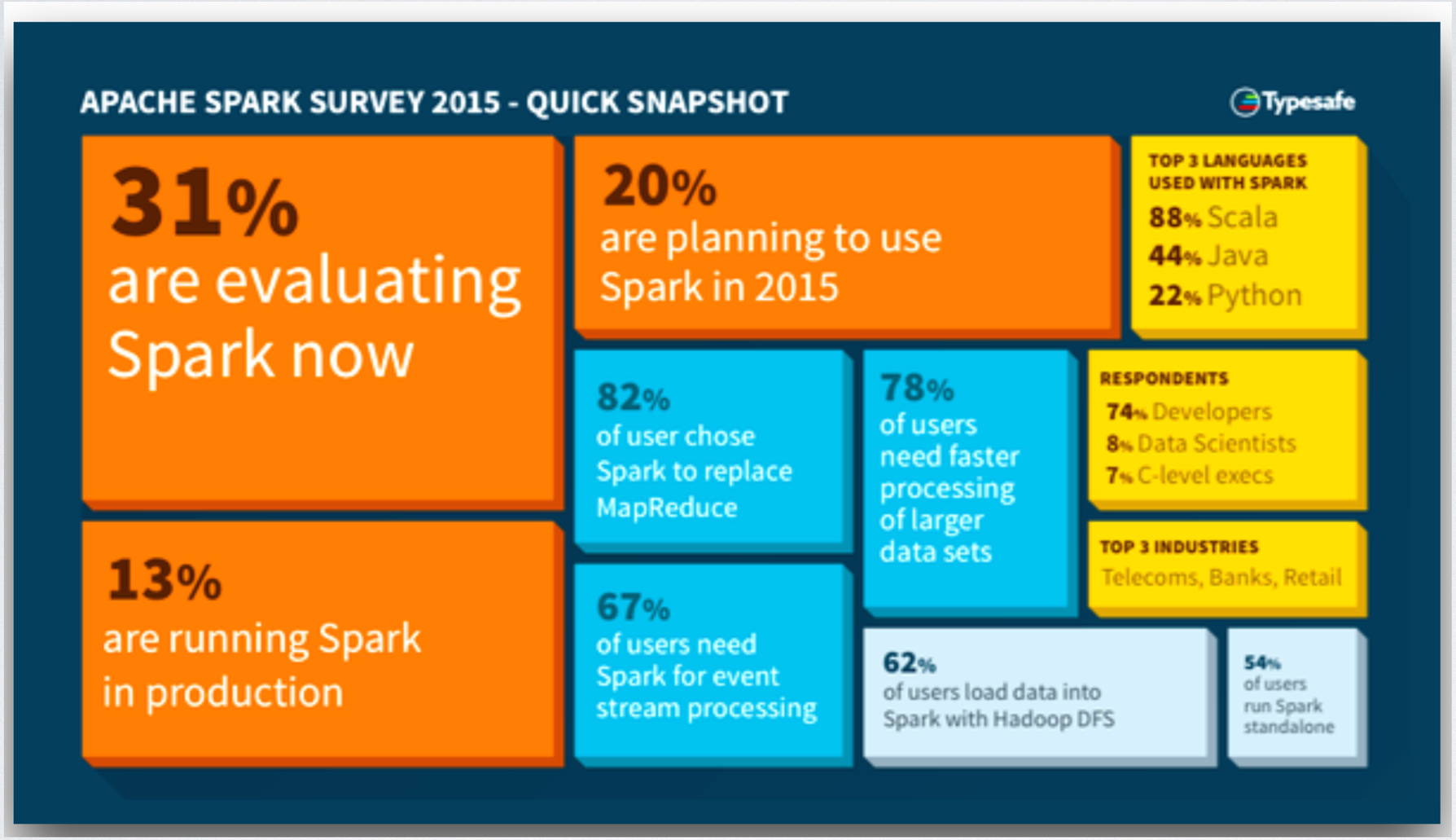
twitter.com/dberkholz/status/568561792751771648

It pays to Spark…



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

Spark Adoption



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

PART III:

APACHE SPARK CONCEPTS…

Resilient **D**istributed **D**atasets (RDDs)

- Spark's main abstraction - a fault-tolerant collection of elements that can be operated on in parallel

- Two ways to create RDDs:

  I. *Parallelized collections*

```
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]
```

  II. *External Datasets*

```
lines = sc.textFile("s3n://error-logs/error-log.txt") \
      .map(lambda x: x.split("\t"))
```

RDD Operations

- Two types: *transformations* and *actions*
- Transformations create a new RDD out of existing one, e.g. rdd.map(…)
- Actions return a value to the driver program after running a computation on the RDD, e.g., rdd.count()

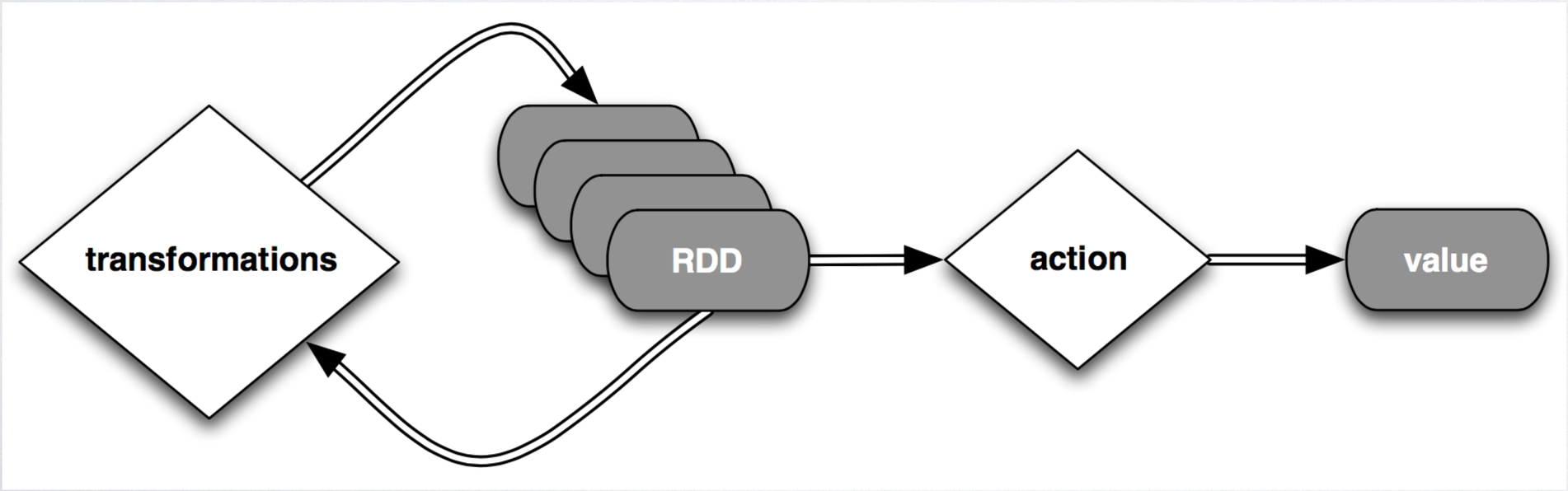


Figure from *SparkCamp*, Strata Hadoop World, San Jose, CA, Feb 2015

Transformations

Transformation	*Meaning*
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).
mapPartitions(*func*)	Similar to map, but runs separately on each partition (block) of the RDD.
mapPartitionsWithIndex(*func*)	Similar to mapPartitions, but also provides *func* with an integer value representing the index of the partition.
sample(*withReplacement, fraction, seed*)	Sample a fraction fraction of the data, with or without replacement, using a given random number generator seed.

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

Transformations

Transformation	*Meaning*
union(*otherDataset*)	Return a new dataset that contains the union of the elements in the source dataset and the argument.
intersection(*otherDataset*)	Return a new RDD that contains the intersection of elements in the source dataset and the argument.
distinct([*numTasks*]))	Return a new dataset that contains the distinct elements of the source dataset.
groupByKey([*numTasks*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<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 func.
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

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

Transformations

Transformation	*Meaning*
join(*otherDataset*, [*numTasks*])	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) 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, (Iterable<V>, Iterable<W>)) tuples.
cartesian(*otherDataset*)	When called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements).
pipe(*command*, [*envVars*])	Pipe each partition of the RDD through a shell *command*. RDD elements are written to the process's stdin and lines output to its stdout are returned as an RDD of strings.
coalesce(*numPartitions*)	Decrease the number of partitions in the RDD to *numPartitions*. Useful for running operations more efficiently after filtering down a large dataset.

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

Ex: Transformations

Python

```
>>> x = ['hello world', 'how are you enjoying the conference']

>>> rdd = sc.parallelize(x)

>>> rdd.filter(lambda x: 'hello' in x).collect()
['hello world']

>>> rdd.map(lambda x: x.split(" ")).collect()
[['hello', 'world'], ['how', 'are', 'you', 'enjoying', 'the', 'conference']]

>>> rdd.flatMap(lambda x: x.split(" ")).collect()
['hello', 'world', 'how', 'are', 'you', 'enjoying', 'the', 'conference']
```

Ex: Transformations

Scala

```
scala> val x = Array(“hello world”, “how are you enjoying the conference”)

scala> val rdd = sc.parallelize(x)

scala> rdd.filter(x => x contains "hello").collect()
res15: Array[String] = Array(hello world)


scala> rdd.map(x => x.split(" ")).collect()
res19: Array[Array[String]] = Array(Array(hello, world),
        Array(how, are, you, enjoying, the, conference))


 scala> rdd.flatMap(x => x.split(" ")).collect()
 res20: Array[String] = Array(hello, world, how, are, you, enjoying,
           the, conference)
```

Actions

Action	*Meaning*
reduce(*func*)	Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one), *func* should be commutative and associative so it can be computed correctly in parallel.
collect()	Return all elements of the dataset as array at the driver program. Usually useful after a filter or other operation that returns sufficiently small 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.
takeSample(*withReplacement*, *num*, [*seed*])	Return an array with a random sample of *num* elements of the dataset, with or without replacement, with optional random number generator *seed*.
takeOrdered(*n*, [*ordering*])	Return the first n elements of the RDD using either their natural order or a custom comparator.

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

Actions

Action	*Meaning*
saveAsTextFile(*path*)	Write the dataset as a text file (or set of text files) in a given *path* 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*) (Java and Scala)	Write the dataset as a Hadoop SequenceFile in a given *path* in the local filesystem, HDFS or any other Hadoop-supported file system.
saveAsObjectFile(*path*) (Java and Scala)	Write the dataset in a simple format using Java serialization, which can then be loaded using SparkContext.objectFile().
countByKey()	For RDD of type (K, V), returns a hashmap of (K, Int) pairs with the count of each key.
foreach(*func*)	Run a function *func* on each element of the dataset. This is usually done for side effects such as updating an accumulator variable or interacting with external storage systems.

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

Ex: Actions

Python

```
>>> from operator import add

>>> x = [“hello world", "hello there", "hello again”]

>>> rdd = sc.parallelize(x)

>>> wordsCounts = rdd.flatMap(lamdba x: x.split(" “)).map(lambda w: (w, 1)) \
                     .reduceByKey(add)

>>> wordCounts.collect()
[(again,1), (hello,3), (world,1), (there,1)]

>>> wordCounts.saveAsTextFile("/Users/vb/wordcounts")
```

Ex: Actions

Scala

```
scala> val x = Array("hello world", "hello there", "hello again")

scala> val rdd = sc.parallelize(x)

scala> val wordsCounts = rdd.flatMap(x => x.split(" ")).map(word => (word, 1)) \
                            .reduceByKey(_ + _)


scala> wordCounts.collect()
res43: Array[(String, Int)] = Array((again,1), (hello,3), (world,1), (there,1))


scala> wordCounts.saveAsTextFile("/Users/vb/wordcounts")
```

RDD Persistence

- Unlike MapReduce, Spark can *persist* (or *cache*) a dataset in memory across operations

- Each node stores any partitions of it that it computes in memory and reuses them in other transformations/actions on that RDD

- 10x increase in speed

- One of the most important Spark features

```
>>> wordCounts = rdd.flatMap(lamdba x: x.split(“ “)) \
                    .map(lambda w: (w, 1)) \
                    .reduceByKey(add) \
                    .cache()
```

RDD Persistence Storage Levels

Storage Level	*Meaning*
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.

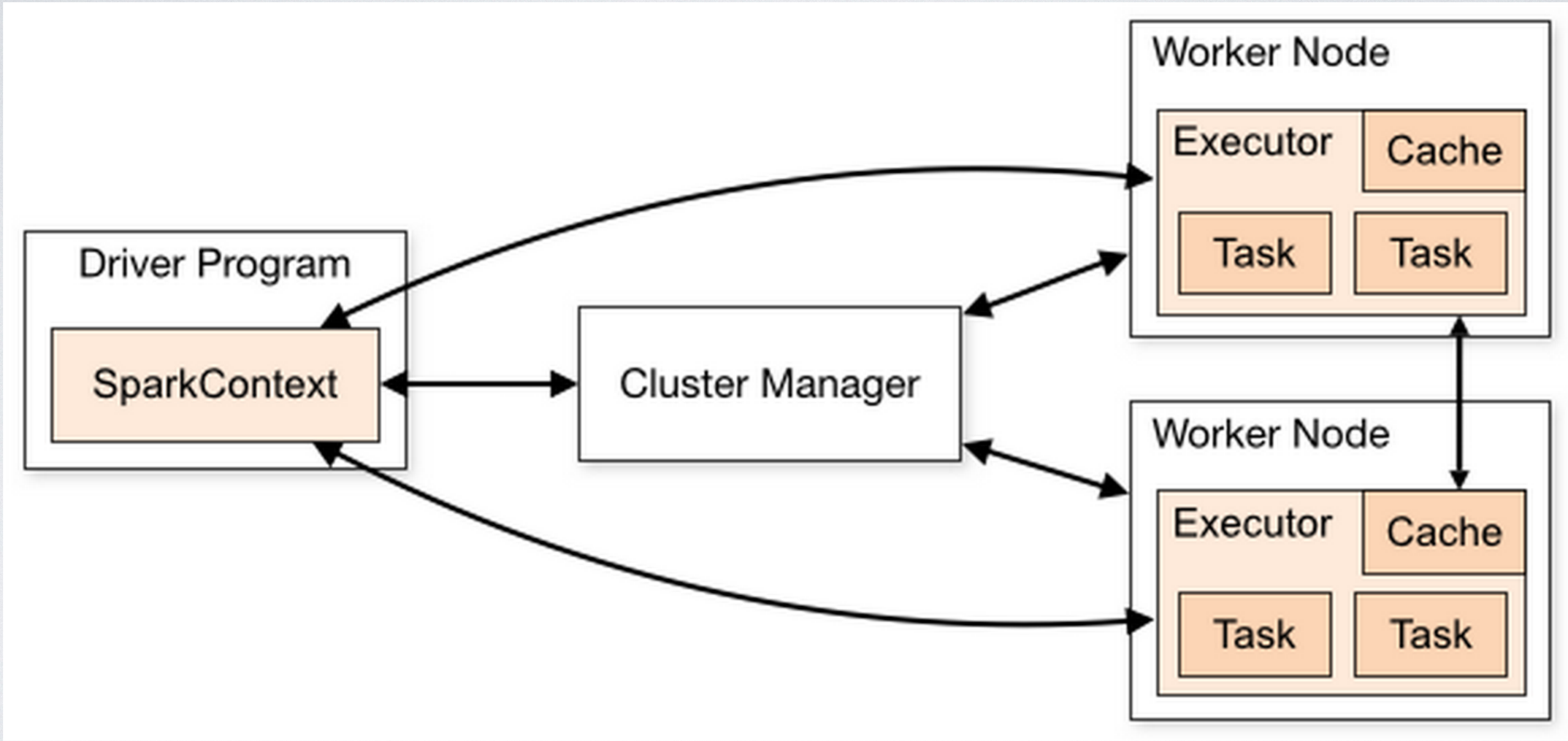
http://spark.apache.org/docs/latest/programming-guide.html

More RDD Persistence Storage Levels

Storage Level	*Meaning*
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 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. Compared to MEMORY_ONLY_SER, OFF_HEAP reduces garbage collection overhead and allows executors to be smaller and to share a pool of memory, making it attractive in environments with large heaps or multiple concurrent applications.

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

Under the hood…



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

And so much more…

- DataFrames and SQL
- Spark Streaming
- MLlib
- GraphX

spark.apache.org/docs/latest/