Introduction to Scala and Spark

SATURN 2016

Bradley (Brad) S. Rubin, PhD
Director, Center of Excellence for Big Data
Graduate Programs in Software
University of St. Thomas, St. Paul, MN
bsrubin@stthomas.edu

Scala

Spark

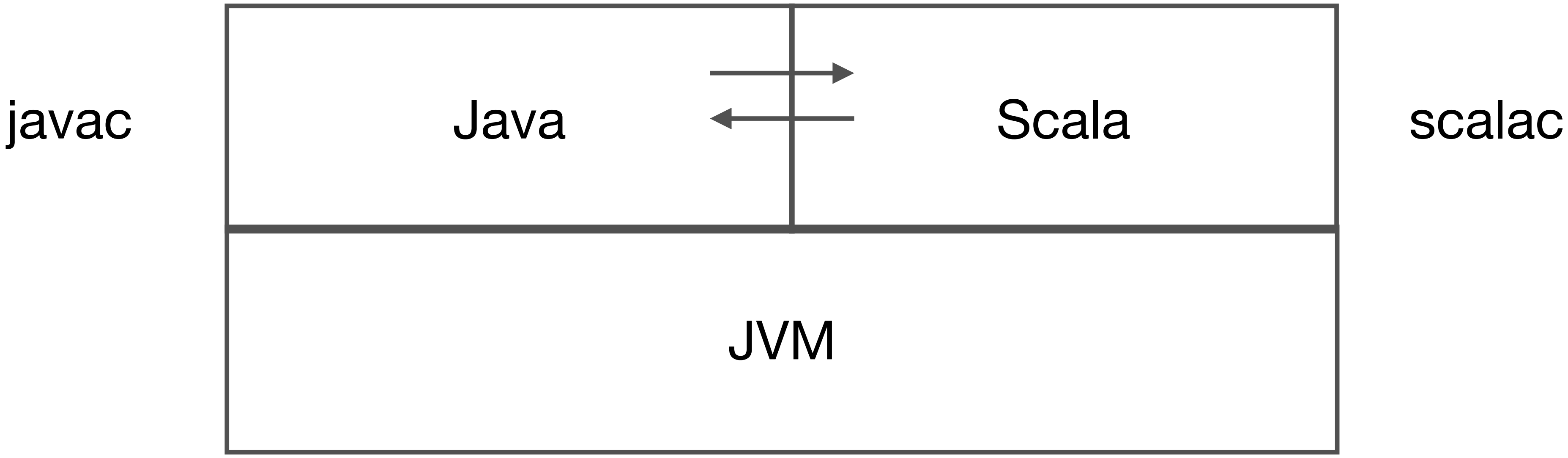
Scala/Spark Examples

Classroom Experience

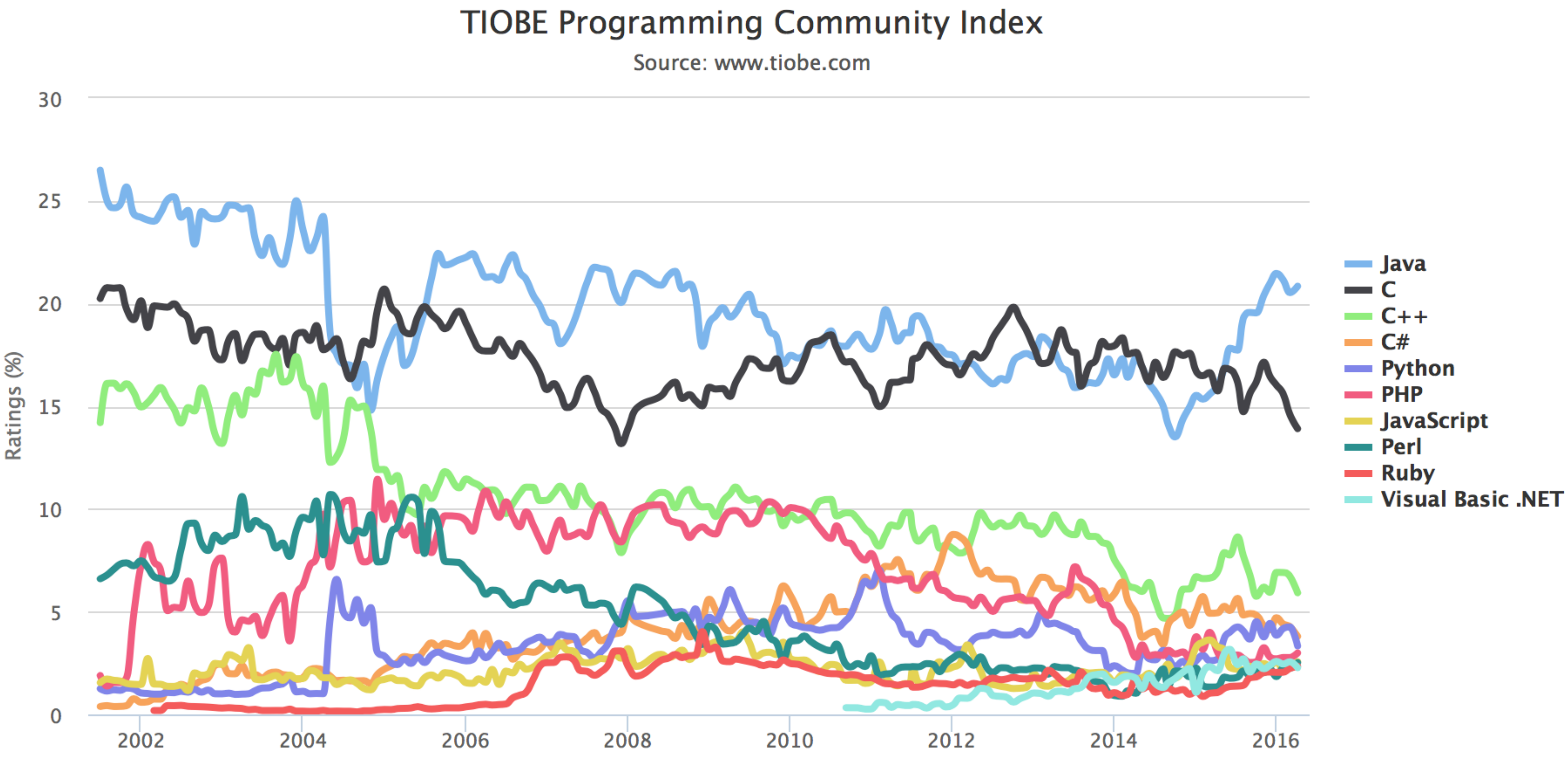
What is Scala?

- JVM-based language that can call, and be called, by Java
  - New: Scala.js (Scala to JavaScript compiler)
  - Dead: Scala.Net
- A more concise, richer, Java + functional programming
- Blends the object-oriented and functional paradigms
- Strongly statically typed, yet feels dynamically typed
- Stands for SCAlable LAnguage
  - Little scripts to big projects, multiple programming paradigms, start small and grow knowledge as needed, multi-core, big data
- Developed by Martin Odersky at EPFL (Switzerland)
  - Worked on Java Generics and wrote javac
- Released in 2004

Scala and Java

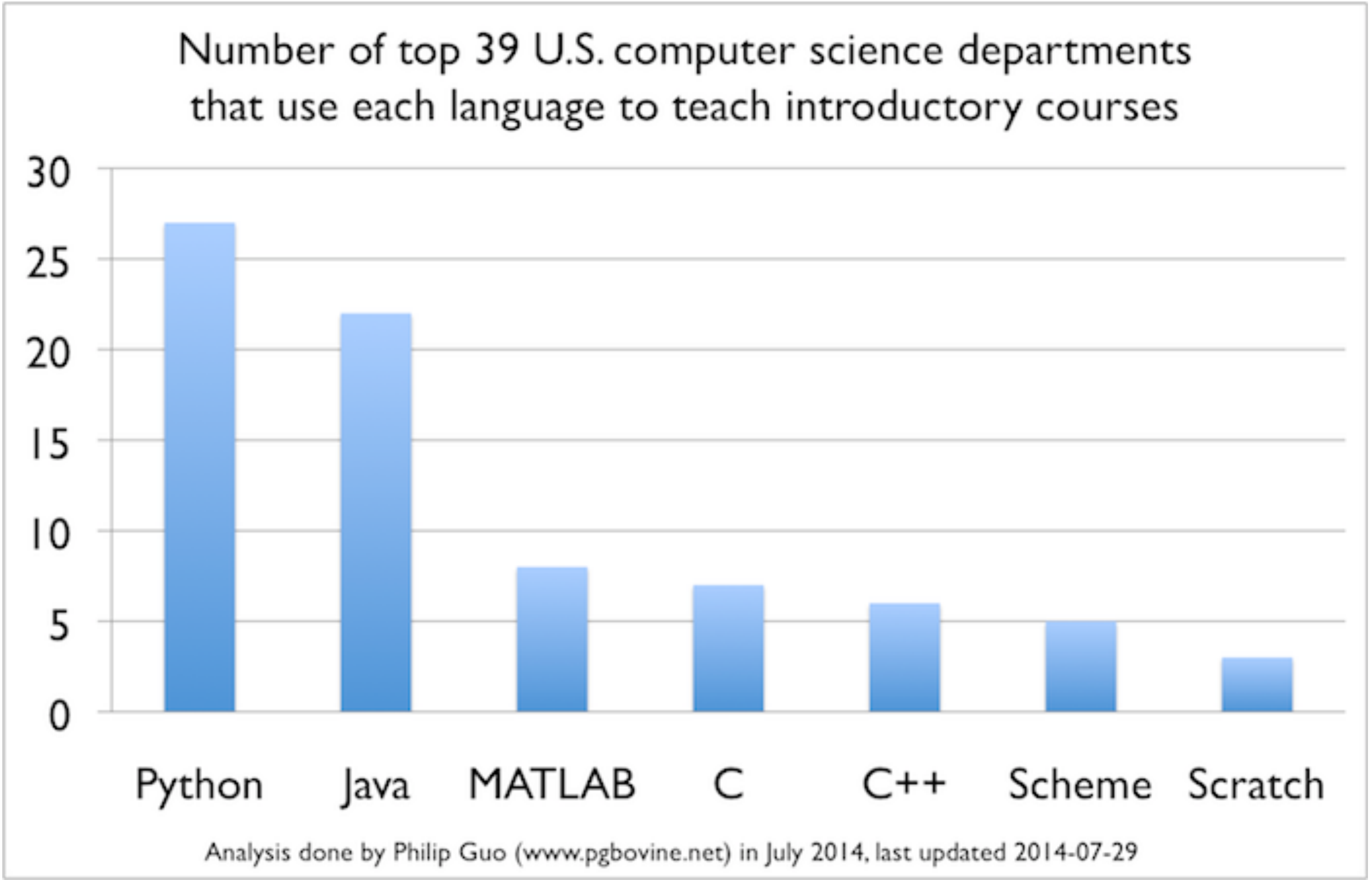


Scala Adoption (TIOBE)



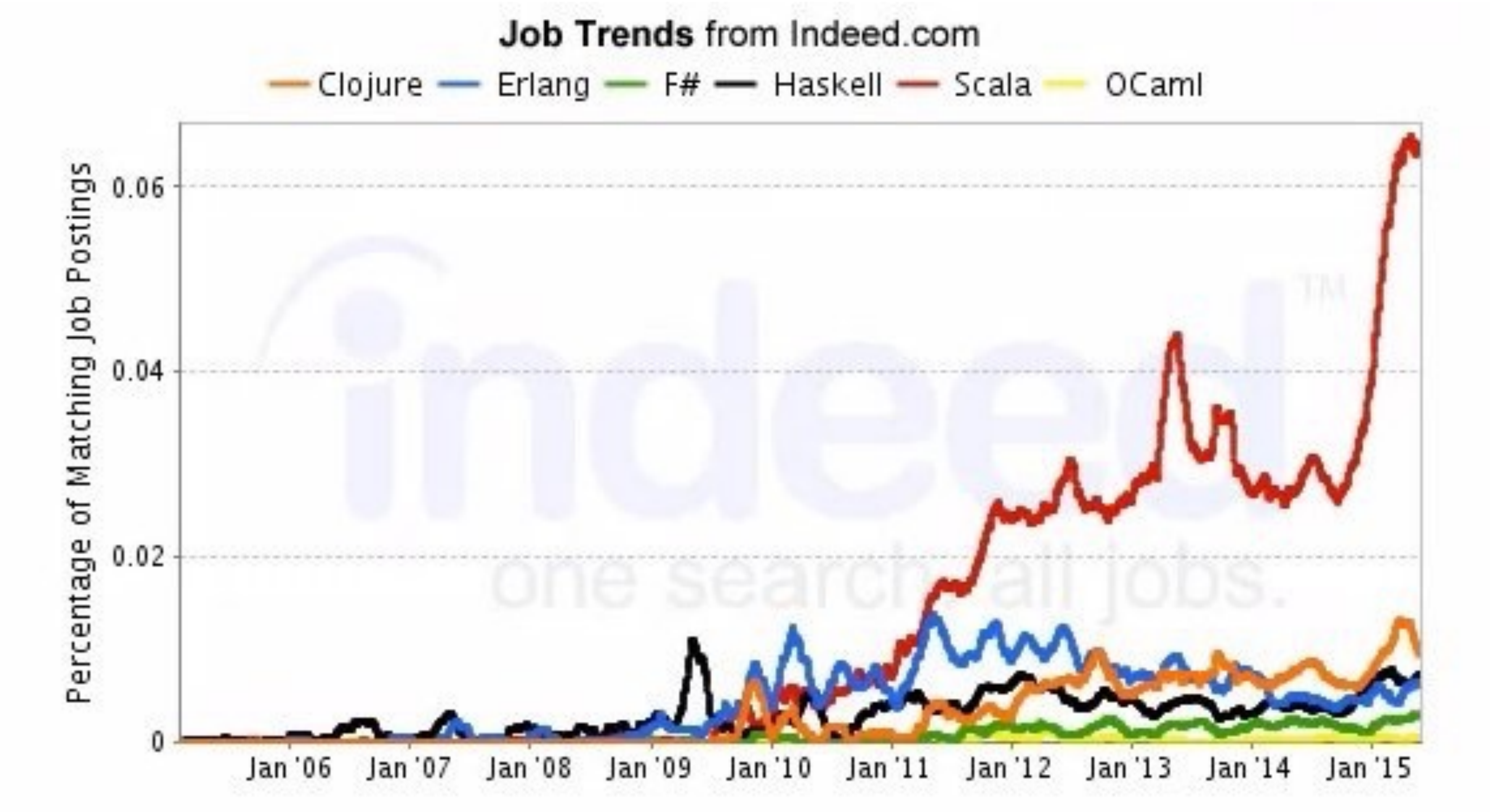
Scala is 31st on the list

Freshman Computer Science



Analysis done by Philip Guo (www.pgbovine.net) in July 2014, last updated 2014-07-29

Job Demand
Functional Languages



Scala Sampler
Syntax and Features

- Encourages the use of immutable state
- No semicolons

  unless multiple statements per line
- No need to specify types in all cases

  types follow variable and parameter names after a colon
- Almost everything is an expression that returns a value of a type
- Discourages using the keyword return
- Traits, which are more powerful Interfaces
- Case classes auto-generate a lot of boilerplate code
- Leverages powerful pattern matching

Scala Sampler
Syntax and Features

- Discourages null by emphasizing the Option pattern
- Unit, like Java void
- Extremely powerful (and complicated) type system
- Implicitly converts types, and lets you extend closed classes
- No checked exceptions
- Default, named, and variable parameters
- Mandatory override declarations
- A pure OO language

    all values are objects, all operations are methods

Language Opinions

There are only two kinds of languages:
the ones people complain about and the ones nobody uses.

— Bjarne Stroustrup

I Like…

- Concise, lightweight feel
- Strong, yet flexible, static typing
- Strong functional programming support
- Bridge to Java and its vast libraries
- Very powerful language constructs, if you need them
- Strong tool support (IntelliJ, Eclipse, Scalatest, etc)
- Good books and online resources

I Don’t Like…

- Big language, with a moderately big learning curve
- More than one way to do things
- Not a top 10 language
- Not taught to computer science freshman

Java 8:
Threat or Opportunity?

- Java 8 supports more functional features, like lambda expressions (anonymous functions), encroaching on Scala's space
- Yet Scala remains more powerful and concise
- The Java 8 JVM offers Scala better performance

    Release 2.12 will support this

- My prediction: Java 8 will draw more attention to functional programming, and drive more Scala interest
- I don't know any Scala programmers who have gone back to Java (willingly)

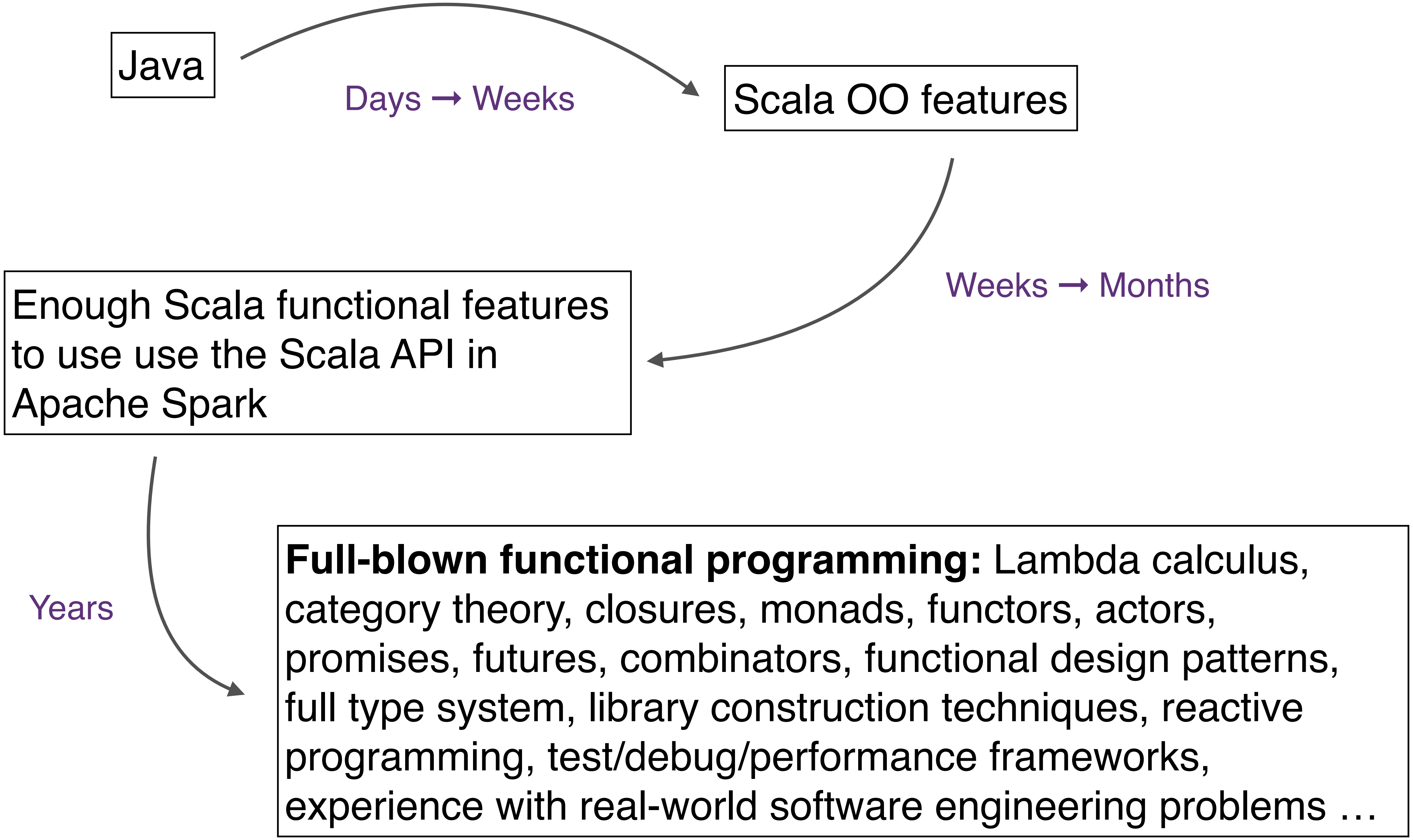
Scala Ecosystem

- Full **Eclipse/IntelliJ** support
- **REPL** Read Evaluate Print Loop interactive shell
- **Scala Worksheet** interactive notebook
- **ScalaTest** unit test framework
- **ScalaCheck** property-based test framework
- **Scalastyle** style checking
- **sbt** Scala build tool
- **Scala.js** Scala to JavaScript compiler

Functional Programming and Big Data

- Big data architectures leverage parallel disk, memory, and CPU resources in computing clusters
- Often, operations consist of independently parallel operations that have the shape of the map operator in functional programming
- At some point, these parallel pieces must be brought together to summarize computations, and these operations have the shape of aggregation operators in functional programming
- The functional programming paradigm is a great fit with big data architectures

The Scala Journey



Java

Days ➞ Weeks

Scala OO features

Weeks ➞ Months

Enough Scala functional features to use use the Scala API in Apache Spark

Years

Full-blown functional programming: Lambda calculus, category theory, closures, monads, functors, actors, promises, futures, combinators, functional design patterns, full type system, library construction techniques, reactive programming, test/debug/performance frameworks, experience with real-world software engineering problems …

Scala

Spark

Scala/Spark Examples

Classroom Experience

Apache Spark

- Apache Spark is an in-memory big data platform that performs especially well with iterative algorithms
- 10-100x speedup over Hadoop with some algorithms, especially iterative ones as found in machine learning
- Originally developed by UC Berkeley starting in 2009

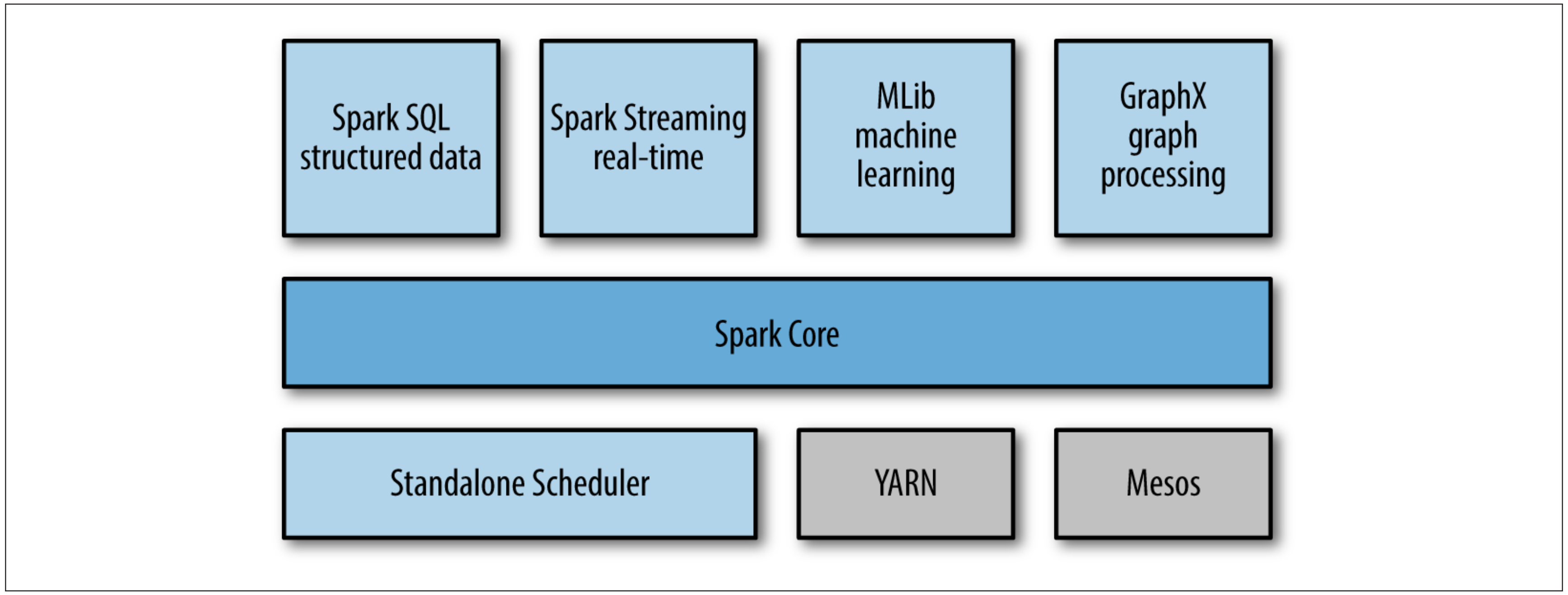
  Moved to an Apache project in 2013
- Spark itself is written in Scala, and Spark jobs can be written in Scala, Python, and Java (and more recently R and SparkSQL)
- Other libraries (Streaming, Machine Learning, Graph Processing)
- Percent of Spark programmers who use each language

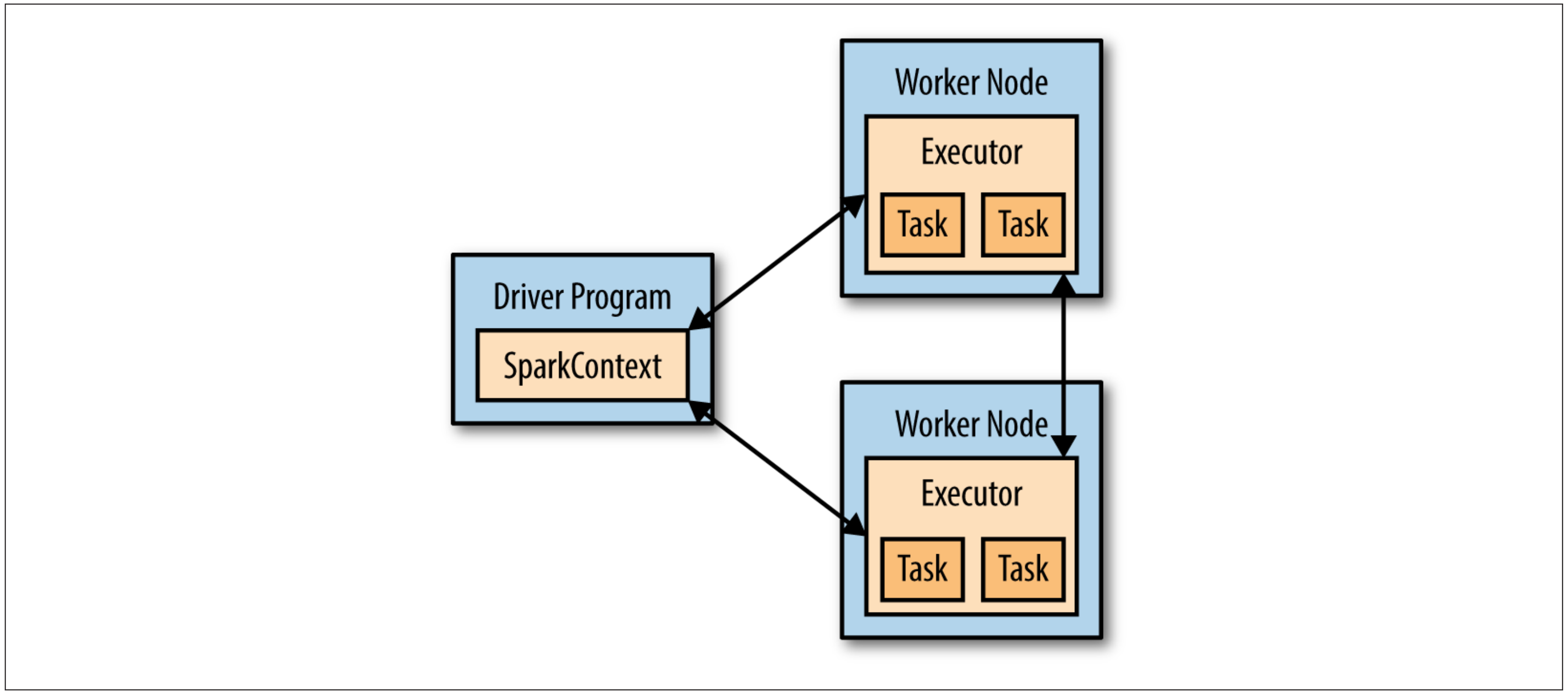
  88% Scala, 44% Java, 22% Python

  Note: This survey was done a year ago. I think if it were done today, we would see the rank as Scala, Python, and Java

Source: Cloudera/Typesafe

Spark Architecture [KARA15]





Basic Programming Model

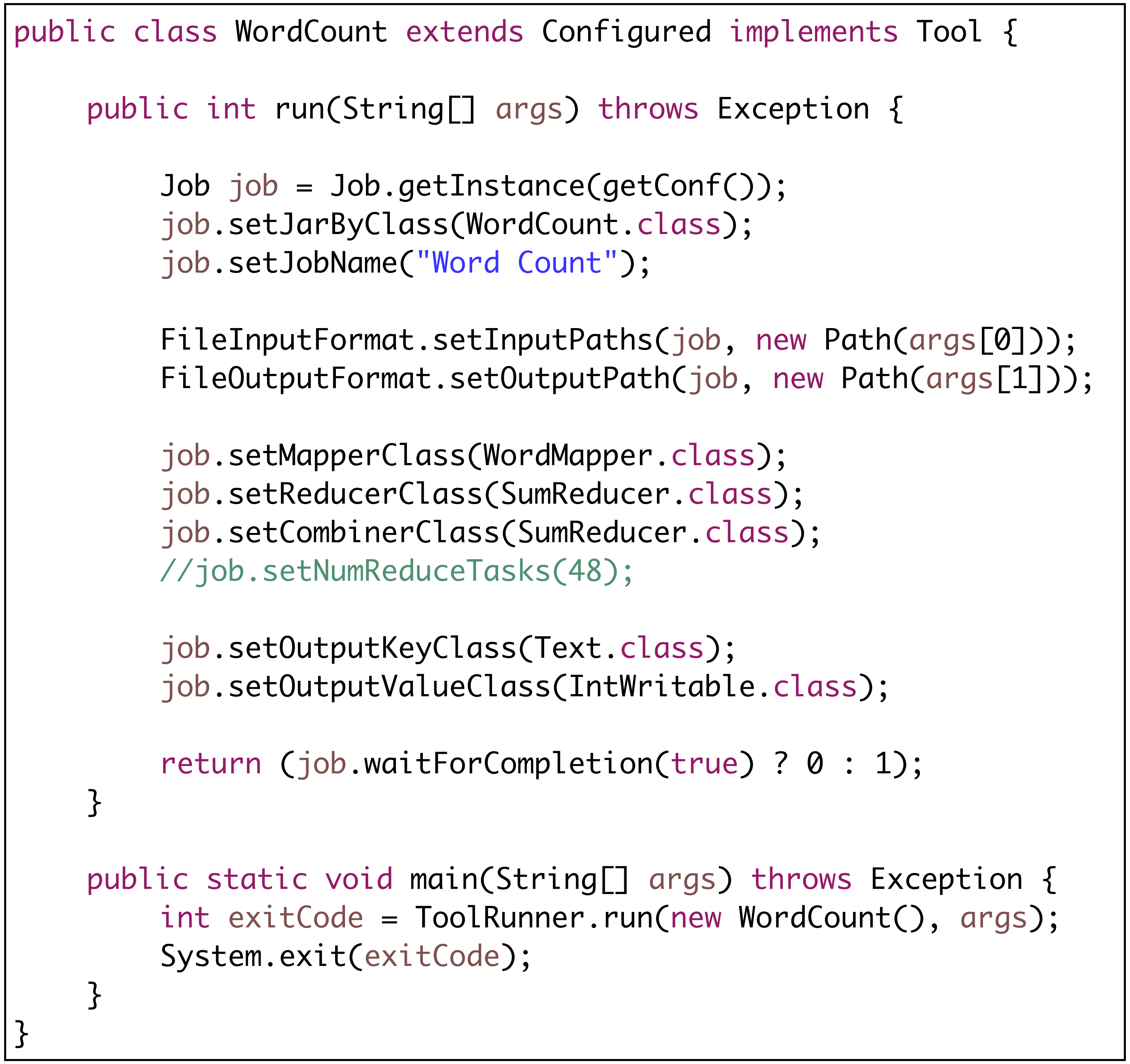
- Spark's data model is called a Resilient Distributed Dataset (RDD)
- Two operations
  - **Transformations**: Transform an RDD into another RDD (i.e. Map)
  - **Actions**: Process an RDD into a result (i.e. Reduce)
- Transformations are lazily processed, only upon an action
- Transformations might trigger an RDD repartitioning, called a **shuffle**
- Intermediate results can be manually cached in memory/on disk
- Spill to disk can be handled automatically
- Application hierarchy
  - An **application** consists of 1 or more **jobs** (an action ends a job)
  - A **job** consists of 1 or more **stages** (a shuffle ends a stage)
  - A **stage** consists of 1 or more **tasks** (tasks execute parallel computations)

Wordcount in Java MapReduce (1/2)

```java
public class WordMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    IntWritable intWritable = new IntWritable(1);
    Text text = new Text();
    @Override
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        for (String word : line.split("\\W+")) {
            if (word.length() > 0) {
                text.set(word);
                context.write(text, intWritable);
            }}}}
```

```java
public class SumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    IntWritable intWritable = new IntWritable();
    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {
        int wordCount = 0;
        for (IntWritable value : values) {
            wordCount += value.get();
        }
        intWritable.set(wordCount);
        context.write(key, intWritable);
    }}
```

Wordcount in Java MapReduce (2/2)



```java
public class WordCount extends Configured implements Tool {

    public int run(String[] args) throws Exception {

        Job job = Job.getInstance(getConf());
        job.setJarByClass(WordCount.class);
        job.setJobName("Word Count");

        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.setMapperClass(WordMapper.class);
        job.setReducerClass(SumReducer.class);
        job.setCombinerClass(SumReducer.class);
        //job.setNumReduceTasks(48);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        return (job.waitForCompletion(true) ? 0 : 1);
    }

    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new WordCount(), args);
        System.exit(exitCode);
    }
}
```

Wordcount in Java

```
JavaRDD<String> file = spark.textFile(“hdfs://...");

JavaRDD<String> words = file.flatMap(new FlatMapFunction<String, String>() {
  public Iterable<String> call(String s) { return Arrays.asList(s.split(" ")); }
});

JavaPairRDD<String, Integer> pairs = words.map(new PairFunction<String, String, Integer>() {
  public Tuple2<String, Integer> call(String s) { return new Tuple2<String, Integer>(s, 1); }
});

JavaPairRDD<String, Integer> counts = pairs.reduceByKey(new Function2<Integer, Integer>() {
  public Integer call(Integer a, Integer b) { return a + b; }
});

counts.saveAsTextFile("hdfs://...");
```

Java 7

```
JavaRDD<String> lines = sc.textFile(“hdfs://…”);
JavaRDD<String> words =
    lines.flatMap(line -> Arrays.asList(line.split(" ")));
JavaPairRDD<String, Integer> counts =
    words.mapToPair(w -> new Tuple2<String, Integer>(w, 1))
         .reduceByKey((x, y) -> x + y);
counts.saveAsTextFile(“hdfs://…”);
```

Java 8

Wordcount in Python

```
file = spark.textFile("hdfs://...")
counts = file.flatMap(lambda line: line.split(" ")) \
                .map(lambda word: (word, 1)) \
                .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

Wordcount in Scala

```
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

Spark Shells

- A shell is a kind of REPL (Run Evaluate Print Loop), commonly found in several languages to support interactive development
- Python is supported via “pyspark” and iPython notebooks
- Scala is supported via “spark-shell”
- Let’s look at an example of interactive development using the Spark Scala shell

Scala

Spark

Scala/Spark Examples

Classroom Experience

Reading in the Data

```
scala> sc.textFile(“/SEIS736/TFIDFsmall”)

res0: org.apache.spark.rdd.RDD[String] =
 /SEIS736/TFIDFsmall MapPartitionsRDD[1] at textFile at <console>:22
```

- We created an RDD out of the input files, but nothing really happens until we do an action, so let’s call collect(), which gathers all the distributed pieces of the RDD and brings them together in our memory (dangerous for large amounts of data)

```
scala> sc.textFile(“/SEIS736/TFIDFsmall”).collect

res1: Array[String] = Array(The quick brown fox jumps over the lazy brown dog.,
 Waltz, nymph, for quick jigs vex Bud.,
 How quickly daft jumping zebras vex.)
```

Getting the Words

- Next, we want to split out the words. To do this, let’s try the map function, which says to consider each item in the RDD array (a line) and transform it to the line split into words with W+
- We read the map as “for each input x, replace it with x split into an array of words”, where x is just a dummy variable
- Note, however, that we end up with an array of arrays of words (one array for each input file)
- To flatten this into just a single array of words, we need to use flatMap() instead of map()

```
scala> sc.textFile("/SEIS736/TFIDFsmall").map(x => x.split(“\\W+")).collect

res3: Array[Array[String]] = Array(Array(The, quick, brown, fox, jumps, over,
 the, lazy, brown, dog), Array(Waltz, nymph, for, quick, jigs, vex, Bud),
 Array(How, quickly, daft, jumping, zebras, vex))
```

flatMap

- This looks better!

```
scala> sc.textFile("/SEIS736/TFIDFsmall").flatMap(x => x.split(“\\W+")).collect

res4: Array[String] = Array(The, quick, brown, fox, jumps, over, the, lazy, brown,
 dog, Waltz, nymph, for, quick, jigs, vex, Bud, How, quickly, daft, jumping, zebras, vex)
```

Creating Key and Value

- Now, we want to make the output look like the wordcount mapper, so we do a map to take each word as input and transform it to (word,1)
- While we are at it, let’s lower case the word

```
sc.textFile("/SEIS736/TFIDFsmall").flatMap(x => x.split(“\\W+")).
map(x => (x.toLowerCase, 1)).collect

res5: Array[(String, Int)] = Array((the,1), (quick,1), (brown,1), (fox,1),
 (jumps,1), (over,1), (the,1), (lazy,1), (brown,1), (dog,1), (waltz,1),
 (nymph,1), (for,1), (quick,1), (jigs,1), (vex,1), (bud,1), (how,1),
 (quickly,1), (daft,1), (jumping,1), (zebras,1), (vex,1))
```

Sum Reducing

- Now, let’s do the sum reducer function with reduceByKey, which says to run through all the elements for each unique key, and sum them up, two at a time
- The underscores are Scala shorthand for “first number, second number”

```
scala> sc.textFile("/SEIS736/TFIDFsmall").flatMap(x => x.split(“\\W+")).
map(x => (x.toLowerCase, 1)).reduceByKey(_ + _).collect

res6: Array[(String, Int)] = Array((fox,1), (bud,1), (vex,2), (jigs,1), (over,1),
 (for,1), (brown,2), (the,2), (jumps,1), (jumping,1), (daft,1), (quick,2), (nymph,1),
 (how,1), (lazy,1), (zebras,1), (waltz,1), (dog,1), (quickly,1))
```

Sorting

- For fun, let’s sort by key

```
scala> sc.textFile("/SEIS736/TFIDFsmall").flatMap(x => x.split(“\\W+")).
map(x => (x.toLowerCase, 1)).reduceByKey(_ + _).sortByKey().collect

res7: Array[(String, Int)] = Array((brown,2), (bud,1), (daft,1), (dog,1), (for,1),
 (fox,1), (how,1), (jigs,1), (jumping,1), (jumps,1), (lazy,1), (nymph,1), (over,1),
 (quick,2), (quickly,1), (the,2), (vex,2), (waltz,1), (zebras,1))
```

Writing to HDFS

- Finally, let’s write the output to HDFS, getting rid of the collect
- Why 3 output files?

  We had 3 partitions when we originally read in the 3 input files, and nothing subsequently changed that

```
scala> sc.textFile("/SEIS736/TFIDFsmall").flatMap(x => x.split(“\\W+")).
map(x => (x.toLowerCase, 1)).reduceByKey(_ + _).sortByKey().saveAsTextFile(“swc")
scala> exit

[brad@hc ~]$ hadoop fs -ls swc
Found 4 items
-rw-r--r--   3 brad supergroup          0 2015-10-24 06:46 swc/_SUCCESS
-rw-r--r--   3 brad supergroup         59 2015-10-24 06:46 swc/part-00000
-rw-r--r--   3 brad supergroup         59 2015-10-24 06:46 swc/part-00001
-rw-r--r--   3 brad supergroup         59 2015-10-24 06:46 swc/part-00002
```

Seeing Our Output

```
[brad@hc ~]$ hadoop fs -cat swc/part-00000
(brown,2)
(bud,1)
(daft,1)
(dog,1)
(for,1)
(fox,1)
(how,1)
[brad@hc ~]$ hadoop fs -cat swc/part-00001
(jigs,1)
(jumping,1)
(jumps,1)
(lazy,1)
(nymph,1)
(over,1)
[brad@hc ~]$ hadoop fs -cat swc/part-00002
(quick,2)
(quickly,1)
(the,2)
(vex,2)
(waltz,1)
(zebras,1)
```

An Alternative Style

- While the on-liner style (also known as a fluent style) is concise, it is often easier to develop and debug by assigning each functional block to a variable
- Note that nothing really happens until the the actions (reduceByKey and saveAsTextFile) are executed

```
scala> val lines = sc.textFile(“/SEIS736/TFIDFsmall”)
scala> val words = lines.flatMap(x => x.split(“\\W+"))
scala> val mapOut = words.map(x => (x.toLowerCase, 1))
scala> val reduceOut =mapOut.reduceByKey(_ + _)
scala> val sortedOut = reduceOut.sortByKey()
scala> sortedOut.saveAsTextFile("swc")
```

Make it a Standalone Program

```
package edu.stthomas.gps.spark

import org.apache.spark.{SparkConf, SparkContext}

object SparkWordCount {

  def main(args: Array[String]) {

    val sparkConf = new SparkConf().setAppName("Spark WordCount")
    val sc = new SparkContext(sparkConf)

    sc.textFile("/SEIS736/TFIDFsmall")
      .flatMap(x => x.split("\\W+"))
      .map(x => (x.toLowerCase, 1))
      .reduceByKey(_ + _)
      .sortByKey()
      .saveAsTextFile("swc")

    System.exit(0)
  }
}
```

```
spark-submit \
    --class edu.stthomas.gps.spark.SparkWordCount \
    --master yarn-cluster \
    --executor-memory 512M \
    --num-executors 2 \
    /home/brad/spark/spark.jar
```

Dataframes

- Dataframes are like RDDs, but they are used for structured data
- They were introduced to support SparkSQL, where a data frame is like a relational table
- But, they are starting to see more general use, outside of SparkSQL, because of the higher-level API and optimization opportunities for performance

Dataframe Example

```
scala> val stocks = List(“NYSE,BGY,2010-02-08,10.25,10.39,9.94,10.28,600900,10.28",
“NYSE,AEA,2010-02-08,4.42,4.42,4.21,4.24,205500,4.24",
“NYSE,CLI,2010-02-12,30.77,31.30,30.63,31.30,1020500,31.30")

scala> case class Stock(exchange: String, symbol: String, date: String, open: Float, high:
Float, low: Float, close: Float, volume: Integer, adjClose: Float)

scala> val Stocks = stocks.map(_.split(“,")).map(x=>Stock(
x(0),x(1),x(2),x(3).toFloat,x(4).toFloat,x(5).toFloat,x(6).toFloat,x(7).toInt,x(8).toFloat))

scala> val StocksRDD = sc.parallelize(Stocks)

scala> val StocksDF = StocksRDD.toDF
```

Dataframe Example

```
scala> StocksDF.count
res0: Long = 3

scala> StocksDF.first
res1: org.apache.spark.sql.Row = [NYSE,BGY,2010-02-08,10.25,10.39,9.94,10.28,600900,10.28]

scala> StocksDF.show
+--------+------+----------+-----+-----+-----+-----+-------+--------+
|exchange|symbol|      date| open| high|  low|close| volume|adjClose|
+--------+------+----------+-----+-----+-----+-----+-------+--------+
|    NYSE|   BGY|2010-02-08|10.25|10.39| 9.94|10.28| 600900|   10.28|
|    NYSE|   AEA|2010-02-08| 4.42| 4.42| 4.21| 4.24| 205500|    4.24|
|    NYSE|   CLI|2010-02-12|30.77| 31.3|30.63| 31.3|1020500|    31.3|
+--------+------+----------+-----+-----+-----+-----+-------+--------+
```

Dataframe Example

```
scala> StocksDF.printSchema
root
 |-- exchange: string (nullable = true)
 |-- symbol: string (nullable = true)
 |-- date: string (nullable = true)
 |-- open: float (nullable = false)
 |-- high: float (nullable = false)
 |-- low: float (nullable = false)
 |-- close: float (nullable = false)
 |-- volume: integer (nullable = true)
 |-- adjClose: float (nullable = false)

scala> StocksDF.groupBy("date").count.show
+----------+-----+
|      date|count|
+----------+-----+
|2010-02-08|    2|
|2010-02-12|    1|
+----------+-----+

scala> StocksDF.groupBy("date").count.filter("count > 1").rdd.collect
res2: Array[org.apache.spark.sql.Row] = Array([2010-02-08,2])
```

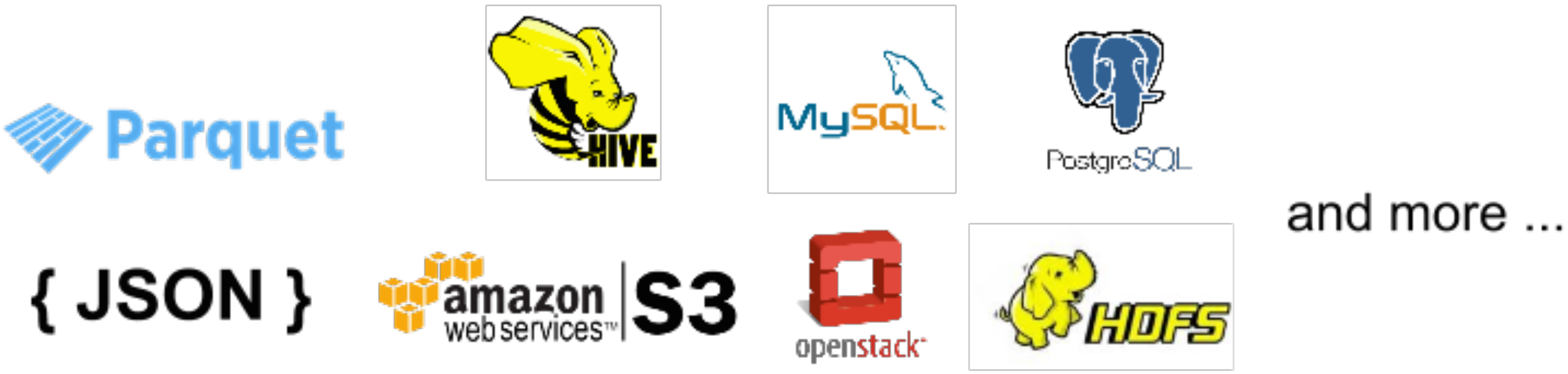
Dataframe Using SQL

```
scala> StocksDF.registerTempTable("stock")

scala> sqlContext.sql("SELECT symbol, close FROM stock WHERE close > 5 ORDER BY symbol").show
+------+-----+
|symbol|close|
+------+-----+
|   BGY|10.28|
|   CLI| 31.3|
+------+-----+
```

Dataframe Read/Write Interface

- The read/write interface makes it very easy to read and write common data formats



Formats and Sources supported by DataFrames

Dataframe Read/Write Interface

- Reading in a JSON file as a Dataframe

```
scala> val df = sqlContext.read.format(“json").load("json/zips.json")

scala> df.printSchema
root
 |-- _id: string (nullable = true)
 |-- city: string (nullable = true)
 |-- loc: array (nullable = true)
 |    |-- element: double (containsNull = true)
 |-- pop: long (nullable = true)
 |-- state: string (nullable = true)

scala> df.count
res0: Long = 29467

scala> df.filter("_id = 55105").show
+-----+----------+--------------------+-----+-----+
|  _id|      city|                 loc|  pop|state|
+-----+----------+--------------------+-----+-----+
|55105|SAINT PAUL|[-93.165148, 44.9...|26216|   MN|
+-----+----------+--------------------+-----+-----+
```

Dataframe Read/Write Interface

- Converting the Dataframe to Parquet format, and then querying it as a Hive table

```
scala> val options = Map("path" -> “/user/hive/warehouse/zipcodes")
scala> df.select(“*").write.format("parquet").options(options).saveAsTable("zipcodes")

hive> DESCRIBE zipcodes;
OK
_id                 string
city                string
loc                 array<double>
pop                 bigint
state               string

hive> SELECT city FROM zipcodes WHERE (`_id` == '55105');
SAINT PAUL
```

Scala

Spark

Scala/Spark Examples

Classroom Experience

Classroom Experience

- After a 1/2 semester of Hadoop Java MapReduce programming, I introduce Scala and Spark in two 3-hour lectures/demos
- Almost all students are able to successfully complete two homework assignments (one heavily guided, one without direction)
- Students enjoy the interactive shell style of development, concise API, expressiveness, and easier/faster overall development time/effort
    - About 50% of students change their course project proposals to use Scala/Spark after this experience
- Two major hurdles
    - Spark is lazy, so errors are initially attributed to actions, yet the root cause is often a preceding transformation
    - Students often confuse the Spark and Scala APIs