SPARK ON HIPERGATOR

Ying Zhang
yingz@ufl.edu

March 15th, 2018

UF | Information Technology

HiPerGator
The University of Florida Supercomputer for Research

AGENDA

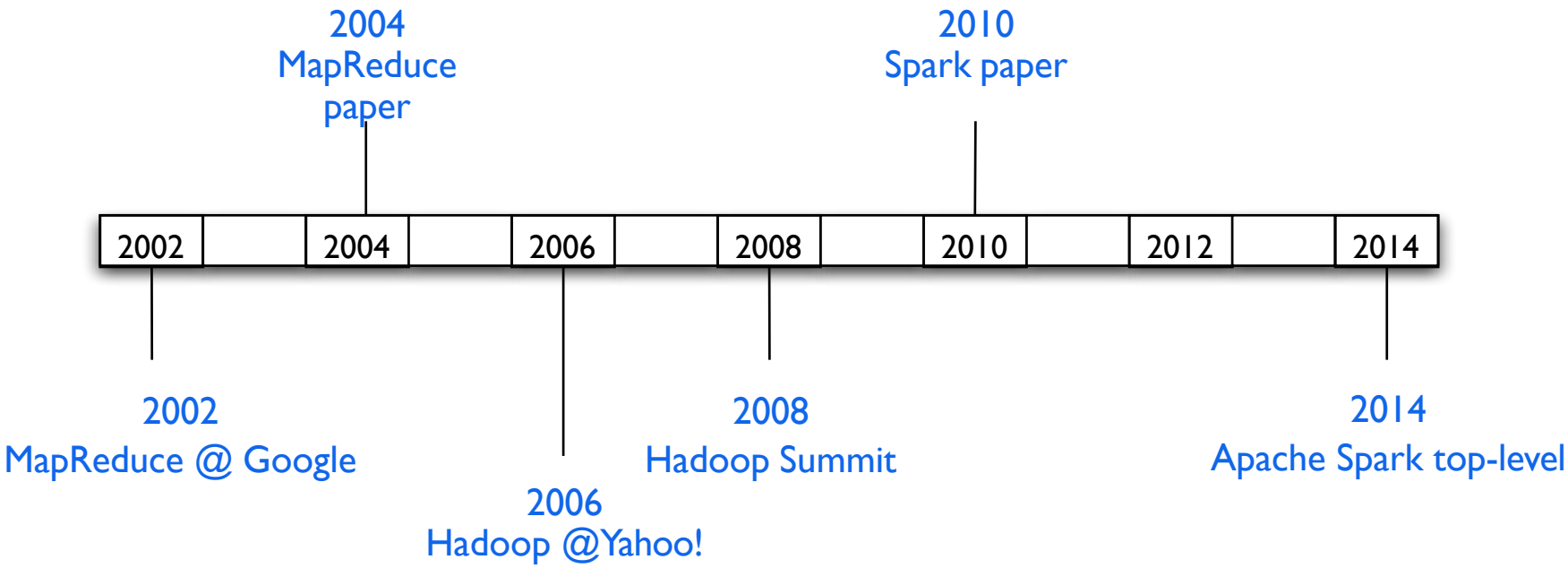
- Introduction
  - Apache Spark
  - Research Computing and HiPerGator
- Spark on HiPerGator
- Hands-on Exercises

AGENDA

- Introduction
  - Apache Spark
  - Research Computing and HiPerGator
- Spark on HiPerGator
- Hands-on Exercises

APACHE SPARK

- A brief history



MAP-REDUCE

- Data-parallel model
  - One operation, run it on all of the data
- A simple programming model that applies to many large-scale computing problems
- Typical problem
  - Read a lot of data
  - Map: extract desired information from each record
  - Shuffle and sort
  - Reduce: aggregate, summarize, filter, or transform
  - Write the results

MAP-REDUCE

- Word count example:
  - *Map* function:
    - Input: a key/value pair

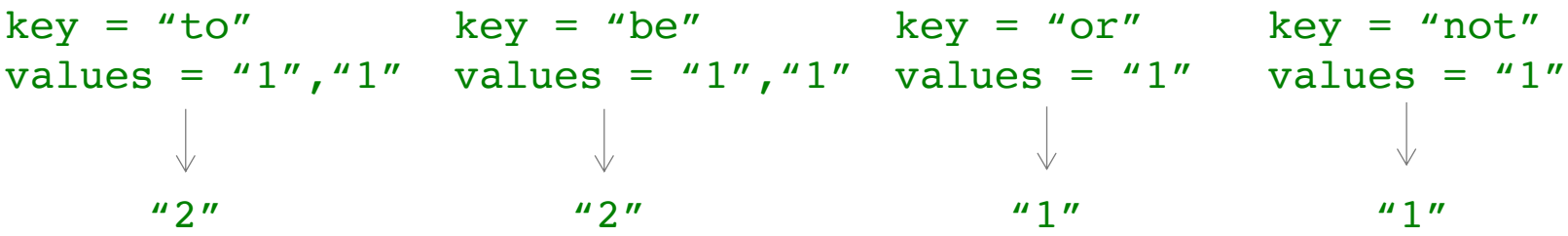
```
Key = “file1”
Value = “to be or not to be”
```

    - Output: key/value pairs

```
“to”, “1”
“be”, “1”
“or”, “1”
“not”, “1”
“to”, “1”
“be”, “1”
```

MAP-REDUCE

- Word count example:
  - *Shuffle/sort*: gathers all pairs with the same key value
  - *Reduce* function combines the values for a key



- Output:

```
"to", "2"
"be", "2"
"or", "1"
"not", "1"
```

MAP-REDUCE

- Major limitations:
  - Difficulty to program directly
  - Performance bottlenecks
- Higher level frameworks, e.g. Hive, Pregel, Dremel, etc.

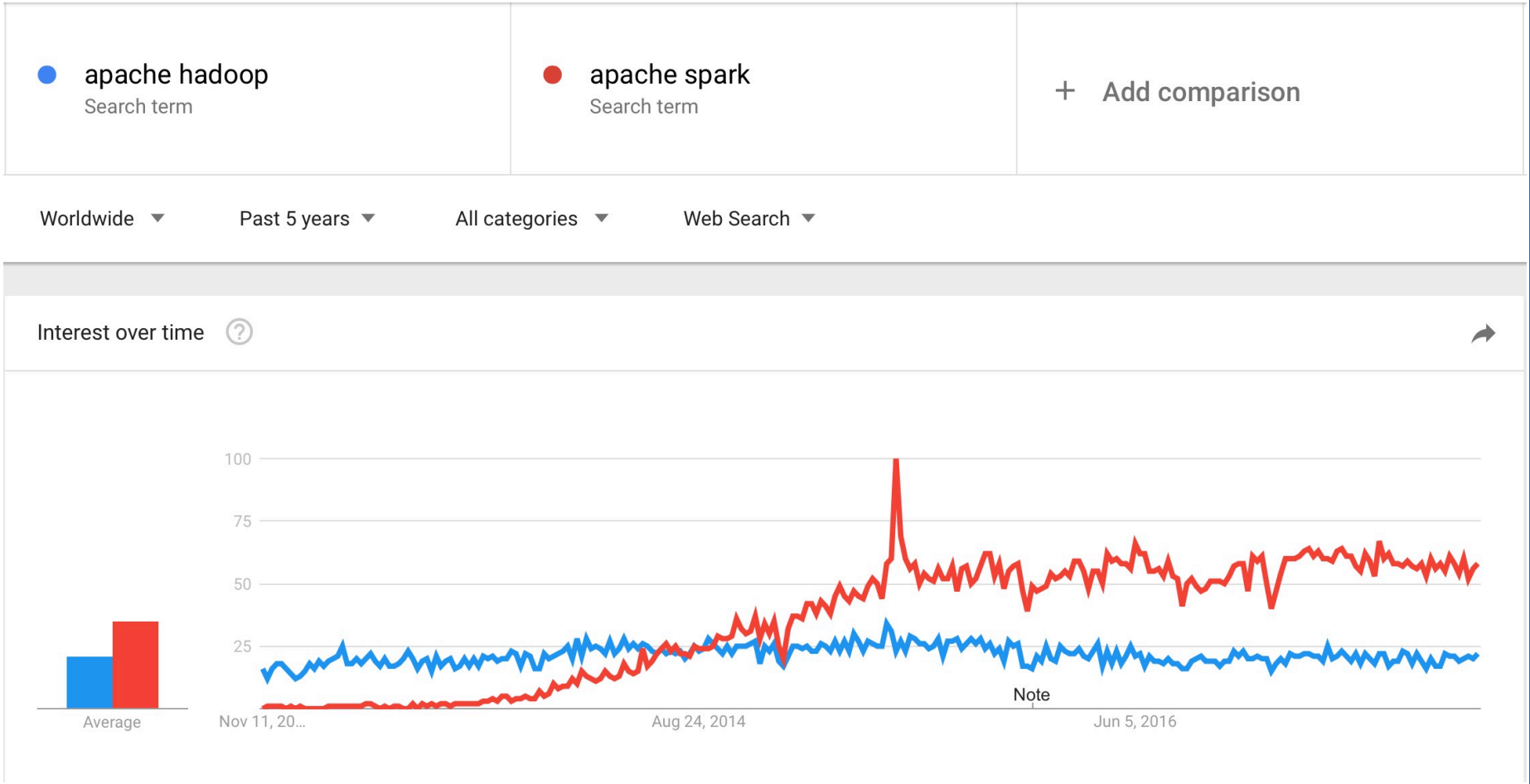
SPARK

- Handles batch, interactive, and real-time within a single framework
- Integration with Java, Python, and R
- Programming at a higher level of abstraction
- More general and beyond map/reduce

HADOOP & SPARK

- Hadoop
  - Started in 2006 at Yahoo
  - HDFS: Hadoop File System
  - YARN: a scheduler coordinates application runs
  - Built in JAVA, support Python and others
- Spark
  - Started in 2008 at AMPLab at UC Berkeley
  - Resilient Distributed Dataset (RDD), in memory process
  - Run in standalone mode or with Hadoop cluster
  - Directed Acyclic Graph (DAG), visualize the order of operations and relationships of operations
  - Written in Scala, support Java, Python and R

HADOOP VS. SPARK



SPARK PROGRAMMABILITY

WordCount in 50+ lines of Java

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

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

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

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

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

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

WordCount in 3 lines of Spark Scala

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

SPARK PERFORMANCE

Sort 100TB of data with 1 Trillion records

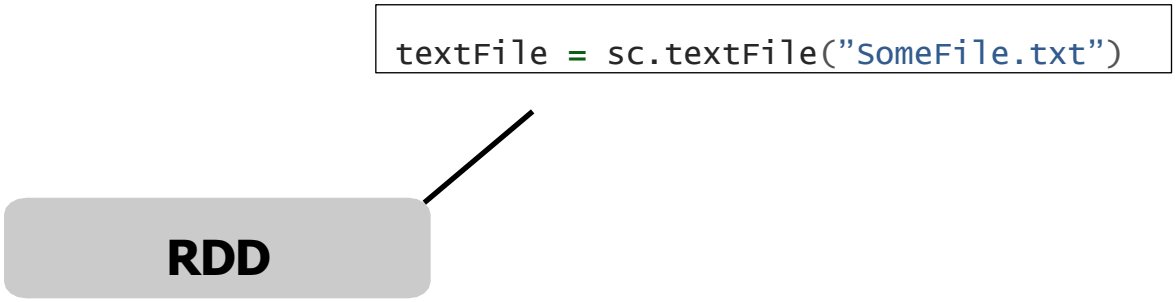
	Hadoop MR Record	Spark Record
Data Size	102.5TB	100TB
Elapsed Time	72 minutes	23 minutes
Number of Nodes	2100	206
Number of Cores	50400 physical	6592 virtualized
Sort Rate	1.42 TB/min	4.27 TB/min
Sort rate/node	0.67 GB/min	20.7 GB/min

Source: Daytona GraySort Competition 2014, https://databricks.com

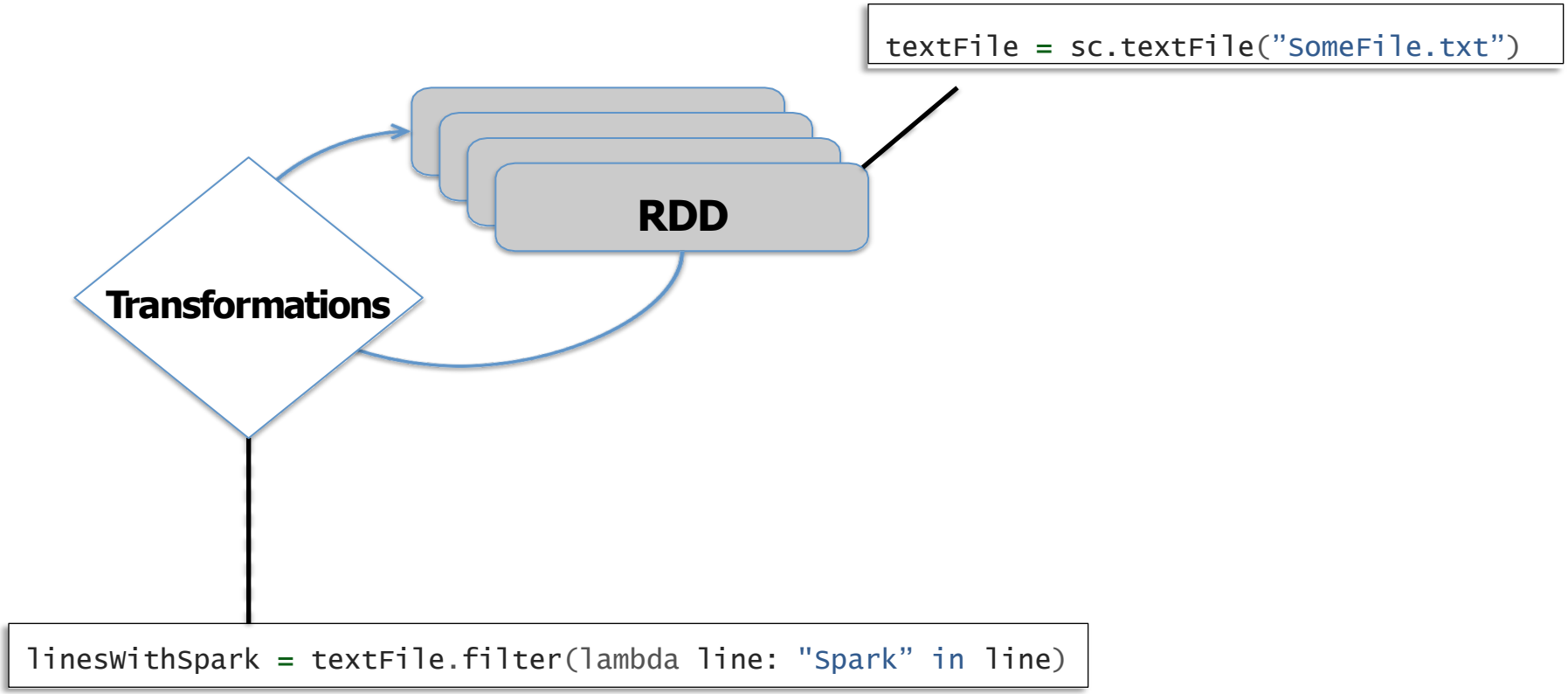
RDD: RESILIENT DISTRIBUTED DATASETS

- Primary abstraction in Spark
- Collection of elements that can be operated on in parallel
  - Transformations
  - Actions
- Fault tolerance: track the series of transformations used to build them (lineage)

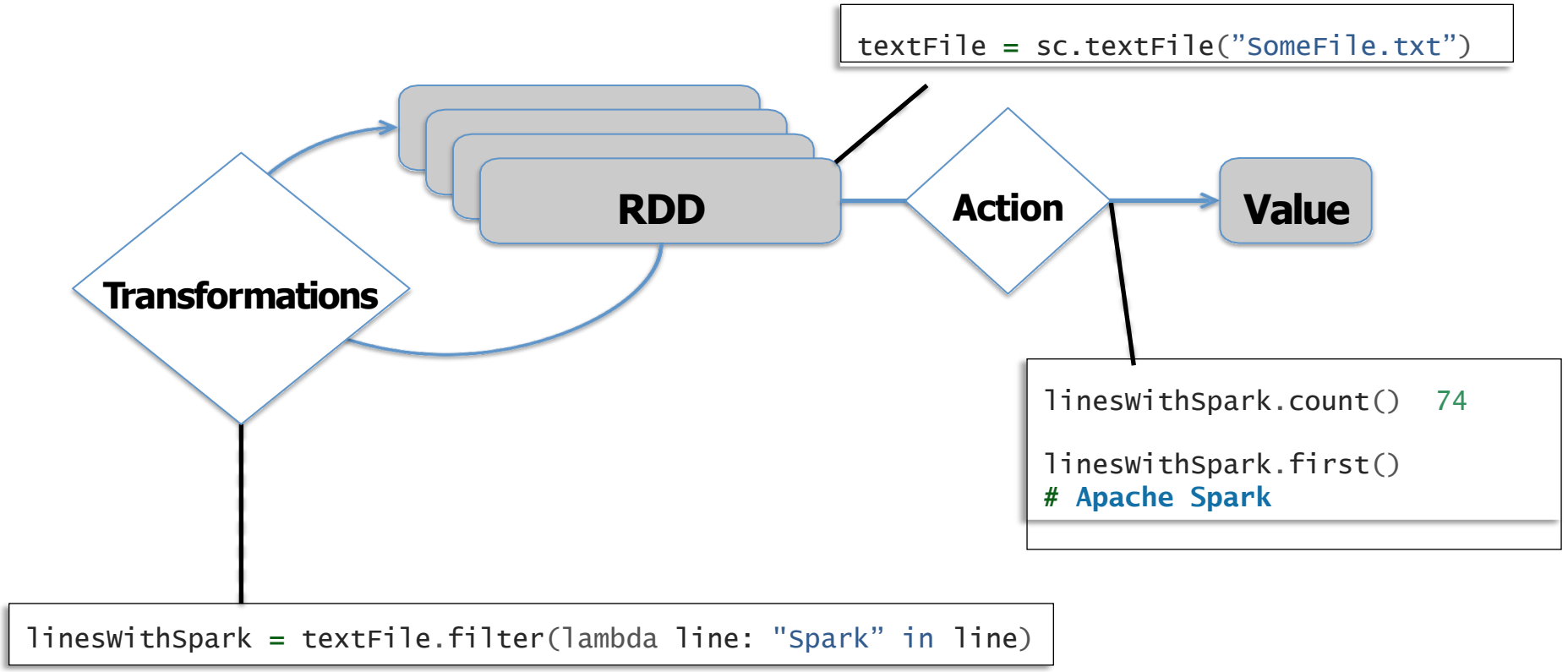
RDD: HOW DOES IT WORK?



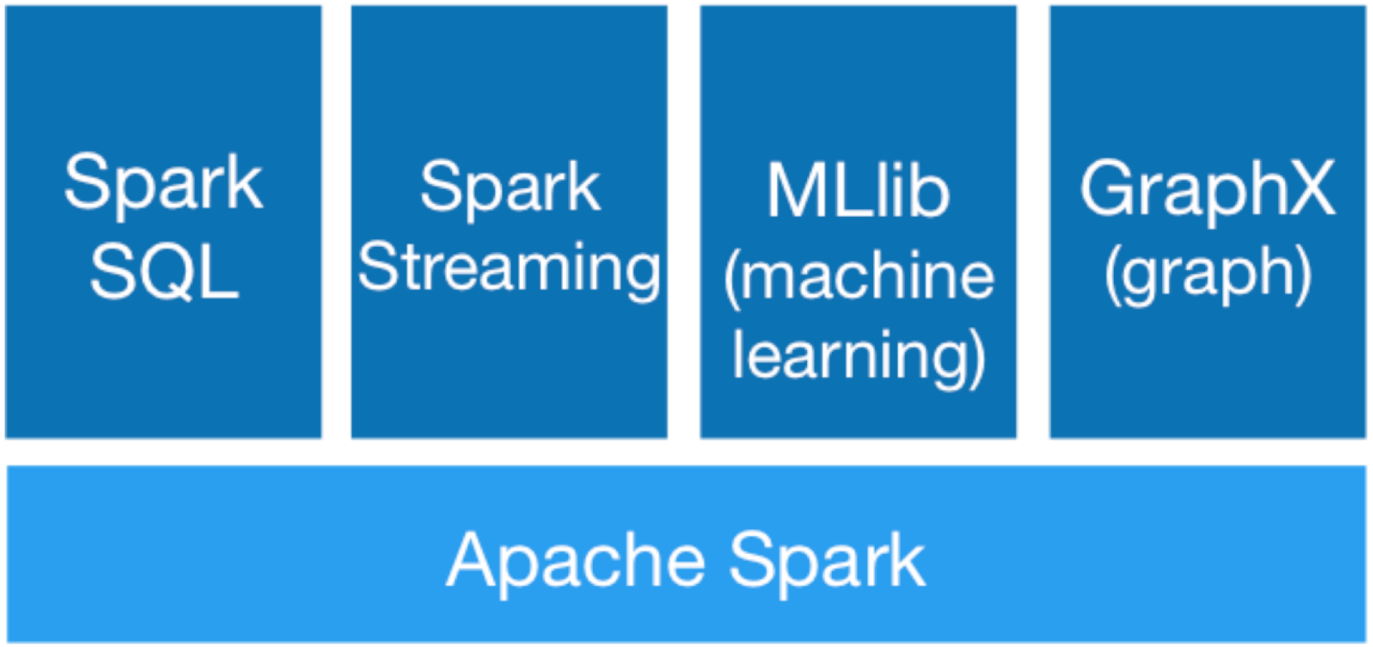
RDD: HOW DOES IT WORK?



RDD: HOW DOES IT WORK?



SPARK ECOSYSTEM

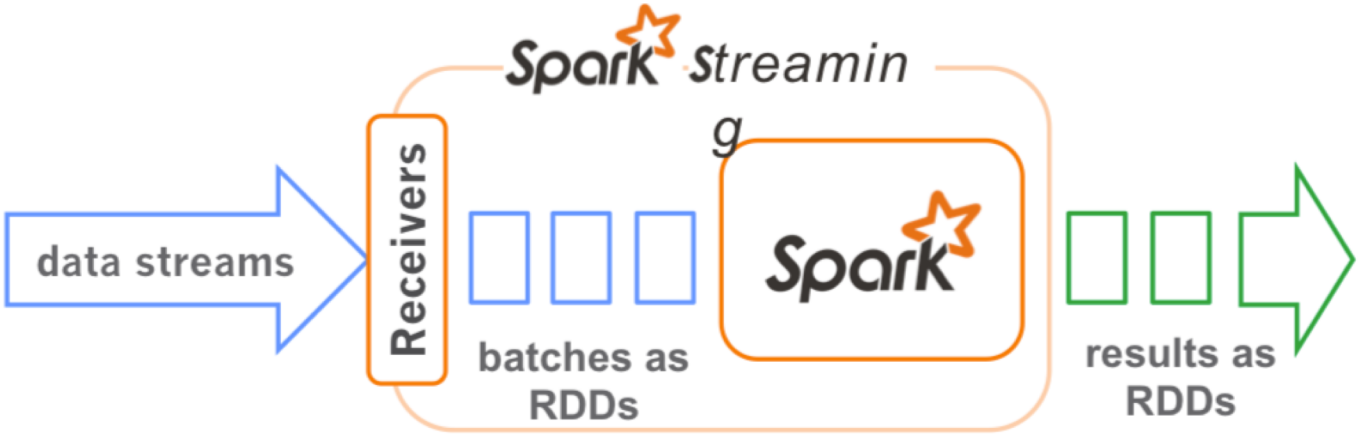


SPARK SQL AND DATAFRAMES

- SparkSQL
  - Allows SQL-like commands on distributed data sets
- Spark DataFrames
  - Developed in Spark 2.0
  - Organizes data into named columns (i.e. RDD with schema)
- SparkSQL allows querying DataFrames
- Support Python, Scala, Java, and R

SPARK STREAMING

- What is it?
  - Receive data streams from input source
  - Break the data streams into small batches as RDDs (Dstream)
  - Process the batches using RDD operations in parallel
  - Output to databases/dashboards
  - Fault tolerant, second-scale latency
  - Support Scala, Java, and Python



SPARK MLLIB

- Provide machine learning primitives
  - Shipped with Spark since version 0.8
- Algorithms
  - Classification: logistic regression, linear SVM, Naïve Bayes
  - Regression: generalized linear regression (GLM)
  - Collaborative filtering: alternating least squares (ALS)
  - Clustering: k-means
  - Decomposition: single value decomposition (SVD), and principal component analysis (PCA)
- Support Java, Scala, and Python

SPARK GRAPHX

- Graph analytics
  - Examples: social networks, page rank, fraud detection, etc.
  - Graph data modeling
  - Graph data processing
- GraphX
  - API for graphs and graph-parallel computation
  - A growing library of graph algorithms
  - Performance comparable to the fastest specialized graph processing systems

AGENDA

- Introduction
  - Apache Spark
  - Research Computing and HiPerGator
- Spark on HiPerGator
- Hands-on Exercises

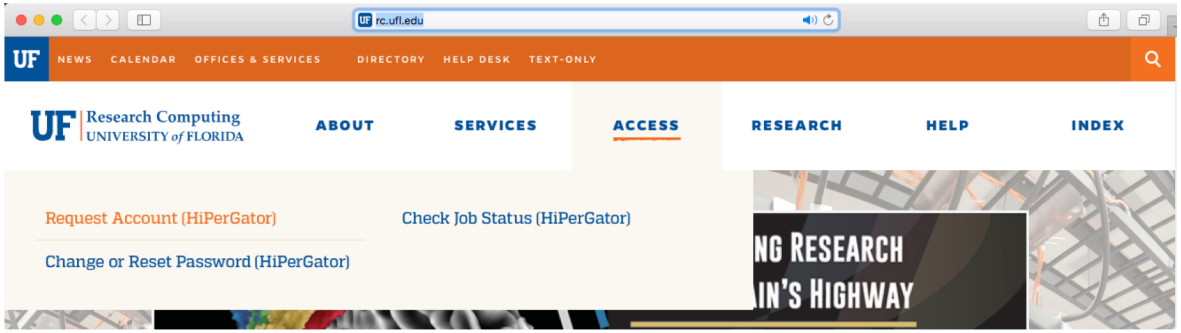
HIPERGATOR

HIPERGATOR LOGISTICS

- Hardware
  - Over 50,000 computing cores
  - 3 PB of data storage
  - 180 TB of memory
  - GPU partition
  - Big memory partition
- Software
  - Over 1000 software applications installed
  - Covering wide range of research disciplines

HIPERGATOR ACCOUNTS

- Apply for a user account at: http://rc.ufl.edu



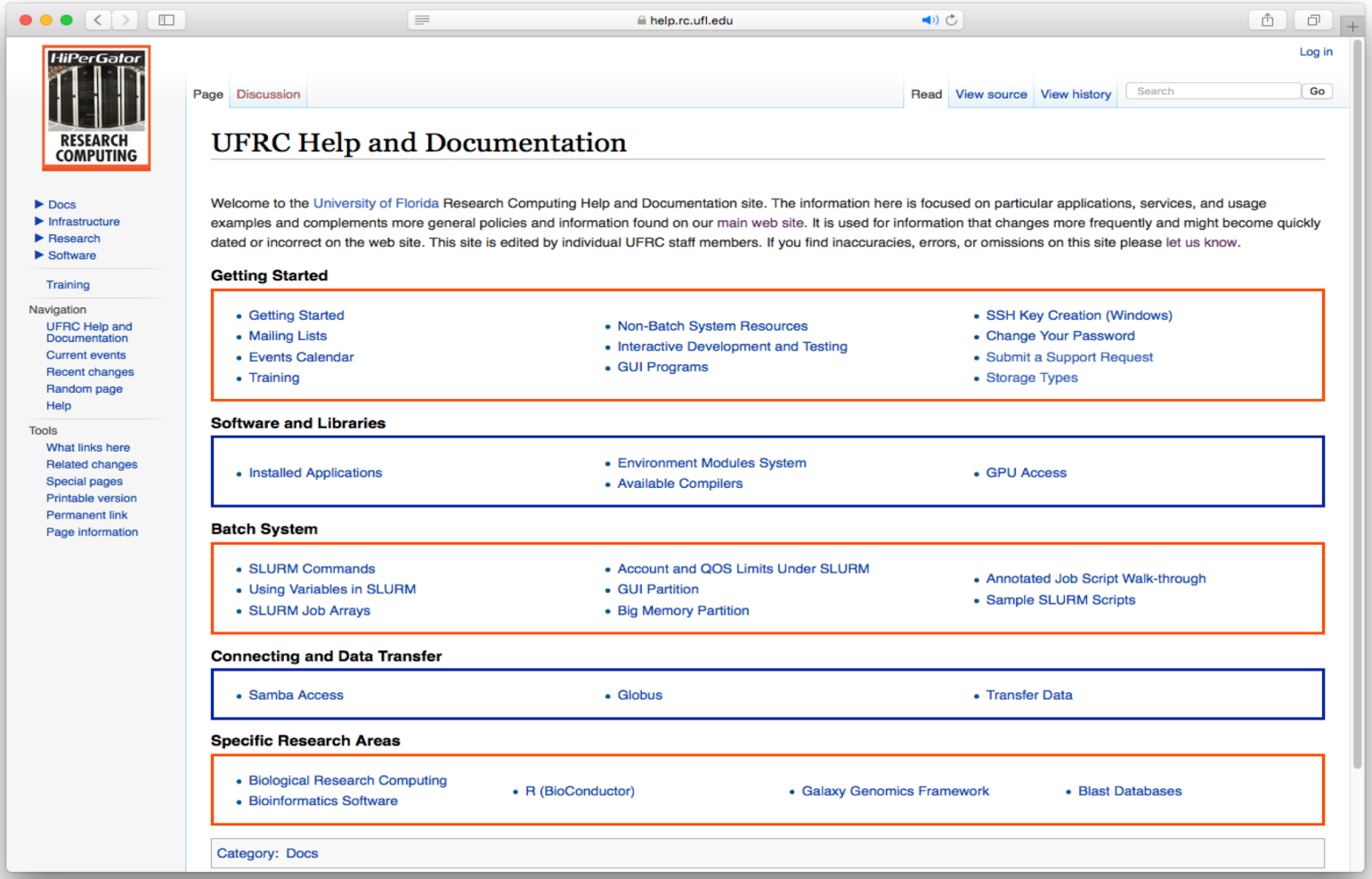
  - Need faculty sponsor
  - GatorLink ID

HIPERGATOR ENVIRONMENT

- A Linux-based system
- Interactive session for development and testing
- Production runs handled by job scheduler – SLURM

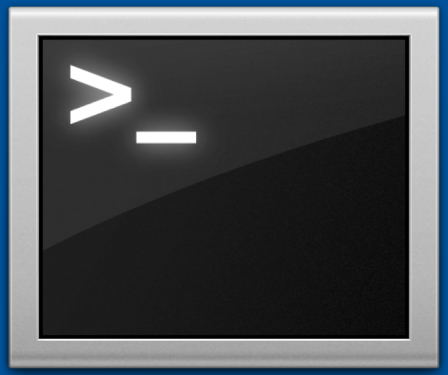
USING HIPERGATOR

- https://help.rc.ufl.edu



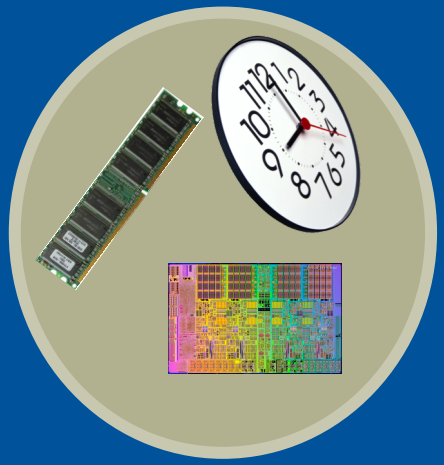
CLUSTER BASICS

User interaction



Login node (Head node)

Scheduler



Tell the scheduler what you want to do

Compute resources

Your job runs on the cluster

Source: Matt Gitzendanner, "Intro to Research Computing and HiPerGator"

SPARK ON HIPERGATOR

- Version 2.1.0 and 2.2.0
- Programming in Scala, Java, Python, or R
- Running standalone Spark jobs via SLURM
- Use spark module

```
module load spark/2.1.0
```

or

```
module load spark/2.2.0
```

- Use programming modules

```
module load scala
```

or

```
module load python (or java, or R)
```

AGENDA

- Introduction
  - Apache Spark
  - Research Computing and HiPerGator
- Spark on HiPerGator
- Hands-on Exercises