BEYOND MAPREDUCE – APACHE SPARK & GRAPH

CKME 134 – BIG DATA ANALYTICS TOOLS

RYERSON UNIVERSITY

SPRING 2015

Instructor: Shaohua Zhang

Course Outline

1. Intro to Big Data
2. Distributed Computing and MapReduce
3. Hadoop Ecosystem
4. Programming Hive
5. Advanced Hive
6. Mid-Term Review
7. Programming Pig
8. Advanced Pig
9. Hadoop Use Cases
10. Building Data Product & Next-Gen Hadoop (Spark)
11. Beyond Hadoop: Graph Analytics and Recommender Systems

Exam and Preparation

- Final Exam Preparation
  - Course Notes – Session 1~Session 9
  - Hive Intro Lab – Session 4 Lab
  - Pig Intro Lab 1 – Session 7 Lab
- Hangout Session 2
  - Engineering Building (lower level)
  - Saturday, April 11
  - 1pm ~ 4pm

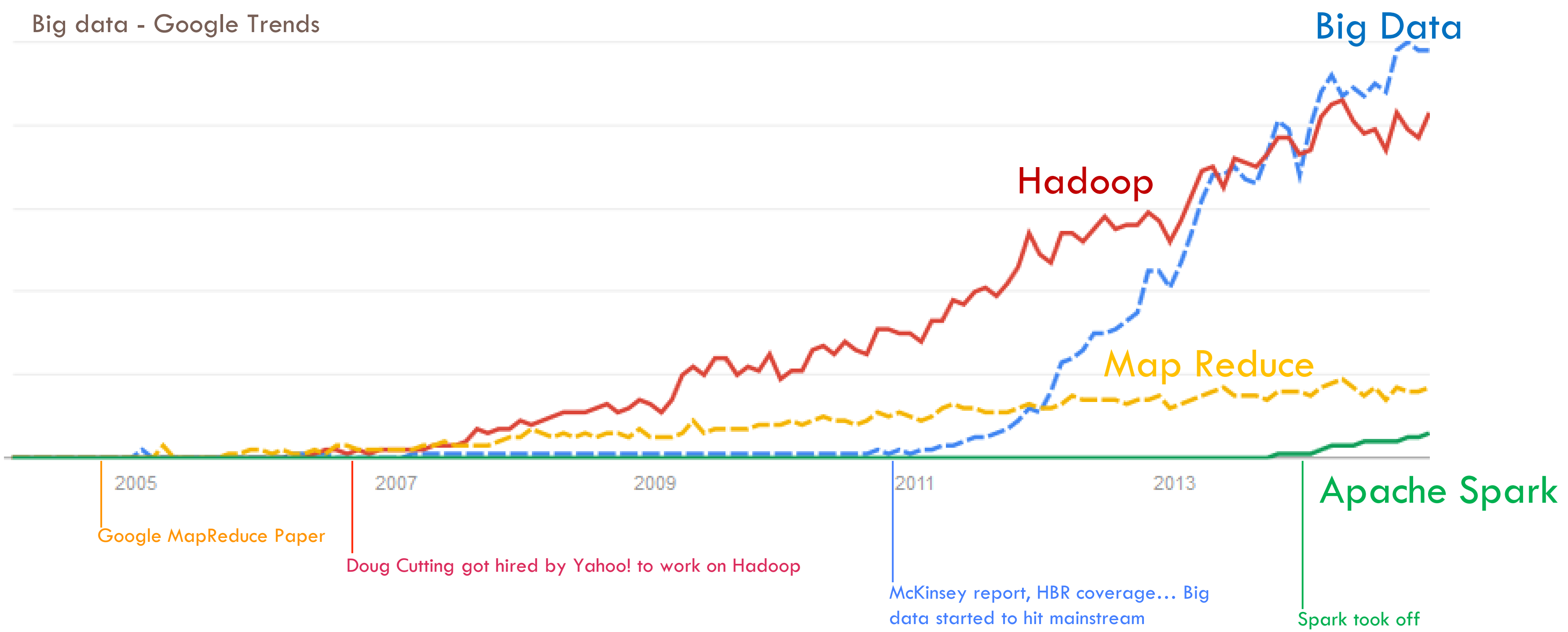
Lecture 10 - Outline

- Apache Spark Introduction
- Graph Processing
  - PageRank Algorithm

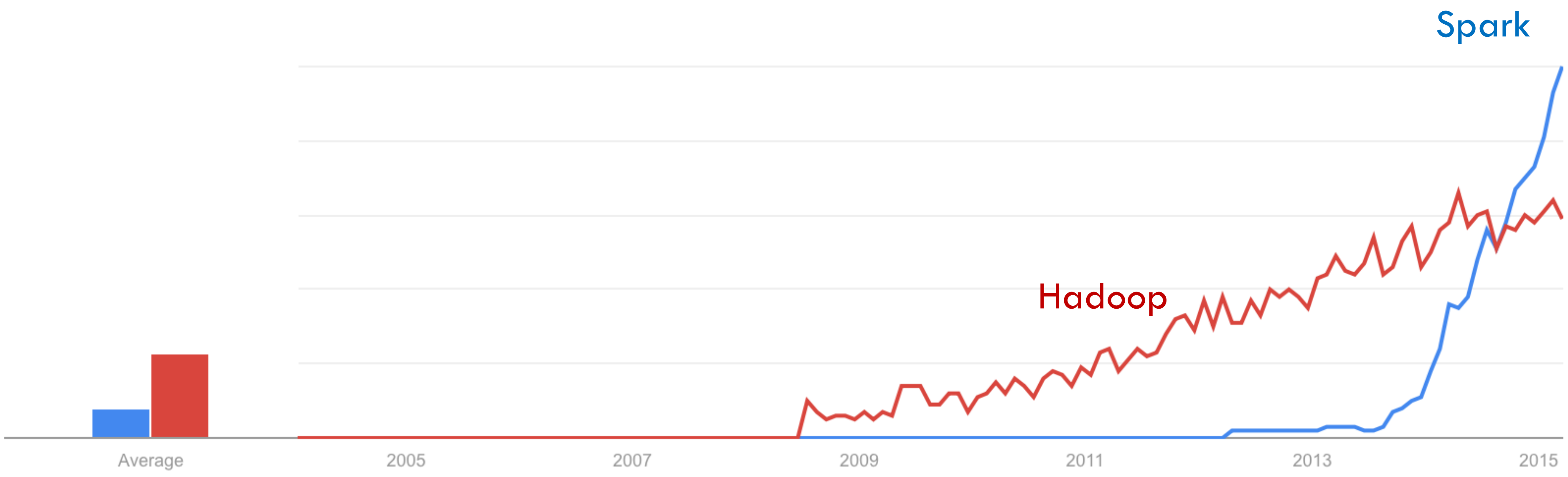
Lecture 11

- GraphLab
- Recommender System
- Spark Mllib
- Spark Streaming

Big Data History



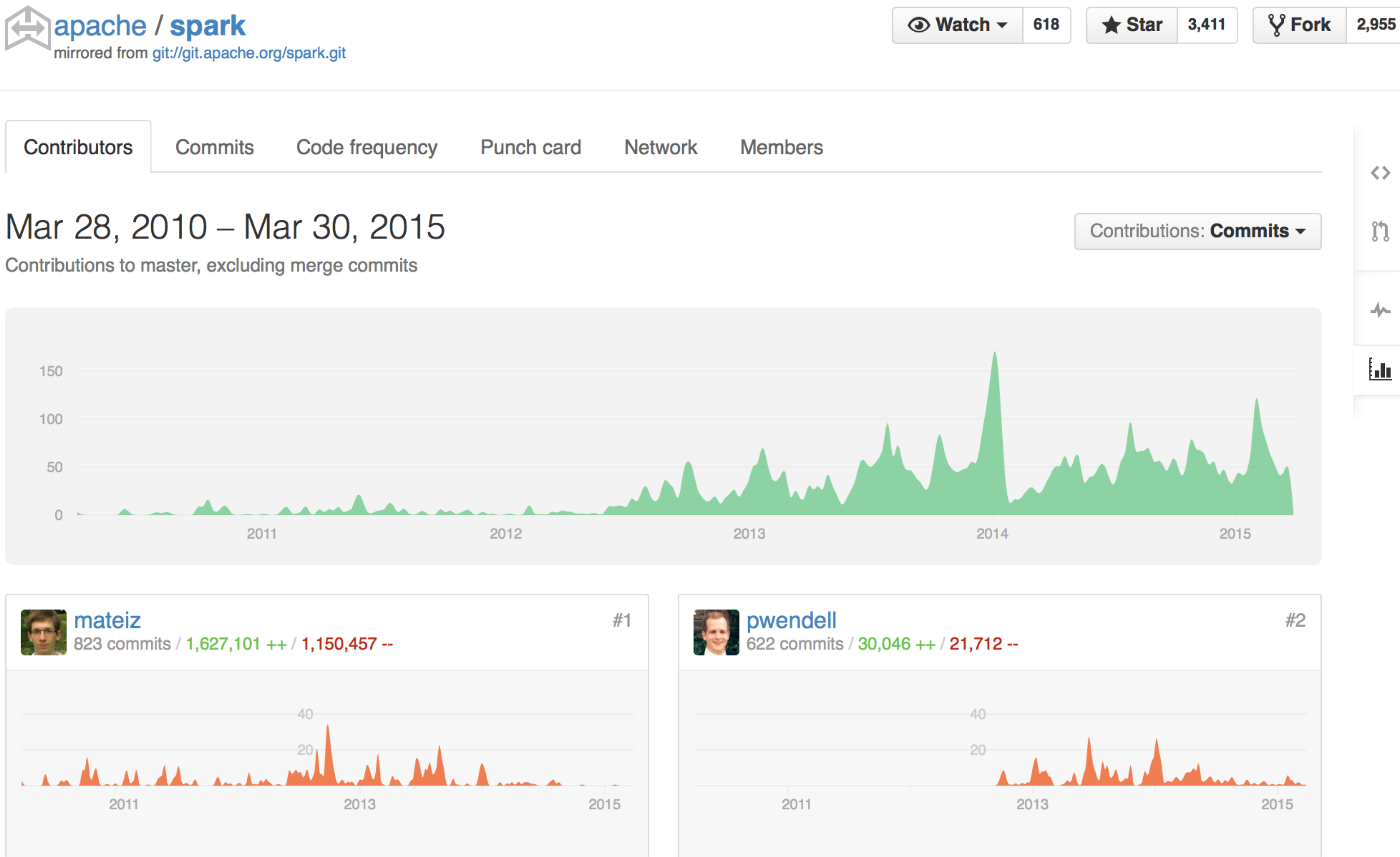
Big Data – New Trend



New TeraSort World Record!

- The previous world record was 72 minutes, set by Yahoo using a Hadoop MapReduce cluster of 2100 nodes.
- New world record set by Spark using Spark on 206 EC2 nodes, basically with **3X** faster using **10X** fewer machines.

	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**


apache / spark
mirrored from git://git.apache.org/spark.git
Watch 618
Star 3,411
Fork 2,955
Contributors
Commits
Code frequency
Punch card
Network
Members
Mar 28, 2010 – Mar 30, 2015
Contributions to master, excluding merge commits
Contributions: Commits
150
100
50
0
2011
2012
2013
2014
2015
mateiz
823 commits / 1,627,101 ++ / 1,150,457 --
#1
40
20
2011
2013
2015
pwendell
622 commits / 30,046 ++ / 21,712 --
#2
40
20
2011
2013
2015

Spark Adoptions

- Yahoo! – Personalization and ad analytics
- Conviva – Real-time video stream optimization
- Ooyala – Cross-device personalized video experience
- Groupon, Shopify, Alibaba, Taobao, etc…

Apache Spark History

- Spark was initially started by Matei Zaharia at UC Berkeley AMPLab in 2009
- Open sourced in 2010
- Donated to Apache Foundation in 2013
- Became an Apache Top-Level Project in Feb 2014

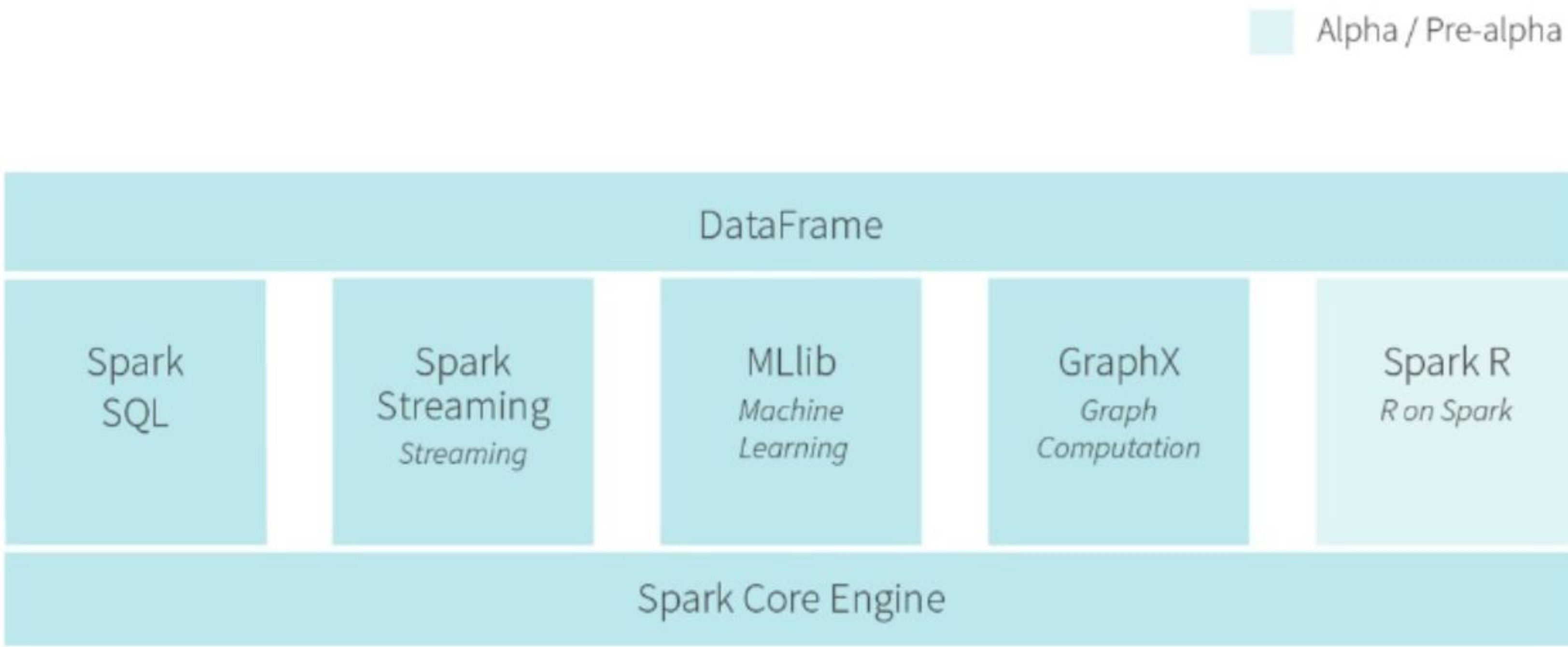
The Berkeley AMPLab

Governmental and industrial funding:

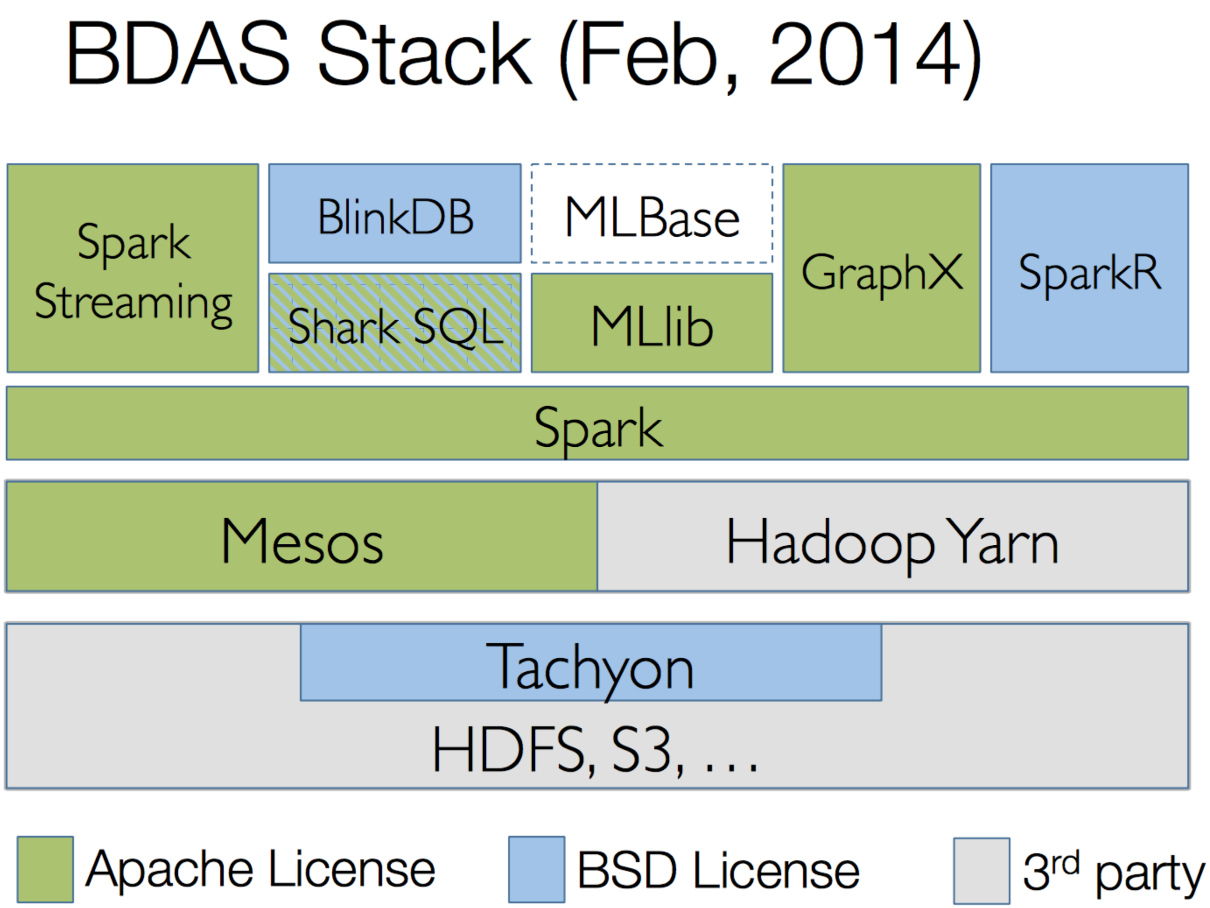
Goal: Next generation of open source data analytics stack for industry & academia: Berkeley Data Analytics Stack (BDAS)

DataDricks

- Databricks is a company founded by the creators of Apache Spark, that aims to help clients with cloud-based big data processing using Spark



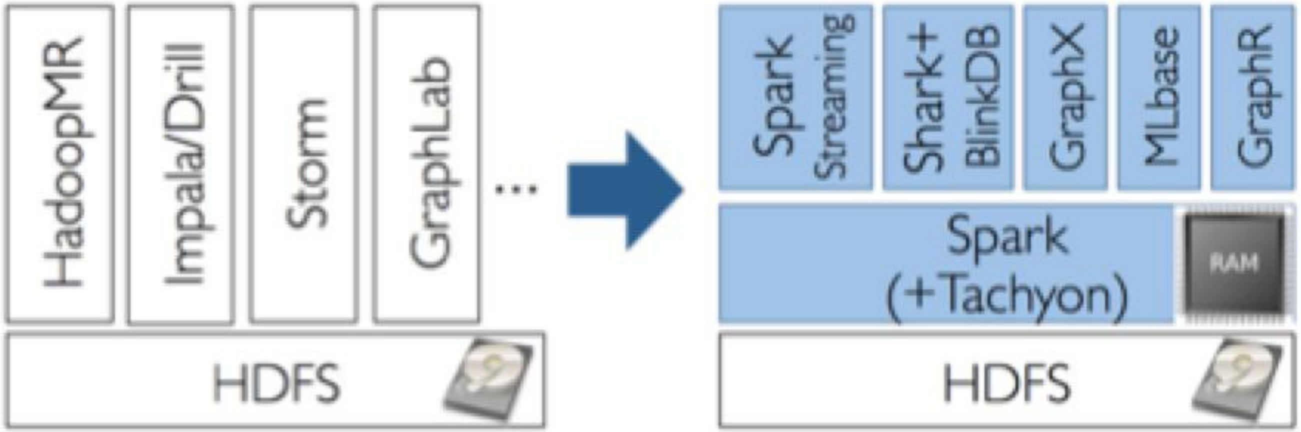
The BDAS – Berkeley Data Analytics Stack



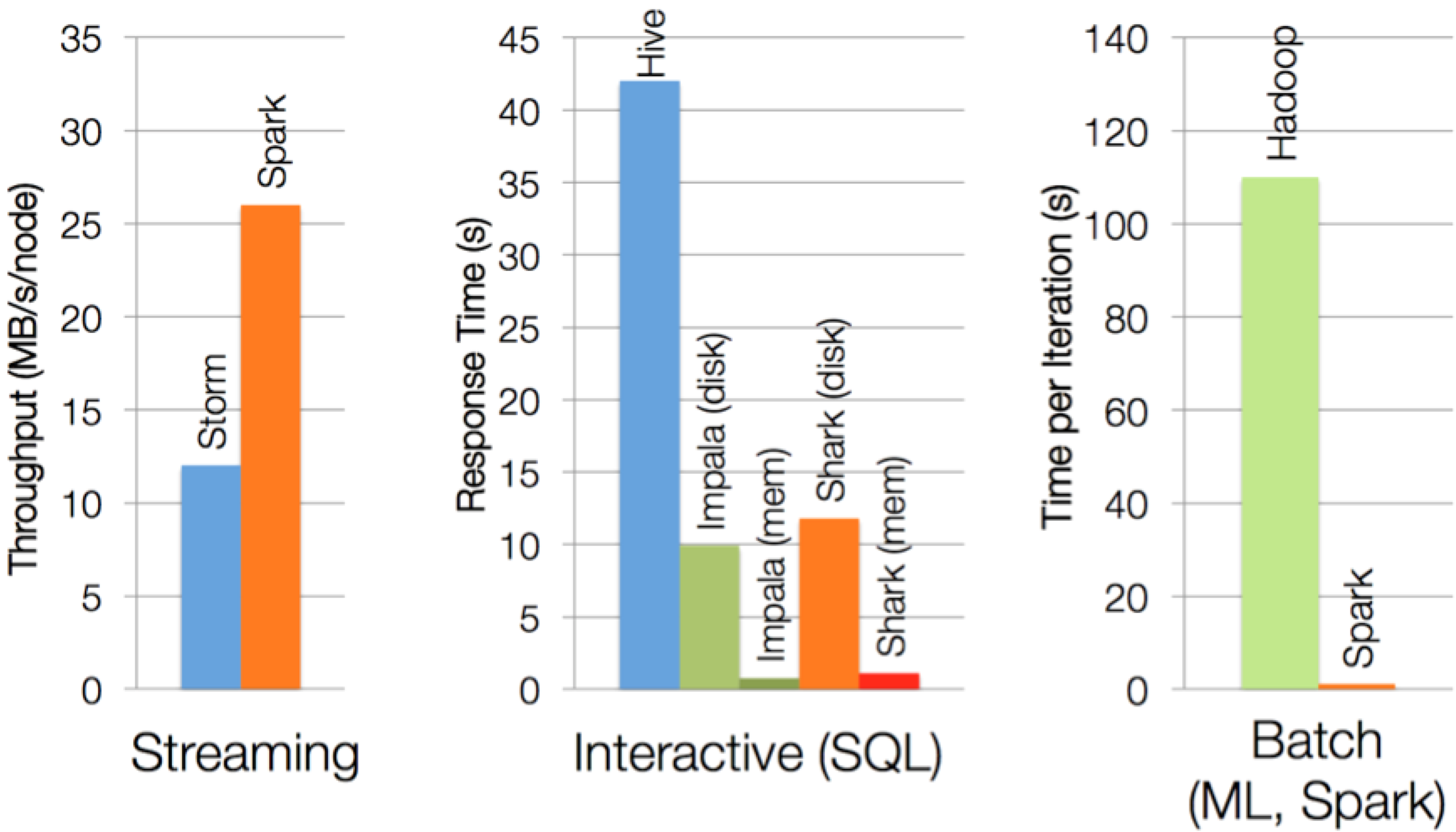
Unified Data Platform

- A unified platform that supports many data processing needs including
  - Batch processing (Spark)
  - Stream processing (SparkX)
  - Interactive (Spark SQL)
  - Iterative (MLlib, GraphX)

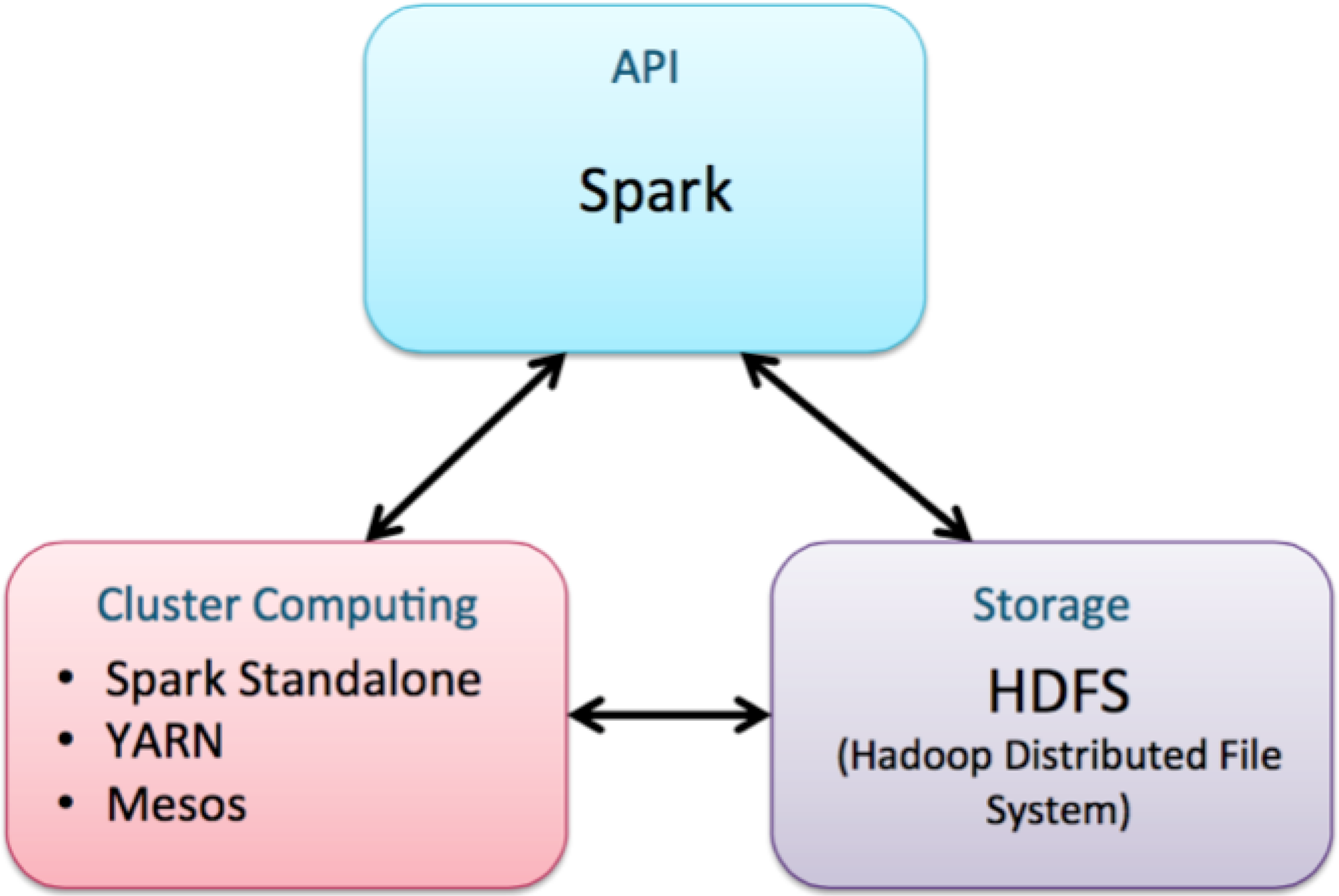
One size fits many!



Performance Benchmarks



Distributed Computing with Spark



Stream Processing

http://www.streamcrab.com

Unify Real-Time and Historical Analytics

- Spark allows one to write virtually the same batch and streaming codes
  - Easy to develop and maintain consistency

```
// count words from a file (batch)
val file = sc.textFile("hdfs://.../pagecounts-*.gz")
val words = file.flatMap(line => line.split(" "))
val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
wordCounts.print()

// count words from a network stream, every 10s (streaming)
val ssc = new StreamingContext(args(0), "NetCount", Seconds(10), ..)
val lines = ssc.socketTextStream("localhost", 3456)
val words = lines.flatMap(_.split(" "))
val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
wordCounts.print()
ssc.start()
```

Spark v. Hadoop MapReduce

- Spark takes the concepts of MapReduce to the next level!
  - Higher-level API = faster, easier development
  - Low latency = near real-time processing
  - In-memory data storage = up to 100x performance improvement

Spark

```
sc.textFile(file) \
  .flatMap(lambda s: s.split()) \
  .map(lambda w: (w,1)) \
  .reduceByKey(lambda v1,v2: v1+v2)
  .saveAsTextFile(output)
```

hadoop MapReduce

```
public class WordCount {
  public static void main(String[] args) throw
   Job job = new Job();
    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.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    boolean success = job.waitForCompletion(true);
    System.exit(success ? 0 : 1);
  }
}

public class WordMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {
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)
        context.write(new Text(word), new IntWritable(1));
      }
    }
  }
}

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

What is Apache Spark

- Apache Spark is a fast and general engine for large-scale data processing
- Written in Scala
  - Functional programming language that runs in a JVM
- Spark Shell
  - Interactive – for learning or data exploration
  - Python or Scala
- Spark Application
  - For large scale data processing
  - Python, Scala, or Java

Spark Shell

- The Spark Shell provides interactive data exploration

Python Shell: **pyspark**

```
$ pyspark

Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.0.0
      /_/

Using Python version 2.6.6 (r266:84292, Jan
22 2014 09:42:36)
SparkContext available as sc.

>>>
```

Scala Shell: **spark-shell**

```
$ spark-shell

Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.0.0
      /_/

Using Scala version 2.10.3 (Java HotSpot(TM)
64-Bit Server VM, Java 1.7.0_51)
Created spark context..
Spark context available as sc.

scala>
```

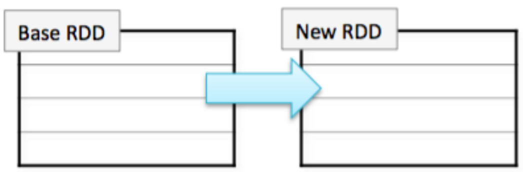
REPL: Read/Evaluate/Print Loop

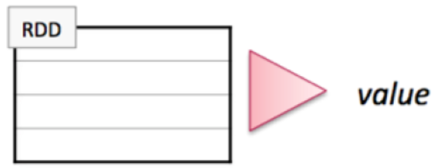
RDD (Resilient Distributed Dataset)

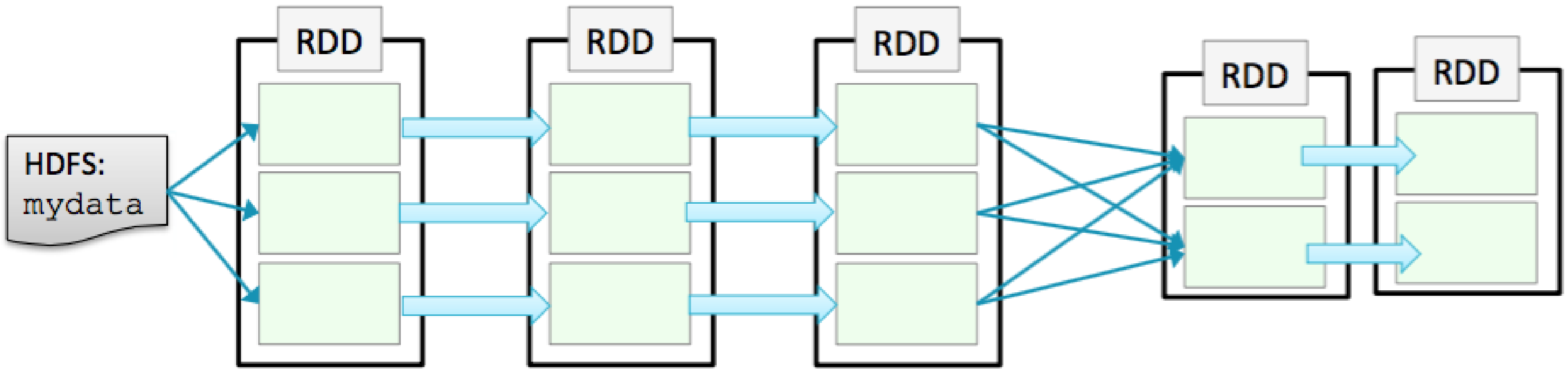
- **Resilient Distributed Datasets**
  - Collections of objects spread across a cluster, stored in RAM or on Disk
    - Analogous to HDFS but in memory
    - Still works efficiently on disks
  - Resilient – if data in memory is lost, it can be recreated
  - Distributed – stored in memory across the cluster
  - Dataset – initial data can come from a file or be created programmatically
- RDDs are the fundamental unit of data in Spark
- Most Spark programming consists of performing operations on RDDs
- Automatically rebuilt on failure

RDD Operations

- Transformations → create new RDDs
  - (e.g. map, filter, groupBy)
- Actions → returns value
  - (e.g. count, collect, save)

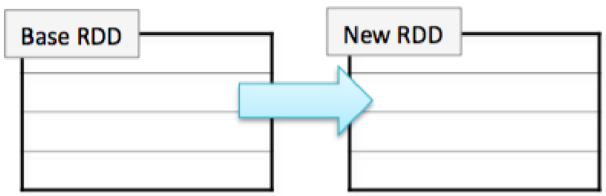






RDD Operations: Transformations

- Transformations create a new RDD from an existing one
- RDDs are immutable
  - Data in an RDD is never changed
  - Transform in sequence to modify the data as needed



- Some common transformations
  - map(function) – creates a new RDD by performing a function on each record in the base RDD
  - filter(function) – creates a new RDD by including or excluding each record in the base RDD according to a boolean function

RDD Operations

- map
- filter
- groupBy
- sort
- union
- join
- leftOuterJoin
- rightOuterJoin

- reduce
- count
- fold
- reduceByKey
- groupByKey
- cogroup
- cross
- zip

sample

take

first

partitionBy

mapWith

pipe

save ...

RDD Operations Explained

Loading messages from a log into memory and search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()

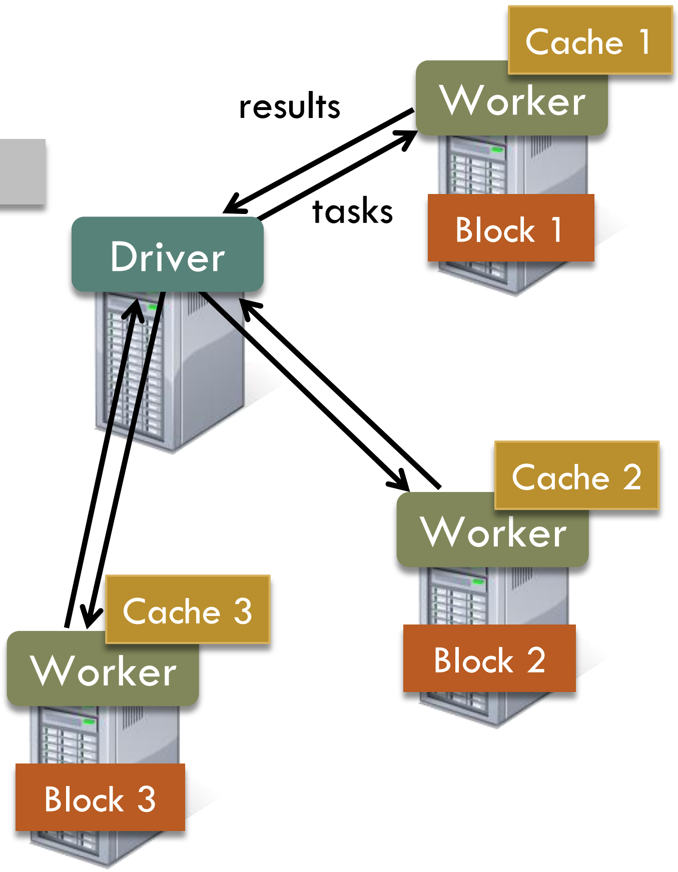
messages.filter(lambda s: "php" in s).count()

. . .
```

Base RDD

Transformed RDD

Action



Creating RDDs

Turn a Python collection into an RDD
> sc.parallelize([1, 2, 3])

Load text file from local FS, HDFS, or S3
> sc.textFile("file.txt")
> sc.textFile("directory/*.txt")
> sc.textFile("hdfs://namenode:9000/path/file")

Use existing Hadoop InputFormat (Java/Scala only)
> sc.hadoopFile(keyClass, valClass, inputFmt, conf)

Basic Transformations

- nums = sc.parallelize([1, 2, 3])

\# Pass each element through a function

- squares = nums.map(lambda x: x*x)   // {1, 4, 9}

\# Keep elements passing a predicate

- even = squares.filter(lambda x: x % 2 == 0) // {4}

\# Map each element to zero or more others

- nums.flatMap(lambda x: => range(x))
  - \# => {0, 0, 1, 0, 1, 2}

Basic Actions

```
> nums = sc.parallelize([1, 2, 3])

# Retrieve RDD contents as a local collection
> nums.collect() # => [1, 2, 3]

# Return first K elements
> nums.take(2)  # => [1, 2]

# Count number of elements
> nums.count()  # => 3

# Merge elements with an associative function
> nums.reduce(lambda x, y: x + y)  # => 6

# Write elements to a text file
> nums.saveAsTextFile("hdfs://file.txt")
```

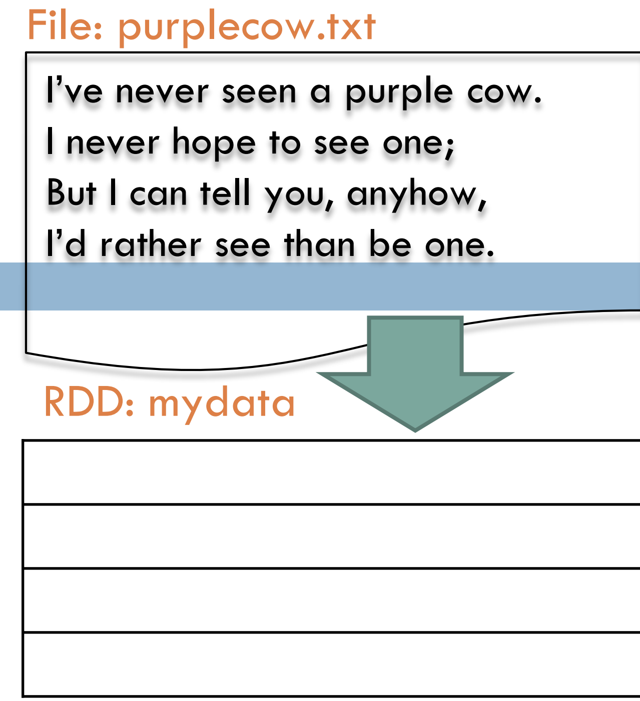
RDD Operations

File: purplecow.txt

I've never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I'd rather see than be one.

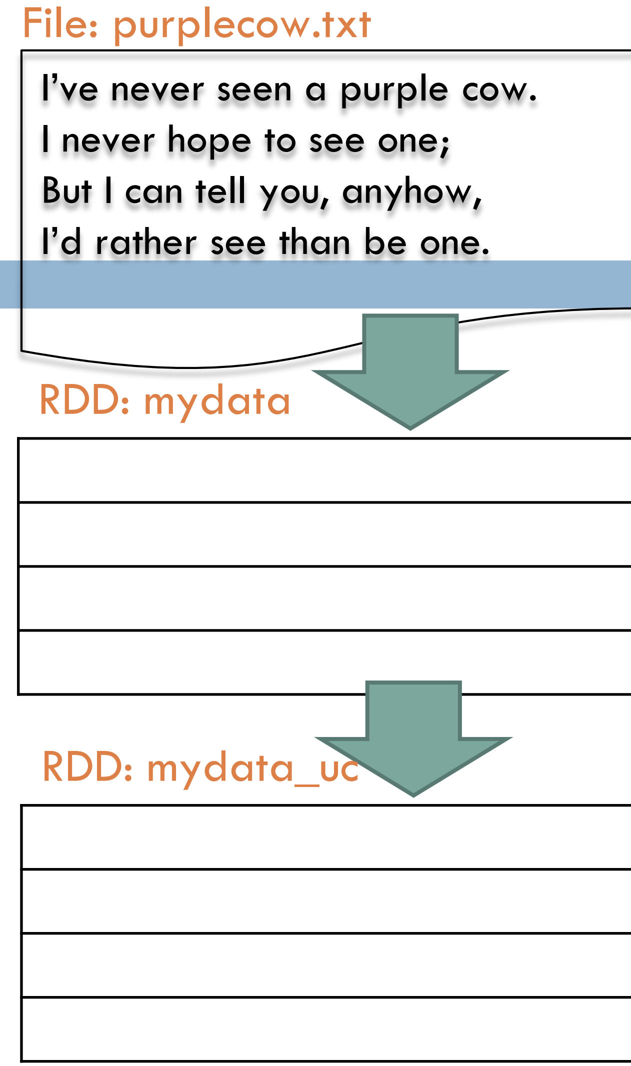
RDD Operations

- mydata = sc.textFile(“purplecow.txt”)



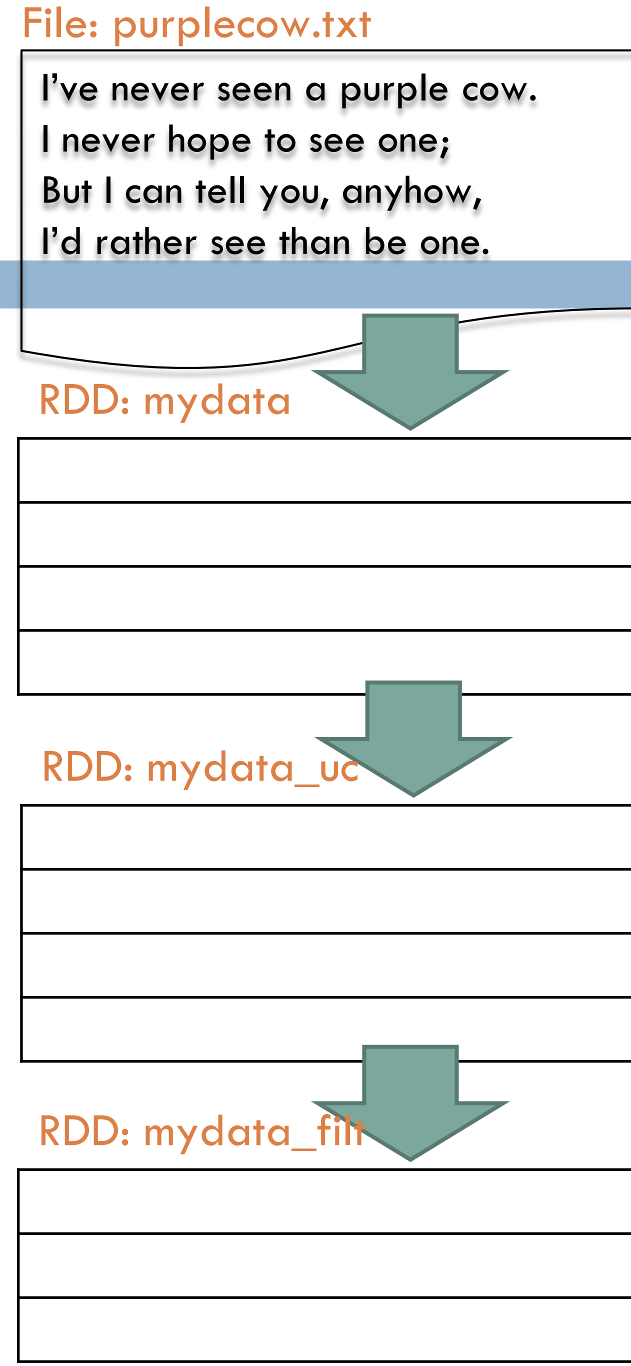
RDD Operations

- mydata = sc.textFile("purplecow.txt")
- mydata_uc = mydata.map(lambda line: line.upper())



RDD Operations

- mydata = sc.textFile("purplecow.txt")
- mydata_uc = mydata.map(lambda line: line.upper())
- mydata_filt = mydata_uc.filter(lambda line: line.startswith('I'))

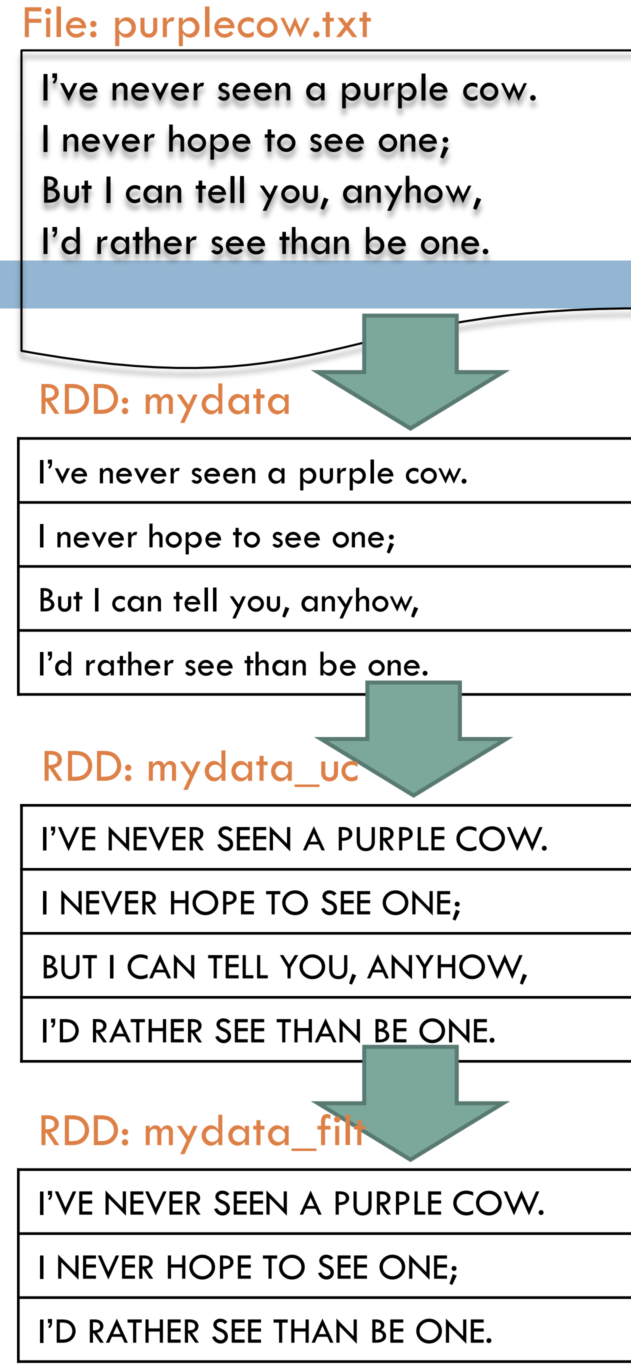


RDD Operations

- mydata = sc.textFile(“purplecow.txt”)
- mydata_uc = mydata.map(lambda line: line.upper())
- mydata_filt = mydata_uc.filter(lambda line: line.startswith(‘I’))
- mydata_filt.count()

Action

3



RDD Map/Reduce Operations

- MapReduce in Spark works on Pair RDDs
- Spark's "distributed reduce" transformations operate on RDDs of key-value pairs

Python:

```
pair = (a, b)
pair[0] # => a
pair[1] # => b
```

Creating Pair RDDs

- The first step in most workflows is to get the data into key/value form
  - What should the RDD be keyed on?
  - What is the value?
- Commonly used functions to create Pair RDDs
  - map
  - flatMap / flatMapValues
  - keyBy

Example: A Simple Pair RDD


```
friends = sc.textFile(file) \
        .map(lambda line: line.split('\t')) \
        .map(lambda fields: (fields[0], field[1]))
```

Amy	Jack
Amy	Lauren
Amy	Hans
Jack	Peter
Jason	Tony

(Amy, Jack)
(Amy, Lauren)
(Amy, Hans)
(Jack, Peter)
(Jason, Tony)

Example: Keying Friend Pairs by Friend ID


- ```
  friends = sc.textFile(file) \
          .keyBy(lambda line: line.split('\t')[0])
  ```

Amy	Jack
Amy	Lauren
Amy	Hans
Jack	Peter
Jason	Tony

(Amy, Amy Jack)
(Amy, Amy Lauren)
(Amy, Amy Hans)
(Jack, Jack Peter)
(Jason, Jason Tony)
  ```

Example: Pairs with Complex Values


```
friends = sc.textFile(file) \
        .map(lambda line: line.split('\t')) \
        .map(lambda fields: (fields[0], (fields[1], fields[2])))
```

```
USER1111 43.00589  -71.01320
USER2222 57.11234  -65.54698
USER3333 66.23324  -64.44612
USER4444 43.00123  -75.22257
```

(USER1111, (43.00589, -71.01320))
(USER2222, (57.11234, -65.54698))
(USER3333, (66.23324, -64.44612))
(USER4444, (43.00123, -75.22257))

Some Key-Value Operations

- pets = sc.parallelize([(“cat”, 1), (“dog”, 1), (“cat”, 2)])
- pets.reduceByKey(lambda x, y: x + y) # => {(cat, 3), (dog, 1)}
- pets.groupByKey() # => {(cat, [1, 2]), (dog, [1])}
- pets.sortByKey() # => {(cat, 1), (cat, 2), (dog, 1)}

reduceByKey also automatically implements combiners on the map side

Spark RDD: WordCount Example

the cat sat on the mat
the aardvark sat on the sofa

aardvark	1
cat	1
mat	1
on	2
sat	2
sofa	1
the	4

Spark RDD: WordCount Example

- counts = sc.textFile(text)

the cat cat on the mat
the aardvark sat on the sofa

Spark RDD: WordCount Example

- counts = sc.textFile(text)

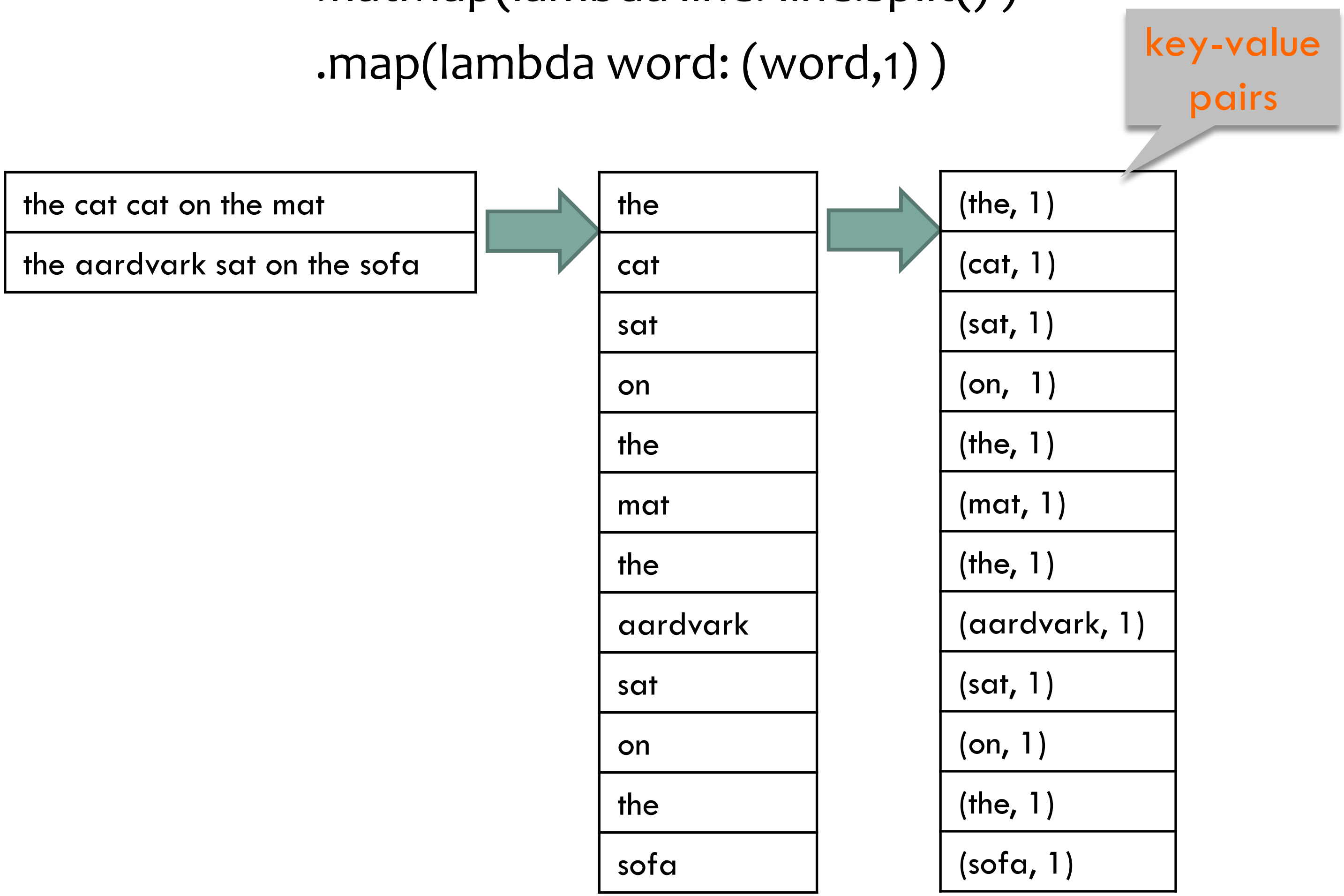
  .flatMap(lambda line: line.split())

the cat cat on the mat
the aardvark sat on the sofa

the
cat
sat
on
the
mat
the
aardvark
sat
on
the
sofa

Spark RDD: WordCount Example

- counts = sc.textFile(text)
  .flatMap(lambda line: line.split())
  .map(lambda word: (word,1))



Spark RDD: WordCount Example

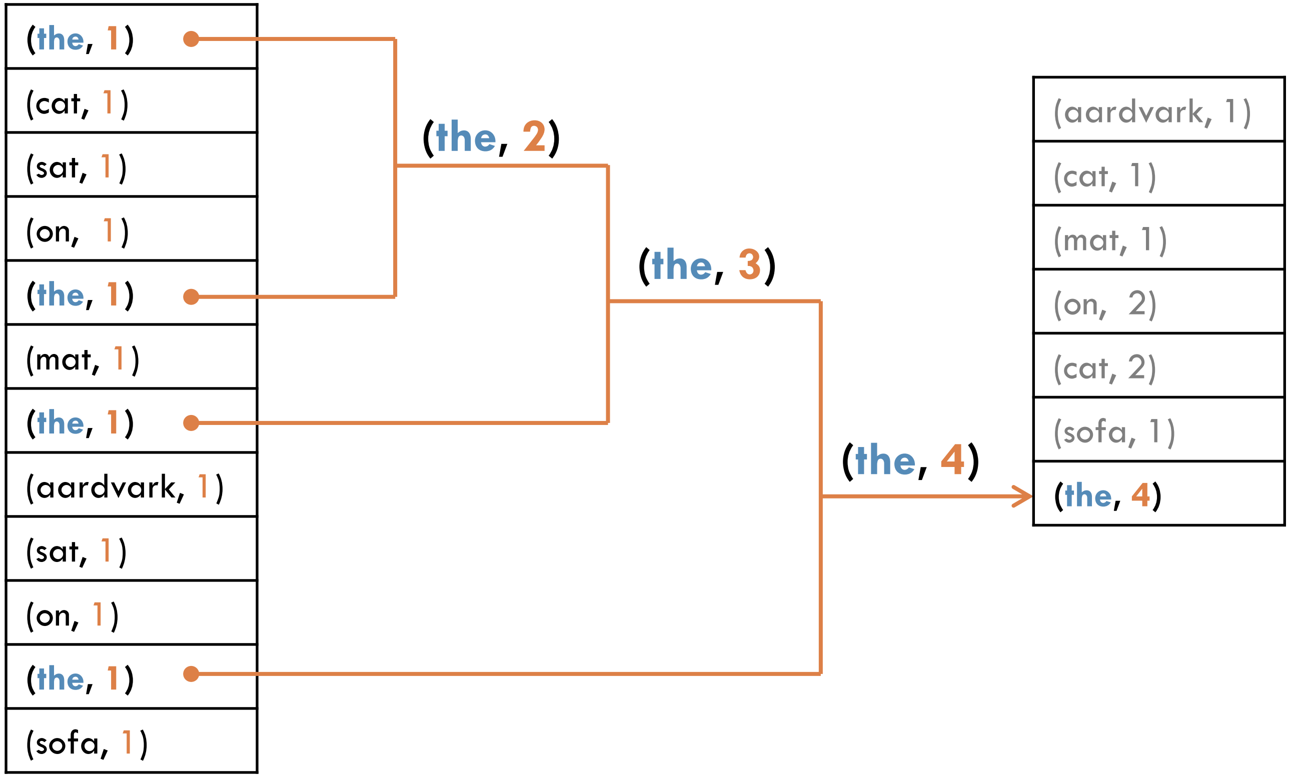
- counts = sc.textFile(text)

```
counts = sc.textFile(text)
        .flatMap(lambda line: line.split() )
        .map(lambda word: (word,1) )
        .reduceByKey(lambda v1,v2: v1+v2)
```

the cat cat on the mat	the	(the, 1)	(aardvark, 1)
the aardvark sat on the sofa	cat	(cat, 1)	(cat, 1)
	sat	(sat, 1)	(mat, 1)
	on	(on, 1)	(on, 2)
	the	(the, 1)	(cat, 2)
	mat	(mat, 1)	(sofa, 1)
	the	(the, 1)	(the, 4)
	aardvark	(aardvark, 1)	
	sat	(sat, 1)	
	on	(on, 1)	
	the	(the, 1)	
	sofa	(sofa, 1)	

Spark RDD: WordCount Example

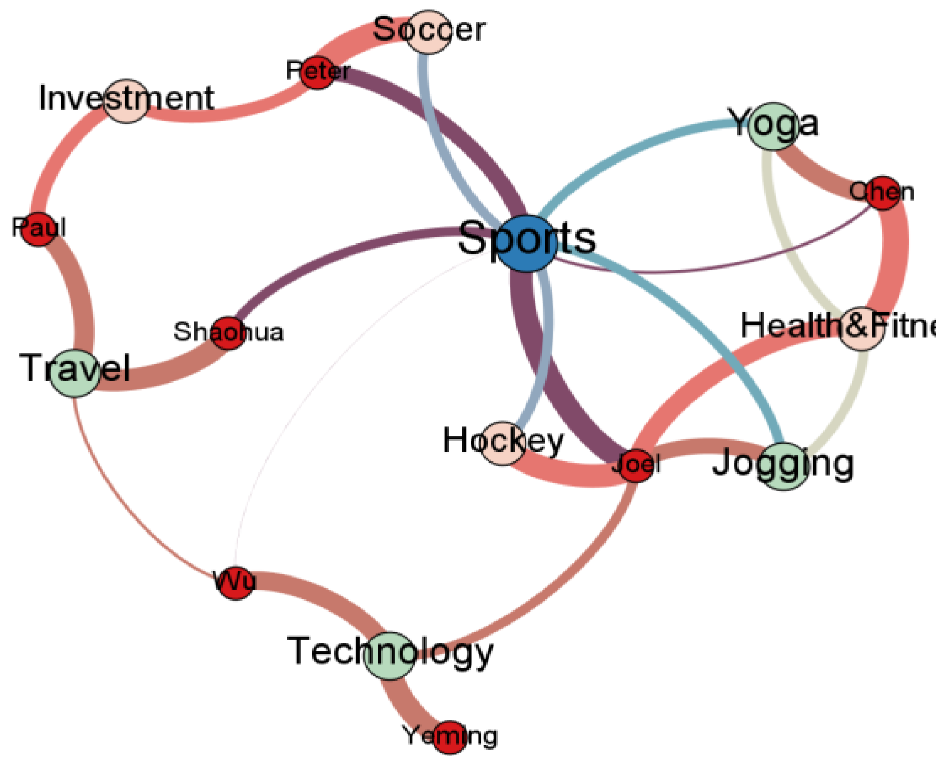
- ReduceByKey functions must be
  - Binary – combines values from two keys
  - Commutative → x+y = y+x
  - Associative → (x+y)+z = x+(y+z)



Graph Processing

The Graph phenomenon!

- Web Graph (Google)
- Social Graph (Facebook)
- Follower Graph (Twitter)
- Interest Graph (Pinterest!)
- Music/Location/Food Graphs



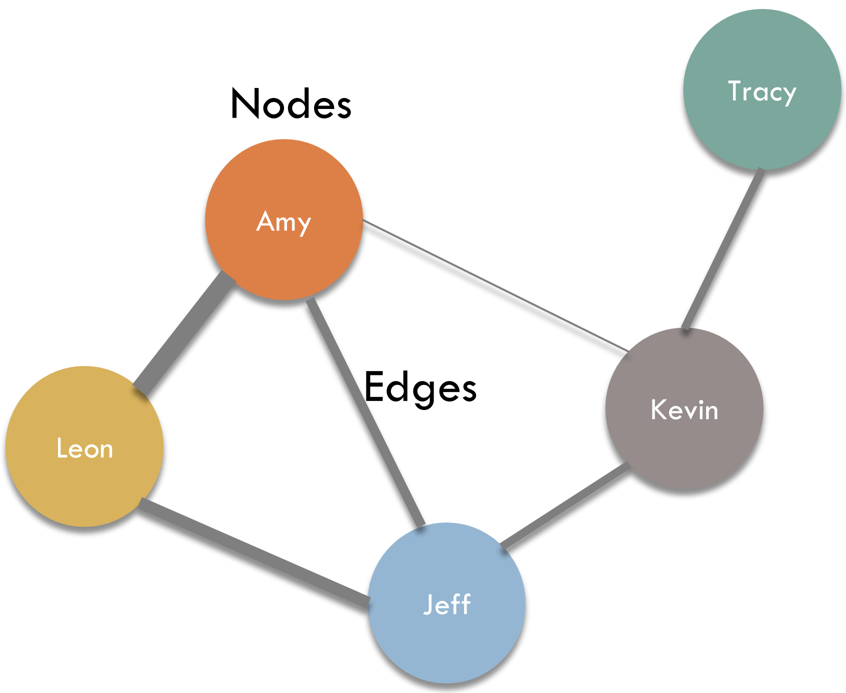
Graph Representation

	Friend				
User	Amy	Kevin	Jeff	Tracy	Leon
Amy		0.8	1.7		4
Kevin	0.8		1.3	1	
Jeff	1.7	1.3			2.2
Tracy		1			
Leon	4		2.2		

Matrix

Key	*Value*
Amy	*(Kevin,0.8), (Jeff, 1.7) (Leon,4)*
Kevin	*(Amy,0.8), (Jeff,1.3), (Tracy,1)*
Jeff	*(Amy, 1.7), (Kevin,1.3), (Leon,2.2)*
Tracy	*(Kevin,1)*
Leon	*(Amy,4), (Jeff,2.2)*

Adjacency List



Graph (undirected)

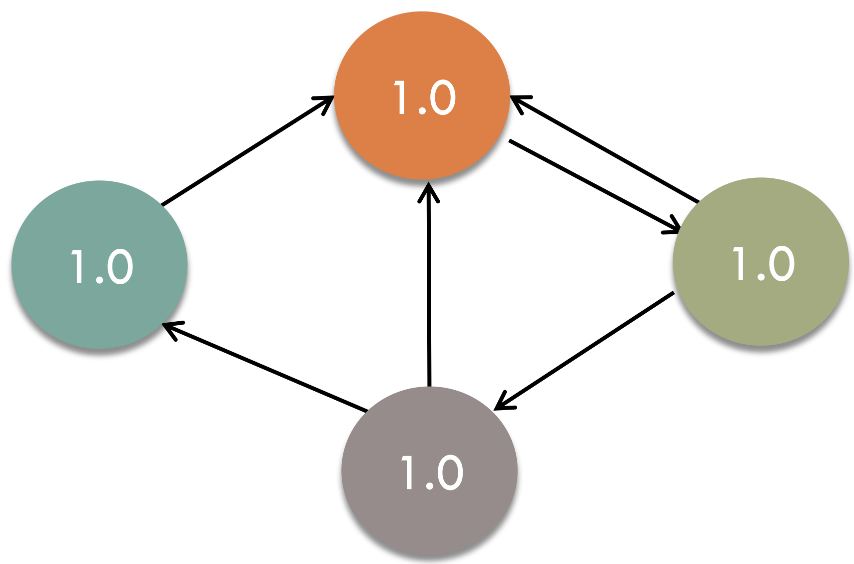
PageRank Algorithm

- PageRank gives web pages a ranking score on links from other pages
  - Links from many pages ➔ high rank
  - Link from a high-rank page ➔ high rank

Graph (directed)

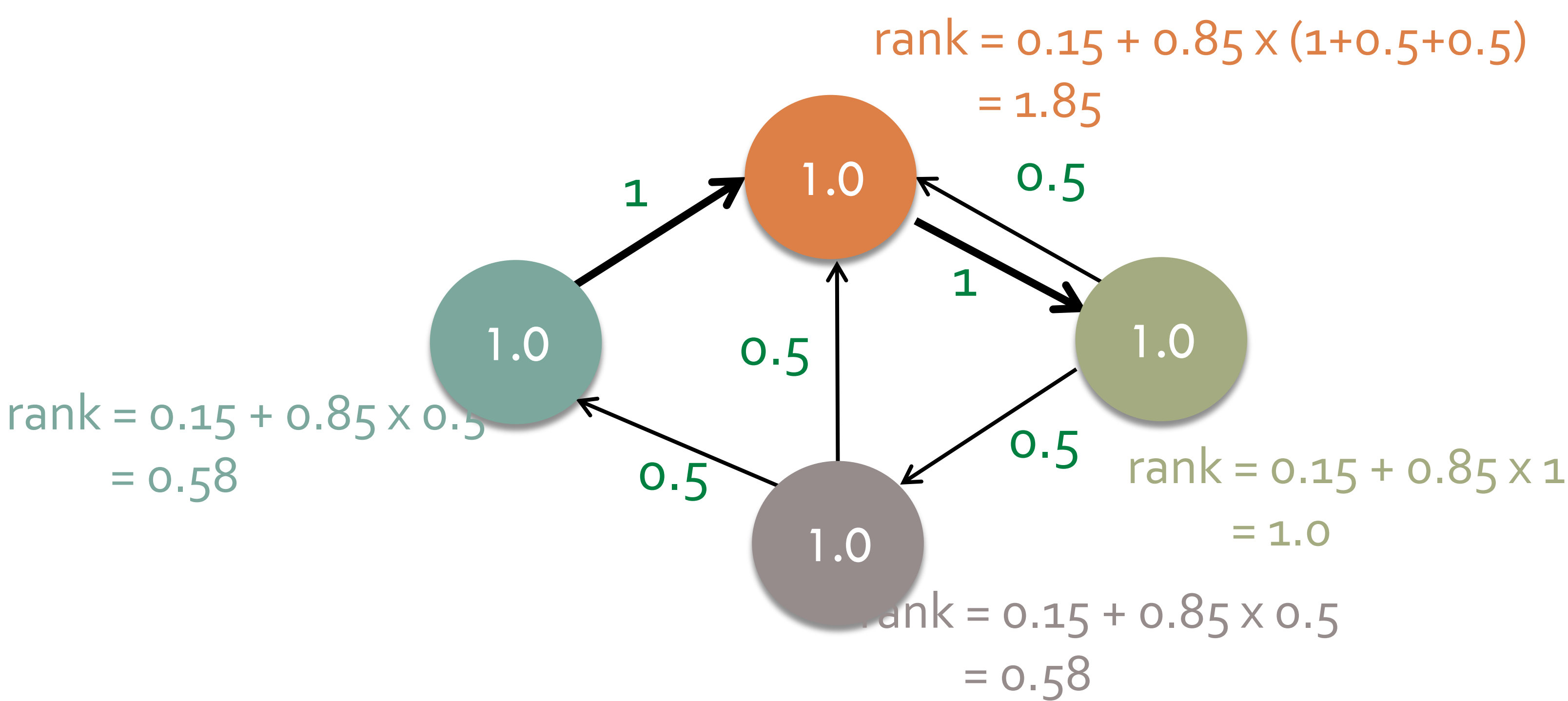
PageRank Explained – Initial State

1. Start each page at a rank of 1



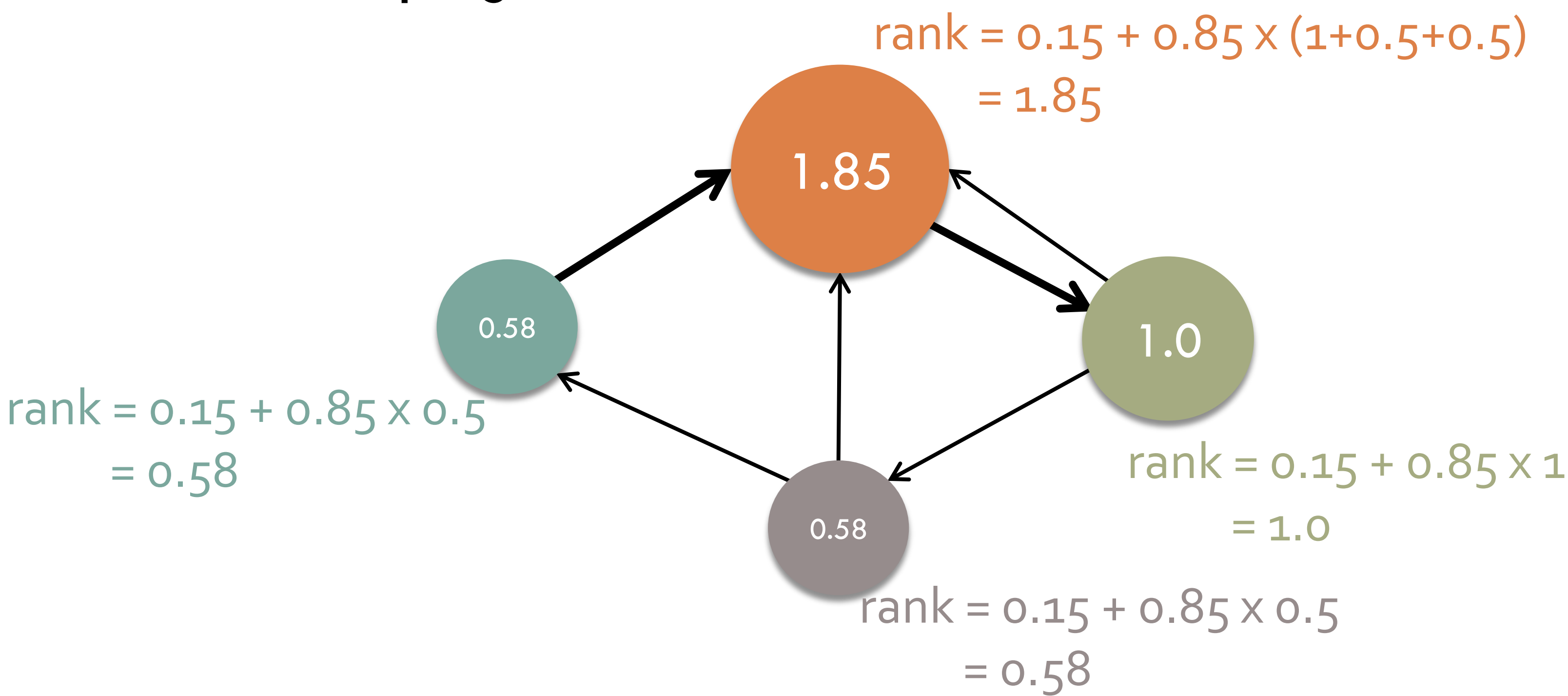
PageRank Explained – Iteration 1

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $contrib_p = rank_p / neighbors_p$ to its neighbors



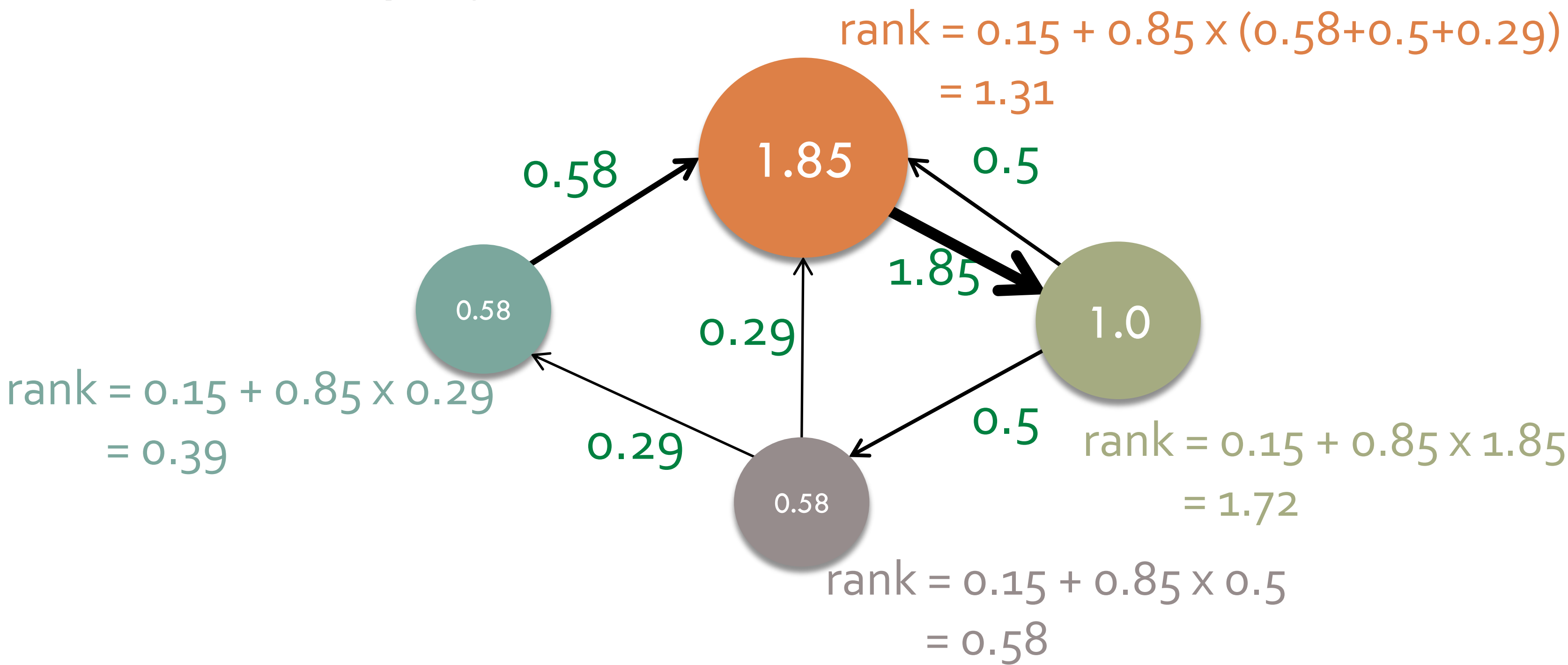
PageRank Explained – Iteration 1

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $contrib_p = rank_p \,/\, neighbors_p$ to its neighbors
3. **Set each page's rank to $0.15 + 0.85 \times$ contribs**



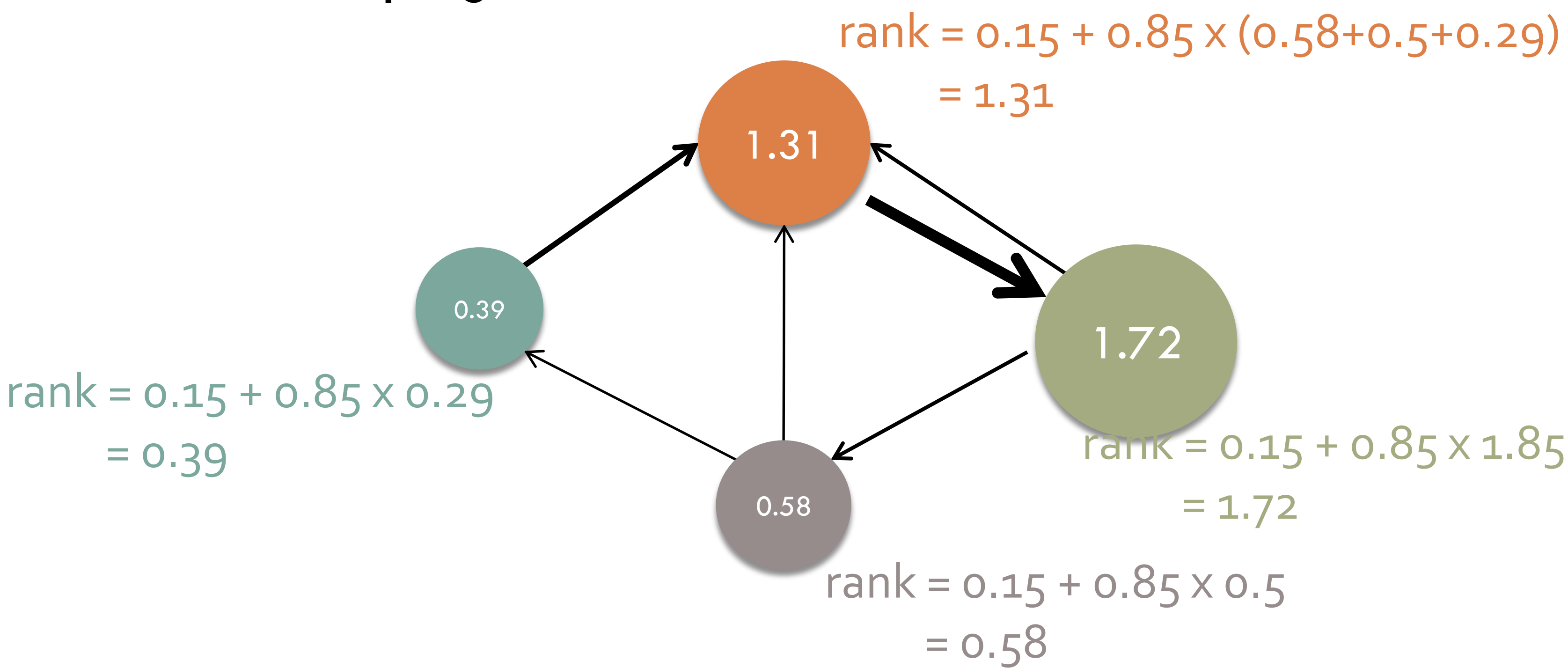
PageRank Explained – Iteration 2

1. Start each page at a rank of 1
2. On each iteration, have page p contribute
   $contrib_p = rank_p / neighbors_p$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



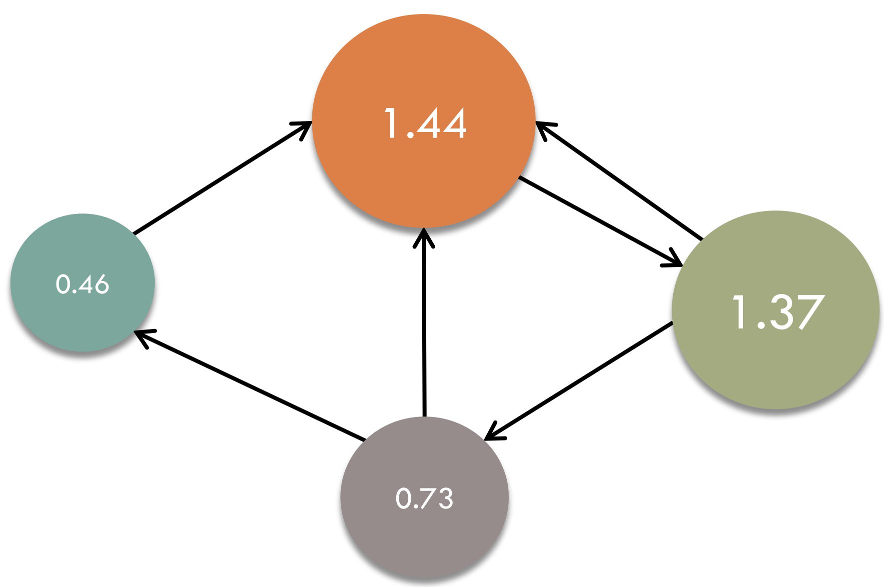
PageRank Explained – Iteration 2

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $contrib_p = rank_p \,/\, neighbors_p$ to its neighbors
3. **Set each page's rank to 0.15 + 0.85 × contribs**



PageRank Explained – Final State

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $contrib_p = rank_p / neighbors_p$ to its neighbors
3. **Set each page's rank to $0.15 + 0.85 \times$ contribs**



PageRank Implementation

- Page with Pig
  - Using PageRank to Detect Anomalies and Fraud in Healthcare (Hortonworks Blog Post)
    - (PART1) http://hortonworks.com/blog/using-pagerank-detect-anomalies-fraud-healthcare/
    - (PART2) http://hortonworks.com/blog/using-pagerank-to-detect-anomalies-and-fraud-in-healthcare-part2/
    - (PART3) http://hortonworks.com/blog/using-pagerank-to-detect-anomalies-and-fraud-in-healthcare-part3/
- Page with Spark
  - Next Lab

Getting Started with Spark

- Install Spark Standalone on Linux
- Cloudera/HDP Distributions
  - http://www.cloudera.com/content/cloudera/en/downloads/quickstart_vms/cdh-5-3-x.html

Spark Documentation

Spark 1.0.0 Documentation: http://spark.apache.org/docs/latest/

Spark Programming Guide (Scala, Python): http://spark.apache.org/docs/latest/programming-guide.html#overview

Spark Cassandra Connector - DataStax (github)

Spark HBase - lighting Spark with HBase (link)

Spark MLlib Documentation (link)

Spark GraphX Documentation (link)

(Spark) Spark Cluster Mode Overview (link)

(Spark) Running Spark on EC2 (link)

(Cloudera) Pig is Flying: Apache Pig on Apache Spark (link)

So many new things to learn ☺

- General Purpose Big Data Processing
  - Pig/Hive
  - Spark
- Specialized Tools
  - Graph: SparkX, Giraph, GraphLab
  - ML: Mahout, MLLib
  - Stream: Storm, Spark Streaming
  - NoSQL: Cassandra, Neo4j
  - Search: Solr, ElasticSearch

Big Data Landscape - Simplified

	Open Source	Commercial	Comments
Big Data Platform	Hadoop (MR, Pig, Hive etc.)	Cloudera, Hortonworks, MapR	Hadoop is going mainstream
	Spark	DataStax	Spark is HOT! considered as next-generation big data platform
		AWS	Elastic MapReduce (EMR) and EC2 from AWS is most popular among startups.
Machine Learning & Statistical Learning	Mahout		Mahout was one of the earliest ML libraries for MapReduce. It is being revamped to take advantage of Spark currently
	MLlib (Spark)		MLlib is Spark's machline learning library. It's written in Scala and also provides Python and Java API
	H2O		H2O is the latest buzzing big data machine learning tool, backed by 0xdata. It works with a Hadoop cluster but also works on Standalone cluster. It has an amazing lineup of algorithms and even supports Deep Learning. The GUI-based predictive analytics suites works like a charm
		SAS	SAS integration with Hadoop will be very powerful. Imaging writing your data steps that runs procedures on hadoop
		Revolution Analytics	Commercial version of open source R. Enterprise-class big data analytics capability
		Alpine	World's first code-free in-cluster web analytics platform to analyze big data and hadoop

Big Data Landscape - Simplified

	Open Source	Commercial	Comments
Graph Processing	***Giraph***		*Graph processing framework on top of Hadoop. Used extensively at Facebook for large-scale graph algorithms*
	GraphLab		*Developed at CMU by Dr. Carlos and his team. Superior graph processing performance. Building the tools to make data scientists' lives easier. Great as a standalone graph processing and machine learning tool but won't fit well into the existing hadoop cluster*
	GraphX (Spark)		*Graph processing on Spark platform*
Search	***Solr***		*Open source search server based on Lucene Java library*
	Elastic Search		*Open source search and analytics engine*
Stream Processing	***Storm***		*Real-time stream processing framework developed at Twitter. Most popular streaming processing tool*
	Spark Streaming		*Streaming processing on Spark. Less mature than Storm at the moment but growing rapidly*
Visualization	***d3.js***		*Fantastic javascript library for visualization*
		Tableu, Qlikview, Zoomdata	*Popular visualization tools widely adopted*
	Kibana		*Log and time series data visualization tool from Elasticsearch*

Choosing The Right "Big Data" Tools - Today

	Java	SAS	R	Python	Spark
Prototyping	*Weka, Java*	*SAS Base, SAS EG*	*R*	*Python*	*Spark/R*
Data Manipulation	*Hadoop, Pig/Hive*	*SAS Connector for Hadoop*	*RHadoop*	*Hadoop Streaming Pig/Hive*	*Spark*
Modeling	*Weka, Mahout*	*Enterprise Miner, SAS Hadoop*	*RHadoop*	*Hadoop Streaming*	*Mllib, GraphX*
Scoring	*Hadoop, Mahout*	*Enterprise Miner, SAS Base, Hadoop PMML*	*RHadoop*	*Hadoop Streaming, Pig*	*Spark*

Today's Lab

- Download Cloudera 5.3 Virtual Machine
  - http://www.cloudera.com/content/cloudera/en/downloads/quickstart_vms/cdh-5-3-x.html
- Read Hortonwork's Blog Post on PageRank with Pig
- Complete Hive/Pig Labs (if you haven't)
- Finish Pig Assignments