

Introduction to Apache Spark

Patrick Wendell - Databricks



What is Spark?

Fast and Expressive Cluster Computing Engine Compatible with Apache Hadoop

Up to **10x** faster on disk,
100x in memory

Efficient

- General execution graphs
- In-memory storage

2-5x less code

Usable

- Rich APIs in Java, Scala, Python
- Interactive shell



The Spark Community

March 27th 2010 - November 30th 2013

Commits to master, excluding merge commits

Contribution Type: **Commits** ▾



Today's Talk

- The Spark programming model
- Language and deployment choices
- Example algorithm (PageRank)

Key Concept: RDD's

Write programs in terms of operations on distributed datasets

Resilient Distributed Datasets

- Collections of objects spread across a cluster, stored in RAM or on Disk
- Built through parallel transformations
- Automatically rebuilt on failure

Operations

- Transformations (e.g. map, filter, groupBy)
- Actions (e.g. count, collect, save)



Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

Transformed RDD

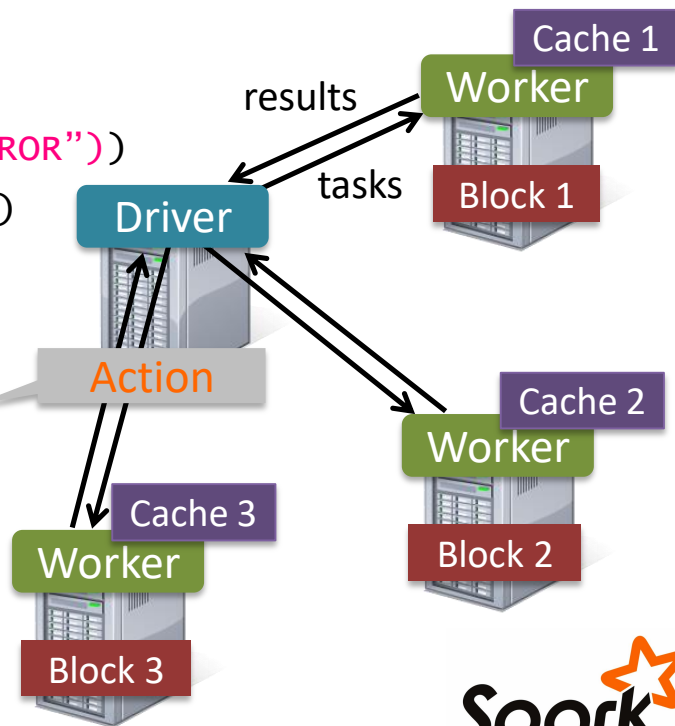
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
```

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

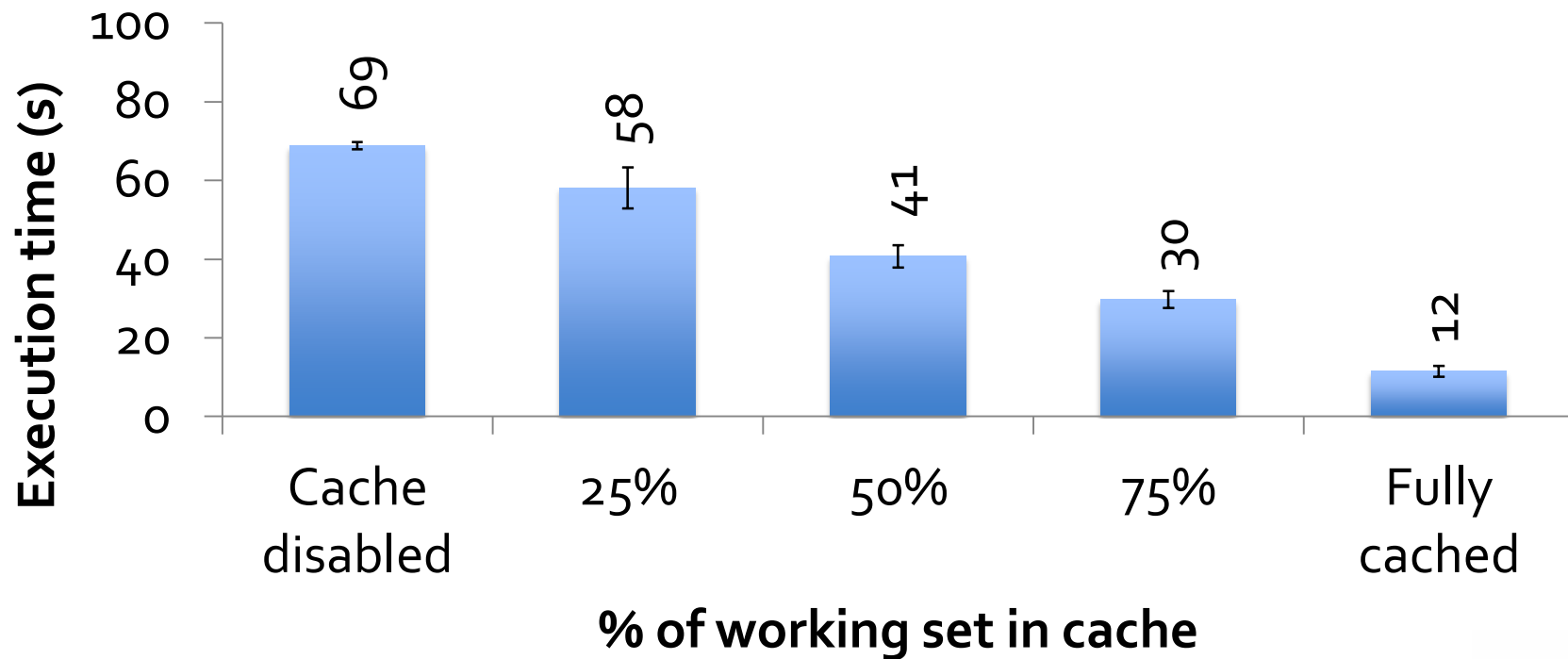
...

Full-text search of Wikipedia

- 60GB on 20 EC2 machine
- 0.5 sec vs. 20s for on-disk



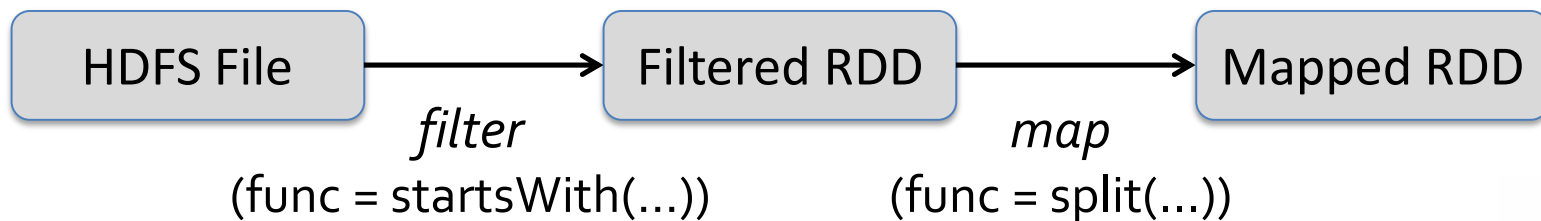
Scaling Down



Fault Recovery

RDDs track *lineage* information that can be used to efficiently recompute lost data

```
msgs = textFile.filter(lambda s: s.startswith("ERROR"))  
                .map(lambda s: s.split("\t")[2])
```



Programming with RDD's

SparkContext

- Main entry point to Spark functionality
- Available in shell as variable **SC**
- In standalone programs, you'd make your own (see later for details)

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}
```

Range object (sequence of numbers 0, 1, ..., x-1)

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")
```

Working with Key-Value Pairs

Spark's "distributed reduce" transformations operate on RDDs of key-value pairs

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

Scala: `val pair = (a, b)`
`pair._1 // => a`
`pair._2 // => b`

Java: `Tuple2 pair = new Tuple2(a, b);`
`pair._1 // => a`
`pair._2 // => b`



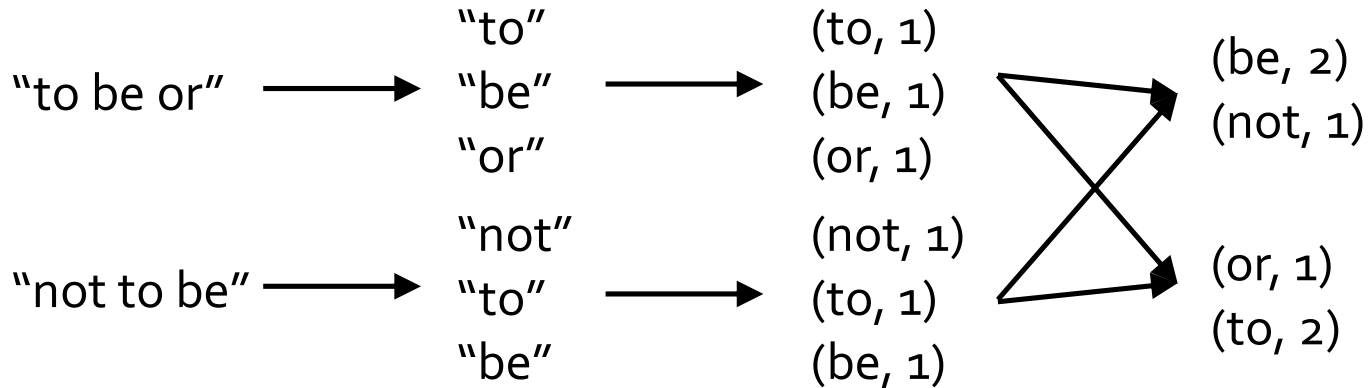
Some Key-Value Operations

```
> 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

Example: Word Count

```
> lines = sc.textFile("hamlet.txt")  
> counts = lines.flatMap(lambda line: line.split(" "))  
                    .map(lambda word => (word, 1))  
                    .reduceByKey(lambda x, y: x + y)
```



Other Key-Value Operations

- > `visits = sc.parallelize([("index.html", "1.2.3.4"), ("about.html", "3.4.5.6"), ("index.html", "1.3.3.1")])`
- > `pageNames = sc.parallelize([("index.html", "Home"), ("about.html", "About")])`
- > `visits.join(pageNames)`
 - # ("index.html", ("1.2.3.4", "Home"))
 - # ("index.html", ("1.3.3.1", "Home"))
 - # ("about.html", ("3.4.5.6", "About"))
- > `visits.cogroup(pageNames)`
 - # ("index.html", (["1.2.3.4", "1.3.3.1"], ["Home"]))
 - # ("about.html", (["3.4.5.6"], ["About"]))

Setting the Level of Parallelism

All the pair RDD operations take an optional second parameter for number of tasks

- > words.reduceByKey(lambda x, y: x + y, 5)
- > words.groupByKey(5)
- > visits.join(pageViews, 5)

Using Local Variables

Any external variables you use in a closure will automatically be shipped to the cluster:

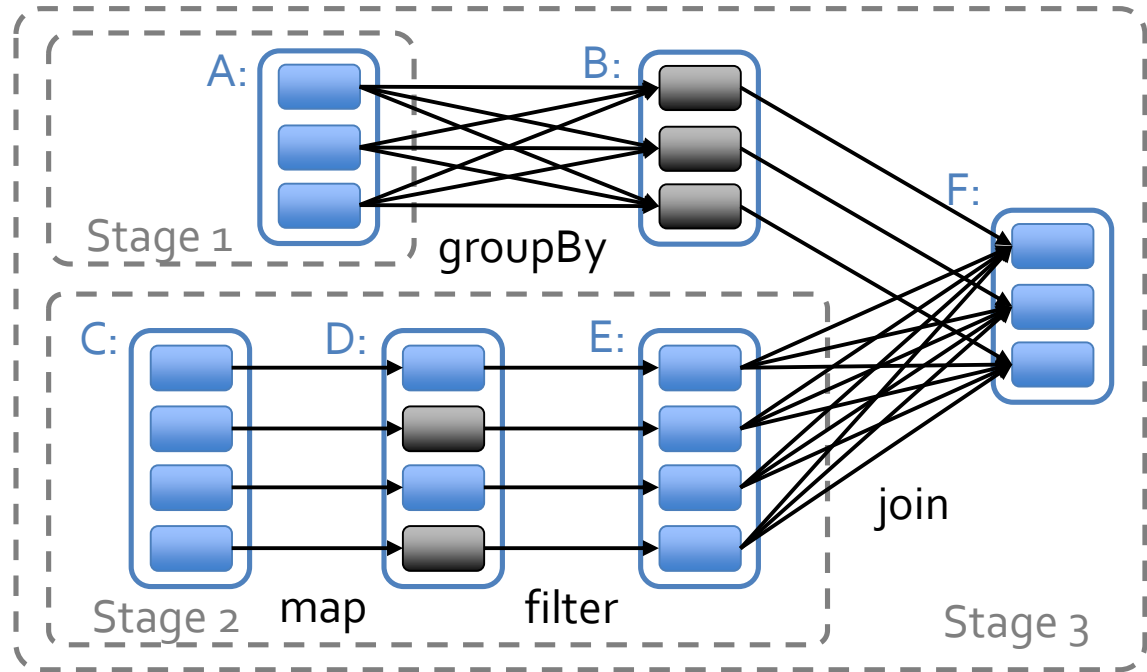
```
> query = sys.stdin.readline()
> pages.filter(lambda x: query in x).count()
```

Some caveats:

- Each task gets a new copy (updates aren't sent back)
- Variable must be Serializable / Pickle-able
- Don't use fields of an outer object (ships all of it!)

Under The Hood: DAG Scheduler

- General task graphs
- Automatically pipelines functions
- Data locality aware
- Partitioning aware to avoid shuffles



= RDD



= cached partition

More RDD Operators

- map
- filter
- groupBy
- sort
- union
- join
- leftOuterJoin
- rightOuterJoin
- reduce
- count
- fold
- reduceByKey
- groupByKey
- cogroup
- cross
- zip
- sample
- take
- first
- partitionBy
- mapWith
- pipe
- save

How to Run Spark

Language Support

Python

```
lines = sc.textFile(...)
lines.filter(lambda s: "ERROR" in s).count()
```

Scala

```
val lines = sc.textFile(...)
lines.filter(x => x.contains("ERROR")).count()
```

Java

```
JavaRDD<String> lines = sc.textFile(...);
lines.filter(new Function<String, Boolean>() {
    Boolean call(String s) {
        return s.contains("error");
    }
}).count();
```

Standalone Programs

- Python, Scala, & Java

Interactive Shells

- Python & Scala

Performance

- Java & Scala are faster due to static typing
- ...but Python is often fine



... or a Standalone Application

```
import sys
from pyspark import SparkContext

if __name__ == "__main__":
    sc = SparkContext("local", "wordCount", sys.argv[0],
None)
    lines = sc.textFile(sys.argv[1])

    counts = lines.flatMap(lambda s: s.split(" ")) \
        .map(lambda word: (word, 1)) \
        .reduceByKey(lambda x, y: x + y)

    counts.saveAsTextFile(sys.argv[2])
```

Create a SparkContext

Scala

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._

val sc = new SparkContext("url", "name", "sparkHome", Seq("app.jar"))
```

Java

```
import org.apache.
JavaSparkContext sc = new JavaSparkContext(
    "masterUrl", "name", "sparkHome", new String[] {"app.jar"});
```

Cluster URL, or local / local[N]

App name

Spark install path on cluster

List of JARs with app code (to ship)

Python

```
from pyspark import SparkContext

sc = SparkContext("masterUrl", "name", "sparkHome", ["library.py"])
```



Add Spark to Your Project

- Scala / Java: add a Maven dependency on

groupId: org.spark-project

artifactId: spark-core_2.10

version: 0.9.0

- Python: run program with our pyspark script

Administrative GUIs

<http://<Standalone Master>:8080> (by default)

The image shows two browser windows. The left window, titled 'Spark Master at spark://mbp-2.local:7077', displays the Spark Master's status. The right window, titled 'Spark shell - Spark Stages', displays the Spark Stages page for a specific application.

Spark Master at spark://mbp-2.local:7077

URL: spark://mbp-2.local:7077
Workers: 3
Cores: 24 Total, 24 Used
Memory: 45.0 GB Total, 1536.0 MB Used
Applications: Running, 0 Completed

Workers

Id
worker-20131102231645-192.168.1.106-56789
worker-20131102231657-192.168.1.106-56801
worker-20131102231705-192.168.1.106-56806

Running Applications

ID	Name
app-20131202231712-0000	Spark shell

Spark shell - Spark Stages

localhost:4040/stages/

Stages Storage Environment Executors

Spark Stages

Total Duration: 3.8 m
Scheduling Mode: FIFO
Active Stages: 0
Completed Stages: 2
Failed Stages: 0

Active Stages (0)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read
----------	-------------	-----------	----------	------------------------	--------------

Completed Stages (2)

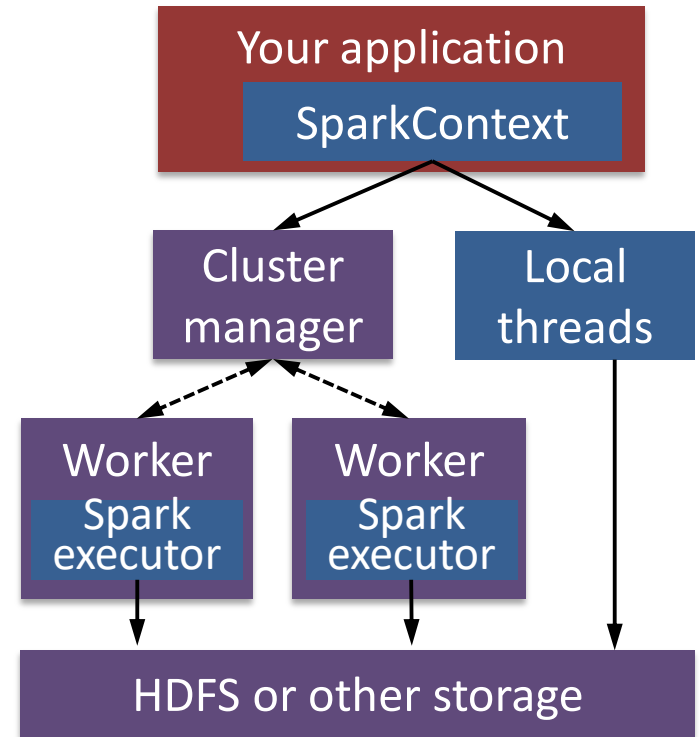
Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle
0	count at <console>:13	2013/12/02 21:07:55	83 ms	<div style="width: 100%; background-color: #0070C0; color: white; text-align: center;">2/2</div>	754.0 B
1	reduceByKey at <console>:13	2013/12/02 21:07:55	345 ms	<div style="width: 100%; background-color: #0070C0; color: white; text-align: center;">2/2</div>	

Failed Stages (0)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read
----------	-------------	-----------	----------	------------------------	--------------

Software Components

- Spark runs as a library in your program (1 instance per app)
- Runs tasks locally or on cluster
 - Mesos, YARN or standalone mode
- Accesses storage systems via Hadoop InputFormat API
 - Can use HBase, HDFS, S3, ...



EXAMPLE APPLICATION: PAGERANK

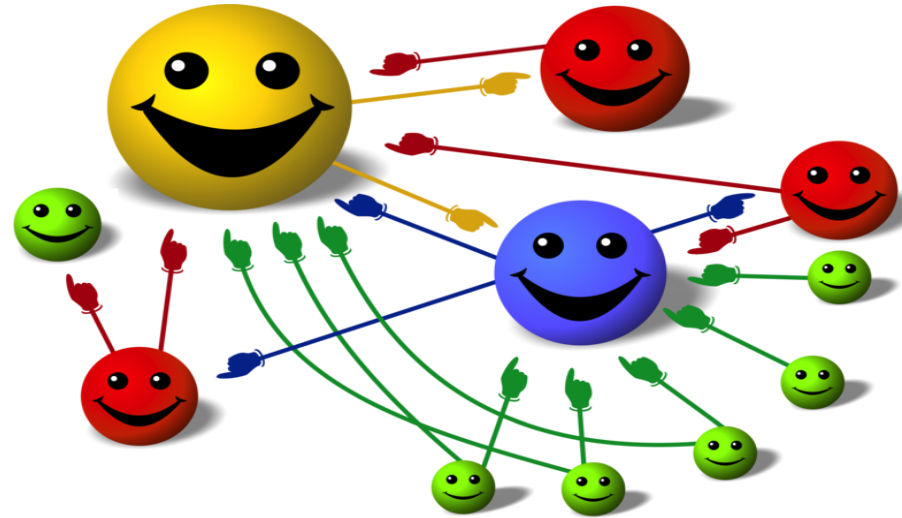
Example: PageRank

- Good example of a more complex algorithm
 - Multiple stages of map & reduce
- Benefits from Spark's in-memory caching
 - Multiple iterations over the same data

Basic Idea

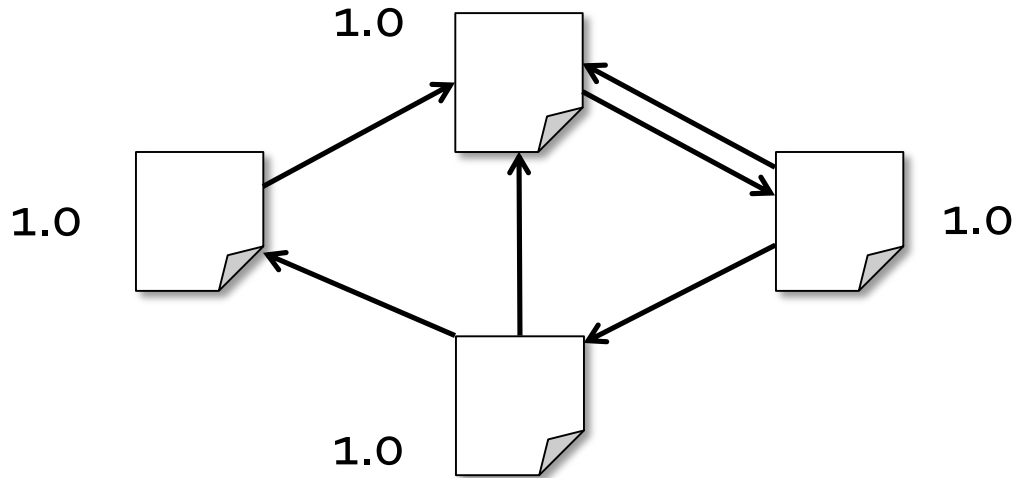
Give pages ranks (scores) based on links to them

- Links from many pages → high rank
- Link from a high-rank page → high rank



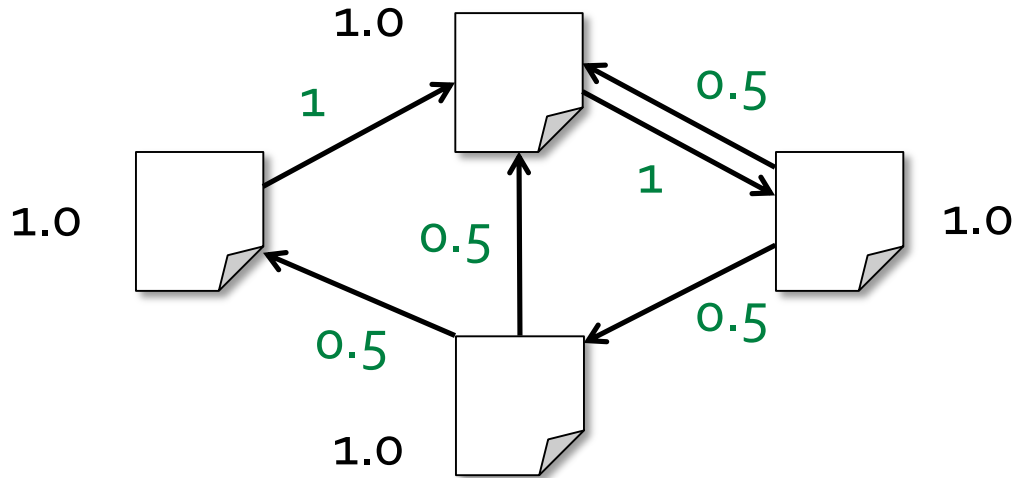
Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $\text{rank}_p / |\text{neighbors}_p|$ to its neighbors
3. Set each page's rank to $0.15 + 0.85 \times \text{contribs}$



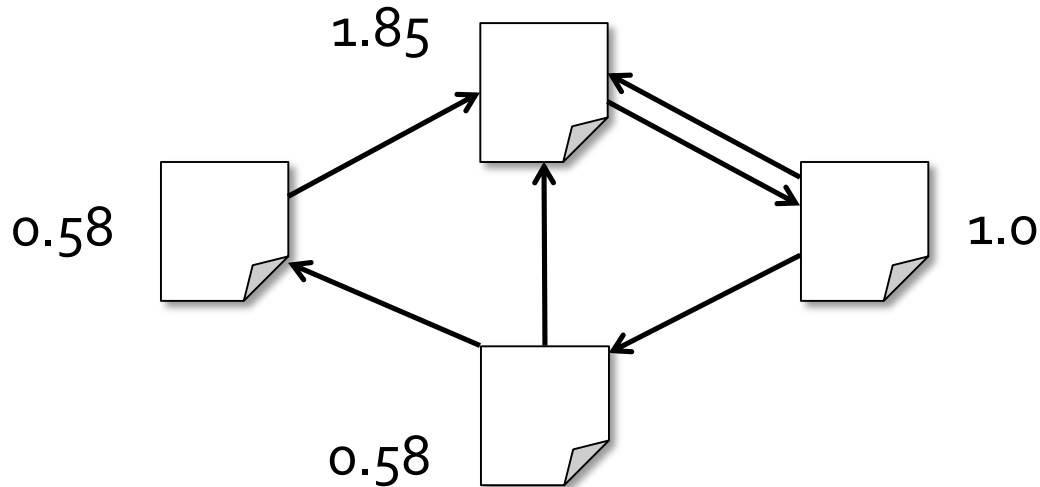
Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $\text{rank}_p / |\text{neighbors}_p|$ to its neighbors
3. Set each page's rank to $0.15 + 0.85 \times \text{contribs}$



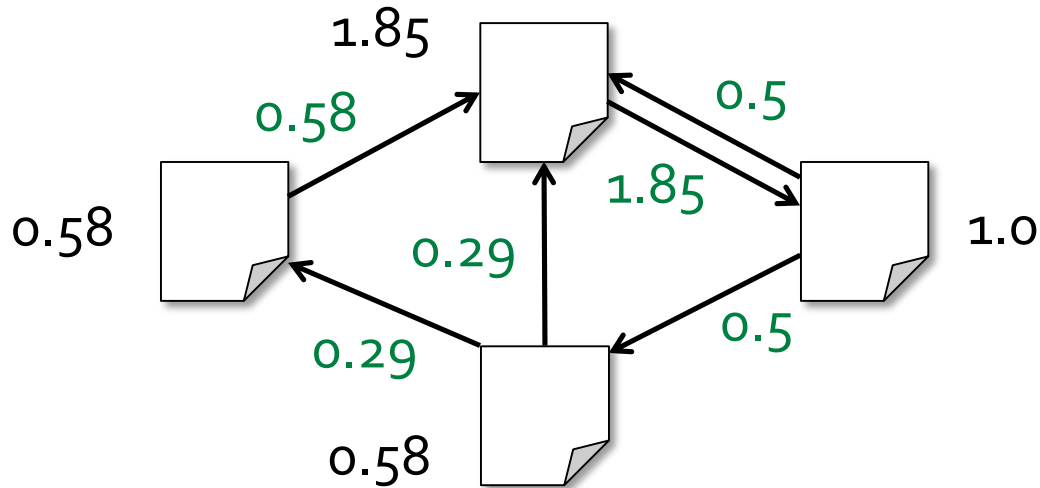
Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $\text{rank}_p / |\text{neighbors}_p|$ to its neighbors
3. Set each page's rank to $0.15 + 0.85 \times \text{contribs}$



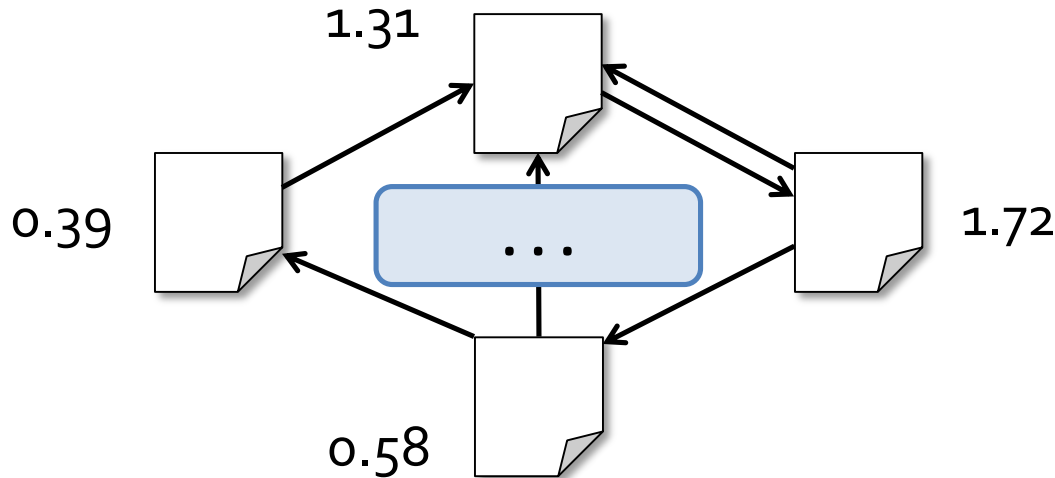
Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $\text{rank}_p / |\text{neighbors}_p|$ to its neighbors
3. Set each page's rank to $0.15 + 0.85 \times \text{contribs}$



Algorithm

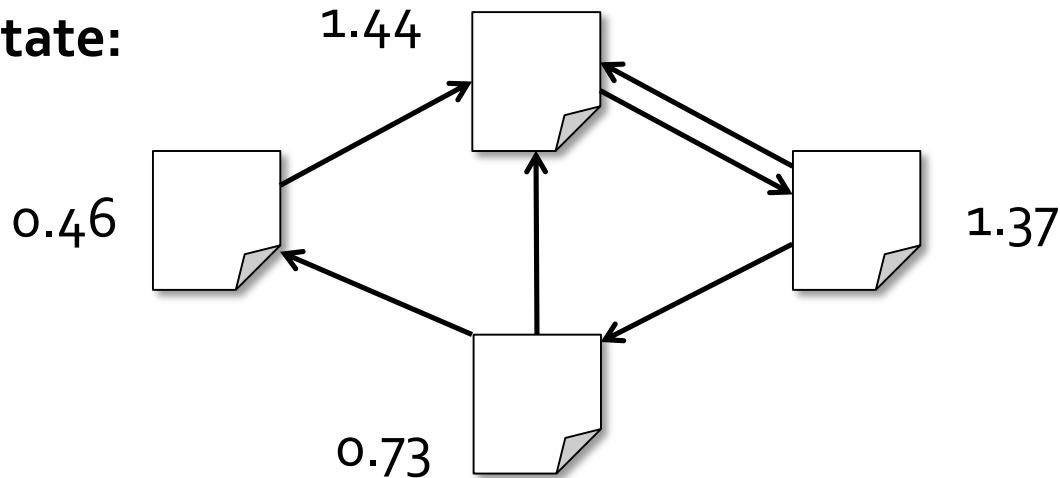
1. Start each page at a rank of 1
2. On each iteration, have page p contribute $\text{rank}_p / |\text{neighbors}_p|$ to its neighbors
3. Set each page's rank to $0.15 + 0.85 \times \text{contribs}$



Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $\text{rank}_p / |\text{neighbors}_p|$ to its neighbors
3. Set each page's rank to $0.15 + 0.85 \times \text{contribs}$

Final state:

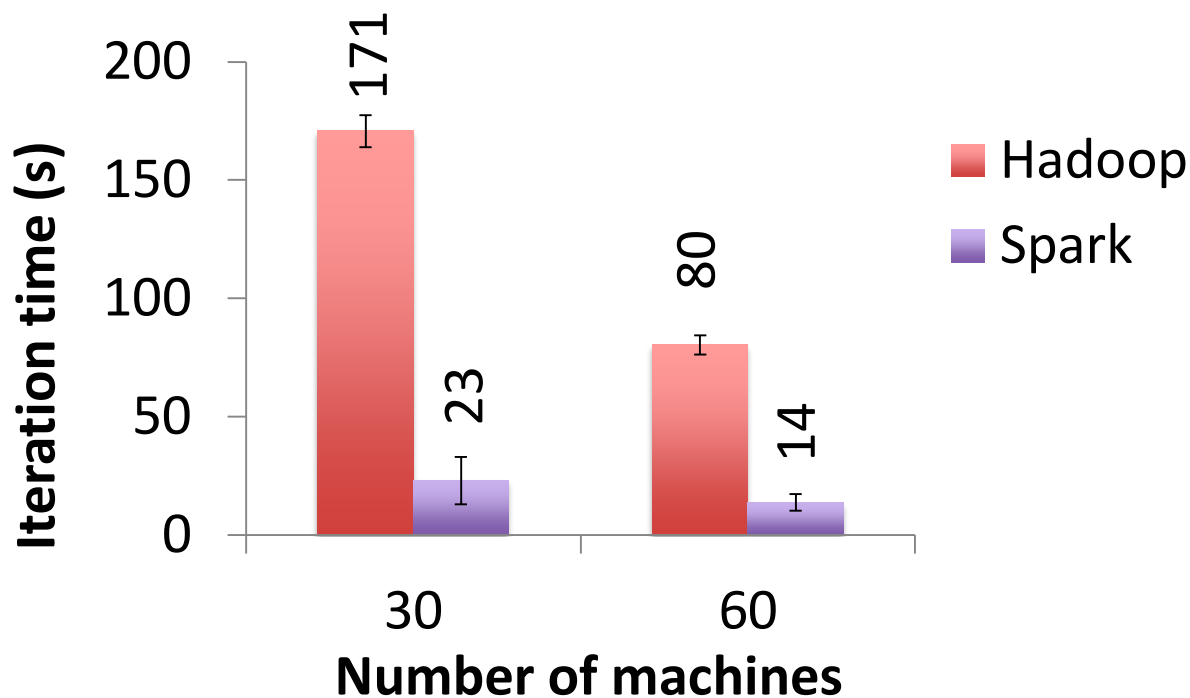


Scala Implementation

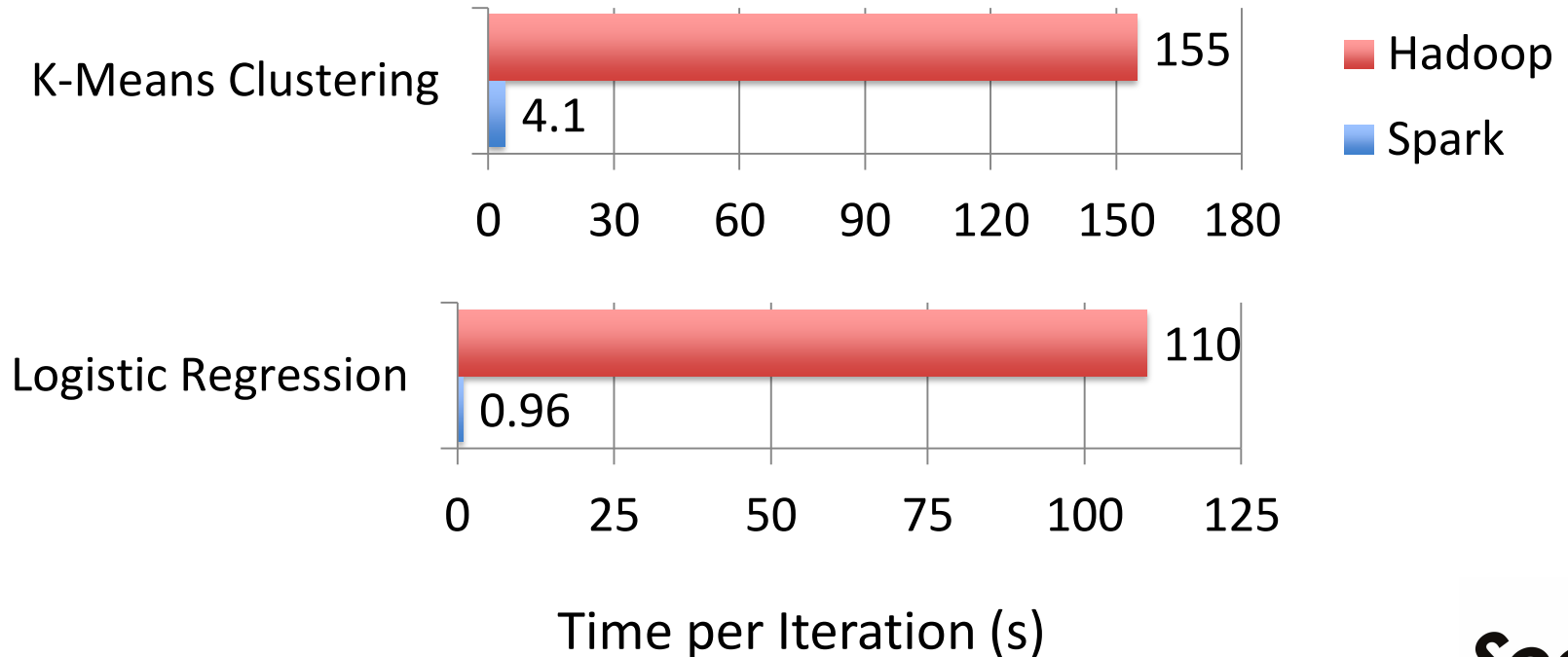
```
val links = // load RDD of (url, neighbors) pairs
var ranks = // load RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  val contribs = links.join(ranks).flatMap {
    case (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }
  ranks = contribs.reduceByKey(_ + _)
    .mapValues(0.15 + 0.85 * _)
}
ranks.saveAsTextFile(...)
```

PageRank Performance



Other Iterative Algorithms



CONCLUSION

Conclusion

- Spark offers a rich API to make data analytics *fast*: both fast to write and fast to run
- Achieves 100x speedups in real applications
- Growing community with 25+ companies contributing

Get Started

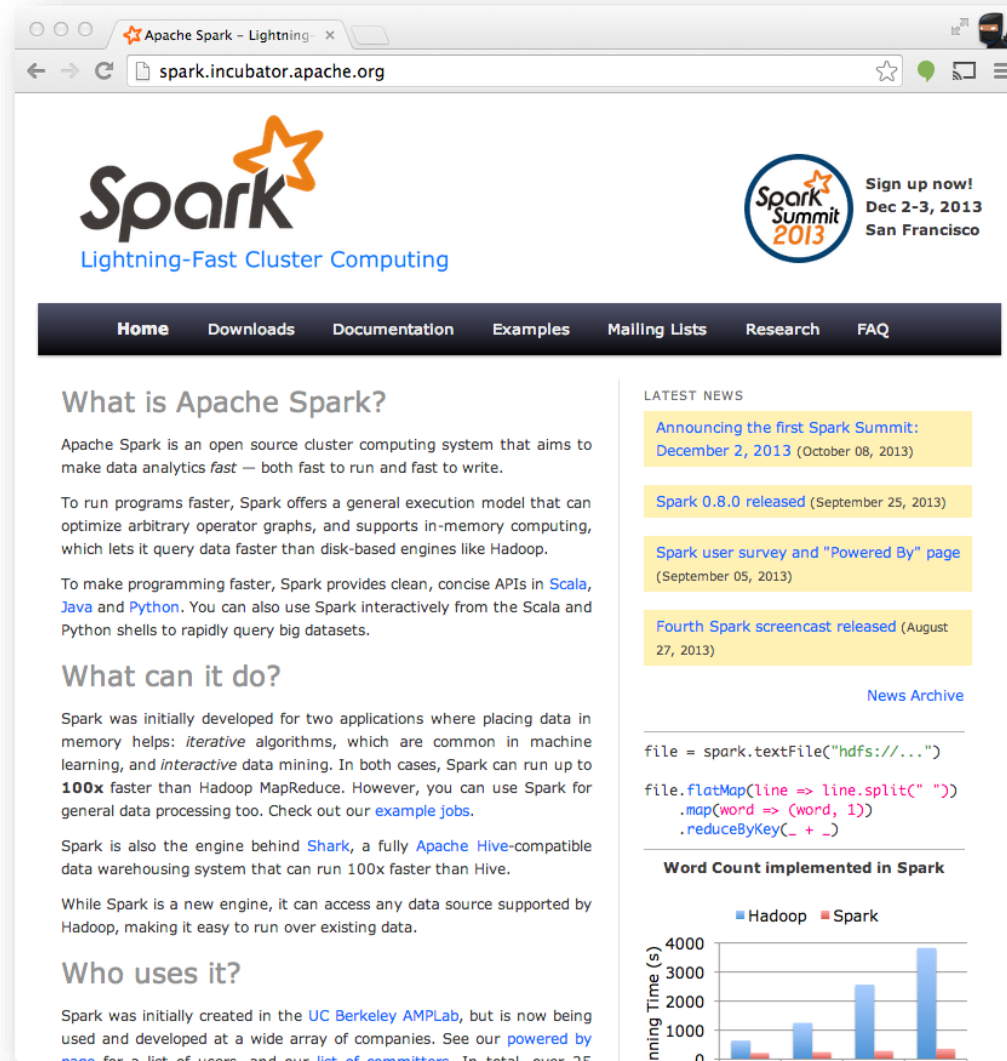
Up and Running in a Few Steps

- Download
- Unzip
- Shell

Project Resources

- Examples on the Project Site
- Examples in the Distribution
- Documentation

<http://spark.incubator.apache.org>



The screenshot shows the Apache Spark website homepage. The browser address bar displays "spark.incubator.apache.org". The page features the Spark logo (a star) and the tagline "Lightning-Fast Cluster Computing". A navigation menu includes links for Home, Downloads, Documentation, Examples, Mailing Lists, Research, and FAQ. The main content area is titled "What is Apache Spark?" and describes it as an open source cluster computing system. It highlights that Spark is 100x faster than Hadoop MapReduce. A sidebar on the right contains "LATEST NEWS" with links to announcements like "Announcing the first Spark Summit: December 2, 2013" and "Spark 0.8.0 released". Below the news is a "Word Count implemented in Spark" section with a bar chart comparing Hadoop and Spark performance. The chart shows Spark's execution time is significantly lower than Hadoop's for the same task.

Apache Spark - Lightning

spark.incubator.apache.org

Spark

Lightning-Fast Cluster Computing

Sign up now!
Dec 2-3, 2013
San Francisco

Home Downloads Documentation Examples Mailing Lists Research FAQ

What is Apache Spark?

Apache Spark is an open source cluster computing system that aims to make data analytics *fast* — both fast to run and fast to write.

To run programs faster, Spark offers a general execution model that can optimize arbitrary operator graphs, and supports in-memory computing, which lets it query data faster than disk-based engines like Hadoop.

To make programming faster, Spark provides clean, concise APIs in [Scala](#), [Java](#) and [Python](#). You can also use Spark interactively from the Scala and Python shells to rapidly query big datasets.

What can it do?

Spark was initially developed for two applications where placing data in memory helps: *iterative* algorithms, which are common in machine learning, and *interactive* data mining. In both cases, Spark can run up to **100x** faster than Hadoop MapReduce. However, you can use Spark for general data processing too. Check out our [example jobs](#).

Spark is also the engine behind [Shark](#), a fully [Apache Hive](#)-compatible data warehousing system that can run 100x faster than Hive.

While Spark is a new engine, it can access any data source supported by Hadoop, making it easy to run over existing data.

Who uses it?

Spark was initially created in the [UC Berkeley AMPLab](#), but is now being used and developed at a wide array of companies. See our [powered by page](#) for a list of users, and our [list of committers](#). In total, over 25

LATEST NEWS

[Announcing the first Spark Summit: December 2, 2013](#) (October 08, 2013)

[Spark 0.8.0 released](#) (September 25, 2013)

[Spark user survey and "Powered By" page](#) (September 05, 2013)

[Fourth Spark screencast released](#) (August 27, 2013)

[News Archive](#)

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

Word Count implemented in Spark

■ Hadoop ■ Spark

Task	Hadoop Time (s)	Spark Time (s)
1	~1000	~500
2	~1500	~750
3	~2500	~1250
4	~3500	~1750

Datasets And Dataframes

- <https://spark.apache.org/docs/latest/sql-programming-guide.html>

Dataset

- A *Dataset* is a distributed collection of data.
 - Like RDDs: Strong typing, ability to use powerful lambda functions
 - Plus the benefits of Spark SQL's optimized execution engine.
 - A Dataset can be constructed from JVM objects and then manipulated using functional transformations (map, flatMap, filter, etc.).
 - The Dataset API is available in Scala and Java.
 - Python does not have the support for the Dataset API. But due to Python's dynamic nature, many of the benefits of the Dataset API are already available
 - i.e. you can access the field of a row by name naturally (row.columnName).
 - The case for R is similar to Python

DataSets

- Datasets are similar to RDDs, however, instead of using Java serialization or Kryo **they use a specialized [Encoder](#) to serialize the objects for processing or transmitting over the network.**
- While both encoders and standard serialization are responsible for turning an object into bytes, encoders are code generated dynamically and use a format that **allows Spark to perform many operations like filtering, sorting and hashing without deserializing the bytes back into an object**
- Data sets also expose more internals to query planning (expressions, fields, etc)

Creating Datasets

- See code here:
 - <https://spark.apache.org/docs/latest/sql-getting-started.html#creating-datasets>

Dataframes

- A *DataFrame* is a *Dataset* organized into named columns.
 - It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood.
- DataFrames can be constructed from a wide array of sources such as: structured data files, tables in Hive, external databases, or existing RDDs.
- The DataFrame API is available in Scala, Java, Python, and R.
 - In Scala and Java, a DataFrame is represented by a Dataset of Rows.
 - In the Scala API, DataFrame is simply a type alias of Dataset[Row].
 - While, in Java API, users need to use Dataset<Row> to represent a DataFrame.

Creating Dataframes

```
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;

Dataset<Row> df = spark.read().json("examples/src/main/resources/people.json");

// Displays the content of the DataFrame to stdout
df.show();
// +----+-----+
// | age|  name|
// +----+-----+
// |null|Michael|
// | 30|  Andy|
// | 19| Justin|
// +----+-----+
```

Creating Dataframes

```
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;

Dataset<Row> df = spark.read().json("examples/src/main/resources/people.json");

// Displays the content of the DataFrame to stdout
df.show();
// +----+-----+
// | age|  name|
// +----+-----+
// |null|Michael|
// | 30|  Andy|
// | 19| Justin|
// +----+-----+
```

RDDs or Datasets and Dataframes?

- Probably Datasets and Dataframes
- Finer grained expressiveness allows more fully decoupled DAG for scheduler
 - This means more opportunities for parallelism.