
Large Scale Distributed Data Science using Apache Spark



James G. Shanahan^{1, 2} and Liang Dai^{1, 3}

¹*NativeX and iSchool*, ²*UC Berkeley, CA*, ³*UC Santa Cruz*

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

**ROOM: Level 1 - Room 5 & 6,
Hilton Hotel
488 George Street, Sydney, NSW 2000
Time: 10:20 – 12:00**

August 10, 2015, KDD 2015, Sydney

-
- NOTE: This is a draft of the slides

Mini Survey to calibrate tutorial pace

- Never used used Map-Reduce
- Have written jobs in Map-Reduce (E.g., Hadoop)
 - Expert
 - Novice
- What is Hadoop shuffle? Spark Shuffle?
- Totally new to Spark (never used it before)
- Python
- Python Notebooks
- R?
- Installed Spark?
- Got Python notebooks running with Spark prompt?

What can you hope to take away

- **What is Spark**
 - Spark is an open-source cluster computing framework for big data.
 - It has emerged as the next generation big data processing engine, overtaking Hadoop MapReduce which helped ignite the big data revolution.
 - This is certainly true for some tasks in particular machine learning.
 - Spark maintains MapReduce's linear scalability and fault tolerance, but extends it in a few important ways (faster, developer friendly, all-in-one system for big data)
- **Install Spark**
- **How to program in Spark**
- **Machine Learning in Spark**
- **Case studies**
- **Beyond the current version of spark**

Tutorial Outline

- **Part 1: Introduction**
 - Welcome Survey
 - Install Spark
 - Background and motivation
- **Part 2: Spark Intro and basics**
 - fundamental Spark concepts, including Spark Core, data frames, the Spark Shell, Spark Streaming, Spark SQL and vertical libraries such as MLlib and GraphX;
- **Part 3: Machine learning in Spark**
 - will focus on hands-on algorithmic design and development with Spark developing algorithms from scratch such as linear regression, logistic regression, graph processing algorithms such as pagerank/shortest path, etc.
- **Part 4: Wrap up**
 - Spark 1.5 and beyond
 - Summary

Part 1

- **Part 1: Introduction**

- Welcome Survey
- Install Spark
- Background and Motivation
 - Big Data Science
 - Functional Programming
 - Poorman's Map-Reduce (to dividing and conquering)

Part 2: Spark Intro and Basics

- **Part 2: Spark Intro and basics**

- Base RDD
- Fault tolerance (and lineage)
- Transformations and Actions
- Persistance
- Animated Example
- Pair RDDs
- Word count example

Part 3: Machine Learning in Spark

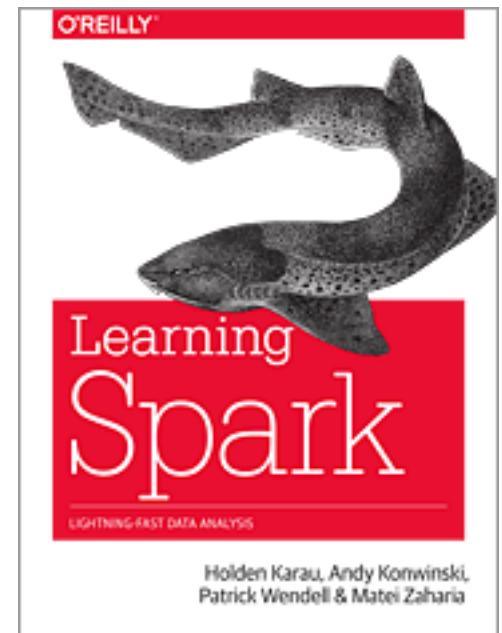
- **Data Frames**
- **MLLib**
- **Write your own algos**
 - Linear regression (Ridge and Lasso)
 - Logistic Regression
- **Pipelines**
- **R and Spark**
 - Linear Regression example
- **Graphs in Spark**
- **Case studies**
 - Flight delay
 - Mobile advertising
 - Social Networks

Tutorial Outline

- **Part 1: Introduction**
 - Welcome Survey
 - Install Spark
 - Background and motivation
- **Part 2: Spark Intro and basics**
 - fundamental Spark concepts, including Spark Core, data frames, the Spark Shell, Spark Streaming, Spark SQL and vertical libraries such as MLlib and GraphX;
- **Part 3: Machine learning in Spark**
 - will focus on hands-on algorithmic design and development with Spark developing algorithms from scratch such as linear regression, logistic regression, graph processing algorithms such as pagerank/shortest path, etc.
- **Part 4: Wrap up**
 - Spark 1.5 and beyond
 - Summary

Reference material

- **Book: Learning Spark: Lightning-Fast Big Data Analysis**
 - By Holden Karau, Andy Konwinski, Patrick Wendell, Matei Zaharia
 - Chapters 2, 3, and 4, and beyond
- <https://spark.apache.org/docs/latest/programming-guide.html>



Spark Online Resources

- <http://spark.apache.org/research.html>
- [Spark 1.4.1 released \(Jul 15, 2015\)](#)
- [Spark Summit 2015 Videos Posted\(Jun 29, 2015\)](#)
- **Reza Zadeh's presentation on Machine Learning in Spark @ Spark Summit 2015 in San Francisco (assumes strong understanding of programming in Spark)**

Full Day tutorial at CIKM 2015 in Melbourne, Monday, October 19, 2015

The screenshot shows a web browser window with the URL <http://www.cikm-2015.org/workshops-and-tutorials.php> in the address bar. The page header features a yellow banner with the text "69 days left until the Conference". Below the banner is the CIKM 2015 logo, which consists of a grid of blue squares overlaid by several colorful, abstract, flame-like or spark-like shapes in red, orange, purple, and green. To the right of the logo, the text "CIKM2015" is written in large, bold, black letters, followed by "19–23 October 2015 Melbourne Australia". Below this text are social media icons for Facebook and Twitter. The background of the page features a collage of images, including a close-up of a bird's head and a building.

Large Scale Distributed Data Science using Apache Spark

Full day

James G. Shanahan & Liang Dai



Join Us!

To apply, scan
the QR code.



About NativeX

NativeX is creating the leading ad technology for games. For developers who want to monetize with advertising that really works, NativeX is the marketing technology platform that is reinventing in-app advertising. We create more effective and engaging ad experiences that enable developers to build successful businesses around their apps. Here at NativeX, our data pipelines process tens of terabytes of data each day.

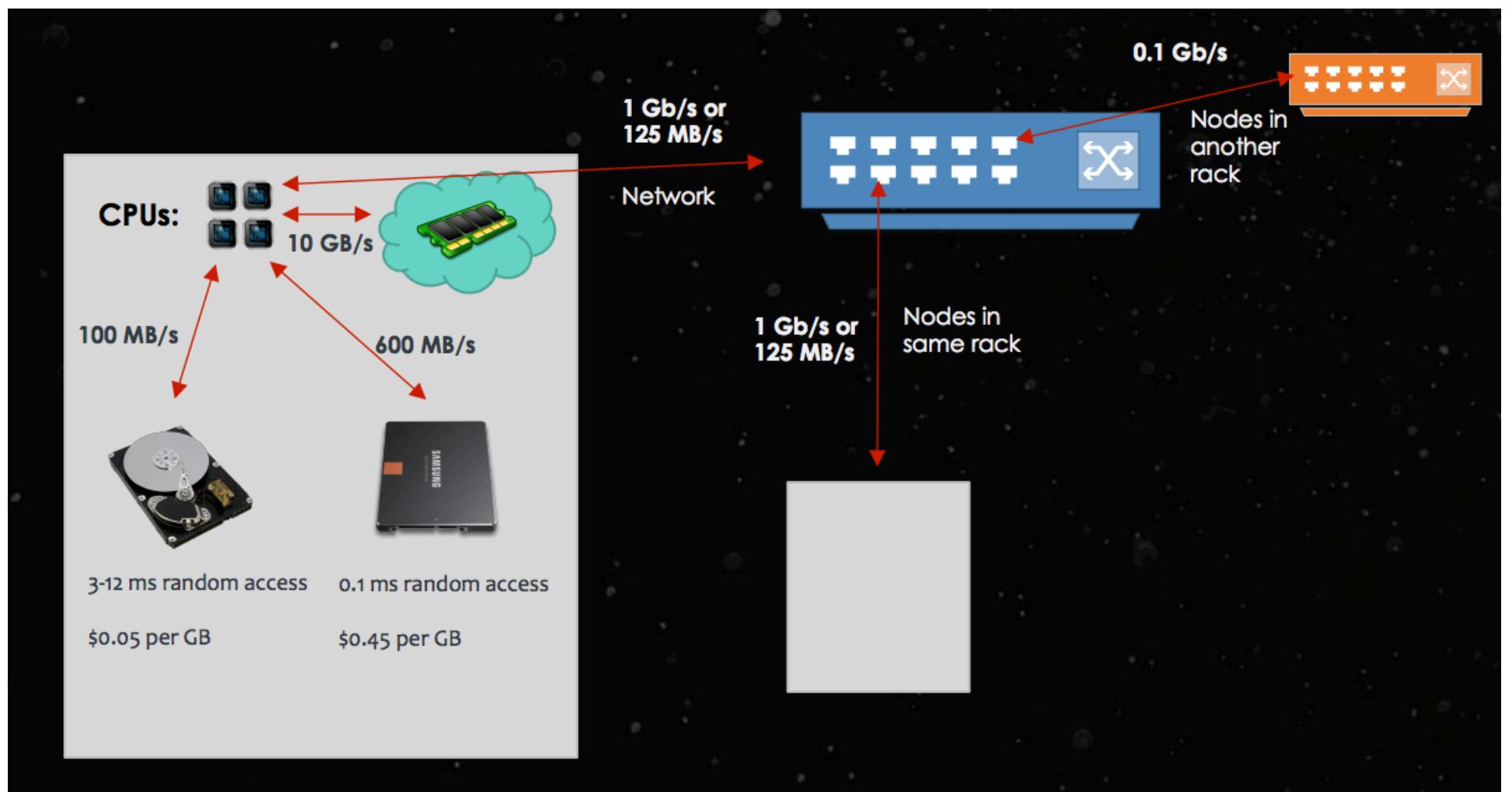


Why work at NativeX?

To help you reach your potential, we offer tools and resources, such as classes, conference attendance and tuition reimbursement. You'll also find mentors eager to help you pursue your dreams. With offices in San Francisco, California, Minneapolis, Minnesota and Sartell, Minnesota -- NativeX offers top notch mentorship and on-the-job experience.

Open positions (in San Francisco, CA or Minneapolis)

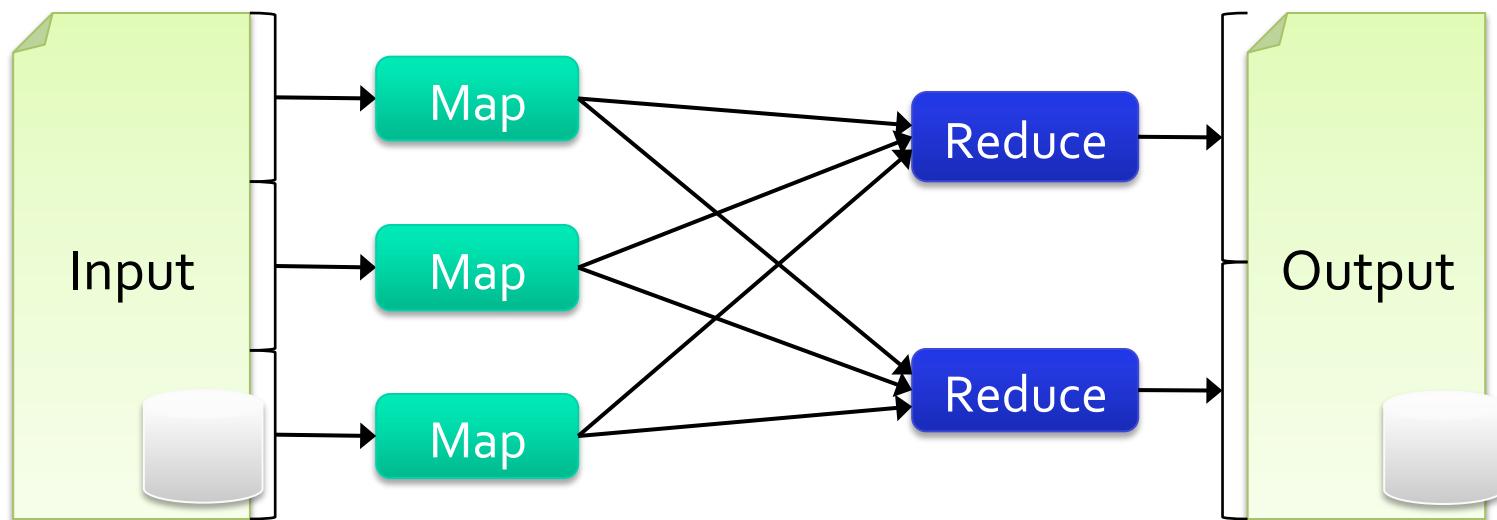
- **Junior Data Scientist**
- **Data Science Interns (Year 4+ of PhD)**
- **Java Engineers**



Motivation: Read from file; Process; Write to file

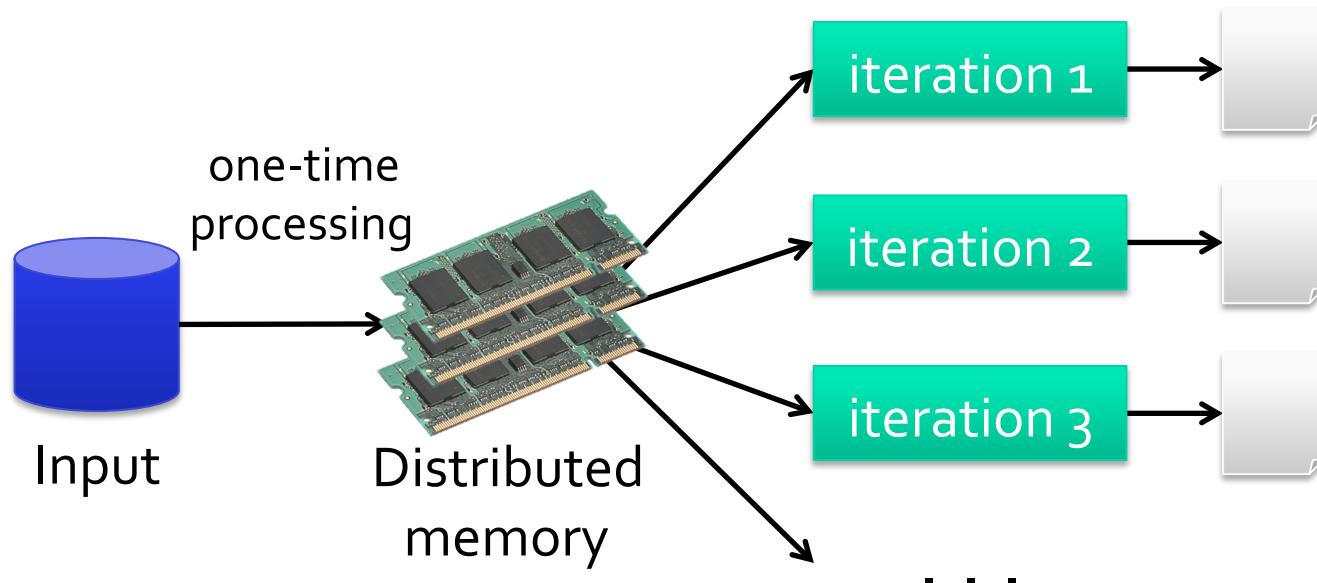
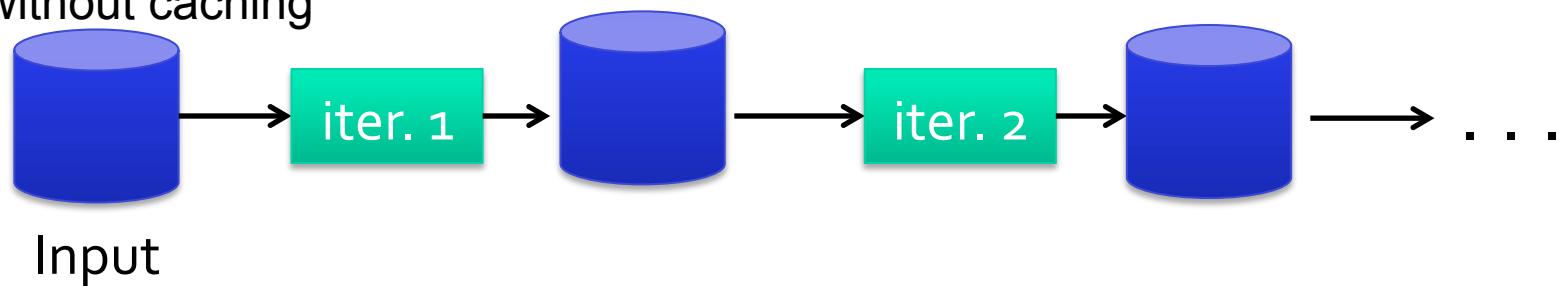
Current popular programming models for clusters transform data flowing from stable storage to stable storage (fault tolerant storage)

e.g., MapReduce:

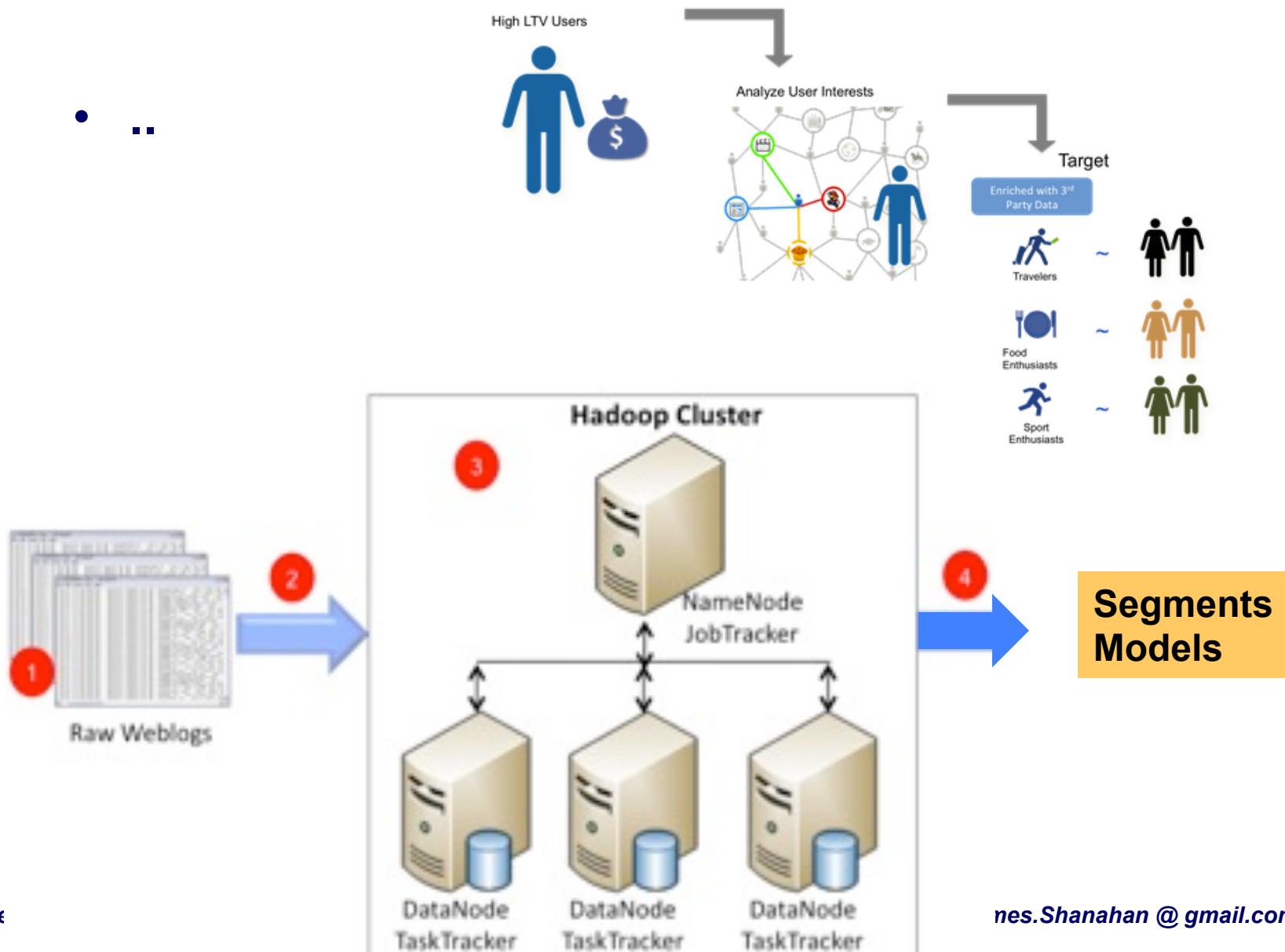


Goal: Keep Working Set in RAM

Hadoop MapReduce
Spark without caching



MapReduce: Typical usecase



Hadoop is a black box and lacks REPL; Complex Jobs not possible

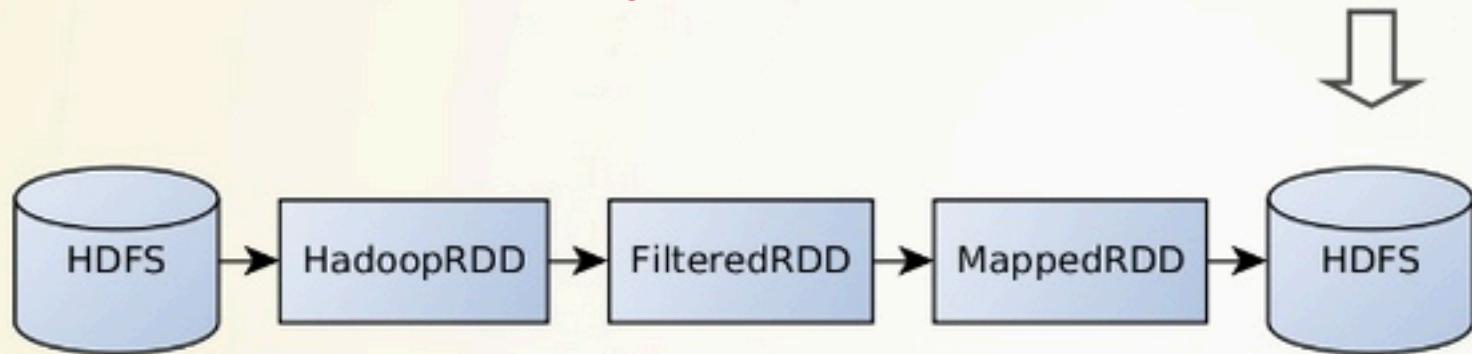
Read–eval–print loop

- Run the WordCount example job written in Java.
 - `hduser@ubuntu:/usr/local/hadoop$ bin/hadoop jar
hadoop-examples*.jar wordcount /user/hduser/
gutenberg /user/hduser/gutenberg-output`
- This command will read all the files in the HDFS directory `/user/hduser/gutenberg`, process it, and store the result in the HDFS directory `/user/hduser/gutenberg-output`.
- Hadoop is a black box
 - provide input, process, get output
 - Difficult to manipulate/access the data in the stream

Hadoop not good with job pipeline

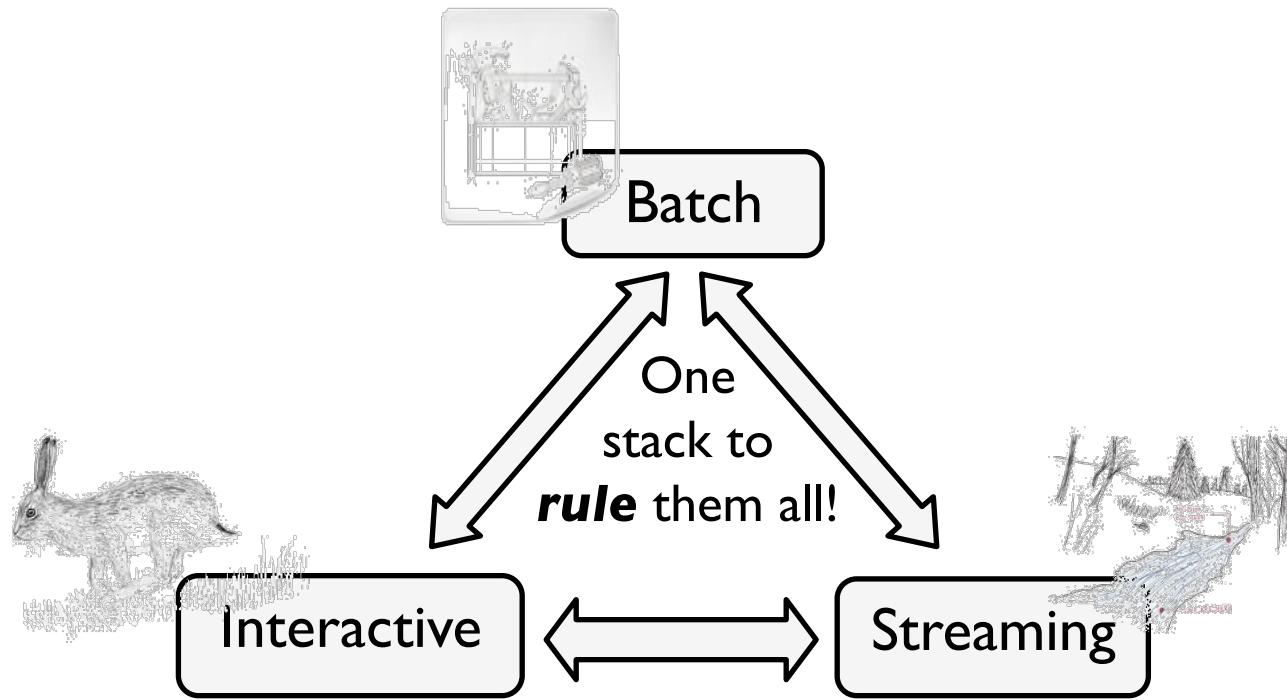
Step by Step

RDD: each row is a key, value pair



The RDD.saveAsTextFile() action triggers a job. Tasks are started on scheduled executors.

Goals



- **Easy** to combine *batch*, *streaming*, and *interactive* computations
- **Easy** to develop *sophisticated* algorithms
- **Compatible** with existing open source ecosystem (Hadoop/HDFS)

Spark

In-Memory Cluster Computing for Iterative and Interactive Applications

[Zaharia, 2009, UC Berkeley]



What is Spark?

Fast and Expressive Cluster Computing System
Compatible with Apache Hadoop

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

2-5x less code

Efficient

- General execution graphs
- In-memory storage

Usable

- Rich APIs in Java, Scala, Python, R (1.4)
- Interactive shell

Spark Metrics

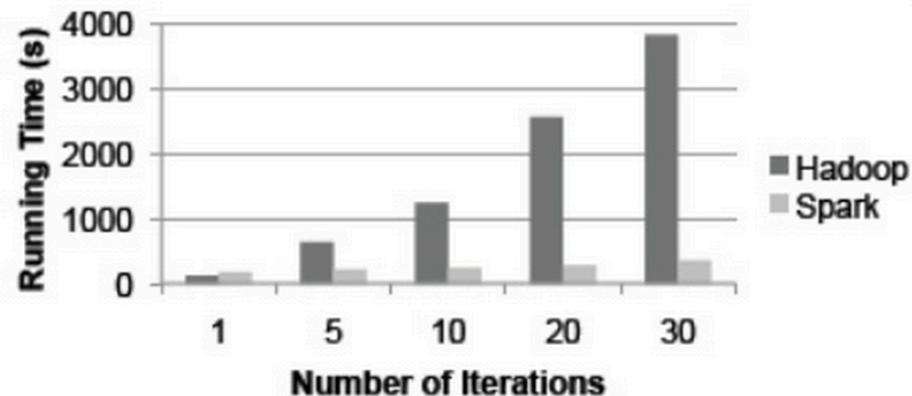
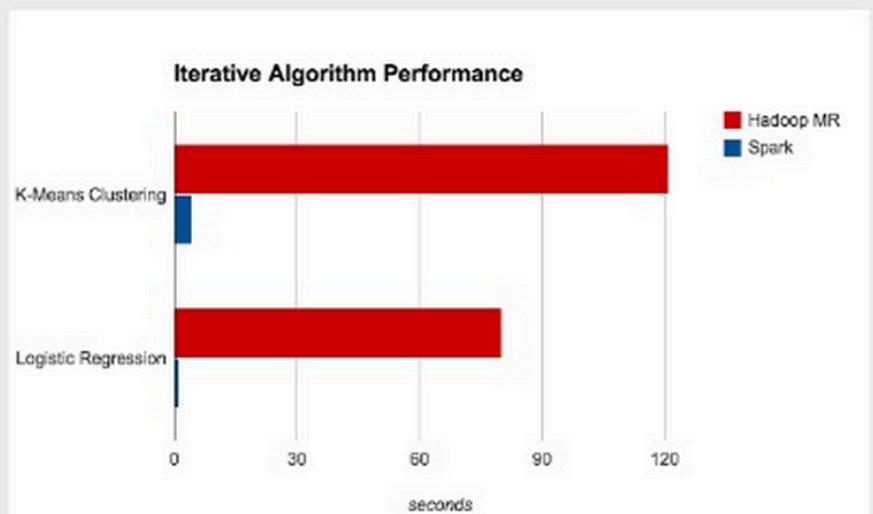


Figure 2: Logistic regression performance in Hadoop and Spark.



Code Size

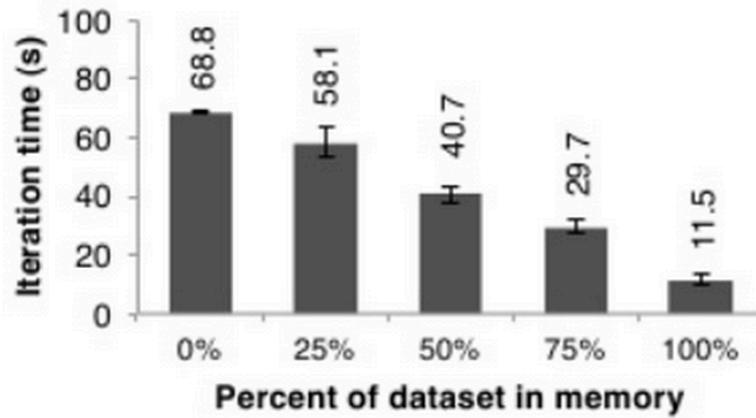
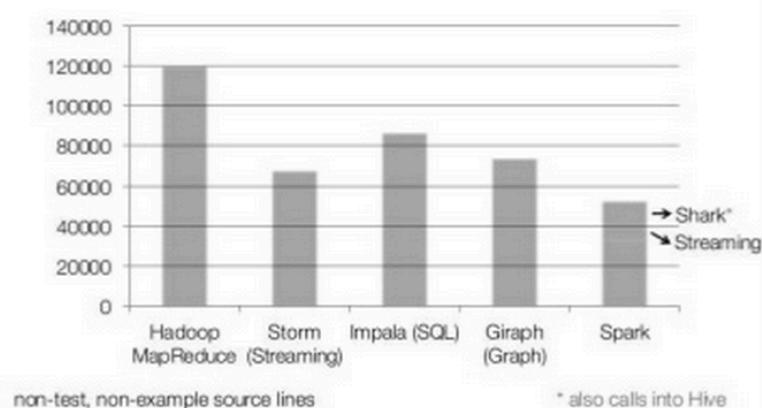
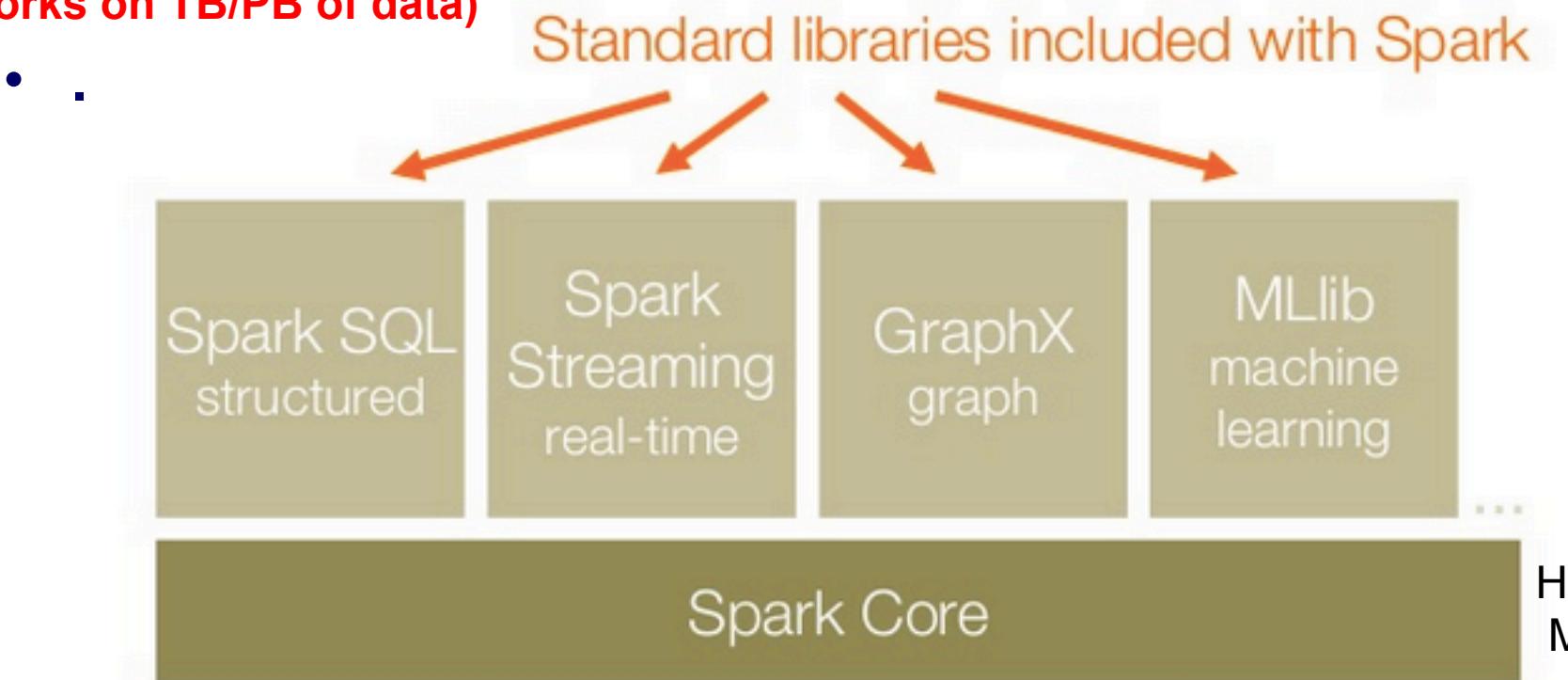


Figure 12: Performance of logistic regression using 100 GB data on 25 machines with varying amounts of data in memory.

- **Data Management**
 - Split data
 - Ship data
 - Replicate data
 - Run tasks
- **Task management**
 - Distribute, track
- **Fault tolerance (nodes crash 1-5 in a 1000 per day)**
- **First class programming framework for distributing programming over huge volumes of data that leverages memory, disk and network resources and constraints**
- **REPL (Read–eval–print loop)**

Spark

Apache Spark is an open-source cluster computing framework for big data (works on TB/PB of data)



Cluster Manager (Yarn/Mesos)

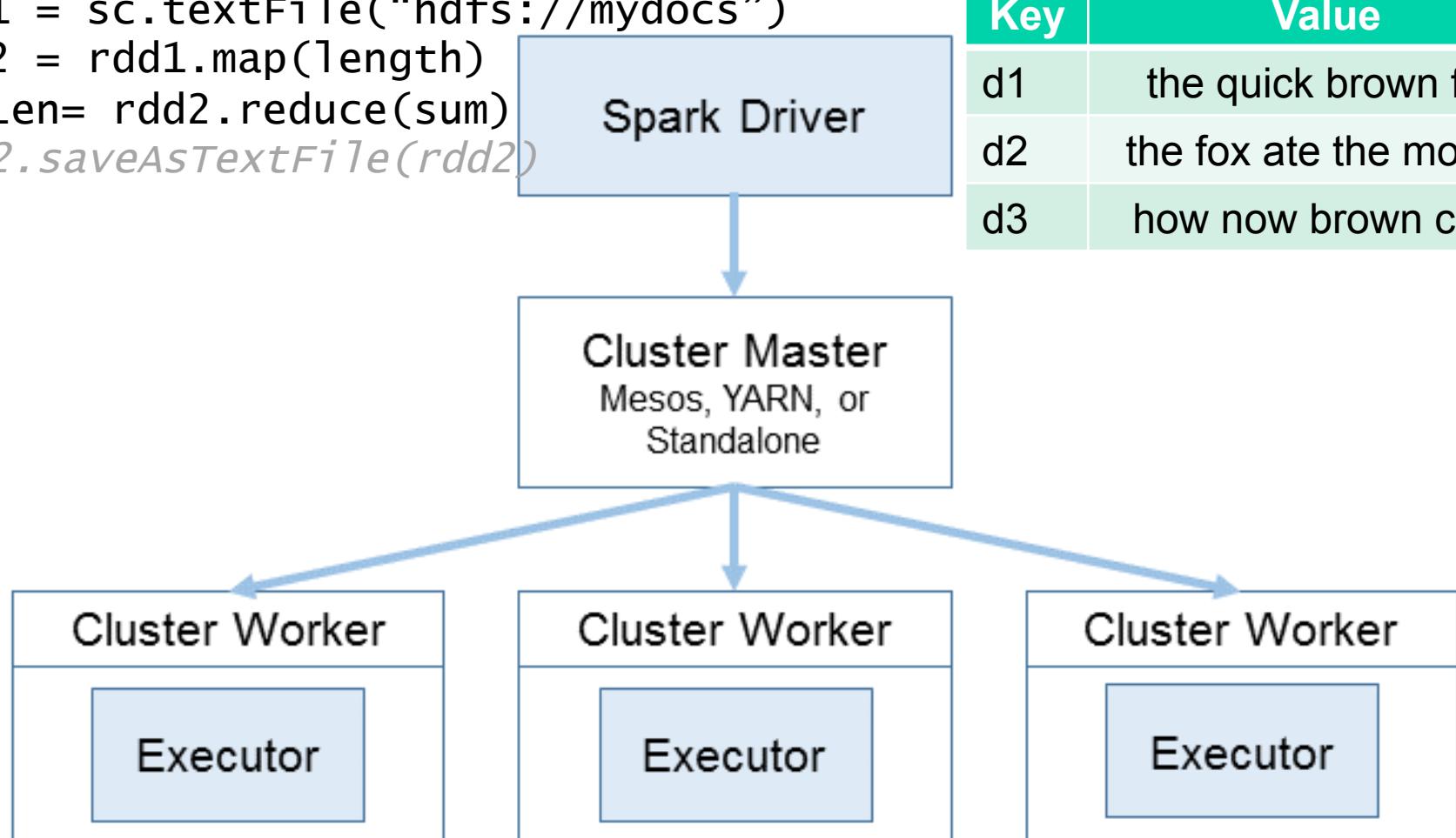
Machines (cores and storage)

Spark in local mode on a single computer or on a Spark Cluster

- Apache Spark is an open-source cluster computing framework for big data (works on TB/PB of data)

```
rdd1 = sc.textFile("hdfs://mydocs")
rdd2 = rdd1.map(length)
strLen= rdd2.reduce(sum)
rdd2.saveAsTextFile(rdd2)
```

Key	Value
d1	the quick brown fox
d2	the fox ate the mouse
d3	how now brown cow

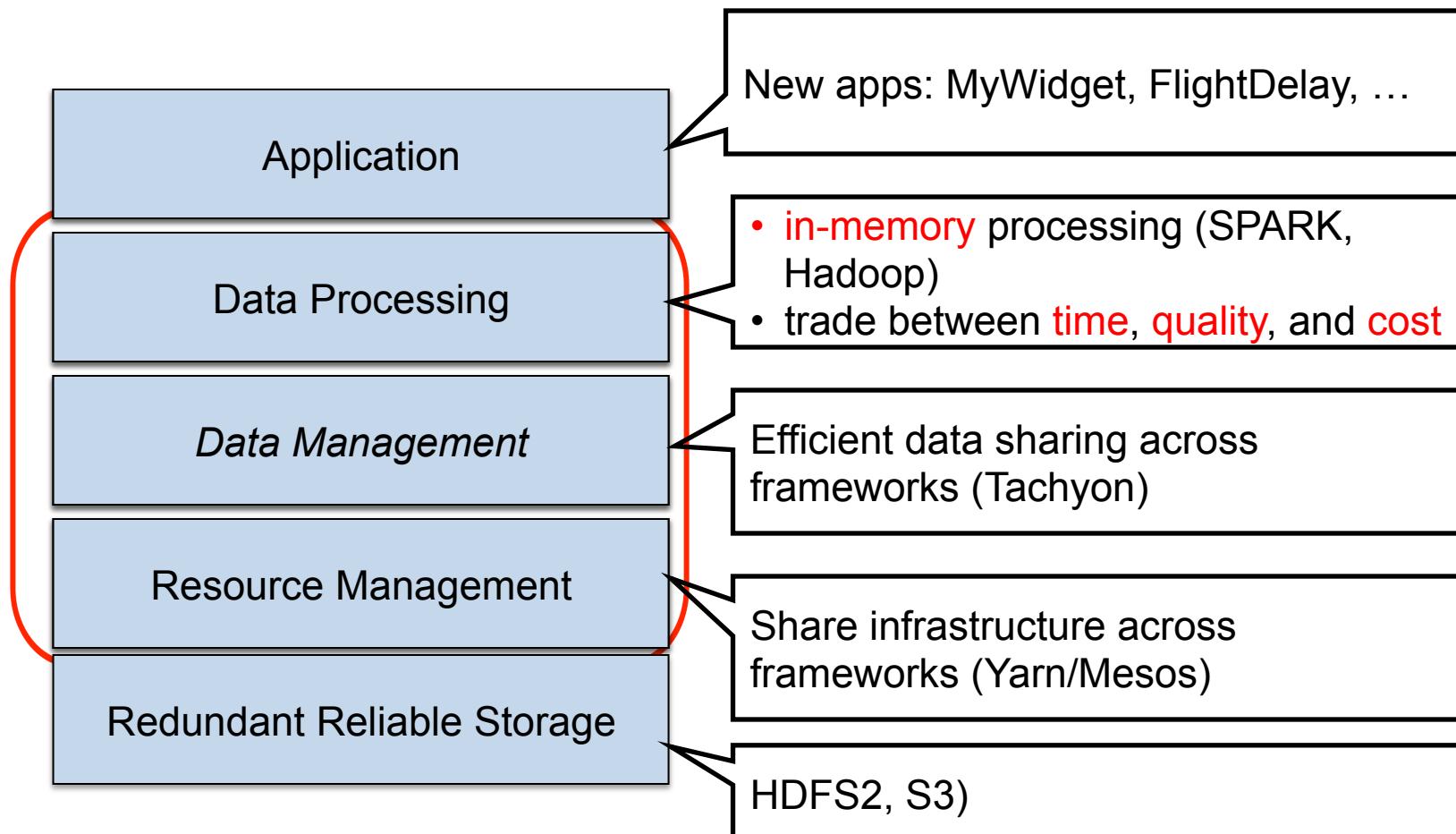


Spark APIs

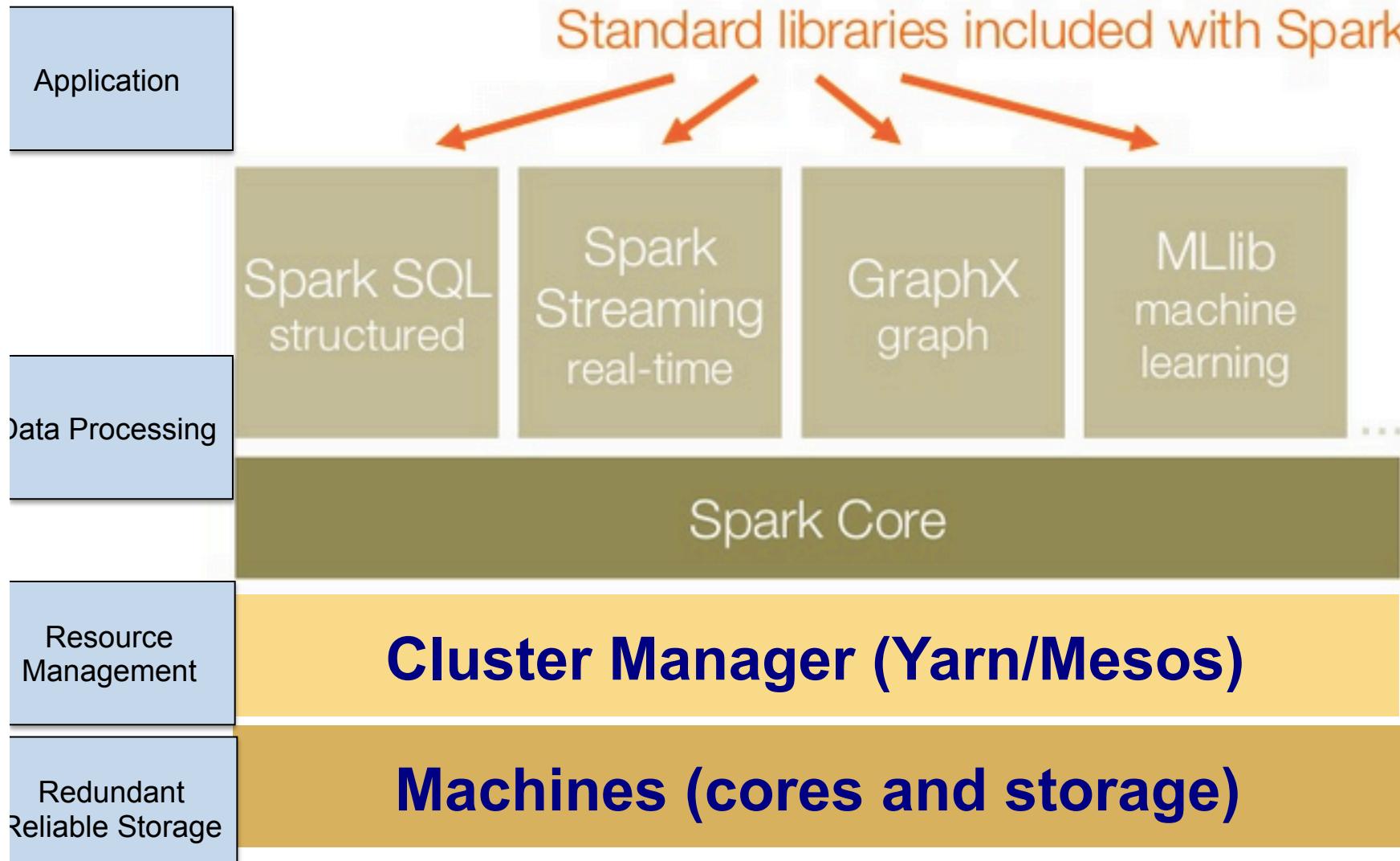
- **API**
 - In computing, a binding from a programming language to a library or operating system service is an application programming interface (API) providing glue code to use that library or service in a particular programming language.
- **Apache Spark** Apache Spark is an alternative big data computing system which can run on Yarn/Mesos and provides
 - An Elegant, Rich and Usable Core API
 - An Expansive set of ecosystem libraries built around the Core API
 - Hive compatibility via SparkSQL
 - Mature Python/R/Java/SQL/Scala support for both core APIs as well as the spark ecosystem
- **Spark has APIs for**
 - Scala, Java, Python, R, and SQL

(IPython Notebook as a Unified Data Science Interface to all of this)

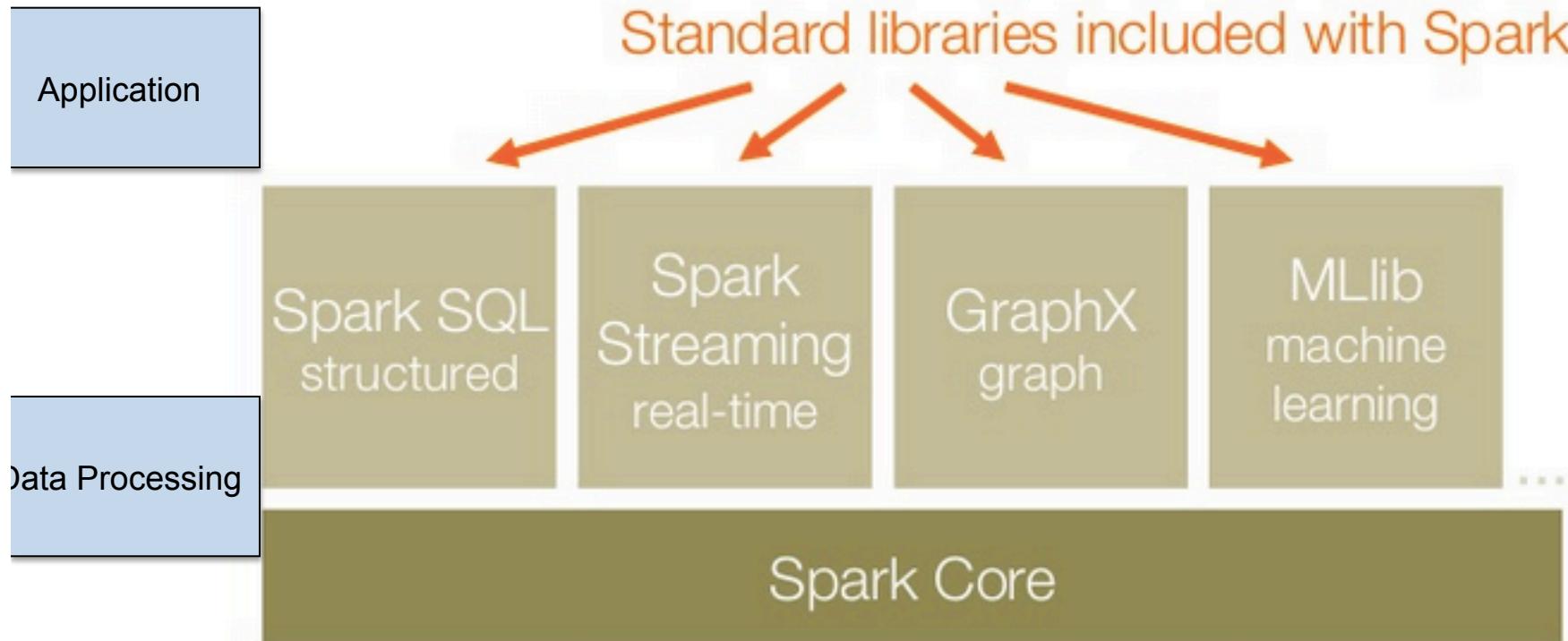
Berkeley Data Analytics Stack (BDAS)



Spark



Spark



Cluster Manager (Yarn/Mesos)

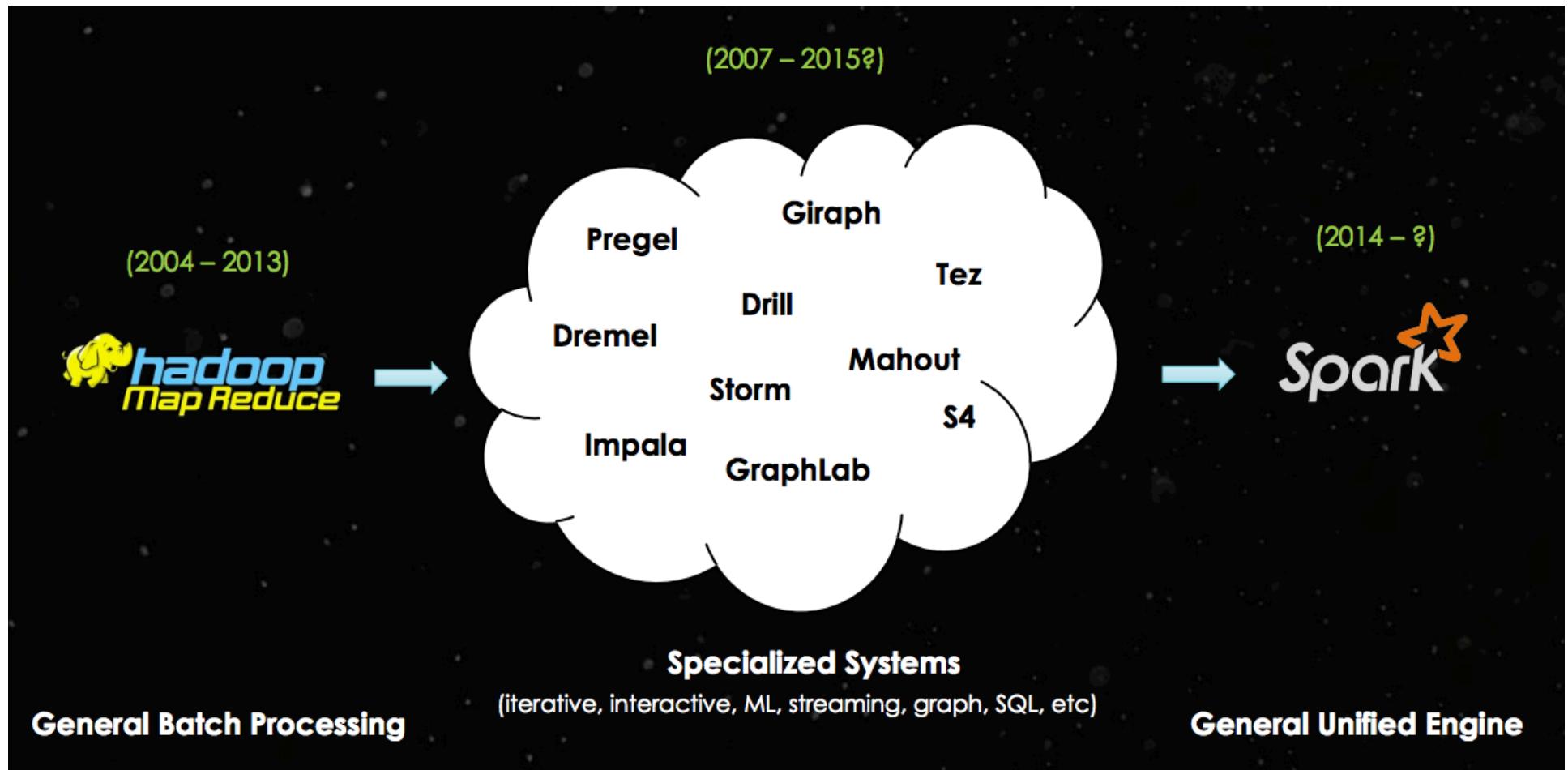
Spark does not deal with distributed storage machines (cores and storage)

Redundant
Reliable Storage

Divide and conquer with Closures

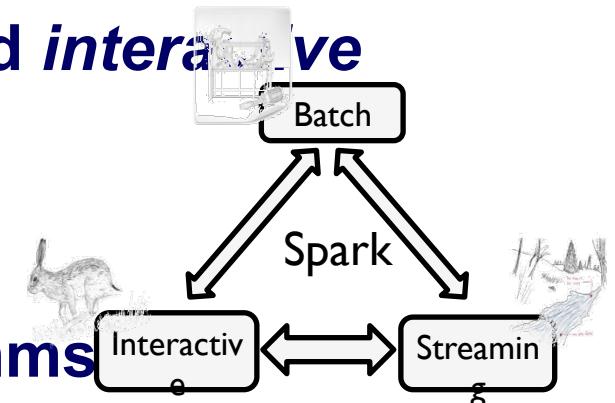
- **Decompose problems in non-overlapping subproblems**
- **Spark's API relies heavily on passing functions in the driver program to run on the cluster. There are three recommended ways to do this:**
 - Lambda expressions, for simple functions that can be written as an expression. (Lambdas do not support multi-statement functions or statements that do not return a value.)
 - Local defs inside the function calling into Spark, for longer code.
 - Top-level functions in a module.
- **Lazy evaluation! Optimize execution graphs**

Spark builds on ...



Summary

- **Support *interactive* and *streaming* computations**
 - In-memory, fault-tolerant storage abstraction, low-latency scheduling,...
- **Easy to combine *batch*, *streaming*, and *interactive* computations**
 - Spark execution engine supports all comp. models
- **Easy to develop *sophisticated* algorithms**
 - Scala interface, APIs for Java, Python, R, Hive QL, ...
 - New frameworks targeted to graph based and ML algorithms
- **Compatible with existing open source ecosystem**
- **Open source (Apache) and fully committed to release *high quality* software**



-
- End section

Part 1

- **Part 1: Introduction**

- Welcome Survey
- Install Spark
- Background and Motivation
 - Big Data Science
 - Functional Programming
 - Poorman's Map-Reduce (to dividing and conquering)

Install the following:

Java JDK

Spark

Anaconda Python (+Jupyter notebooks)

Tuesday, August 11th

- Tuesday, August 11th Registration desk: 8:00am - 6:00pm (Level 3 - Registration Desk)
- BESydney Networking Event (Invitation Only): 7:30am - 8:45am (Level 4 - Room 1)
- KDD15 Networking Lounge (Please drop in!): 10:20am - 6:00pm (Level 1 - Room 3 & 4)
- KDD15 Exhibit Hall: 8:00am - 6:00pm (Level 3 - Exhibition Area)

8:00am	Arrival Coffee 8:00am - 9:00am Level 3 - Exhibition Area						
8:45am	KEYNOTE 1: Ronny Kohavi, Microsoft Online Controlled Experiments: Lessons from Running A/B/n Tests for 12 Years 8:45am - 9:50am Grand Ballroom A&B						
9:50am	Morning Break 9:50am - 10:20am Level 2 Pre-Function Area; Level 3 Exhibit Hall; Level 4 Pre-Function Area						
10:20am	Level 1 - Room 5 & 6 S1 (Tutorial 10) Large Scale Distributed Data Science Using Apache Spark	Level 2 - State Room RT03 Topic Models and Tensors	Level 2 - Room 3 & 4 RT04 Interactivity and Learning	Level 3 - Ballroom A RT01 Social and Graphs 1	Level 3 - Ballroom B RT02 Mining Rich Data Types	Level 4 - Room 2 & 3 IG01 Big Data	Level 4 - Room 4 & 5 IG02 E-Commerce and IR
12:00pm	Lunch 12:00pm - 1:00pm Level 2 Pre-Function Area; Level 3 Exhibit Hall; Level 4 Pre-Function Area						
1:00pm	Level 1 - Room 5 & 6	Level 2 - State Room	Level 2 - Room 2	Level 2 - Room 3 & 4	Level 3 - Ballroom B	Level 4 - Room 2 & 3	Level 4 - Room 4 & 5

Tutorial Material

The screenshot shows a web browser window with the URL www.kdd.org/kdd2015/tutorial.html#s1 in the address bar. The browser's toolbar includes icons for back, forward, and refresh. Below the address bar is a menu bar with links to 'Apps', 'Bookmarks', 'Stanford Machine Le...', 'Getting Started', 'Statistical Analysis H...', and 'eBay/Google 2013: B...'. The main content area has a dark header with the text 'KDD2015' and navigation links for 'CALL FOR', 'ATTENDING', 'PROGRAM', 'WORKSHOPS', and 'TUTORIALS'. The 'TUTORIALS' link is underlined, indicating it is the current section. A red box highlights the 'Tutorial Material' link in the header.

S1 (Tutorial10)

Tutorial Material

Large Scale Distributed Data Science using Apache Spark



James G. Shanahan



Liang Dai

Large Scale Distributed Data Science using Apache Spark

Tutorial Webpage

A tutorial by James Shanahan and Liang Dai



Abstract

Apache Spark is an open-source cluster computing framework. It has emerged as the next generation big data processing engine, overtaking Hadoop MapReduce which helped ignite the big data revolution. Spark maintains MapReduce's linear scalability and fault tolerance, but extends it in a few important ways: it is much faster (100 times faster for certain applications), much easier to program in due to its rich APIs in Python, Java, Scala (and R), and its core data abstraction, the distributed data frame, and it goes far beyond batch applications to support a variety of compute-intensive tasks, including interactive queries, streaming, machine learning, and graph processing.

This tutorial will provide an accessible introduction to those not already familiar with Spark and its potential to revolutionize academic and commercial data science practices. It is divided into two parts: the first part will introduce fundamental Spark concepts, including Spark Core, data frames, the Spark Shell, Spark Streaming, Spark SQL, MLlib, and more; the second part will focus on hands-on algorithmic design and development with Spark (developing algorithms from scratch such as decision tree learning, graph processing algorithms such as pagerank/shortest path, gradient descent algorithms such as support vectors machines and matrix factorization). Industrial applications and deployments of

Larg

<http://kdd2015-sparktutorial.droppages.com/>

CONTACT US

emails:

james.shanahan@gmail.com

liangdai16@gmail.com

HOME

RELATED LINKS

KDD 2015
Apache Spark

Large Scale Distributed Data Science using Apache Spark

A tutorial by James Shanahan and Liang Dai



21st ACM SIGKDD Conference on Knowledge Discovery and Data Mining
10 - 13 August 2015, Hilton, Sydney

Home

Bibliography

Downloads

Preparation

Install JDK

Because Apache spark depends on Java run time environment, you need to have JDK installed in your machine. [Download](#)

Java SE Development Kit 8u51

You must accept the [Oracle Binary Code License Agreement for Java SE](#) to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux x86	146.9 MB	jdk-8u51-linux-i586.rpm
Linux x86	166.95 MB	jdk-8u51-linux-i586.tar.gz
Linux x64	145.19 MB	jdk-8u51-linux-x64.rpm
Linux x64	165.25 MB	jdk-8u51-linux-x64.tar.gz
Mac OS X x64	222.09 MB	jdk-8u51-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.36 MB	jdk-8u51-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	98.8 MB	jdk-8u51-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	139.79 MB	jdk-8u51-solaris-x64.tar.Z
Solaris x64	96.45 MB	jdk-8u51-solaris-x64.tar.gz
Windows x86	176.02 MB	jdk-8u51-windows-i586.exe
Windows x64	180.51 MB	jdk-8u51-windows-x64.exe

CONTACT US

emails:

james.shanahan_at_gmail.com

liangdai16_at_gmail.com

PREPARATION

RELATED LINKS

[KDD 2015](#)
[Apache Spark](#)

Make sure Java JDK is installed

- **Click here:**
<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
- **On windows machine set up \$JAVA_HOME**
 - Right-click the My Computer icon on your desktop and select Properties
 - Click the Advanced tab
 - Click the Environment Variables button
 - Under System Variables, click New
 - Enter the variable name as JAVA_HOME
 - Enter the variable value as the installation path for the Java Development Kit



- Java SE
- Java EE
- Java ME
- Java SE Support
- Java SE Advanced & Suite
- Java Embedded
- Java DB
- Web Tier
- Java Card
- Java TV
- New to Java
- Community
- Java Magazine

Overview

Downloads

Documentation

Community

Technologies

Training

Java SE Development Kit 8 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

See also:

- [Java Developer Newsletter](#) (tick the checkbox under Subscription Center > Oracle Technology News)
- [Java Developer Day hands-on workshops \(free\)](#) and other events
- [Java Magazine](#)

JDK 8u51 Checksum

Looking for JDK 8 on ARM?

JDK 8 for ARM downloads have moved to the [JDK 8 for ARM download page](#).

Java SE Development Kit 8u51

You must accept the [Oracle Binary Code License Agreement for Java SE](#) to download this software.

Accept License Agreement

Decline License Agreement

Product / File Description	File Size	Download
Linux x64	146.9 MB	jdk-8u51-linux-i586.rpm
Linux x64	166.95 MB	jdk-8u51-linux-i586.tar.gz
Mac OS X x64	145.19 MB	jdk-8u51-linux-x64.rpm
Mac OS X x64	165.25 MB	jdk-8u51-linux-x64.tar.gz
Mac OS X x64	222.09 MB	jdk-8u51-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.36 MB	jdk-8u51-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	98.8 MB	jdk-8u51-solaris-sparcv9.tar.gz

On Mac: double click to install

Anaconda Install

Download python+Notebook

Contents

- [Anaconda Install](#)
 - [OS X Install](#)
 - [OS X Uninstall](#)
 - [Linux Install](#)
 - [Linux Uninstall](#)
 - [Windows Install](#)
 - [Windows Uninstall](#)
 - [Updating from older Anaconda versions](#)
 - [What's next?](#)

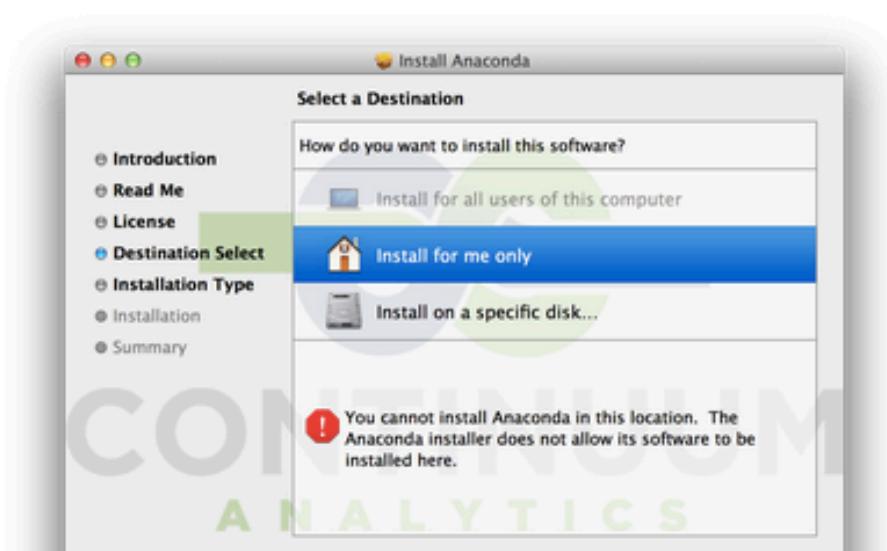
<http://docs.continuum.io/anaconda/install>

OS X Install

Download the Anaconda installer and double click it.

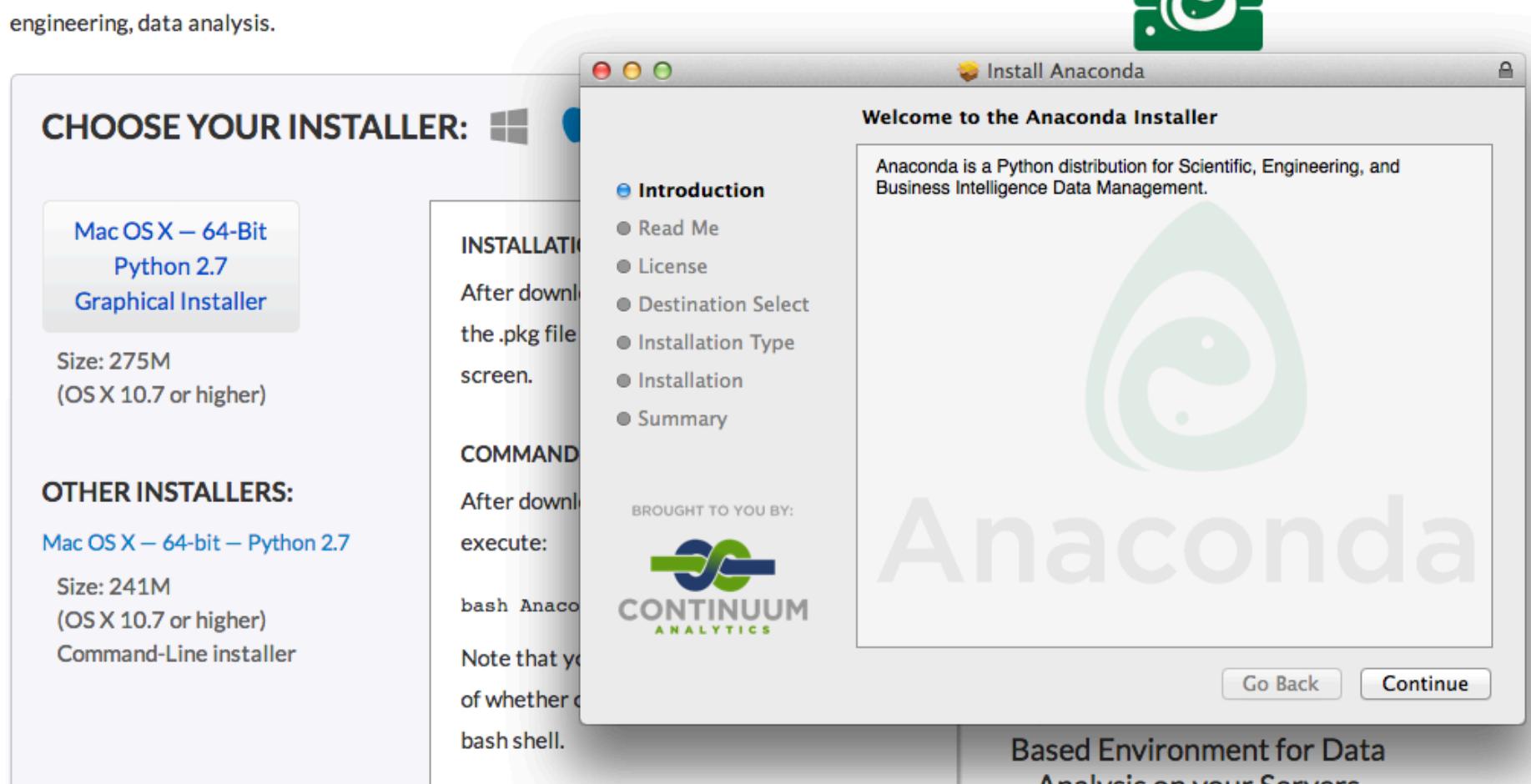
Download the [Anaconda installer](#) and double click it.

NOTE: You may see a screen that says "You cannot install Anaconda in this location. The Anaconda installer does not allow its software to be installed here." To fix this click the "Install for me only" button with the house icon and continue the installation.



Anaconda Installer

engineering, data analysis.



Apache Spark Download

- **Install 1.4.1 by clicking here**
 - <https://dl.dropboxusercontent.com/u/27377155/spark-1.4.1-bin-hadoop2.6.tgz>
- **For other versions click here**
 - <http://spark.apache.org/downloads.html>
 - <http://spark.apache.org/docs/latest/programming-guide.html>

spark.apache.org/downloads.html

Ops Twelve Month Calend Dashboards - All Doc Data Science Home AT&T Messages Senior Leadership Ho Q3 20

Spark *Lightning-fast cluster computing*

Download Libraries Documentation Examples Community FAQ

Get latest version Spark 1.4.1

Download Spark

The latest release of Spark is Spark 1.2.0, released on December 18, 2014 ([release notes](#)) ([git tag](#))

- Step1 1. Chose a Spark release: 1.2.0 (Dec 18 2014)
- Step2 2. Chose a package type: Pre-built for Hadoop 2.4 and later
- Step3 3. Chose a download type: Direct Download
- Step4 4. Download Spark: [spark-1.2.0-bin-hadoop2.4.tgz](#)

5. Verify this release using the 1.2.0 signatures and checksums.

Note: Scala 2.11 users should download the Spark source package and build [with Scala 2.11 support](#).

Untar Spark; start pyspark interpreter

```
tar -xzvf spark-1.4.1-bin-hadoop2.6.tgz  
cd spark-1.4.1-bin-hadoop2.6  
.bin/pyspark  
#if everything goes well, you can see the PySpark interactive shell  
.bin/pyspark
```

```
.....  
JAMES-SHANAHANs-Desktop-Pro:Software jshanahan$ ls spark-1.2.1-bin-hadoop2.4  
LICENSE README.md bin data examples python  
NOTICE RELEASE conf ec2 lib shin  
JAMES-SHANAHANs-Desktop-Pro:Software jshanahan$ spark-1.2.1-bin-hadoop2.4/bin/pyspark  
Python 2.7.3 (v2.7.3:70274d53c1dd, Apr  9 2012, 20:52:43)  
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin  
Type "help", "copyright", "credits" or "license" for more information.  
Spark assembly has been built with Hive, including Datanucleus jars on classpath  
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties  
15/02/11 16:57:02 INFO SecurityManager: Changing view acls to: jshanahan  
15/02/11 16:57:02 INFO SecurityManager: Changing modify acls to: jshanahan
```

Untar the Spark; start pyspark interpreter

- tar -xvf spark-1.2.1-bin-hadoop2.4.tgz
- Start pyspark
- spark-1.2.1-bin-hadoop2.4/bin/pyspark

```
JAMES-SHANAHANS-Desktop-Pro:Software jshanahan$ ls spark-1.2.1-bin-hadoop2.4
LICENSE      README.md    bin        data      examples   python
NOTICE      RELEASE     conf      ec2       lib       sbin
JAMES-SHANAHANS-Desktop-Pro:Software jshanahan$ spark-1.2.1-bin-hadoop2.4/bin/pyspark
Python 2.7.3 (v2.7.3:70274d53c1dd, Apr  9 2012, 20:52:43)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Spark assembly has been built with Hive, including Datanucleus jars on classpath
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
15/02/11 16:57:02 INFO SecurityManager: Changing view acls to: jshanahan
15/02/11 16:57:02 INFO SecurityManager: Changing modify acls to: jshanahan
15/02/11 16:57:02 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; user permissions: Set(jshanahan); users with modify permissions: Set(jshanahan)
15/02/11 16:57:02 INFO Slf4jLogger: Slf4jLogger started
15/02/11 16:57:02 INFO Remoting: Starting remoting
15/02/11 16:57:02 INFO Remoting: Remoting started; listening on addresses :[akka.tcp://sparkDriver@10.0.0.93:54079]
15/02/11 16:57:02 INFO Utils: Successfully started service 'sparkDriver' on port 54079.
15/02/11 16:57:02 INFO SparkEnv: Registering MapOutputTracker
15/02/11 16:57:02 INFO SparkEnv: Registering BlockManagerMaster
15/02/11 16:57:02 INFO DiskBlockManager: Created local directory at /var/folders/j4/95k348x940xcz40fk/T/spark-5081f827-8087-436d-8cc2-bb05a24410a3/spark-c4be330b-173d-4f3f-8659-2a919c2f7bab
15/02/11 16:57:02 INFO MemoryStore: MemoryStore started with capacity 273.0 MB
2015-02-11 16:57:03.065 java[44783:ea03] Unable to load realm info from SCDynamicStore
15/02/11 16:57:03 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using avro classes where applicable
15/02/11 16:57:03 INFO HttpFileServer: HTTP File server directory is /var/folders/j4/95k348x940xcz40fn/T/spark-d6caeef39-c3bc-48d3-8a9e-f4b24e3e4a21/spark-1c7165c4-a18d-4f23-86ae-5d92519ff345
15/02/11 16:57:03 INFO HttpServer: Starting HTTP Server
15/02/11 16:57:03 INFO Utils: Successfully started service 'HTTP file server' on port 54080.
15/02/11 16:57:03 INFO Utils: Successfully started service 'SparkUI' on port 4040.
15/02/11 16:57:03 INFO SparkUI: Started SparkUI at http://10.0.0.93:4040
15/02/11 16:57:03 INFO Executor: Starting executor ID <driver> on host localhost
15/02/11 16:57:03 INFO AkkaUtils: Connecting to HeartbeatReceiver: akka.tcp://sparkDriver@10.0.0.93:54079
15/02/11 16:57:03 INFO NettyBlockTransferService: Server created on 54081
15/02/11 16:57:03 INFO BlockManagerMaster: Trying to register BlockManager
15/02/11 16:57:03 INFO BlockManagerMasterActor: Registering block manager localhost:54081 with 273.0 MB
15/02/11 16:57:03 INFO BlockManagerMaster: Registered BlockManager
Welcome to
   _/\_   _/\_   _/\_   _/\_
  / \ \ / \ \ / \ \ / \ \
 /   \ /   \ /   \ /   \ /   \
 /     \ /     \ /     \ /     \
 /       \ /       \ /       \ /       \
 /         \ /         \ /         \ /         \
 /           \ /           \ /           \ /           \
 /             \ /             \ /             \ /             \
 /               \ /               \ /               \ /               \
 /                 \ /                 \ /                 \ /                 \
 /                   \ /                   \ /                   \ /                   \
 /                     \ /                     \ /                     \ /                     \
 /                       \ /                       \ /                       \ /                       \
 /                         \ /                         \ /                         \ /                         \
 /                           \ /                           \ /                           \ /                           \
 /                             \ /                             \ /                             \ /                             \
 /                               \ /                               \ /                               \ /                               \
 /                                 \ /                                 \ /                                 \ /                                 \
 /                                   \ /                                   \ /                                   \ /                                   \
 /                                     \ /                                     \ /                                     \ /                                     \
 /                                       \ /                                       \ /                                       \ /                                       \
 /                                         \ /                                         \ /                                         \ /                                         \
 /                                           \ /                                           \ /                                           \ /                                           \
 /                                             \ /                                             \ /                                             \ /                                             \
 /                                               \ /                                               \ /                                               \ /                                               \
 /                                                 \ /                                                 \ /                                                 \ /                                                 \
 /                                                   \ /                                                   \ /                                                   \ /                                                   \
 /                                                     \ /                                                     \ /                                                     \ /                                                     \
 /                                                       \ /                                                       \ /                                                       \ /                                                       \
 /                                                         \ /                                                         \ /                                                         \ /                                                         \
 /                                                           \ /                                                           \ /                                                           \ /                                                           \
 /                                                             \ /                                                             \ /                                                             \ /                                                             \
 /                                                               \ /                                                               \ /                                                               \ /                                                               \
 /                                                                 \ /                                                                 \ /                                                                 \ /                                                                 \ /
```

Using Python version 2.7.3 (v2.7.3:70274d53c1dd, Apr 9 2012 20:52:43)
SparkContext available as sc.
>>> █

```
.  
+-- bin  
|   +-- beeline  
|   |   beeline.cmd  
|   |   compute-classpath.cmd  
|   |   compute-classpath.sh  
|   |   load-spark-env.sh  
|   +-- pyspark      bin./pyspark #python  
|   |   pyspark2.cmd  
|   |   pyspark.cmd  
|   +-- run-example  
|   |   run-example2.cmd  
|   |   run-example.cmd  
|   +-- spark-class  
|   |   spark-class2.cmd  
|   |   spark-class.cmd  
|   +-- spark-shell    bin./spark-shell #SCALA  
|   |   spark-shell2.cmd  
|   |   spark-shell.cmd  
|   +-- spark-sql  
|   +-- spark-submit  
|   |   spark-submit2.cmd  
|   |   spark-submit.cmd  
|   +-- utils.sh  
|   |   windows-utils.cmd  
+-- CHANGES.txt  
+-- conf  
|   +-- fairscheduler.xml.template  
|   +-- log4j.properties.template  
|   +-- metrics.properties.template  
|   +-- slaves.template  
|   +-- spark-defaults.conf.template  
|   +-- spark-env.sh.template  
+-- data  
|   +-- mllib
```

```
+-- ec2  
|   +-- deploy.generic  
|   |   README  
|   |   spark-ec2  
|   |   spark_ec2.py  
+-- examples  
|   +-- src  
+-- lib  
|   +-- datanucleus-api-jdo-3.2.6.jar  
|   +-- datanucleus-core-3.2.10.jar  
|   +-- datanucleus-rdbms-3.2.9.jar  
|   +-- spark-1.3.1-yarn-shuffle.jar  
|   +-- spark-assembly-1.3.1-hadoop2.6.0.jar  
|   +-- spark-examples-1.3.1-hadoop2.6.0.jar  
+-- LICENSE  
+-- NOTICE  
+-- python  
|   +-- build  
|   +-- docs  
|   +-- lib  
|   +-- pyspark  
|   +-- run-tests  
|   +-- test_support  
+-- README.md  
+-- RELEASE  
+-- sbin  
|   +-- slaves.sh  
|   +-- spark-config.sh  
|   +-- spark-daemon.sh  
|   +-- spark-daemons.sh  
|   +-- start-all.sh  
|   +-- start-history-server.sh  
|   +-- start-master.sh  
|   +-- start-slave.sh  
|   +-- start-slaves.sh  
|   +-- start-thriftserver.sh  
|   +-- stop-all.sh  
|   +-- stop-history-server.sh  
|   +-- stop-master.sh  
|   +-- stop-slaves.sh  
|   +-- stop-thriftserver.sh
```

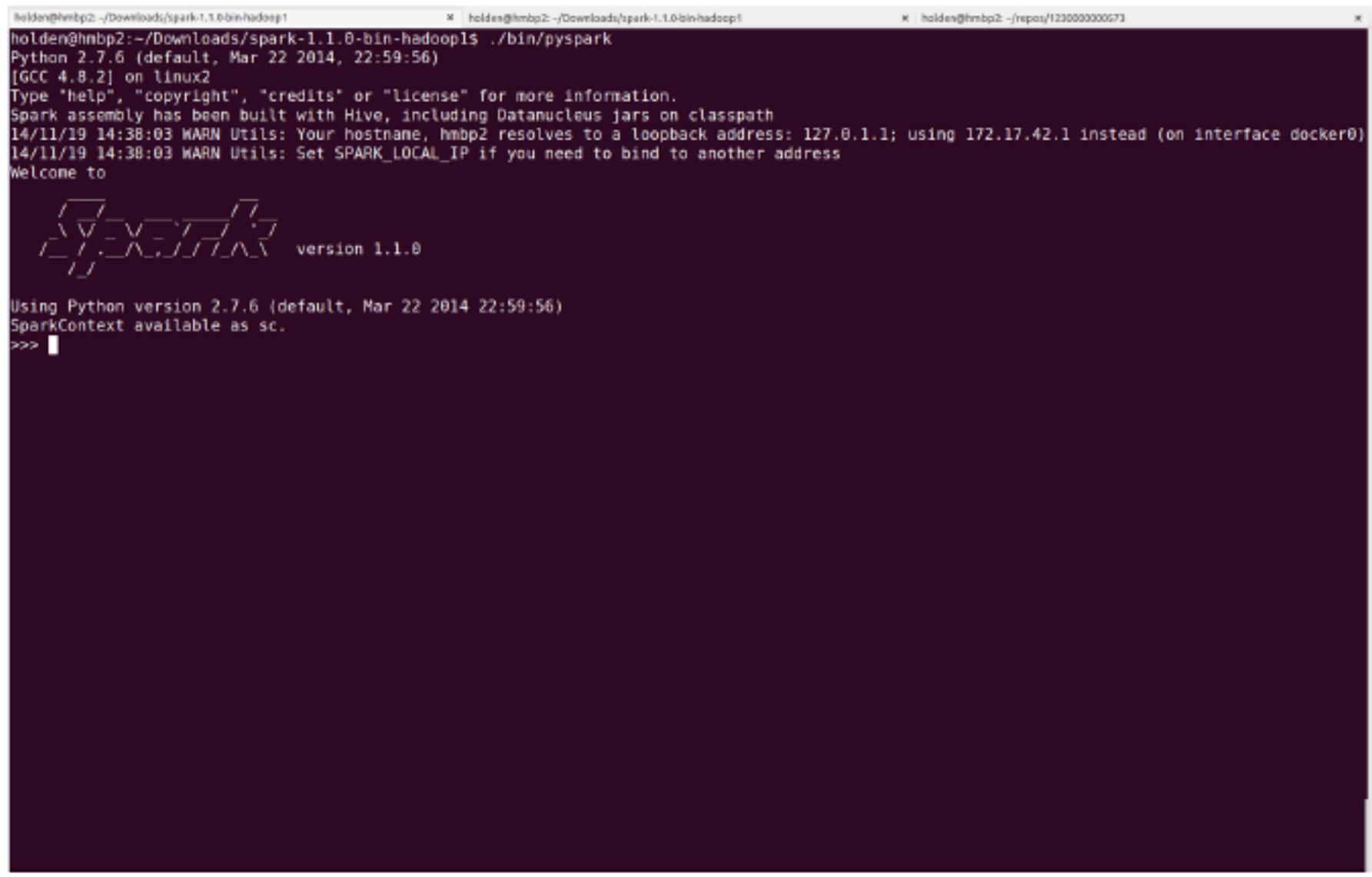
A screenshot of a terminal window showing the PySpark shell. The terminal has three tabs open: 'holden@hmbp2:~/Downloads/spark-1.1.0-bin-hadoop1' (active), 'holden@hmbp2:~/Downloads/spark-1.1.0-bin-hadoop1\$./bin/pyspark', and 'holden@hmbp2:~/repos/1330000000573'. The active tab shows the PySpark shell starting up, displaying Python version information, spark assembly details, and a warning about the hostname. It then displays the Spark logo and the message 'Welcome to'. The shell prompt 'In []' is visible at the bottom.

Figure 2-2. The PySpark shell with less logging output

Bin/sparkR

```
JAMES-SANAHANS-Desktop-Pro:KDD-Notebooks jshanahan$ cd /Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/spark-1.4.0-bin-hadoop2.6
JAMES-SANAHANS-Desktop-Pro:spark-1.4.0-bin-hadoop2.6 jshanahan$ bin/sparkR

R version 3.1.3 (2015-03-09) -- "Smooth Sidewalk"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

Launching java with spark-submit command /Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/spark-1.4.0-bin-hadoop2.6/bin/spark-submit "sparkr-shell" /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T//RtmpjQpRRC/backend_port14725284a4ad
log4j:WARN No appenders could be found for logger (io.netty.util.internal.logging.InternalLoggerFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
15/08/10 13:52:17 INFO SparkContext: Running Spark version 1.4.0
15/08/10 13:52:19 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
15/08/10 13:52:19 INFO SecurityManager: Changing view acls to: jshanahan
15/08/10 13:52:19 INFO SecurityManager: Changing modify acls to: jshanahan
15/08/10 13:52:19 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(jshanahan); users with modify permissions: Set(jshanahan)
```

Iris data check

- `df = createDataFrame(sqlContext, iris)`
- `head(df)`

```
> df
DataFrame[Sepal_Length:double, Sepal_Width:double, Petal_Length:double, Petal_Width:double, Species:string]
> head(df)
15/08/10 13:52:51 INFO SparkContext: Starting job: dfToCols at NativeMethodAccessorImpl.java:-2
15/08/10 13:52:51 INFO DAGScheduler: Got job 1 (dfToCols at NativeMethodAccessorImpl.java:-2) with 1 output partitions (allowLoc al=false)
15/08/10 13:52:51 INFO DAGScheduler: Final stage: ResultStage 1(dfToCols at NativeMethodAccessorImpl.java:-2)
15/08/10 13:52:51 INFO DAGScheduler: Parents of final stage: List()
15/08/10 13:52:51 INFO DAGScheduler: Missing parents: List()
15/08/10 13:52:51 INFO DAGScheduler: Submitting ResultStage 1 (MapPartitionsRDD[4] at dfToCols at NativeMethodAccessorImpl.java: -2), which has no missing parents
15/08/10 13:52:51 INFO MemoryStore: ensureFreeSpace(8776) called with curMem=2134, maxMem=278019440
15/08/10 13:52:51 INFO MemoryStore: Block broadcast_1 stored as values in memory (estimated size 8.6 KB, free 265.1 MB)
15/08/10 13:52:51 INFO MemoryStore: ensureFreeSpace(3621) called with curMem=10910, maxMem=278019440
15/08/10 13:52:51 INFO MemoryStore: Block broadcast_1_piece0 stored as bytes in memory (estimated size 3.5 KB, free 265.1 MB)
15/08/10 13:52:51 INFO BlockManagerInfo: Added broadcast_1_piece0 in memory on localhost:54179 (size: 3.5 KB, free: 265.1 MB)
15/08/10 13:52:51 INFO SparkContext: Created broadcast 1 from broadcast at DAGScheduler.scala:874
15/08/10 13:52:51 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 1 (MapPartitionsRDD[4] at dfToCols at NativeMet hodAccessorImpl.java:-2)
15/08/10 13:52:51 INFO TaskSchedulerImpl: Adding task set 1.0 with 1 tasks
15/08/10 13:52:51 INFO TaskSetManager: Starting task 0.0 in stage 1.0 (TID 1, localhost, PROCESS_LOCAL, 15837 bytes)
15/08/10 13:52:51 INFO Executor: Running task 0.0 in stage 1.0 (TID 1)
15/08/10 13:52:52 INFO Executor: Finished task 0.0 in stage 1.0 (TID 1). 2502 bytes result sent to driver
15/08/10 13:52:52 INFO TaskSetManager: Finished task 0.0 in stage 1.0 (TID 1) in 637 ms on localhost (1/1)
15/08/10 13:52:52 INFO DAGScheduler: ResultStage 1 (dfToCols at NativeMethodAccessorImpl.java:-2) finished in 0.637 s
15/08/10 13:52:52 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
15/08/10 13:52:52 INFO DAGScheduler: Job 1 finished: dfToCols at NativeMethodAccessorImpl.java:-2, took 0.654923 s
  Sepal_Length Sepal_Width Petal_Length Petal_Width Species
1          5.1       3.5       1.4       0.2   setosa
2          4.9       3.0       1.4       0.2   setosa
3          4.7       3.2       1.3       0.2   setosa
4          4.6       3.1       1.5       0.2   setosa
```

Large Scale Distributed Data Science using Spark

A tutorial by James Shanahan and Liang Dai

-



Slides

[Download]

Examples

- Word Count
- Dataframe
- PageRank
- Linear regression via gradient descent
- Pipeline logistic regression

Download

Download sample notebooks for tutorial
`tar -xzf KDD-Notebook.tgz`

To import a notebook, drag the file onto the listing below or [click here](#).

New Notebook

File	Delete
LogisticRegression.ipynb	
SPARK-Lecture1.ipynb	
SparkSQL.ipynb	
SummaryStatisticsExample.ipynb	
WordCount.ipynb	

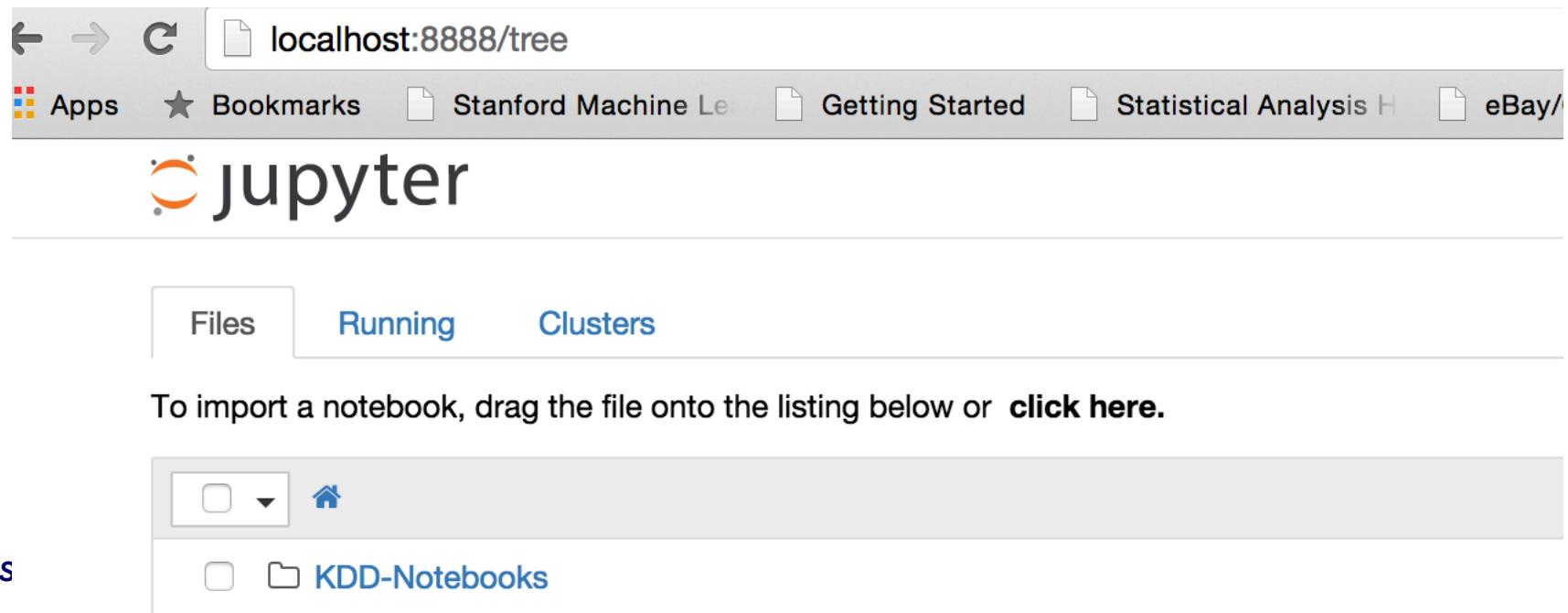
```
#start the python server and notebook in your browser
# from the shell/terminal command line
$ ipython notebook &
```

```
JAMES-SHANAHANS-Desktop-Pro:~ jshanahan$ ipython notebook
2015-02-11 17:51:52.879 [NotebookApp] Using existing profile directory '/Users/jshanahan/.ipython/profile_default'
2015-02-11 17:51:52.879 [NotebookApp] Using MathJax from CDN: https://cdn.mathjax.org/mathjax/latest/MathJax.js
2015-02-11 17:51:52.897 [NotebookApp] Serving notebooks from local directory: /Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/Notebooks
2015-02-11 17:51:52.897 [NotebookApp] 0 active kernels
2015-02-11 17:51:52.898 [NotebookApp] The IPython Notebook is running at: http://localhost:8888/
2015-02-11 17:51:52.898 [NotebookApp] Use Control-C to stop this server and shut
```

Large S

Fire up ipython Notebook in your Browser

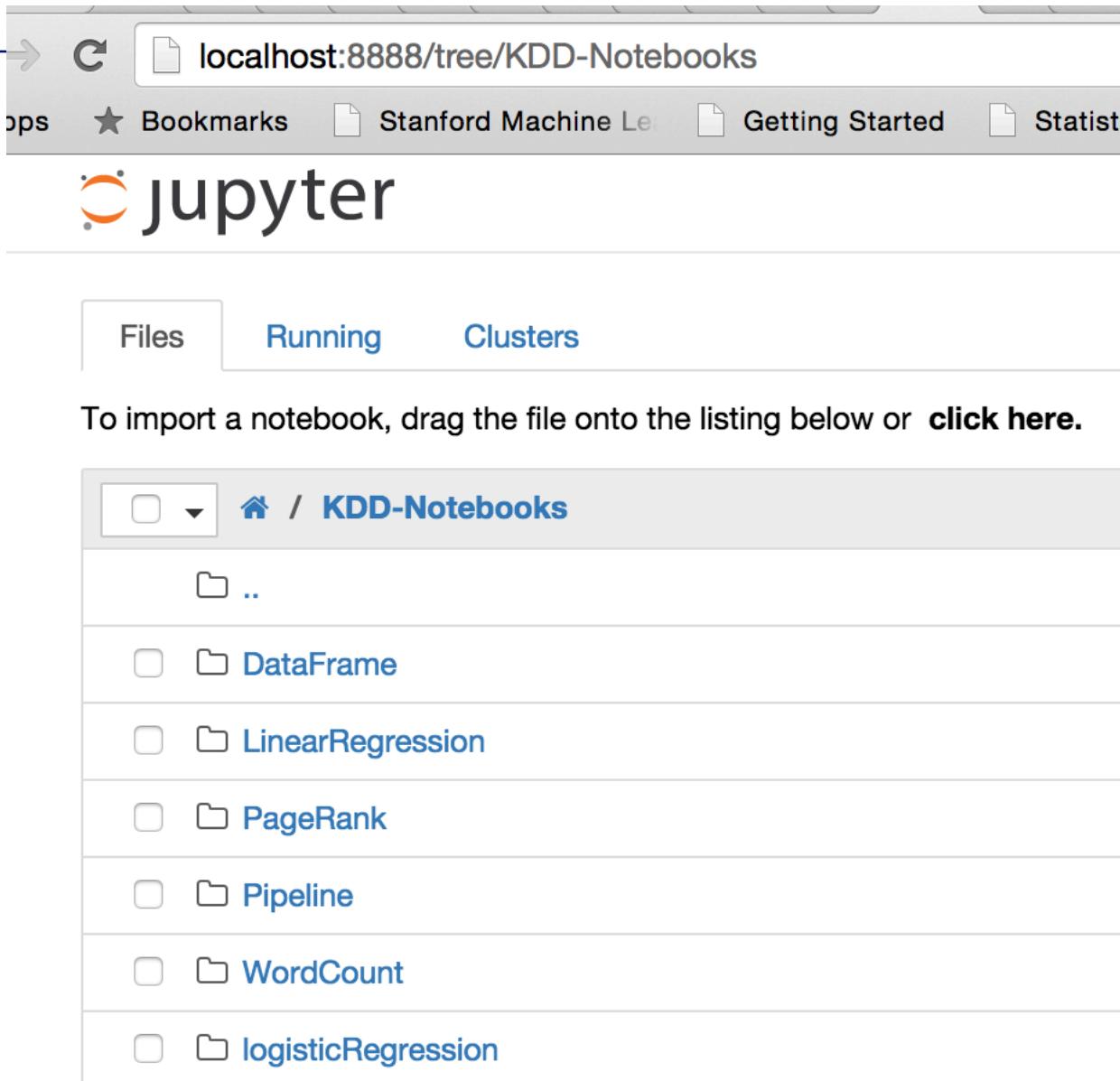
```
#start the python server and notebook in your browser  
# from the shell/terminal command line  
#Change directory to you working directory  
untar KDD-notebooks  
cd KDD-notebooks  
ipython notebook &
```



Jimi's notebooks

- **cd ~/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/Spark-Lectures-Shared/KDD2015-private/**

Notebooks for Tutorial

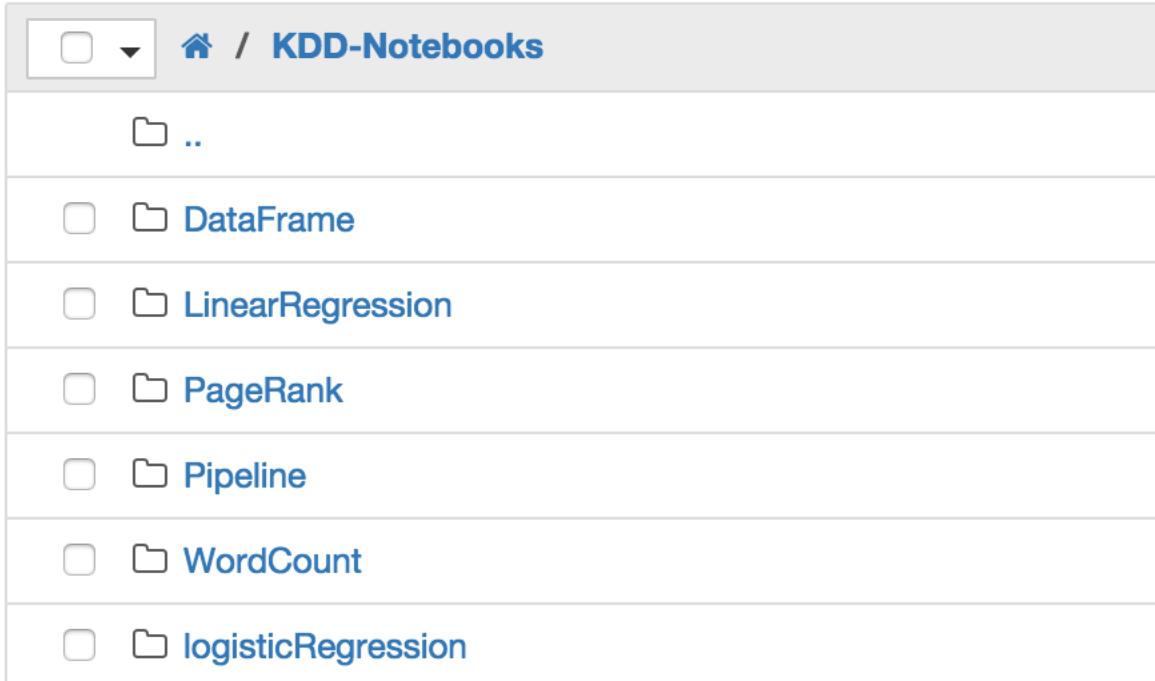


The screenshot shows a web browser window with the URL `localhost:8888/tree/KDD-Notebooks` in the address bar. The browser's toolbar includes icons for back, forward, and search, along with links for Bookmarks, Stanford Machine Learning, Getting Started, and Statistics. Below the toolbar, the Jupyter logo is displayed. A navigation bar with tabs for Files, Running, and Clusters is visible. A message instructs users to import a notebook by dragging it onto the listing or clicking a link. The main content area shows a file tree under the path `/KDD-Notebooks`, listing several notebooks: `..`, `DataFrame`, `LinearRegression`, `PageRank`, `Pipeline`, `WordCount`, and `logisticRegression`.

-  jupyter

Files Running Clusters

To import a notebook, drag the file onto the listing below or [click here](#).


/ KDD-Notebooks

- ..
- DataFrame
- LinearRegression
- PageRank
- Pipeline
- WordCount
- logisticRegression

Modify path for SPARK_HOME

C localhost:8888/notebooks/KDD-Notebooks/WordCount/WordCount.ipynb
★ Bookmarks Stanford Machine Le Getting Started Statistical Analysis H eBay/Google 2013: E Inquiries InferPatents WindAlert - Coyote P SamCam

jupyter WordCount (autosaved) Python 2

In [1]:

```
import os
import sys
spark_home = os.environ['SPARK_HOME'] = '/Users/liang/Downloads/spark-1.4.1-bin-hadoop2.6/'
spark_home = os.environ['SPARK_HOME'] = '/Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/spark-1.4.0-bin-ha
if not spark_home:
    raise ValueError('SPARK_HOME environment variable is not set')
sys.path.insert(0,os.path.join(spark_home,'python'))
sys.path.insert(0,os.path.join(spark_home,'python/lib/py4j-0.8.2.1-src.zip'))
execfile(os.path.join(spark_home,'python/pyspark/shell.py'))
```

Welcome to

version 1.4.0

Using Python version 2.7.9 (default, Dec 15 2014 10:37:34)
SparkContext available as sc, HiveContext available as sqlContext.

To execute the cell
Press
Shift + Return

In [2]:

```
%%writefile wordcount.txt
hello hi hi hallo
bonjour hola hi ciao
nihao konnichiwa ola
hola nihao hello
```

Writing wordcount.txt

KDD-Notebooks — bash — 120x23

```
15/08/09 21:22:33 INFO SparkEnv: Registering MapOutputTracker
15/08/09 21:22:33 INFO SparkEnv: Registering BlockManagerMaster
15/08/09 21:22:33 INFO DiskBlockManager: Created local directory at /private/var/folders/j4/95k348x940xcz40fkdmgy_n40000
gn/T/spark-ac6b33c1-7252-4e53-a335-f67957821ed7/blockmgr-d16baf4-b7e9-4c7e-a33d-15a6d8198406
15/08/09 21:22:33 INFO MemoryStore: MemoryStore started with capacity 265.1 MB
15/08/09 21:22:33 INFO HttpFileServer: HTTP File server directory is /private/var/folders/j4/95k348x940xcz40fkdmgy_n40000
gn/T/spark-ac6b33c1-7252-4e53-a335-f67957821ed7/httpd-a18d50aa-ea36-4339-9c15-604330e30548
15/08/09 21:22:33 INFO HttpServer: Starting HTTP Server
15/08/09 21:22:33 INFO Utils: Successfully started service 'HTTP file server' on port 52389.
15/08/09 21:22:33 INFO SparkEnv: Registering OutputCommitCoordinator
15/08/09 21:22:34 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
15/08/09 21:22:34 WARN Utils: Service 'SparkUI' could not bind on port 4041. Attempting port 4042.
15/08/09 21:22:34 INFO Utils: Successfully started service 'SparkUI' on port 4042.
15/08/09 21:22:34 INFO SparkUI: Started SparkUI at http://192.168.1.51:4042
15/08/09 21:22:34 INFO Executor: Starting executor ID driver on host localhost
15/08/09 21:22:34 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 52390.
15/08/09 21:22:34 INFO NettyBlockTransferService: Server created on 52390
15/08/09 21:22:34 INFO BlockManagerMaster: Trying to register BlockManager
15/08/09 21:22:34 INFO BlockManagerMasterEndpoint: Registering block manager localhost:52390 with 265.1 MB RAM, BlockManagerId(driver, localhost, 52390)
15/08/09 21:22:34 INFO BlockManagerMaster: Registered BlockManager
```

localhost:8891/notebooks/WordCount/WordCount.ipynb

jupyter WordCount Last Checkpoint: 2 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Help

Cell Toolbar: None

In [1]:

```
import os
import sys
spark_home = os.environ['SPARK_HOME'] = '/Users/liang/Downloads/spark-1.4.1-bin-hadoop2.6/'
spark_home = os.environ['SPARK_HOME'] = '/Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/spark-1.4.0-bin-hadoop2.6'
if not spark_home:
    raise ValueError('SPARK_HOME environment variable is not set')
sys.path.insert(0,os.path.join(spark_home,'python'))
sys.path.insert(0,os.path.join(spark_home,'python/lib/py4j-0.8.2.1-src.zip'))
execfile(os.path.join(spark_home,'python/pyspark/shell.py'))
```

Welcome to

version 1.4.0

Using Python version 2.7.9 (default, Dec 15 2014 10:37:34)
SparkContext available as sc, HiveContext available as sqlContext.

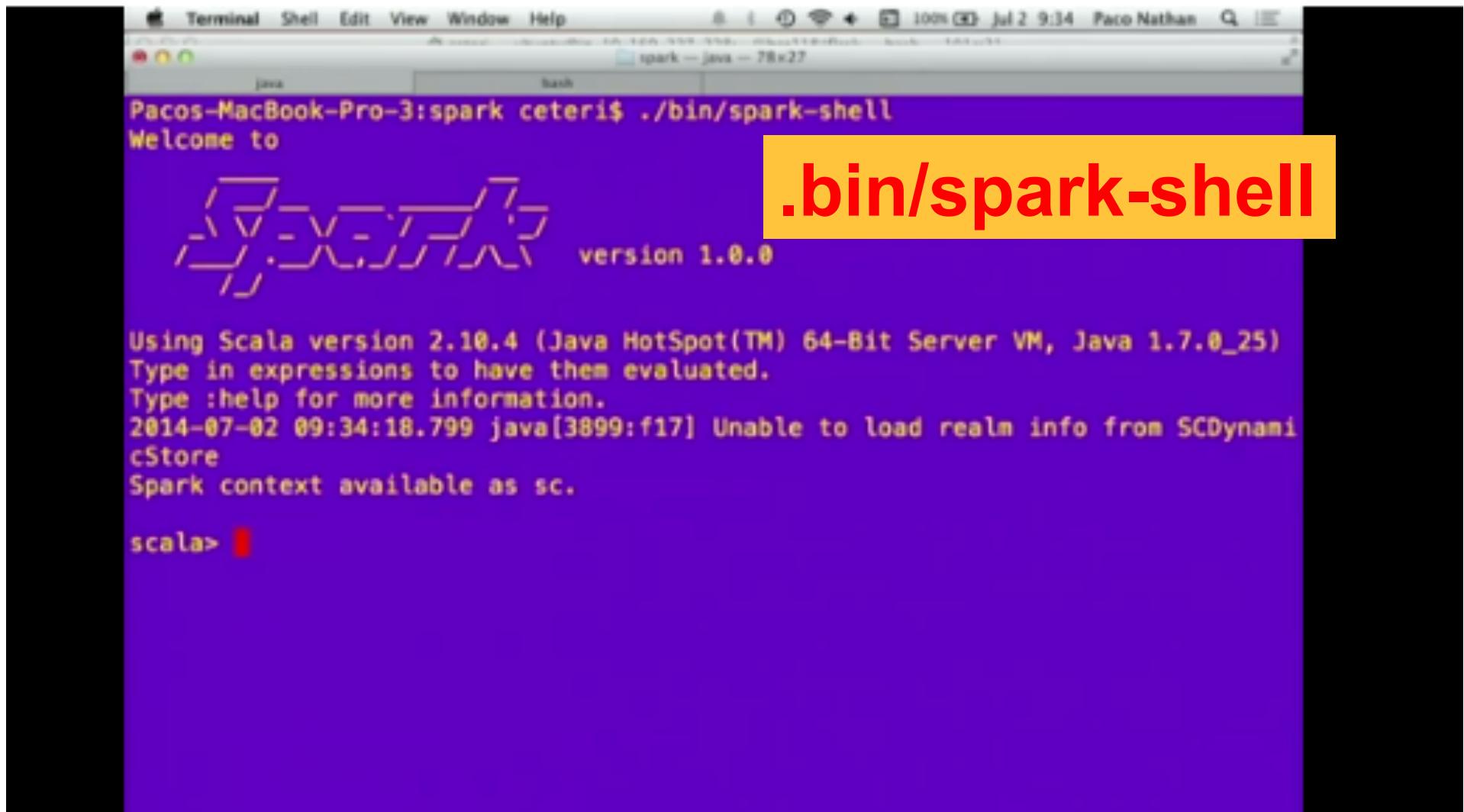
```

KDD-Notebooks — bash — 120x23
15/08/09 21:22:33 INFO SparkEnv: Registering MapOutputTracker
15/08/09 21:22:33 INFO SparkEnv: Registering BlockManagerMaster
15/08/09 21:22:33 INFO DiskBlockManager: Created local directory at /private/var/folders/j4/95k348x940xcz40fkdmgy_n40000
gn/T/spark-ac6b33c1-7252-4e53-a335-f67957821ed7/blockmgr-d16baf4-b7e9-4c7e-a33d-15a6d8198406
15/08/09 21:22:33 INFO MemoryStore: MemoryStore started with capacity 265.1 MB
15/08/09 21:22:33 INFO HttpFileServer: HTTP File server directory is /private/var/folders/j4/95k348x940xcz40fkdmgy_n40000
gn/T/spark-ac6b33c1-7252-4e53-a335-f67957821ed7/httpd-a18d50aa-ea36-4339-9c15-604330e30548
15/08/09 21:22:33 INFO HttpServer: Starting HTTP Server
15/08/09 21:22:33 INFO Utils: Successfully started service 'HTTP file server' on port 52389.
15/08/09 21:22:33 INFO SparkEnv: Registering OutputCommitCoordinator
15/08/09 21:22:34 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
15/08/09 21:22:34 WARN Utils: Service 'SparkUI' could not bind on port 4041. Attempting port 4042.
15/08/09 21:22:34 INFO Utils: Successfully started service 'SparkUI' on port 4042.
15/08/09 21:22:34 INFO SparkUI: Started SparkUI at http://192.168.1.51:4042
15/08/09 21:22:34 INFO Executor: Starting executor ID driver on host localhost
15/08/09 21:22:34 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 52390.
15/08/09 21:22:34 INFO NettyBlockTransferService: Server created on 52390
15/08/09 21:22:34 INFO BlockManagerMaster: Trying to register BlockManager
15/08/09 21:22:34 INFO BlockManagerMasterEndpoint: Registering block manager localhost:52390 with 265.1 MB RAM, BlockManagerId(driver, localhost, 52390)
15/08/09 21:22:34 INFO BlockManagerMaster: Registered block manager localhost:52390 with 265.1 MB RAM
```

The screenshot shows a Mac OS X desktop environment with several open windows:

- Terminal Window:** Titled "KDD-Notebooks — bash — 120x23", displaying log output from a Spark environment.
- Keynote Presentation:** A slide titled "closure" is displayed, showing a slide show interface.
- Safari Browser:** The address bar shows "192.168.1.51:4042/jobs/". The page content is the PySparkShell application UI.
- PySparkShell Application UI:** Titled "PySparkShell application UI", it displays the "Spark Jobs" section. Key information shown includes:
 - Total Uptime: 25 s
 - Scheduling Mode: FIFO
 - Event Timeline: A collapsed section.
 - Enable zooming: A checkbox.
 - Executors: A table with columns for Executors, Added (blue square), and Removed (red square). All cells are empty.
 - Jobs: A table with columns for Jobs, Succeeded (blue square), Failed (red square), and Running (green square). All cells are empty.
 - Date Range: A timeline from Thu 6 August 2015 to Wed 12 August 2015.

Spark shell in Scala



```
Terminal Shell Edit View Window Help
spark -- java - 78x27
java
Pacos-MacBook-Pro-3:spark ceteri$ ./bin/spark-shell
Welcome to
  ____          _ _   _ _ _ 
 / \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ 
version 1.0.0

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_25)
Type in expressions to have them evaluated.
Type :help for more information.
2014-07-02 09:34:18.799 java[3899:f17] Unable to load realm info from SCDynamicStore
Spark context available as sc.

scala> 
```

TAB lists the available methods and attributes

The screenshot shows a Jupyter Notebook interface with a Python 2 kernel. In the top cell (In [1]), the following code is visible:

```
sys.path.insert(0, os.path.join(sp
sys.path.insert(0, os.path.join(sp
execfile(os.path.join(spark_home,

```

In the bottom cell (In [33]), the user has typed "sc." followed by the TAB key. A tooltip labeled "Autocomplete" appears, listing the available methods and attributes for the sc variable:

- sc.environment
- sc.getLocalProperty
- sc.hadoopFile
- sc.hadoopRDD
- sc.master
- sc.newAPIHadoopFile
- sc.newAPIHadoopRDD
- sc.parallelize
- sc.pickleFile
- sc.pythonExec

The word "pythonExec" is highlighted in the list.

Below the autocomplete list, the rest of the code in In [33] is visible:

```
#set up Spark and give us a spark
spark_home = os.environ['SPARK_HOME']
```

Part 1

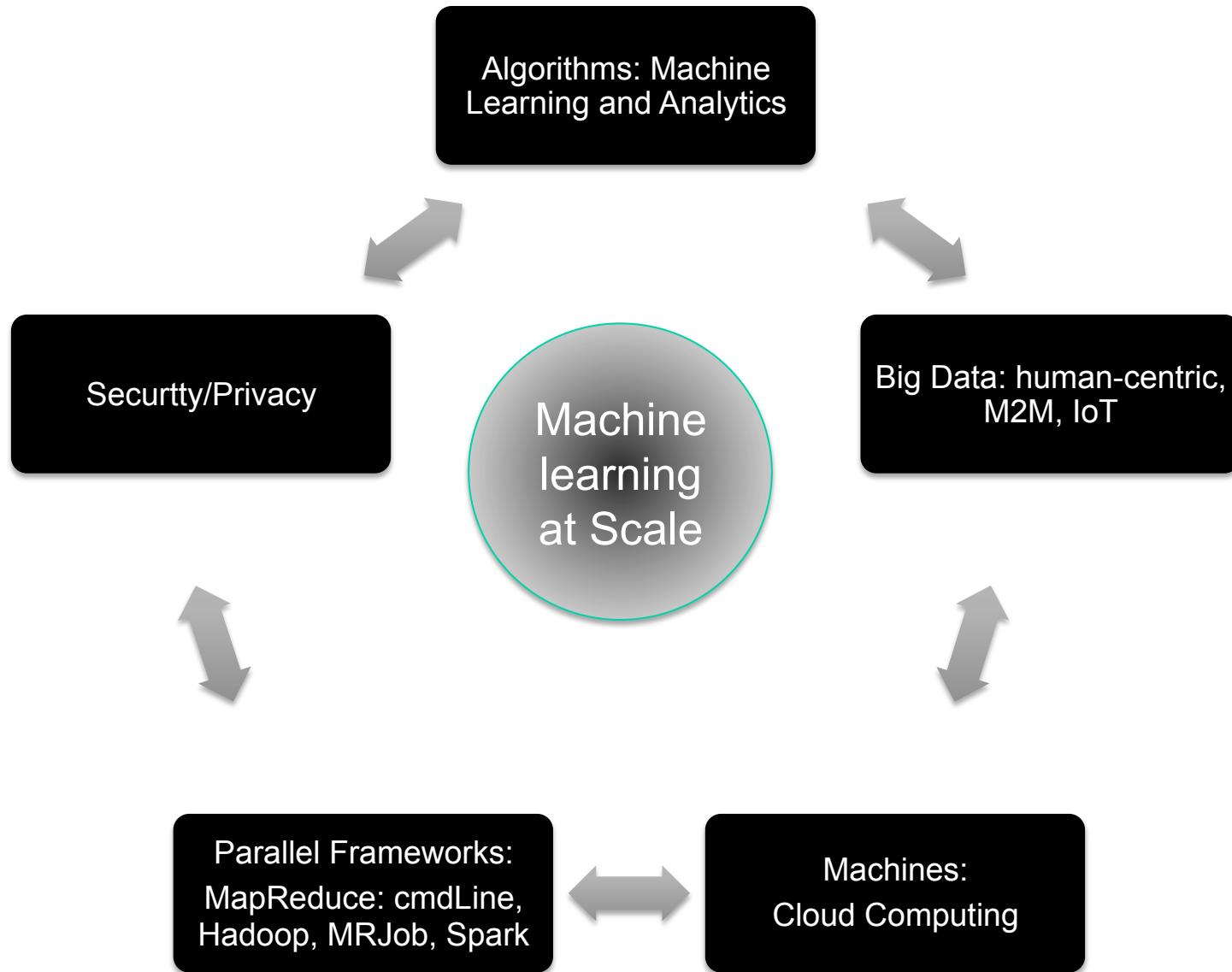
- **Part 1: Introduction**

- Welcome Survey
- Install Spark
- Background and Motivation
 - Big Data Science
 - Functional Programming
 - Poorman's Map-Reduce (to dividing and conquering)

- **Background and Motivation**

- Big Data Science
- Functional Programming
- Poorman's Map-Reduce (to dividing and conquering)

Machine learning at Scale



Big data Definition: use

- Big data is a broad term for data sets so large or complex that traditional data processing applications are inadequate.
 - PROCESSING:
 - Think of your laptop that gets overwhelmed with 3-4 gig of data (disk space is 1TB)
 - STORAGE:
 - Laptop : 1 TB
 - THROUGH-PUT
 - 1TB would take 3 hours to read it using your laptop
- Challenges
 - Challenges include analysis, capture, data curation, search, sharing, storage, transfer, visualization, security, and information privacy.

-
- In 2012, Gartner updated its definition as follows:
"Big data is high volume, high velocity, and/or
high variety information assets that require new
forms of processing to enable enhanced decision
making, insight discovery and process
optimization."[18]

Big Data: V³

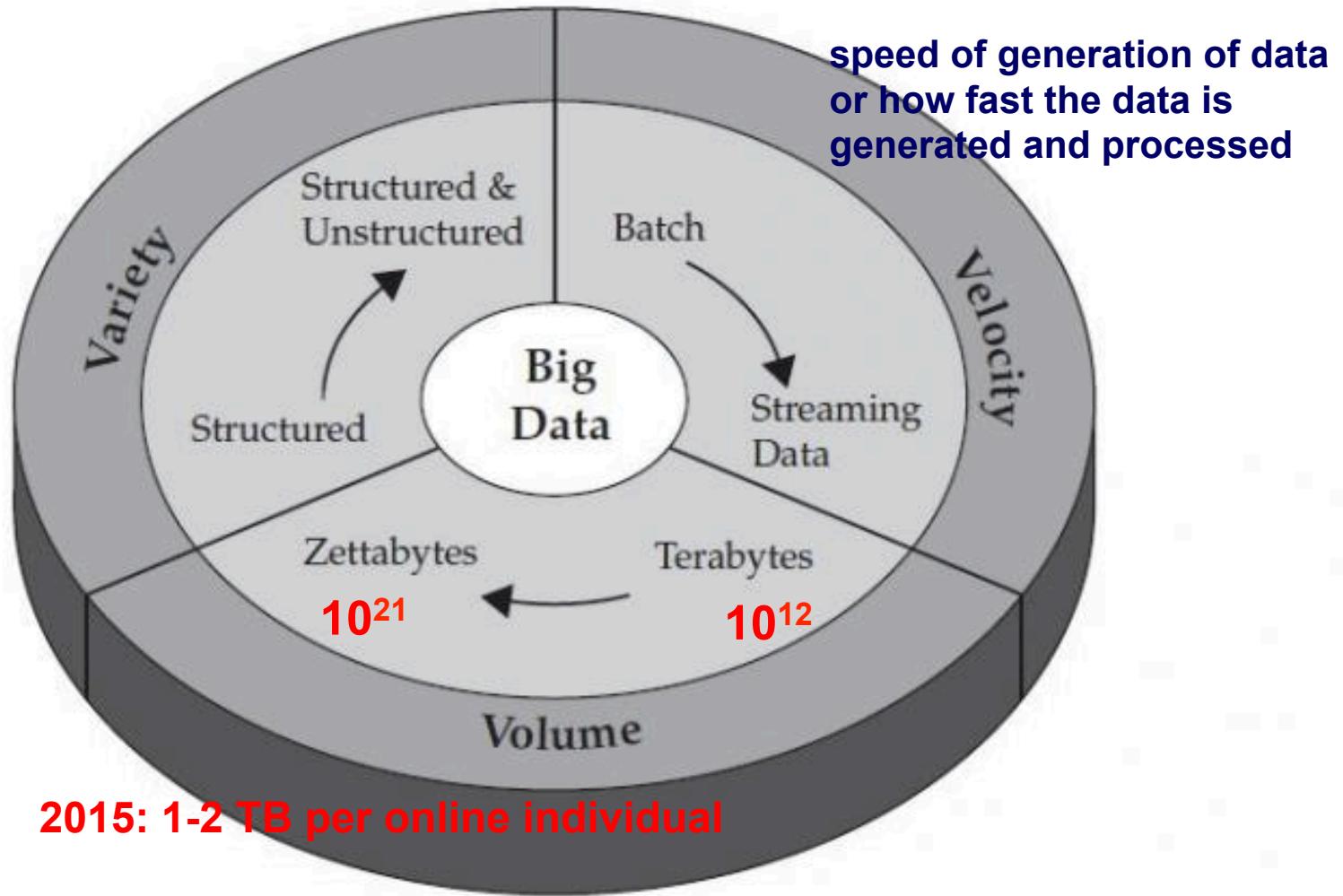
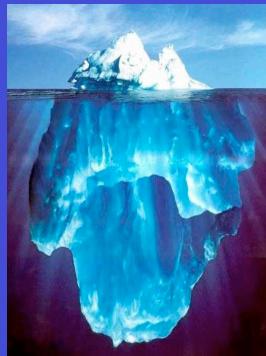


Figure 1-1 IBM characterizes Big Data by its volume, velocity, and variety—or simply, V³.

Sources Driving Big Data

It's All Happening On-line



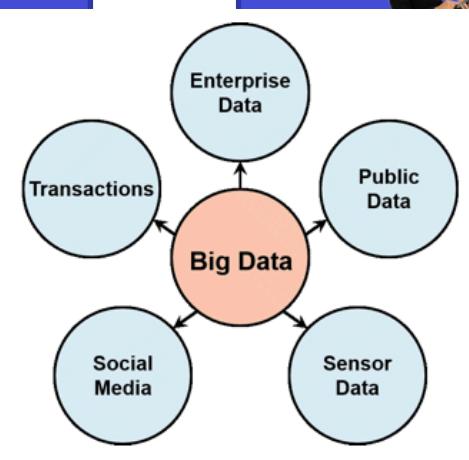
Every:
Click
Ad impression
Billing event
Fast Forward, pause,...
Friend Request
Transaction
Network message
Fault
...

User Generated (Web, Social & Mobile) Quantified Self

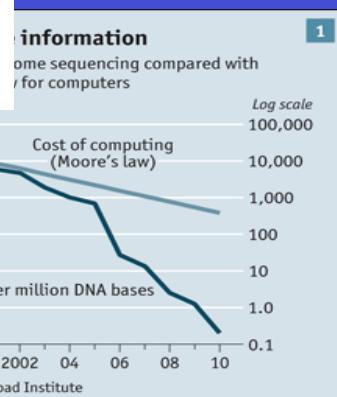


...

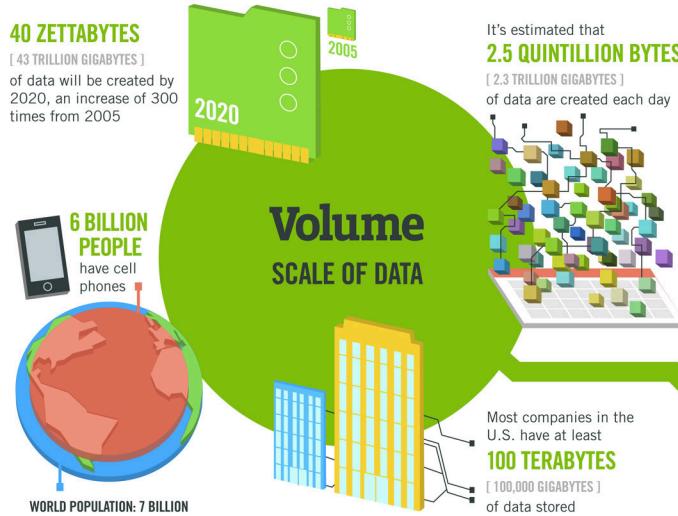
Internet of Things / M2M



Cloud Computing



By 2005 we had $120 \cdot 10^{18}$
 By 2007 we had $280 \cdot 10^{18}$
 By 2020 we will have $40 \cdot 10^{21}$



The New York Stock Exchange captures

1 TB OF TRADE INFORMATION during each trading session



Modern cars have close to **100 SENSORS** that monitor items such as fuel level and tire pressure

Velocity ANALYSIS OF STREAMING DATA

By 2016, it is projected there will be

18.9 BILLION NETWORK CONNECTIONS

- almost 2.5 connections per person on earth



Big Data Infographic

The FOUR V's of Big Data

From traffic patterns and music downloads to web history and medical records, data is recorded, stored, and analyzed to enable the technology and services that the world relies on every day. But what exactly is big data, and how can these massive amounts of data be used?

As a leader in the sector, IBM data scientists break big data into four dimensions: **Volume**, **Velocity**, **Variety** and **Veracity**.

Depending on the industry and organization, big data encompasses information from multiple internal and external sources such as transactions, social media, enterprise content, sensors and mobile devices. Companies can leverage data to adapt their products and services to better meet customer needs, optimize operations and infrastructure, and find new sources of revenue.

By 2015

4.4 MILLION IT JOBS will be created globally to support big data, with 1.9 million in the United States



As of 2011, the global size of data in healthcare was estimated to be

150 EXABYTES [161 BILLION GIGABYTES]



30 BILLION PIECES OF CONTENT

are shared on Facebook every month



Variety DIFFERENT FORMS OF DATA



By 2014, it's anticipated there will be **420 MILLION WEARABLE, WIRELESS HEALTH MONITORS**

4 BILLION+ HOURS OF VIDEO are watched on YouTube each month



400 MILLION TWEETS are sent per day by about 200 million monthly active users

1 IN 3 BUSINESS LEADERS don't trust the information they use to make decisions



Poor data quality costs the US economy around **\$3.1 TRILLION A YEAR**



27% OF RESPONDENTS

in one survey were unsure of how much of their data was inaccurate

Veracity UNCERTAINTY OF DATA

The quality of the data being captured can vary greatly

Sources: McKinsey Global Institute, Twitter, Cisco, Cartier, EMC, SAS, IBM, MERTEC, QAS

<http://www.ibmbigdatahub.com/info/infographic/>

http://www.ibmbigdatahub.com/sites/default/files/infographic_file/4-Vs-of-big-data.jpg

3 Vs of Big Data

1-2 TB per person today 2014/2015

The data from these sources has a number of features that make it a challenge for a data warehouse:

Exponential Growth. An estimated 2.8ZB of data in 2012 is expected to grow to 40ZB by 2020. 85% of this data growth is expected to come from new types; with machine-generated data being projected to increase 15x by 2020. (Source IDC)

40TB per person by 2020

Varied Nature. The incoming data can have little or no structure, or structure that changes too frequently for reliable schema creation at time of ingest.

Value at High Volumes. The incoming data can have little or no value as individual, or small groups of records. But high volumes and longer historical perspectives can be inspected for patterns and used for advanced analytic applications.

Personal; society; M2M; crowdsourcing

- **Society**
 - Graphs: Social, professional;
 - Quantified self: Eating; Sleeping; exercising
 - Voting
 - Education
 - Healthcare.... Economics, shopping, etc.
- **Internet of things**
 - Tracking Wildebeests in Serengeti, Tanzania (not just with GPS tags, but also with cameras at key strategic locations through out the Serengeti
 - Population changes in species; Scheduling safaris
 - 1 Billion smart meters by 2020;
 - 1 Petabyte of data per day? $10^9 = 10^{12} \text{ to } 10^{15}$
 - 1 Billion smart meters (One megabye of data per device per day; Poll meter 1000 times per day; 1000 bytes of data each time)
 - Smart cities

Tipping point: Humans no longer the center to the data universe

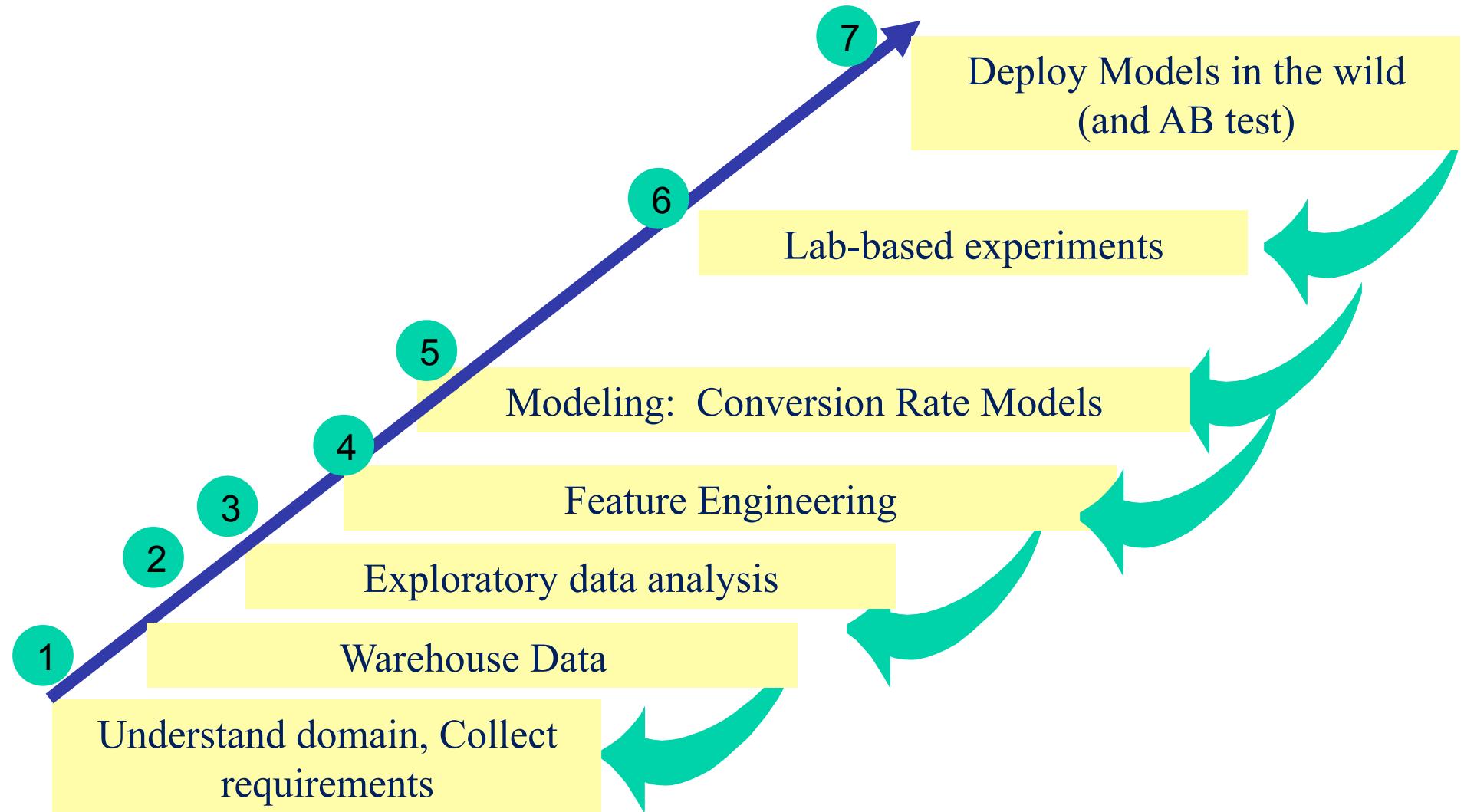
THE INTERNET OF THINGS



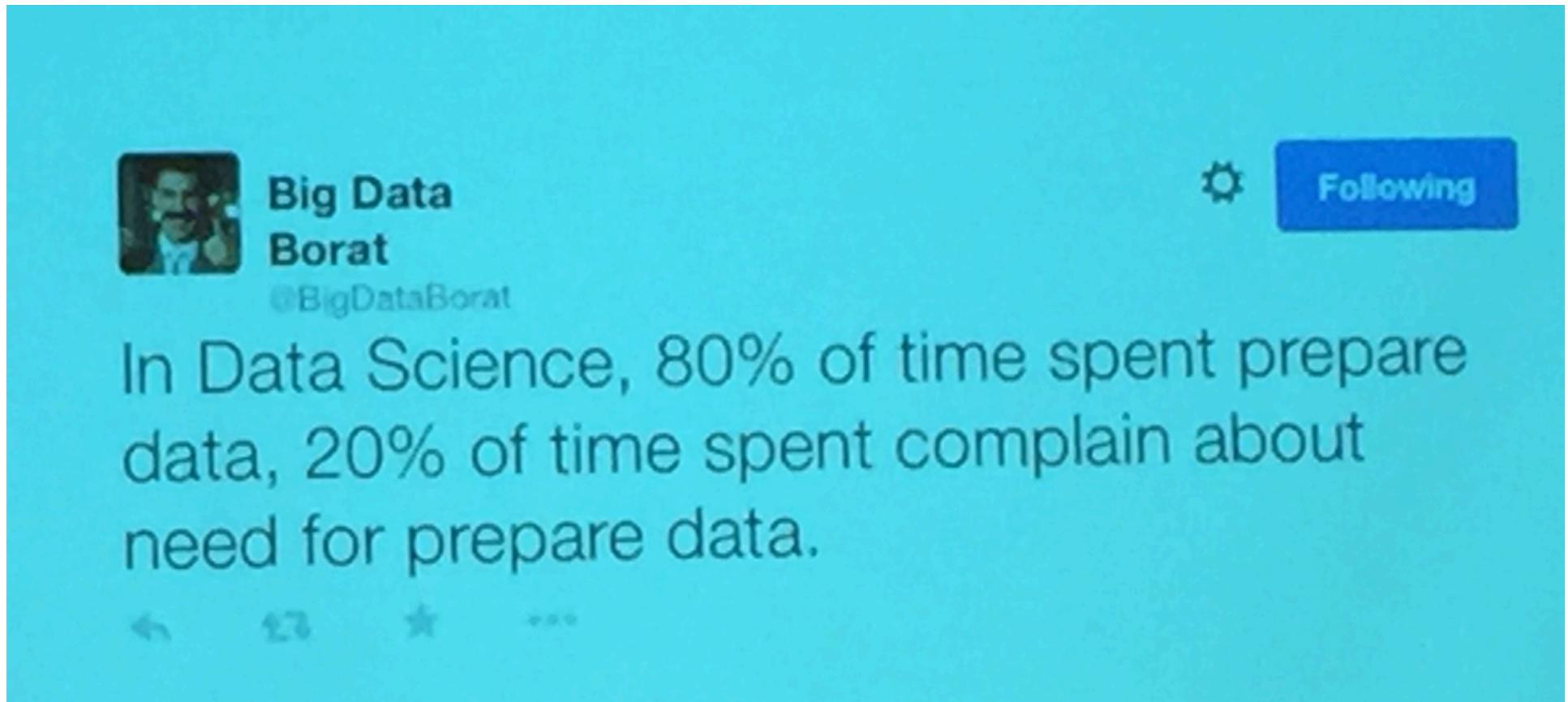
We've reached a tipping point in history: today more data is being manufactured by machines — servers, cell phones, GPS-enabled cars — than by people.

-
- **Big data applications described in the previous sections have generated new opportunities and business needs**
 - **This presents a literally a massive opportunity and challenge for machine learning**
 - **ML does not live in a vacuum (although it used for me in grad school)**
 - **These days ML requires a sophisticated ecosystem.**

7 Steps in Modeling: Conversion Rate Modeling



Where does time go in large scale machine learning?



Big Data Borat
@BigDataBorat

In Data Science, 80% of time spent prepare data, 20% of time spent complain about need for prepare data.

123 3

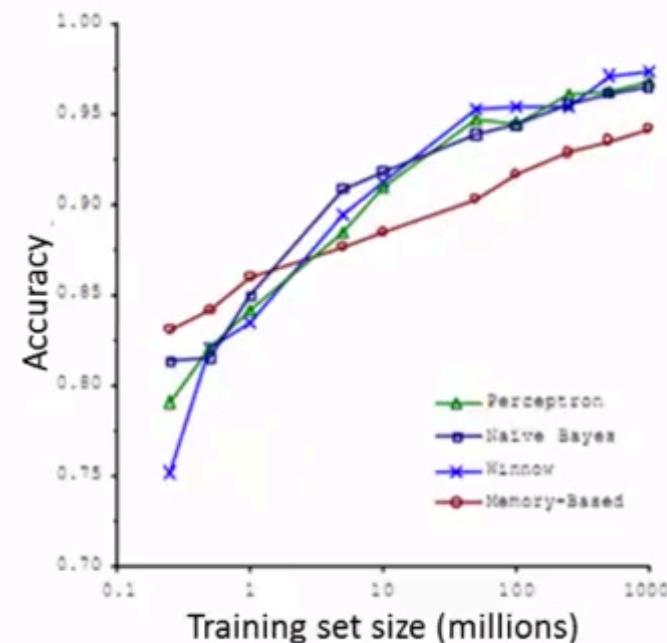
-
- **A popular question today in the advent of big data**
 - **More data scientists versus more data (that another way to ask about do you want a model bias-variance)**
 - **Empirical studies suggest more data....But.....**

More data or more data science?

Machine learning and data

Classify between confusable words.
E.g., {to, two, too}, {then, than}.

For breakfast I ate _____ eggs.

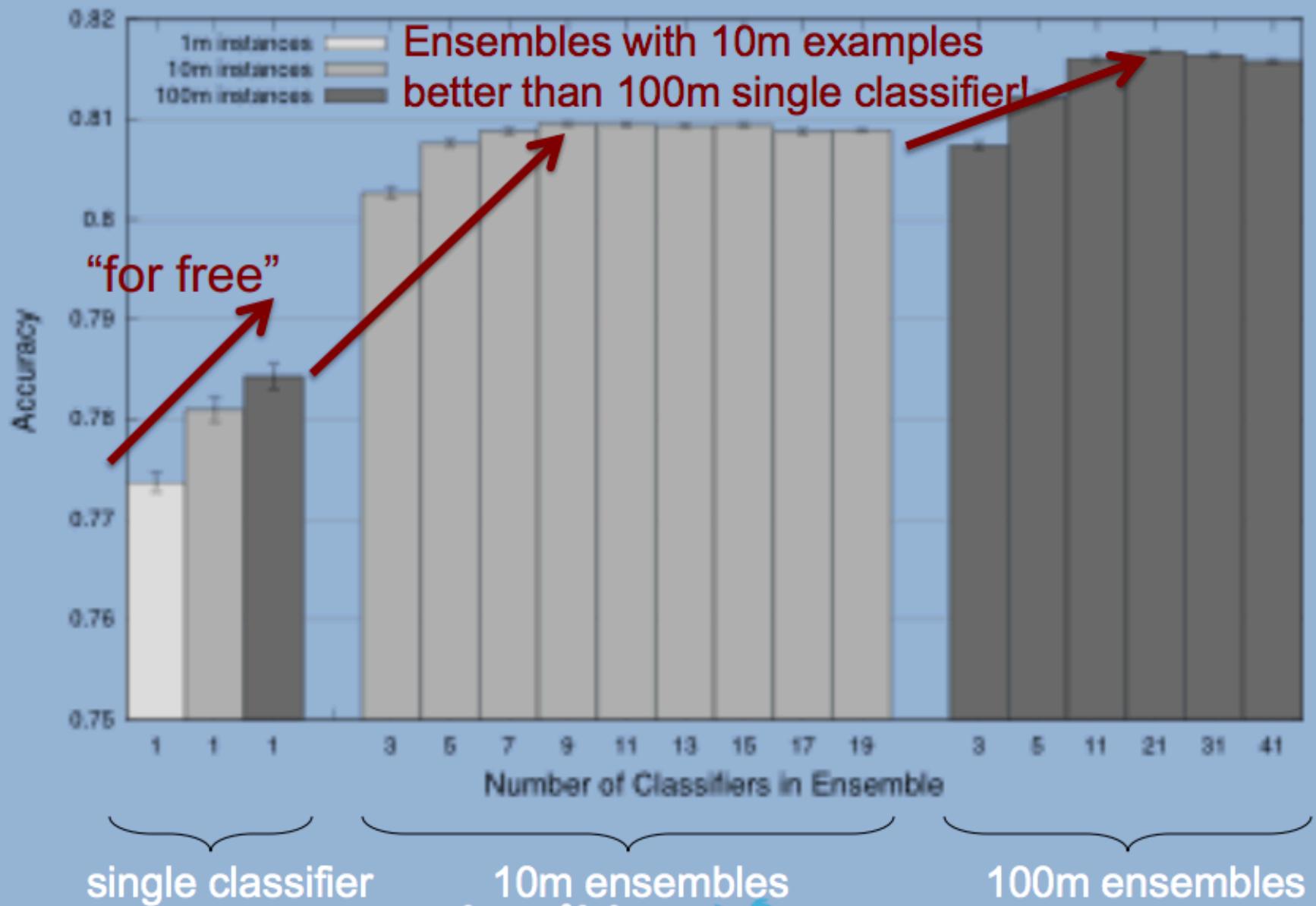


“It’s not who has the best algorithm that wins.
It’s who has the most data.”

[Figure from Banko and Brill, 2001]

Andrew Ng

Diminishing returns...



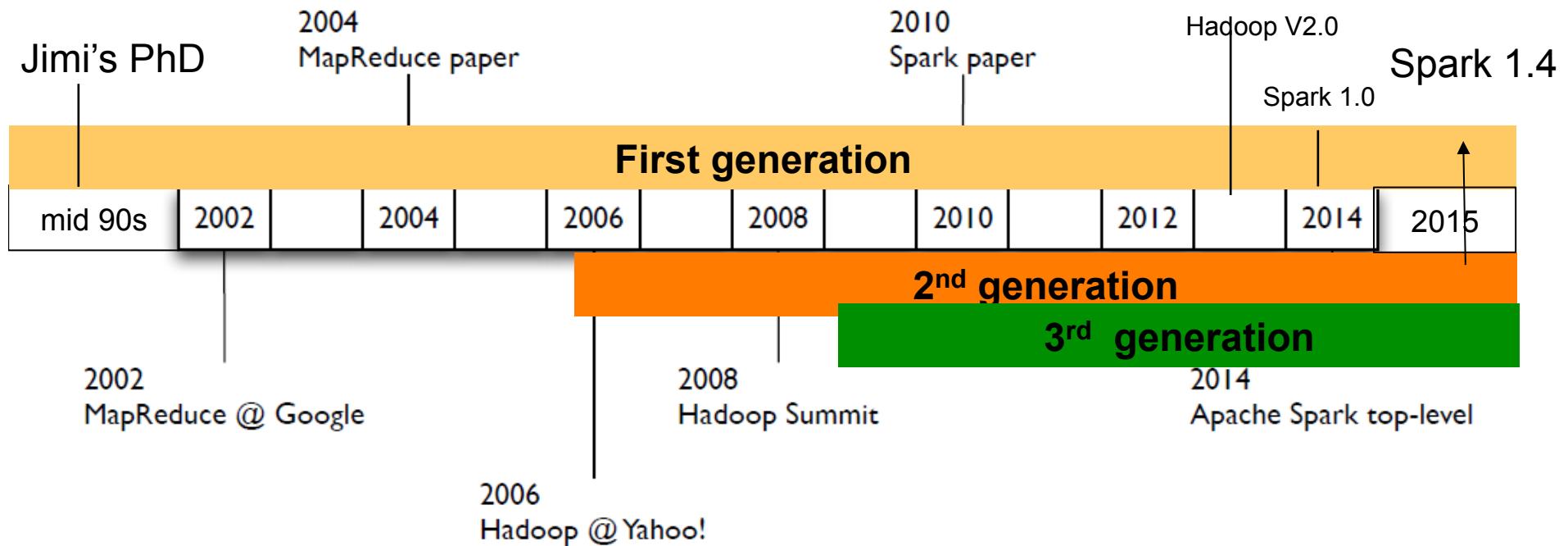
More data or more data scientists

- **Bias (more scientists → increased variance (overfit))**
 - Manage bias via models and features
 - Linear model on nonlinear data?
 - Polynomial model on linear data
- **Variance (more data → reduce variance)**
 - Manage variance with data
- **Look at a more formal characterization of this question in terms of bias-variance**

Three generations of machine learning

- **First generation: dataset that fits in memory**
 - Single node learning summary statistics and some batch modeling (at small scale); SQL, R
 - Down sampling the data
- **Second generation: General purpose clusters and frameworks**
 - Distributed frameworks that allows us to divide and conquer problems
 - Learning using general purpose frameworks such as hadoop big data analysis offline, realtime decision making, homegrown specialist systems (Hadoop for analysis and modeling;), Hadoop, R
 - In-house purpose built systems; specialist sport
- **Third generation: Purpose-built libraries and frameworks**
 - Built for iterative algorithms that are common place in ML
 - huge scale realtime analysis and decision making systems
 - Specialized frameworks for large scale manipulation the type of data you are working with.
 - For example, Machine learning libraries like MLLib in Spark, graph processing libraries like Apache Giraph or GraphX in Spark

Evolution of Map-Reduce frameworks for big data processing



Part 1

- **Part 1: Introduction**

- Welcome Survey
- Install Spark
- Background and Motivation
 - Big Data Science
 - Functional Programming
 - Poorman's Map-Reduce (to dividing and conquering)

Functional Programming as a guide for Spark

- **apply()**
- **map()**
- **filter()**
- **reduce()**
- **Lambda functions**
- **List comprehensions**

Functional programming background

- Treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.
- It is a declarative programming paradigm, which means programming is done with expressions.
- In functional code, the output value of a function depends only on the arguments that are input to the function, so calling a function f twice with the same value for an argument x will produce the same result $f(x)$ each time.
- No side effects
 - (side effects, i.e. changes in state that do not depend on the function inputs)
 - Can make it much easier to understand and predict the behavior of a program, which is one of the key motivations for the development of functional programming.

Functional Programming

In computer science, **functional programming** is a programming paradigm, a style of building the structure and elements of computer programs, that treats **computation** as the evaluation of **mathematical functions** and avoids **changing-state** and **mutable data**. It is a **declarative programming** paradigm, which means programming is done with **expressions**. In functional code, the output value of a function depends only on the arguments that are input to the function, so calling a function f twice with the same value for an argument x will produce the same result $f(x)$ each time. Eliminating **side effects**, i.e. changes in state that do not depend on the function inputs, can make it much easier to understand and predict the behavior of a program, which is one of the key motivations for the development of functional programming.

Functional programming has its roots in **lambda calculus**, a formal system developed in the 1930s to investigate **computability**, the **Entscheidungsproblem**, function definition, function application, and **recursion**. Many functional programming languages can be viewed as elaborations on the lambda calculus. In the other well-known declarative programming paradigm, **logic programming**, **relations** are at the base of respective languages.^[1]

In contrast, **imperative programming** changes state with commands in the source language, the most simple example being assignment. Imperative programming does have functions, not in the mathematical sense, but in the sense of **subroutines**. They can have **side effects** that may change the value of program state. Functions without return values therefore make sense. Because of this, they lack **referential transparency**, i.e. the same language expression can result in different values at different times depending on the state of the executing program.^[1]

Functional programming languages, especially **purely functional** ones such as **Hope** and **Rex**, have largely been emphasized in **academia** rather than in commercial software development. However, prominent functional programming languages such as **Common Lisp**, **Scheme**,^[2]^[3]^[4]^[5] **Clojure**, **Racket**,^[6] **Erlang**,^[7]^[8]^[9] **OCaml**,^[10]^[11] **Haskell**,^[12]^[13] and **F#**^[14]^[15] have been used in industrial and commercial applications by a wide variety of organizations. Functional programming is also supported in **Large Scale Distributed Data Science using Spark** © KDD2015 James G. Shanahan Contact:james.shanahan@gmail.com 00

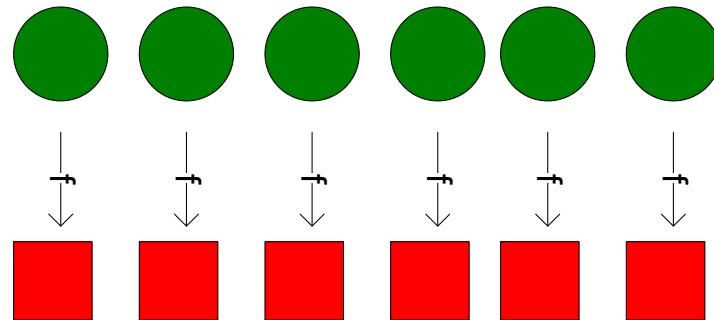
Higher-Order Functions

- A key feature of functional languages is the concept of higher order functions, or functions that can accept other functions as arguments (e.g., APL, Lisp and ML)
- A higher-order function is a function that takes another function as a parameter
- They are “higher-order” because it’s a function of a function
- Examples
 - Map (aka apply to all)
 - Reduce (aka fold)
 - Filter
- Lambda works great as a parameter to higher-order functions if you can deal with its limitations

Map

```
map(function, iterable, ...)
```

- Map applies **function** to each element of **iterable** and creates a list of the results
- You can optionally provide more iterables as parameters to map and it will place tuples in the result list
- Map returns an iterator which can be cast to list

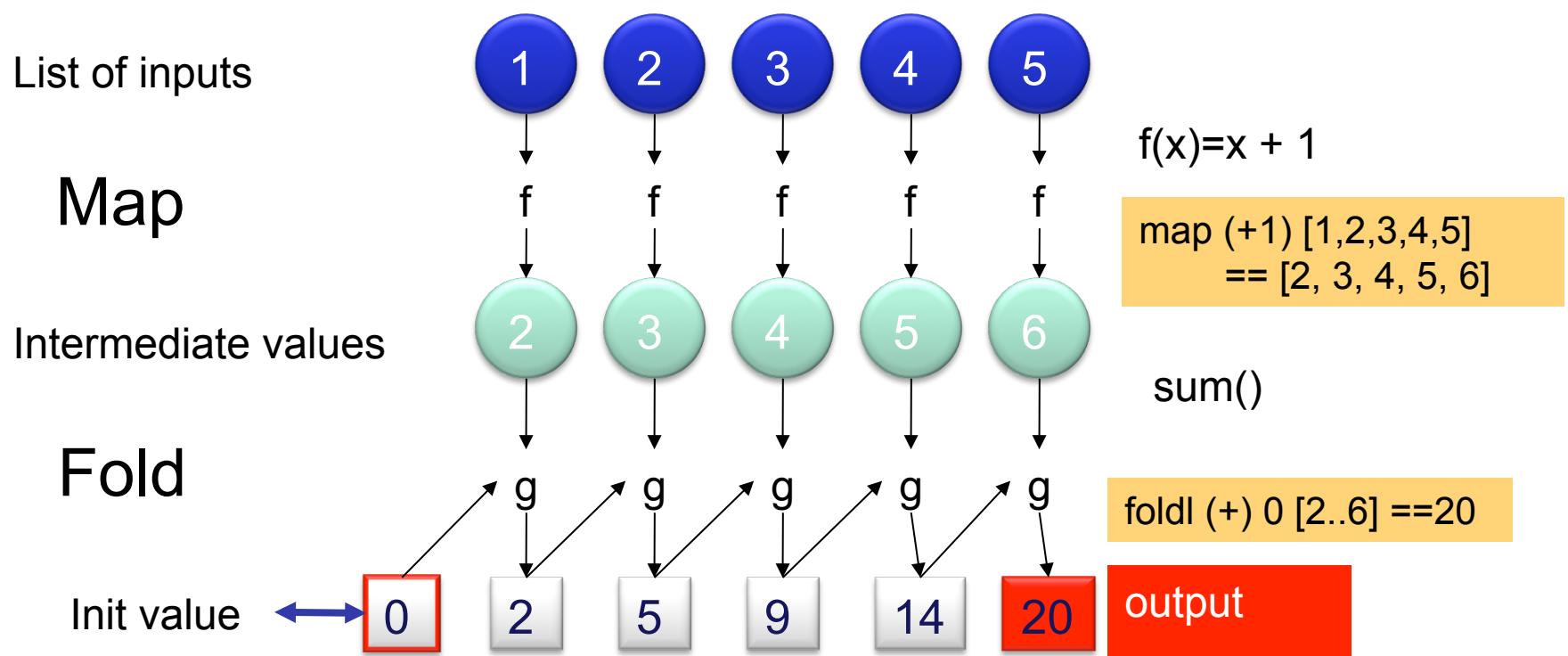


This is an abstract diagram, despite this you can see the potential for parallelism especially in the map phase

MapReduce is derived from FP

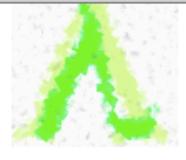
MapReduce ~ Map + Fold from functional programming!

Recursive defn but this unrolls into a loop



Lambda functions

- Shorthand version of def statement, useful for “Inlining” functions and other situations where it's convenient to keep the code of the function close to where it's needed
- Can only contain an expression in the function definition, not a block of statements (e.g., no if statements, etc)
- A lambda returns a function; the programmer can decide whether or not to assign this function to a name



Python 2 Tutorial

- History and Philosophy of Python
- Why Python?
- Interactive Mode
- Execute a Script
- Structuring with Indentation
- Data Types and Variables
- Operators
- input and raw_input via the keyboard
- Conditional Statements
- While Loops
- For Loops
- Formatted output
- Output with Print
- Sequential Data Types
- Dictionaries
- Sets and Frozen Sets
- Shallow and Deep Copy

Lambda, filter, reduce and map

Lambda Operator

Some like it, others hate it and many are afraid of the lambda operator. We are confident that you will like it, when you have finished with this chapter of our tutorial. If not, you can learn all about "[List Comprehensions](#)", Guido van Rossum's preferred way to do it, because he doesn't like Lambda, map, filter and reduce either.

becasu The lambda operator or lambda function is a way to create small anonymous functions, i.e. functions without a name. These functions are throw-away functions, i.e. they are just needed where they have been created. Lambda functions are mainly used in combination with the functions filter(), map() and reduce(). The lambda feature was added to Python due to the demand from Lisp programmers.

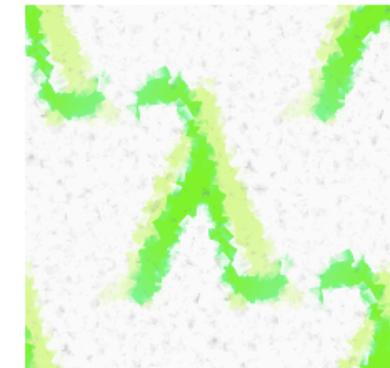
The general syntax of a lambda function is quite simple:

lambda argument_list: expression

The argument list consists of a comma separated list of arguments and the expression is an arithmetic expression using these arguments. You can assign the function to a variable to give it a name.

The following example of a lambda function returns the sum of its two arguments:

```
>>> f = lambda x, y : x + y  
>>> f(1,1)  
2
```



The map() Function

The advantage of the lambda operator can be seen when it is used in combination with the map() function. map() is a function with two arguments:

```
r = map(func, seq)
```

The first argument *func* is the name of a function and the second a sequence (e.g. a list) *seq*. *map()* applies the function *func* to all the elements of the sequence *seq*. It returns a new list with the elements changed by *func*

```
def fahrenheit(T):  
    return ((float(9)/5)*T + 32)  
def celsius(T):  
    return (float(5)/9)*(T-32)  
temp = (36.5, 37, 37.5, 39)  
  
F = map(fahrenheit, temp)  
C = map(celsius, F)
```

In the example above we haven't used lambda. By using lambda, we wouldn't have had to define and name the functions fahrenheit() and celsius(). You can see this in the following interactive session:

```
>>> Celsius = [39.2, 36.5, 37.3, 37.8]  
>>> Farenheit = map(lambda x: (float(9)/5)*x + 32, Celsius)  
>>> print Farenheit  
[102.56, 97.70000000000003, 99.14000000000001, 100.03999999999991]
```

Lambda example

- Simple example:

```
>>> def sum(x,y): return x+y  
>>> ...  
>>> sum(1,2)  
3
```

```
>>> sum2 = lambda x, y: x+y  
>>> sum2(1,2)  
3
```

Closure (computer programming)

From Wikipedia, the free encyclopedia

Closure

For other uses of this term, including in mathematics and computer science, see [Closure](#).

Not to be confused with [Clojure](#).

In programming languages, **closures** (also **lexical closures** or **function closures**) are a technique for implementing lexically scoped name binding in languages with [first-class functions](#). Operationally, a closure is a data structure storing a [function](#)^[a] together with an environment:^[1] a mapping associating each free variable of the function (variables that are used locally, but defined in an enclosing scope) with the value or storage location the name was bound to at the time the closure was created.

Example The following program defines a function `startAt` that returns a function `incrementBy`. The nested function `incrementBy` adds its argument `y` to the value of `x`, even though `x` is not local to `incrementBy`. This is because `incrementBy` "captures" the `x` variable from the outer scope and associates it with the `y` value to the `x` value, and finds it once invoked:

```
function startAt(x)
  function incrementBy(y)
    return x + y
  return incrementBy

variable closure1 = startAt(1)
variable closure2 = startAt(2)
```

Note that, as `startAt` returns a function, `closure1` and `closure2` are associated environments differ, and therefore evaluate to different values, thus evaluating the same code to different results.

A closure is a data structure storing a function[a] together with an environment:

- **a mapping associating each free variable of the function (variables that are used locally, but defined in an enclosing scope) with the value or storage location the name was bound to at the time the closure was created.**
- A closure is a function that encloses its surrounding state by referencing fields external to its body. The enclosed state remains across invocations of the closure.

Closure (computer programming)

From Wikipedia, the free encyclopedia

For other uses of this term, including in mathematics and computer science, see [Closure](#).

Not to be confused with [Clojure](#).

In programming languages, **closures** (also **lexical closures** or **function closures**) are a technique for implementing lexically scoped name binding in languages with [first-class functions](#). Operationally, a closure is a data structure storing a [function](#)^[a] together with an environment:^[1] a mapping associating each [free variable](#) of the function (variables that are used locally, but defined in an enclosing scope) with the [value](#) or [storage location](#) the name was bound to at the time the closure was created.^[b] A closure—unlike a plain function—allows the function to access those *captured variables* through the closure's reference to them, even when the function is invoked outside their scope.

Example The following program fragment defines a [higher-order function](#) `startAt` with a parameter `x` and a [nested function](#) `incrementBy`. The nested function `incrementBy` has access to `x`, because `incrementBy` is in the lexical scope of `x`, even though `x` is not local to `incrementBy`. The function `startAt` returns a closure containing the function `incrementBy`, which adds the `y` value to the `x` value, and a reference to the variable `x` from this invocation of `startAt`, so `incrementBy` will know where to find it once invoked:

```
function startAt(x)
  function incrementBy(y)
    return x + y
  return incrementBy

variable closure1 = startAt(1)
variable closure2 = startAt(5)
```

A closure is a data structure storing a function^[a] together with an environment:^[1] a mapping associating each free variable of the function (variables that are used locally, but defined in an enclosing scope) with the value or storage location the name was bound to at the time the closure was created.

Note that, as `startAt` returns a function, the variables `closure1` and `closure2` are of [function type](#). Invoking `closure1(3)` will return `4`, while invoking `closure2(3)` will return `8`. While `closure1` and `closure2` have the same function `incrementBy`, the associated environments differ, and invoking the closures will bind the name `x` to two distinct variables in the two invocations, with different values, thus evaluating the function to different results.

Lambda as a closure

Spark Essentials: Transformations

Scala:

```
val distFile = sc.textFile("README.md")
distFile.map(l => l.split(" ")).collect()
distFile.flatMap(l => l.split(" ")).collect()
```

A closure is a function that encloses its surrounding state by referencing fields external to its body. The enclosed state remains across invocations of the closure.

Python:

```
distFile = sc.textFile("README.md")
distFile.map(lambda x: x.split(' ')).collect()
distFile.flatMap(lambda x: x.split(' ')).collect()
```

closures

Map Example

Example

```
1  nums = [0, 4, 7, 2, 1, 0, 9, 3, 5, 6, 8, 0, 3]
2
3  nums = list(map(lambda x : x % 5, nums)) Lambda/inline
4
5  print(nums)
6  #[0, 4, 2, 2, 1, 0, 4, 3, 0, 1, 3, 0, 3]
7
8
9  def mod5(val) :
10     return x % 5
11
12 nums1 = list(map(mod5, nums))
print(nums1)
#[0, 4, 2, 2, 1, 0, 4, 3, 0, 1, 3, 0, 3]
```

apply()

- In very general contexts, you may not know ahead of time how many arguments need to get passed to a function (perhaps the function itself is built dynamically)
- The apply() function calls a given function with a list of arguments packed in a tuple:
`def sum(x, y): return x+y
apply(sum, (3, 4))`
7
- Apply can handle functions defined with def or with lambda

Map Example: distance from origin

Goal: given a list of three dimensional points in the form of tuples, create a new list consisting of the distances of each point from the origin

Loop Method:

- $\text{distance}(x, y, z) = \sqrt{x^2 + y^2 + z^2}$
- loop through the list and add results to a new list

Map Problem: distance from origin

Solution

```
1 from math import sqrt  
2  
3 points = [(2, 1, 3), (5, 7, -3), (2, 4, 0), (9, 6, 8)]  
4  
5 def distance(point) :  
6     x, y, z = point  
7     return sqrt(x**2 + y**2 + z**2)  
8  
9 distances = list(map(distance, points))
```

Filter: higher order function

```
filter(function, iterable)
```

- The filter runs through each element of **iterable** (any iterable object such as a List or another collection)
- It applies **function** to each element of **iterable**
- If **function** returns True for that element then the element is put into a List
- This list is returned from filter in versions of python under 3
- In python 3, filter returns an iterator which must be cast to type list with list()

Filter Example

Example

```
1  nums = [0, 4, 7, 2, 1, 0, 9, 3, 5, 6, 8, 0, 3]
2
3  nums = list(filter(lambda x : x != 0, nums))
4
5  print(nums)          #[4, 7, 2, 1, 9, 3, 5, 6, 8, 3]
6
```

Filter Problem

```
NaN = float("nan")
scores = [[NaN, 12, .5, 78, math.pi],
          [2, 13, .5, .7, math.pi / 2],
          [2, NaN, .5, 78, math.pi],
          [2, 14, .5, 39, 1 - math.pi]]
```

Goal: given a list of lists containing answers to an algebra exam, filter out those that did not submit a response for one of the questions, denoted by NaN

Filter Problem

Solution

```
1  NaN = float("nan")
2  scores = [[NaN, 12, .5, 78, pi],[2, 13, .5, .7, pi / 2],
3              [2,NaN, .5, 78, pi],[2, 14, .5, 39, 1 - pi]]
4  #solution 1 - intuitive
5  def has_NaN(answers) :
6      for num in answers :
7          if isnan(float(num)) :
8              return False
9      return True
0  valid = list(filter(has_NaN, scores))
1  print(valid)
2
3  #Solution 2 - sick python solution
4  valid2 = list(filter(lambda x : NaN not in x, scores))
5  print(valid2)
```

Lambda/inline

Reduce

```
reduce(function, iterable[,initializer])
```

- Reduce will apply **function** to each element in **iterable** along with the sum so far and create a cumulative sum of the results
- **function** must take two parameters
- If initializer is provided, initializer will stand as the first argument in the sum
- Unfortunately in python 3 reduce() requires an import statement
 - from functools import reduce

Reduce Example

Example

```
1  nums = [1, 2, 3, 4, 5, 6, 7, 8]
2
3  nums = list(reduce(lambda x, y : (x, y), nums))
4
5  Print(nums)          # (((((1, 2), 3), 4), 5), 6), 7), 8)
6
7
```

Reduce Problem

Goal: given a list of numbers I want to find the average of those numbers in a few lines using `reduce()`

For Loop Method:

- sum up every element of the list
- divide the sum by the length of the list

Reduce Problem

Solution

```
1  nums = [92, 27, 63, 43, 88, 8, 38, 91, 47, 74, 18, 16,  
2      29, 21, 60, 27, 62, 59, 86, 56]  
3  
4  sum = reduce(lambda x, y : x + y, nums) / len(nums)
```

MapReduce

A framework for processing huge datasets on certain kinds of distributable problems

Map Step:

- master node takes the input, chops it up into smaller sub-problems, and distributes those to worker nodes.
- worker node may chop its work into yet small pieces and redistribute again

MapReduce

Reduce Step:

- master node then takes the answers to all the sub-problems and combines them in a way to get the output

MapReduce

Problem: Given an email how do you tell if it is spam?

- Count occurrences of certain words. If they occur too frequently the email is spam.

map_reduce.py

```
1 email = ['the', 'this', 'annoy', 'the', 'the', 'annoy']
2
3 def inEmail (x):
4     if (x == "the"):
5         return 1;
6     else:
7         return 0;
8
9 map(inEmail, 1)                      #[1, 0, 0, 0, 1, 1, 0]
10
11 reduce((lambda x, xs: x + xs), map(inEmail, email)) #3
```

-
- Purely functional

Purely functional

Every function in Haskell is a function in the mathematical sense (i.e., "pure"). Even side-effecting IO operations are but a description of what to do, produced by pure code. There are no statements or instructions, only expressions which cannot mutate variables (local or global) nor access state like time or random numbers.

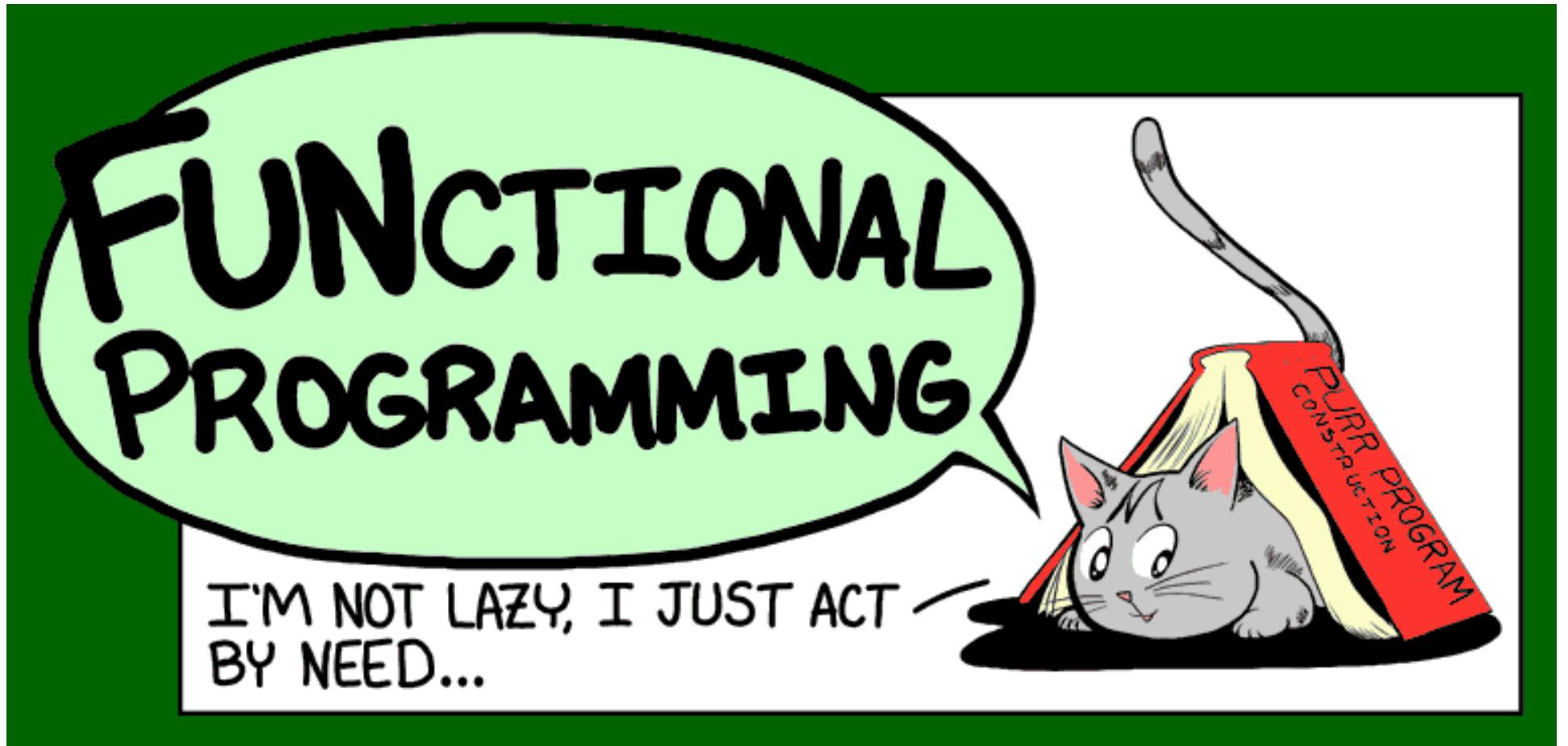
The following function takes an integer and returns an integer. By the type it cannot do any side-effects whatsoever, it cannot mutate any of its arguments.

```
square :: Int -> Int
square x = x * x
```

Functional Programming Review

- Functional operations do not modify data structures: They always create new ones
- Original data still exists in unmodified form
- Data flows are implicit in program design
- Order of operations does not matter
- Lists are primitive data types

Hadoop is not so lazy but Spark is!



-
- In this section we talked about functional programming and have seen the map and reduce higher order functions and we seen the potential to parallelize at least the map function
 - Over the next couple of sections we will see how the Spark framework has adopted FP constructs (albeit not purely observing the FP high standards)
 - Lazy evaluation
 - Data is immutable
 - Map, reduce, and 80 other functions

Part 1

- **Part 1: Introduction**

- Welcome Survey
- Install Spark
- Background and Motivation
 - Big Data Science
 - Functional Programming
 - Poorman's Map-Reduce (to dividing and conquering)

Assume 100 Billion webpages

- **How store them?**
 - $10K \text{ per page } 10^{11} = 10^{15} \text{ Bytes (1 Petabyte of raw data)}$
- **How can we scan them?**
 - $10^{15} \times 10^{-8} \times 10^{-5} = 10^2 \text{ days}$
- **How to do something useful with them?**
 - Document frequency for a term?
 - Extract outlinks of page and say just count the number of inlinks for a webpage

Why Parallelism?

- **Data size is increasing**
 - Single node architecture is reaching its limit
 - Scan 1000 TB on 1 node @ 100 MB/s = 120 days
 - Store that amount of data?
- **Growing demand to leverage data to do useful tasks**
- **Parallelism: Divide and conquer**
- **Standard/Commodity and affordable architecture emerging**
 - Cluster of commodity Linux nodes (CPU, memory and disk)
 - Gigabit ethernet interconnect

Design Goals for Parallelism

1. Scalability to large data volumes:

- Scan 1000 TB (1PB) on 1 node @ 50 MB/s = 120 days
- Scan on 10000-node cluster = 16 minutes!
- $10^{15} \times 10^{-8} \times 10^{-5} = 10^2$ days = 10^7 seconds
- 10^7 seconds X 10⁴ nodes = 10³ seconds = 16 minutes
- 100 Gig per node (10¹¹ bytes at 10⁸ / second)

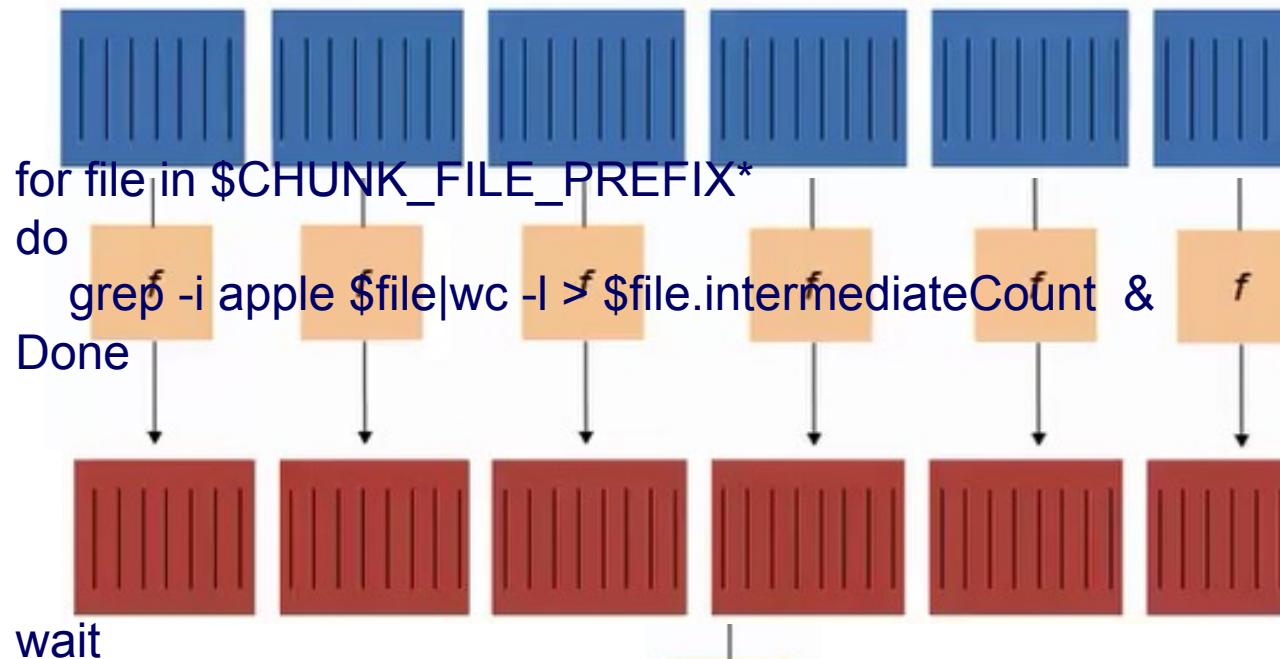
2. Cost-efficiency:

- Commodity nodes (cheap, but unreliable)
- Commodity network
- Automatic fault-tolerance (fewer admins)
- Easy to use (fewer programmers)

120

Schematic of Parallel Processing

1. #Splitting \$ORIGINAL_FILE into chunks ...
2. split -b 10000M \$ORIGINAL_FILE \$CHUNK_FILE_PREFIX



A big data file

File is split into smaller 100Gig chunks and distributed to k nodes

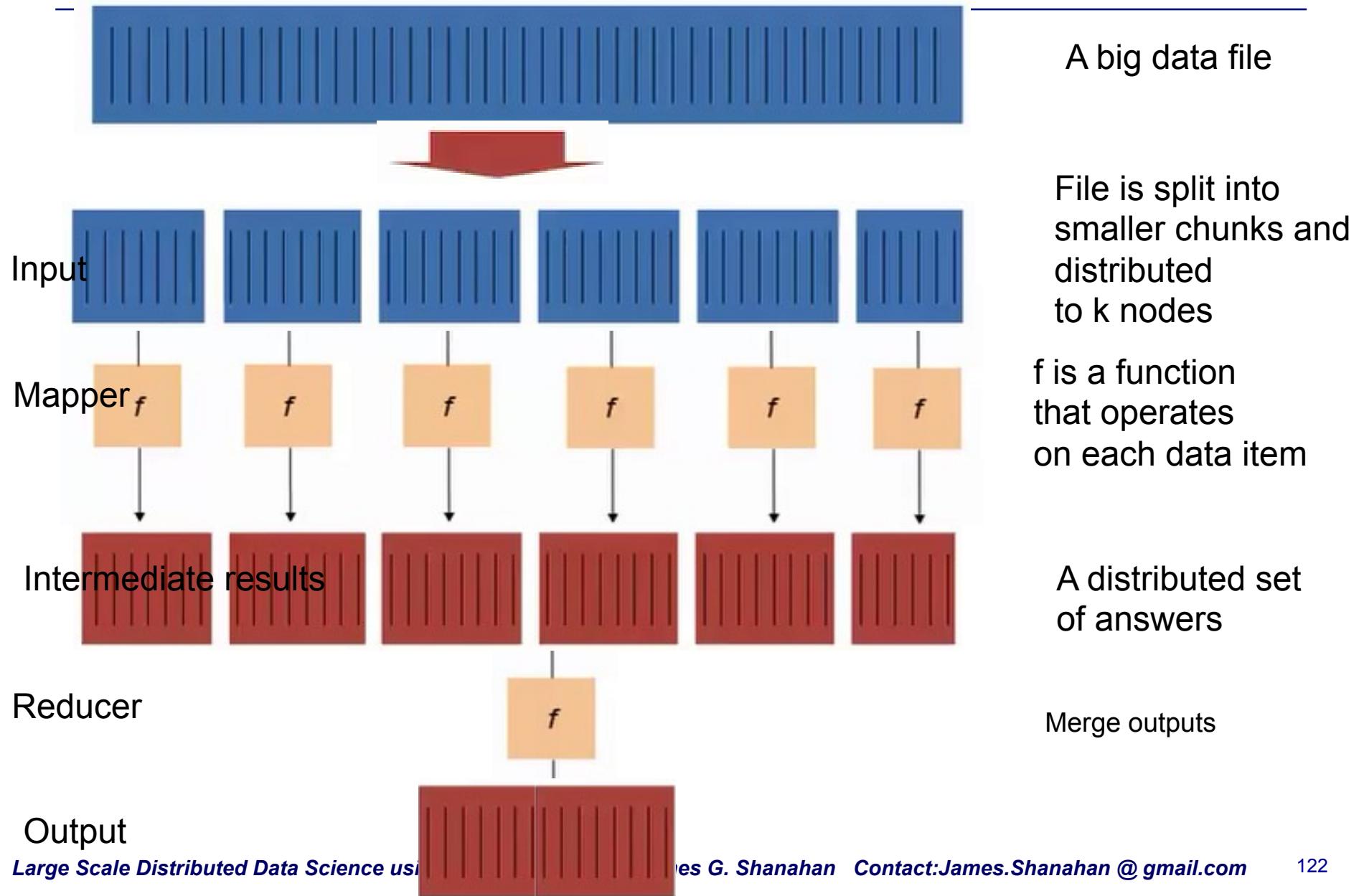
f is a function that operates on each data item

A distributed set of answers

Merge outputs

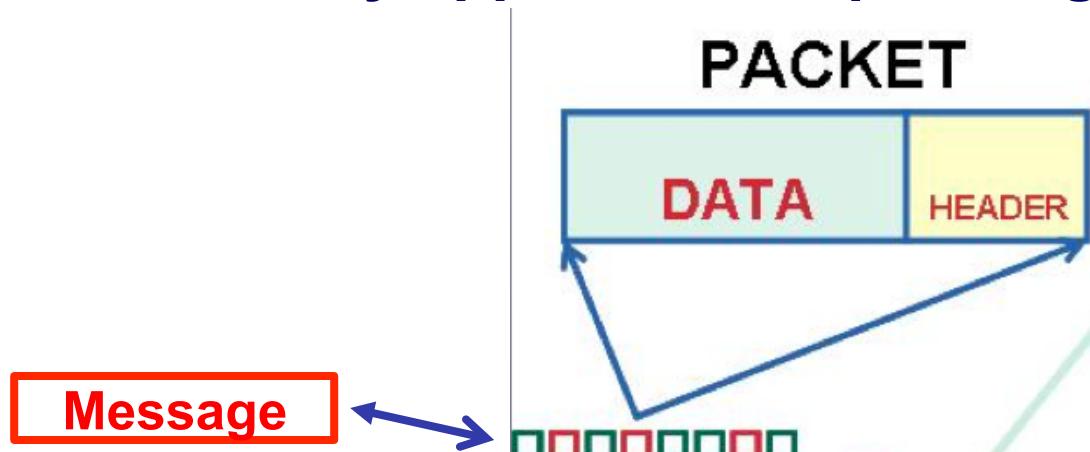
```
#Merging intermediate Count can take first column and total...
echo "found this many apples in the Facebook posts log file"
cat *.intermediateCount cut -f 1 | paste -sd+ - | bc
```

Schematic of Parallel Processing

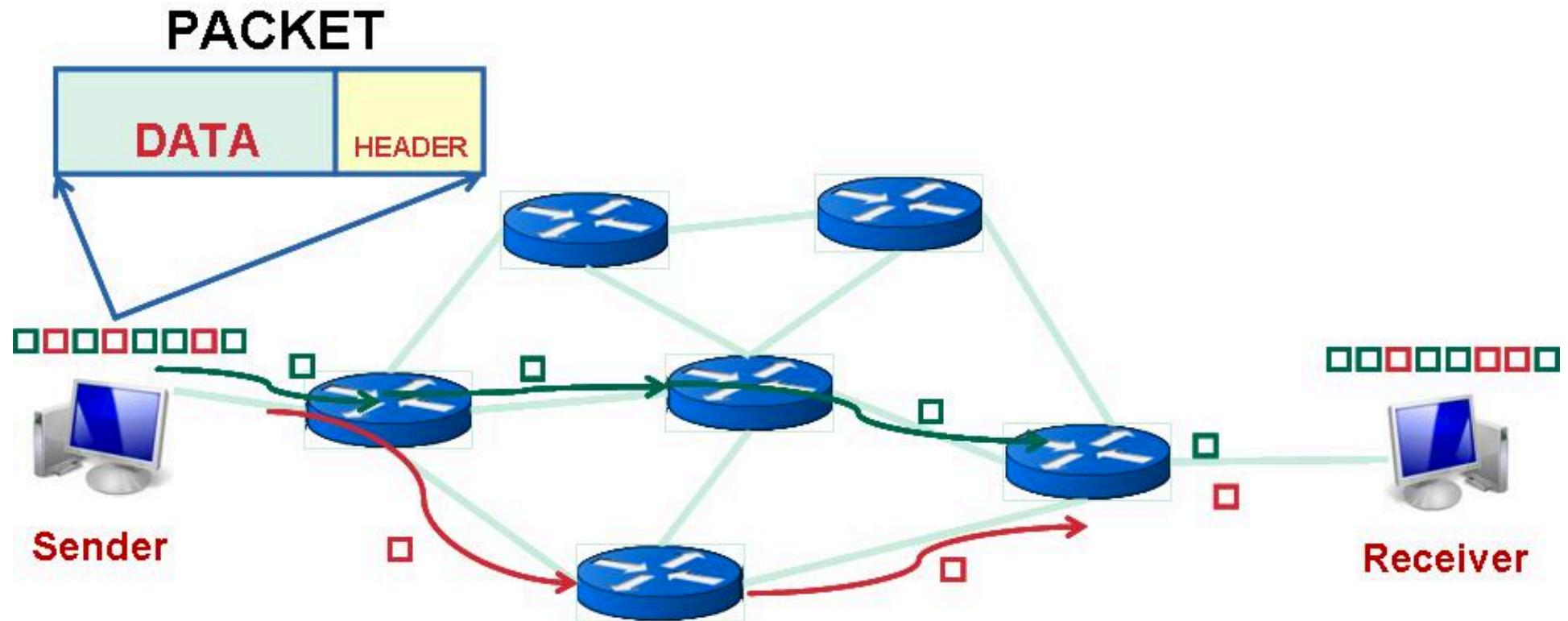


D&C is common: e.g., Packet switching

- Packet switching is a digital networking communications method that transmits messages in chunks or blocks, called *packets*
- Packets are transmitted via a medium that may be shared by multiple simultaneous communication sessions.
- Packet switching increases network efficiency, robustness and enables technological convergence of many applications operating on the same network.

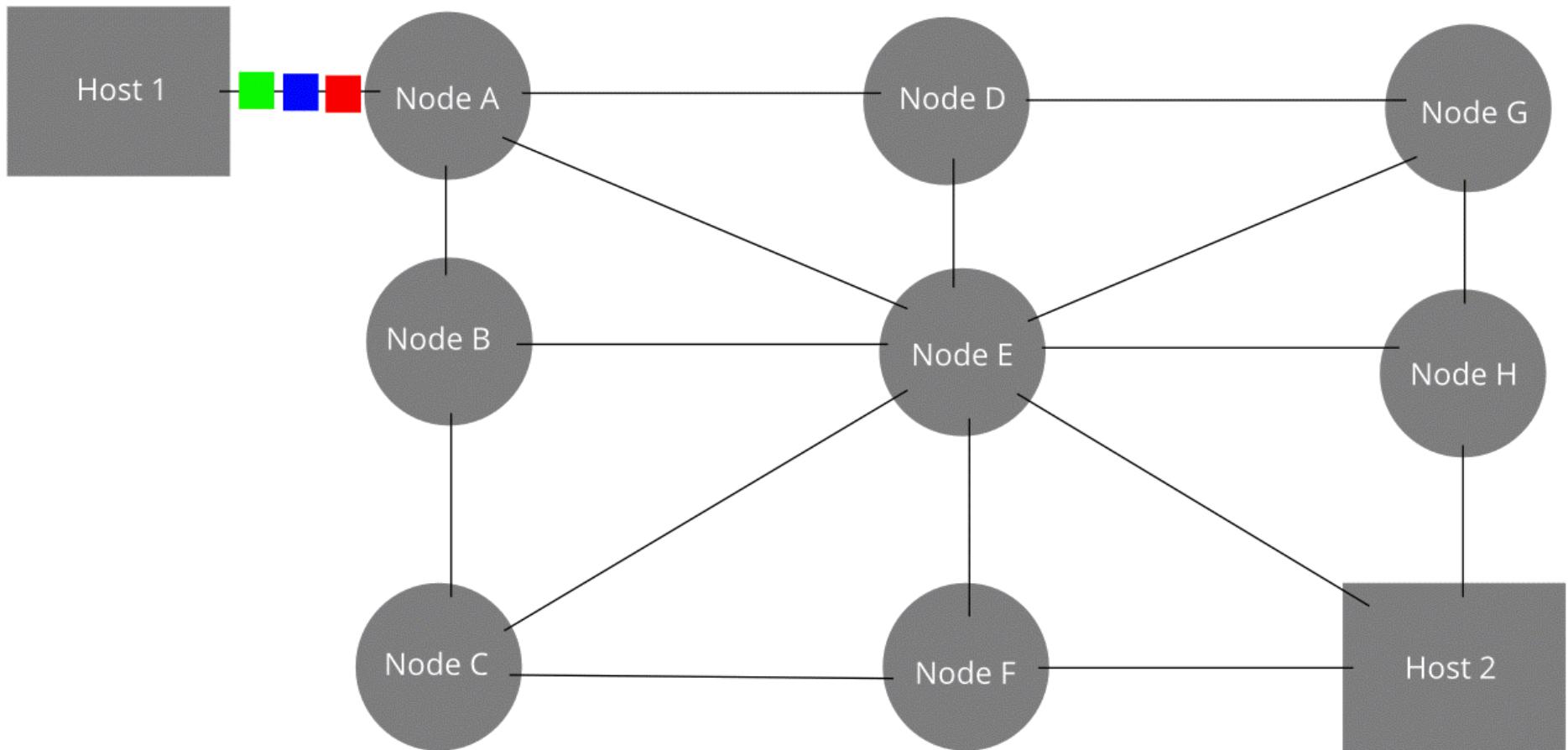


Packet = Header and Payload



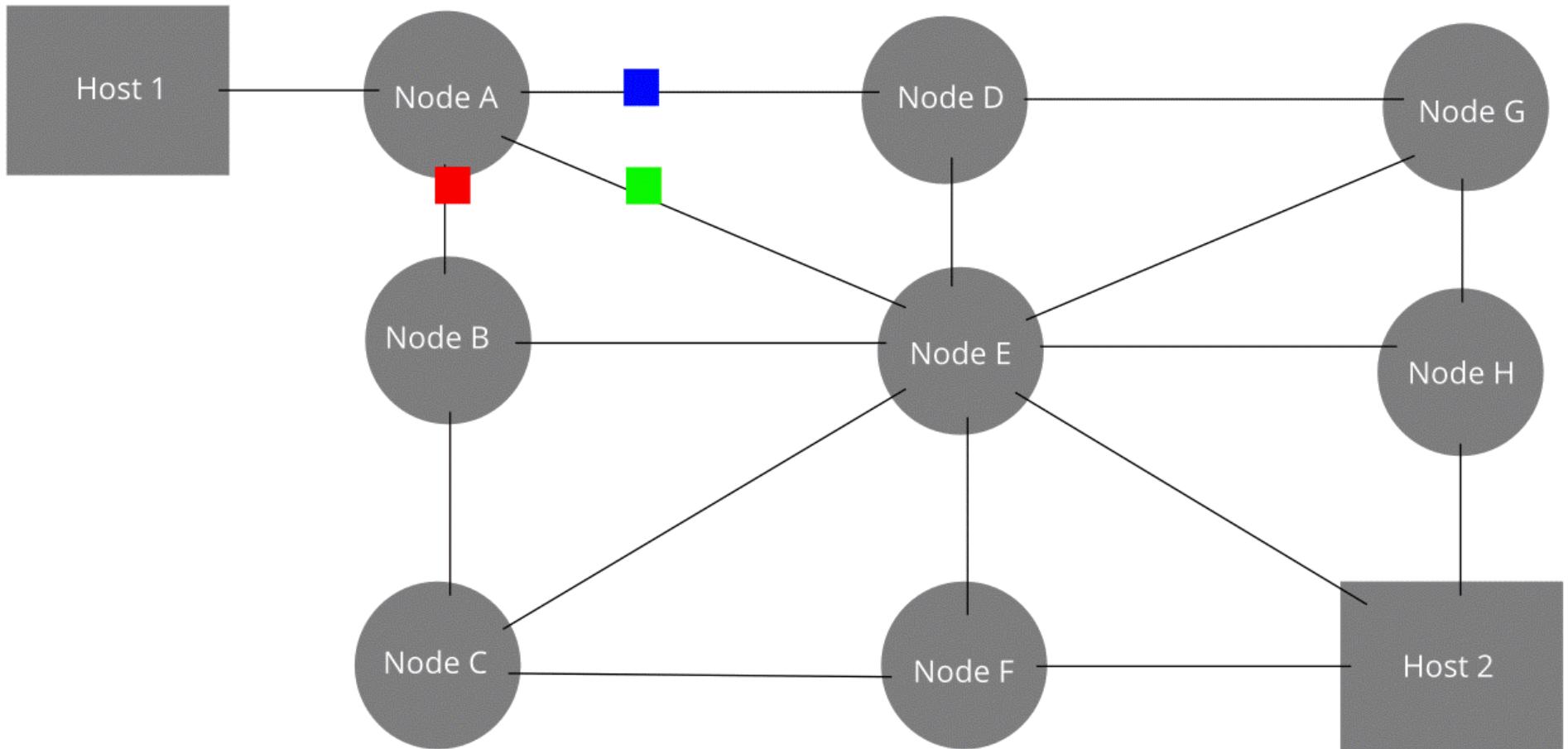
Message Passing: Divide and conquer/send

The original message is Green, Blue, Red.



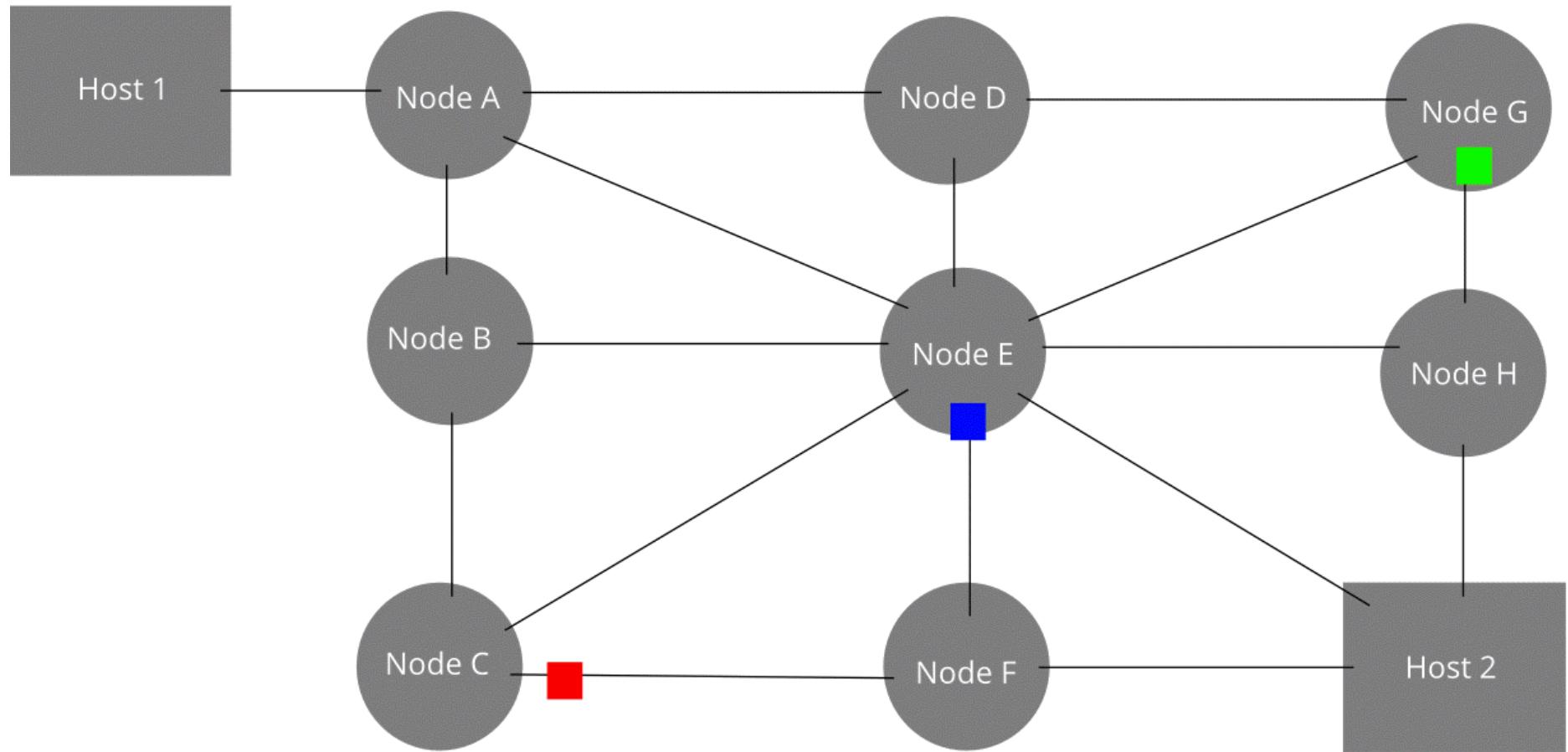
Route packets in parallel if possible

The data packets take different routes to their destinations.



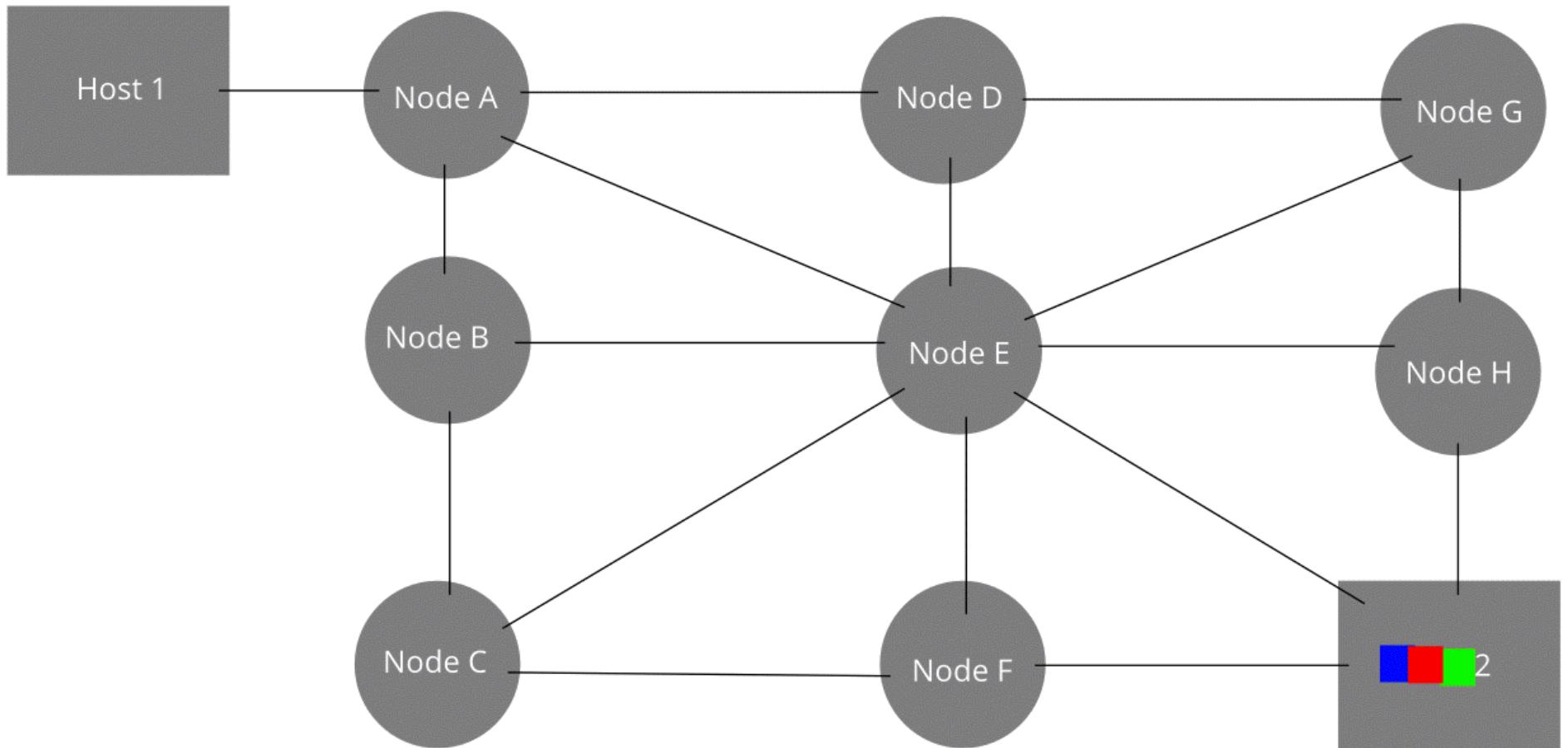
Routing packets in parallel if possible

The data packets take different routes to their destinations.



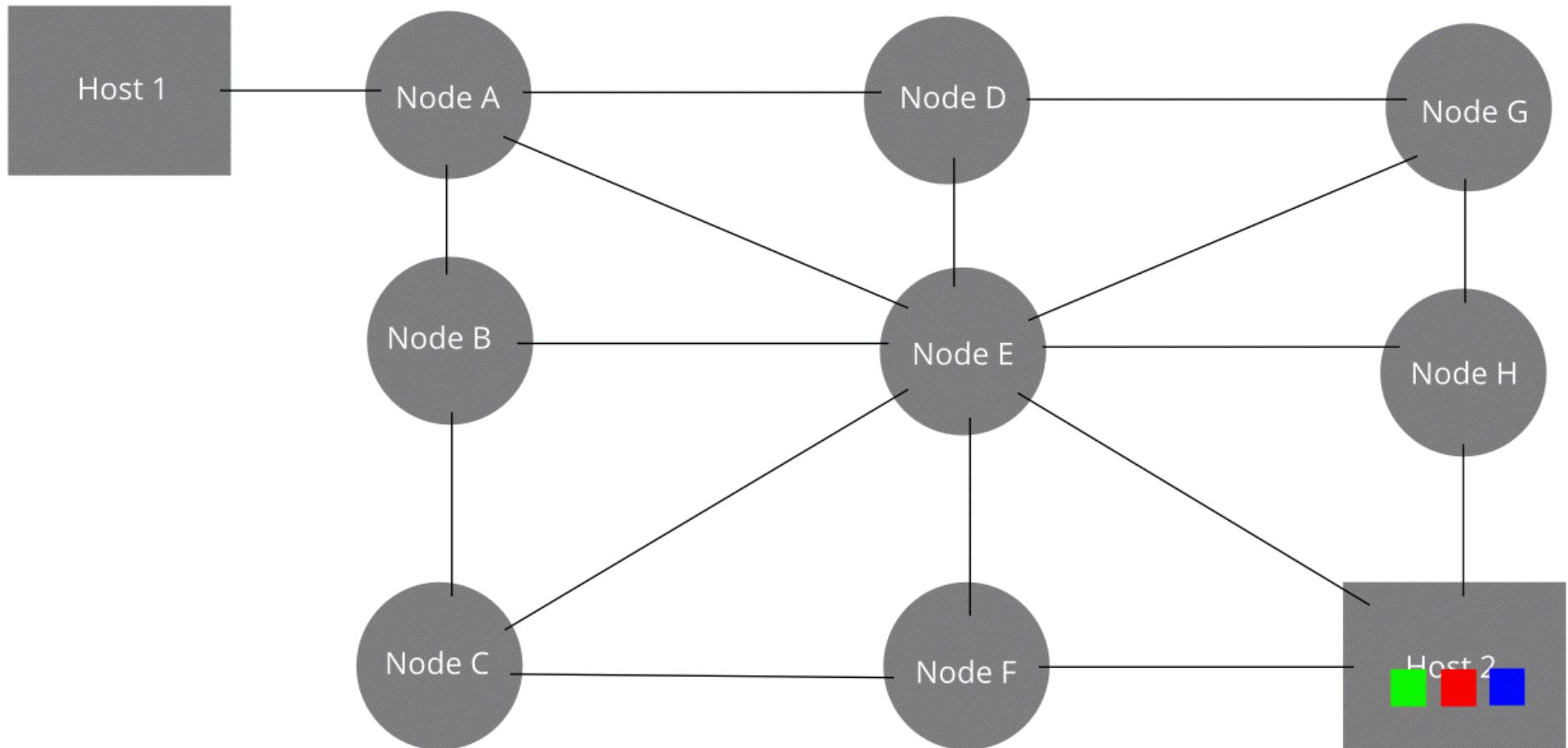
Barrier sync: wait until all packets arrive

The data packets take different routes to their destinations.

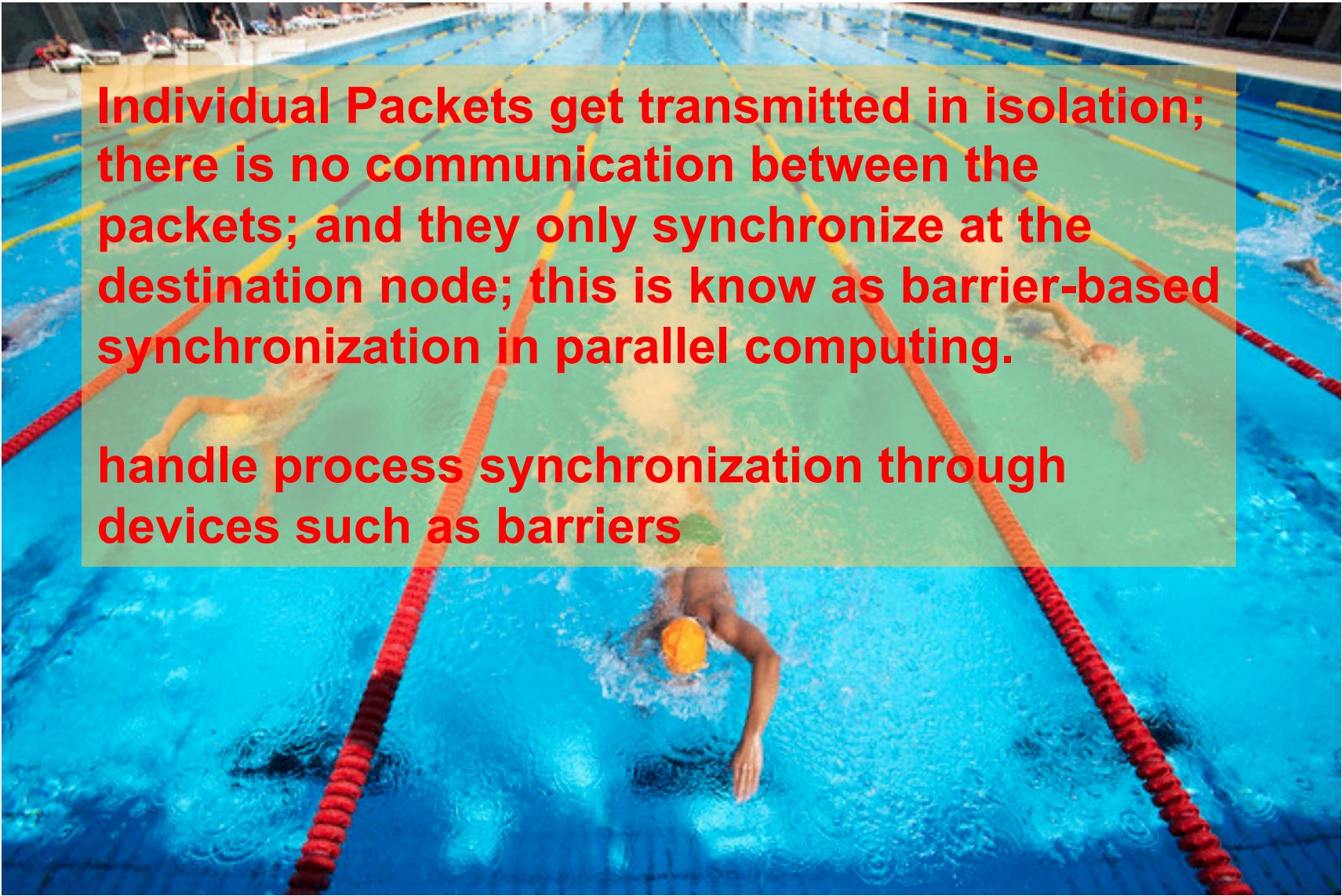


Barrier sync: reassemble payload

The received message is Green, Red, Blue



Individual Packets get transmitted in isolation



Individual Packets get transmitted in isolation; there is no communication between the packets; and they only synchronize at the destination node; this is known as barrier-based synchronization in parallel computing.

handle process synchronization through devices such as barriers

-
- **The key to success here was Divide and Conquer**
 - **Decompose a large task into smaller ones**
 - **We came up with a very nice framework for parallelizing tasks on the command line!!**
 - But it is limited
 - Granularity of task is somewhat coarse
 - No fault tolerance
 - No control over shared filespace
 - **Divide and conquer does not come for free: there are obligations in terms of communication, synchronization, and fault tolerance**

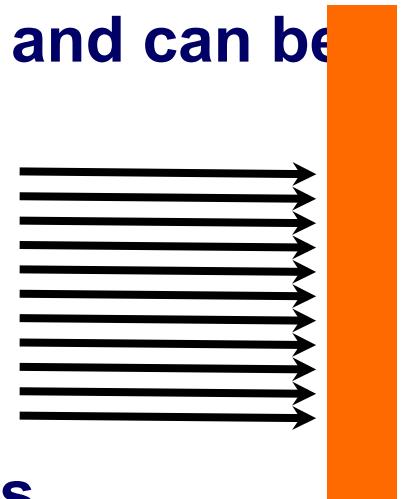
Issues to be addressed

- ▶ How to break large problem into smaller problems? Decomposition for parallel processing
- ▶ How to assign tasks to workers distributed around the cluster?
- ▶ How do the workers get the data?
- ▶ How to synchronize among the workers?
- ▶ How to communicate with works?
- ▶ How to share partial results among workers?
- ▶ How to do all these in the presence of errors and hardware failures?

Divide and conquer does not come for free: there are obligations in terms of communication, synchronization, and fault tolerance

Synchronization thru a Barrier

- A barrier for a group of threads or processes in the source code means any thread/process must stop at this point and cannot proceed until all other threads/processes reach this barrier.
- Another popular way of syncing is the barrier method;
- Explicitly handle process synchronization through devices such as barriers
- it is pretty effective, and very coarse grained and can be great in certain types of problems.
- In parallel computing, a barrier is a type of synchronization method.
- Better than MPI and shared memory solutions



Summary: Communication/Synchronization

- Those problems that can be decomposed into independent subtasks, requiring no communication/synchronization between the subtasks except a join or barrier at the end are very parallelizable (embarrassingly parallel)
 - Mutex pattern
 - Join pattern
 - Barrier pattern

Categories of Parallel Computation Tasks

- Applications are often classified according to how often their subtasks need to synchronize or communicate with each other.
- **Fine-grained parallelism**
 - An application exhibits fine-grained parallelism if its subtasks must communicate many times per second; (share memory programming)
- **Coarse-grained parallelism**
 - It exhibits coarse-grained parallelism if they do not communicate many times per second,
- **Embarrassingly parallel**
 - An application is embarrassingly parallel if it rarely or never has to communicate. Divide and conquer. Summing a list of numbers.
 - Such applications are considered the easiest to parallelize.
 - Can be realized on a shared nothing architecture (see this shortly)

Examples of embarrassingly parallel problems

- An application is embarrassingly parallel if it rarely or never has to communicate
- Applications
 - Summing a list of numbers
 - Matrix multiplication
 - Lots of machine learning algorithms
 - Tree growth step of the random forest machine learning technique.
 - Genetic algorithms and other evolutionary computation metaheuristics.
 - Serving static files on a webserver to multiple users at once.
 - Computer simulations comparing many independent scenarios, such as climate models.
 - Ensemble calculations of numerical weather prediction.
 - Discrete Fourier Transform where each harmonic is independently calculated.
 - Many many more....

Not everything is a MR

- MRs are ideal for “embarrassingly parallel” problems
 - Very little communication
 - Easily distributed
 - Linear computational flow
-
- Happy to report that most of the machine learning algorithms of practical importance are embarrassingly parallel

Tutorial Outline

- **Part 1: Introduction**
 - Welcome Survey
 - Install Spark
 - Background and motivation
- **Part 2: Spark Intro and basics**
 - fundamental Spark concepts, including Spark Core, data frames, the Spark Shell, Spark Streaming, Spark SQL and vertical libraries such as MLlib and GraphX;
- **Part 3: Machine learning in Spark**
 - will focus on hands-on algorithmic design and development with Spark developing algorithms from scratch such as linear regression, logistic regression, graph processing algorithms such as pagerank/shortest path, etc.
- **Part 4: Wrap up**
 - Spark 1.5 and beyond
 - Summary

Part 2: Spark Intro and Basics

- **Part 2: Spark Intro and basics**

- Base RDD
- Fault tolerance (and lineage)
- Transformations and Actions
- Persistance
- Animated Example
- Pair RDDs
- Word count example

-
- **Traditional distributed data processing frameworks have limited APIs such as Map/Reduce**
 - **Spark is a much more expressive API (over 80 functions)**

Spark: A developer's perspective

- At a high level, every Spark application consists of a driver program that runs the user's main function and executes various parallel operations on a cluster.
- The main abstraction Spark provides is a resilient distributed dataset (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel.
 - RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it.
 - Users may also ask Spark to persist an RDD in memory, allowing it to be reused efficiently across parallel operations.
 - Finally, RDDs automatically recover from node failures.

Resilient distributed dataset (RDD)

- An RDD is simply a distributed collection of elements (Key-Value records).
- In Spark all work is expressed as either creating new RDDs, transforming existing RDDs, or calling operations on RDDs to compute a result.

- Under the hood, Spark automatically distributes the data contained in RDDs across your cluster and parallelizes the operations you perform on them.

RDD: Collection of values: Map/Filter/Reduce

Step by Step

RDD: each row is a key, value pair

Focus on simple RDDs in this section, i.e., value only RDDs

Next section talk about RDDs of key/value pairs

Key	Value
d1	the quick brown fox
d2	the fox ate the mouse
d3	how now brown cow

```
graph LR; A[HDFS] --> B[HadoopRDD]; B --> C[FilteredRDD]; C --> D[MappedRDD]; D --> E[HDFS];
```

The diagram illustrates a data processing pipeline. It starts with a cylinder labeled "HDFS", which points to a rectangular box labeled "HadoopRDD". This is followed by a sequence of three more rectangular boxes: "FilteredRDD", "MappedRDD", and finally "HDFS". An arrow points from the last "HDFS" box downwards, indicating the output of the process.

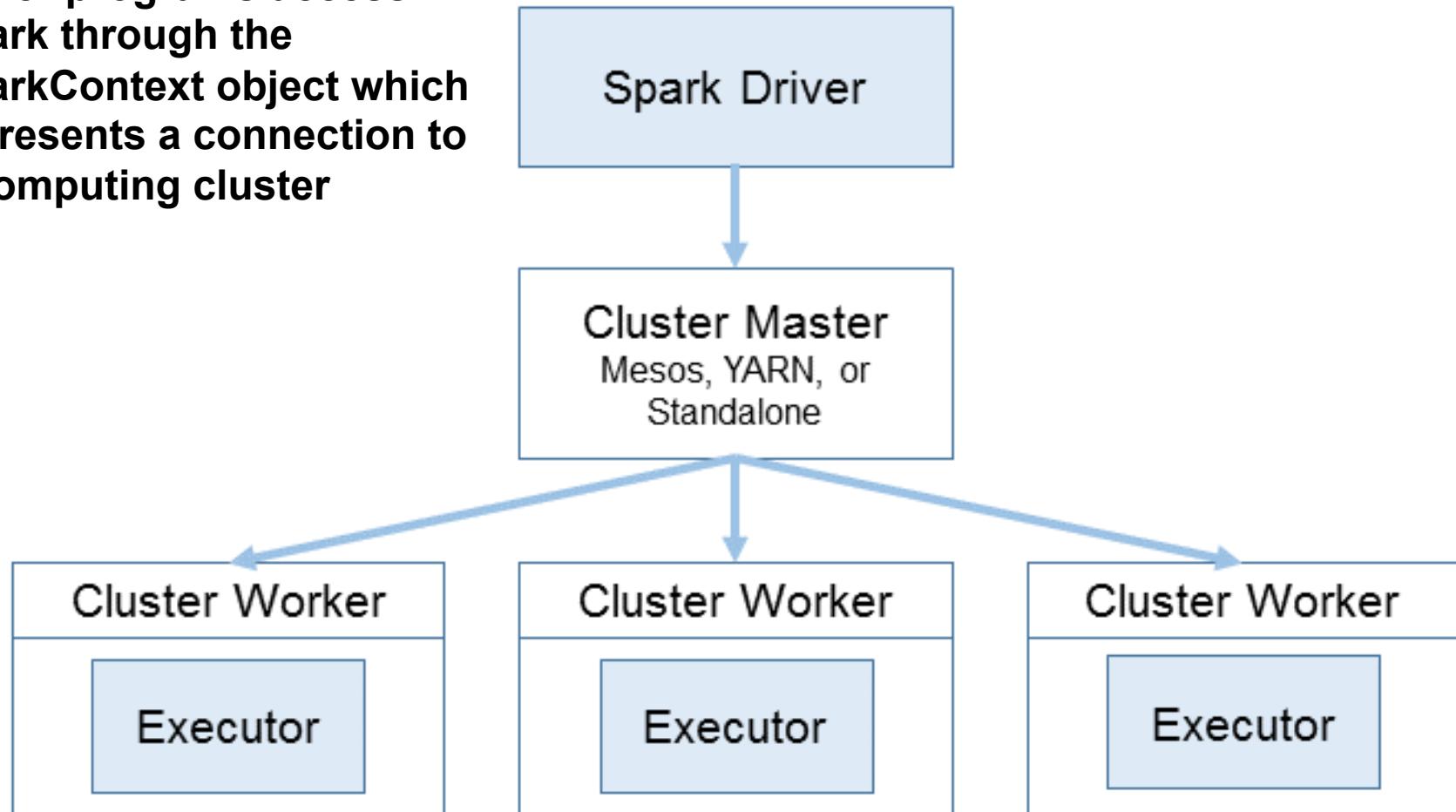
The text below the diagram states: "The RDD.saveAsTextFile() action triggers a job. Tasks are started on scheduled executors."

Spark Programming Interface

- **Spark has APIs available in**
 - Scala
 - Java
 - Python
 - R
- **A lot of Spark's API revolves around passing functions to its operators to run then on the cluster (ships codes to executors)**
- **Provides:**
 - Resilient distributed datasets (RDDs)
 - Partitioned collections with controllable caching, built in fault tolerance
 - Operations on RDDs
 - Transformations (define RDDs), actions (compute results)
 - Restricted shared variables (broadcast, accumulators)
- **Goal: make parallel programs look like local ones**

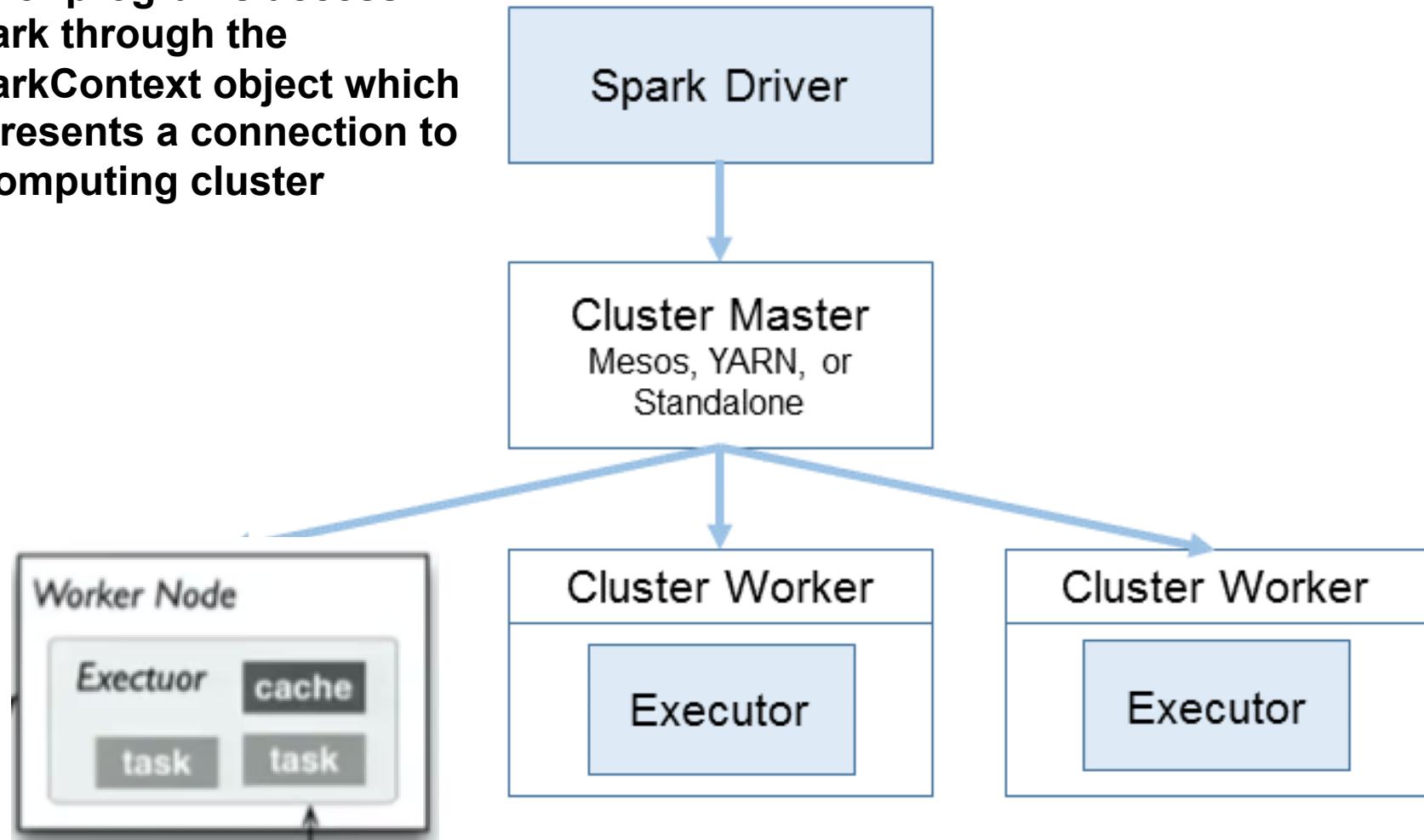
Spark Cluster

Driver programs access Spark through the `SparkContext` object which represents a connection to a computing cluster



Spark Cluster

Driver programs access Spark through the `SparkContext` object which represents a connection to a computing cluster



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

Spark Essentials: Master

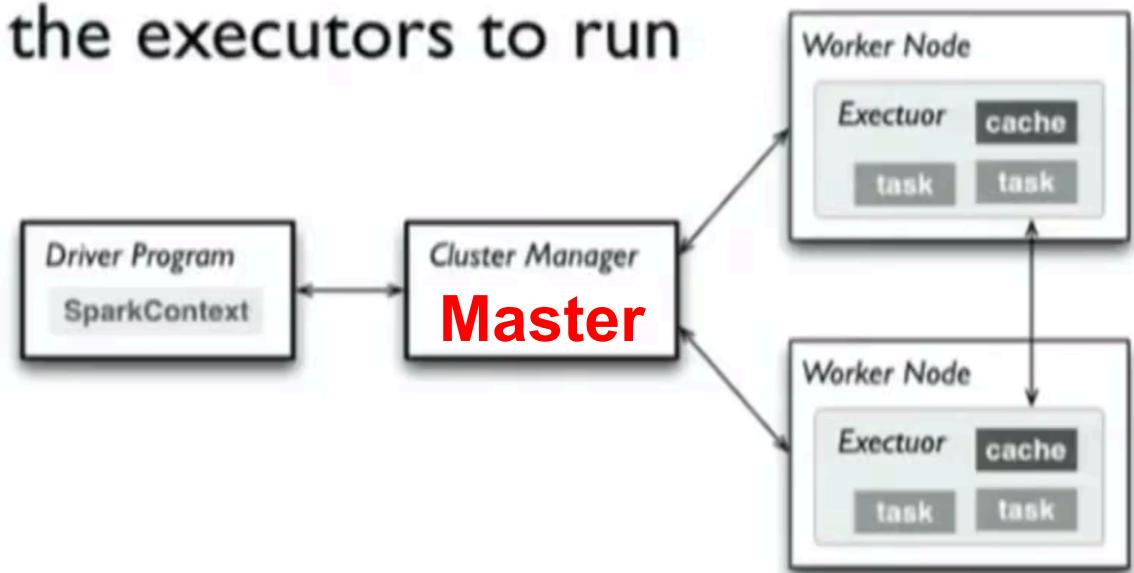
The master parameter for a SparkContext determines which cluster to use

<i>master</i>	<i>description</i>
local	run Spark locally with one worker thread (no parallelism)
local[K]	run Spark locally with K worker threads (ideally set to # cores)
spark://HOST:PORT	connect to a Spark standalone cluster; PORT depends on config (7077 by default)
mesos://HOST:PORT	connect to a Mesos cluster; PORT depends on config (5050 by default)

Spark Essentials: Master

Driver programs access Spark through the `SparkContext` object which represents a connection to a computing cluster

1. connects to a *cluster manager* which allocate resources across applications
2. acquires executors on cluster nodes – worker processes to run computations and store data
3. sends *app code* to the executors **(serializes code and data)**
4. sends *tasks* for the executors to run



Spark Essentials: RDD

Resilient **D**istributed **D**atasets (RDD) are the primary abstraction in Spark – a fault-tolerant collection of elements that can be operated on in parallel

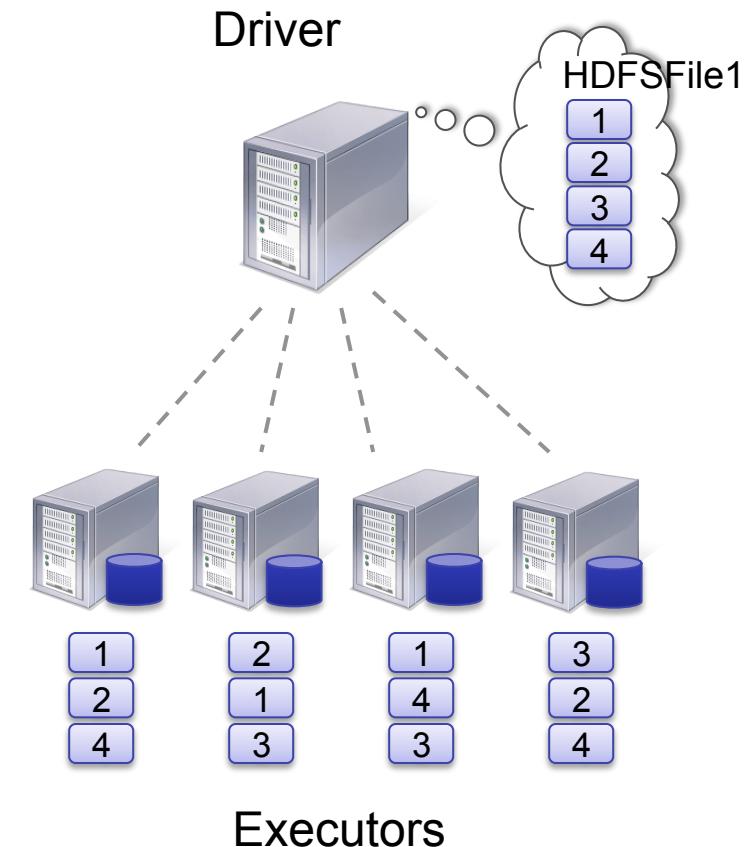
There are currently two types:

- *parallelized collections* – take an existing Scala collection and run functions on it in parallel
- *Hadoop datasets* – run functions on each record of a file in Hadoop distributed file system or any other storage system supported by Hadoop

RDD is a distributed collection of elements

- In Spark all work is expressed as either creating new RDDs, transforming existing RDDs, or calling operations on RDDs to compute a result.
- An RDD is laid out across a cluster of machines as **collection of Partitions**, each including a subset of the data
- The framework processes the objects within a partition in sequence, and processes multiple partitions in parallel.
- Data (e.g., clicks, or record linkage) is stored in a text file, with one observation on each line.
 - JSON, zipped, AVRO, Parquet

Partition and distribute data



Mapper: Create RDD and ship to cluster; then apply mapper X+1)

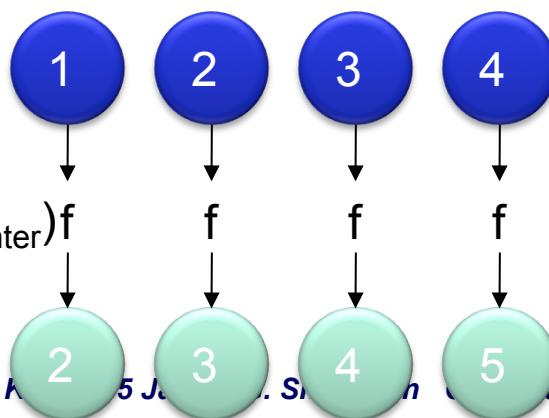
```
In [19]: #RDD mapper Distribute Map Gather back to driver  
sc.parallelize([1,2,3,4]).map(lambda x: x+1).collect()  
  
#returns [2, 3, 4, 4]  
  
Out[19]: [2, 3, 4, 5]
```

- Spark is oriented around *Key-Value* records

- plusOneData= Apply(Mapper=f(x)=x × 1, input=X)

- Map function:

$\text{Map}(\text{Key}_{\text{in}}, \text{Value}_{\text{in}}) \rightarrow \text{list}(\text{K}_{\text{inter}}, \text{V}_{\text{inter}})$ f $f(x)=x \times 1$



RDD with 4 elements

In [19]: #RDD mapper Distribute Map Gather back to driver
sc.parallelize([1,2,3,4]).map(lambda x: x+1).collect()

#returns [2, 3, 4, 4]

Out[19]: [2, 3, 4, 5]

In [19]: #RDD mapper
sc.parallelize([1,2,3,4]).map(lambda x: x+1).collect()

#returns [2, 3, 4, 4]

Out[19]: [2, 3, 4, 5]

In [33]: def powerOfTwo(x):
 return x*x

def plusOne(x):
 return x+1

def myReduce(x, y):
 return x+y

print "Reduce by lambda: %d" % sc.parallelize([1,2,3,4]).map(powerOfTwo).map(plusOne).reduce(lambda x, y: x + y)
print "Reduce by function: %d" % sc.parallelize([1,2,3,4]).map(powerOfTwo).map(plusOne).reduce(myReduce)
#.reduce()

python anonymous function vs. top level function

- Arguments to Map/Reduce operators are closures and can be python anonymous functions (Lambdas) or any top level function

RDD actions and transformations can be used for more complex computations. Let's say we want to find the line with the most words:

Scala Python

```
>>> textFile.map(lambda line: len(line.split())).reduce(lambda a, b: a if (a > b) else b)  
15
```

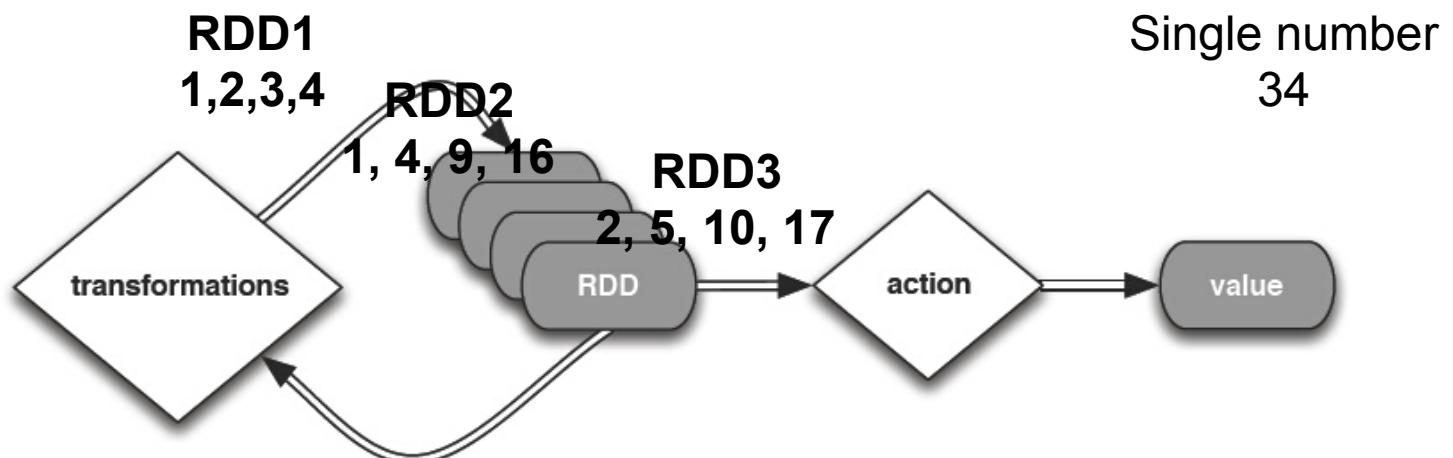
This first maps a line to an integer value, creating a new RDD. reduce is called on that RDD to find the largest line count. The arguments to map and reduce are Python **anonymous functions (lambdas)**, but we can also pass any top-level Python function we want. For example, we'll define a max function to make this code easier to understand:

Spark Essentials: RDD

Spark can create RDDs from any file stored in HDFS or other storage systems supported by Hadoop, e.g., local file system, Amazon S3, Hypertable, HBase, etc.

Spark supports text files, SequenceFiles, and any other Hadoop InputFormat, and can also take a directory or a glob (e.g. /data/201404*)

```
sc.parallelize([1,2,3,4]).map(powerOfTwo).map(plusOne).reduce(lambda x, y: x + y)
```



Example Mappers and Reducers

```
def powerOfTwo(x):
    return x*x

def plusOne(x):
    return x+1

def myReduce(x, y):
    return x+y

print "Reduce by lambda:  %d" %
sc.parallelize([1,2,3,4]).map(powerOfTwo).map(plusOne).reduce(lambda x, y: x + y)
```

```
print "Reduce by function: %d" %
sc.parallelize([1,2,3,4]).map(powerOfTwo).map(plusOne).reduce(myReduce)
#.reduce()
```

Reduce by lambda: 34
Reduce by function: 34

Create RDDs

```
./bin/pyspark
```

Spark's primary abstraction is a distributed collection of items called a Resilient Distributed Dataset (RDD). RDDs can be created from Hadoop InputFormats (such as HDFS files) or by transforming other RDDs. Let's make a new RDD from the text of the README file in the Spark source directory:

```
>>> textFile = sc.textFile("README.md")
```

RDDs have *actions*, which return values, and *transformations*, which return pointers to new RDDs. Let's start with a few actions:

```
>>> textFile.count() # Number of items in this RDD  
126  
  
>>> textFile.first() # First item in this RDD  
u'# Apache Spark'
```

Now let's use a transformation. We will use the *filter* transformation to return a new RDD with a subset of the items in the file.

```
>>> linesWithSpark = textFile.filter(lambda line: "Spark" in line)
```

We can chain together transformations and actions:

```
>>> textFile.filter(lambda line: "Spark" in line).count() # How many lines contain "Spark"?  
15
```

Self-Contained Applications in Python (4 threads)

As an example, we'll create a simple Spark application, SimpleApp.py:

```
"""SimpleApp.py"""
from pyspark import SparkContext

logFile = "YOUR_SPARK_HOME/README.md" # Should be some file on your system
sc = SparkContext("local", "Simple App")
logData = sc.textFile(logFile).cache()

numAs = logData.filter(lambda s: 'a' in s).count()
numBs = logData.filter(lambda s: 'b' in s).count()

print "Lines with a: %i, lines with b: %i" % (numAs, numBs)
```

This program just counts the number of lines containing ‘a’ and the number containing ‘b’ in a text file. Note that you’ll need to replace YOUR_SPARK_HOME with the location where Spark is installed. As with the Scala and Java examples, we use a SparkContext to create RDDs. We can pass Python functions to Spark, which are automatically serialized along with any variables that they reference. For applications that use custom classes or third-party libraries, we can also add code dependencies to spark-submit through its --py-files argument by packaging them into a .zip file (see spark-submit --help for details). SimpleApp is simple enough that we do not need to specify any code dependencies.

We can run this application using the bin/spark-submit script:

```
# Use spark-submit to run your application
$ YOUR_SPARK_HOME/bin/spark-submit \
--master local[4] \
SimpleApp.py
...
Lines with a: 46, Lines with b: 23
```

RDDs in More Detail

- An RDD is an immutable, partitioned, logical collection of records
 - Need not be materialized, but rather contains information to rebuild a dataset from stable storage or computer instructions (e.g., generate 1M random numbers)
 - **Materialize RDDs** through actions, e.g., count or reduce
- Partitioning can be based on a key in each record (using hash or range partitioning)
- Built using bulk transformations on other RDDs
- Can be cached for future reuse (or rebuilt if not cached and required again)

Two types of RDD Operations

- **TRANSFORMATIONS:** Think Map (take an RDD as input and produce an RDD); LAZY
- **ACTIONS:** E.g., Reduce.
 - take an RDD as input and
 - return a result to the driver or write it to storage
 - AND kick off a computation
- A transformed RDD gets (re)computed when an action is run on it
- An RDD can be persisted into memory or disk

Example RDD creation from an existing collection in your program

Scala:

```
scala> val data = Array(1, 2, 3, 4, 5)
data: Array[Int] = Array(1, 2, 3, 4, 5)
```

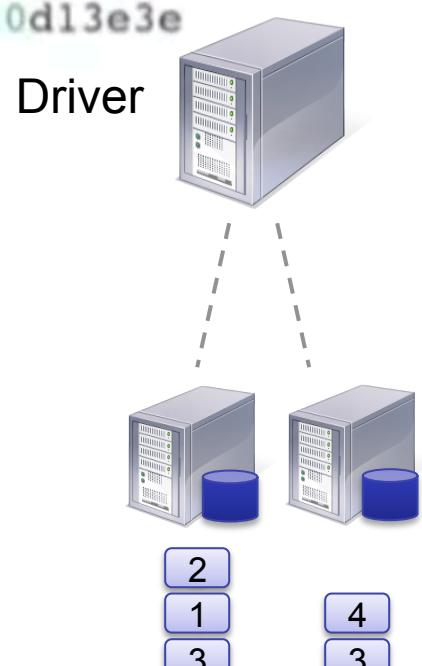
```
scala> val distData = sc.parallelize(data)
distData: spark.RDD[Int] = spark.ParallelCollection@10d13e3e
```

Python:

```
>>> data = [1, 2, 3, 4, 5]
>>> data
[1, 2, 3, 4, 5]
```

```
>>> distData = sc.parallelize(data)
>>> distData
```

```
ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:229
```



Example RDD creation from external data

Scala:

```
scala> val distFile = sc.textFile("README.md")
distFile: spark.RDD[String] = spark.HadoopRDD@1d4cee08
```

Python:

```
>>> distFile = sc.textFile("README.md")
14/04/19 23:42:40 INFO storage.MemoryStore: ensureFreeSpace(36827) called
with curMem=0, maxMem=318111744
14/04/19 23:42:40 INFO storage.MemoryStore: Block broadcast_0 stored as
values to memory (estimated size 36.0 KB, free 303.3 MB)
>>> distFile
MappedRDD[2] at textFile at NativeMethodAccessorImpl.java:-2
```

Mapper: tokenize; Apply tokenize to each input record

tokenize("coffee panda") = List("coffee", "panda")

RDD1
{"coffee panda", "happy panda",
"happiest panda party"}

rdd1.map(tokenize)

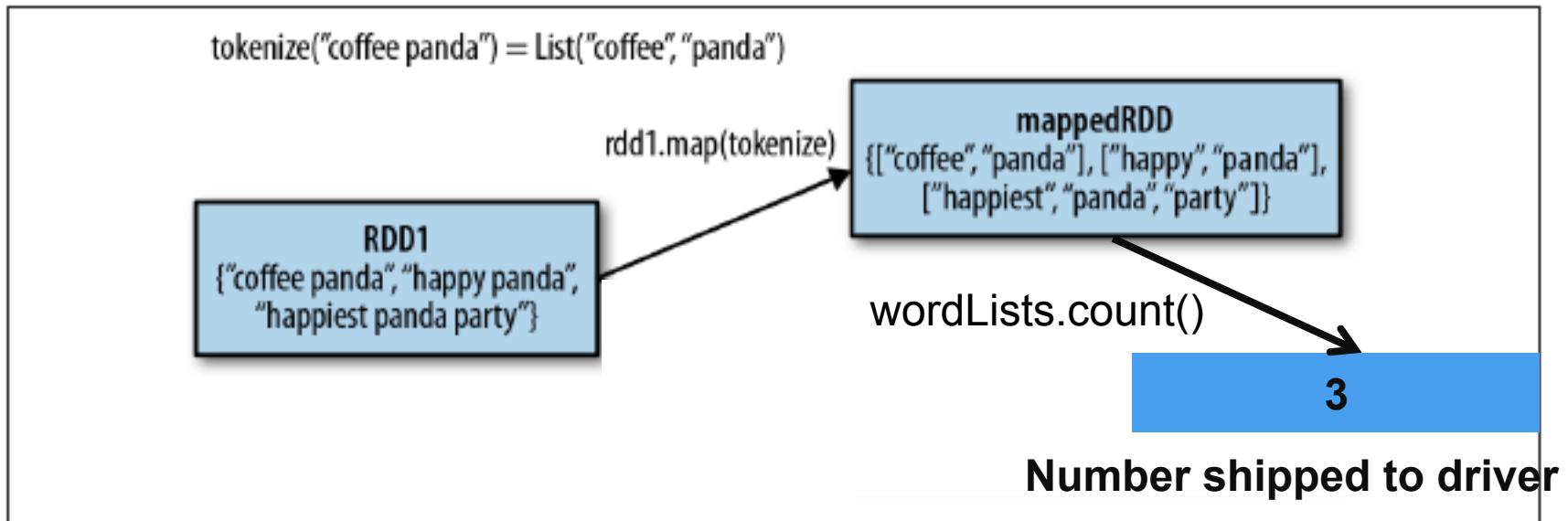
mappedRDD
[["coffee", "panda"], ["happy", "panda"],
["happiest", "panda", "party"]]

```
def tokenize(line):  
    return(line.split(" "))
```

```
rdd1= sc.parallelize(["coffee panda", "happy panda", "happiest panda party"])  
words = rdd.map(tokenize)
```

Framework takes care of passing the mapper's function to the executors (task nodes)

Built in reducer for counting (action)



```
def tokenize(line):  
    return(line.split(" ")))
```

```
rdd1= sc.parallelize(["coffee panda", "happy panda", "happiest panda party"])  
wordLists = rdd1.map(tokenize)  
wordLists.count()
```

The two most common transformations you will likely be using are `map()` and `filter()` (see Figure 3-2). The `map()` transformation takes in a function and applies it to each element in the RDD with the result of the function being the new value of each element in the resulting RDD. The `filter()` transformation takes in a function and returns an RDD that only has elements that pass the `filter()` function.

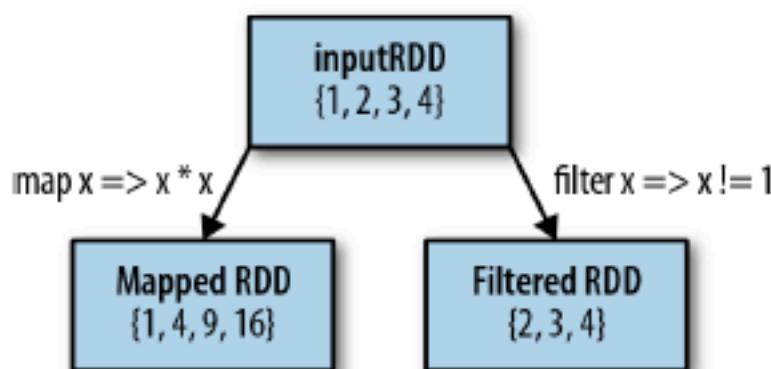


Figure 3-2. Mapped and filtered RDD from an input RDD

We can use `map()` to do any number of things, from fetching the website associated with each URL in our collection to just squaring the numbers. It is useful to note that `map()`'s return type does not have to be the same as its input type, so if we had an RDD `String` and our `map()` function were to parse the strings and return a `Double`, our input RDD type would be `RDD[String]` and the resulting RDD type would be `RDD[Double]`.

Lazy Evaluation

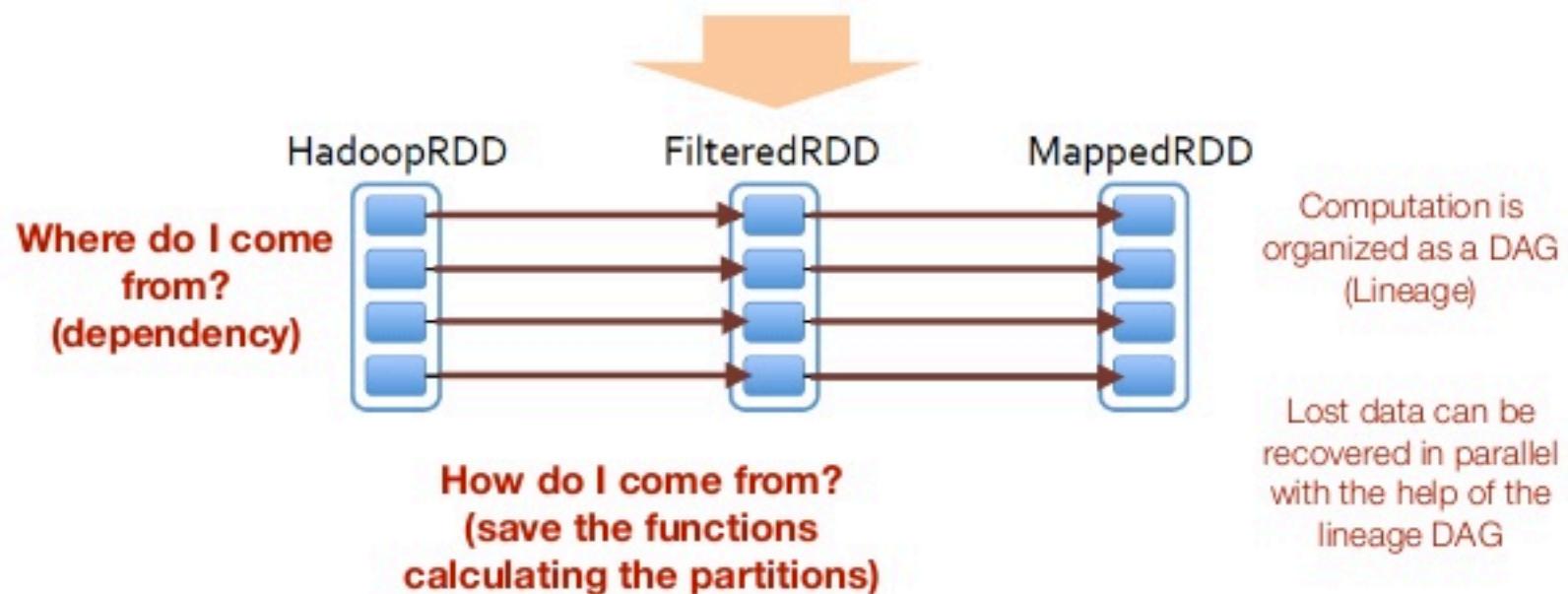
- **Transformations and actions are different because of the way Spark computes RDDs.**
- **Although you can define new RDDs any time, Spark computes them only in a lazy fashion—that is, the first time they are used in an action.**
 - `lines = sc.textFile("exampleFile")` #if not lazy would read whole file in
- **Versus**
 - `lines = sc.textFile("exampleFile").first()`

Computation is organized as a DAG (Lineage/Recipe): resiliency

From data to computation

- Lineage

```
errorMessages= sc.parallelize.filter(lambda x: 'ERROR' in x) \
    .map(lambda line: line.split(" ")[1]) #second word
```



```
errorsRDD = inputRDD.filter(lambda x: "error" in x)
warningsRDD = inputRDD.filter(lambda x: "warning" in x)
badLinesRDD = errorsRDD.union(warningsRDD)
```

RDD: Lineage Graph

`union()` is a bit different than `filter()`, in that it operates on two RDDs instead of one. Transformations can actually operate on any number of input RDDs.



A better way to accomplish the same result as in [Example 3-14](#) would be to simply filter the `inputRDD` once, looking for either *error* or *warning*.

RDD: Spark keeps track of the set of dependancies between different RDDs using Lineage Graph

Finally, as you derive new RDDs from each other using transformations, Spark keeps track of the set of dependencies between different RDDs, called the *lineage graph*. It uses this information to compute each RDD on demand and to recover lost data if part of a persistent RDD is lost. [Figure 3-1](#) shows a lineage graph for [Example 3-14](#).

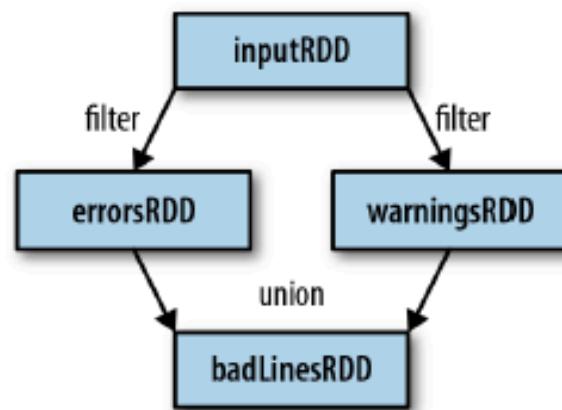


Figure 3-1. RDD lineage graph created during log analysis

counts.toDebugString() #lineage

Slide 1/2

```
In [5]: lines = sc.parallelize(["Data line 1", "Mining line 2", "data line 3", "Data line 4", "Data Mining line 5"])
counts = lines.flatMap(lambda line: line.split(" ")) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda a, b: a + b)

counts.collect()
```

```
Out[5]: [('1', 1),
          ('line', 5),
          ('Mining', 2),
          ('3', 1),
          ('2', 1),
          ('data', 1),
          ('5', 1),
          ('Data', 3),
          ('4', 1)]
```

```
In [ ]: counts.to
counts.toDF
In [2]: counts.toDebugString      and with NO stochasticity!
counts.toLocalIterator
counts.top
    + 1/m Σi(1 - yi(w'xi - b))+  
A # gradient
```

counts.toDebugString() #lineage

```
In [5]: lines = sc.parallelize(["Data line 1", "Mining line 2", "data line 3", "Data line 4", "Data Mining line 5"])
counts = lines.flatMap(lambda line: line.split(" ")) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda a, b: a + b)

counts.collect()
```

Slide 2/2

```
Out[5]: [('1', 1),
          ('line', 5),
          ('Mining', 2),
          ('3', 1),
          ('2', 1),
          ('data', 1),
          ('5', 1),
          ('Data', 3),
          ('4', 1)]
```

```
In [7]: counts.toDebugString()
```

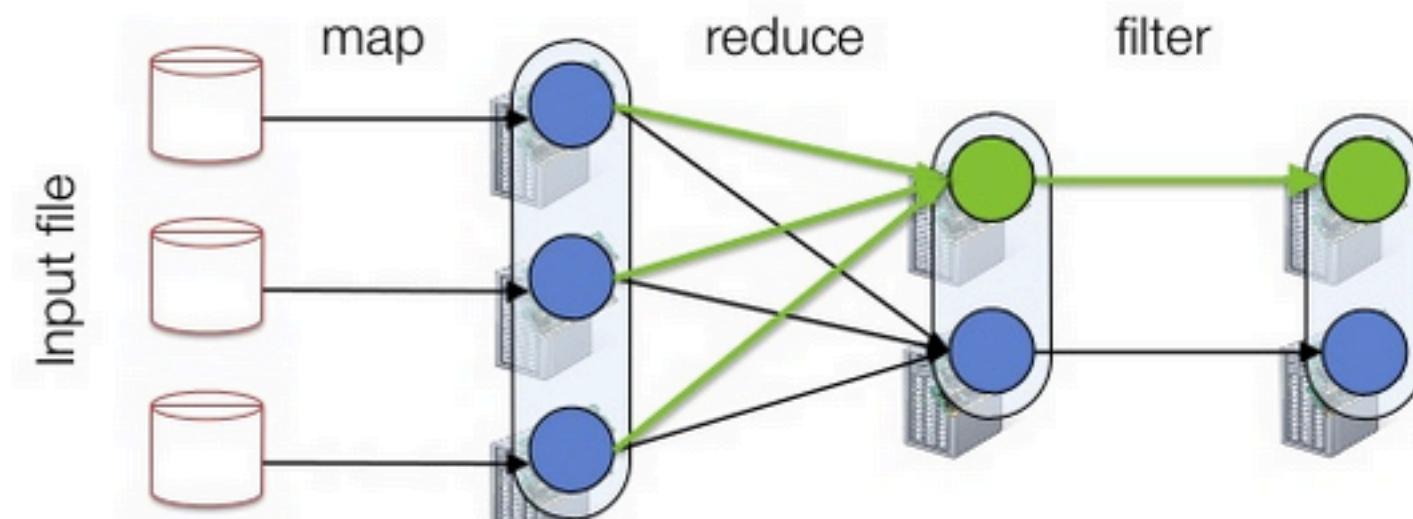
```
Out[7]: '(8) PythonRDD[7] at collect at <ipython-input-5-708c6b1f0496>:4 []\n| MapPartitionsRDD[6] at mapPartitions at PythonRDD.scala:346 []\n| ShuffledRDD[5] at partitionBy at NativeMethodAccessorImpl.java:-2 []\n+-(8) PairwiseRDD[4] at reduceByKey at <ipython-input-5-708c6b1f0496>:2 []\n| PythonRDD[3] at reduceByKey at <ipython-input-5-708c6b1f0496>:2 []\n| ParallelCollectionRDD[2] at parallelize at PythonRDD.scala:396 []'
```

```
PythonRDD[7] at collect at <ipython-input-5-708c6b1f0496>:4 []\n| MapPartitionsRDD[6] at mapPartitions at PythonRDD.scala:346 []\n| ShuffledRDD[5] at partitionBy at NativeMethodAccessorImpl.java:-2 []\n+-(8) PairwiseRDD[4] at reduceByKey at <ipython-input-5-708c6b1f0496>:2 []\n| PythonRDD[3] at reduceByKey at <ipython-input-5-708c6b1f0496>:2 []\n| ParallelCollectionRDD[2] at parallelize at PythonRDD.scala:396 []'
```

Fault Tolerance

RDDs track *lineage* info to rebuild lost data

```
file.map(lambda rec: (rec.type, 1))  
    .reduceByKey(lambda x, y: x + y)  
    .filter(lambda (type, count): count > 10)
```

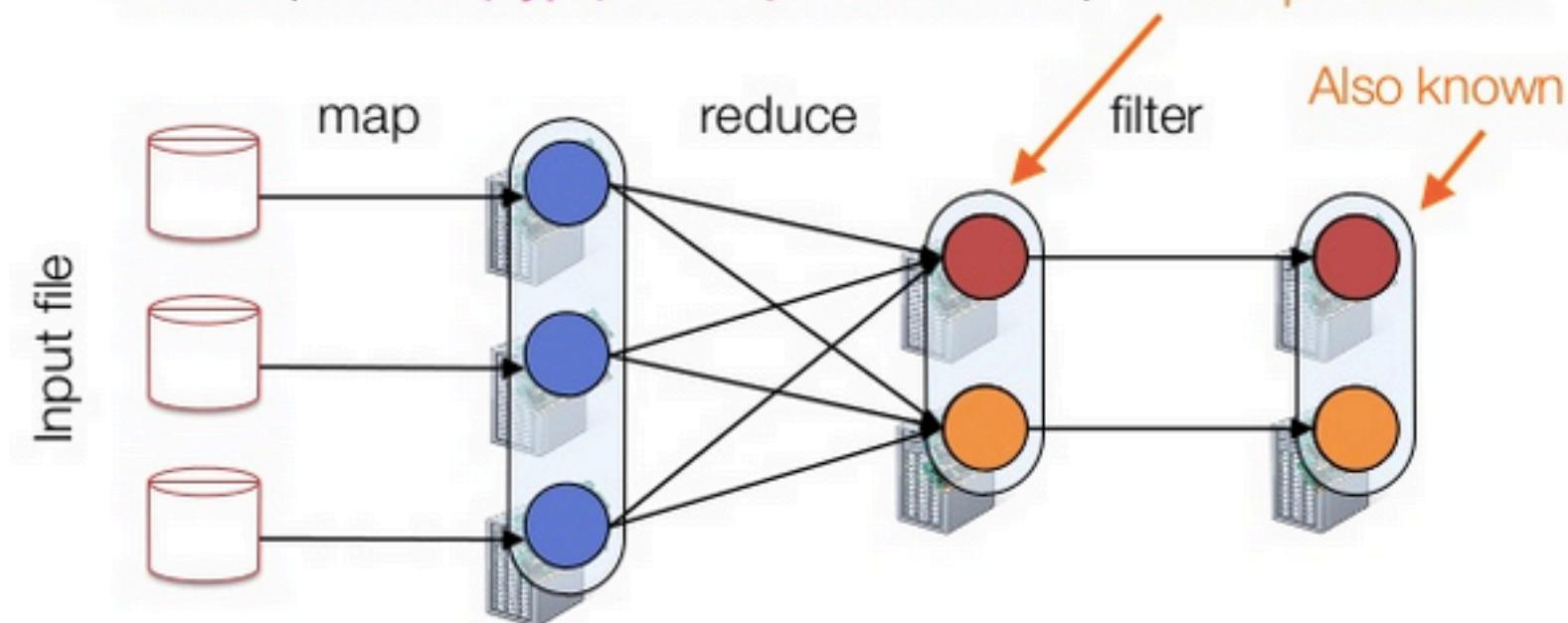


Partitioning

RDDs know their partitioning functions

```
file.map(lambda rec: (rec.type, 1))  
    .reduceByKey(lambda x, y: x + y)  
    .filter(lambda (type, count): count > 10)
```

Known to be
hash-partitioned



In summary

- That was a look at our first Spark program
- Just used very basic map and reduce (count) operations over the distributed key-value store (RDD)
- We explore more operations next and go a little deeper

Part 2: Spark Intro and Basics

- **Part 2: Spark Intro and basics**

- Base RDD
- Fault tolerance (and lineage)
- Transformations and Actions
- Persistance
- Animated Example
- Pair RDDs
- Word count example

This section

- Just used very basic map and reduce (count) operations over the distributed key-value store (RDD)
- We explore more operations next and go a little deeper:
 - Work on base RDDs and how the Spark framework works
 - Write some code

Spark Runtime

-

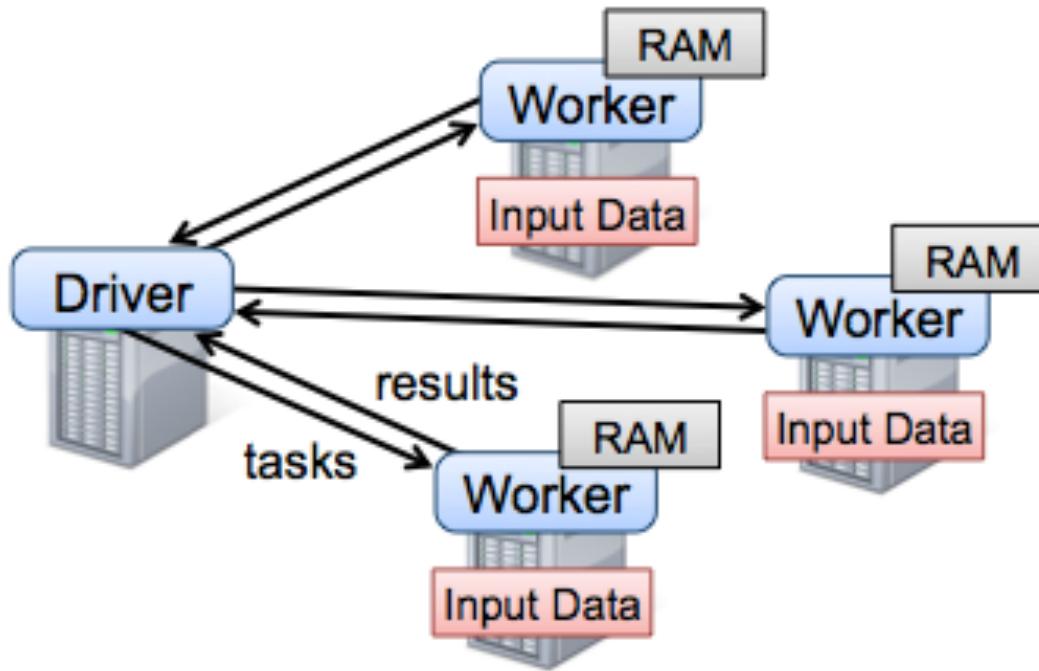


Figure 2: Spark runtime. The user's driver program launches multiple workers, which read data blocks from a distributed file system and can persist computed RDD partitions in memory.

Divide and conquer with Closures

- **Decompose problems in non-overlapping subproblems**
- **Spark's API relies heavily on passing functions in the driver program to run on the cluster. There are three recommended ways to do this:**
 - Lambda expressions, for simple functions that can be written as an expression. (Lambdas do not support multi-statement functions or statements that do not return a value.)
 - Local defs inside the function calling into Spark, for longer code.
 - Top-level functions in a module.
- **Lazy evaluation! Optimize execution graphs**

Driver vs Workers

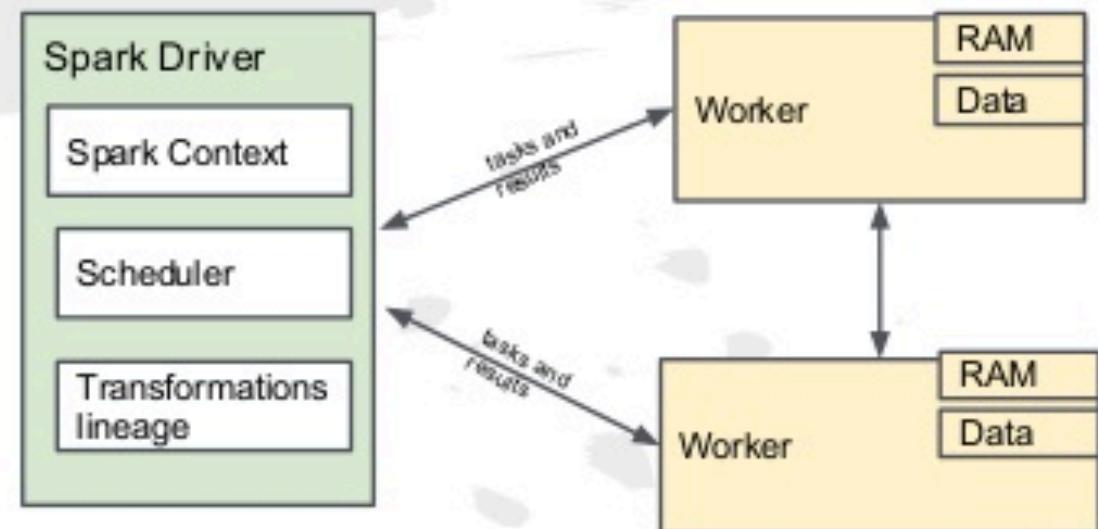
At the core of Spark...

Spark Driver

- Define & launch operations on RDDs
- Keeps track of RDD lineage allowing to recover lost partitions
- Use the RDD DAG in the Scheduler to define the stages and tasks to be executed

Workers

- Read/Transform/Write RDD partitions
- Need to communicate with other workers to share their cache and shuffle data



Closure (computer programming)

From Wikipedia, the free encyclopedia

Closure

For other uses of this term, including in mathematics and computer science, see [Closure](#).

Not to be confused with [Clojure](#).

In programming languages, **closures** (also **lexical closures** or **function closures**) are a technique for implementing lexically scoped name binding in languages with [first-class functions](#). Operationally, a closure is a data structure storing a function^[a] together with an environment:^[1] a mapping associating each free variable (variables that are used locally, but defined in an enclosing scope) with the value or storage location the function to access those captured variables later.

Example The following program defines a function `startAt` that returns a function `incrementBy`. The nested function `incrementBy` adds its argument to the value of `x`, though `x` is not local to `incrementBy`. Instead, it finds `x` in the environment where `startAt` was defined, and thus has the value 1. When `incrementBy` is called, it adds its argument to 1, and returns the result. This means that `closure1` will return 4 when invoked with 3, and `closure2` will return 8 when invoked with 3. Both `closure1` and `closure2` have the same function `incrementBy`, but they have different environments, and thus evaluate the function differently.

A closure is a data structure storing a function[a] together with an environment:

- a mapping associating each free variable of the function (variables that are used locally, but defined in an enclosing scope) with the value or storage location the name was bound to at the time the closure was created.**

```
function startAt(x)
    function incrementBy(y)
        return x + y
    return incrementBy

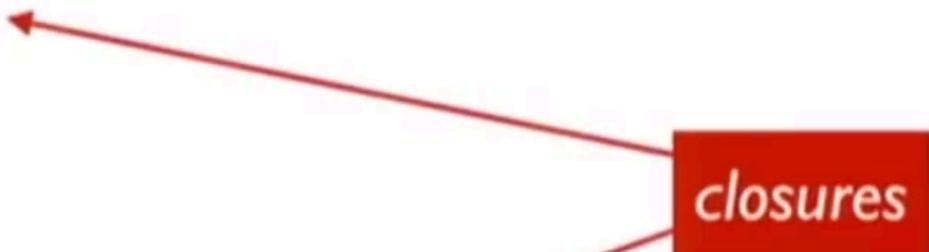
variable closure1 = startAt(1)
variable closure2 = startAt(5)
```

Note that, as `startAt` returns a function, the variables `closure1` and `closure2` are of [function type](#). Invoking `closure1(3)` will return 4, while invoking `closure2(3)` will return 8. While `closure1` and `closure2` have the same function `incrementBy`, the associated environments differ, and invoking the closures will bind the name `x` to two distinct variables in the two invocations, with different values, thus evaluating the function to different results.

Spark Essentials: Transformations

Scala:

```
val distFile = sc.textFile("README.md")
distFile.map(l => l.split(" ")).collect()
distFile.flatMap(l => l.split(" ")).collect()
```



Python:

```
distFile = sc.textFile("README.md")
distFile.map(lambda x: x.split(' ')).collect()
distFile.flatMap(lambda x: x.split(' ')).collect()
```

Spark Essentials: Transformations

Scala:

```
val distFile = sc.textFile("README.md")
distFile.map(l => l.split(" ")).collect()
distFile.flatMap(l => l.split(" ")).collect()
```

closures

Python:

```
distFile = sc.textFile("README.md")
distFile.map(lambda x: x.split(' ')).collect()
distFile.flatMap(lambda x: x.split(' ')).collect()
```

*looking at the output, how would you
compare results for map() vs. flatMap() ?*

A Hello World Example: Summary stats of a large random dataset

- Example from scratch
- Generate a random array of numbers and put them into an RDD
- Transform them by doubling each one
- Filter all numbers > 1
- RDD distributes the data (but not immediately, it waits, it is lazy!!)
- Cache the RDD in memory [still lazy]
- Count the number of numbers < 1 (should be 0)
- Count the number of numbers > 1
- Count the number of numbers > 1 (should be the same as the previous result)

SparkContext:

- A connection to a computing cluster
- Through SparkContext, driver program can access Spark.
- Automatically created in interactive shell
- You can also create your own SparkContext

```
from pyspark import SparkConf, SparkContext  
conf = SparkConf().setMaster("local").setAppName("App")  
sc = SparkContext(conf = conf)
```

File Edit View Insert Cell Kernel Help

Cell Toolbar: None

Start Spark Cluster locally and attach notebook to cluster

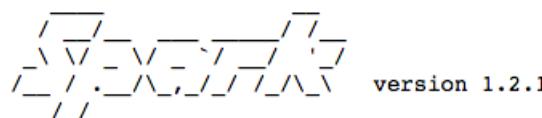
```
In [1]: import os
import sys

#set up Spark and give us a spark context sc
spark_home = os.environ['SPARK_HOME']='/Users/jshanahan/Software/spark-1.2.1-bin-hadoop2.4/' #desktop

print "["+ spark_home+"]"
if not spark_home:
    raise ValueError('SPARK_HOME environment variable is not set')

sys.path.insert(0, os.path.join(spark_home, 'python'))
sys.path.insert(0, os.path.join(spark_home, 'python/lib/py4j-0.8.2.1-src.zip'))
execfile(os.path.join(spark_home, 'python/pyspark/shell.py'))

[/Users/jshanahan/Software/spark-1.2.1-bin-hadoop2.4/]
Welcome to
```



```
Using Python version 2.7.8 (default, Aug 21 2014 15:21:46)
SparkContext available as sc.
```

See next slide for detailed breakdown

```
1 import numpy as np
2 import pylab
3
4 dataRDD = sc.parallelize(np.random.random_sample(1000))
5 data2X= dataRDD.map(lambda x: x*2)
6 dataGreaterThan1 = data2X.filter(lambda x: x > 1.0)
7 cachedRDD = dataGreaterThan1.cache()
```

```
In [29]: 1 cachedRDD.filter(lambda x: x<1).count()

Out[29]: 0
```

```
In [31]: 1 cachedRDD.filter(lambda x: x>1).count()

Out[31]: 514
```

```
In [32]: 1 cachedRDD.filter(lambda x: x>1).count()

Out[32]: 514
```

```
In [28]: 1 import numpy as np
          2 import pylab
          3
          4 dataRDD = sc.parallelize(np.random.random_sample(1000))
          5 data2X= dataRDD.map(lambda x: x*x)
          6 dataGreaterThan1 = data2X.filter(lambda x: x > 1.0)
          7 cachedRDD = dataGreaterThan1.cache()
```

```
In [29]: 1 cachedRDD.filter(lambda x: x<1).count()
```

```
Out[29]: 0
```

```
In [31]: 1 cachedRDD.filter(lambda x: x>1).count()
```

```
Out[31]: 514
```

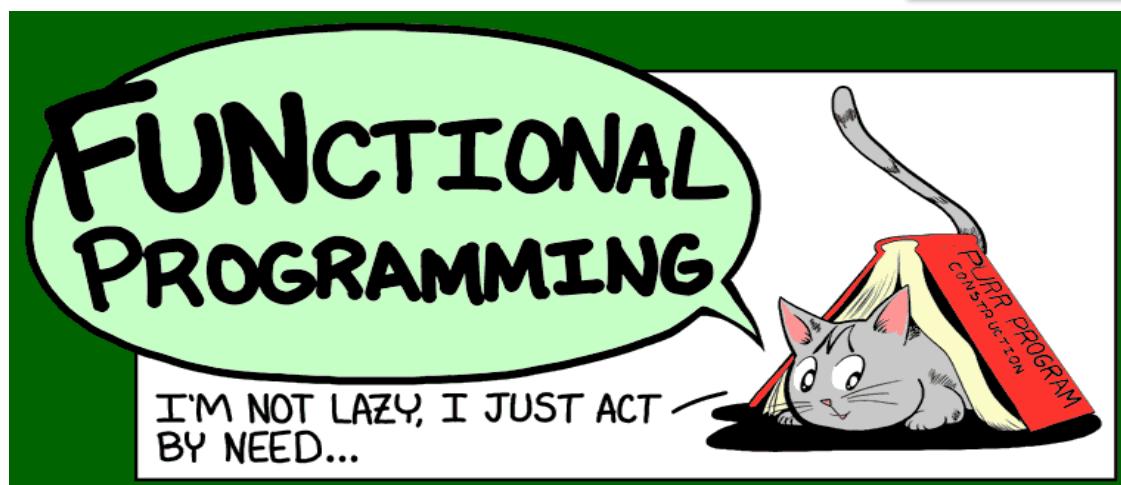
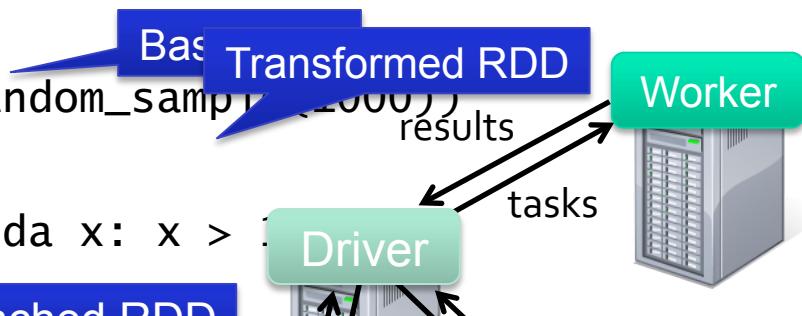
```
In [32]: 1 cachedRDD.filter(lambda x: x>1).count()
```

```
Out[32]: 514
```

Example: Random Numbers

- SC: Spark Context (connection to cluster driver)
- Load a bunch random numbers into an RDD
- Spark is Lazy...

```
dataRDD = sc.parallelize(np.random.random_sample(1000))  
data2x= dataRDD.map(lambda x: x*2)  
dataGreaterThan1 = data2x.filter(lambda x: x > 1)  
cachedRDD = dataGreaterThan1.cache()
```



Two types RDD Operations

Transformations (define a new RDD)

- map
- filter
- sample
- union
- groupByKey
- reduceByKey
- join
- cache
- ...

Nothing is materialized

Parallel operations (Actions) (return a result to driver)

- reduce
- collect
- count
- save
- lookupKey
- ...

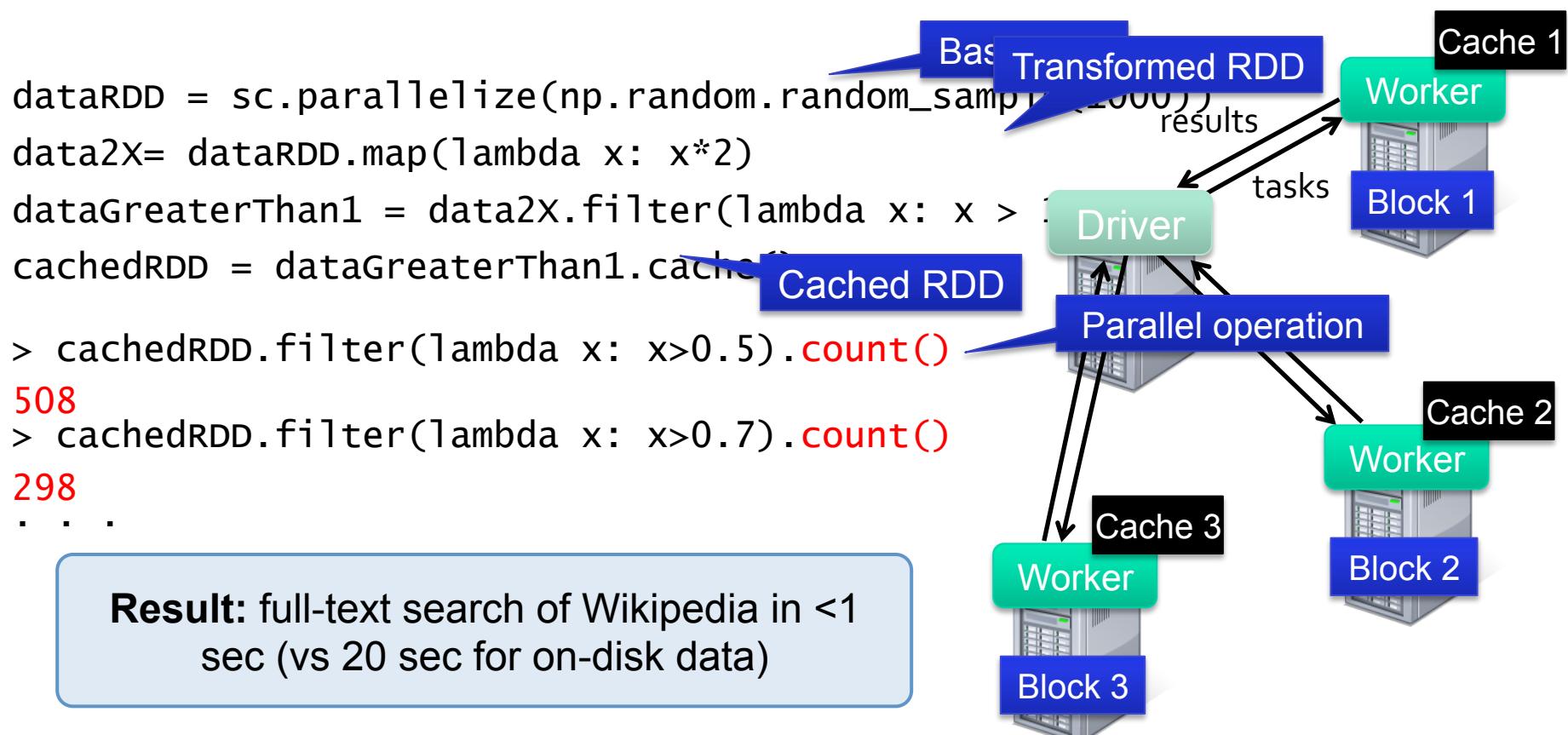
Example: Random Numbers

- Load a bunch random numbers into an RDD



Example: Random Numbers

- Load a bunch random numbers into an RDD
- Transform, filter, and the count (action)



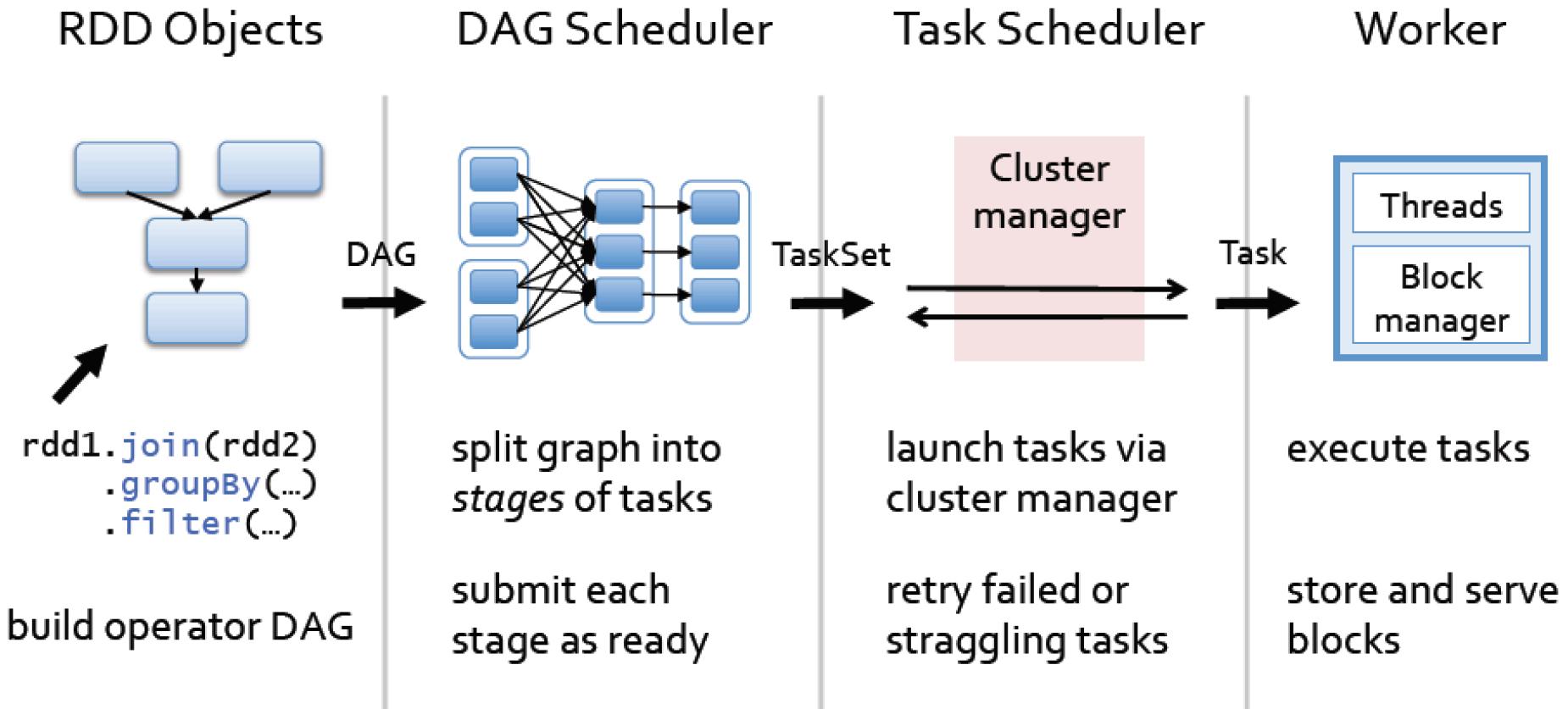
Recompute vs Cache

- Spark's RDDs are by default recomputed each time you run an action on them.
- If you would like to reuse an RDD in multiple actions, you can ask Spark to persist it using `RDD.persist()`.

Shipping code to the cluster

Optimized steps into stages

RDD → Stages → Tasks



Resilient Distributed Datasets, or RDDs

-
-

The `SparkContext` has a long list of methods, but the ones that we're going to use most often allow us to create *Resilient Distributed Datasets*, or *RDDs*. An RDD is Spark's fundamental abstraction for representing a collection of objects that can be distributed across multiple machines in a cluster. There are two ways to create an RDD in Spark:

- Using the `SparkContext` to create an RDD from an external data source, like a file in HDFS, a database table via JDBC, or from a local collection of objects that we create in the Spark shell.
- Performing a transformation on one or more existing RDDs, like filtering records, aggregating records by a common key, or joining multiple RDDs together.

RDDs are a convenient way to describe the computations that we want to perform on our data as a sequence of small, independent steps.

RDD: parallelize, textFile

Resilient Distributed Datasets

An RDD is laid out across the cluster of machines as a collection of *partitions*, each including a subset of the data. Partitions define the unit of parallelism in Spark. The framework processes the objects within a partition in sequence, and processes multiple partitions in parallel. One of the simplest ways to create an RDD is to use the `parallelize` method on `SparkContext` with a local collection of objects:

```
val rdd = sc.parallelize(Array(1, 2, 2, 4), 4) Number of partitions
...
rdd: org.apache.spark.rdd.RDD[Int] = ...
```

The first argument is the collection of objects to parallelize. The second is the number of partitions. When the time comes to compute the objects within a partition, Spark fetches a subset of the collection from the driver process.

To create an RDD from a text file or directory of text files residing in a distributed file system like HDFS, we can pass the name of the file or directory to the `textFile` method:

```
val rdd2 = sc.textFile("hdfs://some/path.txt") Create and RDD from a
file or from a directory
...
rdd2: org.apache.spark.rdd.RDD[String] = ...
```

- **Transformations**

Spark Essentials: Transformations

<i>transformation</i>	<i>description</i>
map(func)	return a new distributed dataset formed by passing each element of the source through a function <i>func</i>
filter(func)	return a new dataset formed by selecting those elements of the source on which <i>func</i> returns true
flatMap(func)	similar to map, but each input item can be mapped to 0 or more output items (so <i>func</i> should return a Seq rather than a single item)
sample(withReplacement, fraction, seed)	sample a fraction <i>fraction</i> of the data, with or without replacement, using a given random number generator <i>seed</i>
union(otherDataset)	return a new dataset that contains the union of the elements in the source dataset and the argument
distinct([numTasks])	return a new dataset that contains the distinct elements of the source dataset

Spark Essentials: Transformations

transformation	description
groupByKey([numTasks])	when called on a dataset of (K, V) pairs, returns a dataset of (K, Seq[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
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, as specified in the boolean ascending argument
join(otherDataset, [numTasks])	when called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs 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, Seq[V], Seq[W]) tuples – also called groupWith
cartesian(otherDataset)	when called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements)

Transformations vs Actions

Transformation: RDD → RDD

Once created, RDDs offer two types of operations: *transformations* and *actions*. *Transformations* construct a new RDD from a previous one. For example, one common transformation is filtering data that matches a predicate. In our text file example, we can use this to create a new RDD holding just the strings that contain the word *Python*, as shown in Example 3-2.

Example 3-2. Calling the filter() transformation

```
>>> pythonLines = lines.filter(lambda line: "Python" in line)
```

Actions: RDD → result is given to the driver or saved to external storage

Actions, on the other hand, compute a result based on an RDD, and either return it to the driver program or save it to an external storage system (e.g., HDFS). One example of an action we called earlier is `first()`, which returns the first element in an RDD and is demonstrated in Example 3-3.

Example 3-3. Calling the first() action

```
>>> pythonLines.first()  
u'## Interactive Python Shell'
```

Spark Essentials: Actions

action	description
reduce(func)	aggregate the elements of the dataset using a function <i>func</i> (which takes two arguments and returns one), and should also be commutative and associative so that it can be computed correctly in parallel
collect()	return all the elements of the dataset as an array at the driver program – usually useful after a filter or other operation that returns a sufficiently small subset of the data
count()	return the number of elements in the dataset
first()	return the first element of the dataset – similar to <i>take(1)</i>
take(n)	return an array with the first <i>n</i> elements of the dataset – currently not executed in parallel, instead the driver program computes all the elements
takeSample(withReplacement, fraction, seed)	return an array with a random sample of <i>num</i> elements of the dataset, with or without replacement, using the given random number generator seed

Spark Essentials: Actions

action	description
saveAsTextFile(path)	write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call <code>toString</code> on each element to convert it to a line of text in the file
saveAsSequenceFile(path)	write the elements of the dataset as a Hadoop SequenceFile in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. Only available on RDDs of key-value pairs that either implement Hadoop's <code>Writable</code> interface or are implicitly convertible to <code>Writable</code> (Spark includes conversions for basic types like <code>Int</code> , <code>Double</code> , <code>String</code> , etc).
countByKey()	only available on RDDs of type <code>(K, V)</code> . Returns a 'Map' of <code>(K, Int)</code> pairs with the count of each key
foreach(func)	run a function <code>func</code> on each element of the dataset – usually done for side effects such as updating an accumulator variable or interacting with external storage systems

Actions

Scala:

```
val f = sc.textFile("README.md")
val words = f.flatMap(l => l.split(" ")).map(word => (word, 1))
words.reduceByKey(_ + _).collect.foreach(println)
```

Python:

```
from operator import add
f = sc.textFile("README.md")
words = f.flatMap(lambda x: x.split(' ')).map(lambda x: (x, 1))
words.reduceByKey(add).collect()
```

Persistency

<i>transformation</i>	<i>description</i>
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.
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 the 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.

<i>transformation</i>	<i>description</i>
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.

RDD Operations (over 80 ops)

Transformations (define a new RDD)

map
filter
sample
union
PAIR RDDs (below)
groupByKey
reduceByKey
join
cache

...

Parallel operations (Actions) (return a result to driver)

reduce
collect
count
save
lookupKey
...

Nothing is
materialized

Other RDD Operations

Transformations (define a new RDD)	map filter sample <i>groupByKey</i> <i>reduceByKey</i> cogroup	flatMap union join cross mapValues ...
Actions (output a result)	collect reduce take fold	count saveAsTextFile saveAsHadoopFile ...

Map versus flatmap

Example 3-26. Python squaring the values in an RDD

```
nums = sc.parallelize([1, 2, 3, 4])
squared = nums.map(lambda x: x * x).collect()
for num in squared:
    print "%i " % (num)
```

Sometimes we want to produce multiple output elements for each input element. The operation to do this is called `flatMap()`. As with `map()`, the function we provide to `flatMap()` is called individually for each element in our input RDD. Instead of returning a single element, we return an iterator with our return values. Rather than producing an RDD of iterators, we get back an RDD that consists of the elements from all of the iterators. A simple usage of `flatMap()` is splitting up an input string into words, as shown in Examples 3-29 through 3-31.

Input produces multiple elements; want to get all into the stream

Example 3-29. `flatMap()` in Python, splitting lines into words

```
lines = sc.parallelize(["hello world", "hi"])
words = lines.flatMap(lambda line: line.split(" "))
words.first() # returns "hello"
```

Mapper: tokenize; Apply tokenize to each input record

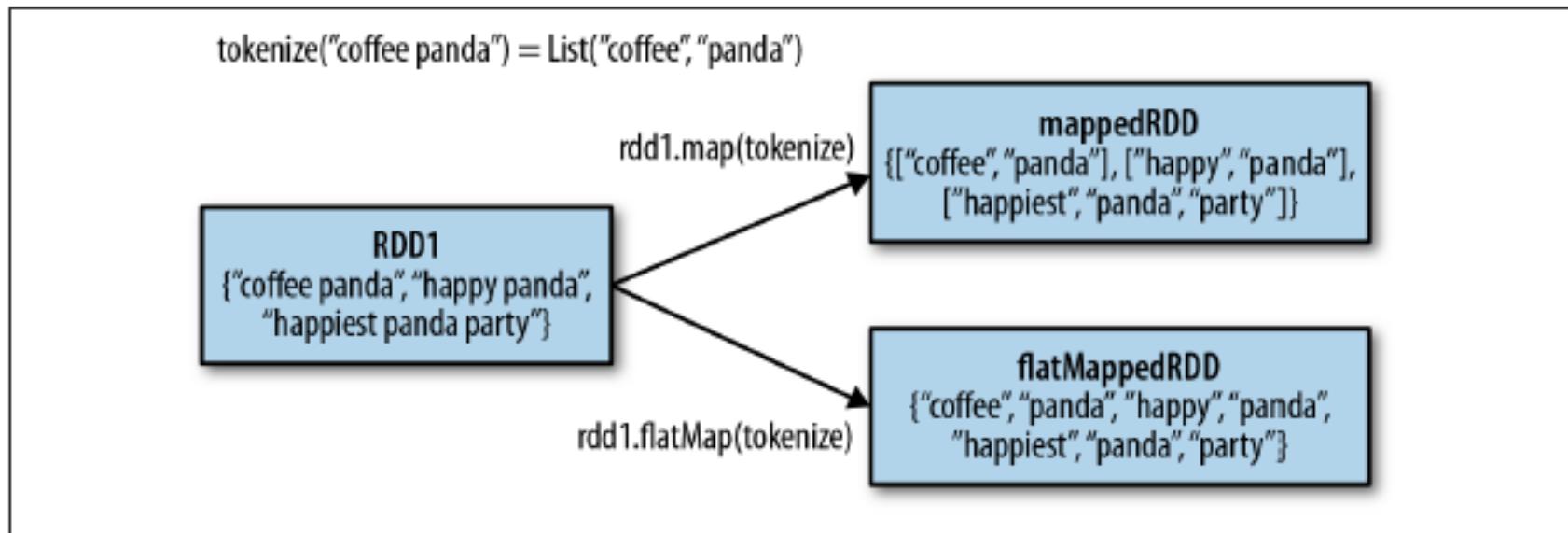


Figure 3-3. Difference between `flatMap()` and `map()` on an RDD

Set operations

RDD1
{coffee, coffee, panda,
monkey, tea}

RDD2
{coffee, money, kitty}

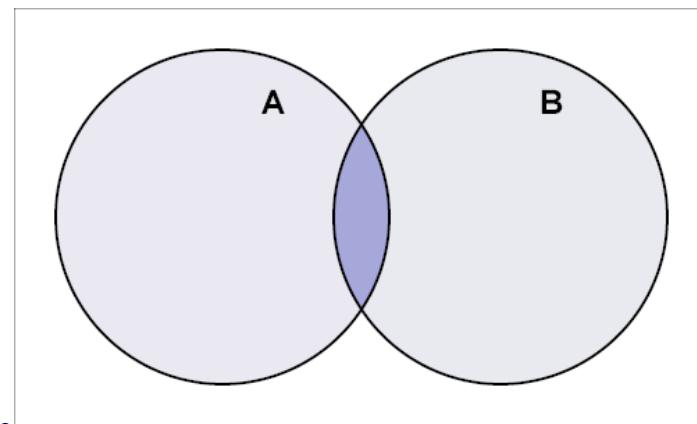
RDD1.distinct()
{coffee, panda,
monkey, tea}

RDD1.union(RDD2)
{coffee, coffee, coffee,
panda, monkey,
monkey, tea, kitty}

RDD1.intersection(RDD2)
{coffee, monkey}

RDD1.subtract(RDD2)
{panda, tea}

- Distinct
- Union
- Intersection
- Subtract



Set operations

RDD1
{coffee, coffee, panda,
monkey, tea}

RDD2
{coffee, money, kitty}

RDD1.distinct()
{coffee, panda,
monkey, tea}

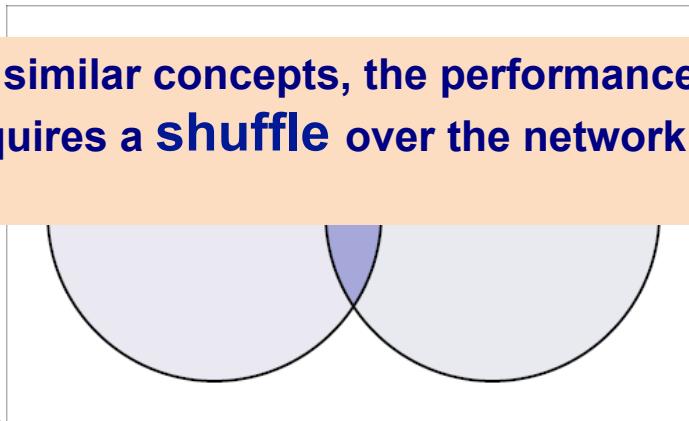
RDD1.union(RDD2)
{coffee, coffee, coffee,
panda, monkey,
monkey, tea, kitty}

RDD1.intersection(RDD2)
{coffee, monkey}

RDD1.subtract(RDD2)
{panda, tea}

While intersection() and union() are two similar concepts, the performance of intersection() is much worse since it requires a **Shuffle** over the network to identify common elements.

- **Intersection**
- **Subtract**



Cartesian Product, intersection, etc.

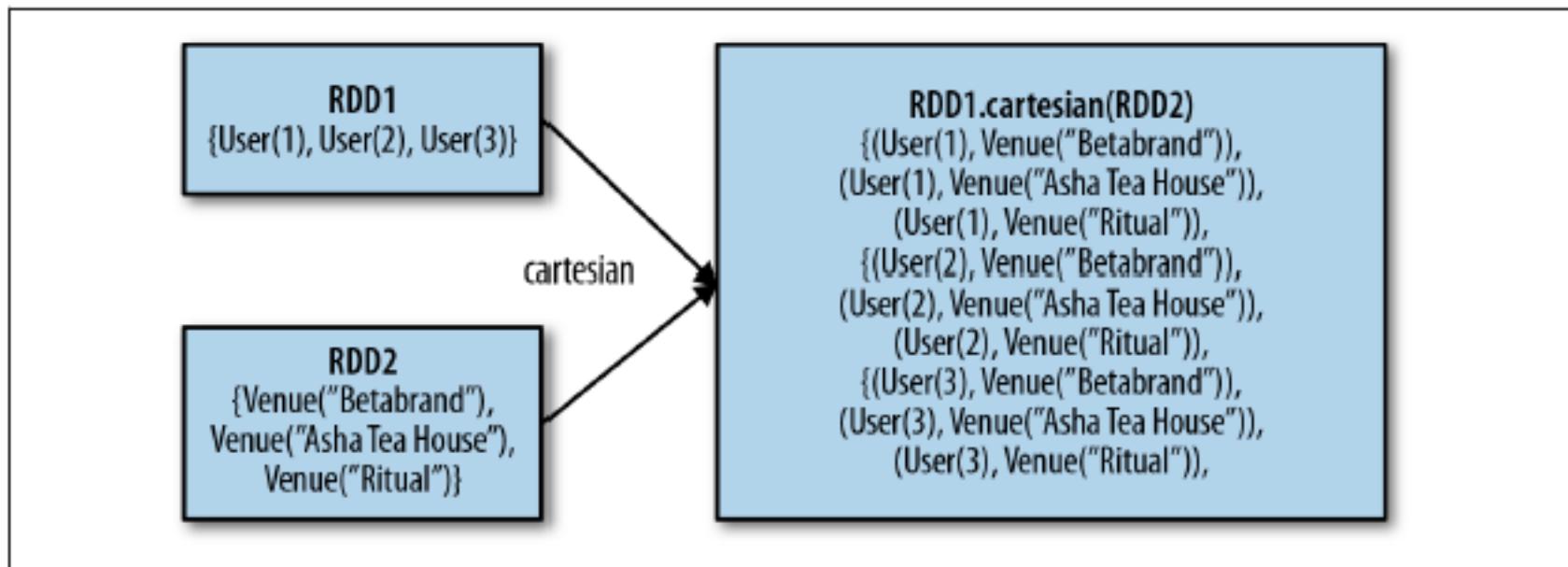


Figure 3-5. Cartesian product between two RDDs

Be warned, however, that the Cartesian product is very expensive for large RDDs.

Table 3-2. Basic RDD transformations on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
<code>map()</code>	Apply a function to each element in the RDD and return an RDD of the result.	<code>rdd.map(x => x + 1)</code>	{2, 3, 4, 4}
<code>flatMap()</code>	Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned. Often used to extract words.	<code>rdd.flatMap(x => x.to(3))</code>	{1, 2, 3, 2, 3, 3, 3}
<code>filter()</code>	Return an RDD consisting of only elements that pass the condition passed to <code>filter()</code> .	<code>rdd.filter(x => x != 1)</code>	{2, 3, 3}
<code>distinct()</code>	Remove duplicates.	<code>rdd.distinct()</code>	{1, 2, 3}
<code>sample(withReplacement, fraction, [seed])</code>	Sample an RDD, with or without replacement.	<code>rdd.sample(false, 0.5)</code>	Nondeterministic

Table 3-3. Two-RDD transformations on RDDs containing {1, 2, 3} and {3, 4, 5}

Function name	Purpose	Example	Result
union()	Produce an RDD containing elements from both RDDs.	rdd.union(other)	{1, 2, 3, 3, 4, 5}
intersection()	RDD containing only elements found in both RDDs.	rdd.intersection(other)	{3}
subtract()	Remove the contents of one RDD (e.g., remove training data).	rdd.subtract(other)	{1, 2}
cartesian()	Cartesian product with the other RDD.	rdd.cartesian(other)	{(1, 3), (1, 4), ... (3,5)}

```
errorsRDD = inputRDD.filter(lambda x: "error" in x)
warningsRDD = inputRDD.filter(lambda x: "warning" in x)
badLinesRDD = errorsRDD.union(warningsRDD)
```

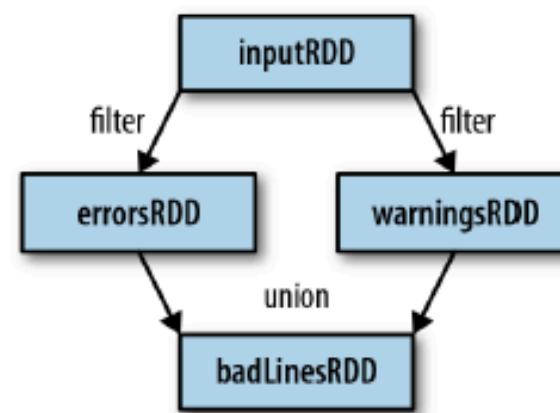
union()

`union()` is a bit different than `filter()`, in that it operates on two RDDs instead of one. Transformations can actually operate on any number of input RDDs.



A better way to accomplish the same result as in [Example 3-14](#) would be to simply filter the `inputRDD` once, looking for either *error* or *warning*.

Finally, as you derive new RDDs from each other using transformations, Spark keeps track of the set of dependencies between different RDDs, called the *lineage graph*. It uses this information to compute each RDD on demand and to recover lost data if part of a persistent RDD is lost. [Figure 3-1](#) shows a lineage graph for [Example 3-14](#).



Large Scale [Figure 3-1](#). RDD lineage graph created during log analysis

Actions: Collect can do a lot of damage

```
errorsRDD = inputRDD.filter(lambda x: "error" in x)
warningsRDD = inputRDD.filter(lambda x: "warning" in x)
badLinesRDD = errorsRDD.union(warningsRDD)
```

Example 3-15. Python error count using actions

```
print "Input had " + badLinesRDD.count() + " concerning lines"
print "Here are 10 examples:"
for line in badLinesRDD.take(10):
    print line
```

Example 3-16. Scala error count using actions

```
println("Input had " + badLinesRDD.count() + " concerning lines")
println("Here are 10 examples:")
badLinesRDD.take(10).foreach(println)
```

Example 3-17. Java error count using actions

```
System.out.println("Input had " + badLinesRDD.cou
System.out.println("Here are 10 examples:")
for (String line: badLinesRDD.take(10)) {
    System.out.println(line);
}
```

In most cases RDDs can't just be collect()ed to the driver because they are **too large**.

Persisting results

```
print "Input had " + badLinesRDD.count() + " concerning lines"
print "Here are 10 examples:"
for line in badLinesRDD.take(10):
    print line
```

In most cases RDDs can **NOT** just be collect()ed to the driver because they are too large. In these cases, it's common to write data out to a distributed storage system such as HDFS or Amazon S3.

- You can save the contents of an RDD using the `saveAsTextFile()` action, `saveAsSequenceFile()`, or any of a number of actions for various built-in formats. We will cover the different options for exporting data in Chapter 5.

`cache()`

It is important to note that each time we call a new action, the entire RDD must be computed “from scratch.” To avoid this inefficiency, users can persist intermediate results, as we will cover in “Persistence (Caching)” on page 44.

count() action

- The act of creating a RDD does not cause any distributed computation to take place on the cluster.
- Rather, RDDs define logical datasets that are intermediate steps in a computation.
- Distributed computation occurs upon invoking an action on an RDD. For example, the count action returns the number of objects in an RDD.

```
rdd.count()  
14/09/10 17:36:09 INFO SparkContext: Starting job: count at <console>:15  
...  
14/09/10 17:36:09 INFO SparkContext: Job finished: count at <console>:15,  
took 0.18273803 s  
res0: Long = 4
```

Get data from Cluster to client (first, collect, take)

- RDDs have a number of methods that allow us to read data from the cluster into the Scala REPL on our client machine. Perhaps the simplest of these is `first`, which returns the first element of the RDD into the client:

```
rawblocks.first
...
res: String = "id_1","id_2","cmp_fname_c1","cmp_fname_c2",...

val head = rawblocks.take(10)
...
head: Array[String] = Array("id_1","id_2","cmp_fname_c1",...

head.length
...
res: Int = 10
```

collect() → array in local memory

- The collect action returns an Array with all the objects from the RDD.
- This Array resides in local memory, not on the cluster.
- Be careful: don't collect too much data; this will cause your driver to crash

```
rdd.collect()
14/09/29 00:58:09 INFO SparkContext: Starting job: collect at <console>:17
...
14/09/29 00:58:09 INFO SparkContext: Job finished: collect at <console>:17,
took 0.531876715 s
res2: Array[(Int, Int)] = Array((4,1), (1,1), (2,2))
```

saveAsTextFile()

- Actions need not only return results to the local process. The `saveAsTextFile` action saves the contents of an RDD to persistent storage like HDFS.

```
rdd.saveAsTextFile("hdfs://user/ds/mynumbers")
14/09/29 00:38:47 INFO SparkContext: Starting job: saveAsTextFile at <console>:15
...
14/09/29 00:38:49 INFO SparkContext: Job finished: saveAsTextFile at <console>:15,
took 1.818305149 s
```

- The action creates a directory and writes out each partition as a file within it. From the command line outside of the Spark shell:

```
hadoop fs -ls /user/ds/mynumbers
```

-rw-r--r--	3	ds	supergroup	0	2014-09-29 00:38	myfile.txt/_SUCCESS
-rw-r--r--	3	ds	supergroup	4	2014-09-29 00:38	myfile.txt/part-00000
-rw-r--r--	3	ds	supergroup	4	2014-09-29 00:38	myfile.txt/part-00001

Foreach(): Print-friendly

- To make it easier to read the contents of an array, we can use the `foreach` method in conjunction with `println` (`print` in Python) to print each value in the array out on its own line:

- **Summary**

- **Spark**

- Transformations
- Actions
- On Base RDDs
- Starting to get excited about spark?

Part 2: Spark Intro and Basics

- **Part 2: Spark Intro and basics**

- Base RDD
- Fault tolerance (and lineage)
- Transformations and Actions
- Persistance
- Animated Example
- Pair RDDs
- Word count example

Example: filter Error lines from logfile

HDFS
file

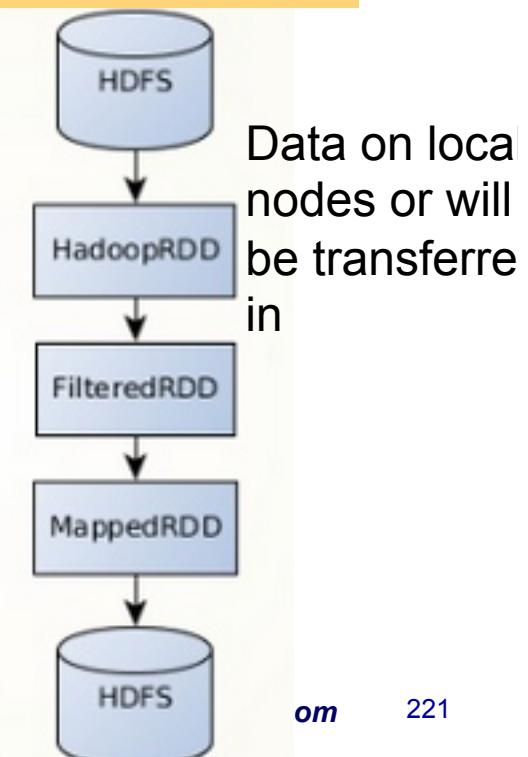
INFO not much going on here
ERROR 30330 task aborted with no status
ERROR 44404 task aborted with no status

pySpark

```
sc.textFile("hdfs://<some input path or any local file>") \
    .filter(lambda x: 'ERROR' in x)      # Error anywhere in the line
    .map(lambda line: line.split(" ")[1])  #second word and the original line
    .saveAsTextFile("hdfs://<output> or any local directory")
```

Scala

```
sc.textFile("hdfs://<input>")
    .filter(_.startsWith("ERROR"))
    .map(_.split(" ")(1))
    .saveAsTextFile("hdfs://<output>")
```



Example: filter Error lines from logfile

HDFS
file

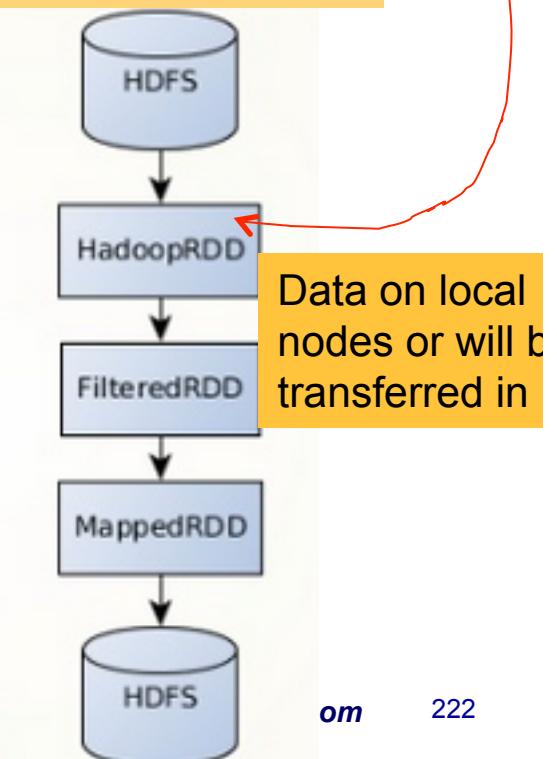
INFO not much going on here
ERROR 30330 task aborted with no status
ERROR 44404 task aborted with no status

pySpark

```
sc.textFile("hdfs://<some input path or any local file>") \
    .filter(lambda x: 'ERROR' in x) # Error anywhere in the line
    .map(lambda line: line.split(" ")[1]) #second word and the original line
    .saveAsTextFile("hdfs://<output> or any local directory")
```

Scala

```
sc.textFile("hdfs://<input>")
    .filter(_.startsWith("ERROR"))
    .map(_.split(" ")(1))
    .saveAsTextFile("hdfs://<output>")
```



Example: filter Error lines from logfile

HDFS
file

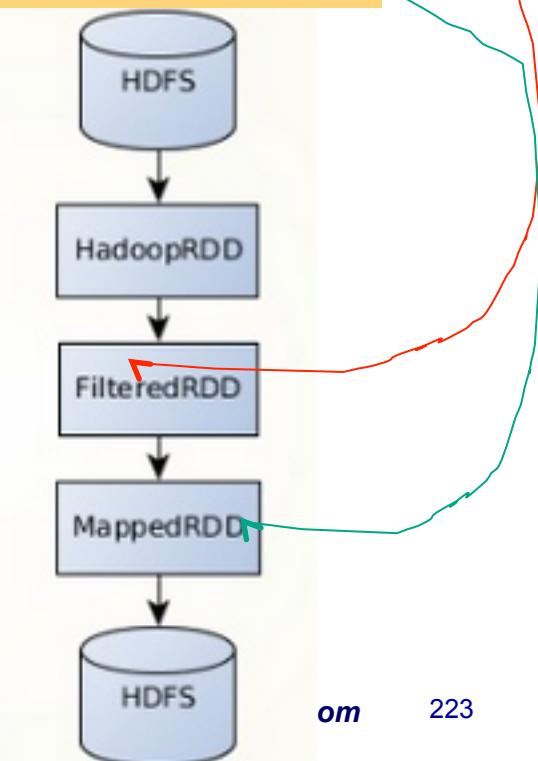
INFO not much going on here
ERROR 30330 task aborted with no status
ERROR 44404 task aborted with no status

pySpark

```
sc.textFile("hdfs://<some input path or any local file>") \
    .filter(lambda x: 'ERROR' in x) # Error anywhere in the line
    .map(lambda line: line.split(" ")[1]) #second word and the original line
    .saveAsTextFile("hdfs://<output> or any local directory")
```

Scala

```
sc.textFile("hdfs://<input>")
    .filter(_.startsWith("ERROR"))
    .map(_.split(" ")(1))
    .saveAsTextFile("hdfs://<output>")
```



saveAsTextFile Action: Step by Step

HDFS
file

INFO not much going on here
ERROR 30330 task aborted with no status
ERROR 44404 task aborted with no status

pySpark

```
sc.textFile("hdfs://<some input path or any local file>") \
    .filter(lambda x: 'ERROR' in x) # Error anywhere in the line
    .map(lambda line: line.split(" ")[1]) #second word and the original line
    .saveAsTextFile("hdfs://<output> or any local directory")
```



The RDD.saveAsTextFile() action triggers a job. Tasks are started on scheduled executors.

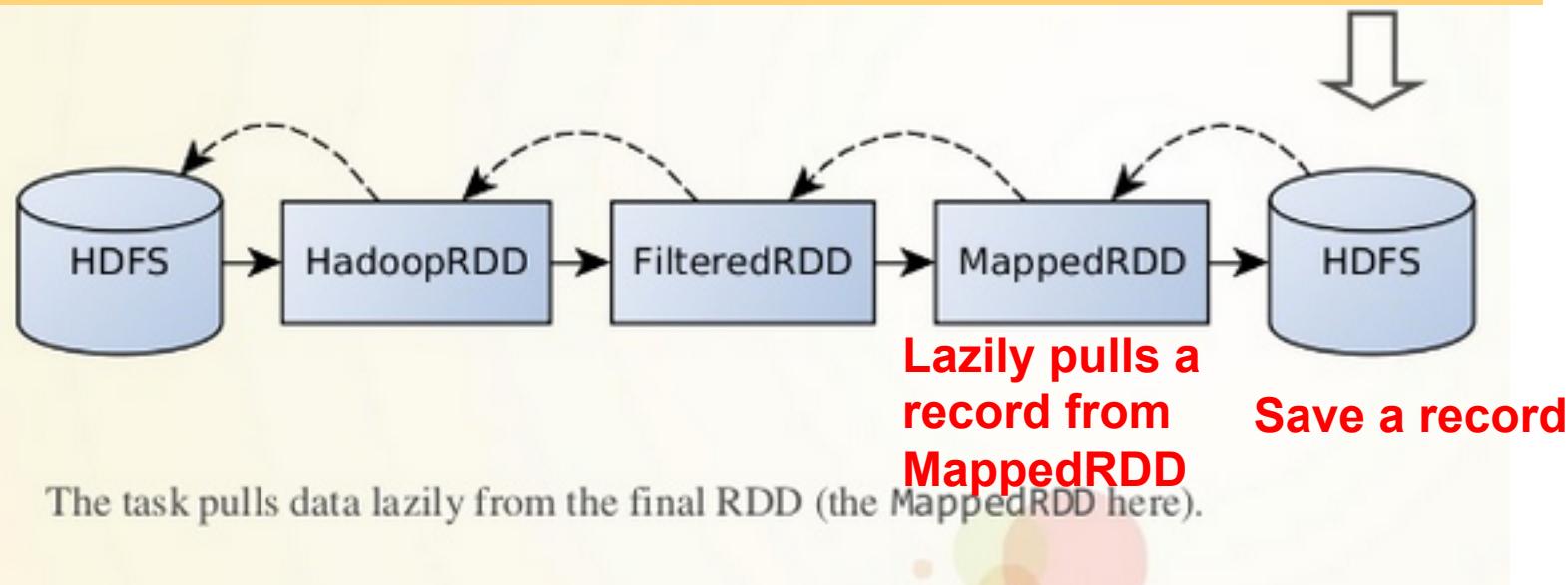
saveAsTextFile has a domino effect

HDFS
file

INFO not much going on here
ERROR 30330 task aborted with no status
ERROR 44404 task aborted with no status

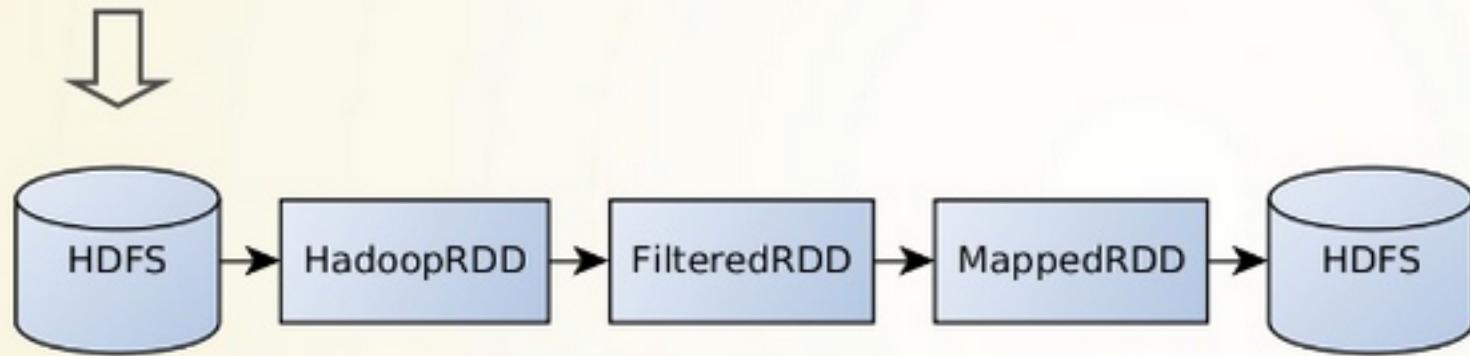
pySpark

```
sc.textFile("hdfs://<some input path or any local file>") \
    .filter(lambda x: 'ERROR' in x)      # Error anywhere in the line
    .map(lambda line: line.split(" ")[1])  #second word and the original line
    .saveAsTextFile("hdfs://<output> or any local directory")
```



Chain effect right back to the source

Step by Step



A record (text line) is pulled out from HDFS...

Materializes a record that is passed forward in the pipeline

HDFS
file

INFO not much going on here

ERROR 30330 task aborted with no status

• ERROR 44404 task aborted with no status

INFO doodle

pySpark

```
sc.textFile("hdfs://<some input path or any local file") \
    .filter(lambda x: 'ERROR' in x)      # Error anywhere in the line
    .map(lambda line: line.split(" ")[1])  #second word and the original line
    .saveAsTextFile("hdfs://<output> or any local directory")
```

INFO ...



... into a HadoopRDD

Record is passed thru the filter

HDFS
file

INFO not much going on here
ERROR 30330 task aborted with no status
• ERROR 44404 task aborted with no status

pySpark

```
sc.textFile("hdfs://<some input path or any local file>") \
    .filter(lambda x: 'ERROR' in x)      # Error anywhere in the line
    .map(lambda line: line.split(" ")[1])  #second word and the original line
    .saveAsTextFile("hdfs://<output> or any local directory")
```

INFO ...



Then filtered with the FilteredRDD

Record is passed thru the filter but Rejected

HDFS
file

INFO not much going on here
ERROR 30330 task aborted with no status
ERROR 44404 task aborted with no status

pySpark

```
sc.textFile("hdfs://<some input path or any local file>") \
    .filter(lambda x: 'ERROR' in x)      # Error anywhere in the line
    .map(lambda line: line.split(" ")[1])  #second word and the original line
    .saveAsTextFile("hdfs://<output> or any local directory")
```



Oops, not a match

Action: Take 2

HDFS
file

- INFO not much going on here
- ERROR 30330 task aborted with no status
- ERROR 44404 task aborted with no status
- INFO doodle

pySpark

```
sc.textFile("hdfs://<some input path or any local file>") \
    .filter(lambda x: 'ERROR' in x)      # Error anywhere in the line
    .map(lambda line: line.split(" ")[1])  #second word and the original line
    .saveAsTextFile("hdfs://<output> or any local directory")
```

ERROR ...



Another record is pulled out...

Pass second record thru filter

HDFS
file

- INFO not much going on here
- ERROR 30330 task aborted with no status
- ERROR 44404 task aborted with no status
- INFO doodle

pySpark

```
sc.textFile("hdfs://<some input path or any local file>") \
    .filter(lambda x: 'ERROR' in x)      # Error anywhere in the line
    .map(lambda line: line.split(" ")[1])  #second word and the original line
    .saveAsTextFile("hdfs://<output> or any local directory")
```

ERROR ...



Filtered again...

Passes file and write to HDFS

HDFS
file

INFO not much going on here

ERROR 30330 task aborted with no status

• ERROR 44404 task aborted with no status

INFO doodle

ERROR 30330 task aborted with no status

pySpark

```
sc.textFile("hdfs://<some input path or any local file") \
    .filter(lambda x: 'ERROR' in x)      # Error anywhere in the line
    .map(lambda line: line.split(" ")[1])  #second word and the original line
    .saveAsTextFile("hdfs://<output> or any local directory")
```



Transformed by the MappedRDD (the error message is extracted)

A Synthesized Iterator

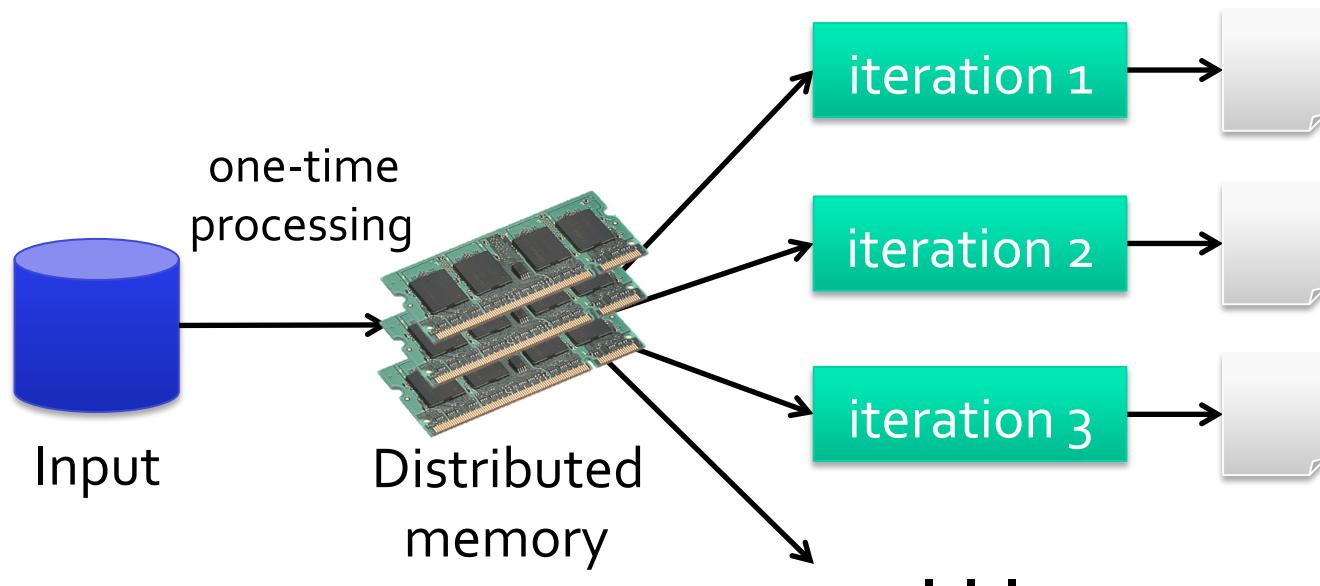
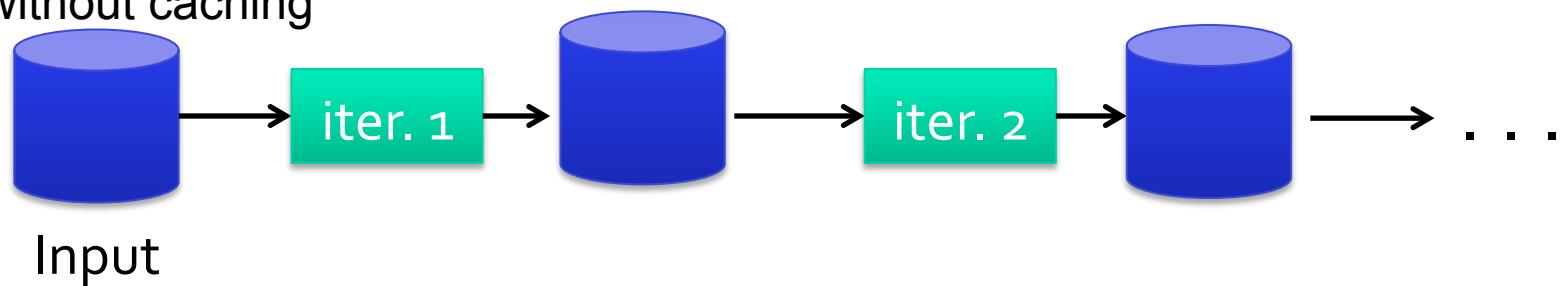
```
new Iterator[String] {  
    private var head: String = _  
    private var headDefined: Boolean = false  
  
    def hasNext: Boolean = headDefined || {  
        do {  
            try head = readOneLineFromHDFS(...)      // (1) read from HDFS  
            catch {  
                case _: EOFException => return false  
            }  
        } while (!head.startsWith("ERROR"))          // (2) filter closure  
        true  
    }  
  
    def next: String = if (hasNext) {  
        headDefined = false  
        head.split(" ")(1)                         // (3) map closure  
    } else {  
        throw new NoSuchElementException("...")  
    }  
}
```

Constant Space Complexity!

- Caching

Goal: Keep Working Set in RAM

Hadoop MapReduce
Spark without caching



Caching

While the contents of RDDs are transient by default, Spark provides a mechanism persisting the data in an RDD. After the first time an action requires computing such an RDD's contents, they are stored in memory or disk across the cluster. The next time an action depends on the RDD, it need not be recomputed from its dependencies. Its data is returned from the cached partitions directly.

```
cached.cache()  
cached.count()  
cached.take(10)
```

The call to `cache` indicates that the RDD should be stored the next time it's computed. The call to `count` computes it initially. The `take` action returns the first 10 elements the RDD as a local `Array`. When `take` is called, it accesses the cached elements of `cached` instead of recomputing them from their dependencies.

Spark defines a few different mechanisms, or `StorageLevel`s, for persisting RDDs. `rdd.cache()` is shorthand for `rdd.persist(StorageLevel.MEMORY)`, which stores the RDD as unserialized Java objects. When Spark estimates that a partition will not fit in memory, it simply will not store it, and it will be recomputed the next time it's needed. This level makes the most sense when the objects will be referenced frequently and/or require low-latency access, as it avoids any serialization overhead. Its drawback is that it takes up larger amounts of memory than its alternatives. Also, holding on to many small objects puts pressure on Java's garbage collection, which can result in stalls and

Large general slowness.

The In-Memory Magic

- With cache
 - One block per RDD partition
 - LRU cache eviction
 - Locality aware
 - Evicted blocks can be *recomputed in parallel* with the help of RDD lineage DAG

Cached intermediate (use by multiple downstream tasks if necessary instead of recalculating)

HDFS
file

- INFO not much going on here
- ERROR 30330 task aborted with no status
- ERROR 44404 task aborted with no status

INFO doodle

pySpark

```
cached= sc.textFile("hdfs://<some input path or any local file") \
    .filter(lambda x: 'ERROR' in x).cache() # Error anywhere in the line

cached.map(lambda line: line.split(" ")[1]) #second word and the original line
    .saveAsTextFile("hdfs://<output> or any local directory")
```

```
val cached = sc
    .textFile("hdfs://<input>")
    .filter(_.startsWith("ERROR"))
    .cache()

cached
    .map(_.split(" ")(1))
    .saveAsTextFile("hdfs://<output>")
```

ERRORs are cached

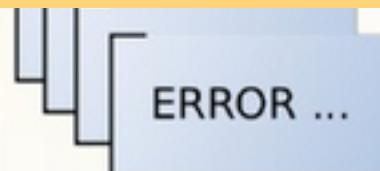
HDFS
file

- INFO not much going on here
- ERROR 30330 task aborted with no status
- ERROR 44404 task aborted with no status

pySpark

```
cached= sc.textFile("hdfs://<some input path or any local file>") \
    .filter(lambda x: 'ERROR' in x).cache() # Error anywhere in the line
```

```
cached.map(lambda line: line.split(" ")[1]) #second word and the original line
    .saveAsTextFile("hdfs://<output> or any local directory")
```



All filtered error messages are cached in memory before being passed to the next RDD. LRU cache eviction is applied when memory is insufficient

help(pyspark)

• ..

Interactive Use

The bin/pyspark script launches a Python interpreter that is configured to run PySpark applications. To use pyspark interactively, first build Spark, then launch it directly from the command line without any options:

```
$ sbt/sbt assembly  
$ ./bin/pyspark
```

The Python shell can be used explore data interactively and is a simple way to learn the API:

```
>>> words = sc.textFile("/usr/share/dict/words")  
>>> words.filter(lambda w: w.startswith("spar")).take(5)  
[u'spar', u'sparable', u'sparada', u'sparadrap', u'sparagrass']  
>>> help(pyspark) # Show all pyspark functions
```

By default, the bin/pyspark shell creates SparkContext that runs applications locally on a single core. To connect to a non-local cluster, or use multiple cores, set the MASTER environment variable. For example, to use the bin/pyspark shell with a [standalone Spark cluster](#):

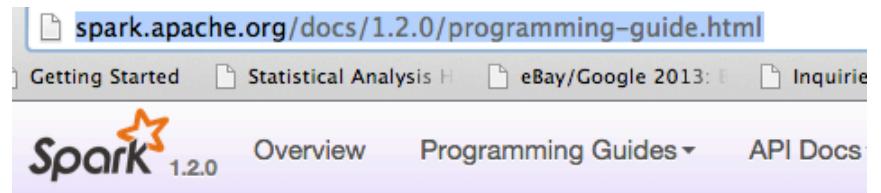
```
$ MASTER=spark://IP:PORT ./bin/pyspark
```

Or, to use four cores on the local machine:

```
$ MASTER=local[4] ./bin/pyspark
```

Spark in one-page!

- <http://spark.apache.org/docs/1.4.0/programming-guide.html>



Spark Programming Guide

- Overview
- Linking with Spark
- Initializing Spark
 - Using the Shell
- Resilient Distributed Datasets (RDDs)
 - Parallelized Collections
 - External Datasets
 - RDD Operations
 - Basics
 - Passing Functions to Spark
 - Working with Key-Value Pairs
 - Transformations
 - Actions
 - RDD Persistence
 - Which Storage Level to Choose?
 - Removing Data
- Shared Variables
 - Broadcast Variables
 - Accumulators
- Deploying to a Cluster
- Unit Testing
- Migrating from pre-1.0 Versions of Spark

Summary

- I hope this example was useful in helping you understand the power of Spark and to ground some of the concepts

help(pyspark)

• ..

Interactive Use

The bin/pyspark script launches a Python interpreter that is configured to run PySpark applications. To use pyspark interactively, first build Spark, then launch it directly from the command line without any options:

```
$ sbt/sbt assembly  
$ ./bin/pyspark
```

The Python shell can be used explore data interactively and is a simple way to learn the API:

```
>>> words = sc.textFile("/usr/share/dict/words")  
>>> words.filter(lambda w: w.startswith("spar")).take(5)  
[u'spar', u'sparable', u'sparada', u'sparadrap', u'sparagrass']  
>>> help(pyspark) # Show all pyspark functions
```

By default, the bin/pyspark shell creates SparkContext that runs applications locally on a single core. To connect to a non-local cluster, or use multiple cores, set the MASTER environment variable. For example, to use the bin/pyspark shell with a [standalone Spark cluster](#):

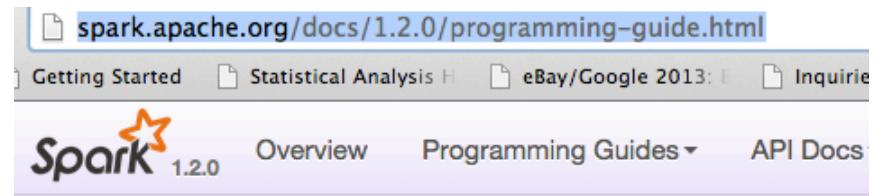
```
$ MASTER=spark://IP:PORT ./bin/pyspark
```

Or, to use four cores on the local machine:

```
$ MASTER=local[4] ./bin/pyspark
```

Spark in one-page!

- <http://spark.apache.org/docs/1.4.0/programming-guide.html>



Spark Programming Guide

- Overview
- Linking with Spark
- Initializing Spark
 - Using the Shell
- Resilient Distributed Datasets (RDDs)
 - Parallelized Collections
 - External Datasets
 - RDD Operations
 - Basics
 - Passing Functions to Spark
 - Working with Key-Value Pairs
 - Transformations
 - Actions
 - RDD Persistence
 - Which Storage Level to Choose?
 - Removing Data
- Shared Variables
 - Broadcast Variables
 - Accumulators
- Deploying to a Cluster
- Unit Testing
- Migrating from pre-1.0 Versions of Spark

Transformations

The following table lists some of the common transformations supported by Spark. Refer to the RDD API doc ([Scala](#), [Java](#), [Python](#)) and pair RDD functions doc ([Scala](#), [Java](#)) for details.

Transformation	Meaning
<code>map(func)</code>	Return a new distributed dataset formed by passing each element of the source through a function <i>func</i> .
<code>filter(func)</code>	Return a new dataset formed by selecting those elements of the source on which <i>func</i> returns true.
<code>flatMap(func)</code>	Similar to map, but each input item can be mapped to 0 or more output items (so <i>func</i> should return a Seq rather than a single item).
<code>mapPartitions(func)</code>	Similar to map, but runs separately on each partition (block) of the RDD, so <i>func</i> must be of type <code>Iterator<T> => Iterator<U></code> when running on an RDD of type T.
<code>mapPartitionsWithIndex(func)</code>	Similar to mapPartitions, but also provides <i>func</i> with an integer value representing the index of the partition, so <i>func</i> must be of type <code>(Int, Iterator<T>) => Iterator<U></code> when running on an RDD of type T.
<code>sample(withReplacement, fraction, seed)</code>	Sample a fraction <i>fraction</i> of the data, with or without replacement, using a given random number generator seed.
<code>union(otherDataset)</code>	Return a new dataset that contains the union of the elements in the source dataset and the argument.
<code>intersection(otherDataset)</code>	Return a new RDD that contains the intersection of elements in the source dataset and the argument.
<code>distinct([numTasks])</code>	Return a new dataset that contains the distinct elements of the source dataset.
<code>groupByKey([numTasks])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. Note: If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using <code>reduceByKey</code> or <code>combineByKey</code> will yield much better performance. Note: By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional numTasks argument to set a different number of tasks.
<code>reduceByKey(func, [numTasks])</code>	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 <i>func</i> , which must be of type <code>(V,V) => V</code> . Like in groupByKey, the number of reduce tasks is configurable through an optional second argument.
<code>aggregateByKey(zeroValue)(seqOp, combOp, [numTasks])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value. Allows an aggregated value type that is different than the input value type, while avoiding unnecessary allocations. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument.
<code>sortByKey([ascending], [numTasks])</code>	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, as specified in the boolean ascending argument.
<code>join(otherDataset, [numTasks])</code>	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with

Actions

The following table lists some of the common actions supported by Spark. Refer to the RDD API doc ([Scala](#), [Java](#), [Python](#)) and pair RDD functions doc ([Scala](#), [Java](#)) for details.

Action	Meaning
<code>reduce(func)</code>	Aggregate the elements of the dataset using a function <i>func</i> (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
<code>collect()</code>	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
<code>count()</code>	Return the number of elements in the dataset.
<code>first()</code>	Return the first element of the dataset (similar to <code>take(1)</code>).
<code>take(n)</code>	Return an array with the first <i>n</i> elements of the dataset. Note that this is currently not executed in parallel. Instead, the driver program computes all the elements.
<code>takeSample(withReplacement, num, [seed])</code>	Return an array with a random sample of <i>num</i> elements of the dataset, with or without replacement, optionally pre-specifying a random number generator seed.
<code>takeOrdered(n, [ordering])</code>	Return the first <i>n</i> elements of the RDD using either their natural order or a custom comparator.
<code>saveAsTextFile(path)</code>	Write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call <code>toString</code> on each element to convert it to a line of text in the file.
<code>saveAsSequenceFile(path)</code> (Java and Scala)	Write the elements of the dataset as a Hadoop SequenceFile in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. This is available on RDDs of key-value pairs that either implement Hadoop's Writable interface. In Scala, it is also available on types that are implicitly convertible to Writable (Spark includes conversions for basic types like Int, Double, String, etc).
<code>saveAsObjectFile(path)</code> (Java and Scala)	Write the elements of the dataset in a simple format using Java serialization, which can then be loaded using <code>SparkContext.objectFile()</code> .
<code>countByKey()</code>	Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.
<code>foreach(func)</code>	Run a function <i>func</i> on each element of the dataset. This is usually done for side effects such as updating an accumulator variable (see below) or interacting with external storage systems.

Source Code available on GitHub

Part 2: Spark Intro and Basics

- **Part 2: Spark Intro and basics**

- Base RDD
- Fault tolerance (and lineage)
- Transformations and Actions
- Persistance
- Animated Example
- Pair RDDs
- Word count example

PAIR RDDS

Key/value RDDs expose new operations (e.g., counting up reviews for each product, grouping together data with the same key, and grouping together two different RDDs).

Pair RDDs

This section covers how to work with RDDs of key/value pairs, which are a common data type required for many operations in Spark.

Key/value RDDs are commonly used to perform aggregations, and often we will do some initial ETL (extract, transform, and load) to get our data into a key/value format.

Key/value RDDs expose new operations (e.g., counting up reviews for each product, grouping together data with the same key, and grouping together two different RDDs).

Pair RDDs

We also discuss an advanced feature that lets users control the layout of pair RDDs across nodes: partitioning.

Using controllable partitioning, applications can sometimes greatly reduce communication costs by ensuring that data will be accessed together and will be on the same node.

JOINS: (see in later sections in this lecture and subsequent lectures)

- This can provide significant speedups. We illustrate partitioning using the PageRank algorithm as an example.
- Choosing the right partitioning for a distributed dataset is similar to choosing the right data structure for a local one—in both cases, data layout can greatly affect performance.

Pair RDDs: Key value pairs

- **Spark provides special operations on RDDs containing key/value pairs. These RDDs are called pair RDDs.**
- **Act on each key in parallel or regroup data across the network**
 - Pair RDDs are a useful building block in many programs, as they expose operations that allow you to act on each key in parallel or regroup data across the network.
- **reduceByKey(): aggregate data separately for each key**
 - For example, pair RDDs have a reduceByKey() method that can aggregate data separately for each key, and a join() method that can merge two RDDs together by grouping elements with the same key.
- **It is common to extract fields from an RDD (representing, for instance, an event time, customer ID, or other identifier) and use those fields as keys in pair RDD operations**

Pair RDDs Example

The way to build key-value RDDs differs by language. In Python, for the functions on keyed data to work we need to return an RDD composed of tuples (see [Example 4-1](#)).

Example 4-1. Creating a pair RDD using the first word as the key in Python

KEY VALUE
pairs = lines.map(lambda x: (x.split(" ")[0], x))

When creating a pair RDD from an in-memory collection in Scala and Python, we only need to call `SparkContext.parallelize()` on a collection of pairs.

Pair RDDs are allowed to use all the transformations available to standard RDDs. The same rules apply from “Passing Functions to Spark” on page 30 [Learning spark book].

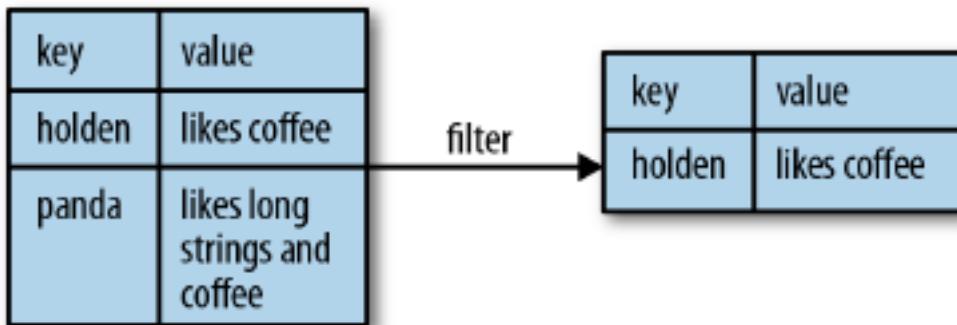


Figure 4-1. Filter on value

Pair RDDs are also still RDDs (of Tuple2 objects in Java/Scala or of Python tuples), and thus support the same functions as RDDs. For instance, we can take our pair RDD from the previous section and filter out lines longer than 20 characters, as shown in Examples 4-4 through 4-6 and Figure 4-1.

Filter on Value being < 20

Example 4-4. Simple filter on second element in Python

```
result = pairs.filter(lambda keyValue: len(keyValue[1]) < 20)
```

Example 4-5. Simple filter on second element in Scala

```
pairs.filter{case (key, value) => value.length < 20}
```

Part 2: Spark Intro and Basics

- **Part 2: Spark Intro and basics**

- Base RDD
- Fault tolerance (and lineage)
- Transformations and Actions
- Persistance
- Animated Example
- Pair RDDs
- Word count example

Simple Spark Apps: WordCount

Definition:

*count how often each word appears
in a collection of text documents*

This simple program provides a good test case for parallel processing, since it:

- requires a minimal amount of code
- demonstrates use of both symbolic and numeric values
- isn't many steps away from search indexing
- serves as a "Hello World" for Big Data apps

A distributed computing framework that can run WordCount **efficiently in parallel at scale** can likely handle much larger and more interesting compute problems

Example 3

```
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));
```

Word Frequency

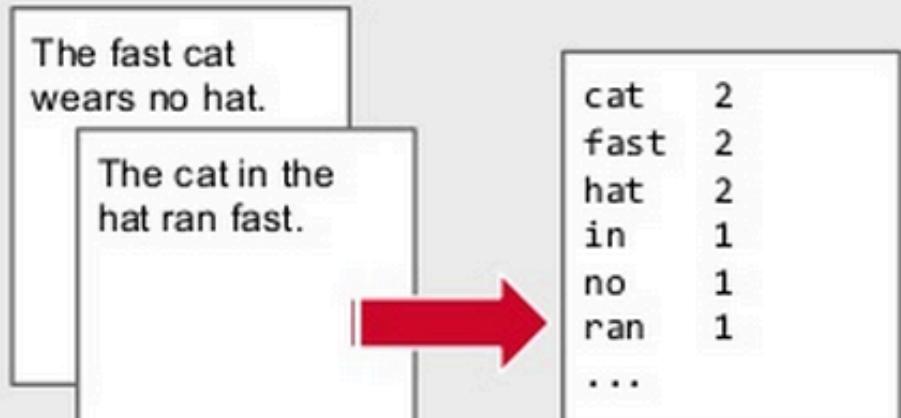
```
def map(key, value):
    for word in value.split():
        emit(word, 1)
```

```
def reduce(key, values):
    count = 0
    for val in values:
        count += val
    emit(key, count)
```

emit is a function that performs distributed I/O

Each document is passed to a mapper, which does the tokenization. The output of the mapper is reduced by key (word) and then counted.

What is the data flow for word count?



Word Frequency

```
from operator import add

def tokenize(text):
    return text.split()

text = sc.textFile("tolstoy.txt")      # Create RDD

# Transform
wc    = text.flatMap(tokenize)
wc    = wc.map(lambda x: (x,1)).reduceByKey(add)

wc.saveAsTextFile("counts")           # Action
```



Word Count

See NOTEBOOK

Write some words into a file

```
%%writefile wordcount.txt
hello hi hi hallo
bonjour hola hi ciao
nihao konnichiwa ola
hola nihao hello
```

Overwriting wordcount.txt

Word Count

Attention:

flatMap works applying a function
that returns a sequence for each
element in the list, and flattening
the results into the original list.

```
#Count words in README.md
logFileName = 'wordcount.txt'
text_file = sc.textFile(logFileName)
counts = text_file.flatMap(lambda line: line.split(" "))
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a + b)
for v in counts.collect():
    print v

(u'ciao', 1)
(u'bonjour', 1)
(u'nihao', 2)
(u'hola', 2)
(u'konnichiwa', 1)
(u'hallo', 1)
(u'hi', 3)
(u'hello', 2)
(u'ola', 1)
```

Word Count

```
1 hello hi hi hallo  
2 bonjour hola hi ciao  
3 nihao konnichiwa ola  
4 hola nihao hello
```

Flat
Map

```
[hello, hi, hi, hallo, bonjour, hola, hi, ciao,  
nihao, konnichiwa, ola, hola, nihao, hello]
```

Map

```
(u'ciao', 1)  
(u'bonjour', 1)  
(u'nihao', 2)  
(u'holo', 2)  
(u'konnichiwa', 1)  
(u'hallo', 1)  
(u'hi', 3)  
(u'hello', 2)  
(u'ola', 1)
```

reduce
ByKey

```
(hello,1),  
(hi,1),  
(hi,1),  
(hallo,1),  
(bonjour,1),  
(holo,1),  
(hi,1),  
(ciao,1),  
(nihao,1),  
(konnichiwa,1),  
(ola,1),  
(holo,1),  
(nihao,1),  
(hello,1)
```

```
]#Example 4-1. Creating a pair RDD using the first word as the key in Python
lines = sc.parallelize(["Data line 1", "Mining line 2", "data line 3", "Data line 4", "Data Mining line 5"])
pairs = lines.map(lambda x: (x.split(" ")[0], x)) #first word and the original line
pairs.collect()

] [('Data', 'Data line 1'),
 ('Mining', 'Mining line 2'),
 ('data', 'data line 3'),
 ('Data', 'Data line 4'),
 ('Data', 'Data Mining line 5')]
```

1:

```
In [15]: #Paired RDD examples
logFileName='log.txt'
#Word count for error messages exercise
# READ Lines And PRINT
recs = sc.textFile(logFileName)
#recs = sc.textFile(logFileName).filter(lambda l: "ERROR" in l)
wcErrors = recs.flatMap(lambda l: l.split(" ")) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda x, y: x + y)
for (key, value) in wcErrors.collect():
    print "WordCount-->> %s:%d"%(key, value)
```

```
WordCount-->> :1
WordCount-->> dying:1
WordCount-->> for:1
WordCount-->> reasons:1
WordCount-->> ERROR      mysql:1
WordCount-->> angry:1
WordCount-->> cluster::1
WordCount-->> context:1
WordCount-->> spark:2
WordCount-->> with:1
WordCount-->> you:1
WordCount-->> me?:1
WordCount-->> WARN       dave,:1
WordCount-->> ERROR      php::1
WordCount-->> unknown:1
WordCount-->> it:1
WordCount-->> replace:1
WordCount-->> cluster:1
WordCount-->> missing...darn:1
WordCount-->> WARN       xylons:1
WordCount-->> at:1
WordCount-->> ERROR      :1
WordCount-->> approaching:1
WordCount-->> are:1
```

All actions available on the Base RDD

Plus more

Transformations on one Pair RDD

$\{(1, 2), (3, 4), (3, 6)\}$

Table 4-1. Transformations on one pair RDD (example: $\{(1, 2), (3, 4), (3, 6)\}$)

Function name	Purpose	Example	Result
reduceByKey(func)	Combine values with the same key.	<code>rdd.reduceByKey((x, y) => x + y)</code>	$\{(1, 2), (3, 4), (3, 6)\}$ $\{(1, 2), (3, 10)\}$
groupByKey()	Group values with the same key.	<code>rdd.groupByKey()</code>	$\{(1, [2]), (3, [4, 6])\}$
combineByKey Key(createCombiner, mergeValue, mergeCombiners, partitioner)	Combine values with the same key using a different result type.	See Examples 4-12 through 4-14.	Value is a List

Transformations on one Pair RDD

$\{(1, 2), (3, 4), (3, 6)\}$

`mapValues(func)`

Apply a function to each value of a pair RDD without changing the key.

`rdd.mapValues(x => x+1)`

$\{(1, 3), (3, 5), (3, 7)\}$

`flatMapValues(func)`

Apply a function that returns an iterator to each value of a pair RDD, and for each element returned, produce a key/value entry with the old key. Often used for tokenization.

`rdd.flatMapValues(x => (x to 5)`

$\{(1, 2), (1, 3), (1, 4), (1, 5), (3, 4), (3, 5)\}$

`keys()`

Return an RDD of just the keys.

`rdd.keys()`

$\{1, 3, 3\}$

$\{1, 3, 3\}$

`values()`

Return an RDD of just the values.

`rdd.values()`

$\{2, 4, 6\}$

$\{2, 4, 6\}$

`sortByKey()`

Return an RDD sorted by the key.

`rdd.sortByKey()`

$\{(1, 2), (3, 4), (3, 5), (3, 6)\}$

an@gmail.com

265

Transformations on one Pair RDD

$\{(1, 2), (3, 4), (3, 6)\}$

$\{(1, 3), (3, 5), (3, 7)\}$

Function name	Purpose	Example	Result
<code>values()</code>	Return an RDD of just the values.	<code>rdd.values()</code>	$\{2, 4, 6\}$
<code>sortByKey()</code>	Return an RDD sorted by the key.	<code>rdd.sortByKey()</code>	$\{(1, 2), (3, 4), (3, 6)\}$

$\{1, 3, 3\}$

Python:

Curly braces create [dictionaries](#) or [sets](#). Square brackets create [lists](#).

The are called literals; set literals:

```
aset = {'foo', 'bar'}
```

or a dictionary literal:

```
adict = {'foo': 42, 'bar': 81}  
empty_dict = {}
```

or a list literal:

```
alist = ['foo', 'bar', 'bar']  
empty_list = []
```

To create an empty set, you can only use `set()`.

Sets are collections of *unique* elements and you cannot order them. Lists are ordered sequences of elements, and values can be repeated. Dictionaries map keys to values, keys must be unique. Set and dictionary keys must meet other restrictions as well, so that Python can actually keep track of them efficiently and know they are and will remain unique.

joins

Joins

Some of the most useful operations we get with keyed data comes from using it together with other keyed data. Joining data together is probably one of the most common operations on a pair RDD, and we have a full range of options including right and left outer joins, cross joins, and inner joins.

The simple `join` operator is an inner join.¹ Only keys that are present in both pair RDDs are output. When there are multiple values for the same key in one of the inputs, the resulting pair RDD will have an entry for every possible pair of values with that key from the two input RDDs. A simple way to understand this is by looking at [Example 4-17](#).

Inner join: Transformation of two pair RDDs

Table 4-2. Transformations on two pair RDDs ($rdd = \{(1, 2), (3, 4), (3, 6)\}$ other = $\{(3, 9)\}$)

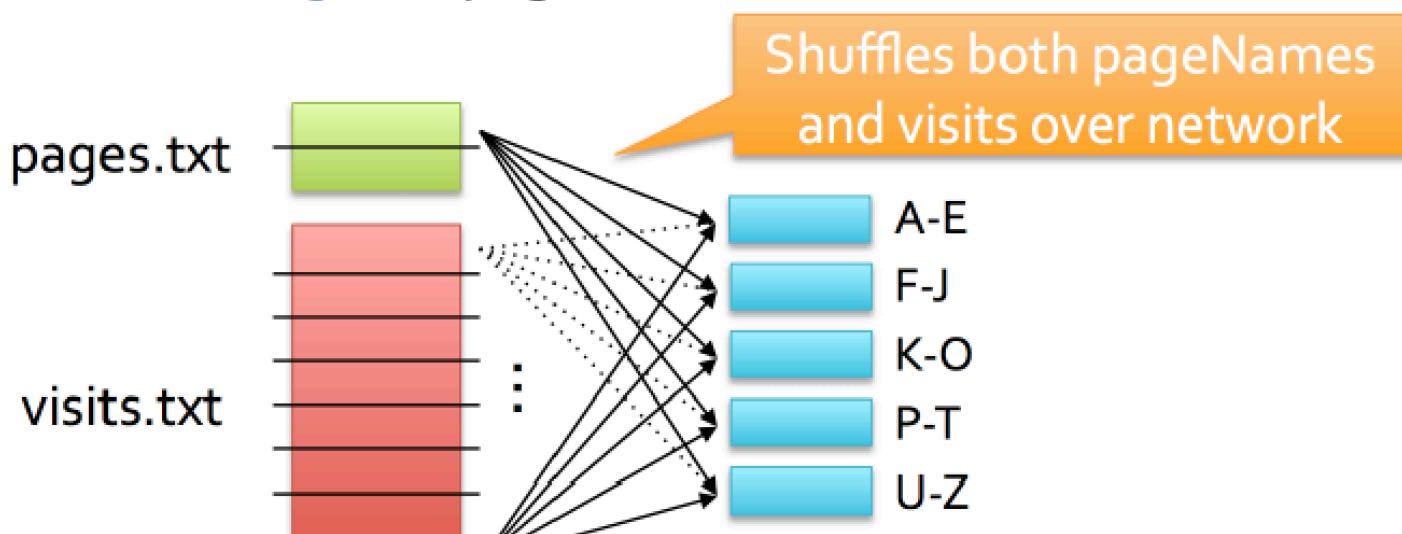
Function name	Purpose	Example	Result
subtractByKey	Remove elements with a key present in the other RDD.	<code>rdd.subtractByKey(other)</code> <code>rdd = {(1, 2), (3, 4), (3, 6)}</code> other = <code>{(3, 9)}</code>	<code>{(1, 2)}</code>
join	Perform an inner join between two RDDs.	<code>rdd.join(other)</code>	<code>{(3, (4, 9)), (3, (6, 9))}</code>

Keys must be present in both RDDs

Joins (not efficient...WHY?)

Example Join

```
// Load RDD of (URL, name) pairs  
val pageNames = sc.textFile("pages.txt").map(...)  
  
// Load RDD of (URL, visit) pairs  
val visits = sc.textFile("visits.txt").map(...)  
  
val joined = visits.join(pageNames)
```



Transformation of two pair RDDs

Table 4-2. Transformations on two pair RDDs ($rdd = \{(1, 2), (3, 4), (3, 6)\}$ other = $\{(3, 9)\}$)

Function name	Purpose	Example	Result
subtractByKey	Remove elements with a key present in the other RDD.	<code>rdd.subtractByKey(other)</code>	$\{(1, 2)\}$
join	Perform an inner join between two RDDs.	<code>rdd.join(other)</code>	$\{(3, (4, 9)), (3, (6, 9))\}$
rightOuterJoin	Perform a join between two RDDs where the key must be present in the first RDD.	<code>rdd.rightOuterJoin(other)</code>	$\{(3, (\text{Some}(4), 9)), (3, (\text{Some}(6), 9))\}$
leftOuterJoin	Perform a join between two RDDs where the key must be present in the other RDD.	<code>rdd.leftOuterJoin(other)</code>	$\{(1, (2, \text{None})), (3, (4, \text{Some}(9))), (3, (6, \text{Some}(9)))\}$
cogroup	Group data from both RDDs sharing the same key.	<code>rdd.cogroup(other)</code>	$\{(1, ([2], [])), (3, ([4, 6], [9]))\}$

Aggregations

When datasets are described in terms of key/value pairs, it is common to want to aggregate statistics across all elements with the same key. We have looked at the `fold()`, `combine()`, and `reduce()` actions on basic RDDs, and similar per-key transformations exist on pair RDDs. Spark has a similar set of operations that combines values that have the same key. These operations return RDDs and thus are transformations rather than actions.

`reduceByKey()` is quite similar to `reduce()`; both take a function and use it to combine values. `reduceByKey()` runs several parallel reduce operations, one for each key in the dataset, where each operation combines values that have the same key. Because datasets can have very large numbers of keys, `reduceByKey()` is not implemented as an action that returns a value to the user program. Instead, it returns a new RDD consisting of each key and the reduced value for that key.

Example 4-7. Per-key average with reduceByKey() and mapValues() in Python

```
rdd.mapValues(lambda x: (x, 1)).reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))
```

Example 4-8. Per-key average with reduceByKey() and mapValues() in Scala

```
rdd.mapValues(x => (x, 1)).reduceByKey((x, y) => (x._1 + y._1, x._2 + y._2))
```

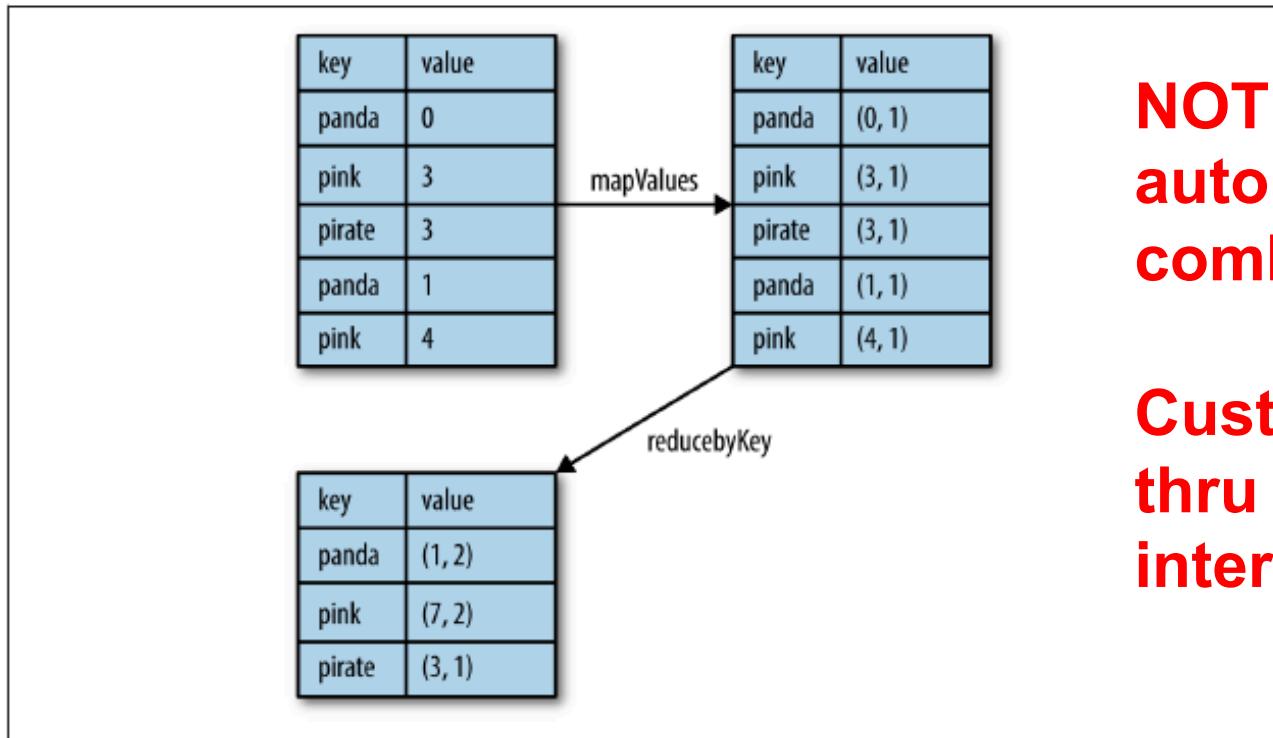


Figure 4-2. Per-key average data flow



Those familiar with the combiner concept from MapReduce should note that calling `reduceByKey()` and `foldByKey()` will automatically perform combining locally on each machine before computing global totals for each key. The user does not need to specify a combiner. The more general `combineByKey()` interface allows you to customize combining behavior.

Pair RDD for wordcount

We can use a similar approach in Examples 4-9 through 4-11 to also implement the classic distributed word count problem. We will use `flatMap()` from the previous chapter so that we can produce a pair RDD of words and the number 1 and then sum together all of the words using `reduceByKey()` as in Examples 4-7 and 4-8.

Example 4-9. Word count in Python

```
rdd = sc.textFile("s3://...")  
words = rdd.flatMap(lambda x: x.split(" "))  
result = words.map(lambda x: (x, 1)).reduceByKey(lambda x, y: x + y)
```

Since each partition is processed independently, we can have multiple accumulators for the same key. When we are merging the results from each partition, if two or more partitions have an accumulator for the same key we merge the accumulators using the user-supplied `mergeCombiners()` function.

Customize the combiner thru `combineByKey` interface

We can disable map-side aggregation in `combineByKey()` if we know that our data won't benefit from it. For example, `groupByKey()` disables map-side aggregation as the aggregation function (appending to a list) does not save any space. If we want to disable map-side combines, we need to specify the partitioner; for now you can just use the partitioner on the source RDD by passing `rdd.partition`.

Since `combineByKey()` has a lot of different parameters it is a great candidate for an explanatory example. To better illustrate how `combineByKey()` works, we will look at computing the average value for each key, as shown in Examples 4-12 through 4-14 and illustrated in Figure 4-3.

Example 4-12. Per-key average using `combineByKey()` in Python

```
sumCount = nums.combineByKey((lambda x: (x,1)),
                             (lambda x, y: (x[0] + y, x[1] + 1)),
                             (lambda x, y: (x[0] + y[0], x[1] + y[1])))
sumCount.map(lambda key, xy: (key, xy[0]/xy[1])).collectAsMap()
```

Partition 1	
coffee	1
coffee	2
panda	3

Partition 2	
coffee	9

```
def createCombiner(value):
    (value, 1)
```

```
def mergeValue(acc, value):
    (acc[0] + value, acc[1] + 1)
```

```
def mergeCombiners(acc1, acc2):
    (acc1[0] + acc2[0], acc1[1] + acc2[1])
```

Partition 1 trace:

(coffee, 1) -> new key

accumulators[coffee] = createCombiner(1)

(coffee, 2) -> existing key

accumulators[coffee] = mergeValue(accumulators[coffee], 2)

(panda, 3) -> new key

accumulators[panda] = createCombiner(3)

Partition 2 trace:

(coffee, 9) -> new key

accumulators[coffee] = createCombiner(9)

Merge Partitions:

mergeCombiners(partition1.accumulators[coffee],
 partition2.accumulators[coffee]))

Figure 4-3. `combineByKey()` sample data flow

All actions available on the Base RDD pls

Actions Available on Pair RDDs

As with the transformations, all of the traditional actions available on the base RDD are also available on pair RDDs. Some additional actions are available on pair RDDs to take advantage of the key/value nature of the data; these are listed in **Table 4-3**.

Table 4-3. Actions on pair RDDs (example ($\{(1, 2), (3, 4), (3, 6)\}$))

Function	Description	Example	Result
countByKey()	Count the number of elements for each key.	rdd.countByKey()	$\{(1, 1), (3, 2)\}$
collectAsMap()	Collect the result as a map to provide easy lookup.	rdd.collectAsMap()	Map{(1, 2), (3, 4), (3, 6)}
lookup(key)	Return all values associated with the provided key.	rdd.lookup(3)	[4, 6]

Parallelism

In a distributed program, communication is very expensive, so laying out data to minimize network traffic can greatly improve performance.

Spark's partitioning is available on all RDDs of key/value pairs, and causes the system to group elements based on a function of each key

Sometimes, we want to change the partitioning of an RDD outside the context of grouping and aggregation operations. For those cases, Spark provides the repartition() function, which shuffles the data across the network to create a new set of partitions. Keep in mind that repartitioning your data is a fairly expensive operation. Spark also has an optimized version of repartition() called coalesce() that allows avoiding data movement, but only if you are decreasing the number of RDD partitions.

Example 4-15. reduceByKey() with custom parallelism in Python

```
data = [("a", 3), ("b", 4), ("a", 1)]
sc.parallelize(data).reduceByKey(lambda x, y: x + y)      # Default parallelism
sc.parallelize(data).reduceByKey(lambda x, y: x + y, 10)   # Custom parallelism
```

Partitioning the data

Example 4-15. reduceByKey() with custom parallelism in Python

```
data = [("a", 3), ("b", 4), ("a", 1)]
sc.parallelize(data).reduceByKey(lambda x, y: x + y)      # Default parallelism
sc.parallelize(data).reduceByKey(lambda x, y: x + y, 10)   # Custom parallelism
```

Example 4-16. reduceByKey() with custom parallelism in Scala

```
val data = Seq(("a", 3), ("b", 4), ("a", 1))
sc.parallelize(data).reduceByKey((x, y) => x + y)      // Default parallelism
sc.parallelize(data).reduceByKey((x, y) => x + y)      // Custom parallelism
```

Sometimes, we want to change the partitioning of an RDD outside the context of grouping and aggregation operations. For those cases, Spark provides the `repartition()` function, which shuffles the data across the network to create a new set of partitions. Keep in mind that repartitioning your data is a fairly expensive operation. Spark also has an optimized version of `repartition()` called `coalesce()` that allows avoiding data movement, but only if you are decreasing the number of RDD partitions. To know whether you can safely call `coalesce()`, you can check the size of the RDD using `rdd.partitions.size()` in Java/Scala and `rdd.getNumPartitions()` in Python and make sure that you are coalescing it to fewer partitions than it currently has.

Repartition means shuffling the data across the network: EXPENSIVE!

For those cases, Spark provides the repartition() function, which shuffles the data across the network to create a new set of partitions. Keep in mind that repartitioning your data is a fairly expensive operation.

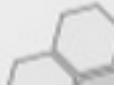
Spark also has an optimized version of repartition() called coalesce() that allows avoiding data movement, but only if you are decreasing the number of RDD partitions.

To know whether you can safely call coalesce(), you can check the size of the RDD using rdd.partitions.size() in Java/Scala and rdd.getNumPartitions() in Python and make sure that you are coalescing it to fewer partitions than it currently has.

Grouping Data
With keyed

Was that comprehensive list really necessary?

Remember in MapReduce you only get two operators - map and reduce; so maybe I'm just excited at the 80+ operations in Spark!

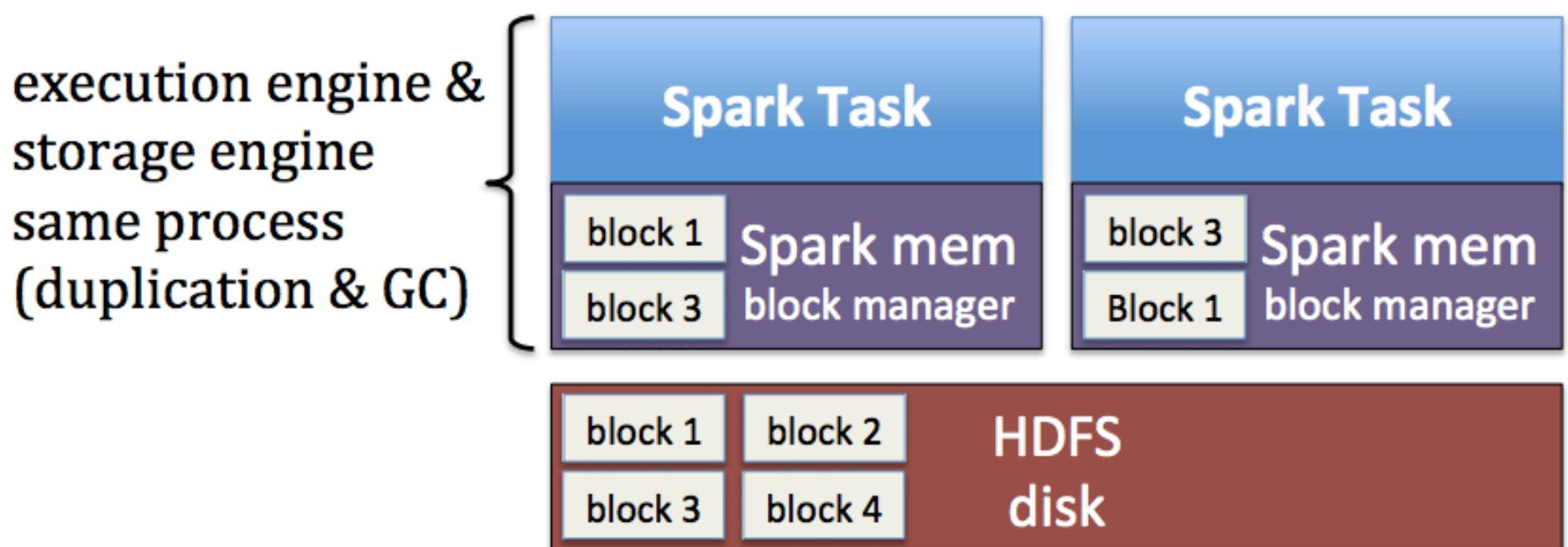


Tachyon

- Tachyon is a memory-centric distributed storage system enabling reliable data sharing at memory-speed across cluster frameworks, such as Spark and MapReduce.
- It achieves high performance by leveraging lineage information and using memory aggressively.
- Tachyon caches working set files in memory, thereby avoiding going to disk to load datasets that are frequently read. This enables different jobs/queries and frameworks to access cached files at memory speed.

Limitations of Spark, Hadoop

duplicate memory per job & GC



Computation Frameworks
(Spark, MapReduce, Impala, Tez, ...)

Tachyon

Existing Storage Systems
(HDFS, S3, GlusterFS, ...)

Tachyon

- Tachyon is a memory-centric distributed storage system enabling reliable data sharing at memory-speed across cluster frameworks, such as Spark and MapReduce.
- It achieves high performance by leveraging lineage information and using memory aggressively.
- Tachyon caches working set files in memory, thereby avoiding going to disk to load datasets that are frequently read. This enables different jobs/queries and frameworks to access cached files at memory speed.

Tutorial Outline

- **Part 1: Introduction**
 - Welcome Survey
 - Install Spark
 - Background and motivation
- **Part 2: Spark Intro and basics**
 - fundamental Spark concepts, including Spark Core, data frames, the Spark Shell, Spark Streaming, Spark SQL and vertical libraries such as MLlib and GraphX;
- **Part 3: Machine learning in Spark**
 - will focus on hands-on algorithmic design and development with Spark developing algorithms from scratch such as linear regression, logistic regression, graph processing algorithms such as pagerank/shortest path, etc.
- **Part 4: Wrap up**
 - Spark 1.5 and beyond
 - Summary

Machine Learning in Spark

- **Data Frames**
- **MLLib**
- **Write your own algos**
 - Linear regression (Ridge and Lasso)
 - Logistic Regression
- **Pipelines**
- **R and Spark**
 - Linear Regression example
- **Graphs in Spark**
- **Case studies**
 - Flight delay
 - Mobile advertising
 - Social Networks

DataFrames

- A DataFrame is a distributed collection of data 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.

-
- **The entry point into all relational functionality in Spark is the SQLContext class, or one of its decedents. To create a basic SQLContext, all you need is a SparkContext.**

Branch: master ▾

[spark](#) / [examples](#) / [src](#) / [main](#) / [resources](#) / **people.json**



yhuai on Jun 17, 2014 [SPARK-2060][SQL] Querying JSON Datasets with SQL and DSL in Spark SQL

1 contributor

4 lines (3 sloc) | 0.073 kB

Raw

```
1 {"name":"Michael"}  
2 {"name":"Andy", "age":30}  
3 {"name":"Justin", "age":19}
```

-
- As an example, the following creates a DataFrame based on the content of a JSON file:

Scala

Java

Python

R

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)

df = sqlContext.read.json("examples/src/main/resources/people.json")

# Displays the content of the DataFrame to stdout
df.show()
```

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)

# Create the DataFrame
df = sqlContext.read.json("examples/src/main/resources/people.json")

# Show the content of the DataFrame
df.show()
## age  name
## null Michael
## 30   Andy
## 19   Justin

# Print the schema in a tree format
df.printSchema()
## root
## |-- age: long (nullable = true)
## |-- name: string (nullable = true)

# Select only the "name" column
df.select("name").show()
## name
## Michael
## Andy
## Justin
```

```
# Select everybody, but increment the age by 1
df.select(df['name'], df['age'] + 1).show()
• ...
## name      (age + 1)
## Michael  null
## Andy      31
## Justin    20

# Select people older than 21
df.filter(df['age'] > 21).show()
## age name
## 30  Andy

# Count people by age
df.groupBy("age").count().show()
## age  count
## null 1
## 19   1
## 30   1
```

- ..
- ## JSON Datasets

Scala Java Python R **Sql**

```
CREATE TEMPORARY TABLE jsonTable
USING org.apache.spark.sql.json
OPTIONS (
    path "examples/src/main/resources/people.json"
)

SELECT * FROM jsonTable
```

DataFrames

```
people.json
```

```
{"name": "Michael"}  
{"name": "Andy", "age": 30}  
{"name": "Justin", "age": 19}
```

Load data from json file and show the schema

```
df = sqlContext.read.json("people.json")  
df.show()
```

```
+----+-----+  
| age| name |  
+----+-----+  
| null| Michael|  
| 30 | Andy |  
| 19 | Justin |  
+----+-----+
```

```
df.printSchema()
```

```
root  
|-- age: long (nullable = true)  
|-- name: string (nullable = true)
```

DataFrames

```
: df.select("name").show()
```

```
+----+  
| name|  
+----+  
|Michael|  
| Andy|  
| Justin|  
+----+
```

```
: df.select(df['name'], df['age'] + 1).show()
```

```
+-----+  
| name|(age + 1)|  
+-----+  
|Michael|      null|  
| Andy|       31|  
| Justin|      20|  
+-----+
```

```
: df.filter(df['age'] > 21).show()
```

```
+----+  
|age|name|  
+----+  
| 30|Andy|  
+----+
```

```
: df.groupBy("age").count().show()
```

```
+----+  
| age|count|  
+----+  
|null|     1|  
| 19|     1|  
| 30|     1|  
+----+
```

DataFrames

```
people.txt'
```

```
Michael, 29  
Andy, 30  
Justin, 19
```

Load data from a text file

Load data into a PythonRDD and then transform an PythonRDD to an DataFrame

```
from pyspark.sql import Row  
# Load a text file and convert each line to a Row.  
lines = sc.textFile("people.txt")  
parts = lines.map(lambda l: l.split(","))  
people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))  
people
```

PythonRDD[152] at RDD at PythonRDD.scala:43 [Row(age=29, name=u'Michael'), Row(age=30, name=u'Andy'), Row(age=19, name=u'Justin')]

```
schemaPeople = sqlContext.createDataFrame(people)  
# schemaPeople = people.toDF() is another way to create DataFrame  
schemaPeople
```

```
DataFrame[age: bigint, name: string] [Row(age=29, name=u'Michael'),  
Row(age=30, name=u'Andy'),  
Row(age=19, name=u'Justin')]
```

DataFrames

- Register the DataFrame as a table and then run SQL queries

```
# Infer the schema, and register the DataFrame as a table.
schemaPeople = sqlContext.createDataFrame(people)
schemaPeople.registerTempTable("people")

# SQL can be run over DataFrames that have been registered as a table.
teenagers = sqlContext.sql("SELECT name FROM people WHERE age >= 13 AND age <= 19")

# The results of SQL queries are RDDs and support all the normal RDD operations.
teenNames = teenagers.map(lambda p: "Name: " + p.name)
for teenName in teenNames.collect():
    print teenName
```

Name: Justin

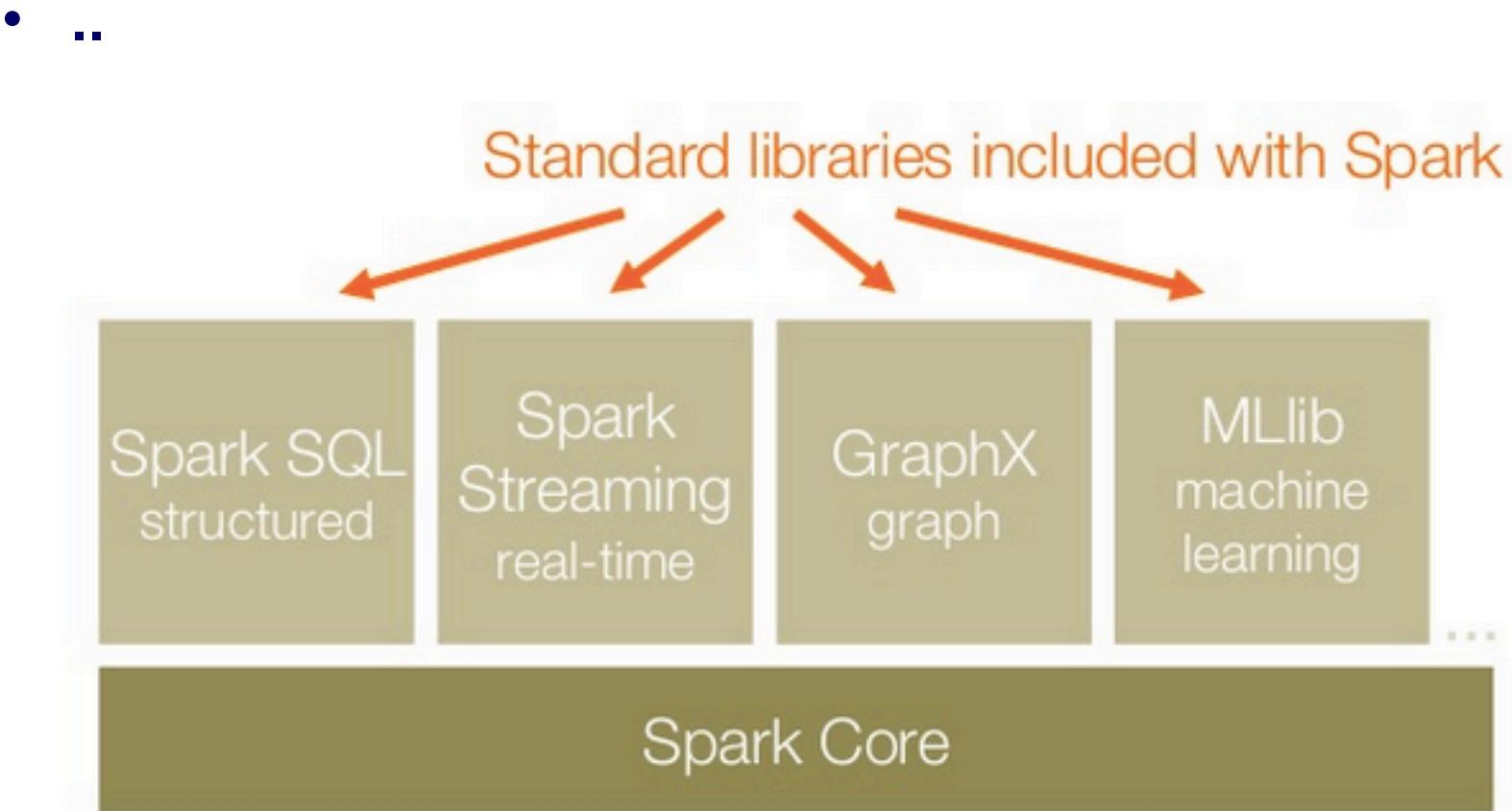
Part 3: Machine Learning in Spark

- **Data Frames**
- **MLLib**
- **Write your own algos**
 - Linear regression (Ridge and Lasso)
 - Logistic Regression
- **Pipelines**
- **R and Spark**
 - Linear Regression example
- **Graphs in Spark**
- **Case studies**
 - Flight delay
 - Mobile advertising
 - Social Networks

- **Machine Learning in Spark**

- Mllib
- Write your own!

Spark Machine Learning



Spark MLLib

MLlib is Apache Spark's scalable machine learning library

- linear SVM and logistic regression
- classification and regression tree
- multinomial naive Bayes
- random forest and gradient-boosted trees
- linear regression with L1- and L2-regularization
- isotonic regression
- recommendation via alternating least squares
- clustering via k-means, Gaussian mixtures, and power iteration clustering
- topic modeling via latent Dirichlet allocation
- singular value decomposition
- frequent itemset mining via FP-growth
- basic statistics
- feature transformations

-
- ...

Machine Learning Library (MLlib)

```
points = context.sql("select latitude, longitude from tweets")
model = KMeans.train(points, 10)
```

- **MLlib is Apache Spark's scalable machine learning library.**

<https://spark.apache.org/mllib/>

Algorithms

MLlib 1.3 contains the following algorithms:

- linear SVM and logistic regression
- classification and regression tree
- random forest and gradient-boosted trees
- recommendation via alternating least squares
- clustering via k-means, Gaussian mixtures, and power iteration clustering
- topic modeling via latent Dirichlet allocation
- singular value decomposition
- linear regression with L₁- and L₂-regularization
- isotonic regression
- multinomial naive Bayes
- frequent itemset mining via FP-growth
- basic statistics
- feature transformations

-
- Data types
 - Basic statistics
 - summary statistics
 - correlations
 - stratified sampling
 - hypothesis testing
 - random data generation
 - Classification and regression
 - linear models (SVMs, logistic regression, linear regression)
 - naive Bayes
 - decision trees
 - ensembles of trees (Random Forests and Gradient-Boosted Trees)
 - isotonic regression
 - Collaborative filtering
 - alternating least squares (ALS)
 - Clustering
 - k-means
 - Gaussian mixture
 - power iteration clustering (PIC)
 - latent Dirichlet allocation (LDA)
 - streaming k-means
 - Dimensionality reduction
 - singular value decomposition (SVD)
 - principal component analysis (PCA)
 - Feature extraction and transformation
 - Frequent pattern mining
 - FP-growth
 - Optimization (developer)
 - stochastic gradient descent
 - limited-memory BFGS (L-BFGS)
 - PMML model export

Kmeans in MLLib

The following examples can be tested in the PySpark shell.

In the following example after loading and parsing data, we use the KMeans object to cluster the data into two clusters. The number of desired clusters is passed to the algorithm. We then compute Within Set Sum of Squared Error (WSSSE). You can reduce this error measure by increasing k . In fact the optimal k is usually one where there is an "elbow" in the WSSSE graph.

```
from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array
from math import sqrt

# Load and parse the data
data = sc.textFile("data/mllib/kmeans_data.txt")
parsedData = data.map(lambda line: array([float(x) for x in line.split(' ')]))

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 2, maxIterations=10,
    runs=10, initializationMode="random")

# Evaluate clustering by computing Within Set Sum of Squared Errors
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))

WSSSE = parsedData.map(lambda point: error(point)).reduce(lambda x, y: x + y)
print("Within Set Sum of Squared Error = " + str(WSSSE))

# Save and load model
clusters.save(sc, "myModelPath")
sameModel = KMeansModel.load(sc, "myModelPath")
```

Evaluate clustering by
computing Within Set Sum of
Squared Errors
Assign point to cluster centre

Classification

The example below demonstrates how to load a [LIBSVM data file](#), parse it as an RDD of `LabeledPoint` and then perform classification using a Random Forest. The test error is calculated to measure the algorithm accuracy.

Scala Java

Python

Classification using Random Forests

```
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils

# Load and parse the data file into an RDD of LabeledPoint.
data = MLUtils.loadLibSVMFile(sc, 'data/mllib/sample_libsvm_data.txt')
# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a RandomForest model.
# Empty categoricalFeaturesInfo indicates all features are continuous.
# Note: Use larger numTrees in practice.
# Setting featureSubsetStrategy="auto" lets the algorithm choose.
model = RandomForest.trainClassifier(trainingData, numClasses=2, categoricalFeaturesInfo={},
                                      numTrees=3, featureSubsetStrategy="auto",
                                      impurity='gini', maxDepth=4, maxBins=32)

# Evaluate model on test instances and compute test error
predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() / float(testData.count())
print('Test Error = ' + str(testErr))
print('Learned classification forest model:')
print(model.toDebugString())

# Save and load model
model.save(sc, "myModelPath")
sameModel = RandomForestModel.load(sc, "myModelPath")
```

Gradient Boosted DTs

Gradient-Boosted Trees (GBTs) are ensembles of [decision trees](#). GBTs iteratively train decision trees in order to minimize a loss function. Like decision trees, GBTs handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions.

MLlib supports GBTs for binary classification and for regression, using both continuous and categorical features. MLlib implements GBTs using the existing [decision tree](#) implementation. Please see the decision tree guide for more information on trees.

Note: GBTs do not yet support multiclass classification. For multiclass problems, please use [decision trees](#) or [Random Forests](#).

Basic algorithm

Gradient boosting iteratively trains a sequence of decision trees. On each iteration, the algorithm uses the current ensemble to predict the label of each training instance and then compares the prediction with the true label. The dataset is re-labeled to put more emphasis on training instances with poor predictions. Thus, in the next iteration, the decision tree will help correct for previous mistakes.

The specific mechanism for re-labeling instances is defined by a loss function (discussed below). With each iteration, GBTs further reduce this loss function on the training data.

Losses

The table below lists the losses currently supported by GBTs in MLlib. Note that each loss is applicable to one of classification or regression, not both.

Notation: N = number of instances. y_i = label of instance i . x_i = features of instance i . $F(x_i)$ = model's predicted label for instance i .

Loss	Task	Formula	Description
Log Loss	Classification	$2 \sum_{i=1}^N \log(1 + \exp(-2y_i F(x_i)))$	Twice binomial negative log likelihood.
Squared Error	Regression	$\sum_{i=1}^N (y_i - F(x_i))^2$	Also called L2 loss. Default loss for regression tasks.
Absolute Error	Regression	$\sum_{i=1}^N y_i - F(x_i) $	Also called L1 loss. Can be more robust to outliers than Squared Error.

The example below demonstrates how to load a [LIBSVM data file](#), parse it as an RDD of LabeledPoint and then perform classification using Gradient-Boosted Trees with log loss. The test error is calculated to measure the algorithm accuracy.

Scala Java **Python**

```
from pyspark.mllib.tree import GradientBoostedTrees, GradientBoostedTreesModel
from pyspark.mllib.util import MLUtils

# Load and parse the data file.
data = MLUtils.loadLibSVMFile(sc, "data/mllib/sample_libsvm_data.txt")
# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a GradientBoostedTrees model.
# Notes: (a) Empty categoricalFeaturesInfo indicates all features are continuous.
# (b) Use more iterations in practice.
model = GradientBoostedTrees.trainClassifier(trainingData,
    categoricalFeaturesInfo={}, numIterations=3)

# Evaluate model on test instances and compute test error
predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() / float(testData.count())
print('Test Error = ' + str(testErr))
print('Learned classification GBT model:')
print(model.toDebugString())

# Save and load model
model.save(sc, "myModelPath")
sameModel = GradientBoostedTreesModel.load(sc, "myModelPath")
```

Part 3: Machine Learning in Spark

- **Data Frames**
- **MLLib**
- **Write your own algos**
 - Linear regression (Ridge and Lasso)
 - Logistic Regression
- **Pipelines**
- **R and Spark**
 - Linear Regression example
- **Graphs in Spark**
- **Case studies**
 - Flight delay
 - Mobile advertising
 - Social Networks

Linear Regression via Gradient Descent (a simpler root finder)

Given: Minimize $f(x)$

STEP 1: Find the zeros of the gradient function $f'(x)$

- Using Bisection method; OR Newton-Raphson; OR Gradient Descent



$$J_q(W, X_1^m) = \text{Minimize} \sum_{i=1}^m (W^T X_i - y_i)^2$$

$RSS = \text{Variance of } \varepsilon$

$$0 = \frac{\partial \sum \hat{\varepsilon}_i^2}{\partial W} = \frac{\partial \left(\sum_{j=1}^n (X_j W - y_i)^2 \right)}{\partial W}$$

Gradient vector of partial derivatives

$$\nabla J(W) = \left(\sum_{j=1}^n (X_j W - y_i) X_j \right)$$

Pull model closer to examples with biggest residual

$$W^{t+1} = W^t - \alpha^i \left(\sum_{j=1}^n (X_j W^t - y_i) X_j \right)$$

Gradient Descent

For another derivation see:

<http://www.stanford.edu/class/cs229/notes/cs229-notes1.pdf>

OLS Via Gradient Descent: The Gradient

$$W_{j,t+1} = W_{j,t} - \alpha * \nabla J_{w_j}(W_{j,t})$$

- In order to implement this algorithm, we have to work out what is the partial derivative term at time t on the right hand side $\nabla f_{w_j}(W) = dF(W)/dw_j$.
- Assume we have only one training example (x, y) , so that we can drop the sum in the definition of J .

$$\nabla J_{w_j}(W) = \frac{\partial}{\partial w_j} J(W) = \frac{\partial}{\partial w_j} \left(\frac{1}{2} (f_W(x) - y)^2 \right) \quad \text{Use chain rule } df/du * du/dx$$

$$= 2 * \frac{1}{2} (f_W(x) - y) \frac{\partial}{\partial w_j} (f_W(x) - y)$$

$$= (f_W(x) - y) \frac{\partial}{\partial w_j} \left(\left(\sum_{i=0}^n w_i x_i \right) - y \right)$$

$$\nabla J_{w_j}(W) = (f_W(x) - y) x_j$$

Assume a single
training example
For a single w_j

For all training data
Across a single w_j

$$\nabla J(W) = \left(\sum_{j=0}^n (f_W(X_j) - y) X_j \right)$$

$$\nabla J(W) = \left(\sum_{j=0}^n (WX_j - y) X_j \right)$$

Recall

$$\begin{aligned} \frac{\partial}{\partial w_j} \left(\left(\sum_{i=0}^n w_i x_i \right) - y \right) &= \frac{\partial}{\partial w_j} (w_0 x_0 + w_1 x_1 + \dots + w_j x_j + \dots + w_n x_n) \\ &= 0 + 0 + \dots + x_j + \dots + 0 \end{aligned}$$

OLS using Gradient Descent

$$J_q(W, X_1^m) = \text{Minimize} \frac{1}{2m} \sum_{i=1}^m (W^T X_i - y_i)^2$$

**OLS Objective Function
With decision variables W**

- Initialize $W = \text{vector or zeros}$
- Repeat until Convergence

$$W^{t+1} = W^t - \alpha^i \left(\sum_{j=1}^n (X_j W^t - y_i) X_j \right)$$

OLS Batch Update Rule

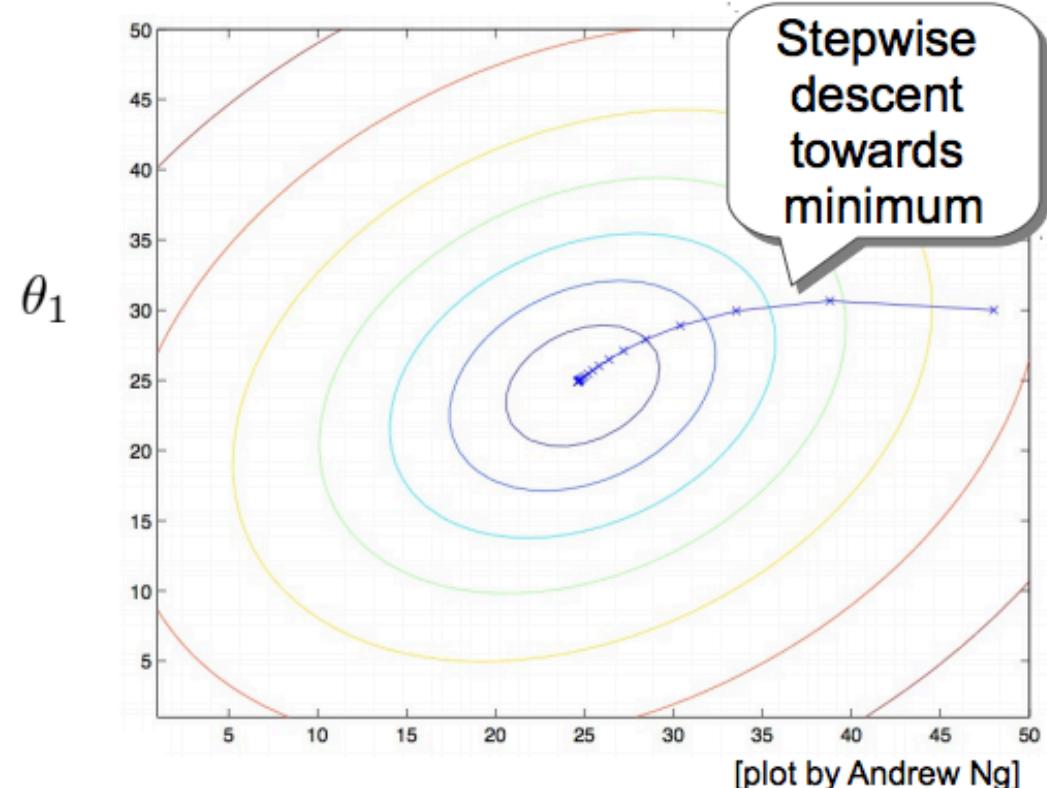
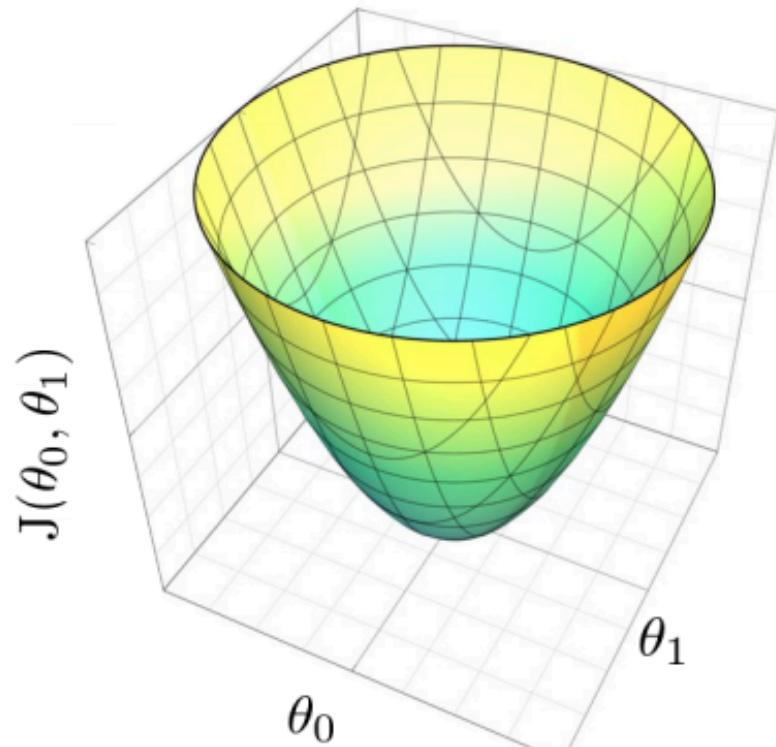
- End Repeat

True gradient is approximated by the gradient of the cost function only evaluated at all examples; adjust parameters proportional to this approx. gradient.

Intuitively, drag weight vector closer to the incorrectly predicted examples

3D plots and contour plots

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m ((\theta_0 + \theta_1 x^{(i)}) - y^{(i)})^2$$



[plot by Andrew Ng]

$$\underset{\theta_0, \theta_1}{\text{minimize}} J(\theta_0, \theta_1) = \arg \min_{\theta} J(\theta)$$

OLS via Gradient Descent

Hypothesis: $h_{\theta}(x) = \theta_0 + \theta_1 x$

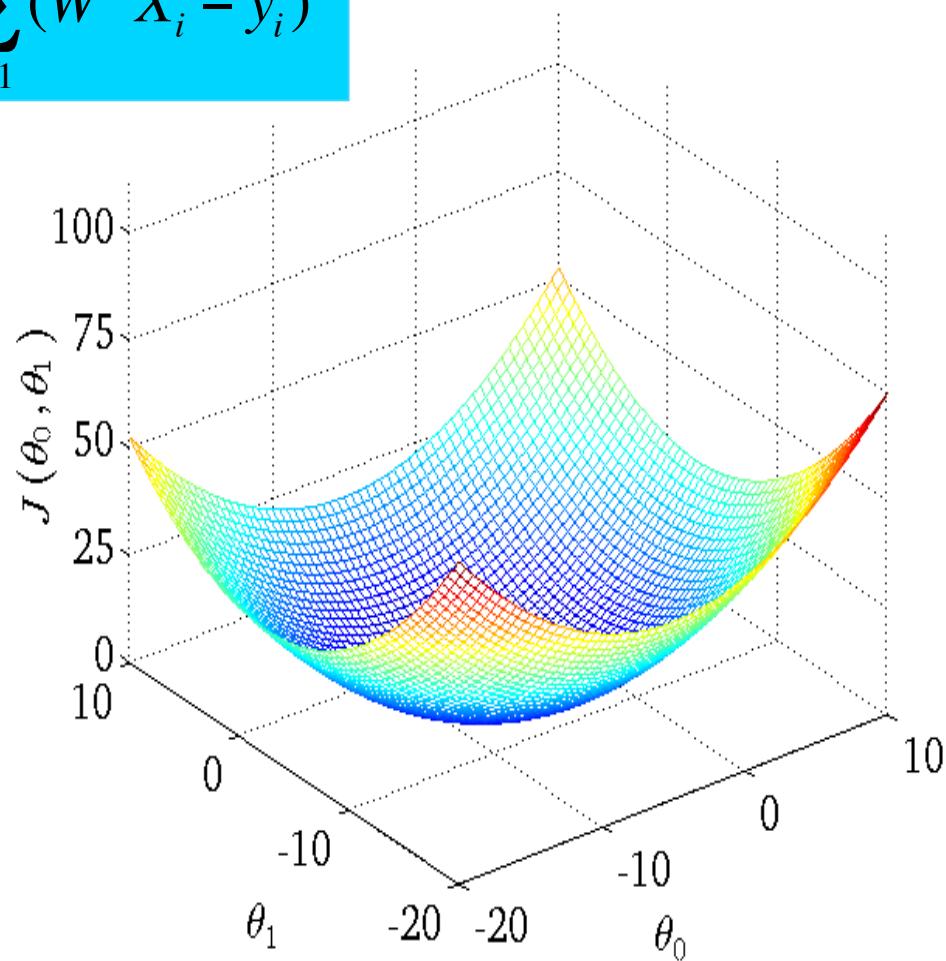
Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}} J(\theta_0, \theta_1)$

Error Surface for OLS: Price~Size

$$J_q(W, X_1^L) = \frac{1}{m} \sum_{i=1}^m (W^T X_i - y_i)^2$$

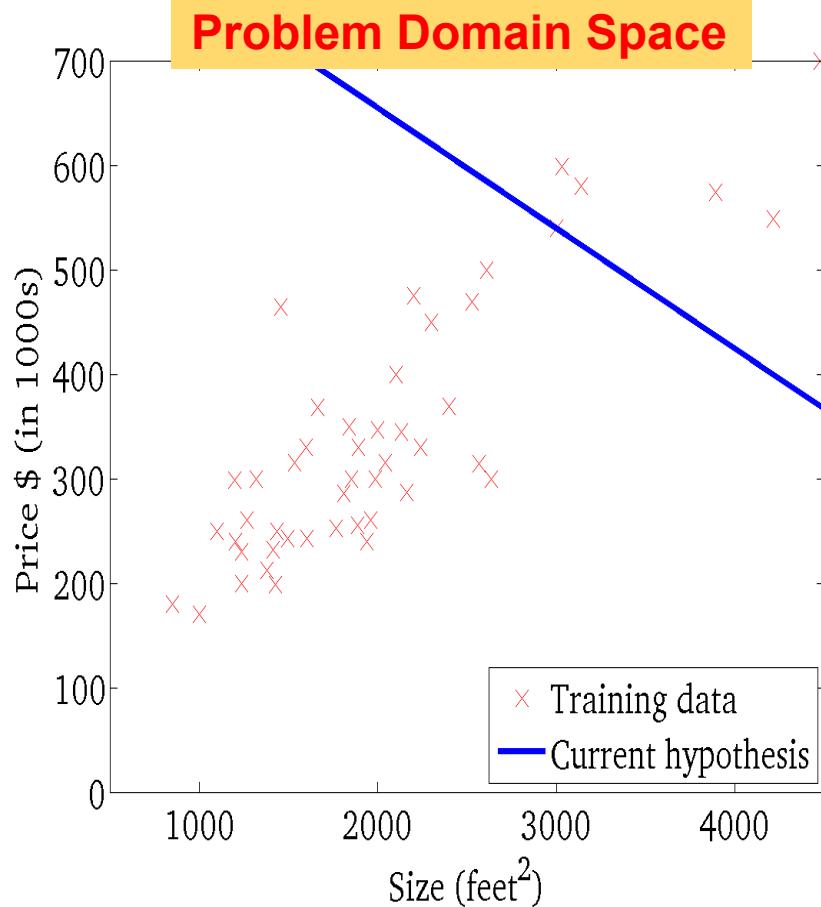


Gradient Descent for LR Price~Size Iteration 0

Eqn Line $y = aX + b$

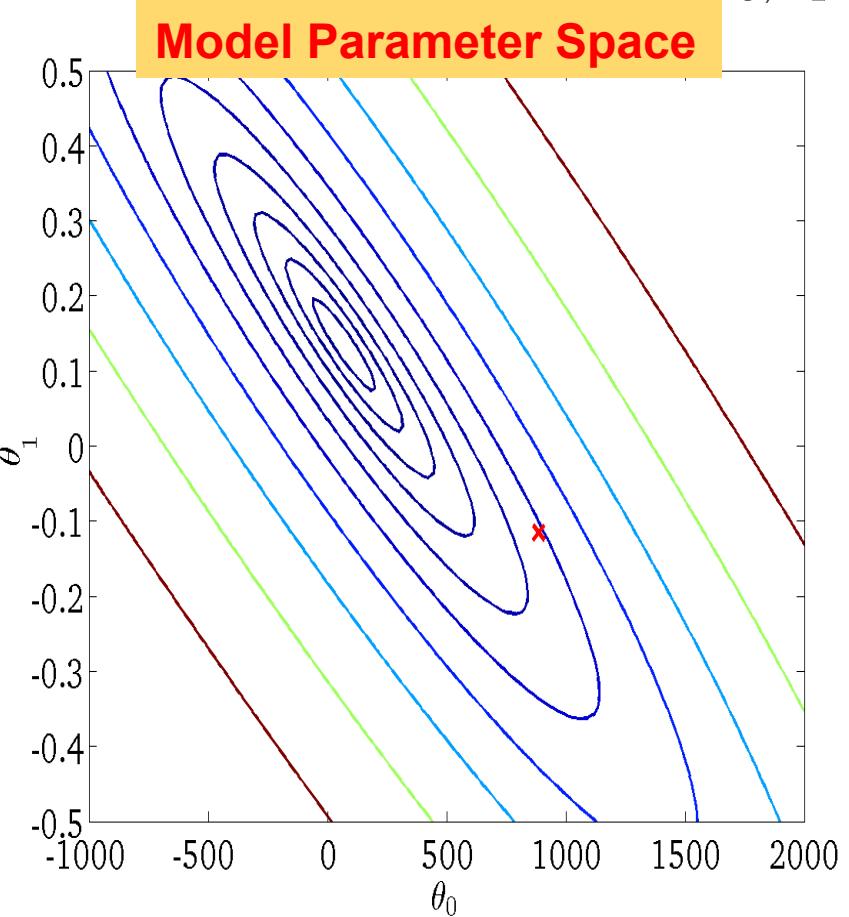
$$Y = w_1 X + w_0 \quad 1$$

(for fixed θ_0, θ_1 , this is a function of x)



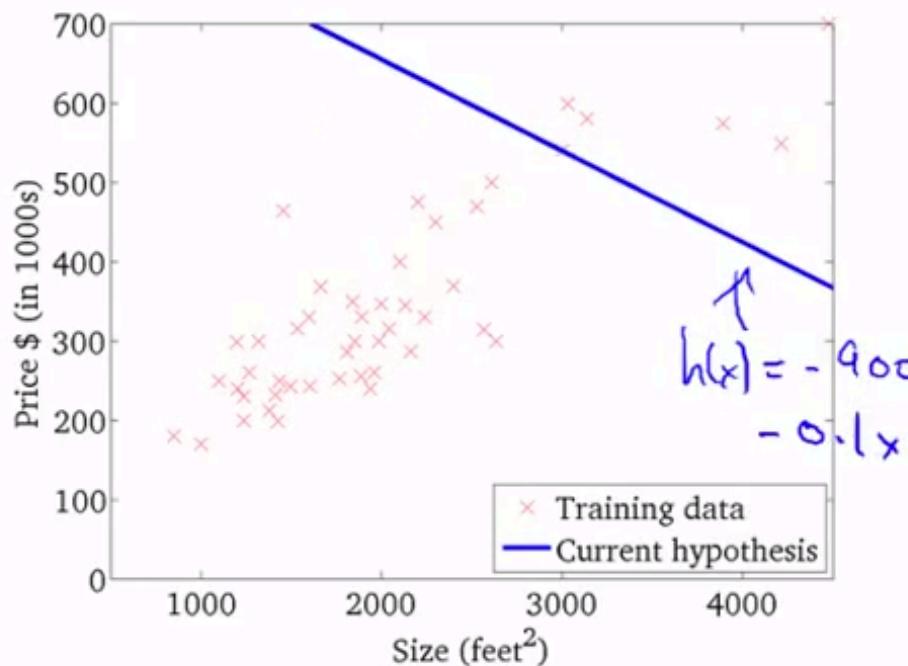
$$J(\theta_0, \theta_1) \quad J_q(W, X^L) = \frac{1}{m} \sum_{i=1}^m (W^T X_i - y_i)^2$$

(function of the parameters θ_0, θ_1)

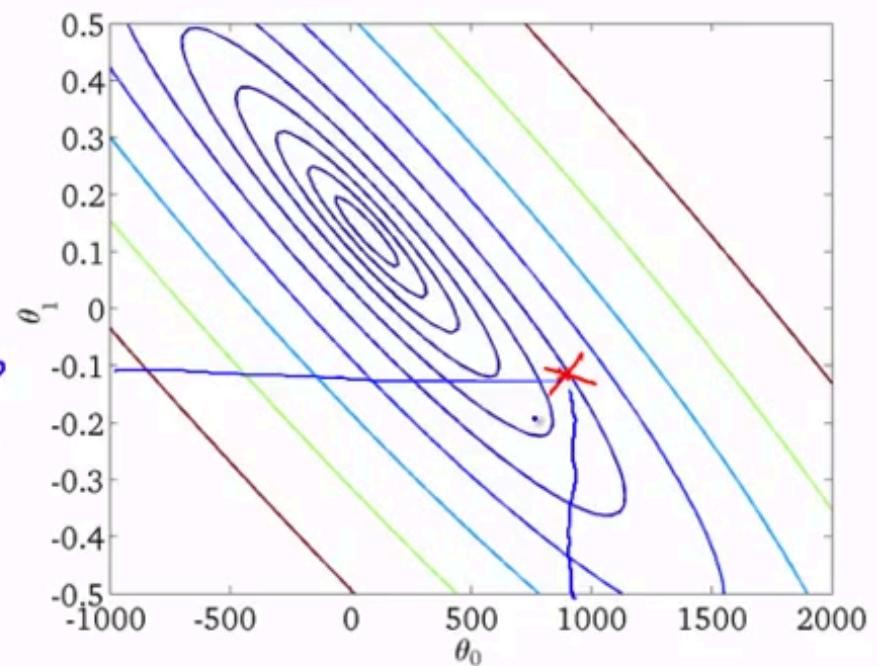


Gradient Descent for LR Price~Size – Iteration 0

$\underline{h_\theta(x)}$
(for fixed θ_0, θ_1 , this is a function of x)



$\underline{J(\theta_0, \theta_1)}$
(function of the parameters θ_0, θ_1)



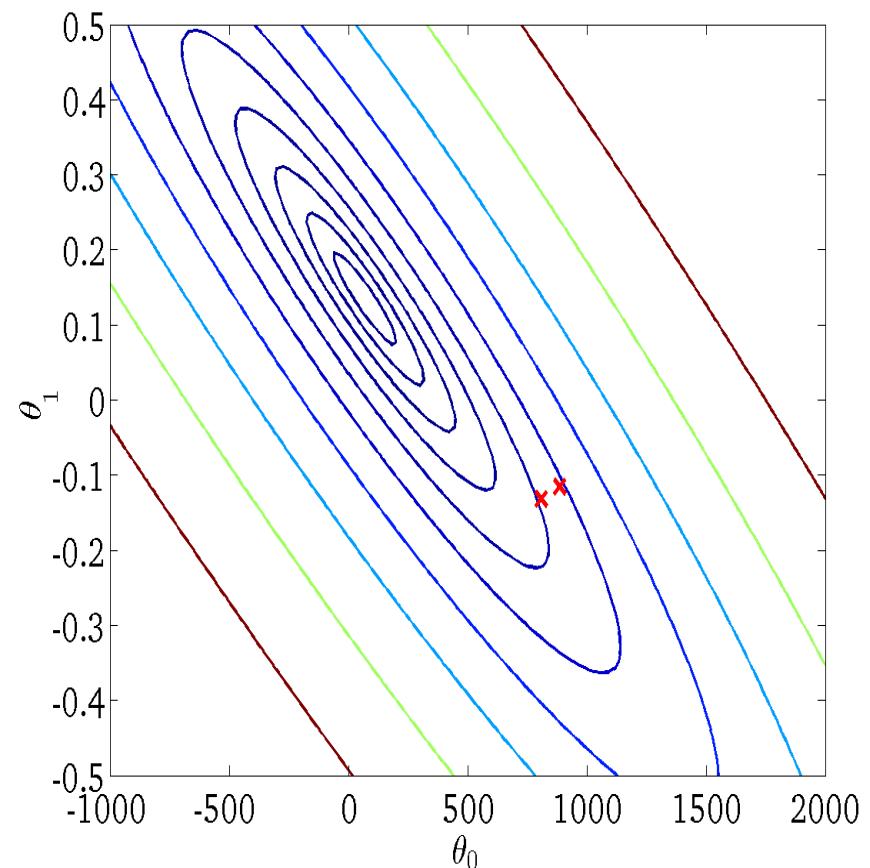
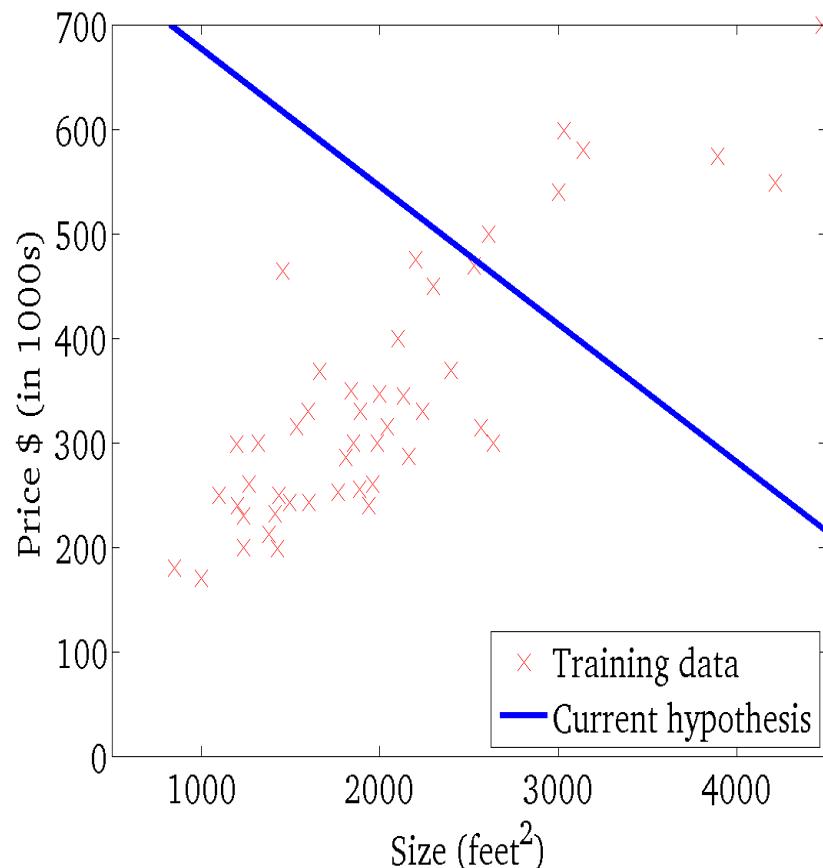
Gradient Descent for LR Price~Size – Iteration 1

$$h_{\theta}(x)$$

(for fixed θ_0, θ_1 , this is a function of x)

$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



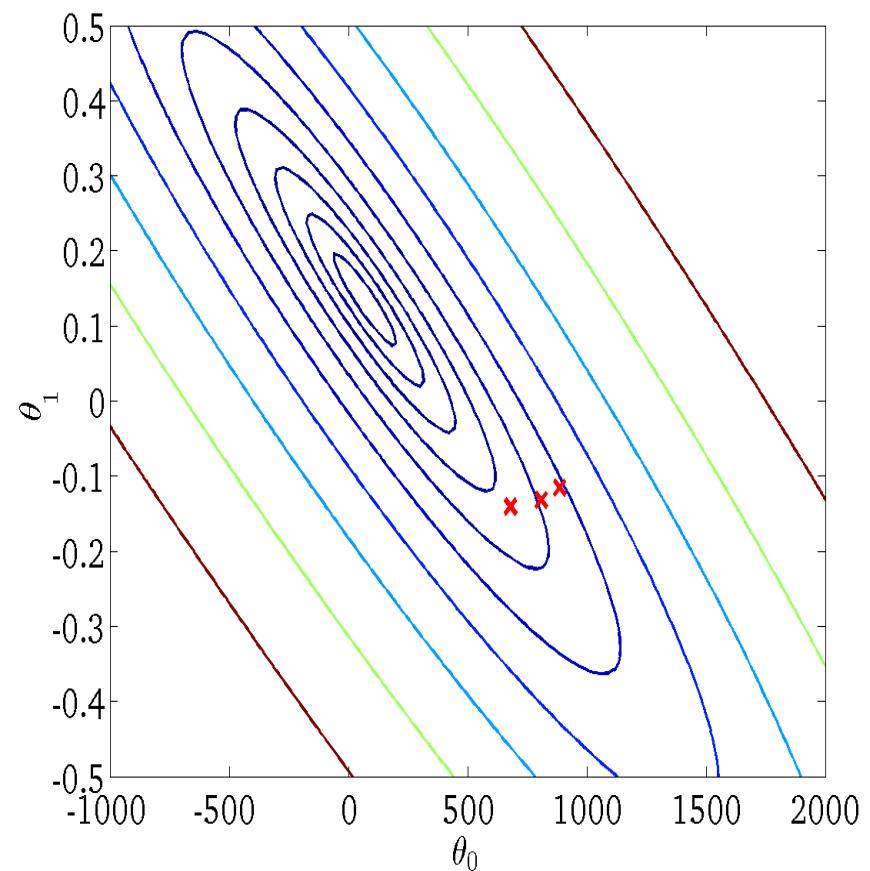
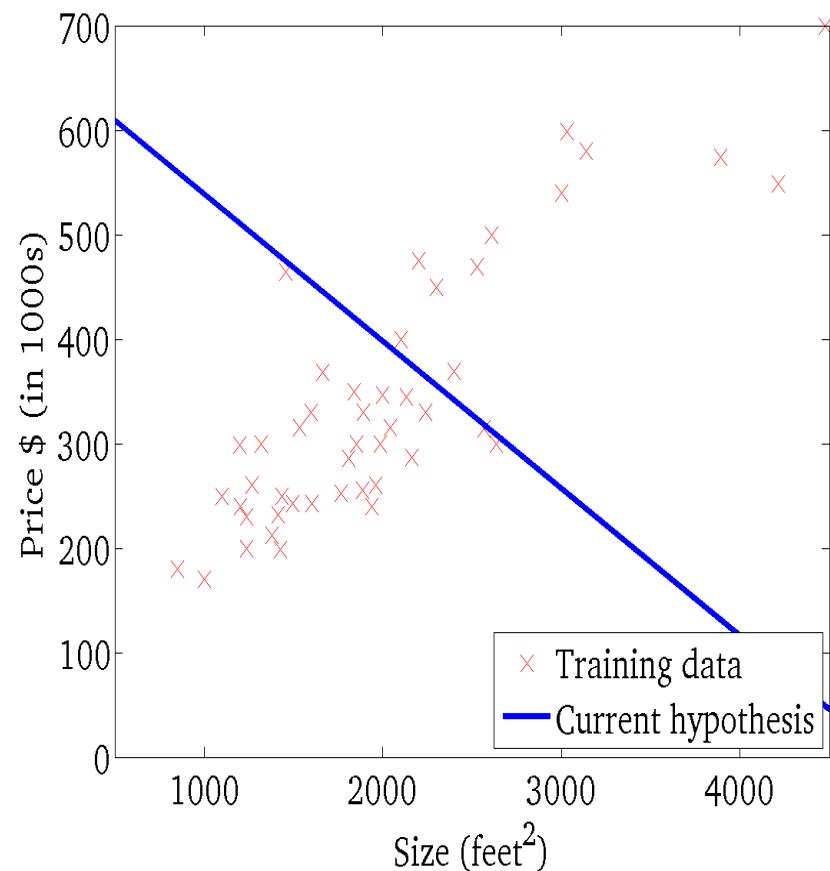
Gradient Descent for LR Price~Size – Iteration 2

$$h_{\theta}(x)$$

(for fixed θ_0, θ_1 , this is a function of x)

$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



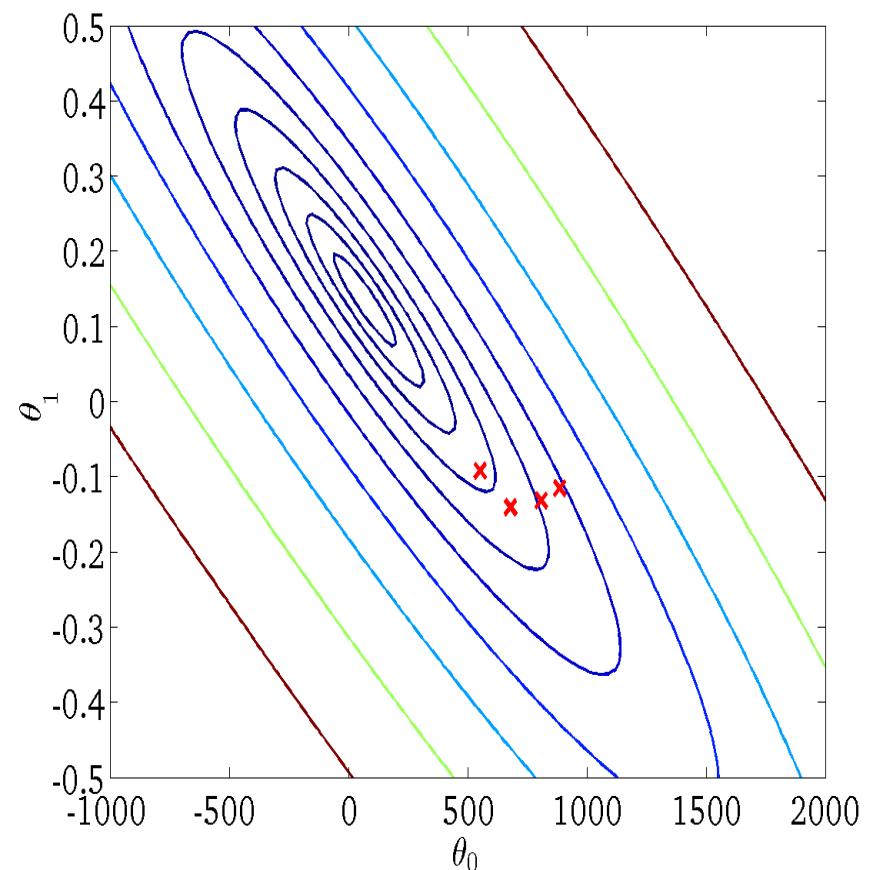
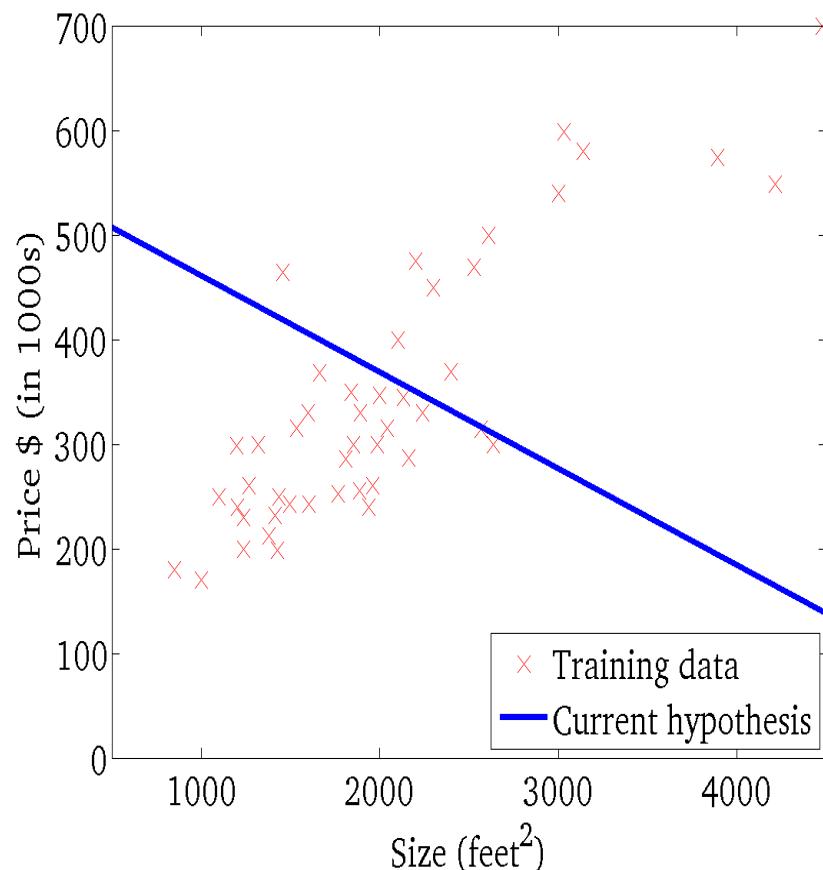
Gradient Descent for LR Price~Size – Iteration 3

$$h_{\theta}(x)$$

(for fixed θ_0, θ_1 , this is a function of x)

$$J(\theta_0, \theta_1)$$

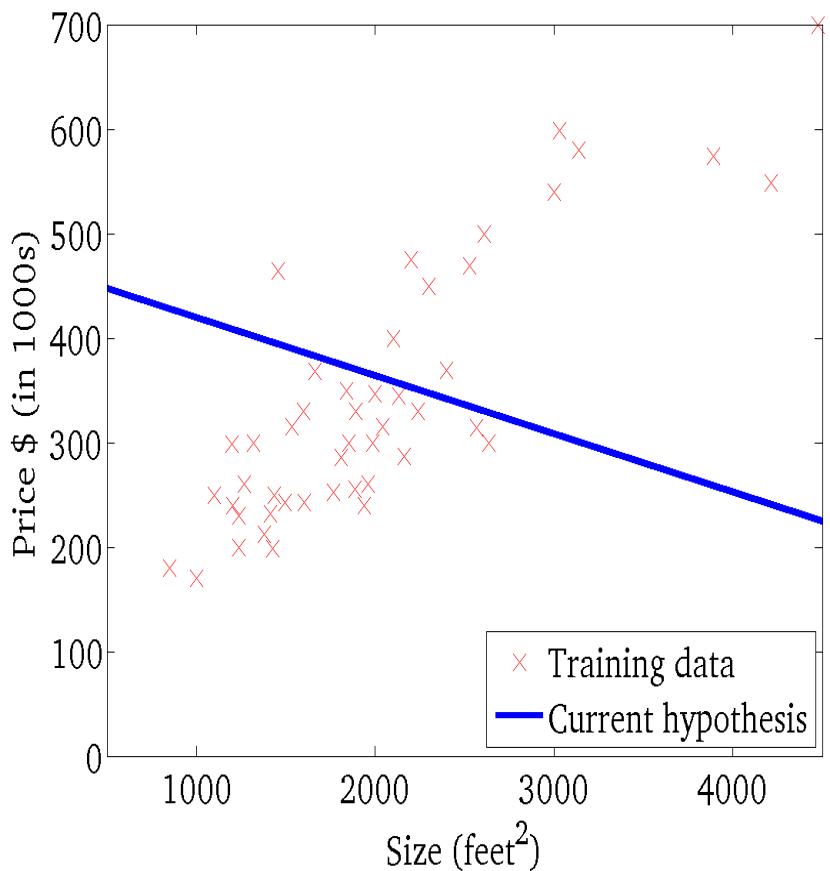
(function of the parameters θ_0, θ_1)



Gradient Descent for LR Price~Size – Iteration 4

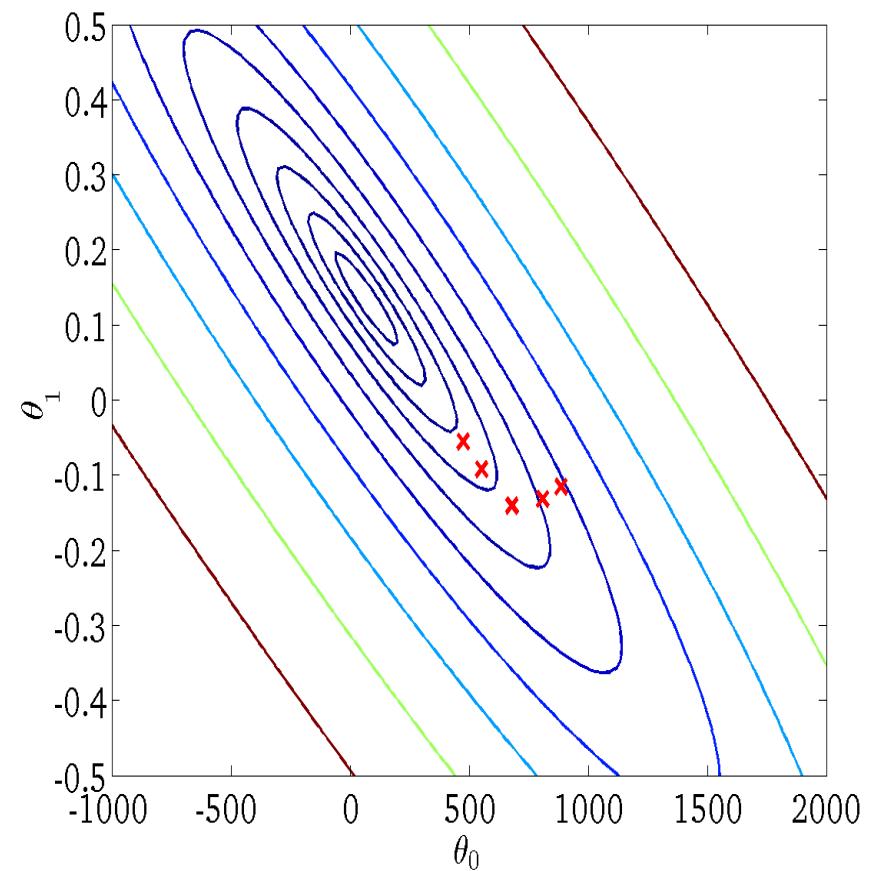
$$h_{\theta}(x)$$

(for fixed θ_0, θ_1 , this is a function of x)



$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



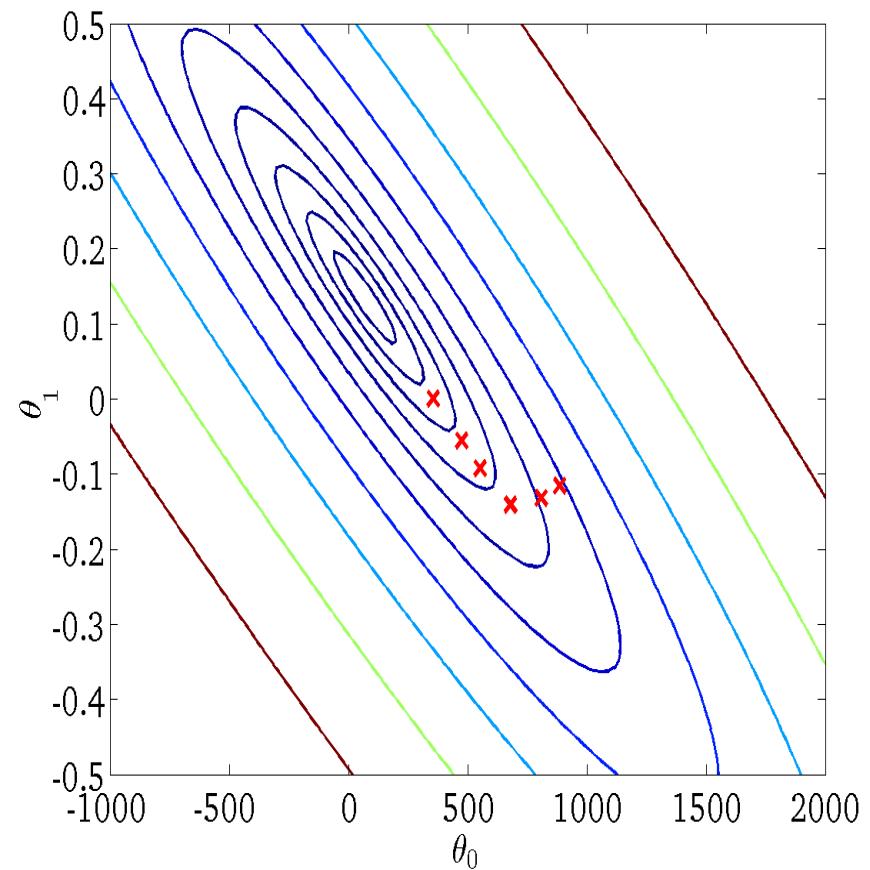
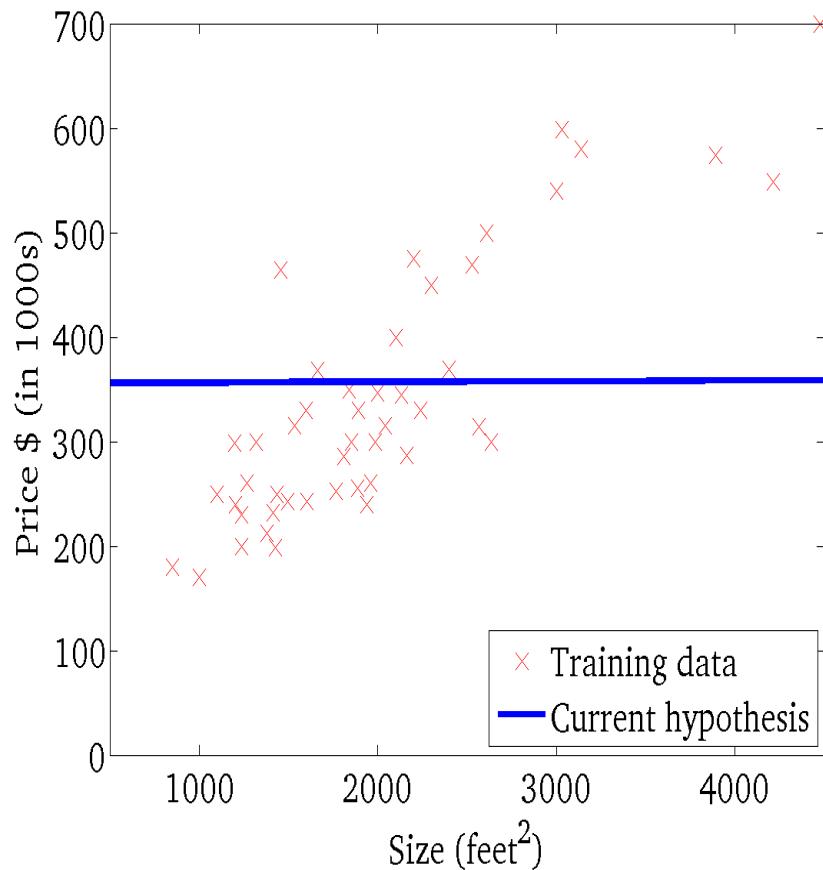
Gradient Descent for LR Price~Size – Iteration 5

$$h_{\theta}(x)$$

(for fixed θ_0, θ_1 , this is a function of x)

$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



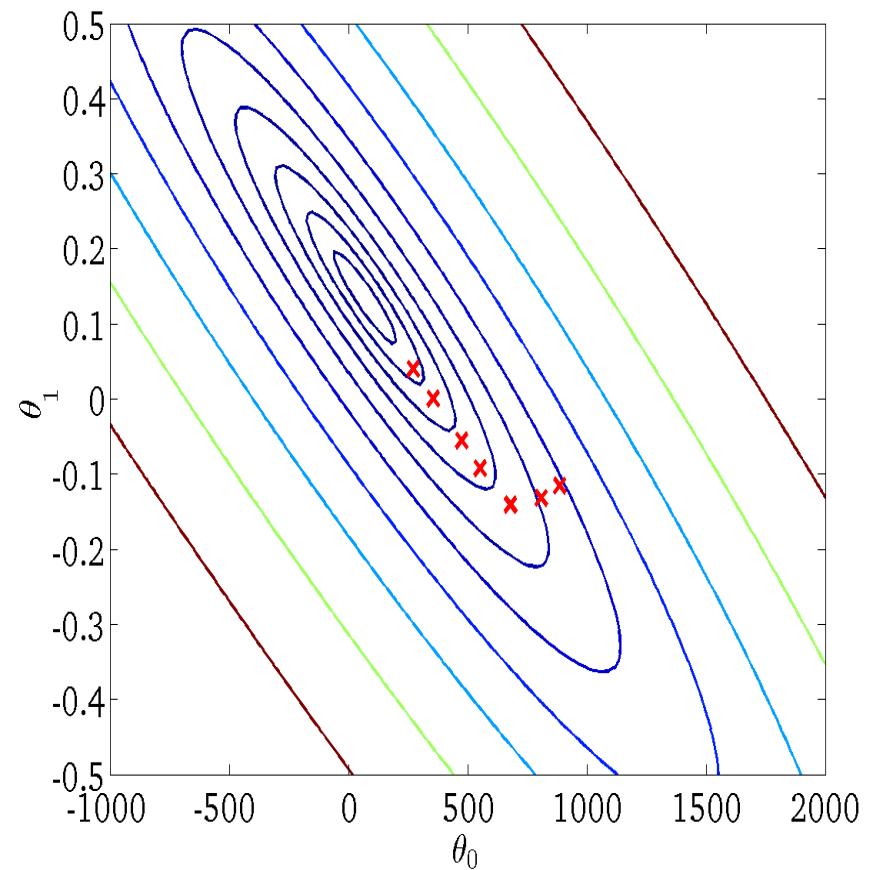
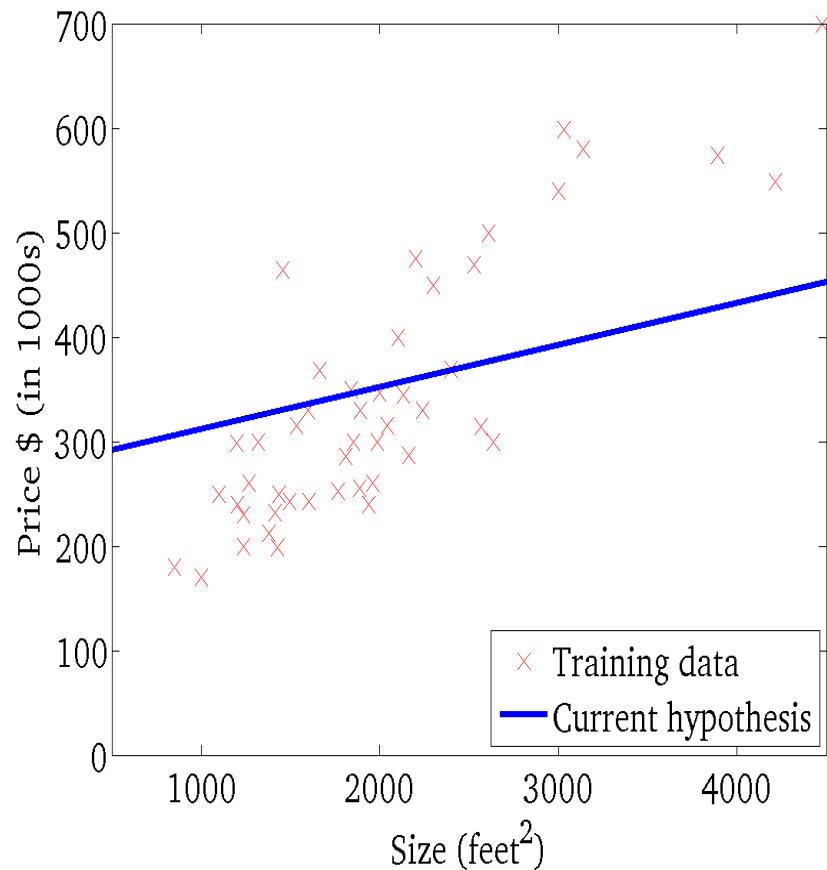
Gradient Descent for LR Price~Size – Iteration 6

$$h_{\theta}(x)$$

(for fixed θ_0, θ_1 , this is a function of x)

$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



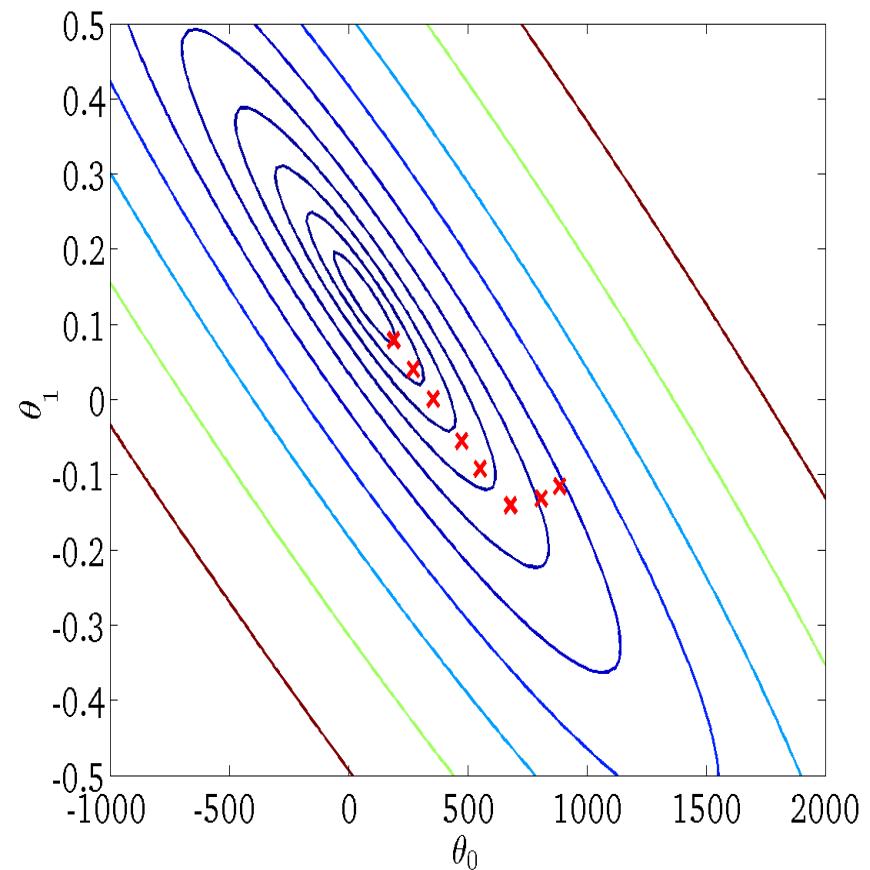
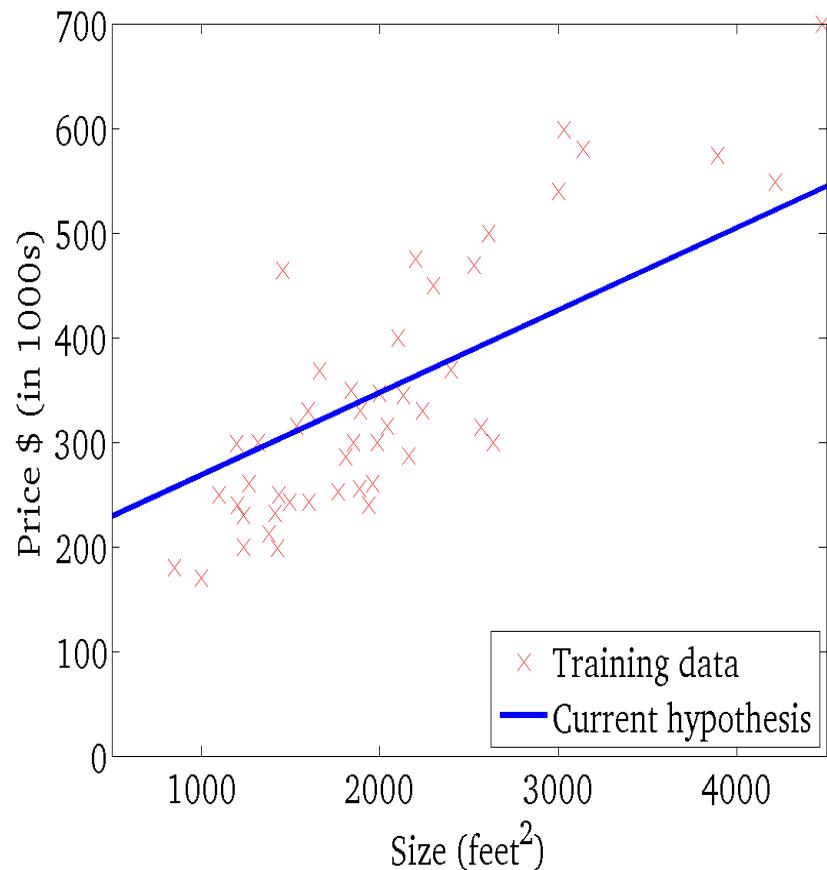
Gradient Descent for LR Price~Size – Iteration 7

$$h_{\theta}(x)$$

(for fixed θ_0, θ_1 , this is a function of x)

$$J(\theta_0, \theta_1)$$

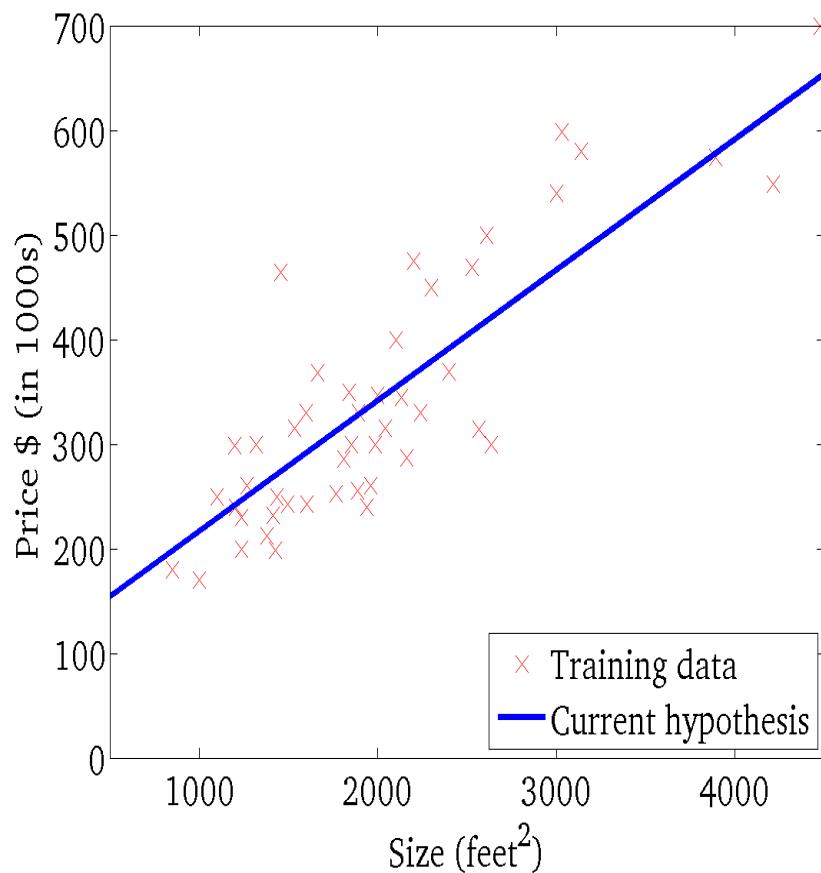
(function of the parameters θ_0, θ_1)



Gradient Descent for LR Price~Size – Converged after 9 steps

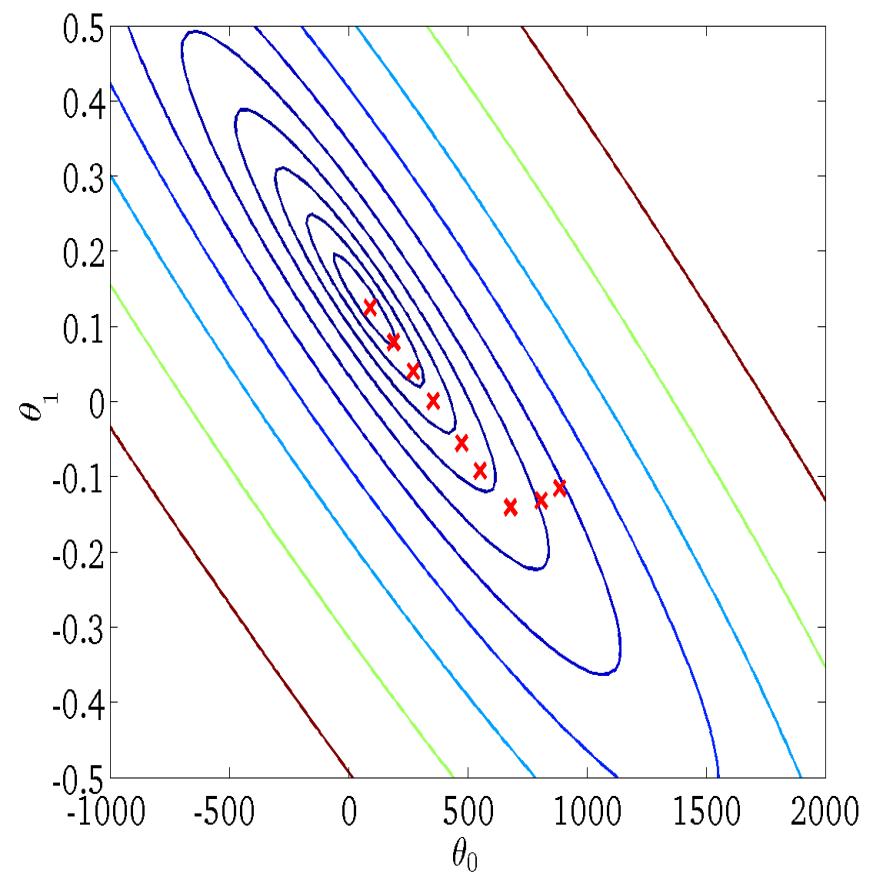
$$h_{\theta}(x)$$

(for fixed θ_0, θ_1 , this is a function of x)



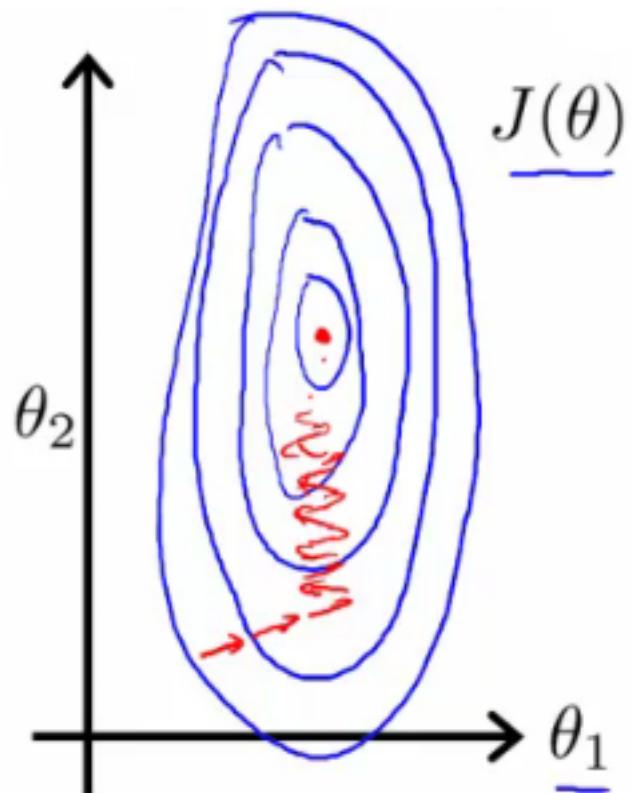
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



Sometimes we overshoot

Alpha our learning rate needs to be small enough to allow us to converge. Otherwise we will oscillate. ...can also decrease ALPHA over time... Gradient descent provides no guarantees to find the minimum but....



$RSS = \text{Variance of } \varepsilon$

$$0 = \frac{\partial \sum \hat{\varepsilon}_i^2}{\partial W} = \frac{\partial \left(\sum_{j=1}^n (X_j W - y_i)^2 \right)}{\partial W}$$

$$\nabla J(W) = \left(\sum_{j=1}^n (X_j W - y_i) X_j \right)$$

$$W^{t+1} = W^t - \alpha^i \left(\sum_{j=1}^n (X_j W^t - y_i) X_j \right)$$

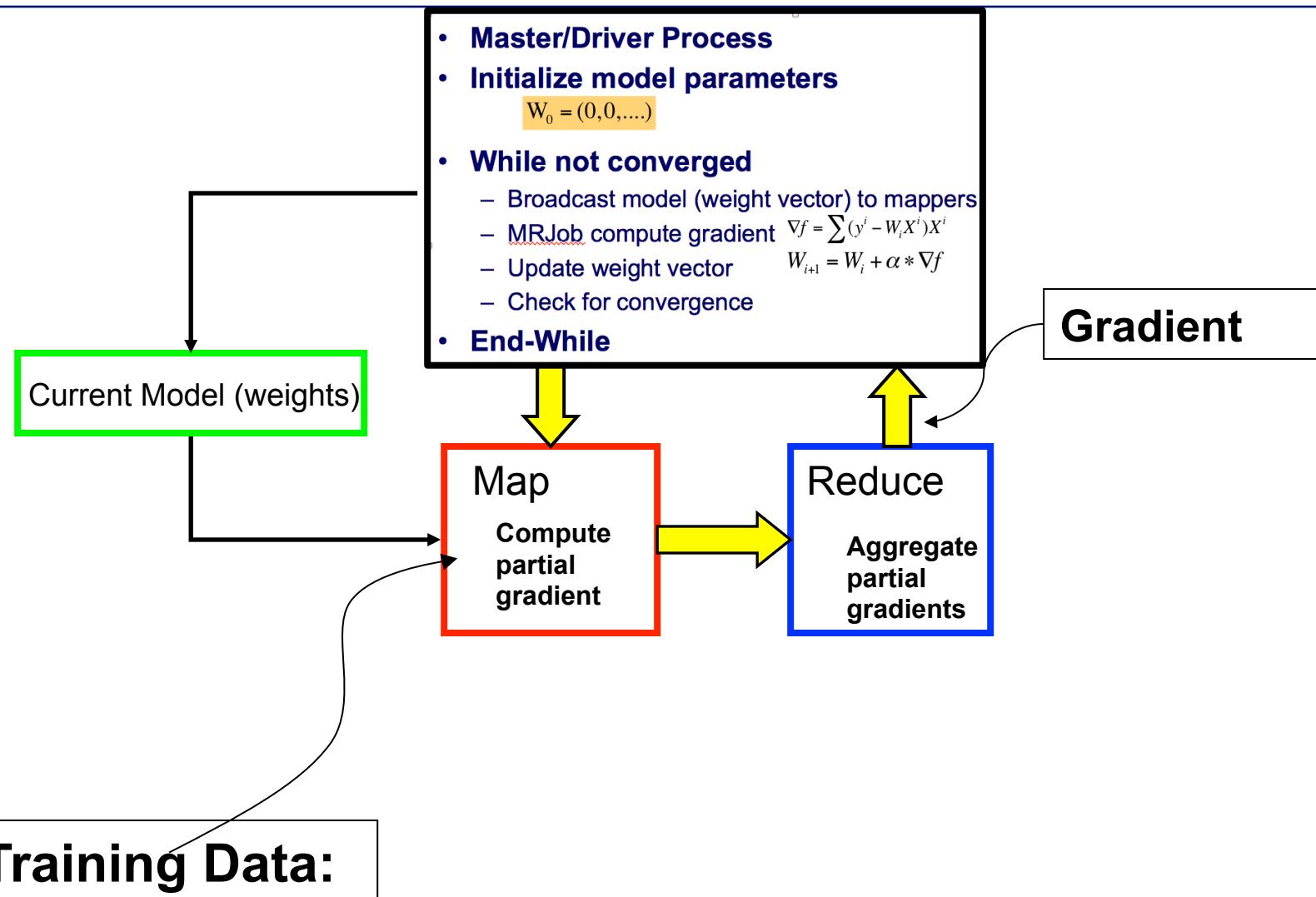
OLS via Distributed Gradient Descent

- **Master/Driver Process**
- **Initialize model parameters**

$$W_0 = (0, 0, \dots)$$

- **While not converged**
 - Broadcast model (weight vector) to mappers
 - MRJob compute gradient $\nabla f = \sum (y^i - W_i X^i) X^i$
 - Update weight vector $W_{i+1} = W_i + \alpha * \nabla f$
 - Check for convergence
- **End-While**

MapReduce Implementation



OLS via Distributed Gradient Descent

- **Master/Driver Process**
- **Initialize model parameters, W = vector of zeros**
 $W = (0,0,...)$
- **While not converged**
 - Broadcast model (e.g., weight vector) to the worker nodes
 - Mapper (MANY mappers)
 - Compute partial gradient for each training example
 - Combine in memory $\nabla f = \sum (y^i - W_i X^i) X^i$
 - Finally Yield the partial gradient
 - Reducer (single Reducer)
 - Aggregate partial gradients
 - Yield full gradient $\nabla f = \sum (y^i - W_i X^i) X^i$
 - Update weight vector $W_{i+1} = W_i + \alpha * \nabla f$
 - Check for convergence

End While

Linear Regression

Data Generation

```
import numpy as np
import csv
def data_generate(fileName, w=[0,0], size = 100):
    np.random.seed(0)
    x = np.random.uniform(-4, 4, size)
    noise = np.random.normal(0, 2, size)
    y = (x * w[0] + w[1] + noise)
    data = zip(y, x)
    with open(fileName, 'wb') as f:
        writer = csv.writer(f)
        for row in data:
            writer.writerow(row)
    return True
```

The true $y = 8x - 2$.

```
w = [8, -2]
data_generate('data.csv', w, 100)
```

True

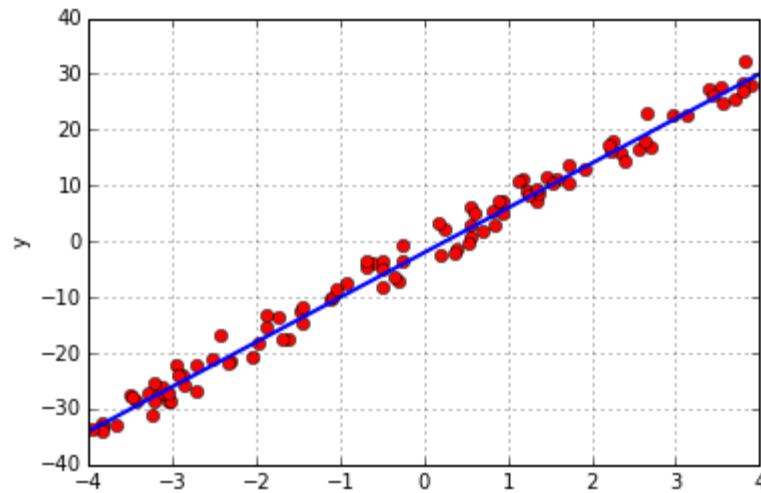
Data format: y, x

```
0.77075643829433071, 2.672964698525508, -0.80725624349150493
8.8154706781482624, 4.9760776507722362, 2.522486096383517
-9.8151480026146789, -0.02985975394819107, -5.9270633829692514
-18.969417561297352, -5.5040670893830468, -8.1264919044925392
10.600223695067733, -6.03874270480752, 9.5105193453227734
0.41965480983655812, 5.2106142439791743, -2.6531234332946063
-3.4052657239448485, -6.6177832687492906, 1.5256793588631936
-16.087150784240489, -8.2332037165197942, -5.0813586409076557
```

Data Visualization

```
%matplotlib inline
import matplotlib.pyplot as plt
def dataPlot(file, w):
    with open(file, 'r') as f:
        reader = csv.reader(f)
        for row in reader:
            plt.plot(float(row[1]), float(row[0]), 'o'+r')
    plt.xlabel("x")
    plt.ylabel("y")
    x = [-4, 4]
    y = [(i * w[0] + w[1]) for i in x]
    plt.plot(x,y, linewidth=2.0)
    plt.grid()
    plt.show()
```

```
dataPlot('data.csv',w)
```



Linear Regression

Objective Function

$$\min_w L(w) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i)^2$$

Gradient Descent

$$\frac{\partial L}{\partial w} = -\frac{2}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i) \cdot x_i$$

Gradient descent (no regularization)

```
: from collections import namedtuple
import numpy as np
Point = namedtuple('Point', 'x y')

def readPoint(line):
    d = line.split(',')
    x = [float(i) for i in d[1:]]
    x.append(1.0) #bias term
    return Point(x, float(d[0]))

wBroadcast = sc.broadcast(w)
gradient = data.map(lambda p: -2 * (p.y - np.dot(wBroadcast.value, p.x)) * np.array(p.x))
    .reduce(lambda a, b: a + b)
w = w - learningRate * gradient/n

for i in range(iterations):
    wBroadcast = sc.broadcast(w)
    gradient = data.map(lambda p: -2 * (p.y - np.dot(wBroadcast.value, p.x)) * np.array(p.x)) \
        .reduce(lambda a, b: a + b) Sum up gradients
    w = w - learningRate * gradient/n
return w

: data = sc.textFile('data.csv').map(readPoint).cache()
linearRegressionGD(data)

: array([ 7.98418741, -1.61969498])
```

Get gradients for each instance

Sum up gradients

Linear Regression

Plot w in iterations

```
def ierationsPlot(fileName, truew):
    x = [-4, 4]

    w = truew
    y = [(i * w[0] + w[1]) for i in x]
    plt.plot(x, y, 'b', label="True line", linewidth=4.0)

    np.random.seed(400)
    w = np.random.normal(0,1,2)
    y = [(i * w[0] + w[1]) for i in x]
    plt.plot(x, y, 'r--', label="After 0 Iterations", linewidth=2.0)

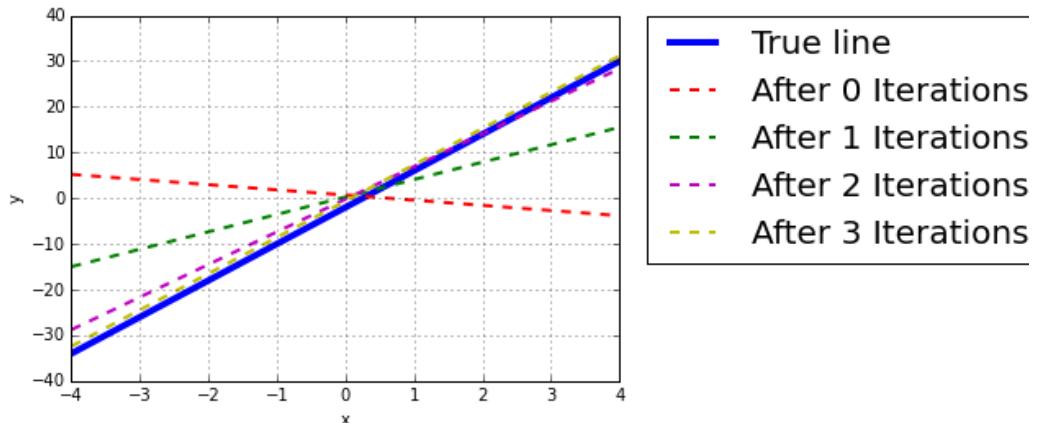
    data = sc.textFile(fileName).map(readPoint).cache()
    w = linearRegressionGD(data, w, iterations=1)
    y = [(i * w[0] + w[1]) for i in x]
    plt.plot(x, y, 'g--', label="After 1 Iterations", linewidth=2.0)

    w = linearRegressionGD(data, w, iterations=2)
    y = [(i * w[0] + w[1]) for i in x]
    plt.plot(x, y, 'm--', label="After 2 Iterations", linewidth=2.0)

    w = linearRegressionGD(data, w, iterations=3)
    y = [(i * w[0] + w[1]) for i in x]
    plt.plot(x, y, 'y--', label="After 3 Iterations", linewidth=2.0)

    plt.legend(bbox_to_anchor=(1.05, 1), loc=2, fontsize=20, borderaxespad=0.)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.grid()
    plt.show()
```

```
ierationsPlot('data.csv',w)
```



- True line
- After 0 Iterations
- - After 1 Iterations
- . After 2 Iterations
- . After 3 Iterations

Ridge and Lasso

- **Ridge (L2 regularization)**

Objective Function

$$\min_w L(w) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i)^2 + \lambda \cdot w^T \cdot w$$

Gradient Descent

$$\frac{\partial L}{\partial w} = -\frac{2}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i) \cdot x_i + \lambda w$$

- **Lasso(L1 regularization)**

Objective Function

$$\min_w L(w) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i)^2 + \lambda \cdot |w|$$

Gradient Descent

$$\frac{\partial L}{\partial w} = -\frac{2}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i) \cdot x_i + \lambda \cdot \text{sgn}(w)$$

I Linear Regression

Gradient descent (regularization)

```
def linearRegressionGDReg(data, wInitial=None, learningRate=0.05, iterations=50, regParam=0.01, regType=None):
    featureLen = len(data.take(1)[0].x)
    n = data.count()
    if wInitial is None:
        w = np.random.normal(size=featureLen) # w should be broadcasted if it is large
    else:
        w = wInitial
    for i in range(iterations):
        wBroadcast = sc.broadcast(w)
        gradient = data.map(lambda p: -2 * (p.y-np.dot(wBroadcast.value,p.x)) * np.array(p.x)) \
                    .reduce(lambda a, b: a + b)
        if regType == "Ridge":
            wReg = w * 1
            wReg[-1] = 0 #last value of weight vector is bias term, ignored in regularization
        elif regType == "Lasso":
            wReg = w * 1
            wReg[-1] = 0 #last value of weight vector is bias term, ignored in regularization
            wReg = (wReg>0).astype(int) * 2-1
        else:
            wReg = np.zeros(w.shape[0])
        gradient = gradient + regParam * wReg #gradient: GD of Squared Error+ GD of regularized term
        w = w - learningRate * gradient / n
    return w
```

Ridge Regression

```
np.random.seed(400)
linearRegressionGDReg(data, iterations=50, regParam=0.1, regType="Ridge")
array([ 7.98398871, -1.60876051])
```

Lasso Regression

Lar Shanhan Contact:James.Shanahan@gmail.com 336

```
np.random.seed(400)
linearRegressionGDReg(data, iterations=50, regParam=0.1, regType="Lasso")
```

Part 3: Machine Learning in Spark

- **Data Frames**
- **MLLib**
- **Write your own algos**
 - Linear regression (Ridge and Lasso)
 - Logistic Regression
- **Pipelines**
- **R and Spark**
 - Linear Regression example
- **Graphs in Spark**
- **Case studies**
 - Flight delay
 - Mobile advertising
 - Social Networks

• Logistic regression

Estimating Parameters using Gradient Descent

- Unfortunately, there is no closed form solution to maximizing $I(W)$ with respect to W . Therefore, one common approach is to use gradient ascent, in which we work with the gradient, which is the vector of partial derivatives. The i th component of the vector gradient has the form

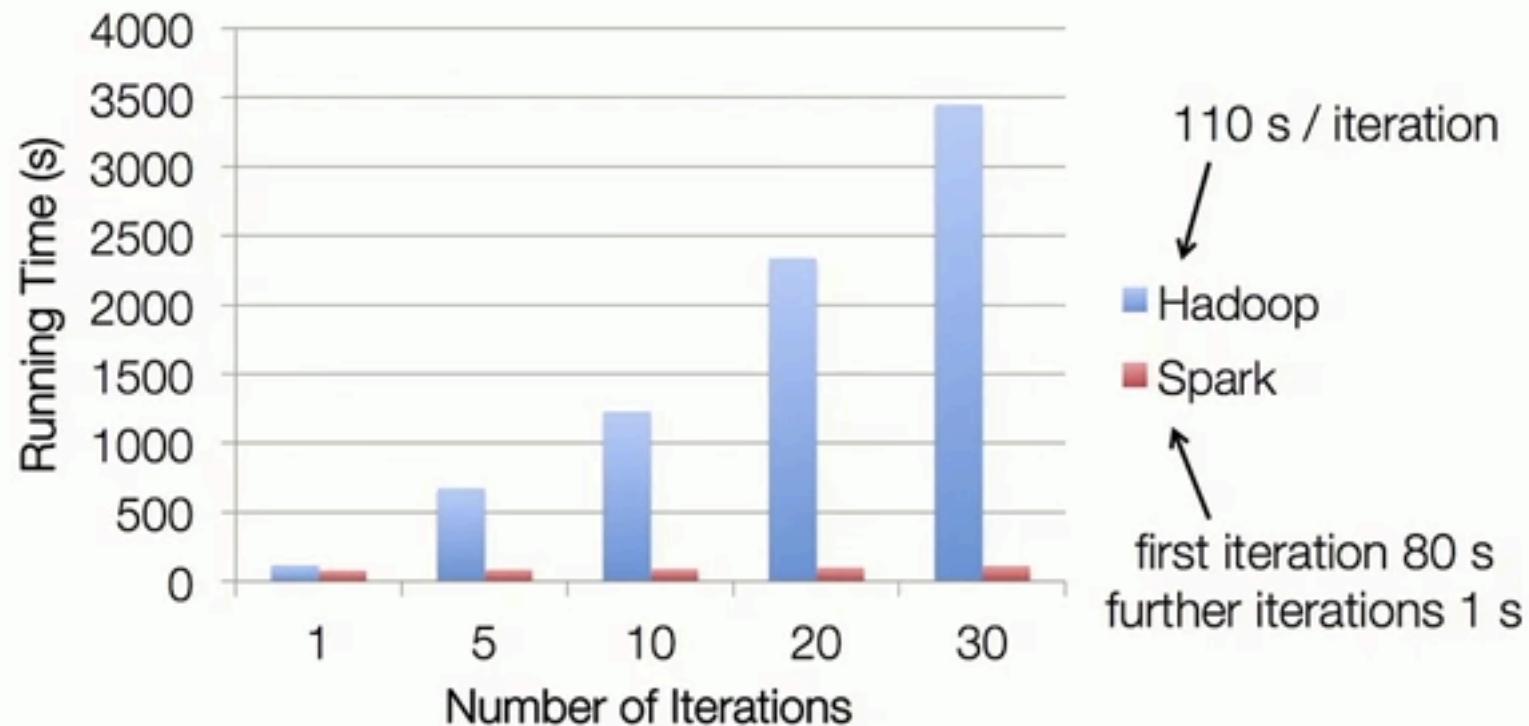
$$l(W) = \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + \exp(w_0 + \sum_i^n w_i X_i^l))$$

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

Beginning with initial weights of zero, we repeatedly update the weights in the direction of the gradient, changing the i th weight according to *this formula*, where η is a small constant (e.g., 0.01) which determines the step size. Effectively we are pulling weight vector closer to the examples where we make mistakes

Logistic Regression Results

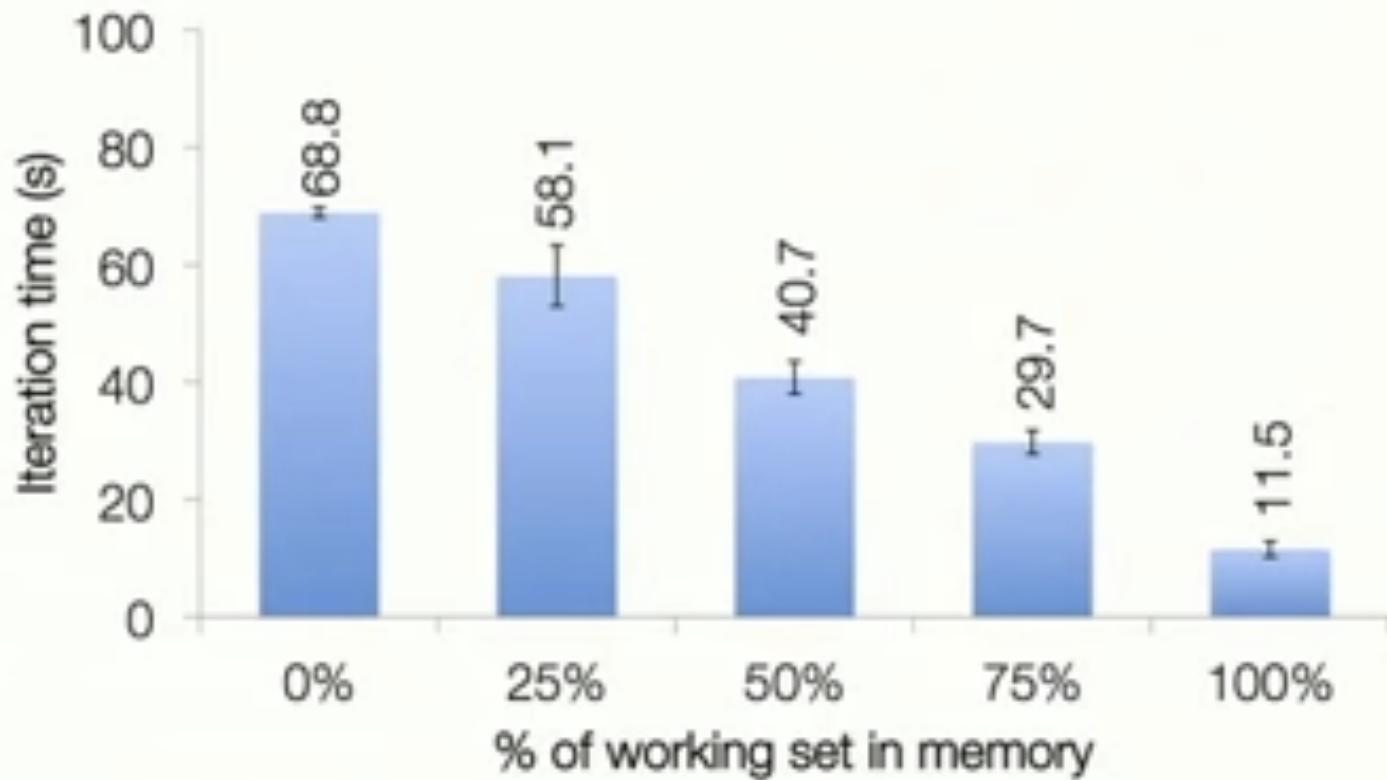


100 GB of data on 50 m1.xlarge EC2 machines

[From Reza Zadeh, Spark Summit 2015]

If the working set does not fit in memory

Behavior with Less RAM



[From Reza Zadeh, Spark Summit 2015]

Logistic Regression

Data Generation

```
import numpy as np
import csv
def data_generate(fileName, w=[0,0,0], size = 100):
    size = 100
    np.random.seed(0)
    x1 = np.random.uniform(-4, 4, size)
    x2 = np.random.uniform(-4, 4, size)
    noise = np.random.normal(0, 3, size)
    v = (x1 * w[0] + x2 * w[1] + w[2] + noise)
    y = (v>0) * 2-1
    data = zip(y, x1, x2)
    with open(fileName,'wb') as f:
        writer = csv.writer(f)
        for row in data:
            writer.writerow(row)
    return True
```

```
w = np.array([8, -3, -1])
data_generate('data.csv', w, 100)
```

```
True
```

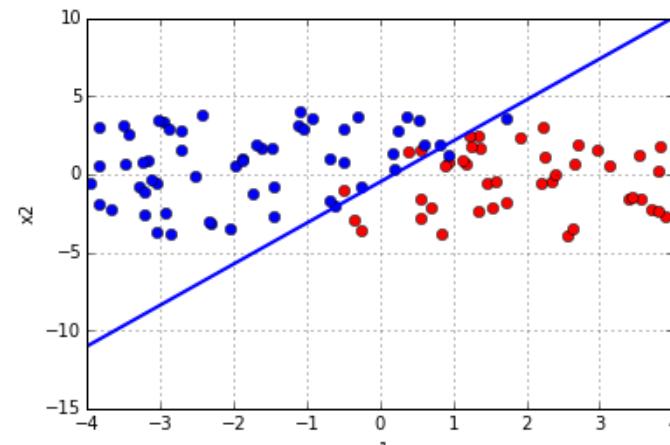
Data format: y, x1,x2

```
1,-6.886661137329684879e-01
-1,1.756760849652599932e+00
-1,7.656676365450297839e-01
-1,3.740612089503772886e+00
```

Data Visualization

```
%matplotlib inline
import matplotlib.pyplot as plt
def dataPlot(file, w):
    cols = {'1': 'r', '-1': 'b'}
    with open(file,'r') as f:
        reader = csv.reader(f)
        for row in reader:
            plt.plot(float(row[1]), float(row[2]), cols[row[0]]+'o')
    plt.xlabel("x1")
    plt.ylabel("x2")
    x1 = [-4,4]
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, linewidth=2.0)
    plt.grid()
    plt.show()
```

```
dataPlot('data.csv',w)
```



Logistic Regression

Objective Function

$$\min_w L(w) = \frac{1}{n} \sum_{i=0}^{n-1} \log(1 + e^{-y_i \cdot w^T \cdot x_i})$$

$y = \{-1, 1\}$

Gradient Descent

$$\frac{\partial L}{\partial w} = \frac{1}{n} \sum_{i=0}^{n-1} \left(\frac{1}{1 + e^{-y_i \cdot w^T \cdot x_i}} - 1 \right) \cdot y_i \cdot x_i$$

Gradient descent (no regularization)

```
from collections import namedtuple
import numpy as np
Point = namedtuple('Point', 'x y')

def readPoint(line):
    d = line.split(',')
    x = [float(i) for i in d[1:]]
    x.append(1.0) #bias term
    return Point(x, float(d[0]))

def logisticRegressionGD(data, wInitial=None, learningRate=0.05, iterations=100):
    featureLen = len(data.take(1)[0].x)
    n = data.count()
    if wInitial is None:
        w = np.random.normal(size=featureLen)
    else:
        w = wInitial
    for i in range(iterations):
        wBroadcast = sc.broadcast(w)
        gradient = data.map(lambda p: (1 / (1 + np.exp(-p.y * np.dot(wBroadcast.value, p.x))) - 1) * p.y * np.array(p.x))\
            .reduce(lambda a, b: a + b) Sum up gradients
        w = w - learningRate * gradient / n
    #w = w / np.linalg.norm(w) #normalization
    return w

data = sc.textFile('data.csv').map(readPoint).cache()
logisticRegressionGD(data)

Le array([ 1.35420091, -0.46407845, -1.01559616])
```

Get gradients for each instance

Logistic Regression

Plot w in iterations

```
def ierationsPlot(fileName, truew):
    x1 = [-4, 4]

    w = truew
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'b', label="True line", linewidth=4.0)

    np.random.seed(800)
    w = np.random.normal(0,1,3)
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'r--', label="After 0 Iterations", linewidth=2.0)

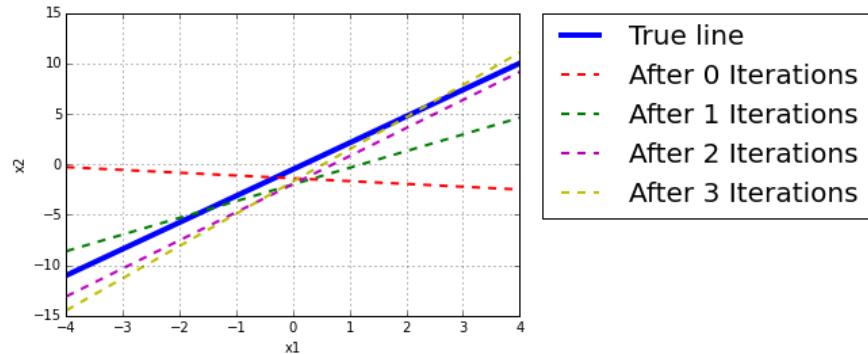
    data = sc.textFile(fileName).map(readPoint).cache()
    w = logisticRegressionGD(data, w, iterations=30)
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'g--', label="After 1 Iterations", linewidth=2.0)

    w = logisticRegressionGD(data, w, iterations=30)
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'm--', label="After 2 Iterations", linewidth=2.0)

    w = logisticRegressionGD(data, w, iterations=30)
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'y--', label="After 3 Iterations", linewidth=2.0)

    plt.legend(bbox_to_anchor=(1.05, 1), loc=2, fontsize=20, borderaxespad=0.)
    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.grid()
    plt.show()

ierationsPlot('data.csv',w)
```



Ridge and Lasso

- **Ridge (L2 regularization)**

Objective Function

$$\min_w L(w) = \frac{1}{n} \sum_{i=0}^{n-1} \text{Log}(1 + e^{-y_i \cdot w^T \cdot x_i}) + \lambda \cdot w^T \cdot w \quad \mathbf{y} = \{-1, 1\}$$

Gradient Descent

$$\frac{\partial L}{\partial w} = \frac{1}{n} \sum_{i=0}^{n-1} \left(\frac{1}{1 + e^{-y_i \cdot w^T \cdot x_i}} - 1 \right) \cdot y_i \cdot x_i + \lambda w$$

- **Lasso(L1 regularization)**

$\mathbf{y} = \{-1, 1\}$

Objective Function

$$\min_w L(w) = \frac{1}{n} \sum_{i=0}^{n-1} \text{Log}(1 + e^{-y_i \cdot w^T \cdot x_i}) + \lambda \cdot |w|$$

Gradient Descent

$$\frac{\partial L}{\partial w} = \frac{1}{n} \sum_{i=0}^{n-1} \left(\frac{1}{1 + e^{-y_i \cdot w^T \cdot x_i}} - 1 \right) \cdot y_i \cdot x_i + \lambda \cdot \text{sgn}(w)$$

Logistic regression

Gradient descent (regularization)

```
def logisticRegressionGDReg(data, wInitial=None, learningRate=0.05, iterations=50, regParam=0.01, regType=None):
    featureLen = len(data.take(1)[0].x)
    n = data.count()
    if wInitial is None:
        w = np.random.normal(size=featureLen) # w should be broadcasted if it is large
    else:
        w = wInitial
    for i in range(iterations):
        wBroadcast = sc.broadcast(w)
        gradient = data.map(lambda p: (1 / (1 + np.exp(-p.y*np.dot(wBroadcast.value, p.x))))-1) * p.y * np.array(p.x))\
            .reduce(lambda a, b: a + b)
        if regType == "Ridge":
            wReg = w * 1
            wReg[-1] = 0 #last value of weight vector is bias term, ignored in regularization
        elif regType == "Lasso":
            wReg = w * 1
            wReg[-1] = 0 #last value of weight vector is bias term, ignored in regularization
            wReg = (wReg>0).astype(int) * 2-1
        else:
            wReg = np.zeros(w.shape[0])
        gradient = gradient + regParam * wReg #gradient: GD of Squared Error+ GD of regularized term
        w = w - learningRate * gradient / n
    return w
```

Ridge Regression

```
np.random.seed(400)
logisticRegressionGDReg(data, iterations=50, regParam=0.1, regType="Ridge")
array([ 0.74835721, -0.17134752, -0.39953122])
```

L Lasso Regression

Contact:James.Shanahan@gmail.com

346

Broadcast variables

- Improve this using broadcast variables
 - Each iteration
 - Create a new RDD (with a DAG tree; map+Reduce that are shipped to the workers)
-
- Fault tolerant, distributed, scaleable gradient descent

Separable Updates

Can be generalized for

- » Unconstrained optimization
- » Smooth or non-smooth
- » LBFGS, Conjugate Gradient, Accelerated Gradient methods, ...

Part 3: Machine Learning in Spark

- **Data Frames**
- **MLLib**
- **Write your own algos**
 - Linear regression (Ridge and Lasso)
 - Logistic Regression
- **Pipelines**
- **R and Spark**
 - Linear Regression example
- **Graphs in Spark**
- **Case studies**
 - Flight delay
 - Mobile advertising
 - Social Networks

- **Pipelines**

Pipelines

- In machine learning, it is common to run a sequence of algorithms to process and learn from data.
- E.g., a simple text document processing workflow might include several stages:
 - Split each document's text into words.
 - Convert each document's words into a numerical feature vector.
 - Normalization
 - Learn a prediction model using the feature vectors and labels.
 - Spark ML represents such a workflow as a Pipeline, which consists of a sequence of PipelineStages (Transformers and Estimators) to be run in a specific order. We will use this simple workflow as a running example in this section.

Pipeline-Training data

In Python, The single asterisk form (*args) is used to pass a non-keyworded, variable-length argument list in function definition. It can also unpacks the sequence/collection into positional arguments

Example of Unpacking

```
def sum(a, b):
    return a + b

values = (1, 2)

s = sum(*values)
```

```
# Prepare training documents, which are labeled.
LabeledDocument = Row("id", "text", "label")
training = sc.parallelize([(0, "a b c d e spark", 1.0),
                           (1, "b d", 0.0),
                           (2, "spark f g h", 1.0),
                           (3, "hadoop mapreduce", 0.0),
                           (4, "b spark who", 1.0),
                           (5, "g d a y", 0.0),
                           (6, "spark fly", 1.0),
                           (7, "was mapreduce", 0.0),
                           (8, "e spark program", 1.0),
                           (9, "a e c l", 0.0),
                           (10, "spark compile", 1.0),
                           (11, "hadoop software", 0.0)
                          ]) \
    .map(lambda x: LabeledDocument(*x)).toDF()
training.collect()
```

```
[Row(id=0, text=u'a b c d e spark', label=1.0),
Row(id=1, text=u'b d', label=0.0),
Row(id=2, text=u'spark f g h', label=1.0),
Row(id=3, text=u'hadoop mapreduce', label=0.0),
Row(id=4, text=u'b spark who', label=1.0),
Row(id=5, text=u'g d a y', label=0.0),
Row(id=6, text=u'spark fly', label=1.0),
Row(id=7, text=u'was mapreduce', label=0.0),
Row(id=8, text=u'e spark program', label=1.0),
Row(id=9, text=u'a e c l', label=0.0),
Row(id=10, text=u'spark compile', label=1.0),
Row(id=11, text=u'hadoop software', label=0.0)]
```

Training Pipeline

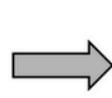
Pipeline
(Estimator)



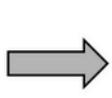
Pipeline.fit()



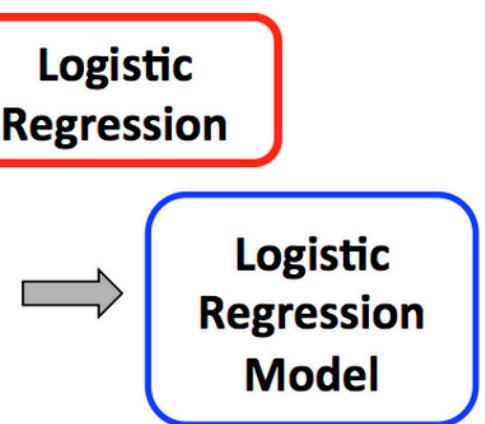
Raw
text



Words



Feature



```
from pyspark import SparkContext
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer
from pyspark.sql import Row, SQLContext

sc = SparkContext(appName="SimpleTextClassificationPipeline")
sqlContext = SQLContext(sc)

# Prepare training documents, which are labeled.
LabeledDocument = Row("id", "text", "label")
training = sc.parallelize([(0L, "a b c d e spark", 1.0),
                           (1L, "b d", 0.0),
                           (2L, "spark f g h", 1.0),
                           (3L, "hadoop mapreduce", 0.0)]) \
    .map(lambda x: LabeledDocument(*x)).toDF()

# Configure an ML pipeline, which consists of tree stages: tokenizer, hashingTF, and lr.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

# Fit the pipeline to training documents.
model = pipeline.fit(training)
```

Training Pipeline

```
from pyspark import SparkContext
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer
from pyspark.sql import Row, SQLContext

sc = SparkContext(appName="SimpleTextClassificationPipeline")
sqlContext = SQLContext(sc)

# Prepare training documents, which are labeled.
LabeledDocument = Row("id", "text", "label")
training = sc.parallelize([(0L, "a b c d e spark", 1.0),
                           (1L, "b d", 0.0),
                           (2L, "spark f g h", 1.0),
                           (3L, "hadoop mapreduce", 0.0)]) \
    .map(lambda x: LabeledDocument(*x)).toDF()

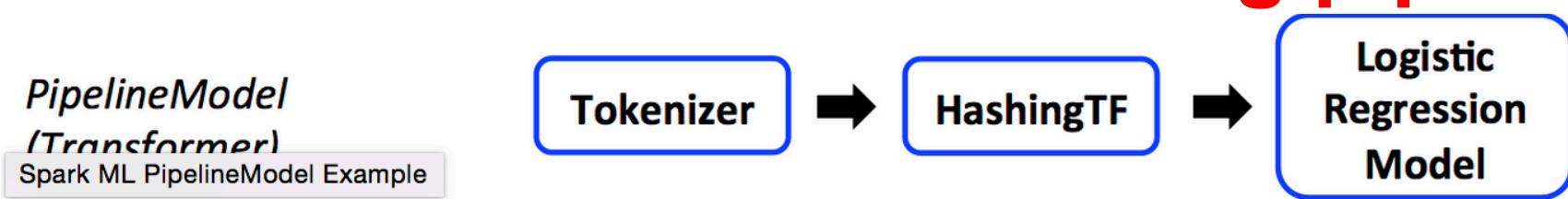
# Configure an ML pipeline, which consists of three stages: tokenizer, hashingTF, and lr.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

# Fit the pipeline to training documents.
model = pipeline.fit(training)
```

.ogistic
gression

→ Logistic
Regression
Model

Testing pipeline



Above, the PipelineModel has the same number of stages as the original Pipeline, but all operations in the original Pipeline

```
# Prepare test documents, which are unlabeled.  
Document = Row("id", "text")  
test = sc.parallelize([(4L, "spark i j k"),  
                      (5L, "l m n"),  
                      (6L, "mapreduce spark"),  
                      (7L, "apache hadoop"))]\br/>.map(lambda x: Document(*x)).toDF()  
  
# Make predictions on test documents and print columns of interest.  
prediction = model.transform(test)  
selected = prediction.select("id", "text", "prediction")  
for row in selected.collect():  
    print row
```

ML Pipelines: use pipeline-based CV

Hyper-parameter Tuning

```
// Build a parameter grid.  
val paramGrid = new ParamGridBuilder()  
  .addGrid(hashingTF.numFeatures, Array(10, 20, 40))  
  .addGrid(lr.regParam, Array(0.01, 0.1, 1.0))  
  .build()  
  
// Set up cross-validation.  
val cv = new CrossValidator()  
  .setNumFolds(3)  
  .setEstimator(pipeline)  
  .setEstimatorParamMaps(paramGrid)  
  .setEvaluator(new BinaryClassificationEvaluator)  
  
// Fit a model with cross-validation.  
val cvModel = cv.fit(trainingDataset)
```

Pipeline - Stages & Cross Validation



```
# Configure an ML pipeline, which consists of tree stages: tokenizer, hashingTF, and lr.
from pyspark import SparkContext
from pyspark.ml import Pipeline
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

# Prepare training documents, which are labeled.
LabeledDocument = Row("id", "text", "label")
training = sc.parallelize([(0L, "a b c d e spark", 1.0),
                           (1L, "b d", 0.0),
                           (2L, "spark f g h", 1.0),
                           (3L, "hadoop mapreduce", 0.0)]) \
    .map(lambda x: LabeledDocument(*x)).toDF()

# Configure an ML pipeline, which consists of tree stages: tokenizer, hashingTF, and lr.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
```

Pipeline - Stages&Cross Validation

3 hashingTF.number of features: 10, 100, 1000

2 logistic regression regularization parameters: 0.1, 0.001

```
# Use a ParamGridBuilder to construct a grid of parameters to search over.  
paramGrid = ParamGridBuilder() \  
    .addGrid(hashingTF.numFeatures, [10, 100, 1000]) \  
    .addGrid(lr.regParam, [0.1, 0.01]) \  
    .build()
```

```
# Run cross-validation, and choose the best set of parameters  
crossval = CrossValidator(estimator=pipeline,  
                          estimatorParamMaps=paramGrid,  
                          evaluator=BinaryClassificationEvaluator(),  
                          numFolds=2) # use 3+ folds in practice  
  
cvModel = crossval.fit(training)
```

For each iteration in cross validation, follow this pipeline

By default, BinaryclassificationEvaluator is areaUnderROC

Pipeline-Prediction

```
# Prepare test documents, which are unlabeled.
Document = Row("id", "text")
test = sc.parallelize([(4, "spark i j k"),
                      (5, "l m n"),
                      (6, "spark hadoop spark"),
                      (7, "apache hadoop")]) \
    .map(lambda x: Document(*x)).toDF()

# Make predictions on test documents and print columns of interest.
prediction = cvModel.transform(test)
selected = prediction.select("id", "text", "prediction")
for row in selected.collect():
    print(row)

sc.stop()

Row(id=4, text=u'spark i j k', prediction=1.0)
Row(id=5, text=u'l m n', prediction=0.0)
Row(id=6, text=u'spark hadoop spark', prediction=1.0)
Row(id=7, text=u'apache hadoop', prediction=0.0)
```

Part 3: Machine Learning in Spark

- **Data Frames**
- **MLLib**
- **Write your own algos**
 - Linear regression (Ridge and Lasso)
 - Logistic Regression
- **Pipelines**
- **R and Spark**
 - Linear Regression example
- **Graphs in Spark**
- **Case studies**
 - Flight delay
 - Mobile advertising
 - Social Networks

-
- **R and Spark**
 - **R API was released in June 2014 as part of Spark 1.4**

Install SparkR from the R command line

Install SparkR at the R command line prompt

STEP 1

```
#install if (!require('devtools')) install.packages('devtools')
install.packages('devtools')
library('devtools')
#download SparkR
devtools::install_github('apache/spark@v1.4.1', subdir='R/pkg')
```

STEP 2

Run the following at the command line prompt or put it into .Rprofile

Change SPARK_HOME to point to same place as in iupyter notebook :
<https://dl.dropboxusercontent.com/u/27377155/linearRegression.R>

```
Sys.setenv(SPARK_HOME='/Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/spark-1.4.0-bin-hadoop2.6/') #be CAREFUL with the COMMAS
#Sys.setenv(SPARK_HOME='/Users/liang/Downloads/spark-1.4.1-bin-hadoop2.6/')
.libPaths(c(file.path(Sys.getenv('SPARK_HOME'), 'R', 'lib'), .libPaths()))
library(SparkR)
sc <- sparkR.init(master='local')
sqlContext <- sparkRSQl.init(sc)
```

Install SparkR from the R command line

Install SparkR at the R command line prompt

STEP 1

```
#install if (!require('devtools')) install.packages('devtools')
install.packages('devtools')
```

```
> library('devtools')
Error: unexpected input in "library,"
>
> library('devtools')
> devtools::install_github('apache/spark@v1.4.1', subdir='R/pkg')
Downloading github repo apache/spark@v1.4.1
Installing SparkR
'/Library/Frameworks/R.framework/Resources/bin/R' --no-site-file --no-environ --no-save \
--no-restore CMD INSTALL \
'/private/var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/RtmpogzQf0/devtools11c727c7c622/
apache-spark-dbaa5c2/R/pkg' \
--library='/Users/jshanahan/Library/R/3.1/library' --install-tests
* installing *source* package 'SparkR' ...
** R
** inst
** tests
** preparing package for lazy loading
Creating a new generic function for 'na.omit' in package 'SparkR'
Creating a new generic function for 'filter' in package 'SparkR'
Creating a new generic function for 'intersect' in package 'SparkR'
Creating a new generic function for 'sample' in package 'SparkR'
Creating a generic function for 'lapply' from package 'base' in package 'SparkR'
Creating a generic function for 'Filter' from package 'base' in package 'SparkR'
Creating a generic function for 'mean' from package 'base' in package 'SparkR'
```

ST

Run
line p

Sys
Cla

#S

.lib

libr

sc

sql

Install SparkR from the R command line

Install SparkR at the R command line prompt

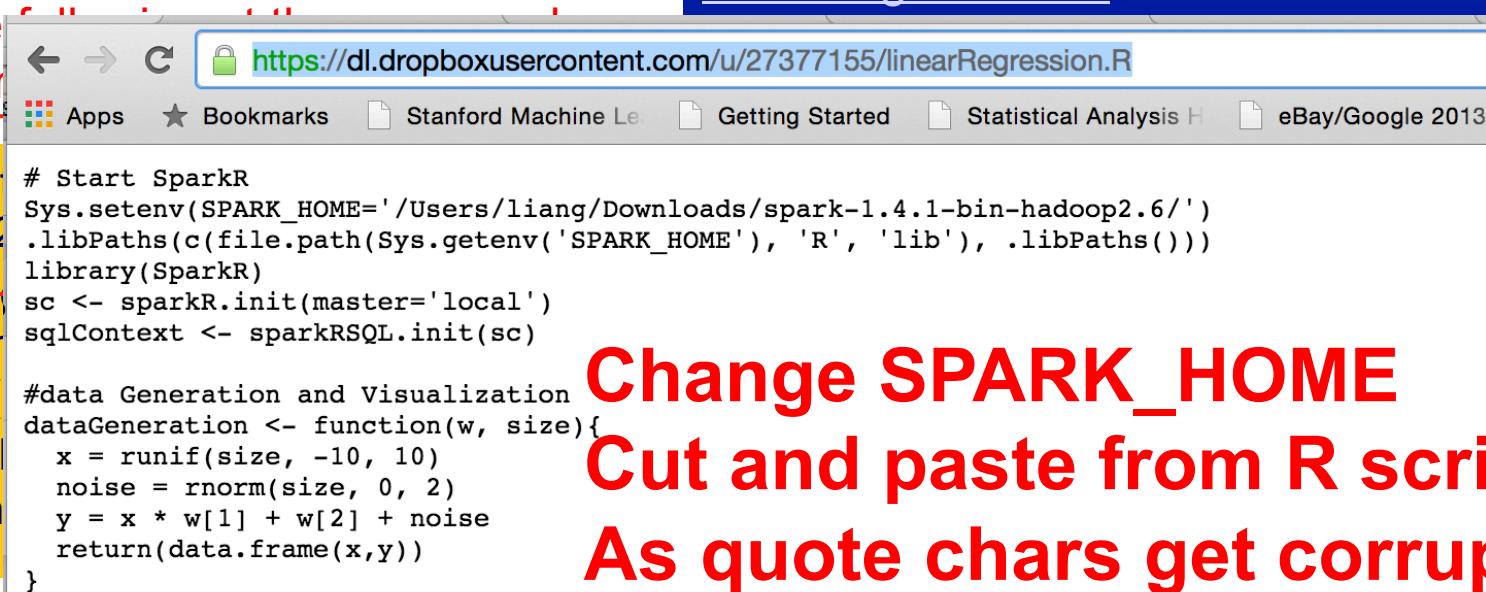
STEP 1

```
#install if (!require('devtools')) install.packages('devtools')
install.packages('devtools')
library('devtools')
#download SparkR
devtools::install_github('apache/spark@v1.4.1', subdir='R/pkg')
```

Change SPARK_HOME to point to same place
as in iPython notebook :
[https://dl.dropboxusercontent.com/u/27377155/
linearRegression.R](https://dl.dropboxusercontent.com/u/27377155/linearRegression.R)

STEP 2

Run the
line prompt



```
# Start SparkR
Sys.setenv(SPARK_HOME='/Users/liang/Downloads/spark-1.4.1-bin-hadoop2.6/')
.libPaths(c(file.path(Sys.getenv('SPARK_HOME')), 'R', 'lib'), .libPaths())
library(SparkR)
sc <- sparkR.init(master='local')
sqlContext <- sparkRSQl.init(sc)

#data Generation and Visualization
dataGeneration <- function(w, size){
  x = runif(size, -10, 10)
  noise = rnorm(size, 0, 2)
  y = x * w[1] + w[2] + noise
  return(data.frame(x,y))
}
```

Change SPARK_HOME
Cut and paste from R script
As quote chars get corrupted

-
- <https://dl.dropboxusercontent.com/u/27377155/linearRegression.R>



The screenshot shows a web browser window with the URL <https://dl.dropboxusercontent.com/u/27377155/linearRegression.R> in the address bar. The browser interface includes standard navigation buttons (back, forward, search), a bookmarks bar, and a tab bar with several open tabs. The main content area displays an R script:

```
# Start SparkR
Sys.setenv(SPARK_HOME='/Users/liang/Downloads/spark-1.4.1-bin-hadoop2.6/')
.libPaths(c(file.path(Sys.getenv('SPARK_HOME')), 'R', 'lib'), .libPaths()))
library(SparkR)
sc <- sparkR.init(master='local')
sqlContext <- sparkRSQl.init(sc)

#data Generation and Visualization
dataGeneration <- function(w, size){
  x = runif(size, -10, 10)
  noise = rnorm(size, 0, 2)
  y = x * w[1] + w[2] + noise
  return(data.frame(x,y))
}

w <- c(4, 2)
size <- 100
data <- dataGeneration(w, size)
plot(data$x,data$y,xlab='x', ylab='y')
x <- c(-10, 10)
y <- x * w[1] + w[2]
lines(x, y, col='red')
```

How to create Spark DataFrames in R

- **From R data frameS**

```
df <- createDataFrame(sqlContext, RDataframeName)
```

- **From Data file (on say HDFS)**

```
df <- read.df(sqlContext, jsonFileName, "json")
df <- read.df(sqlContext, csv, "csv")
```

- **From Hive query**

```
hiveContext <- sparkRHive.init(sc)

sql(hiveContext, "CREATE TABLE IF NOT EXISTS src (key INT, value STRING)")
sql(hiveContext, "LOAD DATA LOCAL INPATH 'examples/src/main/resources/kv1.txt' INTO TABLE src")

# Queries can be expressed in HiveQL.
results <- sql(hiveContext, "FROM src SELECT key, value")
```

R Operations on Spark Dataframes

Selecting rows, columns

```
# Create the DataFrame
df <- createDataFrame(sqlContext, faithful)

# Get basic information about the DataFrame
df
## DataFrame[eruptions:double, waiting:double]

# Select only the "eruptions" column
head(select(df, df$eruptions))
## eruptions
##1 3.600
##2 1.800
##3 3.333

# You can also pass in column name as strings
head(select(df, "eruptions"))

# Filter the DataFrame to only retain rows with wait times shorter than 50 mins
head(filter(df, df$waiting < 50))
## eruptions waiting
##1 1.750      47
##2 1.750      47
##3 1.867      48
```

Linear regression in SparkR

<https://spark.apache.org/docs/latest/api/R/>

- Start SparkR

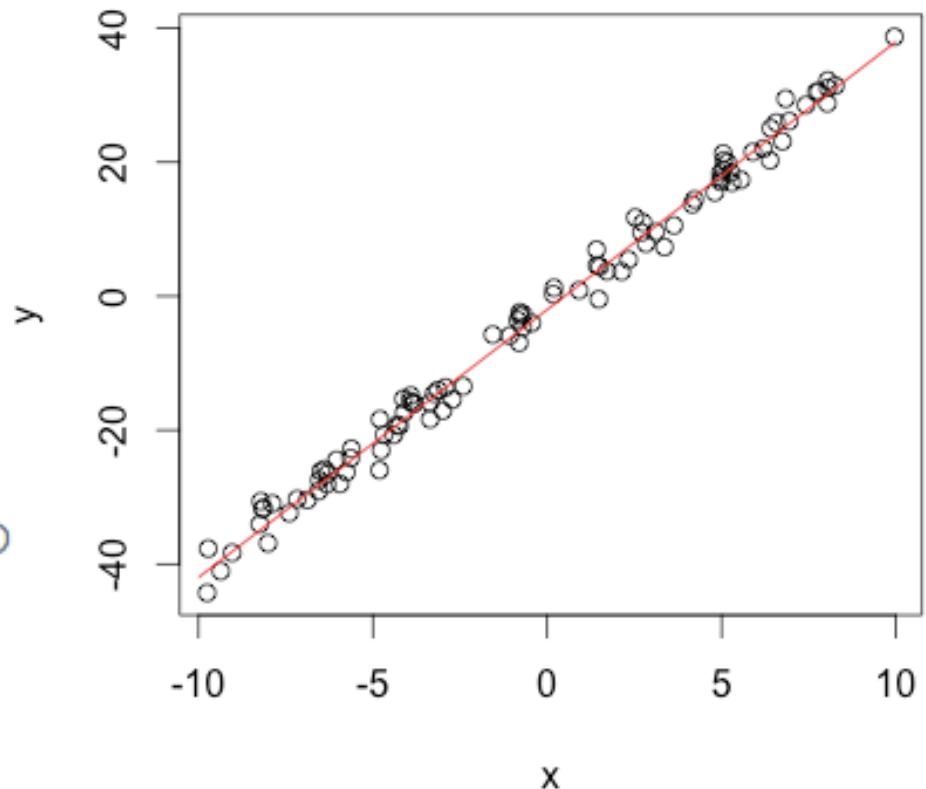
```
# Start SparkR
Sys.setenv(SPARK_HOME='/Users/liang/Downloads/spark-1.4.1-bin-hadoop2.6/')
.libPaths(c(file.path(Sys.getenv('SPARK_HOME')), 'R', 'lib'), .libPaths())
library(SparkR)
sc <- sparkR.init(master='local')
sqlContext <- sparkRSQl.init(sc)
> sc <- sparkR.init(master='local')
Launching java with spark-submit command /Users/liang/Downloads/spark-1.4.1-bin-hadoop2.6//bin/spark-submit  sparkr-shell /var/folders/nn/cq08lj857z584w_96tctfg8c0000gn/T//RtmpPRCm3e/backend_port545420aa48d7
log4j:WARN No appenders could be found for logger (io.netty.util.internal.logging.InternalLoggerFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
15/08/10 05:09:38 INFO SparkContext: Running Spark version 1.4.1
15/08/10 05:09:38 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
15/08/10 05:09:39 INFO SecurityManager: Changing view acls to: liang
15/08/10 05:09:39 INFO SecurityManager: Changing modify acls to: liang
15/08/10 05:09:39 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(liang)
); users with modify permissions: Set(liang)
15/08/10 05:09:40 INFO Slf4jLogger: Slf4jLogger started
15/08/10 05:09:40 INFO Remoting: Starting remoting
15/08/10 05:09:40 INFO Remoting: Remoting started; listening on addresses :[akka.tcp://sparkDriver@10.0.0.9:49291]
```

Linear regression

- Data Generation and Visualization

```
#data Generation and Visualization
dataGeneration <- function(w, size){
  x = runif(size, -10, 10)
  noise = rnorm(size, 0, 2)
  y = x * w[1] + w[2] + noise
  return(data.frame(x,y))
}

w <- c(4, -0.1)      True w
size <- 100
data <- dataGeneration(w, size)
plot(data$x,data$y,xlab='x', ylab='y')
x <- c(-10, 10)
y <- x * w[1] + w[2]
lines(x, y, col='red')
```



Linear regression

- Training Create a Spark dataframe from a R dataframe

```
df <- createDataFrame(sqlContext, data)
cache(df)
w <- c(0, 0)
cat("Initial w: ", w, "\n")
learningRate = 0.01
n <- count(df)
iterations <- 300
savedW <- matrix(NA, iterations+1, 2) The learning trace is saved
savedW[1, ] <- w
for (i in 1:iterations) {
  df$gradientx <- (df$y - df$x * w[1] - w[2]) * df$x * (-2)
  df$gradientbias <- (df$y - df$x * w[1] - w[2]) * (-2)
  w[1] <- w[1] - learningRate * collect(agg(df, sumg = sum(df$gradientx)))[[1]]/n
  w[2] <- w[2] - learningRate * collect(agg(df, sumg2 = sum(df$gradientbias)))[[1]]/n
  savedW[i+1, ] <- w
}
```

R API still needs lots more work

Spark R focuses on highlevel operations

- **SparkR version (partial API):**
 - <https://spark.apache.org/docs/latest/api/R/>
 - Initial support for Spark in R be focused on high level operations instead of low level ETL.
 - This may change in the (1.5) version.
- **amplab (very complete API)**
 - <https://amplab-extras.github.io/SparkR-pkg/rdocs/1.2/index.html>
- **The APIs are quite different**
 - E.g, No map, mappartitions in SparkR but present in amplab

NOTE: As of April 2015, SparkR has been officially merged into Apache Spark and is shipping in an upcoming release (1.4) due early summer 2015. You can contribute and follow SparkR developments on the Apache Spark mailing lists and issue tracker.

NOTE: The API from the upcoming Spark release (1.4) will not have the same API as described here. Initial support for Spark in R be focused on high level operations instead of low level ETL. This may change in the (1.5) version.

<https://amplab-extras.github.io/SparkR-pkg/>

Large Scale Distributed Data Science using Spark © KDD2015 James G. Shanahan Contact:James.Shanahan@gmail.com

371

-
- I have not found a linear regression API in R (so far!)

Linear regression Result in SparkR

- **Result:**

```
Iteration 0     w:  0 0
Iteration 20    w:  4.036858 0.6578106
Iteration 40    w:  4.028114 1.037249
Iteration 80    w:  4.018285 1.463748
Iteration 120   w:  4.013844 1.656458
Iteration 160   w:  4.011838 1.743532
Iteration 200   w:  4.010931 1.782875
Iteration 240   w:  4.010521 1.800652
Iteration 280   w:  4.010336 1.808685
|
```

SparkR shell

```
ldai:spark-1.4.1-bin-hadoop2.6 liang$ ./bin/sparkR  
  
ab5d5fbf/httpd-bb18c5a9-f617-4717-b981-390ddc95b59f  
15/08/10 10:53:19 INFO HttpServer: Starting HTTP Server  
15/08/10 10:53:19 INFO Utils: Successfully started service 'HTTP file server' on  
port 60798.  
15/08/10 10:53:19 INFO SparkEnv: Registering OutputCommitCoordinator  
15/08/10 10:53:20 WARN Utils: Service 'SparkUI' could not bind on port 4040. Att  
empting port 4041.  
15/08/10 10:53:20 WARN Utils: Service 'SparkUI' could not bind on port 4041. Att  
empting port 4042.  
15/08/10 10:53:20 INFO Utils: Successfully started service 'SparkUI' on port 404  
2.  
15/08/10 10:53:20 INFO SparkUI: Started SparkUI at http://10.0.0.9:4042  
15/08/10 10:53:20 INFO Executor: Starting executor ID driver on host localhost  
15/08/10 10:53:20 INFO Utils: Successfully started service 'org.apache.spark.net  
work.netty.NettyBlockTransferService' on port 60799.  
15/08/10 10:53:20 INFO NettyBlockTransferService: Server created on 60799  
15/08/10 10:53:20 INFO BlockManagerMaster: Trying to register BlockManager  
15/08/10 10:53:20 INFO BlockManagerMasterEndpoint: Registering block manager loc  
alhost:60799 with 265.1 MB RAM, BlockManagerId(driver, localhost, 60799)  
15/08/10 10:53:20 INFO BlockManagerMaster: Registered BlockManager  
  
Welcome to SparkR!  
Spark context is available as sc, SQL context is available as sqlContext  
>
```

Iris Dataset (distribute into an RDD/ Spark Dataframe!)

```
df <- createDataFrame(sqlContext, iris)

-----
Warning messages:
1: In FUN(X[[5L]], ...) :
  Use Sepal_Length instead of Sepal.Length  as column name
2: In FUN(X[[5L]], ...) :
  Use Sepal_Width instead of Sepal.Width   as column name
3: In FUN(X[[5L]], ...) :
  Use Petal_Length instead of Petal.Length  as column name
4: In FUN(X[[5L]], ...) :
  Use Petal_Width instead of Petal.Width   as column name
> df
DataFrame[Sepal_Length:double, Sepal_Width:double, Petal_Length:double, Petal_Width:double, Species:string]

> head(df)
```
classmate.java.-2, look 4.24/142 »
 Sepal_Length Sepal_Width Petal_Length Petal_Width Species
1 5.1 3.5 1.4 0.2 setosa
2 4.9 3.0 1.4 0.2 setosa
3 4.7 3.2 1.3 0.2 setosa
4 4.6 3.1 1.5 0.2 setosa
5 5.0 3.6 1.4 0.2 setosa
6 5.4 3.9 1.7 0.4 setosa
```

# Part 3: Machine Learning in Spark

---

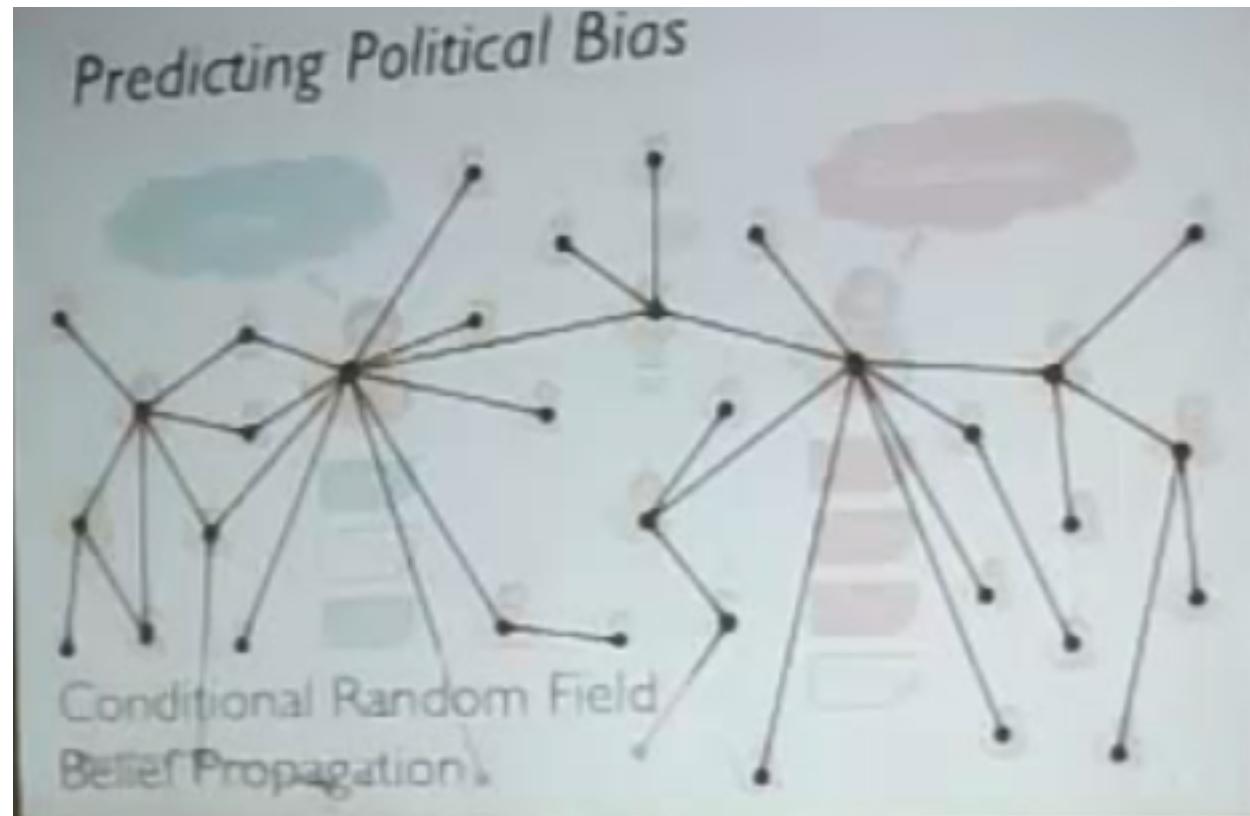
- **Data Frames**
- **MLLib**
- **Write your own algos**
  - Linear regression (Ridge and Lasso)
  - Logistic Regression
- **Pipelines**
- **R and Spark**
  - Linear Regression example
- **Graphs in Spark**
- **Case studies**
  - Flight delay
  - Mobile advertising
  - Social Networks

---

- **Graphs**

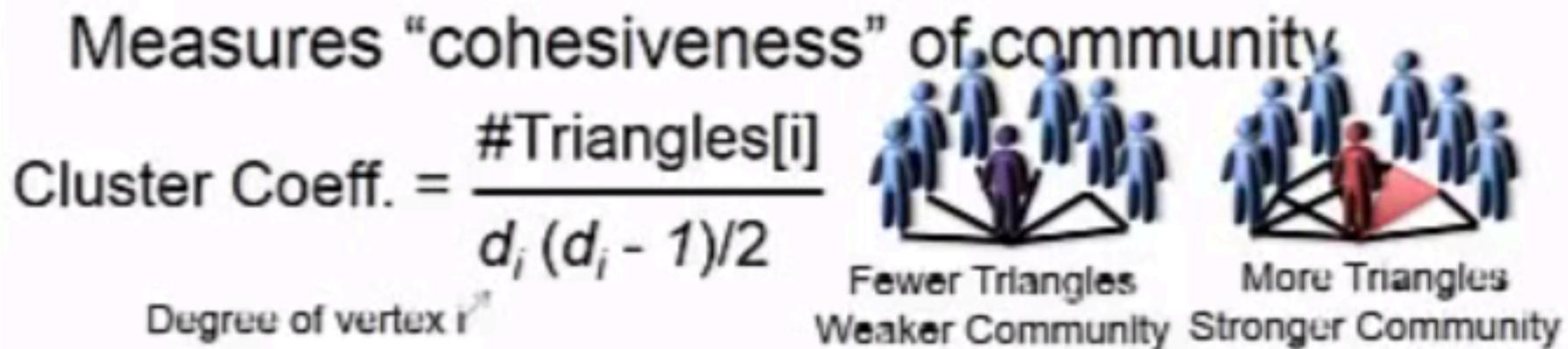
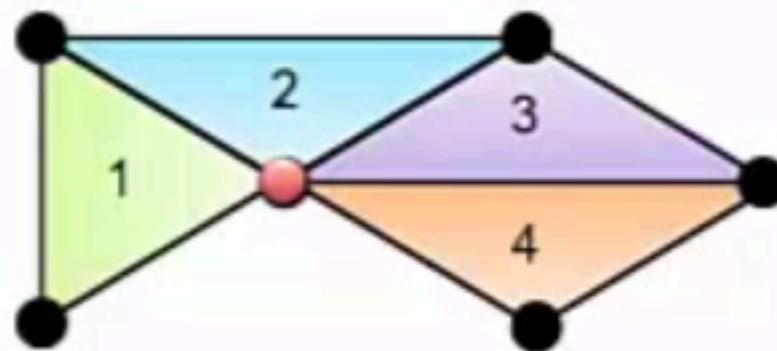
# Political bias

- ..

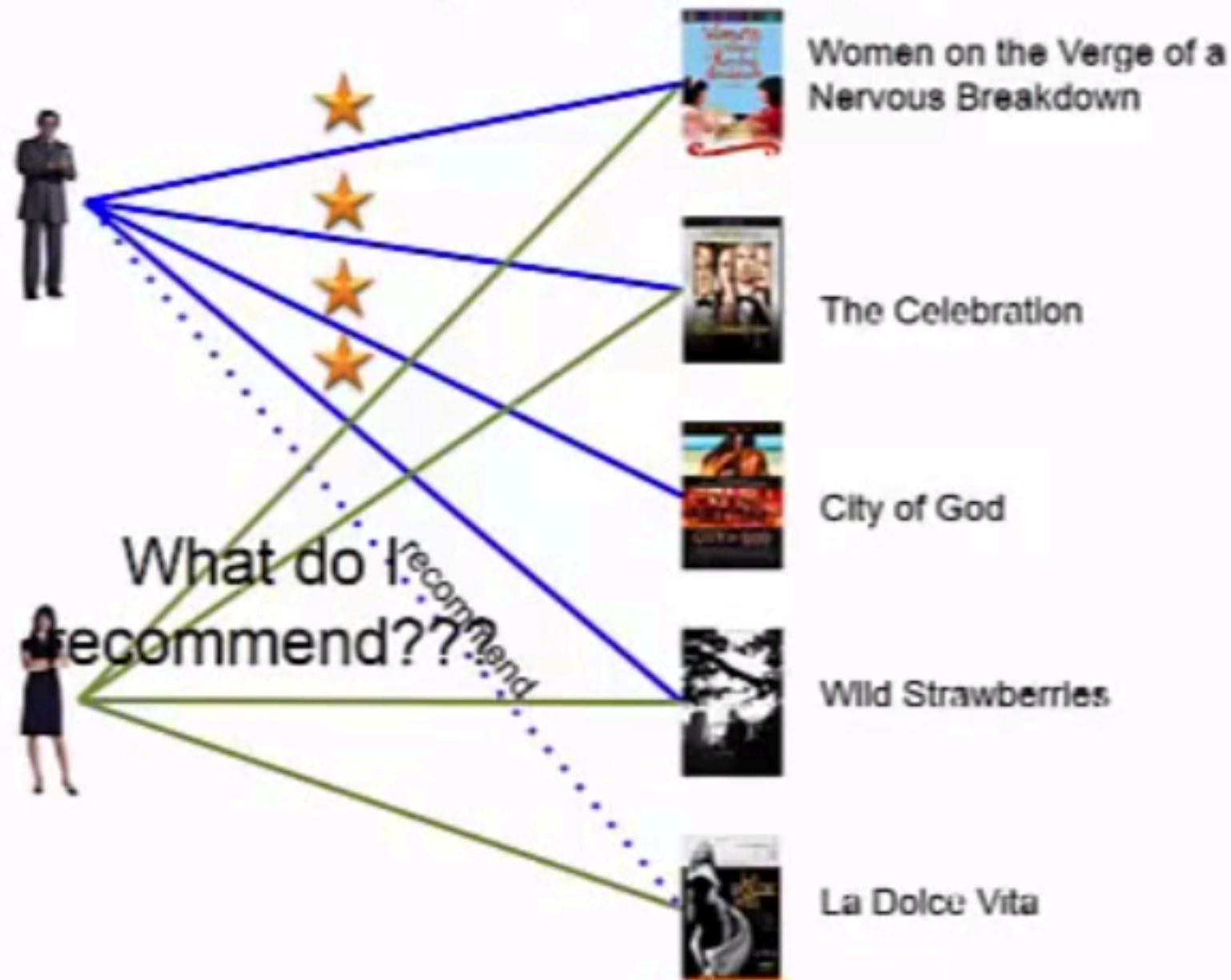


# Triangle Counting

Count the triangles passing through each vertex:

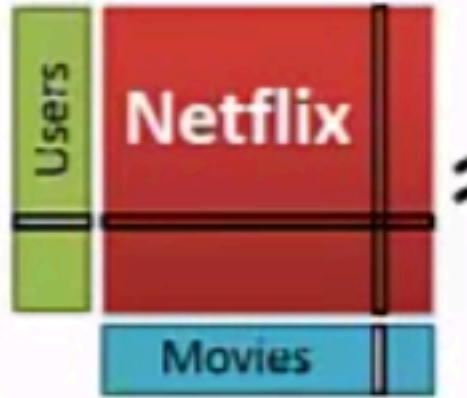


## Collaborative Filtering: Exploiting Dependencies



# Matrix Factorization

## Alternating Least Squares (ALS)

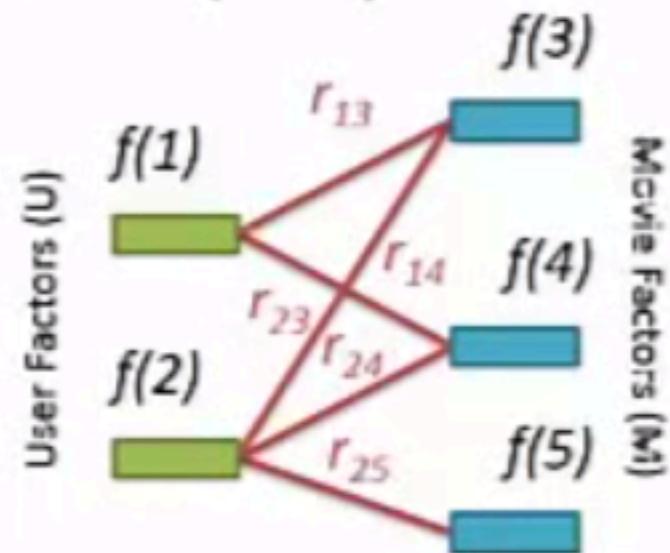


$$\approx \begin{matrix} \text{Users} \\ \times \\ \text{Movies} \end{matrix} \quad f(i) \quad f(j)$$

Iterate:

$$f[i] = \arg \min_{w \in \mathbb{R}^k} \sum_{j \in \text{Nbrs}(i)} (r_{ij} - w^T f[j])^2 + \lambda ||w||_2^2$$

Factor for  
User  $i$



Factor for  
Movie  $j$

# Suggesting links/attribute values in a social network

---

• ..

---

# Predicting and Recommending Links and attributes in Social Networks

Based on a paper by Lars Backstrom Jure Leskovec, WSDM 2011

<http://cs.stanford.edu/people/jure/pubs/linkpred-wsdm11.pdf>

# Link prediction problem: Network+node/edge data

---

- Predicting the occurrence of links is a fundamental problem in networks.
- In the link prediction problem we are given a snapshot of a network and would like to infer which interactions among existing members are likely to occur in the near future or which existing interactions are we missing.
- Network+node/edge data
  - Although this problem has been extensively studied, the challenge of how to effectively combine the information from the network structure with rich node and edge attribute data remains largely open.

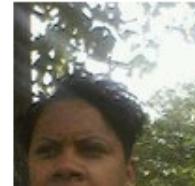
## Find friends from different parts of your life

Use the checkboxes below to discover people you know from your hometown, school, employer and more.

### Hometown

Indianapolis, Indiana

Enter another city



**Judy Pyles**  
36 mutual friends  
[Add Friend](#)



**Rocky Campbell**  
41 mutual friends  
[Add Friend](#)



**Laura White**  
12 mutual friends  
[Add Friend](#)



**King Ro Conley**  
59 mutual friends  
[Add Friend](#)



**Dillon Rhodes**  
43 mutual friends  
[Add Friend](#)



**Rhonda Landrum**  
54 mutual friends  
[Add Friend](#)

### Current City

Indianapolis, Indiana

Enter another city

### High School

North Central High School

Enter another high school



**David Corbitt**  
90 mutual friends  
[Add Friend](#)



**Eric Bettis**  
15 mutual friends  
[Add Friend](#)



**Eric Hughes**  
110 mutual friends  
[Add Friend](#)



**Marki Ann**  
26 mutual friends  
[Add Friend](#)



**Michael Pugh**  
21 mutual friends  
[Add Friend](#)



**Lisa Williams**  
22 mutual friends  
[Add Friend](#)

### Mutual Friend

Enter a name

Growing

### College or University

Martin University

Enter another college



**LouieBaur Digg**  
39 mutual friends  
[Add Friend](#)



**LaTonya Mayberry Bynum**  
51 mutual friends  
[Add Friend](#)



**Durece Johnson**  
2 mutual friends  
[Add Friend](#)



**Kendale Adams**  
64 mutual friends  
[Add Friend](#)



**Bruce T. Caldwell**  
143 mutual friends  
[Add Friend](#)



**Angela Blackwell Miller**  
61 mutual friends  
[Add Friend](#)

### Employer

ARIES GRAPHIC DESIGN

Enter another employer



**Landon Montel**



**Kevin Brown**



**Stanley F. Henry**



**Saundria Mccrackin**



**Ebonye X-Endsley**



**Anita Hawkins**

# Finding Friends

Linking other things such as groups

- Growing body of research captures dynamics of social network graphs

[Latanzi, Sivakumar '08] [Zheleva, Sharara , Getoor '09] [Kumar, Novak, Tomkins '06] [Kossinets, Watts '06] [L., Kleinberg, Faloutsos '05]



- What links will occur next? [LibenNowell, Kleinberg '03]
  - Networks + many other features:  
Location, School, Job, Hobbies, Interests, etc.

# Supervised Link Prediction

- **Q:** How to combine **network structure** and **node and edge features**?
- **A:** Combine **PageRank** with **Supervised learning**
  - **PageRank** is great to capture importances of nodes based on the network structure
  - **Supervised learning** is great with features
- **Idea:** Use **node and edge features** to “guide” the random walk

# PAGERANK: from random walks to pagerank

---

- View pages as states, and webGraph as a transition matrix
- A Markov process who steady state distribution tells about which pages we spend a lot of time on.
- Making some minor adjustments to the transition matrix
  - (stochasticity adjustment to deal with dangling nodes (sinks)); In this adjustment, the 0 rows of matrix H are replaced with  $1/n e$  making H stochastic. This adjustment now allows the random surfer to hyperlink to any page at random after entering a dangling node.
  - Primitivity adjustment via teleportation
- We can then use the power iteration method
  - (first eigenvector, all positive values; eigenvalue ==1)
- This is a proxy for popularity and is a very discriminating feature for in sebsearch context webpages
- Variations: Personalized pagerank

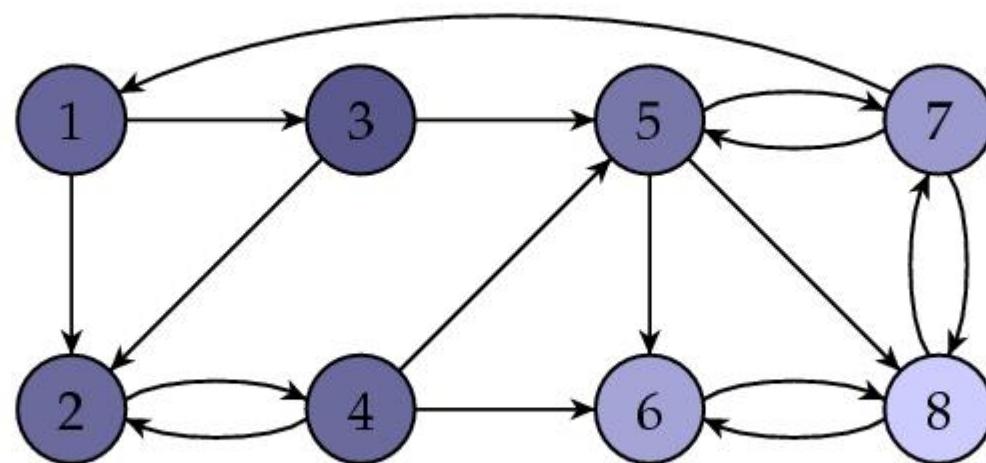
- 
- This random walk algorithm serves as the basis for many more graph-based algorithms
  - The modifications we made also (for personalization) will be revisited in the context of social graphs under a very different guise.
  - Some of which we will look at in later lectures
  - Most web, mobile, social Graphs these days run in the order of trillions of edges...
  -

# Transition matrix; stationary vector

Graph in matrix form

$$H = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1/3 & 0 \\ 1/2 & 0 & 1/2 & 1/3 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 1/3 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 1/3 & 1/3 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 1/3 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 1/3 & 1 & 1/3 & 0 \end{bmatrix} \quad \text{with stationary vector } I = \begin{bmatrix} 0.0600 \\ 0.0675 \\ 0.0300 \\ 0.0675 \\ 0.0975 \\ 0.2025 \\ 0.1800 \\ 0.2950 \end{bmatrix}$$

This shows that page 8 wins the popularity contest. Here is the same figure with the web pages shaded in such a way that the pages with high PageRanks are lighter.



# Computing PR Via Power Method

The method is founded on the following general principle that we will soon investigate.

**General principle:** *The sequence  $I^k$  will converge to the stationary vector  $I$ .*

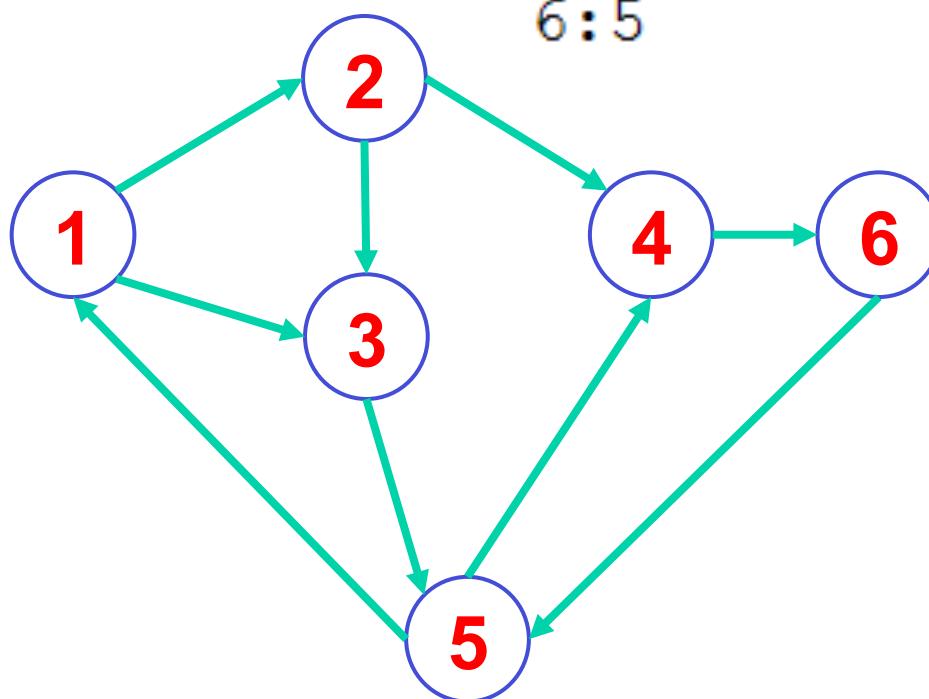
We will illustrate with the example above.

| $I^0$ | $I^1$ | $I^2$ | $I^3$  | $I^4$  | ... | $I^{60}$ | $I^{61}$ |
|-------|-------|-------|--------|--------|-----|----------|----------|
| 1     | 0     | 0     | 0      | 0.0278 | ... | 0.06     | 0.06     |
| 0     | 0.5   | 0.25  | 0.1667 | 0.0833 | ... | 0.0675   | 0.0675   |
| 0     | 0.5   | 0     | 0      | 0      | ... | 0.03     | 0.03     |
| 0     | 0     | 0.5   | 0.25   | 0.1667 | ... | 0.0675   | 0.0675   |
| 0     | 0     | 0.25  | 0.1667 | 0.1111 | ... | 0.0975   | 0.0975   |
| 0     | 0     | 0     | 0.25   | 0.1806 | ... | 0.2025   | 0.2025   |
| 0     | 0     | 0     | 0.0833 | 0.0972 | ... | 0.18     | 0.18     |
| 0     | 0     | 0     | 0.0833 | 0.3333 | ... | 0.295    | 0.295    |

It is natural to ask what these numbers mean. Of course, there can be no absolute measure of a page's importance, only relative measures for comparing the importance of two pages through statements such as "Page A is twice as important as Page B." For this reason, we may multiply all the importance rankings by some fixed quantity without affecting the information they tell us. In this way, we will always assume, for reasons to be explained shortly, that the sum of all the popularities is one.

# PageRank homegrown implementation

Adjacency  
matrix is fixed



1:2,3  
2:3,4  
3:5  
4:6  
5:1,4  
6:5

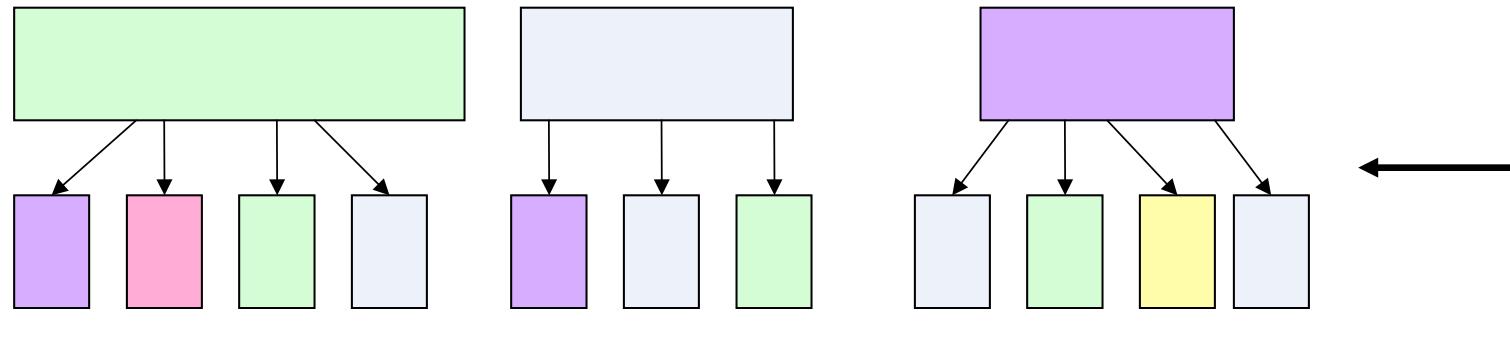
```
: %%writefile PageRank.txt
1:2,3
2:3,4
3:5
4:6
5:1,4
6:5
```

Overwriting PageRank.txt

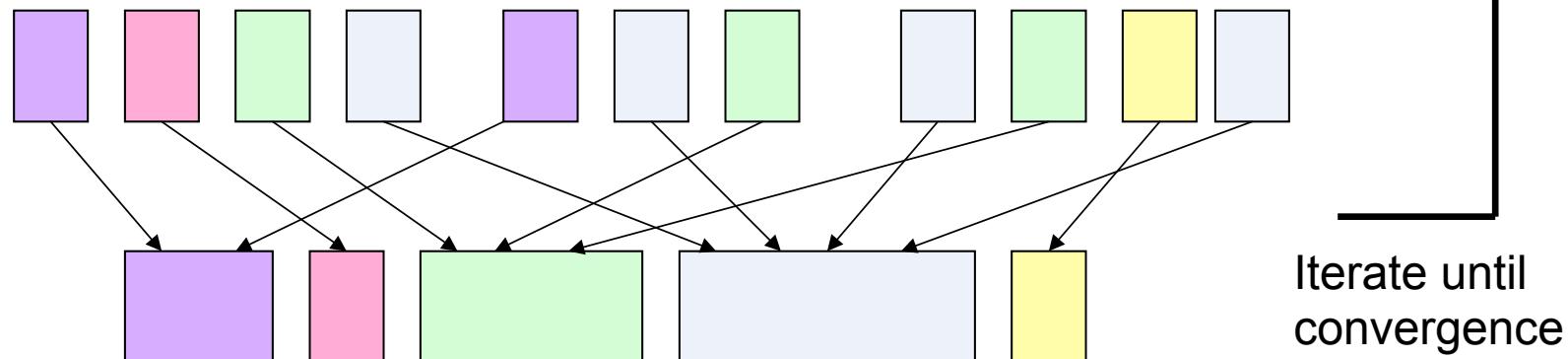
# PageRank in MapReduce

Like a Sparse Matrix by Vector multiplication

Map: distribute PageRank “credit” to link targets



Reduce: gather up PageRank “credit” from multiple sources to compute new PageRank value



Iterate until convergence

# PageRank homegrown

```

def computeContribs(urls, rank):
 num_urls = len(urls)
 for url in urls:
 yield (url, rank / num_urls)

def parseDataGraph(line):
 fields = line.split(':')
 return(fields[0], fields[1].split(','))

links = sc.textFile("PageRank.txt").map(parseDataGraph).cache()
ranks = links.map(lambda (url, neighbors): (url, 1.0))
for iteration in xrange(10):
 contribs = links.join(ranks).flatMap(lambda (url, (urls, rank)): computeContribs(urls, rank))
 ranks = contribs.reduceByKey(lambda x,y: x + y).mapValues(lambda rank: rank * 0.85 + 0.15)
print ranks.collect()
sc.stop()

```

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

|       | NodeID | Neighbors    |
|-------|--------|--------------|
| Links | 1      | 2, 3         |
|       | 2      | 3, 4         |
|       | 3      | 5            |
|       | 4      | 6            |
|       | 5      | 1, 4         |
|       | 6      | 5            |
| Ranks | NodeID | Rank ratings |
|       | 1      | 1.0          |
|       | 2      | 1.0          |
|       | 3      | 1.0          |
|       | 4      | 1.0          |
|       | 5      | 1.0          |
|       | 6      | 1.0          |

Join flatMap

Joined table

| NodeID | Neighbors | Rank Ratings |
|--------|-----------|--------------|
| 1      | 2, 3      | 1.0          |
| 2      | 3, 4      | 1.0          |
| 3      | 5         | 1.0          |
| 4      | 6         | 1.0          |
| 5      | 1, 4      | 1.0          |
| 6      | 5         | 1.0          |

flat Map

| Neighbord | Rank Ratings |
|-----------|--------------|
| 2         | 0.5          |
| 3         | 0.5          |
| 3         | 0.5          |
| 4         | 0.5          |
| 5         | 1.0          |
| 6         | 1.0          |
| 1         | 0.5          |
| 4         | 0.5          |
| 5         | 1.0          |

Contribs

# PageRank homegrown

```

def computeContribs(urls, rank):
 num_urls = len(urls)
 for url in urls:
 yield (url, rank / num_urls)

def parseDataGraph(line):
 fields = line.split(':')
 return(fields[0], fields[1].split(','))

links = sc.textFile("PageRank.txt").map(parseDataGraph).cache()
ranks = links.map(lambda (url, neighbors): (url, 1.0))
for iteration in xrange(10):
 contribs = links.join(ranks).flatMap(lambda (url, (urls, rank)): computeContribs(urls, rank))
 ranks = contribs.reduceByKey(lambda x,y: x + y).mapValues(lambda rank: rank * 0.85 + 0.15)
print ranks.collect()
sc.stop()

```

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

reduceByKey mapValues

teleportation parameter  
d is 0.85

Contribs

| Neighbord | Rank Ratings |
|-----------|--------------|
| 2         | 0.5          |
| 3         | 0.5          |
| 3         | 0.5          |
| 4         | 0.5          |
| 5         | 1.0          |
| 6         | 1.0          |
| 1         | 0.5          |
| 4         | 0.5          |
| 5         | 1.0          |

reduce  
ByKey

| NodeID | Rank ratings |
|--------|--------------|
| 1      | 0.5          |
| 2      | 0.5          |
| 3      | 1.0          |
| 4      | 1.0          |
| 5      | 2.0          |
| 6      | 1.0          |

mapValues

Updated Ranks

| NodeID | Rank ratings |
|--------|--------------|
| 1      | 0.575        |
| 2      | 0.575        |
| 3      | 1.0          |
| 4      | 1.0          |
| 5      | 1.85         |
| 6      | 1.0          |

# GraphX

---

- **GraphX is a new component in Spark for graphs and graph-parallel computation. At a high level, GraphX extends the Spark RDD by introducing a new Graph abstraction: a directed multigraph with properties attached to each vertex and edge.**
- **GraphX exposes a variant of the Pregel API.**
  - At a high level the Pregel operator in GraphX is a bulk-synchronous parallel messaging abstraction constrained to the topology of the graph.
  - The Pregel operator executes in a series of super steps in which vertices receive the sum of their inbound messages from the previous super step, compute a new value for the vertex property, and then send messages to neighboring vertices in the next super step.
- **GraphX exposes a variant of the GraphLab API.**

# Many Graph-Parallel Algorithms

## Collaborative Filtering

- > Alternating Least Squares
- > Stochastic Gradient Descent
- > Tensor Factorization

## Structured Prediction

- > Loopy Belief Propagation
- > Max-Product Linear Programs
- > Gibbs Sampling

## Semi-supervised ML

- > Graph SSL
- > CoEM

## Community Detection

- > Triangle-Counting
- > K-core Decomposition
- > K-Truss

## Graph Analytics

- > PageRank
- > Personalized PageRank
- > Shortest Path
- > Graph Coloring

## Classification

- > Neural Networks

# Graph Primitives

A graph consists of 2 RDDs

**A graph consists**

- » **RDD[Vertice]** (`id: Int, data: VD`)
- » **RDD[Edge]** (`src: Int, dst: Int, data: ED`)

**Relational operators:**

- » E.g. filter vertices, edges, joining vertices with tables

**Graph computation primitives:**

- » `aggregateNeighbors(mapUdf, reduceUdaf)`

Map on the neighbors; reduce

# Specialized API: Pregel

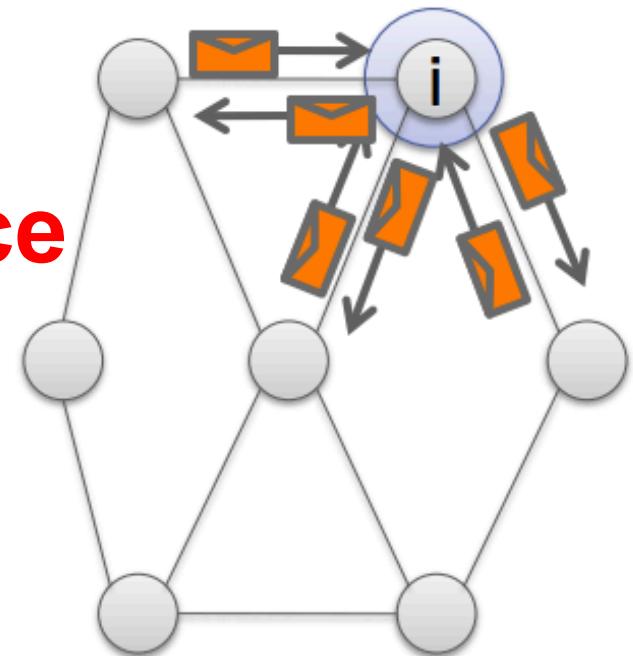
Vertex-Programs interact by sending messages.

```
Pregel_PageRank(i, messages) :
 // Receive all the messages
 total = 0
 foreach(msg in messages) :
 total = total + msg
```

Reduce

```
 // Update the rank of this vertex
 R[i] = 0.15 + total
```

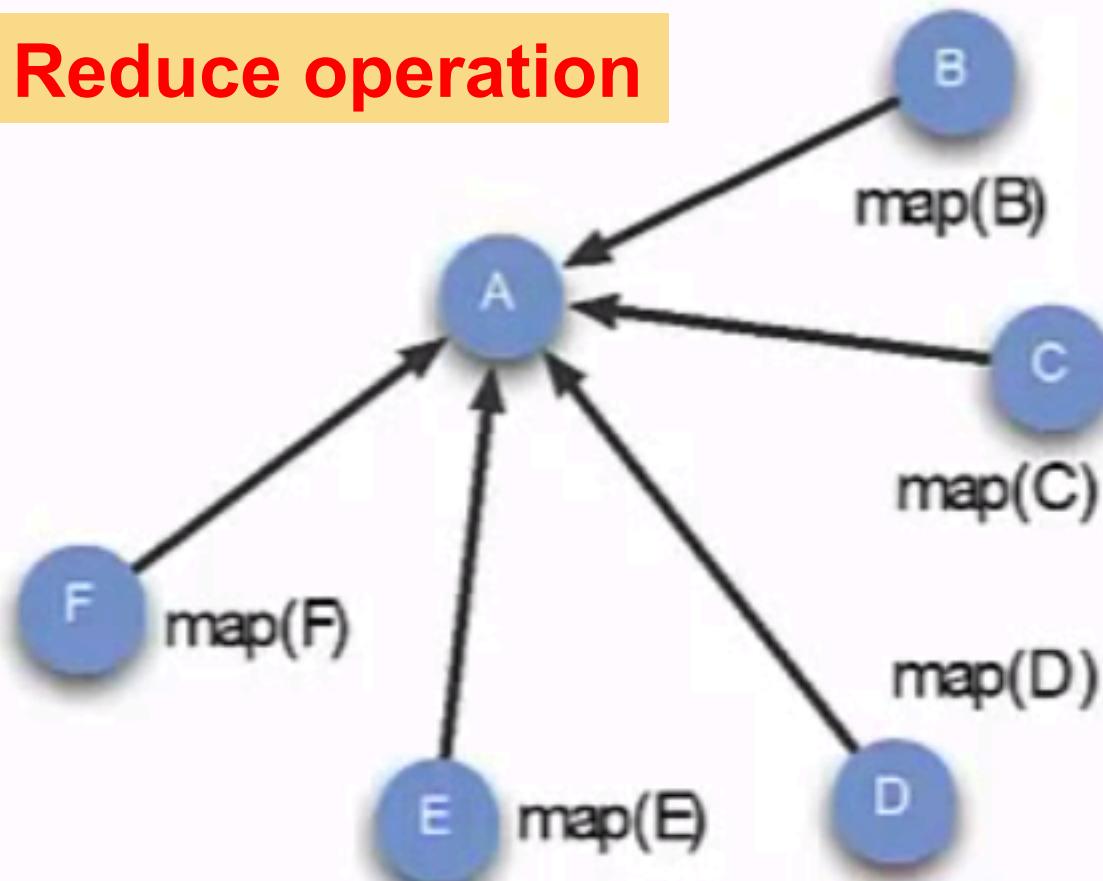
```
 // Send new messages to neighbors
 foreach(j in out_neighbors[i]) :
 Send msg(R[i]) to vertex j
```



# aggregateNeighbors

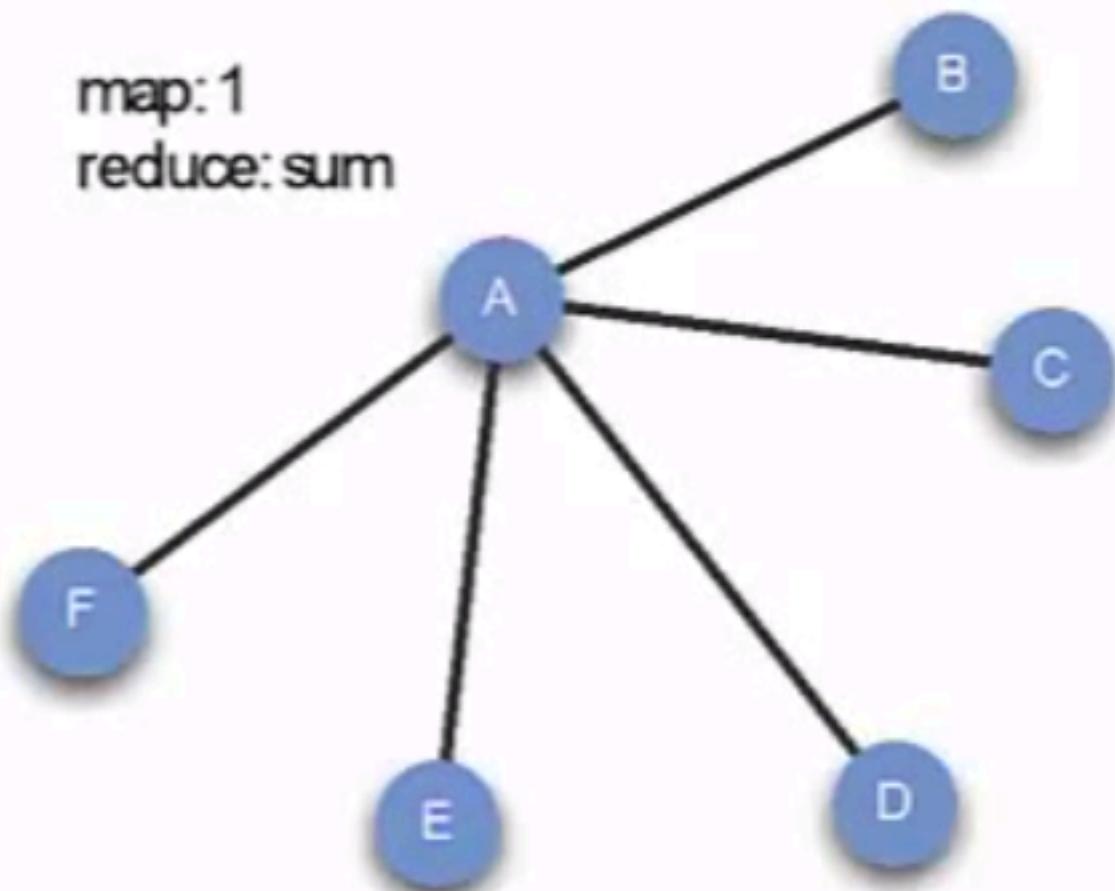
- 

**Reduce operation**



# Example: Vertex Degree

map:1  
reduce:sum



# Neighborhood Aggregation is a key operation

## Neighborhood Aggregation

A key step in many graph analytics tasks is aggregating information about the neighborhood of each vertex. For example, we might want to know the number of followers each user has or the average age of the the followers of each user. Many iterative graph algorithms (e.g., PageRank, Shortest Path, and connected components) repeatedly aggregate properties of neighboring vertices (e.g., current PageRank Value, shortest path to the source, and smallest reachable vertex id).

To improve performance the primary aggregation operator changed from `graph.mapReduceTriplets` to the new `graph.aggregateMessages`. While the changes in the API are relatively small, we provide a transition guide below.

## Aggregate Messages (`aggregateMessages`)

The core aggregation operation in GraphX is `aggregateMessages`. This operator applies a user defined `sendMsg` function to each *edge triplet* in the graph and then uses the `mergeMsg` function to aggregate those messages at their destination vertex.

```
class Graph[VD, ED] {
 def aggregateMessages [Msg: ClassTag](
 sendMsg: EdgeContext[VD, ED, Msg] => Unit,
 mergeMsg: (Msg, Msg) => Msg,
 tripletFields: TripletFields = TripletFields.All)
 : VertexRDD[Msg]
 }
}
```

---

## Graph computation primitives:

- » aggregateNeighbors(mapUdf, reduceUdaf)

**Map on the neighbors; reduce**

A 251: Large-Scale Web Analytics and Machine Learning © 2014 James G. Shanahan James.Shanahan AT gmail.com

We can express both Pregel and  
GraphLab using aggregateNeighbors in  
**40 lines of code!**

# Graph-Parallel Systems

Pregel  
oogle



GraphLab

Expose *specialized APIs* to simplify graph programming.

Exploit graph structure to achieve *orders-of-magnitude performance gains* over more general data-parallel systems.

# GraphX (only in Scala)

The edge list file must be one  
edge per line

```
import org.apache.spark.graphx._
//load graph from edge list
val graph = GraphLoader.edgeListFile(sc, "links.txt")
//run until ranks gets converged
graph.pageRank(tol=0.01, resetProb=0.15).vertices.collect().foreach(println)
//run PageRank for a fixed number of iterations
graph.staticPageRank(numIter=10, resetProb=0.15).vertices.collect().foreach(println)
```

```
(4,1.0039711033263845)
(6,0.9894634744184481)
(2,0.4872265172779962)
(1,0.809841174076437)
(3,0.6813564465279436)
(5,1.552567468415146)
```

```
(4,0.9057198271884564)
(6,0.8928973656016074)
(2,0.44618661447778996)
(1,0.7218265979235846)
(3,0.6300798437426618)
(5,1.3992302889282138)
```

Result from pageRank

Result from staticPageRank (10 iterations)

Initialization is 0.0 instead of 1.0!!!!

# **Others graph algos in GraphX library**

---

- **Connected Components**
- **Triangle counting**
- **Shortest path**

# Part 3: Machine Learning in Spark

---

- **Data Frames**
- **MLLib**
- **Write your own algos**
  - Linear regression (Ridge and Lasso)
  - Logistic Regression
- **Pipelines**
- **R and Spark**
  - Linear Regression example
- **Graphs in Spark**
- **Case studies**
  - Flight delay
  - Mobile advertising
  - Social Networks

# Case study: model airline delays

| Salidas<br>Mostrador<br>Counter<br>Mastrador | Embarcam.<br>Boarding<br>Embarque | Porta<br>Gate<br>Puerta | Observaciones<br>Remarks<br>Observaciones |
|----------------------------------------------|-----------------------------------|-------------------------|-------------------------------------------|
| 698 09a10                                    | 21:00                             |                         | DELAYED                                   |
| 698 09a10                                    | 21:00                             |                         | DELAYED                                   |
| 07                                           | 20:58                             |                         | LAST CALL                                 |
| 2                                            | 20:32                             |                         | LAST CALL                                 |
| 01a08                                        | 21:25                             |                         | DELAYED                                   |
| 19a20                                        | 22:40                             |                         | DELAYED                                   |
| 29a30                                        | 21:30                             |                         | DELAYED                                   |
| 01a08                                        | 22:30                             |                         | DELAYED                                   |

# Flight Delays Problem

---

- Every year approximately 20% of airline flights are delayed or cancelled, resulting in significant costs to both travelers and airlines.
- As our example use-case, we will build a supervised learning model that predicts airline delay from historical flight data and weather information.
- This dataset includes details about flights in the US from the years 1987-2008. Every row in the dataset includes 29 variables

# Case study: model airline delays

---

- <http://hortonworks.com/blog/data-science-hadoop-spark-scala-part-2/>
- <http://nbviewer.ipython.org/github/ofermend/IPython-notebooks/blob/master/blog-part-2.ipynb>
- **We consider flight delays of 15 minutes or more as "delays" and mark it with a label of 1.0, and under 15 minutes as "non-delay" and mark it with a label of 0.0.**
- **Standardize the data**
- **Join in weather data with flight data**

# Problem Scope: ORD delays predictions for 2007 (TRAIN), 2008 (Test)

---

- To simplify, we will build a supervised learning model to predict flight delays for flights leaving O'Hare International airport (ORD), where we "learn" the model using data from 2007, and evaluate its performance using data from 2008.
- Process
  - EDA
  - Extract features
  - Standardize the data
  - Learn models and evaluate using F1 measure
  - Join in weather data with flight data
  - Learn models and evaluate using F1 measure

Let's begin by exploring the airline delay dataset available here: <http://stat-computing.org/dataexpo/2009/the-data.html> This dataset includes details about flights in the US from the years 1987-2008. Every row in the dataset includes 29 variables:

|    | Name              | Description                                                               |
|----|-------------------|---------------------------------------------------------------------------|
| 1  | Year              | 1987-2008                                                                 |
| 2  | Month             | 1-12                                                                      |
| 3  | DayofMonth        | 1-31                                                                      |
| 4  | DayOfWeek         | 1 (Monday) - 7 (Sunday)                                                   |
| 5  | DepTime           | actual departure time (local, hhmm)                                       |
| 6  | CRSDepTime        | scheduled departure time (local, hhmm)                                    |
| 7  | ArrTime           | actual arrival time (local, hhmm)                                         |
| 8  | CRSArrTime        | scheduled arrival time (local, hhmm)                                      |
| 9  | UniqueCarrier     | unique carrier code                                                       |
| 10 | FlightNum         | flight number                                                             |
| 11 | TailNum           | plane tail number                                                         |
| 12 | ActualElapsedTime | in minutes                                                                |
| 13 | CRSElapsedTime    | in minutes                                                                |
| 14 | AirTime           | in minutes                                                                |
| 15 | ArrDelay          | arrival delay, in minutes                                                 |
| 16 | DepDelay          | departure delay, in minutes                                               |
| 17 | Origin            | origin                                                                    |
| 18 | Dest              | destination                                                               |
| 19 | Distance          | in miles                                                                  |
| 20 | TaxiIn            | taxi in time, in minutes                                                  |
| 21 | TaxiOut           | taxi out time in minutes                                                  |
| 22 | Cancelled         | was the flight cancelled?                                                 |
| 23 | CancellationCode  | reason for cancellation (A = carrier, B = weather, C = NAS, D = security) |
| 24 | Diverted          | 1 – yes 0 – no                                                            |

**PLUS**

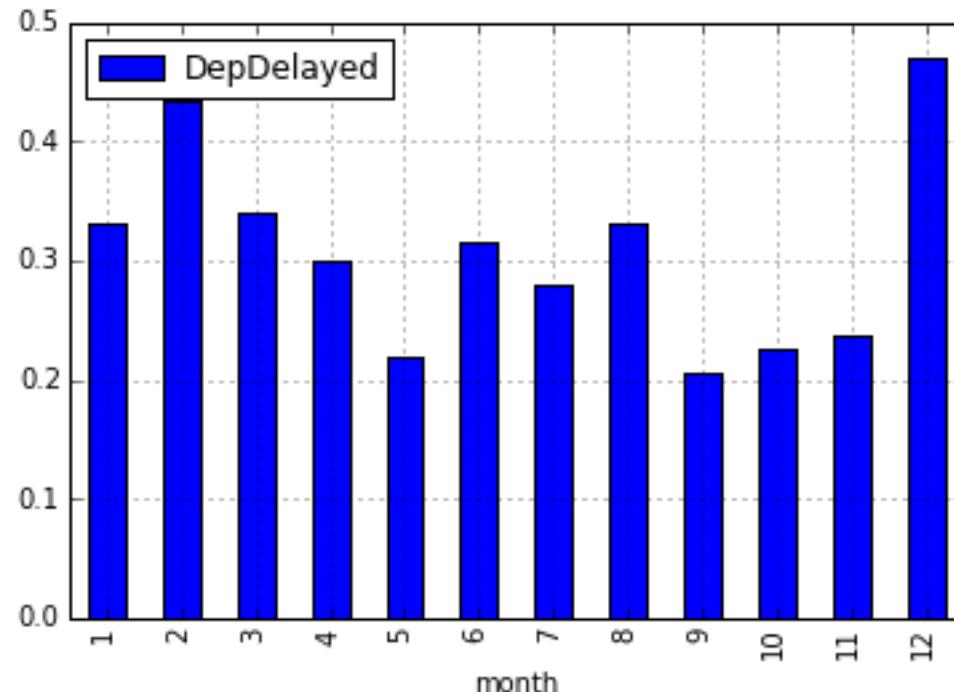
**Weather data that can  
be joined in**

# EDA in python

---

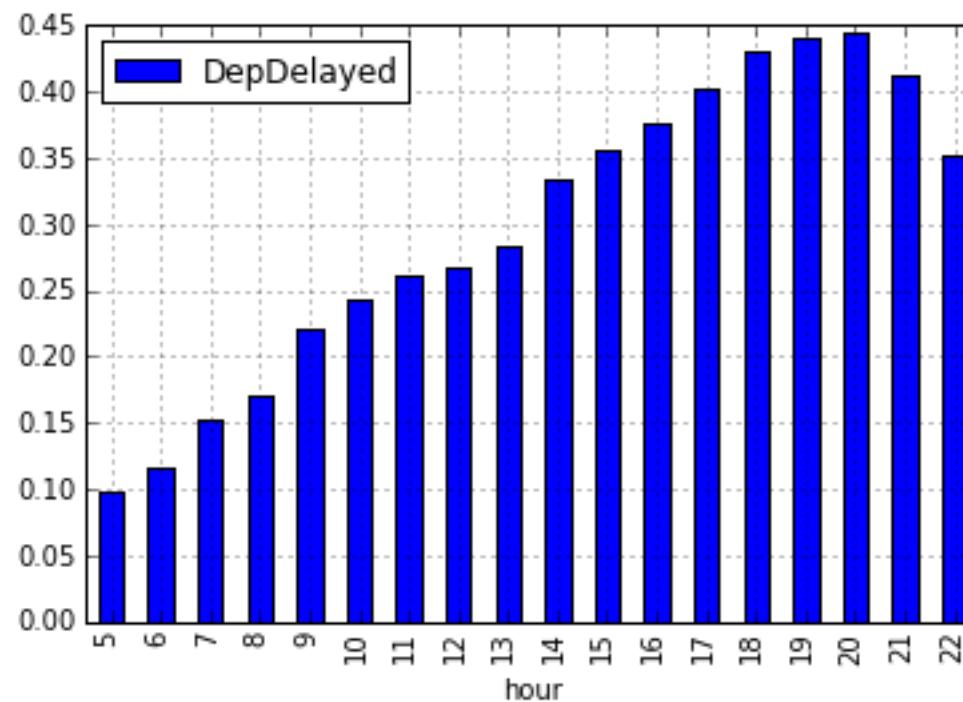
- <http://nbviewer.ipython.org/github/ofermend/IPython-notebooks/blob/master/blog-part-1.ipynb>

average number of delayed flights per month

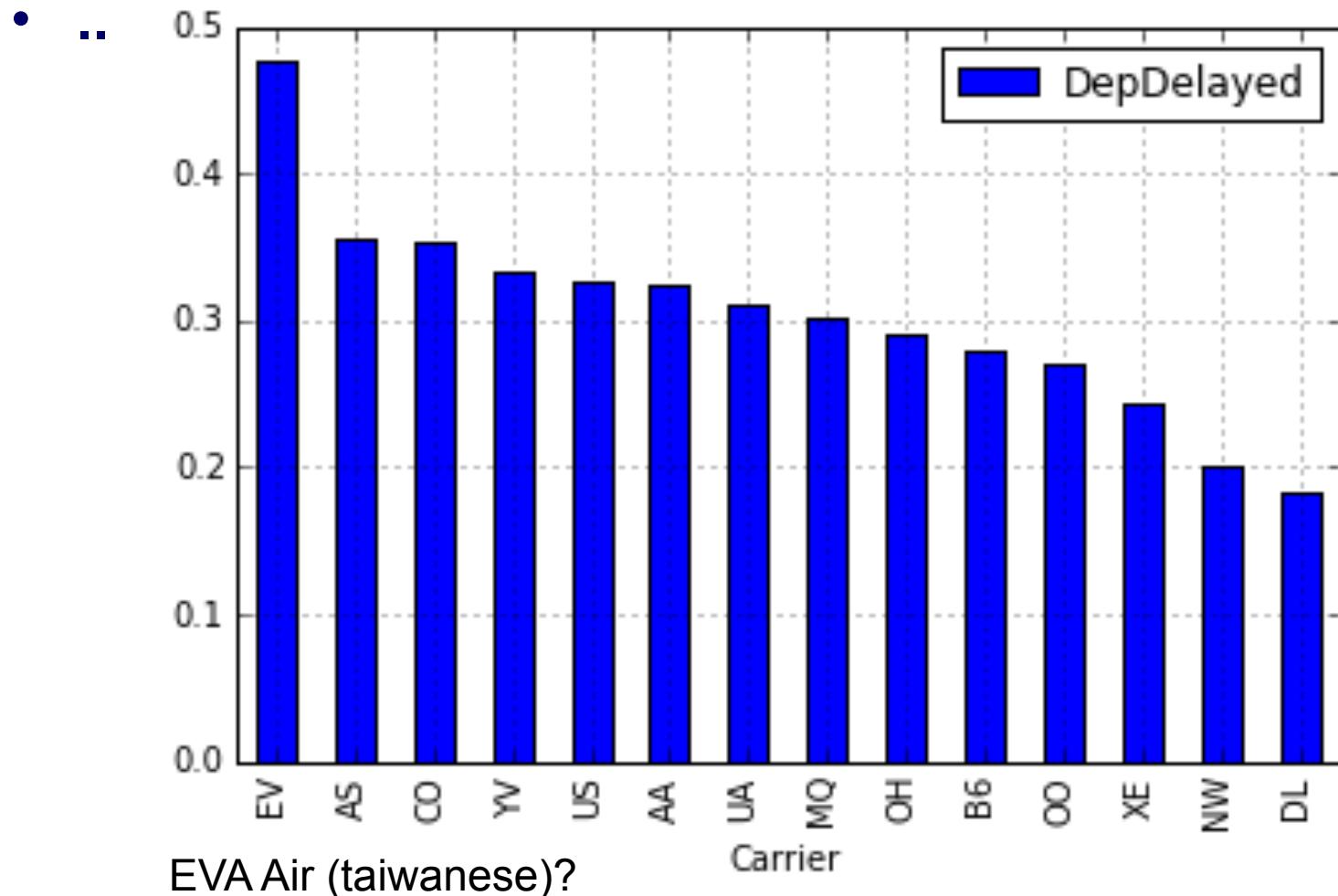


# Avg number of flights delay by hour of day

---



## Average number of delayed flights per carrier



EVA Air (taiwanese)?

# Feature generation

---

After exploring the data for a bit, we now move to building the feature matrix for our predictive model.

Let's look at possible predictive variables for our model:

- **month**: winter months should have more delays than summer months
- **day of month**: this is likely not a very predictive variable, but let's keep it in anyway
- **day of week**: weekend vs. weekday
- **hour of the day**: later hours tend to have more delays
- **Carrier**: we might expect some carriers to be more prone to delays than others
- **Destination airport**: we expect some airports to be more prone to delays than others
- **Distance**: interesting to see if this variable is a good predictor of delay

We will also generate another feature: number of days from closest national holiday, with the assumption that holidays tend to be associated with more delays.

# In Scala (typed language)

We also use ML-Lib's *StandardScaler* class to normalize our feature values for both training and validation sets. This is important because of ML-Lib's use of Stochastic Gradient Descent, which is known to perform best if feature vectors are normalized.

```
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.feature.StandardScaler

def parseData(vals: Array[Double]): LabeledPoint = {
 LabeledPoint(if (vals(0)>=15) 1.0 else 0.0, Vectors.dense(vals.drop(1)))
}

// Prepare training set
val parsedTrainData = data_2007.map(parseData)
parsedTrainData.cache
val scaler = new StandardScaler(withMean = true, withStd = true).fit(parsedTrainData.map(x => x.features))
val scaledTrainData = parsedTrainData.map(x => LabeledPoint(x.label, scaler.transform(Vectors.dense(x.features.toArray))))
scaledTrainData.cache

// Prepare test/validation set
val parsedTestData = data_2008.map(parseData)
parsedTestData.cache
val scaledTestData = parsedTestData.map(x => LabeledPoint(x.label, scaler.transform(Vectors.dense(x.features.toArray))))
scaledTestData.cache

scaledTrainData.take(3).map(x => (x.label, x.features)).foreach(println)

(0.0,[-1.6160463330366548,1.054927299466599,0.03217026353736381,-0.5189244175441321,0.034083933424313526,-0.280168
3099466359])
(1.0,[-1.6160463330366548,1.3961052168540333,1.5354307758475527,0.3624320984120952,0.43165511884343954,-0.02327388
7437334728])
(1.0,[-1.6160463330366548,1.5098311893165113,-1.4710902487728252,1.4641277433573794,-0.7436888225169864,0.06235758
673243232])
```

# Calculate Metrics

---

We also define a helper function to evaluate our metrics: precision, recall, accuracy and the F1-measure

```
: // Function to compute evaluation metrics
def eval_metrics(labelsAndPreds: RDD[(Double, Double)]) : Tuple2[Array[Double], Array[Double]] = {
 val tp = labelsAndPreds.filter(r => r._1==1 && r._2==1).count.toDouble
 val tn = labelsAndPreds.filter(r => r._1==0 && r._2==0).count.toDouble
 val fp = labelsAndPreds.filter(r => r._1==1 && r._2==0).count.toDouble
 val fn = labelsAndPreds.filter(r => r._1==0 && r._2==1).count.toDouble

 val precision = tp / (tp+fp)
 val recall = tp / (tp+fn)
 val F_measure = 2*precision*recall / (precision+recall)
 val accuracy = (tp+tn) / (tp+tn+fp+fn)
 new Tuple2(Array(tp, tn, fp, fn), Array(precision, recall, F_measure, accuracy))
}
```

# Logistic Regression

```
: import org.apache.spark.mllib.classification.LogisticRegressionWithSGD

// Build the Logistic Regression model
val model_lr = LogisticRegressionWithSGD.train(scaledTrainData, numIterations=100)

// Predict
val labelsAndPreds_lr = scaledTestData.map { point =>
 val pred = model_lr.predict(point.features)
 (pred, point.label)
}
val m_lr = eval_metrics(labelsAndPreds_lr)._2
println("precision = %.2f, recall = %.2f, F1 = %.2f, accuracy = %.2f".format(m_lr(0), m_lr(1), m_lr(2), m_lr(3)))

precision = 0.37, recall = 0.64, F1 = 0.47, accuracy = 0.59
```

We have built a model using Logistic Regression with SGD using 100 iterations, and then used it to predict flight delays over the validation set to measure performance: precision, recall, F1 and accuracy.

# SVMs

Next, let's try a Support Vector Machine model:

```
import org.apache.spark.mllib.classification.SVMWithSGD

// Build the SVM model
val svmAlg = new SVMWithSGD()
svmAlg.optimizer.setNumIterations(100)
 .setRegParam(1.0)
 .setStepSize(1.0)
val model_svm = svmAlg.run(scaledTrainData)

// Predict
val labelsAndPreds_svm = scaledTestData.map { point =>
 val pred = model_svm.predict(point.features)
 (pred, point.label)
}
val m_svm = eval_metrics(labelsAndPreds_svm)._2
println("precision = %.2f, recall = %.2f, F1 = %.2f, accuracy = %.2f".format(m_svm(0), m_svm(1), m_svm(2), m_svm(3)))

precision = 0.37, recall = 0.64, F1 = 0.47, accuracy = 0.59
```

# Decision Trees

Since ML-Lib also has a strong Decision Tree implementation, let's use it here:

```
import org.apache.spark.mllib.tree.DecisionTree

// Build the Decision Tree model
val numClasses = 2
val categoricalFeaturesInfo = Map[Int, Int]()
val impurity = "gini"
val maxDepth = 10
val maxBins = 100
val model_dt = DecisionTree.trainClassifier(parsedTrainData, numClasses,
, categoricalFeaturesInfo, impurity, maxDepth, maxBins)

// Predict
val labelsAndPreds_dt = parsedTestData.map { point =>
 val pred = model_dt.predict(point.features)
 (pred, point.label)
}
val m_dt = eval_metrics(labelsAndPreds_dt)._2
println("precision = %.2f, recall = %.2f, F1 = %.2f, accuracy = %.2f".format(m_dt(0), m_dt(1), m_dt(2), m_dt(3)))

precision = 0.41, recall = 0.24, F1 = 0.31, accuracy = 0.69
```

*Lar* Note that overall accuracy is higher with the Decision Tree, but the precision and recall trade-off

# Join in Weather features

---

- To accomplish this with Apache Spark, we rewrite our previous preprocess\_spark function to extract the same base features from the flight delay dataset, and also join those with five variables from the weather datasets: minimum and maximum temperature for the day, precipitation, snow and wind speed. Let's see how this is accomplished.

# Join data

```
def filterMap(wdata:RDD[Array[String]], metric:String):RDD[(String,Double)] = {
 wdata.filter(vals => vals(metric_inx) == metric).map(vals => (vals(date_inx), vals(value_inx).toDouble))
}

val wdata = sc.textFile(weather_file).map(line => line.split(","))
 .filter(vals => vals(station_inx) == "USW00094846")
val w_tmin = filterMap(wdata, "TMIN")
val w_tmax = filterMap(wdata, "TMAX")
val w_prcp = filterMap(wdata, "PRCP")
val w_snow = filterMap(wdata, "SNOW")
val w_awnd = filterMap(wdata, "AWND")

delayRecs.join(w_tmin).map(vals => (vals._1, vals._2._1 ++ Array(vals._2._2)))
 .join(w_tmax).map(vals => (vals._1, vals._2._1 ++ Array(vals._2._2)))
 .join(w_prcp).map(vals => (vals._1, vals._2._1 ++ Array(vals._2._2)))
 .join(w_snow).map(vals => (vals._1, vals._2._1 ++ Array(vals._2._2)))
 .join(w_awnd).map(vals => vals._2._1 ++ Array(vals._2._2))
}
```

We are going to repeat the previous models of SVM and Decision Tree with our enriched feature set. As before, we create an RDD of *LabeledPoint* objects, and normalize our dataset with ML-Lib's *StandardScaler*:

```
: import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.feature.StandardScaler

def parseData(vals: Array[Double]): LabeledPoint = {
 LabeledPoint(if (vals(0)>=15) 1.0 else 0.0, Vectors.dense(vals.drop(1)))
}

// Prepare training set
val parsedTrainData = data_2007.map(parseData)
val scaler = new StandardScaler(withMean = true, withStd = true).fit(parsedTrainData.map(x => x.features))
val scaledTrainData = parsedTrainData.map(x => LabeledPoint(x.label, scaler.transform(Vectors.dense(x.features.toArray))))
parsedTrainData.cache
scaledTrainData.cache

// Prepare test/validation set
val parsedTestData = data_2008.map(parseData)
val scaledTestData = parsedTestData.map(x => LabeledPoint(x.label, scaler.transform(Vectors.dense(x.features.toArray))))
parsedTestData.cache
scaledTestData.cache

scaledTrainData.take(3).map(x => (x.label, x.features)).foreach(println)
```

Next, let's build a Support Vector Machine model using this enriched feature matrix:

```
import org.apache.spark.mllib.classification.SVMWithSGD
import org.apache.spark.mllib.optimization.L1Updater

// Build the SVM model
val svmAlg = new SVMWithSGD()
 .setNumIterations(100)
 .setRegParam(1.0)
 .setStepSize(1.0)
val model_svm = svmAlg.run(scaledTrainData)

// Predict
val labelsAndPreds_svm = scaledTestData.map { point =>
 val pred = model_svm.predict(point.features)
 (pred, point.label)
}
val m_svm = eval_metrics(labelsAndPreds_svm)._2
println("precision = %.2f, recall = %.2f, F1 = %.2f, accuracy = %.2f".format(m_svm(0), m_svm(1), m_svm(2), m_svm(3)))

precision = 0.39, recall = 0.68, F1 = 0.49, accuracy = 0.61
```

# Weather

---

- F1 performance improves with the weather features from 0.47 to 0.49

# Challenge: set up pipeline for flights data

---

- **In PySpark**
  - Set pipelines and compare SVMs, Logistic regression etc.
- **In SparkR**
  - Set pipelines and compare SVMs, Logistic regression etc.

# Part 3: Machine Learning in Spark

---

- **Data Frames**
- **MLLib**
- **Write your own algos**
  - Linear regression (Ridge and Lasso)
  - Logistic Regression
- **Pipelines**
- **R and Spark**
  - Linear Regression example
- **Graphs in Spark**
- **Case studies**
  - Flight delay
  - Mobile advertising
  - Social Networks

---

- **Spark in the Realworld**

- NativeX
- Tencent

# NativeX: Mobile ad network

---

- **Mobile advertising**
  - Show ad in an app
  - Predict whether user installs app or not
- **30 Gig of training (conversion rate modeling using decision trees ensembles)**
- **30 minutes on 5 node cluster**

# Distributed Data Warehouse) Overview

## Tencent, STRATA 2014

The diagram illustrates the architecture of the Distributed Data Warehouse, structured into layers:

- User
- IDE
- Lhoste (Workflow system)
- Hive/Pig
- MapReduce
- Spark
- Storm
- Docker
- GAIA (Based on YARN)
- HDFS

Key features highlighted on the right side of the diagram include:

- Nodes of Gaia cluster: 8000+
- Storage capacity of HDFS: 150PB+
- Daily increased data: 1PB+
- Daily jobs: 1M+
- Daily Computation: 10PB+

**IDE:** Interface for submitting SQL or Scripts.

**Lhoste:** Workflow scheduler system for jobs, similar as Oozie.

**GAIA:** Our optimized Resource Management System based on YARN

# Spark at Tencent

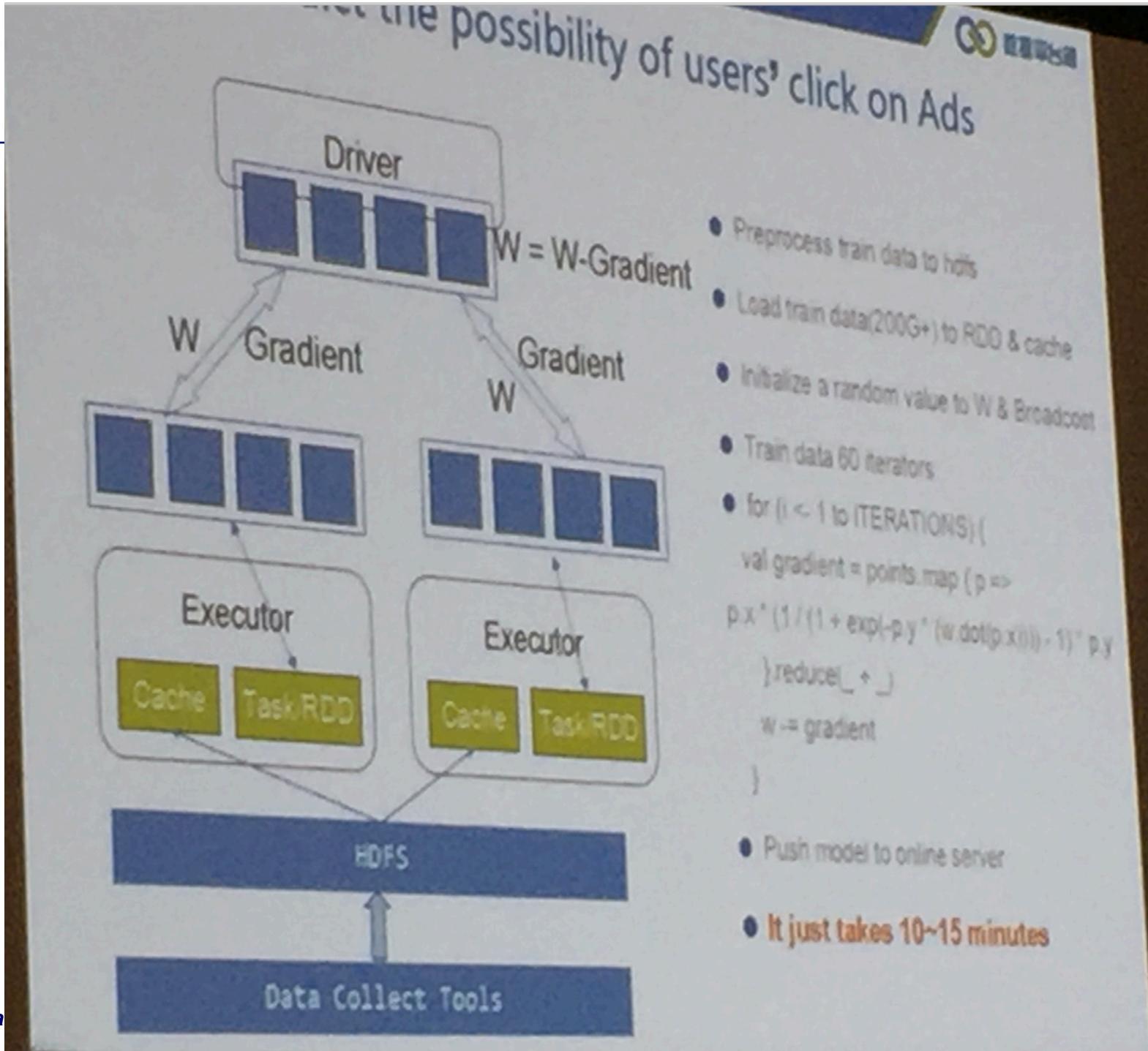
- Type of jobs: ETL, SparkSQL, Machine Learning, Graph Compute, Streaming
- Daily jobs: 10K+
- Running on : Gaia(8000+ Nodes, with 24 cores and 60G memory each)
- Support most of data storage system: HDFS/Hive/HBase/MySQL ...
- Start from 2013's Spark 0.6 version, now our version is Spark 1.2



# Useages & Effects of Spark at Tencent



- Case 1: Predict the possibility of users' click on Ads
- Case 2: Calculate mutual friends between users
- Case 3: SparkSQL or many DAG jobs for ETL



## Case 2: Calculate mutual friends between users

- According to data size of shuffle, determine number of partition
- Use sort-based shuffle to reduce memory usage
- When connection is timeout, it will be considered as lost(Spark-1.2)
- To avoid executor to be killed by GC, set spark.yarn.executor.memoryOverhead

| edges/vertex | Nodes | Time  | Memory      |
|--------------|-------|-------|-------------|
| 50b/0.5b     | 200   | ~1m   | 15G/n/5000  |
| 50b/0.5b     | 1000  | ~1.5m | 15G/n/5000  |
| 100b/1b      | 200   | ~24h  | 256/n/10000 |
| 100b/1b      | 1000  | ~24h  | 256/n/10000 |

One Node, 50G memory, 2\*12T SATA disk, 24\*1.9GHz cpu, 1Gbps

2014

**Mutual friends in Hive  
Possibly used  
subsequently as a  
feature in an ML algo  
Tencent, STRATA**

# Other Spark Applications

---

- See Spark Summit 2015, 2014 (West Coast and East Coast)
- Twitter spam classification (Justin Ma)
- EM alg. for traffic prediction (Mobile Millennium)
- K-means clustering
- Alternating Least Squares matrix factorization

# Tutorial Outline

- **Part 1: Introduction**
  - Welcome Survey
  - Install Spark
  - Background and motivation
- **Part 2: Spark Intro and basics**
  - fundamental Spark concepts, including Spark Core, data frames, the Spark Shell, Spark Streaming, Spark SQL and vertical libraries such as MLlib and GraphX;
- **Part 3: Machine learning in Spark**
  - will focus on hands-on algorithmic design and development with Spark developing algorithms from scratch such as linear regression, logistic regression, graph processing algorithms such as pagerank/shortest path, etc.
- **Part 4: Wrap up**
  - Spark 1.5 and beyond
  - Summary

- 
- **Whats next for Spark?**

# Spark 1.5+

---

<http://www.slideshare.net/databricks/spark-community-update-spark-summit-san-francisco-2015>

- **Project Tungsten**
  - Memory Management and Binary Processing: leveraging application semantics to manage memory explicitly and eliminate the overhead of JVM object model and garbage collection
  - *Cache-aware computation*: algorithms and data structures to exploit memory hierarchy
  - *Code generation*: using code generation to exploit modern compilers and CPUs
- **Spark Streaming: Flow control, optimized stare management**
- **MLlib: Single machine solvers, scalability to many features**
- **SparkR: Integration with Spark's machine learning APIs**

---

# Deep Dive into Project Tungsten: Bringing Spark Closer to Bare Metal

Josh Rosen ([@jshrsn](https://twitter.com/jshrsn))

June 16, 2015



# Project Tungsten Roadmap

## Spark 1.4

- Binary processing for aggregation in Spark SQL / DataFrames
- New Tungsten shuffle manager
- Compression & serialization optimizations

## Spark 1.5

- Optimized code generation
- Optimized sorting in Spark SQL / DataFrames
- End-to-end processing using binary data representations
- External aggregation

## Spark 1.6

- Vectorized / batched processing
- ???

# Model Scaling

Linear models only need to compute the dot product of each example with model

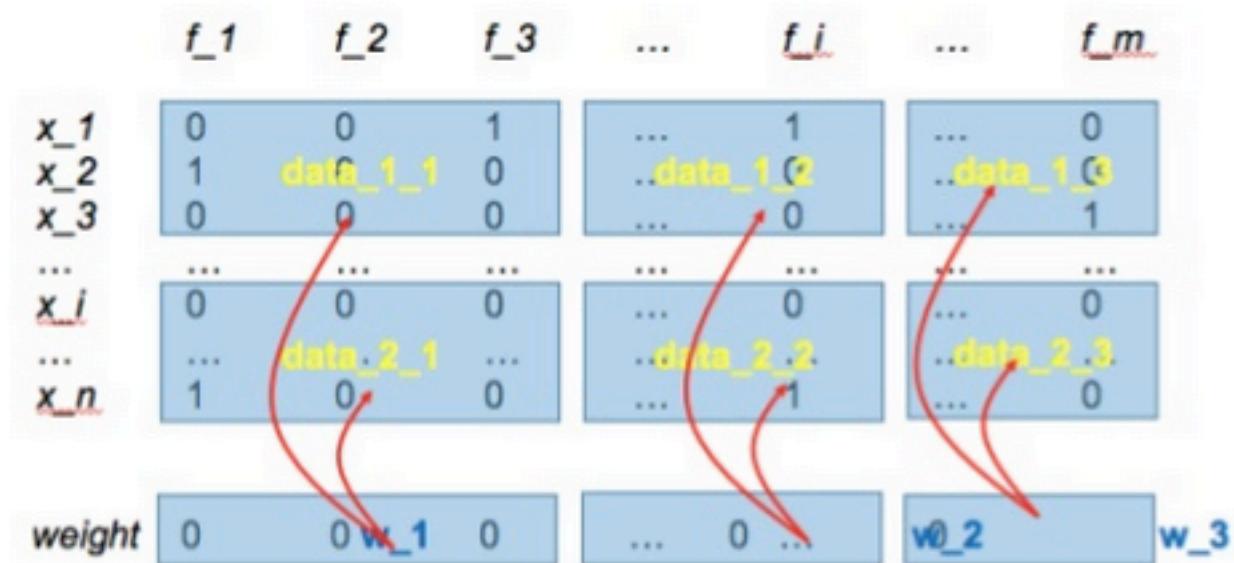
Use a BlockMatrix to store data, use joins to compute dot products

Coming in 1.5

# Each record has 10Billion values

- Model Scaling

Data joined with model (weight):



# MLlib 1.5+

---

- **Log-linear model for survival analysis**

- R-like stats for ML models

- A/B testing

- Pipeline API support

- K-means

- Naïve Bayes

- Isotonic regression

- Model save/load

- Pipeline persistence

- Python API for streaming ML algorithms

- MLlib + SparkR integration

[https://issues.apache.org/jira/browse/SPARK-9658?jql=project%20%3D%20SPARK%20AND%20status%20in%20\(Open%2C%20%22In%20Progress%22%2C%20Reopened\)%20AND%20component%20in%20\(ML%2C%20MLlib\)%20AND%20%22Target%20Version%2Fs%22%20%3D%201.5.0%20ORDER%20BY%20priority%20DESC](https://issues.apache.org/jira/browse/SPARK-9658?jql=project%20%3D%20SPARK%20AND%20status%20in%20(Open%2C%20%22In%20Progress%22%2C%20Reopened)%20AND%20component%20in%20(ML%2C%20MLlib)%20AND%20%22Target%20Version%2Fs%22%20%3D%201.5.0%20ORDER%20BY%20priority%20DESC)

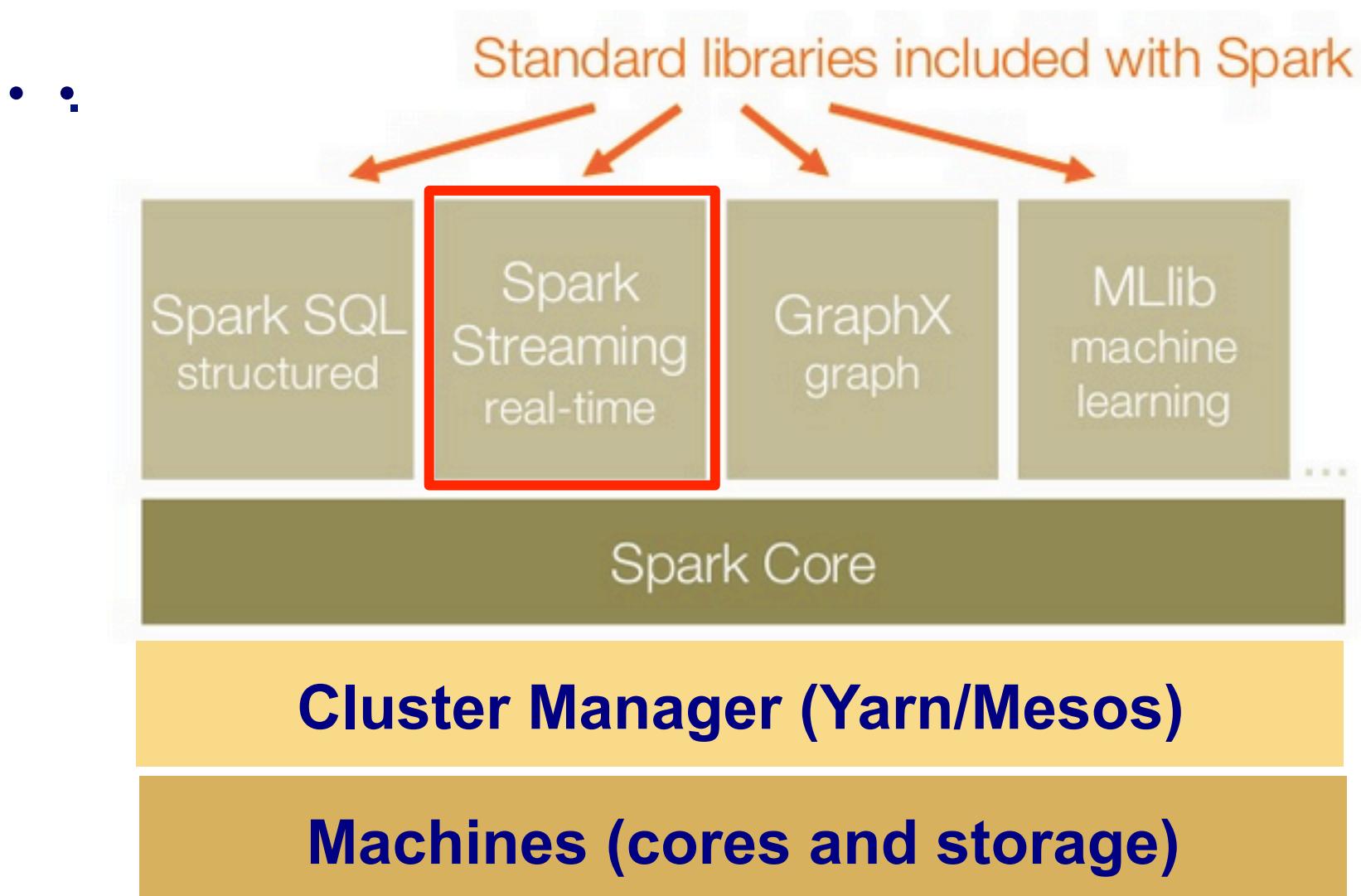
<https://issues.apache.org/jira/browse/SPARK-8445>

# Other1.5+

---

- Core
  - HeartbeatReceiver should not adjust application executor resources
  - Allow additional uris to be fetched with mesos
  - Fix NullPointerException in error-handling path in UnsafeShuffleWriter
- Streaming
  - Add EventHubsReceiver to support Spark Streaming using Azure EventHubs
- SparkR
  - Add EventHubsReceiver to support Spark Streaming using Azure EventHubs

# Did not cover Spark Streaming



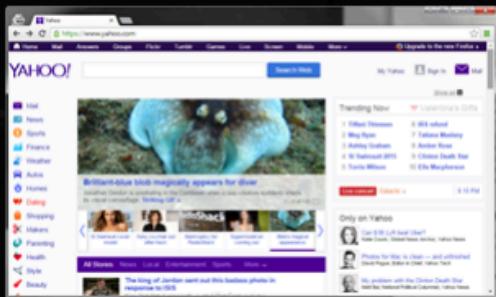
# Spark streaming

• ..

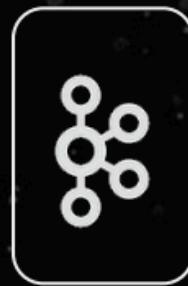


# USE CASES

(live statistics)



Page views

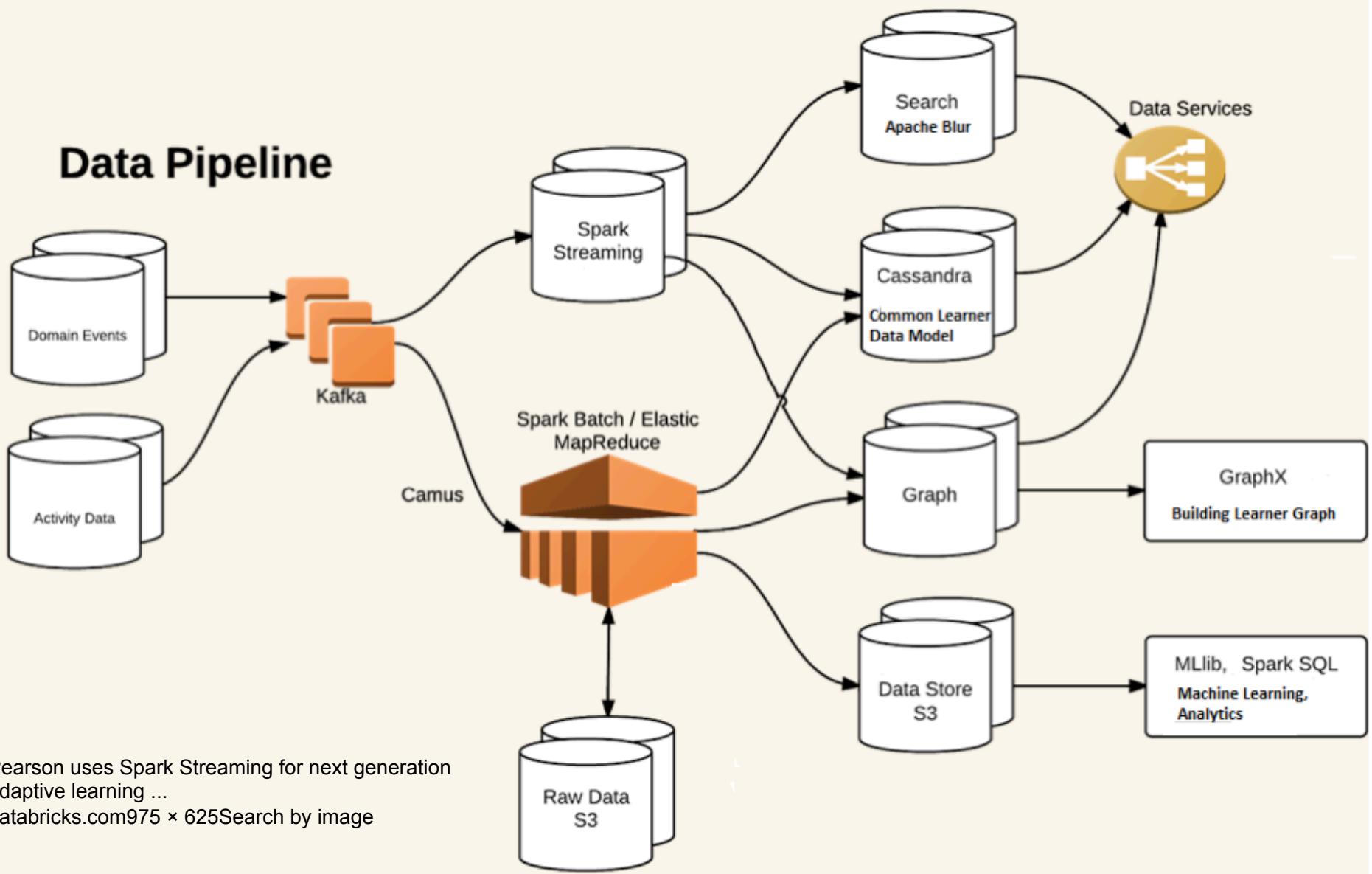


Kafka for buffering



Spark for processing

# Data Pipeline



A screenshot of a web browser window displaying the Spark Packages website ([spark-packages.org](http://spark-packages.org)). The page title is "Spark Packages". The navigation bar includes links for "Feedback", "Register a package", "Login", and "Find a package". A search bar is also present. The main content area features a heading "A community index of packages for Apache Spark." followed by a count of "33 packages". Below this, three package entries are listed:

- databricks/spark-avro**  
Integration utilities for using Spark with Apache Avro data  
@pwendell / Latest release: 0.1 (11/27/14) / Apache-2.0 / ★★★★★ (15)  
3 sql | 3 input | 2 library
- dibbhatt/kafka-spark-consumer**  
Low Level Kafka-Spark Consumer  
@dibbhatt / No release yet / ★★★★★ (3)  
2 streaming | 1 kafka
- sigmoidanalytics/spork**  
Pig on Apache Spark

At the bottom of the page, a footer note states: "Spark Packages is a community site hosting modules that are not part of Apache Spark. Your use of and access to this site is subject to the terms of use. Apache Spark and the Spark logo are trademarks of the Apache Software Foundation. This site is maintained as a community service by Databricks."

- 
- **Spark frameworks**

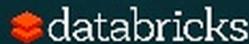
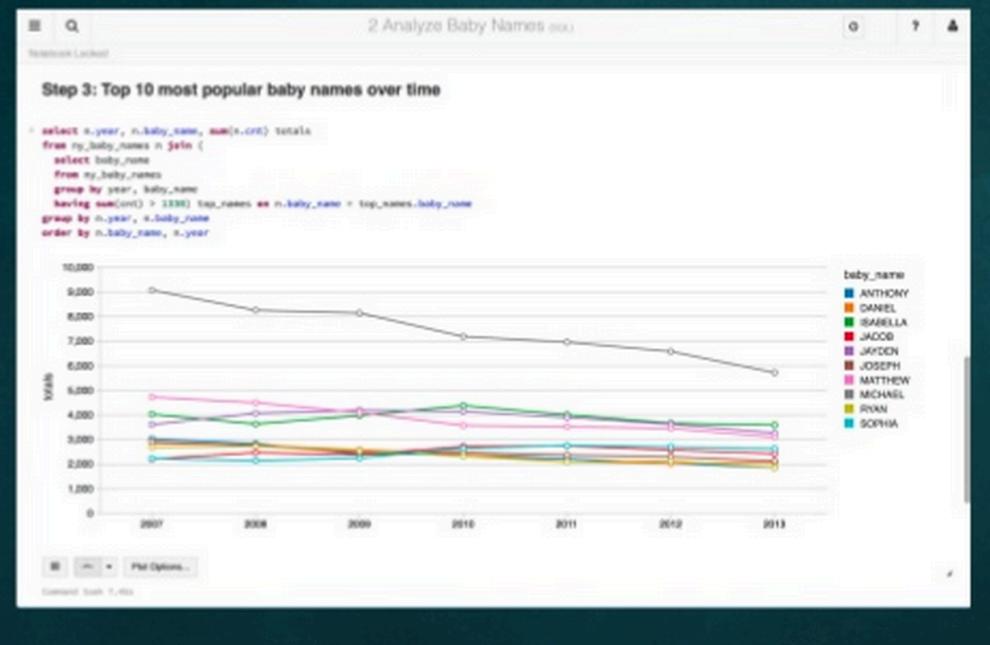
# About Databricks



Founded by creators of Spark and remains largest contributor

Offers a hosted service:

- Spark on EC2
- Notebooks
- Plot visualizations
- Cluster management
- Scheduled jobs



2

---

- **Summary**

---

# Spark

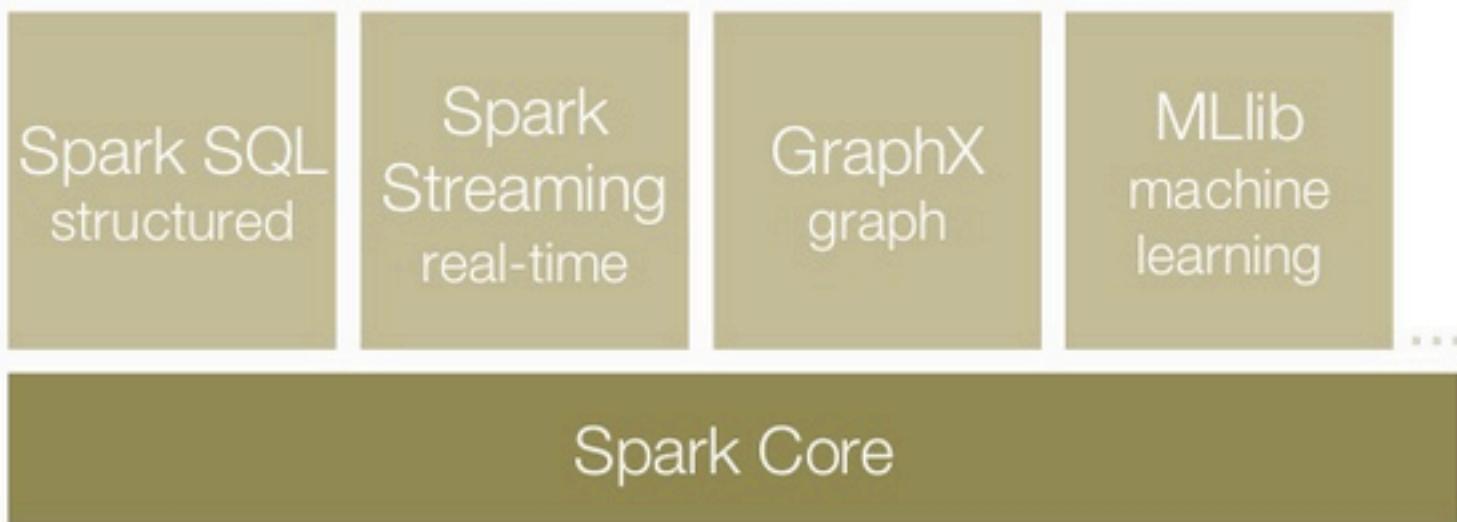
## In-Memory Cluster Computing for Iterative and Interactive Applications

UC Berkeley

# Spark

- 
- 

Standard libraries included with Spark



# What is Spark?

Fast and Expressive Cluster Computing System  
Compatible with Apache Hadoop

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

**2-5x** less code

## Efficient

- General execution graphs
- In-memory storage

## Usable

- Rich APIs in Java, Scala, Python, R (1.4)
- Interactive shell

# Example: data distributed on nodes

```
rdd1 = sc.textFile(D1)
```

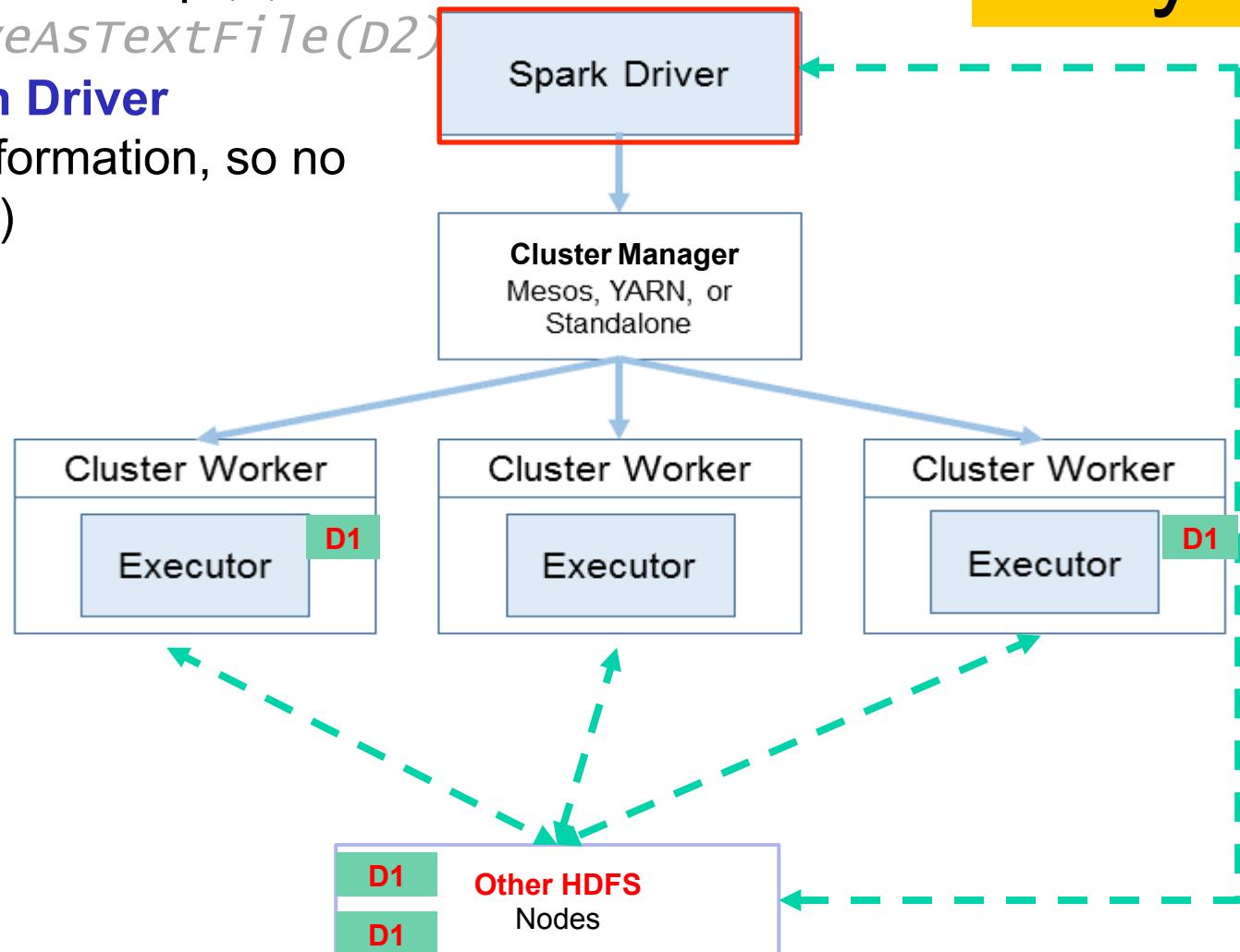
```
rdd2 = rdd1.map(...)
```

```
rdd2.saveAsTextFile(D2)
```

**Build DAG in Driver**

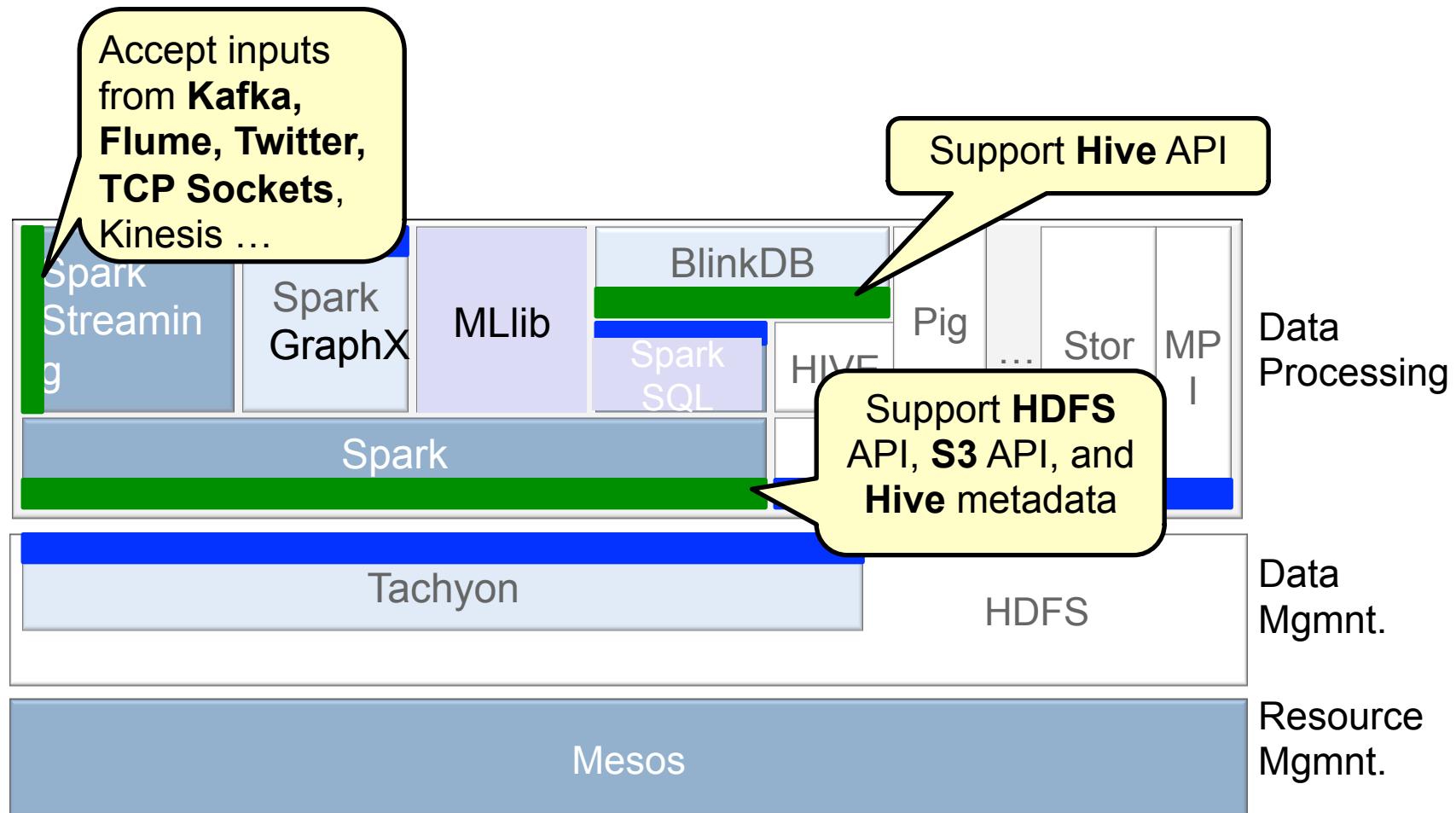
(map is transformation, so no execution yet)

**Lazy!!!**



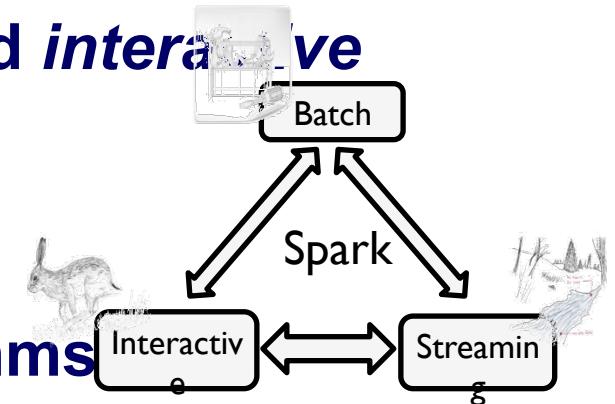
# Compatible with Open Source Ecosystem

- **Use existing interfaces whenever possible**



# Data Science in Spark

- **Support *interactive* and *streaming* computations**
  - In-memory, fault-tolerant storage abstraction, low-latency scheduling,...
- **Easy to combine *batch*, *streaming*, and *interactive* computations**
  - Spark execution engine supports all comp. models
- **Easy to develop *sophisticated* algorithms**
  - Scala interface, APIs for Java, Python, R, Hive QL, ... (**SparkSQL**)
  - New frameworks targeted to graph based (**GraphX**) and ML algorithms (**MLLib**)
- **Compatible with existing open source ecosystem**
- **Open source (Apache) and fully committed to release *high quality* software**



# Contrast different MapReduce Frameworks

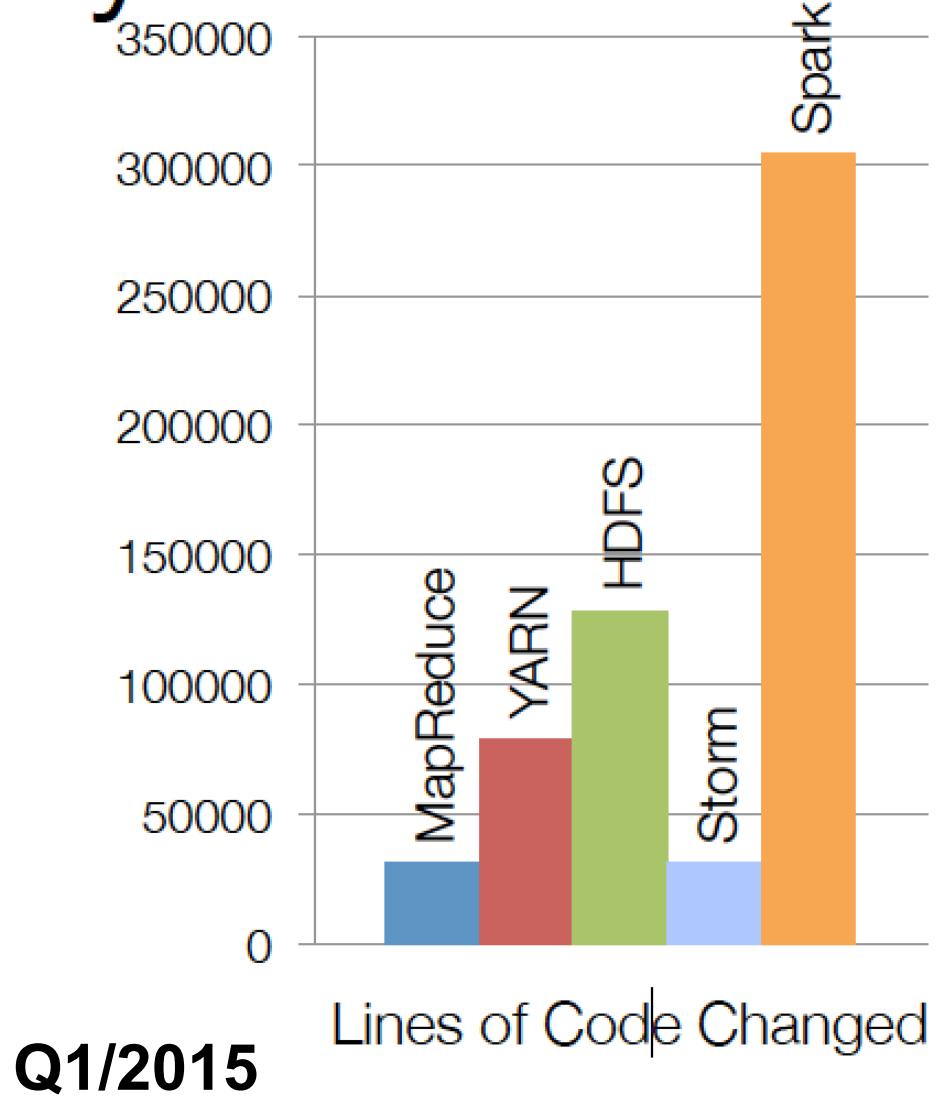
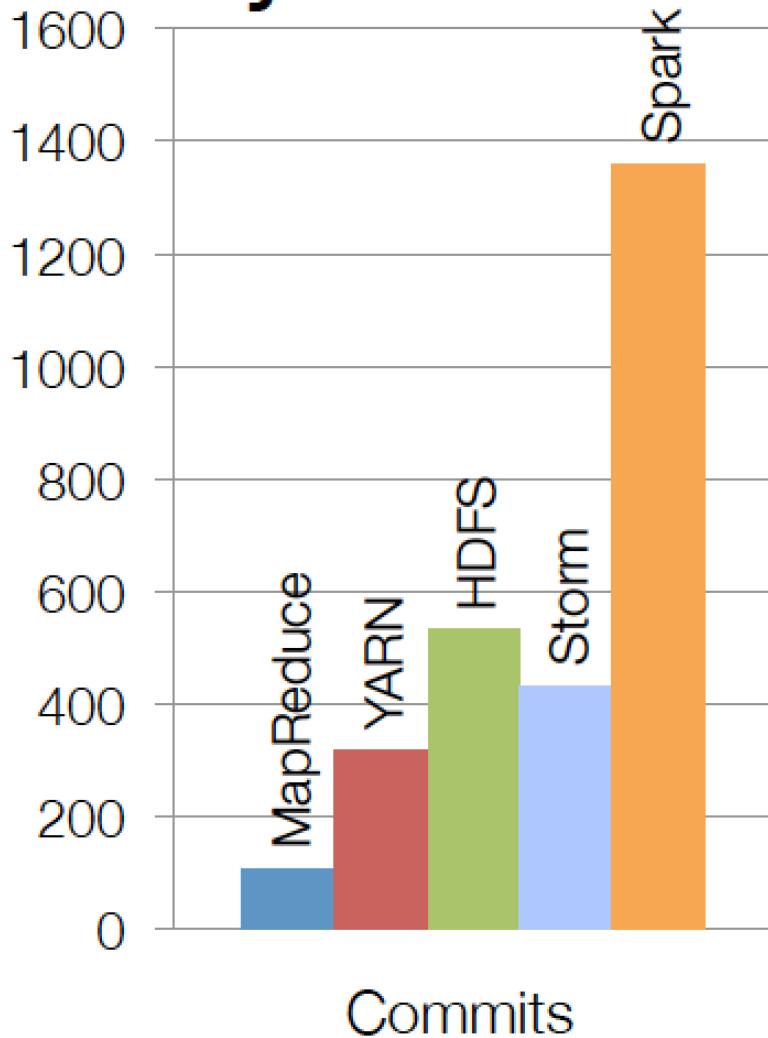
|                            | ML, Haskell | Hadoop MapReduce                         | Spark (Scala)                                 | PySpark                                       |
|----------------------------|-------------|------------------------------------------|-----------------------------------------------|-----------------------------------------------|
| Purely functional prog.    | Yes         | Minimally                                | Partially                                     | Partially                                     |
| Lazy evaluation            | Yes         | No                                       | Yes                                           | Yes                                           |
| Higher order functions     | Yes         | Yes                                      | Yes                                           | Yes                                           |
| Side effects               | No          | Yes                                      | Yes                                           | Yes                                           |
| Parallel Computation       |             |                                          |                                               |                                               |
| Fault tolerance            | No          | Yes                                      | Yes                                           | Yes                                           |
| Serialization              | No          | Yes for Java; no for Streaming           | Yes                                           | Partially                                     |
| Communication              | Barrier     | Barrier, Shuffle and Broadcast, Pipeline | Barrier, Shuffle and Broadcast, BSP, Pipeline | Barrier, Shuffle and Broadcast, BSP, Pipeline |
| Software Engineering       |             |                                          |                                               |                                               |
| Programming                | Yes         | Limited (e.g., no loops)                 | Yes                                           | Yes                                           |
| REPL                       | Yes         | No                                       | Yes                                           | Yes                                           |
| Big Data Processing        | No          | Yes                                      | Yes                                           | Yes                                           |
| Unified Big Data Framework | No          | No                                       | Yes                                           | Limited                                       |
| Custom Libraries           | No          | No                                       | Yes                                           | Limited                                       |

# ML Limitations

---

- Sequential algorithms (e.g., Perceptron)
- Nonlinear SVMs
- Lots of algorithms are bare bones implementations and need fleshing out (need your help!!)
- R API is very limited currently

# Project Activity



**Q1/2015**

Activity in past 6 months

# Full Day tutorial at CIKM 2015 in Melbourne, Monday, October 19, 2015

The screenshot shows a web browser window with the URL <http://www.cikm-2015.org/workshops-and-tutorials.php> in the address bar. The page header features a yellow banner with the text "69 days left until the Conference". Below the banner is the CIKM 2015 logo, which consists of a grid of blue squares overlaid by several colorful, abstract, radiating shapes in red, orange, purple, and green. To the right of the logo, the text "CIKM2015" is written in large, bold, black letters, followed by "19–23 October 2015 Melbourne Australia". Below this text are social media icons for Facebook and Twitter. The main content area of the page is white and contains the title "Large Scale Distributed Data Science using Apache Spark" in large, bold, green text. Below the title, the text "Full day" is written in a smaller, italicized black font. Underneath that, the names "James G. Shanahan & Liang Dai" are listed. The background of the page features a collage of images, including a close-up of a bird's head and a building.



# Join Us!

To apply, scan  
the QR code.



## About NativeX

NativeX is creating the leading ad technology for games. For developers who want to monetize with advertising that really works, NativeX is the marketing technology platform that is reinventing in-app advertising. We create more effective and engaging ad experiences that enable developers to build successful businesses around their apps. Here at NativeX, our data pipelines process tens of terabytes of data each day.



## Why work at NativeX?

To help you reach your potential, we offer tools and resources, such as classes, conference attendance and tuition reimbursement. You'll also find mentors eager to help you pursue your dreams. With offices in San Francisco, California, Minneapolis, Minnesota and Sartell, Minnesota -- NativeX offers top notch mentorship and on-the-job experience.

## Open positions (in San Francisco, CA or Minneapolis)

- **Junior Data Scientist**
- **Data Science Interns (Year 4+ of PhD)**
- **Java Engineers**