Week 10

Spark, MLib

James G. Shanahan [2]

Assistants: Liang Dai[1, 3]

[1]NativeX, [2]iSchool UC Berkeley, CA, [3]UC Santa Cruz

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

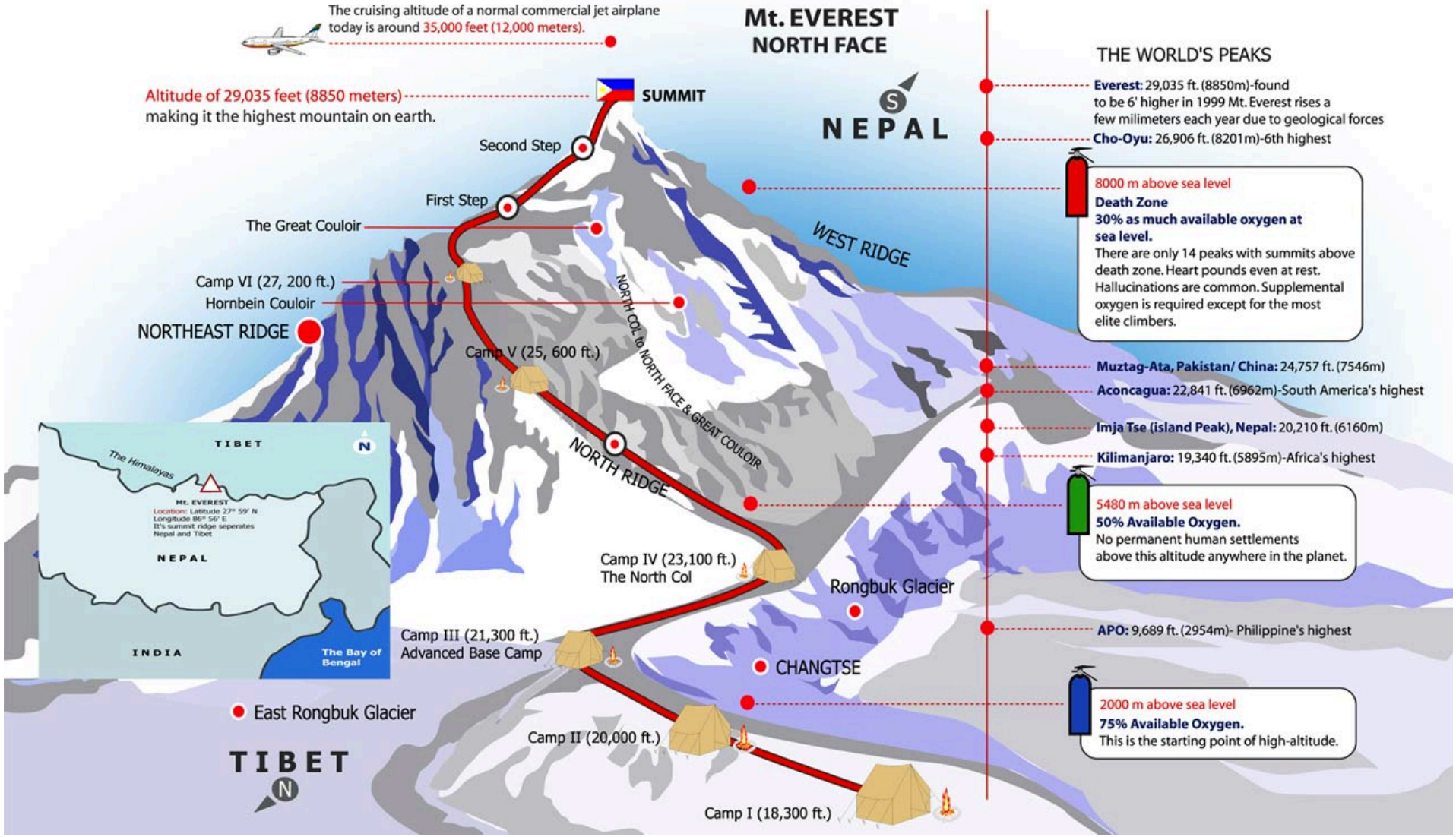
Live Session #10

March 15, 2016

Live Session Outline

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
- **Week**
  - Homework HW7, HW9, HW10
  - AWS: HW10 local
  - Running an MRJob on cloud with the latest version of MRJob, Screen
  - Async lecture recap plus Q&A Spark
  - Install Spark
  - Spark, RDDs, Patterns in Spark
  - DataFrames, Spark SQL
  - Secondary Sorts in Spark
  - MLLib: brief intro
  - Homegrown Kmeans in Spark
- **Wrapup**
  - Finish RECORDING (bonus points for reminding me!)

HW9 (Hilary Step!): Summiting Mt Everest



Important Links for Week 10

- **Live session Slides**

- **Instructions for Peer grading of homework HW**
  - https://www.dropbox.com/s/97m31frthj4ac28/HOMEWORK%20GRADING%20INSTRUCTIONS%20for%20MIDS%20MLS?dl=0
- **Homework HW Folder Questions + Data**
  - HW is a group oriented homework (Data is referenced via dropbox links)
  - https://www.dropbox.com/s/0vnj77z74piklsy/MIDS-MLS-HW-10.txt?dl=0
- **Team assignments**
  - https://docs.google.com/spreadsheets/d/1ncFQl5Tovn-16slD8mYjP_nzMTPSfiGeLLzW8v_sMjg/edit?usp=sharing
- **Please submit your homeworks (one per team) going forward via this form (and not thru the ISVC):**
  - https://docs.google.com/forms/d/1ZOr9RnIe_A06AcZDB6K1mJN4vrLeSmS2PD6Xm3eOiis/viewform?usp=send_form

Live Session Outline

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
- **Week**
  - Homework HW7, HW9, HW10
  - AWS: HW10 local
  - Running an MRJob on cloud with the latest version of MRJob, Screen
  - Async lecture recap plus Q&A Spark
  - Install Spark
  - Spark details
    - RDDs, Patterns in Spark, Combiners, Debugging, Broadcast variables, Flatmaps, Partitions
  - DataFrames, Spark SQL
  - Secondary Sorts in Spark
  - MLLib: brief intro
  - Homegrown Kmeans in Spark
- **Wrapup**
  - Finish RECORDING (bonus points for reminding me!)

13 Lectures, 1 Midterm, End of term exam, plus weekly homework and projects

Semester Week	Week in 2016	Monday	Notes	
1	Week 3	1/11/16		
2	Week 4	1/18/16		
3	Week 5	1/25/16		
4	Week 6	2/1/16		
5	Week 7	2/8/16		
6	Week 8	2/15/16		
7	Week 9	2/22/16		
8	Week 10	2/29/16	Mid Term Exam	
9	Week 11	3/7/16		
10	Week 12	3/14/16		
Spring Break	Week 13	3/21/16	Spring Break	
11	Week 14	3/28/16		
12	Week 15	4/4/16		
13	Week 16	4/11/16		
14	Week 17	4/18/16		
15	Week 18	4/25/16	End of term	4/29/15
	Week 19	5/2/16		
		5/18/2016 (Wednesday)		Grades Due

Schedule and HW Schedule

- **Spring Break week (Mar 22-25). No Class!**
  - During this week there will be no class and there is no planned office hours. The academic calendar can be found the I School website here:
  - https://www.ischool.berkeley.edu/intranet/students/mids/calendar
- **Homework Schedule:**
  - Given the spring break timing, and the fact that some of you requested extra time for HW7, I propose the following homework schedule:
  - **HW7 Peer Grading is due this Saturday (March 19) at 8:00AM**
  - **HW9 is due Saturday (March 19) at midday**
  - **HW10 is due Tuesday (March 29) at 8AM**

Conferences where you can learn more

- **Have a look at the following webpage for ideas:** http://www.kdnuggets.com/meetings/
- **Here is a short list to get this discussion underway:**
  - KDD (in San Francisco this year)
  - NIPS (Every December)
  - Spark Summit Series
  - WWW (Montreal, April, 2016)
  - http://cvpr2016.thecvf.com/ (Computer vision conference)
  - MLConf (San Francisco, December)

Live Session Outline

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
- **Week**
  - Homework HW7, HW9, HW10
  - AWS: HW10 local
  - Running an MRJob on cloud with the latest version of MRJob, Screen
  - Async lecture recap plus Q&A Spark
  - Install Spark
  - Spark, RDDs, Patterns in Spark
  - DataFrames, Spark SQL
  - Secondary Sorts in Spark
  - MLLib: brief intro
  - Homegrown Kmeans in Spark
- **Wrapup**
  - Finish RECORDING (bonus points for reminding me!)

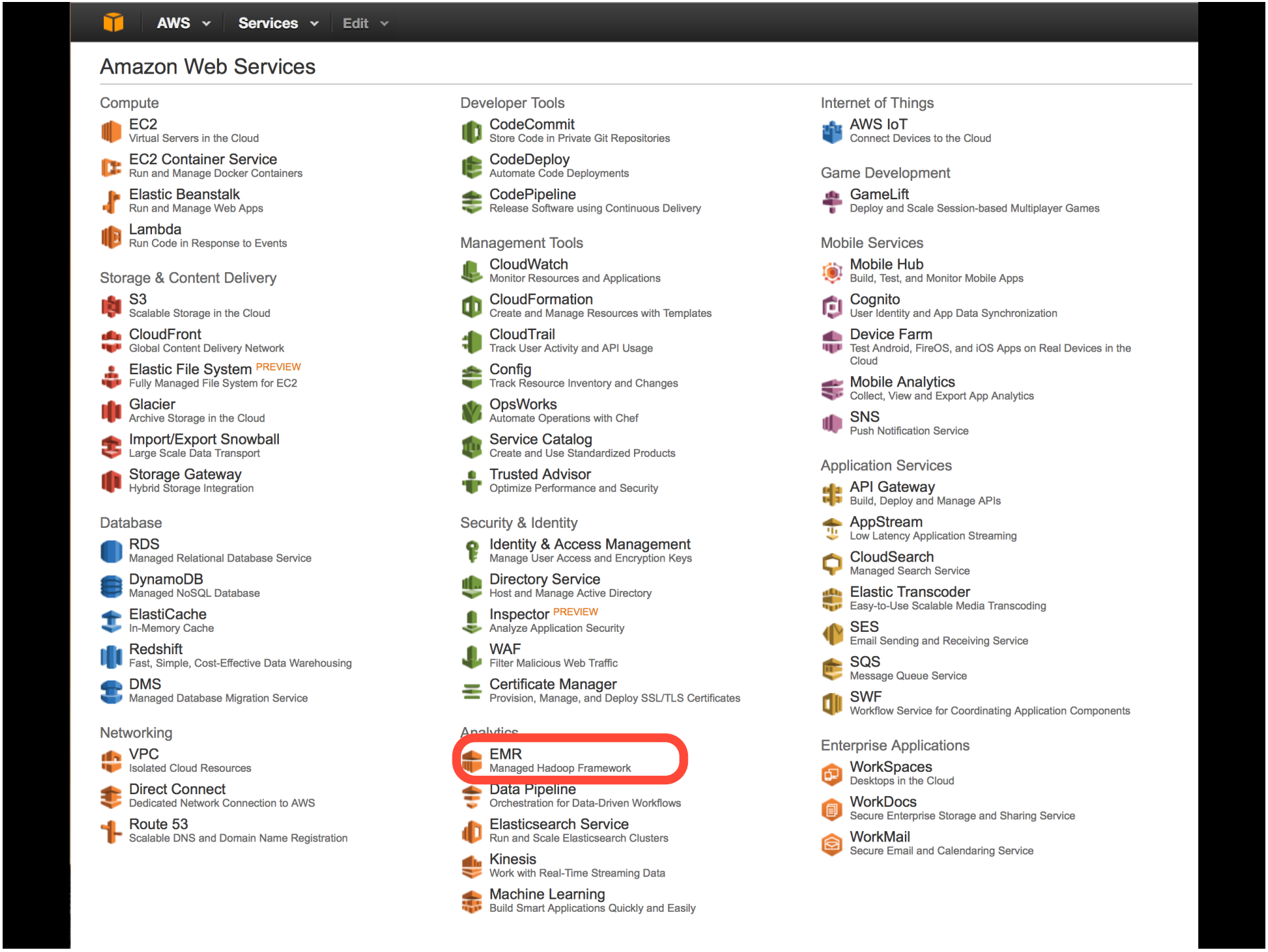
Live Session Outline

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
- **Week**
  - Homework HW7, HW9, HW10
  - AWS: HW10 local
  - Running an MRJob on cloud with the latest version of MRJob, Screen
  - Async lecture recap plus Q&A Spark
  - Install Spark
  - Spark, RDDs, Patterns in Spark
  - DataFrames, Spark SQL
  - Secondary Sorts in Spark
  - MLLib: brief intro
  - Homegrown Kmeans in Spark
- **Wrapup**
  - Finish RECORDING (bonus points for reminding me!)

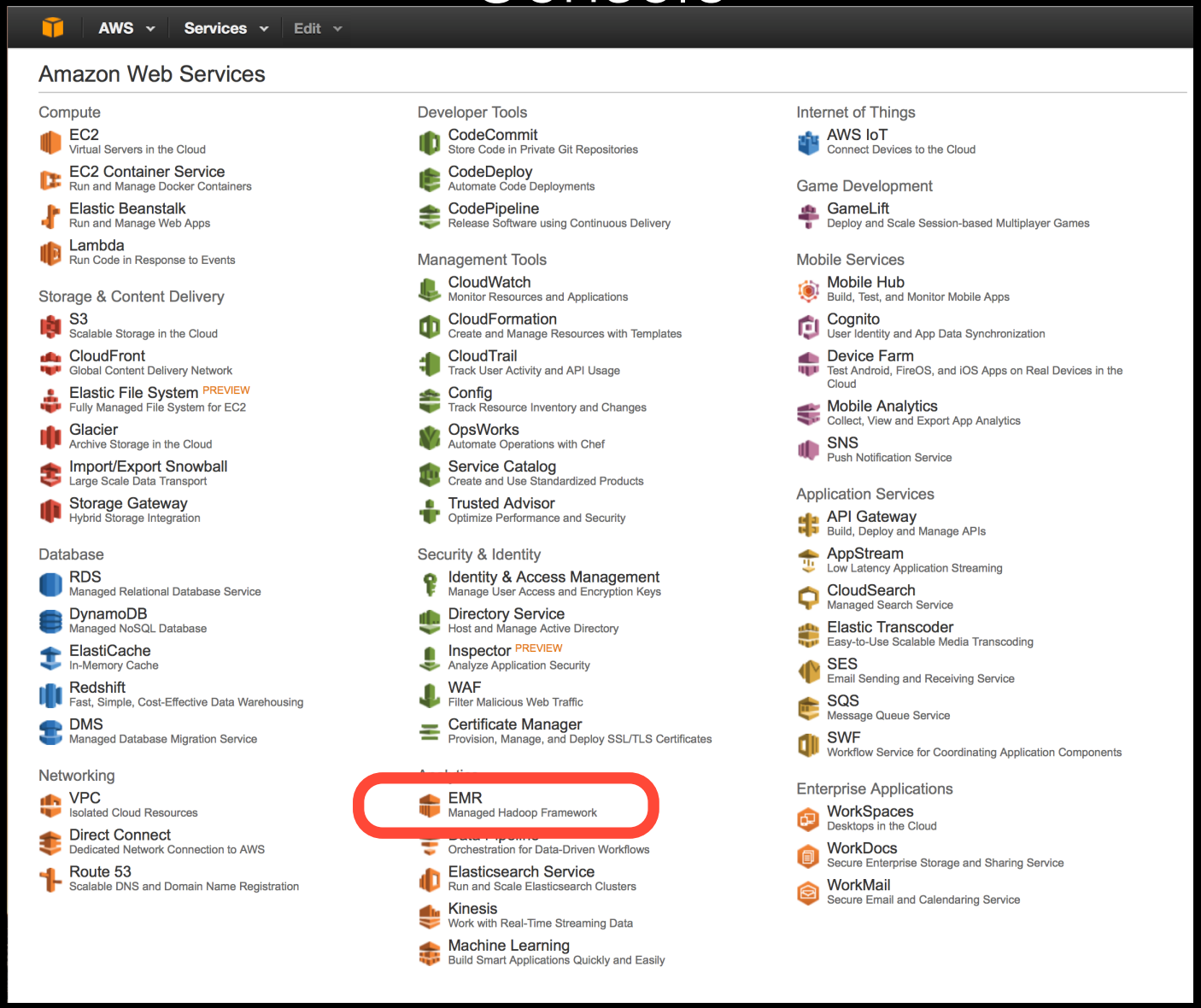
AWS EMR & MRJob

Start a standalone EMR cluster on Amazon
Running an MRJob on the cloud with the latest version of MRJob

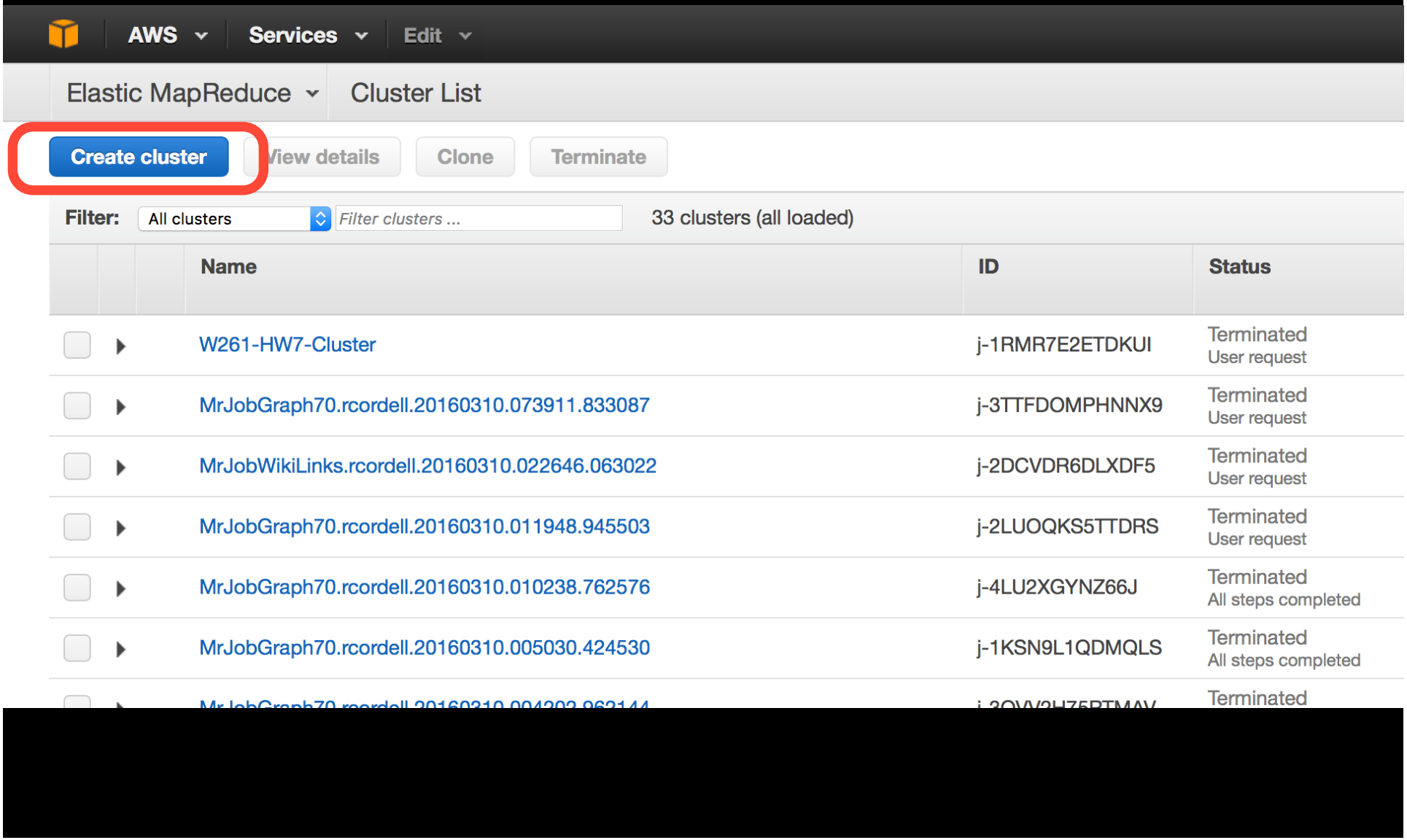
Ron Cordell and Lei Yang, ML@S Spring 2016


AWS
Services
Edit
Amazon Web Services
Compute
EC2
Virtual Servers in the Cloud
EC2 Container Service
Run and Manage Docker Containers
Elastic Beanstalk
Run and Manage Web Apps
Lambda
Run Code in Response to Events
Storage & Content Delivery
S3
Scalable Storage in the Cloud
CloudFront
Global Content Delivery Network
Elastic File System PREVIEW
Fully Managed File System for EC2
Glacier
Archive Storage in the Cloud
Import/Export Snowball
Large Scale Data Transport
Storage Gateway
Hybrid Storage Integration
Database
RDS
Managed Relational Database Service
DynamoDB
Managed NoSQL Database
ElastiCache
In-Memory Cache
Redshift
Fast, Simple, Cost-Effective Data Warehousing
DMS
Managed Database Migration Service
Networking
VPC
Isolated Cloud Resources
Direct Connect
Dedicated Network Connection to AWS
Route 53
Scalable DNS and Domain Name Registration
Developer Tools
CodeCommit
Store Code in Private Git Repositories
CodeDeploy
Automate Code Deployments
CodePipeline
Release Software using Continuous Delivery
Management Tools
CloudWatch
Monitor Resources and Applications
CloudFormation
Create and Manage Resources with Templates
CloudTrail
Track User Activity and API Usage
Config
Track Resource Inventory and Changes
OpsWorks
Automate Operations with Chef
Service Catalog
Create and Use Standardized Products
Trusted Advisor
Optimize Performance and Security
Security & Identity
Identity & Access Management
Manage User Access and Encryption Keys
Directory Service
Host and Manage Active Directory
Inspector PREVIEW
Analyze Application Security
WAF
Filter Malicious Web Traffic
Certificate Manager
Provision, Manage, and Deploy SSL/TLS Certificates
Analytics
EMR
Managed Hadoop Framework
Data Pipeline
Orchestration for Data-Driven Workflows
Elasticsearch Service
Run and Scale Elasticsearch Clusters
Kinesis
Work with Real-Time Streaming Data
Machine Learning
Build Smart Applications Quickly and Easily
Internet of Things
AWS IoT
Connect Devices to the Cloud
Game Development
GameLift
Deploy and Scale Session-based Multiplayer Games
Mobile Services
Mobile Hub
Build, Test, and Monitor Mobile Apps
Cognito
User Identity and App Data Synchronization
Device Farm
Test Android, FireOS, and iOS Apps on Real Devices in the Cloud
Mobile Analytics
Collect, View and Export App Analytics
SNS
Push Notification Service
Application Services
API Gateway
Build, Deploy and Manage APIs
AppStream
Low Latency Application Streaming
CloudSearch
Managed Search Service
Elastic Transcoder
Easy-to-Use Scalable Media Transcoding
SES
Email Sending and Receiving Service
SQS
Message Queue Service
SWF
Workflow Service for Coordinating Application Components
Enterprise Applications
WorkSpaces
Desktops in the Cloud
WorkDocs
Secure Enterprise Storage and Sharing Service
WorkMail
Secure Email and Calendaring Service

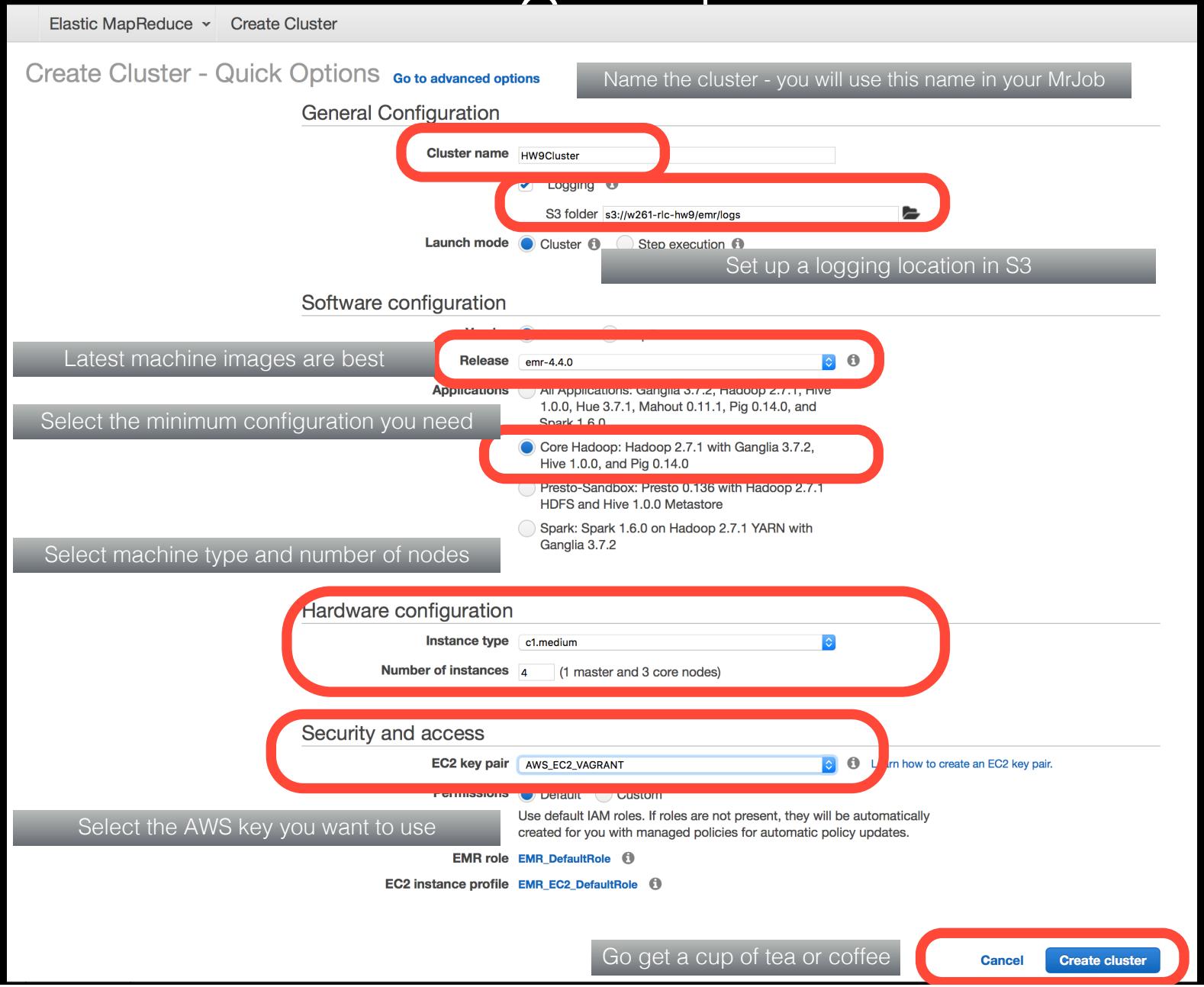
Manually Create EMR Cluster in the AWS Console



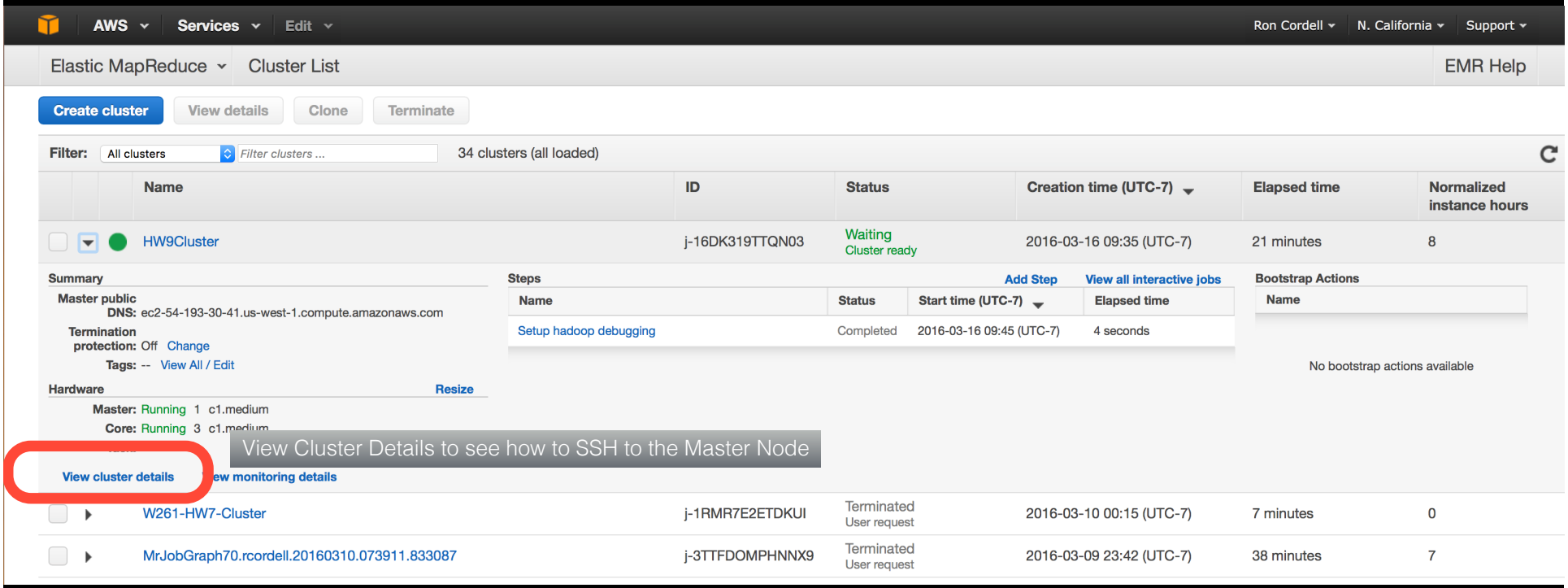
Manually Create EMR Cluster in the AWS Console



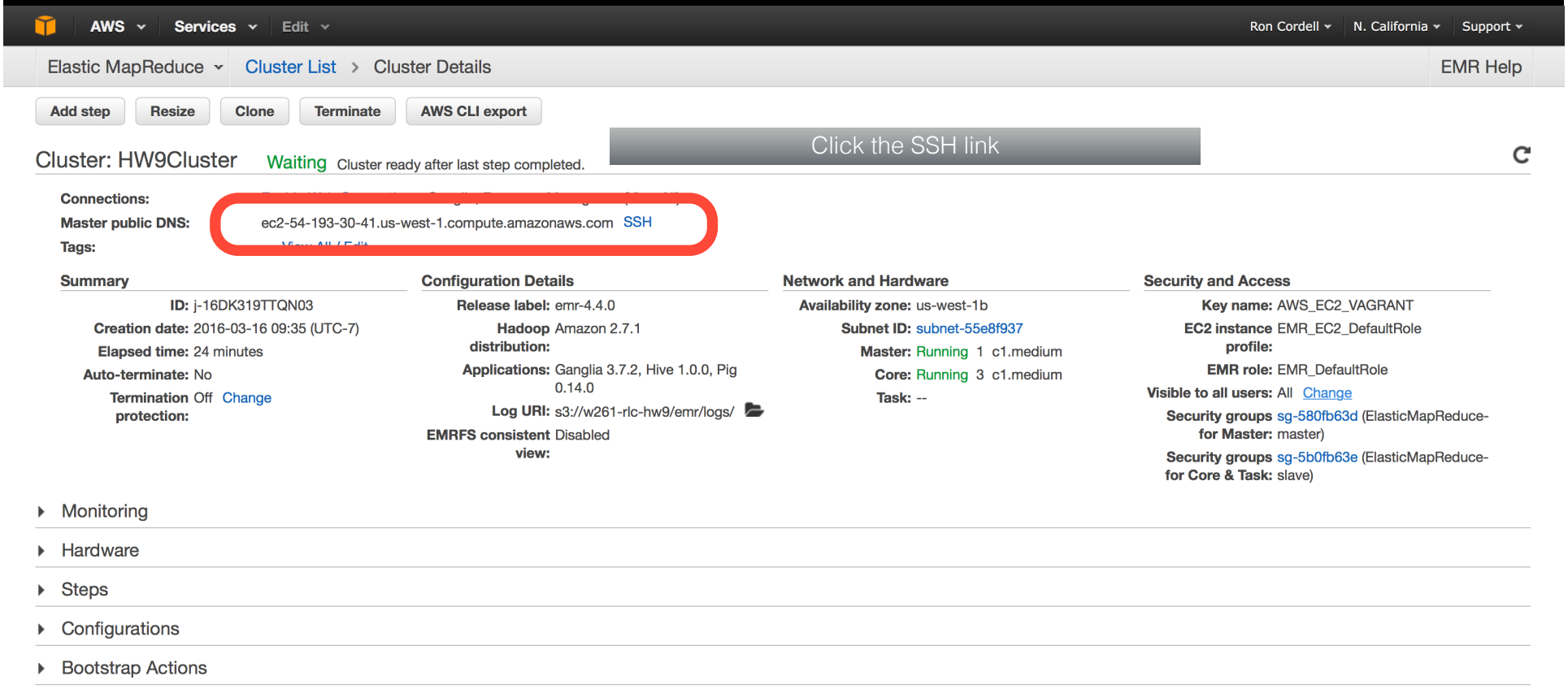
Manually Create EMR Cluster in the AWS



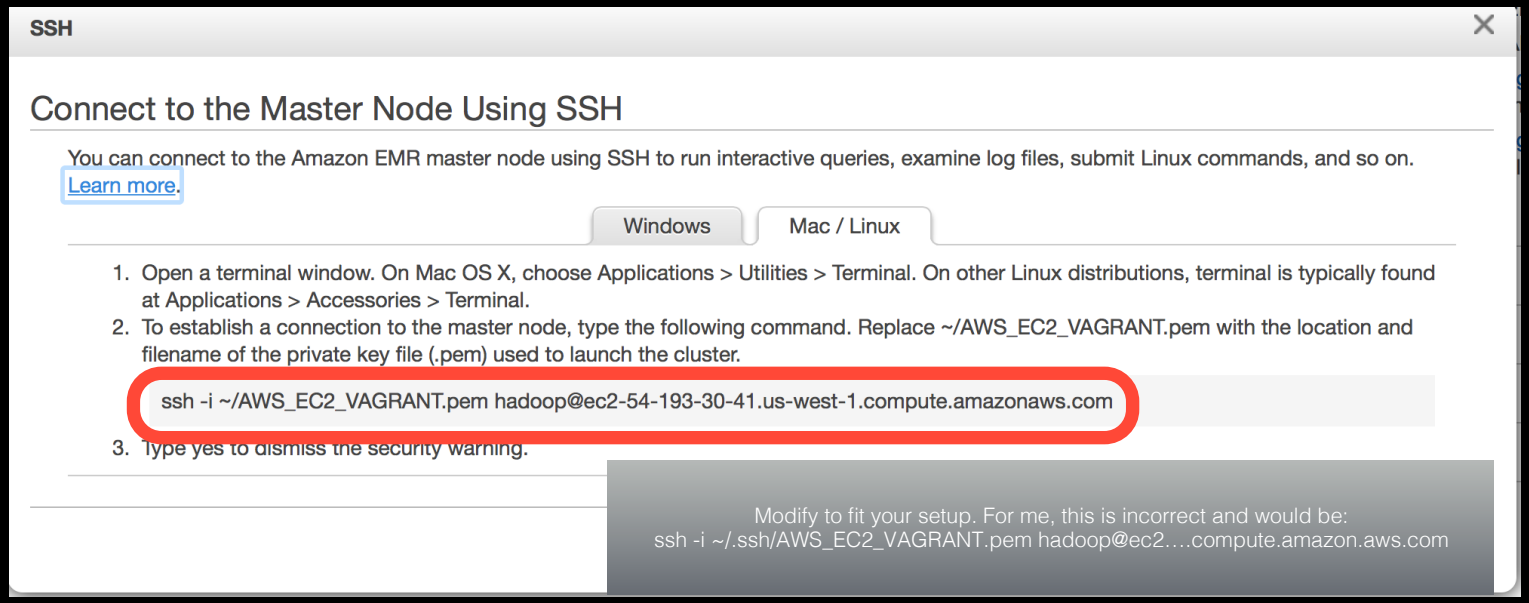
Cluster Ready - Expanded View of the Cluster in the Cluster List

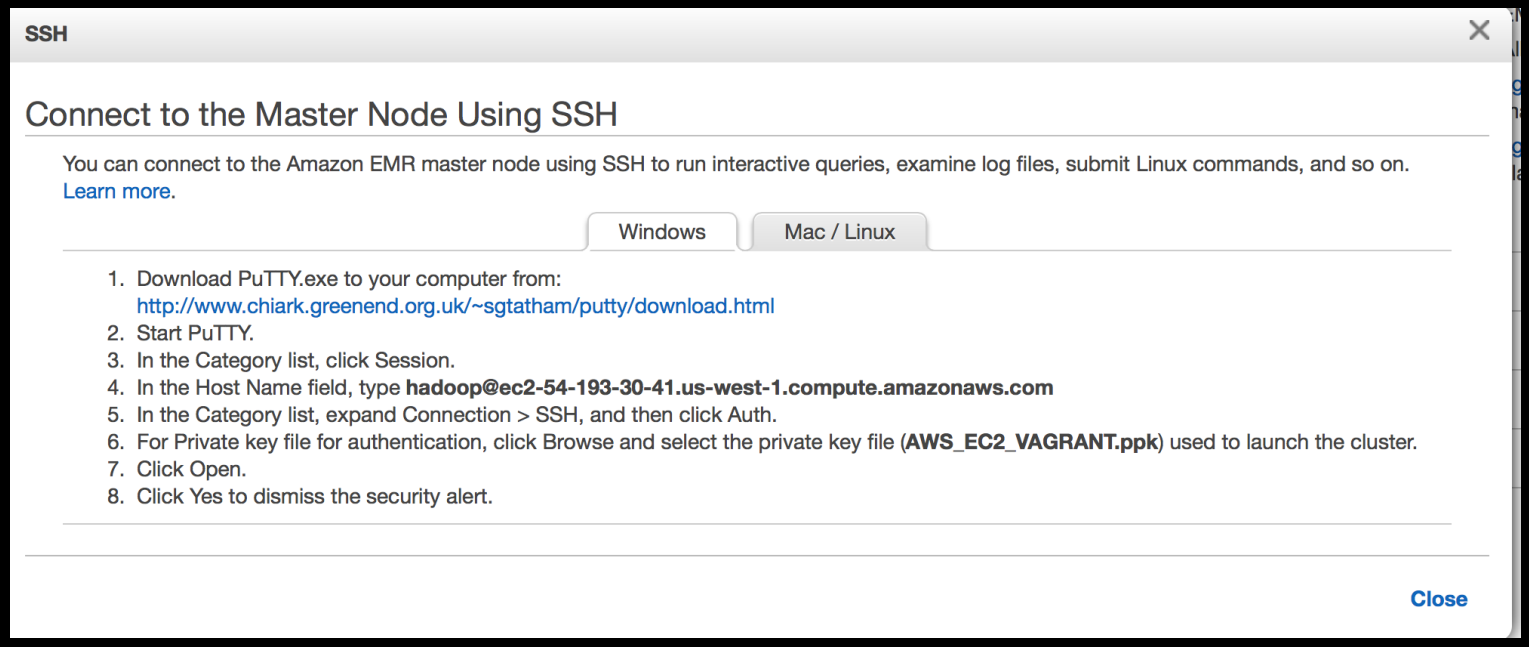


SSH to the Cluster Master Node



SSH instructions - click the SSH link





We're In!

```
(W261env)rcordell@Rons-iMac-Retina:~/Documents/MIDS/W261/week09/HW9$ ssh -i ~/.ssh/AWS_EC2_VAGRANT.pem hadoop@ec2-54-193-30-41.us-west-1.compute.amazonaws.com
The authenticity of host 'ec2-54-193-30-41.us-west-1.compute.amazonaws.com (54.193.30.41)' can't be established.
ECDSA key fingerprint is SHA256:6sEckR8N5WzOx9AlTgT/co2KVcPmKCsqbjqTYPctnVM.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-54-193-30-41.us-west-1.compute.amazonaws.com,54.193.30.41' (ECDSA) to the list of known hosts.
Last login: Wed Mar 16 16:47:37 2016

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2015.09-release-notes/
5 package(s) needed for security, out of 15 available
Run "sudo yum update" to apply all updates.

EEEEEEEEEEEEEEEEEEE MMMMMMM           MMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE::::::EEEEEEEEE:::E M::::::::M       M::::::::M R::::::RRRRRR:::::R
  E:::::E       EEEEE M:::::::::M     M:::::::::M RR:::::R     R:::::R
  E:::::E             M:::::::M::M   M::M:::::::M   R:::R     R:::::R
  E::::::EEEEEEEEEE   M:::::::M M::M M::M M:::::::M   R:::RRRRRR:::::R
  E:::::::::::::::E   M:::::::M  M::M::M  M:::::::M   R:::::::::::RR
  E::::::EEEEEEEEEE   M:::::::M   M:::::M   M:::::::M   R:::RRRRRR:::::R
  E:::::E             M:::::::M    M:::M    M:::::::M   R:::R     R:::::R
  E:::::E       EEEEE M:::::::M     MMM     M:::::::M   R:::R     R:::::R
EE::::::EEEEEEEE:::E M:::::::M             M:::::::M   R:::R     R:::::R
E::::::::::::::::::E M:::::::M             M:::::::M RR:::::R     R:::::R
EEEEEEEEEEEEEEEEEEE MMMMMMM               MMMMMMM RRRRRRR     RRRRRRR

[hadoop@ip-172-31-22-83 ~]$
```

Install Python Packages (MrJob, etc)

```
[hadoop@ip-172-31-22-83 ~]$ sudo pip install mrjob
You are using pip version 6.1.1, however version 8.1.0 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
Collecting mrjob
  Downloading mrjob-0.4.6-py2-none-any.whl (244kB)
    100% |████████████████████████████████| 245kB 655kB/s
Requirement already satisfied (use --upgrade to upgrade): PyYAML>=3.08 in /usr/local/lib64/python2.7/site-packages (from mrjob)
Requirement already satisfied (use --upgrade to upgrade): boto>=2.6.0 in /usr/lib/python2.7/dist-packages (from mrjob)
Requirement already satisfied (use --upgrade to upgrade): simplejson>=2.0.9 in /usr/local/lib64/python2.7/site-packages (from mrjob)
Collecting filechunkio (from mrjob)
  Downloading filechunkio-1.6.tar.gz
Installing collected packages: filechunkio, mrjob
  Running setup.py install for filechunkio
Successfully installed filechunkio-1.6 mrjob-0.4.6
[hadoop@ip-172-31-22-83 ~]$
```

This is with the latest AMI version - it already has Python 2.7.10 and pip; earlier version may not. You can install whatever you need using sudo.

NOTE: you are logged in as the hadoop user so HDFS files will be under hdfs://users/hadoop

Put your code on the Master

SCP

```
scp -i ~/.ssh/your_aws_key.pem project/mrjob.py hadoop@ec2-52-53-232-230.us-wes
```

Copy-Paste

- ssh to the master,
- open an editor,
- copy your code from your local editor
- paste you code in the remote editor

git

- sudo yum install git
- git clone <my repository>
- git pull <my remote repository>

ssh to the master and submit your job using hadoop runner

```
python word_count.py -r hadoop --hadoop-home /usr/lib/hadoop enronemail_1h.txt
no configs found; falling back on auto-configuration
no configs found; falling back on auto-configuration
creating tmp directory /tmp/word_count.hadoop.20160318.050129.032471
writing wrapper script to /tmp/word_count.hadoop.20160318.050129.032471/setup-wrapper.sh
Using Hadoop version 2.7.1
Copying local files into hdfs:///user/hadoop/tmp/mrjob/word_count.hadoop.20160318.050129.032471/files/

PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by default. It's recommended you run your job with --strict-protocols or set up mrjob

HADOOP: packageJobJar: [] [/usr/lib/hadoop/hadoop-streaming-2.7.1-amzn-1.jar] /tmp/streamjob2109238968681413244.jar tmpDir=null
HADOOP: Connecting to ResourceManager at ip-172-31-1-250.us-west-1.compute.internal/172.31.1.250:8032
HADOOP: Connecting to ResourceManager at ip-172-31-1-250.us-west-1.compute.internal/172.31.1.250:8032
HADOOP: MetricsConfigRecord disabledInCluster: false instanceEngineCycleSec: 60 clusterEngineCycleSec: 60 disableClusterEngine: false maxMemoryMb: 307
HADOOP: Created MetricsSaver j-4YUGOAB3MAN3:i-6330f9d6:RunJar:19930 period:60 /mnt/var/em/raw/i-6330f9d6_20160318_RunJar_19930_raw.bin
HADOOP: Loaded native gpl library
HADOOP: Successfully loaded & initialized native-lzo library [hadoop-lzo rev 72c57a1c06c471da40827b432ecff0de6a5c6dcc]
HADOOP: Total input paths to process : 1
HADOOP: number of splits:4
HADOOP: Submitting tokens for job: job_1458275041561_0001
HADOOP: Submitted application application_1458275041561_0001
HADOOP: The url to track the job: http://ip-172-31-1-250.us-west-1.compute.internal:20888/proxy/application_1458275041561_0001/
HADOOP: Running job: job_1458275041561_0001
HADOOP: Job job_1458275041561_0001 running in uber mode : false
HADOOP:  map 0% reduce 0%
HADOOP:  map 25% reduce 0%
HADOOP:  map 83% reduce 0%
HADOOP:  map 100% reduce 0%
HADOOP:  map 100% reduce 33%
HADOOP:  map 100% reduce 67%
HADOOP:  map 100% reduce 100%
HADOOP: Job job_1458275041561_0001 completed successfully
HADOOP: Counters: 51
HADOOP: File System Counters
```

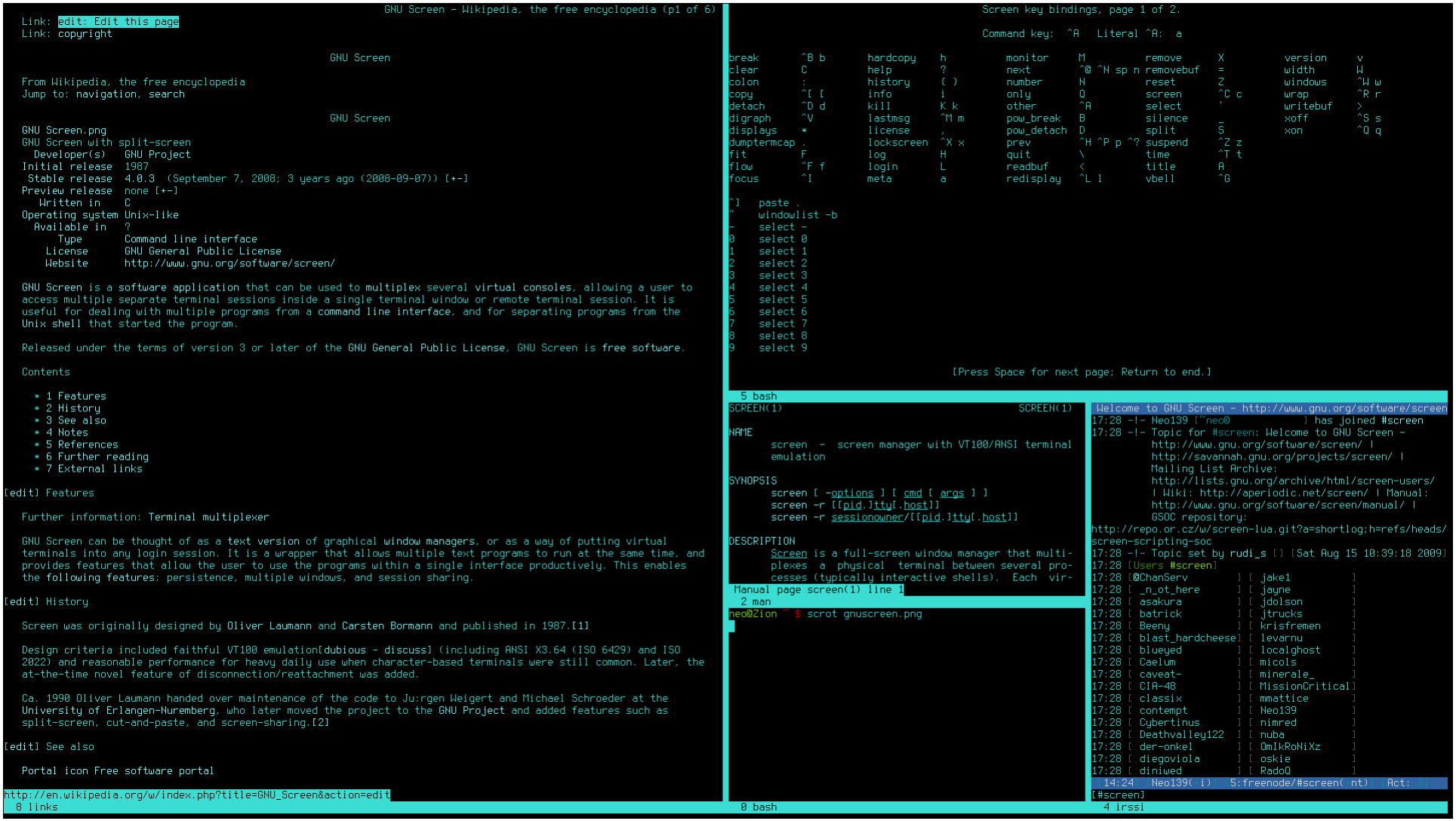
Tmux and screen

- **nohup**
- **screen**
- **tmux**
  - tmux is a software application that can be used to multiplex several virtual consoles, allowing a user to access multiple separate terminal sessions inside a single terminal window or remote terminal session. It is useful for dealing with multiple programs from a command-line interface, and for separating programs from the Unix shell that started the program.[2] It provides much of the same functionality as GNU Screen, but is distributed under a BSD license.

GNU Screen is a terminal multiplexer

- **GNU Screen is a terminal multiplexer, a software application that can be used to multiplex several virtual consoles,**
  - allowing a user to access multiple separate login sessions inside a single terminal window,
  - or detach and reattach sessions from a terminal.
  - It is useful for dealing with multiple programs from a command line interface, and for separating programs from the session of the Unix shell that started the program, **particularly so a remote process continues running even when the user is disconnected**.

screen


GNU Screen - Wikipedia, the free encyclopedia (p1 of 6)
Link: edit: Edit this page
Link: copyright
GNU Screen
From Wikipedia, the free encyclopedia
Jump to: navigation, search
GNU Screen
GNU Screen.png
GNU Screen with split-screen
Developer(s) GNU Project
Initial release 1987
Stable release 4.0.3 (September 7, 2008; 3 years ago (2008-09-07)) [+-]
Preview release none [+-]
Written in C
Operating system Unix-like
Available in ?
Type Command line interface
License GNU General Public License
Website http://www.gnu.org/software/screen/
GNU Screen is a software application that can be used to multiplex several virtual consoles, allowing a user to access multiple separate terminal sessions inside a single terminal window or remote terminal session. It is useful for dealing with multiple programs from a command line interface, and for separating programs from the Unix shell that started the program.
Released under the terms of version 3 or later of the GNU General Public License, GNU Screen is free software.
Contents
* 1 Features
* 2 History
* 3 See also
* 4 Notes
* 5 References
* 6 Further reading
* 7 External links
[edit] Features
Further information: Terminal multiplexer
GNU Screen can be thought of as a text version of graphical window managers, or as a way of putting virtual terminals into any login session. It is a wrapper that allows multiple text programs to run at the same time, and provides features that allow the user to use the programs within a single interface productively. This enables the following features: persistence, multiple windows, and session sharing.
[edit] History
Screen was originally designed by Oliver Laumann and Carsten Bormann and published in 1987.[1]
Design criteria included faithful VT100 emulation[dubious - discuss] (including ANSI X3.64 (ISO 6429) and ISO 2022) and reasonable performance for heavy daily use when character-based terminals were still common. Later, the at-the-time novel feature of disconnection/reattachment was added.
Ca. 1990 Oliver Laumann handed over maintenance of the code to Ju:rgen Weigert and Michael Schroeder at the University of Erlangen-Nuremberg, who later moved the project to the GNU Project and added features such as split-screen, cut-and-paste, and screen-sharing.[2]
[edit] See also
Portal icon Free software portal
http://en.wikipedia.org/w/index.php?title=GNU_Screen&action=edit
8 links
Screen key bindings, page 1 of 2.
Command key: ^A Literal ^A: a
break ^B b
clear C
colon :
copy ^[[
detach ^D d
digraph ^V
displays *
dumptermcap .
fit F
flow ^F f
focus ^I
hardcopy h
help ?
history ( )
info i
kill K k
lastmsg ^M m
license ,
lockscreen ^X x
log H
login L
meta a
monitor M
next ^@ ^N sp n
number N
only Q
other ^A
pow_break B
pow_detach D
prev ^H ^P p ^?
quit \
readbuf <
redisplay ^L l
remove X
removebuf =
reset Z
screen ^C c
select '
silence _
split S
suspend ^Z z
time ^T t
title A
vbell ^G
version v
width W
windows ^W w
wrap ^R r
writebuf >
xoff ^S s
xon ^Q q
^] paste .
" windowlist -b
- select -
0 select 0
1 select 1
2 select 2
3 select 3
4 select 4
5 select 5
6 select 6
7 select 7
8 select 8
9 select 9
[Press Space for next page; Return to end.]
0 bash
SCREEN(1) SCREEN(1)
NAME
screen - screen manager with VT100/ANSI terminal emulation
SYNOPSIS
screen [-options] [cmd [args]]
screen -r [[pid.]tty[.host]]
screen -r sessionowner/[[pid.]tty[.host]]
DESCRIPTION
Screen is a full-screen window manager that multiplexes a physical terminal between several processes (typically interactive shells). Each vir-
Manual page screen(1) line 1
2 man
neo@Zion ~ $ scrot gnuscreen.png
Welcome to GNU Screen - http://www.gnu.org/software/screen
17:28 -!- Neo139 [~neo@] has joined #screen
17:28 -!- Topic for #screen: Welcome to GNU Screen - http://www.gnu.org/software/screen/ | http://savannah.gnu.org/projects/screen/ | Mailing List Archive: http://lists.gnu.org/archive/html/screen-users/ | Wiki: http://aperiodic.net/screen/ | Manual: http://www.gnu.org/software/screen/manual/ | GSOC repository: http://repo.or.cz/w/screen-lua.git?a=shortlog;h=refs/heads/screen-scripting-soc
17:28 -!- Topic set by rudi_s [] [Sat Aug 15 10:39:18 2009]
17:28 [Users #screen]
17:28 [@ChanServ] [jake1]
17:28 [_n_ot_here] [jayne]
17:28 [asakura] [jdolson]
17:28 [batrick] [jtrucks]
17:28 [Beeny] [krisfremen]
17:28 [blast_hardcheese] [levarnu]
17:28 [blueyed] [localghost]
17:28 [Caelum] [micols]
17:28 [caveat-] [minerale_]
17:28 [CIA-48] [MissionCritical]
17:28 [classix] [mmattice]
17:28 [contempt] [Neo139]
17:28 [Cybertinus] [nimred]
17:28 [Deathvalley122] [nuba]
17:28 [der-onkel] [OmIkRoNiXz]
17:28 [diegoviola] [oskie]
17:28 [diniwed] [RadoQ]
14:24 Neo139(i) 5:freenode/#screen(nt) Act:
[#screen]
4 irssi

Screen on wikipedia

GNU Screen is a terminal multiplexer, a software application that can be used to multiplex several virtual consoles, allowing a user to access multiple separate login sessions inside a single terminal window, or detach and reattach sessions from a terminal. It is useful for dealing with multiple programs from a command line interface, and for separating programs from the session of the Unix shell that started the program, particularly so a remote process continues running even when the user is disconnected.

Released under the terms of version 3 or later of the GNU General Public License, GNU Screen is free software.

Contents [hide]

Features [edit]

Further information: Terminal multiplexer

GNU Screen can be thought of as a text version of graphical window managers, or as a way of putting virtual terminals into any login session. It is a wrapper that allows multiple text programs to run at the same time, and provides features that allow the user to use the programs within a single interface productively. This enables the following features: persistence, multiple windows, and session sharing.

Screen is often used when a network connection to the terminal is unreliable, as a dropped network connection typically terminates all programs the user was running (child processes of the login session), due to the session ending and sending a "hangup" signal (SIGHUP) to all the child processes. Running the applications under screen means that the session does not terminate – only the now-defunct terminal gets detached – so applications don't even know the terminal has detached, and allows the user to reattach the session later and continue working from where they left off.

GNU Screen

GNU Screen with split-screen

Developer(s)	Amadeusz Sławiński and the GNU Project
Initial release	1987
Stable release	4.3.1 (June 28, 2015; 7 months ago) [±]
Preview release	None [±]
Development status	Active
Written in	C
Operating system	Unix-like
Type	Terminal multiplexer
License	GNU GPL v3
Website	www.gnu.org/s/screen/

HW9: Group 1

- http://nbviewer.ipython.org/github/maoneto/W261/blob/master/MIDS-W261-2015-HWK-Week09-GroupE-Kasane-Mak-Oneto.ipynb

DATASCI W261: Machine Learning at Scale

Team E: Arthur Mak, Marguerite Oneto, Kasane Utsumi
atm@ischool.berkeley.edu, marguerite.oneto@ischool.berkeley.edu, kasane@ischool.berkeley.edu
W261-1 Fall 2015
Week 9: Homework
November 3, 2015

HW9: Group 2

- http://nbviewer.ipython.org/urls/dl.dropbox.com/s/y96ry8zuwzluyhm/MIDSW2612015HWKWeek09GangwarRouteMafinalv2.0.ipynb

DATASCI W261: Machine Learning at Scale HW9

Vineet Gangwar, James Route, Taylor Ma

vineet.gangwar@gmail.com, james.route@gmail.com, taylorma@berkeley.edu

W261-2: Machine Learning at Scale

Date: Nov 03, 2015

HW9.0

Optional: Pool HW9.3 results

- **Share your results for the top 100 and compare them after 10 and 50 iterations for HW9.3**

- **Use the following GoogleDoc**
  - https://docs.google.com/spreadsheets/d/1zaDjCdWFU5KUmTEJCgYAV1-0qd9HtvmZBWEpnXUMZHc/edit?usp=sharing
  - You should have received an email already that enables you to edit this document
- **Be careful when merging your results**
  - Add a tab for each groups results
  - Then merge your results into the pool sheets for 10 iterations and 50 iterations

2 MR Jobs In the Steps

Thanks James and Team

```
#!/usr/bin/python
from mrjob.job import MRJob
from mrjob.step import MRStep
from mrjob.protocol import RawProtocol
import ast
import sys
import boto
from boto.s3.key import Key

# write the total score for all dangling nodes to an S3 bucket for later retrieval
def write_dangling(p_rank):
    conn = boto.connect_s3("<The credentials are removed for submission>")
    bucket = conn.get_bucket('261-ucb-mids-jroute-output')

    k = Key(bucket)
    mykey = 'dangle'
    k.key = mykey
    k.set_contents_from_string(str(p_rank))
    conn.close()

# MRJob class for running PageRank in EMR
class PageRank9_3(MRJob):
    # user RawProtocol to get rid of quotes in the output
    OUTPUT_PROTOCOL = RawProtocol

    # this is a 2-step job.
    # first step distributes pagerank score and calculates base pagerank per node
    # second step a reducer that handles teleportation and dangling pagerank score
    def steps(self):
        return [MRStep(mapper=self.mapper, reducer=self.reducer),
                MRStep(reducer_init=self.agg_init, reducer=self.aggregate)]

    # set command line options to get number of nodes in graph and teleport/alpha value
    def configure_options(self):
        super(PageRank9_3, self).configure_options()
        self.add_passthrough_option(
            '--num-nodes',
            type=str,
            help='Number of nodes in graph.'
        )
        self.add_passthrough_option(
            '--alpha',
            type=str,
            help='Value for PageRank algorithm.'
```

Live Session Outline

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
- **Week**
  - Homework HW7, HW9, HW10
  - AWS: HW10 local
  - Running an MRJob on cloud with the latest version of MRJob, Screen
  - Async lecture recap plus Q&A Spark
  - Install Spark
  - Spark, RDDs, Patterns in Spark
  - DataFrames, Spark SQL
  - Secondary Sorts in Spark
  - MLLib: brief intro
  - Homegrown Kmeans in Spark
- **Wrapup**
  - Finish RECORDING (bonus points for reminding me!)

Unit 10 | Spark: From Basics to Advanced

Hide Contents

- 10.1 Weekly Introduction (2 mins)
- 10.2 Assigned Readings
- 10.3 Functional Programming Review (8 mins)
- 10.4 Background on Spark (15 mins)
- 10.5 Spark Basics (20 mins)
- 10.6 Programming with Base RDDs (22 mins)
  - 10.6.1 Quiz: Spark Basics
- 10.7 Animated Example Data Flow (8 mins)
- 10.8 Pair RDDs (13 mins)
- 10.9 Basic WordCount in Spark (12 mins)
- 10.10 Spark Summary (6 mins)
- 10.8.1 Quiz: Pair RDDs

Question

- **What is an RDD?**
  - Base RDD
  - Pair RDD

- **Key-Value pairs**
- **Lazy evaluation**
- **Persistence**
- **Lineage**

Live Session Outline

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
- **Week**
  - Homework HW7, HW9, HW10
  - AWS: HW10 local
  - Running an MRJob on cloud with the latest version of MRJob, Screen
  - Async lecture recap plus Q&A Spark
  - Install Spark
  - Spark, RDDs, Patterns in Spark
  - DataFrames, Spark SQL
  - Secondary Sorts in Spark
  - MLLib: brief intro
  - Homegrown Kmeans in Spark
- **Wrapup**
  - Finish RECORDING (bonus points for reminding me!)

Install the following:

- Oracle Java JDK
- Spark
- Anaconda Python (+Jupyter notebooks)

Installing Hadoop

- **Windows:**
  - Hadoop 1.0
  - http://saphanatutorial.com/hadoop-installation-on-windows-7-using-cygwin/
  - Hadoop 2.0 (Hortonworks Data Platform 2.0 for Windows)
  - http://hortonworks.com/blog/install-hadoop-windows-hortonworks-data-platform-2-0/
- **Mac:**
  - http://amodernstory.com/2014/09/23/installing-hadoop-on-mac-osx-yosemite/
  - This link is for haddoop 2.6. I follow the instructions and easily get hadoop installed.
- **Linux (Ubuntu):**
  - http://www.bogotobogo.com/Hadoop/BigData_hadoop_Install_on_ubuntu_single_node_cluster.php

On Mac install HomeBrew

- **Install HomeBrew**
  - Download it from the website at http://brew.sh/ or simply paste the script inside the terminal
  - $ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"

On Windows install CygWin

- **Cygwin is:**
  - a large collection of GNU and Open Source tools which provide functionality similar to a Linux distribution on Windows.

Current Cygwin DLL version

The most recent version of the Cygwin DLL is 2.2.1. Install it by running setup-x86.exe (32-bit installation) or setup-x86_64.exe (64-bit installation).

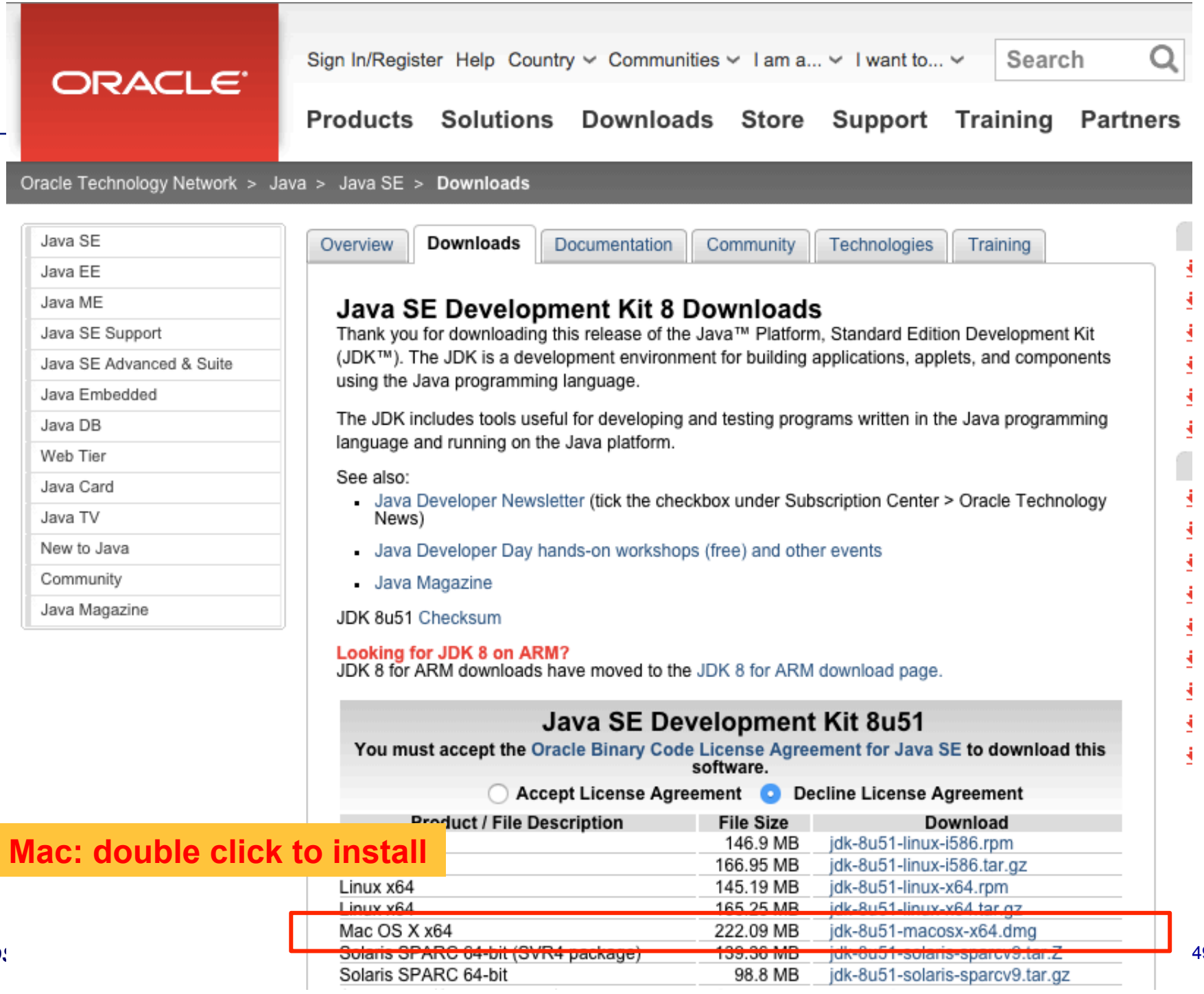
Use the setup program to perform a fresh install or to update an existing installation.

Note that individual packages in the distribution are updated separately from the DLL so the Cygwin DLL version is not useful as a general Cygwin release number.

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**



On Mac: double click to install

Anaconda Install

Download python+Notebook

Contents

- http://docs.continuum.io/anaconda/install

Download the Anaconda installer and double click it.

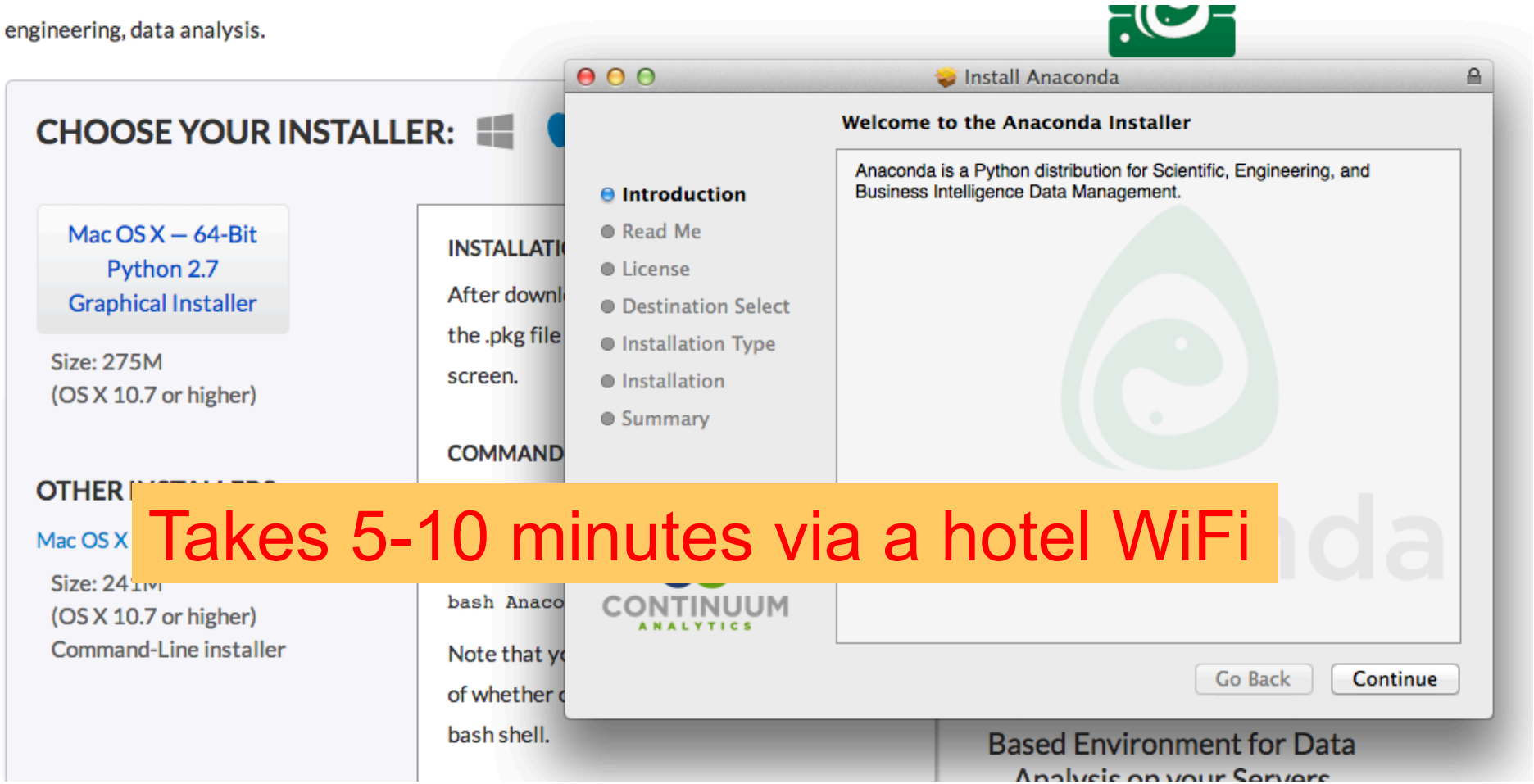
OS X Install

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

Takes 5-10 minutes via a hotel WiFi

Anaconda Installer



conda update --prefix /Users/jshanahan/anaconda anaconda

conda update --prefix /Users/jshanahan/anaconda anaconda

```
boto-2.38.0-py 100% |################################################################################| Time: 0:00:00  90.07 kB/s
conda-env-2.4. 100% |################################################################################| Time: 0:00:00  90.07 kB/s
cython-0.22.1- 100% |################################################################################| Time: 0:00:06 387.59 kB/s
cytoolz-0.7.3- 100% |################################################################################| Time: 0:00:01 202.66 kB/s
decorator-3.4. 100% |################################################################################| Time: 0:00:00   1.17 MB/s
greenlet-0.4.7 100% |################################################################################| Time: 0:00:00  55.86 kB/s
idna-2.0-py27_ 100% |################################################################################| Time: 0:00:00 145.75 kB/s
ipaddress-1.0. 100% |################################################################################| Time: 0:00:00  91.65 kB/s
llvmlite-0.5.0 100% |################################################################################| Time: 0:00:14 435.04 kB/s
lxml-3.4.4-py2 100% |################################################################################| Time: 0:00:04 220.90 kB/s
mistune-0.5.1- 100% |################################################################################| Time: 0:00:01 167.80 kB/s
nose-1.3.7-py2 100% |################################################################################| Time: 0:00:01 175.91 kB/s
astropy-1.0.3- 100% |################################################################################| Time: 0:00:14 369.05 kB/s
bcolz-0.9.0-np 100% |################################################################################| Time: 0:00:01 245.33 kB/s
bottleneck-1.0 100% |################################################################################| Time: 0:00:01 167.26 kB/s
numba-0.19.1-n 100% |################################################################################| Time: 0:00:04 230.86 kB/s
numexpr-2.4.3- 100% |################################################################################| Time: 0:00:00 128.90 kB/s
blz-0.6.2-np19 100% |################################################################################| Time: 0:00:01 221.19 kB/s
pillow-2.8.2-p 100% |################################################################################| Time: 0:00:01 269.09 kB/s
ply-3.6-py27_0 100% |################################################################################| Time: 0:00:00 124.84 kB/s
py-1.4.27-py27 100% |################################################################################| Time: 0:00:00 141.94 kB/s
pycparser-2.14 100% |################################################################################| Time: 0:00:01 145.17 kB/s
cffi-1.1.0-py2 100% |################################################################################| Time: 0:00:01 158.43 kB/s
pycurl-7.19.5. 100% |################################################################################| Time: 0:00:00  83.32 kB/s
pyflakes-0.9.2 100% |################################################################################| Time: 0:00:00  99.50 kB/s
pytest-2.7.1-p 100% |################################################################################| Time: 0:00:01 165.68 kB/s
python-2.7.10- 100% |################################################################################| Time: 0:00:28 412.67 kB/s
python.app-1.2 100% |################################################################################| Time: 0:00:00   3.96 MB/s
pytz-2015.4-py 100% |################################################################################| Time: 0:00:01 163.19 kB/s
pyyaml-3.11-py 100% |################################################################################| Time: 0:00:01 141.79 kB/s
```

To start the Notebook

/Users/jshanahan/anaconda/bin/ipython notebook&

- **/Users/jshanahan/anaconda/bin/ipython notebook&**

Apache Spark Download

- **Install 1.6.1 (or latest version) by clicking here**
  - https://dl.dropboxusercontent.com/u/27377155/spark-1.6.1-bin-hadoop2.6.tgz
- **For other versions click here**
  - http://spark.apache.org/downloads.html
  - http://spark.apache.org/docs/latest/programming-guide.html

Download latest version of Spark

- ..

Lightning-fast cluster computing

Download | Libraries | Documentation | Examples | Community | FAQ

Get latest version Spark 1.5.1 as of October 2, 2015

Download Spark

The latest release of Spark is Spark 1.5.1, released on October 2, 2015 (release notes) (git tag)

Step1 — 1. Choose a Spark release: 1.5.1 (Oct 02 2015)

Step2 — 2. Choose a package type: Pre-built for Hadoop 2.6 and later

3. Choose a download type: Select Apache Mirror

Step4 — 4. Download Spark: spark-1.5.1-bin-hadoop2.6.tgz

5. Verify this release using the 1.5.1 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

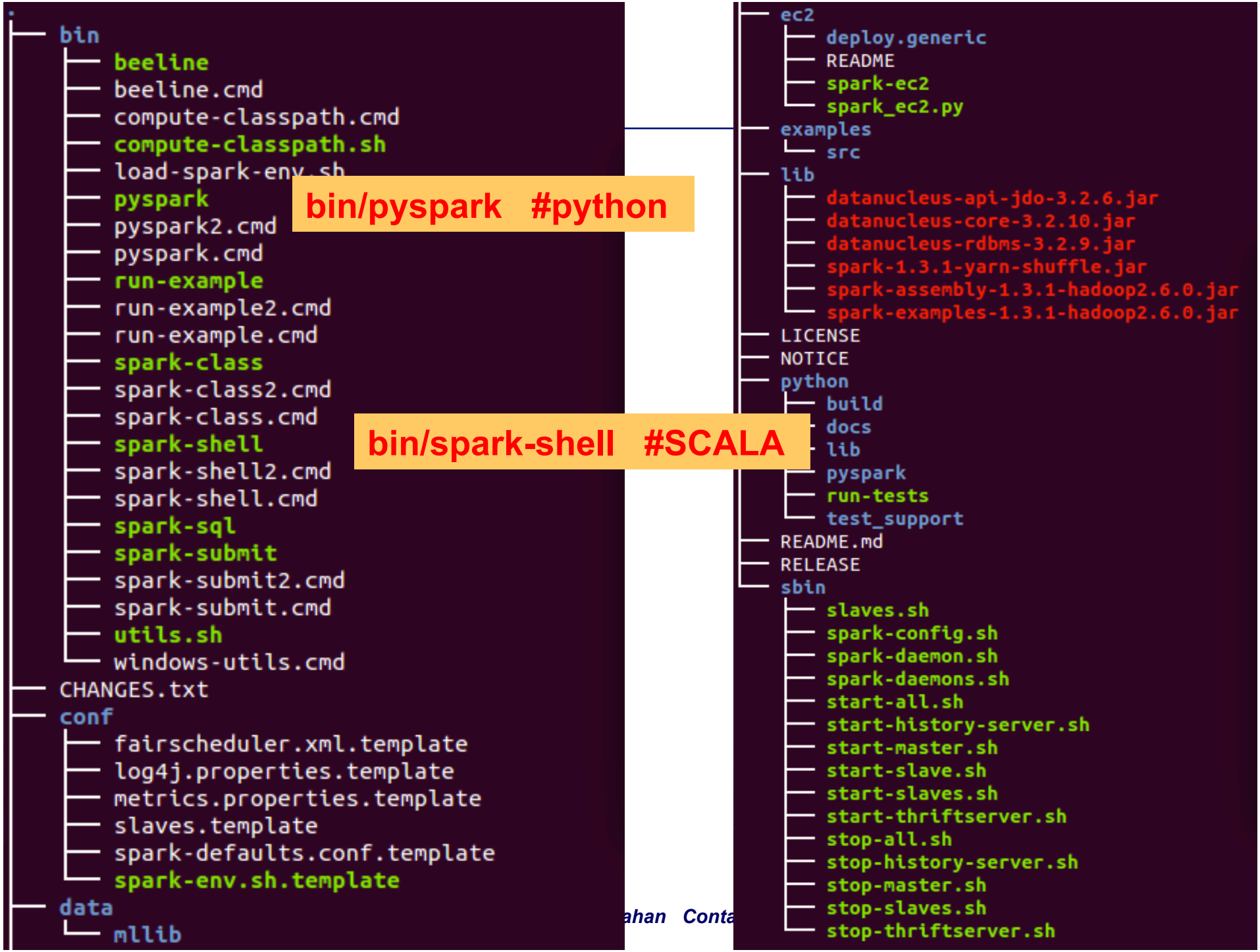
```
tar -xzvf spark-1.4.1-bin-hadoop2.6.tgz
cd spark-1.4.1-bin-hadoop2.6
./bin/pyspark
 #if everything goes well, you can see the PySpark interactive shell
./bin/pyspark
```

```
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             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
```

Untar the Spark; start pyspark interpreter

- tar -xzvf 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; u
w 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
]
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... us
ava classes where applicable
15/02/11 16:57:03 INFO HttpFileServer: HTTP File server directory is /var/folders/j4/95k348x940xcz40f
n/T/spark-d6caef39-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@10.0.0.93:5
rtbeatReceiver
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 I
ManagerId(<driver>, localhost, 54081)
15/02/11 16:57:03 INFO BlockManagerMaster: Registered BlockManager
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.2.1
      /_/

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
pyspark2.cmd
pyspark.cmd
run-example
run-example2.cmd
run-example.cmd
spark-class
spark-class2.cmd
spark-class.cmd
spark-shell
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
bin/pyspark #python
bin/spark-shell #SCALA
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

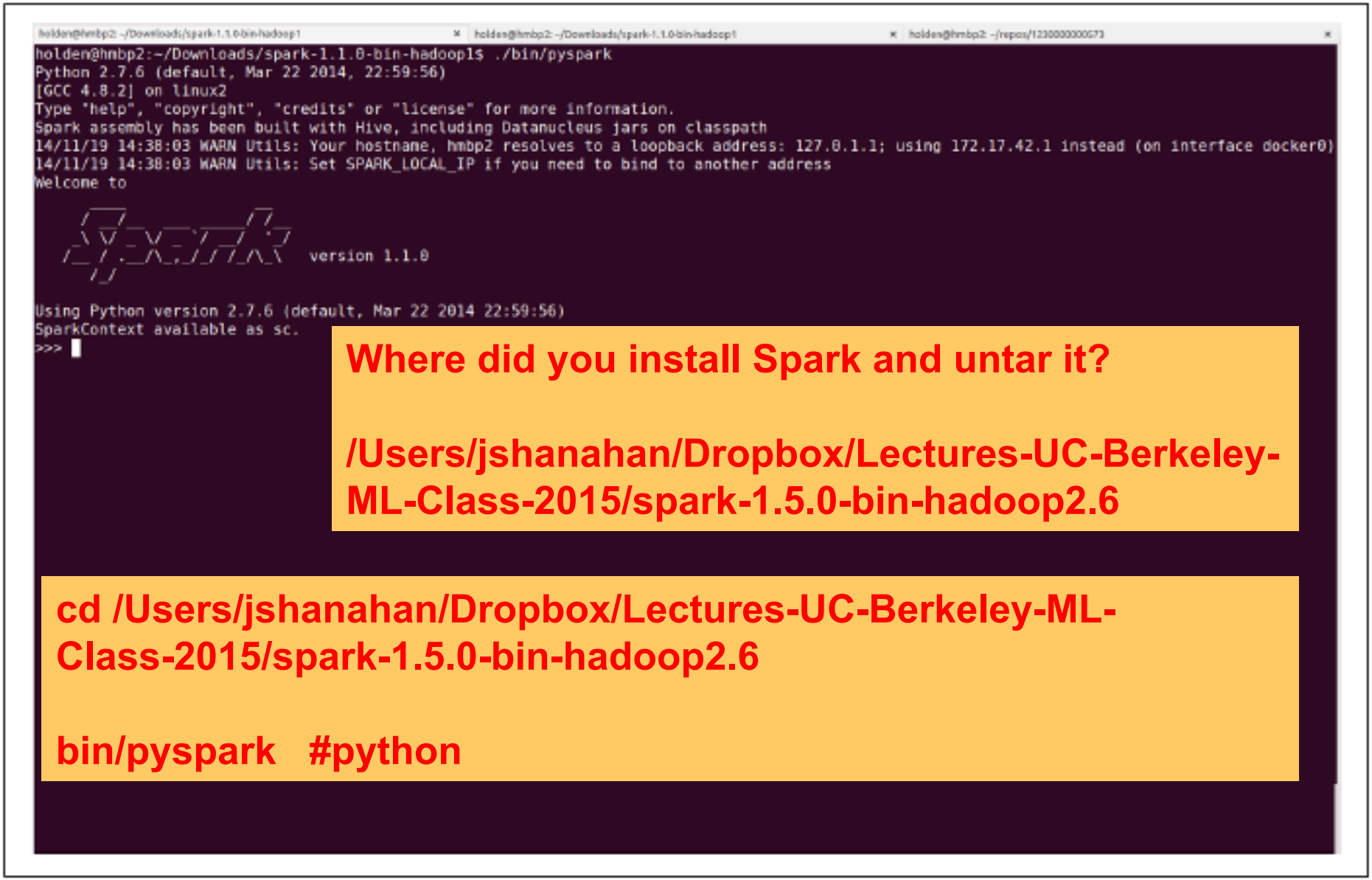


Figure 2-2. The PySpark shell with less logging output

bin/sparkR

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

Lau                                                          6/b
in/
log
log
log
Usi
15/
15/                                                           wh
ere
15/
15/
15/                                                          ns:
 Se
```

Where did you install Spark and untar it?

/Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/spark-1.5.0-bin-hadoop2.6

#To run sparkR
/Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/spark-1.5.0-bin-hadoop2.6/bin/sparkR

Commands in R: iris data

- ?iris
- ?data

iris package:datasets R Documentation

Edgar Anderson's Iris Data

Description:

This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are _Iris setosa_, _versicolor_, and _virginica_.

Usage:

```
iris
iris3
```

Format:

'iris' is a data frame with 150 cases (rows) and 5 variables (columns) named 'Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width', and 'Species'.

'iris3' gives the same data arranged as a 3-dimensional array of size 50 by 4 by 3, as represented by S-PLUS. The first dimension gives the case number within the species subsample, the second the measurements with names 'Sepal L.', 'Sepal W.', 'Petal L.', and 'Petal W.', and the third the species.

```
> head(iris)
  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
>
```

```
Data sets in package 'datasets':

AirPassengers           Monthly Airline Passenger Numbers 1949-1960
BJsales                 Sales Data with Leading Indicator
BJsales.lead (BJsales)
                        Sales Data with Leading Indicator
BOD                     Biochemical Oxygen Demand
CO2                     Carbon Dioxide Uptake in Grass Plants
ChickWeight             Weight versus age of chicks on different diets
DNase                   Elisa assay of DNase
EuStockMarkets          Daily Closing Prices of Major European Stock
                        Indices, 1991-1998
Formaldehyde            Determination of Formaldehyde
HairEyeColor            Hair and Eye Color of Statistics Students
Harman23.cor            Harman Example 2.3
Harman74.cor            Harman Example 7.4
Indometh                Pharmacokinetics of Indomethacin
InsectSprays            Effectiveness of Insect Sprays
JohnsonJohnson          Quarterly Earnings per Johnson & Johnson Share
LakeHuron               Level of Lake Huron 1875-1972
LifeCycleSavings        Intercountry Life-Cycle Savings Data
Loblolly                Growth of Loblolly pine trees
Nile                    Flow of the River Nile
Orange                  Growth of Orange Trees
OrchardSprays           Potency of Orchard Sprays
PlantGrowth             Results from an Experiment on Plant Growth
Puromycin               Reaction Velocity of an Enzymatic Reaction
Seatbelts               Road Casualties in Great Britain 1969-84
Theoph                  Pharmacokinetics of Theophylline
Titanic                 Survival of passengers on the Titanic
ToothGrowth             The Effect of Vitamin C on Tooth Growth in
                        Guinea Pigs
UCBAdmissions           Student Admissions at UC Berkeley
UKDriverDeaths          Road Casualties in Great Britain 1969-84
UKgas                   UK Quarterly Gas Consumption
USAccDeaths             Accidental Deaths in the US 1973-1978
USArrests               Violent Crime Rates by US State
USJudgeRatings          Lawyers' Ratings of State Judges in the US
                        Superior Court
USPersonalExpenditure   Personal Expenditure Data
VADeaths                Death Rates in Virginia (1940)
WWWusage                Internet Usage per Minute
WorldPhones             The World's Telephones
ability.cov             Ability and Intelligence Tests
airmiles                Passenger Miles on Commercial US Airlines,
                        1937-1960
:
```

>data() #lists all datasets

```
airquality              New York Air Quality Measurements
anscombe                Anscombe's Quartet of 'Identical' S
                        Regressions
attenu                  The Joyner-Boore Attenuation Data
attitude                The Chatterjee-Price Attitude Data
austres                 Quarterly Time Series of the Number
                        Australian Residents
beaver1 (beavers)       Body Temperature Series of Two Beav
beaver2 (beavers)       Body Temperature Series of Two Beav
cars                    Speed and Stopping Distances of Car
chickwts                Chicken Weights by Feed Type
co2                     Mauna Loa Atmospheric CO2 Concentra
crimtab                 Student's 3000 Criminals Data
discoveries             Yearly Numbers of Important Discove
esoph                   Smoking, Alcohol and (O)esophageal
euro                    Conversion Rates of Euro Currencies
euro.cross (euro)       Conversion Rates of Euro Currencies
eurodist                Distances Between European Cities
faithful                Old Faithful Geyser Data
fdeaths (UKLungDeaths)
                        Monthly Deaths from Lung Diseases i
freeny                  Freeny's Revenue Data
freeny.x (freeny)       Freeny's Revenue Data
freeny.y (freeny)       Freeny's Revenue Data
infert                  Infertility after Spontaneous and I
                        Abortion
iris                    Edgar Anderson's Iris Data
iris3                   Edgar Anderson's Iris Data
islands                 Areas of the World's Major Landmass
ldeaths (UKLungDeaths)
                        Monthly Deaths from Lung Diseases i
lh                      Luteinizing Hormone in Blood Sample
longley                 Longley's Economic Regression Data
lynx                    Annual Canadian Lynx trappings 1821
mdeaths (UKLungDeaths)
                        Monthly Deaths from Lung Diseases i
morley                  Michelson Speed of Light Data
mtcars                  Motor Trend Car Road Tests
nhtemp                  Average Yearly Temperatures in New
nottem                  Average Monthly Temperatures at Not
                        1920-1939
npk                     Classical N, P, K Factorial Experim
occupationalStatus      Occupational Status of Fathers and
precip                  Annual Precipitation in US Cities
presidents              Quarterly Approval Ratings of US Pr
pressure                Vapor Pressure of Mercury as a Func
:
```

```
morley                  Michelson Speed of Light Data
mtcars                  Motor Trend Car Road Tests
nhtemp                  Average Yearly Temperatures in New Haven
nottem                  Average Monthly Temperatures at Nottingham,
                        1920-1939
npk                     Classical N, P, K Factorial Experiment
occupationalStatus      Occupational Status of Fathers and their Sons
precip                  Annual Precipitation in US Cities
presidents              Quarterly Approval Ratings of US Presidents
pressure                Vapor Pressure of Mercury as a Function of
                        Temperature
quakes                  Locations of Earthquakes off Fiji
randu                   Random Numbers from Congruential Generator
                        RANDU
rivers                  Lengths of Major North American Rivers
rock                    Measurements on Petroleum Rock Samples
sleep                   Student's Sleep Data
stack.loss (stackloss)
                        Brownlee's Stack Loss Plant Data
stack.x (stackloss)     Brownlee's Stack Loss Plant Data
stackloss               Brownlee's Stack Loss Plant Data
state.abb (state)       US State Facts and Figures
state.area (state)      US State Facts and Figures
state.center (state)    US State Facts and Figures
state.division (state)
                        US State Facts and Figures
state.name (state)      US State Facts and Figures
state.region (state)    US State Facts and Figures
state.x77 (state)       US State Facts and Figures
sunspot.month           Monthly Sunspot Data, from 1749 to "Present"
sunspot.year            Yearly Sunspot Data, 1700-1988
sunspots                Monthly Sunspot Numbers, 1749-1983
swiss                   Swiss Fertility and Socioeconomic Indicators
                        (1888) Data
treering                Yearly Treering Data, -6000-1979
trees                   Girth, Height and Volume for Black Cherry Trees
uspop                   Populations Recorded by the US Census
volcano                 Topographic Information on Auckland's Maunga
                        Whau Volcano
warpbreaks              The Number of Breaks in Yarn during Weaving
women                   Average Heights and Weights for American Women

Use 'data(package = .packages(all.available = TRUE))'
to list the data sets in all *available* packages.

(END)
```

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
```

Wordcount Notebook

- **Download the following notebook**
  - https://www.dropbox.com/s/ul0l3q98w54dr8x/WordCount.ipynb?dl=0
- **Cd to the directory that contains the wordcount notebook**
  - cd /Users/jshanahan/Dropbox/NativeX-Internal/Publications/WSDM-2016
- **Launch Notebook**
  - /Users/jshanahan/anaconda/bin/ipython notebook&

localhost:8888/notebooks/Notebooks/WordCount/WordCount.ipynb#

jupyter WordCount (autosaved)

File Edit View Insert Cell Kernel Help Python 2

Code Cell Toolbar: None

In [2]:
```
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-1.5.0-bin-hadoop2.6'


if not spark_home:
    raise ValueError('SPARK_HOME enviroment 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 1.5.0
      /_/

Using Python version 2.7.11 (default, Dec  6 2015 18:57:58)
SparkContext available as sc, HiveContext available as sqlContext.
```

In [3]:
```
%%writefile wordcount.txt
hello hi hi hallo
bonjour hola hi ciao
nihao konnichiwa ola
hola nihao hello
```

```
Writing wordcount.txt
```

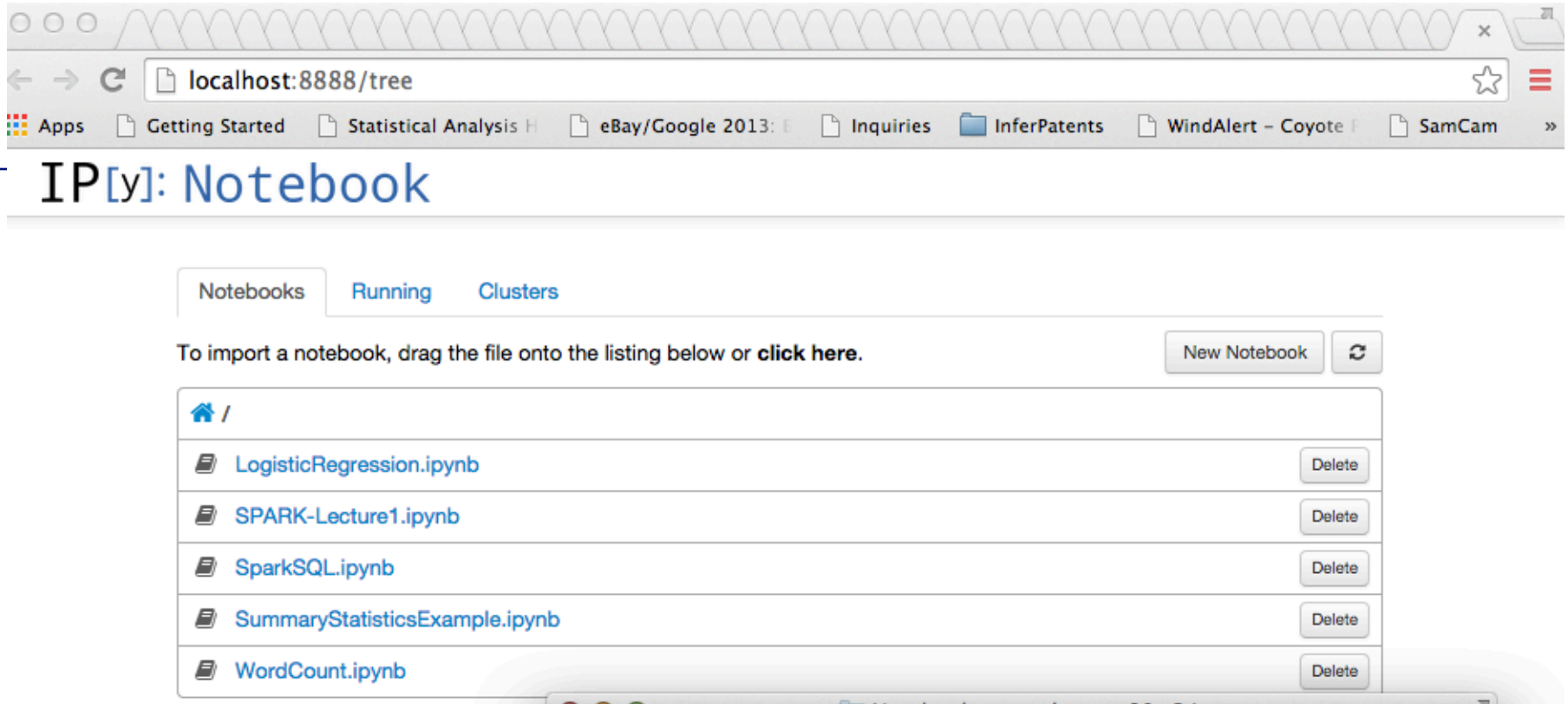
In [4]:
```
cat wordcount.txt
```

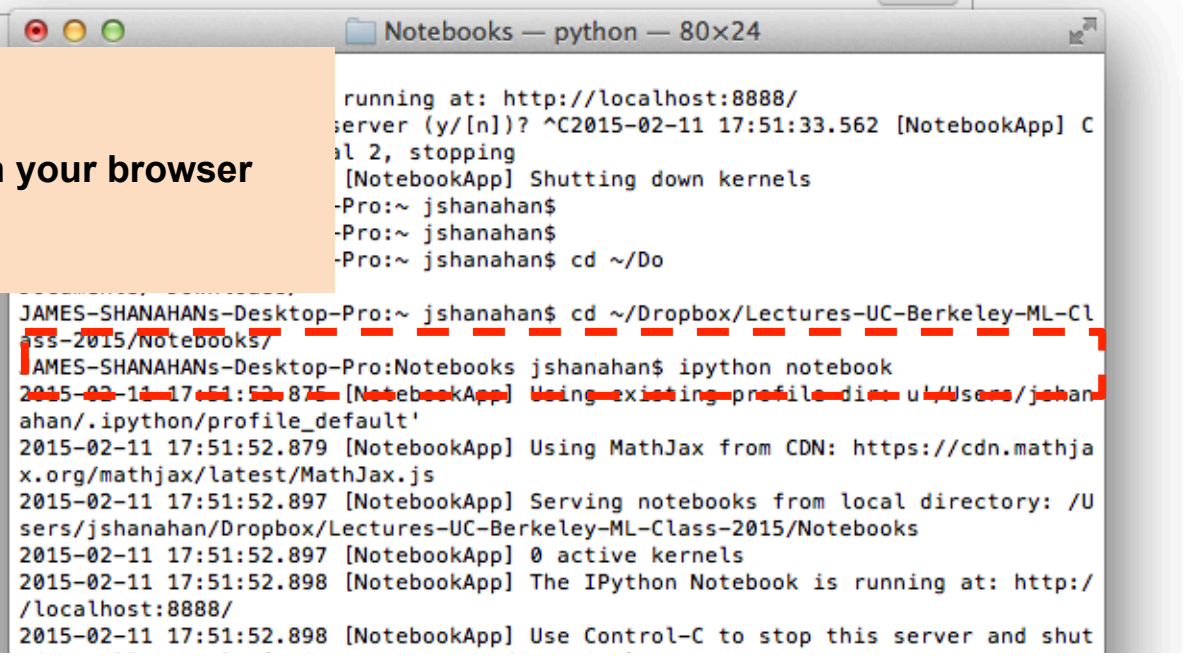
```
hello hi hi hallo
bonjour hola hi ciao
nihao konnichiwa ola
hola nihao hello
```

In [ ]:

In [3]:
```
#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)
```





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

Fire up ipython Notebook in your Browser

#start the python server and notebook in your browser

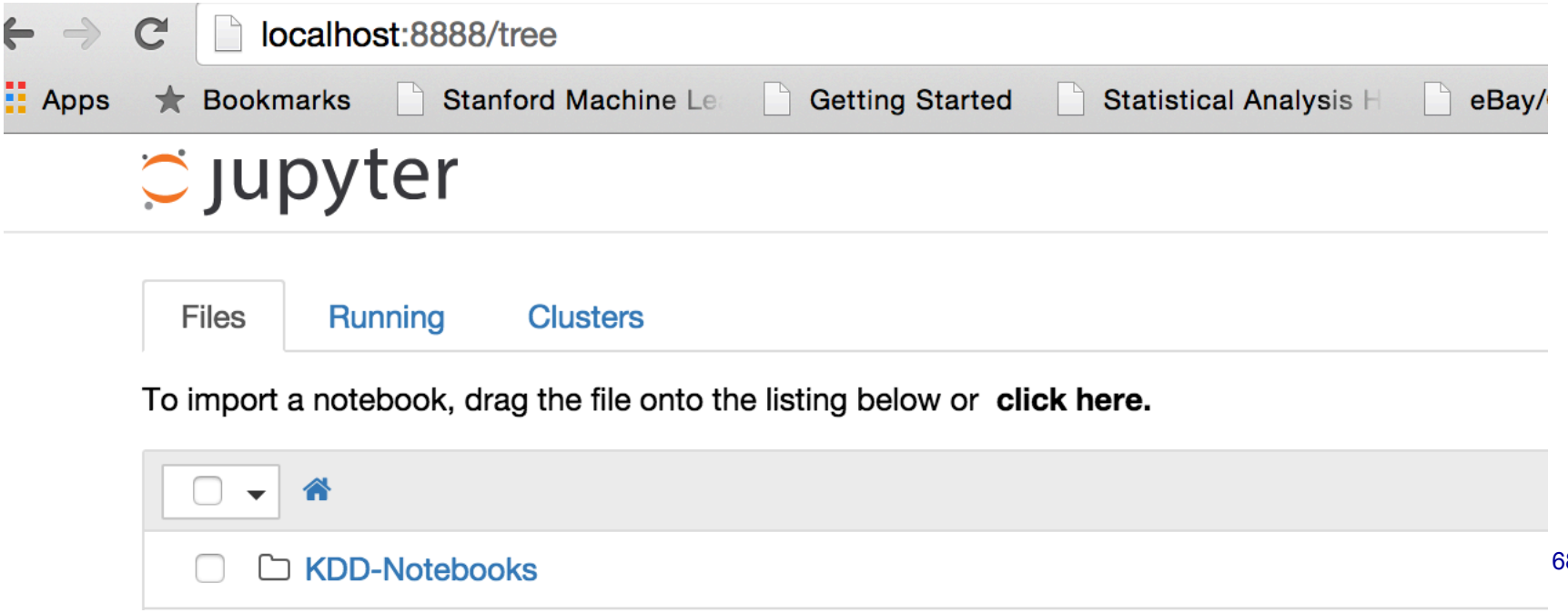
 # from the shell/terminal command line

#Change directory to you working directory

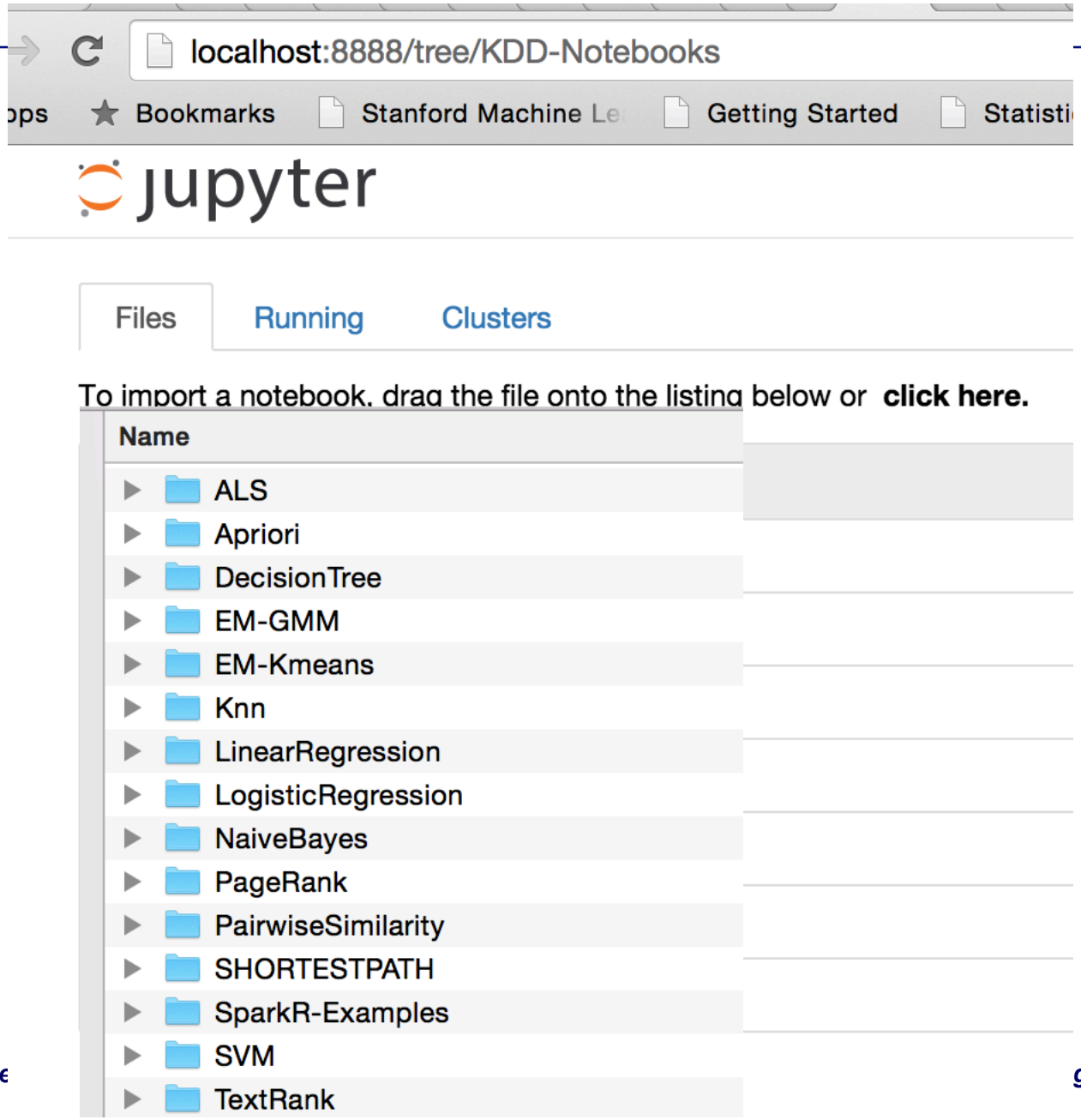
untar CIKM-notebooks

cd CIKM-notebooks

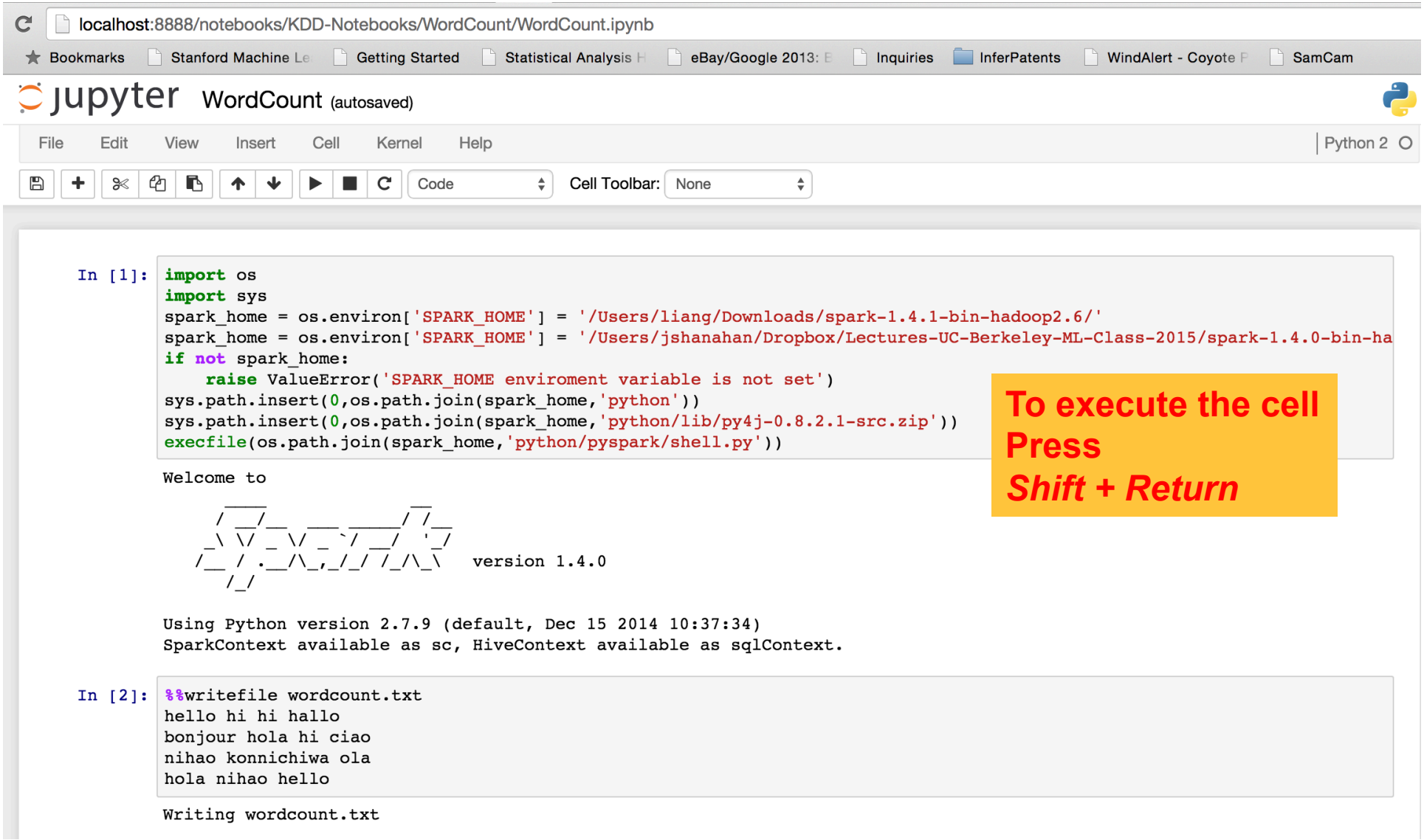
ipython notebook &



Notebooks for Tutorial

- 

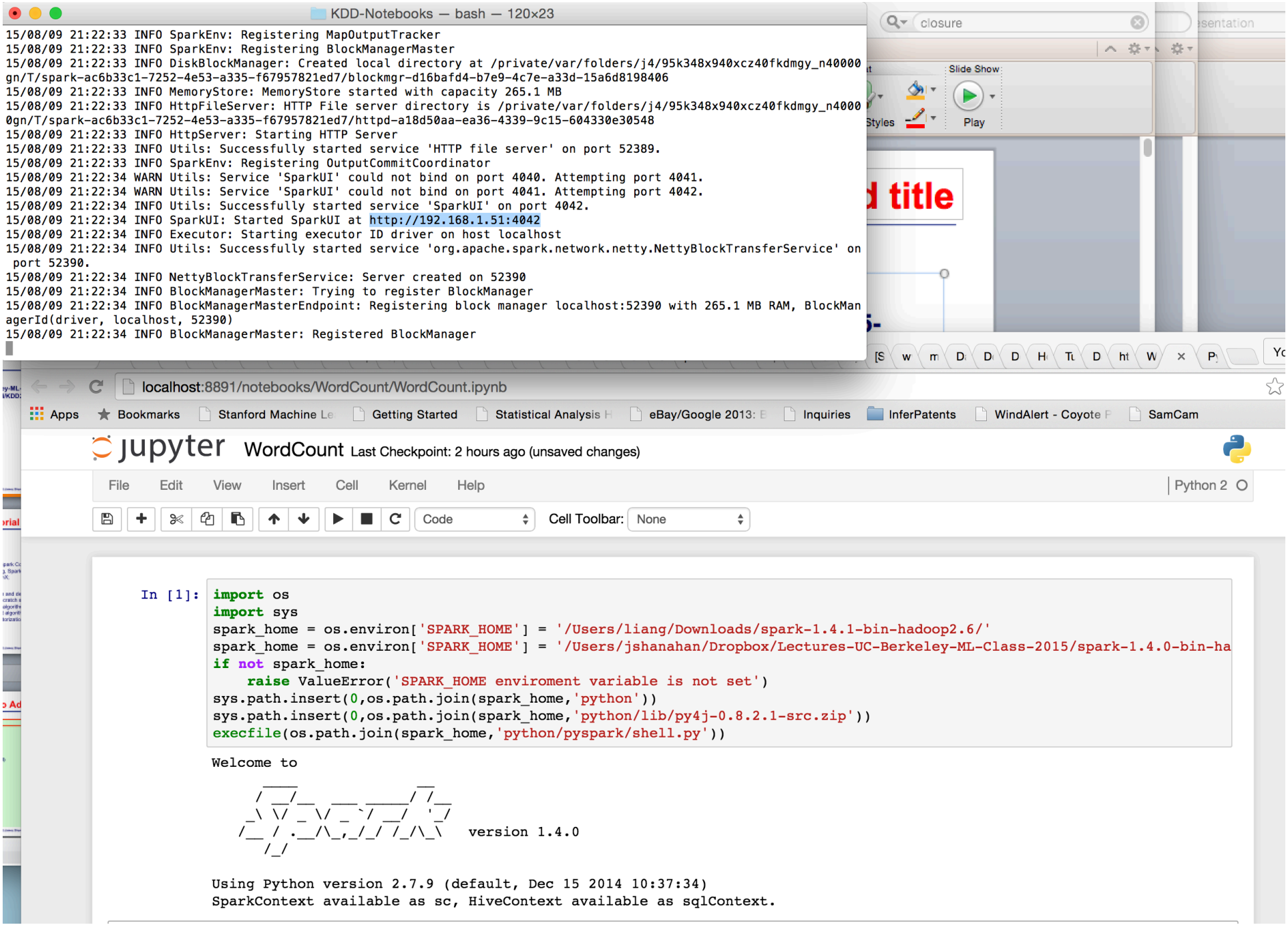
Modify path for SPARK_HOME

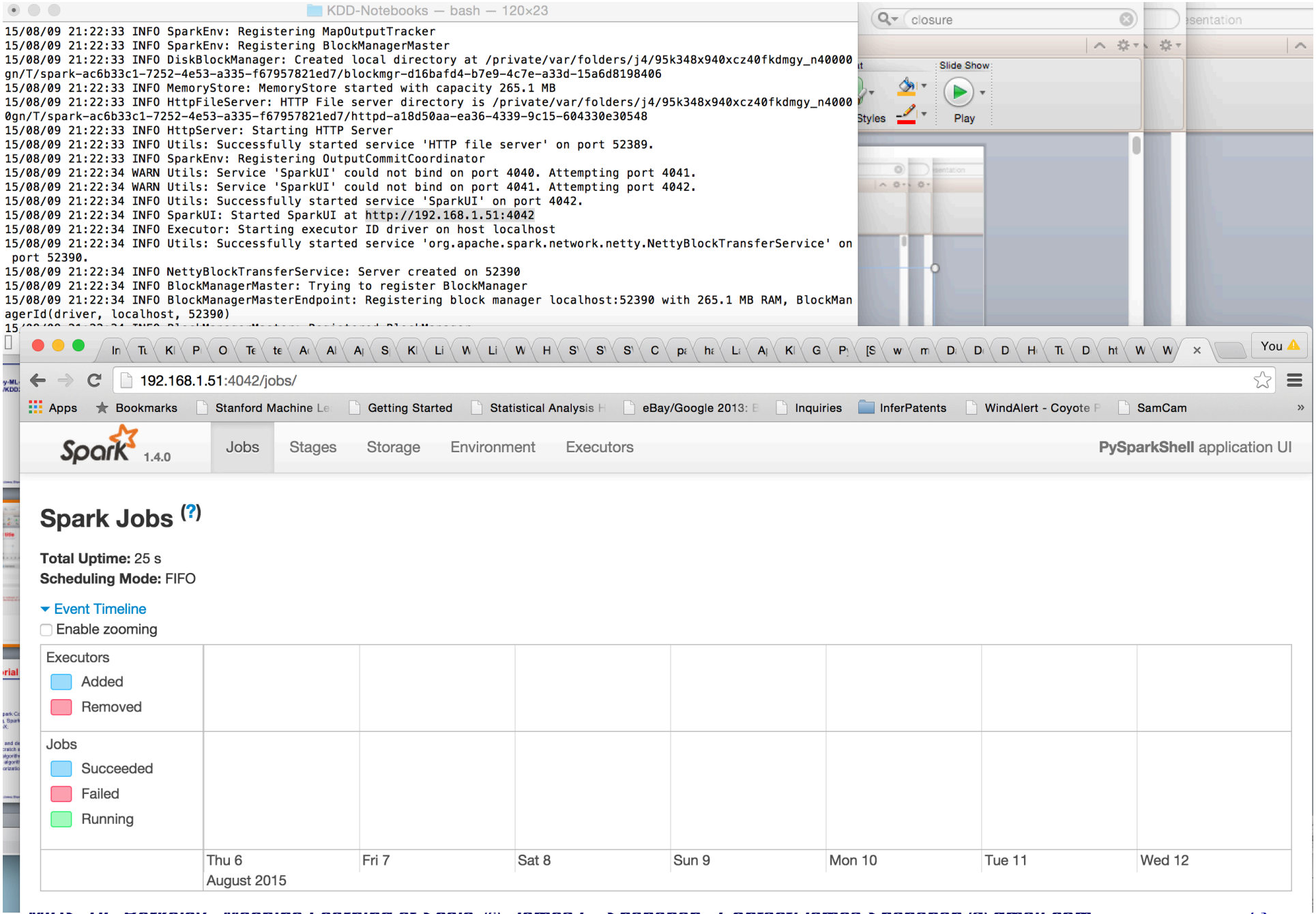


WordCount example test

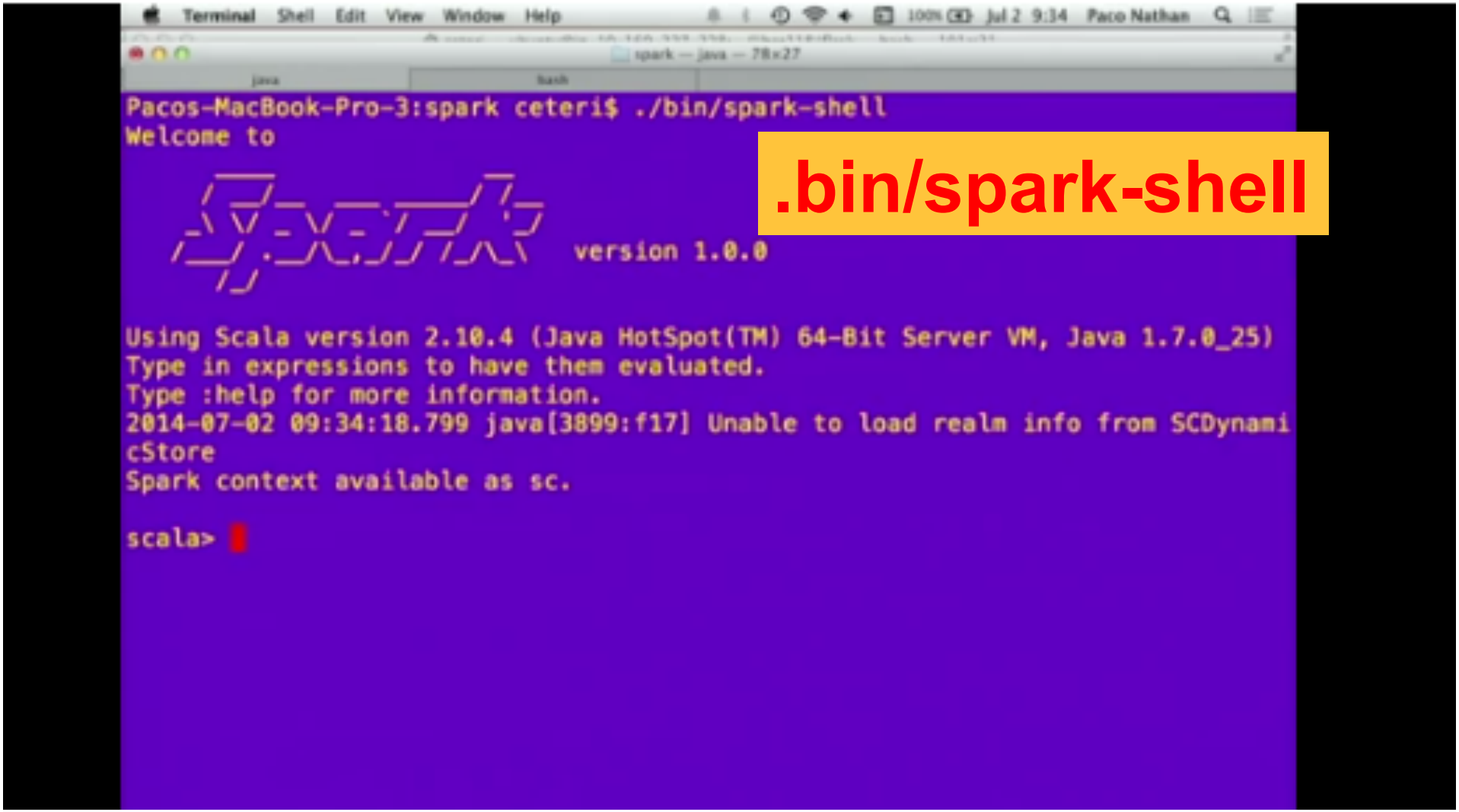
- https://www.dropbox.com/s/ul0l3q98w54dr8x/WordCount.ipynb?dl=0

- **Will discuss in detail later**

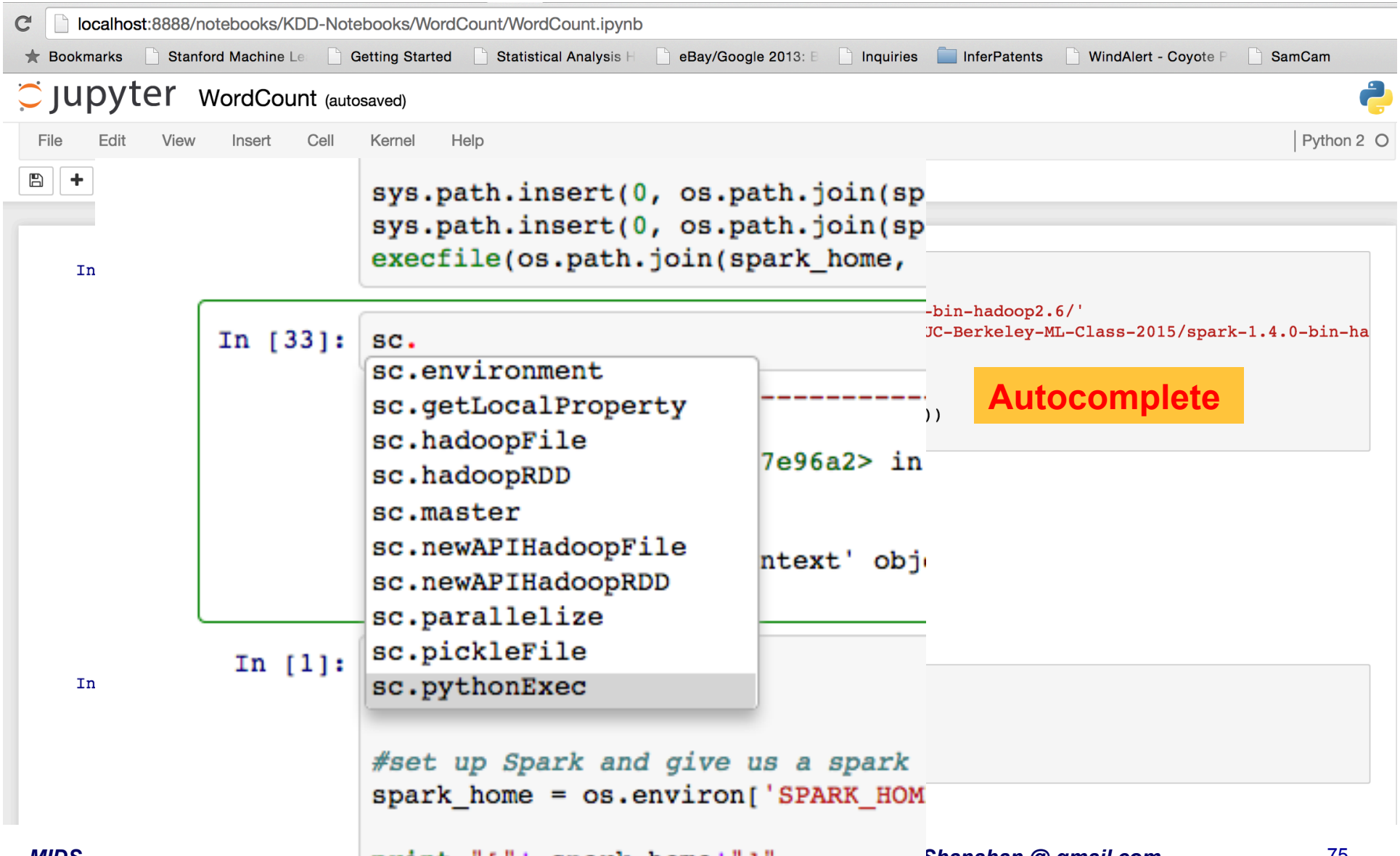

KDD-Notebooks — bash — 120×23
15/08/09 21:22:33 INFO SparkEnv: Registering MapOutputTracker
15/08/09 21:22:33 INFO SparkEnv: Registering BlockManagerMaster
15/08/09 21:22:33 INFO DiskBlockManager: Created local directory at /private/var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/spark-ac6b33c1-7252-4e53-a335-f67957821ed7/blockmgr-d16bafd4-b7e9-4c7e-a33d-15a6d8198406
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_n40000gn/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
closure
Slide Show
Play
Styles
d title
localhost:8891/notebooks/WordCount/WordCount.ipynb
Apps
Bookmarks
Stanford Machine Le
Getting Started
Statistical Analysis H
eBay/Google 2013: E
Inquiries
InferPatents
WindAlert - Coyote F
SamCam
jupyter
WordCount
Last Checkpoint: 2 hours ago (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Help
Python 2
Code
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-ha
if not spark_home:
raise ValueError('SPARK_HOME enviroment 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 — 120×23
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_n40000gn/T/spark-ac6b33c1-7252-4e53-a335-f67957821ed7/blockmgr-d16bafd4-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_n40000gn/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)
closure
Slide Show
Styles
Play
192.168.1.51:4042/jobs/
Apps
Bookmarks
Stanford Machine Le
Getting Started
Statistical Analysis H
eBay/Google 2013: E
Inquiries
InferPatents
WindAlert - Coyote F
SamCam
Spark 1.4.0
Jobs
Stages
Storage
Environment
Executors
PySparkShell application UI
Spark Jobs (?)
Total Uptime: 25 s
Scheduling Mode: FIFO
Event Timeline
Enable zooming
Executors
Added
Removed
Jobs
Succeeded
Failed
Running
Thu 6
August 2015
Fri 7
Sat 8
Sun 9
Mon 10
Tue 11
Wed 12

Spark shell in Scala



TAB lists the available methods and attributes



Live Session Outline

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
- **Week**
  - Homework HW7, HW9, HW10
  - AWS: HW10 local
  - Running an MRJob on cloud with the latest version of MRJob, Screen
  - Async lecture recap plus Q&A Spark
  - Install Spark
  - Spark, RDDs, Patterns in Spark
  - DataFrames, Spark SQL
  - Secondary Sorts in Spark
  - MLLib: brief intro
  - Homegrown Kmeans in Spark
- **Wrapup**
  - Finish RECORDING (bonus points for reminding me!)

Spark

In-Memory Cluster Computing for Iterative and Interactive Applications

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 a 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 ofnumerical weather prediction.
  - Discrete Fourier Transform where each harmonic is independently calculated.
  - Many many more….

Not everything is a MR

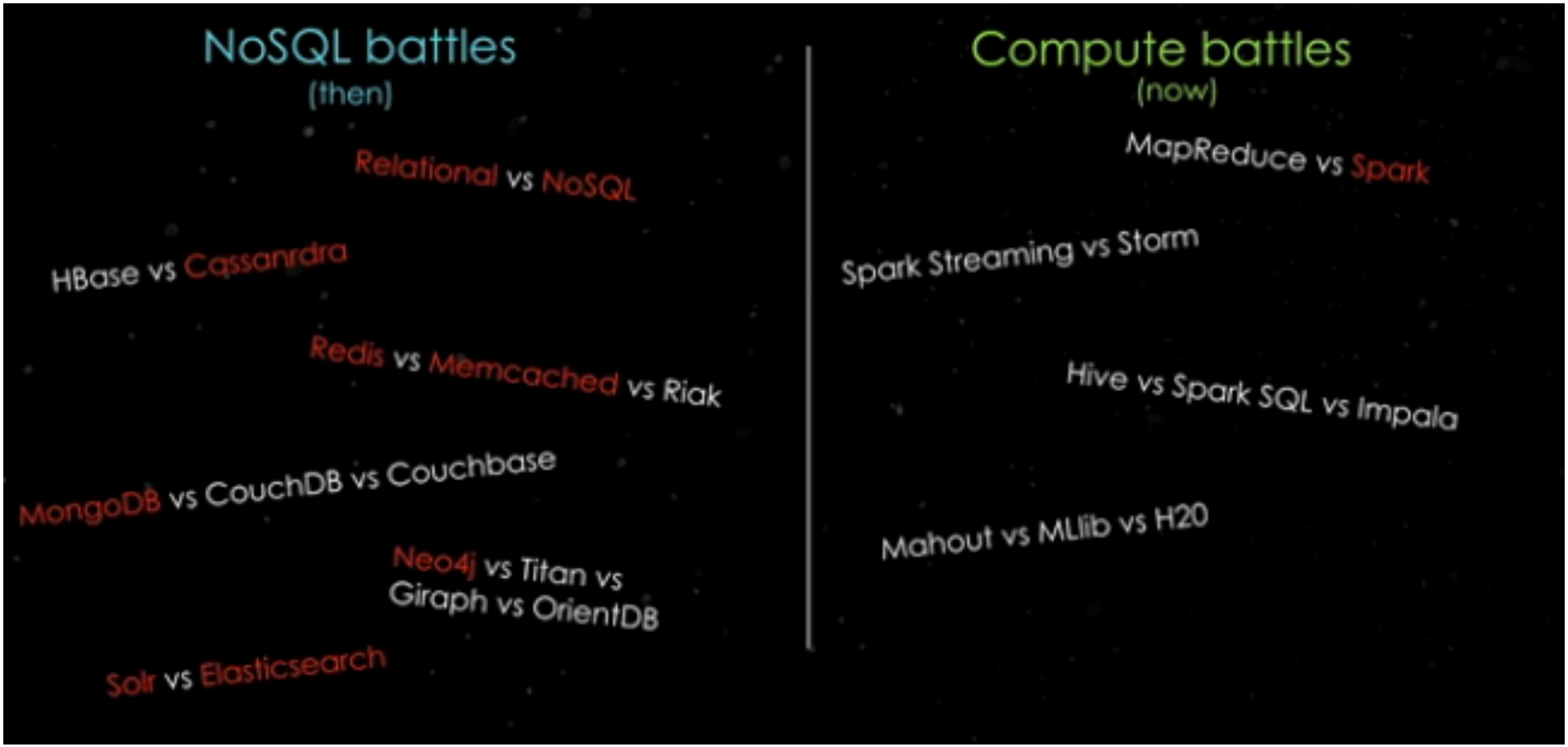
- MRs are ideal for "embarrassingly parallel" problems
- Very little communication
- Easily distributed
- Linear computational flow

- Happy to report that most of the machine learning algorithms of practical importance are embarrassingly parallel

First generation MapReduce

- **0th Generation**
  - Command line
- **1st Generation**
  - MapReduce, Hadoop
  - Native versus Streaming
- **2nd Generation**
  - MrJob, Scalding
  - Write Map-Reduce pipelines
- **3rd Generation**
  - Spark
  - Memory backed, Rich API: 80 operations

Storage vs Processing Wars



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, SQL (1.4)**
- **Interactive shell**

Spark Metrics

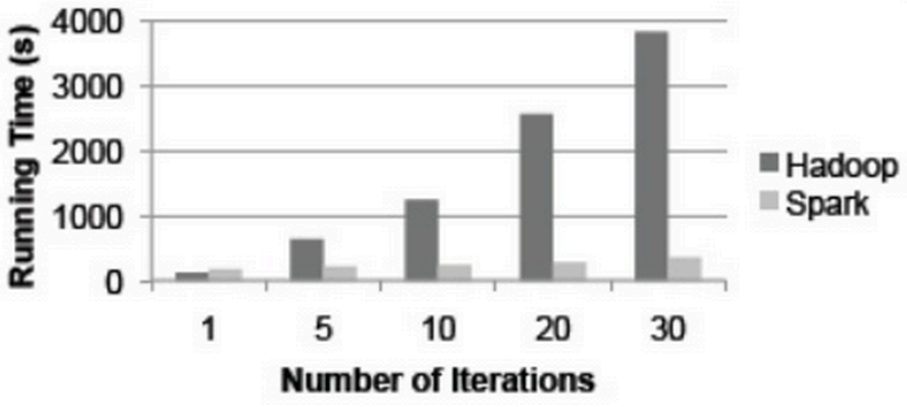
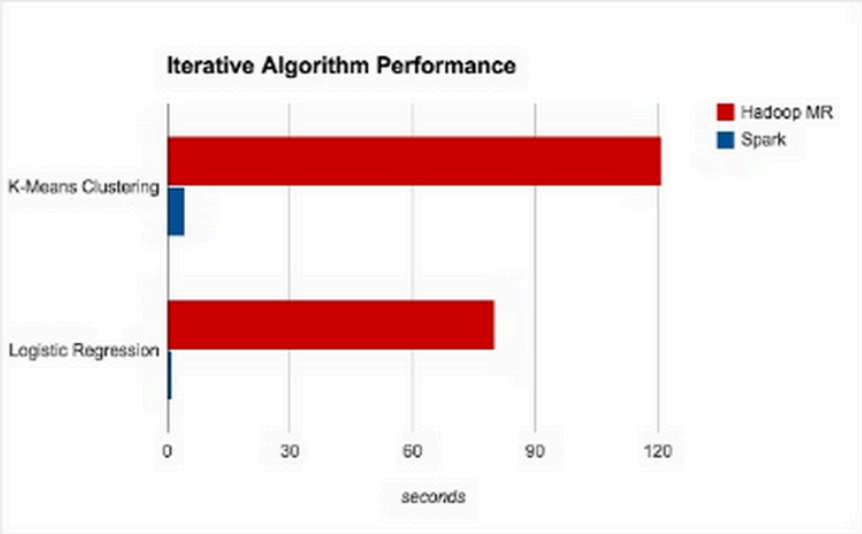
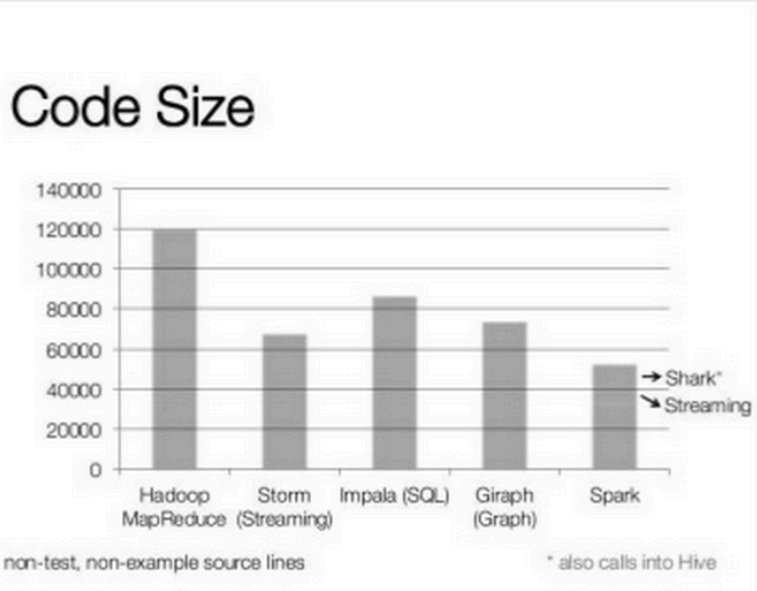


Figure 2: Logistic regression performance in Hadoop and Spark.





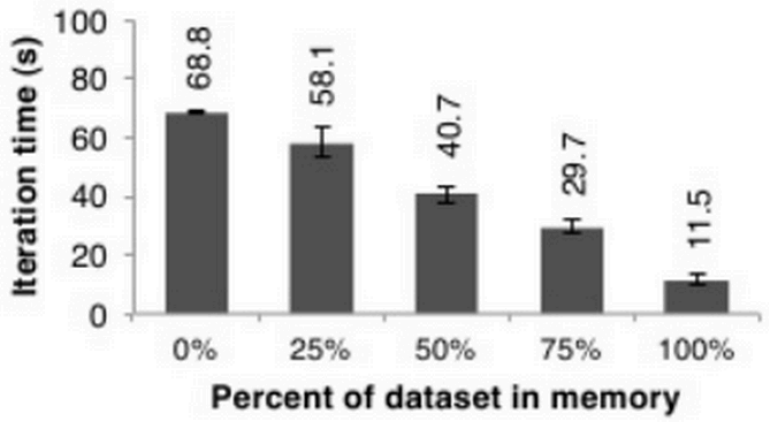


Figure 12: Performance of logistic regression using 100 GB data on 25 machines with varying amounts of data in memory.

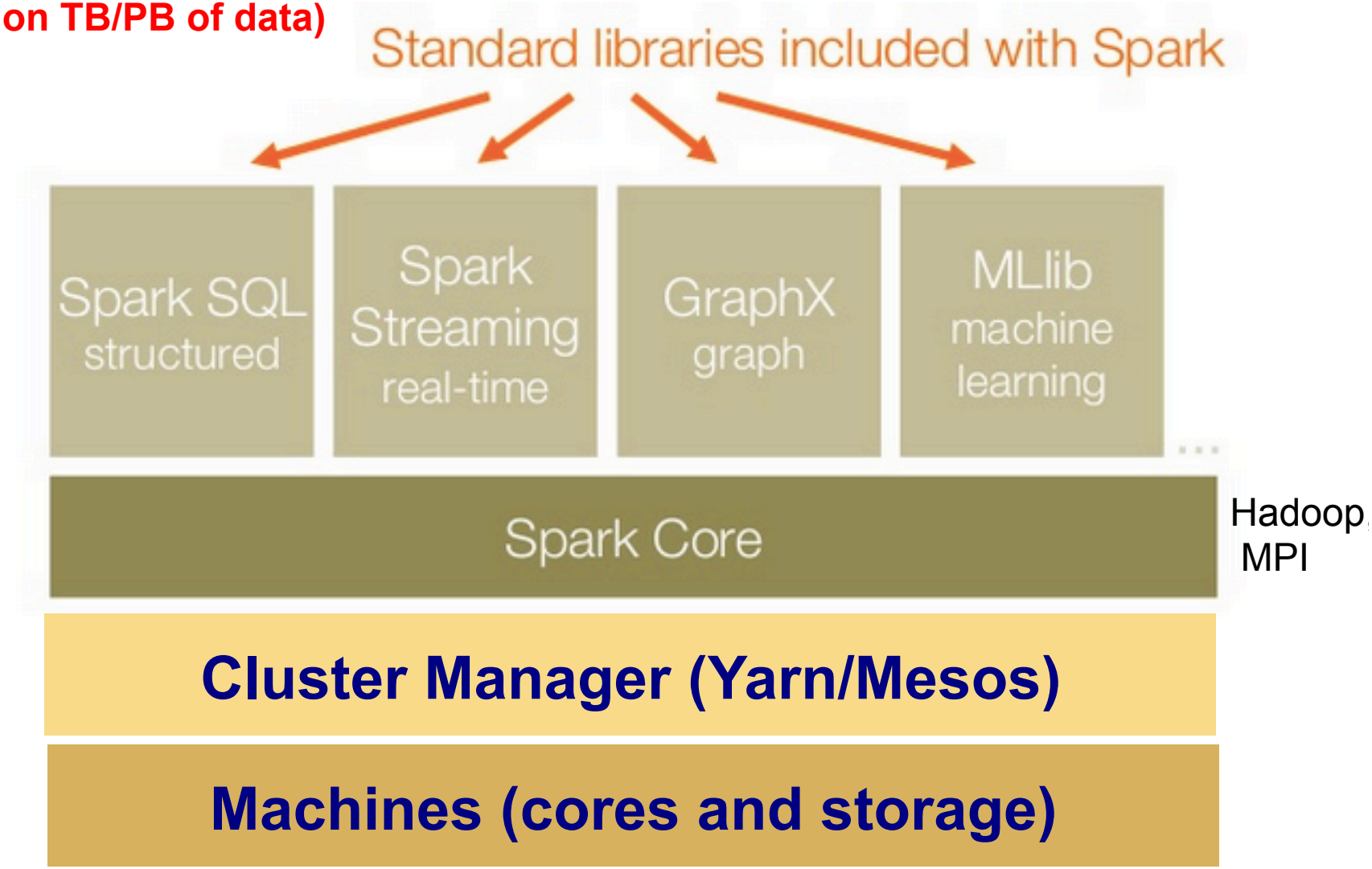
Spark

- **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)

- .

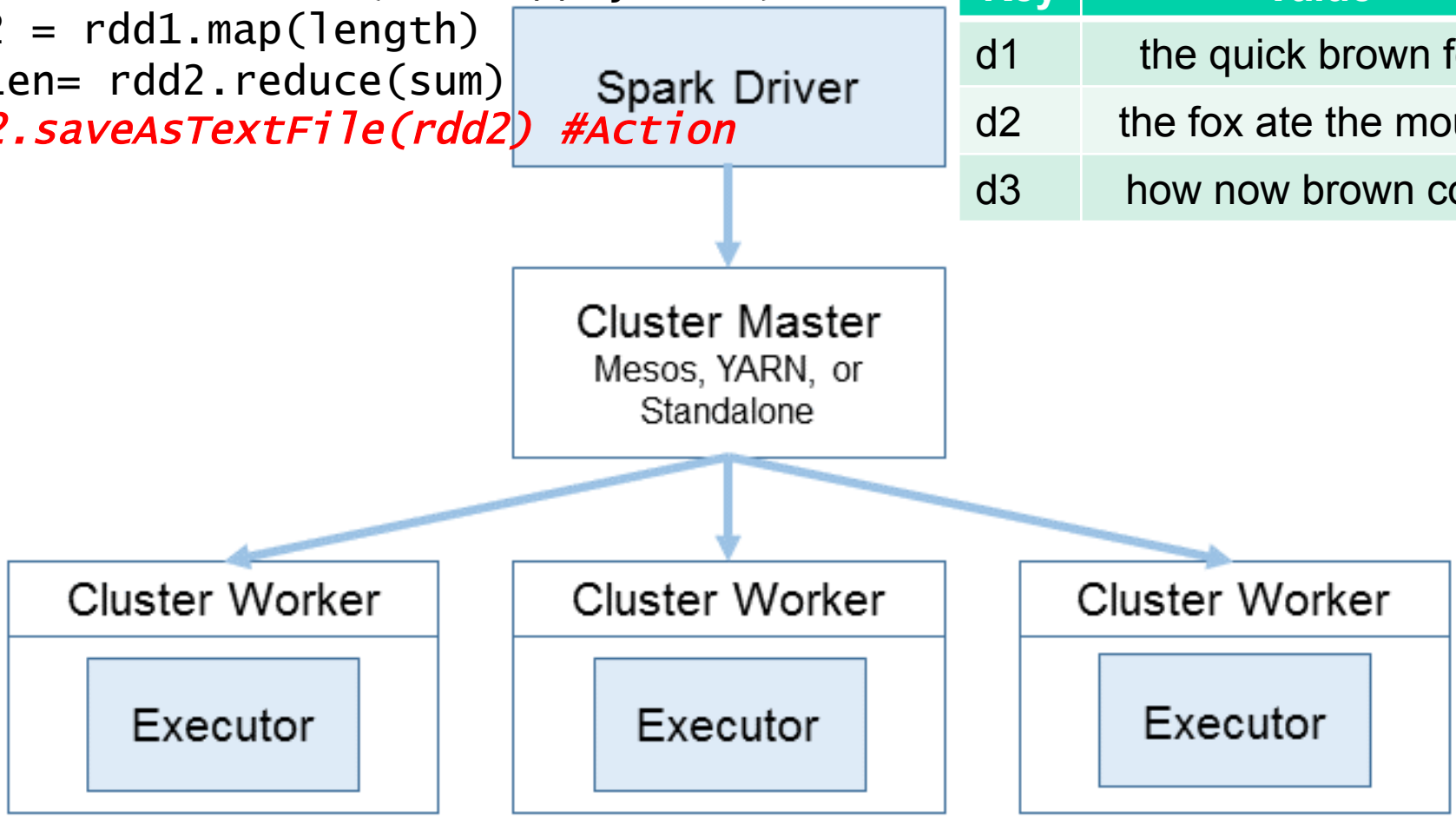


Spark in local mode on a single computer or on a Spark Cluster

- **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) #Action
```

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



Life of a Spark Program

1) Create some input RDDs from external data or parallelize a collection in your driver program.

2) Lazily ***transform*** them to define new RDDs using transformations like `filter()` or `map()`

3) Ask Spark to `cache()` any intermediate RDDs that will need to be reused.

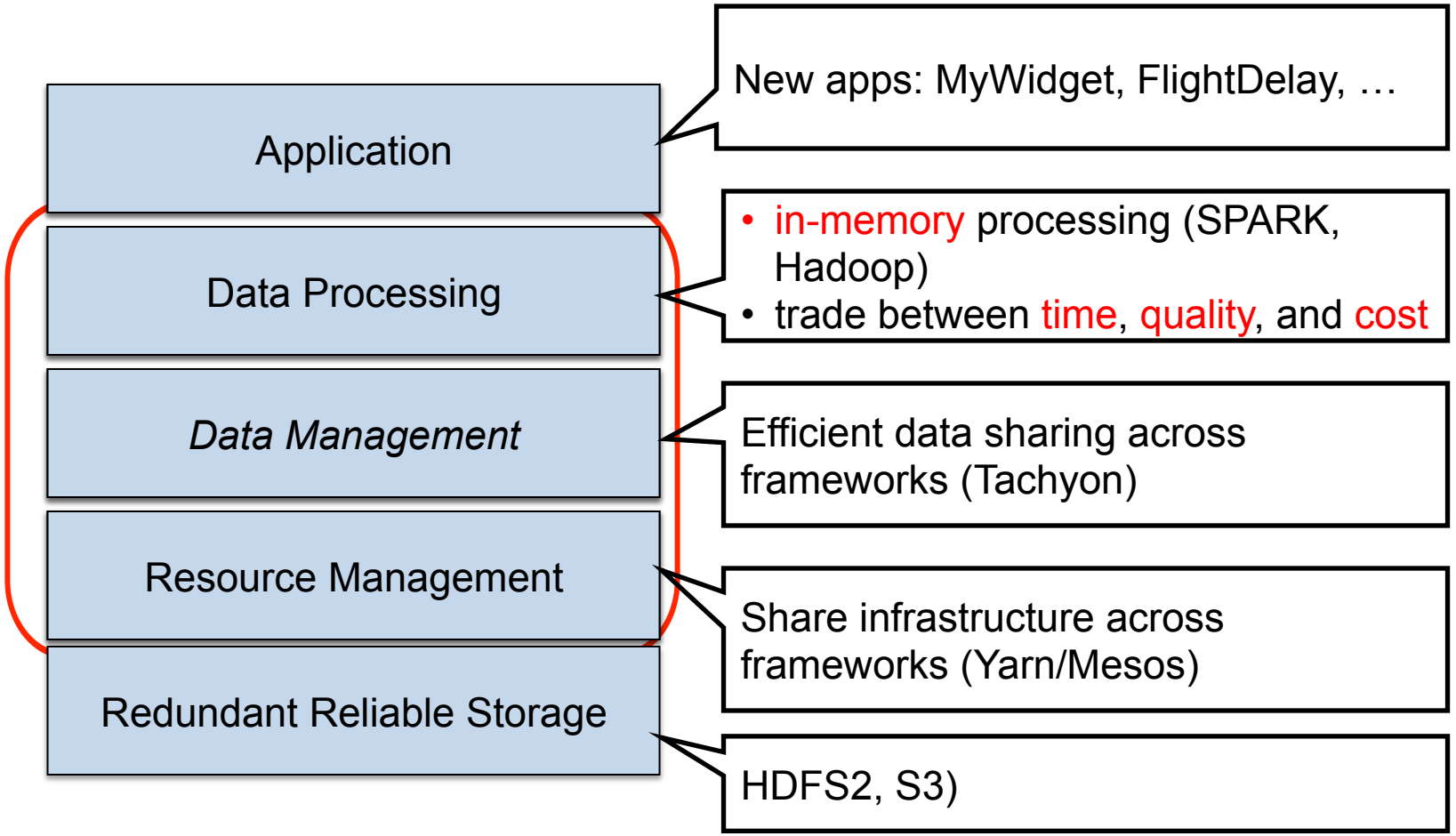
4) Launch ***actions*** such as `count()` and `collect()` to kick off a parallel computation, which is then optimized and executed by Spark.

Spark APIs

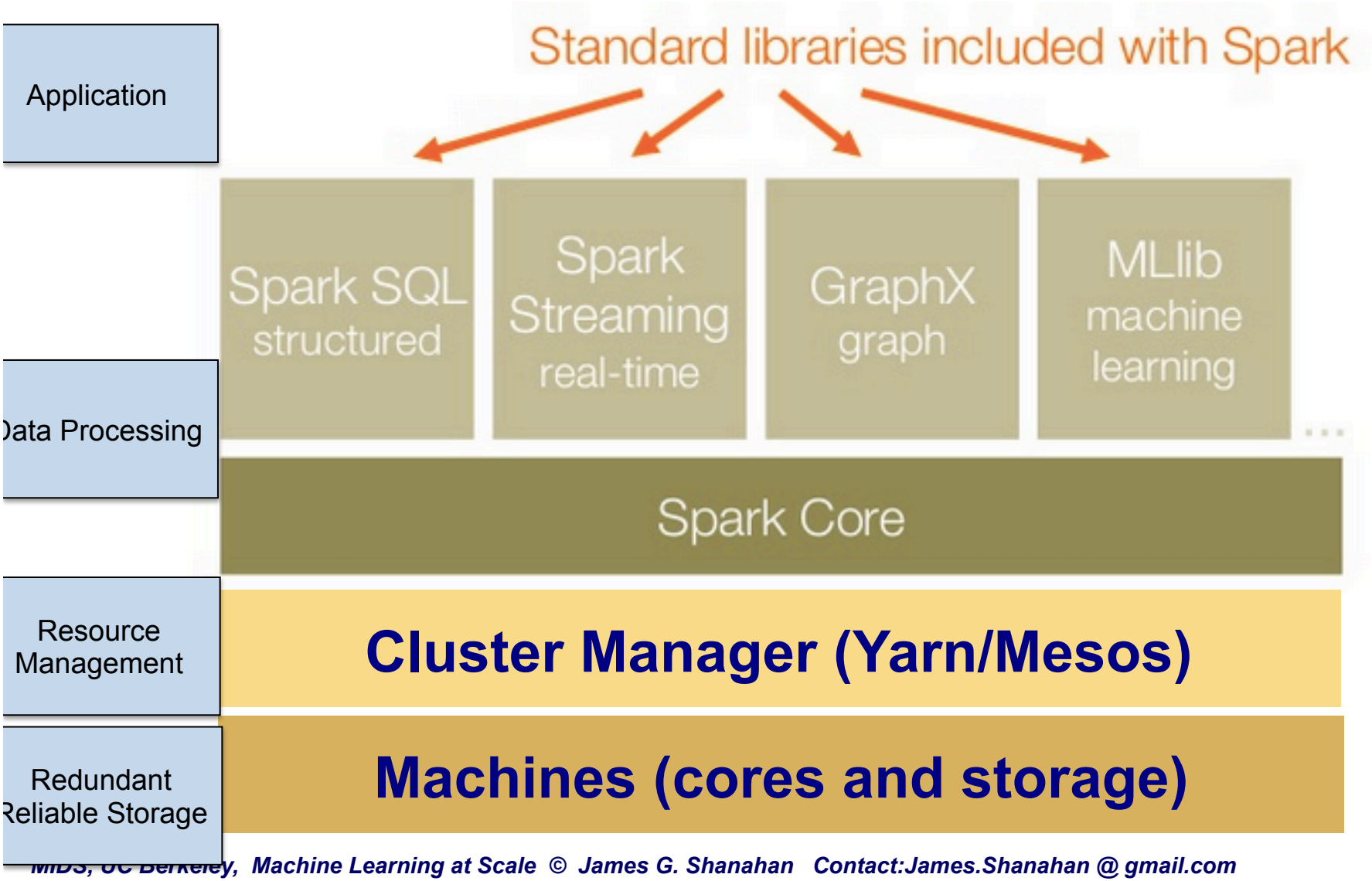
- **API**
  - In computing, a binding from a programming language to a library or operating system service is an application programming interface (API) providing glue code to use that library or service in a particular programming language.
- **Apache Spark Apache Spark is an alternative big data computing 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/Zeppelin Notebook as a Unified Data Science Interface to all of this)

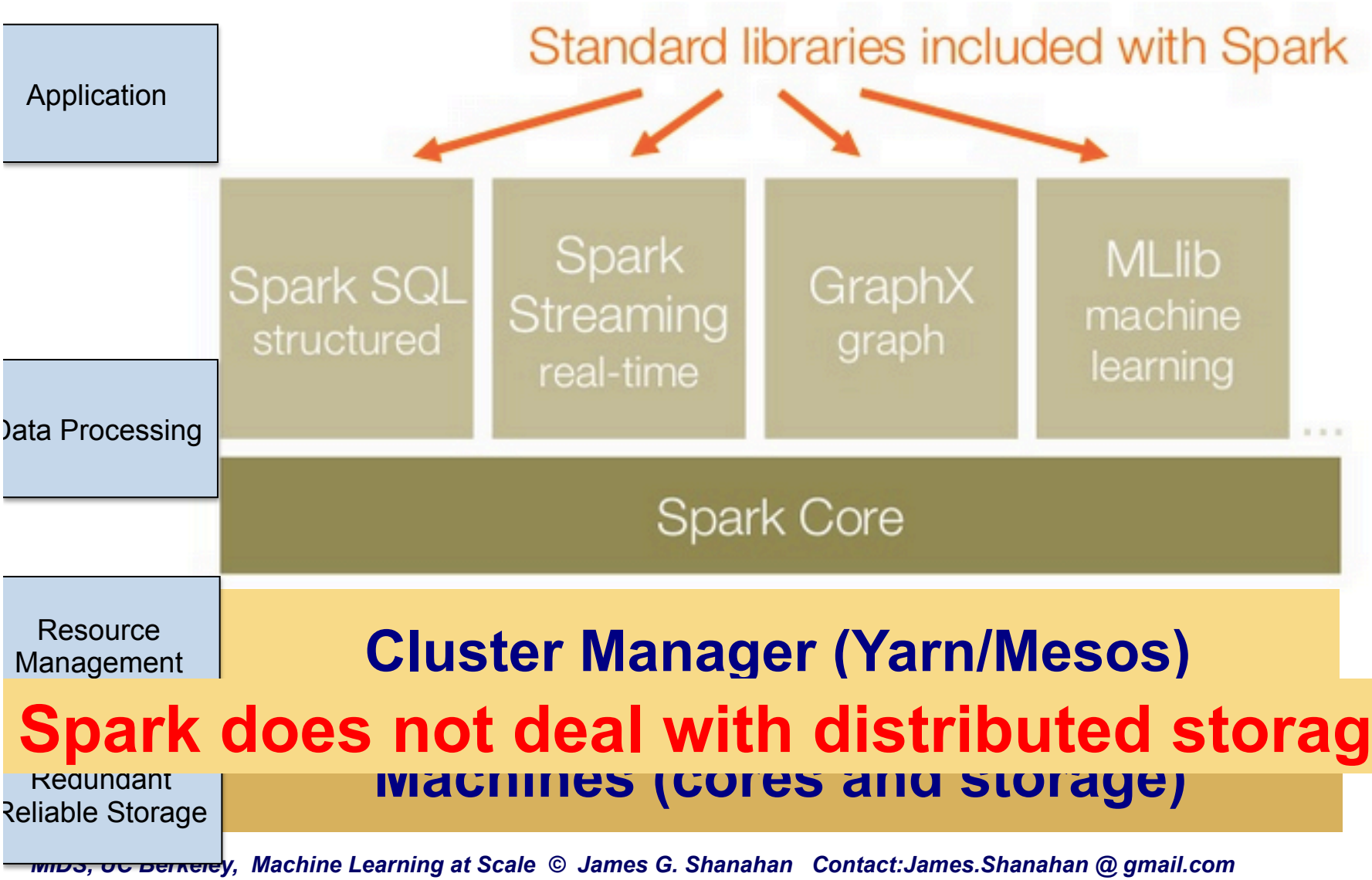
Berkeley Data Analytics Stack (BDAS)



Spark


Application
Data Processing
Resource Management
Redundant Reliable Storage
Standard libraries included with Spark
Spark SQL structured
Spark Streaming real-time
GraphX graph
MLlib machine learning
...
Spark Core
Cluster Manager (Yarn/Mesos)
Machines (cores and storage)

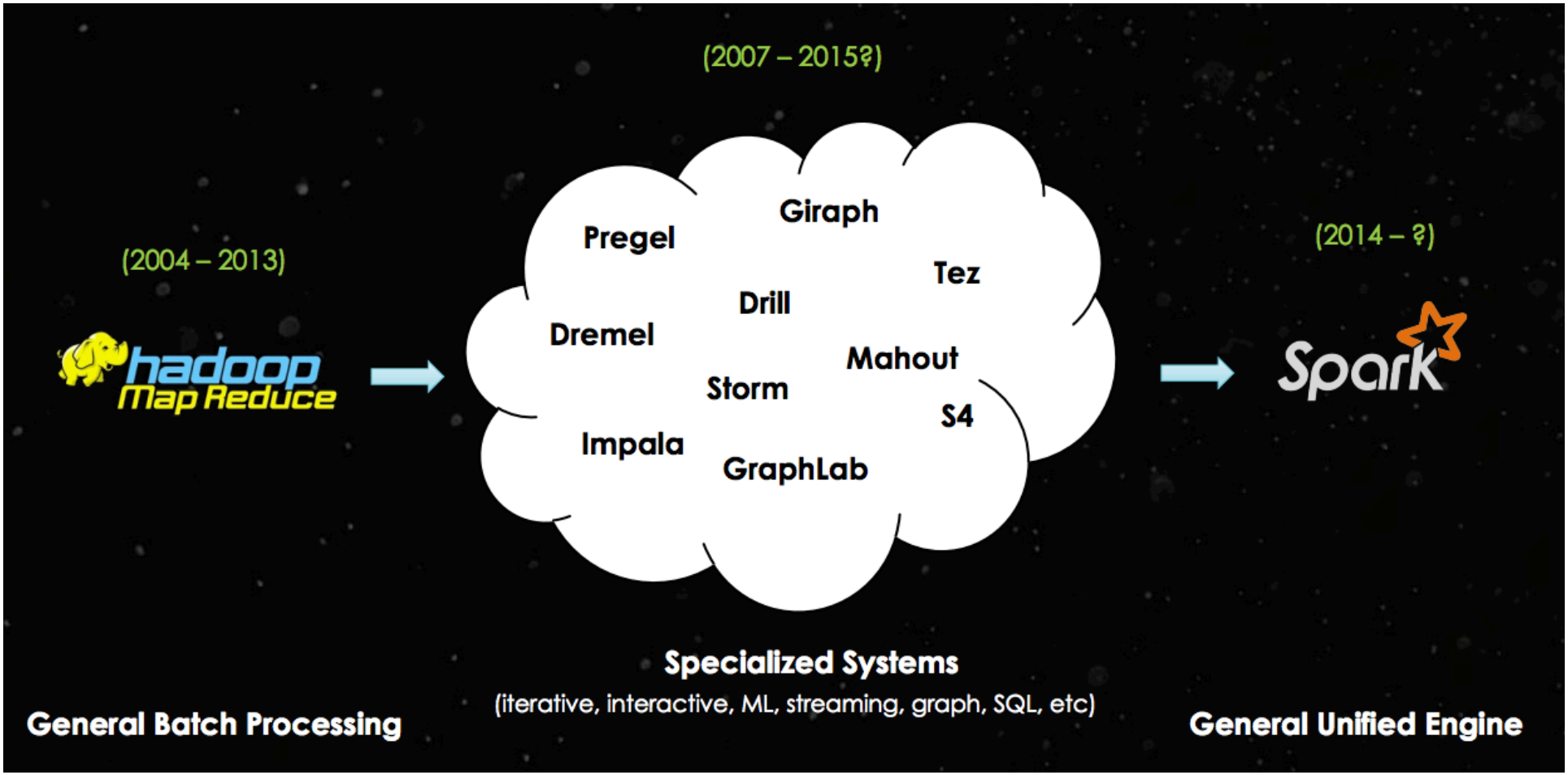
Spark


Application
Standard libraries included with Spark
Spark SQL
structured
Spark
Streaming
real-time
GraphX
graph
MLlib
machine
learning
...
Data Processing
Spark Core
Resource
Management
Cluster Manager (Yarn/Mesos)
Spark does not deal with distributed storage
Redundant
Reliable Storage
Machines (cores and storage)

Divide and conquer with Closures

- **Decompose problems in non-overlapping sub-problems**
- **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 …



Google, Yahoo, Facebook, Twitter, MetaMarkets, DataBricks and many more

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

Live Session Outline

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
- **Week**
  - Homework HW7, HW9, HW10
  - AWS: HW10 local
  - Running an MRJob on cloud with the latest version of MRJob, Screen
  - Async lecture recap plus Q&A Spark
  - Install Spark
  - Spark, RDDs, Patterns in Spark
  - DataFrames, Spark SQL
  - Secondary Sorts in Spark
  - MLLib: brief intro
  - Homegrown Kmeans in Spark
- **Wrapup**
  - Finish RECORDING (bonus points for reminding me!)

Spark details

- RDDs
- Combiners
- WordCount Example (and debugging in Spark)
- Broadcast Variables
- FlatMaps
- Partition functions

Resilient distributed dataset (RDD)

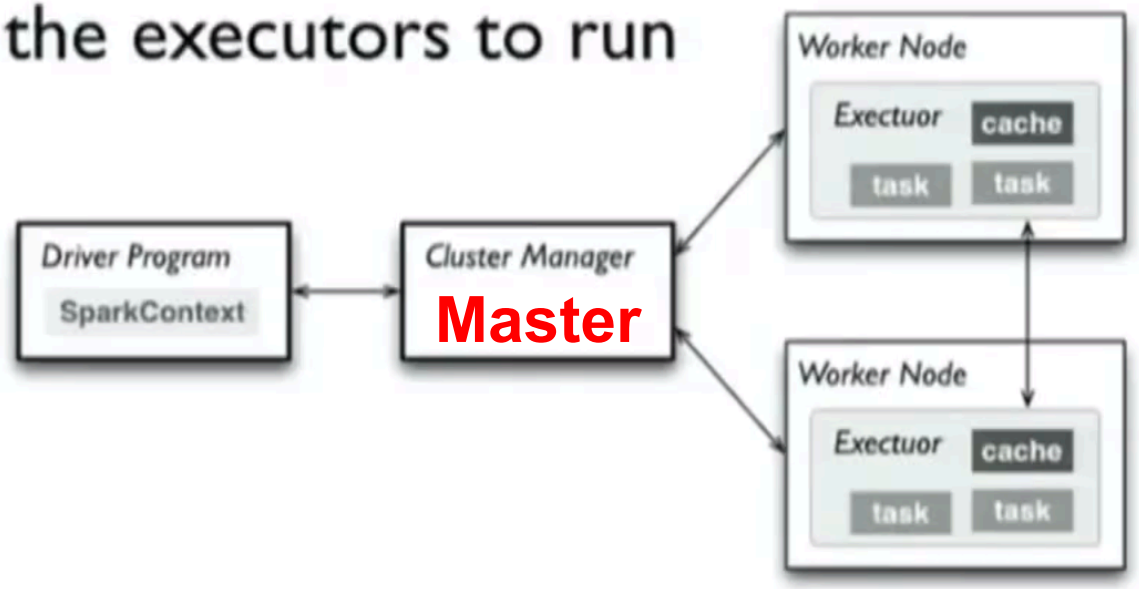
- An RDD is simply a distributed collection of elements (Key-Value records).
- In Spark all work is expressed as either creating new RDDs, transforming existing RDDs, or calling operations on RDDs to compute a result.

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

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



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

```
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]
```

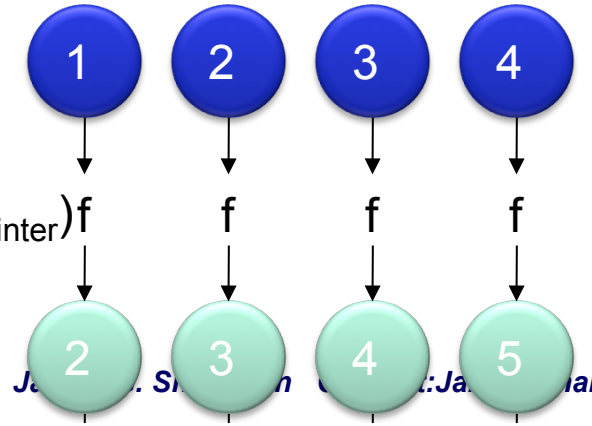
Distribute | Map | Gather back to driver

- **Spark is oriented around *Key-Value* records**

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

- **Map function:**

Map(Key_{in}, $Value_{in}$) ➔ list(K_{inter}, V_{inter})



f(x)=x × 1

RDD with 4 elements

Distribute

Map

Gather back to driver

```
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 [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 fucntion: %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:

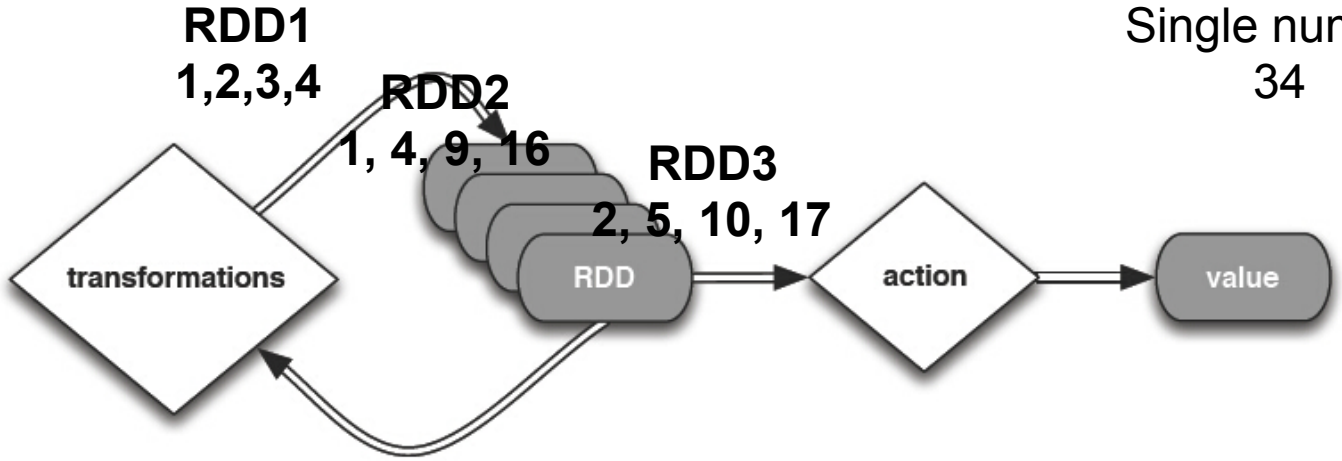
Spark Essentials: *RDD*

Slide Improvisation

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.

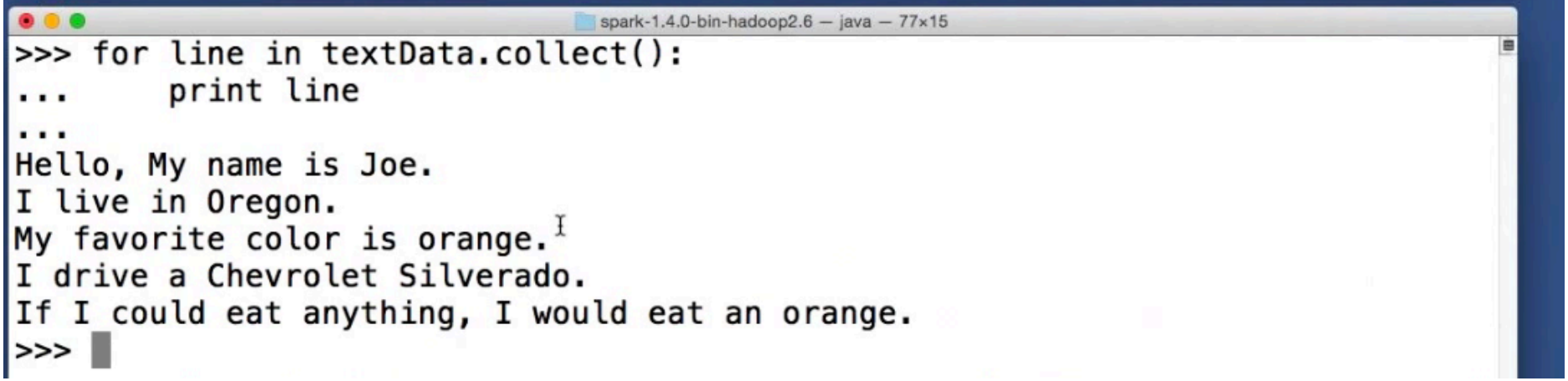
Spark supports text files, SequenceFiles, and any other Hadoop `InputFormat`, and can also take a directory or a glob (e.g. `/data/201404*`)

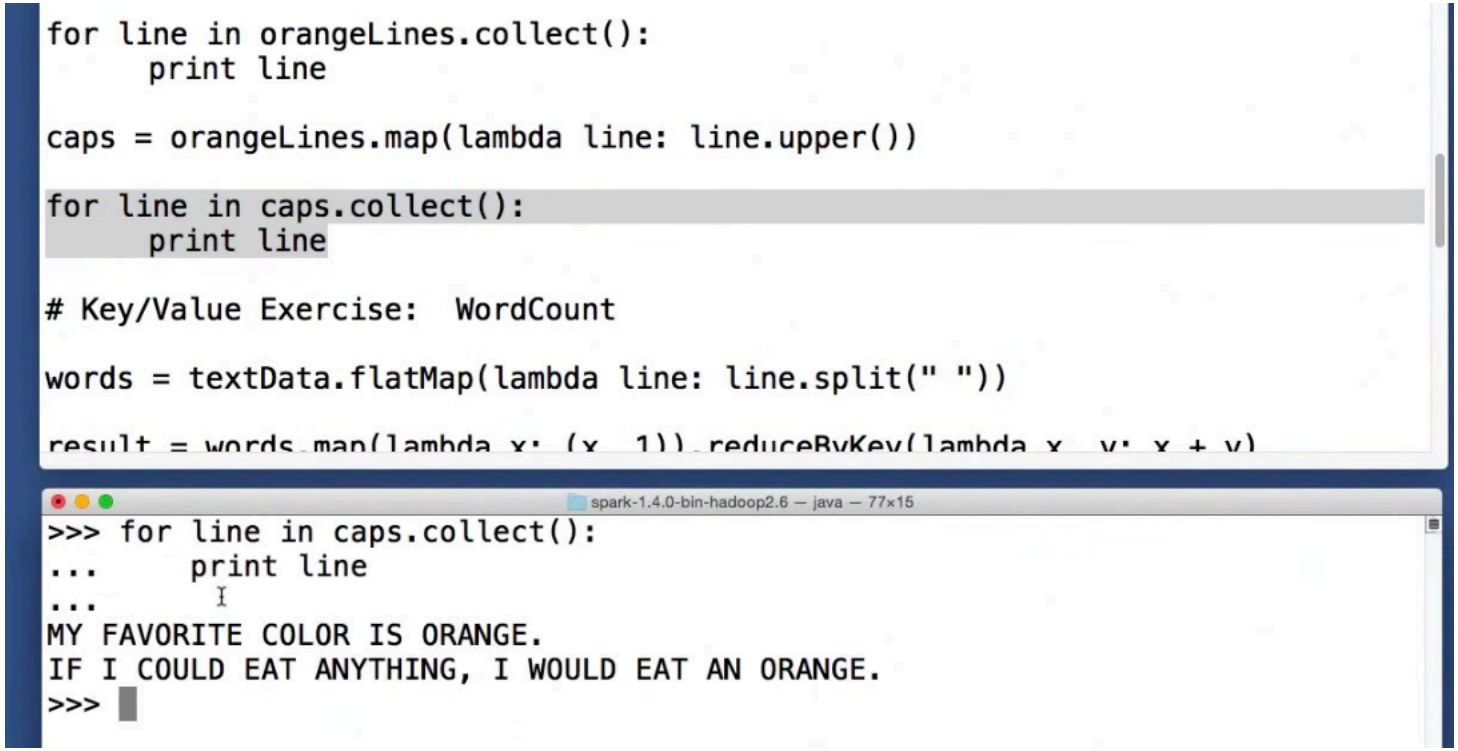
sc.parallelize([1,2,3,4]).map(powerOfTwo).map(plusOne).reduce(lambda x, y: x + y)



- **Does Spark process each record one by one?**
- **NO! It does things in parallel (think of shuffle, barrier sync etc..**

```
# Create an RDD from a local text file

textData = sc.textFile("textData.txt")

# View the contents of the RDD

for line in textData.collect():
    print line

# Lazily filter any lines that contain the word "orange"

orangeLines = textData.filter(lambda line: "orange" in line)
```



- 
for line in orangeLines.collect():
print line
caps = orangeLines.map(lambda line: line.upper())
for line in caps.collect():
print line
# Key/Value Exercise: WordCount
words = textData.flatMap(lambda line: line.split(" "))
result = words.map(lambda x: (x, 1)).reduceByKey(lambda x, y: x + y)
spark-1.4.0-bin-hadoop2.6 — java — 77×15
>>> for line in caps.collect():
... print line
...
MY FAVORITE COLOR IS ORANGE.
IF I COULD EAT ANYTHING, I WOULD EAT AN ORANGE.
>>>

```
from pyspark.mllib.classification import NaiveBayes, NaiveBayesModel
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.regression import LabeledPoint

def parseLine(line):
    parts = line.split(',')
    label = float(parts[0])
    features = Vectors.dense([float(x) for x in parts[1].split(' ')])
    return LabeledPoint(label, features)

data = sc.textFile('sample_naive_bayes_data.txt').map(parseLine)

# Split data aproximately into training (60%) and test (40%)
training, test = data.randomSplit([0.6, 0.4], seed = 0)

# Train a naive Bayes model.
model = NaiveBayes.train(training, 1.0)

# Make prediction and test accuracy.
predictionAndLabel = test.map(lambda p : (model.predict(p.features), p.label))
accuracy = 1.0 * predictionAndLabel.filter(lambda (x, v): x == v).count() / test.count()

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

```
test.collect()
```

```
[LabeledPoint(0.0, [1.0,0.0,0.0]),
 LabeledPoint(1.0, [0.0,2.0,0.0]),
 LabeledPoint(2.0, [0.0,0.0,2.0])]
```

```
for record in test.collect():
    print record.label, record.features
```

```
0.0 [1.0,0.0,0.0]
```

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

```
Reduce by lambda:   34
Reduce by fucntion: 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 of a computation

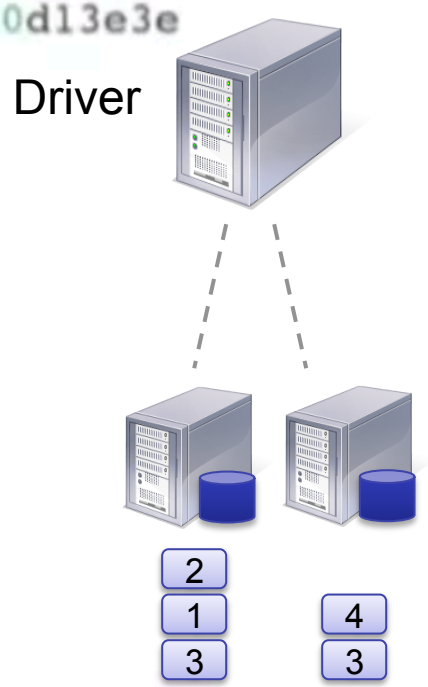
- **A transformed RDD gets (re)computed when an action is run on it**

- **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

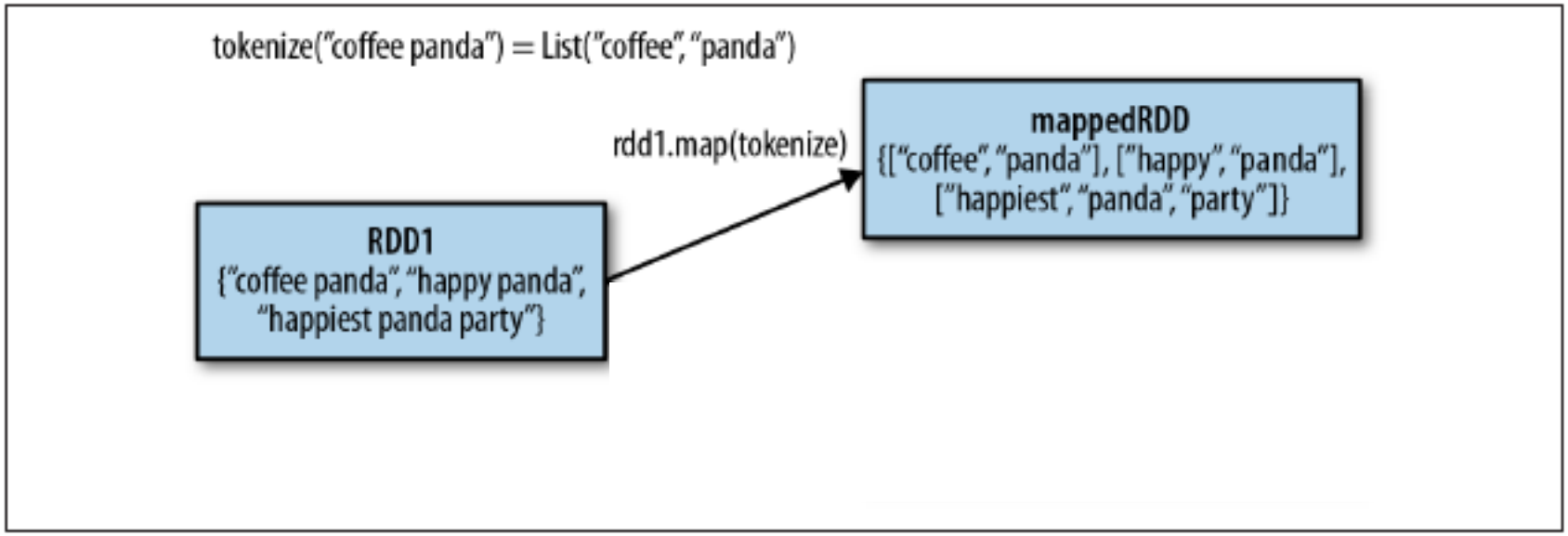
Scala:

```
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
```

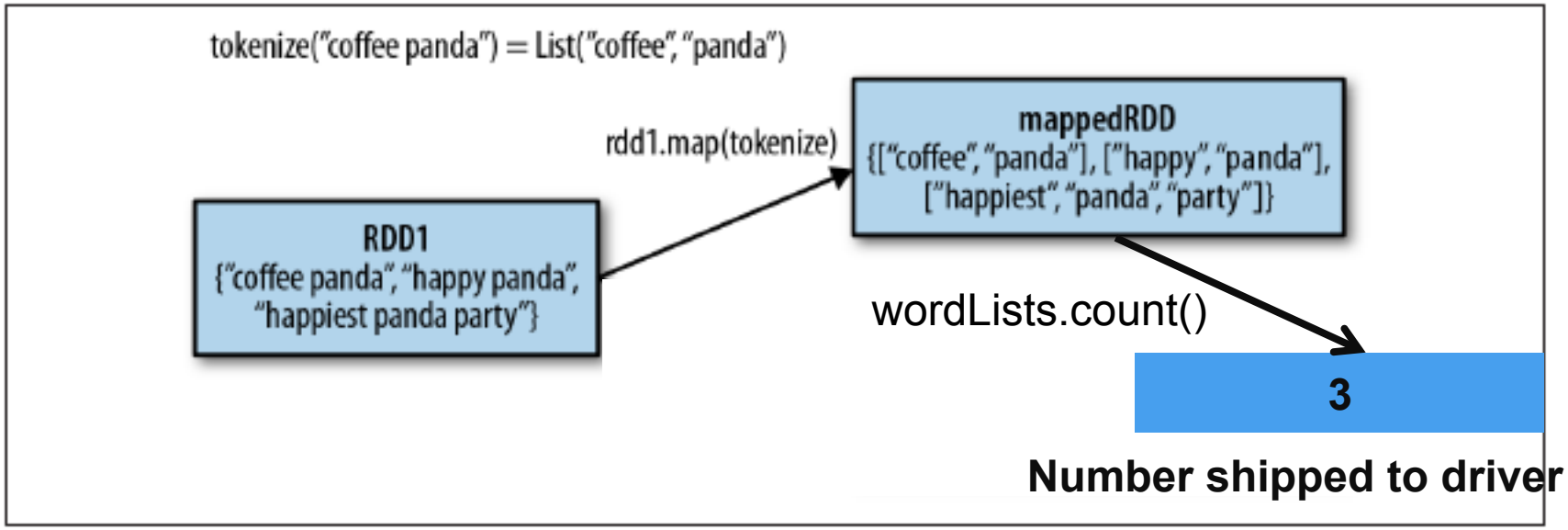
Mapper: tokenize; Apply tokenize to each input record




```
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()
```

counts.toDebugString() #lineage/recipe

```
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()
```

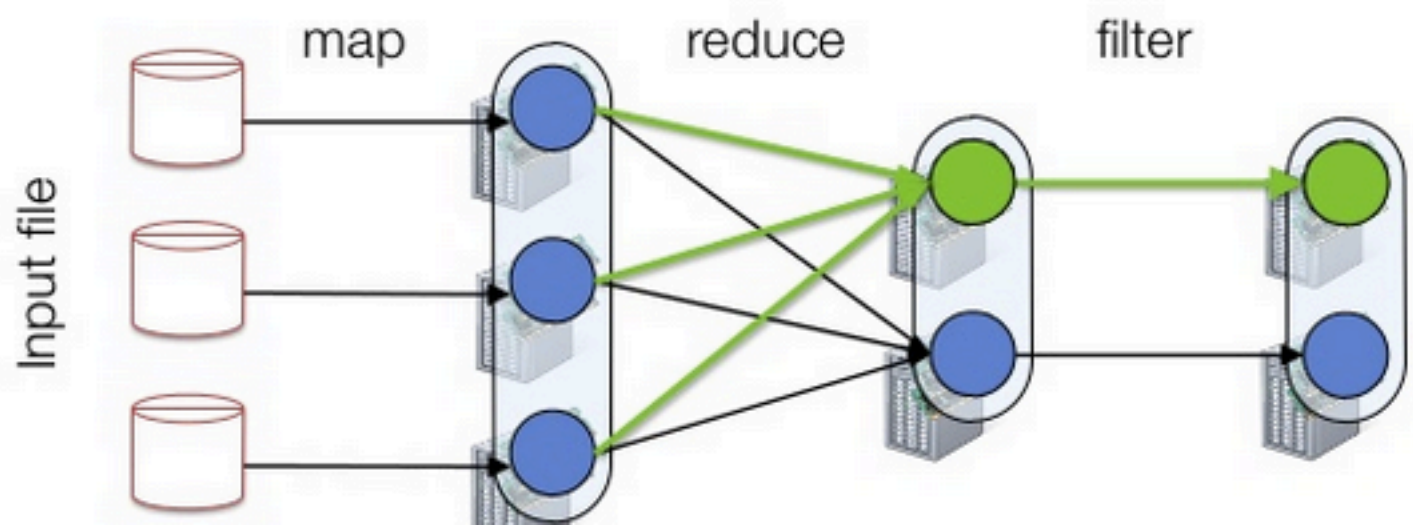
```
Out[7]: '(8) PythonRDD[7] at collect at <ipython-input-5-708c6b1f0496>:4 []\n |  MapPartitionsRDD[6] at mapPartitions at Pytho
        nRDD.scala:346 []\n |  ShuffledRDD[5] at partitionBy at NativeMethodAccessorImpl.java:-2 []\n +-(8) PairwiseRDD[4] at
        reduceByKey at <ipython-input-5-708c6b1f0496>:2 []\n    |  PythonRDD[3] at reduceByKey at <ipython-input-5-708c6b1f049
        6>: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

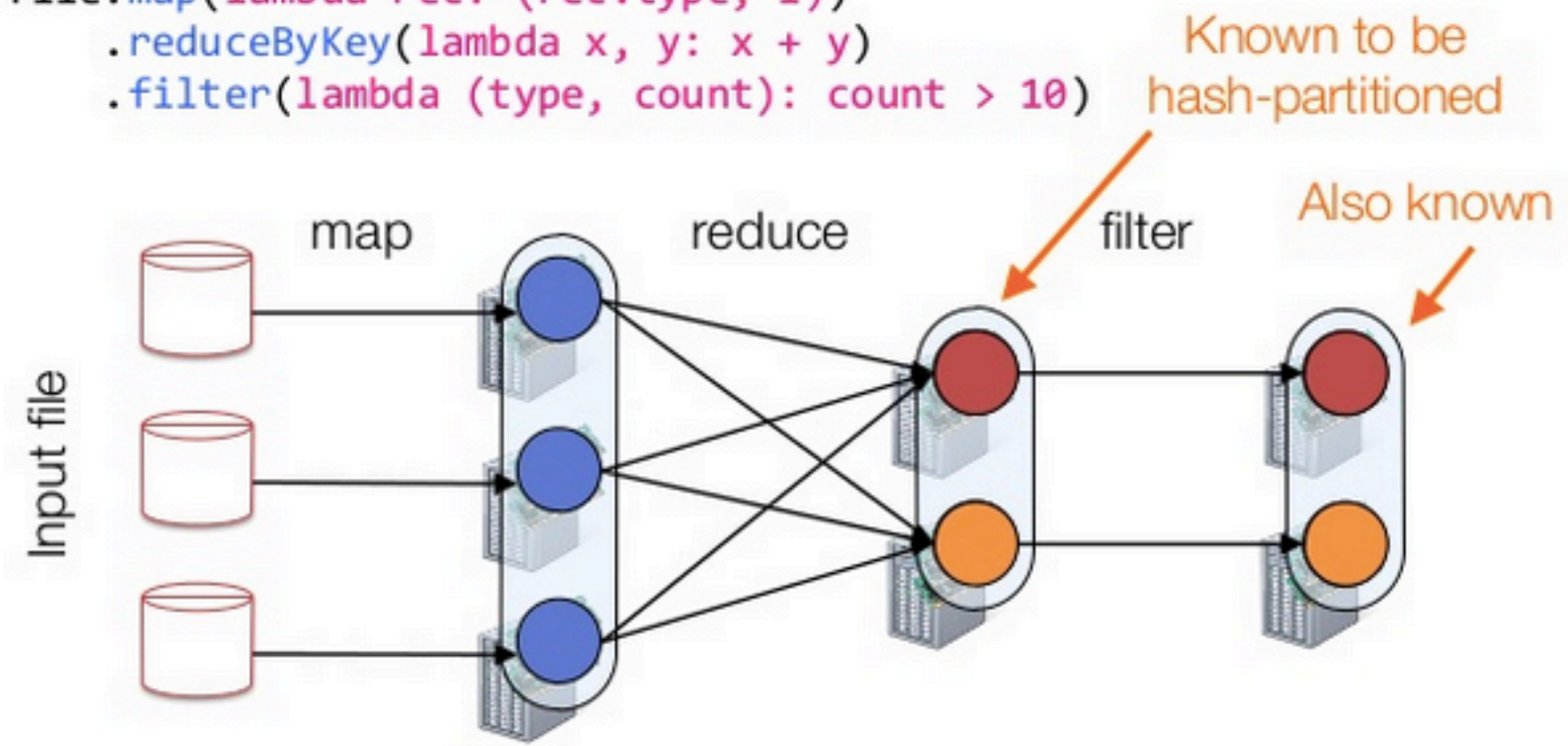
```
file.map(lambda rec: (rec.type, 1))
    .reduceByKey(lambda x, y: x + y)
    .filter(lambda (type, count): count > 10)
```



RDDs know their partioning function

- ..

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



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

bin/pyspark: Spark Shell

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
```

help(pyspark)

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
```

Connect to a cluster versus a local

Or, to use four cores on the local machine:

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

Connect to a 4 cores on the local machine

Spark in one-page!

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

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
map(*func*)	Return a new distributed dataset formed by passing each element of the source through a function *func*.
filter(*func*)	Return a new dataset formed by selecting those elements of the source on which *func* returns true.
flatMap(*func*)	Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item).
mapPartitions(*func*)	Similar to map, but runs separately on each partition (block) of the RDD, so *func* must be of type Iterator<T> => Iterator<U> when running on an RDD of type T.
mapPartitionsWithIndex(*func*)	Similar to mapPartitions, but also provides *func* with an integer value representing the index of the partition, so *func* must be of type (Int, Iterator<T>) => Iterator<U> when running on an RDD of type T.
sample(*withReplacement*, *fraction*, *seed*)	Sample a fraction *fraction* of the data, with or without replacement, using a given random number generator seed.
union(*otherDataset*)	Return a new dataset that contains the union of the elements in the source dataset and the argument.
intersection(*otherDataset*)	Return a new RDD that contains the intersection of elements in the source dataset and the argument.
distinct([*numTasks*]))	Return a new dataset that contains the distinct elements of the source dataset.
groupByKey([*numTasks*])	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 `reduceByKey` or `combineByKey` 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.
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 *func*, which must be of type (V,V) => V. Like in `groupByKey`, the number of reduce tasks is configurable through an optional second argument.
aggregateByKey(*zeroValue*)(*seqOp*, *combOp*, [*numTasks*])	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.
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

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
reduce(*func*)	Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function should 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. This is 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 take(1)).
take(*n*)	Return an array with the first *n* elements of the dataset. Note that this is currently not executed in parallel. Instead, the driver program computes all the elements.
takeSample(*withReplacement*, *num*, [*seed*])	Return an array with a random sample of *num* elements of the dataset, with or without replacement, optionally pre-specifying a random number generator seed.
takeOrdered(*n*, *[ordering]*)	Return the first *n* elements of the RDD using either their natural order or a custom comparator.
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 toString on each element to convert it to a line of text in the file.
saveAsSequenceFile(*path*) (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).
saveAsObjectFile(*path*) (Java and Scala)	Write the elements of the dataset in a simple format using Java serialization, which can then be loaded using `SparkContext.objectFile()`.
countByKey()	Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.
foreach(*func*)	Run a function *func* 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.

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].

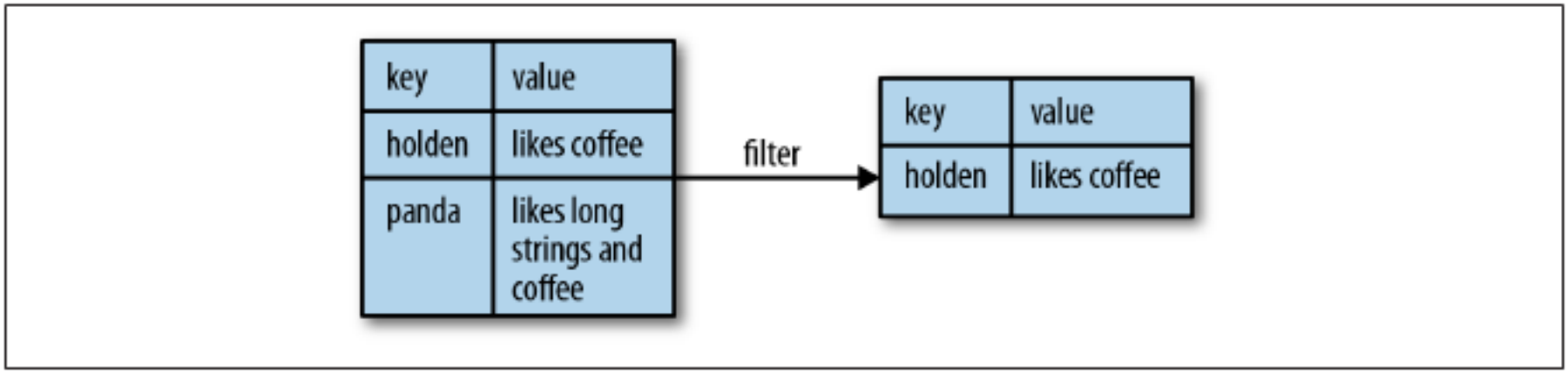


Figure 4-1. Filter on value

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}
```

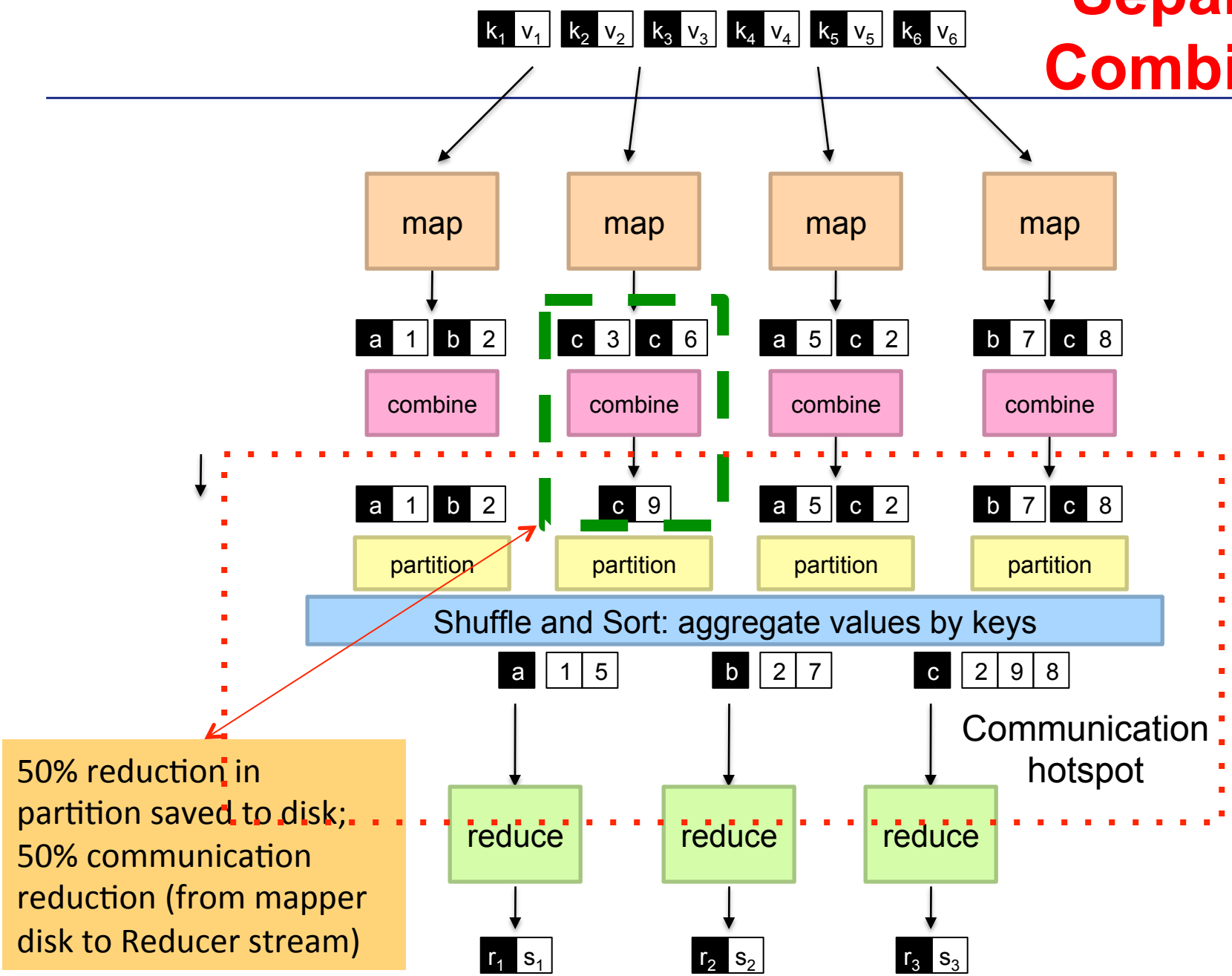
Combiners

Combiner Design

- **Combiners and reducers share same method signature**
  - Sometimes, reducers can serve as combiners
  - Often, not…
- **Remember: combiner are optional optimizations**
  - Should not affect algorithm correctness
  - Hadoop makes no guarantees: could be run 0, 1, or multiple times
- **Combiners need to be associative and commutative**

- **Example: find average of all integers associated with the same key**

Separate Combiner



Reduce, ReduceByKey

reduce(*f*)

Reduces the elements of this RDD using the specified commutative and associative binary operator. Currently reduces partitions locally.

```
>>> from operator import add
>>> sc.parallelize([1, 2, 3, 4, 5]).reduce(add)
15
>>> sc.parallelize((2 for _ in range(10))).map(lambda x: 1).cache().reduce(add)
10
>>> sc.parallelize([]).reduce(add)
Traceback (most recent call last):
    ...
ValueError: Can not reduce() empty RDD
```

Reduces the elements of this RDD using the specified commutative and associative binary operator. Currently reduces partitions locally.

reduceByKey(*func*, *numPartitions=None*, *partitionFunc=<function portable_hash at 0x7f1ac7340578>*)

Merge the values for each key using an associative reduce function.

This will also perform the merging locally on each mapper before sending results to a reducer, similarly to a "combiner" in MapReduce.

Output will be partitioned with **numPartitions** partitions, or the default parallelism level if **numPartitions** is not specified. Default partitioner is hash-partition.

```
>>> from operator import add
>>> rdd = sc.parallelize([("a", 1), ("b", 1), ("a", 1)])
>>> sorted(rdd.reduceByKey(add).collect())
[('a', 2), ('b', 1)]
```

combineByKey

- combineByKey is designed for when your return type from the aggregation is different from the values being aggregated (e.g. you group together objects)

GroupByKey is expensive (vs mapside ops vs combiners)

- When working in a map-reduce framework such Spark or Hadoop one of the steps we can take to ensure maximum performance is to limit the amount of data sent accross the network during the shuffle phase.
- The best option is when all operations can be performed on the map-side exclusively, meaning no data is sent at all to reducers.
- In most cases though, it's not going to be realistic to do map-side operations only. If you need to do any sort of grouping, sorting or aggregation you'll need to send data to reducers. But that doesn't mean we still can't attempt to make some optimizations.

groupByKey is expensive

- In the case of a groupByKey call, every single key-value pair will be shuffled accross the network with identical keys landing on the same reducer.
- To state the obvious, when grouping by key, the need for all matching keys to end up on the same reducer can't be avoided. But one optimization we can attempt is to combine/merge values so we end up sending fewer key-value pairs in total.
- Additionally, less key-value pairs means reducers won't have as much work to do, leading to additional performance gains.
- The groupByKey call makes no attempt at merging/combining values, so it's an expensive operation.

USING THE REDUCEBYKEY TRANSFORMATION

`reduceByKey` lets you merge all the values of a key into a single value of the same type. The merge function you pass it merges two values at a time until there is only one value left. The function should be associative; otherwise you won't get the same result every time you perform `reduceByKey` transformation on the same RDD.

You could use `reduceByKey` for your final task - finding the customer who spent the most overall - but you can also use the very similar `foldByKey` transformation.

USING THE FOLDBYKEY TRANSFORMATION

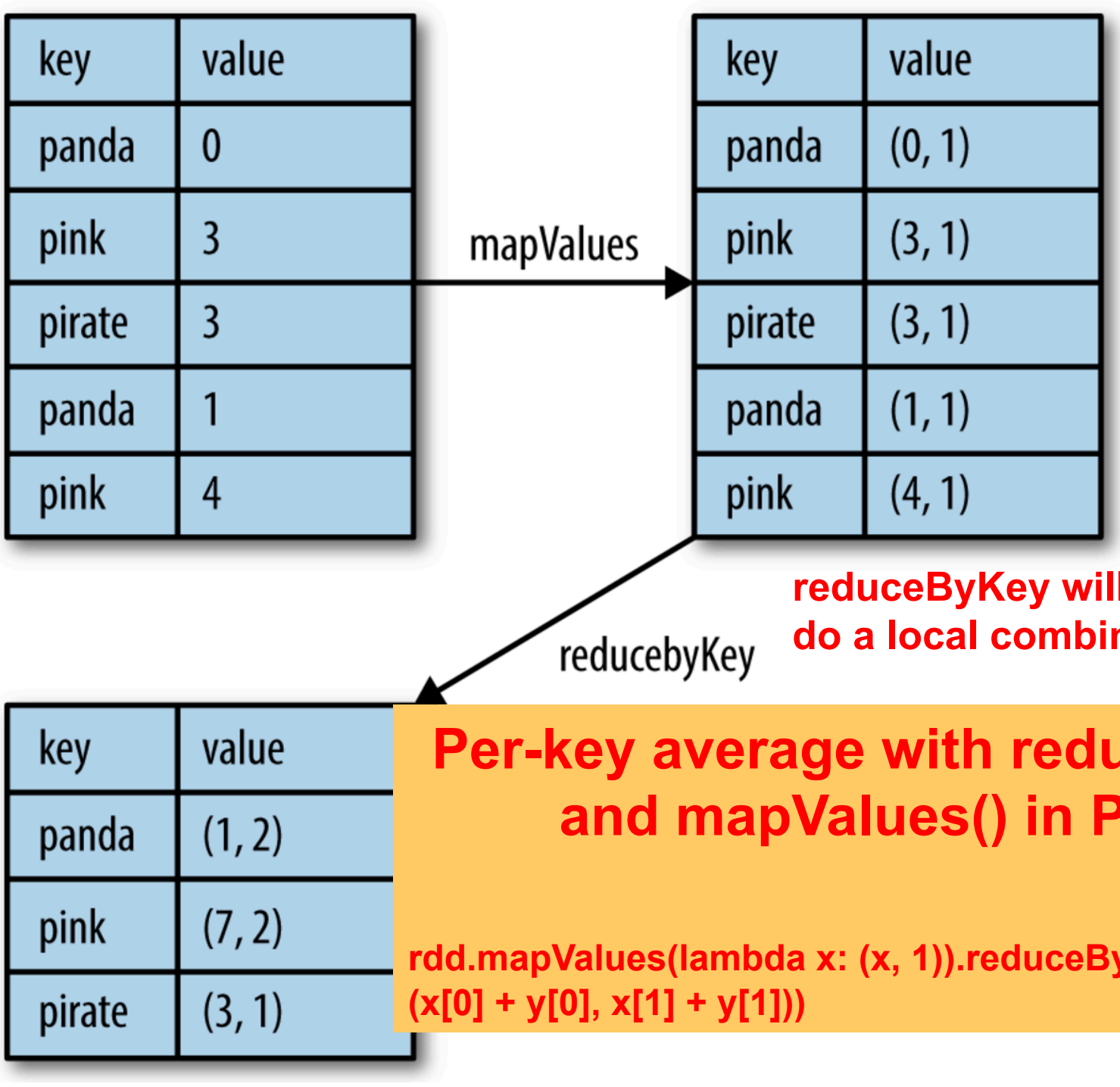
`foldByKey` does the same thing as `reduceByKey`, except that it requires an additional parameter, `zeroValue`, in an extra parameter list that comes before the one with the reduce function. The complete method signature is as follows:

```
foldByKey(zeroValue: V)(func: (V, V) => V): RDD[(K, V)]
```

`zeroValue` should be a neutral value (`0` for addition, `1` for multiplication, `Nil` for lists, and so forth). It's applied on the first value of a key (using the input function), and the result is applied on the second value. You should be careful here because, unlike in `foldLeft` and `foldRight` methods in Scala, `zeroValue` might be applied multiple times. This happens because of RDD's parallel nature.

reduceByKey : returns an pair RDD (key, value)

- 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.



reduceByKey will automatically do a local combine

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 4-7. 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))
```

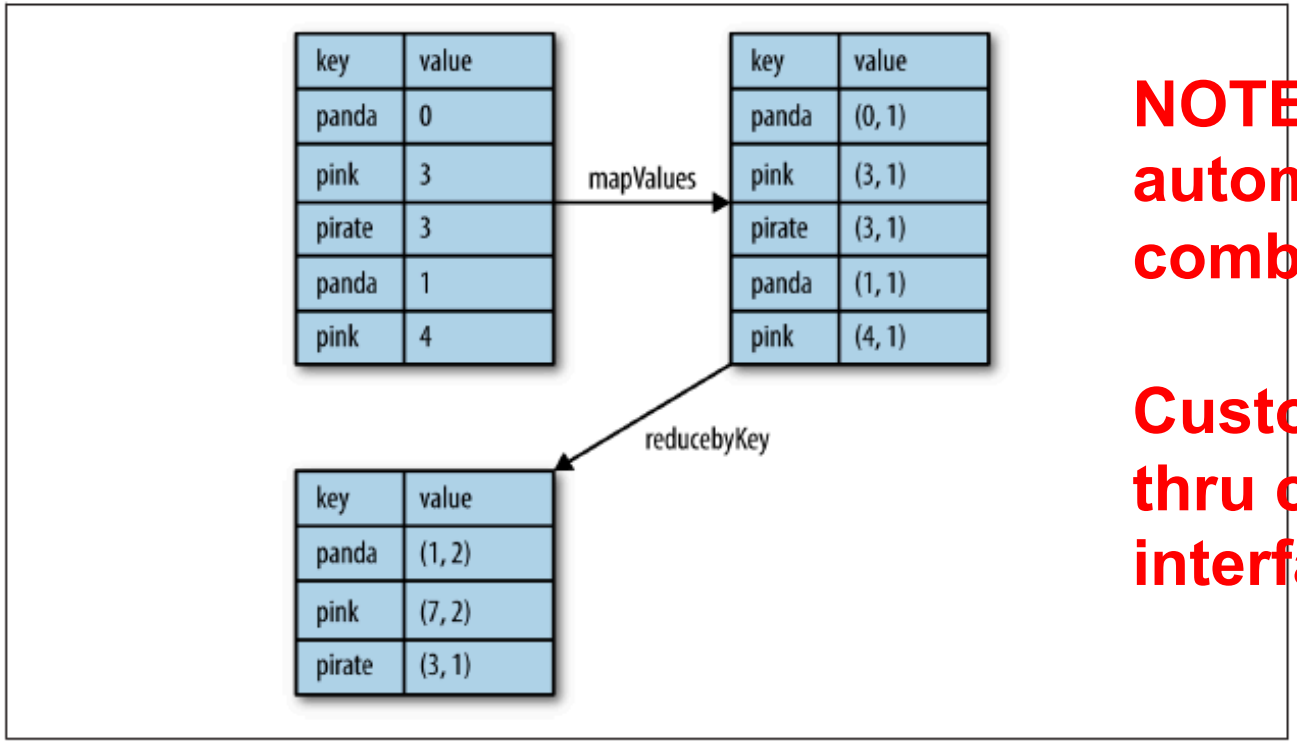


Figure 4-2. Per-key average data flow

NOTE: reduceByKey will automatically do a local combine

Customize the combiner thru combineByKey interface

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.

Live Session Outline

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
- **Week**
  - Homework HW7, HW9, HW10
  - AWS: HW10 local
  - Running an MRJob on cloud with the latest version of MRJob, Screen
  - Async lecture recap plus Q&A Spark
  - Install Spark
  - Spark, RDDs, Patterns in Spark
  - DataFrames, Spark SQL
  - Secondary Sorts in Spark
  - MLLib: brief intro
  - Homegrown Kmeans in Spark
- **Wrapup**
  - Finish RECORDING (bonus points for reminding me!)

- **Word Count Example**

Example 3

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:

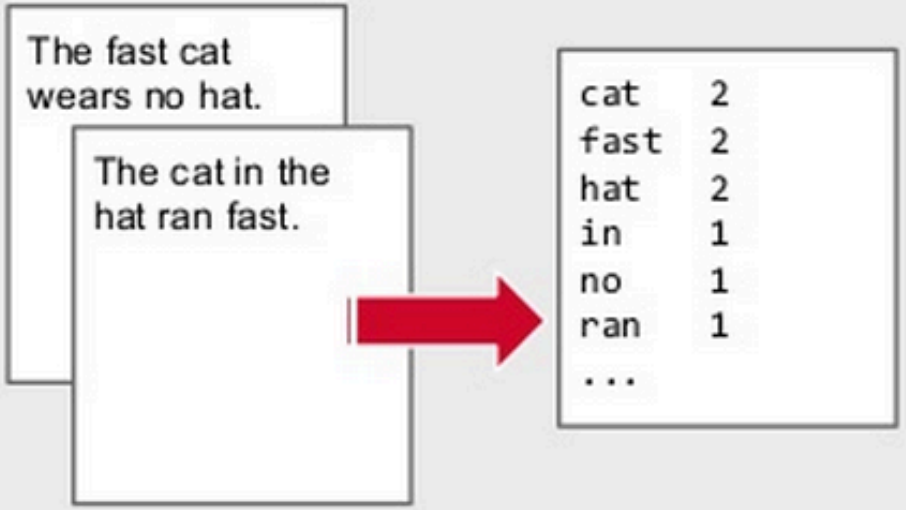
- requires a minimal amount of code
- demonstrates use of both symbolic and numeric values
- isn't many steps away from search indexing
- serves as a "Hello World" for Big Data apps

```
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));
```

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

Word Count in Map-Reduce

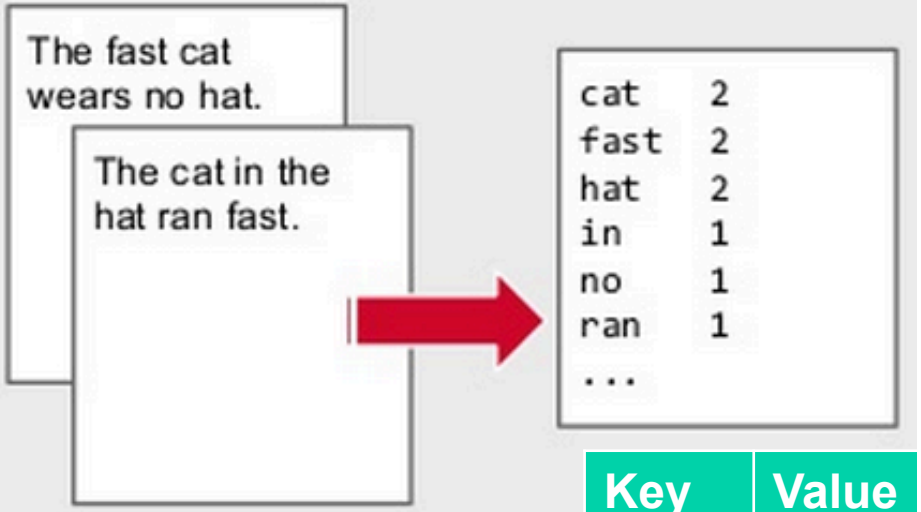
```
def map(key, value):
    for word in value.split():
        emit(word, 1)

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.

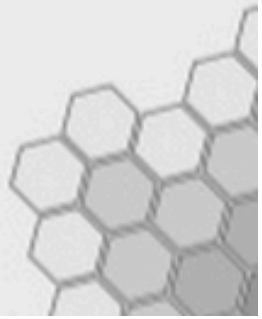
What is the data flow for word count?

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
```

Key	Value
the	1
fast	1
wears	1
cat	1
….	…



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
```

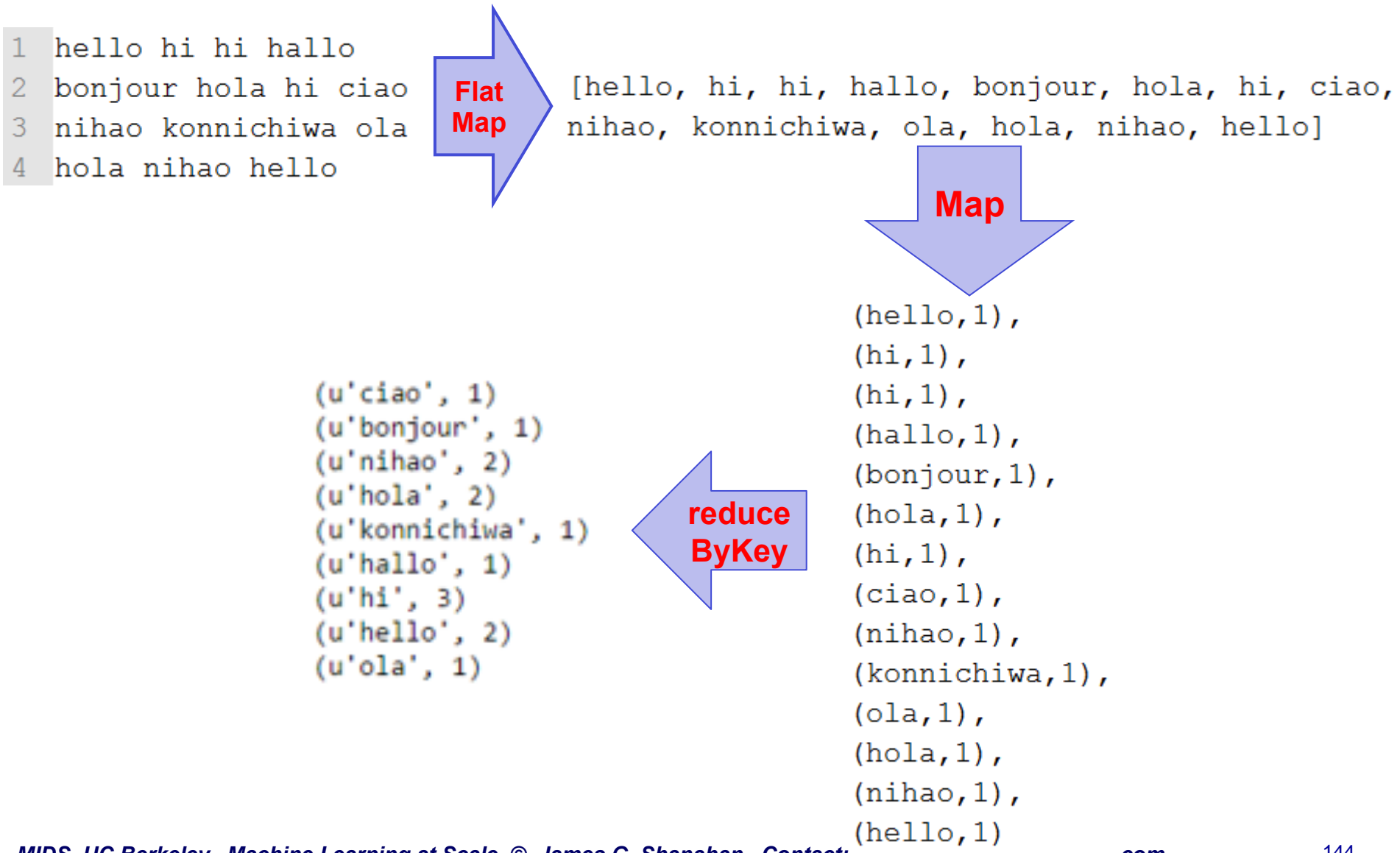
Word Count

```
#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)
```

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

Word Count broken down



```
]: #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')]
```

```
]:
```

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
```

Joins

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)})

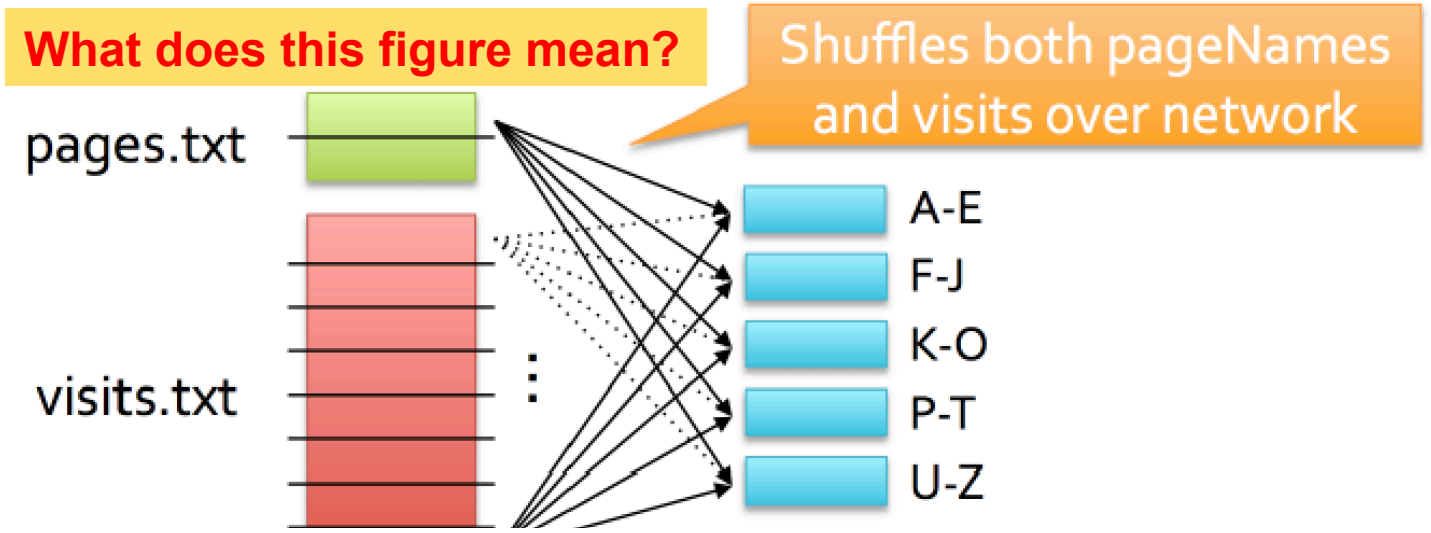
Function name	Purpose	Example	Result
subtractByKey	Remove elements with a key present in the other RDD.	rdd.subtractByKey(other)	{(1, 2)}
		rdd = {(1, 2), (3, 4), (3, 6)} other = {(3, 9)})	
join	Perform an inner join between two RDDs.	rdd.join(other)	{(3, (4, 9)), (3, (6, 9))}

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)
```



- **Broadcast Pages RDD (Smaller RDD) to worker nodes**

Broadcast Variables

Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks. They can be used, for example, to give every node a copy of a large input dataset in an efficient manner. Spark also attempts to distribute broadcast variables using efficient broadcast algorithms to reduce communication cost.

Spark actions are executed through a set of stages, separated by distributed "shuffle" operations. Spark automatically broadcasts the common data needed by tasks within each stage. The data broadcasted this way is cached in serialized form and deserialized before running each task. This means that explicitly creating broadcast variables is only useful when tasks across multiple stages need the same data or when caching the data in deserialized form is important.

Broadcast variables are created from a variable `v` by calling `SparkContext.broadcast(v)`. The broadcast variable is a wrapper around `v`, and its value can be accessed by calling the `value` method. The code below shows this:

Scala **Java** **Python**

```
>>> broadcastVar = sc.broadcast([1, 2, 3])
<pyspark.broadcast.Broadcast object at 0x102789f10>

>>> broadcastVar.value
[1, 2, 3]
```

After the broadcast variable is created, it should be used instead of the value `v` in any functions run on the cluster so that `v` is not shipped to the nodes more than once. In addition, the object `v` should not be modified after it is broadcast in order to ensure that all nodes get the same value of the broadcast variable (e.g. if the variable is shipped to a new node later).

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.	`rdd.subtractByKey(other)`	`{(1, 2)}`
`join`	Perform an inner join between two RDDs.	`rdd.join(other)`	`{(3, (4, 9)), (3, (6, 9))}`
`rightOuterJoin`	Perform a join between two RDDs where the key must be present in the first RDD.	`rdd.rightOuterJoin(other)`	`{(3,(Some(4),9)), (3,(Some(6),9))}`
`leftOuterJoin`	Perform a join between two RDDs where the key must be present in the other RDD.	`rdd.leftOuterJoin(other)`	`{(1,(2,None)), (3, (4,Some(9))), (3, (6,Some(9)))}`
`cogroup`	Group data from both RDDs sharing the same key.	`rdd.cogroup(other)`	`{(1,([2],[])), (3, ([4, 6],[9]))}`

Quiz 10.8.1 Paired RDDs

Given the following paired RDDs
RDD1 = {(1, 2), (3, 4), (3, 6)}
RDD2 = {(3, 9) (3, 6)}

Using PySpark, write code to perform an inner join of these paired RDDs. What is the resulting RDD? Make your PySpark notebook available also (using via an NBViewer link):

A: [(3, (4, 9)), (3, (6, 9))]
B: [(3, (4, 9)), (3, (4, 6)), (3, (6, 9)), (3, (6, 6))]
C: [(3, (4, 9)), (3, (4, 6)), (3, (6, 9)), (3, (6, 9))]
D: None of the above

Quiz 10.8.1 Notebook

```
import os
import sys #current as of 9/26/2015
spark_home = os.environ['SPARK_HOME'] =
   '/Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/spark-1.5.0-bin-hadoop2.6/'

if not spark_home:
    raise ValueError('SPARK_HOME enviroment 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 1.5.0
      /_/

Using Python version 2.7.9 (v2.7.9:648dcafa7e5f, Dec 10 2014 10:10:46)
SparkContext available as sc, HiveContext available as sqlContext.
```

```
rdd1 = sc.parallelize({(1, 2), (3, 4), (3, 6)})
rdd2 = sc.parallelize({(3, 9), (3, 6)})
joinedRdd=rdd1.join(rdd2)
#RDD1 = {(1, 2), (3, 4), (3, 6)}
#RDD2 = {(3, 9) (3, 6)}
joinedRdd.collect()
```

```
[(3, (4, 9)), (3, (4, 6)), (3, (6, 9)), (3, (6, 6))]
```

FlatMapValues

Ecommerce transaction DB

MapValues

USING THE MAPVALUES TRANSFORMATION

A second email from Cowboy demands your attention: "*Wilbur, one more thing: they also want to give a 5% discount for 2 or more Barbie Shopping Mall Playsets bought.*"

"*Marketing guys are out of control today,*" you think to yourself, but reply with a simple "*No problem, boss.*"

`mapValues` transformation helps you do this: it changes the values contained in a pair RDD, without changing the associated keys. And that's exactly what you need. Barbie Shopping Mall Playset has the id 25, so you apply the discount like this:

```
scala> transByCust = transByCust.mapValues(t => {
     if(t(3).toInt == 25 && t(4).toDouble > 1)
         t(5) = (t(5).toDouble * 0.95).toString
     t })
```

The function you gave to `mapValues` transformation checks if the product id (3rd element of the transaction array) is 25 and the quantity (4th element) is greater than 2, and in that case, decreases the total price (6th element) by 5%. Otherwise, it leaves the transaction array untouched.

You assign the new RDD to the same variable `transByCust`, just to make things simpler.

FlatMap

flatMap(*f*, *preservesPartitioning=False*)

Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results.

```
>>> rdd = sc.parallelize([2, 3, 4])
>>> sorted(rdd.flatMap(lambda x: range(1, x)).collect())
[1, 1, 1, 2, 2, 3]
>>> sorted(rdd.flatMap(lambda x: [(x, x), (x, x)]).collect())
[(2, 2), (2, 2), (3, 3), (3, 3), (4, 4), (4, 4)]
```

flatMapValues(*f*)

Pass each value in the key-value pair RDD through a flatMap function without changing the keys; this also retains the original RDD's partitioning.

```
>>> x = sc.parallelize([("a", ["x", "y", "z"]), ("b", ["p", "r"])])
>>> def f(x): return x
>>> x.flatMapValues(f).collect()
[('a', 'x'), ('a', 'y'), ('a', 'z'), ('b', 'p'), ('b', 'r')]
```

FlatMap

USING THE FLATMAPVALUES TRANSFORMATION

You still have two more tasks to do. You decide to tackle the dictionary one first: you need to add a complimentary toothbrush (id 70) to customers who bought 5 or more dictionaries (id 81). That means you need to add additional transactions (represented as arrays) to the `transByCust` RDD.

FlatMap: emit many to one records

If transaction value > $100 then Free widget

Iterate over each record, flatmapping (emit many records to one record)

`flatMapValues` transformation fits the bill because it enables you to change the number of elements corresponding to a key by mapping each value to *one or more* new values. The signature of the transformation function it expects is `V => TraversableOnce[U]` (we said in chapter 2 that `TraversableOnce` is just a special name for a collection). From each of the values in the return collection a new key-value pair is created, for the corresponding key. If the transformation function returns an empty list for one of the values, the resulting Pair RDD will have one element less. If the transformation function returns a list with two elements for one of the values, the resulting Pair RDD will have one element more. Note that the mapped values can be of different type than before.

Solution

So this is what you do:

```
scala> transByCust = transByCust.flatMapValues(tran => {
    if(tran(3).toInt == 81 && tran(4).toInt > 4) { #1
       val cloned = tran.clone() #2
       cloned(5) = "0.00"; cloned(3) = "70"; cloned(4) = "1"; #3
       List(tran, cloned) #4
    }
    else
       List(tran) #5
    })
```

#1 Filter by product and quantity
#2 Clone the transaction array
#3 Set clone's price to 0.00, product id to 70 and quantity to 1
#4 Return two elements
#5 Return one element

The anonymous function given to the transformation maps each transaction into a list. The list contains only the original transaction if the condition is not met, or an additional transaction with a complimentary toothbrush if the condition holds true.

The final `transByCust` RDD now contains 1,006 elements (because there are 6 transactions with 5 or more dictionaries in the file).

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.partitioner`.

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()
```

combineByKey sample data flow

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] = merge Value(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])

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 reparti tion() 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 reparti
tion() 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!

Partitions

The number of RDD partitions is important because, besides influencing data distribution throughout the cluster, it also directly determines the number of tasks that will be running RDD transformations. If this number is too small, the cluster will be underutilized. Furthermore, memory problems could result, because working sets might get too big to fit into memory of executors. We recommend using three to four times more partitions than there are cores in your cluster. Moderately larger values shouldn't pose a problem so feel free to experiment. But don't get too crazy because management of a large number of tasks could create a bottleneck.

Let's see how is data partitioning achieved in Spark.

We recommend using three to four times more partitions than there are cores in your cluster

4.2.1 Using data partitioners

Partitioning of RDDs is performed by `Partitioner` objects that assign a partition index to each RDD element. Two implementations are provided by Spark: `HashPartitioner` and `RangePartitioner`. Pair RDDs also accept custom partitioners.

HASH PARTITIONER

`HashPartitioner` is the default partitioner in Spark. It calculates a partition index based on an element's Java hash code (or a key's hash code in pair RDDs), according to this simple formula `partitionIndex = hashCode % numberOfPartitions`. The partition index is determined quasi-randomly; consequently, the partitions most likely won't be exactly the same size. In large datasets with relatively small number of partitions, though, the algorithm is likely to distribute data evenly among them.

The default number of data partitions when using `HashPartitioner` is determined by the Spark configuration parameter `spark.default.parallelism`. If that parameter isn't specified by the user, it will be set to the number of cores in the cluster. (Chapter 12 covers setting Spark configuration parameters.)

RANGE PARTITIONER

- RangePartitioner partitions data of sorted RDDs into roughly equal ranges. It samples the contents of the RDD passed to it and determines the range boundaries according to the sampled data.
- You aren't likely to be using RangePartitioner directly.

PAIR RDD CUSTOM PARTITIONERS

Pair RDDs can be partitioned with *custom partitioners* when it's important to be precise about the placement of data among partitions (and among tasks working on them). You might want to use a custom partitioner, for example, if it's important that each task processes only a specific subset of key-value pairs, where all of them belong to a single database, single database table, single user, or something similar.

Custom partitioners can be used only on pair RDDs, by passing them to pair RDD transformations. Most pair RDD transformations have two additional overloaded methods: one that takes an additional `Int` argument (the desired number of partitions), and another one that takes an additional argument of the (custom) `Partitioner` type. The method that takes the number of partitions uses the default `HashPartitioner`. For example, these two lines of code are equal, as they both apply `HashPartitioner` with 100 partitions:

```
rdd.foldByKey(afunction, 100)
rdd.foldByKey(afunction, new HashPartitioner(100))
```

Accumulators are variables that are only "added" to through an associative operation and can therefore be efficiently supported in parallel. They can be used to implement counters (as in MapReduce) or sums. Spark natively supports accumulators of numeric types, and programmers can add support for new types. If accumulators are created with a name, they will be displayed in Spark's UI. This can be useful for understanding the progress of running stages (NOTE: this is not yet supported in Python).

An accumulator is created from an initial value `v` by calling `SparkContext.accumulator(v)`. Tasks running on the cluster can then add to it using the `add` method or the `+=` operator (in Scala and Python). However, they cannot read its value. Only the driver program can read the accumulator's value, using its `value` method.

The code below shows an accumulator being used to add up the elements of an array:

Scala **Java** **Python**

```
>>> accum = sc.accumulator(0)
Accumulator<id=0, value=0>

>>> sc.parallelize([1, 2, 3, 4]).foreach(lambda x: accum.add(x))
...
10/09/29 18:41:08 INFO SparkContext: Tasks finished in 0.317106 s

scala> accum.value
10
```

Live Session Outline

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
- **Week**
  - Homework HW7, HW9, HW10
  - AWS: HW10 local
  - Running an MRJob on cloud with the latest version of MRJob, Screen
  - Async lecture recap plus Q&A Spark
  - Install Spark
  - Spark, RDDs, Patterns in Spark
  - DataFrames, Spark SQL
  - Secondary Sorts in Spark
  - MLLib: brief intro
  - Homegrown Kmeans in Spark
- **Wrapup**
  - Finish RECORDING (bonus points for reminding me!)

DataFrames

- As you saw there, DataFrames lets you work with structured data (data organized in rows and columns, where each column contains only values of a certain type). SQL (Structured Query Language), frequently used in relational databases, is the most common way to organize and query this data. SQL also figures as part of the name of the first Spark component we're covering in this book: Spark SQL. As the main component of Spark SQL, DataFrames lets you use SQL (and its HQL variant, but we will get to that later in this chapter) to query data in Spark.

- convert RDDs to DataFrames

DataFrames (new API introduced in 2/2015)

- **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, or a table in MySql, 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
  - JSON, Parquet and more
- **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
```

SQL

- ..

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
```

Create an RDD

```
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'Justi
```

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

Transform RDD to a dataframe

```
DataFrame[age: bigint, name: string]
```

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

DataFrames

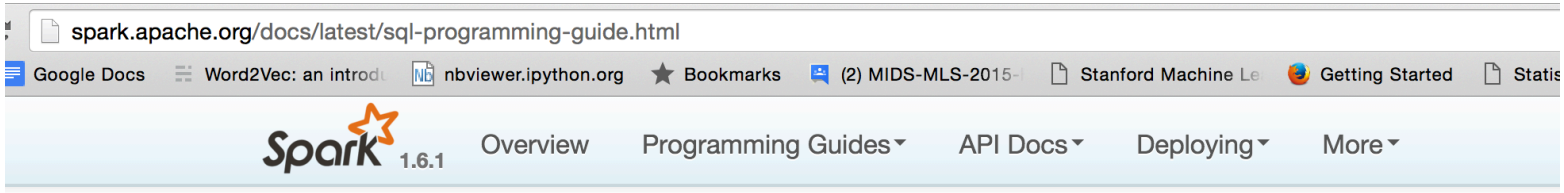
- **Register the DataFrame as a table and then run SQL queries**

```
# 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
```

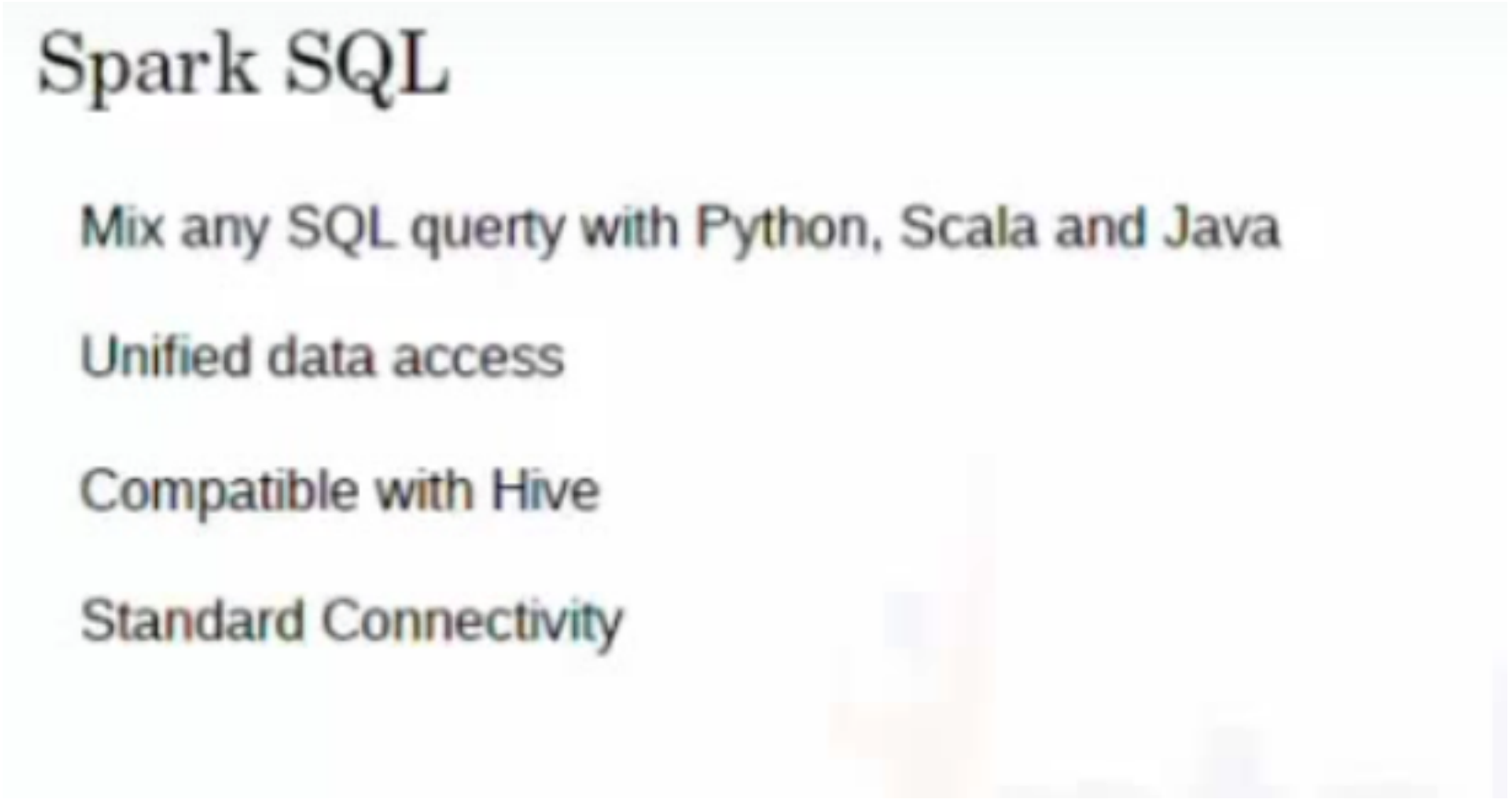
Spark SQL

- **Spark SQL is a Spark module for structured data processing.**
  - Unlike the basic Spark RDD API, the interfaces provided by Spark SQL provide Spark with more information about the structure of both the data and the computation being performed.
- **Internally, Spark SQL uses this extra information to perform extra optimizations.**
- **There are several ways to interact with Spark SQL including SQL, the DataFrames API and the Datasets API.**
  - When computing a result the same execution engine is used, independent of which API/language you are using to express the computation.
  - This unification means that developers can easily switch back and forth between the various APIs based on which provides the most natural way to express a given transformation.
- **All of the examples on this page use sample data included in the Spark distribution and can be run in the spark-shell, pyspark shell, or sparkR shell.**
  - http://spark.apache.org/docs/latest/sql-programming-guide.html



Spark SQL, DataFrames and Datasets Guide

Spark SQL


Spark SQL
Mix any SQL querty with Python, Scala and Java
Unified data access
Compatible with Hive
Standard Connectivity

Spark SQL

- Native SQL Interface to Spark
- Hive, DataFrame, JSON, RDD, etc...
- JDBC Server (B.I. Interface)
- UDFs (Spark SQL and Hive)
- Columnar storage, Predicate Pushdowns, Tuning options

Spark SQL Example

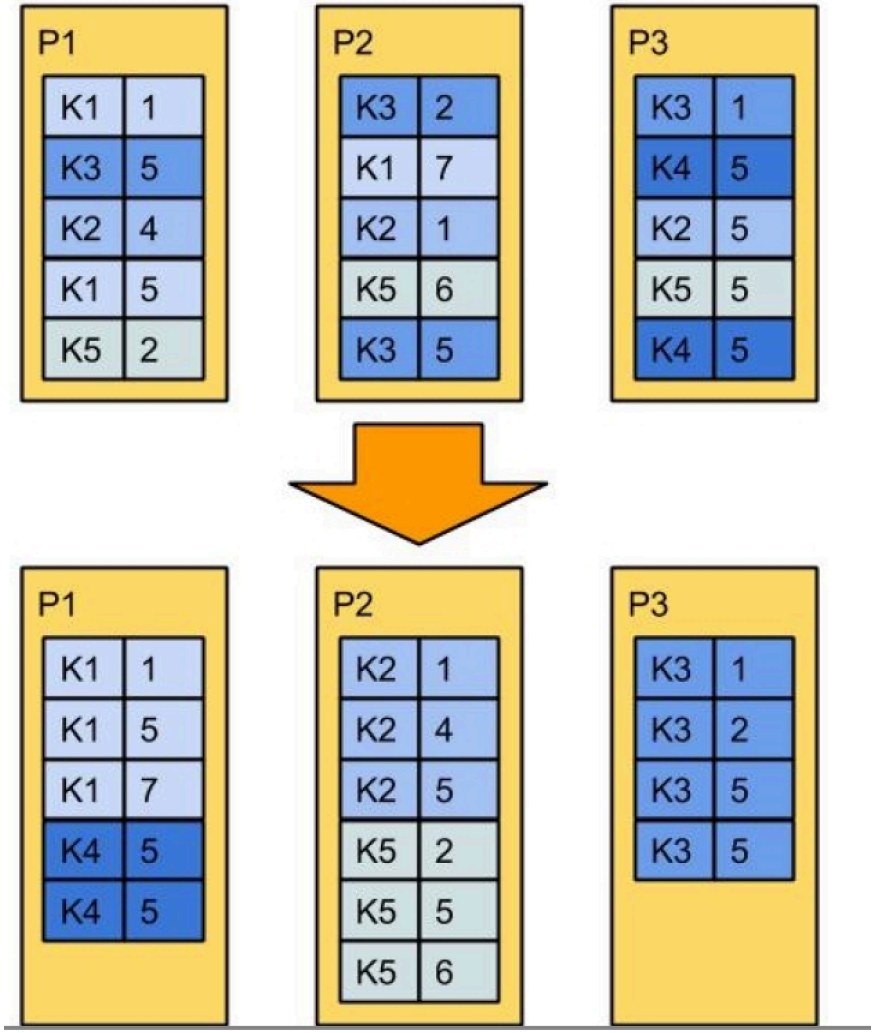
```
hiveCtx = HiveContext(sc)
allData = hiveCtx.jsonFile(filein)
allData.registerTempTable("customers")

query1 = hiveCtx.sql("
                SELECT last, first
                FROM customers
                ORDER BY last
                LIMIT 50")
```

Live Session Outline

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
- **Week**
  - Homework HW7, HW9, HW10
  - AWS: HW10 local
  - Running an MRJob on cloud with the latest version of MRJob, Screen
  - Async lecture recap plus Q&A Spark
  - Install Spark
  - Spark, RDDs, Patterns in Spark
  - DataFrames, Spark SQL
  - Secondary Sorts in Spark
  - MLLib: brief intro
  - Homegrown Kmeans in Spark
- **Wrapup**
  - Finish RECORDING (bonus points for reminding me!)

Secondary Sort in Spark

- **Secondary Sort**



From Spark in Action (from Manning Publishers), Chapter 4

PERFORMING A SECONDARY SORT

A few more things about sorting are worth mentioning. Sometimes you might also want to sort values within keys. For example, you could group transactions by customer id and wonder how to further sort them by transaction time.

There's a relatively new `groupByKeyAndSortValues` transformation, which enables you to do exactly this. For an RDD of `(K, V)` pairs, it expects an implicit `Ordering[V]` object to be present in the scope and a single argument: either a `Partitioner` object or a number of partitions.

It will give you an RDD of `(K, Iterable(V))` elements with values sorted according to the implicit `Ordering` object. But it will first group values by key, which can be expensive on memory and network.

A cheaper method for performing a secondary sort, without grouping, is this:[66]

1. Map an `RDD[(K, V)]` to `RDD[((K, V),null)]`. For example: `rdd.map(kv =>(kv, null))`
2. Use a custom partitioner that will partition only on the K part of the new composite key so that all elements with the same K part end up in the same partition.
3. Call `repartitionAndSortWithinPartition`, which has to sort by the complete composite key `(K, V)` — first by keys, and then by values.
4. Map the RDD back to `RDD[(K, V)]`.

From Spark in Action (from Manning Publishers), Chapter 4

Secondary Sort in Spark (Scala only)

k1,k2,v1,v2 : partition by k1 and sort by k2

https://www.dropbox.com/s/rknn7f5cs0lugrq/Secondary_Sort.ipynb?dl=0

In [2]:
```
%%writefile ss.txt
1,3,a,b
2,5,a,c
1,4,a,f
3,4,d,c
2,1,f,a
1,1,e,r
2,4,o,1
3,2,d,c
```

```
Overwriting ss.txt
```

In [3]:
```
def read_data(line):
    d = line.split(',')
    return [int(d[0]),int(d[1])],[d[2],d[3]]
data = sc.textFile("ss.txt").map(read_data)
```

In [4]:
```
data.collect()
```

Out[4]:
```
[([1, 3], [u'a', u'b']),
 ([2, 5], [u'a', u'c']),
 ([1, 4], [u'a', u'f']),
 ([3, 4], [u'd', u'c']),
 ([2, 1], [u'f', u'a']),
 ([1, 1], [u'e', u'r']),
 ([2, 4], [u'o', u'1']),
 ([3, 2], [u'd', u'c'])]
```

In [5]:
```
ssdata = data.repartitionAndSortWithinPartitions(numPartitions=3,partitionFunc= lambda x: x[0]%3,keyfunc=lambda x: x[1])
```

Check ssdata by partition

In [6]:
```
ssdata.glom().collect()
```

Out[6]:
```
[[([3, 2], [u'd', u'c']), ([3, 4], [u'd', u'c'])],
 [([1, 1], [u'e', u'r']), ([1, 3], [u'a', u'b']), ([1, 4], [u'a', u'f'])],
 [([2, 1], [u'f', u'a']), ([2, 4], [u'o', u'1']), ([2, 5], [u'a', u'c'])]]
```

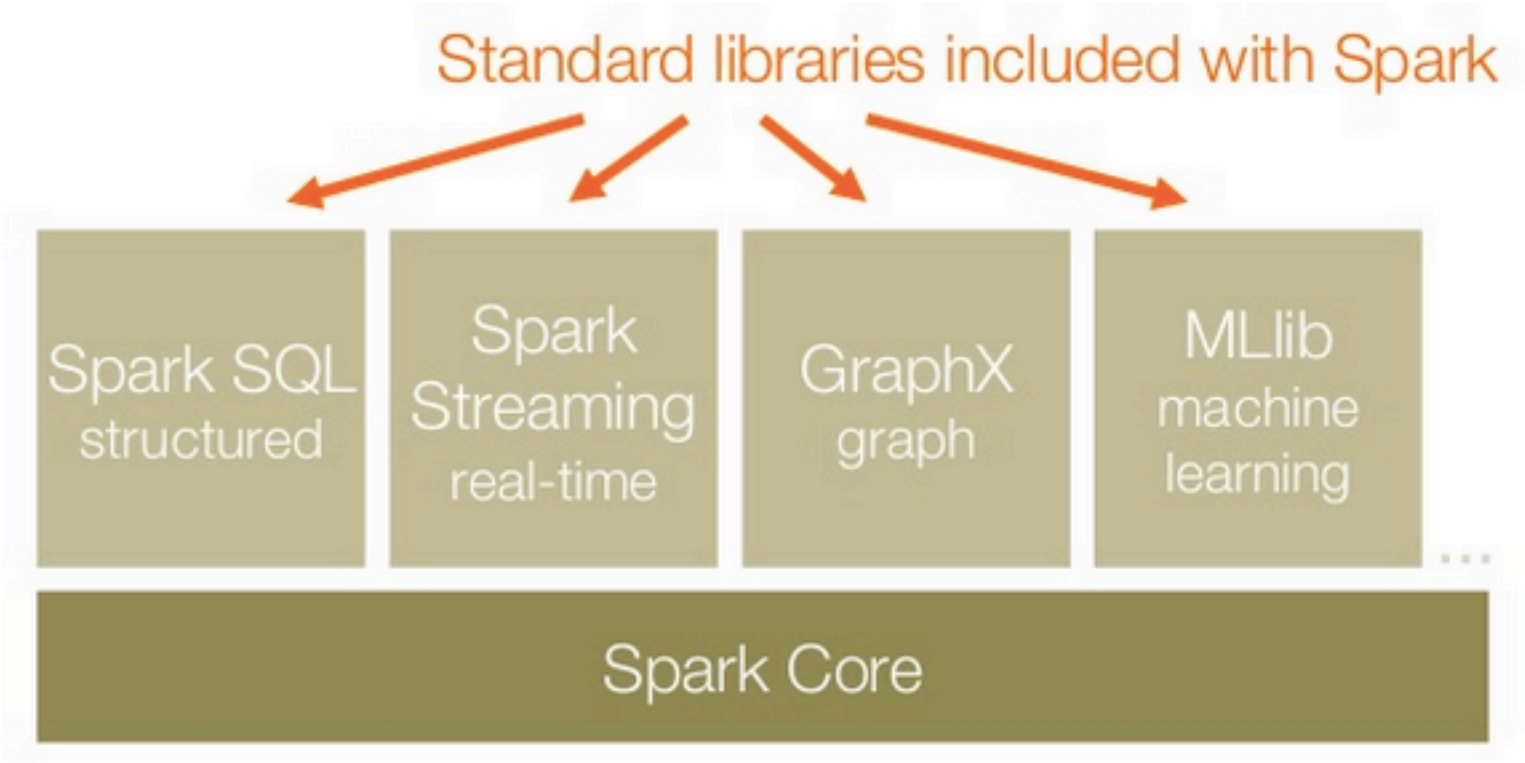
Live Session Outline

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
- **Week**
  - Homework HW7, HW9, HW10
  - AWS: HW10 local
  - Running an MRJob on cloud with the latest version of MRJob, Screen
  - Async lecture recap plus Q&A Spark
  - Install Spark
  - Spark, RDDs, Patterns in Spark
  - DataFrames, Spark SQL
  - Secondary Sorts in Spark
  - MLLib: brief intro
  - Homegrown Kmeans in Spark
- **Wrapup**
  - Finish RECORDING (bonus points for reminding me!)

- **Machine Learning in Spark**
  - Mllib
  - Write your own!

Spark Machine Learning

- ..



Spark MLLib Functionality 1/2

MLlib is Apache Spark's scalable machine learning library

- **Supervised ML**
  - 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**
- **Unsupervised learning**
  - 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**
- **Statistics**
  - Summary statistics, Correlation, Chi-square
- **Feature transformations**

Spark MLLib Functionality 2/2

- **Pipeline API (Train, Evaluation, Testing)**
  - EDA
  - Feature engineering
  - Data engineering
  - Learning
  - Hyperparameter tuning
  - Testing

MLLib

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

Machine Learning Library (MLlib)

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

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

- 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 Tre
  - 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

1.4

data types, algorithms, and utilities

- Data types
- Basic statistics
  - summary statistics
  - correlations
  - stratified sampling
  - hypothesis testing
  - streaming significance 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)
  - bisecting k-means
  - streaming k-means
- Dimensionality reduction
  - singular value decomposition (SVD)
  - principal component analysis (PCA)
- Feature extraction and transformation
- Frequent pattern mining
  - FP-growth
  - association rules
  - PrefixSpan
- Evaluation metrics
- PMML model export
- Optimization (developer)
  - stochastic gradient descent
  - limited-memory BFGS (L-BFGS)

Spark 1.6

MLlib Example

```
# Linear Regression Model Build

model = LinearRegressionWithSGD.train(
            data,
            iterations = 100,
            intercept = True
```

Live Session Outline

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
- **Week**
  - Homework HW7, HW9, HW10
  - AWS: HW10 local
  - Running an MRJob on cloud with the latest version of MRJob, Screen
  - Async lecture recap plus Q&A Spark
  - Install Spark
  - Spark, RDDs, Patterns in Spark
  - DataFrames, Spark SQL
  - Secondary Sorts in Spark
  - MLLib: brief intro
  - Homegrown Kmeans in Spark
- **Wrapup**
  - Finish RECORDING (bonus points for reminding me!)

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

Clustering Algorithms

- **Flat/Non-Hierarchical Clustering**
  - Exclusive Clustering (k-means) [In R, cluster package]
  - Overlapping Clustering (fuzzy c means) [In R, e1071]
  - Probabilistic Clustering (Mixture of Gaussians)

- **Hierarchical Clustering**
  - Agglomerative approach (bottom-up)
    - In R: hclust() and agnes()
  - 2. Divisive approach (top-down)
    - In R: diana()

Kmeans Algorithm

- **k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.**
  - This results in a partitioning of the data space into Voronoi cells.
- **The problem is computationally difficult (NP-hard); however, there are efficient heuristic algorithms that are commonly employed and converge quickly to a local optimum.**
- **These are usually similar to the expectation-maximization algorithm for mixtures of Gaussian distributions via an iterative refinement approach employed by both algorithms.**

K-Means

- **Partition data such that it minimizes the within group sum of squares over all variables**
- **N examples with k clusters**
- **Impractical to explore all possible partitions**

The k-means clustering technique seeks to partition a set of data into a specified number of groups, k, by minimising some numerical criterion, low values of which are considered indicative of a 'good' solution. The most commonly used approach, for example, is to try to find the partition of the n individuals into k groups, which minimises the within-group sum of squares over all variables. The problem then appears relatively simple; namely, consider every possible partition of the n individuals into k groups, and select the one with the lowest within-group sum of squares. Unfortunately, the problem in practise is not so straightforward. The numbers involved are so vast that complete enumeration of *every* possible partition remains impossible even with the fastest computer. The scale of the problem is illustrated by the numbers in Table 18.3.

Table 18.3: Number of possible partitions depending on the sample size n and number of clusters k.

n	k	Number of possible partitions
15	3	2,375,101
20	4	45,232,115,901
25	8	690,223,721,118,368,580
100	5	10^{68}

3^{15} possible partitions

The impracticability of examining every possible partition has led to the development of algorithms designed to search for the minimum values of the clustering criterion by rearranging existing partitions and keeping the new one only if it provides an improvement. Such algorithms do not, of course, guarantee finding the global minimum of the criterion. The essential steps in these algorithms are as follows:

K-Means Clustering Algorithm

- **K-Means Loop**

E
  - The "assignment" step is also referred to as expectation step,

M
  - The "update step" as maximization step, making this algorithm a variant of the *generalized* expectation-maximization algorithm.

- **Until Convergence**

K-means Algorithm

- **For a current set of cluster means, assign each observation as:**

$$C(i) = \arg\min_{1 \le k \le K} \| x_i - m_k \|^2, \; i = 1, \ldots, N$$

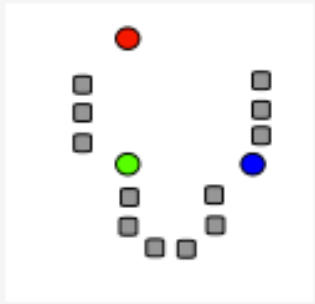
- **For a given cluster assignment *C* of the data points, compute the cluster means m_k:**

$$m_k = \frac{\sum_{i:C(i)=k} x_i}{N_k}, \; k = 1, \ldots, K.$$

- **Iterate above two steps until convergence**

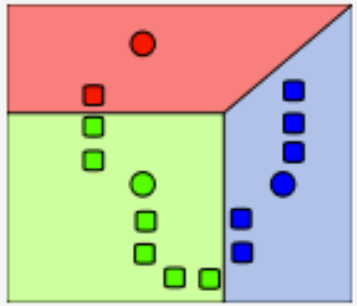
Clustering Algorithms

Initialization



1) *k* initial "means" (in this case *k*=3) are randomly selected from the data set (shown in color).

E-Step "assignment" step

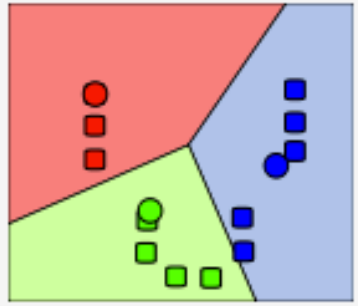


2) *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

M-Step update step

3) The centroid of each of the *k* clusters becomes the new means.

Converged



4) Steps 2 and 3 are repeated until convergence has been reached.

Clustering Algorithms

- **Non-Hierarchical Clustering**
  - Exclusive Clustering (k-means) [In R
  - Overlapping Clustering (fuzzy c mea
  - Probabilistic Clustering (Mixture of G

Question: can both types of algorithm be effectively implemented in Hadoop?

- **Hierarchical Clustering**
  - Agglomerative approach (bottom-up)

Hierarchical Clustering is tough to deploy:
Distributing a bottom-up algorithm is tricky because each distributed process needs the entire dataset to make choices about appropriate clusters. It also needs a list of clusters at its current level so it doesn't add a data point to more than one cluster at the same level.

Kmeans in Spark

- **Notebook**
  - https://www.dropbox.com/s/41q9lgyqhy8ed5g/EM-Kmeans.ipynb?dl=0

- **NBViewer**
  - http://nbviewer.ipython.org/urls/dl.dropbox.com/s/41q9lgyqhy8ed5g/EM-Kmeans.ipynb

Kmeans In Spark:
E-Step: Cluster Assignment Function

```
SparkContext available as sc, HiveContext available as sqlContext.
```

$$C(i) = \arg\min_{1 \le k \le K} \|x_i - m_k\|^2, \; i = 1, \ldots, N$$

```
1  import numpy as np
2
3  #Calculate which class each data point belongs to
4  def nearest_centroid(line):
5      x = np.array([float(f) for f in line.split(',')])
6      closest_centroid_idx = np.sum((x - centroids)**2, axis=1).argmin()
7      return (closest_centroid_idx,(x,1))
8
9  #plot centroids and data points for each iteration
10 def plot_iteration(means):
11     pylab.plot(samples1[:, 0], samples1[:, 1], '.', color = 'blue')
12     pylab.plot(samples2[:, 0], samples2[:, 1], '.', color = 'blue')
13     pylab.plot(samples3[:, 0], samples3[:, 1],'.', color = 'blue')
14     pylab.plot(means[0][0], means[0][1],'*',markersize =10,color = 'red')
15     pylab.plot(means[1][0], means[1][1],'*',markersize =10,color = 'red')
16     pylab.plot(means[2][0], means[2][1],'*',markersize =10,color = 'red')
17     pylab.show()
```

http://nbviewer.ipython.org/urls/dl.dropbox.com/s/41q9lgyqhy8ed5g/EM-Kmeans.ipynb

Kmeans In Spark:

Lines 9-21 are local to driver

Reducer needs to be commutative and associative

```
1  K = 3
2  # Initialization: initialization of parameter is fixed to show an example
3  centroids = np.array([[0.0,0.0],[2.0,2.0],[0.0,7.0]])
4
5  D = sc.textFile("./data.csv").cache()
6  iter_num = 0
7  for i in range(10):
8      res = D.map(nearest_centroid).reduceByKey(lambda x,y : (x[0]+y[0],x[1]+y[1])).collect(
9      #res [(0, (array([  2.66546663e+00,   3.94844436e+03]), 1001)  ),
10     #     (2, (array([ 6023.84995923,  5975.48511018]), 1000)),
11     #     (1, (array([ 3986.85984761,    15.93153464]), 999))]
12     # res[1][1][1] returns 1000 here
13     res = sorted(res,key = lambda x : x[0])  #sort based on clusted ID
14     centroids_new = np.array([x[1][0]/x[1][1] for x in res])  #divide by cluster size
15     if np.sum(np.absolute(centroids_new-centroids))<0.01:
16         break
17     print "Iteration" + str(iter_num)
18     iter_num = iter_num + 1
19     centroids = centroids_new
20     print centroids
21     plot_iteration(centroids)
22 print "Final Results:"
23 print centroids
```

E-Step

M-Step Sum Part 1

$$m_k = \frac{\sum_{i:C(i)=k} x_i}{N_k}, \; k = 1, \ldots, K$$

(Data, count

M-Step Sum Part 2

Kmeans In Spark: Cluster Assignment Function

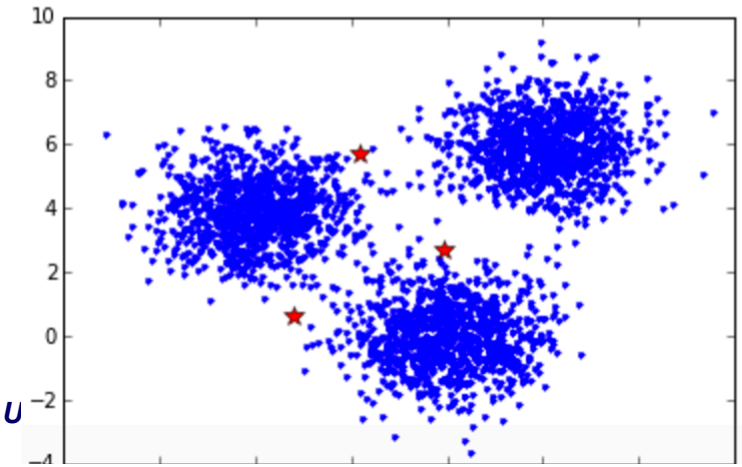
```
Iteration0
[[ 0.80217059  0.61622248]
 [ 3.94191443  2.67285704]
 [ 2.18436067  5.72221018]]
```

```
K = 3
# Initialization: initialization of parameter is fixed to show an example
centroids = np.array([[0.0,0.0],[2.0,2.0],[0.0,7.0]])

D = sc.textFile("./data.csv").cache()
iter_num = 0
for i in range(10):
    res = D.map(nearest_centroid).reduceByKey(lambda x,y : (x[0]+y[0],x[1]+y[1])).collect()
    #res [(0, (array([  2.66546663e+00,   3.94844436e+03]), 1001)  ),
    #     (2, (array([ 6023.84995923,  5975.48511018]), 1000)),
    #     (1, (array([ 3986.85984761,    15.93153464]), 999))]
    # res[1][1][1] returns 1000 here
    res = sorted(res,key = lambda x : x[0])  #sort based on clusted ID
    centroids_new = np.array([x[1][0]/x[1][1] for x in res])  #divide by cluster size
    if np.sum(np.absolute(centroids_new-centroids))<0.01:
        break
    print "Iteration" + str(iter_num)
    iter_num = iter_num + 1
    centroids = centroids_new
    print centroids
    plot_iteration(centroids)
print "Final Results:"
print centroids
```

E-Step

M-Step Sum Part 1

```
Iteration0
[[ 0.80217059  0.61622248]
 [ 3.94191443  2.67285704]
 [ 2.18436067  5.72221018]]
```



Kmeans In Spark: Cluster Assignment Function

Kmeans in Spark

- **Notebook**
  - https://www.dropbox.com/s/41q9lgyqhy8ed5g/EM-Kmeans.ipynb?dl=0

- **NBViewer**
  - http://nbviewer.ipython.org/urls/dl.dropbox.com/s/41q9lgyqhy8ed5g/EM-Kmeans.ipynb

- **End of Lecture**