OPS 813: Cloud Computing

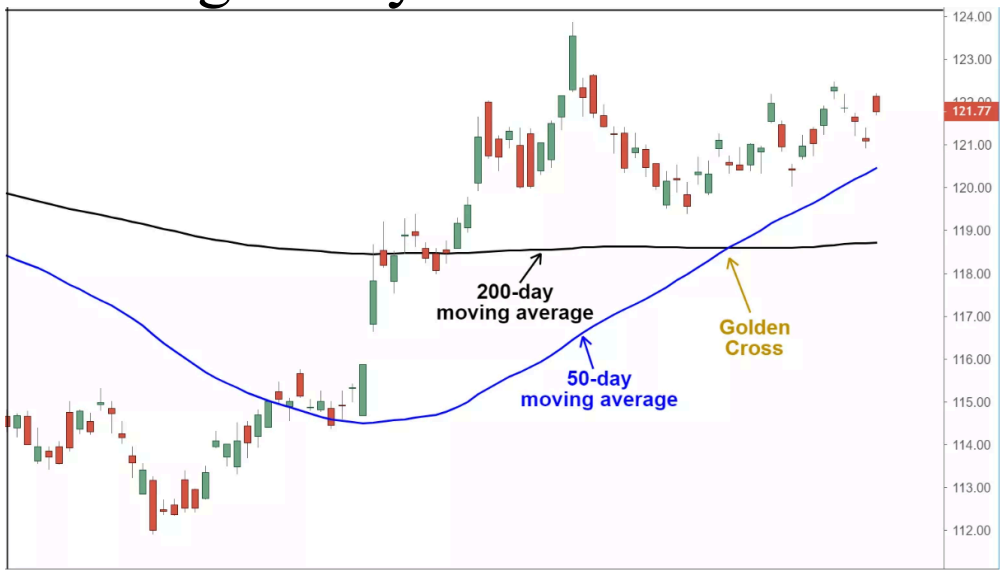
-Today's plan:

1) Case #3: Stock Market data repository & secondary analysis
2) Hadoop/Map Reduce
3) Spark
4) More Cloud SQL (if time permits)
5) Dataproc (if time permits)

Case #2

For your case #2:

- Please download the stock data from this link to a cloud bucket:
  https://www.kaggle.com/qks1lver/amex-nyse-nasdaq-stock-histories/home
- Put everything into one SQL database
- Then (1) find all stocks that are currently below their 200 day moving average, (2) find all stocks that are currently above their 200 day moving average using Datalab. (3) Also calculate the 50 DMA and see how it relates to the 200 DMA. (4) Is there a correlation between the two and the direction a stock moves? (5) as a group choose 6 or 7 stocks that seem interesting to buy and hold for a month. (6) if you have time, are there are metrics that predict direction?
- Every group will be handing in a report. One group will also present.



Qwik Labs (HW1-HW3)

1) You should now have three badges. Congratulations! Showcase them on your resume/linkedin.

2) Remember to please follow these instructions: https://support.google.com/qwiklabs/answer/9222527?hl=en&ref_topic=9139328 and email me (ns27) the link to your public profile.

GCP Essentials

Baseline: Data, ML, AI

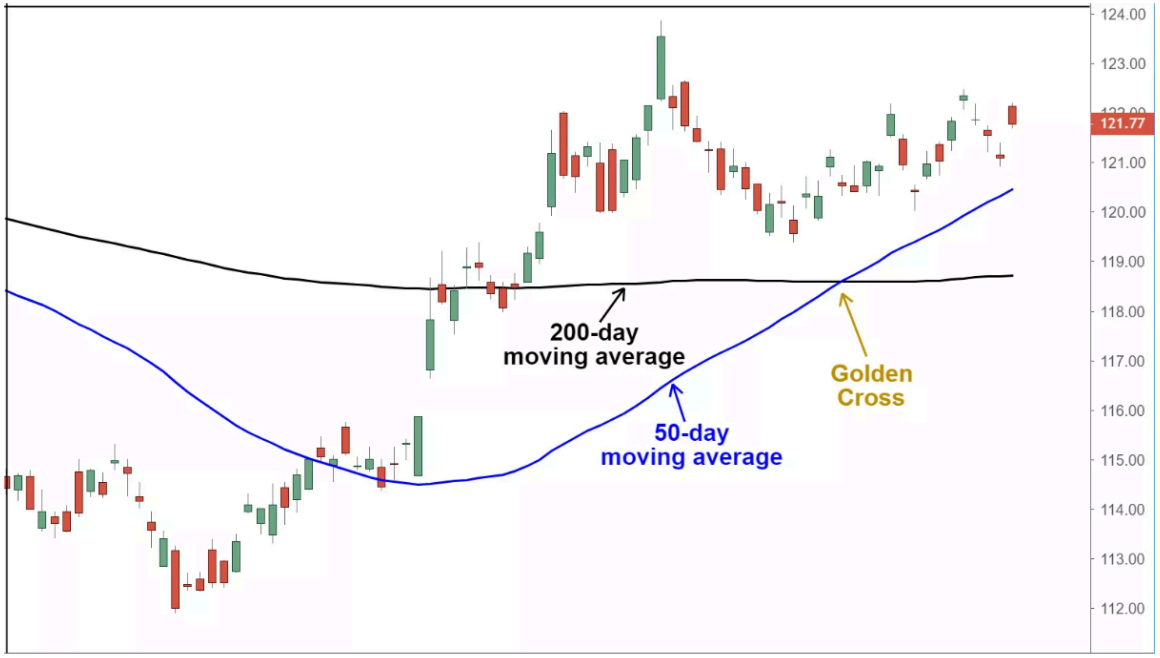
Data Engineering

Case #3

- Use the stock data from this link:
  https://www.kaggle.com/qks1lver/amex-nyse-nasdaq-stock-histories/home
  and Dataproc or some parallelization technology in order to:
- Steps (1) find all ‘golden crosses’, (2) what percentage of those golden crosses result in a price increase by more than 1%, 2%, 3%, 4%, 5%, 10%, 15%, 20% over a week, two weeks, three weeks, a month, two months, three months, four months, five months, six months? (3) find all ‘death crosses’ (when 50DMA crosses below the 200DMA), (4) what percentage of those death crosses result in a price decrease by more than 1%, 2%, 3%, 4%, 5%, 10%, 15%, 20% over a week, two weeks, three weeks, a month, two months, three months, four months, five months, six months?, (5) Does your analytics support the hypothesis of a golden cross or death cross?, (6) as a group choose 6 or 7 stocks that seem interesting to buy and hold for a month.
- Every group will be handing in a report along with their Python code. One group will also present.
- Note that efficiency will be important.

Case #3

‘Golden cross’:



‘Death cross’:

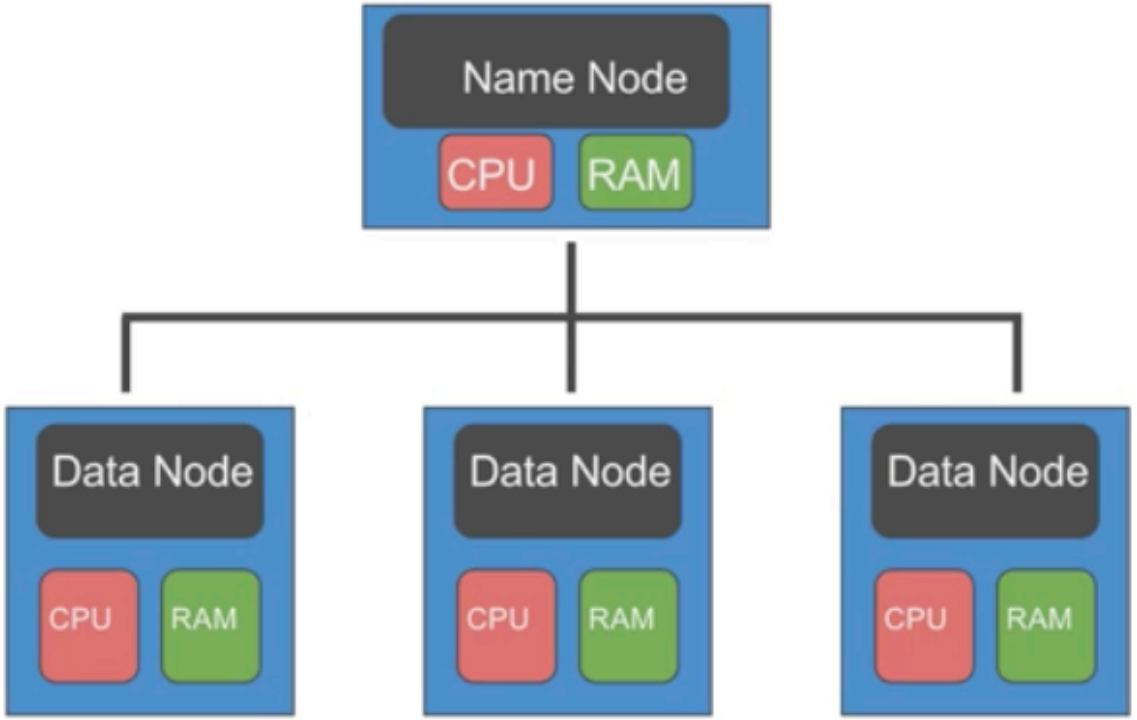
Last time: Docker containers, Datalab, Git/Ungit Cloud SQL

Platform for Big Data: Hadoop

- Hadoop is a way to distribute very large files across multiple machines
- It uses the Hadoop Distributed File System (HDFS)
- HDFS allows a user to work with large data sets
- HDFS also duplicates blocks of data for fault tolerance
  It also then uses MapReduce
- MapReduce allows computations on that data

Platform for Big Data: HDFS

- HDFS will use blocks of data, with a size of 128 MB by default
- Each of these blocks is replicated 3 times
- The blocks are distributed in a way to support fault tolerance

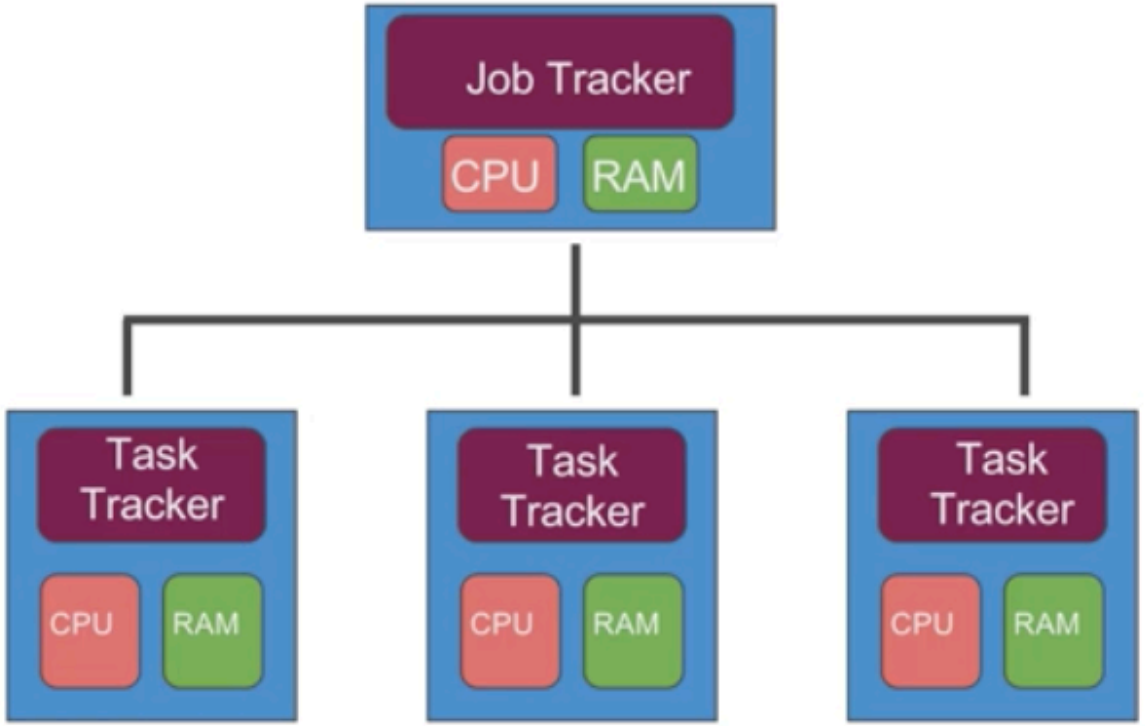


Platform for Big Data: MapReduce

- MapReduce distributes the processing of very large multi-structured data files across a large cluster of ordinary (**commodity**) machines/processors
- Goal - achieving high **parallel** performance with "simple" computers
- Developed and popularized by Google (>20PB per day processed)
- Good at processing and analyzing large volumes of multi-structured data in a timely manner
- Used by Yahoo!, Facebook, Amazon, and the list is growing …

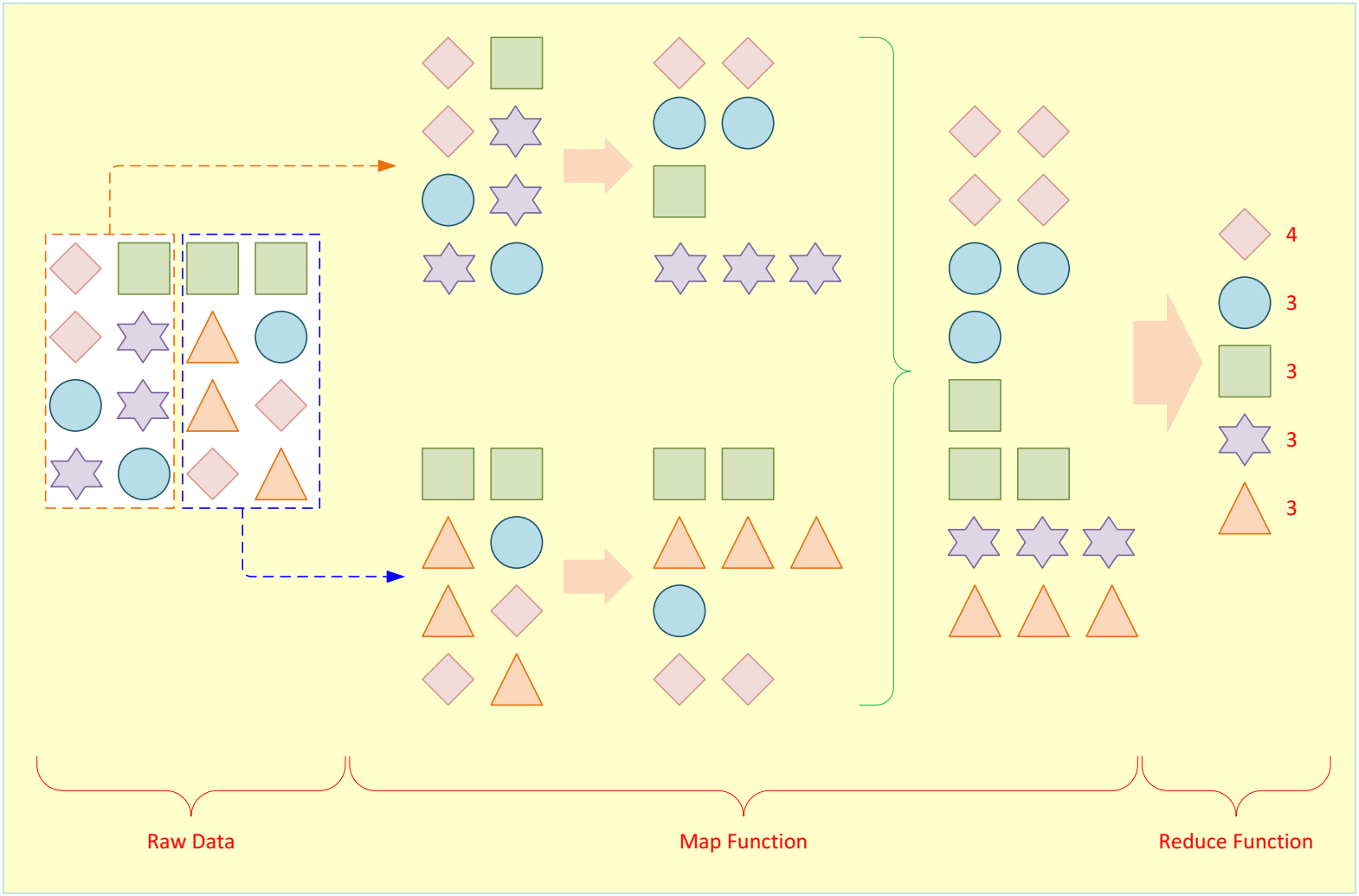
Platform for Big Data: MapReduce

- MapReduce is a way of splitting a computation task to a distributed set of files (such as HDFS)
- It consists of a Job Tracker and multiple Task Trackers



Big Data Technologies - MapReduce

How does MapReduce work?

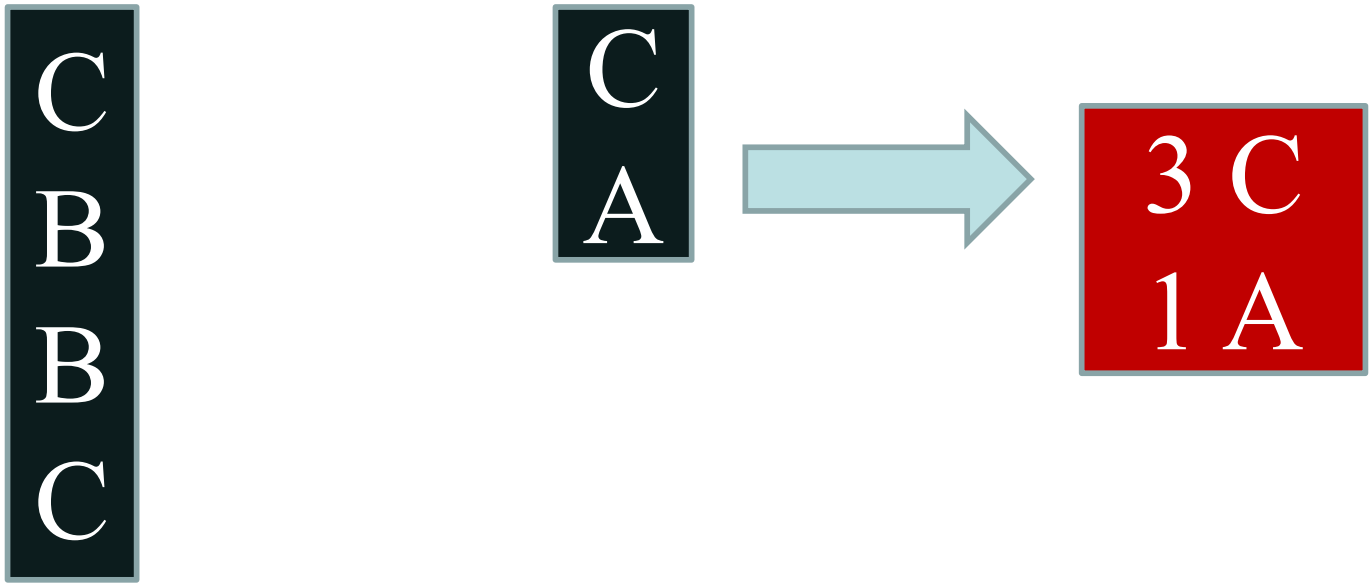


Example 1: Distributed Search

For Unix guru:

grep -Eh <regex> <inDir>/* | sort | uniq -c | sort -nr

- counts lines in all files in <inDir> that match <regex> and displays the counts in descending order



- grep -Eh 'A|C' in/* | sort | uniq -c | sort -nr

- **Actual application:** Analyzing web server access logs to find the top requested pages that match a given pattern

Example 1: Distributed Search

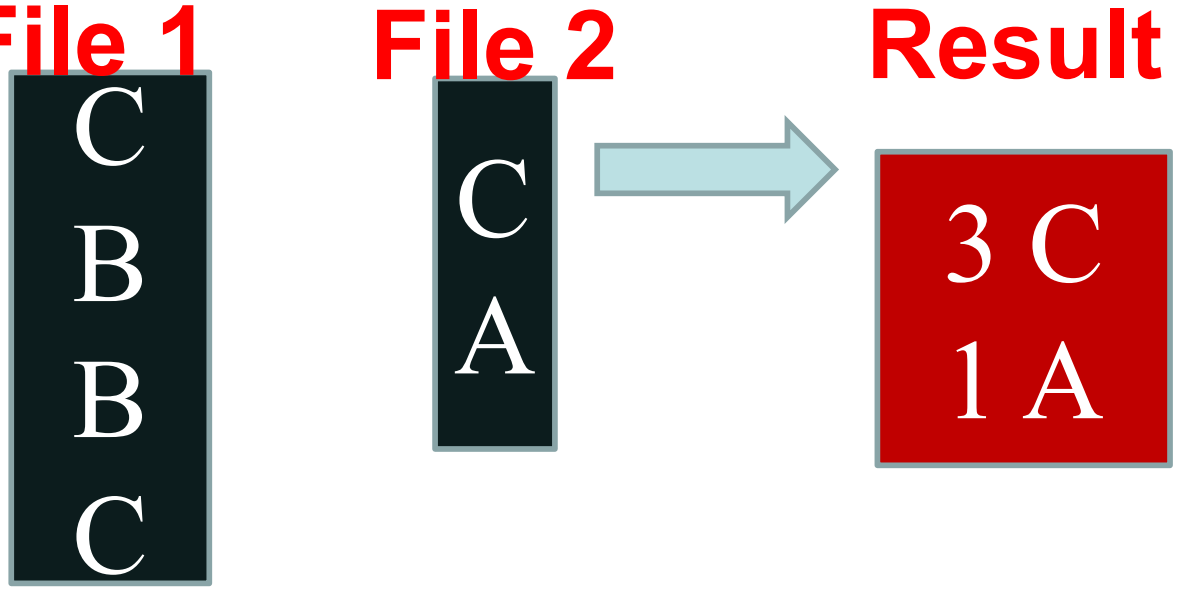
Map function in this case:

- input is (file offset, line)
- output is either:
    1. an empty list [] (the line does not match)
    2. a key-value pair denoted by line:1 (if it matches)

Reduce function in this case:

- input is key-values eg) line: [1, 1, ...]
- output is line:n where n is the number of 1s in the list.

Example 1: Distributed Search



1) Map tasks:
(File1, C) -> C: 1
(File1, B) -> []
(File1, B) -> []
(File1, C) -> C: 1
(File2, C) -> C: 1
(File2, A) -> A: 1

2) Grouping values together by key

3) Reduce tasks:
A: [1] -> A: 1
C: [1, 1, 1] -> C: 3

Example 2: Counting number of words of a given length

- Map would take a word and output the key:value pair where key=length of word & value=the word.

  Eg) map(steve) would return 5:steve and map(savannah) would return 8:savannah.

Example 2: Counting number of words of a given length

Before we get to the reduce function, the MapReduce framework groups all of the values together by key.

Eg) If map function outputs the following key:value pairs

3 : the
3 : and
3 : you
4 : then
4 : what
4 : when
5 : steve
5 : where
8 : savannah
8 : research

They get grouped as:

3 : [the, and, you]
4 : [then, what, when]
5 : [steve, where]
8 : [savannah, research]

Example 2: Counting number of words of a given length

Given the key:values below

3 : [the, and, you]
4 : [then, what, when]
5 : [steve, where]
8 : [savannah, research]]

Each line is passed on to the reduce function, which accepts a key and a list of values. Our reduce function counts the number of items in the list and outputs:

3 : 3
4 : 3
5 : 2
8 : 2

Example 3: Helping Facebook calculate everyone's common friends once a day

Assume the friends are stored as Person->[List of Friends], our friends list is then:

- A -> B C D
- B -> A C D E
- C -> A B D E
- D -> A B C E
- E -> B C D

Each line will be an argument to a mapper. For every friend in the list of friends, the mapper will output a key-value pair. The key will be a friend along with the person. The value will be the list of friends. The key will be sorted so that the friends are in order, causing all pairs of friends to go to the same reducer.

Example 3: Helping Facebook calculate everyone's common friends once a day

- For map(A -> B C D) :
- (A B) -> B C D
- (A C) -> B C D
- (A D) -> B C D
- For map(B -> A C D E) : (Note that A comes before B in the key)
- (A B) -> A C D E
- (B C) -> A C D E
- (B D) -> A C D E
- (B E) -> A C D E

Example 3: Helping Facebook calculate everyone's common friends once a day

For map(C -> A B D E) :
(A C) -> A B D E
(B C) -> A B D E
(C D) -> A B D E
(C E) -> A B D E

For map(D -> A B C E) :
(A D) -> A B C E
(B D) -> A B C E
(C D) -> A B C E
(D E) -> A B C E

And finally for map(E -> B C D):

(B E) -> B C D
(C E) -> B C D
(D E) -> B C D

Example 3: Helping Facebook calculate everyone's common friends once a day

Before we send these key-value pairs to the reducers, we group them by their keys and get:

(A B) -> (A C D E) (B C D)
(A C) -> (A B D E) (B C D)
(A D) -> (A B C E) (B C D)
(B C) -> (A B D E) (A C D E)
(B D) -> (A B C E) (A C D E)
(B E) -> (A C D E) (B C D)
(C D) -> (A B C E) (A B D E)
(C E) -> (A B D E) (B C D)
(D E) -> (A B C E) (B C D)

Then, each line will be passed as an argument to a reducer. The reduce function will simply intersect the lists of values and output the same key with the result of the intersection. Eg) reduce((A B) -> (A C D E) (B C D)) will output (A B) : (C D) and means that friends A and B have C and D as common friends.

Example 3: Helping Facebook calculate everyone's common friends once a day

- The result after reduction is:

  (A B) -> (C D)
  (A C) -> (B D)
  (A D) -> (B C)
  (B C) -> (A D E)
  (B D) -> (A C E)
  (B E) -> (C D)
  (C D) -> (A B E)
  (C E) -> (B D)
  (D E) -> (B C)

- Now when D visits B's profile, we can quickly look up (B D) and see that they have three friends in common, (A C E).

Hadoop

- Hadoop is an open source framework for storing and analyzing massive amounts of distributed, unstructured data
- Originally created by Doug Cutting at Yahoo!
- Hadoop clusters run on inexpensive commodity hardware so projects can scale-out inexpensively
- Hadoop is now part of Apache Software Foundation
- Open source - hundreds of contributors continuously improve the core technology
- MapReduce + Hadoop = Big Data core technology

Hadoop

- How Does Hadoop Work?
  - Access unstructured and semi-structured data (e.g., log files, social media feeds, other data sources)
  - Break the data up into "parts," which are then loaded into a file system made up of multiple nodes running on commodity hardware using HDFS
  - Each "part" is replicated multiple times and loaded into the file system for replication and failsafe processing
  - A node acts as the Facilitator and another as Job Tracker
  - Jobs are distributed to the clients, and once completed the results are collected and aggregated using MapReduce

Hadoop

- Hadoop Technical Components
  - Hadoop Distributed File System (HDFS)
  - Name Node (primary facilitator)
  - Secondary Node (backup to Name Node)
  - Job Tracker
  - Slave Nodes (the grunts of any Hadoop cluster)
  - Additionally, Hadoop ecosystem is made-up of a number of complementary sub-projects: NoSQL (Cassandra, Hbase), DW (Hive), …
    - NoSQL = not only SQL

Hadoop - Demystifying Facts

- Hadoop consists of multiple products
- Hadoop is open source but available from vendors, too
- Hadoop is an ecosystem, not a single product
- HDFS is a file system, not a DBMS
- Hive resembles SQL but is not standard SQL
- Hadoop and MapReduce are related but not the same
- MapReduce provides control for analytics, not analytics
- Hadoop is about data diversity, not just data volume.
- Hadoop complements a DW; it's rarely a replacement.
- Hadoop enables many types of analytics, not just Web analytics.

Spark

- Spark is one of the latest technologies being used to quickly and easily handle Big Data
- It is an open source project on Apache
- It was first released in February 2013 and has exploded in popularity due to it's ease of use and speed
- It was created at the AMPLab at UC Berkeley

Spark

- You can think of Spark as a flexible alternative to MapReduce
- Spark can use data stored in a variety of formats
  - Cassandra
  - AWS S3
  - HDFS
  - And more

Spark vs MapReduce

- MapReduce requires files to be stored in HDFS, Spark does not!
- Spark also can perform operations up to 100x faster than MapReduce
- So how does it achieve this speed?

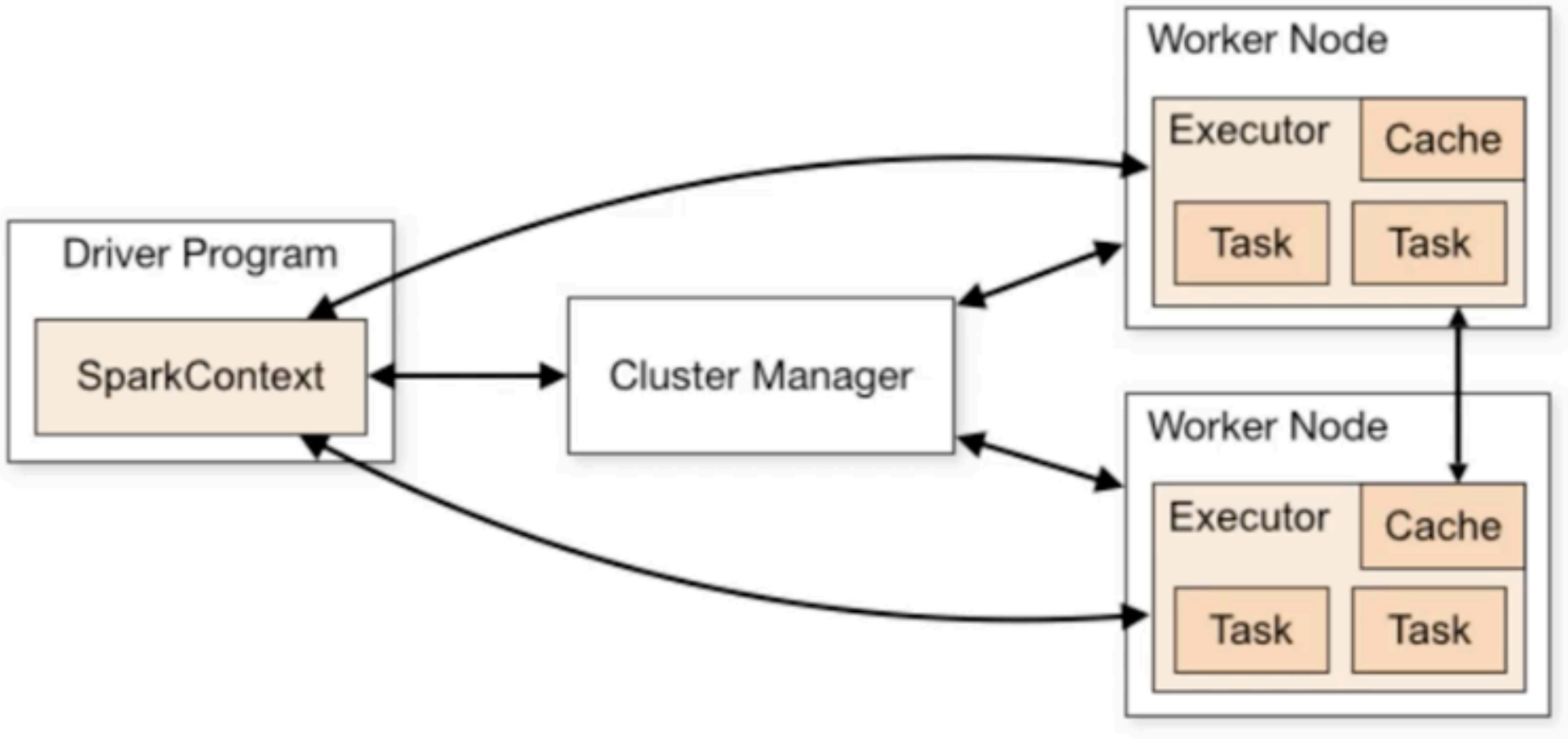
Spark vs MapReduce

- MapReduce writes most data to disk after each map and reduce operation
- Spark keeps most of the data in memory after each transformation
- Spark can spill over to disk if the memory is filled

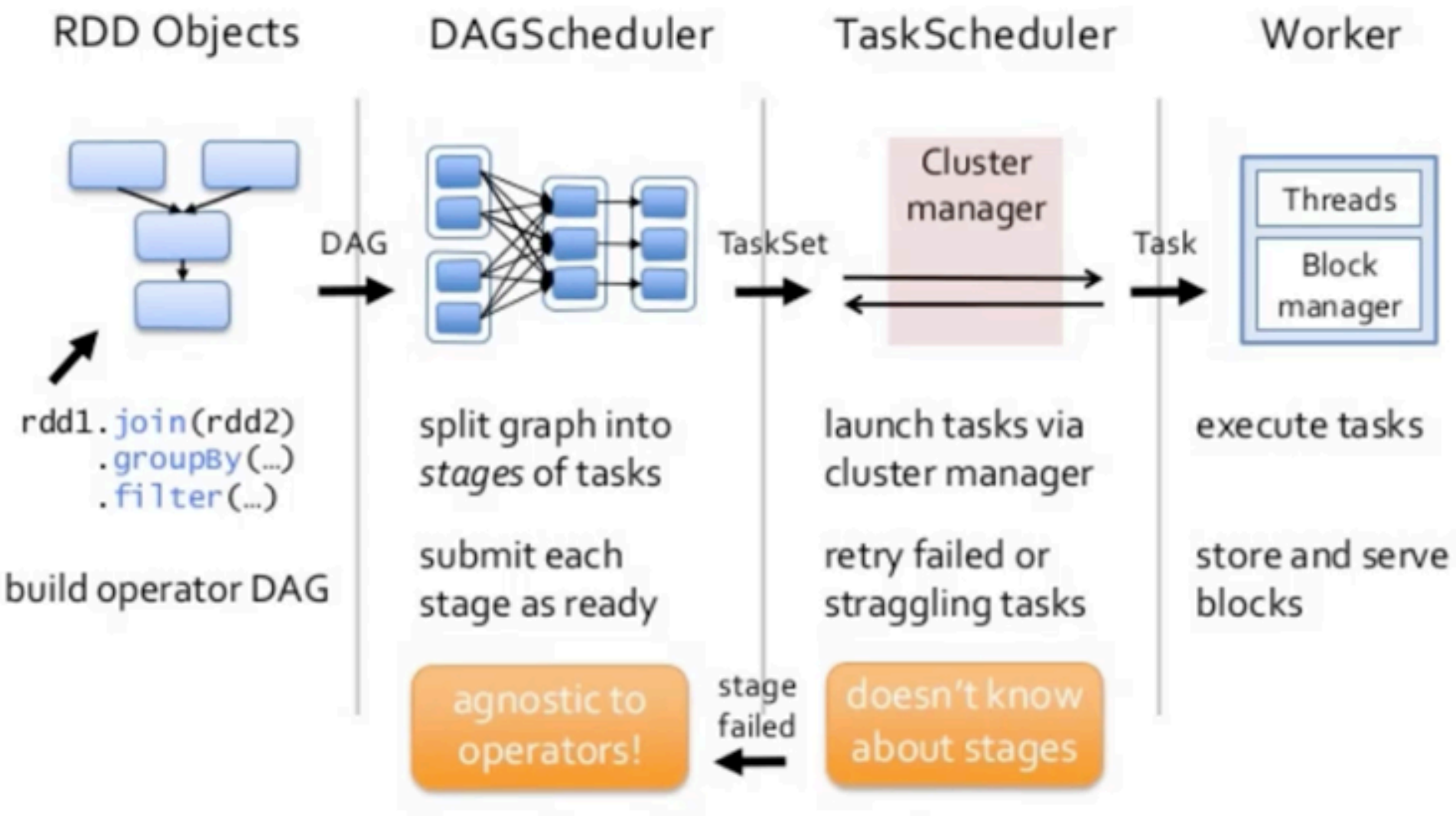
Spark RDDs

- At the core of Spark is the idea of a Resilient Distributed Dataset (RDD)
- Resilient Distributed Dataset (RDD) has 4 main features:
  - Distributed Collection of Data
  - Fault-tolerant
  - Parallel operation - partioned
  - Ability to use many data sources

Spark RDDs


Worker Node
Executor
Cache
Task
Task
Driver Program
SparkContext
Cluster Manager
Worker Node
Executor
Cache
Task
Task

Spark RDDs


RDD Objects
DAGScheduler
TaskScheduler
Worker
DAG
TaskSet
Task
Cluster manager
Threads
Block manager
rdd1.join(rdd2)
.groupBy(…)
.filter(…)
build operator DAG
split graph into *stages* of tasks
submit each stage as ready
agnostic to operators!
launch tasks via cluster manager
retry failed or straggling tasks
doesn't know about stages
stage failed
execute tasks
store and serve blocks

Spark RDDs

- RDDs are immutable, lazily evaluated, and cacheable
- There are two types of RDD operations:
  - Transformations
  - Actions

Spark RDD Actions

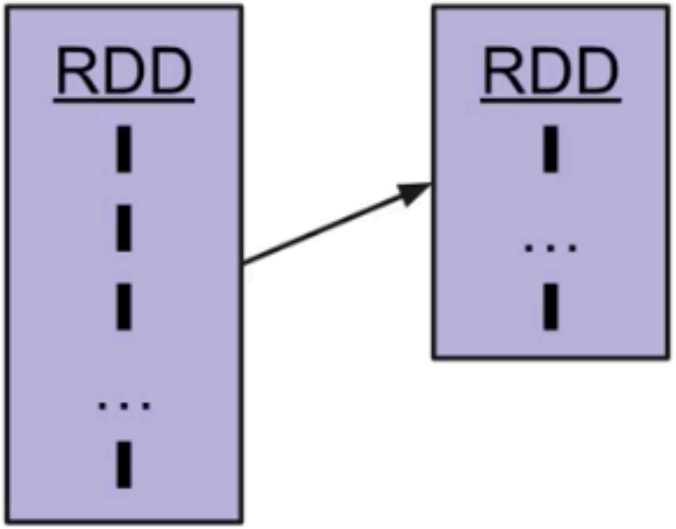
- Collect - Return all the elements of the RDD as an array at the driver program.
- Count - Return the number of elements in the RDD
- First - Return the first element in the RDD
- Take - Return an array with the first n elements of the RDD

Spark RDD Transformations

- Basic Transformations
  - Filter
  - Map
  - FlatMap

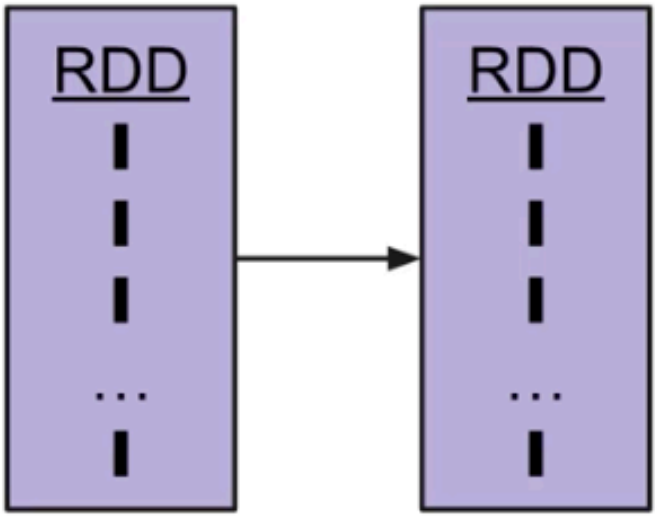
Spark RDD Transformations

- RDD.filter()
  - Applies a function to each element and returns elements that evaluate to true



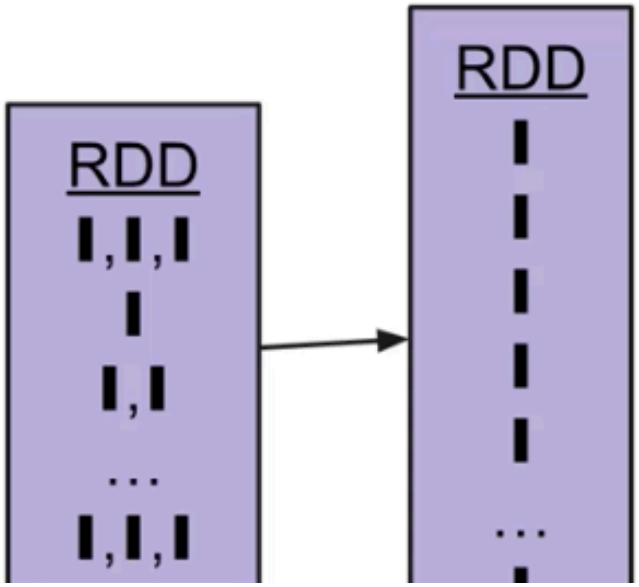
Spark RDD Transformations

- RDD.map()
  - Transforms each element and preserves # of elements, very similar idea to pandas .apply()



Spark RDD Transformations

- RDD.flatMap()
  - Transforms each element into 0-N elements and changes # of elements



- Map()
  - Grabbing first letter of a list of names
- FlatMap()
  - Transforming a corpus of text into a list of words

Pair RDDs

- Often RDDs will be holding their values in tuples
  - (key,value)
- This offers better partitioning of data and leads to functionality based on reduction

Reduce and ReduceByKey

- Reduce()
  - An action that will aggregate RDD elements using a function that returns a single element
- ReduceByKey()
  - An action that will aggregate Pair RDD elements using a function that returns a Pair RDD
- These ideas are similar to a Group By operation

Spark rapidly changing

- Spark is being continually developed and new releases come out often!
- The Spark Ecosystem now includes:
  - Spark SQL
  - Spark DataFrames
  - MLlib
  - GraphX
  - Spark Streaming

Let's setup Spark

- Let's use the bare metal instance on which you installed Jupyter.
- Need Scala (scala depends on Java):
  sudo apt-get update
  sudo apt-get install default-jre (installs Java)
  java –version
  sudo apt-get install scala
  scala –version
- Need py4j:
  which pip (otherwise need conda install pip)
  pip install py4j
- Install Spark:
  wget https://archive.apache.org/dist/spark/spark-2.3.3/spark-2.3.3-bin-hadoop2.7.tgz
  uncompress: gunzip and tar -xvf

Let's setup Spark

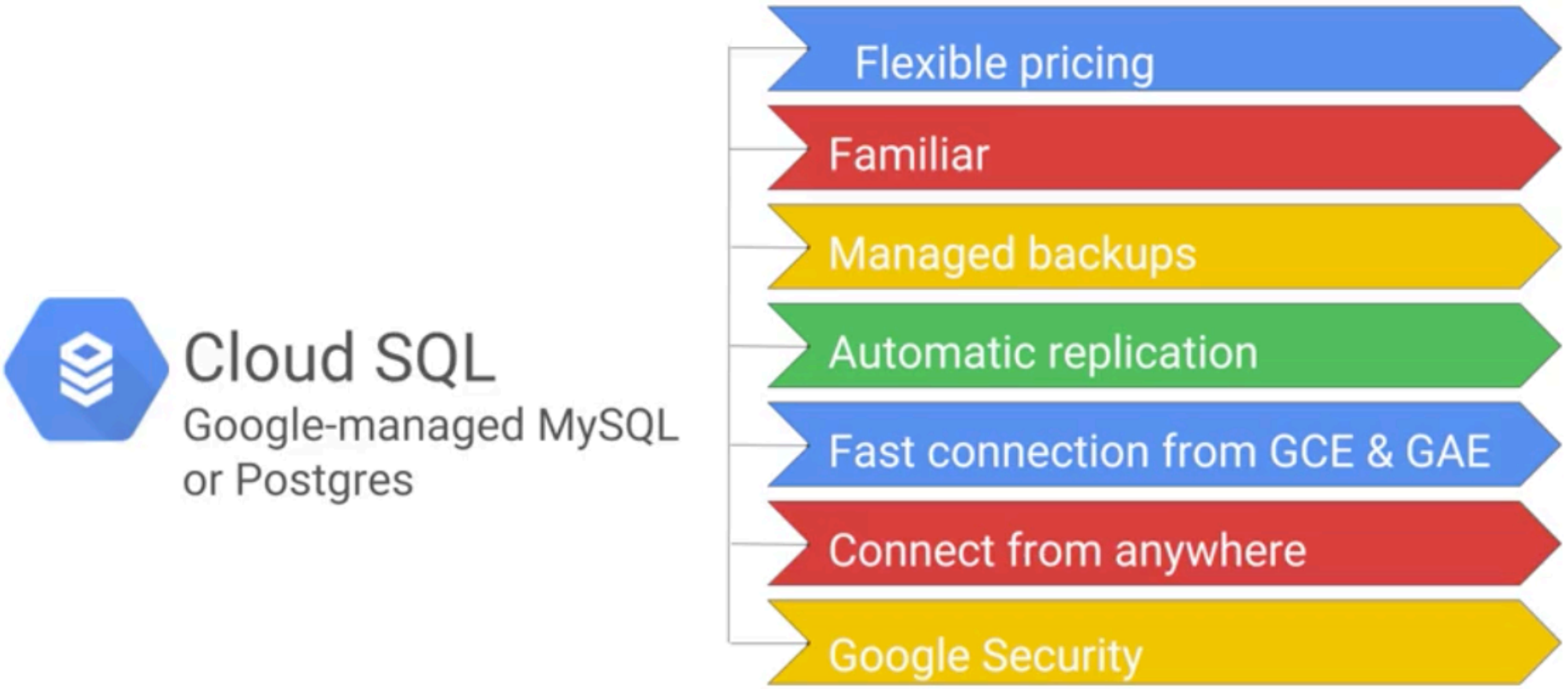
- Tell it where to find Spark:
  export SPARK_HOME='/home/ns27/spark-2.3.3-bin-hadoop2.7'
  export PATH=$SPARK_HOME:$PATH
  export PYTHONPATH=$SPARK_HOME/python:$PYTHONPATH

- Now open up a Jupyter notebook from the bare metal instance and type:
  from pyspark import SparkContext
  sc=SparkContext()
  (if it is installed correctly it should go through fine)

SQL Database in the Cloud

	Cloud Storage	Cloud SQL	Datastore	Bigtable	BigQuery
Capacity	Petabytes +	Gigabytes	Terabytes	Petabytes	Petabytes
Access metaphor	Like files in a file system	Relational database	Persistent Hashmap	Key-value(s), HBase API	Relational
Read	Have to copy to local disk	SELECT rows	filter objects on property	scan rows	SELECT rows
Write	One file	INSERT row	put object	put row	Batch/stream
Update granularity	An object (a "file")	Field	Attribute	Row	Field
Usage	Store blobs	No-ops SQL database on the cloud	Structured data from AppEngine apps	No-ops, high throughput, scalable, flattened data	Interactive SQL* querying fully managed warehouse

What is Cloud SQL?



Rentals Case Study

- upload mtg04.zip to your Cloud Shell (not your instance)
- start your Cloud Shell and unzip mtg04
- move into the rentals-sql folder (within the mtg04 folder)
- check out the files in the cloudsql folder
- try:
  head *.csv
- move these files into a bucket, how?

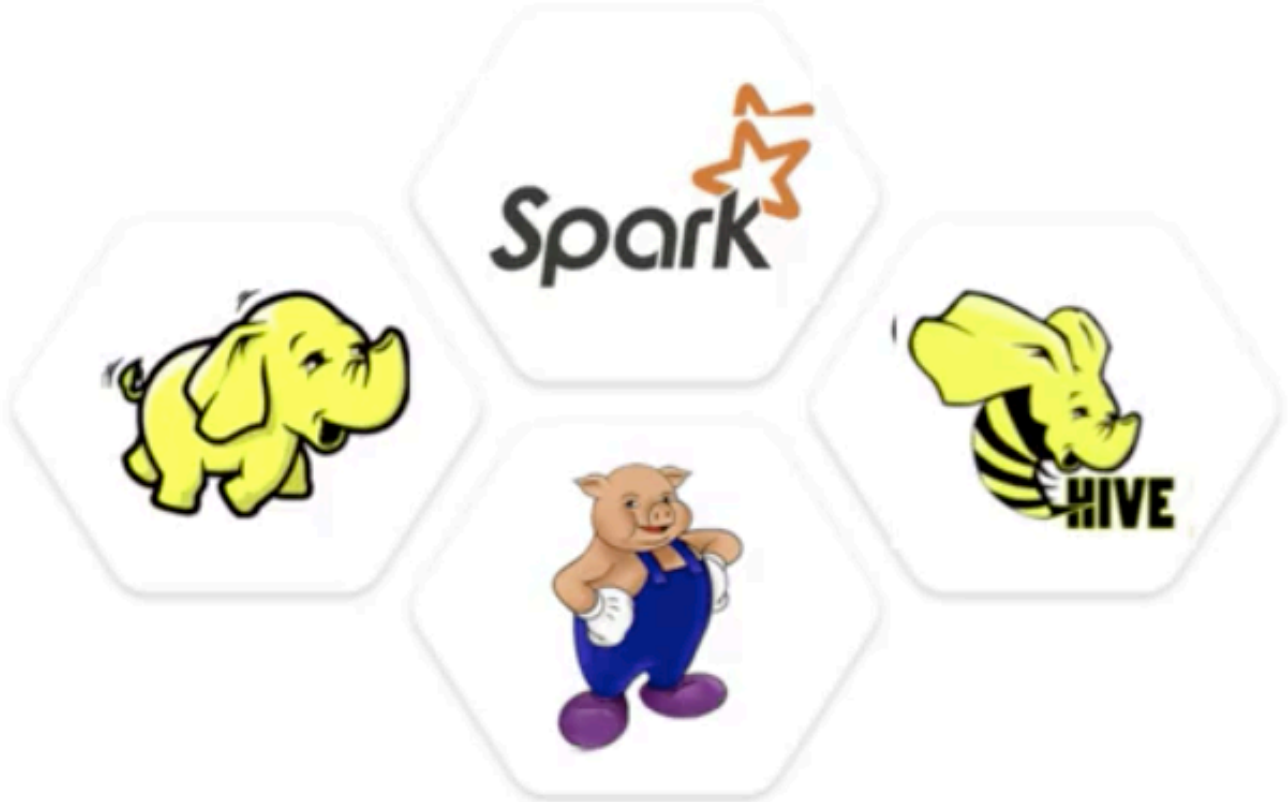
- now create a Cloud SQL instance named rentals
  (when creating instance, choose second generation, and authorize networks under show configuration options, run script to ip address) – note the password
- keep track of public IP address for SQL instance
- go into the SQL instance and click import, and select SQL file
- go into SQL instance and click import again, and use recommendation spark for the csv files

Rentals Case Study

- You can now use the mysql cli from your Cloud Shell:
  mysql --host=PUBLIC_IP –user=root --password
- Enter your password

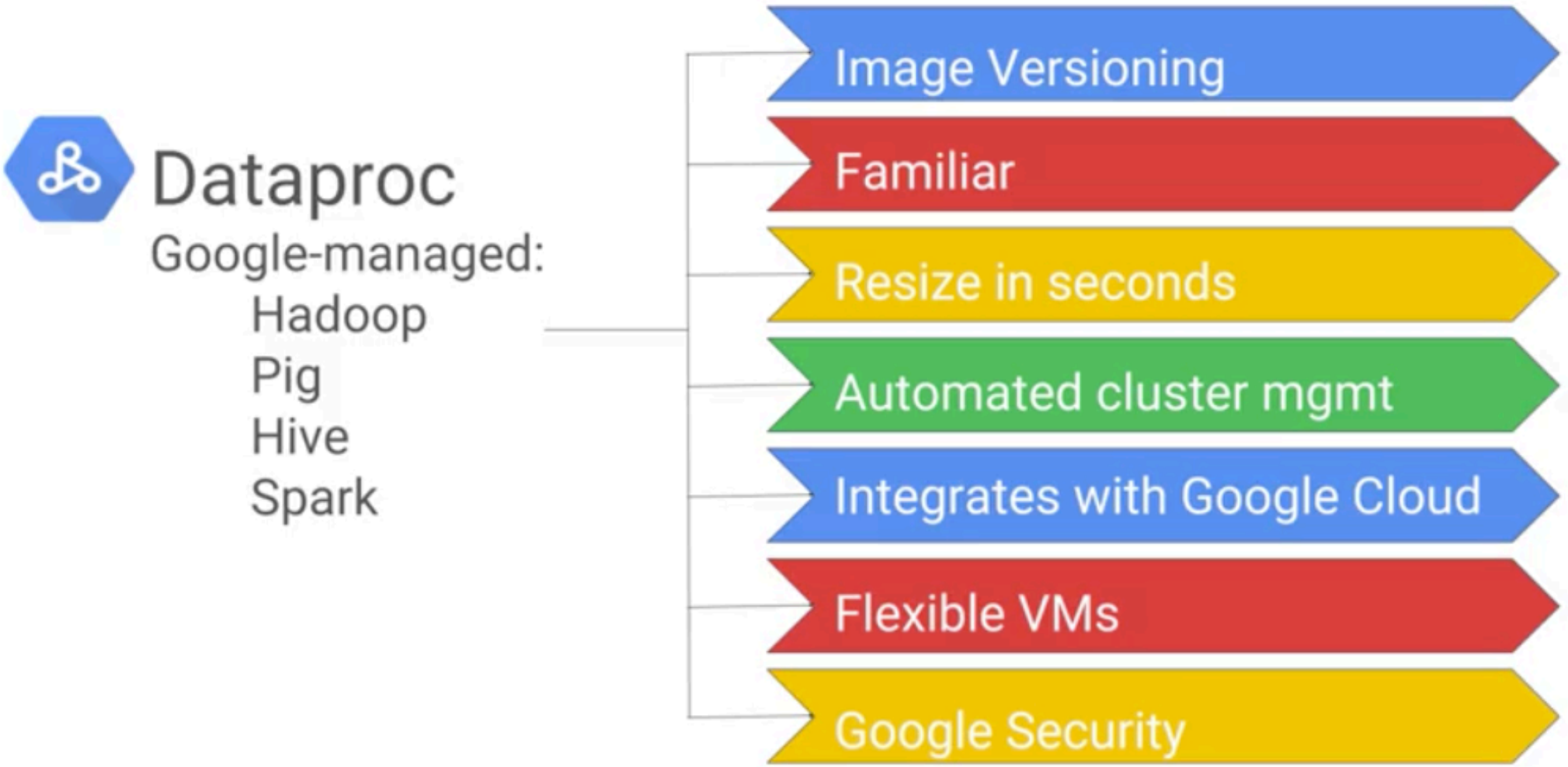
- SQL time:
  use recommendation_spark;
  show tables;
  select * from Rating;
  select * from Accomodation where type = ‘castle’ and price < 1500;

Big Data Open-Source Ecosystem



http://hadoop.apache.org/
http://pig.apache.org/
http://hive.apache.org/
http://spark.apache.org/

What is Dataproc? With Dataproc you can separate the storage and compute lifecycles.



Let’s use Dataproc.