Introduction to Spark

Galvanize, Denver
Ibotta, Denver

About Me...

- Graduated from Wesleyan University with a degree in Mathematics
- Attended the Galvanize Data Science Immersive in 2015
- Worked as an Instructor of Data Science for a year and a half
- Recently joined Ibotta Inc.'s Data Science team where Spark is used to deploy a variety of Machine Learning models

Introduction to Spark

Galvanize, Denver
Ibotta, Denver

OBJECTIVES

- **Describe** the pros/cons of Spark compared to Hadoop MapReduce
- **Define** what an RDD is, by its properties and operations
- **Explain** the difference between transformations/actions on an RDD and how they affect the DAG
- **Introduce** Spark DataFrames and the Spark ML Library
- **Walk** through demo applying LDA to text data

Why Spark?

Data science friendly parallel computing

- Fast and general engine for large-scale data processing
- Highly efficient distributed operations
- Supports acyclic data flow & in-memory computing
- Supports Python, Scala, Java, & R

Apache Hadoop integration

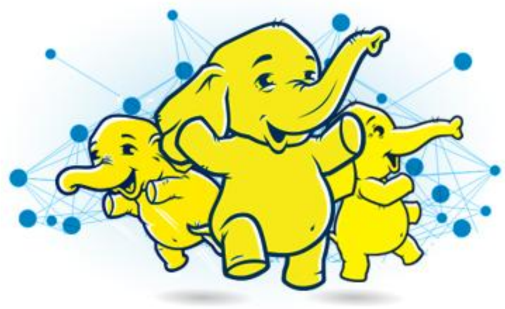
- ~~Seamless~~ Relatively easy integration into existing eco-systems (HDFS)
- Scalability, reliability, resilience

And… machine learning functions available!

Spark Vs. Hadoop

Hadoop Ecosystem

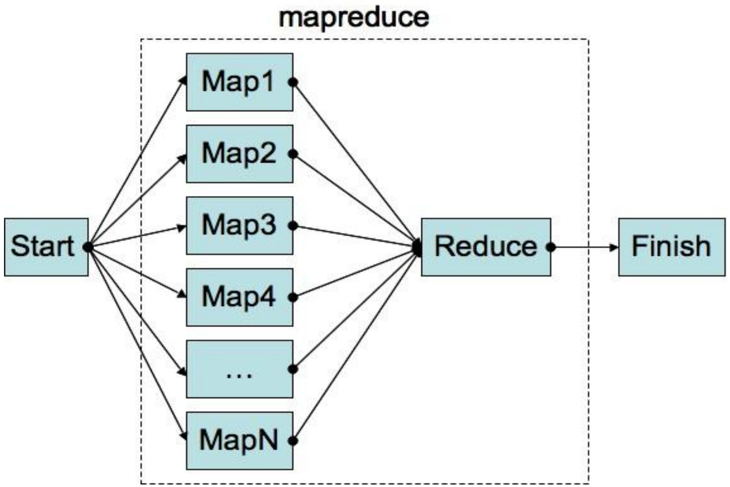
- The Hadoop Ecosystem consists of a variety of modules including:
    - Hadoop Common
    - Hadoop Distributed File System (HDFS)
    - Hadoop YARN
    - Hadoop MapReduce



- When comparing Spark Vs. Hadoop, we are usually comparing Spark Vs. Hadoop MapReduce

Hadoop MapReduce

- Hadoop MapReduce is a batch-processing engine that exclusively works from disk
    - I.e. MapReduce sequentially reads data from disk, performs an operation on the data, & writes the results back to the cluster
- Specialization of the *split-apply-combine* strategy for data analysis



Spark

- Comes with a user-friendly API for Scala, Java, Python, R, and Spark SQL
- Capable of running in an interactive mode
- Can work with data in-memory leading to lightning fast data operations
  - Up to 100x faster than Hadoop MapReduce when performing in-memory operations (up to 10x faster when operating on disk)
- Capable of integrating with existing Hadoop Ecosystem
  - A Spark application can be run on Hadoop clusters through YARN
  - A Spark application can read directly from HDFS

Spark Ecosystem

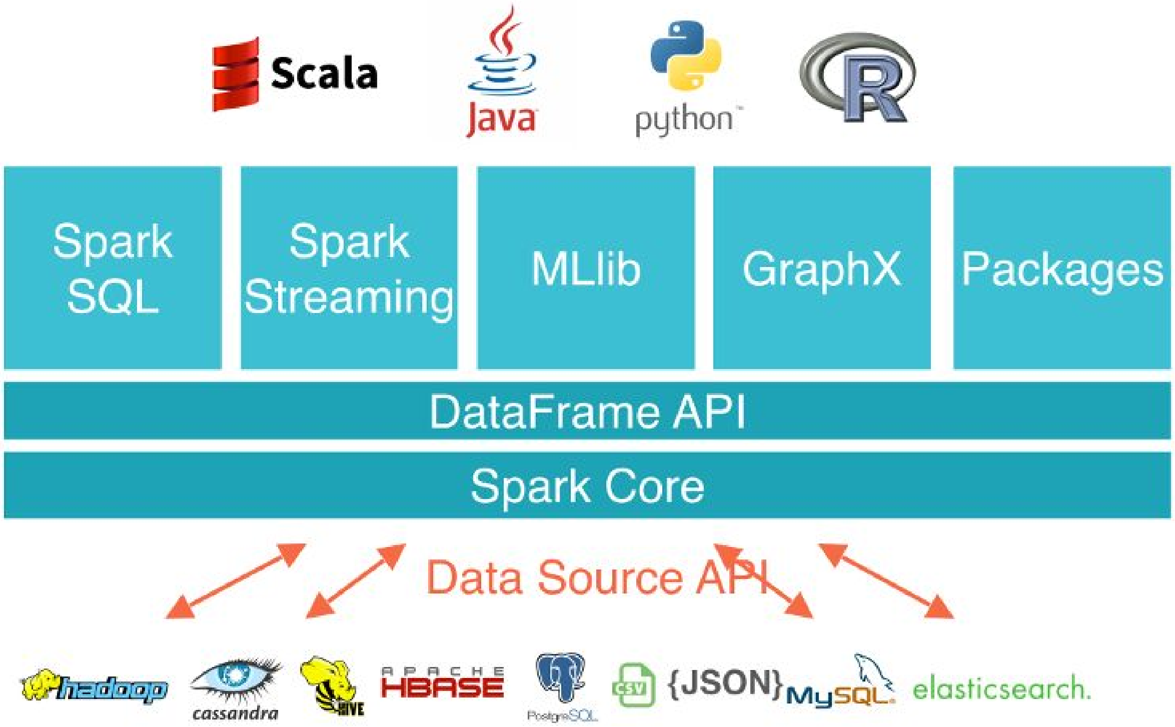


Image Source

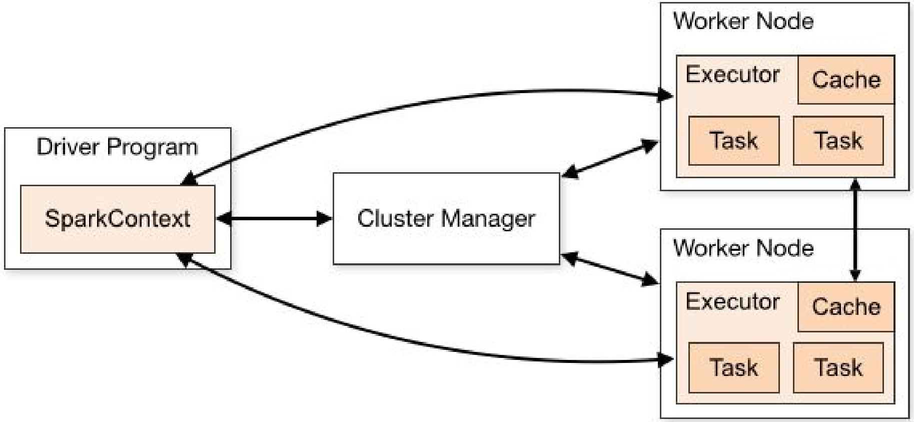
HDFS & Spark

"If Hadoop [HDFS] were ancient Egyptian hieroglyphics, Spark would be the Rosetta stone to understanding the data in Hadoop"

- Donna-M. Fernandez

Spark Execution Model

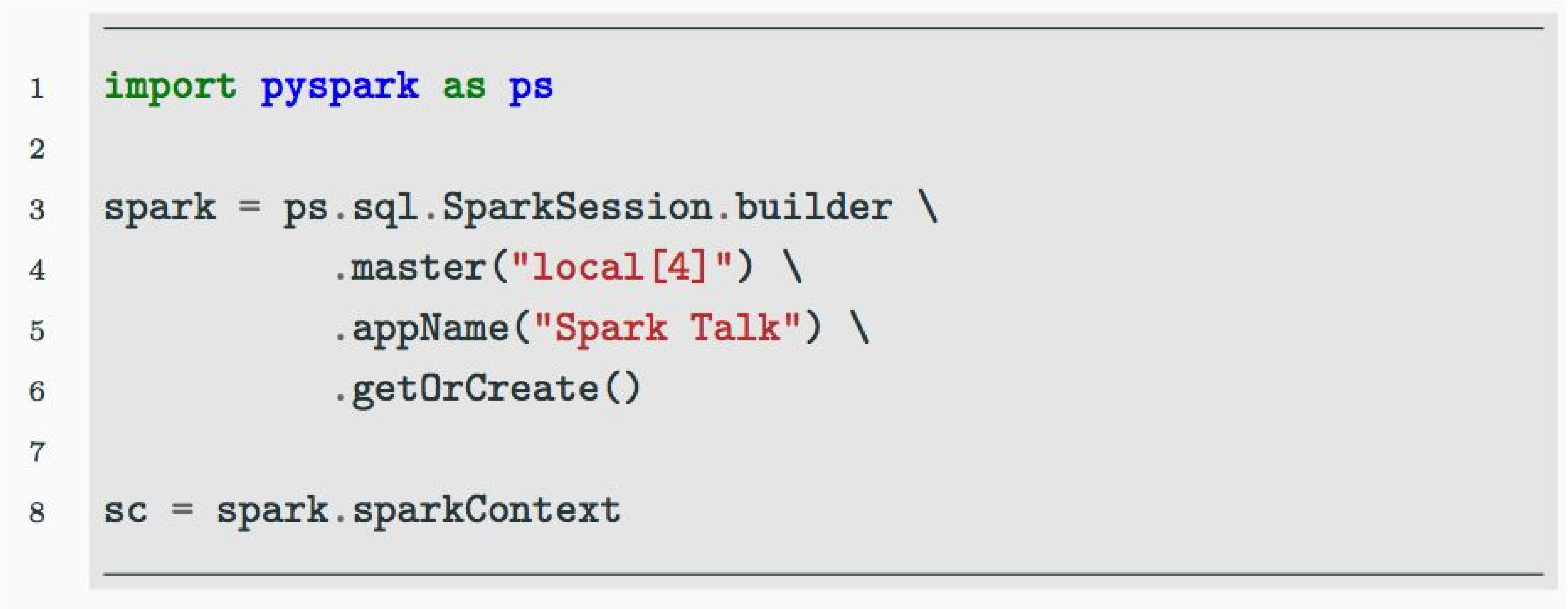
- Each application in Spark has a driver program that distributes tasks among the executors running on nodes in a cluster
- The *client* (e.g. iPython/iPython notebook) will have a `SparkContext` which allows you to interact with the driver program on the *master*. This will....
  - Act as a gateway between the client and the Spark master
  - Sends code/data from iPython to the master (who then sends it to the workers)



SparkSession in Spark 2.0+

- Prior to Spark 2.0, there were a variety of contexts that were required to interact with different aspects of the Spark ecosystem (e.g. `SparkContext`, `SQLContext`, `HiveContext`, `StreamingContext`)
- Spark 2.0+ introduces the concept of a `SparkSession` which can access all of Spark's functionality through a single-unified point of entry
- This serves to minimize the number of concepts to remember or construct as well as making it easier to access DataFrame and Dataset APIs

Using SparkSession Builder



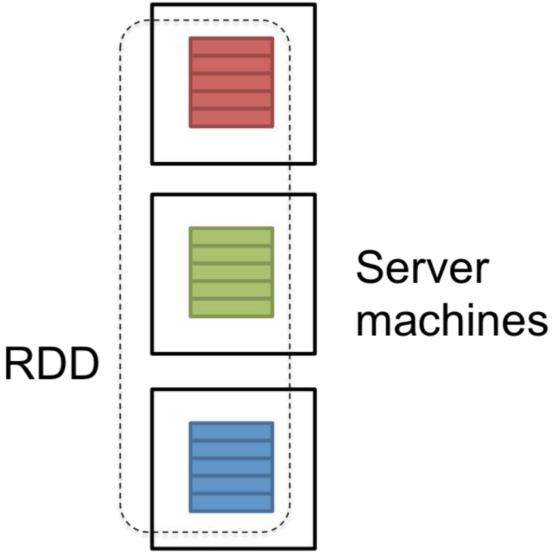
```
import pyspark as ps

spark = ps.sql.SparkSession.builder \
          .master("local[4]") \
          .appName("Spark Talk") \
          .getOrCreate()

sc = spark.sparkContext
```

Spark Resilient Distributed Datasets

(RDDs)

Resilient Distributed Datasets

- Created from HDFS, S3, HBase, JSON, text, local
- Distributed across the cluster as partitions (atomic chunks of data)
- Can recover from errors (node failure, slow process)
- **Immutable** : you *cannot* modify an RDD in place



[Image Source]

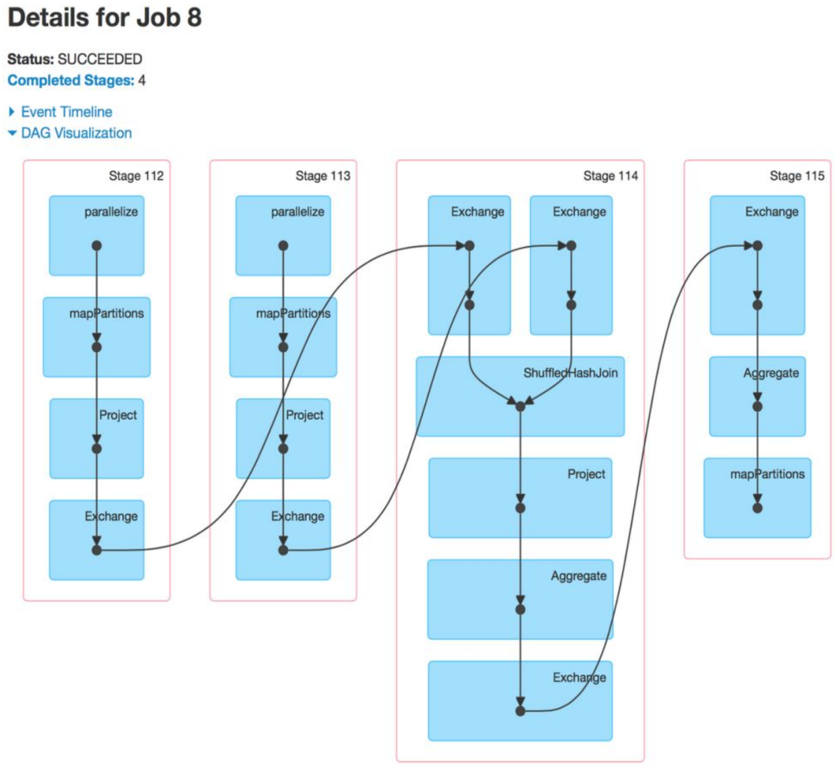
Properties of RDDs

- Spark can create RDDs from any storage source supported by Hadoop (e.g. HDFS, HBase, & c.), including local storage
- RDDs can be persistent in order to cache a dataset to memory; this can lead to significant speedups if a dataset needs to utilized repeatedly in a given application
- RDDs are fault tolerant---if any given partition of an RDD is lost, it will automatically be recomputed by using the recorded transformations in the DAG

Before diving deeper into RDDs, we should touch on what a DAG is...

Directed Acyclic Graph

- A Directed Acyclic Graph (or DAG) is how Spark keeps track of what transformations & actions need to be computed
- ***Lazy Evaluation***
- All operations fall into one of two categories…
  - **Transformations**: Adds a step to the DAG and returns a new RDD object
  - **Actions**: Forces the execution of the DAG and returns some sort of result



Details for Job 8

Status: SUCCEEDED
Completed Stages: 4

Notable RDD Transformations

- These will be evaluated *lazily*---i.e. a step will be added to the DAG but **WILL NOT** launch any computation

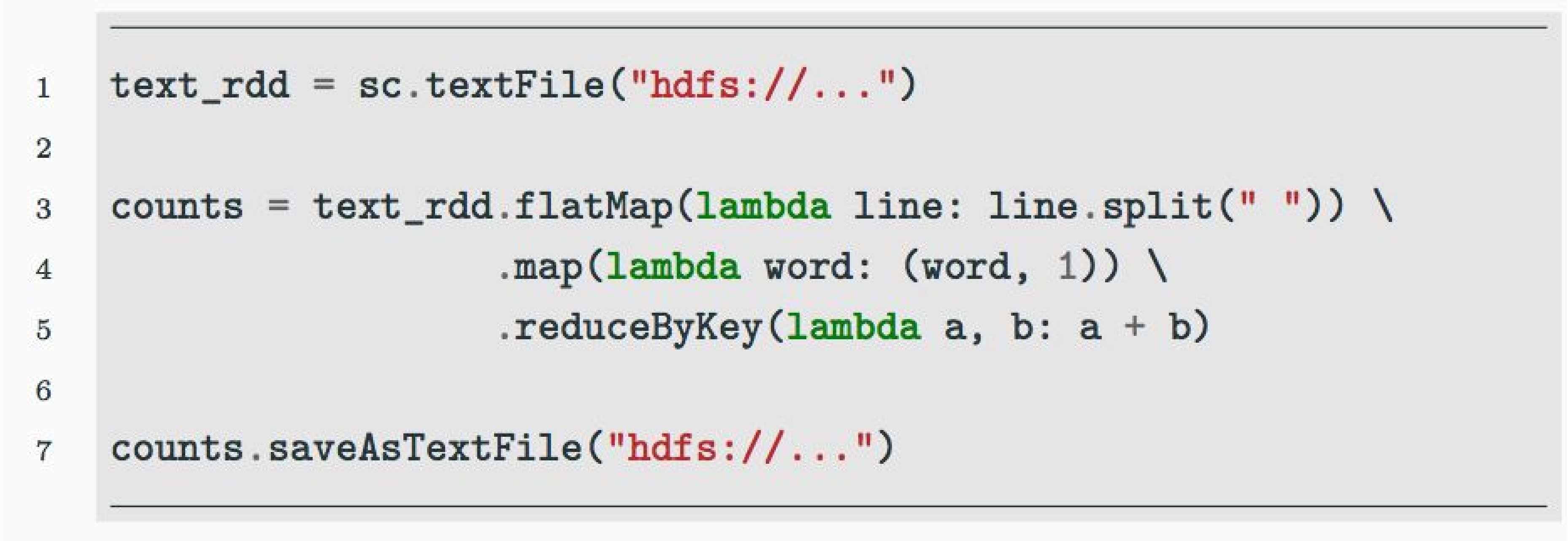
Method	**Category**	**Description**
`.map(func)`	mapping	Return a new RDD by applying a function to each element of this RDD
`.flatMap(func)`	mapping	Return a new RDD by applying a function to each element of this RDD
`.filter(func)`	reduction	Return a new RDD containing only the elements that satisfy a predicate
`.sample()`	reduction	Return a sampled subset of this RDD
`.distinct()`	reduction	Return a new RDD containing the distinct elements in this RDD
`.join(rddB)`	`<k, v>`	Return an RDD containing all pairs of elements with matching keys in self and other. Each pair of elements will be returned as a (k, (v1, v2)) tuple, where (k, v1) is in self and (k, v2) is in other

Notable RDD Actions

- These will transform an RDD into something else (a python object, or a statistic)
- Actions will launch the processing of the DAG; this is where Spark stops being lazy!

Method	**Type**	**Description**
`.collect()`	action	Return a list that contains all of the elements in this RDD. Note that this method should only be used if the resulting array is expected to be small, as all the data is loaded into the driver’s memory
`.count()`	action	Return the number of elements in this RDD
`.take()`	action	Take the first \`n\` elements of the RDD
`.sum()`	action	Add up the elements in this RDD

Word Count Example



```
text_rdd = sc.textFile("hdfs://...")

counts = text_rdd.flatMap(lambda line: line.split(" ")) \
                 .map(lambda word: (word, 1)) \
                 .reduceByKey(lambda a, b: a + b)

counts.saveAsTextFile("hdfs://...")
```

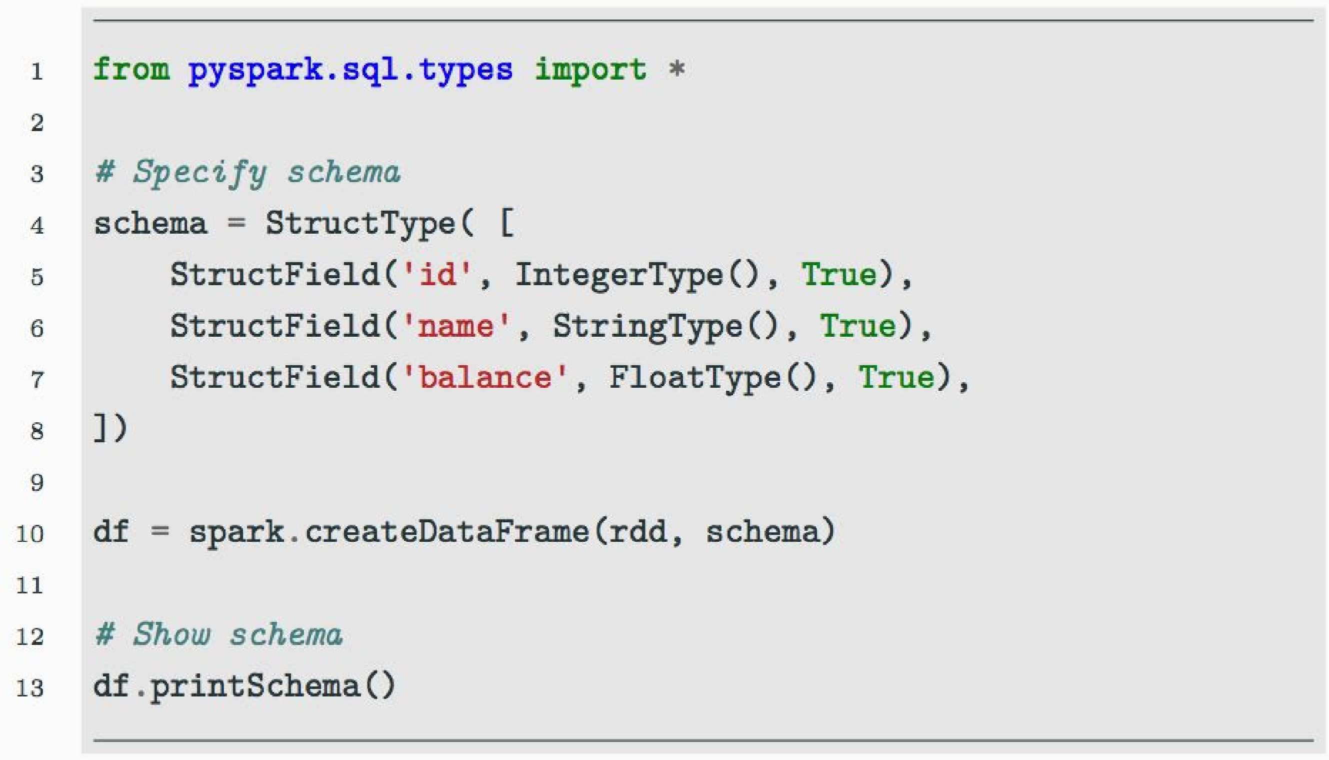
Spark SQL & Spark DataFrames

Spark SQL & Spark DataFrames

- Unlike the traditional Spark RDD API, the Spark SQL module adds additional information about the schema of the data contained in an RDD, thereby allowing extra optimization
- But what is a schema?
  - Schemas are metadata about your data
  - Schema = Table Names + Column Names + Column Types
- What are the Pros of Schemas?
  - Schemas enable **queries** using SQL and DataFrame syntax
  - Schemas also make your data more **structured**

Specifying Your Own Schema

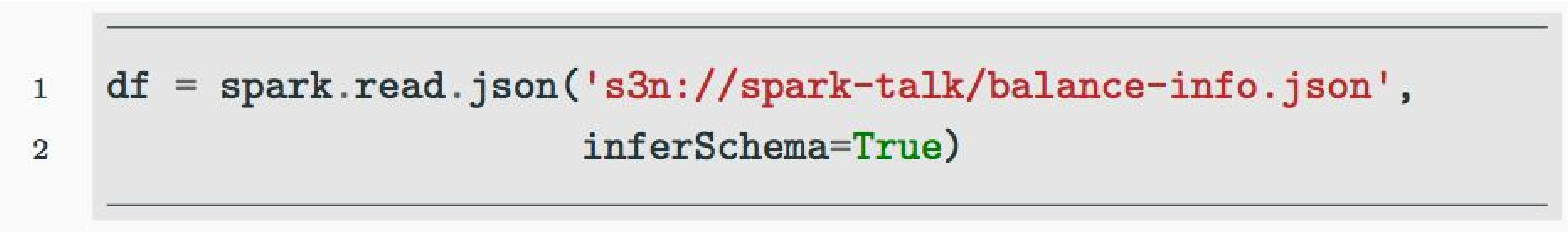
Let’s say we have an RDD with `[(id, name, balance), ... ]` (e.g. `(1234, Erich Wellinger, 42.00)`), we could create a DataFrame like so…



```python
from pyspark.sql.types import *

# Specify schema
schema = StructType( [
    StructField('id', IntegerType(), True),
    StructField('name', StringType(), True),
    StructField('balance', FloatType(), True),
])

df = spark.createDataFrame(rdd, schema)

# Show schema
df.printSchema()
```

Inferring a Schema

We can also have Spark infer a schema when reading data in rather than having to specify it manually…

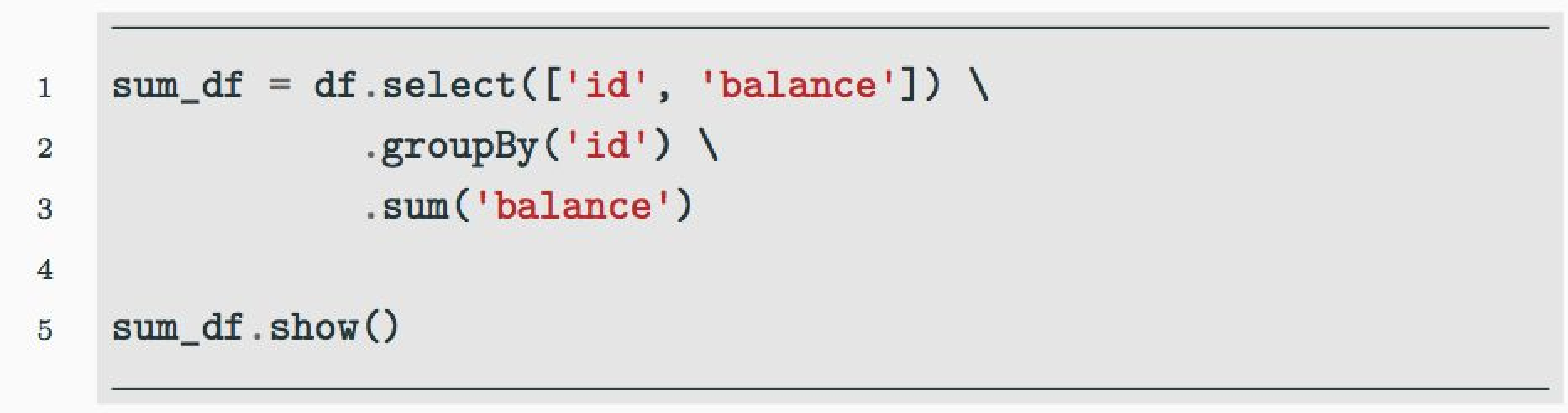


```
df = spark.read.json('s3n://spark-talk/balance-info.json',
                     inferSchema=True)
```

From here we can use either the DataFrame API or the SQL API to perform operations on our DataFrame object

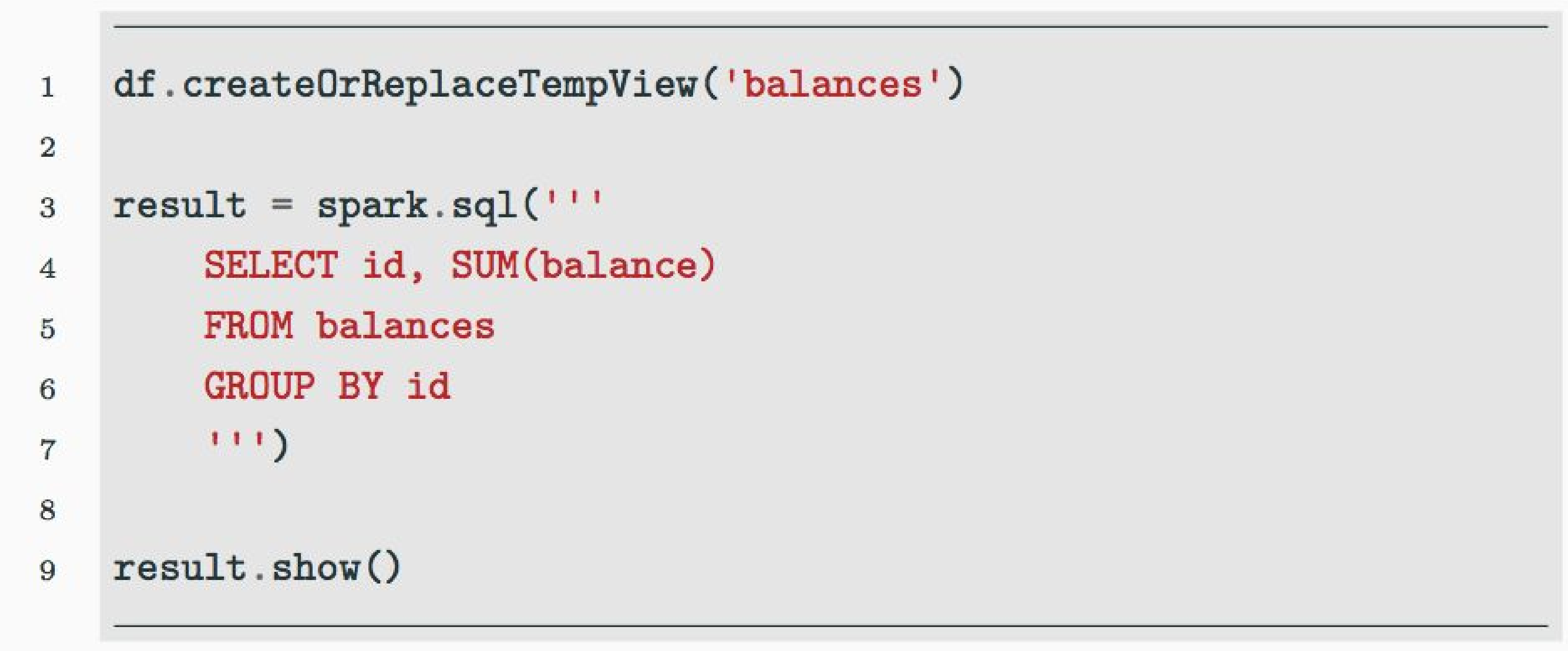
Example of DataFrame API

Let’s sum the balance for each user-id...



```
sum_df = df.select(['id', 'balance']) \
              .groupBy('id') \
              .sum('balance')

sum_df.show()
```

Example of SQL API



```
df.createOrReplaceTempView('balances')

result = spark.sql('''
    SELECT id, SUM(balance)
    FROM balances
    GROUP BY id
    ''')

result.show()
```

Spark ML

- One of the niceties of working within Spark are the Machine Learning models that are built out to take advantage of the distributed nature of Spark
- The `pyspark.ml` package provides DataFrame-based machine learning APIs to quickly assemble and configure ML pipelines, including tools such as:
  - ML Algorithms such as classification, regression, clustering, & collaborative filtering
  - Featurization
  - Pipelines
  - Persistence
- **NOTE:** You will also see reference to Spark MLlib
  - `pyspark.ml` is the MLlib DataFrame-based API
  - The RDD-based APIs contained in the `pyspark.mllib` package are now in maintenance mode and are planned to be deprecated in Spark 2.2

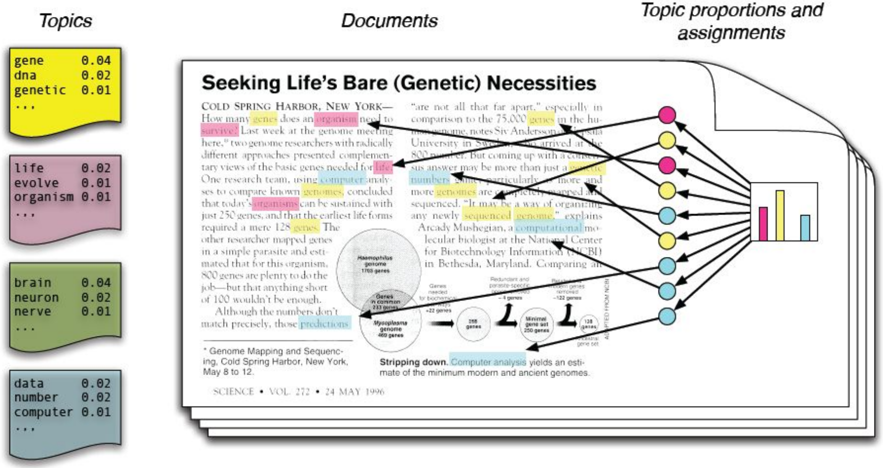
LDA: Latent Dirichlet Allocation (clustering)

With

- w: word
- d: document
- D: the corpus of docs
- k: a given number of topics

A **topic** is a distribution over words.
Each **document** is a mixture of corpus-wide topics.
Each **word** is drawn from these topics.

Find the k distributions that would likely “generate” every document d in D.



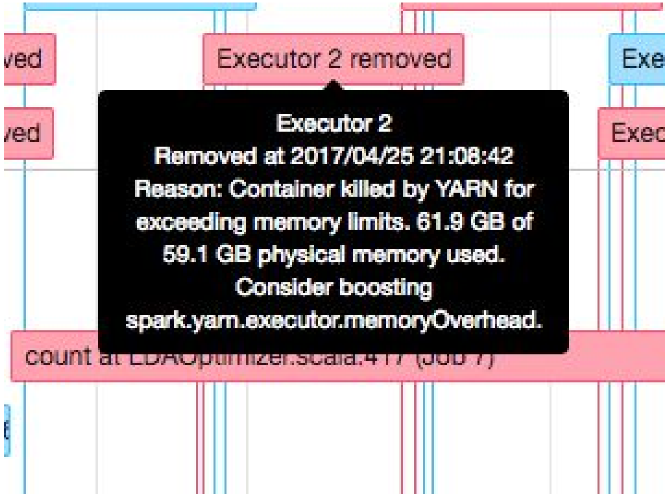
Keywords: LDA, Latent Semantic Analysis
Recommended reading: [Topic modeling for Humanists, a guided tour]

Demo...

LDA with Amazon Book Review Data

Pitfalls of Using Spark

- Under massive development (also a plus…)
- Can be a memory hog if jobs are not tuned well, resulting in frustrating out-of-memory errors
- Not all features are available in every API



Executor 2
Removed at 2017/04/25 21:08:42
Reason: Container killed by YARN for exceeding memory limits. 61.9 GB of 59.1 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead.

Questions!

- The code and data is freely available at github.com/ewellinger/spark-talk and the `spark-talk` s3 bucket
- Big shout out to Miles Erickson, Jeff Omhover, & many others from the Galvanize community
- If you want to learn more you should check out the Intro to Spark for Data Science Workshop (bit.ly/2pDLwLu) happening at the Galvanize Platte location on July 14th and 15th!