Microsoft

2018
Global Azure
BOOTCAMP

TEKLINKS

TSYS®

ALFA®
INSURANCE

GDH
GOVERNMENT SERVICES

The
National Security
Group

Sponsors for Azure Bootcamp

Predictive Analytics with Spark in Azure HDInsight

Ingyu Lee

Troy University

https://github.com/Azure-Bootcamp-Troy/SparksInHDInsight

Content

- What is HDInsight?
- What is Spark?
- What is RDD?
- What is DataFrames?
- What is Spark SQL?
- What is MLLib?

- References
  - **Microsoft DAT202.3x**: Spark in Azure HDInsight
  - Microsoft DAT202.2x: Real-Time Analytics with Hadoop in Azure HDInsight
  - Microsoft DAT202.1x: Big Data with Hadoop in Azure HDInsight
  - Microsoft Virtual Academy: Spark on HDInsight
  - UCB Sparks: CS100, CS105, CS110, CS120, CS190
  - UCSD Big Data Series (Coursera)
  - Yandex: Big Data Series (Coursera)

HDInsight – What is it?

A standard Apache Hadoop distribution offered as a managed service on Microsoft Azure

- Based on Hortonworks Data Platform (HDP)
- Provisioned as clusters on Azure that can run on Windows or Linux servers
- Offers capacity-on-demand, pay-as-you-go pricing model
- Integrates with:
  - Azure Blob Storage and Azure Data Lake Store for Hadoop File System (HDFS)
  - Azure Portal for management and administration
  - Visual Studio for application development tooling

In addition to the core, HDInsight supports the Hadoop ecosystem

What is Apache Spark?

General, open-source cluster computing engine

Well-suited for machine learning

- Fast iterative procedures
- Efficient communication primitives

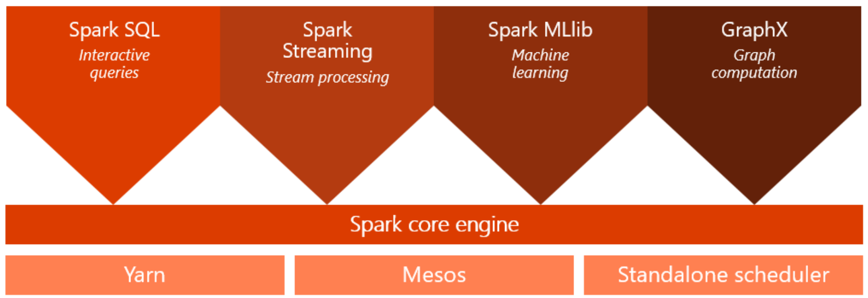
Simple and Expressive

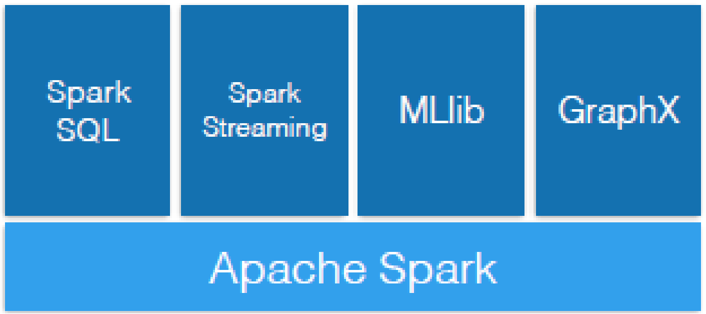
- APIs in Scala, Java, Python, R
- Interactive Shell

Integrated Higher-Level Libraries

Apache Spark: A unified framework

A unified, open source, parallel data processing framework for big data analytics

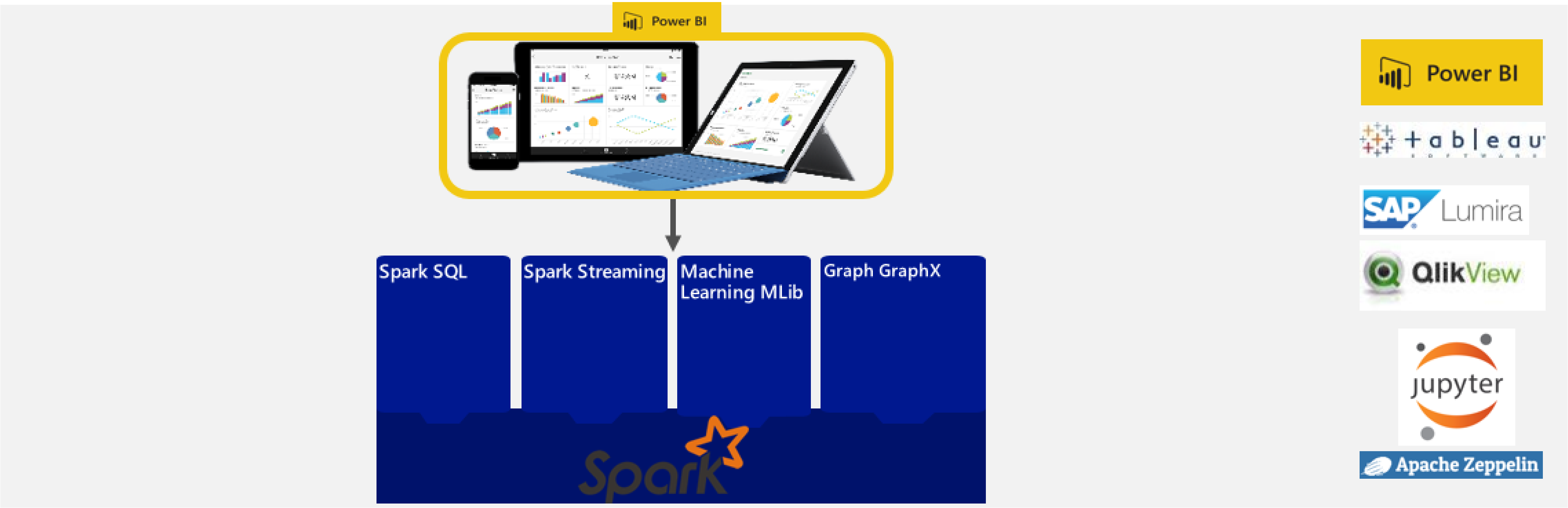




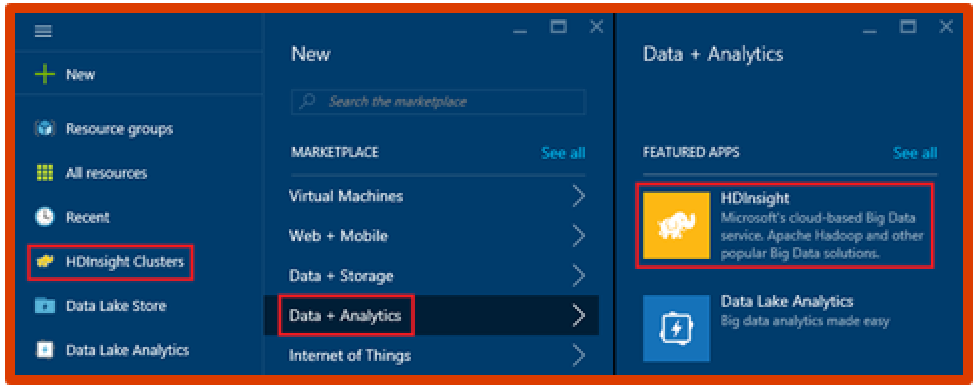
HDInsight supports Spark

In-memory processing on multiple workloads

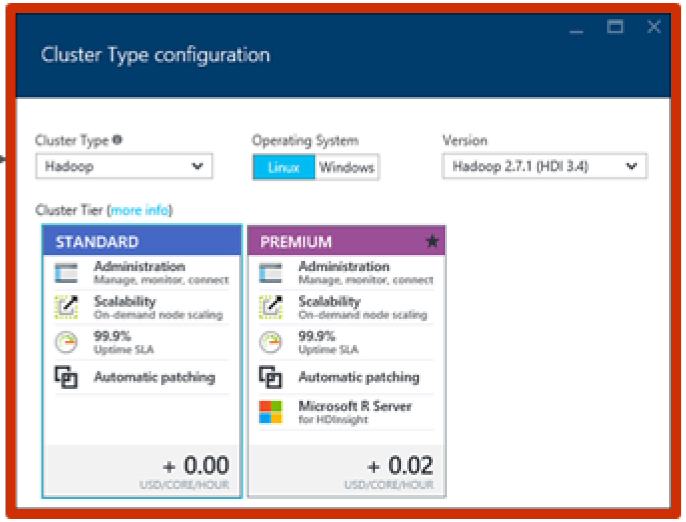
Single execution model for multiple tasks (SQL Query, Spark Streaming, Machine Learning, and Graph)
Processing up to 100x faster performance
Developer friendly (Java, Python, Scala)
BI tool of choice (Power BI, Tableau, Qlik, SAP)
Notebook experience (Jupyter/iPython, Zeppelin)



Demo: Creating an HDInsight Spark cluster

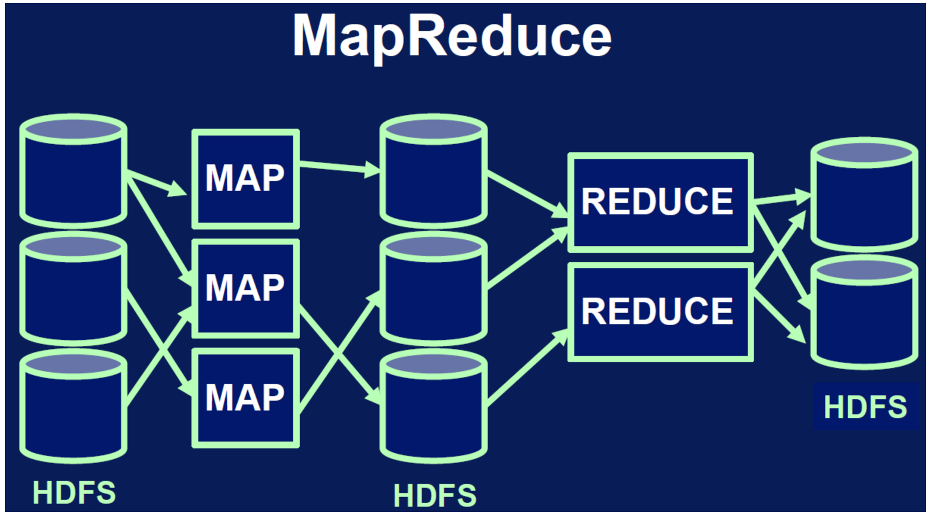


The Azure console lists all types of HDInsight clusters (HBase, Storm, and Spark) currently provisioned.



Apache Spark use cases

Hadoop vs. Spark



Force your pipeline into Map and Reduce steps

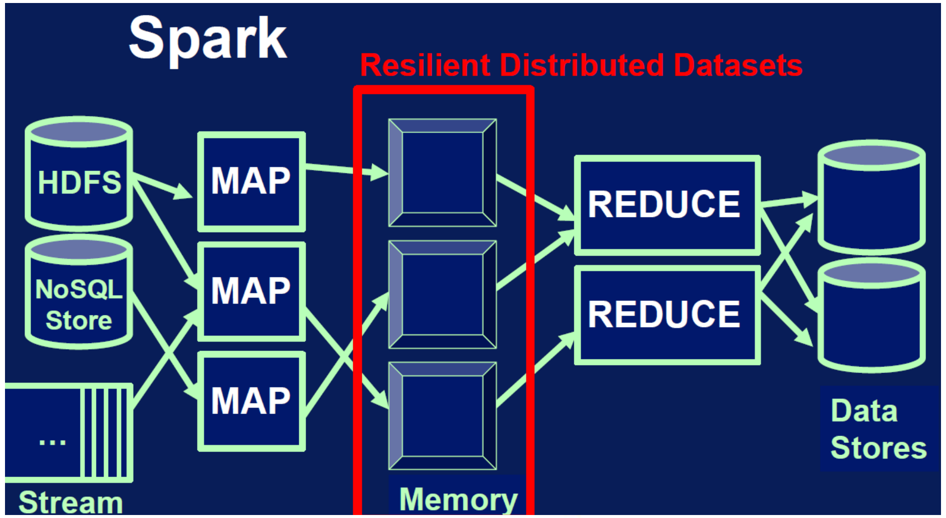
Other workflows? i.e. join, filter, map-reduce-map

Read from disk for each MapReduce job

Iterative algorithms? i.e. machine learning

Only native JAVA programming interface

Other languages? Interactivity?



- New framework: same features of MapReduce and more
- Capable of reusing Hadoop ecosystem, e.g. HDFS, YARN…
- Born at UC Berkeley

Other workflows? i.e. join, filter, map-reduce-map

~20 highly efficient distributed operations, any combination of them

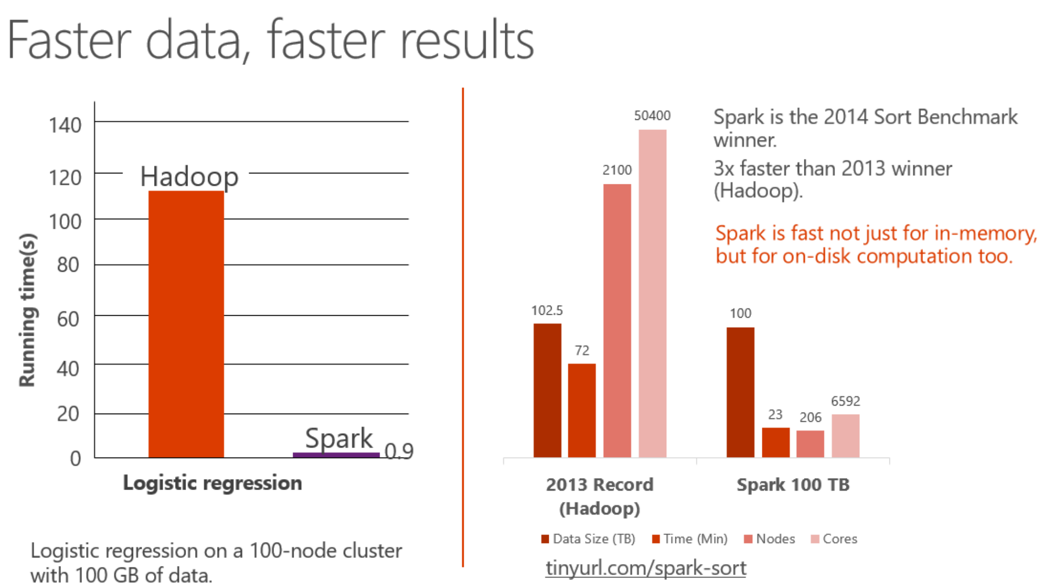
Interactivity? Other languages?

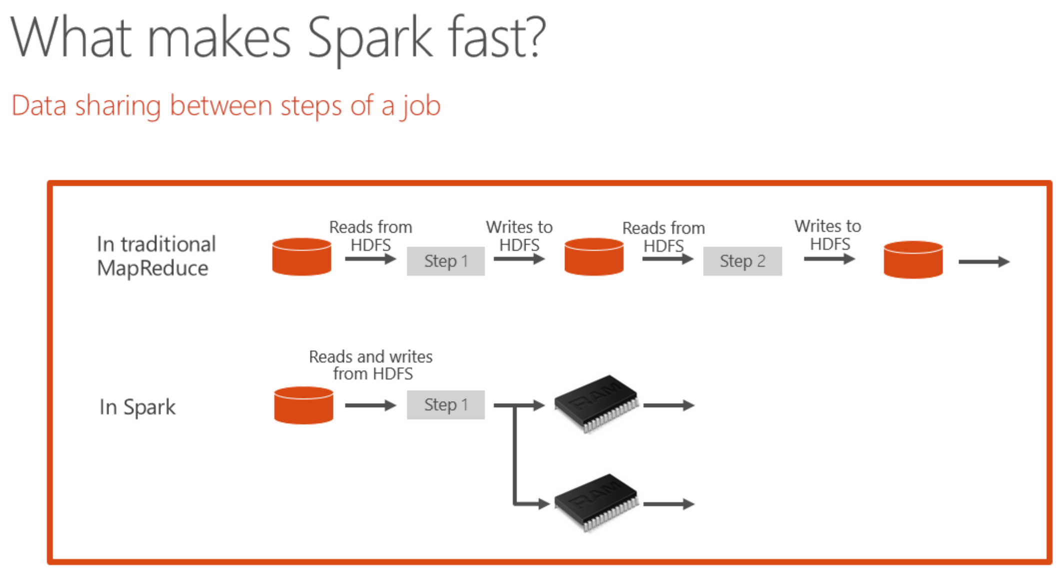
Native Python, Scala (, R) interface. Interactive shells.

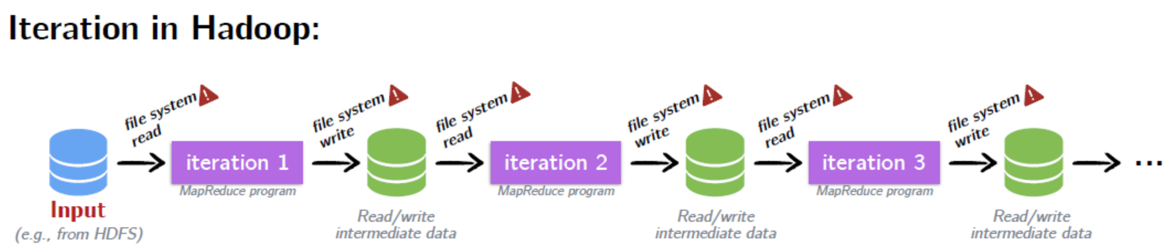
Iterative algorithms? i.e. machine learning

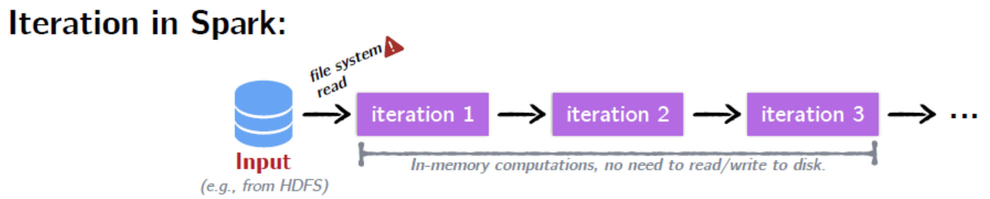
in-memory caching of data, specified by the user

Why sparks is faster?



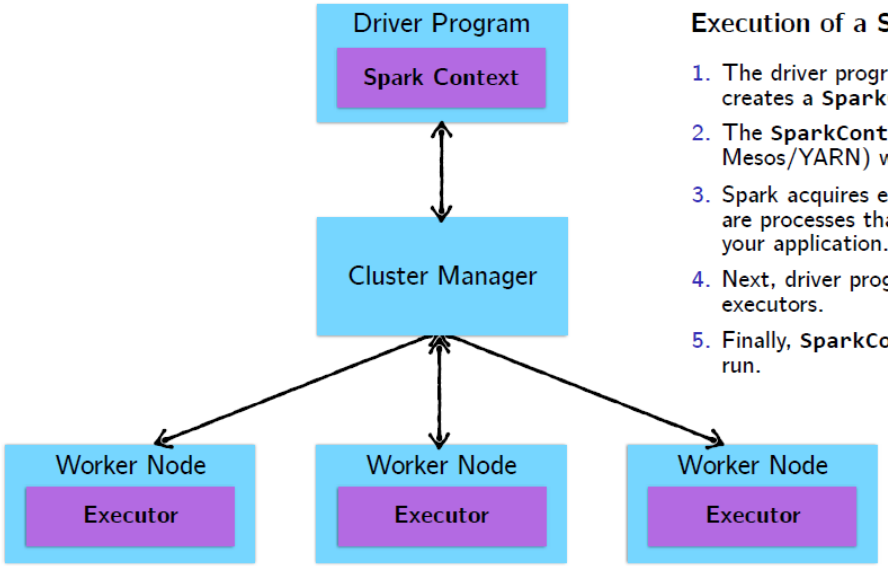






Spark cluster architecture

- Distributed processing architecture consists of:
  - A *driver program*
  - One or more *worker nodes*
- The driver program uses a spark context to connect to the cluster…
- …and uses worker nodes to perform operations on RDDs



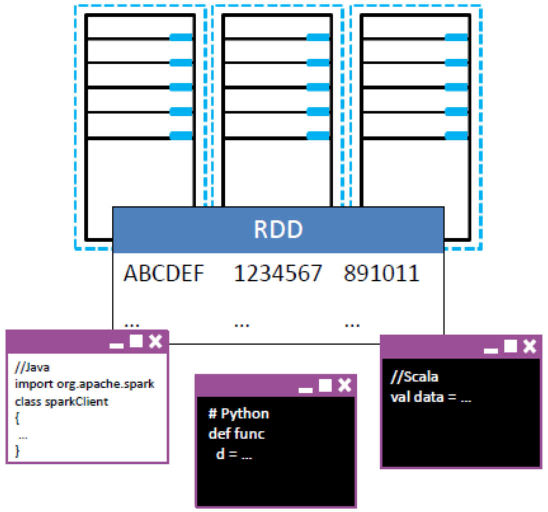
Execution of a Spark program:

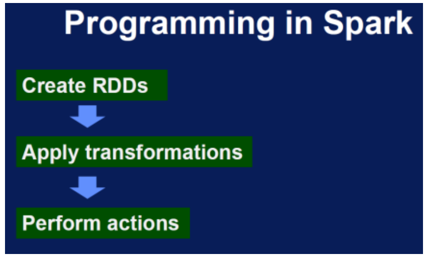
1. The driver program runs the Spark application, which creates a `SparkContext` upon start-up.
2. The `SparkContext` connects to a cluster manager (e.g., Mesos/YARN) which allocates resources.
3. Spark acquires executors on nodes in the cluster, which are processes that run computations and store data for your application.
4. Next, driver program sends your application code to the executors.
5. Finally, `SparkContext` sends tasks for the executors to run.

What is RDD?

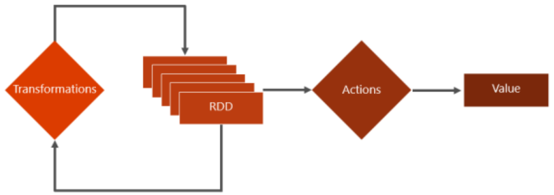
- The core abstraction for data in Spark is the *resilient distributed dataset* (RDD)
- An RDD represents a collection of items that can be distributed across compute nodes
- APIs for working with RDDs are provided for Java, Python, and Scala
  - HDInsight distribution includes Python and Scala shells

Dataset	Distributed	Resilient
Data storage created from: HDFS, S3, HBase, JSON, text, Local hierarchy of folders	Distributed across the cluster of machines	Recover from errors, e.g. node failure, slow processes
Or created transforming another RDD	Divided in partitions, atomic chunks of data	Track history of each partition, re-run





RDDs: Transformations and actions



RDD Operations

- To create a Spark Context:
  1. Create a configuration for your cluster and application
  2. Use the configuration to create a context

  (Spark shells have one pre-created)
- To create an RDD
  - Load from a source
    - Text file, JSON, XML, etc.
  - Parallelize a collection

Cluster URL

```
cfg = SparkConf().setMaster("local").setAppName("App")
sc = SparkContext(conf = cfg)
```

Your application name

Path to file
(default text delimiter is newline)

```
txtRDD = sc.textFile("/data/tweets.txt")
```

```
lstRDD = sc.parallelize(["A", "B", "C"])
```

List

RDD Transformations

Common Transformations:

- filter: Creates a filtered RDD
- flatMap: Applies a function to each element that returns multiple elements into a new RDD
- map: Applies a function to each element that returns an element in a new RDD
- reduceByKey: Aggregates values for each key in a key-value pair RDD

```
txt = sc.parallelize(["the owl and the pussycat",
                    "went to sea"])
```

{["the owl and the pussycat"], ["went to sea"]}

```
owlTxt = txt.filter(lambda t: "owl" in t)
```

{["the owl and the pussycat"]}

```
words = owlTxt.flatMap(lambda t: t.split(" "))
```

{["the"], ["owl"], ["and"], ["the"], ["pussycat"]}

```
kv = words.map(lambda key: (key, 1))
```

{["the",1], ["owl",1], ["and",1], ["the",1], ["pussycat",1]}

```
counts = kv.reduceByKey(lambda a, b: a + b)
```

{["the",2], ["owl",1], ["and",1], ["pussycat",1]}

RDD Actions

Common Actions:

- **reduce**: Aggregates the elements of an RDD using a function that takes two arguments
- **count**: Returns the number of elements in the RDD
- **first**: Returns the first element in the RDD
- **collect**: Returns the RDD as an array to the driver program
- **saveAsTextFile**: Saves the RDD as a text file in the specified path

```
nums = sc.parallelize([1, 2, 3, 4])
```

{[1], [2], [3], [4]}

```
nums.reduce(lambda x, y: x + y)
```

10

```
nums.count()
```

4

```
nums.first()
```

1

```
nums.collect()
```

[1, 2, 3, 4]

```
nums.saveAsTextFile("/results")
```

/results/part-00000

Demo: Working with RDDs in Python

- Most operations involve passing a function to a transformation or action
- Functions can be:
  - Explicitly declared
  - Passed inline
    - Python uses **lambda** keyword
    - Scala uses => syntax
    - Java uses function classes or lambdas (Java 8)

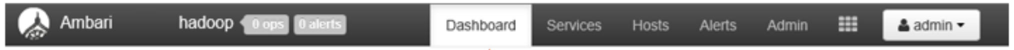
```
RDD.filter(function)
```

```
def containsMSTag(txt):
    return "#ms" in txt

msTwts = txtRDD.filter(containsMSTag)
```

```
#Python
msTwts = txtRDD.filter(lambda txt: "#ms" in txt)
```

```
//Scala
val msTwts = txtRDD.filter(txt => txt.contains("#ms"))
```

Ambari Dashboard

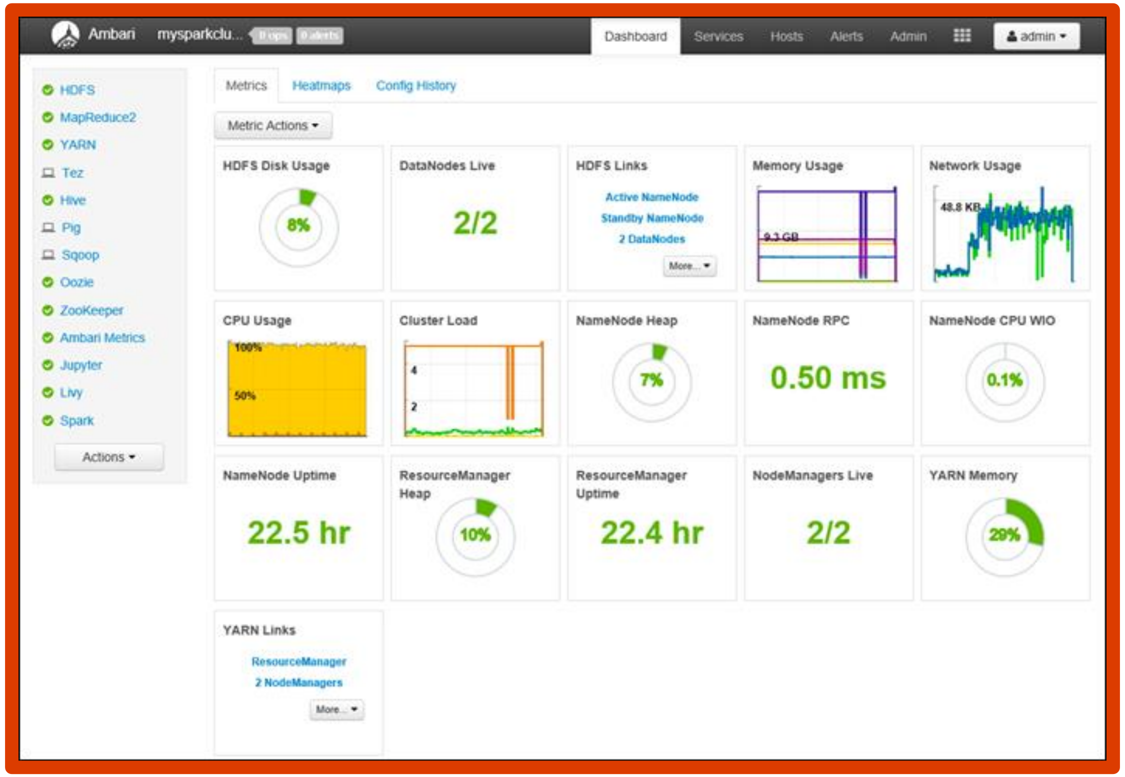


The Ambari Dashboard provides links to access:

Ambari web UI – monitors Spark clusters

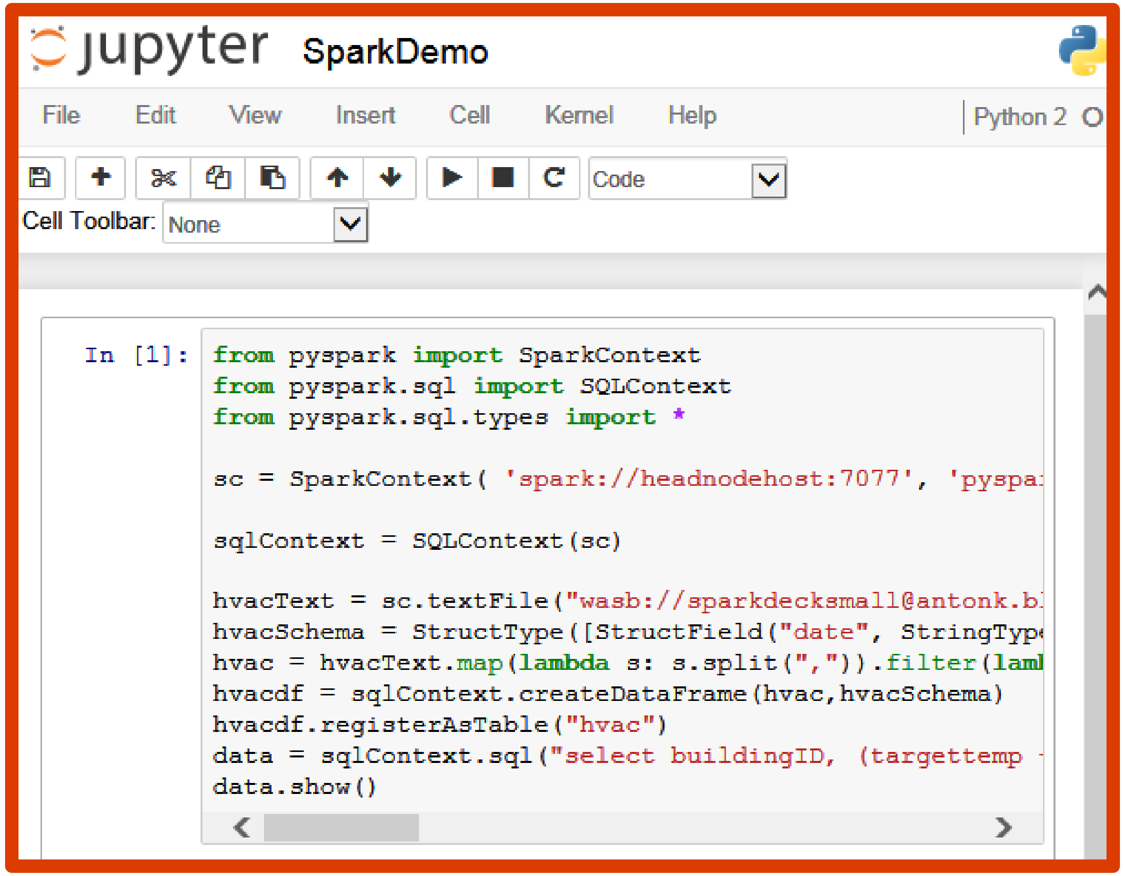
Hosts (Resource Manager) – controls amount of resources allocated to various Spark cluster components

Notebooks – interactive web-based tools to develop and run Spark programs



HDInsight Spark: Jupyter notebooks

- Web-based interactive consoles for
  - Experimentation
  - Collaboration
- Spark HDInsight clusters include Jupyter
  - Interactive Python
  - Interactive Scala

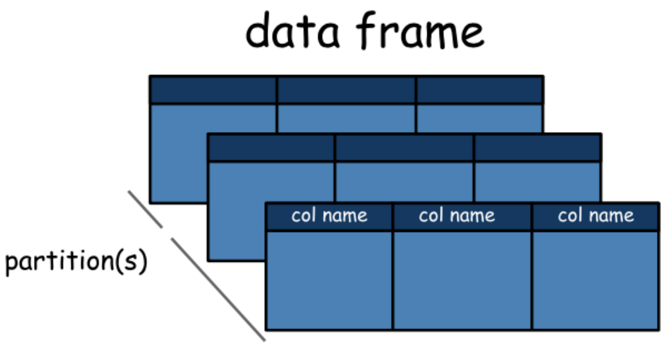


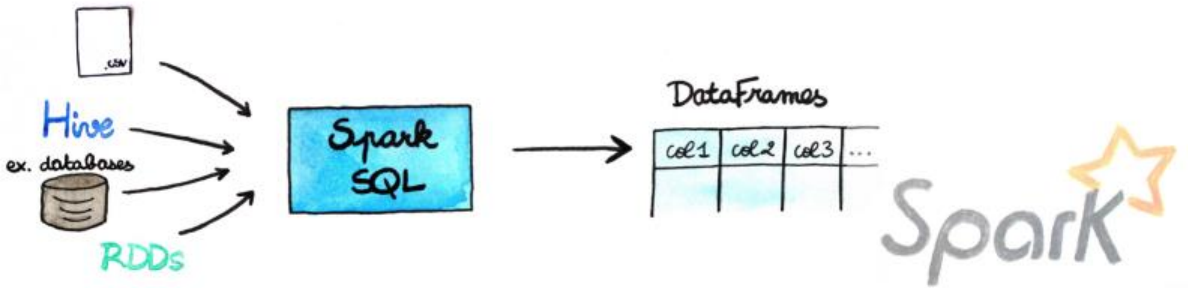
What is DataFrames?

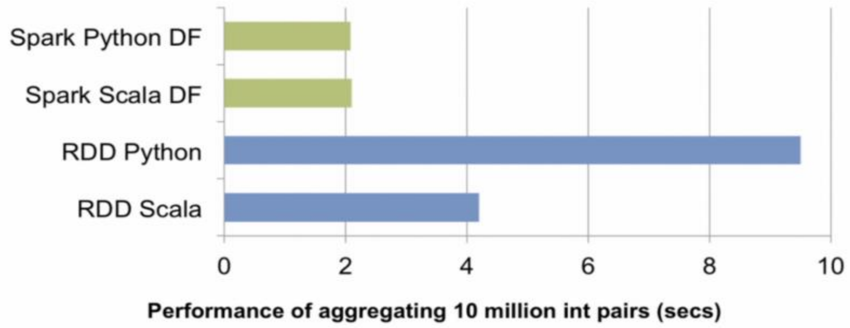
DataFrames

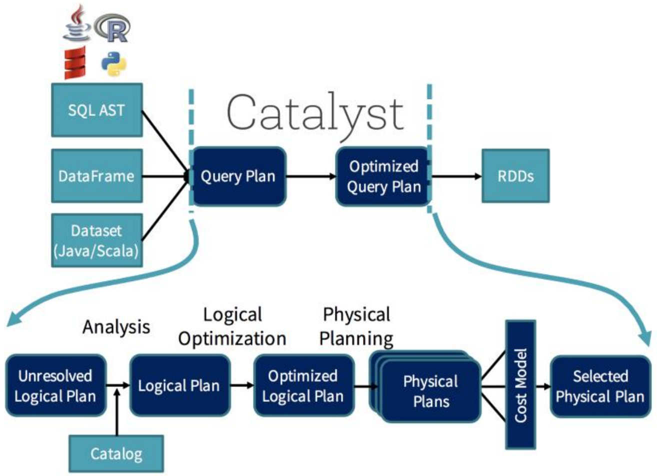
Distributed Data organized as named columns

Look just like a table in relational databases



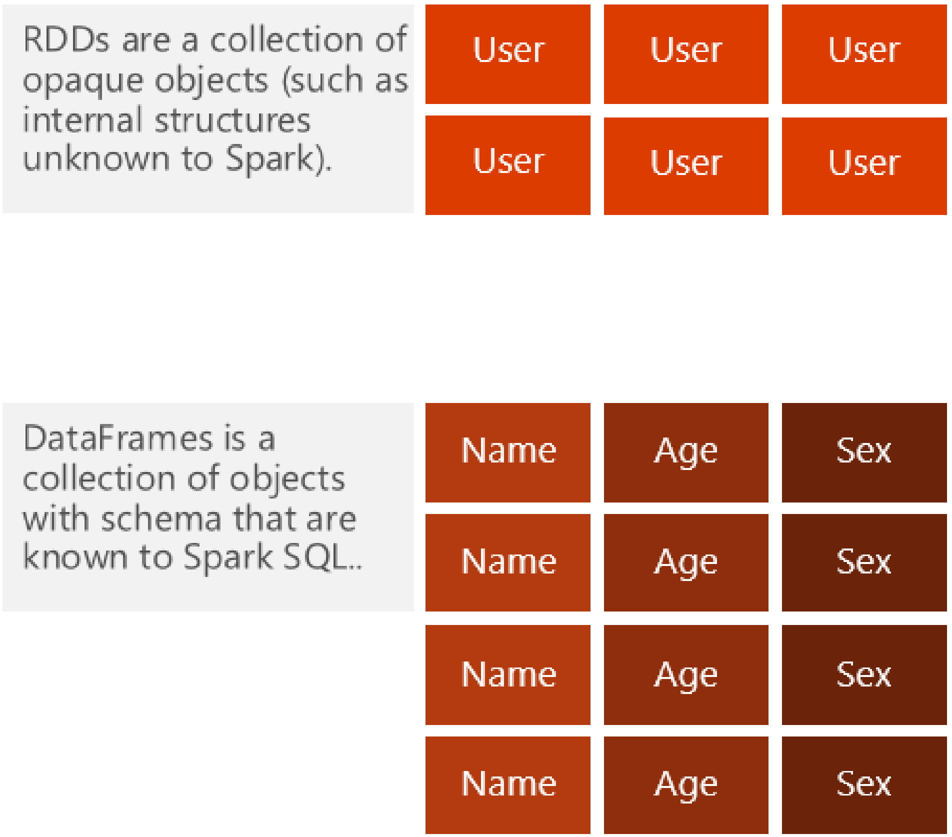






Core abstraction: DataFrames

- A distributed collection of data organized into named columns.
- Similar to RDDs with schema.
- Conceptually equivalent to tables in relational database, or to DataFrames in R/Python.
- With domain-specific functions designed for common tasks:
    - Metadata
    - Sampling
    - Project, filter, aggregation, and join
    - UDFs



DataFrame Operations

DataFrames provide a domain-specific language for structured data manipulation in Scala, Java, and Python.

```
>>> val df = sqlContext.jsonFile(”somejasonfile.json“)

>>> df.show()                                     // Show the contents of the DataFrame

>>> df.printSchema()                              // Print the schema in a tree format

>>> df.select(“name”).show()                      // Select and show the name columns

>>> df.select(df(”name”),df (“age”) +1).show()    // Select all but increment the age by 1

>>> df.filter(df(”age”) > 21).show()              // Select people older than 21

>>> df.groupBy(“age”).count().show()              // Count people by age
```

Tables and Queries

A DataFrame can be registered as a table that can then be used in SQL queries.

```
// First create a DataFrame from JSON file.
>>> val df = sqlContext.jsonFile(”Users.json“)

// Register the DataFrame as a temporary table. Temp tables exist only during lifetime of this SQLContext
instance.
 >>> val usertable = sqlContext.registerDataFrameAsTable(df, “UserTable”)

// Alternatively, execute a SQL query on the table. The query returns a DataFrame.
>>> val teenagers = sqlContext.sql(“select Age as Years from UserTable where age > 13 and age <= 19”)
```

What is Spark SQL?

Spark SQL

- Enables querying structured and unstructured data through Spark
- Provides a common query language
- Has APIs for Scala, Java and Python to convert results into RDDs

Relational Operations

Perform Relational Processing such as Declarative Queries

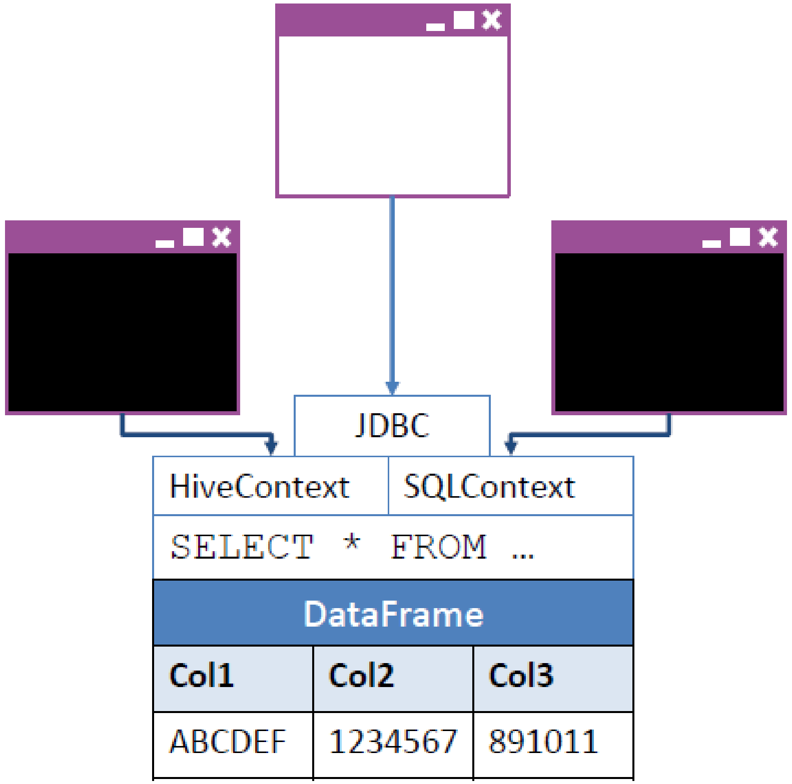
Embed SQL queries inside Spark Programs

Install pandas to see the SQL magic on jupyter

```
sudo -HE /usr/bin/anaconda/bin/conda install pandas
```

Spark SQL

- Spark SQL provides a query interface for structured data
- DataFrames are used to abstract RDDs and define a schema
- There are two API entry points:
  - HiveContext
  - SQLContext
- Client applications can connect to Spark SQL using JDBC



Spark SQL

How to go Relational in Spark ?

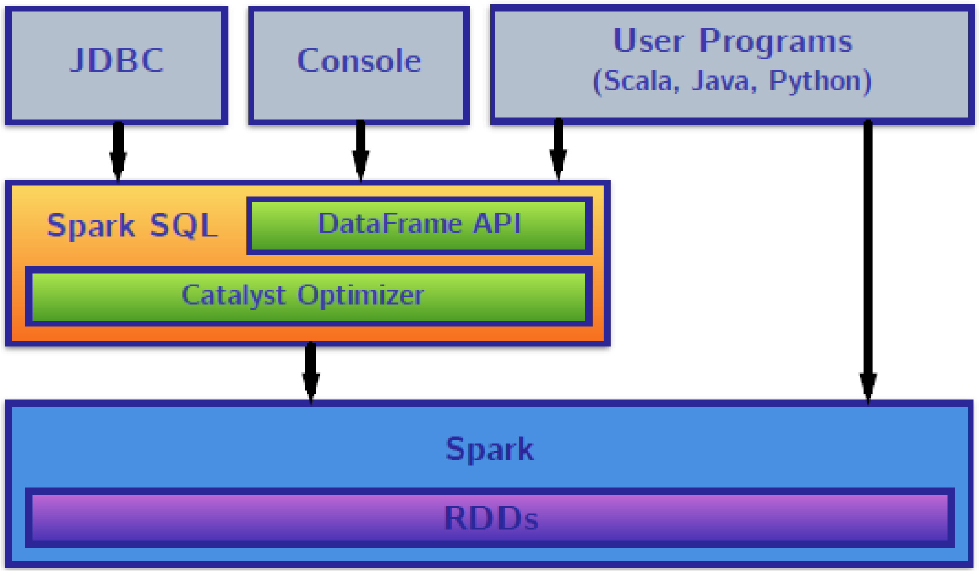
Step 1: Create a SQLContext

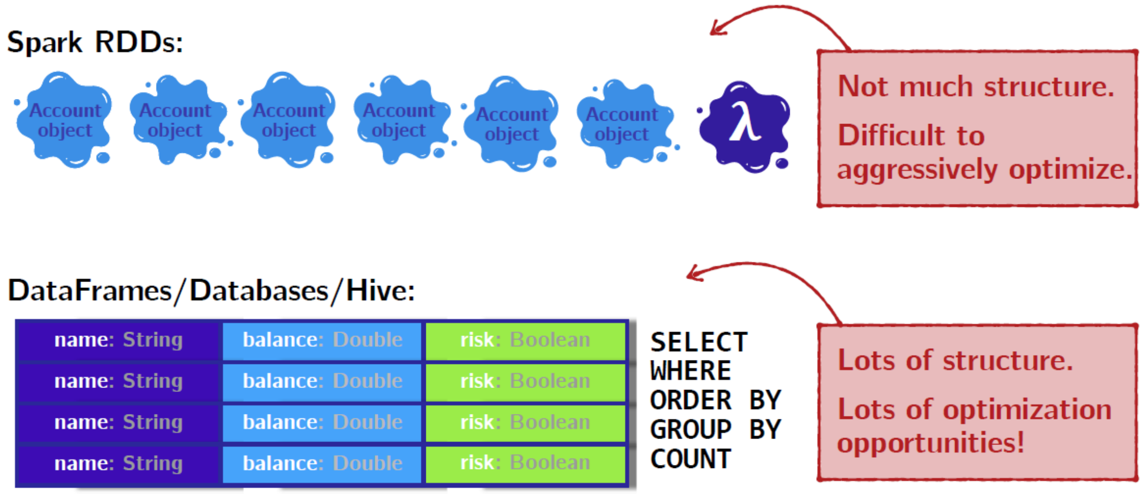
```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
```

How to go Relational in Spark ?

Create a DataFrame from

- an existing RDD
- a Hive table
- data sources





Spark SQL

JSON → DataFrame

```
# Read
df = sqlContext.read.json("/filename.json")

# Display
df.show()
```

DataFrames are just like tables

http://spark.apache.org/

```
# Show the content of the DataFrame
df.show()

# Print the schema
df.printSchema()

# Select only the "X" column
df.select("X").show()

# Select everybody, but increment the discount by 5%
df.select(df["name"], df["discount"] + 5).show()

# Select people height greater than 4.0 ft
df.filter(df["height"] > 4.0).show()

# Count people by zip
df.groupBy("zip").count().show()
```

RDD of Row objects → DataFrame

```
# Read
from pyspark.sql import SQLContext, Row
sqlContext = SQLContext(sc)

# Load a text file and convert each line to a Row.
lines = sc.textFile("filename.txt")
cols = lines.map(lambda l: l.split(","))
data = cols.map(lambda p: Row(name=p[0], zip=int(p[1])))

# Create DataFrame
df = sqlContext.createDataFrame(data)

# Register the DataFrame as a table
df.registerTempTable(”table")

# Run SQL
Output = sqlContext.sql("SELECT * FROM table WHERE …")
```

Spark SQL

Relational on Spark

Connect to variety of databases

Deploy business intelligence tools over Spark

What is Machine Learning?

A Definition

Constructing and studying methods that learn from and make predictions on data

Broad area involving tools and ideas from various domains

- Computer Science
- Probability and Statistics
- Optimization
- Linear Algebra

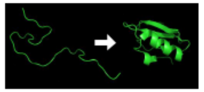
Some Examples

Face recognition

Link prediction

Text or document classification, e.g., spam detection

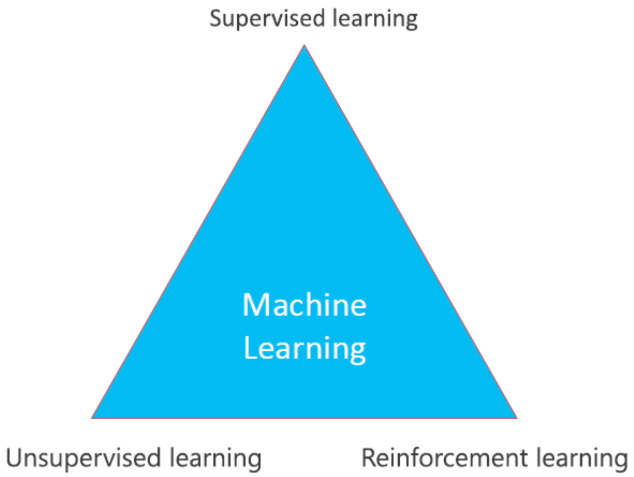
Protein structure prediction

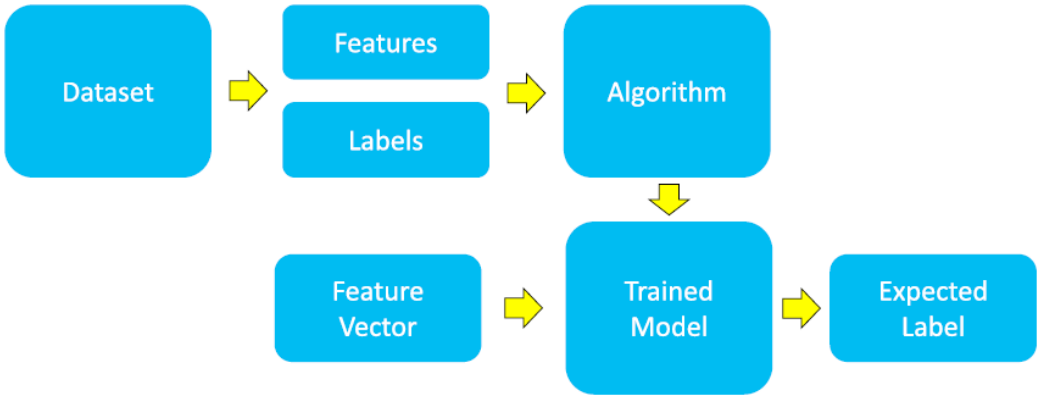


Games, e.g., Backgammon or Jeopardy

How does Machine Learning work?

- Data labels
- Direct feedback
- Predict outcome





Supervised learning. Learning from labeled observations

- Labels ‘teach’ algorithm to learn mapping from observations to labels

Unsupervised learning. Learning from unlabeled observations

- Learning algorithm must find latent structure from features alone
- Can be goal in itself (discover hidden patterns, exploratory data analysis)
- Can be means to an end (preprocessing for supervised task)

Observations. Items or entities used for learning or evaluation, e.g., emails

Features. Attributes (typically numeric) used to represent an observation, e.g., length, date, presence of keywords

Labels. Values / categories assigned to observations, e.g., *spam*, *not-spam*

Training and Test Data. Observations used to train and evaluate a learning algorithm, e.g., a set of emails along with their labels

- Training data is given to the algorithm for training
- Test data is withheld at train time

What is MLlib?

Spark MLlib

- Scalable machine learning library
- Provides distributed implementations of common machine learning algorithms and utilities
- Has APIs for Scala, Java, Python, and R

MLlib Algorithms & Techniques

- Machine Learning
  - Classification, regression, clustering, etc.
  - Evaluation metrics
- Statistics
  - Summary statistics, sampling, etc.
- Utilities
  - Dimensionality reduction, transformation, etc.

How to.. Machine Learning in Apache Spark?

- All primitives in Spark Machine Learning are *Vectors*
- *Features* are represented by a Vector
- Vectors can contain other Vectors and so be Dense or Sparse
- Spark uses *LabeledPoints* to encapsulate a Vector and a Label
- RDDs are transformed into Vectors through map functions

MLlib Terminology

ML: Transformer

- A *Transformer* is a class which can transform one DataFrame into another DataFrame
- A Transformer implements `transform()`

- Examples
  - HashingTF
  - LogisticRegressionModel
  - Binarizer

```
val tokenizer = new Tokenizer().setInputCol("text").setOutputCol("words")
val hashingTF = new
HashingTF().setNumFeatures(1000).setInputCol(tokenizer.getOutputCol).s
etOutputCol("features")
val lr = new LogisticRegression().setMaxIter(10).setRegParam(0.01)
val pipeline = new Pipeline().setStages(Array(tokenizer, hashingTF, lr))

val model = pipeline.fit(training)

val modelem = model.transform(test).select("id", "label", "text",
"probability", "prediction")
```

ML: Estimator

- An *Estimator* is a class which can take a DataFrame and produce a Transformer
- An Estimator implements `fit()`

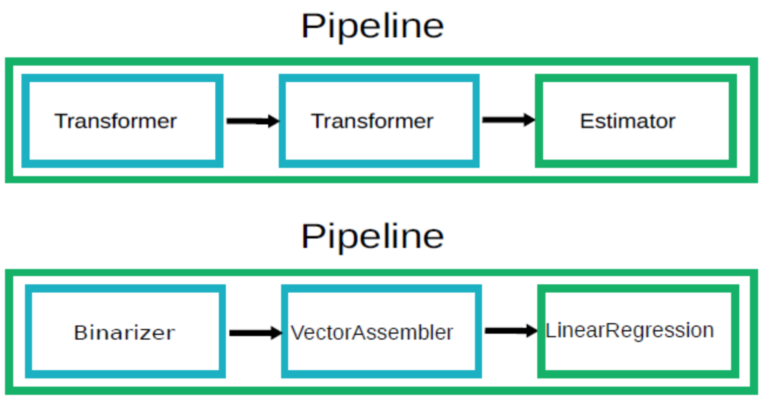
- Examples
  - LogisticRegression
  - StandardScaler
  - Pipeline

```
val tokenizer = new Tokenizer().setInputCol("text").setOutputCol("words")
val hashingTF = new
HashingTF().setNumFeatures(1000).setInputCol(tokenizer.getOutputCol).s
etOutputCol("features")
val lr = new LogisticRegression().setMaxIter(10).setRegParam(0.01)
val pipeline = new Pipeline().setStages(Array(tokenizer, hashingTF, lr))

val model = pipeline.fit(training)

val modelem = model.transform(test).select("id", "label", "text",
"probability", "prediction")
```

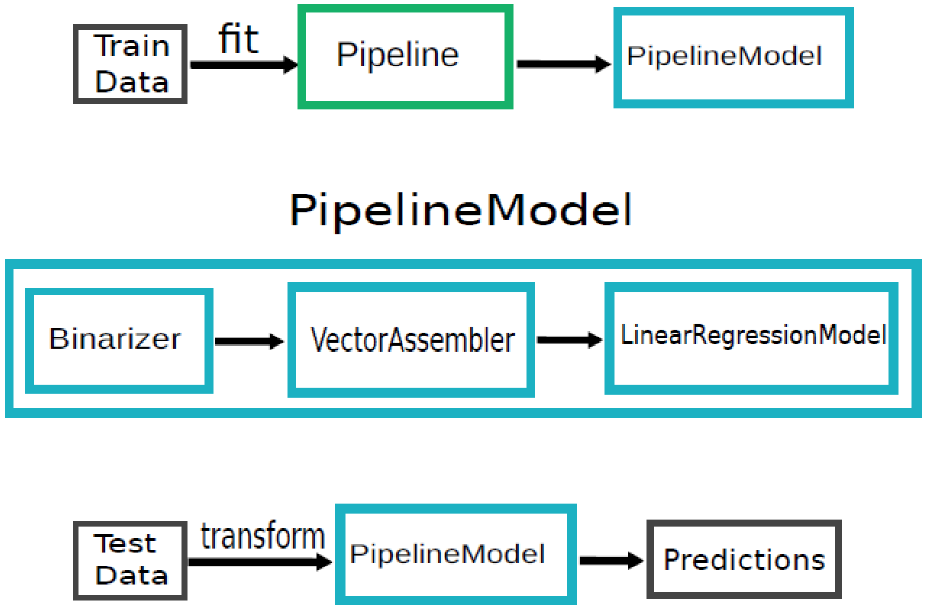
MLlib Terminology

ML: Pipelines

A *Pipeline* is an estimator that contains stages representing a resusable workflow. Pipeline stages can be either estimators or transformers.



ML: PipelineModel



Examples: Classification

Import Spark SQL and Spark ML Libraries

First, import the libraries you will need:

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

from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import VectorAssembler
```

You will load this data into a DataFrame and display it.

Load Source Data

```
csv = spark.read.csv('wasb:///data/flights.csv', inferSchema=True, header=True)
csv.show()
```

```
+----------+---------+-------+---------------+-------------+--------+--------+
|DayofMonth|DayOfWeek|Carrier|OriginAirportID|DestAirportID|DepDelay|ArrDelay|
+----------+---------+-------+---------------+-------------+--------+--------+
|        19|        5|     DL|          11433|        13303|      -3|       1|
|        19|        5|     DL|          14869|        12478|       0|      -8|
```

Prepare the Data

```
data = csv.select("DayofMonth", "DayOfWeek", "OriginAirportID", "DestAirportID", "DepDelay",
                   ((col("ArrDelay") > 15).cast("Int").alias("Late")))
data.show()
```

```
+----------+---------+---------------+-------------+--------+----+
|DayofMonth|DayOfWeek|OriginAirportID|DestAirportID|DepDelay|Late|
+----------+---------+---------------+-------------+--------+----+
|        19|        5|          11433|        13303|      -3|   0|
|        19|        5|          14869|        12478|       0|   0|
|        19|        5|          14057|        14869|      -4|   0|
```

Split the Data

```
splits = data.randomSplit([0.7, 0.3])
train = splits[0]
test = splits[1]
train_rows = train.count()
test_rows = test.count()
print "Training Rows:", train_rows, " Testing Rows:", test_rows
```

```
assembler = VectorAssembler(inputCols = ["DayofMonth", "DayOfWeek", "OriginAirportID", "DestAirportID", "DepDelay"],
                             outputCol="features")
training = assembler.transform(train).select(col("features"), col("Late").alias("label"))
training.show()
```

```
+--------------------+-----+
|            features|label|
+--------------------+-----+
|[1.0,1.0,10140.0,...|    0|
```

Prepare the Training Data

Examples: Classification

Train a Classification Model

```
lr = LogisticRegression(labelCol="label",featuresCol="features",maxIter=10,regParam=0.3)
model = lr.fit(training)
print "Model trained!"
```

```
Model trained!
```

Prepare the Testing Data

```
testing = assembler.transform(test).select(col("features"), col("Late").alias("trueLabel"))
testing.show()
```

```
+--------------------+---------+
|            features|trueLabel|
+--------------------+---------+
|[1.0,1.0,10140.0,...|        0|
|[1.0,1.0,10140.0,...|        0|
```

Test the Model

```
prediction = model.transform(testing)
predicted = prediction.select("features", "prediction", "probability", "trueLabel")
predicted.show(100)
```

```
+--------------------+----------+--------------------+---------+
|            features|prediction|         probability|trueLabel|
+--------------------+----------+--------------------+---------+
|[1.0,1.0,10140.0,...|       0.0|[0.83121028425635...|        0|
|[1.0,1.0,10140.0,...|       0.0|[0.82357817040192...|        0|
```

Example: Regression

Import Spark SQL and Spark ML Libraries

First, import the libraries you will need:

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

from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorAssembler
```

```
Starting Spark application
```

Load Source Data

```
csv = spark.read.csv('wasb:///data/flights.csv', inferSchema=True, header=True)
csv.show()
```

```
+----------+---------+-------+---------------+-------------+--------+--------+
|DayofMonth|DayOfWeek|Carrier|OriginAirportID|DestAirportID|DepDelay|ArrDelay|
+----------+---------+-------+---------------+-------------+--------+--------+
|        19|        5|     DL|          11433|        13303|      -3|       1|
|        19|        5|     DL|          14869|        12478|       0|      -8|
|        19|        5|     DL|          1405
```

Prepare the Data

```
data = csv.select("DayofMonth", "DayOfWeek", "OriginAirportID", "DestAirportID", "DepDelay", "ArrDelay")
data.show()
```

```
+----------+---------+---------------+-------------+--------+--------+
|DayofMonth|DayOfWeek|OriginAirportID|DestAirportID|DepDelay|ArrDelay|
+----------+---------+---------------+-------------+--------+--------+
|        19|        5|          11433|        13303|      -3|       1|
```

Split the Data

```
splits = data.randomSplit([0.7, 0.3])
train = splits[0]
test = splits[1]
train_rows = train.count()
test_rows = test.count()
print "Training Rows:", train_rows, " Testing Rows:", test_rows
```

```
Training Rows: 1891270  Testing Rows: 810948
```

Prepare the Training Data

```
assembler = VectorAssembler(inputCols = ["DayofMonth", "DayOfWeek", "OriginAirportID", "DestAirportID", "DepDelay"],
                            outputCol="features")
training = assembler.transform(train).select(col("features"), (col("ArrDelay").cast("Int").alias("label")))
training.show()
```

```
+--------------------+-----+
|            features|label|
+--------------------+-----+
|[1.0,1.0,10140.0,...|  -11|
```

Example: Regression

```
lr = LinearRegression(labelCol="label",featuresCol="features", maxIter=10, regParam=0.3)
model = lr.fit(training)
print "Model trained!"
```

```
Model trained!
```

Train a Regression Model

```
testing = assembler.transform(test).select(col("features"), (col("ArrDelay")).cast("Int").alias("trueLabel"))
testing.show()
```

```
+--------------------+---------+
|            features|trueLabel|
+--------------------+---------+
|[1.0,1.0,10140.0,...|       -5|
|[1.0,1.0,10140.0,...|      -25|
```

Prepare the Testing Data

```
prediction = model.transform(testing)
predicted = prediction.select("features", "prediction", "trueLabel")
predicted.show()
```

```
+--------------------+-------------------+---------+
|            features|         prediction|trueLabel|
+--------------------+-------------------+---------+
|[1.0,1.0,10140.0,...|-3.7579519642834542|       -5|
|[1.0,1.0,10140.0,...| -9.743936006015753|      -25|
```

Test the Model

Example: Pipeline

Define the Pipeline

```
strIdx = StringIndexer(inputCol = "Carrier", outputCol = "CarrierIdx")
catVect = VectorAssembler(inputCols = ["CarrierIdx", "DayofMonth", "DayOfWeek", "OriginAirportID", "DestAirportID"],
                          outputCol="catFeatures")
catIdx = VectorIndexer(inputCol = catVect.getOutputCol(), outputCol = "idxCatFeatures")
numVect = VectorAssembler(inputCols = ["DepDelay"], outputCol="numFeatures")
minMax = MinMaxScaler(inputCol = numVect.getOutputCol(), outputCol="normFeatures")
featVect = VectorAssembler(inputCols=["idxCatFeatures", "normFeatures"], outputCol="features")
dt = DecisionTreeClassifier(labelCol="label", featuresCol="features")
pipeline = Pipeline(stages=[strIdx, catVect, catIdx, numVect, minMax, featVect, dt])
```

```
piplineModel = pipeline.fit(train)
print "Pipeline complete!"
```

Run the Pipeline as an Estimator

```
prediction = piplineModel.transform(test)
predicted = prediction.select("features", "prediction", "trueLabel")
predicted.show(100, truncate=False)
```

Test the Pipeline Model

More Example: Text Analysis

Define the Pipeline

The pipeline for the model consist of the following stages:

- A Tokenizer to split the tweets into individual words.
- A StopWordsRemover to remove common words such as "a" or "the" that have little predictive value.
- A HashingTF class to generate numeric vectors from the text values.
- A LogisticRegression algorithm to train a binary classification model.

```
tokenizer = Tokenizer(inputCol="SentimentText", outputCol="SentimentWords")
swr = StopWordsRemover(inputCol=tokenizer.getOutputCol(), outputCol="MeaningfulWords")
hashTF = HashingTF(inputCol=swr.getOutputCol(), outputCol="features")
lr = LogisticRegression(labelCol="label", featuresCol="features", maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, swr, hashTF, lr])
```

:

```
piplineModel = pipeline.fit(train)
print "Pipeline complete!"
```

Run the Pipeline as an Estimator

Test the Pipeline Model

```
prediction = piplineModel.transform(test)
predicted = prediction.select("SentimentText", "prediction", "trueLabel")
predicted.show(100, truncate = False)
```

More Example: Clustering

Create the K-Means Model

```
assembler = VectorAssembler(inputCols = ["Age", "MaritalStatus", "IncomeRange", "Gender", "TotalChildren",
                                         "ChildrenAtHome", "Education", "Occupation", "HomeOwner", "Cars"], outputCol="features")
train = assembler.transform(customers)

kmeans = KMeans(featuresCol=assembler.getOutputCol(), predictionCol="cluster", k=5, seed=0)
model = kmeans.fit(train)
print "Model Created!"
```

```
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

Get the Cluster Centers

```
prediction = model.transform(train)
prediction.groupBy("cluster").count().orderBy("cluster").show()
```

```
prediction.select("CustomerName", "cluster").show(50)
```

Predict Clusters

More Example: Recommendation

Import the ALS class

In this exercise, you will use the Alternating Least Squai

```
from pyspark.ml.recommendation import ALS
```

Load Source Data

The source data for the recommender is in two files - one containing numeric IDs for movies and users, a
of the movies.

```
ratings = spark.read.csv('wasb:///data/ratings.csv', inferSchema=True, header=True)
movies = spark.read.csv('wasb:///data/movies.csv', inferSchema=True, header=True)
ratings.join(movies, "movieId").show()
```

Prepare the Data

To prepare the data, split it into a training se

```
data = ratings.select("userId", "movieId", "rating")
splits = data.randomSplit([0.7, 0.3])
train = splits[0].withColumnRenamed("rating", "label")
test = splits[1].withColumnRenamed("rating", "trueLabel")
train_rows = train.count()
test_rows = test.count()
print "Training Rows:", train_rows, " Testing Rows:", test_rows
```

Build the Recommender

The ALS class is an estimator, so you can use its **fit** method to traing a model, or you can include it in a pipeline. Ratl
label, the ALS algorithm requries a numeric user ID, item ID, and rating.

```
als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="label")
model = als.fit(train)
```

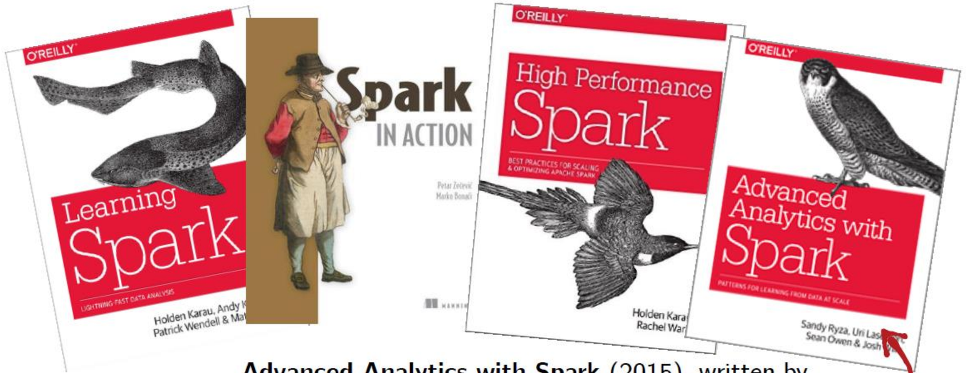
Test the Recommender

Now that you've trained the recommender, you can see how accurately it predicts known ratings in the test set.

```
prediction = model.transform(test)
prediction.join(movies, "movieId").select("userId", "title", "prediction", "trueLabel").show(100, truncate=False)
```

Now what?

Many excellent books released in the past year or two!



Advanced Analytics with Spark (2015), written by Sandy Ryza, Uri Laserson, Sean Owen, and Josh Wills

Big Data for Data Engineers

Yandex

Learn More

Big Data Analysis with Apache Spark

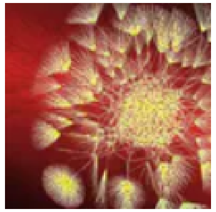
Learn how to apply data science techniques using parallel programming in Apache Spark to explore big data.

Distributed Machine Learning with Apache Spark

Learn the underlying principles required to develop scalable machine learning pipelines and gain hands-on experience using Apache Spark.

Introduction to Apache Spark

Learn the fundamentals and architecture of Apache Spark, the leading cluster-computing framework among professionals.



Big Data

University of California, San Diego

Learn More