MAS DSE 230
Scalable Analytics

Big Data Analytics

Mai H. Nguyen

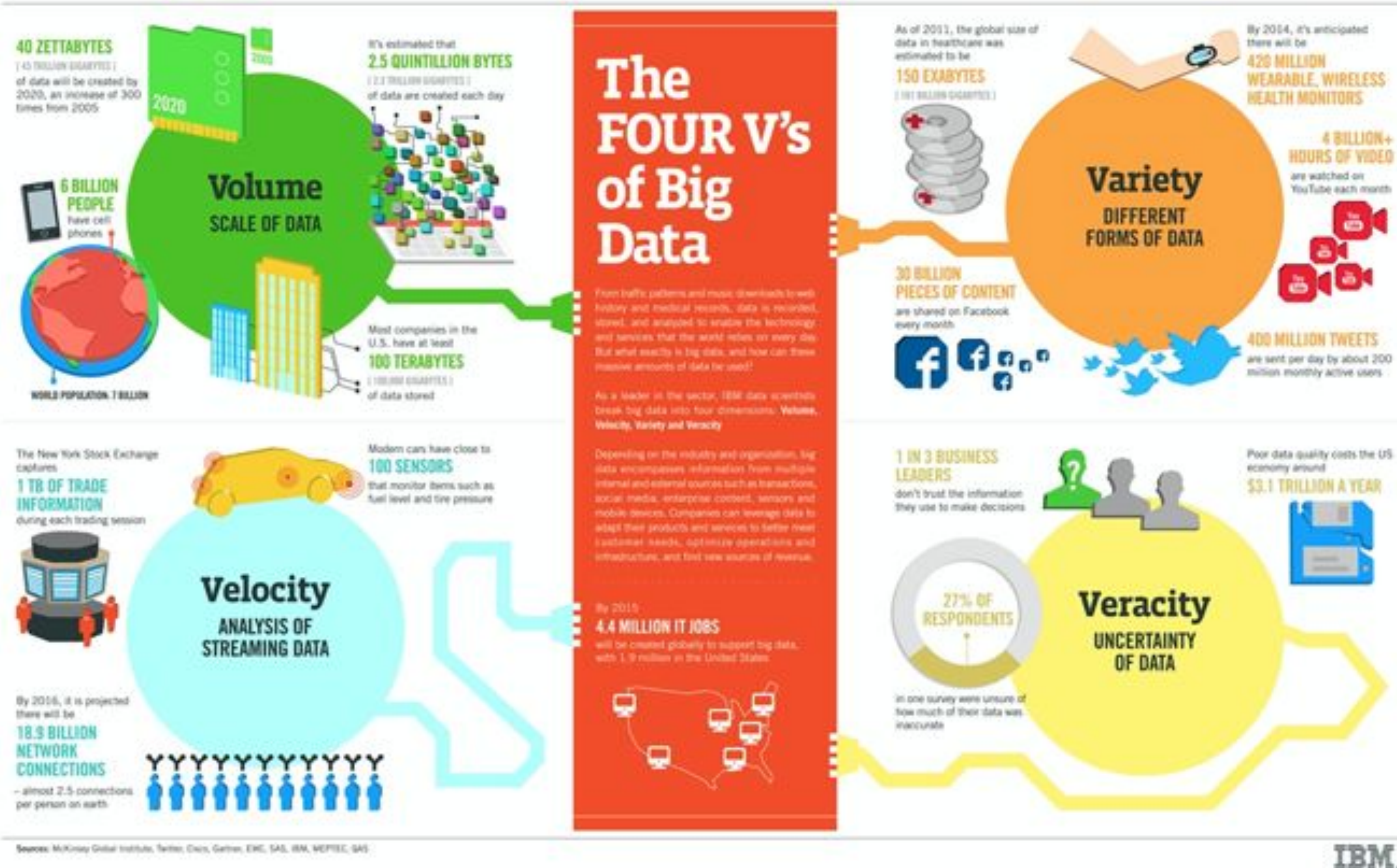
TODAY'S AGENDA

- Review
- Big Data Analytics
  - Spark Core & Libraries
  - MLlib
  - MLlib PySpark Examples
- Spark Exercise
- Guest Lecture
  - Peter Rose, Ph.D.
  - Director, Structural Bioinformatics Laboratory
  - "Scalable, Interactive, and Reproducible Data Mining of 3D Macromolecular Structures"
- Project Proposal Presentations

BIG DATA & DISTRIBUTED PROCESSING REVIEW

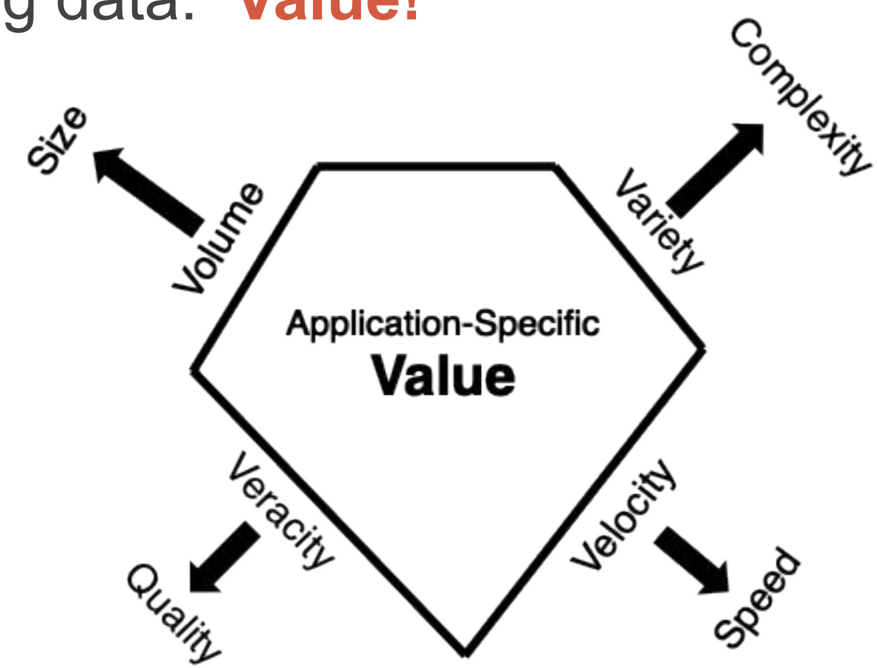
- Big Data Overview
- Scalable Systems
- Hadoop
- Spark

CHARACTERISTICS OF BIG DATA



CHARACTERISTICS OF BIG DATA

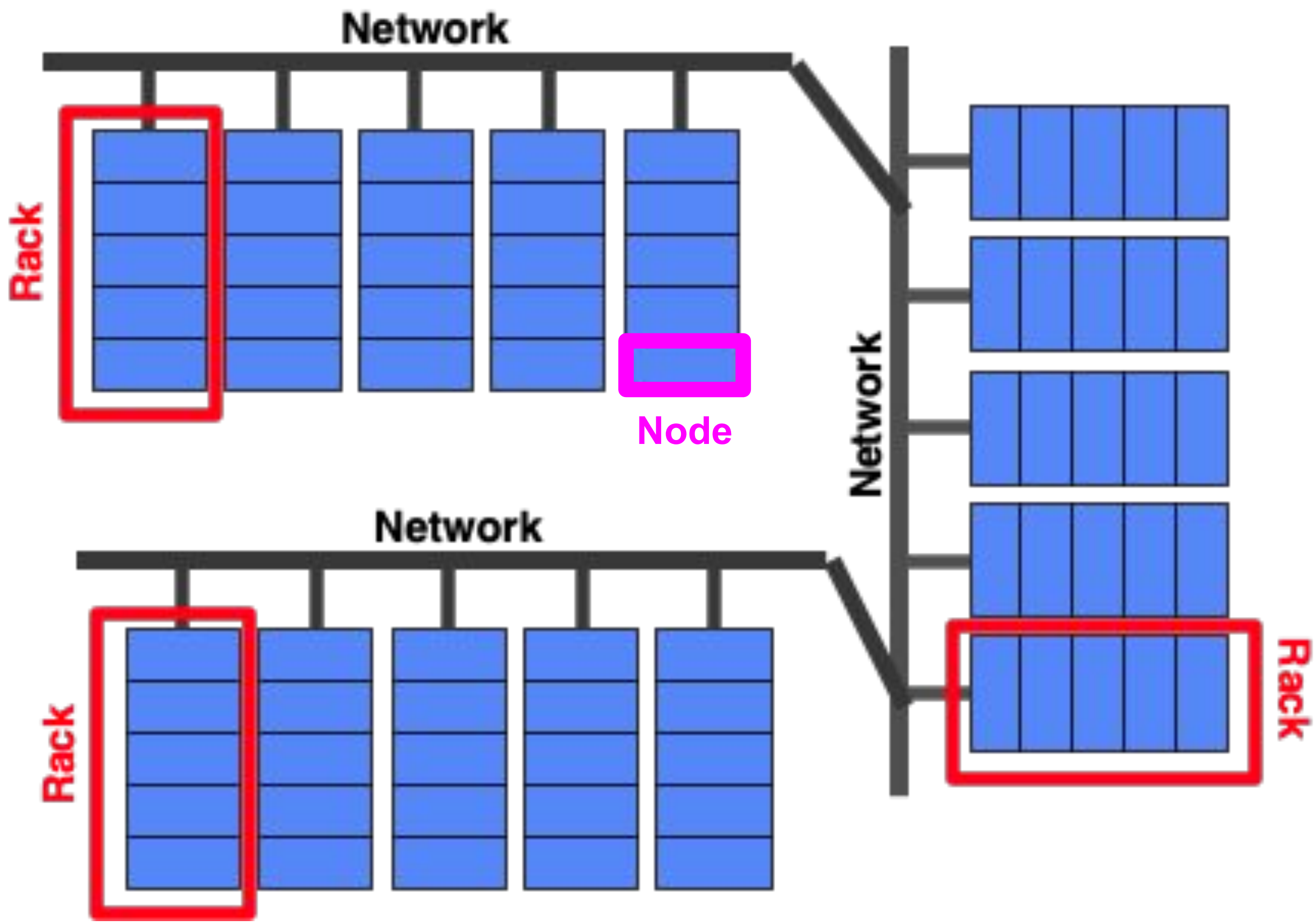
- Goal of processing data is to extract value from data
- Not sufficient to collect data
- Need to analyze data to make sense of it and gain insights
- So 5th ‘V’ of big data: **Value!**



SCALABLE SYSTEMS

- Key components
  - Distributed Computing
    - Processing of large data volumes
    - Scalability
    - Fault tolerance
    - Support for various workloads
  - Distributed File System
    - Data Partitioning
    - Data Replication

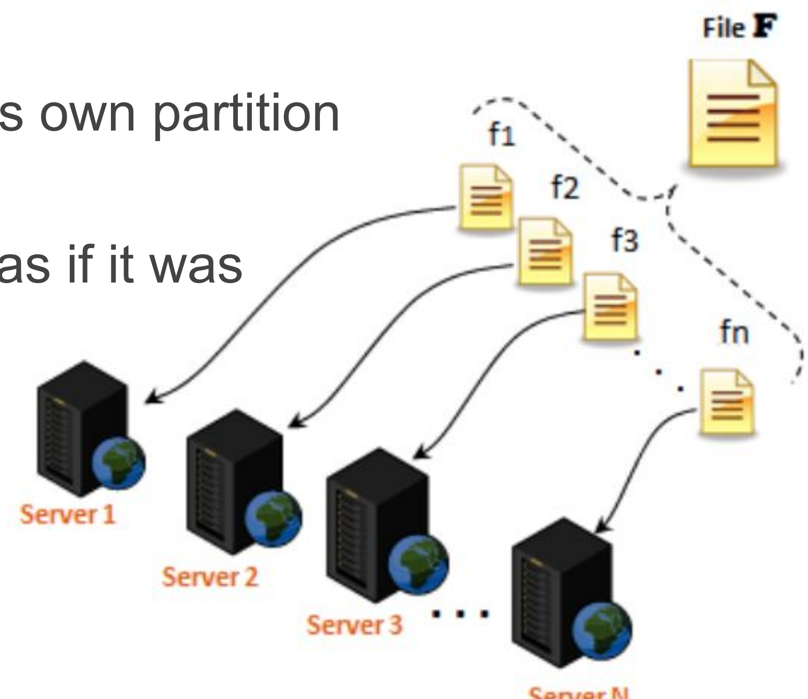
DISTRIBUTED COMPUTING



- Large data volumes
- Scalability
- Fault tolerance
- Diverse workloads

DISTRIBUTED FILE SYSTEM

- For efficient processing of very large data file
  - **Partition** data across many computer systems (aka **sharding**)
- **Distributed file system (DFS)**
  - Manages data that is distributed across many networked systems
  - Each local file system manages its own partition
  - Works on top of local file systems
  - Data is accessed and processed as if it was stored on local client machine
  - *Virtualization*: Gives illusion of a single local file
    - Generalization of virtual memory on single system



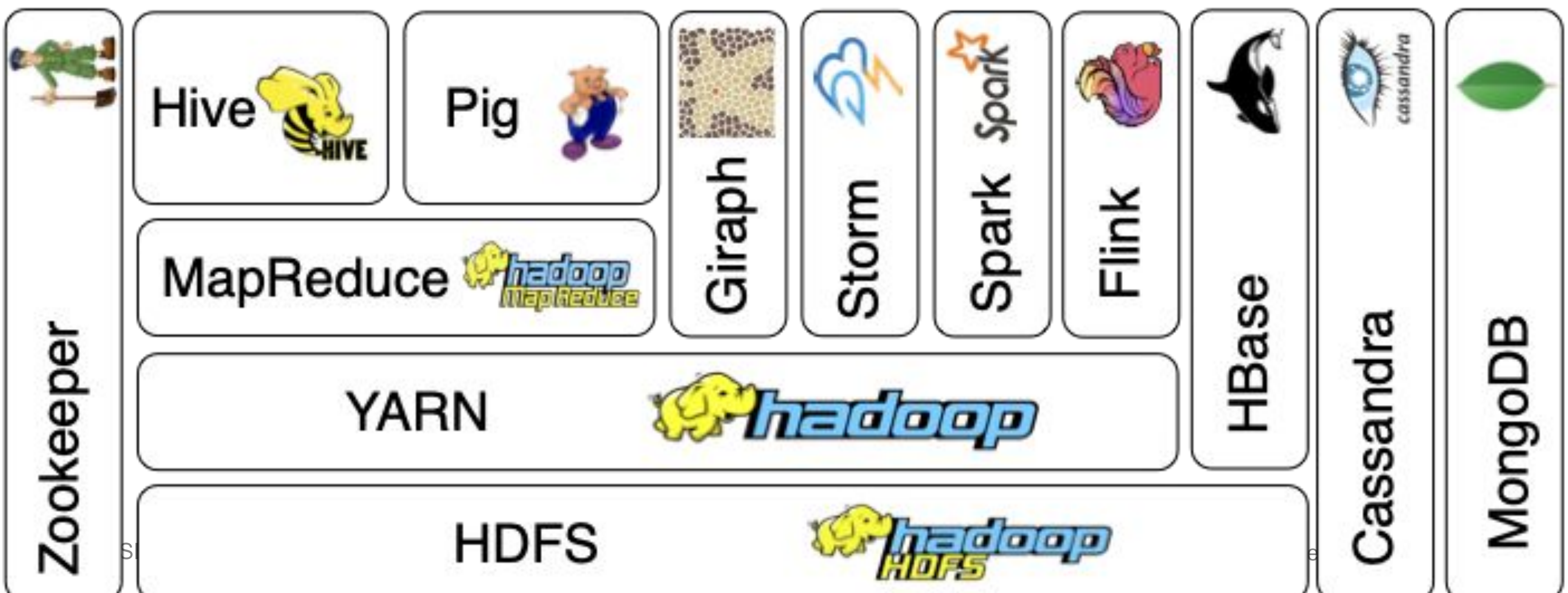
https://www.linkedin.com/pulse/distributed-file-system-google-replica-sridhar-ramasamy

DISTRIBUTED FILE SYSTEM

- Data Partitioning
  - Divide large dataset and distribute subsets across nodes
  - Enables handling of large data files via data parallelism
  - Provides scalability
- Data Replication
  - Data partitions are copied, and copies are distributed across nodes
  - Enables fault tolerance and high concurrency

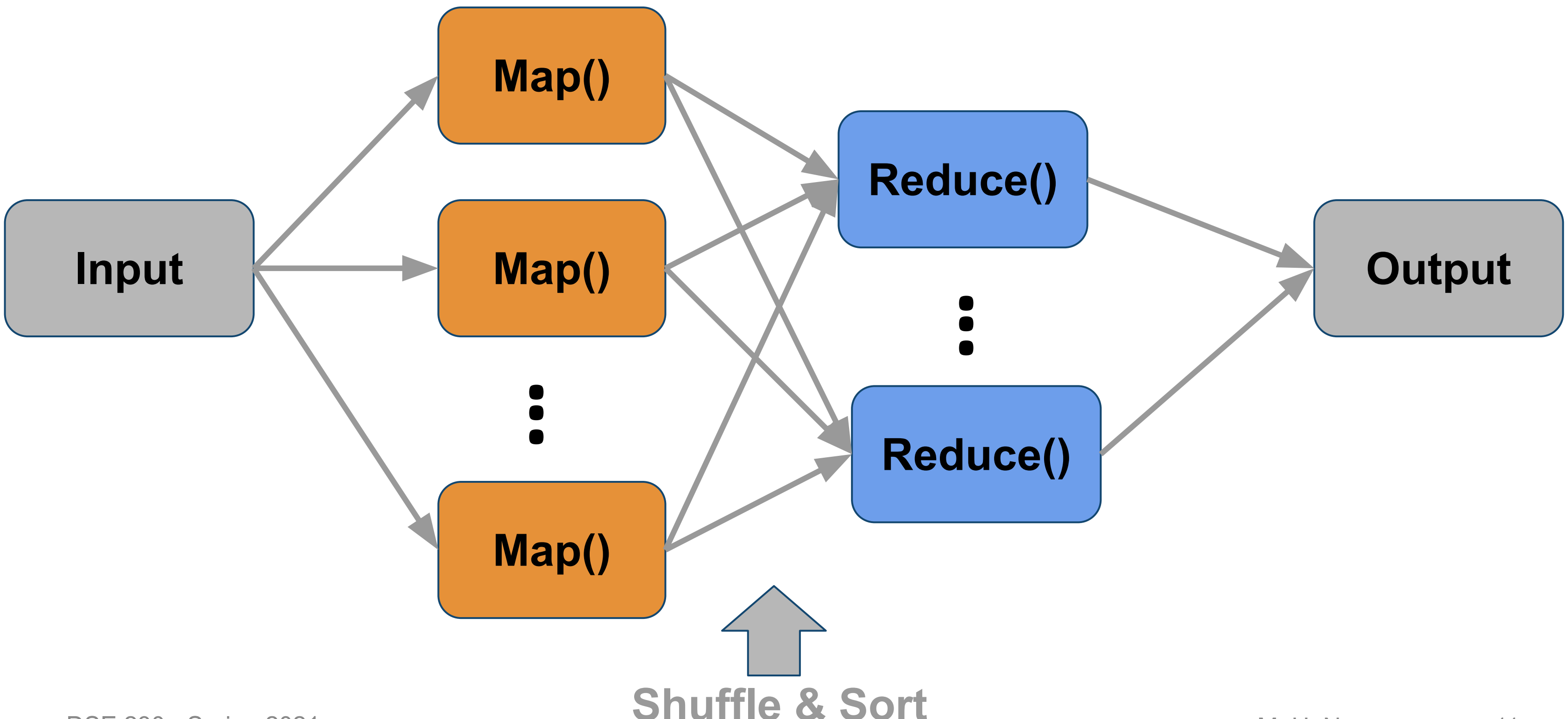
HADOOP

- System for distributed processing of large data sets across clusters of computers using simple programming models.
  - Data partitioning, fault tolerance, etc. all handled by the Hadoop library under the covers
  - Scalable platform on commodity clusters

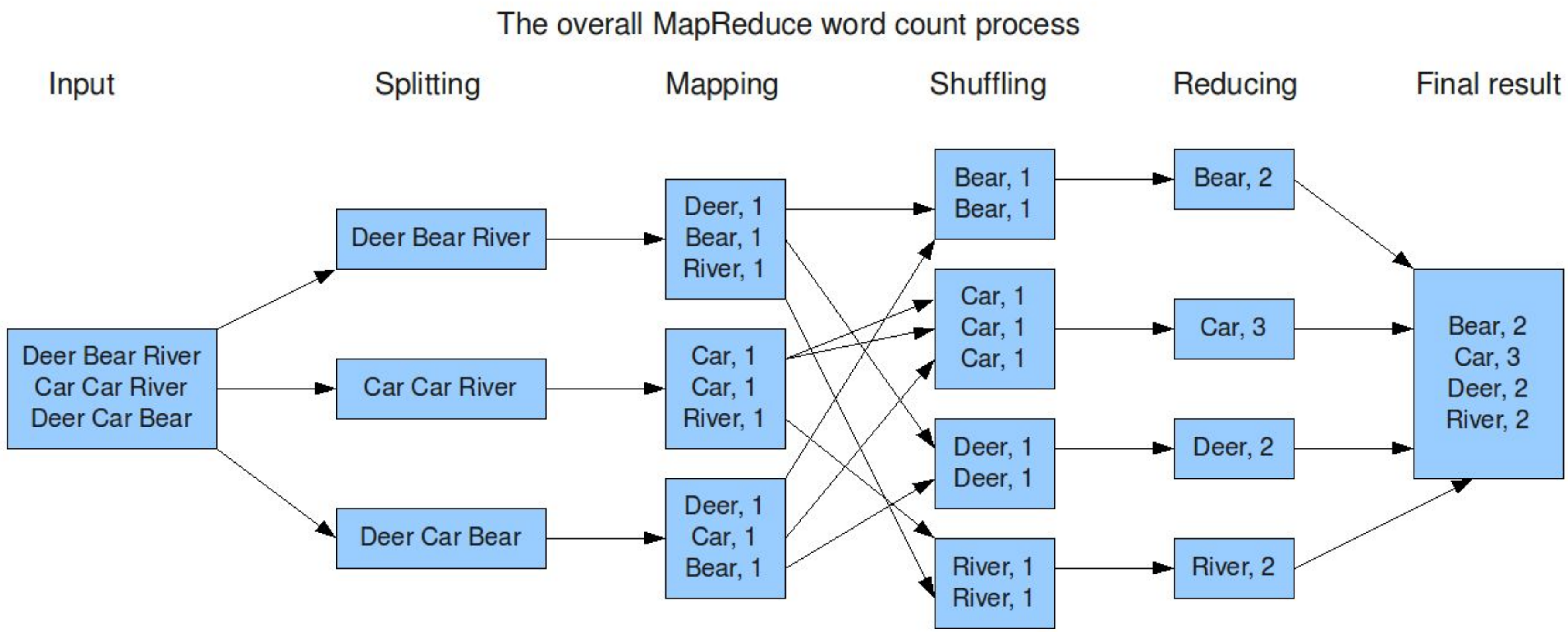


MapReduce

- **Map**: Apply operation to all data elements
- **Reduce**: Summarize elements



MapReduce: WORDCOUNT IN DETAIL



https://www.todaysoftmag.com/article/1358/hadoop-mapreduce-deep-diving-and-tuning

Data is partitioned across nodes

Map generates key-value pairs

Pairs with same key moved to same node

Reduce sums values for each key

SPARK

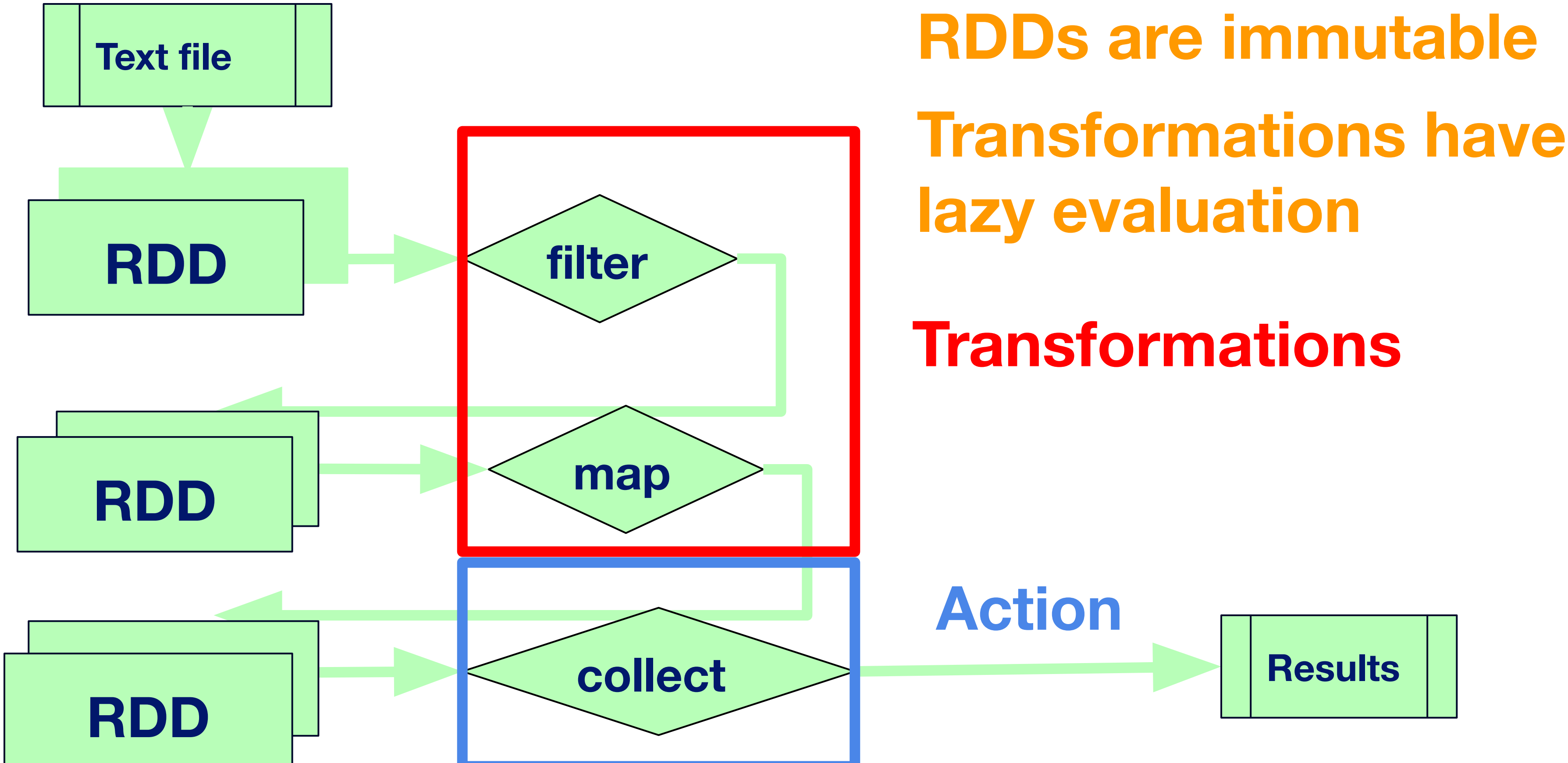
- Computing platform for distributed computing
- Runs on commodity clusters
- Provides built-in parallelism and fault tolerance
- Goals: **speed**, ease of use, generality, unified platform
- In-memory processing
  - Exploits distributed memory to cache data
  - Intermediate results written to memory instead of disk
- How does Spark manage data in distributed system?

RDDs

- Resilient Distributed **Dataset**
  - Collection of data
    - From files in local filesystem (text, JSON, etc.)
    - From data store (HDFS, RDBMS, NoSQL, etc.)
    - Created from another RDD
- Resilient **Distributed** Dataset
  - Data is divided into partitions
  - Partitions are distributed across nodes in cluster
- **Resilient** Distributed Dataset
  - Provides resilience (e.g., fault tolerance) to failures
  - History of operations performed on each partition is tracked to provide lineage-based fault tolerance
- All provided automatically by Spark engine

PROCESSING RDDs

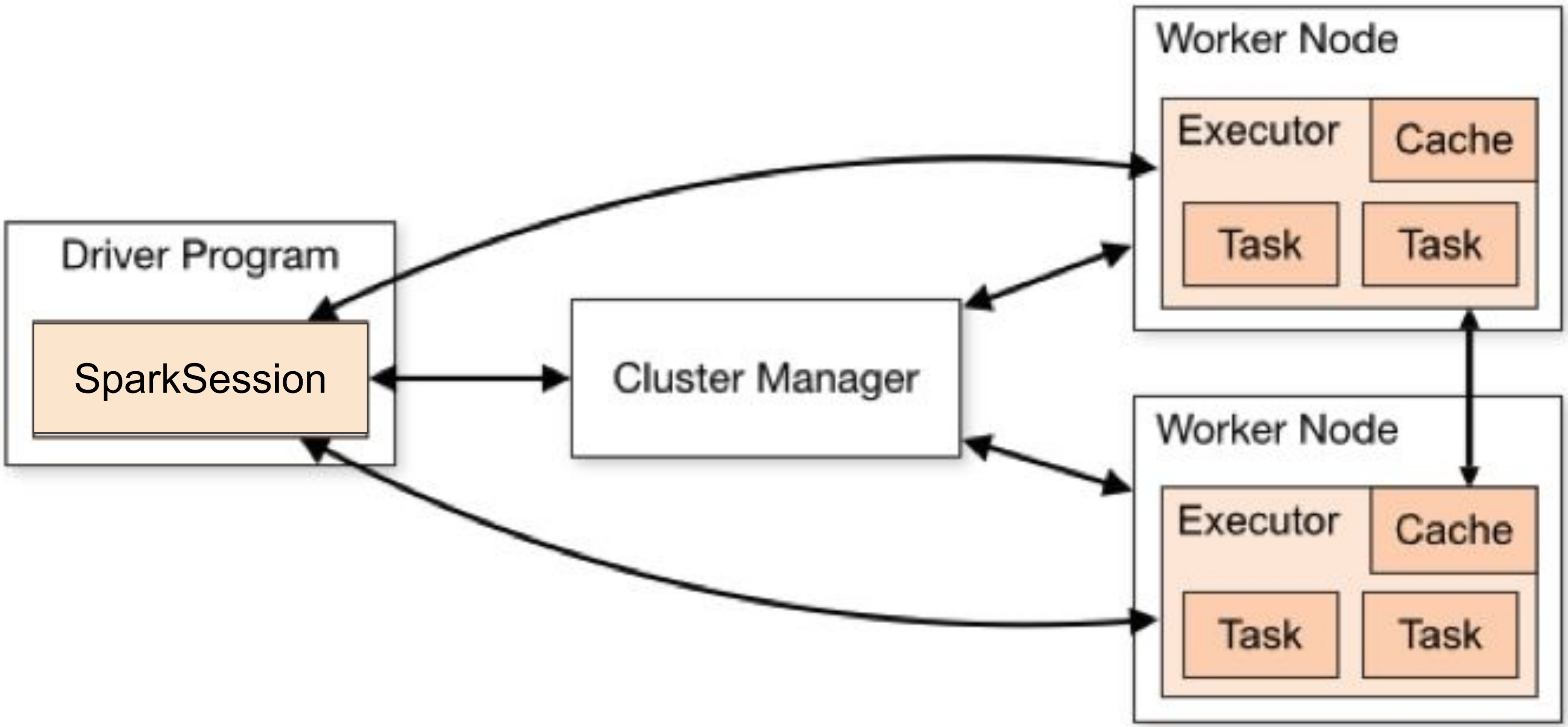
- RDDs can be processed using 2 types of operations
  - **Transformation**: Creates new RDD from existing RDD
  - **Action**: Runs computation(s) on RDD and returns value



DATAFRAMES & DATASETS

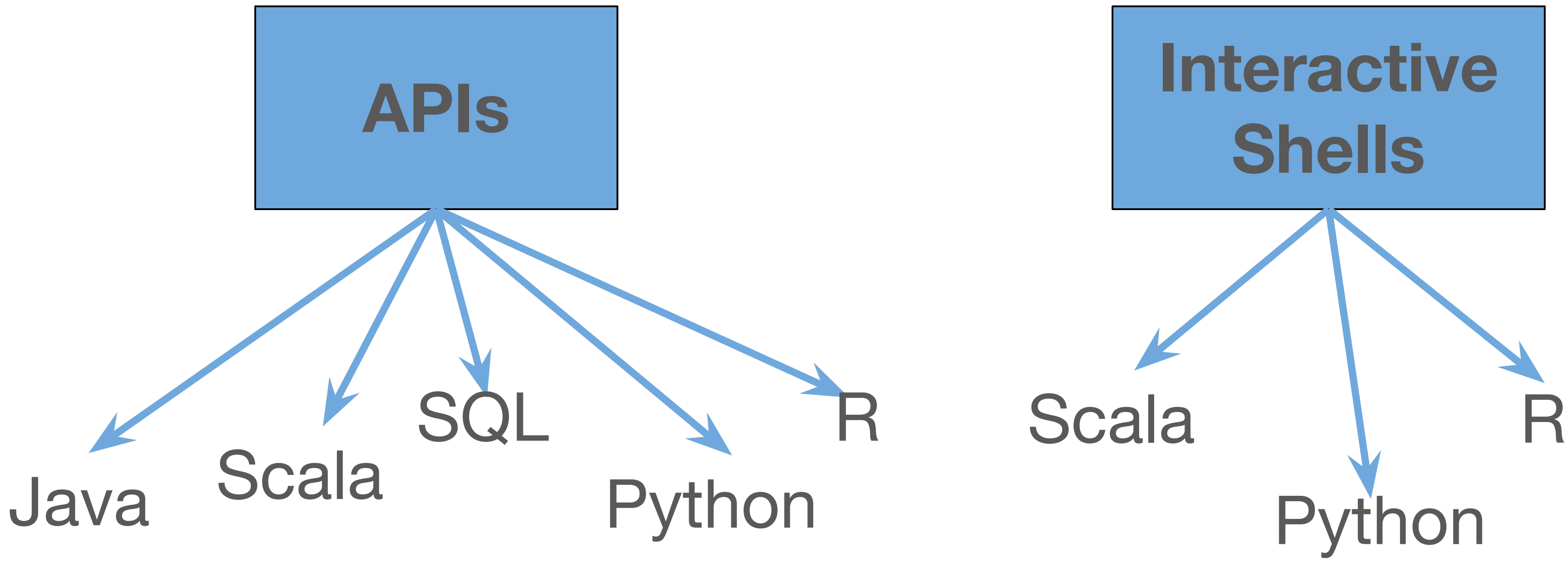
- Extensions to RDDs
  - Higher-level abstractions
  - Improved performance
  - Better scalability
- DataFrame
  - No static type checking
  - APIs in Java, Scala, Python, R
- DataSet
  - Static type checking
  - APIs in Java and Scala
- Can convert to/from RDDs and use with RDDs

SPARK ARCHITECTURE



SPARK INTERFACE

Goals: speed, **ease of use**, generality, unified platform



BIG DATA ANALYTICS REVIEW

- Machine Learning Overview
- Data Exploration
- Data Preparation
- Modeling
  - Categories of Machine Learning Techniques
  - Building and Applying a Model
  - Classification
  - Regression
  - Cluster Analysis

WHAT IS MACHINE LEARNING?

learning from data

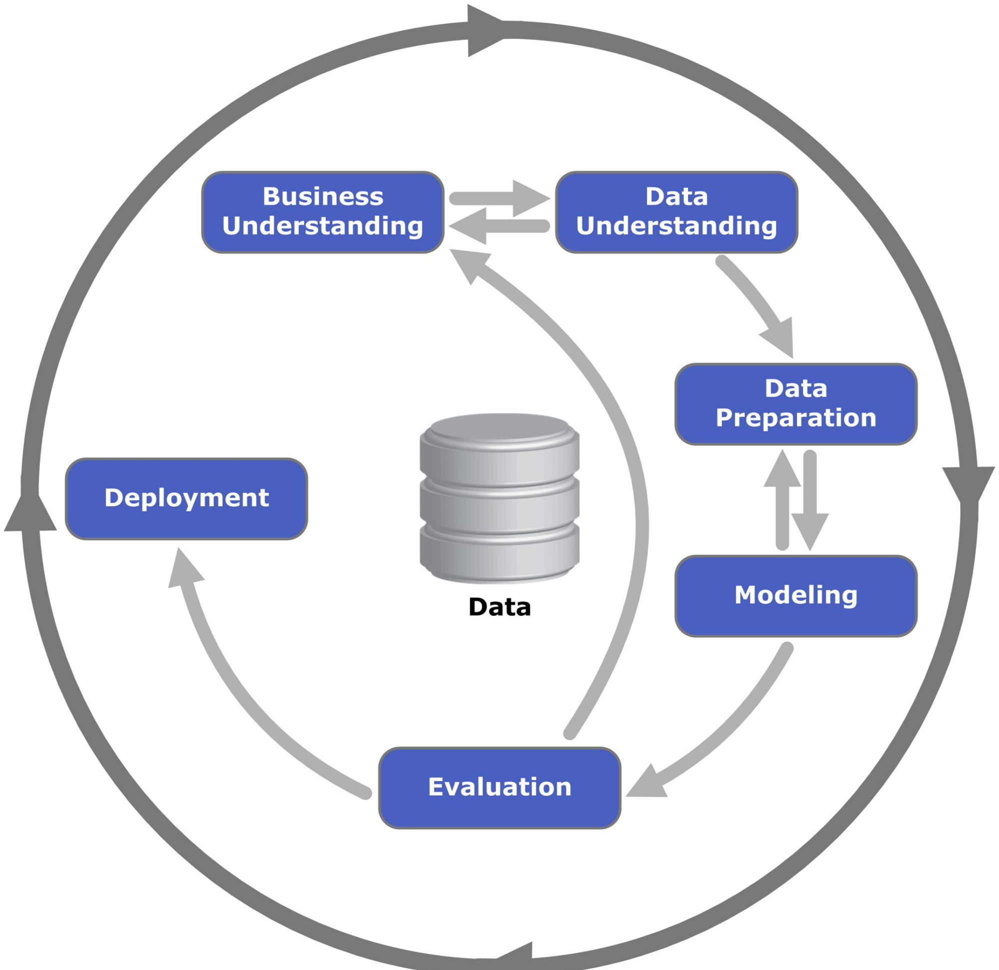
no explicit programming

discover hidden patterns

data-driven decisions

The field of machine learning focuses on the study and construction of computer systems that can learn from data without being explicitly programmed. Machine learning algorithms and techniques are used to build models to discover hidden patterns and trends in the data, allowing for data-driven decisions to be made.

MACHINE LEARNING PROCESS

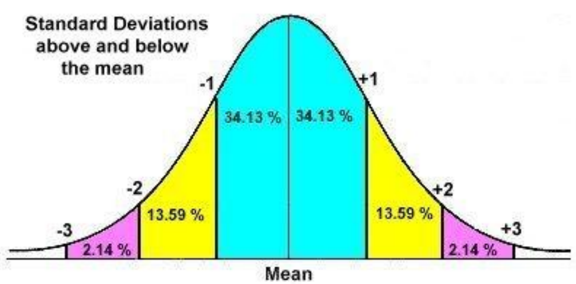


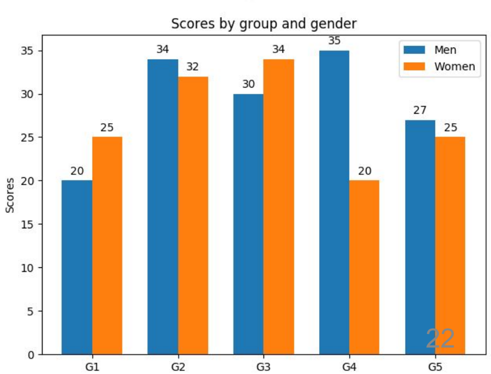
https://en.wikipedia.org/wiki/Cross_Industry_Standard_Process_for_Data_Mining

CRoss Industry Standard Process for Data Mining

DATA EXPLORATION

- Definition
  - Preliminary investigation of your data
- Purpose
  - To gain better understanding of specific characteristics of the data
  - To look for: Correlations, general trends, outliers, etc.
- Also referred to as 'EDA'
  - Exploratory Data Analysis
- Techniques
  - Data validation
  - Summary statistics
  - Visualization



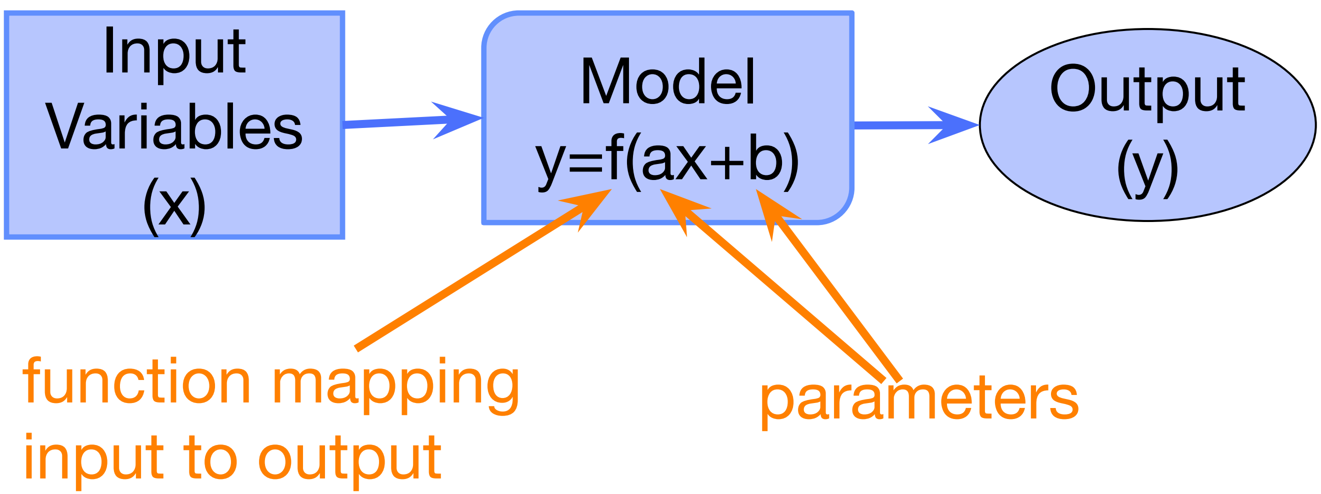


DATA PREPARATION

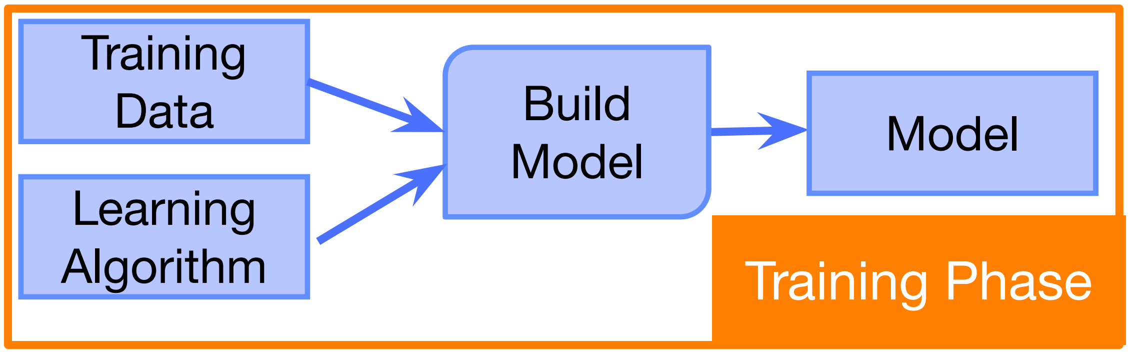
- Goal
  - Create data for analysis
- Activities
  - Clean data:
    - Identify and address data quality issues
    - Examples: missing data, duplicate data, invalid data
  - Feature engineering
    - Feature selection
      - Add, remove, combine features
    - Feature transformation
      - scaling
      - aggregation
      - discretization
      - one-hot encoding
      - dimensionality reduction

BUILDING MACHINE LEARNING MODEL

- Model parameters are adjusted during model training to change input-output mapping
- Parameters are learned or estimated from data
  - “fitting the model”, “training the model”, “building the model”
- Goal: Minimize some error function

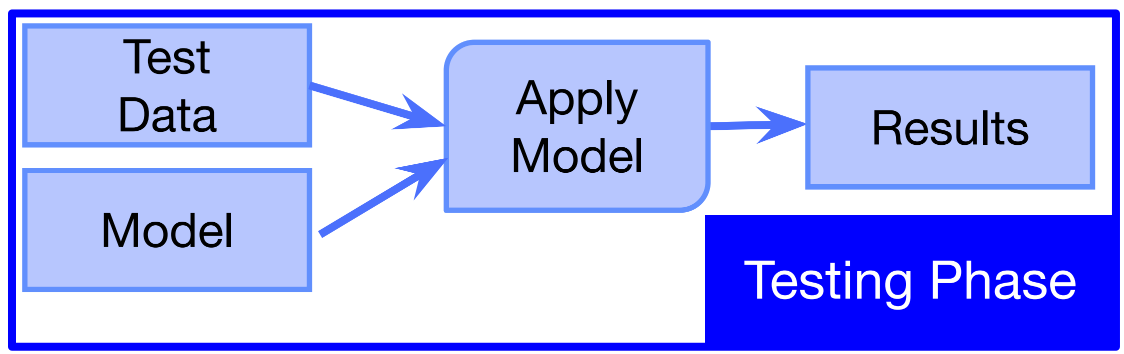


BUILDING VS APPLYING MODEL



Adjust model parameters
"Train"

Test model on new data
"Inference"

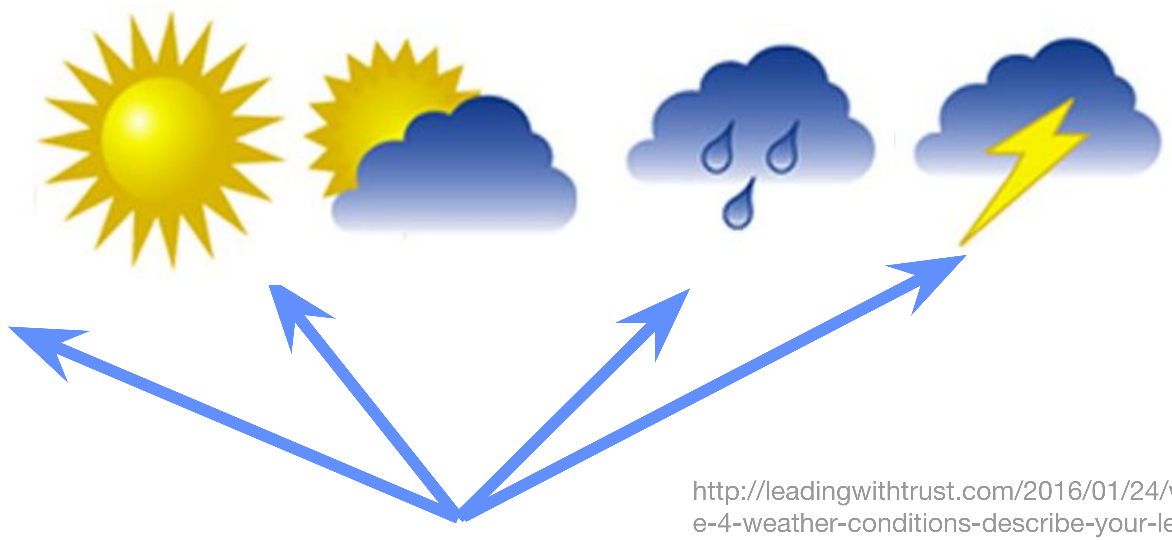


OVERFITTING & GENERALIZATION

- Overfitting
  - Model is fitting to noise in data instead of to underlying distribution of data
- Reasons for overfitting
  - Training set is too small
  - Model is too complex, i.e., has too many parameters
- Overfitting leads to poor generalization
  - Model that overfits will not generalize well to new data

CLASSIFICATION

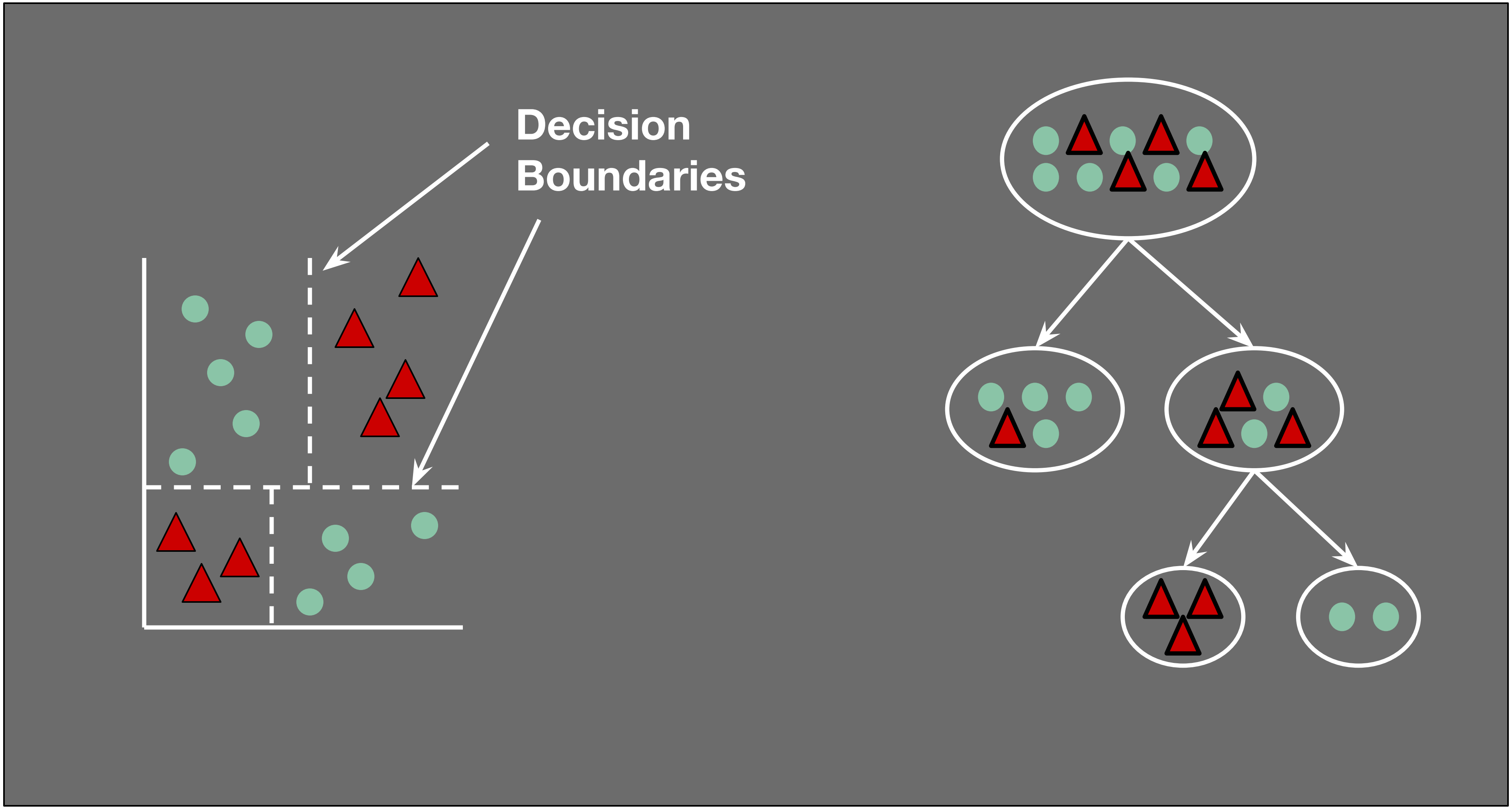
- Goal: Predict category given input data
  - Target is categorical variable



http://leadingwithtrust.com/2016/01/24/which-of-these-4-weather-conditions-describe-your-leadership/

- Examples
  - Classify tumor as benign or malignant
  - Determine if credit card transaction is legitimate or fraudulent
  - Identify customer as residential, commercial, public
  - Predict if weather will be sunny, cloudy, windy, or rainy

DECISION TREE MODEL



REGRESSION

- Goal: Predict numeric value given input data
  - Target is numeric variable

www.wallstreetpoint.com

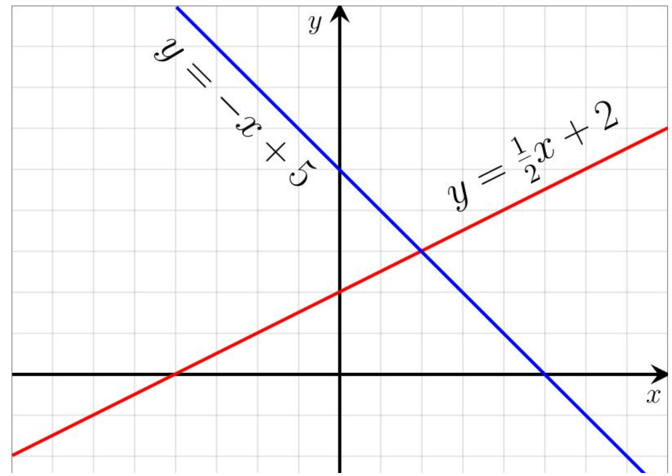
- Examples
  - Predict price of stock
  - Estimate demand for a product based on time of year
  - Determine risk of loan application
  - Predict amount of rain

LINEAR REGRESSION

Special case (2D):

$$y = mx + b$$

slope *y-intercept*



General case:

Weights: parameters we need to find to model y

$$y = w_0 + w_1 x_1 + \ldots + w_n x_n$$

Output

Inputs: features we are trying to model output y with

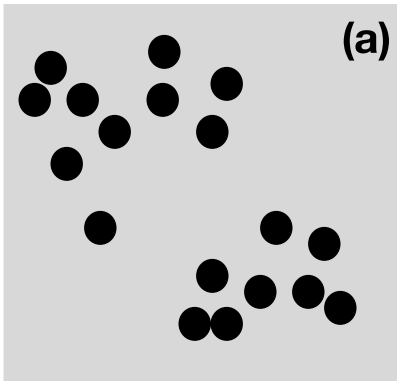
CLUSTER ANALYSIS

- Goal: Organize similar items into groups

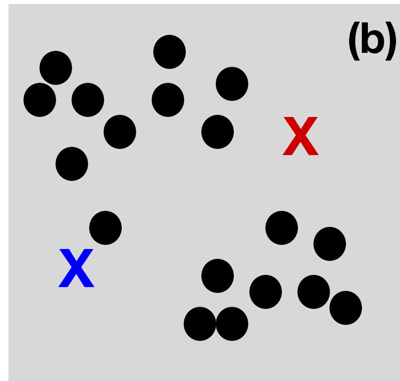
http://www.bostonlogic.com/blog/2014/01/segment-your-leads-to-get-better-results/

- Examples
  - Group customer base into segments for effective targeted marketing
  - Identify areas of similar topography (desert, grass, etc.)
  - Categorize different types of tissues from medical images
  - Discover crime hot spots

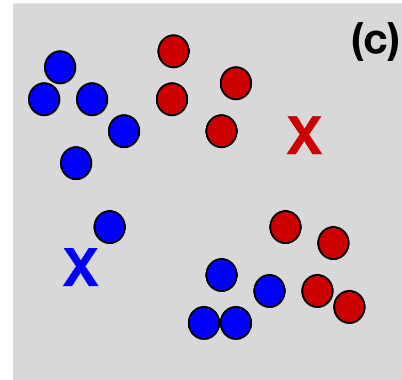
k-MEANS CLUSTERING ILLUSTRATION



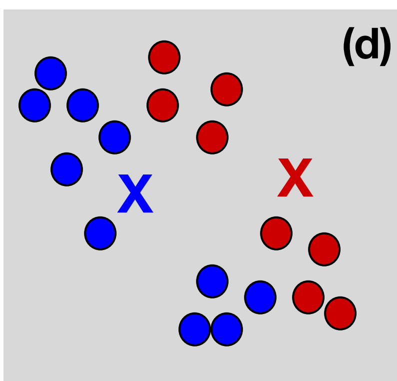
Original samples



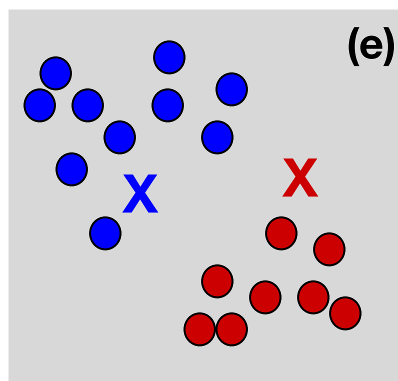
Initial Centroids



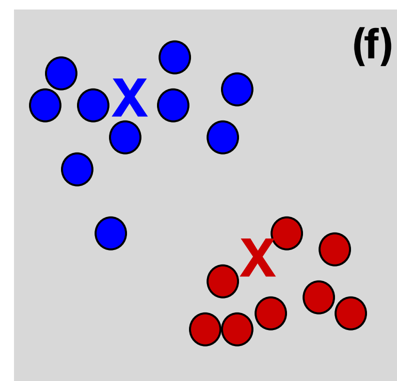
Assign Samples



Re-calculate Centroids



Assign Samples



Re-calculate
Centroids

BIG DATA ANALYTICS

- Machine Learning Overview
- Data Exploration
- Data Preparation
- Modeling
- Spark Core & Libraries
- PySpark MlLib Examples
- PySpark Exercise
- Assignments
- Guest Lecture
- Project Proposal Presentations

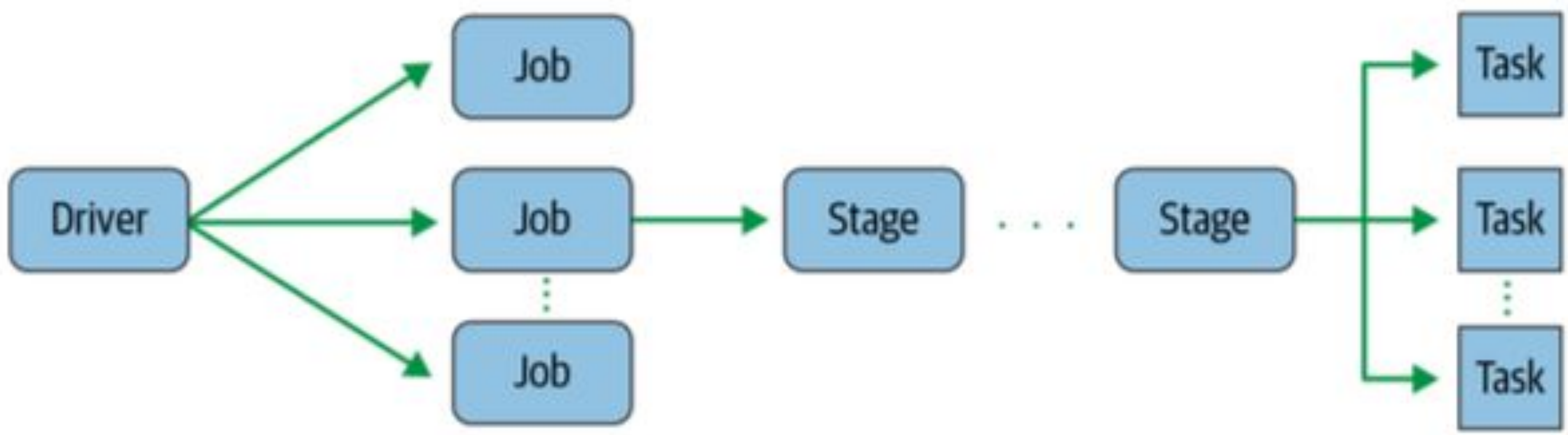
SPARK PROGRAM STRUCTURE

- Start Spark session
- Create distributed dataset
- Apply transformations
- Perform actions
- Stop Spark session
  - spark.stop()

SPARK PROCESS CONCEPTS

- **Application**
  - User program consists of driver program and executors on cluster
- **SparkSession**
  - Object that provides point of entry to interact with underlying Spark functionality using Spark APIs
- **Job**
  - Parallel computation consisting of multiple operations that are executed in response to Spark actions
- **Stage**
  - Each job gets divided into smaller units called stages
- **Task**
  - Single unit of work or execution sent to executor

SPARK APPLICATION

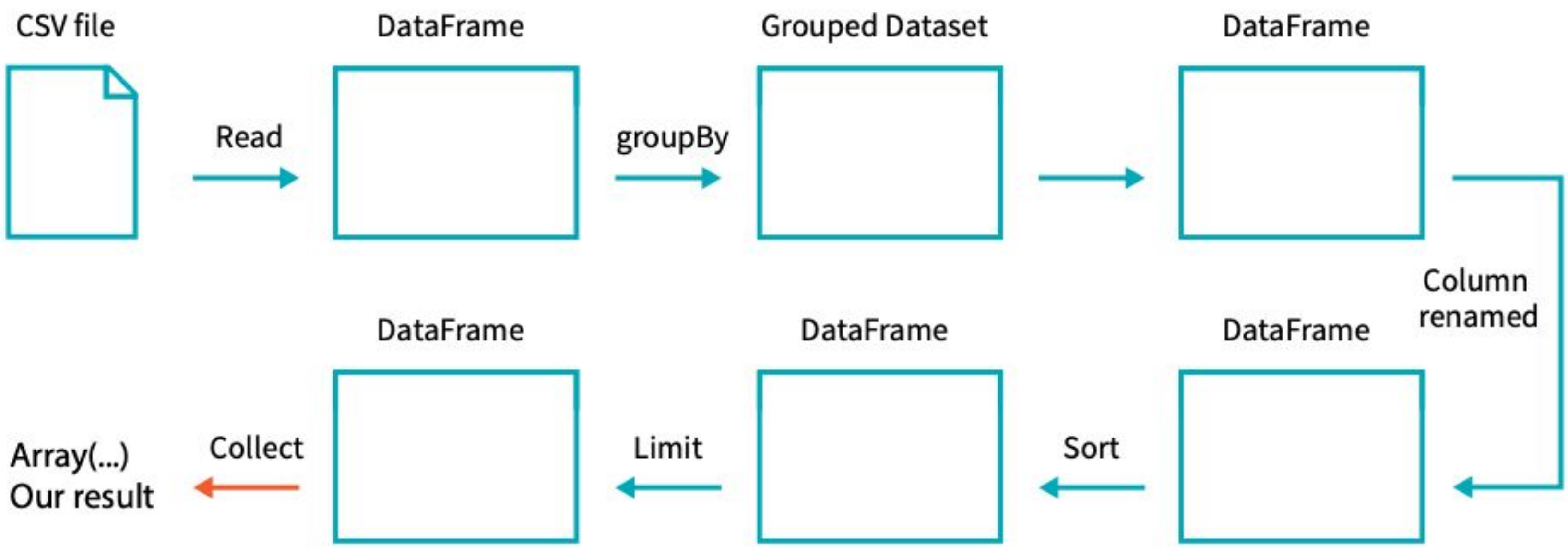


- Driver creates SparkSession object
- Driver converts Spark application into one or more jobs
- An action creates a job
- DAG (directed acyclic graph) of instructions built for each job
- Each node in DAG is single or multiple Spark stages
- Each stage is broken down into tasks
- Tasks are distributed to executors

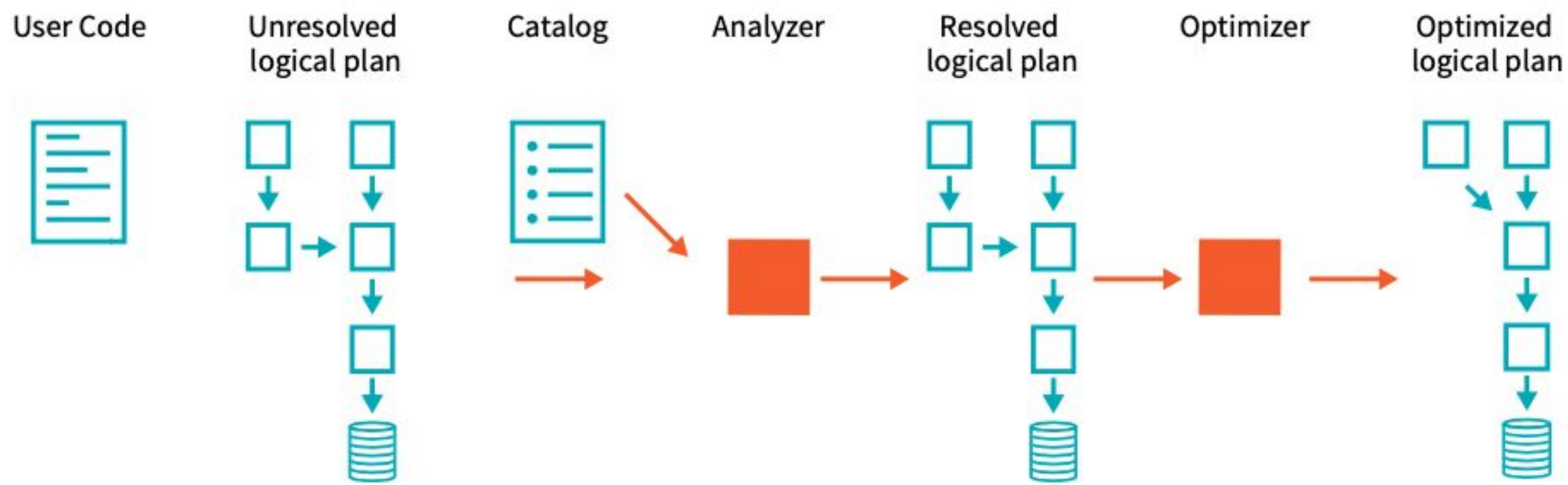
LOGICAL PLAN

DEST_COUNTRY_NAME, ORIGIN_COUNTRY_NAME, count
United States, France, 517

```
flightData2015\
    .groupBy("DEST_COUNTRY_NAME")\
    .sum("count")\
    .withColumnRenamed("sum(count)", "destination_total")\
    .sort(desc("destination_total"))\
    .limit(5)\
    .collect()
```

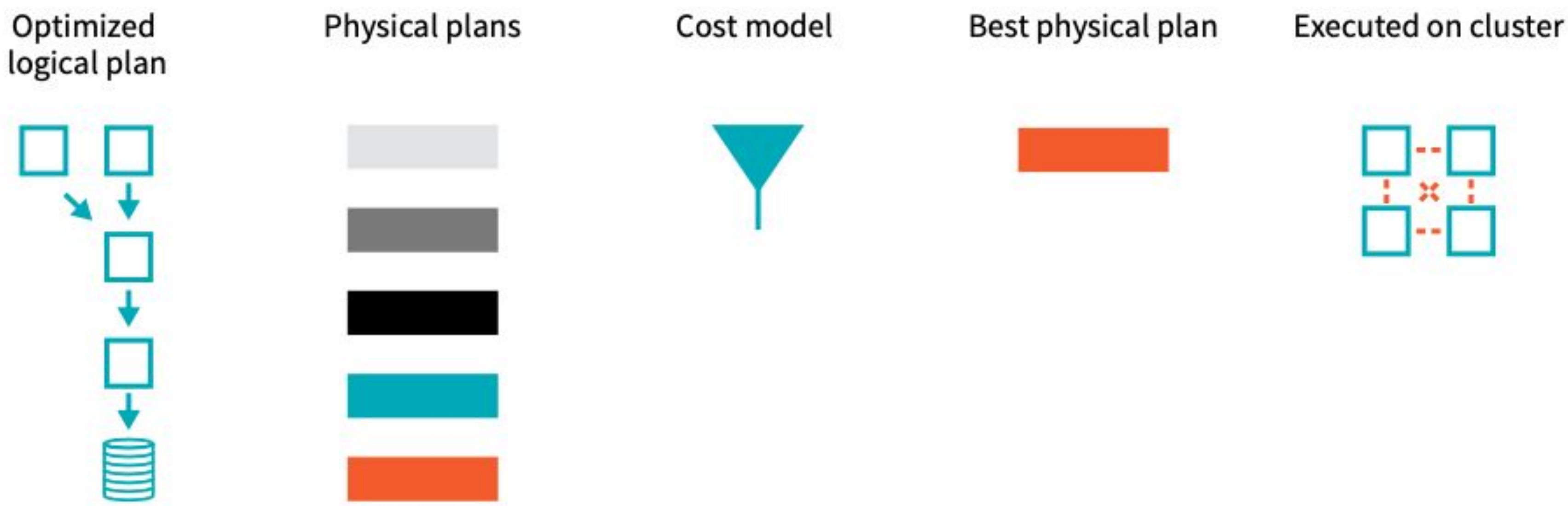


LOGICAL PLAN OPTIMIZATION



- User code is converted to unresolved logical plan
- Analyzer uses catalog of tables to resolve columns & tables
- Optimizer optimizes logical plan

PHYSICAL PLAN



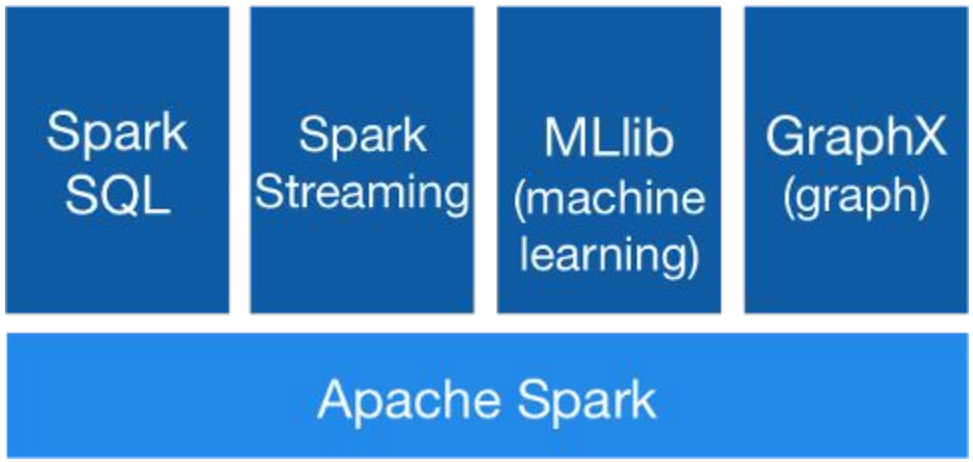
- Different physical execution strategies are generated and compared
- Physical plan generates series of RDDs and transformations
- Code executed on RDDs by executors in cluster
- Summary: Spark code -> logical plan -> physical plan -> code executed in cluster

SPARK - GENERALITY

- Goals: speed, ease of use, **generality,** unified platform
- Support for several data sources
  - Local file systems, HDFS, RDBMSs, MongoDB, Kafka, AWS S3, etc.
- Can run on various platforms
  - Hadoop, Kubernetes, cloud, standalone
- Support for multiple workloads
  - batch, streaming
  - machine learning, SQL, graph processing

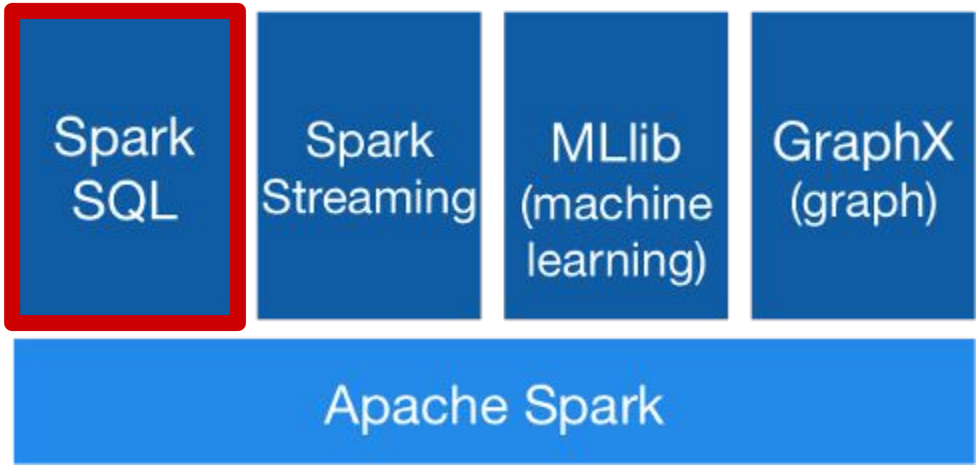
SPARK AS UNIFIED PLATFORM

- Goals: speed, ease of use, generality, **unified platform**



- Provides unified platform for various analytics processing
- **Spark engine** provides core capabilities for distributed processing
- **Spark libraries** provide additional higher-level functionality for diverse workloads

SPARK SQL



- Structured Data Processing
  - Provides support for SQL and query processing
  - Structure of data and computations allow for efficient query plan can be constructed
  - Has APIs for SQL, Scala, Java, Python, and R
  - Generated underlying code is identical

SPARK SQL

- Provides engine for DataFrame and DataSet APIs
- Provides API to issue SQL queries on structured data
- Provides uniform access to different data sources
  - JSON, CSV, HDFS, Cassandra, etc.
- Connects to all tools using JDBC/ODBC
- Supports ANSI SQL:2003 and HiveQL
- Can integrate SQL queries with Spark commands

SPARK SQL

- Execute SQL queries
  - SQL

    ```
    spark.sql(“SELECT max(count)
    FROM flight_data”).take(1)
    ```

  - PySpark

    ```
    from pyspark.sql.functions import max
    flight_data.select(max(“count”)).take(1)
    ```

SPARK SQL

- Create databases and tables

  spark.sql(“CREATE DATABASE my_table”)

  spark.sql(“USE my_table”)

- Perform queries

  df = spark.read.csv(“data.csv”, header=True, inferSchema=True)

  df.filter(df[“Col”] == “value”).show()

  df.groupBy(“Col”).count().show()

- Print schema

  df.printSchema()

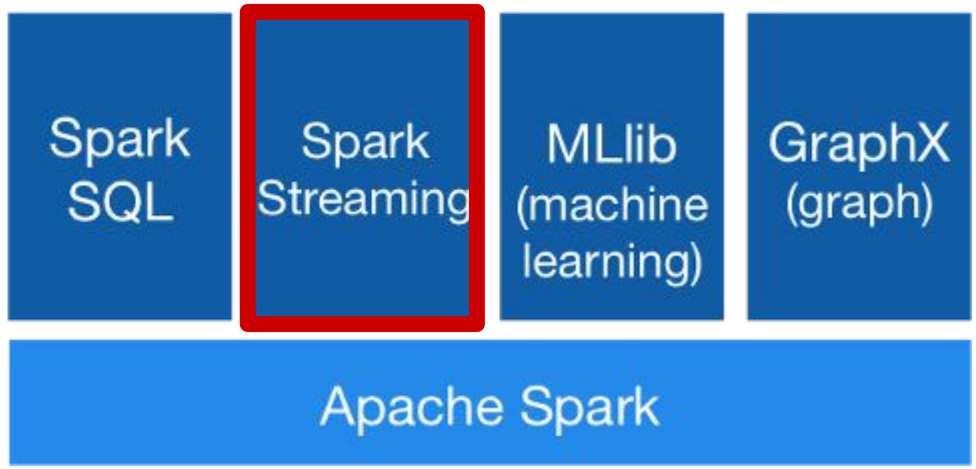
SPARK SQL

- Integrate SQL queries with Spark commands

```
df = spark.sql (“SELECT * FROM Employees”)
df.show(100)

num_employees =
  df.select(“Age”,”Dept”,”Salary”)
    .groupBy(“Dept”)
    .where(df.Salary > 80000)
    .count()
```

SPARK STREAMING



- Streaming Data Processing
  - Scalable processing for real-time analytics
  - Structured streaming
    - Data stream is divided into micro-batches of data
    - Same operations for static data can be used
  - Has APIs for Scala, Java, and Python

REAL-TIME ANALYTICS

- (Near) Real-Time Analytics
  - Analysis and use of data as it enters system
- Examples
  - Identifying fraudulent credit card transaction at point-of-sale
  - Viewing orders as they happen for up-to-date inventory tracking and trend analysis
  - Understanding trending topics of tweets/news articles/etc.

SPARK STREAMING

- Enables scalable processing of live data streams
- Can ingest data from multiple sources
- Processes data in cluster using algorithms available in Spark core and other libraries

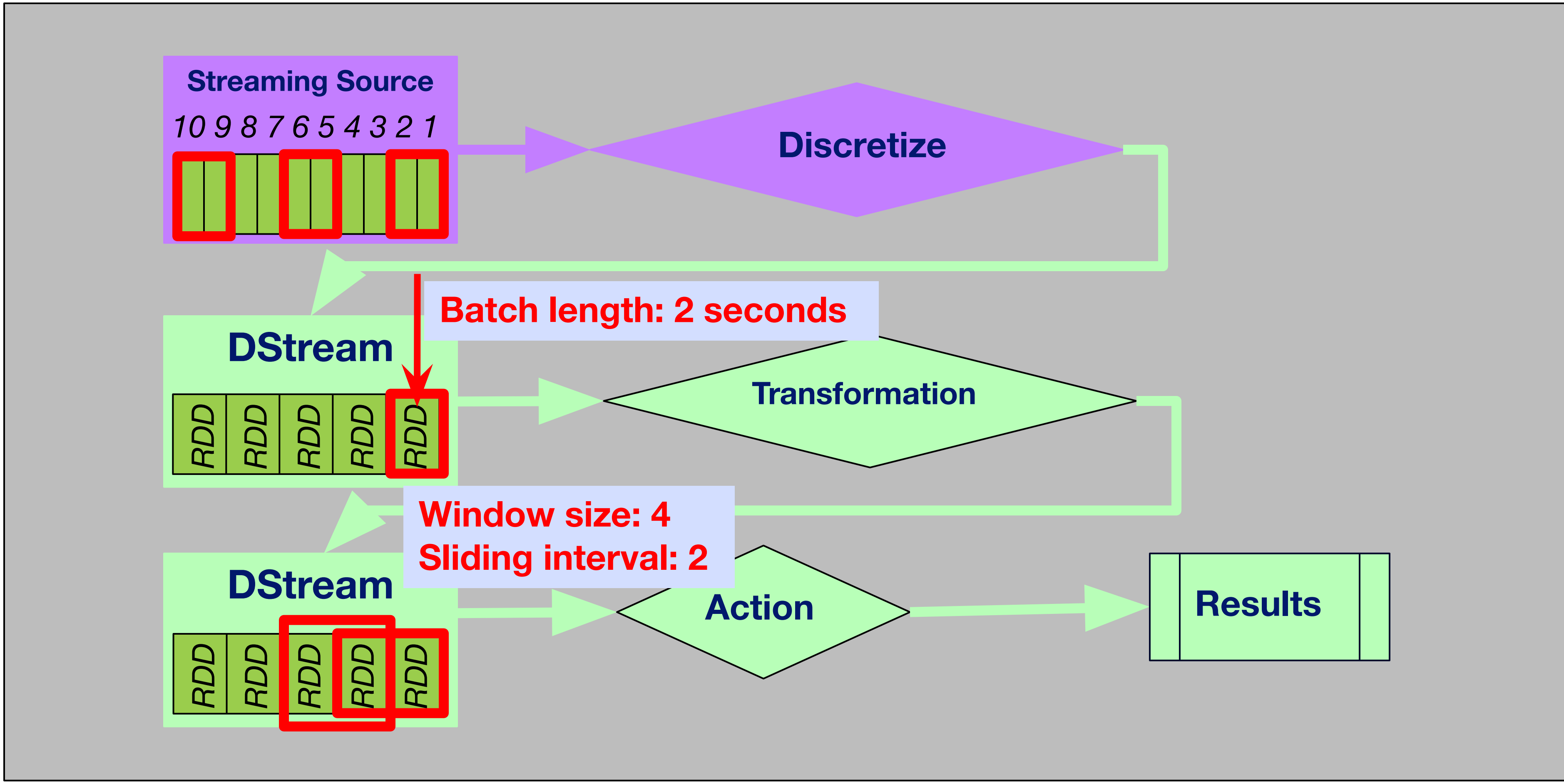
https://spark.apache.org/docs/latest/streaming-programming-guide.html

SPARK STREAMING

- Input data stream is divided into batches of data that are processed by Spark engine
- DStream: high-level abstraction
  - Implemented as sequence of RDDs
- Any Spark operation can be applied to DStreams

https://spark.apache.org/docs/latest/streaming-programming-guide.html

SPARK STREAMING

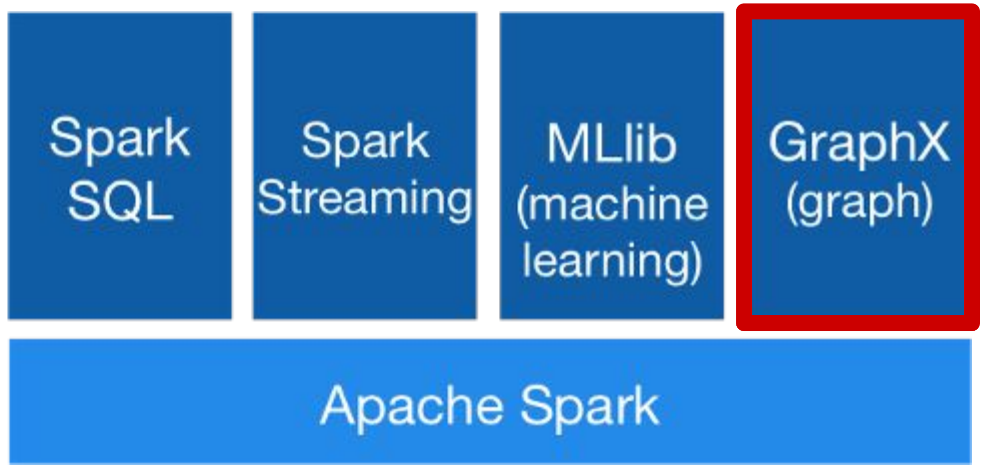


SPARK STREAMING

- Example: Count number of words in streaming text from socket

```
sc = SparkContext()
ssc = Streamingcontext(sc,1)   // Batch interval of 1 second
lines = ssc.socketTextStream(<hostname>,<portnumber>)
words = lines.flatMap(lambda line: line.split(" "))
pairs = words.map(lambda word: (word,1))
wordCounts = pairs.reduceByKey(lambda x,y: x+y)
wordCounts.pprint(20)
```

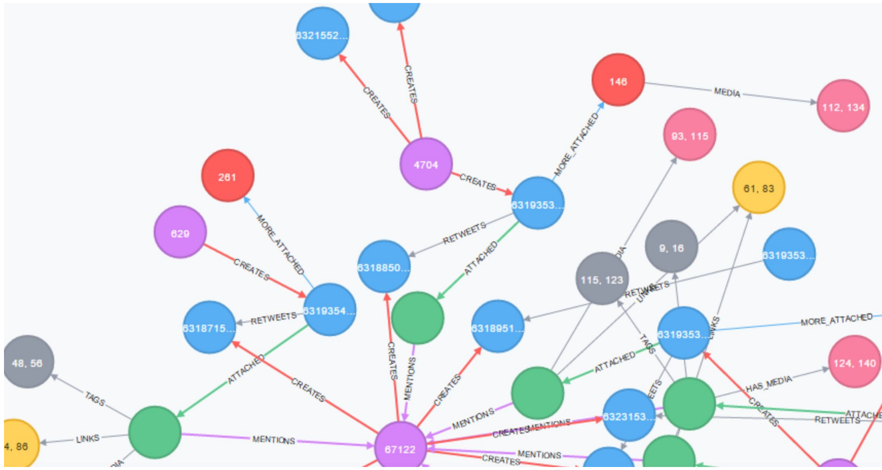
SPARK GRAPHX / GRAPHFRAMES



- Graph Computation
  - Distributed graph processing
  - Special structures for storing vertex and edge information & operations for manipulating graphs
  - GraphX (RDD-based) & GraphFrames (DF-based)
  - Has APIs in Scala, Java, Python (GraphFrames)

SPARK GRAPHX / GRAPHFRAMES

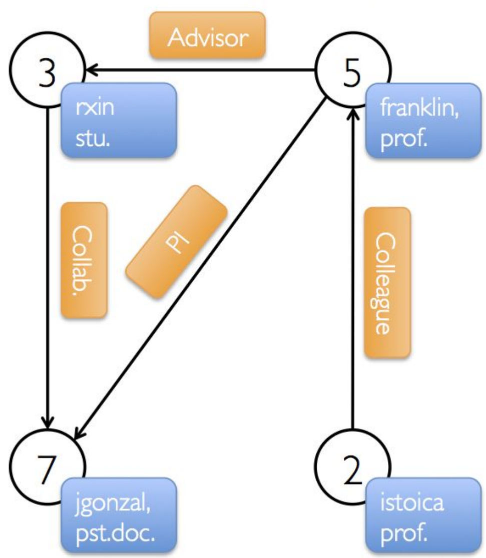
- Graph analytics
  - Analysis of relations among entities
- Data represented as graph
  - Entities are vertices
  - Relationships are edges
- Example: Analyzing tweets
  - Extract conversation threads
  - Find interacting groups
  - Find influencers in community



SPARK GRAPHX / GRAPHFRAMES

- Uses property graph model
  - o Both nodes and edges can have attributes and values
  - o Node/Edge properties stored in Vertex/Edge Table

Property Graph



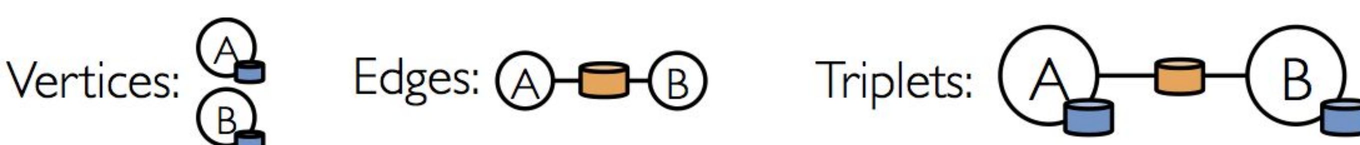
Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

- Triplets join vertex and edge properties

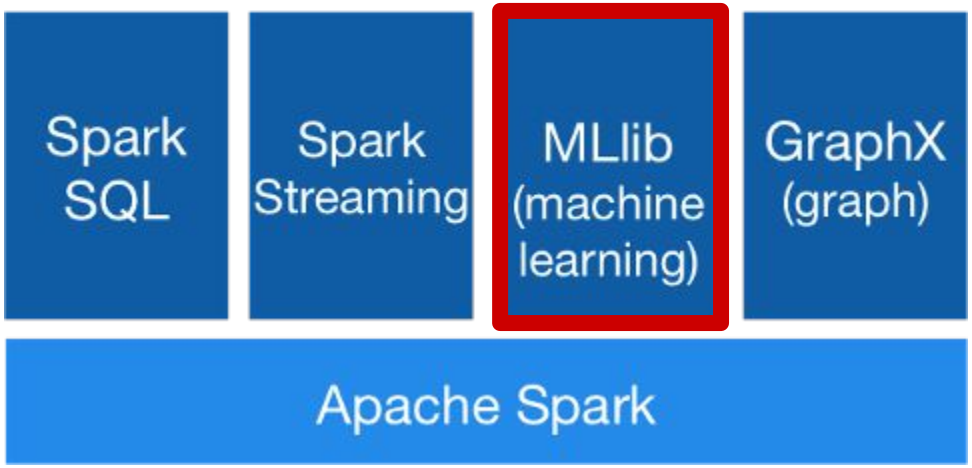


https://spark.apache.org/docs/latest/graphx-programming-guide.html

SPARK GRAPHX / GRAPHFRAMES

- Graph operators & algorithms
  - Connected Components
  - PageRank
  - Triangle Counting
  - Label Propagation Algorithm
  - Shortest Paths

SPARK MLLIB



- Machine Learning
  - Scalable machine learning library
  - Distributed implementations of machine learning algorithms and utilities
  - Has APIs for Scala, Java, Python, and R

SPARK MLLIB ALGORITHMS

- Machine Learning
  - Classification, regression, clustering, etc.
  - Evaluation metrics
- Statistics
  - Summary statistics, sampling, etc.
- Utilities
  - Dimensionality reduction, transformation, etc.
- ML Pipelines
  - Similar to scikit-learn

MLLIB EXAMPLE: STATISTICS

```
from pyspark.sql.functions import rand

# Generate random numbers
df = sqlContext.range(0,10)
      .withColumn("rand1", rand(seed=10))
      .withColumn("rand2", rand(seed=27))


# Show summary statistics
df.describe().show()


# Compute correlation
df.stat.corr("rand1","rand2")
```

MLLIB EXAMPLE: CLUSTER ANALYSIS

```
from pyspark.ml.clustering import KMeans

# Read and parse data
data = spark.read.csv("data.csv",inferSchema="true",
              header="true")


# k-means model for clustering
kmeans = Kmeans().setK(3).setSeed(123)
model = kmeans.fit (data)
for center in model.clusterCenters()
   print (center)
```

MLLIB EXAMPLE: CLASSIFICATION

```
from pyspark.ml.classification import DecisionTreeClassifier

# Split data into train & test sets
trainDF,testDF = data.randomSplit([0.7,0.3],seed=123)

# Build model
dt = DecisionTreeClassifier(
         featuresCol='features',
         labelCol='label',
         predictionCol='prediction')
model = dt.fit(trainDF)

# Test model
predictions = model.transform(testDF)
```

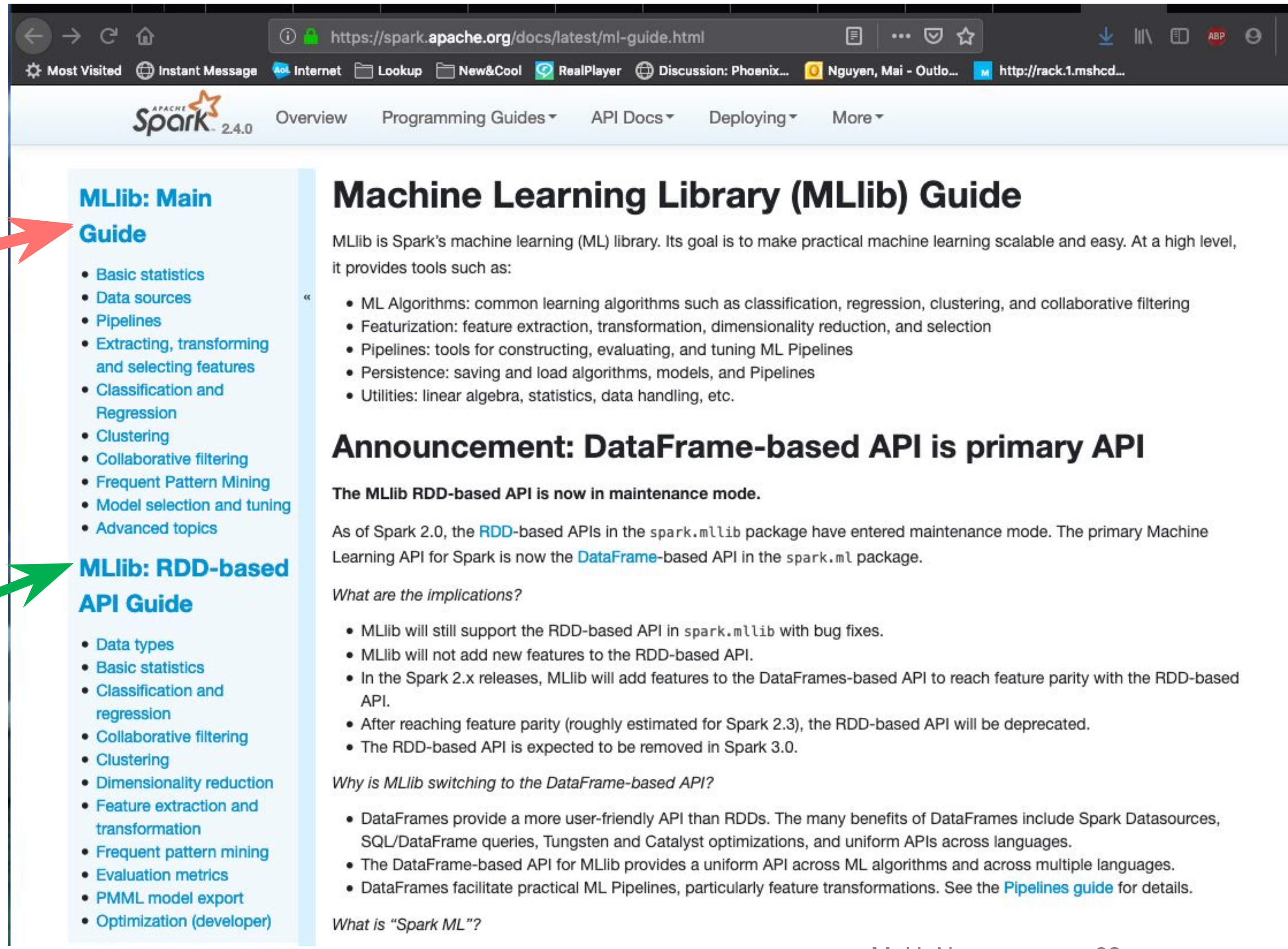
MLLIB LIBRARIES

ml

(DataFrame-based)

mllib

(RDD-based)



MLlib: Main Guide

- Basic statistics
- Data sources
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics

MLlib: RDD-based API Guide

- Data types
- Basic statistics
- Classification and regression
- Collaborative filtering
- Clustering
- Dimensionality reduction
- Feature extraction and transformation
- Frequent pattern mining
- Evaluation metrics
- PMML model export
- Optimization (developer)

Machine Learning Library (MLlib) Guide

MLlib is Spark's machine learning (ML) library. Its goal is to make practical machine learning scalable and easy. At a high level, it provides tools such as:

- ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering
- Featurization: feature extraction, transformation, dimensionality reduction, and selection
- Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
- Persistence: saving and load algorithms, models, and Pipelines
- Utilities: linear algebra, statistics, data handling, etc.

Announcement: DataFrame-based API is primary API

The MLlib RDD-based API is now in maintenance mode.

As of Spark 2.0, the RDD-based APIs in the `spark.mllib` package have entered maintenance mode. The primary Machine Learning API for Spark is now the DataFrame-based API in the `spark.ml` package.

What are the implications?

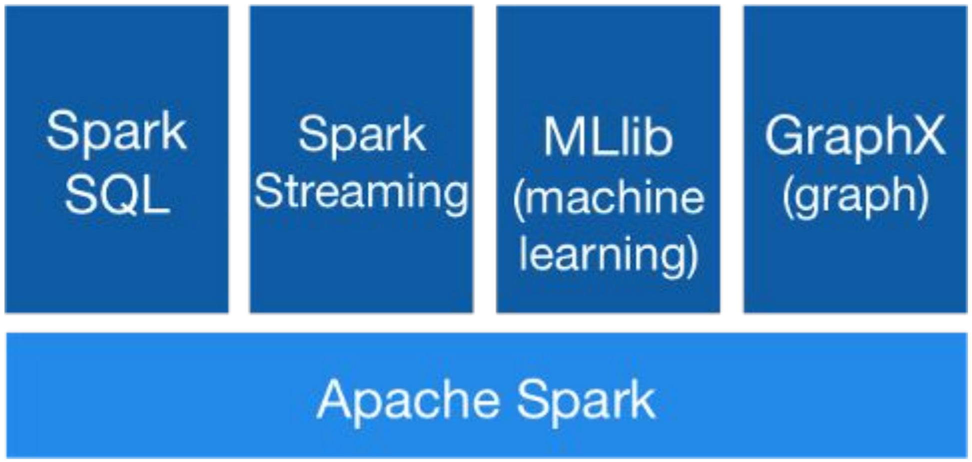
- MLlib will still support the RDD-based API in `spark.mllib` with bug fixes.
- MLlib will not add new features to the RDD-based API.
- In the Spark 2.x releases, MLlib will add features to the DataFrames-based API to reach feature parity with the RDD-based API.
- After reaching feature parity (roughly estimated for Spark 2.3), the RDD-based API will be deprecated.
- The RDD-based API is expected to be removed in Spark 3.0.

Why is MLlib switching to the DataFrame-based API?

- DataFrames provide a more user-friendly API than RDDs. The many benefits of DataFrames include Spark Datasources, SQL/DataFrame queries, Tungsten and Catalyst optimizations, and uniform APIs across languages.
- The DataFrame-based API for MLlib provides a uniform API across ML algorithms and across multiple languages.
- DataFrames facilitate practical ML Pipelines, particularly feature transformations. See the Pipelines guide for details.

What is "Spark ML"?

SPARK LIBRARIES



- Spark Libraries
  - Use Spark engine as core
  - Extend functionality to particular applications
  - Third-party packages: https://spark-packages.org

SPARK BEST PRACTICES

- Start small
  - Work with data sample first to work through pipeline & debugging
- Be careful with actions such as collect() and take()
  - This collects all data to Driver. Will get OOM if insufficient memory.
- Cache data that is frequently accessed
  - e.g., data in training set for machine learning
  - Note that DF is not fully cached until action is invoked
- Use built-in functions instead of UDF
  - Built-in functions are optimized

SPARK BEST PRACTICES

- Debugging
  - Use explain() to see plan for transformations for DF
    - e.g., df.sort("Name").explain()
  - Use Spark UI
    - List of scheduler stages and tasks
    - Summary of DF/RDD sizes and memory usage
    - DAG visualization
    - Information about environment
    - Information about driver and running executors
    - Can access on port 4040 (localhost:4040)
    - Information available only for duration of application

SPARK UI

Spark 2.4.3 | Jobs | Stages | Storage | Environment | Executors | SQL | **Word Count** application UI

Spark Jobs (?)

User: jovyan
Total Uptime: 25 min
Scheduling Mode: FIFO
Completed Jobs: 6

▸ Event Timeline

▾ Completed Jobs (6)

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
5	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2020/01/30 06:48:52	1 s	2/2	202/202
4	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2020/01/30 06:48:51	1.0 s	2/2	3/3
3	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2020/01/30 06:48:51	79 ms	1/1	1/1
2	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2020/01/30 06:48:50	76 ms	1/1	1/1
1	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2020/01/30 06:48:50	43 ms	1/1	1/1
0	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2020/01/30 06:48:49	0.4 s	1/1	1/1

For more details, see: https://spark.apache.org/docs/3.0.0-preview/web-ui.html

BIG DATA ANALYTICS

- Machine Learning Overview
- Data Exploration
- Data Preparation
- Modeling
- Spark Core & Libraries
- PySpark MlLib Examples
- PySpark Exercise
- Assignments
- Guest Lecture
- Project Proposal Presentations

START SPARK SESSION

Use * to use all available cores, or integer value to specify number of cores to use

```
import pyspark
from pyspark.sql import SparkSession

conf = pyspark.SparkConf().setAll([
        ('spark.master’,   'local[*]'),
        ('spark.app.name', 'PySpark Demo')])

spark = SparkSession.builder.config(conf=conf).getOrCreate()
```

Configuration parameters for Spark session

Get existing Spark session or create new one

LOAD DATA

- Loading data from local file system

```
df = spark.read.text("file:///<path>/<file>.txt")

df = spark.read.csv("file:///<path>/<file>.csv",
                    header=True).cache()
```

- Loading data from HDFS

```
df = spark.read.text("hdfs:///<path>/<file>.txt") \
                     .cache()

df = spark.read.csv("hdfs:///<path>/<file>.csv",
                     header=True,
                     interSchema=True).cache()
```

Indicates whether column headers exist

Automatically infer data types of columns

Cache data in memory

CHAINING

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

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

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

Line continuation indicator

Chaining: Making multiple method calls on same object

DROP ROWS WITH NULLS

- Drop rows with null values

```
df.dropna()
df.dropna(how='any')
df.dropna(how='all')
```

- Check number of rows before and after dropping rows

```
df.count()
```

FILL IN MISSING VALUES

- Replace null values with empty string

```
df.na.fill(' ')
```

- Count number of rows with nulls before and after filling nulls

```
df.count()
```

PARTITION DATA

- Partition available data into train and test data sets

```
train_df,test_df = df.randomSplit(0.8,0.2),seed=<seed>)
```

Percentage of samples for train dataset

Percentage of samples for test dataset

CREATE FEATURE VECTOR COLUMN

- Create feature vector column
  - Combines given list of columns into single vector column
  - To feed data to machine learning models

```
from pyspark.ml.feature import VectorAssembler

features  = ['air_temp','relative_humidity']
assembler = VectorAssembler(inputCols=features,
                            outputCol='featureVector')
features_df = assembler.transform(df)
features_df.show()
```

New column appended to features_df

```
air_temp|relative_humidity
62.96   | 63.9
```

```
air_temp|relative_humidity|featureVector
62.96   | 63.9            |[62.96, 63.9]
```

SCALE DATA

- Scale input data values
  - Standardize values to have zero mean and unit standard deviation
  - Each feature is scaled separately
  - Create scale transformer using train data, then apply to train/test data

```
from pyspark.ml.feature import StandardScaler
scaler = StandardScaler(inputCol="features_unscaled",
                        outputCol="features_scaled",
                        withStd=True, withMean=True)
scalerModel = scaler.fit(train_df)

train_df = scalerModel.transform(train_df)
test_df  = scalerModel.transform(test_df)
```

BUILD MODEL

- Build decision tree classifier
  - Create model
  - Use fit() to train model

```
from pyspark.ml.classification import DecisionTreeClassifier

dt = DecisionTreeClassifier(
         featuresCol='featureVector',
         labelCol='label',
         predictionCol='prediction')

dt_model = dt.fit(<train>)
```

Can specify name of columns for features, labels, and predictions

APPLY MODEL

- Apply trained model
    - Use transform()

```
predictions = <model>.transform(<data>)
```

Will have <data> and new ‘prediction’ column appended at the end

Trained model

Input data

EVALUATE MODEL

- Evaluator for classification model
  - Calculates F1, precision, recall, accuracy

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

mc_evaluator = MulticlassClassificationEvaluator(
   predictionCol=‘prediction’,
   labelCol=‘label’,
   metricName=‘f1’)

mc_evaluator.evaluate(<predictions>)
```

Column with predictions

Column with labels

Evaluation metric

Contains predictions and labels

SAVING DATAFRAME TO FILE

```
employees_sortedDF.coalesce(1).\
   write.csv("file:/<path>/employees.csv", \
                 header=True, mode="overwrite")
```

Save column headers

Overwrite existing file

DataFrame contents are coalesced into 1 partition and written to employees_sorted.csv/part-00000-*.csv

	name	dept	state	salary
1	Jane	Sales	WA	160000
2	Mary	Finance	NY	120000
3	James	Sales	CA	100000

SAVING DATAFRAME TO HDFS

```
employees_sortedDF.coalesce(1).\
   write.csv("hdfs:/<path>/employees.csv", \
                header=True, mode="overwrite")
```

DataFrame contents are coalesced into 1 partition and written to HDFS

	name	dept	state	salary
1	Jane	Sales	WA	160000
2	Mary	Finance	NY	120000
3	James	Sales	CA	100000

BIG DATA ANALYTICS

- Machine Learning Overview
- Data Exploration
- Data Preparation
- Modeling
- Spark Core & Libraries
- PySpark MlLib Examples
- PySpark Exercise
- Assignments
- Guest Lecturer
- Project Proposal Presentations

BIG DATA ANALYTICS

- Machine Learning Overview
- Data Exploration
- Data Preparation
- Modeling
- Spark Core & Libraries
- PySpark MlLib Examples
- PySpark Exercise
- Assignments
- Guest Lecture
- Project Proposal Presentations

SESSION 3 ASSIGNMENTS

- Programming Assignment
  - Regression analysis on Housing data
    - Predict median value of homes
    - Prepare data, build models, print metrics, plot results
    - Skeleton notebook on Canvas
    - Data file: Boston_Housing.csv (on Canvas)
    - Use PySpark DataFrame
  - Submit
    - Jupyter notebook (.ipynb)
    - Python script (.py)
  - Due Friday 2021-05-14 at 11:59pm Pacific Time
- Project
  - Continue to work on requirements in Project Description doc

BIG DATA ANALYTICS

- Machine Learning Overview
- Data Exploration
- Data Preparation
- Modeling
- Spark Core & Libraries
- PySpark MlLib Examples
- PySpark Exercise
- Assignments
- Guest Lecture
- Project Proposal Presentations

GUEST LECTURE

Peter Rose, Ph.D.

Director of Structural Bioinformatics Lab, SDSC at UCSD

Scalable, Interactive, and Reproducible Data Mining of 3D Macromolecular Structures

Protein Data Bank (PDB) (https://www.rcsb.org) collects and curates the 3D shapes of proteins, nucleic acids, and complex assemblies that helps students and researchers understand all aspects of biomedicine from protein synthesis to health and disease. The rapid growth of the PDB (> 170,000 structures) enables large-scale data mining, such as the development of knowledge-based potentials, docking and scoring functions, and machine learning for protein structure and function prediction.

The interactive analysis of the PDB archive, working with thousands of individual data files, is not scalable due to the large I/O and parsing overhead. We present our approach of using Apache Spark and columnar data formats to scale structural analysis to enable the interactive exploration of the PDB archive, as well as scalable data integration.

BIG DATA ANALYTICS

- Machine Learning Overview
- Data Exploration
- Data Preparation
- Modeling
- Spark Core & Libraries
- PySpark MlLib Examples
- PySpark Exercise
- Assignments
- Guest Lecture
- Project Proposal Presentations

PROJECT PROPOSAL PRESENTATIONS

- 10 minutes: 8 minutes for presentation + 2 minutes for Q&A
- All team members must present
- To include in your presentation: problem to address, dataset, analysis task planned, insights you hope to gain, and potential challenges
- Q&A: At least 2 questions from other teams
- Presentation order: https://piazza.com/class/kmpqua0caj53vr?cid=31

SPARK RESOURCES

- PySpark SQL Basics Cheat Sheet
  - PDF on Canvas
- Spark Main Page
  - https://spark.apache.org/
- Spark Overview
  - https://spark.apache.org/docs/latest/index.html
- Spark Examples
  - https://spark.apache.org/examples.html
- Spark SQL, DataFrames and DataSets Programming Guide
  - https://spark.apache.org/docs/latest/sql-programming-guide.html
- Spark MLlib Programming Guide
  - https://spark.apache.org/docs/latest/ml-guide.html
- PySpark API Documentation
  - https://spark.apache.org/docs/latest/api/python/index.html
- Note: Spark version 3.1.1, Python, DataFrame API