Distributed Machine Learning with Apache Spark

DataFrames & Machine Learning Library (MLlib)

Dr. Salman Salloum
Research Fellow, NUS School of Computing

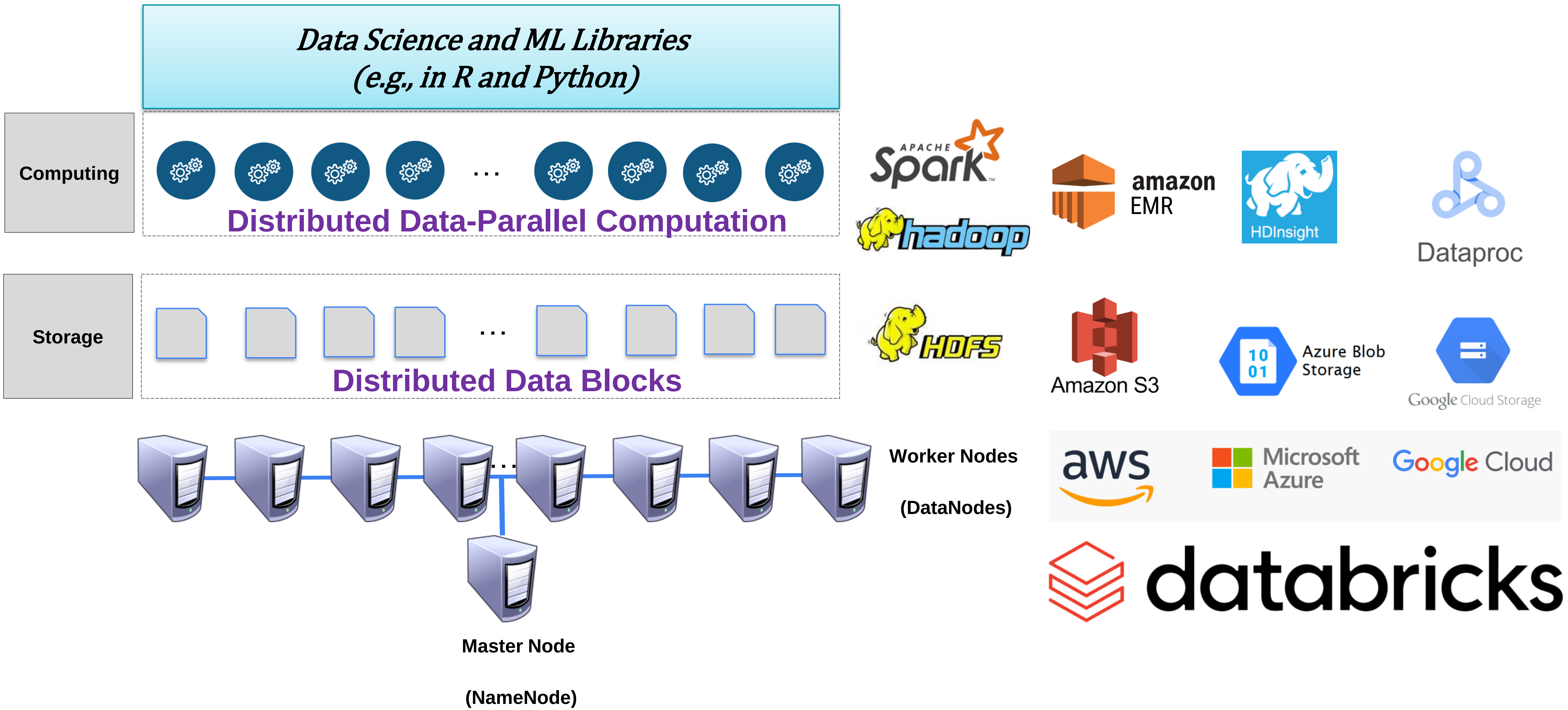
Wednesday, 16 August 2023

Agenda

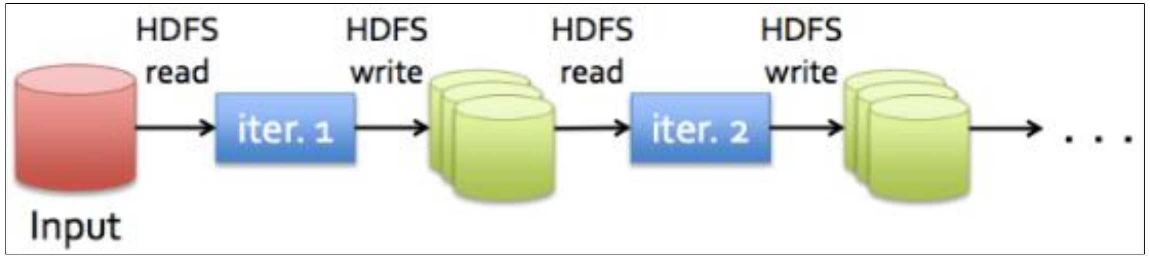
<u>**Lecture 1:**</u> • **Overview of Apache Spark** • **Spark DataFrames**	**9:30 - 10:30**
Break 1	**10:30 - 11: 00**
<u>**Lecture 2:**</u> • **Spark Machine Learning Library (MLlib)** • **Distributed ML Algorithms** • **ML Pipelines API**	**11:00 - 13:00**
Break 2	**13:00 - 14:00**
<u>**Lab 1:**</u> • **PySpark, DataFrames and MLlib**	**14:00 - 15:30**
Break 3	**15:30 - 16:00**
<u>**Lab 2:**</u> • **ML Pipelines**	**16:00 - 17:30**

Overview of Apache Spark

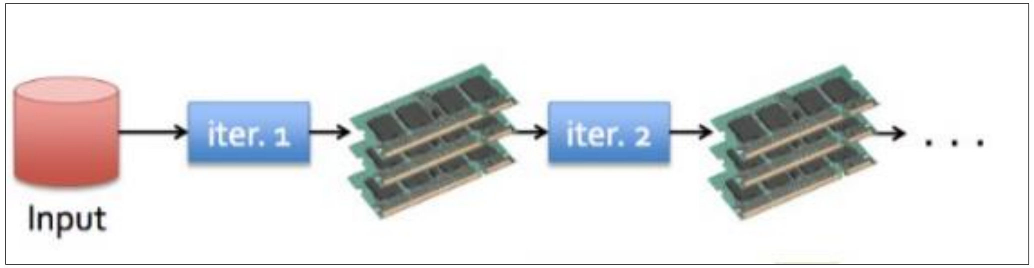
Big Data Platforms



Spark at UC Berkeley (AMPLab)



Iterative processing with Hadoop MapReduce



Iterative processing with Spark

Spark: Cluster Computing with Working Sets

Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica
University of California, Berkeley

Abstract

MapReduce and its variants have been highly successful in implementing large-scale data-intensive applications on commodity clusters. However, most of these systems are built around an acyclic data flow model that is not suitable for other popular applications. This paper focuses on one such class of applications: those that reuse a working set of data across multiple parallel operations. This includes many iterative machine learning algorithms, as well as interactive data analysis tools. We propose a new framework called Spark that supports these applications while retaining the scalability and fault tolerance of MapReduce. To achieve these goals, Spark introduces an abstraction called resilient distributed datasets (RDDs). An RDD is a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Spark can outperform Hadoop by 10x in iterative machine learning jobs, and can be used to interactively query a 39 GB dataset with sub-second response time.

MapReduce/Dryad job, each job must reload the data from disk, incurring a significant performance penalty.

- **Interactive analytics**: Hadoop is often used to run ad-hoc exploratory queries on large datasets, through SQL interfaces such as Pig [21] and Hive [1]. Ideally, a user would be able to load a dataset of interest into memory across a number of machines and query it repeatedly. However, with Hadoop, each query incurs significant latency (tens of seconds) because it runs as a separate MapReduce job and reads data from disk.

This paper presents a new cluster computing framework called Spark, which supports applications with working sets while providing similar scalability and fault tolerance properties to MapReduce.

The main abstraction in Spark is that of a *resilient distributed dataset* (RDD), which represents a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Users can explicitly cache an RDD in memory across machines and reuse it

https://spark.apache.org/research.html

Apache Spark

- 2013: The AMPLab contributed Spark to the Apache Software Foundation and launched Databricks
- 2014: Spark 1.0
- 2016: Spark 2.0
- 2020: Spark 3.0
- June 2023: Spark 3.4.1

SIGMOD Systems Award for Apache Spark

Apache Spark received the SIGMOD Systems Award this year, given by SIGMOD (the ACM's data management research organization) to impactful real-world and research systems:

> The 2022 ACM SIGMOD Systems Award goes to "Apache Spark", an innovative, widely-used, open-source, unified data processing system encompassing relational, streaming, and machine-learning workloads.

https://spark.apache.org/news/sigmod-system-award.html

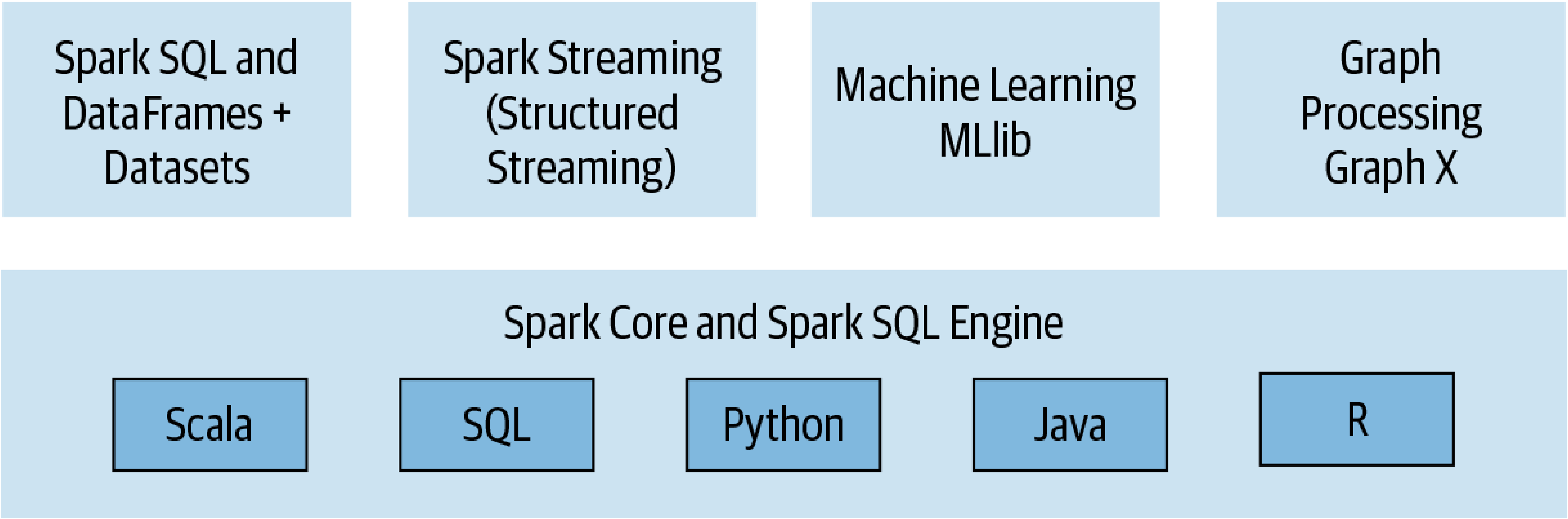
DOI:10.1145/2934664

This open source computing framework unifies streaming, batch, and interactive big data workloads to unlock new applications.

BY MATEI ZAHARIA, REYNOLD S. XIN, PATRICK WENDELL, TATHAGATA DAS, MICHAEL ARMBRUST, ANKUR DAVE, XIANGRUI MENG, JOSH ROSEN, SHIVARAM VENKATARAMAN, MICHAEL J. FRANKLIN, ALI GHODSI, JOSEPH GONZALEZ, SCOTT SHENKER, AND ION STOICA

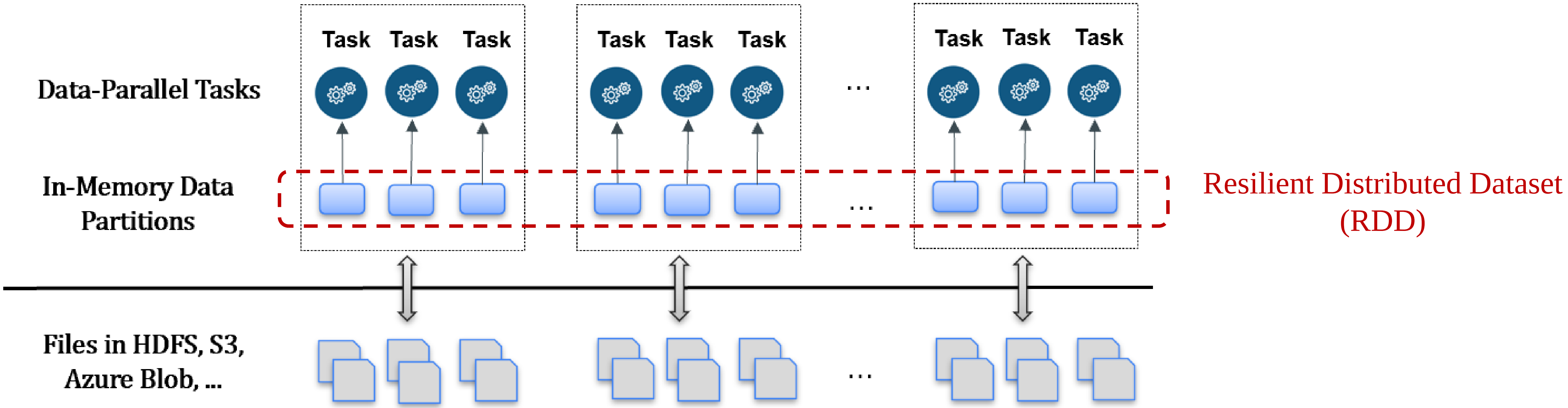
Apache Spark: A Unified Engine for Big Data Processing

Apache Spark Components



Apache Spark is a unified computing engine designed for large-scale distributed data processing, on single-node machines, on-premises clusters, or in the cloud.

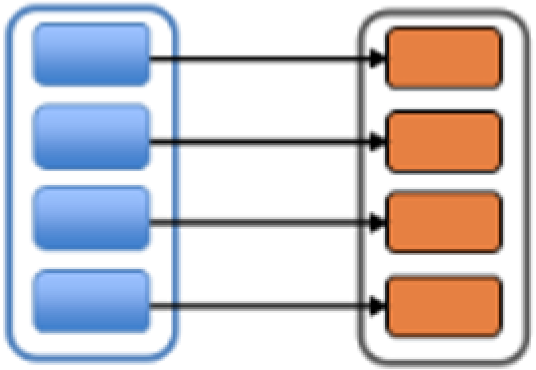
In-Memory Distributed Data Processing



Transformations and Actions

- Transformations are simple ways of specifying a series of data manipulation.
- Two types of transformations: narrow transformations, and wide transformations.
- Spark computes transformations only in a lazy fashion.
- Actions trigger the computation in Spark.
- Actions can be used to view/collect/write the output data after a series of transformations.

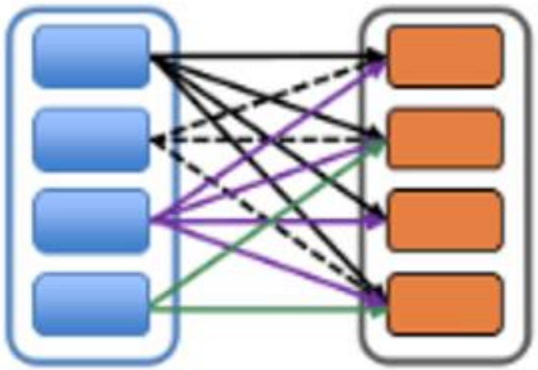
Narrow Dependencies



Each input partition contributes to only one output partition

```
map()
filter()
select()
mapPartition()
```

Wide Dependencies

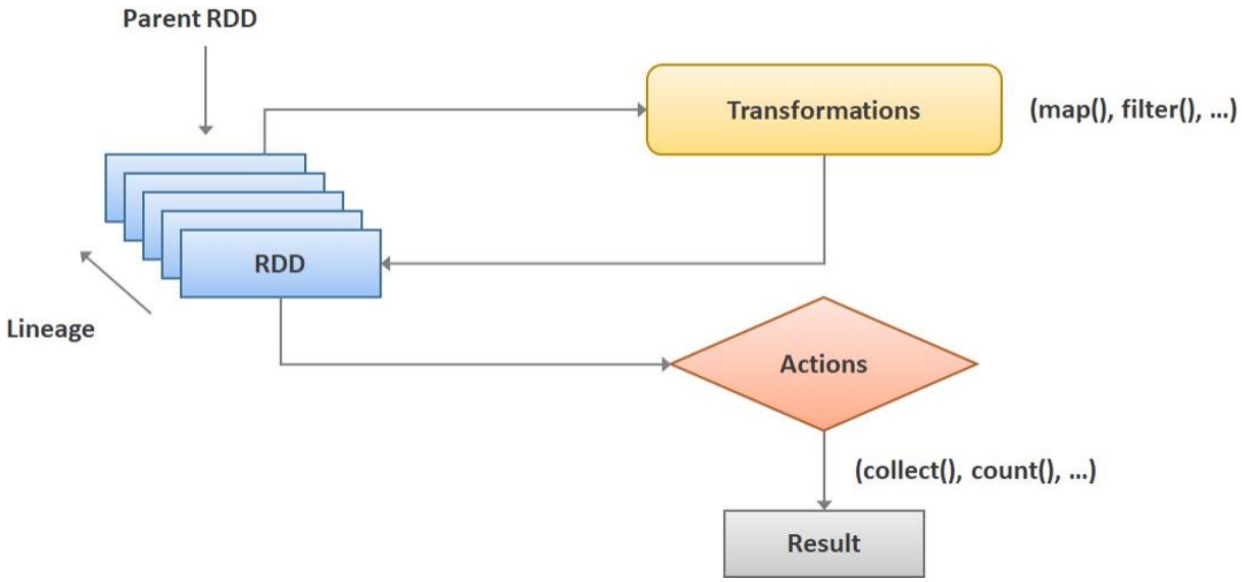


Each input partition may contribute to many output partitions

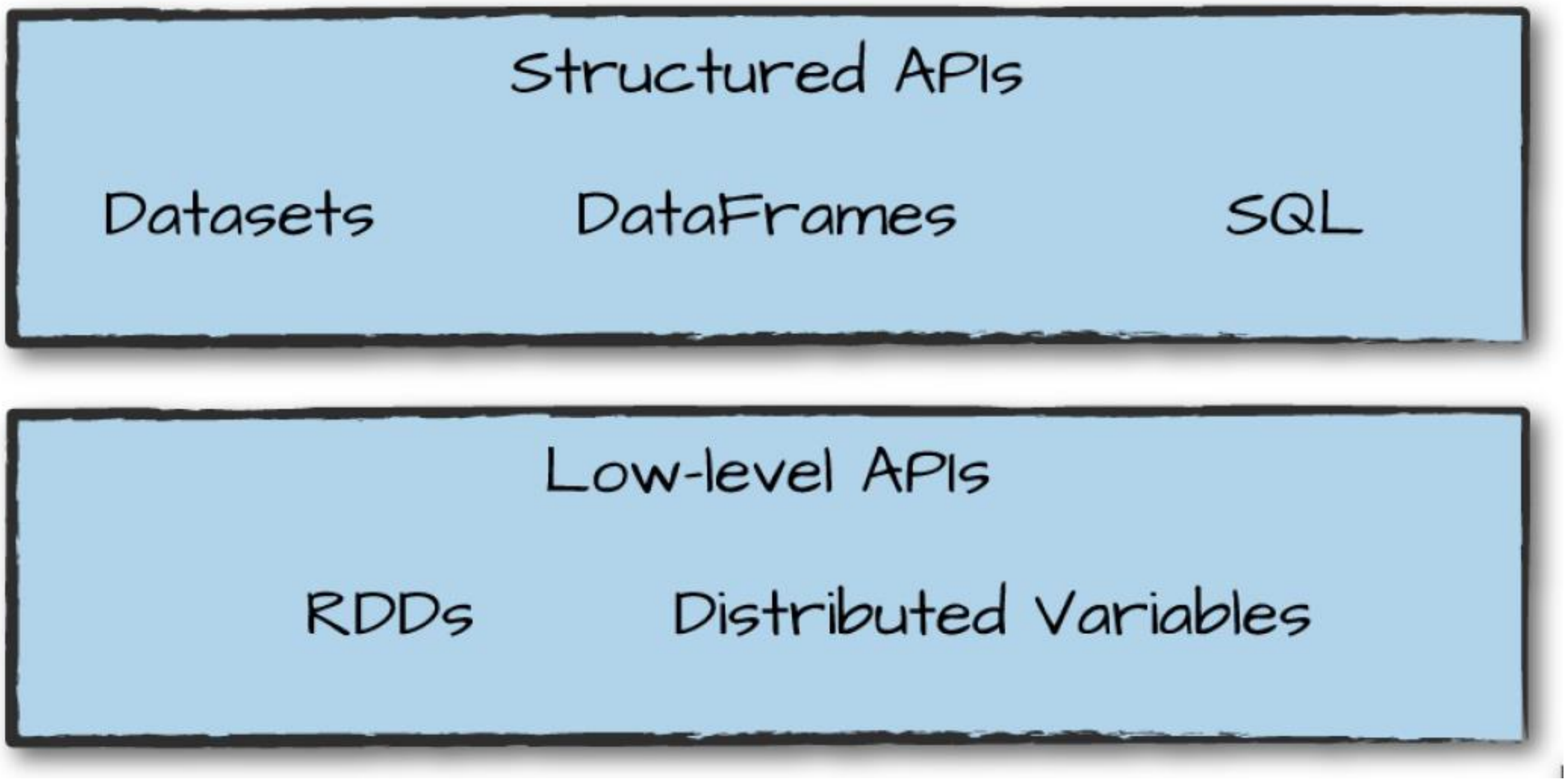
```
groupByKey()
join()
sort()
repartition()
```

Lazy Evaluation

- **Lazy evaluation is Spark's strategy for delaying execution until an action is invoked.**
- **This allows spark to optimize the execution plan.**
- **Transformations are recorded in a lineage.**
- **Lineage and immutability provide fault- tolerance.**



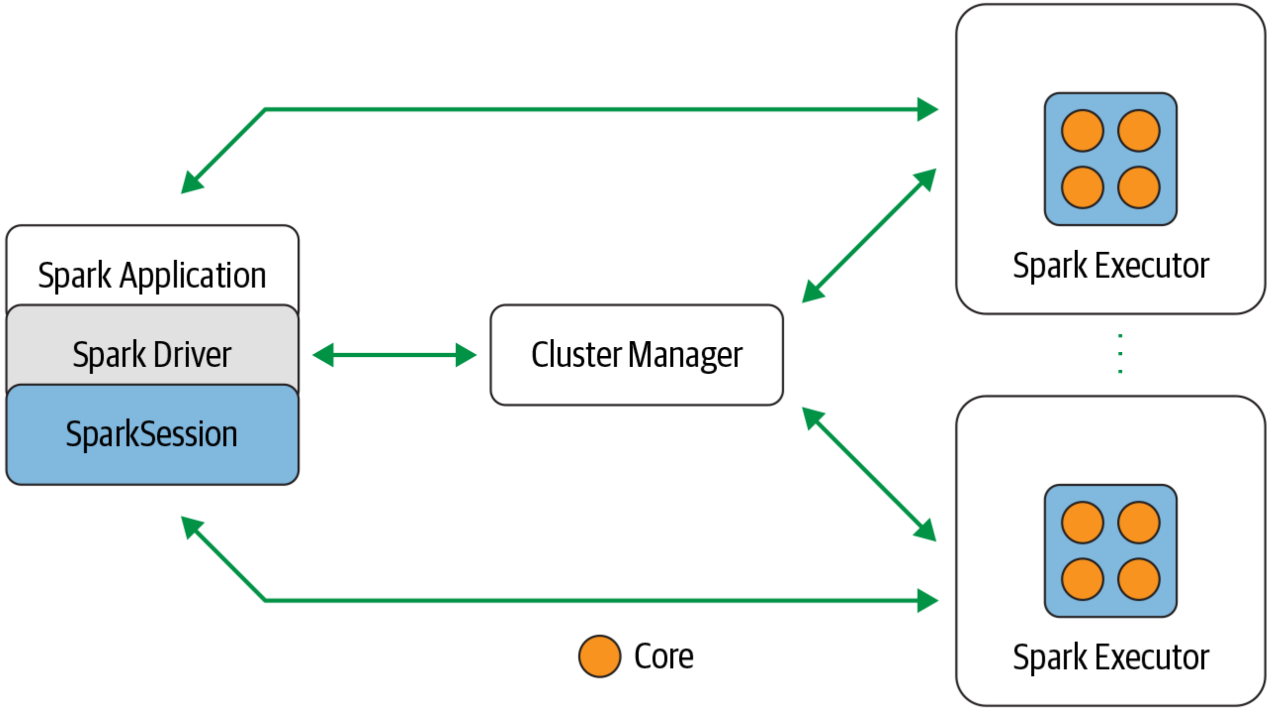
Spark APIs



Structured APIs
Datasets
DataFrames
SQL
Low-level APIs
RDDs
Distributed Variables

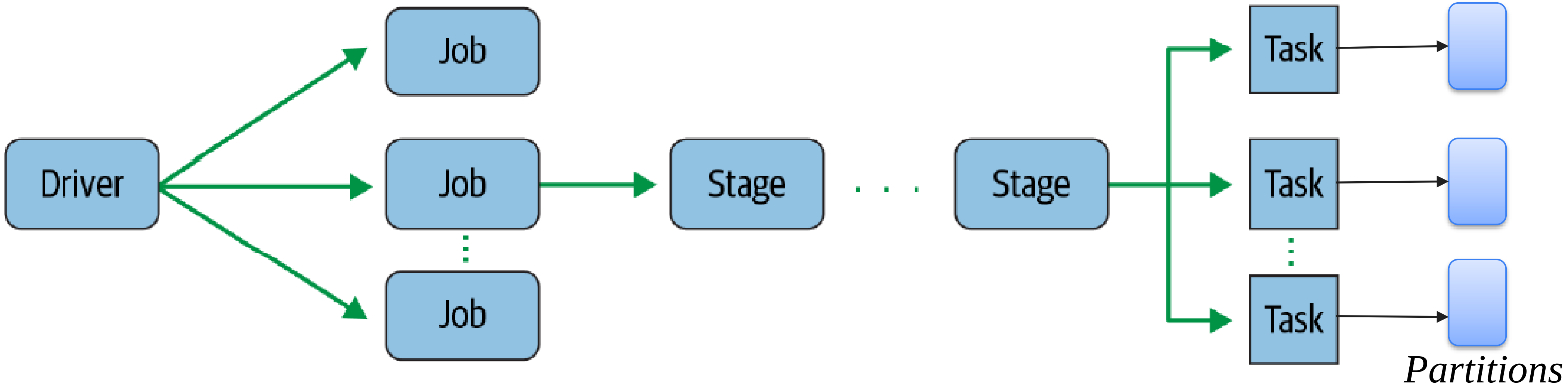
Spark's Distributed Architecture

- **Spark Driver: orchestrating parallel operations on the Spark cluster.**
- **Spark Session: a unified entry point to all Spark functionality.**
- **Spark Executors: running the tasks assigned by the driver.**
- **Cluster Manager: allocating resources to Spark Applications.**

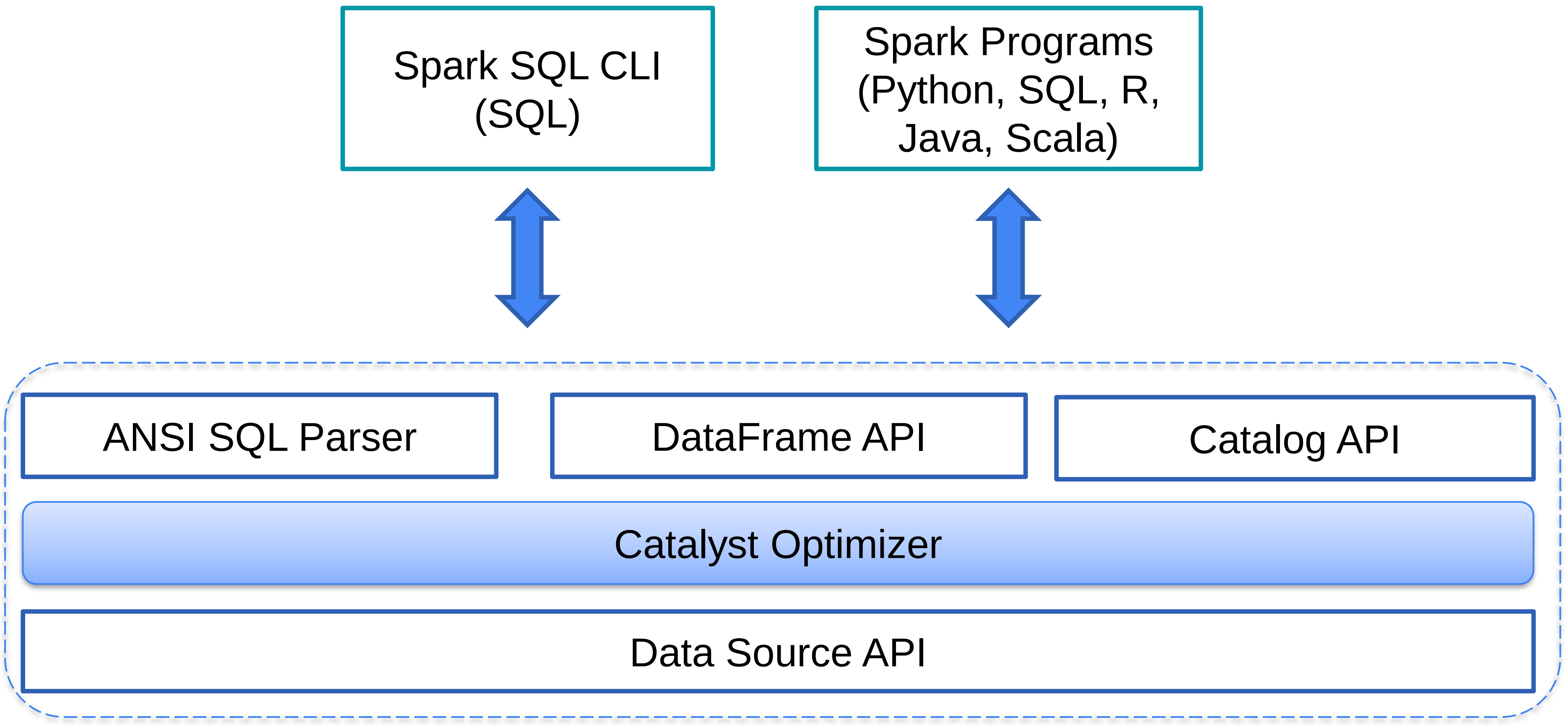


Anatomy of a Spark Application

- **Each Spark Application is made up of one or more Spark Jobs.**
- **A Spark Job is a parallel computation that is triggered in response to a Spark action.**
- **Each job gets divided into smaller stages that depend on each other.**
- **Each Stage consists of a set of tasks**
- **Spark Task is a single unit of work or execution that will be sent to a Spark Executor.**
- **Each Spark Task is assigned a data partition**

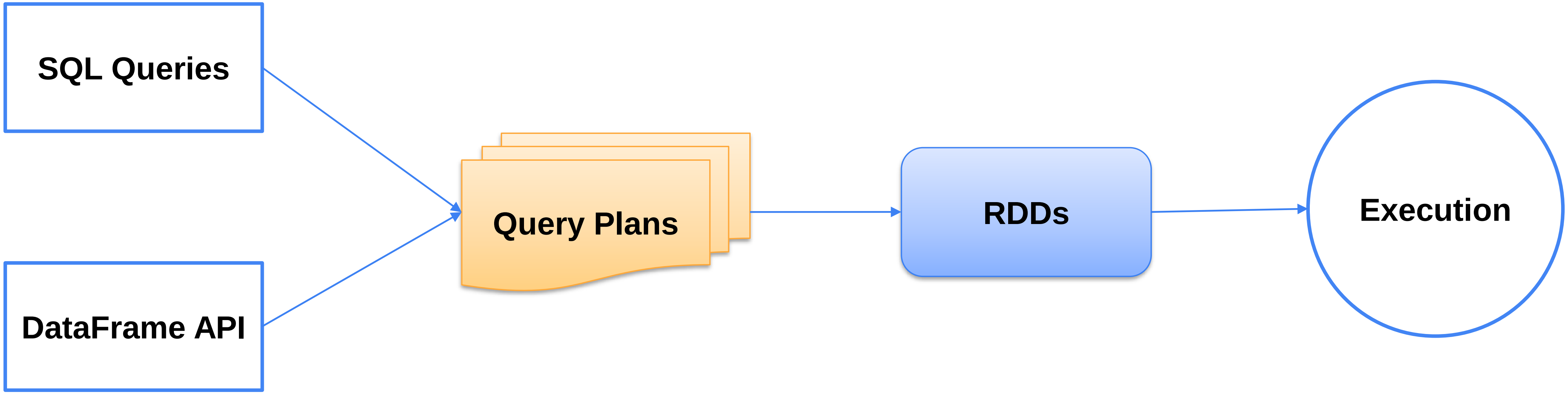


Spark SQL: Main Components



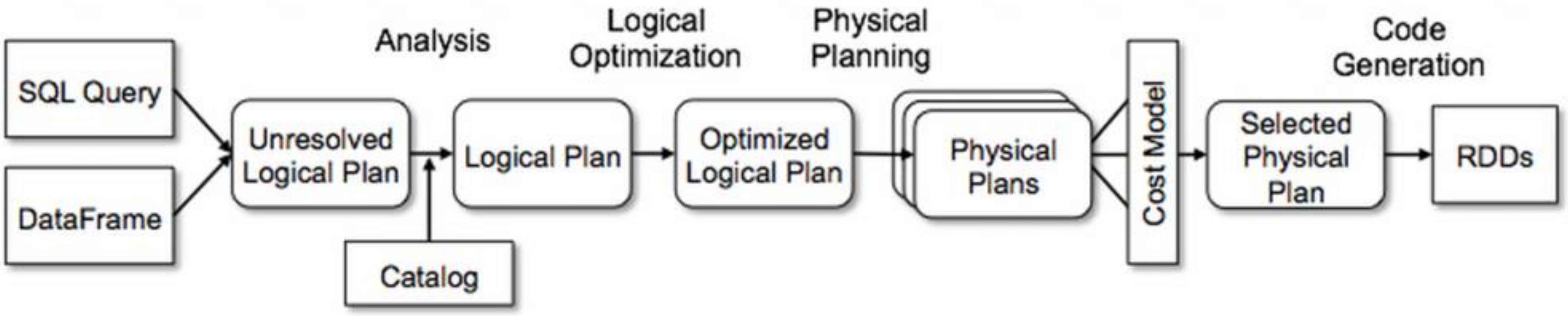
Spark SQL: Execution

- **All queries/operations are optimized and executed in the same way**



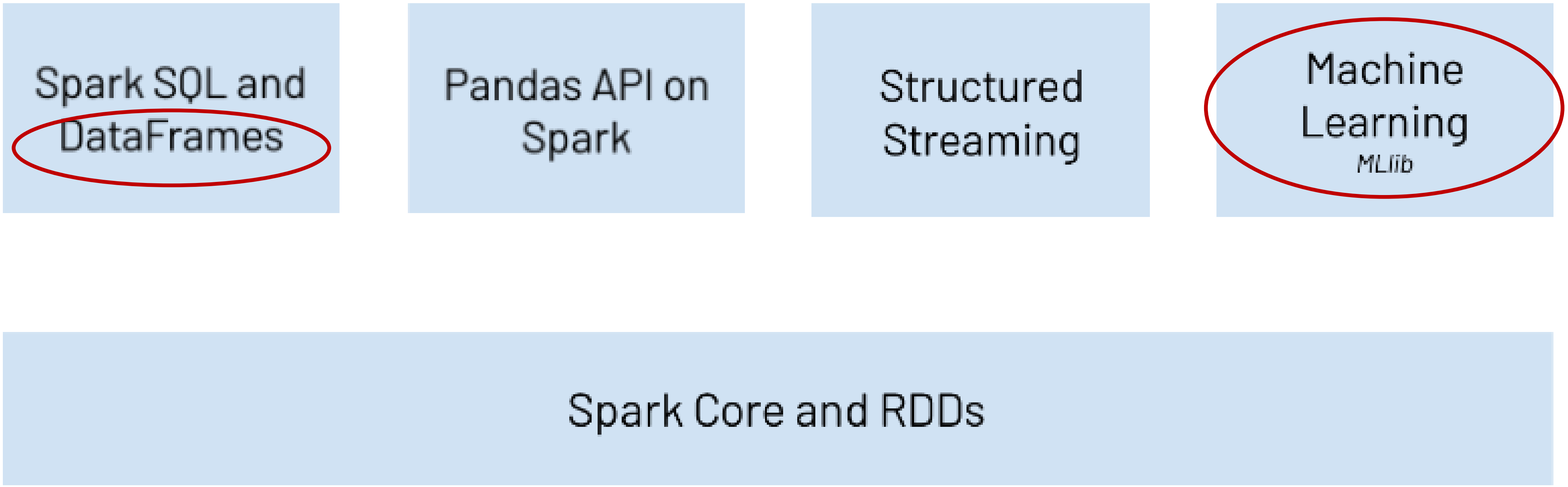
Spark SQL: The Catalyst Optimizer

- **It converts queries/DataFrame operations into an execution plan**



https://www.databricks.com/glossary/catalyst-optimizer

PySpark: The Python API for Apache Spark



https://spark.apache.org/docs/latest/api/python/index.html

PySpark: The Python API for Apache Spark

- Install as any Python package
- PySpark Shell

```
!pip install pyspark
```

SparkSession

```
[ ]  # Import SparkSession
     from pyspark.sql import SparkSession

     # Create a Spark Session
     spark = SparkSession.builder\
             .master("local[2]")\
             .appName("Hello Spark")\
             .config('spark.ui.port', '4050')\
             .getOrCreate()
```

https://spark.apache.org/downloads.html
https://spark.apache.org/docs/latest/quick-start.html

Spark in the Cloud

- **Managed Spark environments**

amazon EMR
HDInsight
Dataproc
aws
Microsoft Azure
Google Cloud
databricks

- **Spark is a core computing engine in the Lakehouse Architecture**

Spark Ecosystem

Data science and Machine learning

scikit learn
pandas
TensorFlow
PyTorch
mlflow
R
NumPy

SQL analytics and BI

Apache Superset
Power BI
Looker
redash
tableau software
dbt

Storage and Infrastructure

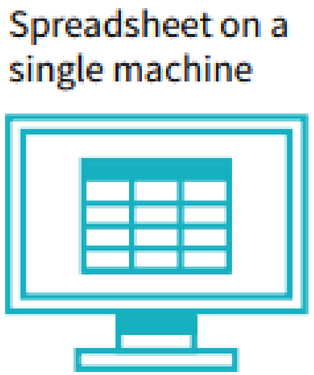
elasticsearch
mongoDB.
Apache kafka.
DELTA LAKE
kubernetes
Apache Airflow
Parquet
Microsoft SQL Server
cassandra
Apache ORC

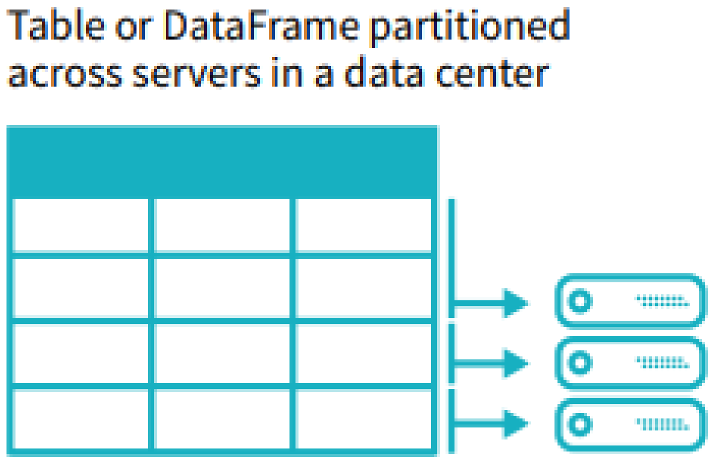
https://spark.apache.org/

Spark DataFrames

DataFrames: Distributed Data

- **A Spark DataFrame is a distributed collection of data organized into named columns.**
- **It is conceptually equivalent to a table in a relational database or a data frame in R/Python.**
- **It is divided into chunks of data called *partitions.***
- **A partition is a set of rows that sit on one machine.**
- **A DataFrame is like RDD, but it has a structure**
- **DataFrames benefit from the Catalyst optimizer**





DataFrames: Schema

```
+-------+---------+-------+-----+
|Book_Id|Book_Name| Author|Price|
+-------+---------+-------+-----+
|      1|      PHP| Sravan|  250|
|      2|      SQL|Chandra|  300|
|      3|   Python| Harsha|  250|
|      4|        R| Rohith| 1200|
|      5|   Hadoop| Manasa|  700|
+-------+---------+-------+-----+
```

```
root
 |-- Book_Id: long (nullable = true)
 |-- Book_Name: string (nullable = true)
 |-- Author: string (nullable = true)
 |-- Price: long (nullable = true)
```

- **A Schema defines the column names and types of a DataFrame**
- **Infer schema (e.g., JSON, CSV)**
- **Extract schema (e.g., Parquet)**
- **Define schema programmatically**
    - **Schema is a StructType: it can be saved as JSON and loaded later**
- **It is advisable to save DataFrames in structured formats that keeps the schema (e.g., parquet) for future use**

DataFrames: Spark Data Types

- **Spark has its own custom data types to facilitate efficient distributed data processing (with the Catalyst optimizer)**
- **Each Spark data type is mapped into Python data type**
- **Same Python data type may be used to represent several Spark types (but representing different number of bytes)**

Value in Python	Spark Data Type
int (1 byte)	ByteType
int (2 byte)	ShortType
int (4 byte)	IntegerType
int (8 byte)	LongType

https://spark.apache.org/docs/latest/sql-ref-datatypes.html

pyspark.sql.types

```
ByteType
ShortType
IntegerType
LongType
FloatType
DoubleType
DecimalType
StringType
BinaryType
BooleanType
TimestampType
TimestampNTZType
DateType
DayTimeIntervalType
ArrayType
MapType
StructType
StructField
```

DataFrames: Transformations and Actions

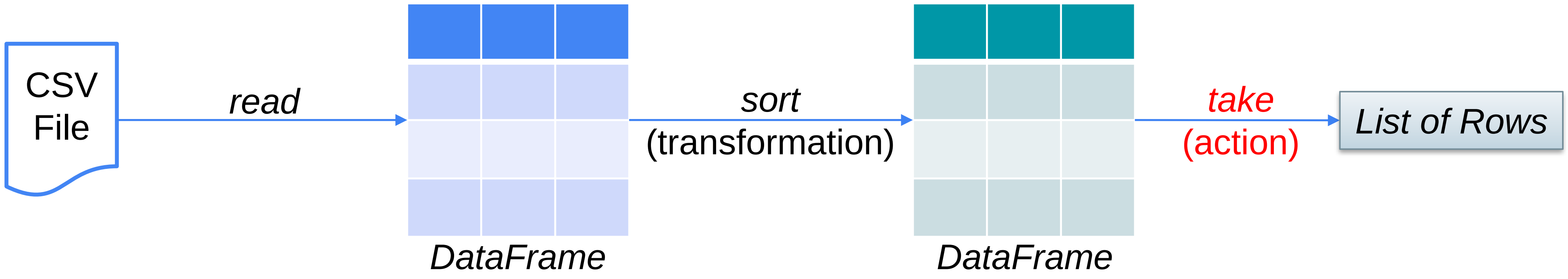
- **DataFrame transformations are methods that return a new DataFrame and are lazily evaluated.**
- **DataFrame actions are methods that trigger computation.**
- **An action is needed to trigger the execution of any DataFrame transformation.**

```
df.select("id", "result")
  .where("result > 70")
  .orderBy("result")
```

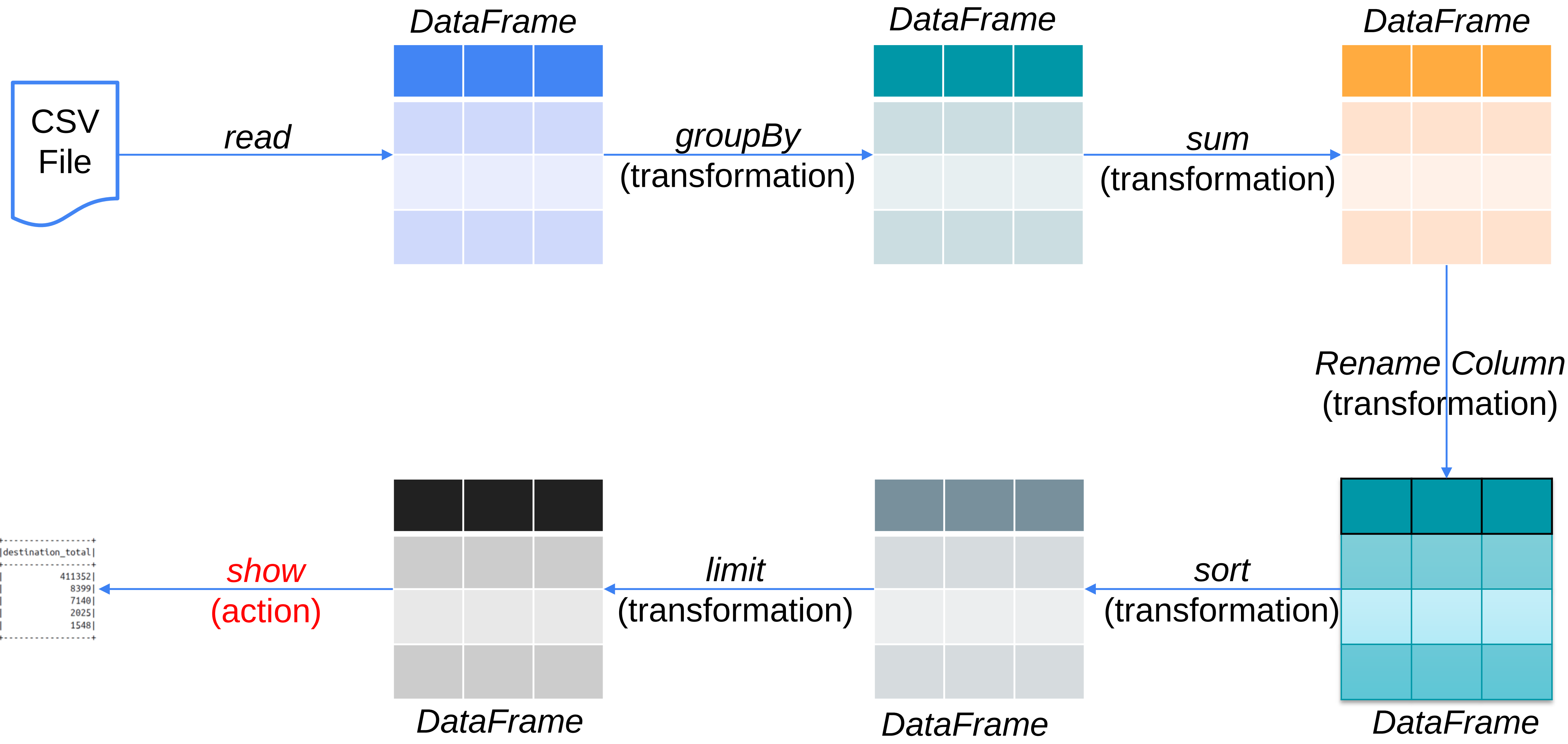
```
df.count()
df.collect()
df.show()
```

```
df.select("id", "result")
  .where("result > 70")
  .orderBy("result")
  .show()
```

DataFrames: Transformations and Actions



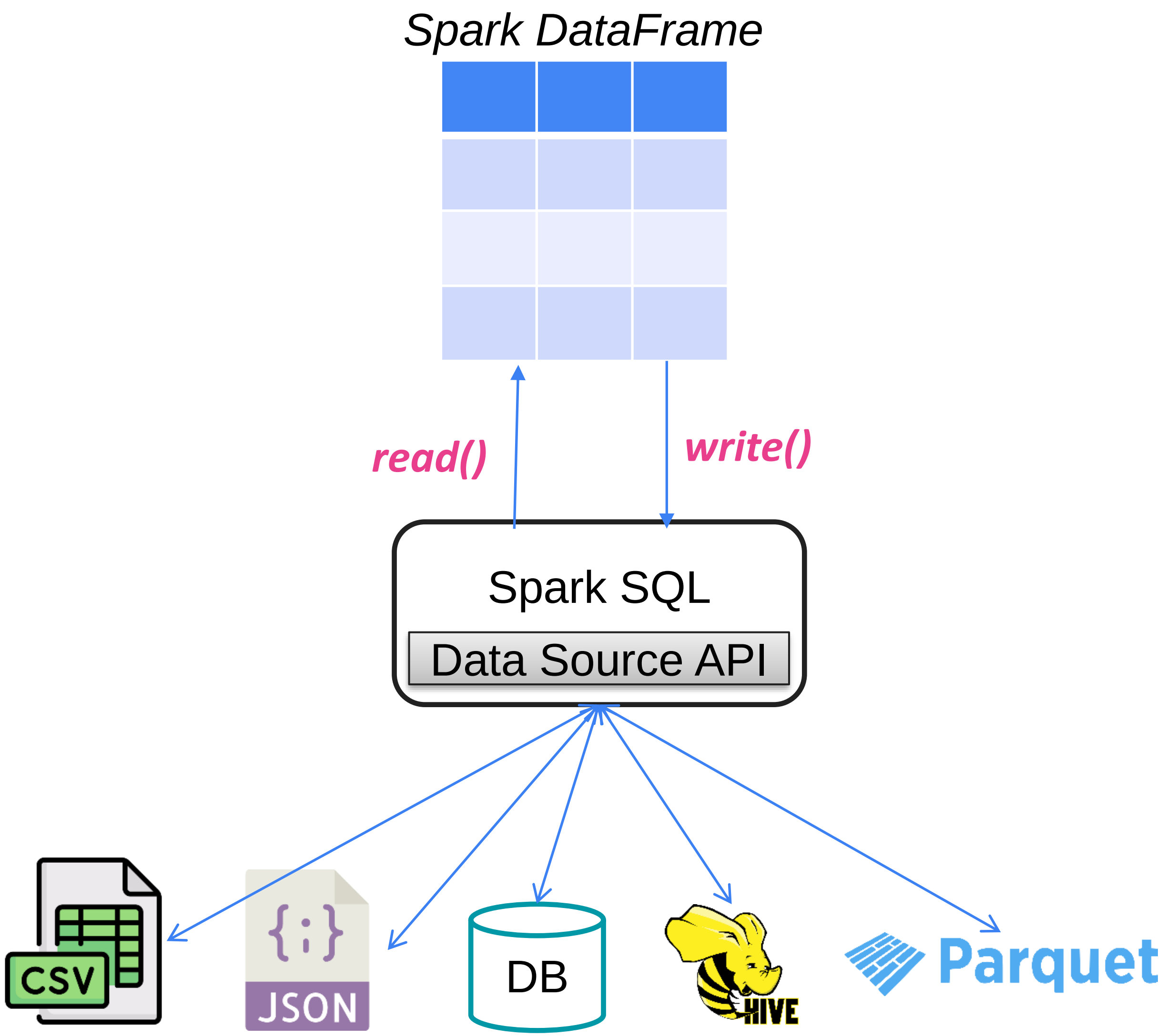
DataFrames: Transformations and Actions



DEST_COUNTRY_NAME	destination_total
United States	411352
Canada	8399
Mexico	7140
United Kingdom	2025
Japan	1548

Data Sources

- Spark SQL supports many data sources
- The default data source in Spark is Parquet (a columnar data file format)
- Image data source
  - Each image is represented as BinaryType with additional columns
- Binary file data source
  - Each binary file is represented as a BinaryType with additional columns



DataFrameReader

- **Reading DataFrames**
  - pyspark.sql.DataFrameReader
  - pyspark.sql.SparkSession.read

Method	Arguments	Description
format()	"parquet", "csv", "txt", "json", "jdbc", "orc", "avro", etc.	If you don't specify this method, then the default is Parquet or whatever is set in spark.sql.sources.default.
option()	("mode", {PERMISSIVE | FAILFAST | DROPMALFORMED }) ("inferSchema", {true | false}) ("path", "path_file_data_source")	A series of key/value pairs and options. The Spark documentation shows some examples and explains the different modes and their actions. The default mode is PERMISSIVE. The "inferSchema" and "mode" options are specific to the JSON and CSV file formats.
schema()	DDL String or StructType, e.g., 'A INT, B STRING' or StructType(...)	For JSON or CSV format, you can specify to infer the schema in the option() method. Generally, providing a schema for any format makes loading faster and ensures your data conforms to the expected schema.
load()	"/path/to/data/source"	The path to the data source. This can be empty if specified in option("path", "...").

- **Directly load DataFrames from data sources (csv, parquet, json, …)**

```
spark.read.parquet("/mnt/training/ecommerce/events.parquet")
```

DataFrameWriter

- **Writing DataFrames**
  - pyspark.sql.DataFrameWriter
  - pyspark.sql.DataFrame.write

Method	Arguments	Description
format()	"parquet", "csv", "txt", "json", "jdbc", "orc", "avro", etc.	If you don't specify this method, then the default is Parquet or whatever is set in spark.sql.sources.default.
option()	("mode", {append | overwrite | ignore | error or errorifexists}) ("mode", {SaveMode.Overwrite | SaveMode.Append, SaveMode.Ignore, SaveMode.ErrorIfExists}) ("path", "path_to_write_to")	A series of key/value pairs and options. The Spark documentation shows some examples. This is an overloaded method. The default mode options are error or errorifexists and SaveMode.ErrorIfExists; they throw an exception at runtime if the data already exists.
bucketBy()	(numBuckets, col, col..., coln)	The number of buckets and names of columns to bucket by. Uses Hive's bucketing scheme on a filesystem.
save()	"/path/to/data/source"	The path to save to. This can be empty if specified in option("path", "...").
saveAsTable()	"table_name"	The table to save to.

- **Directly write DataFrames to data sources (csv, parquet, json, …)**

```
(df.write
  .option("compression", "snappy")
  .mode("overwrite")
  .parquet(outPath)
)
```

Spark Machine Learning Library (MLlib)

Spark MLlib

- Spark’s scalable machine learning library
- Work with bigger data and train models faster

MLlib: Machine Learning in Apache Spark

Xiangrui Meng† MENG@DATABRICKS.COM
Databricks, 160 Spear Street, 13th Floor, San Francisco, CA 94105
Joseph Bradley JOSEPH@DATABRICKS.COM
Databricks, 160 Spear Street, 13th Floor, San Francisco, CA 94105
Burak Yavuz BURAK@DATABRICKS.COM
Databricks, 160 Spear Street, 13th Floor, San Francisco, CA 94105
Evan Sparks SPARKS@CS.BERKELEY.EDU
UC Berkeley, 465 Soda Hall, Berkeley, CA 94720
Shivaram Venkataraman SHIVARAM@EECS.BERKELEY.EDU
UC Berkeley, 465 Soda Hall, Berkeley, CA 94720
Davies Liu DAVIES@DATABRICKS.COM
Databricks, 160 Spear Street, 13th Floor, San Francisco, CA 94105
Jeremy Freeman FREEMANJ11@JANELIA.HHMI.ORG
HHMI Janelia Research Campus, 19805 Helix Dr, Ashburn, VA 20147
DB Tsai DBT@NETFLIX.COM
Netflix, 970 University Ave, Los Gatos, CA 95032
Manish Amde MANISH@ORIGAMILOGIC.COM
Origami Logic, 1134 Crane Street, Menlo Park, CA 94025
Sean Owen SOWEN@CLOUDERA.COM
Cloudera UK, 33 Creechurch Lane, London EC3A 5EB United Kingdom
Doris Xin DORX0@ILLINOIS.EDU
UIUC, 201 N Goodwin Ave, Urbana, IL 61801
Reynold Xin RXIN@DATABRICKS.COM
Databricks, 160 Spear Street, 13th Floor, San Francisco, CA 94105
Michael J. Franklin FRANKLIN@CS.BERKELEY.EDU
UC Berkeley, 465 Soda Hall, Berkeley, CA 94720
Reza Zadeh REZAB@STANFORD.EDU
Stanford and Databricks, 475 Via Ortega, Stanford, CA 94305
Matei Zaharia MATEI@MIT.EDU
MIT and Databricks, 160 Spear Street, 13th Floor, San Francisco, CA 94105
Ameet Talwalkar† AMEET@CS.UCLA.EDU
UCLA and Databricks, 4732 Boelter Hall, Los Angeles, CA 90095

http://www.jmlr.org/papers/volume17/15-237/15-237.pdf

Spark MLlib: Two APIs

- **pyspark.mllib**: RDD-based API
- **pyspark.ml**: DataFrame-based API

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

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.

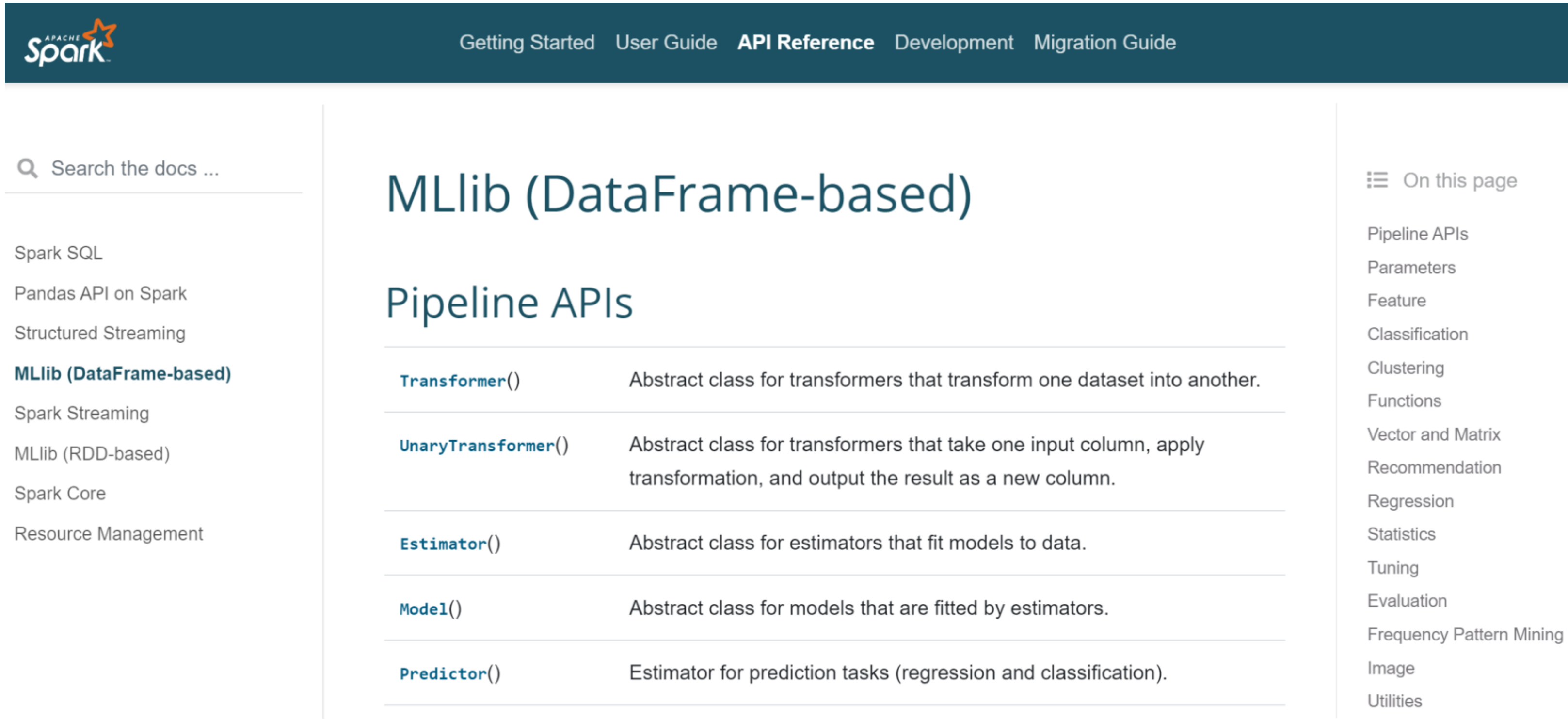
What are the implications?

- 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.

Why is MLlib switching to the DataFrame-based API?

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

Spark MLlib: pyspark.ml



Getting Started User Guide **API Reference** Development Migration Guide

Search the docs ...

Spark SQL
Pandas API on Spark
Structured Streaming
MLlib (DataFrame-based)
Spark Streaming
MLlib (RDD-based)
Spark Core
Resource Management

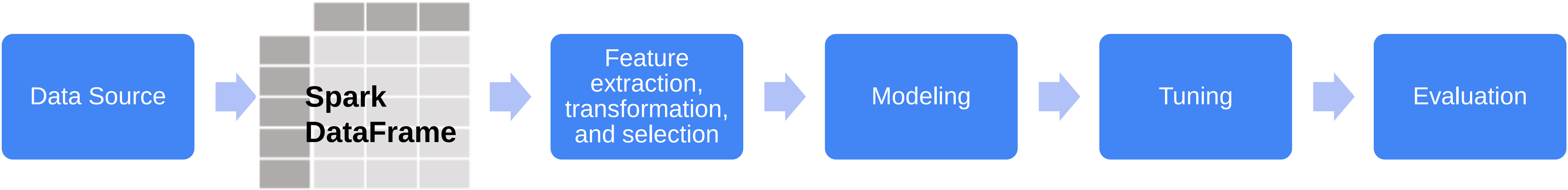
MLlib (DataFrame-based)

Pipeline APIs

`Transformer()`	Abstract class for transformers that transform one dataset into another.
`UnaryTransformer()`	Abstract class for transformers that take one input column, apply transformation, and output the result as a new column.
`Estimator()`	Abstract class for estimators that fit models to data.
`Model()`	Abstract class for models that are fitted by estimators.
`Predictor()`	Estimator for prediction tasks (regression and classification).

On this page

Pipeline APIs
Parameters
Feature
Classification
Clustering
Functions
Vector and Matrix
Recommendation
Regression
Statistics
Tuning
Evaluation
Frequency Pattern Mining
Image
Utilities



Spark MLlib: Linear Algebra

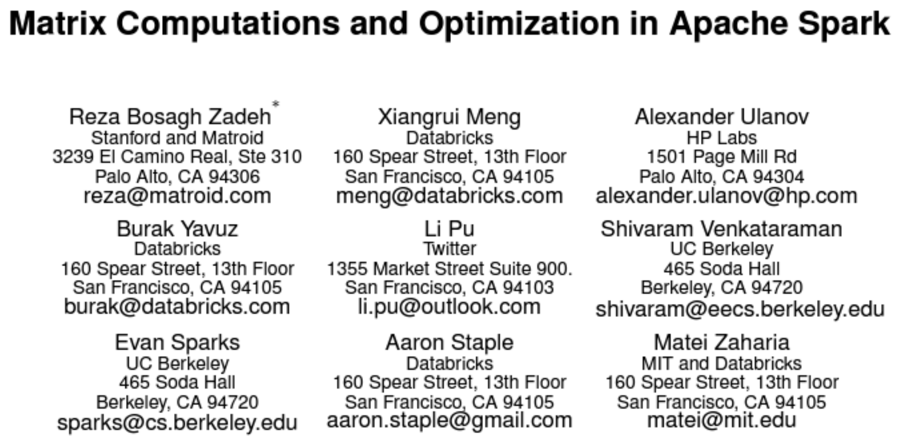
- **MLlib requires input data to be represented using its dedicated data types**

pyspark.ml.linalg

```
Vectors
DenseVector
SparseVector
Matrices
DenseMatrix
SparseMatrix
```

$$\text{dense}: \ 1.\ \ 0.\ \ 0.\ \ 0.\ \ 0.\ \ 0.\ \ 3.$$

$$\text{sparse}: \begin{cases} \text{size}: 7 \\ \text{indices}: \ 0\ \ 6 \\ \text{values}: \ 1.\ \ 3. \end{cases}$$



Matrix Computations and Optimization in Apache Spark

Reza Bosagh Zadeh*
Stanford and Matroid
3239 El Camino Real, Ste 310
Palo Alto, CA 94306
reza@matroid.com

Xiangrui Meng
Databricks
160 Spear Street, 13th Floor
San Francisco, CA 94105
meng@databricks.com

Alexander Ulanov
HP Labs
1501 Page Mill Rd
Palo Alto, CA 94304
alexander.ulanov@hp.com

Burak Yavuz
Databricks
160 Spear Street, 13th Floor
San Francisco, CA 94105
burak@databricks.com

Li Pu
Twitter
1355 Market Street Suite 900.
San Francisco, CA 94103
li.pu@outlook.com

Shivaram Venkataraman
UC Berkeley
465 Soda Hall
Berkeley, CA 94720
shivaram@eecs.berkeley.edu

Evan Sparks
UC Berkeley
465 Soda Hall
Berkeley, CA 94720
sparks@cs.berkeley.edu

Aaron Staple
Databricks
160 Spear Street, 13th Floor
San Francisco, CA 94105
aaron.staple@gmail.com

Matei Zaharia
MIT and Databricks
160 Spear Street, 13th Floor
San Francisco, CA 94105
matei@mit.edu

https://stanford.edu/~rezab/papers/linalg.pdf

Spark MLlib: Statistics

pyspark.ml.stat

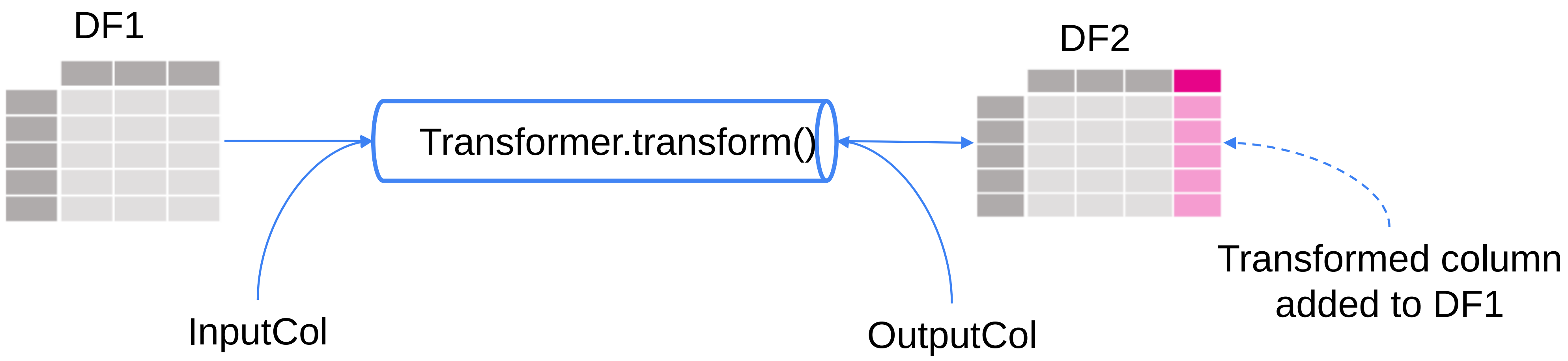
```
Summarizer
Correlation
ChiSquareTest
```

- **Summarizer: vector column summary statistics (max, min, mean, sum, variance, std, numNonzeros, count, normL2, normL1)**
- **Correlation: Pearson or Spearman**
- **Hypothesis testing: Pearson's Chi-squared tests for independence**

Spark MLlib: Transformers

pyspark.ml.Transformer

- **Transformer: an algorithm that transforms one DataFrame into another DataFrame.**

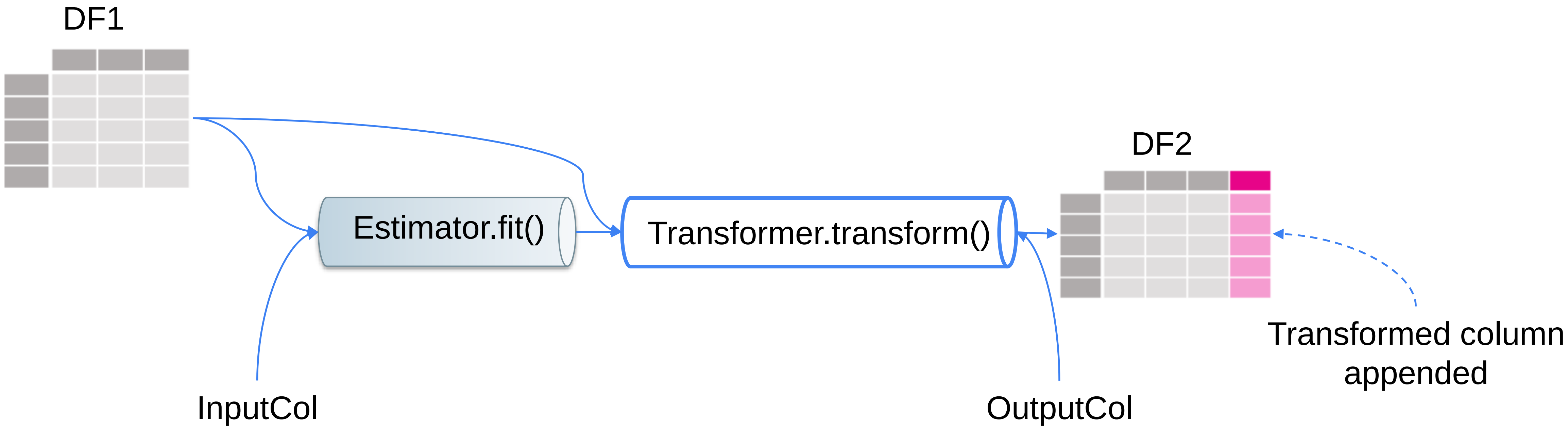


- **Transformers either prepare data for model training or generate predictions using a trained ML model.**
- **An ML model is a Transformer that transforms a DataFrame (e.g., test data) into another DataFrame with predictions.**

Spark MLlib: Estimators

pyspark.ml.Estimator

- **Estimator: an algorithm that fits or train on a DataFrame to produce a Transformer.**



- **A learning algorithm is an Estimator that trains on a DataFrame and produces a model, which is a Transformer.**

pyspark.ml.feature

Spark MLlib: extracting, transforming, and selecting features

Continuous Feature

```
Binarizer
Bucketizer
Normalizer
StandardScaler
MinMaxScaler
MaxAbsScaler
QuantileDiscretizer
```

Categorical Feature

```
StringIndexer
IndexToString
VectorIndexer
OneHotEncoder
```

Text Data

```
Tokenizer
RegexTokenizer
HashingTF
StopWordsRemover
NGram
CountVectorizer
IDF
Word2Vec
```
FeatureHasher

Others

```
Imputer
PCA (Principal
Component Analysis)
DCT (Discrete
Cosine Transform)
ElementwiseProduct
Interaction
PolynomialExpansion
VectorAssembly
SQLTransformer
```

Feature Selectors

VectorSlicer
ChiSqSelector
```
VarianceThresholdSelector
FeatureHasher
UnivariateFeatureSelector
```

Locality Sensitive Hashing (LSH)

BucketedRandomProjectionLSH
MinHashLSH

https://spark.apache.org/docs/latest/ml-features.html

Spark MLlib: Distributed ML Algorithms

- **Each algorithm is implemented as an Estimator that produces a Model**

pyspark.ml.regression

LinearRegression
GeneralizedLinearRegression
DecisionTreeRegressor
RandomForestRegressor
GBTRegressor (Gradient-Boosted Tree)
AFTSurvivalRegression
IsotonicRegression
FMRegressor (Factorization Machines)

pyspark.ml.classification

LogisticRegression
LinearSVC
DecisionTreeClassifier
RandomForestClassifier
GBTClassifier
NaiveBayes
MultilayerPerceptronClassifier
OneVsRest
FMClassifier (Factorization Machines)

pyspark.ml.clustering

KMeans
BisectingKMeans
LDA (Latent Dirichlet Allocation)
GaussianMixture
PowerIterationClustering

pyspark.ml.recommendation

ALS (Alternating Least Squares)

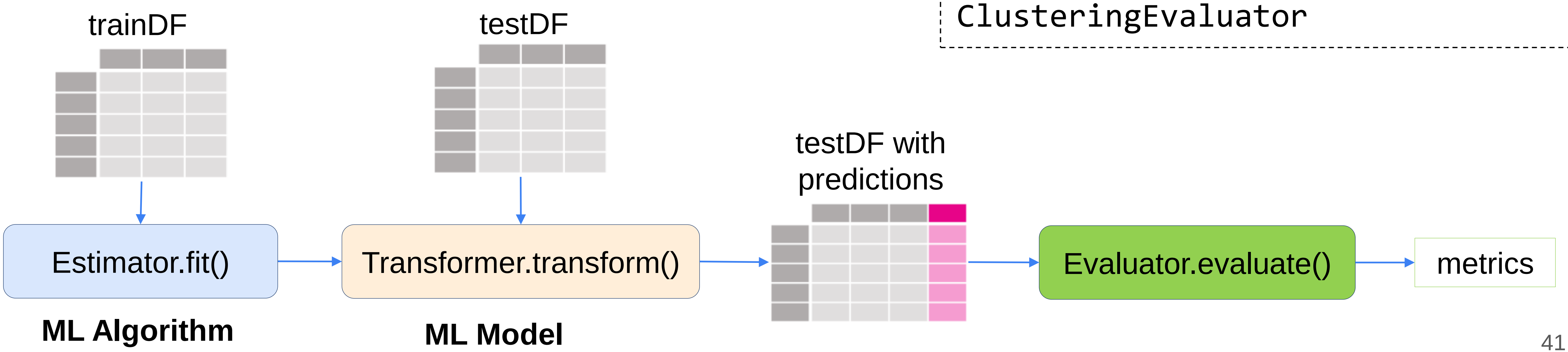
pyspark.ml.fpm
(Frequent Pattern Mining)

FPGrowth
PrefixSpan

Spark MLlib: Model Evaluation

pyspark.ml.evaluation

```
Evaluator
RegressionEvaluator
BinaryClassificationEvaluator
MulticlassClassificationEvaluator
MultilabelClassificationEvaluator
RankingEvaluator
ClusteringEvaluator
```

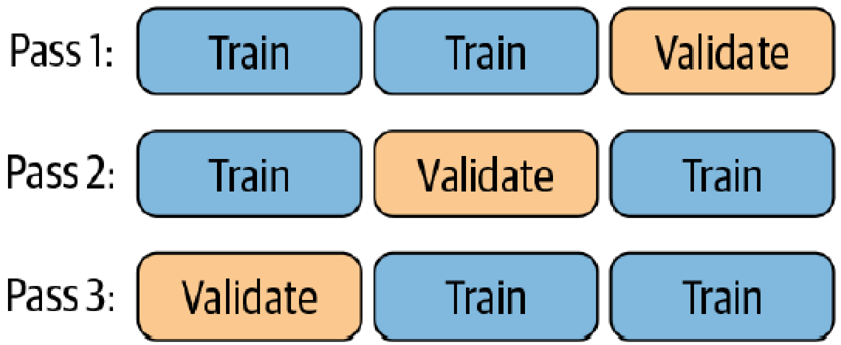


Spark MLlib: Hyperparameter Tuning (model selection)

- **Hyperparameter: a value you prior to training (e.g., k in Kmeans, numTrees in a Random Forest)**
- **Parameter Gird: a grid or table of parameters with a discrete number of values for each one**
- **Estimators and Transformers use a uniform API for specifying parameters.** (pyspark.ml.param)
- **Train-Validation Split (one split)**
- **Cross-Validation (multiple splits)**

pyspark.ml.tuning

```
ParamGridBuilder
CrossValidator (k-fold)
CrossValidatorModel
TrainValidationSplit
TrainValidationSplitModel
```



k-fold cross-validation

https://spark.apache.org/docs/latest/ml-tuning.html

Spark MLlib: Pipelines

- **ML Pipelines API: A high-level API built on top of DataFrames to organize a pipeline**
- **A Pipeline is composed of a series of stages (Transformers and Estimators) to be applied together to a DataFrame**
- **Pipeline is an Estimator**
- **PipelineModel is a fitted Pipeline (i.e., a Transformer)**

Pipelines API

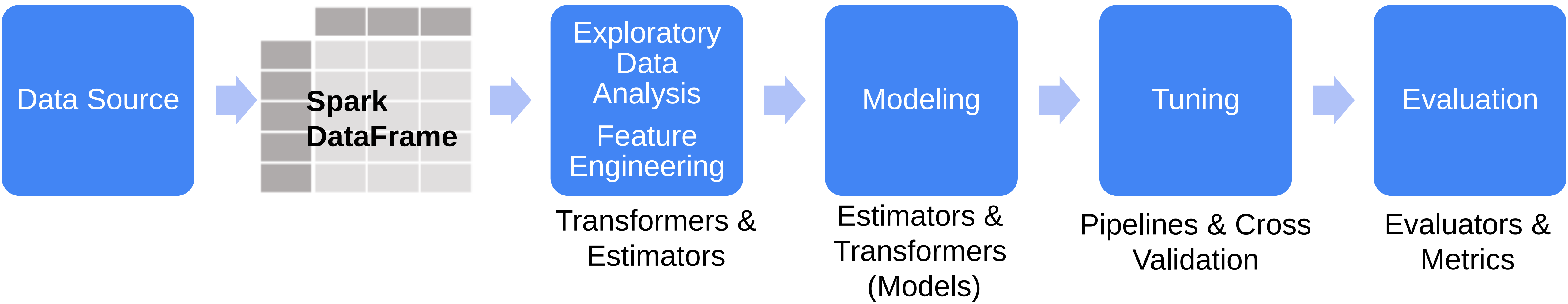
```
pyspark.ml.Pipeline
pyspark.ml.PipelineModel
pyspark.ml.Transformer
pyspark.ml.Estimator
pyspark.ml.Evaluator
pyspark.ml.Model
pyspark.ml.Predictor
pyspark.ml.PredictorModel
```

Pipelines and PipelineModels ensure that training and test data go through identical feature processing steps.

Spark MLlib: Other Utilities

- **ML Persistence: MLlib enables saving and loading all Models and Pipelines**
  - **save()**
  - **load()**

ML Workflow with Spark



Distributed ML Algorithms

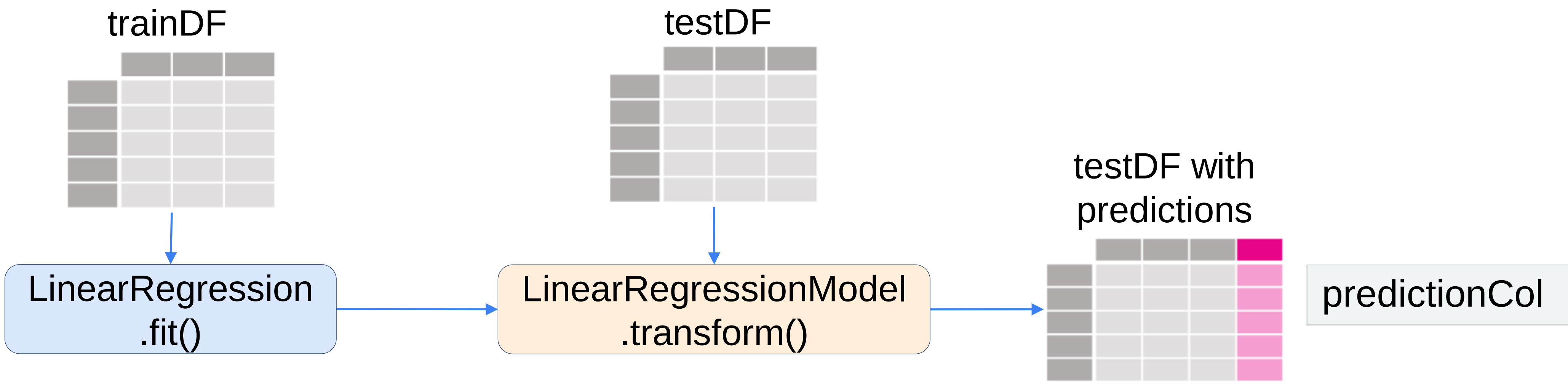
`pyspark.ml.regression.LinearRegression`

Linear Regression

- **Predict a numerical response from a vector of features**

- featuresCol
- labelCol

- maxIter

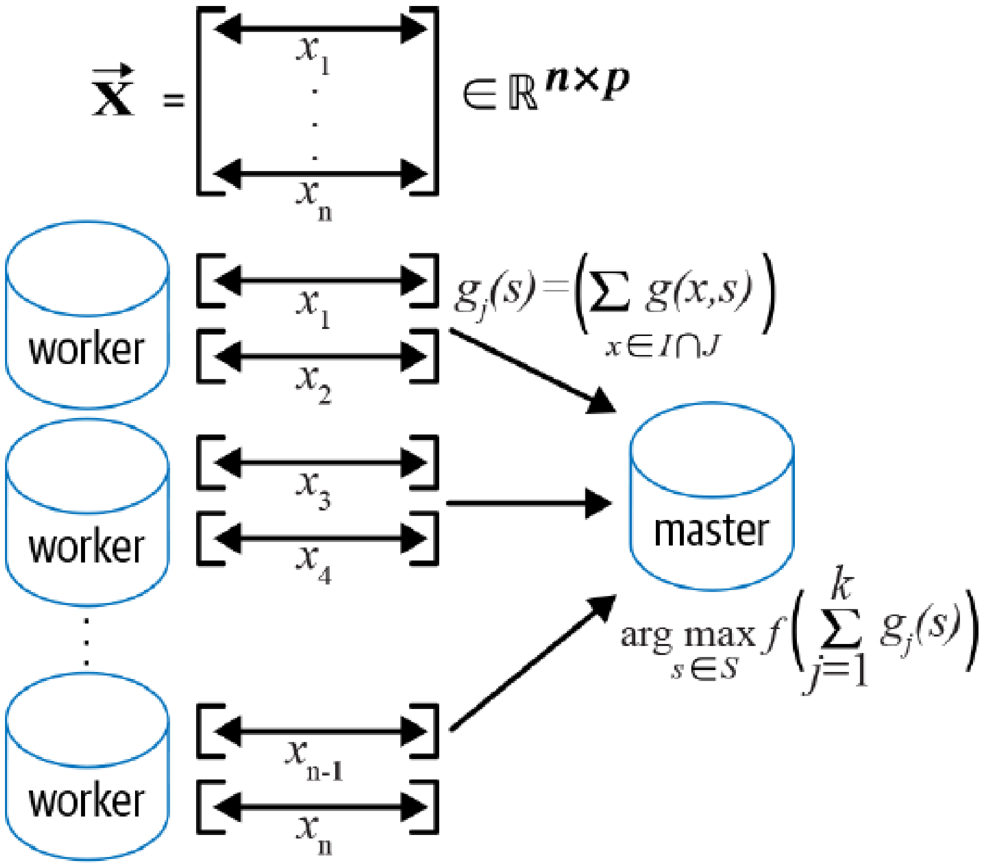


Decision Trees

`pyspark.ml.regression.DecisionTreeClassifier`

`pyspark.ml.regression.DecisionTreeRegressor`

- **Decision tree: a series of if-then-else rules learned from data for classification or regression tasks.**



PLANET implementation of distributed decision trees

PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce

Biswanath Panda, Joshua S. Herbach, Sugato Basu, Roberto J. Bayardo
Google, Inc.
[bpanda, jsherbach, sugato]@google.com, bayardo@alum.mit.edu

ABSTRACT

Classification and regression tree learning on massive datasets is a common data mining task at Google, yet many state of the art tree learning algorithms require training data to reside in memory on a single machine. While more scalable implementations of tree learning have been proposed, they typically require specialized parallel computing architectures. In contrast, the majority of Google's computing infrastructure is based on commodity hardware.

In this paper, we describe PLANET: a scalable distributed framework for learning tree models over large datasets. PLANET defines tree learning as a series of distributed computations, and implements each one using the *MapReduce* model of distributed computation. We show how this framework supports scalable construction of classification and regression trees, as well as ensembles of such models. We discuss the benefits and challenges of using a MapReduce compute cluster for tree learning, and demonstrate the scalability of this approach by applying it to a real world learning task from the domain of computational advertising.

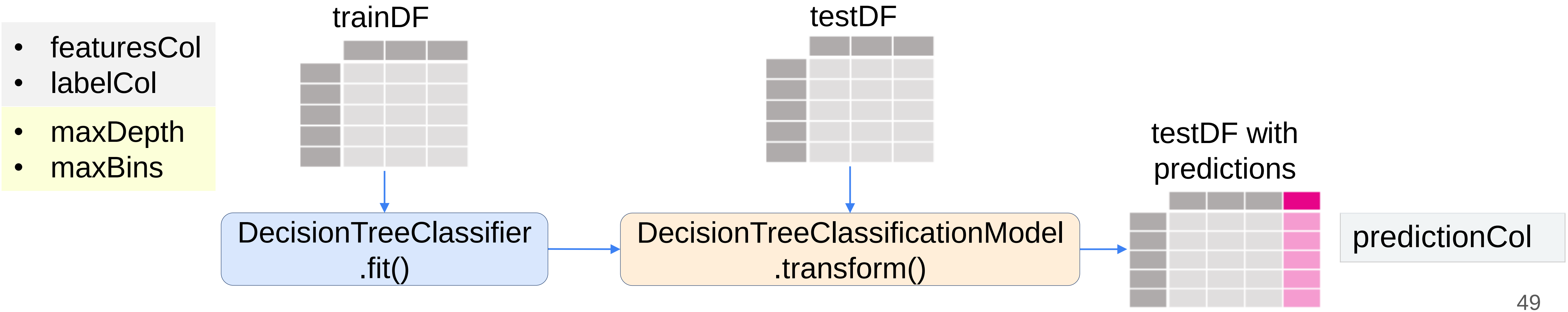
plexities such as data partitioning, scheduling tasks across many machines, handling machine failures, and performing inter-machine communication. These properties have motivated many technology companies to run MapReduce frameworks on their compute clusters for data analysis and other data management tasks. MapReduce has become in some sense an industry standard. For example, there are open source implementations such as Hadoop that can be run either in-house or on cloud computing services such as Amazon EC2.[1] Startups like Cloudera[2] offer software and services to simplify Hadoop deployment, and companies including Google, IBM and Yahoo! have granted several universities access to Hadoop clusters to further cluster computing research.[3]

Despite the growing popularity of MapReduce [12], its application to certain standard data mining and machine learning tasks remains poorly understood. In this paper we focus on one such task: tree learning. We believe that a tree learner capable of exploiting a MapReduce cluster can effectively address many scalability issues that arise in building tree models on massive datasets. Our choice of focusing `pyspark.ml.regression.DecisionTreeClassifier`

`pyspark.ml.regression.DecisionTreeRegressor`

Decision Trees

- **MLlib supports decision trees for binary and multiclass classification and for regression, using both continuous and categorical features.**

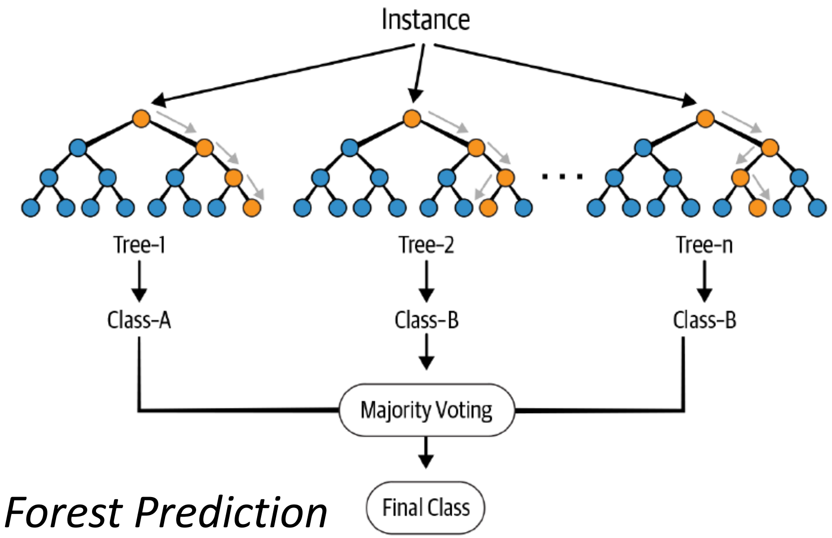


Random Forests

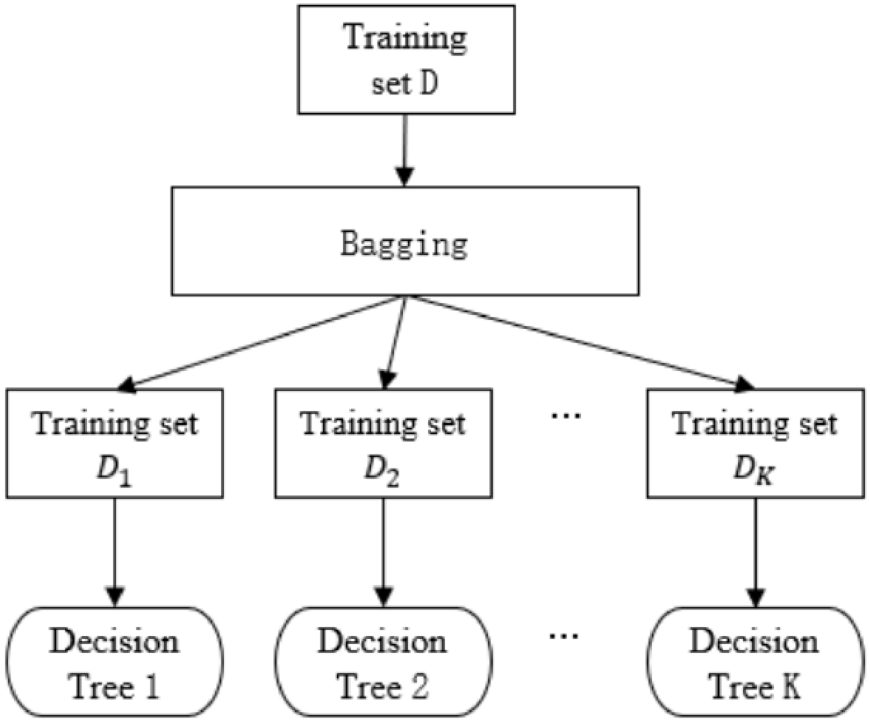
`pyspark.ml.classification.RandomForestClassifier`

`pyspark.ml.classification.RandomForestRegressor`

- **Random forest: ensemble of decision trees**
- **Bootstrapping samples by rows**
- **Random feature selection by columns**
- **In addition to the parallelization for each single tree, multiple trees can be trained in parallel**



Random Forest Prediction



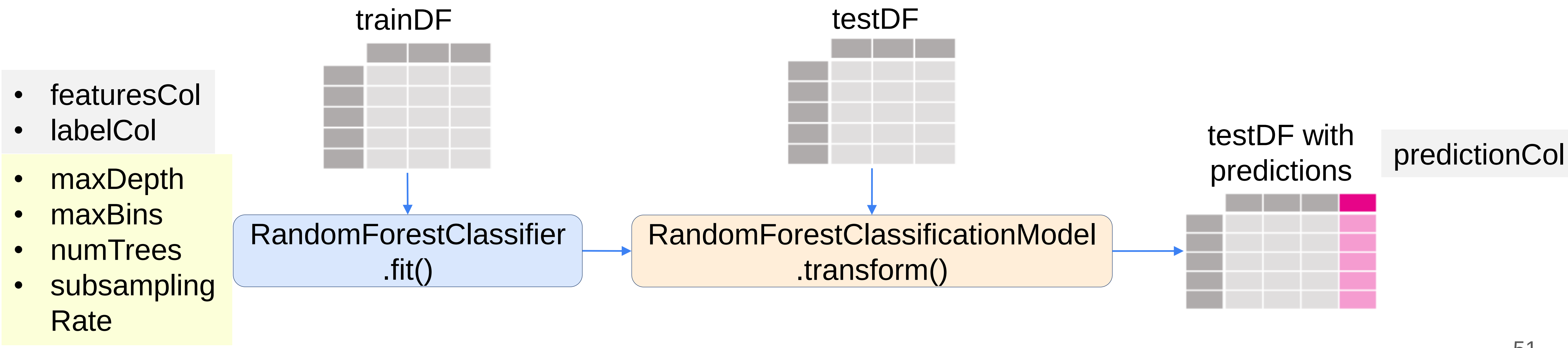
Random Forest Training

Random Forests

`pyspark.ml.classification.RandomForestClassifier`

`pyspark.ml.classification.RandomForestRegressor`

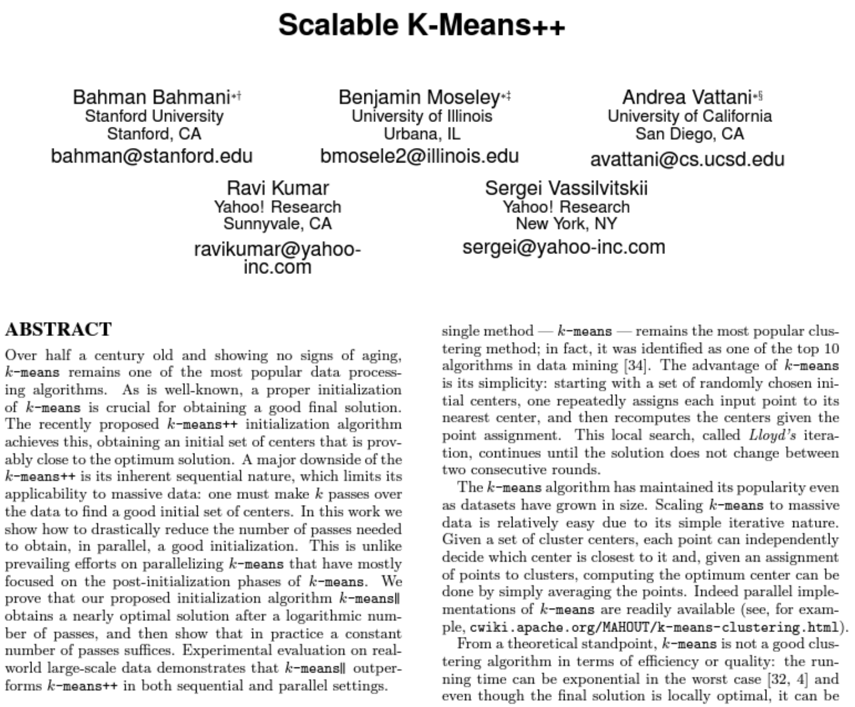
- **MLlib supports Random Forests for binary and multiclass classification and for regression, using both continuous and categorical features.**



`pyspark.ml.clustering.KMeans`

Kmeans

- **Kmeans: clustering algorithm that group the data points into a predefined number of clusters based on a similarity distance.**



Scalable K-Means++

Bahman Bahmani
Stanford University
Stanford, CA
bahman@stanford.edu

Benjamin Moseley
University of Illinois
Urbana, IL
bmosele2@illinois.edu

Andrea Vattani
University of California
San Diego, CA
avattani@cs.ucsd.edu

Ravi Kumar
Yahoo! Research
Sunnyvale, CA
ravikumar@yahoo-inc.com

Sergei Vassilvitskii
Yahoo! Research
New York, NY
sergei@yahoo-inc.com

ABSTRACT

Over half a century old and showing no signs of aging, k-`means` remains one of the most popular data processing algorithms. As is well-known, a proper initialization of k-`means` is crucial for obtaining a good final solution. The recently proposed k-`means++` initialization algorithm achieves this, obtaining an initial set of centers that is provably close to the optimum solution. A major downside of the k-`means++` is its inherent sequential nature, which limits its applicability to massive data: one must make k passes over the data to find a good initial set of centers. In this work we show how to drastically reduce the number of passes needed to obtain, in parallel, a good initialization. This is unlike prevailing efforts on parallelizing k-`means` that have mostly focused on the post-initialization phases of k-`means`. We prove that our proposed initialization algorithm k-`means||` obtains a nearly optimal solution after a logarithmic number of passes, and then show that in practice a constant number of passes suffices. Experimental evaluation on real-world large-scale data demonstrates that k-`means||` outperforms k-`means++` in both sequential and parallel settings.

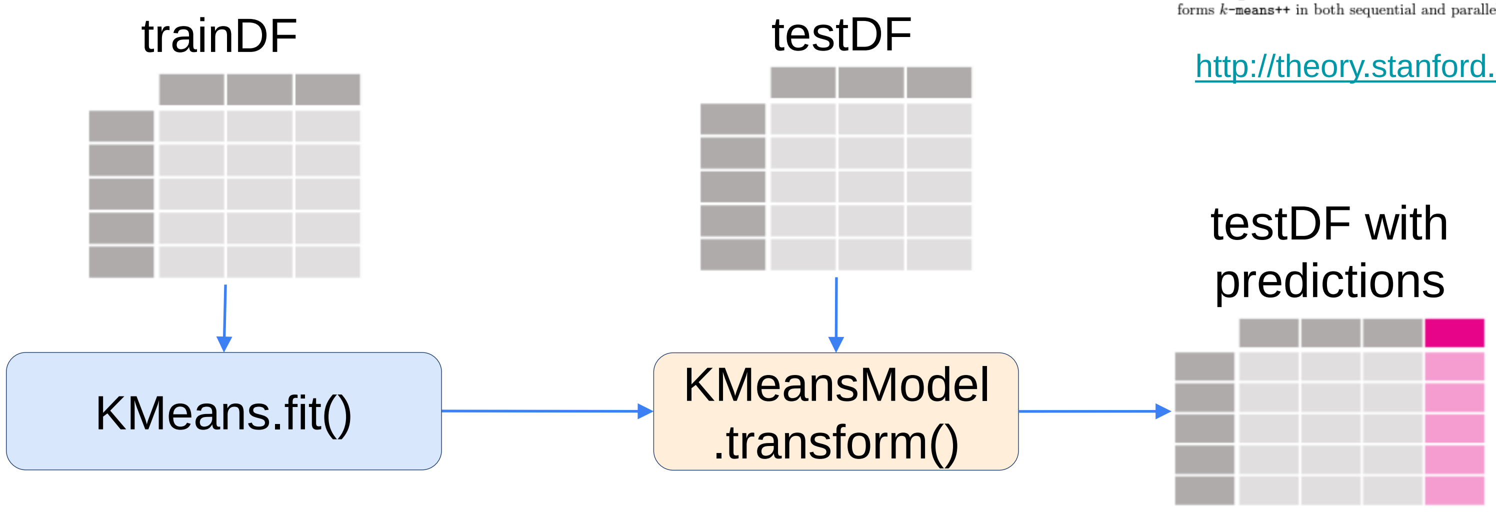
single method — k-`means` — remains the most popular clustering method; in fact, it was identified as one of the top 10 algorithms in data mining [34]. The advantage of k-`means` is its simplicity: starting with a set of randomly chosen initial centers, one repeatedly assigns each input point to its nearest center, and then recomputes the centers given the point assignment. This local search, called *Lloyd's* iteration, continues until the solution does not change between two consecutive rounds.

The k-`means` algorithm has maintained its popularity even as datasets have grown in size. Scaling k-`means` to massive data is relatively easy due to its simple iterative nature. Given a set of cluster centers, each point can independently decide which center is closest to it and, given an assignment of points to clusters, computing the optimum center can be done by simply averaging the points. Indeed parallel implementations of k-`means` are readily available (see, for example, `cwiki.apache.org/MAHOUT/k-means-clustering.html`).

From a theoretical standpoint, k-`means` is not a good clustering algorithm in terms of efficiency or quality: the running time can be exponential in the worst case [32, 4] and even though the final solution is locally optimal, it can be

http://theory.stanford.edu/~sergei/papers/vldb12-kmpar.pdf

- featuresCol
- k
- initMode
- initSteps



predictionCol

Summary

- **Apache Spark is a unified computing engine designed for large-scale distributed data processing, on single-node machines, on-premises clusters, or in the cloud.**
- **Spark SQL is a foundational component of Apache Spark**
- **Spark DataFrames provides structured and high-level API**
- **Spark Data Source connectors enables reading /writing data from/to different sources**

Summary

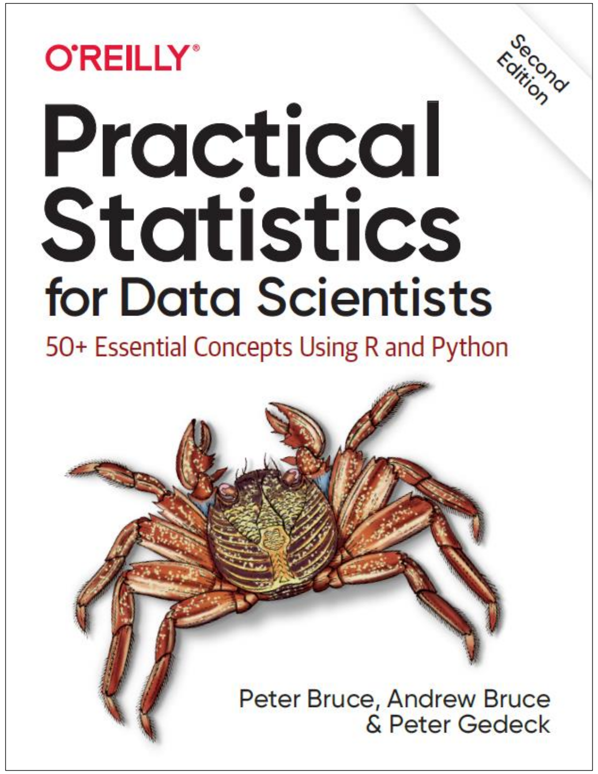
- **Spark MLlib: DataFrame-Based API**
- **Featurization: feature extraction, transformation, and selection**
- **Distributed ML Algorithms: classification, regression, clustering, …**
- **Pipelines API: Transformers, Estimators, and Pipelines**
- **Hyperparameter Tuning: cross-validation**

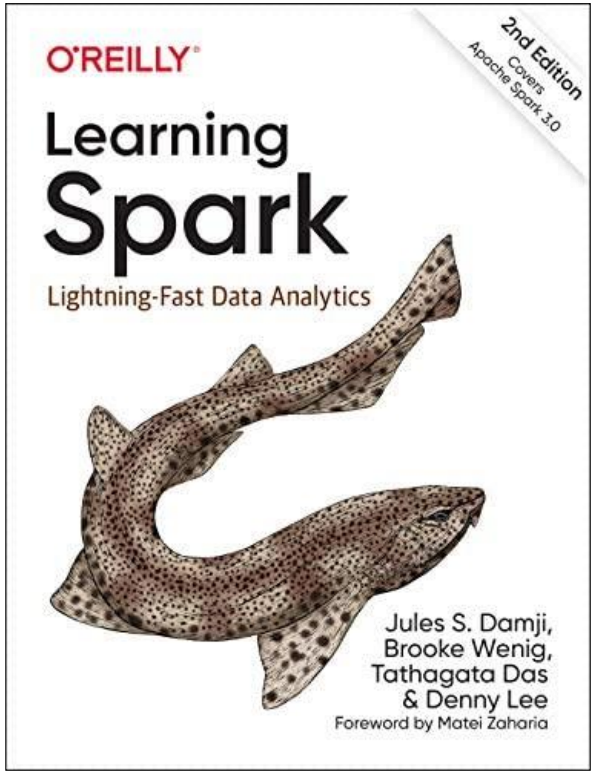
Thank You

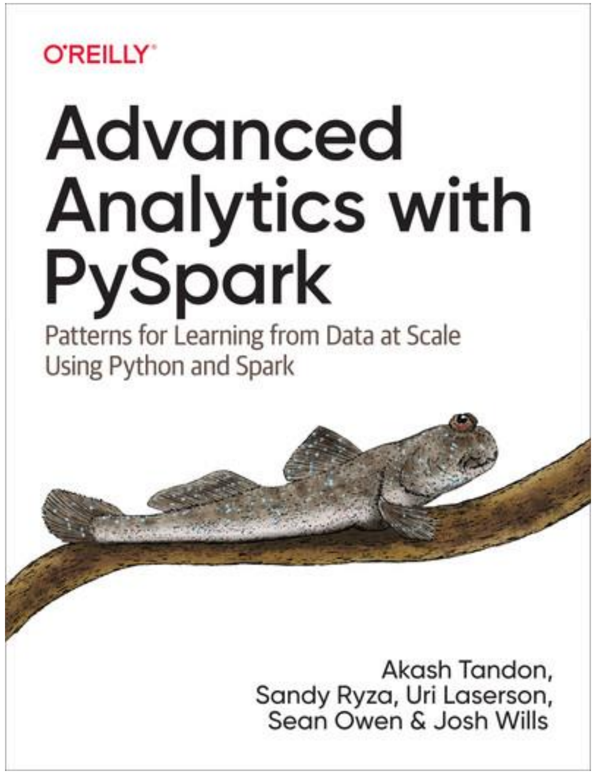
Salman Salloum
www.linkedin.com/in/ssalloum/

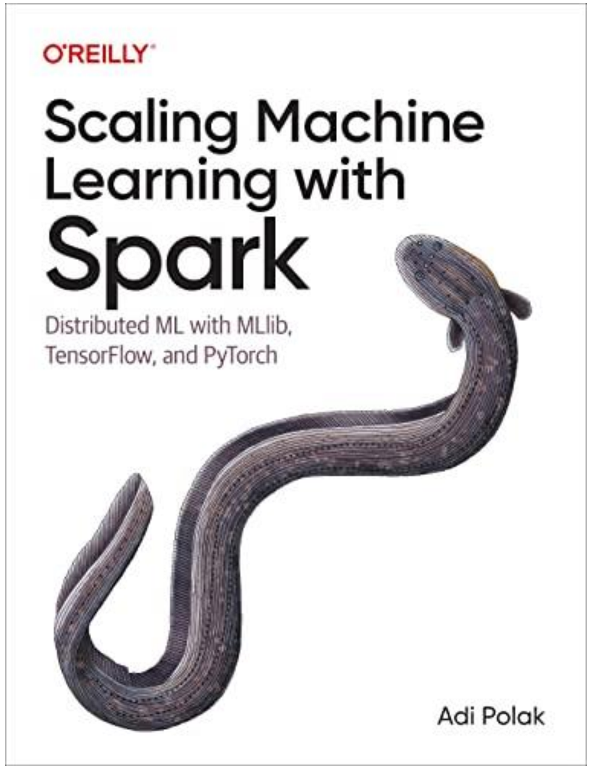
https://github.com/ssalloum/SDSC-Spark4

Books


O'REILLY
Second Edition
Practical Statistics for Data Scientists
50+ Essential Concepts Using R and Python
Peter Bruce, Andrew Bruce & Peter Gedeck


O'REILLY
2nd Edition
Covers Apache Spark 3.0
Learning Spark
Lightning-Fast Data Analytics
Jules S. Damji, Brooke Wenig, Tathagata Das & Denny Lee
Foreword by Matei Zaharia


O'REILLY
Advanced Analytics with PySpark
Patterns for Learning from Data at Scale Using Python and Spark
Akash Tandon, Sandy Ryza, Uri Laserson, Sean Owen & Josh Wills


O'REILLY
Scaling Machine Learning with Spark
Distributed ML with MLlib, TensorFlow, and PyTorch
Adi Polak