# IEMS5730 & IERG4330

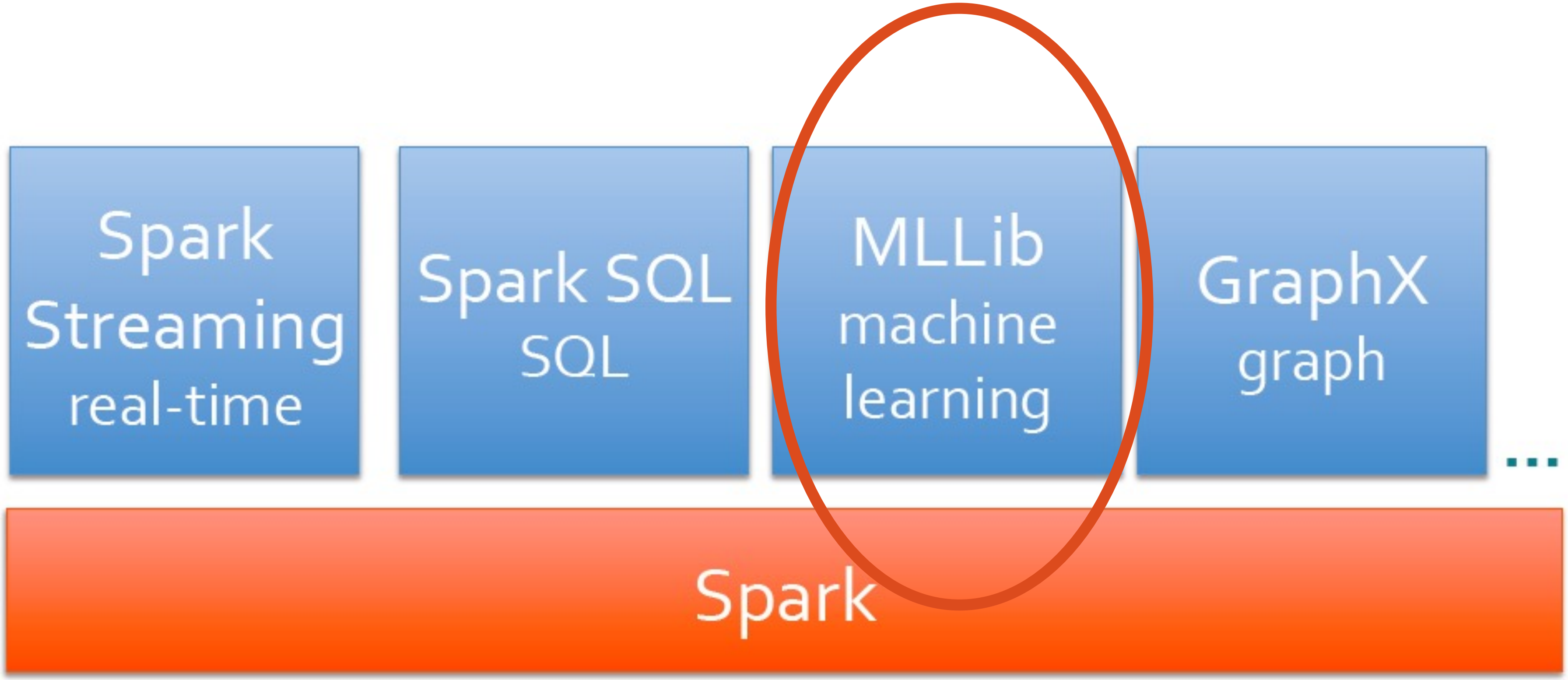
# Machine Learning Support and Beyond

Prof. Wing C. Lau
Department of Information Engineering
wclau@ie.cuhk.edu.hk

# Acknowledgements

- These slides are adapted from the following sources:
  - Matei Zaharia, “Spark 2.0,” Spark Summit East Keynote, Feb 2016.
  - Reynold Xin, “The Future of Real-Time in Spark,” Spark Summit East Keynote, Feb 2016.
  - Michael Armburst, “Structuring Spark: SQL, DataFrames, DataSets, and Streaming,” Spark Summit East Keynote, Feb 2016.
  - Ankur Dave, “GraphFrames: Graph Queries in Spark SQL,” Spark Summit East, Feb 2016.
  - Michael Armburst, “Spark DataFrames: Simple and Fast Analytics on Structured Data,” Spark Summit Amsterdam, Oct 2015.
  - Michael Armburst et al, “Spark SQL: Relational Data Processing in Spark,” SIGMOD 2015.
  - Michael Armburst, “Spark SQL Deep Dive,” Melbourne Spark Meetup, June 2015.
  - Reynold Xin, “Spark,” Stanford CS347 Guest Lecture, May 2015.
  - Joseph K. Bradley, “Apache Spark MLlib’s past trajectory and new directions,” Spark Summit Jun 2017.
  - Joseph K. Bradley, “Distributed ML in Apache Spark,” NYC Spark MeetUp, June 2016.
  - Ankur Dave, “GraphFrames: Graph Queries in Apache Spark SQL,” Spark Summit, June 2016.
  - Joseph K. Bradley, “GraphFrames: DataFrame-based graphs for Apache Spark,” NYC Spark MeetUp, April 2016.
  - Joseph K. Bradley, “Practical Machine Learning Pipelines with MLlib,” Spark Summit East, March 2015.
  - Joseph K. Bradley, “Spark DataFrames and ML Pipelines,” MLconf Seattle, May 2015.
  - Ameet Talwalkar, “MLlib: Spark’s Machine Learning Library,” AMPCamps 5, Nov. 2014.
  - Shivaram Venkataraman, Zongheng Yang, “SparkR: Enabling Interactive Data Science at Scale,” AMPCamps 5, Nov. 2014.
  - Tathagata Das, “Spark Streaming: Large-scale near-real-time stream processing,” O’Reilly Strata Conference, 2013.
  - Joseph Gonzalez et al, “GraphX: Graph Analytics on Spark,” AMPCAMP 3, 2013.
  - Jules Damji, “Jumpstart on Apache Spark 2.X with Databricks,” Spark Sat. Meetup Workshop, Jul 2017.
  - Sameer Agarwal, “What’s new in Apache Spark 2.3,” Spark+AI Summit, June 2018.
  - Reynold Xin, Spark+AI Summit Europe, 2018.
  - Hyukjin Kwon of Hortonworks, “What’s New in Spark 2.3 and Spark 2.4,” Oct 2018.
  - Matel Zaharia, “MLflow: Accelerating the End-to-End ML Lifecycle,” Nov. 2018.
  - Jules Damji, “MLflow: Platform for Complete Machine Learning Lifecycle,” PyData, Jan 2019.
- All copyrights belong to the original authors of the materials.

# About Spark for MLlib



# About Apache Spark MLlib

Started at Berkeley AMPLab
(Apache Spark 0.8)

By Apache Spark 2.1 (circa Apr 2017)

•Contributions from 75+ orgs, ~250 individuals

•Development driven by Databricks: roadmap + 50% of PRs

•Growing coverage of *distributed* algorithms

# MLlib Goals

General Machine Learning library for big data

- Scalable & robust
- Coverage of common algorithms
- Leverages Apache Spark

Tools for practical workflows

Integration with existing data science tools

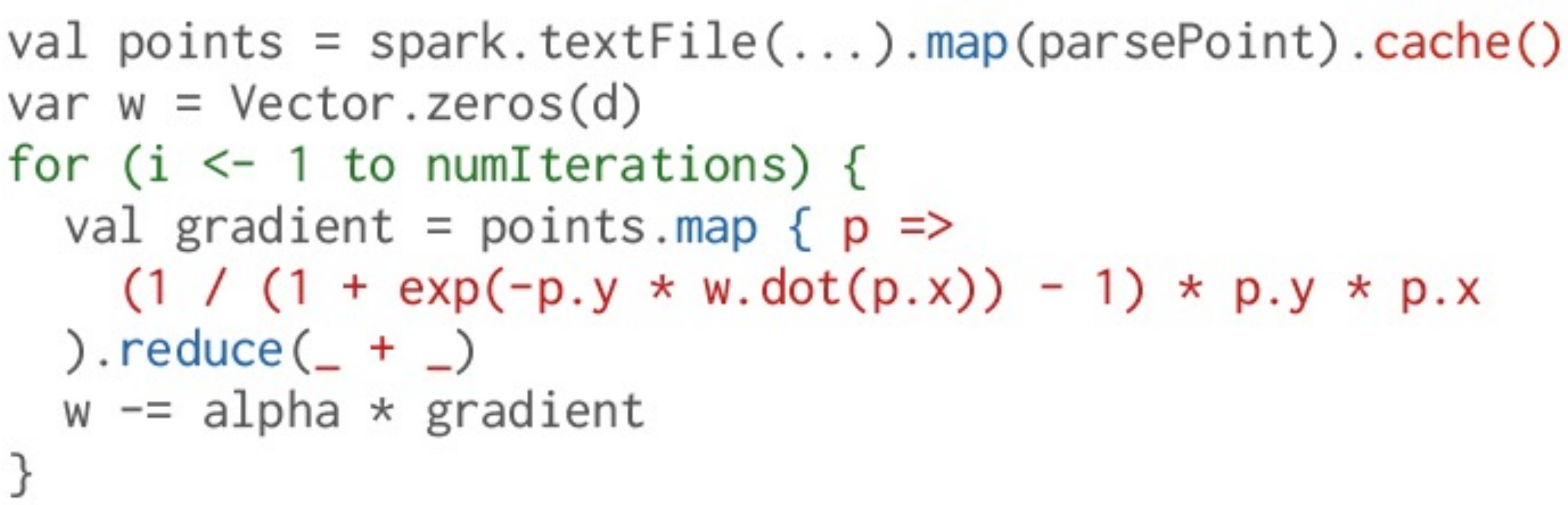
# Apache Spark MLlib

- spark.mllib
- Pre MLlib < Spark 1.4
- Spark mllib was a lower level library that used Spark RDDs
- Use LabeledPoint, Vectors and Tuples
- Maintenance Mode only after Spark 2.X

```
// Load and parse the data

val data = sc.textFile("data/mllib/ridge-data/lpsa.data")

val parsedData = data.map { line =>

  val parts = line.split(',')

  LabeledPoint(parts(0).toDouble, Vectors.dense(parts(1).split(' ').map(_.toDouble)))

}.cache()


// Building the model

val numIterations = 100

val stepSize = 0.00000001

val model = LinearRegressionWithSGD.train(parsedData, numIterations, stepSize)


// Evaluate model on training examples and compute training error

val valuesAndPreds = parsedData.map { point =>

  val prediction = model.predict(point.features)

  (point.label, prediction)

}
```

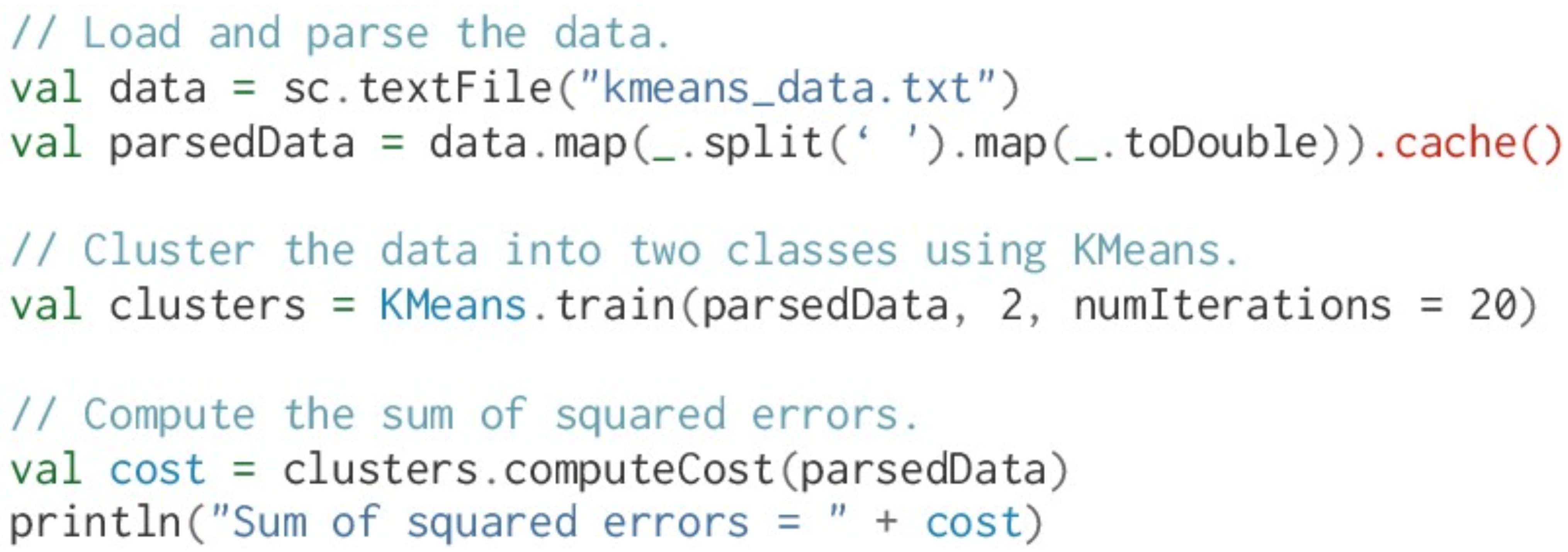
# Gradient Descent

$$w \leftarrow w - \alpha \cdot \sum_{i=1}^{n} g(w; x_i, y_i)$$



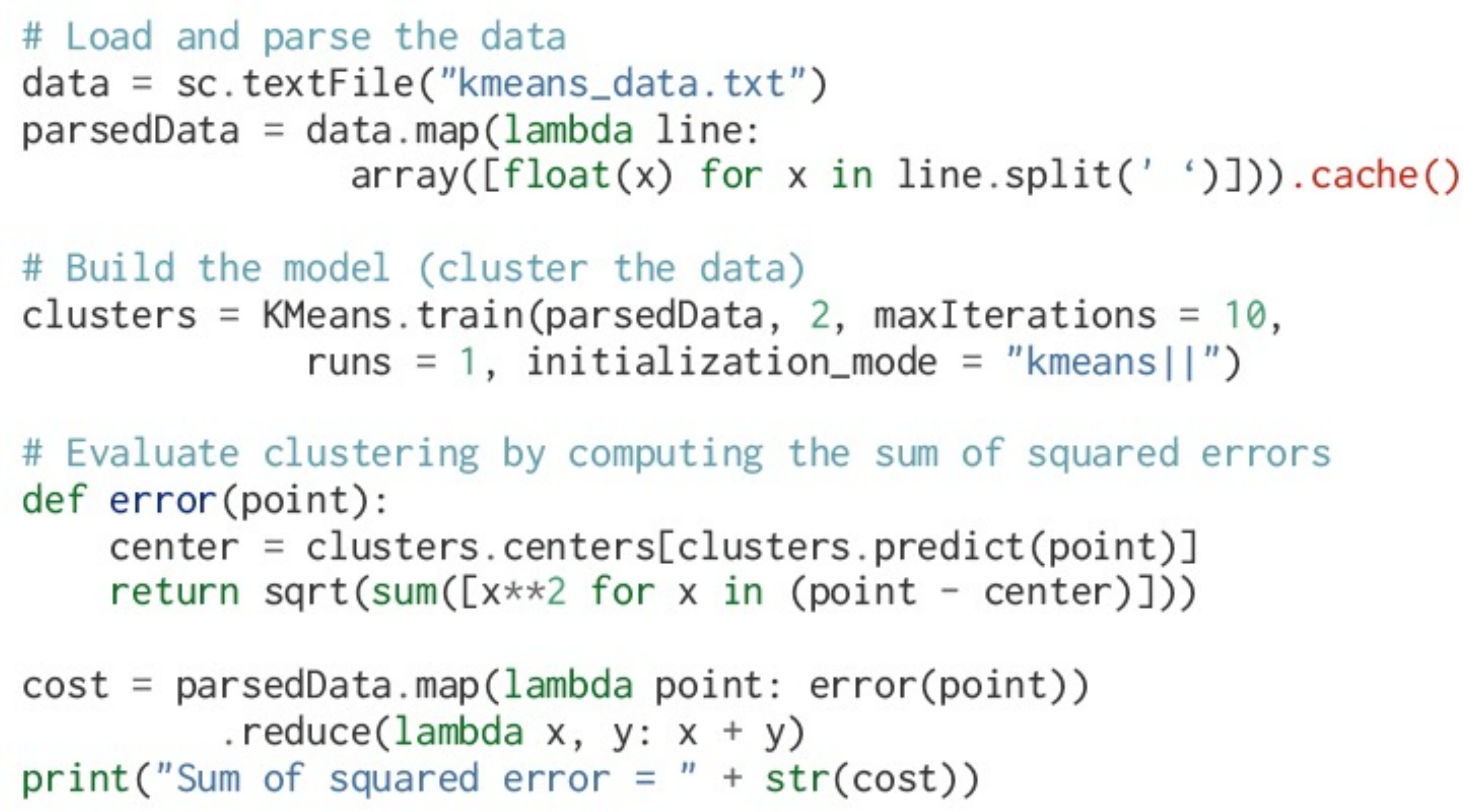
```
val points = spark.textFile(...).map(parsePoint).cache()
var w = Vector.zeros(d)
for (i <- 1 to numIterations) {
  val gradient = points.map { p =>
    (1 / (1 + exp(-p.y * w.dot(p.x)) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= alpha * gradient
}
```

# k-means (scala)



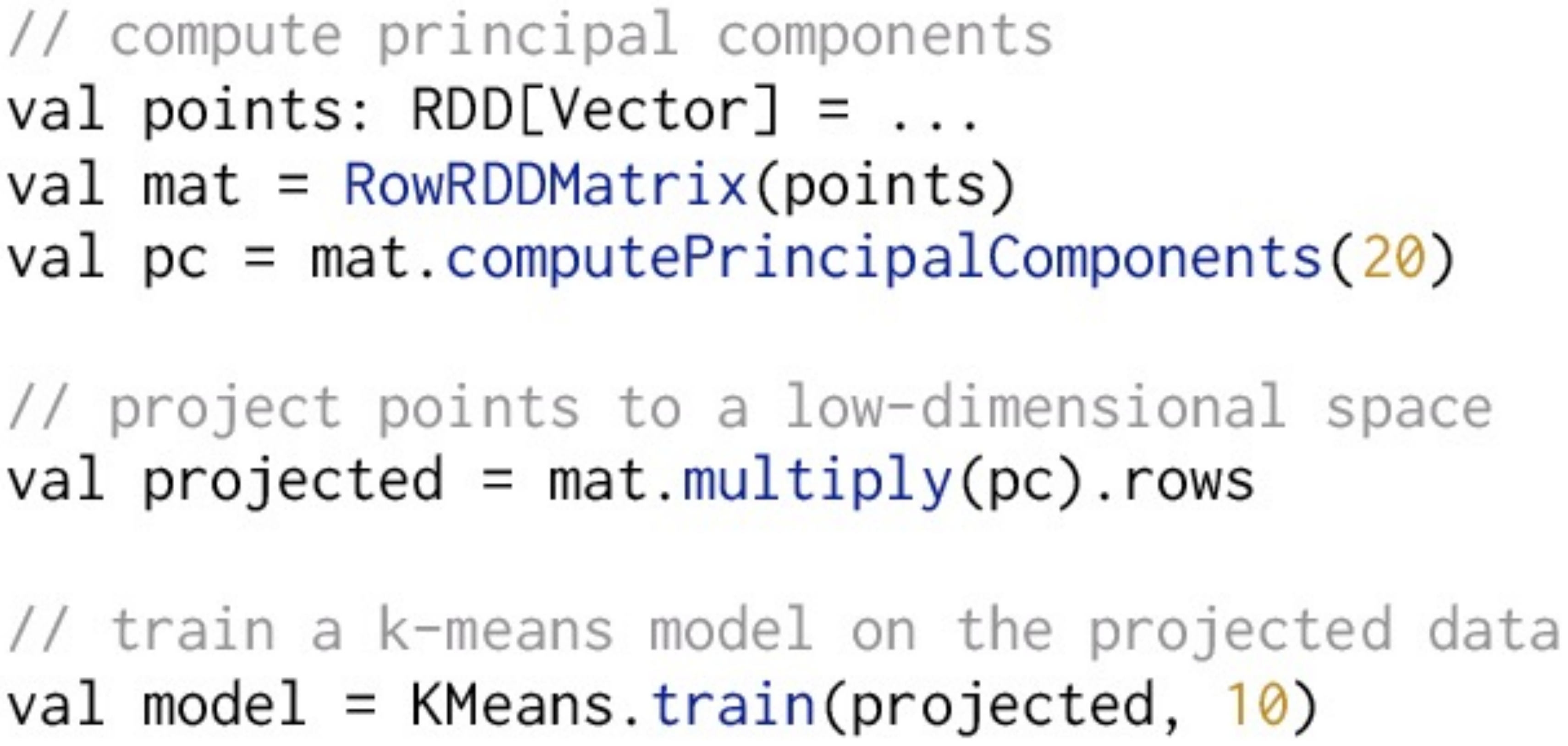
```
// Load and parse the data.
val data = sc.textFile("kmeans_data.txt")
val parsedData = data.map(_.split(‘ ’).map(_.toDouble)).cache()

// Cluster the data into two classes using KMeans.
val clusters = KMeans.train(parsedData, 2, numIterations = 20)

// Compute the sum of squared errors.
val cost = clusters.computeCost(parsedData)
println("Sum of squared errors = " + cost)
```

# k-means (python)



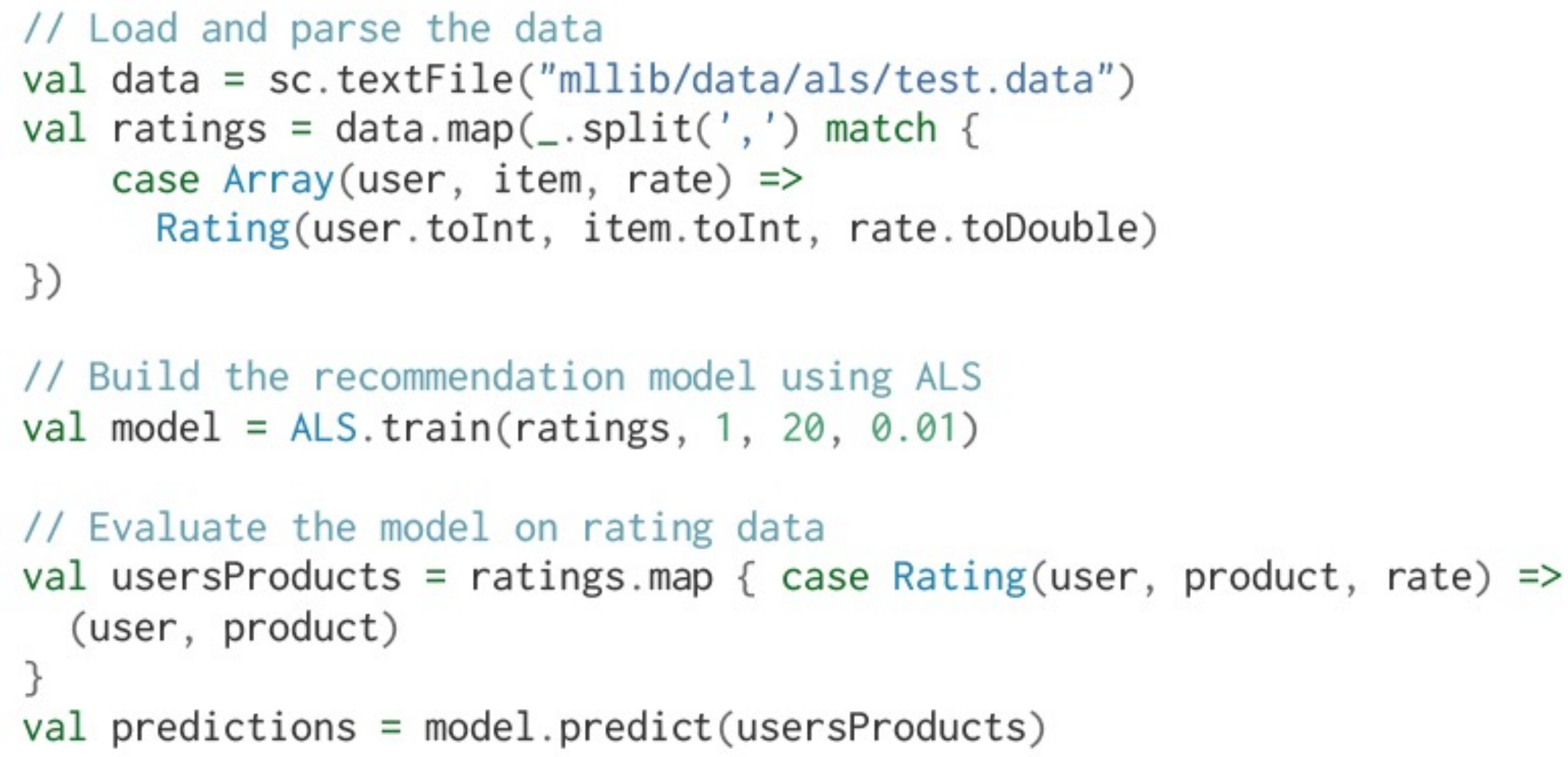
```
# Load and parse the data
data = sc.textFile("kmeans_data.txt")
parsedData = data.map(lambda line:
                    array([float(x) for x in line.split(' ‘)])).cache()

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 2, maxIterations = 10,
                 runs = 1, initialization_mode = "kmeans||")

# Evaluate clustering by computing the sum of squared errors
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))

cost = parsedData.map(lambda point: error(point))
          .reduce(lambda x, y: x + y)
print("Sum of squared error = " + str(cost))
```

# Dimension Reduction + k-means



```
// compute principal components
val points: RDD[Vector] = ...
val mat = RowRDDMatrix(points)
val pc = mat.computePrincipalComponents(20)

// project points to a low-dimensional space
val projected = mat.multiply(pc).rows

// train a k-means model on the projected data
val model = KMeans.train(projected, 10)
```

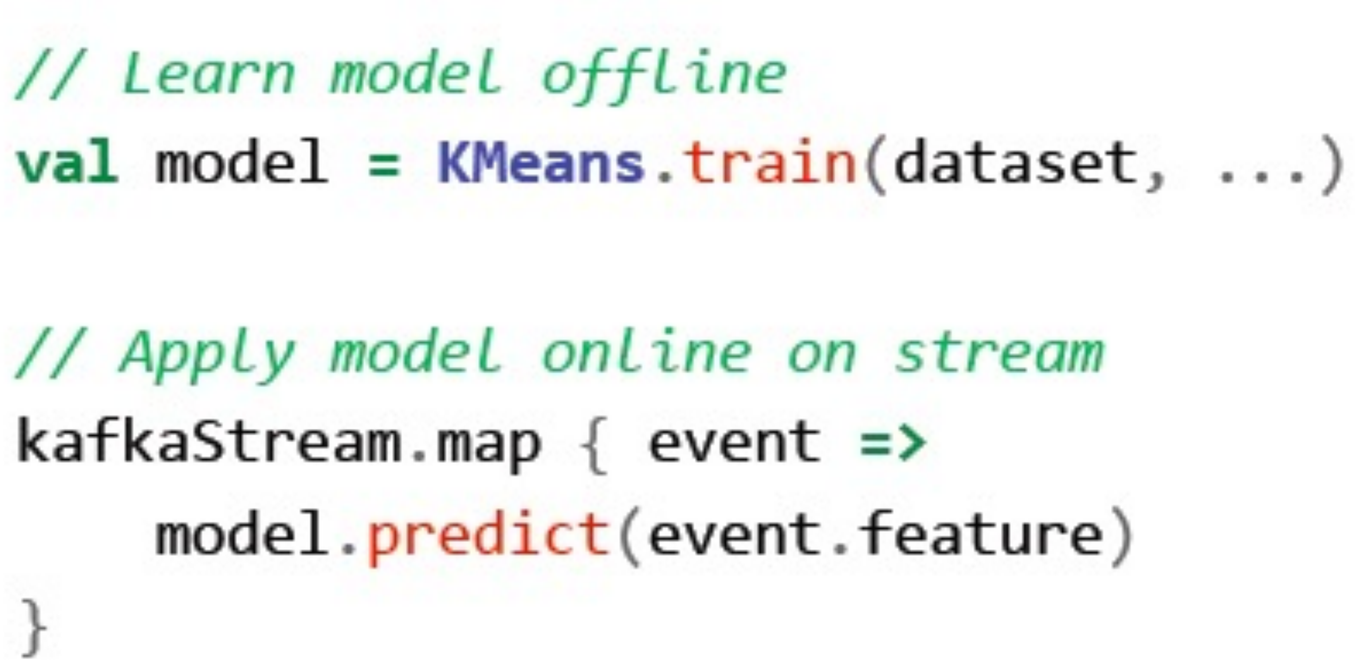
# Collaborative Filtering via Alternating Least Square (ALS) method



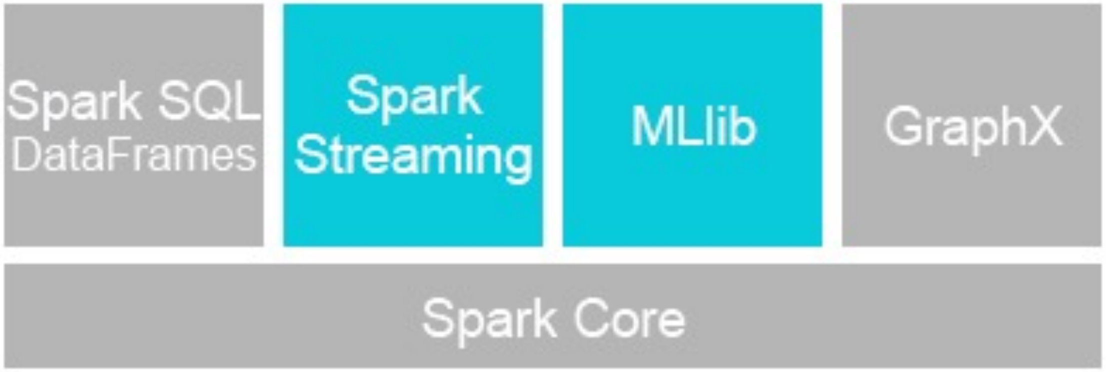
```
// Load and parse the data
val data = sc.textFile("mllib/data/als/test.data")
val ratings = data.map(_.split(',') match {
    case Array(user, item, rate) =>
      Rating(user.toInt, item.toInt, rate.toDouble)
})

// Build the recommendation model using ALS
val model = ALS.train(ratings, 1, 20, 0.01)

// Evaluate the model on rating data
val usersProducts = ratings.map { case Rating(user, product, rate) =>
  (user, product)
}
val predictions = model.predict(usersProducts)
```

# Combine Machine Learning with Streaming

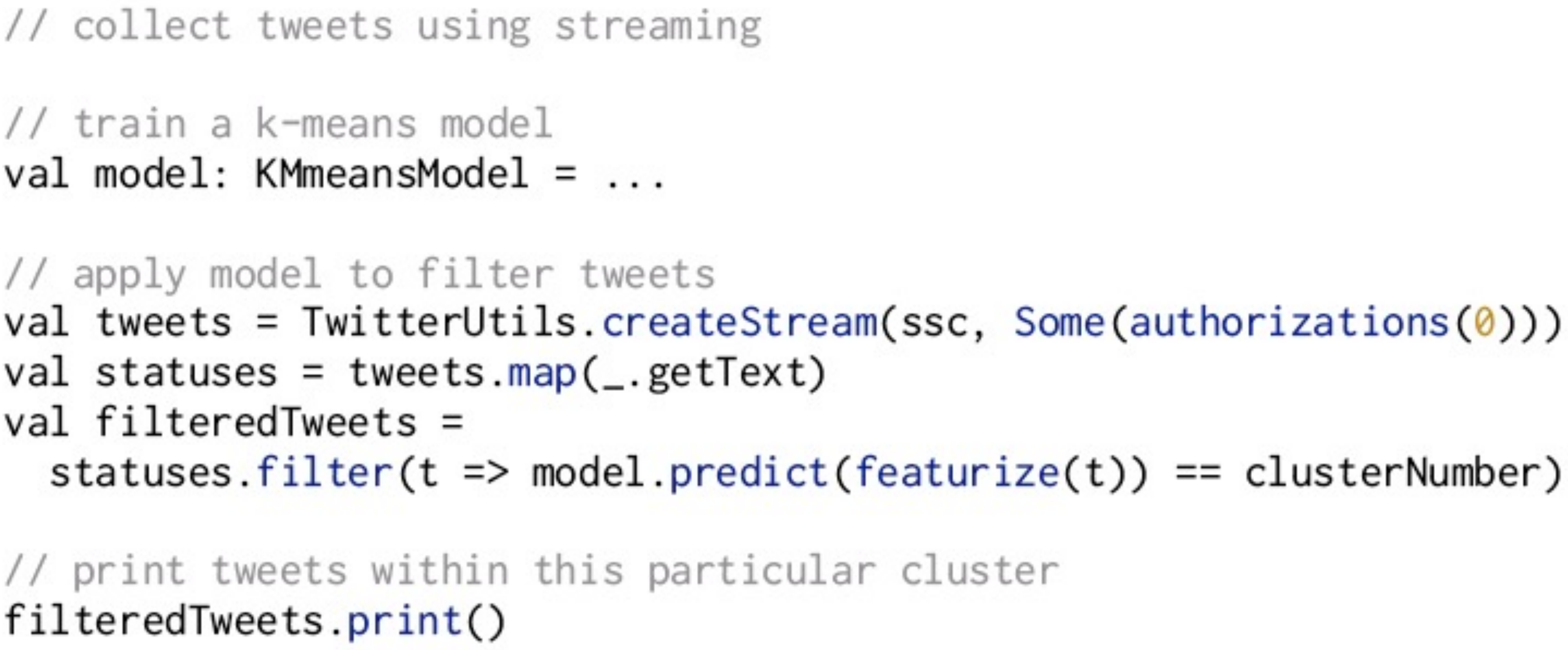
- Learn models offline, apply them online



```scala
// Learn model offline
val model = KMeans.train(dataset, ...)

// Apply model online on stream
kafkaStream.map { event =>
    model.predict(event.feature)
}
```



# Spark Streaming + MLlib

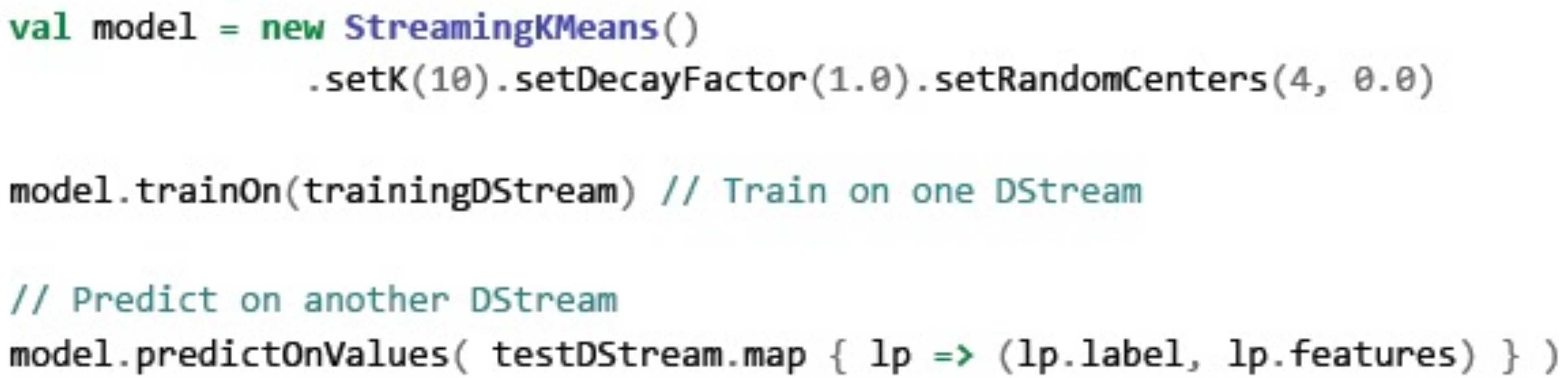


```
// collect tweets using streaming

// train a k-means model
val model: KMmeansModel = ...

// apply model to filter tweets
val tweets = TwitterUtils.createStream(ssc, Some(authorizations(0)))
val statuses = tweets.map(_.getText)
val filteredTweets =
  statuses.filter(t => model.predict(featurize(t)) == clusterNumber)

// print tweets within this particular cluster
filteredTweets.print()
```

# Streaming MLlib Algorithms

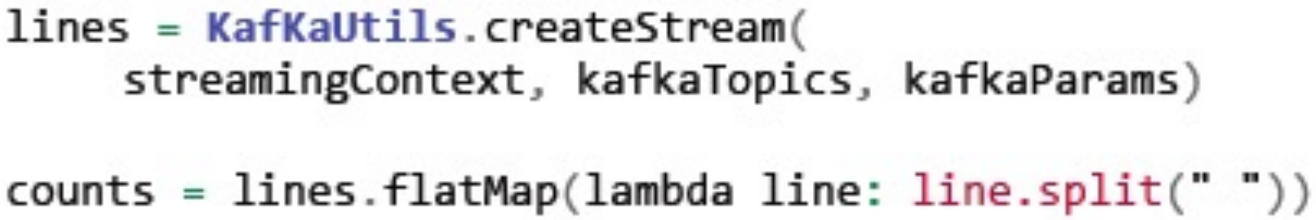
## Continuous learning and prediction on streaming data

StreamingLinearRegression, StreamingKMeans,
StreamingLogisticRegression



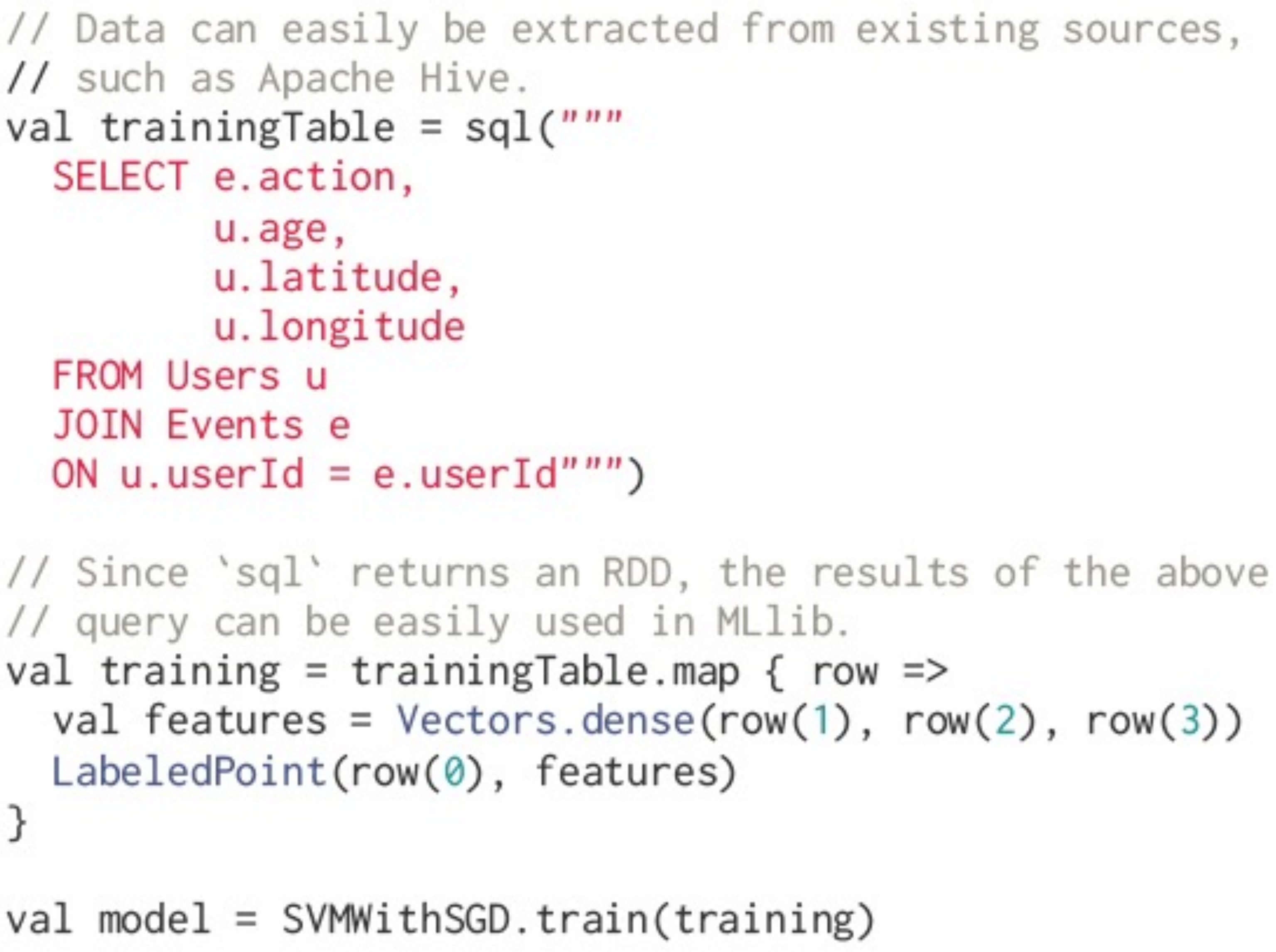
```
val model = new StreamingKMeans()
                  .setK(10).setDecayFactor(1.0).setRandomCenters(4, 0.0)

model.trainOn(trainingDStream) // Train on one DStream

// Predict on another DStream
model.predictOnValues( testDStream.map { lp => (lp.label, lp.features) } )
```

https://databricks.com/blog/2015/01/28/introducing-streaming-k-means-in-spark-1-2.html



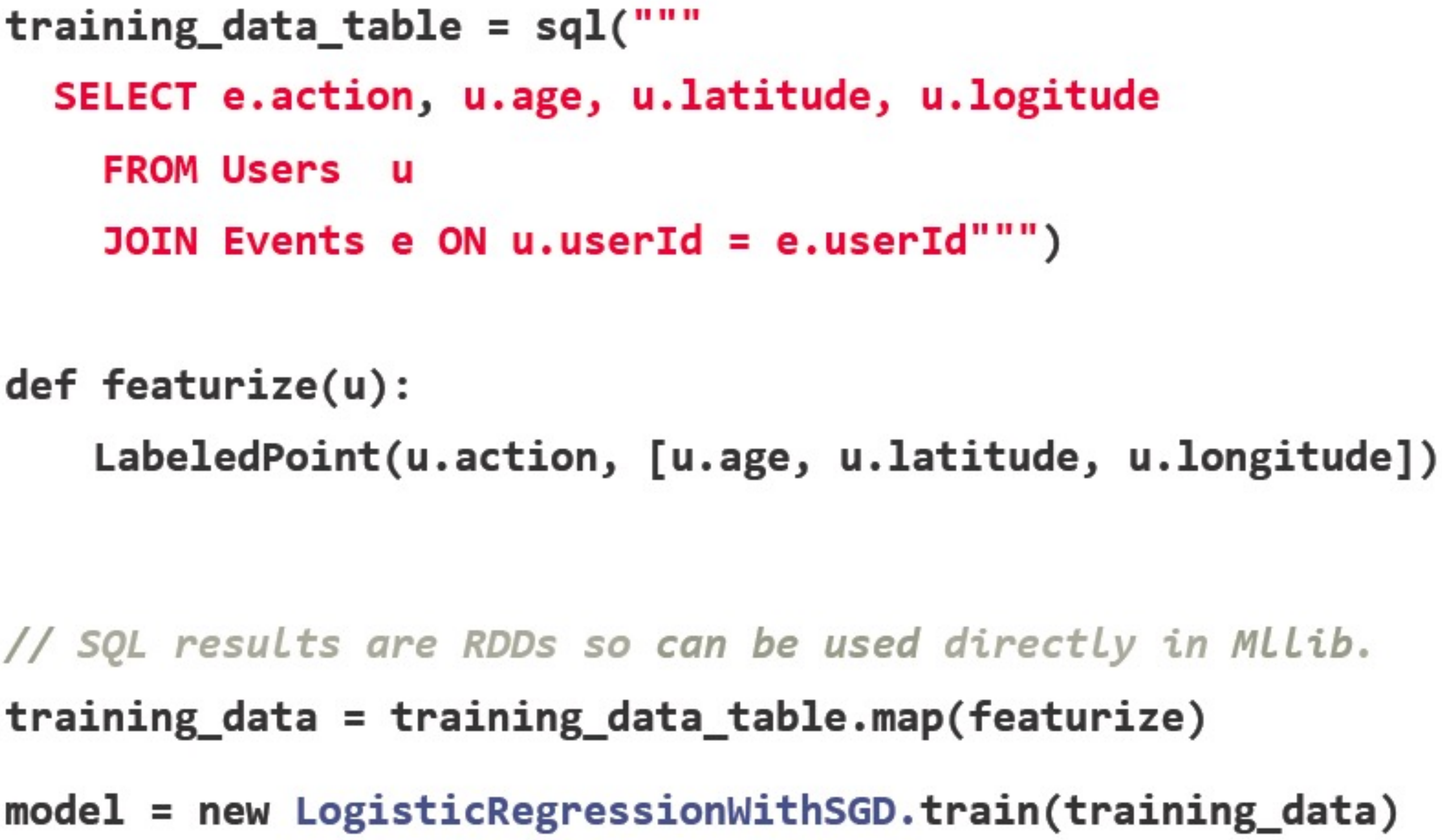
```
lines = KafKaUtils.createStream(
    streamingContext, kafkaTopics, kafkaParams)

counts = lines.flatMap(lambda line: line.split(" "))
```

# Spark SQL + MLlib



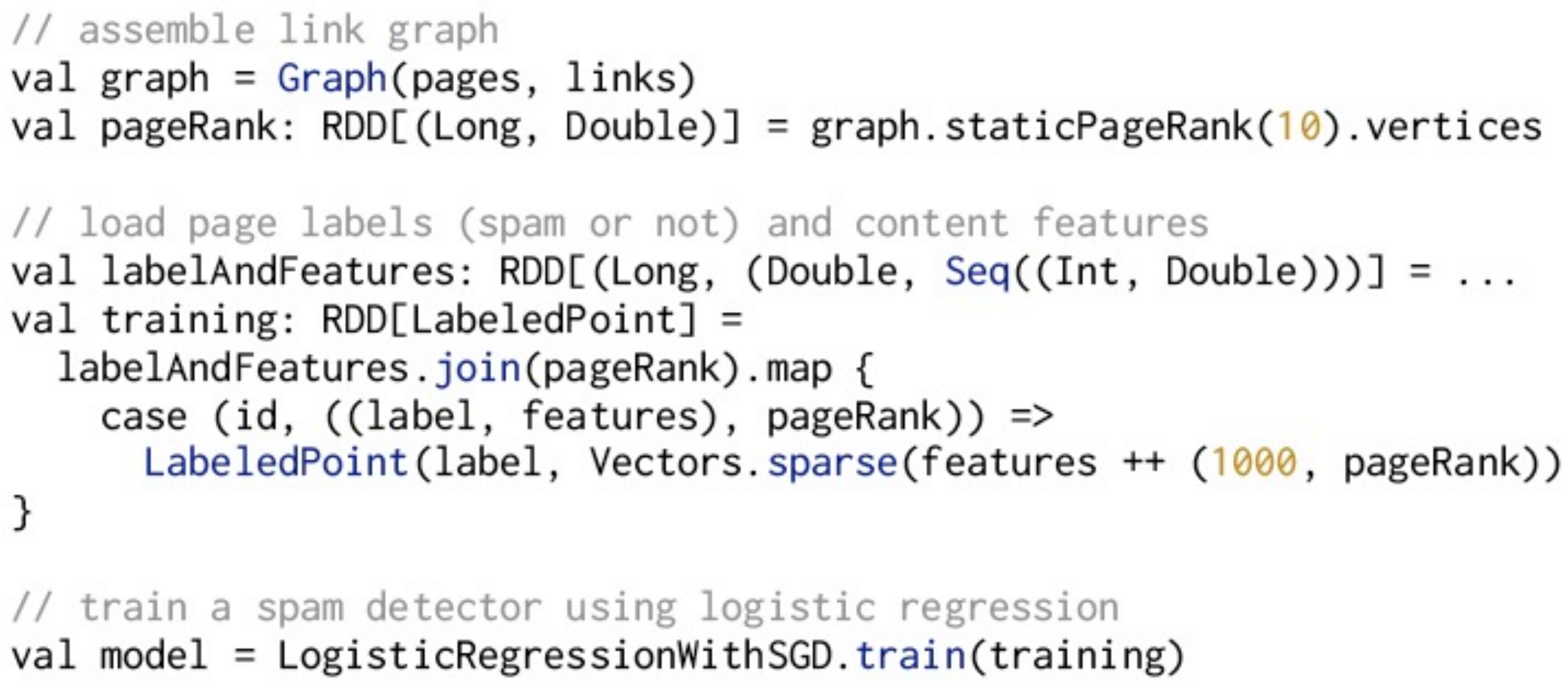
```
// Data can easily be extracted from existing sources,
// such as Apache Hive.
val trainingTable = sql("""
  SELECT e.action,
         u.age,
         u.latitude,
         u.longitude
  FROM Users u
  JOIN Events e
  ON u.userId = e.userId""")

// Since `sql` returns an RDD, the results of the above
// query can be easily used in MLlib.
val training = trainingTable.map { row =>
  val features = Vectors.dense(row(1), row(2), row(3))
  LabeledPoint(row(0), features)
}

val model = SVMWithSGD.train(training)
```

# Spark SQL and MLlib



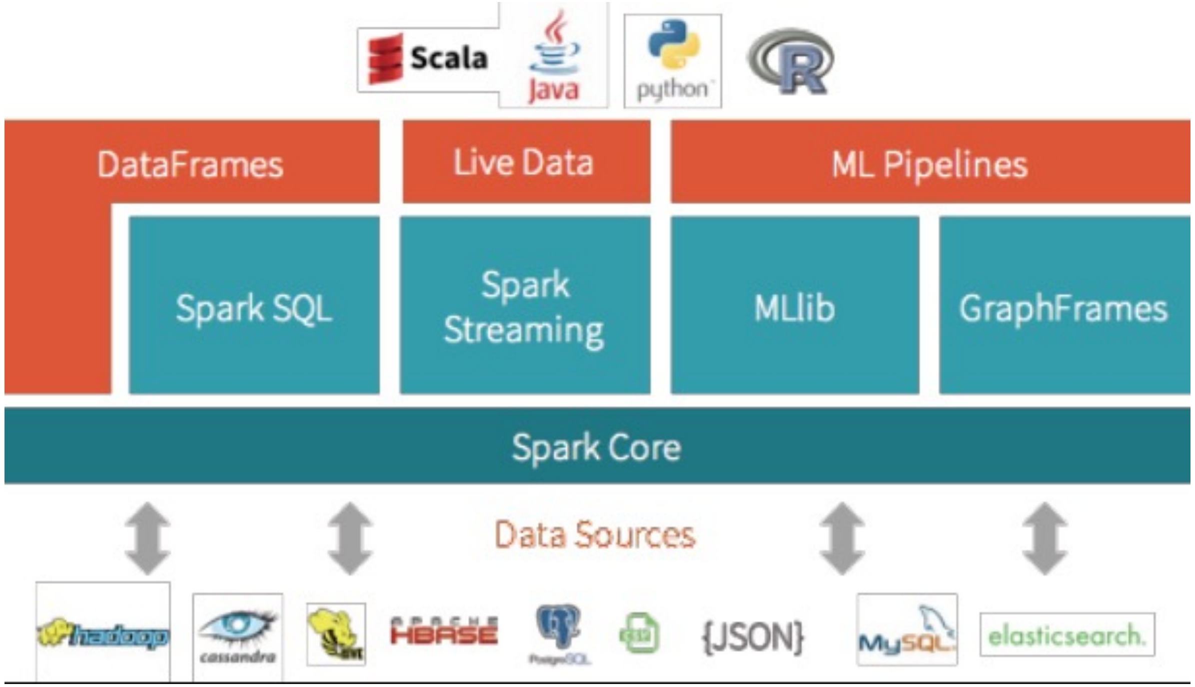
```
training_data_table = sql("""
  SELECT e.action, u.age, u.latitude, u.logitude
    FROM Users  u
    JOIN Events e ON u.userId = e.userId""")

def featurize(u):
    LabeledPoint(u.action, [u.age, u.latitude, u.longitude])


// SQL results are RDDs so can be used directly in MLlib.
training_data = training_data_table.map(featurize)
model = new LogisticRegressionWithSGD.train(training_data)
```

# GraphX + MLlib



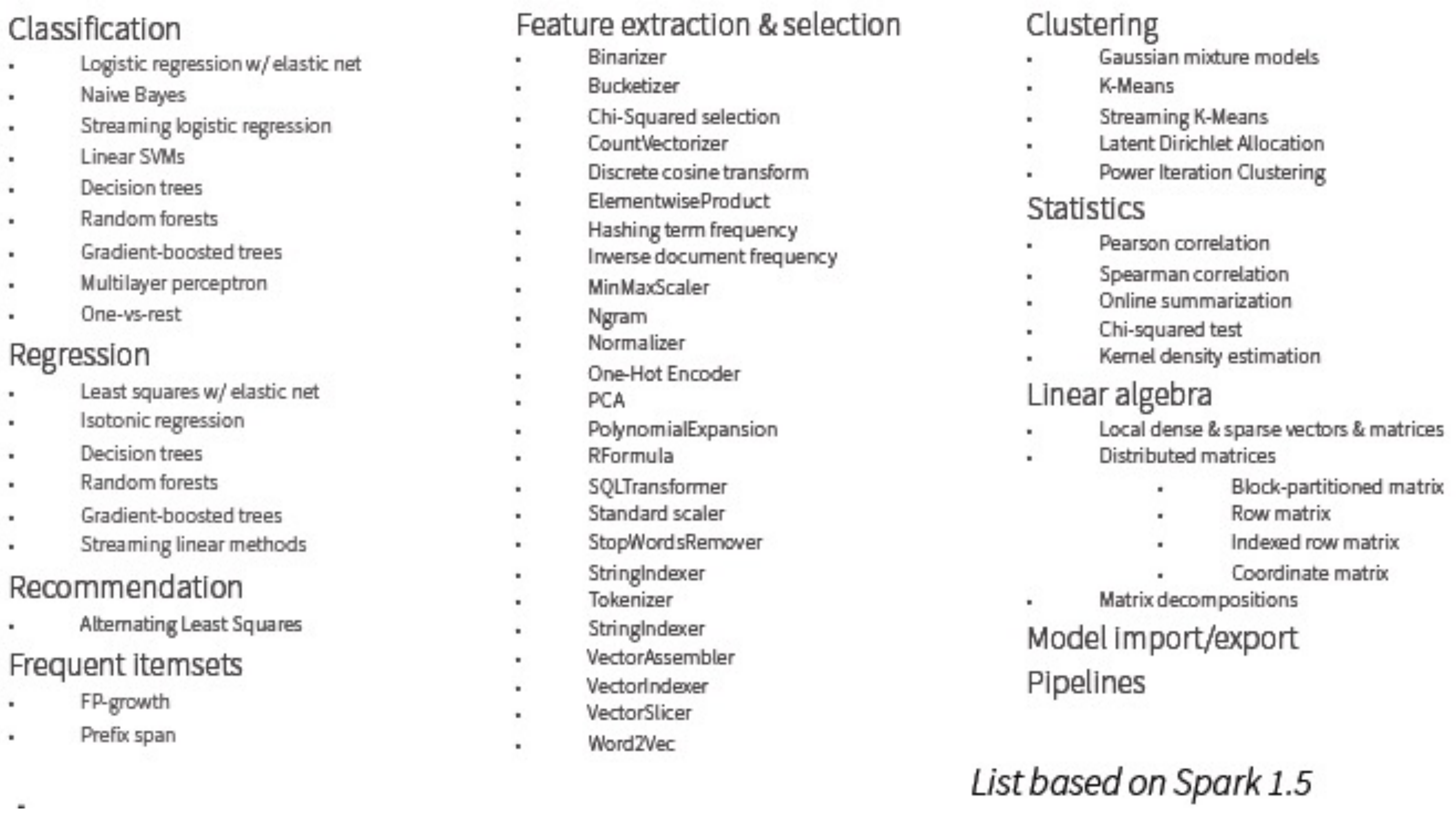
```
// assemble link graph
val graph = Graph(pages, links)
val pageRank: RDD[(Long, Double)] = graph.staticPageRank(10).vertices

// load page labels (spam or not) and content features
val labelAndFeatures: RDD[(Long, (Double, Seq((Int, Double)))] = ...
val training: RDD[LabeledPoint] =
  labelAndFeatures.join(pageRank).map {
    case (id, ((label, features), pageRank)) =>
      LabeledPoint(label, Vectors.sparse(features ++ (1000, pageRank))
}

// train a spam detector using logistic regression
val model = LogisticRegressionWithSGD.train(training)
```

# Evolution of the  Apache Spark MLlib



- Started with Spark 0.8 in AMPLab in 2014 with RDD-based API.
- Migration to Spark DataFrames (aka spark.ml) since v1.3 and aims to achieve feature-parity by v2.3.
- RDD-based MLlib API entered maintenance mode (i.e. no new features added) since v2.0 ; expected to be removed by v3.0.
- Contributions by 75+ orgs, ~250 individuals.
- Distributed algorithms that scale linearly with the data.
- In 2018, the MLflow platform (mlflow.spark) emerged to provide API for logging & loading MLlib models to enhance the complete ML lifecycle.

# Machine Learning Support for Spark (via MLlib or spark.ml)



## Classification

- Logistic regression w/ elastic net
- Naive Bayes
- Streaming logistic regression
- Linear SVMs
- Decision trees
- Random forests
- Gradient-boosted trees
- Multilayer perceptron
- One-vs-rest

## Regression

- Least squares w/ elastic net
- Isotonic regression
- Decision trees
- Random forests
- Gradient-boosted trees
- Streaming linear methods

## Recommendation

- Alternating Least Squares

## Frequent itemsets

- FP-growth
- Prefix span

## Feature extraction & selection

- Binarizer
- Bucketizer
- Chi-Squared selection
- CountVectorizer
- Discrete cosine transform
- ElementwiseProduct
- Hashing term frequency
- Inverse document frequency
- MinMaxScaler
- Ngram
- Normalizer
- One-Hot Encoder
- PCA
- PolynomialExpansion
- RFormula
- SQLTransformer
- Standard scaler
- StopWordsRemover
- StringIndexer
- Tokenizer
- StringIndexer
- VectorAssembler
- VectorIndexer
- VectorSlicer
- Word2Vec

## Clustering

- Gaussian mixture models
- K-Means
- Streaming K-Means
- Latent Dirichlet Allocation
- Power Iteration Clustering

## Statistics

- Pearson correlation
- Spearman correlation
- Online summarization
- Chi-squared test
- Kernel density estimation

## Linear algebra

- Local dense & sparse vectors & matrices
- Distributed matrices
  - Block-partitioned matrix
  - Row matrix
  - Indexed row matrix
  - Coordinate matrix
- Matrix decompositions

## Model import/export

## Pipelines

*List based on Spark 1.5*

# Machine Learning Support for Spark (via MLlib or spark.ml)

*List based on Spark 2.0*

## Classification

- Logistic regression
- Naive Bayes
- Streaming logistic regression
- Linear SVMs
- Decision trees
- Random forests
- Gradient-boosted trees
- Multilayer perceptron

## Regression

- Ordinary least squares
- Ridge regression
- Lasso
- Isotonic regression
- Decision trees
- Random forests
- Gradient-boosted trees
- Streaming linear methods
- **Generalized Linear Models**

## Frequent itemsets

- FP-growth
- PrefixSpan

## Recommendation

- Alternating Least Squares

## Feature extraction & selection

- Word2Vec
- Chi-Squared selection
- Hashing term frequency
- Inverse document frequency
- Normalizer
- Standard scaler
- Tokenizer
- One-Hot Encoder
- StringIndexer
- VectorIndexer
- VectorAssembler
- Binarizer
- Bucketizer
- ElementwiseProduct
- PolynomialExpansion
- Quantile discretizer
- SQL transformer

## Model import/export

## Pipelines

## Clustering

- Gaussian mixture models
- K-Means
- Streaming K-Means
- Latent Dirichlet Allocation
- Power Iteration Clustering
- Bisecting K-Means

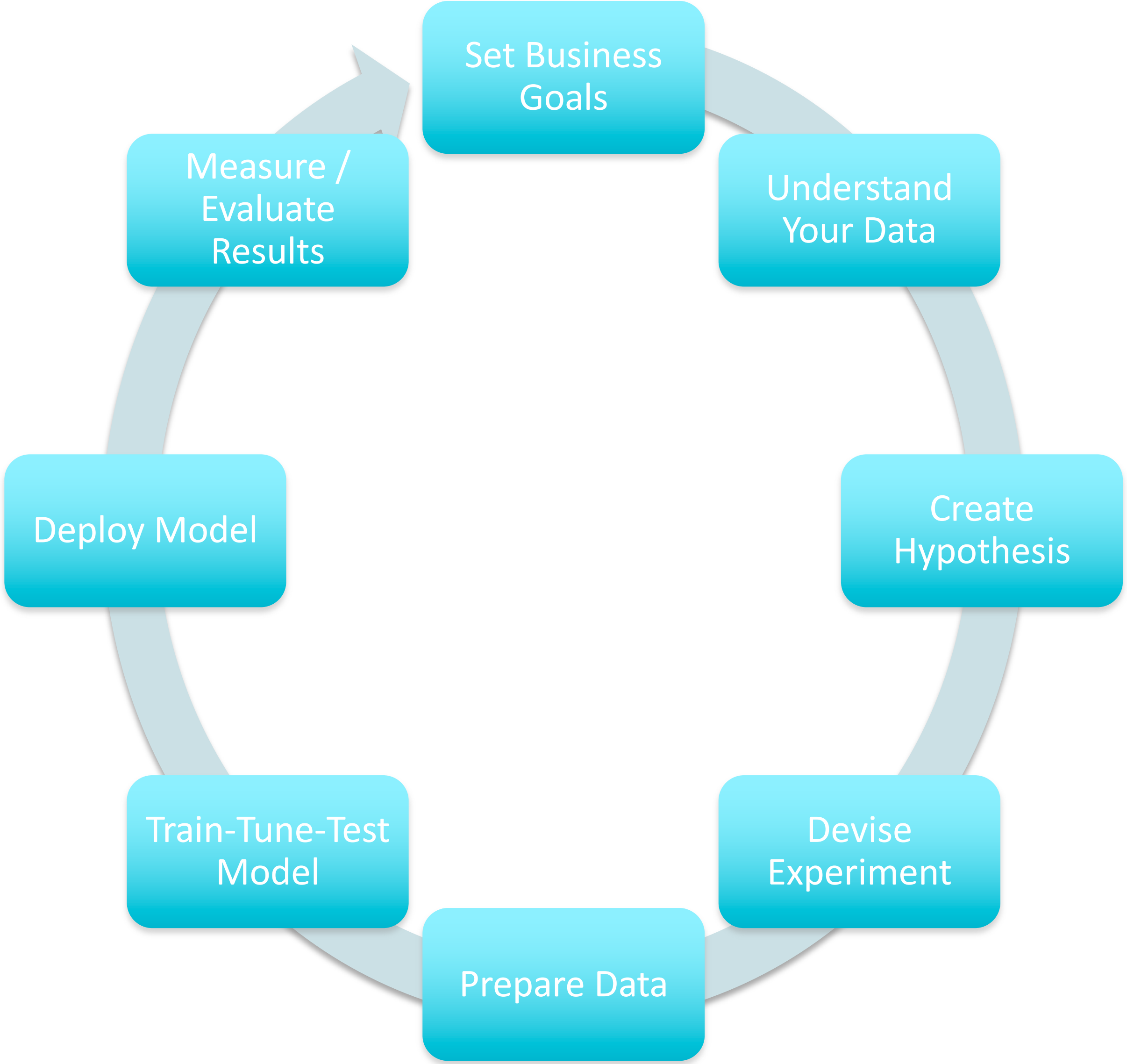
## Statistics

- Pearson correlation
- Spearman correlation
- Online summarization
- Chi-squared test
- Kernel density estimation
- Kolmogorov–Smirnov test
- Online hypothesis testing
- Survival analysis

## Linear algebra

- Local dense & sparse vectors & matrices
- Normal equation for least squares
- Distributed matrices
    - Block-partitioned matrix
    - Row matrix
    - Indexed row matrix
    - Coordinate matrix
- Matrix decompositions

# An Ideal ML Workflow

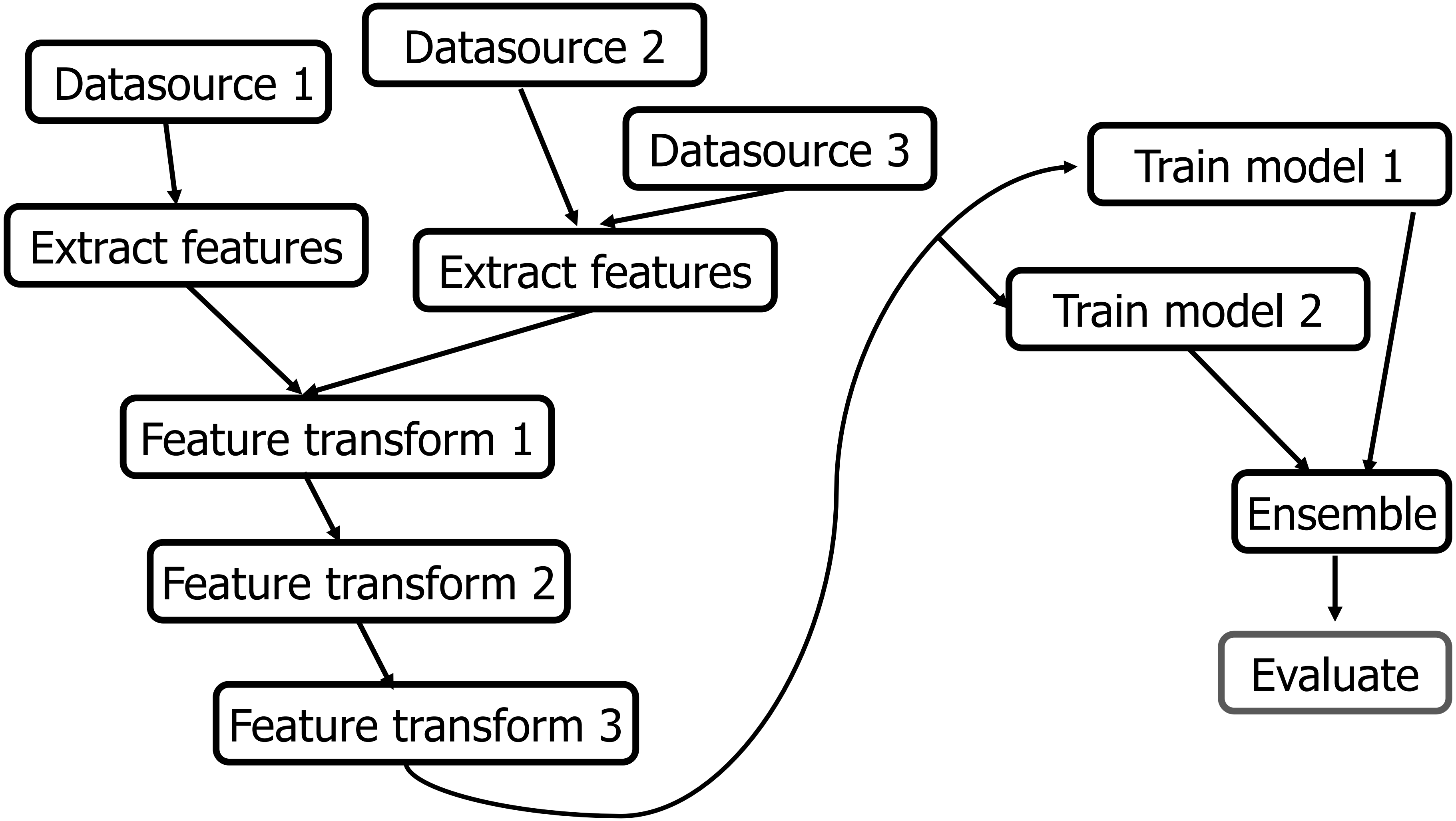


# But Real-World ML workflows are complex

- Specify the pipeline
- Re-run on new data
- Inspect the results
- Tune the parameters

- Usually, each step of a pipeline is easier with one framework

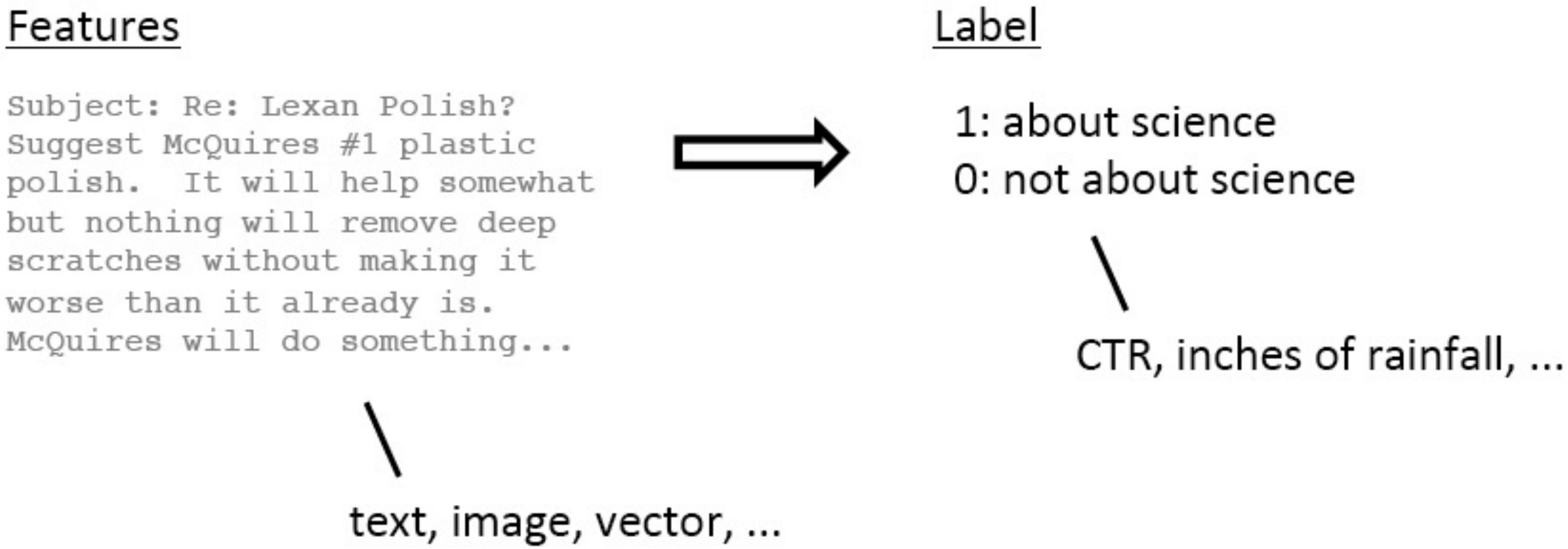
# Real-world ML Workflows (Pipelines) are Complex



# Example: Text Classification

Goal: Given a text document, predict its topic.

Dataset: "20 Newsgroups"
*From UCI KDD Archive*



Features

```
Subject: Re: Lexan Polish?
Suggest McQuires #1 plastic
polish.  It will help somewhat
but nothing will remove deep
scratches without making it
worse than it already is.
McQuires will do something...
```

text, image, vector, ...

Label

1: about science
0: not about science

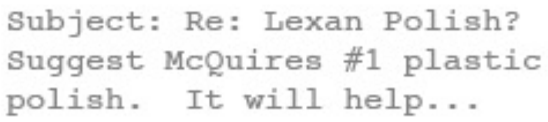
CTR, inches of rainfall, ...

# Training & Testing

## Training

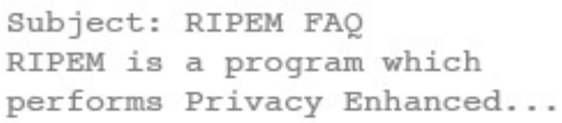
Given labeled data:

RDD of (features, label)



```
Subject: Re: Lexan Polish?
Suggest McQuires #1 plastic
polish.  It will help...
```

Label 0



```
Subject: RIPEM FAQ
RIPEM is a program which
performs Privacy Enhanced...
```

Label 1

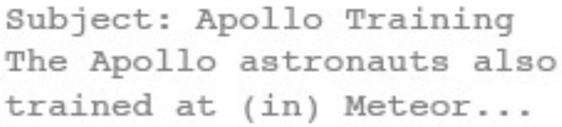
...

Learn a model.

## Testing/Production

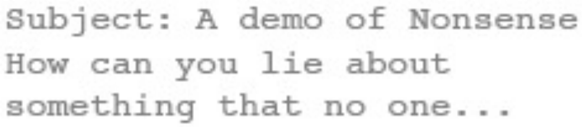
Given new unlabeled data:

RDD of features



```
Subject: Apollo Training
The Apollo astronauts also
trained at (in) Meteor...
```

→ Label 1

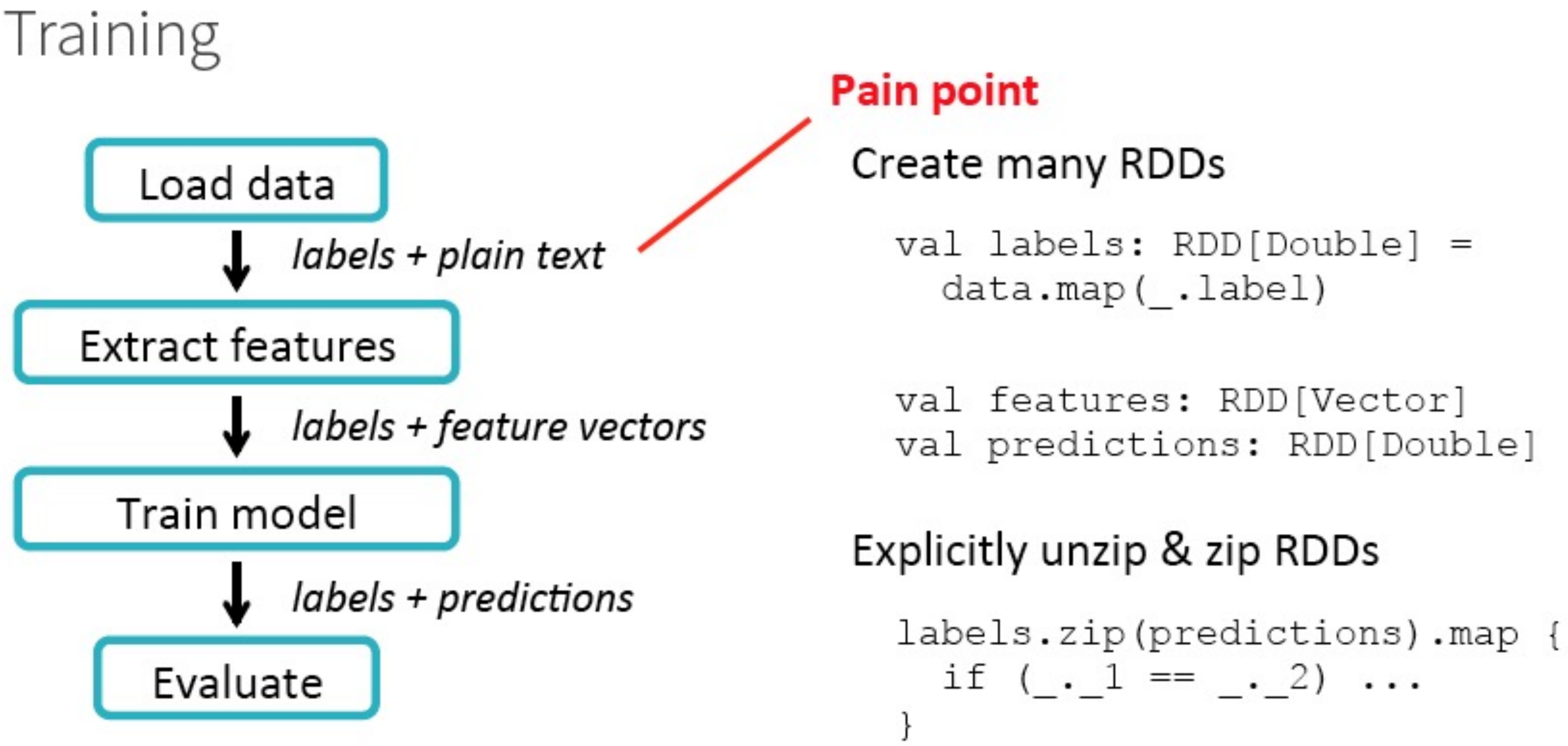


```
Subject: A demo of Nonsense
How can you lie about
something that no one...
```

→ Label 0

Use model to make predictions.

# Example ML Workflow



## Training

Load data

↓ *labels + plain text*

Extract features

↓ *labels + feature vectors*

Train model

↓ *labels + predictions*

Evaluate

**Pain point**

Create many RDDs

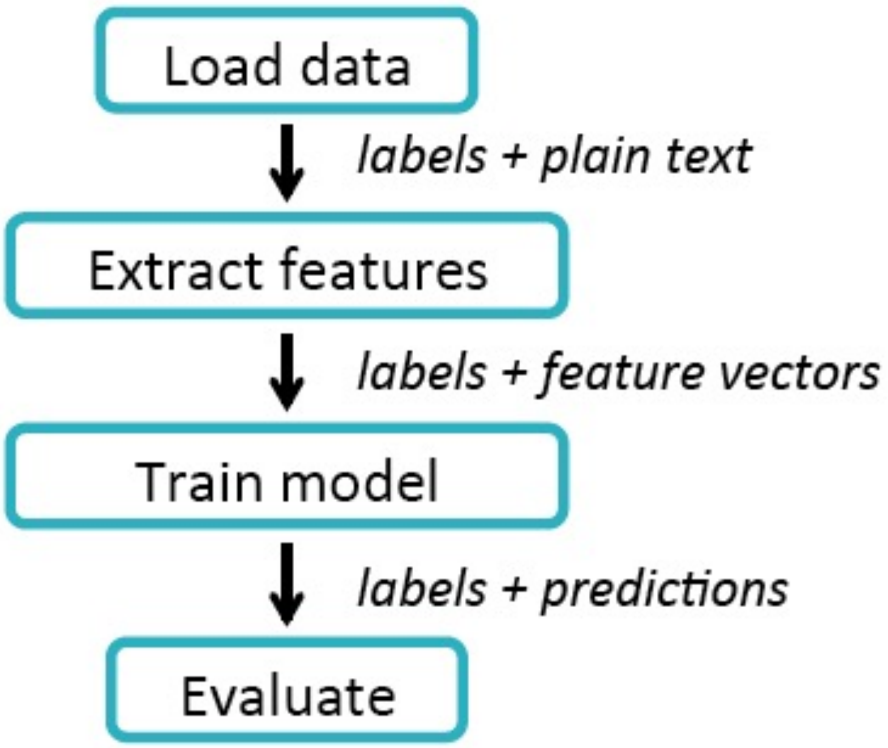
```
val labels: RDD[Double] =
  data.map(_.label)


val features: RDD[Vector]
val predictions: RDD[Double]
```

Explicitly unzip & zip RDDs

```
labels.zip(predictions).map {
  if (_._1 == _._2) ...
}
```

# Example ML Workflow

## Training



Load data

*labels + plain text*

Extract features

*labels + feature vectors*

Train model

*labels + predictions*

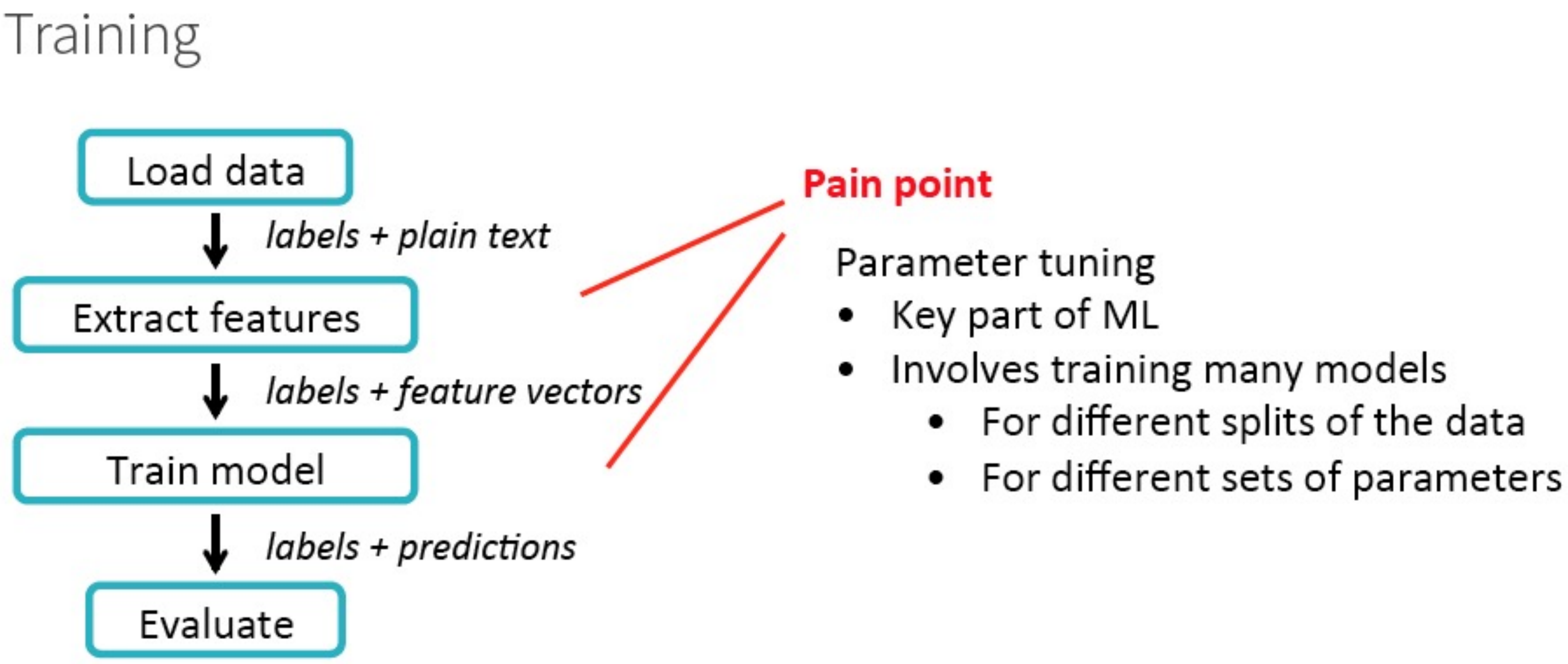
Evaluate

**Pain point**

Write as a script

- Not modular
- Difficult to re-use workflow

# Example ML Workflow



Training

Load data

*labels + plain text*

Extract features

*labels + feature vectors*

Train model

*labels + predictions*

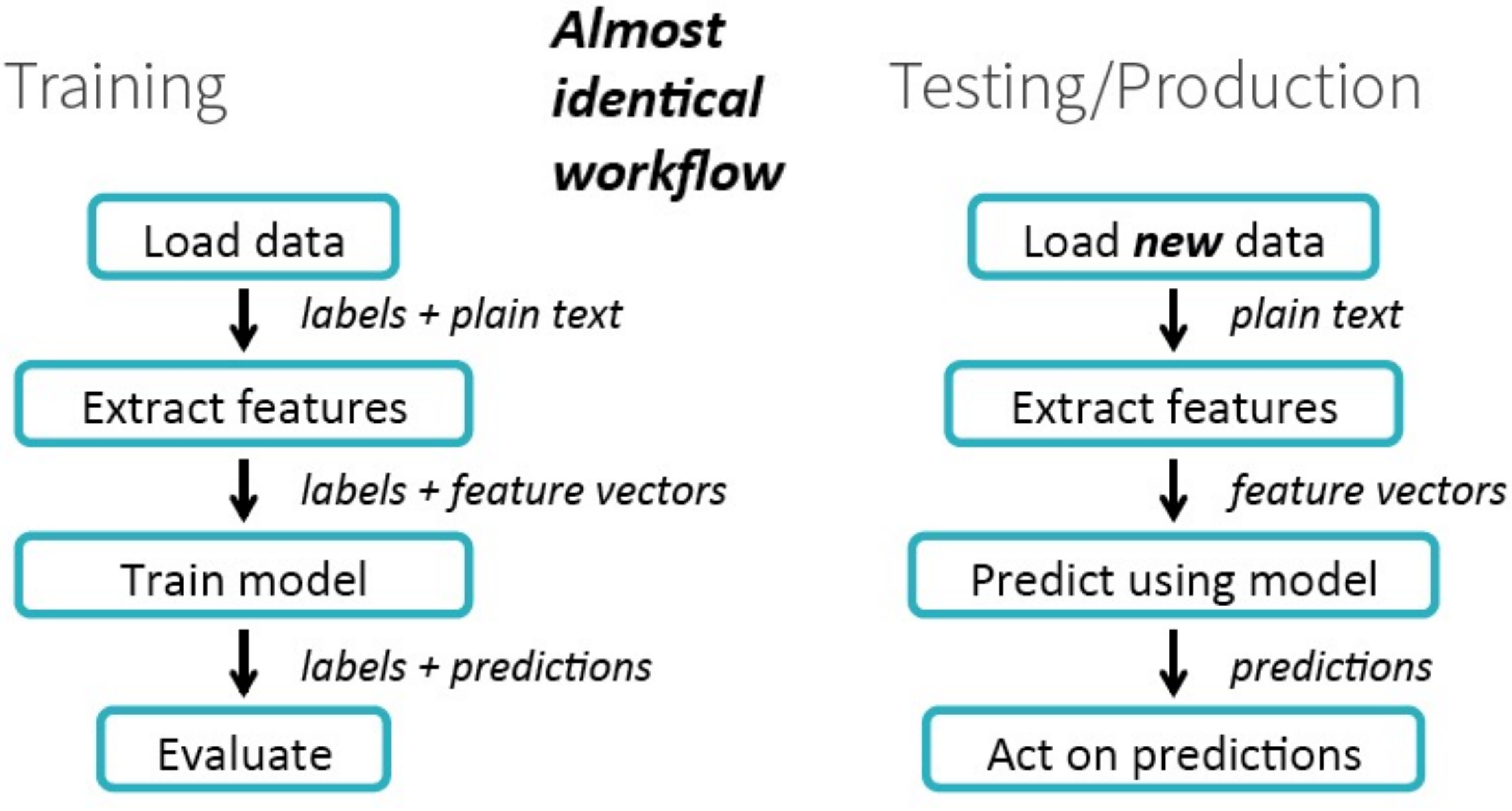
Evaluate

**Pain point**

Parameter tuning

- Key part of ML
- Involves training many models
  - For different splits of the data
  - For different sets of parameters

# Example ML Workflow



Training

*Almost identical workflow*

Testing/Production

Load data

*labels + plain text*

Extract features

*labels + feature vectors*

Train model

*labels + predictions*

Evaluate

Load ***new*** data

*plain text*

Extract features

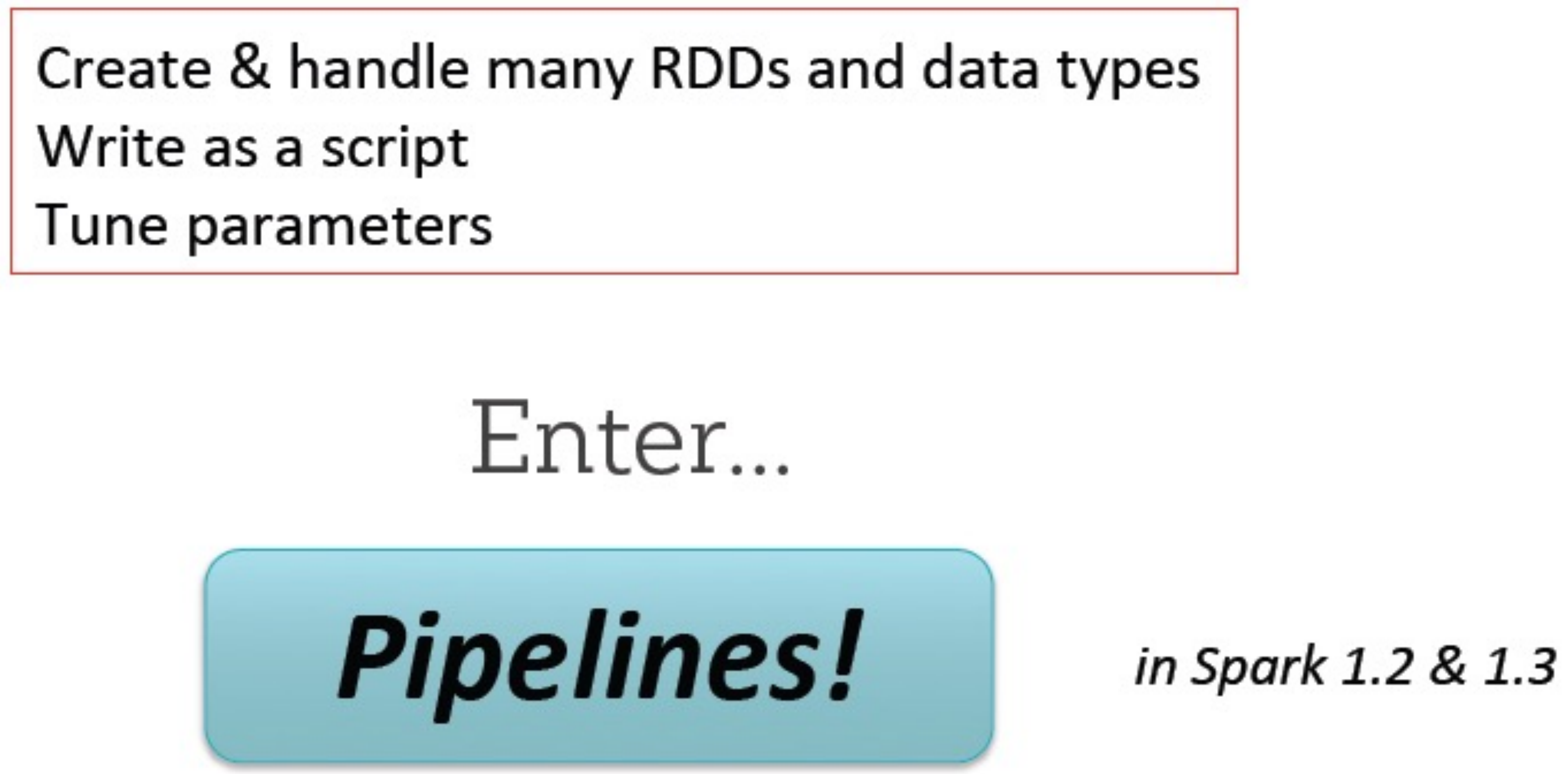
*feature vectors*

Predict using model

*predictions*

Act on predictions

# Recap the Pain Points



Create & handle many RDDs and data types

Write as a script

Tune parameters

Enter...

**_Pipelines!_**

*in Spark 1.2 & 1.3*

# Apache Spark – ML Pipelines

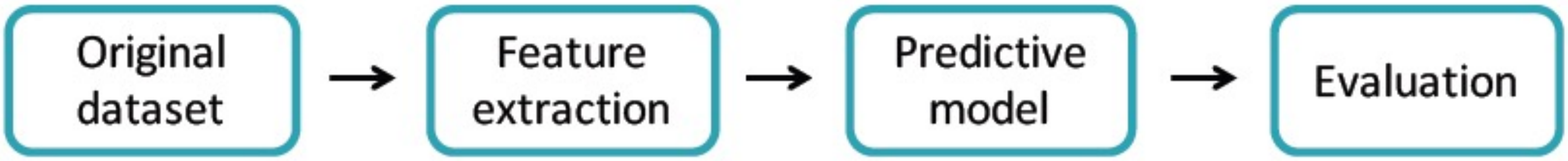
- spark.ml
- Spark > 1.4
- Spark.ML pipelines – able to create more complex models
- Integrated with DataFrames

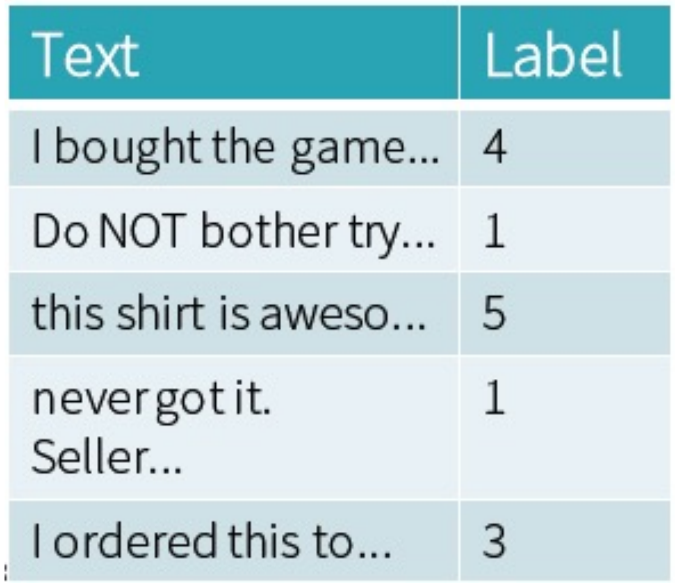
```
// Let's initialize our linear
regression learner
val lr = new LinearRegression()

// Now we set the parameters for the
method
lr.setPredictionCol("Predicted_PE")
.setLabelCol("PE").setMaxIter(100).setRe
gParam(0.1)

// We will use the new spark.ml pipeline
API. If you have worked with scikit-
learn this will be very familiar.

val lrPipeline = new Pipeline()
lrPipeline.setStages(Array(vectorizer,
lr))

// Let's first train on the entire
dataset to see what we get
val lrModel =
lrPipeline.fit(trainingSet)
```

# Key Concepts of ML Pipeline in Spark

- DataFrames: Provide a Unified API to hold the ML Dataset
  - Structured Data with Flexible Types
  - Add & Remove Columns during ML Pipeline execution
  - Distributed, Optimized Implementation
- The notion of Pipeline in Spark MLlib
  - To support Simple construction and Tuning of Machine Learning Workflows
- Abstractions for ML Pipeline:
  - Transformers, Estimators and Evaluators
- Parameters: API & Tuning
- ML Pipeline API similar to scikit-learn

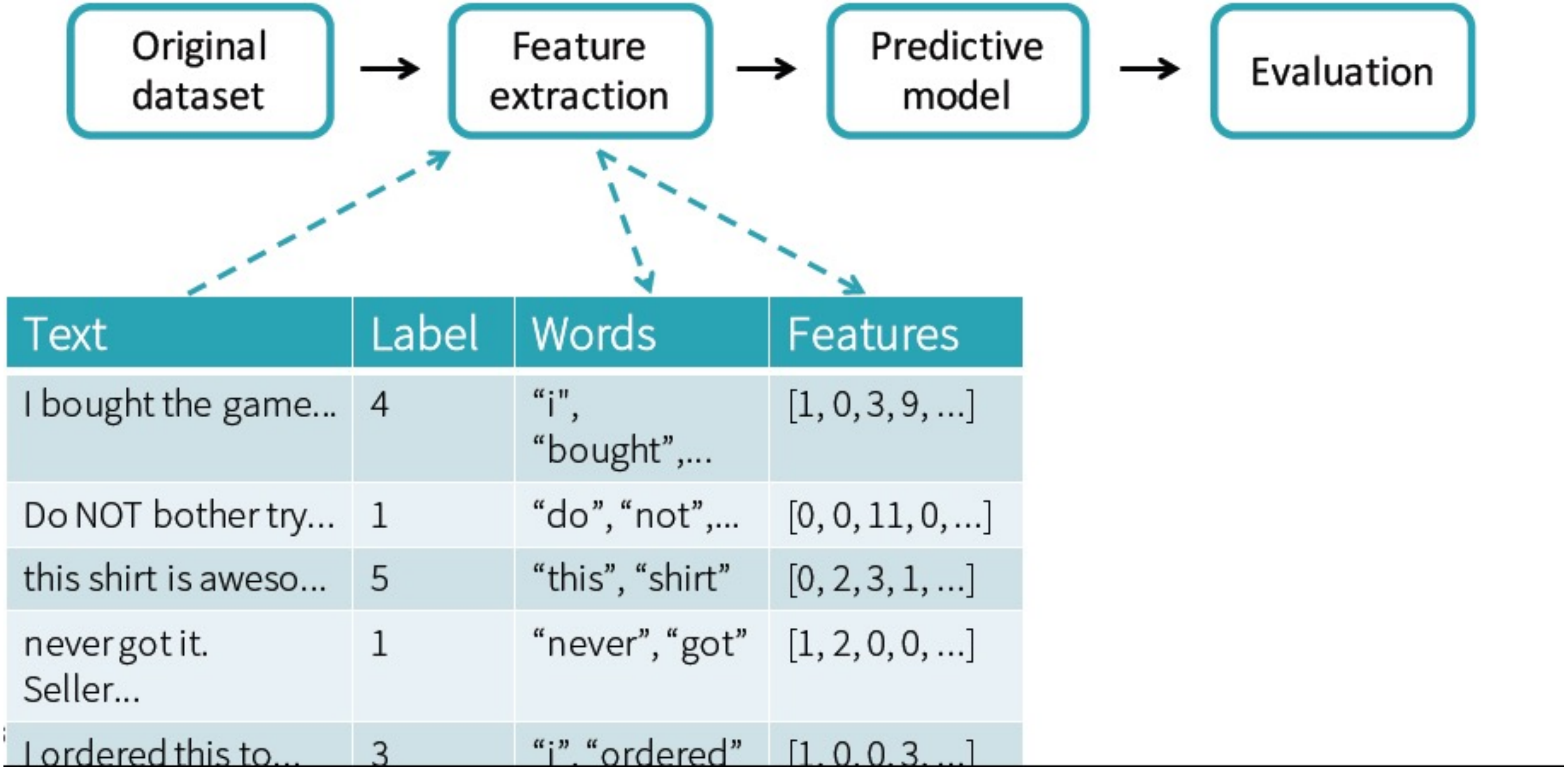
# Use a DataFrame to hold the ML dataset under processing


Original dataset → Feature extraction → Predictive model → Evaluation



| Text | Label |
|---|---|
| I bought the game... | 4 |
| Do NOT bother try... | 1 |
| this shirt is aweso... | 5 |
| never got it. Seller... | 1 |
| I ordered this to... | 3 |

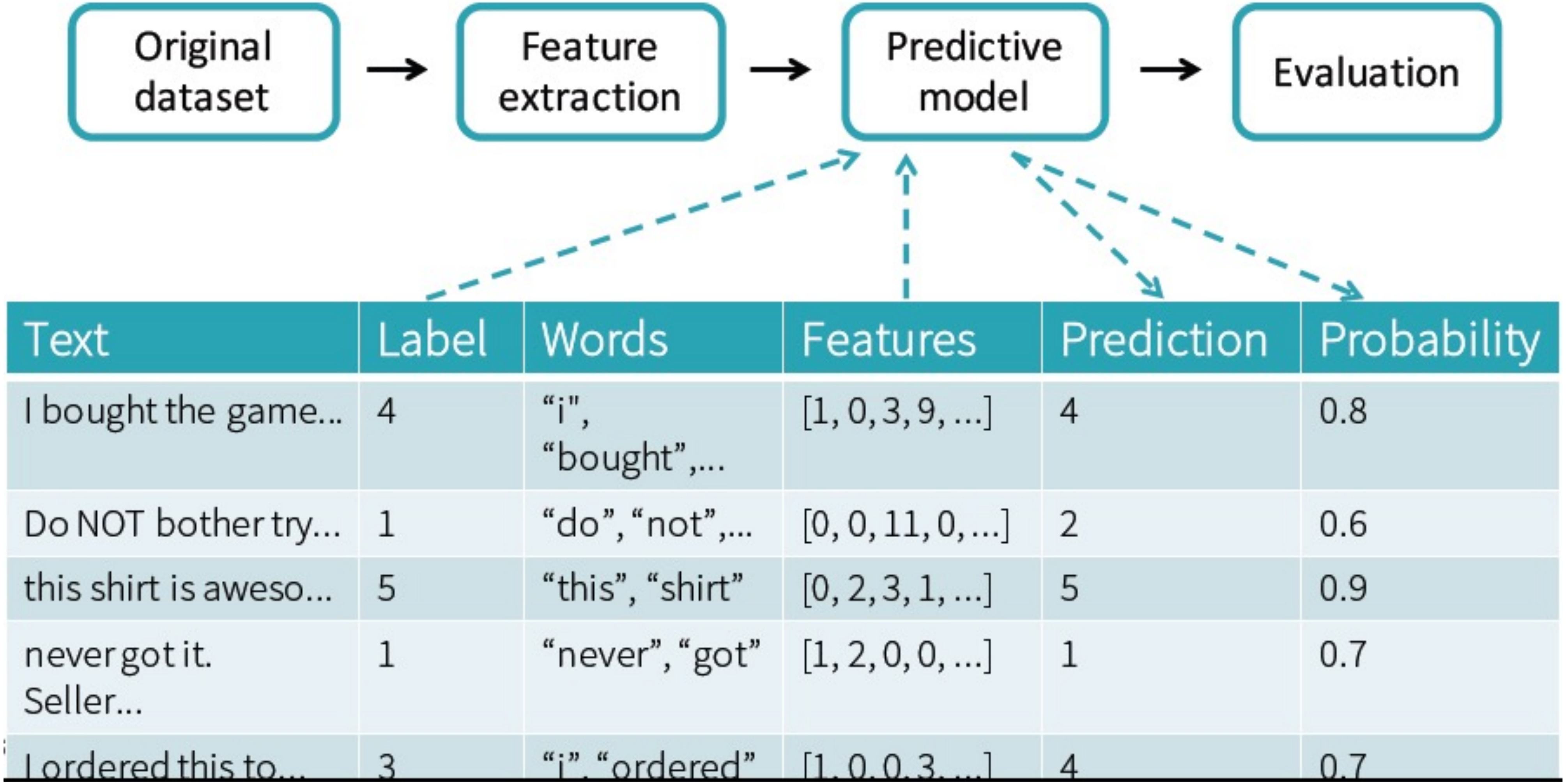
# Extract Features



Original dataset → Feature extraction → Predictive model → Evaluation

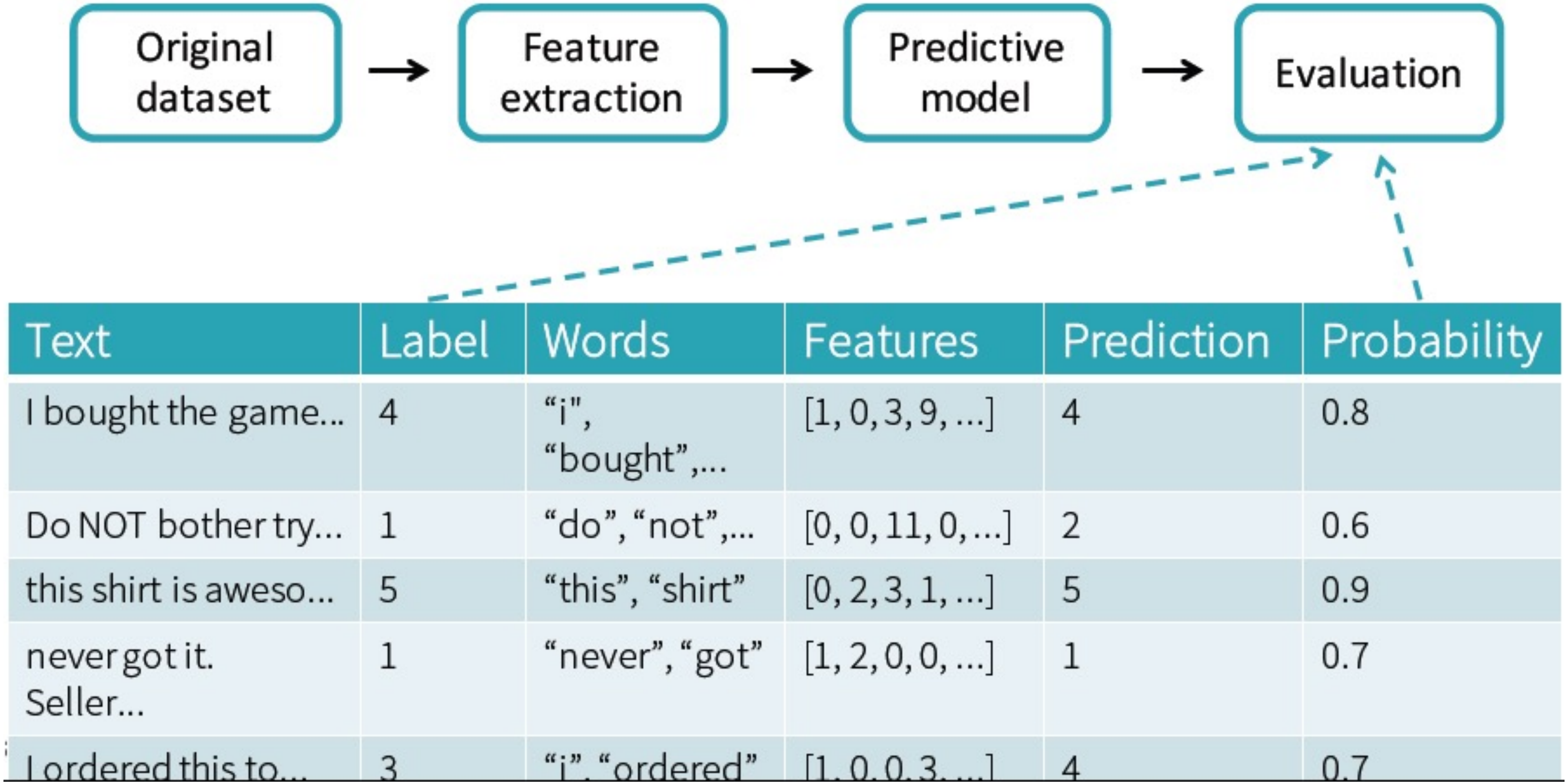
| Text | Label | Words | Features |
|---|---|---|---|
| I bought the game... | 4 | "i",<br>"bought",... | [1, 0, 3, 9, ...] |
| Do NOT bother try... | 1 | "do", "not",... | [0, 0, 11, 0, ...] |
| this shirt is aweso... | 5 | "this", "shirt" | [0, 2, 3, 1, ...] |
| never got it. Seller... | 1 | "never", "got" | [1, 2, 0, 0, ...] |
| I ordered this to... | 3 | "i", "ordered" | [1, 0, 0, 3, ...] |

# Fit a Model



| Text | Label | Words | Features | Prediction | Probability |
|---|---|---|---|---|---|
| I bought the game... | 4 | "i",<br>"bought",... | [1, 0, 3, 9, ...] | 4 | 0.8 |
| Do NOT bother try... | 1 | "do", "not",... | [0, 0, 11, 0, ...] | 2 | 0.6 |
| this shirt is aweso... | 5 | "this", "shirt" | [0, 2, 3, 1, ...] | 5 | 0.9 |
| never got it. Seller... | 1 | "never", "got" | [1, 2, 0, 0, ...] | 1 | 0.7 |
| I ordered this to... | 3 | "i", "ordered" | [1, 0, 0, 3, ...] | 4 | 0.7 |

# Evaluate Performance



Original dataset → Feature extraction → Predictive model → Evaluation

| Text | Label | Words | Features | Prediction | Probability |
|---|---|---|---|---|---|
| I bought the game... | 4 | "i", "bought",... | [1, 0, 3, 9, ...] | 4 | 0.8 |
| Do NOT bother try... | 1 | "do", "not",... | [0, 0, 11, 0, ...] | 2 | 0.6 |
| this shirt is aweso... | 5 | "this", "shirt" | [0, 2, 3, 1, ...] | 5 | 0.9 |
| never got it. Seller... | 1 | "never", "got" | [1, 2, 0, 0, ...] | 1 | 0.7 |
| I ordered this to... | 3 | "i", "ordered" | [1, 0, 0, 3, ...] | 4 | 0.7 |

# Key Concepts in ML Pipeline in Spark

- Transformer:
  - An algorithm which can transform one Dataframe into another Dataframe.

  Example: ML model is a Transformer which transforms a DataFrame with features into a DataFrame with predictions
- Estimator:
  - An algorithm which can be fitted on a DataFrame to produce a Transformer.

  Example: A learning algorithm is an Estimator which trains on a Dataframe and produces a model (i.e. a Transformer)
- Pipeline:
  - A sequence of PipelineStages (i.e. a chain of Transformers and Estimators) to be run in a specific order to realize a ML workflow.
- Evaluator:
  - A Transformer which computes a Metric to measure how well a fitted model does on held-out test data
- Parameters: API & Tuning

References:
https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-mllib/spark-mllib-evaluators.html
https://spark.apache.org/docs/latest/ml-pipeline.html#parameters

# More details on Pipeline Components

**Transformers**

- A Transformer is an abstraction that includes feature transformers and learned models. Technically, a Transformer implements a method transform(), which converts one DataFrame into another, generally by appending one or more columns. For example:
- A feature transformer might take a DataFrame, read a column (e.g., text), map it into a new column (e.g., feature vectors), and output a new DataFrame with the mapped column appended.
- A learning model might take a DataFrame, read the column containing feature vectors, predict the label for each feature vector, and output a new DataFrame with predicted labels appended as a column.

**Estimators**

- An Estimator abstracts the concept of a learning algorithm or any algorithm that fits or trains on data. Technically, an Estimator implements a method fit(), which accepts a DataFrame and produces a Model, which is a Transformer. For example, a learning algorithm such as LogisticRegression is an Estimator, and calling fit() trains a LogisticRegressionModel, which is a Model and hence a Transformer.
- A Predictor is a specialization of Estimator for a PredictionModel with its own abstract train method()

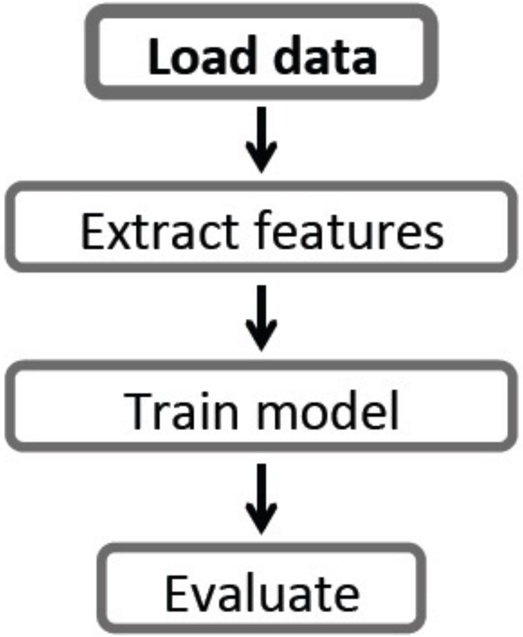
**Evaluators**

- An Evaluator is a transformer that maps a Dataframe into a metric showing how good a model is.

**Params and ParamMap:**

- MLlib Estimators and Transformers use an uniform API for specifying parameters
- A Param is a named parameter with self-contained doc ; A ParamMap is a set of (parameter, value) pairs
- Two main ways to pass parameters to an algorithm:
  - Set parameters for an instance. E.g., if lr is an instance of LogisticRegression, one could call lr.setMaxIter(10) to make lr.fit() use at most 10 iterations. This API resembles the API used in spark.mllib package.
  - Pass a ParamMap to fit() or transform(). Any parameters in the ParamMap will override parameters previously specified via setter methods.

# Load Data

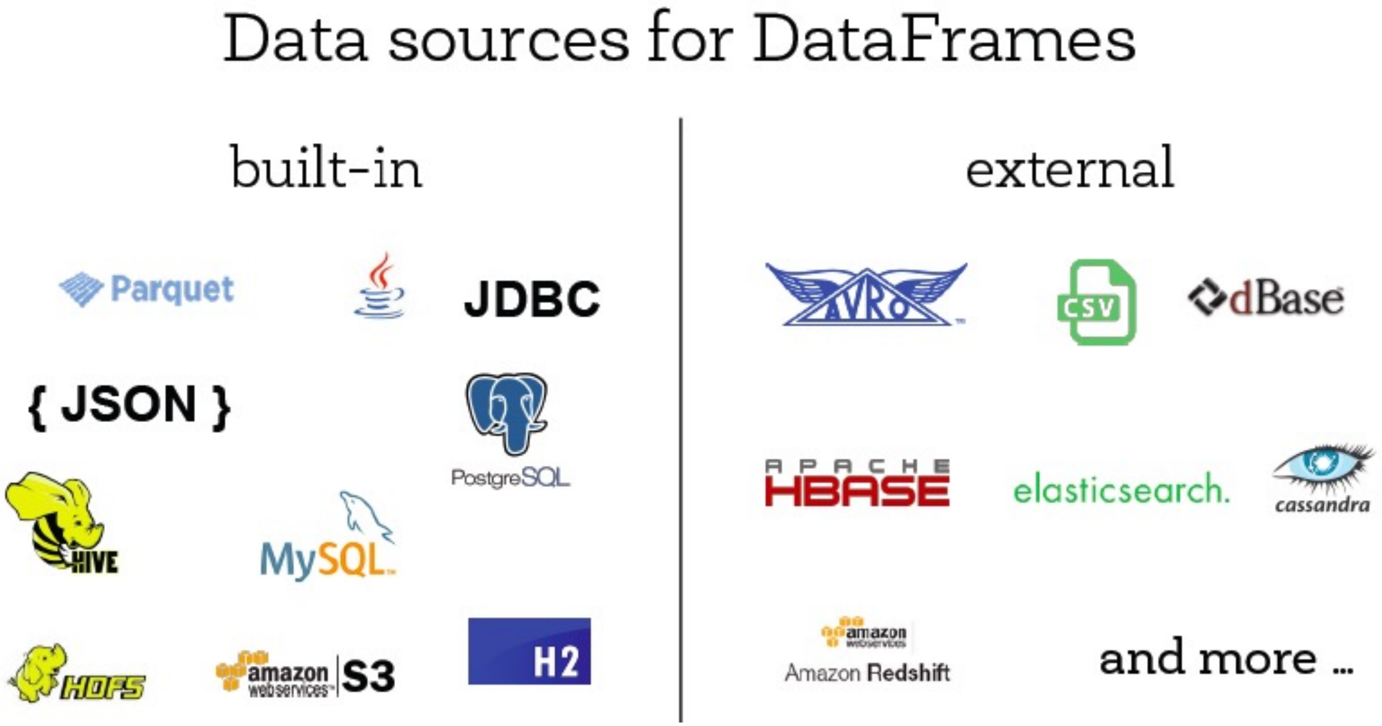


Load data

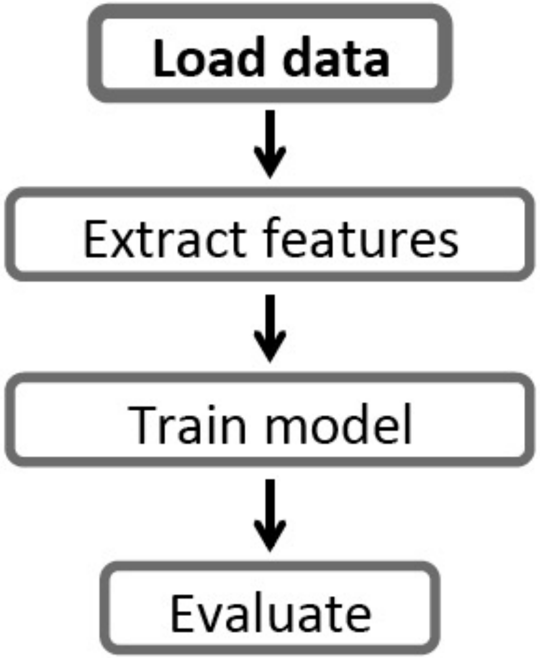
Extract features

Train model

Evaluate



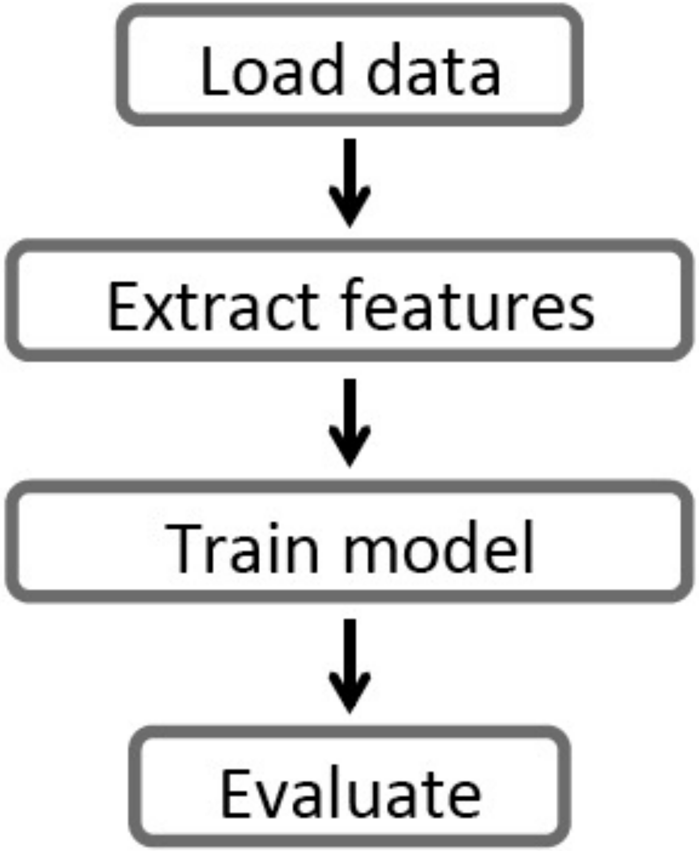
# Load Data



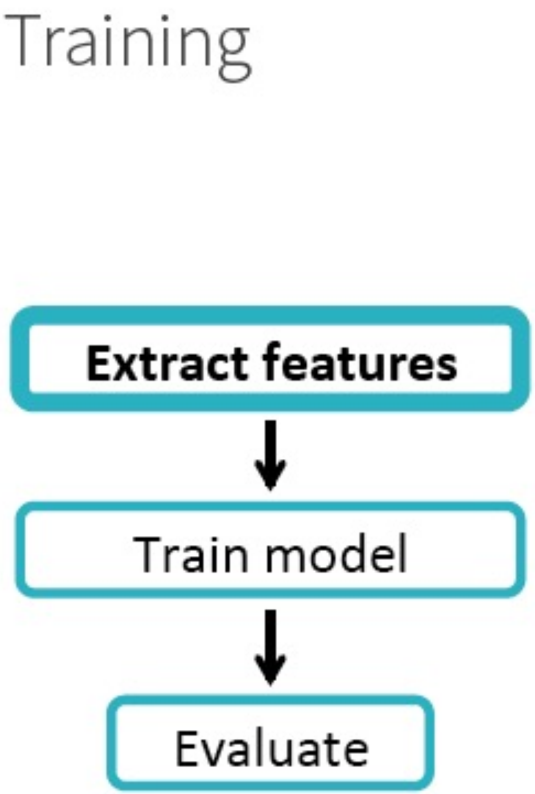
Current data schema

```
label: Int
text: String
```

# Training Workflow



# Abstraction: Transformer



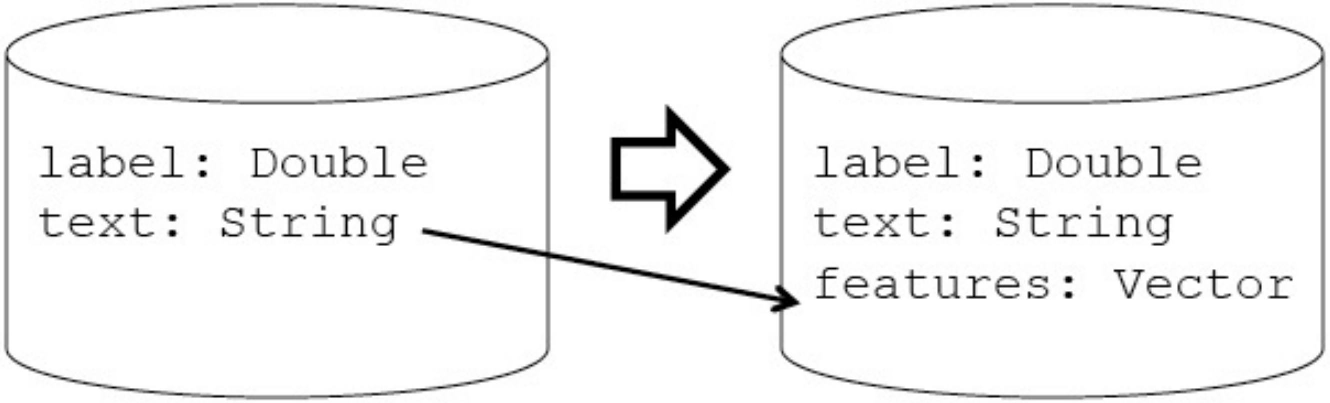
Training

Extract features

Train model

Evaluate

```
def transform(DataFrame): DataFrame
```

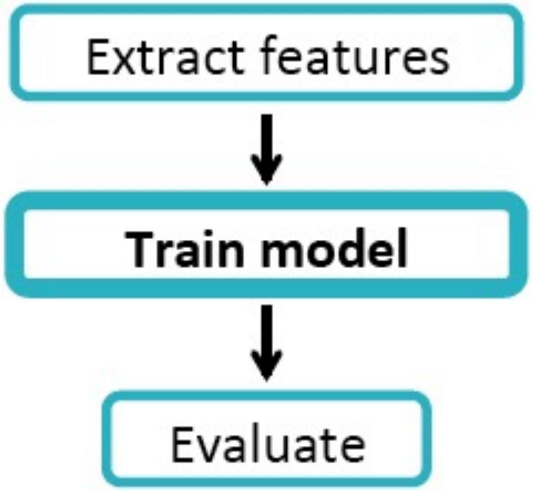


label: Double
text: String

label: Double
text: String
features: Vector

# Abstraction: Estimator

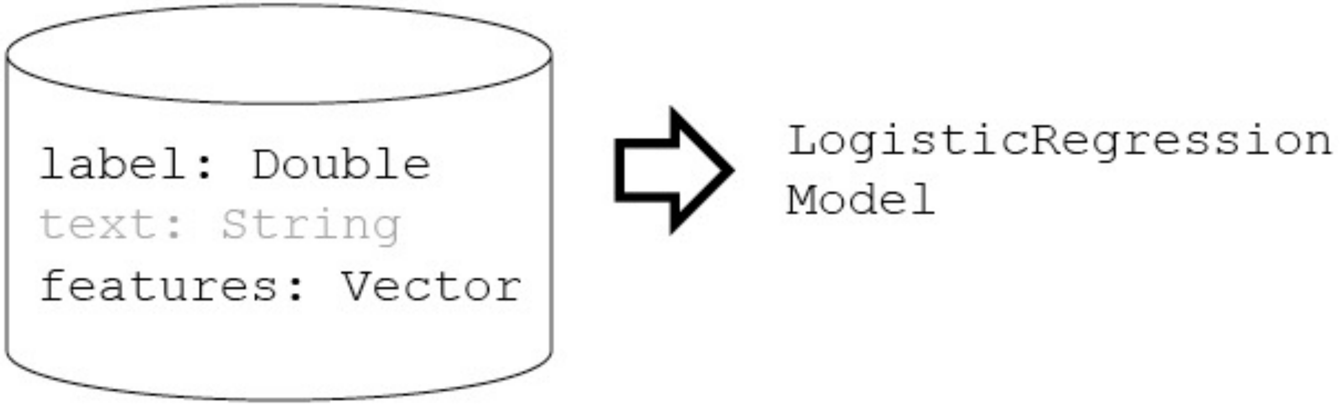
Training


Extract features

Train model

Evaluate

```
def fit(DataFrame): Model
```

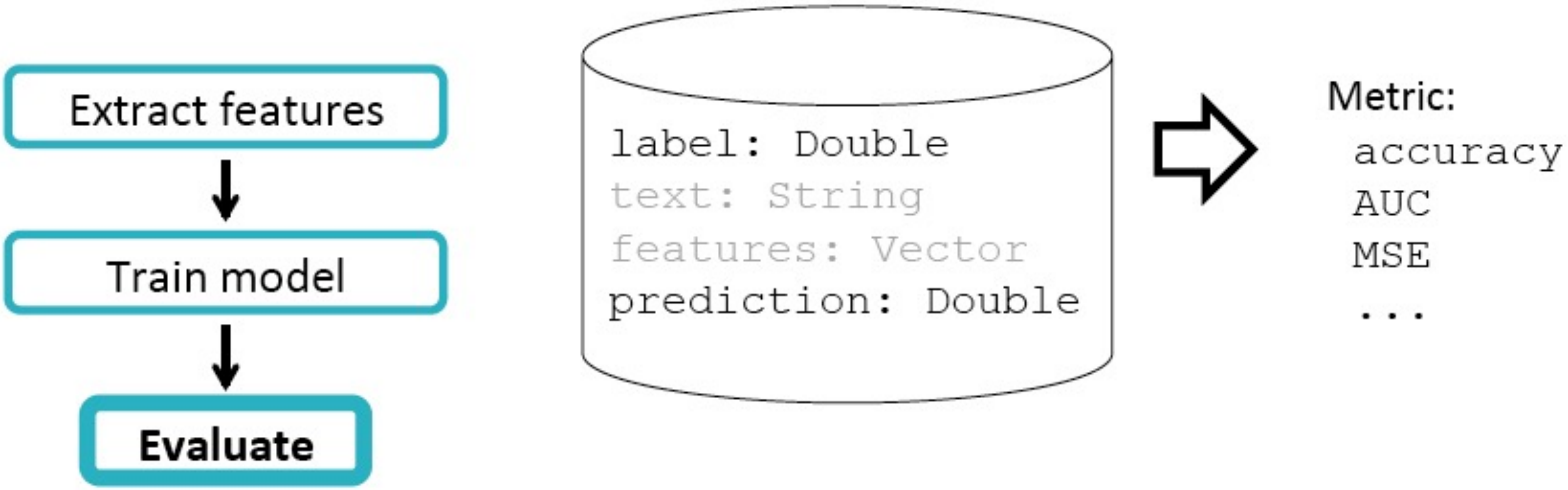

```
label: Double
text: String
features: Vector
```

LogisticRegression
Model

# Abstraction: Evaluator

Training

```
def evaluate(DataFrame): Double
```



Extract features

Train model

**Evaluate**

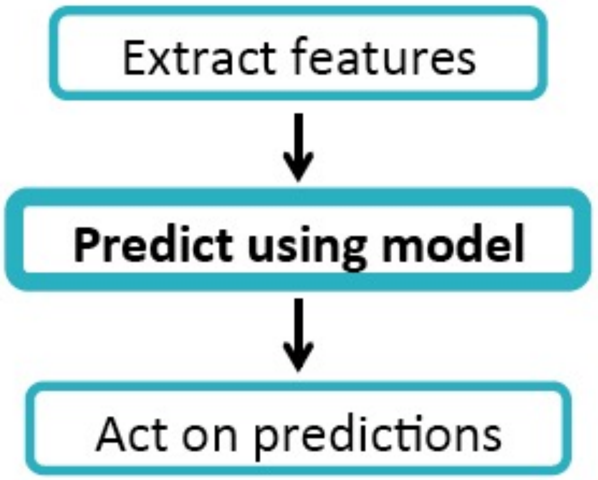
```
label: Double
text: String
features: Vector
prediction: Double
```

Metric:

```
accuracy
AUC
MSE
...
```

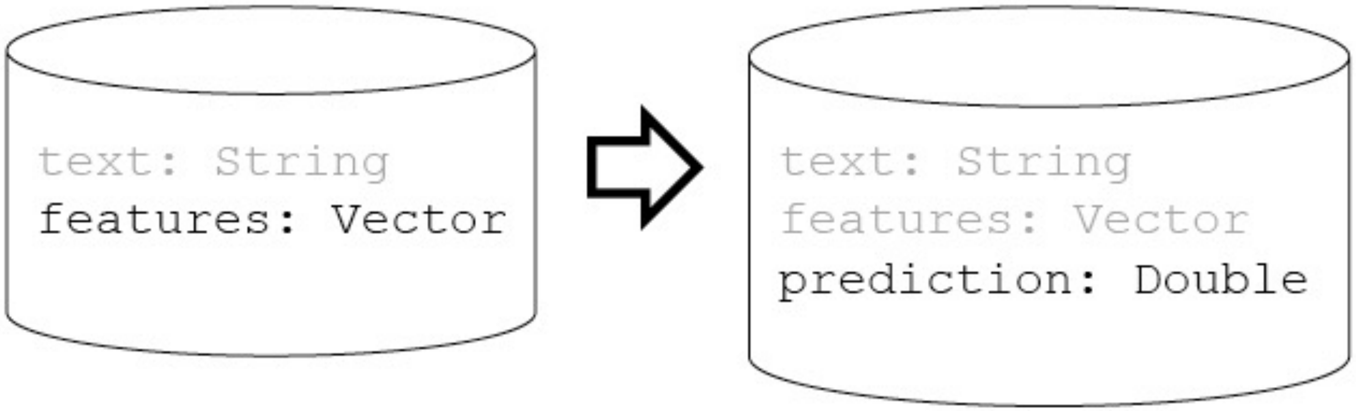
# Abstraction: Model

Testing/Production

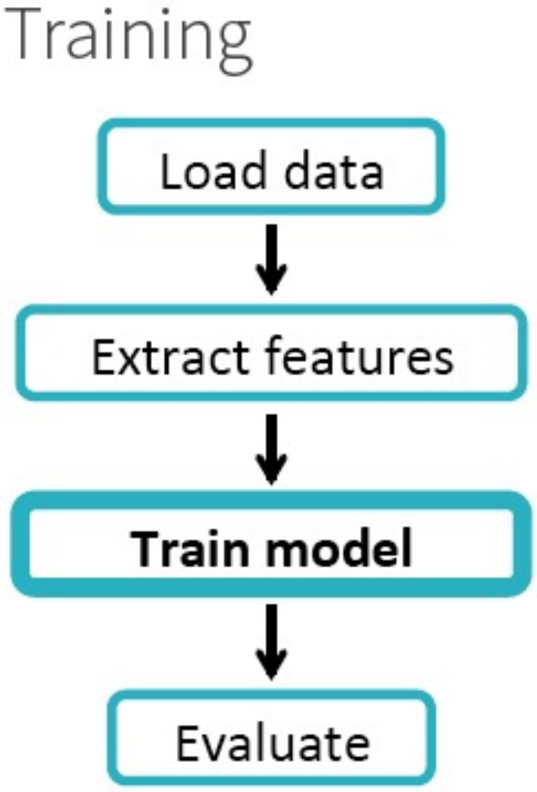


**Model is a type of Transformer**

```
def transform(DataFrame): DataFrame
```



# (Recall) Abstraction: Estimator



Training

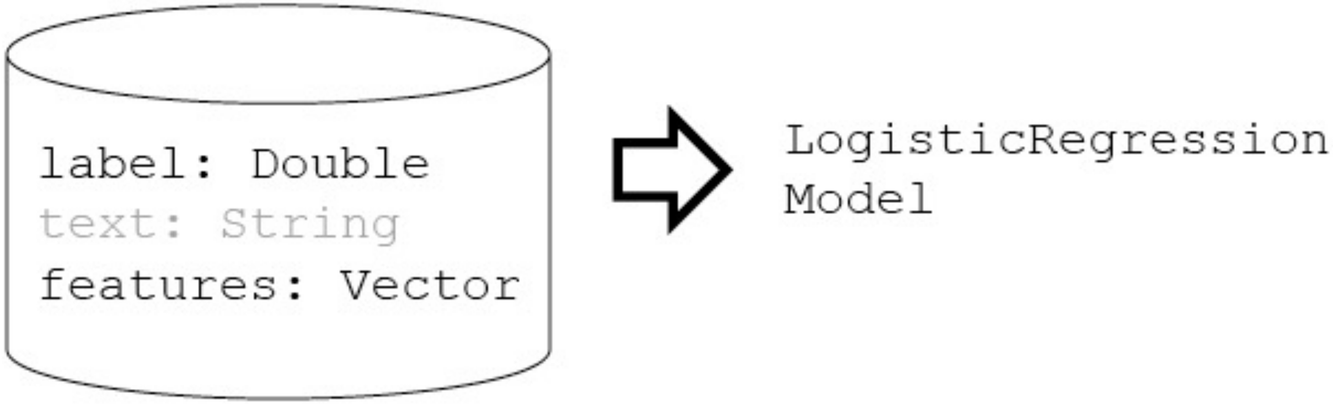
Load data

Extract features

**Train model**

Evaluate

```
def fit(DataFrame): Model
```

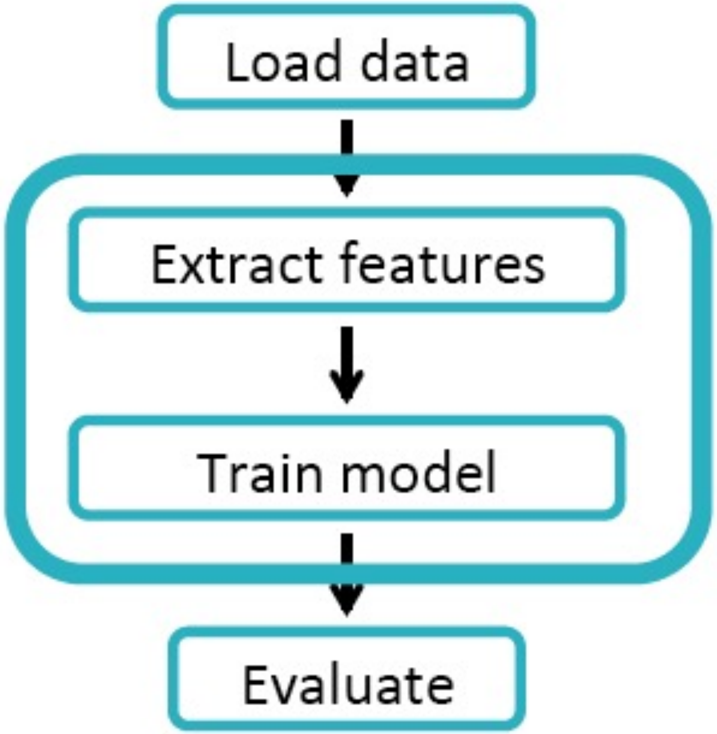


```
label: Double
text: String
features: Vector
```

LogisticRegression Model

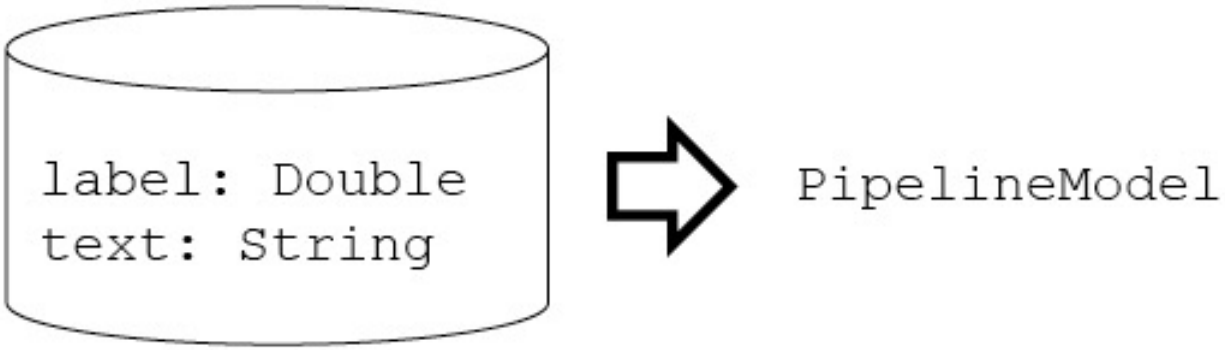
# Abstraction: Pipeline

Training



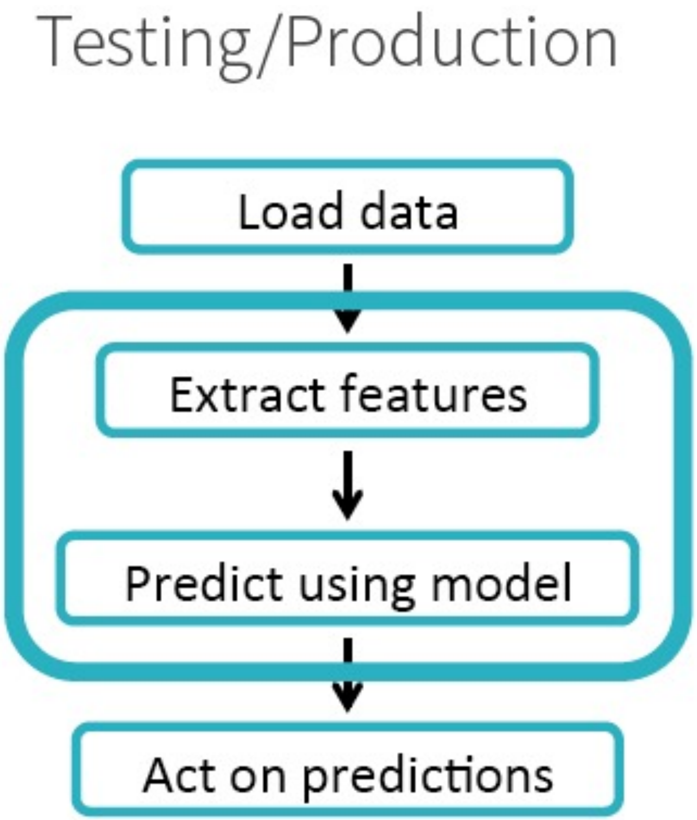
**Pipeline is a type of Estimator**

```
def fit(DataFrame): Model
```



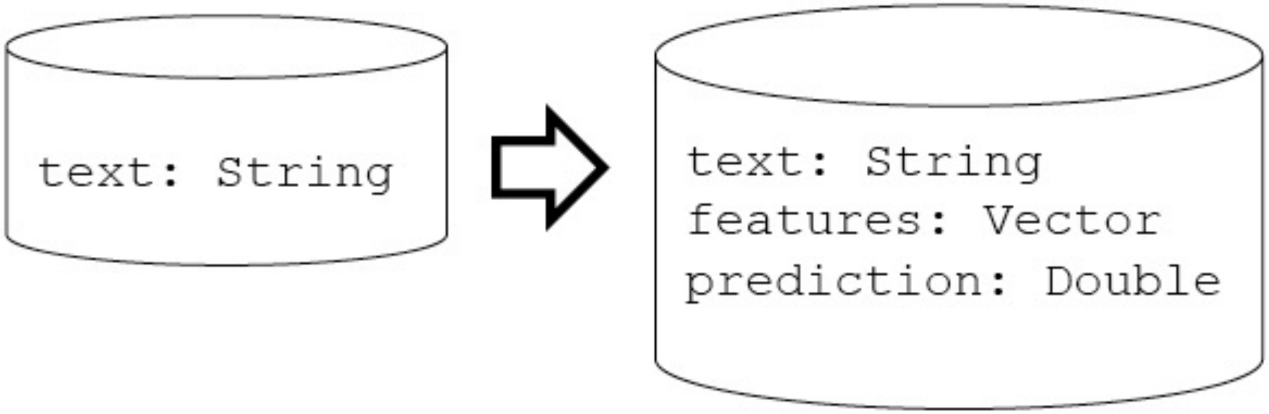
```
label: Double
text: String
```

# Abstraction: PipelineModel

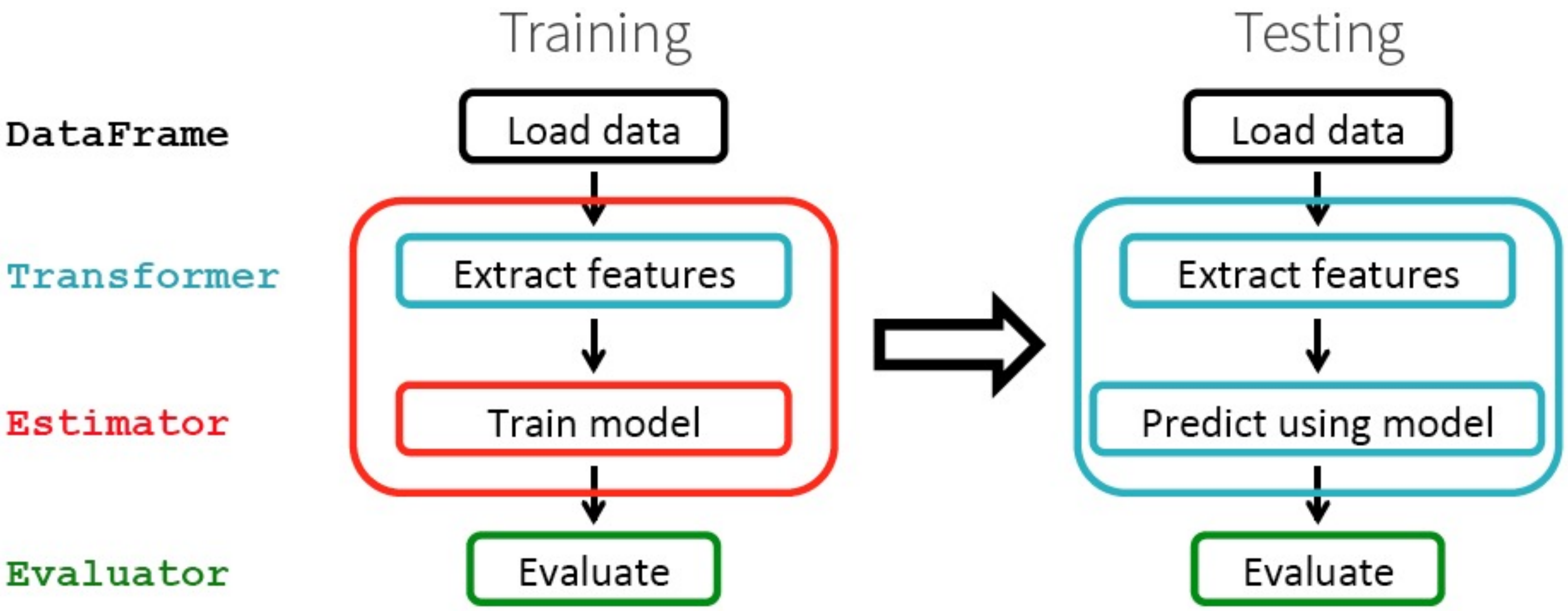


**PipelineModel is a type of Transformer**

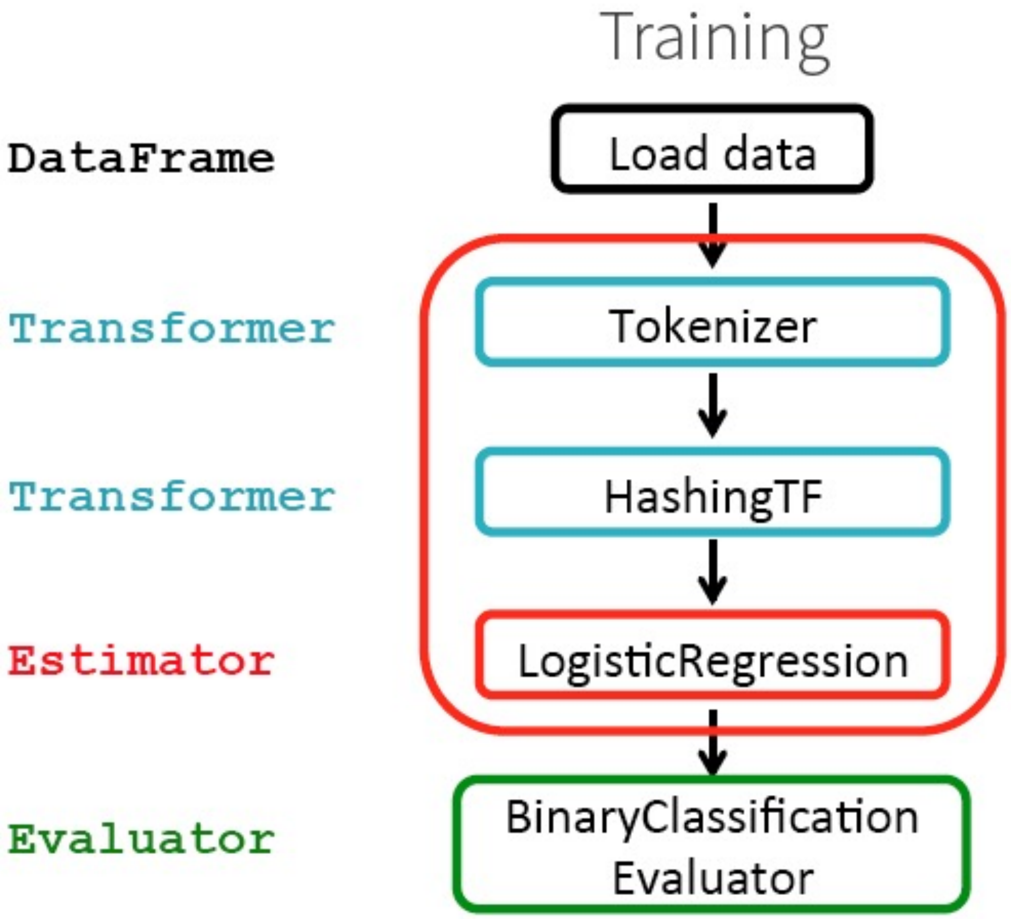
```
def transform(DataFrame): DataFrame
```



# Summary of Abstractions



# A Pipeline Example



Training

DataFrame — Load data

Transformer — Tokenizer

Transformer — HashingTF

Estimator — LogisticRegression

Evaluator — BinaryClassification Evaluator

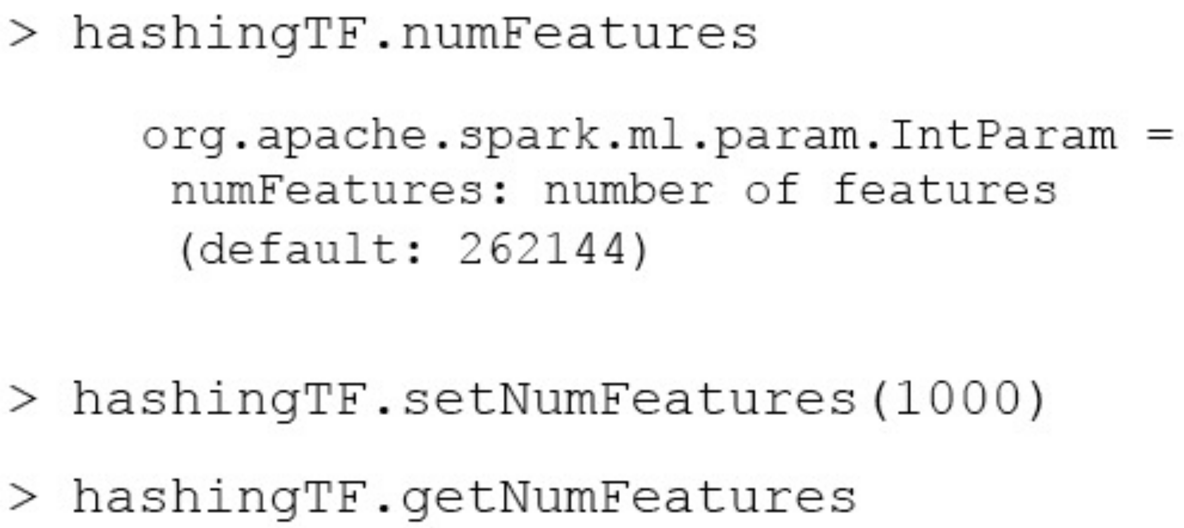
Current data schema

```
label: Double
text: String

words: Seq[String]

features: Vector

prediction: Double
```

# Parameters

Standard API

- Typed
- Defaults
- Built-in doc
- Autocomplete



```
> hashingTF.numFeatures

    org.apache.spark.ml.param.IntParam =
     numFeatures: number of features
     (default: 262144)


> hashingTF.setNumFeatures(1000)

> hashingTF.getNumFeatures
```

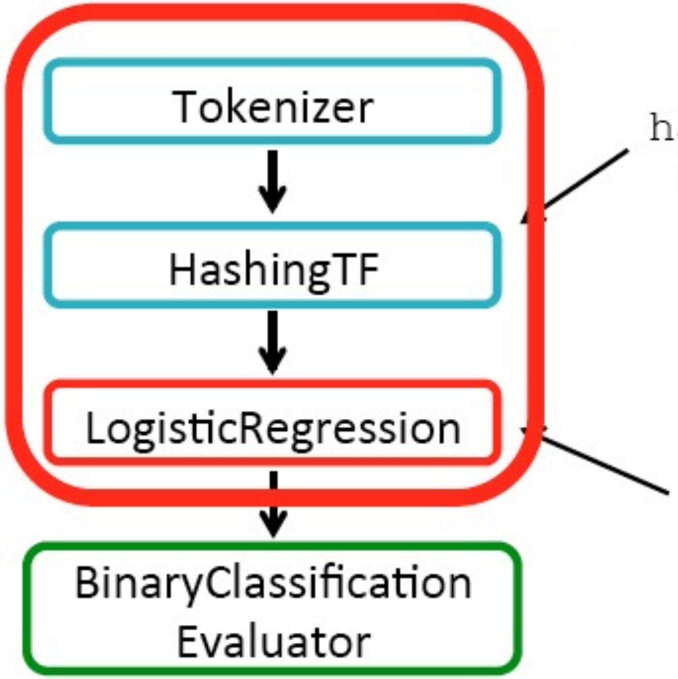
# Parameter Tuning

Given:

- Estimator
- Parameter grid
- Evaluator

Find best parameters

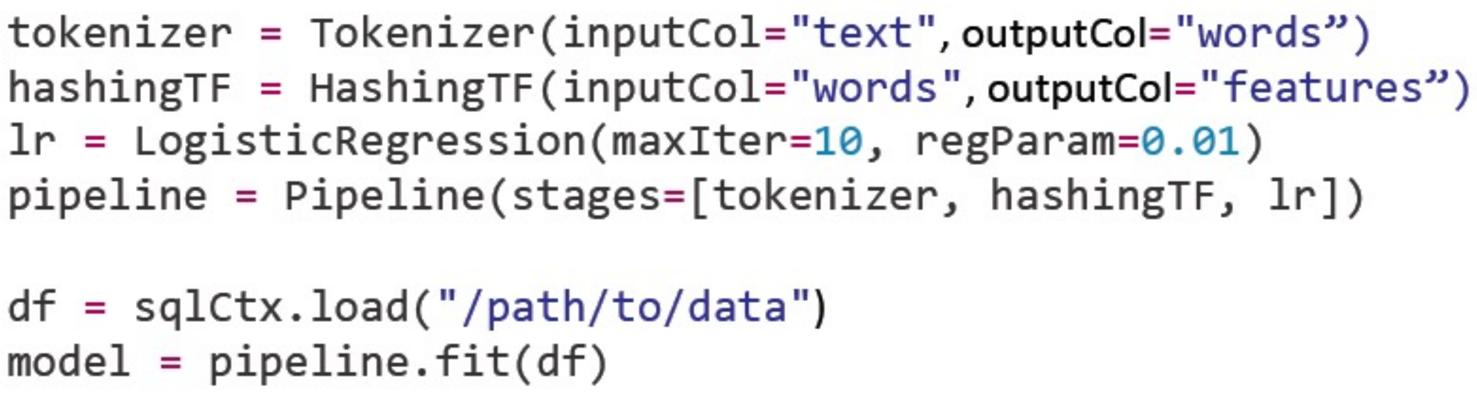
**CrossValidator**



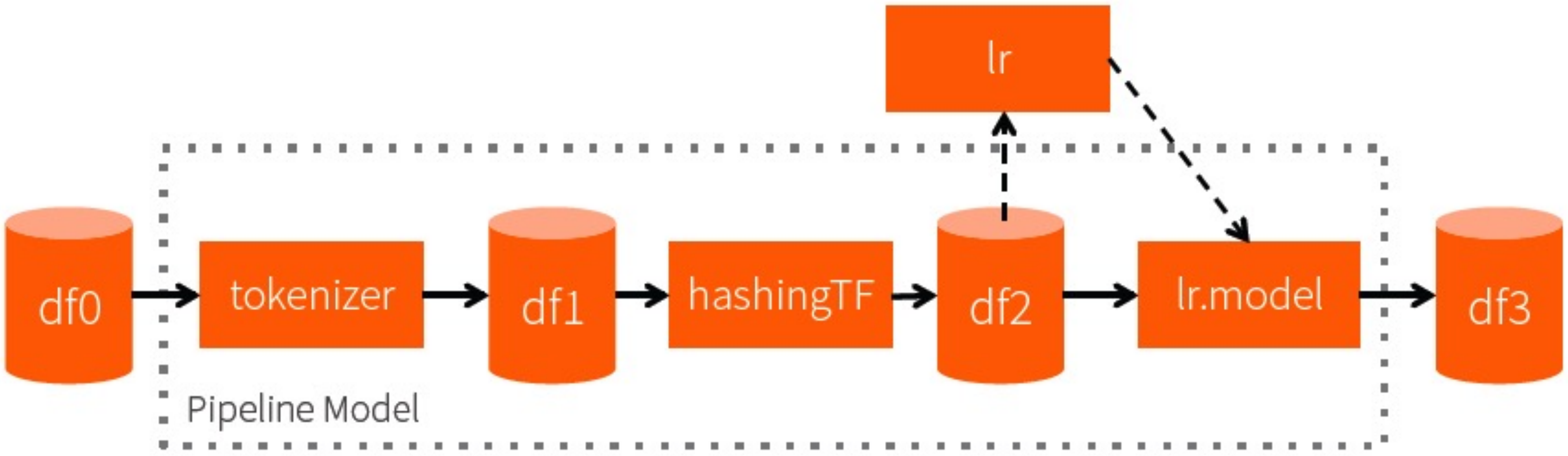
```
hashingTF.numFeatures
 {100, 1000, 10000}
```

```
lr.regParam
 {0.01, 0.1, 0.5}
```

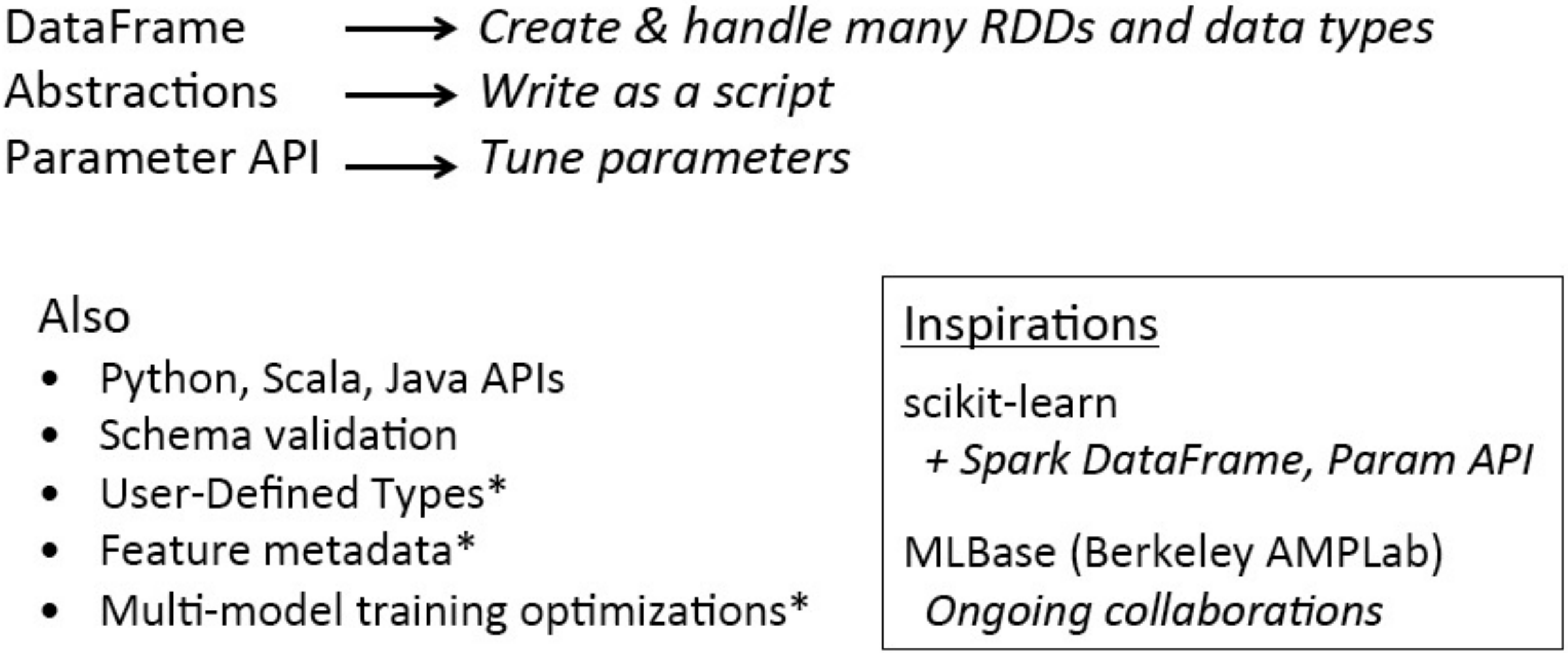
# Sample Code for an ML Pipeline



```
tokenizer = Tokenizer(inputCol="text",outputCol="words”)
hashingTF = HashingTF(inputCol="words",outputCol="features”)
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

df = sqlCtx.load("/path/to/data")
model = pipeline.fit(df)
```



# Summary of Spark ML Pipelines



DataFrame → *Create & handle many RDDs and data types*

Abstractions → *Write as a script*

Parameter API → *Tune parameters*

Also

- Python, Scala, Java APIs
- Schema validation
- User-Defined Types*
- Feature metadata*
- Multi-model training optimizations*

Inspirations

scikit-learn

*+ Spark DataFrame, Param API*

MLBase (Berkeley AMPLab)

*Ongoing collaborations*

- Also Support Models Import and Export via “ML Persistence”

# Enabling Interactive (Big) Data Science with SparkR

Spark early adopters

Users

Understands
MapReduce
& functional APIs

Data Engineers
Data Scientists
Statisticians
R users
PyData …

Fast!

Statistical!

Scalable

+

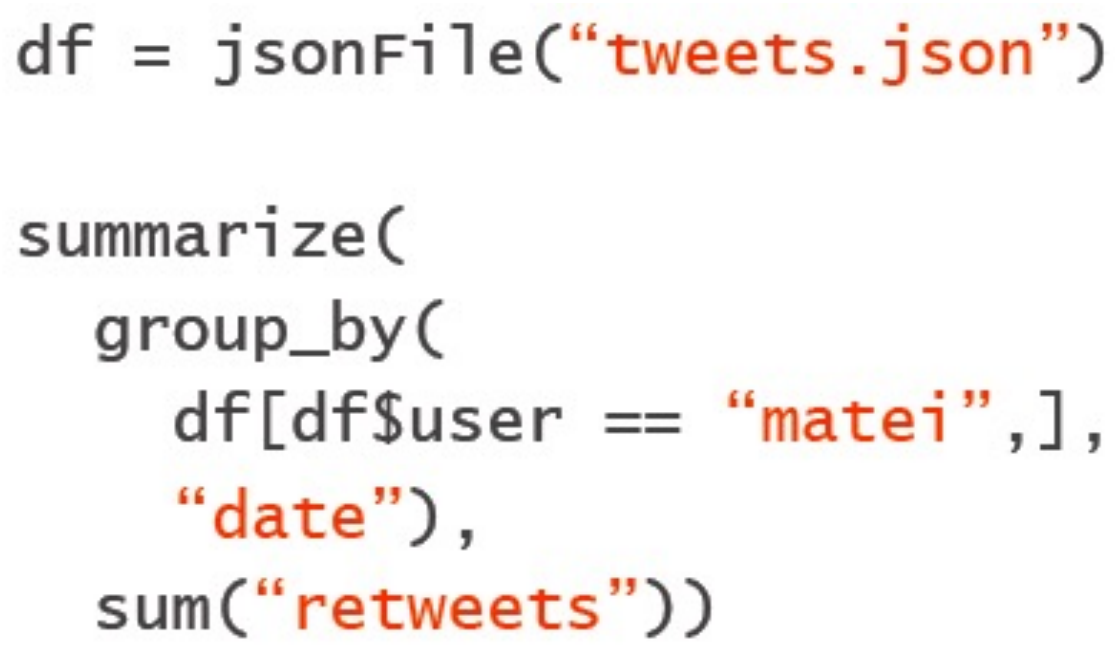
Packages

Flexible

Interactive

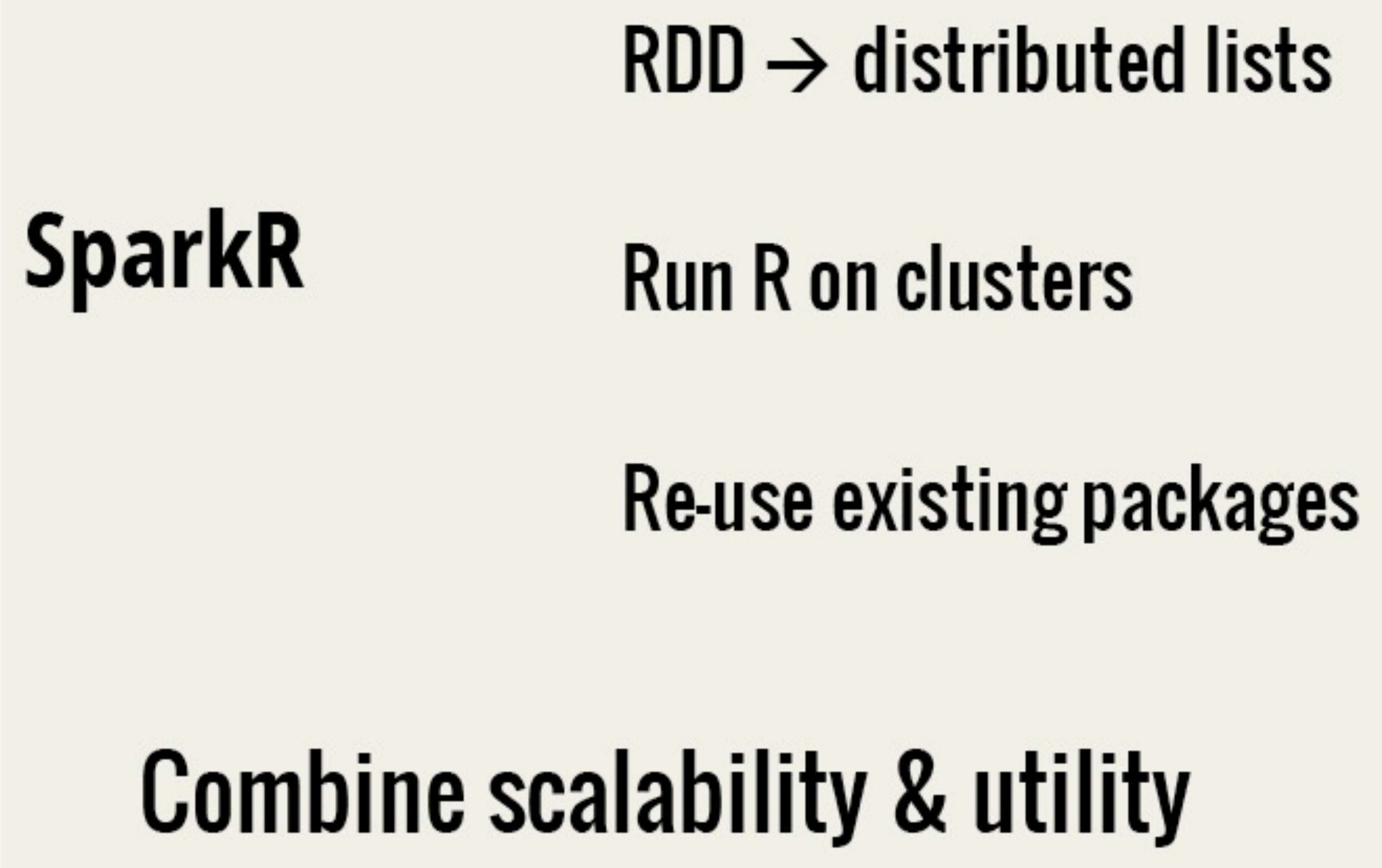
# SparkR – R package for Spark

- R Interface support via SparkR (R with RDD = R2D2) since Spark 1.4 (released since June 2015)
    - Exposes DataFrames and MLlib in R:



```
df = jsonFile("tweets.json")

summarize(
   group_by(
      df[df$user == "matei",],
      "date"),
   sum("retweets"))
```

# SparkR – R package for Spark



SparkR

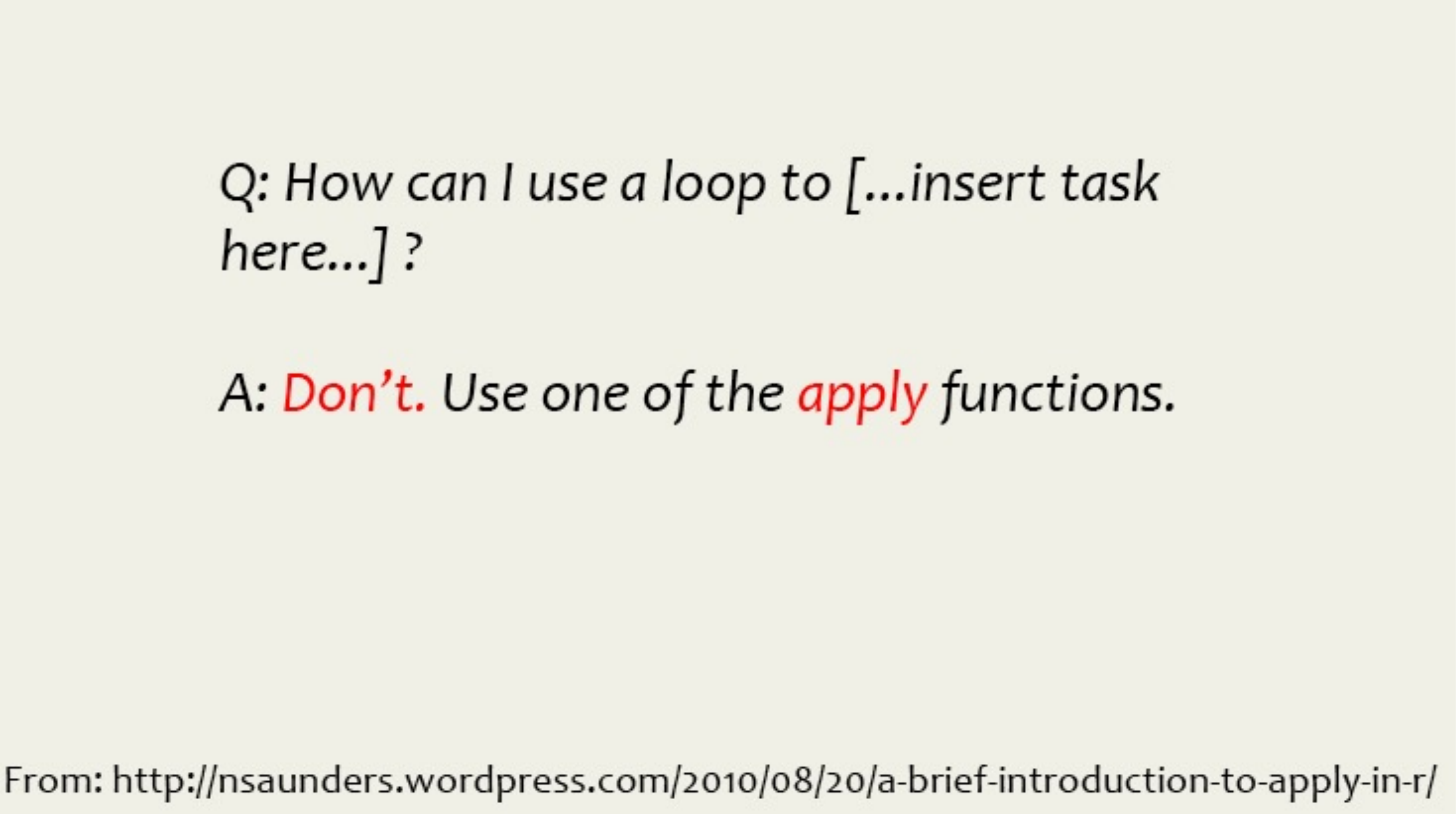
RDD → distributed lists

Run R on clusters

Re-use existing packages

Combine scalability & utility

# Getting closer to Idiomatic R

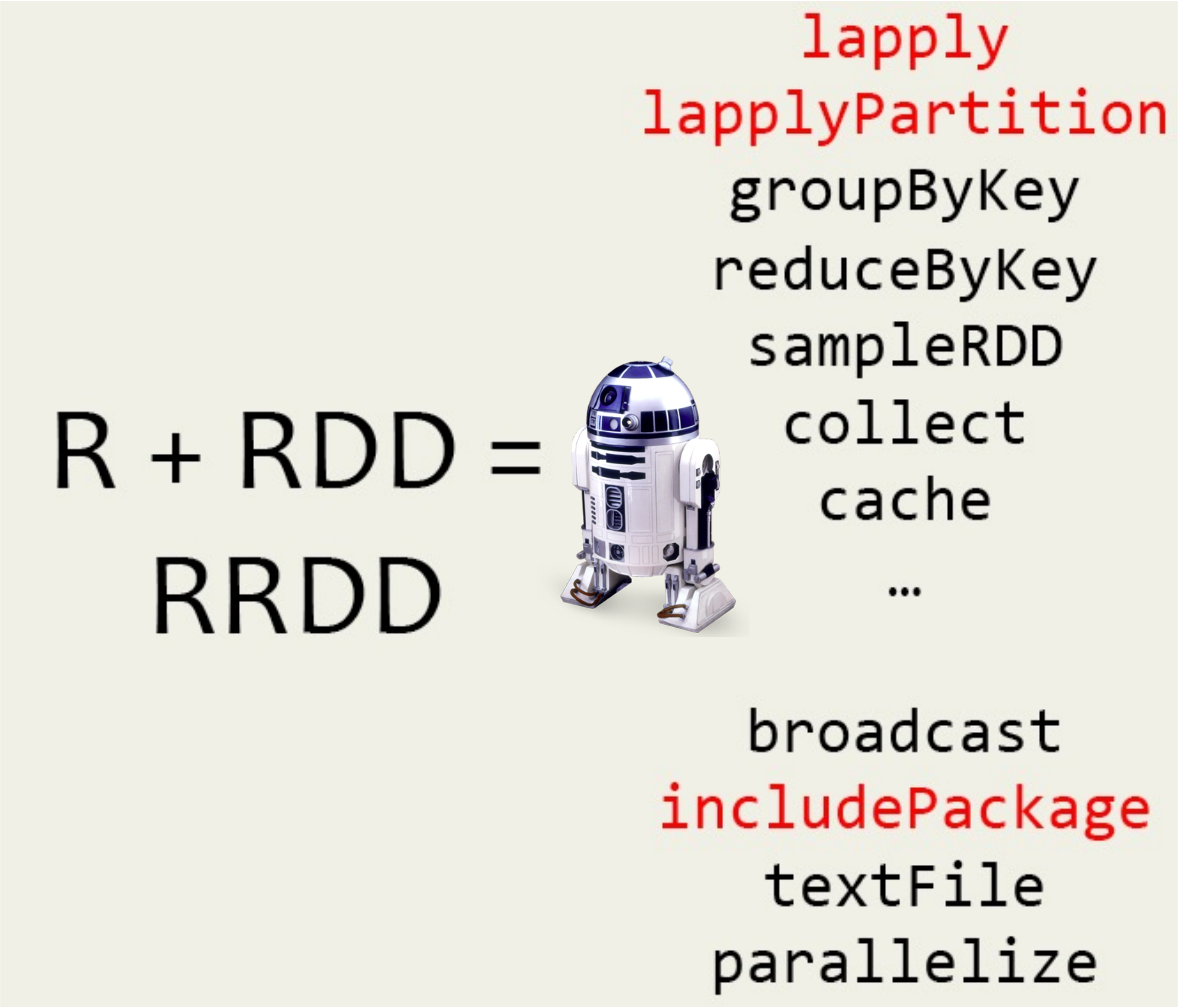


*Q: How can I use a loop to [...insert task here...] ?*

A: Don't. Use one of the *apply* functions.

From: http://nsaunders.wordpress.com/2010/08/20/a-brief-introduction-to-apply-in-r/

# SparkR



R + RDD =

RRDD

lapply

lapplyPartition

groupByKey

reduceByKey

sampleRDD

collect

cache

…

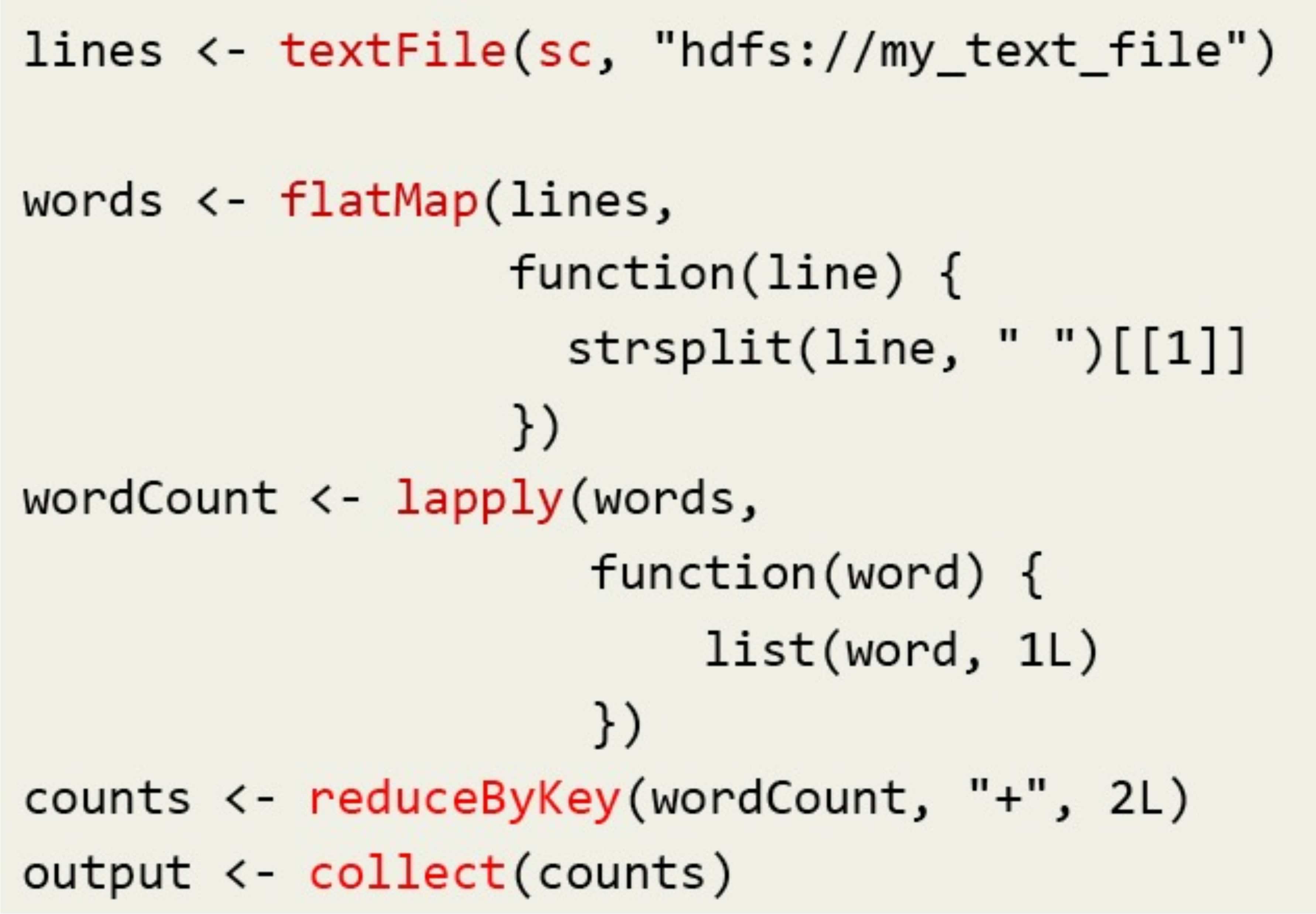
broadcast

includePackage

textFile

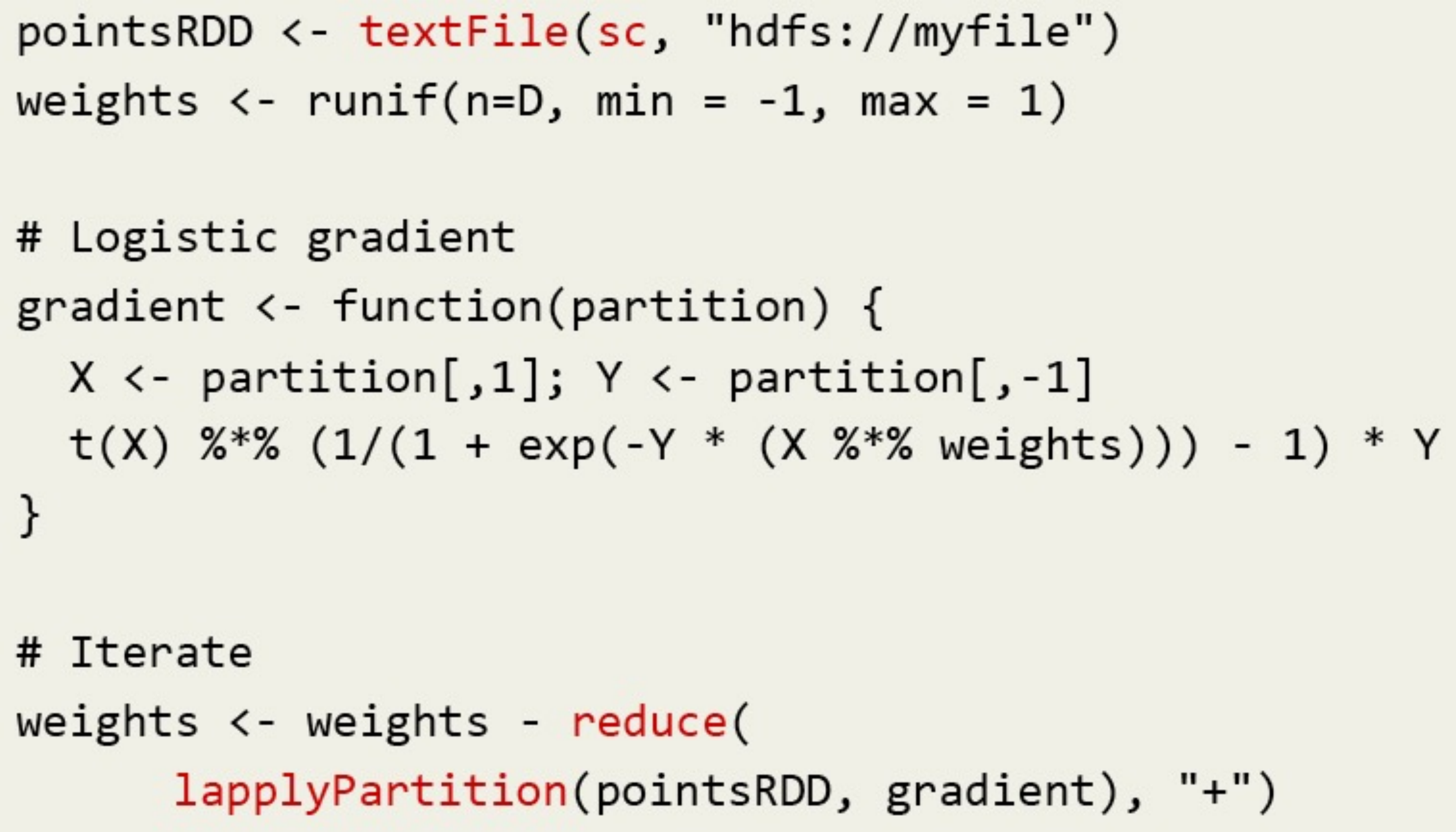
parallelize

# Example: Word Counting with SparkR



```
lines <- textFile(sc, "hdfs://my_text_file")

words <- flatMap(lines,
                 function(line) {
                   strsplit(line, " ")[[1]]
                 })
wordCount <- lapply(words,
                    function(word) {
                        list(word, 1L)
                    })
counts <- reduceByKey(wordCount, "+", 2L)
output <- collect(counts)
```

# Example: Logistic Regression with SparkR



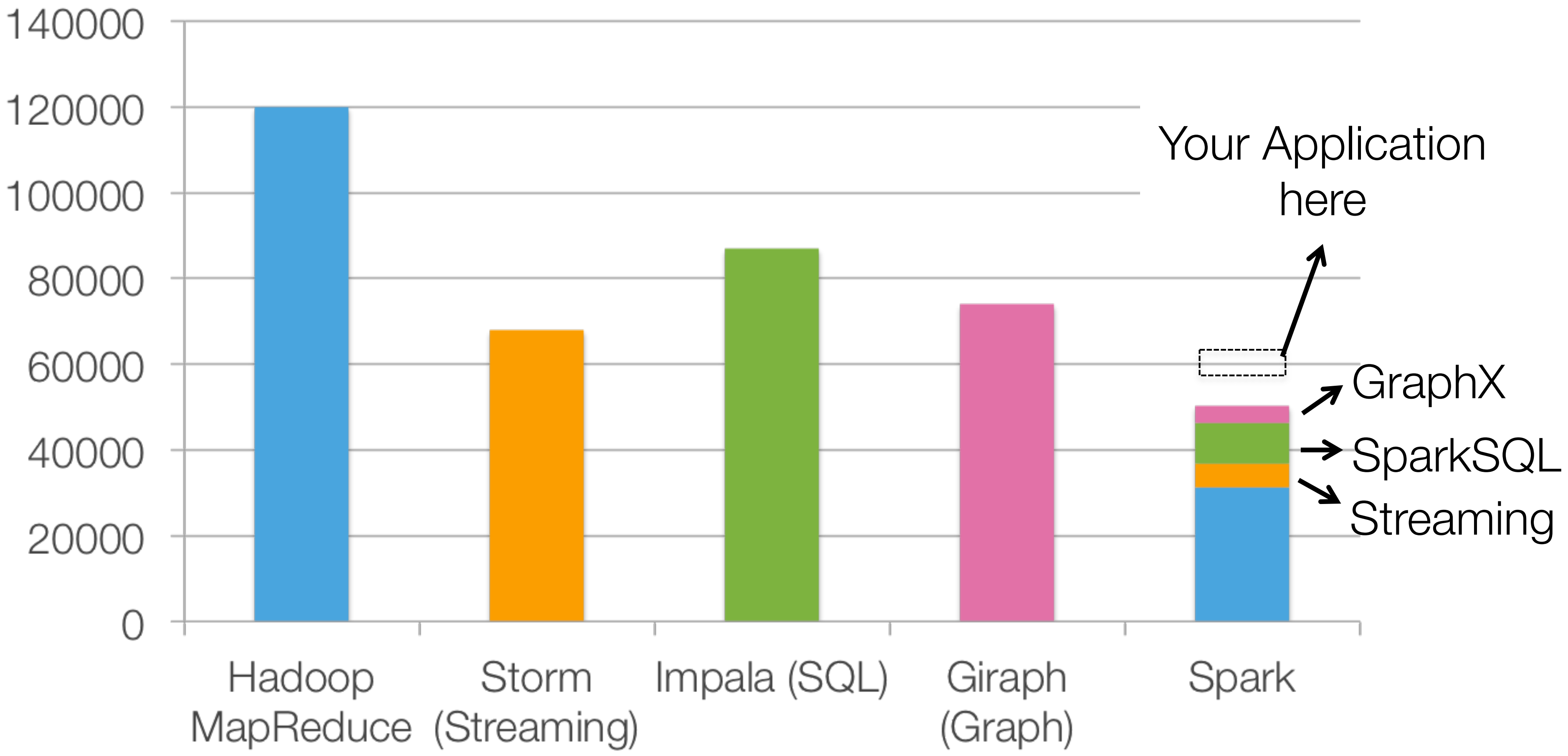
```
pointsRDD <- textFile(sc, "hdfs://myfile")
weights <- runif(n=D, min = -1, max = 1)

# Logistic gradient
gradient <- function(partition) {
  X <- partition[,1]; Y <- partition[,-1]
  t(X) %*% (1/(1 + exp(-Y * (X %*% weights))) - 1) * Y
}

# Iterate
weights <- weights - reduce(
      lapplyPartition(pointsRDD, gradient), "+")
```

# SparkR Implementation

- Very similar to PySpark
- Relatively easy to extend Spark
  - 329 lines of Scala code
  - 2079 lines of R code
  - 693 lines of Test code in R

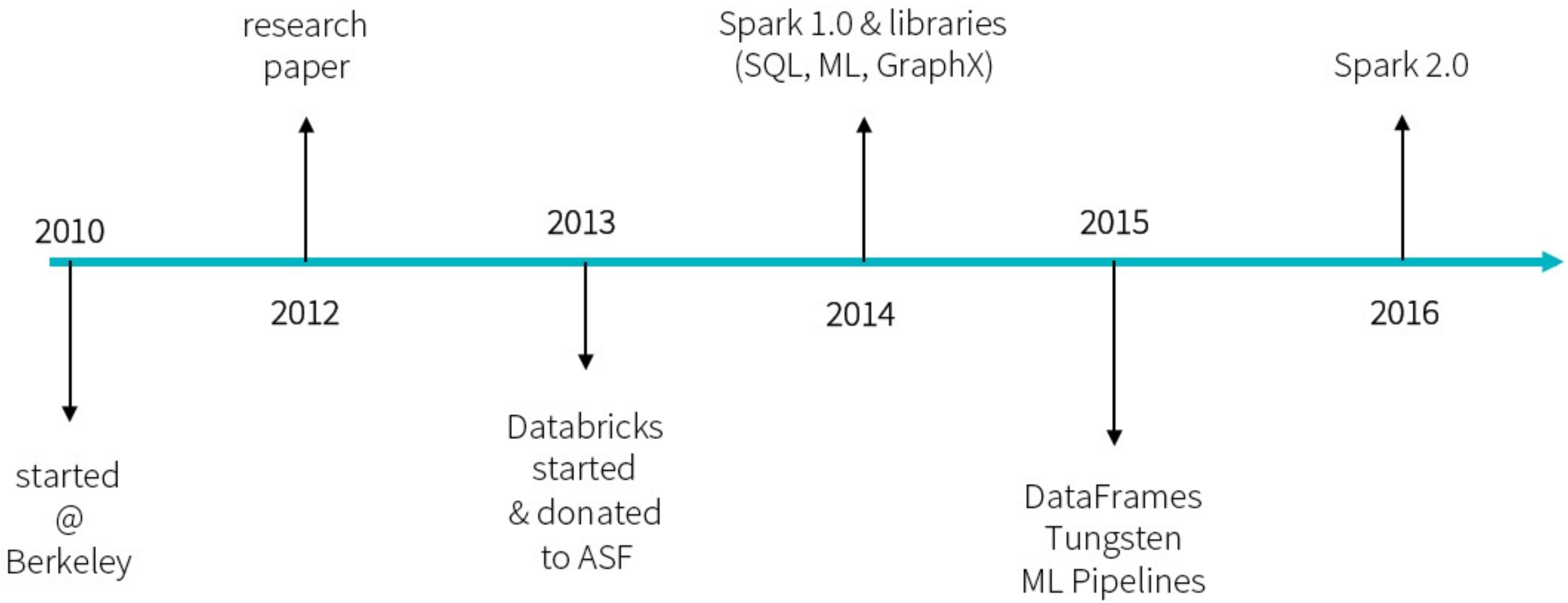
# Spark: A Recap and Future Directions

# Powerful Stack – Agile Development

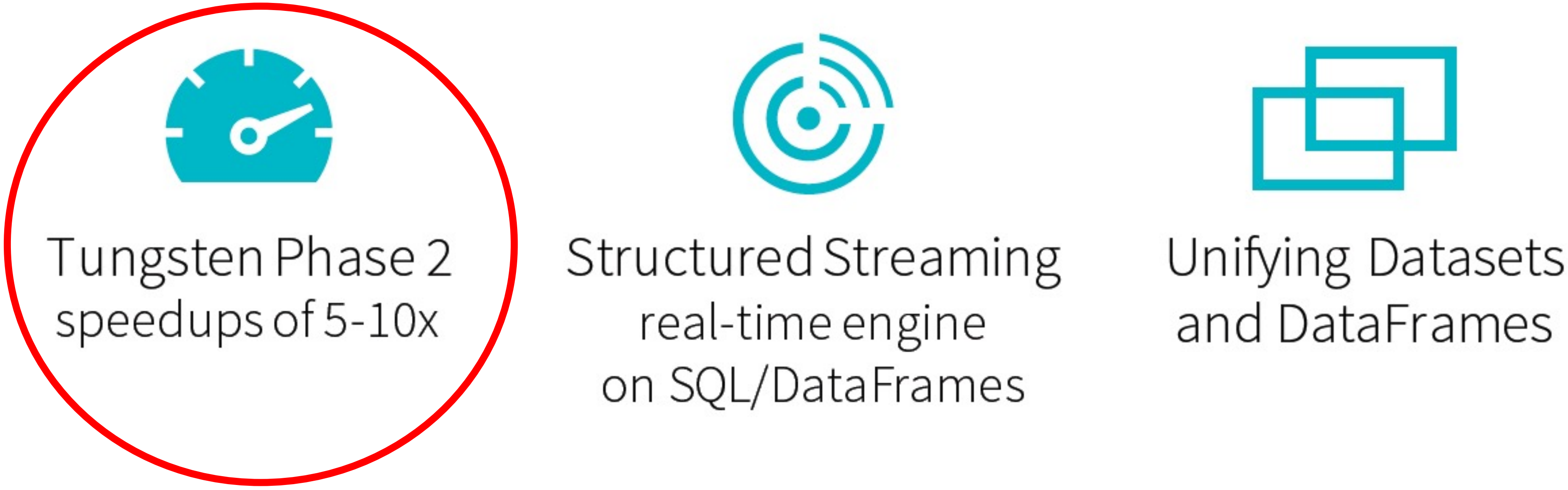


non-test, non-example source lines

# Spark/ BDAS Timeline till v2.0


research paper
Spark 1.0 & libraries (SQL, ML, GraphX)
Spark 2.0
2010
2013
2015
2012
2014
2016
started @ Berkeley
Databricks started & donated to ASF
DataFrames Tungsten ML Pipelines

# Major Features in Spark 2.0



Tungsten Phase 2
speedups of 5-10x

Structured Streaming
real-time engine
on SQL/DataFrames

Unifying Datasets
and DataFrames

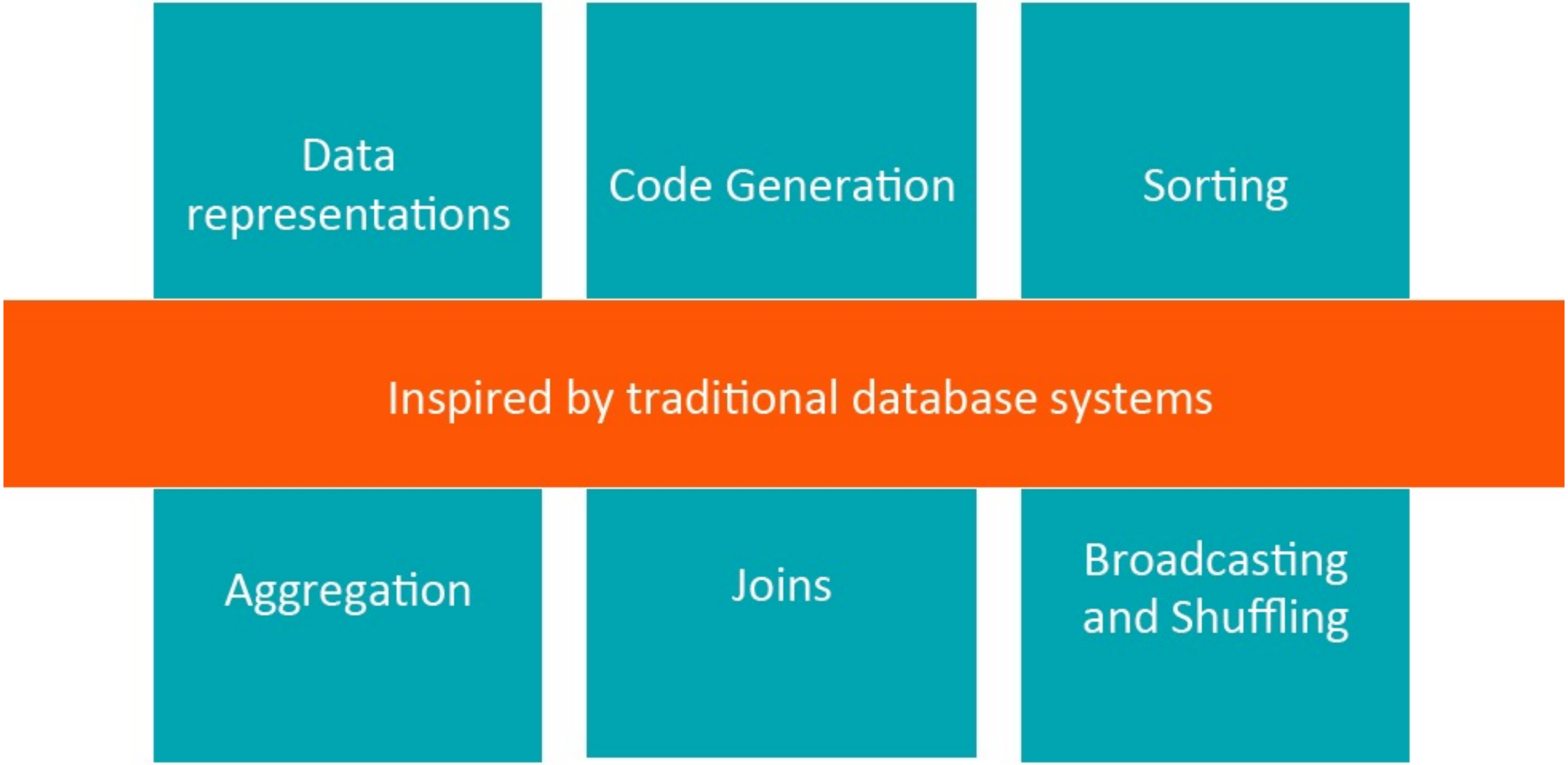
# Boosting Spark Performance via Project Tungsten

**Goal**

To Overcome JVM Performance limitations and bring Spark performance closer to Bare Metal via:

- **Native Memory Management and Binary Processing**: leveraging application semantics to manage memory explicitly and eliminate the overhead of JVM object model and garbage collection
- **Cache-aware computation**: algorithms and data structures to exploit memory hierarchy
- **Runtime Code generation**: using code generation to exploit modern compilers and CPUs

# Project Tungsten: Key areas of Optimization



Data representations

Code Generation

Sorting

Inspired by traditional database systems

Aggregation

Joins

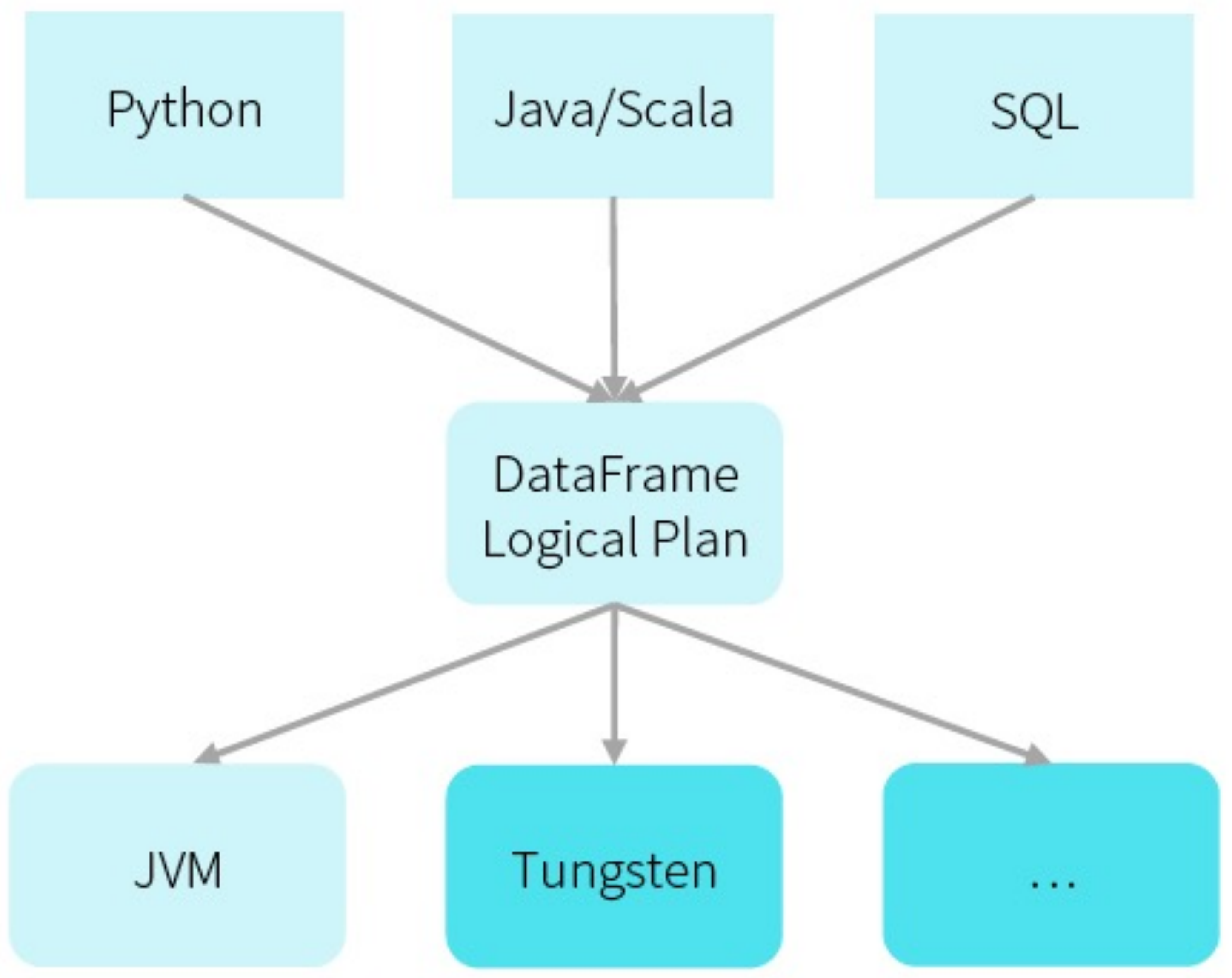
Broadcasting and Shuffling

# Optimized Data Representations

- Java Objects have two downsides:
  - Space overheads
  - Garbage collection overheads

- Tungsten sidesteps these problems by performing its own manual memory management

# Further Performance Optimization via Project Tungsten



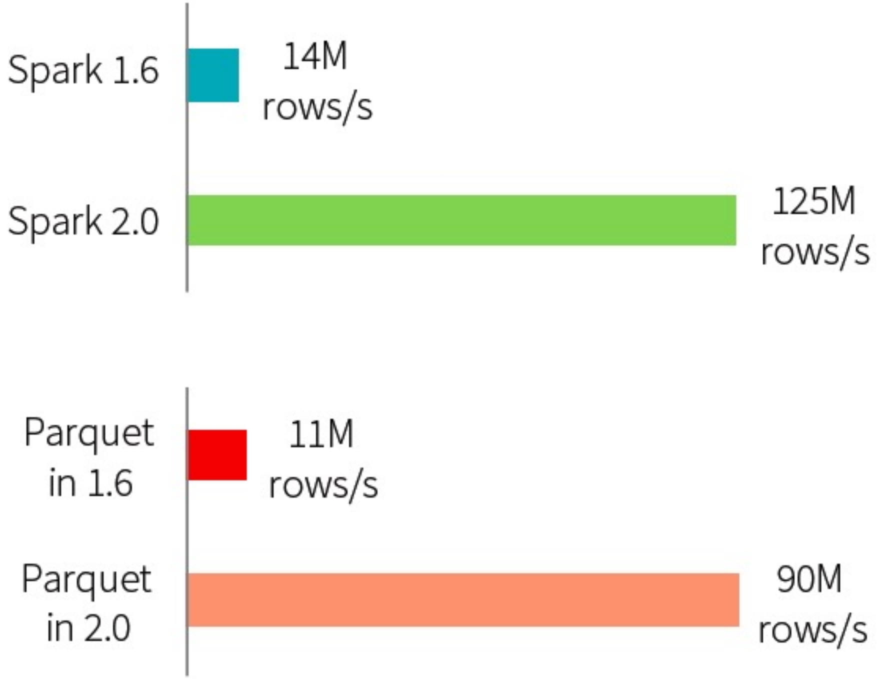
# Phased introduction of Tungsten

In Spark 1.4-1.6

- Added Binary Storage and Basic Code Generation
- DataFrame + Dataset APIs enable Tungsten in User Programs
- Tungsten also being used under SparkSQL + parts of MLlib

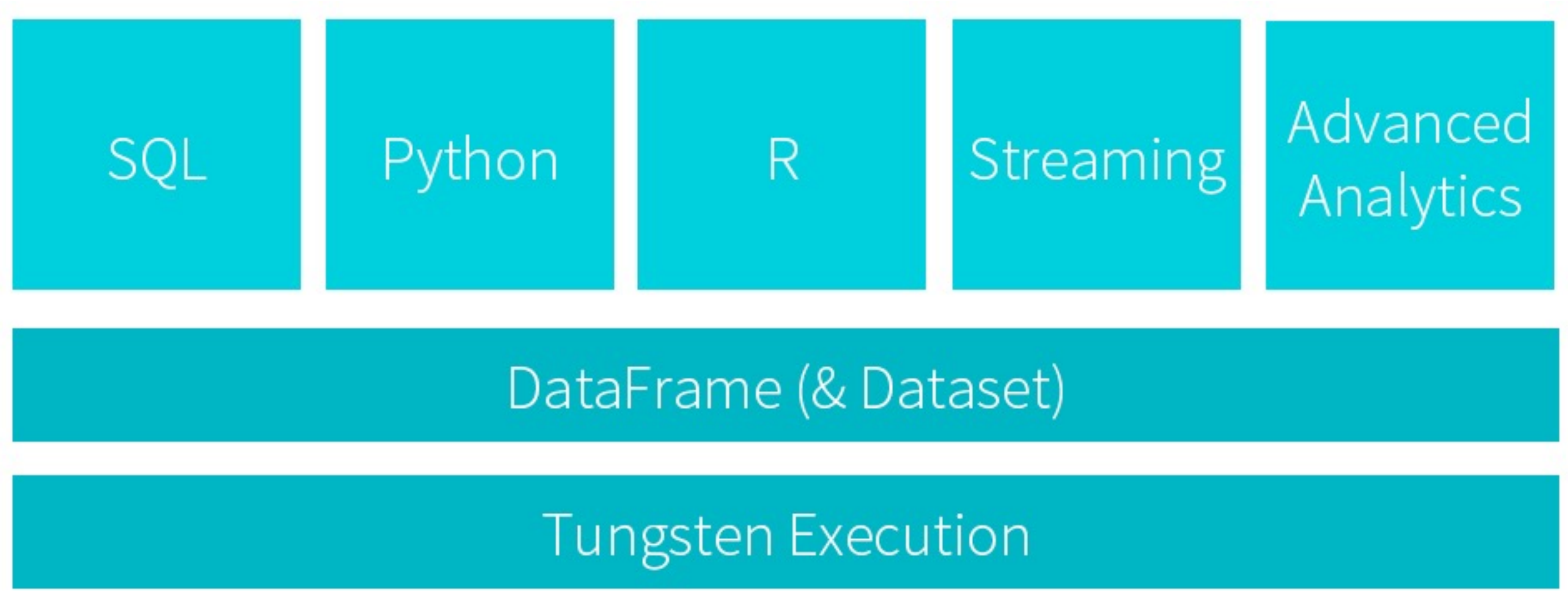
By Spark 2.0

- Whole-stage Code Generation
  - Remove expensive Iterator calls
  - Fuse across multiple operators
- Vector Processing
- Optimized Input/Output
  - Parquest + Built-in Cache



Automatically applies to SQL, DataFrames, Datasets

# Spark Ver. 2.0 Stack (circa 2015) DataFrame + Tungsten



SQL

Python

R

Streaming

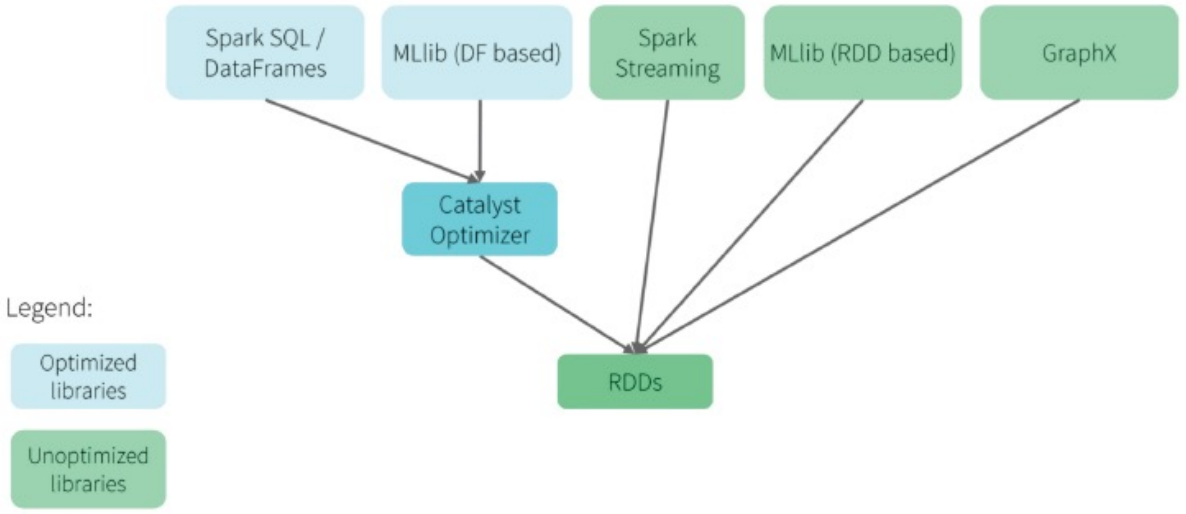
Advanced Analytics

DataFrame (& Dataset)

Tungsten Execution

# Spark 1.6 vs. Spark 2.x

Spark 1.6.x



Spark SQL / DataFrames

MLlib (DF based)

Spark Streaming

MLlib (RDD based)

GraphX

Catalyst Optimizer

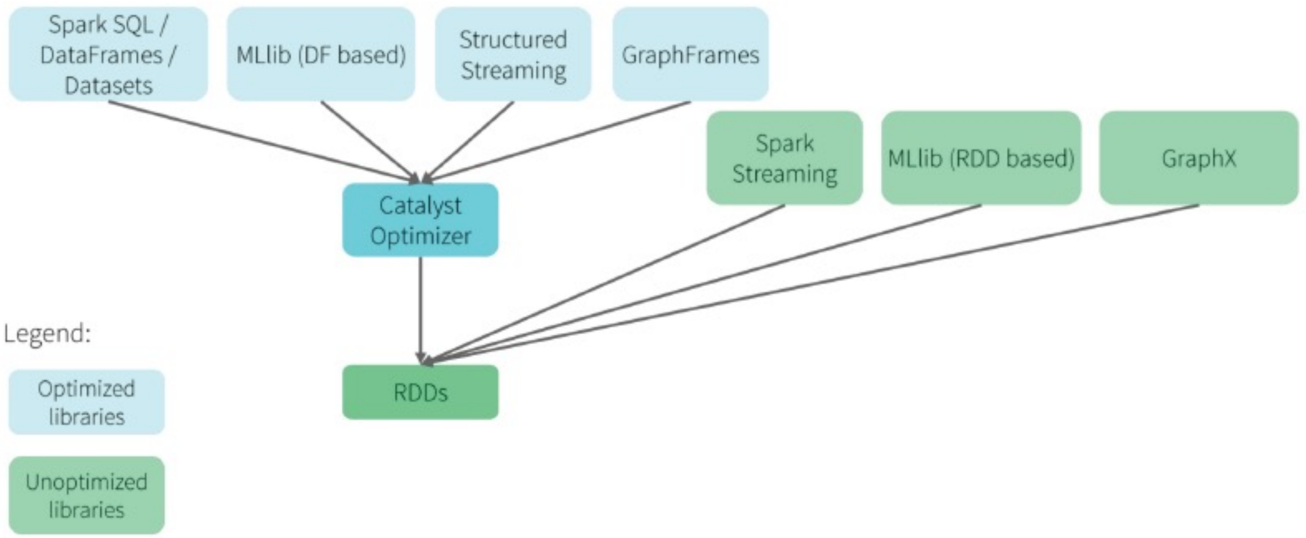
RDDs

Legend:

Optimized libraries

Unoptimized libraries

Spark 2.x.x



Spark SQL / DataFrames / Datasets

MLlib (DF based)

Structured Streaming

GraphFrames

Spark Streaming

MLlib (RDD based)

GraphX

Catalyst Optimizer

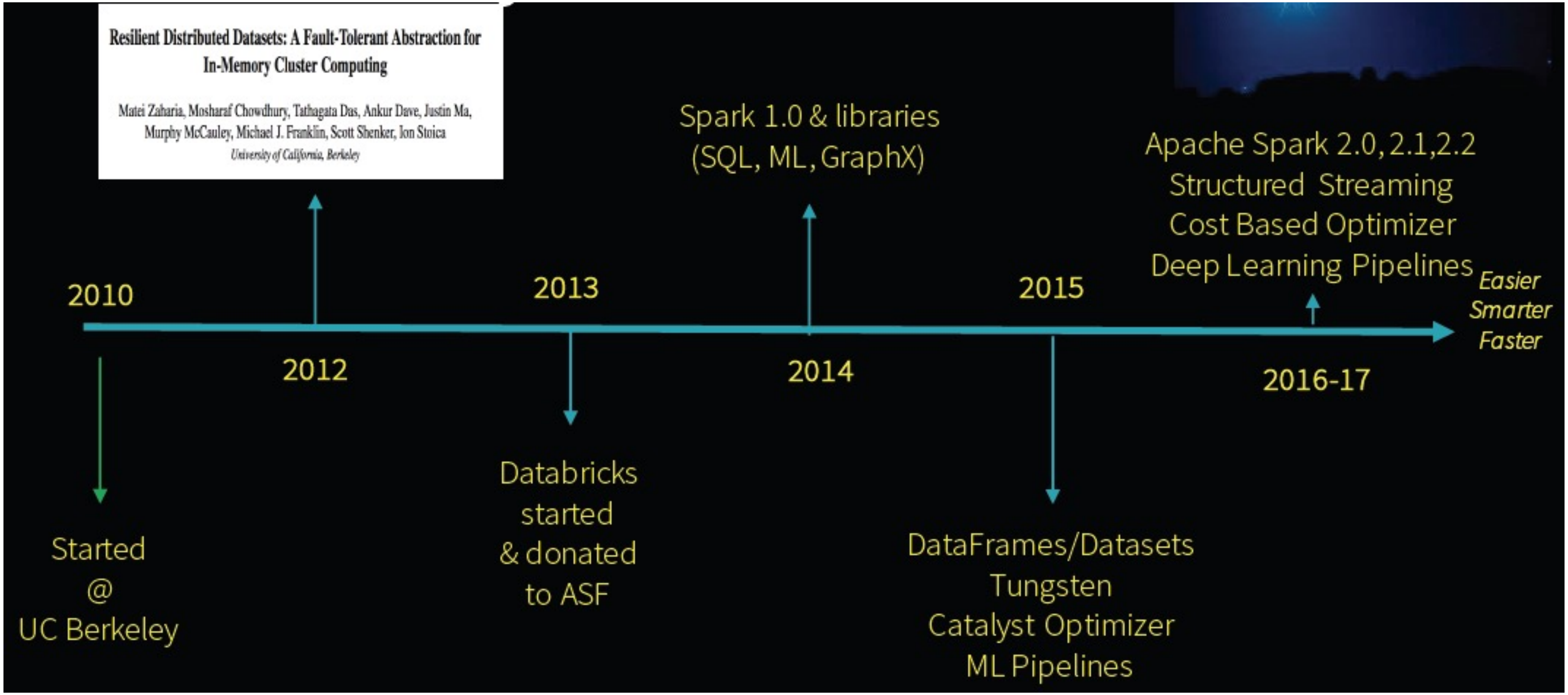
RDDs

Legend:

Optimized libraries

Unoptimized libraries

# Evolution Timeline of Spark



**Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing**

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica
*University of California, Berkeley*

2010

Started
@
UC Berkeley

2012

2013

Databricks
started
& donated
to ASF

2014

Spark 1.0 & libraries
(SQL, ML, GraphX)

2015

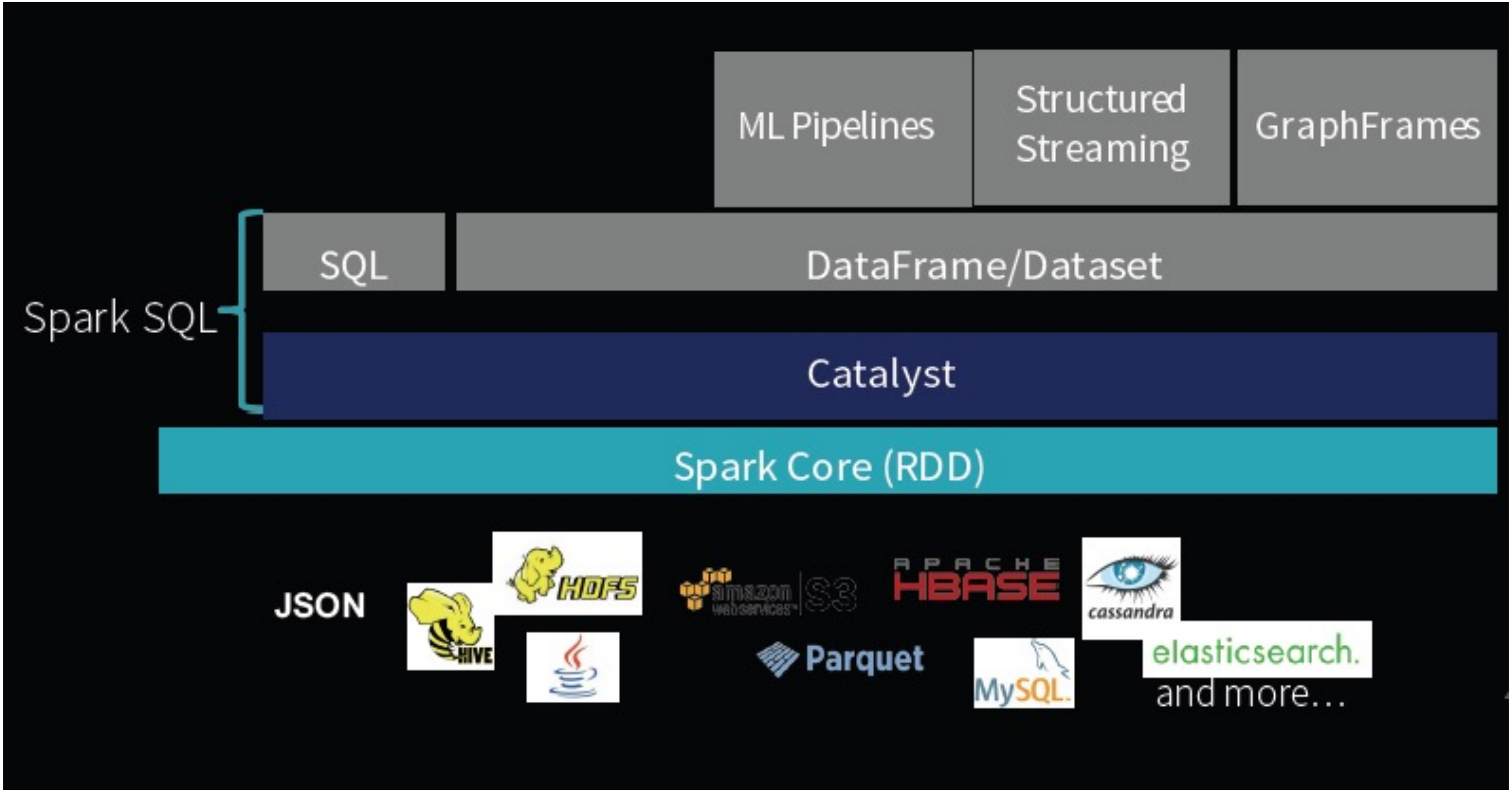
DataFrames/Datasets
Tungsten
Catalyst Optimizer
ML Pipelines

2016-17

Apache Spark 2.0, 2.1,2.2
Structured Streaming
Cost Based Optimizer
Deep Learning Pipelines

*Easier*
*Smarter*
*Faster*

# Foundational Spark 2.x Components

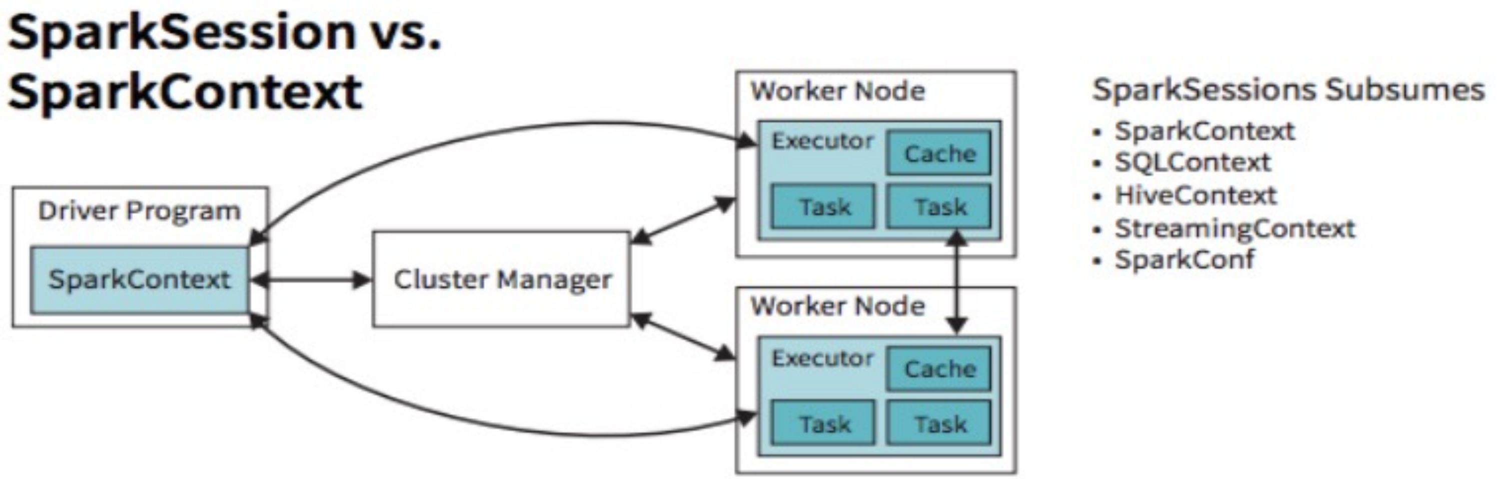


# Long Term Role of RDD, DataFrames & DataSets on Spark

- RDD as the low-level API in Spark
  - For control and certain type-safety in Java/ Scala
- Datasets & DataFrames give richer semantics & optimizations
  - For semi-structured data and DSL like operations
  - New libraries will increasingly use these as interchange format
  - Examples: Structured Streaming, MLib, GraphFrames, and Deep Learning Pipelines

# SparkSession subsumes SparkContext

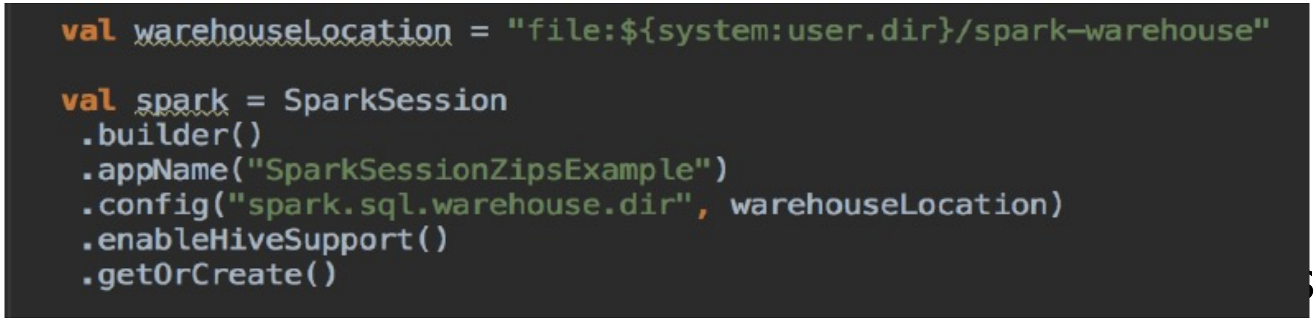
- Starting v2.0, SparkSession becomes the unified entry point, i.e. a Conduit, to Spark
  - Create Datasets/ DataFrames
  - Read/Write Data
  - Work with metadata
  - Set/Get Spark Configuration
  - Driver uses for Cluster Resource Management



**SparkSession vs. SparkContext**

SparkSessions Subsumes

- SparkContext
- SQLContext
- HiveContext
- StreamingContext
- SparkConf



```
val warehouseLocation = "file:${system:user.dir}/spark-warehouse"

val spark = SparkSession
  .builder()
  .appName("SparkSessionZipsExample")
  .config("spark.sql.warehouse.dir", warehouseLocation)
  .enableHiveSupport()
  .getOrCreate()
```

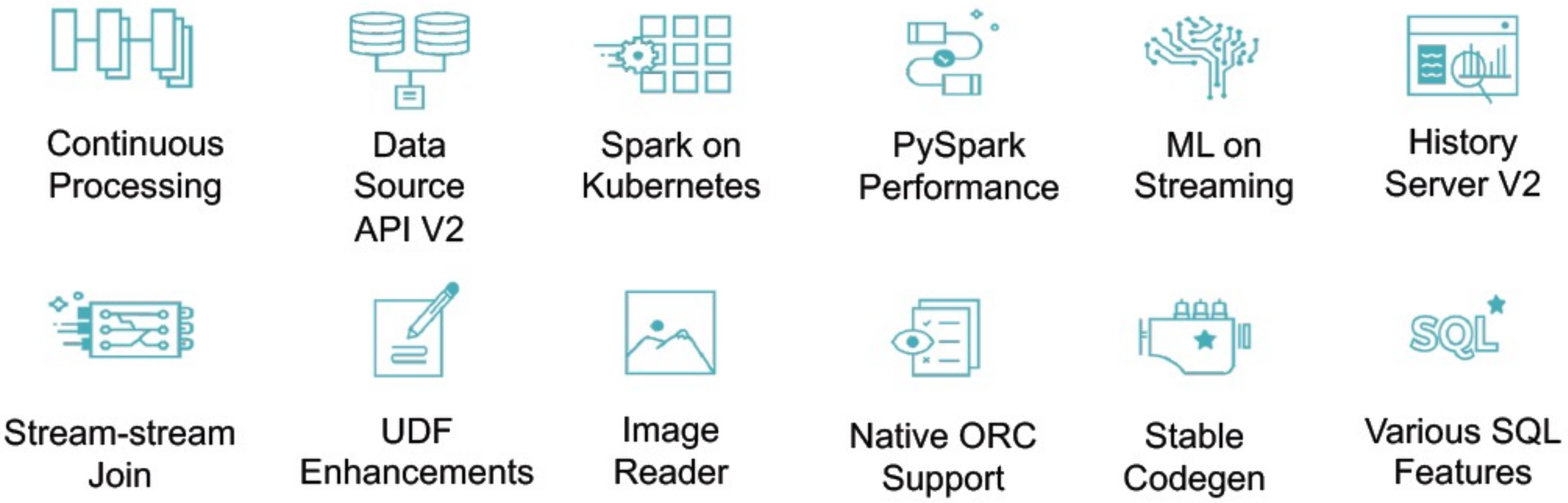
# Major Features in Spark 2.0

Tungsten Phase 2
speedups of 5-10x

Structured Streaming
real-time engine
on SQL/DataFrames

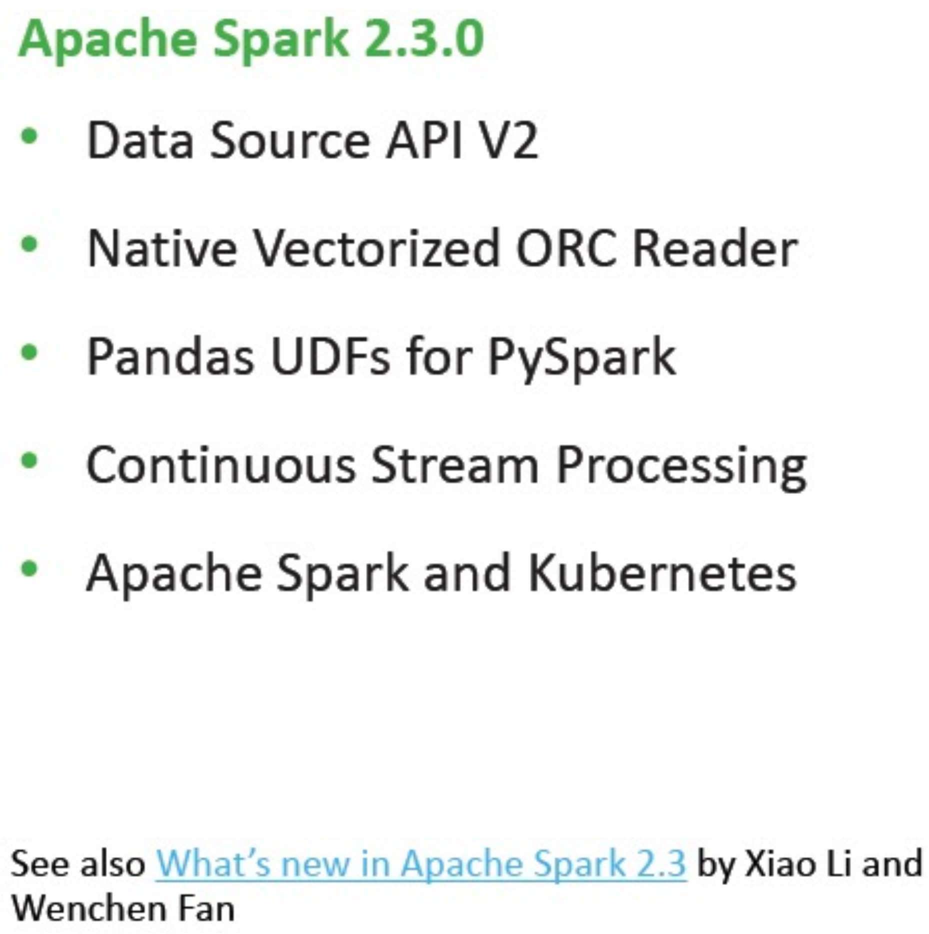
Unifying Datasets
and DataFrames

# Major Features since Spark 2.2



- Continuous Processing
- Data Source API V2
- Spark on Kubernetes
- PySpark Performance
- ML on Streaming
- History Server V2
- Stream-stream Join
- UDF Enhancements
- Image Reader
- Native ORC Support
- Stable Codegen
- Various SQL Features

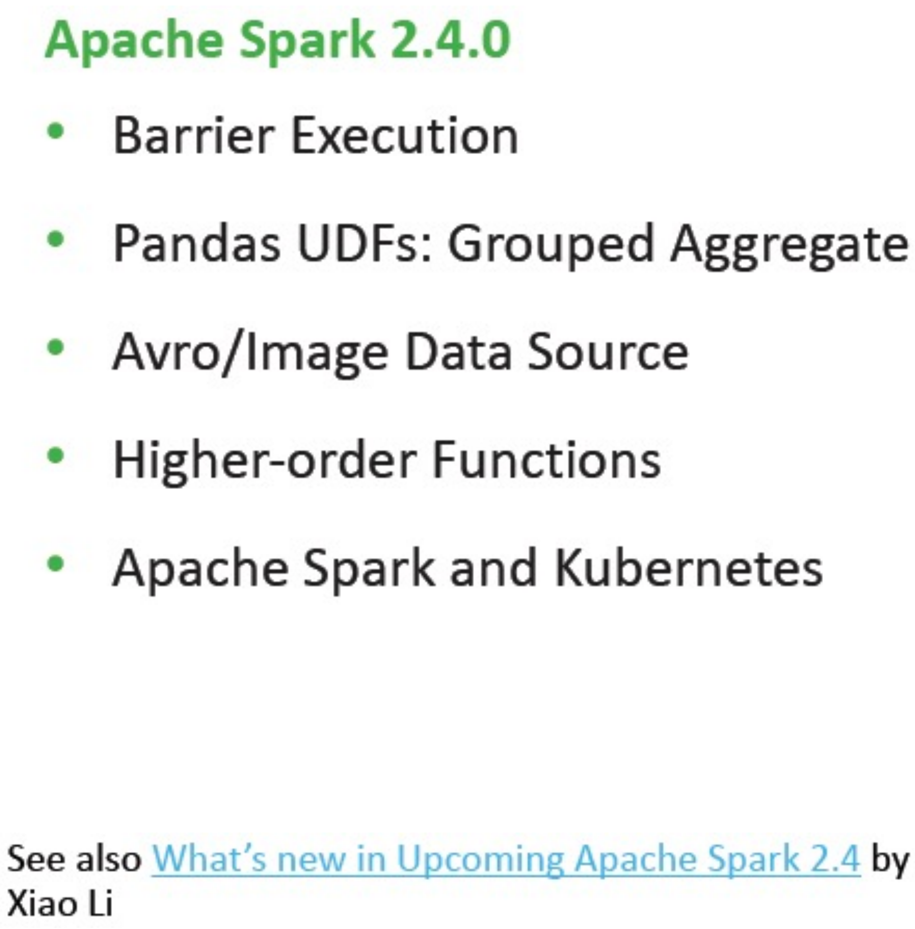
# Key Features in Apache Spark 2.3 & 2.4



### Apache Spark 2.3.0

- Data Source API V2
- Native Vectorized ORC Reader
- Pandas UDFs for PySpark
- Continuous Stream Processing
- Apache Spark and Kubernetes

See also What's new in Apache Spark 2.3 by Xiao Li and Wenchen Fan



### Apache Spark 2.4.0

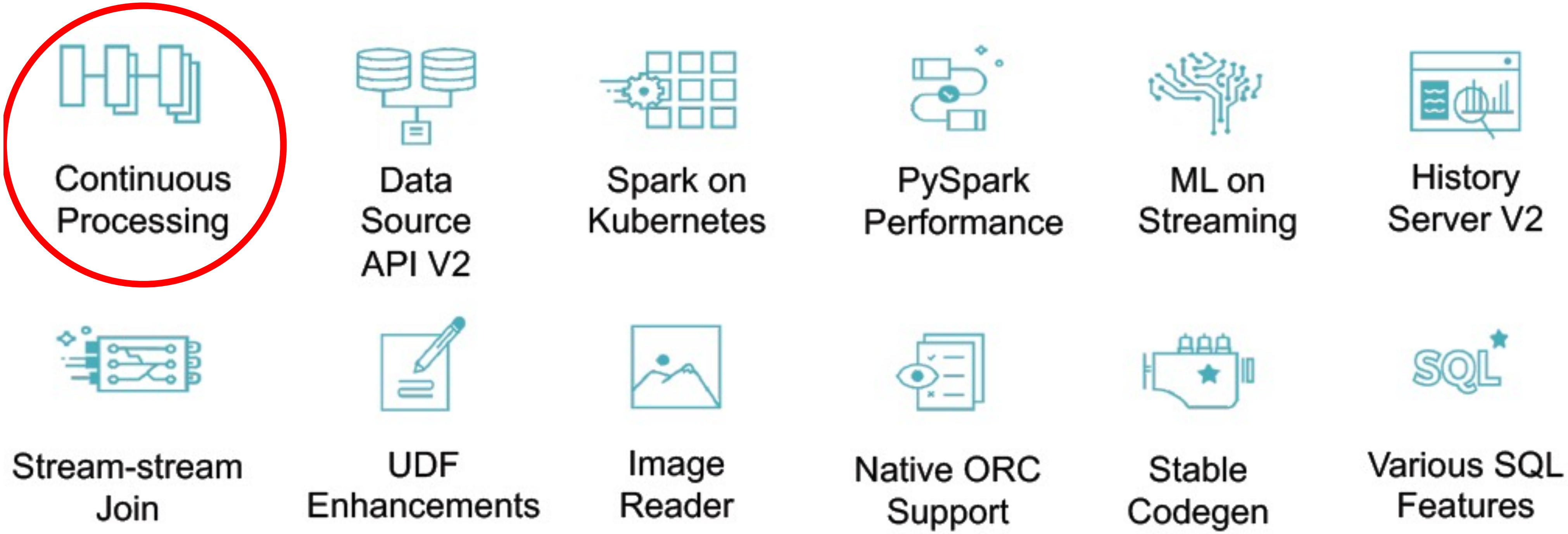
- Barrier Execution
- Pandas UDFs: Grouped Aggregate
- Avro/Image Data Source
- Higher-order Functions
- Apache Spark and Kubernetes

See also What's new in Upcoming Apache Spark 2.4 by Xiao Li

# Summary of
# Key Efforts in Spark 2.X (ver2.4 circa Nov 2018)

- Structured Streaming
  - Unification of the APIs
  - Event-time Aggregations/ Processing to handle out-of-order/late data
  - Other Streaming sources/sinks
  - Support Structured Streaming in other libraries, e.g. MLlib, GraphFrames
  - Support of Continuous Processing model, i.e. true (low-latency) streaming instead of stream processing via micro-batching.
  - Spark over Kubernetes: deploying Spark not only as a framework but also as a containerized distributed application/ library !
  - Machine Learning – Optimized Model Tuning
- Iteration as a First-Class concept in DataFrames
- Cost-based Query Optimization for ML/Graph Algorithms
  - Caching, Communication, Serialization, Compression
- Spark + GPUs
- High-level API for Deep-Learning Pipeline in Spark MLlib
  - Built on TensorFlow, Keras, BigDL
- Project Hydrogen - enhancing Integration of other ML frameworks with Spark
- Better Infrastructure support of Production-level Complete ML Life-cycle with MLflow

# More Details on some
# Key Features since Spark 2.2



Continuous Processing

Data Source API V2

Spark on Kubernetes

PySpark Performance

ML on Streaming

History Server V2

Stream-stream Join

UDF Enhancements

Image Reader

Native ORC Support

Stable Codegen

Various SQL Features

# Continuous (Stream) Processing

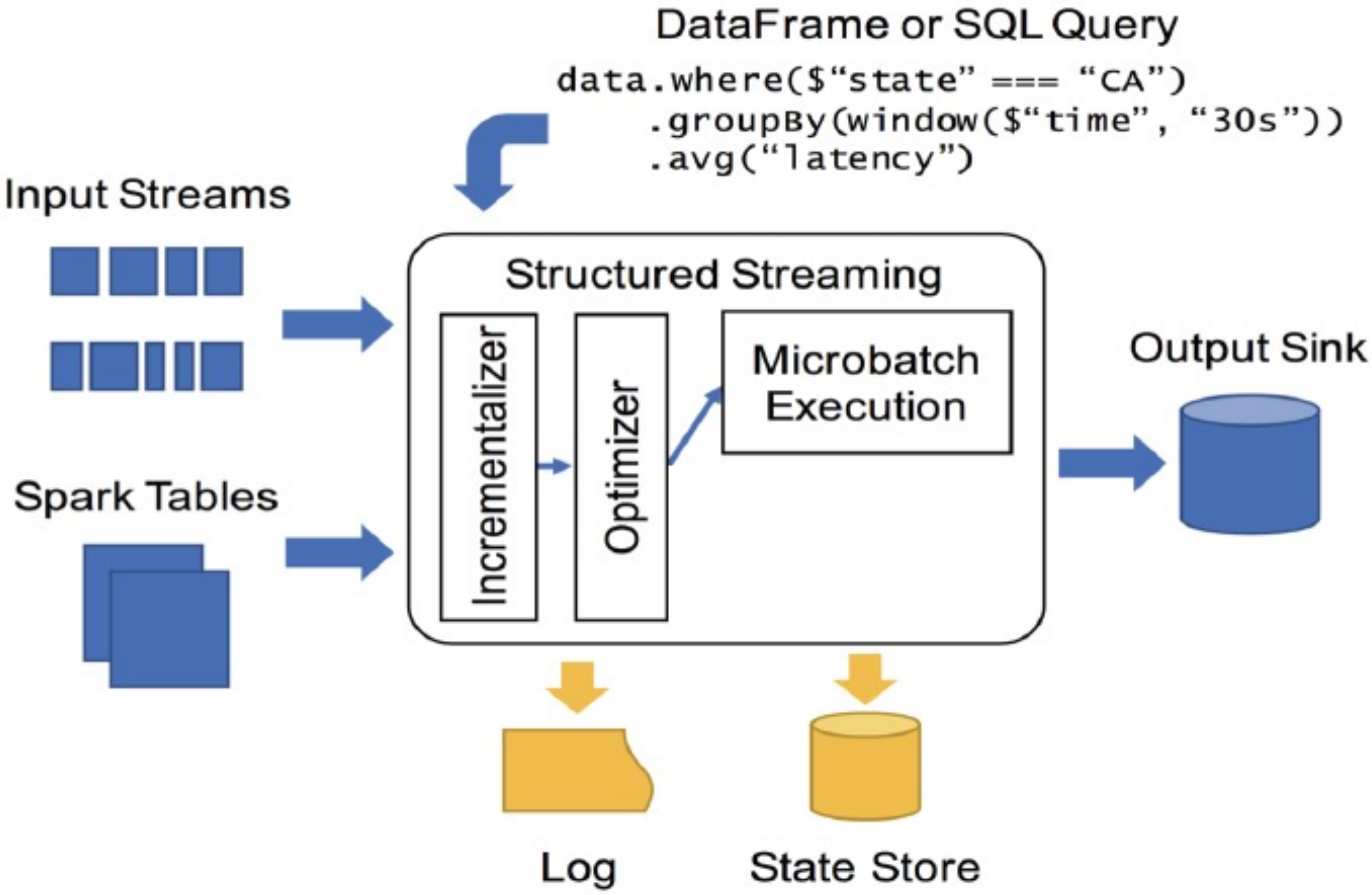
- A new execution mode introduced since V2.2 that allows fully pipelined execution (like Flink)
  - Streaming execution without micro-batches
  - Support asynchronous checkpoints and ~1msec latency

  => To enable Spark to stay competitive with Flink

  - No changes required for user codes.
- Still WIP, not all features are supported as of Mar 2019.
- See initial proposal at:
  - https://issues.apache.org/jira/browse/SPARK-20928

# Continuous (Stream) Processing (cont’d)

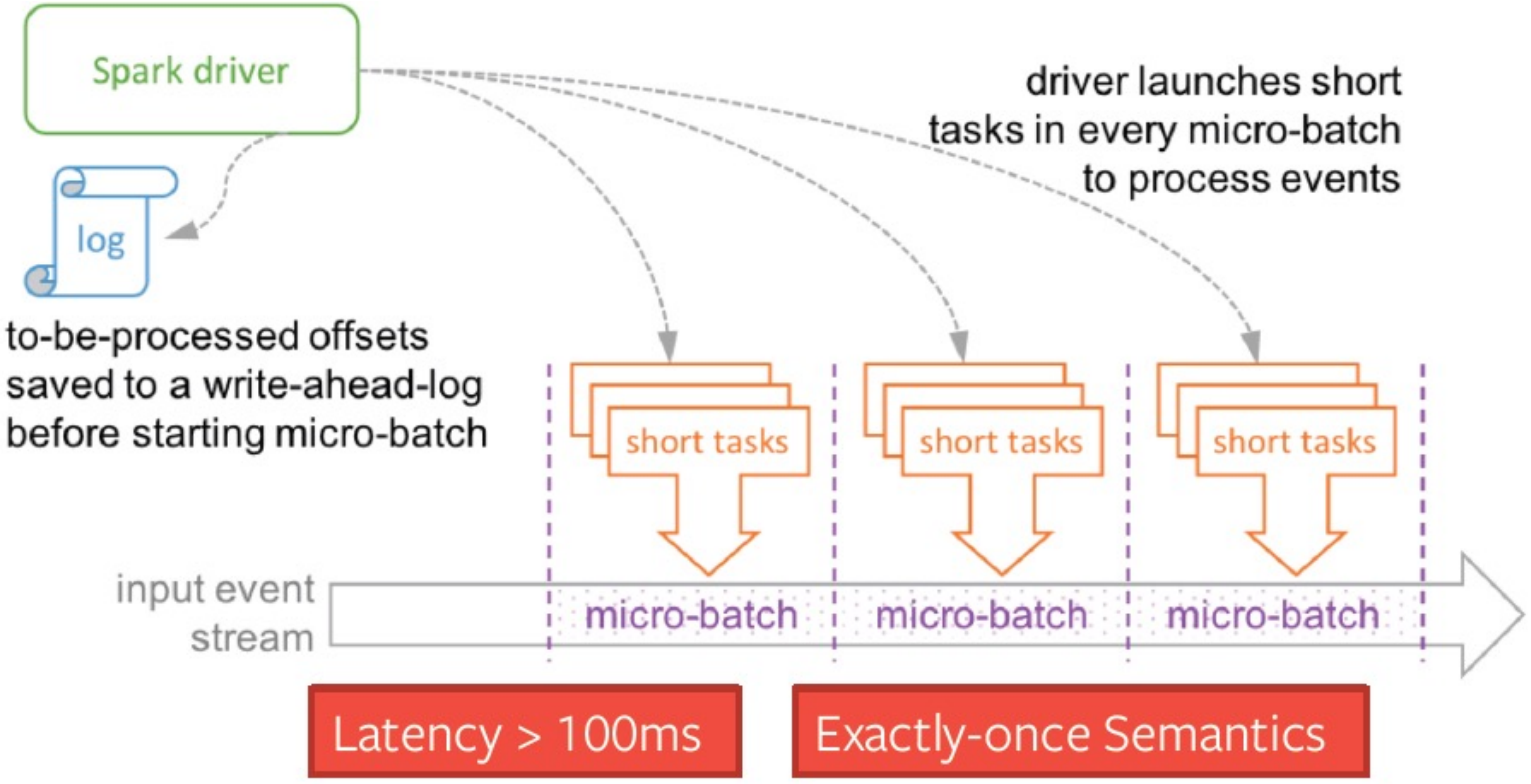
## Structured Streaming



```
data.where($“state” === “CA”)
    .groupBy(window($“time”, “30s”))
    .avg(“latency”)
```

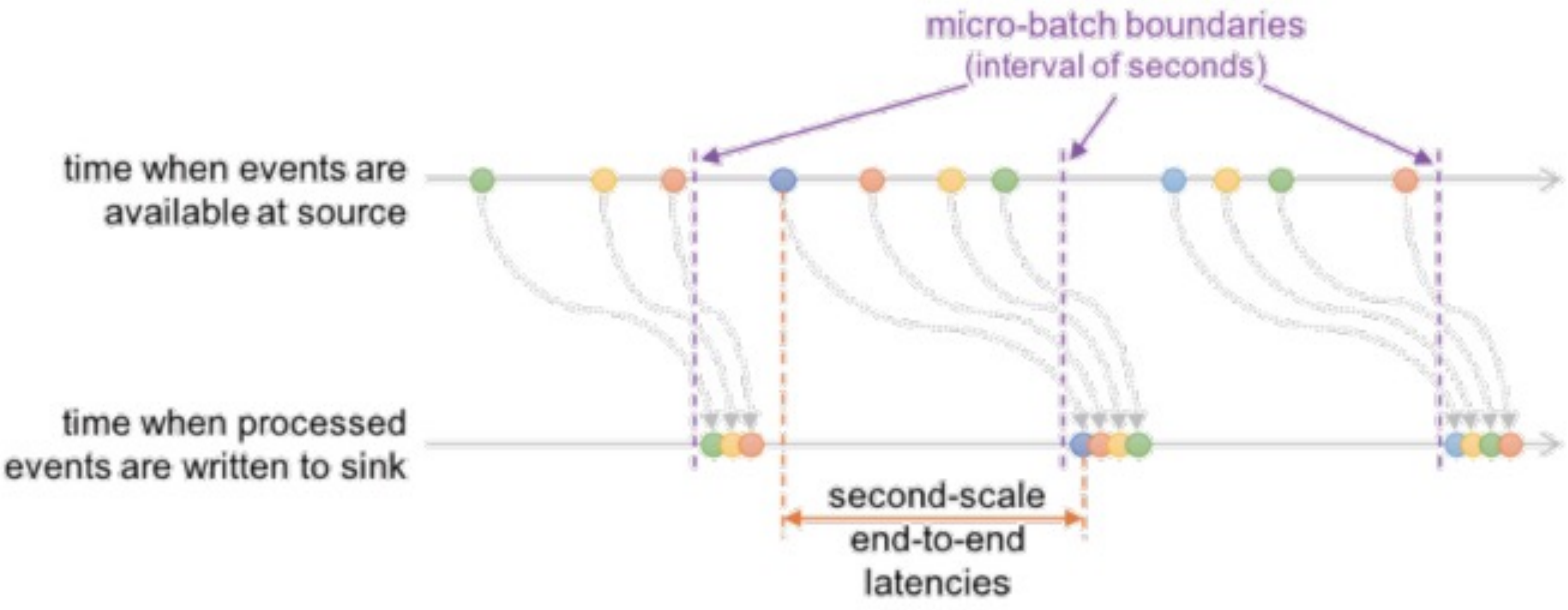
# Continuous (Stream) Processing (cont'd)

## Micro Batch Execution


Spark driver
log
to-be-processed offsets saved to a write-ahead-log before starting micro-batch
driver launches short tasks in every micro-batch to process events
short tasks
short tasks
short tasks
input event stream
micro-batch
micro-batch
micro-batch
Latency > 100ms
Exactly-once Semantics

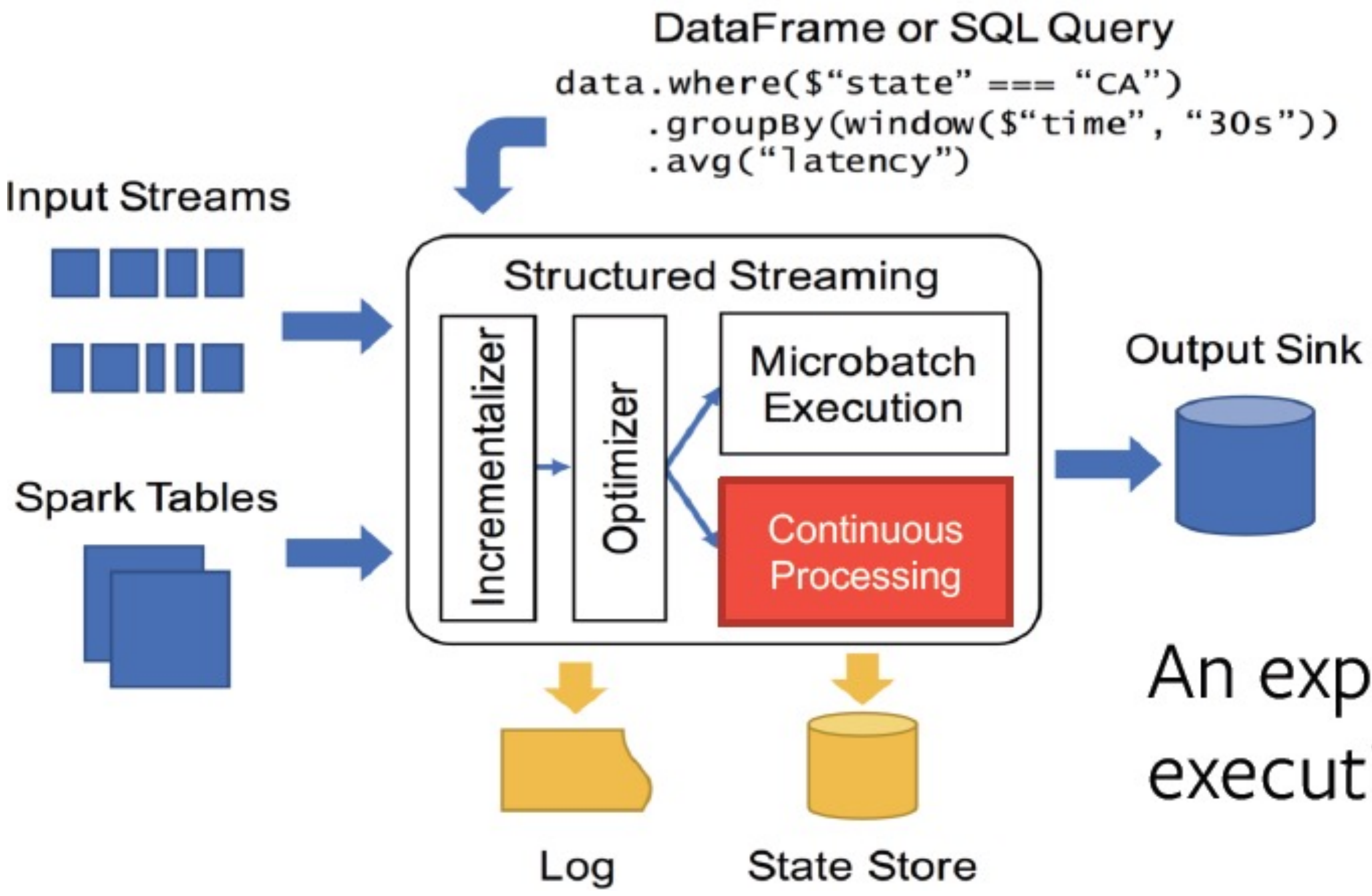
# Continuous (Stream) Processing (cont’d)

## Micro Batch Execution


micro-batch boundaries
(interval of seconds)
time when events are available at source
time when processed events are written to sink
second-scale end-to-end latencies

See also Continuous Processing in Structured Streaming by Josh Torres

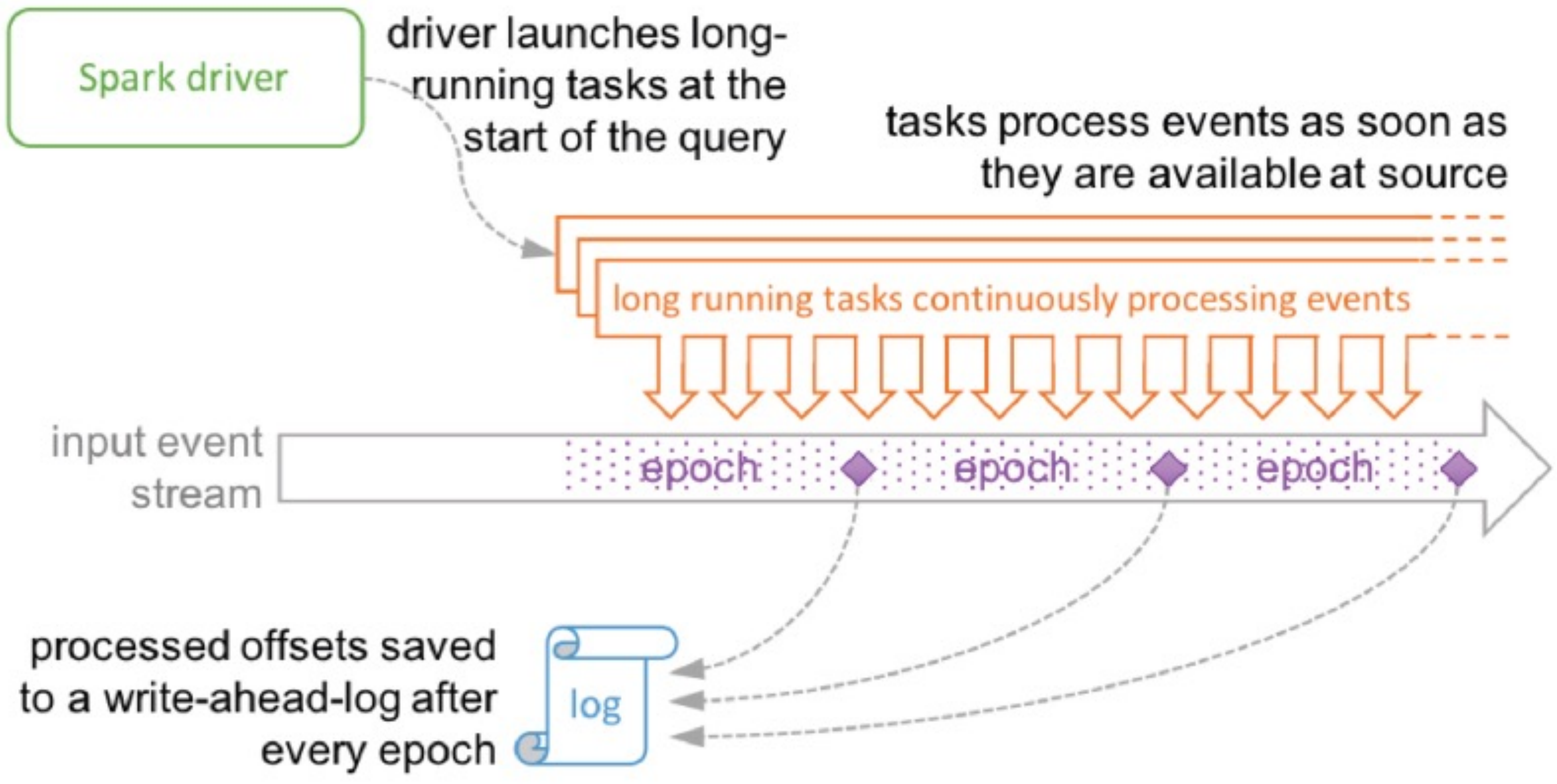
# Continuous (Stream) Processing (cont’d)



```
data.where($“state” === “CA”)
    .groupBy(window($“time”, “30s”))
    .avg(“latency”)
```

An experimental execution mode

# Continuous (Stream) Processing (cont'd)



driver launches long-running tasks at the start of the query

tasks process events as soon as they are available at source

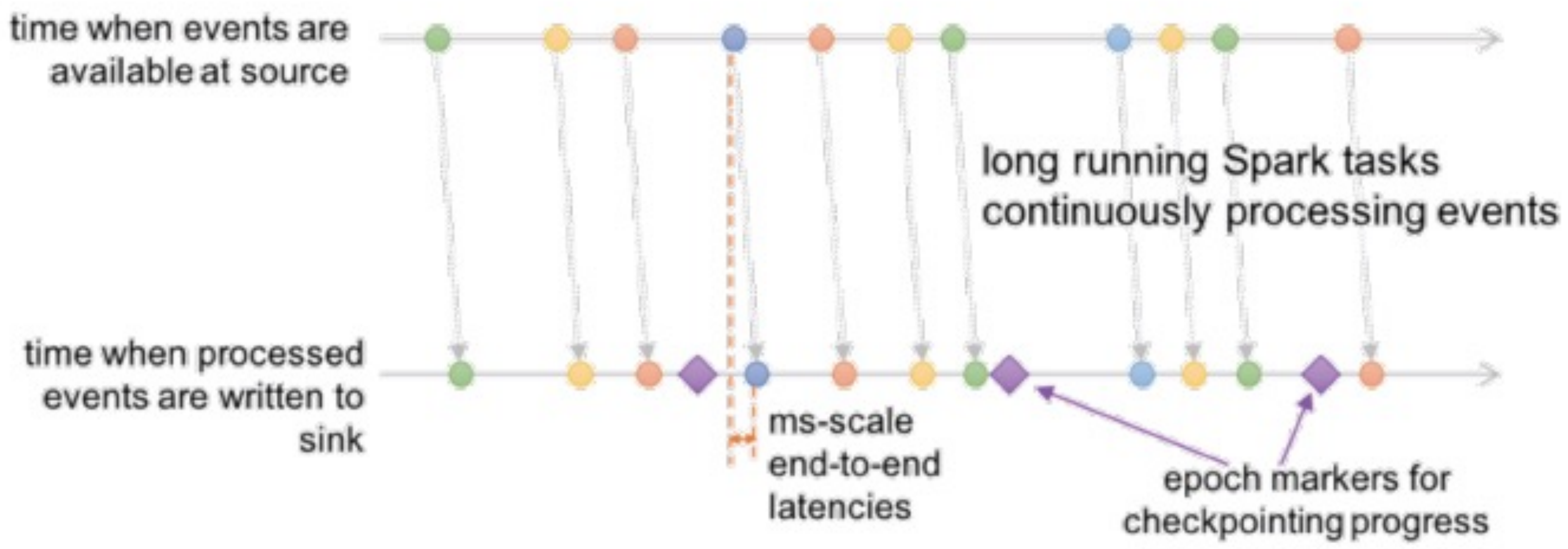
processed offsets saved to a write-ahead-log after every epoch

Latency ~1ms

At-least once Semantics

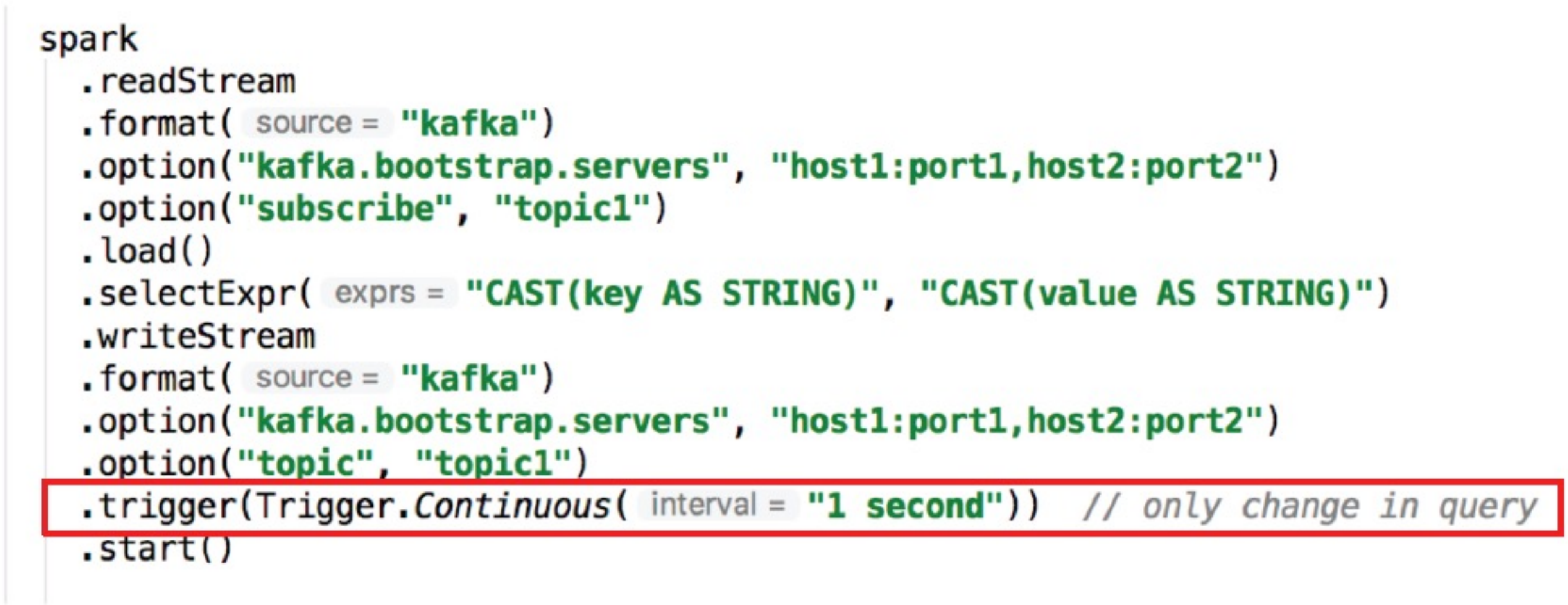
# Continuous (Stream) Processing (cont'd)

## Structured Streaming: Continuous Processing


time when events are available at source
long running Spark tasks continuously processing events
time when processed events are written to sink
ms-scale end-to-end latencies
epoch markers for checkpointing progress

See also Continuous Processing in Structured Streaming by Josh Torres

# Continuous (Stream) Processing (cont'd)

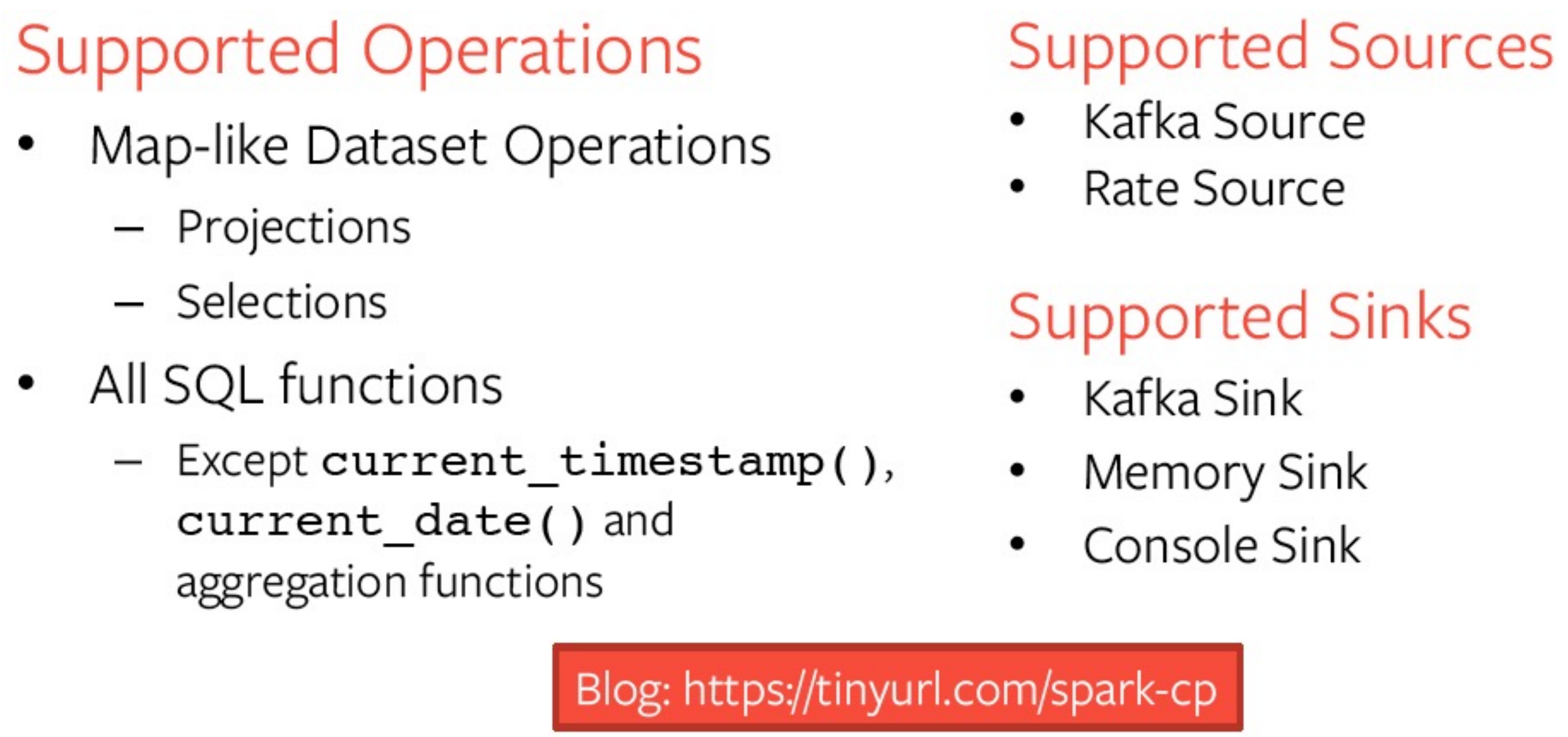


```
spark
  .readStream
  .format( source = "kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("subscribe", "topic1")
  .load()
  .selectExpr( exprs = "CAST(key AS STRING)", "CAST(value AS STRING)")
  .writeStream
  .format( source = "kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("topic", "topic1")
  .trigger(Trigger.Continuous( interval = "1 second"))  // only change in query
  .start()
```

https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#continuous-processing

See also Spark Summit Keynote Demo by Michael Armbrust

# Continuous (Stream) Processing (cont'd)



## Supported Operations

- Map-like Dataset Operations
  - Projections
  - Selections
- All SQL functions
  - Except `current_timestamp()`, `current_date()` and aggregation functions

## Supported Sources

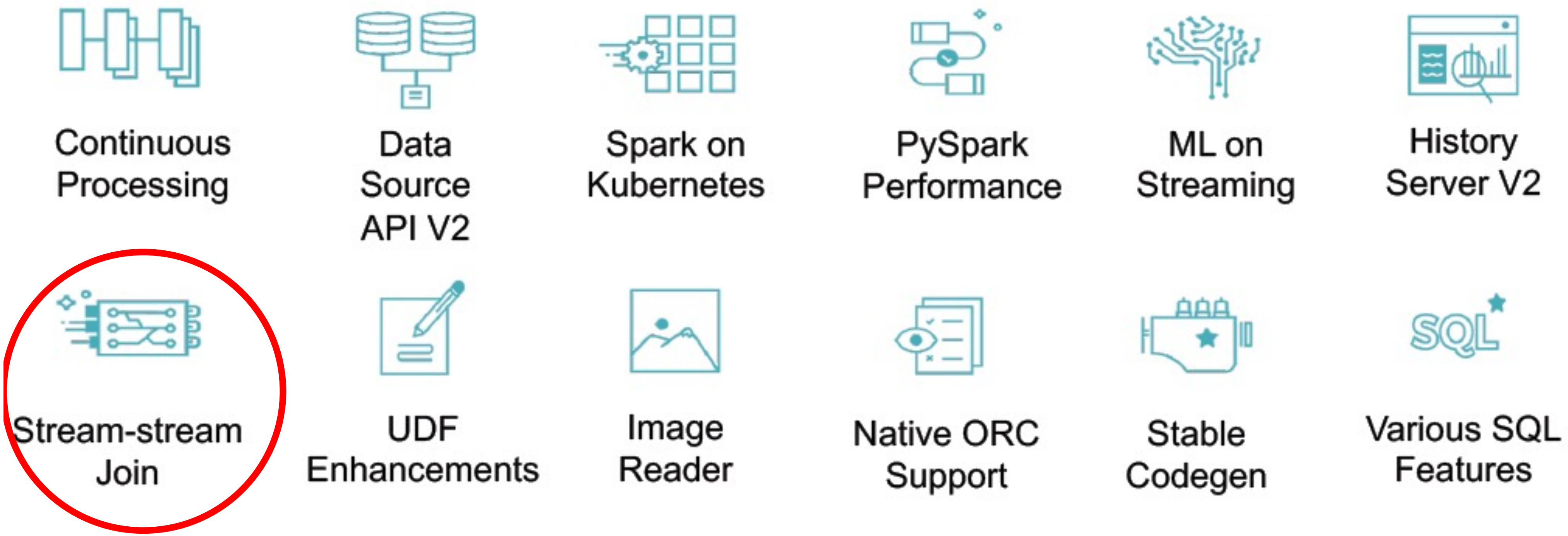
- Kafka Source
- Rate Source

## Supported Sinks

- Kafka Sink
- Memory Sink
- Console Sink

Blog: https://tinyurl.com/spark-cp

# Major Features since Spark 2.2



Continuous Processing

Data Source API V2

Spark on Kubernetes

PySpark Performance

ML on Streaming

History Server V2

Stream-stream Join

UDF Enhancements

Image Reader

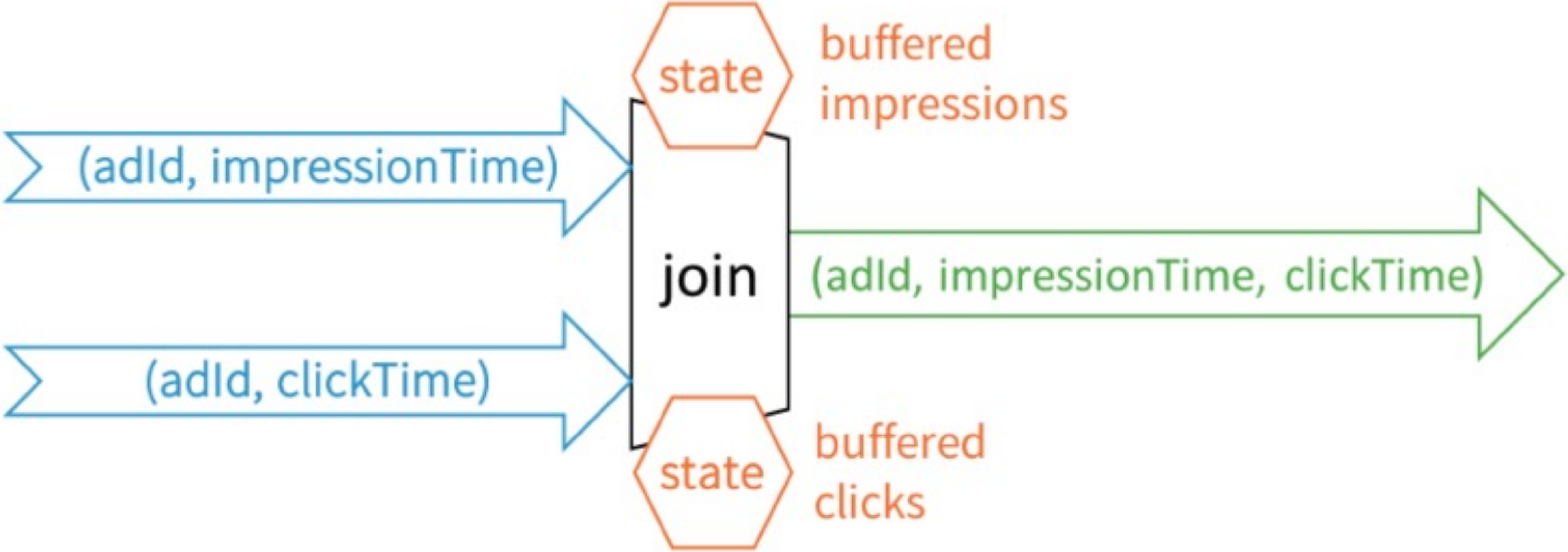
Native ORC Support

Stable Codegen

Various SQL Features

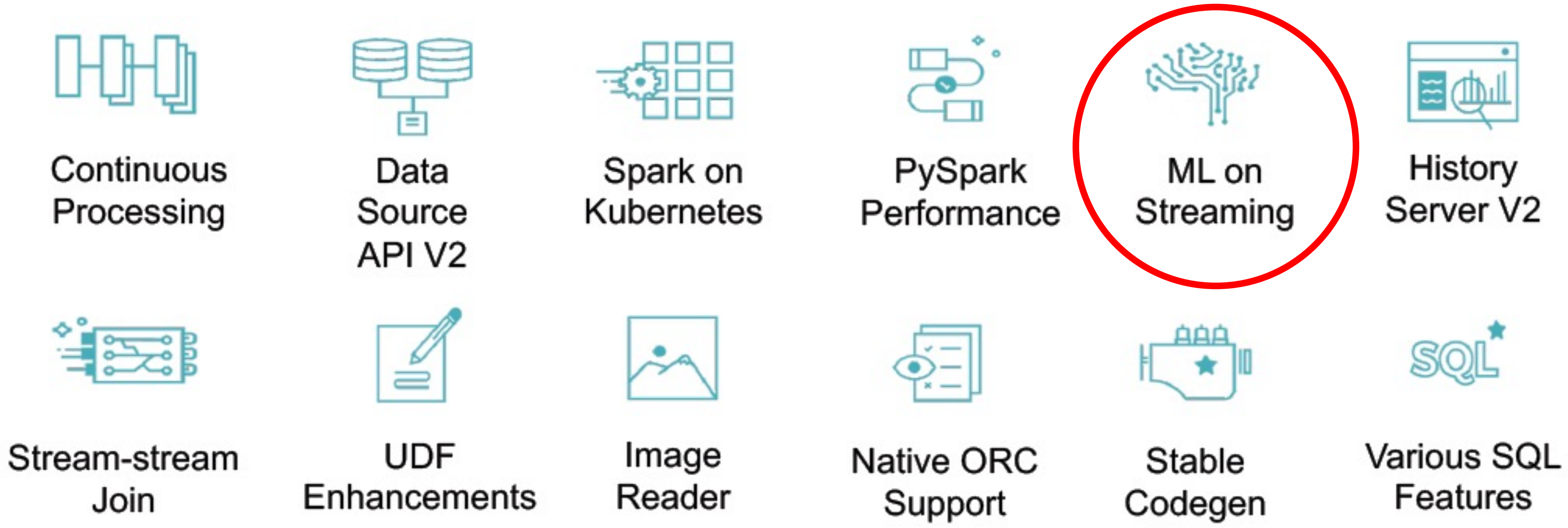
# Stream to Stream Joins (in V2.3)

Stream-stream Join



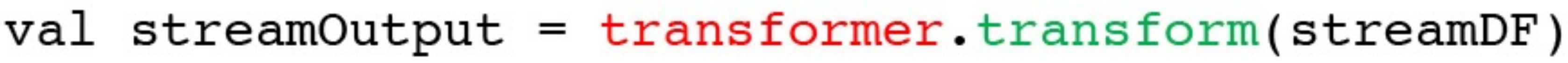
See also Introducing Stream-Stream Joins in Apache Spark 2.3 by Tathagata Das and Joseph Torres

# Major Features since Spark 2.2



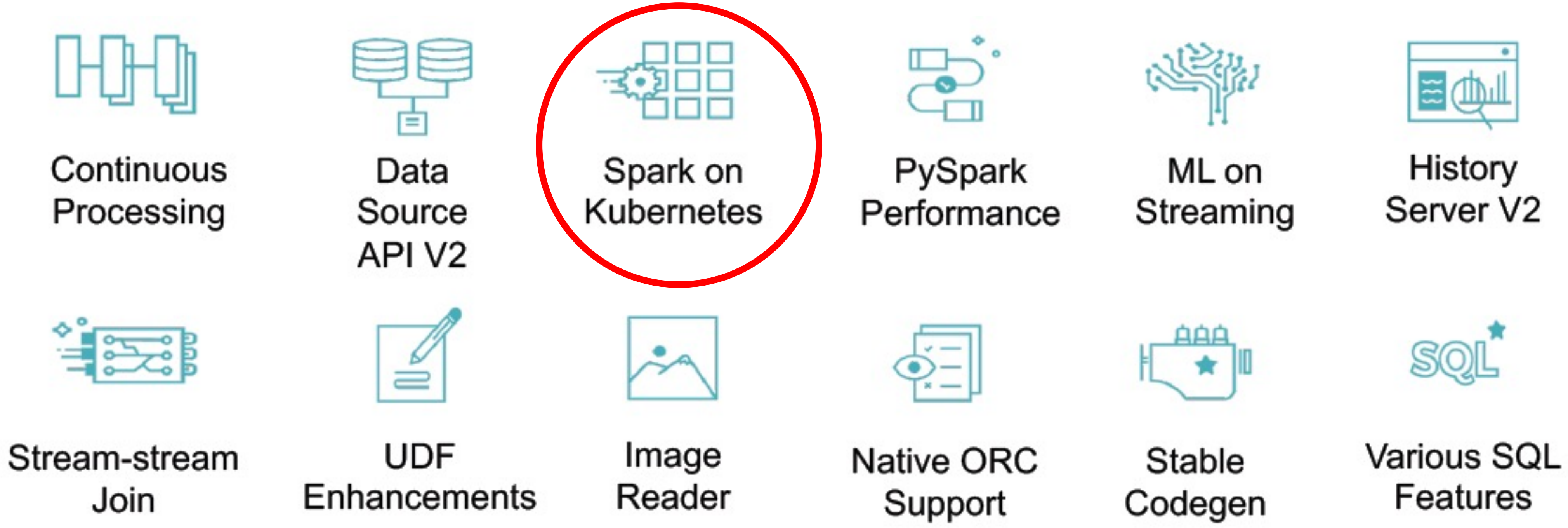
- Continuous Processing
- Data Source API V2
- Spark on Kubernetes
- PySpark Performance
- ML on Streaming
- History Server V2
- Stream-stream Join
- UDF Enhancements
- Image Reader
- Native ORC Support
- Stable Codegen
- Various SQL Features

# ML on Streaming

- ML model transformation/ prediction on Batch and Streaming data with Unified API
- After fitting a ML model or ML Pipeline, user can deploy it in a Streaming job
  - 

    ```
    val streamOutput = transformer.transform(streamDF)
    ```

# Major Features since Spark 2.2



Continuous Processing

Data Source API V2

Spark on Kubernetes

PySpark Performance

ML on Streaming

History Server V2

Stream-stream Join

UDF Enhancements

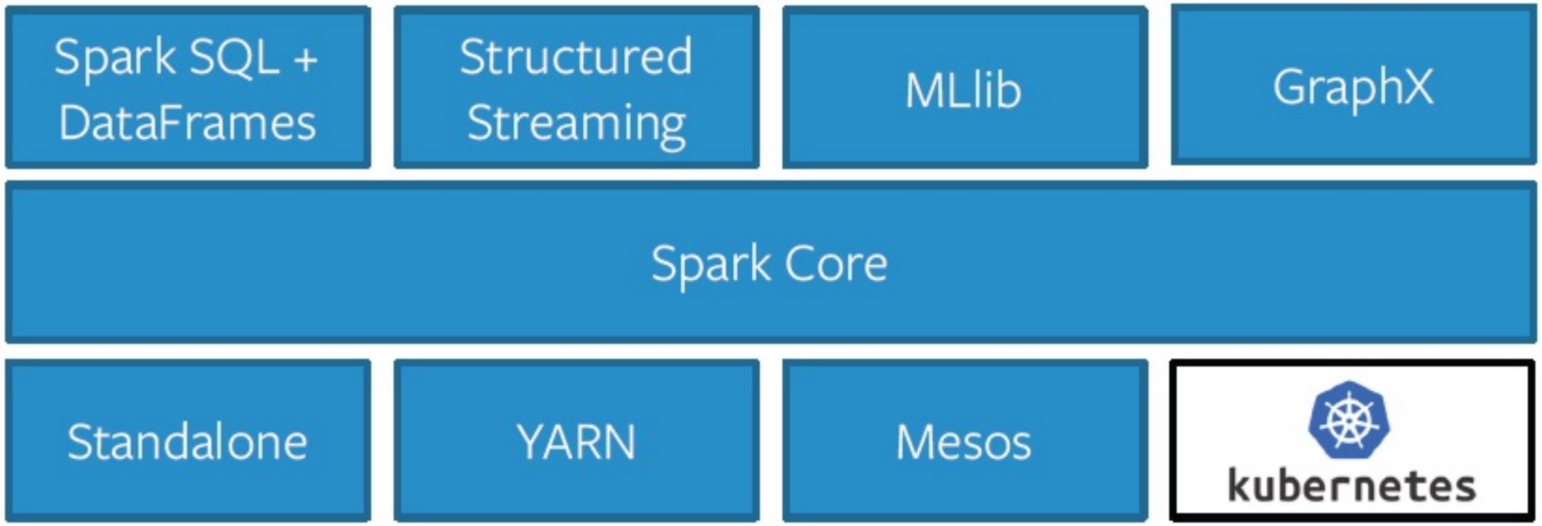
Image Reader

Native ORC Support

Stable Codegen

Various SQL Features

# Apache Spark on Kubernetes



Spark SQL + DataFrames

Structured Streaming

MLlib

GraphX

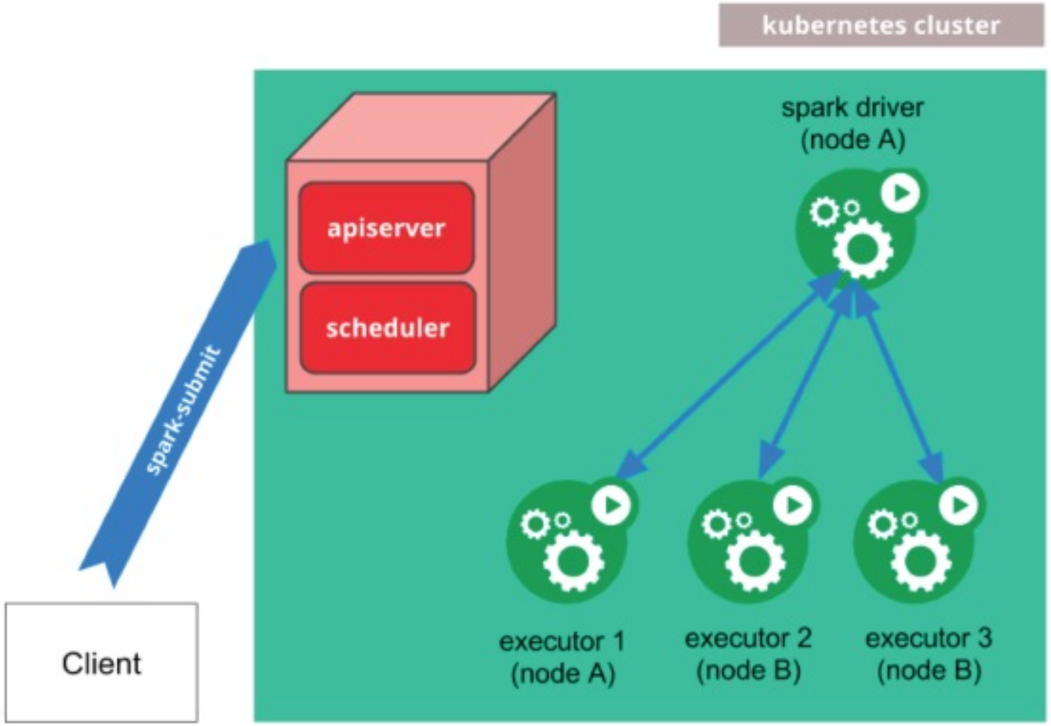
Spark Core

Standalone

YARN

Mesos

kubernetes

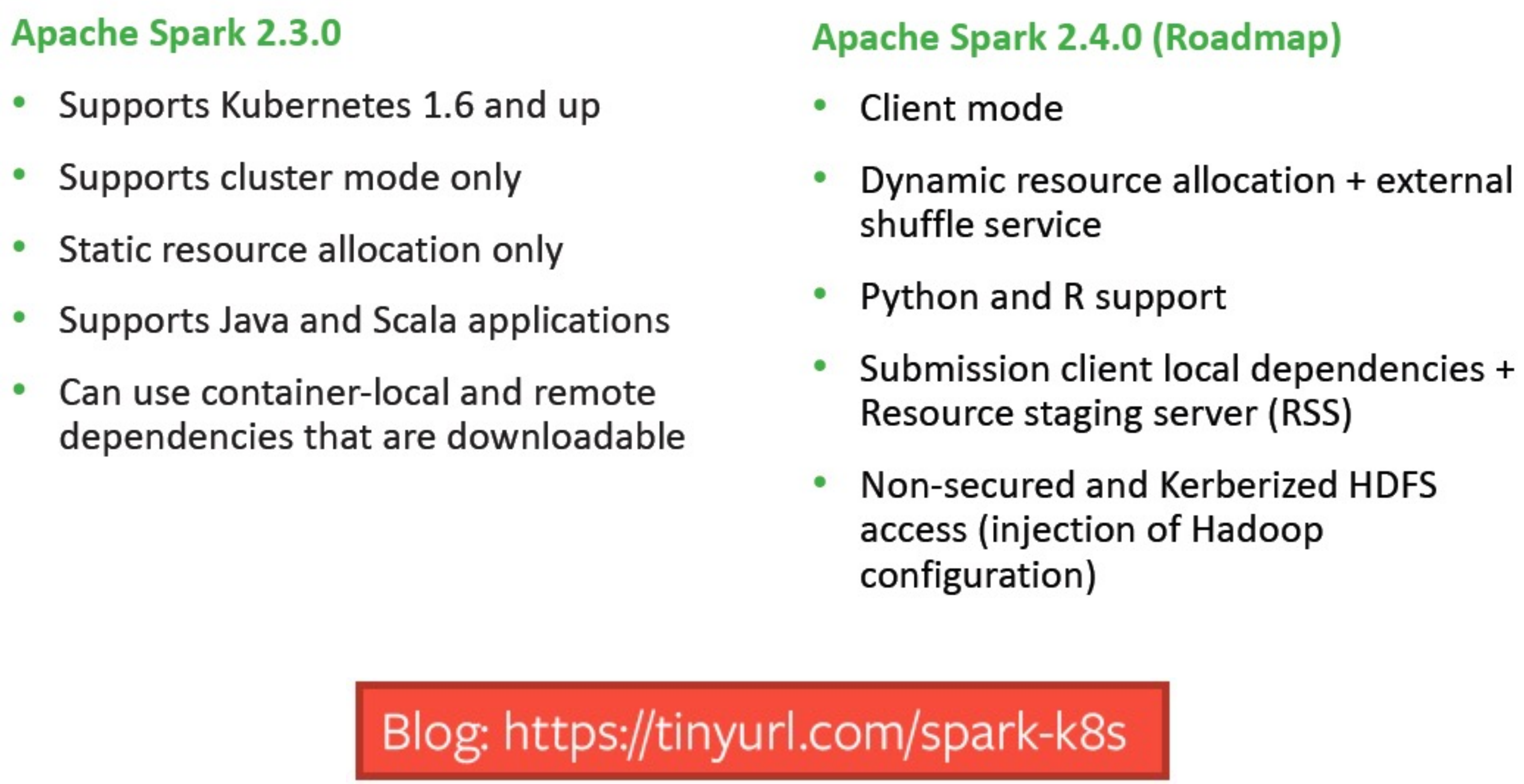


See also: https://spark.apache.org/docs/2.3.2/running-on-kubernetes.html

# Apache Spark on Kubernetes (cont'd)

- Driver runs in a Kubernetes pod created by the submission client and creates pods that run the executors in response to requests from the Spark Scheduler
- Make direct use of Kubernetes clusters for Multi-tenancy and sharing through Namespaces and Quotas, as well as administrative features such as Pluggable Authorization and Logging.

# Apache Spark and Kubernetes (cont'd)



**Apache Spark 2.3.0**

- Supports Kubernetes 1.6 and up
- Supports cluster mode only
- Static resource allocation only
- Supports Java and Scala applications
- Can use container-local and remote dependencies that are downloadable

**Apache Spark 2.4.0 (Roadmap)**

- Client mode
- Dynamic resource allocation + external shuffle service
- Python and R support
- Submission client local dependencies + Resource staging server (RSS)
- Non-secured and Kerberized HDFS access (injection of Hadoop configuration)

Blog: https://tinyurl.com/spark-k8s

# Better Support ML/ AI in Production with MLflow

# Hidden Technical Debt in Machine Learning Systems
## (A NIPS 2015 paper from Google)

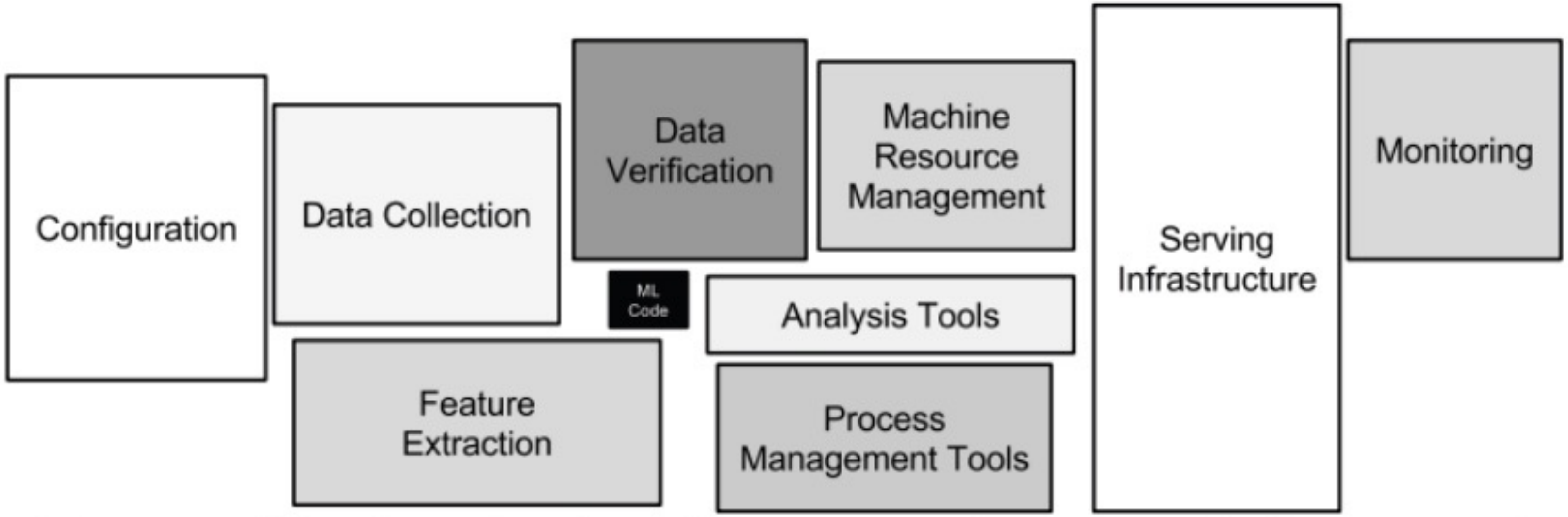


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

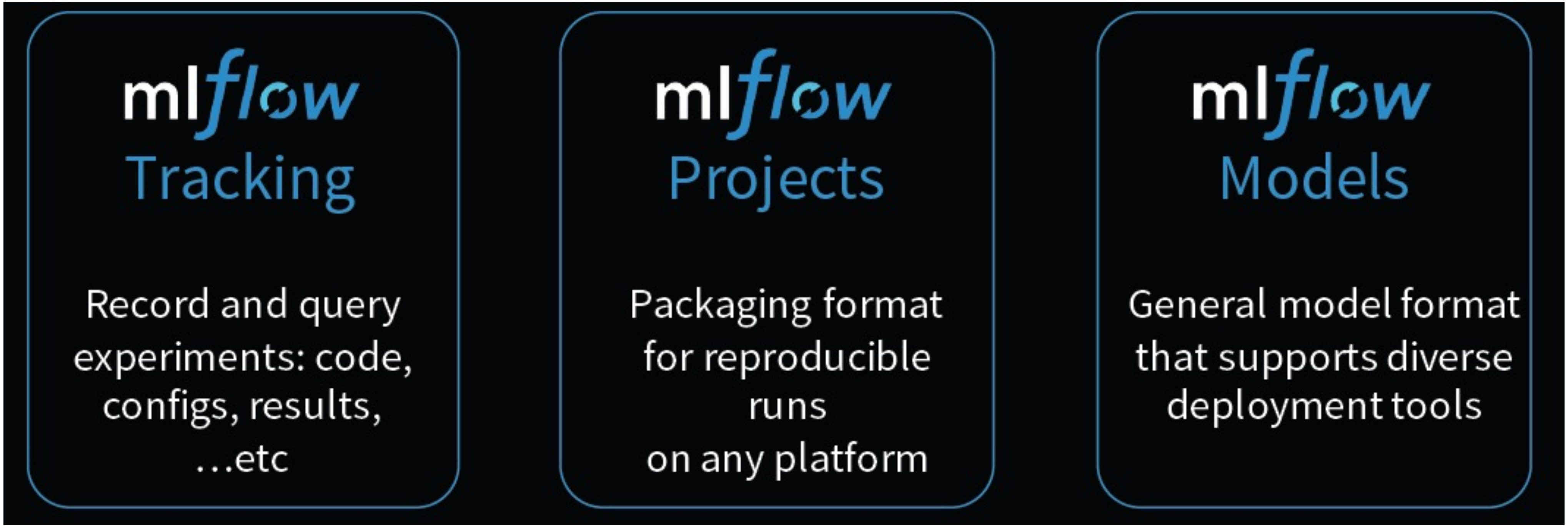
# MLflow

Goal of MLflow:

- To provide the tools to simplify the ML lifecycle (in an industrial production-grade environment)
- A Lightweight, open platform that integrates with other ML systems readily

- Available APIs: Python, Java and R
- Develop model locally and track runs locally or remotely
- Deploy locally, cloud or on premise
- Visualize experiments

## Components of MLflow:



mlflow Tracking

Record and query experiments: code, configs, results, …etc

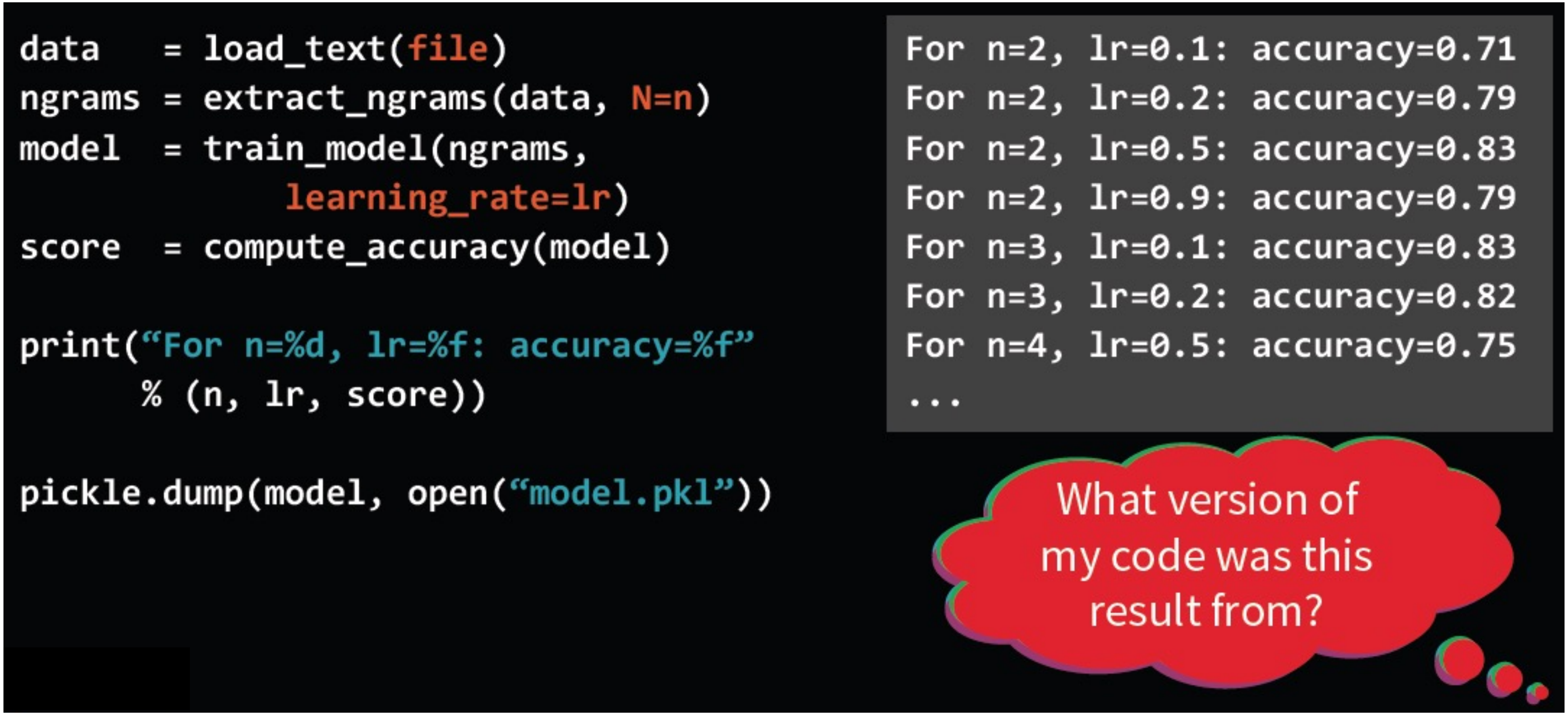
mlflow Projects

Packaging format for reproducible runs on any platform

mlflow Models

General model format that supports diverse deployment tools

# Model Development without MLflow



```
data   = load_text(file)
ngrams = extract_ngrams(data, N=n)
model  = train_model(ngrams,
             learning_rate=lr)
score  = compute_accuracy(model)

print(“For n=%d, lr=%f: accuracy=%f”
      % (n, lr, score))

pickle.dump(model, open(“model.pkl”))
```

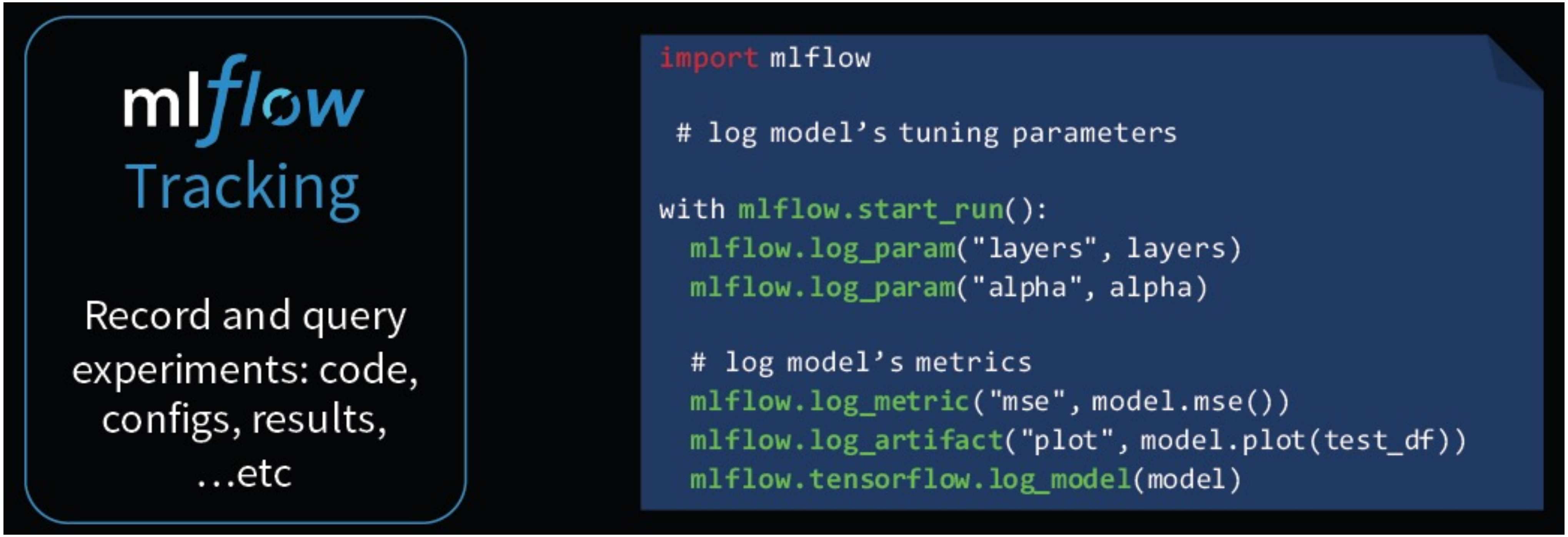
```
For n=2, lr=0.1: accuracy=0.71
For n=2, lr=0.2: accuracy=0.79
For n=2, lr=0.5: accuracy=0.83
For n=2, lr=0.9: accuracy=0.79
For n=3, lr=0.1: accuracy=0.83
For n=3, lr=0.2: accuracy=0.82
For n=4, lr=0.5: accuracy=0.75
...
```

What version of my code was this result from?

# Key Concepts in Tracking with MLflow

- Parameters: Key-value inputs to your code
- Metrics: numeric values (can update over time)
- Tags and Notes: information about a run
- Artifacts: files, data and models
- Source: what code was run ?
- Version: Which version of the code ?

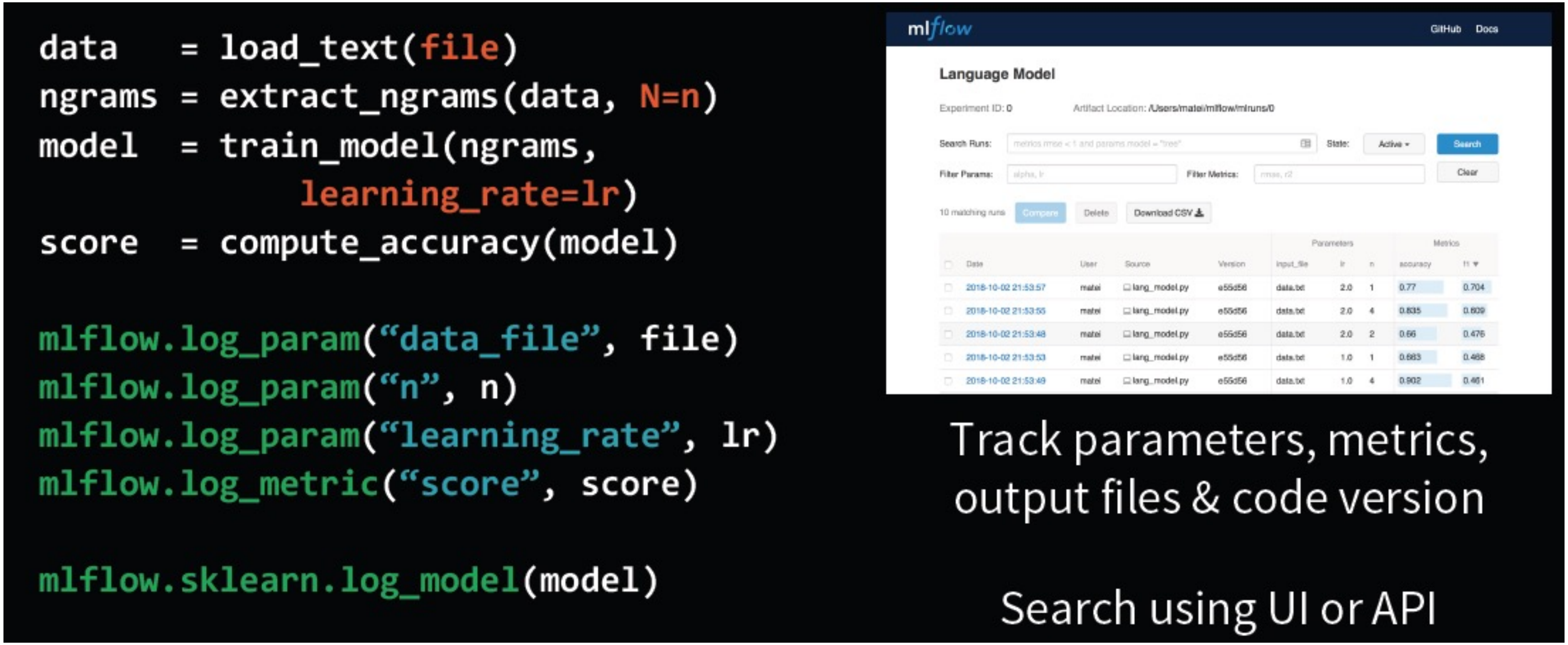
# MLflow Tracking API



Record and query experiments: code, configs, results, …etc

```
import mlflow

 # log model’s tuning parameters

with mlflow.start_run():
  mlflow.log_param("layers", layers)
  mlflow.log_param("alpha", alpha)

  # log model’s metrics
  mlflow.log_metric("mse", model.mse())
  mlflow.log_artifact("plot", model.plot(test_df))
  mlflow.tensorflow.log_model(model)
```

# Model Development with MLflow



```
data   = load_text(file)
ngrams = extract_ngrams(data, N=n)
model  = train_model(ngrams,
             learning_rate=lr)
score  = compute_accuracy(model)

mlflow.log_param("data_file", file)
mlflow.log_param("n", n)
mlflow.log_param("learning_rate", lr)
mlflow.log_metric("score", score)

mlflow.sklearn.log_model(model)
```

Language Model

Experiment ID: 0    Artifact Location: /Users/matei/mlflow/mlruns/0

Search Runs: metrics.rmse < 1 and params.model = "tree"    State: Active ▾    Search

Filter Params: alpha, lr    Filter Metrics: rmse, r2    Clear

10 matching runs

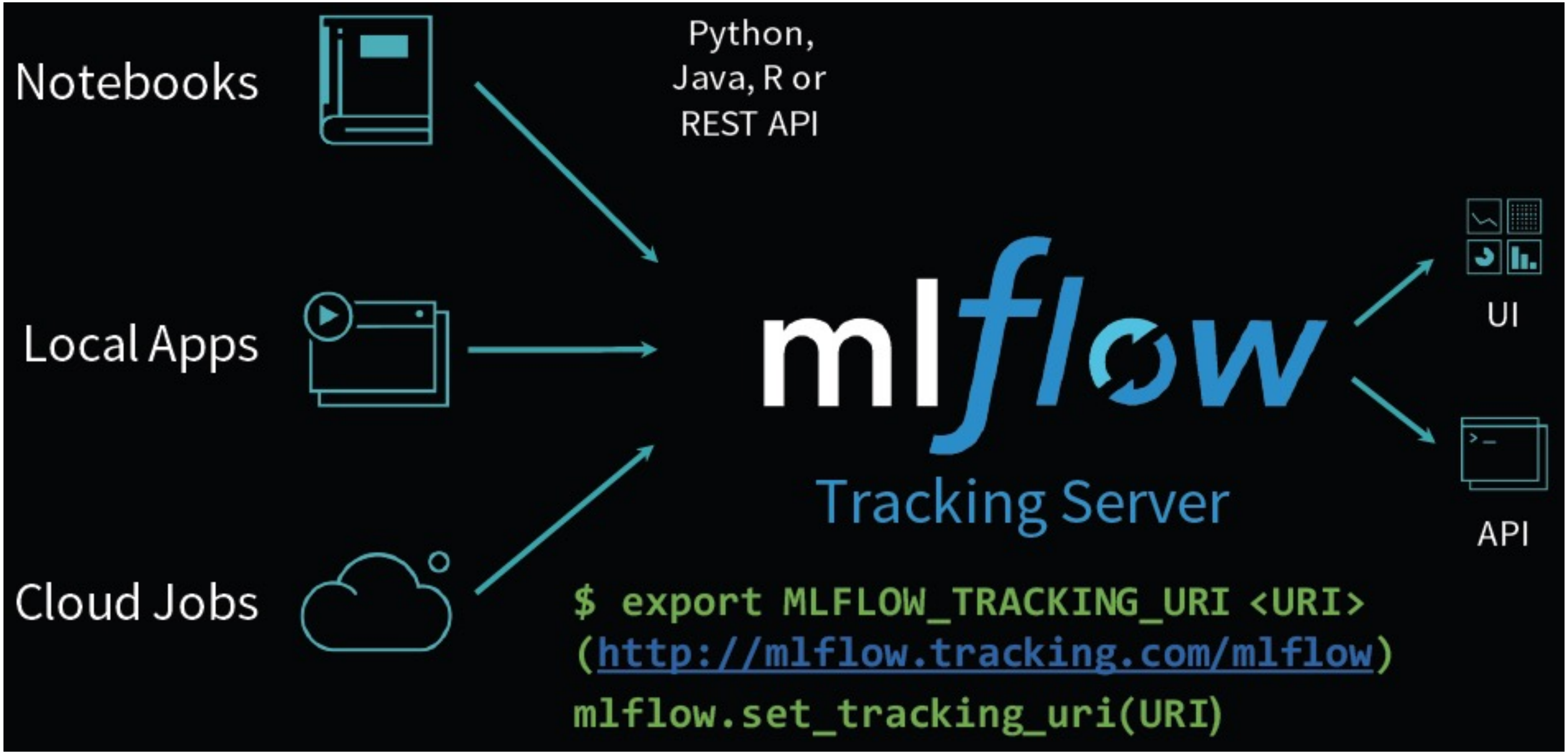
| | | | | | Parameters | | | Metrics | |
|---|---|---|---|---|---|---|---|---|---|
| □ | Date | User | Source | Version | input_file | lr | n | accuracy | f1 ▾ |
| □ | 2018-10-02 21:53:57 | matei | lang_model.py | e55d56 | data.txt | 2.0 | 1 | 0.77 | 0.704 |
| □ | 2018-10-02 21:53:55 | matei | lang_model.py | e55d56 | data.txt | 2.0 | 4 | 0.835 | 0.609 |
| □ | 2018-10-02 21:53:48 | matei | lang_model.py | e55d56 | data.txt | 2.0 | 2 | 0.66 | 0.476 |
| □ | 2018-10-02 21:53:53 | matei | lang_model.py | e55d56 | data.txt | 1.0 | 1 | 0.683 | 0.468 |
| □ | 2018-10-02 21:53:49 | matei | lang_model.py | e55d56 | data.txt | 1.0 | 4 | 0.902 | 0.461 |

Track parameters, metrics, output files & code version

Search using UI or API

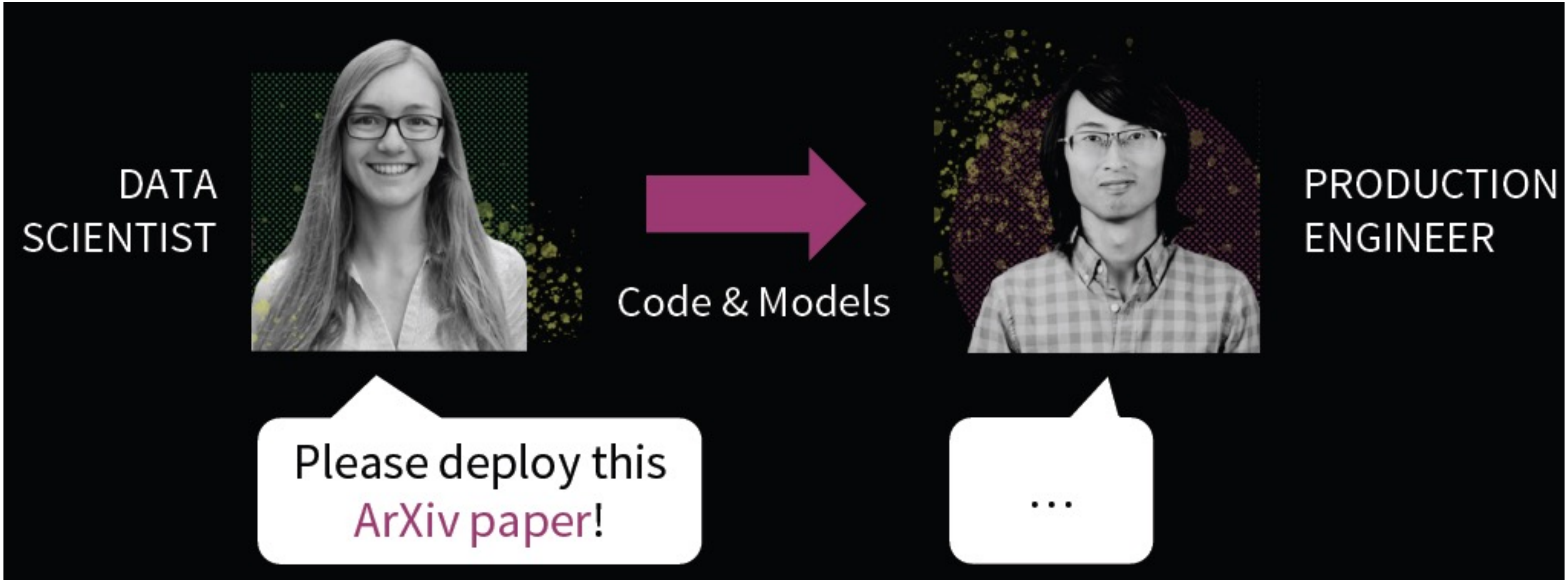
- Data Scientist/ Model developer can track, inspect and compare the results of the running of  different models/ parameters via the MLflow UI

# MLflow Tracking

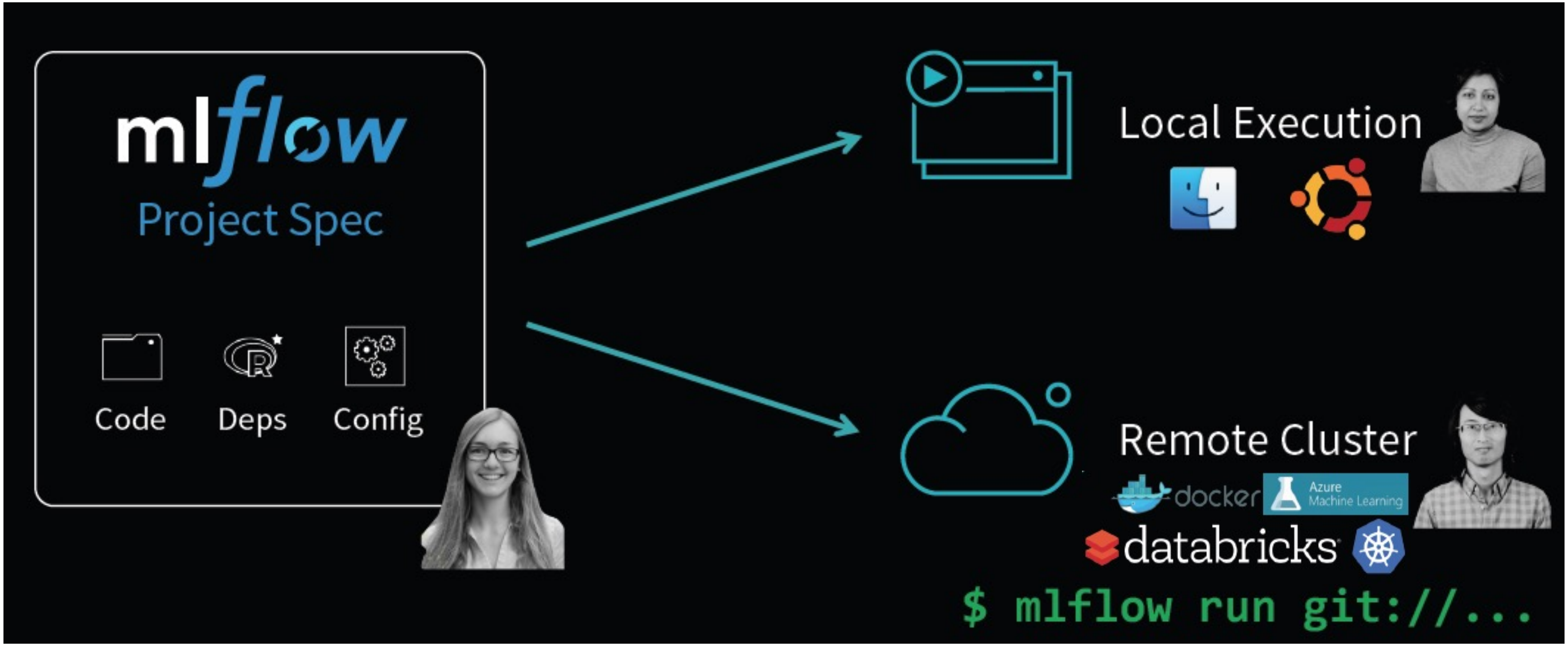


```
$ export MLFLOW_TRACKING_URI <URI>
(http://mlflow.tracking.com/mlflow)
mlflow.set_tracking_uri(URI)
```

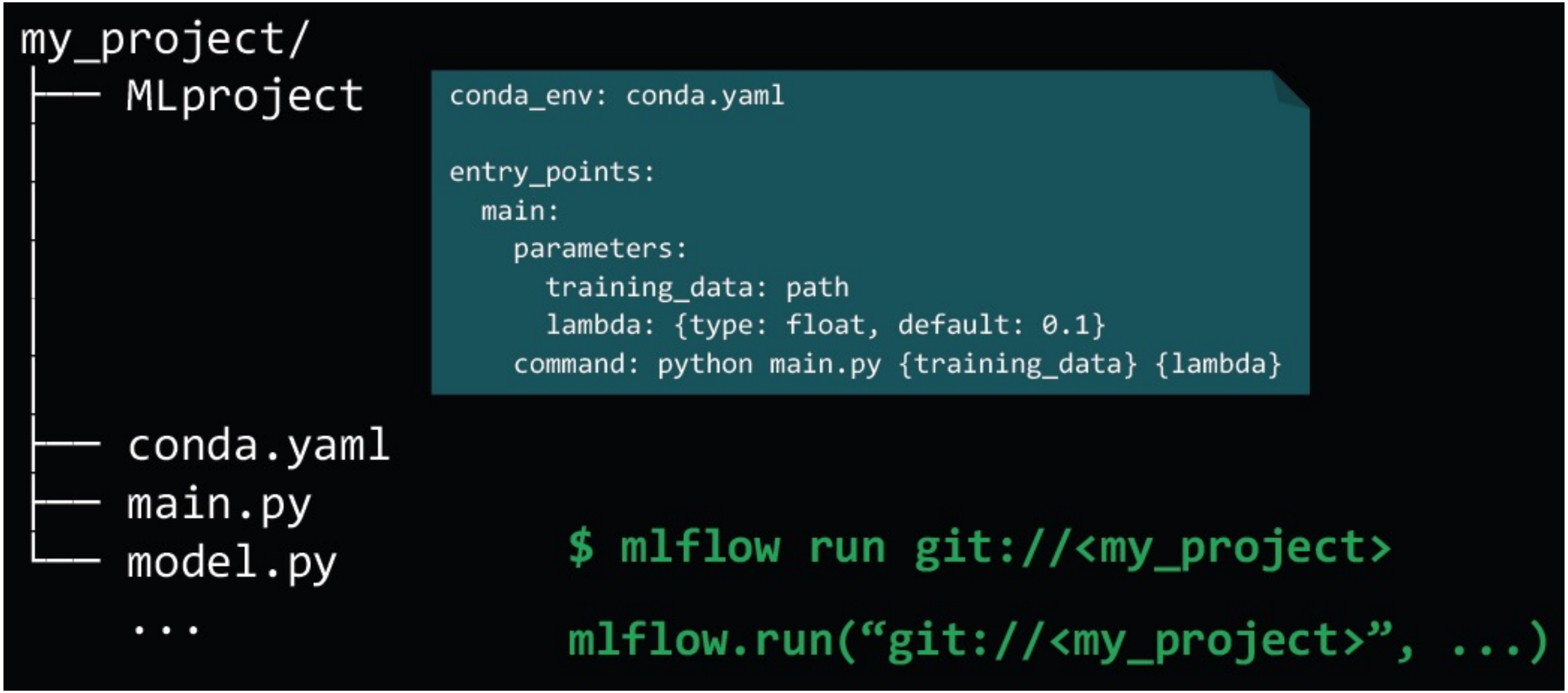
# Model Deployment without MLflow



# Packaging Code: MLflow Projects



# Example MLflow Project

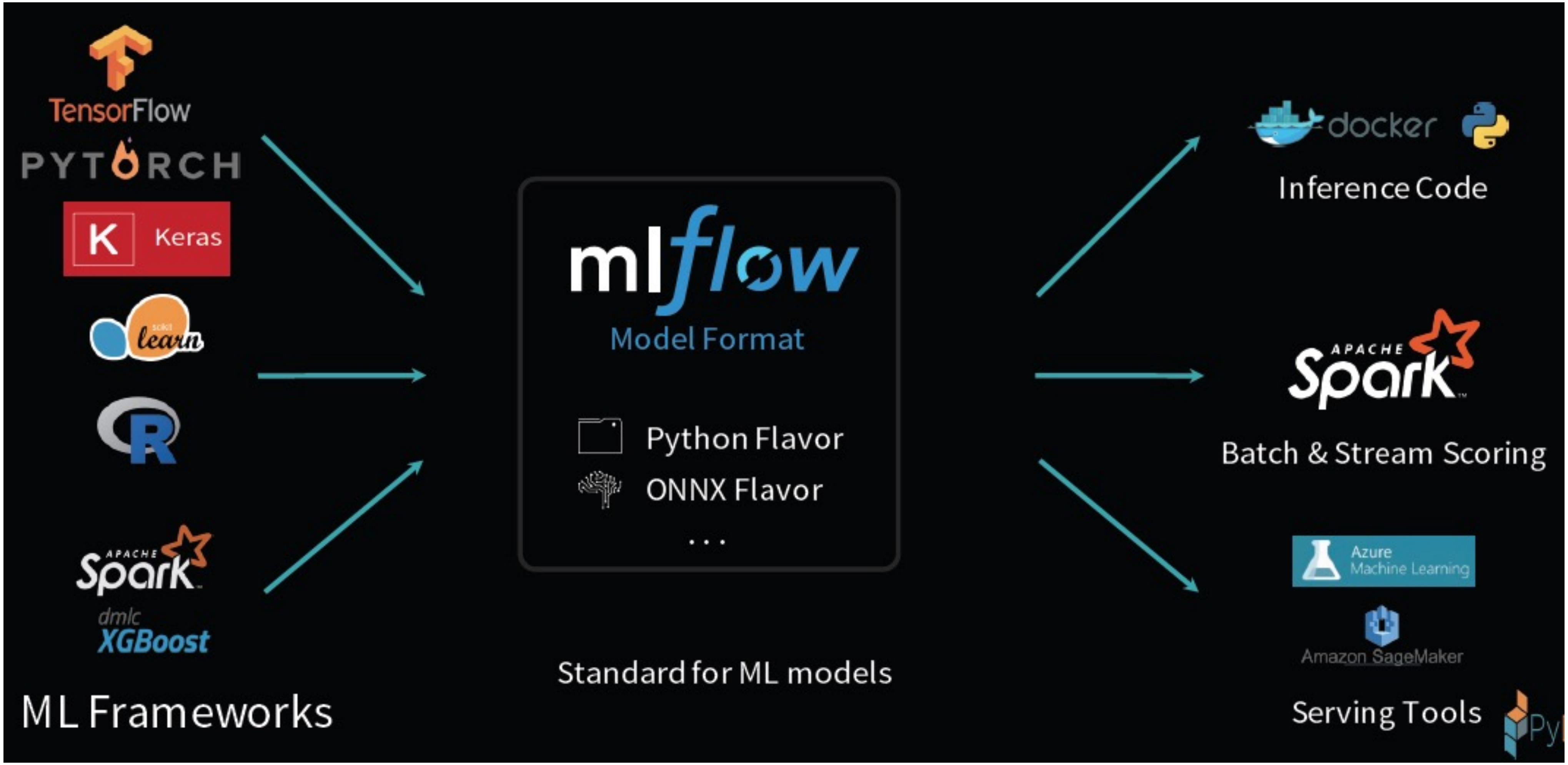


```
my_project/
├── MLproject
│
│
│
│
│
├── conda.yaml
├── main.py
└── model.py
    ...
```

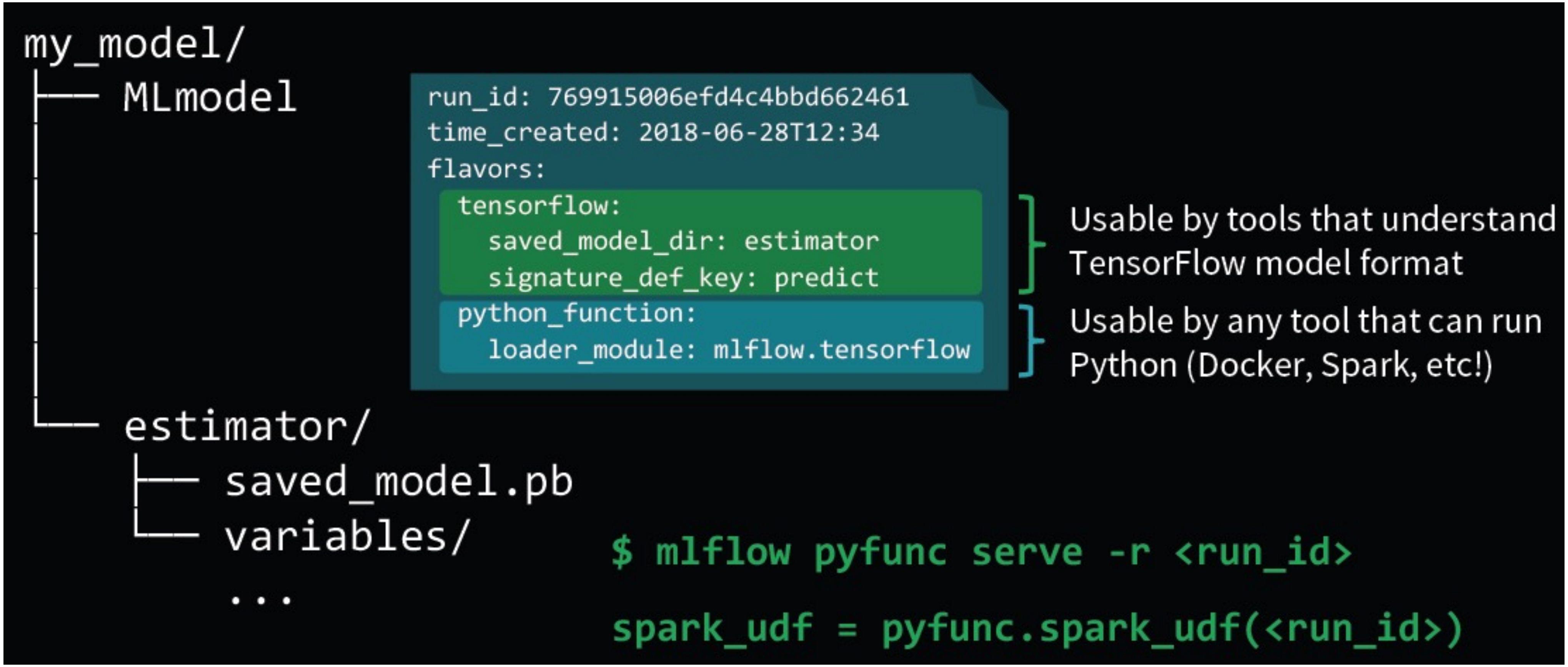
```
conda_env: conda.yaml

entry_points:
  main:
    parameters:
      training_data: path
      lambda: {type: float, default: 0.1}
    command: python main.py {training_data} {lambda}
```

```
$ mlflow run git://<my_project>

mlflow.run(“git://<my_project>”, ...)
```

# Packaging Models: MLflow Models



# Example MLflow Model



```
my_model/
├── MLmodel
│
│
│
│
│
│
└── estimator/
    ├── saved_model.pb
    └── variables/
        ...
```

```
run_id: 769915006efd4c4bbd662461
time_created: 2018-06-28T12:34
flavors:
  tensorflow:
    saved_model_dir: estimator
    signature_def_key: predict
  python_function:
    loader_module: mlflow.tensorflow
```

Usable by tools that understand TensorFlow model format

Usable by any tool that can run Python (Docker, Spark, etc!)

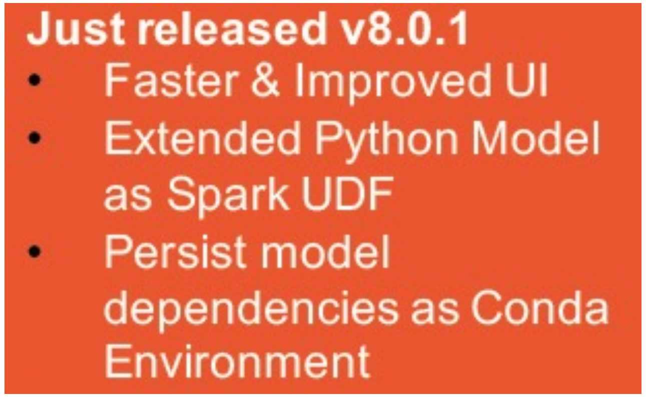
```
$ mlflow pyfunc serve -r <run_id>

spark_udf = pyfunc.spark_udf(<run_id>)
```

# Model Deployment with MLflow

DATA SCIENTIST
PRODUCTION ENGINEER
Please run this MLflow Project nightly for updates!
Don't even tell me what ArXiv paper that's from...

# Ongoing MLflow Roadmap (circa Jan 2019)

- Tensorflow, Keras, PyTorch, H2O, MLleap, MLlib integrations
- Java and R MLflow Client language APIs
- Multi-step Workflows
- Hyperparameter Tuning
- Integration with Databricks Tracking Server
- Support for Data Store (e.g. MySQL)
- Stablize MLflow APIs 1.0
- Model metadata, management and registry
- Hosted MLflow



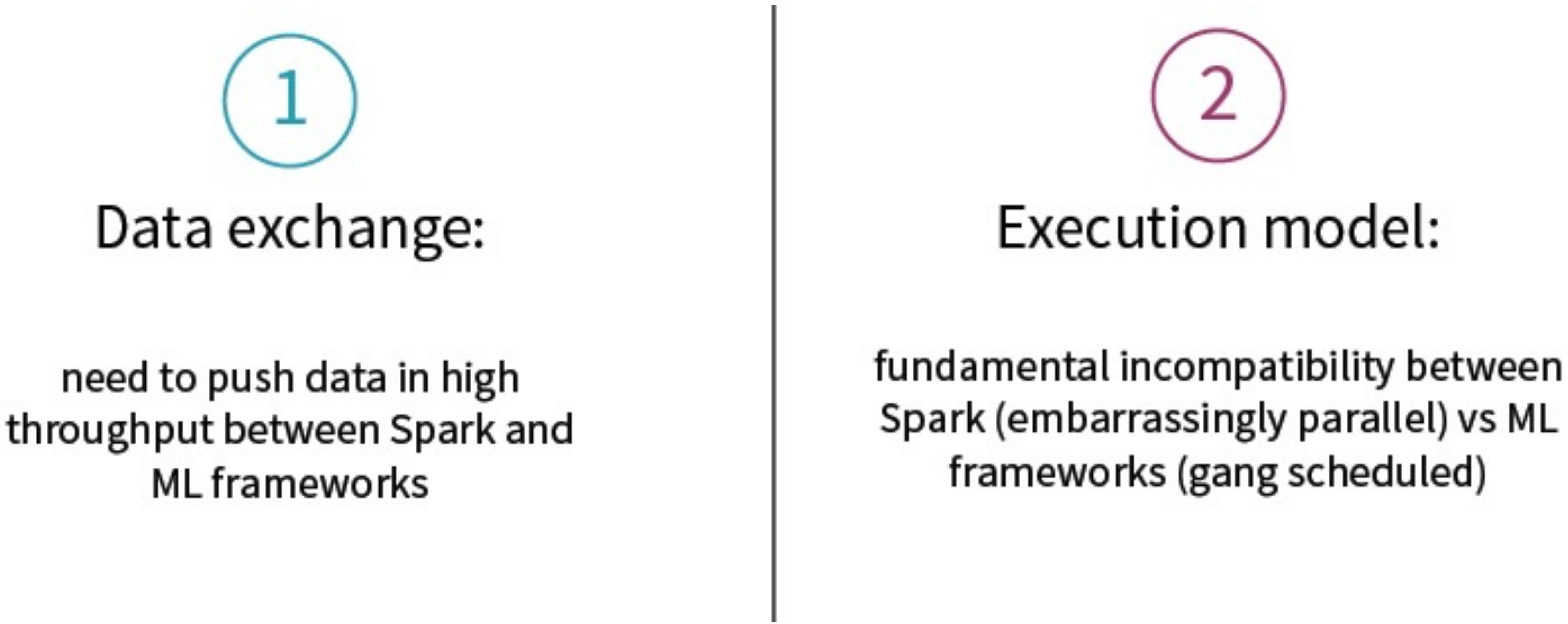
**Just released v8.0.1**

- Faster & Improved UI
- Extended Python Model as Spark UDF
- Persist model dependencies as Conda Environment

# Project Hydrogen:

## Better Integration of other ML/ AI frameworks with Spark

# Two Challenges in supporting ML frameworks in Spark



1

Data exchange:

need to push data in high throughput between Spark and ML frameworks

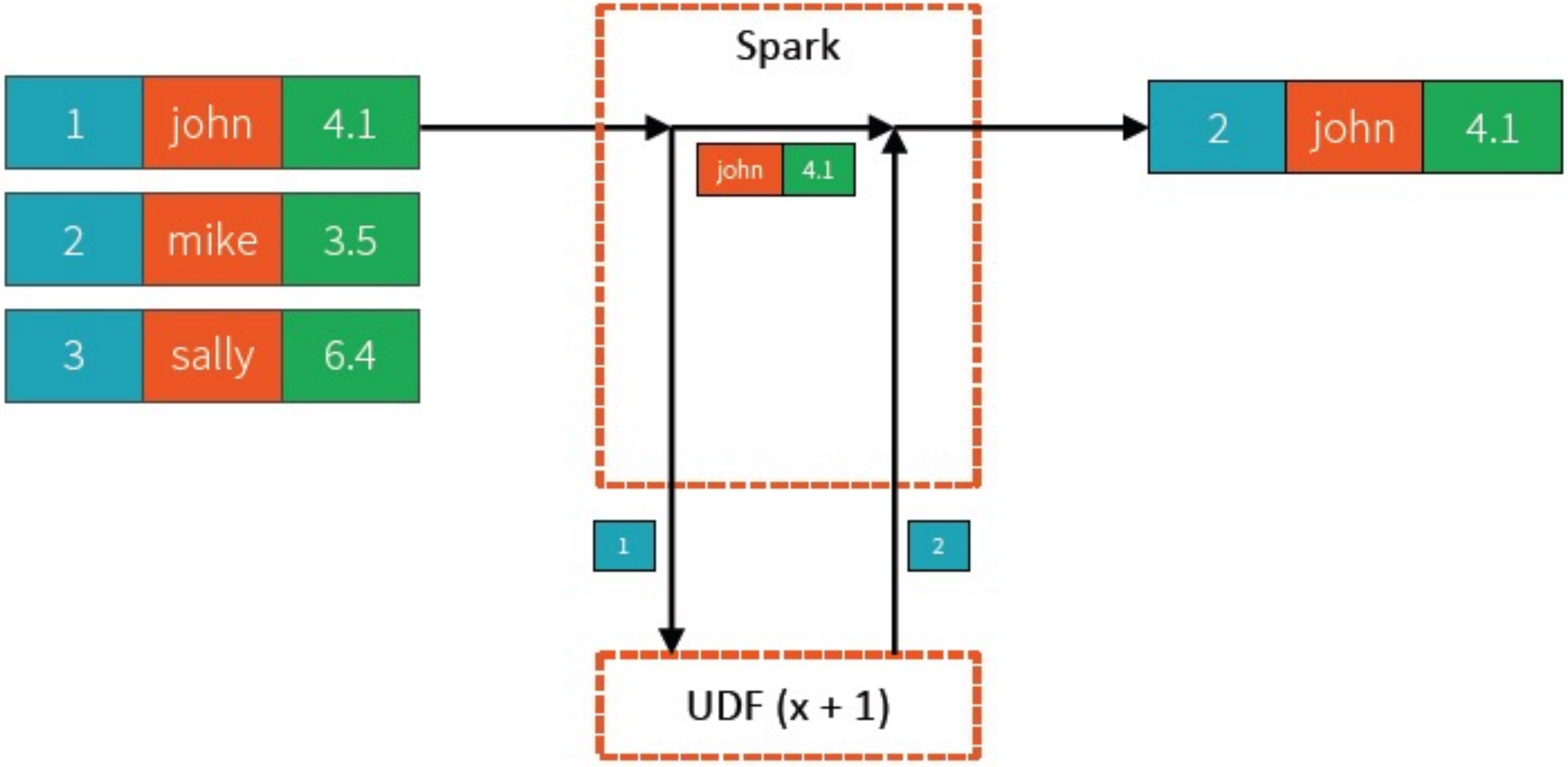
2

Execution model:

fundamental incompatibility between Spark (embarrassingly parallel) vs ML frameworks (gang scheduled)

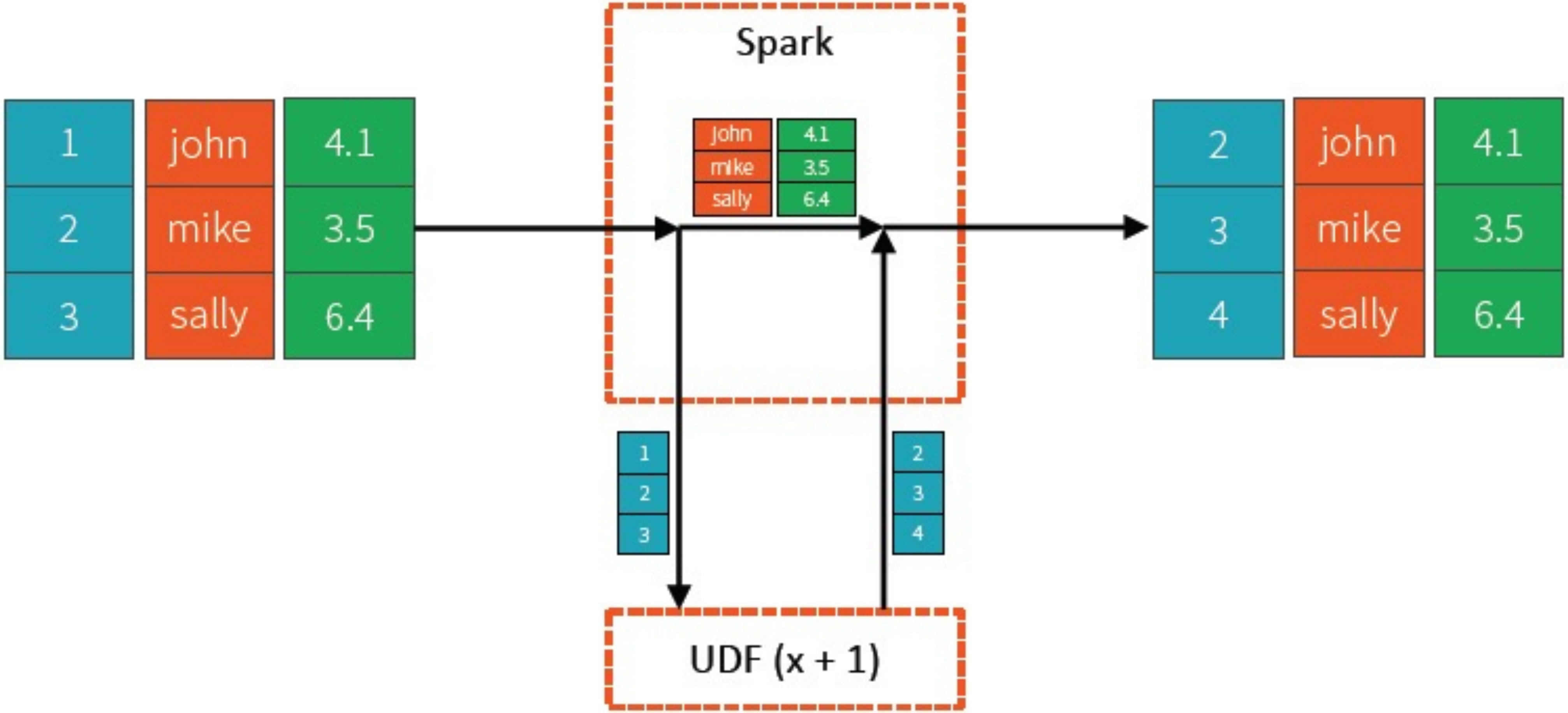
# User Defined Functions (UDFs)

- UDFs allow the execution of arbitrary code ; often used for integration with ML frameworks
  - e.g., Prediction on data using Tensorflow
- But Exchanging data with UDFs only is carried out only One-Row-at-a-Time => Waste CPU cycles



# Introducing "Vectorized Data Exchange"

- UDFs run 3x to 240x faster !

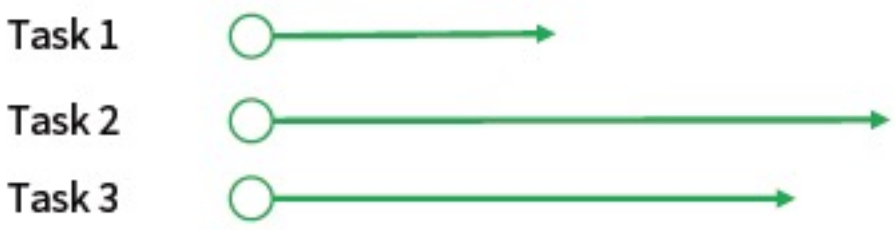


# Execution Models

## Spark

Tasks are independent of each other

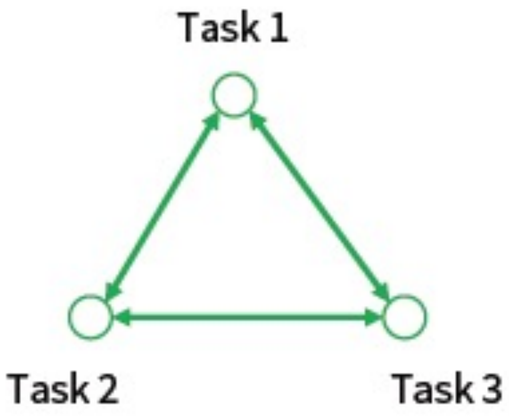
Embarrassingly parallel & massively scalable



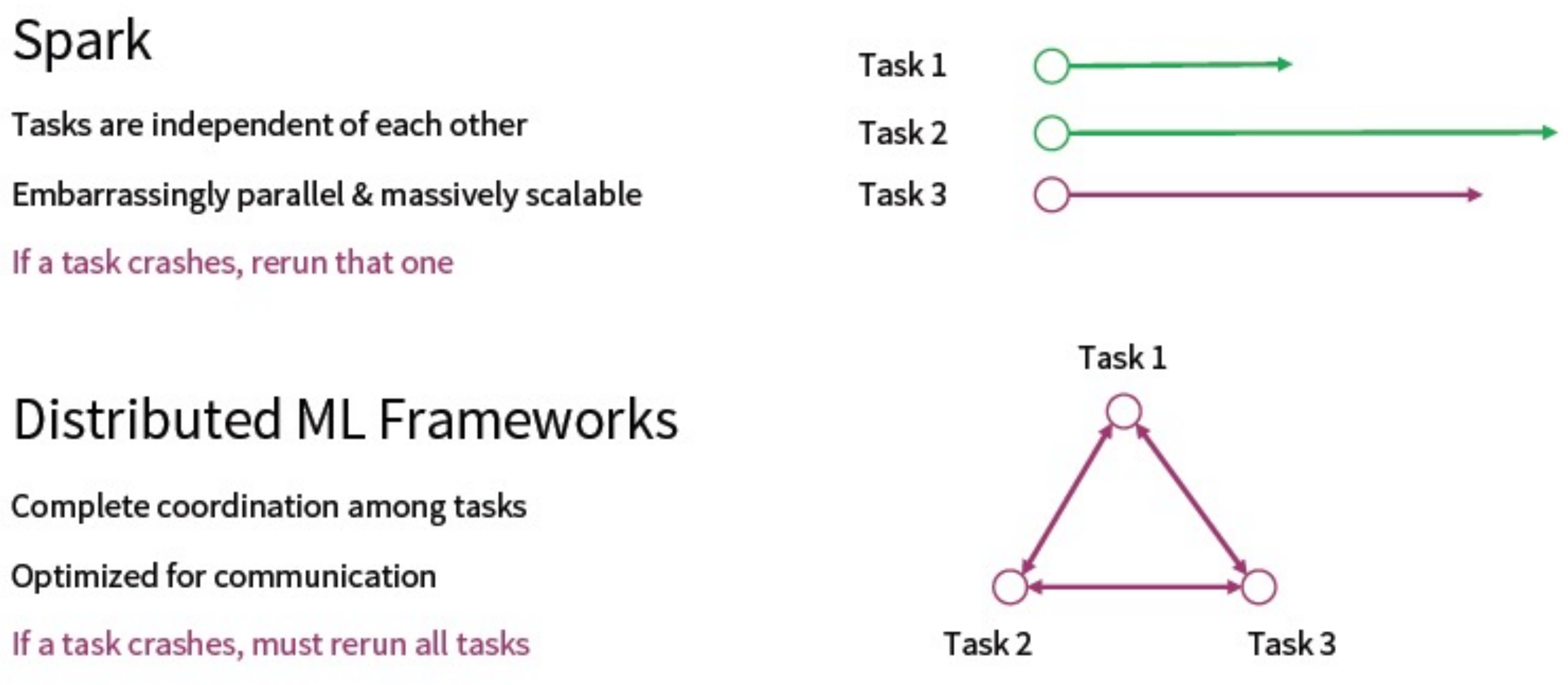
## Distributed ML Frameworks

Complete coordination among tasks

Optimized for communication



# What if a Task Crashes ?



## Spark

Tasks are independent of each other

Embarrassingly parallel & massively scalable

If a task crashes, rerun that one

## Distributed ML Frameworks

Complete coordination among tasks

Optimized for communication

If a task crashes, must rerun all tasks

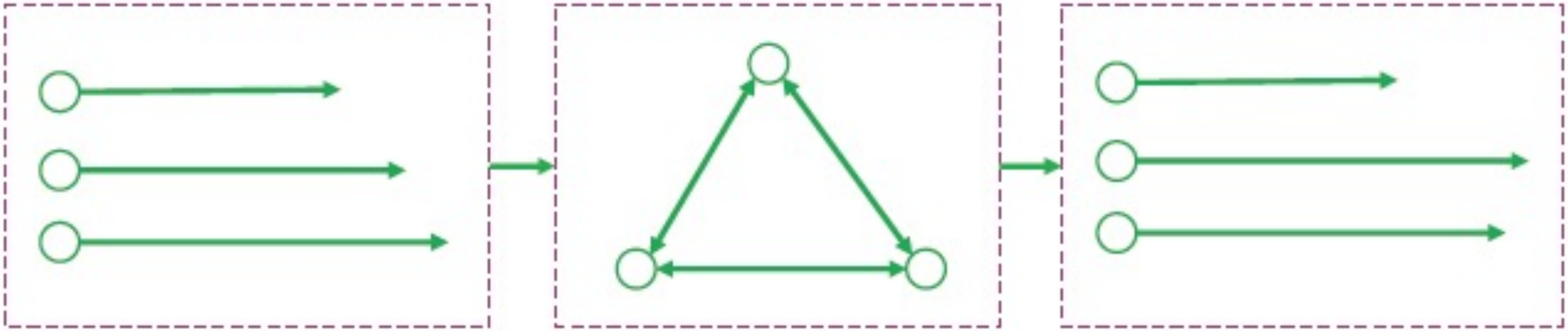
=> Incompatible Execution models !

# Unifying Execution Models with Barriers Execution (aka Gang Execution)

Stage 1
data prep
embarrassingly parallel

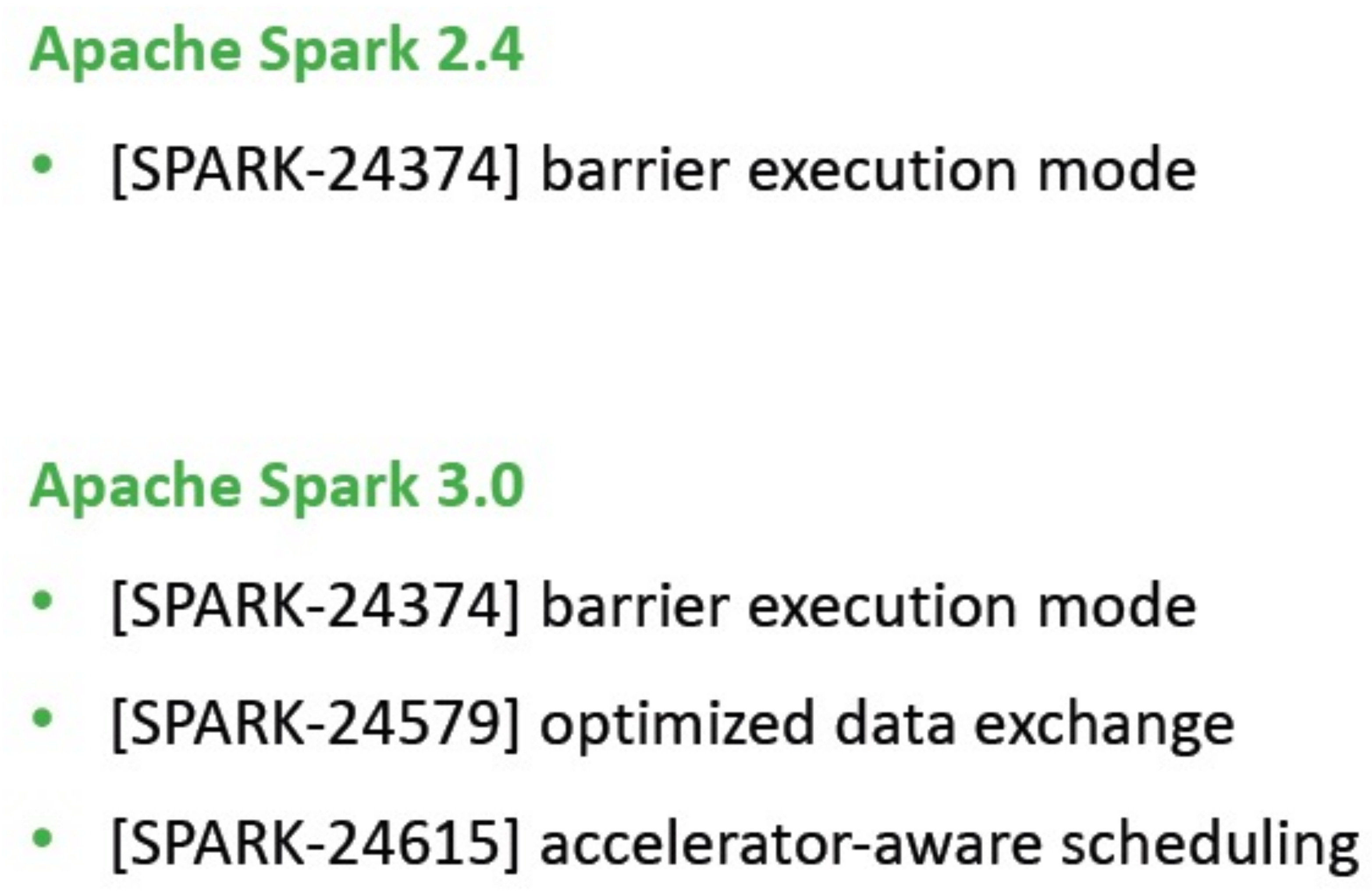
Stage 2
distributed ML training
gang scheduled

Stage 3
data sink
embarrassingly parallel



tasks "all or nothing"
to reconcile fundamental incompatibility
between Spark and distributed ML frameworks

# Roadmap to support
# Barrier Execution (aka Gang Execution)



## Apache Spark 2.4

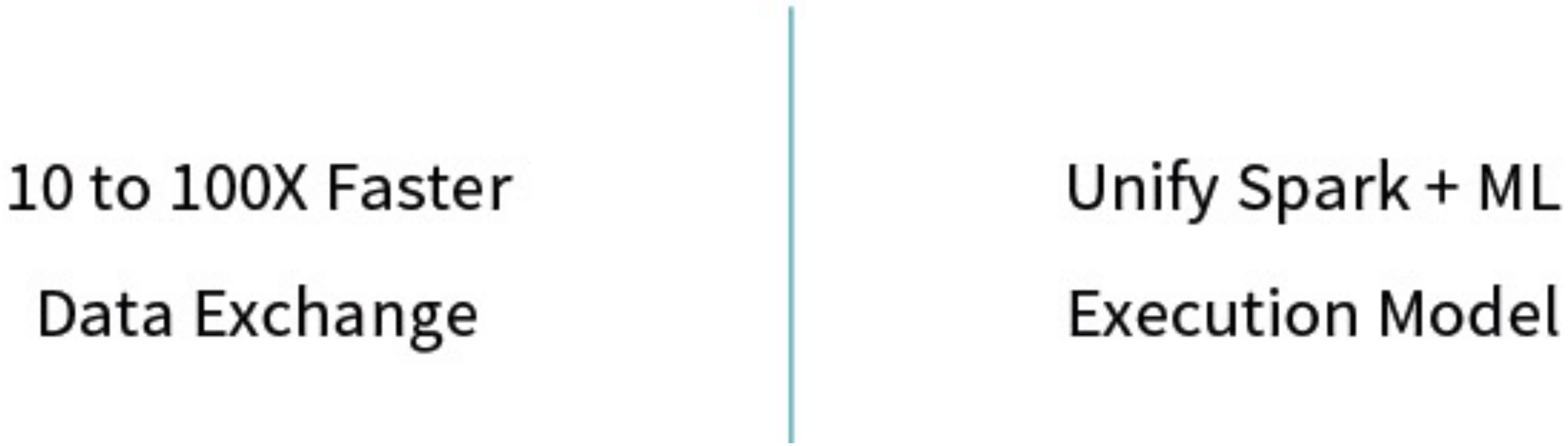
- [SPARK-24374] barrier execution mode

## Apache Spark 3.0

- [SPARK-24374] barrier execution mode
- [SPARK-24579] optimized data exchange
- [SPARK-24615] accelerator-aware scheduling

See also Project Hydrogen: Unifying State-of-the-art AI and Big Data in Apache Spark by Reynold Xin
See also Project Hydrogen: State-of-the-Art Deep Learning on Apache Spark by Xiangrui Meng

# Project Hydrogen



10 to 100X Faster

Data Exchange

Unify Spark + ML

Execution Model

Timeline

- Spark 2.3 (Spring 2018): Basic Vectorized UDFs
- Spark 2.4 (Fall 2018): Barrier Scheduler and more Vectorized UDFs support
- Spark 3.0 (2019): General Availability (GA) and standard format for data

See also Project Hydrogen: Unifying State-of-the-art AI and Big Data in Apache Spark by Reynold Xin
See also Project Hydrogen: State-of-the-Art Deep Learning on Apache Spark by Xiangrui Meng

# Summary of Key Efforts in Spark 2.X (ver2.4 circa Nov 2018)

- Structured Streaming
  - Unification of the APIs
  - Event-time Aggregations/ Processing to handle out-of-order/late data
  - Other Streaming sources/sinks
  - Support Structured Streaming in other libraries, e.g. MLlib, GraphFrames
  - Support of Continuous Processing model, i.e. true (low-latency) streaming instead of stream processing via micro-batching.
  - Spark over Kubernetes: deploying Spark not only as a framework but also as a containerized distributed application/ library !
  - Machine Learning – Optimized Model Tuning
- Iteration as a First-Class concept in DataFrames
- Cost-based Query Optimization for ML/Graph Algorithms
  - Caching, Communication, Serialization, Compression
- Spark + GPUs
- High-level API for Deep-Learning Pipeline in Spark MLlib
  - Built on TensorFlow, Keras, BigDL
- Project Hydrogen - enhancing Integration of other ML frameworks with Spark
- Better Infrastructure support of Production-level Complete ML Life-cycle with MLflow

# Summary of Key New Features in Spark 2.4.x (part of Databricks Runtime 5.2 ML) (circa Jan 2019)

- Using HorovodRunner for Distributed Deep Learning Training
  - Integrating Horovod with Spark’s Barrier mode
  - Simplified workflow for multi-GPU machines

  https://docs.databricks.com/applications/machine-learning/train-model/distributed-training/horovod-runner.html

  https://databricks.com/session/distributed-deep-learning-with-apache-spark-and-tensorflow

- GraphFrames to add a Pregel-like API

- Enhance Databricks Runtime support for TensorBoard (visualization toolkit for Tensorflow)

- Speed-up Cluster start-time when Pytorch is included.

https://databricks.com/blog/2019/01/30/databricks-runtime-5-2-ml-features-multi-gpu-workflow-pregel-api-and-performant-graphframes.html

# Summary of Key New Features in Spark 3.0 (part of Databricks Runtime 7.0) (circa June 2020)

- ANSI SQL Compliance
- 2x performance improvement on TPC-DS (SQL benchmark) over Spark 2.4 by Adaptive Query execution, Dynamic Partition Pruning and other optimizations.
- New UI for Structured Streaming
- Improvements in Pandas APIs
- Better Python error Handling
- Speed-up in calling R UDF (upto 40x)

https://databricks.com/blog/2020/06/18/introducing-apache-spark-3-0-now-available-in-databricks-runtime-7-0.html

# Summary of
# Key New Features in Spark 3.1
# (part of Databricks Runtime 8.0)
# (circa March 2021)

- ANSI SQL Compliance
- More Query Optimization
- Shuffle Hash Join improvements
- History Server support of Structured Streaming
- Project Zen has been initiated to:
  - Provide Better Interoperability with other Python libraries
  - Improve PySpark’s Usability

https://databricks.com/blog/2021/03/02/introducing-apache-spark-3-1.html

# Summary of Key New Features in Spark 3.2 (part of Databricks Runtime 10.0) (circa Oct 2021)

- Pandas API layer on PySpark – (from Project Zen)
  - Also provide Interactive Data visualization
- ANSI SQL Compliance - ANSI Mode GA
- Productionize Adaptive Query Execution to speedup Spark SQL at runtime
- Introduce RockDB State-store to enable scalable state processing
- Event-time based Session Window support
- Support push-based Shuffle

https://databricks.com/blog/2021/10/19/introducing-apache-spark-3-2.html

https://spark.apache.org/releases/spark-release-3-2-0.html

https://spark.apache.org/third-party-projects.html