Lecture 2: Spark DataFrame, Dataset, and ML Pipelines

Haiping Lu

http://www.dcs.shef.ac.uk/~haiping

COM6012 Scalable Machine Learning

Spring 2018

Week 2 Contents

- Spark Recap – Example on Cache
- Spark DataFrame & Dataset
- Spark MLlib & ML Pipelines
- GitHub Classroom & Quiz 1

Week 2 Contents

- **Spark Recap – Example on Cache**
- Spark DataFrame & Dataset
- Spark MLlib & ML Pipelines
- GitHub Classroom & Quiz 1

Apache Spark

- Fast and general cluster computing system, interoperable with Hadoop, included in all major distros
- Improves efficiency through:
  - In-memory computing primitives
  - General computation graphs
- Improves usability through:
  - Rich APIs in Scala, Java, Python
  - Interactive shell

→ Up to 100× faster (2-10× on disk)

→ 2-5× less code

Spark Model

- *Write programs in terms of transformations on distributed datasets*

- Resilient Distributed Datasets (RDDs)
  - Collections of objects that can be stored in memory or disk across a cluster
  - Parallel functional transformations (map, filter, …)
  - Automatically rebuilt on failure

Spark for Data Science

- DataFrames
  - Structured data
  - Familiar API based on R & Python Pandas
  - Distributed, optimized implementation

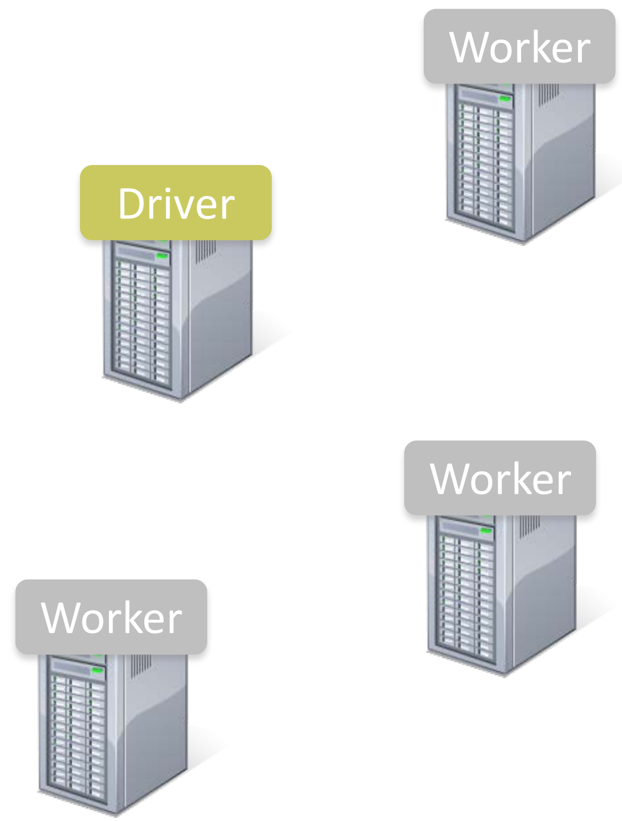
- Machine Learning Pipelines
  - Simple construction and tuning of ML workflows

Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

Spark Example: Log Mining

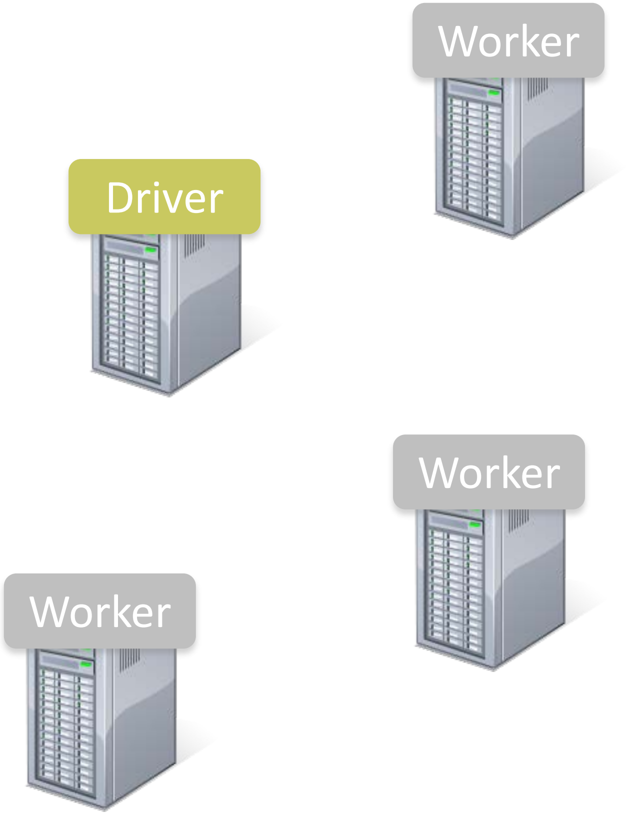
Load error messages from a log into memory, then interactively search for various patterns



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
```

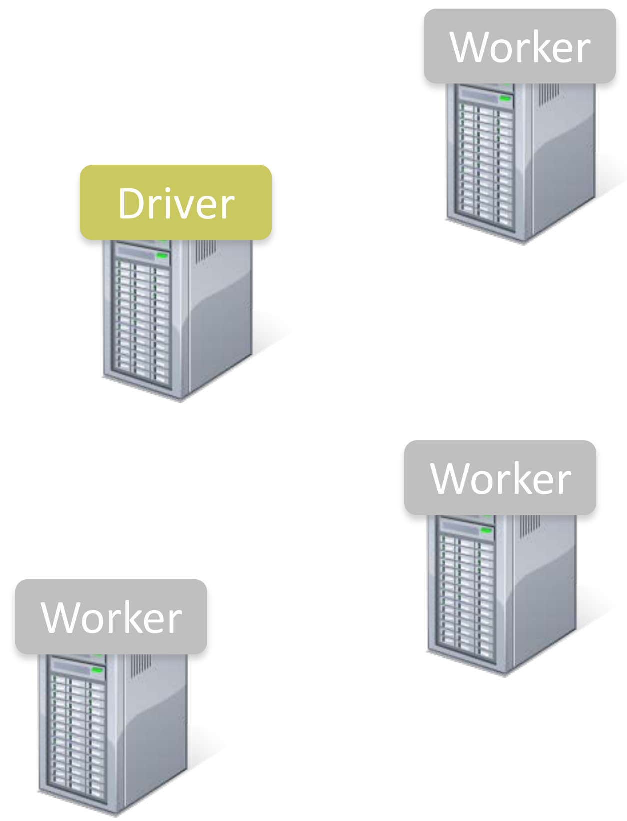


Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

Base RDD

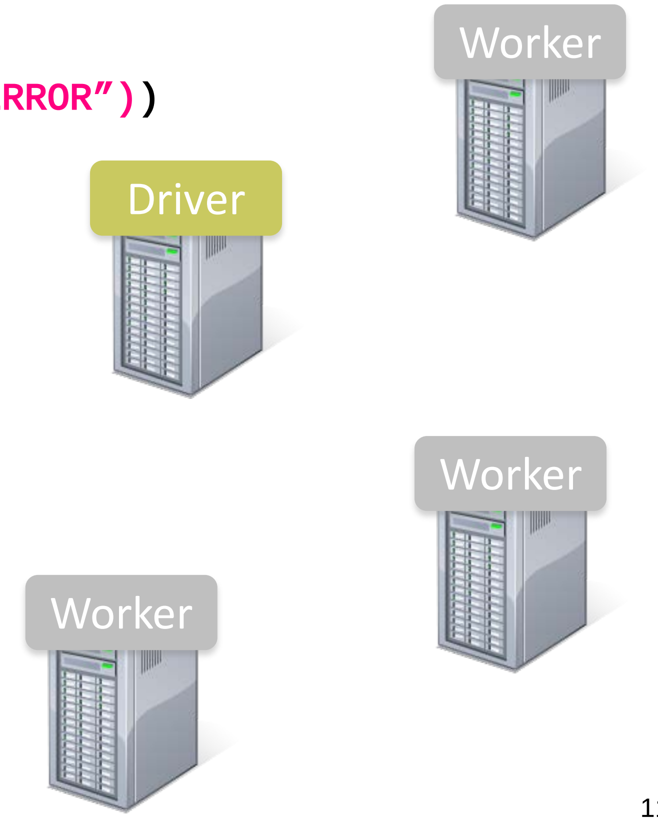
```
lines = spark.textFile("hdfs://...")
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
```

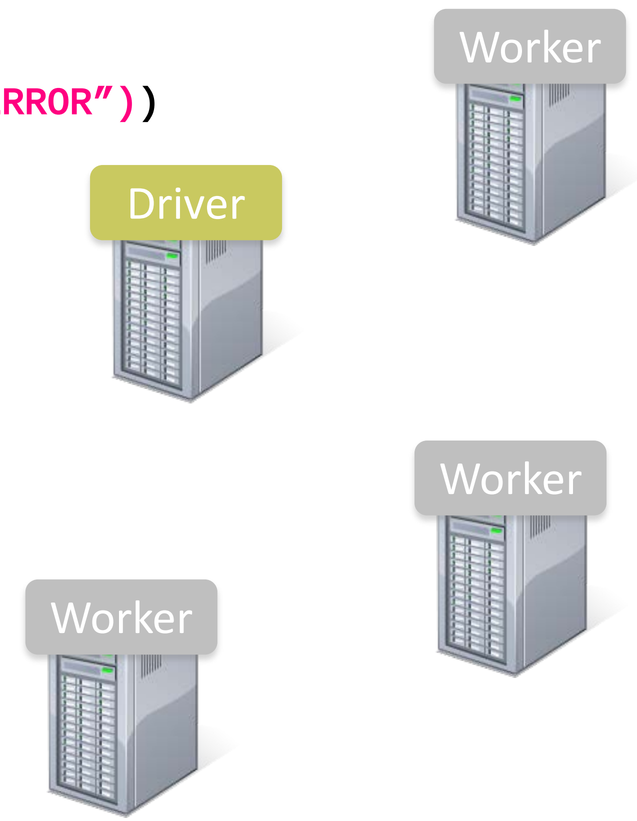


Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

Transformed RDD

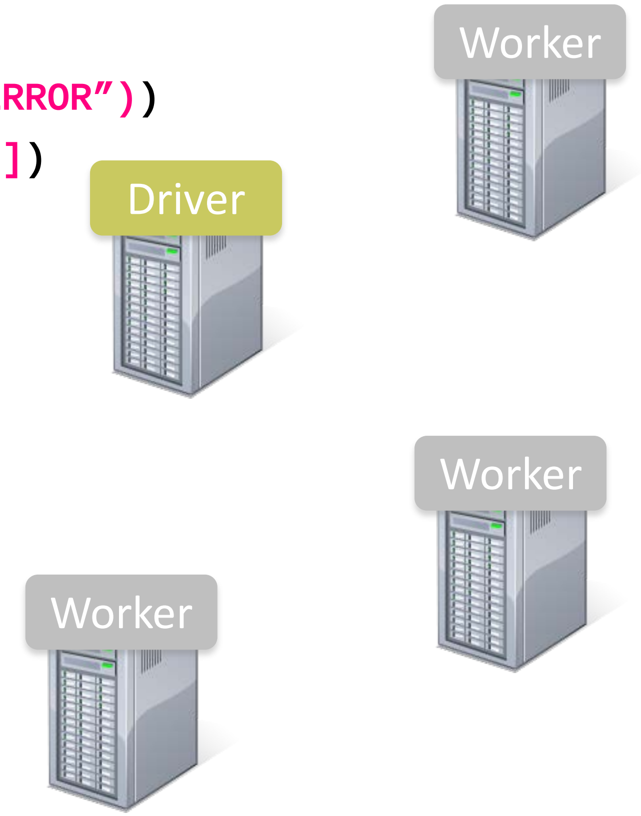
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

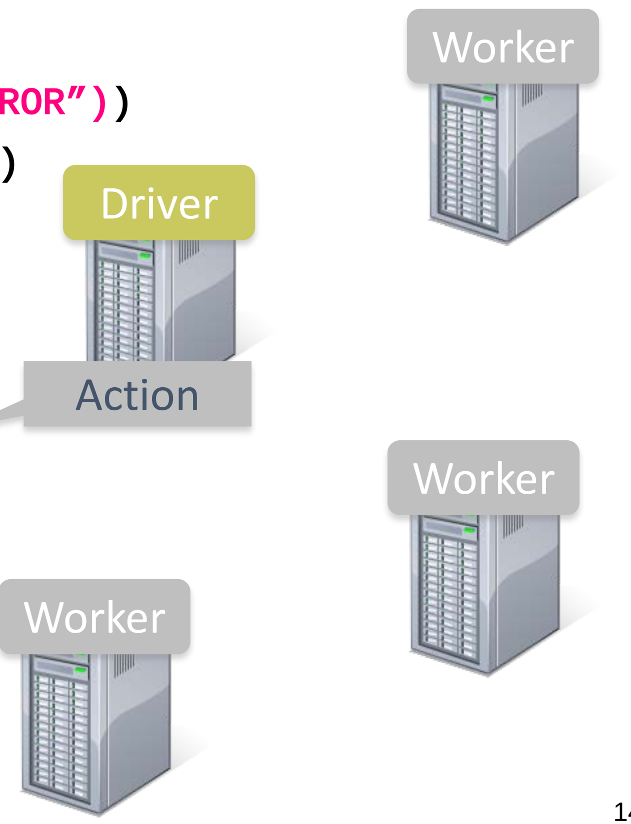
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

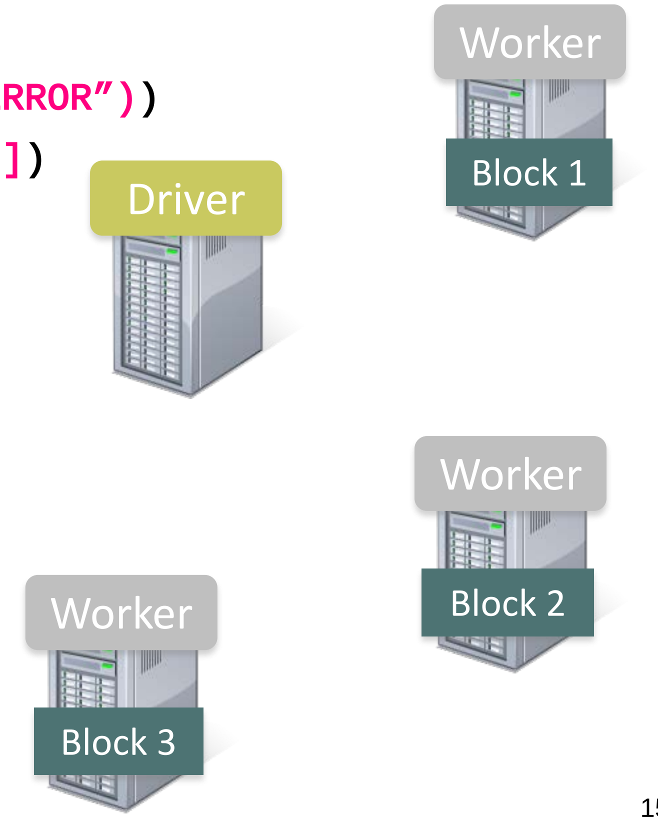
```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()


messages.filter(lambda s: “mysql” in s).count()
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

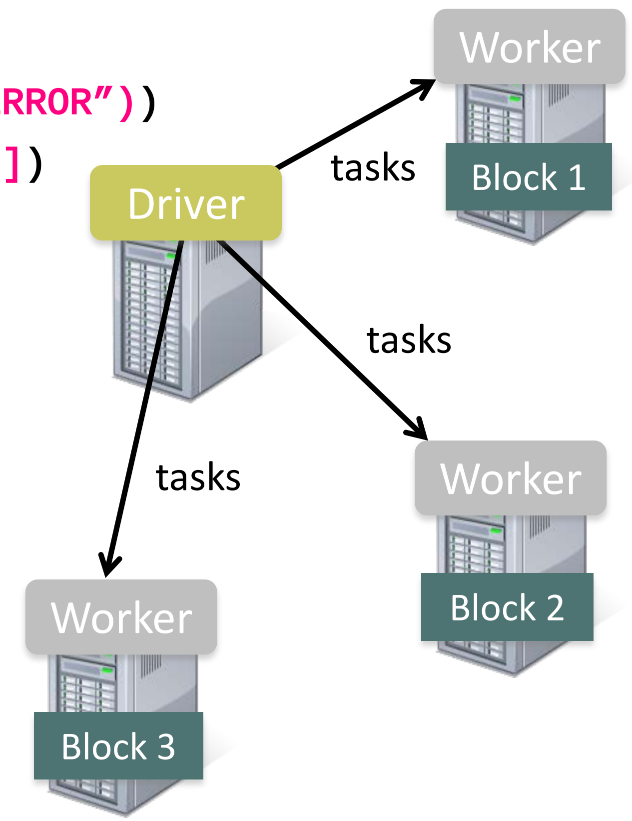
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

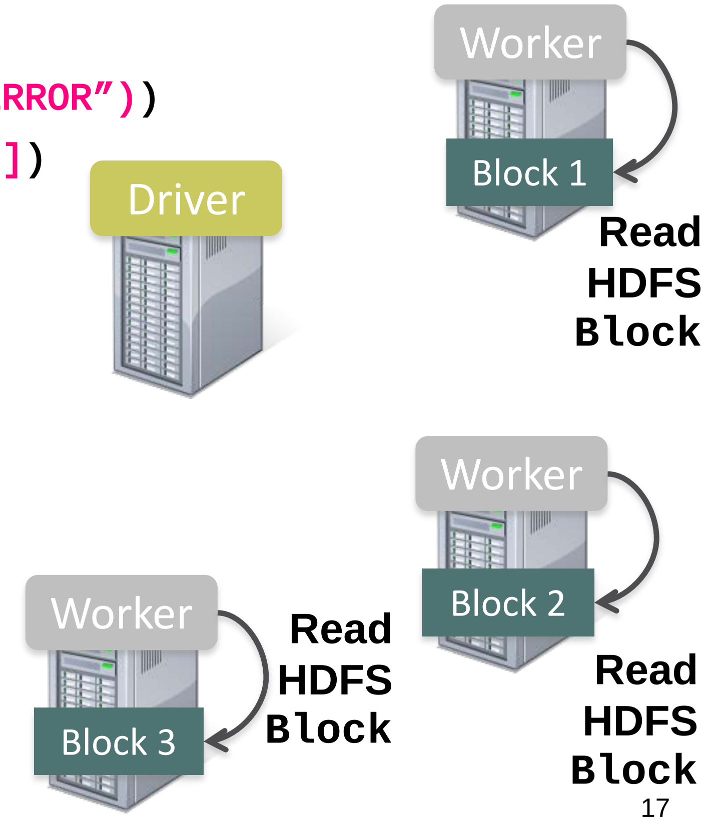
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

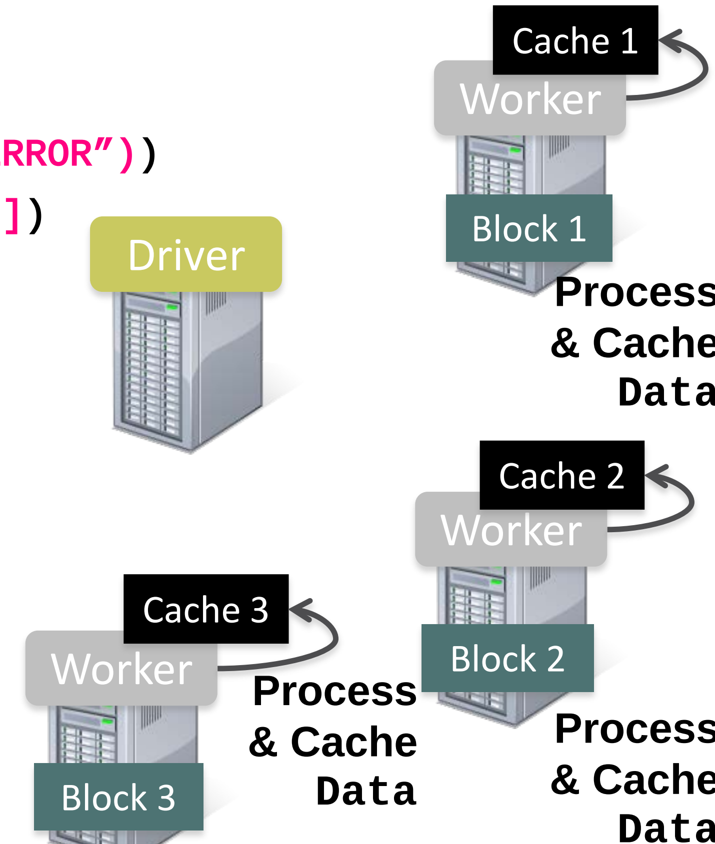
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

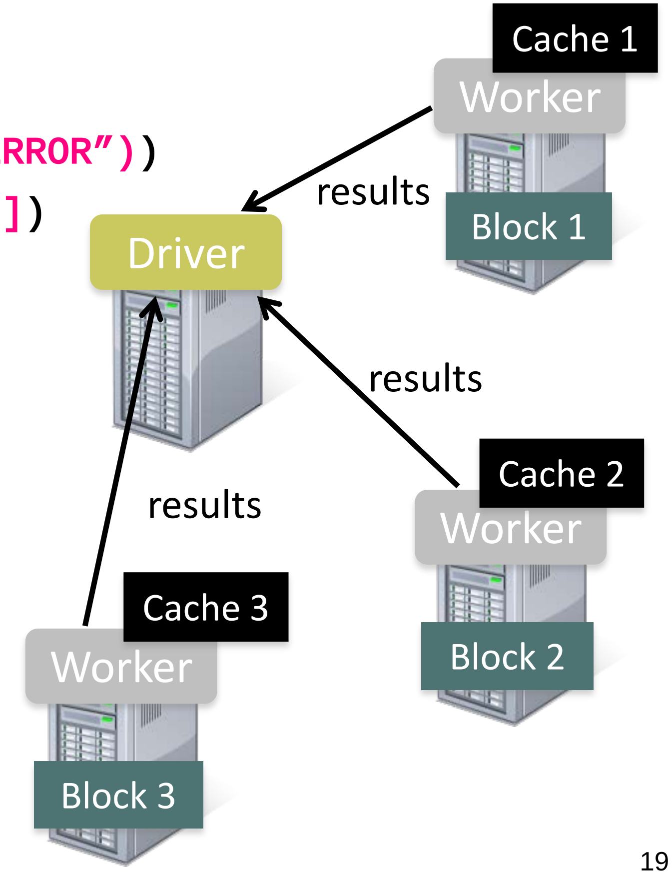
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

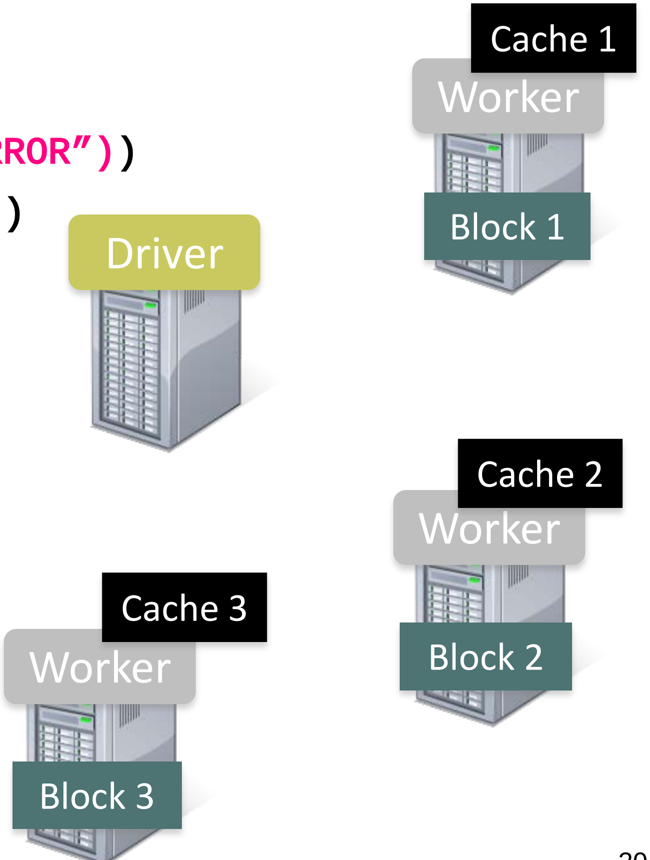
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

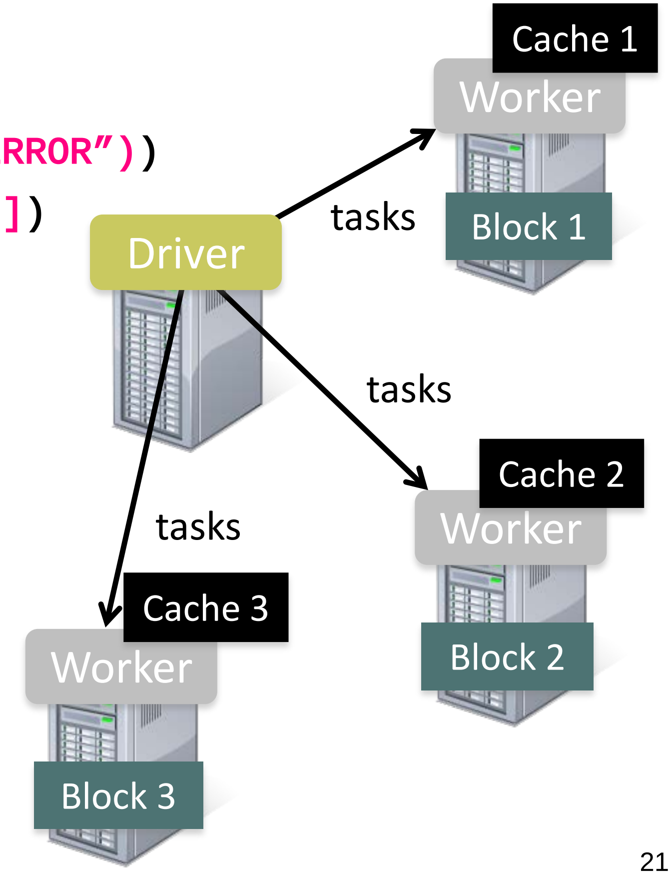
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

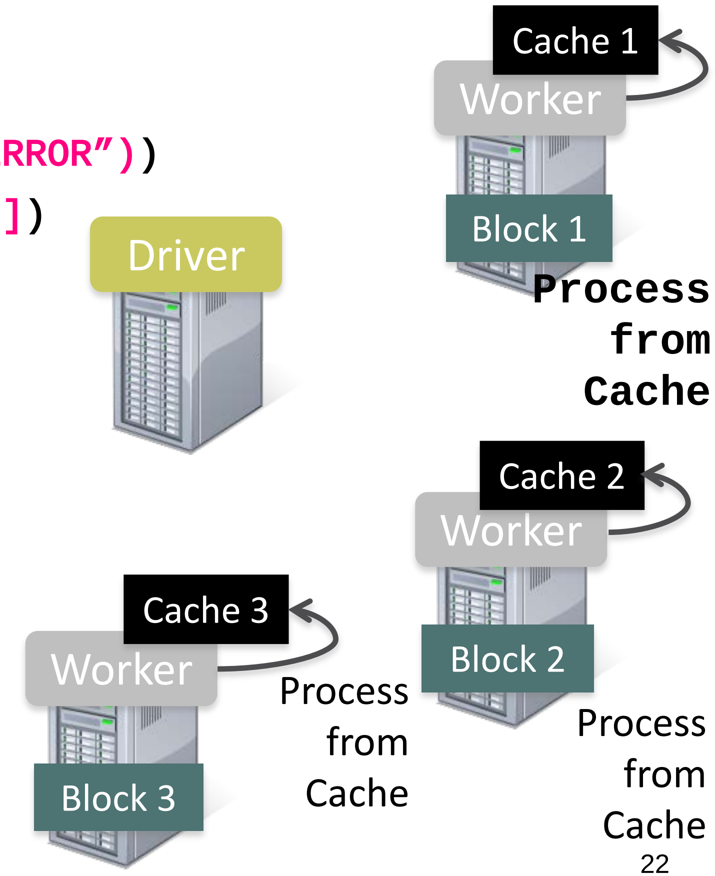
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

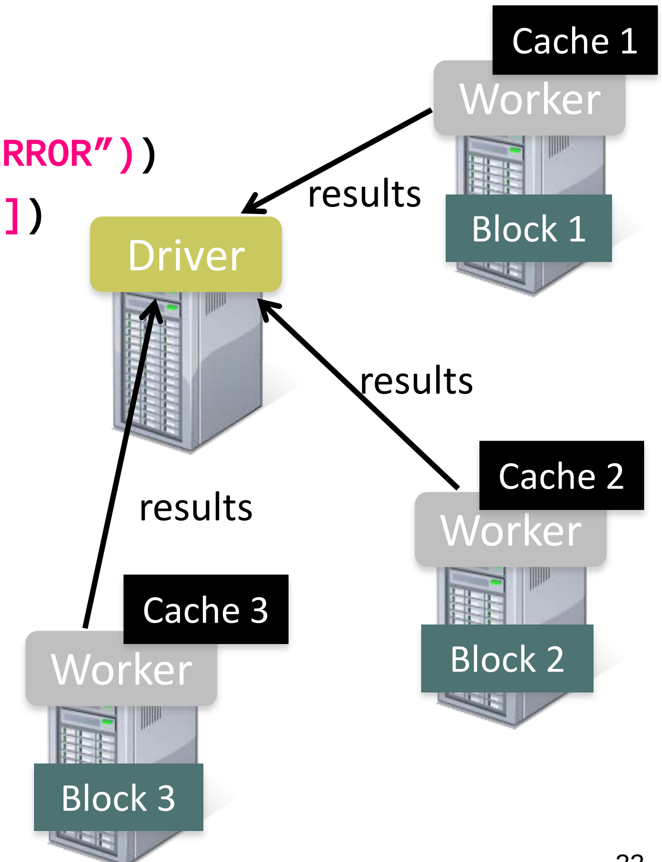
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

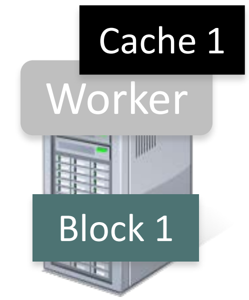
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

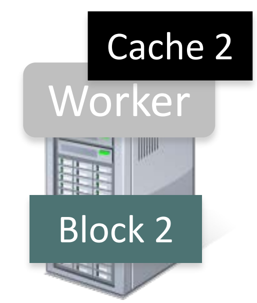


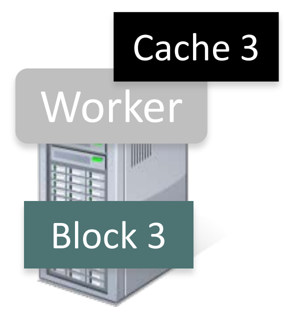
Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```







Cache your data ➔ Faster Results

Full-text search of Wikipedia

- 60GB on 20 EC2 machines
- 0.5 sec from mem vs. 20s for on-disk

Week 2 Contents

- Spark Recap – Example on Cache
- **Spark DataFrame & Dataset**
- Spark MLlib & ML Pipelines
- GitHub Classroom & Quiz 1

Challenges & Solutions

- Perform ETL to and from various (semi- or unstructured) data sources
- Perform advanced analytics (e.g. machine learning, graph processing) that are hard to express in relational systems

- A *DataFrame* API that can perform relational operations on both external data sources and Spark's built-in RDDs.
- A highly extensible optimizer, *Catalyst*, that uses features of Scala to add composable rule, control code gen., and define extensions.

DataFrame-based API for MLlib

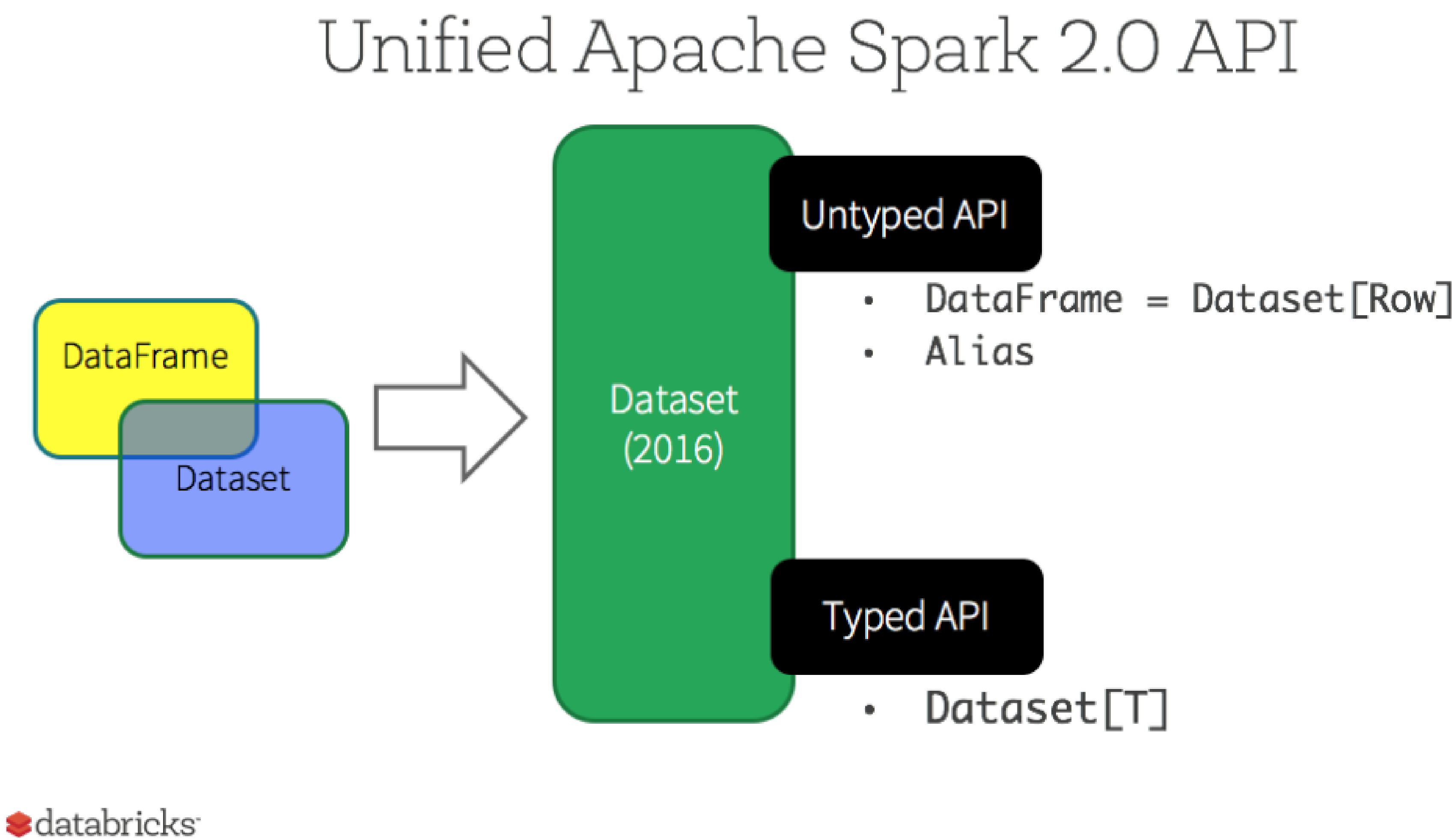
- a.k.a. “Pipelines” API, with utilities for constructing ML Pipelines
- In 2.0, the DataFrame-based API will become the primary API for MLlib
  - Voted by community
  - org.apache.spark.**ml**, pyspark.**ml**
- The RDD-based API will entermaintenance mode
  - Still maintained with bug fixes, but no new features
  - org.apache.spark.**mllib**, pyspark.**mllib**

Structuring Spark: DataFrames and Datasets

- DataFrames and Datasets
  - DataFrame (schema, generic untyped)
    - Index access, named columns (like a table)
  - Dataset (static typing, strongly-typed)
    - Object access
  - DataFrame = Dataset[Row] (row: generic untyped)
  - Unified in Apache Spark 2.0
- RDDs are low-level (like assembler), DataFrames & Datasets are built on top of RDDs
- New libraries: built on Datasets and DataFrames

A Tale of Three Apache Spark APIs: RDDs, DataFrames, and Datasets

Unified API


Unified Apache Spark 2.0 API
DataFrame
Dataset
Dataset
(2016)
Untyped API
• DataFrame = Dataset[Row]
• Alias
Typed API
• Dataset[T]
databricks

Typed and Un-typed APIs

Language	Main Abstraction
Scala	Dataset[T] & DataFrame (alias for Dataset[Row])
Java	Dataset[T]
Python*	DataFrame
R*	DataFrame

Note: *Since Python and R have no compile-time type-safety, we only have untyped APIs, namely DataFrames.*

Benefits of Dataset APIs

- Static-typing and runtime type-safety
  - SQL least restrictive, no syntax error until runtime
  - DF/DS: syntax error detected at compile time
- High-level abstraction and custom view into structured and semi-structured data, e.g. JSON
- Ease-of-use of APIs with structure
  - Rich semantics and domain specific operations
- Performance and Optimization
  - SQL Catalyst

DataFrame

```
ctx = new HiveContext()
users = ctx.table("users")
young = users.where(users("age") < 21)
println(young.count())
```

- A distributed collection of rows with the same schema
- Can be constructed from external data sources or RDDs into essentially an RDD of Row objects
- Supports relational operators (e.g. *where*, *groupby*) as well as Spark operations.
- Evaluated lazily → unmaterialized *logical* plan

DataFrames

dept	age	name
Bio	48	H Smith
CS	54	A Turing
Bio	43	B Jones
Chem	61	M Kennedy

Data grouped into named columns

RDD API

```
pdata.map(lambda x: (x.dept, [x.age, 1])) \
  .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \
  .map(lambda x: [x[0], x[1][0] / x[1][1]]) \
  .collect()
```

DataFrame API

```
data.groupBy("dept").avg("age")
```

DataFrames

dept	age	name
Bio	48	H Smith
CS	54	A Turing
Bio	43	B Jones
Chem	61	M Kennedy

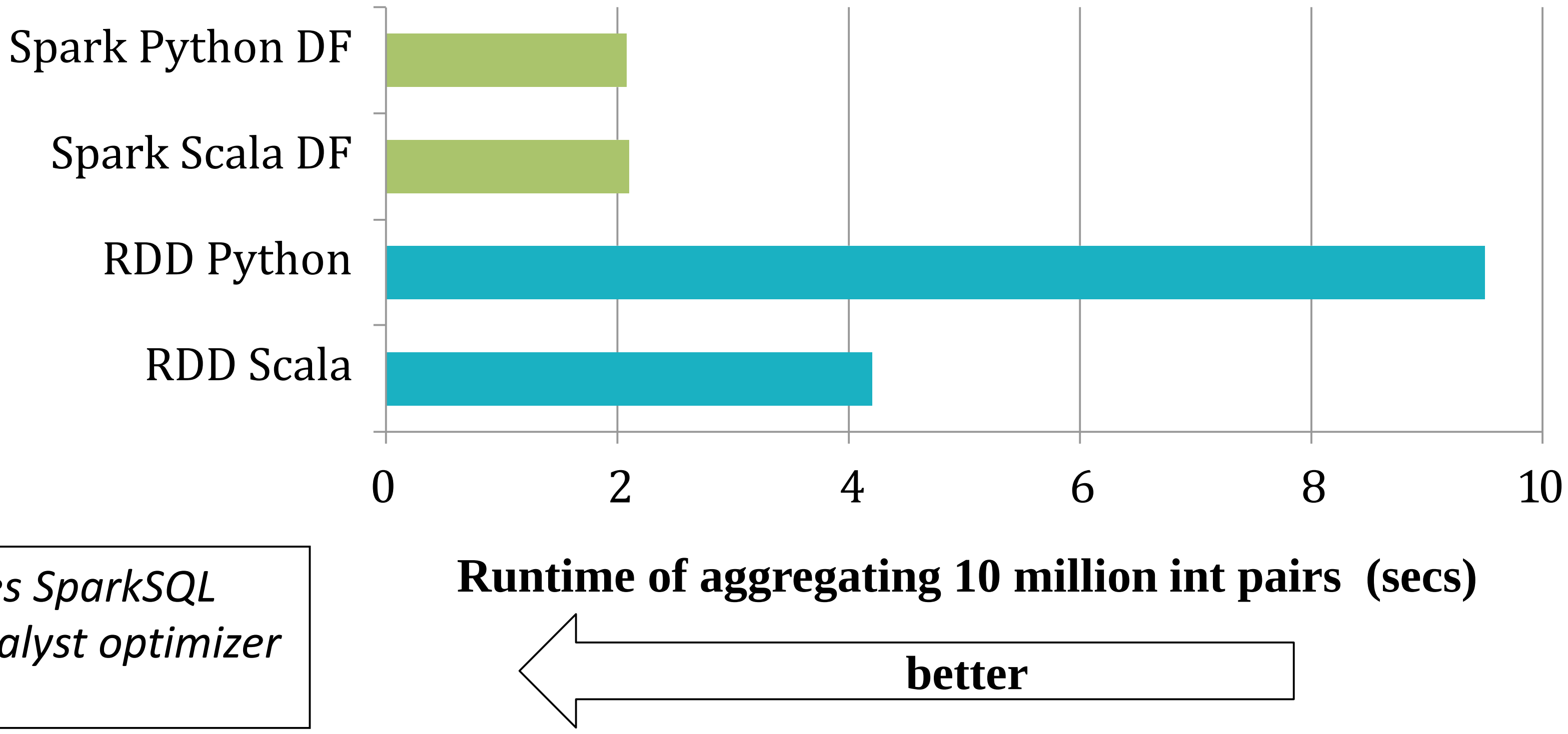
Data grouped into named columns

DSL for common tasks

- Project, filter, aggregate, join, …
- Metadata
- UDFs

```
data.groupBy("dept").avg("age")
```

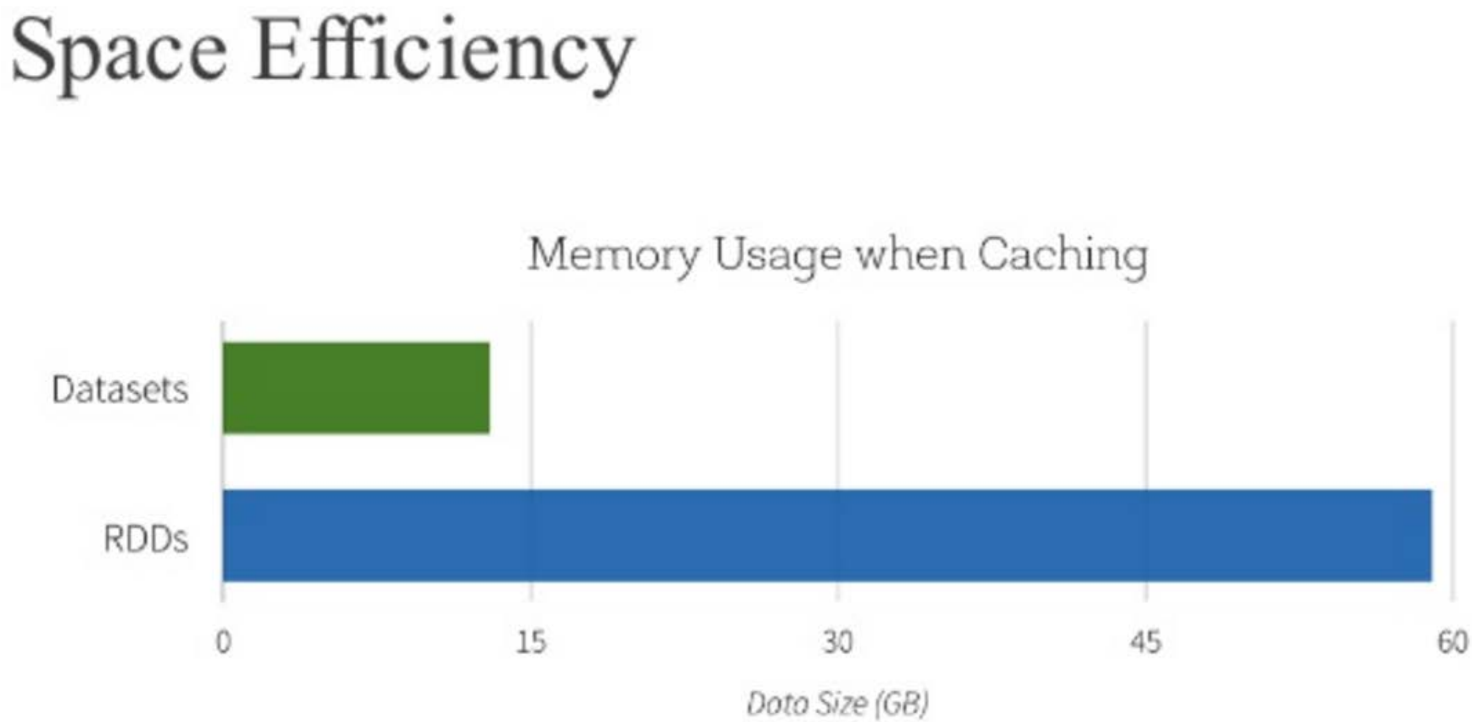
Spark DataFrames are *fast*



Uses SparkSQL Catalyst optimizer

35

Space Efficiency



Week 2 Contents

- Spark Recap – Example on Cache
- Spark DataFrame & Dataset
- **Spark MLlib & ML Pipelines**
- GitHub Classroom & Quiz 1

Machine Learning Library (MLlib)

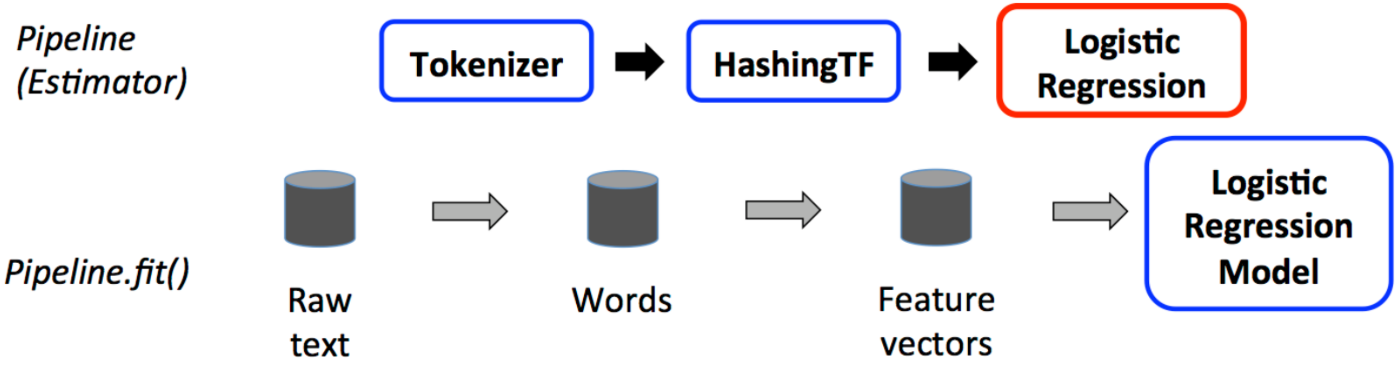
- ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering
- Featurization: feature extraction, transformation, dimensionality reduction, and selection
- Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
- Persistence: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

Main Concepts in Pipelines

- DataFrame: an ML dataset, which can hold a variety of data types. E.g., different columns storing text, feature vectors, true labels, and predictions.
- Transformer: algorithm transforming one DataFrame into another DataFrame. E.g., ML model➔ features into predictions.
- Estimator: algorithm fit on a DataFrame to produce a Transformer. E.g., ML algorithm DataFrame →model
- Pipeline: chains multiple Transformers and Estimators together to specify an ML workflow.
- Parameter: All Transformers and Estimators now share a common API for specifying parameters

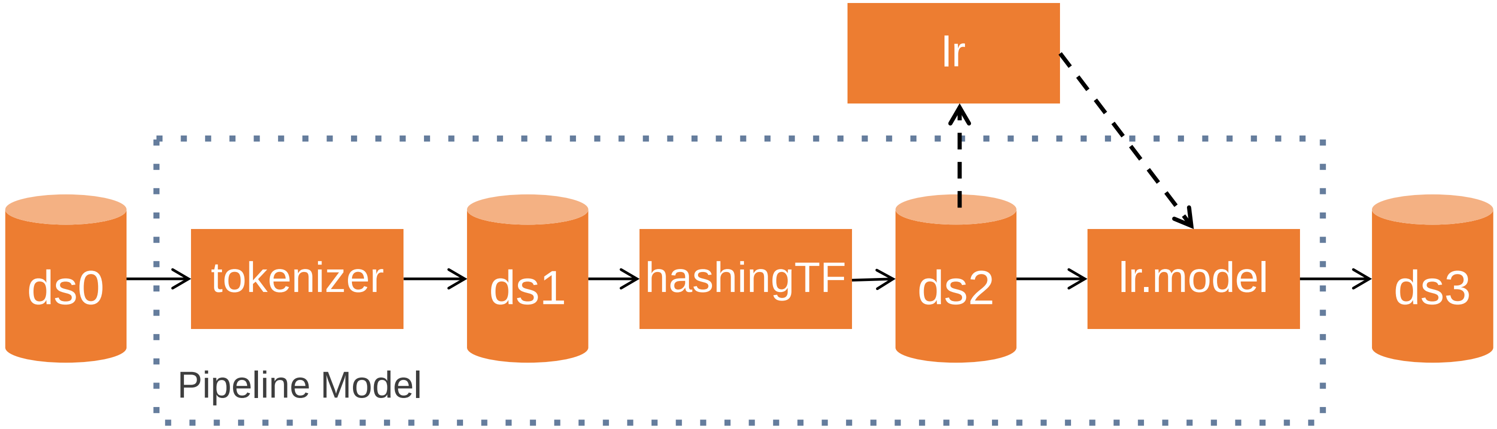
ML Pipelines

- ML Pipelines: high-level APIs to create and tune machine learning pipelines.
- Spark DataFrame: distributed collection of data organized into named columns.
    - Table in a relational database
    - Data frame in R or Python



Spark MLlib Pipelines

```
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)
```



Example: Text Classification

Goal: Given a text document, predict its topic.

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...
```

⟹

Label

1: about science
0: not about science

Dataset: "20 Newsgroups"
From UCI KDD Archive

ML Workflow

Load data

↓

Extract features

↓

Train model

↓

Evaluate

Load Data

Load data

Extract features

Train model

Evaluate

Data sources for DataFrames

built-in

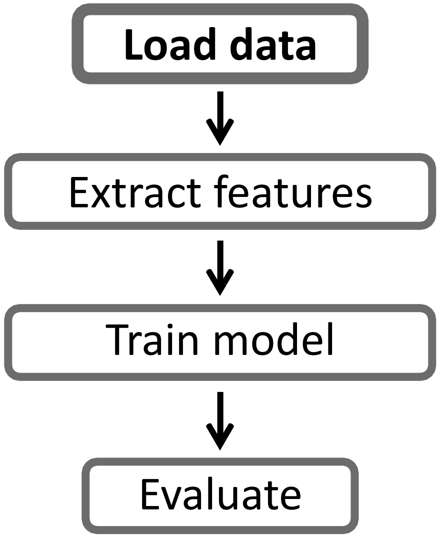
Parquet
JDBC
{ JSON }
PostgreSQL
HIVE
MySQL
HDFS
amazon web services S3
H2

external

AVRO
CSV
dBase
APACHE HBASE
elasticsearch.
cassandra
amazon web services Amazon Redshift

and more ...

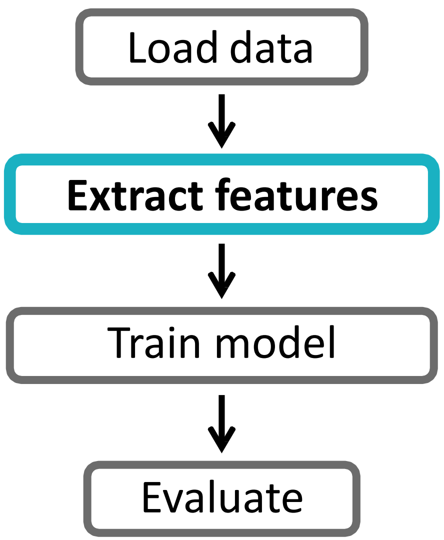
Load Data



<u>Current data schema</u>

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

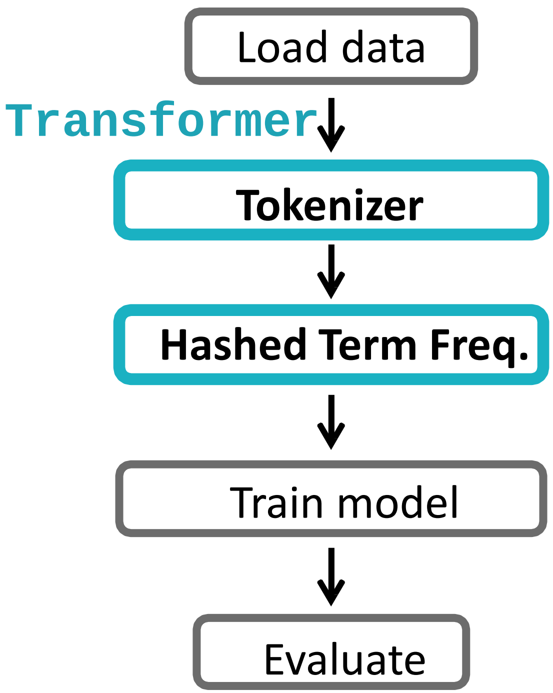
Extract Features



<u>Current data schema</u>

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

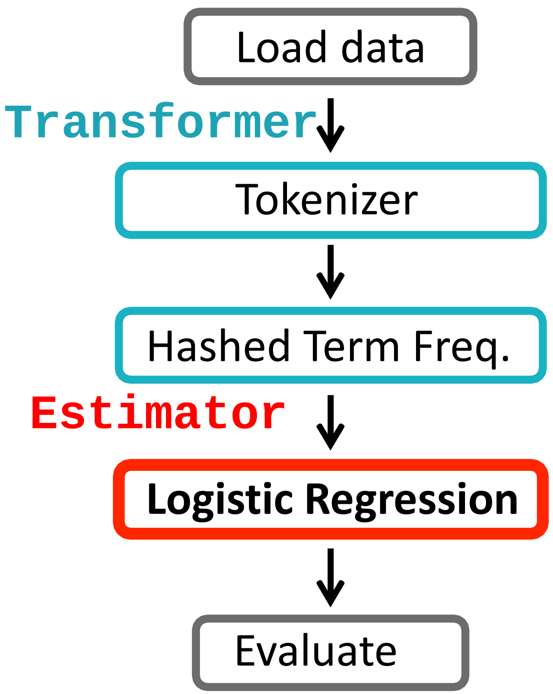
Extract Features



<u>Current data schema</u>

```
label: Int
text: String

words: Seq[String]

features: Vector
```

Train a Model

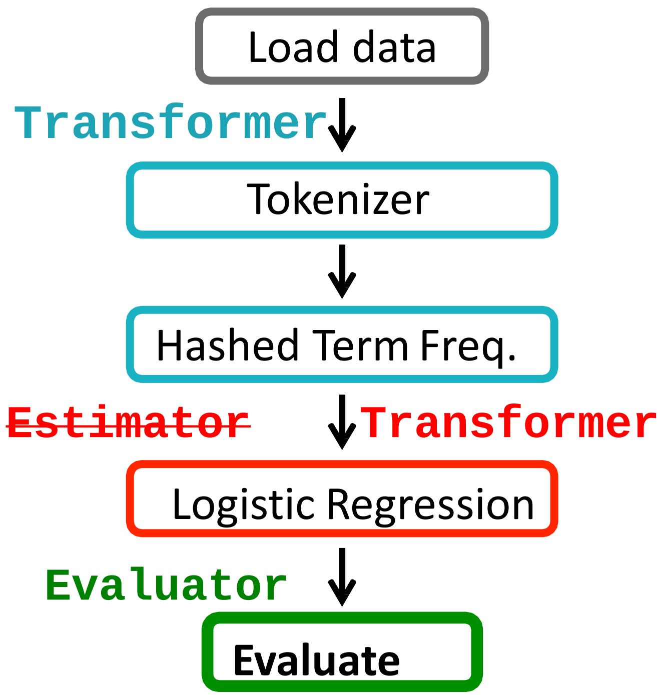


<u>Current data schema</u>

```
label: Int
text: String

words: Seq[String]

features: Vector
```

`model parameters`

Evaluate the Model



<u>Current data schema</u>

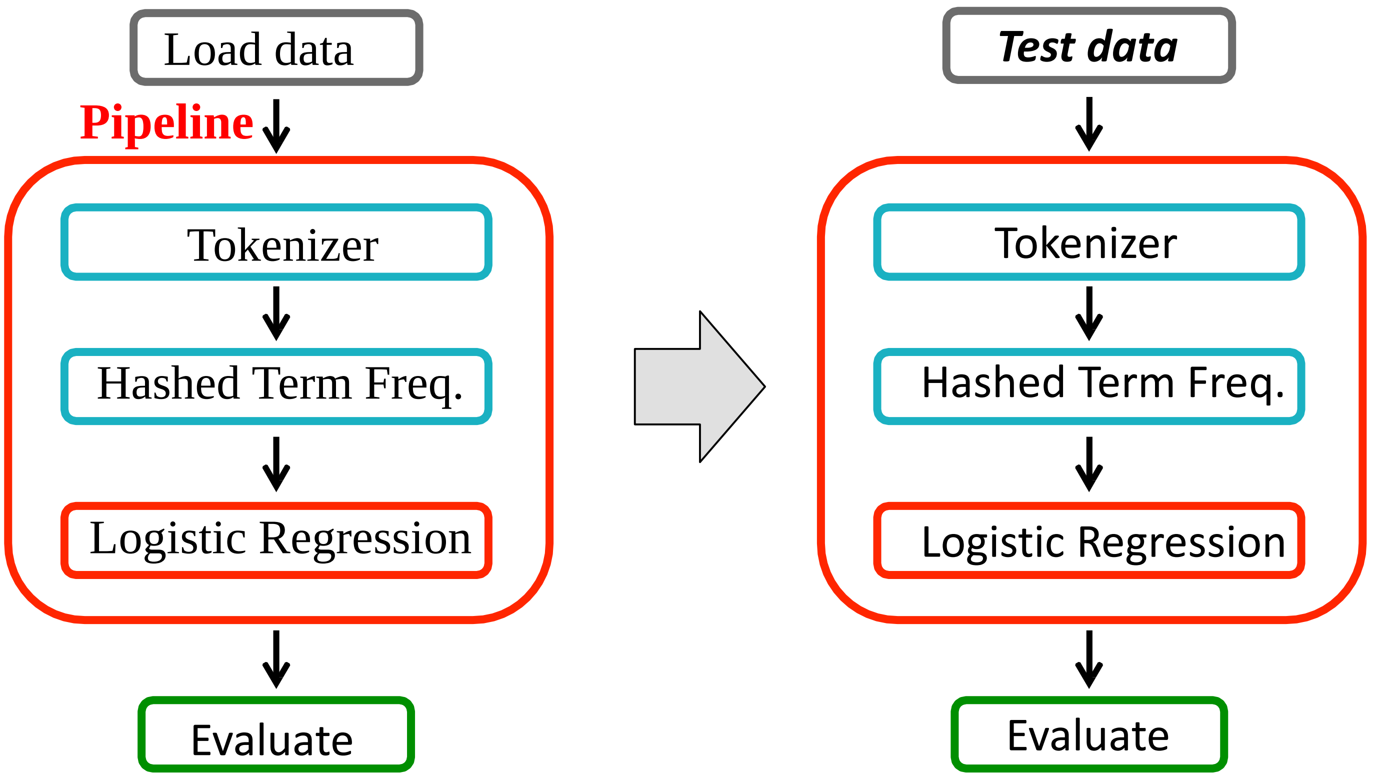
```
label: Int
text: String

words: Seq[String]

features: Vector

prediction: Int
```

By default, always append new columns

→ Can go back & inspect intermediate results

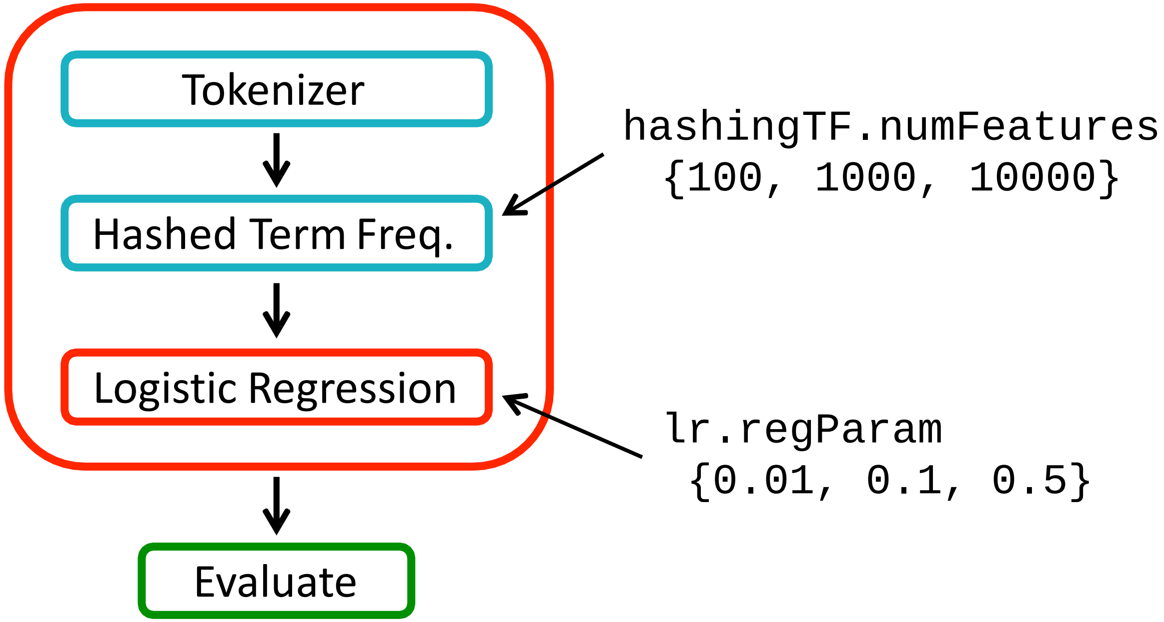
→ Made efficient by DataFrame optimizations

ML Pipelines



Re--run exactly the same way

Parameter Tuning



CrossValidator

Given:

•Estimator

•Parameter grid

•Evaluator

Find best parameters

Week 2 Contents

- Spark Recap – Example on Cache
- Spark DataFrame & Dataset
- Spark MLlib & ML Pipelines
- **GitHub Classroom & Quiz 1**

GitHub Classroom & Quiz 1

- GitHub Classroom
  - Lab sessions
  - Quizzes
  - Assignments
  - [Feedbacks]
- **Quiz 1**
  - **22 Feb 2018 10am in lab session**
  - 50 minutes max
  - **10%** of your total mark

Recommended Reading

- Sections 2.4.2 and 2.4.3 of the MMDS book (3rd edition)

  http://i.stanford.edu/~ullman/mmds/ch2n.pdf

- The full book http://i.stanford.edu/~ullman/mmds/book0n.pdf