

Apache Spark

Advanced in-memory BigData Analytics

Vitalii Bondarenko

Data Platform Competency Manager at Eleks

Vitaliy.bondarenko@eleks.com

Agenda



- Spark Platform
- Spark Core
- Spark Extensions
- Using Apache Spark

About me

Vitalii Bondarenko

Data Platform Competency Manager

Eleks

www.eleks.com

20 years in software development

9+ years of developing for MS SQL Server

3+ years of architecting Big Data Solutions

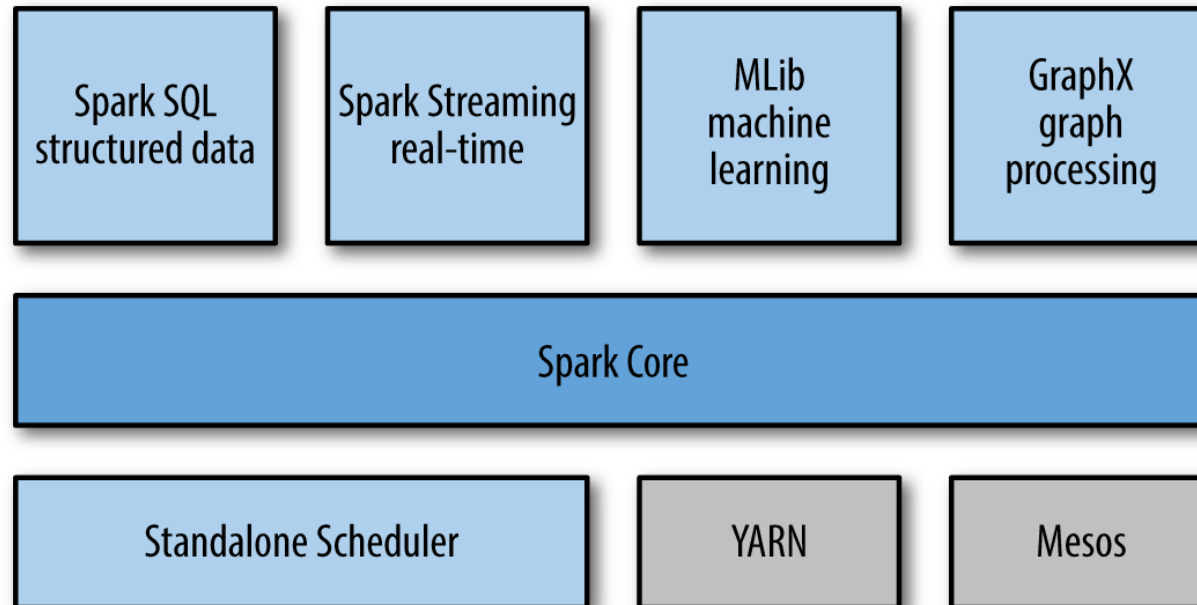
- DW/BI Architect and Technical Lead
- OLTP DB Performance Tuning
- Big Data Data Platform Architect

eleks®



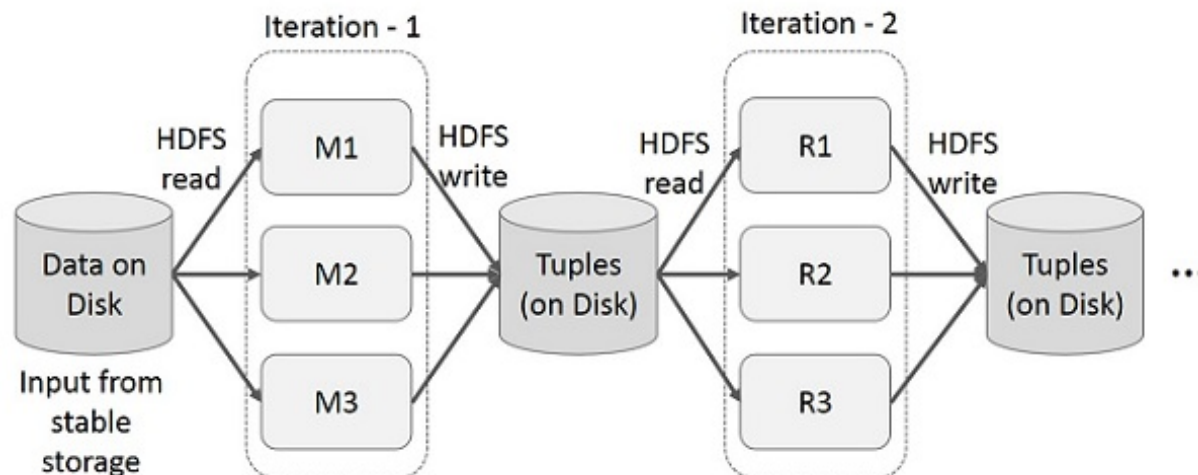
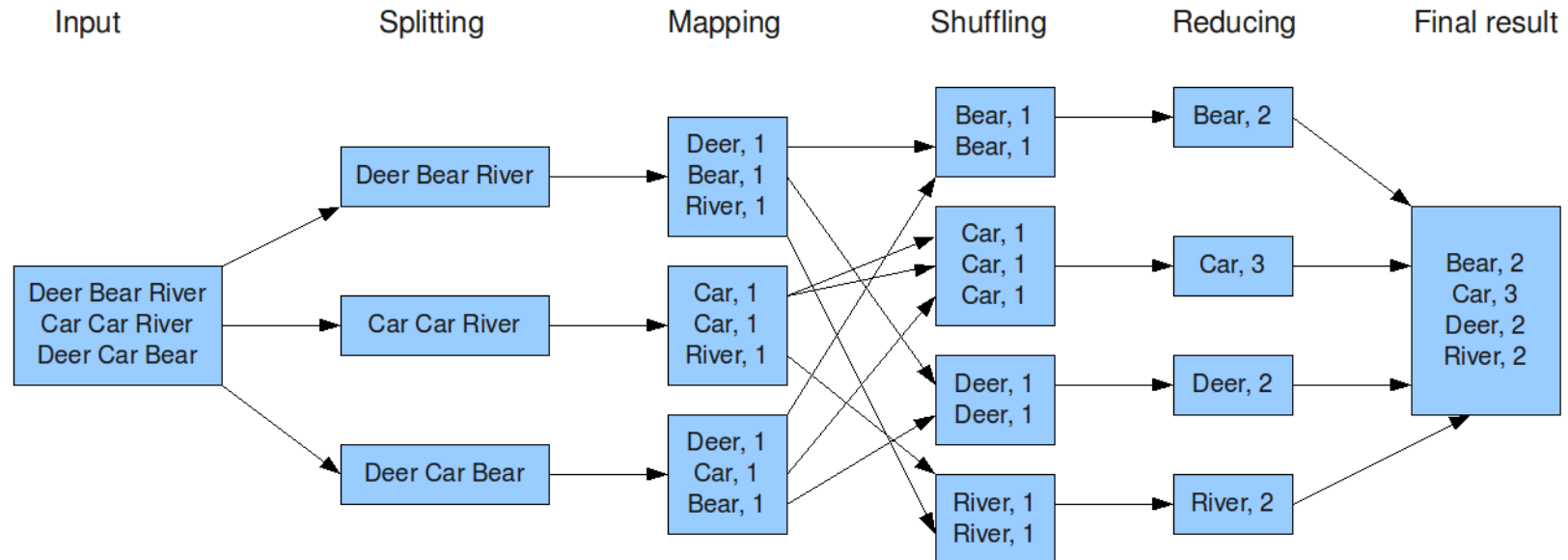
Spark Stack

- Clustered computing platform
- Designed to be fast and general purpose
- Integrated with distributed systems
- API for Python, Scala, Java, clear and understandable code
- Integrated with Big Data and BI Tools
- Integrated with different Data Bases, systems and libraries like Cassandra, Kafka, H2O
- First Apache release 2013, Aug 2016 v.2.0 has been released

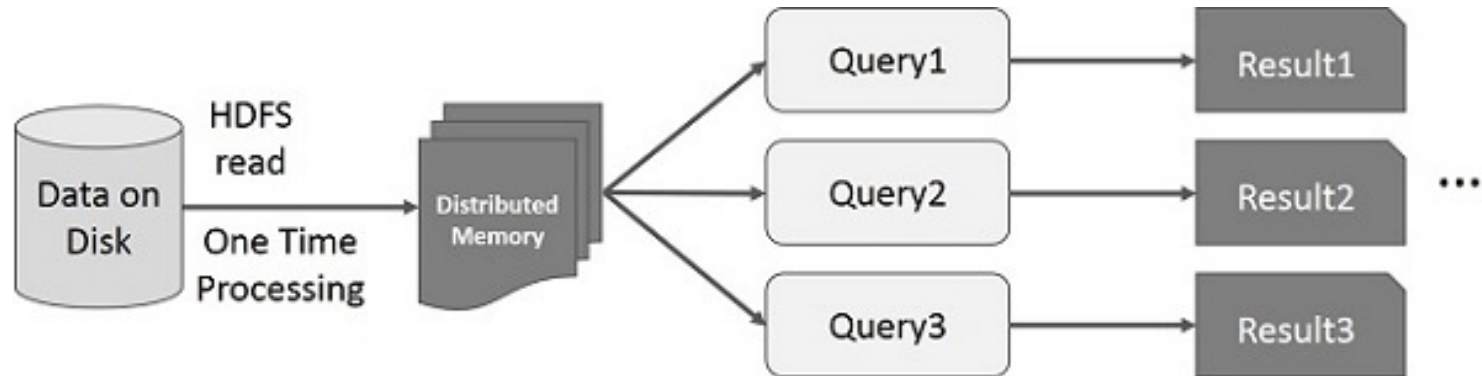
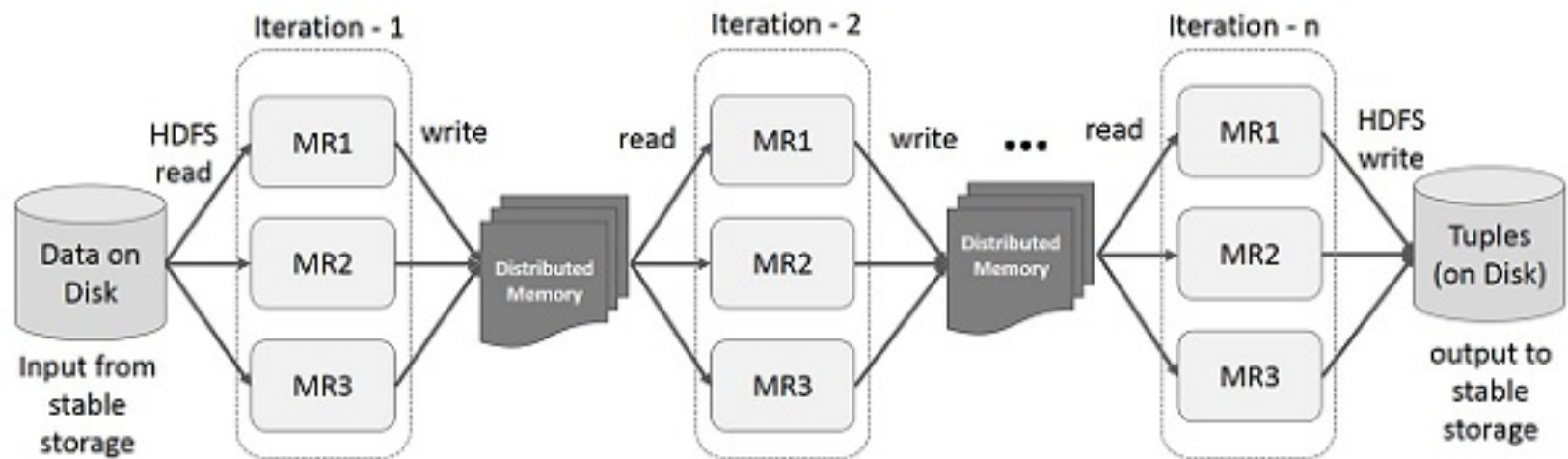


Map-reduce computations

The overall MapReduce word count process



In-memory map-reduce



Execution Model

Spark Execution

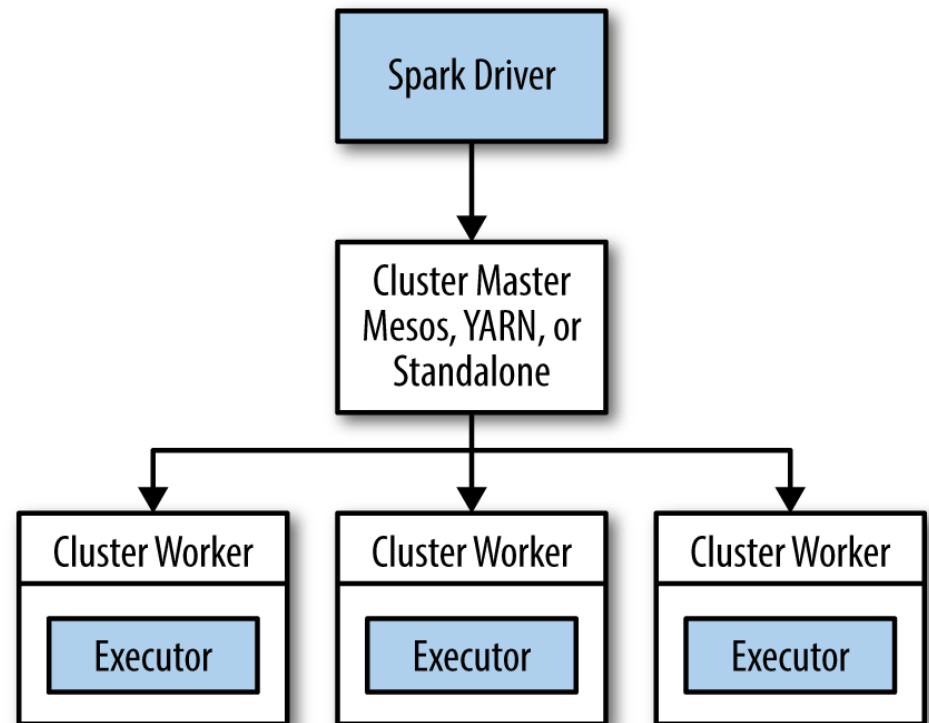
- Shells and Standalone application
- Local and Cluster (Standalone, Yarn, Mesos, Cloud)

Spark Cluster Architecture

- Master / Cluster manager
- Cluster allocates resources on nodes
- Master sends app code and tasks to nodes
- Executors run tasks and cache data

Connect to Cluster

- Local
- SparkContext and Master field
- spark://host:7077
- Spark-submit



RDD: resilient distributed dataset

- Parallelized collections with fault-tolerant (Hadoop datasets)
- **Transformations** set new RDDs (filter, map, distinct, union, subtract, etc)
- **Actions** call to calculations (count, collect, first)
- Transformations are lazy

```
from pyspark import SparkContext as sc
```

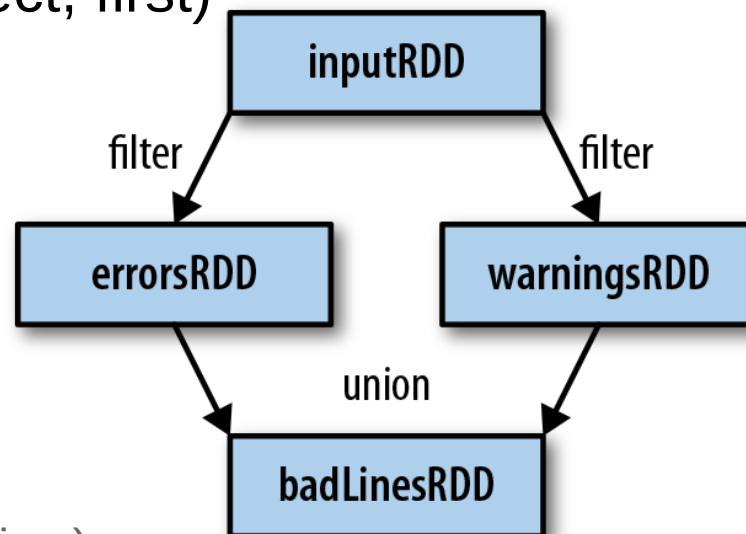
```
inputRDD = sc.textFile("log.txt")
```

```
errorsRDD = inputRDD.filter(lambda x: "error" in x)
```

```
warningsRDD = inputRDD.filter(lambda x: "warning" in x)
```

```
badLinesRDD = errorsRDD.union(warningsRDD)
```

```
print "Input had " + badLinesRDD.count() + " concerning lines"
```



Transformations (1)

Function name	Purpose	Example	Result
<code>map()</code>	Apply a function to each element in the RDD and return an RDD of the result.	<code>rdd.map(x => x + 1)</code>	{2, 3, 4, 4}
<code>flatMap()</code>	Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned. Often used to extract words.	<code>rdd.flatMap(x => x.to(3))</code>	{1, 2, 3, 2, 3, 3, 3}
<code>filter()</code>	Return an RDD consisting of only elements that pass the condition passed to <code>filter()</code> .	<code>rdd.filter(x => x != 1)</code>	{2, 3, 3}
<code>distinct()</code>	Remove duplicates.	<code>rdd.distinct()</code>	{1, 2, 3}

Transformations (2)

Function name	Purpose	Example	Result
<code>union()</code>	Produce an RDD containing elements from both RDDs.	<code>rdd.union(other)</code>	{1, 2, 3, 3, 4, 5}
<code>intersection()</code>	RDD containing only elements found in both RDDs.	<code>rdd.intersection(other)</code>	{3}
<code>subtract()</code>	Remove the contents of one RDD (e.g., remove training data).	<code>rdd.subtract(other)</code>	{1, 2}
<code>cartesian()</code>	Cartesian product with the other RDD.	<code>rdd.cartesian(other)</code>	{(1, 3), (1, 4), ..., (3, 5)}

Actions (1)

Function name	Purpose	Example	Result
<code>collect()</code>	Return all elements from the RDD.	<code>rdd.collect()</code>	{1, 2, 3, 3}
<code>count()</code>	Number of elements in the RDD.	<code>rdd.count()</code>	4
<code>countByValue()</code>	Number of times each element occurs in the RDD.	<code>rdd.countByValue()</code>	{(1, 1), (2, 1), (3, 2)}
<code>take(num)</code>	Return num elements from the RDD.	<code>rdd.take(2)</code>	{1, 2}
<code>top(num)</code>	Return the top num elements the RDD.	<code>rdd.top(2)</code>	{3, 3}

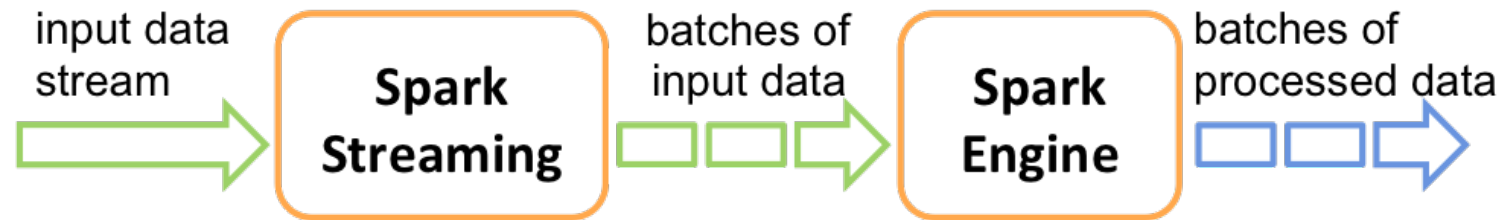
Actions (2)

<code>takeOrdered(num) (ordering)</code>	Return numelements based on provided ordering.	<code>rdd.takeOrdered(2) (myOrdering)</code>	{3, 3}
<code>reduce(func)</code>	Combine the elements of the RDD together in parallel (e.g.,sum).	<code>rdd.reduce((x, y) => x + y)</code>	9
<code>fold(zero)(func)</code>	Same as <code>reduce()</code> but with the provided zero value.	<code>rdd.fold(0)((x, y) => x + y)</code>	9
<code>aggregate(zeroValue) (seqOp, combOp)</code>	Similar to <code>reduce()</code> but used to return a different type.	<code>rdd.aggregate((0, 0)) ((x, y) =>(x._1 + y, x._2 + 1), (x, y) => (x._1 + y._1, x._2 + y._2))</code>	(9, 4)

DEMO: Execution Environments

- Local Spark installation
- Shells and Notebook
- Spark Examples
- HDInsight Spark Cluster
- SSH connection to Spark in Azure
- Jupyter Notebook connected to HDInsight Spark
- Transformations
- Actions

Spark Streaming Architecture



- Micro-batch architecture
- SparkStreaming Context
- Batch interval from 500ms
- Transformation on Spark Engine
- Output operations instead of Actions
- Different sources and outputs



Spark Streaming Example

```
from pyspark.streaming import StreamingContext

ssc = StreamingContext(sc, 1)

input_stream = ssc.textFileStream("sampleTextDir")

word_pairs = input_stream.flatMap(
    lambda l:l.split(" ")).map(lambda w: (w,1))

counts = word_pairs.reduceByKey(lambda x,y: x + y)

counts.print()

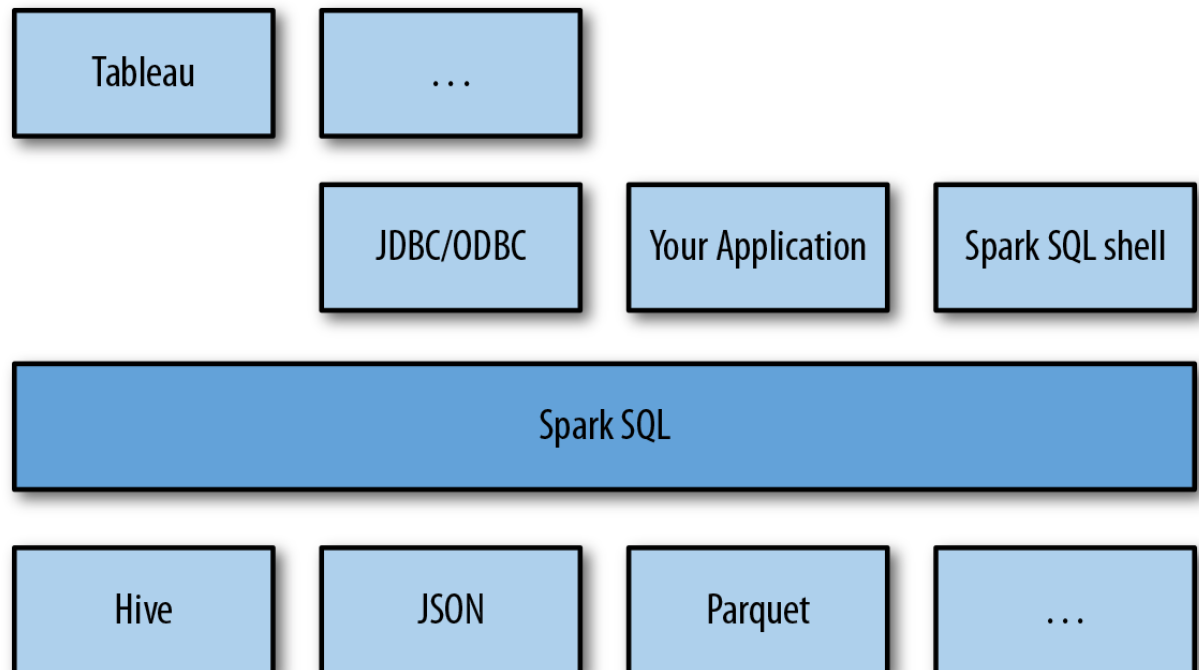
ssc.start()

ssc.awaitTermination()
```

- Process RDDs in batches
- Start after ssc.start()
- Output to console on Driver
- Awaiting termination

Spark SQL

- SparkSQL interface for working with structured data by SQL
- Works with Hive tables and HiveQL
- Works with files (Json, Parquet etc) with defined schema
- JDBC/ODBC connectors for BI tools
- Integrated with Hive and Hive types, uses HiveUDF
- DataFrame abstraction



Spark DataFrames

```
# Import Spark SQL from pyspark.sql
import HiveContext, Row
```

```
# Or if you can't include the hive requirements from pyspark.sql
import SQLContext, Row
```

```
sc = new SparkContext(...)
hiveCtx = HiveContext(sc)
sqlContext = SQLContext(sc)
```

```
input = hiveCtx.jsonFile(inputFile)
```

```
# Register the input schema RDD
input.registerTempTable("tweets")
```

```
# Select tweets based on the retweet
CounttopTweets = hiveCtx.sql("""SELECT text, retweetCount FROM tweets ORDER BY retweetCount LIMIT 10""")
```

- `hiveCtx.cacheTable("tableName")`, in-memory, column-store, while driver is alive
- `df.show()`
- `df.select("name", df("age")+1)`
- `df.filtr(df("age") > 19)`
- `df.groupBy(df("name")).min()`

Spark ML

Spark ML

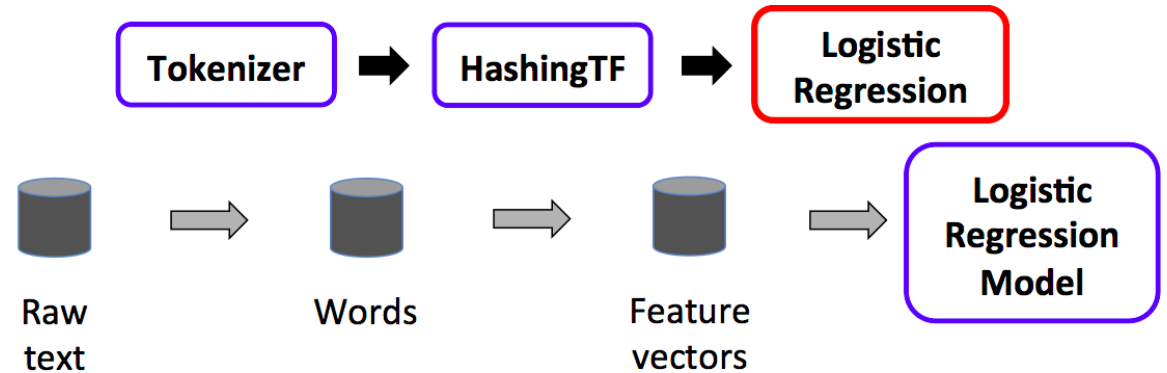
- Classification
- Regression
- Clustering
- Recommendation
- Feature transformation, selection
- Statistics
- Linear algebra
- Data mining tools

Pipeline Components

- DataFrame
- Transformer
- Estimator
- Pipeline
- Parameter

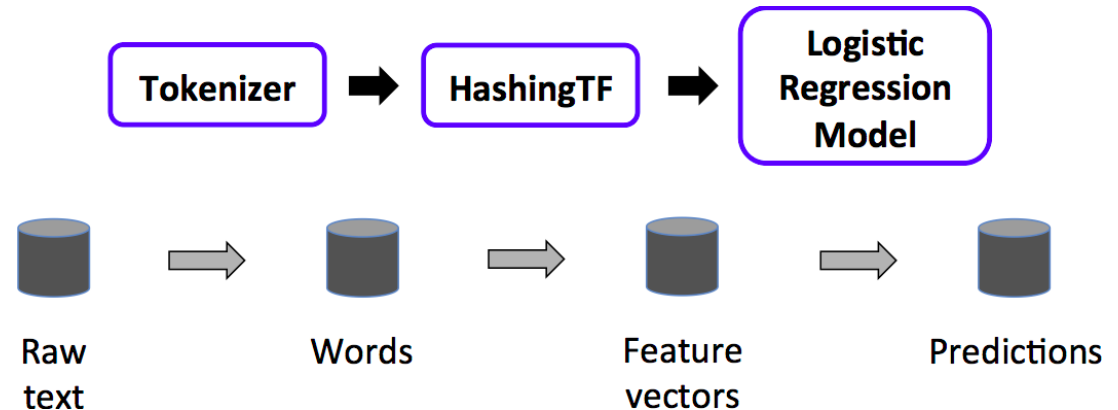
*Pipeline
(Estimator)*

Pipeline.fit()



*PipelineModel
(Transformer)*

*PipelineModel
.transform()*



DEMO: Spark

- Spark Streaming
- ActionsSimple SparkSQL querying
- Data Frames
- Data exploration with SparkSQL
- Connect from BI
- Training a model
- Data visualization

Using Spark

1. Visual data exploration and interactive analysis (HDFS)
2. Spark with NoSQL (HBase and Cassandra)
3. Spark with Data Lake
4. Spark with Data Warehouse
5. Machine Learning using R Server, Mllib
6. Putting it all together in a notebook experience
7. Using BI with Spark

Q&A