eleks®

HDInsight: Spark

Advanced in-memory BigData Analytics with Microsoft Azure

Vitalii Bondarenko
Data Platform Competency Manager at Eleks
Vitaliy.bondarenko@eleks.com

Agenda

- Spark Platform
- Spark Core
- Spark Extensions
- Using HDInsight Spark

About me

Vitalii Bondarenko

Data Platform Competency Manager

Eleks

www.eleks.com

20 years in software development

8+ year developint for MS SQL Server

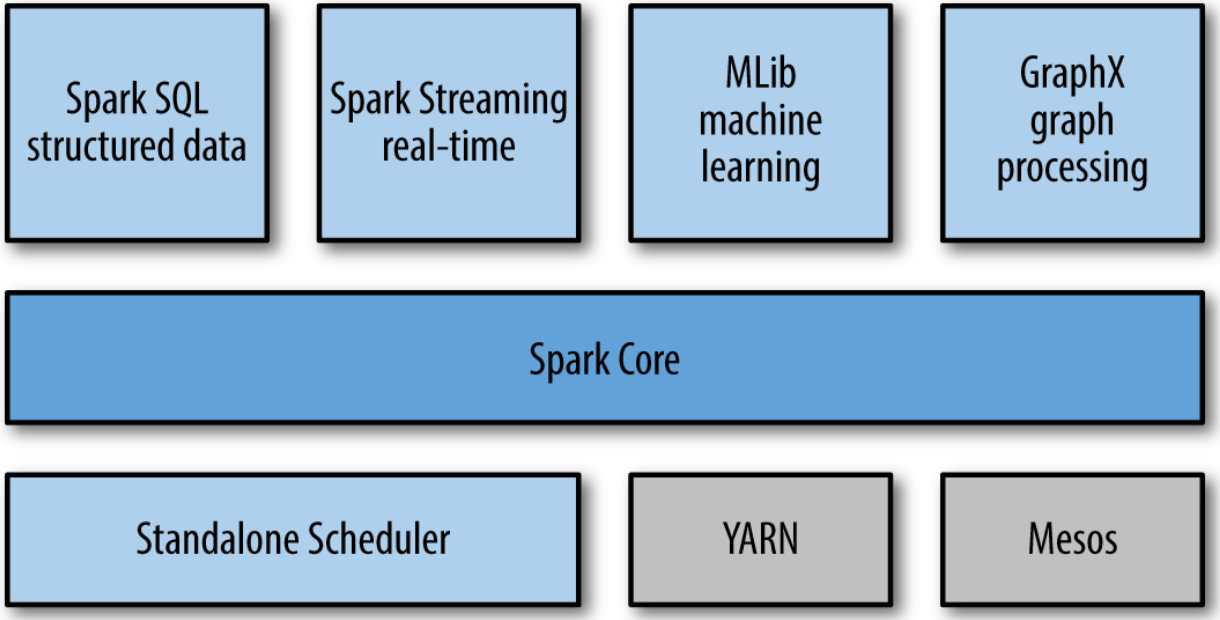
3+ year architecting Big Data Solutions

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

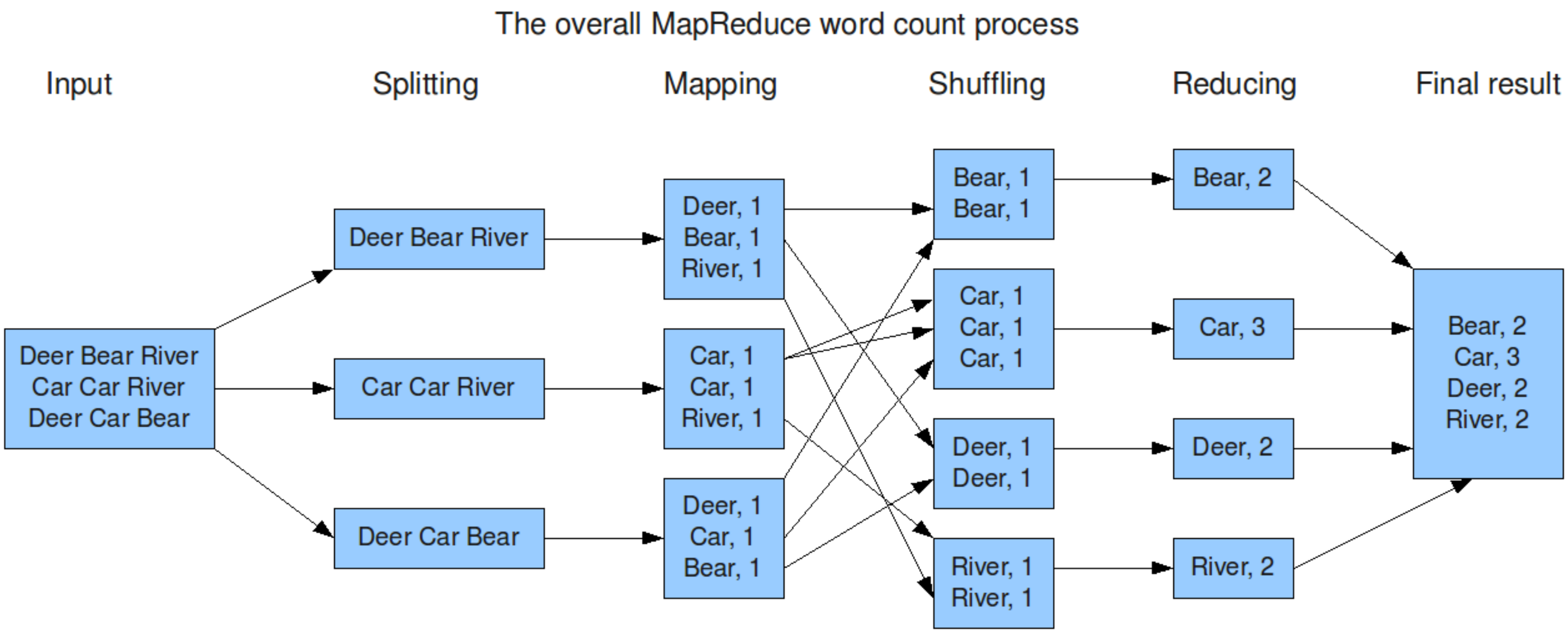
Spark Platform

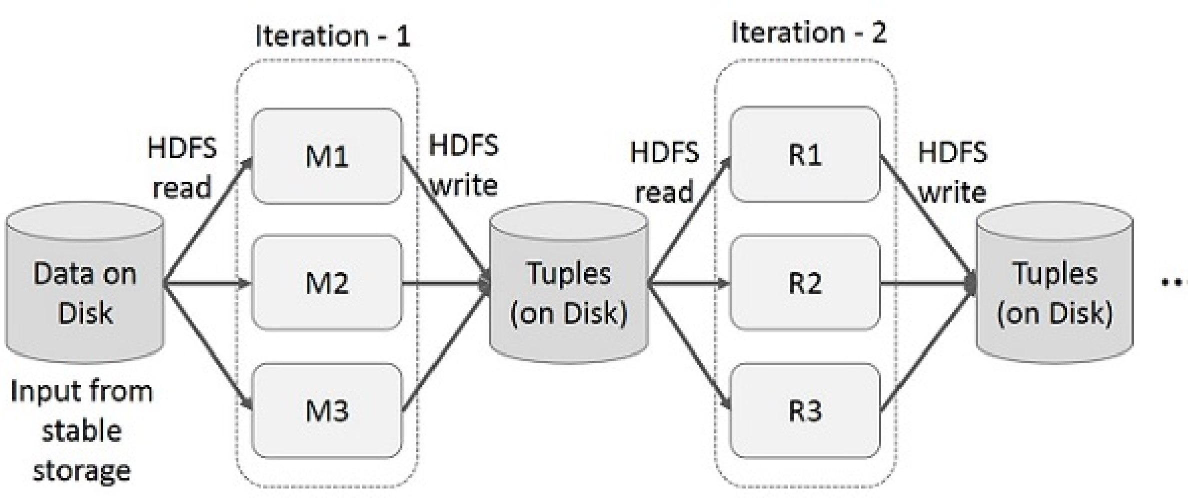
Spark Stack

- Clustered computing platform
- Designed to be fast and general purpose
- Integrated with distributed systems
- API for Python, Scala, Java for less coding
- Integrated with Big Data and BI Tools
- Integrated with different Data Bases, systems and libraries like Cassanda, Kafka, H2O
- SQL, ML, Streaming Analytics
- Direct Acyclic Graph (DAG) vs MapReduce

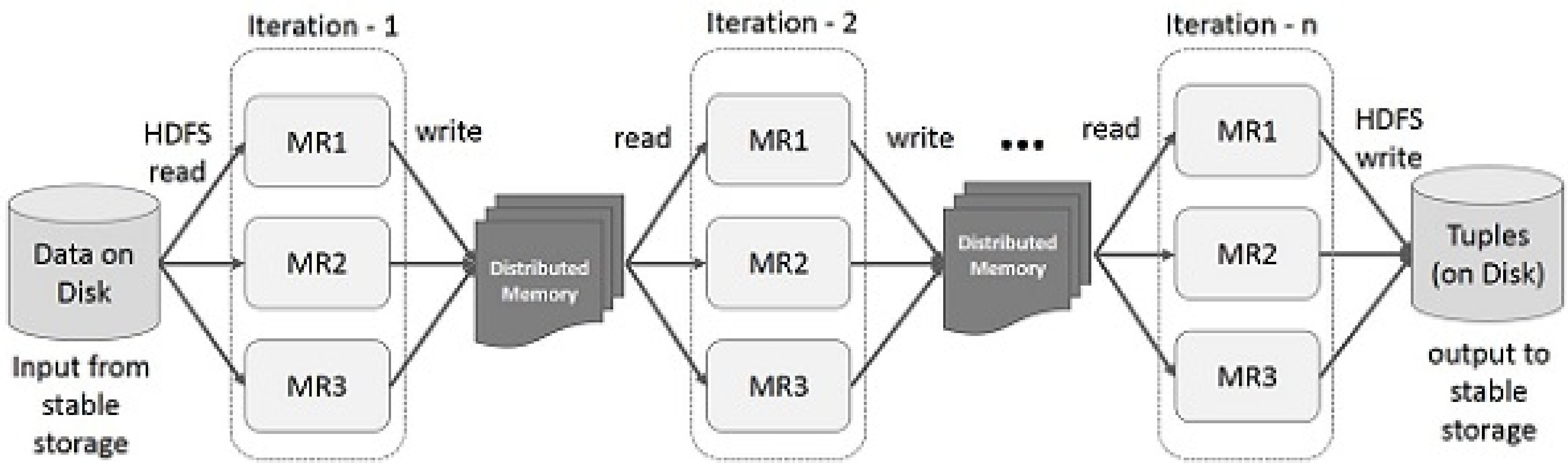


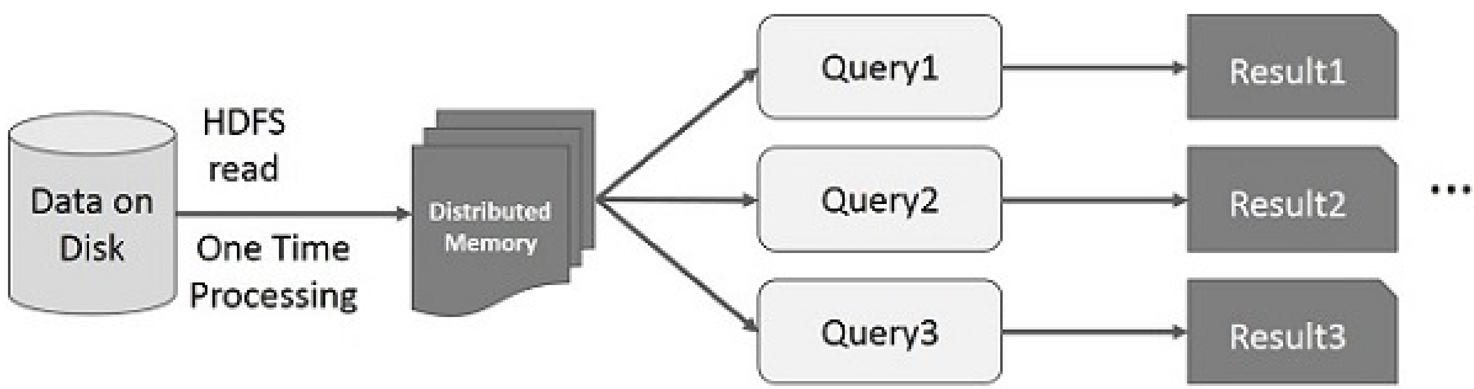
Map-reduce computations





In-memory map-reduce





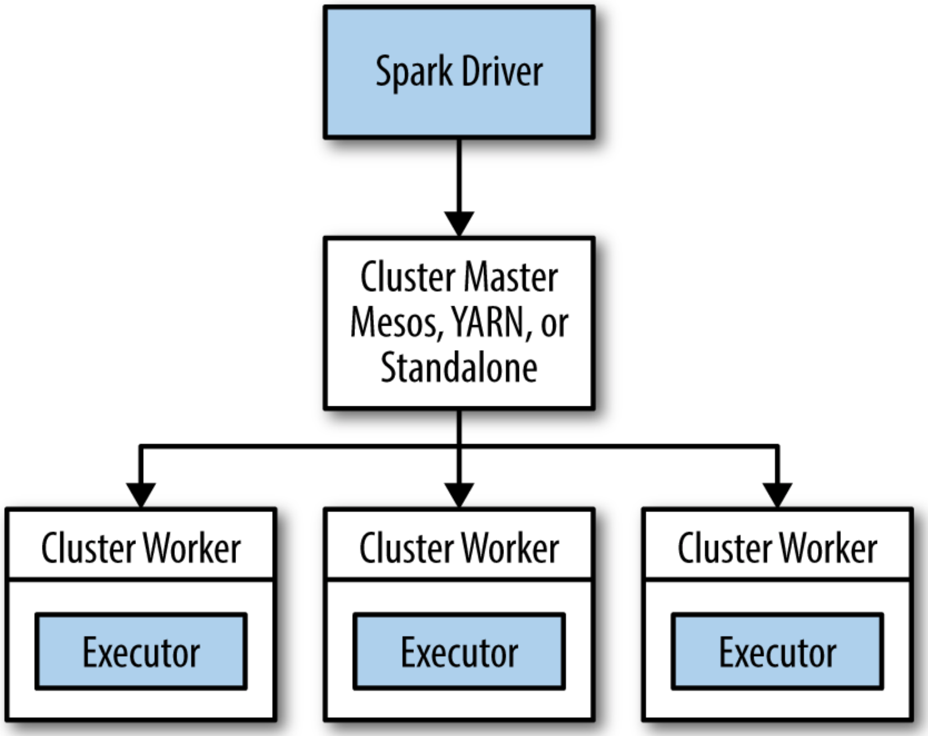
Execution Model

Spark Execution

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

Spark Cluster Arhitecture

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



Connect to Cluster

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

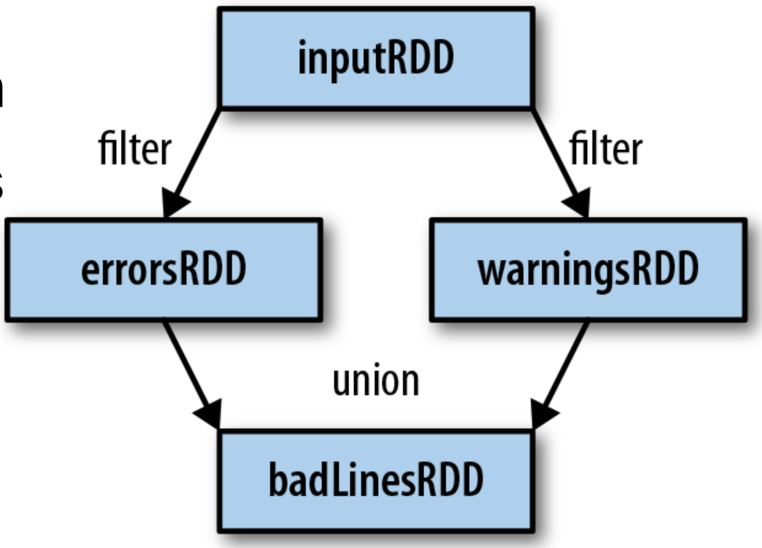
DEMO: Execution Environments

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

Spark Core

RDD: resilient distributed dataset

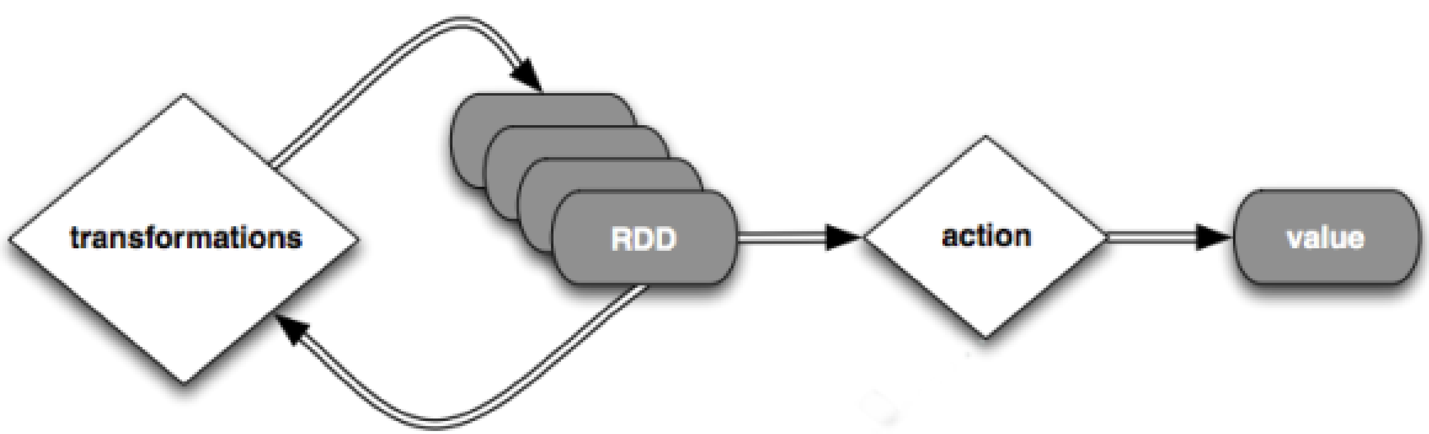
- Parallelized collections with fault-tolerant (Hadoop datasets)
- **Transformations** set new RDDs (filter, map, distinct, union, subtract, etc)
- **Actions** call to calculations (count, collect, first)
- Transformations are lazy
- Actions trigger transformations computation
- Broadcast Variables send data to executors
- Accumulators collect data on driver




```
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"
```

Spark program scenario

- Create RDD (loading external datasets, parallelizing a collection on driver)
- Transform
- Persist intermediate RDDs as results
- Launch actions
- Transformations are lazy
- Actions trigger transformations computation



Persistence (Caching)

- Avoid recalculations
- 10x faster in-memory
- Fault-tolerant
- Persistence levels
- Persist before first action

Level	Space used	CPU time	In memory	On disk	Comments
MEMORY_ONLY	High	Low	Y	N	
MEMORY_ONLY_SER	Low	High	Y	N	
MEMORY_AND_DISK	High	Medium	Some	Some	Spills to disk if there is too much data to fit in memory.
MEMORY_AND_DISK_SER	Low	High	Some	Some	Spills to disk if there is too much data to fit in memory. Stores serialized representation in memory.
DISK_ONLY	Low	High	N	Y	

```
input = sc.parallelize(xrange(1000))

result = input.map(lambda x: x ** x)

result.persist(StorageLevel.MEMORY_ONLY)

result.count()

result.collect()
```

Transformations (1)

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

Transformations (2)

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

Actions (1)

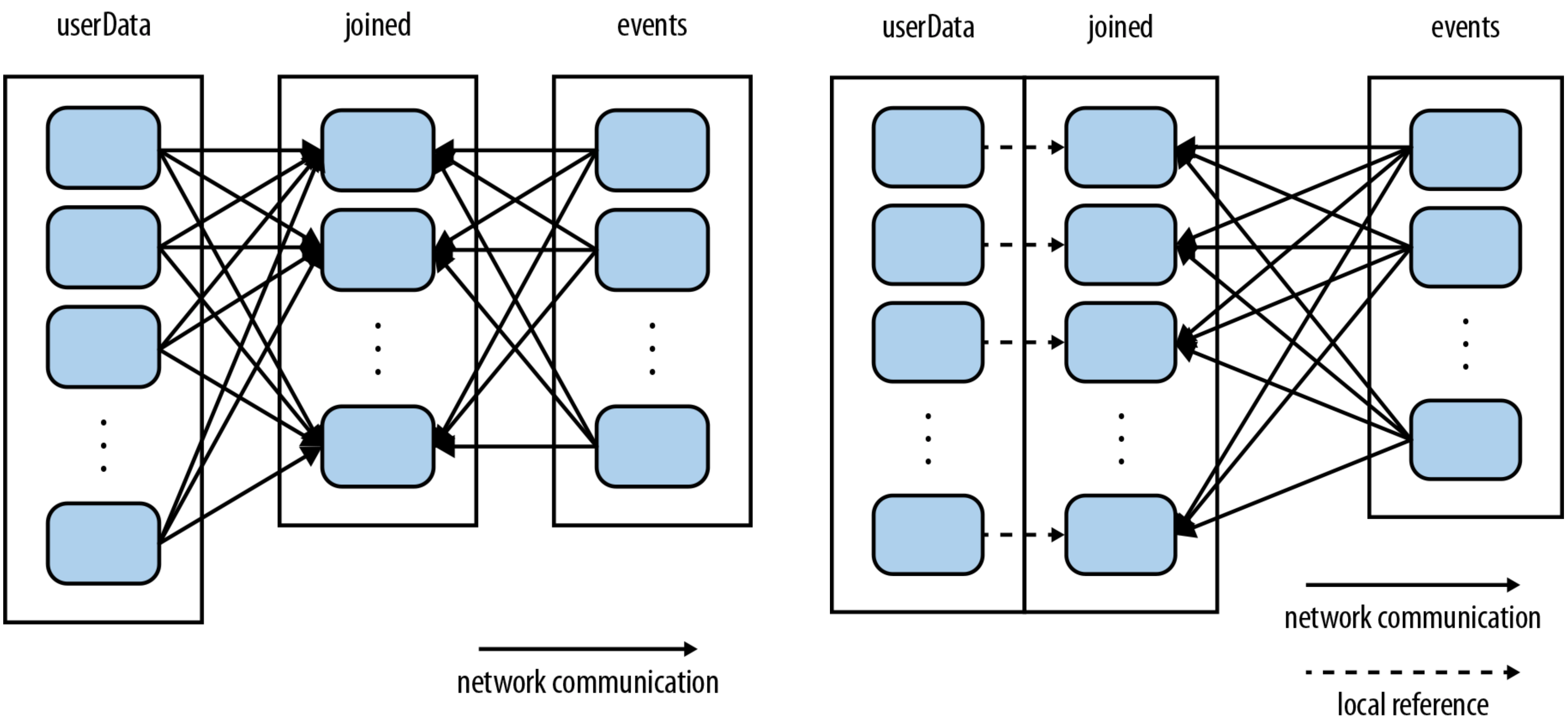
Function name	Purpose	Example	Result
`collect()`	Return all elements from the RDD.	`rdd.collect()`	`{1, 2, 3, 3}`
`count()`	Number of elements in the RDD.	`rdd.count()`	`4`
`countByValue()`	Number of times each element occurs in the RDD.	`rdd.countByValue()`	`{(1, 1), (2, 1), (3, 2)}`
`take(num)`	Return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

Actions (2)

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

Data Partitioning

- userData.join(events)
- userData.partitionBy(100).persist()
- 3-4 partitions on CPU Core
- userData.join(events).mapValues(...).reduceByKey(...)



DEMO: Spark Core Operations

- Transformations
- Actions

Spark Extensions

Spark Streaming Architecture

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

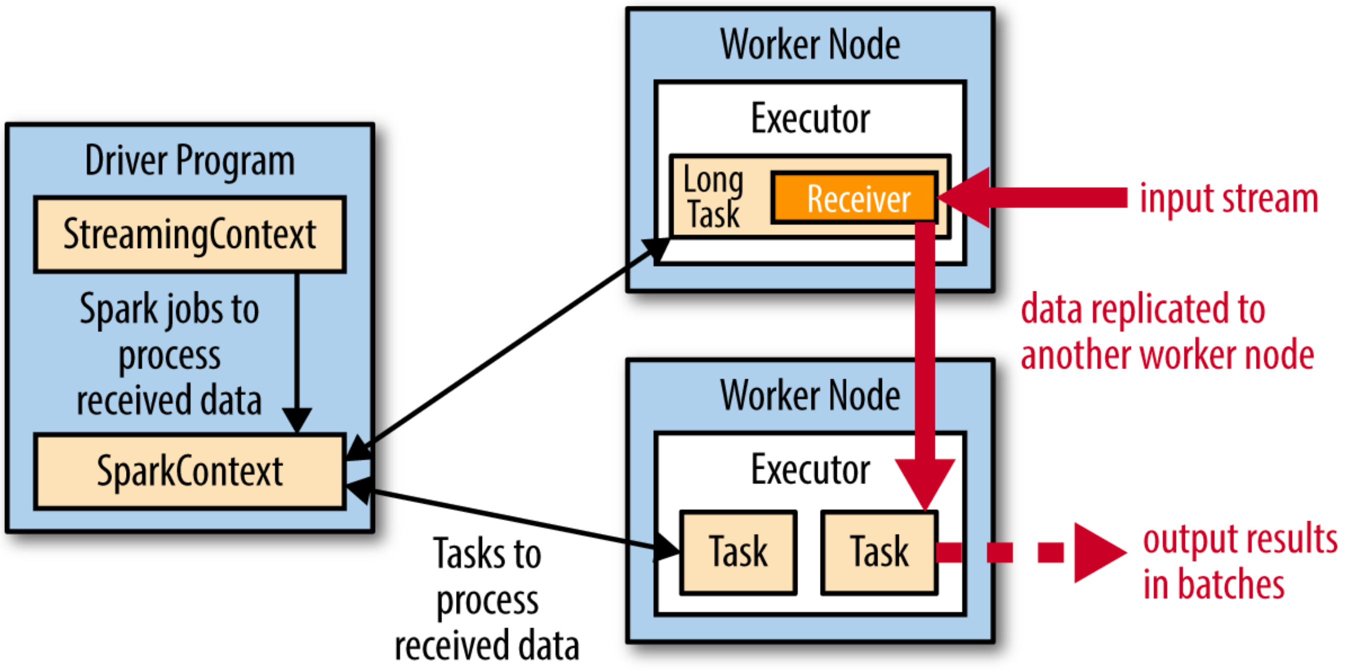
Spark Streaming Example

```
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

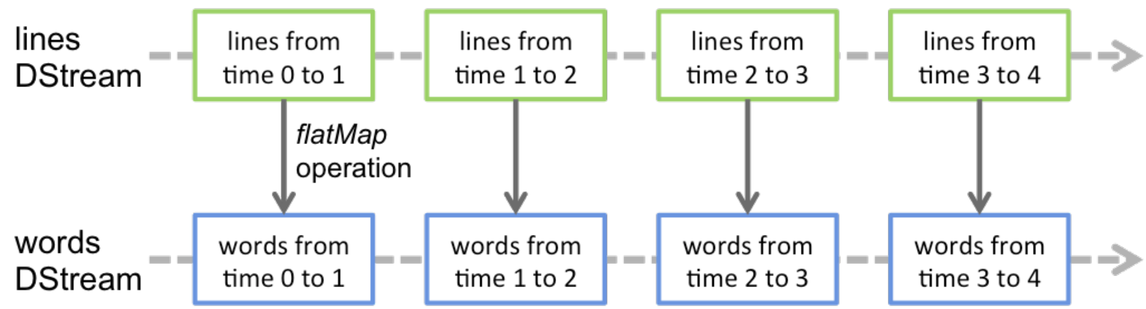
Streaming on a Cluster

- Receivers with replication
- SparkContext on Driver
- Output from Exectors in batches saveAsHadoopFiles()
- spark-submit for creating and scheduling periodical streaming jobs
- Chekpointing for saving results and restore from the point ssc.checkpoint(“hdfs://...”)

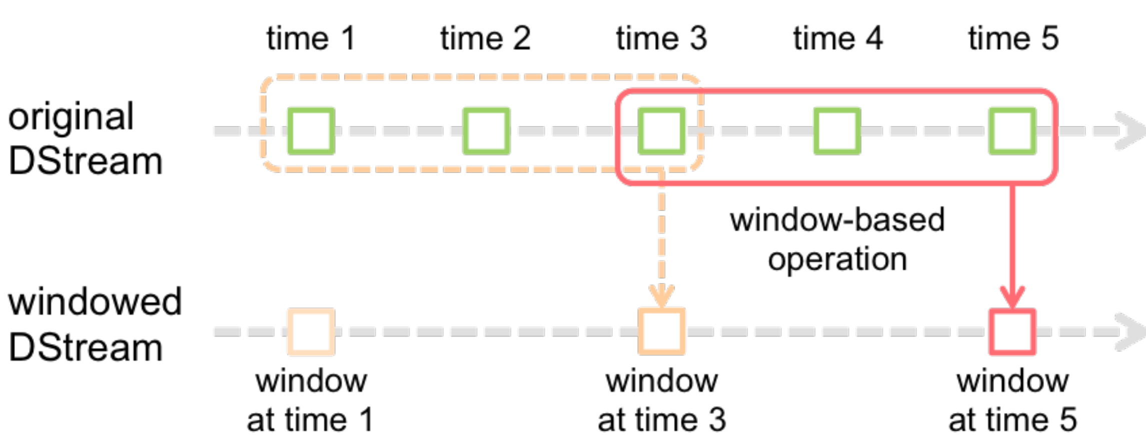


Streaming Transformations

- DStreams
- Stateless transformantions
- Stagefull transformantions
- Windowed transformantions
- UpdateStateByKey
- ReduceByWindow, reduceByKeyAndWindow
- Recomended batch size from 10 sec




```
val ipDStream = accessLogsDStream.map(logEntry => (logEntry.getIpAddress(), 1))
val ipCountDStream = ipDStream.reduceByKeyAndWindow(
  {(x, y) => x + y}, // Adding elements in the new batches entering the window
  {(x, y) => x - y}, // Removing elements from the oldest batches exiting the window
  Seconds(30),       // Window duration
  Seconds(10))       // Slide duration
```

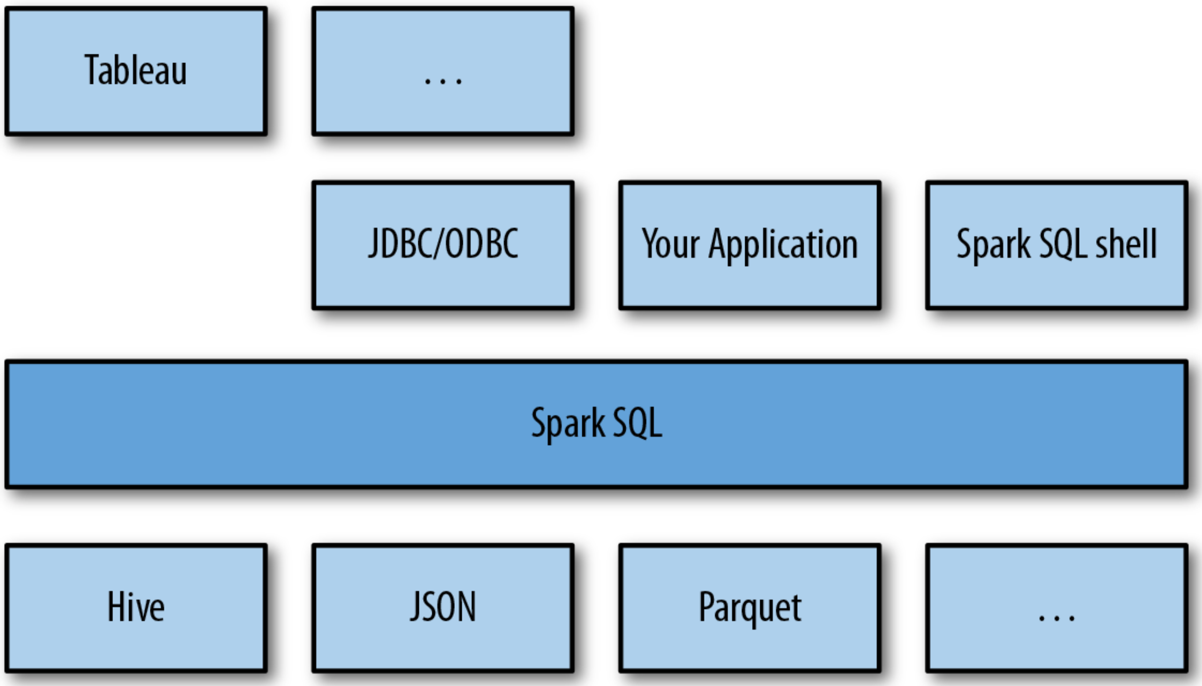


DEMO: Spark Streaming

- Simple streaming with PySpark

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 SQLfrom pyspark.sql
import HiveContext, Row

# Or if you can't include the hive requirementsfrom 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()

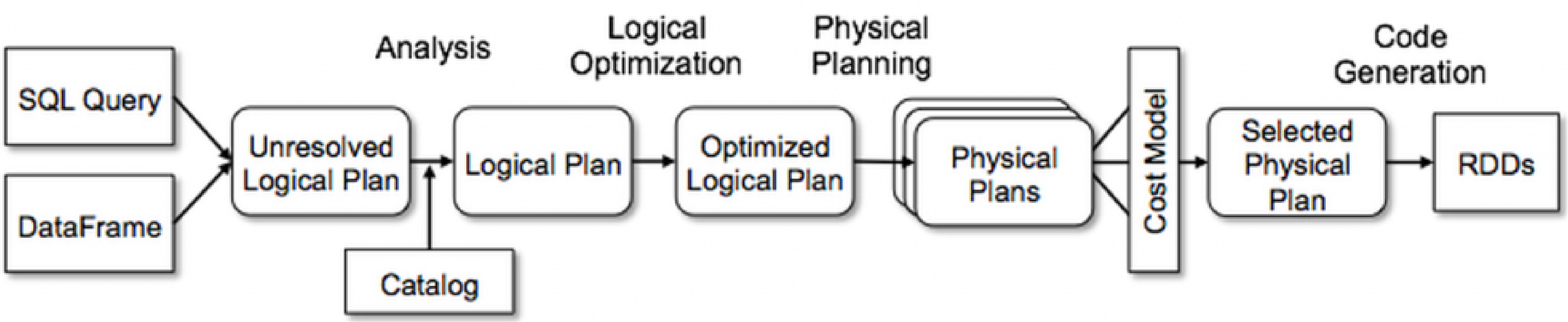
Using HiveContext

```
from pyspark.sql import HiveContext

hiveCtx = HiveContext(sc)

rows = hiveCtx.sql("SELECT key, value FROM mytable")

keys = rows.map(lambda row: row[0])
```

- saveAsTable(“TableName”)
- Hive format, text files, RCFiles, ORC, Parquet, Avro, protocol Buffers
- JDBC/ODBC with Trhift Server on Driver Node for BI Tools
- Beeline and spark-sql shells
- EXPLAIN SELECT … for execution plan

Catalyst: Query Optimizer

SELECT name
FROM (
    SELECT id, name
    FROM People) p
WHERE p.id = 1



- Analysis: map tables, columns, function, create a logical plan
- Logical Optimization: applies rules and optimize the plan
- Physical Planing: physical operator for the logical plan execution
- Cost estimation

DEMO: Using SparkSQL

- Simple SparkSQL querying
- Data Frames
- Data exploration with SparkSQL
- Connect from BI

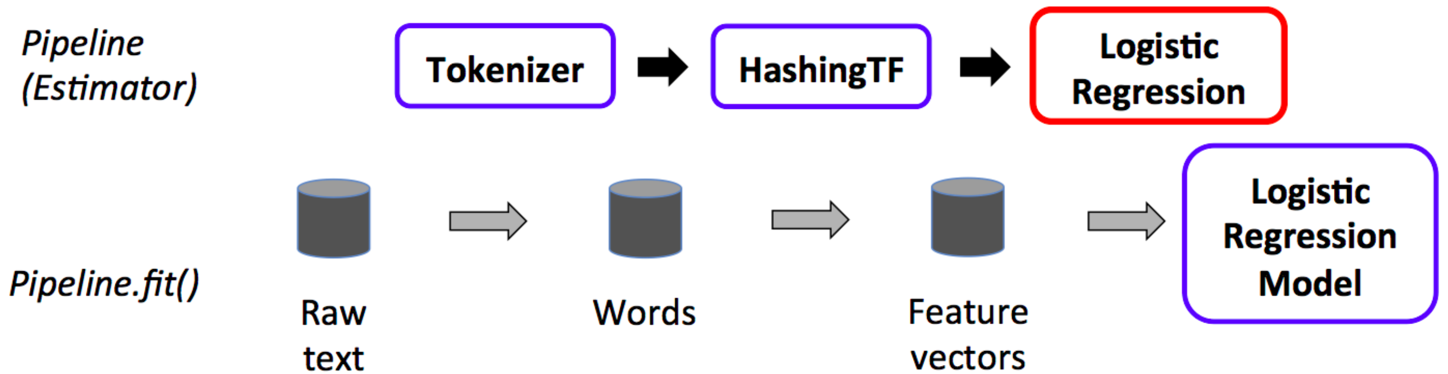
Spark ML

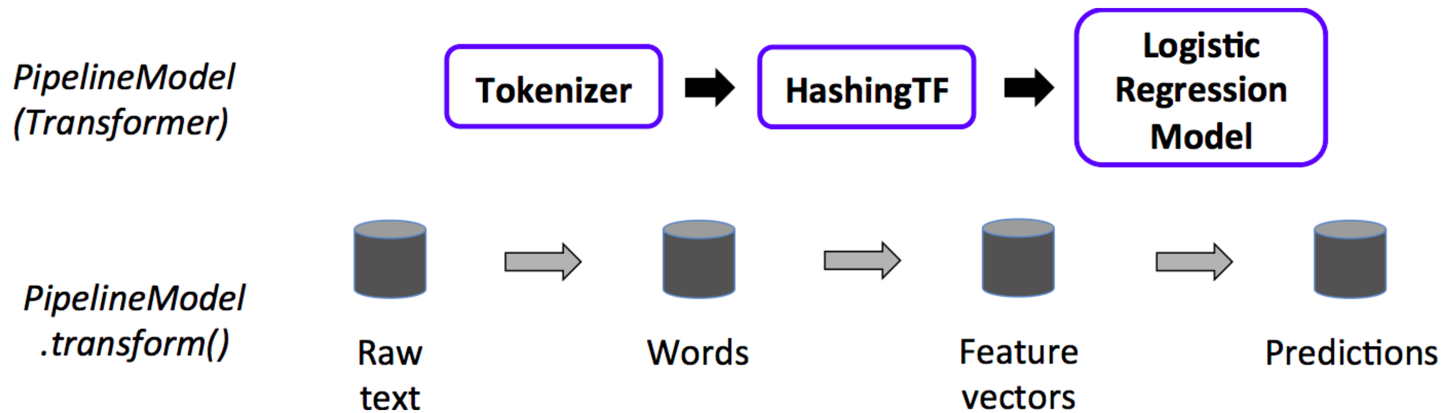
Spark ML

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

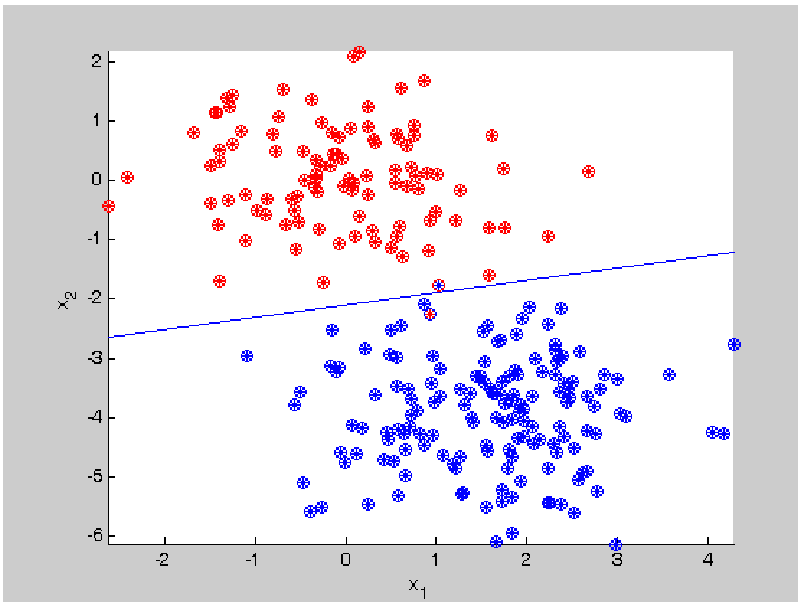
Pipeline Cmponents

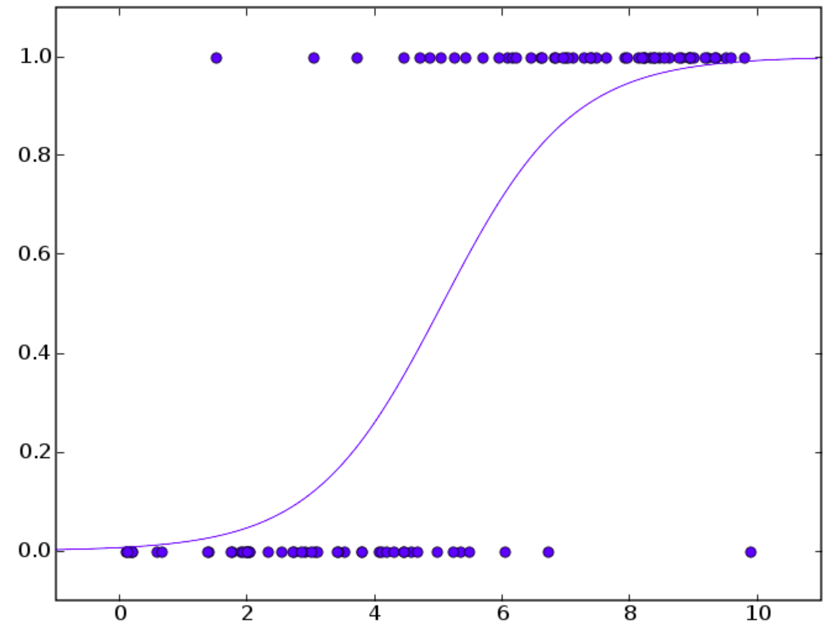
- DataFrame
- Transformer
- Estimator
- Pipeline
- Parameter





Logistic Regression





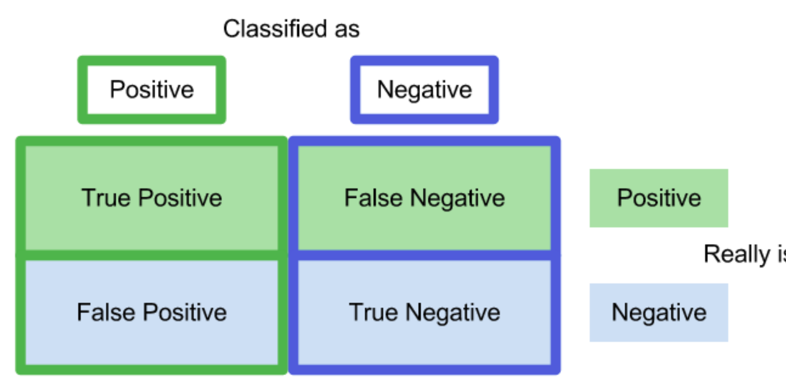
$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

Repeat $\{$

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update all θ_j)

$\}$

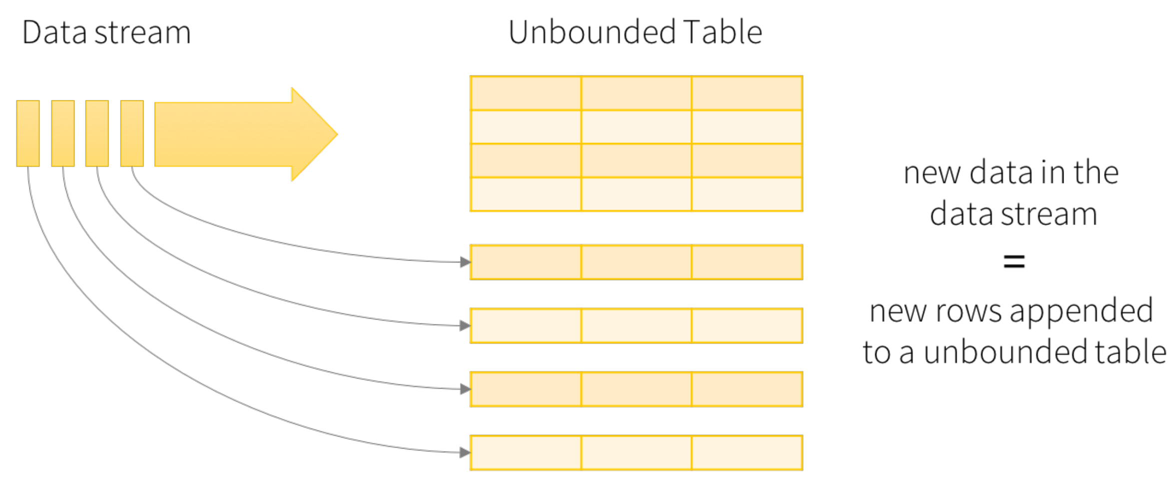


DEMO: Spark ML

- Training a model
- Data visualization

New in Spark 2.0

- **Unifying DataFrames and Datasets** in Scala/Java (compile time syntax and analysis errors). Same performance and convertible.
- **SparkSession**: a new entry point that supersedes SQLContext and HiveContext.
- **Machine learning pipeline persistence**
- **Distributed algorithms in R**
- **Faster Optimizer**
- **Structured Streaming**



Data stream as an unbounded table

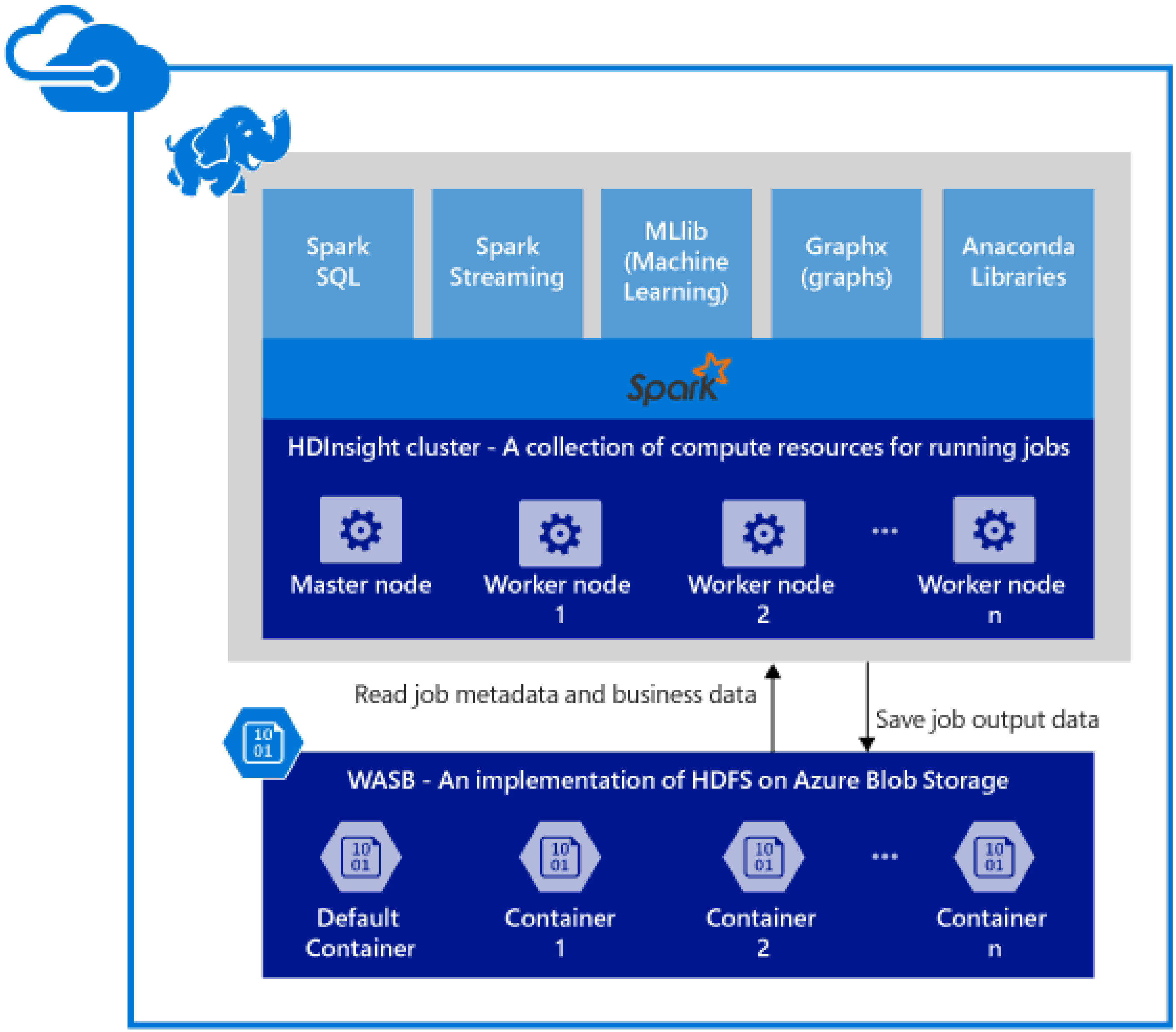
New in Spark 2.0

```
spark = SparkSession\
    .builder()\
    .appName("StructuredNetworkWordCount")\
    .getOrCreate()

# Create DataFrame representing the stream of input lines from connection to localhost:9999
lines = spark\
   .readStream\
   .format('socket')\
   .option('host', 'localhost')\
   .option('port', 9999)\
   .load()

# Split the lines into words
words = lines.select(
   explode(
       split(lines.value, ' ')
   ).alias('word')
)

# Generate running word count
wordCounts = words.groupBy('word').count()


# Start running the query that prints the running counts to the console
query = wordCounts\
    .writeStream\
    .outputMode('complete')\
    .format('console')\
    .start()

query.awaitTermination()
```

```
windowedCounts = words.groupBy(
    window(words.timestamp, '10 minutes', '5 minutes'),
    words.word
).count()
```

HDInsight: Spark

Spark in Azure



HDInsight benefits

- Ease of creating clusters (Azure portal, PowerShell, .Net SDK)
- Ease of use (noteboks, azure control panels)
- REST APIs (Livy: job server)
- Support for Azure Data Lake Store (adl://)
- Integration with Azure services (EventHub, Kafka)
- Support for R Server (HDInsight R over Spark)
- Integration with IntelliJ IDEA (Plugin, create and submit apps)
- Concurrent Queries (many users and connections)
- Caching on SSDs (SSD as persist method)
- Integration with BI Tools (connectors for PowerBI and Tableau)
- Pre-loaded Anaconda libraries (200 libraries for ML)
- Scalability (change number of nodes and start/stop cluster)
- 24/7 Support (99% up-time)

HDInsight Spark Scenarious

1. Streaming data, IoT and real-time analytics
2. Visual data exploration and interactive analysis (HDFS)
3. Spark with NoSQL (HBase and Azure DocumentDB)
4. Spark with Data Lake
5. Spark with SQL Data Warehouse
6. Machine Learning using R Server, Mllib
7. Putting it all together in a notebook experience
8. Using Excel with Spark

Q&A