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

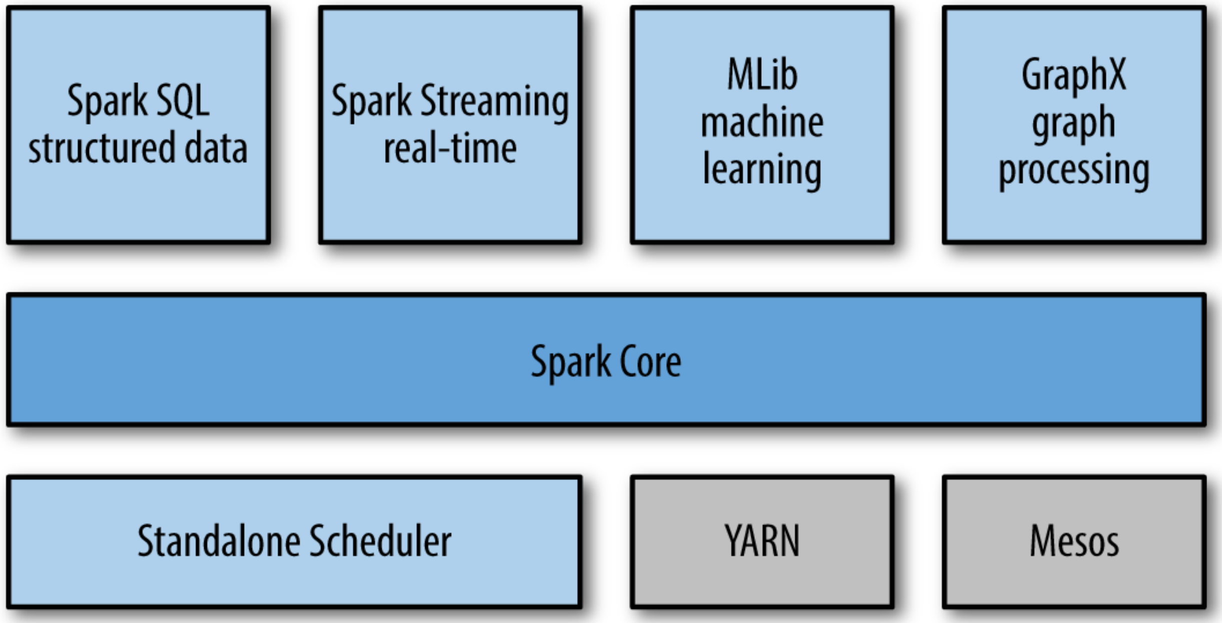
9+ years of developing for MS SQL Server

3+ years of architecting Big Data Solutions

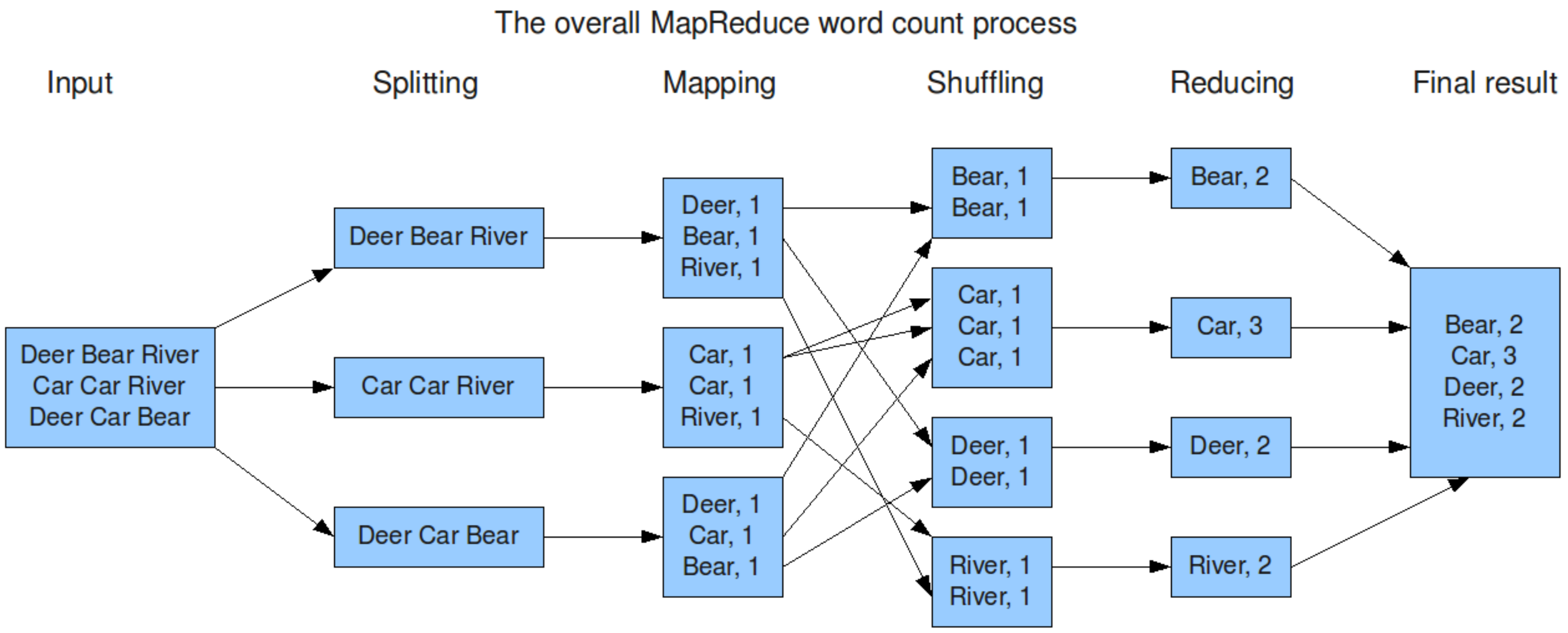
- DW/BI Architect and Technical Lead
- OLTP DB Performance Tuning
- Big Data Data Platform Architect

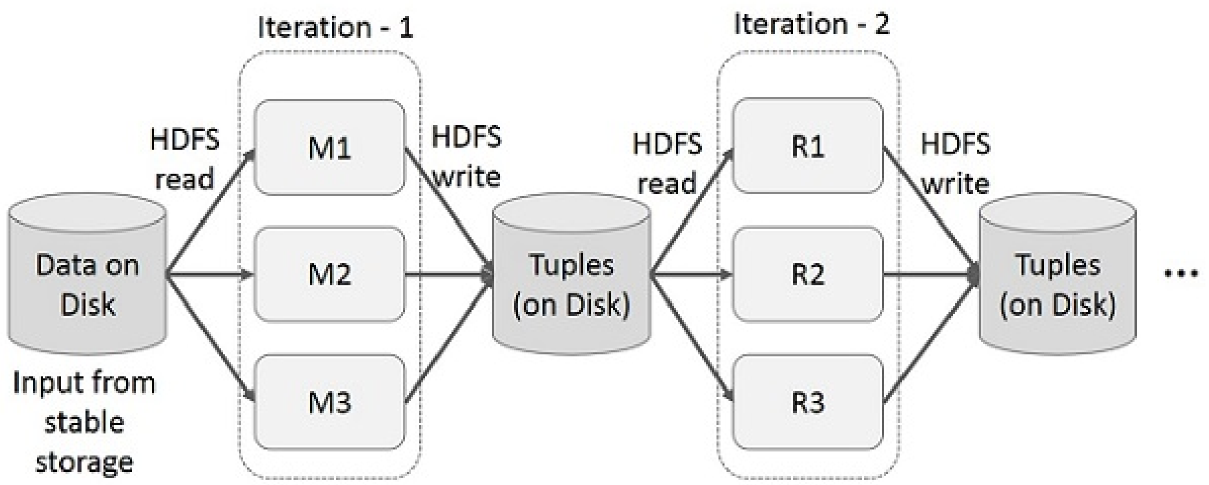
Spark Stack

- 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 Cassanda, Kafka, H2O
- First Apache release 2013, Aug 2016 v.2.0 has been released

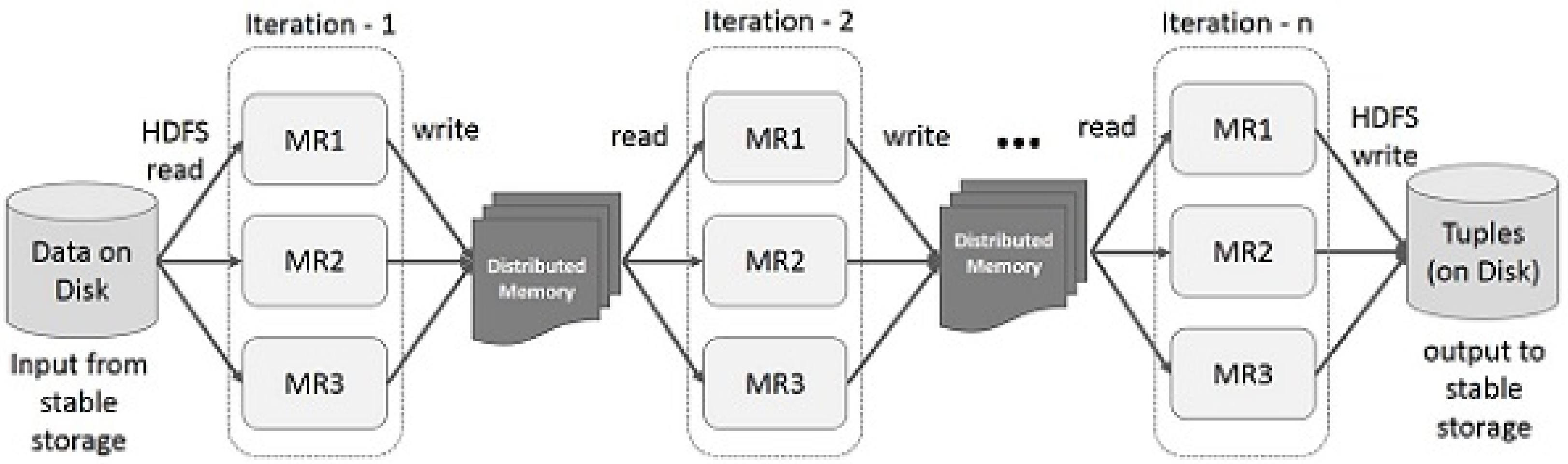


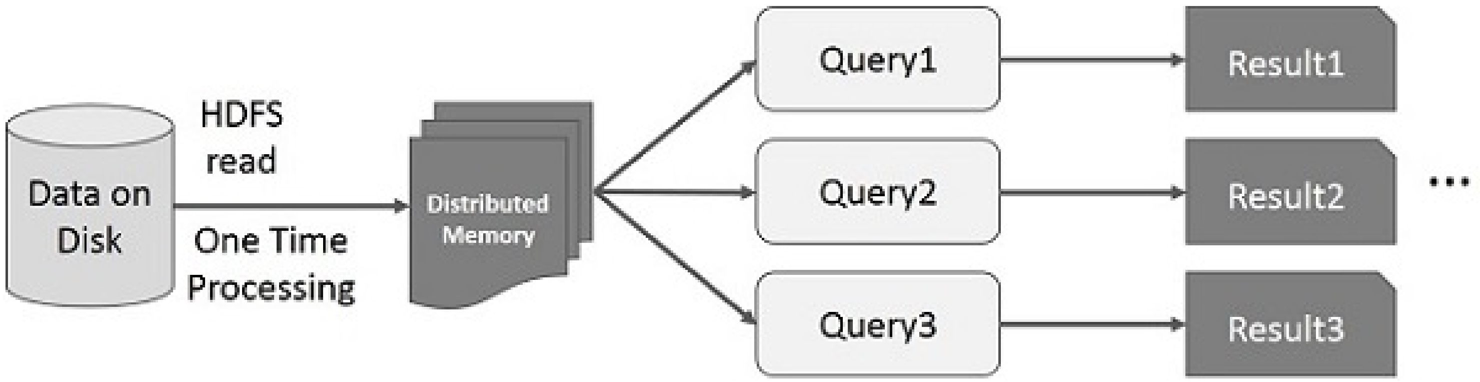
Map-reduce computations





In-memory map-reduce





Execution Model

Spark Execution

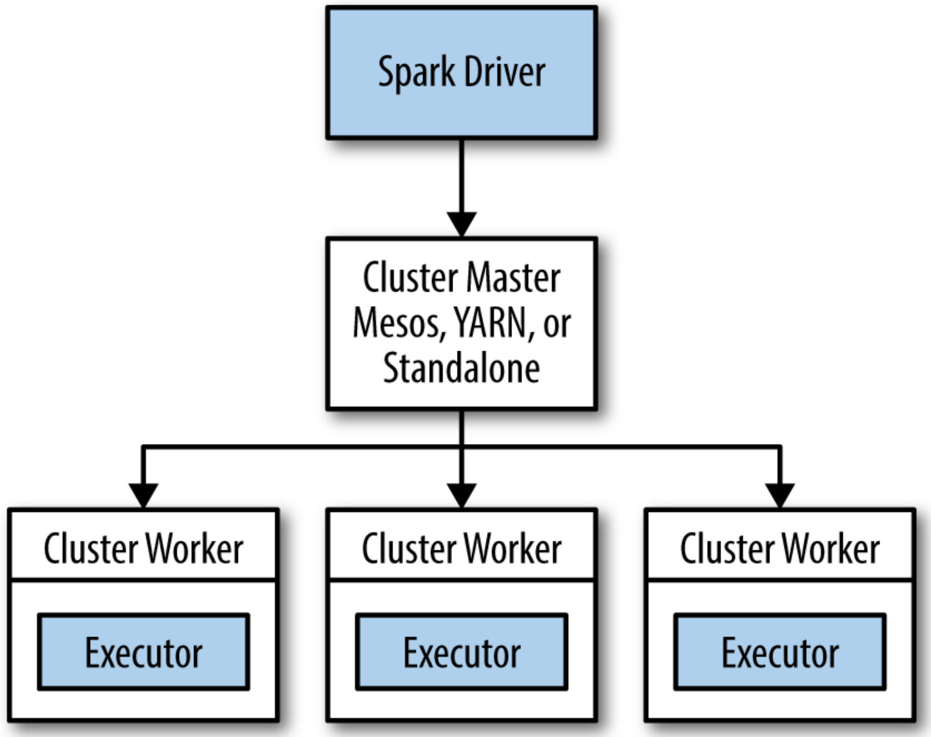
- Shells and Standalone application
- Local and Cluster (Standalone, Yarn, Mesos, Cloud)

Spark Cluster Architecture

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

Connect to Cluster

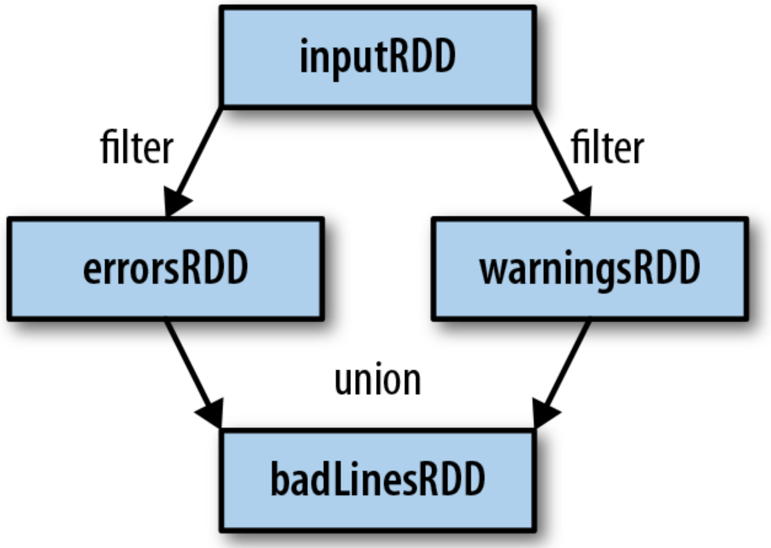
- Local
- SparkContext and Master field
- spark://host:7077
- Spark-submit



RDD: resilient distributed dataset

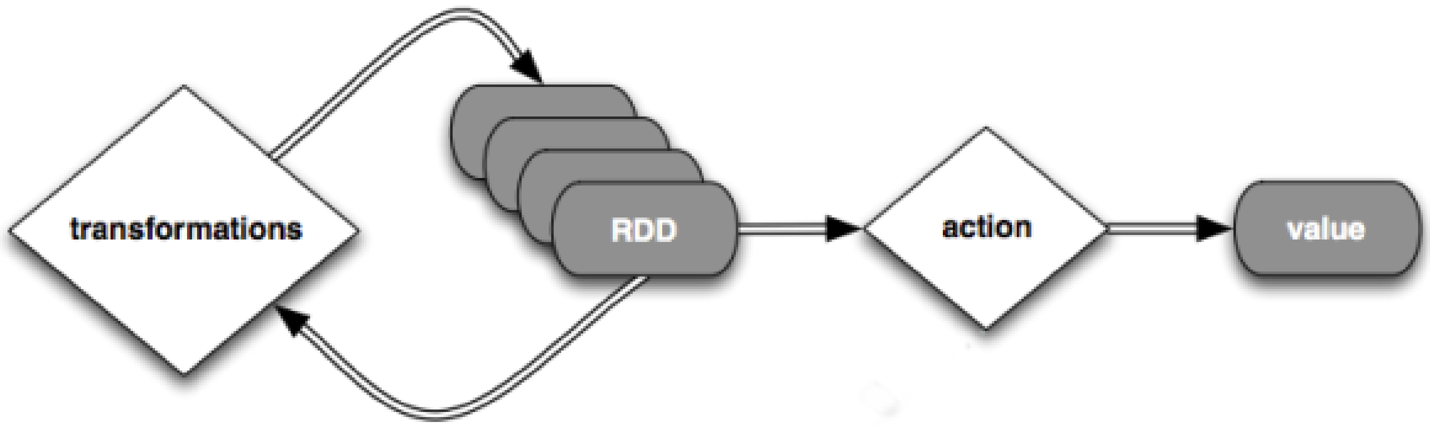
- Parallelized collections with fault-tolerant (Hadoop datasets)
- **Transformations** set new RDDs (filter, map, distinct, union, subtract, etc)
- **Actions** call to calculations (count, collect, first)
- Transformations are lazy
- Actions trigger transformations computation
- Broadcast Variables send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

```
from pyspart 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"
```

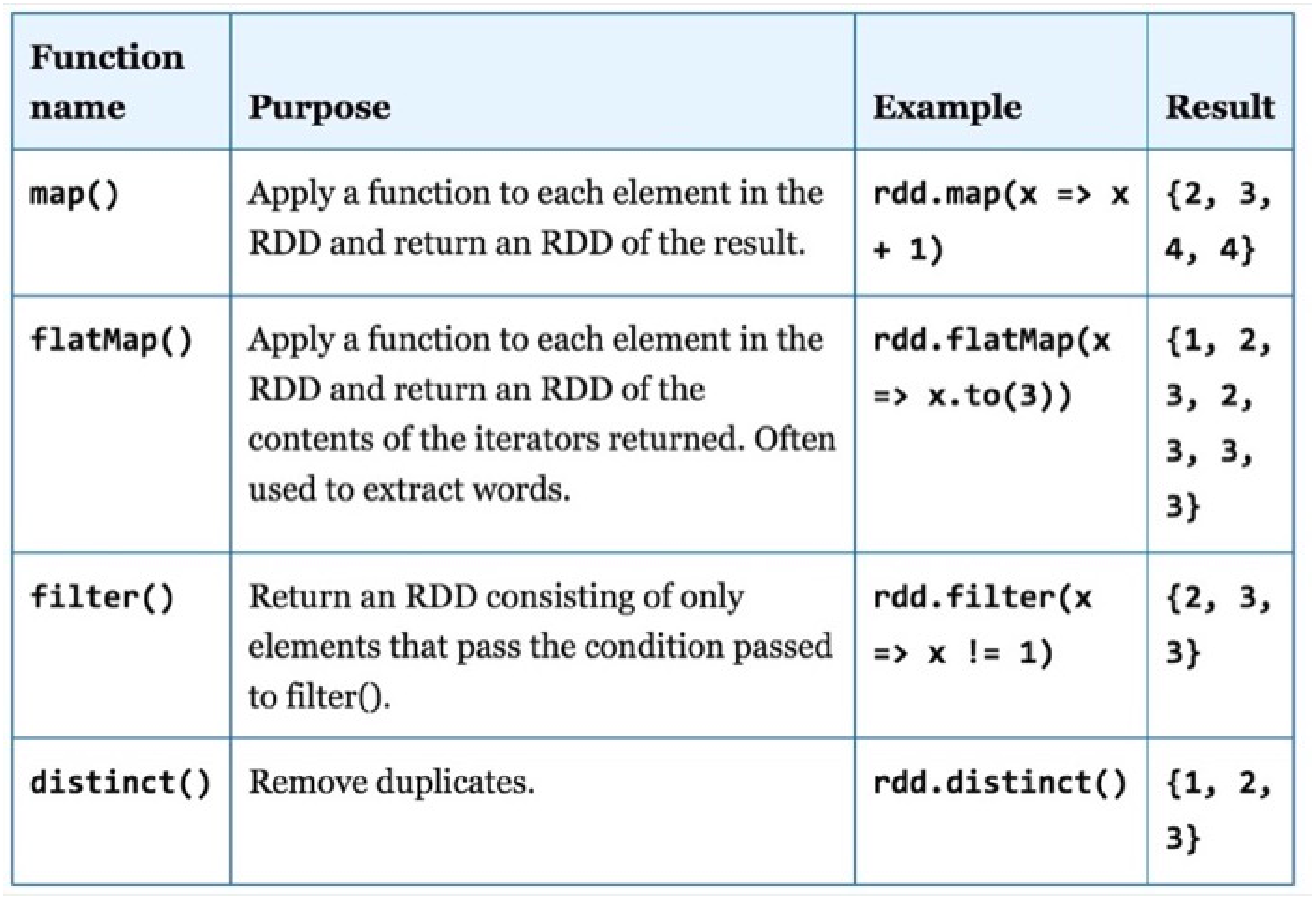


Spark program scenario

- Create RDD (loading external datasets, parallelizing a collection on driver)
- Transform
- Persist intermediate RDDs as results
- Launch actions

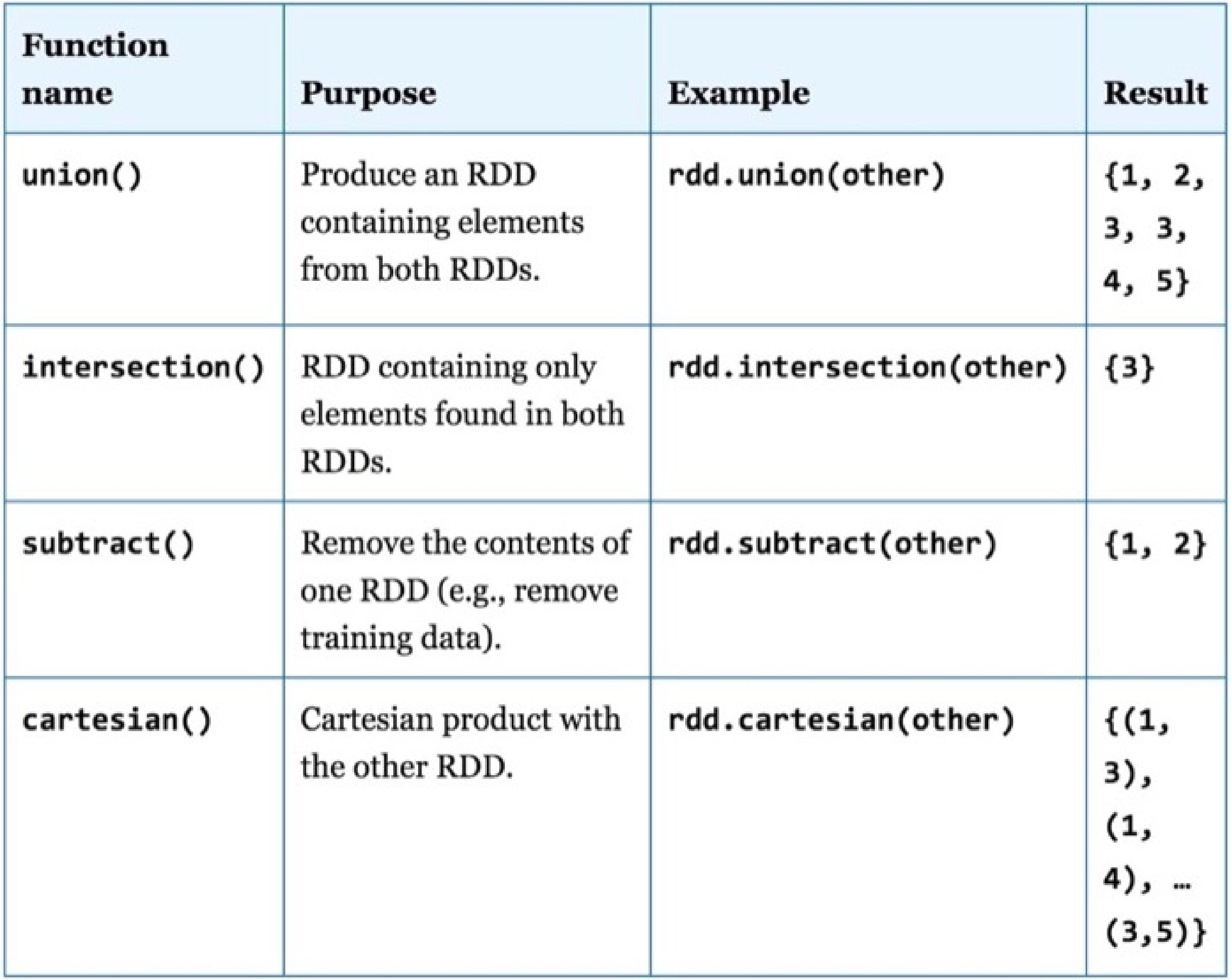


Transformations (1)



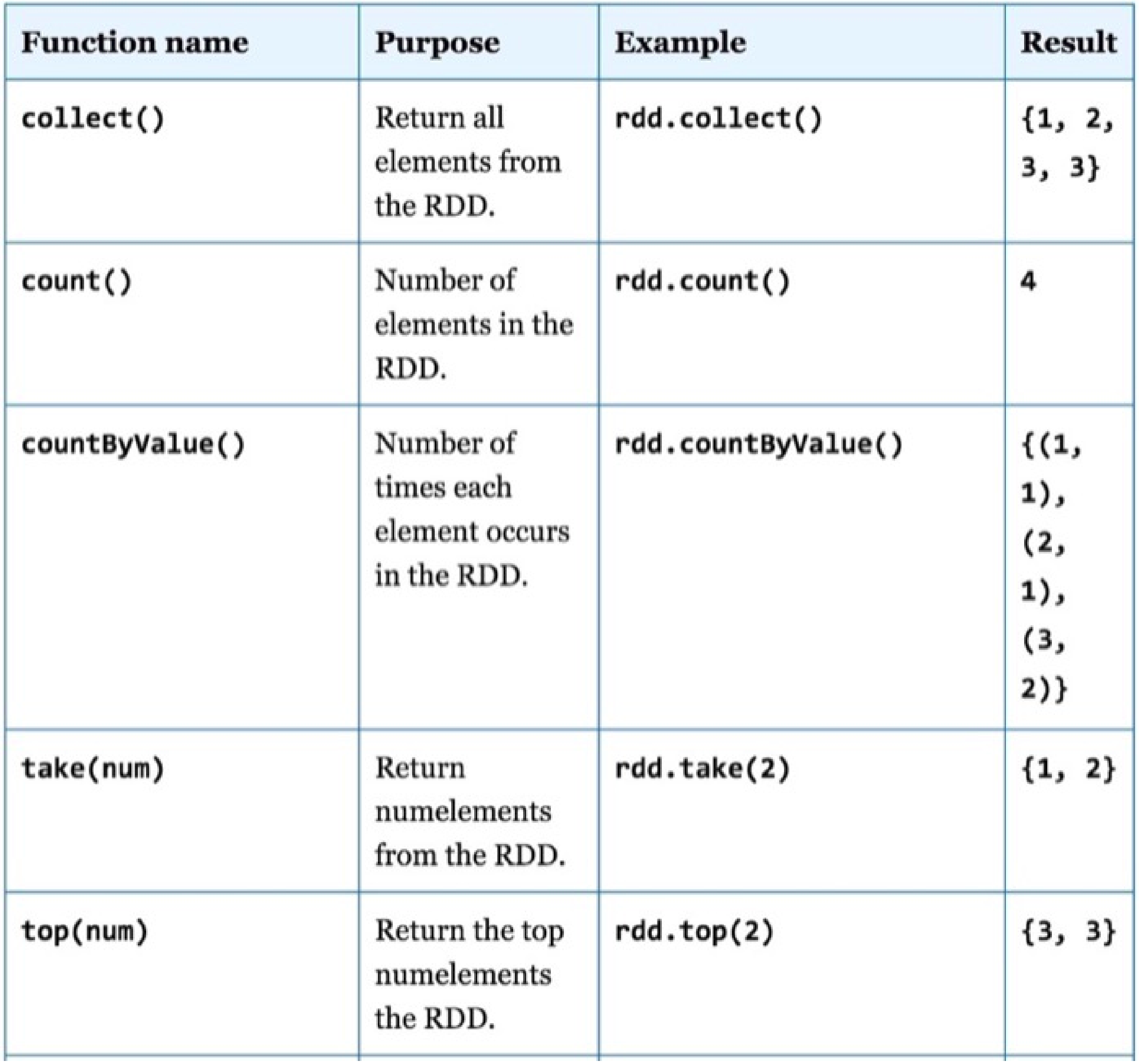
Function name	Purpose	Example	Result
`map()`	Apply a function to each element in the RDD and return an RDD of the result.	`rdd.map(x => x + 1)`	`{2, 3, 4, 4}`
`flatMap()`	Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned. Often used to extract words.	`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)



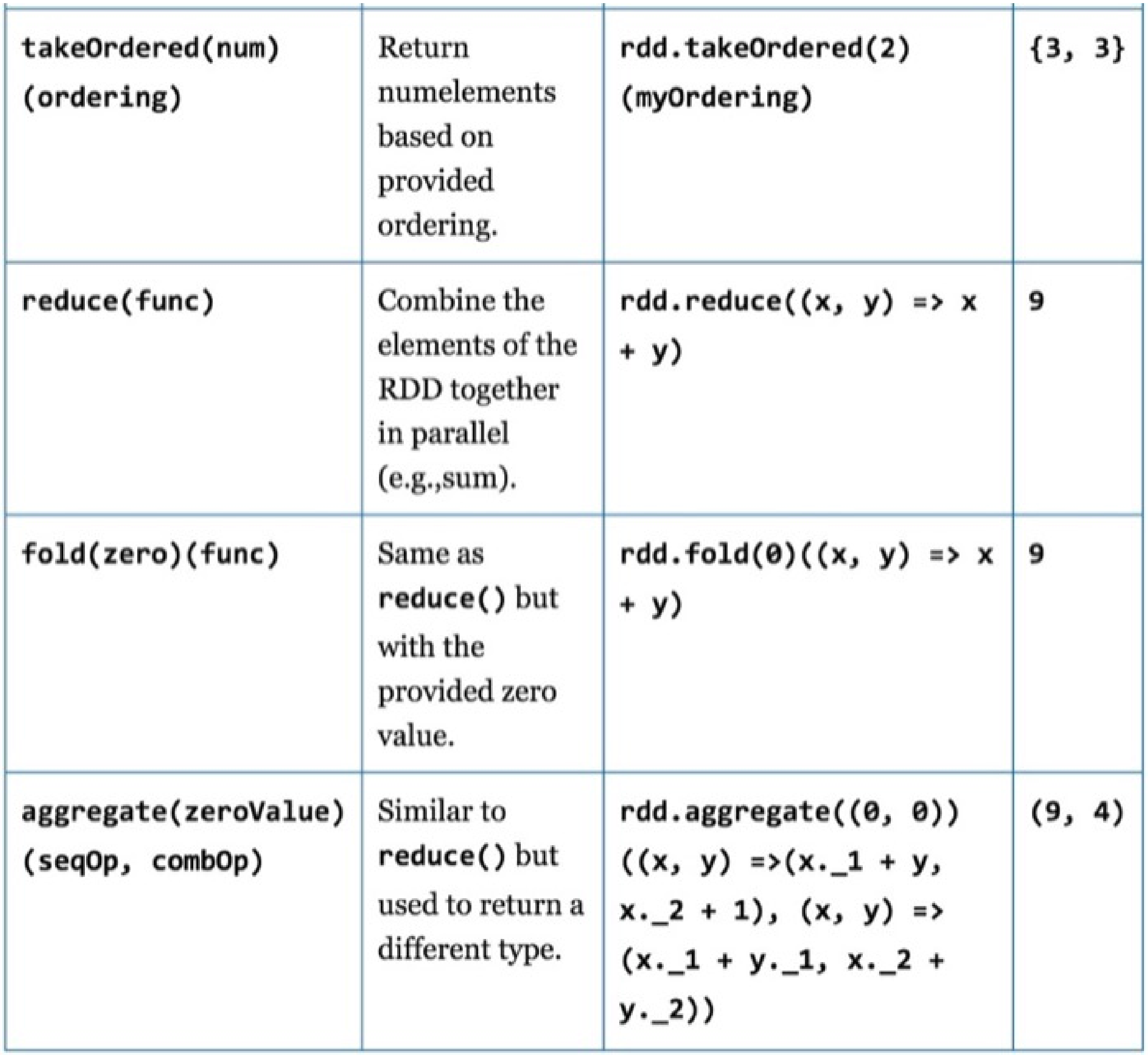
Function name	Purpose	Example	Result
`union()`	Produce an RDD containing elements from both RDDs.	`rdd.union(other)`	`{1, 2, 3, 3, 4, 5}`
`intersection()`	RDD containing only elements found in both RDDs.	`rdd.intersection(other)`	`{3}`
`subtract()`	Remove the contents of one RDD (e.g., remove training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

Actions (1)



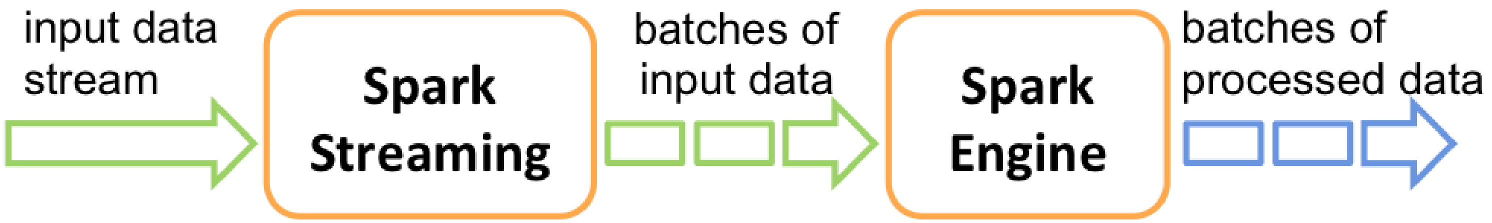
Function name	Purpose	Example	Result
`collect()`	Return all elements from the RDD.	`rdd.collect()`	`{1, 2, 3, 3}`
`count()`	Number of elements in the RDD.	`rdd.count()`	`4`
`countByValue()`	Number of times each element occurs in the RDD.	`rdd.countByValue()`	`{(1, 1), (2, 1), (3, 2)}`
`take(num)`	Return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

Actions (2)

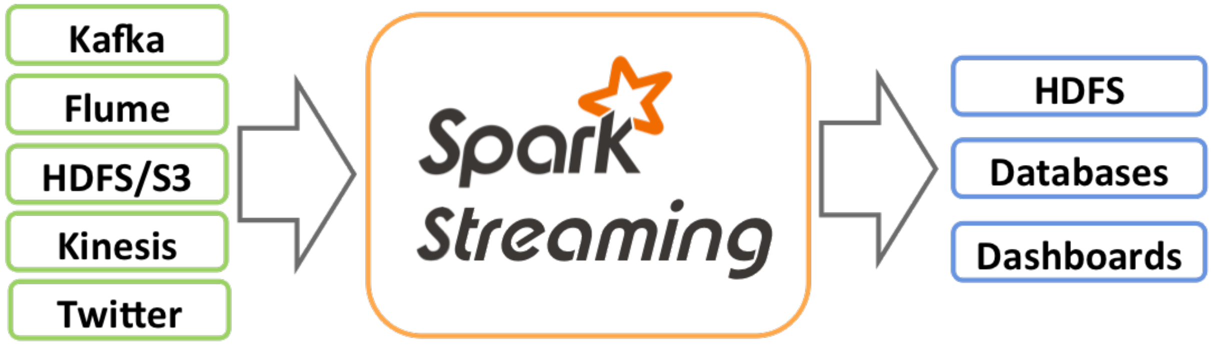


takeOrdered(num)(ordering)	Return numelements based on provided ordering.	rdd.takeOrdered(2)(myOrdering)	{3, 3}
reduce(func)	Combine the elements of the RDD together in parallel (e.g.,sum).	rdd.reduce((x, y) => x + y)	9
fold(zero)(func)	Same as **reduce()** but with the provided zero value.	rdd.fold(0)((x, 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)

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



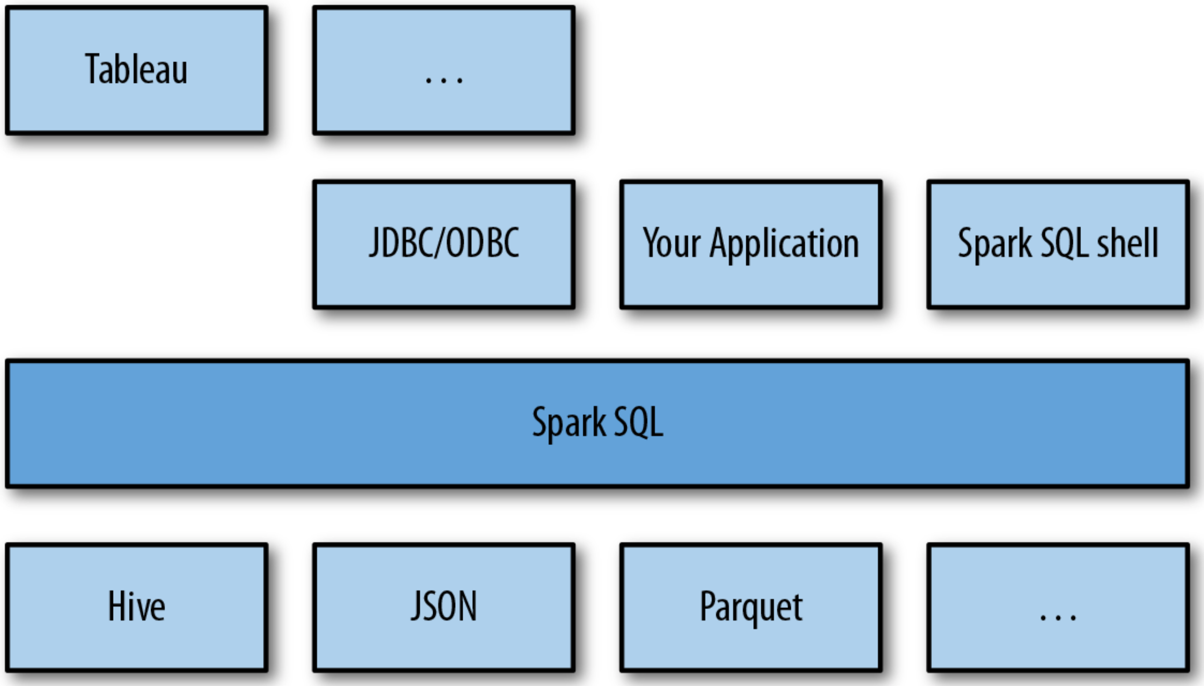
Spark Streaming Example

```
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

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()

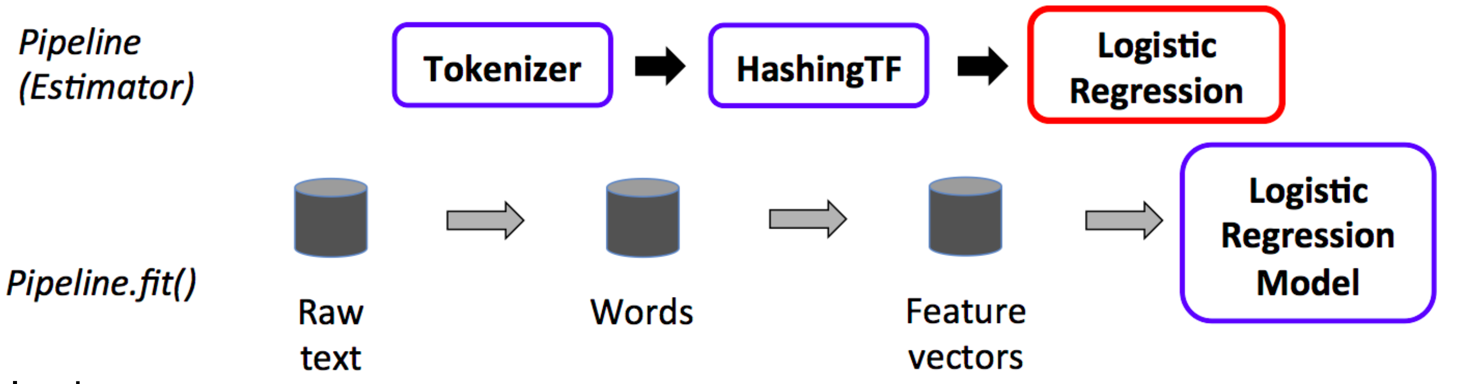
Spark ML

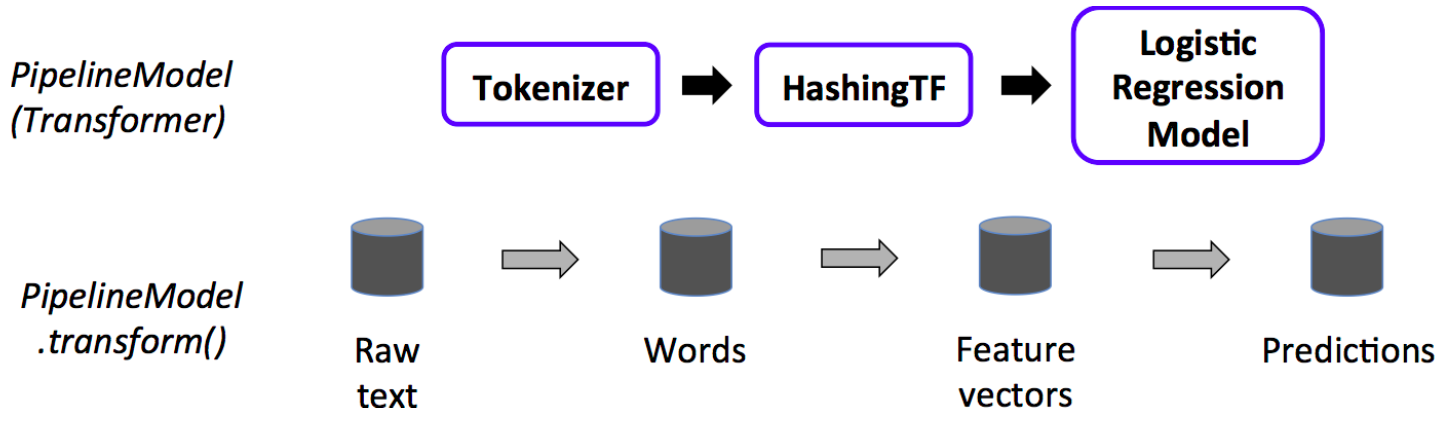
Spark ML

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

Pipeline Cmponents

- DataFrame
- Transformer
- Estimator
- Pipeline
- Parameter

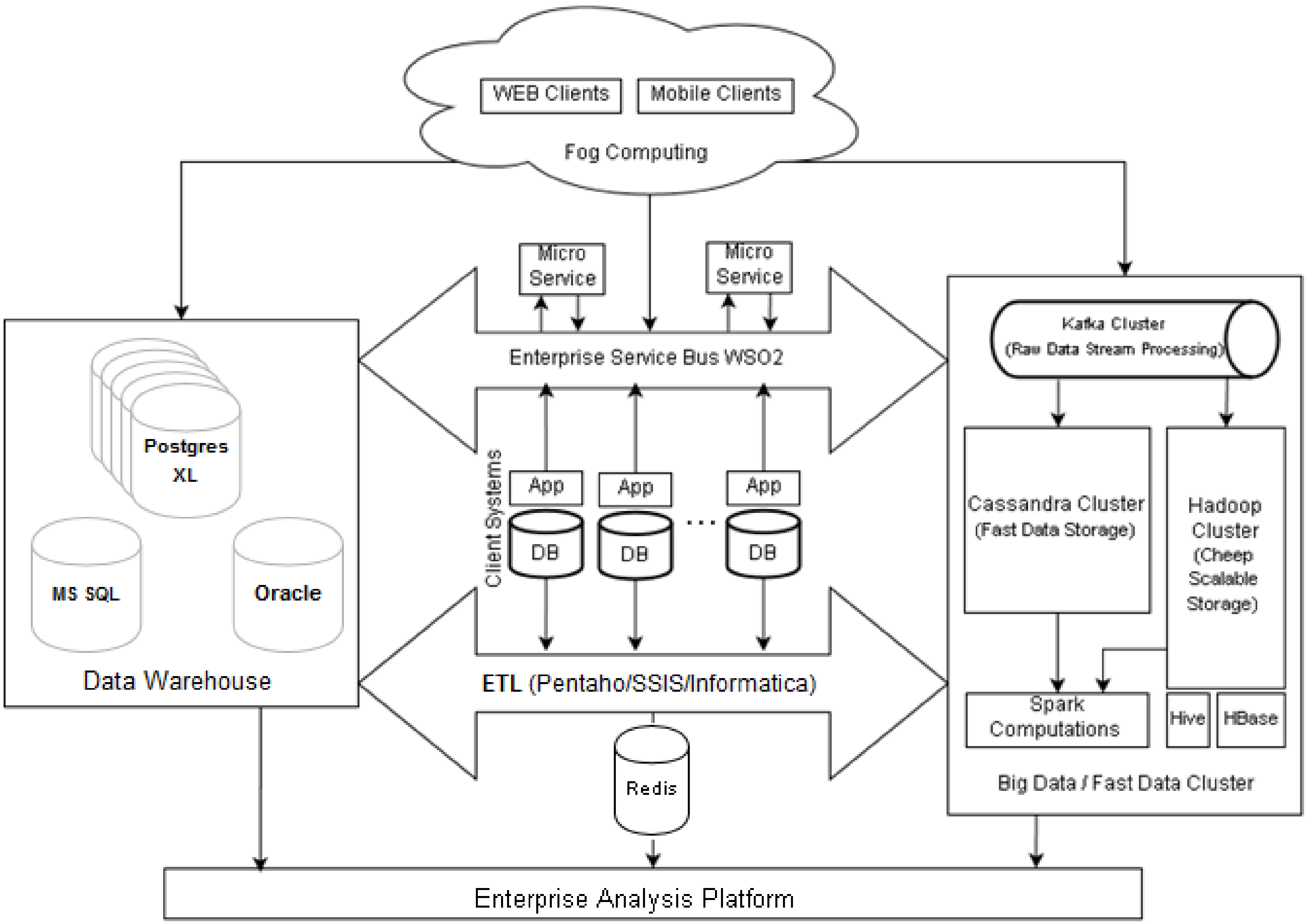



PipelineModel (Transformer)
Tokenizer
HashingTF
Logistic Regression Model
PipelineModel .transform()
Raw text
Words
Feature vectors
Predictions

DEMO: Spark

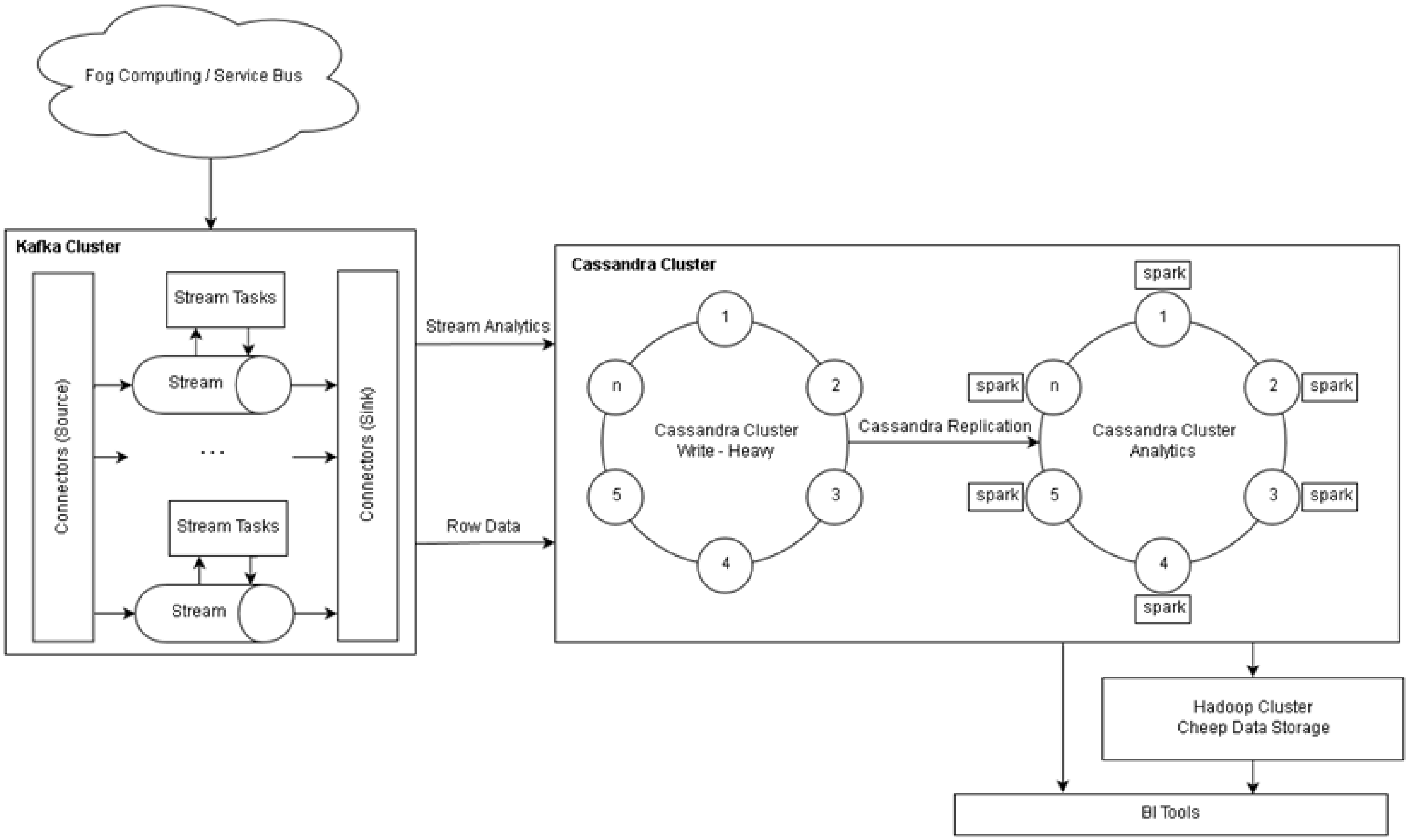
- Local Spark installation
- Shells and Notebook
- Spark Examples
- HDInsight Spark Cluster
- SSH connection to Spark in Azure
- Jupyter Notebook connected to HDInsight Spark
- Transformations
- ActionsSimple SparkSQL querying
- Data Frames
- Data exploration with SparkSQL
- Connect from BI
- Training a model
- Data visualization

Eleks Enterprice Platform

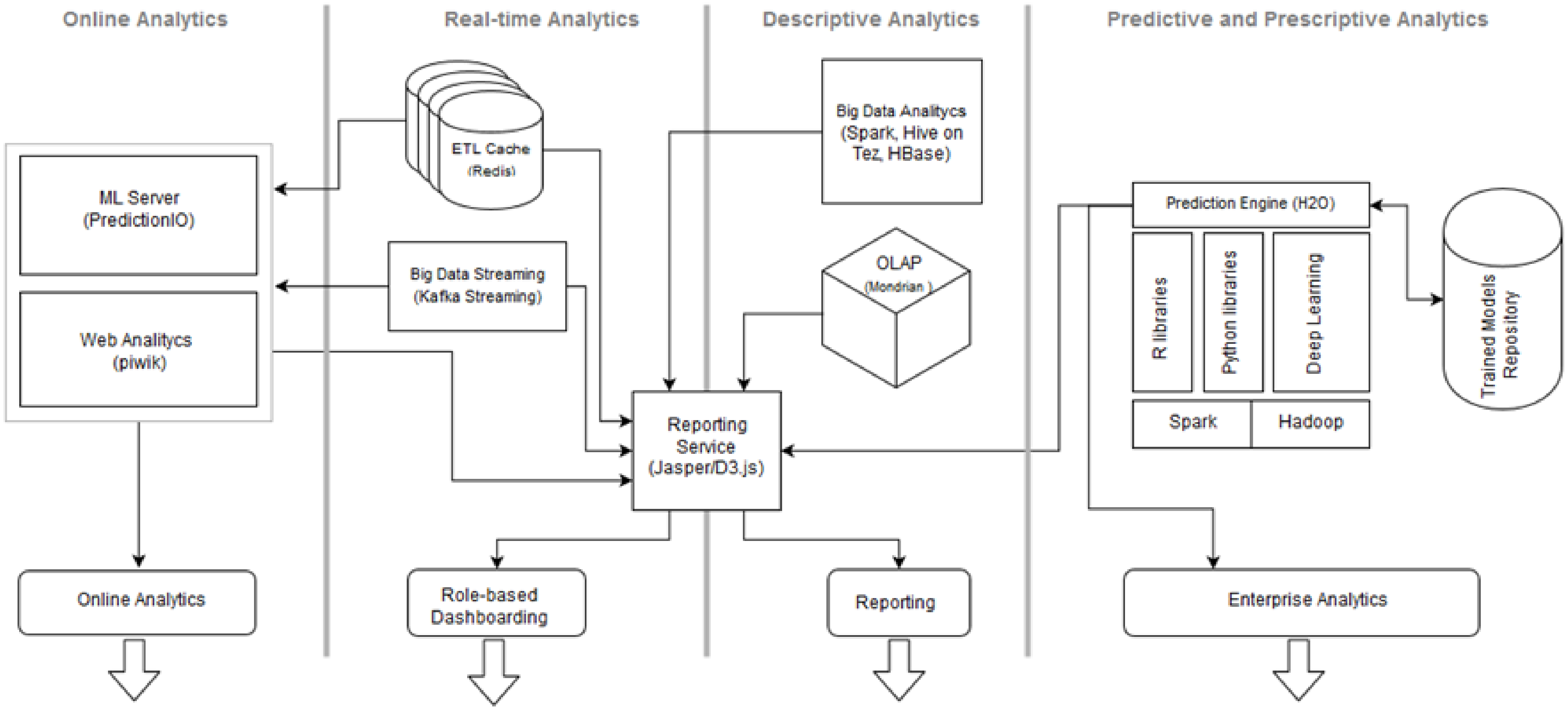


WEB Clients
Mobile Clients
Fog Computing
Micro Service
Micro Service
Enterprise Service Bus WSO2
Postgres XL
MS SQL
Oracle
Data Warehouse
Client Systems
App
App
App
DB
DB
DB
...
ETL (Pentaho/SSIS/Informatica)
Redis
Kafka Cluster (Raw Data Stream Processing)
Cassandra Cluster (Fast Data Storage)
Hadoop Cluster (Cheep Scalable Storage)
Spark Computations
Hive
HBase
Big Data / Fast Data Cluster
Enterprise Analysis Platform

Fast Data Processing



BI / DS Platform



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

Spark Environmens

- On-Premis, Cloudera, Hortonworks, DataBricks
- HDInsight, AWS, DataBriks Cloud
- Sparkling Water (H2O), prediction.io

Q&A