Big Data Analytics with Spark and Oscar BAO

Tamas Jambor, Lead Data Scientist at Massive Analytic

About me

- Building a scalable Machine Learning platform at MA
- Worked in Big Data and Data Science in the last few years
- Did PhD at UCL in Recommender Systems

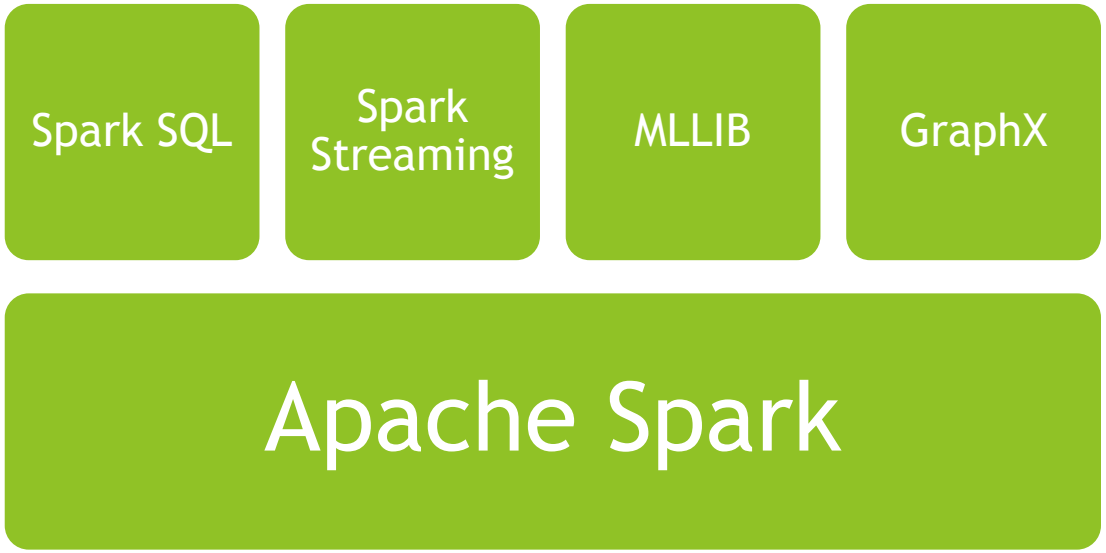
Overview

- Introduction to Spark
  - Architecture and internals
  - Setup and basic config
  - Spark SQL
  - Spart MLLIB
  - Spark Streaming
- Oscarbao
  - Analytics platform
  - Data Visualisation

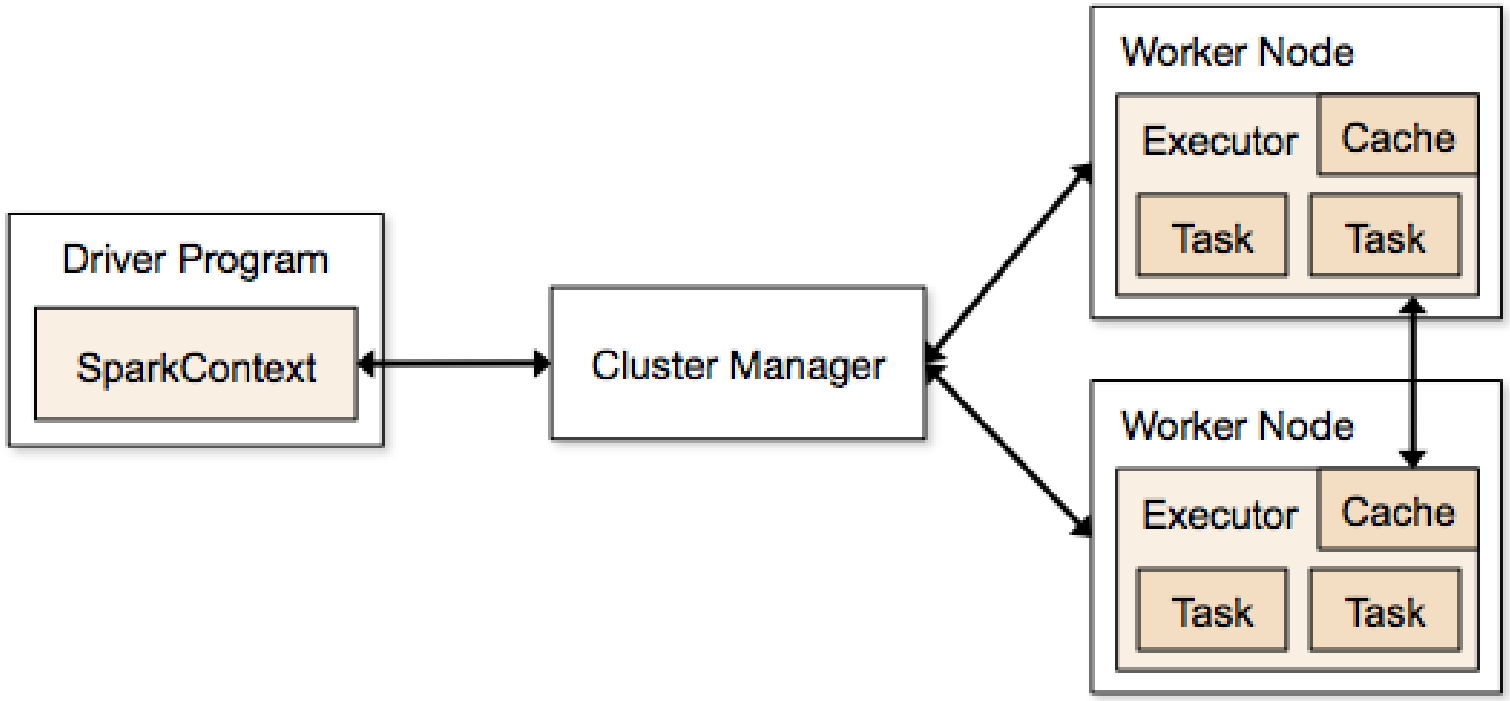
What is Spark?

- **Fast** and **Expressive** Cluster Computing Engine Compatible with Apache Hadoop
- Efficient
  - Up to 10x faster, 100x in memory
  - General execution graphs
  - In-memory storage
- Usable
  - 2-5x less code
  - Rich API in Java, Scala and Python
  - Interactive shell

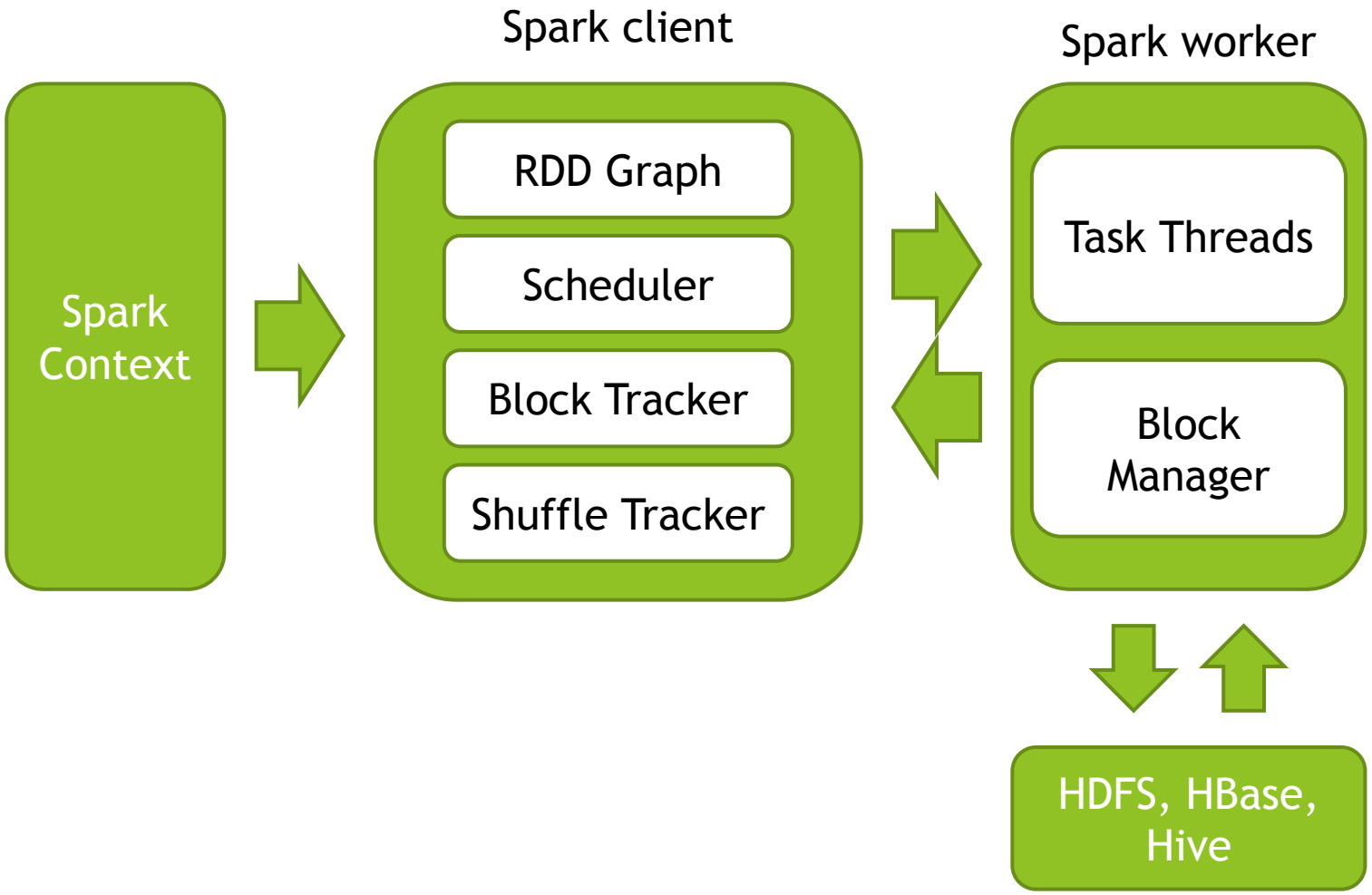
Apache Spark



Components


Driver Program
SparkContext
Cluster Manager
Worker Node
Executor
Cache
Task
Task
Worker Node
Executor
Cache
Task
Task

Spark Internals



Cluster managers

- Standalone
  - Ideal for running jobs locally
  - Uses Spark's own resource manager
- Apache Mesos
  - Dynamic partitioning between Spark and others
  - Scalable partitioning between multiple instances of Spark
- Hadoop YARN
  - Integrate well with other components of the Hadoop ecosystem
  - Supported by all Hadoop vendors (e.g Cloudera, Hortonworks)

Spark Setup

- Locally
  - Get prebuild version from (http://spark.apache.org/)
  - Setup configuration file (in ./conf/spark-env.sh)
  - Start spark master and worker (./sbin/start-all.sh)
  - Start spark interactive shell
    - Scala: (./bin/spark-shell --master spark://…)
    - Python (./bin/pyspark --master spark://..)
- EC2 start-up scripts are available (clone https://github.com/apache/spark)

Example Job

```
sc = SparkContext("spark://…","myJob", home, jars)

file = sc.textFile("hdfs://…")

errors = file.filter(lambda line: line.startswith("Error"))

errors.cache()

errors.count()
```

Resilient distributed datasets (RDD)

Action

Basic Operations - Transformations

- map(*func*)
  - data.map(lambda a: a+1)
- filter(*func*)
  - Data.filter(lambda a: a>1)
- mapPartitions(*func*)
  - Runs separately on each partition (block) of the RDD
- groupByKey()
  - When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs
- reduceByKey(*func)*
  - When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function *func*
- join(*otherDataset)*
  - When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key

Basic Operations - Actions

- reduce(*func*)
  - data.reduce(lambda a,b : a+b)
- collect()
  - Return all the elements of the dataset as an array at the driver program.
- take(*n*)
  - Return an array with the first *n* elements of the dataset.
- saveAsTextFile(*path*)
  - Write the elements of the dataset as a text file.

Broadcast Variables and Accumulators

- Broadcast variables
  - To Keep a read-only variable cached on each machine rather than shipping a copy of it with tasks
  - broadcastVar = sc.broadcast([1,2,3])
- Accumulators
  - Variables that are only added to through an associative operation and can therefore be efficiently supported in parallel
  - accum = sc.accumulator(0)
  - sc.parallelize([1, 2, 3, 4]).foreach(lambda x: accum.add(x))

Spark SQL

- Enables loading and querying structured data in Spark
  - From Hive:

    ```
    c = HiveContext(sc)
    rows = c.sql("select date, vale from hivetable")
    rows.filter(lambda r: r.value > 2013).collect()
    ```

  - From JSON:

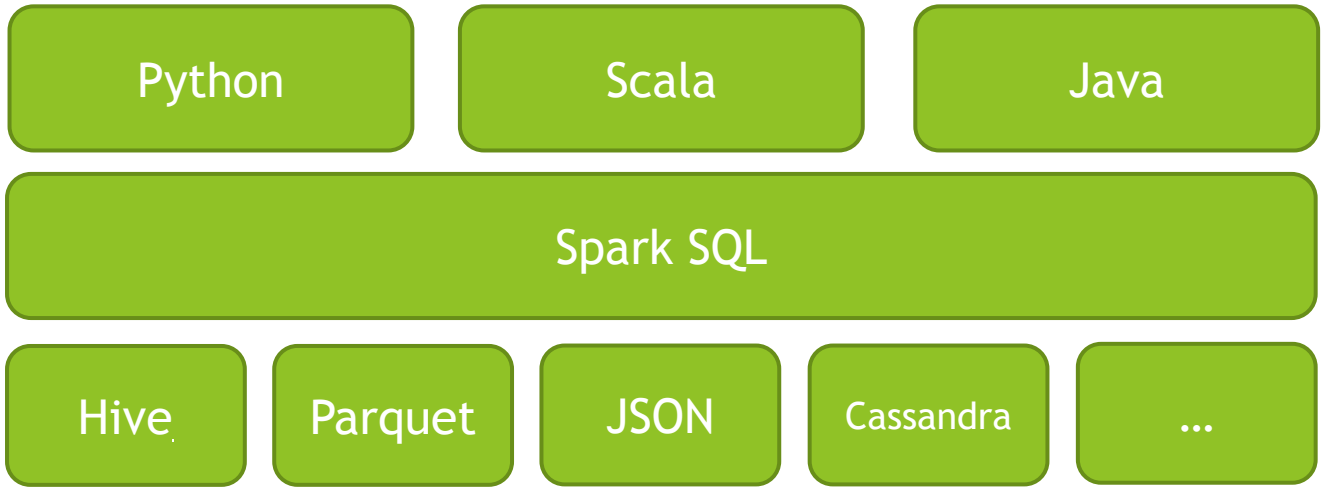
    ```
    c.jsonFile("account_info.json").registerAsTable("accounts")
    c.sql("select surename, address.postcode from accounts")
    ```

account_info.json:

```
{"first_name": "Tamas"
 "surname": "Jambor"
        "address": {
                "postcode": "N7 9UP"
        }
}
```

Spark SQL

- Integrates closely with Spark's language APIs
  - c.registerFunction("hasSpark", lambda text: "Spark" in text)
  - c.sql("select * from accounts where hasSpark(text)")
- Uniform interface for data access



Spark MLLIB

- Standard library of distributed machine learning algorithms
- Provides some of the most common machine learning algorithms
  - Basic Statistics
  - Classification and regression
    - Linear models
    - Naïve Bayes
  - Collaborative Filtering
  - Clustering
  - Dimensionality reduction
  - Optimisers

Spark MLLIB

- Now includes 15+ algorithms
  - New in 1.0
    - Decision trees
    - SVD, PCA, L-BFGS (limited memory parameter estimation method)
  - New in 1.1:
    - Non-negative matrix factorization, ALS
    - Support for common statistical functions
      - Sampling, correlations
      - Statistical hypothesis testing
  - New in 1.2 (to be released):
    - Random forest

Spark MLLIB Example (Classification)

```
from pyspark.mllib.classification import LogisticRegressionWithSGD
from pyspark.mllib.regression import LabeledPoint
from numpy import array

 # Load and parse the data

def parsePoint(line):
    values = [float(x) for x in line.split(' ')]
    return LabeledPoint(values[0], values[1:])

data = sc.textFile("data/mllib/sample_svm_data.txt")
parsedData = data.map(parsePoint)

# Build the model

model = LogisticRegressionWithSGD.train(parsedData)

# Evaluating the model on training data

labelsAndPreds = parsedData.map(lambda p: (p.label,
model.predict(p.features)))
trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() /
(parsedData.count())
print("Training Error = " + (trainErr))
```

Spark MLLIB Example (Clustering)

```
from pyspark.mllib.clustering import Kmeans
from numpy import array
from math import sqrt


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

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 2, maxIterations=10, runs=10,
initializationMode="random")


# Evaluate clustering by computing Within Set Sum of Squared Errors
def error(point): center = clusters.centers[clusters.predict(point)] return sqrt(([x**2 for x in
(point - center)]))

WSSSE = parsedData.map(lambda point: error(point)).reduce(lambda x, y: x + y)

print("Within Set Sum of Squared Error = " + (WSSSE))
```

Spark Streaming

- Many big-data applications need to process large data streams in real-time
  - Web-site monitoring
  - Fraud detection
- Batch model for iterative ML algorithms and data processing
- Extends Spark for doing big data stream processing
  - Efficiently recover from failures

Integration with Batch Processing

- Many environments require processing same data in live streaming as well as batch post-processing
- Existing frameworks cannot do both
  - Either, stream processing of 100s of MB/s with low latency
  - Or, batch processing of TBs of data with high latency

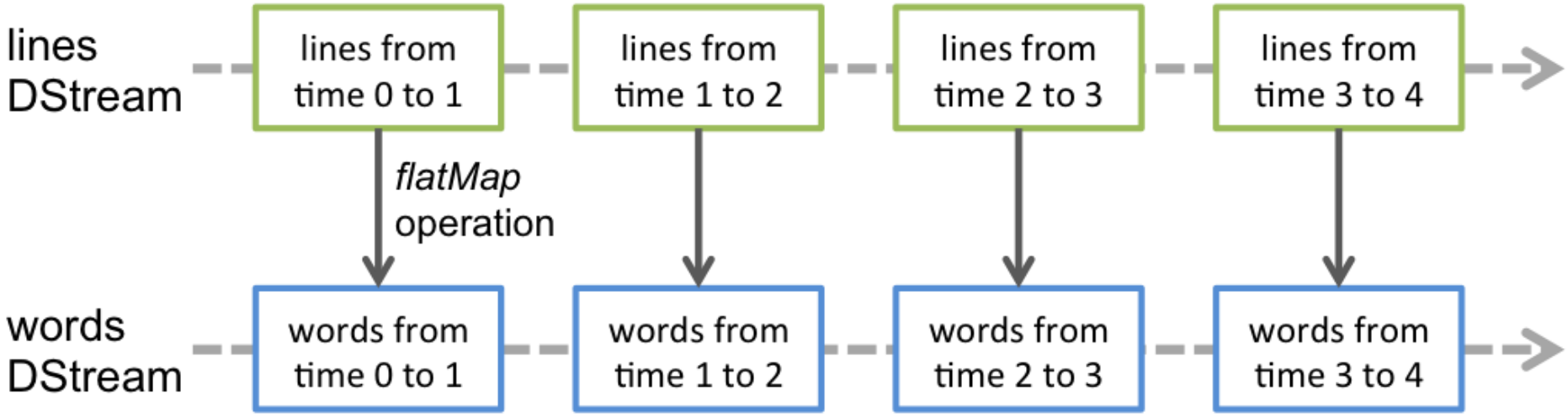
Spark Streaming Workflow

Spark Streaming Workflow

Spark Streaming Example (Scala)

```
import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._

val conf = new SparkConf().setMaster("spark://…").setAppName("NetworkWordCount")
val ssc = new StreamingContext(conf, Seconds(1))

val lines = ssc.socketTextStream("localhost", 9999, StorageLevel.MEMORY_AND_DISK_SER)

val words = lines.flatMap(_.split(" "))

val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)

wordCounts.print()

ssc.start()
ssc.awaitTermination()
```

Spark Streaming – Word count

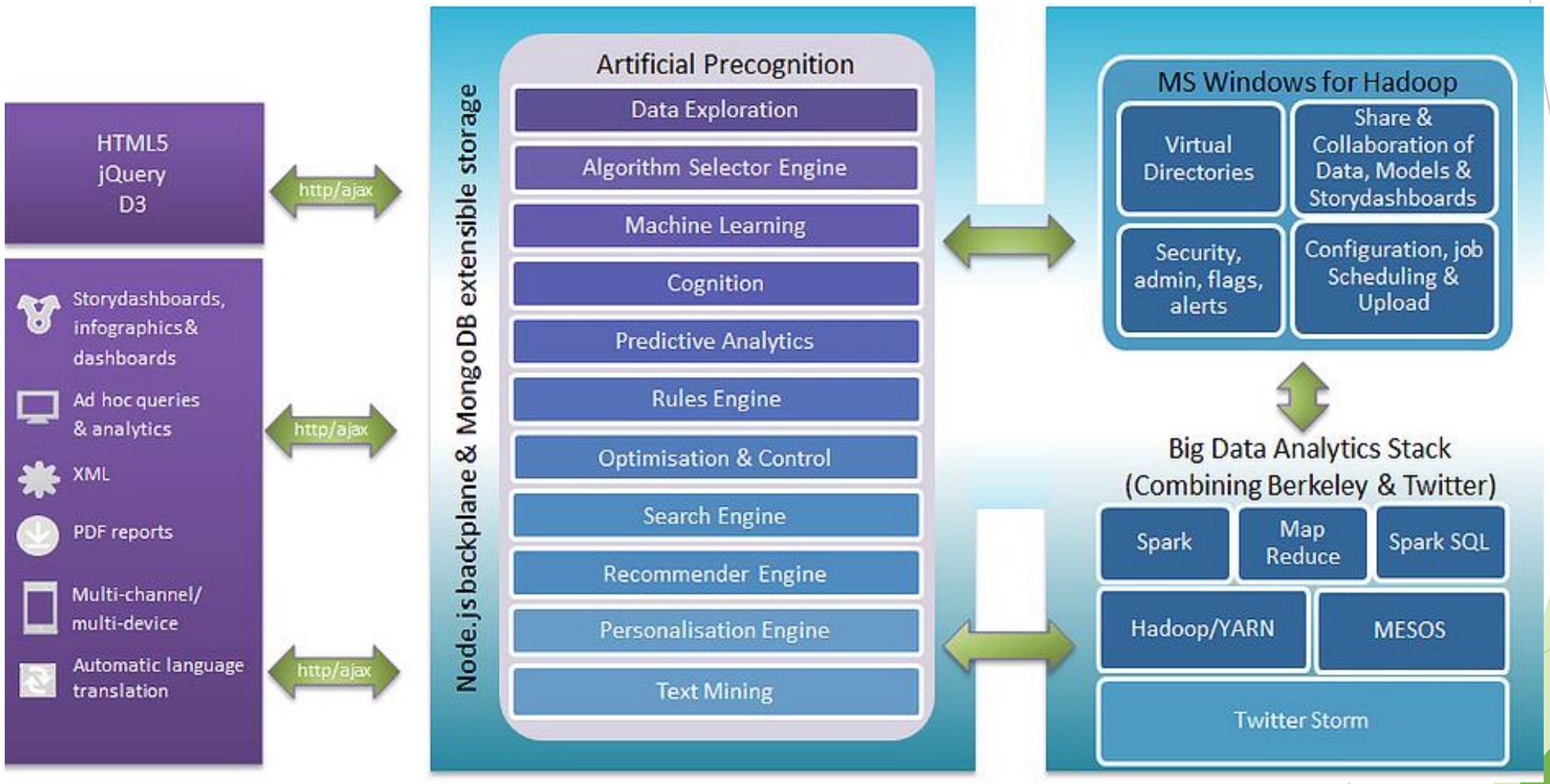


Demo

Oscar BAO

- Business analytics optimisation
- Automates certain data science processes
- Runs on Spark
- Visualisation Interface

Oscarbao Architecture



Modules

- Understanding
  - Data exploration
  - Rules engine
  - Cognition
- Predicting
  - Predictive analytics
  - Text mining
  - Personalisation
- Optimising
  - Algorithm selection engine
  - Recommended algorithm
  - Optimisation and control

Big Data Workflow for Predictive Analytics

Data Ingestion	• Parsing • Cleansing
Data Pre-processing	• Transformation • Statistics
Model Training	• Parameter tuning • Performance
Algorithm Selection	• Evaluation • Best model for the purpose
Visualisation	• Representing results • Insights
Reuse	• Store model • Apply to new data

Demo