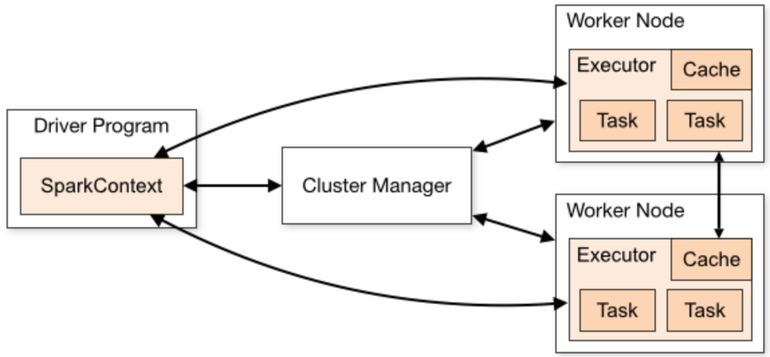
Welcome to Apache Spark

Architecture

A Spark program consists of a driver application and worker programs



- Worker nodes run on different machines in a cluster, or in local threads.
- Data is distributed among workers.

Spark Context

The `SparkContext` contains all of the necessary info on the cluster to run Spark code.

In [1]:
```
from pyspark import SparkContext, SparkConf

conf = SparkConf().setAppName('spark-app').setMaster('local[*]')
sc = SparkContext.getOrCreate(conf=conf)

sc
```

Out[1]:

SparkContext

Spark UI

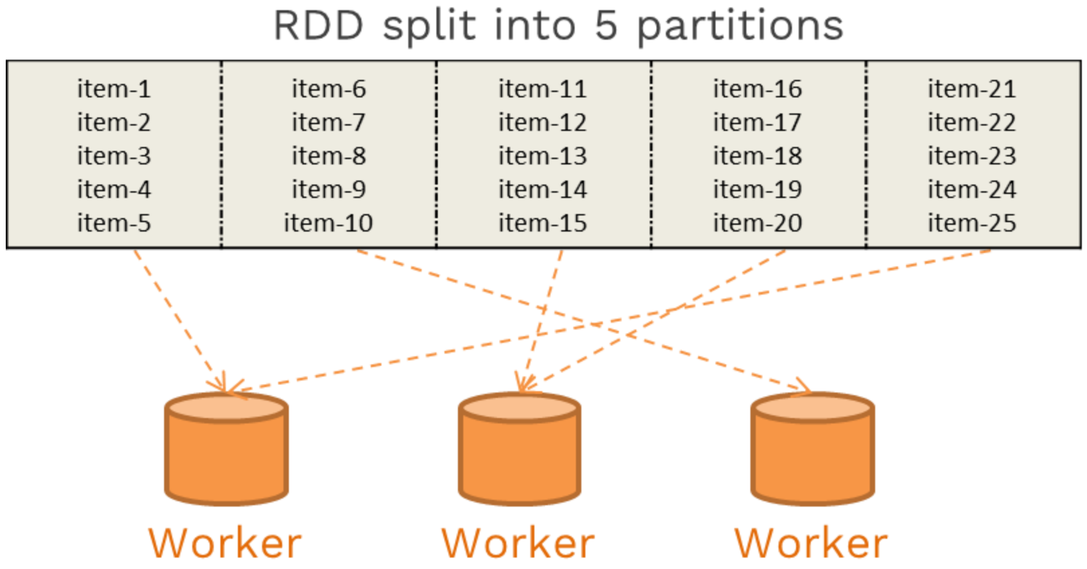
Version
`v2.2.1`

Master
`local[*]`

AppName
`spark-app`

Resilient Distributed Dataset

A partitioned collection of objects spread accross a cluster, stored in memory or on disk.



Number of partitions is defined by the programmer.

More partitions = more parallelism

3 ways of creating a RDD

- by parallelizing an existing collection

In [2]:
```
array = range(10)
array
```

Out[2]:
```
range(0, 10)
```

In [3]:
```
rdd = sc.parallelize(array)
rdd
```

Out[3]:
```
PythonRDD[1] at RDD at PythonRDD.scala:48
```

3 ways of creating a RDD

- from files in a storage system

In [4]:
```
titanic = sc.textFile('data/titanic.csv')
titanic
```

Out[4]:
```
data/titanic.csv MapPartitionsRDD[3] at textFile at <unknown>:0
```

In [5]:
```
titanic.take(3)
```

Out[5]:
```
['PassengerId,Survived,Pclass,Name,Sex,Age,SibSp,Parch,Ticket,Fare,Cabin,Embarked',
 '1,0,3,"Braund, Mr. Owen Harris",male,22,1,0,A/5 21171,7.25,,S',
 '2,1,1,"Cumings, Mrs. John Bradley (Florence Briggs Thayer)",female,38,1,0,PC 17599,71.2833,C85,C']
```

3 ways of creating a RDD

- by transforming another RDD

In [6]:
```
rdd.map(lambda number: number * 2)
```

Out[6]:
```
PythonRDD[5] at RDD at PythonRDD.scala:48
```

In [7]:
```
rdd.map(lambda number: number * 2).collect()
```

Out[7]:
```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Working with RDDs

Let's create a RDD from a list of numbers, and play with it.

In [8]:
```
rdd = sc.parallelize(range(16), 4)
rdd.cache()
```

Out[8]:
```
PythonRDD[8] at RDD at PythonRDD.scala:48
```

Remember !

- A RDD is immutable

In [9]:
```
print(rdd)              # prints only info on RDD, no evaluation
```

```
PythonRDD[8] at RDD at PythonRDD.scala:48
```

- A RDD is evaluated lazily

In [10]:
```
print(rdd.map(lambda x: x*2))       # specific methods to gather data back to driver
```

```
PythonRDD[9] at RDD at PythonRDD.scala:48
```

- Only tracks its lineage so it can reconstruct itself

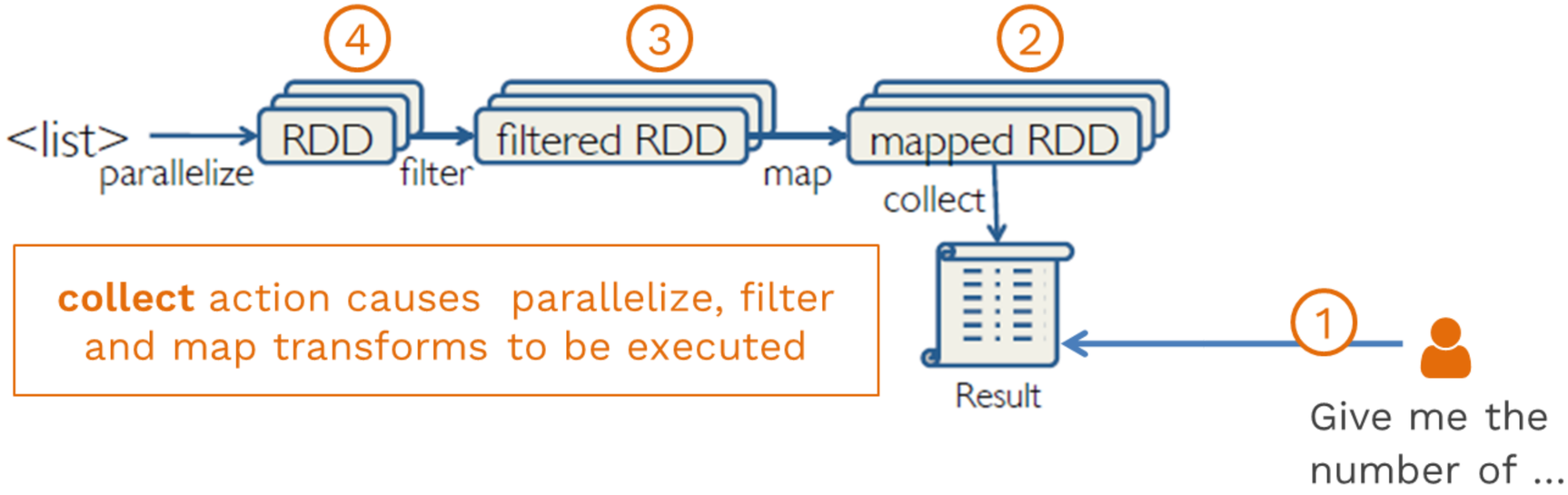
In [11]:
```
print(rdd.map(lambda num: num + 1).toDebugString())  # check RDD lineage
```

```
b'(4) PythonRDD[10] at RDD at PythonRDD.scala:48 []\n |  PythonRDD[8] at RDD at PythonRDD.scala:48 []\n |  ParallelCollectionRD
D[7] at parallelize at PythonRDD.scala:489 []'
```

Spark operations

Come in two types : transformations / actions

- Transformations are lazy *(not computed immediately)*
- Only an action on a RDD will trigger the execution of all subsequent transformations.



Transformations

Transformations shape your dataset

Filter

Return a new RDD containing only the elements that satisfy a predicate.

Ex : return only even numbers.

In [12]:
```
rdd.filter(lambda x: x % 2 == 0).collect()
```

Out[12]:
```
[0, 2, 4, 6, 8, 10, 12, 14]
```

Map

Return a new RDD by applying a function to each element of this RDD.

Ex : multiply all numbers by 2.

In [13]:
```
rdd.map(lambda x: x * 2).collect()
```

Out[13]:
```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]
```

FlatMap

Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results.

Ex : return a long matrix of rows [1, 2, 3] of dimension the number of elements in the `rdd` variable, then flatten it.

In [14]:
```
rdd.flatMap(lambda num: [1, 2, 3]).take(6)
```

Out[14]:
```
[1, 2, 3, 1, 2, 3]
```

Distinct

Return a new RDD containing the distinct elements in this RDD.

In [15]:
```
rdd.map(lambda num: 0 if num % 2 == 0 else 1).distinct().collect()
```

Out[15]:
```
[0, 1]
```

Actions

Actions execute the task and associated transformations

Collect / take

Return a list that contains all of the elements in this RDD.

Note this method should only be used if the resulting array is expected to be small, as all the data is loaded into the driver's memory

```
In [16]: rdd.take(5)
```

```
Out[16]: [0, 1, 2, 3, 4]
```

Count

Return the number of elements in this RDD.

In [17]:
```
rdd.count()
```

Out[17]: 16

Reduce

Reduces the elements of this RDD using the specified commutative and associative binary operator. Currently reduces partitions locally.

Ex : sum all the numbers in the RDD.

In [18]:
```
rdd.reduce(lambda x,y: x + y)
```

Out[18]: 120

Key-value transformations

- Key/value RDDs are commonly used to perform aggregations, and often we will do some initial ETL (extract, transform, and load) to get our data into a key/value format.
- Key/value RDDs expose new operations (e.g., counting up reviews for each product, grouping together data with the same key, and grouping together two different RDDs).

ReduceByKey

Merge the values for each key using an associative and commutative reduce function.

Ex : Add all numbers associated to each key.

In [19]:
```
rdd = sc.parallelize([('a', 1), ('b', 0), ('b', 2), ('a', 5)], 4)
rdd.reduceByKey(lambda x,y: x + y).collect()
```

Out[19]:
```
[('b', 2), ('a', 6)]
```

Join

Return an RDD containing all pairs of elements with matching keys in self and other.

Ex : Add all numbers associated to vowels and consonants.

In [20]:
```
countLetter = sc.parallelize([('a', 1), ('b', 6), ('c', 2), ('a', 5)], 4)
defLetter = sc.parallelize([('a', 'vowel'), ('b', 'consonant'), ('c', 'consonant'), ('d', 'consonant')], 4)
countLetter.join(defLetter).map(lambda x: (x[1][1], x[1][0])).reduceByKey(lambda x,y: x + y).collect()
```

Out[20]:
```
[('consonant', 8), ('vowel', 6)]
```

Wordcount !

In [21]:
```
rdd = sc.textFile('data/lorem.txt')
rdd.flatMap(lambda row: [(r, 1) for r in row.split(' ')]).reduceByKey(lambda x,y: x + y).take(6)
```

Out[21]:
```
[('Lorem', 1),
 ('ipsum', 3),
 ('consectetur', 2),
 ('elit.', 2),
 ('est', 4),
 ('mattis', 5)]
```

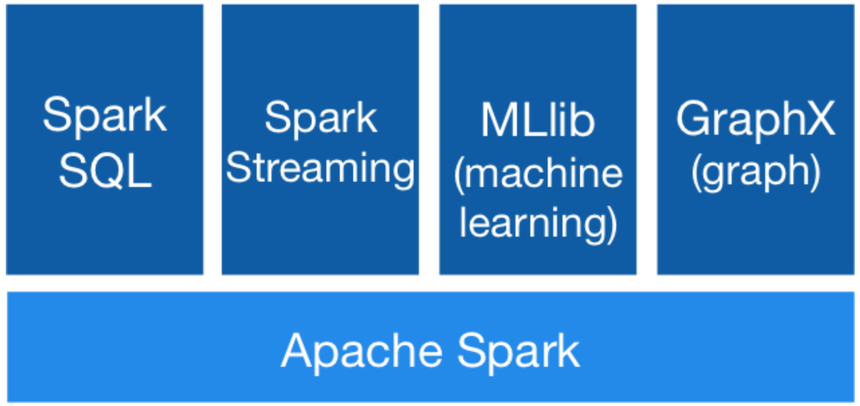
RDD conclusion

Resilient Distributed Datasets (RDDs) are a distributed collection of immutable JVM objects that allow you to perform calculations very quickly, and they are the backbone of Apache Spark

In [22]:
```
sc.stop()
```

Combine SQL, streaming, and complex analytics.

Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application.



SparkSQL

This chapter introduces Spark SQL, Spark's interface for working with structured and semistructured data.

SparkSession

The entry point to programming Spark with the Dataset and DataFrame API.

In [23]:
```
from pyspark import SparkConf
from pyspark.sql import SparkSession

conf = SparkConf().setAppName('spark-app').setMaster('local[*]')
spark = SparkSession.builder.config(conf=conf).getOrCreate()
spark
```

Out[23]:

SparkSession - in-memory

SparkContext

Spark UI

Version
`v2.2.1`

Master
`local[*]`

AppName
`spark-app`

Dataframes

Under the hood, a Dataframe is an RDD composed of Row objects with additional schema information of the types in each col- umn. Row objects are just wrappers around arrays of basic types.

In [24]:
```
titanic = spark.read.option('header', 'true').option('inferSchema', 'true').csv('data/titanic.csv')
titanic.createOrReplaceTempView('titanic')
titanic.show(8)
```

```
+-----------+--------+------+--------------------+------+----+-----+-----+----------------+-------+-----+--------+
|PassengerId|Survived|Pclass|                Name|   Sex| Age|SibSp|Parch|          Ticket|   Fare|Cabin|Embarked|
+-----------+--------+------+--------------------+------+----+-----+-----+----------------+-------+-----+--------+
|          1|       0|     3|Braund, Mr. Owen ...|  male|22.0|    1|    0|       A/5 21171|   7.25| null|       S|
|          2|       1|     1|Cumings, Mrs. Joh...|female|38.0|    1|    0|        PC 17599|71.2833|  C85|       C|
|          3|       1|     3|Heikkinen, Miss. ...|female|26.0|    0|    0|STON/O2. 3101282|  7.925| null|       S|
|          4|       1|     1|Futrelle, Mrs. Ja...|female|35.0|    1|    0|          113803|   53.1| C123|       S|
|          5|       0|     3|Allen, Mr. Willia...|  male|35.0|    0|    0|          373450|   8.05| null|       S|
|          6|       0|     3|    Moran, Mr. James|  male|null|    0|    0|          330877| 8.4583| null|       Q|
|          7|       0|     1|McCarthy, Mr. Tim...|  male|54.0|    0|    0|           17463|51.8625|  E46|       S|
|          8|       0|     3|Palsson, Master. ...|  male| 2.0|    3|    1|          349909| 21.075| null|       S|
+-----------+--------+------+--------------------+------+----+-----+-----+----------------+-------+-----+--------+
only showing top 8 rows
```

Two ways of interacting

- Domain-specific language for structured data manipulation

In [25]:
```
titanic.filter(titanic.Sex == 'male').select(['Name', 'Sex', 'Survived']).show(3)
```

```
+--------------------+----+--------+
|                Name| Sex|Survived|
+--------------------+----+--------+
|Braund, Mr. Owen ...|male|       0|
|Allen, Mr. Willia...|male|       0|
|    Moran, Mr. James|male|       0|
+--------------------+----+--------+
only showing top 3 rows
```

- `sql` function on SparkSession to run SQL queries programmatically on temporary tables

In [26]:
```
spark.sql('SELECT Name, Sex, Survived FROM titanic WHERE Sex = "male"').show(3)
```

```
+--------------------+----+--------+
|                Name| Sex|Survived|
+--------------------+----+--------+
|Braund, Mr. Owen ...|male|       0|
|Allen, Mr. Willia...|male|       0|
|    Moran, Mr. James|male|       0|
+--------------------+----+--------+
only showing top 3 rows
```

Unified data source interaction

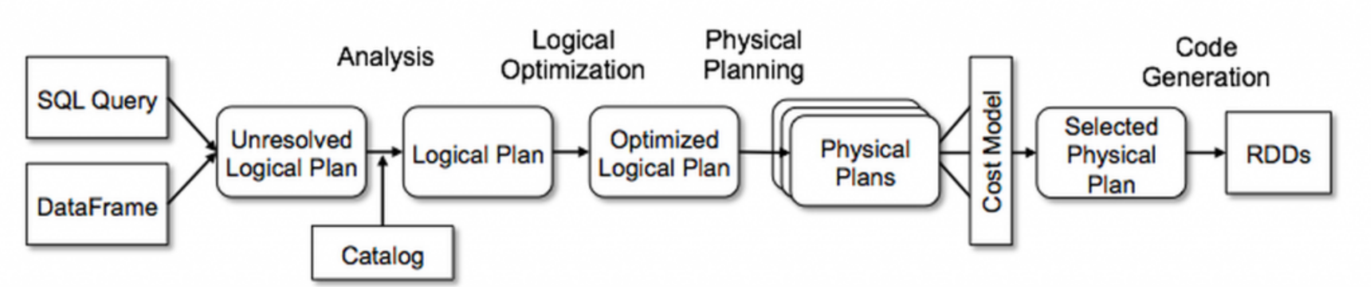
Spark provides with a unique interface for reading/saving data, which is then implemented for multiple data storage formats : *json*, *parquet*, *jdbc*, *orc*, *libsvm*, *csv*, *text*.

In [27]:
```
ransomware = spark.read.json('data/ransomware.json')
ransomware.printSchema()
```

```
root
 |-- comment: string (nullable = true)
 |-- decryptor: string (nullable = true)
 |-- encryptionAlgorithm: string (nullable = true)
 |-- extensionPattern: string (nullable = true)
 |-- extensions: string (nullable = true)
 |-- iocs: string (nullable = true)
 |-- microsoftDetectionName: string (nullable = true)
 |-- microsoftInfo: string (nullable = true)
 |-- name: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- ransomNoteFilenames: string (nullable = true)
 |-- resources: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- sandbox: string (nullable = true)
 |-- screenshots: string (nullable = true)
 |-- snort: string (nullable = true)
```

Catalyst optimization

Catalyst is an extensible query optimizer used internally by SparkSQL for planning and defining the execution of SparkSQL queries.



In [28]:
```
titanic[titanic['Sex'] == 'male'].select(['Name', 'Sex']).explain()
```

```
== Physical Plan ==
*Project [Name#15, Sex#16]
+- *Filter (isnotnull(Sex#16) && (Sex#16 = male))
   +- *FileScan csv [Name#15,Sex#16] Batched: false, Format: CSV, Location: InMemoryFileIndex[file:/C:/workspaceperso/pyspark-interactive-lecture/notebooks/data/titanic.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Sex), EqualTo(Sex,male)], ReadSchema: struct<Name:string,Sex:string>
```

Machine Learning

MLlib is Spark's machine learning (ML) library. It has an RDD-based API in maintenance mode and a Dataframe-based API.

- Dataframe API = Spark Datasources, SQL/DataFrame queries, Tungsten and Catalyst optimizations, uniform APIs across languages.
- ML Pipelines are set of high-level APIs on top of DataFrames that help users create and tune practical machine learning pipelines

Transformers

A Transformer implements a method transform(), which converts one DataFrame into another

In [29]:
```
from pyspark.ml.feature import StringIndexer

indexer = StringIndexer(inputCol="Sex", outputCol="SexIndex")
titanic_indexed = indexer.fit(titanic).transform(titanic)
titanic_indexed.show(8)
```

```
+-----------+--------+------+--------------------+------+----+-----+-----+----------------+-------+-----+--------+--------+
|PassengerId|Survived|Pclass|                Name|   Sex| Age|SibSp|Parch|          Ticket|   Fare|Cabin|Embarked|SexIndex|
+-----------+--------+------+--------------------+------+----+-----+-----+----------------+-------+-----+--------+--------+
|          1|       0|     3|Braund, Mr. Owen ...|  male|22.0|    1|    0|       A/5 21171|   7.25| null|       S|     0.0|
|          2|       1|     1|Cumings, Mrs. Joh...|female|38.0|    1|    0|        PC 17599|71.2833|  C85|       C|     1.0|
|          3|       1|     3|Heikkinen, Miss. ...|female|26.0|    0|    0|STON/O2. 3101282|  7.925| null|       S|     1.0|
|          4|       1|     1|Futrelle, Mrs. Ja...|female|35.0|    1|    0|          113803|   53.1| C123|       S|     1.0|
|          5|       0|     3|Allen, Mr. Willia...|  male|35.0|    0|    0|          373450|   8.05| null|       S|     0.0|
|          6|       0|     3|    Moran, Mr. James|  male|null|    0|    0|          330877| 8.4583| null|       Q|     0.0|
|          7|       0|     1|McCarthy, Mr. Tim...|  male|54.0|    0|    0|           17463|51.8625|  E46|       S|     0.0|
|          8|       0|     3|Palsson, Master. ...|  male| 2.0|    3|    1|          349909| 21.075| null|       S|     0.0|
+-----------+--------+------+--------------------+------+----+-----+-----+----------------+-------+-----+--------+--------+
only showing top 8 rows
```

Estimators

An Estimator implements a method fit(), which accepts a DataFrame and produces a Model, which is a Transformer.

In [30]:
```
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(inputCols=["SexIndex", "Fare"], outputCol="features")
titanic_train = assembler.transform(titanic_indexed)

rf = RandomForestClassifier(labelCol="Survived", featuresCol="features", numTrees=10)
model = rf.fit(titanic_train)
model.transform(titanic_train).select(["Survived", "prediction", "probability"]).show(8)
```

```
+--------+----------+--------------------+
|Survived|prediction|         probability|
+--------+----------+--------------------+
|       0|       0.0|[0.94189369125263...|
|       1|       1.0|[0.21883383407637...|
|       1|       1.0|[0.46619780756453...|
|       1|       1.0|[0.02089552238805...|
|       0|       0.0|[0.87832770415448...|
|       0|       0.0|[0.84656818503583...|
|       0|       0.0|[0.66412205718598...|
|       0|       0.0|[0.86223713039307...|
+--------+----------+--------------------+
only showing top 8 rows
```

Pipelines

MLlib standardizes APIs for machine learning algorithms to make it easier to combine multiple algorithms into a single pipeline, or workflow.

In [31]:
```
from pyspark.ml import Pipeline

pipeline = Pipeline(stages=[indexer, assembler, rf])
model = pipeline.fit(titanic)
model.transform(titanic).select(["Survived", "prediction", "probability"]).show(8)
```

```
+--------+----------+--------------------+
|Survived|prediction|         probability|
+--------+----------+--------------------+
|       0|       0.0|[0.94189369125263...|
|       1|       1.0|[0.21883383407637...|
|       1|       1.0|[0.46619780756453...|
|       1|       1.0|[0.02089552238805...|
|       0|       0.0|[0.87832770415448...|
|       0|       0.0|[0.84656818503583...|
|       0|       0.0|[0.66412205718598...|
|       0|       0.0|[0.86223713039307...|
+--------+----------+--------------------+
only showing top 8 rows
```

Spark Streaming

Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.

In [32]:
```
# Prepare a netcat client before launching launchSparkStreaming
import nclib

#nc = nclib.Netcat(listen=('localhost', 9999), verbose=True)
```

In [33]:
```
#for i in range(1000):
    #nc.send_line(b'hello world')
```

In [34]:
```
#nc.close()
```

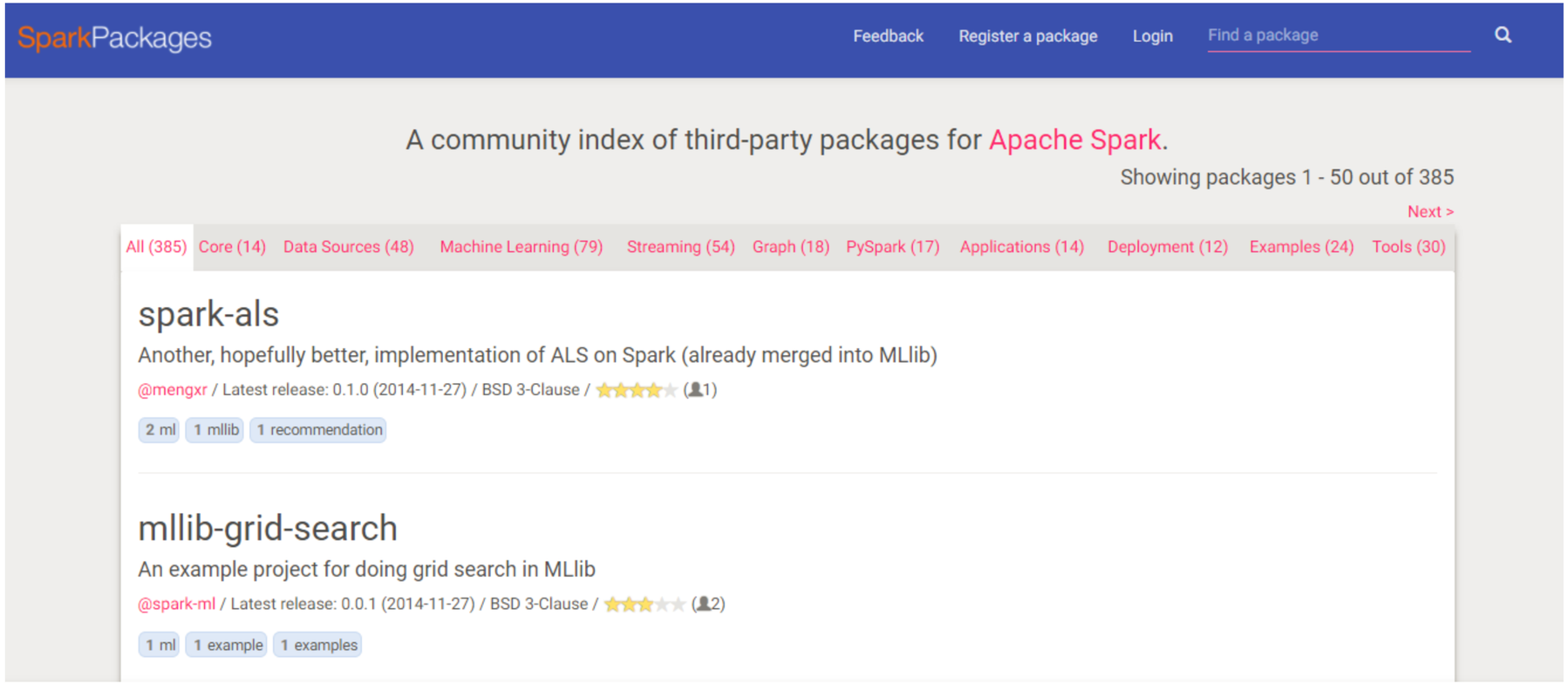
GraphX

To support graph computation, GraphX extends the Spark RDD by introducing a new Graph abstraction: a directed multigraph with properties attached to each vertex and edge.

NB : No active development of Python bindings on GraphX...take a look on GraphFrames for graph computation on Dataframes, which is the unofficial GraphX Dataframe-based API.

Going further

Spark packages



Unified engine

Spark's main contribution is to enable previously disparate cluster workloads to be composed. In the following example, we build a logistic model on the titanic dataset, save it on disk and push it to spark streaming for realtime inference.

MLlib

Core

Spark Streaming

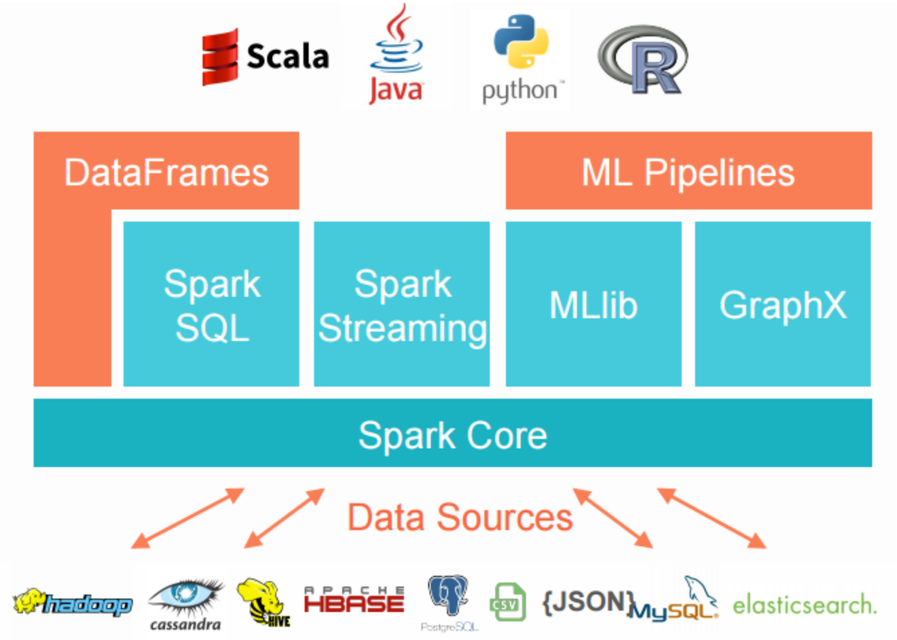
```
val predictions = model.predictOn(trainingData)

predictions.foreachRDD { rdd =>
  val modelString = model.latestModel().clusterCenters
    .map(c => c.toString.slice(1, c.toString.length-1)).mkString("\n")
  val predictString = rdd.map(p => p.toString).collect().mkString("\n")
  val dateString = Calendar.getInstance().getTime.toString.replace(" ", "-")
  Utils.printToFile(outputDir, dateString + "-model", modelString)
  Utils.printToFile(outputDir, dateString + "-predictions", predictString)
}

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

In [35]:
```
spark.stop()
```

Conclusion


Scala
Java
python
DataFrames
ML Pipelines
Spark SQL
Spark Streaming
MLlib
GraphX
Spark Core
Data Sources
hadoop
cassandra
HIVE
APACHE HBASE
PostgreSQL
CSV
{JSON}
MySQL
elasticsearch.