Spark

Enterprise Architectures for Big Data

What is Spark?

- **Big data processing engine** for
  fast calculations on **distributed in-memory datasets**
- Apache Open Source
- Supported languages: Python (PySpark), Scala, R, Java, SQL

Source: Hortonworks

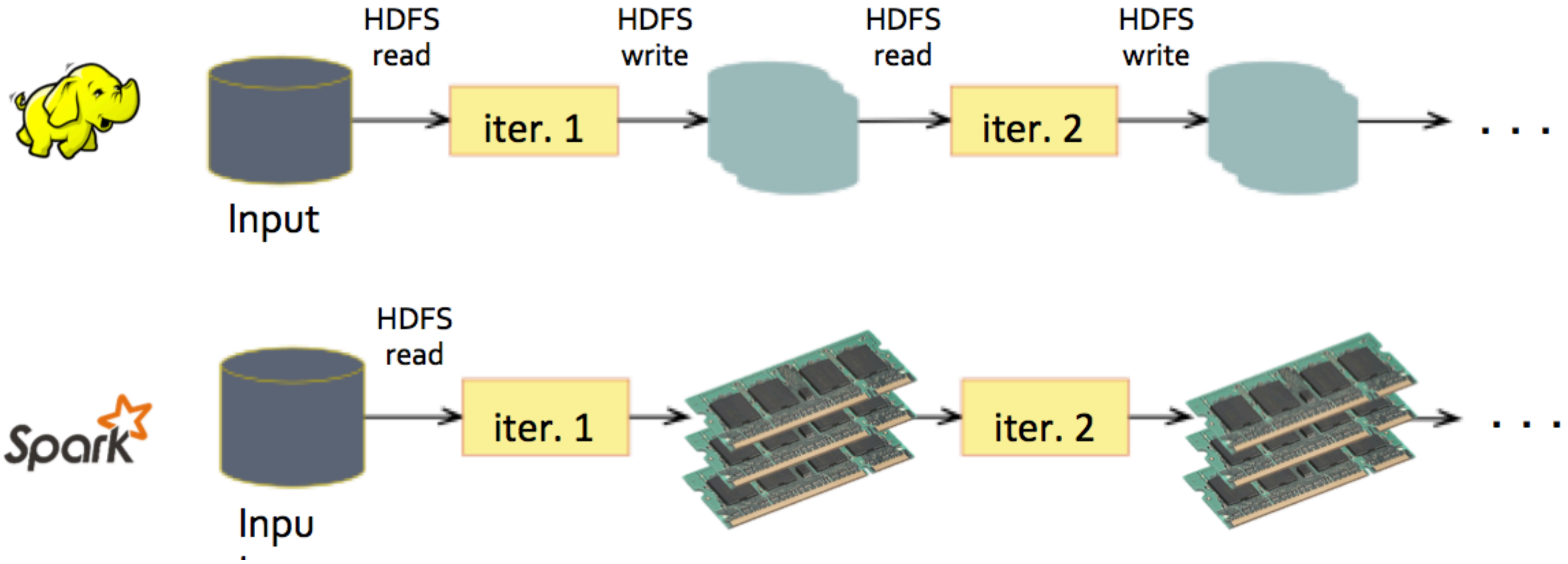
Disadvantages of Hadoop

- MapReduce is difficult to program
- However, there exist high level interfaces
  - Hive
  - Pig

- Performance is slow
- Mainly for Batch Processing, no interactive (online) processing

Spark vs. MapReduce

- MapReduce – involves lots of disk I/O
- Disk I/O is very slow



Why is Spark faster?

- Caching data to memory can avoid extra reads from disk
- Scheduling of tasks from 15-20s to 15-20ms
- Resources are dedicated the entire life of the application
- Can link multiple maps and reduces together without having to write intermediate data to HDFS
- Every reduce doesn’t require a map

Source: Hortonworks

Motivation: Spark vs. MapReduce

- Higher level API
- Distributed in-memory data storage
- Up to 100x performance improvement

```
df = spark.read.json("logs.json")
df.where("age > 21")
  .select("name.first").show()
```

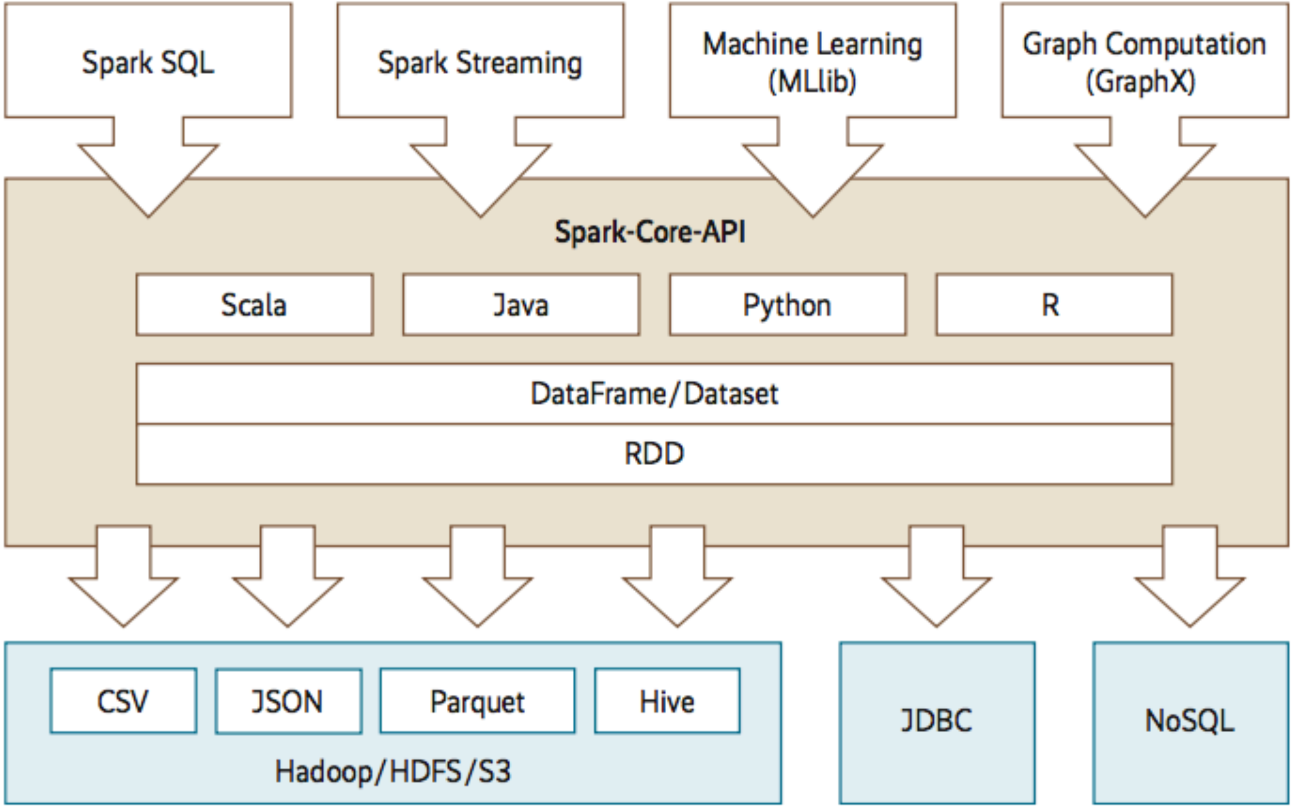
Spark's Python DataFrame API
Read JSON files with automatic schema inference

Spark Timeline

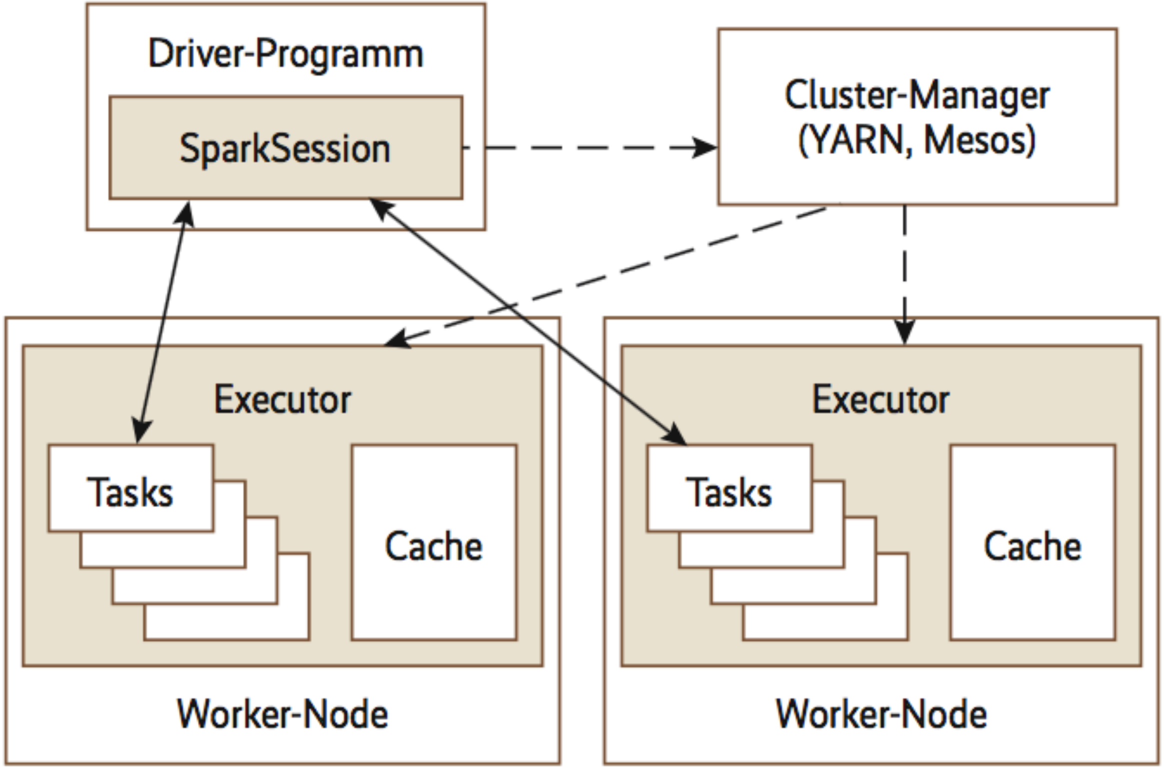
Time	Development State
2009	Start of the development at the AMPLab of the Berkeley University
June 2013	Apache Incubation
February 2014	Apache Top Level Project
May 2014	Spark 1.0: Spark SQL, MLLib, GraphX, Streaming
March 2015	Spark 1.3: DataFrame API
January 2016	Spark 1.6: Dataset API
July 2016	Spark 2.0: revised DataFrames and Dataset API, Performance, improved Spark SQL
December 2016	Spark 2.1: Improved Streaming and Machine Learning
February 2020	Spark 2.4.5 Latest Version

Source: Hortonworks

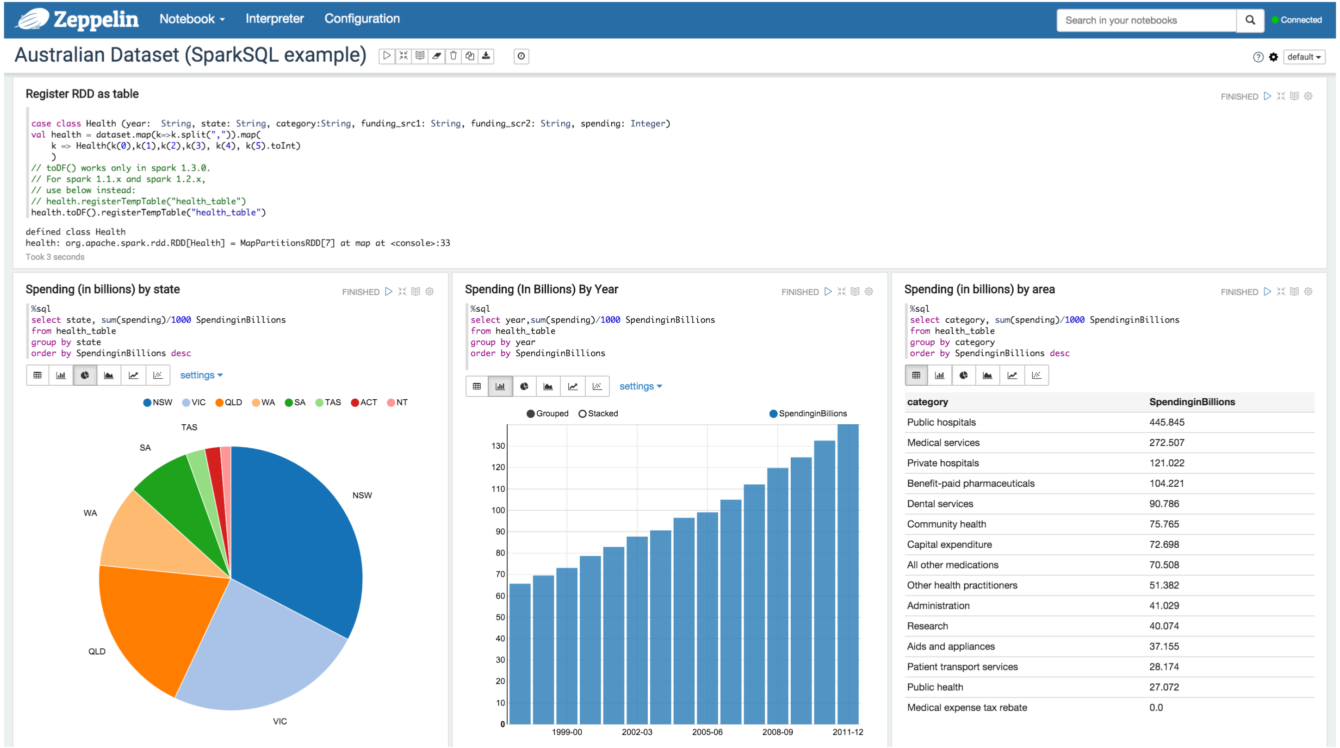
Spark Components



Spark Cluster



Zeppelin Notebook



category	SpendinginBillions
Public hospitals	445.845
Medical services	272.507
Private hospitals	121.022
Benefit-paid pharmaceuticals	104.221
Dental services	90.786
Community health	75.765
Capital expenditure	72.698
All other medications	70.508
Other health practitioners	51.382
Administration	41.029
Research	40.074
Aids and appliances	37.155
Patient transport services	28.174
Public health	27.072
Medical expense tax rebate	0.0

Spark Data Interfaces

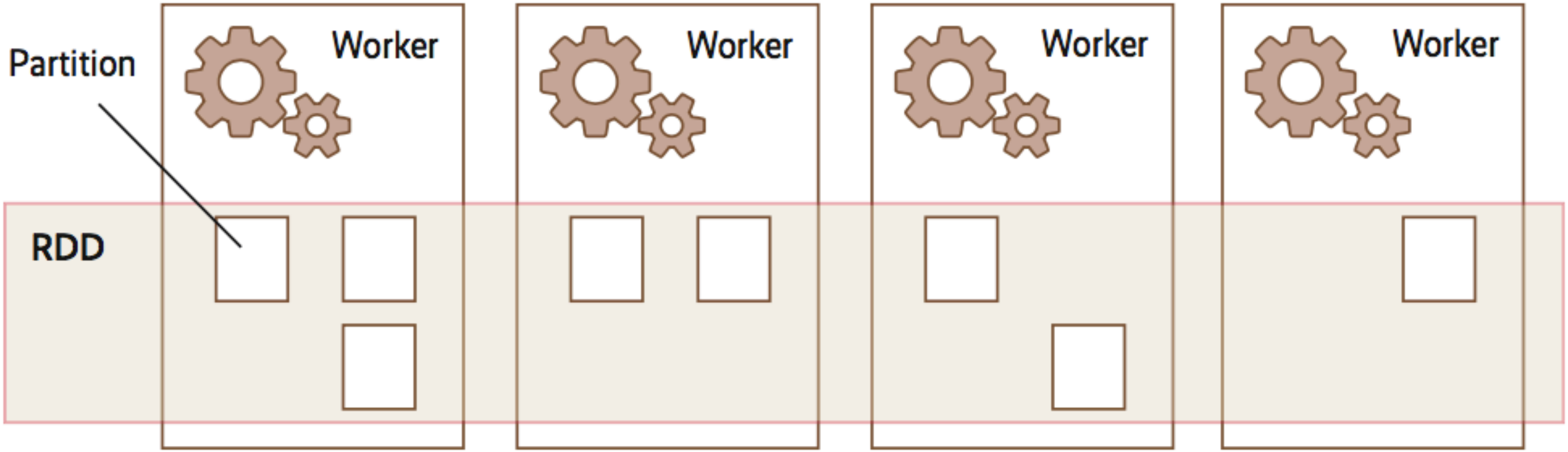
- RDD (Resilient Distributed Dataset)
  - Sequence of data objects that consist of one or more types that are located across a variety of machines in a cluster.
- DataFrame
  - For structured data
  - PySpark DataFrame is an immutable distributed collection of data with named columns
  - Provides flexible interface, similar to DataFrames in Python (Pandas)
  - DataFrames in PySpark support both
    - SQL queries (SELECT * from table) or
    - expression methods (df.select())

PySpark RDD

Resilient Distributed Dataset: RDD

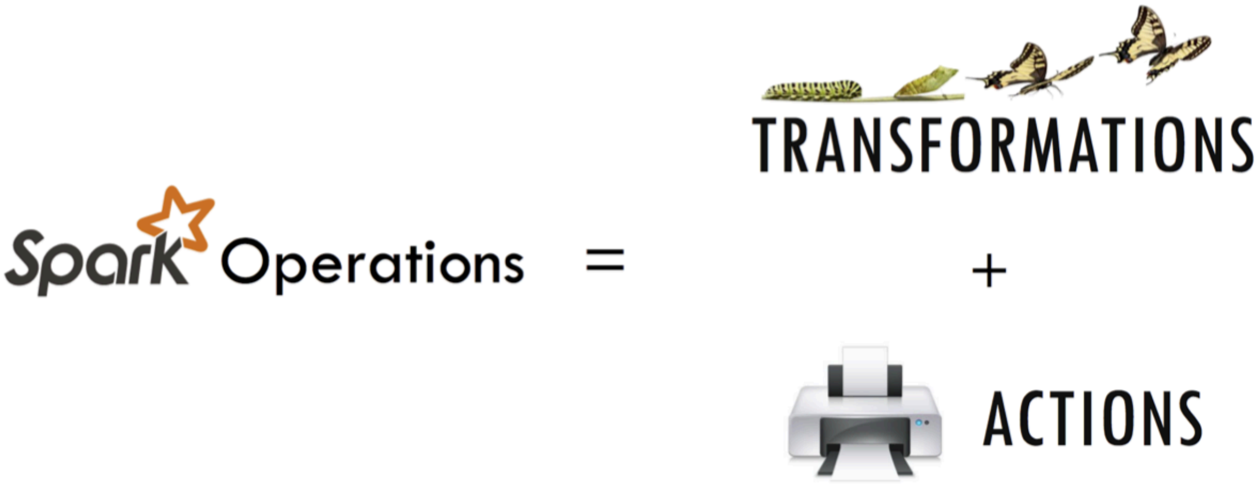
- RDD is the basis for what Spark enables
- Resilient: the models can be recreated on the fly from known state
- Distributed: the dataset is partitioned across multiple nodes for increased scalability and parallelism
- Immutable: every transformation creates a new RDD

RDDs: distributed collections on Worker-Nodes in a Cluster



Overview of PySpark RDD operations

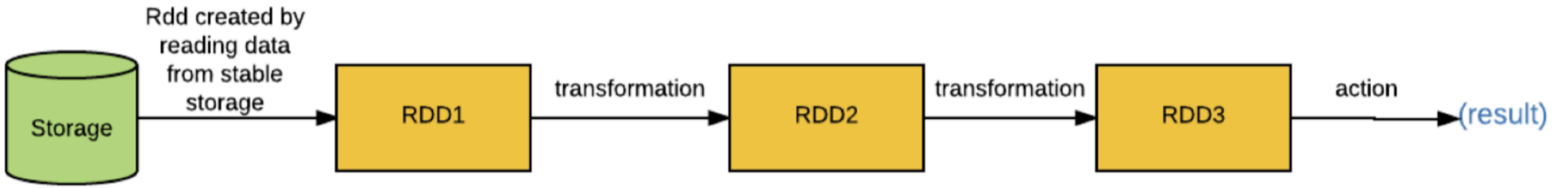
- Transformations create new RDDs
- Actions perform computation on the the RDDs



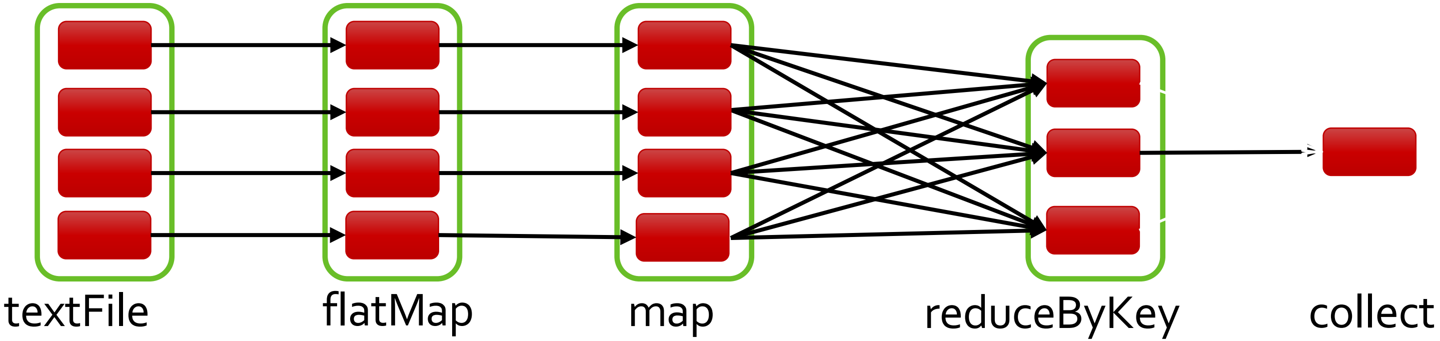
Source: Hortonworks

RDD Transformations

- Transformations follow Lazy evaluation
- Basic RDD Transformations
  - map()
  - filter()
  - flatMap()
  - union()



RDD with MapReduce



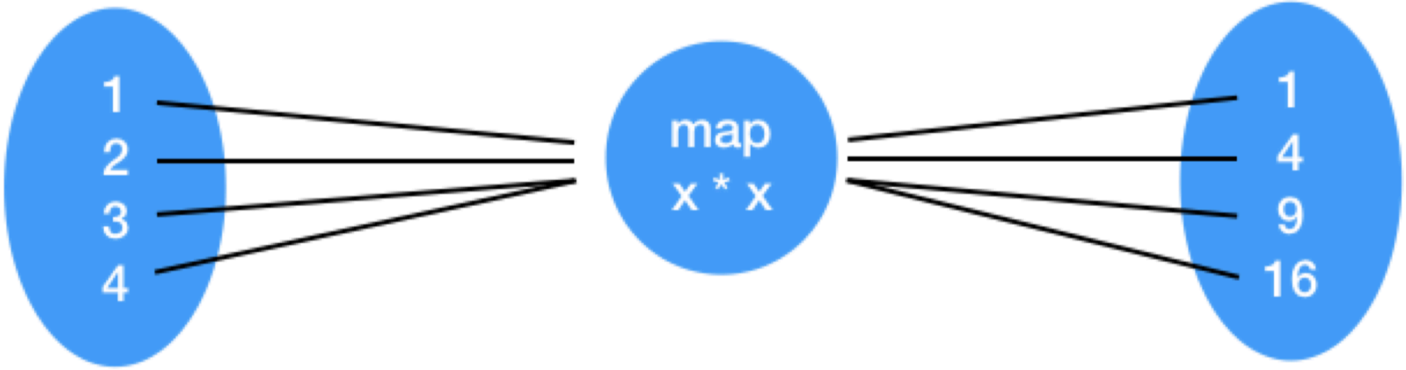
Spark Context sc

- Spark Context (sc) is the object to create RDDs
- `rdd = sc.textFile(…)`

Source: Hortonworks

map() Transformation

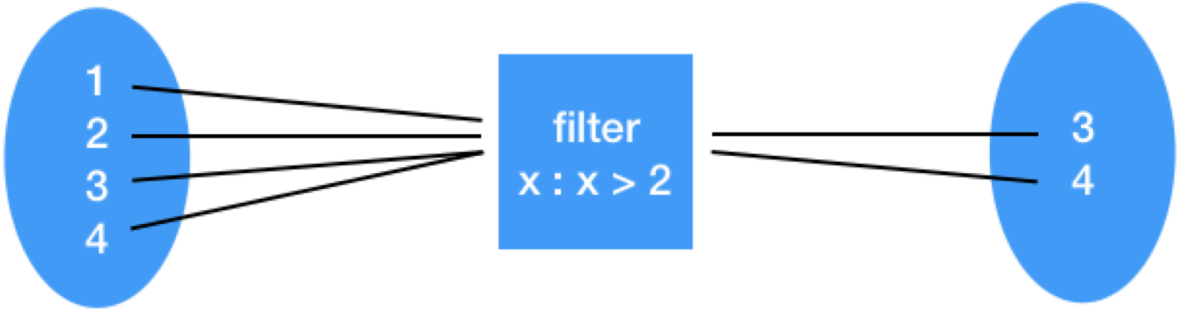
- map() applies a function to all elements in the RDD




```
RDD = sc.parallelize([1,2,3,4])
RDD_map = RDD.map(lambda x: x * x)
```

filter() Transformation

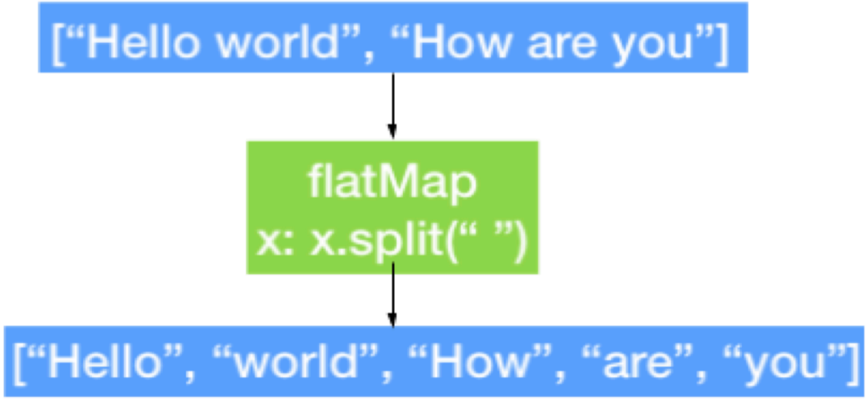
- Filter returns a new RDD with
  only the elements that pass the condition




```
RDD = sc.parallelize([1,2,3,4])
RDD_filter = RDD.filter(lambda x: x > 2)
```

flatMap() Transformation

- flatMap() returns multiple values for each element in the original RDD




```
RDD = sc.parallelize(["hello world", "how are you"])
RDD_flatmap = RDD.flatMap(lambda x: x.split(" "))
```

reduce() Transformation

- reduce() is used for aggregating the elements of a regular RDD
- The function should be commutative and associative

```
x = [1,3,4,6]
RDD = sc.parallelize(x)
RDD.reduce(lambda x, y : x + y)

14
```

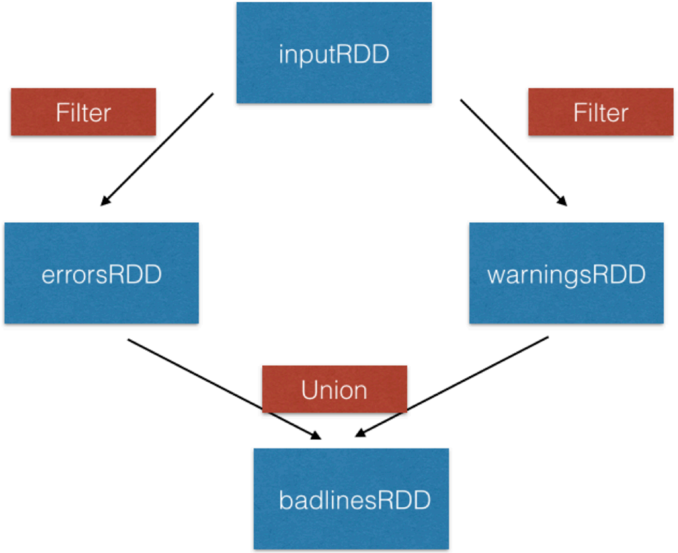
reduceByKey() Transformation

- reduceByKey() combines values with the same key
- It runs parallel operations for each key in the dataset
- It is a transformation and not action

```
regularRDD = sc.parallelize([("Messi", 23), ("Ronaldo", 34), ("Neymar", 22), ("M
pairRDD_reducebykey = regularRDD.reduceByKey(lambda x,y : x + y)
pairRDD_reducebykey.collect()

[('Neymar', 22), ('Ronaldo', 34), ('Messi', 47)]
```

union() Transformation




```
inputRDD = sc.textFile("logs.txt")
errorRDD = inputRDD.filter(lambda x: "error" in x.split())
warningsRDD = inputRDD.filter(lambda x: "warnings" in x.split())
combinedRDD = errorRDD.union(warningsRDD)
```

RDD Actions

- Action returns a value after running a computation on the RDD
- Basic RDD Actions
  - collect()
  - take(N)
  - first()
  - count()

- collect() returns all the elements of the dataset as an array
- take(N) returns an array with the first N elements of the dataset

Source: Hortonworks

PySpark DataFrame

Spark DataFrame

- Since Spark 2.0 the preferred data API
- High level interface similar to
  Python Pandas DataFrame

- Spark Session is the object to create DataFrames

```
df = spark.read.csv(..)
```

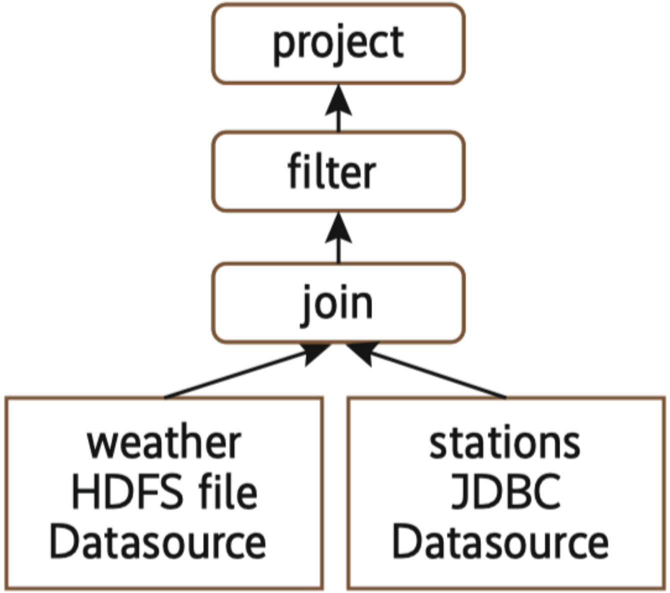
Source: Hortonworks

Transformations & Actions
Lazy Execution

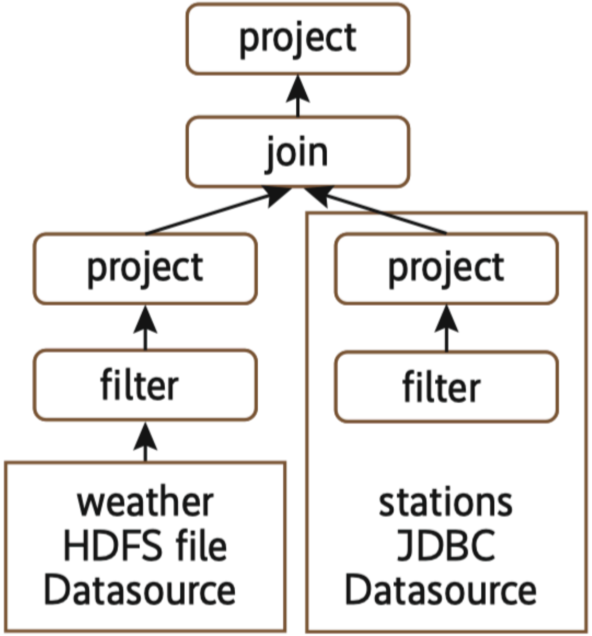
Transformations *(lazy)*	Actions
`select`	`show`
`distinct`	`count`
`groupBy`	`collect`
`sum`	`save`
`orderBy`	
`filter`	
`limit`	

Optimizing of Execution Plans

No Optimization



Optimization



Create Dataset from Files using read()

- Datasets can be created easily from certain structured file types, including CSV, JSON

```
df = spark.read.csv("diamonds.csv", header=True, inferSchema=True)
```

sqlContext.show() and display()

- When displaying DataFrame contents,
  use show() to display the contents on-screen

```
df.show()
```

- Create a table with visualizations

```
display(df)
```

Save DataFrames as Files Using write()

- DataFrames can be saved to HDFS or to the local file system as files of many commonly used file formats, including CSV, JSON

```
df.write.csv.save(”filename.csv")

df.write.json.save(”filename.json")
```

Source: Hortonworks

Register Dataset as a Temporary Tables

- Use createOrReplaceTempView() to make the Dataset available to SQL within the current context

```
df.createOrReplaceTempView("someTableName")
```

sqlContext.sql()

- The spark API enables a user to run native SQL commands using the sql() function and the name of the spark session

```
df = spark.sql("SELECT * FROM permcd")
```

Source: Hortonworks

Manipulate SQL Tables using SQL Commands

- Tables can be manipulated by using standard SQL commands via Spark SQL or within the DataFrames API using spark.sql()

```
%sql
select code from permab
```

code
AA
BB

show(), printSchema()

- show() displays the DataFrame or SQL result
- printSchema() displays the schema for a DataFrame

describe() and distinct()

- describe() returns count, mean, standard deviation, minimum, and maximum values for named columns in a DataFrame

- distinct() returns a new DataFrame of only the unique rows from the original DataFrame

```
%pyspark
dfP.describe("value").show()
dfP.distinct().show()

+-------+-----------------+
|summary|            value|
+-------+-----------------+
|  count|                2|
|   mean|         115000.0|
| stddev|49497.47468305832|
|    min|            80000|
|    max|           150000|
+-------+-----------------+

+----+------+
|code| value|
+----+------+
|  BB| 80000|
|  AA|150000|
+----+------+
```

withColumnRenamed() and select()

- withColumnRenamed() returns a new Dataframe with a renamed column
- select() returns a new DataFrame with only the specified columns and data

```
%pyspark
dataframeRename = dataframeAdd.withColumnRenamed("multiplied", "annual")
dataframeRename.show()
dataframeSelect = dataframeRename.select("code","annual")
dataframeSelect.show()
```

```
+----+------+------+
|code| value|annual|
+----+------+------+
|  AA|150000|300000|
|  BB| 80000|160000|
+----+------+------+

+----+------+
|code|annual|
+----+------+
|  AA|300000|
|  BB|160000|
+----+------+
```

filter() and limit()

- filter() / where() returns a DataFrame with only rows that have column values that meet a defined criteria

- limit() returns a DataFrame with a defined number of rows

```
%pyspark
dfP.filter(dfP['value'] < 100000).show()
dfP.limit(1).show()
```

```
+----+-----+
|code|value|
+----+-----+
|  BB|80000|
+----+-----+
+----+------+
|code| value|
+----+------+
|  AA|150000|
+----+------+
```

drop() and groupBy()

- drop() returns a DataFrame without the named column(s)
- groupBy() groups rows by matching column values, which can then have some other function performed on them and returned as the result
- Two examples here, performing a count() on the code column, and then performing a sum() on the values column

```
%pyspark
dfP.drop("value").show()
dfP.groupBy("code").count().show()
dfP.groupBy("value").sum().show()

+----+
|code|
+----+
|  AA|
|  BB|
+----+

+----+-----+
|code|count|
+----+-----+
|  AA|    1|
|  BB|    1|
+----+-----+

+------+----------+
| value|sum(value)|
+------+----------+
|150000|    150000|
| 80000|     80000|
```

count(), take(), and head()

- count() returns the number of rows in the DataFrame as a result
- take() returns a number of rows in the DataFrame and returns them as Row objects
- head() returns the first row in the DataFrame as a Row object

```
%pyspark
print dfP.count()
print dfP.take(1)
print dfP.head()
```

```
2
[Row(code=u'AA', value=150000)]
Row(code=u'AA', value=150000)
```

Spark Machine Learning

Spark Machine Learning Library (MLlib) Overview

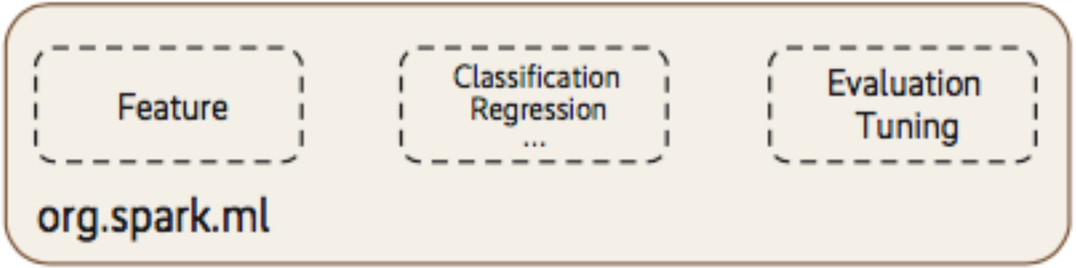
- A Spark implementation of common machine learning algorithms and utilities
- Includes:
  - Classification
  - Regression
  - Clustering
  - Collaborative filtering
  - Dimensionality reduction

- MLlib allows data scientists the ability to easily scale machine learning algorithms on a Spark cluster.

Source: Hortonworks

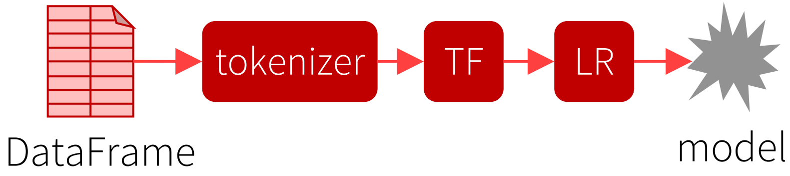
Spark Machine Learning

Data Selection → **Data Transformation** → **Modeling** → **Model Evaluation and Tuning** → **Model Usage**



Spark Machine Learning Pipeline

- Based on DataFrames
- Pipeline API similar to scikit-learn
- Many functions like grid search, cross-validation, etc.




```
tokenizer = Tokenizer()
tf = HashingTF(numFeatures=1000)
lr = LogisticRegression()

pipe = Pipeline([tokenizer, tf, lr])
model = pipe.fit(df)
```

Summary

- Spark is a distributed in-memory processing engine
- Spark is not a database (no transactions or mutable datasets)
- Data processing, ETL, machine learning, stream processing, SQL querying
- Supported languages: Scala, Java, Python, and R
- Maintains dedicated resources and fast task scheduler
- Spark SQL has two different APIs:
  - Resilient Distributed Dataset (RDD)
  - DataFrame similar to Python Pandas with SQL querying
- Spark Machine Learning Library (MLlib) allows easy scaling of machine learning algorithms
- Apache Zeppelin offers Notebooks style interfaces.
- You can also connect Jupyter Notebooks to a Spark cluster