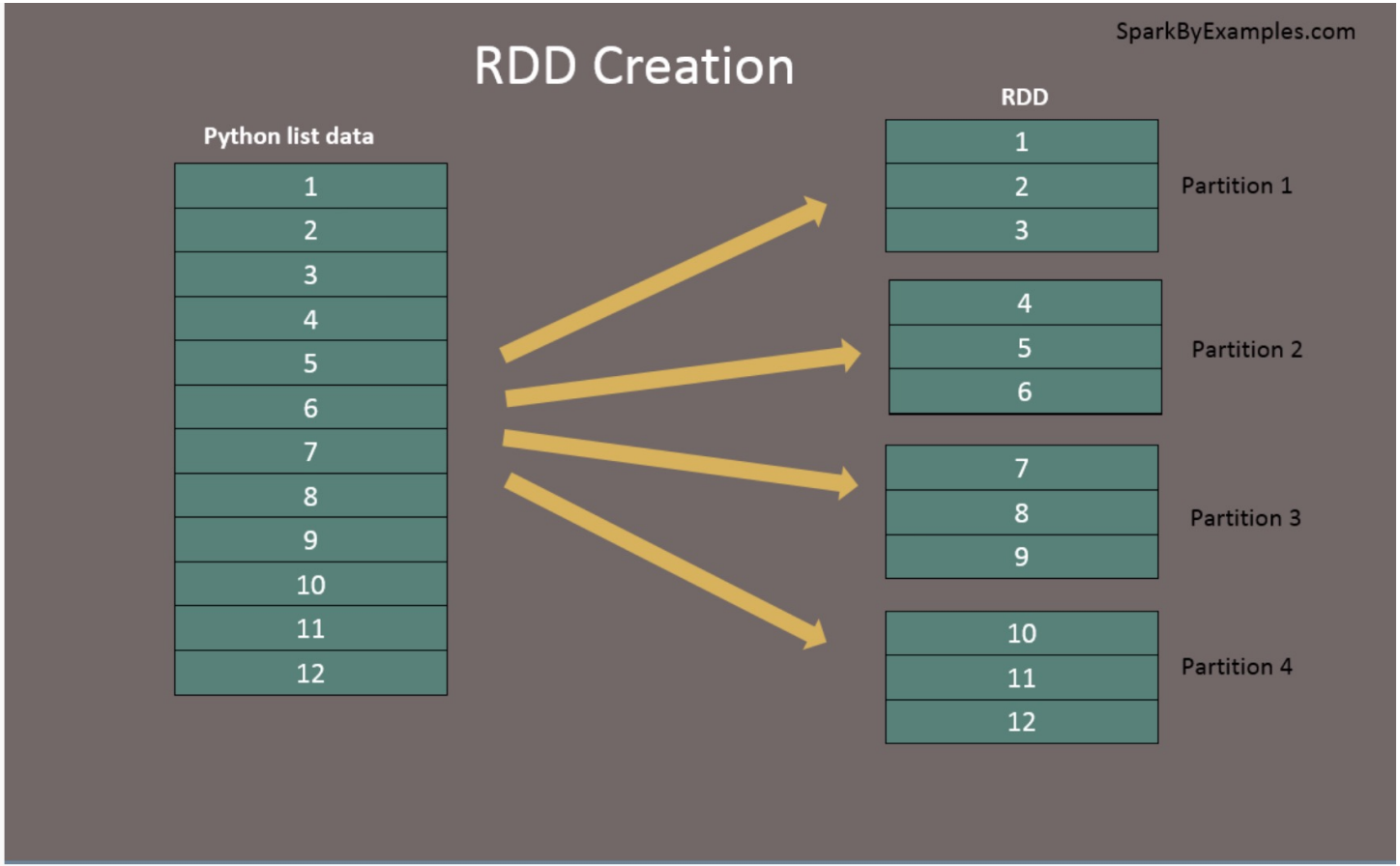
Урок 5. RDD

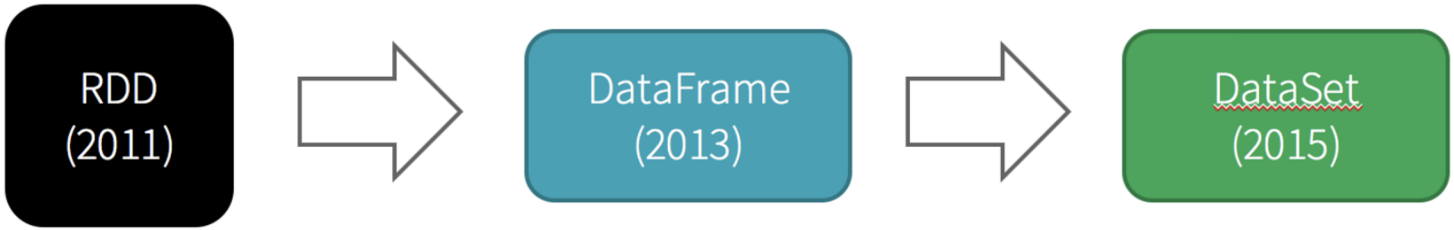
RDD


SparkByExamples.com
RDD Creation
Python list data
1
2
3
4
5
6
7
8
9
10
11
12
RDD
1
2
3
Partition 1
4
5
6
Partition 2
7
8
9
Partition 3
10
11
12
Partition 4

Creating RDD from DataFrame and vice-versa

```
# Converts RDD to DataFrame
dfFromRDD1 = rdd.toDF()
# Converts RDD to DataFrame with column names
dfFromRDD2 = rdd.toDF("col1","col2")
# using createDataFrame() - Convert DataFrame to RDD
df = spark.createDataFrame(rdd).toDF("col1","col2")
# Convert DataFrame to RDD
rdd = df.rdd
```

History of Spark APIs



Distribute collection of JVM objects

Functional Operators (map, filter, etc.)

Distribute collection of Row objects

Expression-based operations and UDFs

Logical plans and optimizer

Fast/efficient internal representations

Internally rows, externally JVM objects

Almost the "Best of both worlds": **type safe + fast**

But slower than DF
Not as good for interactive analysis, especially Python

databricks

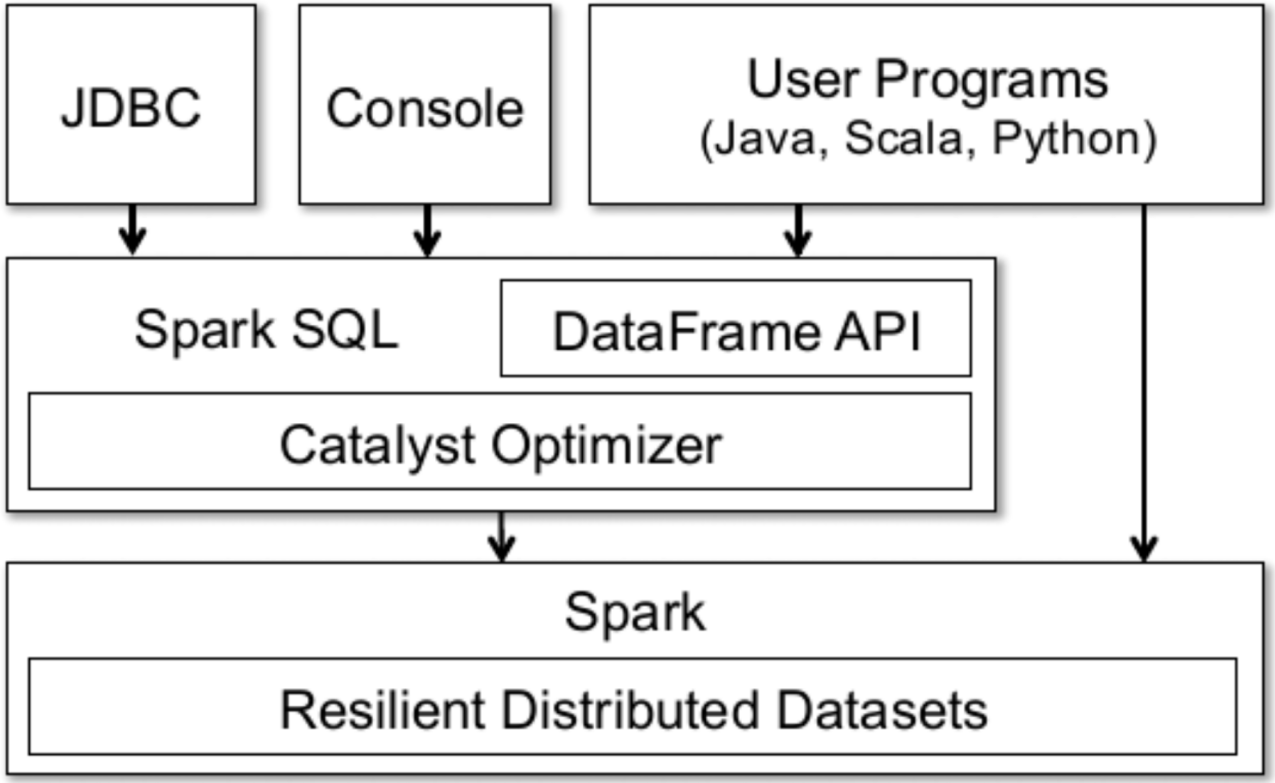
https://stackoverflow.com/questions/31508083/difference-between-dataframe-dataset-and-rdd-in-spark

Typed and Un-typed APIs

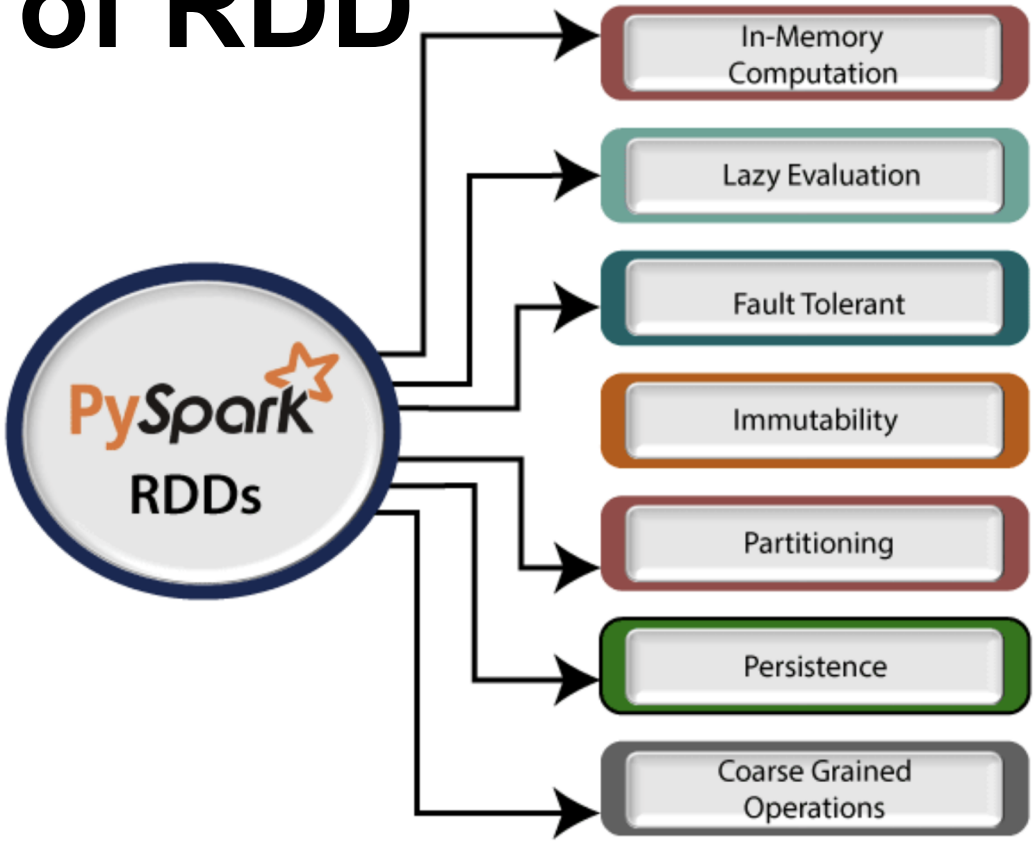
Language	Main Abstraction
Scala	Dataset[T] & DataFrame (alias for Dataset[Row])
Java	Dataset[T]
Python*	DataFrame
R*	DataFrame

Note: *Since Python and R have no compile-time type-safety, we only have untyped APIs, namely DataFrames.*

https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html


JDBC
Console
User Programs
(Java, Scala, Python)
Spark SQL
DataFrame API
Catalyst Optimizer
Spark
Resilient Distributed Datasets

Features of RDD



https://www.javatpoint.com/pyspark-rdd

Pair RDDs

Distributed key-value pairs are represented as Pair RDDs in Spark.

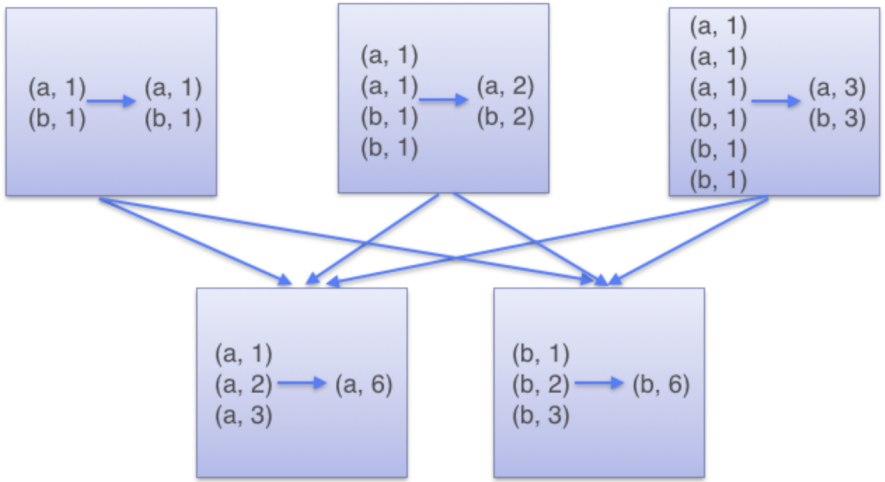
They are useful because they allow us to act on each key in parallel or regroup data across the network.

An RDD parameterized by a pair are treated as Pair RDD

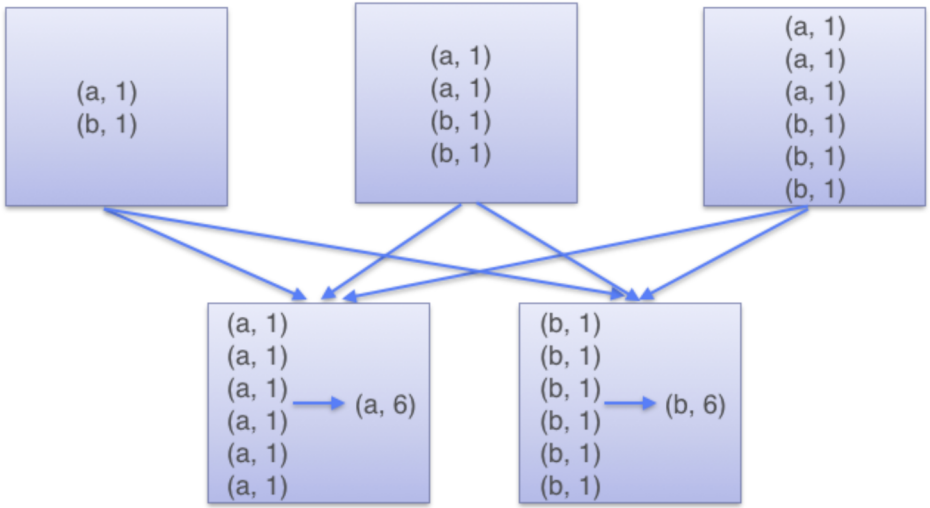
```
RDD[(k, v)] // <------- treated specially by Spark
```

https://medium.com/@goyalsaurabh66/pair-rdds-transformations-and-actions-1a286fc999f8

ReduceByKey



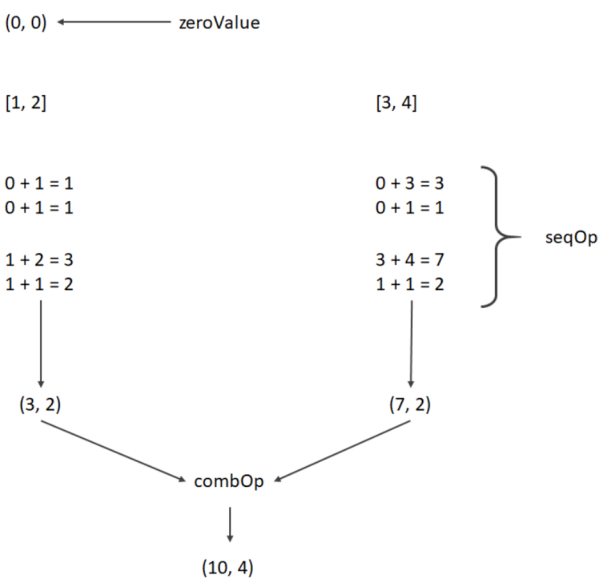
GroupByKey



https://databricks.gitbooks.io/databricks-spark-knowledge-base/content/best_practices/prefer_reducebykey_over_groupbykey.html

With the help of the schema we can summarize the operation step by step :

aggregate(zeroValue, seqOp, combOp)



schema of aggregate() function

```
new_rdd=sc.parallelize([1, 2, 3, 4])

rdd.aggregate((0,0),(lambda x,y:(x[0]+y,x[1]+1)),(lambda
x,y:(x[0]+y[0],x[1]+y[1])))

(10, 4)
```

https://sparkbyexamples.com/apache-spark-rdd/spark-rdd-aggregate-example/

PySpark Shared Variables

- Broadcast variables (read-only shared variable)
- Accumulator variables (updatable shared variables)

Broadcast read-only Variables

- Переменные доступны в каждой таске всех stages
- Команда sc.broadcast(variable) не отправляет переменную на экзекьюторы, это происходит при первом их использовании

```
broadcastVar = sc.broadcast(Array(0, 1, 2, 3))
broadcastVar.value
```

https://sparkbyexamples.com/pyspark/pyspark-broadcast-variables/

Spark Accumulators

- write-only
- initialize once variables where only tasks that are running on workers are allowed to update
- updates from the workers get propagated automatically to the driver program.
- only the driver program is allowed to access the Accumulator variable using the **value** property
- named (preferred, observable from spark ui) and unnamed (ex. below)

```
accum=sc.accumulator(0)
rdd=spark.sparkContext.parallelize([1,2,3,4,5])
rdd.foreach(lambda x:accum.add(x))
print(accum.value) #Accessed by driver
```

https://sparkbyexamples.com/pyspark/pyspark-accumulator-with-example/

pyspark.AccumulatorParam

Helper object that defines how to accumulate values of a given type.

```
>>> from pyspark.accumulators import AccumulatorParam
>>> class VectorAccumulatorParam(AccumulatorParam):
...     def zero(self, value):
...         return [0.0] * len(value)
...     def addInPlace(self, val1, val2):
...         for i in range(len(val1)):
...              val1[i] += val2[i]
...         return val1
>>> va = sc.accumulator([1.0, 2.0, 3.0], VectorAccumulatorParam())
>>> va.value
[1.0, 2.0, 3.0]
>>> def g(x):
...     global va
...     va += [x] * 3
>>> rdd = sc.parallelize([1,2,3])
>>> rdd.foreach(g)
>>> va.value
[7.0, 8.0, 9.0]
```

Links

http://datacamp-community-prod.s3.amazonaws.com/02213cb4-b391-4516-adcd-57243ced8eed

https://sparkbyexamples.com/pyspark-rdd/