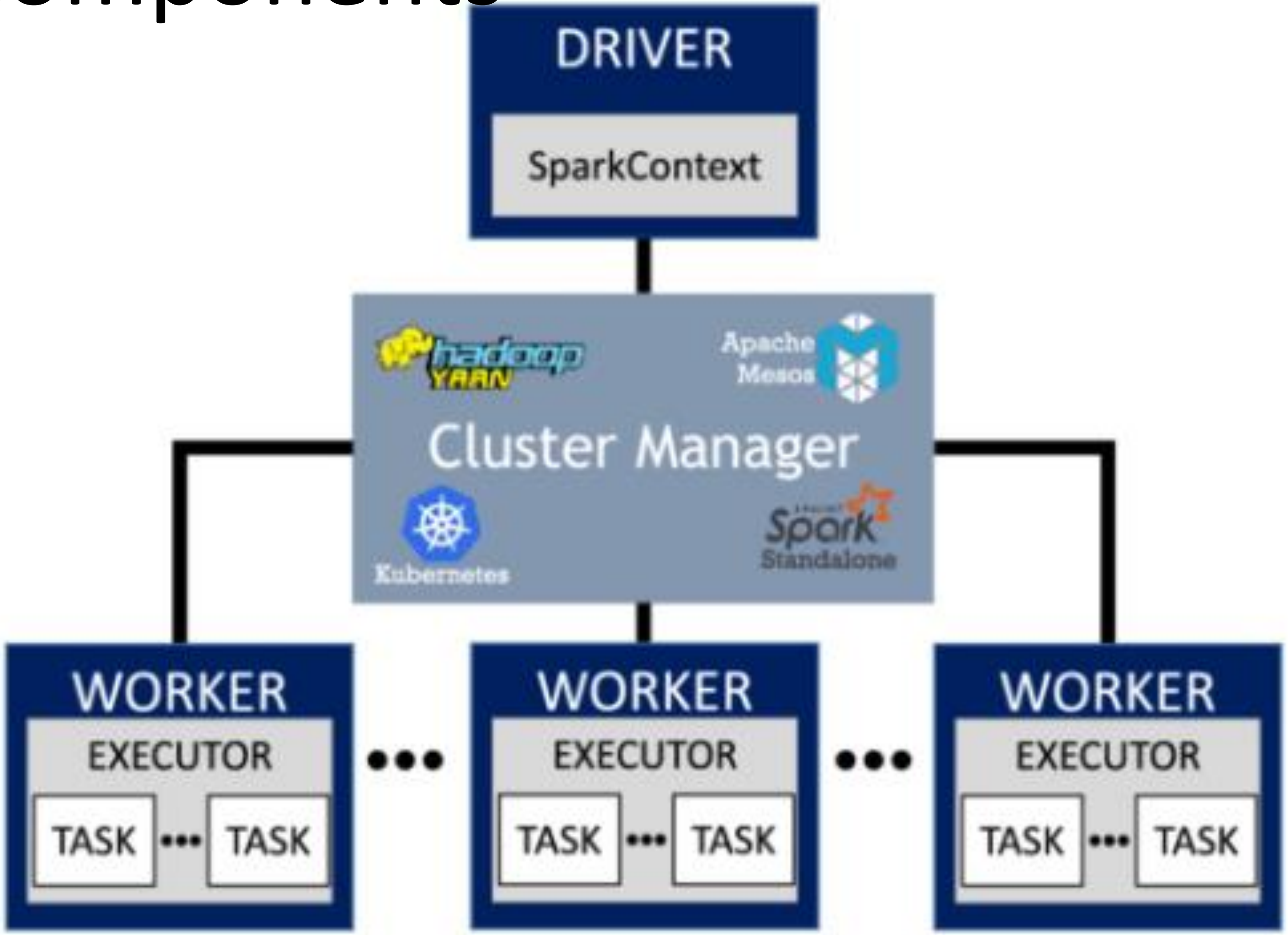
Systems, paradigms and algorithms for Big Data
TD 1

Why Spark ?

- Ease creation of complicated data analysis task
- Support iterative algorithms, like gradient descent
- Ease data exploration

- Make it easier to explore data interactively
    - Developer friendly, no need to create tons of classes and jobs…
    - Repl mode
    - Scala/Python API similar to functional programming

Main Components



How to run it

- Repl : Spark-shell, notebooks ➔ exploratory mode
- Spark-submit : runs a script ➔ scheduled job
- Api for Java, Scala, Python.
- Deploy Mode : client vs cluster

Dependencies

- Depends on Hadoop Libraries (HDFS and Yarn)
- Requires a Java Runtime Environment (need Java in your system path)

RDD

Resilient Distributed Dataset

- The base building block of your application
- Lazy : doesn’t compute anything before it really needs to do so.
- Immutable : a transformation doesn’t change the data set, it returns a new RDD.
- Fault Tolerant : partition can be recomputed in case of failure

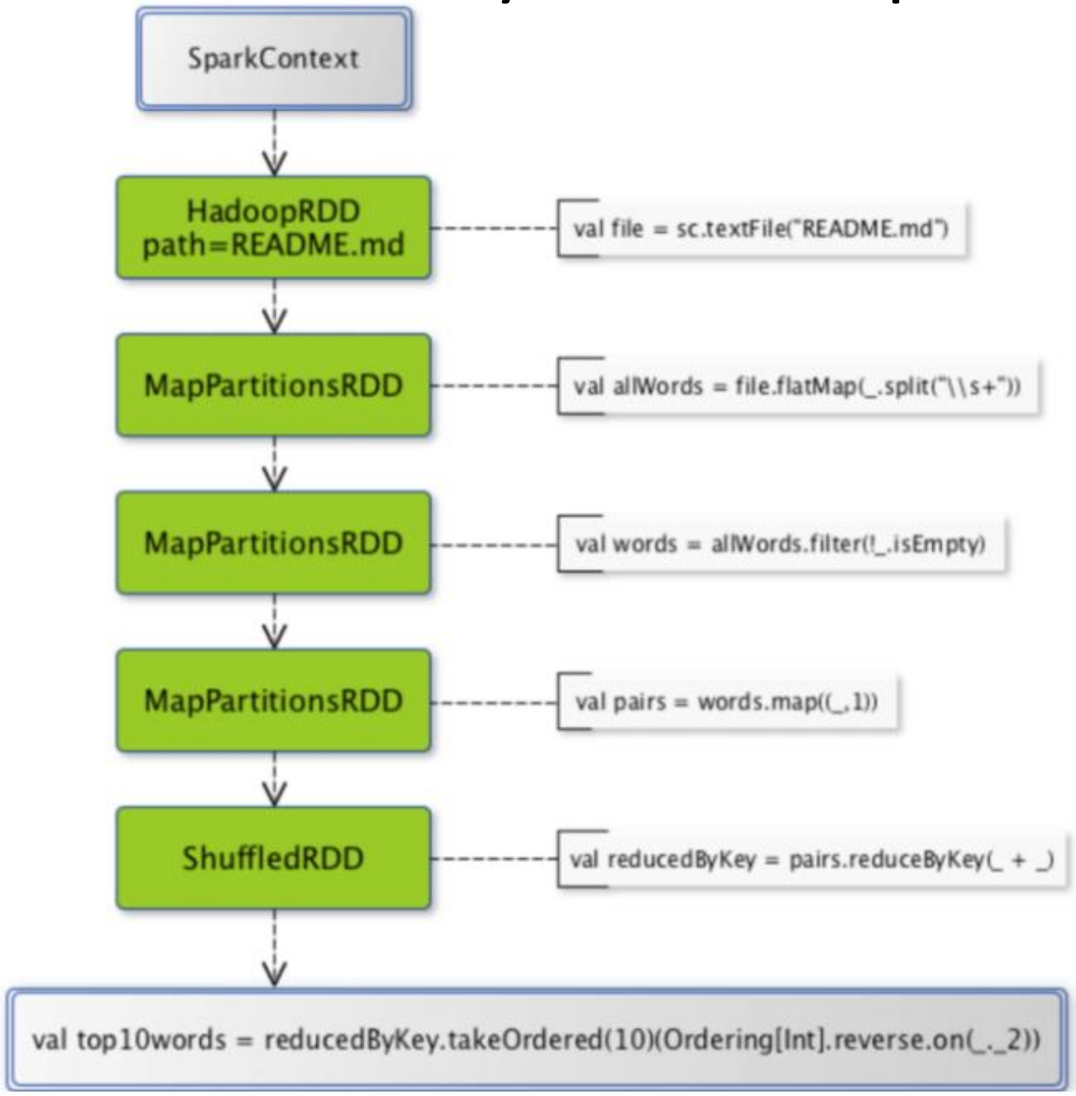
Graph

- Conceptually, an RDD is a node of a graph of functions.
- API is pretty much similar with **Functional Programming. Forget the for-loop.**

Inside the RDD:

- Knows the partitions he is working-on, how data is partitionned
- Knows how to iterate over each partition to yield records
- Know RDD’s it depends-on

RDD as a Directed Acyclic Graph



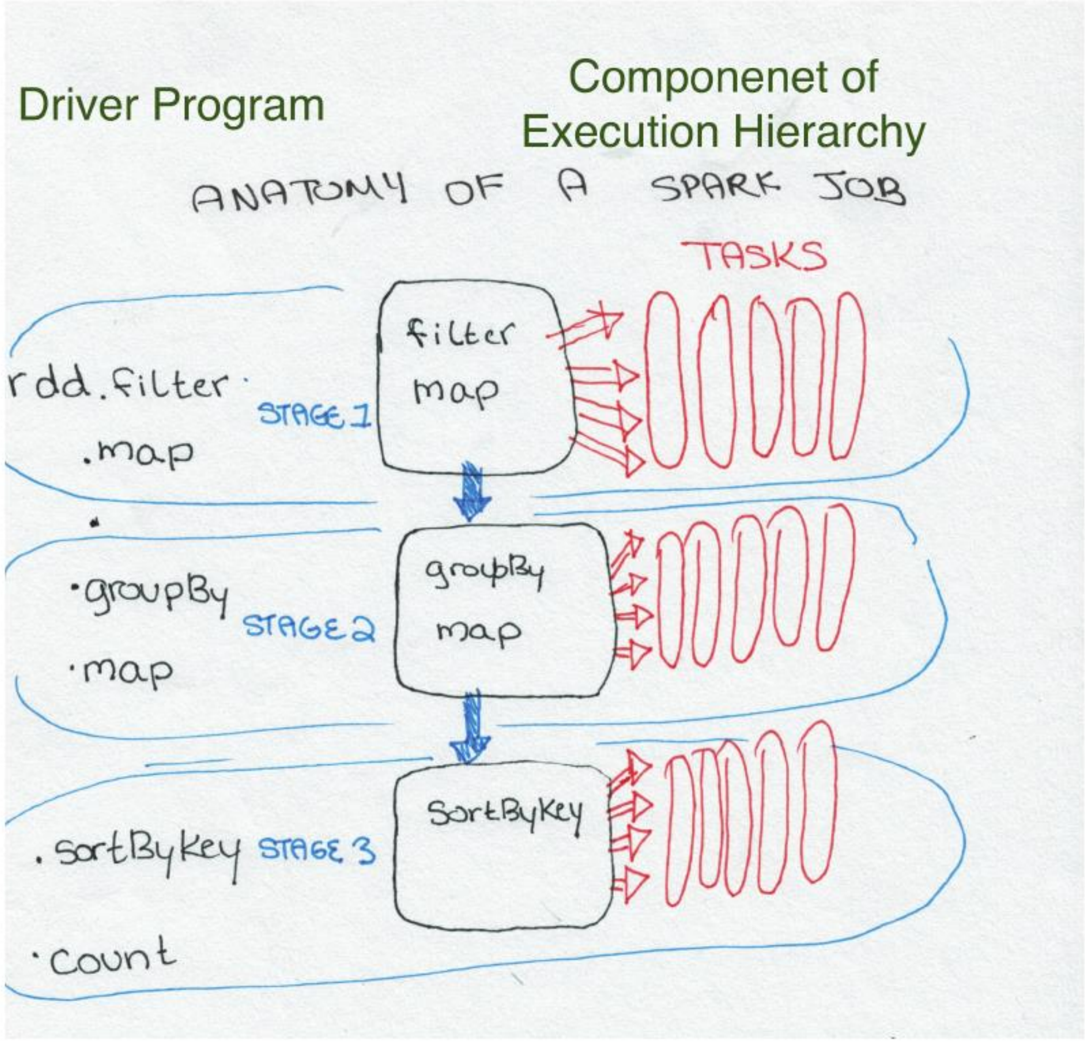
```
val file = sc.textFile("README.md")
val allWords = file.flatMap(_.split("\\s+"))
val words = allWords.filter(!_.isEmpty)
val pairs = words.map((_,1))
val reducedByKey = pairs.reduceByKey(_ + _)
val top10words = reducedByKey.takeOrdered(10)(Ordering[Int].reverse.on(_._2))
```

New RDD is created after every transformation.(DAG graph)

MAP

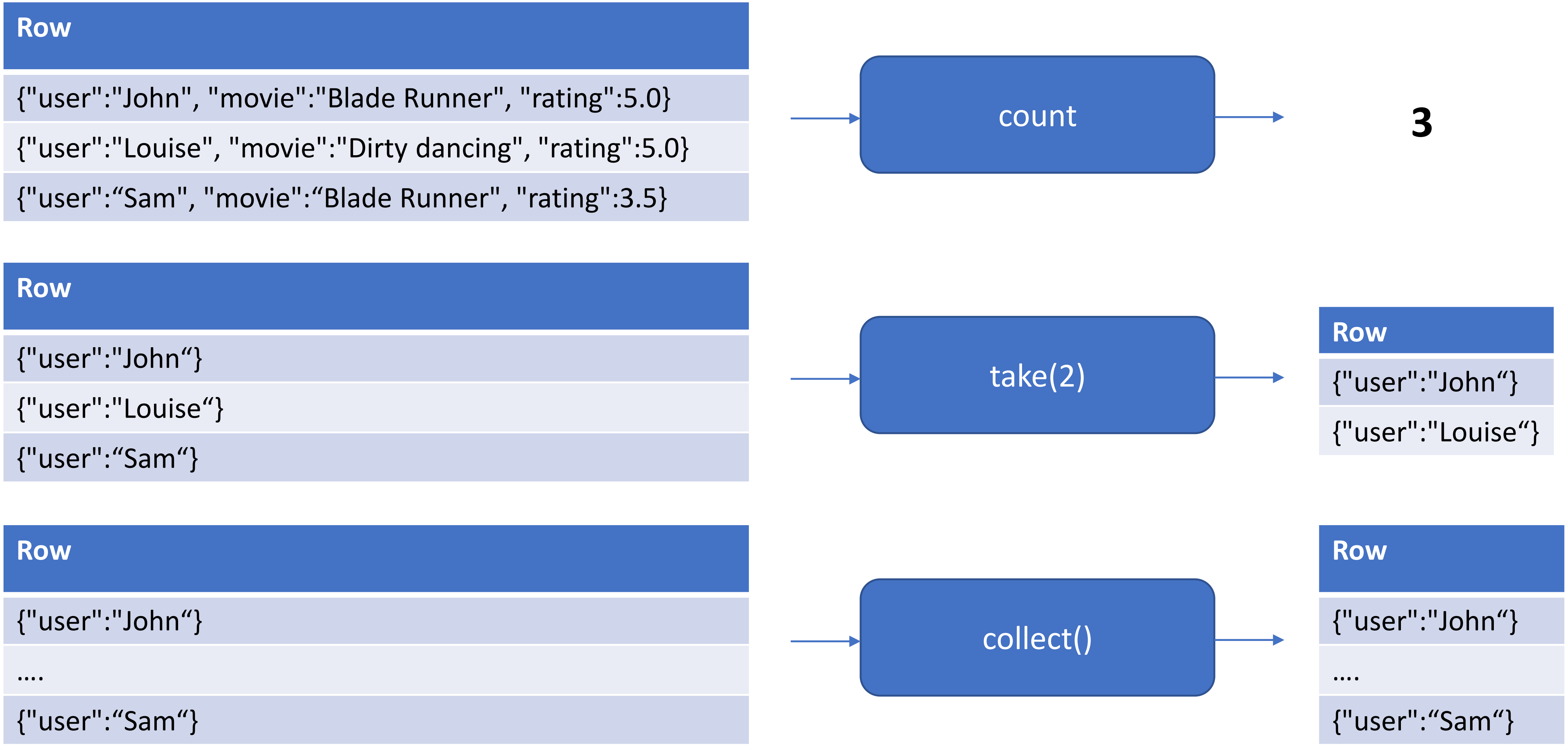
REDUCE

Jobs, Stages, Tasks



Source : high performance Spark

RDD API - Actions



Row
{"user":"John", "movie":"Blade Runner", "rating":5.0}
{"user":"Louise", "movie":"Dirty dancing", "rating":5.0}
{"user":“Sam", "movie":“Blade Runner", "rating":3.5}

count

3

Row
{"user":"John“}
{"user":"Louise“}
{"user":“Sam“}

take(2)

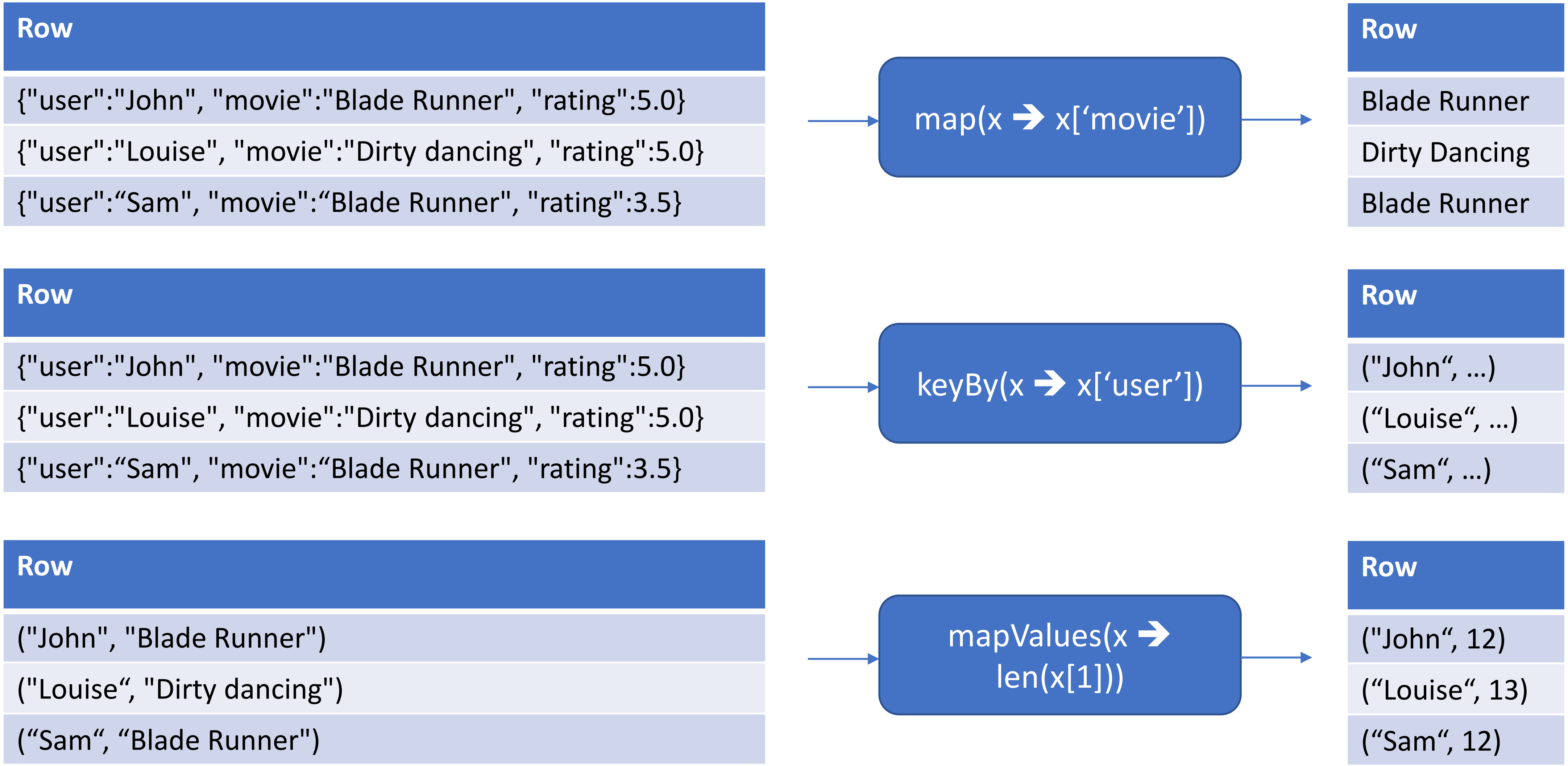
Row
{"user":"John“}
{"user":"Louise“}

Row
{"user":"John“}
….
{"user":“Sam“}

collect()

Row
{"user":"John“}
….
{"user":“Sam“}

RDD API



Row
{"user":"John", "movie":"Blade Runner", "rating":5.0}
{"user":"Louise", "movie":"Dirty dancing", "rating":5.0}
{"user":“Sam", "movie":“Blade Runner", "rating":3.5}

map(x ➔ x[‘movie’])

Row
Blade Runner
Dirty Dancing
Blade Runner

Row
{"user":"John", "movie":"Blade Runner", "rating":5.0}
{"user":"Louise", "movie":"Dirty dancing", "rating":5.0}
{"user":“Sam", "movie":“Blade Runner", "rating":3.5}

keyBy(x ➔ x[‘user’])

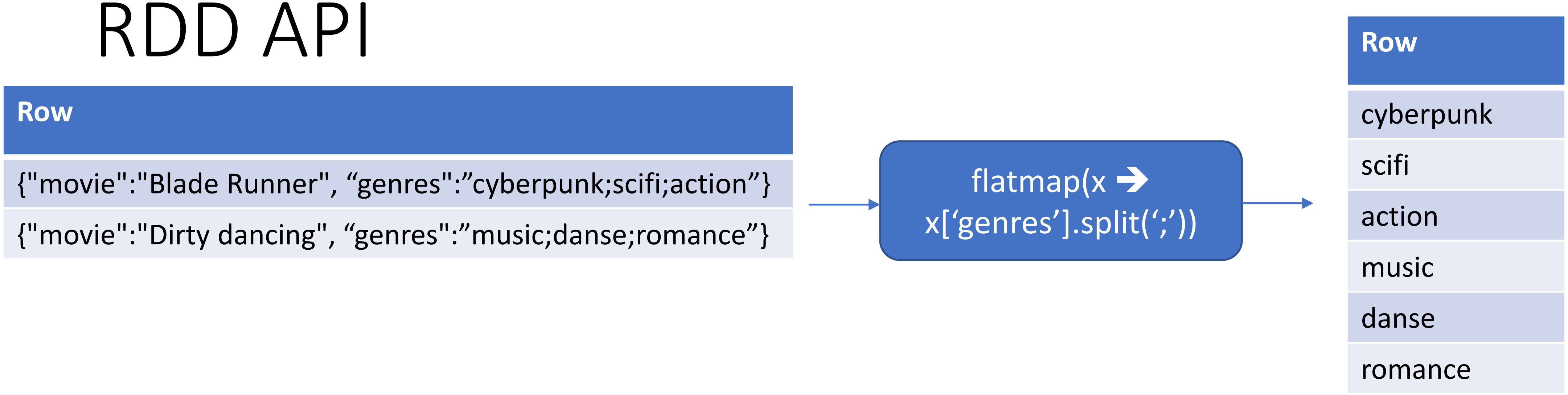
Row
("John“, …)
(“Louise“, …)
(“Sam“, …)

Row
("John", "Blade Runner")
("Louise“, "Dirty dancing")
(“Sam“, “Blade Runner")

mapValues(x ➔ len(x[1]))

Row
("John“, 12)
(“Louise“, 13)
(“Sam“, 12)

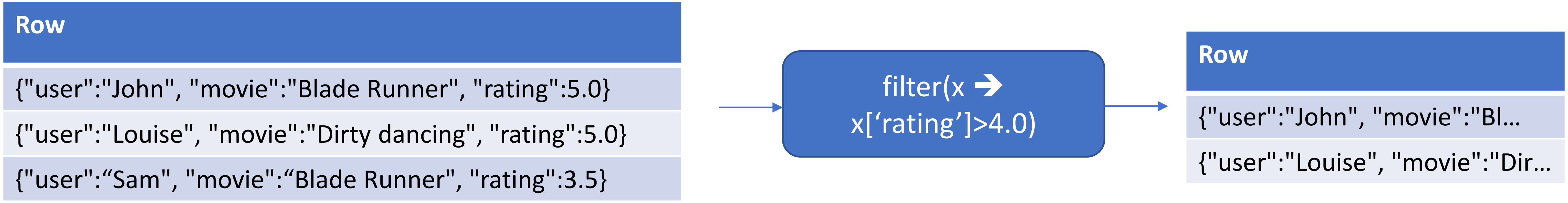
RDD API



Row
{"movie":"Blade Runner", “genres":”cyberpunk;scifi;action”}
{"movie":"Dirty dancing", “genres":”music;danse;romance”}

flatmap(x ➔ x[‘genres’].split(‘;’))

Row
cyberpunk
scifi
action
music
danse
romance

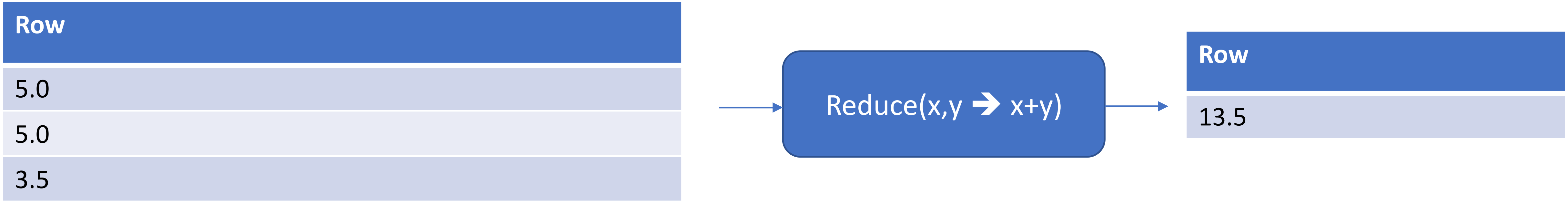


Row
{"user":"John", "movie":"Blade Runner", "rating":5.0}
{"user":"Louise", "movie":"Dirty dancing", "rating":5.0}
{"user":“Sam", "movie":“Blade Runner", "rating":3.5}

filter(x ➔ x[‘rating’]>4.0)

Row
{"user":"John", "movie":"Bl…
{"user":"Louise", "movie":"Dir…

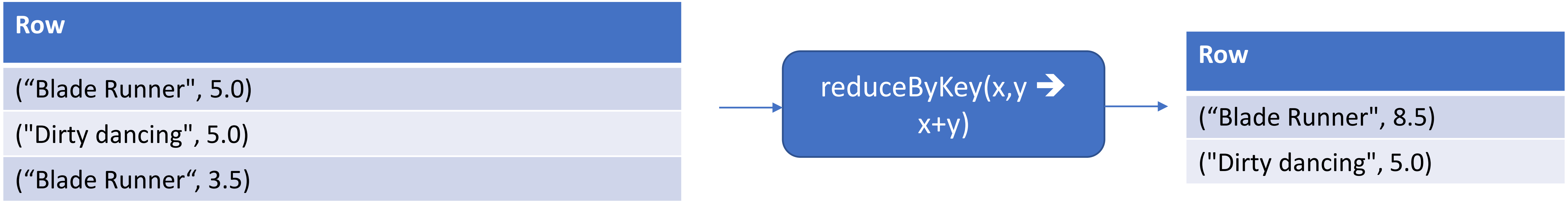
RDD API - Aggregations



Row
5.0
5.0
3.5

Reduce(x,y ➔ x+y)

Row
13.5



Row
(“Blade Runner", 5.0)
("Dirty dancing", 5.0)
(“Blade Runner“, 3.5)

reduceByKey(x,y ➔ x+y)

Row
(“Blade Runner", 8.5)
("Dirty dancing", 5.0)

Other useful functions

- **Join (shuffle ? Lazy ?)**
- **Sample**
- **mappartitions**
- **zippartitions**

Links

- https://spark.apache.org/docs/latest/api/scala/org/apache/spark/api/java/JavaPairRDD.html
- https://0x0fff.com/hadoop-mapreduce-comprehensive-description/