Apache Spark vs MapReduce

Apache Spark

Distributed computing framework (we will run it on YARN)

API in several languages: Scala, Java, **Python (PySpark)**

Includes a lot of stuff: Spark ML, Spark SQL, Spark Streaming, etc.

http://spark.apache.org

SQL join query

Table **a** – purchases (**user**, product, …)

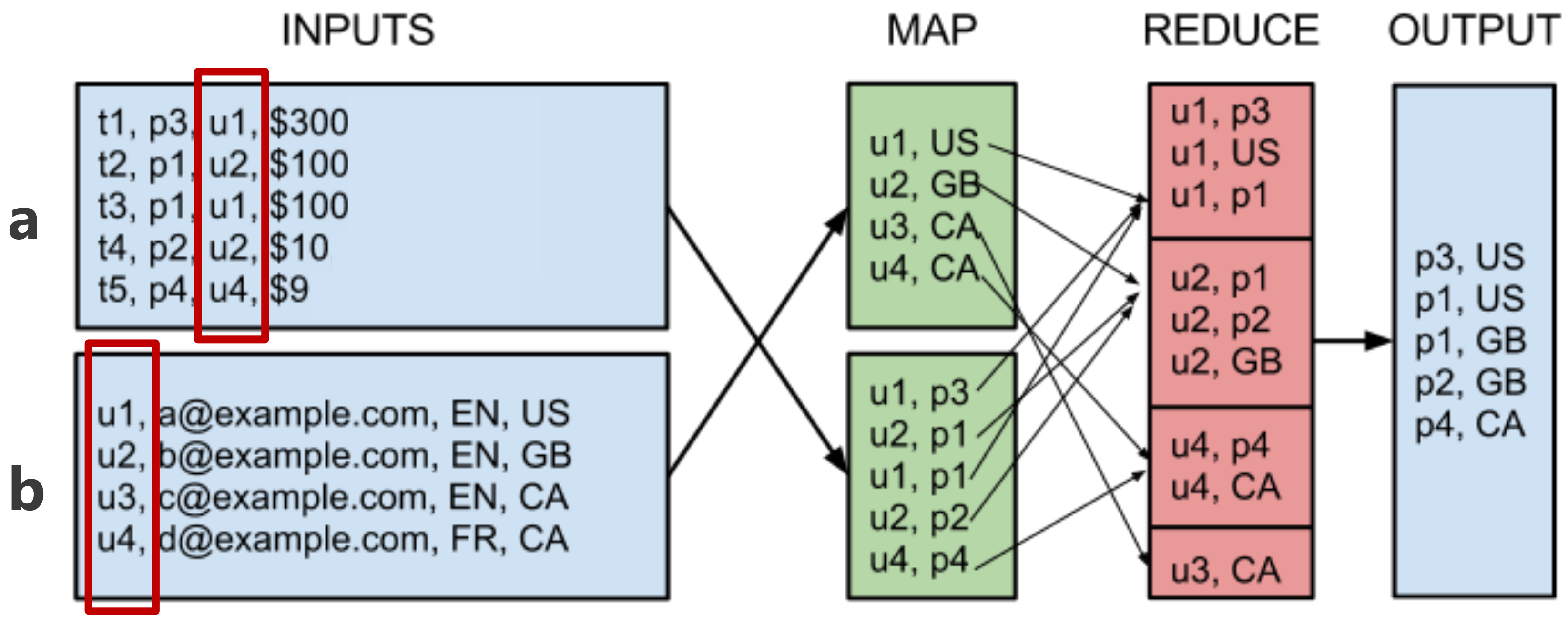
Table **b** – contact info (**user**, country, …)

We want to know the country where each purchase took place.

We need to join these tables on **user**:

```
select
  a.product,
  b.country
from
  a join b on a.user = b.user
```

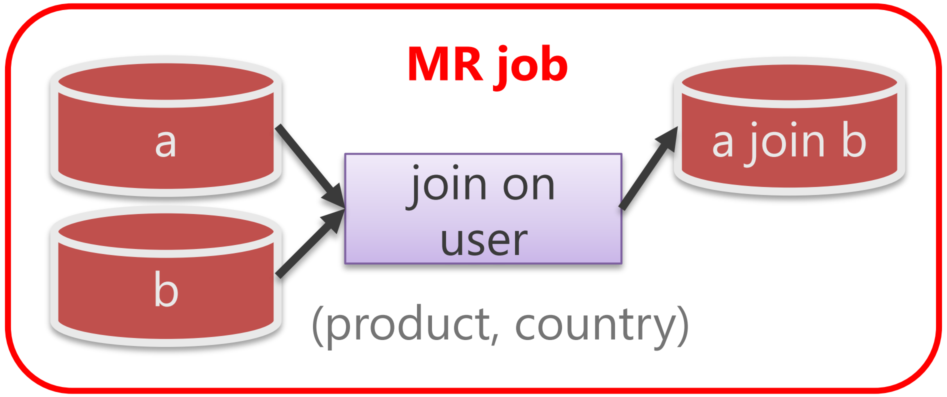
SQL join on MapReduce



One more join on MapReduce

Table **c** contains product info (**product**, category, ...)

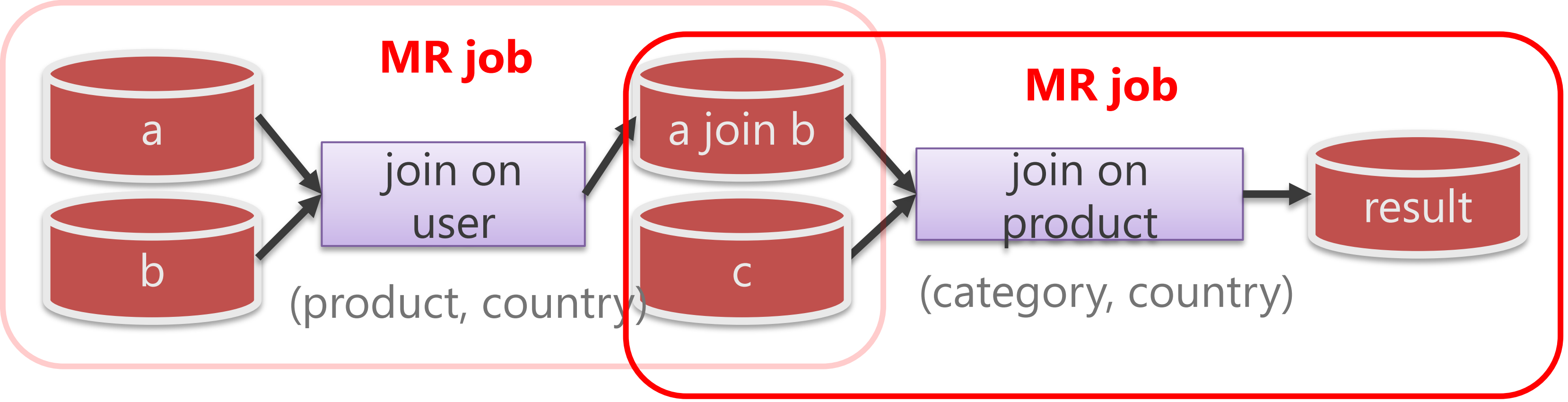
First join on MapReduce:



One more join on MapReduce

Table **c** contains product info (**product**, category, …)

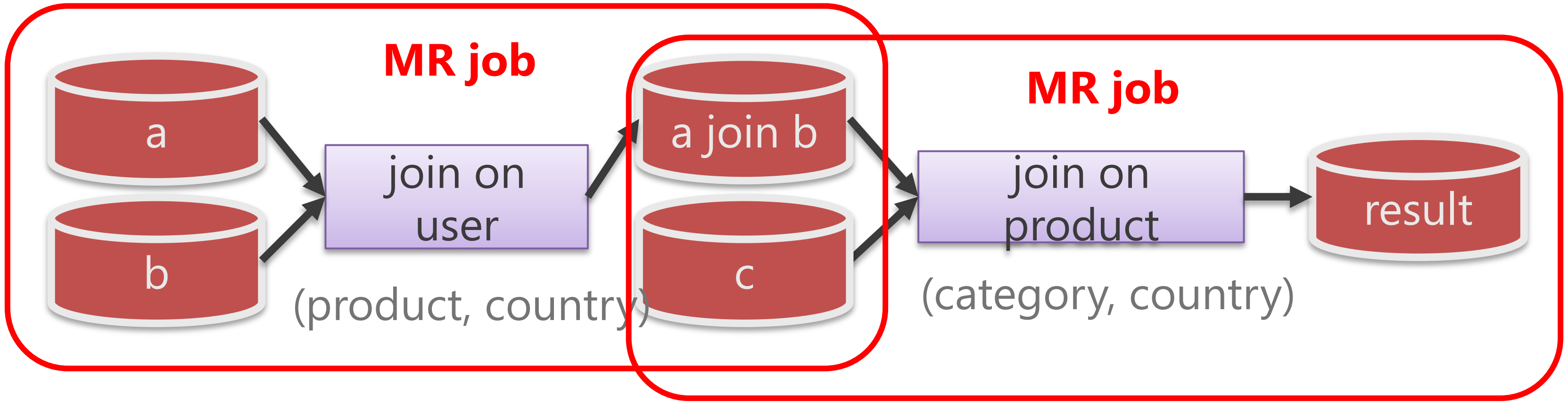
Second join on MapReduce:



One more join on MapReduce

Table **c** contains product info (**product**, category, ...)

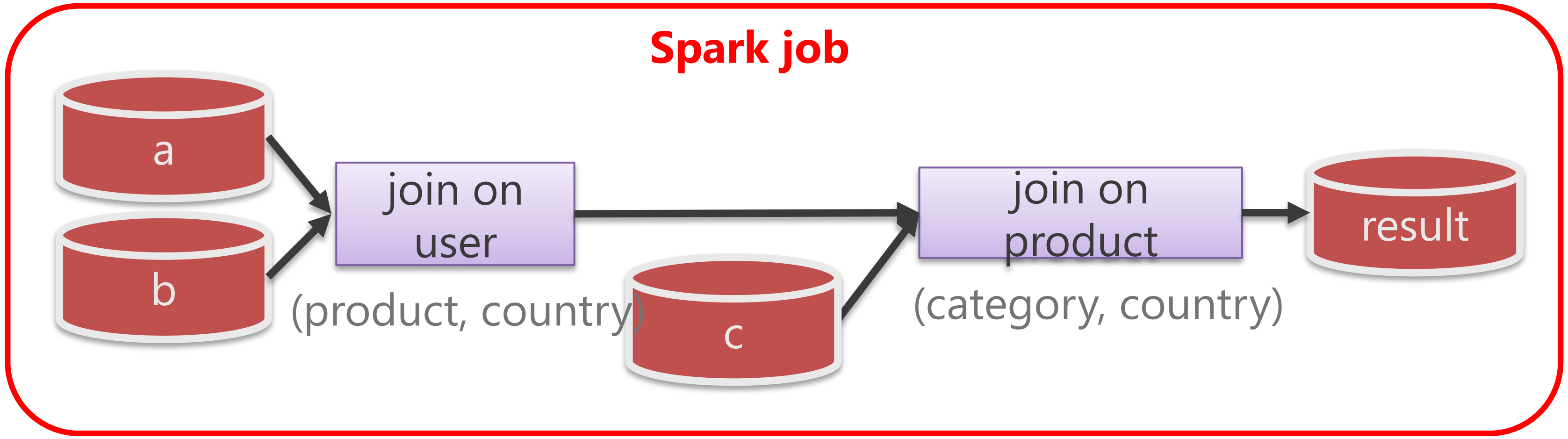
Two joins on MapReduce:



- MapReduce results are stored in HDFS
- So, for “a join b” we spend time on writing to HDFS and reading it back

That's where Spark shines

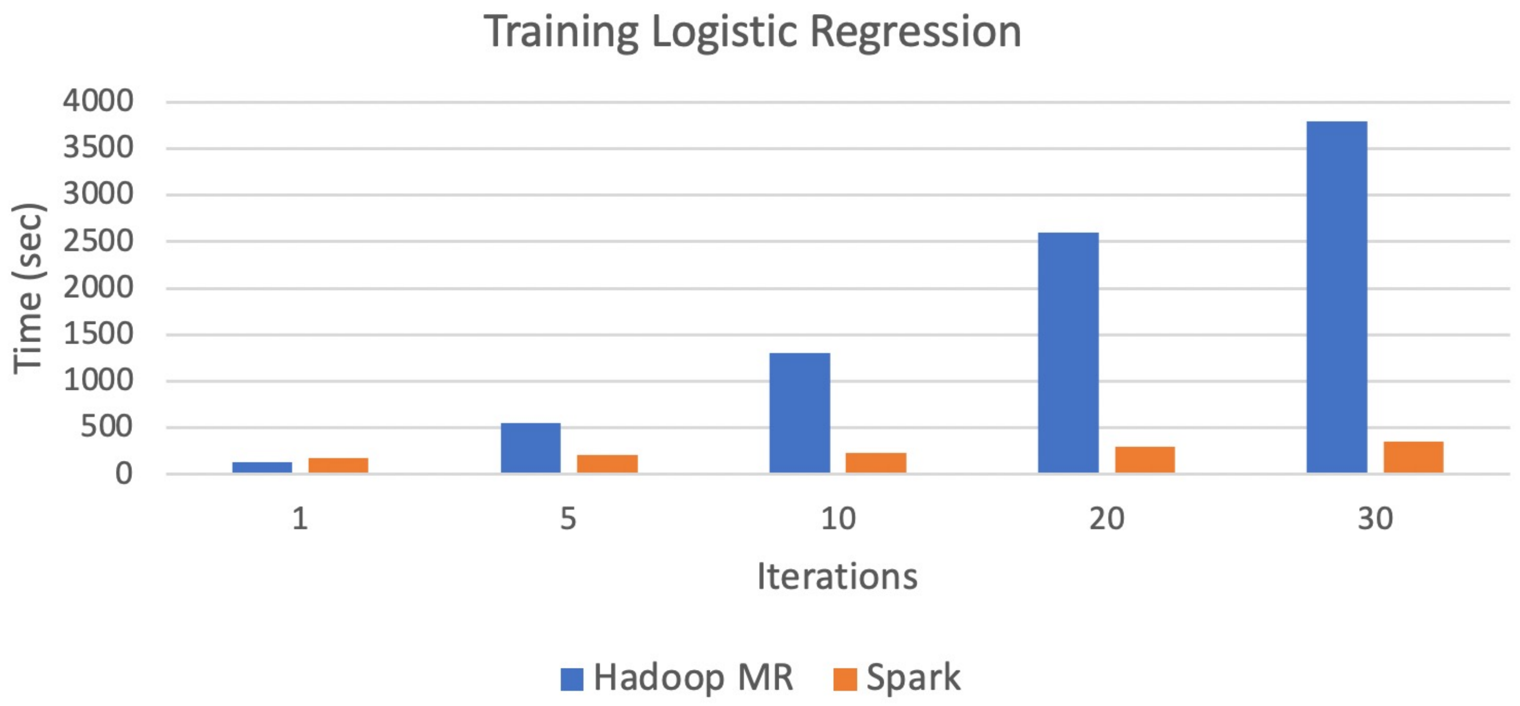
Two joins on Spark:



- Computation is described as a DAG (Directed Acyclical Graph)
- Intermediate results can be cached in memory or on disk, skipping HDFS

Iterative algorithms

- MapReduce has overheads:
  - Reading and writing to/from HDFS
  - Starting up each MR job on YARN cluster (scheduling, starting JVM, etc.)
- In ML we have lots of iterative algorithms and Spark works best on them:



https://www.edureka.co/blog/apache-spark-vs-hadoop-mapreduce

Spark vs MapReduce

	Spark	MapReduce
Usage	Iterative and interactive computation	Heavy load offline processing
Simplicity	Python API	Hadoop text-based streaming
Storage	Utilizes RAM for cache	Everything is stored in HDFS

Spark RDD API

Spark RDD

Spark program is a set of operations on RDDs

Abstract RDD – resilient distributed dataset:

- Job inputs and outputs are RDDs
- All intermediate results are RDDs as well (you know the DAG and can restore missing parts if you lose them)

https://spark.apache.org/docs/latest/rdd-programming-guide.html

Spark RDD

Spark program is a set of operations on RDDs

Abstract RDD – resilient distributed dataset:

- Job inputs and outputs are RDDs
- All intermediate results are RDDs as well (you know the DAG and can restore missing parts if you lose them)

How to make an RDD:

- A file from HDFS (already resilient and distributed)
- Parallelize a Python collection (list, generator, …)
- Transform another RDD

https://spark.apache.org/docs/latest/rdd-programming-guide.html

RDD operations

Transformations (RDD → RDD):

Transformation is lazy (i.e. computed when it's needed)

Example: **map** is a transformation that passes each dataset element through a function and returns a new RDD representing the results

Other examples: **reduceByKey, join**

RDD operations

Transformations (RDD → RDD):

Transformation is lazy (i.e. computed when it's needed)

Example: **map** is a transformation that passes each dataset element through a function and returns a new RDD representing the results

Other examples: **reduceByKey, join**

Actions:

Action triggers DAG execution to compute RDD

Examples: **saveAsTextFile, collect, count**

RDD operations

Transformations (RDD → RDD):

Transformation is lazy (i.e. computed when it's needed)

Example: **map** is a transformation that passes each dataset element through a function and returns a new RDD representing the results

Other examples: **reduceByKey, join**

Actions:

Action triggers DAG execution to compute RDD

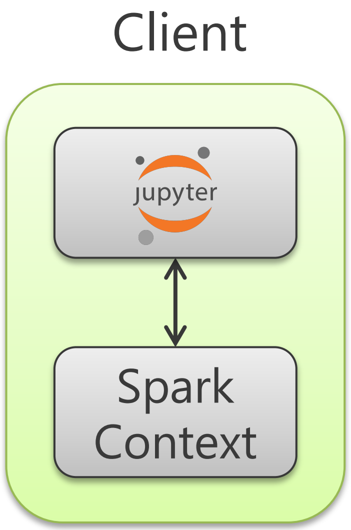
Examples: **saveAsTextFile, collect, count**

Other operations:

Example: **persist, cache** will force Spark to keep the RDD in memory for much faster access the next time you query it

How PySpark program works (yarn cluster mode)

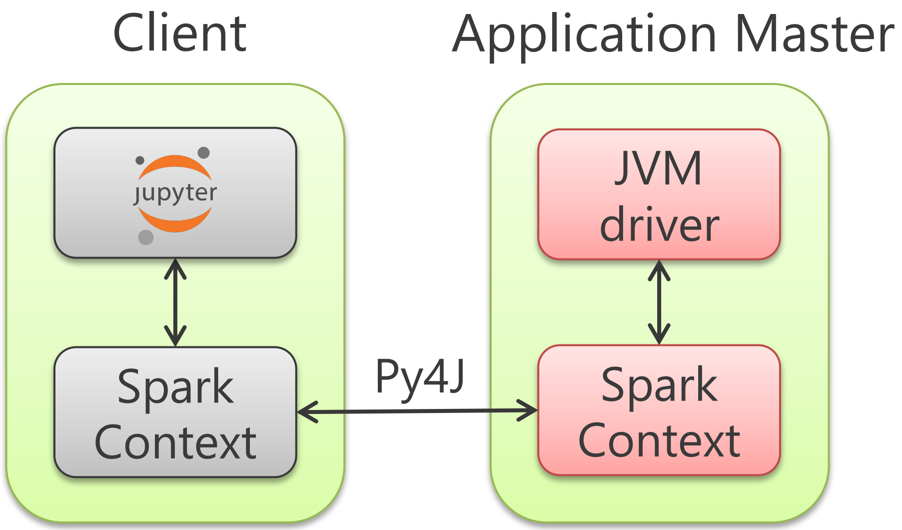
You create **SparkContext** in Python to communicate with Spark cluster.



How PySpark program works (yarn cluster mode)

You create **SparkContext** in Python to communicate with Spark cluster.

YARN application is started, running **driver** as Application Master, which creates JVM version of **SparkContext** (stores configuration, DAGs for all RDDs, etc.).

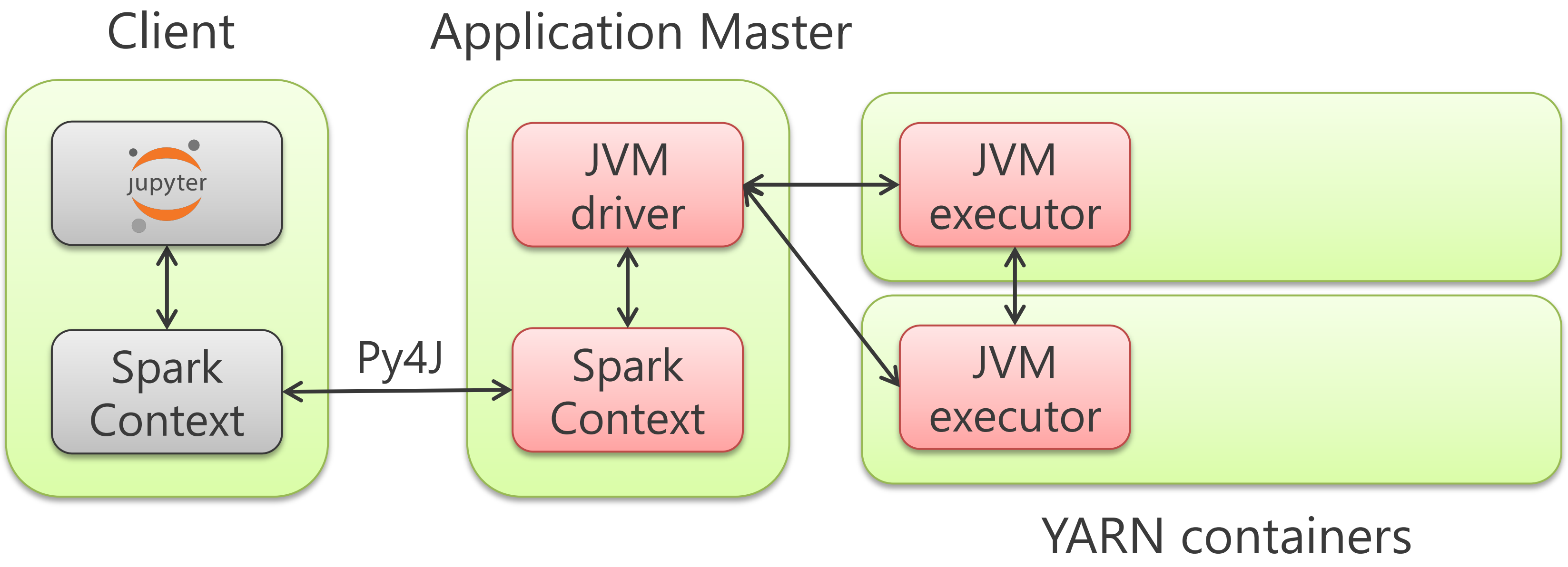


How PySpark program works (yarn cluster mode)

You create **SparkContext** in Python to communicate with Spark cluster.

YARN application is started, running **driver** as Application Master, which creates JVM version of **SparkContext** (stores configuration, DAGs for all RDDs, etc.).

Driver coordinates the work of **Executors** (making actual computation).

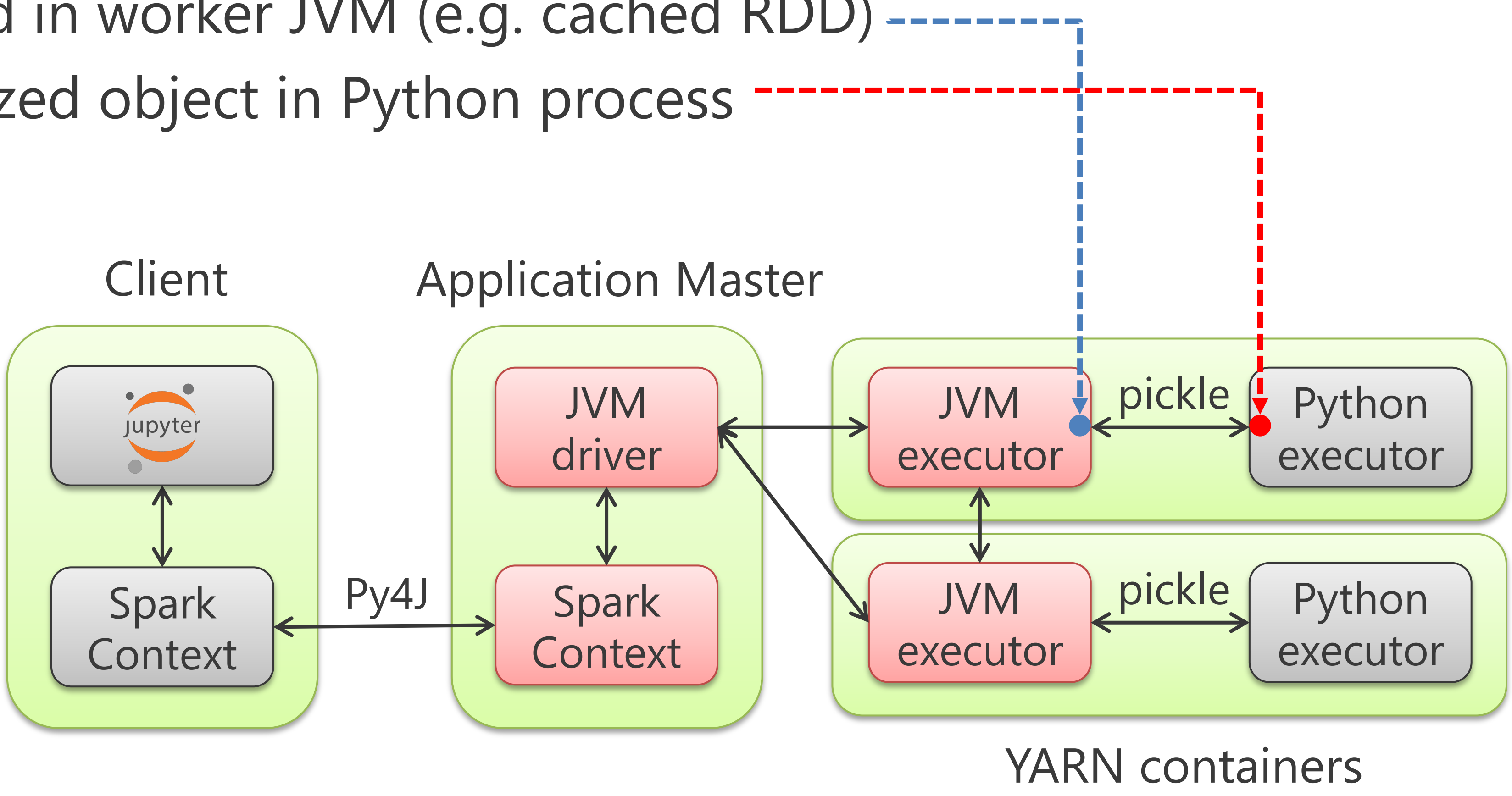


How PySpark program works (yarn cluster mode)

Python objects are serialized with pickle and need deserialization to be processed

Which means objects are stored **twice** in memory:

- Serialized in worker JVM (e.g. cached RDD)
- Deserialized object in Python process



PySpark code example

```
rdd = (sc                                # SparkContext
       .parallelize([1, 2, 3, 4])        # make RDD
       .map(lambda x: x * 2))            # transform RDD
print rdd                                # lazy DAG
print rdd.collect()                      # run DAG
```

```
PythonRDD[17] at RDD at PythonRDD.scala:48
[2, 4, 6, 8]
```

One more transformation example

```
rdd = (sc
       .parallelize([1, 2, 3, 4])
       .flatMap(lambda x: [x, x * 2]))
print rdd
print rdd.collect()
```

```
PythonRDD[19] at RDD at PythonRDD.scala:48
[1, 2, 2, 4, 3, 6, 4, 8]
```

One more action example

```
rdd = (sc
       .parallelize(np.random.random((1000,)))
       .flatMap(lambda x: [x, x * 2]))
print rdd
print rdd.takeOrdered(2, lambda x: -x)
```

```
PythonRDD[29] at RDD at PythonRDD.scala:48
[1.9987386963918603, 1.997388155520317]
```

Word Count in Spark

```
rdd = (
    sc
   .parallelize(["this is text", "text too"])
   .flatMap(lambda x: [(w, 1) for w in x.split()])
   .reduceByKey(lambda a, b: a + b))
print rdd
print rdd.collect()
```

```
PythonRDD[61] at RDD at PythonRDD.scala:48
[('text', 2), ('too', 1), ('is', 1), ('this', 1)]
```

Spark quiz

```
rdd = sc.parallelize(range(1000))
rdd = rdd.map(lambda x: (x % 100, 1))
rdd = rdd.reduceByKey(lambda a, b: a + b)
rdd = rdd.map(lambda (a, b): (b, a))
rdd = rdd.reduceByKey(lambda a, b: max(a, b))
rdd.collect()
```

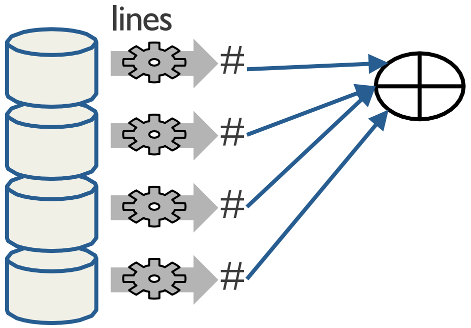
Spark quiz

```
rdd = sc.parallelize(range(1000))
rdd = rdd.map(lambda x: (x % 100, 1))
rdd = rdd.reduceByKey(lambda a, b: a + b)
rdd = rdd.map(lambda (a, b): (b, a))
rdd = rdd.reduceByKey(lambda a, b: max(a, b))
rdd.collect()
```

```
[(10, 99)]
```

Caching in RAM

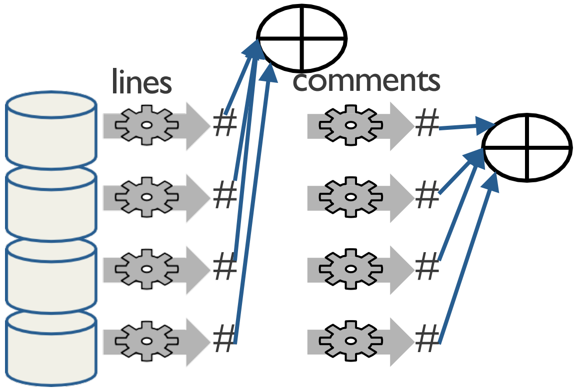
```
lines = sc.textFile("...", 4)

print lines.count()
```



`count()` causes Spark to:
- read data
- sum within partitions
- combine sums in driver

Caching in RAM

```
lines = sc.textFile("...", 4)
comments = lines.filter(isComment)
print lines.count(), comments.count()
```

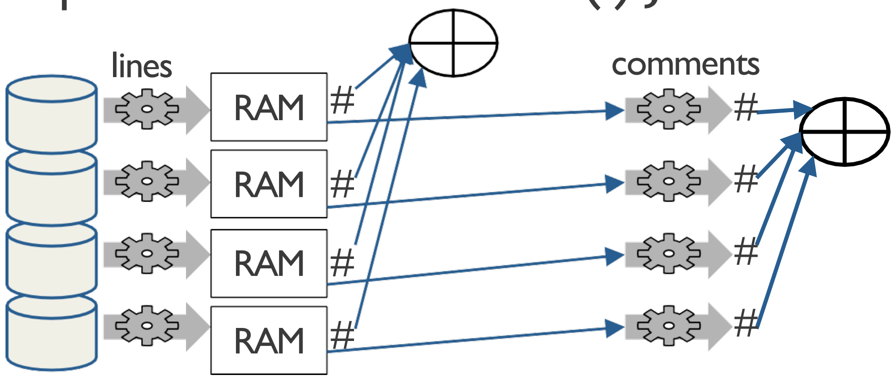


Spark recomputes lines:

- read data (again)
- sum within partitions
- combine sums in driver

Caching in RAM

```
lines = sc.textFile("...", 4)
lines.cache() # save, don't recompute!
comments = lines.filter(isComment)
print lines.count(),comments.count()
```



Conclusion

- Spark is easier to use than MapReduce
- Very flexible due to pickle serialization (works with virtually any Python object)
- But not as fast as Scala/Java version (you can compensate with more nodes)