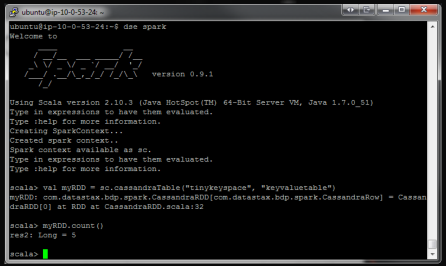
PySpark at a Glance

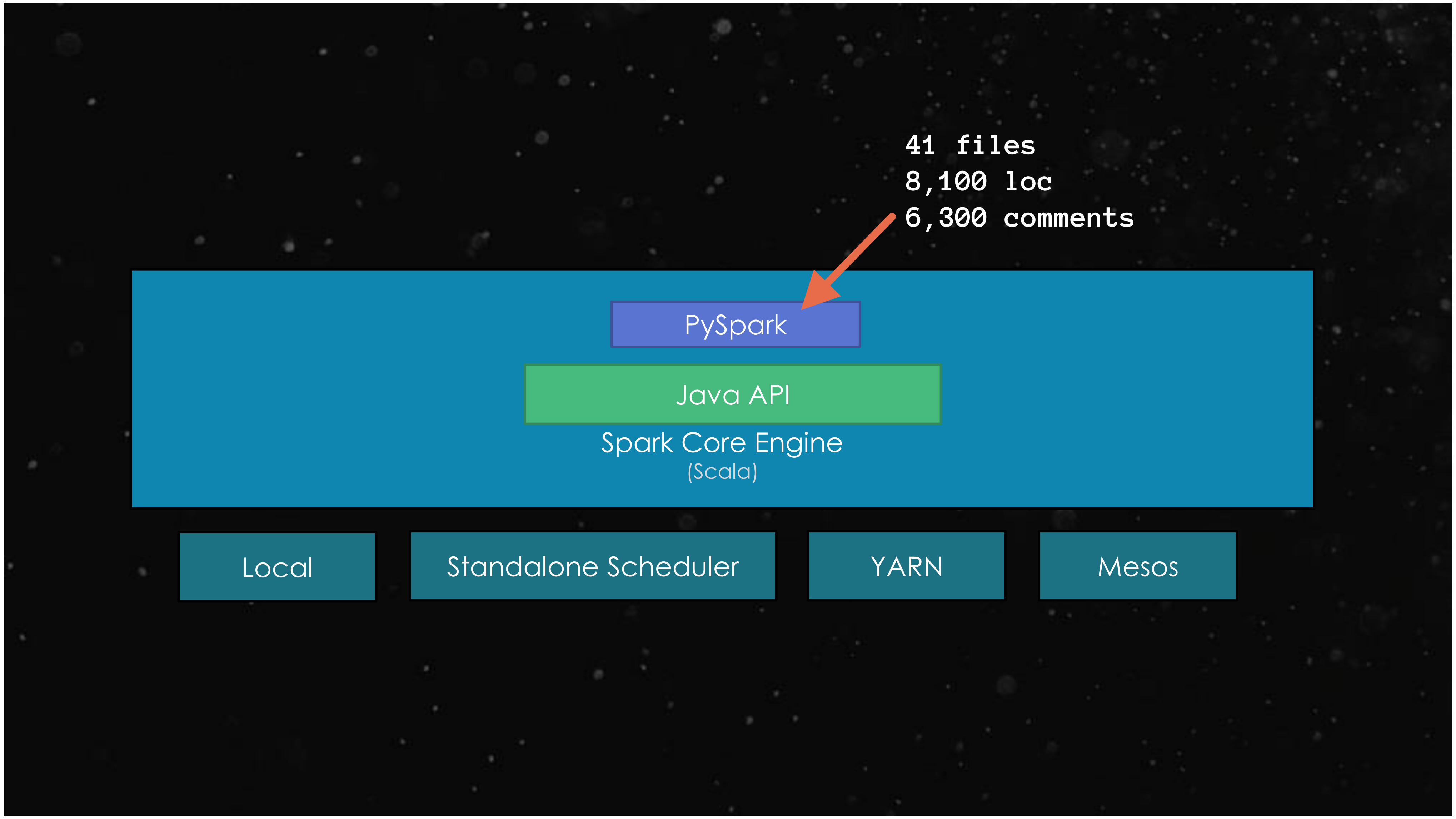
Write Spark jobs in Python



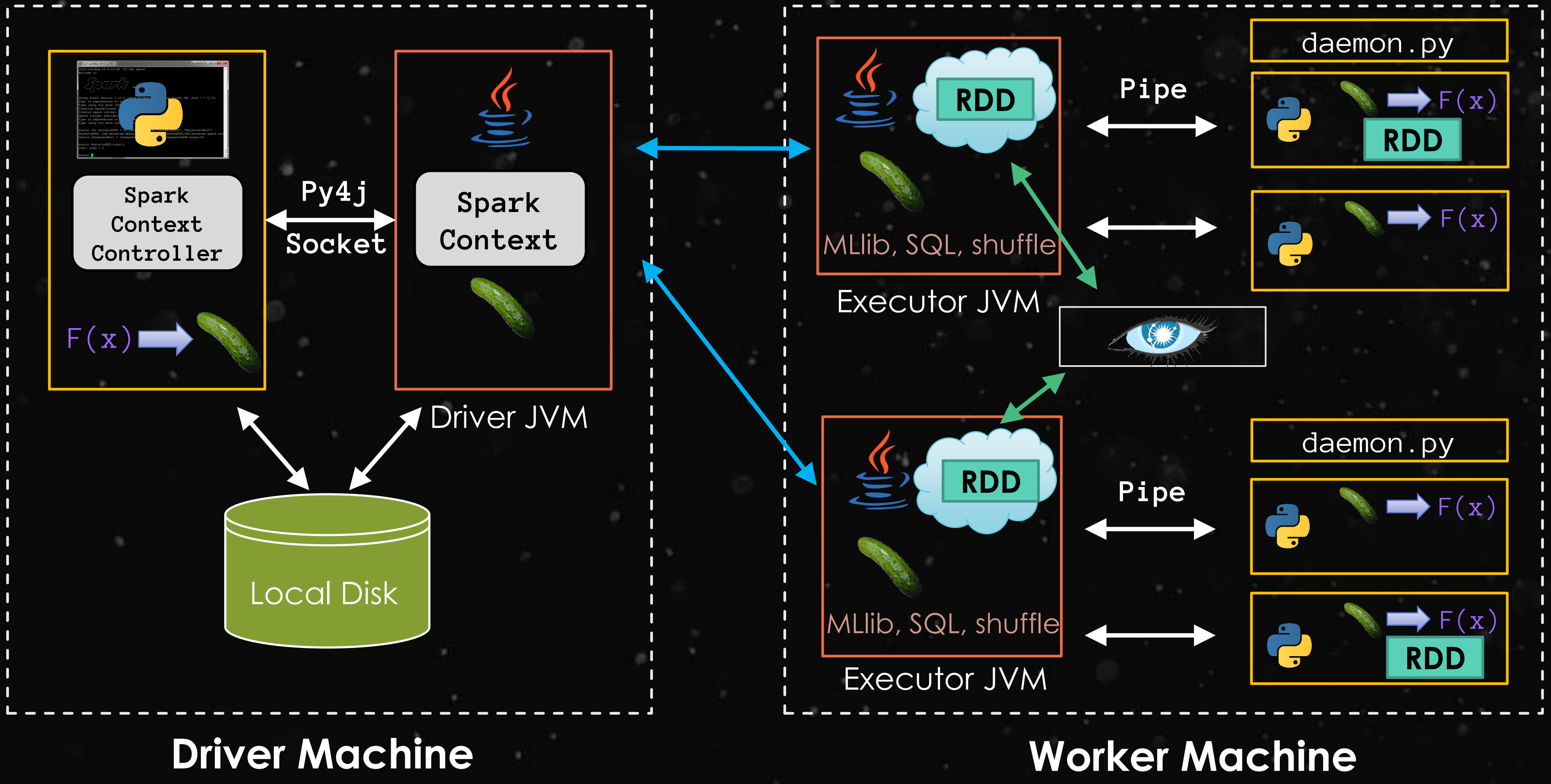
```
ubuntu@ip-10-0-53-24:~$ dse spark
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 0.9.1
      /_/

Using Scala version 2.10.3 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_51)
Type in expressions to have them evaluated.
Type :help for more information.
Creating SparkContext...
Created spark context..
Spark context available as sc.
Type in expressions to have them evaluated.
Type :help for more information.

scala> val myRDD = sc.cassandraTable("tinykeyspace", "keyvaluetable")
myRDD: com.datastax.bdp.spark.CassandraRDD[com.datastax.bdp.spark.CassandraRow] = Cassan
draRDD[0] at RDD at CassandraRDD.scala:32

scala> myRDD.count()
res2: Long = 5

scala>
```

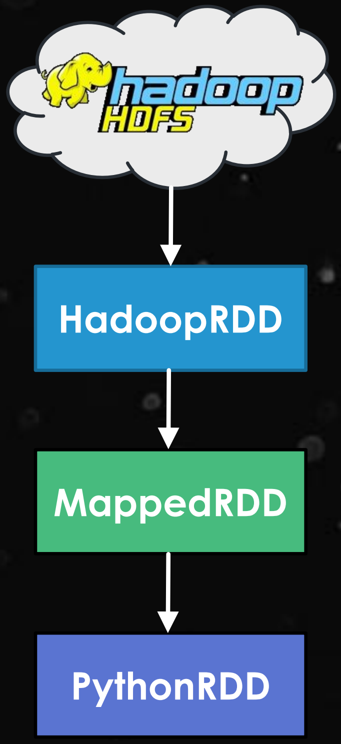
Run interactive jobs in the shell

Supports C extensions


41 files
8,100 loc
6,300 comments
PySpark
Java API
Spark Core Engine
(Scala)
Local
Standalone Scheduler
YARN
Mesos

PYSPARK ARCHITECTURE

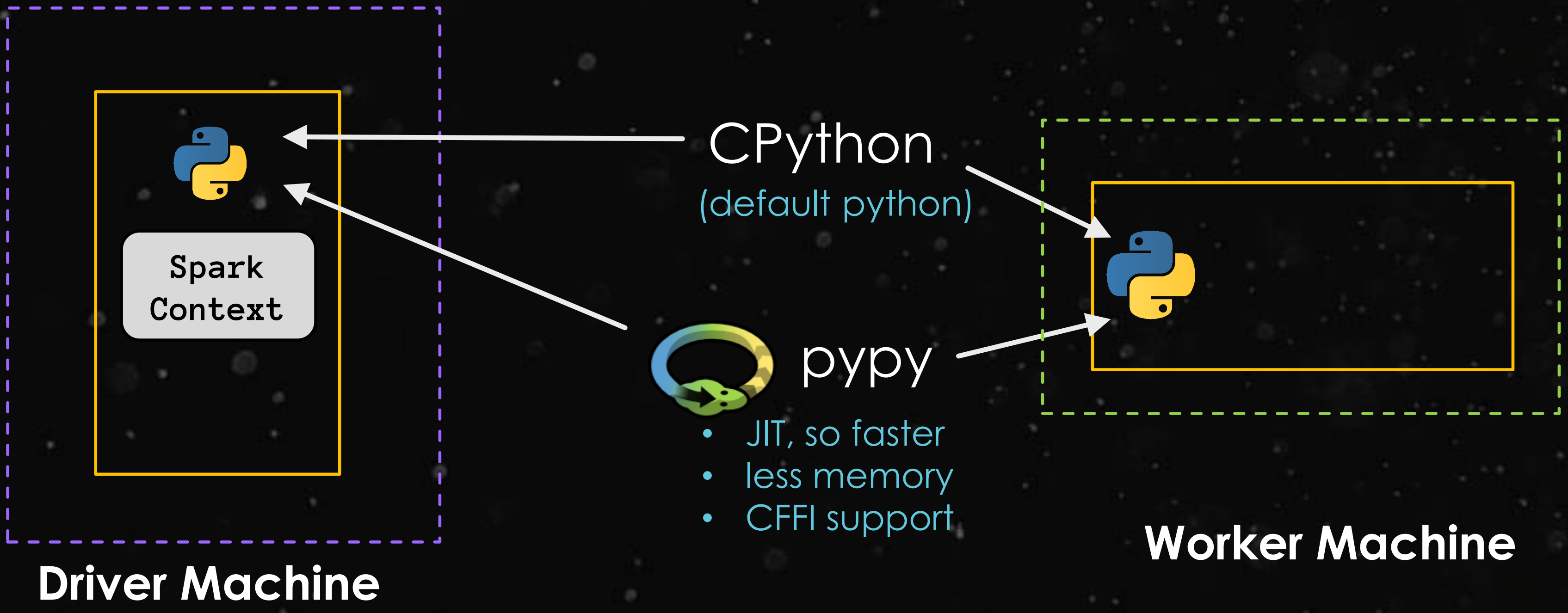




Data is stored as Pickled objects in an RDD[Array[Byte]]

(100 KB – 1MB each picked object)

Choose Your Python Implementation



```
$ PYSPARK_DRIVER_PYTHON=pypy PYSPARK_PYTHON=pypy ./bin/pyspark
```

OR

```
$ PYSPARK_DRIVER_PYTHON=pypy PYSPARK_PYTHON=pypy ./bin/spark-submit wordcount.py
```

The performance speed up will depend on work load (from 20% to 3000%).

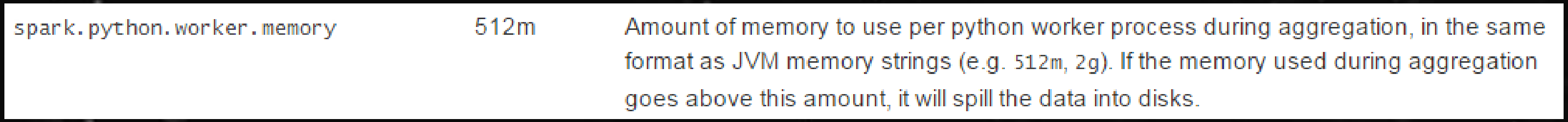
Here are some benchmarks:

Job	CPython 2.7	PyPy 2.3.1	Speed up
Word Count	41 s	15 s	2.7 x
Sort	46 s	44 s	1.05 x
Stats	174 s	3.6 s	48 x

Here is the code used for benchmark:

```
rdd = sc.textFile("text")
def wordcount():
    rdd.flatMap(lambda x:x.split('/'))\
        .map(lambda x:(x,1)).reduceByKey(lambda x,y:x+y).collectAsMap()
def sort():
    rdd.sortBy(lambda x:x, 1).count()
def stats():
    sc.parallelize(range(1024), 20).flatMap(lambda x: xrange(5024)).stats()
```

https://github.com/apache/spark/pull/2144



`spark.python.worker.memory`	512m	Amount of memory to use per python worker process during aggregation, in the same format as JVM memory strings (e.g. `512m`, `2g`). If the memory used during aggregation goes above this amount, it will spill the data into disks.