pySpark

Python API for Spark

Josh Rosen

amplab
UC BERKELEY

What is Spark?

Spark

Fast and expressive cluster computing system

Compatible with Hadoop-supported file systems and data formats (HDFS, S3, SequenceFile, ...)

Improves *efficiency* through in-memory computing primitives and general computation graphs

As much as 30x faster

Improves *usability* through rich APIs in Scala, Python, and Java, and an interactive shell

Often 2-10x less code

RDDs

Resilient Distributed Datasets

Immutable, partitioned collections of objects

Transformations

```
map
filter
groupBy
join
...
```

Actions

```
count
collect
save
...
```

Example: Log Mining

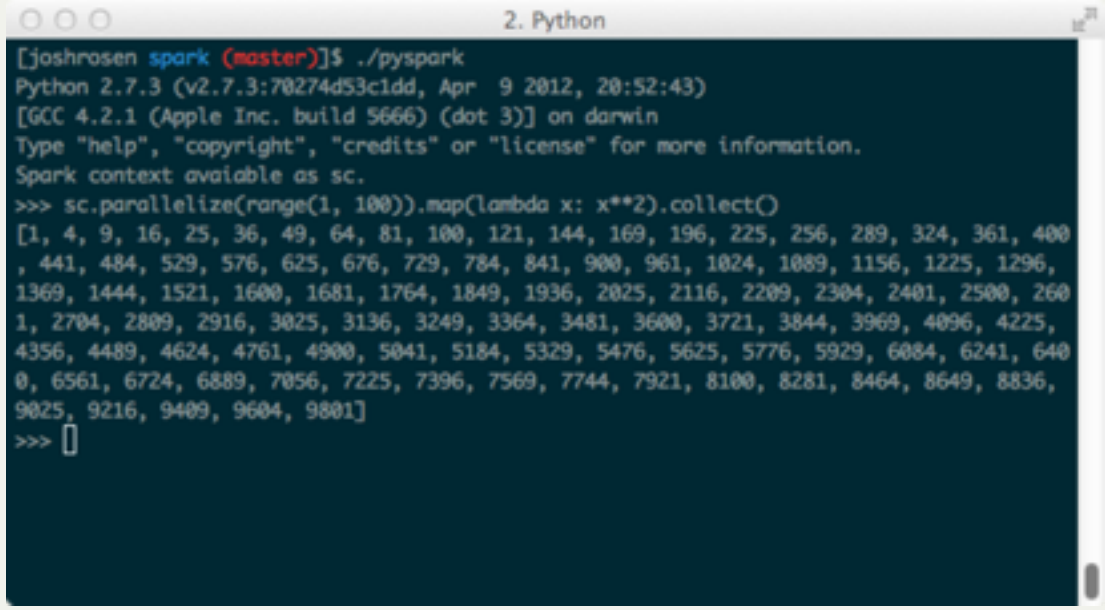
```
val lines = spark.textFile(“hdfs://...”)
val errors = lines.filter(_.startsWith(“ERROR”))
val messages = errors.map(_.split(‘\t’)(2))

messages.filter(_.contains(“foo”)).count
```

What is PySpark?

PySpark at a Glance

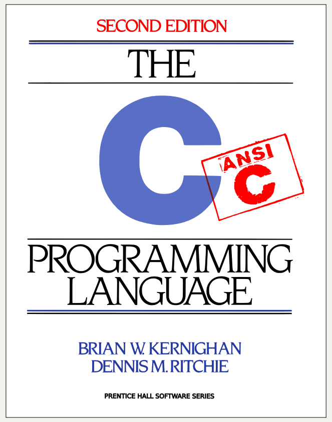
Write Spark jobs in Python

Run interactive jobs in the shell

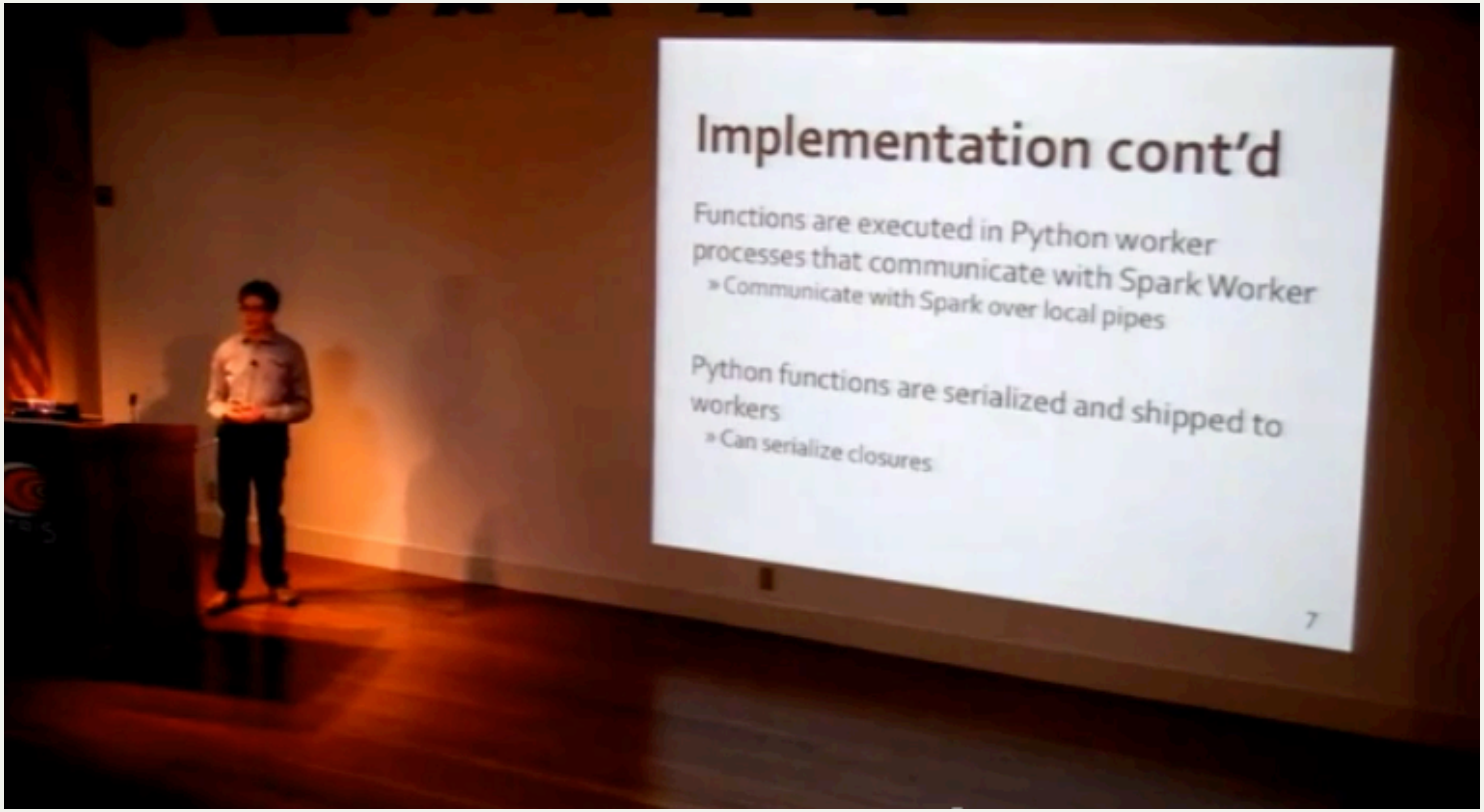


```
[joshrosen spark (master)]$ ./pyspark
Python 2.7.3 (v2.7.3:70274d53c1dd, Apr  9 2012, 20:52:43)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Spark context avaiable as sc.
>>> sc.parallelize(range(1, 100)).map(lambda x: x**2).collect()
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400
, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 961, 1024, 1089, 1156, 1225, 1296,
1369, 1444, 1521, 1600, 1681, 1764, 1849, 1936, 2025, 2116, 2209, 2304, 2401, 2500, 260
1, 2704, 2809, 2916, 3025, 3136, 3249, 3364, 3481, 3600, 3721, 3844, 3969, 4096, 4225,
4356, 4489, 4624, 4761, 4900, 5041, 5184, 5329, 5476, 5625, 5776, 5929, 6084, 6241, 640
0, 6561, 6724, 6889, 7056, 7225, 7396, 7569, 7744, 7921, 8100, 8281, 8464, 8649, 8836,
9025, 9216, 9409, 9604, 9801]
>>> 
```

Supports C extensions



Previewed at AMP Camp 2012



Implementation cont'd

Functions are executed in Python worker processes that communicate with Spark Worker

- » Communicate with Spark over local pipes

Python functions are serialized and shipped to workers

- » Can serialize closures

Available now in 0.7 release

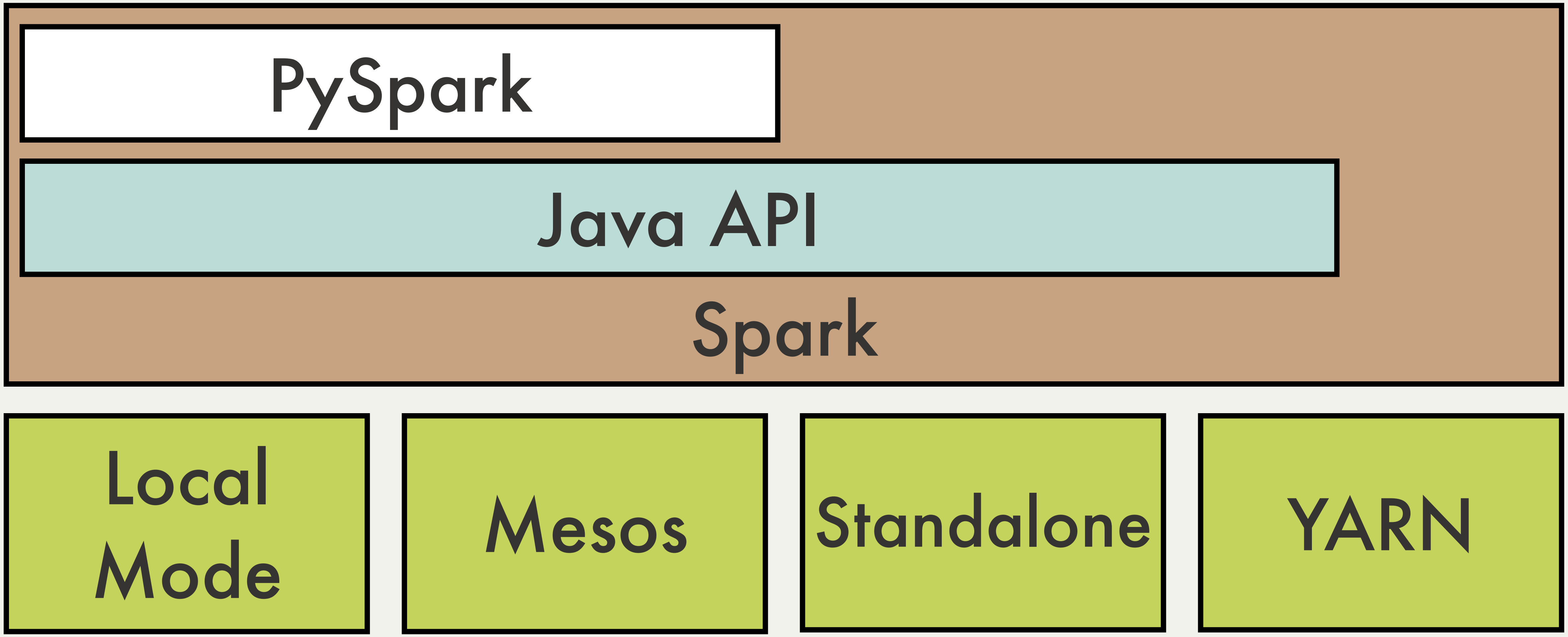
Example: Word Count

```
from pyspark.context import SparkContext

sc = SparkContext(...)
lines = sc.textFile(sys.argv[2], 1)
counts = lines.flatMap(lambda x: x.split(' ')) \
         .map(lambda x: (x, 1)) \
         .reduceByKey(lambda x, y: x + y)

for (word, count) in counts.collect():
  print "%s : %i" % (word, count)
```

Demo

Implementation

Built on top of Java API



Process data
in Python

and persist /
transfer it in Java

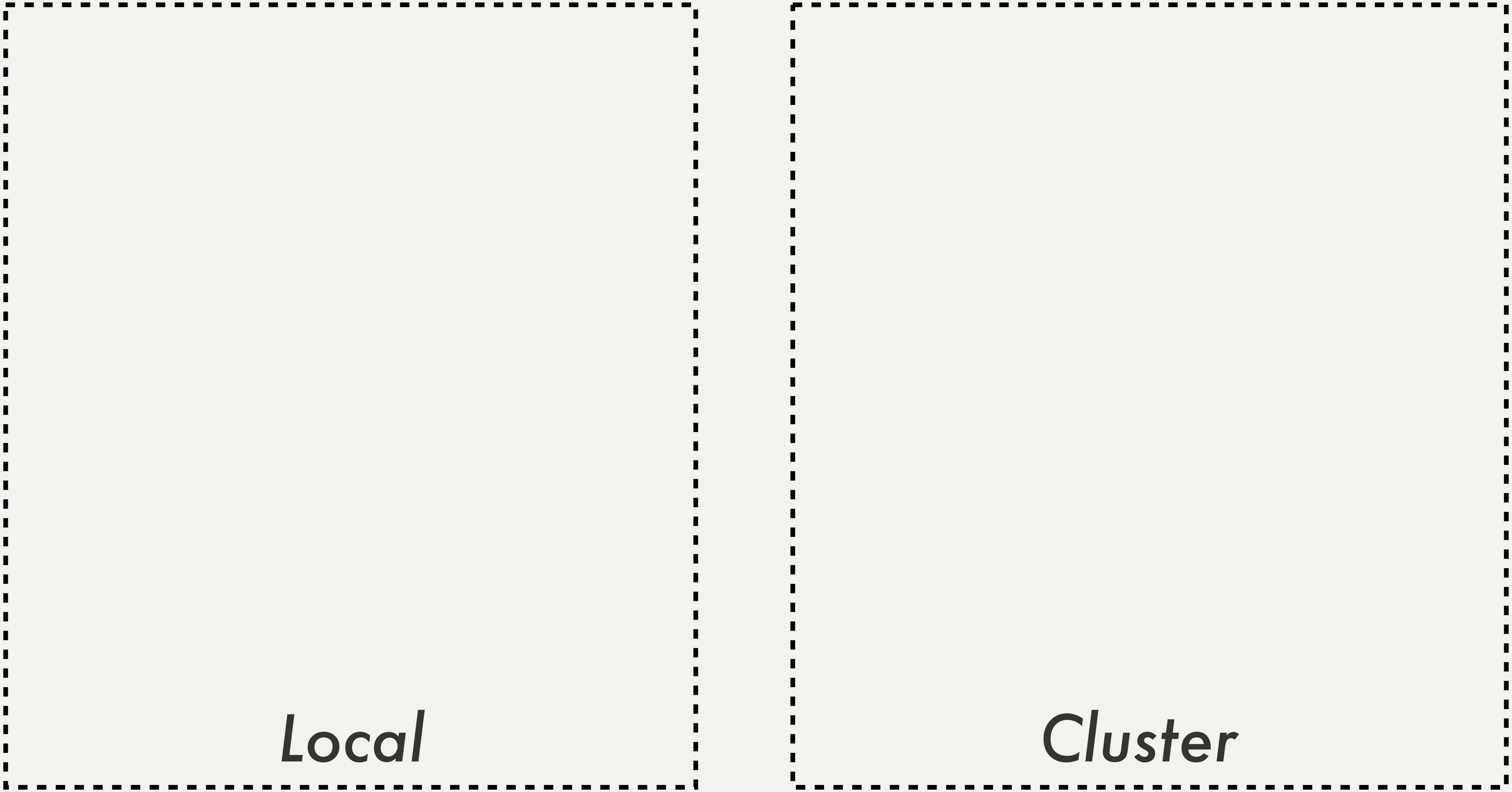
Re-uses Spark's scheduling
broadcast
checkpointing
networking
fault-recovery
HDFS access

PySpark has a small codebase:

```
---------------------------------------------------------------------------------------------------------------
File                                                                        blank        comment           code
---------------------------------------------------------------------------------------------------------------
python/pyspark/rdd.py                                                         115            345            302
core/src/main/scala/spark/api/python/PythonRDD.scala                           33             45            231
python/pyspark/context.py                                                      32            101            133
python/pyspark/tests.py                                                        26             11             84
python/pyspark/accumulators.py                                                 37             91             70
python/pyspark/serializers.py                                                  21              7             55
python/pyspark/join.py                                                         15             27             50
python/pyspark/worker.py                                                        8              7             44
core/src/main/scala/spark/api/python/PythonPartitioner.scala                    5              9             34
pyspark                                                                         9              8             27
python/pyspark/java_gateway.py                                                  5              7             26
python/pyspark/files.py                                                         7             14             17
python/pyspark/broadcast.py                                                     8             16             15
python/pyspark/shell.py                                                         4              6              8
python/pyspark/__init__.py                                                      6             14              7
---------------------------------------------------------------------------------------------------------------
SUM:                                                                          331            708           1103
---------------------------------------------------------------------------------------------------------------
```

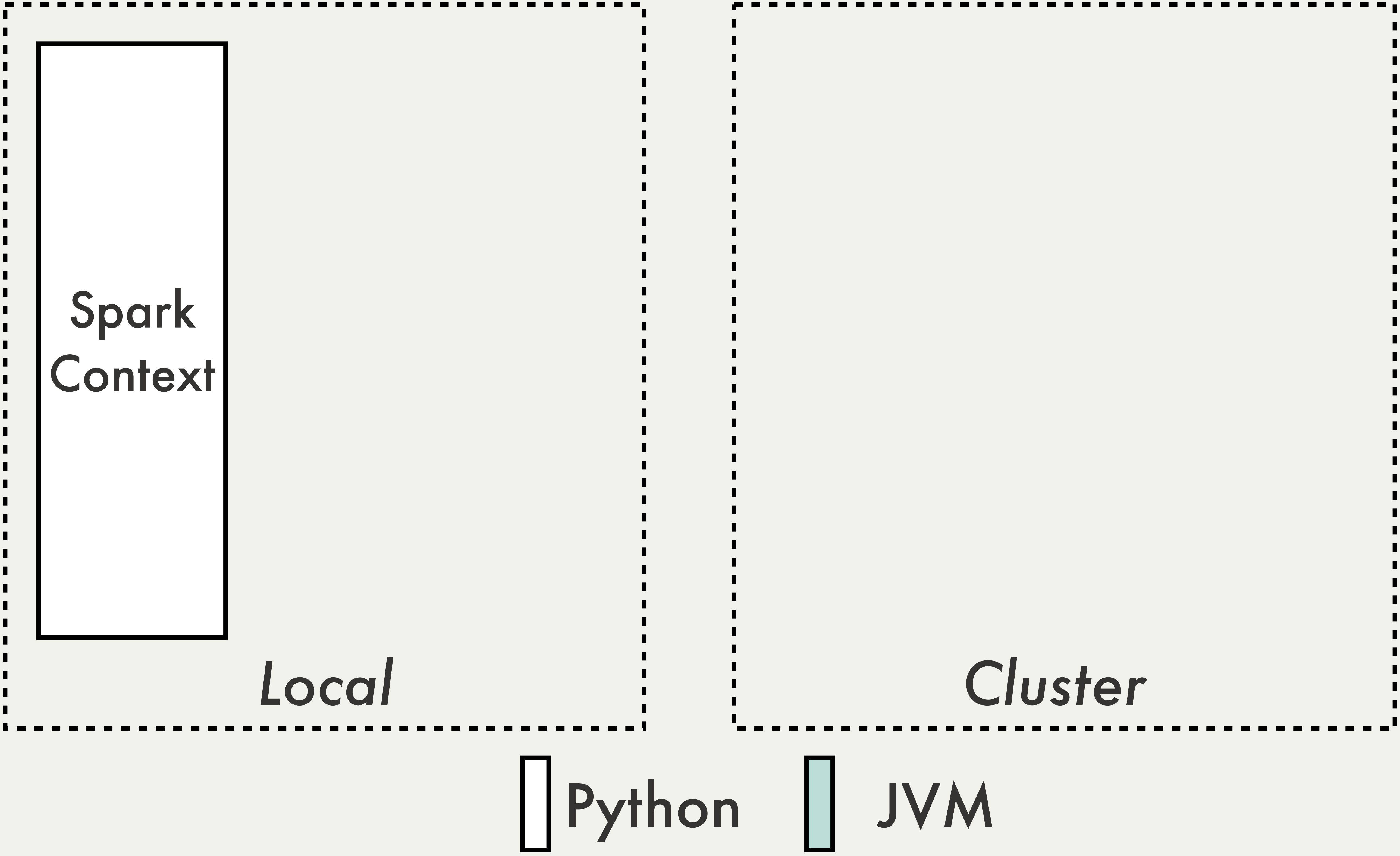
< 2K lines, including comments

Data Flow

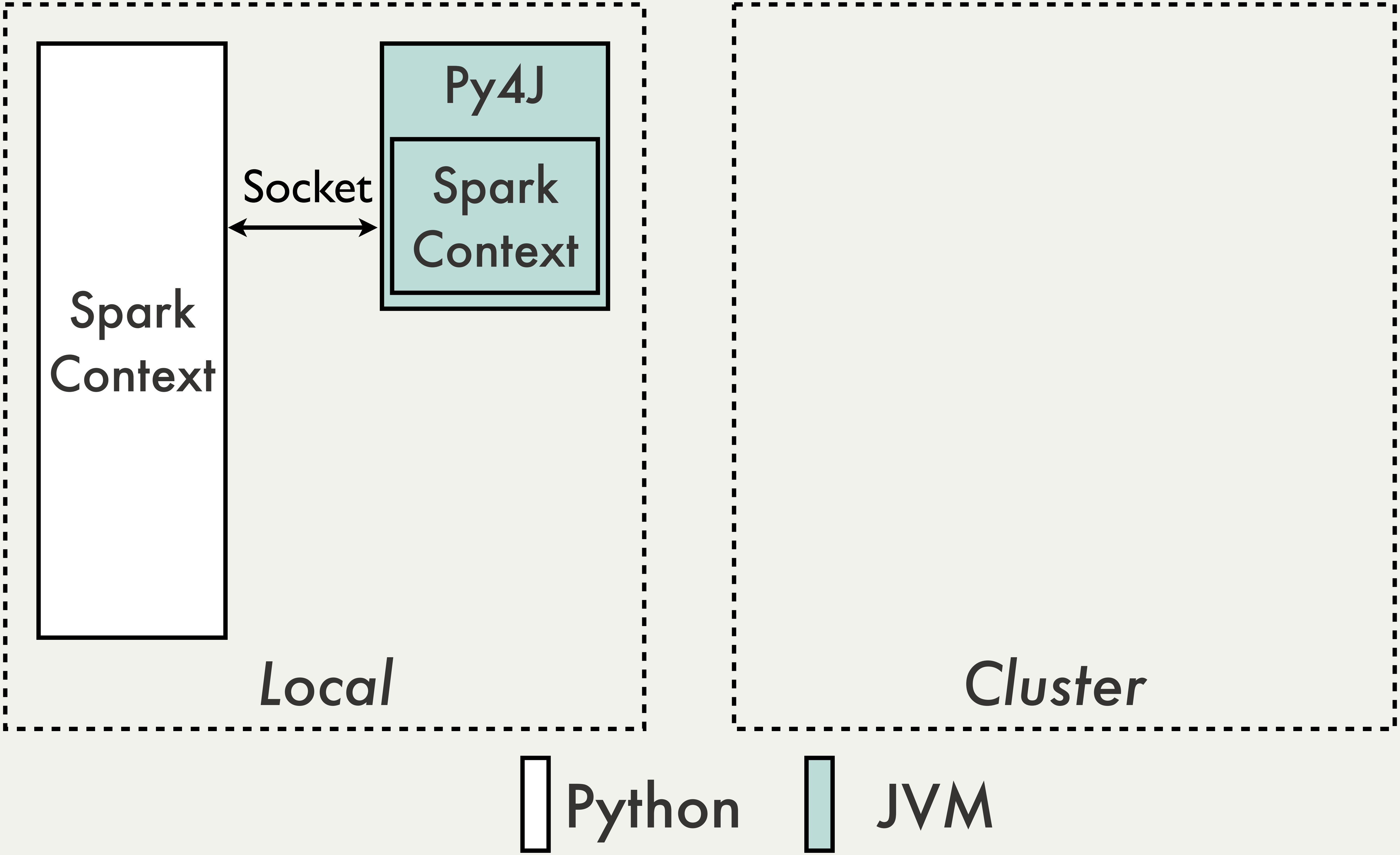


□ Python ▨ JVM

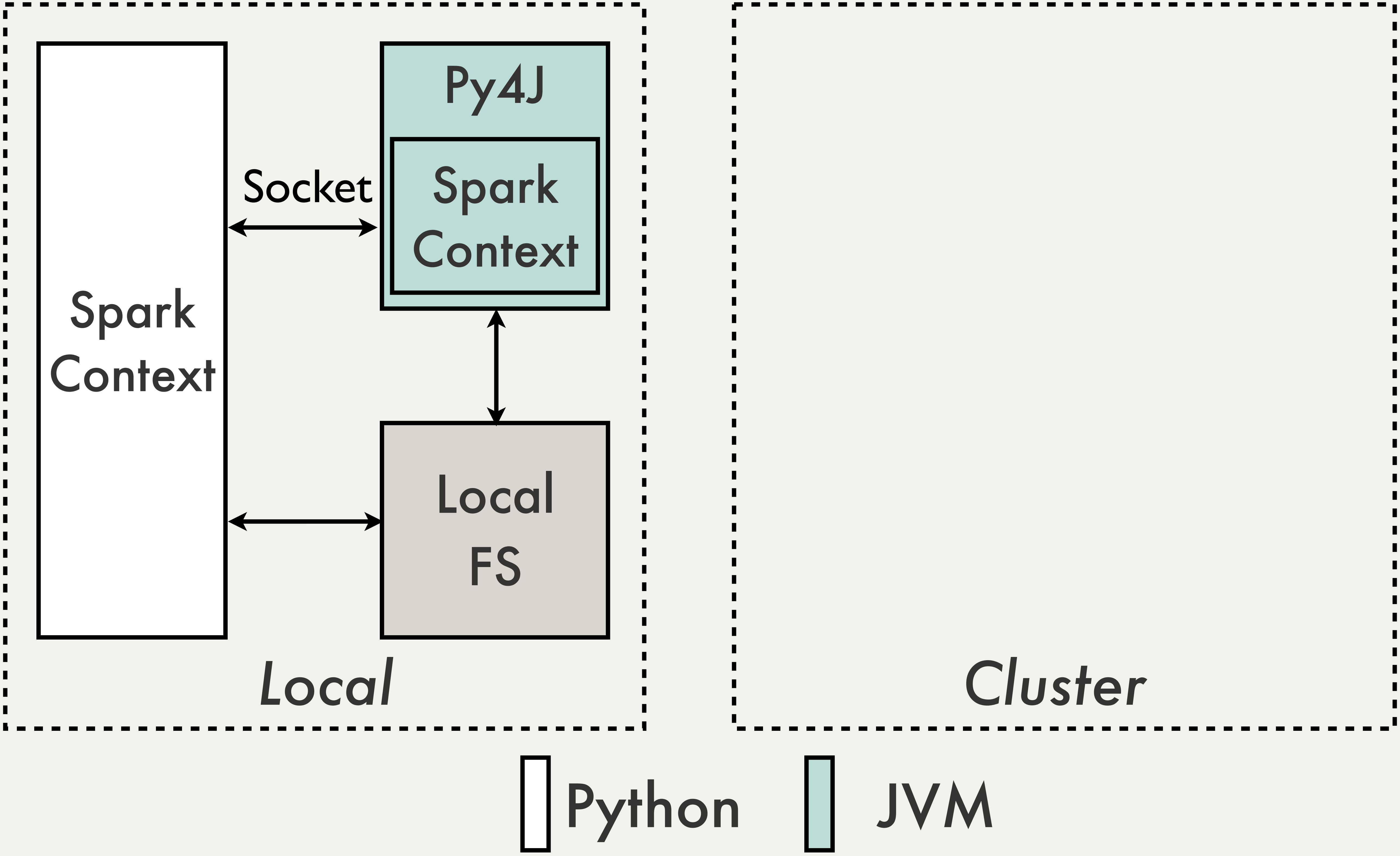
Data Flow



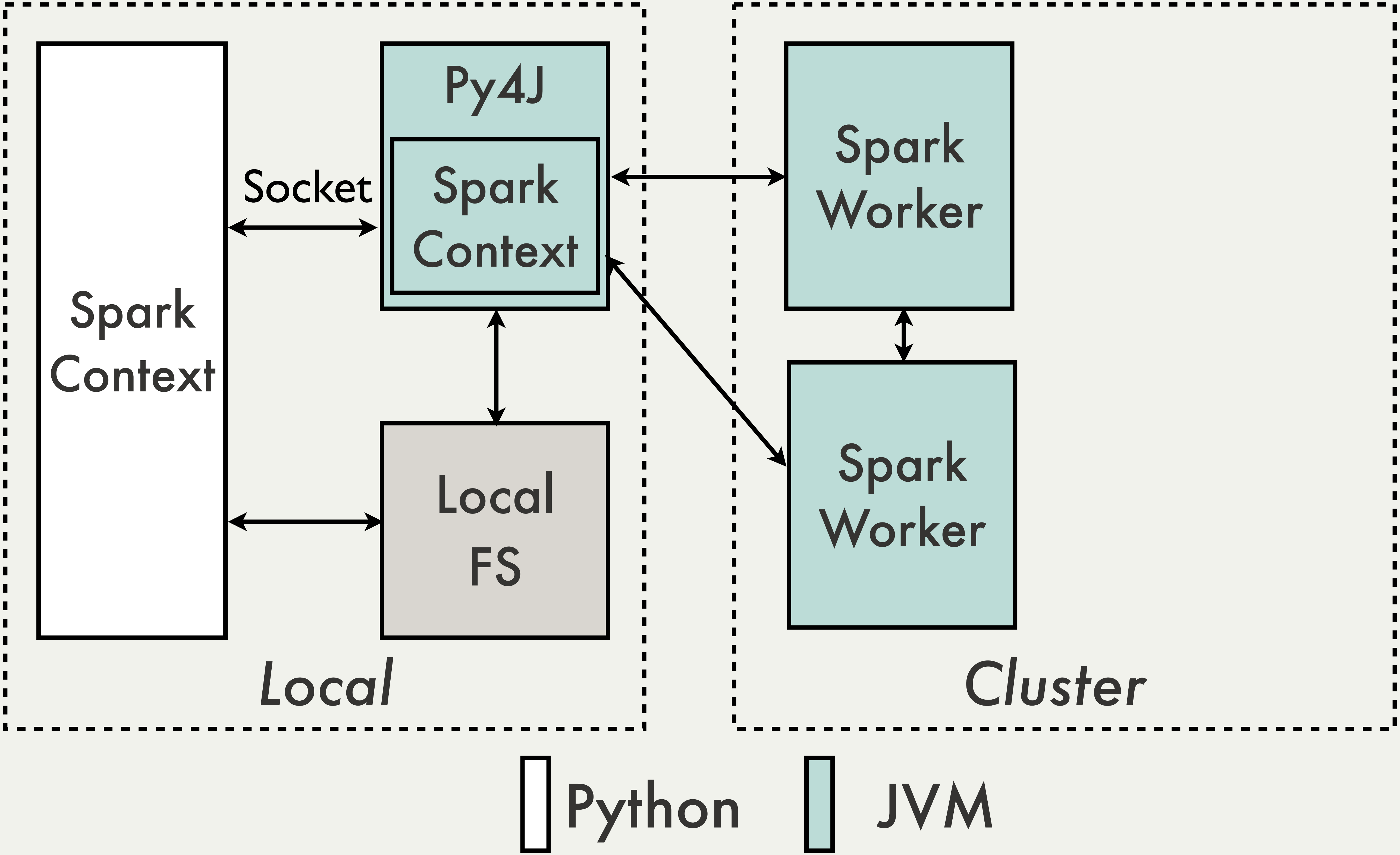
Data Flow


Spark Context
Socket
Py4J
Spark Context
Local
Cluster
Python
JVM

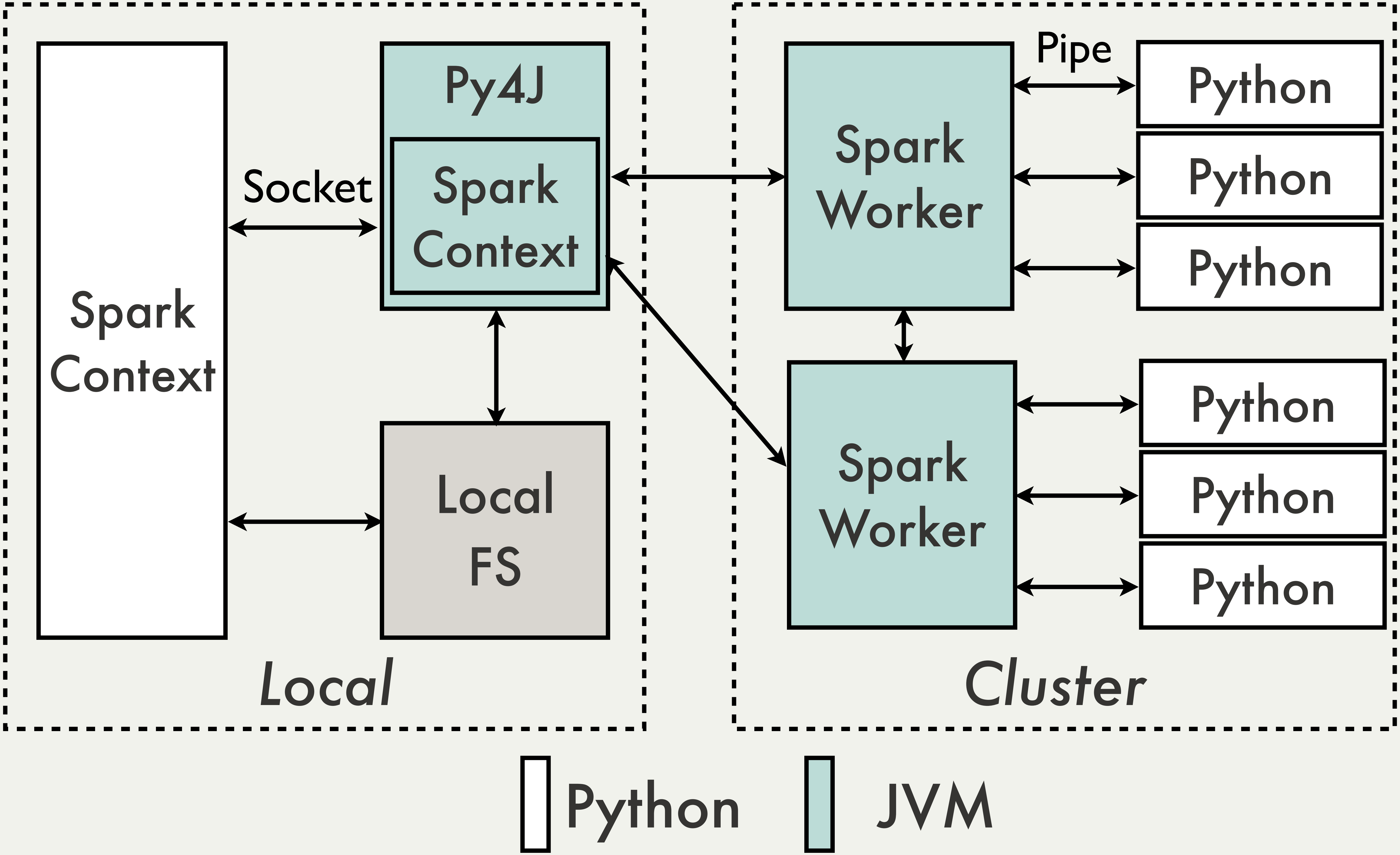
Data Flow


Spark
Context
Socket
Py4J
Spark
Context
Local
FS
Local
Cluster
Python
JVM

Data Flow


Spark Context
Socket
Py4J
Spark Context
Local FS
Local
Spark Worker
Spark Worker
Cluster
Python
JVM

Data Flow

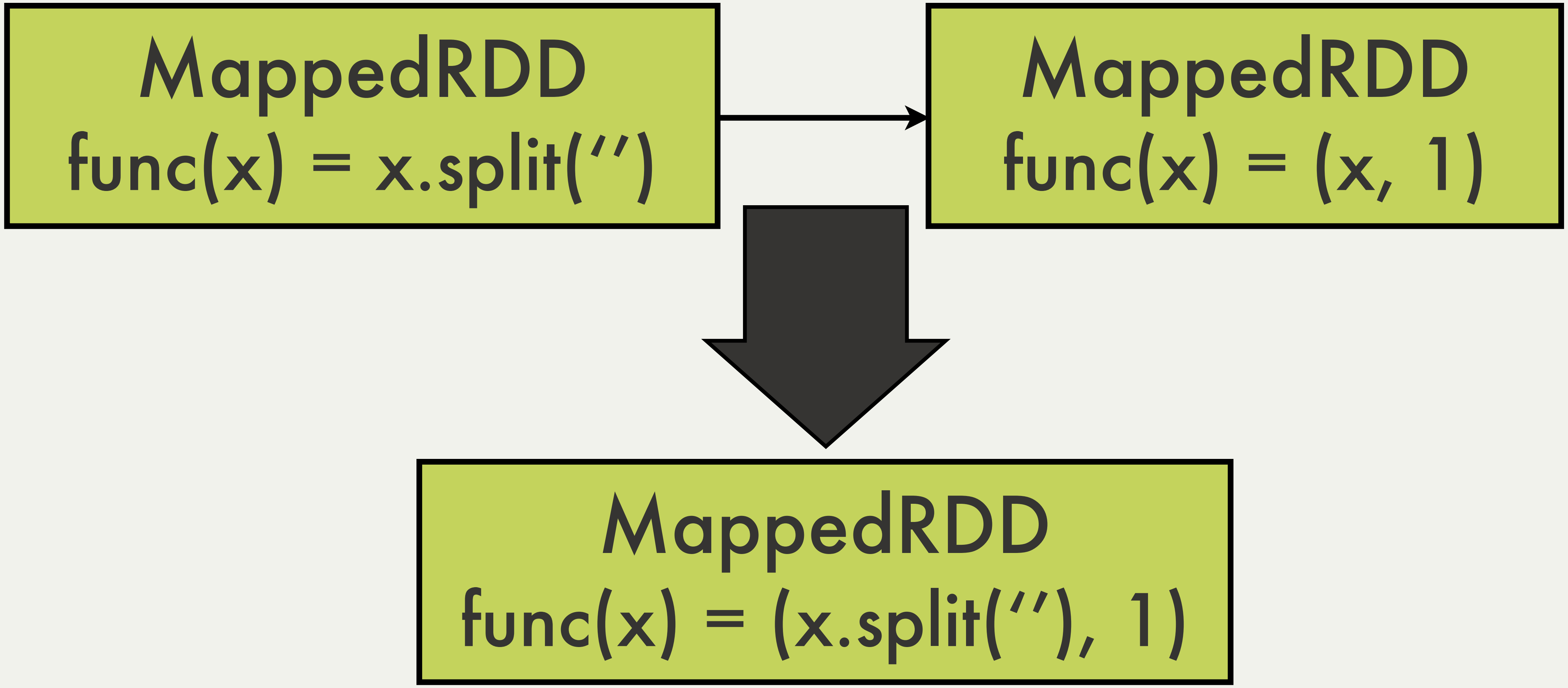

Spark Context
Socket
Py4J
Spark Context
Local FS
Local
Pipe
Spark Worker
Python
Python
Python
Spark Worker
Python
Python
Python
Cluster
Python
JVM

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

Storing *batches* of Python objects in one Scala object reduces overhead

When possible, RDD transformations are pipelined:

```
lines.flatMap(lambda x: x.split(' ')) \
      .map(lambda x: (x, 1))
```



MappedRDD
func(x) = x.split('')

MappedRDD
func(x) = (x, 1)

MappedRDD
func(x) = (x.split(''), 1)

Python functions and closures are serialized using PiCloud's `CloudPickle` module

Roadmap

Available in Spark 0.7

pySpark

Thanks!

Bonus Slides

Pickle is a miniature stack language

```
>>> x = ["Hello", "World!"]
>>> pickletools.dis(cPickle.dumps(x, 2))
    0: \x80 PROTO      2
    2: ]    EMPTY_LIST
    3: q    BINPUT     1
    5: (    MARK
    6: U        SHORT_BINSTRING 'Hello'
   13: q        BINPUT     2
   15: U        SHORT_BINSTRING 'World!'
   23: q        BINPUT     3
   25: e        APPENDS    (MARK at 5)
   26: .    STOP
highest protocol among opcodes = 2
```

You can do crazy stuff, like converting a collection of pickled objects into a pickled collection.

https://gist.github.com/JoshRosen/3384191

Bulk depickling can be faster *even if it involves Pickle opcode manipulation:*

```
10000 integers:
Bulk depickle (chunk size = 2): 0.266709804535
Bulk depickle (chunk size = 10): 0.0797798633575
Bulk depickle (chunk size = 100): 0.0388460159302
Bulk depickle (chunk size = 1000): 0.0333180427551
Individual depickle: 0.0540158748627

10000 dicts (dict([ (str(n), n) for n in range(100) ])):
Bulk depickle (chunk size = 2): 2.70617198944
Bulk depickle (chunk size = 10): 2.30310201645
Bulk depickle (chunk size = 100): 2.22087192535
Bulk depickle (chunk size = 1000): 2.22118020058
Individual depickle: 2.44124102592
```

https://gist.github.com/JoshRosen/3401373