Cloud Computing and Big Data

Apache Spark and More

Oxford University
Software Engineering Programme

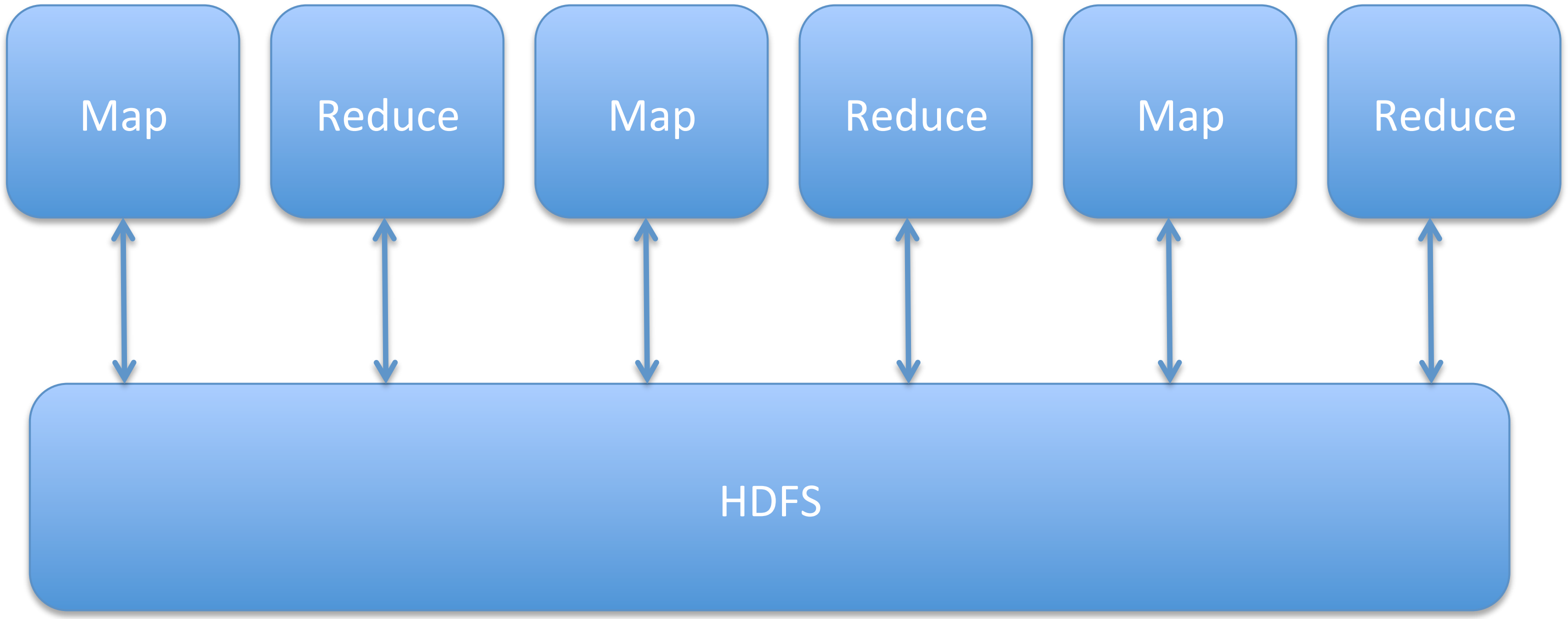
Nov 2015

Contents

Issues with Hadoop

- Hadoop is fundamentally all about Map Reduce
  - Though v2 did allow for other approaches
- Based on cheap commodity hardware
- But….
  - Not based on cheap commodity hardware with lots of memory!

Hadoop Model



Hadoop and Disk

- Hadoop does everything via replicated disk images
- Intermediate results are stored on disk
  - Slow for many operations
  - Including Machine Learning
  - No support for interactive processing

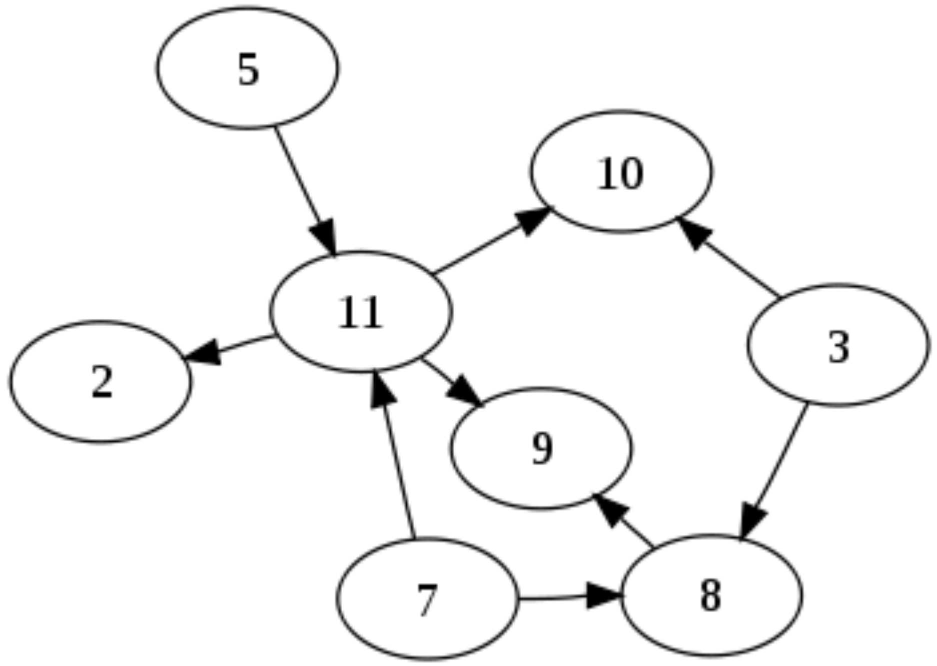
Improved Approach

- A new model based on memory
  - Based on Directed Acyclic Graphs
  - And partitions
- What about reliability?

DAG

Directed Acyclic Graph

No Loops!



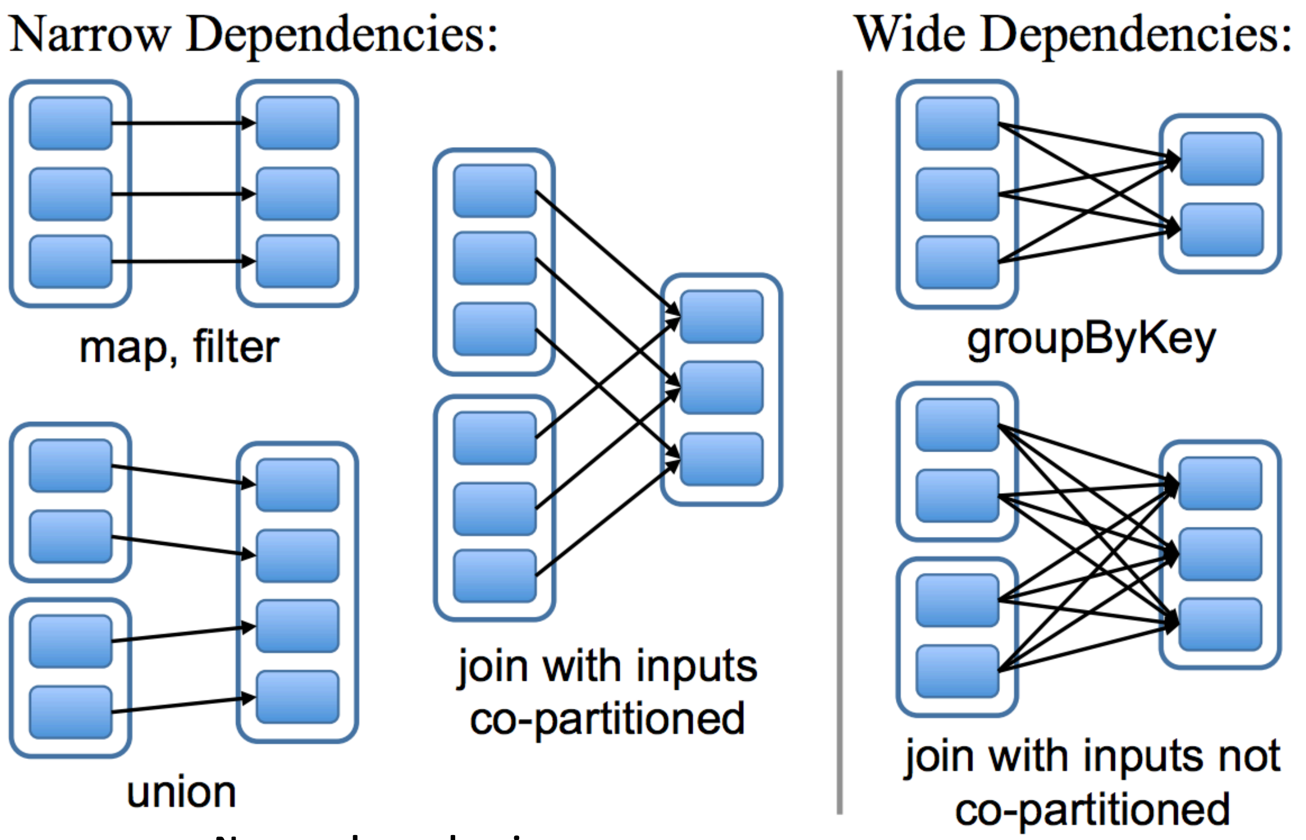
Apache Spark

- Started in 2009 at UC Berkeley
- Donated to Apache in 2013
- Written on top of JVM mainly in Scala
- 10x-100x faster than Hadoop
- Supports coding in:
  - Scala
  - Java
  - Python
  - R
- Supports an interactive shell
- More details in this paper:
  - http://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf

Resilient Distributed Datasets

- A logical collection of data
  - Partitioned across multiple machines
- Logs the lineage of the current data
  - If there is a failure, recreate the data
  - Solves the reliability problem
- Developers can specify the *persistence* and *partitioning* of RDDs

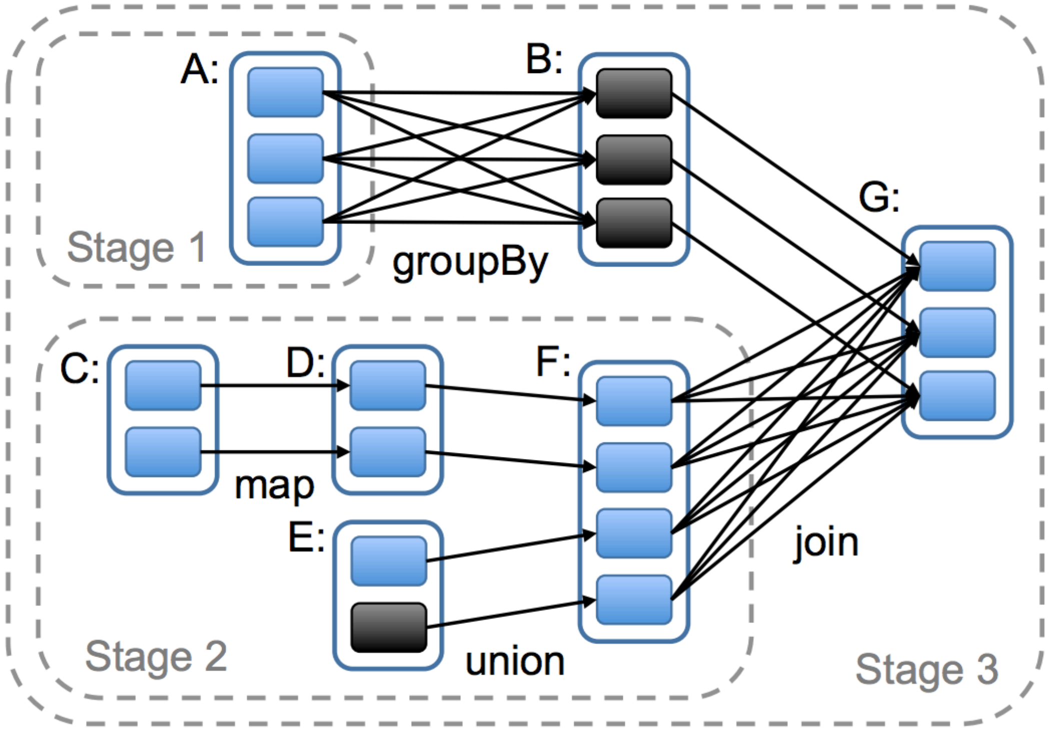
Narrow and Wide dependencies


Narrow Dependencies:
Wide Dependencies:
map, filter
union
join with inputs co-partitioned
groupByKey
join with inputs not co-partitioned

Narrow dependencies:
Each partition of the parent is used by one child partition
Wide Dependencies:
multiple child dependencies depend upon it

Source: http://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf

How Spark computes jobs



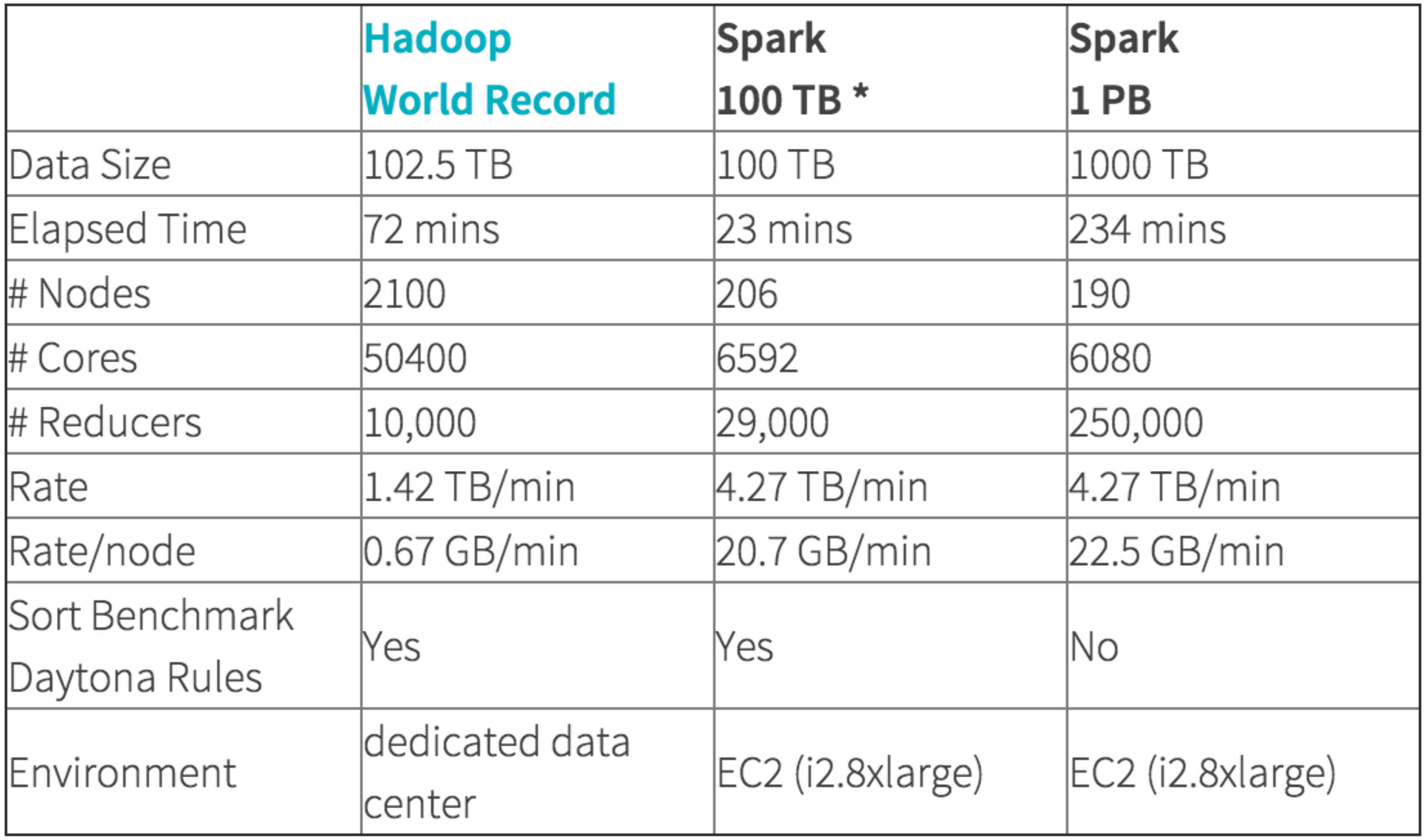
Boxes with solid outlines are **RDDs.**
Partitions are shaded rectangles, in black if they are **already in memory**.

To run an action on RDD G, build **stages** at wide dependencies and **pipeline** narrow transformations inside each stage.

In this case, stage 1's output RDD is already in RAM, so we run stage 2 and then 3.

Source: http://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf

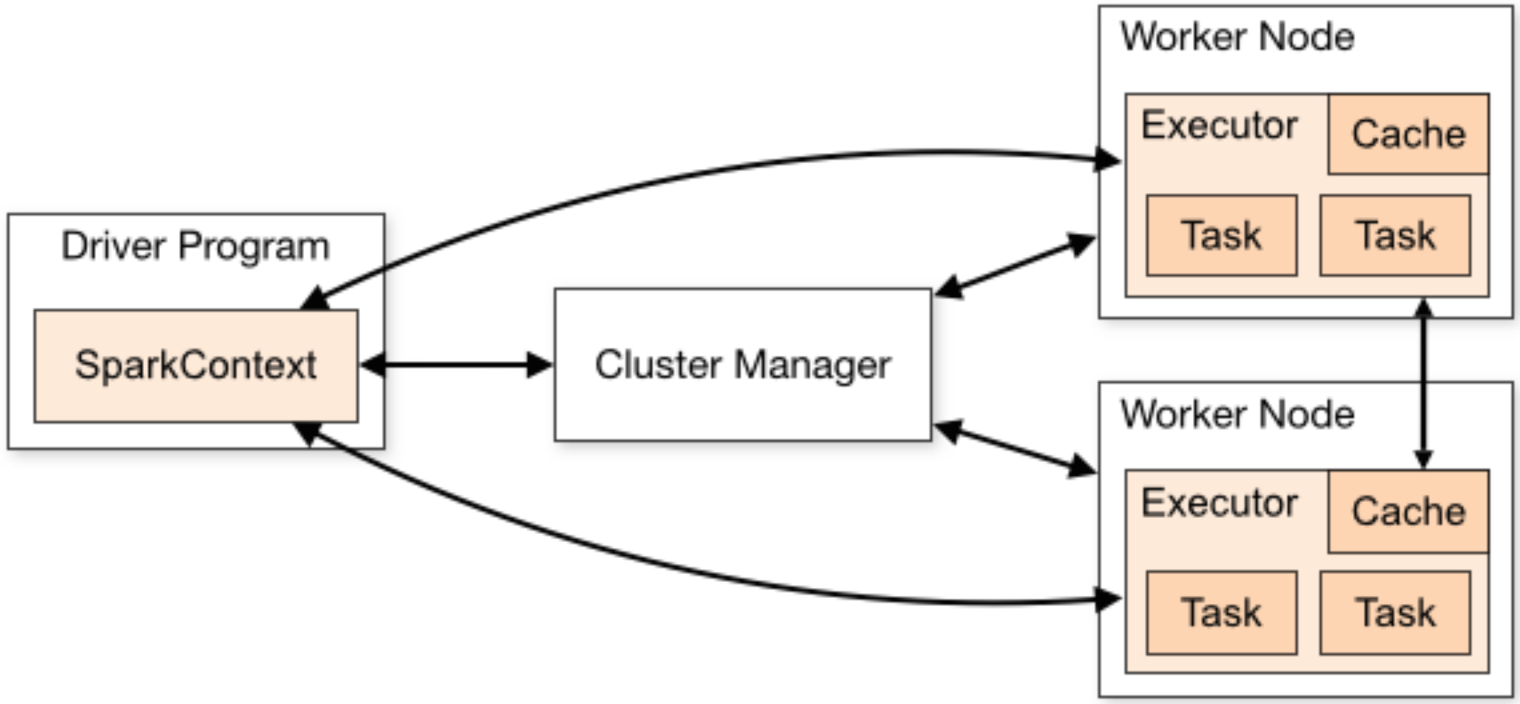
Hadoop vs Spark sorting



	Hadoop World Record	Spark 100 TB *	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400	6592	6080
# Reducers	10,000	29,000	250,000
Rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min
Sort Benchmark Daytona Rules	Yes	Yes	No
Environment	dedicated data center	EC2 (i2.8xlarge)	EC2 (i2.8xlarge)

* not an official sort benchmark record

Apache Spark cluster model



Apache Spark RDD objects

- Typical operations include
  - map: apply a function to each line/element
  - flatMap: can return a sequence not just an element
  - filter: return element if func(element) is true
  - reduceByKey: reduces a set of [K,V] key/value pairs
  - reduce: apply a reducer function
  - collect: get all the results back to the master (driver) server in the cluster
  - foreach: apply a function across each element
- Operations on RDDs will happen across machines
  - Be careful!

Serialization

Storage Level	**Meaning**
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
MEMORY_ONLY_SER	Store RDD as *serialized* Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.
MEMORY_AND_DISK_SER	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.	Same as the levels above, but replicate each partition on two cluster nodes.
OFF_HEAP (experimental)	Store RDD in serialized format in Tachyon. Compared to MEMORY_ONLY_SER, OFF_HEAP reduces garbage collection overhead and allows executors to be smaller and to share a pool of memory, making it attractive in environments with large heaps or multiple concurrent applications. Furthermore, as the RDDs reside in Tachyon, the crash of an executor does not lead to losing the in-memory cache. In this mode, the memory in Tachyon is discardable. Thus, Tachyon does not attempt to reconstruct a block that it evicts from memory. If you plan to use Tachyon as the off heap store, Spark is compatible with Tachyon out-of-the-box. Please refer to this page for the suggested version pairings.

Lambda syntax

- Lambda’s are unnamed functions
  - From Alonzo Church’s 1930s work on the Lambda Calculus
  - Recently added to Java

Lambda syntax in Python

- Simply:

```
lambda x: x.split()
lambda x,y: x+y
```

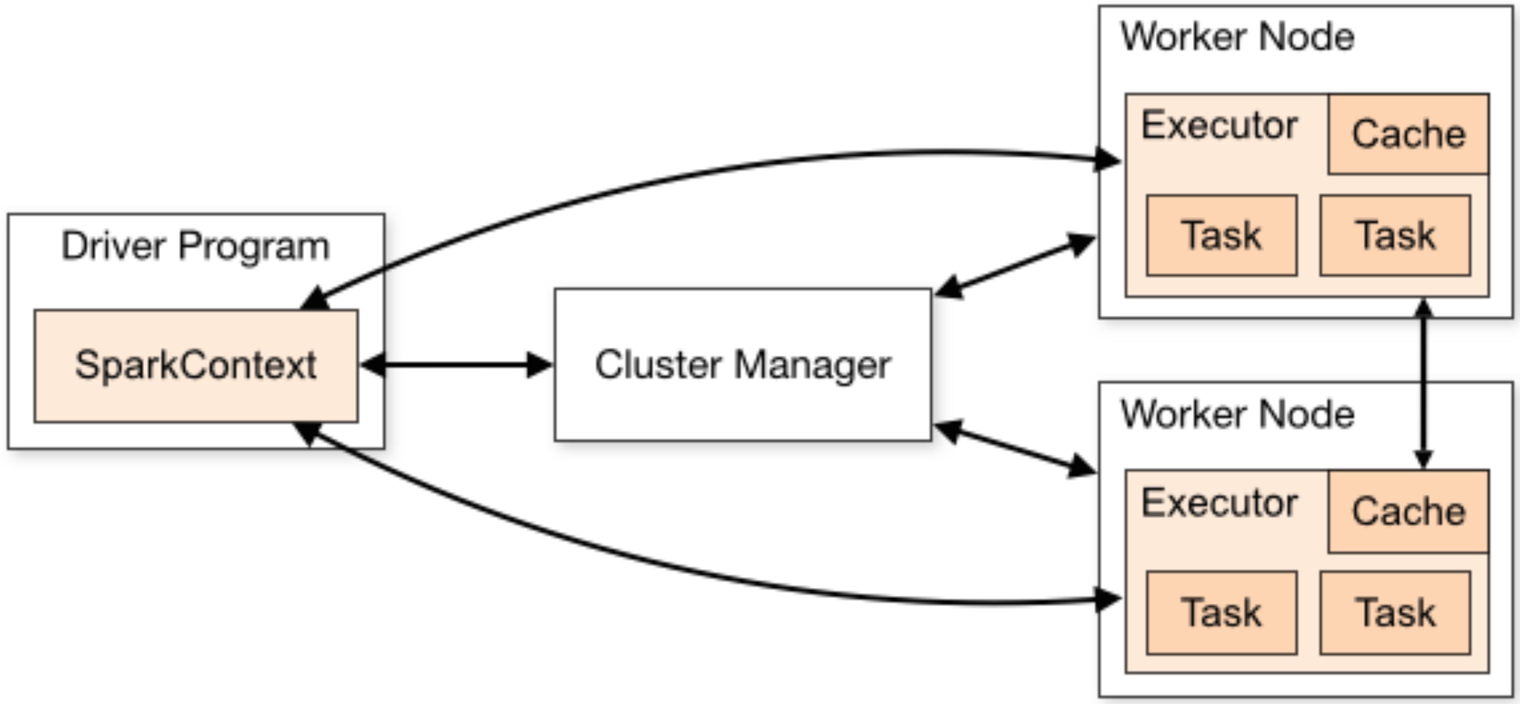
Example

```
sc = SparkContext()

books = sc.textFile("books/*")
split = books.flatMap(lambda line: line.split())
numbered = split.map(lambda word: (word, 1))
wordcount = numbered.reduceByKey(lambda a,b: a+b)

for k,v in wordcount.collect():
  print k,v

sc.stop()
```

What doesn’t work in a cluster

```
counter = 0
rdd = sc.parallelize(data)

# Wrong: Don't do this!!
rdd.foreach(lambda x: counter += x)

print("Counter value: " + counter)
```

Apache Spark cluster model



How to count across a cluster?

- Accumulators

```
acc = sc.accumulator()
rdd = sc.parallelize(data)
rdd.foreach(lambda x: acc.add(x))
```

What also doesn't work

- rdd.foreach(println)
- Of course this *will* work when you test in local mode

Spark packages

- A wide set of plugins
  - Currently 148 community donated plugins
- Data connectors
  - Cassandra, Couchbase, Mongo, CSV, etc
- Machine Learning, Neural networks
- Streaming
- etc

Using Spark Packages

Automatic download from the web:

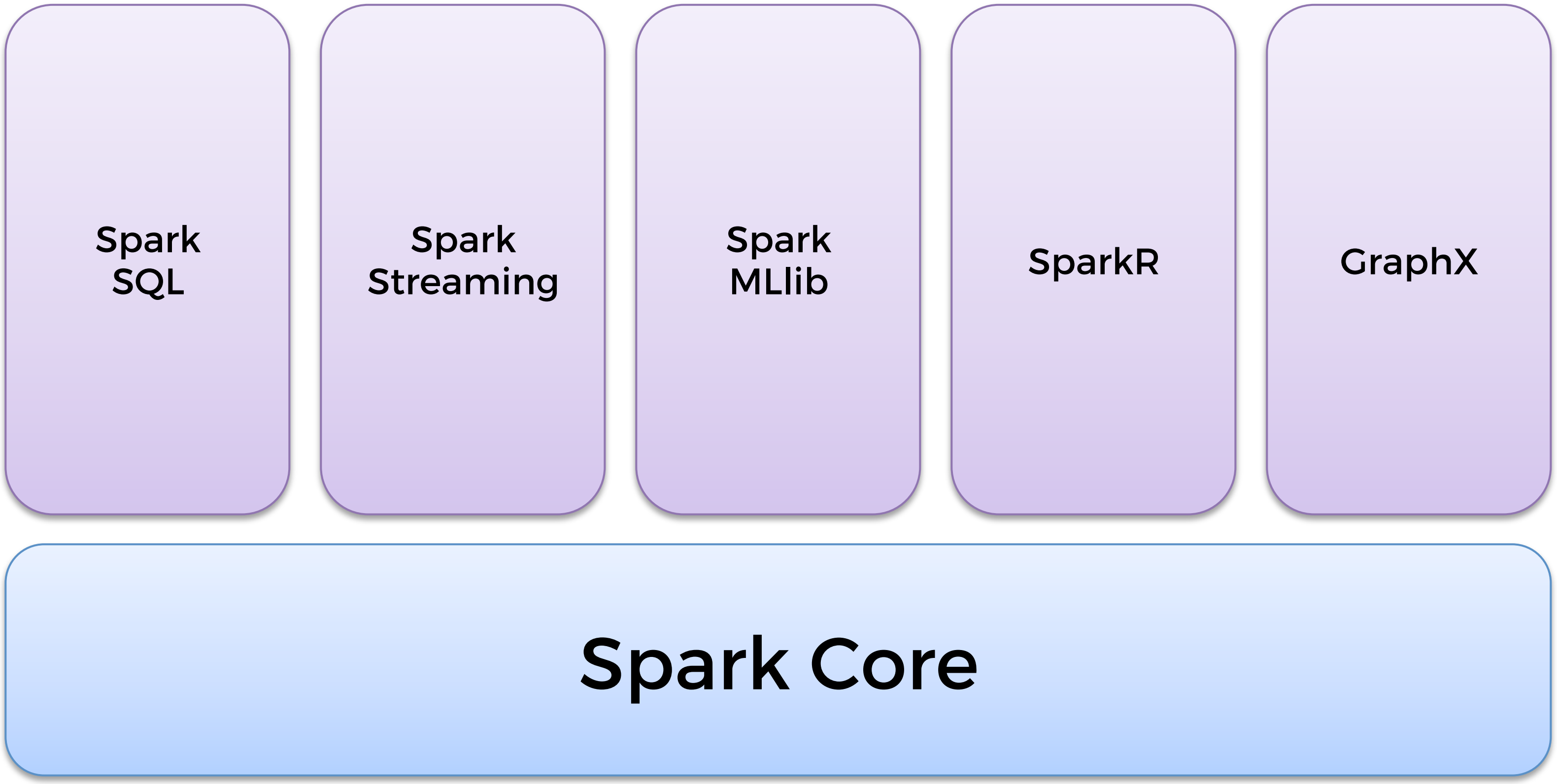
```
bin/spark-shell
    --packages com.databricks:spark-csv_2.11:1.2.0
```

Time for a lab!

Locality

- Spark understands the locality of data:
  - Already in memory
  - HDFS location
  - Cassandra location

Spark Extras



Spark Extras

- Spark SQL
  - Like Apache Hive – use SQL in Spark
- Spark Streaming
  - Realtime analysis in Spark
- Spark MLLib
  - Like Mahout – Machine learning in Spark
- GraphX
  - Graph processing in Spark
- SparkR
  - R statistical analysis on Spark

SparkSQL

- Integrates into existing Spark programs
  - Mixes SQL with Python, Scala or Java
- Integrates data from CSV, Avro, Parquet, JDBC, ODBC, JSON, etc
  - Including joins across them
- Fully supports Apache Hive
  - *If you build it with Hive support*
- Fits into the resilient scalable model of Spark

Spark SQL example

```
from pyspark.sql import SQLContext, Row
sqlContext = SQLContext(sc)

lines = sc.textFile("examples/src/main/resources/people.txt")
parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))

schemaPeople = sqlContext.createDataFrame(people)
schemaPeople.registerTempTable("people")

teenagers = sqlContext.sql("SELECT name FROM people WHERE age >= 13 
AND age <= 19")

teenNames = teenagers.map(lambda p: "Name: " + p.name)
for teenName in teenNames.collect():
  print(teenName)
```

Spark MLlib

- Simple stats and correlation testing
- Classification and regression
- Collaborative Filtering
  - Alternating Least Squares
- Clustering
  - k-means, etc
- Frequent Pattern Mining
- Plus more

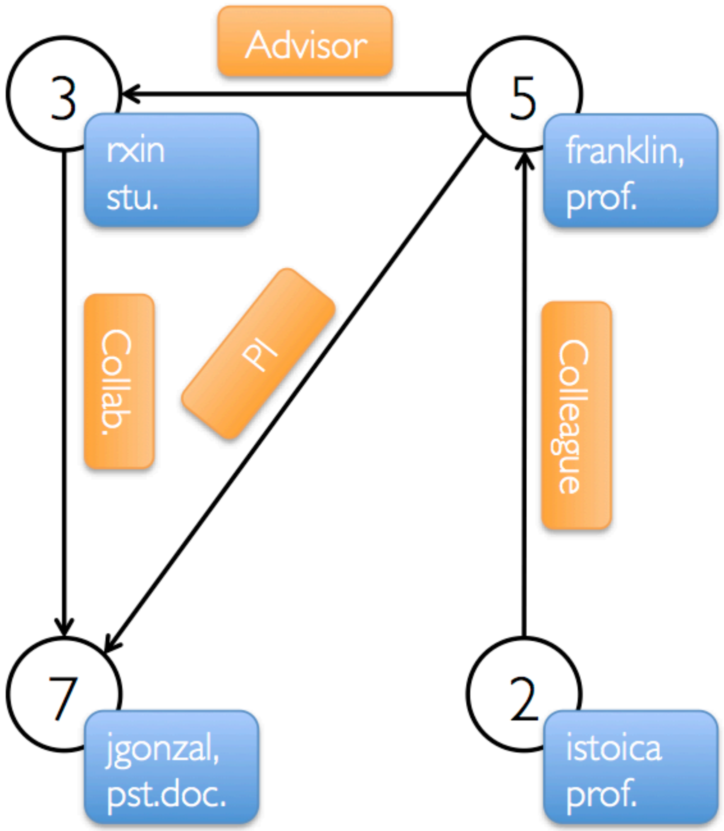
MLlib example

```
from pyspark.mllib.fpm import FPGrowth

data = sc.textFile("data/mllib/sample_fpgrowth.txt")

transactions = data.map(lambda line: line.strip().split(' '))

model = FPGrowth.train(transactions, minSupport=0.2,
    numPartitions=10)

result = model.freqItemsets().collect()
for fi in result:
    print(fi)
```

GraphX

Property Graph

•



Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

R

- R is an open source system for statistics and graphics
  - Based on the S language from AT&T Bell Labs
- Supports a wide variety of statistical techniques and graphing tools
- An extensible set of packages that provide extra functions via CRAN
  - The Comprehensive R Archive Network

SparkR

- A lightweight approach to use Spark from within R
- Also works with MLlib for machine learning
- Allows complex statistical analysis to be done on a Spark cluster

Questions?