Apache Spark

CS240A

T Yang

Some of them are based on P. Wendell's Spark slides

Parallel Processing using Spark+Hadoop

- **Hadoop: Distributed file system that connects machines.**
- **Mapreduce: parallel programming style built on a Hadoop cluster**
- **Spark: Berkeley design of Mapreduce programming**
- **Given a file treated as a big list**
  - A file may be divided into multiple parts (splits).
- **Each record (line) is processed by a Map function,**
  - produces a set of intermediate key/value pairs.
- **Reduce: combine a set of values for the same key**

>>> words = 'The quick brown fox jumps over the lazy dog'.split()

Python Examples for List Processing

```
>>> lst = [3, 1, 4, 1, 5]
>>> lst.append(2)
>>> len(lst)
5
>>> lst.sort()
>>> lst.insert(4,"Hello")
>>> [1]+ [2]      → [1,2]
>>> lst[0] ->3
```

```
for i in [5, 4, 3, 2, 1] :
    print i
```

```
Python tuples

>>> num=(1, 2, 3, 4)
>>> num +(5)   →
   (1,2,3,4, 5)
```

```
>>>M = [x for x in S if x % 2 == 0]
>>>  S = [x**2 for x in range(10)]
[0,1,4,9,16,…,81]
```

```
>>> words ='hello  lazy dog'.split()
→ ['hello', 'lazy', 'dog']
>>> stuff = [(w.upper(),  len(w)] for w in words]
→ [ ('HELLO', 5) ('LAZY', 4) , ('DOG', 4)]
```

>>>numset=set([1, 2, 3, 2])
Duplicated entries are deleted

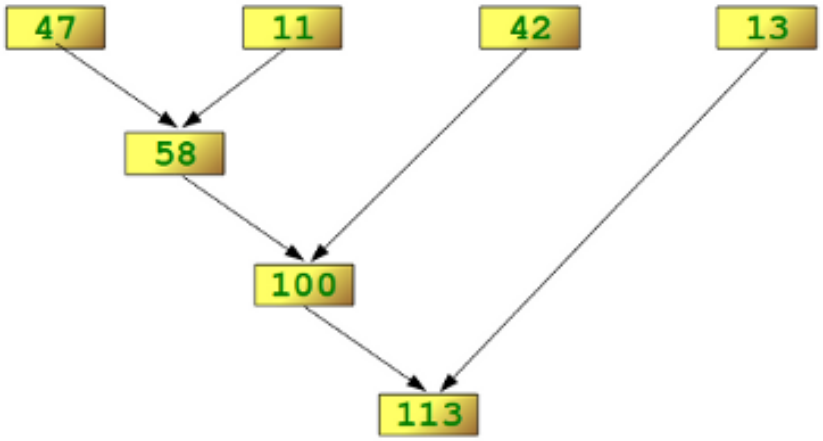
>>>numset=frozenset([1, 2,3])
Such a set cannot be modified

UCSB

Python map/reduce

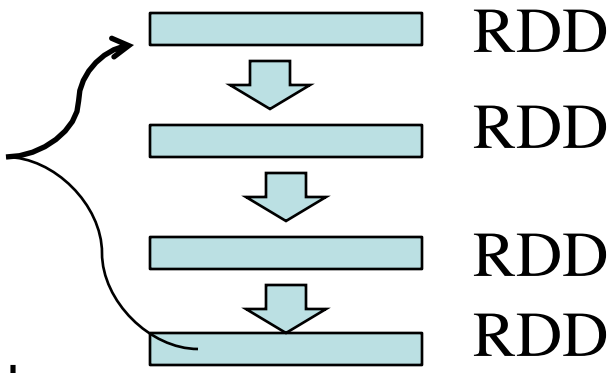
```
a = [1, 2, 3]
b = [4, 5, 6, 7]
c = [8, 9, 1, 2, 3]
f= lambda x:   len(x)
L = map(f, [a, b, c])
[3, 4, 5]
```

```
g=lambda x,y:    x+y
reduce(g, [47,11,42,13])
113
```



Mapreduce programming with SPAK: key concept

Write programs in terms of **operations** on implicitly distributed **datasets (RDD)**



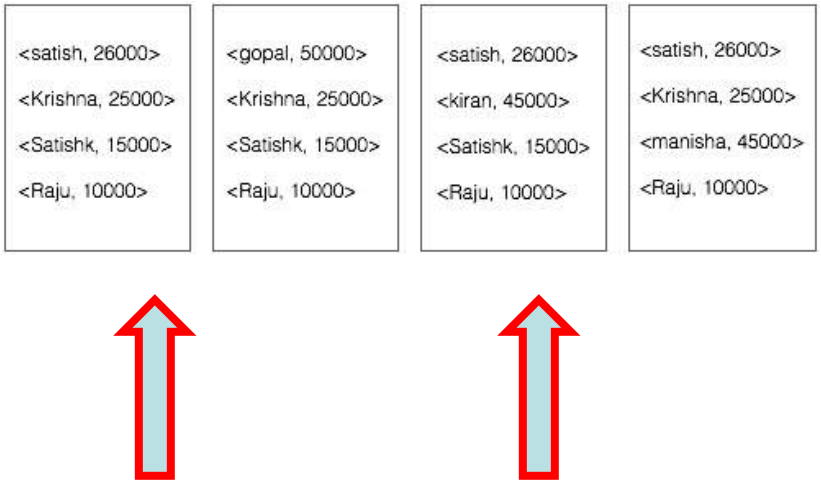
RDD: Resilient Distributed Datasets

- **Like a big list:**
  - Collections of objects spread across a cluster, stored in RAM or on Disk
- **Built through parallel transformations**
- **Automatically rebuilt on failure**

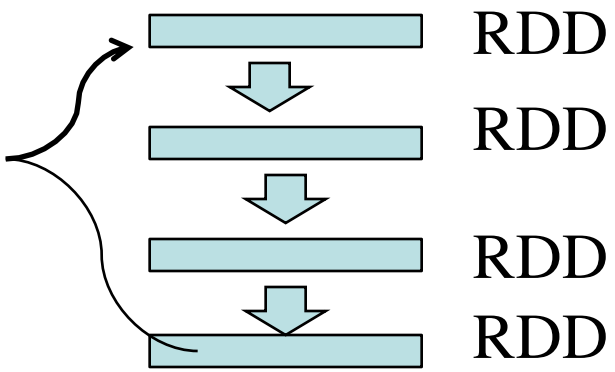
Operations

- **Transformations (e.g. map, filter, groupBy)**
- **Make sure input/output match**

MapReduce vs Spark



Map and reduce tasks operate on key-value pairs



Spark operates on **RDD** with aggressive memory caching

UCSB

Language Support

Python

```
lines = sc.textFile(...)
lines.filter(lambda s: "ERROR" in s).count()
```

Scala

```
val lines = sc.textFile(...)
lines.filter(x => x.contains("ERROR")).count()
```

Java

```
JavaRDD<String> lines = sc.textFile(...);
lines.filter(new Function<String, Boolean>() {
  Boolean call(String s) {
    return s.contains("error");
  }
}).count();
```

Standalone Programs

- Python, Scala, & Java

Interactive Shells

- Python & Scala

Performance

- Java & Scala are faster due to static typing
- …but Python is often fine

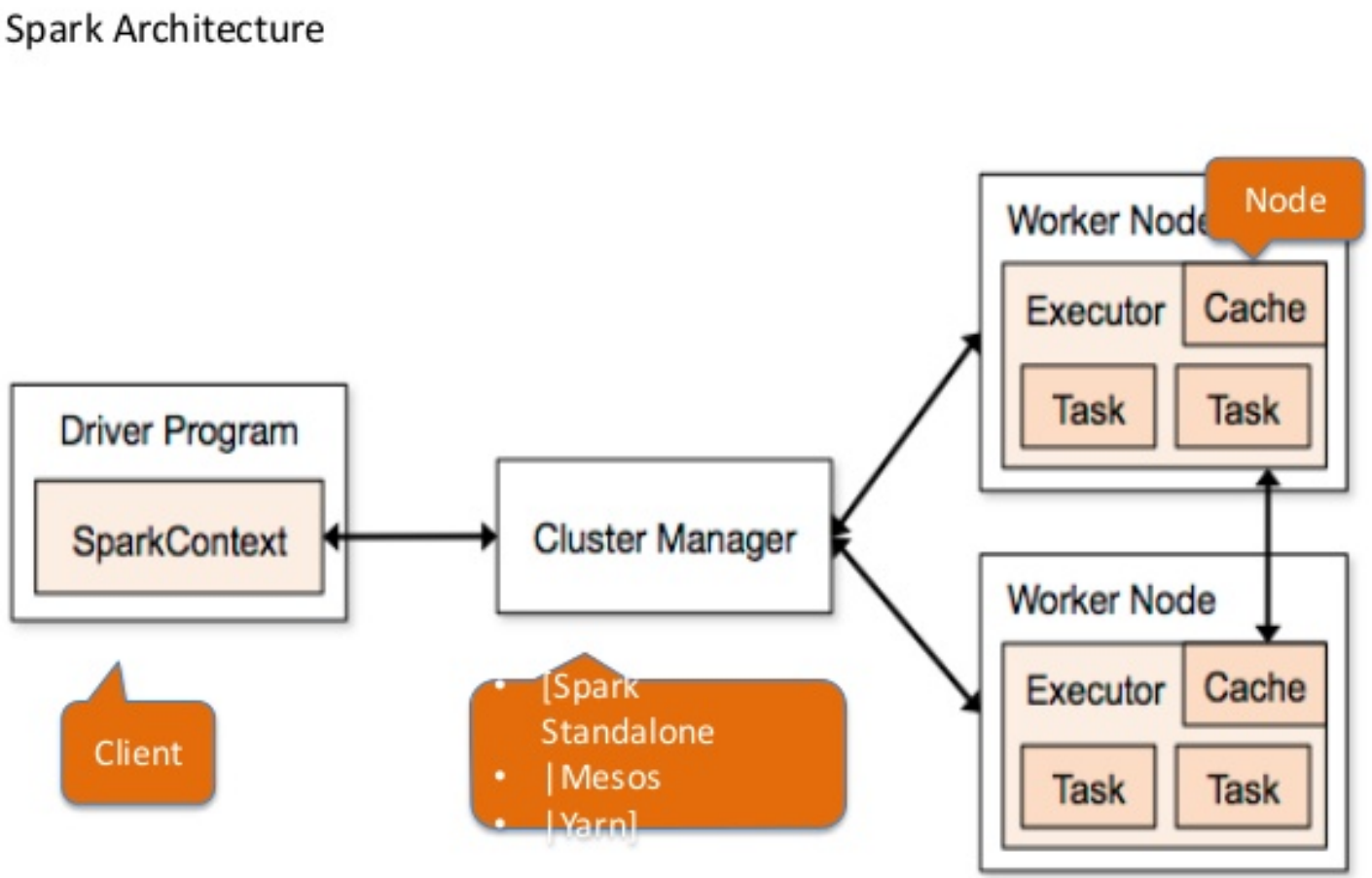
Spark Context and Creating RDDs

```
#Start with sc – SparkContext as
```
Main entry point to Spark functionality

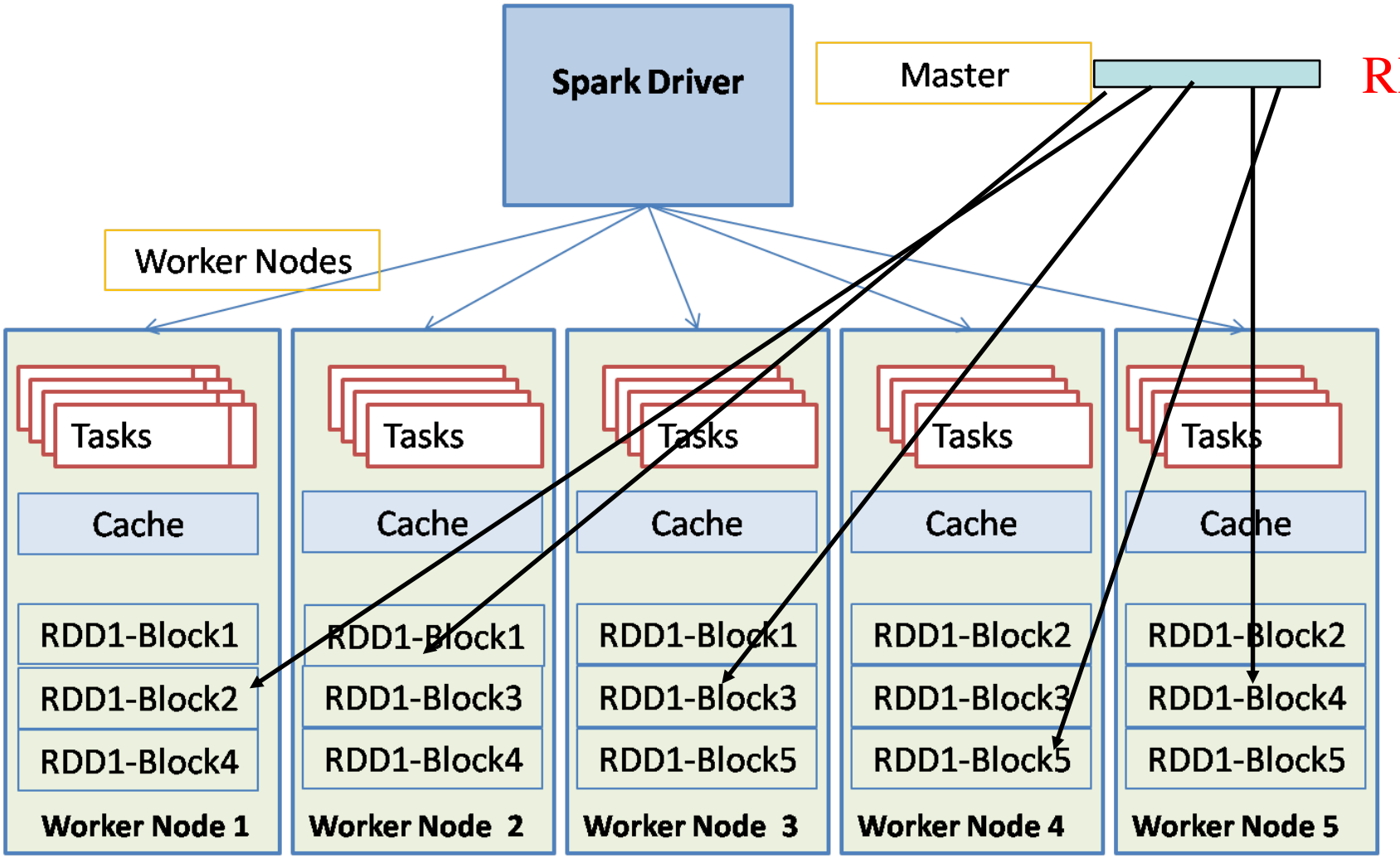
```
#Turn a Python collection into an RDD
>sc.parallelize([1, 2, 3])

# Load text file from local FS, HDFS, or S3
>sc.textFile("file.txt")
>sc.textFile("directory/*.txt")
>sc.textFile("hdfs://namenode:9000/path/file")
```

RDD

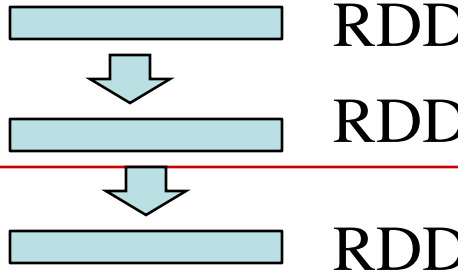
Spark Architecture



Spark Components



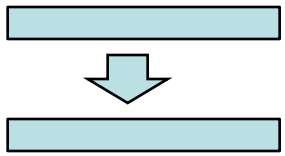
Basic Transformations




```
#read a text file and count number of lines
containing error

lines = sc.textFile("file.log")
lines.filter(lambda s: "ERROR" in s).count()
```

```
> nums = sc.parallelize([1, 2, 3])

# Pass each element through a function
> squares = nums.map(lambda x: x*x)   // {1, 4, 9}

# Keep elements passing a predicate
> even = squares.filter(lambda x: x % 2 == 0) // {4}
```

Basic Actions




```
> nums = sc.parallelize([1, 2, 3])

# Retrieve RDD contents as a local collection
> nums.collect() # => [1, 2, 3]

# Return first K elements
> nums.take(2)   # => [1, 2]

# Count number of elements
> nums.count()   # => 3

# Merge elements with an associative function
> nums.reduce(lambda x, y: x + y)  # => 6

# Write elements to a text file
> nums.saveAsTextFile("hdfs://file.txt")
```

Working with Key-Value Pairs

Spark's "distributed reduce" transformations operate on RDDs of key-value pairs

Python:

```
pair = (a, b)
       pair[0] # => a
                  pair[1] # => b
```

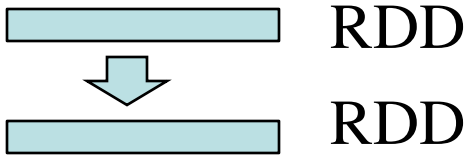
Scala:

```
val pair = (a, b)
                  pair._1 // => a
                  pair._2 // => b
```

Java:

```
Tuple2 pair = new Tuple2(a, b);
                  pair._1 // => a
                  pair._2 // => b
```

Some Key-Value Operations

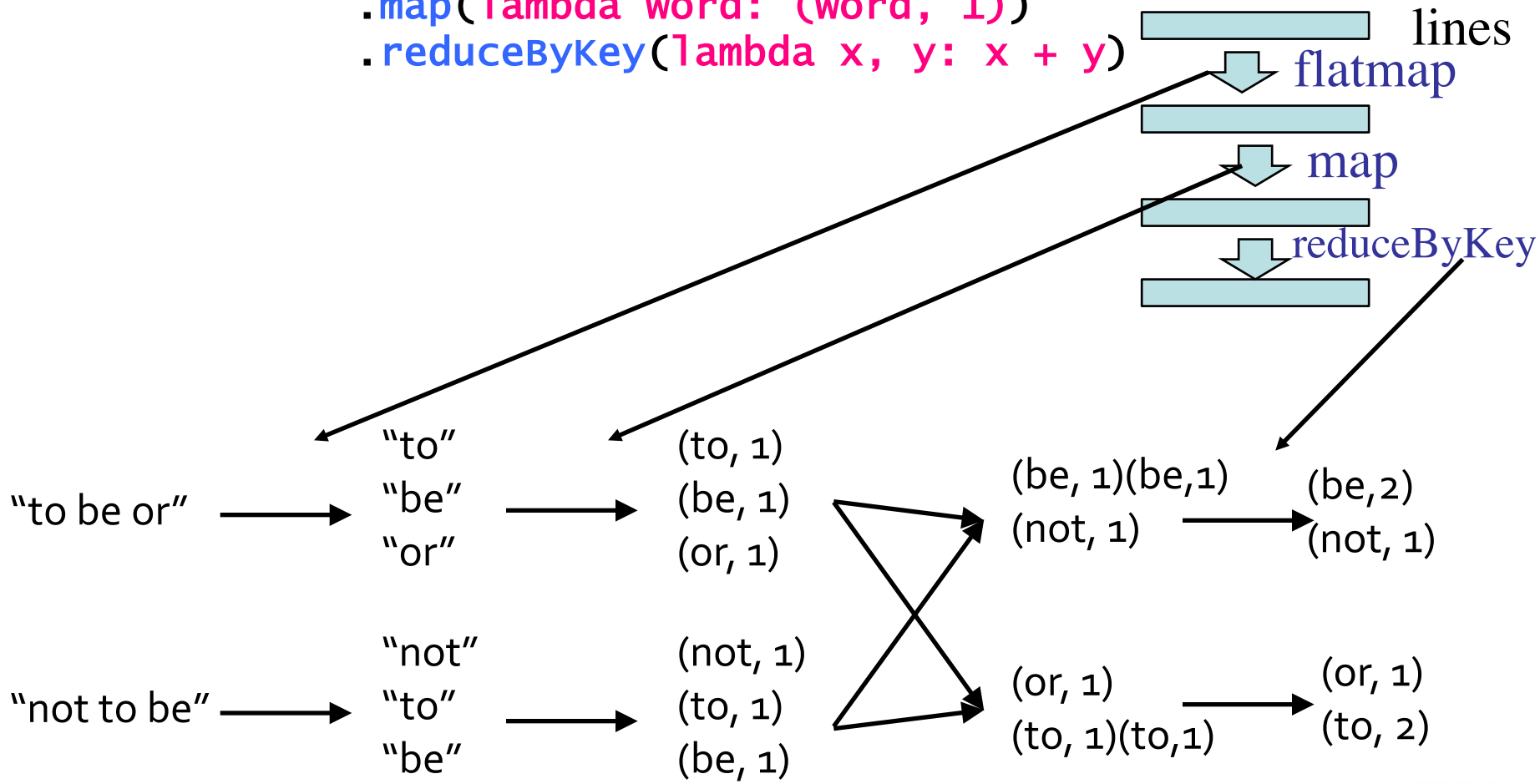



```
> pets = sc.parallelize(
    [("cat", 1), ("dog", 1), ("cat", 2)])
> pets.reduceByKey(lambda x, y: x + y)
                        # => {(cat, 3), (dog, 1)}
> pets.groupByKey() # => {(cat, [1, 2]), (dog, [1])}
> pets.sortByKey()  # => {(cat, 1), (cat, 2), (dog, 1)}
```

`reduceByKey()` also automatically implements combiners on the map side

Example: Word Count

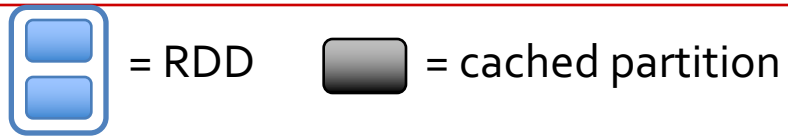
```
> lines = sc.textFile("hamlet.txt")
> counts = lines.flatMap(lambda line: line.split(" "))
                .map(lambda word: (word, 1))
                .reduceByKey(lambda x, y: x + y)
```



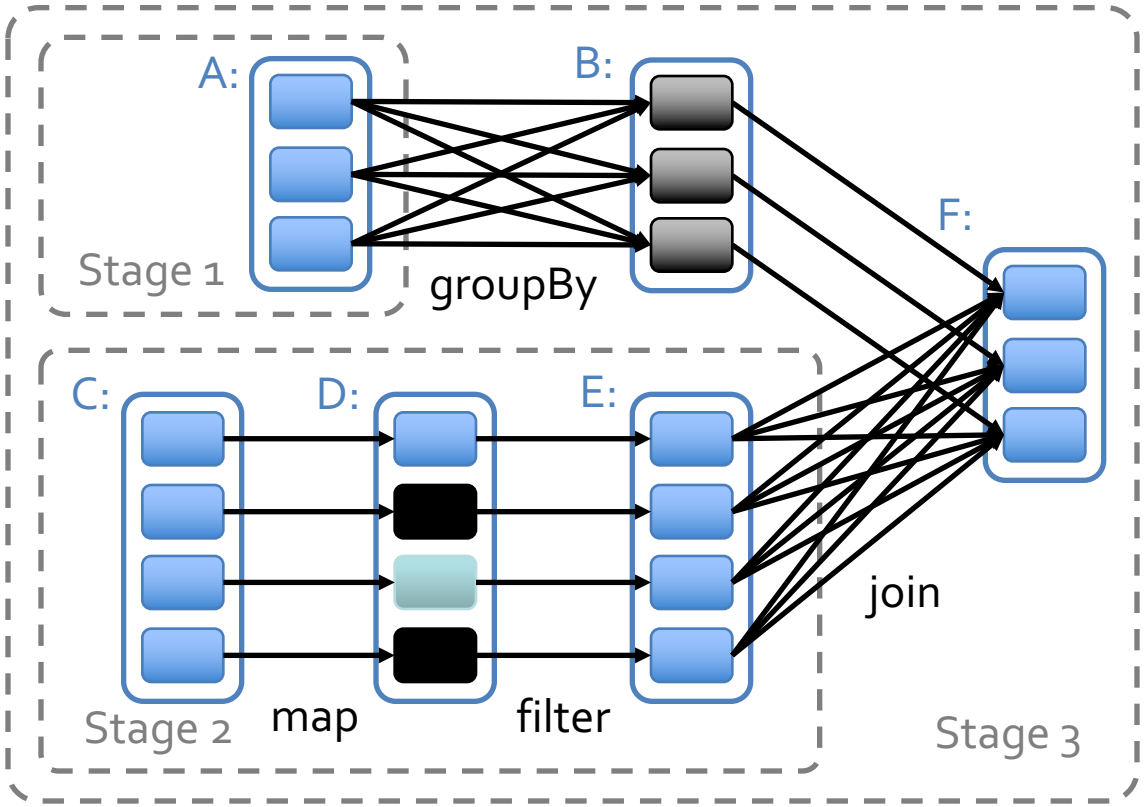
Other Key-Value Operations

```
> visits = sc.parallelize([ (“index.html”, “1.2.3.4”),
                            (“about.html”, “3.4.5.6”),
                            (“index.html”, “1.3.3.1”) ])

> pageNames = sc.parallelize([ (“index.html”, “Home”),
                               (“about.html”, “About”) ])

> visits.join(pageNames)
  # (“index.html”, (“1.2.3.4”, “Home”))
  # (“index.html”, (“1.3.3.1”, “Home”))
  # (“about.html”, (“3.4.5.6”, “About”))

> visits.cogroup(pageNames)
  # (“index.html”, ([“1.2.3.4”, “1.3.3.1”], [“Home”]))
  # (“about.html”, ([“3.4.5.6”], [“About”]))
```

Under The Hood: DAG Scheduler



- General task graphs
- Automatically pipelines functions
- Data locality aware
- Partitioning aware to avoid shuffles



Setting the Level of Parallelism

All the pair RDD operations take an optional second parameter for number of tasks

```
> words.reduceByKey(lambda x, y: x + y, 5)
> words.groupByKey(5)
> visits.join(pageViews, 5)
```

More RDD Operators

- map
- filter
- groupBy
- sort
- union
- join
- leftOuterJoin
- rightOuterJoin
- reduce
- count
- fold
- reduceByKey
- groupByKey
- cogroup
- cross
- zip

sample

take

first

partitionBy

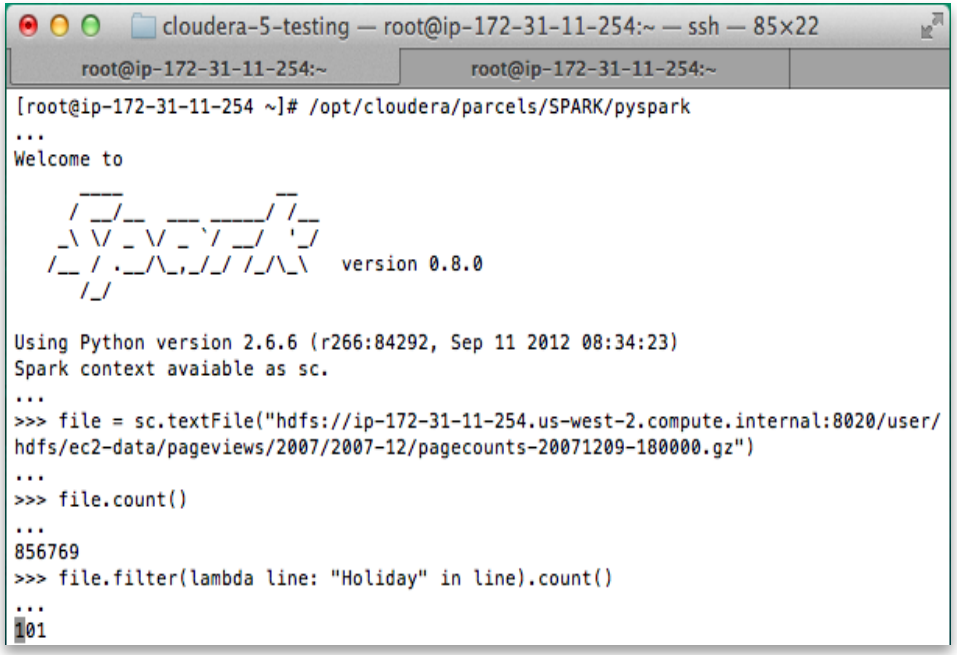
mapWith

pipe

save ...

Interactive Shell

- The Fastest Way to Learn Spark
- Available in Python and Scala
- Runs as an application on an existing Spark Cluster…
- OR Can run locally



UCSB

… or a Standalone Application

```
import sys
from pyspark import SparkContext

if __name__ == "__main__":
    sc = SparkContext( “local”, “WordCount”, sys.argv[0],
None)
    lines = sc.textFile(sys.argv[1])

    counts = lines.flatMap(lambda s: s.split(“ ”)) \
                  .map(lambda word: (word, 1)) \
                  .reduceByKey(lambda x, y: x + y)

    counts.saveAsTextFile(sys.argv[2])
```

Create a SparkContext

Scala

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._

val sc = new SparkContext("url", "name", "sparkHome", Seq("app.jar"))
```

Java

```
import org.apache.

JavaSparkContext sc = new JavaSparkContext(
    "masterUrl", "name", "sparkHome", new String[] {"app.jar"}));
```

Python

```
from pyspark import SparkContext

sc = SparkContext("masterUrl", "name", "sparkHome", ["library.py"]))
```

Cluster URL, or local / local[N]

App name

Spark install path on cluster

List of JARs with app code (to ship)

Administrative GUIs

http://<Standalone Master>:8080 (by default)