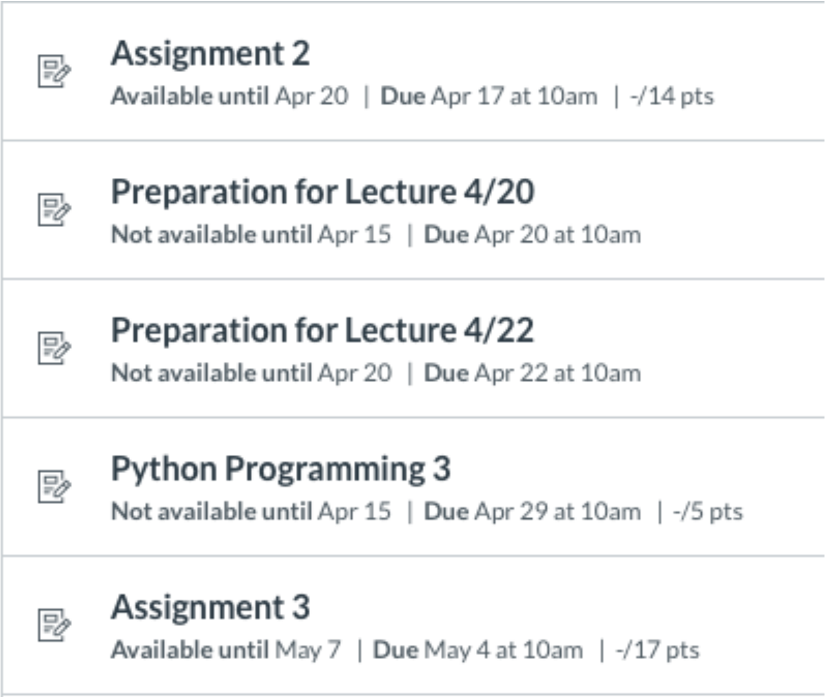
DIT 873 & DAT346 Techniques for Large-Scale Data

Lecture 5

Agenda for today:

- Map/Reduce
- Spark

Upcoming Deadlines



Assignment 2
Available until Apr 20 | **Due** Apr 17 at 10am | -/14 pts

Preparation for Lecture 4/20
Not available until Apr 15 | **Due** Apr 20 at 10am

Preparation for Lecture 4/22
Not available until Apr 20 | **Due** Apr 22 at 10am

Python Programming 3
Not available until Apr 15 | **Due** Apr 29 at 10am | -/5 pts

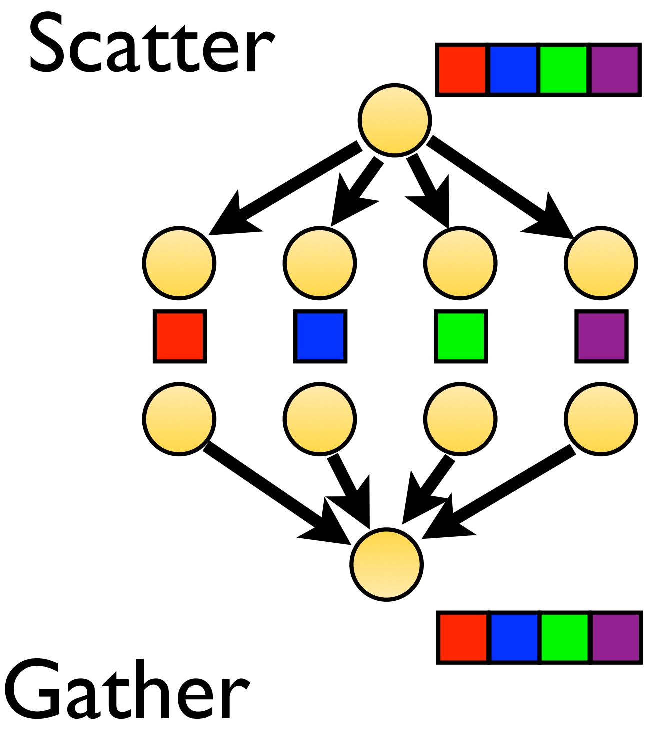
Assignment 3
Available until May 7 | **Due** May 4 at 10am | -/17 pts

Announcements

- Bayes:
  - Only run very short test jobs interactively (e.g. < 8 threads, a few seconds)
  - Make sure your that jobs in the queue are quite granular (e.g., not one giant computation)
  - Monitor your job … if it takes too long, it might be better to kill it

Recap

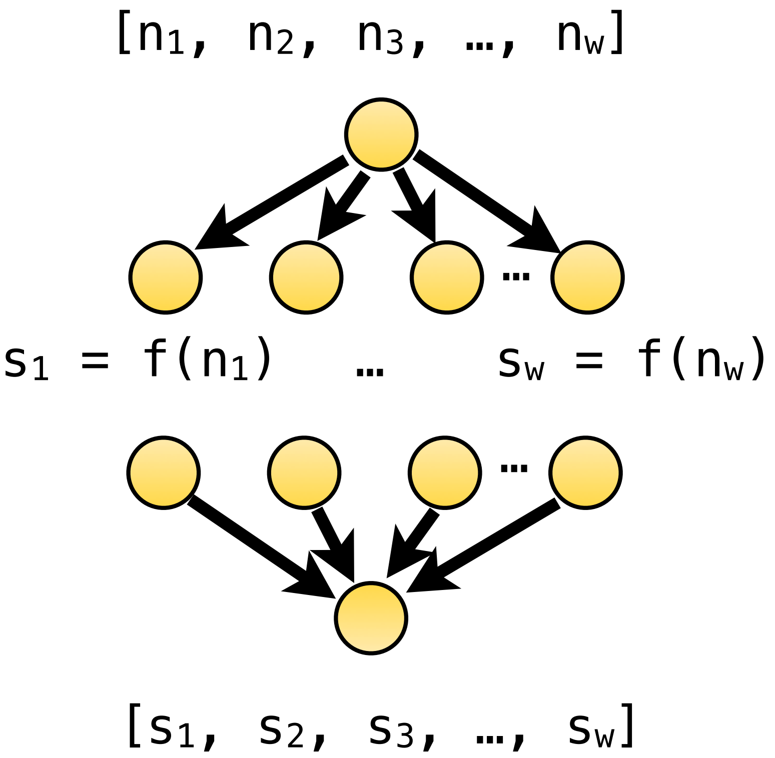
MPI scatter/gather vs. mp.map



Scatter

Gather

```
s = p.map(f, [n] * w)
```

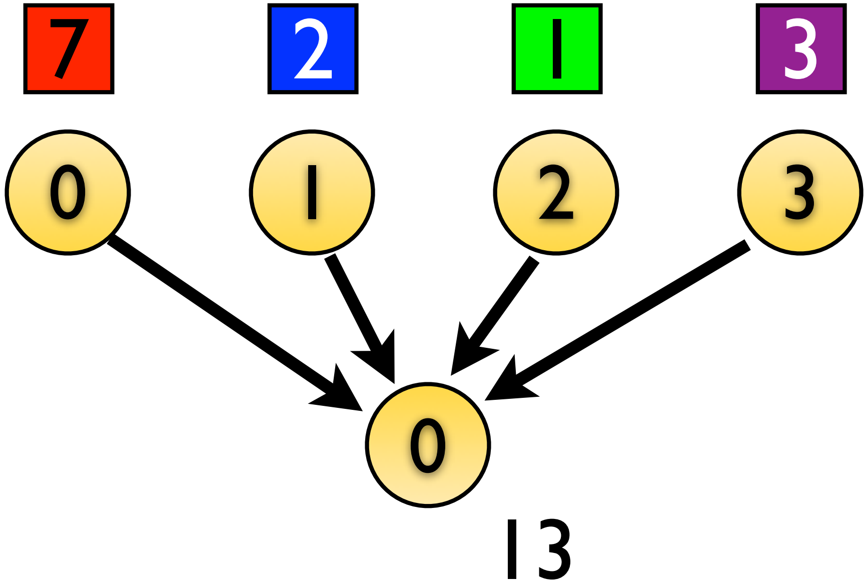


$[n_1, n_2, n_3, \ldots, n_w]$

$s_1 = f(n_1)$ … $s_w = f(n_w)$

$[s_1, s_2, s_3, \ldots, s_w]$

MPI Reduce

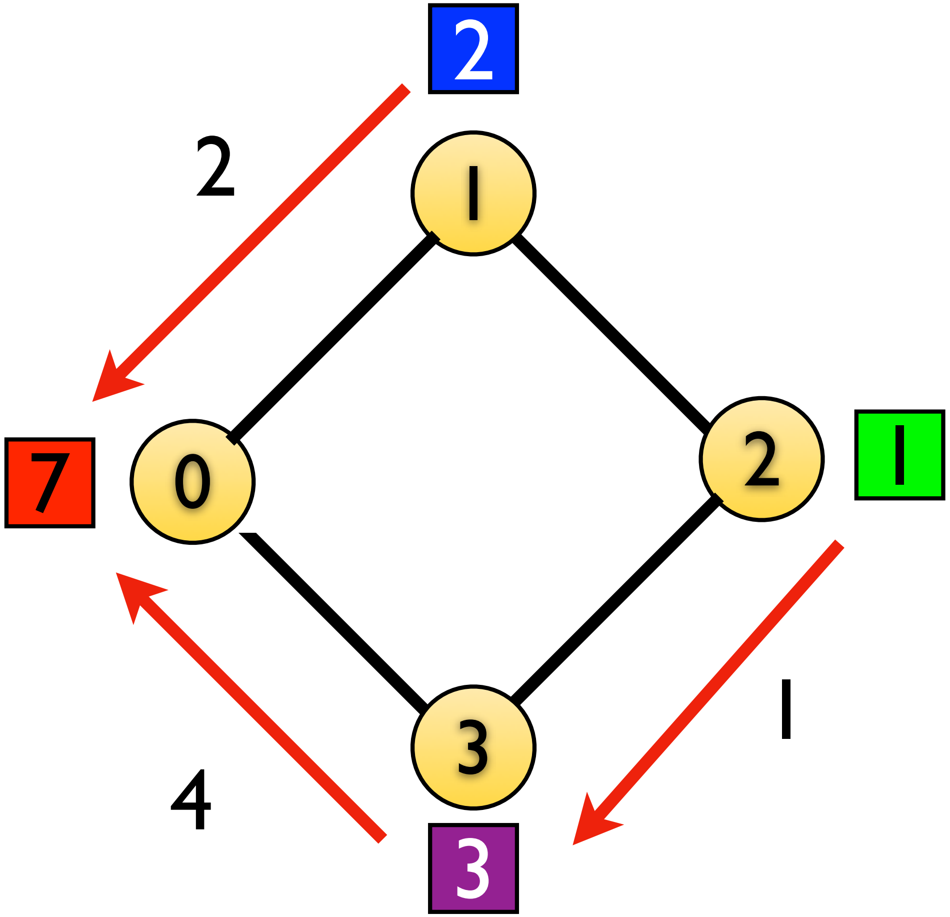


Sum

Reduce supports: Sum, Prod, Max, Min, Argmax, Argmin, Logic operations

Reduce is *not* Gather + Function call

(a) Ring



2+7 = 9
9+4 = 13

3+1 = 4

Sum

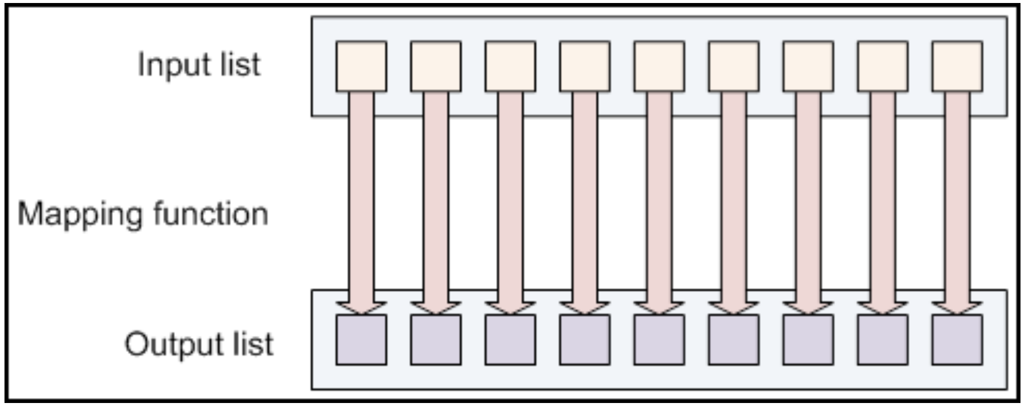
Message sent

Map/reduce assumptions

- Input files are distributed over nodes (implicit)
- All data are (key, value) tuples
- Main parallel operations are:
  - Map:
    (key, value) → $(key_1, value_1)[,(key_2, value_2), \ldots]$
  - Reduce:
    $(key_1, [value_1, value_2, value_3, ...])$ →
    $(key_{new}, value_{new})[,(key_{new2}, value_{new2}), \ldots]$

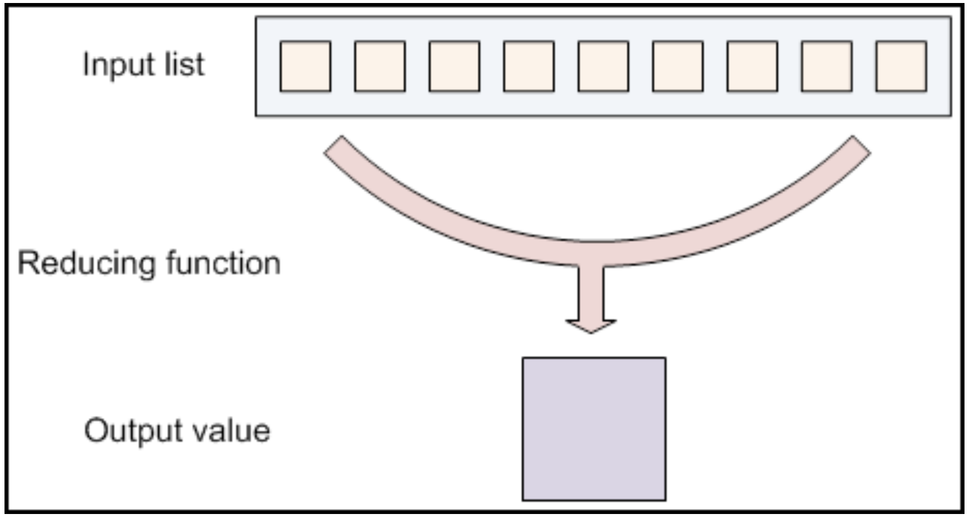
[ ] optional

Map

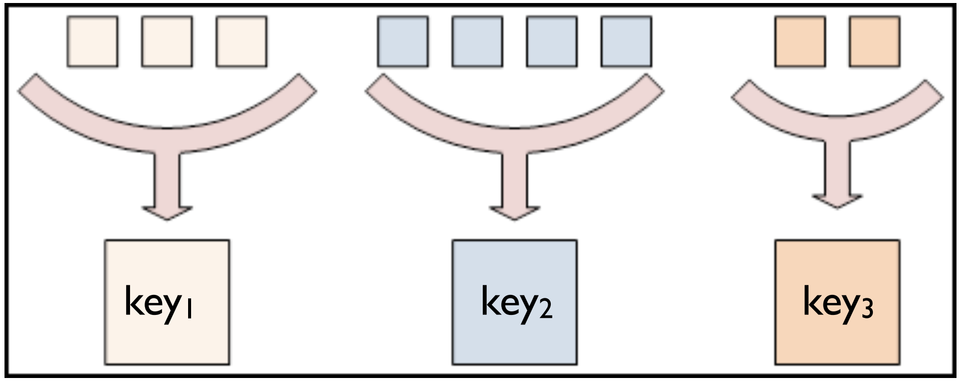


Each map call only needs its input; no assertions about other inputs, execution order, assignment to nodes

Reduce



keys control amount of parallelism



Each reduce call receives all values for its key; no assertions about other keys, execution order, assignment to nodes

n = # lines
k= #nodes for mapper

```
class WC(MRJob):

    def mapper(self, _, line):
        nr_words = len(line.split())
        nr_chars = len(line)
        yield ("words", nr_words)
        yield ("chars", nr_chars)

    def combiner(self, key, counts):
        if key == "words":
            yield ("words", sum(counts))
        elif key == "chars":
            yield ("chars", sum(counts))

    def reducer(self, key, counts):
        if key == "words":
            yield ("words", sum(counts))
        elif key == "chars":
            yield ("chars", sum(counts))

if __name__ == '__main__':
    WC.run()
```

Creates 2n tuples

Parallel: up to one node per line

Creates 2k tuples

As parallel as mapper

O(k) communication in Shuffle

Called twice with O(k) input

Parallel: two nodes

Continuing …

Implementing wordcount with Map-reduce

Q: how to get line count?

```python
from mrjob.job import MRJob

class WC(MRJob):

    def mapper(self, _, line):
        nr_words = len(line.split())
        nr_chars = len(line)
        yield ("words", nr_words)
        yield ("chars", nr_chars)

    def reducer(self, key, counts):
        if key == "words":
            yield ("words", sum(counts))
        elif key == "chars":
            yield ("chars", sum(counts))

if __name__ == '__main__':
     WC.run()
```

```
from mrjob.job import MRJob

class WC(MRJob):

    def mapper(self, _, line):
        nr_words = len(line.split())
        nr_chars = len(line)
        yield ("words", nr_words)
        yield ("chars", nr_chars)

    def reducer(self, key, counts):
        if key == "words":
            total = 0
            for i, c in enumerate(counts):
                total += c
            yield ("words", total)
            yield ("lines", i+1)
        elif key == "chars":
            yield ("chars", sum(counts))

if __name__ == '__main__':
     WC.run()
```

One
(words, nr_words)
tuple per line

line count with combiner?

```python
    def mapper(self, _, line):
        nr_words = len(line.split())
        nr_chars = len(line)
        yield ("words", nr_words)
        yield ("chars", nr_chars)

    def combiner(self, key, counts):
        if key == "words":
            yield ("words", sum(counts))
        elif key == "chars":
            yield ("chars", sum(counts))
```

One
(words, nr_words)
tuple per line on node

```python
    def reducer(self, key, counts):
        if key == "words":
            total = 0
            for i, c in enumerate(counts):
                total += c
            yield ("words", total)
            yield ("lines", i+1)
        elif key == "chars":
            yield ("chars", sum(counts))

if __name__ == '__main__':
    WC.run()
```

One
(words, nr_words)
tuple per node

```
    def combiner(self, key, counts):
        if key == "words":
            total = 0
            for i, c in enumerate(counts):
                total += c
            yield ("words", (i+1,total))
        elif key == "chars":
            yield ("chars", sum(counts))

    def reducer(self, key, counts):
        if key == "words":
            total_lines = 0
            total_words = 0
            for c in counts:
                total_lines += c[0]
                total_words += c[1]
            yield ("lines", total_lines)
            yield ("words", total_words)
        elif key == "chars":
            yield ("chars", sum(counts))

__name__ == '__main__':
```

Creates k tuples w/ aggregated word, line counts

```
    def combiner(self, key, counts):
        if key == "words":
            total = 0
            for i, c in enumerate(counts):
                total += c
            yield ("words", (i+1,total))
        elif key == "chars":
            yield ("chars", sum(counts))

    def reducer(self, key, counts):
        if key == "words":
            total_lines = 0
            total_words = 0
            for c in counts:
                total_lines += c[0]
                total_words += c[1]
            yield ("lines", total_lines)
            yield ("words", total_words)
        elif key == "chars":
            yield ("chars", sum(counts))

__name__ == '__main__':
```

Sums over the k tuples

Implementing wordcount with Map-reduce

Q: how to get average #words and #character per line??

The following code is correct

```
def reducer(self, key, counts):
    if key == "words":
        total_lines = 0
        total_words = 0
        for c in counts:
            total_lines += c[0]
            total_words += c[1]
        yield ("lines", total_lines)
        yield ("words", total_words)
        yield ("wordsperline",
               float(total_words)/total_lines)
    elif key == "chars":
        yield ("chars", sum(counts))
        charsperline = float(sum(counts)) / total_lines
```

A	B	C	D	E
Yes	No			

```
def reducer(self, key, counts):
    if key == "words":
        total_lines = 0
        total_words = 0
        for c in counts:
            total_lines += c[0]
            total_words += c[1]
        yield ("lines", total_lines)
        yield ("words", total_words)
        yield (“wordsperline",
               float(total_words)/total_lines)
    elif key == "chars":
        yield ("chars", sum(counts))
```

Not available at same time!

Possibly not available on the same machine average #characters per word?

```
class WC(MRJob):

    def mapper(self, _, line):
        nr_words = len(line.split())
        nr_chars = len(line)
        yield ("linestats", (nr_words, nr_chars))

    def combiner(self, key, counts):
        nr_words = 0
        nr_chars = 0
        for i, c in enumerate(counts):
            nr_words += c[0]
            nr_chars += c[1]
        nr_lines = i + 1
        yield ("stats", (nr_lines, nr_words, nr_chars))


    def reducer(self, key, counts):
        total_lines = 0
        total_words = 0
        total_chars = 0
        for c in counts:
            total_lines += c[0]
```

n = # lines

```
def mapper(self, _, line):
    nr_words = len(line.split())
    nr_chars = len(line)
    yield ("linestats", (nr_words, nr_chars))

def combiner(self, key, counts):
    nr_words = 0
    nr_chars = 0
    for i, c in enumerate(counts):
        nr_words += c[0]
        nr_chars += c[1]
    nr_lines = i + 1
    yield ("stats", (nr_lines, nr_words, nr_chars))
```

The maximal speedup achievable by the reducer (e.g. on how many machines can it be run) is …

A	B	C	D	E
1	2	4	log(n)	n

```
        yield ("stats", (nr_lines, nr_words, nr_chars))


    def reducer(self, key, counts):
        total_lines = 0
        total_words = 0
        total_chars = 0
        for c in counts:
            total_lines += c[0]
            total_words += c[1]
            total_chars += c[2]

        yield ("lines", total_lines)
        yield ("words", total_words)
        yield ("chars", total_chars)

        yield ("wordsperline", float(total_words)/total_lines)
        yield ("charsperline", float(total_chars)/total_lines)
        yield ("charsperword", float(total_chars)/total_words)
```

Working with Map-reduce

Q: how to pass parameters?

```
""" Find duplicate keys in a file containing lines consisting of
    key,value
"""
from mrjob.job import MRJob

class FindDuplicates(MRJob):

    def mapper(self, _, line):
        key, value = line.split(',')
        yield (key, 1)

    def combiner(self, key, counts):
        yield (key, sum(counts))

    def reducer(self, key, counts):
        s = sum(counts)
        if s > 1:
            yield (key, s)


if __name__ == '__main__':
    FindDuplicates.run()
```

Set threshold from command line

```python
from mrjob.job import MRJob

class FindDuplicates(MRJob):

    def mapper(self, _, line):
        key, value = line.split(',')
        yield (key, 1)

    def combiner(self, key, counts):
        yield (key, sum(counts))

    def reducer_init(self):
        self.threshold = 42

    def reducer(self, key, counts):
        s = sum(counts)
        if s > self.threshold:
            yield (key, s)


if __name__ == '__main__':
     FindDuplicates.run()
```

Executed once (per node)
before running reducer

Used in every call

```
ss FindDuplicates(MRJob):

    def mapper(self, _, line):
        key, value = line.split(',')
        yield (key, 1)

    def combiner(self, key, counts):
        yield (key, sum(counts))

    def configure_args(self):
        super(FindDuplicates, self).configure_args()
        self.add_passthru_arg('--threshold', default=1,
                              help="Frequency threshold")

    def reducer_init(self):
        self.threshold = int(self.options.threshold)

    def reducer(self, key, counts):
        s = sum(counts)
        if s > self.threshold:
            yield (key, s)


__name__ == '__main__':
```

Define command line arguments for program

```
ss FindDuplicates(MRJob):

    def mapper(self, _, line):
        key, value = line.split(',')
        yield (key, 1)

    def combiner(self, key, counts):
        yield (key, sum(counts))

    def configure_args(self):
        super(FindDuplicates, self).configure_args()
        self.add_passthru_arg('--threshold', default=1,
                              help="Frequency threshold")

    def reducer_init(self):
        self.threshold = int(self.options.threshold)

    def reducer(self, key, counts):
        s = sum(counts)
        if s > self.threshold:
            yield (key, s)


name__ == '__main__':
```

Configure arguments for super class

```
   ss FindDuplicates(MRJob):

    def mapper(self, _, line):
        key, value = line.split(',')
        yield (key, 1)

    def combiner(self, key, counts):
        yield (key, sum(counts))

    def configure_args(self):
        super(FindDuplicates, self).configure_args()
        self.add_passthru_arg('--threshold', default=1,
                              help="Frequency threshold")

    def reducer_init(self):
        self.threshold = int(self.options.threshold)

    def reducer(self, key, counts):
        s = sum(counts)
        if s > self.threshold:
            yield (key, s)


     name    == '  main  ':
```

Add a new argument. Mak sure does not clash with MrJob

```
ss FindDuplicates(MRJob):

    def mapper(self, _, line):
        key, value = line.split(',')
        yield (key, 1)

    def combiner(self, key, counts):
        yield (key, sum(counts))

    def configure_args(self):
        super(FindDuplicates, self).configure_args()
        self.add_passthru_arg('--threshold', default=1,
                              help="Frequency threshold")

    def reducer_init(self):
        self.threshold = int(self.options.threshold)

    def reducer(self, key, counts):
        s = sum(counts)
        if s > self.threshold:
            yield (key, s)

__name__ == '__main__':
```

self.options is a namespace holding all command line arguments

threshold = 1 unless specified on the command line

```
MacBookPro13:Sandbox schliep$ python mrjob-duplicates-combiner-2.py --threshold 121 test.data
No configs found; falling back on auto-configuration
No configs specified for inline runner
Running step 1 of 1...
Creating temp directory /var/folders/wx/9cjwzycx77j7dxc0nh5zbn8m0000gn/T/mrjob-duplicates-
combiner-2.schliep.20180425.213944.509621
Streaming final output from /var/folders/wx/9cjwzycx77j7dxc0nh5zbn8m0000gn/T/mrjob-
duplicates-combiner-2.schliep.20180425.213944.509621/output...
"0000000129" 126
"0000000189" 126
"0000000454" 131
"0000000480" 123
"0000000551" 133
"0000000591" 126
"0000000639" 140
"0000000669" 124
"0000000847" 130
"0000000934" 128
"0000000945" 133
```

Working with Map-reduce

Q: Multiple steps or how to find the most frequent item?

most frequent item?

```python
from mrjob.job import MRJob

class FindDuplicates(MRJob):

    def mapper(self, _, line):
        key, value = line.split(',')
        yield (key, 1)

    def combiner(self, key, counts):
        yield (key, sum(counts))


    def reducer(self, key, counts):
        s = sum(counts)
        if s > self.threshold:
            yield (key, s)

if __name__ == '__main__':
    FindDuplicates.run()
```

After reducer has completed all words and their frequencies are available

Need one more step!

```
from mrjob.job import MRJob, MRStep

class FindDuplicates(MRJob):

    def mapper(self, _, line):
  ...
    def combiner(self, key, counts):
        ...

    def reducer(self, key, counts):
        s = sum(counts)
        yield None, (s, key)

    def findmax(self, _, count_word_tuples):
        yield max(count_word_tuples)

    def steps(self):
        return [MRStep(mapper=self.mapper,
                       combiner=self.combiner,
                       reducer=self.reducer),
                MRStep(reducer=self.findmax)]

if __name__ == '__main__':
```

Produces (freq,word)
pairs

```
from mrjob.job import MRJob, MRStep

class FindDuplicates(MRJob):

    def mapper(self, _, line):
 ...
    def combiner(self, key, counts):
        ...

    def reducer(self, key, counts):
        s = sum(counts)
        yield None, (s, key)

    def findmax(self, _, count_word_tuples):
        yield max(count_word_tuples)

    def steps(self):
        return [MRStep(mapper=self.mapper,
                       combiner=self.combiner,
                       reducer=self.reducer),
                MRStep(reducer=self.findmax)]

if __name__ == '__main__':
```

max looks at first item in tuple

```
from mrjob.job import MRJob, MRStep

class FindDuplicates(MRJob):

    def mapper(self, _, line):
  ...
    def combiner(self, key, counts):
        ...

    def reducer(self, key, counts):
        s = sum(counts)
        yield None, (s, key)

    def findmax(self, _, count_word_tuples):
        yield max(count_word_tuples)

    def steps(self):
        return [MRStep(mapper=self.mapper,
                       combiner=self.combiner,
                       reducer=self.reducer),
                MRStep(reducer=self.findmax)]

if __name__ == '__main__':
```

Manually define multi-step execution

```
from mrjob.job import MRJob, MRStep

class FindDuplicates(MRJob):

    def mapper(self, _, line):
  ..
```

Default MRJob steps method

```
 def steps(self):
         return [MRStep(mapper=self.mapper,
                        combiner=self.combiner,
                        reducer=self.reducer)]
```

Can specify init functions. Steps can have one or all of the commands

fine

P

n

```
                        combiner=self.combiner,
                        reducer=self.reducer),
                 MRStep(reducer=self.findmax)]

if __name__ == '__main__':
```

Parallel programming models

Spark

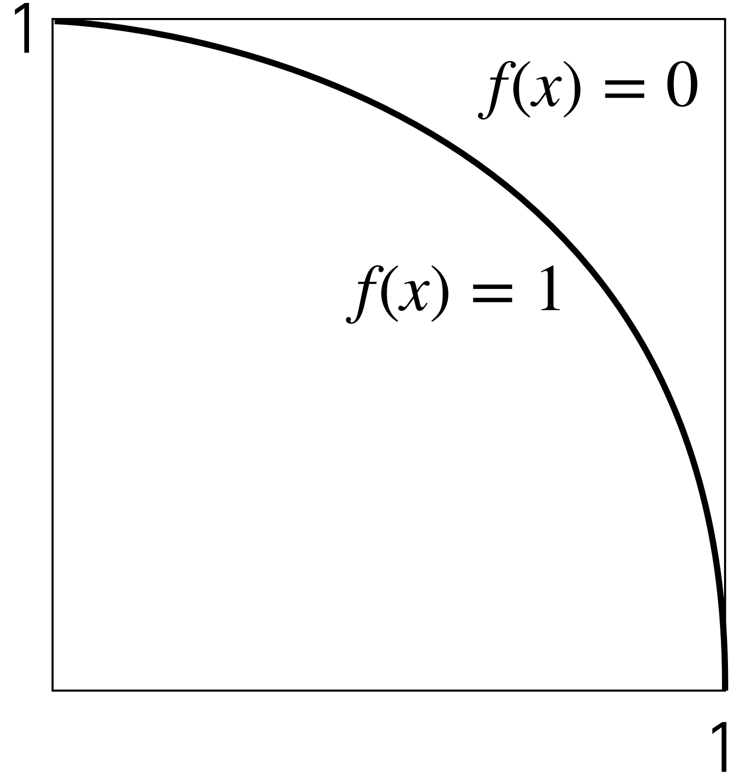
Apache Spark

- More general model of computation
- Automated parallelization
- In memory (with disk caching) computation
- Explicit program flow

Working with Spark

Computing pi with Monte Carlo

Monte Carlo: Computing Pi



- Random variable $X \sim \pi$
- When x_i drawn from π

$$\frac{1}{n}\sum f(x_i) \rightarrow E_\pi[f(X)]$$

- Example:

$$\frac{1}{n}\sum f(x_i) \rightarrow \frac{\pi}{4} \quad x_i \sim U[0,1]^2$$

$$\frac{1}{n}\sum f(x_i) \rightarrow E_\pi[f(X)] = \int_0^1 \sqrt{1 - x^2}dx$$

Reminder: Computing Pi with multiprocessing

```
def sample_pi(n):
    """ Perform n steps of Monte Carlo simulation for estimating Pi/4.
        Returns the number of successes."""
    s = 0
    for i in range(n):
        x,y = random.random(), random.random()
        if x**2 + y**2 <= 1.0:
            s += 1
    return s


def compute_pi(args):
    n = args.steps / args.workers

    p = multiprocessing.Pool(args.workers)
    s = p.map(sample_pi, [n]*args.workers)

    n_total = n*args.workers
    s_total = sum(s)
    pi_est = (4.0*s_total)/n_total
    print " Steps\tSuccess\tPi est.\tError"
    print "%6d\t%7d\t%1.5f\t%1.5f" % (n_total, s_total, pi_est, pi-pi_est)
```

Computing Pi with multiprocessing

```
def sample_pi(n):
    """ Perform n steps of Monte Carlo simulation for estimating Pi/4.
        Returns the number of successes."""
    s = 0
    for i in range(n):
        x,y = random.random(), random.random()
        if x**2 + y**2 <= 1.0:
            s += 1
    return s


def compute_pi(args):
    n = args.steps / args.workers

    p = multiprocessing.Pool(args.workers)
    s = p.map(sample_pi, [n]*args.workers)

    n_total = n*args.workers
    s_total = sum(s)
    pi_est = (4.0*s_total)/n_total
    print " Steps\tSuccess\tPi est.\tError"
    print "%6d\t%7d\t%1.5f\t%1.5f" % (n_total, s_total, pi_est, pi-pi_est)
```

Sample x,y uniform and
check whether it is within
the quarter circle

Computing Pi with multiprocessing

```
def sample_pi(n):
    """ Perform n steps of Monte Carlo simulation for estimating Pi/4.
        Returns the number of successes."""
    s = 0
    for i in range(n):
        x,y = random.random(), random.random()
        if x**2 + y**2 <= 1.0:
            s += 1
    return s


def compute_pi(args):
    n = args.steps / args.workers

    p = multiprocessing.Pool(args.workers)
    s = p.map(sample_pi, [n]*args.workers)

    n_total = n*args.workers
    s_total = sum(s)
    pi_est = (4.0*s_total)/n_total
    print " Steps\tSuccess\tPi est.\tError"
    print "%6d\t%7d\t%1.5f\t%1.5f" % (n_total, s_total, pi_est, pi-pi_est)
```

[n,…,n]

Create workers and let
each worker process
sample_pi(n)

Computing Pi with PySpark

```
import random
from pyspark import SparkContext
sc = SparkContext(master = 'local[4]')

NUM_SAMPLES = 100000000

def inside(p):
    x, y = random.random(), random.random()
    return x*x + y*y < 1

count = sc.parallelize(xrange(0, NUM_SAMPLES)) \
             .filter(inside).count()

print "Pi is roughly %f" % (4.0 * count / NUM_SAMPLES)
```

Computing Pi with PySpark

```
import random
from pyspark import SparkContext
sc = SparkContext(master = 'local[4]')

NUM_SAMPLES = 100000000

def inside(p):
    x, y = random.random(), random.random()
    return x*x + y*y < 1

count = sc.parallelize(xrange(0, NUM_SAMPLES)) \
             .filter(inside).count()

print "Pi is roughly %f" % (4.0 * count / NUM_SAMPLES)
```

Boilerplate for
computations on 4
local cores

Computing Pi with PySpark

```
import random
from pyspark import SparkContext
sc = SparkContext(master = 'local[4]')

NUM_SAMPLES = 100000000

def inside(p):
    x, y = random.random(), random.random()
    return x*x + y*y < 1

count = sc.parallelize(xrange(0, NUM_SAMPLES)) \
                .filter(inside).count()

print "Pi is roughly %f" % (4.0 * count / NUM_SAMPLES)
```

Sample x,y uniform and check whether it is within the quarter circle. p is ignored

Computing Pi with PySpark

```
import random
from pyspark import SparkContext
sc = SparkContext(master = 'local[4]')

NUM_SAMPLES = 100000000

def inside(p):
    x, y = random.random(), random.random()
    return x*x + y*y < 1

count = sc.parallelize(xrange(0, NUM_SAMPLES)) \
             .filter(inside).count()

print "Pi is roughly %f" % (4.0 * count / NUM_SAMPLES)
```

Computing Pi with PySpark

```
count = sc.parallelize(xrange(0, NUM_SAMPLES)) \
             .filter(inside).count()
```

Computing Pi with PySpark

```
count = sc.parallelize(xrange(0, NUM_SAMPLES)) \
                .filter(inside).count()
```

Example

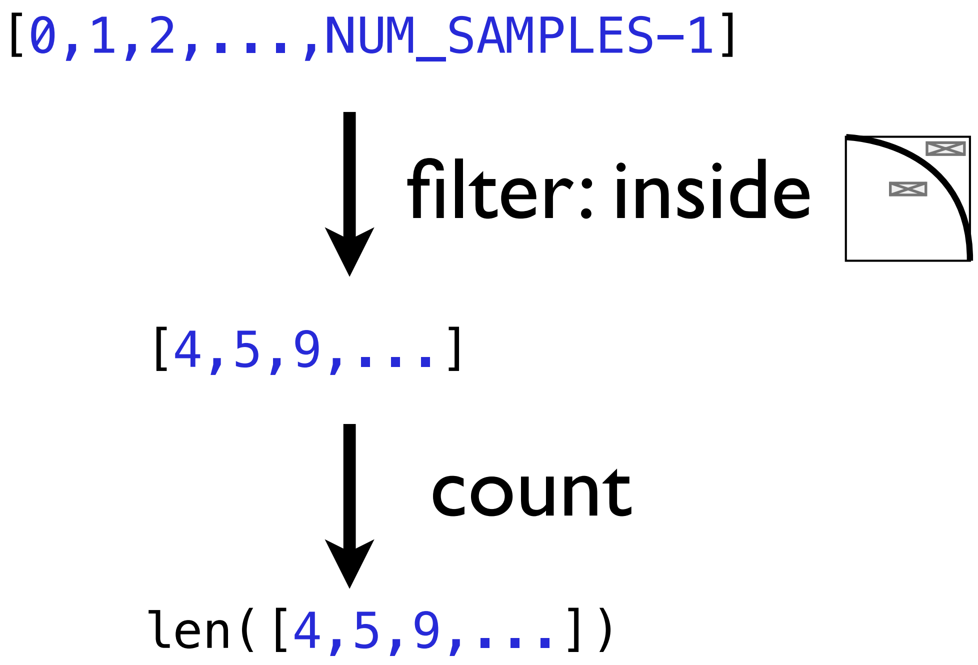
```
>>> def odd(x):
...     return x % 2
...
>>> a = [2,3,5,8,9]
>>> b = filter(odd, a)
>>> b
[3, 5, 9]
>>> len(b)
3
```

Computing Pi with PySpark

```
count = sc.parallelize(xrange(0, NUM_SAMPLES)) \
             .filter(inside).count()
```

Example

```
>>> def odd(x):
...     return x % 2
...
>>> a = [2,3,5,8,9]
>>> b = filter(odd, a)
>>> b
[3, 5, 9]
>>> len(b)
3
```



[0,1,2,...,NUM_SAMPLES-1]

filter: inside

[4,5,9,...]

count

len([4,5,9,...])

Note: array not created

Computing Pi with PySpark

```
data = [0,1,2,3,4,5]
distData = sc.parallelize(data)
```

- RDD = Resilient Distributed Dataset
- Fault-tolerant collection of records which can be processed in parallel

```
distData.reduce(lambda a, b: a + b)
```

- Obtained from Driver program (your Python script), read from HDFS or other data sources

Find duplicates with MrJob

```
""" Find duplicate keys in a file containing lines consisting of
    key,value
"""
from mrjob.job import MRJob

class FindDuplicates(MRJob):

    def mapper(self, _, line):
        key, value = line.split(',')
        yield (key, 1)

    def combiner(self, key, counts):
        yield (key, sum(counts))

    def reducer(self, key, counts):
        s = sum(counts)
        if s > 1:
            yield (key, s)


if __name__ == '__main__':
    FindDuplicates.run()
```

Find duplicates with MrJob

```python
""" Find duplicate keys in a file containing lines consisting of
    key,value
"""
from mrjob.job import MRJob

class FindDuplicates(MRJob):

    def mapper(self, _, line):
        key, value = line.split(',')
        yield (key, 1)

    def combiner(self, key, counts):
        yield (key, sum(counts))

    def reducer(self, key, counts):
        s = sum(counts)
        if s > 1:
            yield (key, s)


if __name__ == '__main__':
    FindDuplicates.run()
```

Create one tuple
per line of file

Find duplicates with MrJob

```
""" Find duplicate keys in a file containing lines consisting of
    key,value
"""
from mrjob.job import MRJob

class FindDuplicates(MRJob):

    def mapper(self, _, line):
        key, value = line.split(',')
        yield (key, 1)

    def combiner(self, key, counts):
        yield (key, sum(counts))

    def reducer(self, key, counts):
        s = sum(counts)
        if s > 1:
            yield (key, s)


if __name__ == '__main__':
    FindDuplicates.run()
```

Collect all tuples
with same key per
node and sum

Find duplicates with MrJob

```python
""" Find duplicate keys in a file containing lines consisting of
    key,value
"""
from mrjob.job import MRJob

class FindDuplicates(MRJob):

    def mapper(self, _, line):
        key, value = line.split(',')
        yield (key, 1)

    def combiner(self, key, counts):
        yield (key, sum(counts))

    def reducer(self, key, counts):
        s = sum(counts)
        if s > 1:
            yield (key, s)


if __name__ == '__main__':
    FindDuplicates.run()
```

Collect all tuples
with same key across
nodes and sum.
Output duplicates.

Find duplicates with PySpark

```
import random
from pyspark import SparkContext
sc = SparkContext(master = 'local[4]')

distFile = sc.textFile("test.data")

counts = distFile.map(lambda l: l.split(',')) \
         .map(lambda t:(int(t[0]),1)) \
         .reduceByKey(lambda a, b: a + b) \
         .filter(lambda t:t[1] > 1)


cc = counts.collect()
print cc
```

Distribute Local File

Collect output

Find duplicates with PySpark

```
import random
from pyspark import SparkContext
sc = SparkContext(master = 'local[4]')

distFile = sc.textFile("test.data")

counts = distFile.map(lambda l: l.split(',')) \
         .map(lambda t:(int(t[0]),1)) \
         .reduceByKey(lambda a, b: a + b) \
         .filter(lambda t:t[1] > 1)


cc = counts.collect()
print cc
```

Collect output

Lambda expressions

```
>>> def square(x):
...     return x**x
...
>>> map(square,range(10))
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> map(lambda x:x**2, range(10))
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

- Lambda expression are anonymous (nameless) functions
- Often used in sort ...
- lambda x,y,z,...:<expression>

Find duplicates with PySpark

```
counts = distFile.map(lambda l: l.split(',')) \
         .map(lambda t:(int(t[0]),1)) \
         .reduceByKey(lambda a, b: a + b) \
         .filter(lambda t:t[1] > 1)
```

Split key,value string into tuple

Find duplicates with PySpark

```
counts = distFile.map(lambda l: l.split(',')) \
         .map(lambda t:(int(t[0]),1)) \
         .reduceByKey(lambda a, b: a + b) \
         .filter(lambda t:t[1] > 1)
```

Create (key,1) tuples

Find duplicates with PySpark

```
counts = distFile.map(lambda l: l.split(',')) \
         .map(lambda t:(int(t[0]),1)) \
         .reduceByKey(lambda a, b: a + b) \
         .filter(lambda t:t[1] > 1)
```

Collect all tuples
with same key and
reduce.

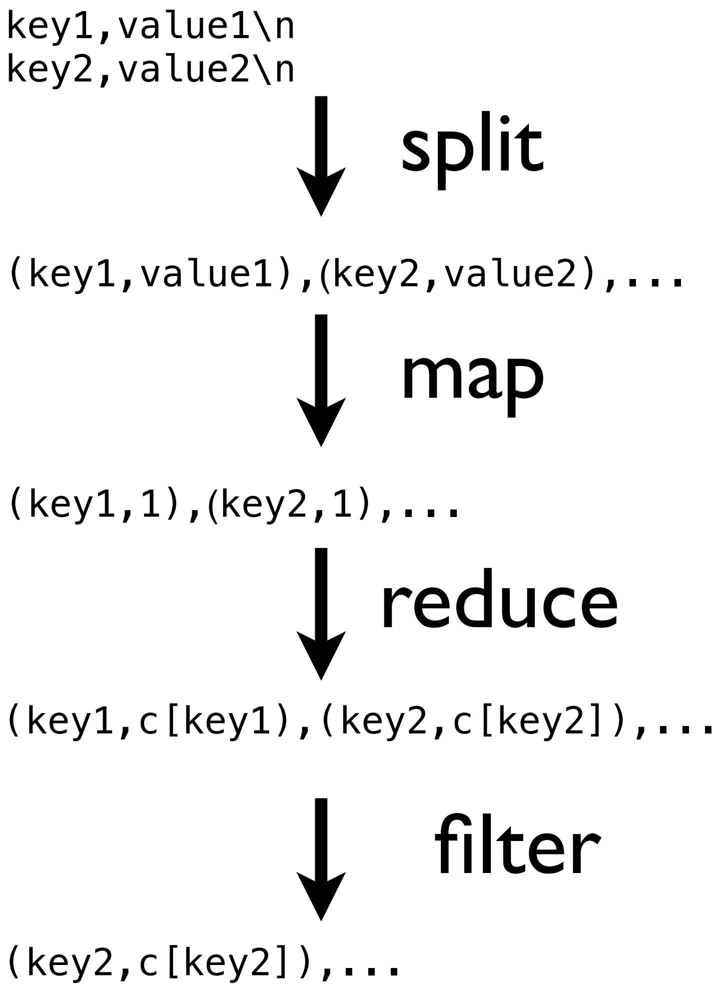
Find duplicates with PySpark

```
counts = distFile.map(lambda l: l.split(',')) \
         .map(lambda t:(int(t[0]),1)) \
         .reduceByKey(lambda a, b: a + b) \
         .filter(lambda t:t[1] > 1)
```

Filter

Find duplicates with PySpark

```
counts = distFile.map(lambda l: l.split(',')) \
         .map(lambda t:(int(t[0]),1)) \
         .reduceByKey(lambda a, b: a + b) \
         .filter(lambda t:t[1] > 1)
```



```
key1,value1\n
key2,value2\n
```

split

```
(key1,value1),(key2,value2),...
```

map

```
(key1,1),(key2,1),...
```

reduce

```
(key1,c[key1),(key2,c[key2]),...
```

filter

```
(key2,c[key2]),...
```

Find duplicates with PySpark

```
counts = distFile.map(lambda l: l.split(',')) \
         .map(lambda t:(int(t[0]),1)) \
         .reduceByKey(lambda a, b: a + b) \
         .filter(lambda t:t[1] > 1)
```

PySpark == MapReduce ???

Assume you want a bit more than duplicates, e.g.

```
There are 1001 unique keys of which 461 appear
more than 100 times (sum = 50100). The
following keys appear over 125 times (sum =
1173)
[(00454, 131), (00934, 128), (00639, 140),
(00189, 126), (00847, 130), (00591, 126),
(00129, 126), (05051, 133), (00945, 133)]
```

How would you implement this in MapReduce?

Assume you want a bit more than duplicates, e.g.

```
There are 1001 unique keys of which 461 appear
more than 100 times (sum = 50100). The
following keys appear over 125 times (sum =
1173)
[(00454, 131), (00934, 128), (00639, 140),
(00189, 126), (00847, 130), (00591, 126),
(00129, 126), (05051, 133), (00945, 133)]
```

Can the output above be produced effectively using MapReduce?

A	B	C	D	E
True	False			

```python
import random
from pyspark import SparkContext
sc = SparkContext(master = 'local[4]')

distFile = sc.textFile("test.data")

keytuples = distFile.map(lambda l: l.split(',')) \
         .map(lambda t:(int(t[0]),1))

countsPerKey = keytuples.reduceByKey(lambda a, b: a + b)

unique_keys_count = countsPerKey.count()

frequent_keys = countsPerKey.filter(lambda t:t[1] > 100)

frequent_keys_count = frequent_keys.count()
frequent_keys_sum = frequent_keys.values().sum()

very_frequent_keys = frequent_keys.filter(lambda t:t[1] > 125)
very_frequent_keys_sum = very_frequent_keys.values().sum()
vfk = very_frequent_keys.collect()

print(f"There are {unique_keys_count} unique keys in the RDD of which {frequent_keys_count} "
      f"appear more than 100 times (sum = {frequent_keys_sum}). The following keys appear "
      f"over 125 times (sum = {very_frequent_keys_sum})")
print(vfk)
```

Q&A from Chat

Q: Speedup for reduce? A: Reduce is run once for each distinct key used in tuples yielded by combine. If there are only keys A, B, C, D, then reduce is called once each for A, B, C, D, most likely in parallel on 4 nodes. If there is only one key, say A, then reduce is called exactly once.

Q: multiprocessing starmap vs map. A: Starmap is exactly like map in multiprocessing, only allowing for multiple arguments to the function applied. E.g. for f(x) = sin(x) use map[f, [0.1,0.2]) and for g(x,y) = x^2 + y^2 use starmap(g, [(0.0,1.0), (0.5, 1.2), ..

Q: Why collect? A: In the last example, counts is a RDD, with data likely existing on different nodes, not a basic Python data type or numpy array, and with collect() we transform that into something we can process in Python.

Q: Where will filter (or generally RDD transformations) be executed? That is not specified. We will briefly touch on scheduling computations in Spark. Generally, Spark is more flexible and dynamic than MapReduce, so where computations will be executed is less predictable, but probably more efficient than MapReduce. Note that combiners are a “poor-man’s” version of reduce in some sense. Spark embraces the efficiency of reduce at all levels.

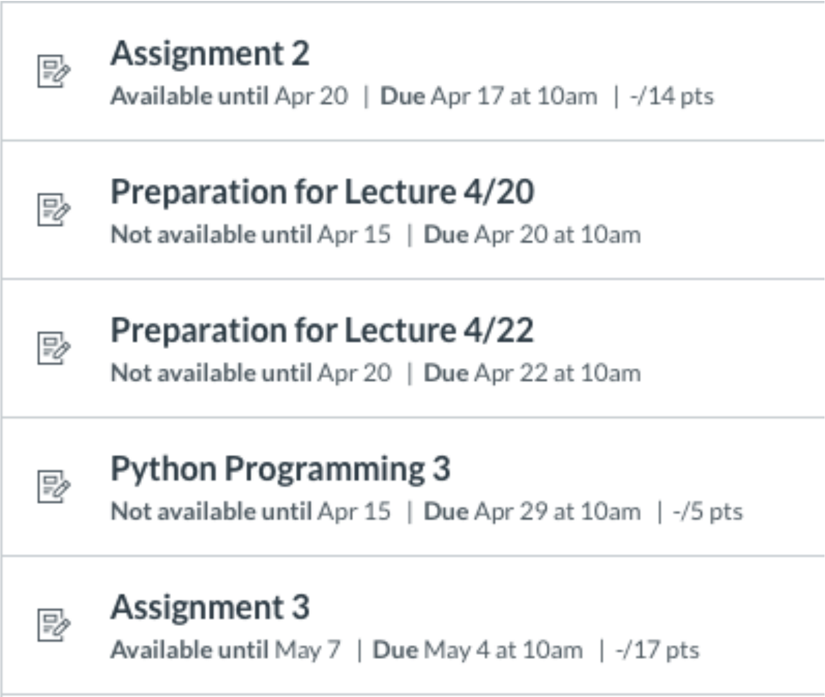
DIT 873 & DAT346 Techniques for Large-Scale Data

Lecture 5

Agenda for today:

- Map/Reduce
- Spark

Upcoming Deadlines



Assignment 2
Available until Apr 20 | **Due** Apr 17 at 10am | -/14 pts

Preparation for Lecture 4/20
Not available until Apr 15 | **Due** Apr 20 at 10am

Preparation for Lecture 4/22
Not available until Apr 20 | **Due** Apr 22 at 10am

Python Programming 3
Not available until Apr 15 | **Due** Apr 29 at 10am | -/5 pts

Assignment 3
Available until May 7 | **Due** May 4 at 10am | -/17 pts