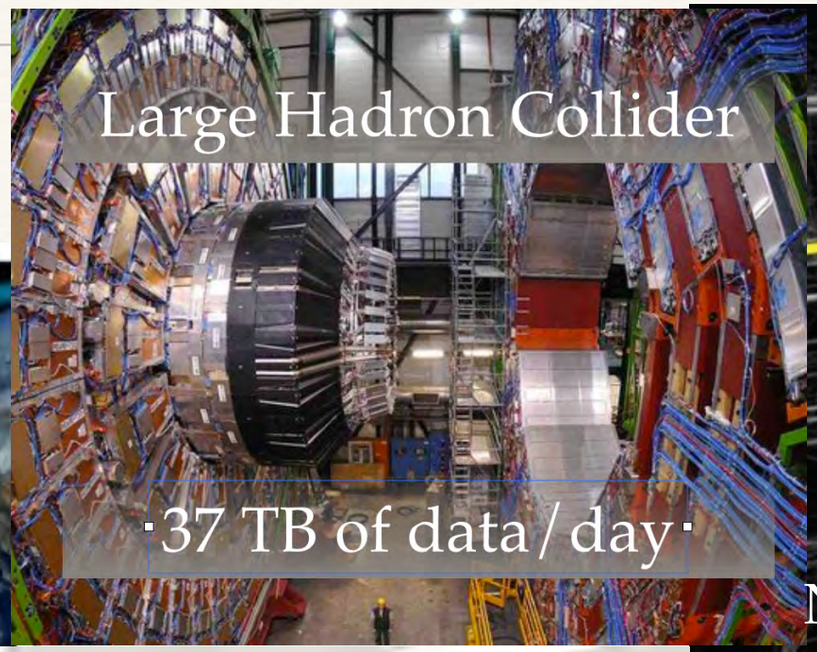
Michael Grossberg

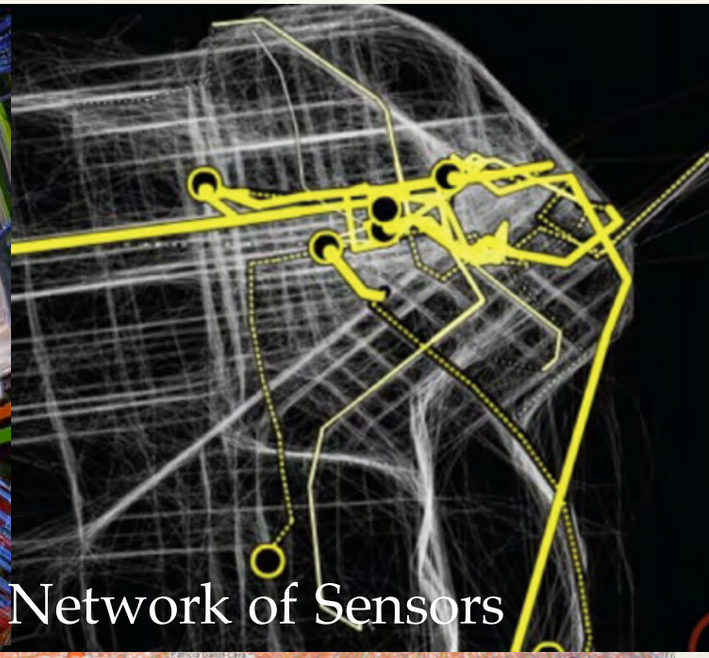
Intro to Data Science

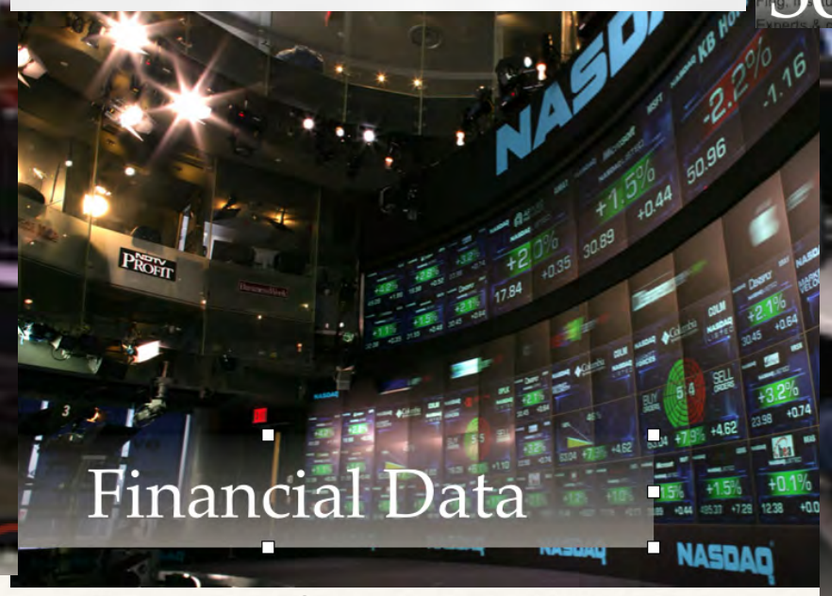
CS59969

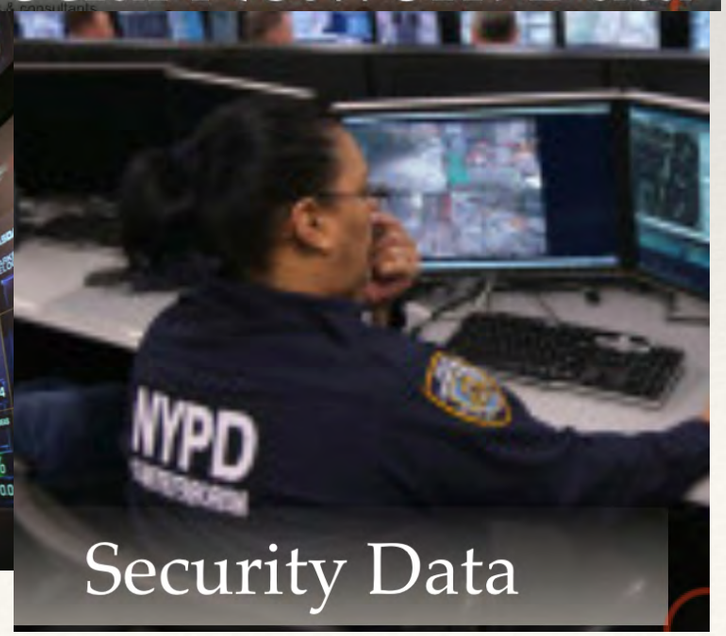
Map Reduce

Flood of Data









How do we run algs on Big Data?

Split Jobs
across many machine

How?

Python “pipes” with subprocess

```
import subprocess

print('word-count:')
proc = subprocess.Popen(
    'head -n 5 ./aliceinwonderland.txt',
    shell=True,
    stdin=subprocess.PIPE,
    stdout=subprocess.PIPE,
    stderr=subprocess.PIPE,
)

msg = ''.encode('utf-8')
stdout_value, stderr_value = proc.communicate(msg)
print('pass through:', stdout_value.decode('utf-8'))
print('stderr      :', stderr_value.decode('utf-8'))
```

Connecting Pipes

```
import subprocess

print('word-count:')
# Pipe once process
procCut = subprocess.Popen(
    'head -n 20 ./aliceinwonderland.txt',
    shell=True,
    stdin=subprocess.PIPE,
    stdout=subprocess.PIPE,
    stderr=subprocess.PIPE,
)
# Into another
msg = ''.encode('utf-8')
stdout_value, stderr_value = procCut.communicate(msg)
procCount = subprocess.Popen(
    'wc',
    shell=True,
    stdin=subprocess.PIPE,
    stdout=subprocess.PIPE,
    stderr=subprocess.PIPE,
)
msg = stdout_value
stdout_value, stderr_value = procCount.communicate(msg)
print('pass through:', stdout_value.decode('utf-8'))
print('stderr      :', stderr_value.decode('utf-8'))
```

Execution in Chunks

```
import subprocess
outputs = []; step = 1000; start=1; end= 4000
# Compute one chunk at-a-time (not actually parallel)
for startLine in range(start,end,step):
    print(startLine)
    cmd = ('tail -n +' + str(startLine)
           + ' ./aliceinwonderland.txt | head -n '
           + str(step) + ' | wc')
    print("cmd", cmd)
    procCut = subprocess.Popen(
        cmd,
        shell=True,
        stdin=subprocess.PIPE,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
    )
    msg = ''.encode('utf-8')
    stdout_value, stderr_value = procCut.communicate(msg)
    outputs.append(stdout_value)
# Combine
print('word-count:')
line_count, words_count, chars_count = 0, 0, 0

for output in outputs:
    lines, words, chars = [int(val) for val in output.decode('utf-8').split()]
    line_count += lines
    words_count += words
    chars_count += chars

print('line_count: ', line_count,
      'words_count: ', words_count,
      'chars_count: ', chars_count)
```

Parallel with subprocess

- Complex … need to set up threads or use file-io
- Need to keep track of when sub-jobs complete before compiling results

Other python libraries

- multiprocessing (handles the threading)
- toolz: makes python like functional language (pipes)
- Dask: exploits multiple core and to-big to fit in memory

Still use 1 machine

How to use multiple-machines?

- Could remotely execute using ssh or python-fabric
- Complex management issues:
  - How do you synchronize multiple processes?
  - How do you deal with node failure?
  - How do you provision and manage multiple machines?

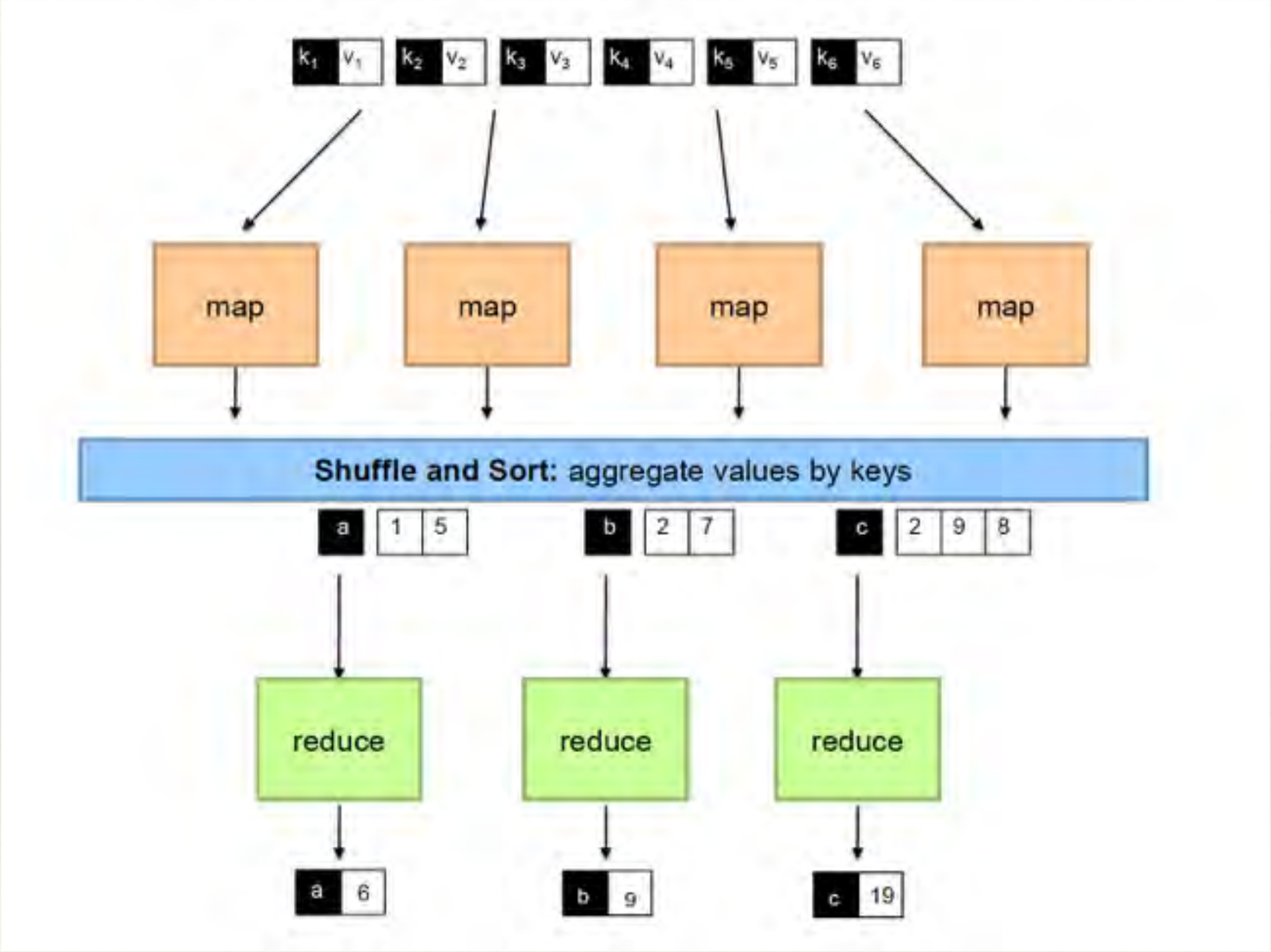
Map-Reduce

- Force specific form of parallel problem
- Framework takes care of the complexity

Python cartoon of map reduce

```
unique_keys = list(set(list_of_keys))
collected_data = {key:[] for key in unique_keys}

# Map step
for key, data in zip(list_of_keys,list_of_data):
    collected_data[key].append(map_process(data))
    # note map_process can be independent of others

#Reduce step
{key:reduce_process(key,values) for key, values
                                in collected_data}
```

Simple Map Reduce Diagram



mrjob (python library)

mrjob v0.5.6 documentation

Need help?

Join the mailing list by visiting the Google group page or sending an email to mrjob+subscribe@googlegroups.com.

mrjob

mrjob lets you write MapReduce jobs in Python 2.6+/3.3+ and run them on several platforms.
You can:

- Write multi-step MapReduce jobs in pure Python
- Test on your local machine
- Run on a Hadoop cluster
- Run in the cloud using Amazon Elastic MapReduce (EMR)
- Run in the cloud using Google Cloud Dataproc (Dataproc)

To get started, install with `pip`:

```
pip install mrjob
```

and begin reading the tutorial below.

Guides

Why mrjob?

WC map reduce

```
from mrjob.job import MRJob


class MRWordFrequencyCount(MRJob):

    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        yield "lines", 1

    def reducer(self, key, values):
        yield key, sum(values)


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

Run as

```
mamilla:mapreduce michael$ python mr_word_count.py aliceinwonderland.txt
No configs found; falling back on auto-configuration
Creating temp directory /var/folders/zb/29s1rt191nd255rn_x8_42qc0000gn/T/mr_word_count.michael.2
0161108.044646.306951
Running step 1 of 1...
Streaming final output from /var/folders/zb/29s1rt191nd255rn_x8_42qc0000gn/T/mr_word_count.micha
el.20161108.044646.306951/output...
"chars" 141064
"lines" 3337
"words" 26444
Removing temp directory /var/folders/zb/29s1rt191nd255rn_x8_42qc0000gn/T/mr_word_count.michael.2
0161108.044646.306951...
mamilla:mapreduce michael$
```

Actually running locally but same code

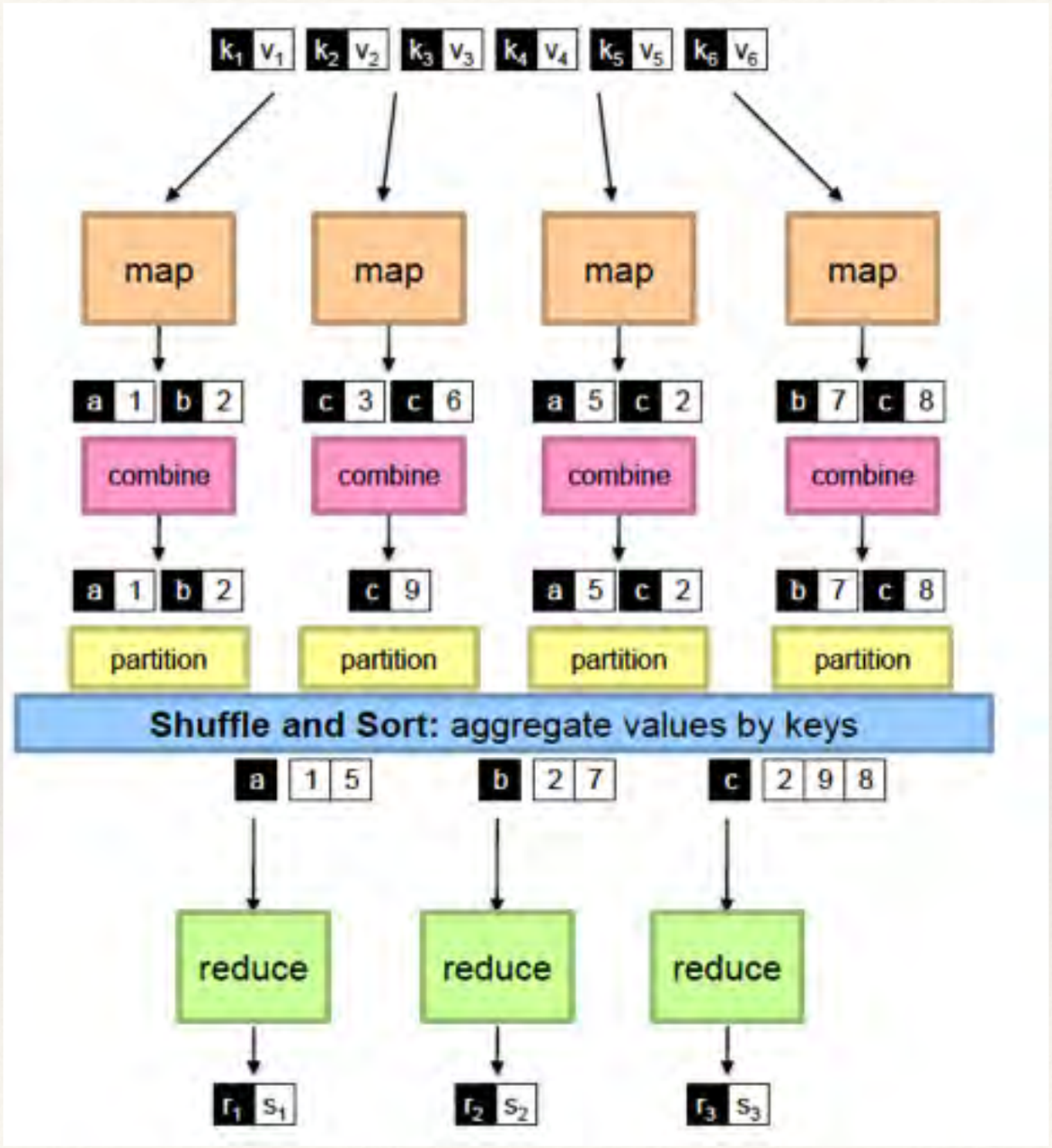
Word Frequency as MR

```
from mrjob.job import MRJob
import re

class MRWordFrequencyCount(MRJob):

    def mapper(self, key, line):
        # strip punctuation and extra spaces
        line = re.sub(r'  ',' ',
               re.sub(r'[^0-9A-Za-z]',' ',line))
        for word in line.split():
            yield word.lower(), 1

    def reducer(self, word, occurrences):
        yield word, sum(occurrences)


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

Output

```
mamilla:mapreduce michael$ python mr_word_count.py aliceinwonderland.txt
No configs found; falling back on auto-configuration
Creating temp directory /var/folders/zb/29s1rt191nd255rn_x8_42qc0000gn/T/mr_word_count.michael.2
0161108.184111.540061
Running step 1 of 1...
Streaming final output from /var/folders/zb/29s1rt191nd255rn_x8_42qc0000gn/T/mr_word_count.micha
el.20161108.184111.540061/output...
"a"     632
"abide" 1
"able"  1
"about" 94
"above" 3
"absence"       1
"absurd"        2
"acceptance"    1
"accident"      2
"accidentally"  1
"account"       1
"accounting"    1
"accounts"      1
"accusation"    1
"accustomed"    1
"ache"  1
"across"        5
"act"   1
"actually"      1
"ada"   1
"added" 23
```

Advanced Map Reduce



Rules of Combiners

- Optional mini-reducer
- Runs before shuffle-sort
- Can reduce traffic
- (may) get only **some** mappers
- May **not** get all of a key
- May not run at ALL
- Same output as mapper!
- May run multiple times!

Word Frequency with Combiner

```
from mrjob.job import MRJob
import re

class MRWordCount(MRJob):

    def mapper(self, key, line):
        # strip punctuation and extra spaces
        line = re.sub(r'  ',' ',
                  re.sub(r'[^0-9A-Za-z]',' ',line))
        for word in line.split():
            yield word.lower(), 1

    def combiner(self, word, occurrences):
        yield word, sum(occurrences)

    def reducer(self, word, occurrences):
        yield word, sum(occurrences)


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

Compute Mean Basic V1

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)

1: class Reducer
2:     method Reduce(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

Can we just copy / past reducer code as combiner?

Compute Mean Basic V2

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)

1: class Combiner
2:     method Combine(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         Emit(string t, pair (sum, cnt))          ▷ Separate sum and count

1: class Reducer
2:     method Reduce(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

Is this ok?

Compute Mean Basic V3

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, pair (r, 1))

1: class Combiner
2:     method Combine(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         Emit(string t, pair (sum, cnt))

1: class Reducer
2:     method Reduce(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         r_avg ← sum/cnt
9:         Emit(string t, pair (r_avg, cnt))
```

How about now?