# Introduction to Map Reduce

Continuum Analytics

CONTINUUM
ANALYTICS

## Large Relative to What?

- Can’t fit into Excel

## Large Relative to What?

- Can’t fit into Excel
  - Increase Memory

# Large Relative to What?

- Can’t fit into Excel
    - Increase Memory
- Can’t fit into R

## Large Relative to What?

- Can’t fit into Excel
  - Increase Memory
- Can’t fit into R
  - Increase Memory

## Large Relative to What?

- Can’t fit into Excel
  - Increase Memory
- Can’t fit into R
  - Increase Memory
- Can’t fit into Memory
  - Increase Memory

## Large Relative to What?

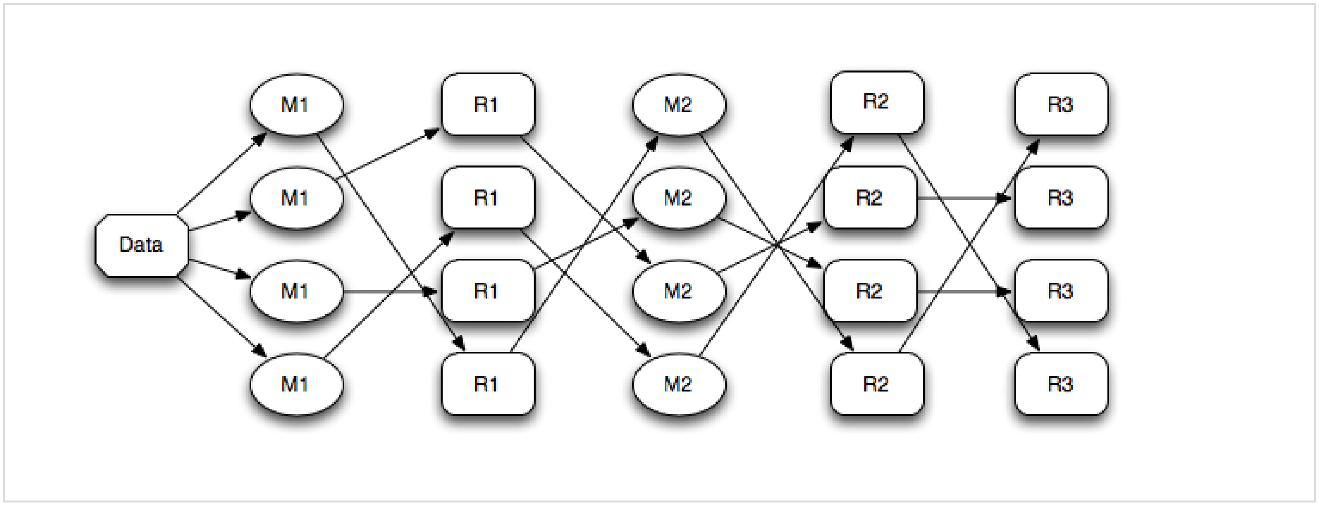
- Can’t fit into Excel
  - Increase Memory
- Can’t fit into R
  - Increase Memory
- Can’t fit into Memory
  - Increase Memory
- Can’t fit on a single disk
  - Distributed Filesystem: SAN, HDFS/DDFS, AWS: S3, Redshift, etc.

# MapReduce

Framework to help solve the problem of distributed computation for distributed data

- A mass of data: records
- Split/**Map** records into key-values pairs
- Collect/Partition kv pairs (Optional Sort)
- Buckets are passed to **Reduce** function
- Result is returned

# MapReduce Workflow



- Push Code to Data
- Lots of Network Traffic

# MR Implementations

- Disco: Python + Erlang
  - Distributed FileSystem: DDFS

- Hadoop: Java
  - Streaming with Python
  - Dumbo
  - MRJob
  - Hadoopy

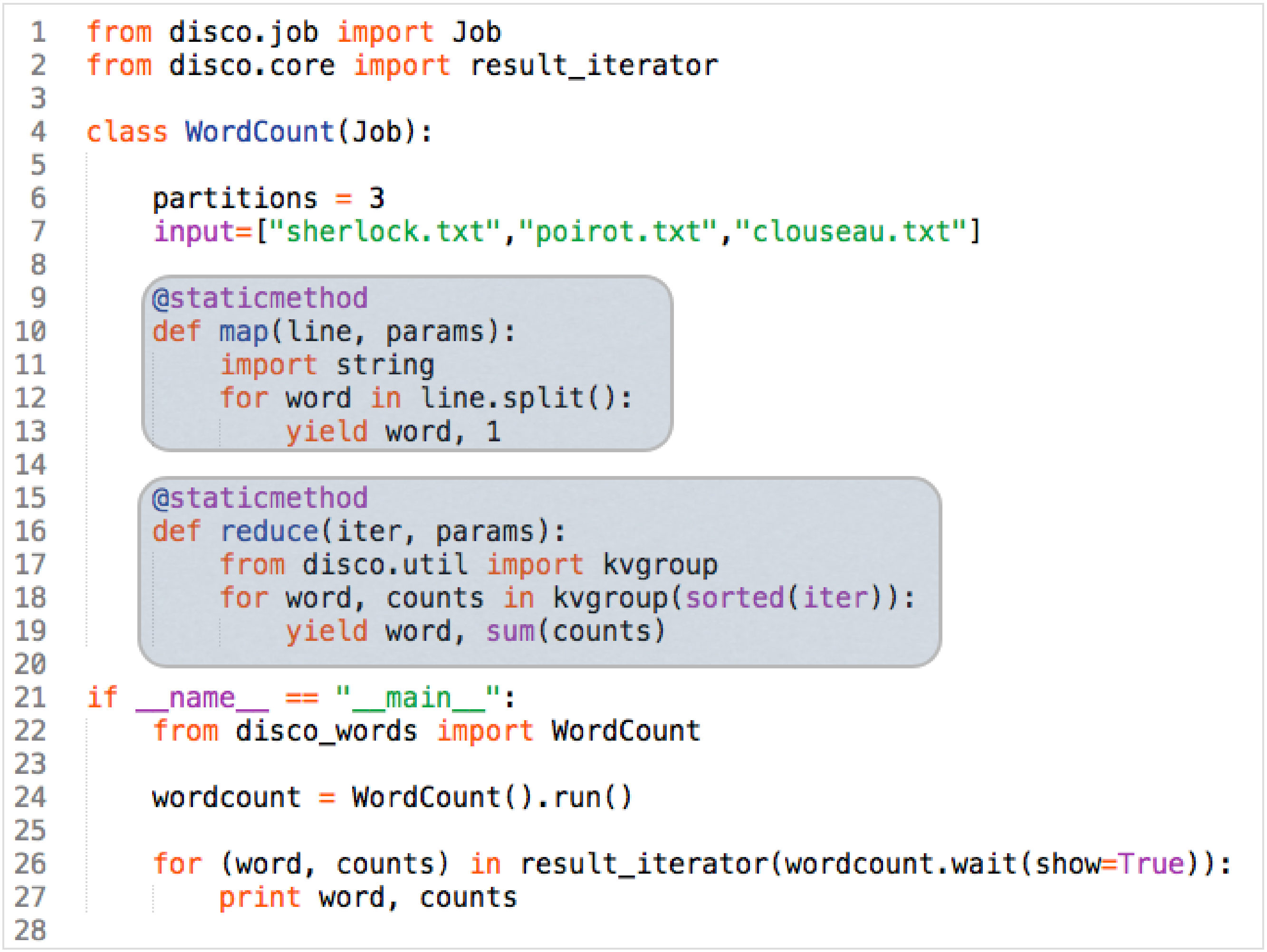
# MapReduce: It's a Party

CONTINUUM ANALYTICS

## Buddies Included

- NumPy
- SciPy
- pandas
- scikits-learn
- OpenCV
- ...

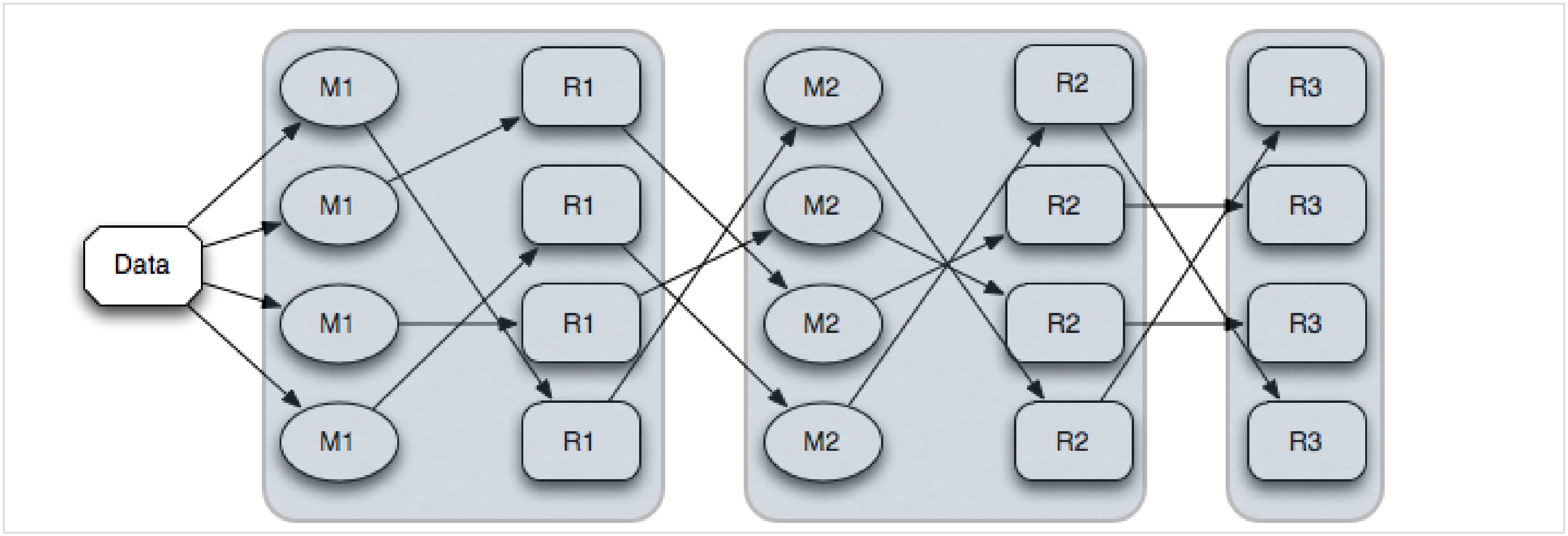
## Canonical Example



```python
from disco.job import Job
from disco.core import result_iterator

class WordCount(Job):

    partitions = 3
    input=["sherlock.txt","poirot.txt","clouseau.txt"]

    @staticmethod
    def map(line, params):
        import string
        for word in line.split():
            yield word, 1

    @staticmethod
    def reduce(iter, params):
        from disco.util import kvgroup
        for word, counts in kvgroup(sorted(iter)):
            yield word, sum(counts)

if __name__ == "__main__":
    from disco_words import WordCount

    wordcount = WordCount().run()

    for (word, counts) in result_iterator(wordcount.wait(show=True)):
        print word, counts

```

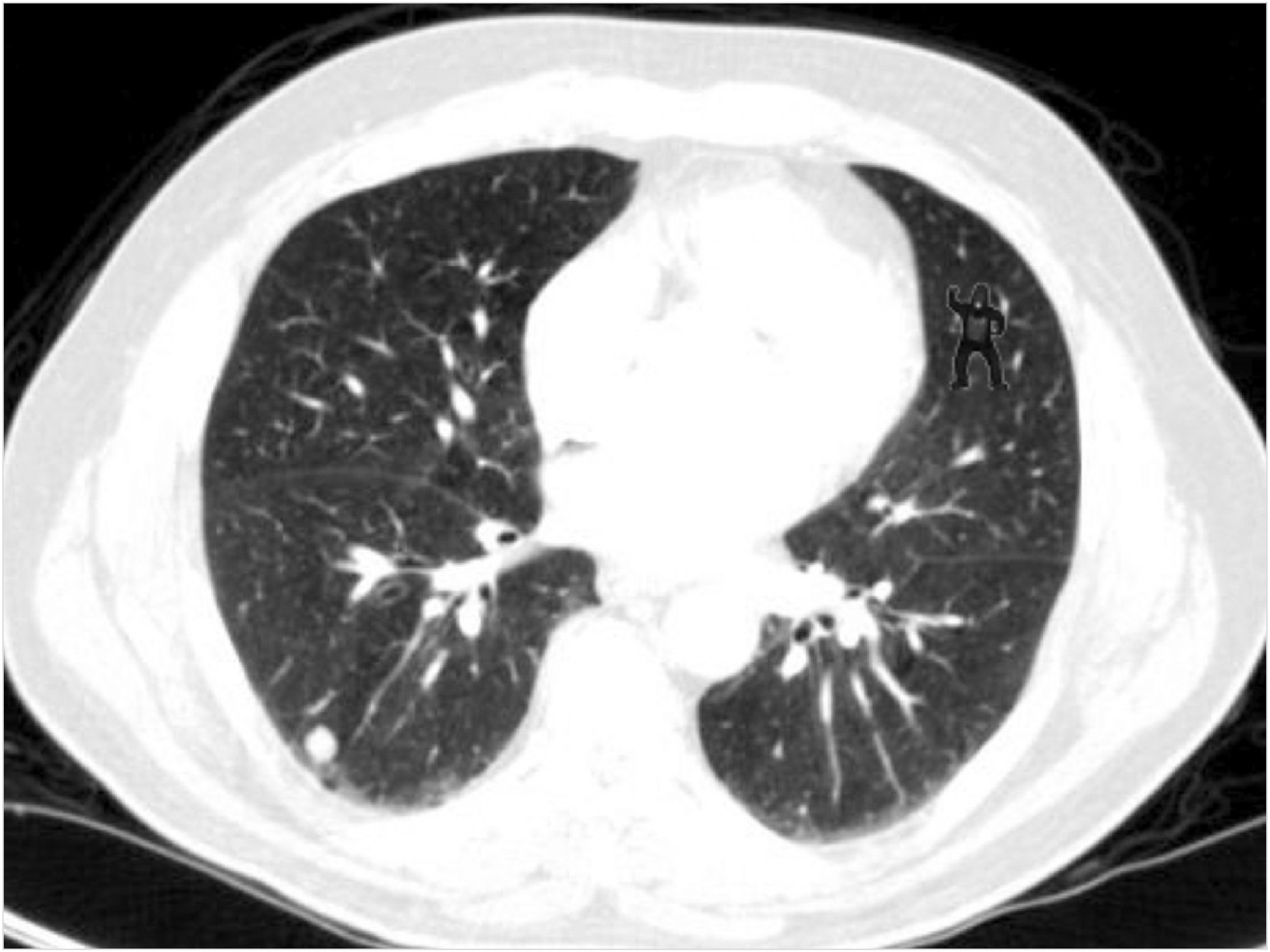
## Demo 1

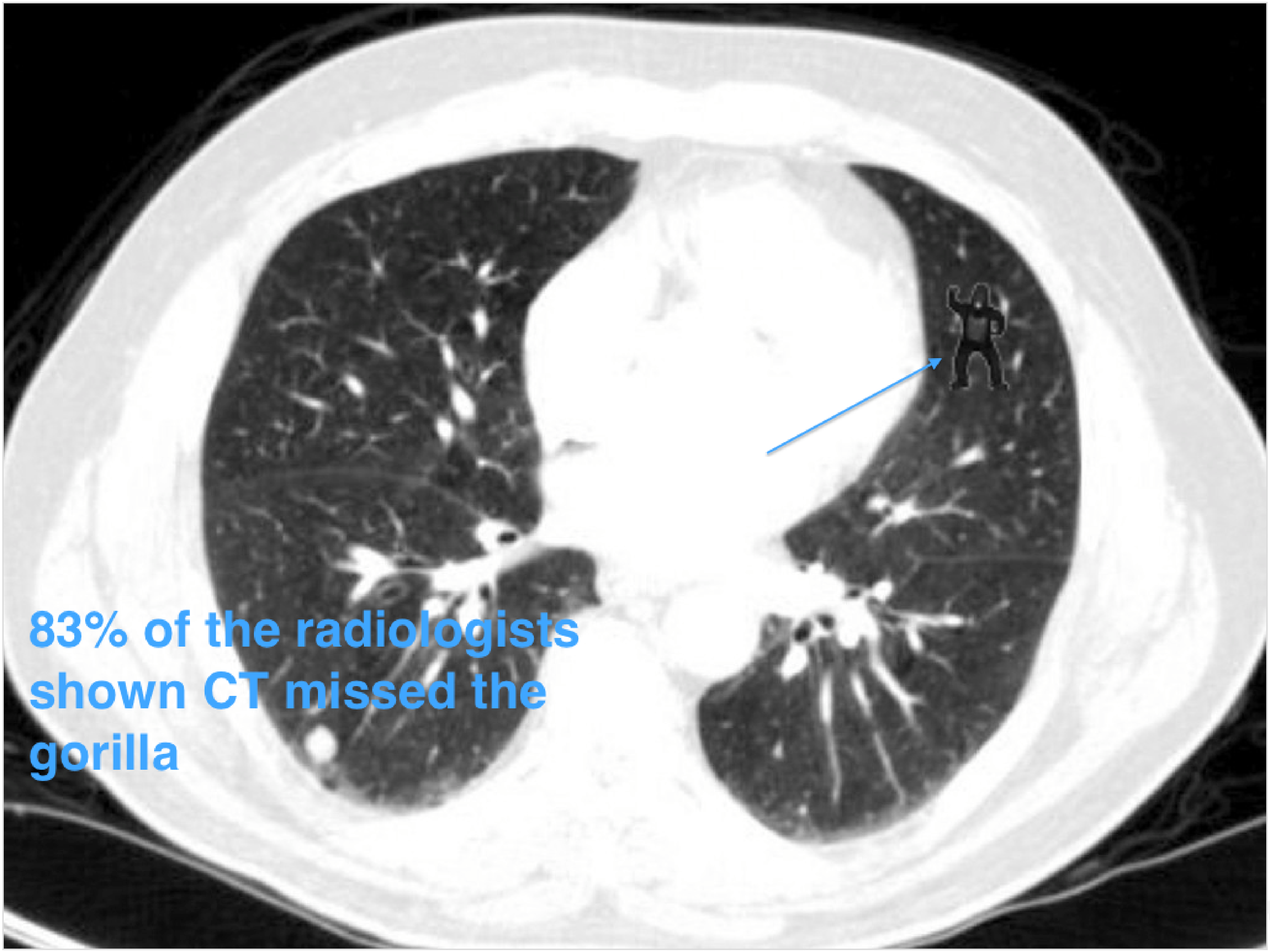
## Chaining Jobs



# MapReduce Thoughts

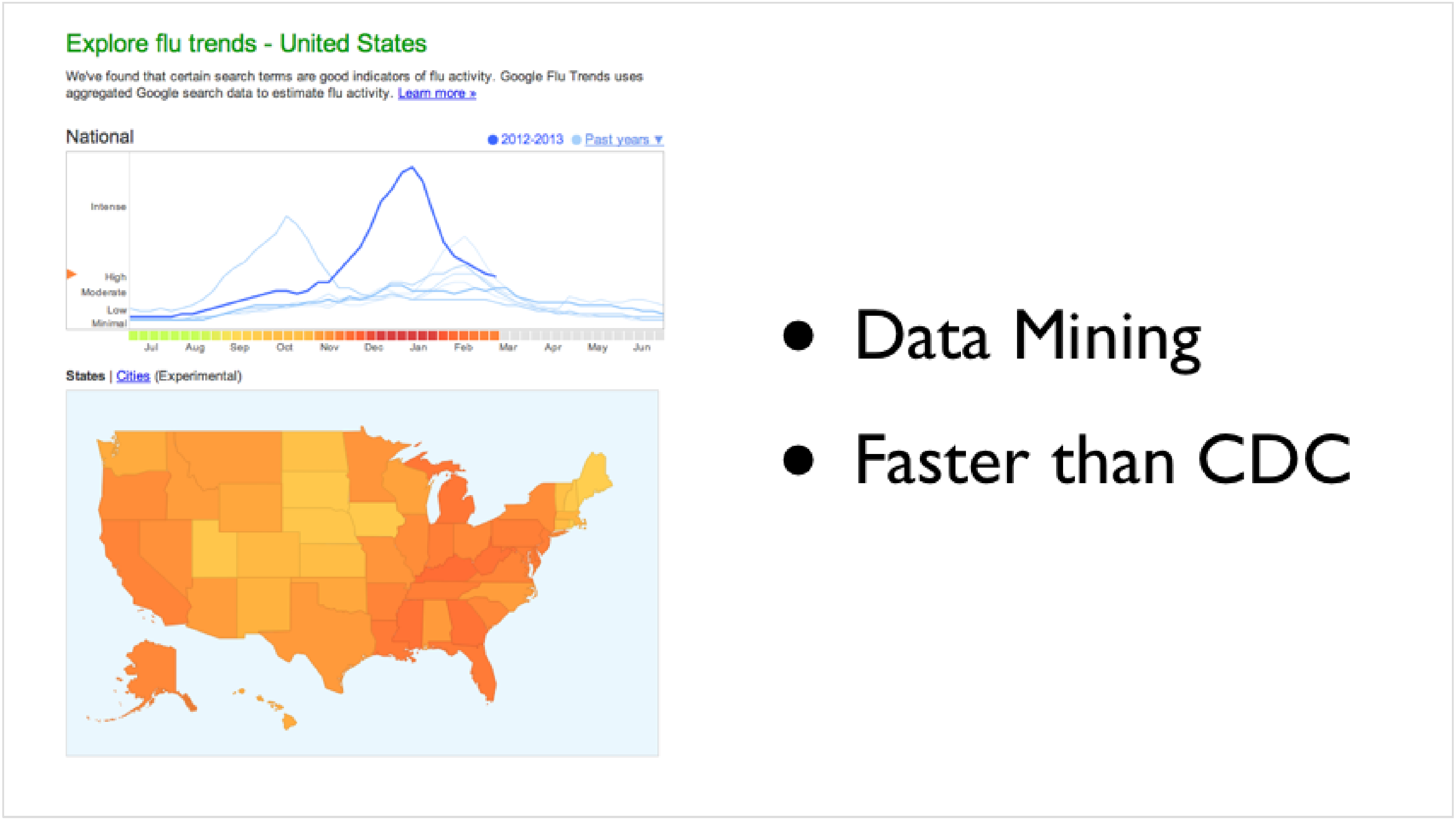
- Data Cleansing
  - Everyone’s pain point
- Task Deconstruction
  - Good for code management
  - Hides -- in a good way -- data management
- Can Be Inefficient
  - Network traffic
  - Job organization


M



83% of the radiologists shown CT missed the gorilla

## Google Flu

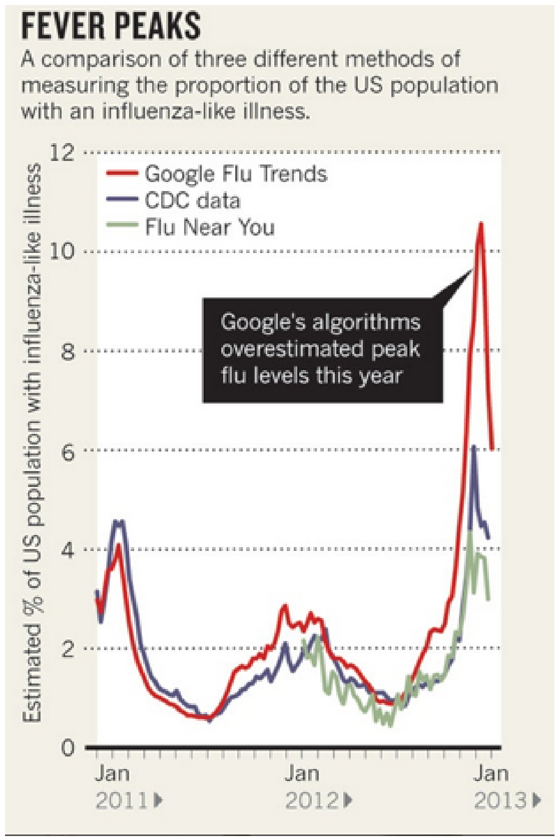

Explore flu trends - United States

We've found that certain search terms are good indicators of flu activity. Google Flu Trends uses aggregated Google search data to estimate flu activity. Learn more »

National

- Data Mining
- Faster than CDC

## Google Get's It Wrong



- Typically, prediction is great!
- This year not so much
- Google: No comment!
- Feedback mechanism from hype-up media

## Data Philosophy

- Invisible Gorillas will stay Invisible
  - Inattentional Blindess
- Machine Learning without Oversight
  - Turnkey analytics is dangerous
- Good Analysis
  - Requires iterative exploration
  - Peer review and collaboration

## Canonical Example

```
class WordCount(Job):
    partitions = 3
    input=["sherlock.txt","poirot.txt","clouseau.txt"]

    @staticmethod
    def map(line, params):
        import string
        for word in line.split():
            yield word, 1

    @staticmethod
    def reduce(iter, params):
        from disco.util import kvgroup
        for word, counts in kvgroup(sorted(iter)):
            yield word, sum(counts)

if __name__ == "__main__":
    from count_words import WordCount

    for (word, counts) in result_iterator(WordCount.wait(show=True)):
        print word, counts
```