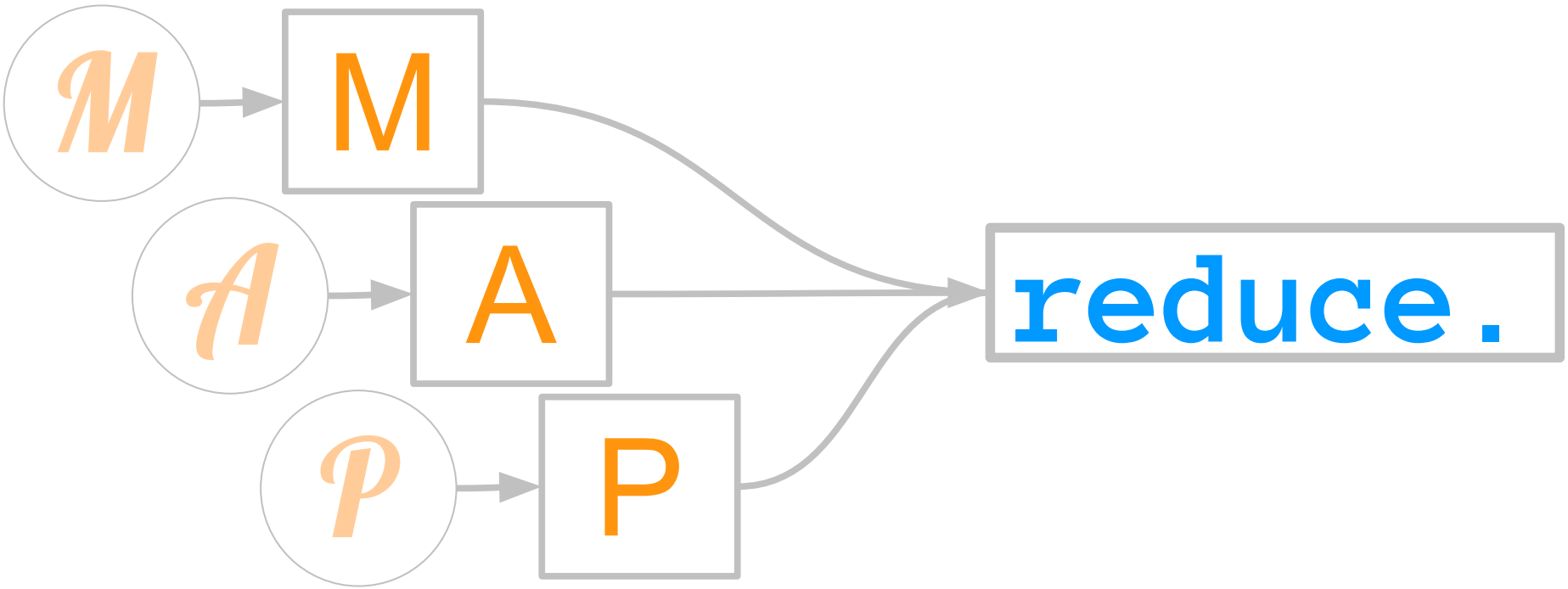
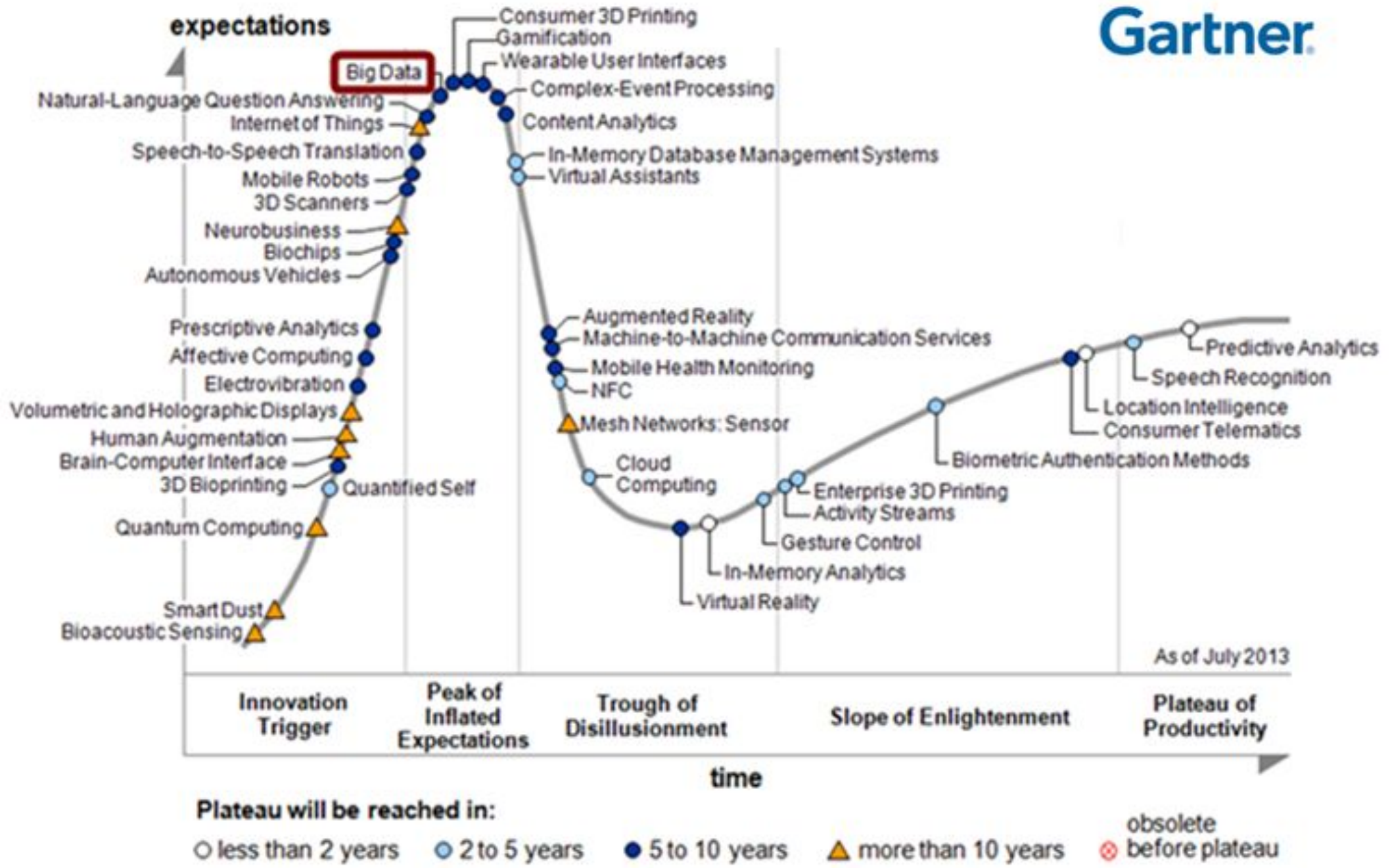



Kimberly Wilber

Sep 4, 2018

# “Hype Cycle for Emerging Technologies (2013).”



# OpenLife.cc

Open Life: The Philosophy of Open Source

Home » Blogs » hingo's blog

## Terabytes is not big data, petabytes is

Submitted by **hingo** on Fri, 2011-01-07 22:12

**datamining** | **MySQL** | **NoSQL**

I often wonder what's behind the increased trend behind Hadoop and other NoSQL technologies. I realize if you're Yahoo that such technology makes sense. I don't get why everyone else wants to use it.

Reading **Stephen O'Grady's self-review of his predictions**

Look guys... **Terabytes is not Big Data!** (And Gigabytes never was big data, even Excel can now take that amount of data :-)

**DevOps Borat**
@DEVOPS_BORAT

Follow

# Big Data is any thing which is crash Excel.

Reply Retweet Favorite ••• More

| RETWEETS | FAVORITES |
| --- | --- |
| 1,977 | 423 |

9:25 AM - 8 Jan 2013

# This lecture: Show off two tools for working with large datasets

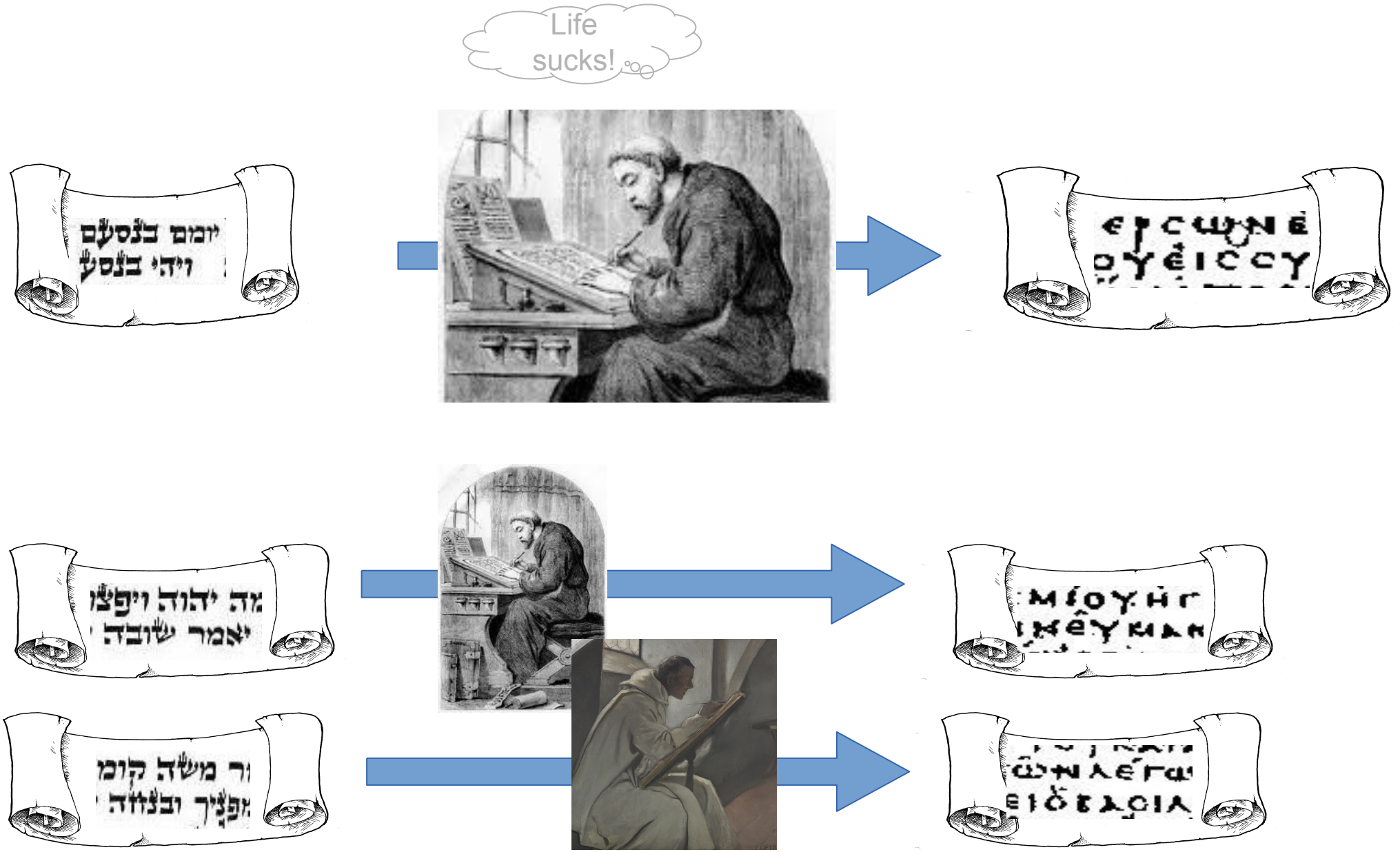
- **Multiprocessing and parallelism**
  - Chop into pieces
- **Code demo**
- **MapReduce**
  - Add a reduction step
- **When** is it appropriate to use each?

# Multiprocessing since ancient times

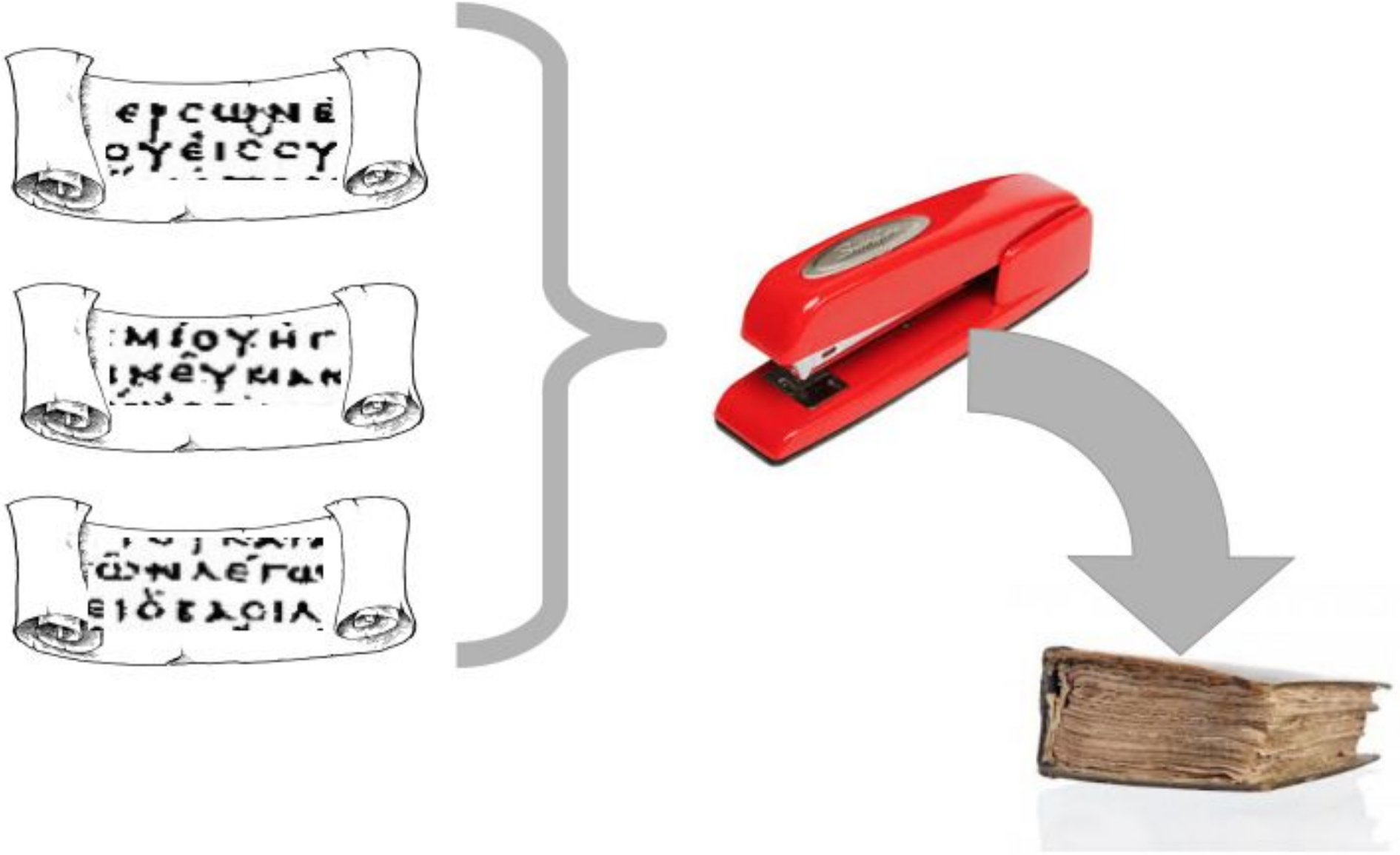
**Septuagint**: Translating the Old Testament from Hebrew to Greek in 2nd century BCE
Legend: 72 scholars completed the translation in seventy days.

# Step 1: Partition

# Step 2: Map



# Step 3: Reduce/Aggregate



# When does this work?

Tasks that are **“embarrassingly parallel”** are easy to split up.

This means there should be **no dependencies** between tasks.

- Computing statistics on different slices of the dataset
- Running experiments with different parameters each time

# How to use multiprocessing?

There are several good multiprocessing libraries for Python:

- The **`multiprocessing`** library
- IPython Parallel

# Code demo

https://gist.github.com/gcr/a25d6a0810185a806623b051024673dc

# Large datasets need more than ordinary parallelization!

Adapted to problems at Internet scale

> "A typical MapReduce computation processes many terabytes of data on thousands of machines." (2004)

- Involving **many machines** (network transfer…)
- **Recovering from failures** is important
- **Aggregating results** is tricky
- **Storing data** becomes a problem

"MapReduce: Simplified Data Processing on Large Clusters," J. Dean and S. Ghemawat, OSDI 2004

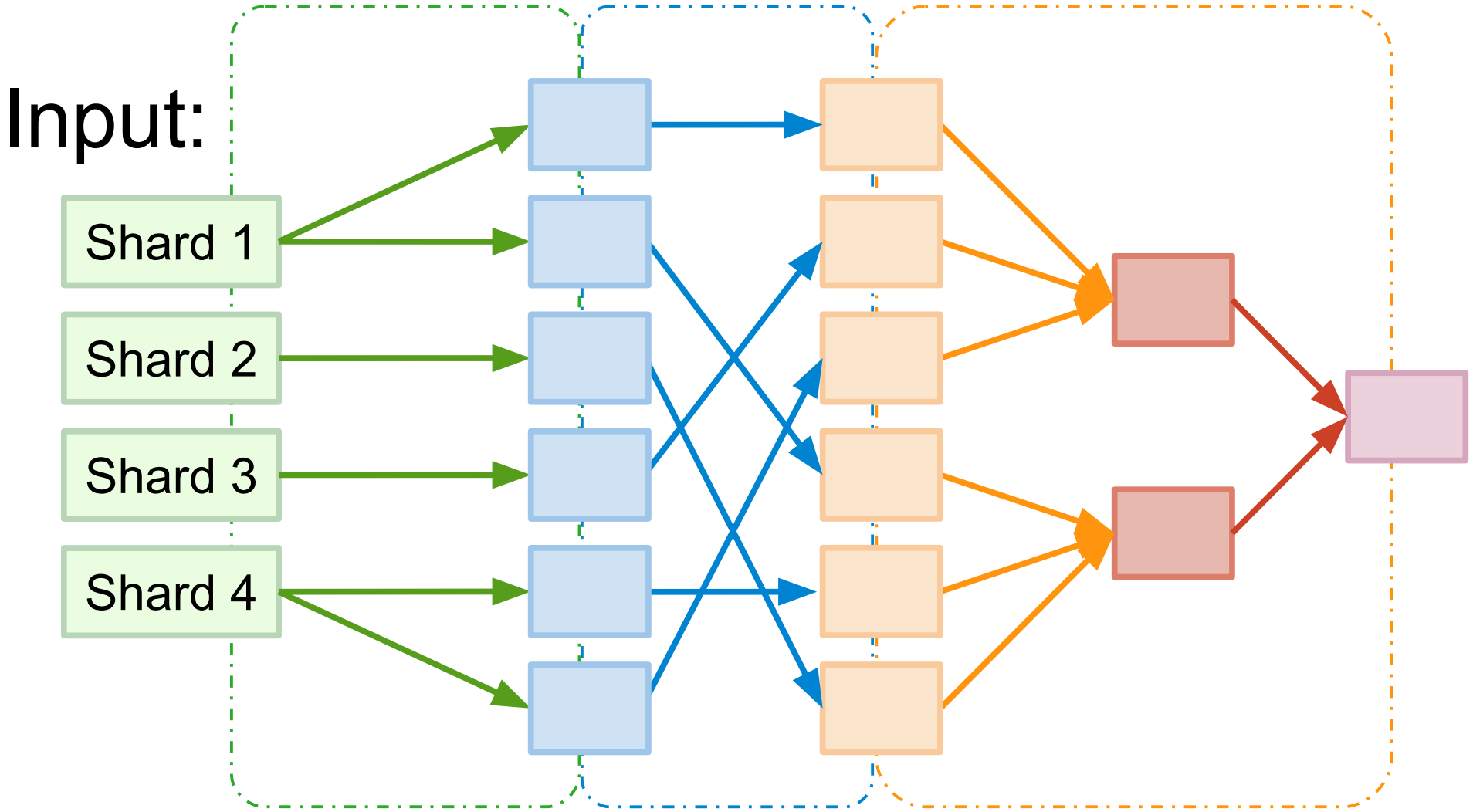
# “MapReduce”

Multiprocessing is great, but what if you need even more flexibility?

Full-blown **MapReduce** pipelines have several steps:

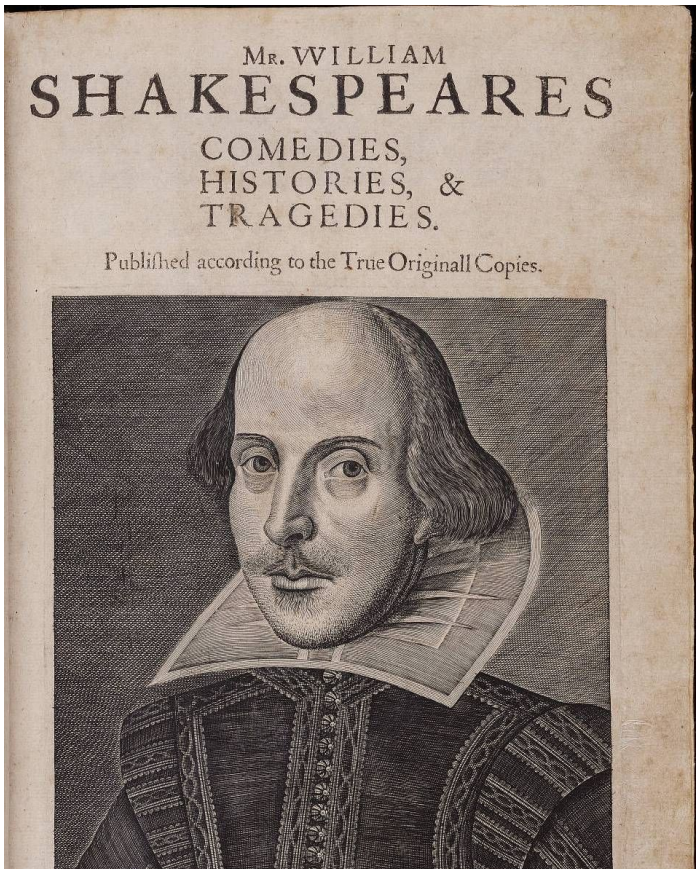
- **Partition / Split** the data into key/values
- **Map** each datum
- **Sort** the results
- **Reduce** / aggregate the results into the final output

# Formal definition



# Example: What are the most frequent words in Shakespeare's plays?

Until very recently, this problem required **multiple man-years** to solve!




Barnardo. Who's there?
Fran. Nay answer me: Stand & unfold your selfe
Bar. Long live the King
Fran. Barnardo?
Bar. He
Fran. You come most carefully upon your houre
Bar. 'Tis now strook twelve, get thee to bed Francisco

# Step 1: define the “mapper”

```
def mapper(line):
    for word in line.split():
        emit(word, 1)
```

```
mapper(“Tis now strook twelve…”)
    {“Tis”: 1}
    {“Now”: 1}
    {“Strook”: 1}
    …
mapper(“Gee, what’s this ghost doing here?”)
    {“Gee”: 1}
    {“What’s”: 1}
    …
```

# Step 1: define the “mapper”

```
def mapper(line):
    for word in line.split():
        emit(word.lower(), 1)
```

```
mapper(“Tis now strook twelve…”)
    {“tis”: 1}
    {“now”: 1}
    {“strook”: 1}
    …
mapper(“Gee, what’s this ghost doing here?”)
    {“gee”: 1}
    {“what’s”: 1}
    …
```

# Step 2: Sorting

Provided by the framework -- you don't write it!

The sorting step aggregates all results with the same key **together** into a single list.

```
{"tis": 1}
{"now": 1}
{"strook": 1}
{"the": 1}
{"twelve": 1}
{"romeo": 1}
{"the": 1}
…
```

```
{"tis":     [1,1,1,1,1]}
{"now":     [1,1,1]}
{"strook":  [1,1]}
{"the":     [1,1,1,1,1,1,1,1,1,1,
{"twelve":  [1,1]}
{"romeo":   [1,1,1,1,1,1,1]}
{"juliet":  [1,1,1,1,1,1,1]}
…
```

# Step 3: define the “reducer”

Aggregates all the results together.

```
def reducer(word, counts):
    emit(word, sum(counts))
```

```
reducer(“tis”, [1,1,1,1,1])
    {“tis”: 5}
reducer(“the”, [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
    {“the”: 23590}
reducer(“strook”, [1,1])
    {“strook”: 2}
    ...
```

# Step 3: define the “reducer”

This is incorrect. Can you see why?

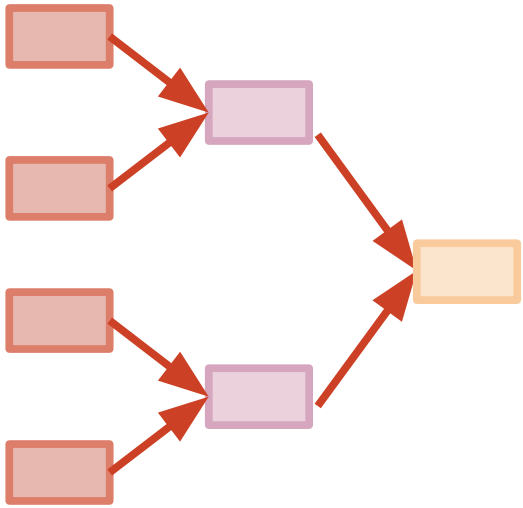
```
def reducer(word, counts):
    emit(word, len(counts))
```

# Step 3: define the “reducer”

This is incorrect. Can you see why?

```
def reducer(word, counts):
    emit(word, len(counts))
```

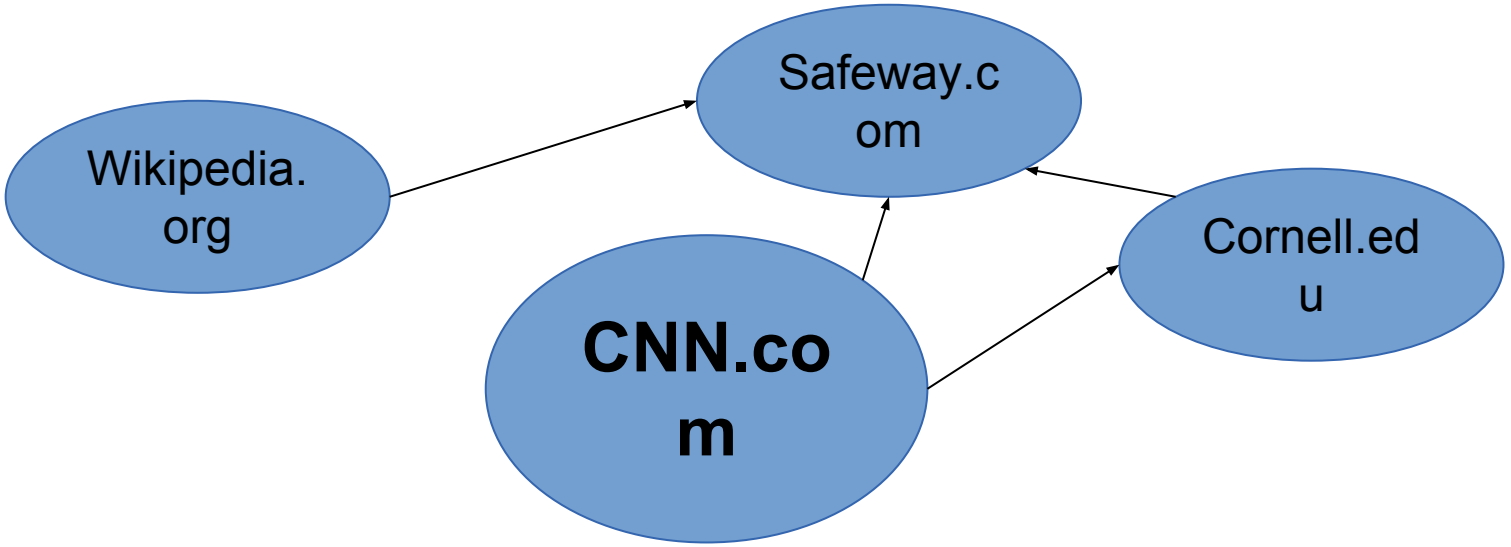
Reducers run in a **tree**, which means reduce output output may be sent **through other reducers**!



```
reducer(“the”, [23590, 10201, 330])
    {“the”: 3} # bad!
```

# Example 2: An iteration of PageRank

How popular is a certain webpage? Determined by the number of pages that link to it, and the quality of those pages




We want to compute a score for each page concurrently

```
def mapper(url):
    webpage = download(url)
    for outgoing_link in webpage.links:
        emit(outgoing_link, url)
```

**In 1:** “cnn.com”

**Out 1:** → {“wikipedia.org/...”: “cnn.com”,
“safeway.com/...”: “cnn.com”,
“olivegarden.com/...”: “cnn.com”}

**In 2:** “wikipedia.org”

**Out 2:** → {“safeway.com/...”: “wikipedia.org”,
“starbucks.com/...”: “wikipedia.org”,
“cornell.edu/...”: “wikipedia.org”}

```
def reducer(dest, sources):
    weight = 0
    for source in sources:
        weight += popularity[source]
    emit(dest, weight)
```

**In 1:**
```
{“starbucks.com”:
 [“cnn.com/Starbucks-out-of-coffee.htm”,
  “wikipedia.org/2014-Coffee-Crisis.htm”,
  “cornell.edu/impact-of-bean-growing.htm”
 ]}
```

**Out 1:**
```
{“starbucks.com”: 10 + 8 + 4 }
```

(This is not quite PageRank, but it's similar)

# What does a **real** implementation do for you?

- **Machine failures:** With so many machines, P(no failures) is quite small
- **Stragglers at the end of the job**: Performance may dependent on the slowest machine. Some implementations resend tasks, etc...
- **Data access and transfer issues**: How can you get TBs of data onto your compute machines?

# Do we need MapReduce?

(...Maybe.)

# Word frequency as a shell script

```
find -iname ‘*.txt’ \
   | xargs cat       \
   | tr ‘ ‘ ‘\n’     \
   | sort            \
   | uniq -c         \
   | sort -n         \
   | tail
```

# Word frequency as a shell script

```
find -iname '*.txt' \      Partition step
  | xargs cat       \      Map step
  | tr ' ' '\n'     \
  | sort            \      Sort step
  | uniq -c         \
  | sort -n         \      Reduce step
  | tail
```

Took a few minutes to write; runs in seconds on a simple laptop.

Why fight for weeks with mapreduce tools?

# Command-line tools can be 235x faster than your Hadoop cluster

Sat 25 January 2014 by Adam Drake

## Introduction

As I was browsing the web and catching up on some sites I visit periodically, I found a cool article from Tom Hayden about using Amazon Elastic Map Reduce (EMR) and mrjob in order to compute some statistics on win/loss ratios for chess games he downloaded from the millionbase archive, and generally have fun with EMR. Since the data volume was only about 1.75GB containing around 2 million chess games, I was skeptical of using Hadoop for the task, but I can understand his goal of learning and having fun with mrjob and EMR

http://fastml.com/the-emperors-new-clothes-distributed-machine-learning/

# Jure Leskovec

## Organizer of “Mining Large Datasets” MOOC


Jure said every grad student is his lab has one of these machines, and that almost every data set of interest fits in RAM. Contemplate that for a moment.

http://fastml.com/the-emperors-new-clothes-distributed-machine-learning/

# C.J. Lin

Principal Author of LibSVM

"In my recent visit to a large company, their people did say that most analytics works are still done on one machine."

# Frank McSherry

Microsoft, coauthor of Naiad

"Lots of people struggle with the complexities of getting big data systems up and running, when they possibly shouldn't be using the systems in the first place. The data sets above are certainly not small (billions of edges), but still run just fine on a laptop. Much faster than the distributed systems, at least."

http://www.frankmcsherry.org/graph/scalability/cost/2015/01/15/COST.html

# Efficient algorithms can be faster

Label propagation to fixed-point (graph connectivity)

| System | cores | twitter_rv | uk_2007_05 |
|---|---|---|---|
| Spark | 128 | 1784s | 8000s+ |
| Giraph | 128 | 200s | 8000s+ |
| GraphLab | 128 | 242s | 714s |
| GraphX | 128 | 251s | 800s |
| Single thread | 1 | 153s | 417s |

# Efficient algorithms can be faster

Label propagation to fixed-point (graph connectivity)

| System | cores | twitter_rv | uk_2007_05 |
|---|---|---|---|
| Single thread (simple) | 1 | 153s | 417s |
| Single thread (smarter) | 1 | 15s | 30s |

# Should we “just split the work,” or do we need all of mapreduce?

**Use the right tool for the job.** Though the formal model is more general, it comes at a heavy cost and is harder to set up.
Most of the data you interact with will not be big enough to justify the work.

- My rule of thumb: Only take the effort if it saves more time than you spend

Sometimes you can also use **better algorithms** to simplify your life. MapReduce feels like brute force.

http://www.bigdatarepublic.com/author.asp?section_id=2840&doc_id=269178

# GraphChi: How a Mac Mini Outperformed a 1,636 Node Hadoop Cluster

**Christian Prokopp**, Data Scientist, Rangespan

10/29/2013

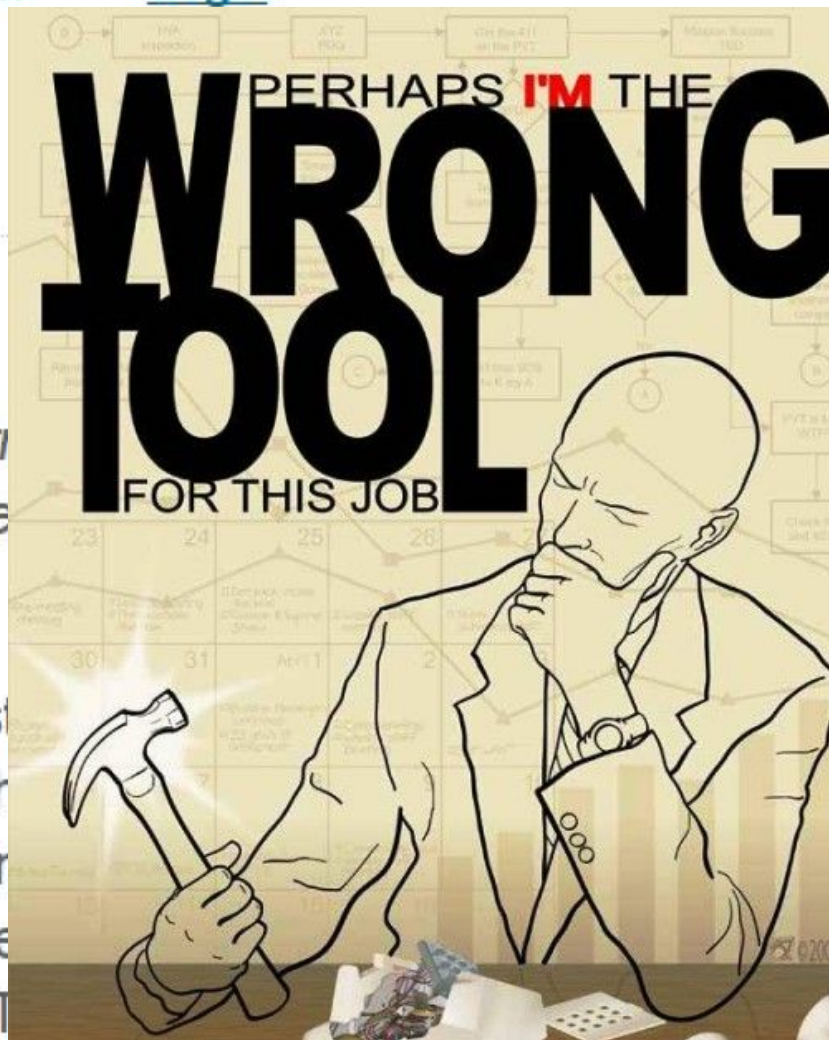
Comment

8 comments

Login

*Last year, GraphChi, a spin-off of GraphLab, a distr graph-based, high-performance computation frame something remarkable.*

GraphChi outperformed a 1,636 node Hadoop clust processing a Twitter graph (dataset from 2010) with edges -- using a single Mac Mini. The task was triar counting and the Hadoop cluster required over seve while GraphChi on the Mac Mini did it in one hour! T



# When does MapReduce shine?

When you **cannot move data around** – the dataset is too big to copy to your local machine

When the output of the map stage is **much smaller than its input**, or when the reduce phase is non-existent

When **fault-tolerance** is important

When **data storage and transfer** is important