Introduction to Big Data with Apache Spark

This Lecture

The Big Data Problem

Hardware for Big Data

Distributing Work

Handling Failures and Slow Machines

Map Reduce and Complex Jobs

Apache Spark

Some Traditional Analysis Tools

- Unix shell commands, Pandas, R

All run on a
single machine!

The Big Data Problem

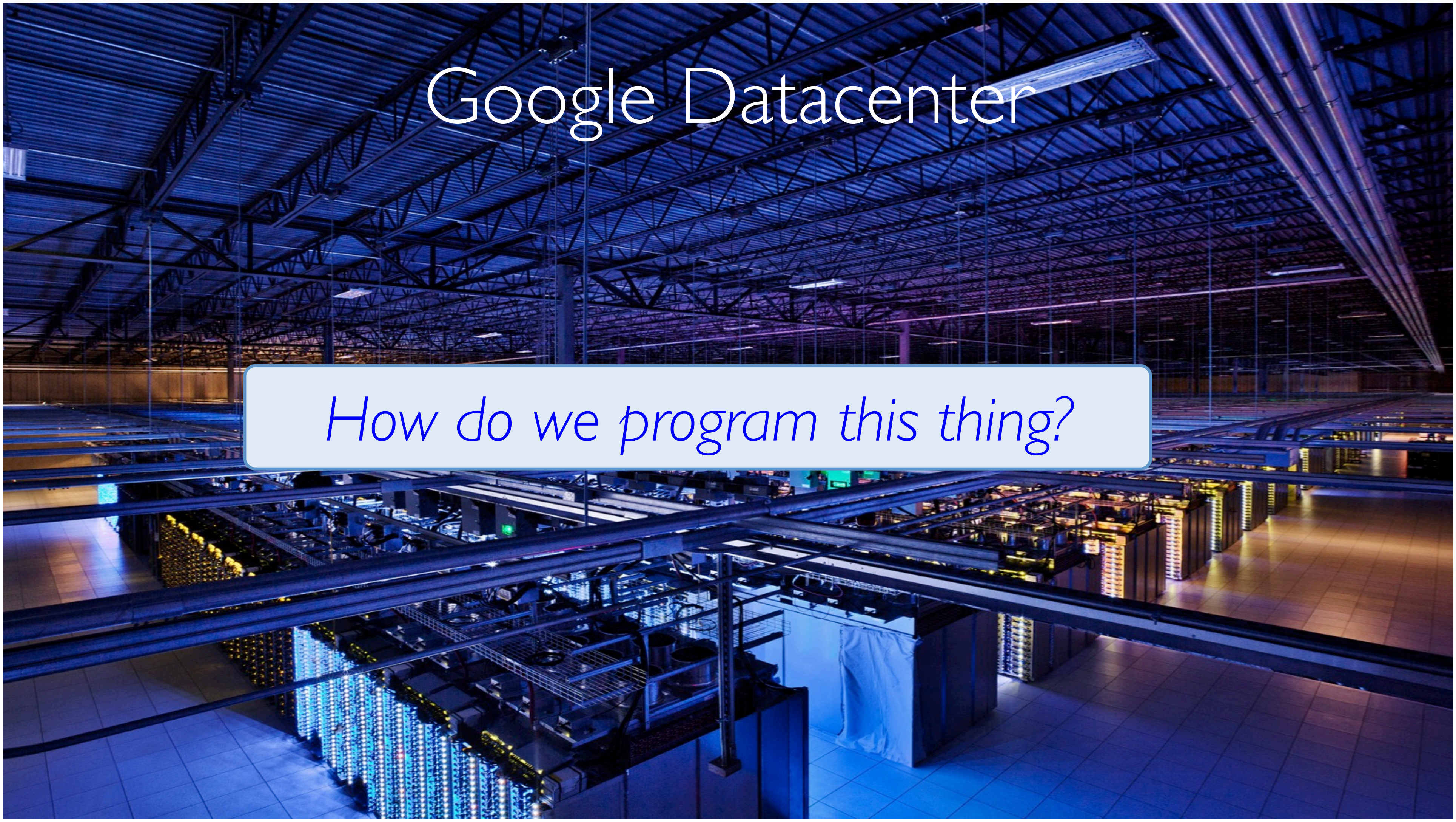
- Data growing faster than computation speeds
- Growing data sources
  » Web, mobile, scientific, …
- Storage getting cheaper
  » Size doubling every 18 months
- But, stalling CPU speeds and storage bottlenecks

Big Data Examples

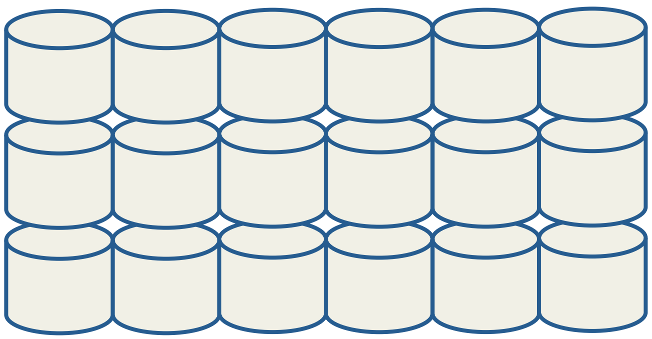
- Facebook's daily logs: 60 TB
- 1,000 genomes project: 200 TB
- Google web index: 10+ PB
- Cost of 1 TB of disk: ~$35
- Time to read 1 TB from disk: 3 hours (100 MB/s)

The Big Data Problem

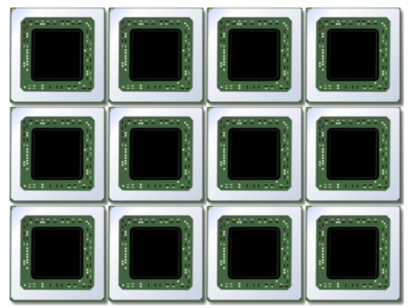
- A single machine can no longer process or even store all the data!
- Only solution is to distribute data over large clusters


Google Datacenter
How do we program this thing?

Hardware for Big Data



Lots of hard drives



… and CPUs

Hardware for Big Data

One big box?
(1990's solution)

But, expensive
- » Low volume
- » All "premium" hardware

And, still not big enough!

Image: Wikimedia Commons / User:Tonusamuel

Hardware for Big Data

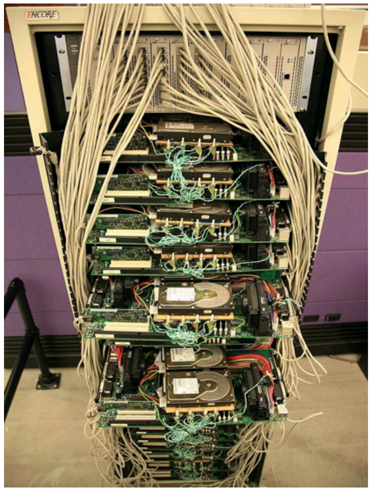


Image: Steve Jurvetson/Flickr

Consumer-grade hardware

- Not "gold plated"

Many desktop-like servers

- Easy to add capacity
- Cheaper per CPU/disk

Complexity in software

Problems with Cheap Hardware

Failures, Google's numbers:

- 1-5% hard drives/year
- 0.2% DIMMs/year

Network speeds versus shared memory

- *Much* more latency
- Network slower than storage

Uneven performance

What's Hard About Cluster Computing?

- How do we split work across machines?

How do you count the number of occurrences of each word in a document?

"I am Sam
I am Sam
Sam I am
Do you like
Green eggs and ham?"

I: 3
am: 3
Sam: 3
do: 1
you: 1
like: 1
…

One Approach: Use a Hash Table

"I am Sam
I am Sam
Sam I am
Do you like
Green eggs and ham?"

{}

One Approach: Use a Hash Table

"$\boxed{\text{I}}$ am Sam
I am Sam
Sam I am
Do you like
Green eggs and ham?"

{I :1}

One Approach: Use a Hash Table

"I $\boxed{\text{am}}$ Sam
I am Sam
Sam I am
Do you like
Green eggs and ham?"

{I: 1,
am: 1}

One Approach: Use a Hash Table

"I am $\boxed{\text{Sam}}$
I am Sam
Sam I am
Do you like
Green eggs and ham?"

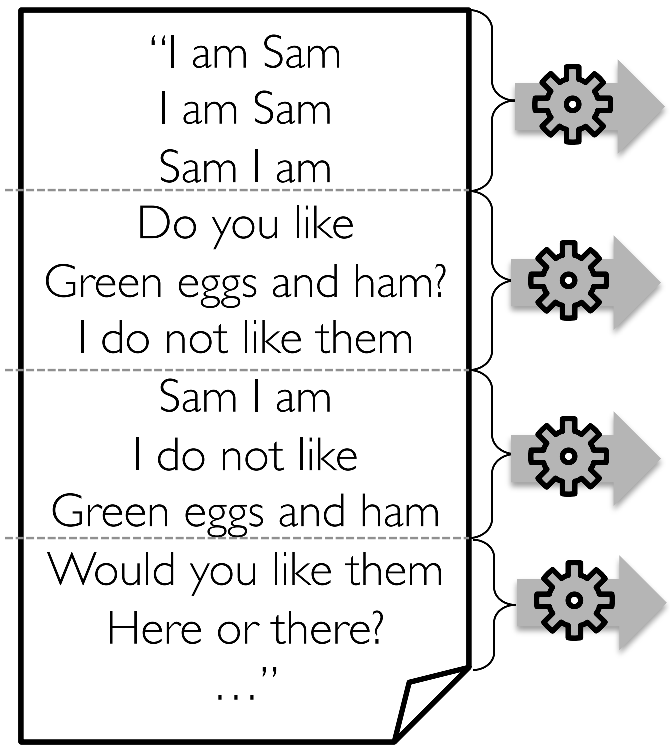
{I: 1,
am: 1,
Sam: 1}

One Approach: Use a Hash Table

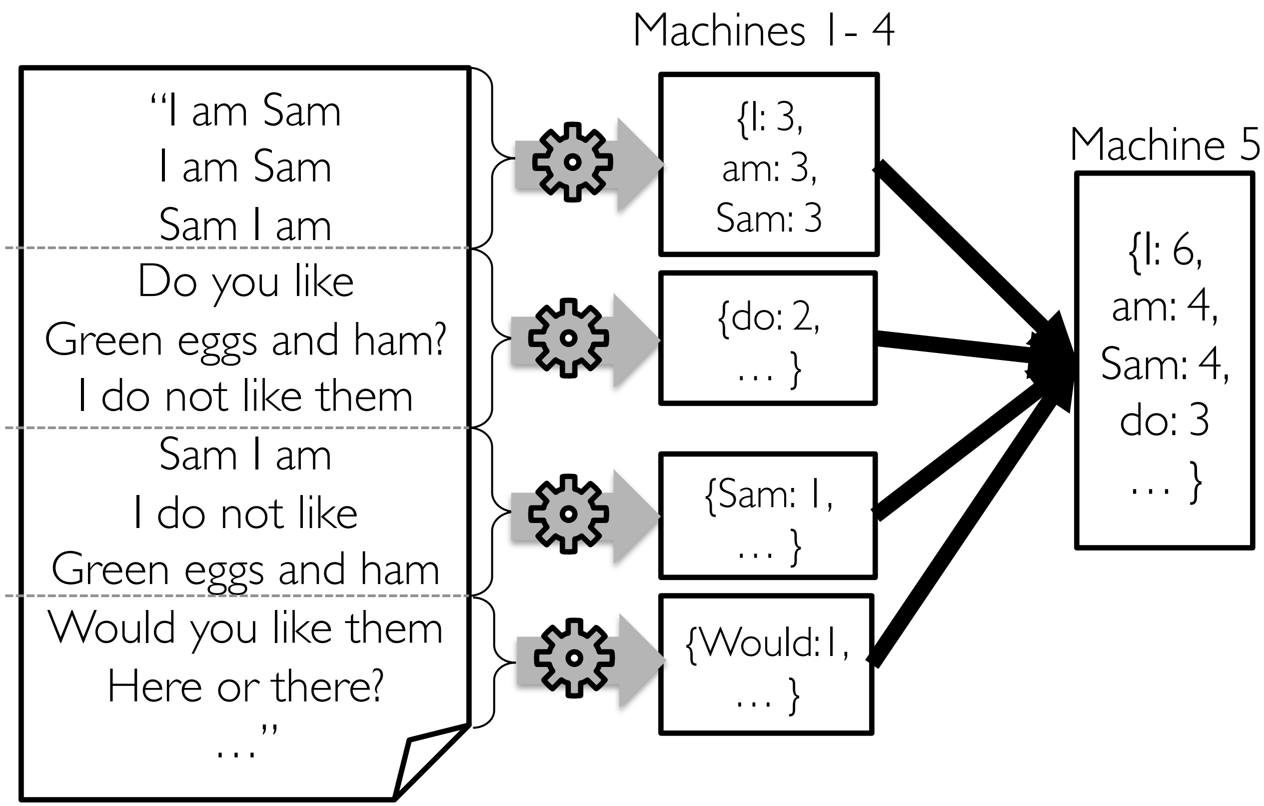
"I am Sam
I am Sam
Sam I am
Do you like
Green eggs and ham?"

{I: 2,
am: 1,
Sam: 1}

What if the Document is Really Big?

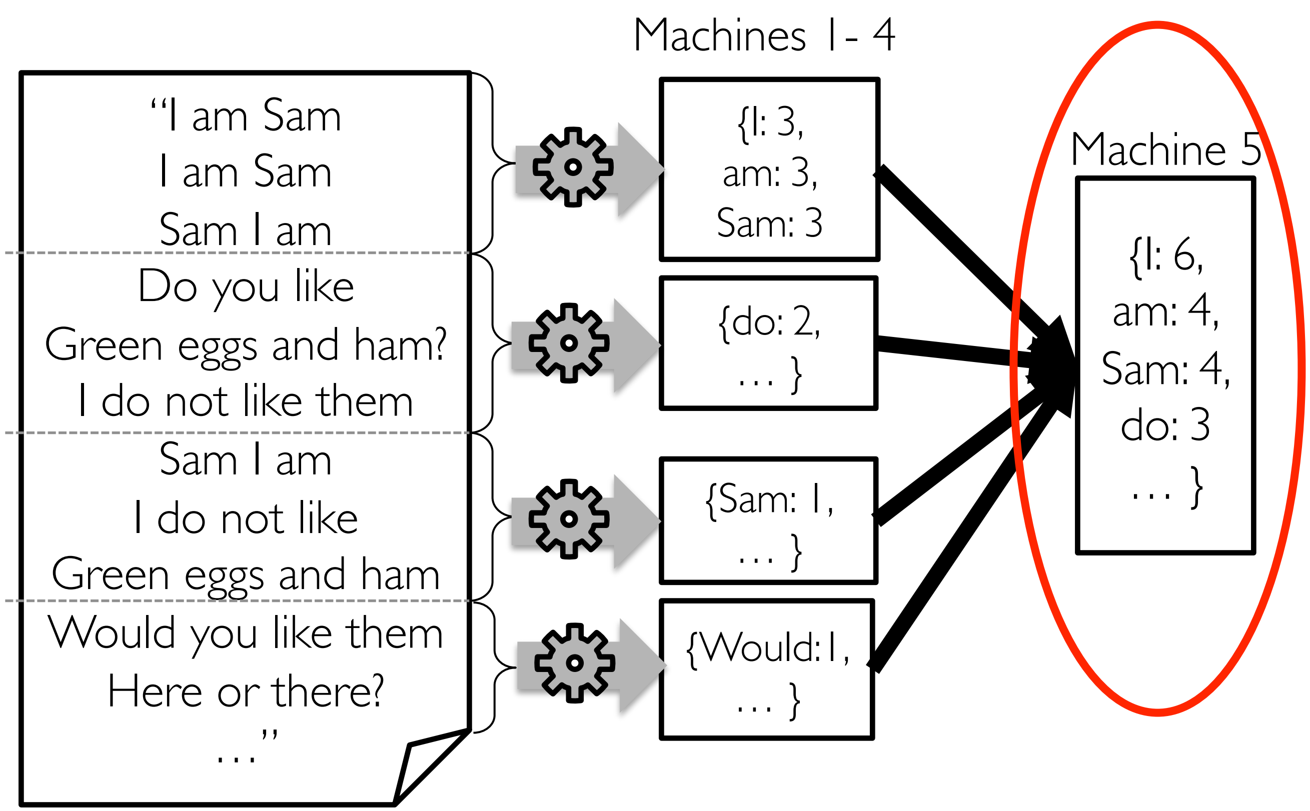


What if the Document is Really Big?



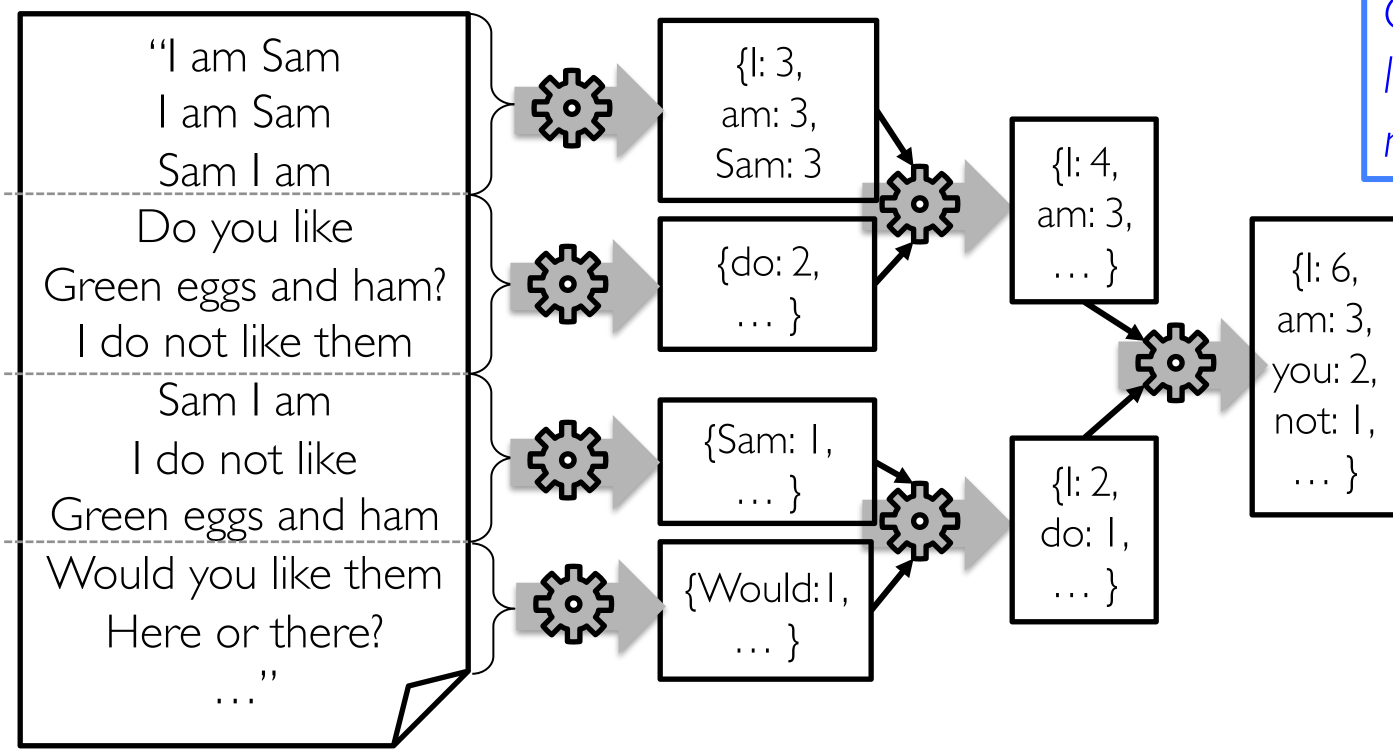
What’s the problem with this approach?

What if the Document is Really Big?



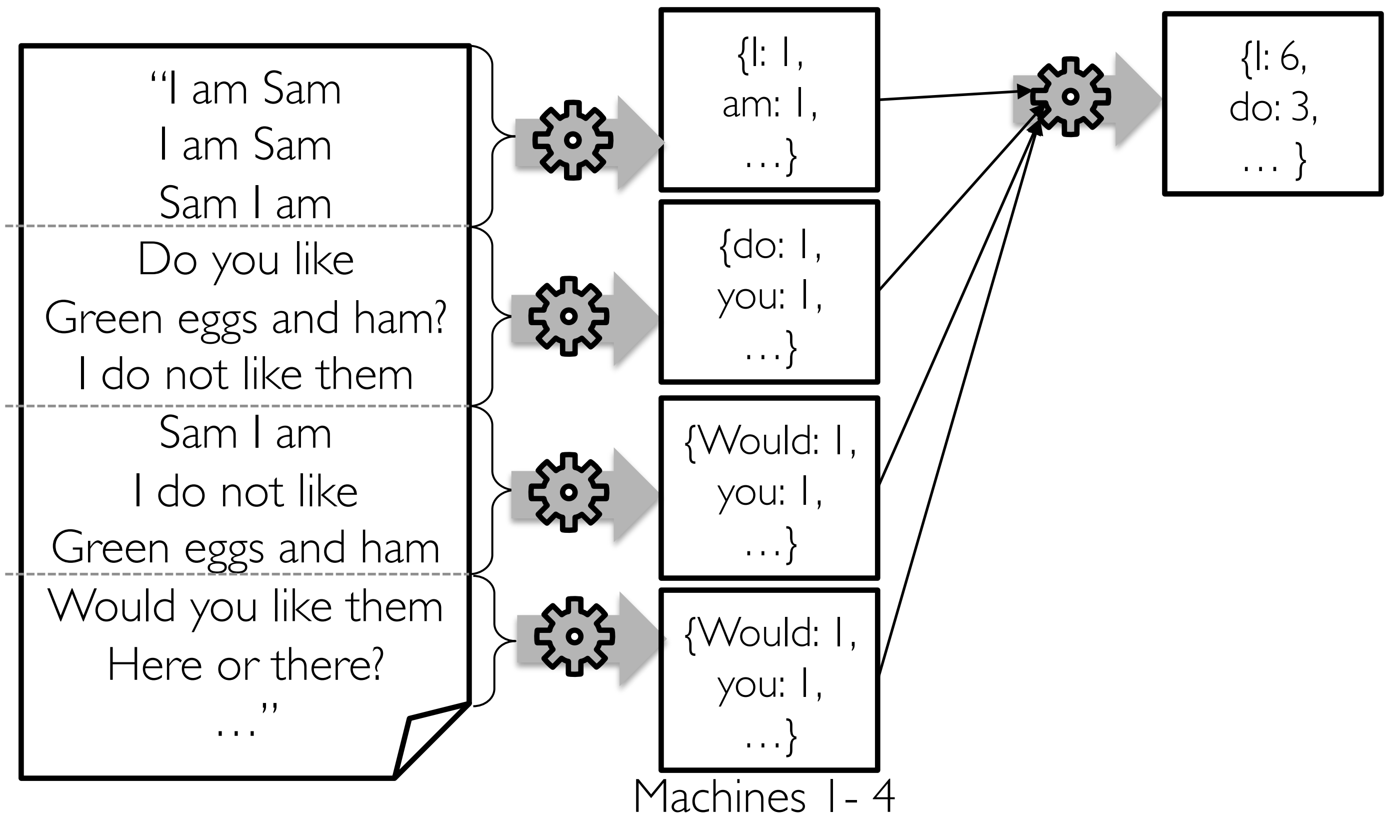
Results have to fit on one machine

What if the Document is Really Big?



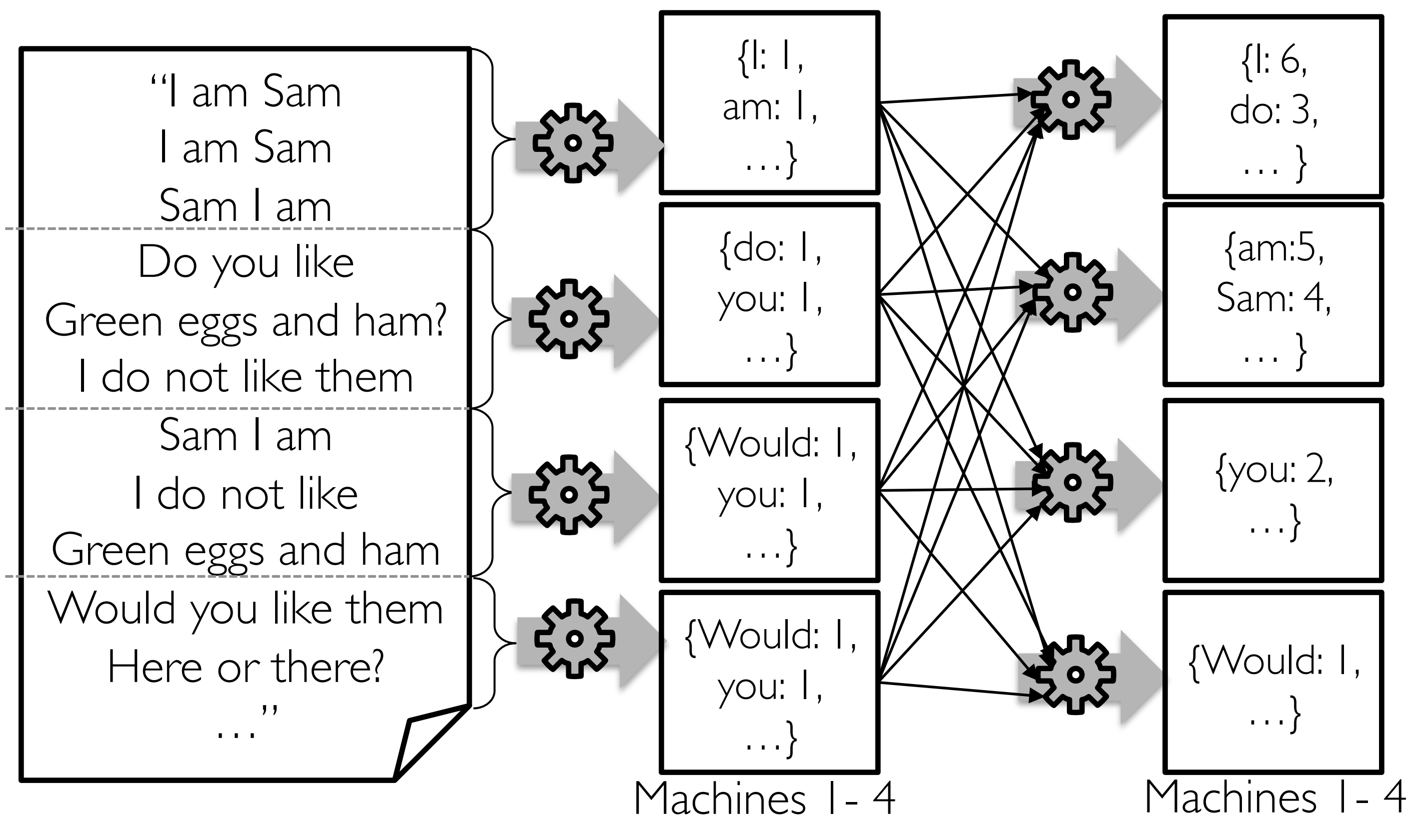
Can add aggregation layers but results still must fit on one machine

What if the Document is Really Big?



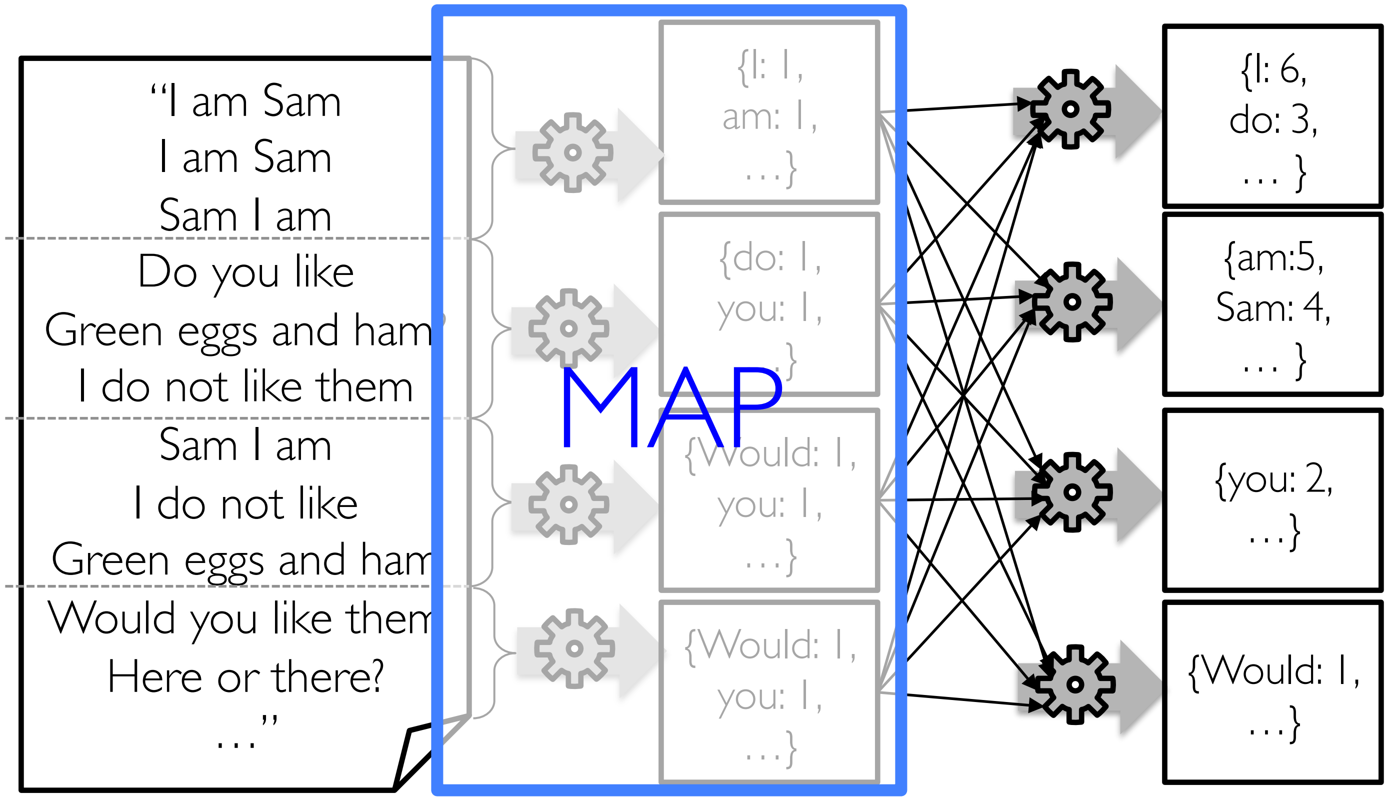
Use Divide and Conquer!!

What if the Document is Really Big?



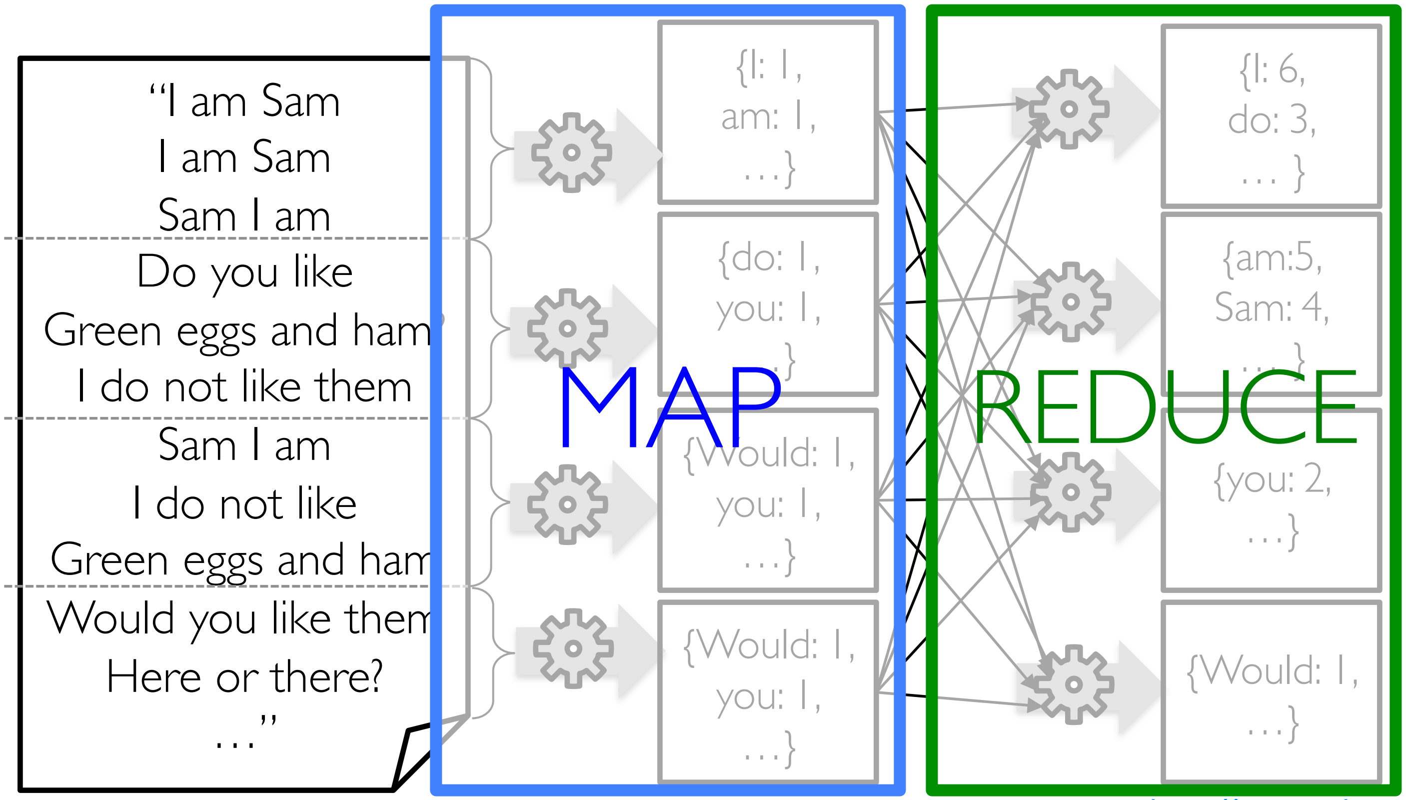
Use Divide and Conquer!!

What if the Document is Really Big?



Use Divide and Conquer!!

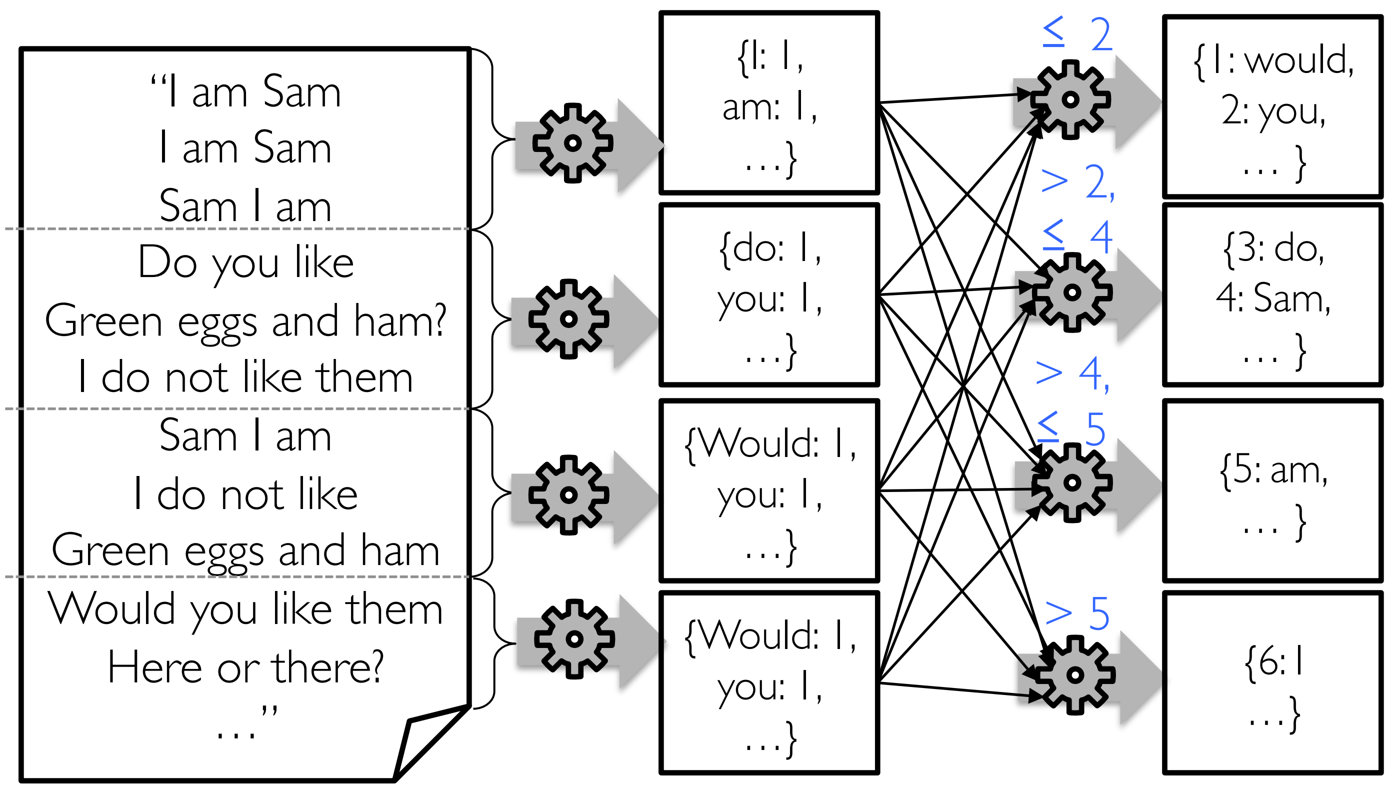
What if the Document is Really Big?



Google
Map Reduce 2004

http://research.google.com/archive/mapreduce.html

Map Reduce for Sorting



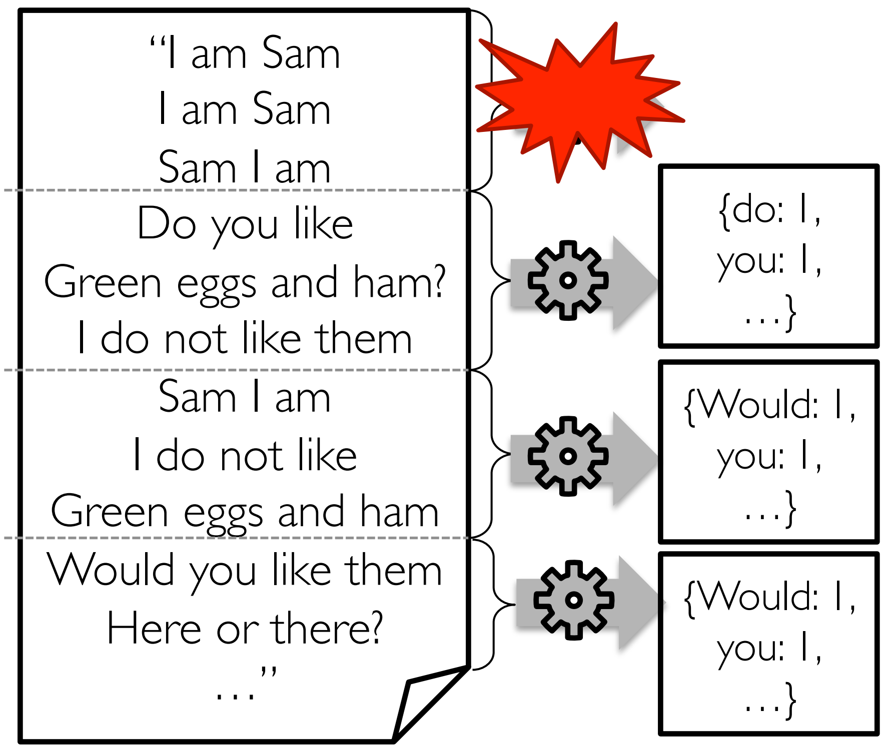
"What word is used most?"

What's Hard About Cluster Computing?

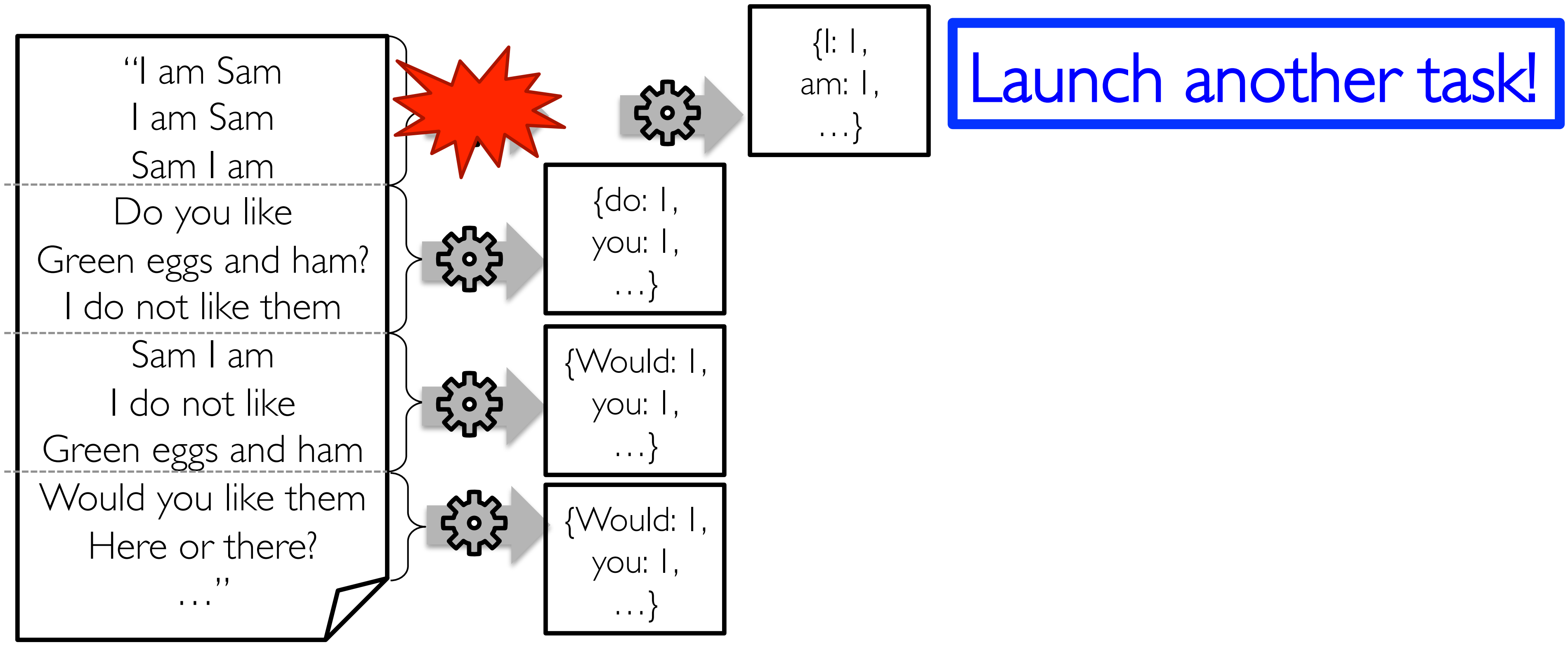
- How to divide work across machines?
  - » Must consider network, data locality
  - » Moving data may be very expensive

- How to deal with failures?
  - » 1 server fails every 3 years ➔ with 10,000 nodes see 10 faults/day
  - » Even worse: stragglers (not failed, but slow nodes)

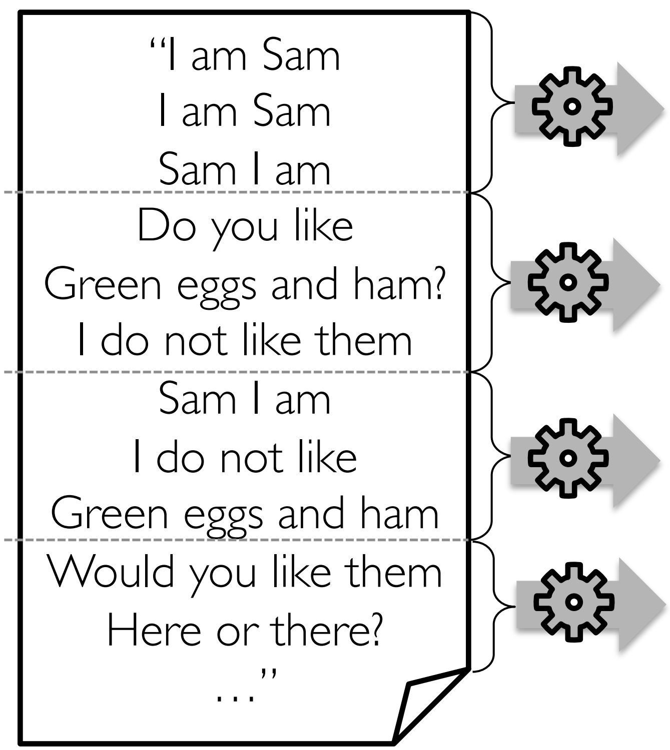
How Do We Deal with Failures?



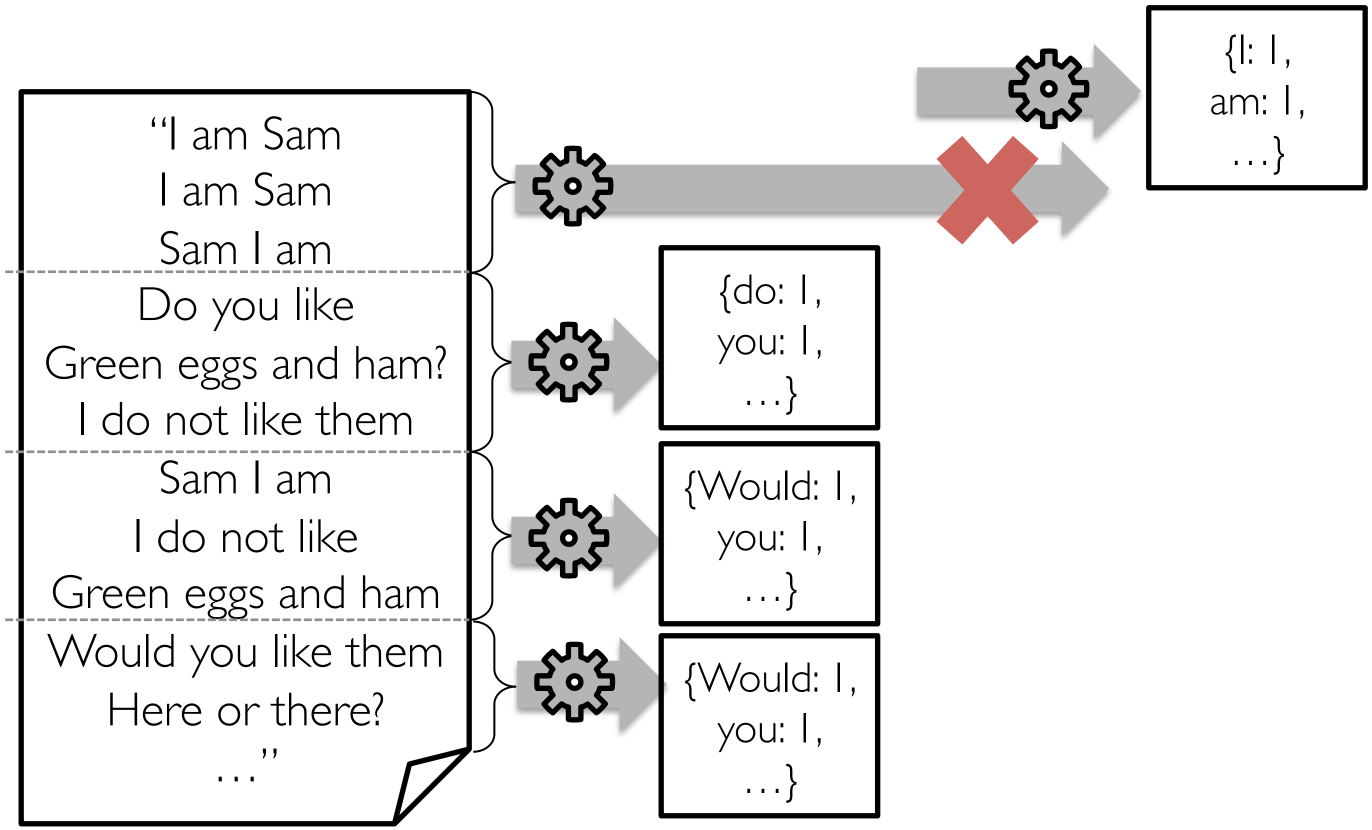
How Do We Deal with Machine Failures?



How Do We Deal with Slow Tasks?

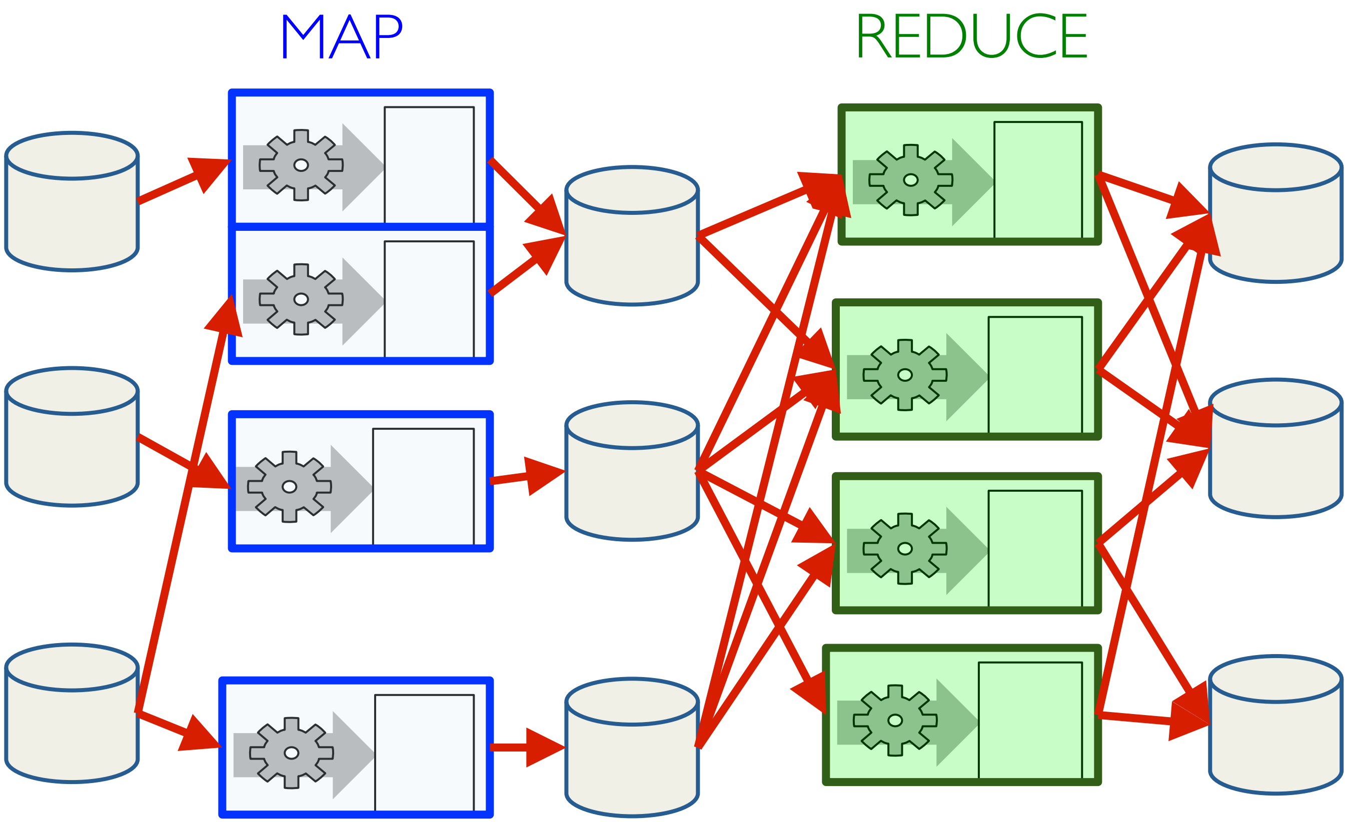


How Do We Deal with Slow Tasks?



Launch another task!

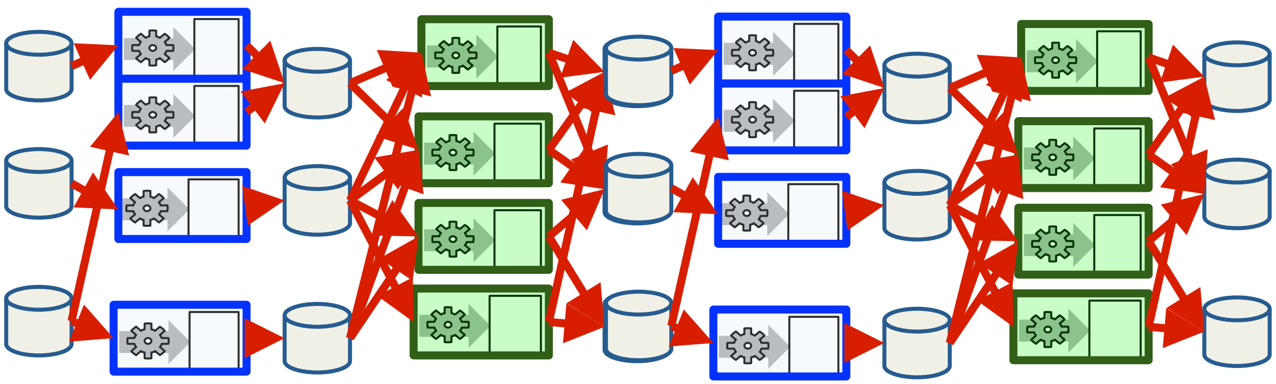
Map Reduce: Distributed Execution



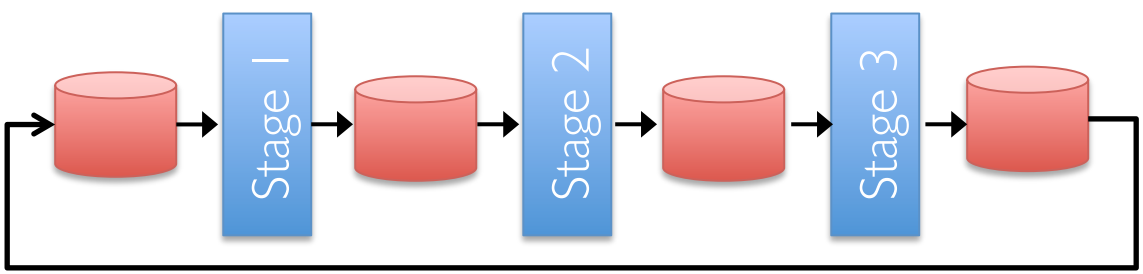
Each stage passes through the hard drives

Map Reduce: Iterative Jobs

- Iterative jobs involve a lot of disk I/O for each repetition

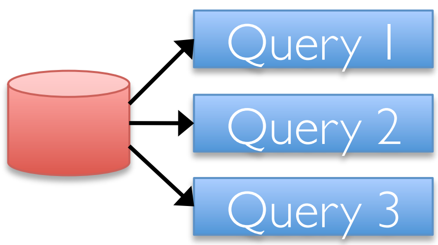


Disk I/O is very slow!

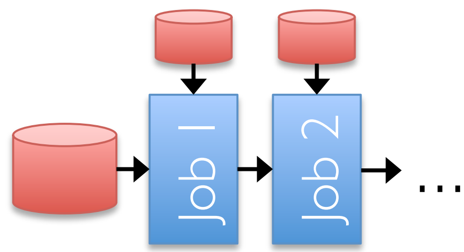


Apache Spark Motivation

- Using Map Reduce for complex jobs, interactive queries and online processing involves *lots of disk I/O*



Interactive mining

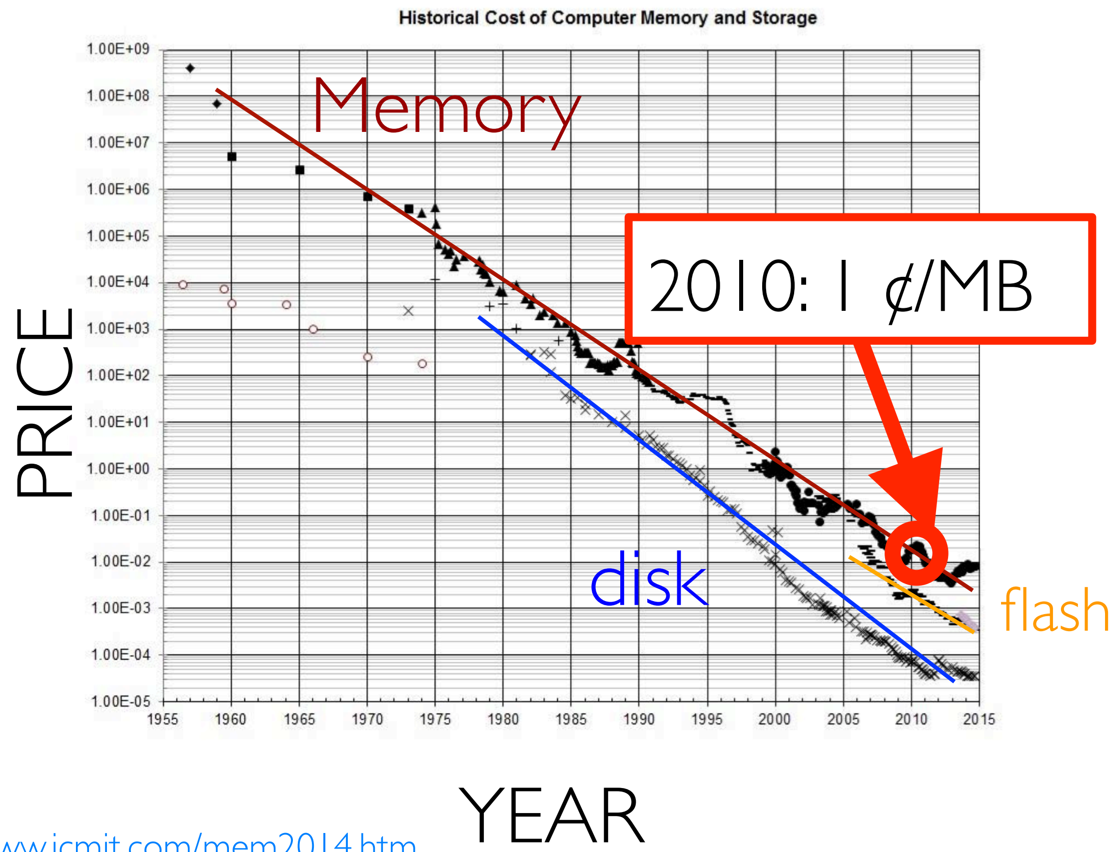


Stream processing

Also, iterative jobs

Disk I/O is very slow

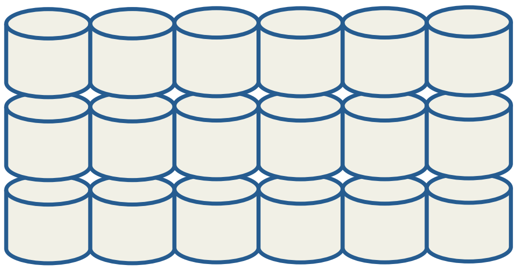
Tech Trend: Cost of Memory



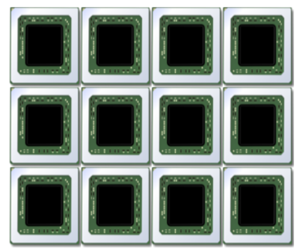
Lower cost means can put more memory in each server

http://www.jcmit.com/mem2014.htm

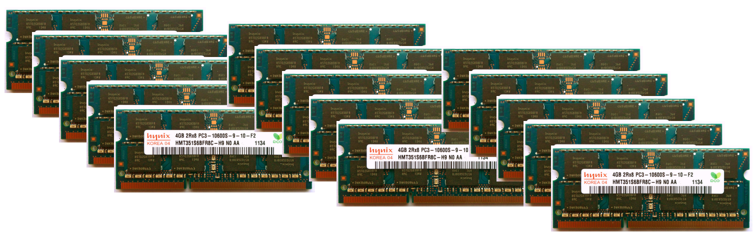
Hardware for Big Data



Lots of hard drives



… and CPUs



… and memory!

Opportunity

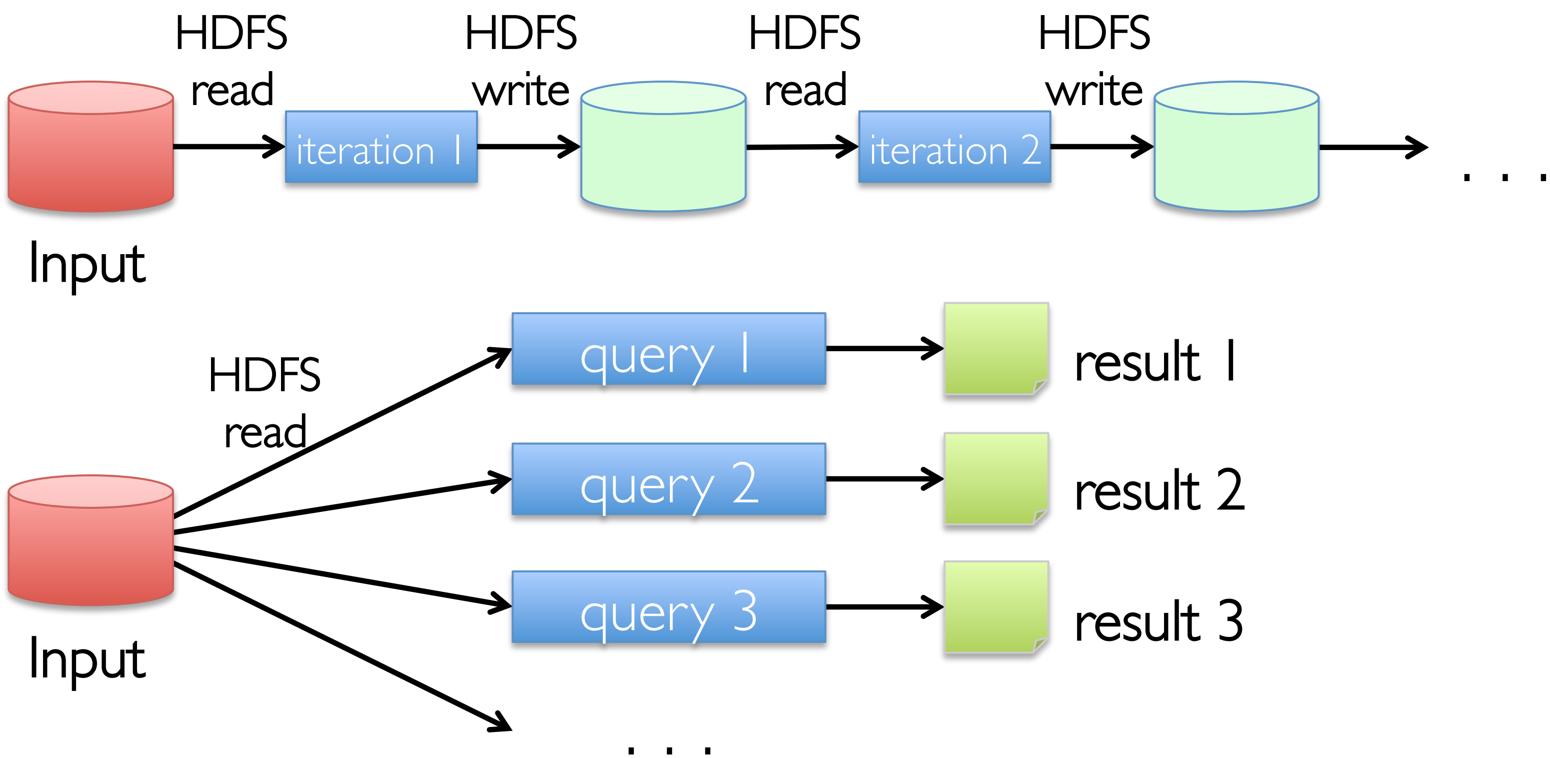
- Keep more data *in-memory*

- Create new distributed execution engine:

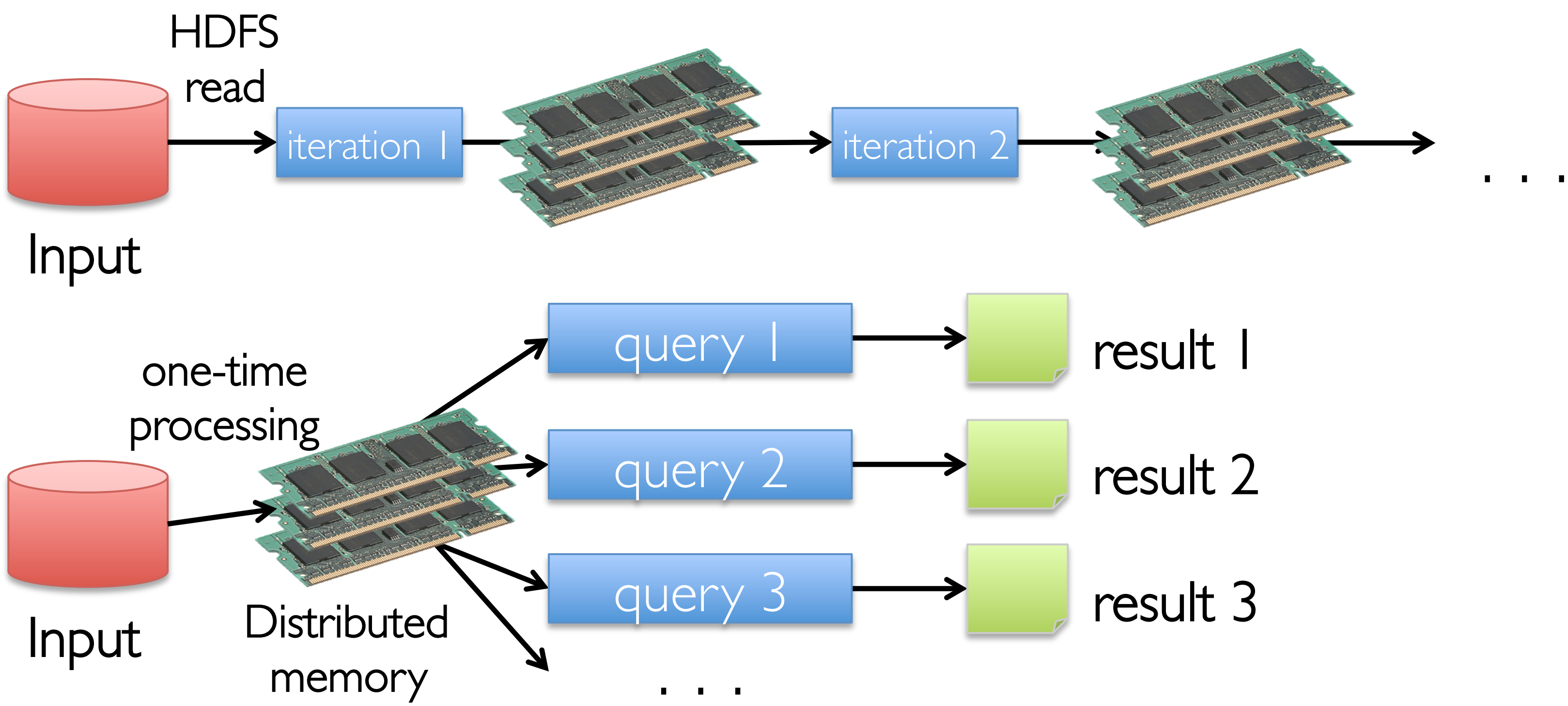
Spark

http://people.csail.mit.edu/matei/papers/2010/hotcloud_spark.pdf

Use Memory Instead of Disk



In-Memory Data Sharing



10-100x faster than network and disk

Resilient Distributed Datasets (RDDs)

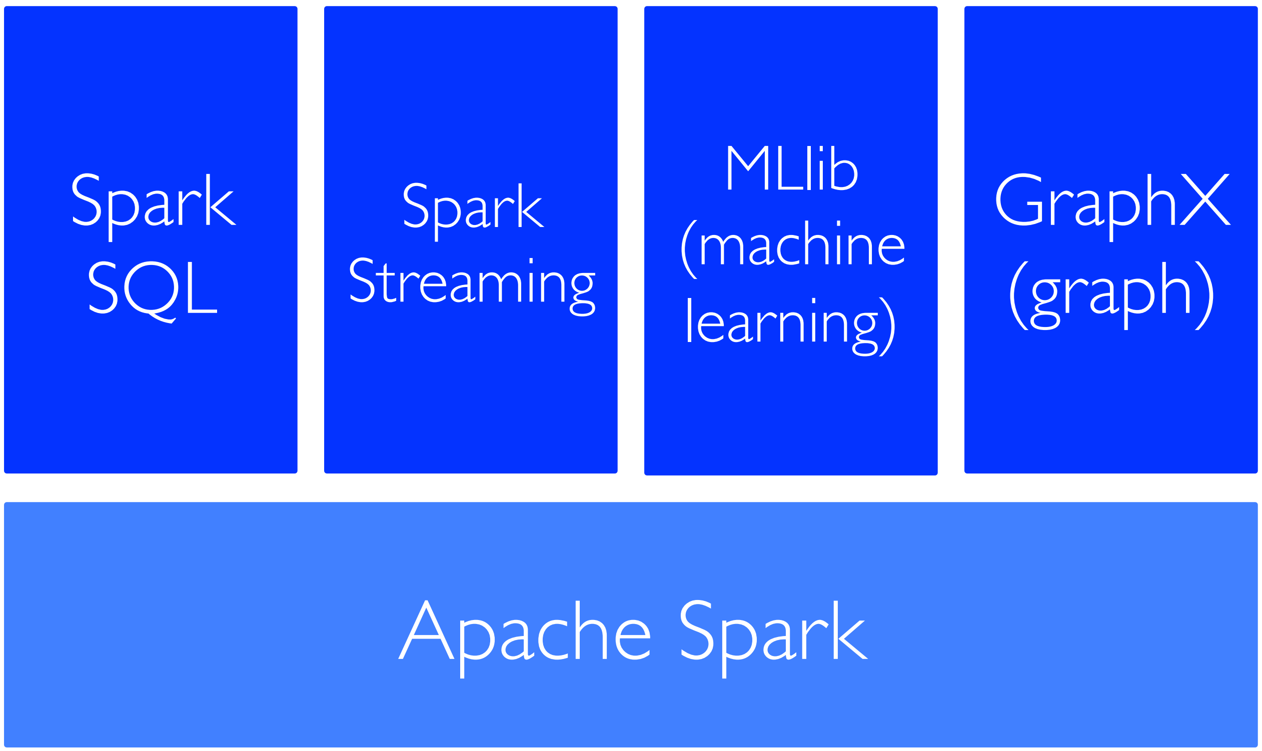
- Write programs in terms of operations on distributed datasets
- Partitioned collections of objects spread across a cluster, stored in memory or on disk
- RDDs built and manipulated through a diverse set of parallel transformations (map, filter, join) and actions (count, collect, save)
- RDDs automatically rebuilt on machine failure

The Spark Computing Framework

- Provides programming abstraction and parallel runtime to hide complexities of fault-tolerance and slow machines

- "Here's an operation, run it on all of the data"
  - » I don't care where it runs (you schedule that)
  - » In fact, feel free to run it twice on different nodes

Spark Tools


Spark SQL
Spark Streaming
MLlib (machine learning)
GraphX (graph)
Apache Spark

Spark and Map Reduce Differences

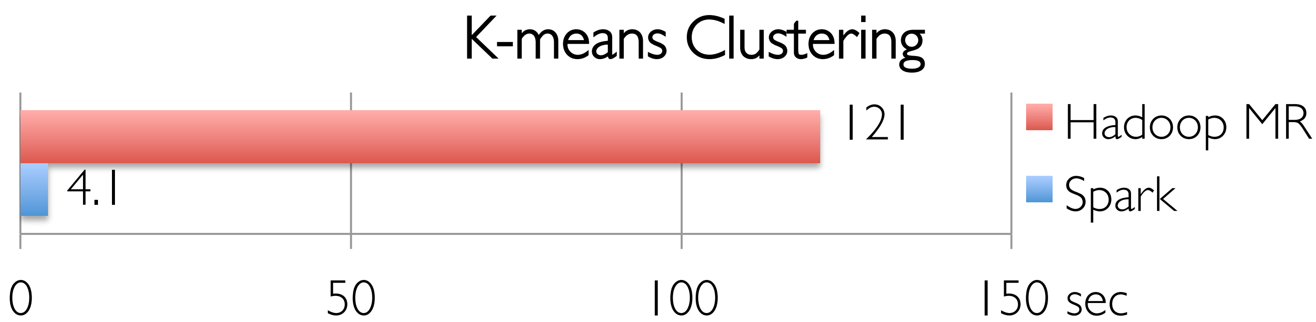
	Hadoop Map Reduce	Spark
Storage	Disk only	In-memory or on disk
Operations	Map and Reduce	Map, Reduce, Join, Sample, etc…
Execution model	Batch	Batch, interactive, streaming
Programming environments	Java	Scala, Java, R, and Python

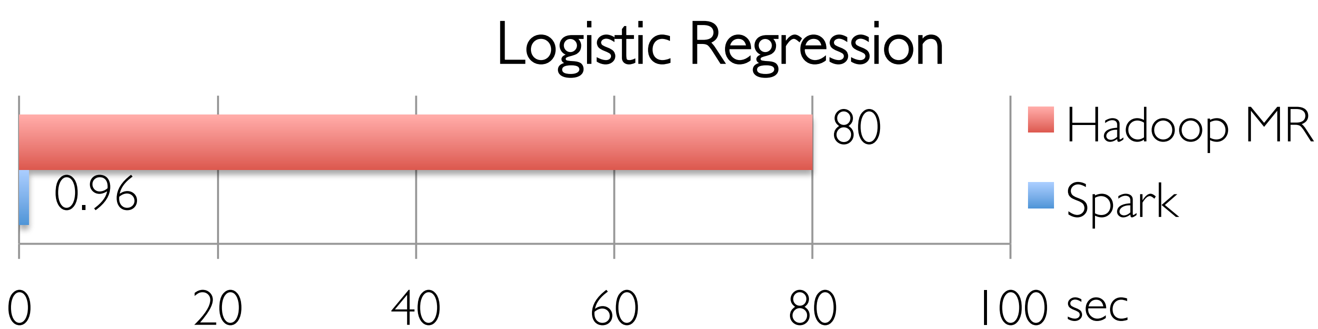
Other Spark and Map Reduce Differences

- Generalized patterns
  ⇒ unified engine for many use cases
- Lazy evaluation of the lineage graph
  ⇒ reduces wait states, better pipelining
- Lower overhead for starting jobs
- Less expensive shuffles

In-Memory Can Make a Big Difference

- Two iterative Machine Learning algorithms:





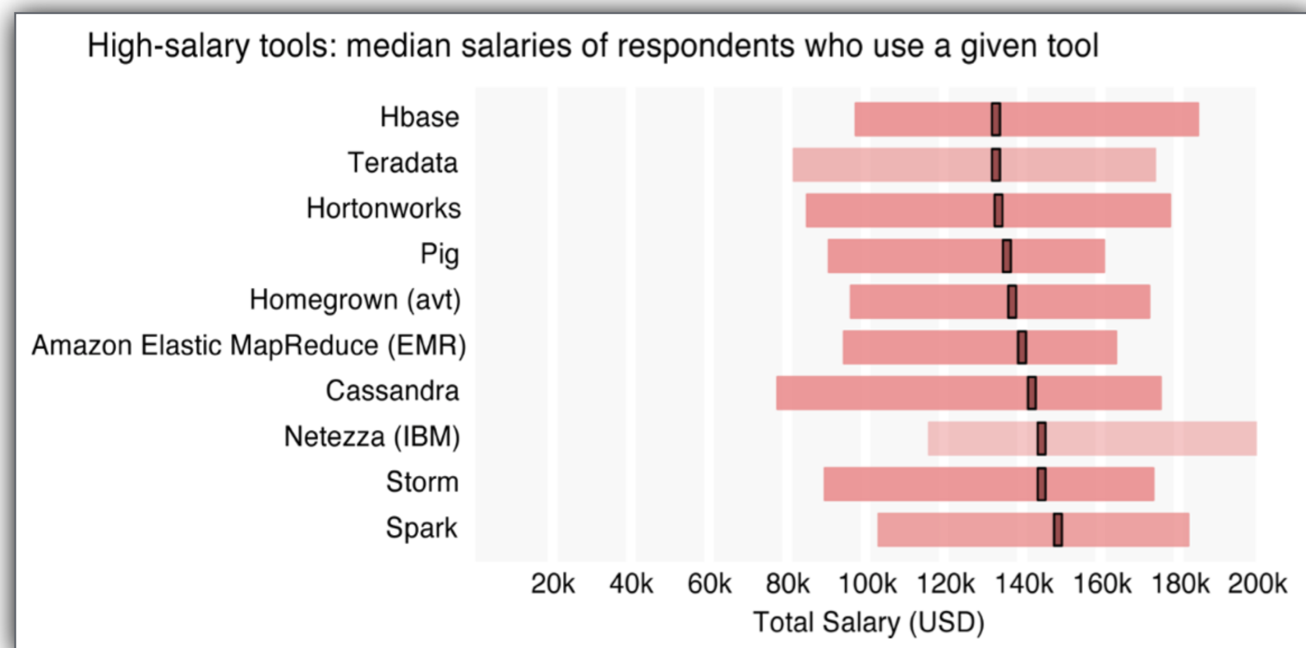
First Public Cloud Petabyte Sort

	Hadoop MR Record	**Spark Record**	**Spark 1 PB**
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**	**22.5 GB/min**

Daytona Gray 100 TB sort benchmark record (tied for 1st place)

http://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

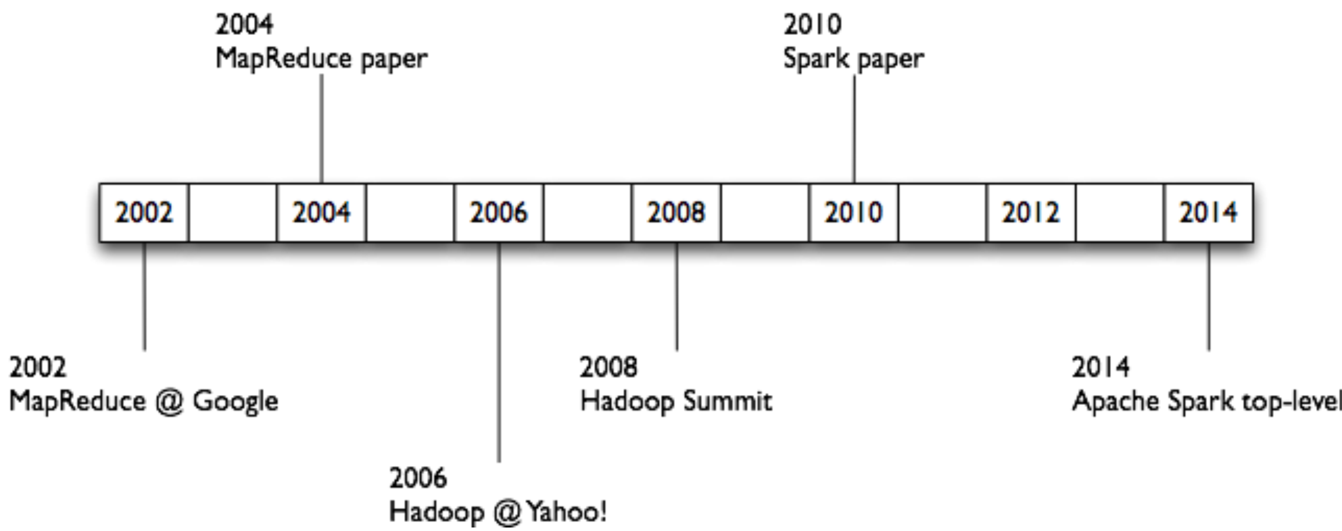
Spark Expertise Tops Big Data Median Salaries



Over 800 respondents across 53 countries and 41 U.S. states

http://www.oreilly.com/data/free/2014-data-science-salary-survey.csp

History Review



Historical References

- circa 1979 – **Stanford, MIT, CMU**, etc.: set/list operations in LISP, Prolog, etc., for parallel processing
  http://www-formal.stanford.edu/jmc/history/lisp/lisp.htm
- circa 2004 – **Google:** *MapReduce: Simplified Data Processing on Large Clusters*
  Jeffrey Dean and Sanjay Ghemawat
  http://research.google.com/archive/mapreduce.html
- circa 2006 – **Apache** *Hadoop*, originating from the Yahoo!'s Nutch Project
  Doug Cutting
  http://research.yahoo.com/files/cutting.pdf
- circa 2008 – **Yahoo!:** web scale search indexing
  Hadoop Summit, HUG, etc.
  http://developer.yahoo.com/hadoop/
- circa 2009 – **Amazon AWS:** Elastic MapReduce
  Hadoop modified for EC2/S3, plus support for Hive, Pig, Cascading, etc.
  http://aws.amazon.com/elasticmapreduce/

Spark Research Papers

- *Spark: Cluster Computing with Working Sets*
  Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica
  USENIX HotCloud (2010)
  people.csail.mit.edu/matei/papers/2010/hotcloud_spark.pdf

- *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing*
  Matei Zaharia, Mosharaf Chowdhury, Tathagata Das,
  Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin,
  Scott Shenker, Ion Stoica
  NSDI (2012)
  usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf