10-605/805 – ML for Large Datasets Lecture 2: Distributed Computing

Henry Chai

9/1/22

Front Matter

- HW1 released 8/30, due 9/13 at 11:59 PM
  - **For HW1 only, the programming part is optional (but strongly encouraged)**
  - The written part is nominally about PCA but can be solved using pre-requisite knowledge (linear algebra)
- Recitations on Friday, 11:50 – 1:10 (**different from lecture**) in GHC 4401 (**same as lecture**)
  - Recitation 1 on 9/2: Introduction to PySpark/Databricks

Recall: Machine Learning with Large Datasets

- Premise:
  - There exists some pattern/behavior of interest
  - The pattern/behavior is difficult to describe
  - There is data (sometimes a lot of it!)
  - More data *usually* helps
  - Use data efficiently/intelligently to “learn” the pattern
- Definition:
  - A computer program **learns** if its *performance*, *P*, at some *task*, *T*, improves with *experience*, *E*.

Okay, but how much data are we talking about here?

- Premise:
  - There exists some pattern/behavior of interest
  - The pattern/behavior is difficult to describe
  - There is data (sometimes **a lot** of it!)
  - More data *usually* helps
  - Use data efficiently/intelligently to “learn” the pattern
- Definition:
  - A computer program **learns** if its *performance*, *P*, at some *task*, *T*, improves with *experience*, *E*.

Units of Data

Unit	Value	Scale
Kilobyte (KB)	1000 bytes	A paragraph of text
Megabyte (MB)	1000 KB	A short novel
Gigabyte (GB)	1000 GB	Beethoven's 5^{th} symphony
Terabyte (TB)	1000 TB	All the x-rays in a large hospital
Petabyte (PB)	1000 PB	$\approx {}^{1}/_{2}$ of the content of all US research libraries
Exabyte (EB)	1000 EB	$\approx {}^{1}/_{5}$ of the words humans have ever spoken

A DAY IN DATA

The exponential growth of data is undisputed, but the numbers behind this explosion - fuelled by internet of things and the use of connected devcies – are hard to comprehend, particularly when looked at in the context of one day

DEMYSTIFIYING DATA UNITS

From the more familiar 'bit' or 'megabyte', larger units of measurement are more frequently being used to explain the masses of data

Unit		Value	Size
b	bit	0 or 1	1/8 of a byte
B	byte	8 bits	1 byte
KB	kilobyte	1,000 bytes	1,000 bytes
MB	megabyte	$1,000^2$ bytes	1,000,000 bytes
GB	gigabyte	$1,000^3$ bytes	1,000,000,000 bytes
TB	terabyte	$1,000^4$ bytes	1,000,000,000,000 bytes
PB	petabyte	$1,000^5$ bytes	1,000,000,000,000,000 bytes
EB	exabyte	$1,000^6$ bytes	1,000,000,000,000,000,000 bytes
ZB	zettabyte	$1,000^7$ bytes	1,000,000,000,000,000,000,000 bytes
YB	yottabyte	$1,000^8$ bytes	1,000,000,000,000,000,000,000,000 bytes

*A lowercase "b" is used as an abbreviation for bits, while an uppercase "B" represents bytes.

463EB

of data will be created every day by 2025

IDC

500m

tweets are sent every day

Twitter

4PB

of data created by Facebook, including

350m photos

100m hours of video watch time

Facebook Research

95m

photos and videos are shared on Instagram

Instagram Business

320bn

emails to be sent each day by 2021

306bn

emails to be sent each day by 2020

294bn

billion emails are sent

Radicati Group

65bn

messages sent over WhatsApp and two billion minutes of voice and video calls made

Facebook

3.9bn

people use emails

4TB

of data produced by a connected car

Intel

28PB

to be generated from wearable devices by 2020

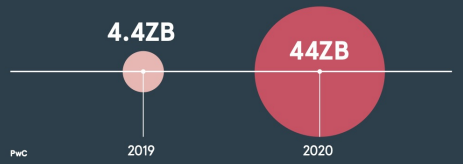
Statista

Searches made a day — 5bn

Searches made a day from Google — 3.5bn

Smart Insights

ACCUMULATED DIGITAL UNIVERSE OF DATA



PwC

RACONTEUR

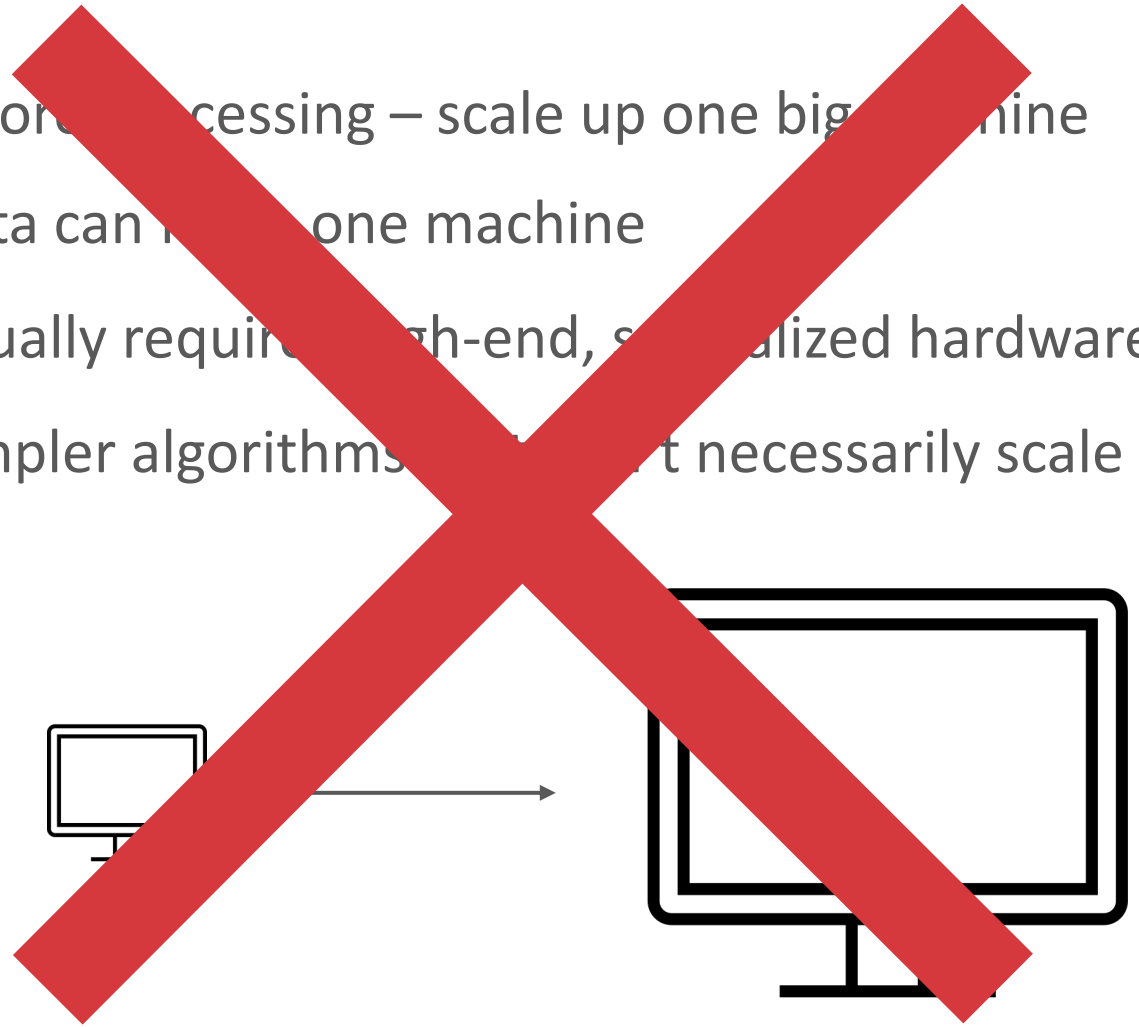
Source: https://res.cloudinary.com/yumyoshojin/image/upload/v1/pdf/future-data-2019.pdf

The Big Data Problem

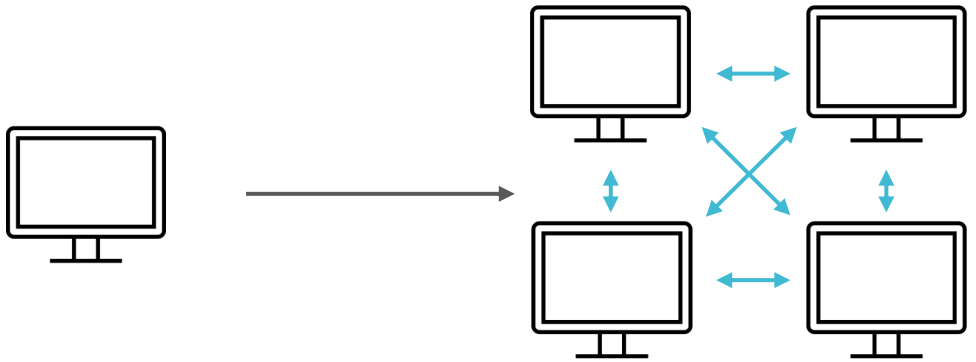
- The sources and amount of data keep growing
- Data storage and processing can’t keep up
  - A 1 TB hard disk costs ≈ $25 USD
  - Reading 1 TB from disk ≈ 5 hours
  - So it would cost $100,000 USD to store Facebook’s 4PB of data/day and take ≈ 2.5 years to read!

Recall: Parallel Computing

- Multi-core processing – scale up one big machine
  - Data can fit on one machine
  - Usually requires high-end, specialized hardware
  - Simpler algorithms but not necessarily scale well



Recall: Parallel Computing



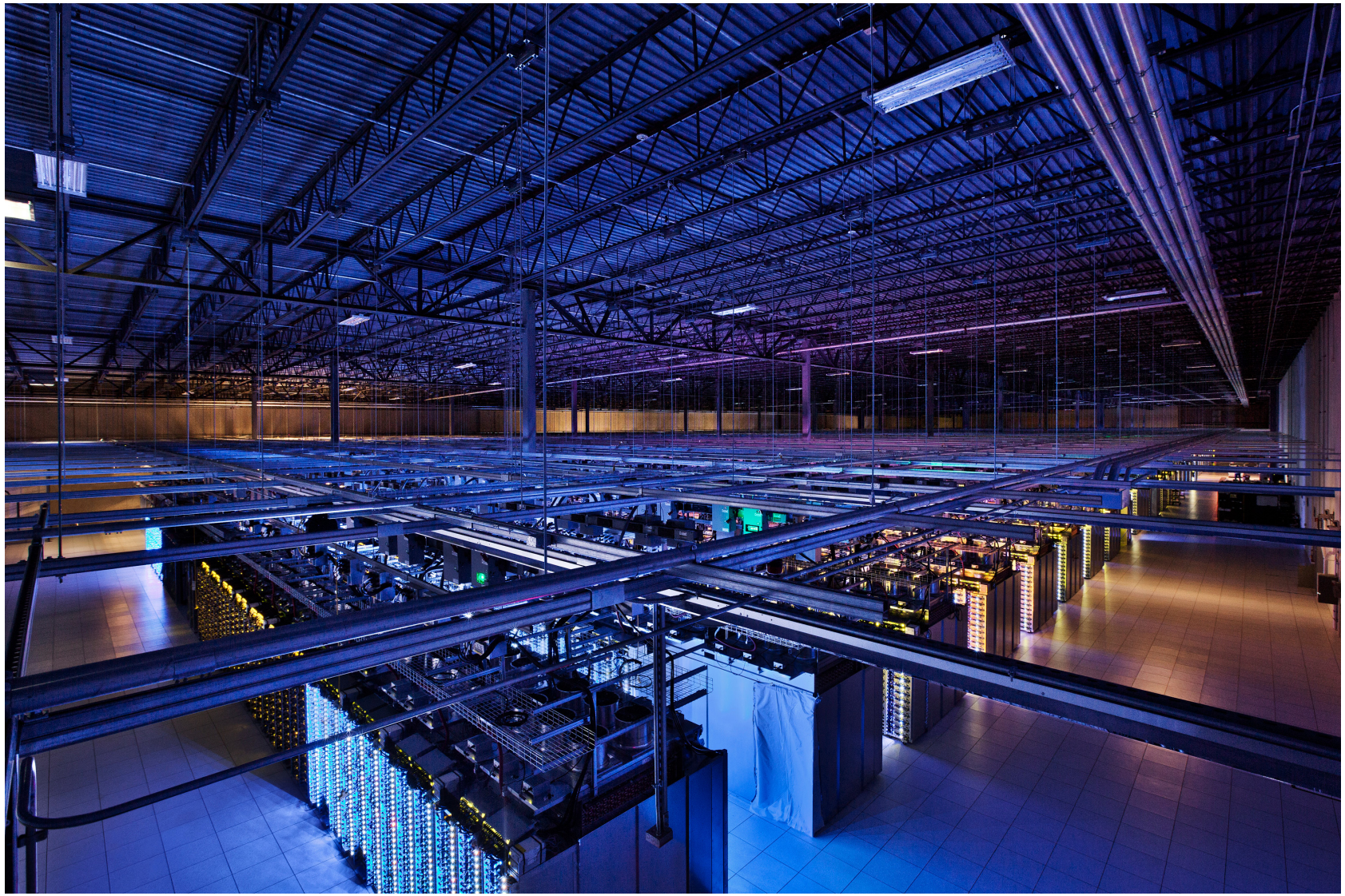
- Distributed processing – scale out many machines
  - Data stored across multiple machines
  - Scales to massive problems on standard hardware
  - Added complexity of network communication

Cloud Computing

- Enables distributed processing by democratizing access to storage and computational resources
- Costs continue to decrease each year

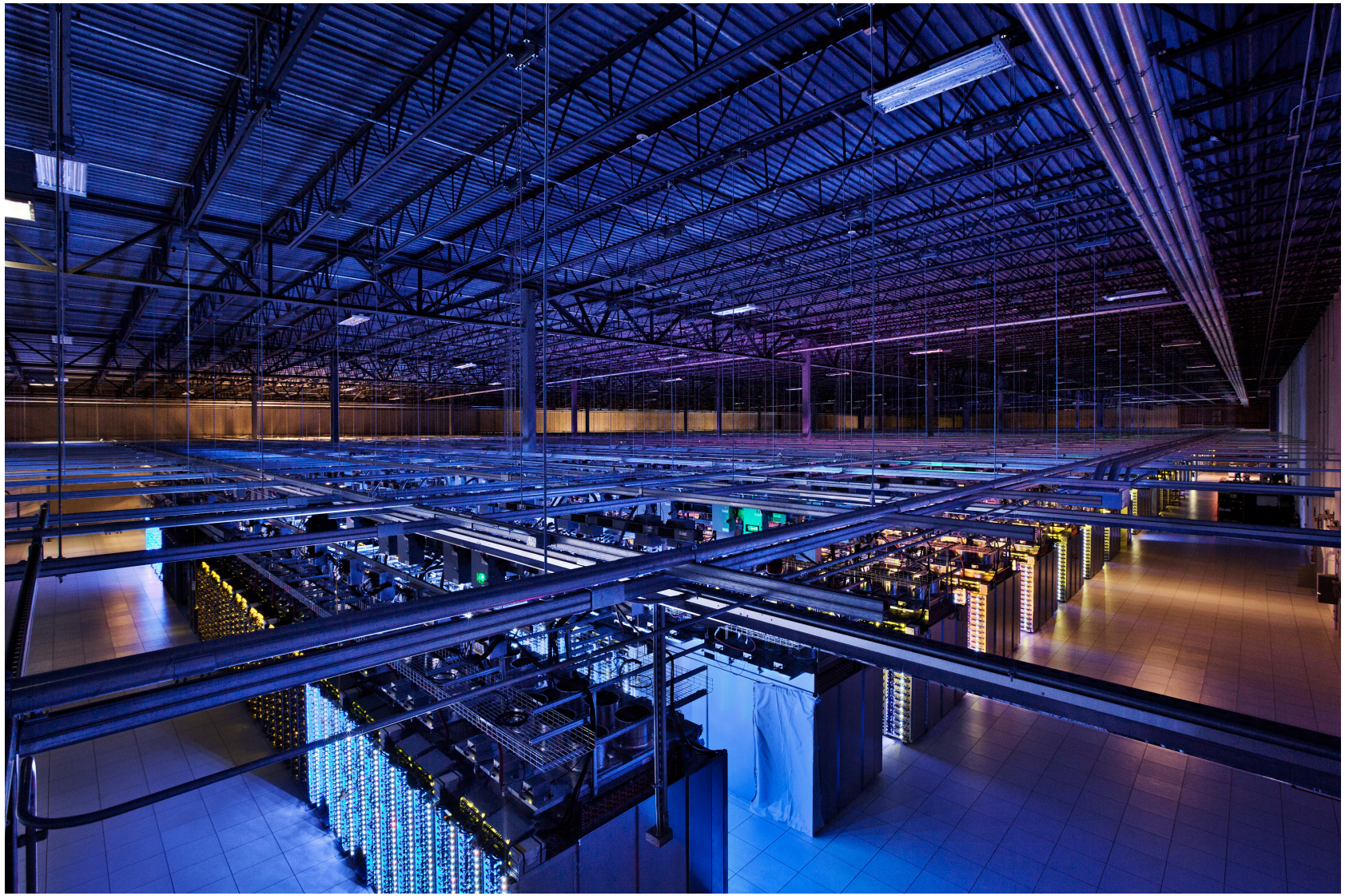
Google Cloud

Cloud Computing



Source: https://www.google.com/about/datacenters/gallery/

How can we program in this setting?



Source: https://www.google.com/about/datacenters/gallery/

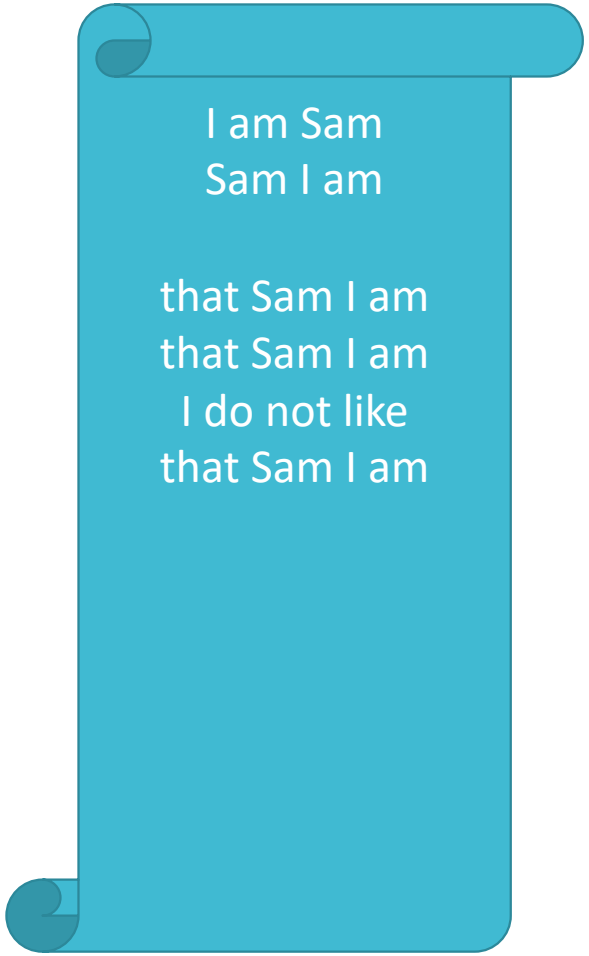
Challenges in Cloud Computing

1. How can we divide work across multiple machines?

2. What can we do if a machine fails?

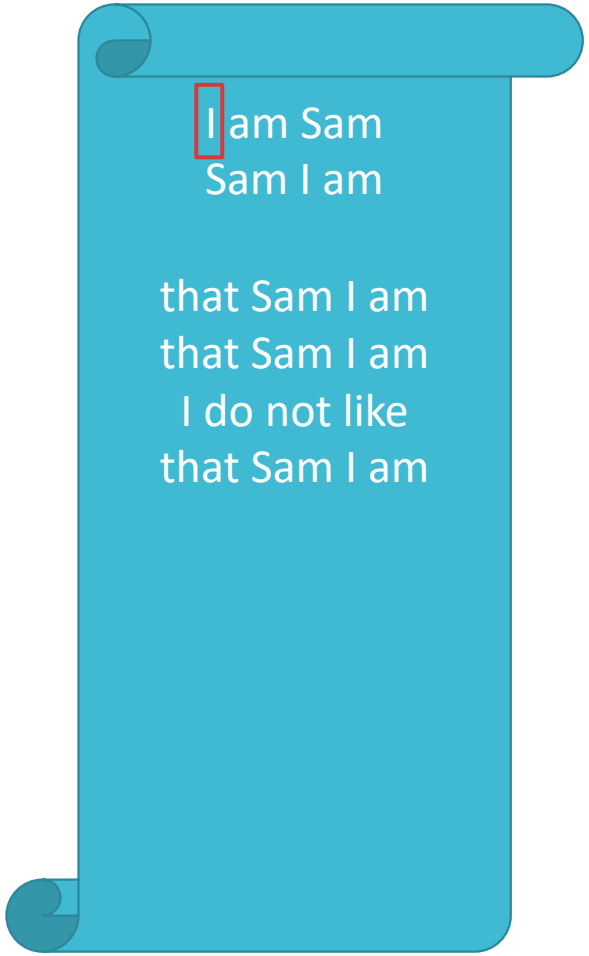
Example: Word Counting

- Given a text, count the number of times each word appears



Example: Word Counting

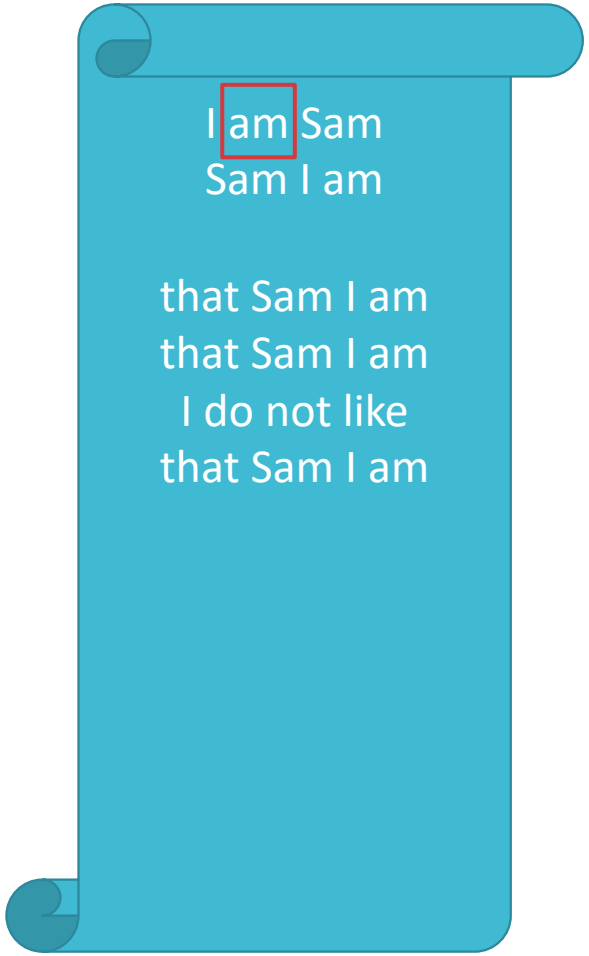
- Given a text, count the number of times each word appears



Word	Count
I	1

Example: Word Counting

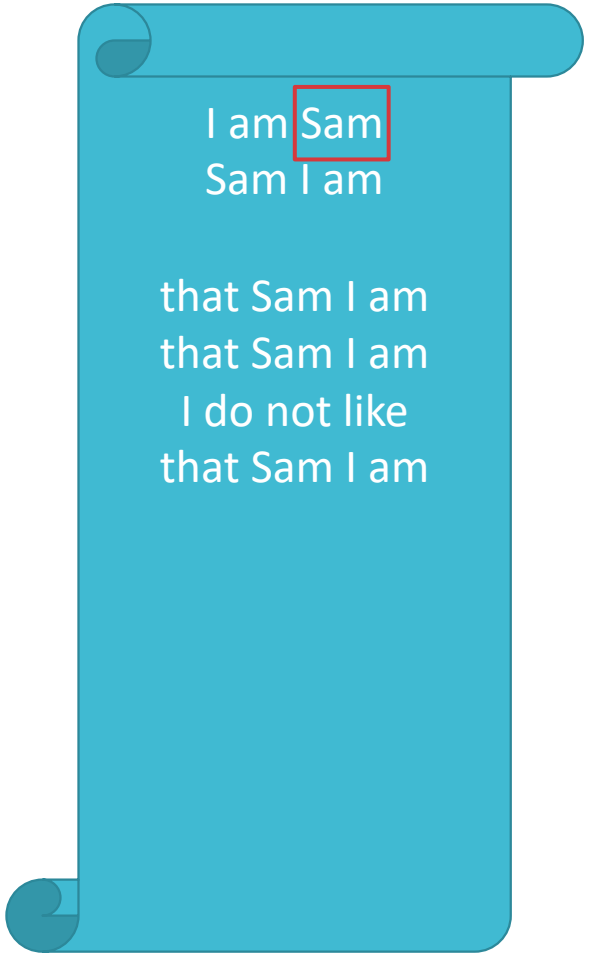
- Given a text, count the number of times each word appears



Word	Count
I	1
am	1

Example: Word Counting

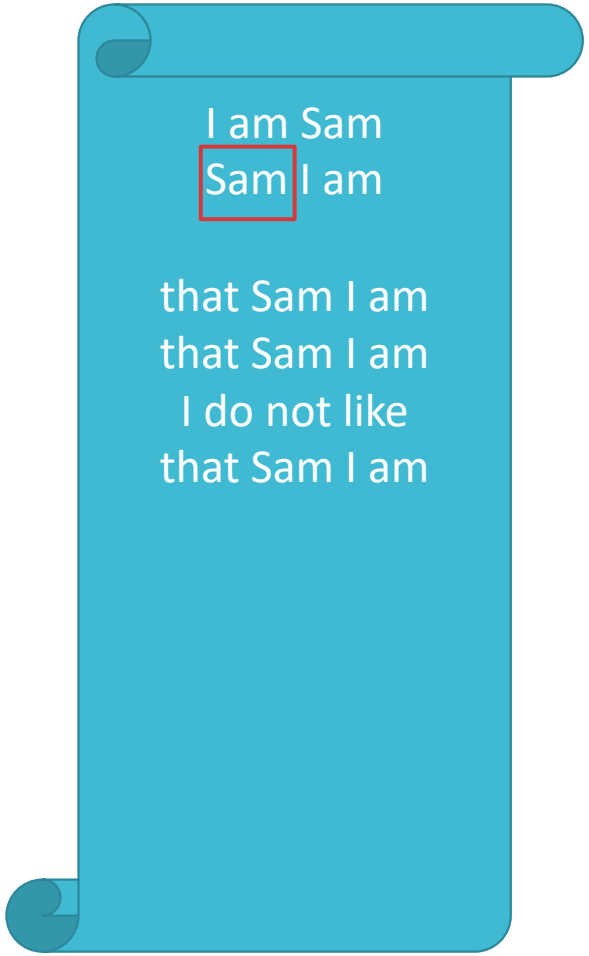
- Given a text, count the number of times each word appears



Word	Count
I	1
am	1
Sam	1

Example: Word Counting

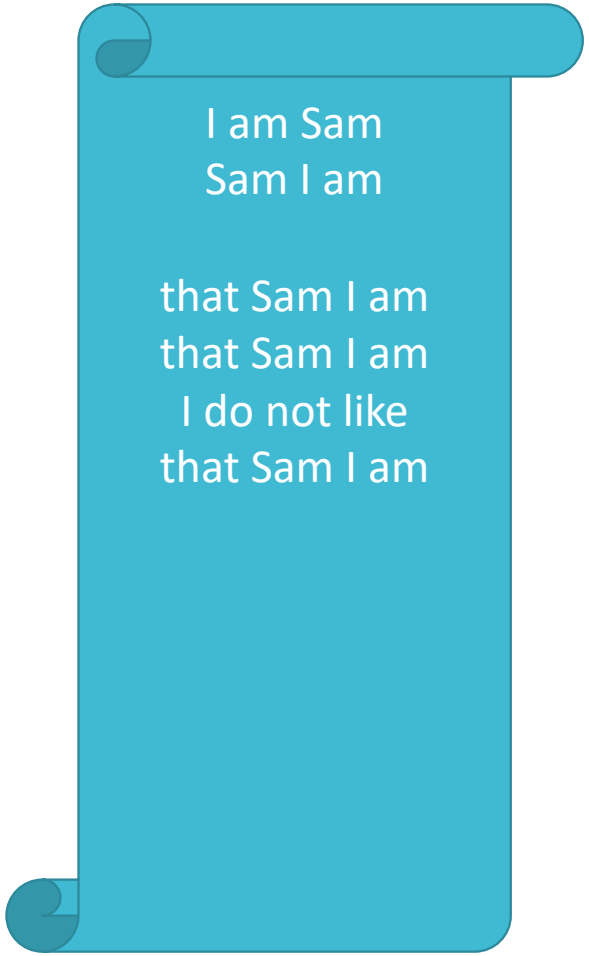
- Given a text, count the number of times each word appears



Word	Count
I	1
am	1
Sam	2

Example: Word Counting

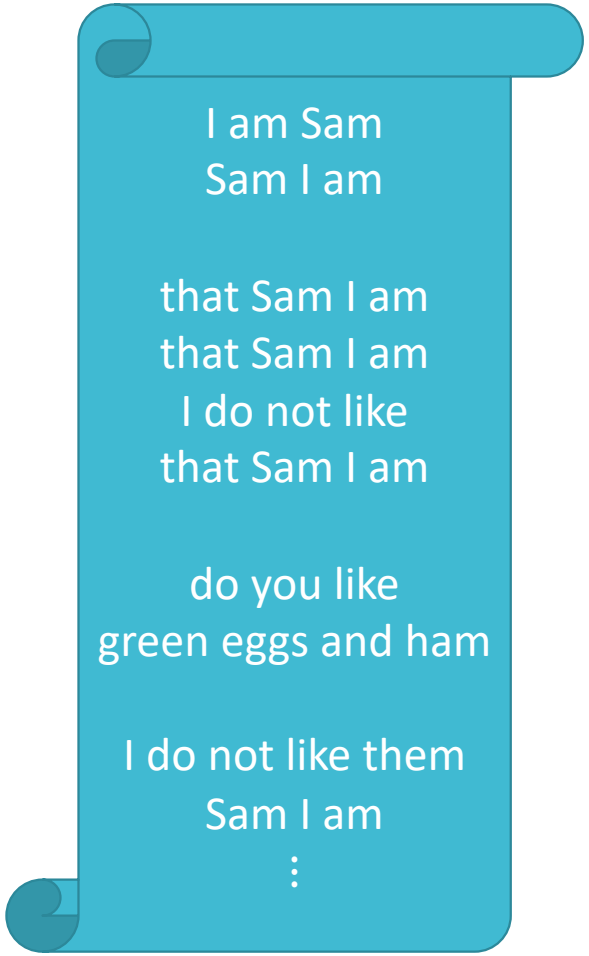
- Given a text, count the number of times each word appears



Word	Count
I	6
am	5
Sam	5
that	3
do	1
not	1
like	1

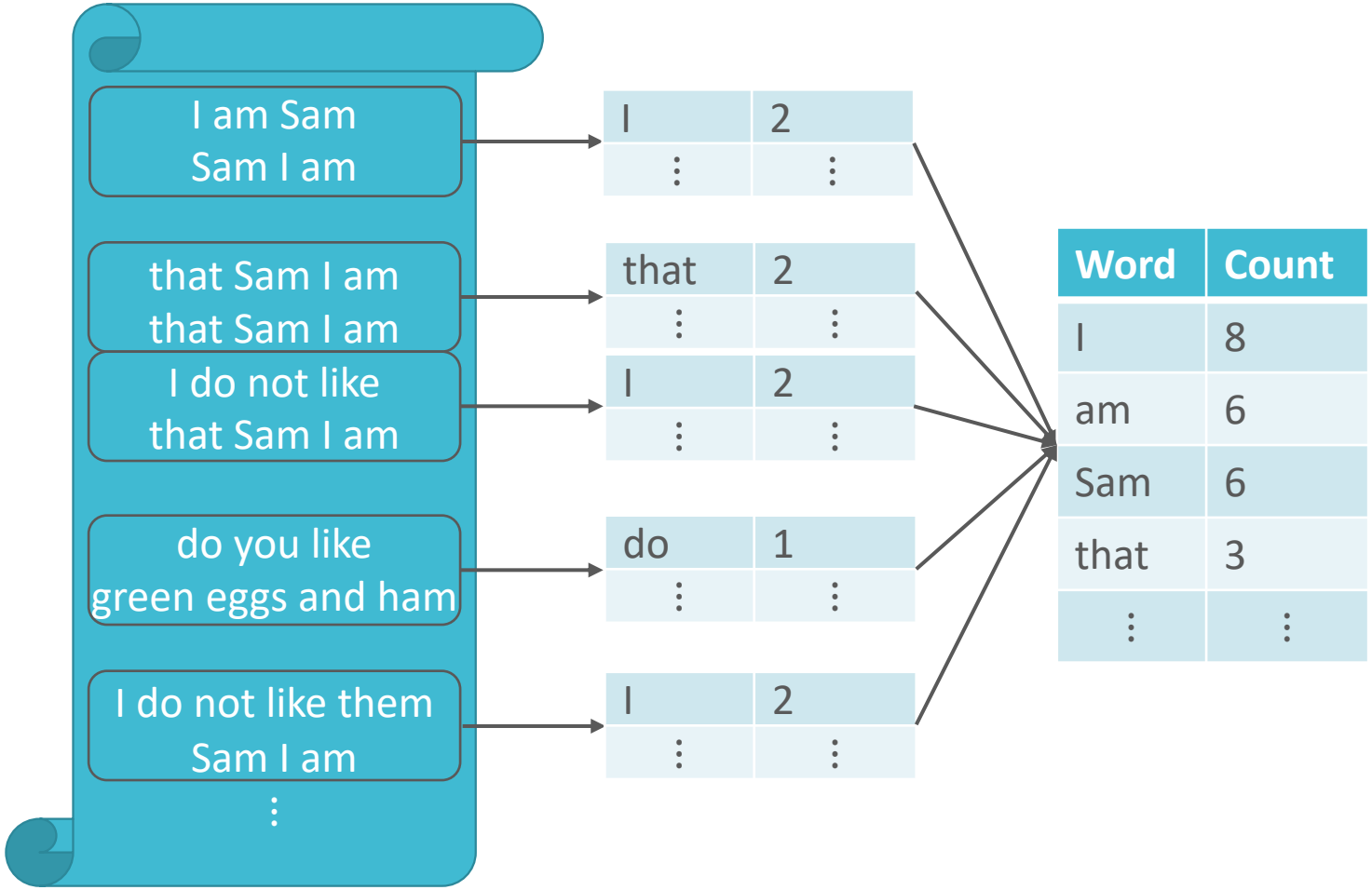
Example: Word Counting

- Given a text, count the number of times each word appears



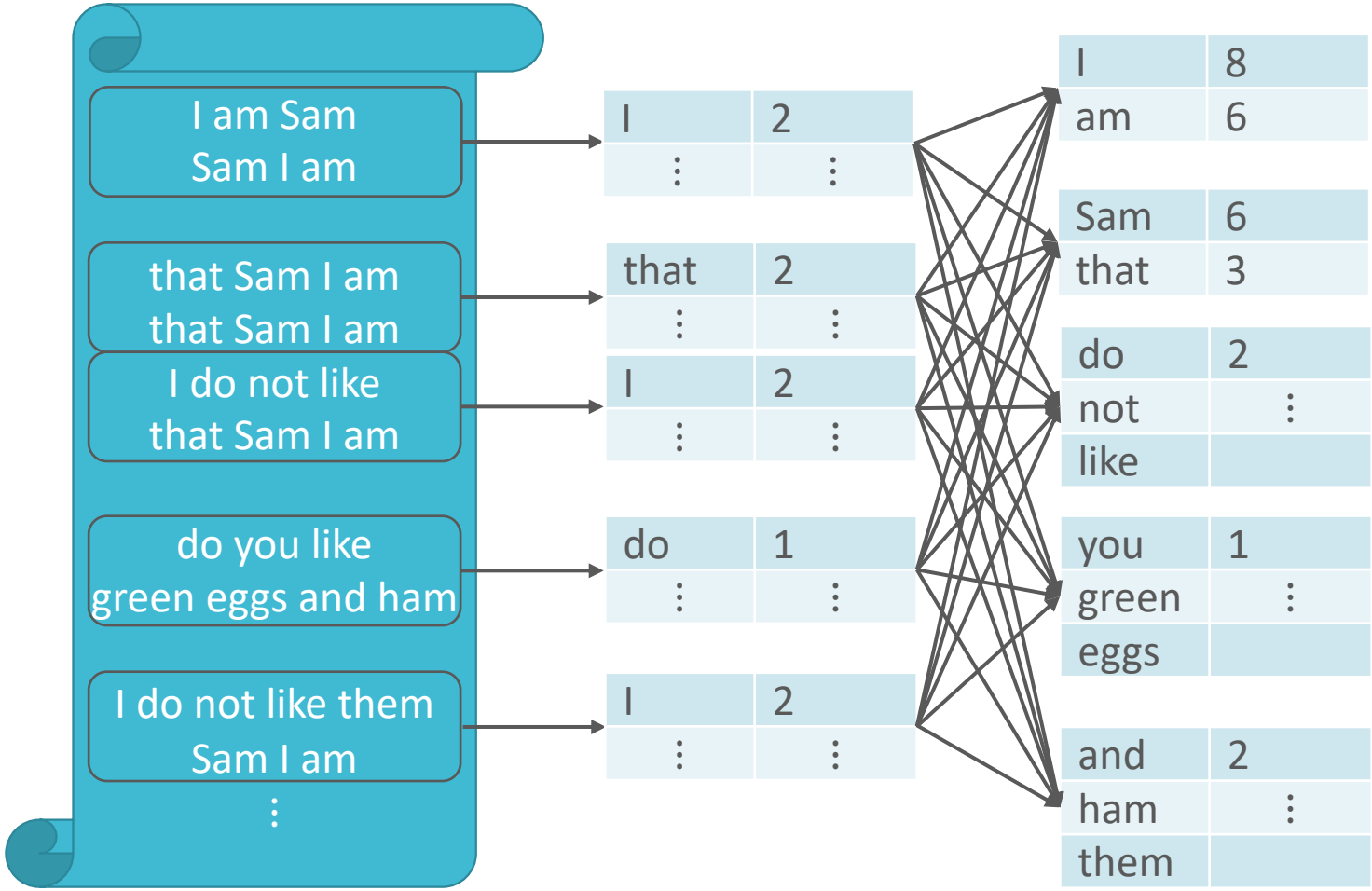
Example: Word Counting

- Given a text, count the number of times each word appears



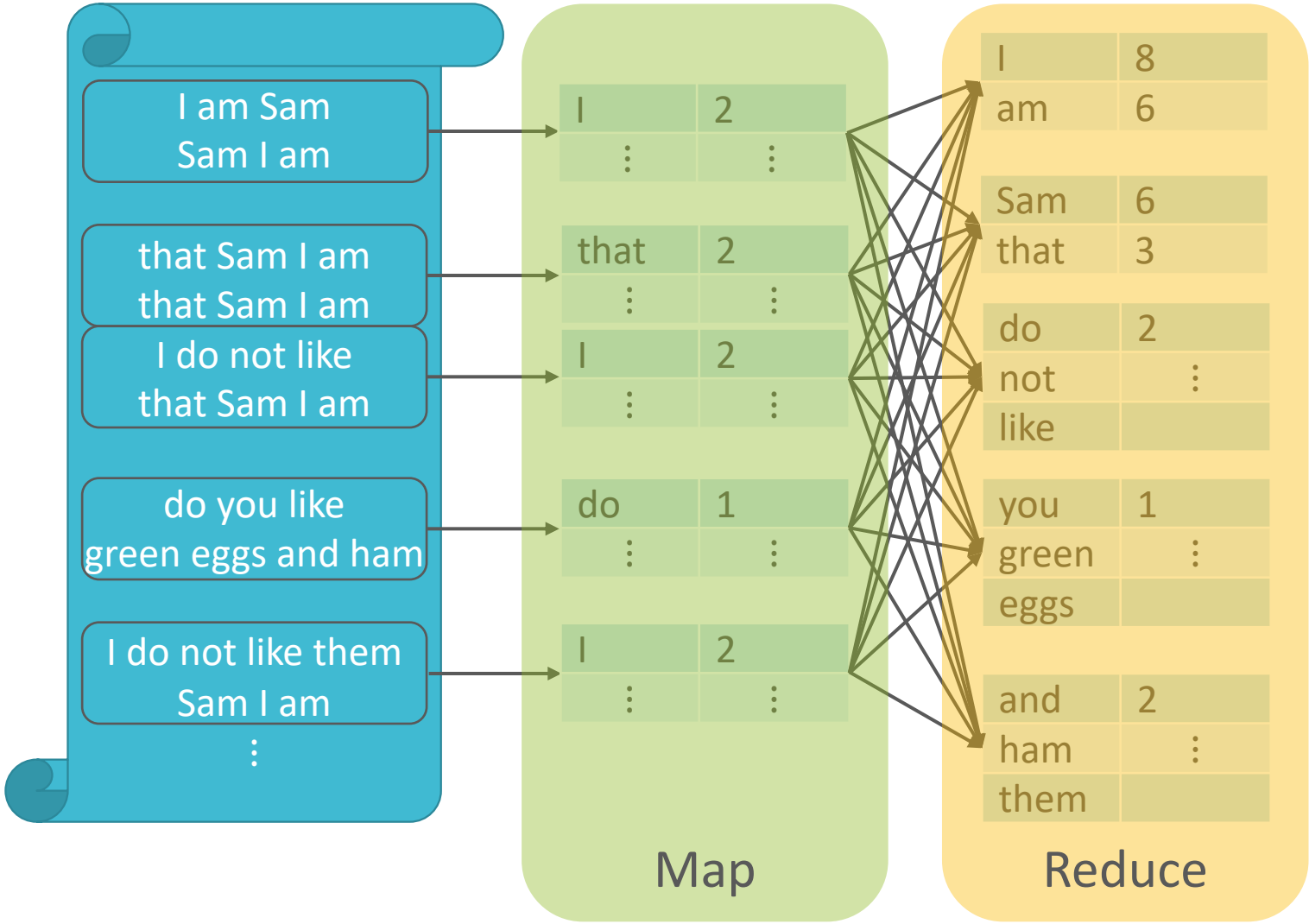
Example: Word Counting

- Given a text, count the number of times each word appears



MapReduce (Apache Hadoop)

- Given a text, count the number of times each word appears



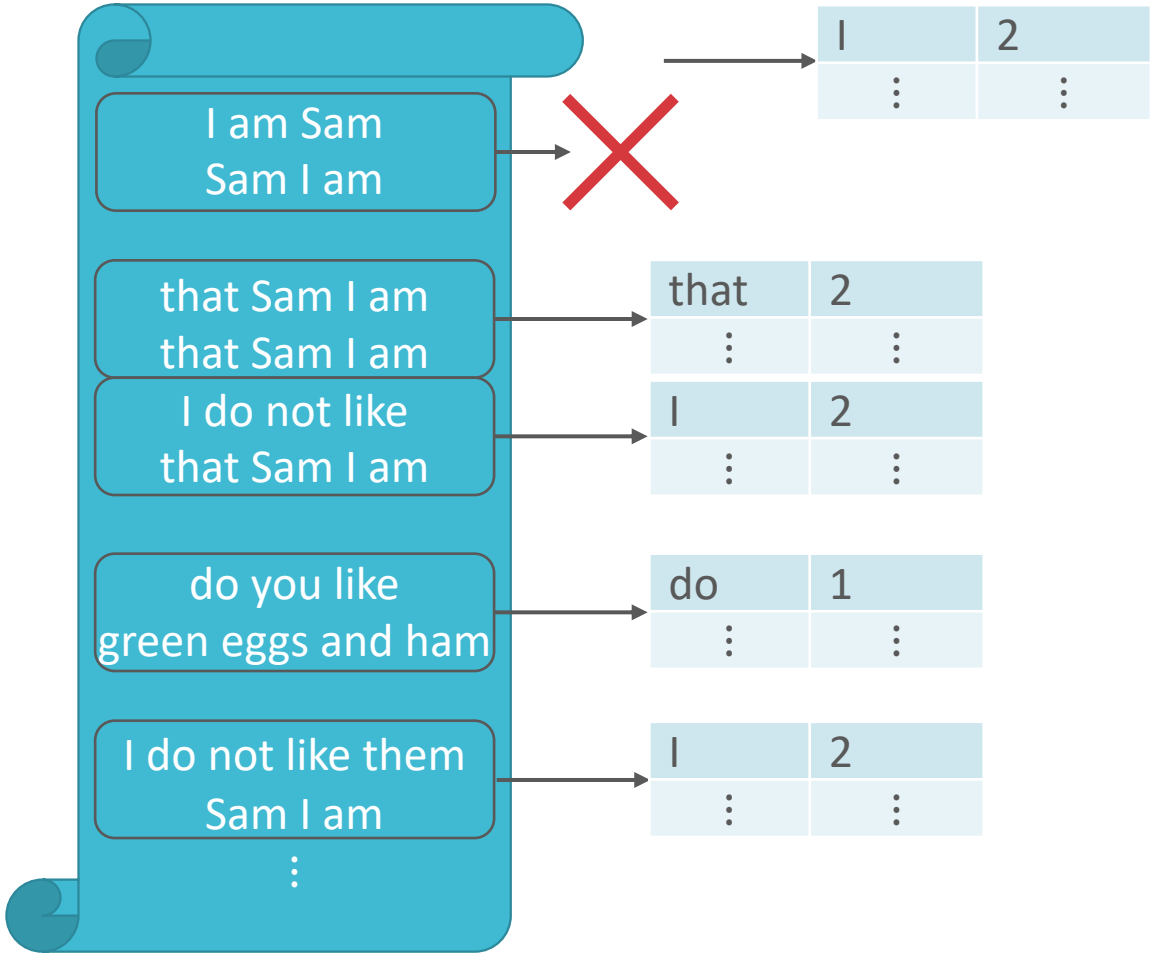
Challenges in Cloud Computing

1. How can we divide work across multiple machines?

2. What can we do if a machine fails?
   - If a node fails every 3 years …
   - … then a center with 10,000 nodes will have 10 faults/day!
   - Even worse are stragglers, or nodes that run slower than others

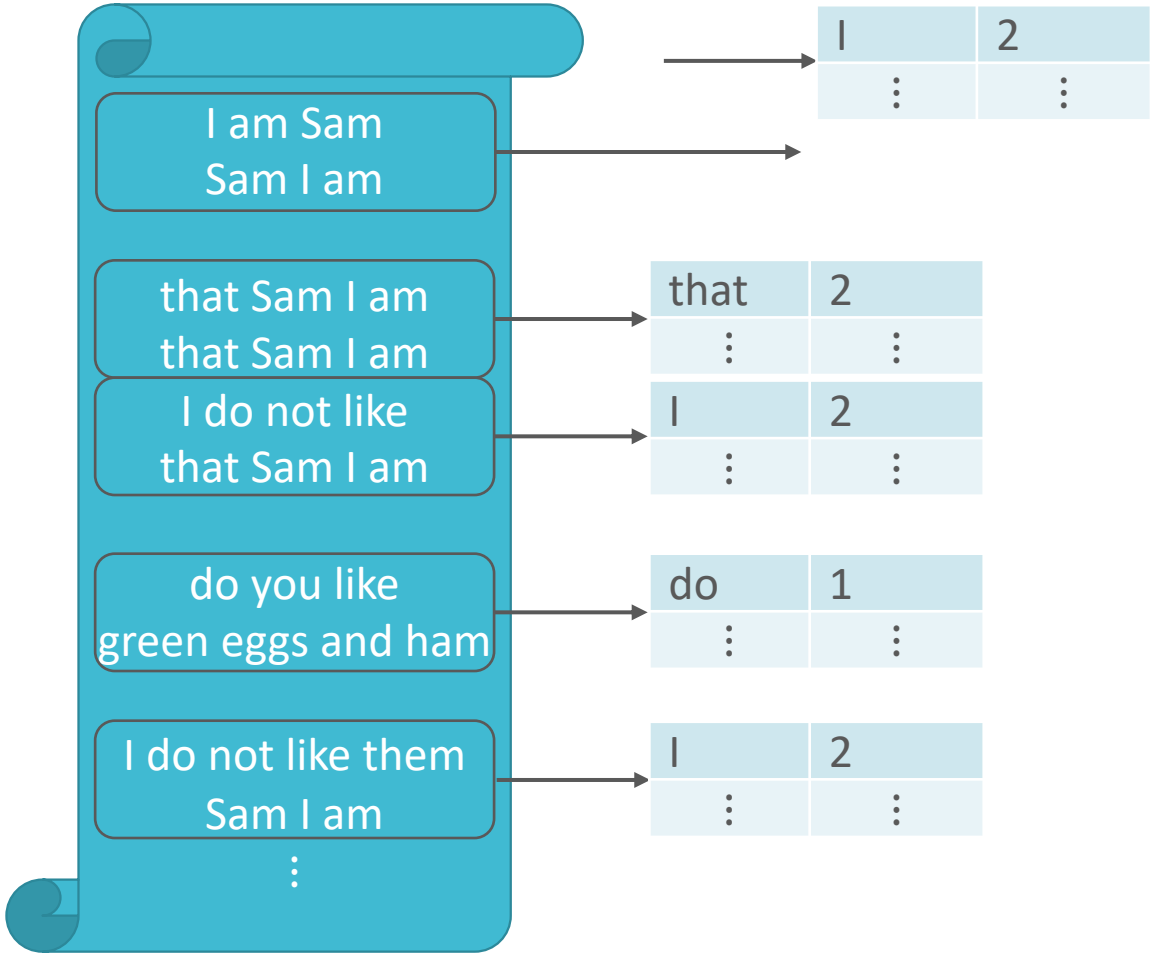
Machine Failures

- Machines are plentiful, so just launch another job!



Machine Failures and Stragglers

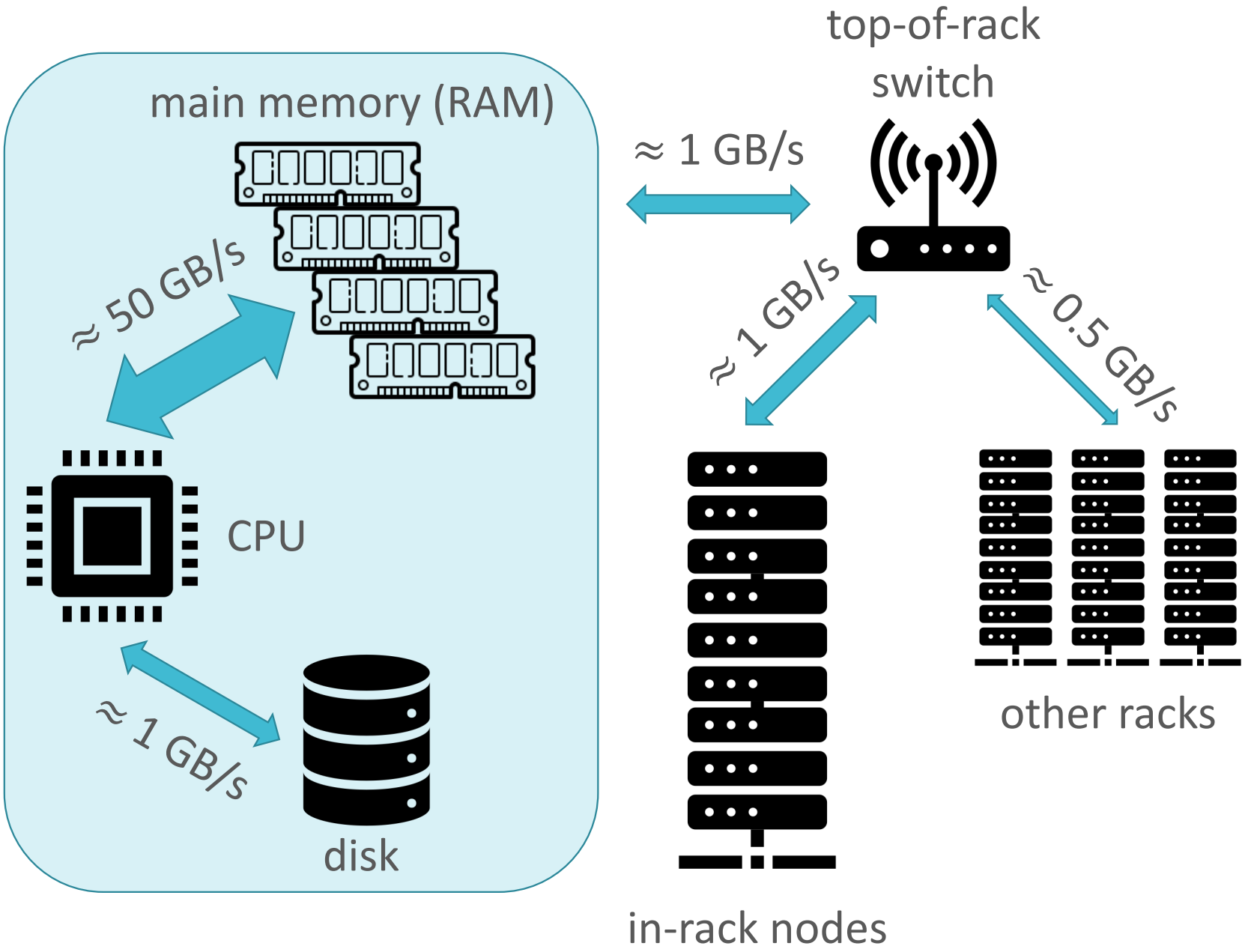
- Machines are plentiful, so just launch another job!



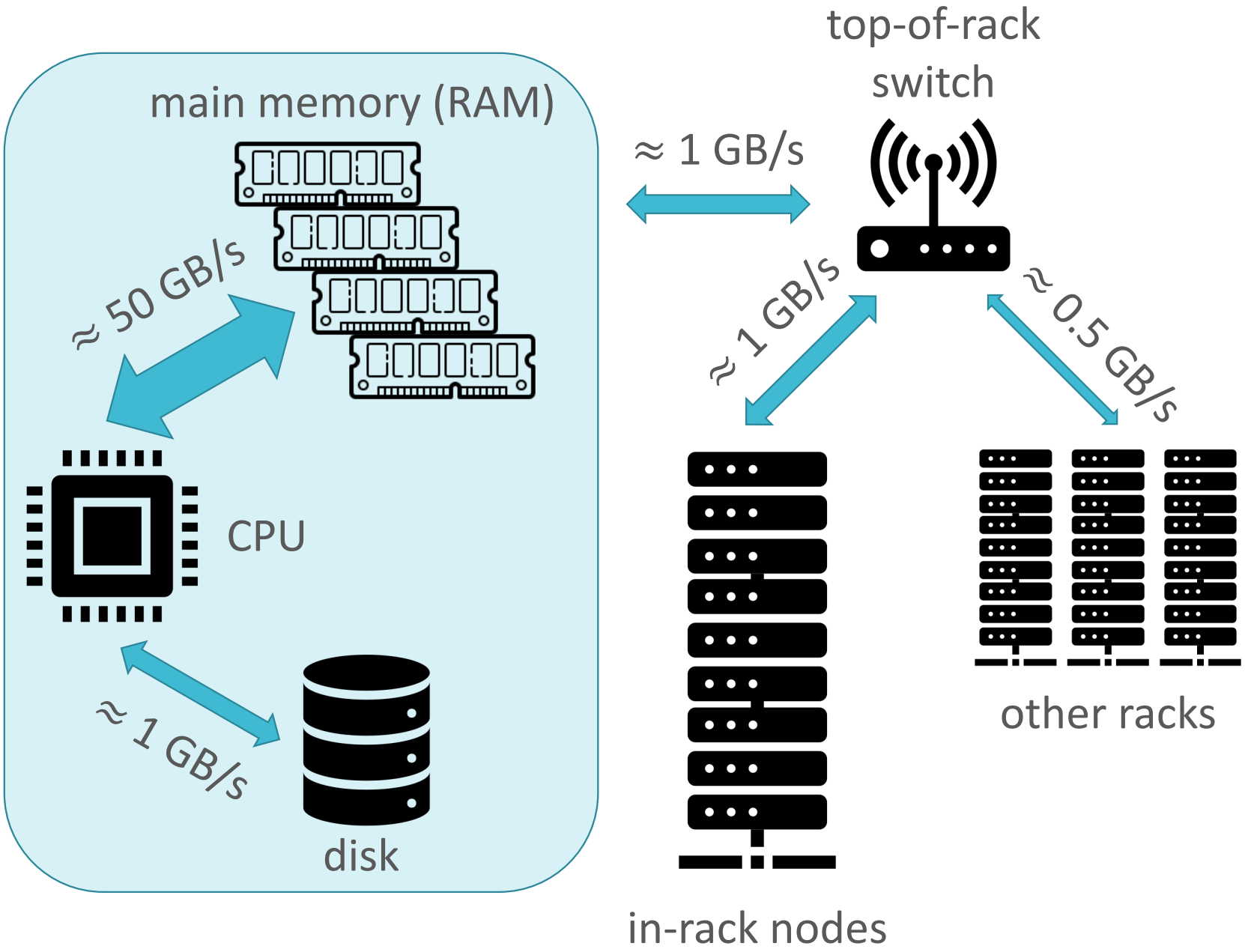
Challenges in Cloud Computing

1. How can we divide work across multiple machines?
   - Key consideration: moving data across machines/over a network is costly!
2. What can we do if a machine fails?
   - If a node fails every 3 years …
   - … then a center with 10,000 nodes will have 10 faults/day!
   - Even worse are stragglers, or nodes that run slower than others

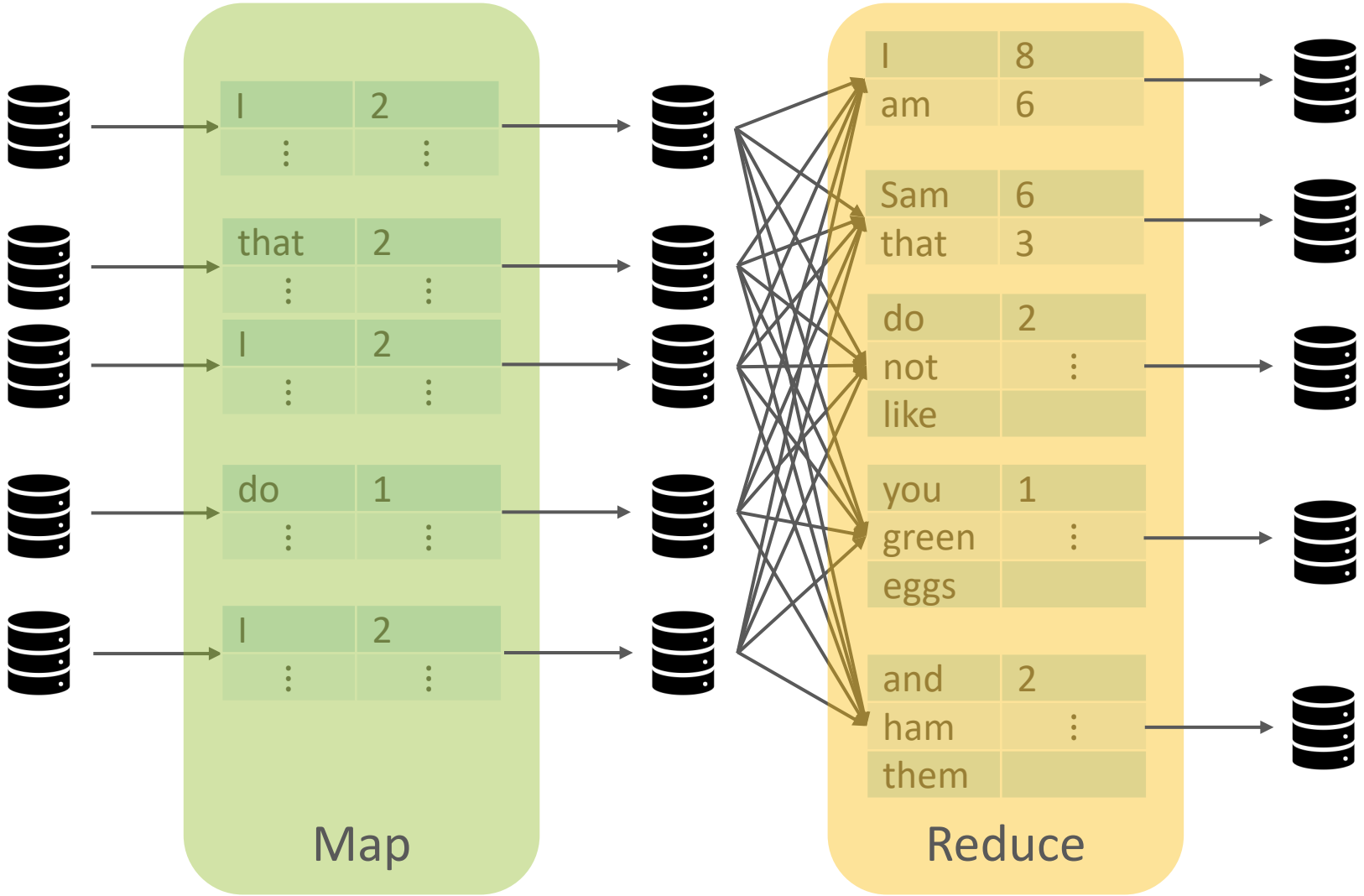
Communication Hierarchy



What are the implications for MapReduce?

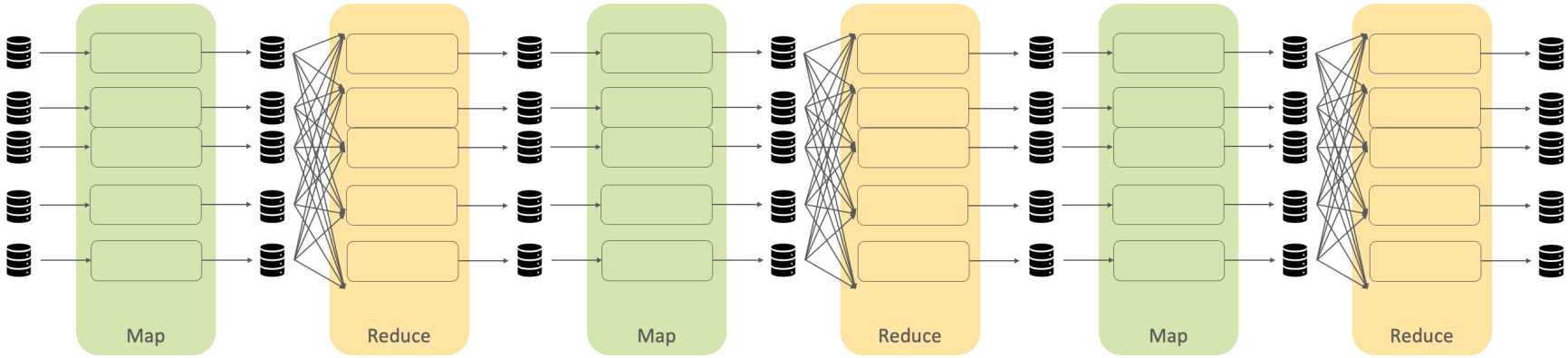


MapReduce


I 2
that 2
I 2
do 1
I 2
Map
I 8
am 6
Sam 6
that 3
do 2
not
like
you 1
green
eggs
and 2
ham
them
Reduce

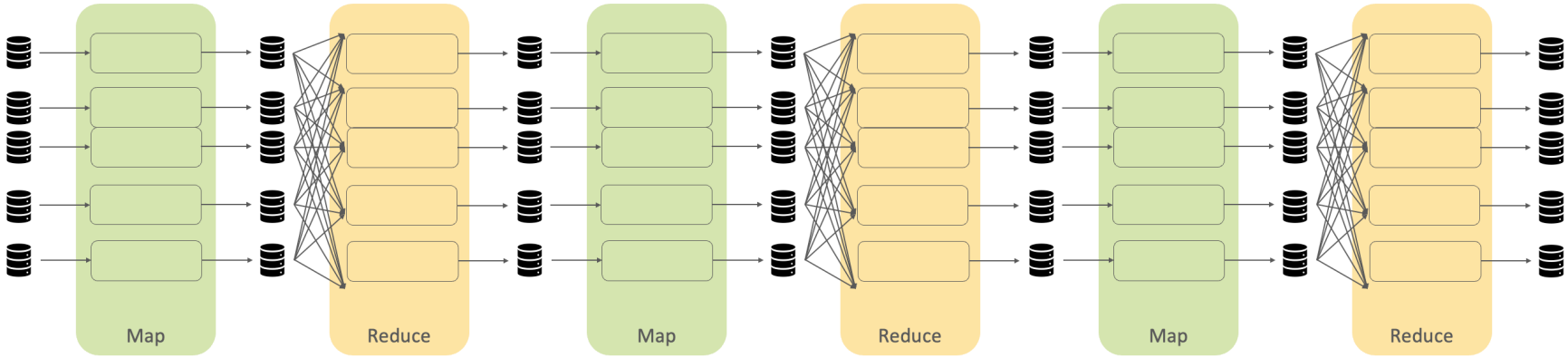
MapReduce & Iterative Procedures

- Iterative jobs involve lots of disk reading/writing per iteration, which is slow!



MapReduce & ~~Iterative Procedures~~ Machine Learning

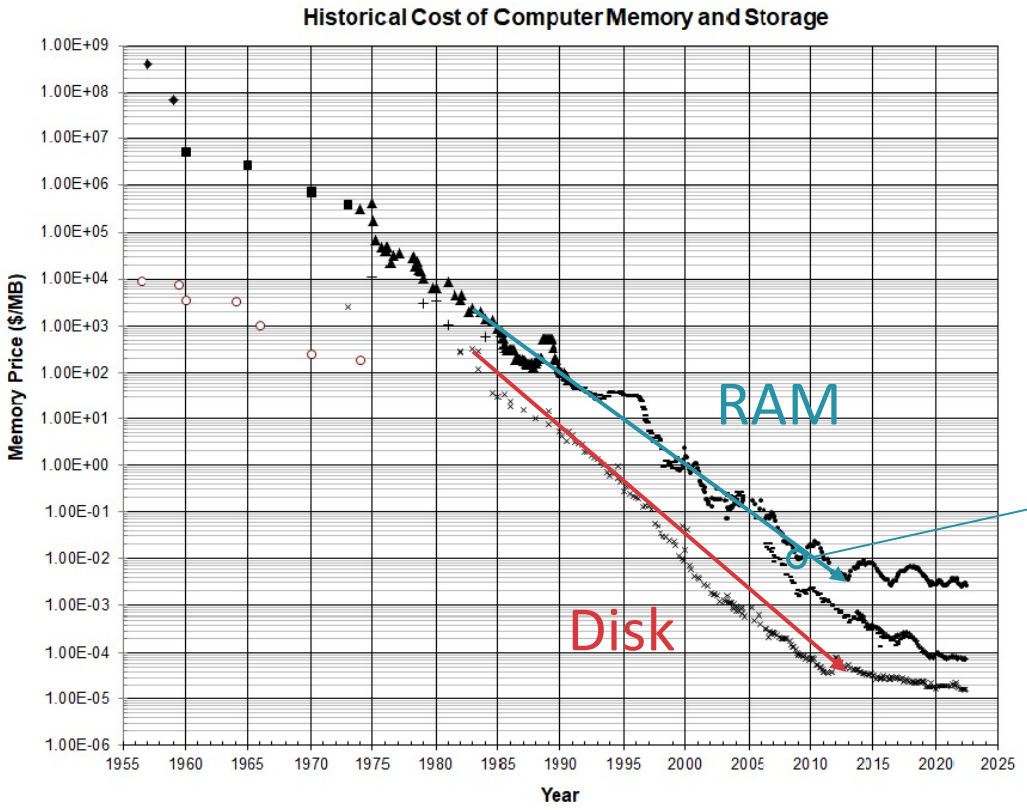
- Iterative jobs involve lots of disk reading/writing per iteration, which is slow!



- Issue: many machine learning/optimization algorithms are inherently iterative!

MapReduce & ~~Iterative Procedures~~ Machine Learning

- Insight: the cost of RAM has been steadily decreasing

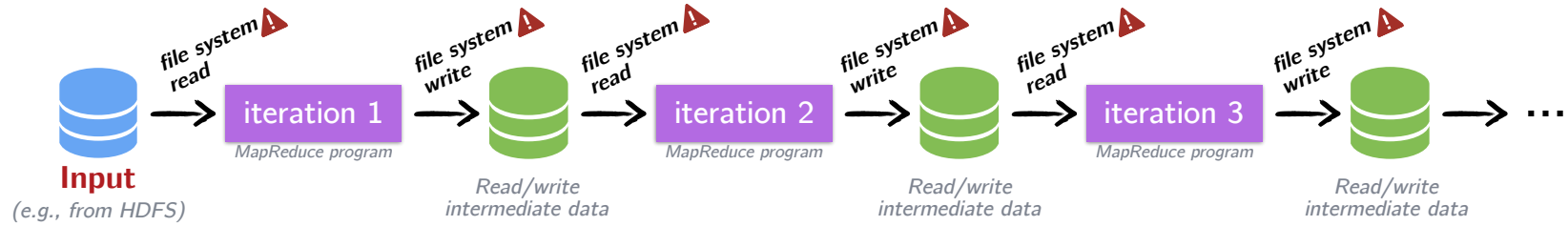


- Idea: shove more RAM into each machine and hold more data in main memory → Apache Spark (circa 2010)

Source: https://jcmit.net/mem2015.htm

Apache Spark vs. Hadoop

Iteration in Hadoop:



Iteration in Spark:

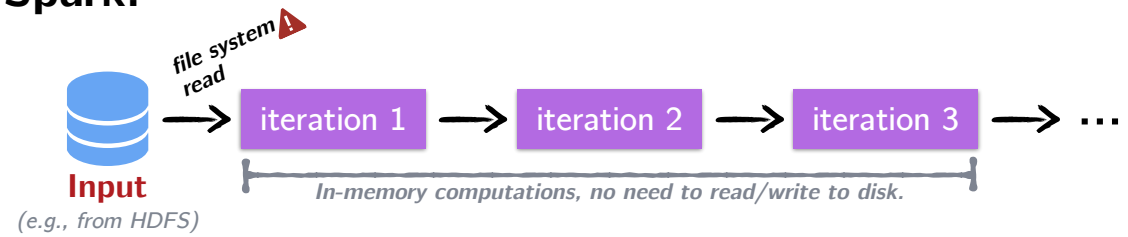
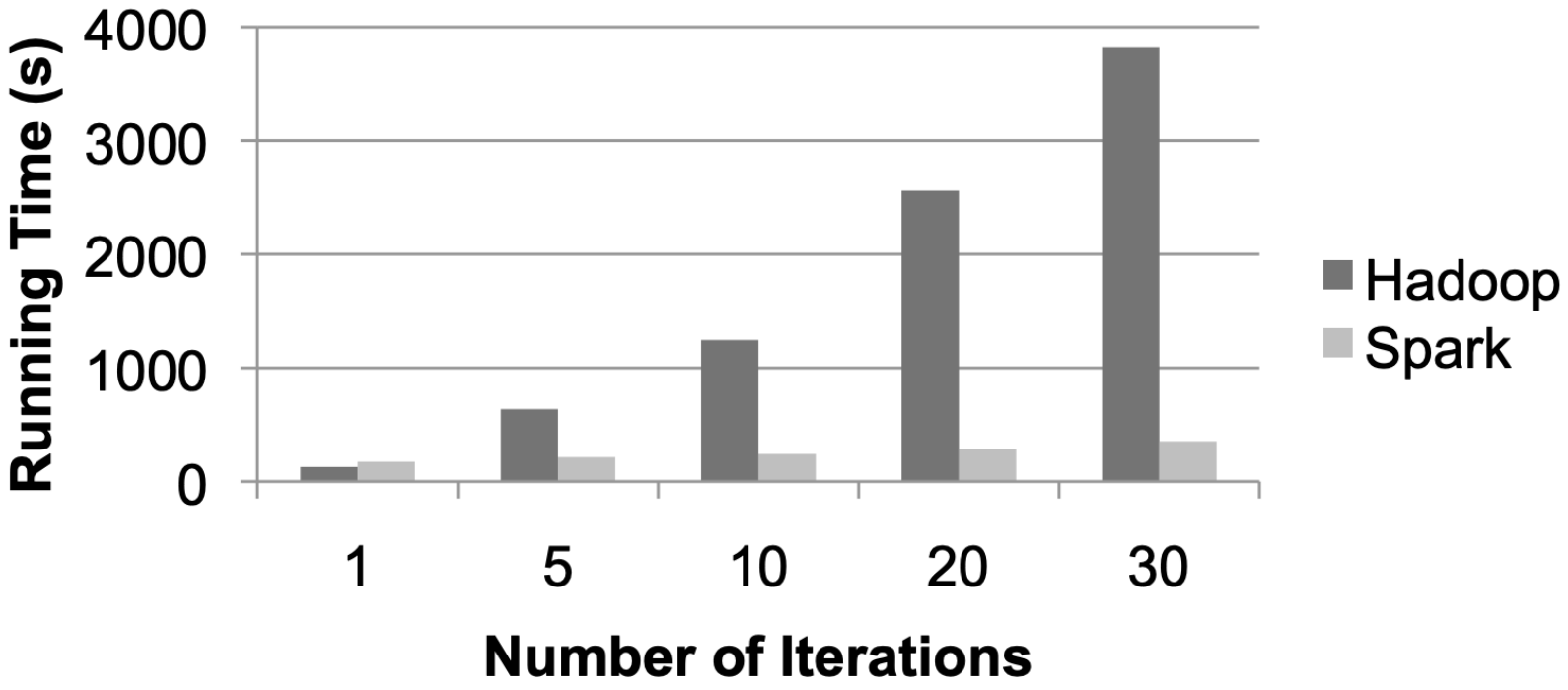


Figure courtesy of Virginia Smith

Apache Spark vs. Hadoop



Logistic regression performance in Hadoop and Spark.

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

Apache Spark: 3 Main APIs

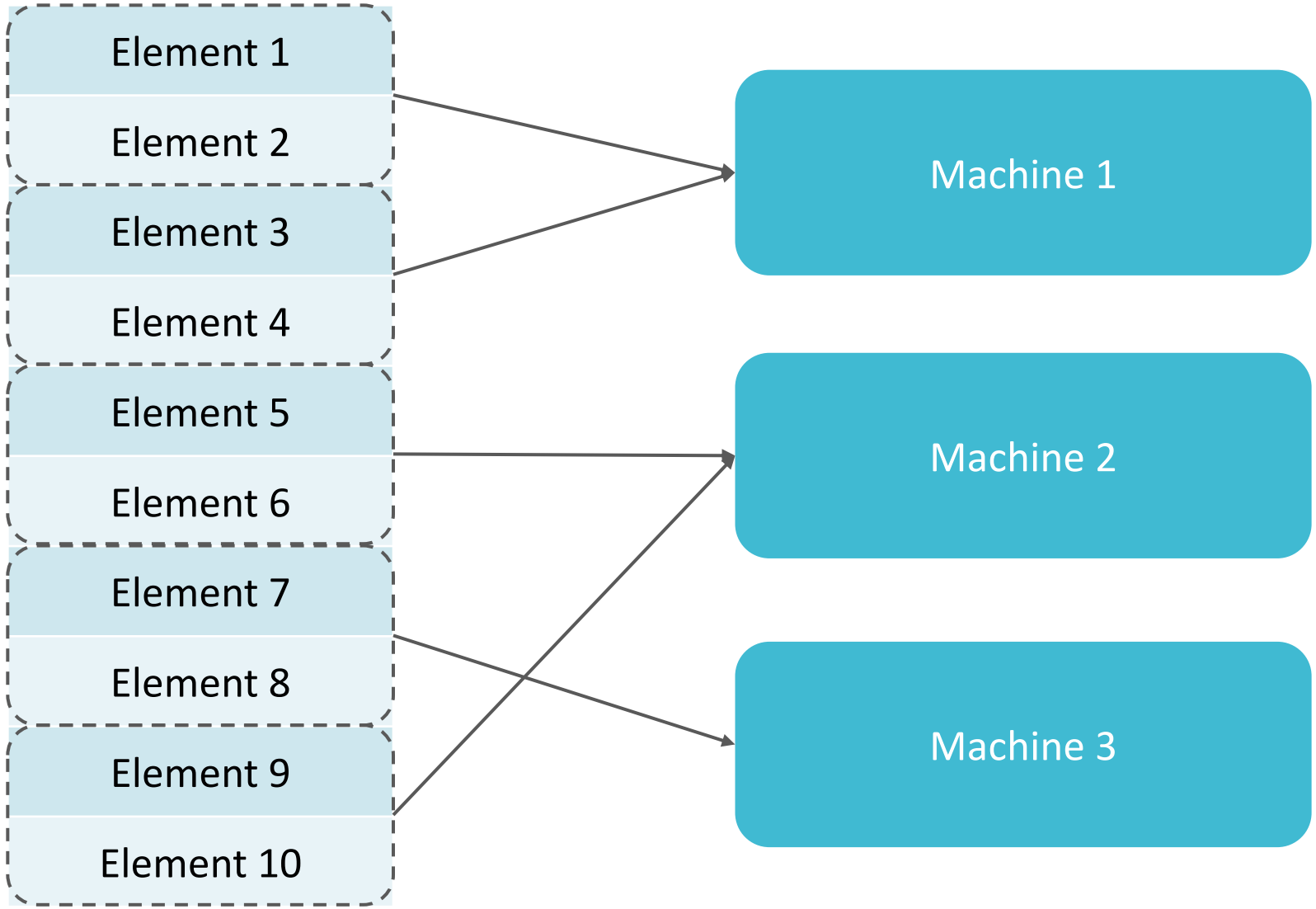
1. Resilient distributed datasets (RDDs):
   - A collection of elements partitioned across machines in a fault-tolerant way that can be operated on in parallel
2. Dataframes:
   - An abstraction on top of the RDD API
   - Similar to tables in a relational database with named and typed columns that support SQL-like queries
3. Datasets
   - A combination of RDDs and Dataframes
   - Not available in PySpark

Apache Spark: 3 Main APIs

1. **Resilient distributed datasets (RDDs):**
   - A collection of elements partitioned across machines in a fault-tolerant way that can be operated on in parallel
2. Dataframes:
   - An abstraction on top of the RDD API
   - Similar to tables in a relational database with named and typed columns that support SQL-like queries
3. Datasets
   - A combination of RDDs and Dataframes
   - Not available in PySpark

RDDs

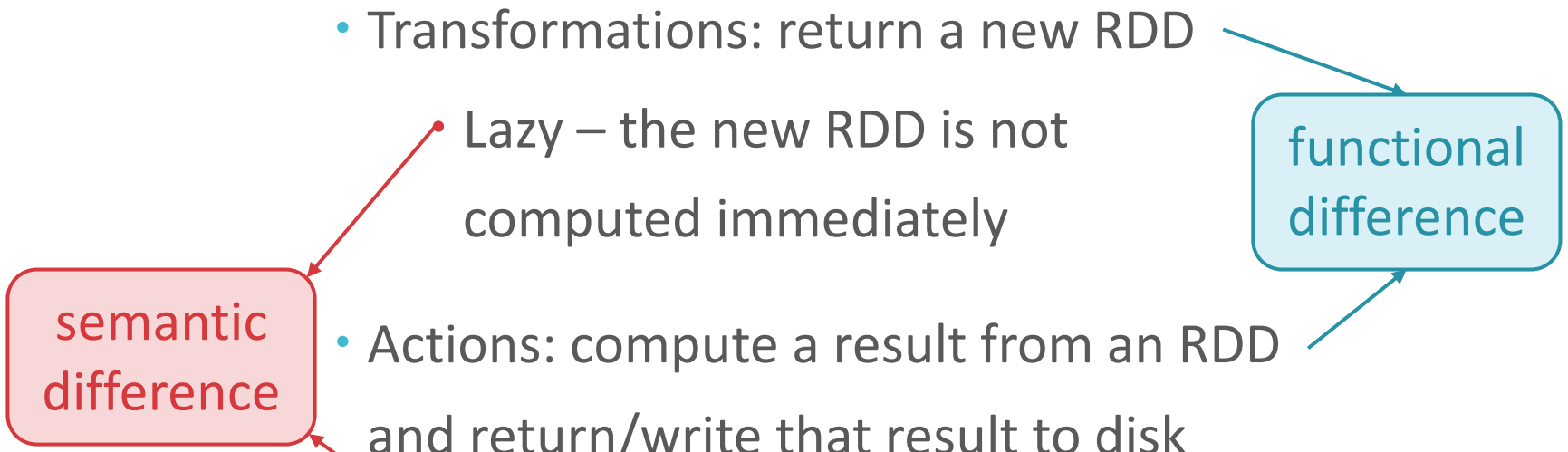
An RDD split into 5 partitions



Rule of thumb: 2-4 partitions per CPU

Operations on RDDs

- Transformations: return a new RDD
  - Lazy – the new RDD is not computed immediately
- Actions: compute a result from an RDD and return/write that result to disk
  - Eager – the result is computed immediately
- Lazy vs. eager operations allows Spark to efficiently manage data transfer



Transformations

- Transformations: return a new RDD
  - Lazy – the new RDD is not computed immediately
- Sequence of transformations defines a "recipe" for computing some result
  - Allows Spark to efficiently recover from failures/stragglers by recalling the steps required to bring a new machine up to speed

Transformations: Examples

Transformation	Description
`map(`*`func`*`)`	returns a new RDD by applying the function *`func`* to each element of the source RDD
`flatMap(`*`func`*`)`	similar to `map` except each element of the source RDD can be mapped to 0 or more outputs
`filter(`*`func`*`)`	returns a new RDD consisting of the elements where *`func`* evaluates to `TRUE`
`distinct()`	returns a new RDD consisting of the unique elements of the source RDD
`union(`*`otherRDD`*`)`	returns a new RDD consisting of the union of elements in the source RDD and *`otherRDD`*
`intersection (`*`otherRDD`*`)`	returns a new RDD consisting of the intersection of elements in the source RDD and *`otherRDD`*

Actions

- Actions: compute a result from an RDD and return/write that result to disk
  - Eager – the result is computed immediately
- Required to get Spark to “do” anything, i.e., generate the output we want

Actions: Examples

Action	Description
`reduce(`*`func`*`)`	aggregates the elements of the source RDD using the function *`func`*, a commutative and associative function that takes two arguments and returns one (allowing for parallelization)
`collect()`	returns all elements of the source RDD as an array to the driver program
`take(`*`n`*`)`	returns the first *`n`* elements of the source RDD as an array to the driver program
`first()`	returns the first element of the source RDD as an array to the driver program
`count()`	returns the number of elements in the source RDD

For A Detailed Walkthrough…

- Recitation 1 and HW1 will walk you step-by-step through…
  - The architecture of a typical Spark job
  - Lambda functions
  - RDDs and caching
  - Dataframes and schema
  - Getting setup in Databricks

Key Takeaways

- MapReduce as a framework for distributed processing
  - Heavy disk I/O → poorly suited for iterative procedures
- Apache Spark
  - Big idea: keep data in memory as much as possible
  - RDDs are the foundational API
    - Transformations (lazy) vs actions (eager)
    - Laziness/eagerness allows Spark to optimize the execution of operations

Homework 1 Programming Preview

- Part 1: Entity Resolution with RDDs
  - Identifying and linking different occurrences of the same object across multiple data sources
    - Different titles/names for the same person
    - Different pictures of the same physical object
  - Motivating example: product listings on different e-commerce websites
    - Google shopping: clickart 950000 - premier image pack (dvd-rom) massive collection of images & fonts for all your design needs …
    - Amazon: clickart 950 000 - premier image pack (dvd-rom)

Homework 1 Programming Preview

- Part 1: Entity Resolution with RDDs
  - Determine if two product listings correspond to the same product using text similarity
  - Distance between texts defined as the *cosine similarity* of their *TF-IDF representations*:
    - Similar to bag-of-words model except instead of occurrences, each word is defined by its term frequency times its inverse document frequency
    - $\mathrm{TF}(word, doc) = \frac{\#\text{ of times } word \text{ appears in } doc}{\text{total \# of words in } doc}$
    - $\mathrm{IDF}(word) = \frac{\text{total \# of documents}}{\#\text{ of documents that contain } word}$
    - Upweights words that occur a lot in a few specific documents and rarely in other documents

Homework 1 Programming Preview

- Part 2: Machine Learning Pipeline with DataFrames
  - Task: predicting power generation of power plants under different environmental conditions
  1. Data ETL (extract-transform-load): converting raw data into a workable form
  2. Exploration and visualization
  3. Modelling
  4. Hyperparameter optimization and model selection