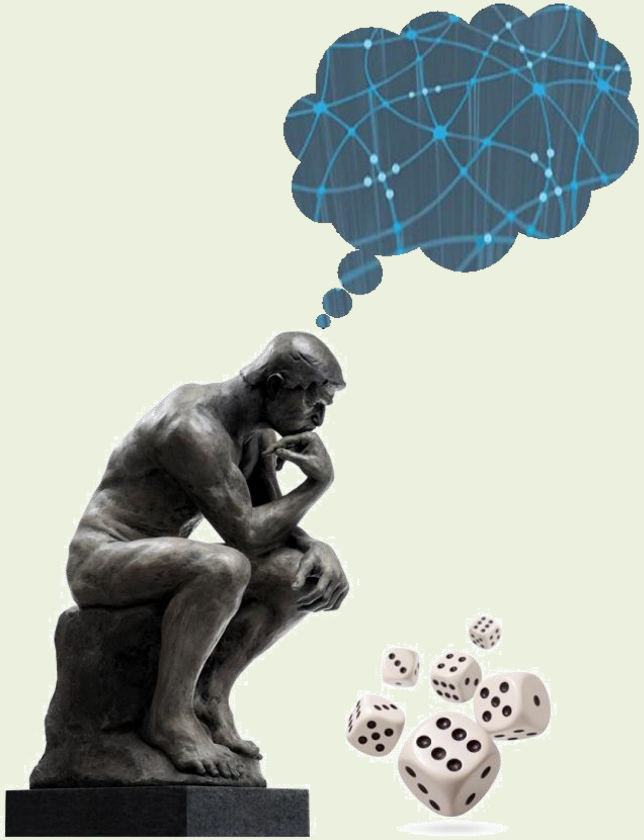
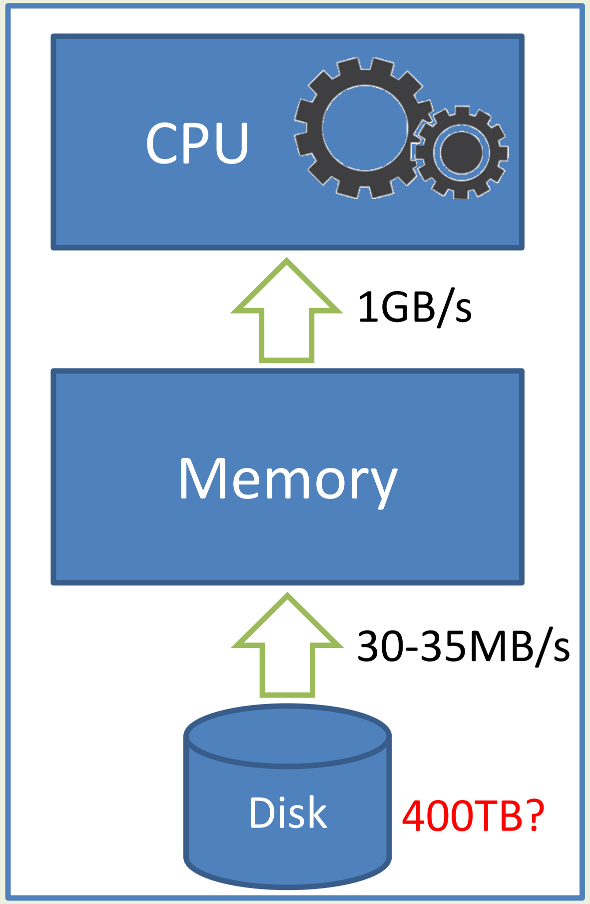


The introduction to MapReduce

Dang Ha The Hien
PhD. UiO
eSmart Systems

Single Node Architecture



Traditional data analysis and machine learning algorithms:

- Load all required data from disk to memory
- Run all the algorithms on memory

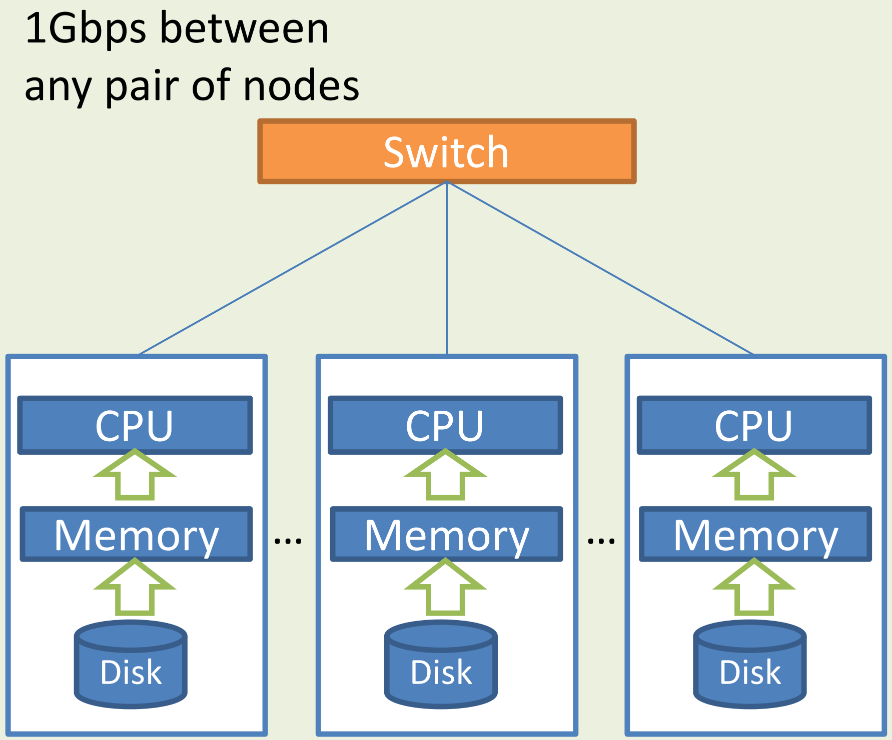
Google Problem:

- They have 20+ billion web pages
- Each web page is about 20KB (real number at the time)

→ 400 TB+ in total

- It takes ~4 months to read the data from disk to memory
- Take longer to do something useful with the data

Distribute data and computation over large cluster

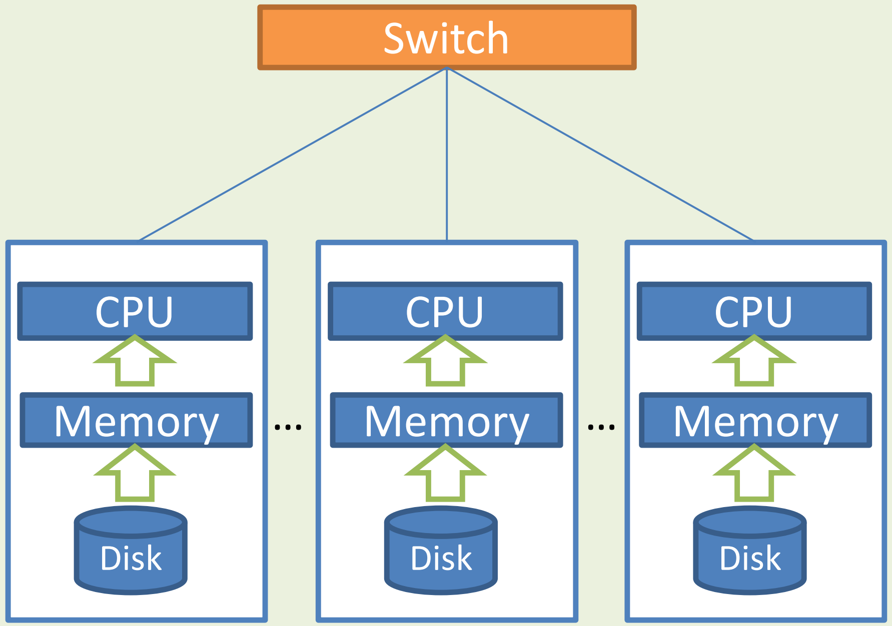


Recently standard architecture for big data problems:

- Cluster of consumer-grade hardware
- Many desktop-like Linux servers
  - Easy to add capacity
  - Cheaper per CPU/disk
  - Commodity Network (Ethernet) to connect them

Distribute data and computation over large cluster

1Gbps between
any pair of nodes

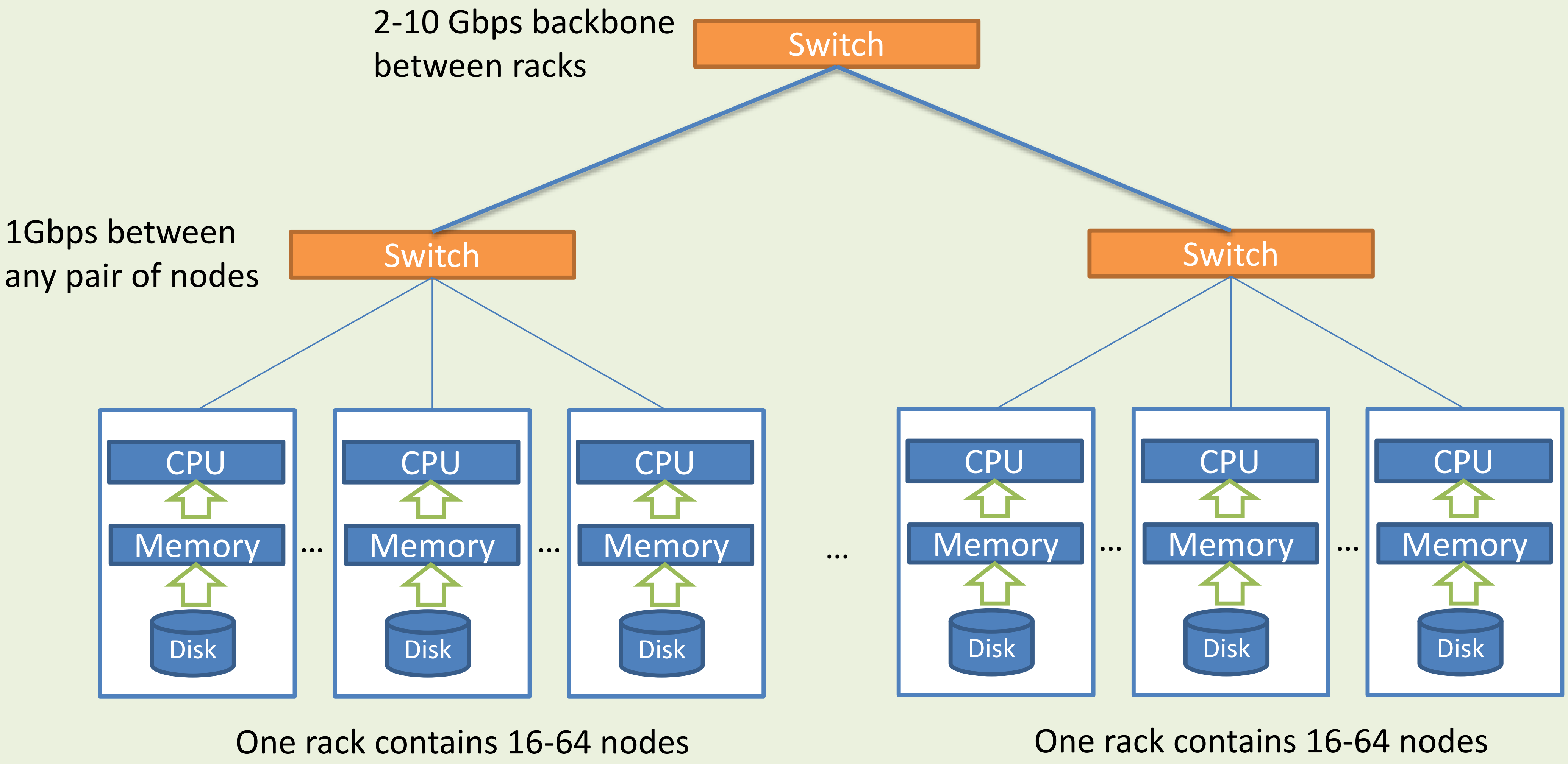


One rack contains 16-64 nodes

Each node is a cheap Linux server
Easy to add capacity
Easy to replace

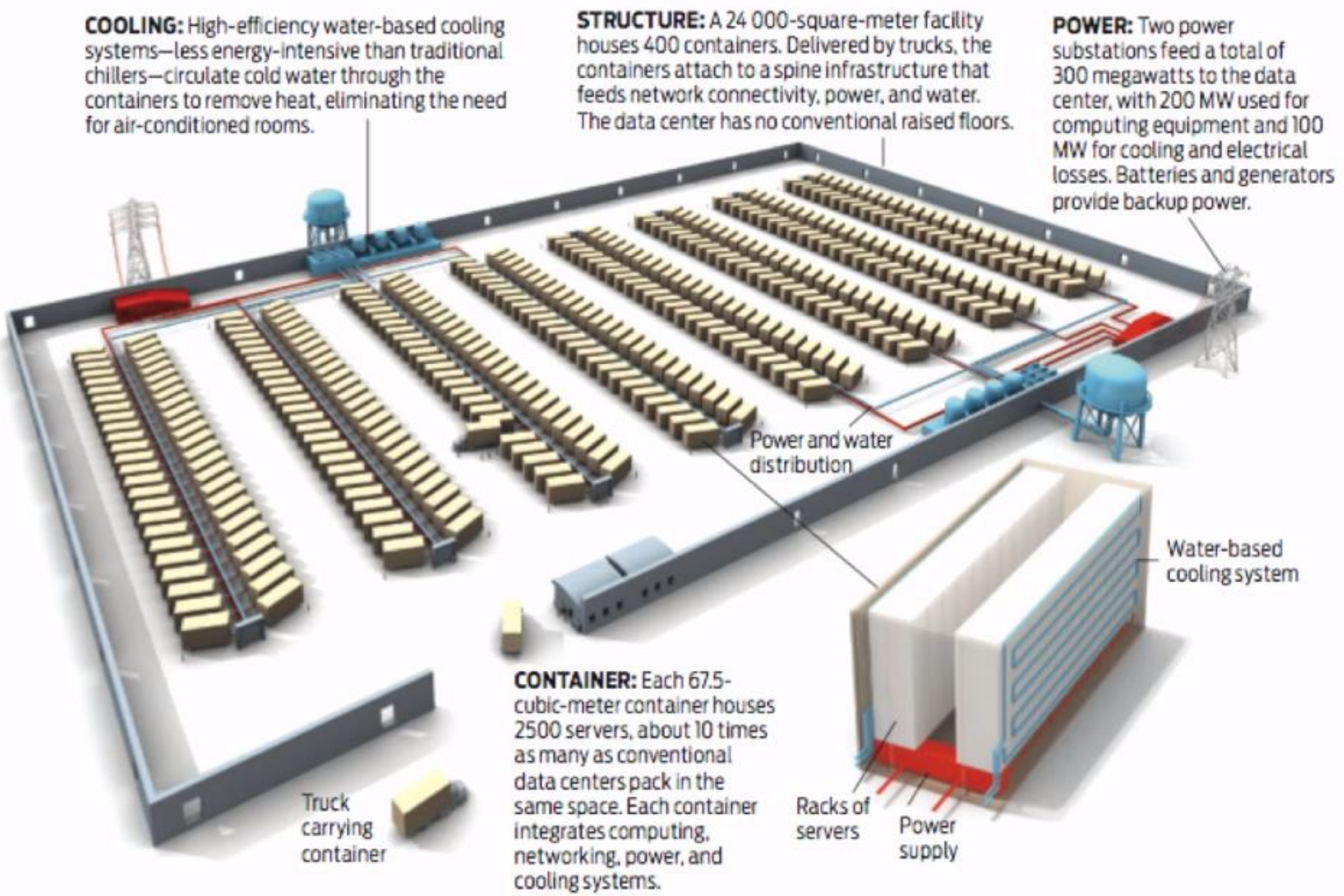
One rack can only contains around 16-64 nodes, since more than that, there will be a lot of collision in while transferring data through the switch.

Cluster Architecture



Cluster Architecture - Example

The Million-Server Data Center



http://spectrum.ieee.org/tech-talk/semiconductors/devices/what-will-the-data-center-of-the-future-look-like

In 2011, it was estimated that Google had 1M machines

Problems with Cheap Hardware

Machines fail:

- One server may stay up 3 years (1,000 days)
- If you have 1,000 servers, expect to loose 1/day
- With 1M machines, 1,000 servers fail every day!

Uneven performance:

- Those machines that don't completely fail but just ask for job and do it slowly

  (even bigger problem)

Network speed is much slower than shared memory:

- Copy data over a network takes time

Distributed Programming is hard! We need a simple model that hides most of complexity

Idea and Solution

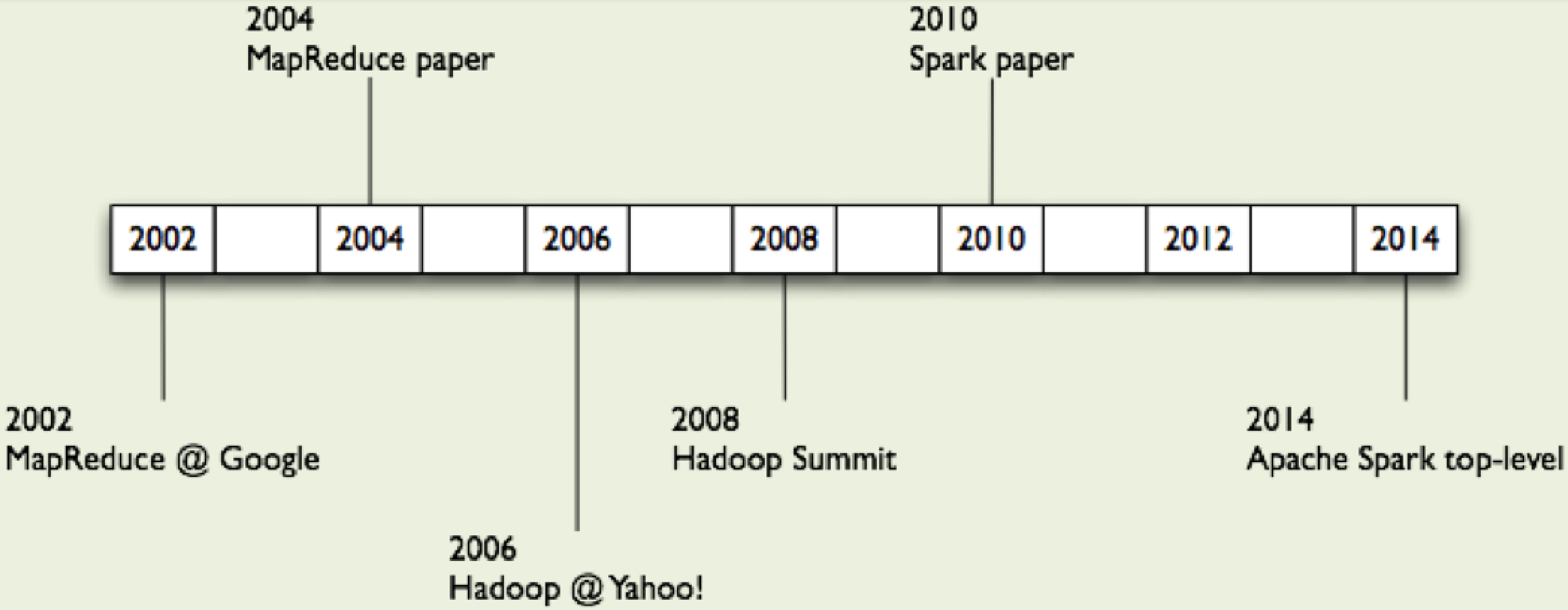
Idea:

- Store data redundantly on multiple nodes for persistence and availability
- Move computation close to data to minimize data movement
- Simple programming model to hide the complexity of all this magic

MapReduce addresses these problems:

- Storage Infrastructure (Hadoop Distributed File System- HDFS)
- Map-Reduce programming model.

History of Distributed Programming



2002: Google start using MapReduce
2004: MapReduce paper was published “Google MapReduce: Simplified Data Processing on Large Clusters”
2006: Apache Hadoop, originating from the Yahoo!’s Nutch Project
2008: Yahoo! Web scale search indexing – Hadoop Summit, Hadoop User Group
2009: Cloud computing with Amazon Web Services Elastic MapReduce,
a Hadoop version modified for Amazon Elastic Cloud Computing (EC2)

Distributed File System

Example: Hadoop's HDFS, Google's GFS

- Support redundancy and availability

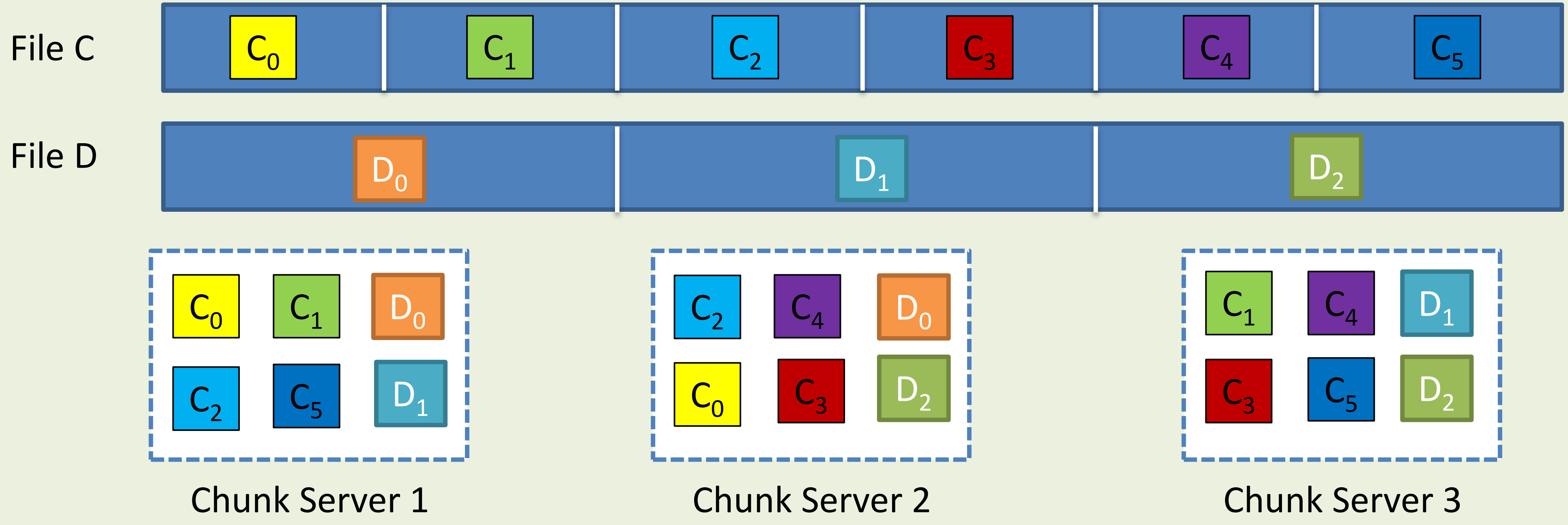
Typical usage pattern:

- Huge files (100s of GB or TB)
- Data is rarely updated in place (No random access)
- Reads and Appends are common

Once collected, you don't update data inside a file. You just append to the file, or read the whole file for analyzing.

Distributed File System

This is what happens when you store a file in an HDFS cluster:



Data kept in “chunks” spread across machines (chunk nodes)
Each chunk replicated on different machines
Chunk servers also serve as compute servers → We can bring computation to data

Distributed File System Summary

Chunk Servers:

- File split into continuous chunks (16-64MBs)
- Each chunk replicated (2x or 3x)
- Try to keep at least 1 replica in a different rack

Master node:

- Name node in Hadoop's HDFS
- Stores metadata about where files are stored
- Might be replicated (but usually not)

Client library for file access:

- Talk to master to find chunk servers
- Connects directly to chunk servers to access data

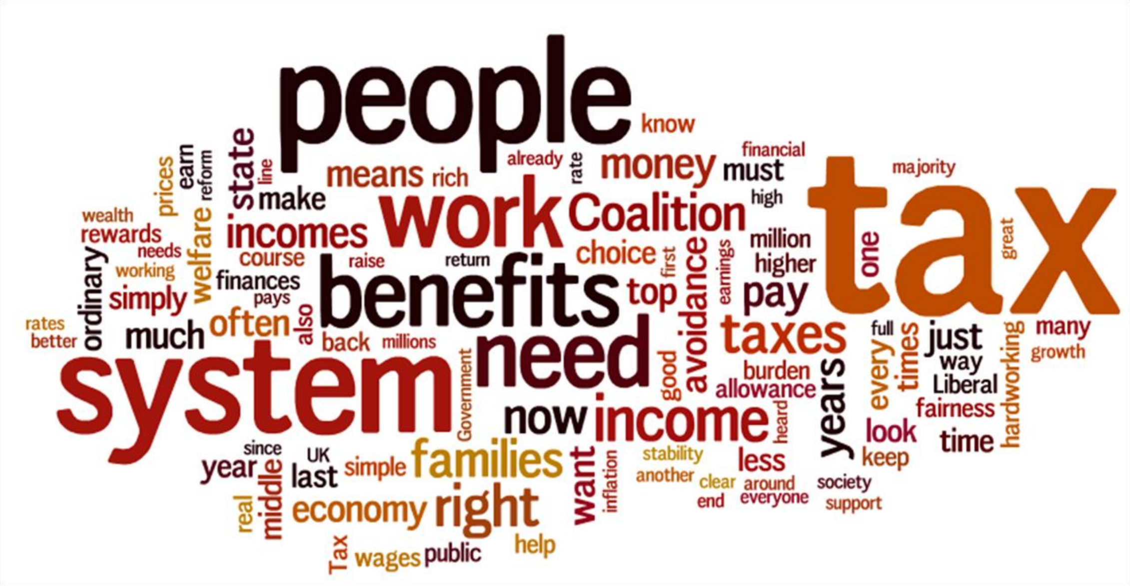
Programming Model: MapReduce

Classic Motivation Example:

- You have a huge text document
- Want to count the number of times each distinct word appears in the file

Potential Applications:

- Draw a wordcloud like this
- Analyze web server log files to find popular URLs



Simple Approach: Use a Hash Table

"I am Sam
I am Sam
Sam I am
Do you like
Green eggs and ham
I do not like them
Sam I am
I do not like
Green eggs and ham
Would you like them
Here or there?
…"

Key	Value
I	2
am	1
Sam	1

Hash table is a data structure that can map keys to values

Simple word count approach using hash table:

- Start with empty hash table.
- Read words sequentially
- For each word:
  - If I can't see that word as a key in the hash table
    - Add that word as a key with value = 1
  - Otherwise, increase the value of that word to +1

What if the Document is Really Big?

What if the Document is Really Big

"I am Sam
I am Sam
Sam I am

Do you like
Green eggs and ham
I do not like them

Sam I am
I do not like
Green eggs and ham

Would you like them
Here or there?
…"

Machine 1

Key	Value
I	3
am	3
Sam	3

Machine 2

Do	1
…	…

Machine 3

Sam	1
…	…

Machine 4

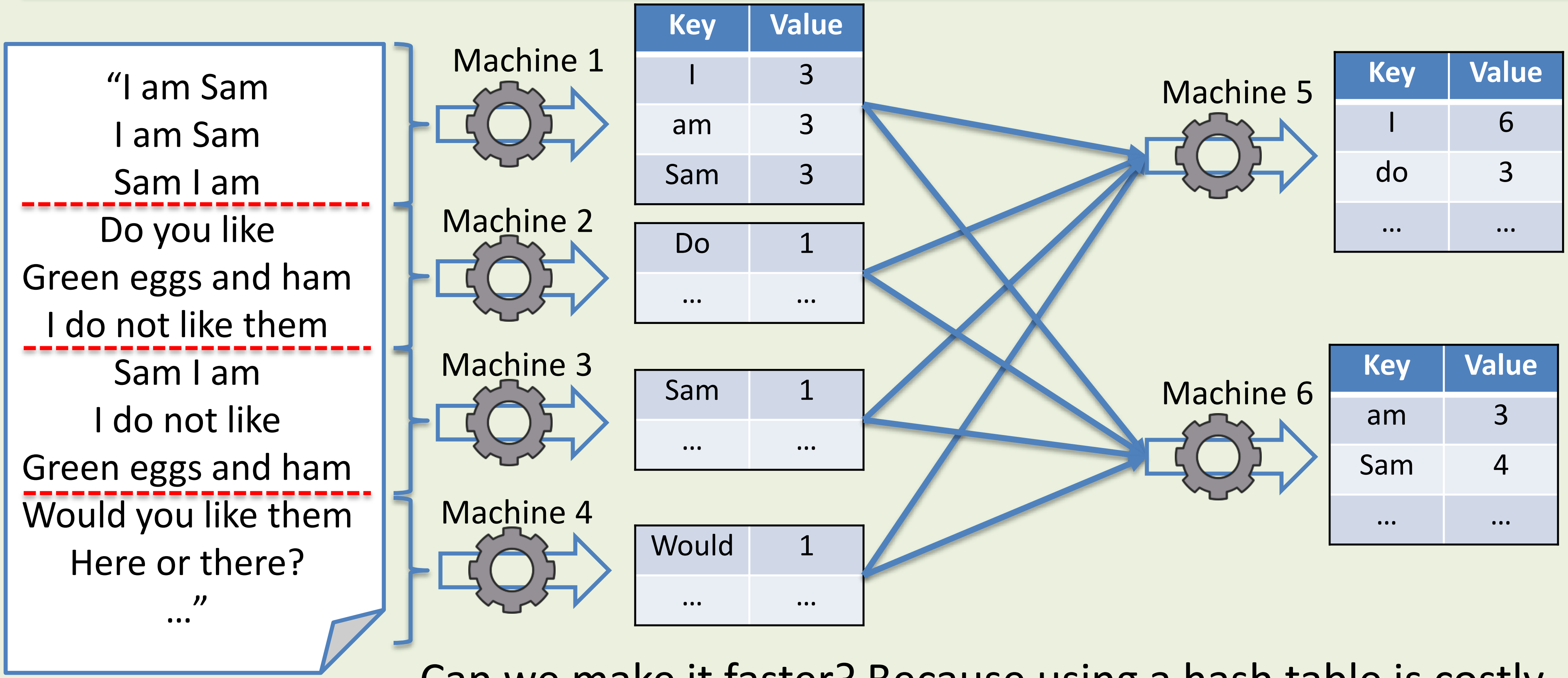
Would	1
…	…

Machine 5

Key	Value
I	6
am	4
Sam	4
do	3
…	…

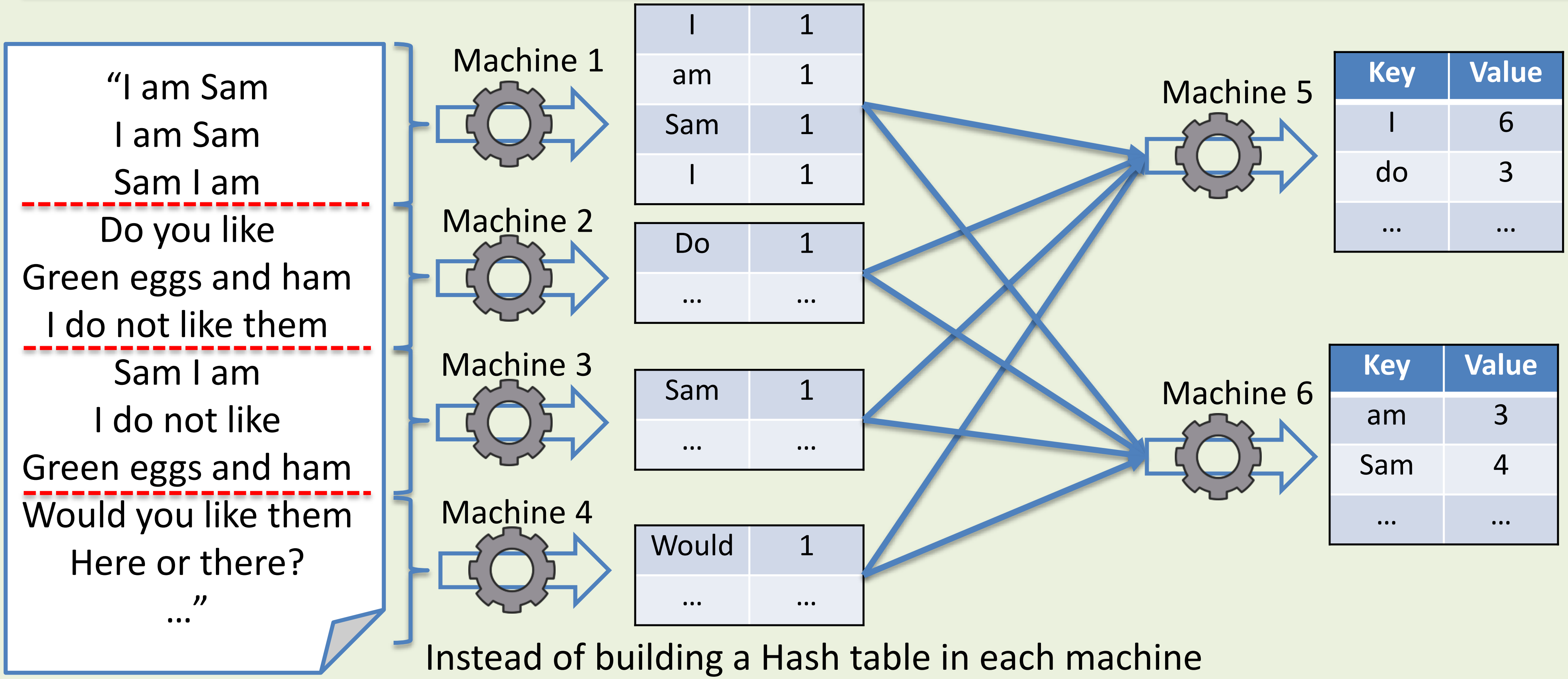
What if the Document is really Big and we have bottleneck at Machine 5?
Or if the result too big that cannot even fit into one machine memory?

What if the Document is Really Big?



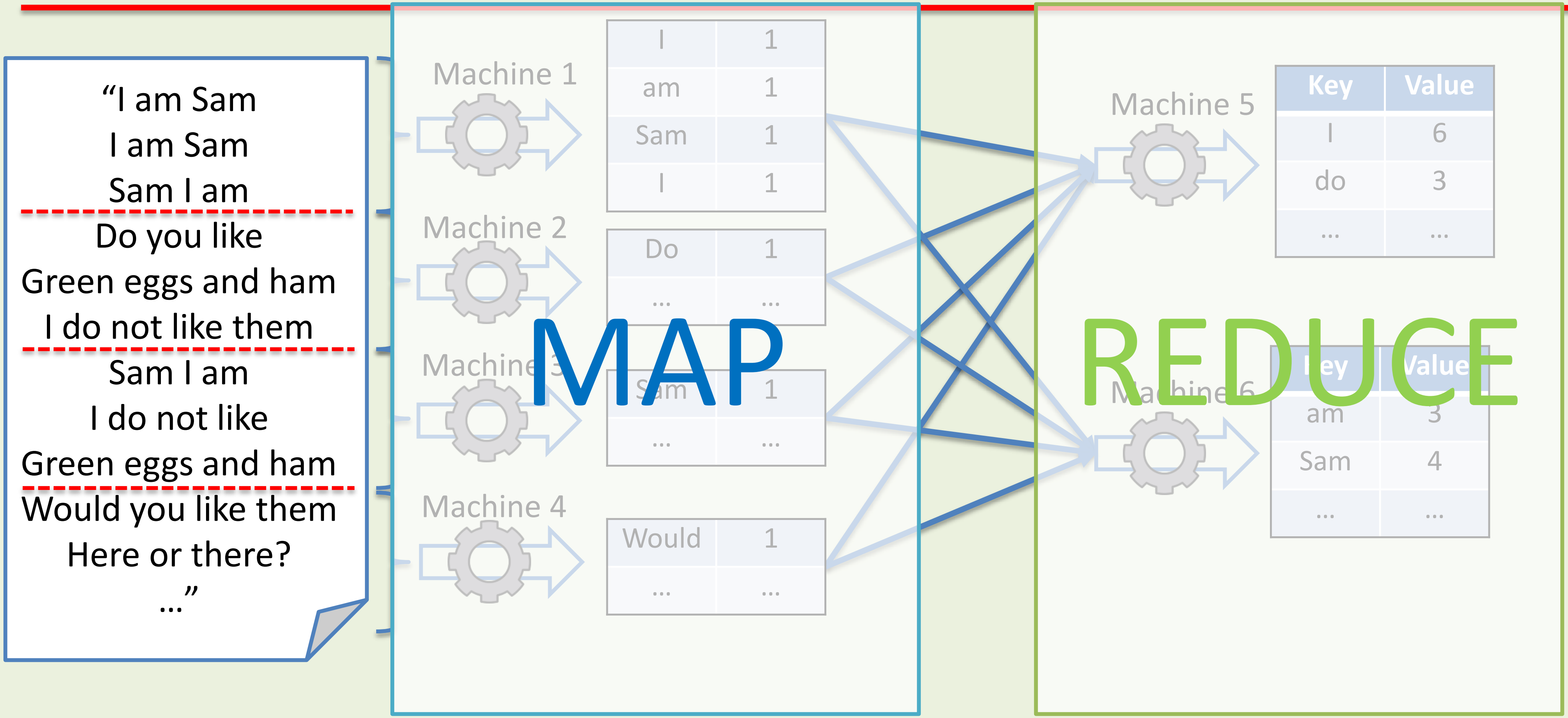
Can we make it faster? Because using a hash table is costly
(need to call the hash function for every access)

What if the Document is Really Big?



Instead of building a Hash table in each machine
why don’t we just create <key, value> pairs, with key can be duplicated
We will group them by key later any way, so we don’t have to do that multiple times

What if the Document is Really Big?



Why MapReduce for cluster programming model?

When an algorithm/program is expressed in terms of a Map and Reduce operations, it's simple to build an execution engine that:

- Optimize the computing schedule
- Handle failures
- Optimize inter-machine communication

It's powerful enough to express most of data manipulation jobs

- All dplyr verbs can be easily expressed in term of Map Reduce operations
- Most of SQL statements can be translated into set of Map Reduce operations
- Matrix Multiplication can be expressed naturally using Map Reduce

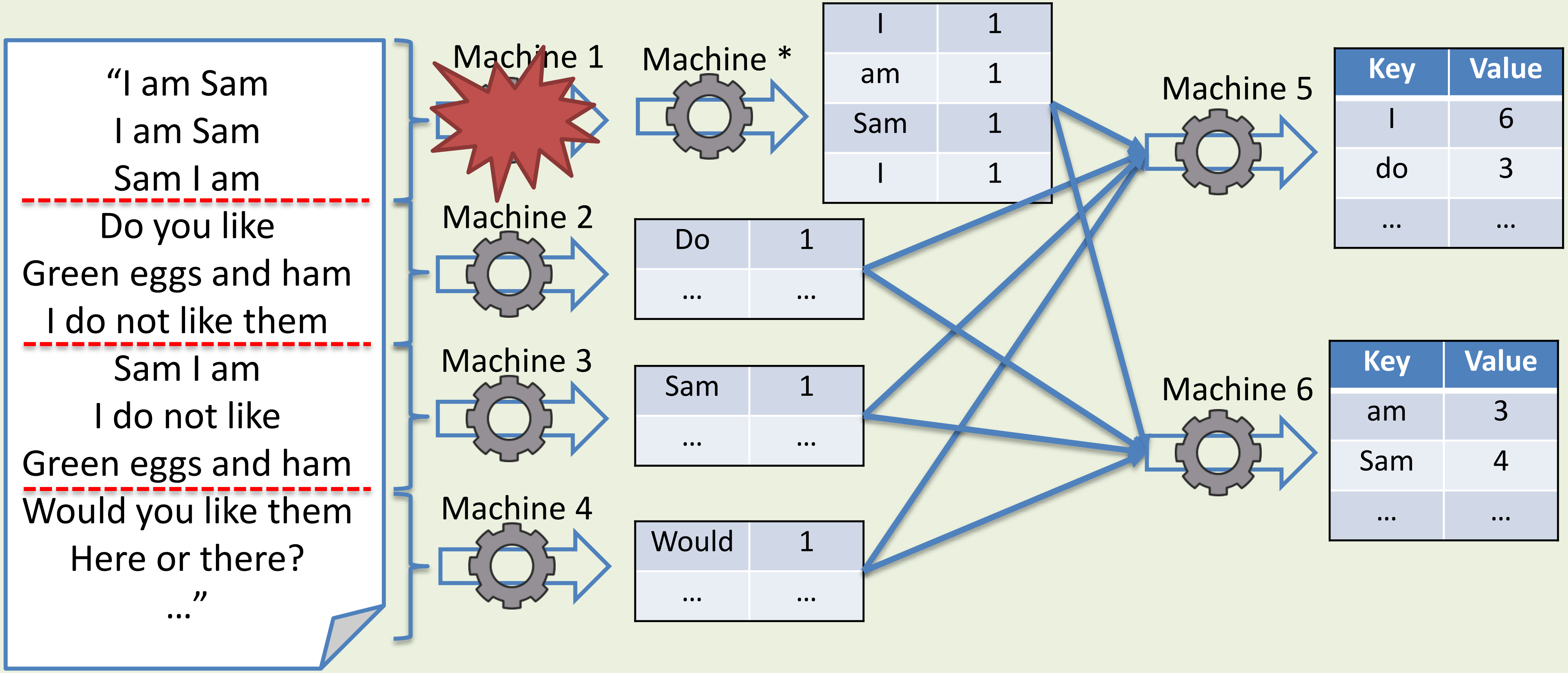
Coordination: Master

Master node takes care of coordination:

- **Task status**: (idle, in-process, completed)
- **Idle tasks** get scheduled as workers become available
- When a map task completes, it sends the master the location and sizes of its intermediate files, one for each reducer
- Master pushes this info to reducers

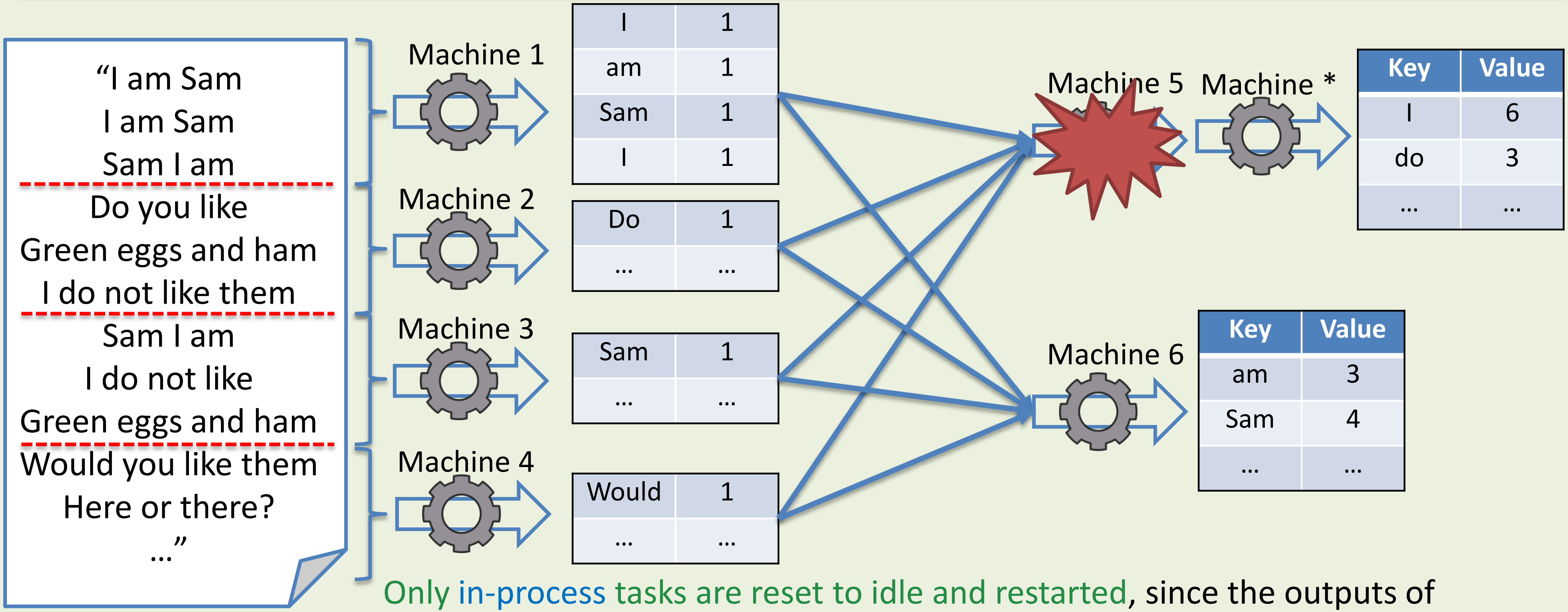
Master pings workers periodically to detect failures

Dealing with Map Worker Failures



Map tasks completed or in-process at the fail worker are reset to idle
These tasks are rescheduled on another worker

Dealing with Reduce Worker Failures



Only in-process tasks are reset to idle and restarted, since the outputs of completed tasks are stored redundantly in HDFS file system

Dealing with Master failure

- MapReduce task is aborted and client is notified.
- There is only one master node, so the chance that it fails is much smaller than the chance that a worker fail

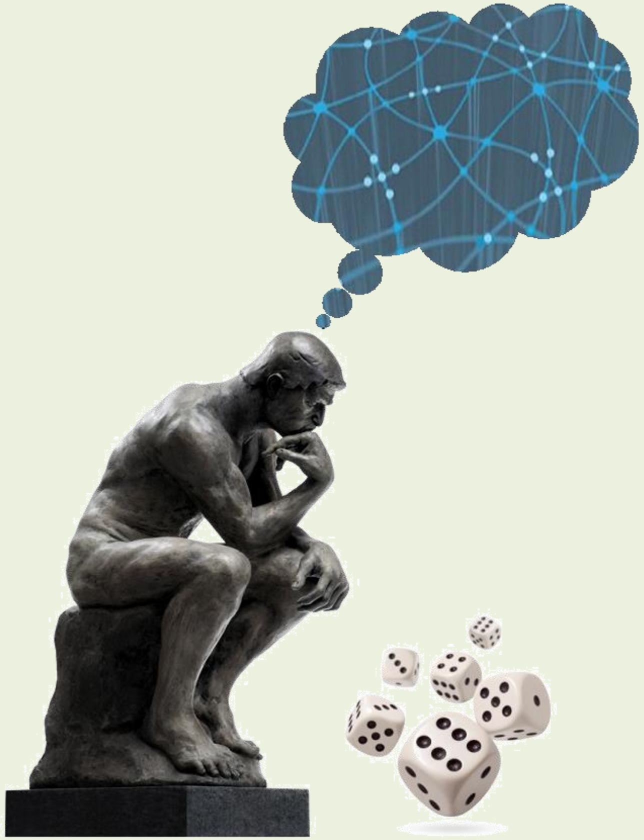
Problems suitable / NOT suitable for MapReduce

MapReduce is great for:

- Problems that require sequential data access
- Large batch jobs (not interactive, real-time)

MapReduce is inefficient for problems where random (or irregular) access to data required:

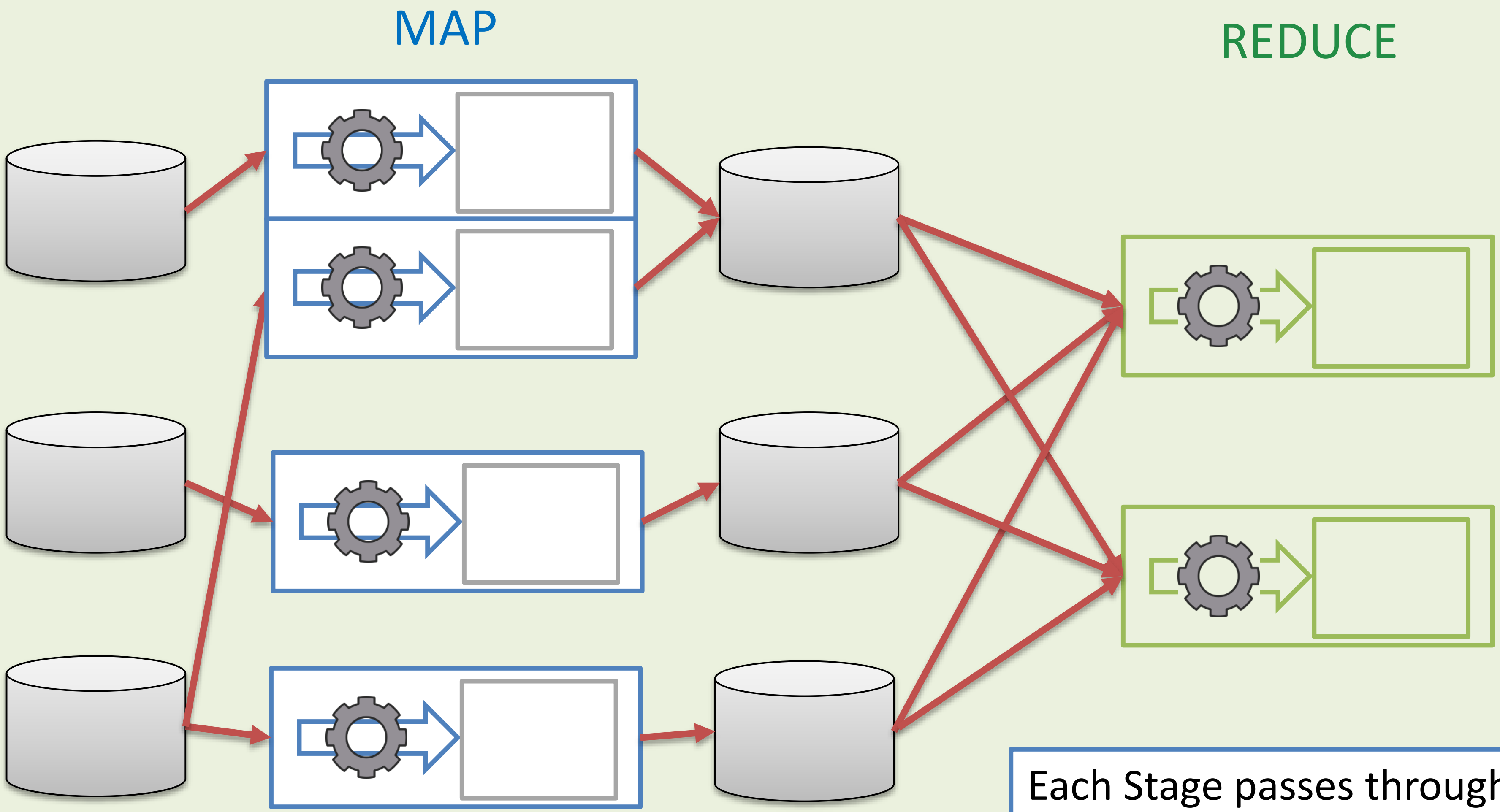
- Graphs
- Iterative algorithms (machine learning)



The Introduction to Apache Spark

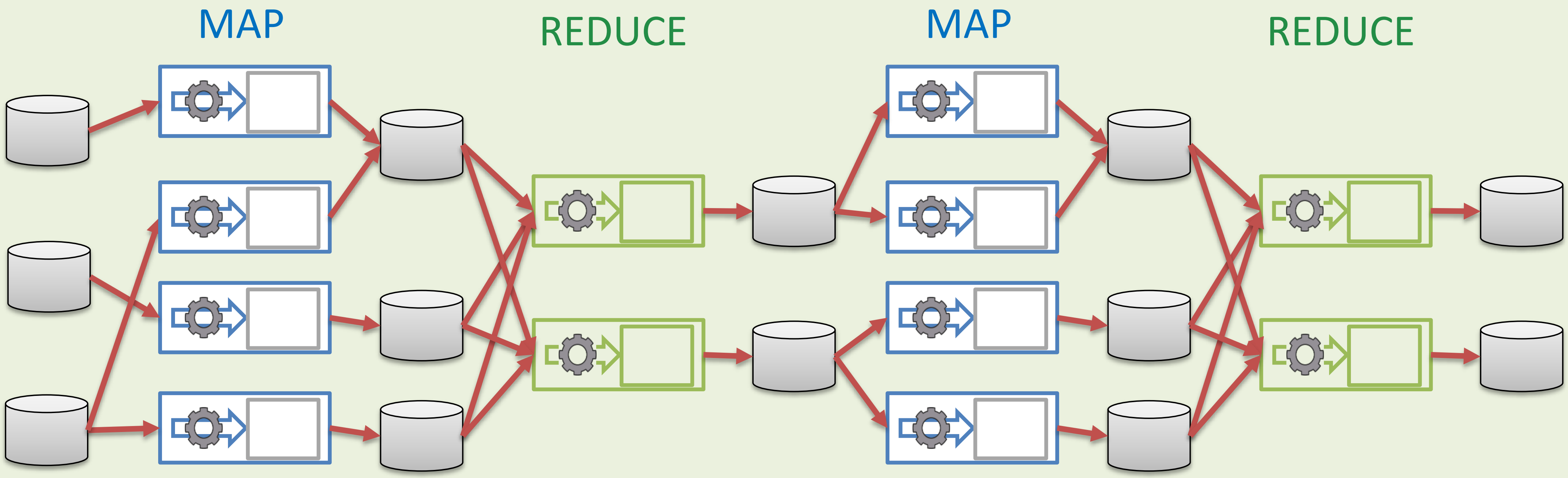
Dang Ha The Hien
PhD. UiO
eSmart Systems

MapReduce Problem



Each Stage passes through the hard drives

MapReduce Problem: Iterative Jobs



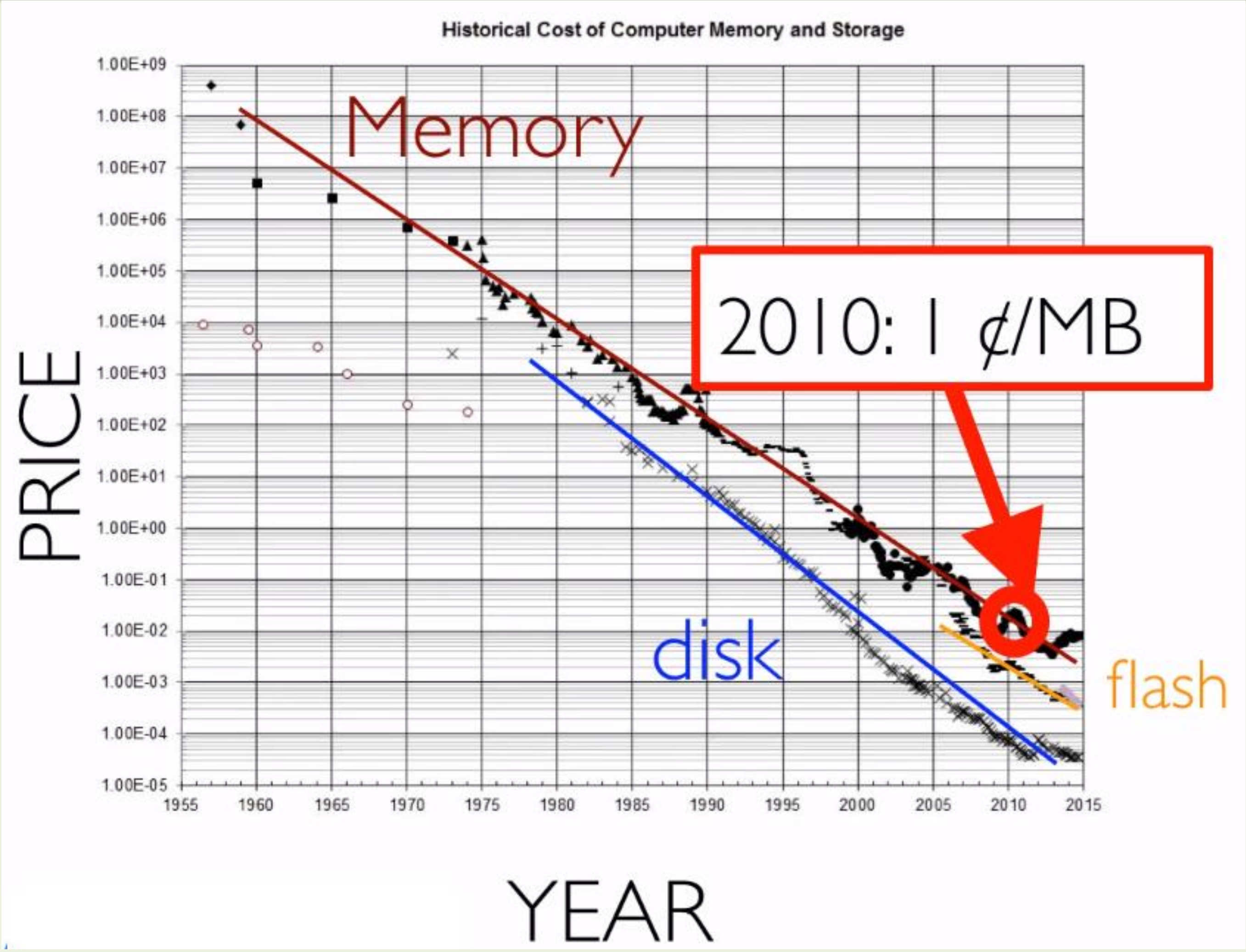
Iterative jobs involves a lot of disk I/O for each repetition

Disk I/O is very slow!

Spark Motivation

- Keep more data in-memory
- Support better programming interfaces
- Create new distributed execution engine:

Tech Trend: Cost of Memory

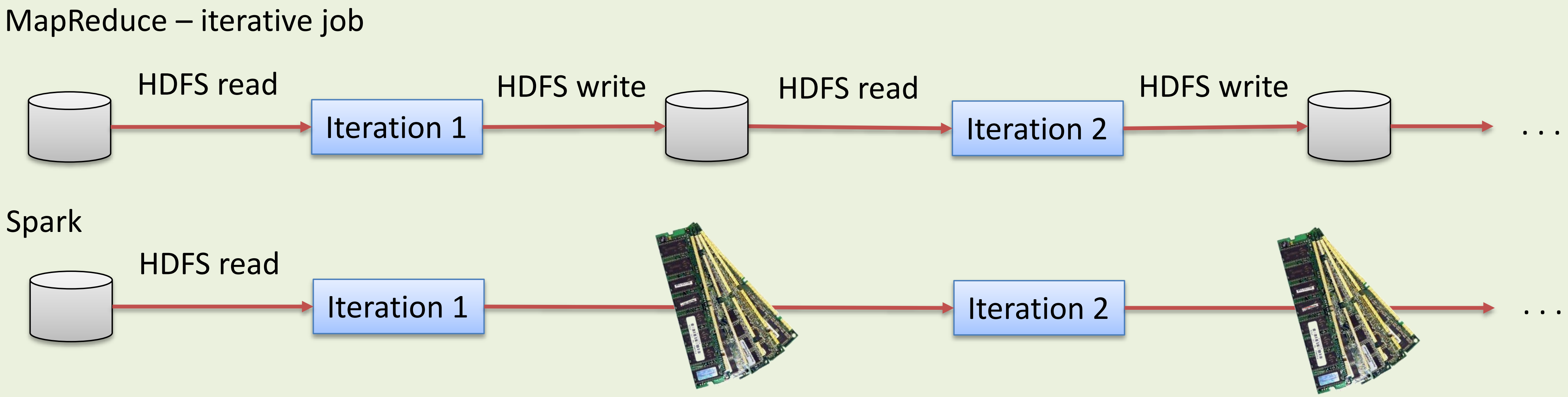


Lower cost means you can put more memory in each server

In-memory database solution

Data centers that use main memory exclusively

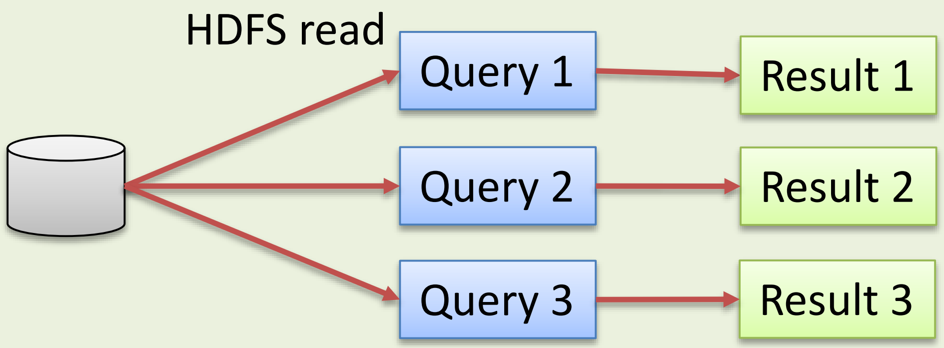
Use Memory Instead of Disk



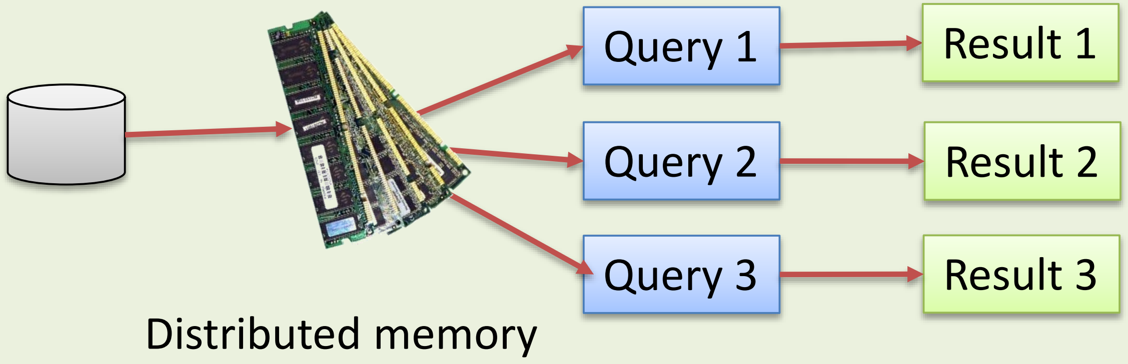
Memory: 10-100 times faster than disk or network

Use Memory Instead of Disk

MapReduce – interactive query

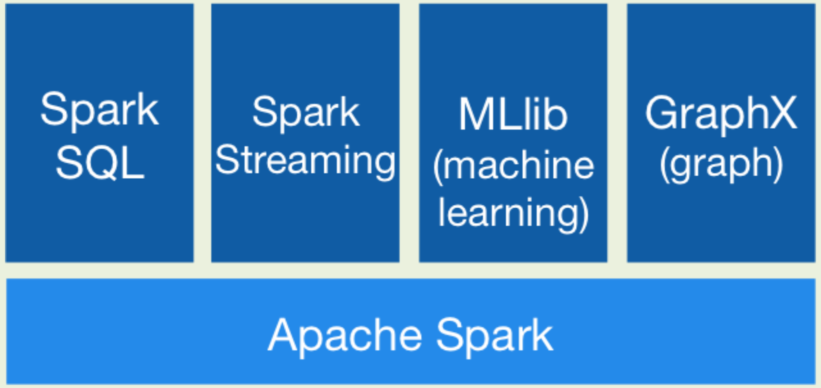


Spark



Memory: 10-100 times faster than disk or network

Spark Tools



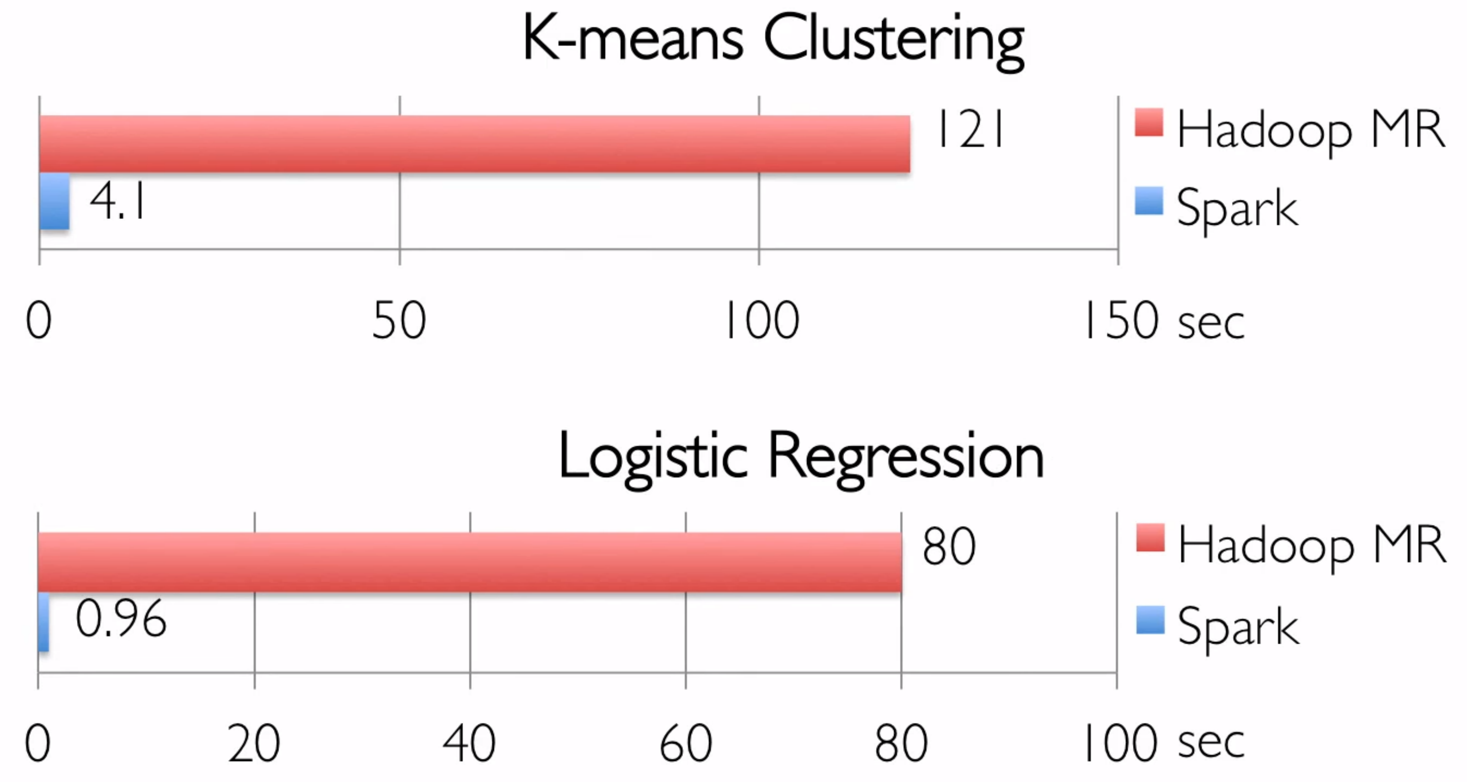
Spark SQL:	Let you query structured data using either SQL or recently DataFrame API (dplyr-like) Can connect to many different data sources: CSV, JSON, JDBC, ODBC
Spark Streaming:	Stream processing
MLlib:	Implementation of many scalable machine learning algorithms
GraphX:	Support variety of graph processing and graph algorithms

Spark and MapReduce Differences

	Hadoop Map Reduce	Spark
Storage	Disk only	In-memory or on disk
Operations	Map and Reduce	Map, Reduce, Join, Sample, …
Execution model	Batch	Batch, interactive, real-time,…
Programming environments	Java	Scala, Java, Python, R
Programming Interface	Map and Reduce functions	DataFrame Interface SQL-like interface

In-Memory Can Make a Big Difference!

Can achieve 100x speed up!



In-Memory Can Make a Big Difference!

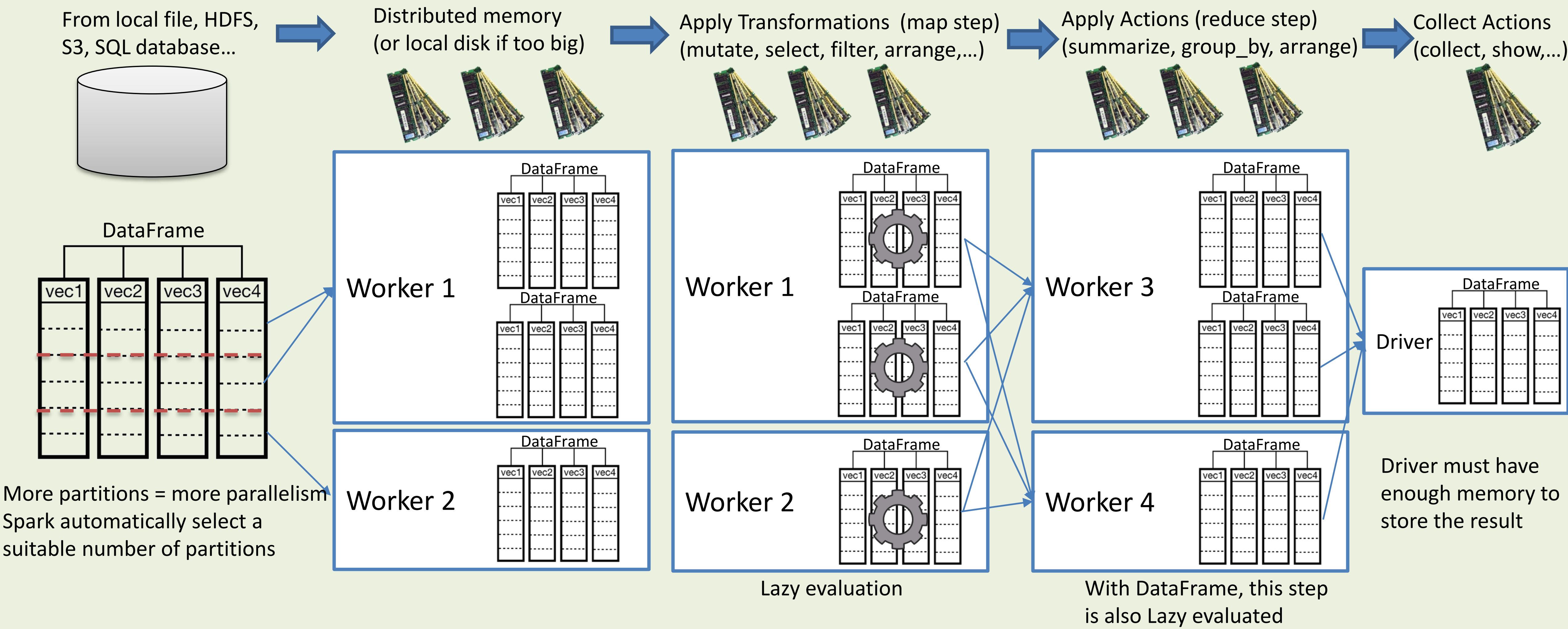
First Public Cloud Petabyte Sort

	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

Constructing and Working with DataFrames




thank you
danke
рахмат
謝謝
ngiyabonga
teşekkür ederim
Баярлалаа
mersi
kia ora
barka
welalin
tack
spas
faafetai lava
vinaka
blagodaram
dank je
misaotra
matondo
paldies
grazzi
mahalo
tanadh leat
kiitos
dankie
спасибі
dhanyavad
gracias
gracie
hvala
māuruuru
köszönöm
dankon
aciu
akun
djiere dieuf
tau
дякую
chnorakaloutioun
gratias ago
gràcies
sulpáy
dziękuję
děkuji
sagolun
sukriya
kop khun krap
taiku
arigatô
takk
dakujem
mèsi
didi madloba
najis tuke
ありがとう
tanemirt
grazie
kam sah hamnida
rahmat
terima kasih
rahmet
diolch
dhanyavadagalu
shukriya
merce
감사합니다
xièxie
ευχαριστώ
merci