

Introduction to Apache Spark



This Lecture

Big Data Problems: Distributing Work, Failures, Slow Machines

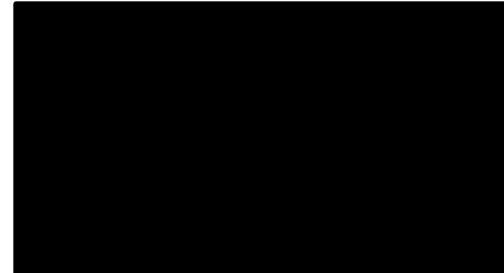
HW/SW for Big Data: Map Reduce and Apache Spark

The Structured Query Language (SQL)

SparkSQL

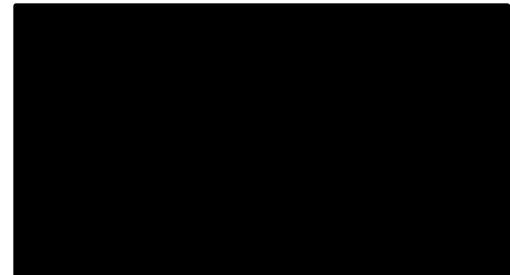
Apache Spark Resources and Community

Apache Web Server Log Files



What is Apache Spark?

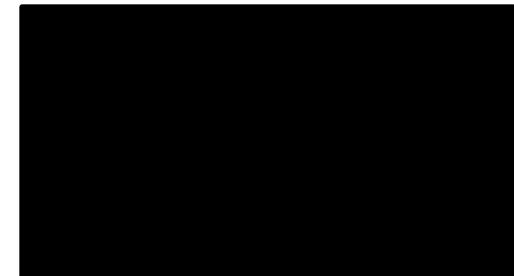
Scalable, efficient analysis of Big Data



What is Apache Spark?

Scalable, efficient analysis of Big Data

This lecture

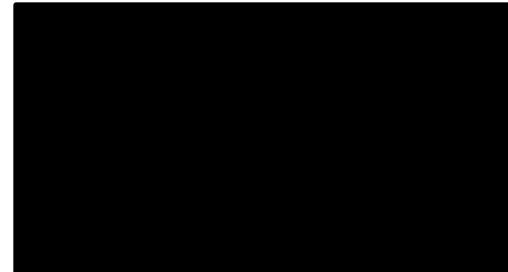


The Big Data Problem

Data growing faster than CPU speeds

Data growing faster than per-machine storage

Can't process or store all data on one machine



Google Datacenter

How do we program this thing?

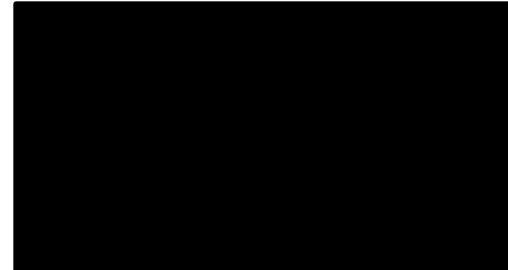
The Opportunity

Cloud computing is a game-changer!

Provides access to low-cost computing and storage

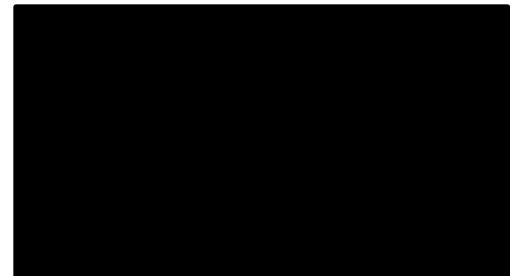
Costs decreasing every year

The challenge is programming the resources

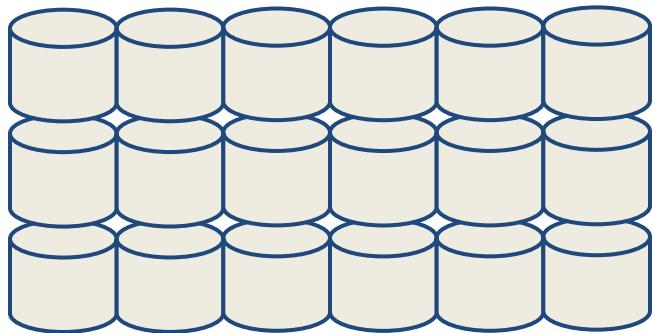


What is Apache Spark?

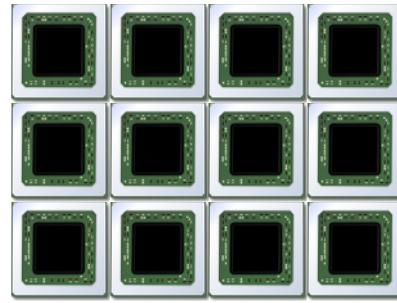
- Scalable, efficient analysis of Big Data



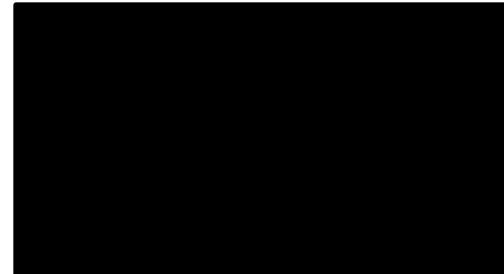
A Brief History of Big Data Processing



Lots of hard drives



... and CPUs



Yesterday's Hardware for Big Data

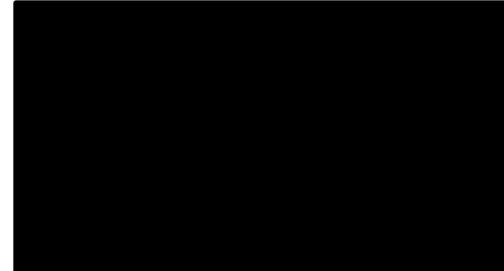
One big box!
(1990's solution)

- » All processors share memory

Very expensive

- » Low volume
- » All “premium” hardware

And, still not big enough!



Hardware for Big Data

Consumer-grade hardware
Not “gold plated”

Many desktop-like servers

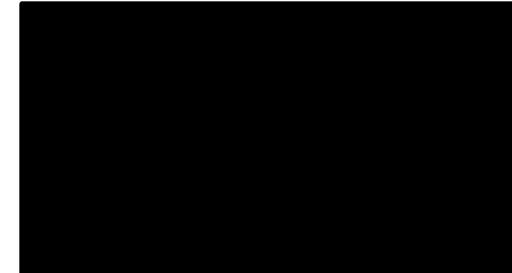
Easy to add capacity

Cheaper per CPU/disk

But, requires complexity in software



Image: Steve Jurvetson/Flickr



Problems with Cheap Hardware

Failures, Google's numbers:

1-5% hard drives/year

0.2% DIMMs/year



Facebook Datacenter (2014)

Network speeds versus shared memory

Much more latency

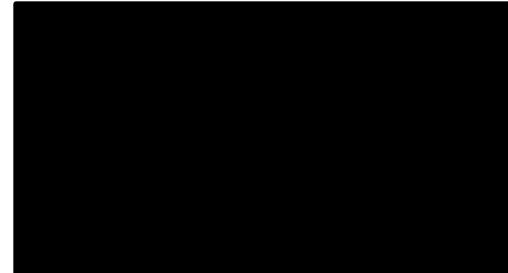
Network slower than storage

Uneven performance

What's Hard About Cluster Computing?

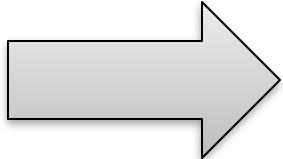
How do we split work across machines?

Let's look at a simple task: word counting



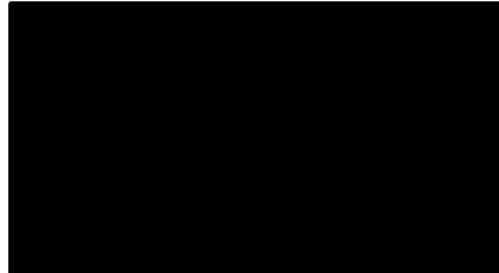
How do you count the number of occurrences of each word
in a document?

“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

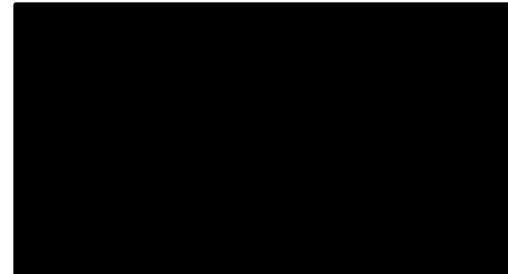
{}

I am Sam

Sam I am

Do you like

Green eggs and ham?”



One Approach: Use a Hash Table

“I am Sam

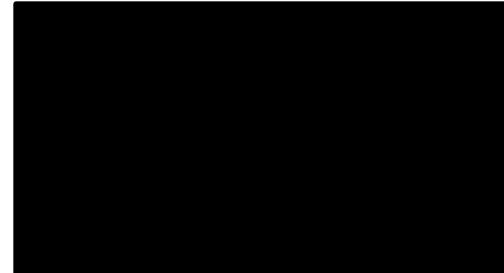
I am Sam

Sam I am

Do you like

Green eggs and ham?”

{I :1}



One Approach: Use a Hash Table

“I **am** Sam

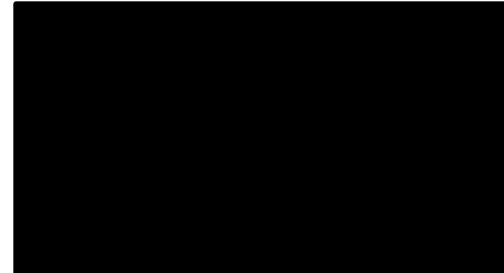
I am Sam

Sam I am

Do you like

Green eggs and ham?”

{|: 1,
am: 1}



One Approach: Use a Hash Table

“I am **Sam**

I am Sam

Sam I am

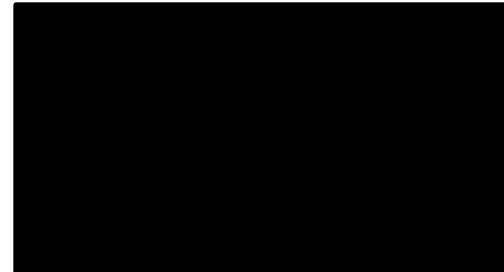
Do you like

Green eggs and ham?”

{|: 1,

am: 1,

Sam: 1}



One Approach: Use a Hash Table

“I am Sam

I am Sam

Sam I am

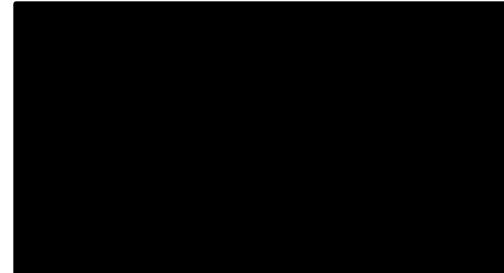
Do you like

Green eggs and ham?”

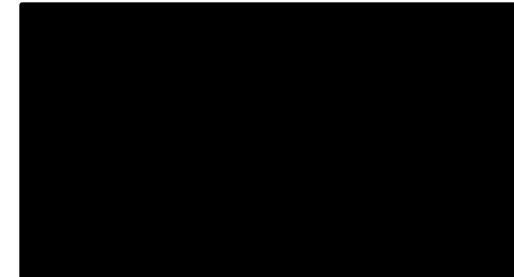
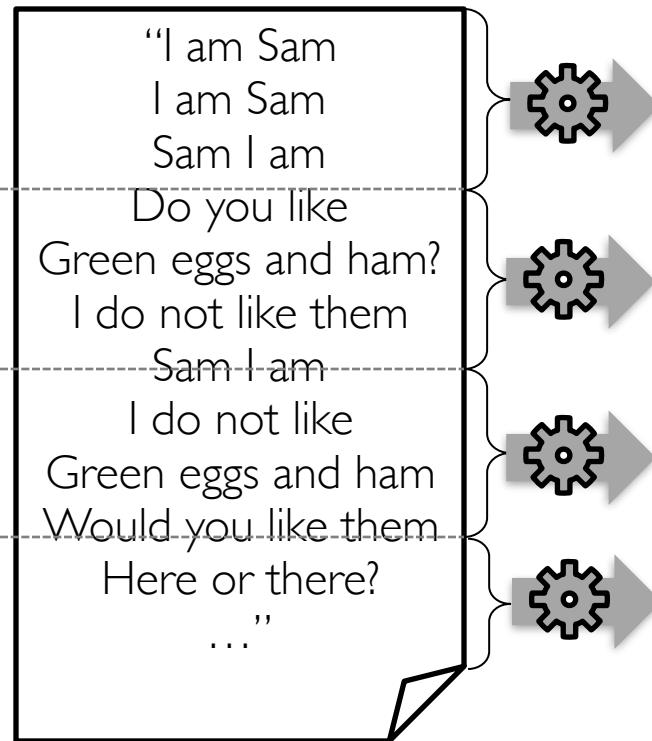
{|: 2,

am: 1,

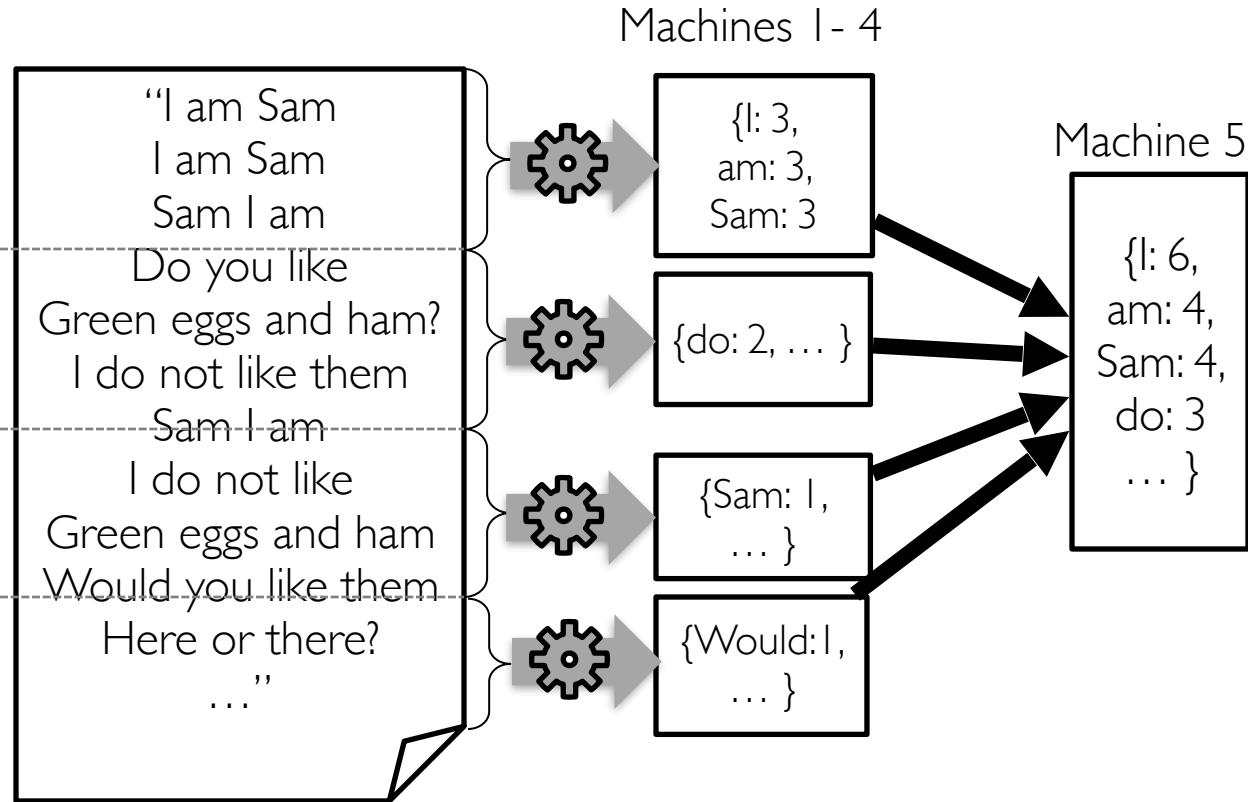
Sam: 1}



What if the Document is Really Big?

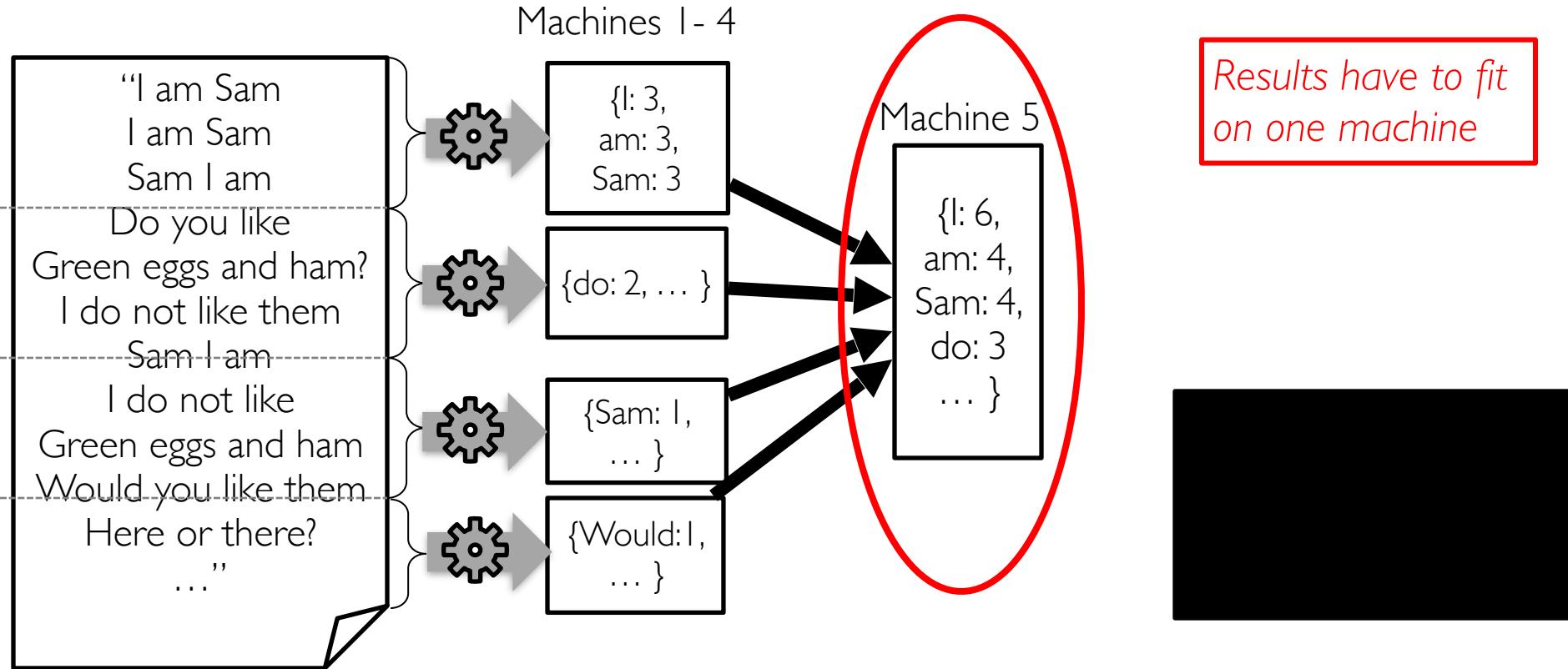


What if the Document is Really Big?

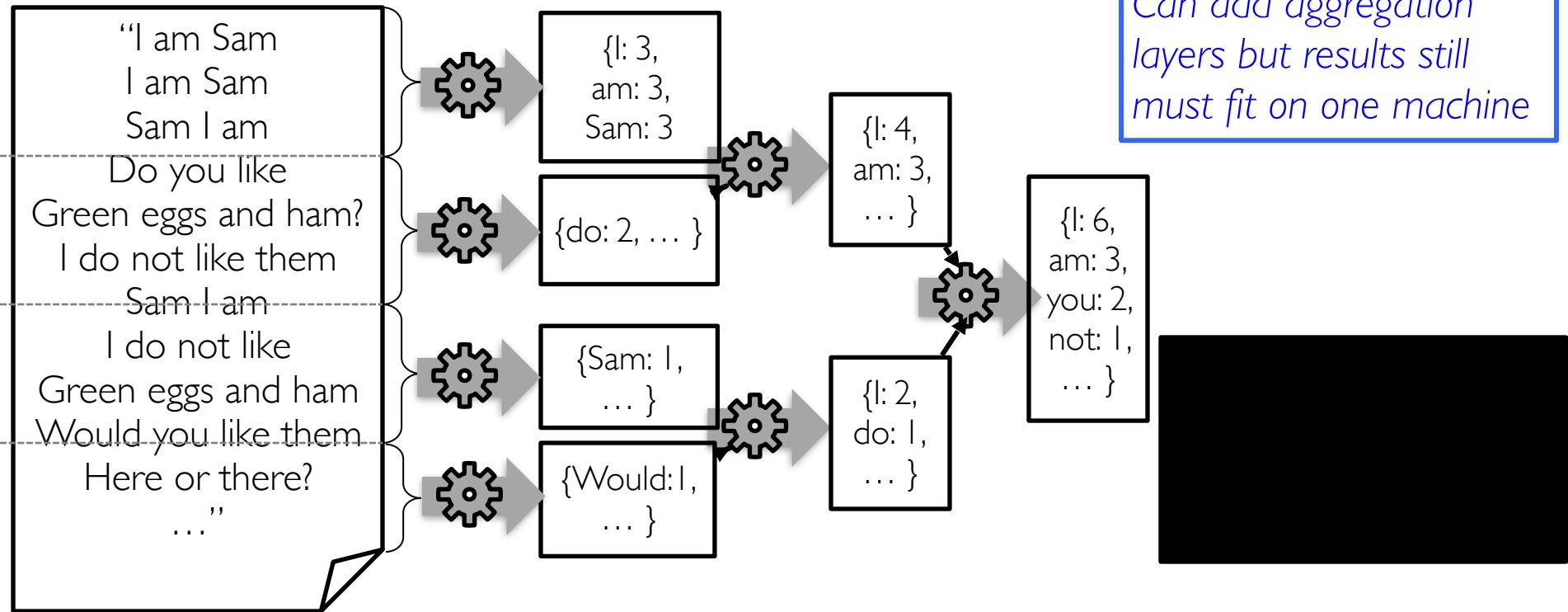


What's the problem with this approach?

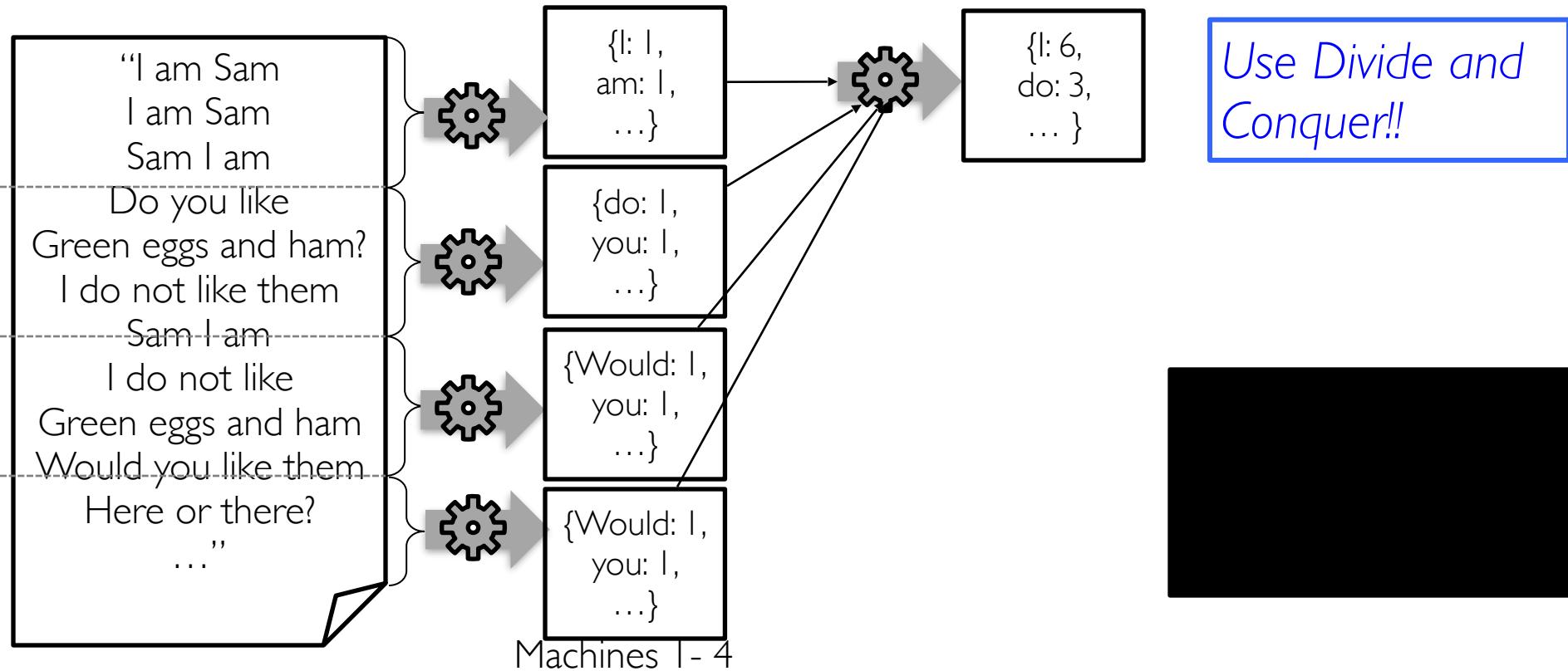
What if the Document is Really Big?



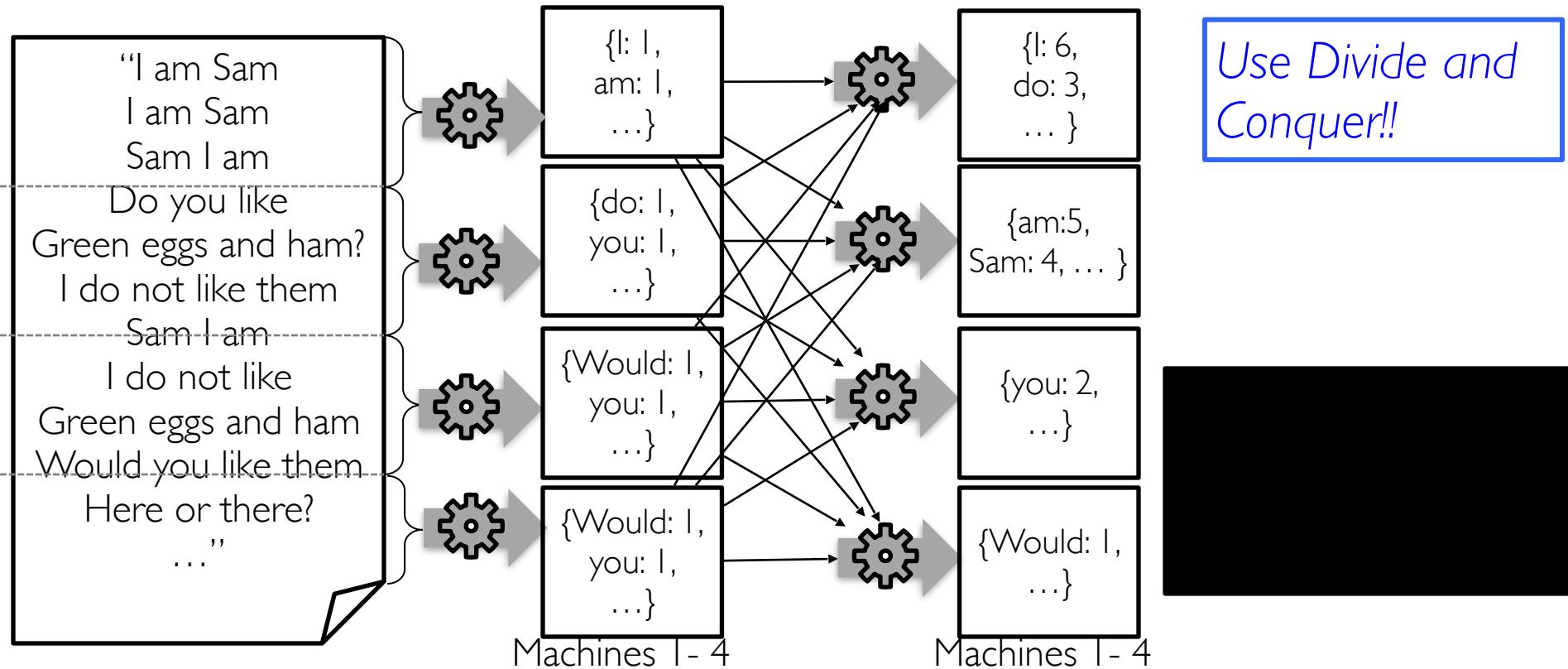
What if the Document is Really Big?



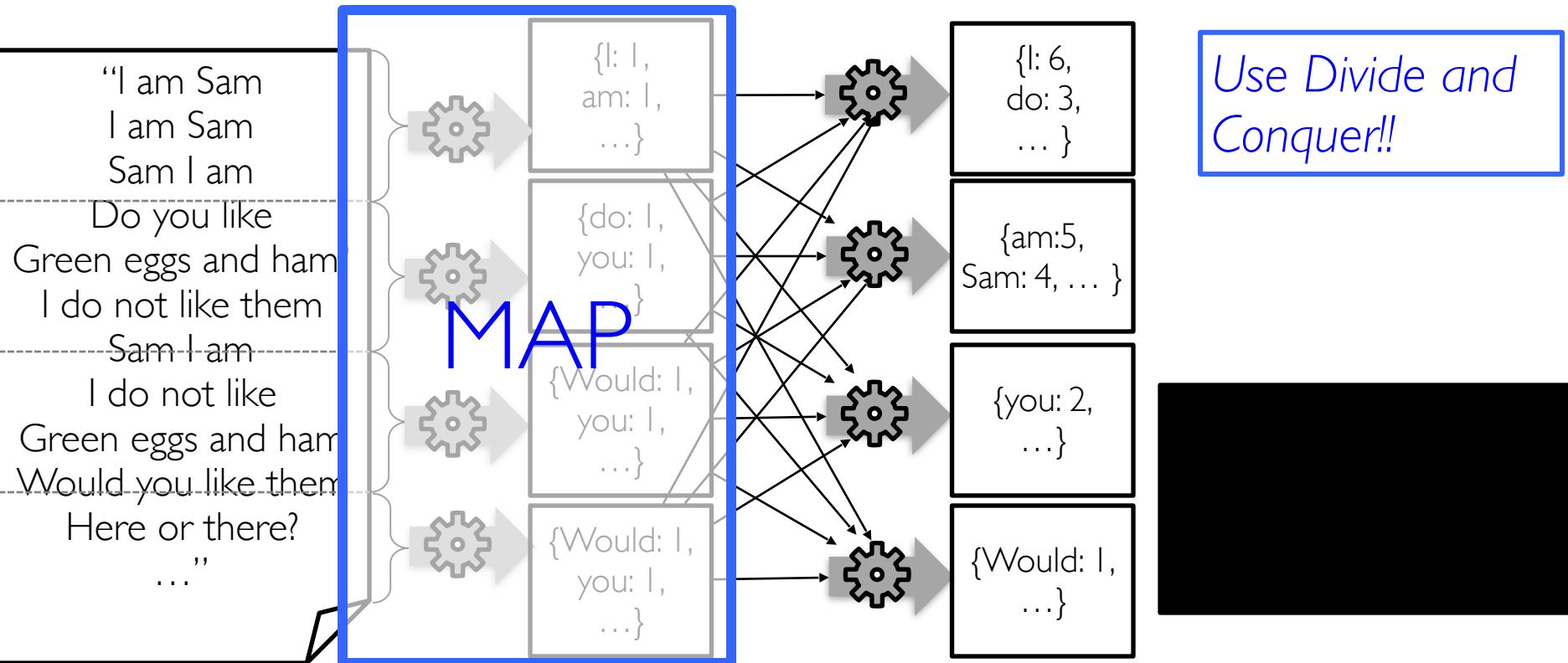
What if the Document is Really Big?



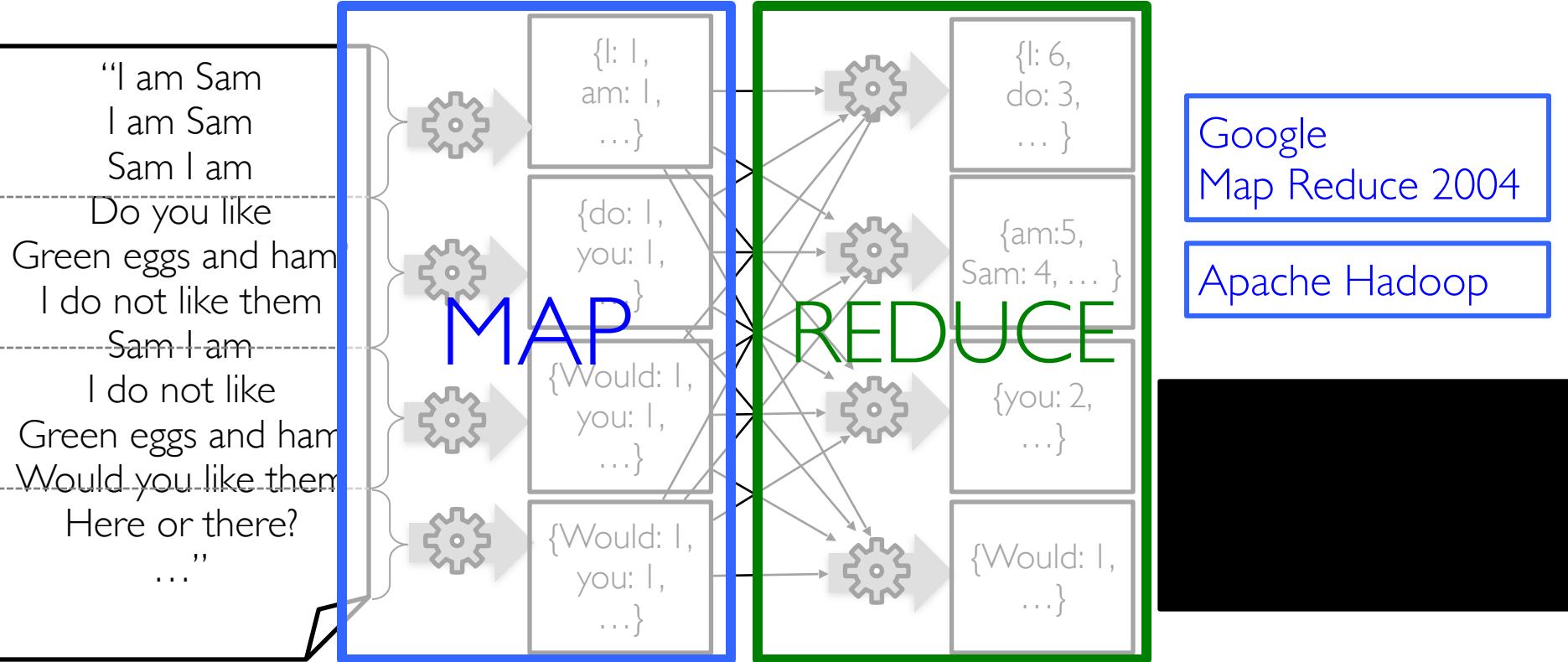
What if the Document is Really Big?



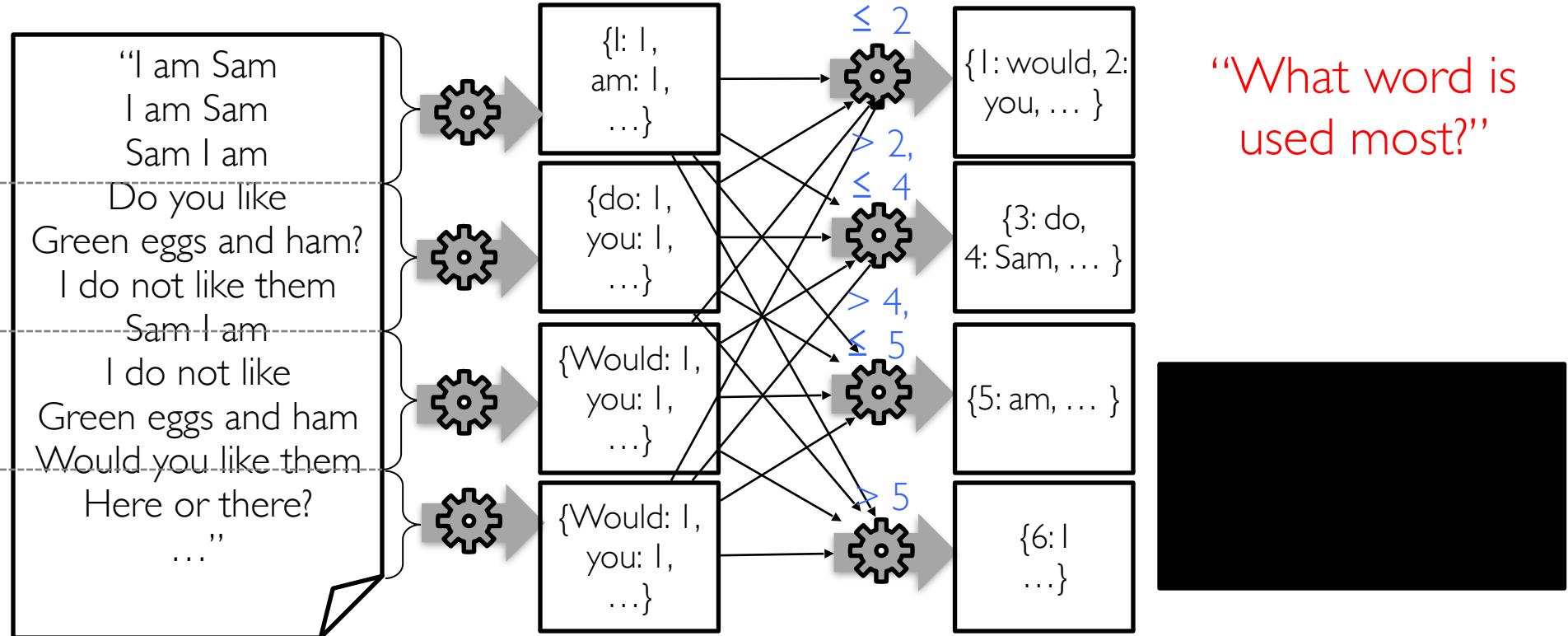
What if the Document is Really Big?



What if the Document is Really Big?



Map Reduce for Sorting



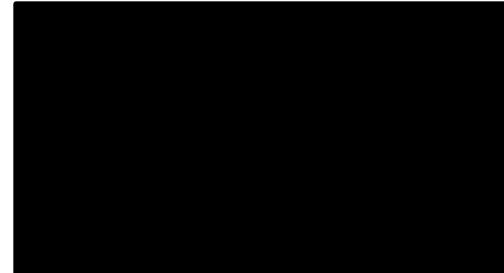
What's Hard About Cluster Computing?

How to divide work across machines?

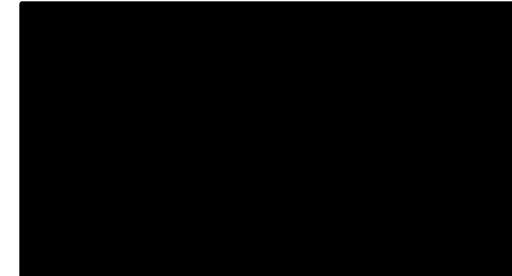
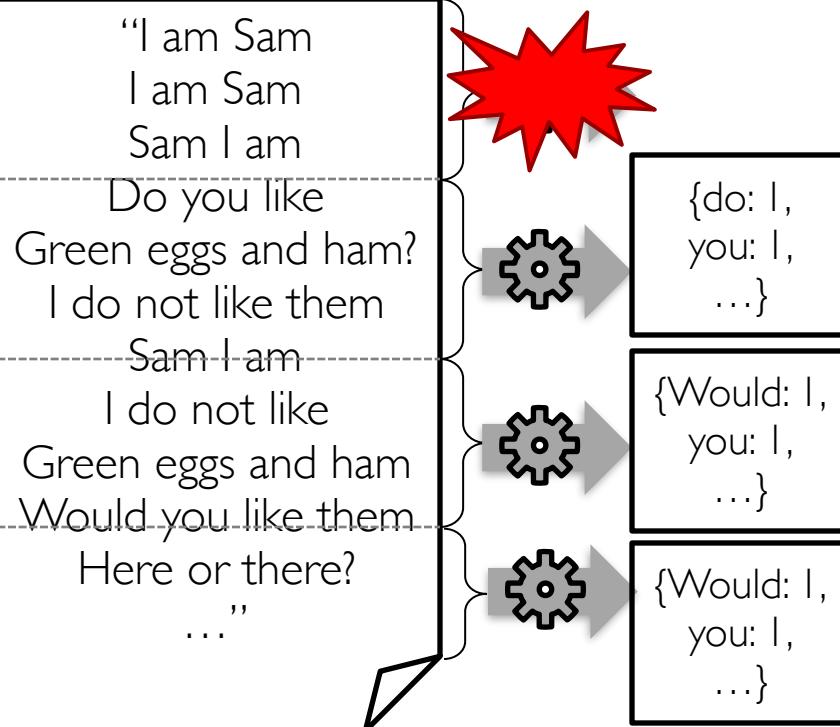
- » Must consider network, data locality
- » Moving data may be very expensive

How to deal with failures?

- » 1 server fails every 3 years \Rightarrow with 10,000 nodes see 10 faults/day
- » Even worse: stragglers (not failed, but slow nodes)



How Do We Deal with Failures?

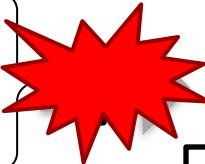


How Do We Deal with Machine Failures?

"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?
..."



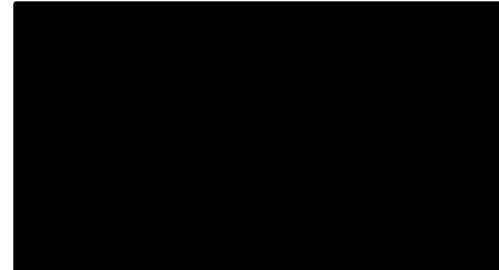
{do: I,
you: I,
...}

{Would: I,
you: I,
...}

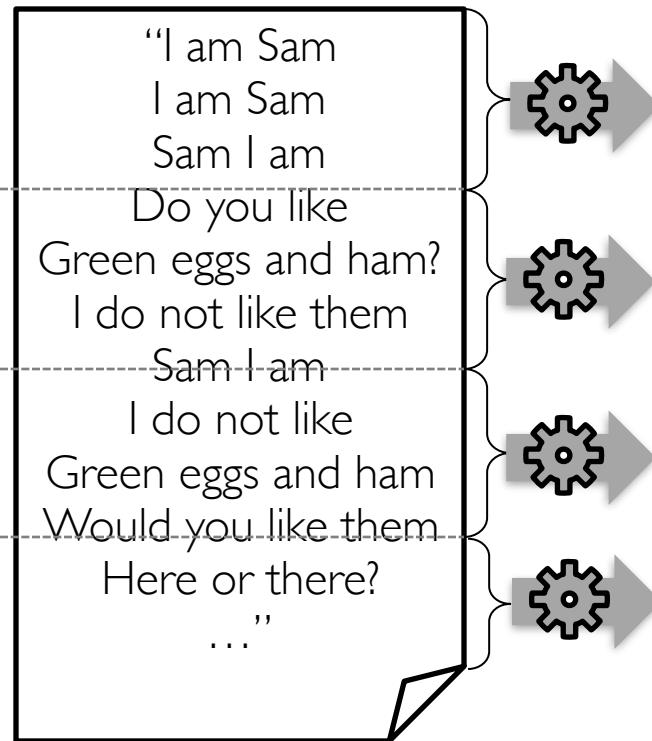
{Would: I,
you: I,
...}

{I: I,
am: I,
...}

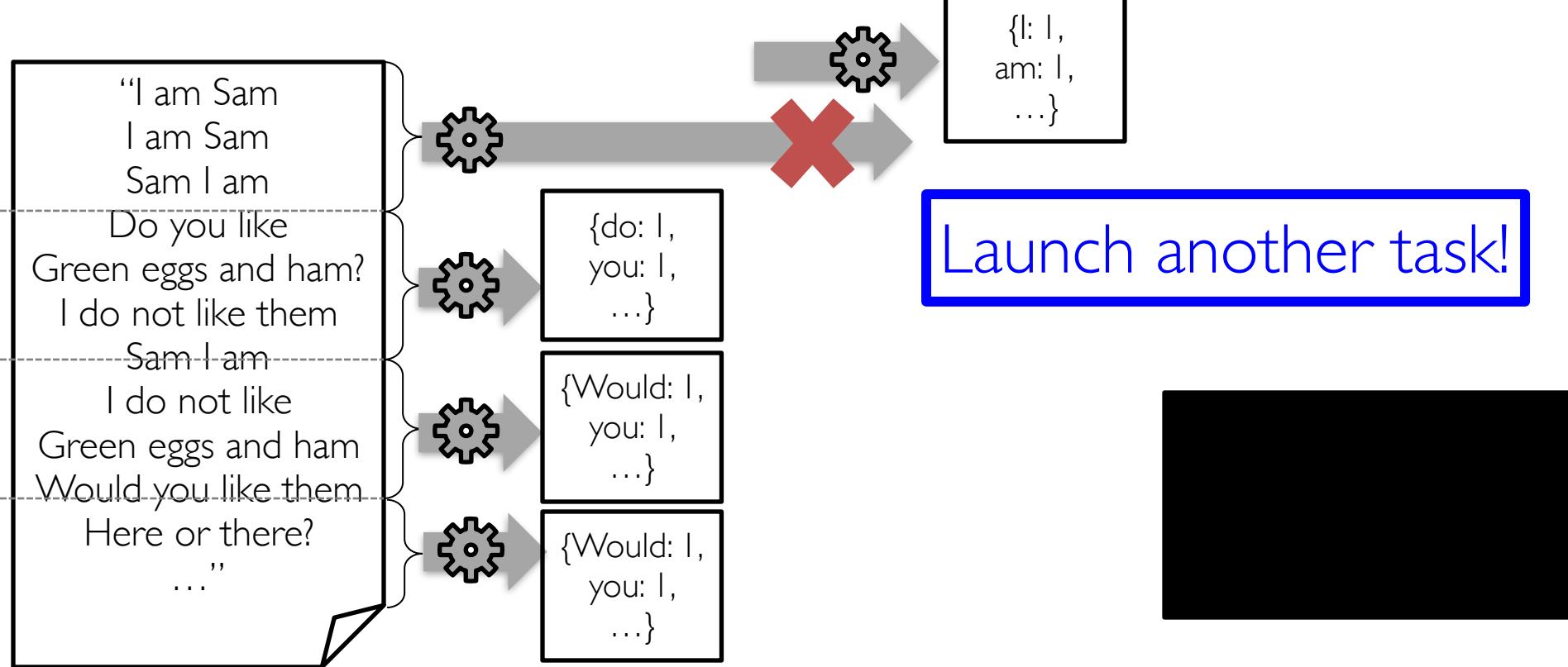
Launch another task!



How Do We Deal with Slow Tasks?

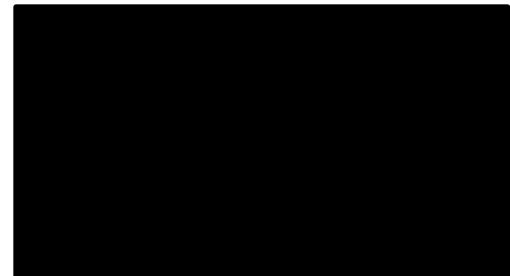


How Do We Deal with Slow Tasks?

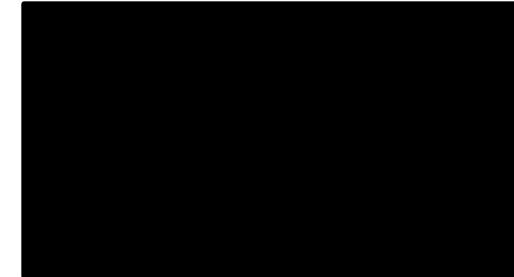
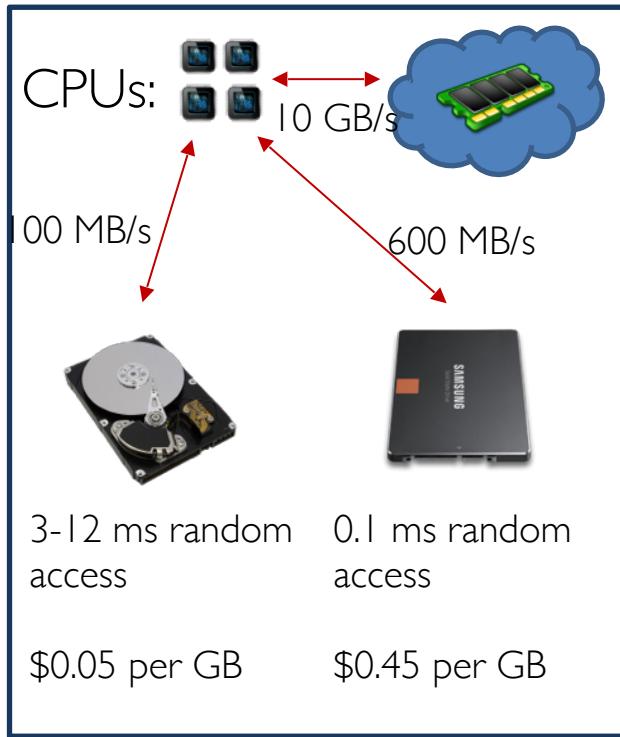


What is Apache Spark?

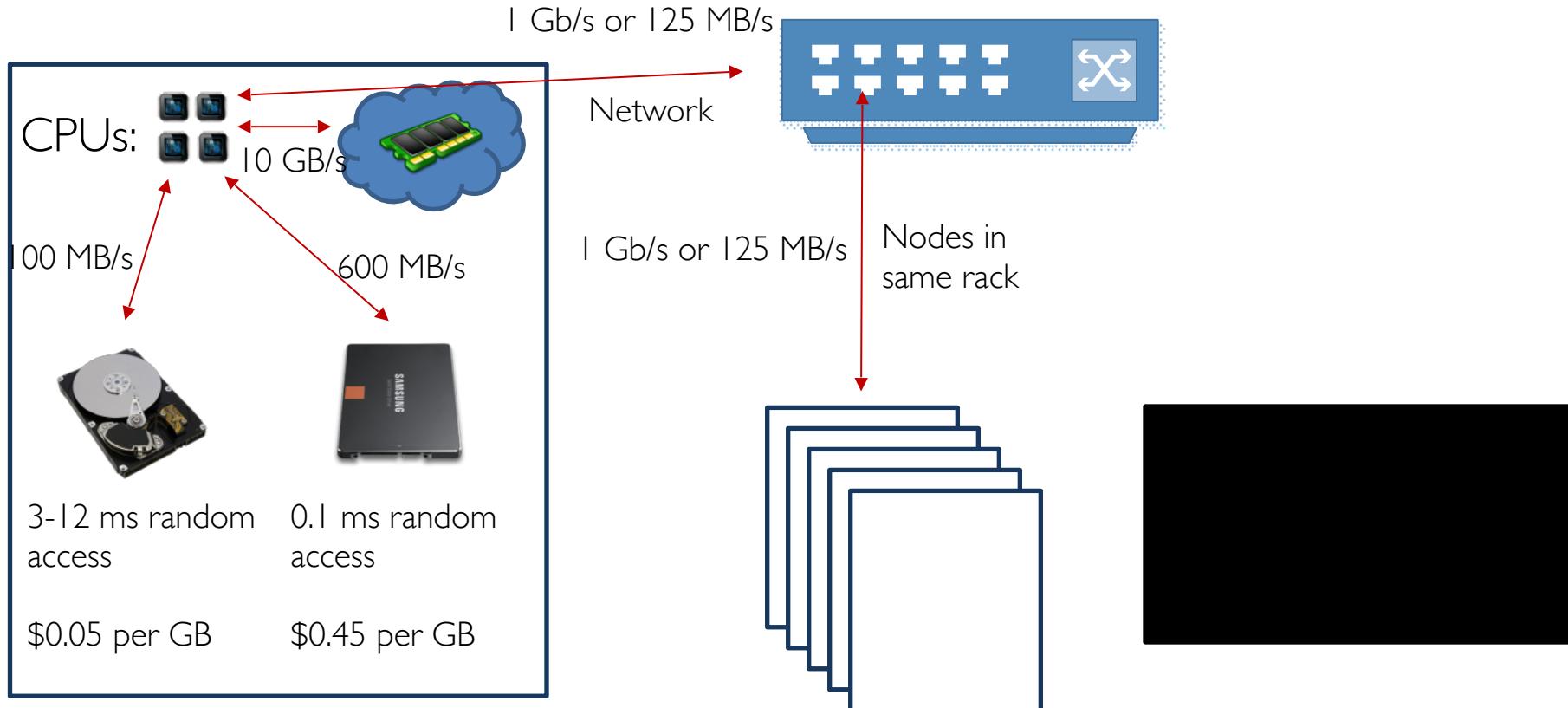
- Scalable, efficient analysis of Big Data



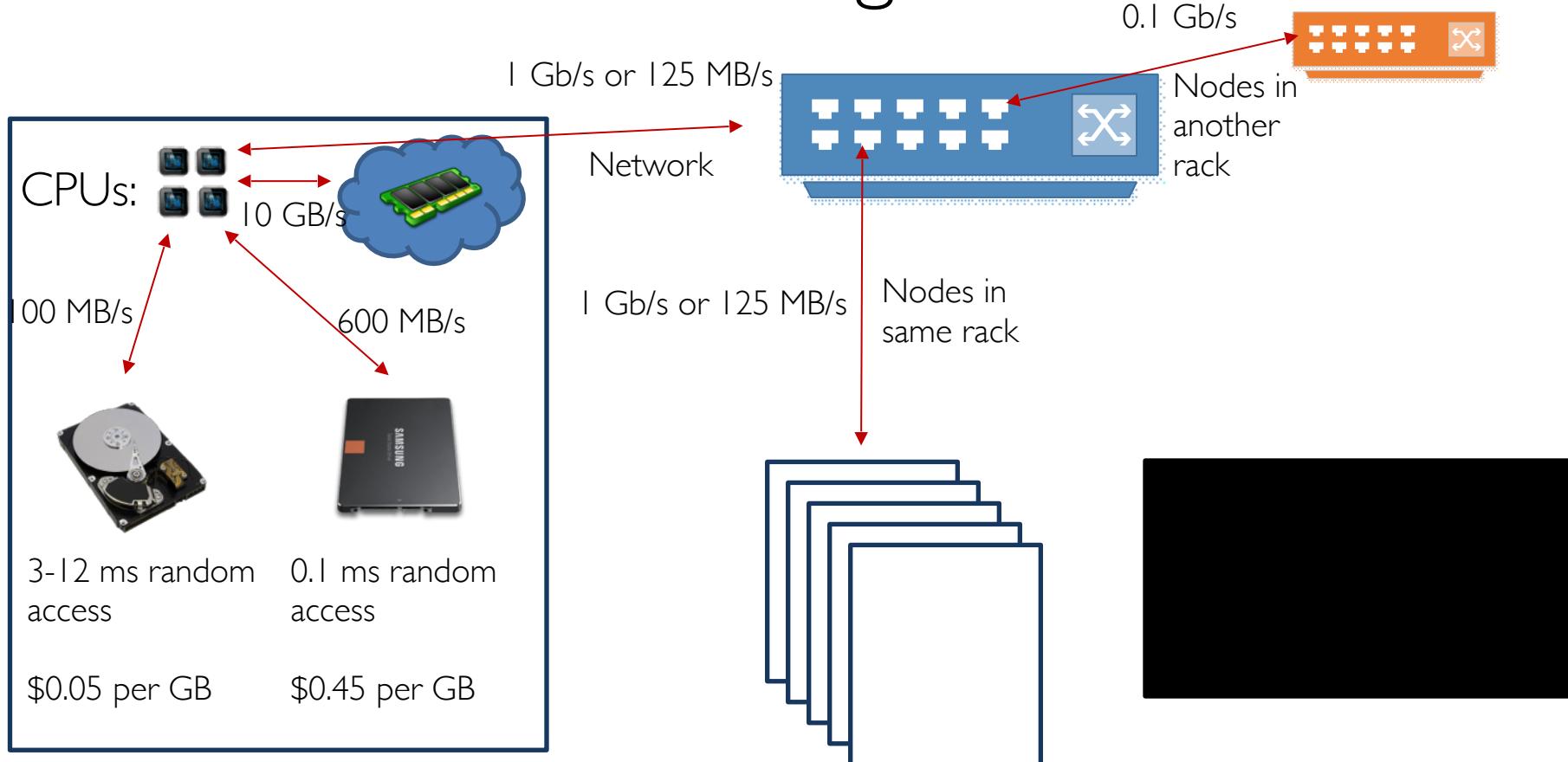
Datacenter Organization



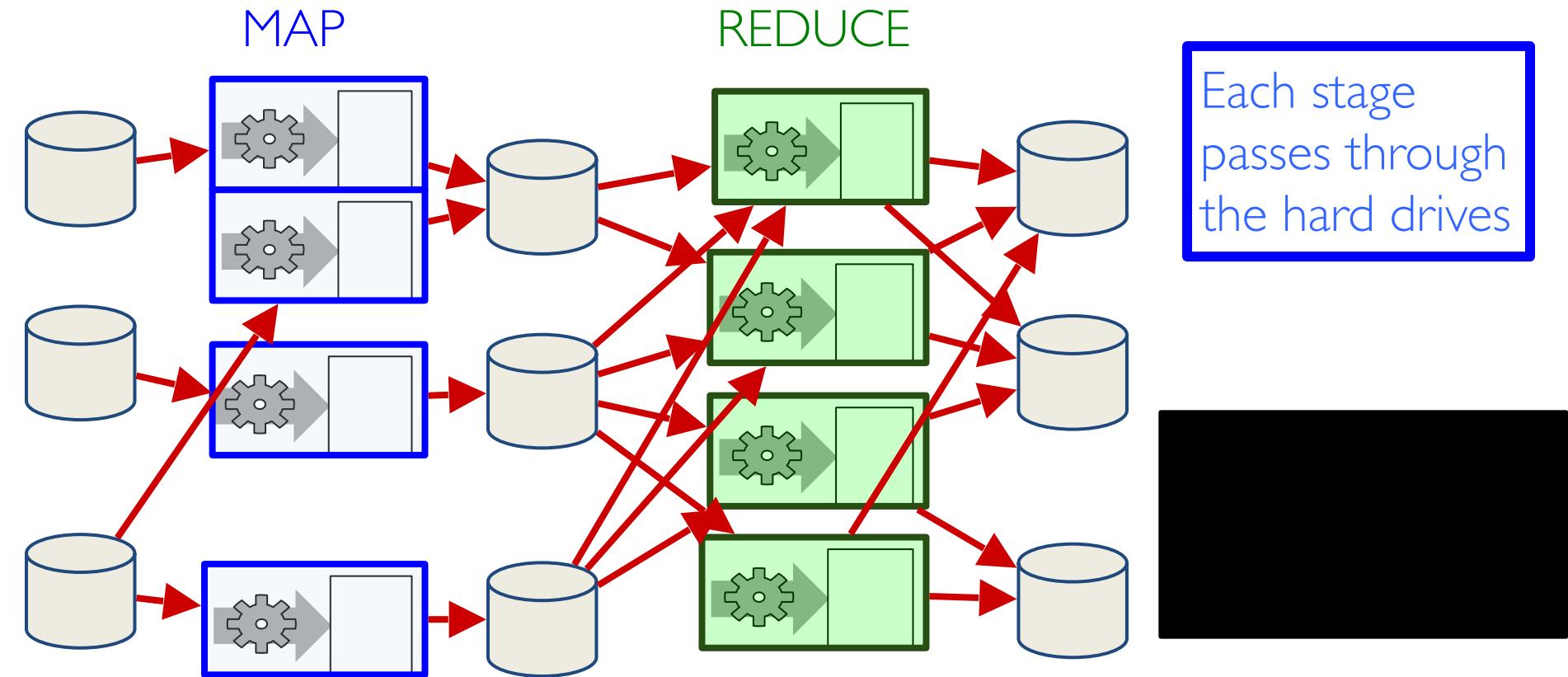
Datacenter Organization



Datacenter Organization

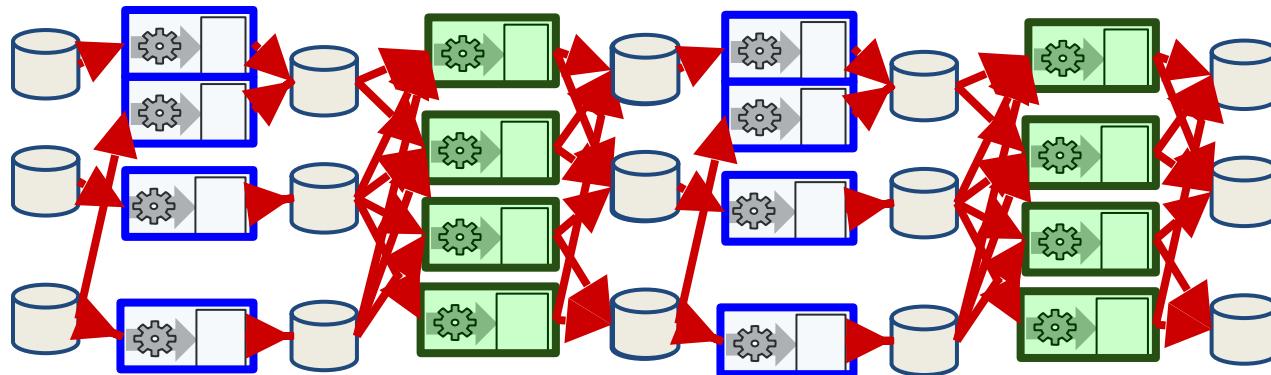


Map Reduce: Distributed Execution

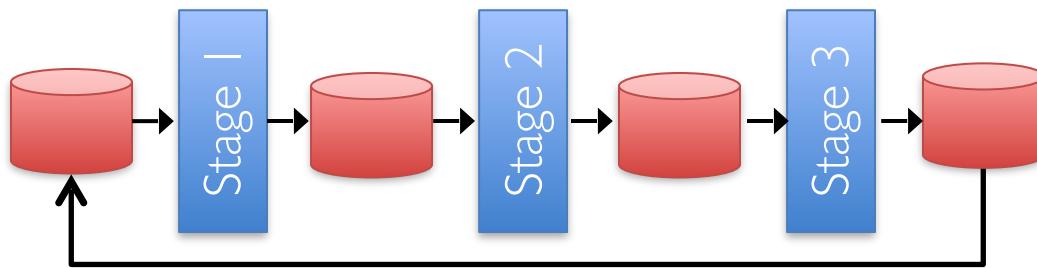


Map Reduce: Iterative Jobs

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

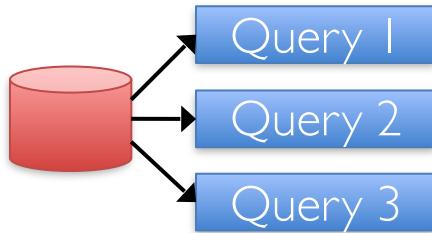


Disk I/O is
very slow!

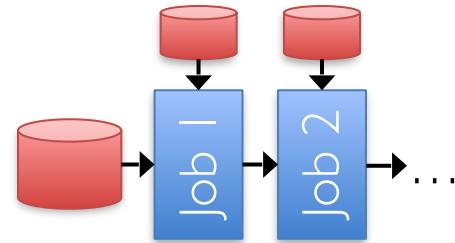


Apache Spark Motivation

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



Interactive mining

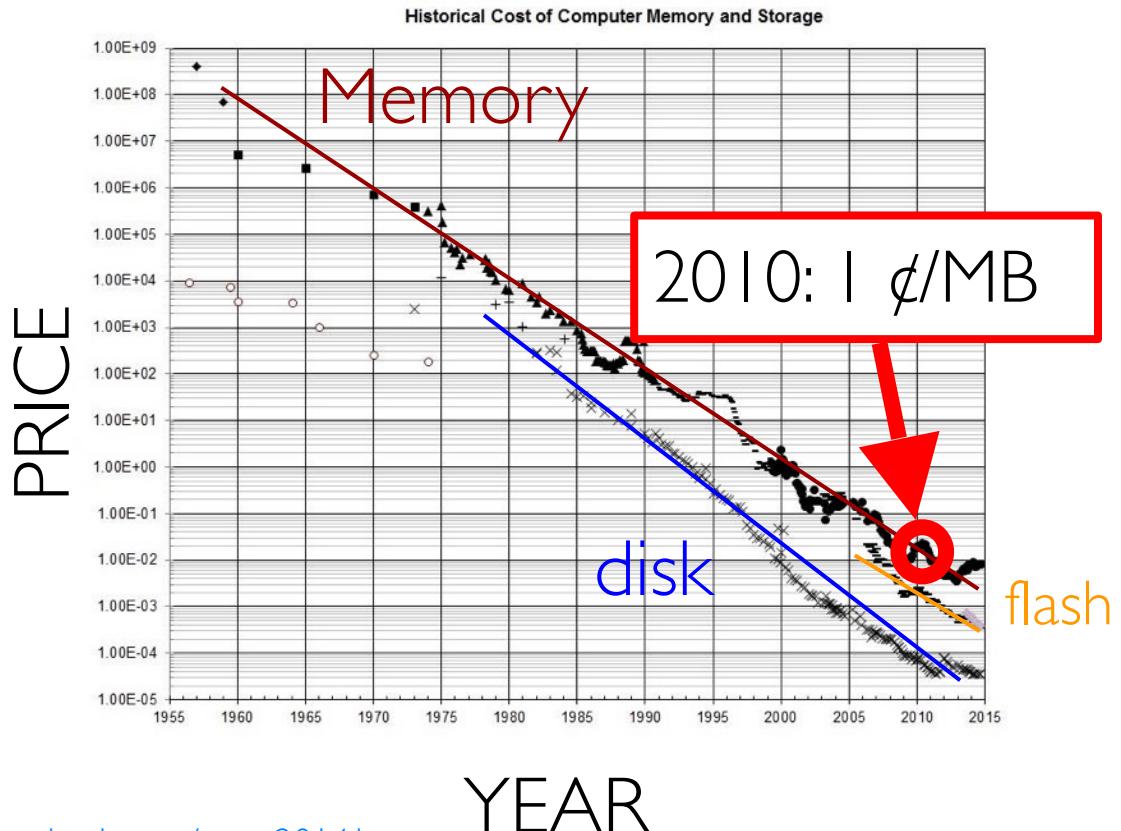


Stream processing

Also, iterative jobs

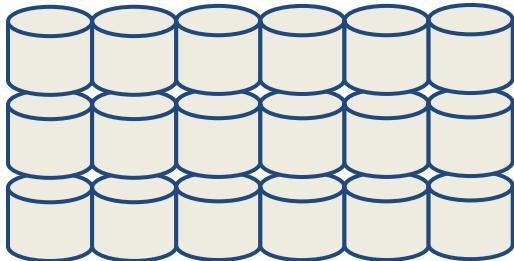
Disk I/O is very slow

Tech Trend: Cost of Memory

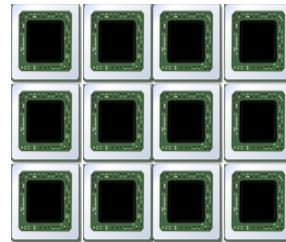


Lower cost means can put more memory in each server

Modern Hardware for Big Data



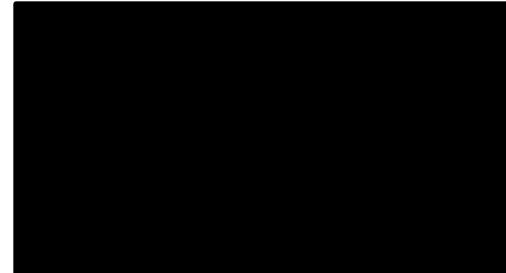
Lots of hard drives



... and CPUs

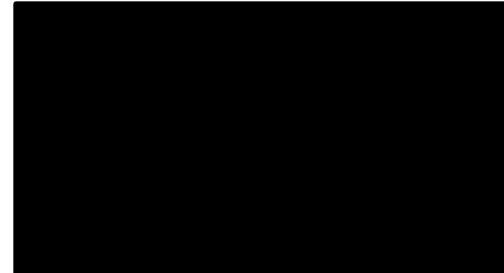


... and memory!

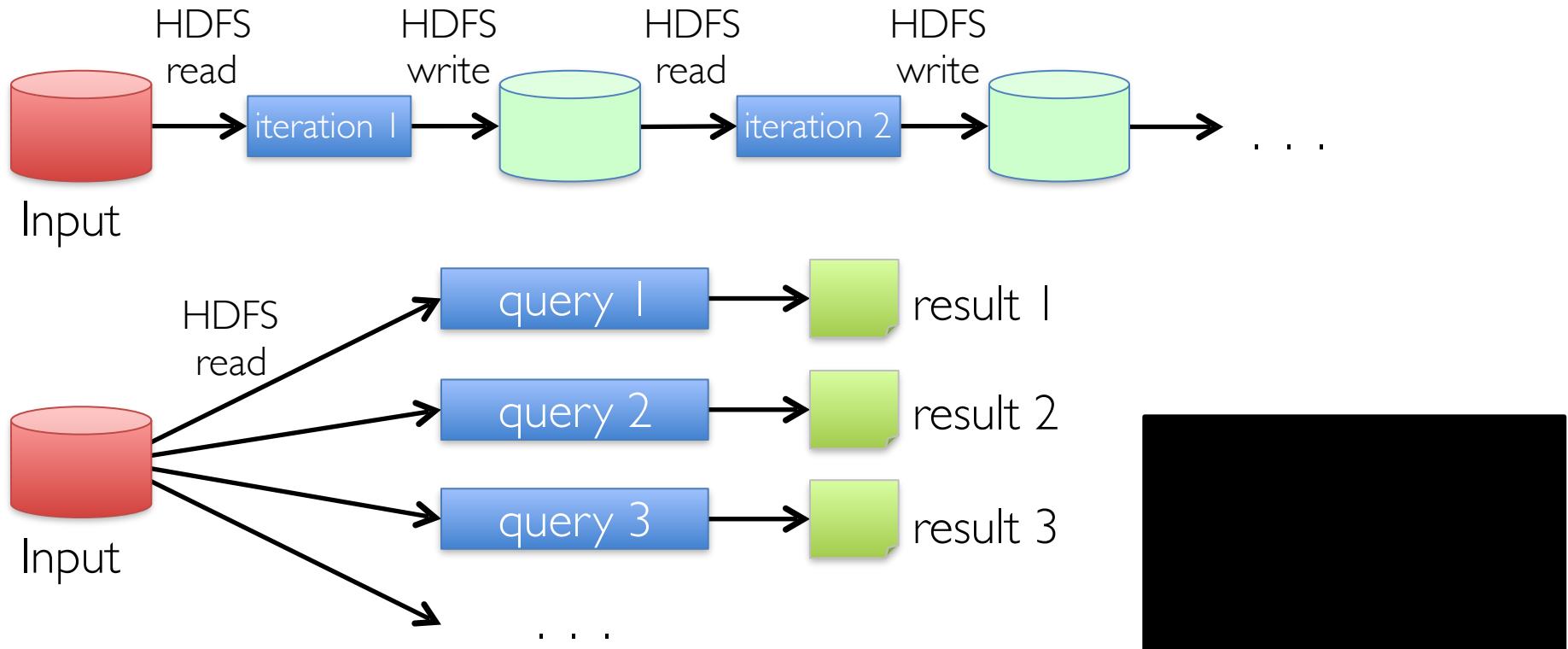


Opportunity

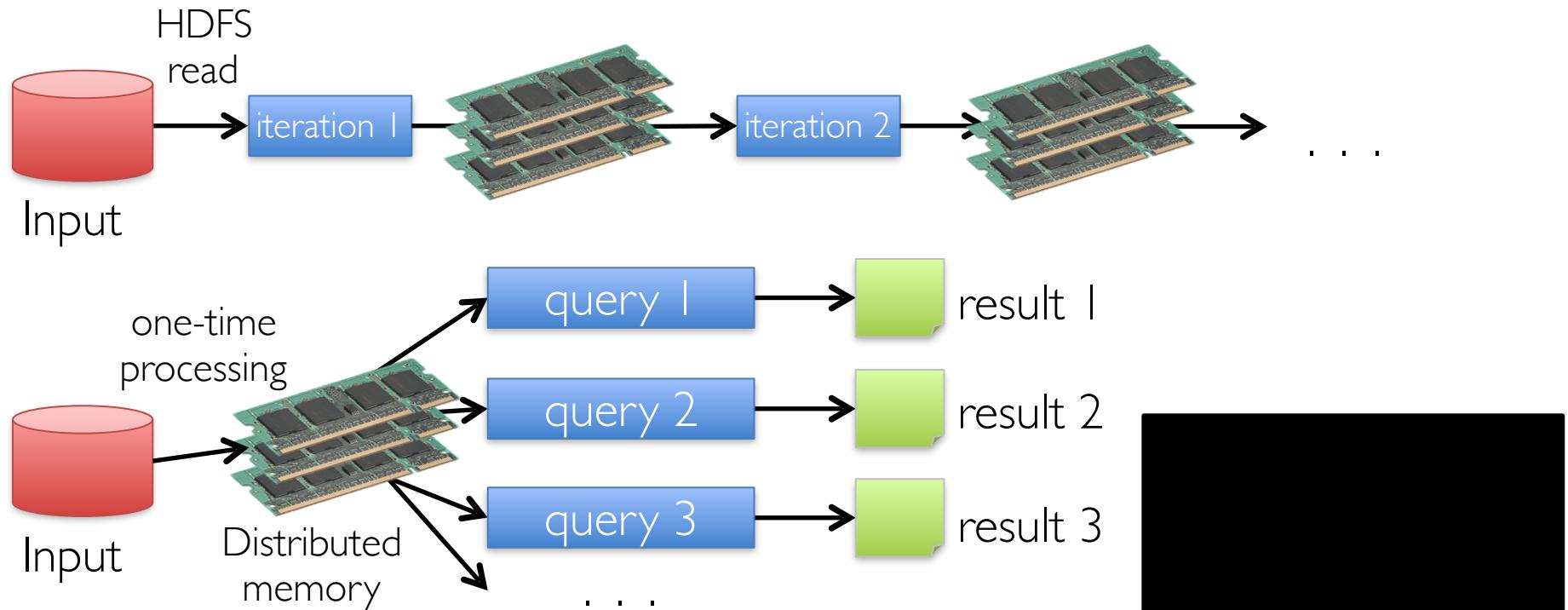
- Keep more data *in-memory*
- Create new distributed execution engine:



Use Memory Instead of Disk



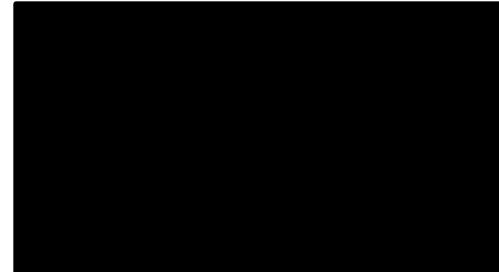
In-Memory Data Sharing



10-100x faster than network and disk

Spark and Map Reduce Differences

	Apache Hadoop Map Reduce	Apache Spark
Storage	Disk only	In-memory or on disk
Operations	Map and Reduce	Many transformation and actions, including Map and Reduce
Execution model	Batch	Batch, interactive, streaming
Languages	Java	Scala, Java, R, and Python



Other Spark and Map Reduce Differences

Generalized patterns for computation

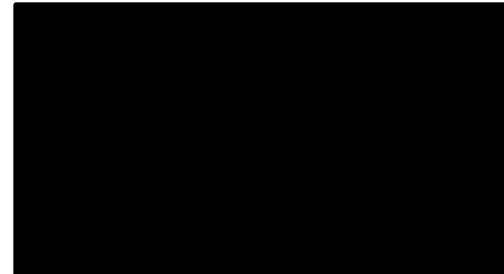
- ⇒ provide unified engine for many use cases
- ⇒ require 2-5x less code

Lazy evaluation of the lineage graph

- ⇒ can optimize, reduce wait states, pipeline better

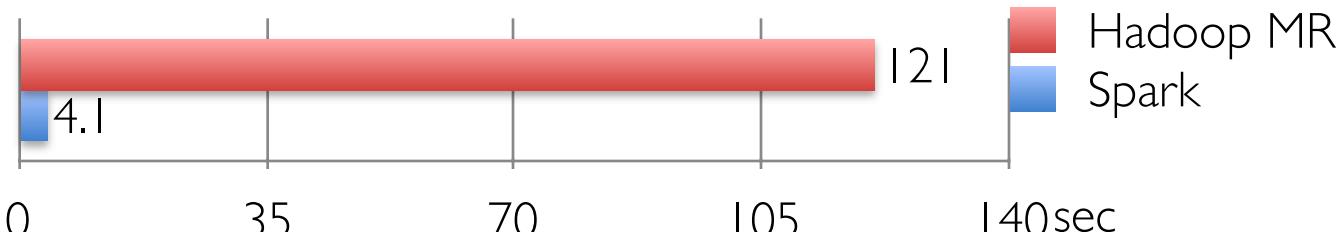
Lower overhead for starting jobs

Less expensive shuffles

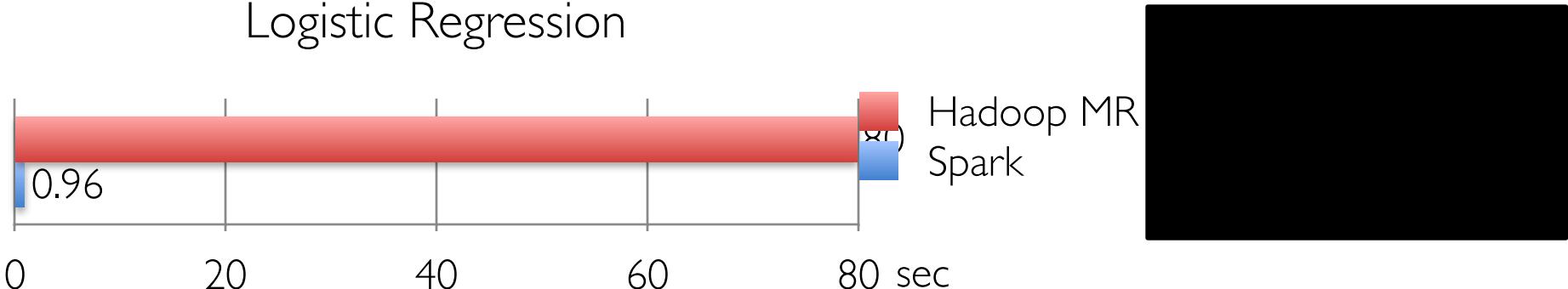


In-Memory Can Make a Big Difference

(2013) Two iterative Machine Learning algorithms:
K-means Clustering



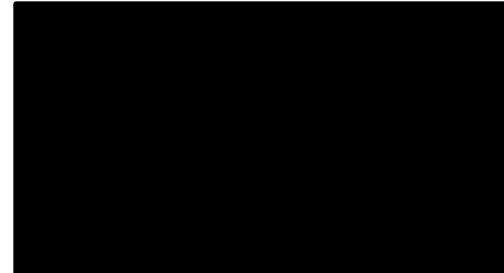
Logistic Regression



First Public Cloud Petabyte Sort (2014)

	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)



Recent Spark Performance Optimizations

Spark has added two key performance optimizations

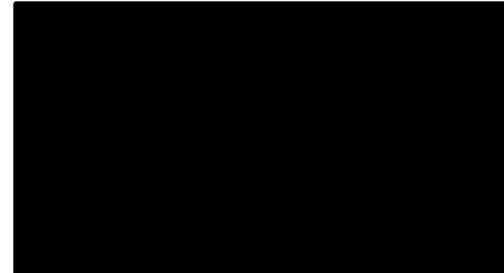
- » In addition to using memory instead of disk

Catalyst Optimization Engine

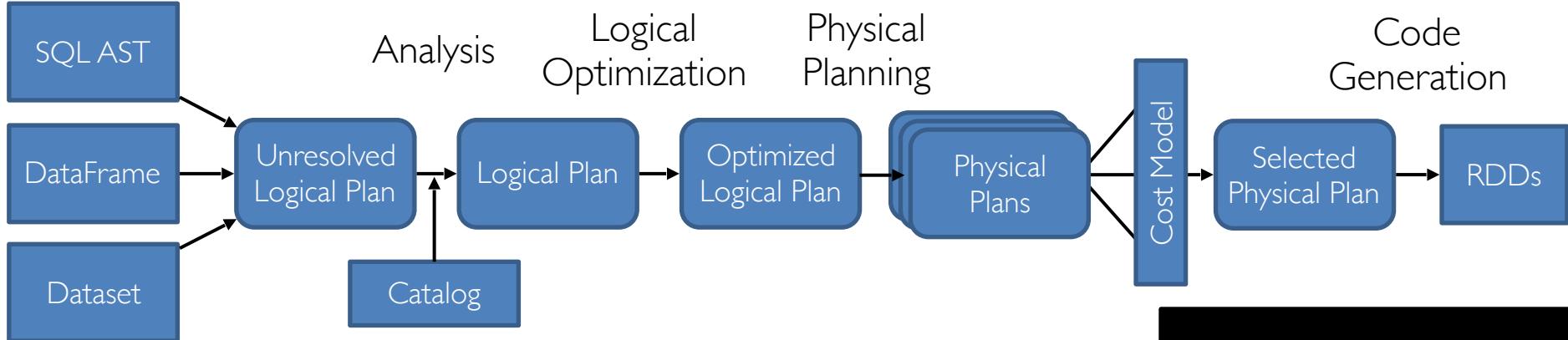
- » 75% reduction in execution time

Project Tungsten off-heap memory management

- » 75+% reduction in memory usage (less GC)



Catalyst: Shared Optimization & Execution



DataFrames, Datasets, and Spark SQL
share the same optimization/execution pipeline

Java Virtual Machine Object Overhead

“abcd” → Native: 4 bytes with UTF-8 encoding
Java: 48 bytes

java.lang.String object internals:

OFFSET	SIZE	TYPE	DESCRIPTION
0	4		(object header)
4	4		(object header)
8	4		(object header)
12	4	char[]	String.value
16	4	int	String.hash
20	4	int	String.hash32

VALUE

...

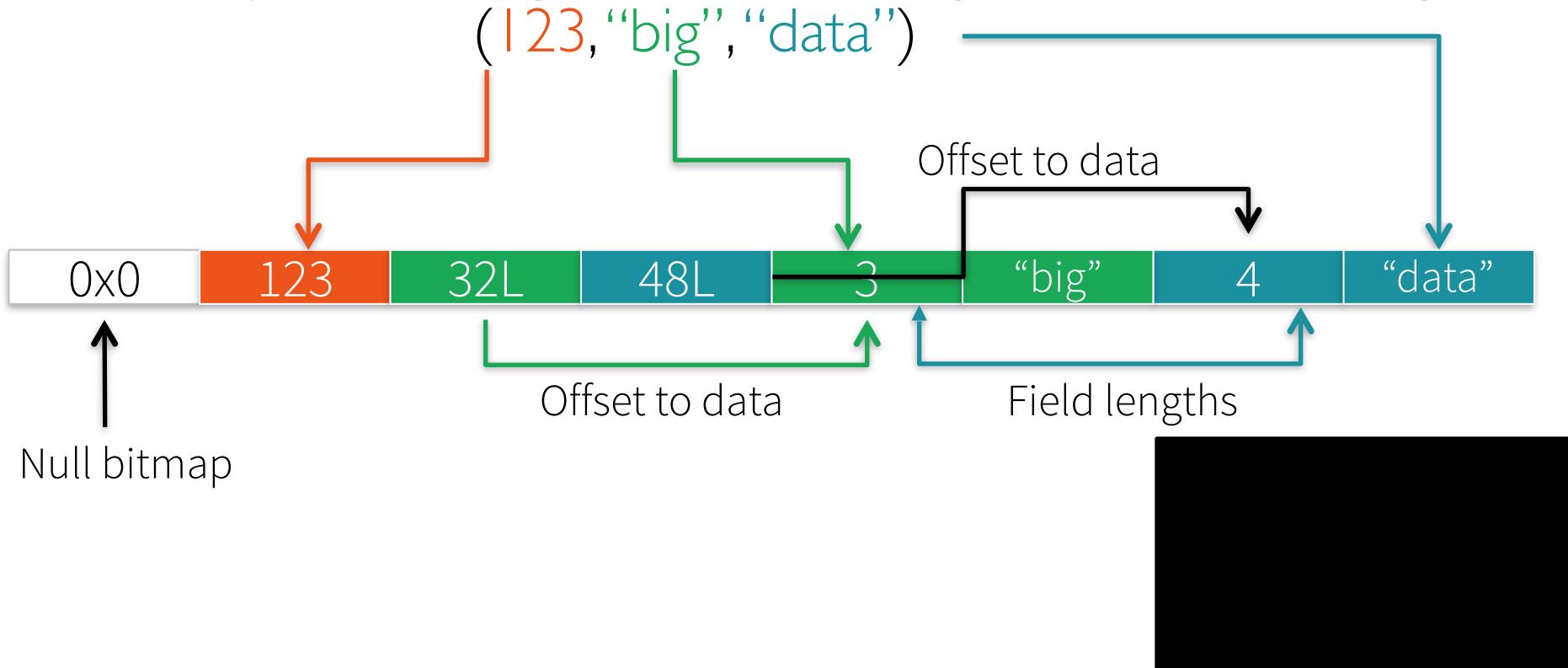
12 byte object header

20 bytes data + overhead

8 byte hashcode

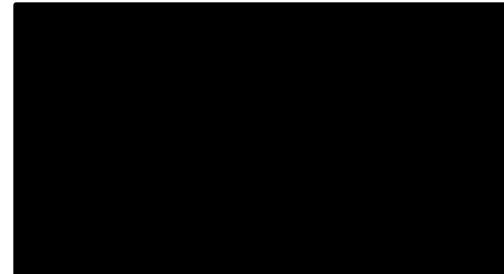
Instance size: 24 bytes (reported by Instrumentation API)

Project Tungsten's Compact Encoding



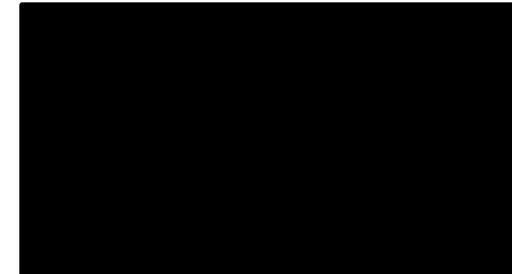
Review: Key Data Management Concepts

- A **data model** is a collection of concepts for describing data
- A **schema** is a description of a particular collection of data, using a given data model
- A **relational data model** is the most used data model
 - » **Relation**, a table with rows and columns
 - » Every relation has a **schema** defining fields in columns

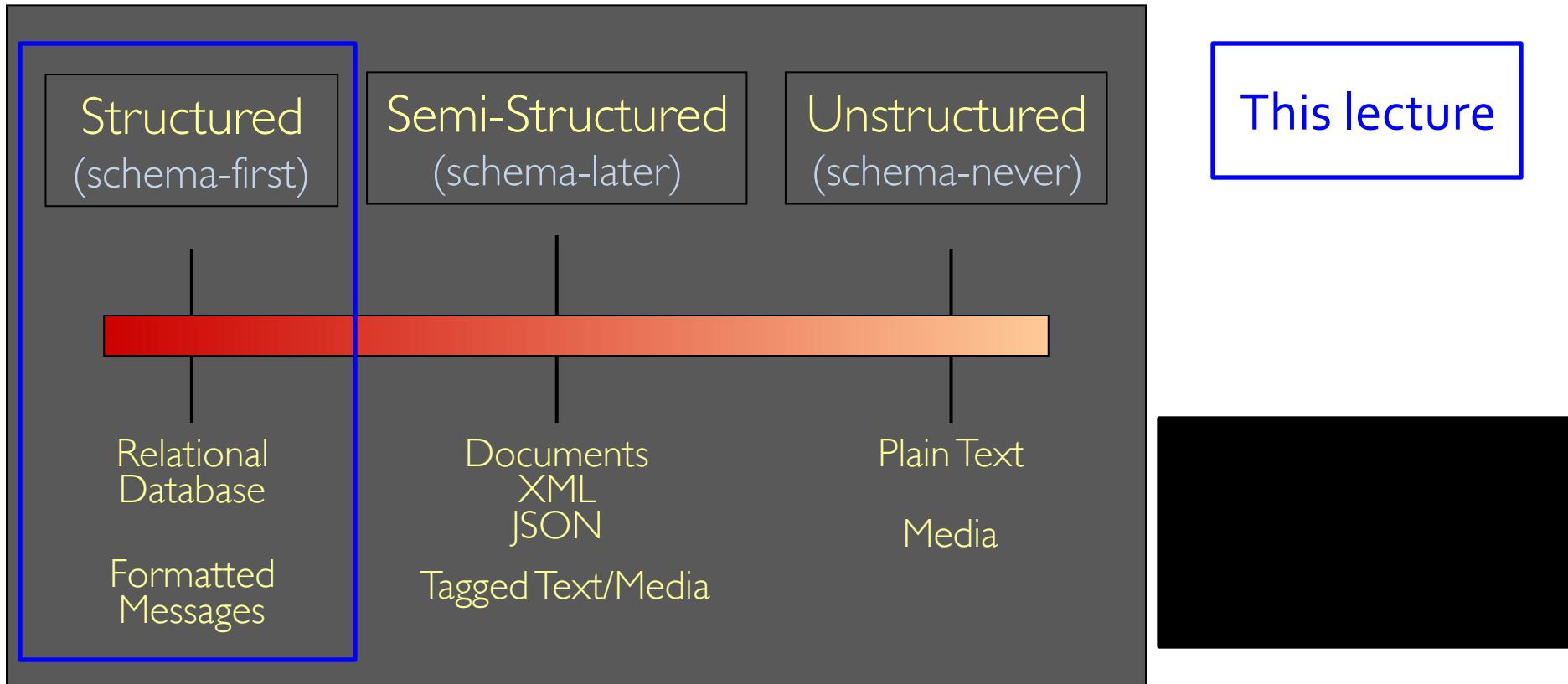


Review: Key Data Management Concepts

- A **data model** is a collection of concepts for describing data
- A **schema** is a description of a particular collection of data, using a given data model
- A **relational data model** is the most used data model
 - » **Relation**, a table with rows and columns
 - » Every relation has a **schema** defining fields in columns



The Structure Spectrum

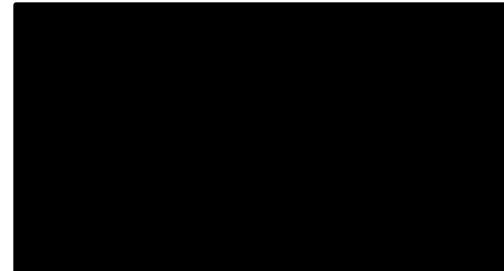


Relational Database: Definitions

- *Relational database*: a set of *relations*
- Two parts to a *Relation*:
Schema: specifies name of relation, plus each column's name and type
**Students(*sid*: string, *name*: string, *email*: string,
 age: integer, *gpa*: real)**

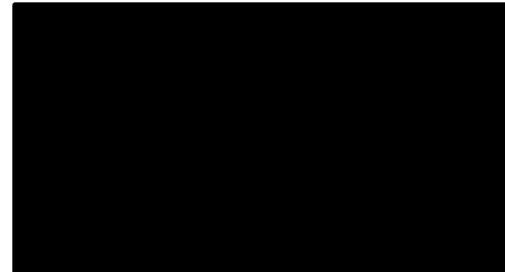
Instance: the actual data at a given time

- #rows = *cardinality*
- #fields = *degree*



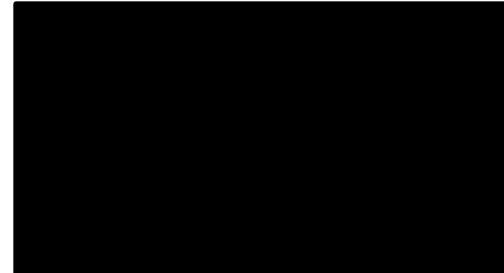
What is a Database?

- A large organized collection of data
 - » Transactions used to modify data
- Models real world, e.g., enterprise
 - » Entities (e.g., teams, games)
 - » Relationships, e.g.,
 - » A plays against B in The World Cup



Large Databases

- US Internal Revenue Service: [150 Terabytes](#)
- Australian Bureau of Stats: [250 Terabytes](#)
- AT&T call records: [312 Terabytes](#)
- eBay database: [1.4 Petabytes](#)
- Yahoo click data: [2 Petabytes](#)
- *What matters for these databases?*



Large Databases

- US Internal Revenue Service: 150 Terabytes ← Accuracy, Consistency, Durability, Rich queries
- Australian Bureau of Stats: 250 Terabytes ← Fast, Rich queries
- AT&T call records: 312 Terabytes ← Accuracy, Consistency, Durability
- eBay database: 1.4 Petabytes
- Yahoo click data: 2 Petabytes ← Availability, Timeliness
- *What matters for these databases?*

Example: Instance of Structured Data

`Students(sid:string, name:string, login:string, age:integer, gpa:real)`

sid	name	login	age	gpa
3366	Jones	jones@eecs	18	3.4
53688	Smith	smith@statistics	18	3.2
53650	Smith	smith@math	19	3.8

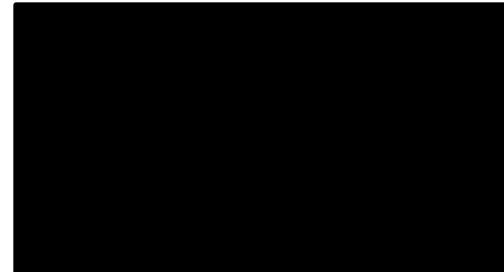
- Cardinality = 3 (rows)

- Tuples or rows = 5 (columns)

- All rows (tuples) are distinct

SQL - A language for Relational DBs

- SQL = Structured Query Language
- Supported by Spark DataFrames (SparkSQL)
- Some of the functionality SQL provides:
 - » Create, modify, delete relations
 - » Add, modify, remove tuples
 - » *Specify queries to find tuples matching criteria*



Queries in SQL

- Single-table queries are straightforward
- To find all 18 year old students, we can write:

```
SELECT *
  FROM Students S
 WHERE S.age=18
```

- To find just names and logins:

```
SELECT S.name, S.login
  FROM Students S
 WHERE S.age=18
```

Querying Multiple Relations

- Can specify a **join** over two tables as follows:

```
SELECT S.name, E.cid  
FROM Students S, Enrolled E  
WHERE S.sid=E.sid
```

Enrolled

E	E.sid	E.cid	E.grade
	53831	Physics203	A
	53650	Topology112	A
	53341	History105	B

S

S	S.sid	S.name	S.login	S.age	S.gpa
	53341	Jones	jones@cs	18	3.4
	53831	Smith	smith@ee	18	3.2

s, S and E

Cross Join

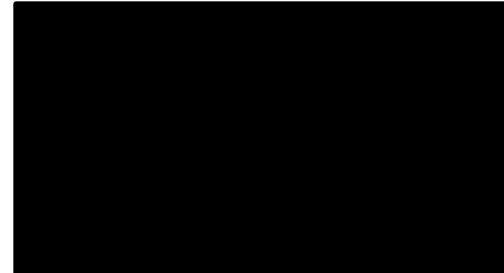
- Cartesian product of two tables ($E \times S$):

Enrolled

E	E.sid	E.cid	E.grade
	53831	Physics203	A
	53650	Topology112	A
	53341	History105	B

S

Students	S.sid	S.name	S.login	S.age	S.gpa
	53341	Jones	jones@cs	18	3.4
	53831	Smith	smith@ee	18	3.2



Cross Join

- Cartesian product of two tables ($E \times S$):

Enrolled

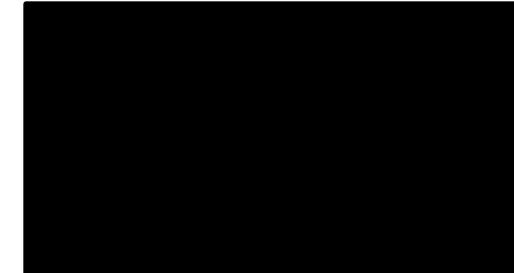
E	E.sid	E.cid	E.grade
	53831	Physics203	A
	53650	Topology112	A
	53341	History105	B

S

S	S.sid	S.name	S.login	S.age	S.gpa
	53341	Jones	jones@cs	18	3.4
	53831	Smith	smith@ee	18	3.2

Students

E.sid	E.cid	E.grade	S.sid	S.name	S.login	S.age	S.gpa
53831	Physics203	A	53341	Jones	jones@cs	18	3.4
53650	Topology112	A	53341	Jones	jones@cs	18	3.4
53341	History105	B	53341	Jones	jones@cs	18	3.4
53831	Physics203	A	53831	Smith	smith@ee	18	3.2
53650	Topology112	A	53831	Smith	smith@ee	18	3.2
53341	History105	B	53831	Smith	smith@ee	18	3.2



Where Clause

- Choose matching rows using Where clause:

```
SELECT S.name, E.cid  
FROM Students S, Enrolled E  
WHERE S.sid=E.sid
```

E.sid	E.cid	E.grade	S.sid	S.name	S.login	S.age	S.gpa
53831	Physics203	A	53341	Jones	jones@cs	18	3.4
53650	Topology112	A	53341	Jones	jones@cs	18	3.4
53341	History105	B	53341	Jones	jones@cs	18	3.4
53831	Physics203	A	53831	Smith	smith@ee	18	3.2
53650	Topology112	A	53831	Smith	smith@ee	18	3.2
53341	History105	B	53831	Smith	smith@ee	18	3.2

Select Clause

- Filter columns using Select clause:

```
SELECT S.name, E.cid  
FROM Students S, Enrolled E  
WHERE S.sid=E.sid
```

E.sid	E.cid	E.grade	S.sid	S.name	S.login	S.age	S.gpa
53831	Physics203	A	53341	Jones	jones@cs	18	3.4
53650	Topology112	A	53341	Jones	jones@cs	18	3.4
53341	History105	B	53341	Jones	jones@cs	18	3.4
53831	Physics203	A	53831	Smith	smith@ee	18	3.2
53650	Topology112	A	53831	Smith	smith@ee	18	3.2
53341	History105	B	53831	Smith	smith@ee	18	3.2

Result

- Can specify a *join* over two tables as follows:

```
SELECT S.name, E.cid  
FROM Students S, Enrolled E  
WHERE S.sid=E.sid
```

Enrolled

E	E.sid	E.cid	E.grade
	53831	Physics203	A
	53650	Topology112	A
	53341	History105	B

S

S.sid	S.name	S.login	S.age	S.gpa
53341	Jones	jones@cs	18	3.4
53831	Smith	smith@ee	18	3.2

Students

Result =

S.name	E.cid
Jones	History105
Smith	Physics203

Explicit SQL Joins

```
SELECT S.name, E.classid  
FROM Students S INNER JOIN Enrolled E ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

Result

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150

Equivalent SQL Join Notations

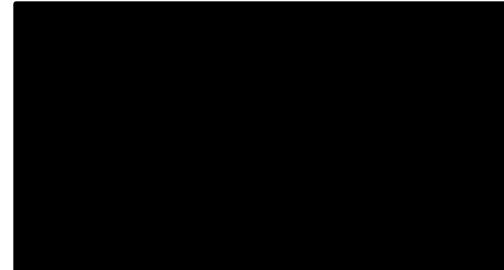
- Explicit Join notation (preferred):

```
SELECT S.name, E.classid  
FROM Students S INNER JOIN Enrolled E ON S.sid=E.sid
```

```
SELECT S.name, E.classid  
FROM Students S JOIN Enrolled E ON S.sid=E.sid
```

- Implicit join notation (deprecated):

```
SELECT S.name, E.cid  
FROM Students S, Enrolled E  
WHERE S.sid=E.sid
```



SQL Types of Joins

```
SELECT S.name, E.classid  
FROM Students S INNER JOIN Enrolled E ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

Result

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150

Unmatched keys

The type of join controls how unmatched keys are handled

SQL Joins: Left Outer Join

```
SELECT S.name, E.classid  
FROM Students S LEFT OUTER JOIN Enrolled E ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

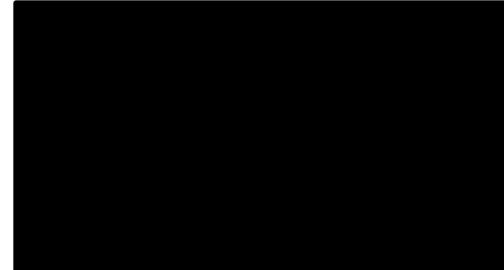
E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

Result

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
Brown	<NULL>

Unmatched keys



SQL Joins: Right Outer Join

```
SELECT S.name, E.classid  
FROM Students S RIGHT OUTER JOIN Enrolled E ON  
S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

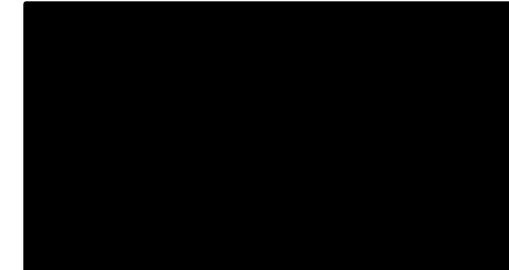
E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

Result

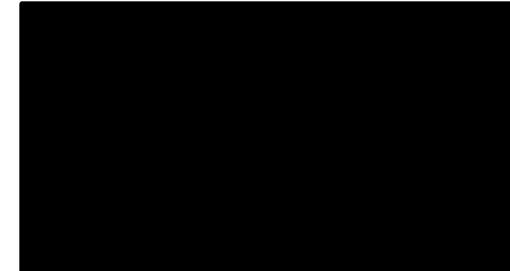
S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
<NULL>	English10

Unmatched keys



Spark Joins

- SparkSQL and Spark DataFrames support joins
- `join(other, on, how)`:
 - » other – right side of the join
 - » on – join column name, list of column (names), or join expression
 - » how – inner, outer, left_outer, right_outer, left_semi



Spark Join Examples(I)

```
>>> df = sqlContext.createDataFrame(data, ['name', 'age'])  
[Row(name=u'Alice', age=1), Row(name=u'Bob', age=2)]  
>>> df2 = sqlContext.createDataFrame(data2,[ 'name', 'height'])  
[Row(name=u'Chris', height=80), Row(name=u'Bob', height=85)]  
  
>>> df.join(df2, 'name')  
[Row(name=u'Bob', age=2, height=85)]
```

Inner Join – X.**join**(Y, cols)

» Return DF of rows with matching **cols** in both X and Y

Spark Join Examples(II)

```
>>> df = sqlContext.createDataFrame(data, ['name', 'age'])  
[Row(name=u'Alice', age=1), Row(name=u'Bob', age=2)]  
>>> df2 = sqlContext.createDataFrame(data2,[ 'name', 'height'])  
[Row(name=u'Chris', height=80), Row(name=u'Bob', height=85)]  
  
>>> df.join(df2, 'name').select(df.name, df2.height)  
[Row(name=u'Bob', height=85)]
```

Inner Join – X.**join**(Y, cols)

» Return DF of rows with matching **cols** in both X and Y

Spark Join Examples(III)

```
>>> df = sqlContext.createDataFrame(data, ['name', 'age'])  
[Row(name=u'Alice', age=1), Row(name=u'Bob', age=2)]  
>>> df2 = sqlContext.createDataFrame(data2,[ 'name', 'height'])  
[Row(name=u'Chris', height=80), Row(name=u'Bob', height=85)]
```

```
>>> df.join(df2, 'name', 'outer')  
[Row(name=u'Chris', age=None, height=80),  
 Row(name=u'Alice', age=1, height=None),  
 Row(name=u'Bob', age=2, height=85)]
```

Outer Join – X.**join**(Y, cols, 'outer')

» Return DF of rows with matching **cols** in either X and Y

Spark Join Examples(IV)

```
>>> df = sqlContext.createDataFrame(data, ['name', 'age'])  
[Row(name=u'Alice', age=1), Row(name=u'Bob', age=2)]  
>>> df2 = sqlContext.createDataFrame(data2,[ 'name', 'height'])  
[Row(name=u'Chris', height=80), Row(name=u'Bob', height=85)]  
  
>>> df.join(df2, 'name', 'outer').select('name', 'height')  
[Row(name=u'Chris', height=80),  
 Row(name=u'Alice', height=None),  
 Row(name=u'Bob', height=85)]
```

Outer Join – X.**join**(Y, cols, 'outer')

» Return DF of rows with matching **cols** in either X and Y

Spark Join Examples(▽)

```
>>> df = sqlContext.createDataFrame(data, ['name', 'age'])  
[Row(name=u'Alice', age=1), Row(name=u'Bob', age=2)]  
>>> df2 = sqlContext.createDataFrame(data2,[ 'name', 'height'])  
[Row(name=u'Chris', height=80), Row(name=u'Bob', height=85)]
```

```
>>> df.join(df2, 'name', 'left_outer')  
[Row(name=u'Alice', age=1, height=None),  
 Row(name=u'Bob', age=2, height=85)]
```

Left Outer Join – X.**join**(Y, cols,
 'left_outer')

» Return DF of rows with matching cols in X

Spark Join Examples(VI)

```
>>> df = sqlContext.createDataFrame(data, ['name', 'age'])  
[Row(name=u'Alice', age=1), Row(name=u'Bob', age=2)]  
>>> df2 = sqlContext.createDataFrame(data2,[ 'name', 'height'])  
[Row(name=u'Chris', height=80), Row(name=u'Bob', height=85)]  
  
>>> df.join(df2, 'name', 'right_outer')  
[Row(name=u'Chris', age=None, height=80),  
 Row(name=u'Bob', age=2, height=85)]
```

Right Outer Join – X.**join**(Y, cols,
 'right_outer')

» Return DF of rows with matching cols in Y



Online Documentation

<https://spark.apache.org/docs/latest/>

Spark 1.6.1 Overview Programming Guides API Docs Deploying More

Spark Overview

Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including [Spark SQL](#) for SQL and structured data processing, [MLlib](#) for machine learning, [GraphX](#) for graph processing, and [Spark Streaming](#).

Downloading

Get Spark from the [downloads page](#) of the project website. This documentation is for Spark version 1.6.1. Spark uses Hadoop's client libraries for HDFS and YARN. Downloads are pre-packaged for a handful of popular Hadoop versions. Users can also download a "Hadoop free" binary and run Spark with any Hadoop version [by augmenting Spark's classpath](#).

If you'd like to build Spark from source, visit [Building Spark](#).

Spark runs on both Windows and UNIX-like systems (e.g. Linux, Mac OS). It's easy to run locally on one machine — all you need is to have java installed on your system PATH, or the JAVA_HOME environment variable pointing to a Java installation.

Spark runs on Java 7+, Python 2.6+ and R 3.1+. For the Scala API, Spark 1.6.1 uses Scala 2.10. You will need to use a compatible Scala version (2.10.x).



Databricks Guide

Visualizations / Visualizations Overview (Python)

Import Notebook
R Dataset Util - Example

Sending Email - py
Sending Email - scala
Server-Side Encryption

Visualizations

Visualizations Overview
Visualizations in Python
Visualizations in R
Visualizations in SQL
Visualizations in Scala
Charts & Graphs - py
Charts & Graphs - scala
HTML, D3, and SVG
Matplotlib and GGPlot

Spark

Spark
Intro Datasets
Debugging
Caching
Run C++ Code - py
Run C++ Code - scala
Monte Carlo Simulation

Visualizations Overview

This notebook outlines your various options for visualizations with Databricks.

Utilize the built-in visualizations for Databricks

Spark DataFrames can be displayed visually in Databricks and can be configured with just a few clicks.

Learn about the basics of using visualizations in your preferred language:

- [Visualization Basics in Python](#)
- [Visualization Basics in Scala](#)
- [Visualization Basics in SQL](#)
- [Visualization Basics in R](#)

To get more information about the various charts and graph types:

- [Charts & Graphs in Python](#)
- [Charts & Graphs in Scala](#)
- For SQL, see either of the notebooks above. Python or Scala is just used to generate the data displayed in the graph.





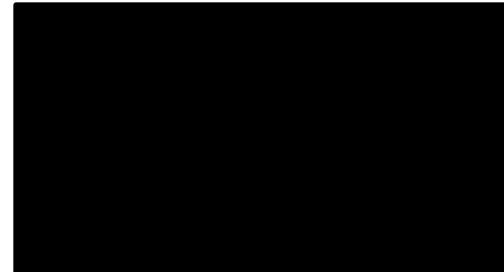
Spark Technical Blogs

Databricks: <https://databricks.com/blog/category/engineering>

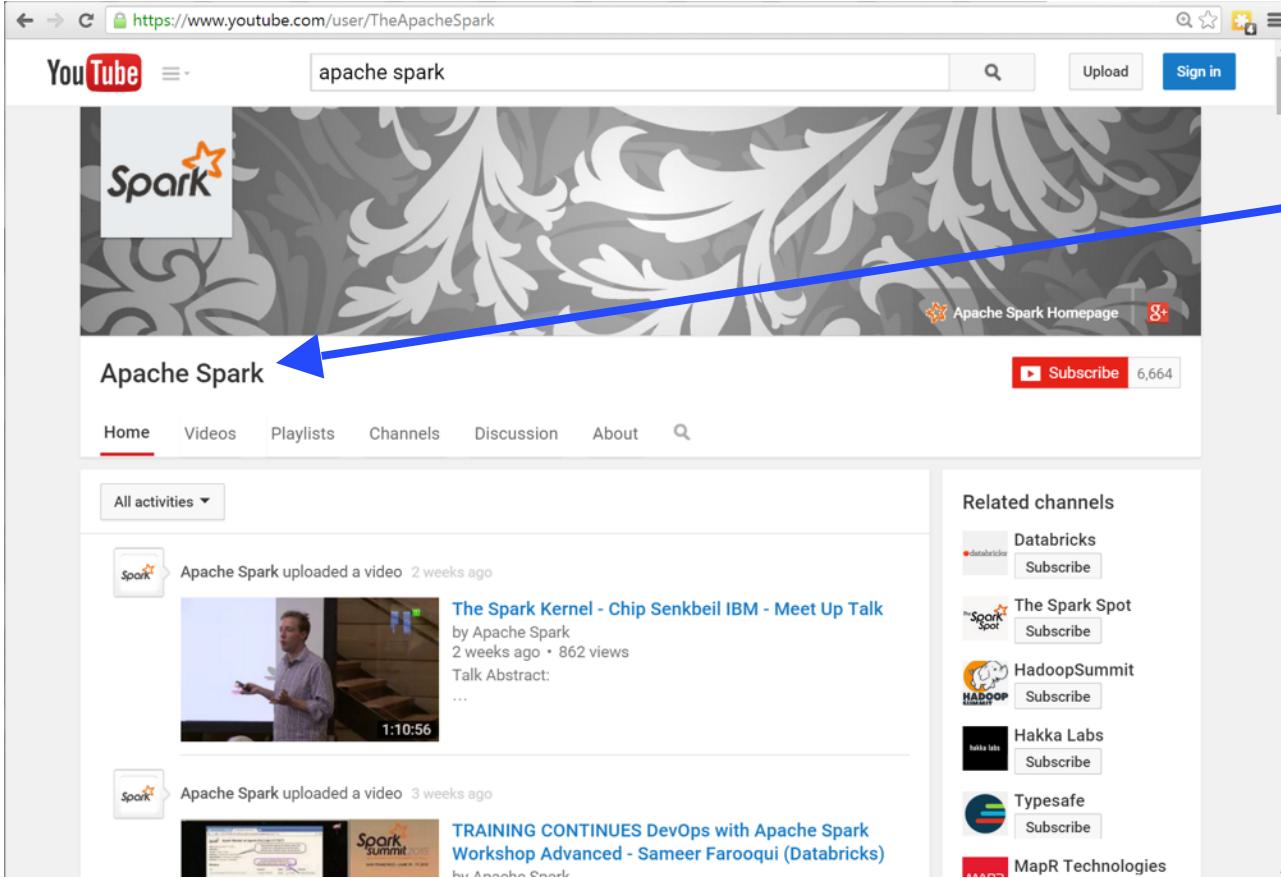
Cloudera: <http://blog.cloudera.com/blog/category/spark/>

IBM: <http://www.spark.tc/blog/>

- Hortonworks: <http://hortonworks.com/blog/category/spark/>
- Many more! (eBay, AWS, MapR, Datastax, etc)



Spark on YouTube



A screenshot of a web browser showing the Apache Spark YouTube channel page. The URL in the address bar is <https://www.youtube.com/user/TheApacheSpark>. The search bar contains the text "apache spark". The main header features the Apache Spark logo. Below the header, there's a large decorative image with a floral pattern. A blue arrow points from the text "Apache Spark" in the heading area down to the channel name "Apache Spark" in the top left of the main content area. The main content area includes a "Subscribe" button with 6,664 subscribers. Below the subscribe button, there are links for "Home", "Videos", "Playlists", "Channels", "Discussion", and "About". On the left, there's a "All activities" section showing a recent upload by Apache Spark: "The Spark Kernel - Chip Senkbeil IBM - Meet Up Talk" uploaded 2 weeks ago with 862 views. On the right, there's a "Related channels" section listing "Databricks", "The Spark Spot", "HadoopSummit", "Hakka Labs", "Typesafe", and "MapR Technologies", each with a "Subscribe" button.

Check out the
Apache Spark
YouTube
Channel!



Community

<http://spark.apache.org/community.html>

The screenshot shows the Apache Spark community page. A blue circle highlights the "Mailing Lists" section under "Spark Community". Another blue circle highlights the "Events and Meetups" section under "Conferences".

Spark Community

Mailing Lists

Get help using Spark or contribute to the project on our mailing lists:

- user@spark.apache.org is for usage questions, help, and announcements. ([subscribe](#)) ([unsubscribe](#)) ([archives](#))
- dev@spark.apache.org is for people who want to contribute code to Spark. ([subscribe](#)) ([unsubscribe](#)) ([archives](#))

The StackOverflow tag [apache-spark](#) is an unofficial but active forum for Spark users' questions and answers.

Events and Meetups

Conferences

- Spark Summit Europe 2015. Oct 27 - Oct 29 in Amsterdam.
- Spark Summit 2015. June 15 - 17 in San Francisco.

[Download Spark](#)

Latest News

Spark 1.4.0 released (Jun 11, 2015)
One month to Spark Summit 2015 in San Francisco (May 15, 2015)
Announcing Spark Summit Europe (May 15, 2015)
Spark Summit East 2015 Videos Posted (Apr 20, 2015)

[Archive](#)

Spark Meetups

Apache Spark

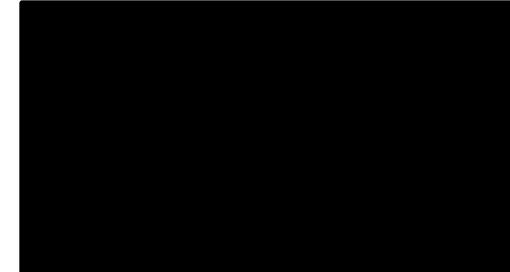
Find out what's happening in Apache Spark Meetup groups around the world and start meeting up with the ones near you.

186,279
members | 421
Meetups

[Join Apache Spark Meetups](#)

<http://spark.meetup.com/>

Related topics: [Big Data](#) · [Hadoop](#) · [Machine Learning](#) · [Data Analytics](#) · [Big Data Analytics](#) · [Data Science](#) · [Apache Kafka](#) · [MapReduce](#) · [Data Mining](#) · [Scala](#)



Databricks Forums

<https://forums.databricks.com/>

The screenshot shows the Databricks Forums homepage. At the top, there's a search bar with placeholder text "Find posts, topics, and users..." and a magnifying glass icon. To the right of the search bar are buttons for "Ask a question" and "Sign in". Below the search bar, there's a "Popular Topics" sidebar with a grid of tags: data-management, aws, notebooks, spark sql, s3, spark, cluster-resources, pyspark, admin-debugging-tuning, python, library-management, dataframes, rdd, performance, jdbc, memory, data frames, ec2, milib, cluster, scala, cluster provisioning, hive, dbfs, jobs, parquet files, sql, spark streaming, visualizations, streaming. A "View all >" link is at the bottom of the sidebar.

The main content area displays a list of questions:

- Can the REST API be configured to use a different port number?
1 Reply 0 Likes
rest-api · rest
- Source Control and Debugging options
0 Replies 0 Likes
debug · debugging · git · source control
- Why do I get 'java.io.IOException: File already exists' for saveAsTable with Overwrite mode?
0 Replies 0 Likes
hive · dataframe · saveable
- What s3 bucket does DBFS use? And where is the local cache files?
0 Replies 0 Likes
dbfs · dbutils
- bug in notebook - type mismatch error
0 Replies 0 Likes
notebook · case class · type mismatch
- How do I set up VPC Peering to connect my servers/databases to my Databricks Spark Clusters?
0 Replies 0 Likes
aws · vpc
- Read HDFS file from python script
0 Replies 0 Likes
python · hadoop

Community forum for Databricks users

Mostly Databricks-specific Q&A

Some general Spark Q&A

Spark Packages

A screenshot of a web browser displaying the [Spark Packages](http://spark-packages.org) website. The page shows a community index of packages for Apache Spark. There are 73 packages listed. The first three packages shown are:

- spark-avro**: Integration utilities for using Spark with Apache Avro data. From: @databricks / Owner: @pwendell / Latest release: 1.0.0 (04/10/15) / Apache-2.0 / ★★★★★ (17)
4 sql | 3 input | 2 library
- spark-redshift**: Spark and Redshift integration. From: @databricks / Owner: @pwendell / Latest release: 0.4.0-hadoop2 (05/20/15) / Apache-2.0 / ★★★★★ (12)
1 input | 1 sql | 1 redshift
- kafka-spark-consumer**: Low Level Kafka-Spark Consumer. From: @dibbhatt / Latest release: 1.0.2 (06/02/15) / Apache-2.0 / ★★★★★ (14)
3 streaming | 2 kafka

At the bottom of the page, there is a note: "Spark Packages is a community site hosting modules that are not part of Apache Spark. Your use of and access to this site is subject to the terms of use. Apache Spark and the Spark logo are trademarks of the Apache Software Foundation. This site is maintained as a community service by Databricks."

<http://spark-packages.org/>

232 software packages for Spark

- » User-provided Spark extensions
- » Community votes  (8)

Spark Source Code

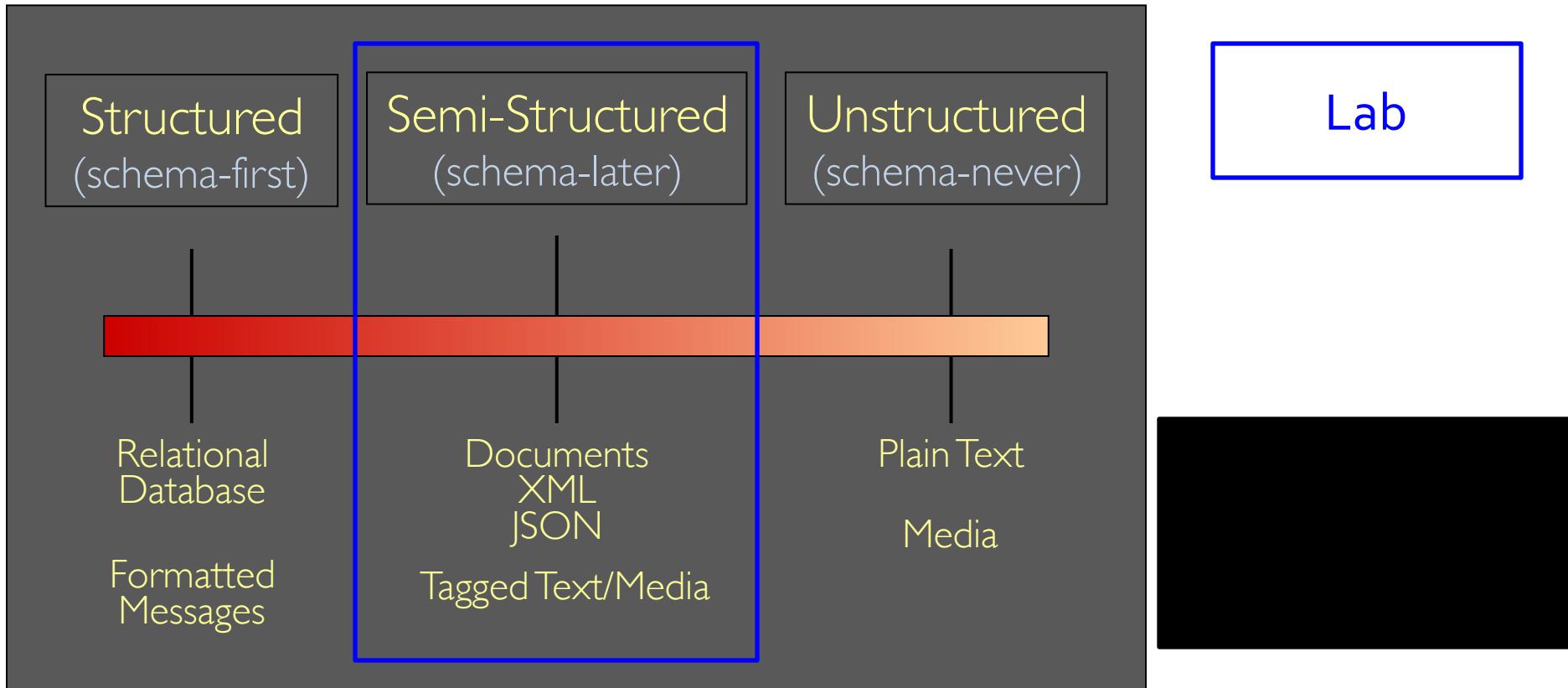
The screenshot shows the GitHub repository page for Apache Spark. At the top, there's a navigation bar with links for 'Pull requests', 'Issues', and 'Gist'. Below the navigation is a header for the 'apache / spark' repository, which is a mirror from `git://git.apache.org/spark.git`. The repository has 12,036 commits, 12 branches, 36 releases, and 611 contributors. A dropdown menu shows the current branch is 'master'. The main area displays a list of recent commits, each with a small icon, the commit message, and the time since it was made. On the right side, there's a sidebar with options for 'Code', 'Pull requests' (325), 'Pulse', 'Graphs', and cloning options via SSH or HTTPS. A large black rectangular redaction box covers the bottom right portion of the page.

Commit	Message	Time Ago
R	[SPARK-9201] [ML] Initial integration of MLlib + SparkR using RFormula	14 hours ago
assembly	[SPARK-7801] [BUILD] Updating versions to SPARK 1.5.0	2 months ago
bagel	[SPARK-7801] [BUILD] Updating versions to SPARK 1.5.0	2 months ago
bin	[SPARK-7733] [CORE] [BUILD] Update build, code to use Java 7 for 1.5.0+	a month ago
build	[SPARK-8933] [BUILD] Provide a --force flag to build/mvn that always ...	7 days ago
conf	[SPARK-3071] Increase default driver memory	20 days ago
core	[SPARK-5423] [CORE] Register a TaskCompletionListener to make sure re...	an hour ago
data/mllib	[MLLIB] [DOC] Seed fix in mllib naive bayes example	3 days ago
dev	[SPARK-8401] [BUILD] Scala version switching build enhancements	8 hours ago
docker	[SPARK-8954] [BUILD] Remove unneeded deb repository from Dockerfile t...	8 days ago
docs	[SPARK-5989] [MLLIB] Model save/load for LDA	38 minutes ago
ec2	[SPARK-8596] Add module for rstdio link to spark	8 days ago
examples	[SPARK-7977] [BUILD] Disallowing println	11 days ago
external	[SPARK-8962] Add Scalastyle rule to ban direct use of Class.forName; ...	7 days ago

<https://github.com/apache/spark/>

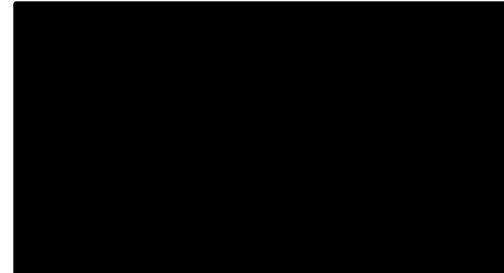
Hint: For detailed explanations, check out comments in code

The Structure Spectrum



Semi-Structured Log Files

- Created by `printf` statements in server processes:
 - » Web, database, network file servers, operating system components
- Human-readable text format files
 - » Very rarely actually read by a human
 - » Can store/archive in binary or compressed format
- Format published or “defined” by code
 - » Can be very difficult to parse



Recall: Apache Web Server Log

```
uplherc.upl.com - - [01/Aug/1995:00:00:07 -0400] "GET / HTTP/1.0" 304 0
uplherc.upl.com - - [01/Aug/1995:00:00:08 -0400] "GET /images/ksclogo-medium.gif HTTP/
1.0" 304 0
uplherc.upl.com - - [01/Aug/1995:00:00:08 -0400] "GET /images/MOSAIC-logosmall.gif
HTTP/1.0" 304 0
uplherc.upl.com - - [01/Aug/1995:00:00:08 -0400] "GET /images/USA-logosmall.gif HTTP/
1.0" 304 0
ix-esc-ca2-07.ix.netcom.com - - [01/Aug/1995:00:00:09 -0400] "GET /images/launch-
logo.gif HTTP/1.0" 200 1713
uplherc.upl.com - - [01/Aug/1995:00:00:10 -0400] "GET /images/WORLD-logosmall.gif HTTP/
1.0" 304 0
slppp6.intermind.net - - [01/Aug/1995:00:00:10 -0400] "GET /history/skylab/skylab.htm
HTTP/1.0" 200 1687
piweba4y.prodigy.com - - [01/Aug/1995:00:00:10 -0400] "GET /images/launchmedium.gif
HTTP/1.0" 200 11853
tampico.usc.edu - - [14/Aug/1995:22:57:13 -0400] "GET /welcome.html HTTP/1.0" 200 790
```

Apache Web Server Log Format

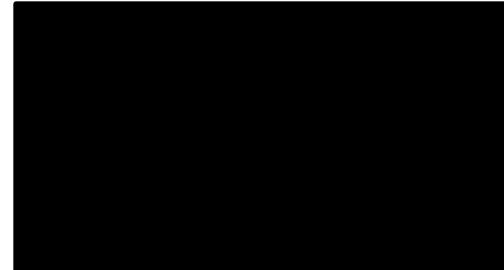
- [Apache Common Log Format](#) specifies log file format
- Example line from log file:
 - » `127.0.0.1 - - [01/Aug/1995:00:00:01 -0400] "GET /images/launch-logo.gif HTTP/1.0"`
`200 1839`
- Components:
 - » `127.0.0.1` Client IP address

Apache Web Server Log Format

- Apache Common Log Format specifies log file format
- Example line from log file:
 - » 127.0.0.1 - - [01/Aug/1995:00:00:01 -0400] "GET /images/launch-logo.gif HTTP/1.0" 200 1839
- Components:
 - » - User *identity from remote machine*
(hyphen means not available)
 - » - User *identity from local logon*
(hyphen means not available)

Apache Web Server Log Format

- [Apache Common Log Format](#) specifies log file format
- Example line from log file:
 - » 127.0.0.1 - - [01/Aug/1995:00:00:01 -0400] "GET /images/launch-logo.gif HTTP/1.0" 200 1839
- Components:
 - » [01/Aug/1995:00:00:01 -0400]
 - Request time



Apache Web Server Log Format

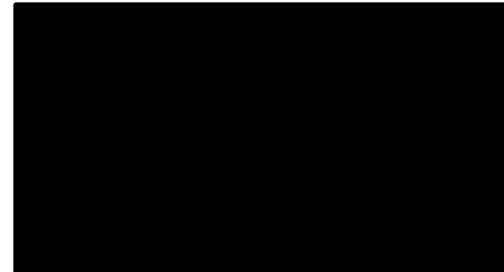
- [Apache Common Log Format](#) specifies log file format
- Example line from log file:
 - » 127.0.0.1 - - [01/Aug/1995:00:00:01 -0400] "GET /images/launch-logo.gif HTTP/1.0" 200 1839
- Components:
 - » "GET /images/launch-logo.gif HTTP/1.0"
Client request
 - Request method (e.g., GET, POST, etc.)
 - Endpoint (a Uniform Resource Identifier)
 - Client protocol version

Apache Web Server Log Format

- Apache Common Log Format specifies log file format
- Example line from log file:
 - » 127.0.0.1 - - [01/Aug/1995:00:00:01 -0400] "GET /images/launch-logo.gif HTTP/1.0"
200 1839
- Components:
 - » **200** Status code the server sent back to the client
 - OK response (2xx), others: 3xx, 4xx, 5xx
 - » **1839** Size of the object returned to client
"-" if no content returned, or sometimes 0

Lab: Explore Web Server Access Log

- NASA HTTP server access log
 - » <http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html>
- Log covers 21 days (1 Aug, 3 Aug – 22 Aug 1995)
- Log includes 1,043,177 requests
- Log partially cleaned for you
 - » Removed some very hard to parse requests



Some Log Analysis Questions

- Overall:
 - » What are the statistics for content being returned? Sizes, statuses?
 - » What are the types of return codes?
 - » How many 404 (page not found) errors are there?
- Temporal:
 - » How many unique hosts per day?
 - » How many requests per day?
 - » On average, how many requests per host?
 - » How many 404 errors per day?



Research Papers

Spark: Cluster Computing with Working Sets

Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica
University of California, Berkeley

Abstract

MapReduce and its variants have been highly successful in implementing large-scale data-intensive applications on commodity clusters. However, most of these systems are built around an acyclic data flow model that is not suitable for other popular applications. This paper focuses on one such class of applications: those that reuse a working set of data across multiple parallel operations. This includes many iterative machine learning algorithms, as well as interactive data analysis tools. We propose a new framework called Spark that supports these applications while retaining the scalability and fault tolerance of MapReduce. To achieve these goals, Spark introduces an abstraction called resilient distributed datasets (RDDs). An RDD is a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Spark can outperform Hadoop by 10x in iterative machine learning jobs, and can be used to interactively query a 39 GB dataset with sub-second response time.

1 Introduction

A new model of cluster computing has become widely popular, in which data-parallel computations are executed on clusters of unreliable machines by systems that automatically handle job scheduling, fault tolerance, and load balancing. MapReduce [11] pioneered this model, while systems like Dryad [17] and MapReduce-Merge [24] generalized the types of data flows supported. These systems achieve their scalability and fault tolerance by providing a programming model where the user creates acyclic data flow graphs to pass input data through a set of operators. This allows the underlying system to manage scheduling and to react to faults without user intervention.

While this data flow programming model is useful for a large class of applications, there are applications that cannot be expressed efficiently as acyclic data flows. In this paper, we focus on one such class of applications: those that reuse a *working set* of data across multiple parallel operations. This includes two use cases where we have seen Hadoop users report that MapReduce is deficient:

- **Iterative jobs:** Many common machine learning algorithms apply a function repeatedly to the same dataset to optimize a parameter (e.g., through gradient descent). While each iteration can be expressed as a

MapReduce/Dryad job, each job must reload the data from disk, incurring a significant performance penalty.

- **Interactive analytics:** Hadoop is often used to run ad-hoc exploratory queries on large datasets, through SQL-like interfaces like Pig [23] or Hive [1]. Ideally, a user would be able to load a dataset of interest into memory across a number of machines and query it repeatedly. However, with Hadoop, each query incurs significant latency (tens of seconds) because it runs as a separate MapReduce job and reads data from disk.

This paper presents a new cluster computing framework called Spark, which supports applications with working sets while providing similar scalability and fault tolerance properties to MapReduce.

The main abstraction in Spark is that of a *resilient distributed dataset* (RDD), which represents a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Users can explicitly cache an RDD in memory across machines and reuse it in multiple MapReduce-like *parallel operations*. RDDs achieve fault tolerance through a notion of *lineage*: if a partition of an RDD is lost, the RDD has enough information about how it was derived from other RDDs to be able to rebuild just that partition. Although RDDs are not as expressive as MapReduce, they represent a sweet-spot between expressivity on the one hand and scalability and reliability on the other hand, and we have found them well-suited for a variety of applications.

Spark is implemented in Scala [5], a statically typed high-level programming language for the Java VM, and exposes a functional programming interface similar to DryadLINQ [25]. In addition, Spark can be used interactively from a modified version of the Scala interpreter, which allows the user to define RDDs, functions, variables and classes and use them in parallel operations on a cluster. We believe that Spark is the first system to allow an efficient, general-purpose programming language to be used interactively to process large datasets on a cluster.

Although our implementation of Spark is still a prototype, early experience with the system is encouraging. We show that Spark can outperform Hadoop by 10x in iterative machine learning workloads and can be used interactively to scan a 39 GB dataset with sub-second latency.

This paper is organized as follows. Section 2 describes

Spark: Cluster Computing with Working Sets

http://www.cs.berkeley.edu/~matei/papers/2010/hotcloud_spark.pdf



Research Papers

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

Matei Zaharia, Mosharaf Chowdhury, Taghata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica
University of California, Berkeley

Abstract

We present Resilient Distributed Datasets (RDDs), a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner. RDDs are motivated by two types of applications that current computing frameworks handle inefficiently: iterative algorithms and interactive data mining tools. In both cases, keeping data in memory can improve performance by an order of magnitude. To achieve fault tolerance efficiently, RDDs provide a restricted form of shared memory, based on coarse-grained transformations rather than fine-grained updates to shared state. However, we show that RDDs are expressive enough to support a wide class of computations, including recent specialized programming models for iterative jobs, such as Pregel, and new applications that these models do not capture. We have implemented RDDs in a system called Spark, which we evaluate through a variety of user applications and benchmarks.

1 Introduction

Cluster computing frameworks like MapReduce [10] and Dryad [19] have been widely adopted for large-scale data analytics. These systems let users write parallel computations using a set of high-level operators, without having to worry about work distribution and fault tolerance.

Although current frameworks provide numerous abstractions for accessing a cluster's computational resources, they lack an abstraction for in-memory shared memory. This makes them inefficient for an important class of emerging applications: those that reuse intermediate results across multiple computations. Data reuse is common in many *iterative* machine learning and graph algorithms, including PageRank, K-means clustering, and logistic regression. Another compelling use case is *interactive* data mining, where a user runs multiple ad-hoc queries on the same subset of the data. Unfortunately, in most current frameworks, the only way to reuse data between computations (e.g., between two MapReduce jobs) is to write it to an external stable storage system, e.g., a distributed file system. This incurs substantial overheads due to data replication, disk I/O, and serializa-

tion, which can dominate application execution times. Recognizing this problem, researchers have developed specialized frameworks for some applications that require data reuse. For example, Pregel [22] is a system for iterative graph computations that keeps intermediate data in memory, while HaLoop [7] offers an iterative MapReduce interface. However, these frameworks only support specific computation patterns (e.g., looping a series of MapReduce steps), and perform data sharing implicitly for these patterns. They do not provide abstractions for more general programs that let users load several datasets into memory and run ad-hoc queries across them.

In this paper, we propose a new abstraction called *resilient distributed datasets (RDDs)* that enables efficient data reuse in a broad range of applications. RDDs are fault-tolerant, parallel data structures that let users explicitly persist intermediate results in memory, control their partitioning to optimize data placement, and manipulate them using a rich set of operators.

The main challenge in designing RDDs is defining a programming interface that can provide fault tolerance *efficiently*. Existing abstractions for in-memory storage on clusters, such as distributed shared memory [24], key-value stores [25], databases, and Pseudo RAM, offer an interface based on fine-grained updates to mutable state (e.g., cells in a table). With this interface, the only ways to provide fault tolerance are to replicate the data across machines or to log updates across machines. Both approaches are expensive for data-intensive workloads, as they require copying large amounts of data over the cluster network, whose bandwidth is far lower than that of RAM, and they incur substantial storage overhead.

In contrast to these systems, RDDs provide an interface based on *coarse-grained* transformations (e.g., map, filter and join) that apply the same operation to many data items. This allows them to efficiently provide fault tolerance by logging the state of the data as it is built up (in *image*) rather than the actual data.¹ If a portion of an RDD is lost, the RDD has enough information about how it was derived from other RDDs to recompute

¹Checkpointing the data in some RDDs may be useful when a lineage chain grows large, however, and we discuss how to do it in §5.4.

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

http://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf

April 2012



Spark SQL: Relational Data Processing in Spark

Michael Armbrust[†], Reynold S. Xin[†], Cheng Lian[†], Yin Huai[†], Davies Liu[†], Joseph K. Bradley[†], Xiangrui Meng[†], Tomer Kaftan[‡], Michael J. Franklin[‡], Ali Ghodsi[‡], Matei Zaharia^{†*}

[†]Databricks Inc. [‡]MIT CSAIL [‡]AMPLab, UC Berkeley

ABSTRACT

Spark SQL is a new module in Apache Spark that integrates relational processing with Spark's functional programming API. Built on our experience with Shark, Spark SQL lets Spark programmers leverage the benefits of relational processing (*e.g.*, declarative queries and optimized storage), and lets SQL users call complex analytics libraries in Spark (*e.g.*, machine learning). Compared to previous systems, Spark SQL makes two main additions. First, it offers much tighter integration between relational and procedural processing, through a declarative DataFrame API that integrates with procedural Spark code. Second, it includes a highly extensible optimizer, Catalyst, built using features of the Scala programming language, that makes it easy to add composable rules, control code generation, and define extension points. Using Catalyst, we have built a variety of features (*e.g.*, schema inference for JSON, machine learning types, and query federation to external databases) tailored for the complex needs of modern data analysis. We see Spark SQL as an evolution of both SQL-on-Spark and of Spark itself, offering richer APIs and optimizations while keeping the benefits of the Spark programming model.

Categories and Subject Descriptors

H.2 [Database Management]: Systems

Keywords

Databases; Data Warehouse; Machine Learning; Spark; Hadoop

1 Introduction

Big data applications require a mix of processing techniques, data sources and storage formats. The earliest systems designed for these workloads, such as MapReduce, gave users a powerful, but

While the popularity of relational systems shows that users often prefer writing declarative queries, the relational approach is insufficient for many big data applications. First, users want to perform ETL to and from various data sources that might be semi- or unstructured, requiring custom code. Second, users want to perform advanced analytics, such as machine learning and graph processing, that are challenging to express in relational systems. In practice, we have observed that most data pipelines would ideally be expressed with a combination of both relational queries and complex procedural algorithms. Unfortunately, these two classes of systems—relational and procedural—have until now remained largely disjoint, forcing users to choose one paradigm or the other.

This paper describes our effort to combine both models in Spark SQL, a major new component in Apache Spark [39]. Spark SQL builds on our earlier SQL-on-Spark effort, called Shark. Rather than forcing users to pick between a relational or a procedural API, however, Spark SQL lets users seamlessly intermix the two.

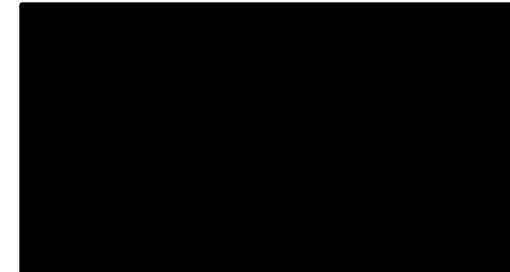
Spark SQL bridges the gap between the two models through two contributions. First, Spark SQL provides a *DataFrame API* that can perform relational operations on both external data sources and Spark's built-in distributed collections. This API is similar to the widely used data frame concept in R [32], but evaluates operations lazily so that it can perform relational optimizations. Second, to support the wide range of data sources and algorithms in big data, Spark SQL introduces a novel extensible optimizer called *Catalyst*. Catalyst makes it easy to add data sources, optimization rules, and data types for domains such as machine learning.

The DataFrame API offers rich relational/procedural integration within Spark programs. DataFrames are collections of structured records that can be manipulated using Spark's procedural API, or using new relational APIs that allow richer optimizations. They can

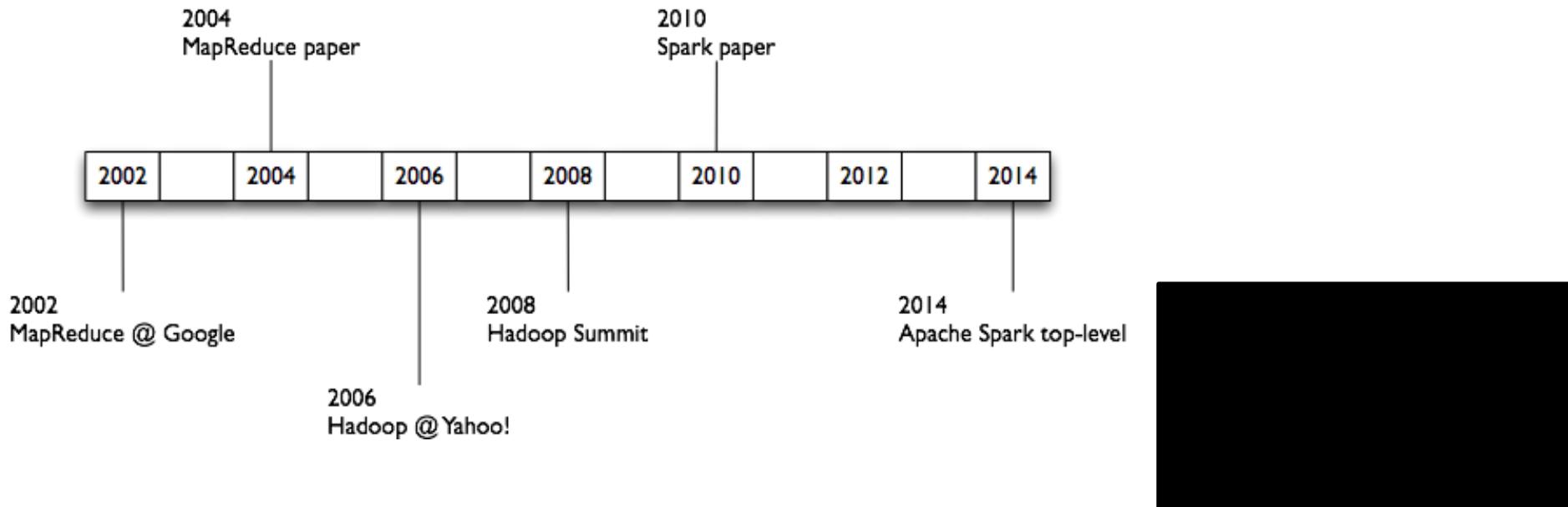
Spark SQL: Relational Data Processing in Spark

Seamlessly mix SQL queries with Spark programs

June 2015



History Summary



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