

# Introduction to Apache Spark



# This Lecture

Big Data Problems: Distributing Work, Failures, Slow Machines

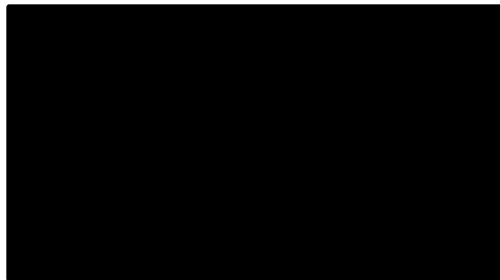
HW/SW for Big Data: Map Reduce and Apache Spark

The Structured Query Language (SQL)

SparkSQL

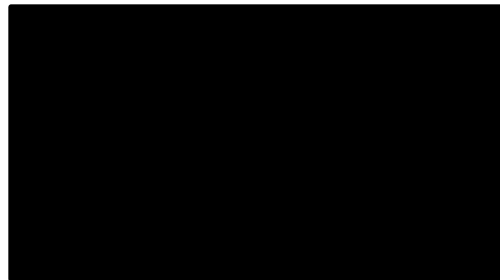
Apache Spark Resources and Community

Apache Web Server Log Files



# What is Apache Spark?

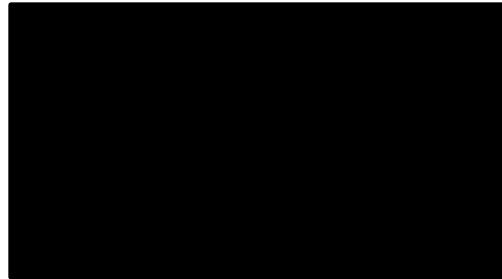

*Scalable, efficient analysis of Big Data*



# What is Apache Spark?

Scalable, efficient analysis of Big Data

*This lecture*

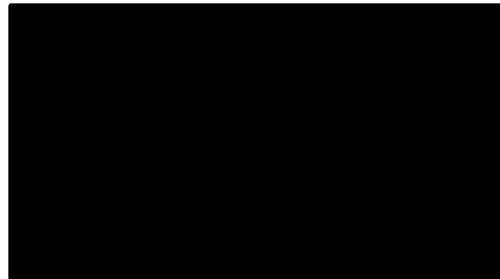


# The Big Data Problem

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?*

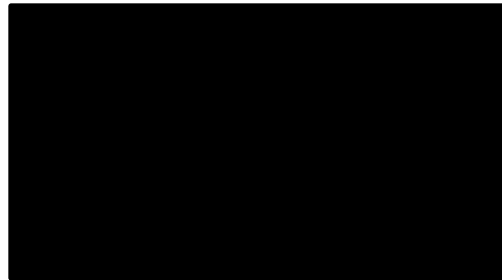
# The Opportunity

Cloud computing is a game-changer!

Provides access to low-cost computing and storage

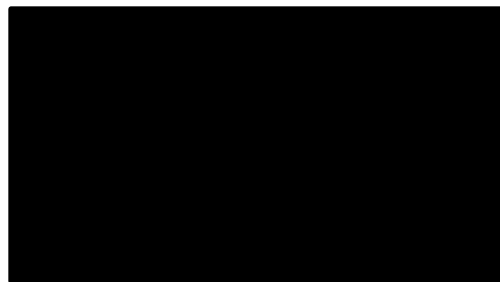
Costs decreasing every year

*The challenge is programming the resources*



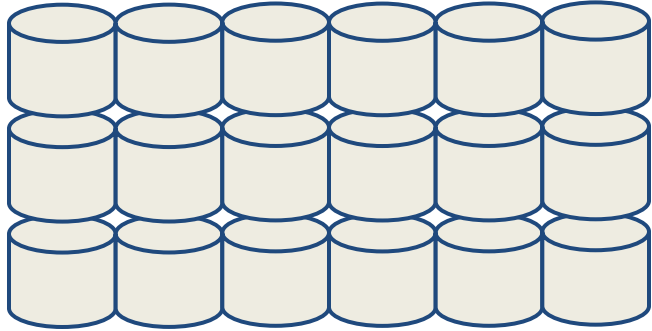
# What is Apache Spark?

- Scalable, efficient analysis of Big Data

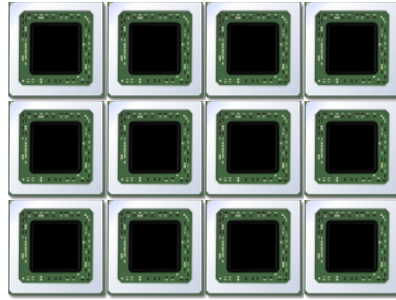




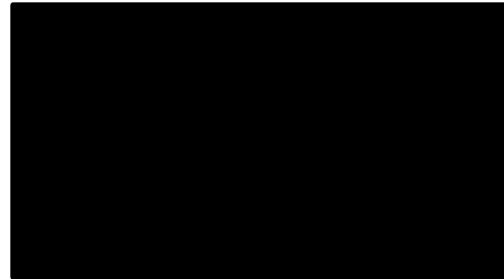
# A Brief History of Big Data Processing



Lots of hard drives



... and CPUs



# Yesterday's Hardware for Big Data

One big box!

(1990's solution)

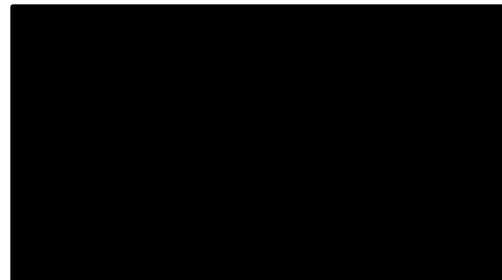
» All processors share memory

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

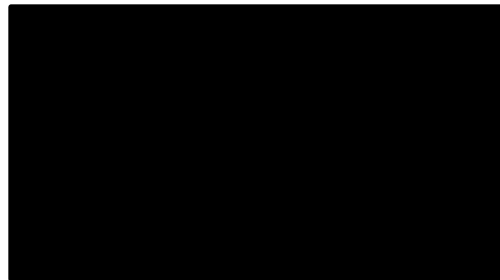
**Easy to add capacity**

**Cheaper** per CPU/disk

**But, requires complexity in software**



Image: Steve Jurvetson/Flickr



# Problems with Cheap Hardware

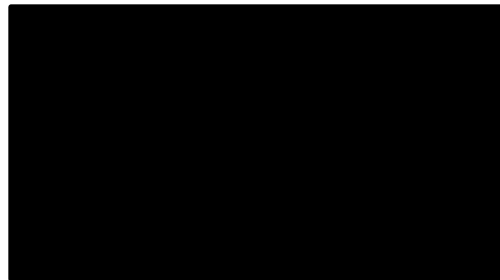
**Failures**, Google's numbers:  
1-5% hard drives/year  
0.2% DIMMs/year



Facebook Datacenter (2014)

**Network** speeds versus shared memory  
*Much* more latency  
Network slower than storage

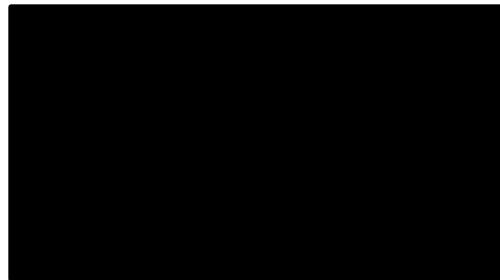
**Uneven** performance



# What's Hard About Cluster Computing?

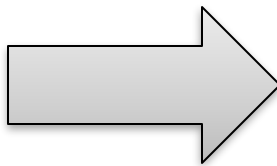
*How do we split work across machines?*

Let's look at a simple task: word counting

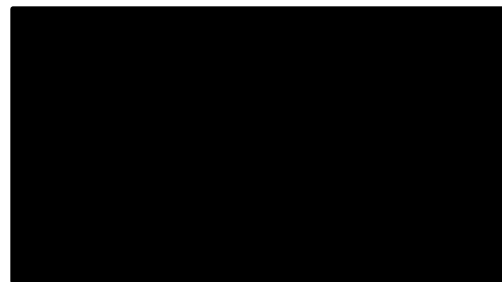


How do you count the number of occurrences of each word  
in a document?

“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

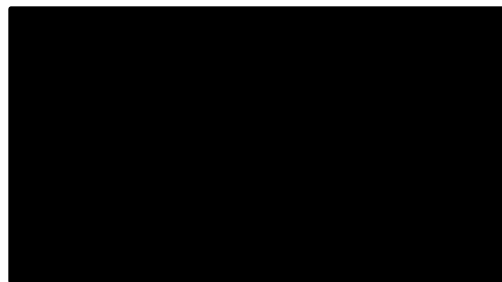
I am Sam

Sam I am

Do you like

Green eggs and ham?”

}



# One Approach: Use a Hash Table

“I am Sam

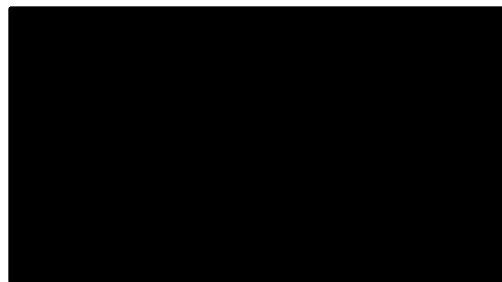
I am Sam

Sam I am

Do you like

Green eggs and ham?”

{I : 1}





# One Approach: Use a Hash Table

“I am Sam

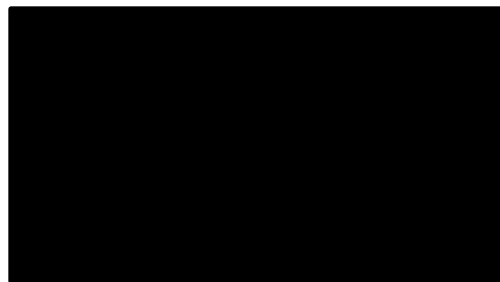
I am Sam

Sam I am

Do you like

Green eggs and ham?”

{I: **1**,  
am: **1**}



# One Approach: Use a Hash Table

“I am Sam

I am Sam

Sam I am

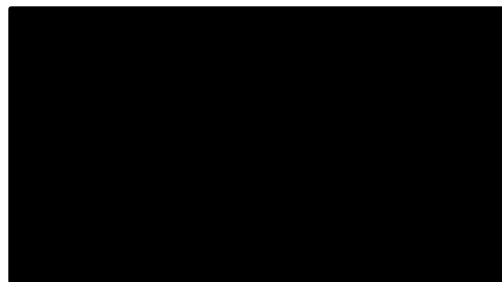
Do you like

Green eggs and ham?”

{I: **1**,

am: **1**,

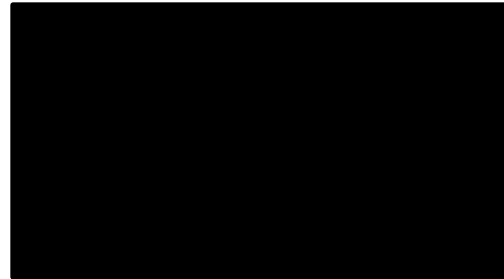
Sam: **1**}



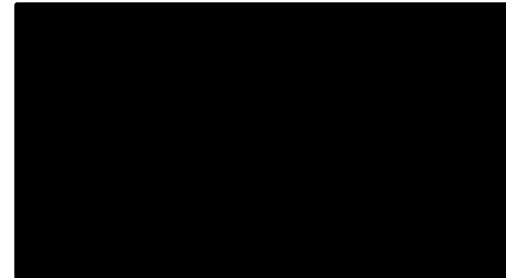
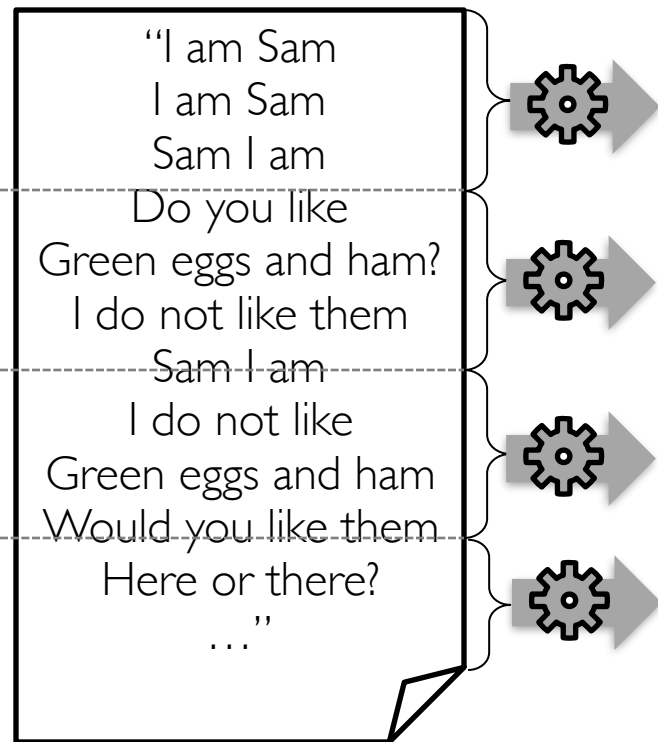
# One Approach: Use a Hash Table

“I am Sam  
I am Sam  
Sam I am  
Do you like  
Green eggs and ham?”

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



# What if the Document is Really Big?

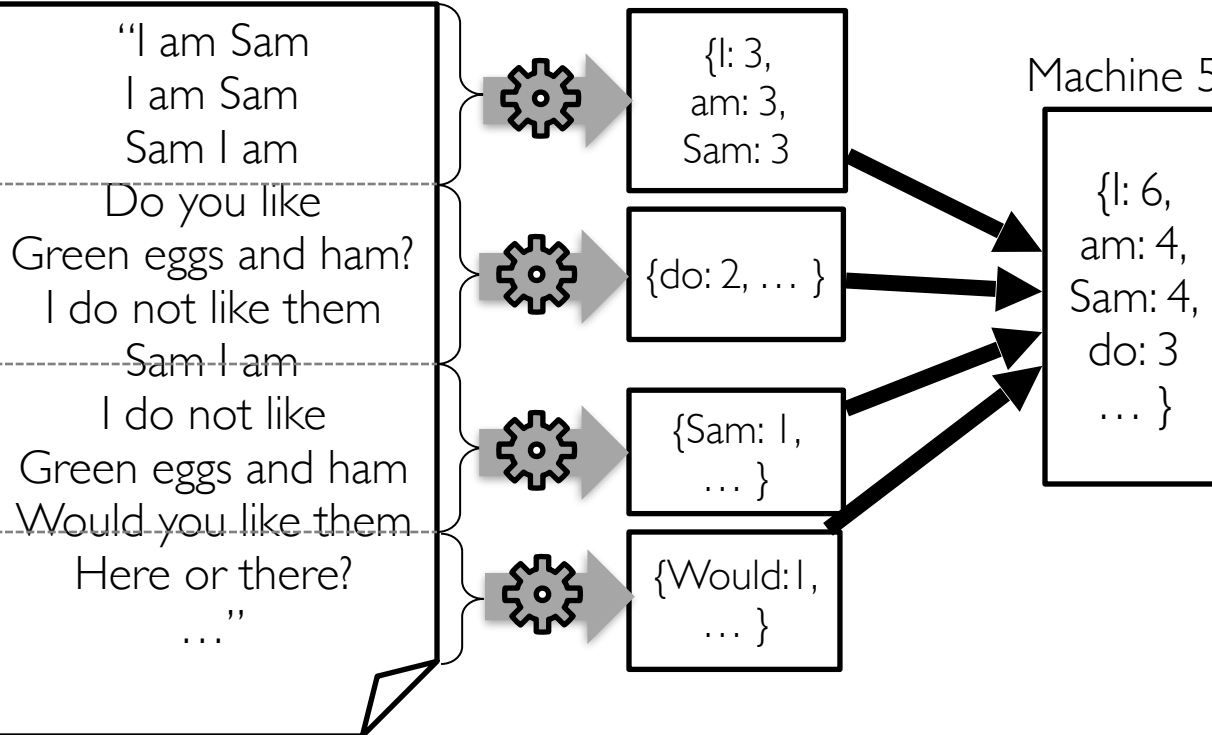


# What if the Document is Really Big?

Machines 1 - 4

Machine 5

*What's the  
problem with this  
approach?*

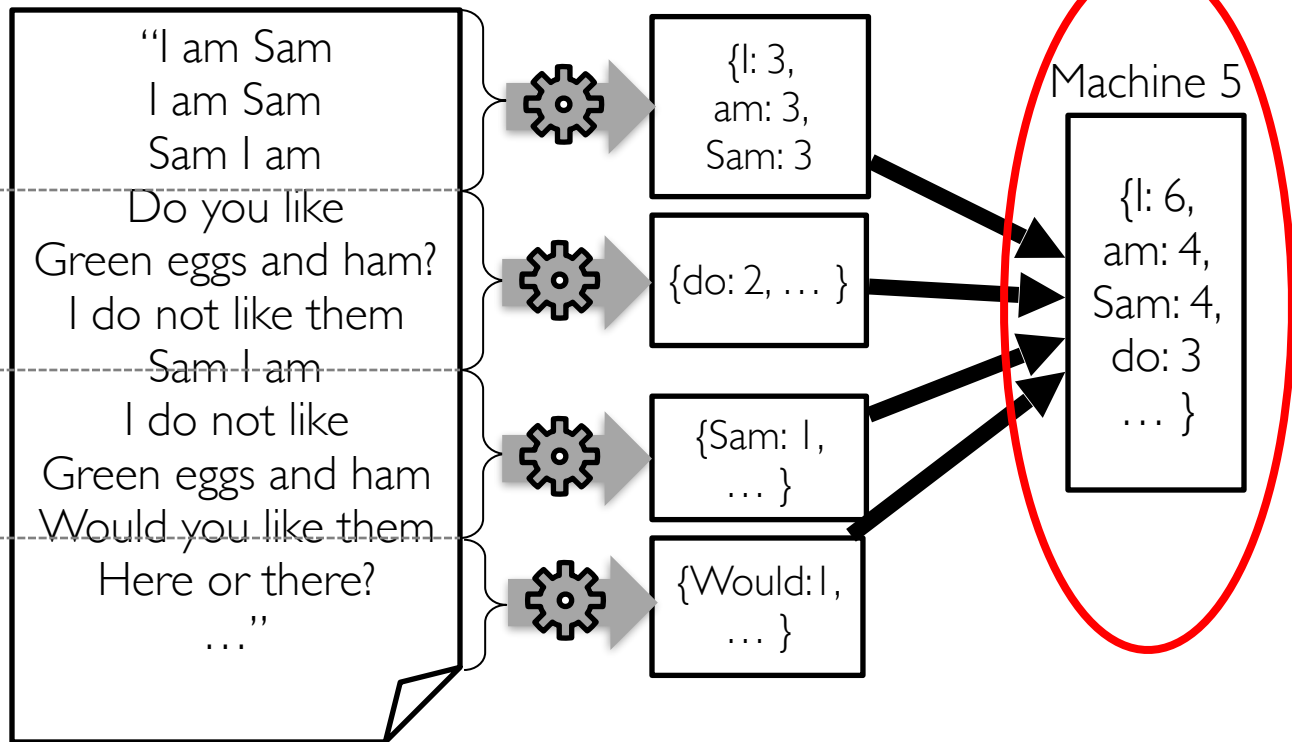


# What if the Document is Really Big?

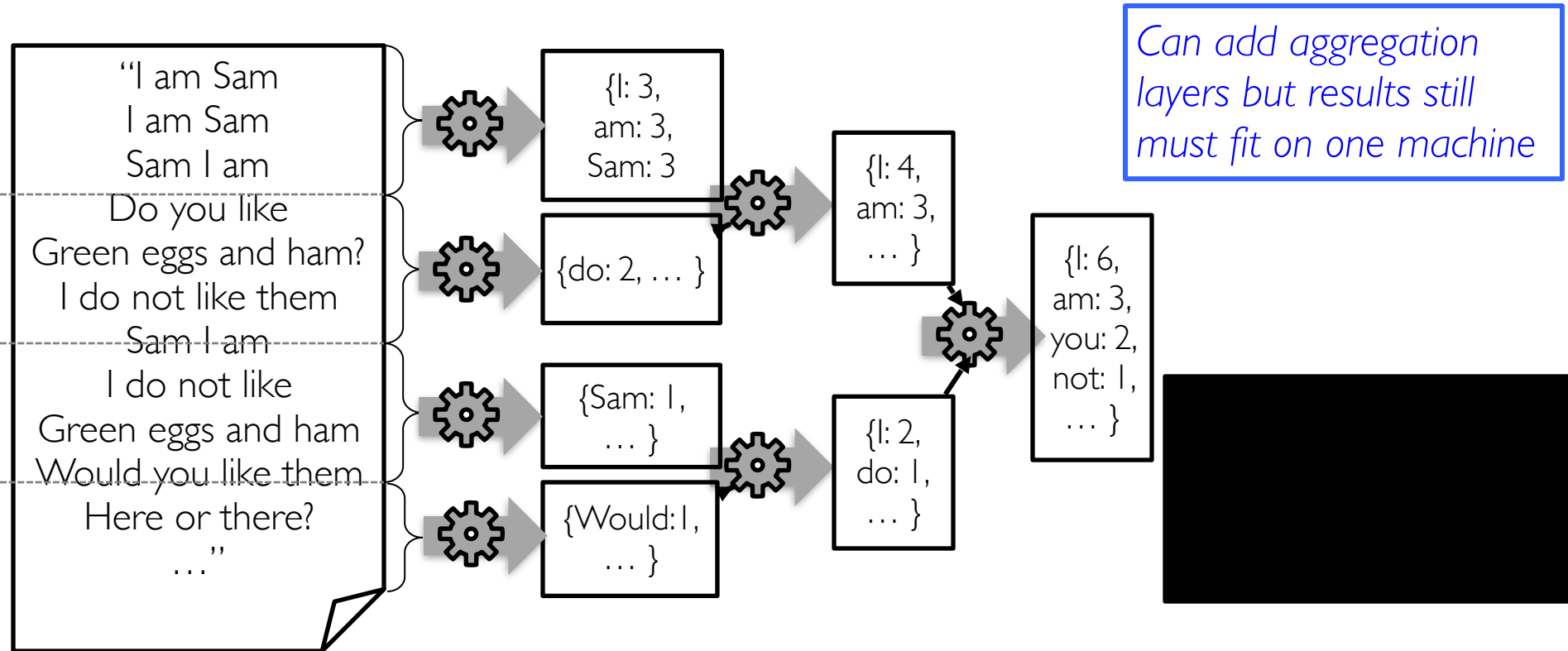
Machines 1 - 4

Machine 5

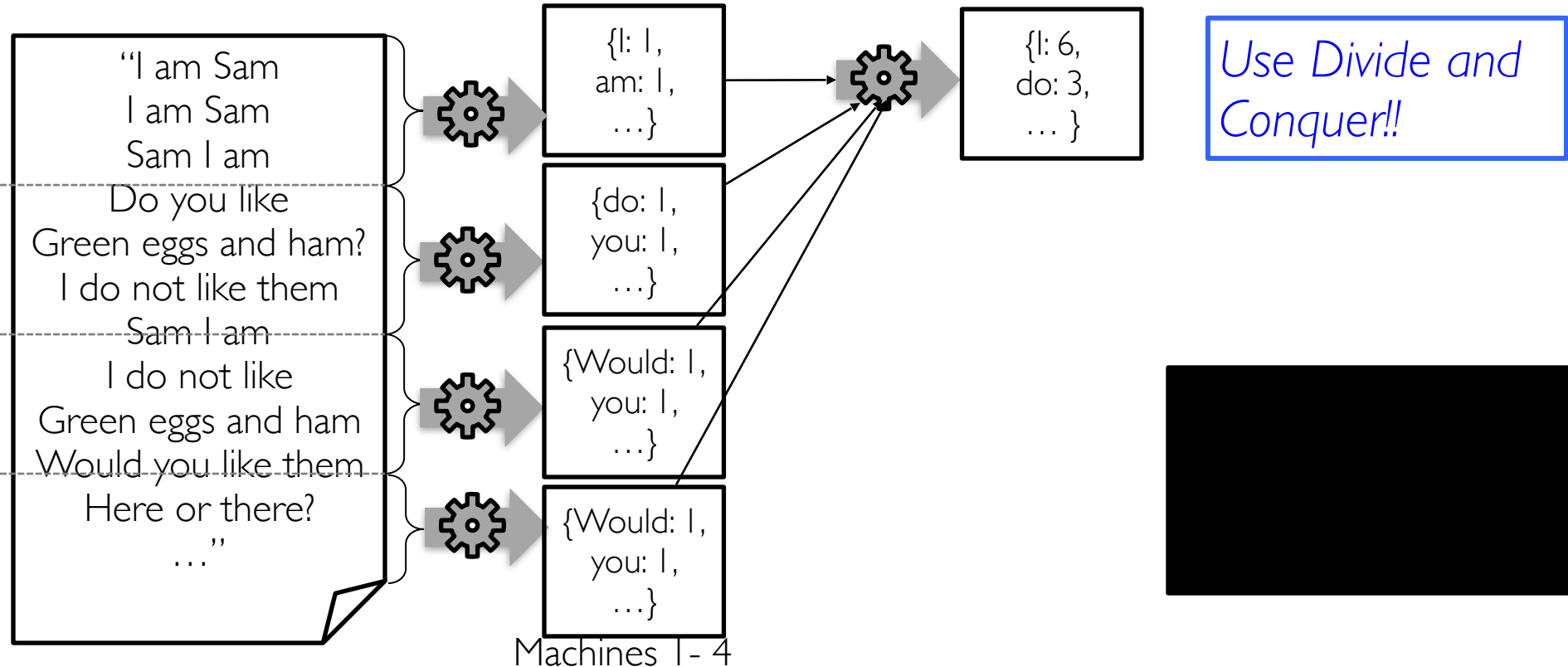
*Results have to fit  
on one machine*



# What if the Document is Really Big?

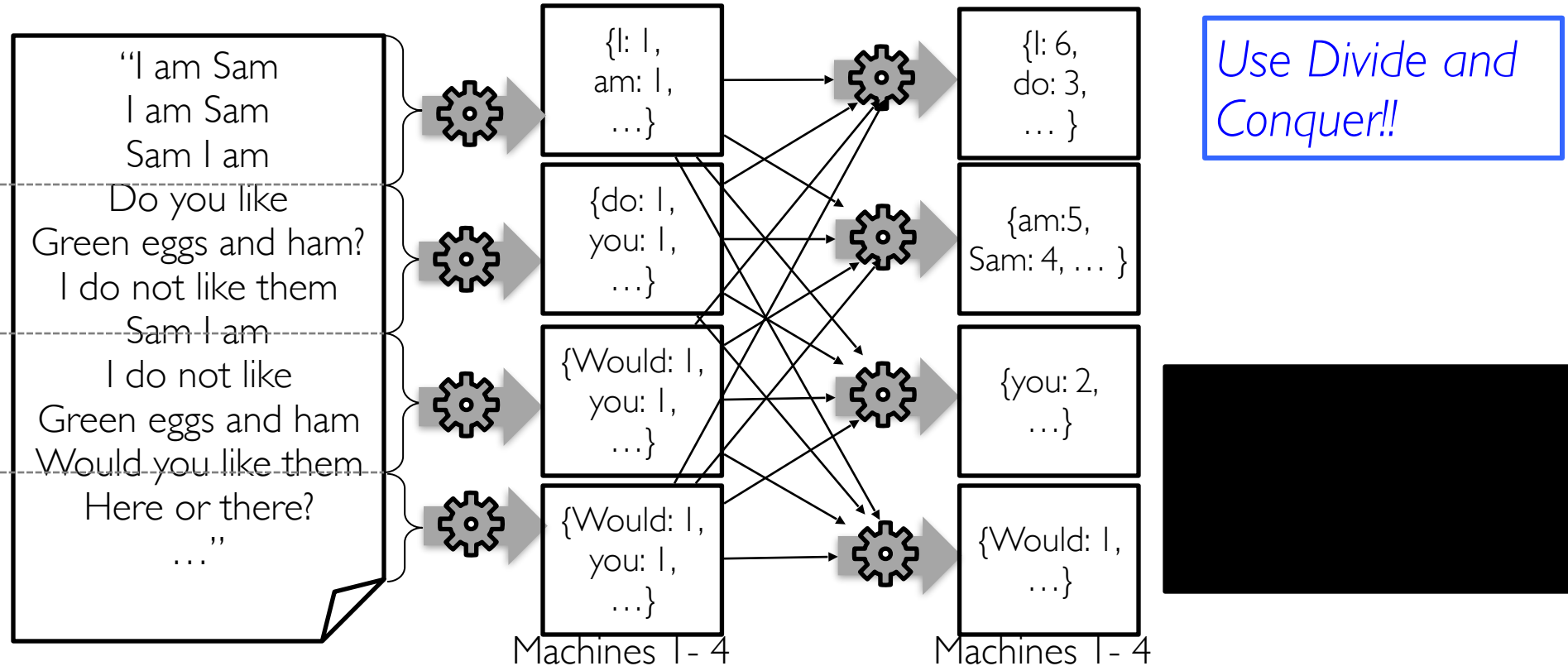


# What if the Document is Really Big?

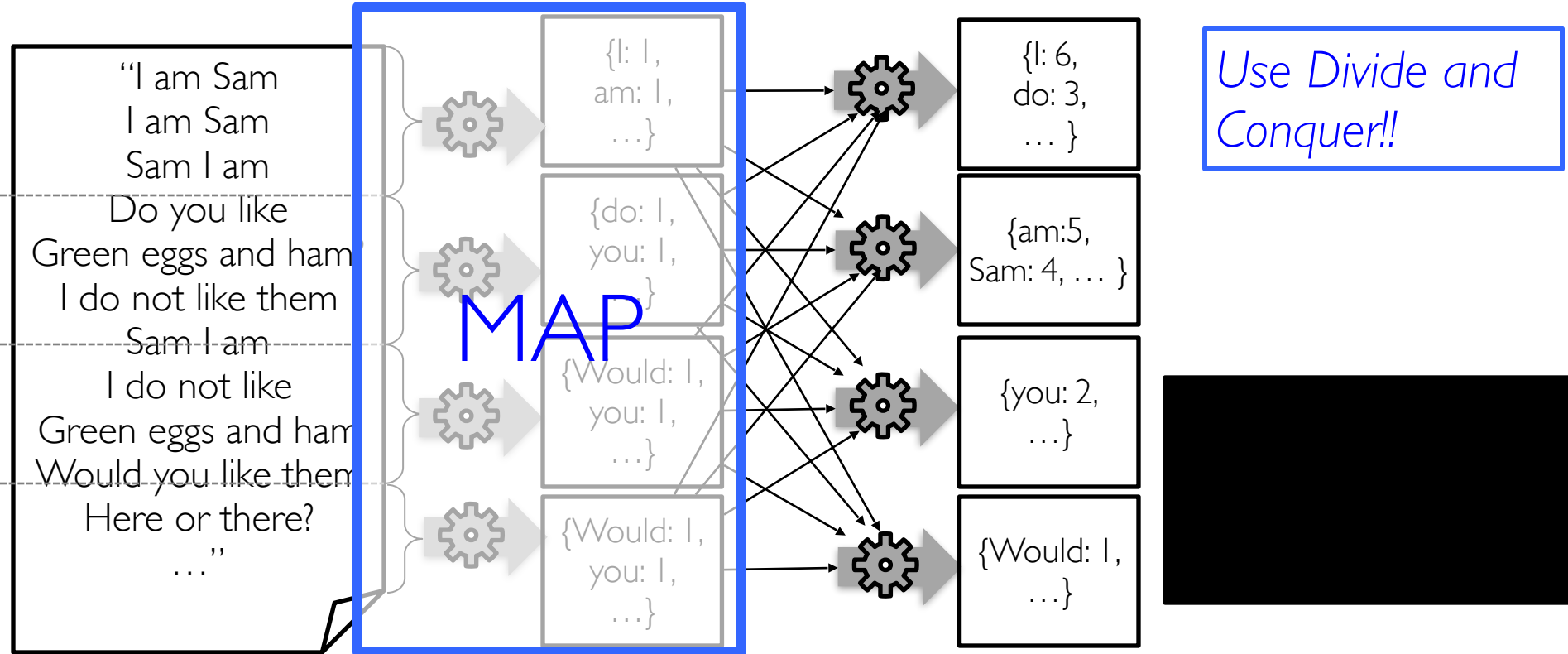




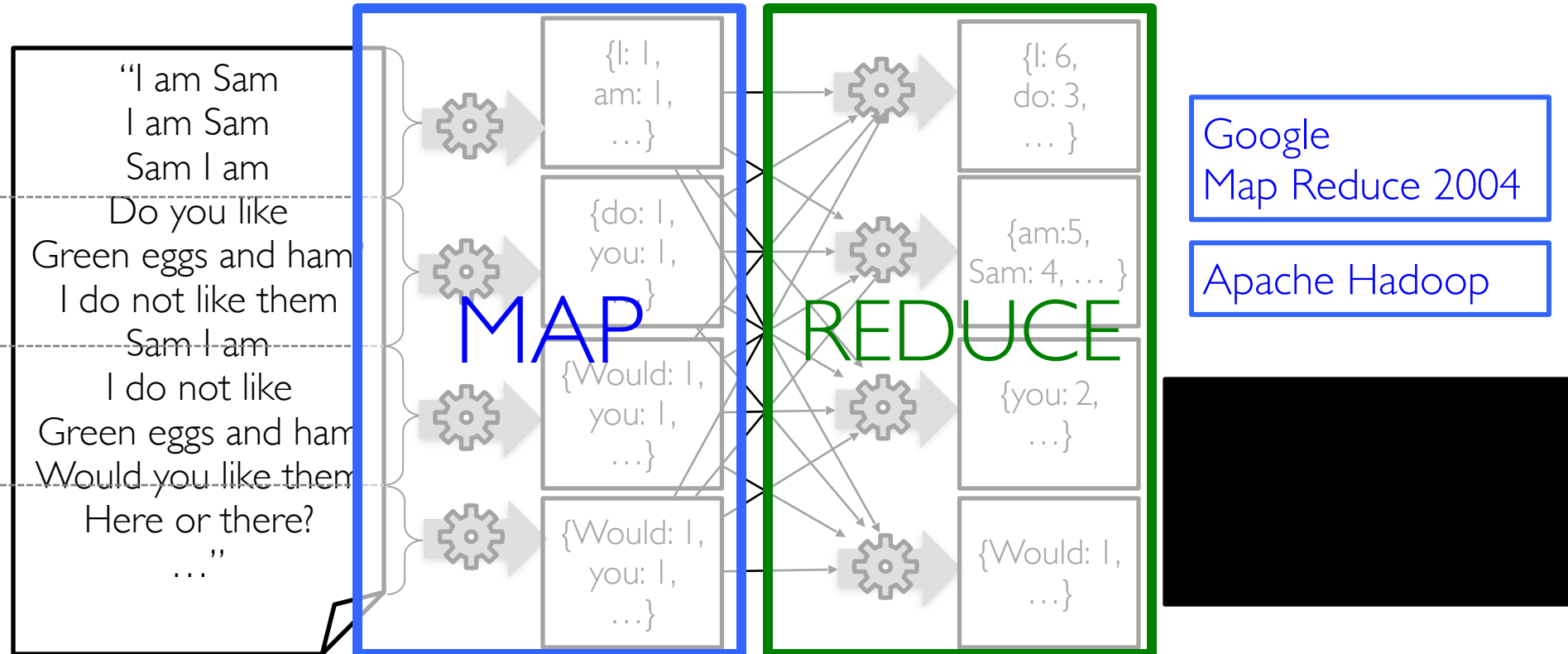
# What if the Document is Really Big?



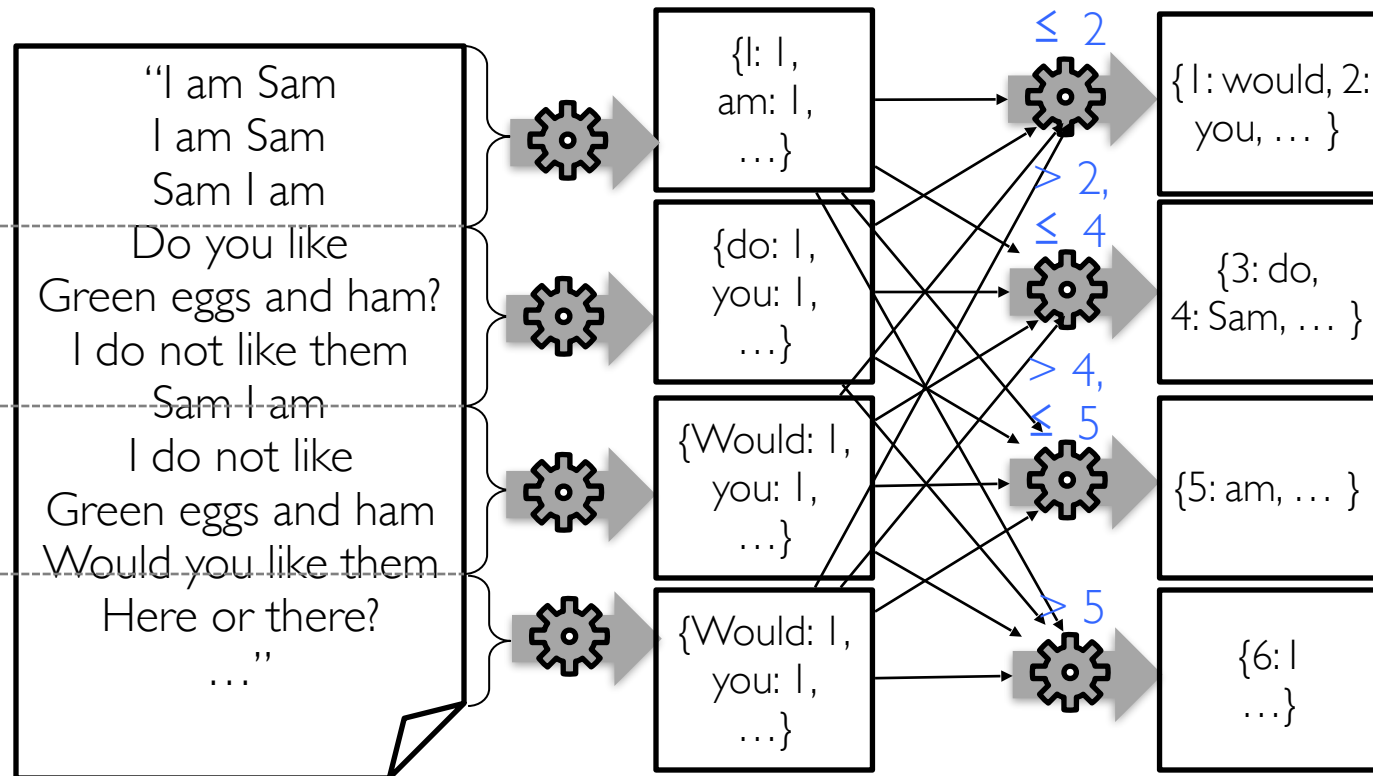
# What if the Document is Really Big?



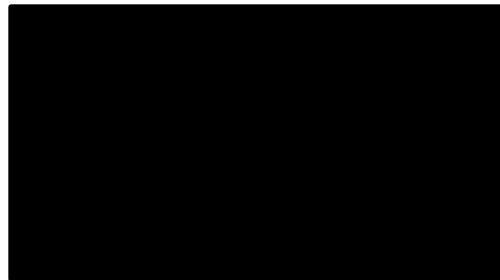
# What if the Document is Really Big?



# Map Reduce for Sorting



“What word is used most?”



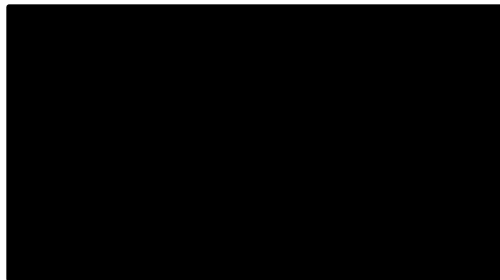
# What's Hard About Cluster Computing?

How to divide work across machines?

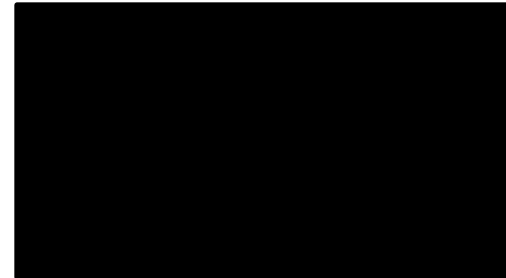
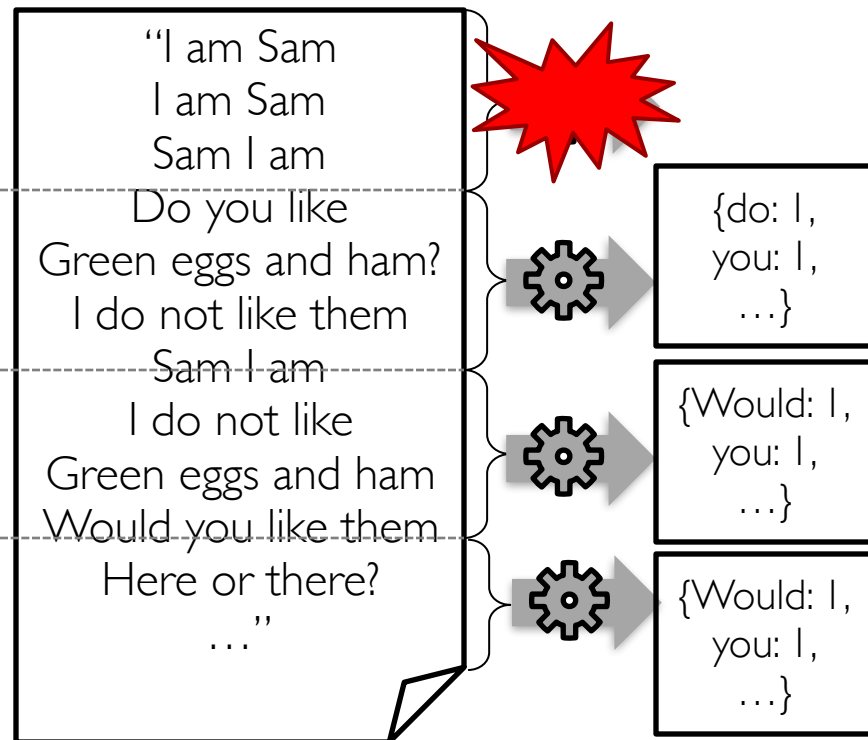
- » Must consider network, data locality
- » Moving data may be very expensive

How to deal with failures?

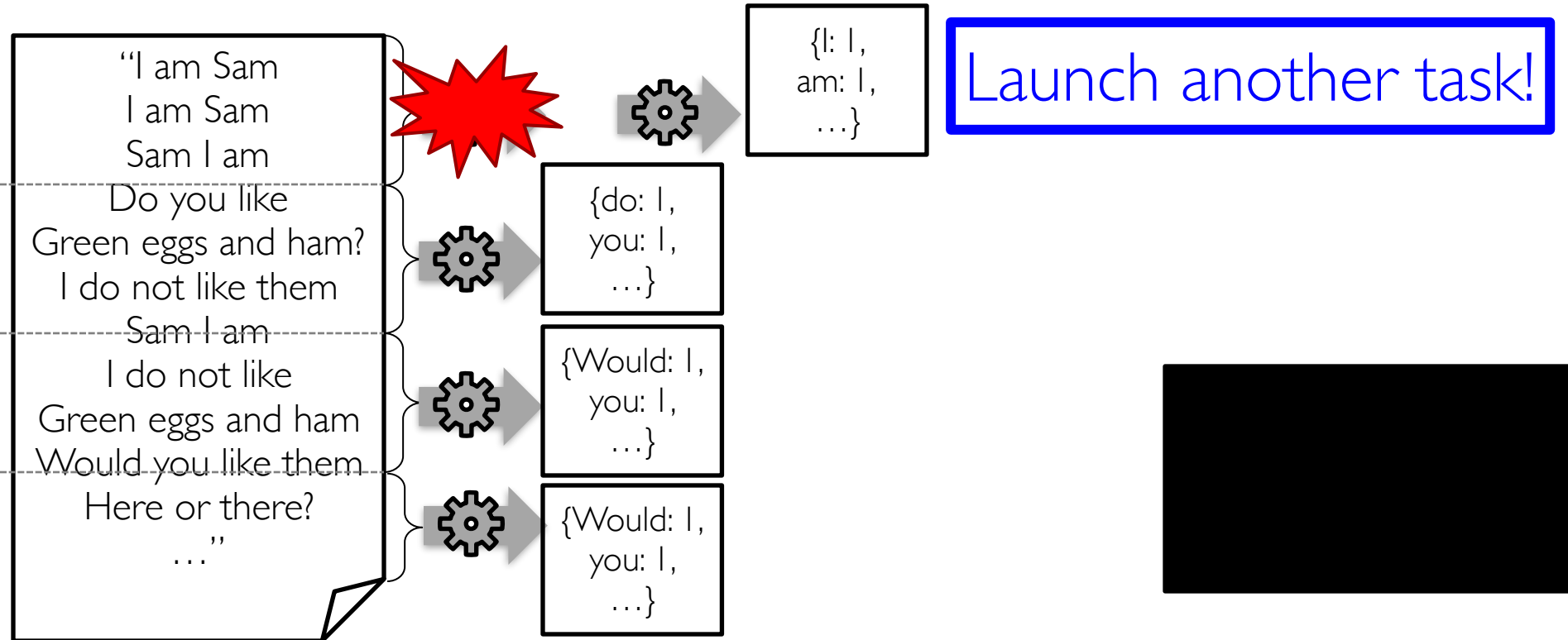
- » 1 server fails every 3 years  $\Rightarrow$  with 10,000 nodes see 10 faults/day
- » Even worse: stragglers (not failed, but slow nodes)



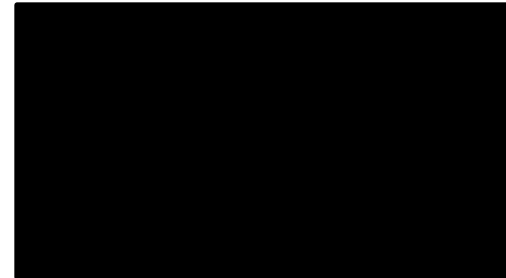
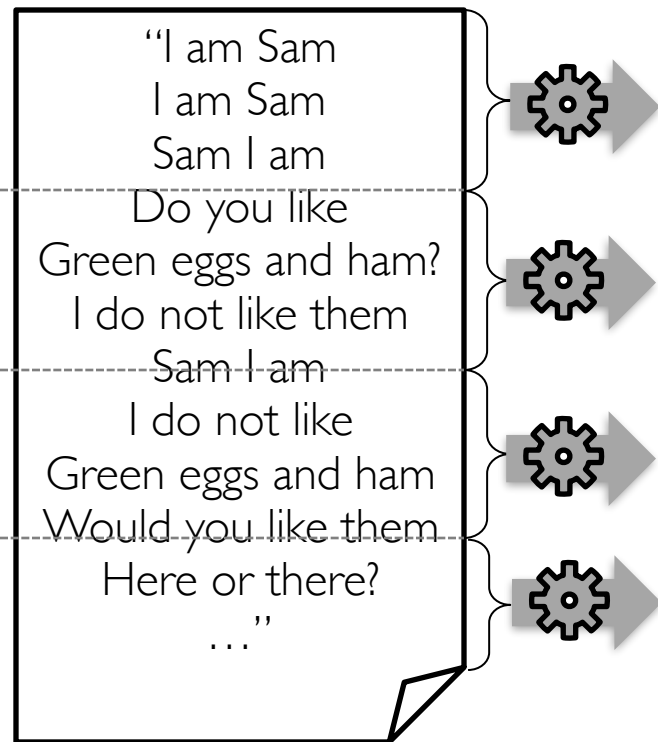
# How Do We Deal with Failures?



# How Do We Deal with Machine Failures?

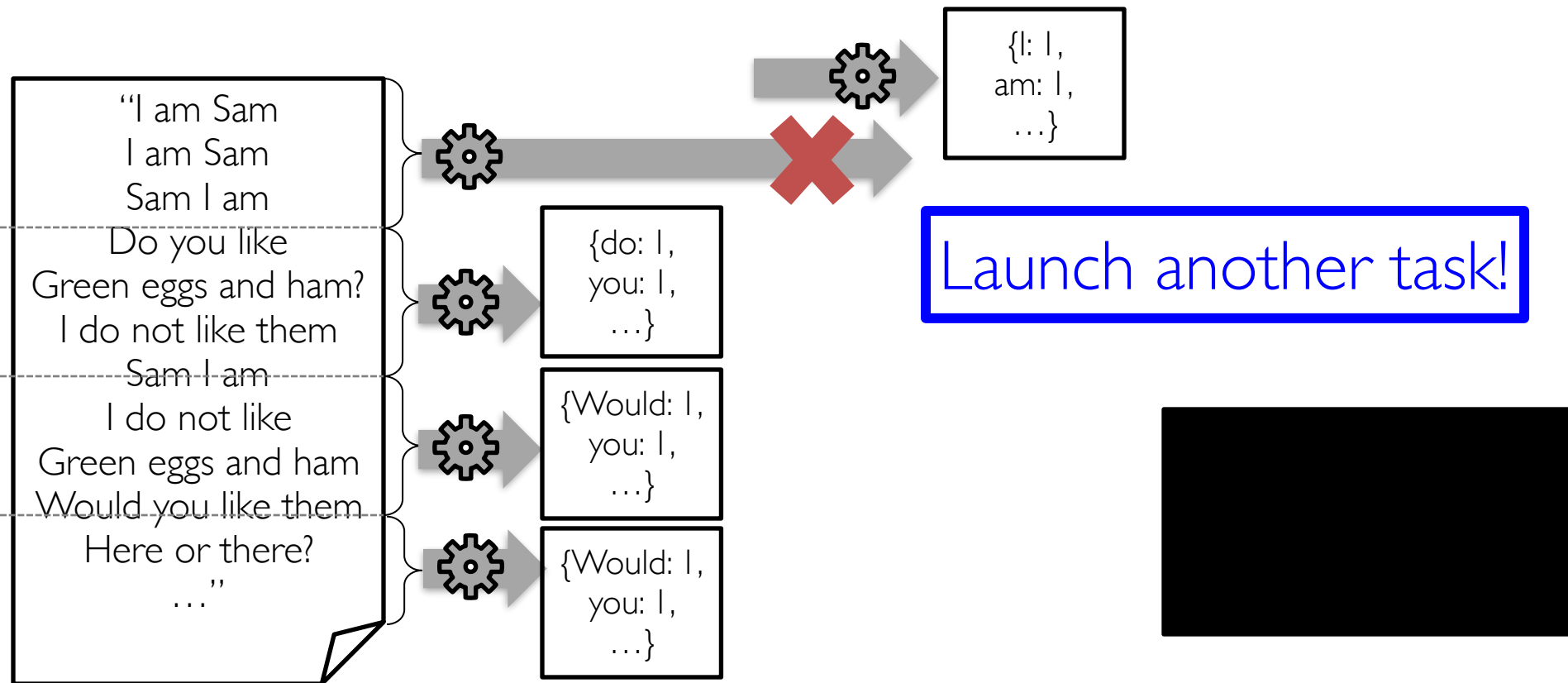


# How Do We Deal with Slow Tasks?



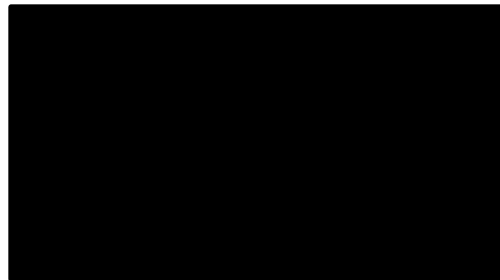


# How Do We Deal with Slow Tasks?

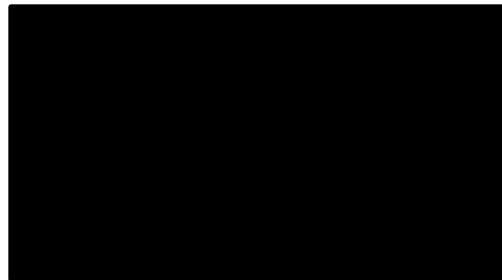
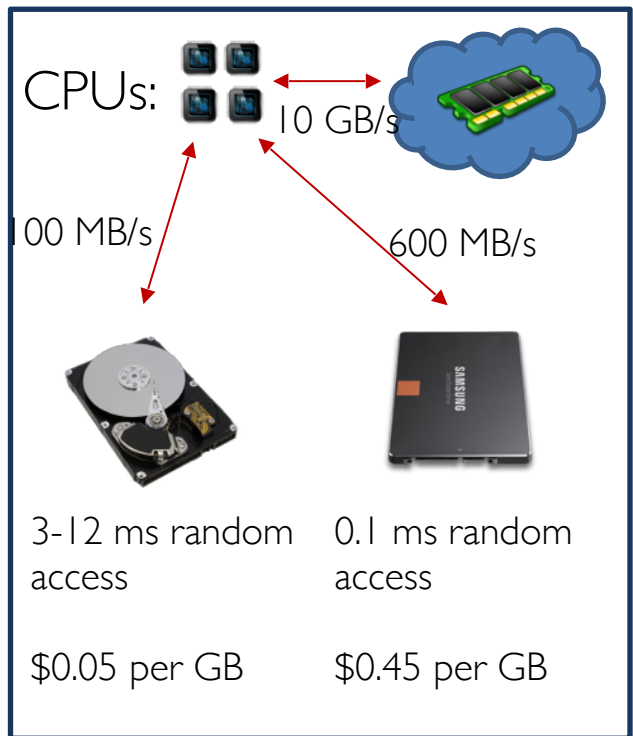


# What is Apache Spark?

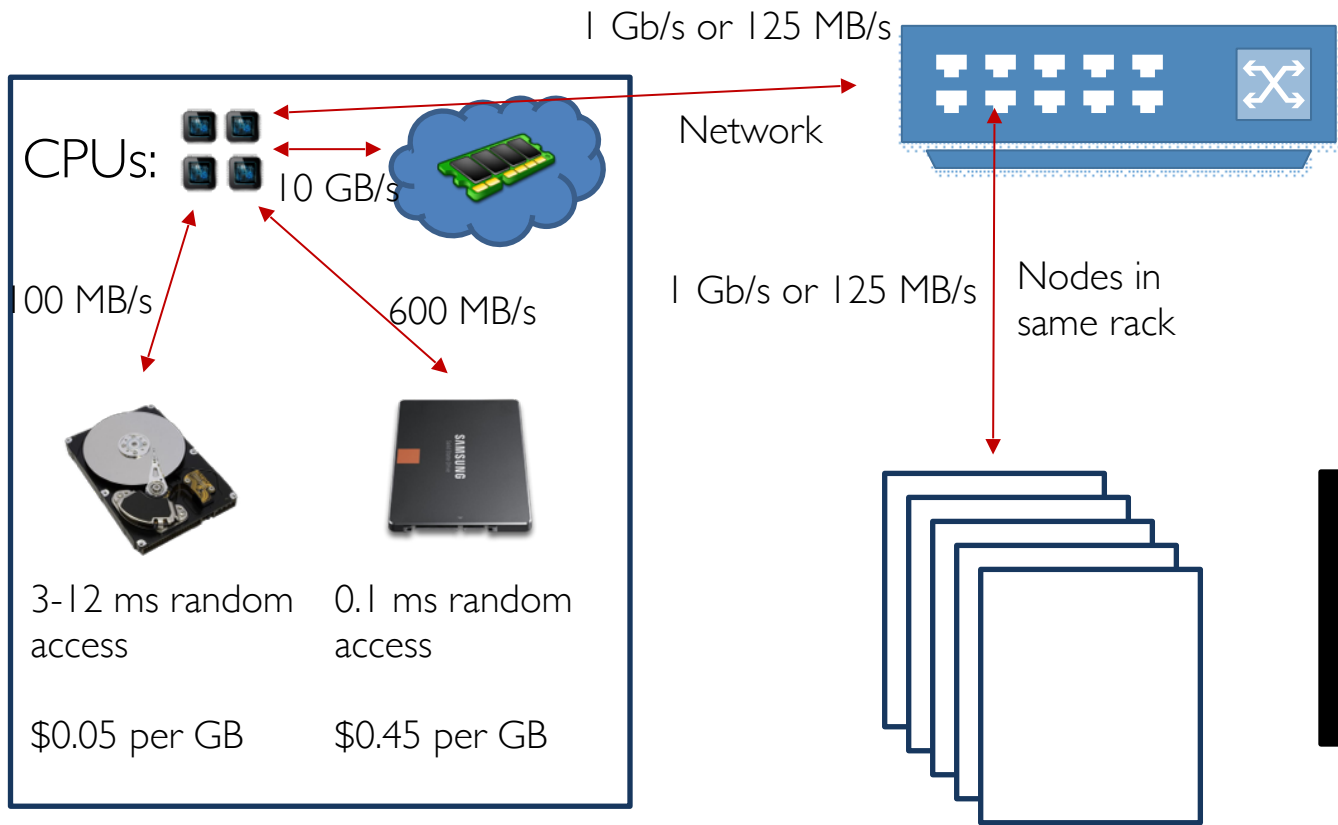
- *Scalable, efficient analysis of Big Data*



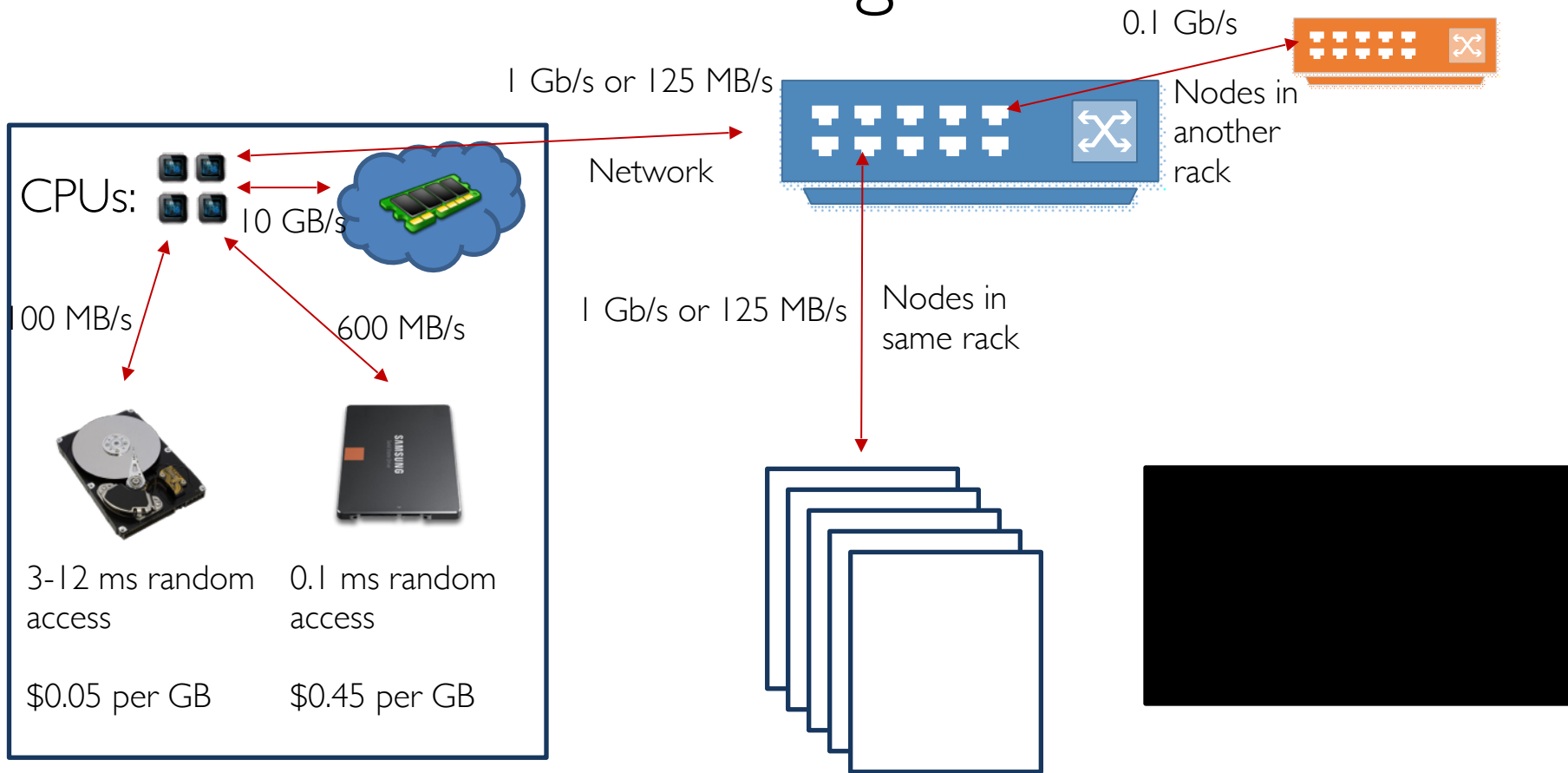
# Datacenter Organization



# Datacenter Organization



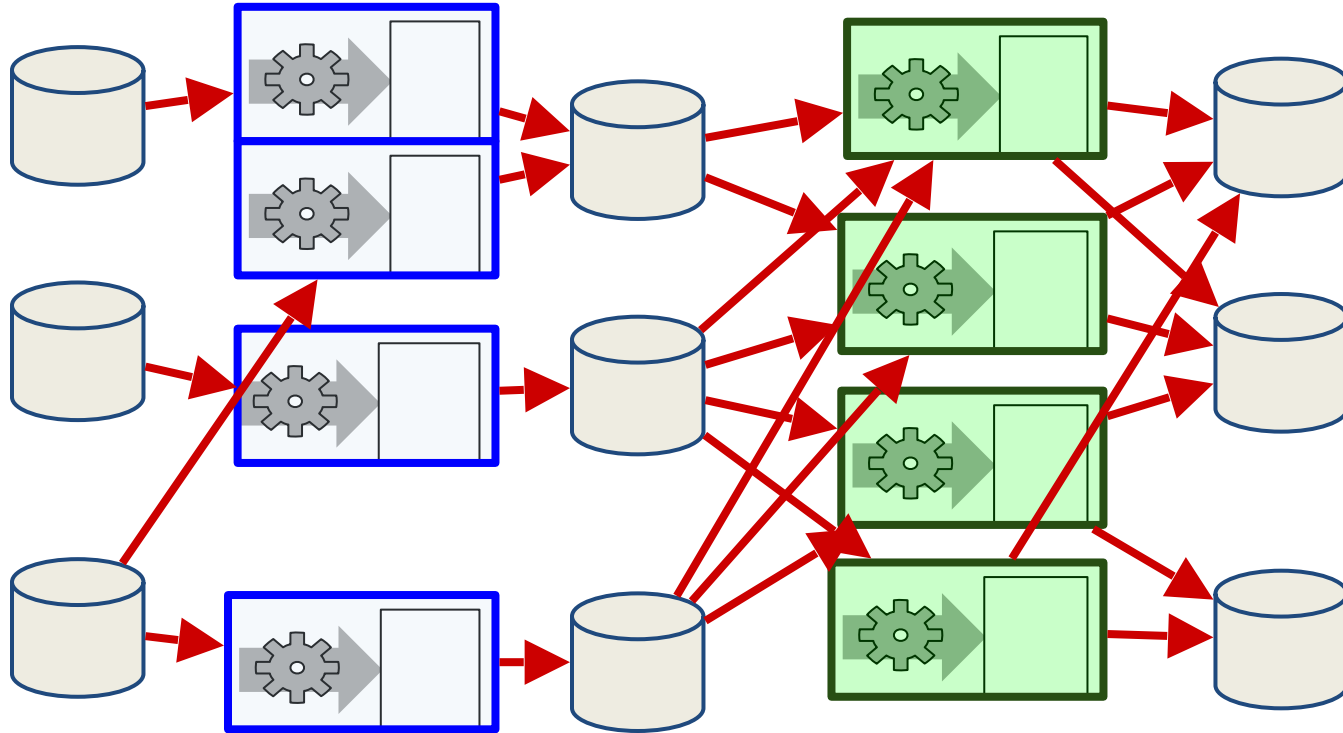
# Datacenter Organization



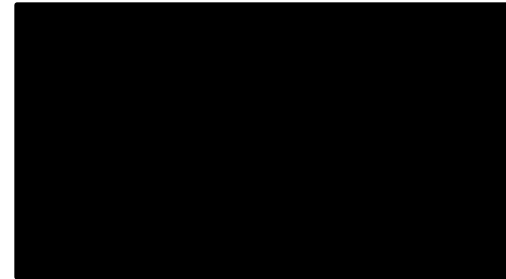
# Map Reduce: Distributed Execution

MAP

REDUCE

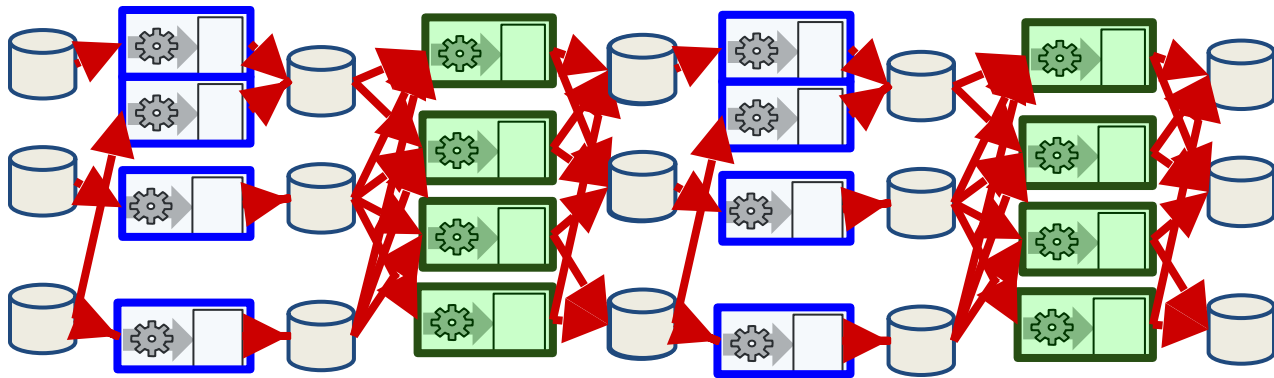


Each stage  
passes through  
the hard drives

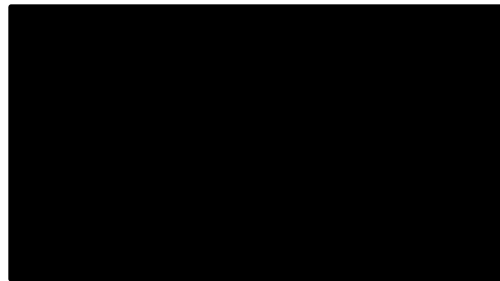
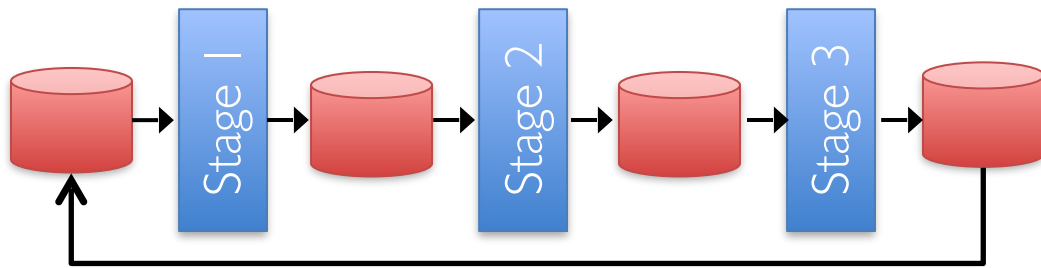


# Map Reduce: Iterative Jobs

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

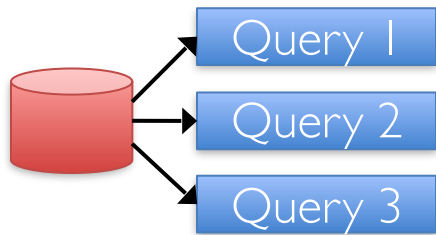


Disk I/O is  
very slow!

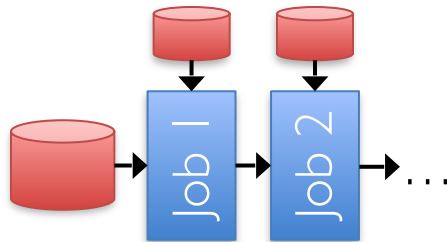


# Apache Spark Motivation

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



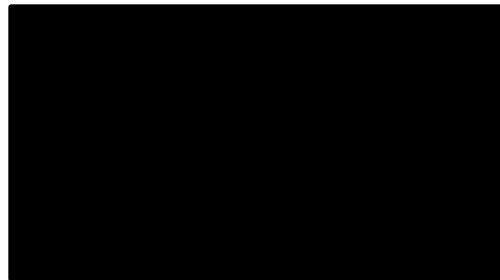
Interactive mining



Stream processing

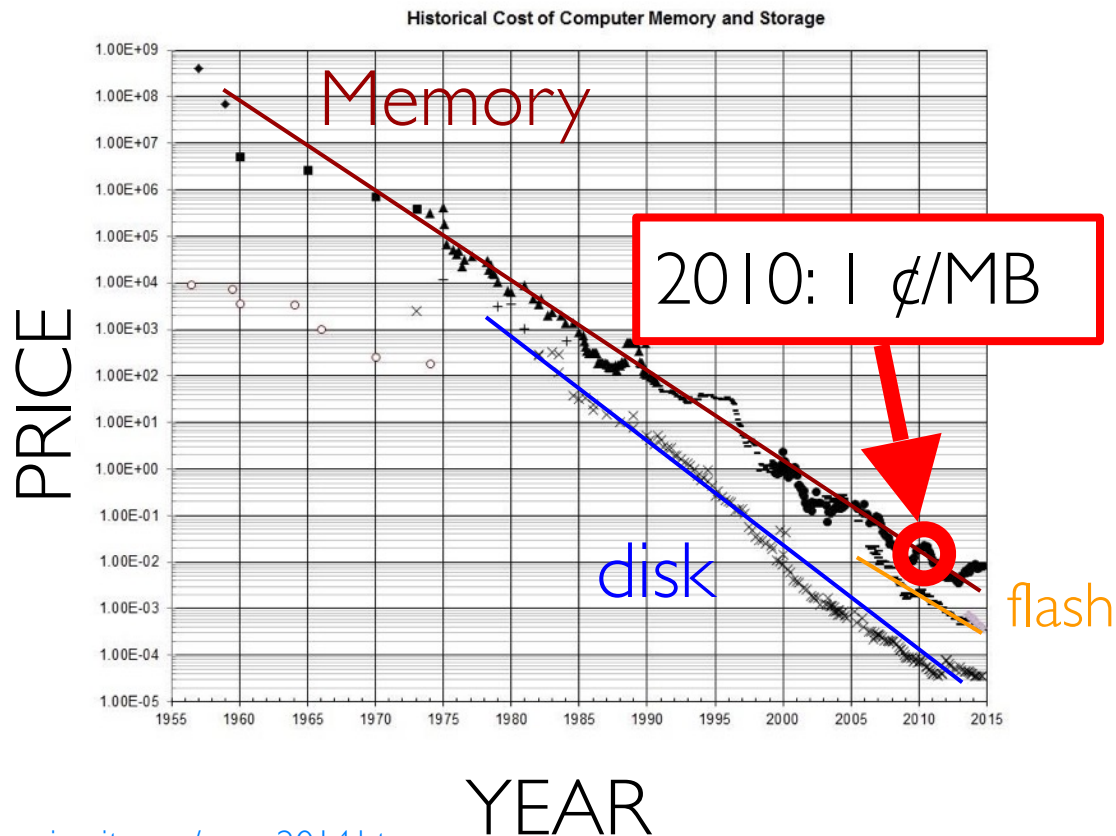
Also, iterative jobs

Disk I/O is very slow



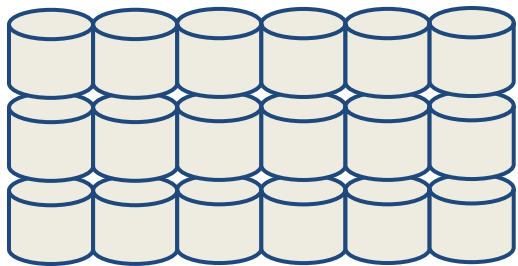


# Tech Trend: Cost of Memory



Lower cost means can  
put more memory in  
each server

# Modern Hardware for Big Data



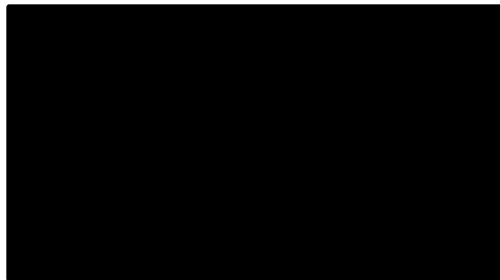
Lots of hard drives



... and CPUs

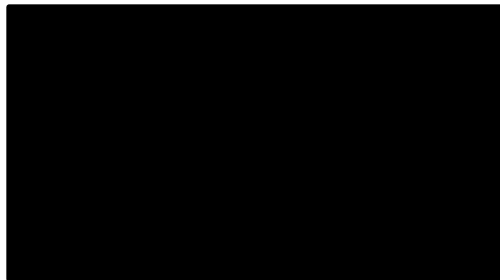


... and memory!

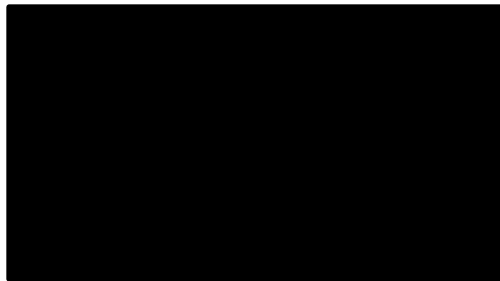
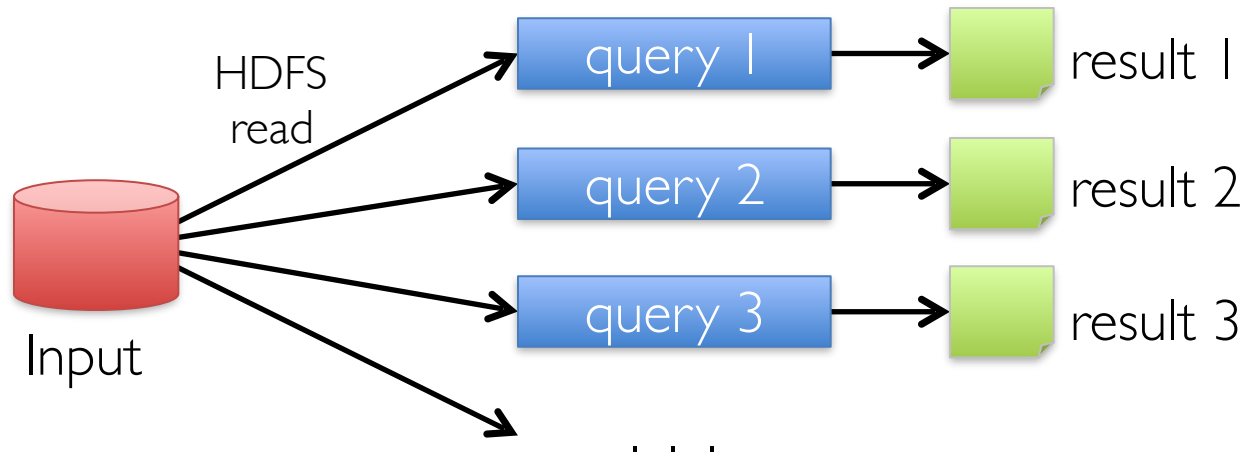
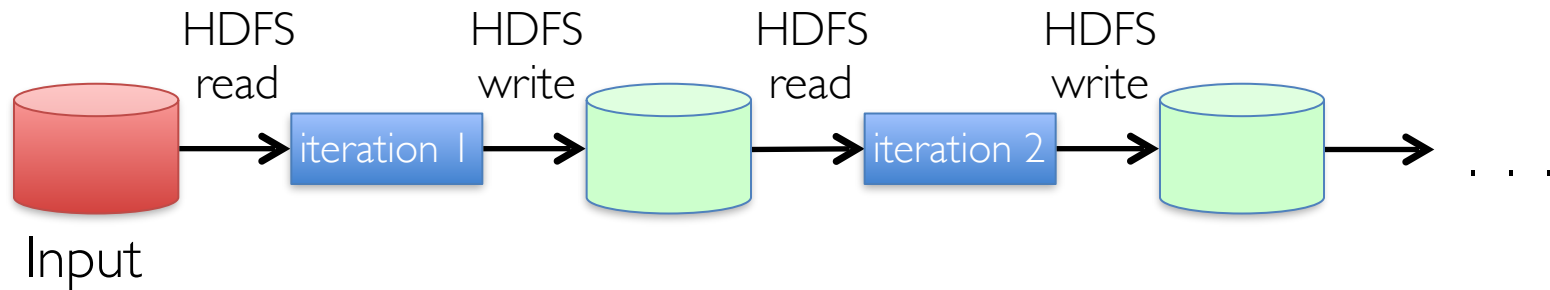


# Opportunity

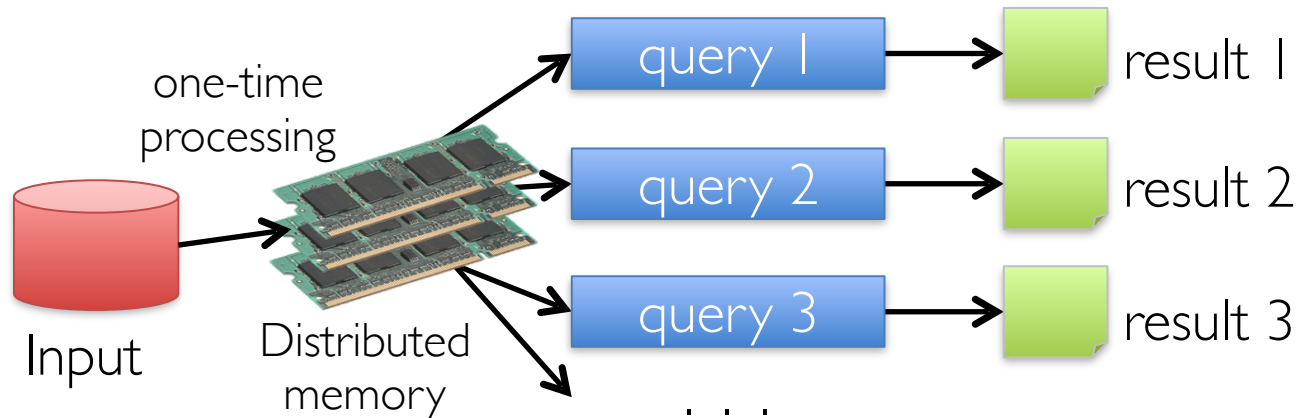
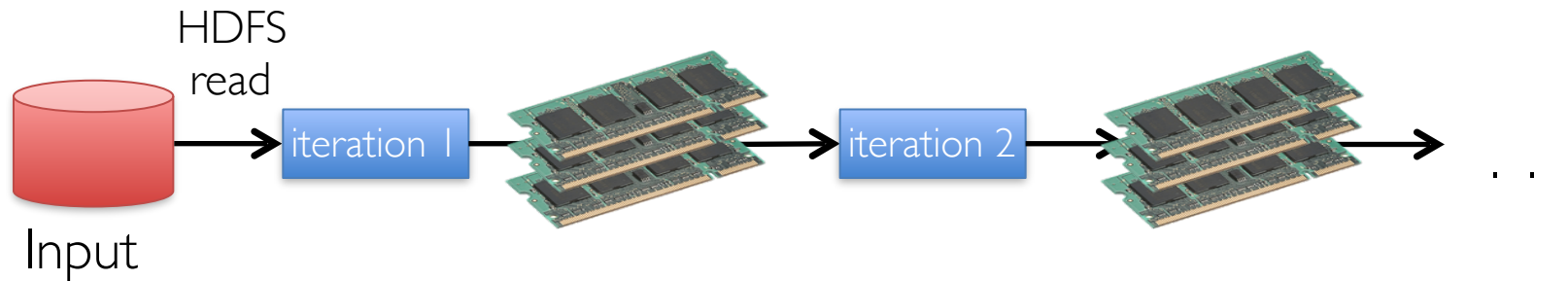
- Keep more data *in-memory*
- Create new distributed execution engine:



# Use Memory Instead of Disk



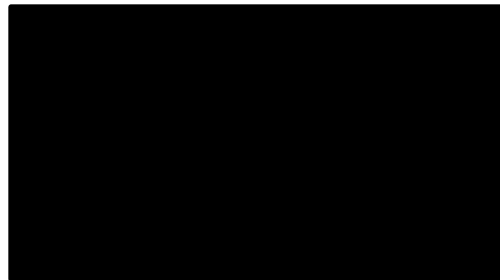
# In-Memory Data Sharing



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

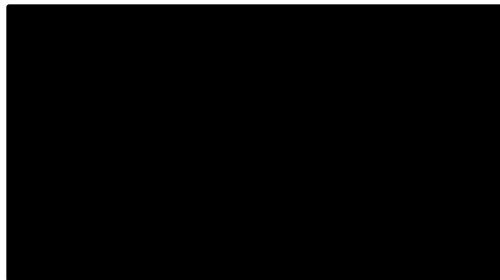
⇒ require 2-5x less code

Lazy evaluation of the lineage graph

⇒ can optimize, reduce wait states, pipeline better

Lower overhead for starting jobs

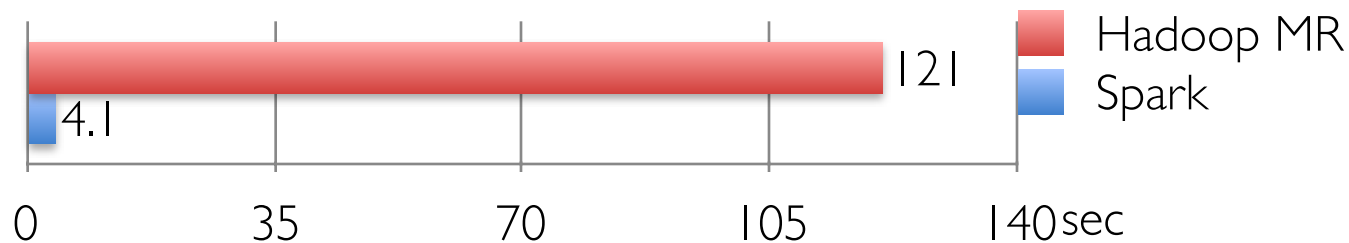
Less expensive shuffles



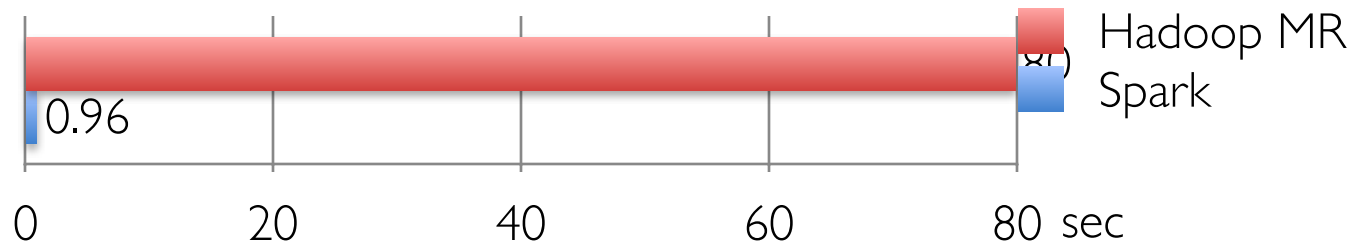
# In-Memory Can Make a Big Difference

(2013) Two iterative Machine Learning algorithms:

K-means Clustering



Logistic Regression

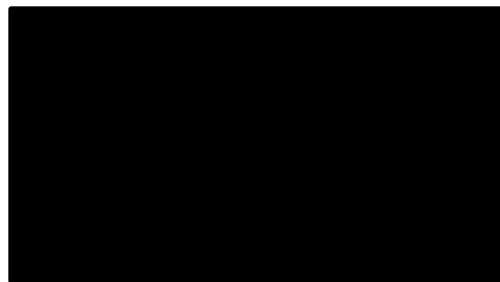




# First Public Cloud Petabyte Sort (2014)

	<b>Hadoop MR Record</b>	<b>Spark Record</b>	<b>Spark 1 PB</b>
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
<b>Sort rate</b>	<b>1.42 TB/min</b>	<b>4.27 TB/min</b>	<b>4.27 TB/min</b>
<b>Sort rate/node</b>	<b>0.67 GB/min</b>	<b>20.7 GB/min</b>	<b>22.5 GB/min</b>

[Daytona Gray 100TB](#)  
sort benchmark record  
(tied for 1<sup>st</sup> place)



# Recent Spark Performance Optimizations

Spark has added two key performance optimizations

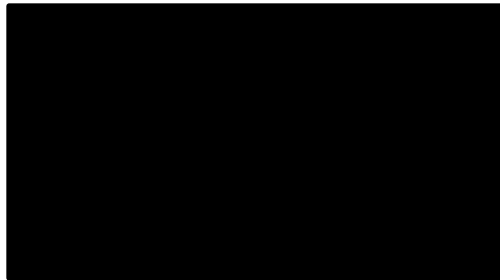
- » In addition to using memory instead of disk

Catalyst Optimization Engine

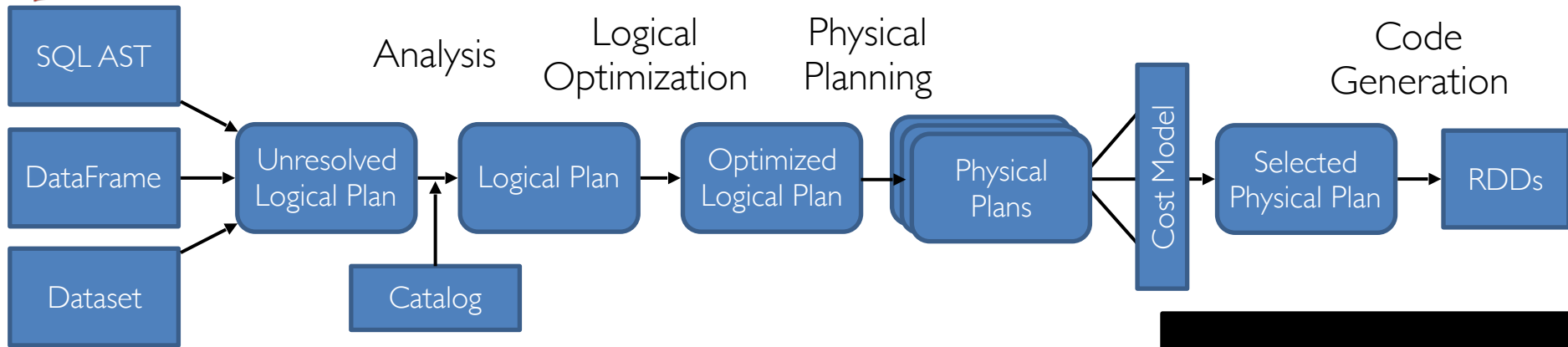
- » 75% reduction in execution time

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

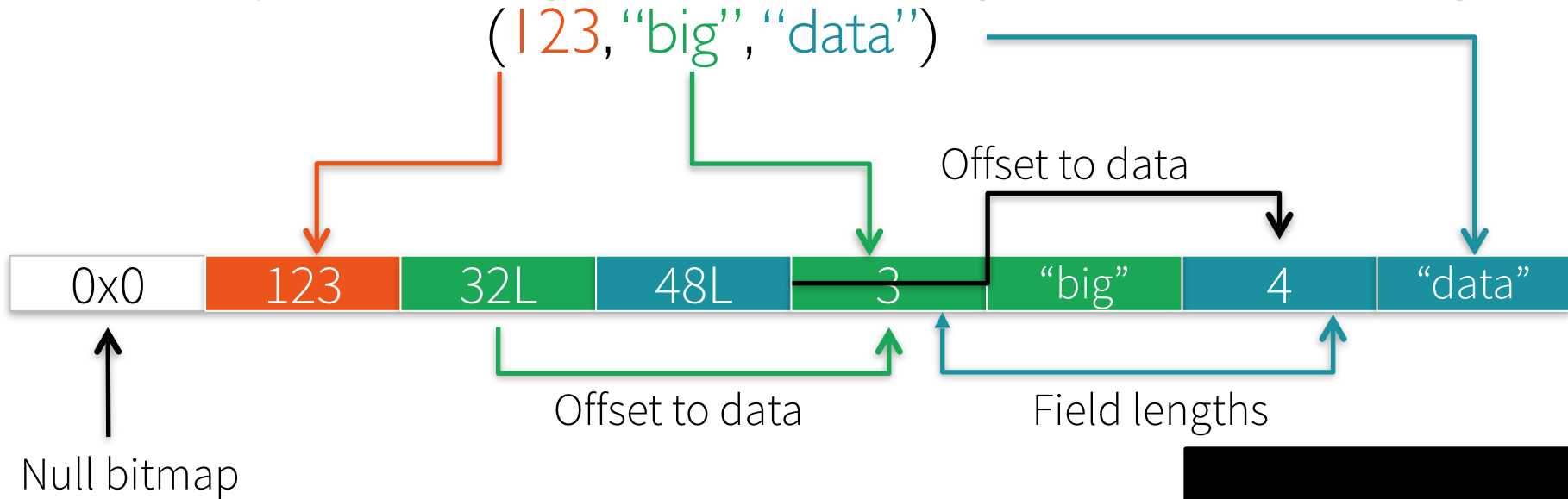
“abcd” → **Native:** 4 bytes with UTF-8 encoding  
**Java:** 48 bytes

java.lang.String object internals:

OFFSET	SIZE	TYPE	DESCRIPTION	VALUE	
0	4		(object header)	...	} 12 byte object header
4	4		(object header)	...	
8	4		(object header)	...	
12	4	char[]	String.value	□	} 20 bytes data + overhead
16	4	int	String.hash	0	
20	4	int	String.hash32	0	} 8 byte hashCode

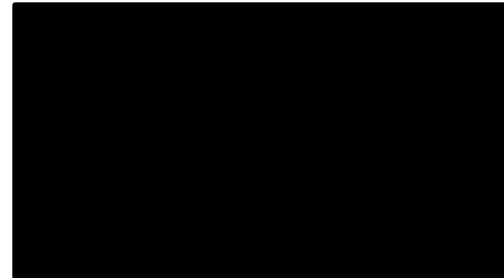
Instance size: 24 bytes (reported by Instrumentation API)

# Project Tungsten's Compact Encoding



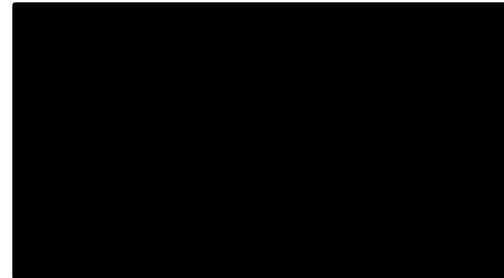
# Review: Key Data Management Concepts

- 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

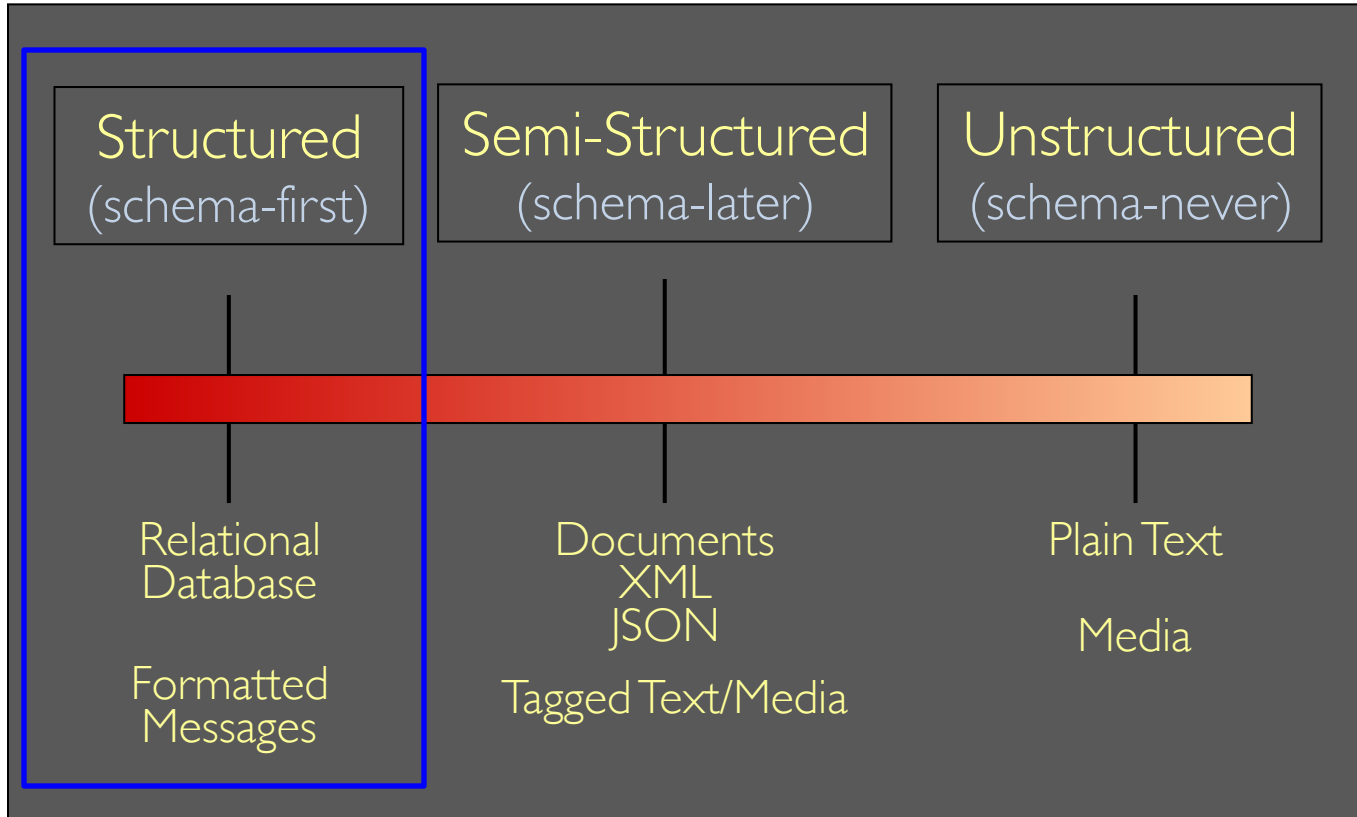


# Review: Key Data Management Concepts

- 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



This lecture



# Relational Database: Definitions

- *Relational database*: a set of *relations*

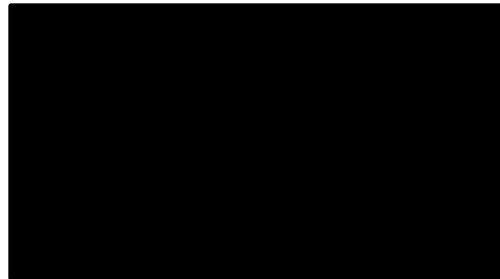
- Two parts to a *Relation*:

*Schema*: specifies name of relation, plus each column's name and type

**Students(*sid*: string, *name*: string, *email*: string,  
*age*: integer, *gpa*: real)**

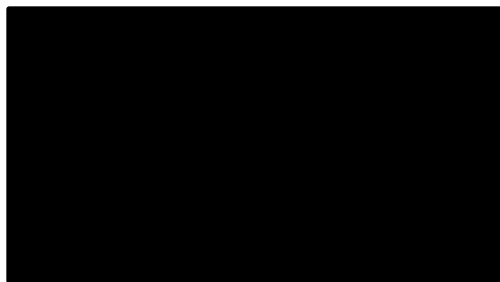
*Instance*: the actual data at a given time

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



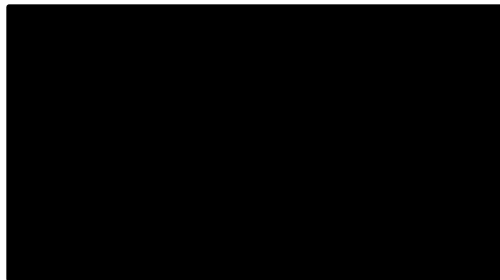
# What is a Database?

- 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](#)

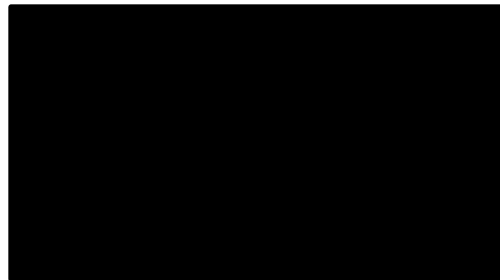
Accuracy, Consistency, Durability

- eBay database: [1.4 Petabytes](#)

- Yahoo click data: [2 Petabytes](#)

Availability  
Timeliness

- *What matters for these databases?*



# Example: Instance of Students

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

sid	name	login	age	gpa
53666	Jones	jones@eecs	18	3.4
53688	Smith	smith@statistics	18	3.2
53650	Smith	smith@math	19	3.8

*Table name*

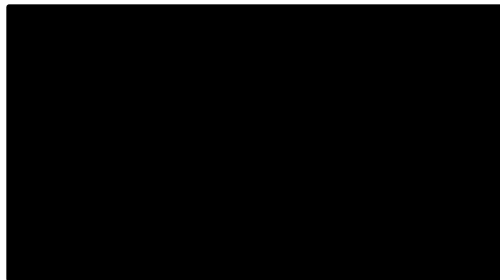
*Attribute names*

- Cardinality = 3 (rows)

*Tuples or rows*

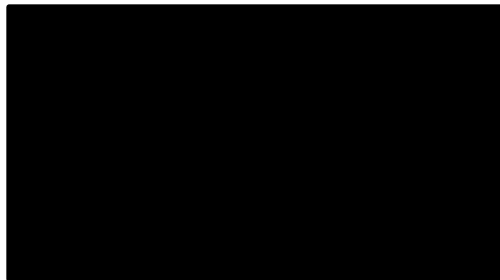
5 (columns)

- 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*



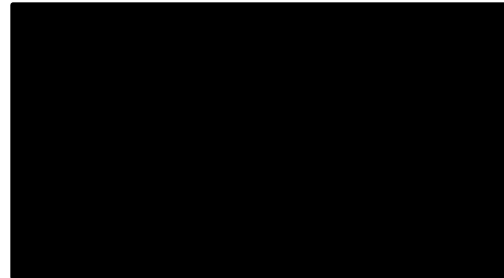
# Queries in SQL

- 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

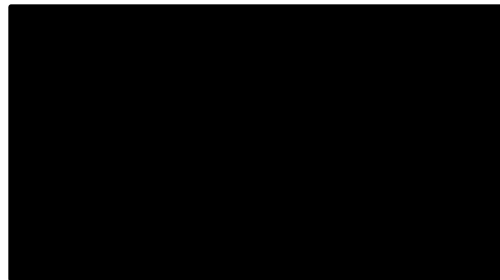
E	E.sid	E.cid	E.grade
	53831	Physics203	A
	53650	Topology112	A
•	53341	History105	B

S

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

Students

s, S and E





# Cross Join

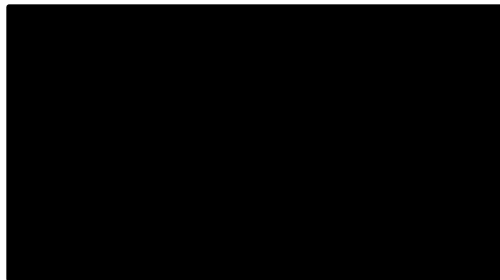
- Cartesian product of two tables ( $E \times S$ ):

Enrolled

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

Students

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



# Cross Join

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

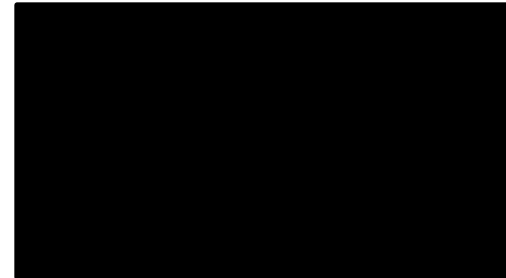
Enrolled

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

Students

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

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

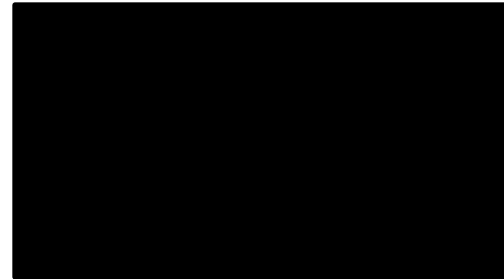


# Where Clause

- 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

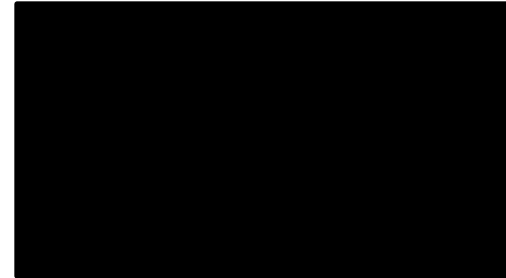


# Select Clause

- 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

Students

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

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**

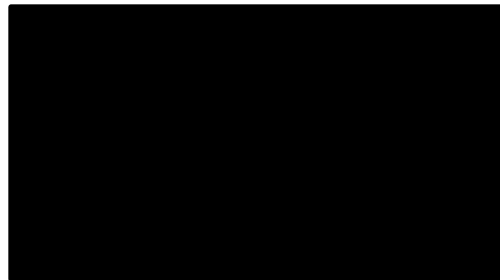
S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

**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

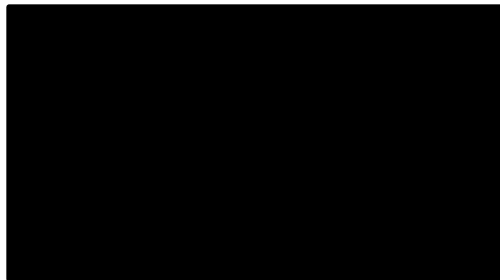
- Explicit Join notation (preferred):

```
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

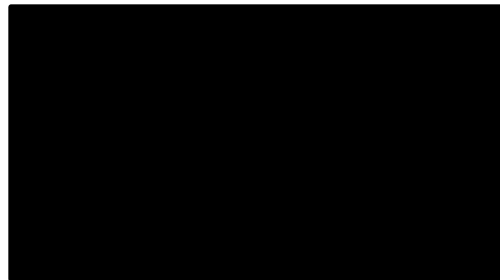
**E**

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

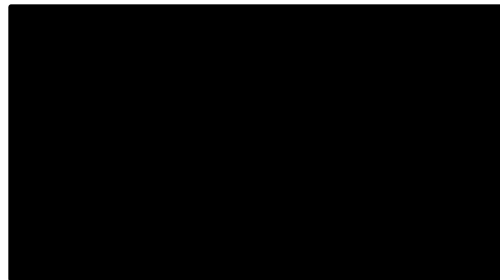
**E**

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

**Result**

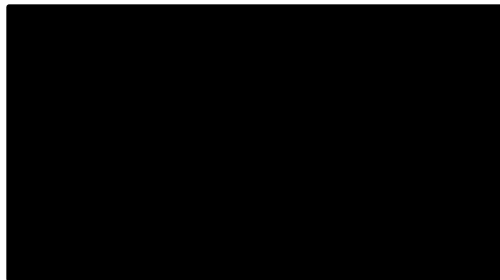
S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
<NULL>	English10

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

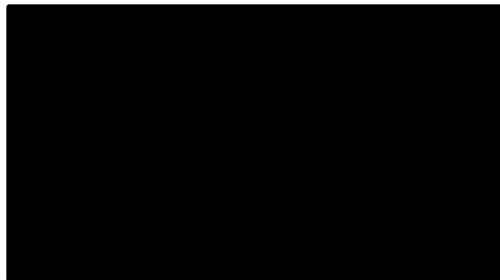


# Spark Join Examples(I)

```
>>> 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**

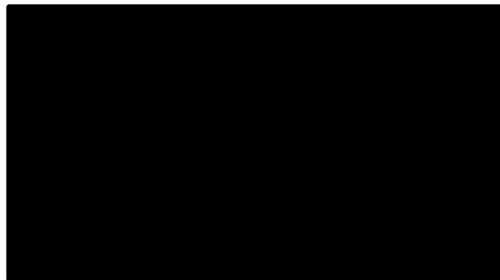


# Spark Join Examples(II)

```
>>> 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')**

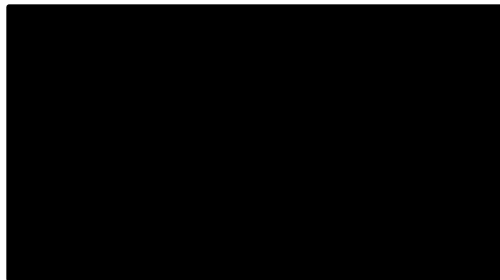
» Return DF of rows with matching **cols** in either **X** and **Y**

# Spark Join Examples(V)

```
>>> 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**



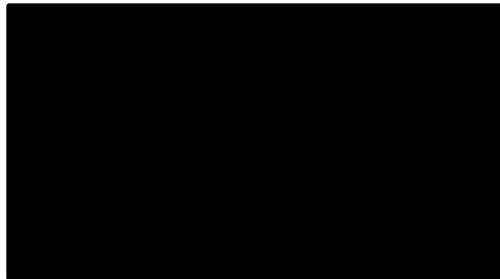


# Spark Join Examples(VI)

```
>>> 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/>



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).

API Docs ▾

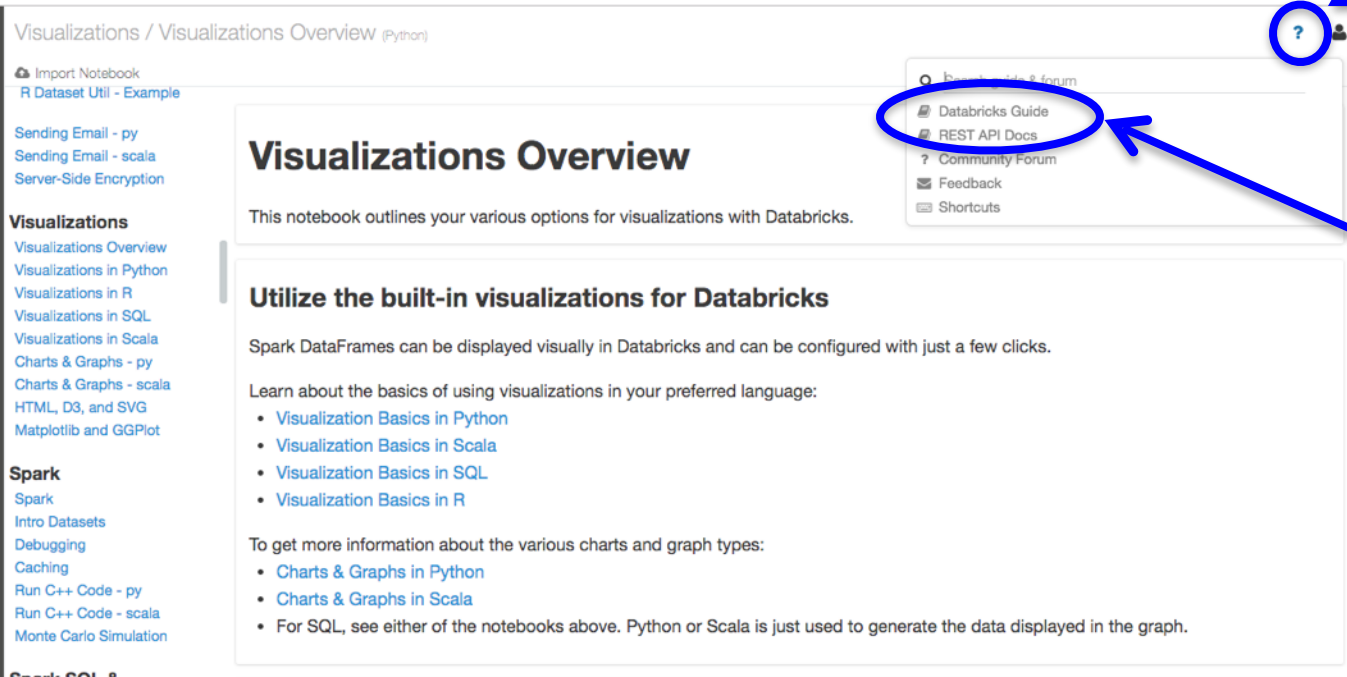
Scala

Java

Python

R

# Databricks Guide



The screenshot shows the Databricks interface. At the top, the title 'Visualizations / Visualizations Overview (Python)' is visible. On the left sidebar, there are links for 'Import Notebook', 'Dataset Util - Example', and a section for 'Visualizations' with sub-links for Overview, Python, R, SQL, Scala, and various chart types. A blue circle with a question mark is placed over the user profile icon in the top right. A blue oval highlights the 'Databricks Guide' link in the top right dropdown menu, with a blue arrow pointing to it from the right. The main content area has the heading 'Visualizations Overview' and a sub-heading 'Utilize the built-in visualizations for Databricks'. It includes a paragraph about Spark DataFrames and a list of links for learning basics in different languages (Python, Scala, SQL, R). Another list of links for more information about charts and graph types is provided at the bottom.

Visualizations / Visualizations Overview (Python)

Import Notebook  
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.

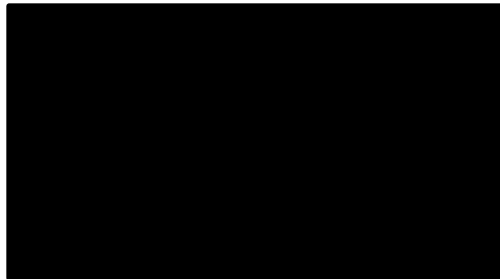
# Spark Technical Blogs

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

The screenshot shows the Apache Spark YouTube channel page. The header features the Spark logo and a search bar with 'apache spark' entered. Below the header, the channel name 'Apache Spark' is displayed with a 'Subscribe' button and '6,664' subscribers. The main content area shows a video titled 'The Spark Kernel - Chip Senkbeil IBM - Meet Up Talk' with a duration of 1:10:56. The right sidebar lists related channels including Databricks, The Spark Spot, HadoopSummit, Hakka Labs, Typesafe, and MapR Technologies.

Check out the [Apache Spark](https://www.youtube.com/user/TheApacheSpark) YouTube Channel!



# Community

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

Community | Apache Spark

<https://spark.apache.org/community.html#mailing-lists>

**Spark** *Lightning-fast cluster computing*

Download Libraries Documentation Examples Community FAQ

## Spark Community

### Mailing Lists

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

- [user@spark.apache.org](mailto:user@spark.apache.org) is for usage questions, help, and announcements. ([subscribe](#)) ([unsubscribe](#)) ([archives](#))
- [dev@spark.apache.org](mailto: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.

#### 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](#)

[Download Spark](#)

# 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.

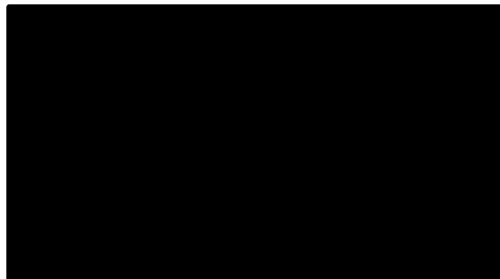
186,279  
members

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 website. At the top, there's a navigation bar with the Databricks logo, a search bar, and links to 'Ask a question' and 'Sign in'. Below the navigation bar, the main content area is divided into two columns. The left column lists several questions with their respective tags, number of replies, and likes. The right column features a 'Popular Topics' section with a grid of topic tags. A 'View all >' link is located at the bottom right of the popular topics grid.

Find posts, topics, and users...

Ask a question Sign in

All Questions ▾

Can the REST API be configured to use a different port number?  
rest-api · rest  
1 Reply 0 Likes

Source Control and Debugging options  
debug · debugging · git · source-control  
0 Replies 0 Likes

Why do I get 'java.io.IOException: File already exists' for saveAsTable with Overwrite mode?  
hive · dataframe · saveasstable  
0 Replies 0 Likes

What s3 bucket does DBFS use? And where is the local cache files?  
dbfs · dbutils  
0 Replies 0 Likes

bug in notebook - type mismatch error  
notebook · case class · type mismatch  
0 Replies 0 Likes

How do I set up VPC Peering to connect my servers/databases to my Databricks Spark Clusters?  
aws · vpc  
0 Replies 0 Likes

Read HDFS file from python script  
python · hadoop  
0 Replies 0 Likes

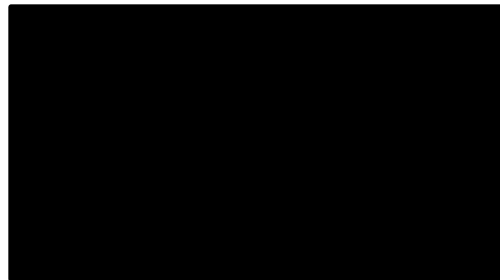
Popular Topics

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 mllib cluster scala  
cluster provisioning hive dbfs jobs parquet files sql  
spark streaming visualizations streaming  
View all >

Community forum for  
Databricks users

Mostly Databricks-specific Q&A

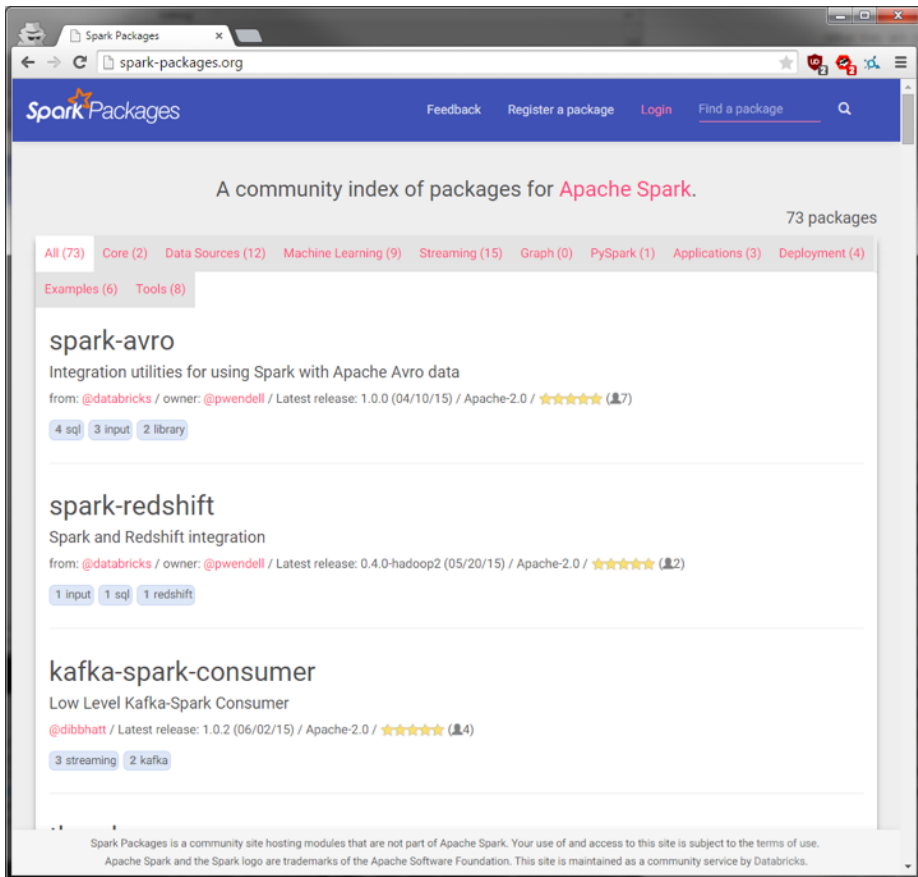
Some general Spark Q&A





# Spark Packages

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



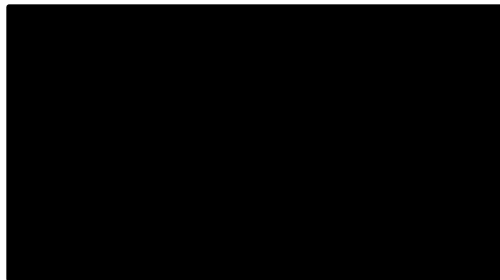
The screenshot shows the Spark Packages website interface. At the top, there's a navigation bar with links for Feedback, Register a package, Login, and Find a package. Below the navigation bar, a header states "A community index of packages for Apache Spark." and indicates "73 packages". A filter bar shows categories like All (73), Core (2), Data Sources (12), Machine Learning (9), Streaming (15), Graph (0), PySpark (1), Applications (3), and Deployment (4). Below this, there are tabs for Examples (6) and Tools (8). The main content area lists three packages: spark-avro, spark-redshift, and kafka-spark-consumer. Each package entry includes its description, source/owner information, latest release date, version, and a star rating with the number of votes. For example, spark-avro has a 4.7 rating (17 votes) and includes tags for sql, input, and library. The footer contains a disclaimer about the site's purpose and its relationship to Apache Spark.

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.

232 software packages for Spark

» User-provided Spark extensions

» Community votes ★★★★★ (8)



# Spark Source Code

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

Hint: For detailed explanations, check out comments in code

The screenshot shows the GitHub repository for Apache Spark. At the top, the repository name 'apache / spark' is displayed, along with statistics: 12,036 commits, 12 branches, 36 releases, and 611 contributors. Below this, a table lists recent commits, including their commit IDs, titles, and the time since they were made. The right sidebar contains links to 'Code', 'Pull requests', 'Pulse', and 'Graphs', as well as options to clone the repository or download the source code as a ZIP file.

Commit ID	Commit Title	Time Ago
[SPARK-5989] [MLLIB] Model save/load for LDA		38 minutes ago
[SPARK-9201] [ML] Initial integration of MLlib + SparkR using RFormula		14 hours ago
[SPARK-7801] [BUILD] Updating versions to SPARK 1.5.0		2 months ago
[SPARK-7801] [BUILD] Updating versions to SPARK 1.5.0		2 months ago
[SPARK-7733] [CORE] [BUILD] Update build, code to use Java 7 for 1.5.0+		a month ago
[SPARK-8933] [BUILD] Provide a --force flag to build/mvn that always ...		7 days ago
[SPARK-3071] Increase default driver memory		20 days ago
[SPARK-5423] [CORE] Register a TaskCompletionListener to make sure re...		an hour ago
[MLLIB] [DOC] Seed fix in mllib naive bayes example		3 days ago
[SPARK-8401] [BUILD] Scala version switching build enhancements		8 hours ago
[SPARK-8954] [BUILD] Remove unneeded deb repository from Dockerfile t...		8 days ago
[SPARK-8598] [MLLIB] Model save/load for LDA		38 minutes ago
[SPARK-8598] Add module for rstudio link to spark		8 days ago
[SPARK-7977] [BUILD] Disallowing println		11 days ago
[SPARK-8962] Add Scalastyle rule to ban direct use of Class.forName; ...		7 days ago



# Research Papers

## Spark: Cluster Computing with Working Sets

### 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 provide locality-aware scheduling, fault tolerance, and load balancing. MapReduce [11] pioneered this model, while systems like Dryad [17] and Map-Reduce-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 interfaces such as Pig [21] and 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 a general shared memory abstraction, 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

[http://www.cs.berkeley.edu/~matei/papers/2010/hotcloud\\_spark.pdf](http://www.cs.berkeley.edu/~matei/papers/2010/hotcloud_spark.pdf)

June 2010



# Research Papers

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

### 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  
*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 capture 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 abstractions for leveraging distributed 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 HalLoop [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 reuse, e.g., to let a user 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 Piccolo [27], 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 transformations used to build a dataset (its *lineage*) rather than the actual data.<sup>1</sup> If a partition of an RDD is lost, the RDD has enough information about how it was derived from other RDDs to recompute

<sup>1</sup>Checkpointing 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.

[http://www.cs.berkeley.edu/~matei/papers/2012/nsdi\\_spark.pdf](http://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf)

April 2012



# SQL

## Spark SQL: Relational Data Processing in Spark

Michael Armbrust<sup>1</sup>, Reynold S. Xin<sup>1</sup>, Cheng Lian<sup>1</sup>, Yin Hai<sup>1</sup>, Davies Liu<sup>1</sup>, Joseph K. Bradley<sup>1</sup>,  
Xiangrui Meng<sup>1</sup>, Tomer Kaftan<sup>1</sup>, Michael J. Franklin<sup>1\*</sup>, Ali Ghodsi<sup>1</sup>, Matei Zaharia<sup>1\*</sup>

<sup>1</sup>Databricks Inc.    <sup>\*</sup>MIT CSAIL    <sup>†</sup>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

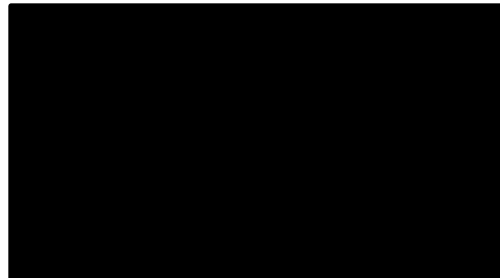
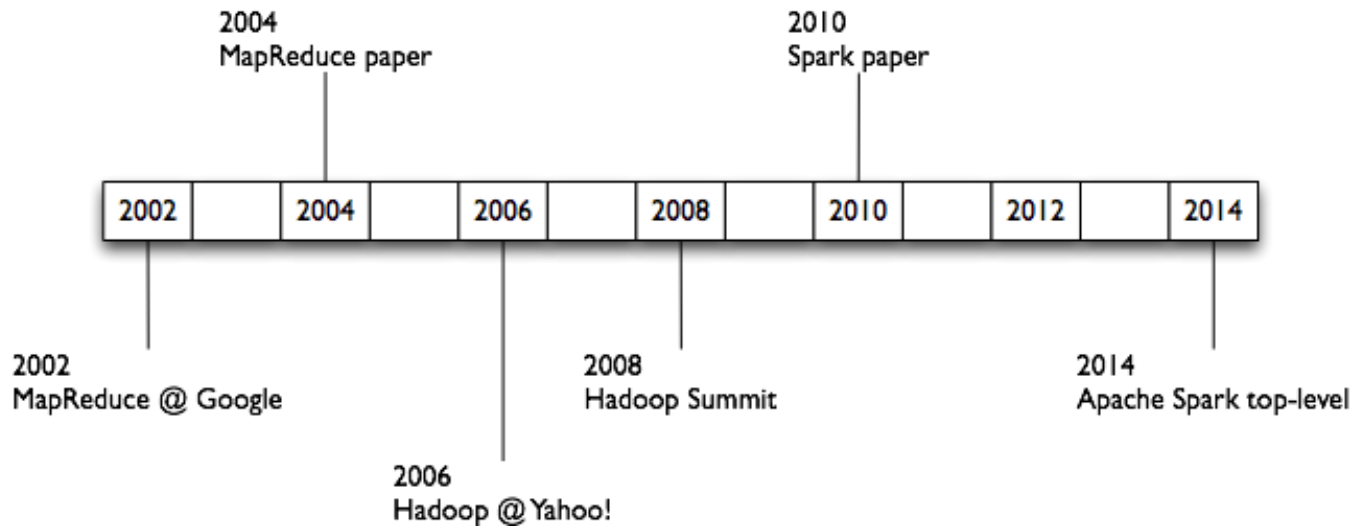
# Spark SQL: Relational Data Processing in Spark

Seemlessly mix SQL queries with Spark programs

June 2015

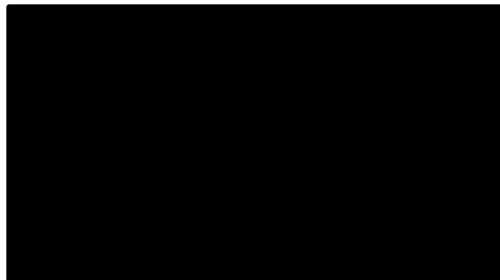
<https://amplab.cs.berkeley.edu/wp-content/uploads/2015/03/SparkSQLSigmod2015.pdf>

# 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.html>
- 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://nutch.apache.org/>
- circa 2008 – Yahoo!: web scale search indexing  
*Hadoop Summit*, HUG, etc.  
<http://hadoop.apache.org/>
- 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](http://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](http://usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf)

