CS186: Introduction to Database Systems

Eric Brewer
Fall 2015

(based on slides from Joe Hellerstein)

About the instructor…

- Berkeley undergrad
- Faculty since 1994
- A "systems" person
- Pioneered clusters, early search engines, "cloud" computing

Also VP Infrastructure at Google

Queries for Today

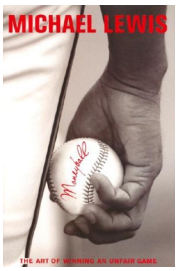
- Why?
- What?
- Who?
- How?
- For instance?

Why?

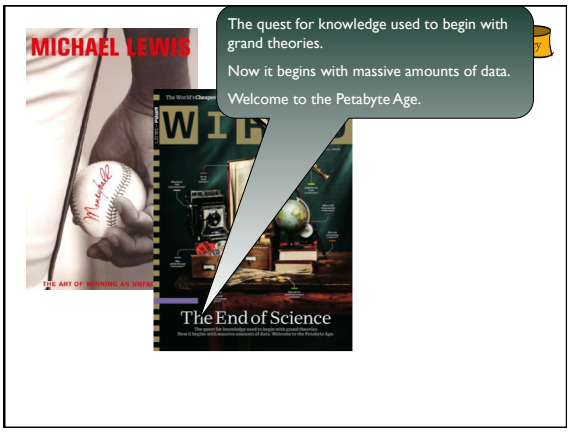
Data is at the center of many things.

Why?

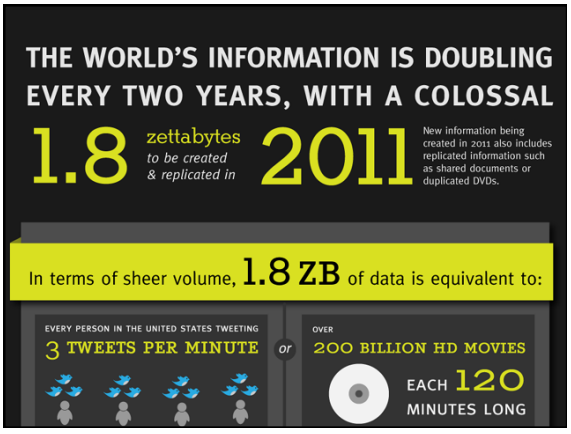
Data is at the center of ~~many things~~ *everything*.

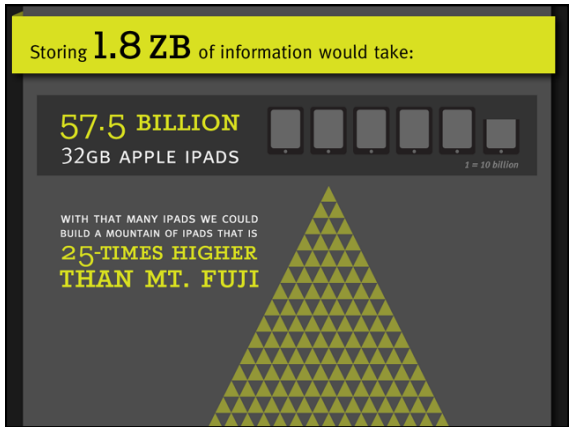


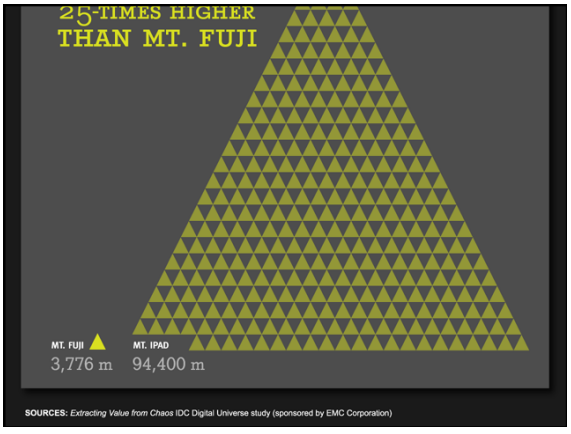

MICHAEL LEWIS
Berkeley
WIRED
The End of Science


The quest for knowledge used to begin with grand theories.
Now it begins with massive amounts of data.
Welcome to the Petabyte Age.
MICHAEL LEWIS
WIRED
The End of Science

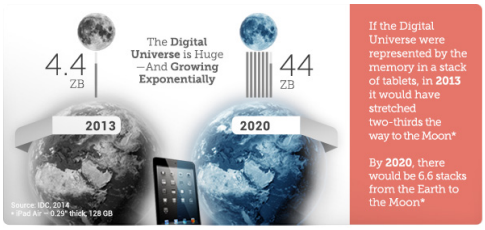

MICHAEL LEWIS
Berkeley
WIRED
The End of Science
The Economist
The data deluge


THE WORLD'S INFORMATION IS DOUBLING EVERY TWO YEARS, WITH A COLOSSAL
1.8 zettabytes to be created & replicated in 2011
New information being created in 2011 also includes replicated information such as shared documents or duplicated DVDs.
In terms of sheer volume, 1.8 ZB of data is equivalent to:
EVERY PERSON IN THE UNITED STATES TWEETING
3 TWEETS PER MINUTE
or
OVER
200 BILLION HD MOVIES
EACH 120 MINUTES LONG


Storing 1.8 ZB of information would take:
57.5 BILLION
32GB APPLE IPADS
1 = 10 billion
WITH THAT MANY IPADS WE COULD BUILD A MOUNTAIN OF IPADS THAT IS
25-TIMES HIGHER
THAN MT. FUJI


25-TIMES HIGHER
THAN MT. FUJI
MT. FUJI
3,776 m
MT. IPAD
94,400 m
SOURCES: Extracting Value from Chaos IDC Digital Universe study (sponsored by EMC Corporation)

Updated…



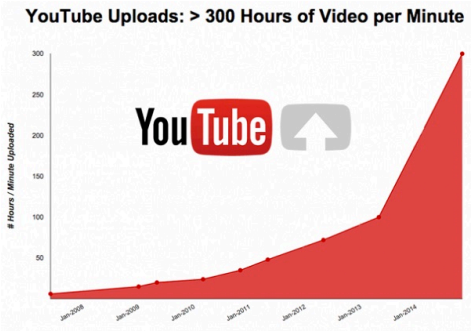
What is going on?

- the internet?

What is going on?

http://flickr.com/photos/mricon/1836673/

What is going on?



Industrial Revolution of Data!

- Software Logs
- RFID
- GPS
- IoT
- Quantified self
- Microphones
- Cameras
- …

This makes me feel…

This makes me feel…

- information is knowledge
  - albert einstein
- knowledge is power
  - sir francis bacon
- with great power comes great responsibility
  - uncle ben

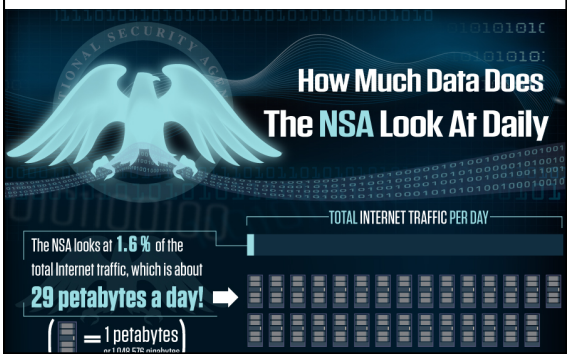
This makes me feel…

"With a collaborative spirit, with a collaborative platform where people can upload data, explore data, compare solutions, discuss the results, build consensus, we can … engage passionate people, local communities, media and this will raise – incredibly – the amount of people who can understand what is going on.

And this would have fantastic outcomes: the engagement of people, especially new generations; it would increase knowledge, unlock statistics, improve transparency and accountability of public policies, change culture, increase numeracy, and in the end, improve democracy and welfare."

– Enrico Giovannini, Chief Statistician, OECD. June, 2007

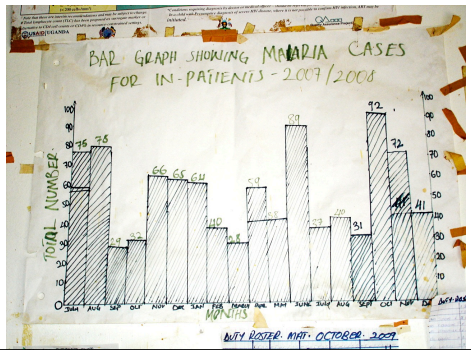
This makes me feel…



This makes me feel…

- 76% of children born in sub-Saharan Africa are unregistered
  - UNICEF, 2008.

This makes me feel…



So…Why?

Data will be at the center of the major issues and events of our lives.

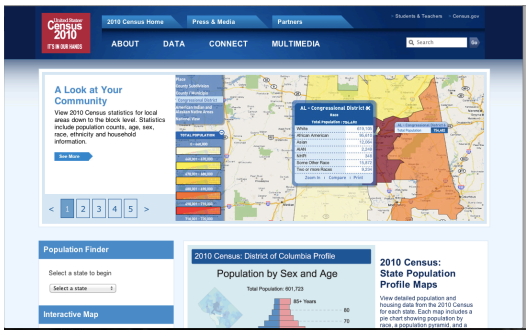
Queries for Today

- Why?
- What?
- Who?
- How?
- For instance?

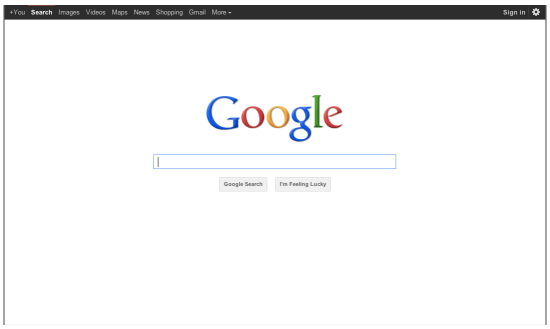
What: Spot the Database

What: Spot the Database

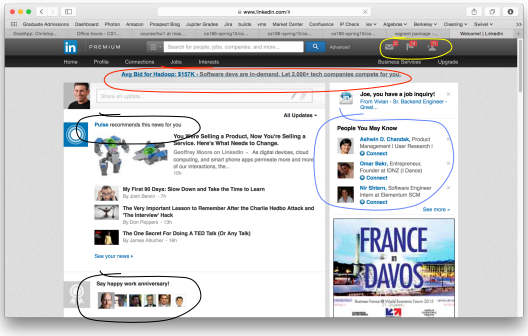
What: Spot the Database



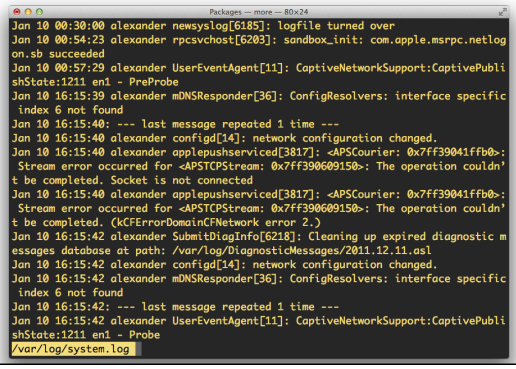
What: Spot the Database



What: Spot the Database



What: Spot the Database



What: Spot the Database

What is a Database?

- Let's not split hairs.

A *database* is a large collection of structured data

What is a DBMS?

A *Database Management System (DBMS)* is software that stores, manages and/or facilitates access to databases.

- Traditionally, term was used narrowly
  - Relational databases with transactions
- Now market and terms in rapid transition
  - The tech remains (roughly) the same
  - Various pressures to remix this tech in new ways.
    - HW
    - Volume
    - Widening variety of usage
  - Good time to focus on fundamentals!

What: Is an OS a DBMS?

- Data can be stored in RAM
  - every programming language offers this
  - RAM is fast, and random access
  - isn't this heaven?

What: Is an OS a DBMS?

- Data can be stored in RAM
  - every programming language offers this
  - RAM is fast, and random access
  - isn't this heaven?
- Every OS includes a File System
  - manages *files* on a persistent disk
  - allows *open, read, seek, close* on a file
  - allows protections to be set on a file
  - drawbacks relative to RAM?

What: File System vs DBMS

- Thought Experiment 1:
  - You and your project partner edit the same file.
  - You both save it at the same time.
  - Whose changes survive?

a) yours
b) partner's
c) both
d) neither
e) ???

What: File System vs DBMS

- Thought Experiment 1:
  - You and your project partner edit the same file.
  - You both save it at the same time.
  - Whose changes survive?

a) yours
b) partner's
c) both
d) neither
e) ???

- Thought Experiment 2:
  - You're updating a file.
  - The power goes out.
  - Which changes survive?

a) all
b) none
c) all since last save
d) ???

What: File System vs DBMS

- Thought Experiment 1:
  - You and your project partner edit the same file.

Q: How do you code against an API that guarantees you "???" ?

A: *Very carefully.*

  - The power goes out.
  - Which changes survive?

a) all
b) none
c) all since last save
d) ???

What: Database Systems

- What more could we want than a file system?
  - Clear *API contracts* regarding data
    - concurrency control, replication, recovery
  - Simple, efficient, well-defined *ad hoc*[1] queries
  - Efficient, scalable bulk processing
  - Benefits of good data modeling
- S.M.O.P.[2]? Not really…

[1]ad hoc: formed or used for specific or immediate problems or needs
[2]SMOP: Small Matter Of Programming

What: Current Market

- Relational DBMSs still anchor the software industry
  - Elephants: Oracle, Microsoft, IBM, Teradata, HP, EMC, …
  - Open source: MySQL, PostgreSQL
  - Emerging Variants: In-Memory, Column-oriented
- Open Source "NoSQL" is growing
  - Analytics: Hadoop MapReduce, Spark
  - Key-value stores: Cassandra, Mongo, Couch, …
- Search SW is an important special case
  - Google & Bing, Solr, Lucene
- Cloud services are expanding quickly
  - Amazon Redshift/ElasticSearch, Google Cloud Dataflow

What will we learn?

- Design patterns for computing with data
- When, why and how to structure your data
- Basics of how Oracle and Google work
- SQL … and noSQL
- Managing concurrency
- Fault tolerance and Recovery
- Scaling out: parallelism and replication

- The full challenge is really hard…

What: Summing up

- Data is at the center of many things.

What: Summing up

- Data is at the center of ~~many things~~ *everything*.

What: Summing up

- Data is at the center of ~~many things~~ *everything*.
- For instance: computer science.

What: Summing up

You might think: in CS186 we learn to apply computer science to Big Data.

No. The techniques we'll learn in CS186 are the key to scalable computer science.

This class should apply very broadly.

Be the tip of the spear

- These professions are just emerging:
  - Cloud programmer
  - Data scientist
  - Data engineer
  - Machine Learning architect
- In 5 years, this will be a large fraction of the computing workforce.
- Now is the time!

Who?

- Instructor
  - Prof. Eric Brewer
- TAs
  - Michelle Nguyen
  - Matthew Deng
  - Richard Liaw
  - Andrew Liu
  - Bryan Munar
  - Ajay Solanky
  - Evan Ye
  - Pete Yeh

How? Workload

- 45%: Projects:
  use and implementation of a database
- 10%: Short weekly quizzes ("vitamins")
- 45% Exams – 3 midterms, but no final

How? Administrivia, cont.

- Textbook
  - *Database Management Systems, 3rd Edition*
    - Ramakrishnan and Gehrke
- Suggested
  - I wouldn't buy any more textbooks
  - Website has links to programming resources
- Grading, hand-in policies, etc. are on Piazza
- Cheating policy: zero tolerance
  - We have the technology…

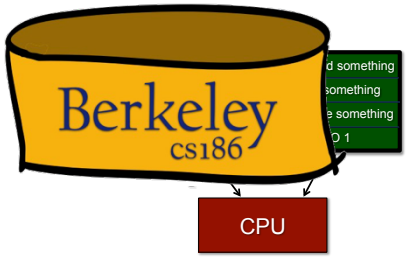
How? Administrivia, cont.

- All class communication via Piazza
  - piazza.com/berkeley/fall2015/cs186
  - announcements and discussion
  - *read it regularly*
  - post all questions/comments there
  - *direct email is not a good idea*
- Solo and Team Homework Projects
  - Teams of 2
  - Think about this now! Find a partner ASAP.

Queries for Today

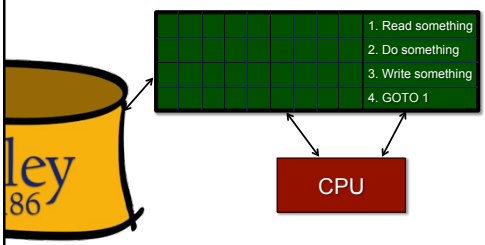
- Why?
- What?
- Who?
- How?
- For instance?

Dealing with Big Data



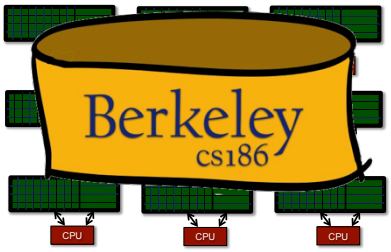
The Von Neumann machine

Dealing with Big Data



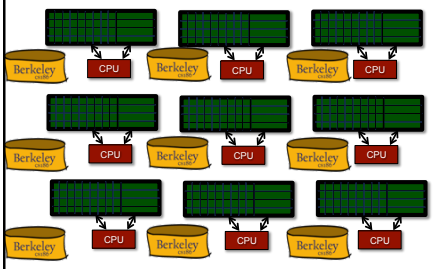
- *"Out-of-Core"* algorithms.

Dealing with Big Data



- Scaling up

Dealing with Big Data



- Distributed Computing and Parallelism

Basic Patterns for Big Data

- Streaming
- Divide-and-Conquer

Simplifying Assumption

Unordered *collections* of data items.

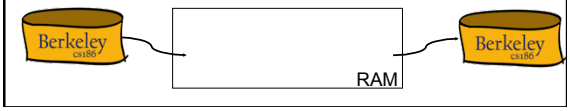
- Corollary: can reorder handling of items!
- The opposite of Von Neumann.
  - unordered handling of unordered data

Disorder is a friend of Scaling

- We can order things to our liking
  - For cache locality, rendezvous, etc.
- We can work on things in *batches*
  - Pick batch sizes to fit our memory hierarchy
  - Pick batch contents based on data affinities
  - OK to postpone data that doesn't fit nicely in the current batch
- We can tolerate non-deterministic orders
  - E.g. the result of parallel execution
  - For efficiency

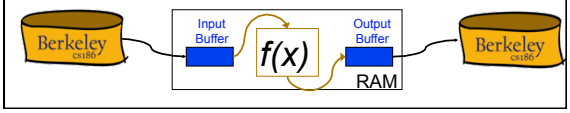
Streaming through RAM

- Simple case: "Map".
  - Goal: Compute *f(x)* for each record, write out the result
  - Challenge: minimize RAM, call read/write rarely



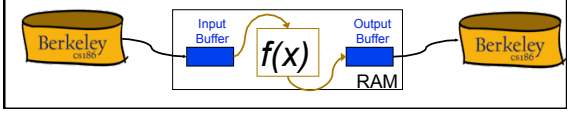
Streaming through RAM

- Simple case: "Map".
  - Goal: Compute *f(x)* for each record, write out the result
  - Challenge: minimize RAM, call read/write rarely
- Approach
  - Read a sizable chunk from INPUT to an *Input Buffer*
  - Write *f(x)* for each item to an *Output Buffer*
  - When Input Buffer is consumed, read another chunk
  - When Output Buffer fills, write it to OUTPUT

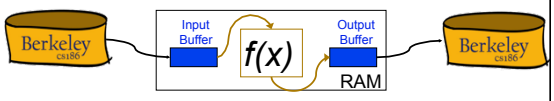


Streaming through RAM

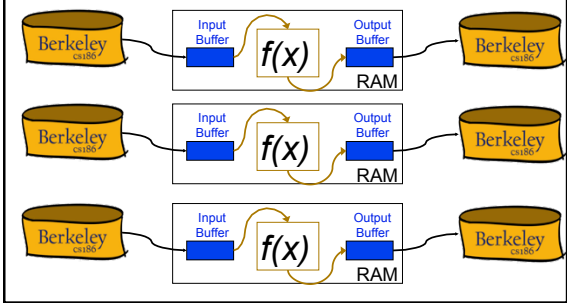
- Simple case: "Map".
  - Goal: Compute *f(x)* for each record, write out the result
  - Challenge: minimize RAM, call read/write rarely
- Approach
  - Read a chunk from INPUT to an *Input Buffer*
  - Write *f(x)* for each item to an *Output Buffer*
  - When Input Buffer is consumed, read another chunk
  - When Output Buffer fills, write it to OUTPUT
- Reads and Writes are *not* coordinated
  - E.g., if f() is Compress(), you read many chunks per write.
  - E.g., if f() is DeCompress(), you write many chunks per read.



Parallelize Me



Parallelize Me



UNIX Pipes

- STDIN and STDOUT streams
- streaming UNIX utilities get/put lines
- Connect 'em up with |
  - OS will do chunking for you
- e.g. "find students who got 100 on one assignment, and got 0 on no assignments"

```
% sed 1d grades.csv | grep ',100'|
grep -v ',0' | cut -f 1 -d ','
```

Handy streaming UNIX utils

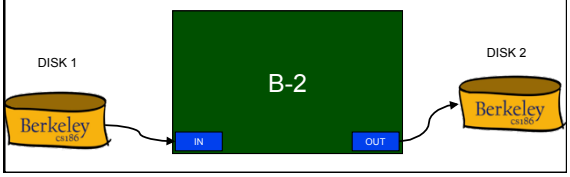
- head, tail, nl, tr, tail, tee, od, cut, grep, sed, awk, split, csplit, …

Rendezvous

- Streaming: one chunk at a time. Easy.
- But some algorithms need certain items to be co-resident in memory
  - not guaranteed to appear in the same input chunk
- *Time-space Rendezvous*
  - in the same place (RAM) at the same time
- There may be many combos of such items

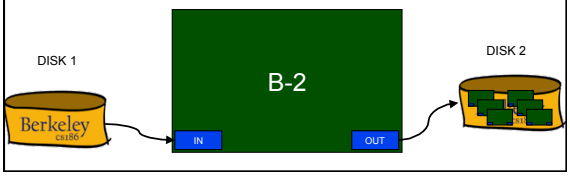
Divide and Conquer

- *Out-of-core* algorithms orchestrate rendezvous.
- Typical RAM Allocation:
  - Assume B chunks worth of RAM available
  - Use 1 chunk of RAM to read into
  - Use 1 chunk of RAM to write into
  - B-2 chunks of RAM as space for rendezvous



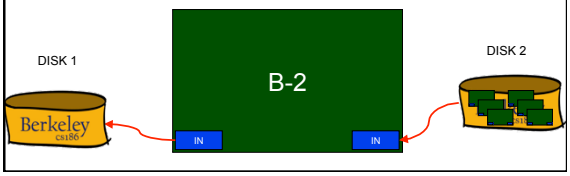
Divide and Conquer

- Phase 1
  - "streamwise" *divide* into N/(B-2) megachunks
  - *conquer* each and write to disk



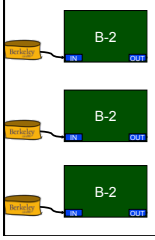
Divide and Conquer

- Phase 2
  - a streaming algorithm over *conquered megachunks.*
  - the streaming must ensure *rendezvous*
    - but among rendezvous groups, order still immaterial!



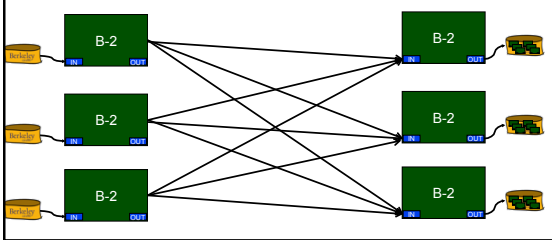
Parallelize Me?

- Phase 1



Parallelize Me?

- Phase 1+: partition data, communicate for rendezvous in space!



Handy UNIX r'vous utils

- non-streaming: sort, wc, tsort
- "stream rendezvous": uniq, join, paste
  - require sorted inputs
- for more UNIX goodness:
  - http://en.wikipedia.org/wiki/List_of_Unix_utilities
  - sort table on Category, search for "Text Processing"

Summing Up 1

- Unordered collection model
- Read in chunks to avoid fixed I/O costs
- Two main techniques
  - Streaming
  - Divide & Conquer for rendezvous
- Parallelism falls out fairly naturally

Summing Up 2

- Pure streaming is fast and low-memory
  - one-pass
  - chunking minimizes I/O fixed costs
- Try to avoid ordering requirements

Up Next

- Two kernels for rendezvous in detail:
  - out-of-core sorting
  - out-of-core hashing

...they differ in the order of divide vs conquer