Database Systems I

CMPT 354 Summer 2024

Zhengjie Miao

10 May 2024

Course Setup

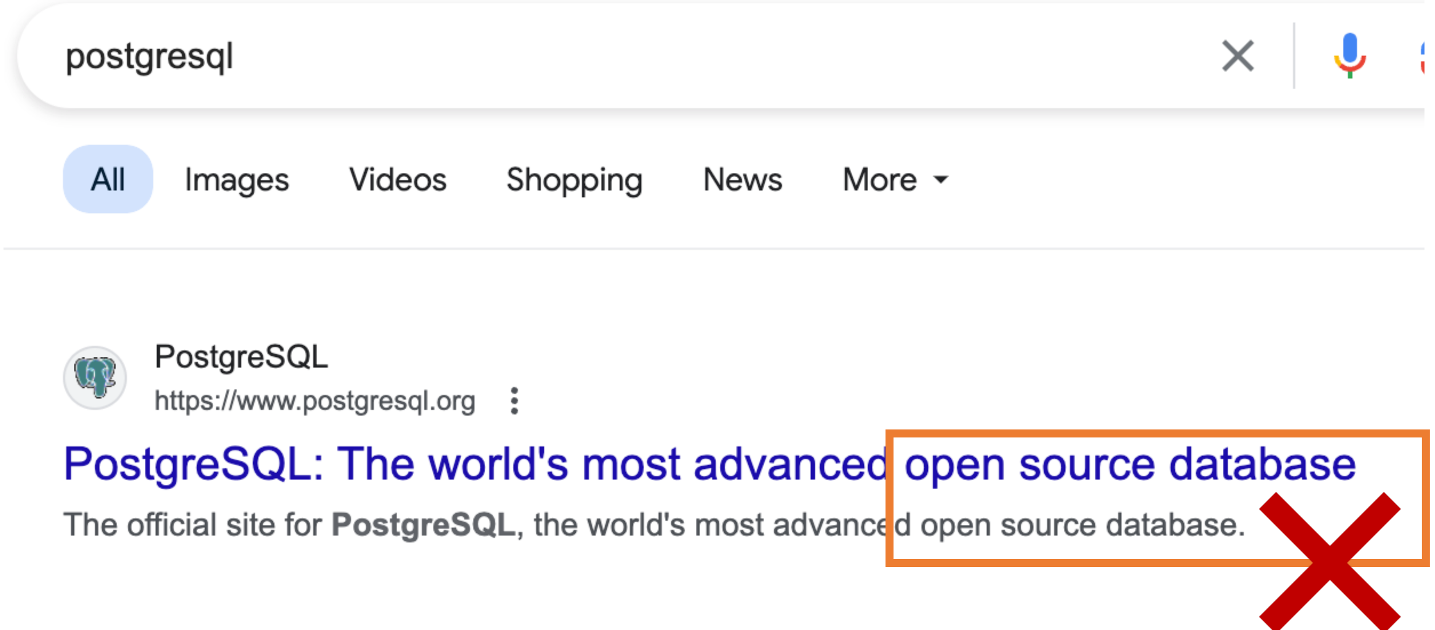
CMPT 354 Topics (schedule subjects to change)

- **Week 1** Introduction & Relational Data Model
- **Week 2** Relational Data Model & Relational Algebra
- **Week 3** Database Design
- **Week 4-5** SQL
- **Week 6** **Midterm I** & SQL (continue)
- **Week 7** NoSQL (part I)
- **Week 8-9** Data Storage & Query Processing
- **Week 10** Query Processing & **Midterm II**
- **Week 11** Query Optimization
- **Week 12-13** Transaction
- **Week 13** NoSQL (part II) & **Midterm III**

Recap: Database & DBMS

Remember the difference between DB and DBMS

- What is a database?
  - An organized collection of data
- Often confused with Database Management Systems (DBMS)
  - A DBMS is a software system that stores, manages, and facilitates access to data

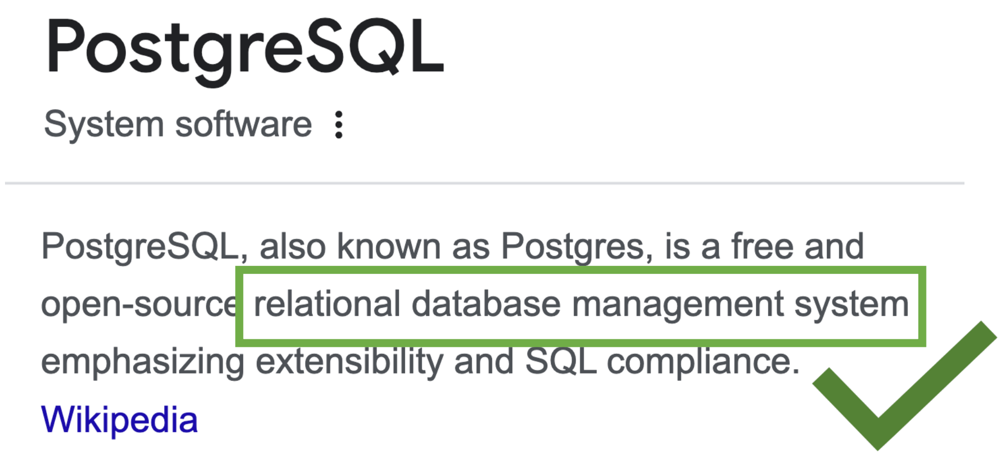


postgresql

PostgreSQL
https://www.postgresql.org

PostgreSQL: The world's most advanced open source database

The official site for **PostgreSQL**, the world's most advanced open source database.



PostgreSQL

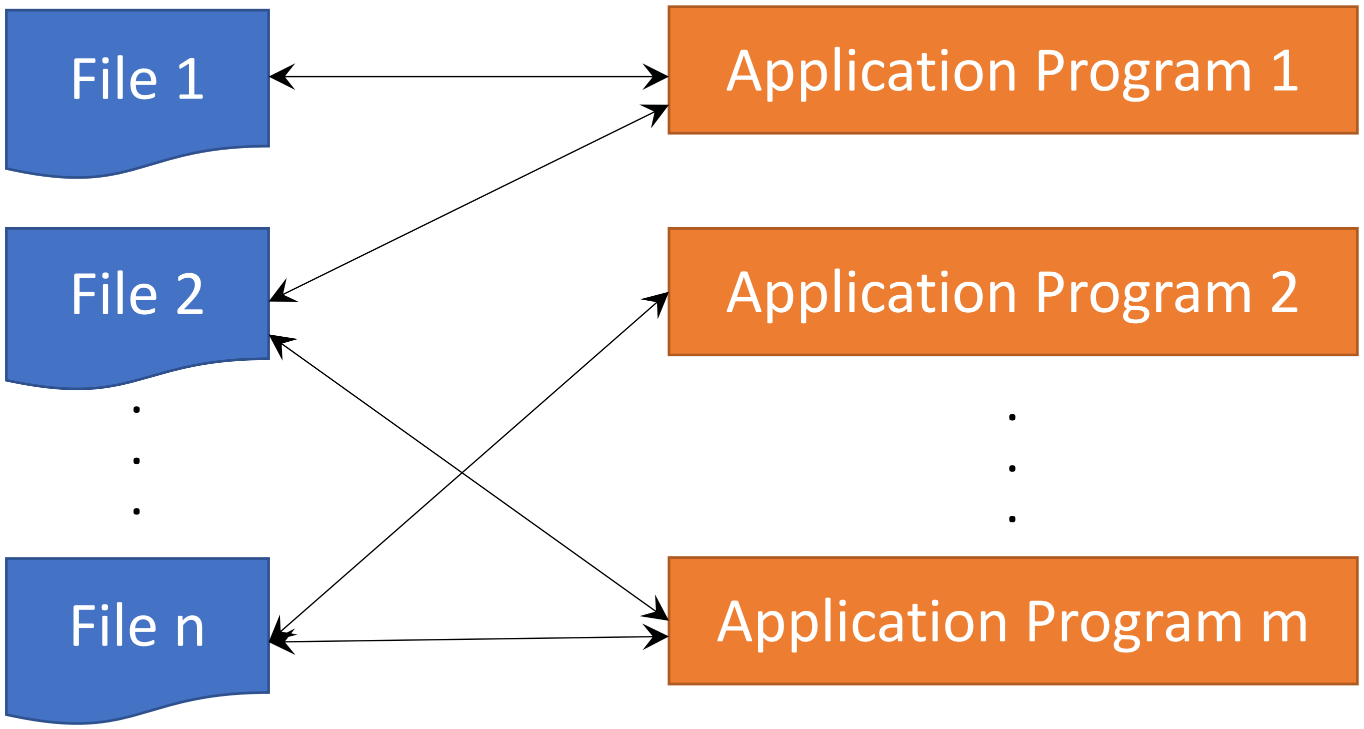
System software

PostgreSQL, also known as Postgres, is a free and open-source relational database management system emphasizing extensibility and SQL compliance.

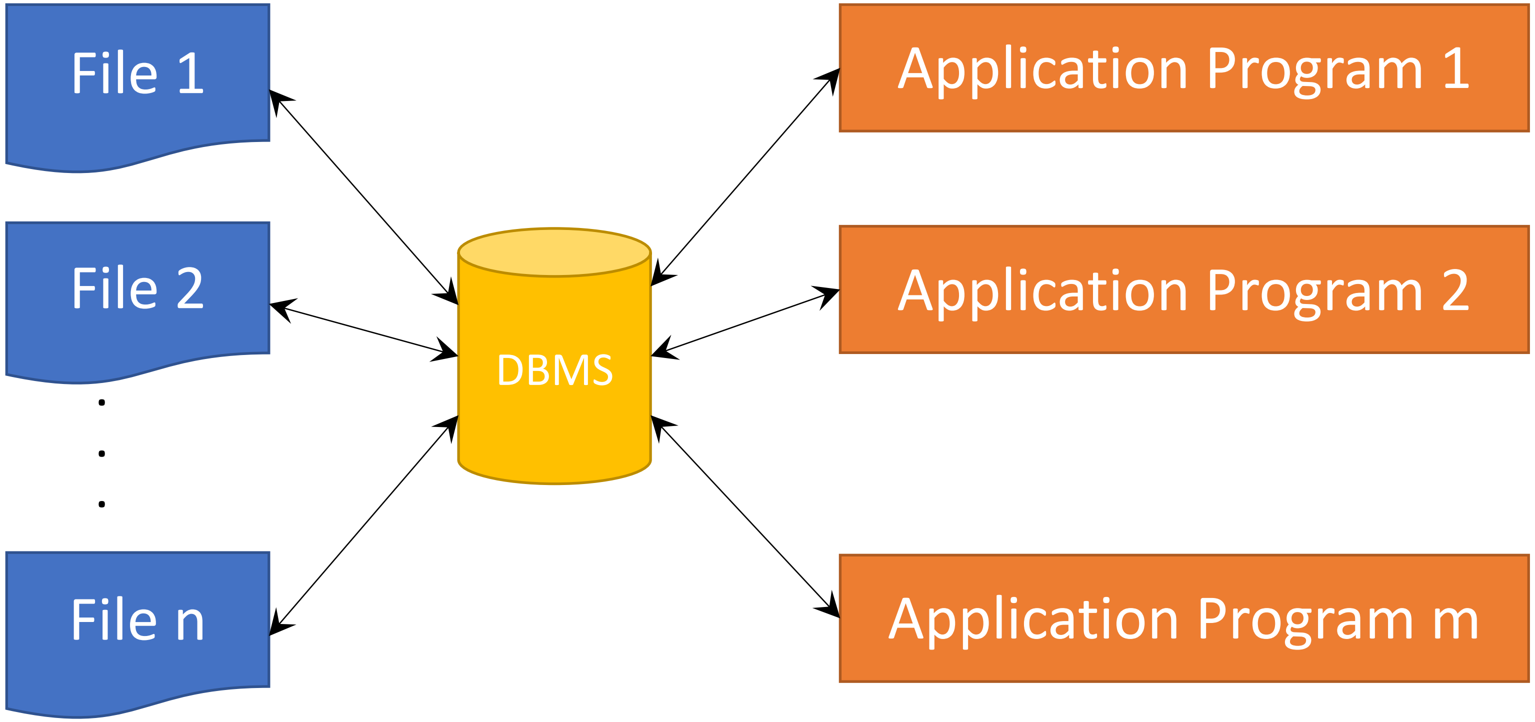
Wikipedia

Data Storage without DBMS

- Data would be collected in many different files and
- Used by many application programs



Data Storage with DBMS



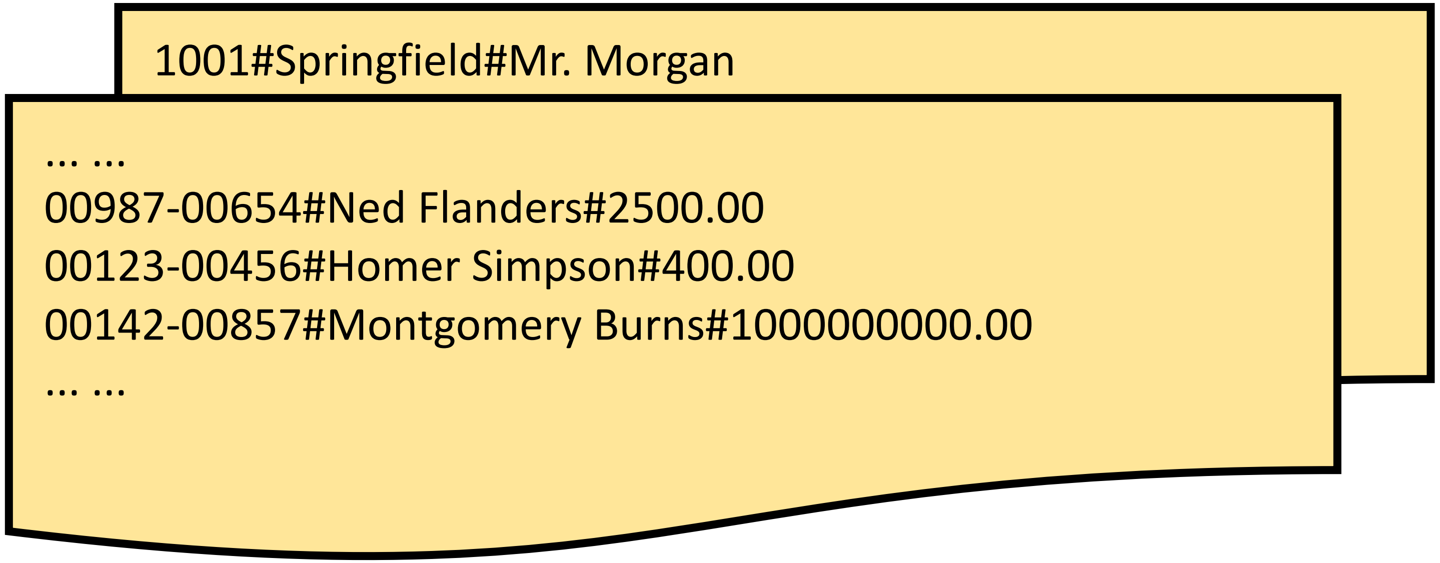
What do you want from a DBMS?

Persistent data structures are different from the use of persistent here

- Keep data around (persistent) Durability?
- Answer questions (queries) about data
- Update data

- Example: a traditional banking application
  - Data: Each account belongs to a branch, has a number, an owner, a balance, …; each branch has a location, a manager, …
  - Persistency: Balance can't disappear after a power outage
  - Query: What's the balance in Homer Simpson's account? What's the difference in average balance between Springfield and Capitol City accounts?
  - Modification: Homer withdraws $100; charge accounts with lower than $500 balance a $5 fee

Sounds simple!

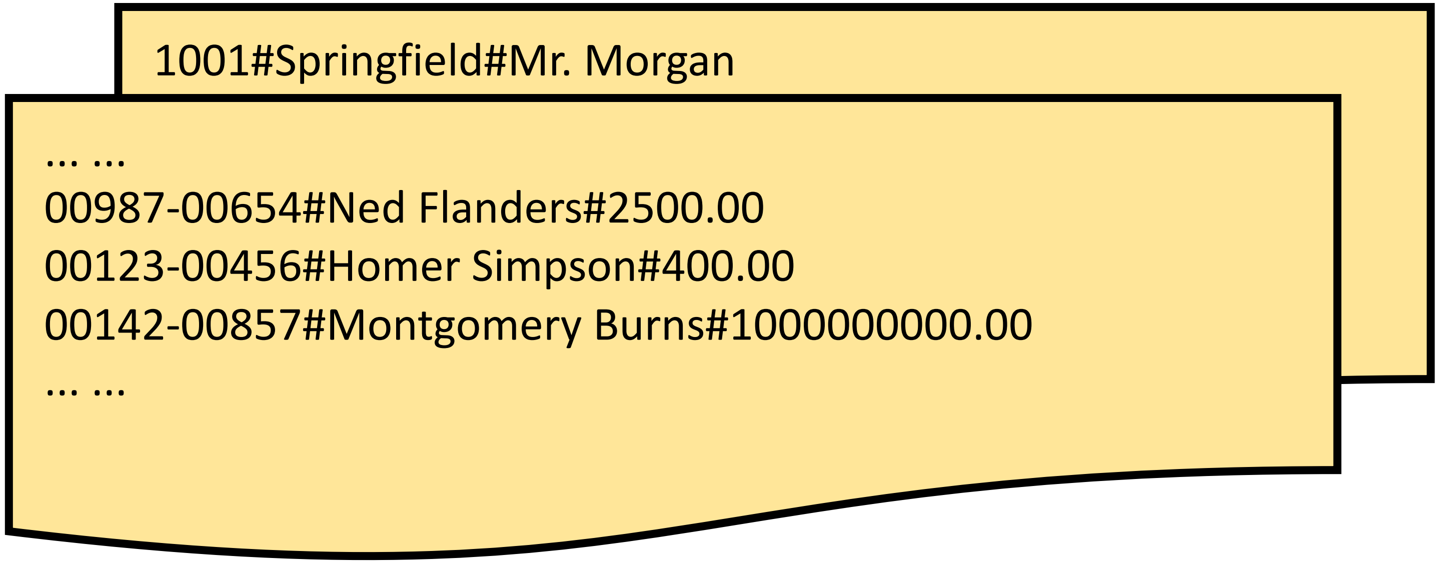


```
1001#Springfield#Mr. Morgan
```

```
… …
00987-00654#Ned Flanders#2500.00
00123-00456#Homer Simpson#400.00
00142-00857#Montgomery Burns#1000000000.00
… …
```

- Text files
- Accounts/branches separated by newlines
- Fields separated by #’s

Query by programming



```
1001#Springfield#Mr. Morgan
```

```
… …
00987-00654#Ned Flanders#2500.00
00123-00456#Homer Simpson#400.00
00142-00857#Montgomery Burns#1000000000.00
… …
```

- What’s the balance in Homer Simpson’s account?
- A simple script
  - Scan through the accounts file
  - Look for the line containing “Homer Simpson”
  - Print out the balance

Query processing tricks

- Tens of thousands of accounts are not Homer's
  - Cluster accounts by owner's initial: those owned by "A..." go into file A; those owned by "B..." go into file B; etc. → decide which file to search using the initial

    Partitioning?
  - Keep accounts sorted by owner name → binary search?
  - Hash accounts using owner name → compute file offset directly
  - Index accounts by owner name: index entries have the form ⟨*owner_name, file_offset*⟩ → search index to get file offset
  - And the list goes on…

Should we not have a unique id for each user?

- What happens when the query changes to: *What's the balance in account 00142-00857?*

Standard DBMS features

- Persistent storage of data
- Logical data model; declarative queries and updates → physical data independence
    - Relational model is the dominating technology today

☞What else?

Concurrency

DBMS is multi-user

- Example

```
get account balance from database;
if balance > amount of withdrawal then
    balance = balance - amount of withdrawal;
    dispense cash;
    store new balance into database;
```

- Homer at ATM1 withdraws $100
- Marge at ATM2 withdraws $50
- Initial balance = $400, final balance = ?
  - Should be $250 no matter who goes first

Concurrency control in DBMS

- Similar to concurrent programming problems?
  - But data not main-memory variables
- Similar to file system concurrent access?
  - Lock the whole table before access
    - Approach taken by MySQL in the old days
    - Still used by SQLite (as of Version 3)
  - But want to control at much finer granularity
    - Or else one withdrawal would lock up all accounts!

Final balance = $300

Homer withdraws $100:

```
read balance; $400




if balance > amount then
   balance = balance - amount; $300
   write balance; $300
```

Marge withdraws $50:

```
read balance; $400
if balance > amount then
   balance = balance - amount; $350
   write balance; $350
```

Final balance = $ 350

Homer withdraws $100:

```
read balance;          $400

if balance > amount then
   balance = balance - amount;
   write balance;          $300
```

Marge withdraws $50:

```
read balance;          $400

$300

if balance > amount then
   balance = balance - amount;                    $350
   write balance;          $350
```

Recovery in DBMS

- Example: balance transfer
  decrement the balance of account X by $100;
  increment the balance of account Y by $100;
- Scenario 1: Power goes out after the first operation
- Scenario 2: DBMS buffers and updates data in memory (for efficiency); before they are written back to disk, power goes out
- How can DBMS deal with these failures?

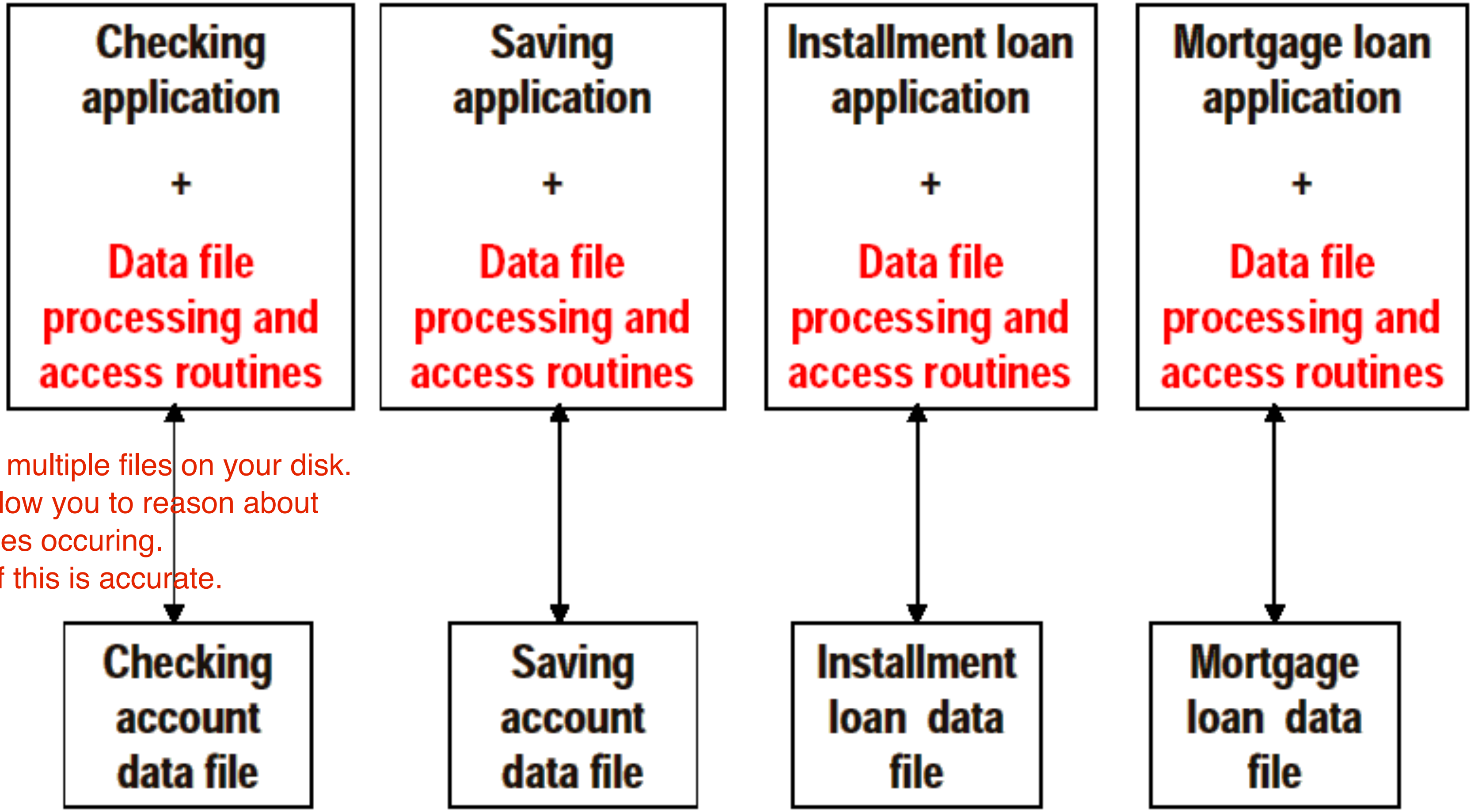
Standard DBMS features: summary

- Persistent storage of data
- Logical data model; declarative queries and updates → physical data independence
- Multi-user concurrent access
- Safety from system failures
- Performance, performance, performance
    - Massive amounts of data (terabytes~petabytes)
    - High throughput (thousands~millions transactions/hour)
    - High availability (≥ 99.999% uptime)

Observations

- There are many techniques—not only in storage and query processing, but also in concurrency control, recovery, etc.
- These techniques get used over and over again in different applications
- Different techniques may work better in different usage scenarios

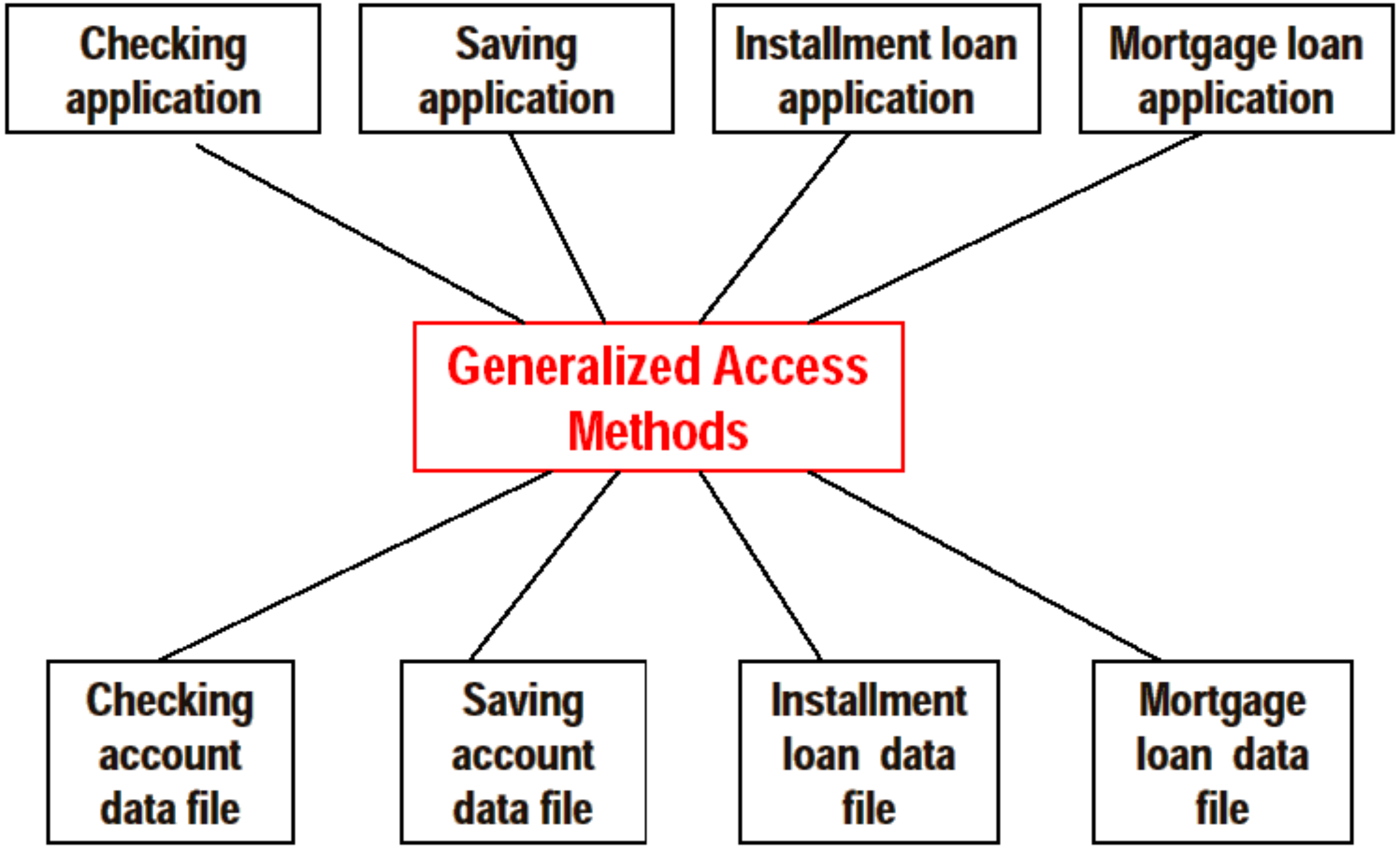
The birth of DBMS – 1


Checking application
+
Data file processing and access routines
Saving application
+
Data file processing and access routines
Installment loan application
+
Data file processing and access routines
Mortgage loan application
+
Data file processing and access routines
Checking account data file
Saving account data file
Installment loan data file
Mortgage loan data file

Applications may access multiple files on your disk.
Having a DBMS will allow you to reason about
the accesses occuring.
Ask professor if this is accurate.

From Hans-J. Schek's *VLDB* 2000 slides

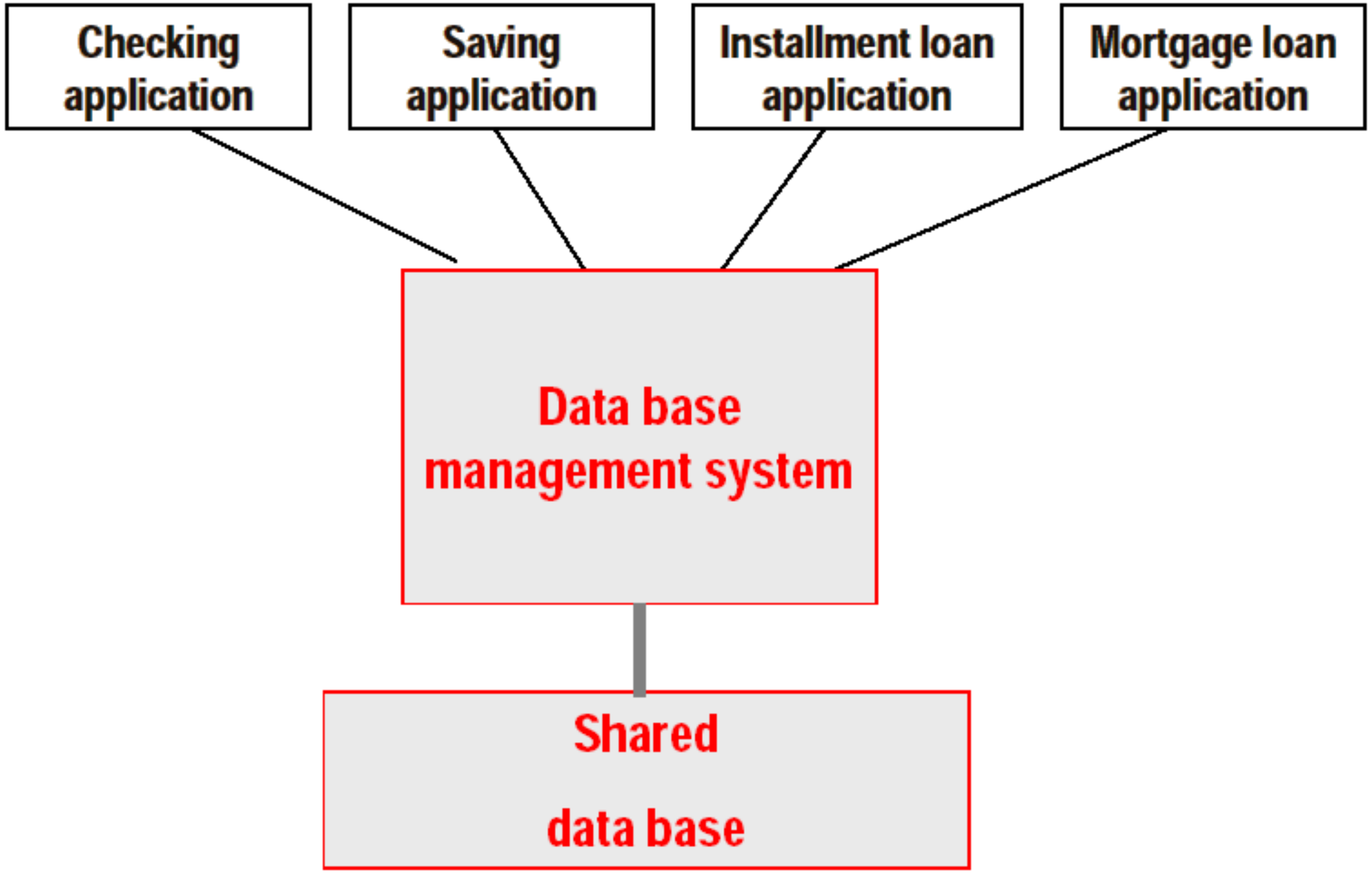
The birth of DBMS – 2



Checking application
Saving application
Installment loan application
Mortgage loan application
Generalized Access Methods
Checking account data file
Saving account data file
Installment loan data file
Mortgage loan data file

From Hans-J. Schek's *VLDB* 2000 slides

The birth of DBMS – 3


Checking application
Saving application
Installment loan application
Mortgage loan application
Data base management system
Shared data base

From Hans-J. Schek's *VLDB* 2000 slides

DBMSs in > a Half Century (1960s–2020s)

When	What
Early 1960s – Early 1970s	Navigational DBMSs
Mid 1970s – Mid 1980s	The Relational Revolution
Mid 1980s – Early 2000s	The Relational DBMS Empire
Mid 2000s – Now	NoSQL and NewSQL Movement

References.

- https://en.wikipedia.org/wiki/Database#History
- What Goes Around Comes Around (Michael Stonebraker, Joe Hellerstein)
- 40 Years VLDB Panel

Early Efforts in Navigational DBMSs (Early1960s – Early 1970s)

- Data Model
  - How to organize data
  - How to access data
- Navigational Data Model
  - Organize data into a multi-dimensional space (i.e., A space of records)
  - Access data by following pointers between records
- Inventor: Charles Bachman
  - The 1973 ACM Turing Award
  - Turing Lecture: "The Programmer As Navigator"

Early Efforts in Navigational DBMSs (Early1960s – Early 1970s)

- Representative Navigational Database Systems
    - Integrated Data Store (IDS), 1964, GE
    - Information Management System (IMS), 1966, IBM
    - Integrated Database Management System (IDMS), 1973, Goodrich
- CODASYL
    - Short for "Conference/Committee on Data Systems Languages"
    - Define navigational data model as standard database interface (1969)

Early Efforts in Navigational DBMSs (Early1960s – Early 1970s)

Mainstream DBMSs in 1960s use Hierarchical data model:

```
      Department (dname, floor_number, budget)
          {
          Employee (name, salary birthdate)
          }
```

Pseudo-code in a high-level hierarchical language:

```
   For all Departments where floor = 1
   Select Employee.name
```

Michael Stonebraker, “Those Who Forget the Past Are Doomed to Repeat It”

More complex query in CODASYL

- Query: Who have accounts with 0 balance managed by a branch in Springfield?
- Pseudo-code of a CODASYL application:

```
Use index on account(balance) to get accounts with 0 balance;
For each account record:
  Get the branch id of this account;
  Use index on branch(id) to get the branch record;
  If the branch record’s location field reads “Springfield”:
    Output the owner field of the account record.
```

Queries become intricate in this model

- Programmer controls “navigation”: accounts → branches
  - How about branches → accounts?

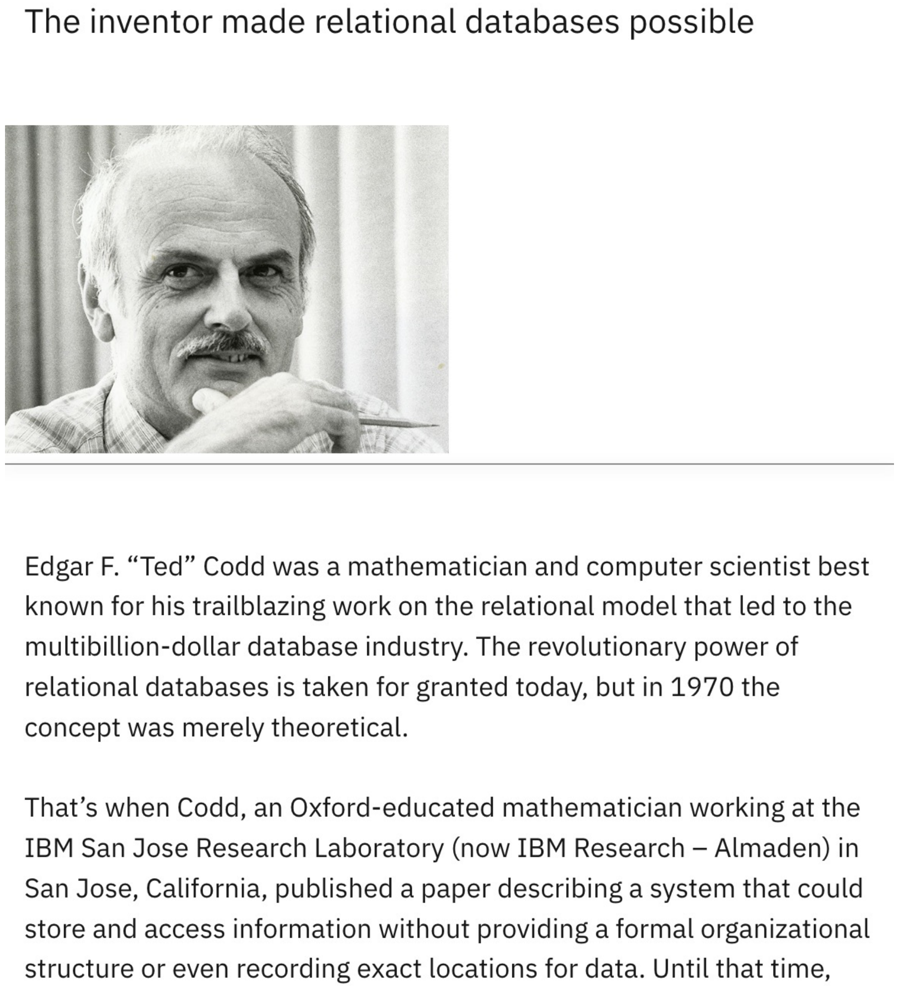
What's wrong?

- Queries become inevitably complex and hard to code when the database becomes complex
- The best navigation strategy & the best way of organizing the data depend on data/workload characteristics
  - The programmer needs to "navigate" in the database.
  - To write efficient code, programmers also need to worry about data/workload characteristics.
  - The programmer has to rewrite everything when the database changes.

Codd: Relational Databases

- **"No, let's do it this way instead:"** we need a model that provides data independence
  - Applications should NOT worry about how data is physically structured and stored and should be independent from the growth in data types and changes in data representation.
  - Programmer specifies what answers a query should return, but not how the query is executed, leaving the optimization to DBMS.
  - "A Relational Model of Data for Large Shared Data Banks" in 1970

Data Independence



The inventor made relational databases possible

Edgar F. "Ted" Codd was a mathematician and computer scientist best known for his trailblazing work on the relational model that led to the multibillion-dollar database industry. The revolutionary power of relational databases is taken for granted today, but in 1970 the concept was merely theoretical.

That's when Codd, an Oxford-educated mathematician working at the IBM San Jose Research Laboratory (now IBM Research – Almaden) in San Jose, California, published a paper describing a system that could store and access information without providing a formal organizational structure or even recording exact locations for data. Until that time,

Michael Stonebraker, "Those Who Forget the Past Are Doomed to Repeat It"

Codd: Relational Databases

- A simple model: data is stored in relations (tables)
- In modern SQL:

```
SELECT Account.owner
FROM Account, Branch
WHERE Account.balance = 0
AND Branch.location = 'Springfield'
AND Account.branch_id = Branch.branch_id;
```

You do not need to know how the tables are stored on the disk

- Programmer specifies what answers a query should return, but not how the query is executed
- DBMS picks the best execution strategy based on availability of indexes, data/workload characteristics, etc.
  - Provides physical data independence

Michael Stonebraker, “Those Who Forget the Past Are Doomed to Repeat It”

Physical data independence

- Applications should NOT worry about how data is physically structured and stored
- Applications should work with a logical data model and declarative query language
- Leave the implementation details and optimization to DBMS
- The single most important reason behind the success of DBMS today
    - And a Turing Award for E. F. Codd in 1981

The 1974 Debate on Data Models

- One Slide (Navigational Model)
  - Led by Charles Bachman (1973 ACM Turing Award)
  - Has built mature systems
  - Dominated the database market
- The other Slide (Relational Model)
  - Led by Ted Codd (mathematical programmer, IBM)
  - A theoretical paper with no system built
  - Little support from IBM

The 1974 Debate on Data Models

Sublanguage example: SQL query language

- At ACM SIGFIDET (precursor of SIGMOD)
  - Question 1: Are high-level data sublanguages a good idea?
  - Question 2: Are tables the best data structure or should one use a network or hierarchy?
- Navigational model is bad
  - Data Access: No declarative language
  - Data Organization: So complex
- Relational model is bad
  - Data Access: No system proof that declarative language is viable
  - Data Organization: A special case of navigational model

Michael Stonebraker, “Those Who Forget the Past Are Doomed to Repeat It”

The Relational Revolution

1. Which data model is better in theory?
   - (the 1974 debate)

Relational model

2. Which data model is better in practice?
   - (Late 1970 – Early 1980)

Ask professor about practice vs buisiness.

3. Which data model is better in business?
   - (Early 1980 – Mid 1980)

The “Practice” Campaign

- The Big Question
  - Can a relational database system perform as good as a navigational system?
- System prototypes
  - Ingres at UC Berkeley (early and pioneering)
  - System R at IBM (arguably got more stuff “right”)
- The System R Team
  - Query Optimization (Patricia P. Griffiths et al.)
  - SQL (Donald D. Chamberlin et al.)
  - Transaction (Jim Gray et al.)

The “Business” Campaign

- Commercialization of Relational Database Systems
- Not as easy as we thought
- Three reasons that led relational database systems to won
  - The minicomputer revolution (1977)
  - Competing products (e.g. IDMS) could not be ported to the minicomputer
  - Relational front end was not added to navigational database systems

What Can We Learn?

- Lesson 1.

The winning of theory ≠ The winning of practice

Until the invention of System R, people did not think SQL could be practical

- Lesson 2.

The winning of practice ≠ The winning of business

Ask professor to clarify about the difference between practice and buisiness.

Does buisiness mean large use cases?

- Lesson 3.

Everyone can get a chance to win

Early Efforts in Navigational DBMSs (Early1960s – Early 1970s)

- Parallel and distributed DBs (1980 – 1990)
    - SystemR*, Distributed Ingres, Gamma, etc.

- Objected-oriented DB (1980 – 1990)
    - Objects: Data/Code Integration
    - Extensibility: User-defined functions, User-defined data types

- MySQL and PostgreSQL (1990s)
    - Widely used open-source relational DB systems

DBMSs in > a Half Century (1960s–2020s)

When	What
Early 1960s – Early 1970s	Navigational DBMSs
Mid 1970s – Mid 1980s	The Relational Revolution
Mid 1980s – Early 2000s	The Relational DBMS Empire
Mid 2000s – Now	NoSQL and NewSQL Movement

References.

- https://en.wikipedia.org/wiki/Database#History
- What Goes Around Comes Around (Michael Stonebraker, Joe Hellerstein)
- 40 Years VLDB Panel

OLTP

OnLine Transaction Processing

Workload

High-frequent Updates + Small Queries

OLAP

OnLine Analytical Processing

Workload

Low-frequent Updates + Big Queries

The NoSQL & NewSQL Movement

1. Which design is better for OLTP?
   - NoSQL vs. Relational DBMS?
2. Which design is better for OLAP?
   - MapReduce vs. Relational DBMS?

What Happened To OLTP?

- RDBMS (1970 – Now)
- NoSQL (2000 – Now)
- NewSQL (2010 – Now)

RDBMS (1970 – Now)

- Traditional SQL vendors (“OldSQL”)

Microsoft SQL Server   ORACLE DATABASE   IBM DB2   . . .

Still very big market!!!

- Limitation 1: Not Scalable
- Limitation 2: Pre-defined Schema

The advent of Web 2.0

Read-only Web → Read-write Web

- Highly Scalable
  - Scale to 1,000,000 users and 1000 servers
- Highly Available
  - Available 24 hours a day, 7 days a week
- Highly Flexible
  - Flexible schema and flexible data types

Emerging of NoSQL DBMS

- Internet Boom (Early 2000)
  - Larger data volume that cannot be fit in a single machine
  - Faster data updates that cannot be handled by a single machine

- Commercial distributed database systems are expensive

- Open-source database systems do not support distributed computing well

NoSQL Pioneers

- Memcached [Fitzpatrick 2004]
  - In-memory indexes can be highly scalable
- BigTable [Chang et al. 2006]
  - Persistent record storage could be scaled to thousands of nodes
- Dynamo [DeCandia et al. 2007]
  - Eventual consistency allows for higher availability and scalability

Ex:

you buy a book on Amazon when at that same moment another customer has bought the last copy of the book.
You can complete the transaction, but you will be notified that there are no books left

NoSQL Categories

Will focus on MongoDB

NoSQL	Data Model	Example Systems
Key-value Stores	Hash	DynamoDB, Riak, Redis, Membase
Document Stores	Json	SimpleDB, CouchBase, MongoDB
Wide-column Stores	Big Table	Hbase, Cassandra, HyperTable
Graph Database	Graph	Neo4J, InfoGrid, GraphBase

Graph Database is for social media.
Store relationships among accounts

NoSQL Limitations

- Low-level Language
  - Simple read/write database operators

what are simple operators? What are complex operators?

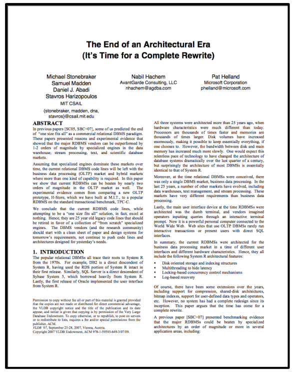
- Weak Consistency
  - Eventual Consistency

Eventual consistency may not be suitable for different scenarios.

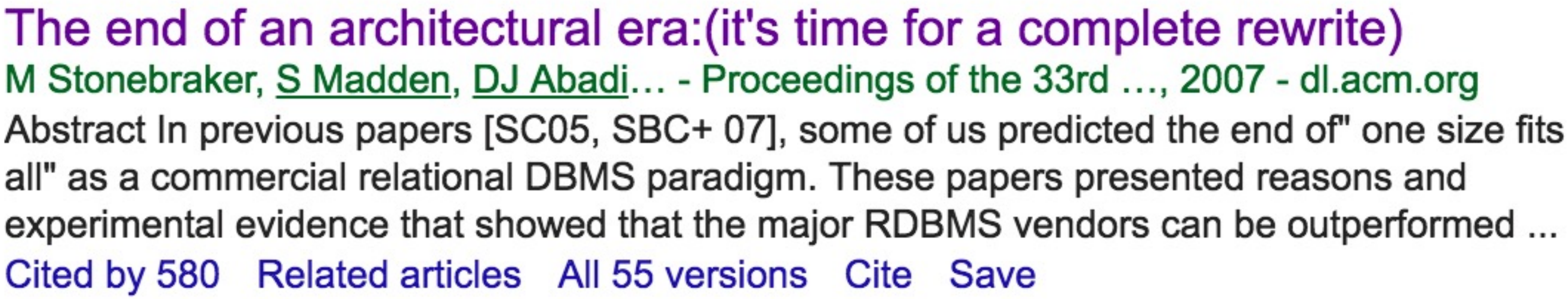
- Lack of Standardization
  - 100+ NoSQL systems

NewSQL

Strong Consistency + High Scalability



The End of an Architectural Era
(It's Time for a Complete Rewrite)



The end of an architectural era:(it's time for a complete rewrite)

M Stonebraker, S Madden, DJ Abadi… - Proceedings of the 33rd …, 2007 - dl.acm.org

Abstract In previous papers [SC05, SBC+ 07], some of us predicted the end of" one size fits all" as a commercial relational DBMS paradigm. These papers presented reasons and experimental evidence that showed that the major RDBMS vendors can be outperformed ...

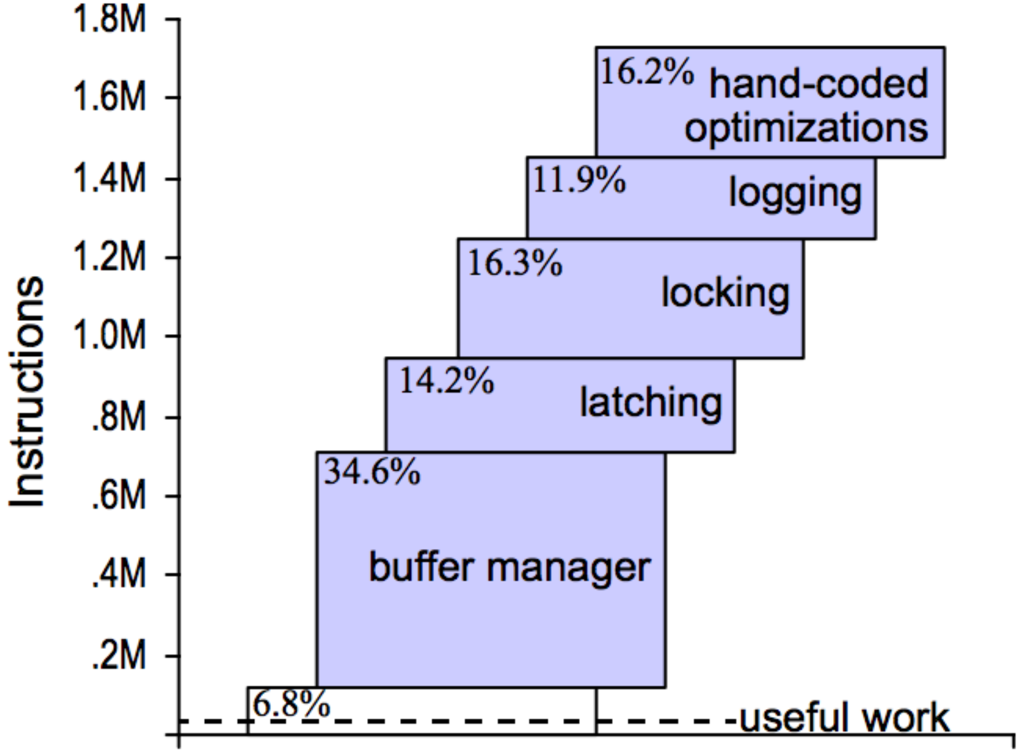
90% query time spent on overhead

OLTP Through the Looking Glass, and What We Found There

Stavros Harizopoulos
HP Labs
Palo Alto, CA
stavros@hp.com

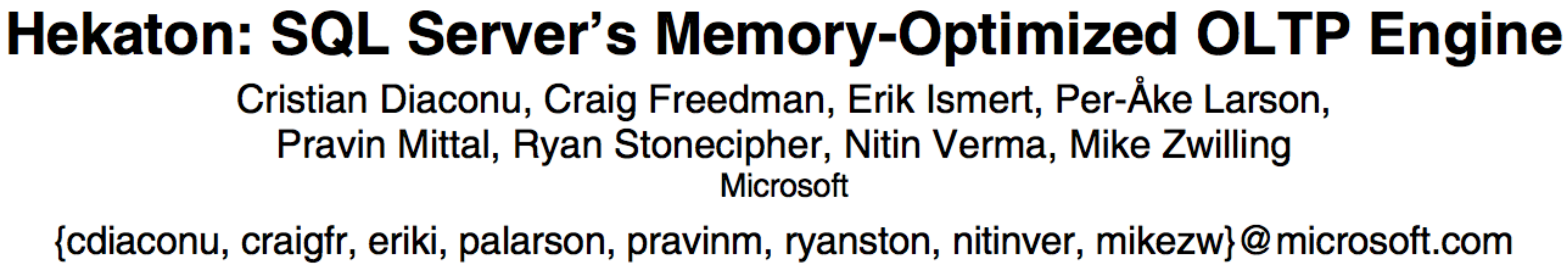
Daniel J. Abadi
Yale University
New Haven, CT
dna@cs.yale.edu

Samuel Madden, Michael Stonebraker
Massachusetts Institute of Technology
Cambridge, MA
{madden, stonebraker}@csail.mit.edu



NewSQL

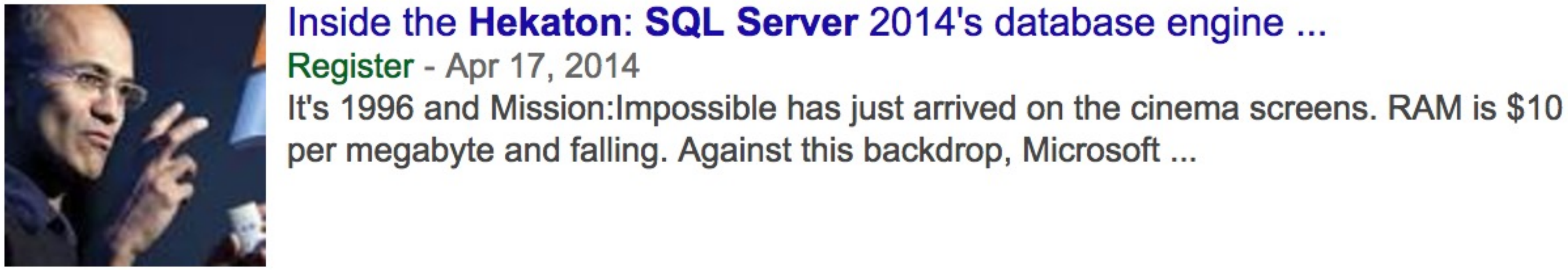
- **Limitations**
  - Scalable but not highly scalable
  - Available but not highly available
  - Flexible but not highly flexible



Hekaton: SQL Server’s Memory-Optimized OLTP Engine

Cristian Diaconu, Craig Freedman, Erik Ismert, Per-Åke Larson,
Pravin Mittal, Ryan Stonecipher, Nitin Verma, Mike Zwilling
Microsoft
{cdiaconu, craigfr, eriki, palarson, pravinm, ryanston, nitinver, mikezw}@microsoft.com

Hekaton: SQL Server's Memory-optimized OLTP Engine (2013)



Inside the **Hekaton**: **SQL Server** 2014's database engine ...

Register - Apr 17, 2014

It's 1996 and Mission:Impossible has just arrived on the cinema screens. RAM is $10 per megabyte and falling. Against this backdrop, Microsoft ...

Summary

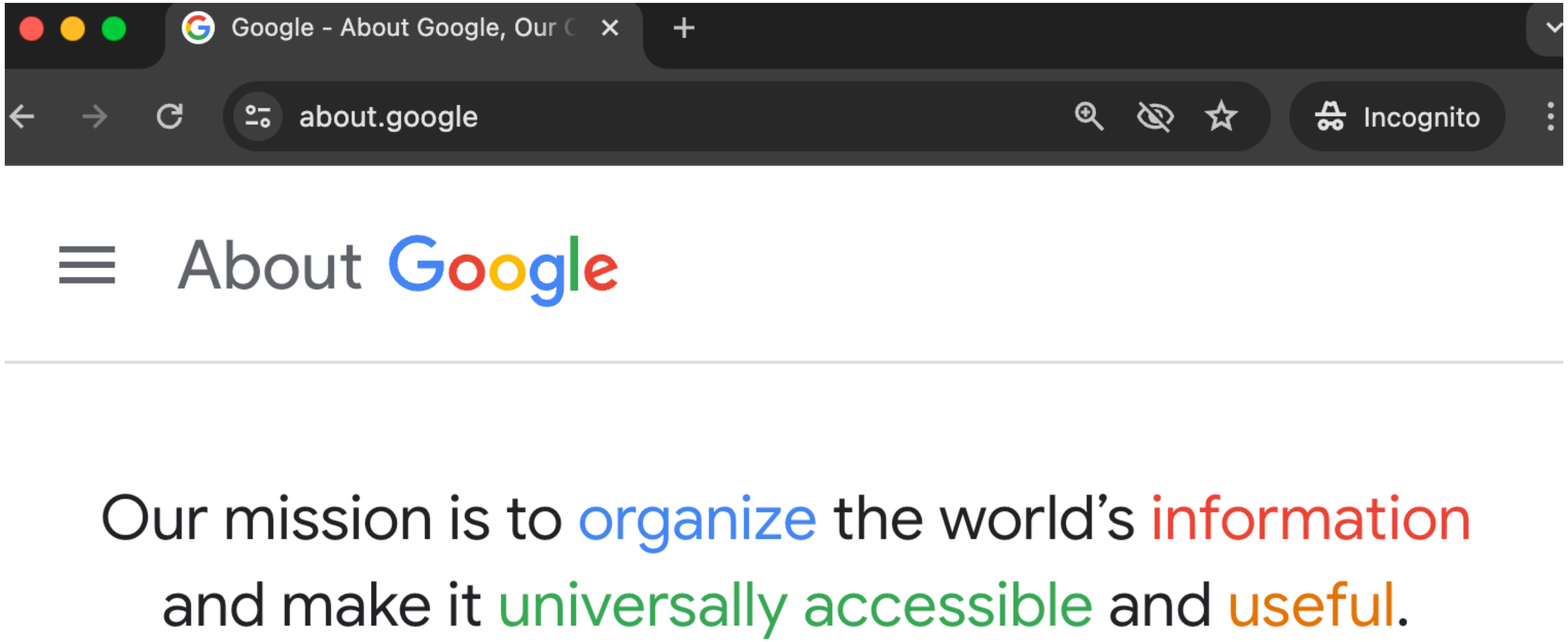
NoSQL supports scalability and availability

RBDMS has consistency

- Why RDBMS (“OldSQL”)?
- Why NoSQL?
- Why NewSQL?

The NoSQL & NewSQL Movement

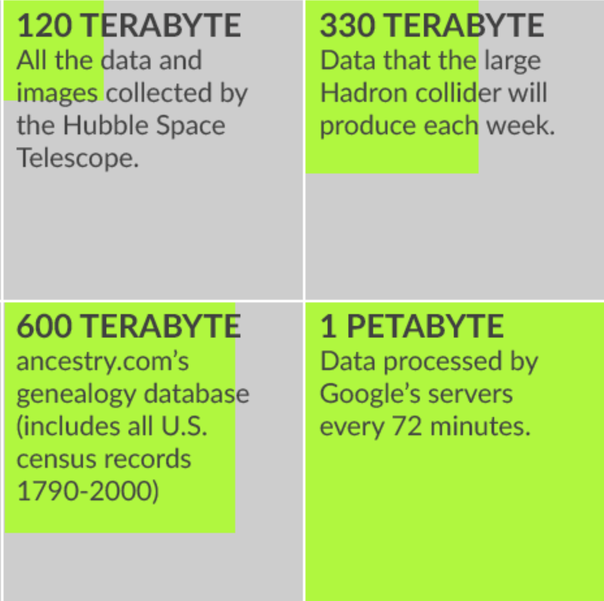
1. Which design is better for OLTP?
   - NoSQL vs. Relational DBMS?
2. **Which design is better for OLAP?**
   - **MapReduce vs. Relational DBMS?**



About Google

Our mission is to organize the world's information and make it universally accessible and useful.

What is GFS?



120 TERABYTE
All the data and images collected by the Hubble Space Telescope.

330 TERABYTE
Data that the large Hadron collider will produce each week.

600 TERABYTE
ancestry.com's genealogy database (includes all U.S. census records 1790-2000)

1 PETABYTE
Data processed by Google's servers every 72 minutes.

1 PB = 100GB *10,000 Machines

How to store 1PB using 10,000 machines?

GFS (HDFS)

How to process 1PB using 10,000 machines?

MapReduce

MapReduce

- Many problems can be processed in this pattern:
  - Given a lot of unsorted data
  - Map: extract something of interest from each record
  - Shuffle: group the intermediate results in some way
  - Reduce: further process (e.g., aggregate, summarize, analyze, transform) each group and write final results

  (Customize map and reduce for problem at hand)
- Make this pattern easy to program and efficient to run
  - Original Google paper in *OSDI* 2004
  - Hadoop has been the most popular open-source implementation
  - Spark still supports it

Why MapReduce?

1. Fault Tolerant
   - (Your program will be OK when failures happen)

Assuming each machine is independent, we can reduce the probability to a near zero value?

# of machines	Failure Probability
1	0.1%
10	0.9%
100	9.5%
1000	63.2%
10,000	??? 99.9%

What is a node?

Cost for 5 nodes	Failure Probability
$100/day	0.1%
$1/day	10%

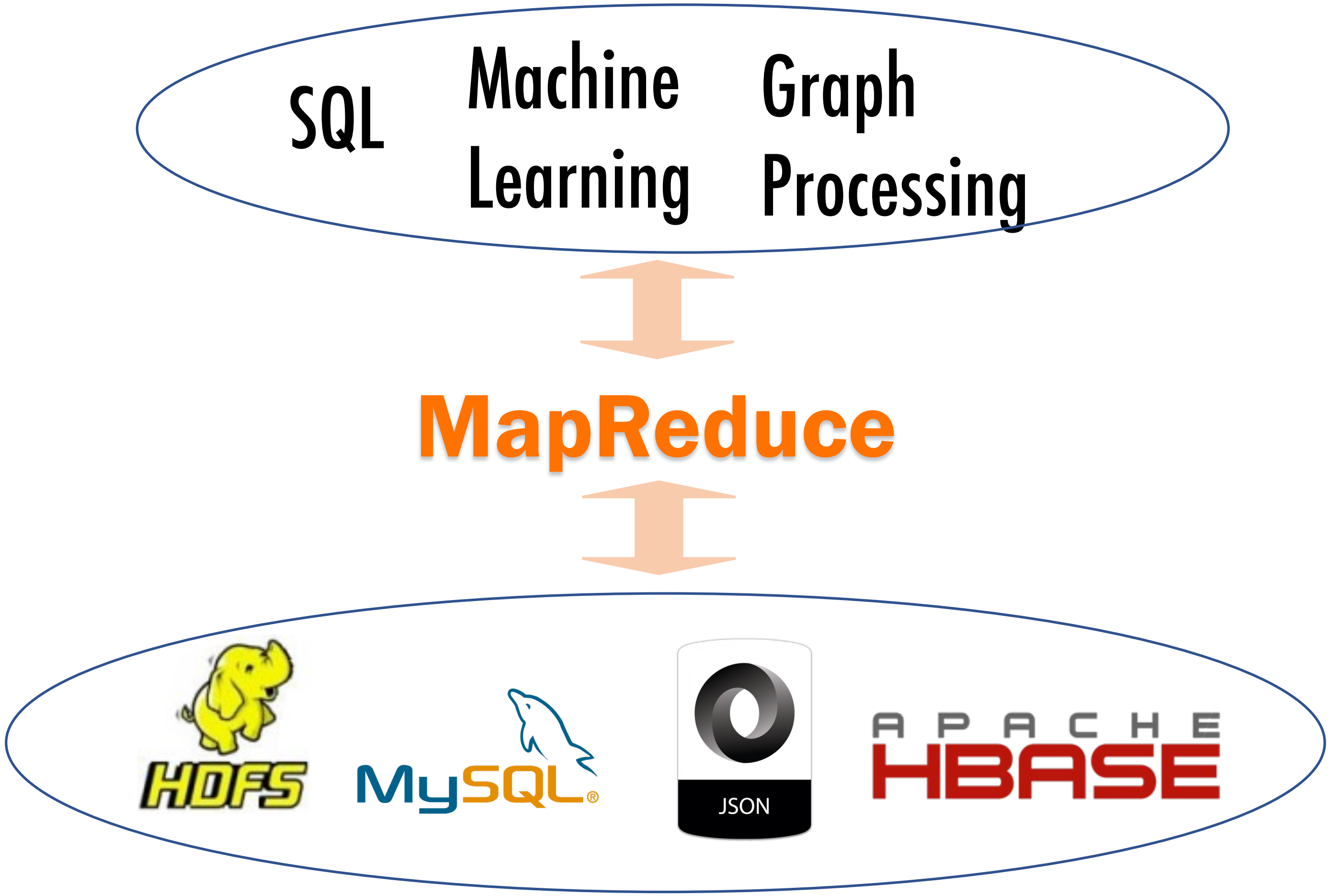
Reserved Instance

Spot Instance

Why MapReduce?

2. Complex Analytics

3. Heterogeneous Storage Systems



SQL

Machine Learning

Graph Processing

MapReduce

MapReduce vs. SQL

Map (k, v) =

- **SELECT** Map(v)
- **FROM** Table

Reduce (k, v_list) =

SELECT Reduce(v)
FROM Table
GROUP BY k

MapReduce: A Major Step Backwards

- 1. MapReduce is a step backwards in database access
- 2. MapReduce is a poor implementation
- 3. MapReduce is not novel
- 4. MapReduce is missing features
- 5. MapReduce is incompatible with the DBMS tools

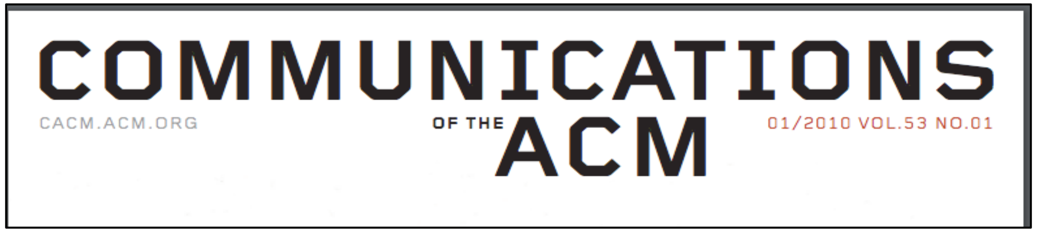
Cannot implement all the queries

Dewitt, D. and Stonebraker, M. *MapReduce: A Major Step Backwards blogpost*; January 17, 2008

Comments From The Other Side

vs.

MapReduce is a program model
rather than a database system

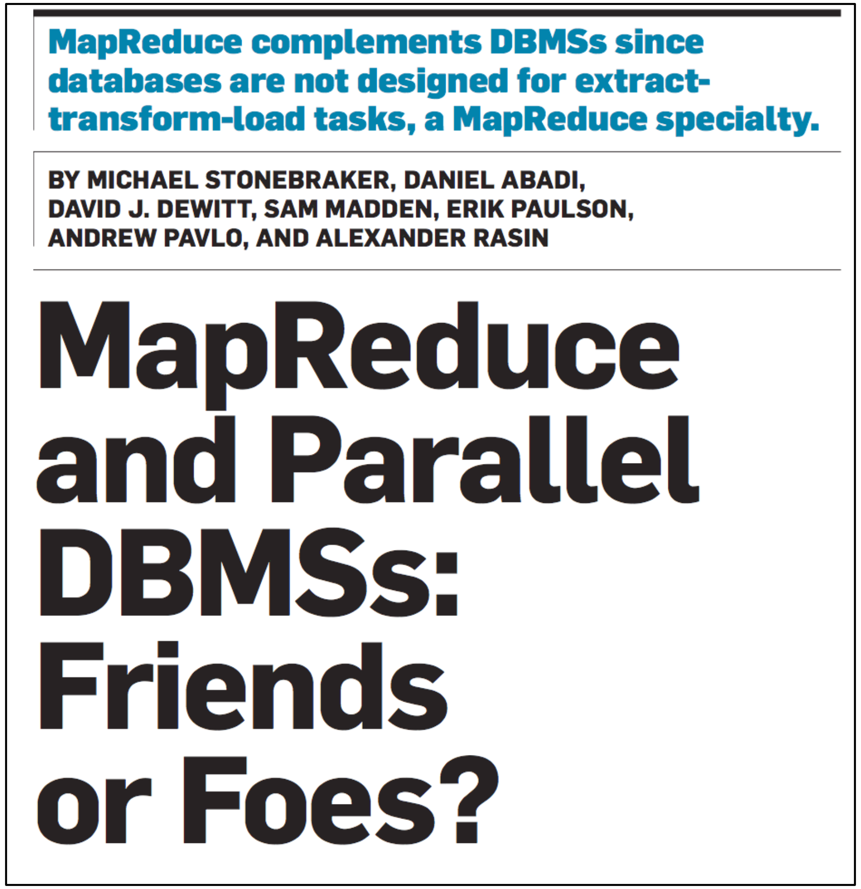


COMMUNICATIONS OF THE ACM

CACM.ACM.ORG

01/2010 VOL.53 NO.01

From Stonebraker et al.

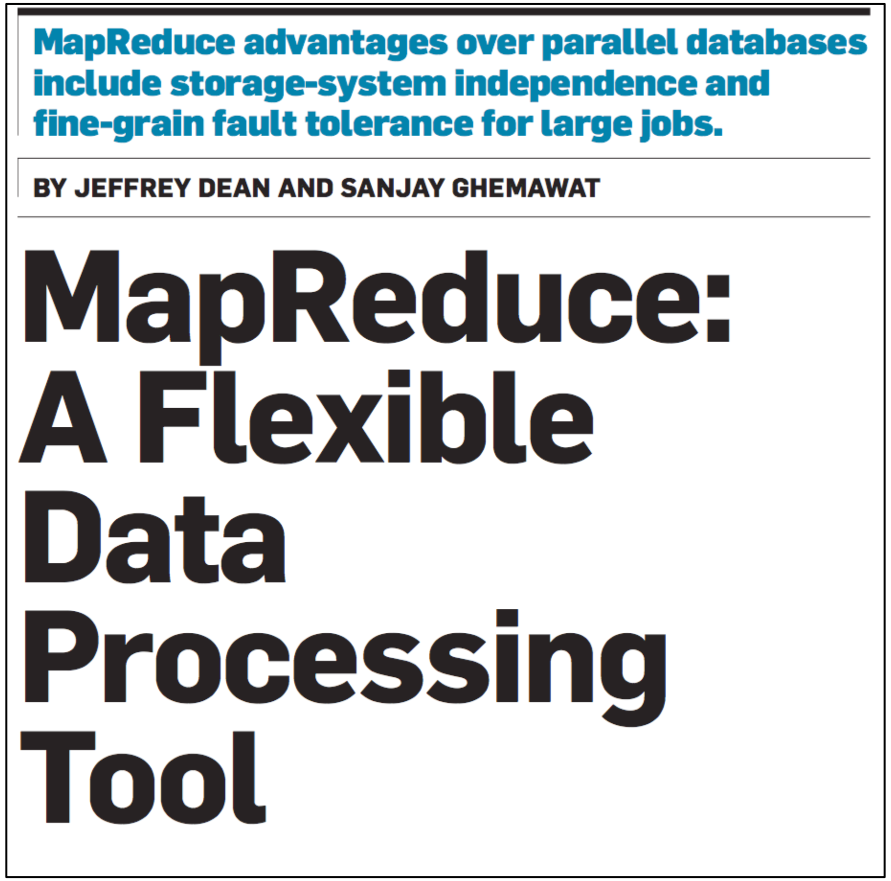


MapReduce complements DBMSs since databases are not designed for extract-transform-load tasks, a MapReduce specialty.

BY MICHAEL STONEBRAKER, DANIEL ABADI, DAVID J. DEWITT, SAM MADDEN, ERIK PAULSON, ANDREW PAVLO, AND ALEXANDER RASIN

MapReduce and Parallel DBMSs: Friends or Foes?

From Dean and Ghemawat



MapReduce advantages over parallel databases include storage-system independence and fine-grain fault tolerance for large jobs.

BY JEFFREY DEAN AND SANJAY GHEMAWAT

MapReduce: A Flexible Data Processing Tool

What They Agree On?

- Advantages of MapReduce:
  1. Fault Tolerant
  2. Complex Analytics
  3. Heterogeneous Storage Systems
  4. No Data Loading Requirement

Both should Learn from Each Other

Who won the debate?

Nobody is writing MapReduce code right now

Many new systems (e.g., Spark, HIVE) were built on MapReduce

Spark stores the intermediate results on the disk

Trends

- 1 Hybrid Transactional and Analytical Processing

- 2. Cloud-Native Databases

- 3. Data Lake

- 4. Lakehouse = Data Lake + Data Warehouse

Summary

- Why Relational Database? (Mid 1970 – Now)
- Why NoSQL? (Mid 2000 – Now)
- Why MapReduce? (Mid 2000 – Now)

What’s Next

- Next class:
  - Relational model & relational algebra
- Decide whether this is the right course for you
- Fill your availability for office hours
  - https://forms.gle/miLhAj56wv8cq7p39
- Sign up Piazza
  - https://piazza.com/sfu.ca/summer2024/cmpt354d100
  - Access code: qc76kdzsz9l
- Check out the course website on Canvas

Acknowledgements

- Some lecture slides were copied from or inspired by the following course materials
  - “CMPT354: Database System I” by Jiannan Wang at Simon Fraser University
  - “CS316: Introduction to Database systems” by Jun Yang at Duke University