

CS 348 Lecture 1

Course Overview & Organization

Semih Salihoglu

Jan 6th, 2022



UNIVERSITY OF
WATERLOO | **DSg** Data
Systems
Group

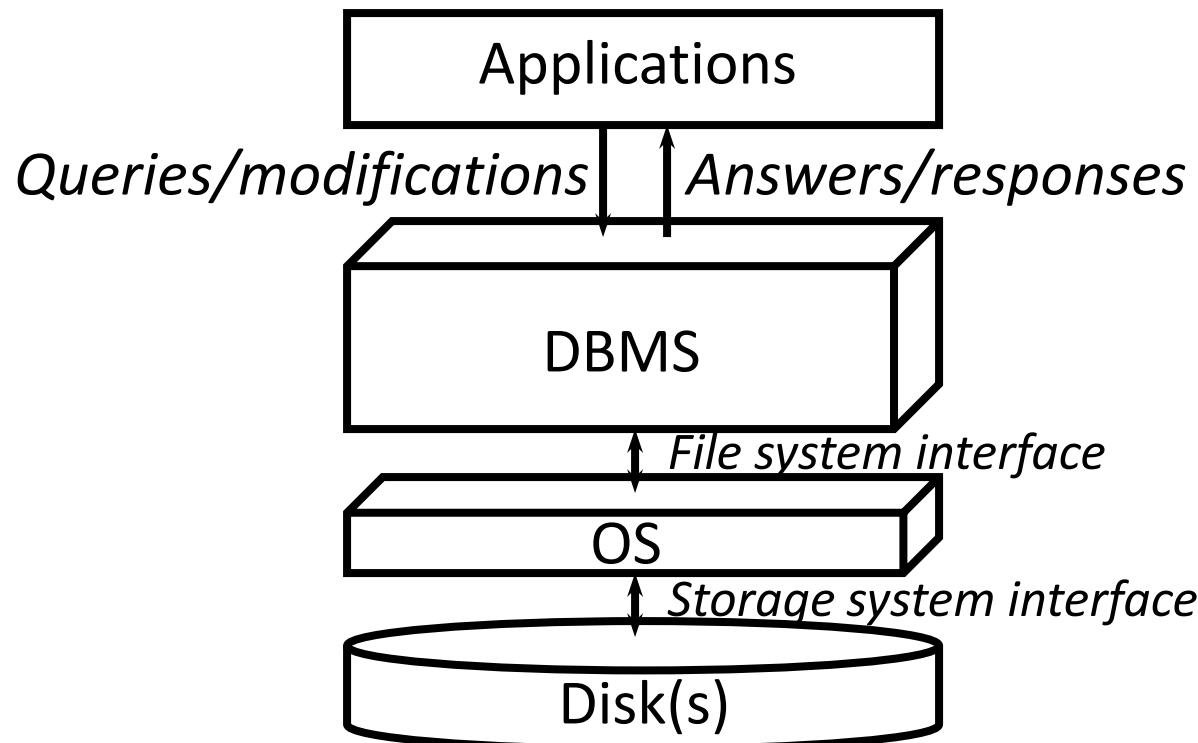
Outline For Today

1. Overview of DBMSs: 3 Major Contributions of the Field
 1. Set of DBMS Features for Applications
 2. Physical Data Independence/High-level Query Languages
 3. Transactions
2. Course Diagram & Administrative Information

Outline For Today

1. Overview of DBMSs: 3 Major Contributions of the Field
 1. Set of DBMS Features for Applications
 2. Physical Data Independence/High-level Query Languages
 3. Transactions
2. Course Diagram & Administrative Information

What is a Database Management System (DBMS)?

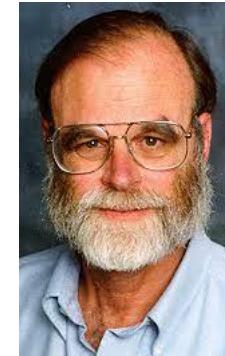
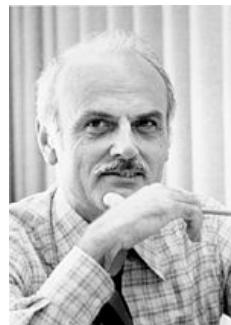


Main Set of DBMS Features

- High-level Data Model and Query Language
- Efficient access and processing of data
- Scalability:
 - Handling of Large Data, i.e., Out-of-memory Data
 - 10-100Ks of concurrent data access/sec
- Safe access and processing of data:
 - Maintenance of the integrity of the data upon updates
 - Multi-User access to data (Concurrency)
 - Fault tolerant storage of data

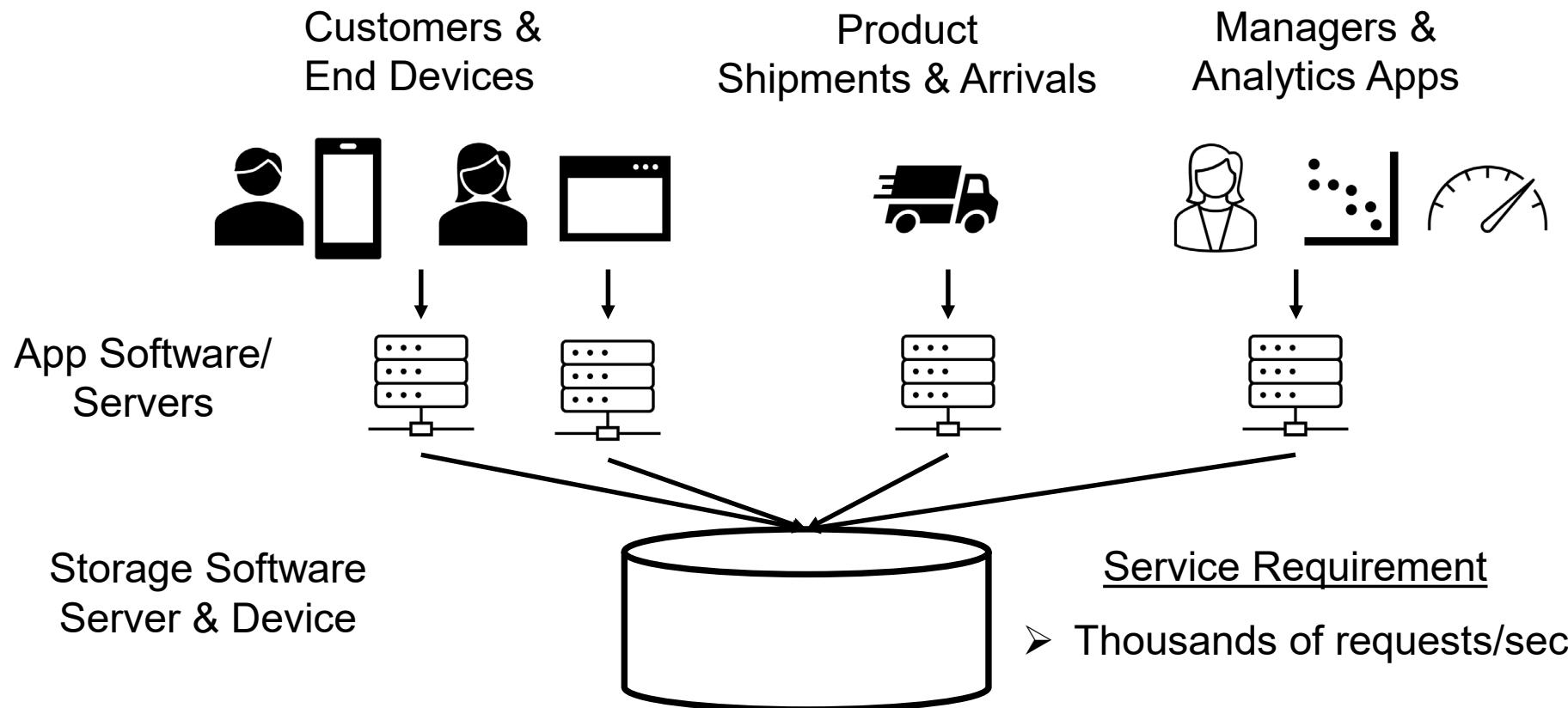
Main Contributions of the Field?

1. The System (Charles Bachmann)
2. High-level/Declarative Programming (Ted Codd)
 - Ingredients: Relational Data Model & Algebra: A model based on set theory (so a formal mathematical theory)
 - Provides ability to generate automatic efficient algorithms for many data processing tasks
3. Transactions (Jim Gray)
 - Principles of concurrent data-manipulating app. development



Why App Developers Need a DBMS?

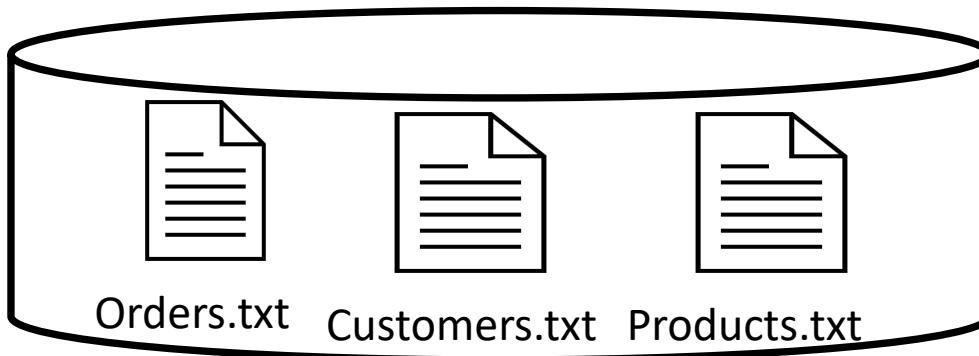
- Application: Order & Inventory Management in E-commerce
- E.g.: Amazon or Alibaba



Let's simplify the design: assume a single server will accept requests from app software to keep track of and serve your records: orders, new products, etc.

Bad Idea: Write Storage Software in Java/C++

- Possible Approach: Directly use the file system of the OS.
 - E.g: one or more files for orders, customers, products etc.



- Problem: Physical Record Design?
 - Suppose you record: name, birthdate for each customer
 - How many bytes for each fact?
 - E.g.: Encoding of string names? Fixed or variable length?
 - Many sub-problems: E.g.: How to quickly find a record?

PR1: Example Physical Record Designs (1)

➤ Variable-length design

name-len (bytes)		name payload		birthdate (fixed 4 bytes)	
11	Alice Smith	2001/09/08		19	Alexander Desdemona
6	Ali Jo	1992/02/25		26	Montgomery Cambridgeshire
...

Customers.txt

➤ Fixed-length design

Overflow	ptr	len	name (16 byte)	birthdate (4 bytes)
null	11		Alice Smith -----	2001/09/08
0	19		Alexander Desdem	2002/05/20
null	19		Ali Jo -----	1992/02/25
...

Customers.txt

The diagram shows a curved arrow originating from the 'name (16 byte)' field in the first row of the 'Customers.txt' table and pointing towards the 'Customer-overflow.txt' table. This indicates that the overflowed names are stored in a separate file.

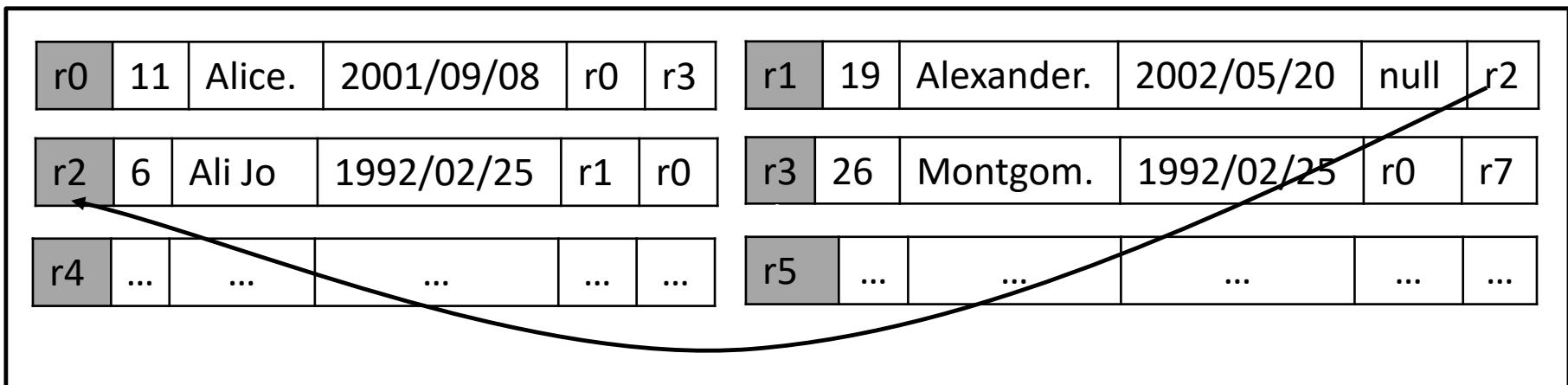
ona	idgeshire
...

Customer-overflow.txt

PR1: Example Physical Record Designs (2)

- Chained Design: Maybe to keep in sorted alphabetical order

name-leng (bytes)	name payload	birthdate (fixes 4 bytes)	prev ptr	next ptr
-------------------	--------------	---------------------------	----------	----------



Customers.txt

Takeaway 1: Many designs options & difficult for app developers!

Takeaway 2: Bytes not the right data abstraction to program apps.

PR2: Efficient Query/Analytics Algorithms

- Managers Ask: Who are top paying customers?
 - Task: Compute total sales by customer
 - Assume in record layout every field is fixed length
 - **Problem: App developer needs to implement an algorithm.**

Possible Algorithm 1:

```
file = open("Orders.txt")
HashTable ht;
for each line in file:
// some code to parse custID and price
if (ht.contains(custID))
    ht.put(custID, ht.get(custID) + price)
else: ht.put(custID, price);
file.close();
```

O1	Cust1	BookA	\$20
O2	Cust2	WatchA	\$120
O3	Cust1	DiapersB	\$30
O4	Cust3	MasksA	\$15
...
...

Possible High-level Algorithm 2:

sort Orders.txt on CustID

orders of $Cust_i$ are now consecutive *Which sorting algorithm to use?*
read sorted records sequentially → *Should one parallelize sorting? How?*
and sum prices for each C_i

Orders.txt

PR2: Efficient Query/Analytics Algorithms

- That is only for 1 question. There will be many questions:
 - List of Orders that bought a product that cost > \$500
 - Last Order from Cust4?
 - Who are closest co-purchasers of Cust4? (i.e., who bought the same item as Cust4, ordered by the #co-purchases.)
 - Many many more (thousands) important business questions:
 - For each question numerous possible algorithms and implementations.

Takeaway 1: Many algs & implementations. Difficult to choose.

Takeaway 2: Writing an algorithm for each task won't scale!

PR3: Scalability

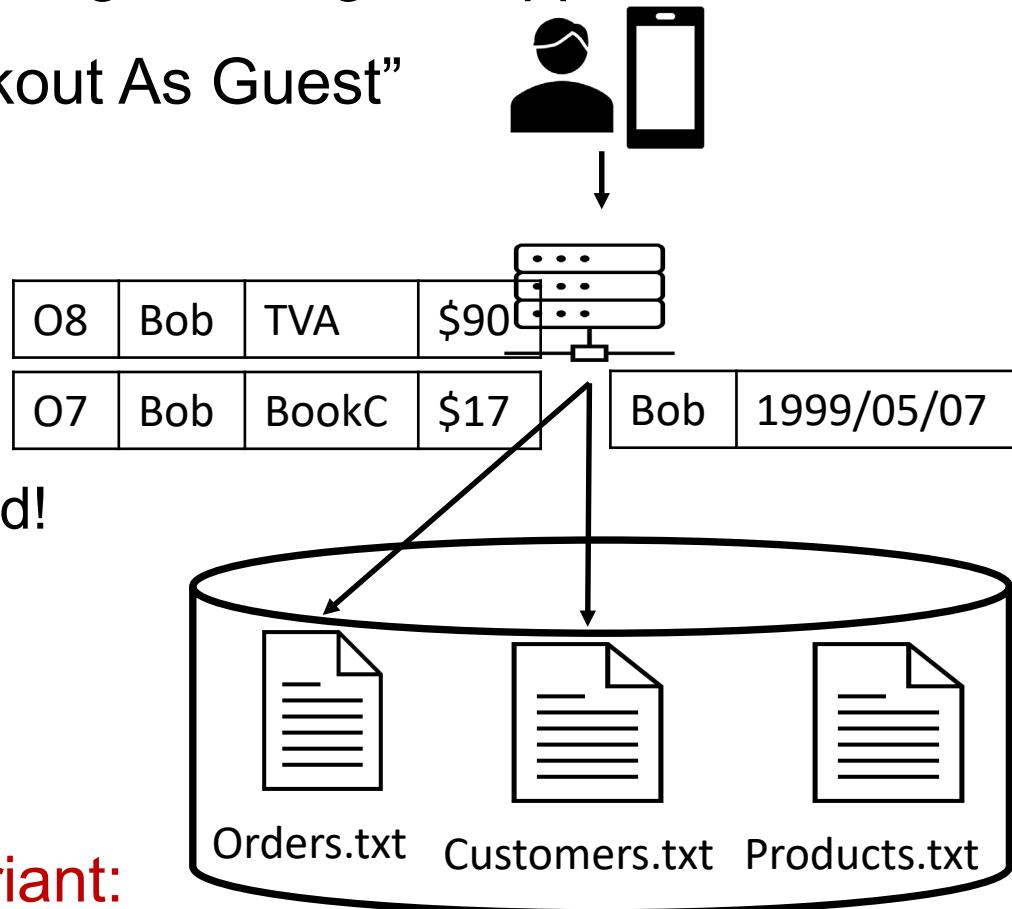
- Large-scale Data: Data > Memory
 - E.g. Orders.txt grows to terabytes & does not fit in memory.
 - Often the case for data-intensive applications
 - Need ``External'' algorithms, i.e., uses disk to scale
 - Hard to write such algorithms. Challenge:
 - *Try implementing a good external sorting algorithm?*
- Scale to: 10K~100Ks of requests/sec
 - Hard to write code that efficiently supports such workloads.

Takeaway: Hard and have nothing to do w/ the app logic!

App developers should focus on the app!

PR4: Integrity/Consistency of The Data (1)

- Many ways data can be corrupted:
 - Often: Wrong application logic or bugs in application
- E.g: Checkout App's “Checkout As Guest”
 - Writes the Order record
 - And the Customer record
 - Assume Bob shops again
 - (Bob, 1999/05/07) is duplicated!



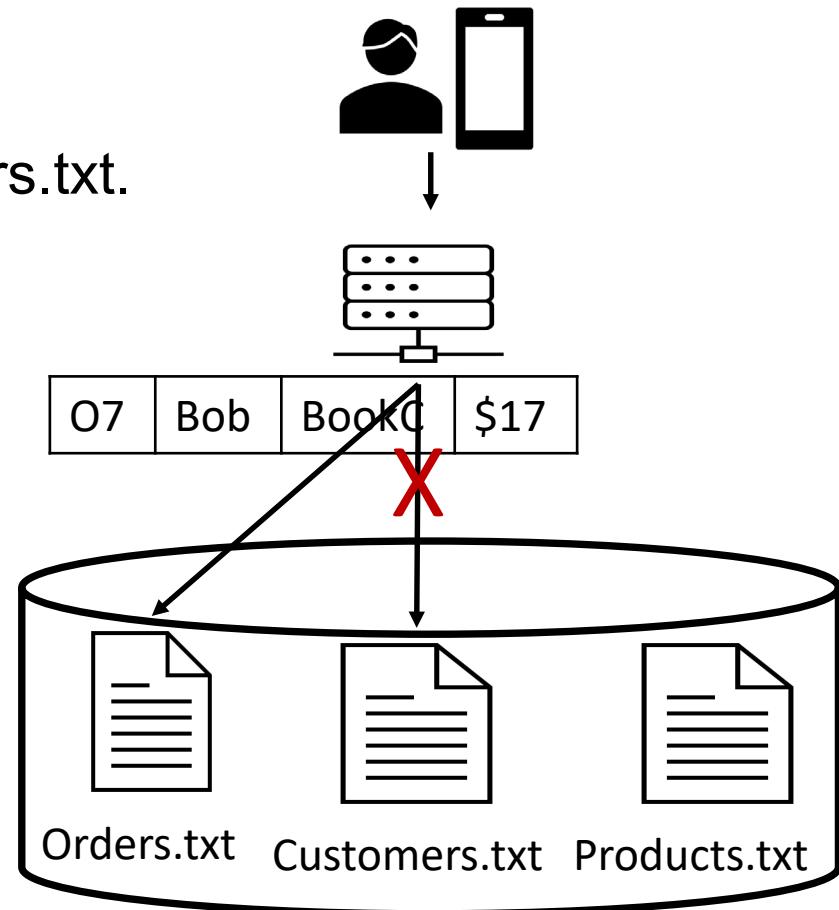
Likely an inconsistency.

We'd want to enforce the invariant:

No duplicate cust records!

PR4: Integrity/Consistency of The Data (2)

- E.g: Checkout App's “Checkout As Guest”
 - Writes the Order record
 - But not the Customer record
 - (Bob, 1999/05/07) is not in Customers.txt.



Likely an inconsistency.

We'd want to enforce the invariant:

Every order's cust record exists!

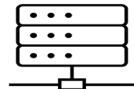
- Another example momentarily in concurrency

PR5: Concurrency: Multiple Conflicting Requests

- Alice & Bob concurrently order BookA: suppose 1 left in stock.

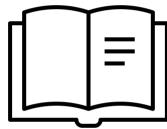
Product	NumInStock
...	...
BookA	1
...	...

```
Buy_Product_Subroutine(string prodName):  
    (prod, numInStock) = readProduct(prodName)  
    if (numInStock > 0):  
        writeProduct((prod, numInStock - 1))  
    else throw("Cannot buy product!");
```



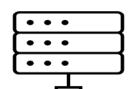
r: (A, 1)

w: (A, 0)



r: (A, 0)

No Book



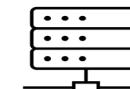
r: (A, 1)

w: (A, 0)



r: (A, 0)

No Book

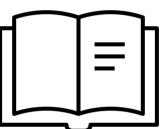
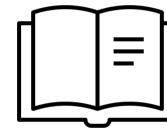


r: (A, 1)

r: (A, 1)

w: (A, 0)

w: (A, 0)



time

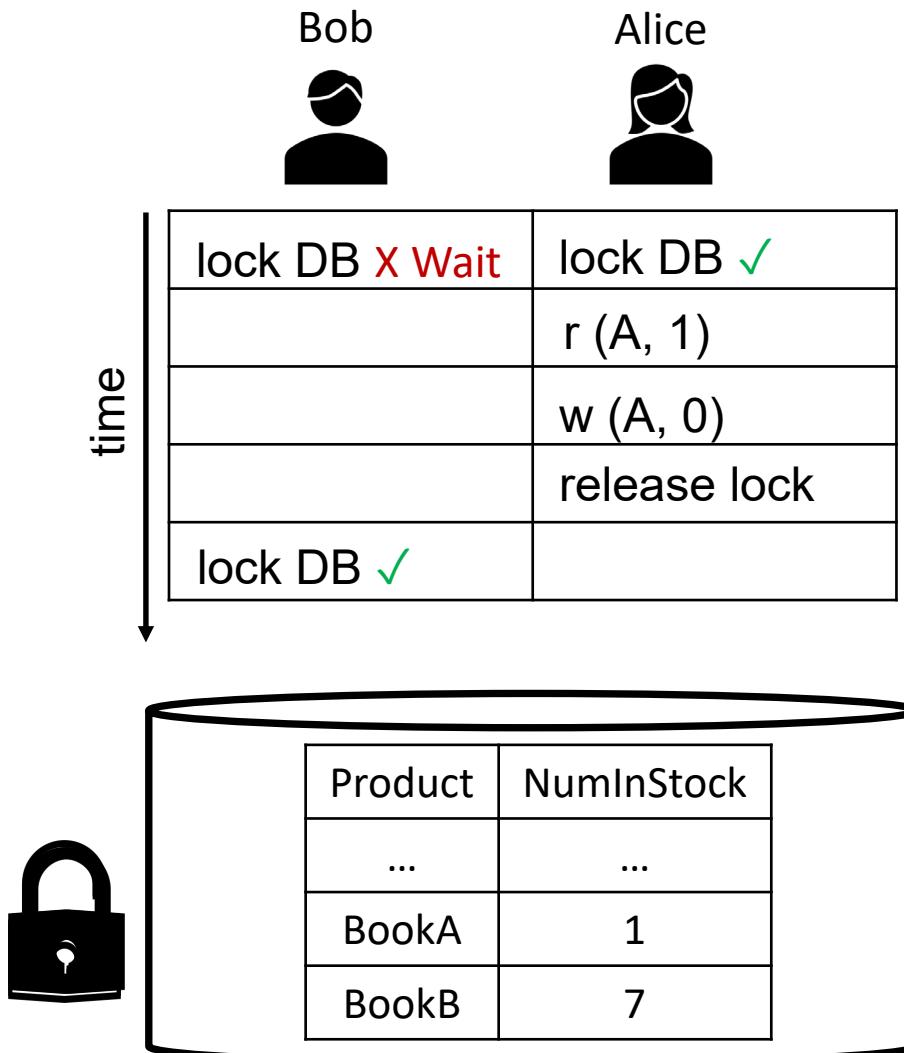


Concurrency Questions

- What is a correct/incorrect state upon concurrent updates?
 - Theoretical formalism to explain these states: Serializability
- What protocols/algorithms can ensure a correct state?
 - Locking-based protocols
 - Pessimistic: set of lock acquisitions to prevent bad state
 - Optimistic protocols
 - Detect bad state and recover from it
- Set of guarantees that a DBMS should satisfy
 - ACID guarantees: atomicity, consistency, isolation, durability

Concurrency Avoidance Ex: Global DB Lock

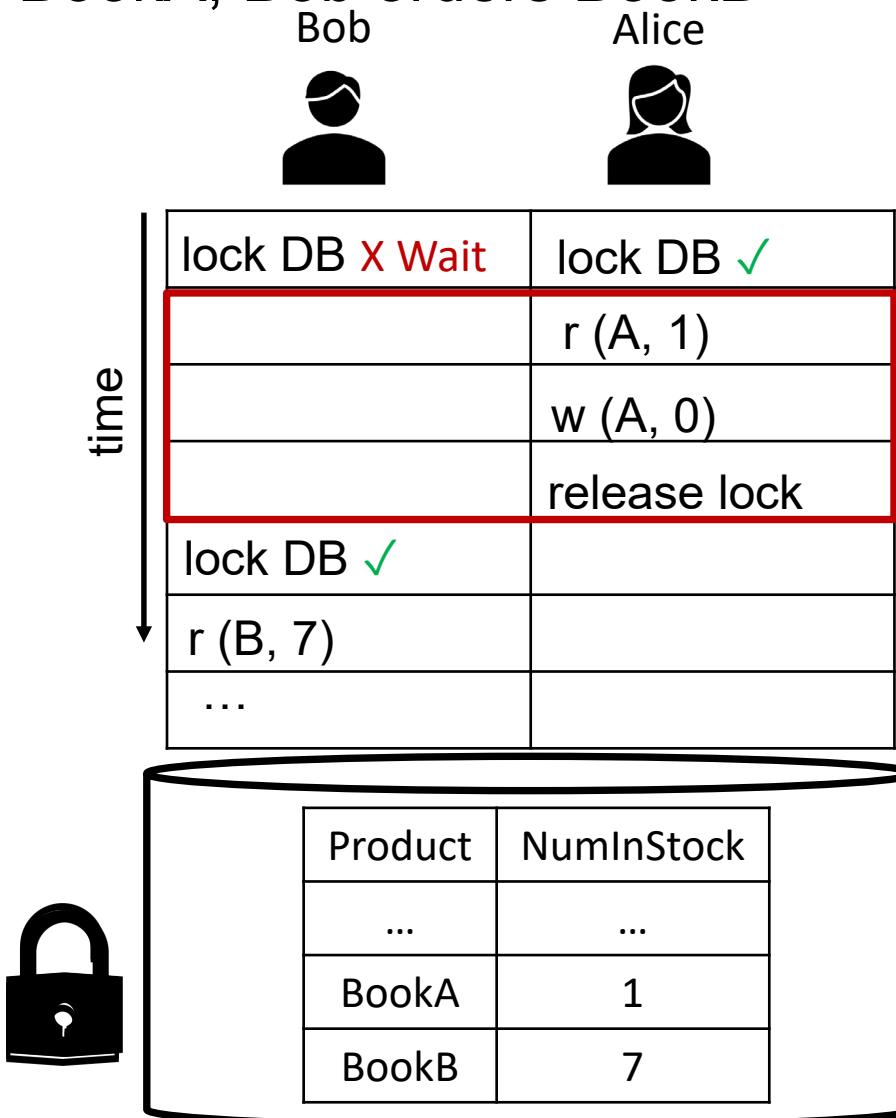
- Alice and Bob order BookA



Safe but inefficient. Why?

Concurrency Avoidance Ex: Global DB Lock

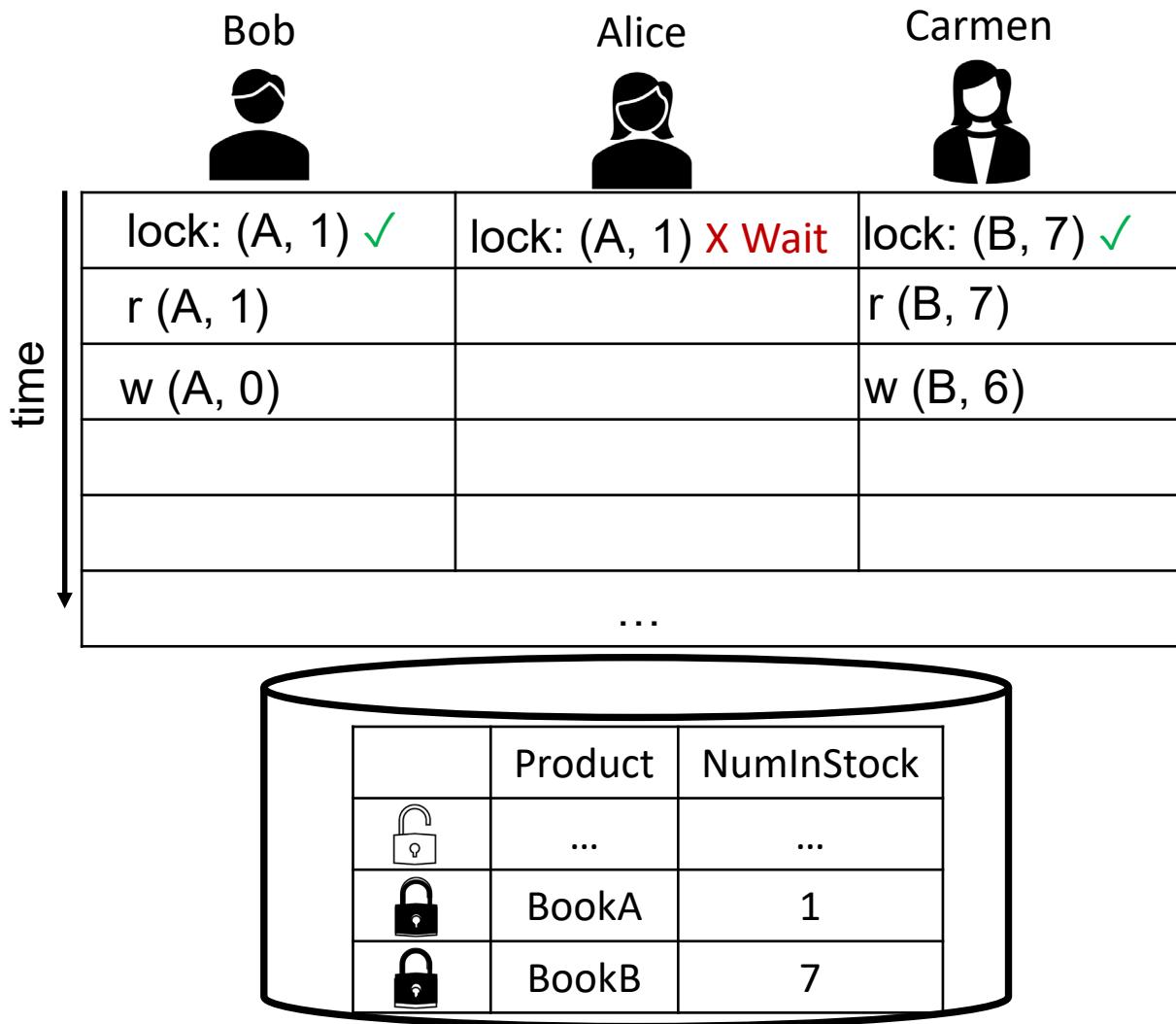
- Alice orders BookA, Bob orders BookB



Bob had no conflicts; so was “unnecessarily” blocked.

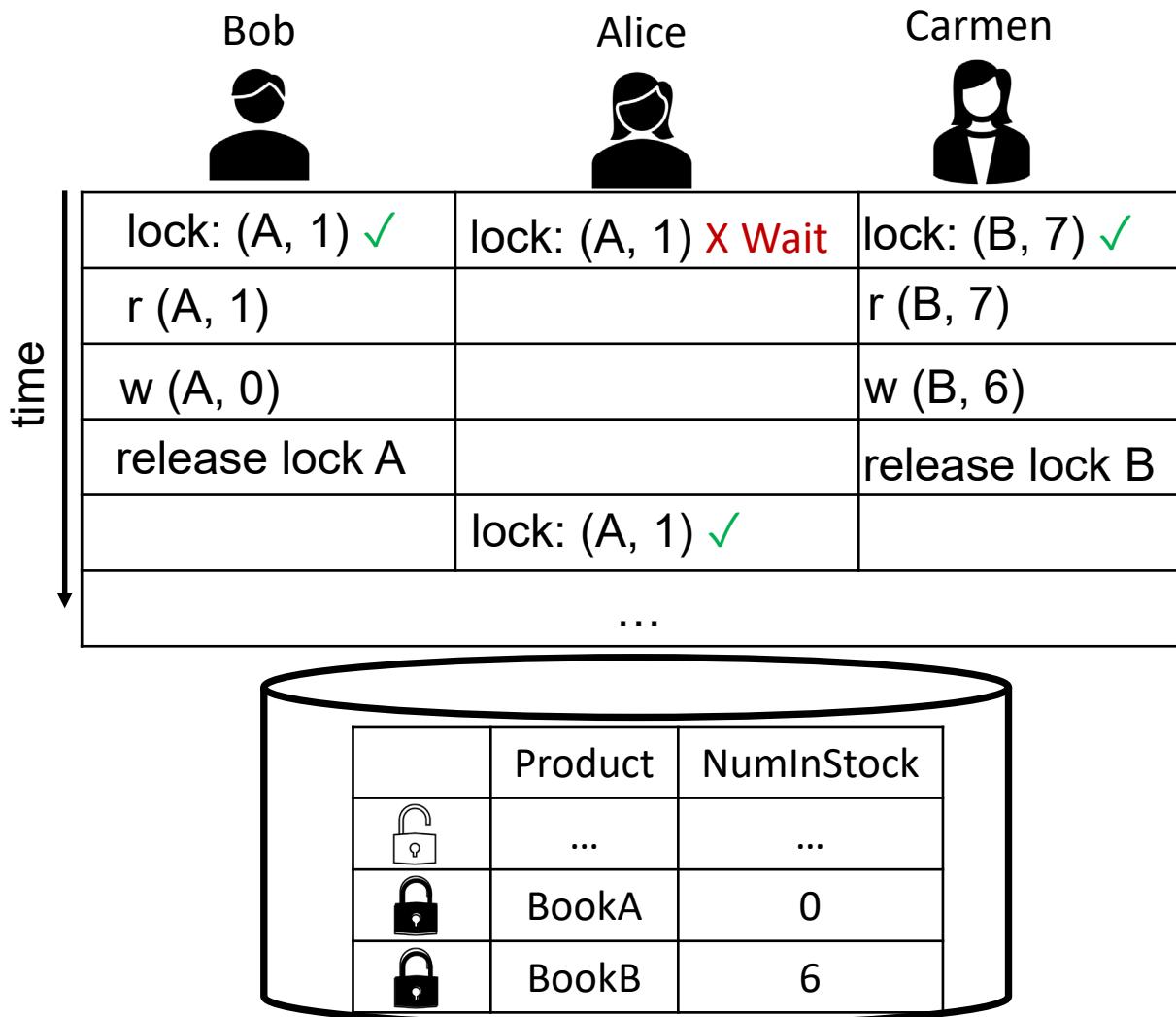
Concurrency Avoidance Ex: Record-level Lock

- Alice, Bob as before want BookA, Carmen orders Book B



Concurrency Avoidance Ex: Record-level Lock

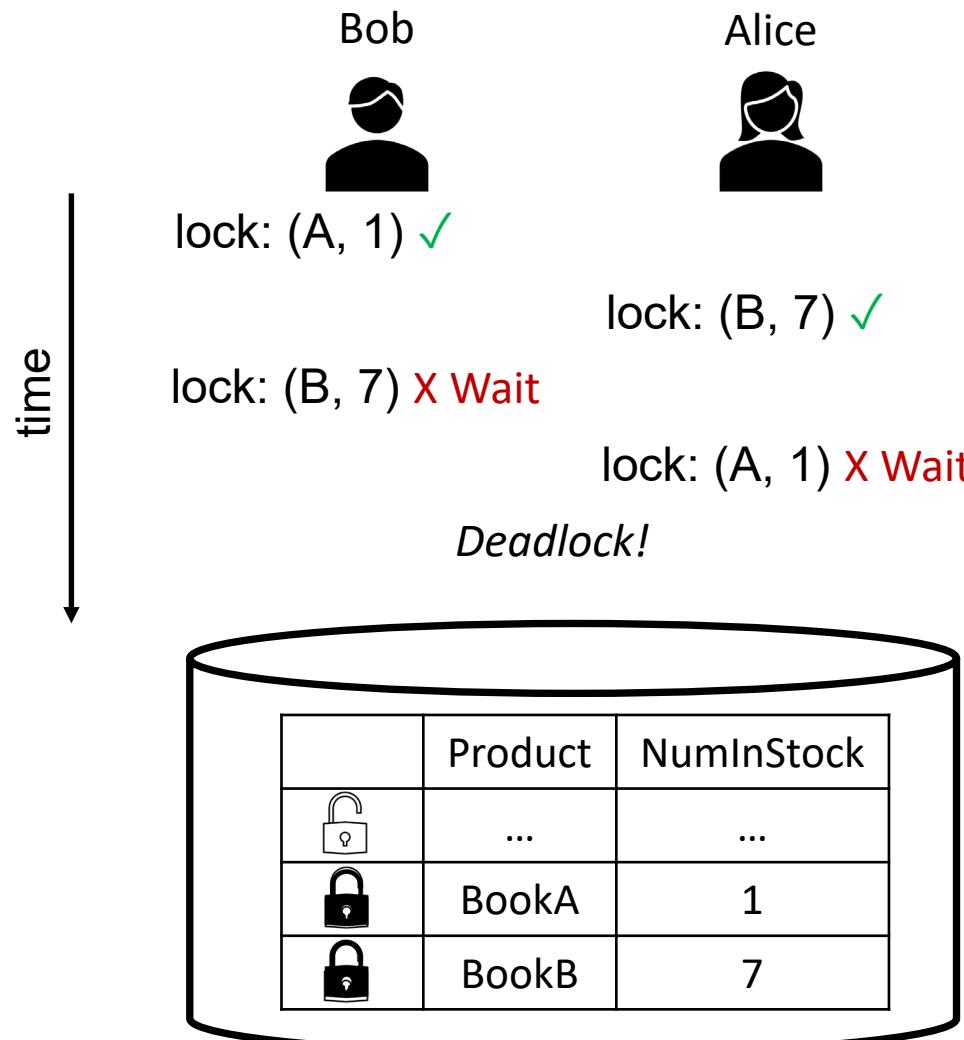
- Alice, Bob as before want BookA, Carmen orders Book B



Safe and achieves parallelism. What can go wrong?

Where There is Locking, There is Deadlocks!

- Alice, Bob both order both BookA and BookB together



How can we detect & avoid deadlocks?

Failure & Recovery

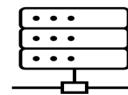
- What if your disk fails in the middle of an order?
- What if your server software fails due to a bug?
- What if there is a power outage in the machine storing files?



Product	NumInStock
...	...
BookA	1
BookB	7

Failure & Recovery

- What if your disk fails in the middle of an order?
- What if your server software fails due to a bug?
- What if there is a power outage in the machine storing files?
- Suppose Alice orders both BookA and BookB



w (A, 0)

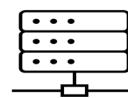
Product	NumInStock
...	...
BookA	1
BookB	7

Failure & Recovery

- What if your disk fails in the middle of an order?
- What if your server software fails due to a bug?
- What if there is a power outage in the machine storing files?
- Suppose Alice orders both BookA and BookB

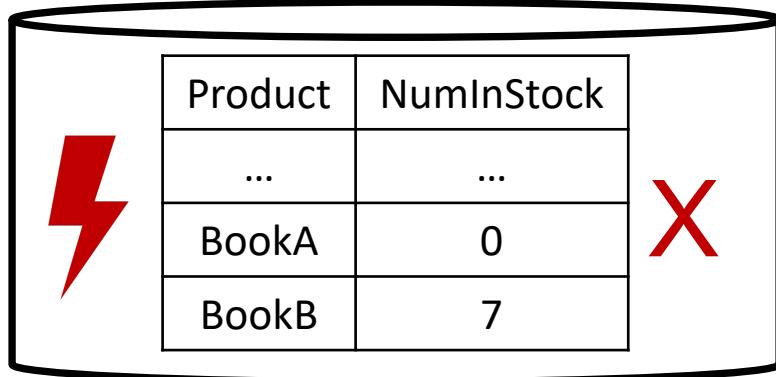


*Before (B, 6) is written failure!
Inconsistent data state!*



w (A, 0)

PR: How to recover from inconsistent state?



A large cylinder icon representing a hard disk drive, containing a red lightning bolt symbol.

Product	NumInStock
...	...
BookA	0
BookB	7

X

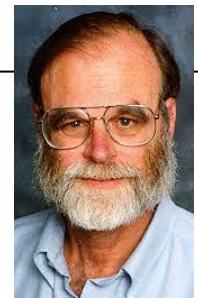


A green checkmark icon positioned above the second table.

Product	NumInStock
...	...
BookA	0
BookB	6

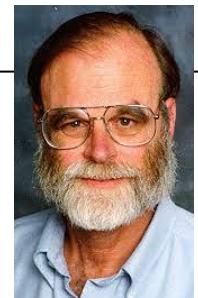
Contributions of DBMSs To Computing

- DBMSs provide solutions to all of the problems we identified!
- Allows app developers to focus on the application logic.

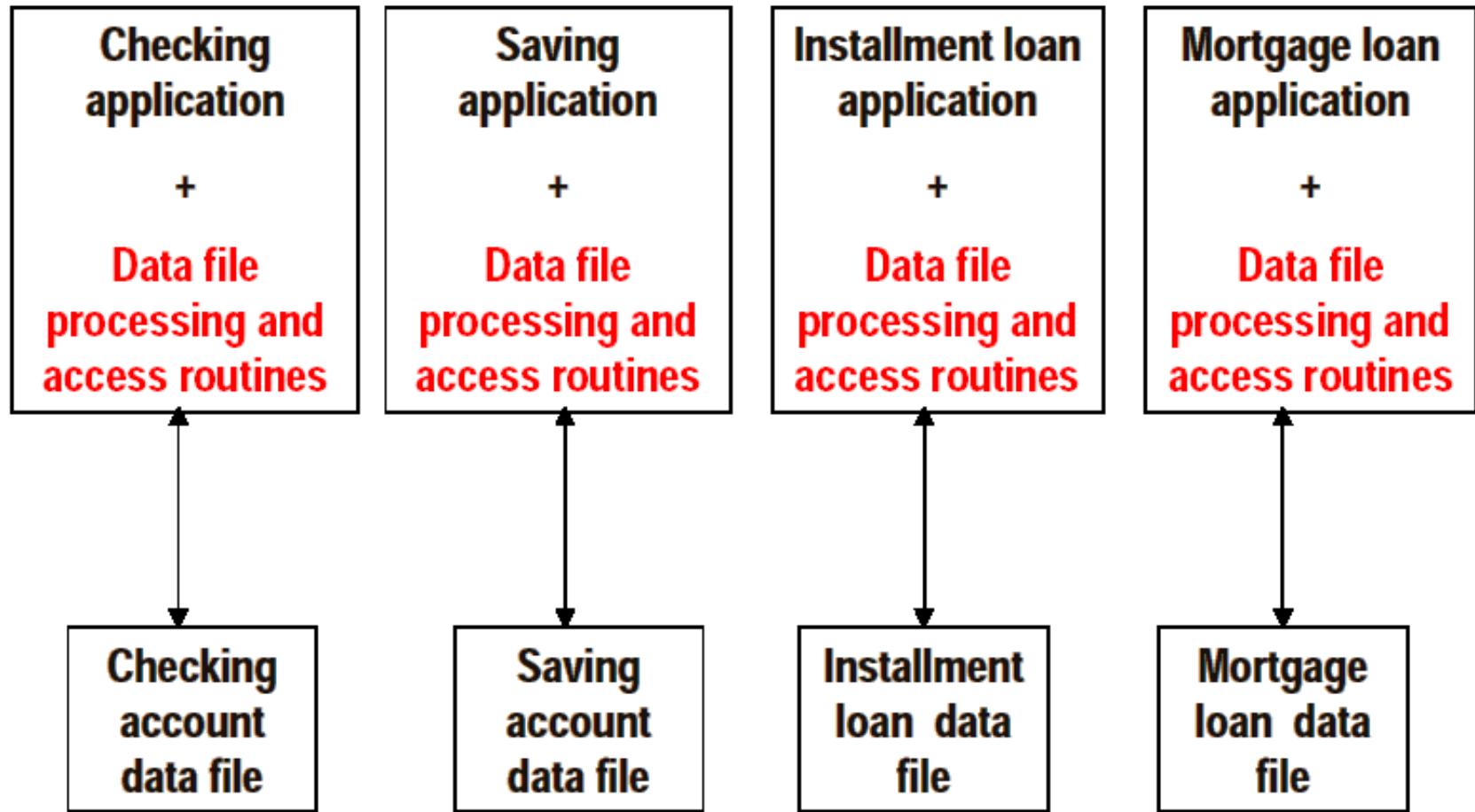
Problems	Solutions	Contribution 2
1. Physical record design and access to records	Data Model (Higher-level than bits/bytes)	
2. Efficiency	High Level Data Query/Manipulation Language Automatic compilation of queries to efficient algs/query plans	
3. Scalability: 3.1: Large-scale data 3.2: Large # of requests	Persistent-disk-based data Scale to 10-100K requests/seconds	Contribution 3
4. Safe Concurrency	Transactions & ACID guarantees	
5. Other Safety Features:	Data Integrity and Failure Recovery	

Contribution 1: The System

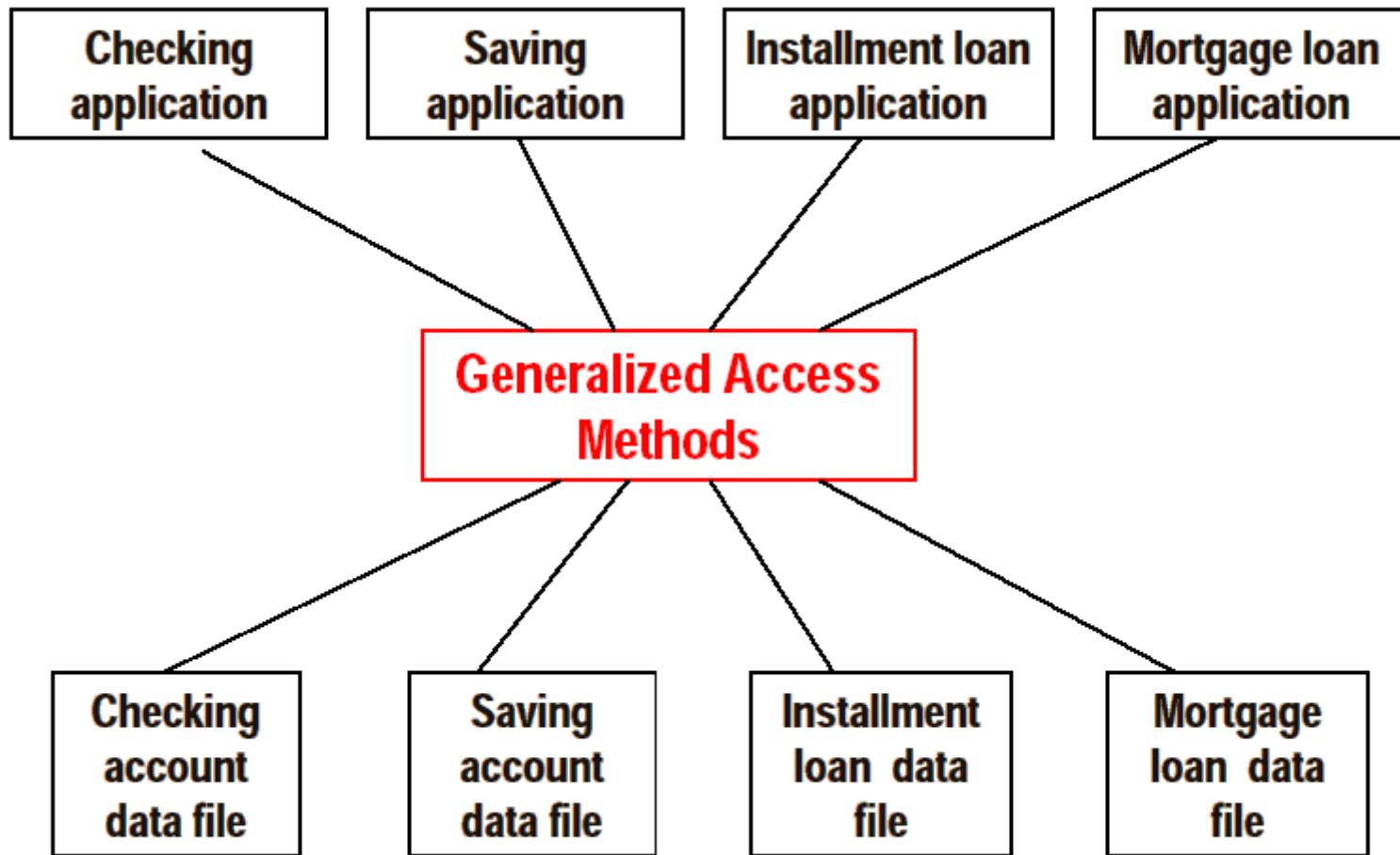
- IDS (1960s): First DBMS
 - Had a data model and a primitive “query” language
 - Had scalability for its era and integrity and recovery
 - No transactions



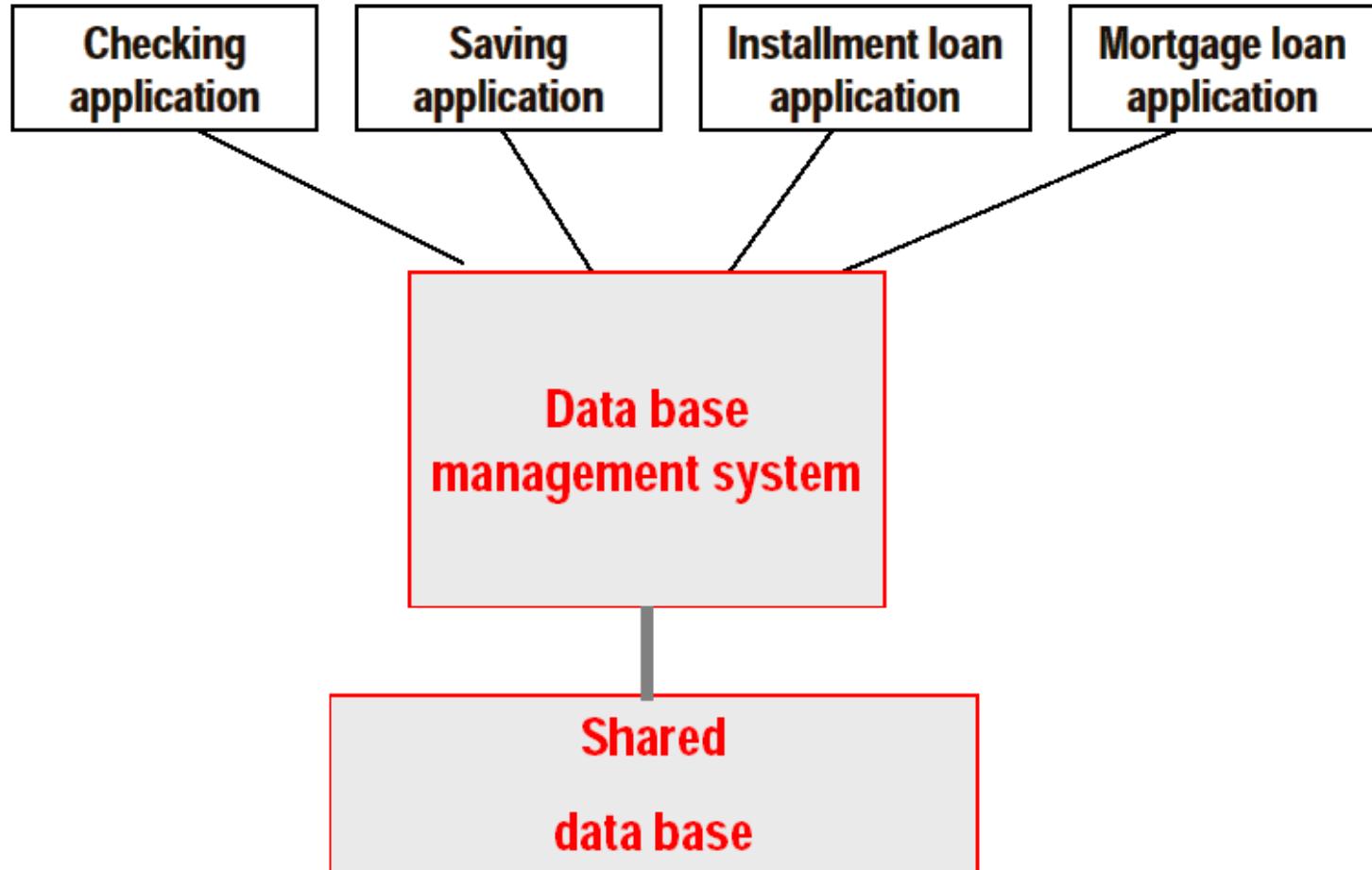
The Birth of DBMS (1)



The Birth of DBMS (2)



The Birth of DBMS (3)



DBMS is an excellent example of a successful abstraction!

A Side Note on Spotting an Opportunity For New Systems or System Components

- Sometimes (but not always) you spot that a new system/system component is needed by observing functionality duplication.
- Ex 1: Map Reduce Large-Scale Dataflow System
 - CS 451: Data-Intensive Distributed Computing

Over the past five years, the authors and many others at Google have implemented hundreds of special-purpose computations that process large amounts of raw data, such as crawled documents, web request logs, etc., to compute various kinds of derived data, such as inverted indices, various representations of the graph structure of web documents, summaries of the number of pages crawled per host, the set of most frequent queries in a given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat
jeff@google.com, sanjay@google.com
Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key-value pair to produce zero or more intermediate pairs, and a reduce function that merges all intermediate values associated with the same key. This allows programmers without any experience in distributed systems to express complex computations on large distributed systems.

Our implementation of MapReduce runs on a large cluster of commodity hardware. It is designed to support a typical MapReduce computation processes many terabytes of data in a reasonable amount of time. It makes it easy to find the system easy to use. Hundreds of MapReduce programs have been implemented and specialized for the same MapReduce jobs are executed on Google's clusters every day.

The major contribution of this work are a simple and powerful interface that supports common computations with an implementation of this interface that achieves high performance on a cluster of commodity PCs.

Section 2 describes the basic programming model and distributed execution environment. Section 3 describes our implementation of the MapReduce interface tailored towards our cluster-based computing environment. Section 4 describes the MapReduce interface and its applications. Section 5 has performance analysis of the system. Section 6 explores the use of MapReduce within Google including our experiences in using it as the basis

To appear in OSDI 2004

The Birth of MapReduce (1)

Google Inverted Index
Constructor

Code for:
Computation parallelization
across a cluster of machines
Distributing data files,
Cluster failure recovery

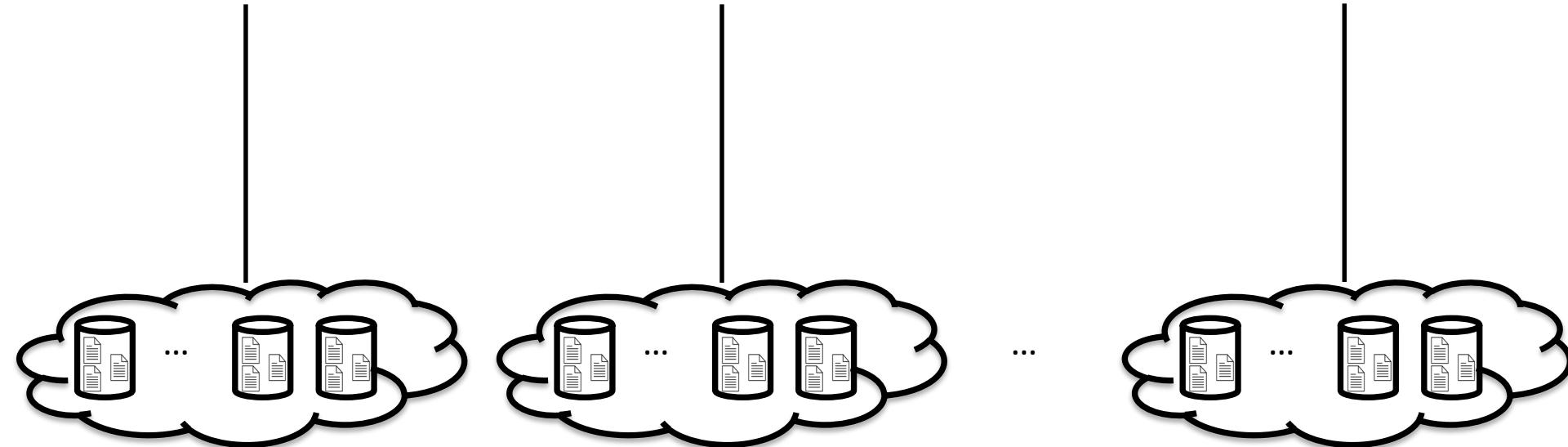
Google PageRank
Computation

Code for:
Computation parallelization
across a cluster of machines
Distributing data files,
Cluster failure recovery

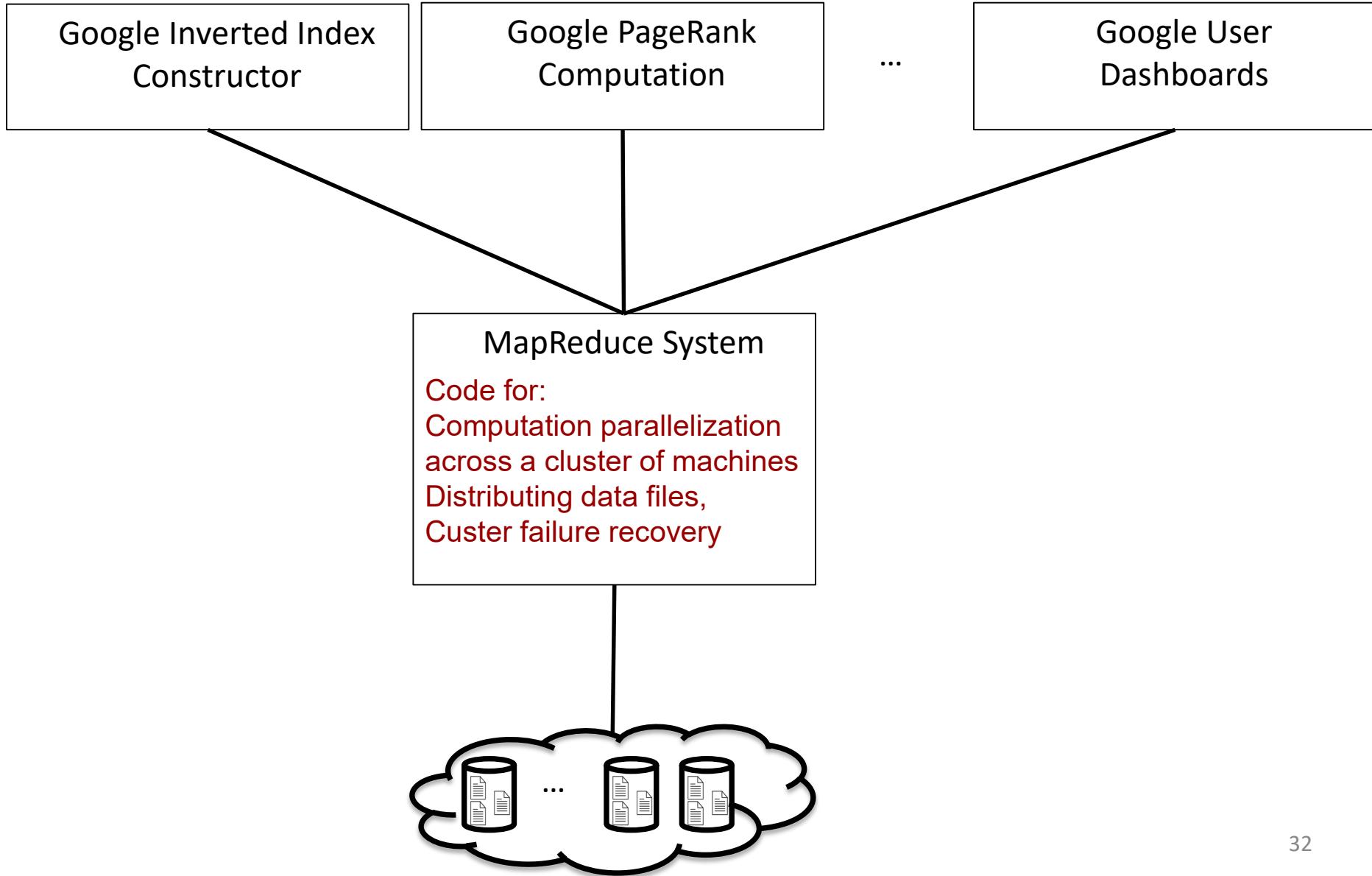
...

Google User
Dashboards

Code for:
Computation parallelization
across a cluster of machines
Distributing data files,
Cluster failure recovery

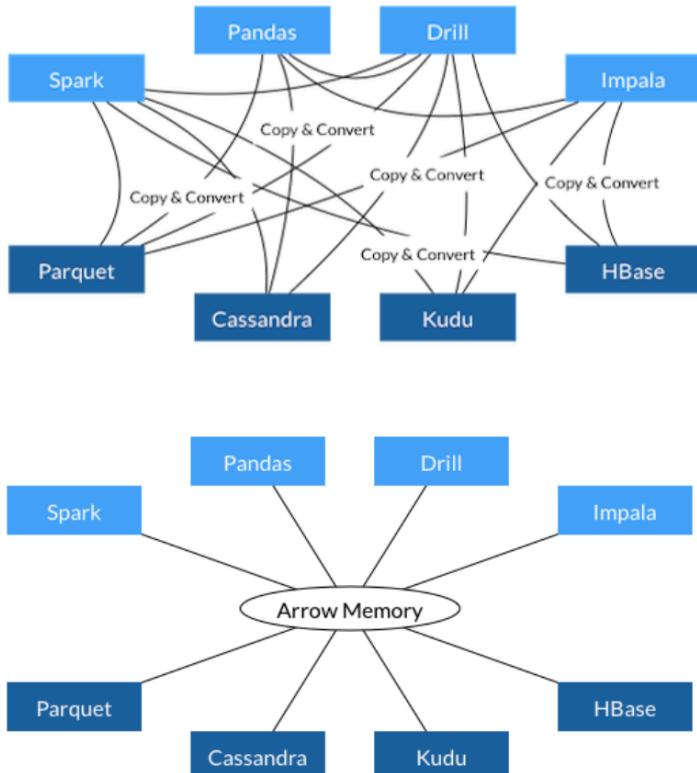


The Birth of MapReduce (2)



Another Recent Example: [Apache Arrow](#)

➤ A New Columnar Data Format and Library for Data Science



Standardization Saves

Without a standard columnar data format, every database and language has to implement its own internal data format. This generates a lot of waste. Moving data from one system to another involves costly serialization and deserialization. In addition, common algorithms must often be rewritten for each data format.

Arrow's in-memory columnar data format is an out-of-the-box solution to these problems. Systems that use or support Arrow can transfer data between them at little-to-no cost. Moreover, they don't need to implement custom connectors for every other system. On top of these savings, a standardized memory format facilitates reuse of libraries of algorithms, even across languages.

Same Application Development W/ a DBMS

- We will use a Relational DBMS (RDBMS) but can use other DBMSs too (*e.g., a graph database management system*)
 - Ex: PostgreSQL, Oracle, MySQL, SAP HANA, Snowflake...

Data Modeling With an RDBMS (1)

- Relational Model: Data is modeled as a set of tables
 - Much higher-level abstraction than bits/bytes

Customers		Orders				Products	
<u>name</u>	<u>birthday</u>	<u>old</u>	<u>cust</u>	<u>product</u>	<u>price</u>	<u>product</u>	<u>numInStock</u>
Alice	2001/09/08	O1	2001/09/08	BookA	20	BookA	1
Bob	2002/05/20	O2	2002/05/20	TVB	100	TVB	78
...

Example SQL Command in an RDBMS:

```
CREATE TABLE Customers
    name varchar(255),
    birthdate DATE;
```

- The RDBMS takes care of physical record design: Fixed-length/var-length, columnar, row, chained etc.
- The physical record design is transparent to the developer, i.e. the developer does not need to know the design.

Data Modeling With an RDBMS (2)

- Physical Data Independence:

- Throughout the lifetime of the app, the RDBMS can change the physical layout for performance or other reasons and the applications keep working because the design is transparent.

- E.g:

- A new column can be added that changes the record design
 - A compressed column can be uncompressed

Takeaway: A high-level data model delegates the responsibility of physical record design and access to these records to the DBMS

High-level Query Language (1)

- Structured Query Language (SQL)
- SQL is so high-level that it's called a *declarative* language: i.e., one in which you can describe the output of the computation *but not how to perform the computation.*
- Recall managers' question: Who are top paying customers?

```
SELECT cust, sum(price) as sumPay  
FROM Orders  
ORDER BY sumPay DESC
```

Orders			
<u>oID</u>	<u>cust</u>	<u>product</u>	<u>price</u>

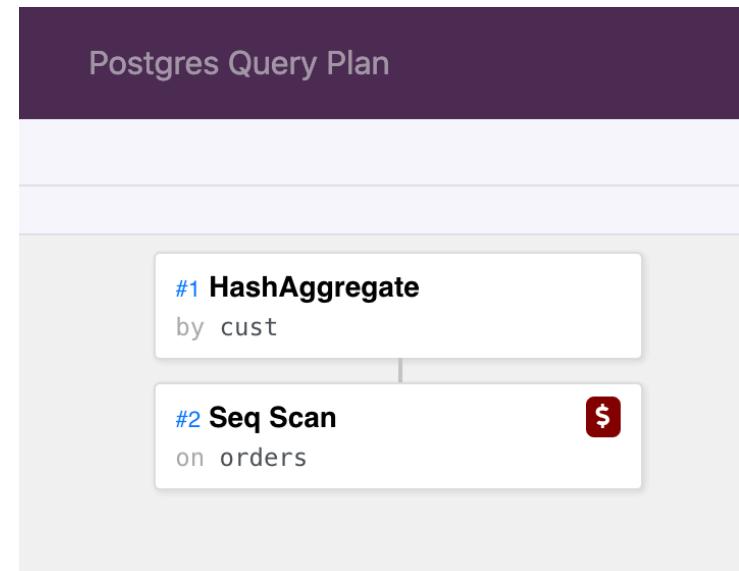
- No procedural description of how to group-by and aggregate: hash-based, sort-based, what sorting algorithm to use etc.

High-level Query Language (2)

- RDBMS automatically generates an algorithm for the query:
 - We call those algorithms *query plans*

```
SELECT cust, sum(price) as sumPay  
FROM Orders  
ORDER BY sumPay DESC
```

- High-level QLs are perhaps the best examples of *automatic programming*



Takeaway: A high-level QL delegates the responsibility of finding an efficient algorithm for queries to the DBMS.

Other efficiency benefits: The DBMS will handle large data and automatically parallelize these algorithms.

Integrity Constraints

- Recall the bug in Checkout App's "Checkout As Guest":
 - Writes the Customer record
 - Assume Bob shops again
 - (Bob, 1999/05/07) is duplicated!
- In RDBMSs: add uniqueness constraints (Primary Key Constraints)

```
CREATE TABLE Customers (name varchar(255), birthdate DATE,  
PRIMARY KEY (name));
```

```
template1=# INSERT INTO Customers Values ('Bob', '1999/05/07');  
INSERT 0 1  
template1=# INSERT INTO Customers Values ('Bob', '1999/05/07');  
ERROR: duplicate key value violates unique constraint "customers_pkey"  
DETAIL: Key (name)=(Bob) already exists.
```

Takeaway: DBMSs will enforce the constraint and maintain the data's integrity at all times on behalf of the app!

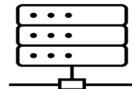
- Can enforce other integrity constraints (e.g., foreign key)

Concurrency When Using an RDBMS

- Recall Alice & Bob concurrently ordering BookA:

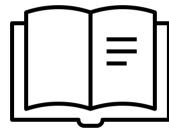
Product	NumInStock
...	...
BookA	1
...	...

```
Buy_Product_Subroutine(string prodName):  
    (prod, numInStock) = readProduct(prodName)  
    if (numInStock > 0):  
        writeProduct((prod, numInStock - 1))  
    else throw("Cannot buy product!");
```



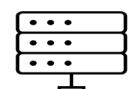
r: (A, 1)

w: (A, 0)



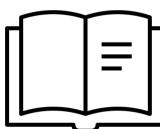
r: (A, 0)

No Book



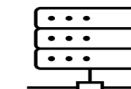
r: (A,1)

w:(A,0)



r:(A,0)

No Book

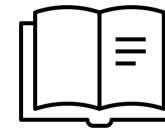


r: (A,1)

r: (A,1)

w:(A,0)

w:(A,0)



time

↓

Concurrency When Using an RDBMS

(Simplified) SQL:

BEGIN TRANSACTION

UPDATE Products

SET numInStock = numInStock - 1

WHERE name = "BookA"

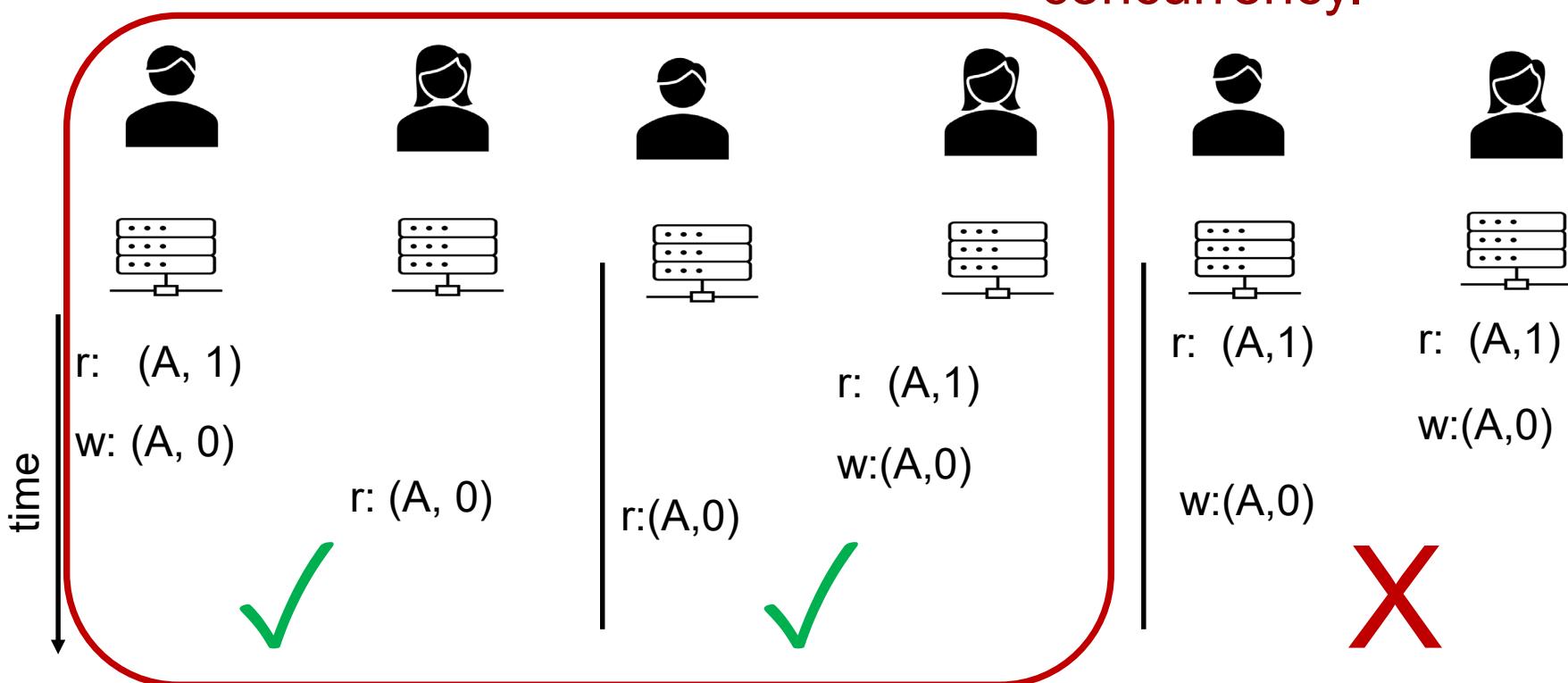
INSERT INTO Orders

VALUES ("Alice", "BookA", \$20)

COMMIT

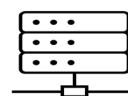
- Will ensure a correct end state
- Will avoid any deadlocks
- Will error for Alice or Bob

Take away: DBMS ensures safe concurrency.



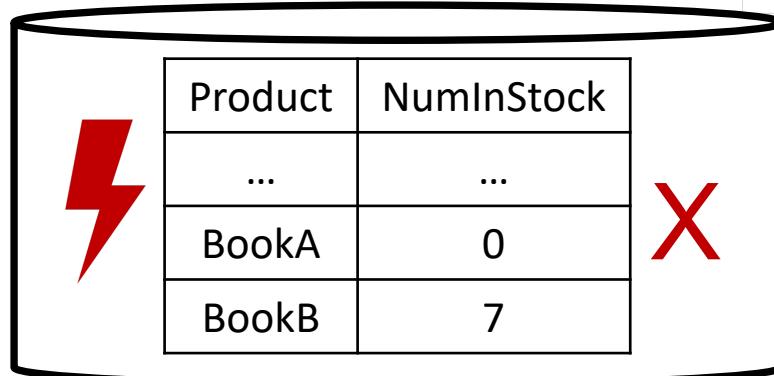
Backup and Recovery

- Recall failure scenario: Alice orders both BookA and BookB
- Suppose a power failure occurs and the DBMS fails in the middle of committing the transaction



r (A, 1)

w (A, 0)



Product	NumInStock
...	...
BookA	0
BookB	7

X

```
1 InnoDB: Log scan progressed past the checkpoint lsn 369163704
2 InnoDB: Doing recovery: scanned up to log sequence number 374340608
3 InnoDB: Doing recovery: scanned up to log sequence number 379583488
4 InnoDB: Doing recovery: scanned up to log sequence number 384826368
5 InnoDB: Doing recovery: scanned up to log sequence number 390069248
6 InnoDB: Doing recovery: scanned up to log sequence number 395312128
7 InnoDB: Doing recovery: scanned up to log sequence number 400555008
8 InnoDB: Doing recovery: scanned up to log sequence number 405797888
9 InnoDB: Doing recovery: scanned up to log sequence number 411040768
10 InnoDB: Doing recovery: scanned up to log sequence number 414724794
11 InnoDB: Database was not shutdown normally!
12 InnoDB: Starting crash recovery.
13 InnoDB: 1 transaction(s) which must be rolled back or cleaned up in
14 total 518425 row operations to undo
15 InnoDB: Trx id counter is 1792
16 InnoDB: Starting an apply batch of log records to the database...
17 InnoDB: Progress in percent: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
18 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
19 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
20 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
21 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
22 InnoDB: Apply batch completed
23 ...
24 InnoDB: Starting in background the rollback of uncommitted transactions
25 InnoDB: Rolling back trx with id 1511, 518425 rows to undo
26 ...
27 InnoDB: Waiting for purge to start
28 InnoDB: 5.7.18 started; log sequence number 414724794
29 ...
30 ./mysqld: ready for connections.
```

Product	NumInStock
...	...
BookA	1
BookB	7

✓

Summary

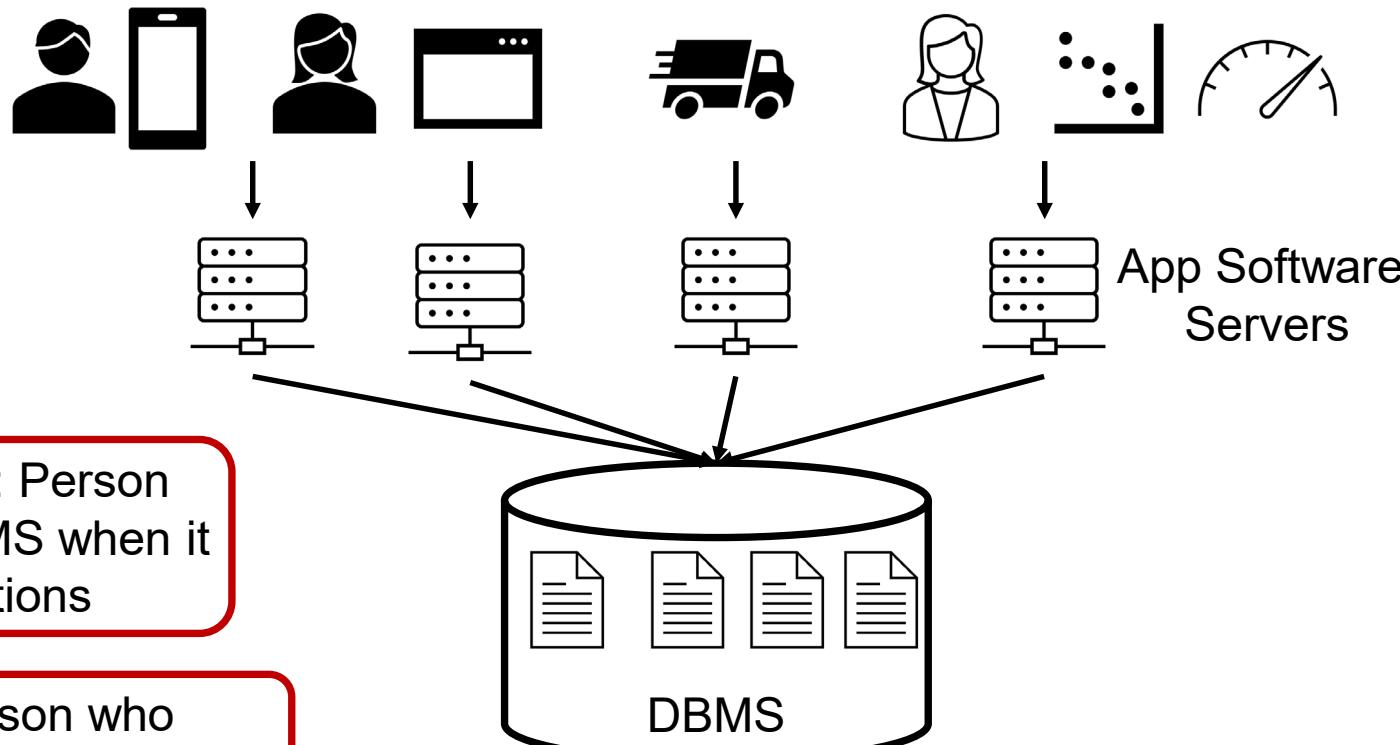
DBMS is an indispensable core system software to develop any application that stores, queries, or processes data.

Outline For Today

1. Overview of DBMSs: 3 Major Contributions of the Field
 1. Set of DBMS Features for Applications
 2. Physical Data Independence/High-level Query Languages
 3. Transactions
2. Course Diagram & Administrative Information

Key People When Developing Data-Intensive Applications

1: Application End Users



4. DBMS Architect: Person who actually implements the DBMS

- Won't differentiate between 2&3
 - ~2/3rd from the perspective of app developers
 - ~1/3rd on DBMS internals and architecture
- Want to learn more about internals of DBMSs: CS 448

CS 348 Diagram

User/Administrator Perspective

Primary Database Management System Features

- Data Model: Relational Model
- High Level Query Language: Relational Algebra & SQL, Datalog
- Integrity Constraints
- Indexes/Views
- Transactions

Relational Database Design

- E/R Models
- Normal Forms

How To Program A DBMS (0.5-1 lecture)

- Embedded vs Dynamic SQL
- Frameworks

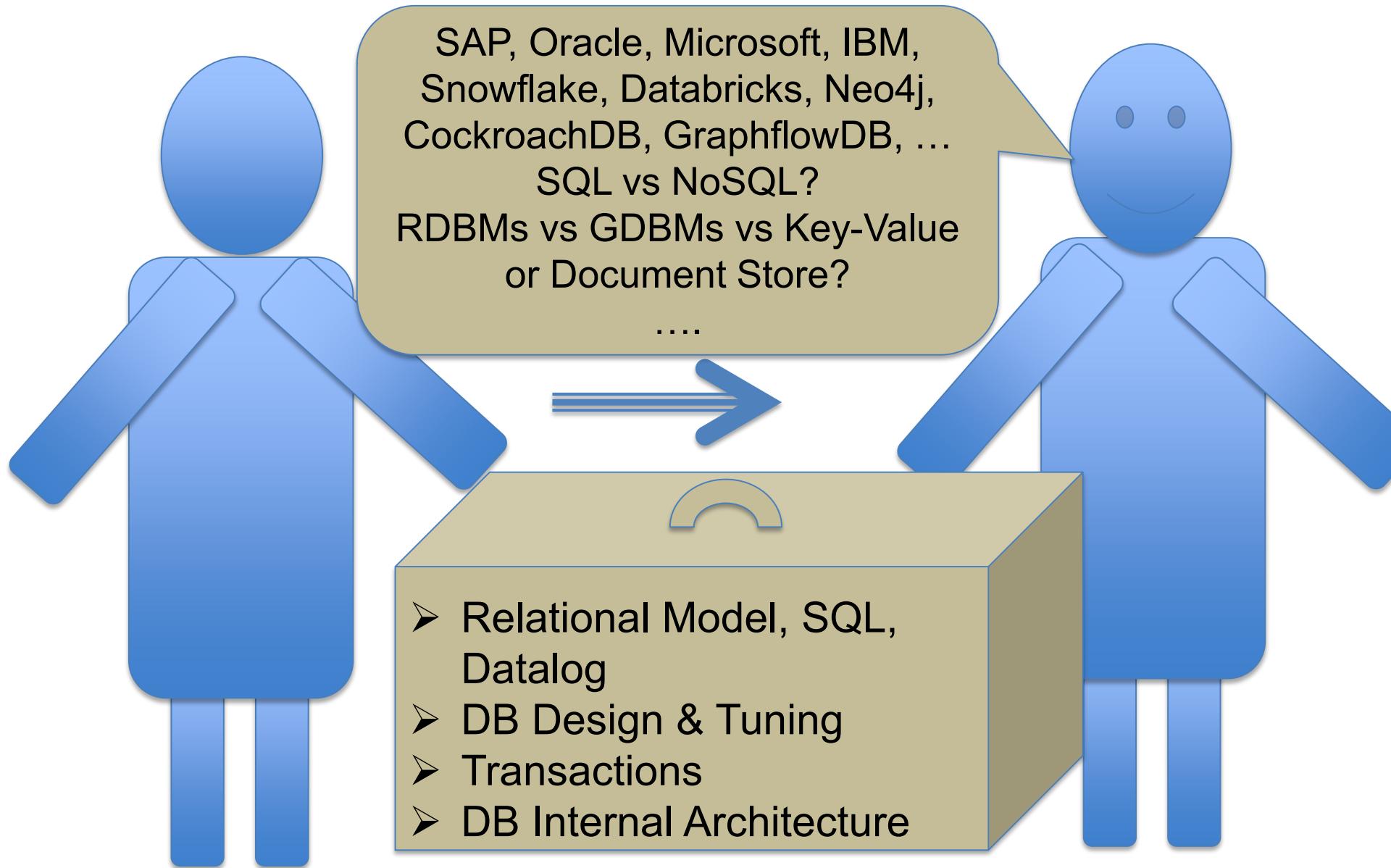
DBMS Architect/Implementer Perspective

- Physical Record Design
- Query Planning and Optimization
- Indexes
- Transactions

Other (Last 1/2 Lectures)

- Graph DBMSs
- MapReduce: Distributed Data Processing Systems

Before/After CS 348



A Glimpse of Current DBMS Market



Google



neo4j



mongoDB®

ORACLE

Cockroach DB



Materialize



databricks

amazon
DynamoDB



Hundreds of companies producing DBMSs: Many RDBMS/SQL,
but also graph, RDF, Document DB, Key-value stores etc..
Not even including companies to tune, ingest, visualize etc..

What I Want You To Watch

- [Short Video On History of DBMSs by Computer History Museum \(CHM\)](#)
- [Charles Bachmann's Talk at CHM on Developing IDS](#)
 - IDS: The First DBMS in the world (developed around 1960-1964)
- [Jim Gray's Interview on Microsoft Research on His Career & Projects](#)
 - [This short video is his high-level overview of transactions & ACID](#)
- No Public Talk of Ted Codd I know of. If you ever find one let me know!

What I Want You To Never Forget

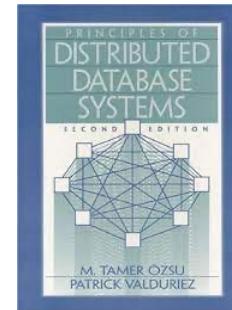
- I have highlighted 3 main intellectual contributions and some of the key people behind those contributions
- But as any field: the field is much bigger than these people
- Excellent researchers and engineers continue to push the field forward
- E.g: Patricia Selinger: core engineer in System R, first RDBMS prototype.
Lead of first cost-based query optimizer.
- E.g: Prof. Tamer Ozsu of Waterloo DSG: key figure in distributed DBMSs
- E.g: Ken Salem of Waterloo DSG (co-inventor of disk-striping, which is the foundations of [RAID disks](#))



Patricia Selinger



Tamer Ozsu



Ken Salem

Administrative Info (1)

- Instructor: Semih Salihoglu (semih.salihoglu@uwaterloo.ca)
- Planning for an online instead of a hybrid course
 - School is officially online until Feb but expect a longer delay
 - If we go in person, we *might* have an actual midterm and final.
 - Plan for course organization aside from in-person lectures is same
- OHs: Mondays 2:00pm-3:00pm over Zoom (see Piazza)
- Unofficial Co-Instructor: [Prof Xi He!](#)
 - Will use much of her material and her videos
- TAs: Guy Coccimiglio, Glaucia Melo, Karl Knopf, Chang Liu, Amine, Mhedhbi
- TA OHs: 2 hours on weeks assignments are due
- Lectures: For online part: recorded and available on Learn
 - If we are in-person, I will deliver in-person lectures

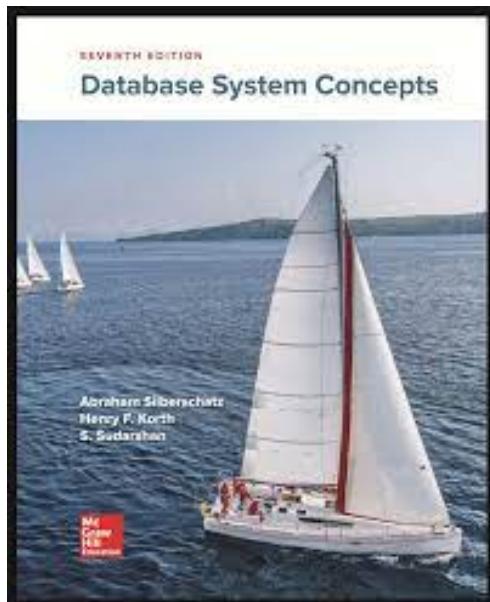


Administrative Info (2)

- Learn: <https://learn.uwaterloo.ca/d2l/home/761209>
- Piazza: <https://piazza.com/class/kxhps7c94rj9r>
 - Unless urgent, we will wait for students to answer
 - We will interfere when there is confusion
 - *Please be active! This our best forum for communication.*

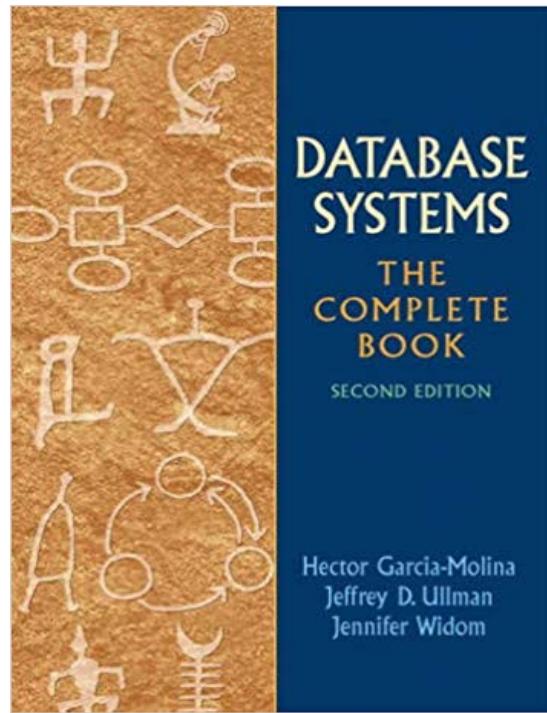
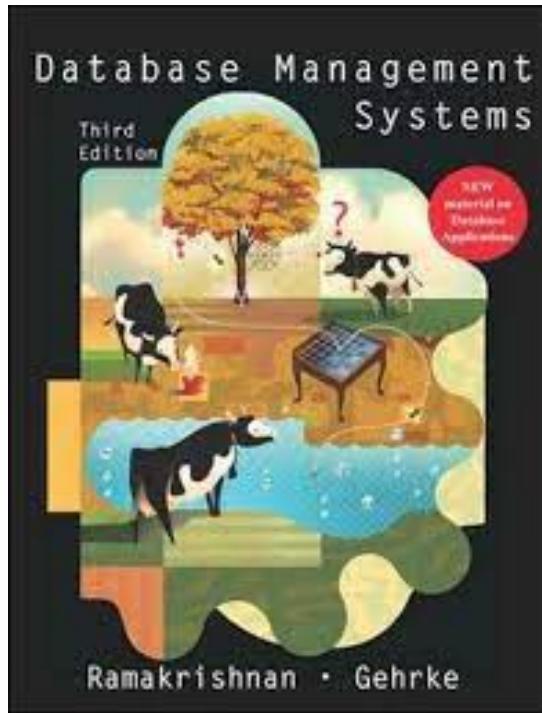
Administrative Info (3)

- Textbook: [Database System Concepts](#), Silberschatz et al., 7th edition
 - *"The library has electronic access only to the 2006 edition of "Database System Concepts by Silberschatz." This access is through Hathi Trust which is emergency access. Access is restricted to one user and for one hour at a time."*
- (Rare) Optional: [Designing Data Intensive Applications](#), Kleppmann



Administrative Info (3)

- 2 Other Main Textbooks in the Field



Administrative Info (4)

➤ Workload & Mark Distribution

- 4 assignments:
 - ~3 will have programming questions.
 - 1 or 2 weeks (not guaranteed to be 2 weeks) to complete each.
 - Submit through Crowdmark/Marmoset
 - First one is out next Friday, Jan 14th. Due Jan 28th.
 - 1 “Midterm Assignment”. You will have 1 day
 - 1 “Final Assignment”. You will have 1 day.
 - Midterm & Final Assignments are comprehensive w/ different late policy
- Late Policy: You have 2 extra days
 - For 4 Assignments: Lose 5% for each additional day
 - For Midterm and Final Assignments: Lose 15% for each additional day

Prerequisites

- CS 240/240E is listed but not strictly necessary.
- Programming in a standard language: TBA
- General Interest in Software Systems, Data-Intensive Application
Programming and Data Management/Processing Systems