CS 348 Lecture 1

Course Overview & Organization

Semih Salihoğlu

Jan 6th, 2025

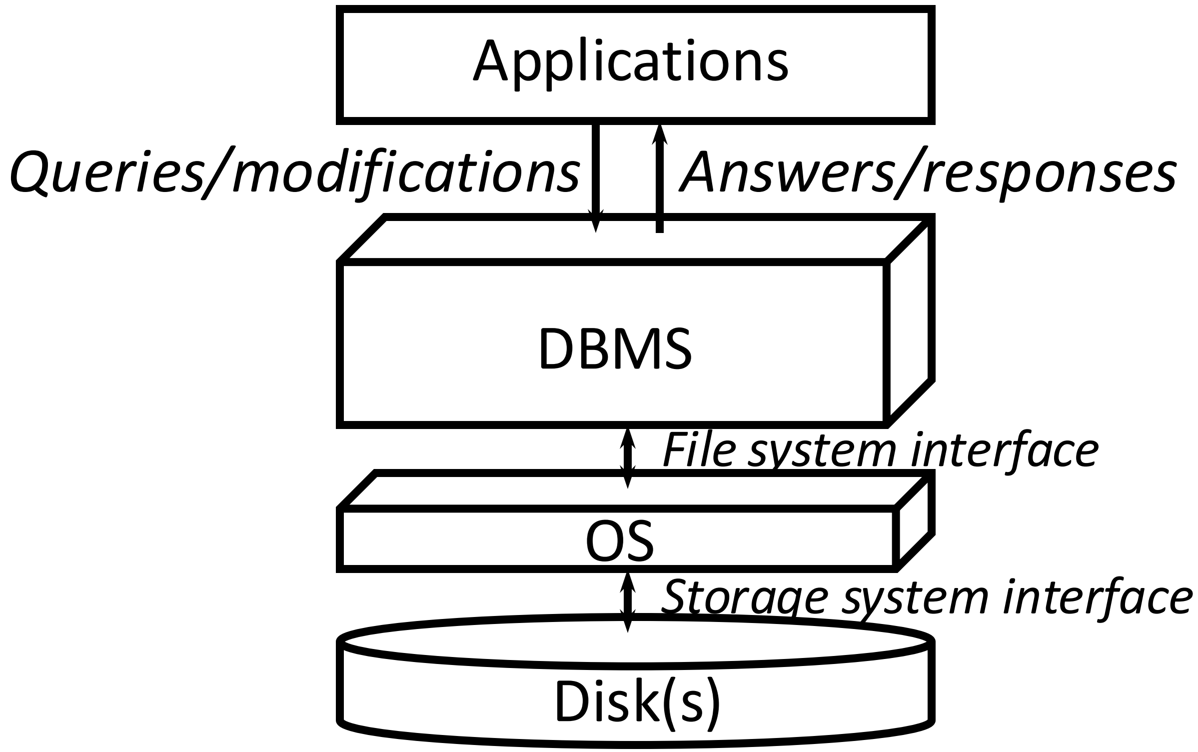
Outline For Today

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

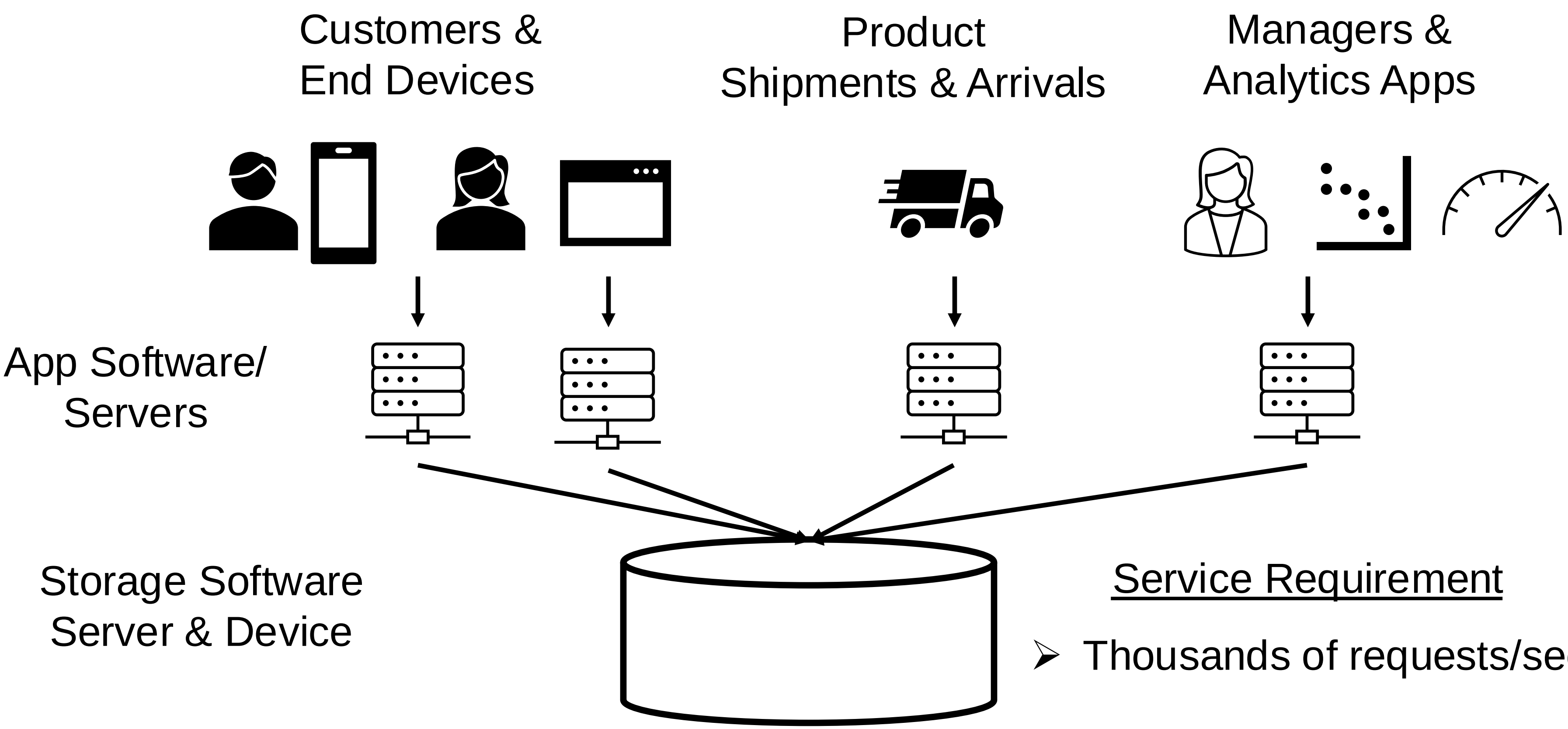
- High-level Data Model and Query Language
- Efficient access and processing of data
- Scalability:
  - Handling of Large Data, i.e., Out-of-memory Data
  - 10-100Ks of concurrent data access/sec
- Safe access and processing of data:
  - Maintenance of the integrity of the data upon updates
  - Multi-User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

Main Contributions of the Field?

1. The System
2. High-level/Declarative Programming
   - 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
   - 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



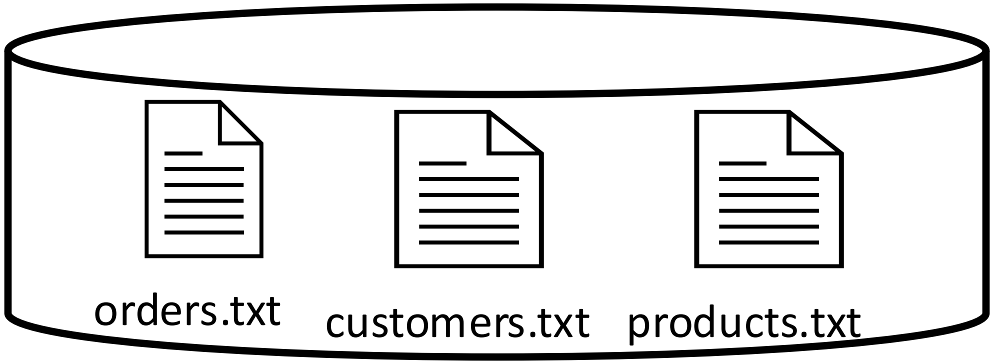
<u>Service Requirement</u>

- Thousands of requests/sec

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	2002/05/20
6	Ali Jo	1992/02/25	26	Montgomery Cambridgeshire	1992/02/25
...	...	...	...	...	...
...	...	...			

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

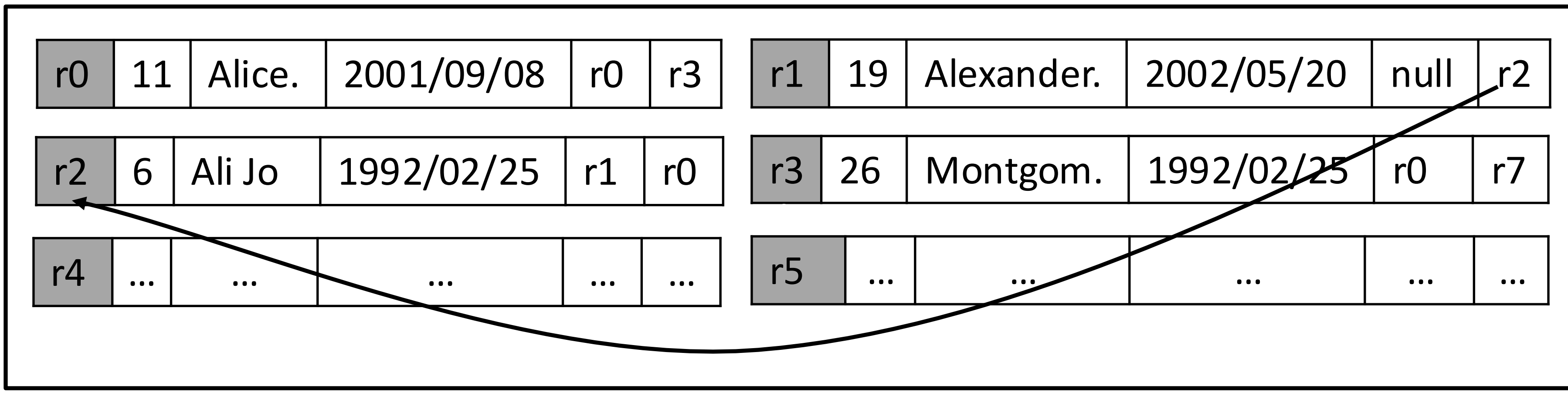
ona	idgeshire	....
...	...	....

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
…	…	…	…
…	…	…	…

orders.txt

```
Possible High-level Algorithm 2:
sort orders.txt on CustID
orders of Cust_i are now consecutive
read sorted records sequentially
and sum prices for each C_i
```

→ *Which sorting algorithm to use?*
Should one parallelize sorting? How?

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

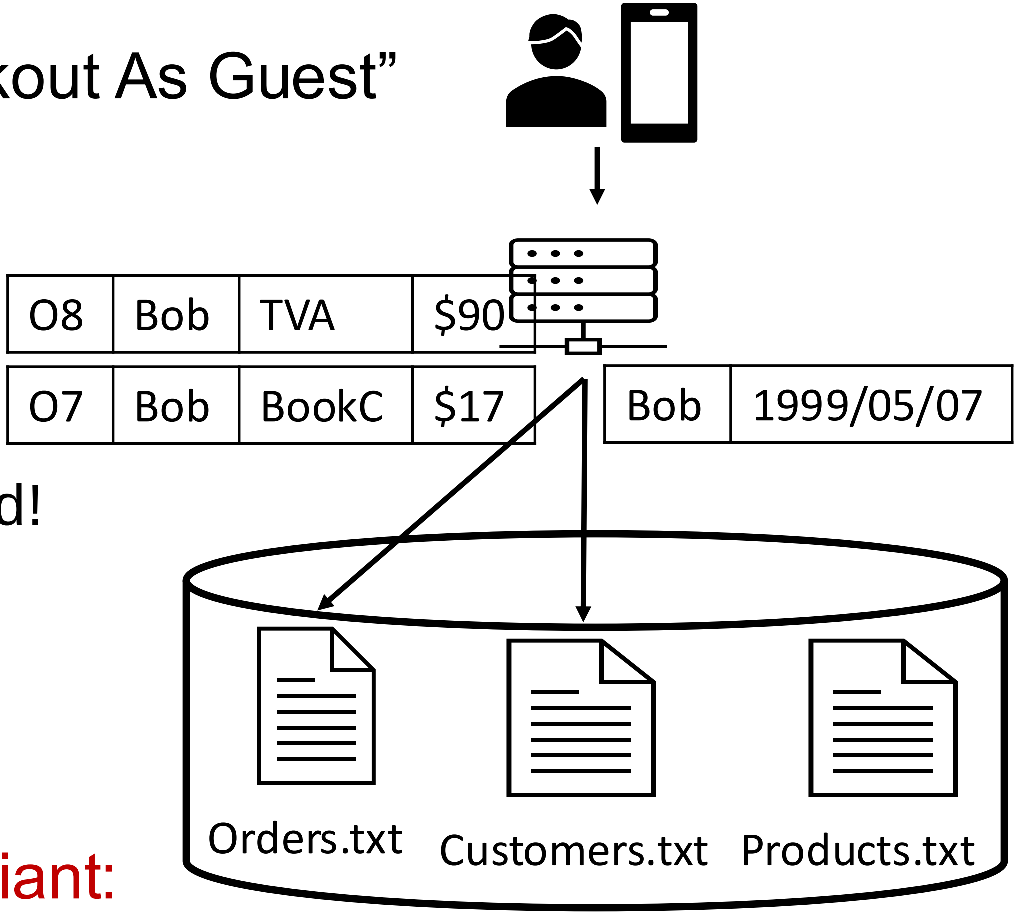
- Large-scale Data: Data > Memory
  - E.g. orders.txt grows to terabytes & does not fit in memory.
  - Often the case for data-intensive applications
  - Need ``External'' algorithms, i.e., uses disk to scale
  - Hard to write such algorithms. Challenge:
    - *Try implementing a good external sorting 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!



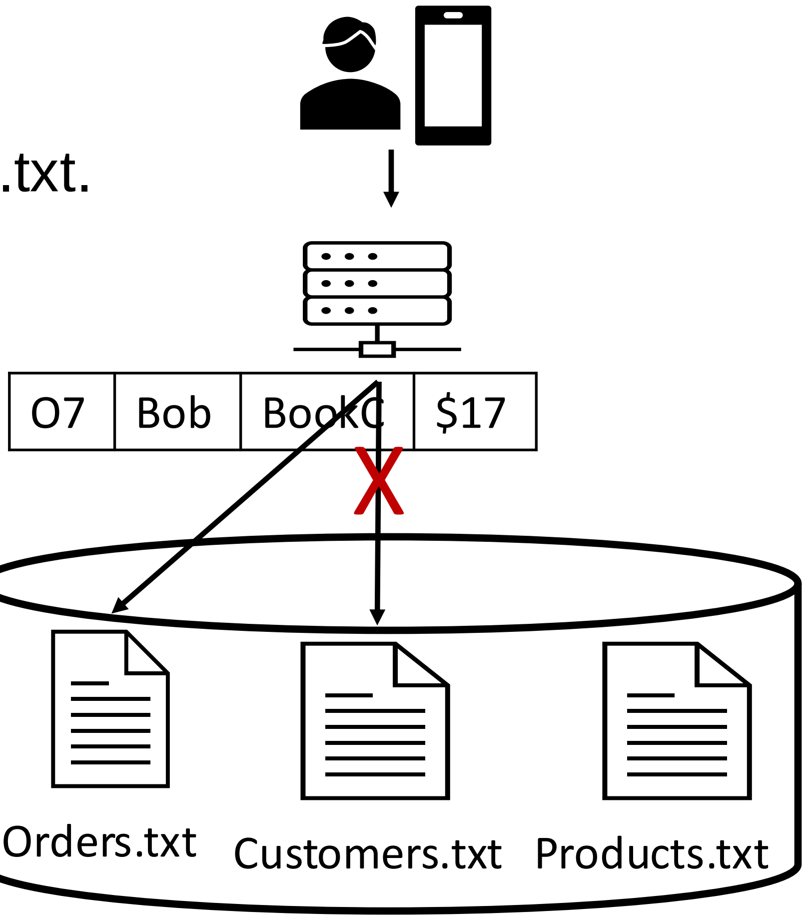
Likely an inconsistency.

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!

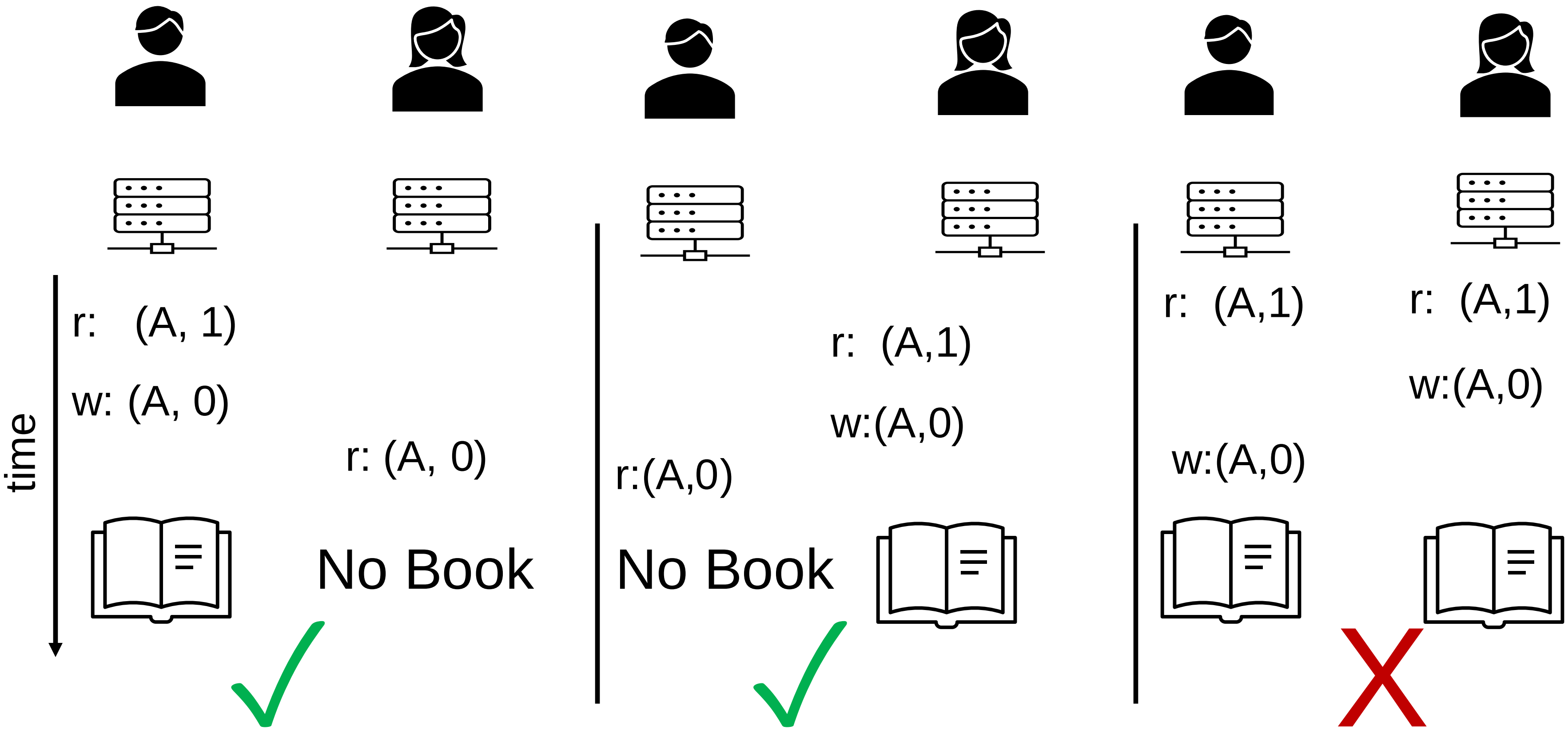
- Another example momentarily in concurrency

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!”);
```



Concurrency Questions

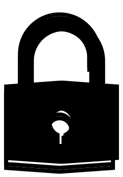
- What is a correct/incorrect state upon concurrent updates?
  - Theoretical formalism to explain these states: Serializability
- What protocols/algorithms can ensure a correct state?
  - Locking-based protocols
    - Pessimistic: set of lock acquisitions to prevent bad state
  - Optimistic protocols
    - 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

time ↓

Bob	Alice
lock DB X Wait	lock DB ✓
	r (A, 1)
	w (A, 0)
	release lock
lock DB ✓	

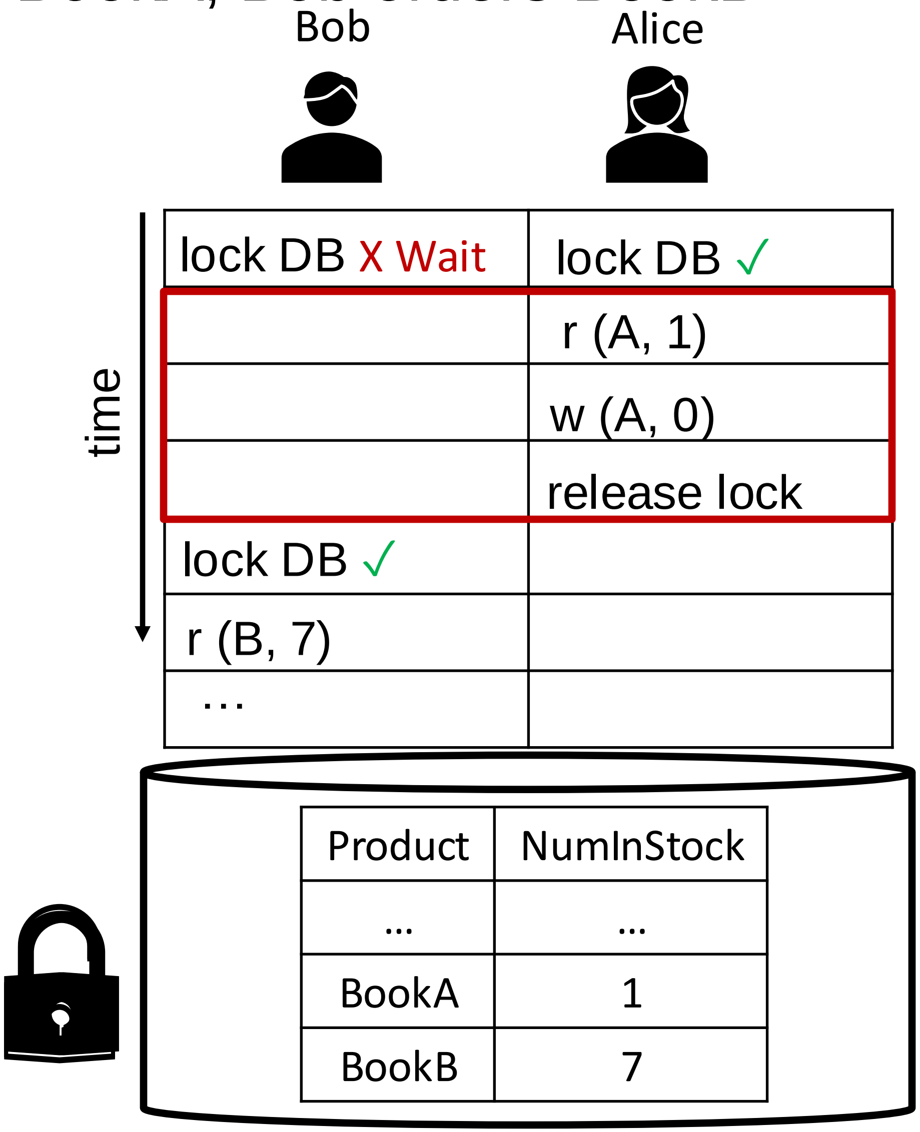


Product	NumInStock
…	…
BookA	1
BookB	7

Safe but inefficient. Why?

Concurrency Avoidance Ex: Global DB Lock

- Alice orders BookA, Bob orders BookB

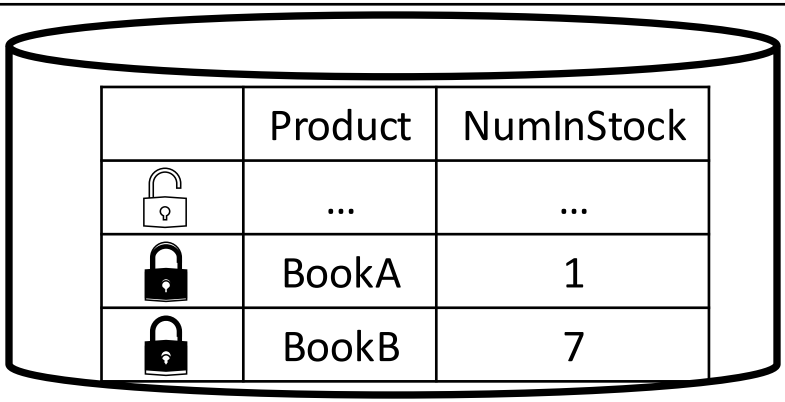


Bob had no conflicts; so was "unnecessarily" blocked.

Concurrency Avoidance Ex: Record-level Lock

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

time ↓	Bob	Alice	Carmen
	lock: (A, 1) ✓	lock: (A, 1) X Wait	lock: (B, 7) ✓
	r (A, 1)		r (B, 7)
	w (A, 0)		w (B, 6)
	…		



	Product	NumInStock
	…	…
	BookA	1
	BookB	7

Concurrency Avoidance Ex: Record-level Lock

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

time ↓

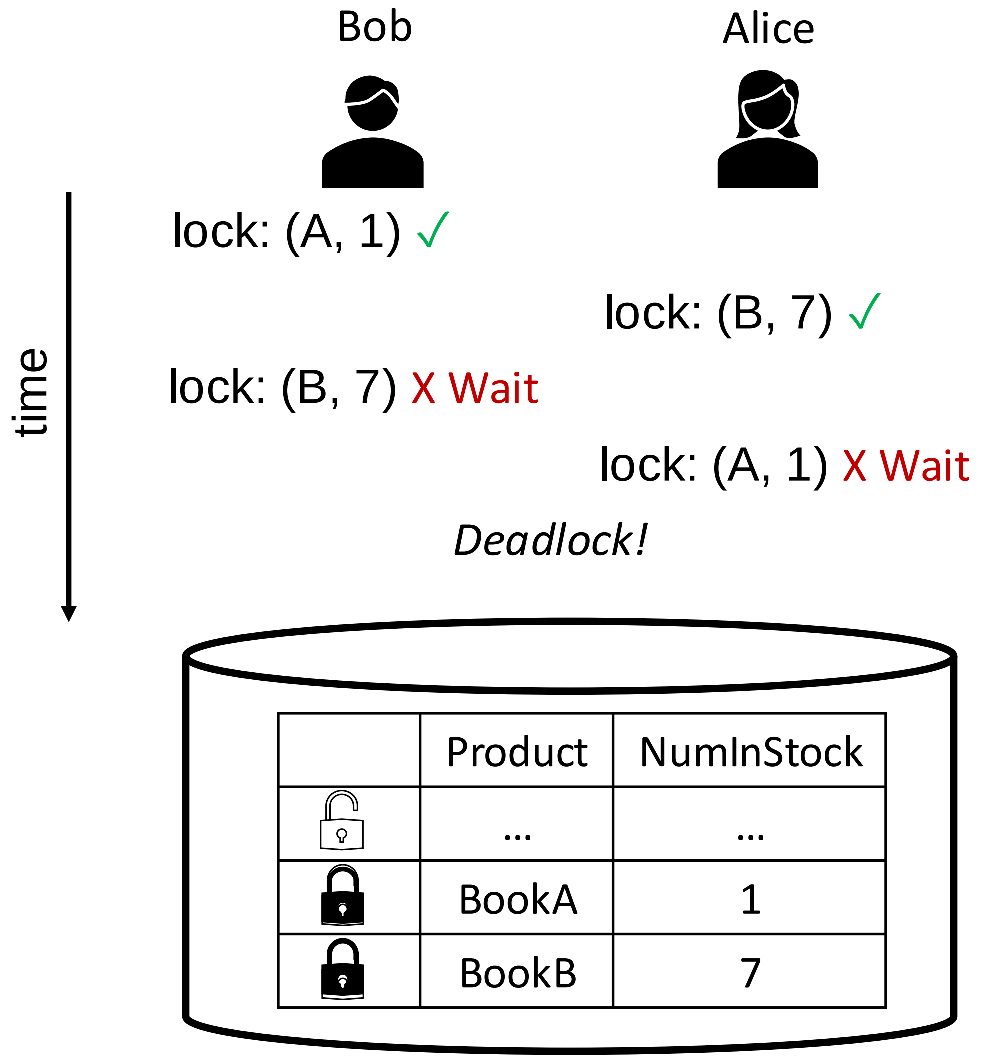
Bob	Alice	Carmen
lock: (A, 1) ✓	lock: (A, 1) X Wait	lock: (B, 7) ✓
r (A, 1)		r (B, 7)
w (A, 0)		w (B, 6)
release lock A		release lock B
	lock: (A, 1) ✓	
…		

	Product	NumInStock
	…	…
	BookA	0
	BookB	6

Safe and achieves parallelism. What can go wrong?

Where There is Locking, There is Deadlocks!

- Alice, Bob both order both BookA and BookB together

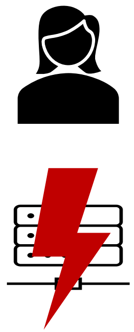


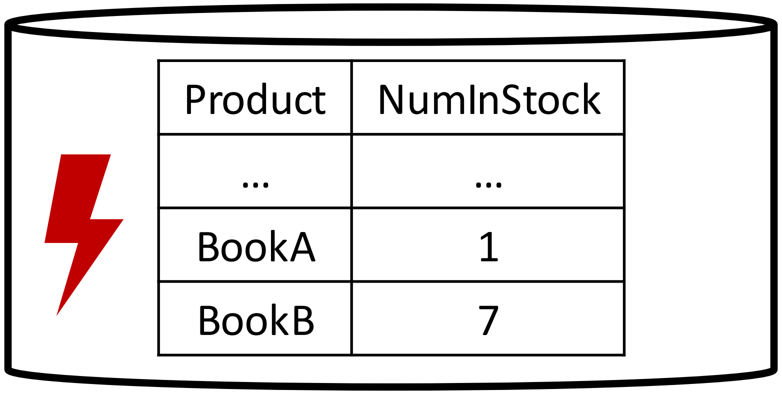
	Product	NumInStock
	…	…
	BookA	1
	BookB	7

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?

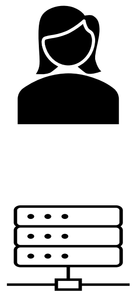




Product	NumInStock
…	…
BookA	1
BookB	7

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

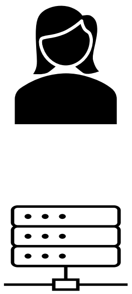


w (A, 0)

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



w (A, 0)

Before (B, 6) is written failure!
Inconsistent data state!
PR: How to recover from inconsistent state?

Product	NumInStock
…	…
BookA	0
BookB	7

X

✓

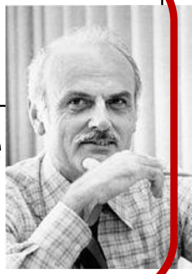
Product	NumInStock
…	…
BookA	0
BookB	6

Contributions of DBMSs To Computing

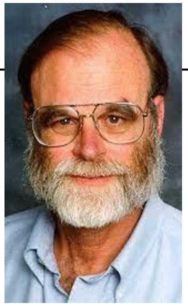
- DBMSs provide solutions to all of the problems we identified!
- Allows app developers to focus on the application logic.

Problems	Solutions
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
4. Safe Concurrency	Transactions & ACID guarantees
5. Other Safety Features:	Data Integrity and Failure Recovery

Contribution 2



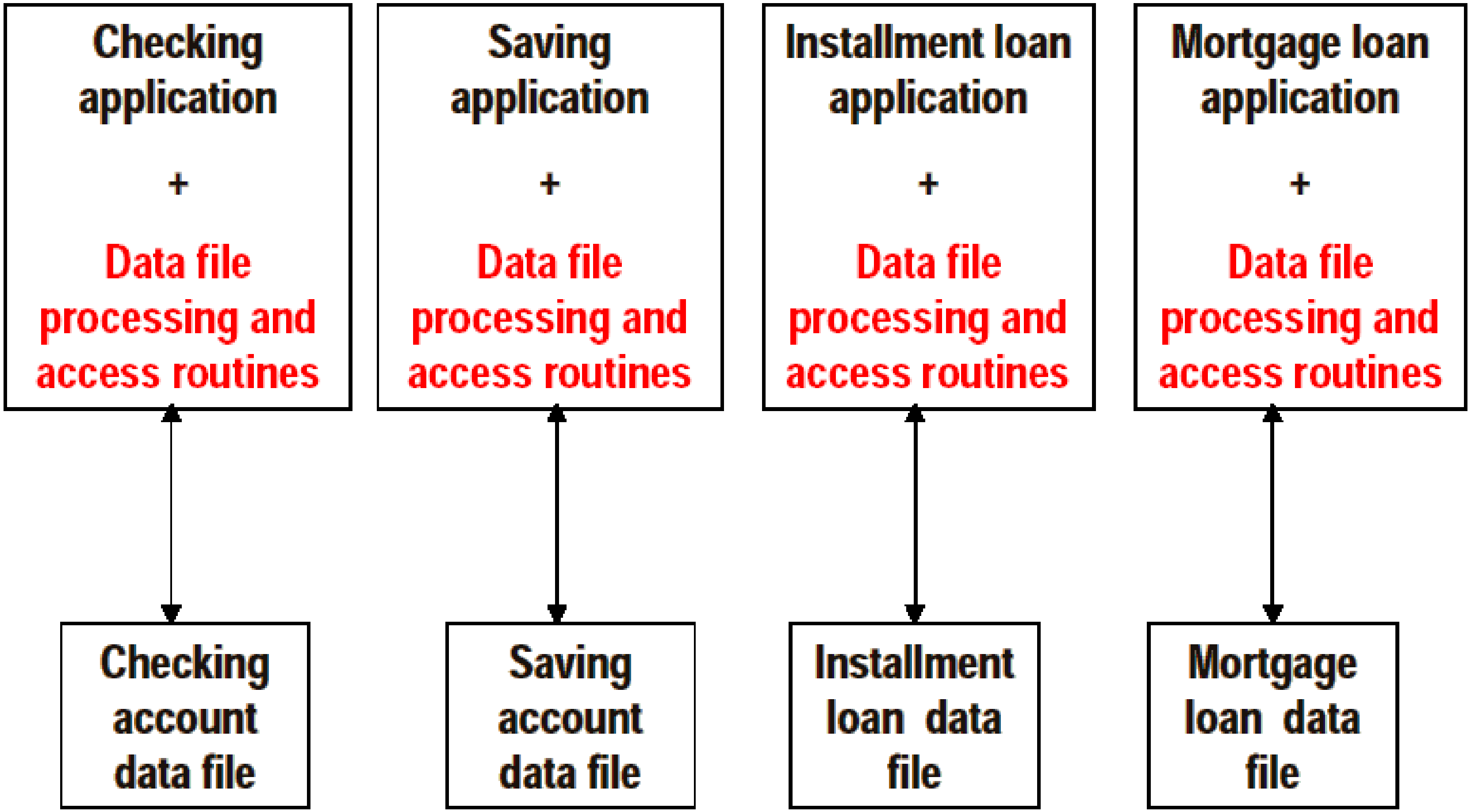
Contribution 3



Contribution 1: The System

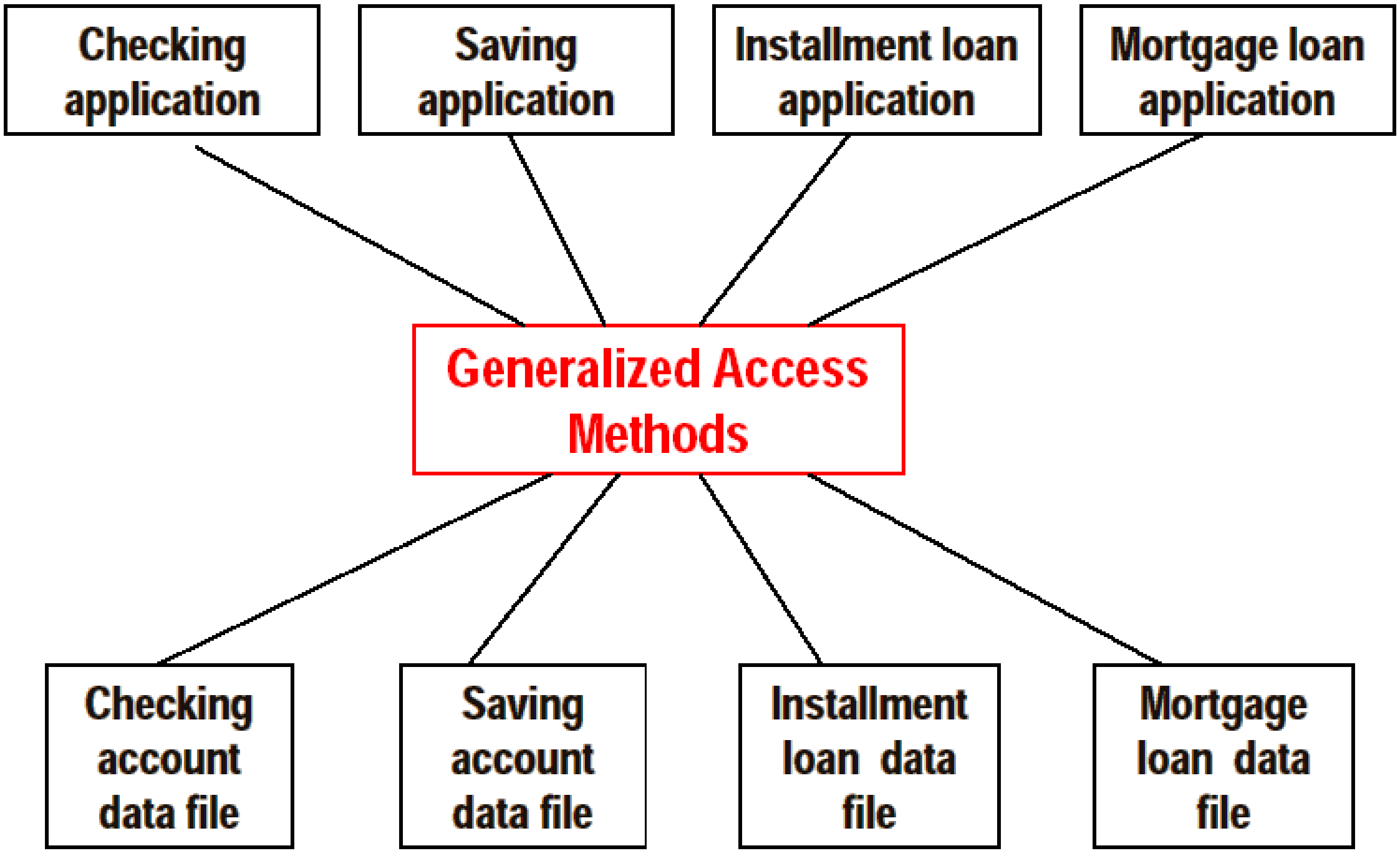
- IDS (1960s): First DBMS
  - Had a data model and a primitive "query" language
  - Had scalability for its era and integrity and recovery
  - No transactions

The Birth of DBMS (1)



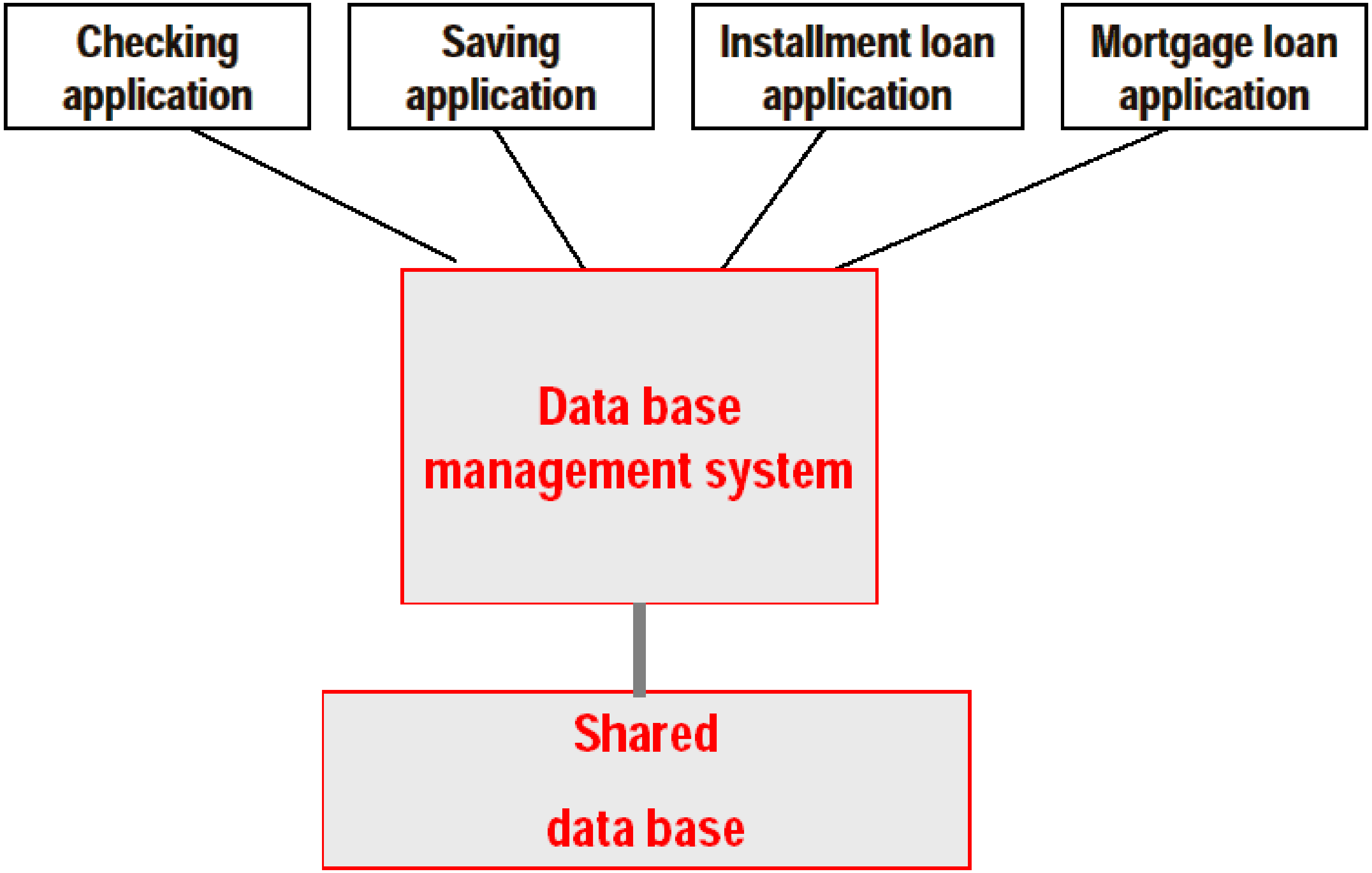
From Hans-J. Schek's *VLDB* 2000 slides

The Birth of DBMS (2)



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

The Birth of DBMS (3)



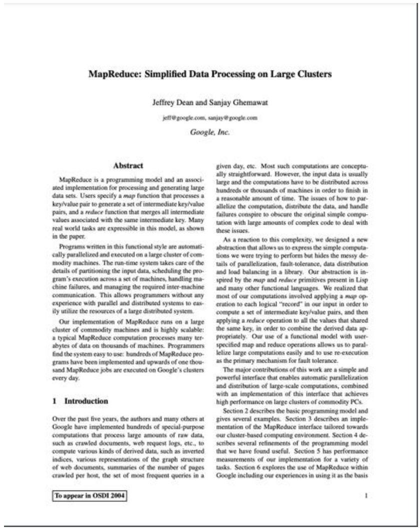
DBMS is an excellent example of a successful abstraction!

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

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 generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

1 Introduction

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.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then applying a *reduce* operation to all the values that shared the same key, in order to combine the derived data appropriately. Our use of a functional model with user-specified map and reduce operations allows us to parallelize large computations easily and to use re-execution as the primary mechanism for fault tolerance.

The major contributions of this work are a simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs.

Section 2 describes the basic programming model and gives several examples. Section 3 describes an implementation of the MapReduce interface tailored towards our cluster-based computing environment. Section 4 describes several refinements of the programming model that we have found useful. Section 5 has performance measurements of our implementation for a variety of tasks. Section 6 explores the use of MapReduce within Google including our experiences in using it as the basis

To appear in OSDI 2004

1

The Birth of MapReduce (1)

Google Inverted Index Constructor

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

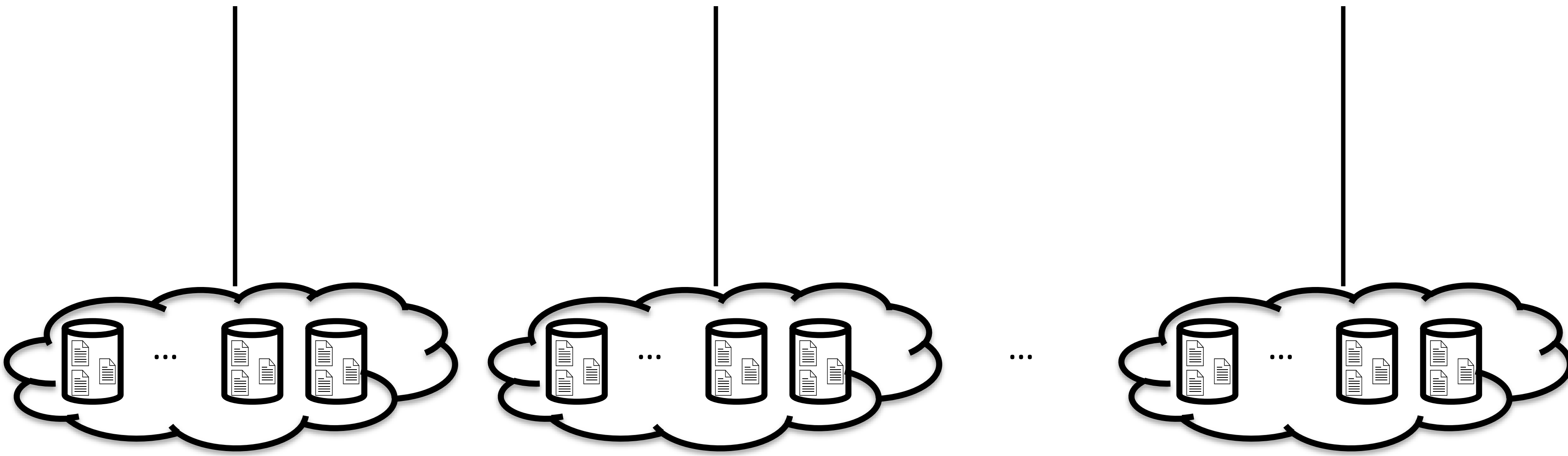
Google PageRank Computation

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

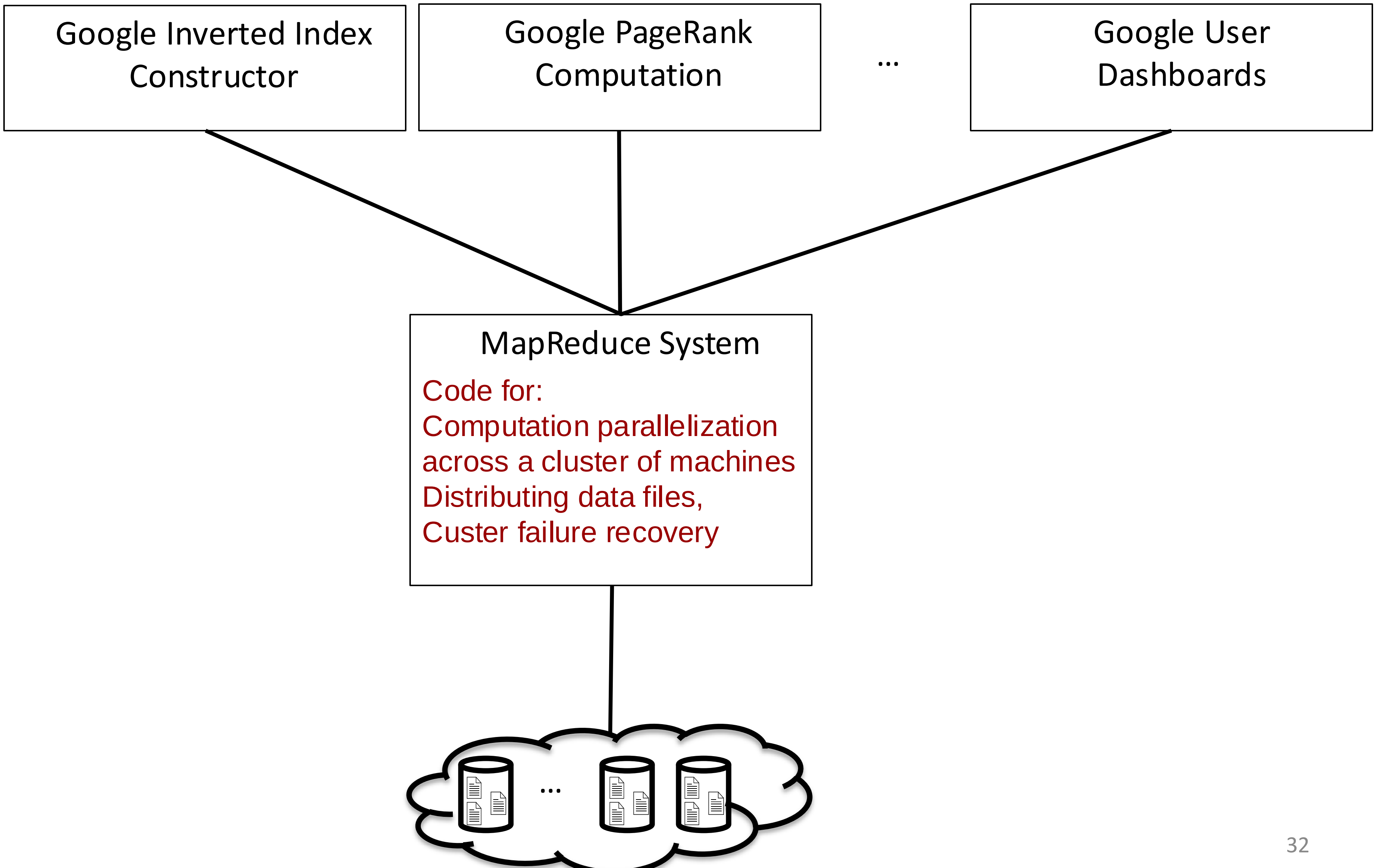
…

Google User Dashboards

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



The Birth of MapReduce (2)



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	
name	birthday
Alice	2001/09/08
Bob	2002/05/20
…	…

Orders			
oID	cust	product	price
O1	2001/09/08	BookA	20
O2	2002/05/20	TVB	100
…	…	…	…

Products	
product	numInStock
BookA	1
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 referred to as a *declarative* language:
  - Describe outputs of computation *but not how to perform it*
- "Declarative"ness is subjective and relative:
  - E.g. Prolog > SQL > {C,C++,Java}
- Recall managers' question: Who are top paying customers?

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

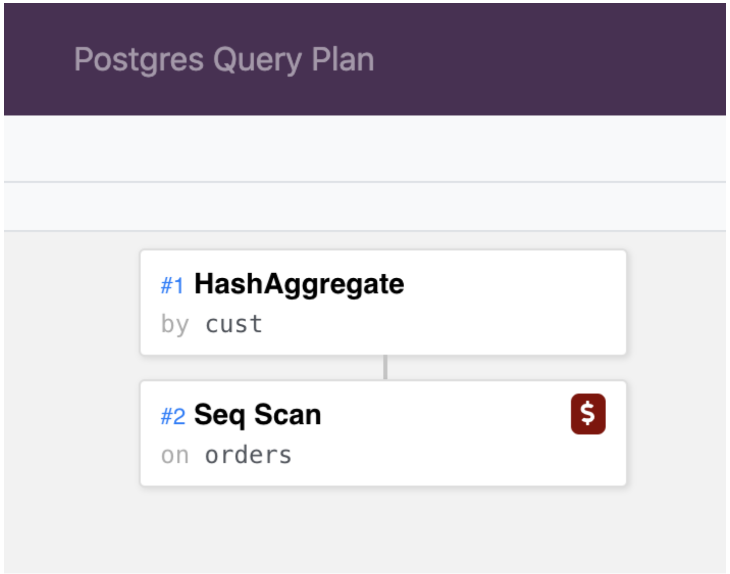
Orders			
oID	cust	product	price

- 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!

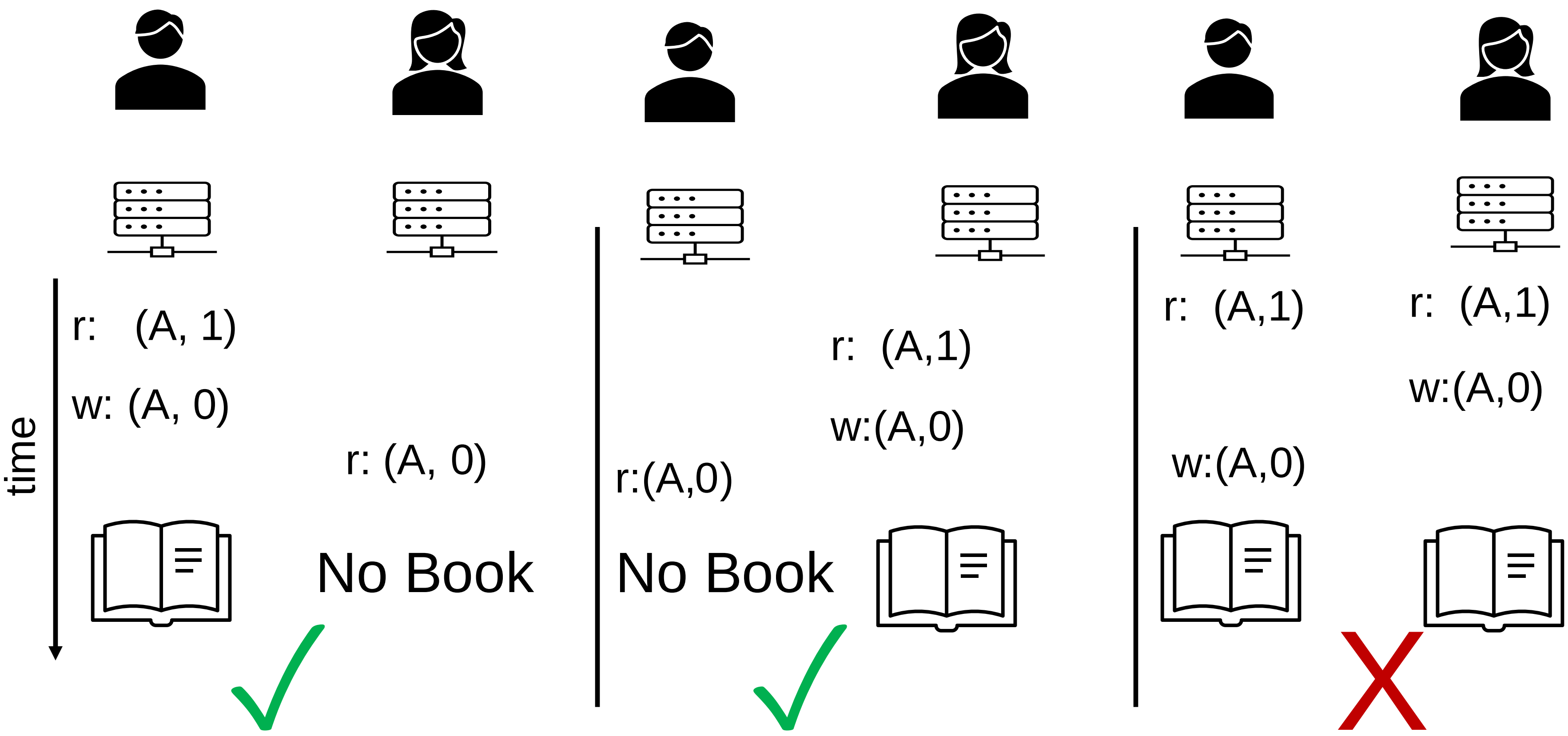
- Can enforce other integrity constraints (e.g., foreign key)

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!”);
```



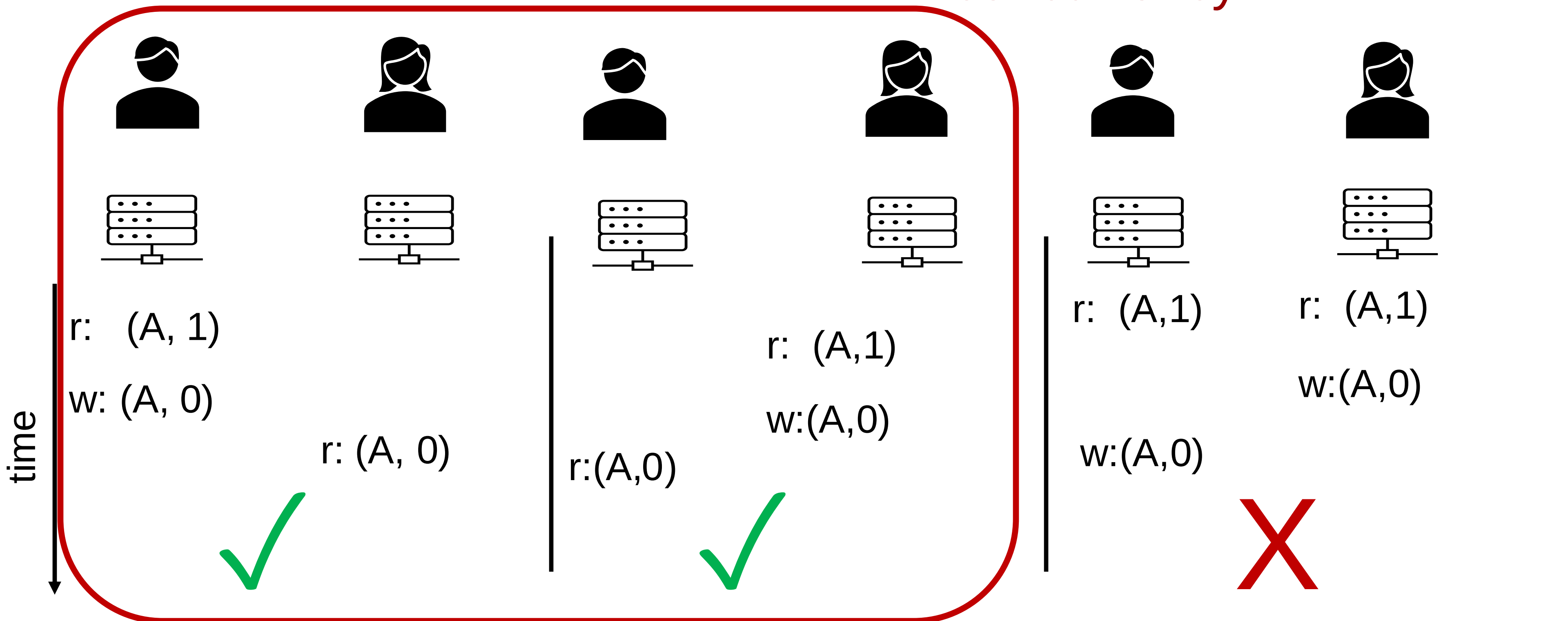
Concurrency When Using an RDBMS

```
(Simplified) SQL:
BEGIN TRANSACTION
UPDATE Products
SET numInStock = numInStock - 1
WHERE name = “BookA”

INSERT INTO Orders
VALUES (“Alice”, “BookA”, $20)
COMMIT
```

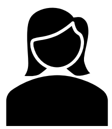
- Will ensure a correct end state
- Will avoid any deadlocks
- Will error for Alice or Bob

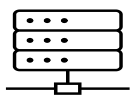
Take away: DBMS ensures safe concurrency.



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)

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

Product	NumInStock
…	…
BookA	0
BookB	7

X

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

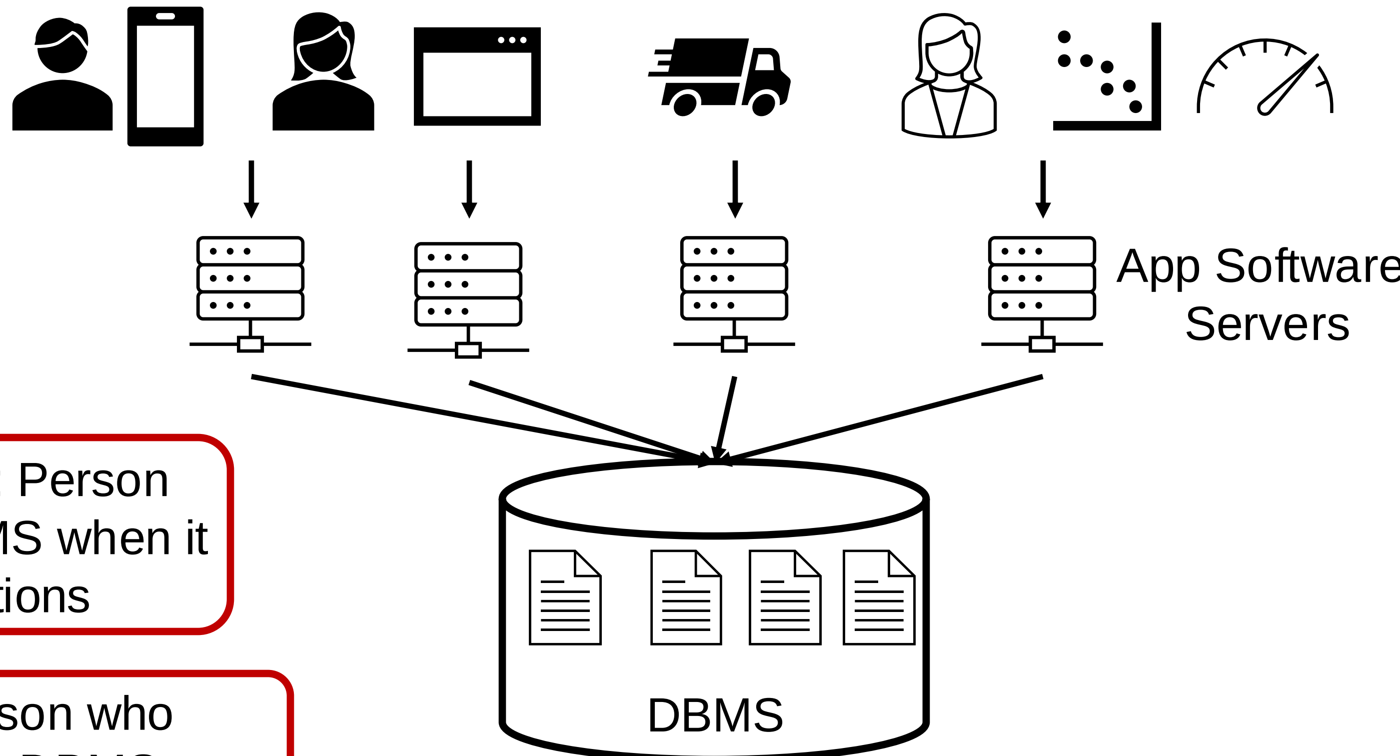
Key People When Developing Data-Intensive Applications

1: Application
End Users

2. App Developers:
DBMS Users

3. DBMS Administrator: Person who maintains the DBMS when it serves multiple applications

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 (6 lectures)

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

Relational Database Design (4 lectures)

- E/R Models
- Normal Forms

How To Program A DBMS (0.5-1 lecture)

- Embedded vs Dynamic SQL
- Frameworks

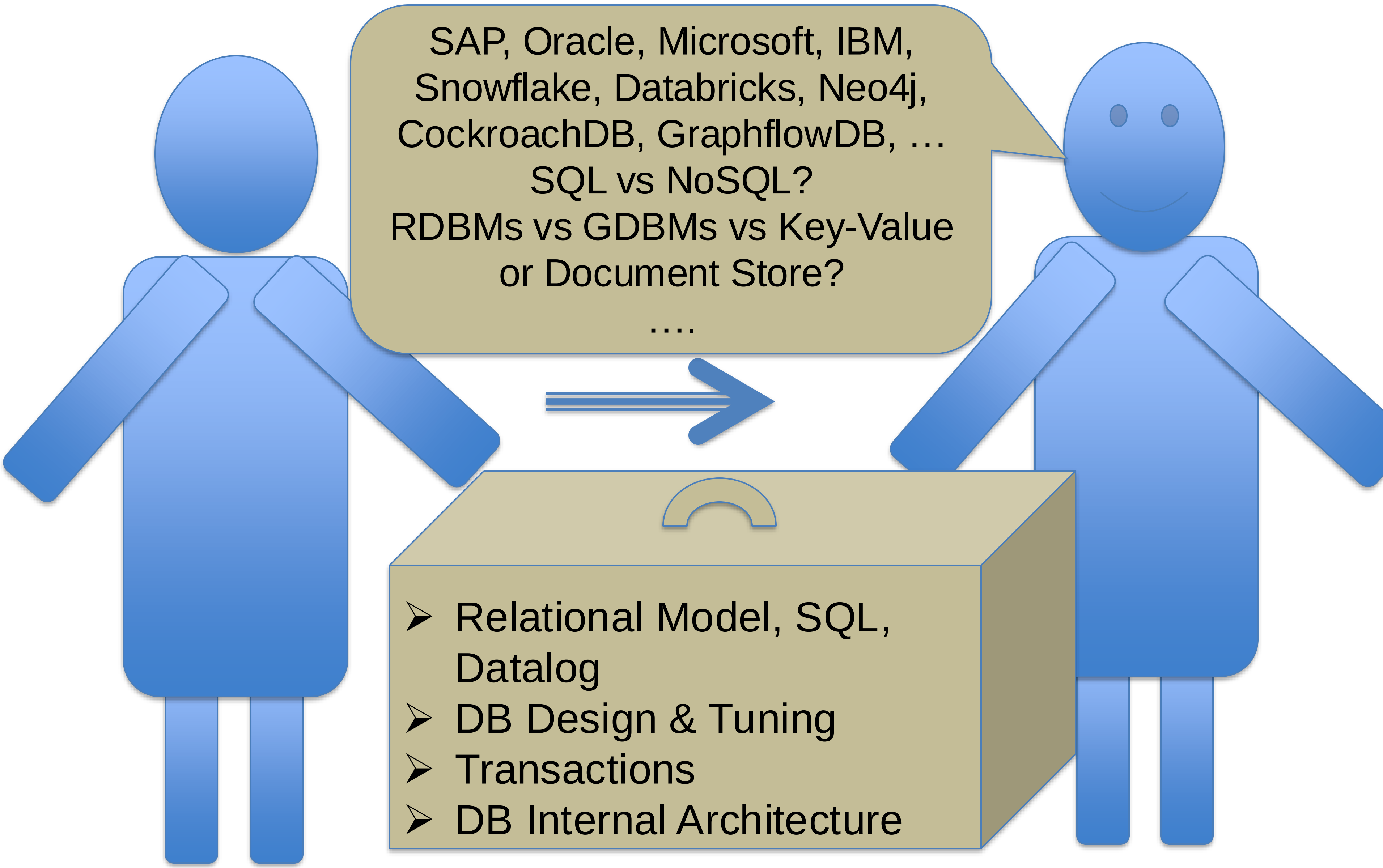
DBMS Architect/Implementer Perspective (8 lectures)

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

Other (Last 1/2 Lectures)

- Graph DBMSs or
- RDF Systems

Before/After CS 348



A Glimpse of Current DBMS Market

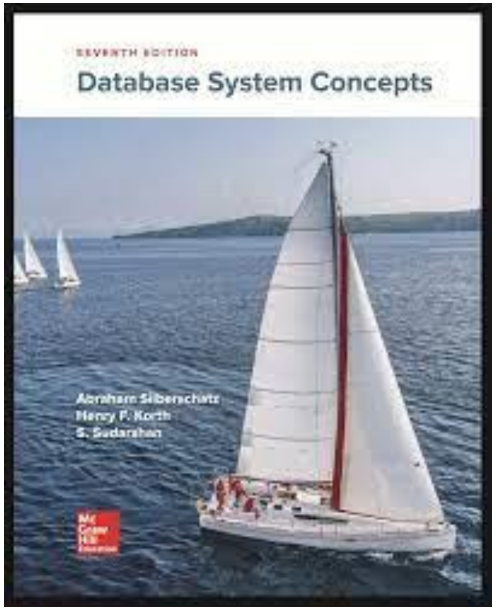
Hundreds of companies producing DBMSs: Many RDBMS/SQL, but also graph, RDF, Document DB, Key-value stores etc..
Not even including companies to tune, ingest, visualize etc..

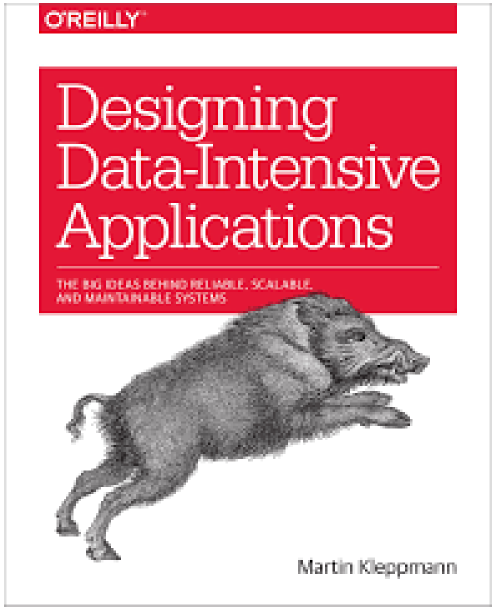
Administrative Info

- Instructor: Semih Salihoglu (semih.salihoglu@uwaterloo.ca)
- OHs: Mondays 4:00pm-5:00pm @ DC 3351
- TAs: Guy Coccimiglio, Shubhankar Mohapatra, Anurag Chakraborty, David Rui, Gaurav Sehgal, Nimmi Rashinika Weeraddana
- TA OHs: a few hours on weeks assignments are due
- Course Coordinator: Sylvie Davies
- Website: https://student.cs.uwaterloo.ca/~cs348/
- Learn: https://learn.uwaterloo.ca/d2l/home/1098090
- Piazza: https://piazza.com/class/m4vnhnp05wrc4
  - 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

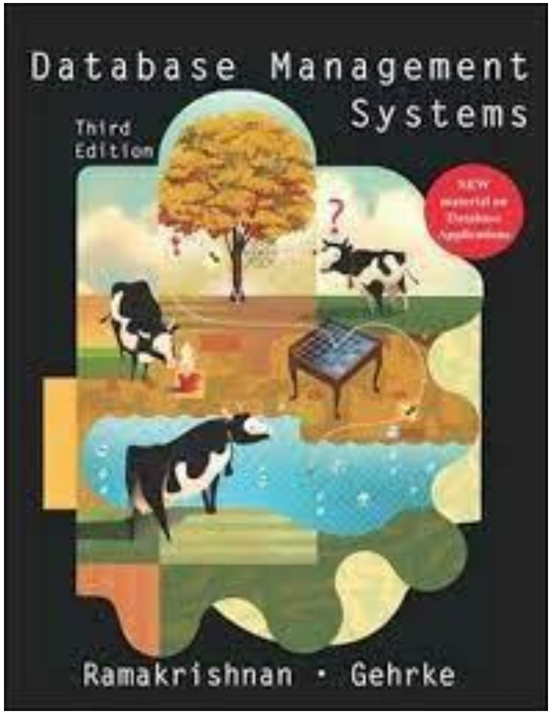
- Textbook: Database System Concepts, Silberschatz et al., 7th edition
  - *"The library has electronic access only to the 2006 edition of "Database System Concepts by Silberschatz." This access is through Hathi Trust which is emergency access. Access is restricted to one user and for one hour at a time."*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

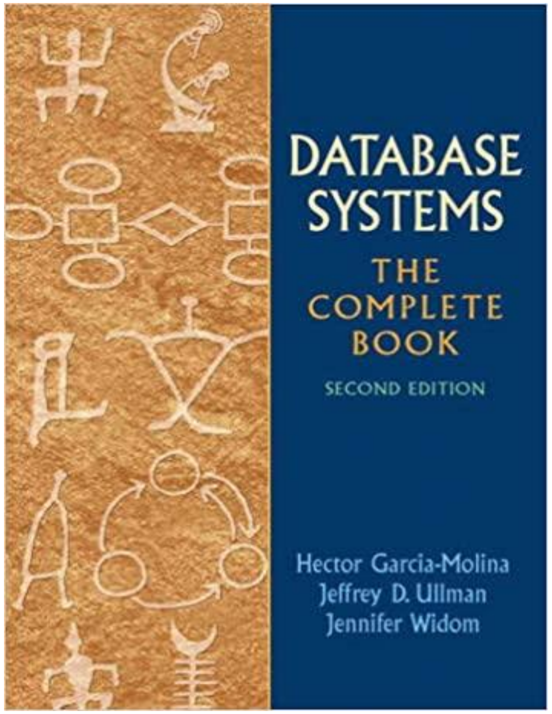




Administrative Info

- 2 Other Main Textbooks in the Field





Administrative Info

- Workload & Mark Distribution: 2 options

	Assignments	Group Project	Midterm	Final
Option 1	30%	--	30%	40%
Option 2	30%	30%	15%	25%

- Midterm: Feb 28th (4:30-6pm)
- Final: Not yet announced by the university
- Late Policy: 2 extra days for each assignment or project milestone
  - For Assignments: Lose 5% for each additional day
  - For Project milestones: Lose 25% for each day

Projects

- Teams of 4 or 5 students
- Implementing a database application
- Detailed information next week
- 4 milestones (deadlines are tentative but will finalize this week)
  - Milestone 0: form a team due Jan 22 (not marked)
  - Milestone 1: proposal due Feb 15
  - Milestone 2: mid-term report Mar 14
  - Milestone 3: report + demo (week of March 31 but due latest by April 2nd)

Prerequisites

- CS 240/240E is listed but not strictly necessary.
- Programming in a standard language: e.g., Python
- General interest in software systems, data-intensive application development and data management and processing systems