CMPT 606

Database concepts and Architecture

Dr. Abdelkarim Erradi

Department of Computer Science & Engineering

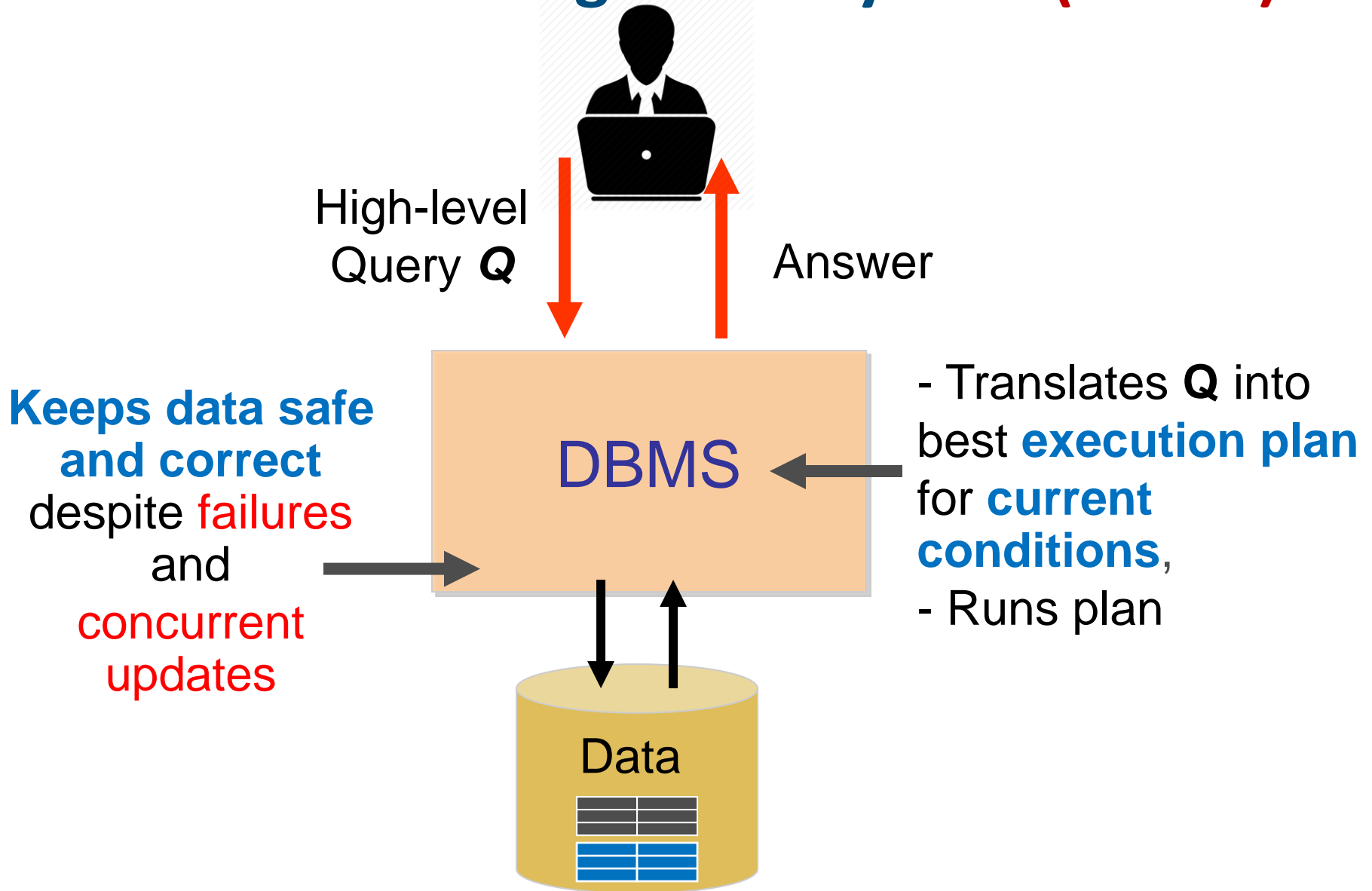**QU**

Outline

Why DBMS?

What is a database system?

- **Database**: Organized collection of **inter-related data** that models some aspect of the real-world
  - DB is core component of any software system

- **DataBase Management System (DBMS)**: a software system that allows the definition, **creation**, **querying**, update, and management of databases

DataBase Management System (DBMS)

High-level Query **Q**

Answer

Keeps data safe and correct despite failures and concurrent updates

DBMS

- Translates **Q** into best **execution plan** for **current conditions**,
- Runs plan

Data

Example Queries: At a Company

Query 1: Is there an employee named "Nemo"?
Query 2: What is "Nemo's" salary?
Query 3: How many departments are there in the company?
Query 4: What is the name of "Nemo's" department?
Query 5: How many employees are there in the
            "Accounts" department?

Employee

ID	Name	DeptID	Salary	...
10	Nemo	12	120K	...
20	Ali	156	79K	...
40	Fatima	89	76K	...
52	Saleh	34	85K	...
...	...	...	...	...

Department

ID	Name	...
12	IT	...
34	Accounts	...
89	HR	...
156	Marketing	...
...	...	...

Example: Store that Sells Cars

Owners of Honda Accord who are <= 23 years old

Make	Model	OwnerID	ID	Name	Age
Honda	Accord	12	12	Nemo	22
Honda	Accord	156	156	Ali	21

Join (Cars.OwnerID = Owners.ID)

Filter (Make = Honda and Model = Accord)

Filter (Age <= 23)

Cars

Make	Model	OwnerID
Honda	Accord	12
Toyota	Camry	34
Mini	Cooper	89
Honda	Accord	156
…	…	…

Owners

ID	Name	Age
12	Nemo	22
34	Fatima	42
89	Saleh	36
156	Ali	21
…	…	…

History

Pre-relational DB (70)

• Data stored and managed in files

Relational Database Systems (80)

• No redundancy in data storage

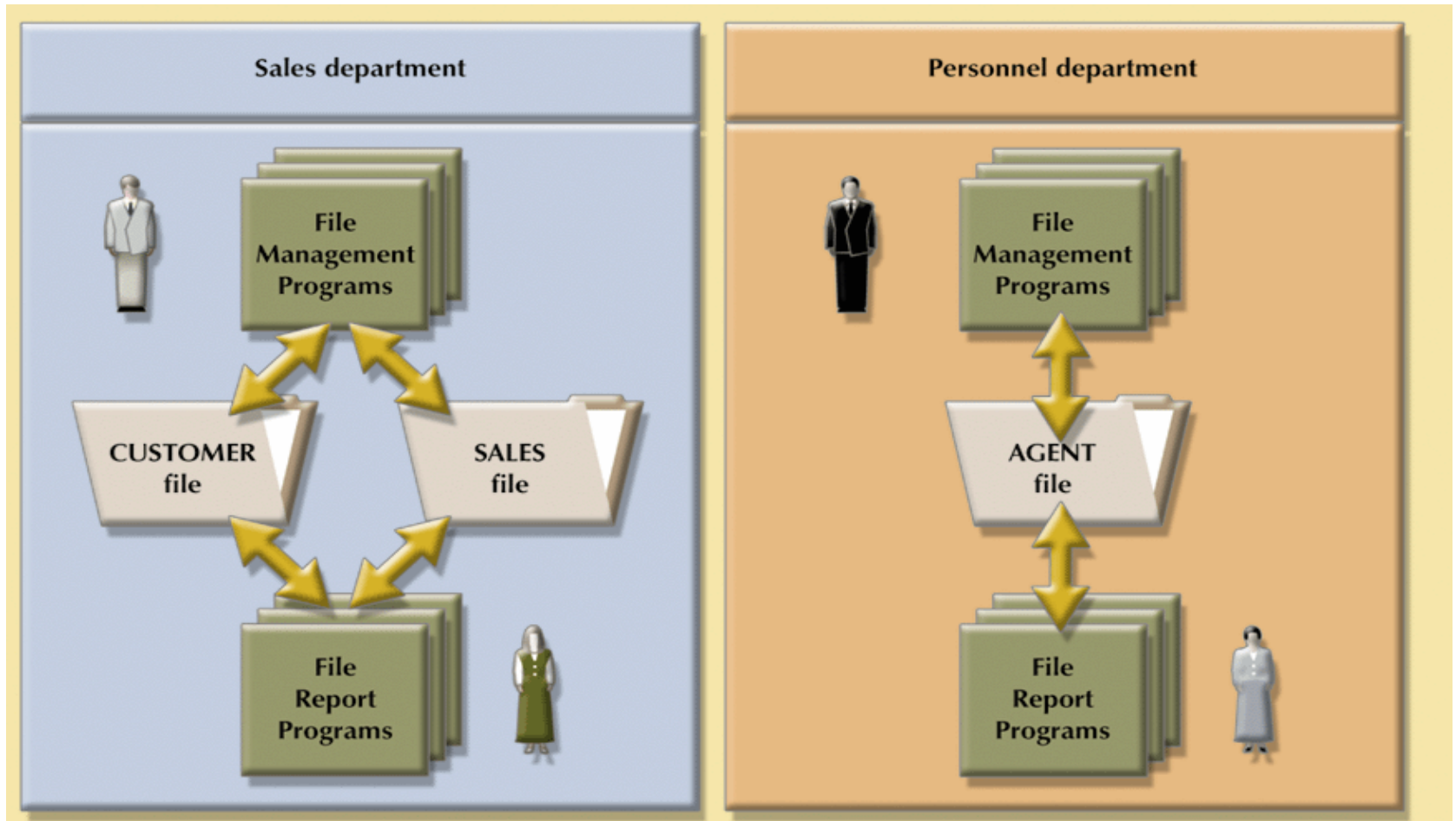• Multiuser operation and high performance

• ACID Properties

Post-relational Databases (90)

• Object-oriented Databases
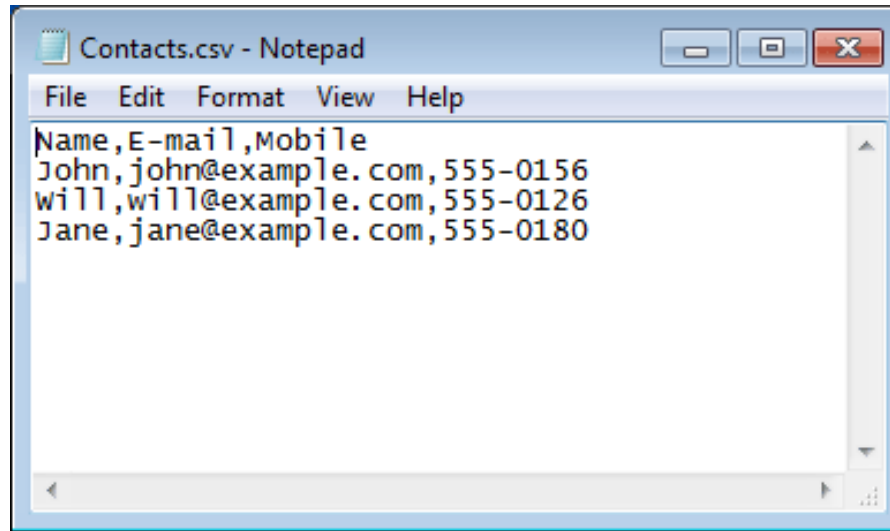
• Distributed databases

• Datawarehouses

NoSQL Movement (21st Centry)

=> Latest Trends: NoSQL, NewSQL and Cloud Data Services

A Simple File-based Data Management System

Data file physical organization

Contacts.csv - Notepad

```
Name,E-mail,Mobile
John,john@example.com,555-0156
Will,will@example.com,555-0126
Jane,jane@example.com,555-0180
```

- Store data as comma-separated value (CSV) files that we manage in our own code.

- Use a separate file per entity, e.g. Contact, Account

Query: What is John's phone number?

Answering Query using <u>Imperative</u> Approach

- What is John's phone number?

    => Write a simple script:

    – Scan through the accounts file

    – Look for the line containing "John"

    – Print out the phone number

- Query processing tricks when having thousands contacts:

    – Cluster contacts: Those with names starting with "A…" go into file A; those starting with "B…" go into file B; etc.

    – Keep the contacts sorted by name

    – Hash contacts according to the name

    – And the list goes on…

Observations

- To write correct code, application programmers need to know **how data is organized physically** (e.g., row layout inside the file)
  - Burden on programmer to figure out right tricks to retrieve the data fast
- Tons of tricks (not only in storage and query processing, but also in concurrency control, recovery, etc.)
- Different tricks may work better in different usage scenarios
- Same tricks get used over and over again in different applications

Drawbacks of using file systems to store data

- To retrieve data from a file system, extensive programming is often needed - both ***what*** and ***how*** are programmer's responsibility

- *Ad hoc queries* require programming

- Each file must have its own file-management program to create the file structure, add data to file, delete data from it, modify it and list its contents

- All data access programs are subject to change when the file structure changes (e.g., a field is deleted or its position is changed)
  - ***Structural dependency***

- Even a change in the data type of a field (e.g., integer to real) requires all data access programs to change
  - ***Data dependency***

# Drawbacks of using file systems to store data

- Data redundancy and inconsistency

    - Multiple file formats, duplication of information in different files

- Difficulty in accessing data

    - Need to write a new program to carry out each new task

- Data isolation: multiple files and formats => difficulty to make joins

- Integrity problems

    - Integrity constraints (e.g. account balance > 0) become part of program code

    - Hard to add new constraints or change existing ones

# Drawbacks of using file systems (cont.)

- Hard to support **Atomicity** of updates
  - Failures may leave database in an inconsistent state with **partial** updates carried out
  - e.g. transfer of funds from one account to another should either complete or not happen at all
- Hard to support **Concurrent access** by multiple users
  - Concurrent accessed needed for performance
  - Uncontrolled concurrent accesses can lead to inconsistencies
    - E.g. two people reading a balance and updating it at the same time
- Security problems

=> Database systems offer solutions to all the above problems

DBMS vs. File Systems?
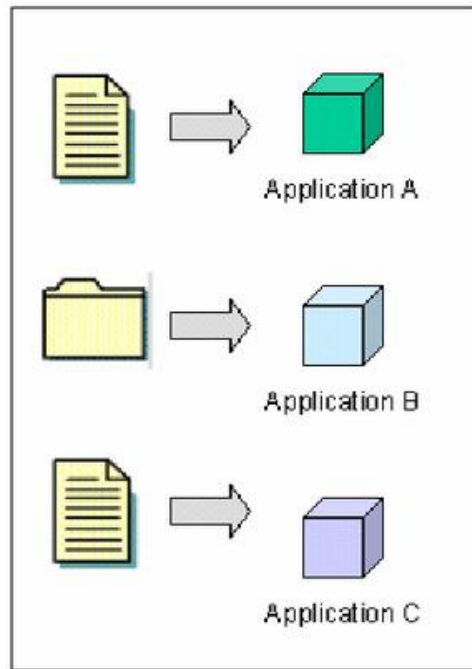
Independent files
(partial redundancy
and incoherence of data)
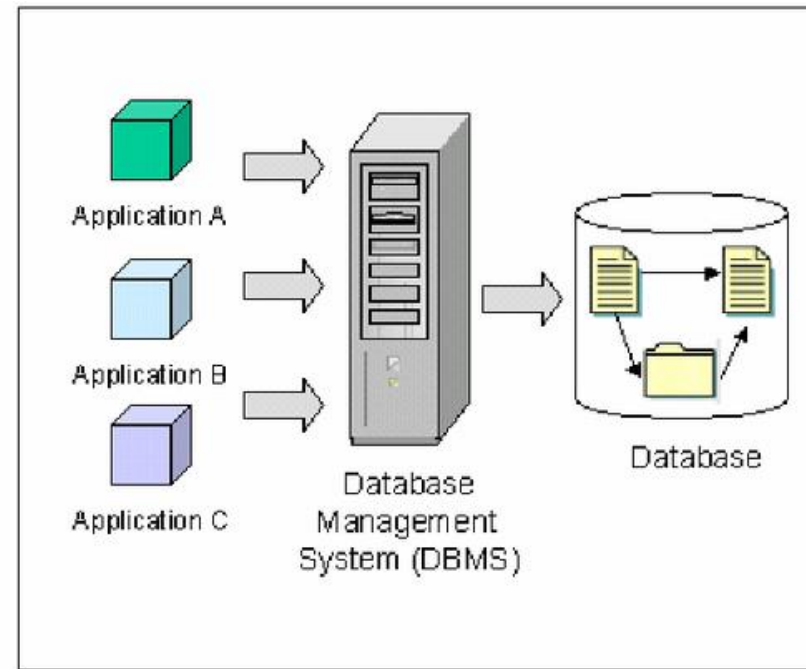
Integrated database
+ DBMS
(Integration and non-redundant data)

- Database consists of logically related data **stored in a single repository**
- Provides advantages over file system management approach
  - Eliminates inconsistency, data anomalies, data dependency, and structural dependency problems
  - Stores data structures, relationships, and access paths (i.e., indexes)

DBMS added value

- The developer has only 2 responsibilities:
  - **Data model**: Used to specify how data are conceptually structured
  - **Query language**: Used to specify data processing/management tasks

- DBMS takes care of:
  - **Physical data organization**: Store and index data in smart ways to speed up access
  - **Query processing and optimization**: Figure out the most efficient method to carry out a given task

The birth of DBMS

The relational revolution (1970's)

- **A simple data model**: data is stored in relations (i.e., tables)

- **A declarative query language for data access**: SQL

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

- 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

Goal of Data Management and Storage
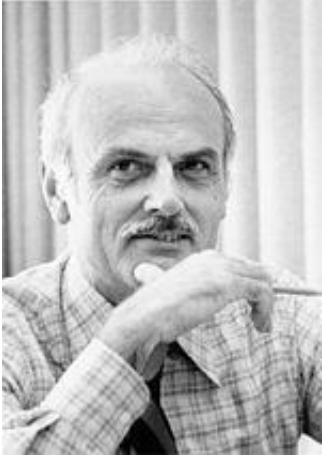
"Future users of large data banks must be protected from **having to know how the data is organized** in the machine (the internal representation)."
- Edgar Frank Codd, 1970

This **Physical Data Independence** principle if the one of the most important reason behind the success of DBMS today
- Edgar Frank **Codd** got a Turing Award for this

Physical data independence

- Applications should not need to worry about how data is physically structured and stored

- The application should not be affected by changes of the physical storage of data:
  - Adding/removing indexes
  - Physical organization on disk
  - Physical plans for accessing the data
  - Parallelism: multicore, distributed

- Applications should work with a **logical data** model and **declarative query language**
  - **Specify what you want, not how to get it**
  - Leave the implementation details and optimization to the DBMS

DB Levels of Abstraction

Views hide details of the underlying tables (e.g., hide employee's salary for security purposes)

view level

| view 1 | view 2 | ... | view *n* |
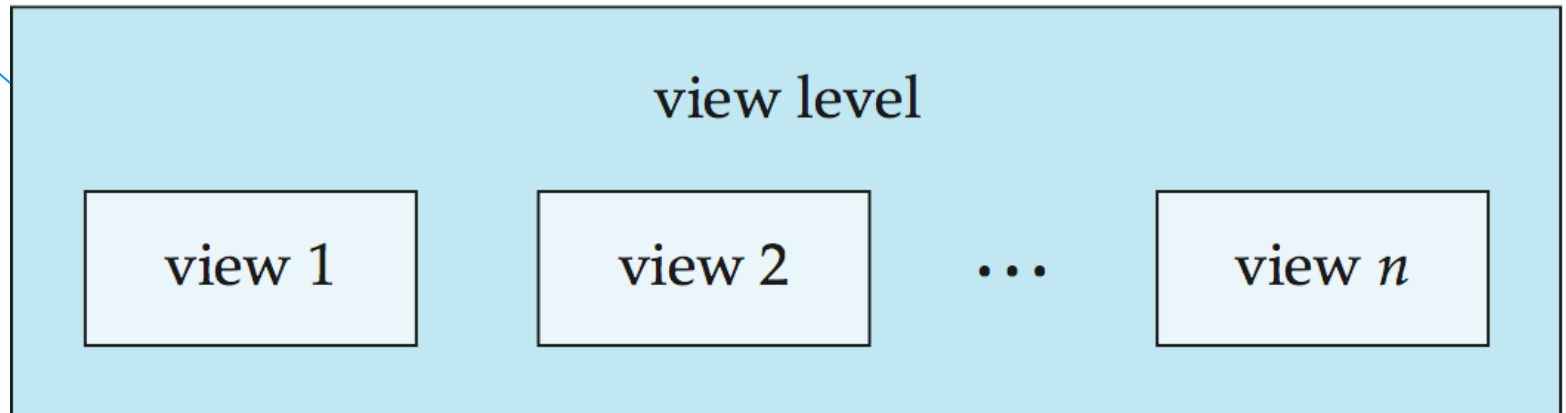
logical level

describes data stored in database, and the relationships among the data.

physical level

DBMS takes care of this level
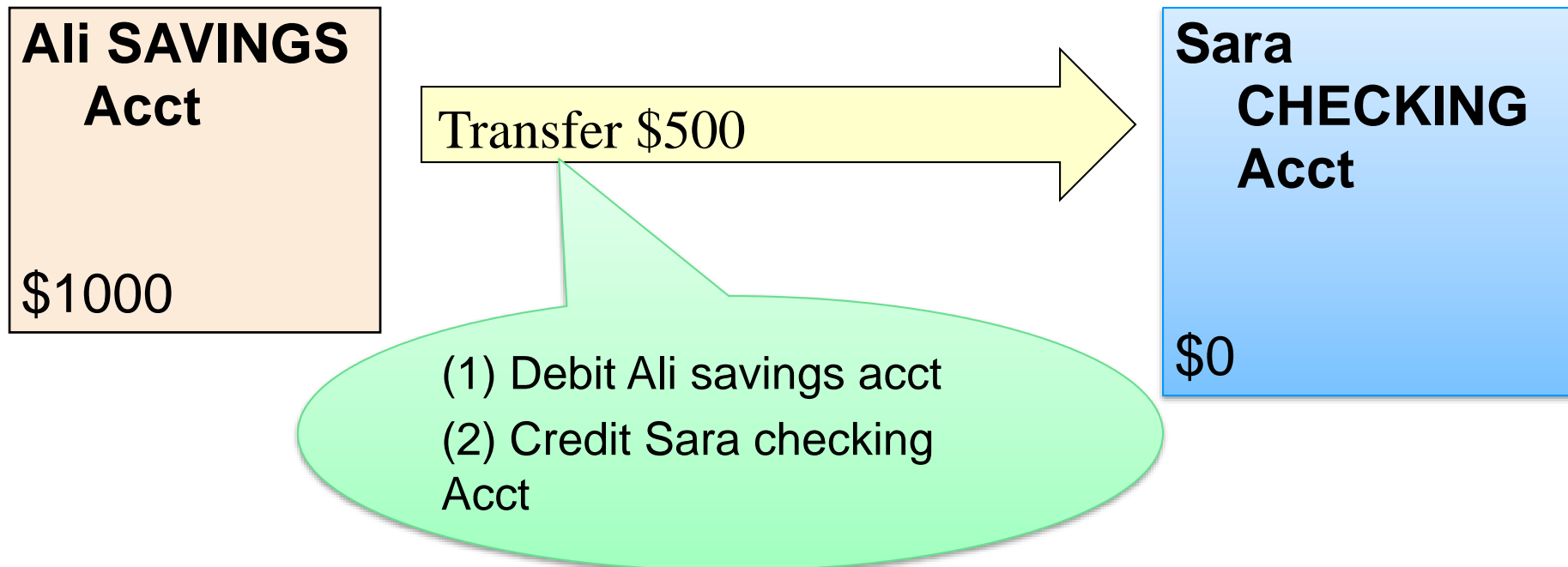
describes how a record (e.g., customer) is stored on disk

ACID Guarantees

A transaction is a sequence of operations that must be executed as a whole

Ali SAVINGS Acct

$1000

Transfer $500

Sara CHECKING Acct

$0

(1) Debit Ali savings acct
(2) Credit Sara checking Acct

Either both (1) and (2) happen or neither!

Every DB action takes place inside a transaction

ACID Guarantees

- **Atomicity**: Everything in a transaction succeeds or the entire transaction is rolled back (All or Nothing)

- **Consistency**: data inserts/updates/deletes do not violate any defined rules such as constraints

- **Isolation**: Transactions cannot interfere with each other => The updates of a transaction must not be made visible to other transactions until it is committed

- **Durability**: Updates made from a committed transaction are persistent despite failures (e.g., power loss, crashes, or errors)

Example of consistency issues caused by concurrent updates

- **Example to illustrate consistency issues that can be introduced by concurrent updates:**

Get account balance from database;
If balance > amount of withdrawal then
    balance = balance - amount of withdrawal;
    dispense cash;
    store new balance into database;

- Ali at ATM_1 withdraws $100

- Sara at ATM_2 withdraws $50

- Initial balance = $400, final balance = ?

  – Should be $250 no matter who goes first

Sequential Transactions -> Final balance = $250

Ali withdraws $100:

read balance; $400
if balance > amount then
    balance = balance - amount; $300
    write balance; $300

Sara withdraws $50:

read balance; $300
if balance > amount then
    balance = balance - amount; $250
    write balance; $250

Concurrent Transactions (Scenario 1) -> Final balance = $300

Ali withdraws $100:

Sara withdraws $50:

read balance; $400

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

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

Concurrent Transactions (Scenario 2) -> Final balance = $350

Ali withdraws $100:

Sara withdraws $50:

read balance; $400

read balance; $400

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

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

Example of consistency issues caused by failures

- **Example to illustrate consistency issues that can be introduced by failures:**

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 instruction
  - **Such failures may leave database in an inconsistent state with partial updates carried out**
  - Transfer of funds from one account to another should either complete or not happen at all

=> Database transactions come to the rescue!
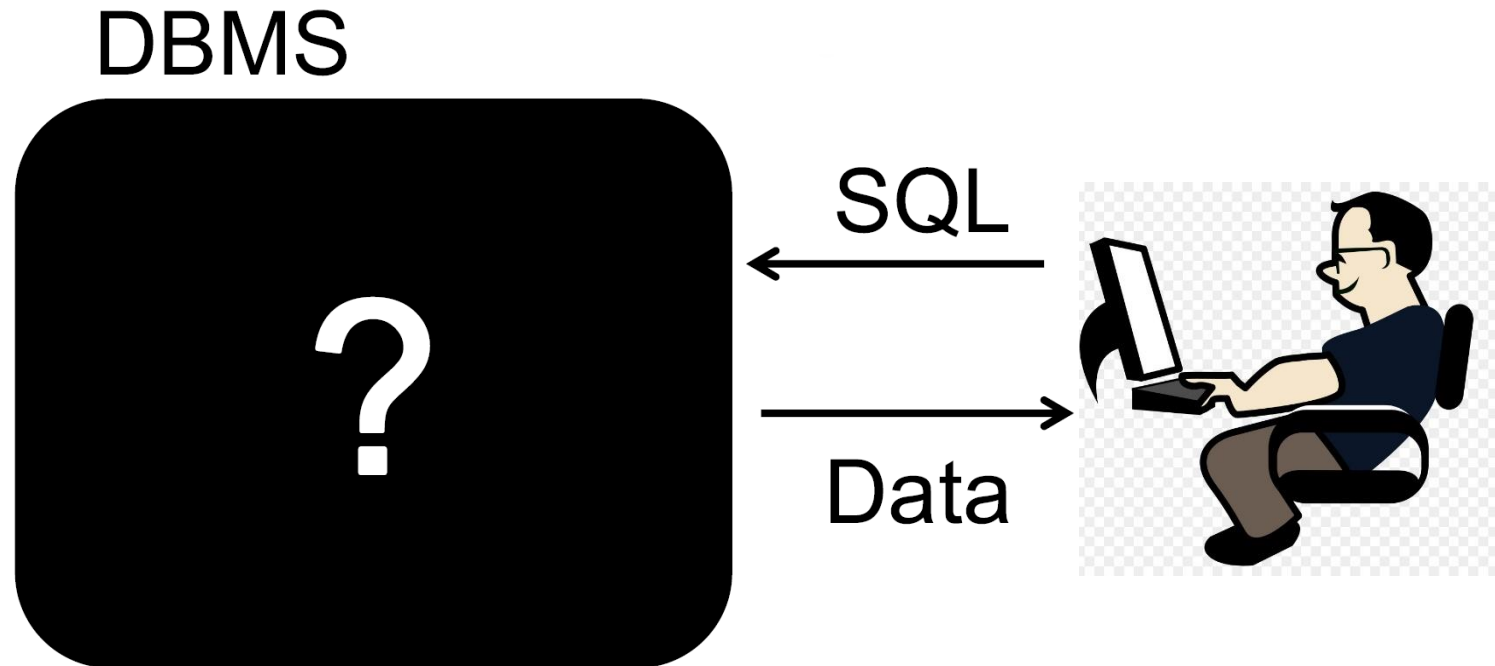
ACID Summary

- **Atomicity:** "all or nothing"

- **Consistency:** "it looks correct to me"

- **Isolation:** "as if alone"
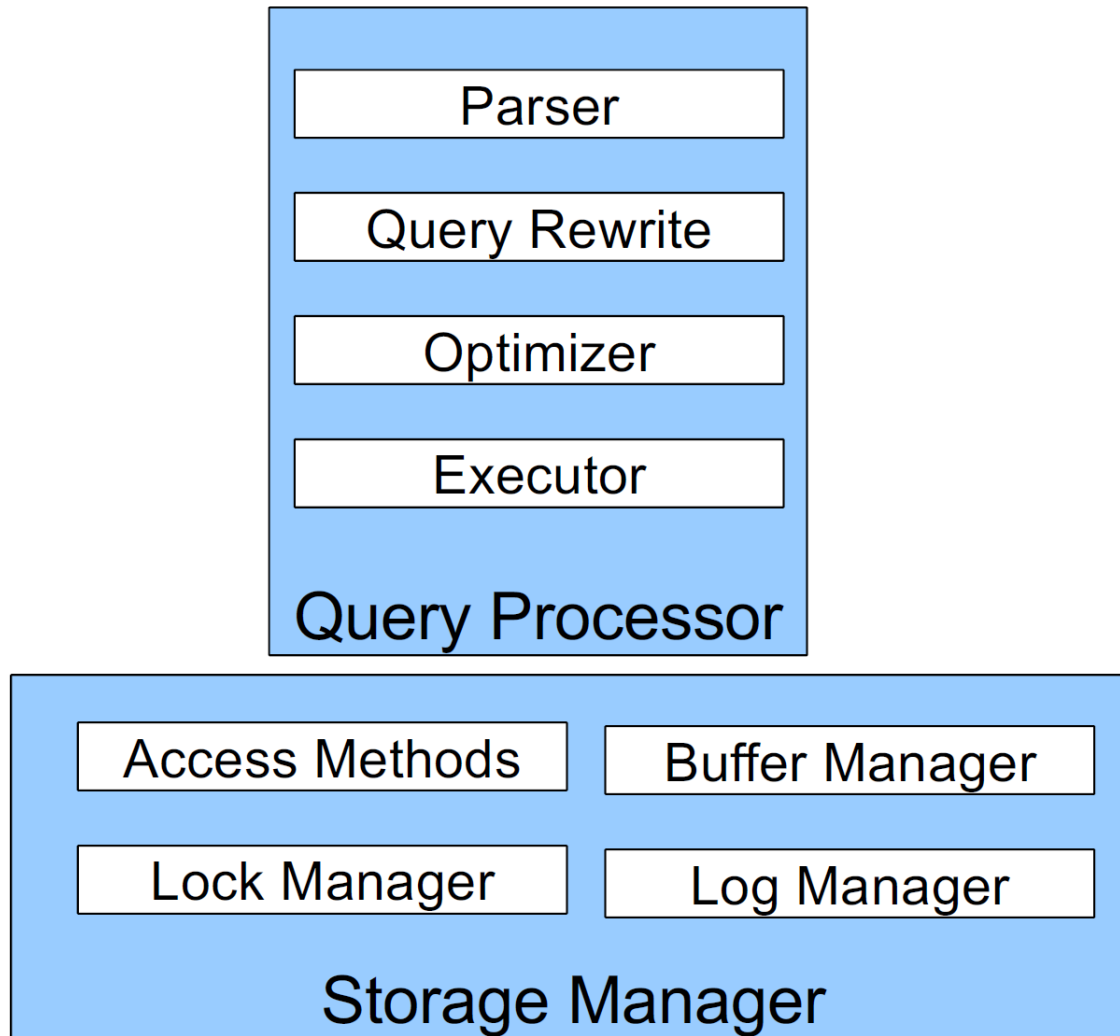
- **Durability:** "survive failures"

DBMS Architecture

How to Implement a Relational DBMS?

Key challenges:

- Achieve high performance on a large database

- Handle ACID guarantees

DBMS

?

SQL

Data

DBMS Architecture

Query Processor

- Parser
- Query Rewrite
- Optimizer
- Executor

Storage Manager

- Access Methods
- Buffer Manager
- Lock Manager
- Log Manager

[Anatomy of a Db System. J. Hellerstein & M. Stonebraker. Red Book. 4ed.]

Query Processing

1. **Parser**: Parsing and translation of the query to a relational-algebra expression
2. **Optimizer**: Select the optimal query execution plan
3. **Executor**: Execution of selected query plan
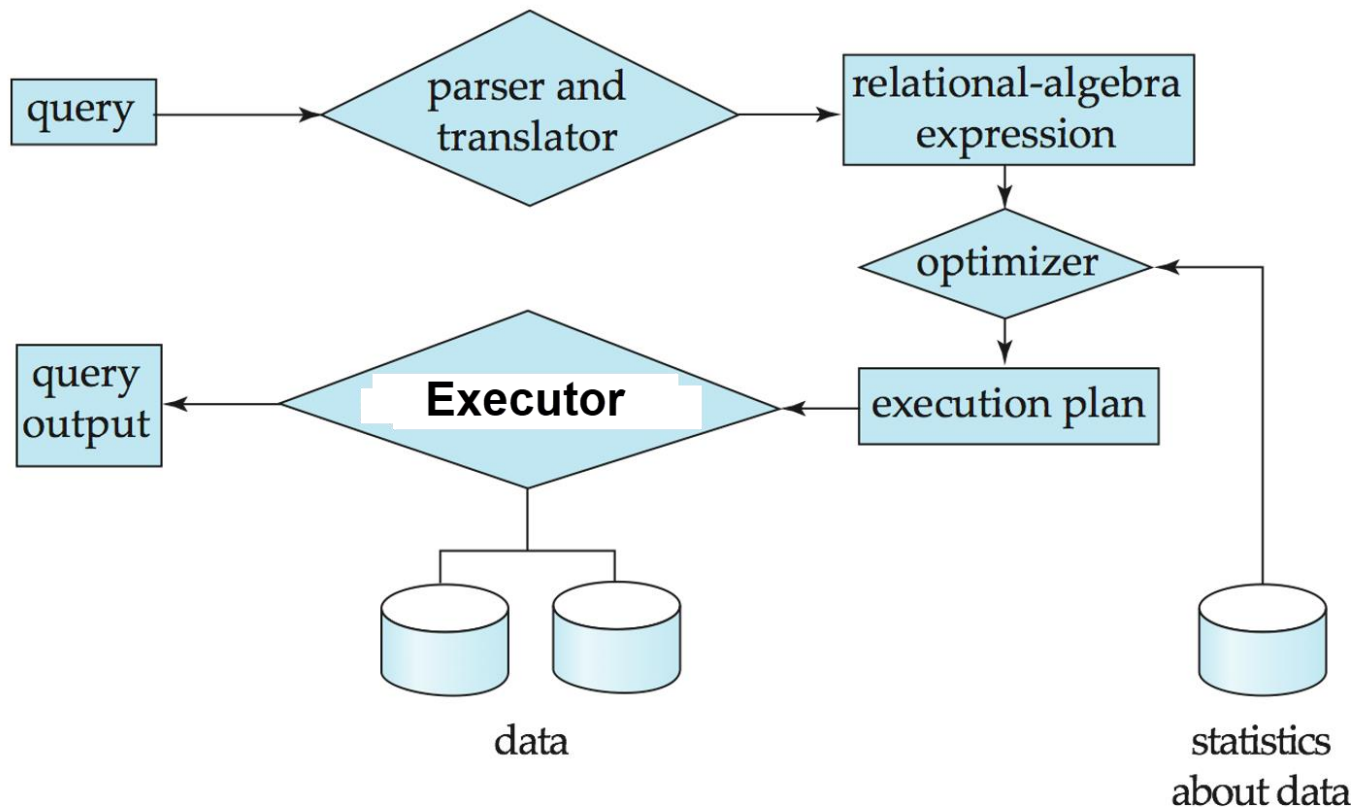
Query Optimizer

- Find an efficient query plan for executing the query. A **query plan** defines:

  o Access method to use for each relation (Use a file scan or use an index)

  o Implementation to use for each relational operator (Implement a join using a Nested loop)

Physical Query Plan - Example

```
SELECT S.sname
FROM Supplier S, Supplies U
WHERE
S.scity='Seattle'
AND S.sstate='WA'
AND S.sno = U.sno
AND U.pno = 2;
```

(On the fly) π_{sname}

(On the fly) $\sigma_{\text{sscity='Seattle'} \wedge \text{sstate='WA'} \wedge \text{pno=2}}$

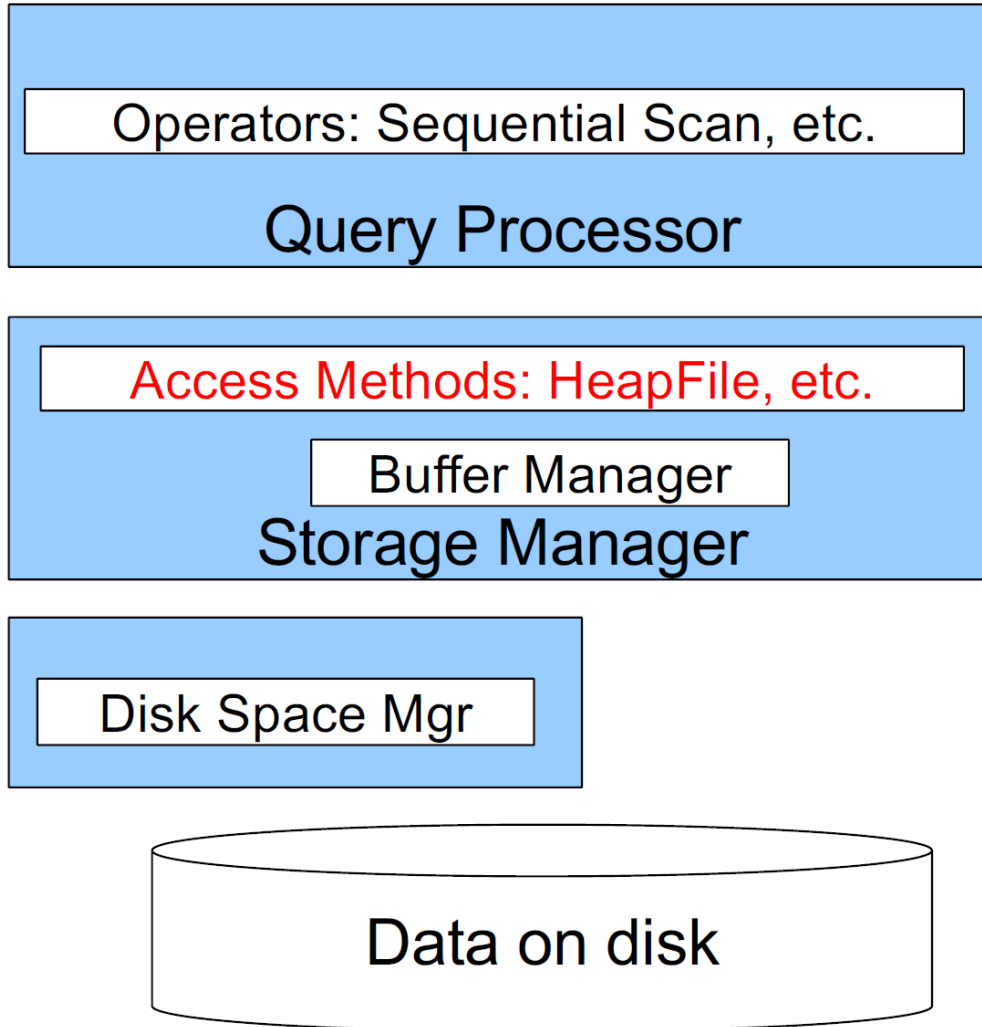(Nested loop)   ⋈ sno = sno

Suppliers
(File scan)   Supplies
                     (File scan)

Storage Management

- The storage manager is responsible of:
  - Interaction with the file manager
  - **Efficient storing, retrieving and updating of data**

- Issues addressed:
  - Storage access
  - File organization
  - Indexing and hashing

Storage Management

Operators: Sequential Scan, etc.

Query Processor

Access Methods: HeapFile, etc.

Buffer Manager

Storage Manager

Disk Space Mgr

Data on disk

- **Access Methods**:

Organize data to support fast access to desired subsets of records:

- Organize pages in DB *files*

- Organize data inside pages

- Provide an API for operators to access data in these files

- **Buffer Manager**:
  - Caches data in memory
  - Reads/writes data to/from disk as needed

- **Disk Space Manager**:

- Allocates space on disk for data files/indexes

Transaction Management

- **Lock Manager:** manages **concurrent data access.** Controls the interaction among the concurrent transactions, to ensure the consistency of the database (when multiple users concurrently update the same data)

- **Log Manager:** manages recovery to ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures

Summary of Relational Database Features

- Storage and access of data that is Concurrently accessed, Consistently modified, Structured (tabular) and Fast to access

- Logical data model + declarative queries and updates **=> physical data independence**
  - Provides a declarative interface to data management **=> Hides complexity and increases flexibility**

- ACID Guarantees to handle:
  - Multi-user concurrent access
  - Safety from system failures

- But RDBMS is optimized for single server
  - Lack the capacity of scale-out (to a cluster of commodity servers)
  - NoSQL and NewSQL movements address these limitations

NoSQL and NewSQL Databases

NoSQL Databases

- NoSQL highly available and scalable distributed data management systems designed to address performance and scalability requirements which cannot be addressed by traditional relational databases
  - Has built-in data partitioning and replication
  - Horizontally scalable (distributed DB on a cluster of servers) as opposed to vertical scaling in RDBMS.
  - Availability and scaling have highest priority with relaxed consistency (sacrificing strict ACID)
  - Sacrificing strong consistency and opting for weak consistency
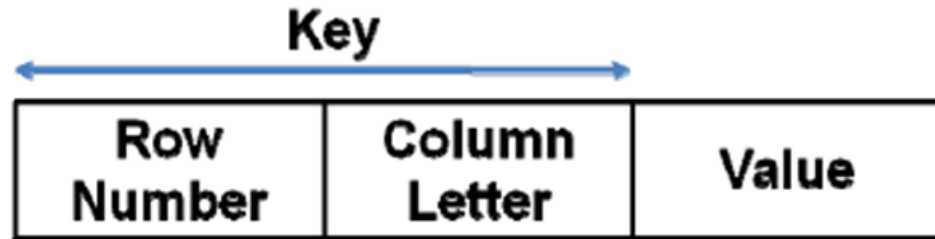
NoSQL Taxonomy

Conceptual Structures:

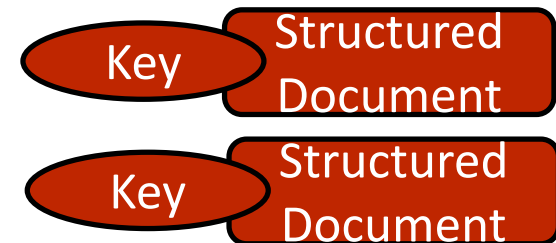**Key Value Stores**
    Schema-less system

Key Value

Column Family databases
    key is mapped to a value that is
    a set of columns. Store columns
    of data together, instead of rows.
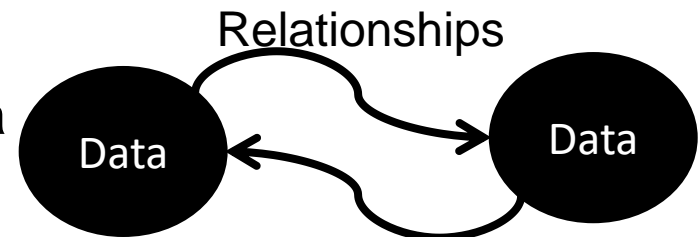
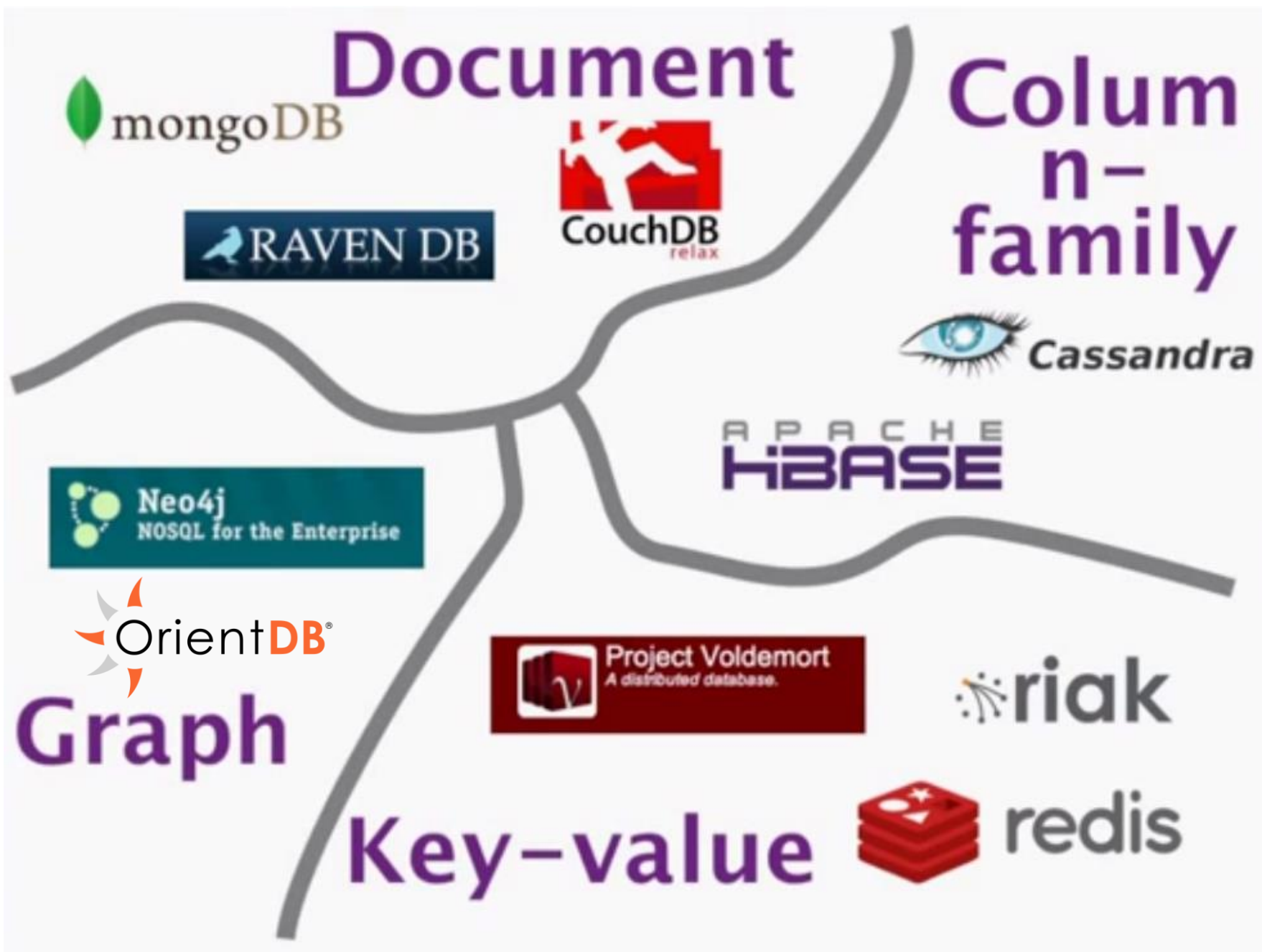|	Key	
Row Number	Column Letter	Value

Document Oriented Database
    Stores documents that are semi-structured

Key Structured Document

Key Structured Document

Graph Databases
    Uses nodes and edges to represent data

Relationships

Data Data

Relational vs. Aggregate-oriented - an Example

ID: 1001
customer: Ann

line items:

0321293533	2	$48	$96
0321601912	1	$39	$39
0131495054	1	$51	$51

payment details:

Card: Amex
CC Number: 12345
expiry: 04/2001

orders

customers

order lines

credit cards
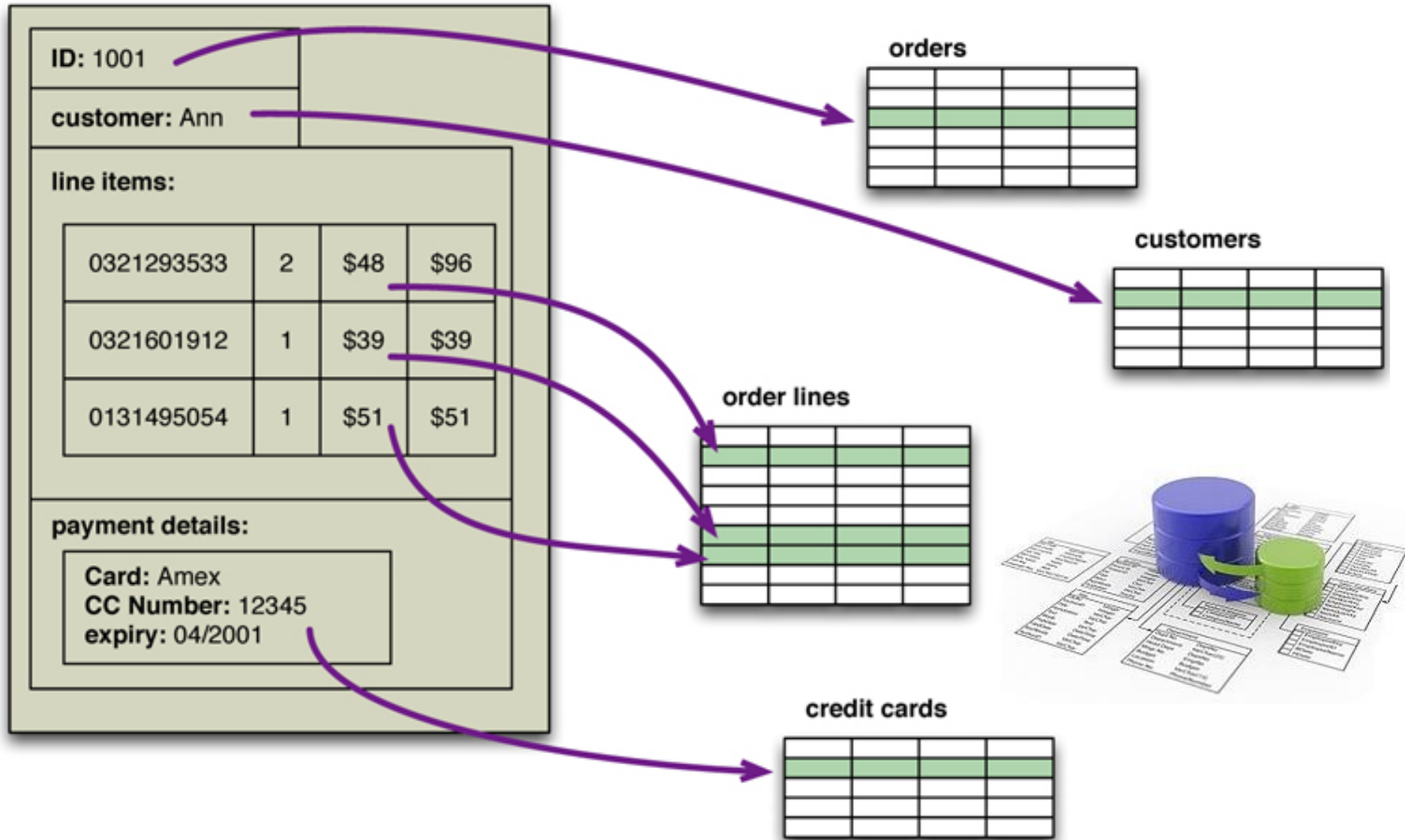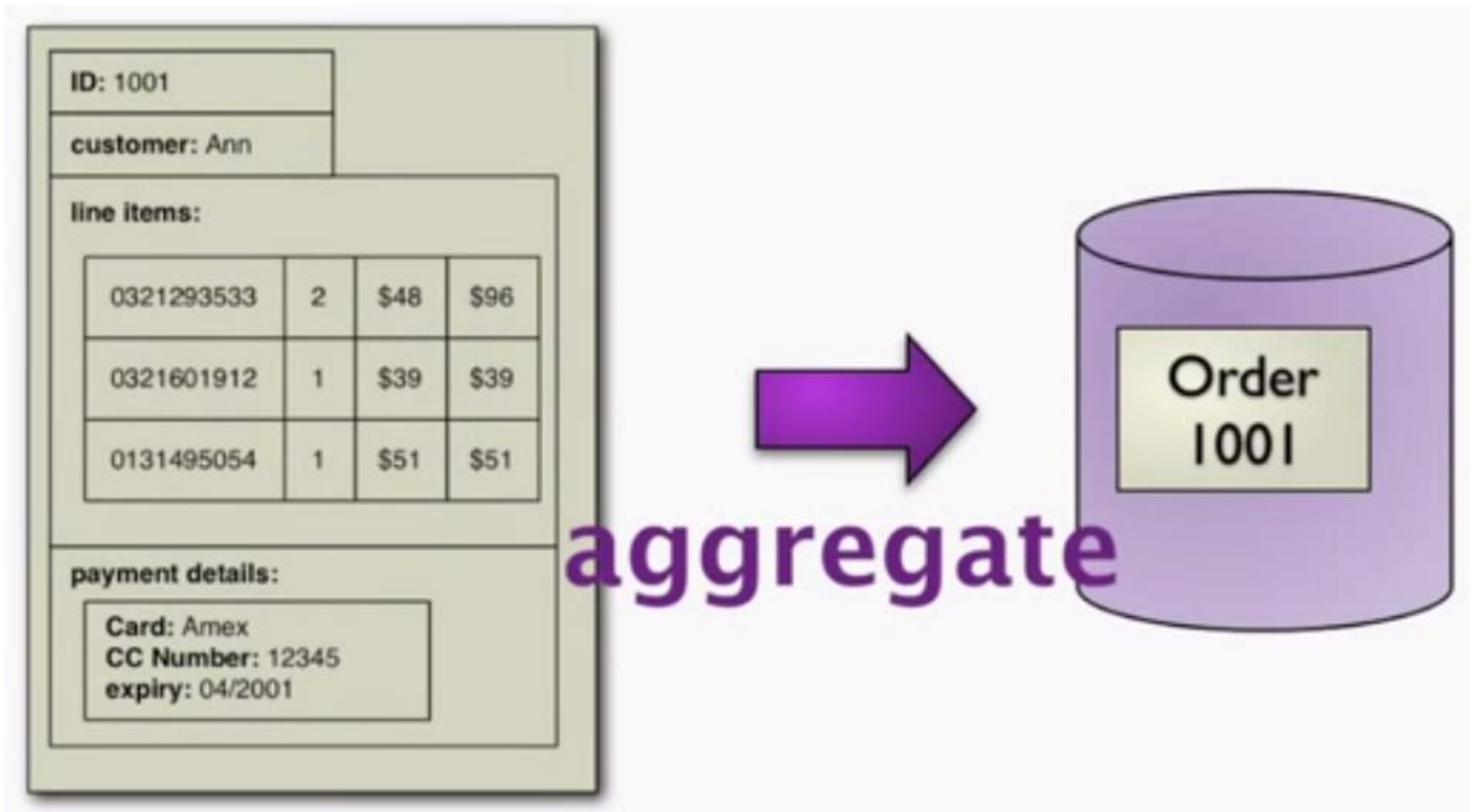
Aggregate Example

- Aggregate brings **cluster friendliness** as a whole aggregate can be stored in one node of the cluster

Colum-family examples

Key	Tweets/00000000-0000-0000-0000-000000000001	
Data		
	Application	TweetDeck
	Private	true
	Text	Err, is this on?
Key	Tweets/00000000-0000-0000-0000-000000000002	
Data		
	App	TweetDeck
	Public	true
	Text	Well, I guess this is my mandatory hello world
	Version	1.2

Keys | Columns

Peter

Name	Number	Mobile Phone
Peter...	234786459	994398909

Joseph

Name	Number
Joseph	234786459

Example Graph Model
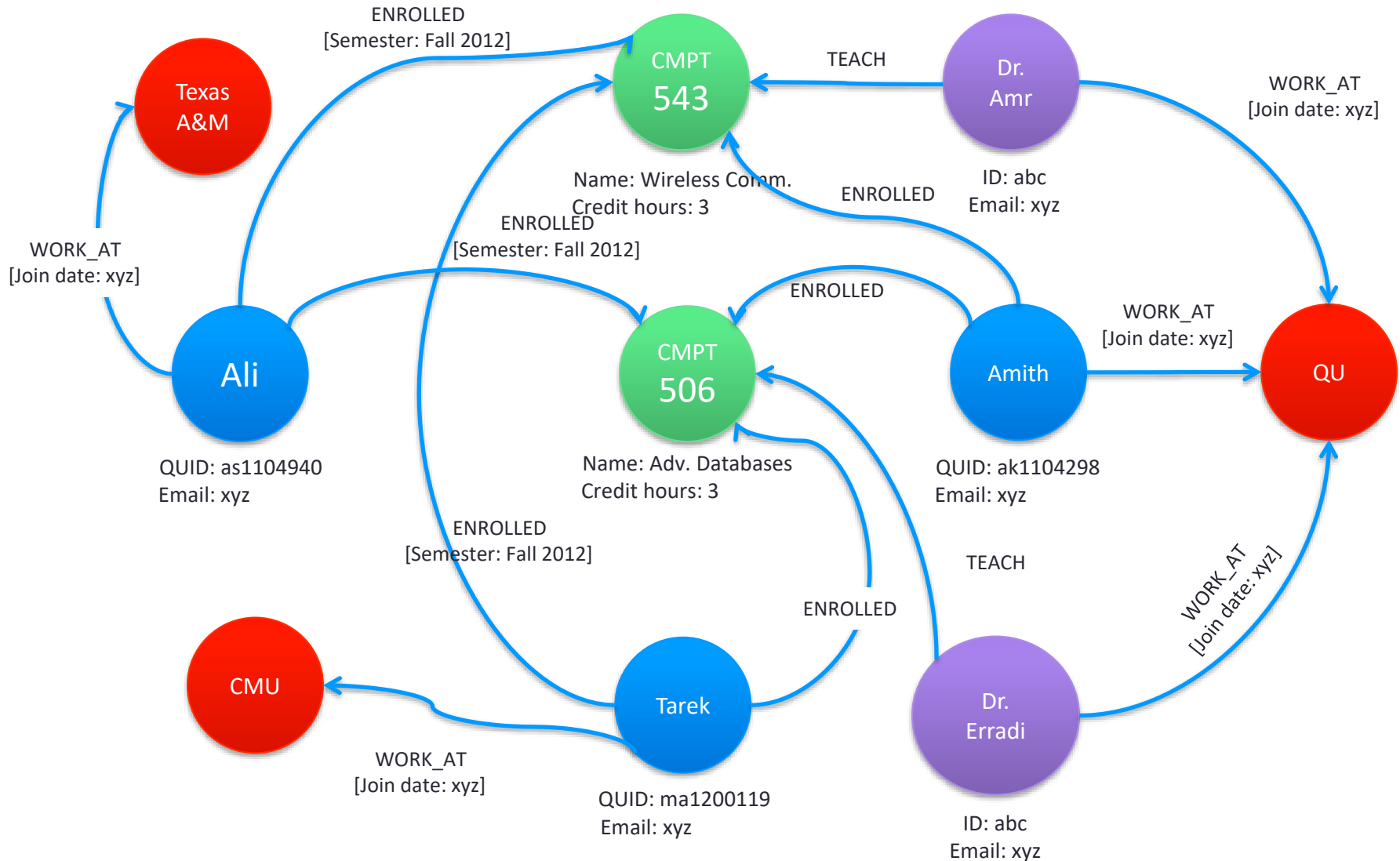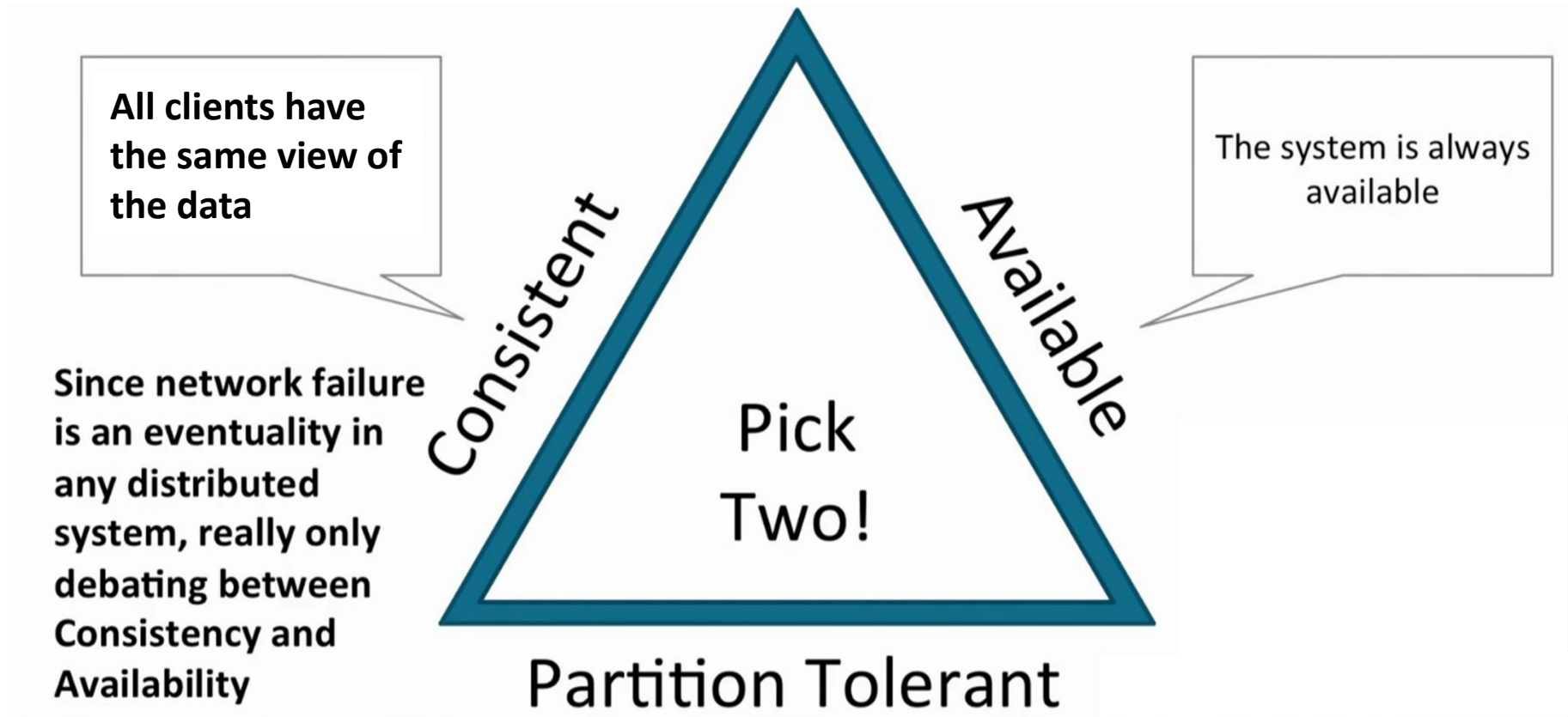
Important Design Goals

- **Scale out: designed for scale**
  - Horizontal scaling on commodity hardware
  - Low latency updates
  - Sustain high update/insert throughput

- **High availability** (as downtime implies lost revenue)
  - Replication (with peer to peer replication)
  - Automated failure recovery

BASE instead of ACID

- BASE is the most notable weak consistency model used by NoSQL systems:

  - **<u>B</u>asically <u>A</u>vailable**: the system responses basically all the time even if some nodes fail (thanks to replication). Performance may degrade but the system remains available.

  - **<u>S</u>oft-state**: is not necessary to be always consistent

  - **<u>E</u>ventual consistency**: has to come to a consistent state eventually

- BASE is based on Brewer's **CAP theorem**

CAP-Theorem

- When data is partitioned, in case of network/node failure, we can only maintain **consistency** or **availability** but NOT both at the same time

All clients have the same view of the data

The system is always available

Since network failure is an eventuality in any distributed system, really only debating between Consistency and Availability

Consistent

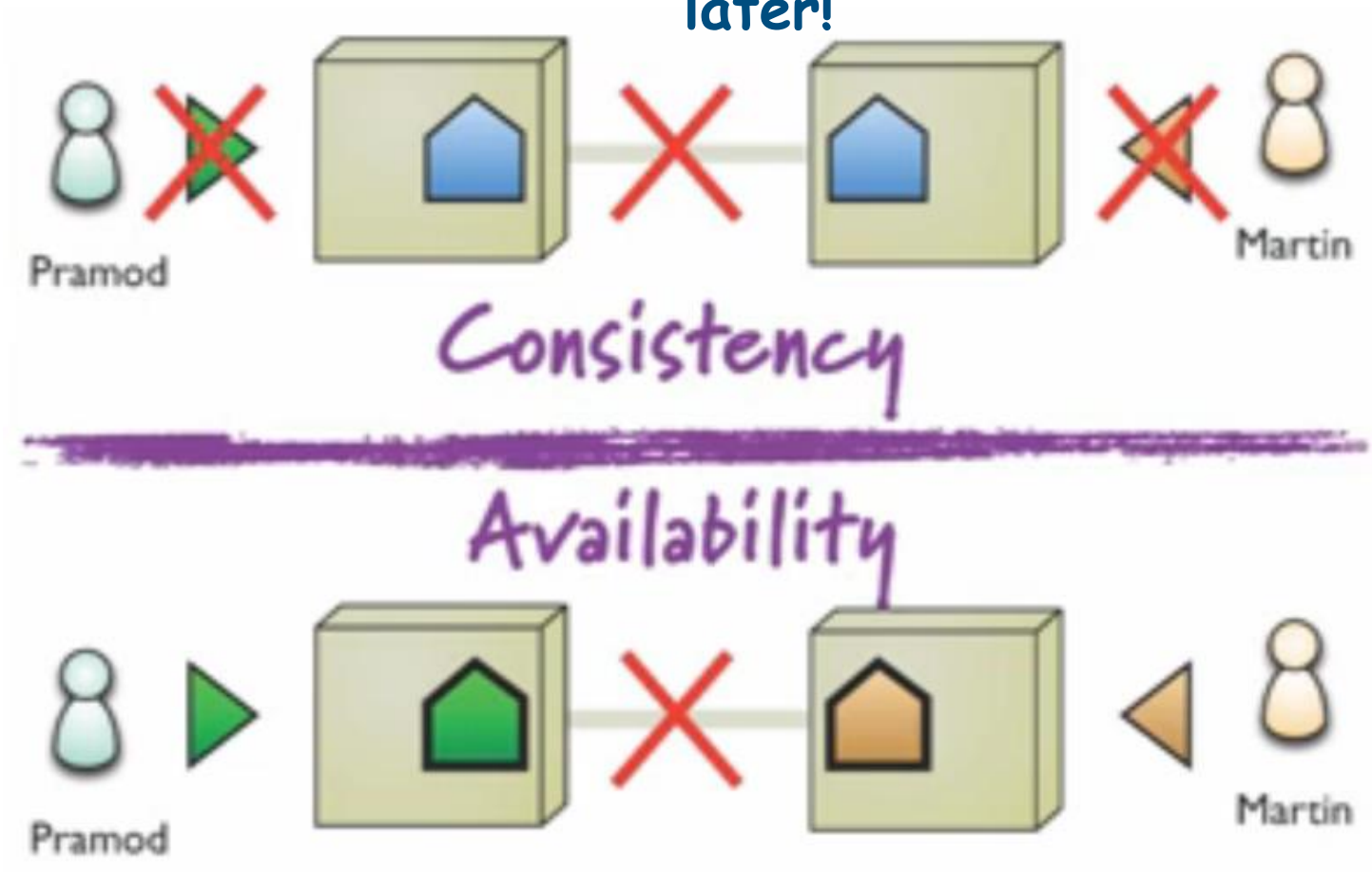Available

Pick Two!

Partition Tolerant

Booking Last Room Scenario by two customers interacting with replicated data in 2 different servers

But the network connection between the two nodes is lost... what is the solution?
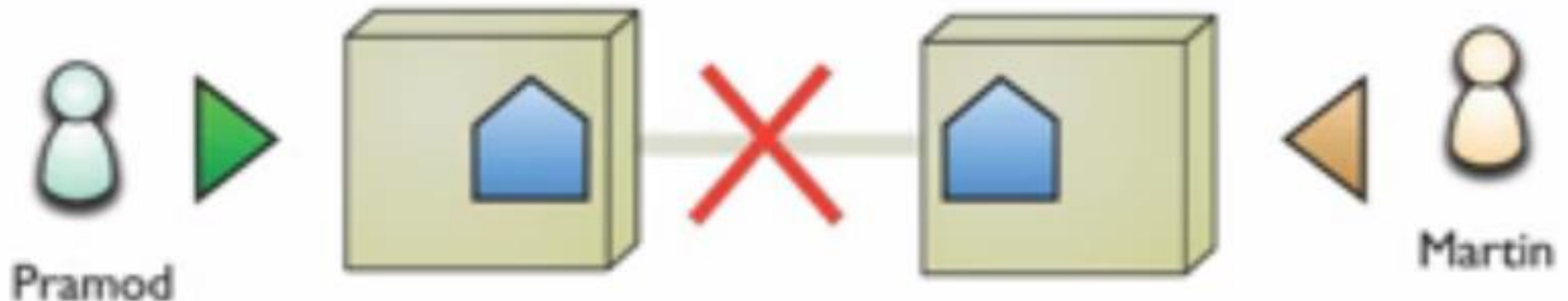
Option 1: Preserve Consistency (not book the room twice) by saying to the customers 'System is down please try again later!'

Consistency

Availability

Option 2: Sacrifice consistency (let the room be booked twice) but get higher availability

NewSQL

- NewSQL is a class of database systems that aims to provide the scalability of NoSQL systems while still maintaining the ACID guarantees of a traditional single-node database system.

(e.g., VoltDB, Google Spanner, MemSQL, NuoDB, and TokuDB)

- When should you use NewSQL?
  - When the application needs to handle very large datasets or a very large number of short-lived transactions
  - When ACID guarantees are required (e.g., manage critical data which requires safety guarantees such as financial and order processing systems)
  - When the application can significantly benefit from the use of the relational model and SQL

NewSQL Database Features

1. Support the relational data model

2. Use SQL as the primary mechanism for application interaction

3. ACID support for transactions

4. A non-locking concurrency control mechanism so real-time reads will not conflict with writes, and thereby cause them to stall

5. A scale-out architecture, capable of running on a large number of nodes without bottlenecking

Conclusion – No1DB

- Database systems differ in their **data model**, **querying** capabilities/languages and **approaches to scale**

- Relational databases:
  - Optimal for single machine
  - Strong ACID guarantees
    - Locking-based concurrency control mechanisms
    - Log-based recovery

- Distributed databases:
  - Scale-out using automated partitioning and replication
  - Distributed across a cluster of machines
  - NoSQL
    - Focused on low latency, trading consistency for **scalability**
    - Move away from ACID properties
    - Come in several data models and query languages

  - NewSQL
    - A return to the relational model, but **re-architected for modern hardware** and web scale use cases
    - Maintain ACID properties
    - Uses the relational model and SQL