CMPT 606

Database concepts and Architecture

Dr. Abdelkarim Erradi

Department of Computer Science & Engineering

QU

Outline

1. Why DBMS?
2. ACID = Gold Standard For RDBMS Design
3. DBMS Architecture
4. NoSQL and NewSQL Databases

Why DBMS?

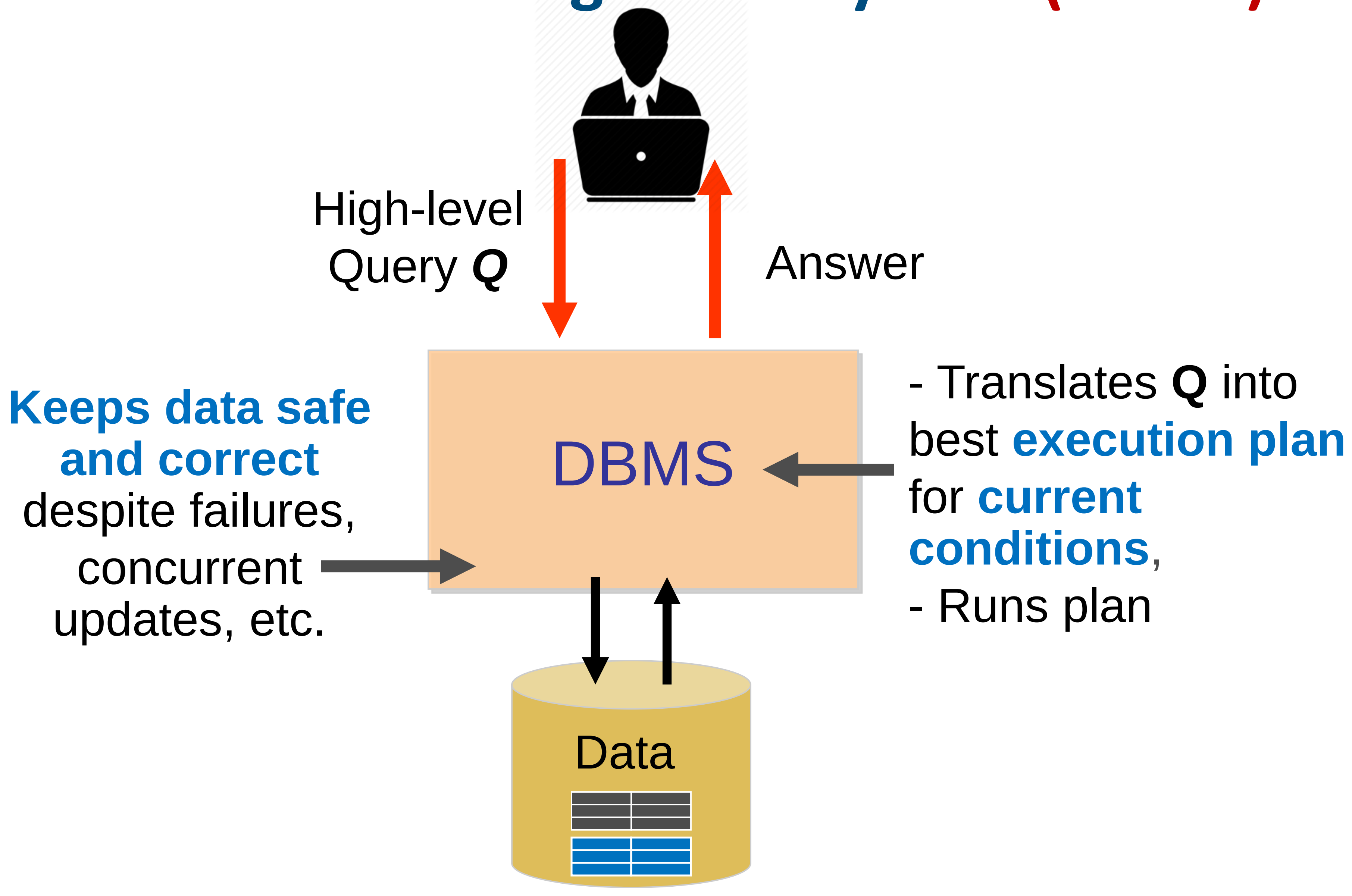
< Back

What is a database system?

From Oxford Dictionary:

- Database: an organized body of related information
- DataBase Management System (DBMS): a software system for providing **efficient**, **convenient**, and **safe storage** of and **multi-user access** to, possibly **massive**, amounts of **persistent** data

DataBase Management System (DBMS)



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 mananed in files

Relational Database Systems (80)

- No redundancy in data storage
- Multiuser operation and high performance
- ACID Properties

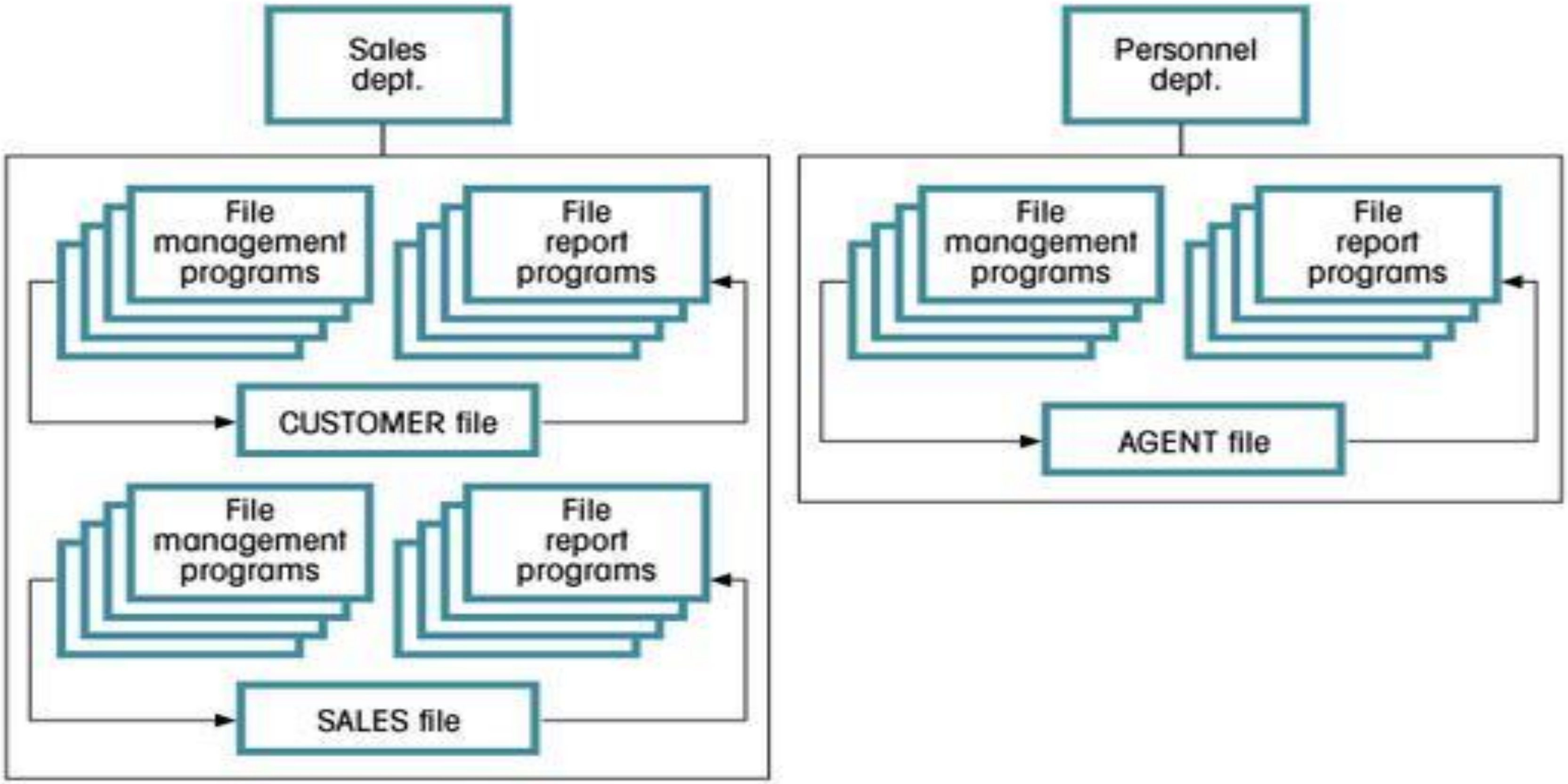
Post-relational Databases (90)

- Object-oriented Databases
- Distributed databases
- Datawarehouses

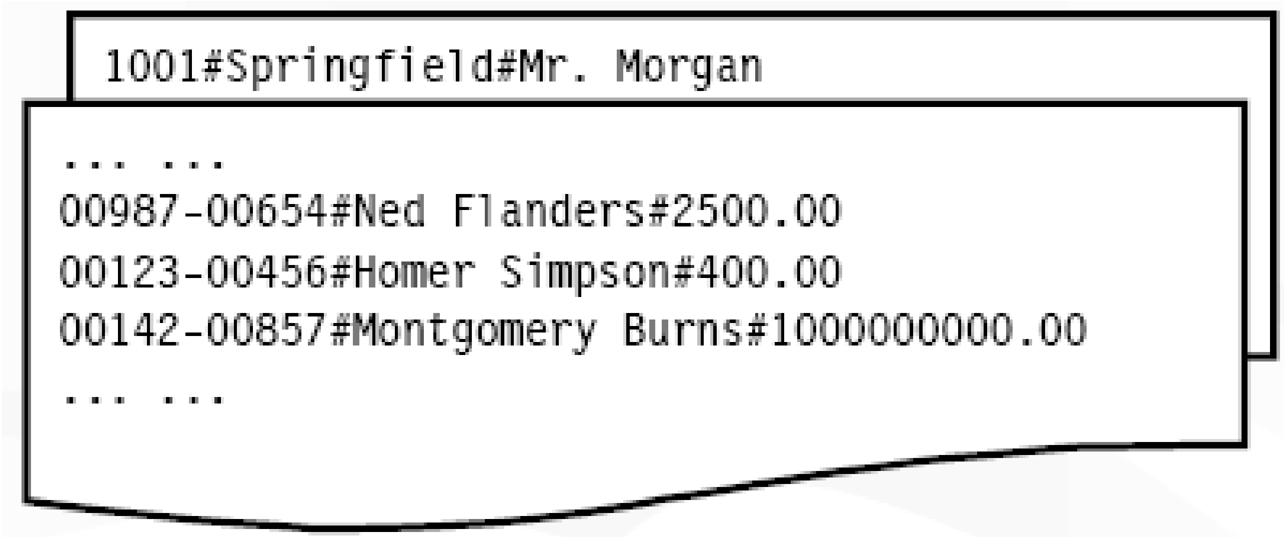
NoSQL Movement (21st Centry)

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

A Simple File System



Data file physical organization



- ASCII file
- Accounts/branches separated by newlines
- Fields separated by #'s

Query: What's the balance in Homer Simpson's account?

Answering Query using Imperative Approach

- What's the balance in Homer Simpson's account?

  => A simple script:

  – Scan through the accounts file
  – Look for the line containing "Homer Simpson"
  – Print out the balance
- Query processing tricks when having thousands accounts:
  – Cluster accounts: Those owned by "A..." go into file A; those owned by "B..." go into file B; etc.
  – Keep the accounts sorted by owner name
  – Hash accounts according to owner name
  – And the list goes on…

Observations

- To write correct code, application programmers need to know **how data is organized physically** (e.g., which indexes exist)
  - 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

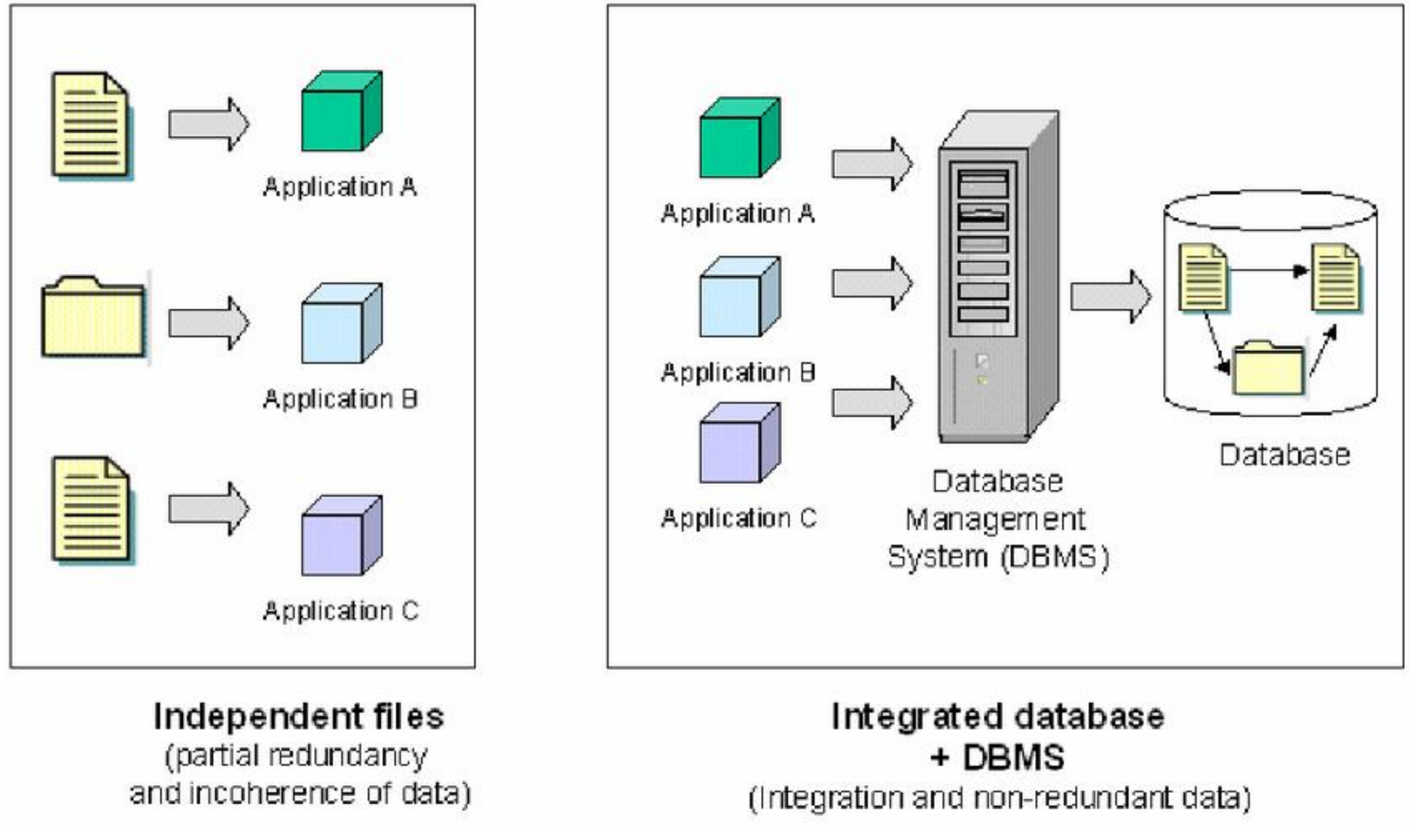
- Data redundancy and inconsistency
  - Multiple file formats, duplication of information in different files
- Difficulty in accessing data
  - Need to write a new program to carry out each new task
- Data isolation: multiple files and formats => difficulty to make joins
- Integrity problems
  - Integrity 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?



- 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

DBMS two important questions

- What’s the right interface to be used by the programmer?
  - **Data model**: Used to specify how data are conceptually structured
  - **Query language**: Used to specify data processing/management tasks

- How DBMS support this interface efficiently?
  - **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)

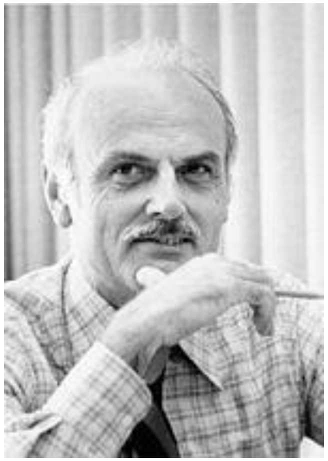
- **A simple data model**: data is stored in relations (tables)
- **A declarative query language**: SQL

```
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



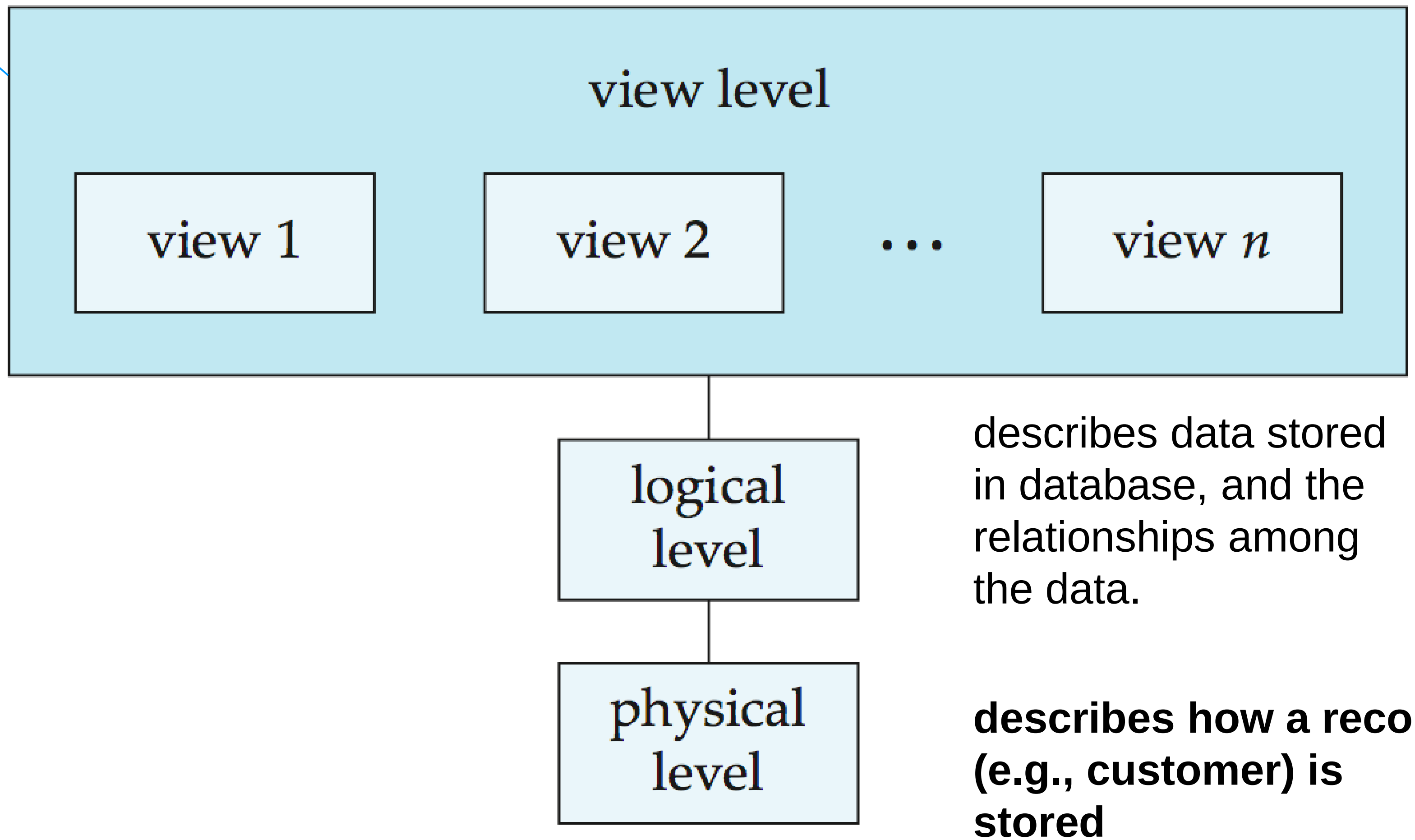
"Future users of large data banks must be protected from **having to know how the data is organized** in the machine (the internal representation)."
- Edgar Frank Codd, 1970

Physical data independence

- Applications should not need to worry about how data is physically structured and stored
- 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 DBMS
- This principle if the most important reason behind the success of DBMS today
  - Edgar Frank **Codd** got a Turing Award for this

DB Levels of Abstraction

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



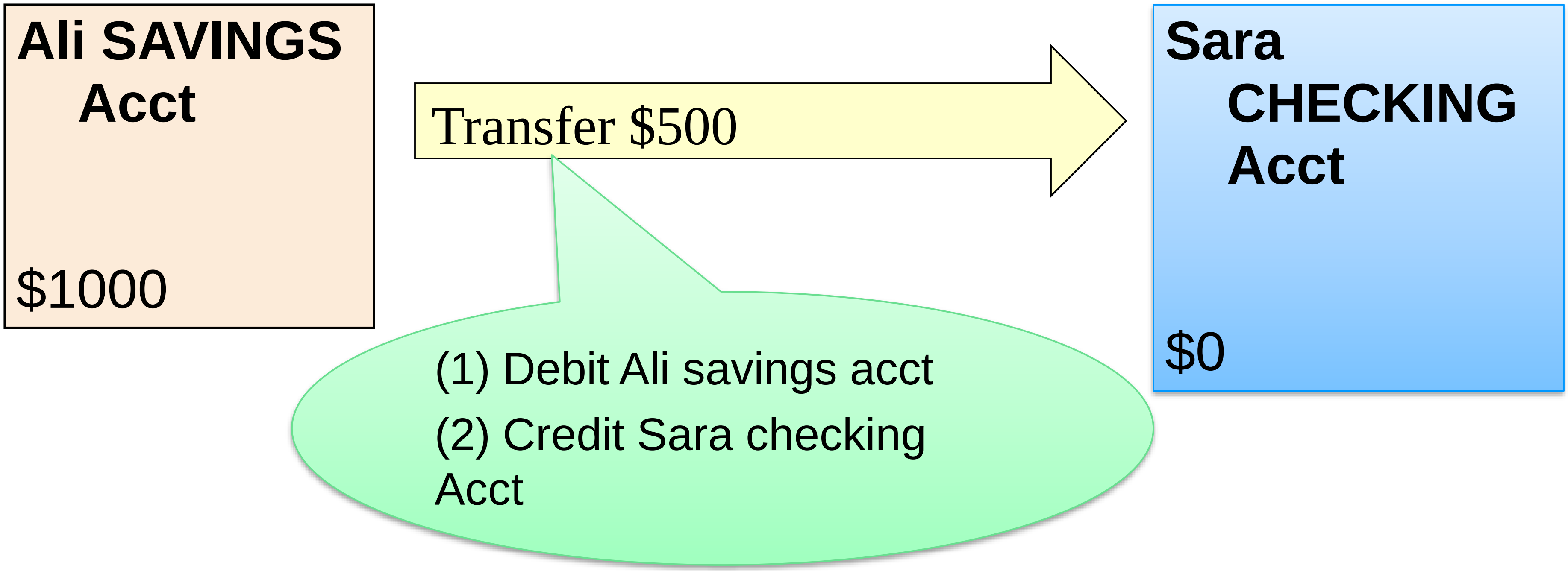
describes data stored in database, and the relationships among the data.

describes how a record (e.g., customer) is stored

ACID = Gold Standard For RDBMS Design

< Back

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



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

Every DB action takes place inside a transaction

RDMS ACID properties

- **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**: Results from completed transactions survive 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:

read balance; $400

Sara withdraws $50:

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!

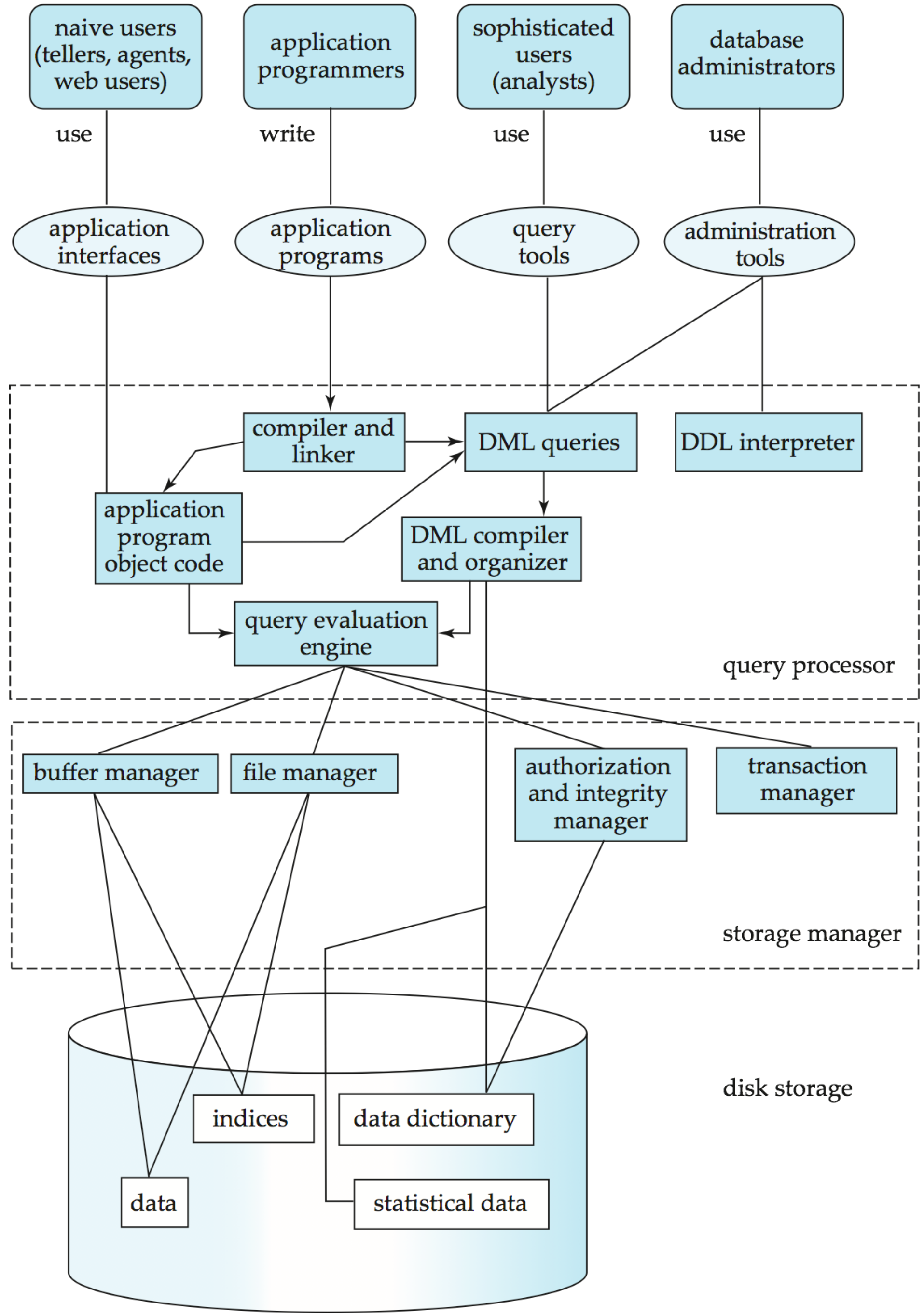
Summary of modern DBMS features

- Persistent storage of data
- Logical data model + declarative queries and updates **=> physical data independence**
  - Provides a declarative interface to data management **=> Hides complexity and increases flexibility**
- Multi-user concurrent access
- Safety from system failures
- Supports high performance:
  - Massive amounts of data (terabytes ~ petabytes)
  - High throughput (thousands ~ millions transactions per minute)
  - High availability (~99.999% uptime)

DBMS Architecture

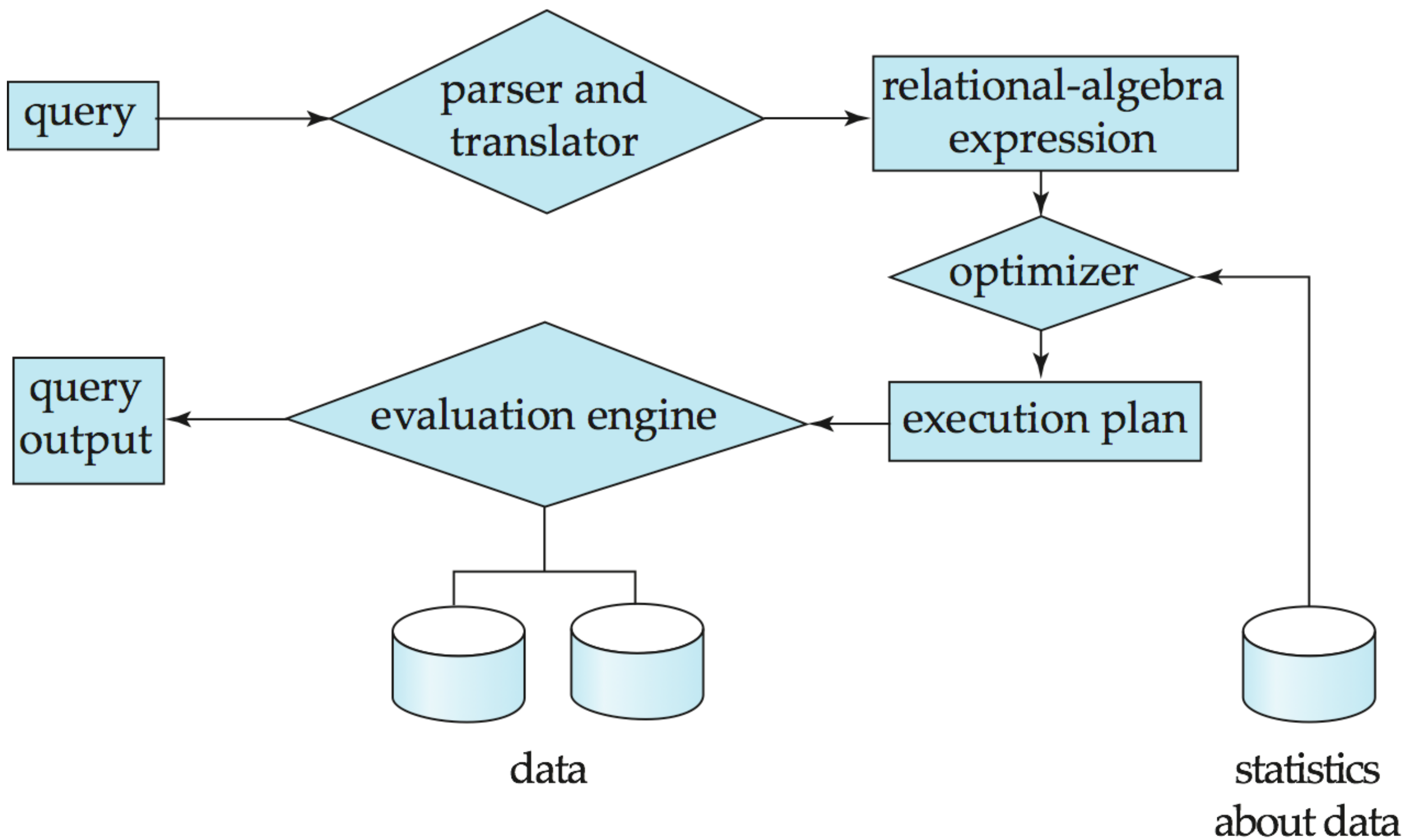
< Back

DBMS Architecture



Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



Query Processing (Cont.)

- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation
- Cost difference between a good and a bad way of evaluating a query can be enormous
- Need to estimate the cost of operations
  - Depends critically on statistical information about relations which the database must maintain
  - Need to estimate statistics for intermediate results to compute cost of complex expressions

Storage Management

- The storage manager is responsible of the following tasks:
  - Interaction with the file manager
  - **Efficient storing, retrieving and updating of data**
- Issues addressed:
  - Storage access
  - File organization
  - Indexing and hashing

Transaction Management

- **Transaction-management component** ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.
  - A **transaction** is a collection of operations that performs a single logical function in a database application

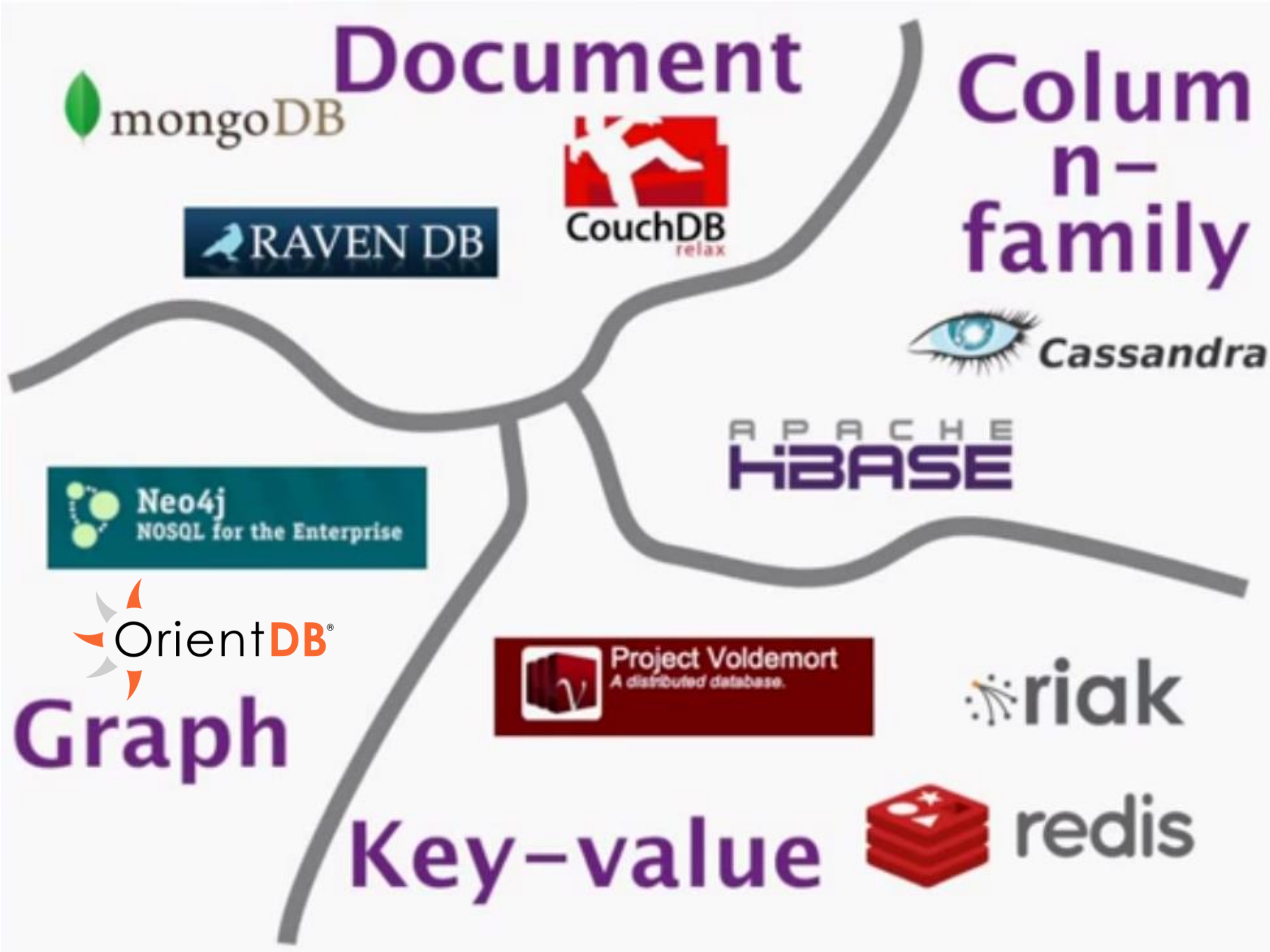
- **Concurrency-control manager** controls the interaction among the concurrent transactions, to ensure the consistency of the database (when multiple users concurrently update the same data)

Summary of Relational Database Features

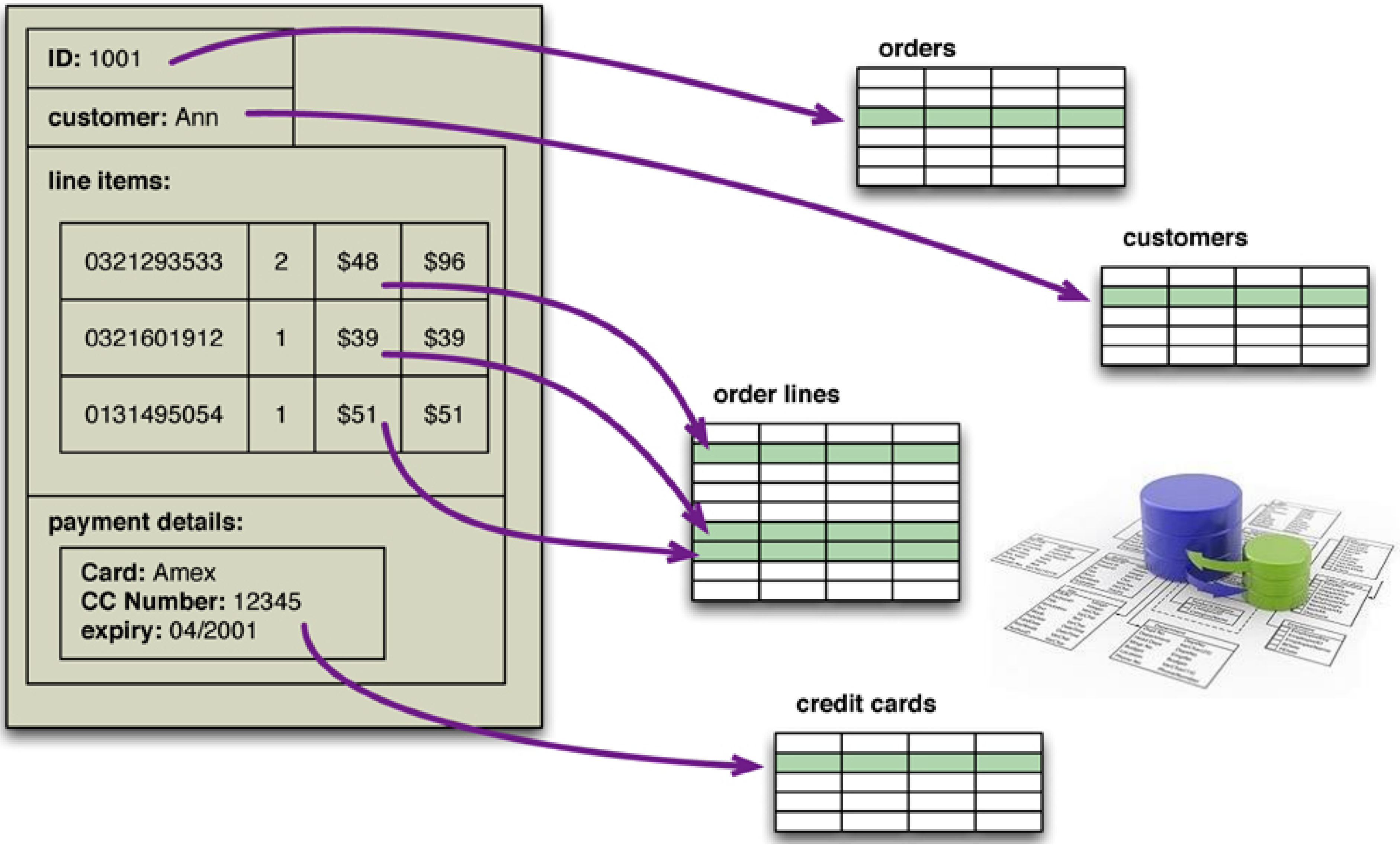
- Storage and access of data that is:
  - **Persistently** stored
  - **Concurrently** accessed
  - **Consistently** modified
  - **Structured** (tabular)
  - **Fast** to access
- Compare: files on disk
  - No concurrency/transactional capabilities (typically)
  - Not as fast (i.e., doesn't scale well)
  - Not structured formally

NoSQL and NewSQL Databases

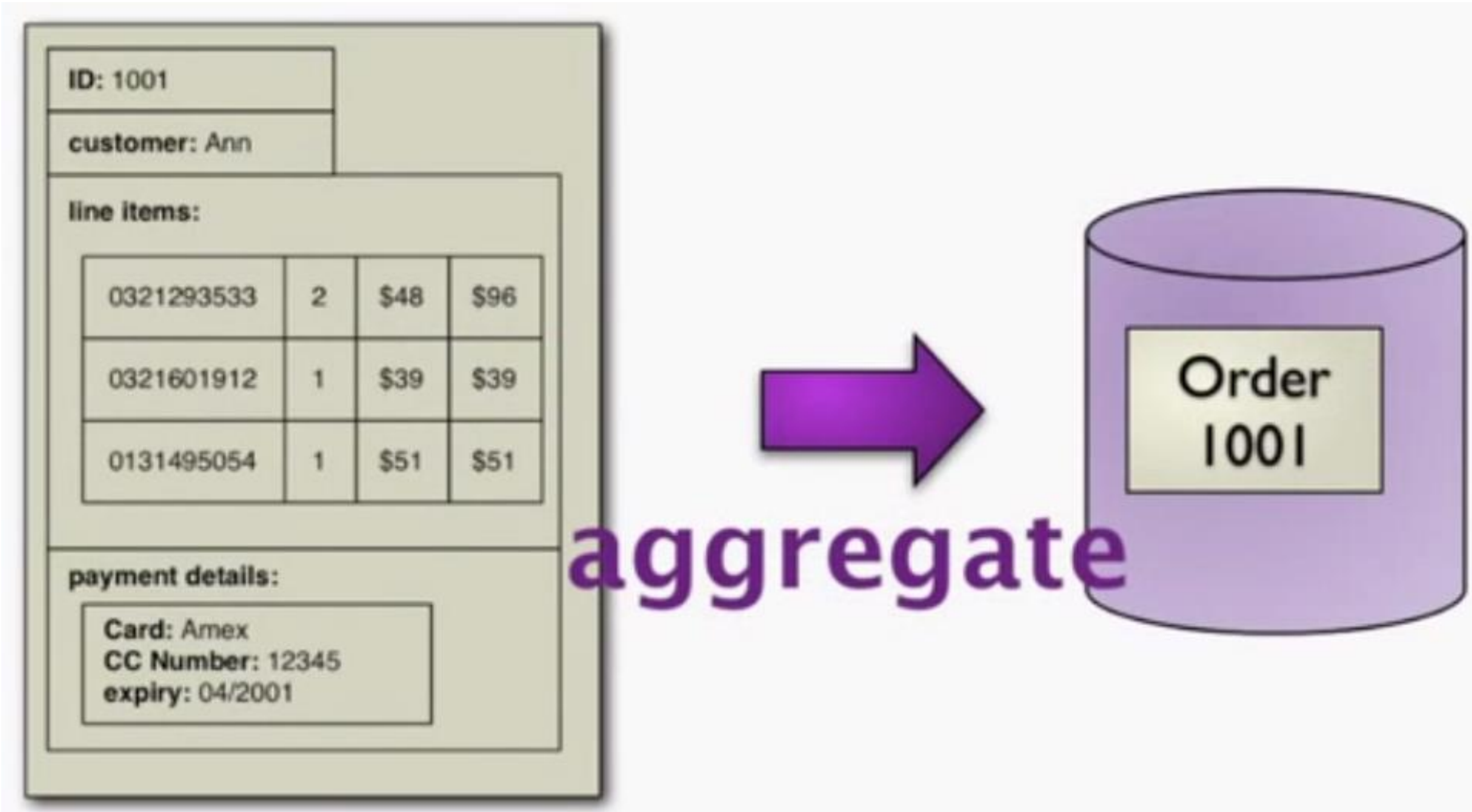
< Back


Document
mongoDB
RAVEN DB
CouchDB
relax
Colum
n-
family
Cassandra
APACHE
HBASE
Neo4j
NOSQL for the Enterprise
OrientDB
Graph
Project Voldemort
A distributed database.
riak
Key-value
redis

Relational vs. Aggregate-oriented - an Example

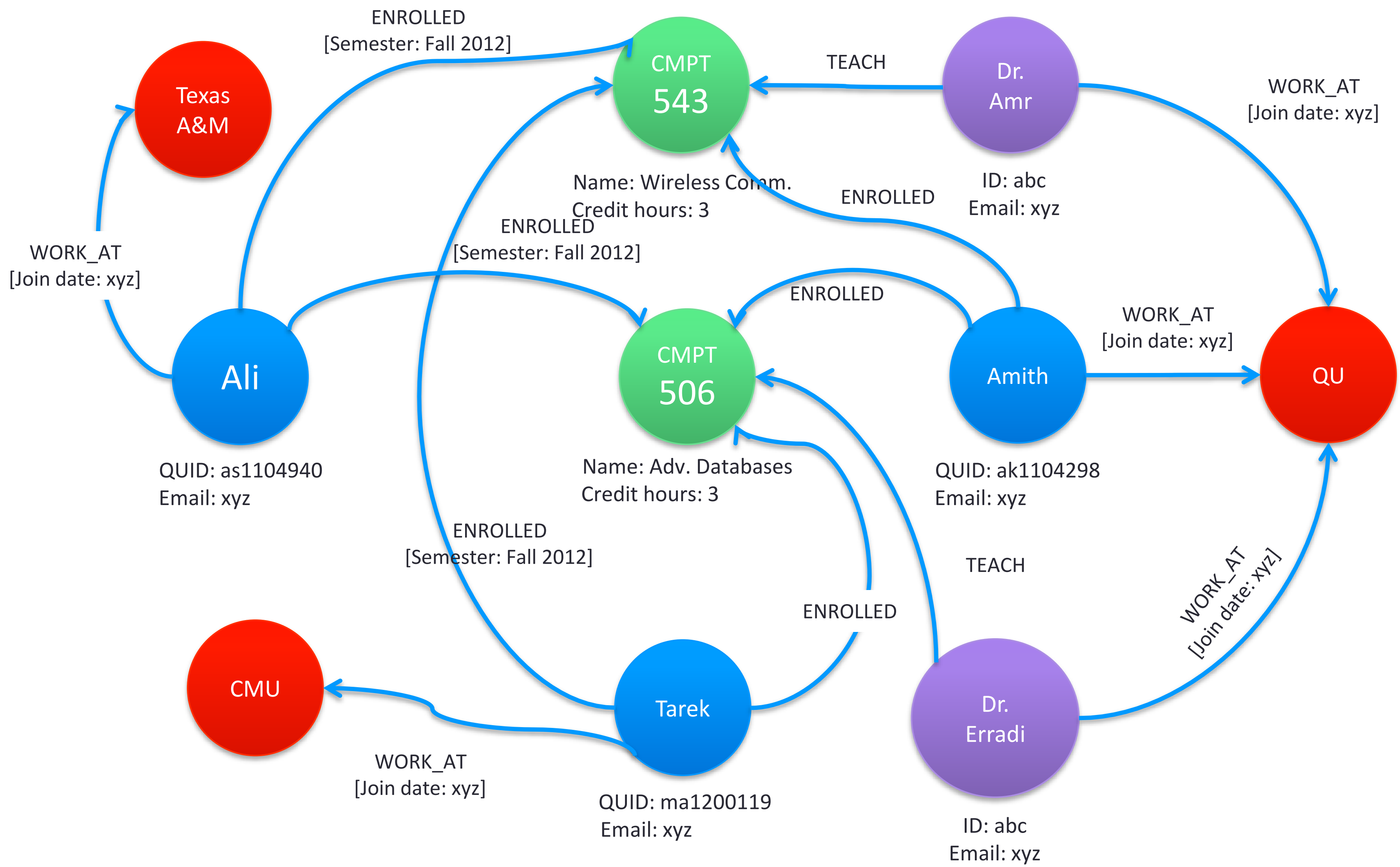


Aggregate Example



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

Example Graph Model



NoSQL Taxonomy

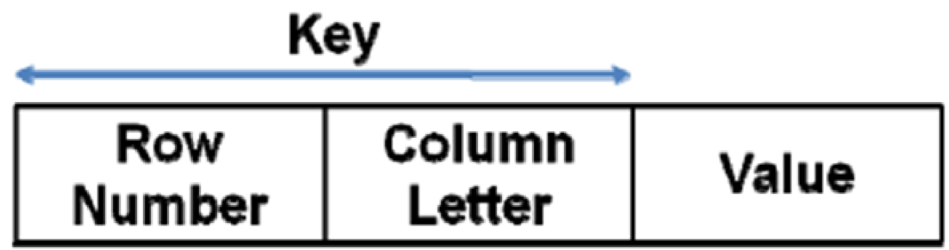
Conceptual Structures:

Key Value Stores

Schema-less system

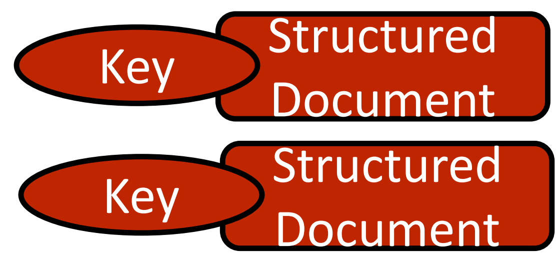
Column Family databases

key is mapped to a value that is a set of columns



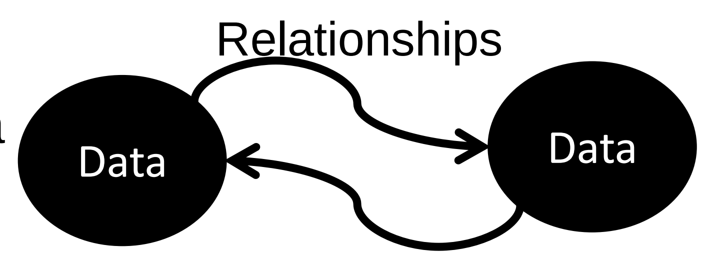
Document Oriented Database

Stores documents that are semi-structured



Graph Databases

Uses nodes and edges to represent data



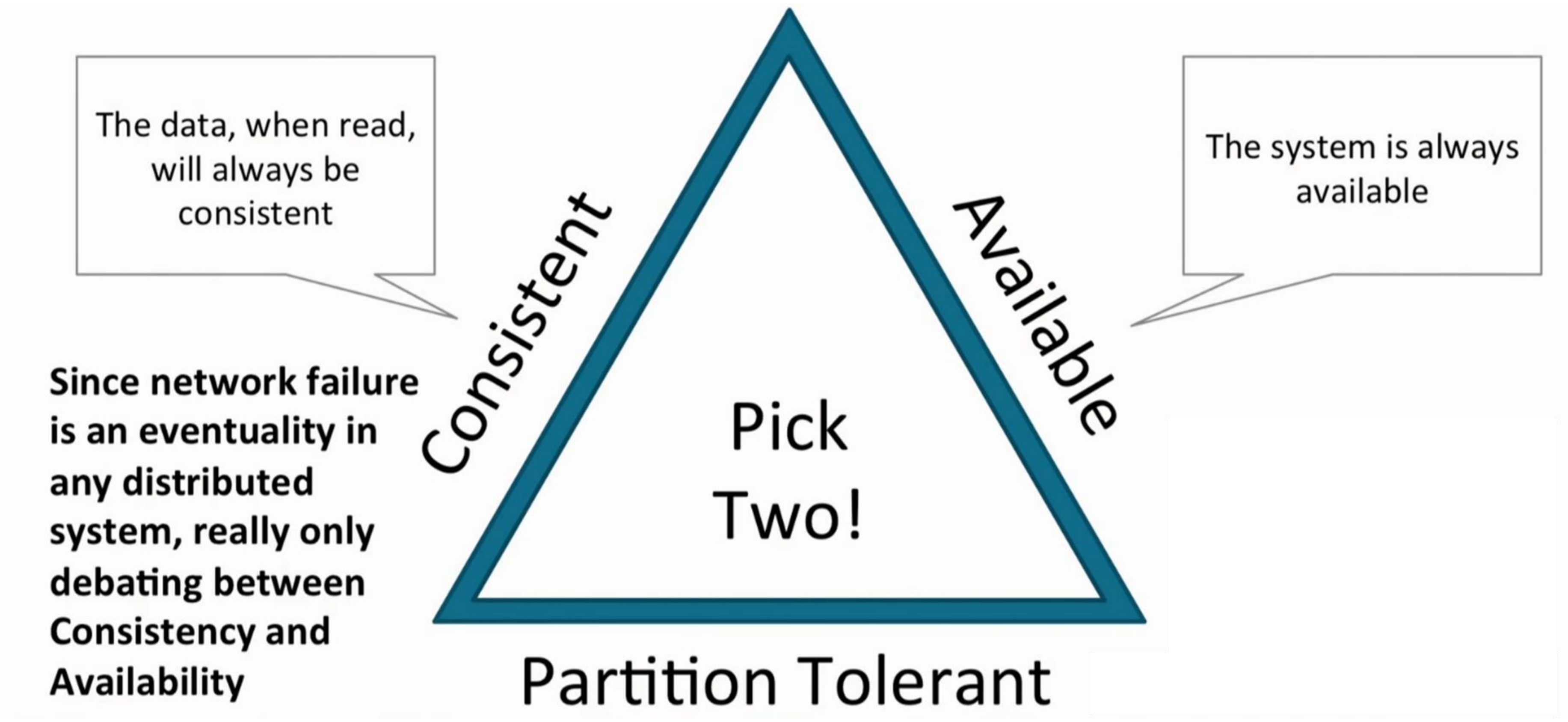
Important Design Goals

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

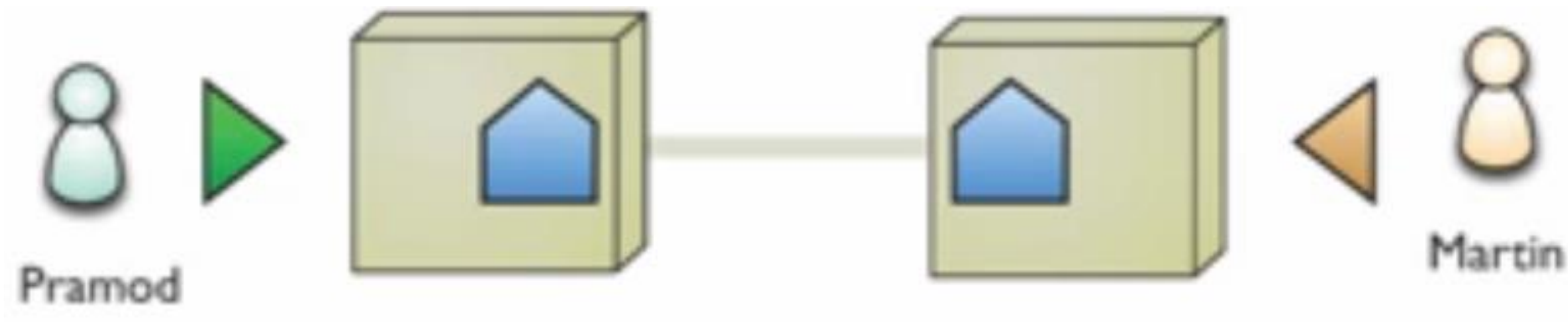
- **High availability** (as downtime implies lost revenue)
  - Replication (with peer to peer replication)
  - Geographic replication
  - Automated failure recovery

CAP-Theorem

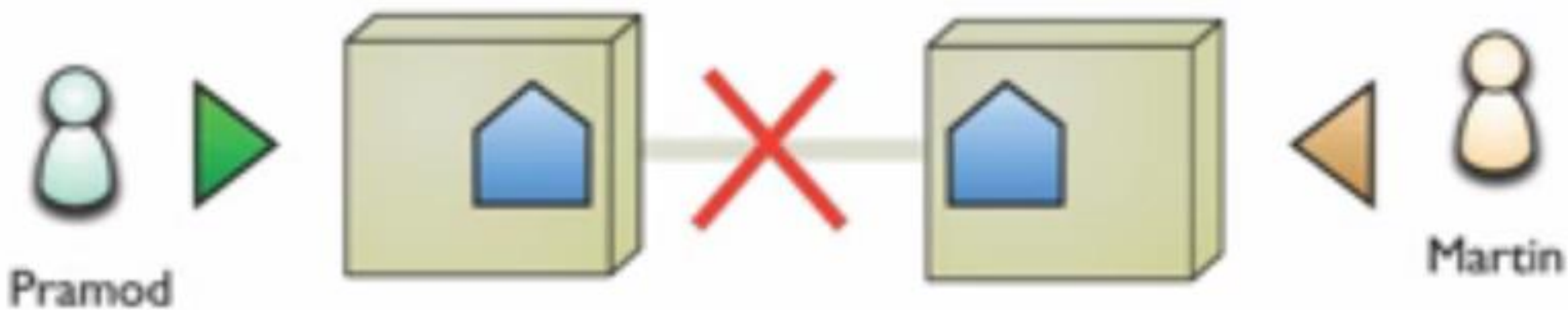
- When data is partitioned, in case of network/node failure, we can only maintain **consistency** or **availability** but NOT both at the same time



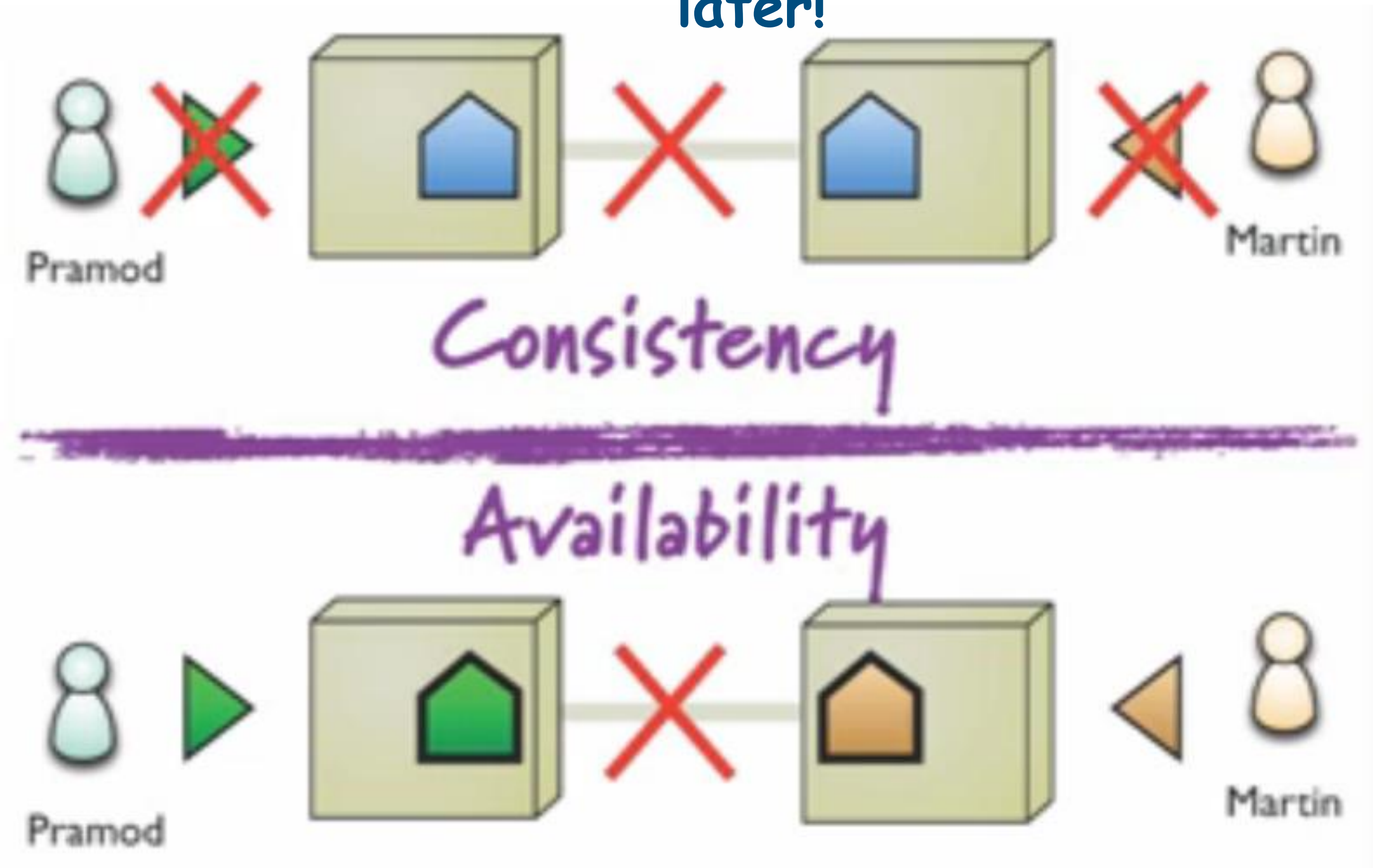
Booking Last Room Scenario by two customers interacting with replicated data in 2 different servers



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!'



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 same scalable performance of NoSQL systems while still maintaining the ACID guarantees of a traditional single-node database system
- When should you use NewSQL?
  - When the application needs to handle very large datasets or a very large number of transactions
  - When ACID guarantees are required
  - 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 – NoOneDB

- Database systems differ in their **data model**, **querying** and **approaches to scale**
- Relational
  - Optimal for single machine
  - Strong ACID guarantees

- Distributed databases:
  - scale-out using automated partitioning and replication
  - distributed across a cluster of machines
  - NoSQL
    - move away from ACID properties
    - come in several data models and query languages

  - NewSQL
    - maintain ACID properties
    - Uses the relational model and SQL