

# Database Management Systems

## Chapter 1

**ECS 165A – Winter 2025**



**Mohammad Sadoghi**

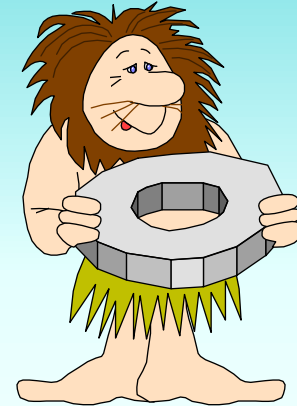
*Exploratory Systems Lab*

*Department of Computer Science*

**UC DAVIS**  
UNIVERSITY OF CALIFORNIA



# What Is a DBMS?



- ❖ A very large, integrated collection of data.
- ❖ Models real-world enterprise.
  - Entities (e.g., students, courses)
  - Relationships (e.g., Madonna is taking ECS165)
- ❖ A Database Management System (DBMS) is a software package designed to store and manage databases.

# *Files vs. DBMS*

- ❖ Application must stage large datasets between main memory and secondary storage (e.g., buffering, page-oriented access, 64-bit addressing, etc.)
- ❖ Special code for different queries
- ❖ Must protect data from inconsistency due to multiple concurrent users
- ❖ Crash recovery
- ❖ Security and access control

# *Why Use a DBMS?*



- ❖ Data independence and efficient access.
- ❖ Reduced application development time.
- ❖ Data integrity and security.
- ❖ Uniform data administration.
- ❖ Concurrent access, recovery from crashes.

# Why Study Databases??



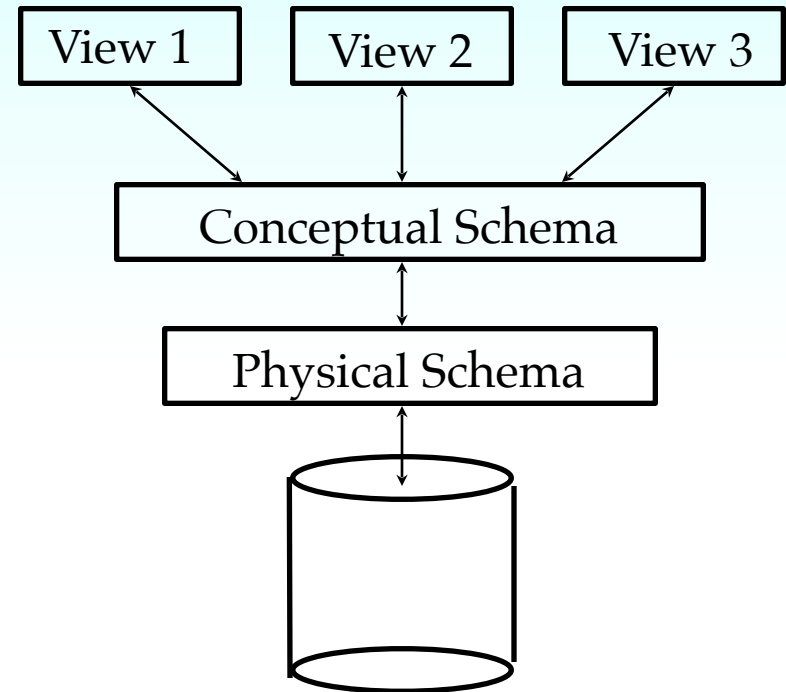
- ❖ Shift from computation to information
- ❖ Datasets increasing in diversity and volume.
  - Digital libraries, interactive video, Human Genome project, EOS project
  - ... need for DBMS exploding
- ❖ DBMS encompasses most of CS
  - OS, languages, theory, AI, multimedia, logic

# Data Models

- ❖ A data model is a collection of concepts for describing data.
- ❖ A schema is a description of a particular collection of data, using the a given data model.
- ❖ The relational model of data is the most widely used model today.
  - Main concept: relation, basically a table with rows and columns.
  - Every relation has a schema, which describes the columns, or fields.

# Levels of Abstraction

- ❖ Many views, single conceptual (logical) schema and physical schema.
  - Physical schema describes the files and indexes used.
  - Conceptual schema defines logical structure
  - Views describe how users see the data.



# Example: University Database

## ❖ Conceptual schema:

- *Students*(*sid: string, name: string, login: string, age: integer, gpa:real*)
- *Courses*(*cid: string, cname:string, credits:integer*)
- *Enrolled*(*sid:string, cid:string, grade:string*)

## ❖ Physical schema:

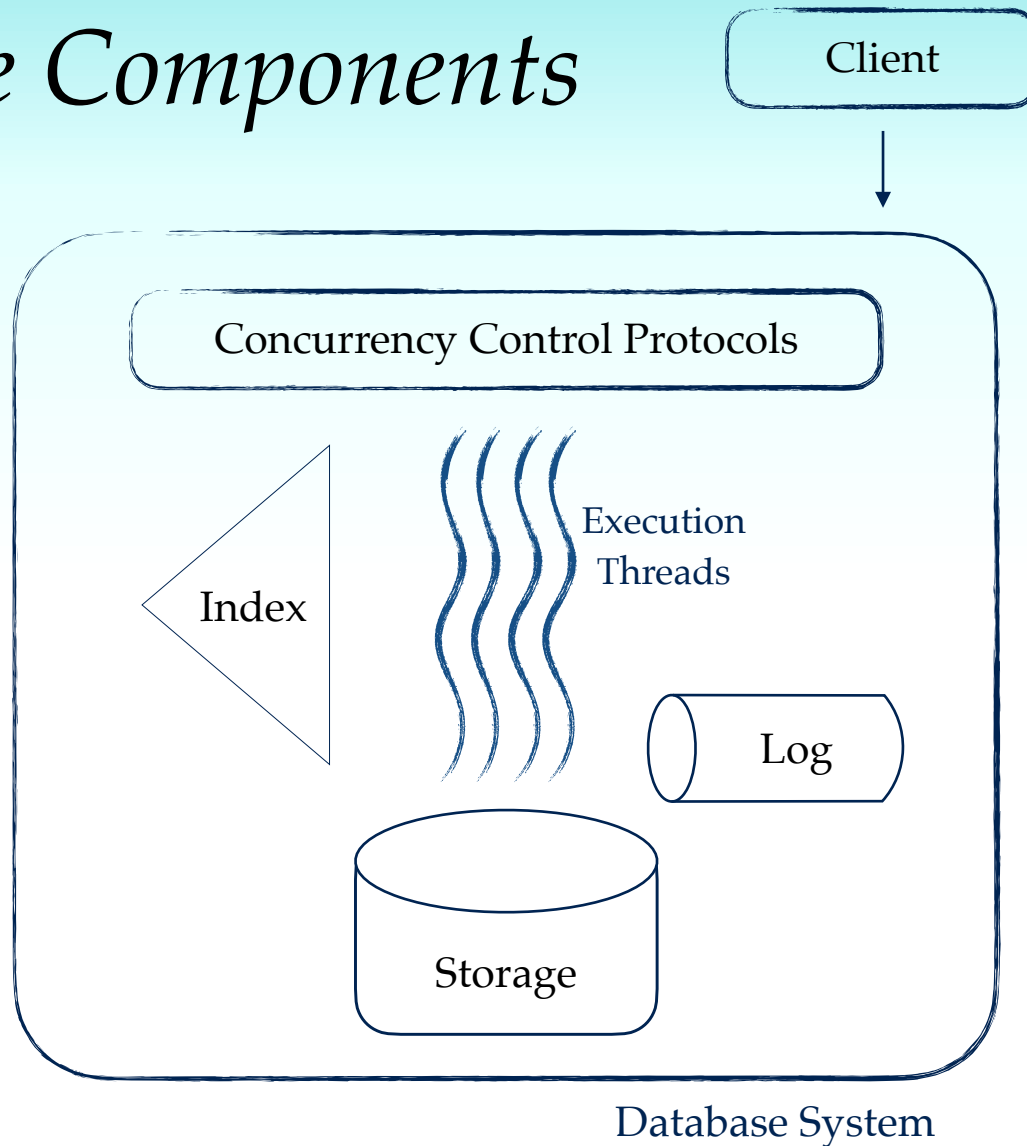
- Relations stored as unordered files.
- Index on first column of Students.

## ❖ External Schema (View):

- *Course\_info*(*cid:string,enrollment:integer*)

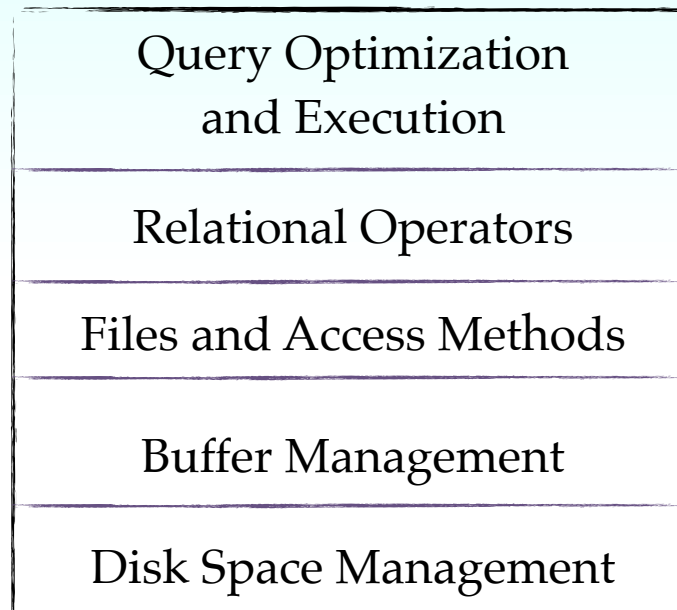


# Database Components



# Structure of a DBMS

These layers must consider concurrency control and recovery



# *Data Independence*

- ❖ Applications insulated from how data is structured and stored.
- ❖ *Logical data independence*: Protection from changes in *logical* structure of data.
- ❖ *Physical data independence*: Protection from changes in *physical* structure of data.

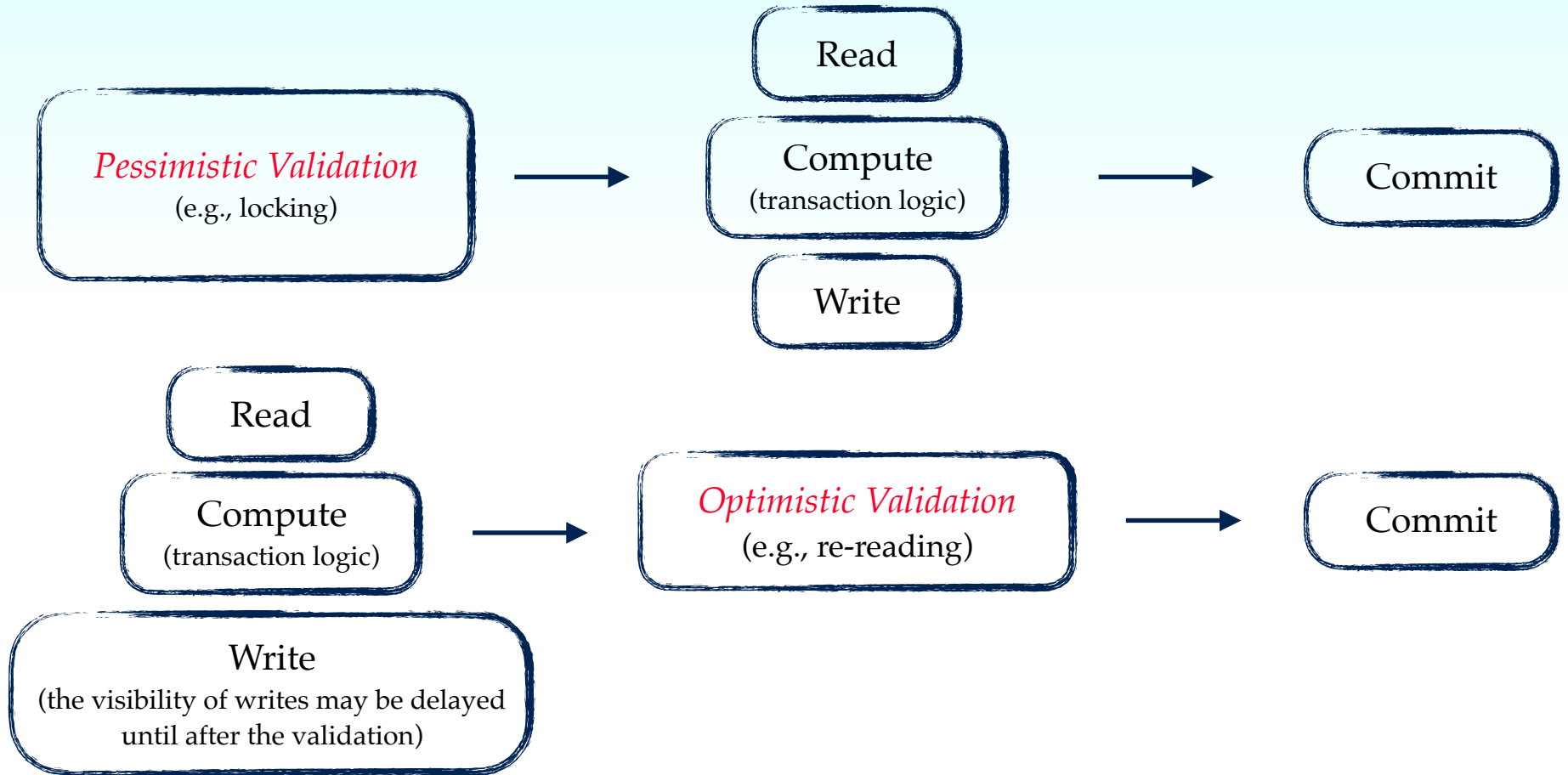
# Concurrency Control

- ❖ Concurrent execution of user programs is essential for good DBMS performance.
  - Because disk (or even memory?) accesses are frequent, and **relatively** slow, it is important to keep the cpu humming by working on several user programs concurrently.
- ❖ Interleaving actions of different user programs can lead to inconsistency: e.g., check is cleared while account balance is being computed.
- ❖ DBMS ensures such problems don't arise: users can pretend they are using a single-user system.

# Transaction: An Execution of a DB Program

- ❖ Key concept is transaction, which is an *atomic* sequence of database actions (reads / writes).
- ❖ Each transaction, executed completely, must leave the DB in a consistent state if DB is consistent when the transaction begins.
  - Users can specify some simple integrity constraints on the data, and the DBMS will enforce these constraints.
  - Beyond this, the DBMS does not really understand the semantics of the data. (e.g., it does not understand how the interest on a bank account is computed).
  - Thus, ensuring that a transaction (run alone) preserves consistency is ultimately the **user's** responsibility!

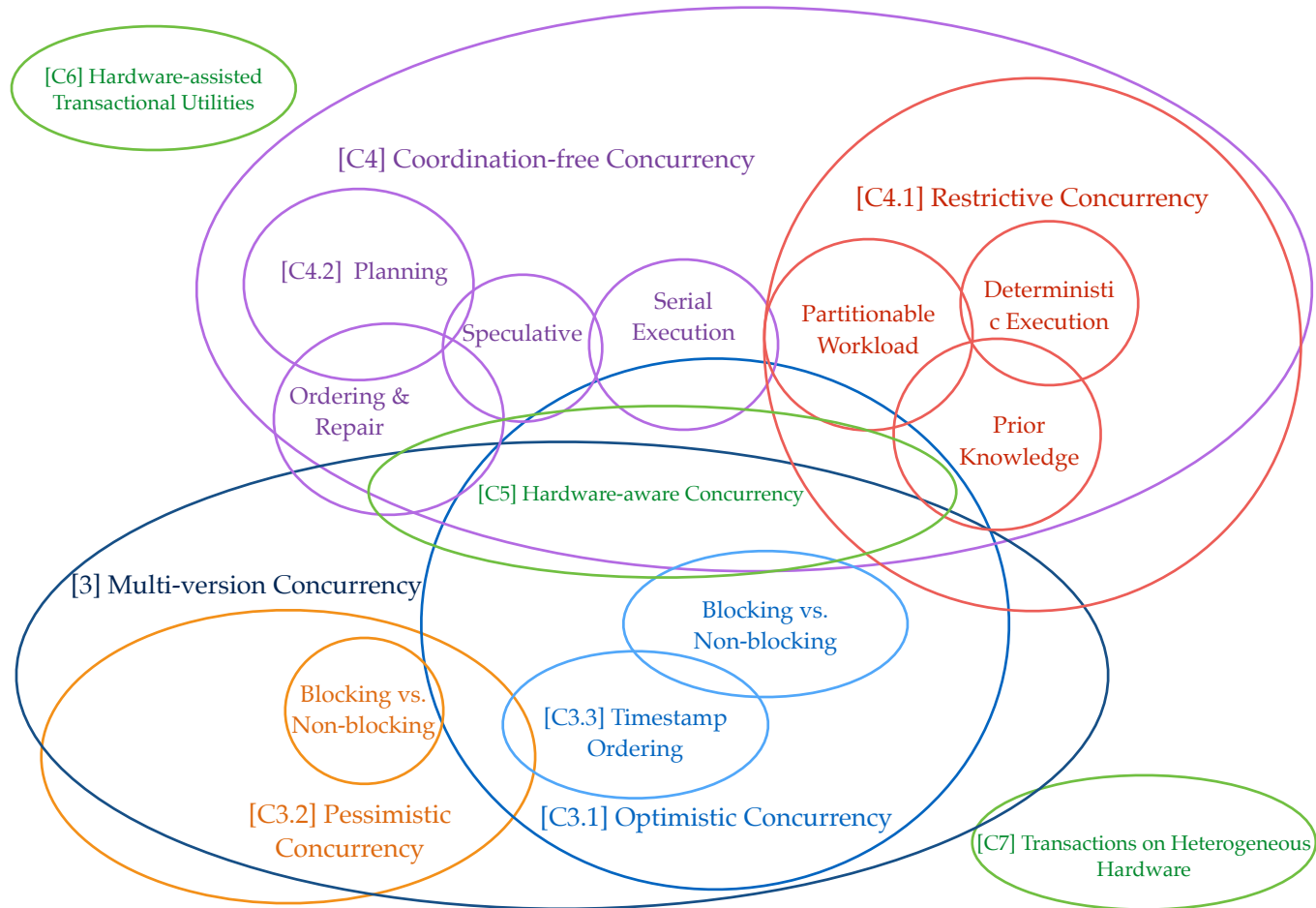
# Pessimistic vs. Optimistic CC Schedule



# Scheduling Concurrent Transactions

- ❖ DBMS ensures that execution of  $\{T_1, \dots, T_n\}$  is equivalent to some serial execution  $T_1' \dots T_n'$ .
  - Before reading/writing an object, a transaction requests a lock on the object, and waits till the DBMS gives it the lock. All locks are released at the end of the transaction. (e.g., Strict 2PL locking protocol, a *pessimistic protocol*.)
  - **Idea:** Suppose an action of  $T_i$  (say, writing  $X$ ) affects  $T_j$  (which perhaps reads  $X$ ). Let's say  $T_i$  will obtain the lock on  $X$  first and  $T_j$  is forced to wait until  $T_i$  completes; this effectively orders the transactions.
  - What if  $T_j$  already has a lock on  $Y$  and  $T_i$  later requests a lock on  $Y$ ? (Deadlock!)  $T_i$  or  $T_j$  is aborted and restarted!

# Pessimistic vs. Optimistic CC Schedule

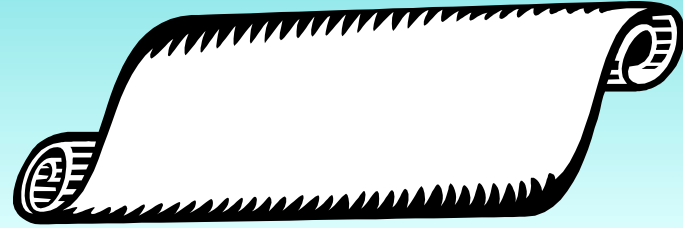




# Ensuring Atomicity

- ❖ DBMS ensures *atomicity* (all-or-nothing property) even if system crashes in the middle of a Xact.
- ❖ **Idea:** Keep a log (history) of all actions carried out by the DBMS while executing a set of Xacts:
  - **Before** a change is made to the database, the corresponding log entry is forced to a safe location. (WAL protocol; OS support for this is often inadequate.)
  - After a crash, the effects of partially executed transactions are undone using the log. (Thanks to WAL, if log entry wasn't saved before the crash, corresponding change was not applied to database!)

# The Log



- ❖ The following actions are recorded in the log:
  - *Ti writes an object*: The old value and the new value.
    - Log record must go to disk *before* the changed page!
  - *Ti commits/aborts*: A log record indicating this action.
- ❖ Log records chained together by Xact id, so it's easy to undo a specific Xact (e.g., to resolve a deadlock).
- ❖ Log is often *duplexed* and *archived* on “stable” storage.
- ❖ All log related activities (and in fact, all CC related activities such as lock/unlock, dealing with deadlocks etc.) are handled transparently by the DBMS.

# *Databases make these folks happy ...*

- ❖ End users and DBMS vendors
- ❖ DB application programmers
  - E.g., Instagram App
- ❖ *Database administrator (DBA)*
  - Designs logical / physical schemas
  - Handles security and authorization
  - Data availability, crash recovery
  - Database tuning as needs evolve



*Must understand how a DBMS works!*

# Summary

- ❖ DBMS used to maintain, query large datasets.
- ❖ Benefits include recovery from system crashes, concurrent access, quick application development, data integrity and security.
- ❖ Levels of abstraction give data independence.
- ❖ A DBMS typically has a layered architecture.
- ❖ DBAs & Data Scientists are **well-paid!** 😊
- ❖ DBMS R&D is one of the broadest, most exciting areas in CS.

