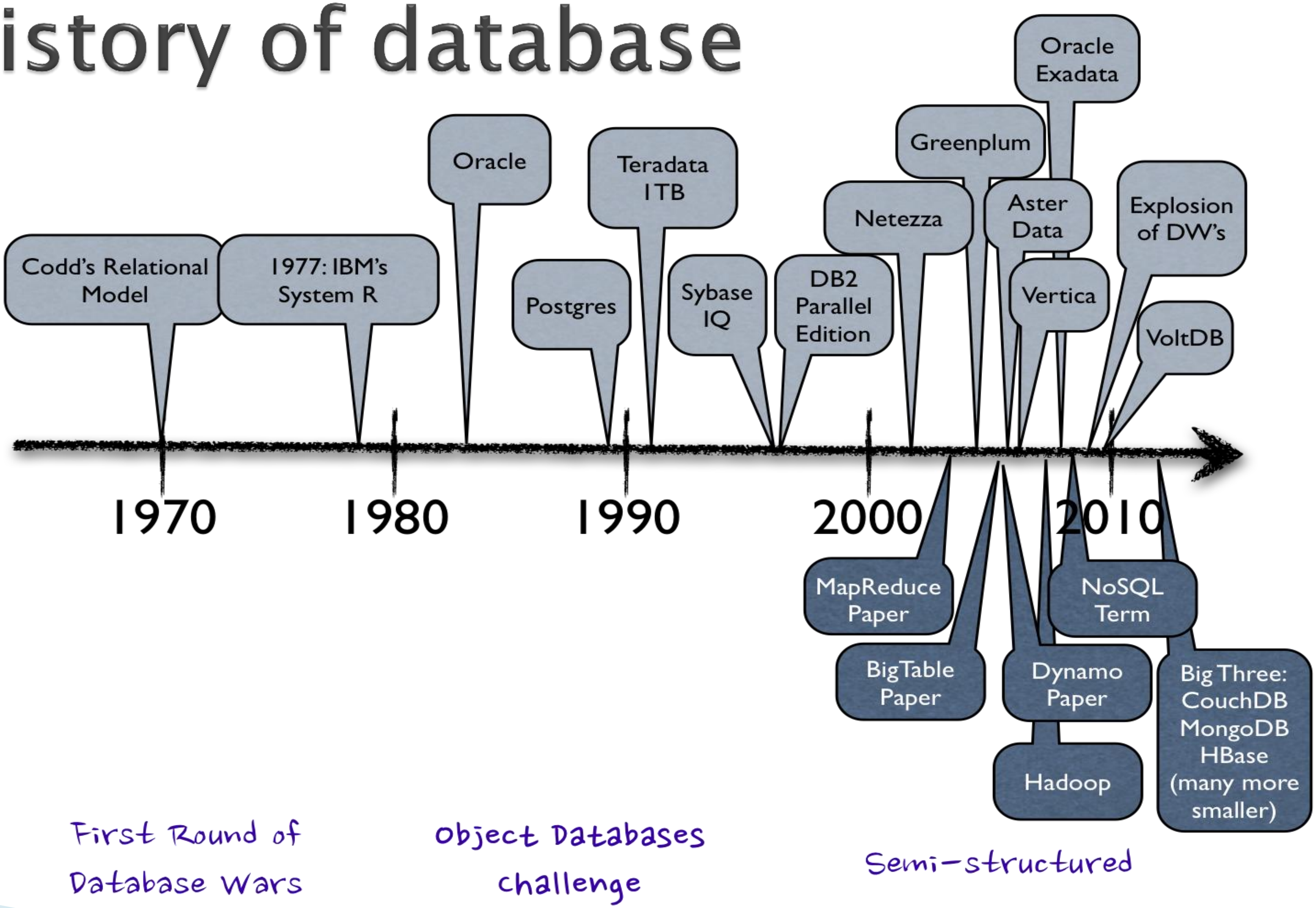
Lecture 1

Database Management Systems

Software generation

- **Machine Language** →0 and 1 binary digits
- **Assembly Language**→ Code words or Mnemonics
  - E.g. ADD MOV
- **High Level Language**(Third Generation Language or 3GL)
  - English Like E.g. Pascal ,C
  - Compilers are used to debug the programs.
- **Application Program Generators** or Report Program Generators(Fourth Generation Language or 4GL)
- **Artificial Intelligence**(5GL)
  - Robotics
  - Expert Systems

History of database



Difference between 3GL and 4 GL

3GL	4GL
Professional Programmers	Professional and Non Professional programmers
Requires specification of how to perform the task	Requires specification of what task to perform
Large number of procedural instructions required	Less instructions
Debugging difficult	Debugging easy
File oriented	Database oriented

Terminologies

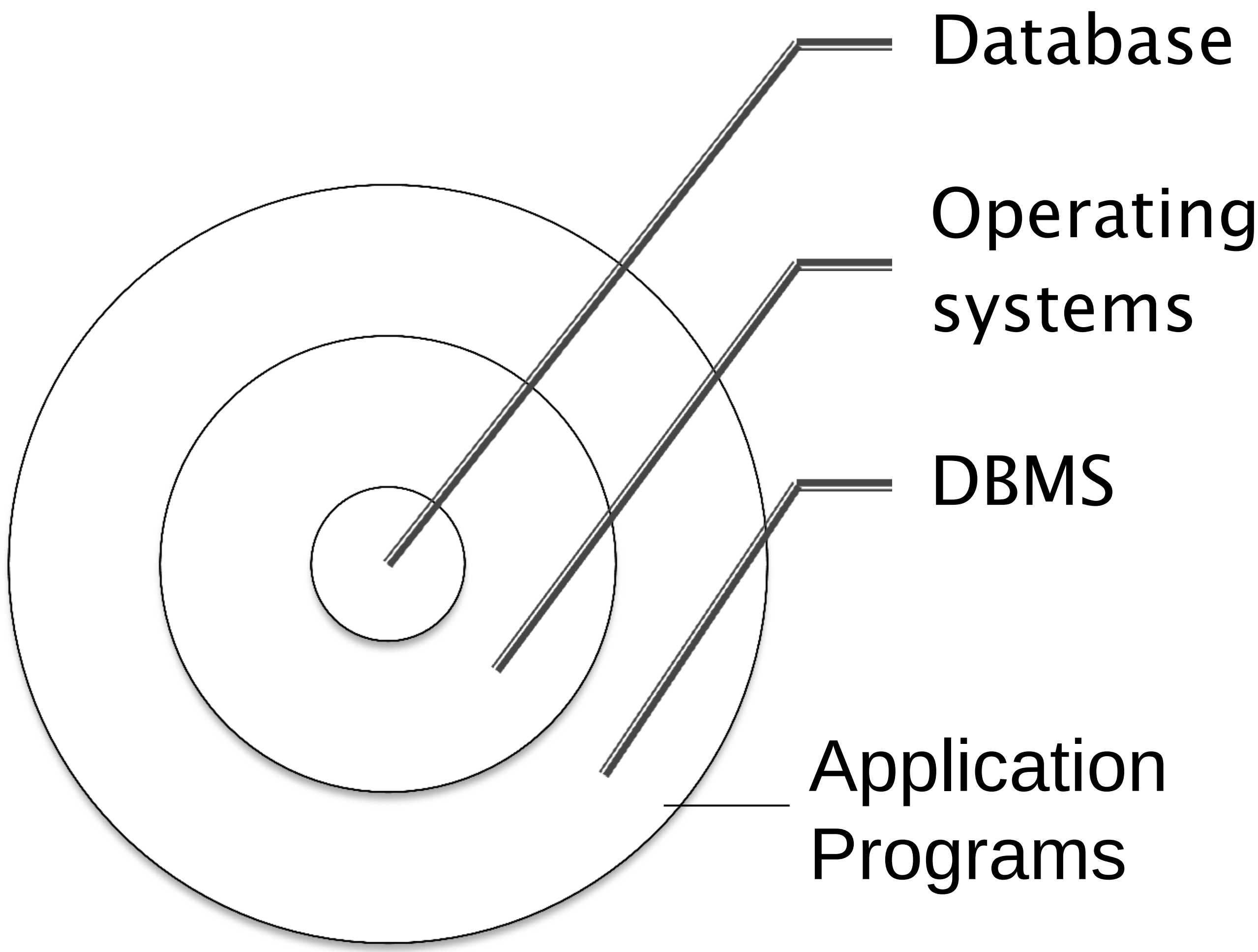
- **Database**

Collection of data of a particular entity

- **Database Management Systems**

Software which provides to tools for managing data stored in the database

- **Data defining data→ Metadata**
- Data


Database
Operating systems
DBMS
Application Programs

Database Management System (DBMS)

- DBMS contains information about a particular enterprise
  - Collection of interrelated data
  - Set of programs to access the data
  - An environment that is both *convenient* and *efficient* to use
- Database Applications:
  - Banking: transactions
  - Airlines: reservations, schedules
  - Universities: registration, grades
  - Sales: customers, products, purchases
  - Online retailers: order tracking, customized recommendations
  - Manufacturing: production, inventory, orders, supply chain
  - Human resources: employee records, salaries, tax deductions
- Databases can be very large.
- Databases touch all aspects of our lives

University Database Example

- **Application program examples**
  - Add new students, instructors, and courses
  - Register students for courses, and generate class rosters
  - Assign grades to students, compute grade point averages (GPA) and generate transcripts
- In the early days, database applications were built directly on top of file systems

Drawbacks of using file systems to store data

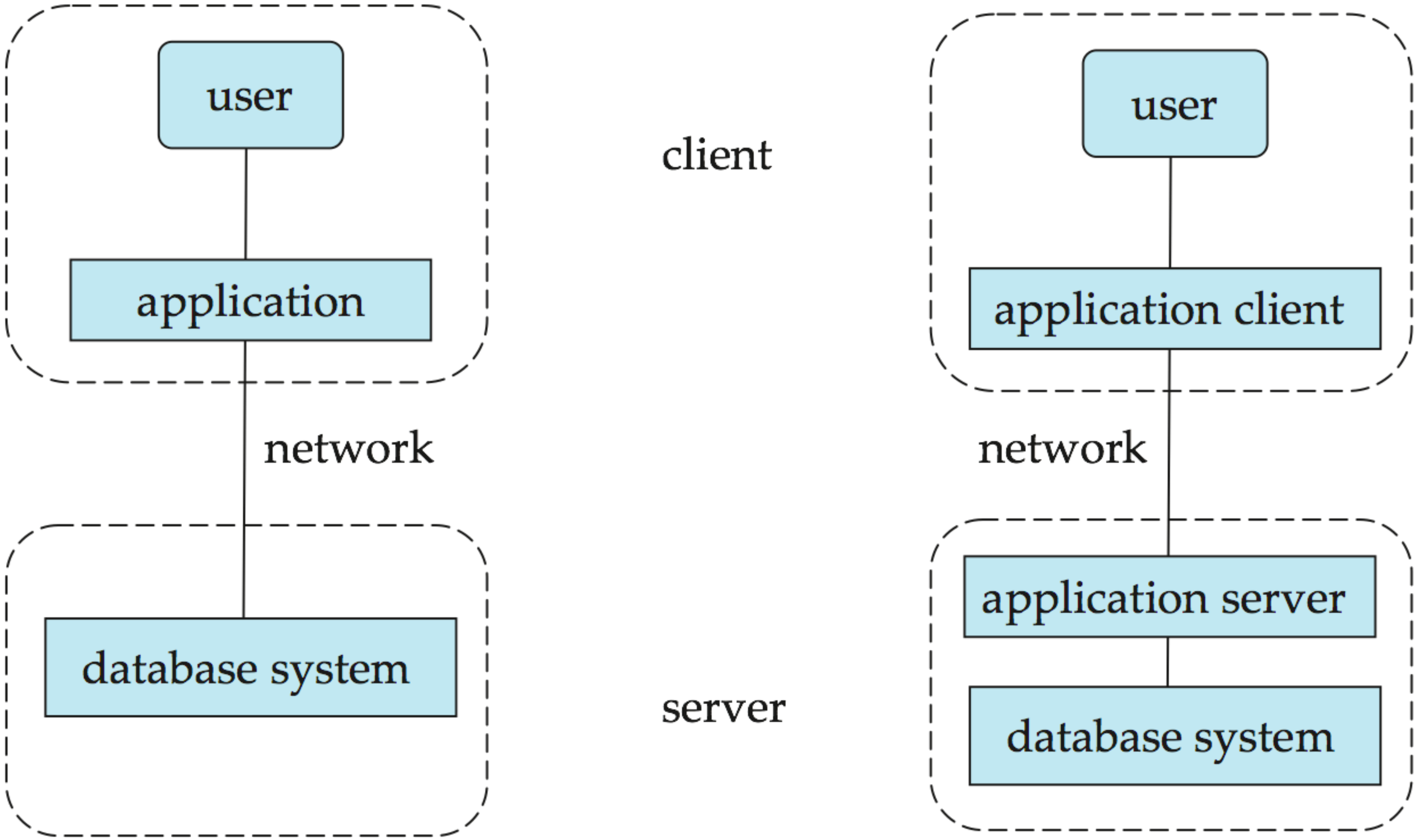
- **Data redundancy and inconsistency**
  - Multiple file formats, duplication of information in different files
- **Difficulty in accessing data**
  - Need to write a new program to carry out each new task
- **Data isolation** — multiple files and formats
- **Integrity problems**
  - **Integrity constraints** (e.g., account balance > 0) become "buried" in program code rather than being stated explicitly
  - Hard to add new constraints or change existing ones

Drawbacks of using file systems to store data (Cont.)

- **Atomicity of updates**
  - Failures may leave database in an inconsistent state with partial updates carried out
  - Example: Transfer of funds from one account to another should either complete or not happen at all
- **Concurrent access by multiple users**
  - Concurrent access needed for performance
  - Uncontrolled concurrent accesses can lead to inconsistencies
    - Example: Two people reading a balance (say 100) and updating it by withdrawing money (say 50 each) at the same time
- **Security problems**
  - Hard to provide user access to some, but not all, data

Database systems offer solutions to all the above problems

Database Architecture



(a) Two-tier architecture

(b) Three-tier architecture

Advantages of two tier architecture

Understanding and maintenances is easier.

Disadvantages:

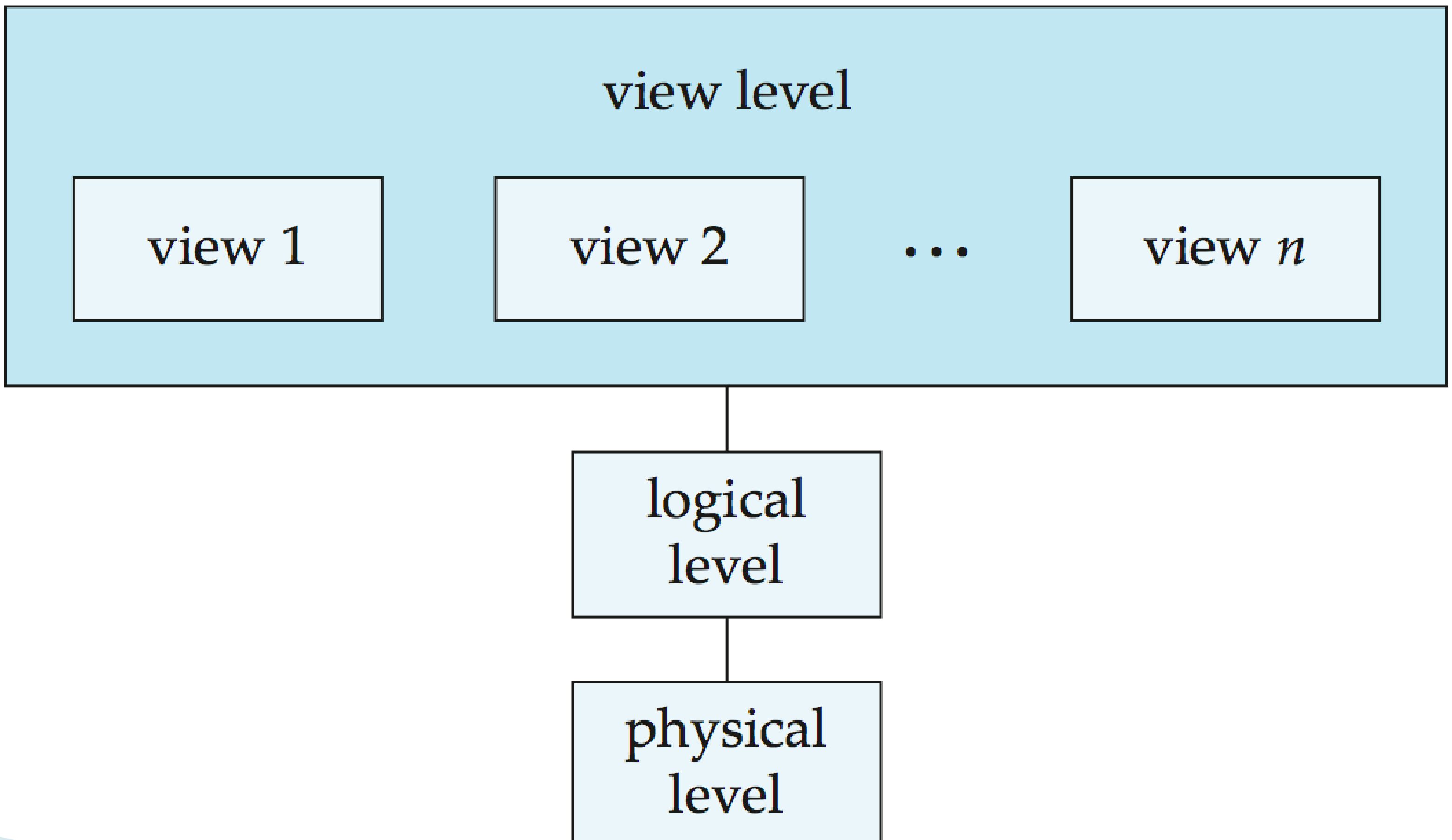
Performance will be reduced when there are more users.

Advantages Three tier Architecture

- Easy to modify with out affecting other modules
- Fast communication
- Performance will be good in three tier architecture.

View of Data

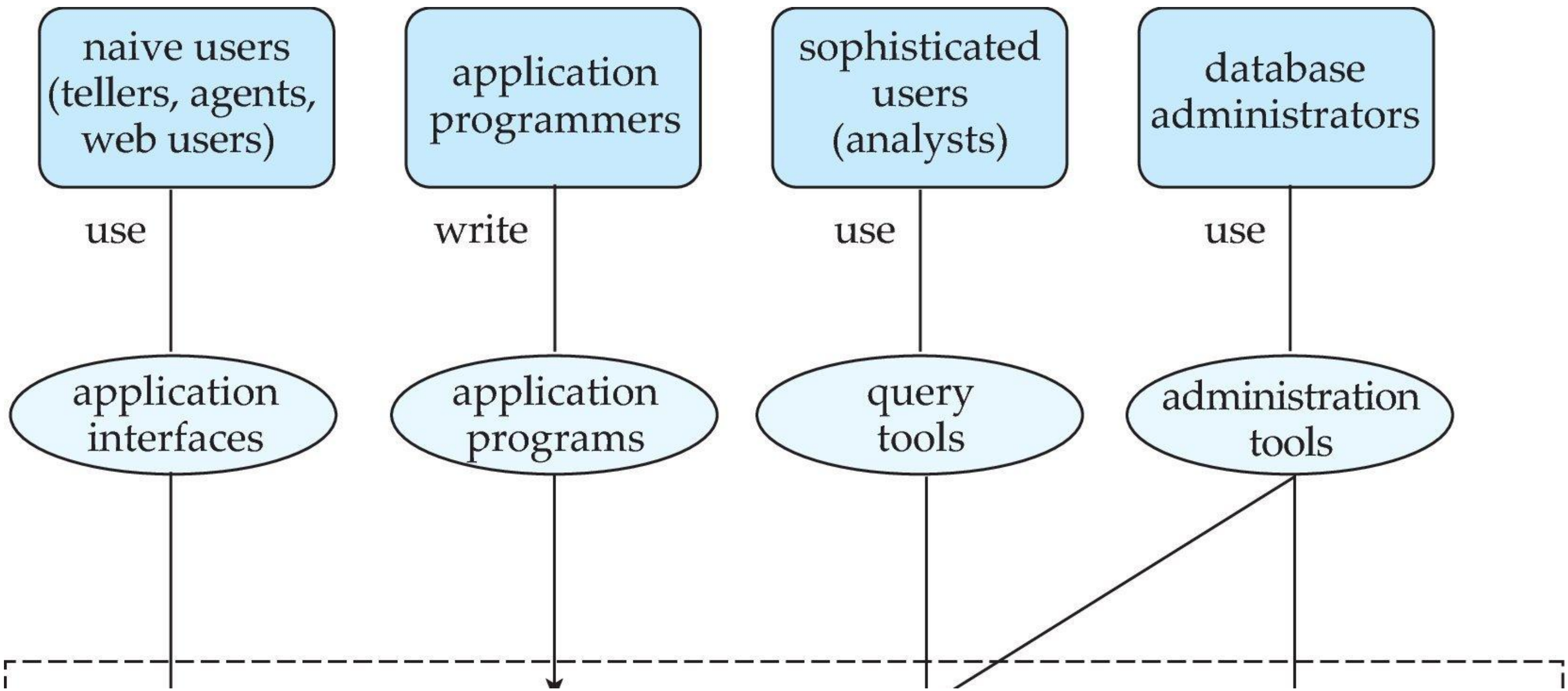
An architecture for a database system



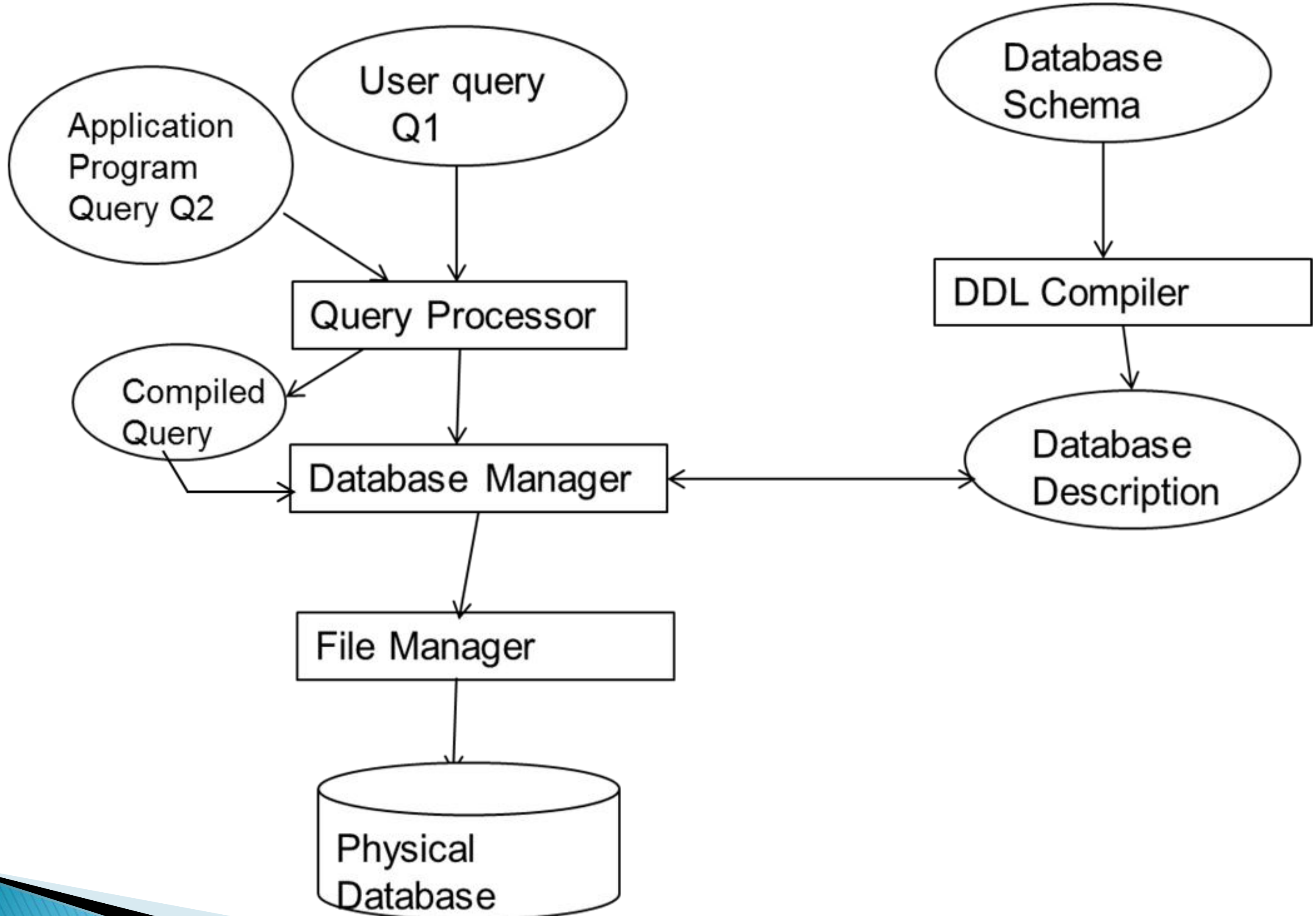
Instances and Schemas

- Similar to types and variables in programming languages
- **Schema** – the logical structure of the database
  - Example: The database consists of information about a set of customers and accounts and the relationship between them
  - Analogous to type information of a variable in a program
  - **Physical schema**: database design at the physical level
  - **Logical schema**: database design at the logical level
- **Instance** – the actual content of the database at a particular point in time
  - Analogous to the value of a variable
- **Physical Data Independence** – the ability to modify the physical schema without changing the logical schema
  - Applications depend on the logical schema
  - In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.
- **Logical Data Independence**
  - is the ability to modify the **logical** schema without causing application program to be rewritten.

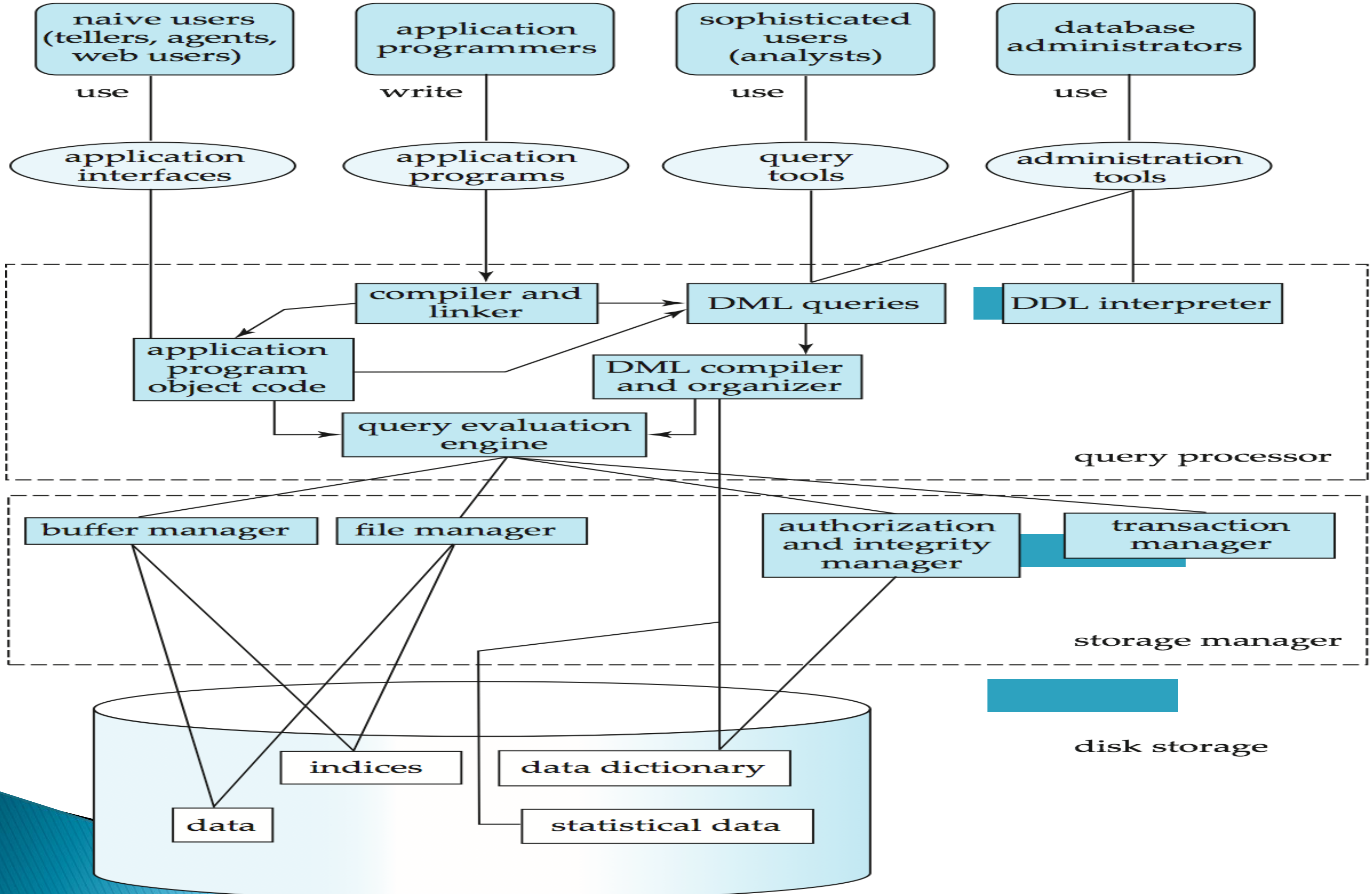
Database Users and Administrators



Database Architecture



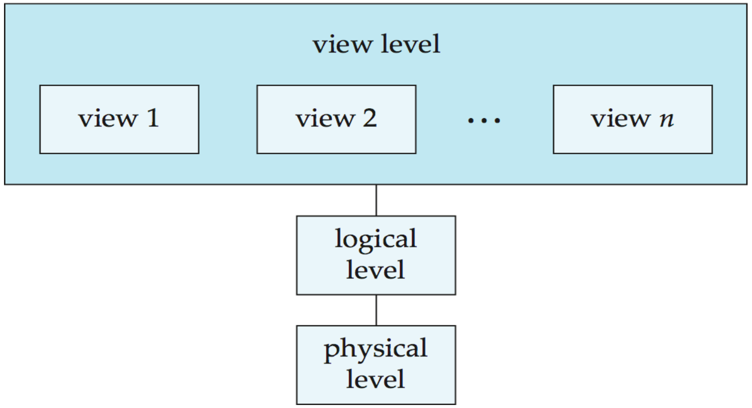
Database System Internals



Data Abstraction Levels of Abstraction

- **Physical level:** describes how a record (e.g., customer) is stored.
- **Logical level:** describes data stored in database, and the relationships among the data.

type *instructor* = **record**
    ID : string;
    name : string;
    dept_name : string;
    salary : integer;
end;



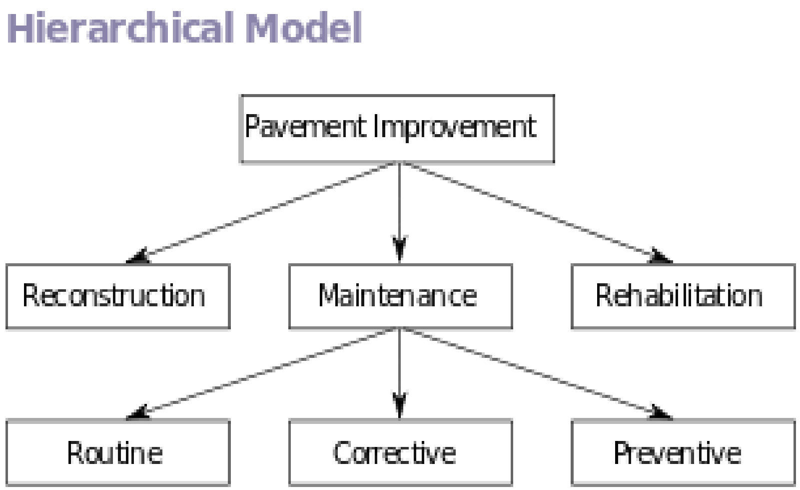
- **View level:** application programs hide details of data types. Views can also hide information (such as an employee's salary) for security purposes.

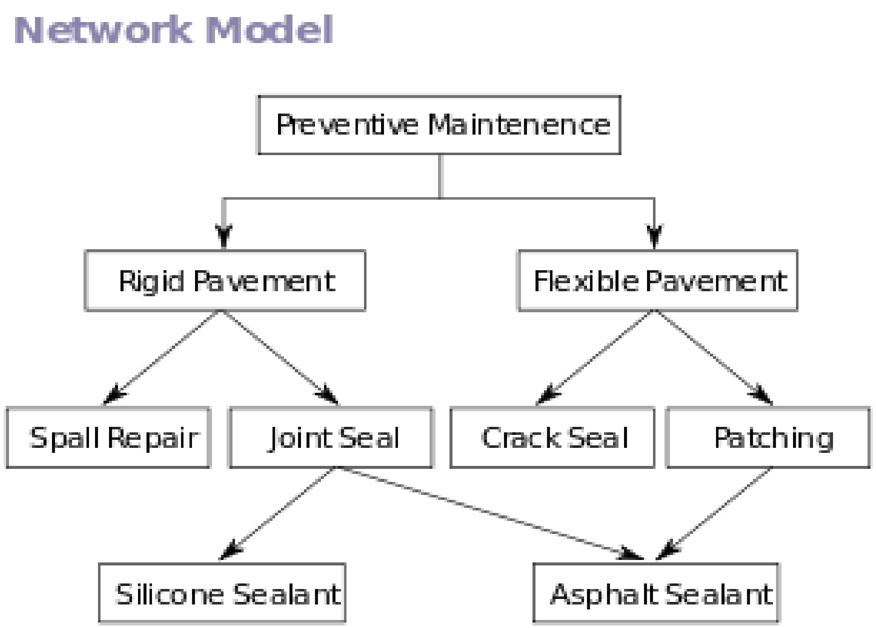
Data Models

- Data Model
- A collection of tools for describing
  - Data
  - Data relationships
  - Data semantics
  - Data constraints
- **Relational model**
- Entity-Relationship data model (mainly for database design)
- **Object-based data models** (Object-oriented and Object-relational)
- **Semi structured data model** (XML)

Data Models

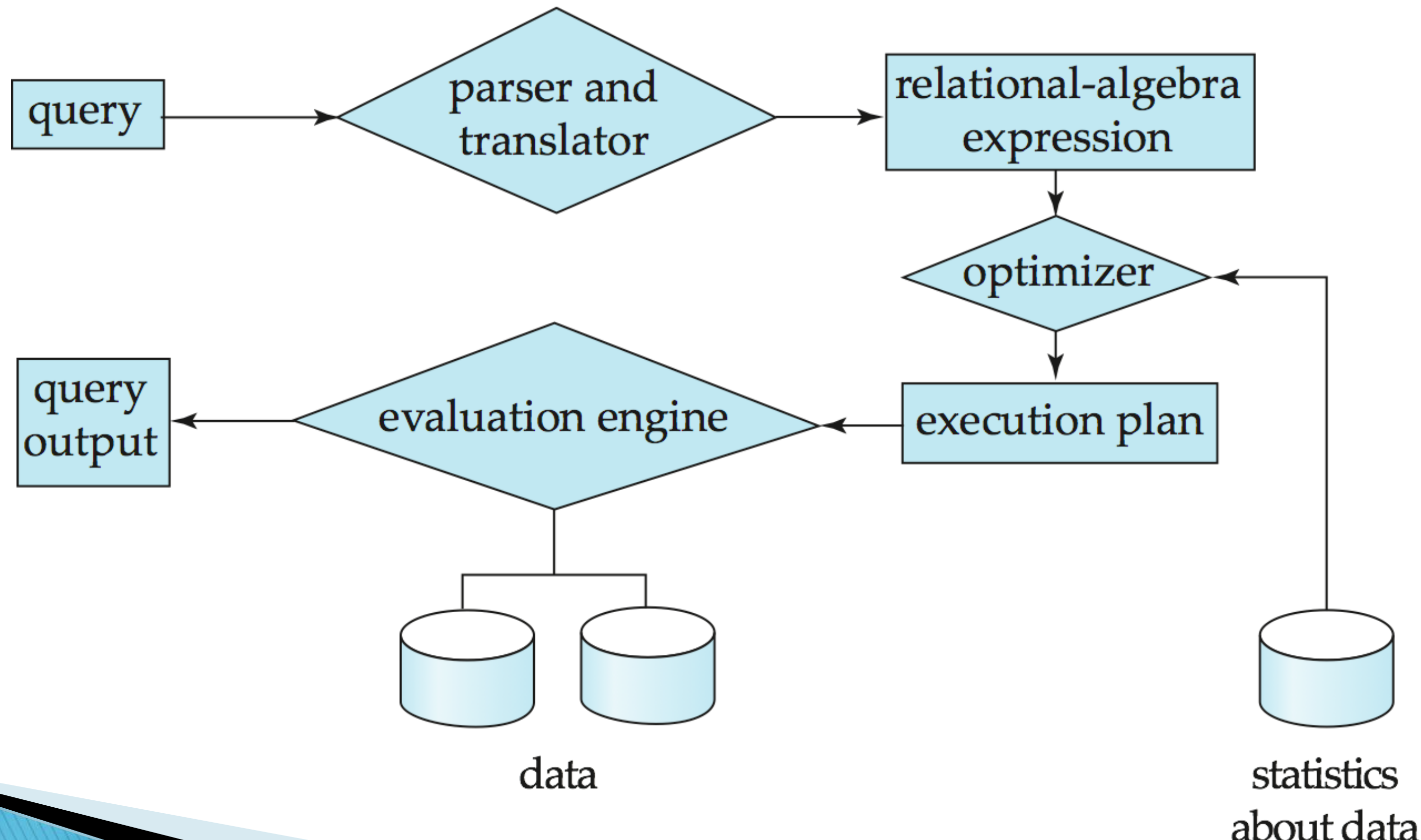
- Other older models:
  - Hierarchical model
    - Information Management System(IMS)
  - Network model
    - Integrated Data Store(IDS)





Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



Relational Model

- Relational model
- Example of tabular data in the relational model

Columns

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Rows

(a) The *instructor* table

A Sample Relational Database

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

Data Definition Language (DDL)

- Specification notation for defining the database schema

  Example: **create table** *instructor* (

  ```
  ID          char(5),
  name        varchar(20),
  dept_name   varchar(20),
  salary      numeric(8,2))
  ```

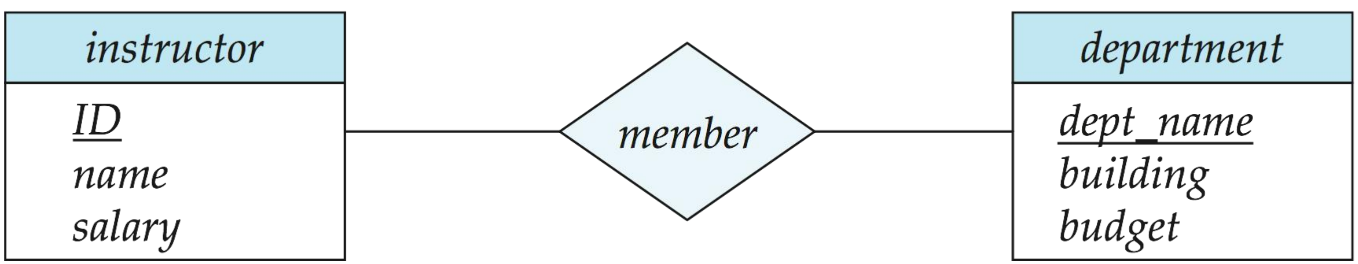
- DDL compiler generates a set of table templates stored in a ***data dictionary***
- Data dictionary contains metadata (i.e., data about data)
  - Database schema
  - Integrity constraints
    - Primary key (ID uniquely identifies instructors)
    - Referential integrity (**references** constraint in SQL)
      - e.g. *dept_name* value in any *instructor* tuple must appear in *department* relation
  - Authorization

Design Approaches

- Normalization Theory
- Formalize what designs are bad, and test for them
- Entity Relationship Model
- Models an enterprise as a collection of *entities* and *relationships*
    - **Entity:** a "thing" or "object" in the enterprise that is distinguishable from other objects
        - Described by a set of *attributes*
    - **Relationship**: an association among several entities
  - Represented diagrammatically by an ***entity-relationship diagram:***

The Entity-Relationship Model

- Models an enterprise as a collection of *entities* and *relationships*
  - **Entity**: a "thing" or "object" in the enterprise that is distinguishable from other objects
    - Described by a set of *attributes*
  - **Relationship**: an association among several entities
- Represented diagrammatically by an *entity-relationship diagram:*



What happened to dept_name of instructor and student?

Object–Relational Data Models

- Relational model: **flat, "atomic" values**
- **Object Relational Data Models**
  - Extend the relational data model by including object orientation and constructs to deal with added data types.
  - Allow attributes of tuples to have complex types, including non-atomic values such as nested relations.
  - Preserve relational foundations, in particular the declarative access to data, while extending modeling power.
  - Provide upward compatibility with existing relational languages.

XML: Extensible Markup Language

- Defined by the **WWW Consortium (W3C)**
- Originally intended as a document markup language not a database language
- The ability to specify new tags, and to create nested tag structures made XML a great way to exchange **data**, not just documents
- XML has become the basis for all new generation data interchange formats.
- A wide variety of tools is available for parsing, browsing and querying XML documents/data

Storage Management

- **Storage manager** is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible to the following tasks:
  - Interaction with the file manager
  - Efficient storing, retrieving and updating of data
- Issues:
  - Storage access
  - File organization
  - Indexing and hashing

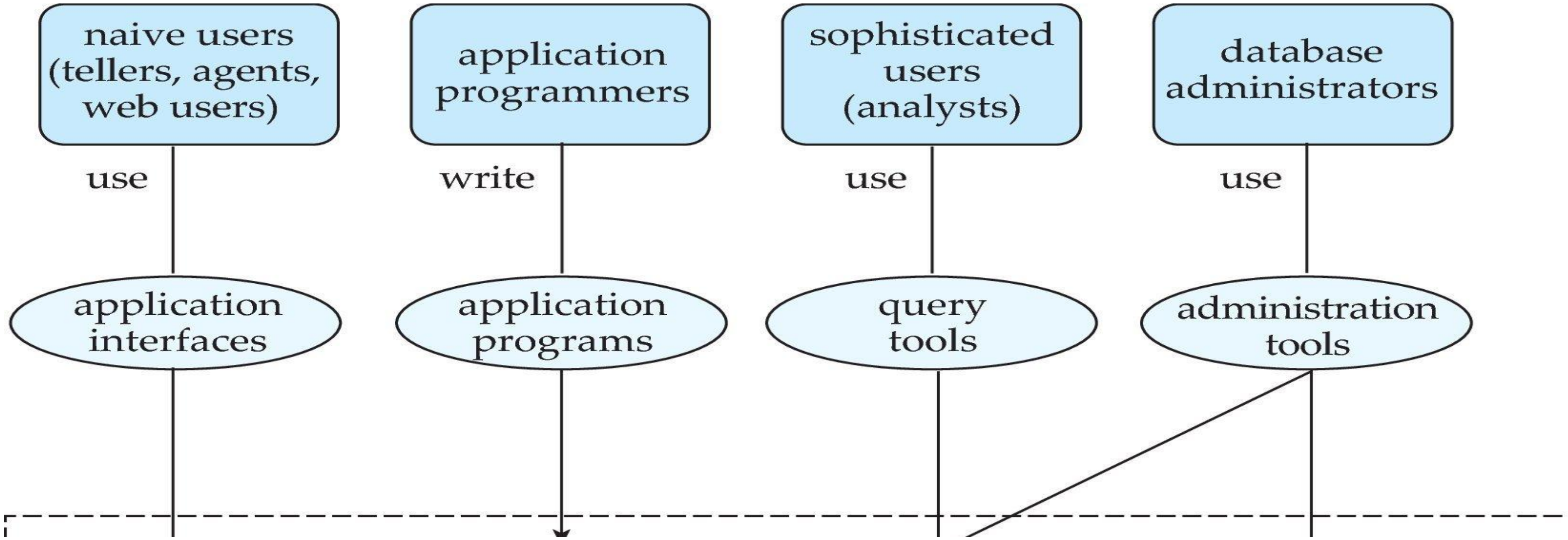
Query Processing (Cont.)

- 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

Transaction Management

- What if the system fails?
- What if more than one user is concurrently updating the same data?
- A **transaction** is a collection of operations that performs a single logical function in a database application
- **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.
- **Concurrency-control manager** controls the interaction among the concurrent transactions, to ensure the consistency of the database.

Database Users and Administrators



Database

Database Architecture

The architecture of a database systems is greatly influenced by
 the underlying computer system on which the database is running:

- Centralized
- Client-server
- Parallel (multi-processor)
- Distributed

History of Database Systems

- 1950s and early 1960s:
  - Data processing using magnetic tapes for storage
    - Tapes provided only sequential access
  - Punched cards for input
- Late 1960s and 1970s:
  - Hard disks allowed direct access to data
  - Network and hierarchical data models in widespread use
  - Ted Codd defines the relational data model
    - Would win the ACM Turing Award for this work
    - IBM Research begins System R prototype
    - UC Berkeley begins Ingres prototype
  - High-performance (for the era) transaction processing

History (cont.)

- 1980s:
  - Research relational prototypes evolve into commercial systems
    - SQL becomes industrial standard
  - Parallel and distributed database systems
  - Object-oriented database systems
- 1990s:
  - Large decision support and data-mining applications
  - Large multi-terabyte data warehouses
  - Emergence of Web commerce
- Early 2000s:
  - XML and XQuery standards
  - Automated database administration
- Later 2000s:
  - Giant data storage systems
    - Google BigTable, Yahoo PNuts, Amazon, ..