Advanced Database Management Systems

Lecture 1
Fundamental Database Concepts

What is a Database?

- A database is any collection of data.
- A DBMS is a software system designed to maintain a database.
- We use a DBMS when
  - there is a large amount of data
  - security and integrity of the data are important
  - many users access the data concurrently

Example Database Application

- Consider a Phone Company, such as AT&T
- Kinds of information they deal with:

customer records

employee records

billing information

management records

switching and wiring diagrams

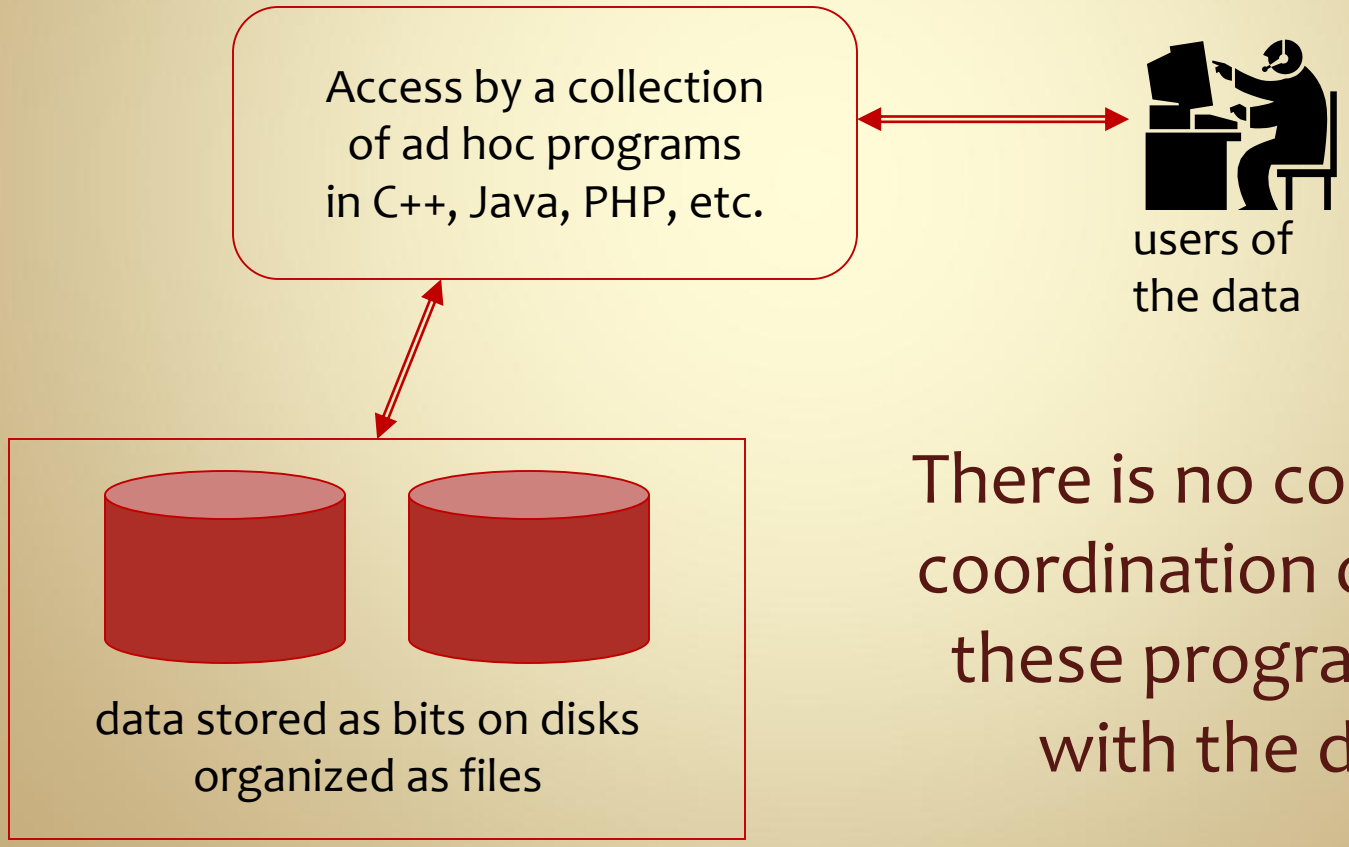
customer service orders

Concerns of a Database User

- With all that data,
  AT&T must be concerned with questions such as:
  - Where is the information kept?
  - How is the data structured?
  - How is the data kept consistent?
  - How is the data described?
  - How is the data kept secure?
  - How do different pieces of data interrelate?

Why Use a DBMS?

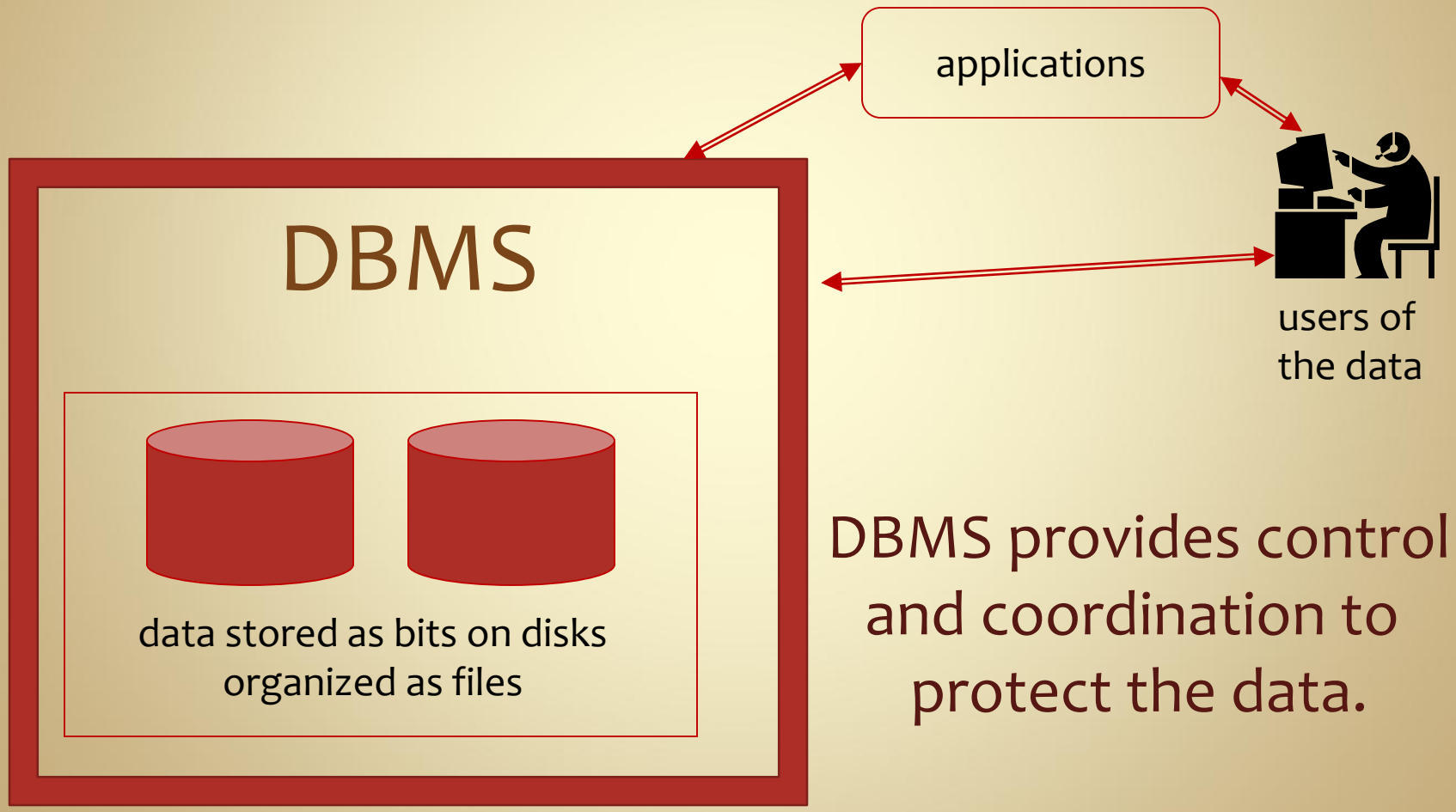
- Without a DBMS, we'd have:



There is no control or coordination of what these programs do with the data

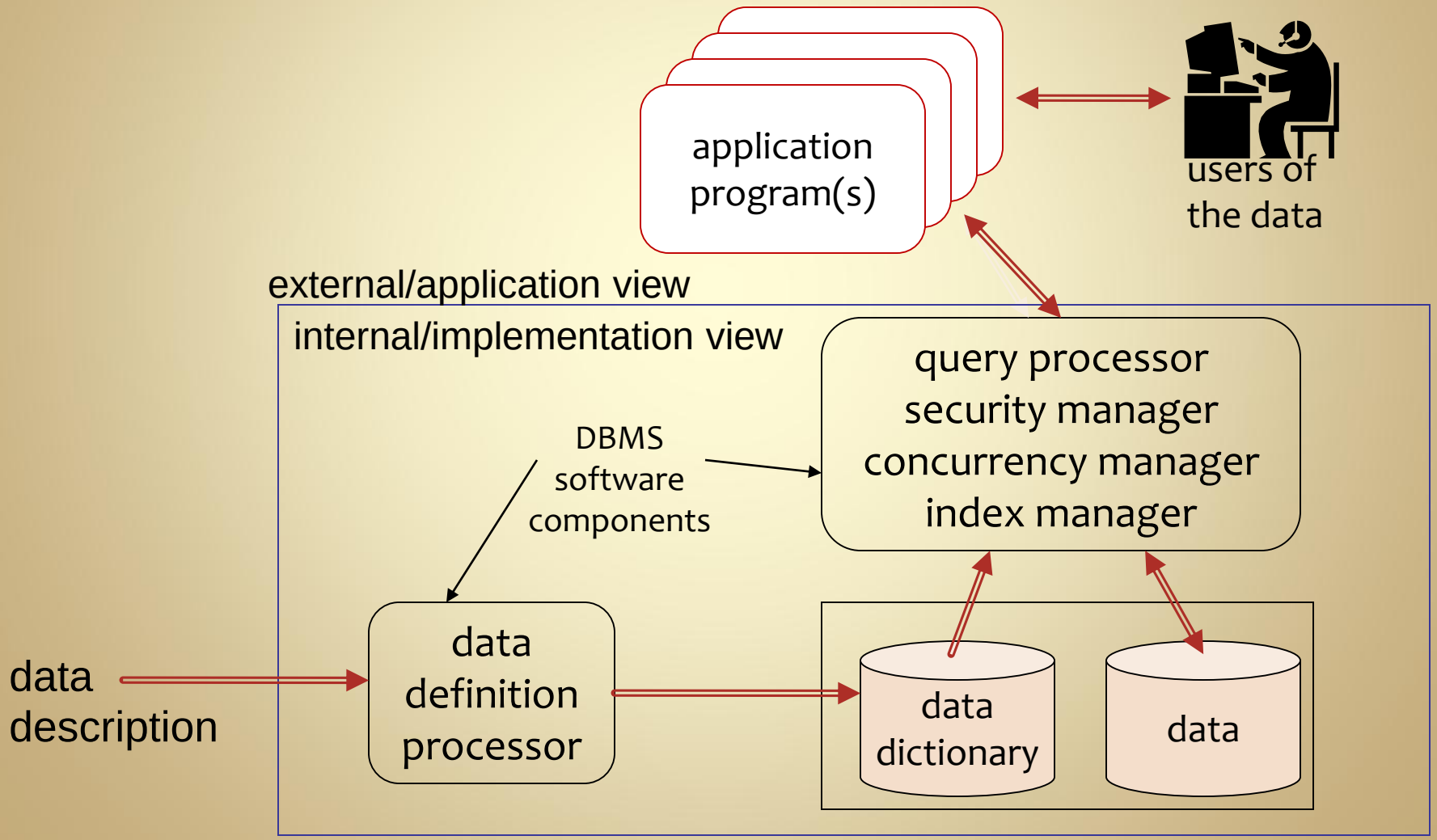
Why Use a DBMS?

- With a DBMS, we have:



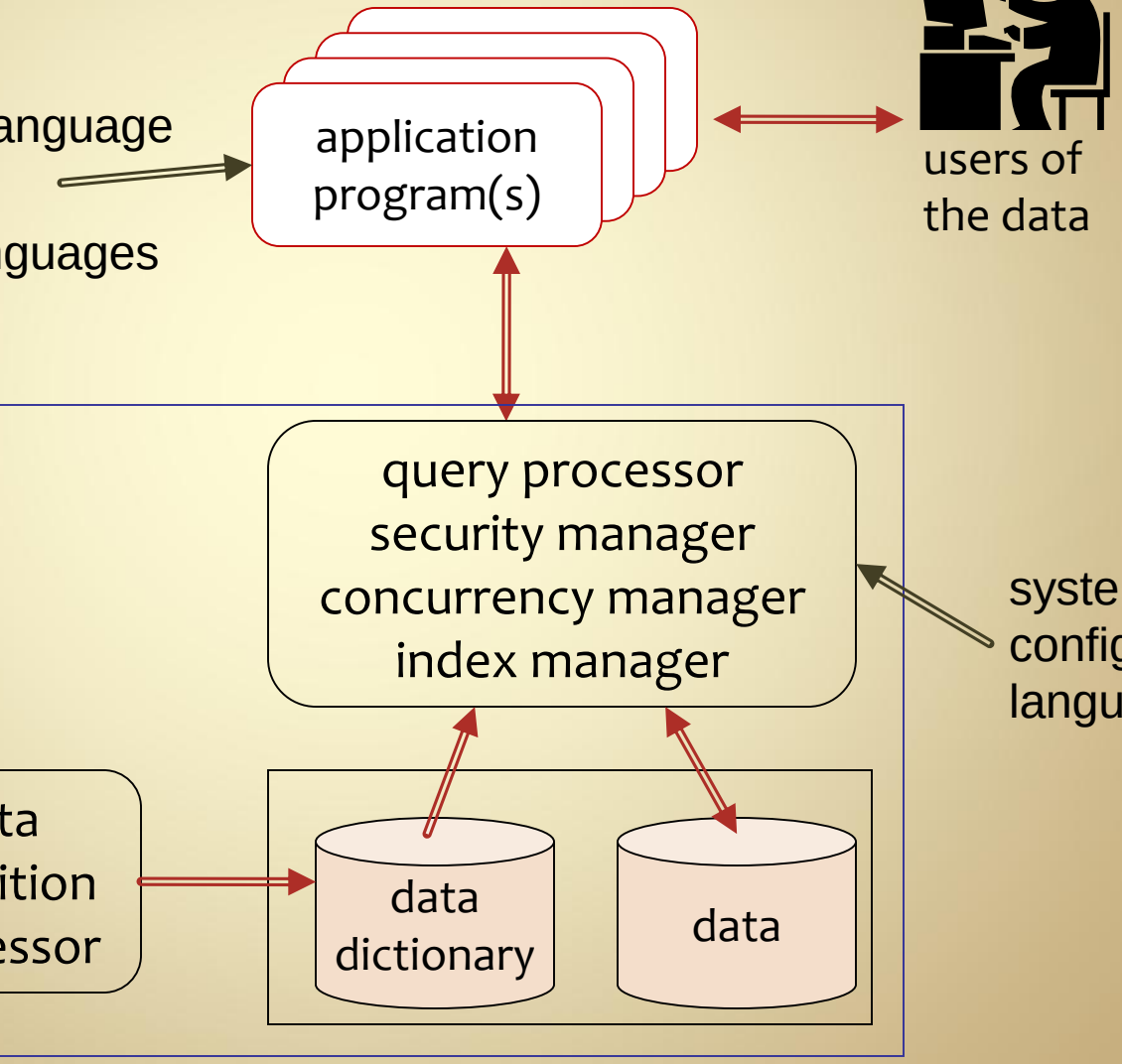
DBMS provides control and coordination to protect the data.

DBMS Structure



DBMS Languages

DML: data manipulation language
QL: query language
GPL: general purpose languages

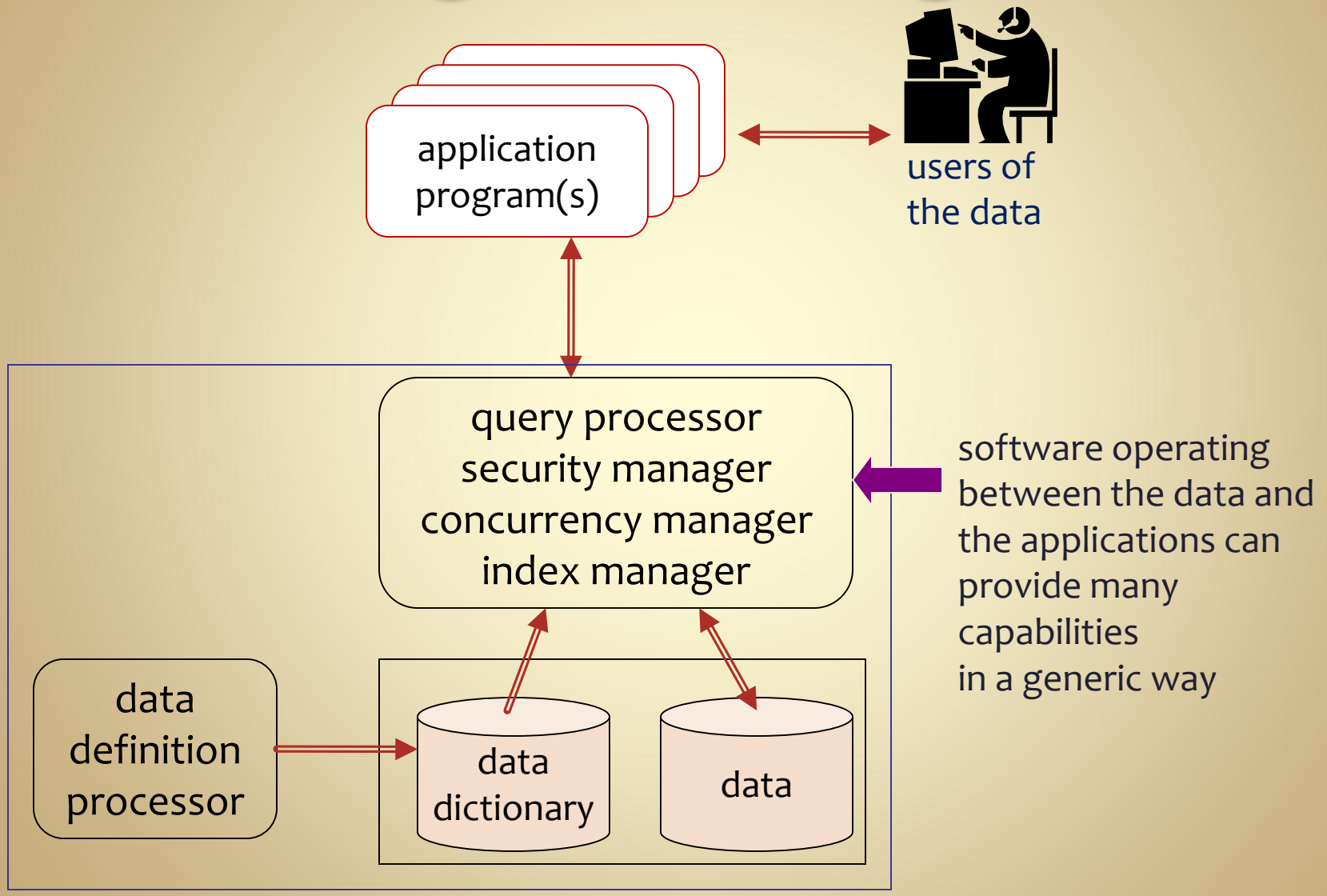


Advantages of Using a DBMS

- Anything you can do with a DBMS, you can do with a file system, a network and a heap of C code
- So why spend the money to buy a DBMS?
  - there is a well defined collection of capabilities common to a certain class of applications
  - for applications in this class, the DBMS already has these capabilities and probably does them better than you could with home-brewed code

Database Functionality

Advantages of Using a DBMS



Persistence

- A DBMS provides
  persistent objects, types and data structures

  - ***persistent*** = having a lifetime longer than
    the programs that use the data

  - any information that fits the data model
    of a particular DBMS
    can be made persistent with little effort

  - ***data model*** = concepts that can be used to describe the data

Concurrency

- A DBMS supports access by concurrent users
  - *concurrent* = happening at the same time
  - concurrent access, particularly writes (data changes), can result in inconsistent states (even when the individual operations are correct)
  - the DBMS can check the actual operations of concurrent users, to prevent activity that will lead to inconsistent states

Access Control

- A DBMS can restrict access to authorized users
  - security policies often require control that is more fine-grained than that provided by a file system
  - since the DBMS understands the data structure, it can enforce fairly sophisticated and detailed security policies
    - on subsets of the data
    - on subsets of the available operations

Redundancy Control

- A DBMS can assist in controlling redundancy
  - *redundancy* = multiple copies of the same data
  - with file storage, it's often convenient to store multiple copies of the same data, so that it's "local" to other data and applications
  - this can cause many problems:
    - wasted disk space
    - inconsistencies
    - need to enter the data multiple times

Complex Semantics

- A DBMS supports representation of complex relationships and integrity constraints
  - the semantics (meaning) of an application often includes many relationships and rules about the relative values of subsets of the data
  - these further restrict the possible instances of the database
  - relationships and constraints can be defined as part of the schema

Backup and Recovery

- A DBMS can provide backup and recovery
  - ***backup*** = snapshots of the data particular times
  - ***recovery*** = restoring the data to a consistent state after a system crash
  - the higher level semantics (relationships and constraints) can make it difficult to restore a consistent state
  - ***transaction analysis*** can allow a DBMS to reconstruct a consistent state from a number of backups

Views and Interfaces

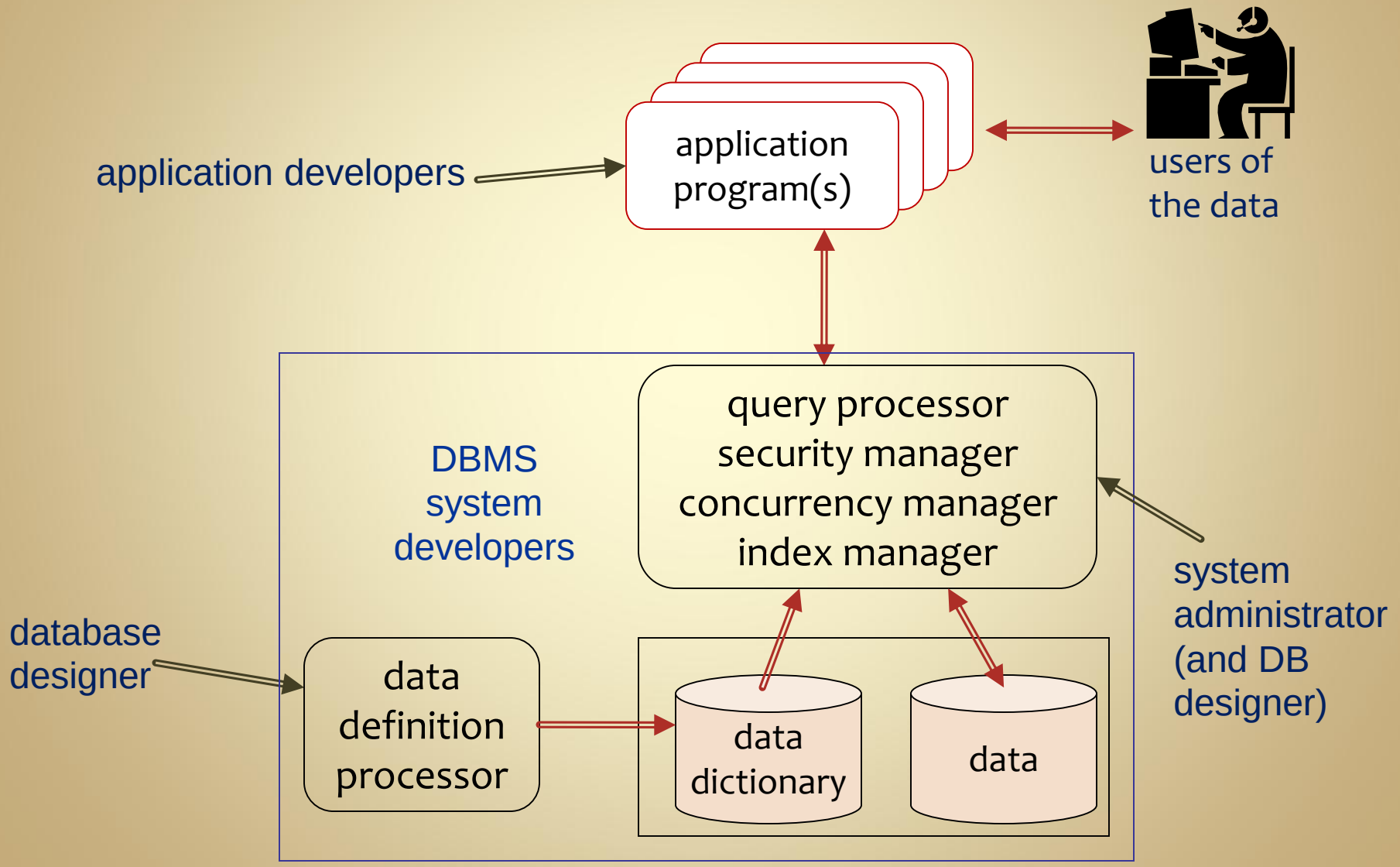
- A DBMS can support
  multiple user interfaces and user views
  - since the DBMS provides a well-defined data model and a persistent data dictionary, many different interfaces can be developed to access the same data
  - data independence ensures that these UIs will not be made invalid by most changes to the data
  - new user views can be supported as new schemas defined against the conceptual schema

Advantages of Using a DBMS

- persistent objects, types and data structures
- control of concurrent users
- controlling of redundancy
- restricting access (security)
- representation of complex relationships and integrity constraints
- backup and recovery
- multiple user interfaces and user views

Database Users and Roles

DBMS Roles


application developers
application program(s)
users of the data
DBMS system developers
query processor
security manager
concurrency manager
index manager
database designer
data definition processor
data dictionary
data
system administrator (and DB designer)

DBMS Roles

- **Actors On the Scene**
  (people interested in the actual data):
  - database administrators
  - database designers
  - systems analysts and application programmers
  - end users

Actors on the Scene

- Database Administrators
  - acquiring a DBMS
  - managing the system
  - acquiring HW and SW to support the DBMS
  - authorizing access (security policies)
  - managing staff, including DB designers

Actors on the Scene

- Database Designers
  - identifying the information of interested in the Universe of Discourse (UoD)
  - designing the database conceptual schema
  - designing views for particular users
  - designing the physical data layout and logical schema
  - adjusting data parameters for performance

Actors on the Scene

- Systems Analysts and Application Programmers (generic database developers)
  - provide specialized knowledge to optimize database usage
  - provide generic (canned) application programs

Actors on the Scene

- End Users
  - ***casual users*:** ad-hoc queries
  - ***naïve or parametric users*:** canned queries such as menus for a phone company customer service agent
  - ***sophisticated users*:** people who understand the system and the data and use it in many novel ways
  - ***standalone users*:** people who use personal easy-to-use databases for personal data

DBMS Roles

- **Actors Behind the Scene:**
  people who maintain the environment
  but aren't interested in the actual data

  - DBMS designers and implementers
  - tools developers
  - operators and maintenance personnel
  - database researchers

Actors Behind the Scene

- DBMS designers and implementers
  - work for the company that supplies the DBMS (i.e. Microsoft , Oracle, Sybase, MySQL …)
  - programmers and engineers
  - design and implement the DBMS

Actors Behind the Scene

- Tools Developers
  - design and implement DBMS add-ons or plug-ins
  - may work for DBMS supplier or be independent
  - kinds of tools: database design aids, performance monitoring tools, user and designer interfaces

Actors Behind the Scene

- Operators and maintenance personnel
  - run and maintain the computer environment in which a DBMS operates
  - probably work for the database administrator (DBA)

Actors Behind the Scene

- Database Researchers
  - academic or industrial researchers
  - develop new theory, new designs, new data models and new algorithms to improve future database management systems

Data Organization

Schemas and Instances

- A *database instance* is the collective values of all database objects at some point in time
  - also called the *(data) instance* or *(database) state*
- A *schema* describes the database and defines the possible instances
  - also called the *data definition, data dictionary, or meta-data*

Conceptual Data Models

- A *data model* describes the possible schemas (essentially the *meta-schema)*
- A DBMS is designed around a particular data model
  - this is what allows all system components (and humans) to understand the schema and data
- possible data models
  - ***relational***,
    object-oriented, object-relational,
    entity-relationship,
    semantic, network, hierarchical, etc.

Physical Data Models

- A ***physical data model*** describes the way in which data is stored in the computer
  - typically only of interest to database designers, implementers and maintainers … not end users
  - must provide a well-defined structure that can be mapped to the conceptual schema
  - allows optimization strategies to be defined generically

Three-Schema Architecture

user-specific views

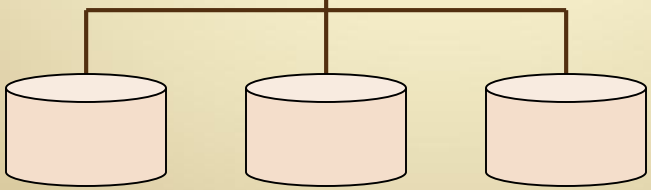
External View | External View | External View

Conceptual Schema

generic view

Internal Schema

physical view



Data Independence

- **physical data independence**
  - conceptual and external schema are defined in terms of the data model, rather than the actual data layout
  - ensures that conceptual and external schemas are not affected by changes to the physical data layout
- **logical data independence**
  - ensures that changes to the conceptual schema don't affect the external views
  - (this is not always achievable)

Transactions

Transactions

- ***transaction*** = an indivisible unit of data processing

- All transactions must have the ***ACID*** properties:

  - **Atomicity:** all or nothing
  - **Consistency:** no constraint violations
  - **Isolation:** no interference from other concurrent transactions
  - **Durability:** committed changes must not be lost due to any kind of failure

Atomic Transactions

- Fred wants to move $200 from his savings account to his checking account.

1) Money must be subtracted from savings account.
2) Money must be added to checking count.

If both happen, Fred and the bank are both happy.
If neither happens, Fred and the bank are both happy.

If only one happens, either Fred or the bank will be unhappy.

Fred's transfer must be *all or nothing*.

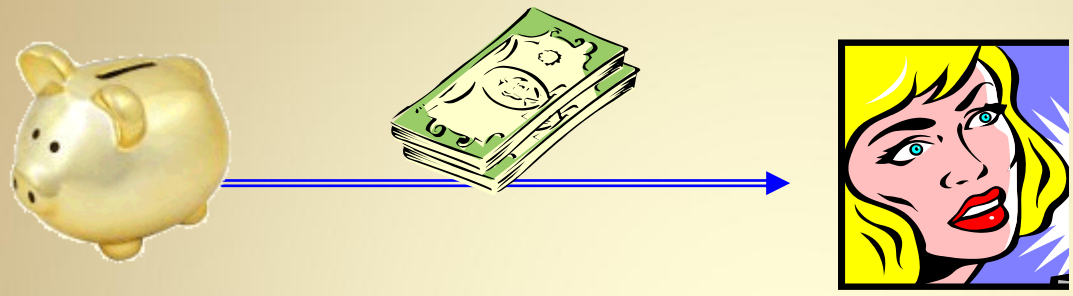
Transactions are Atomic

- Transactions must be ***atomic*** (indivisible)
  - the DBMS must ***ensure*** atomicity
  - everything happens, or nothing happens
  - boundaries of transaction (in time) are generally set by the application …

    the DBMS has no means of determining the intention of a transaction

Correct Transactions

- Wilma tries to withdraw $1000 from account 387.



Accounts

No	PIN	Balance
101	8965	10965.78
387	6643	652.55
543	4287	8720.12

constraint:
account.Balance must be non-negative

any transaction withdrawing
more than $652.55 from acct 387
will violate this constraint

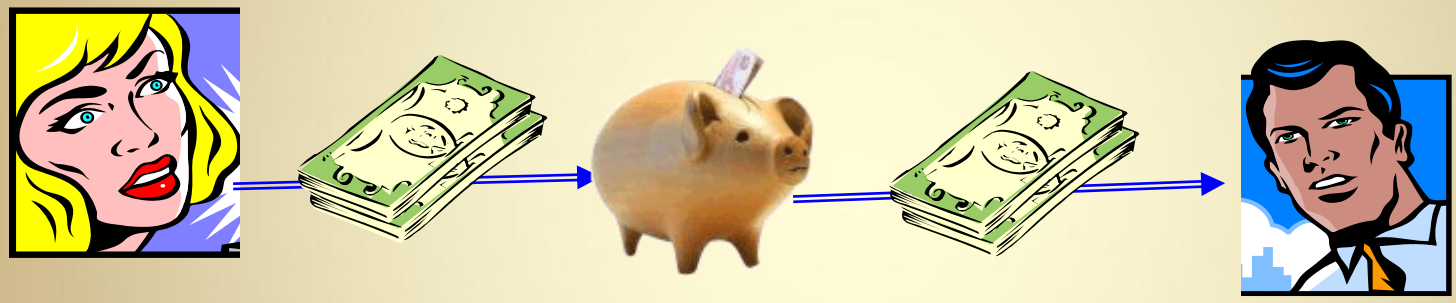
Wilma's transaction cannot be accepted.

Transactions are Consistent

- A transaction must leave the database in an valid or consistent state
  - valid state == no constraint violations
- A *constraint* is a declared rule defining specifying database states
- Constraints may be violated temporarily … but must be corrected before the transaction completes

Concurrent Transactions

- Fred is withdrawing $500 from account 543.
- Wilma's employer is depositing $1983.23 to account 543.
- These transactions are happening *at the same time*.



Accounts

No	PIN	Balance
101	8965	10965.78
387	6643	652.55
543	4287	8720.12

Accounts

No	PIN	Balance
101	8965	10965.78
387	6643	652.55
543	4287	10233.35

Combined result of both transactions must be correct

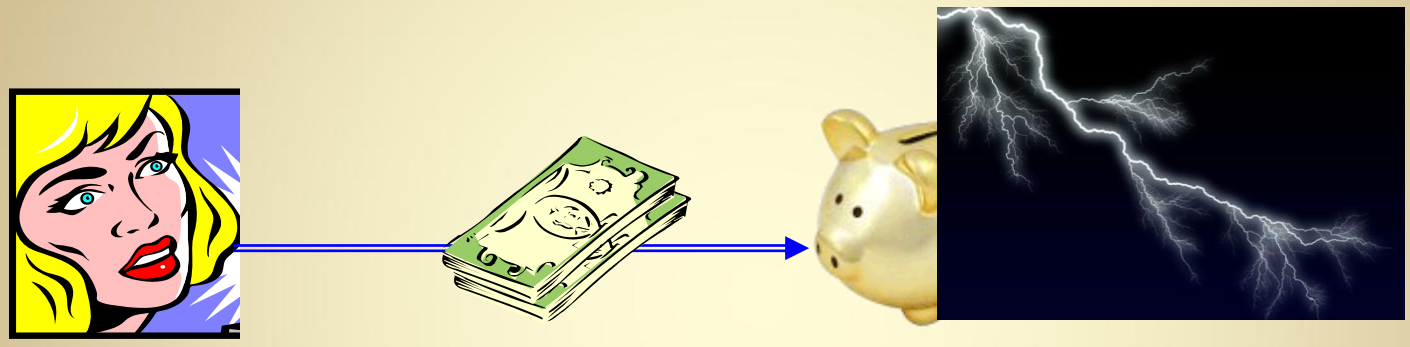
Transactions are Isolated

- If two transactions occur at the same time, the cumulative effect must be the same as if they had been done in *isolation*
  - ($8720.12 - $500) + $1983.23 = $10233.35
  - ($8720.12 + $1983.23) - $500 = $10233.35
  - $$\$8720.12 + \begin{bmatrix} -\$500.00 \\ +\$1983.23 \end{bmatrix} = \$10233.35$$

    happen concurrently
- Ensuring isolation is the task of *concurrency control*

Durable Transactions

- Wilma deposits $50,000 to account 387.
- Later, the bank's computer crashes due to a lightning storm.



Accounts

No	PIN	Balance
101	8965	10965.78
387	6643	652.55
543	4287	8720.12

Accounts

No	PIN	Balance
101	8965	10965.78
387	6643	50652.55
543	4287	8720.12

Wilma's deposit cannot be lost.

Transactions are Durable

- Once a transaction's effect on the database state has been *committed*, it must be *permanent*
- The DBMS must ensure *persistence*, even in the event of system failures
- Sources of failure:
  - computer or operating system crash
  - disk failure
  - fire, theft, power outage, earthquake, operator errors, …

Transactions

- ***transaction*** = an indivisible unit of data processing

- All transactions must have the ***ACID*** properties:

  - **Atomicity:** all or nothing
  - **Consistency:** no constraint violations
  - **Isolation:** no interference from other concurrent transactions
  - **Durability:** committed changes must not be lost due to any kind of failure