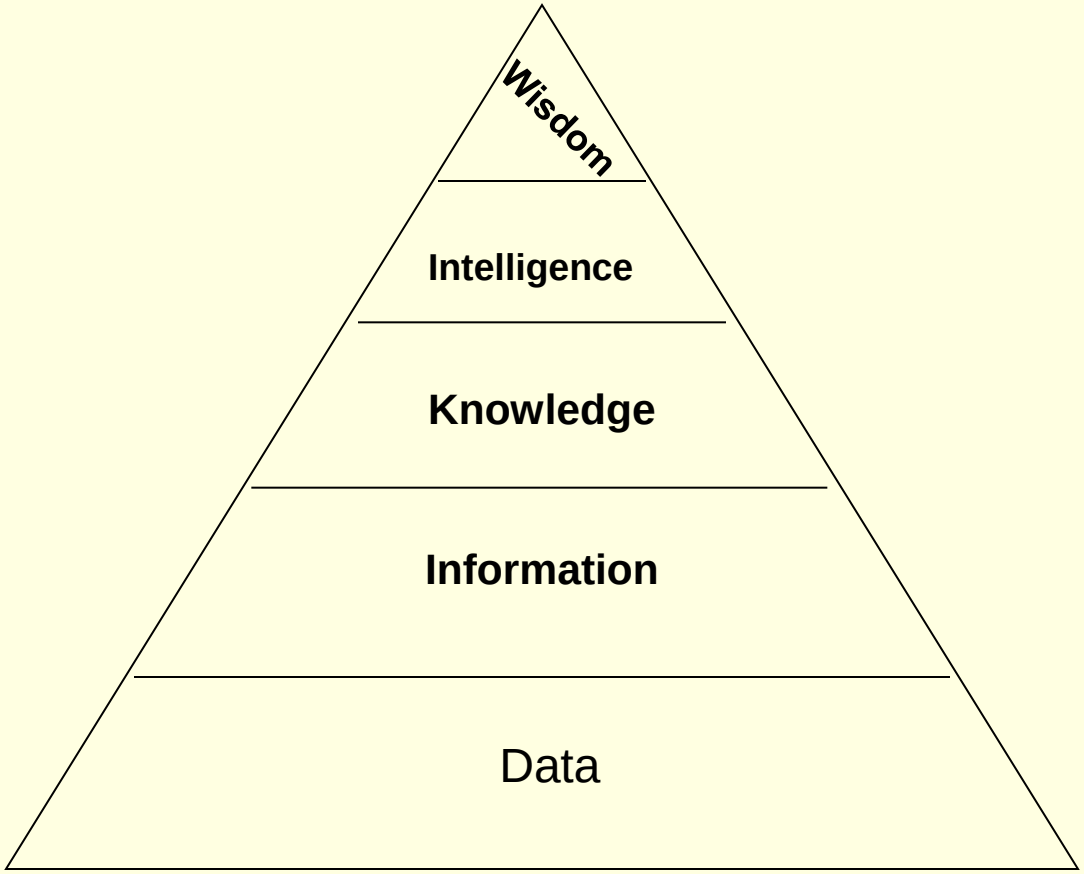
UNIT - 1

INTRODUCTION TO DATA BASE

Basic Definitions

- Data – A collection of facts from which conclusion may be drawn such as “statistical data”. Data is the plural form of datum.
- Information – It is the result of processing, manipulating and organizing data in a way that adds to the knowledge of the person receiving it.
- Knowledge – Collection of information is known as knowledge.
- Intelligence – It is the property of mind that encompasses many related abilities such as to reason, to plan, to solve, to think abstractly, to learn.
- Wisdom – The ability to discern or judge what is true, right or lasting, insight.

HIERARCHICAL STRUCTURE



What is a Database?

- Database is a collection of related data, that contains information relevant to an enterprise.
- For example:

1. University database
2. Employee database
3. Student database
4. Airlines database

etc…..

PROPERTIES OF A DATABASE

- A database represents some aspect of the real world, sometimes called the miniworld or the universe of discourse (UoD).
- A database is a logically coherent collection of data with some inherent meaning.
- A database is designed, built and populated with data for a specific purpose.

What is a Database System?

A DBMS is a general-purpose software system that facilitates the processes of *defining, constructing, manipulating and sharing* database among various users and application.

Typical DBMS Functionality

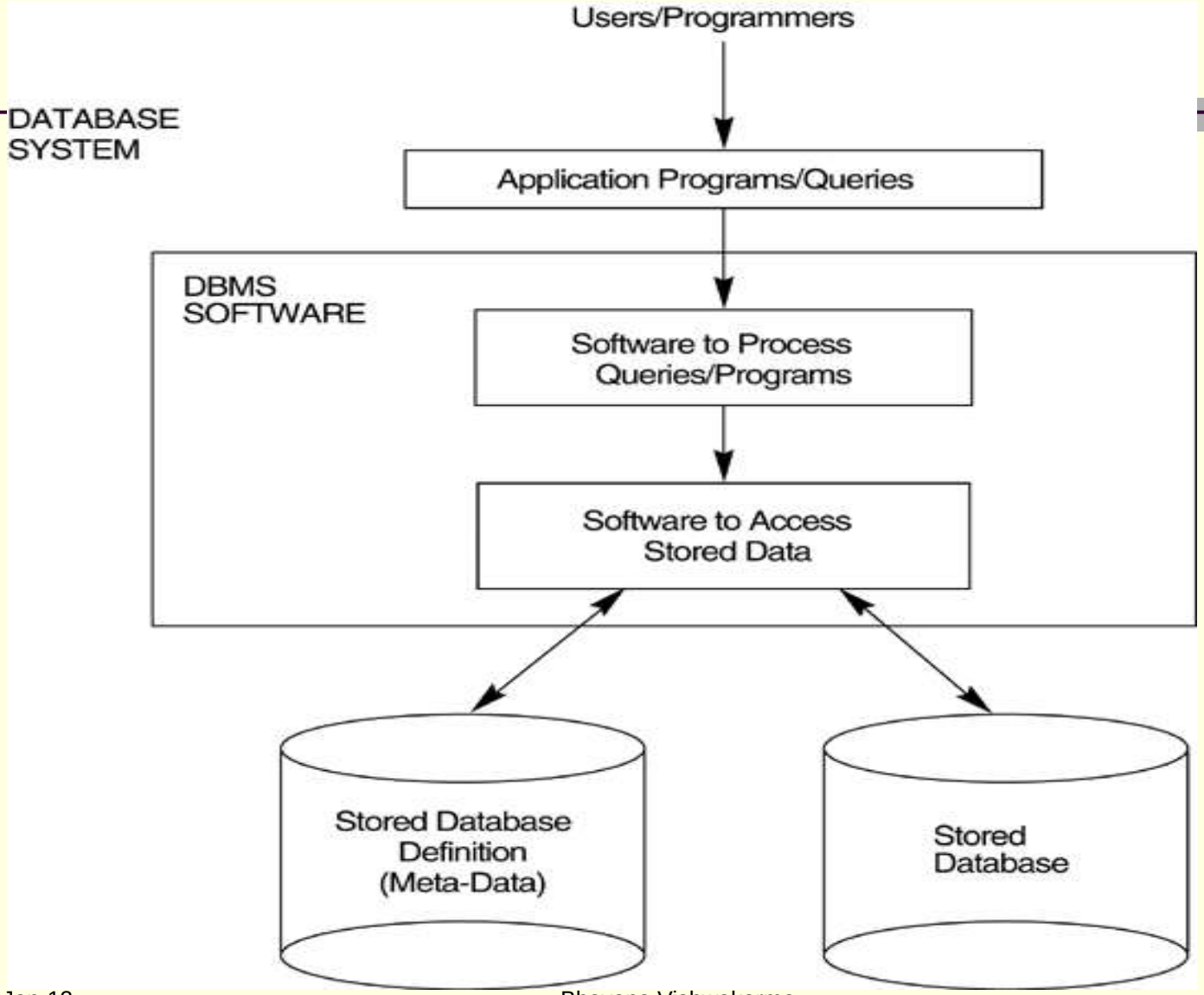
- Define a database : in terms of data types, structures and constraints
- Construct or Load the Database on a secondary storage medium
- Manipulating the database : querying, generating reports, insertions, deletions and modifications to its content
- Concurrent Processing and Sharing by a set of users and programs – yet, keeping all data valid and consistent

Typical DBMS Functionality

Other features:

- Protection or Security measures to prevent unauthorized access
- “Active” processing to take internal actions on data
- Presentation and Visualization of data

FIGURE 1.1 A simplified database system environment.



Database System Applications

- Banking
- Airlines
- Universities
- Credit card transactions
- Telecommunication
- Finance
- Sales
- Manufacturing
- Human resources

Disadvantages of File System over DBMS

- 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
  - Must satisfy certain types of consistency constraints.
  - Hard to add new constraints or change existing ones

- Drawbacks of using file systems (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 accessed needed for performance
    - Uncontrolled concurrent accesses can lead to inconsistencies
      - Example: Two people reading a balance and updating it 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

Advantages of DBMS

- Controlling Redundancy
- Restricting Unauthorized Access
- Providing Storage Structures for Efficient Query Processing
- Providing Backup and Recovery
- Providing Multiple User Interfaces
- Representing Complex Relationship among Data
- Enforcing Integrity Constraints
- Permitting Inferencing and Actions using Rules

Database Users

Users may be divided into those who actually use and control the content (called “Actors on the Scene”) and those who enable the database to be developed and the DBMS software to be designed and implemented (called “Workers Behind the Scene”).

Database Users

Actors on the scene

- **Database administrators:** responsible for authorizing access to the database, for co-ordinating and monitoring its use, acquiring software, and hardware resources, controlling its use and monitoring efficiency of operations.
- **Database Designers:** responsible to define the content, the structure, the constraints, and functions or transactions against the database. They must communicate with the end-users and understand their needs.
- **End-users:** they use the data for queries, reports and some of them actually update the database content.

Categories of End-users

- **Casual** : access database occasionally when needed
- **Naïve or Parametric** : they make up a large section of the end-user population. They use previously well-defined functions in the form of “canned transactions” against the database. Examples are bank-tellers or reservation clerks who do this activity for an entire shift of operations.

Categories of End-users

- **Sophisticated** : these include business analysts, scientists, engineers, others thoroughly familiar with the system capabilities. Many use tools in the form of software packages that work closely with the stored database.
- **Stand-alone** : mostly maintain personal databases using ready-to-use packaged applications. An example is a tax program user that creates his or her own internal database.

Data Models

- **Data Model**: A set of concepts to describe the *structure* of a database, and certain *constraints* that the database should obey.
- **Data Model Operations**: Operations for specifying database retrievals and updates by referring to the concepts of the data model. Operations on the data model may include *basic operations* and *user-defined operations*.

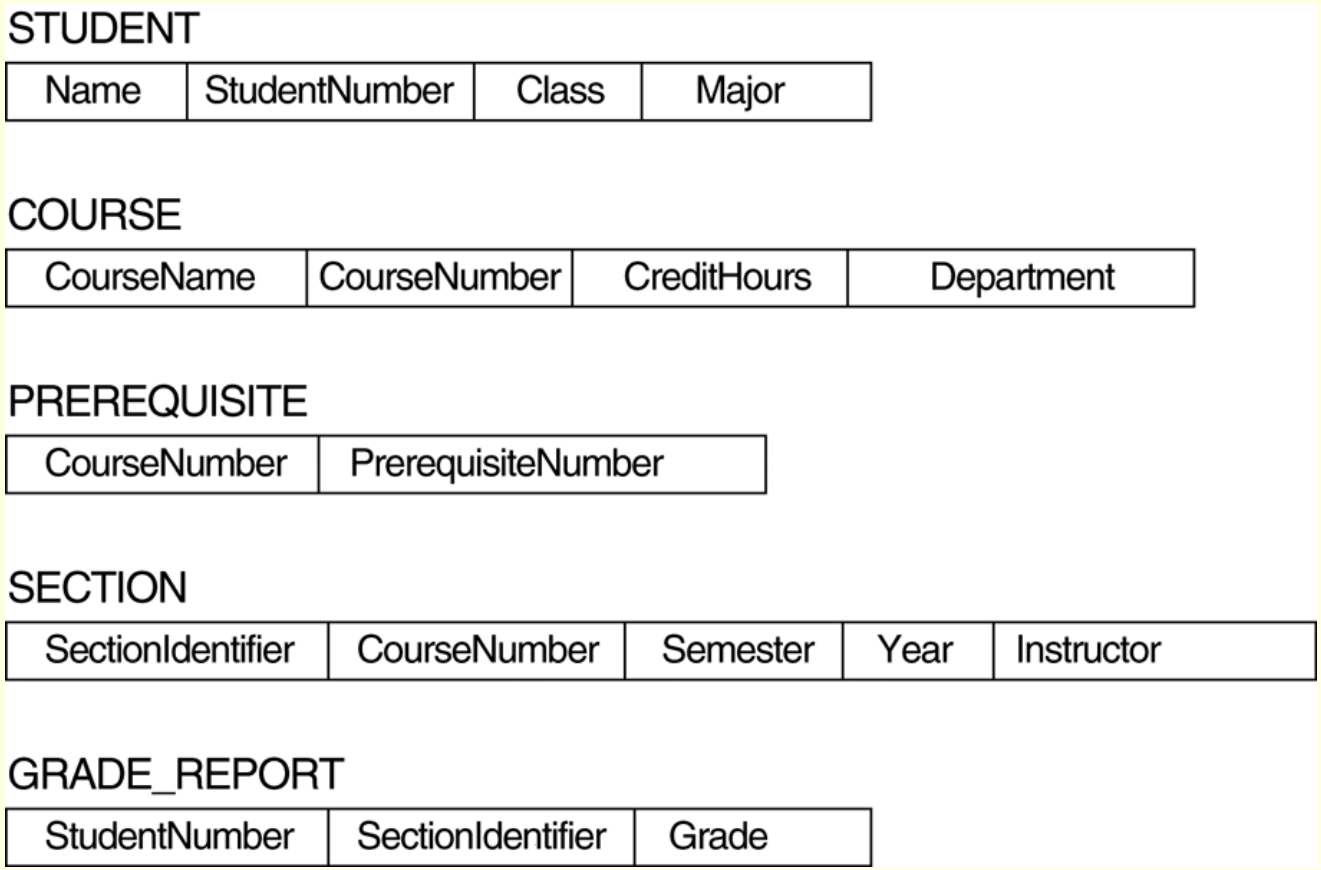
Categories of data models

- **Conceptual** (**high-level**, **semantic**) data models: Provide concepts that are close to the way many users *perceive* data. (Also called **entity-based** or **object-based** data models.)
- **Physical** (**low-level**, **internal**) data models: Provide concepts that describe details of how data is stored in the computer.
- **Implementation** (**representational**) data models: Provide concepts that fall between the above two, balancing user views with some computer storage details.

Definitions

- **Database Schema**: The *description* of a database. Includes descriptions of the database structure and the constraints that should hold on the database.
- **Schema Diagram**: A diagrammatic display of (some aspects of) a database schema.
- **Schema Construct**: A component of the schema or an object within the schema, e.g., STUDENT, COURSE.
- **Database Instance**: The actual data stored in a database at a *particular moment in time*. Also called **database state** (or **occurrence**).

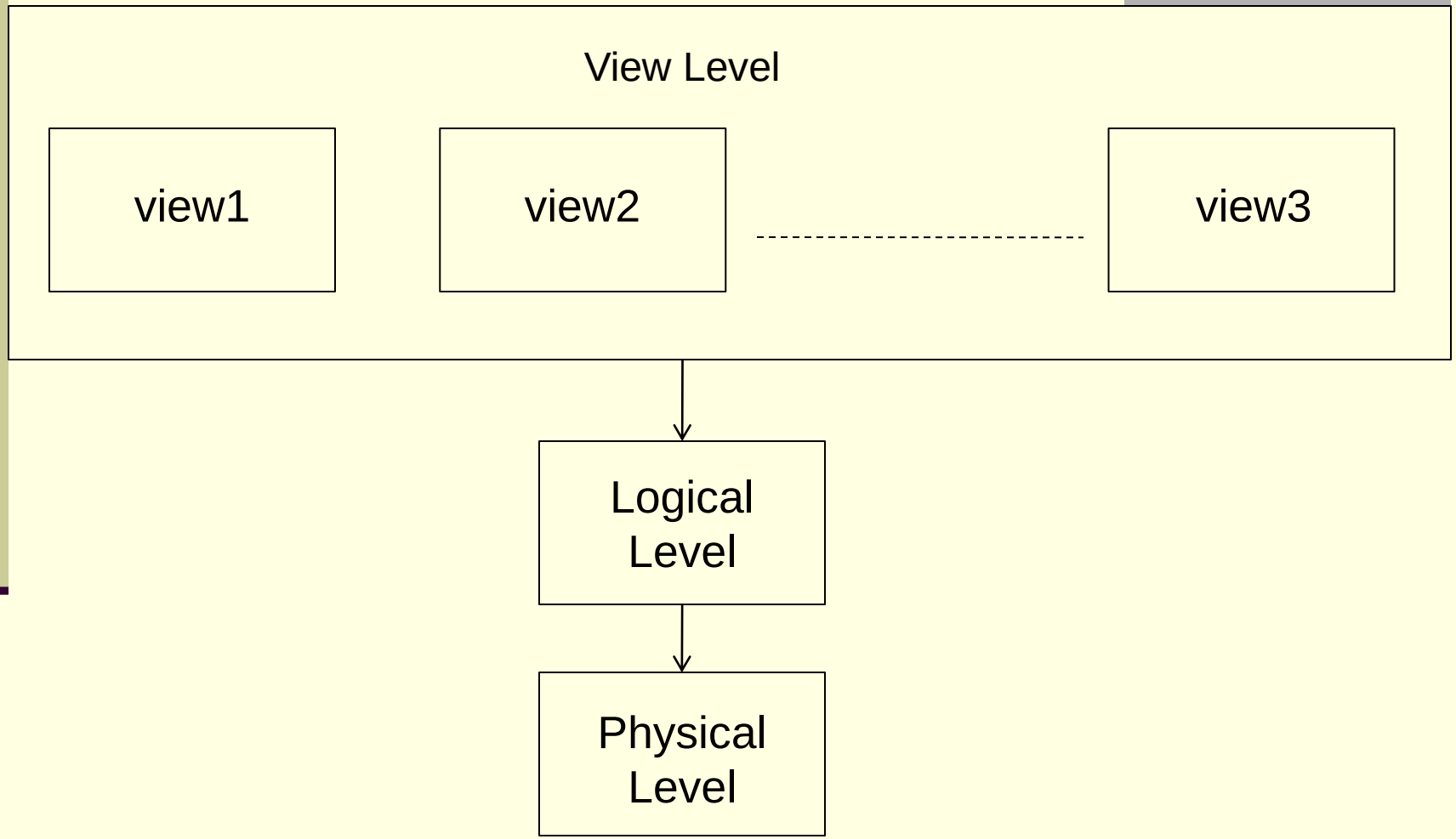
Schema diagram for the database



Database Schema Vs. Database State

- **Database State:** Refers to the content of a database at a moment in time.
- **Initial Database State:** Refers to the database when it is loaded
- **Valid State:** A state that satisfies the structure and constraints of the database.
- **Distinction**
  - The **database schema** changes *very infrequently*. The **database state** changes *every time the database is updated.*
  - **Schema** is also called **intension**, whereas **state** is called **extension**.

LEVELS OF DATA ABSTRACTION



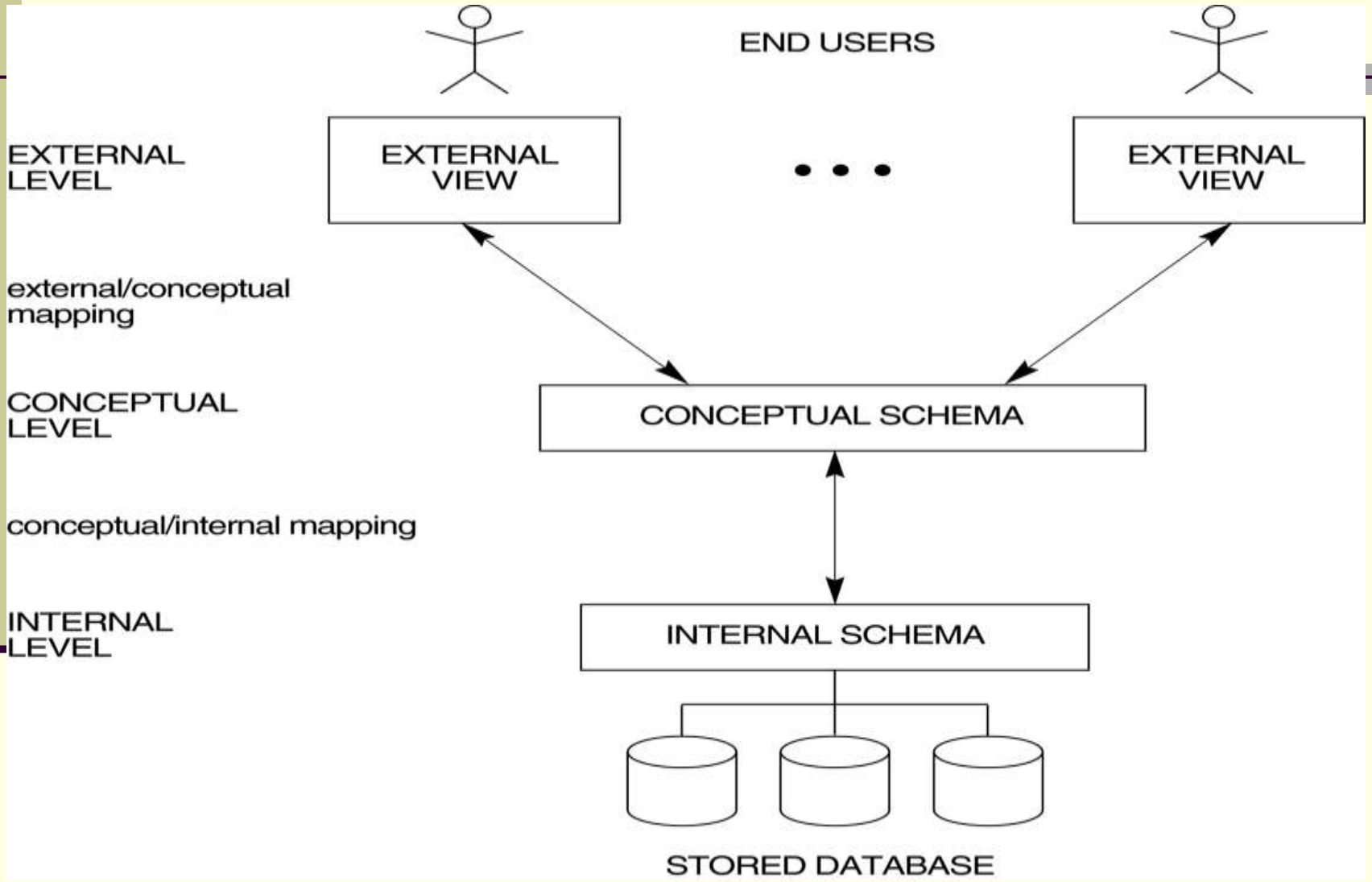
LEVELS OF DATA ABSTRACTION

- Physical Level –
  - Lowest level of abstraction
  - Describes how data are actually stored
  - Describes complex low-level data structures
- Logical Level –
  - Describes what data are stored in the database.
  - Describes relationship among the data.
  - Describes entire database in terms of a small no.of relatively simple structures.
- View Level –
  - Describes only part of the entire database.
  - Exists to simplify data interaction with the system

Three-Schema Architecture

- Proposed to support DBMS characteristics of:
  - **Program-data independence**.
  - Support of **multiple views** of the data.

The three-schema architecture.



Three-Schema Architecture

- Defines DBMS schemas at *three levels*:
  - **Internal schema** at the internal level to describe physical storage structures and access paths. Typically uses a *physical* data model.
  - **Conceptual schema** at the conceptual level to describe the structure and constraints for the *whole* database for a community of users. Uses a *conceptual* or an *implementation* data model.
  - **External schemas** at the external level to describe the various user views. Usually uses the same data model as the conceptual level.

Three-Schema Architecture

Mappings among schema levels are needed to transform requests and data. Programs refer to an external schema, and are mapped by the DBMS to the internal schema for execution.

Ex: University Database

- Conceptual Schema
  - Student(sid: string, name: string, age: number, percent: real)
  - Courses(cid: string, cname: string, credits: number)
  - Enrolled(sid:string, cid: string, grade: string)
- Physical Schema
  - Relations stored as unordered files
  - Index on first column of Students
- External Schema
  - Course_info(cid: string, enrollment: integer)

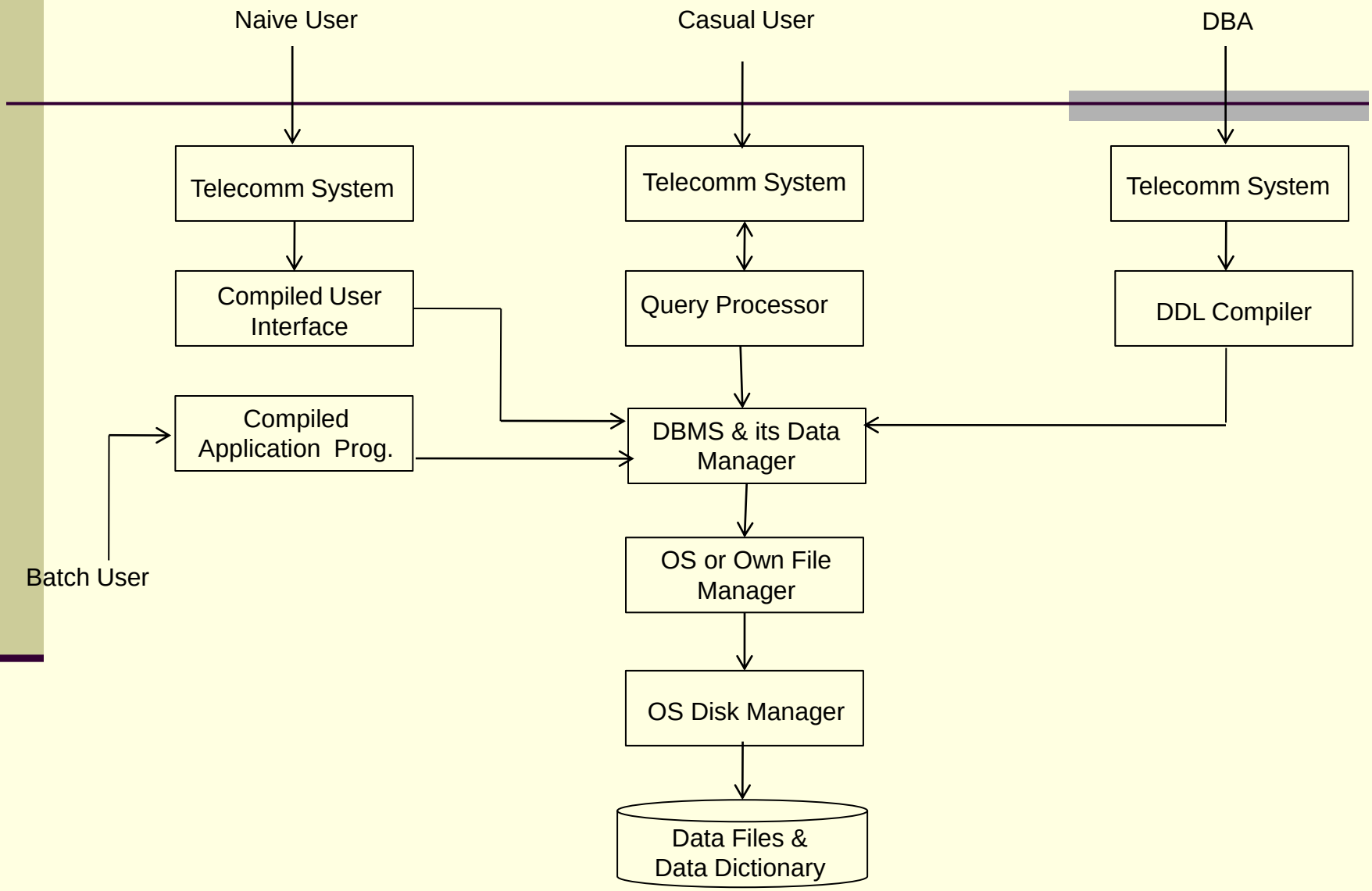
Data Independence

When a schema at a lower level is changed, only the **mappings** between this schema and higher-level schemas need to be changed in a DBMS that fully supports data independence. The higher-level schemas themselves are *unchanged*. Hence, the application programs need not be changed since they refer to the external schemas.

Data Independence

- **Logical Data Independence**: The capacity to change the conceptual schema without having to change the external schemas and their application programs.
- **Physical Data Independence**: The capacity to change the internal schema without having to change the conceptual schema.

DBMS ARCHITECTURE



E-R Model

- Stands for Entity-Relational Model.
- It is an abstract conceptual representation of structured data.
- It is not implemented but have the design for creating the database.

Terms used in E-R model:

Field – Attribute

Record – Entity

File – Entity Type

E-R Model

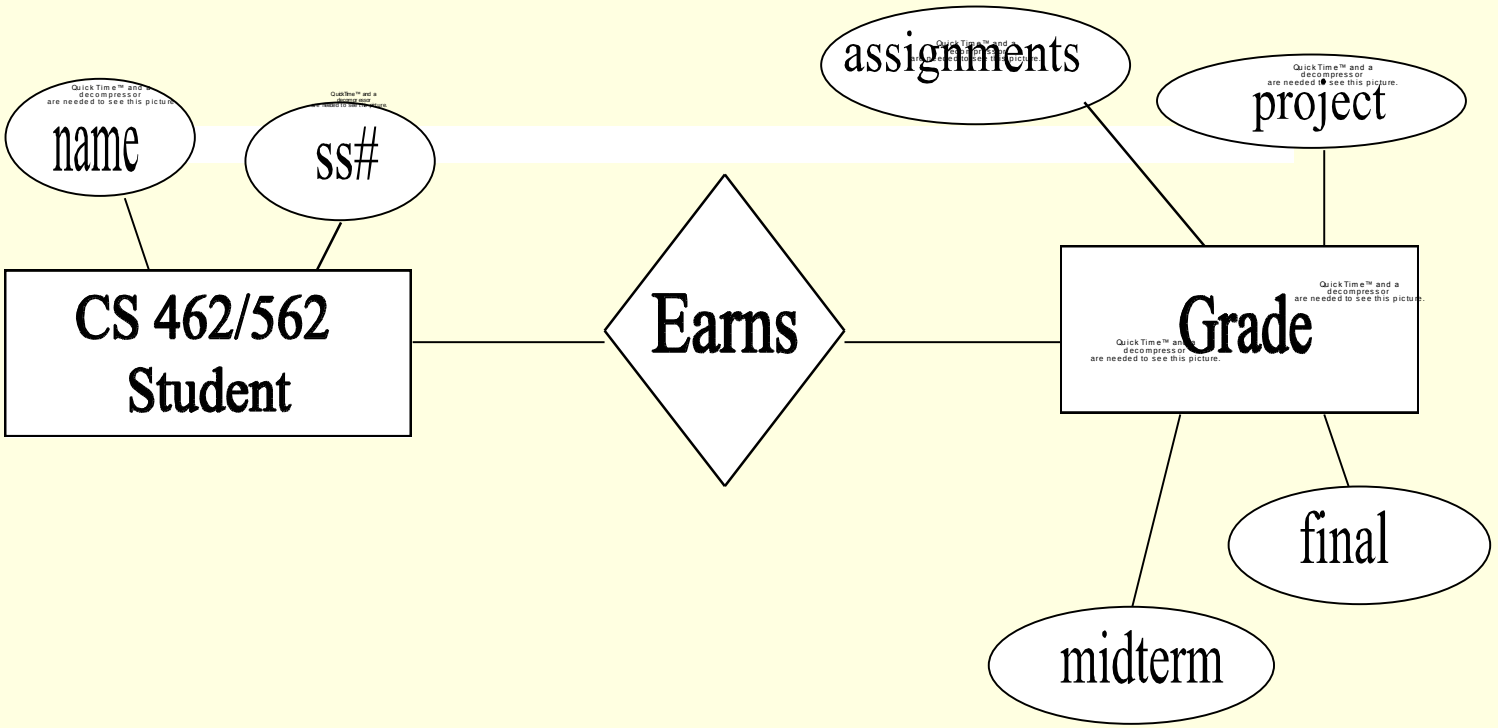
Entity – It is an object with a physical existence.

Ex: An object with a physical existence – a person, a car, a house or it may be an object with conceptual existence – a company, a job or a university.

Attribute – Attributes are the particular properties that describe an entity.

Ex: A STUDENT entity may be described by student's name, age, address, class, grade.

EXAMPLE

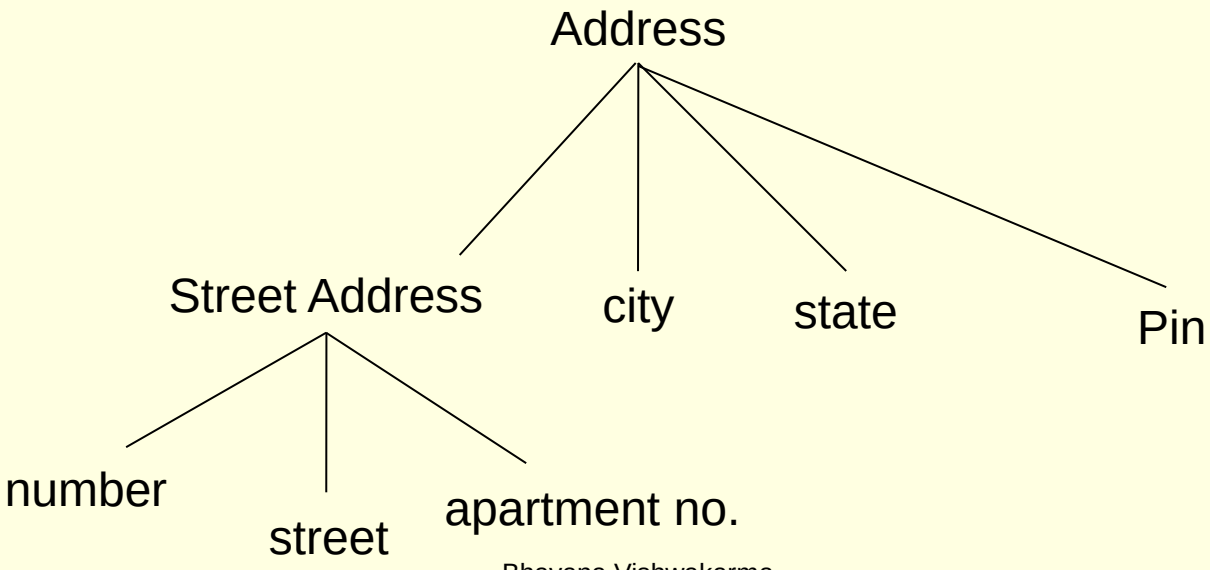

assignments
project
name
ss#
CS 462/562
Student
Earns
Grade
final
midterm

TYPES OF ATTRIBUTES

Simple and Composite Attributes –

- Simple attributes are not divisible into parts.

  Ex: Name, Age
- Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings.



TYPES OF ATTRIBUTES

Single Valued & Multivalued Attributes –

- Single-valued attributes have a single value for particular entity.

  Ex: Roll_no, Age
- Multivalued attributes may have more than one value for a single enitity .

  Ex: Phone_no

TYPES OF ATTRIBUTES

Stored and Derived Attributes –

Derived attribute is not stored in the database but it is derived from some attributes.

Ex: If DOB is stored in the database then we can calculate age of a student by subtracting DOB from current date.

Hence, in this case DOB is the stored attribute and age is considered as derived.

TYPES OF ATTRIBUTES

Null Valued Attributes –

Null value is a value which is not inserted but it does not hold zero value. The attributes which can have a null value called null valued attributes.

Ex: Mobile_no attributes of a person may not be having mobile phones.

TYPES OF ATTRIBUTES

Complex Attributes –

Complex attribute is a combination of composite and multivalued attributes. Complex attributes are represented by { } and composite attributes are represented by ( ).

Ex: Address_phone attribute will hold both the address and phone_no of any person.

{(2-A,St-5,Sec-4,Bhilai), 2398124}

Entity Type & Entity Sets

- Entity Type –
  - An entity type defines a collection of entities that have the same attributes. Each entity type in the database is described by its name and attributes.

    Ex: STUDENT, UNIVERSITY

STUDENT

Name	Age	Rollno

- Entity Set –
  - The collection of all entities of a particular entity type in the database at any point in time is called entity set.

    Ex: Set of all rows

10 rows of STUDENT

TYPES OF ENTITY TYPES

Strong entity type –

Entity types that have at least one key attribute.

Weak entity type –

Entity type that does not have any key attribute.

* An entity in a weak entity type is identified by a relationship with a strong entity type and that relationship is called Identifying Relationship and that strong entity type is called the owner of the weak entity type.

TYPES OF ENTITY TYPES

Student

Roll No.	Name	Age
1	Rakesh	20
2	Nikhil	21
3	Nikhil	21

Secured

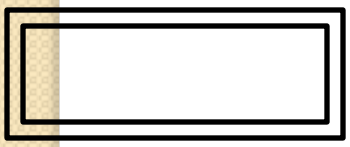
Marks

Name	M1	M2	M3
Nikhil	50	45	40
Nikhil	80	75	82

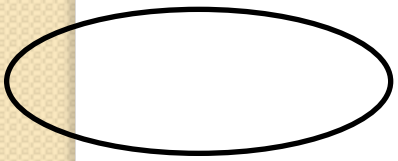
Identifying Relationship

NOTATIONS USED IN E-R DIAGRAM

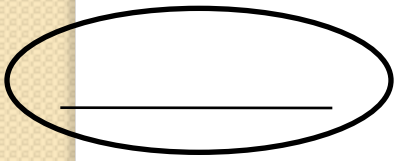
Entity Type



Weak Entity Type

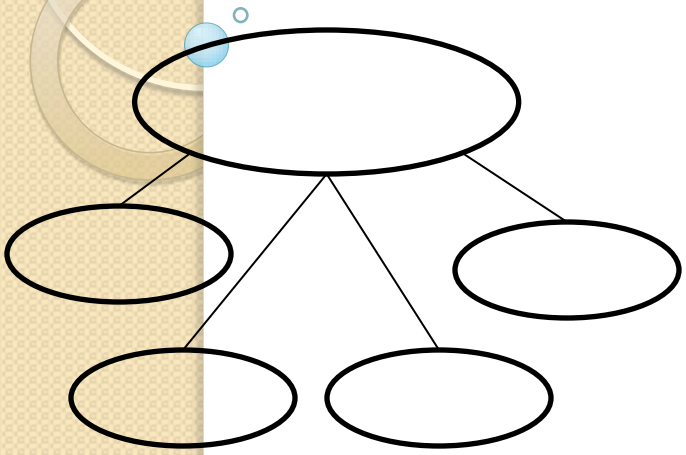


Attribute

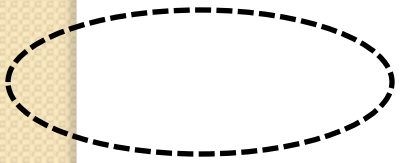


Key Attribute

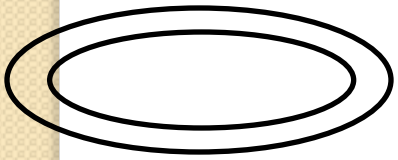
NOTATIONS USED IN E-R DIAGRAM



Composite Attribute

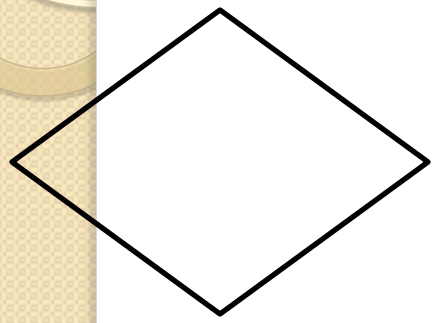


Derived Attribute

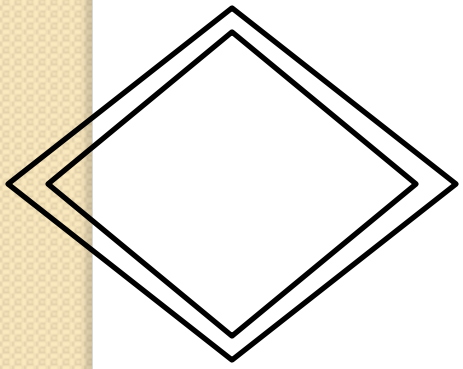


Multivalued Attribute

NOTATIONS USED IN E-R DIAGRAM

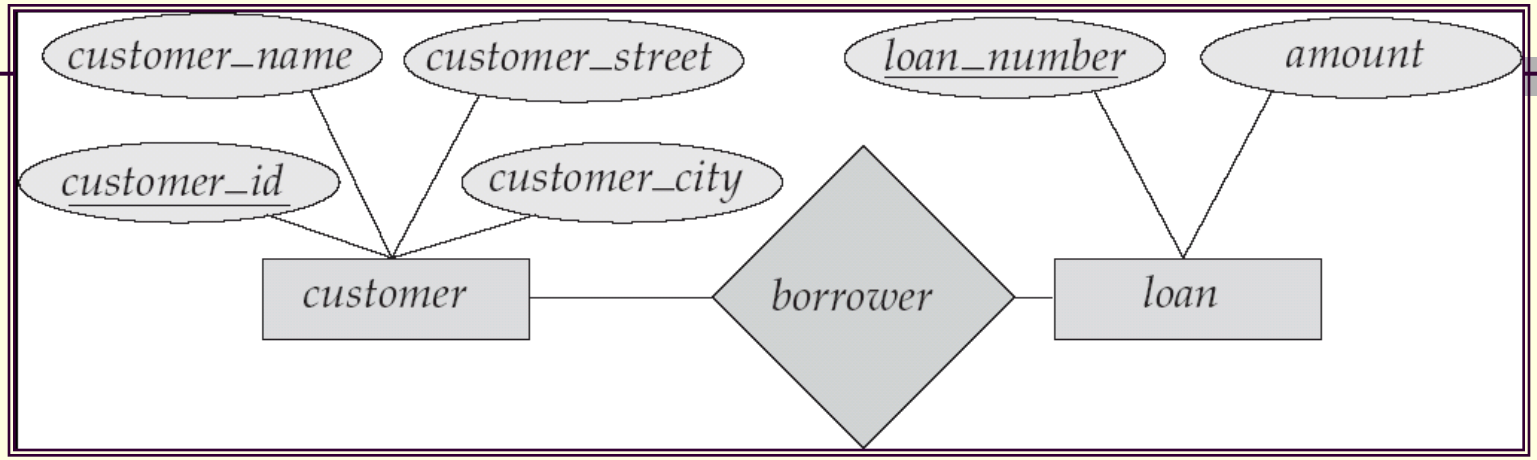


Relationship Type



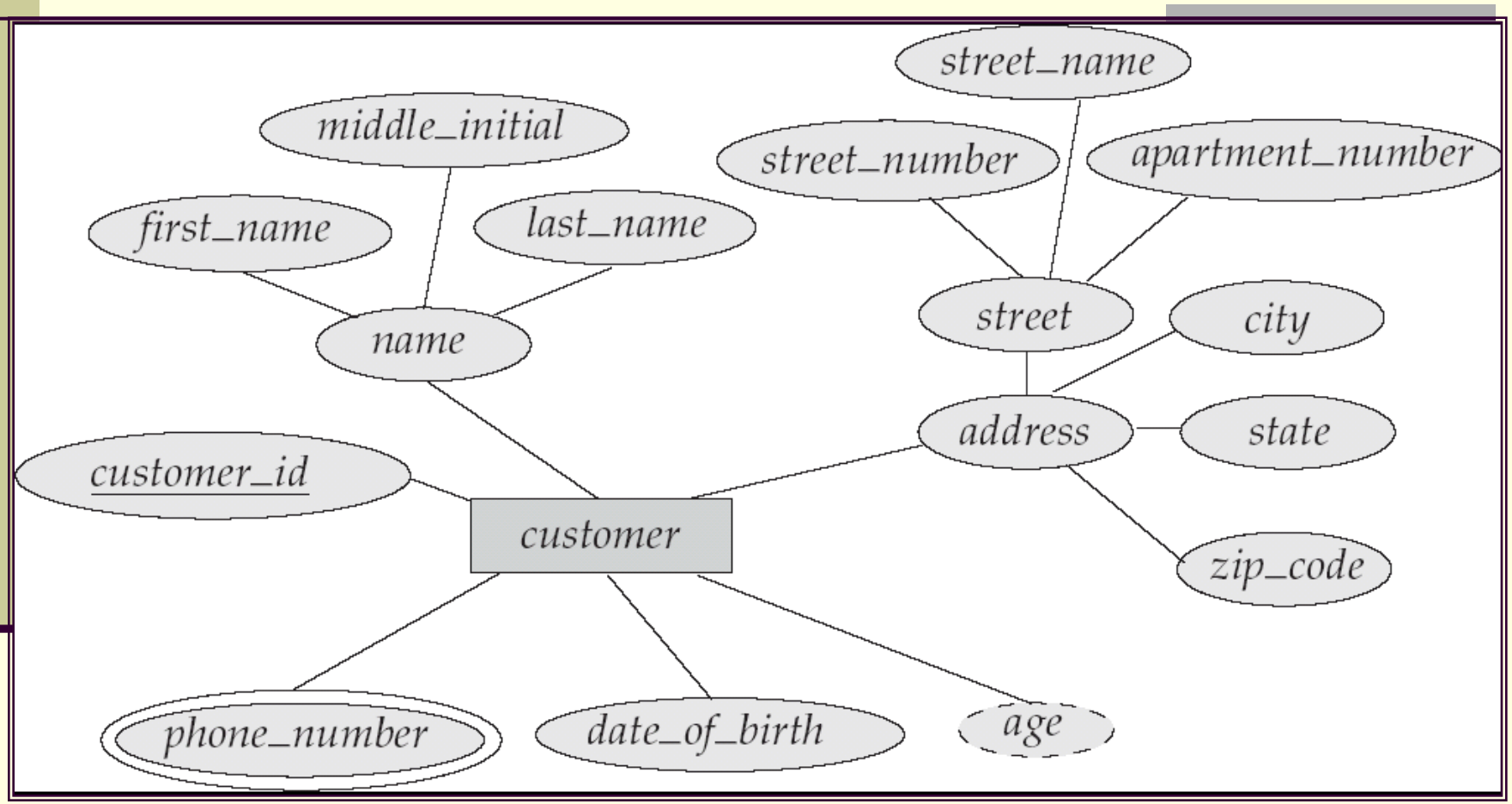
Identifying Relationship

E-R Diagrams

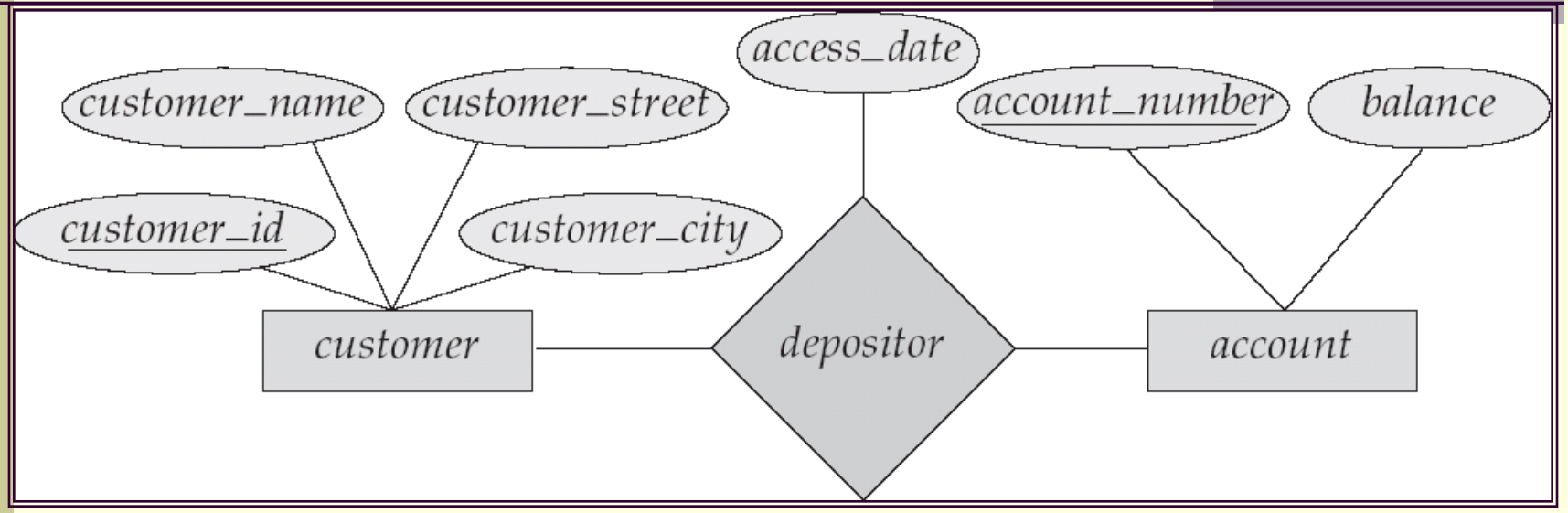


- Rectangles represent entity types.
- Diamonds represent relationship types.
- Lines link attributes to entity types and entity types to relationship types.
- Ellipses represent attributes
  - Double ellipses represent multivalued attributes.
  - Dashed ellipses denote derived attributes.
- Underline indicates primary key attributes (will study later)

E-R Diagram With Composite, Multivalued, and Derived Attributes

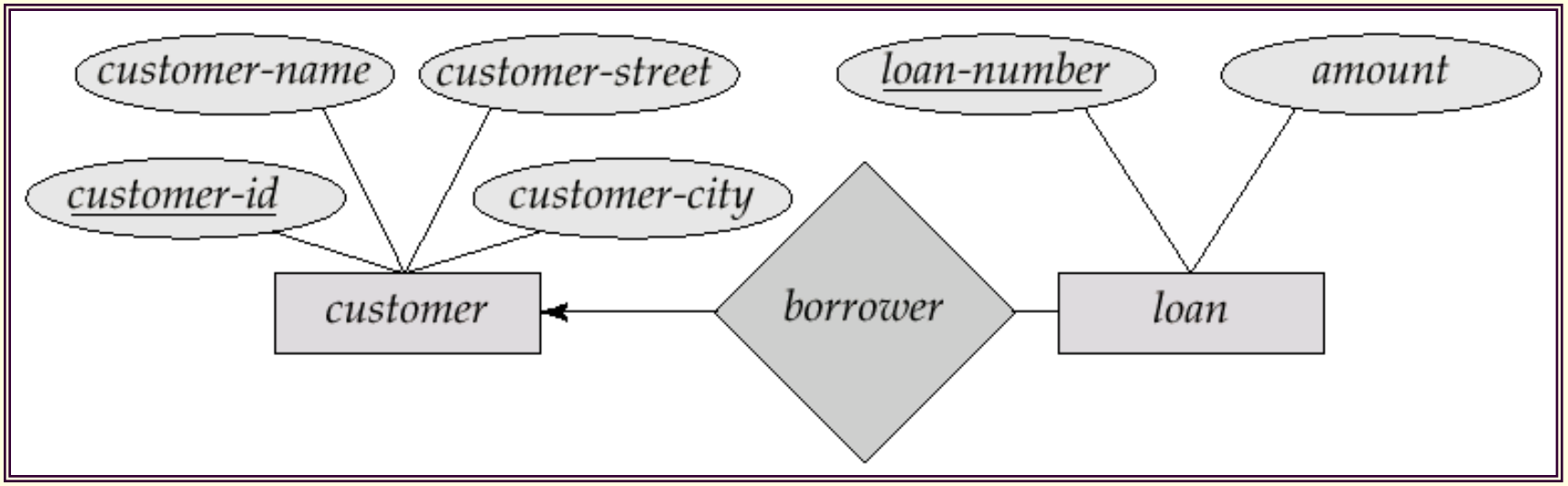


Relationship Types with Attributes



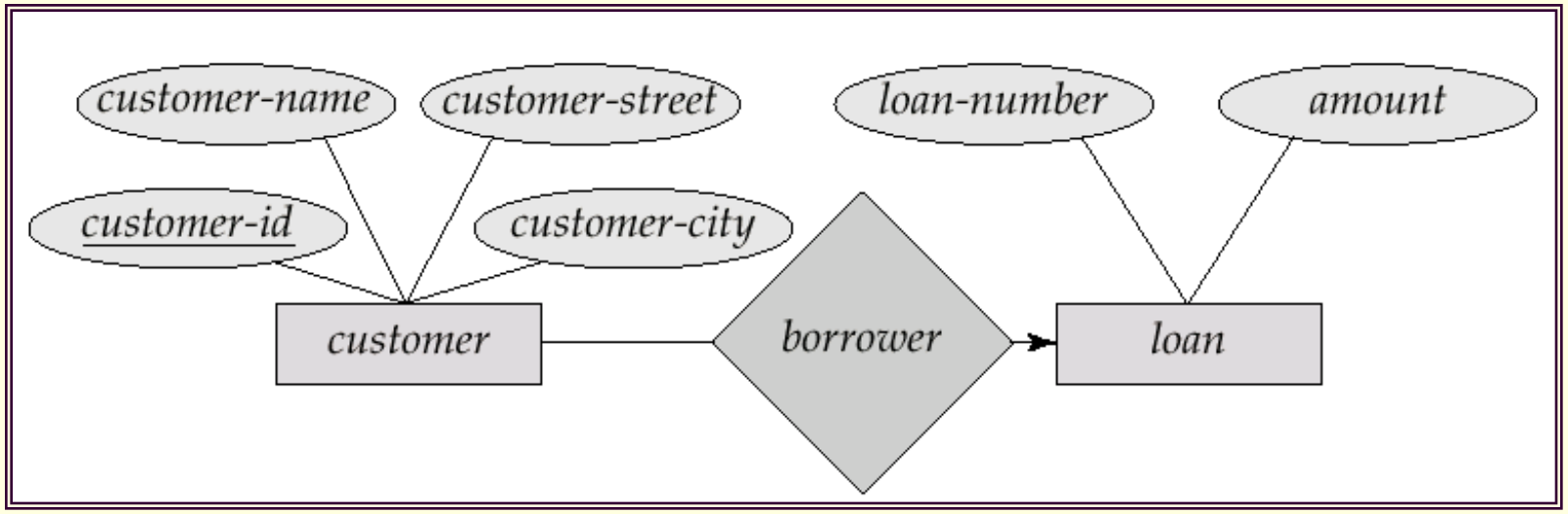
One-To-Many Relationship

- In the one-to-many relationship a loan is associated with at most one customer via *borrower*, a customer is associated with several (including 0) loans via *borrower*



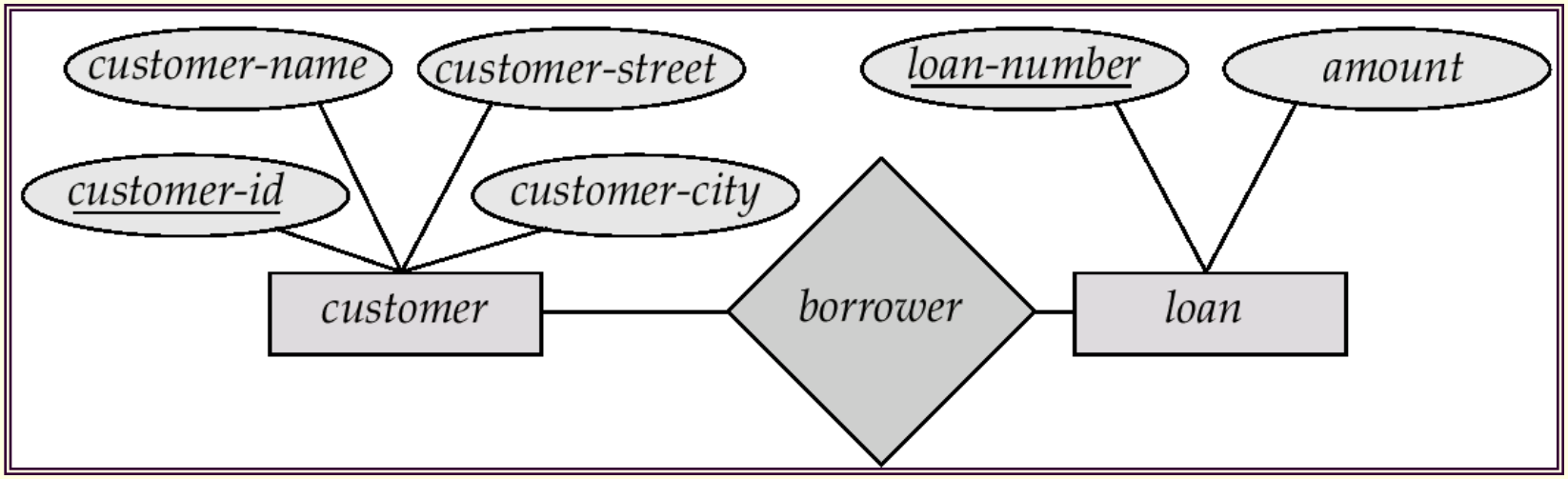
Many-To-One Relationships

- In a many-to-one relationship a loan is associated with several (including 0) customers via *borrower*, a customer is associated with at most one loan via *borrower*



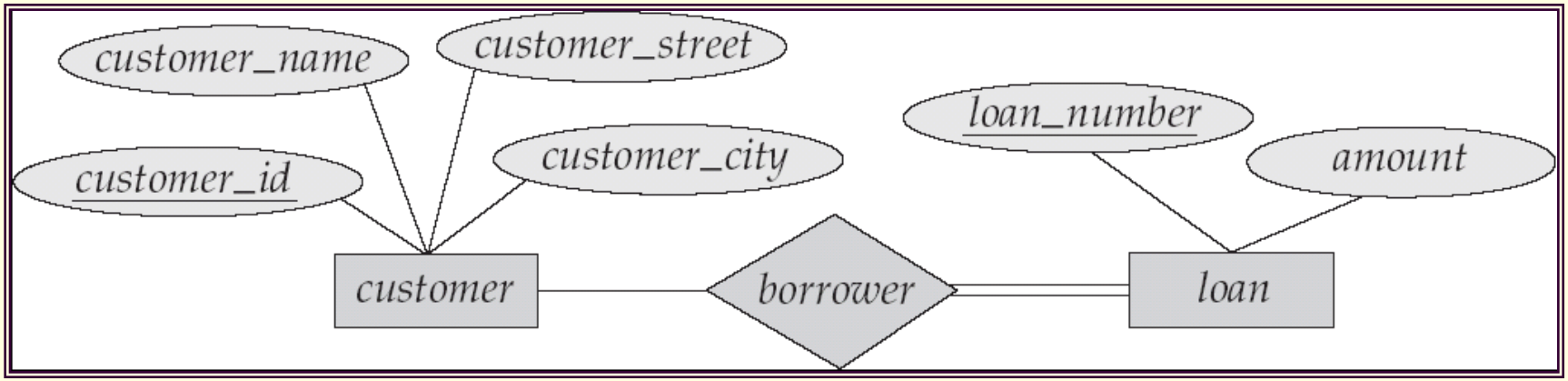
Many-To-Many Relationship

- A customer is associated with several (possibly 0) loans via borrower
- A loan is associated with several (possibly 0) customers via borrower



Participation of an Entity type in a Relationship type

- Total participation (indicated by double line): every entity in the entity type participates in at least one relationship in the relationship type
  - E.g. participation of loan in borrower is total
    - every loan must have a customer associated to it via borrower
- Partial participation: some entities may not participate in any relationship in the relationship type
  - Example: participation of customer in borrower is partial

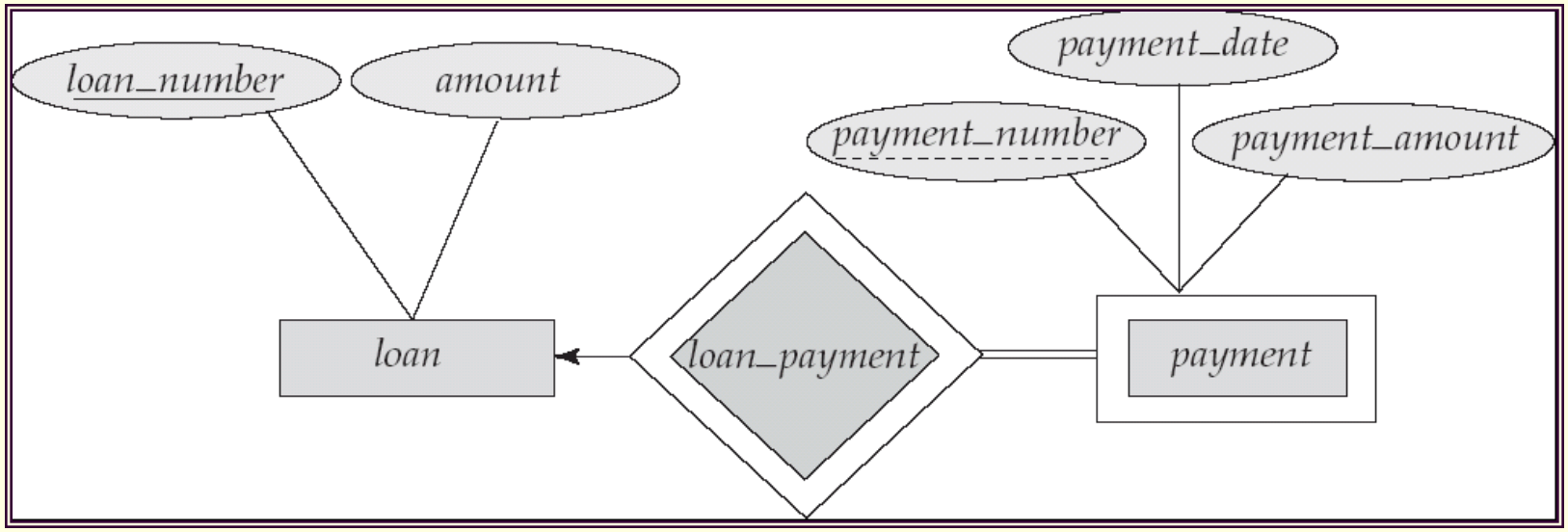


Weak Entity Types

- An entity type that does not have a primary key is referred to as a **weak entity type**.
- The existence of a weak entity type depends on the existence of a **identifying entity type**
  - it must relate to the identifying entity type via a total, one-to-many relationship type from the identifying to the weak entity type
  - Identifying relationship depicted using a double diamond
- The **discriminator** (*or partial key)* of a weak entity type is the type of attributes that distinguishes among all the entities of a weak entity type.
- The primary key of a weak entity type is formed by the primary key of the strong entity type on which the weak entity type is existence dependent, plus the weak entity type's discriminator.

Weak Entity types (Cont.)

- We depict a weak entity type by double rectangles.
- We underline the discriminator of a weak entity type with a dashed line.
- payment_number – discriminator of the *payment* entity type
- Primary key for *payment* – (*loan_number*, *payment_number*)

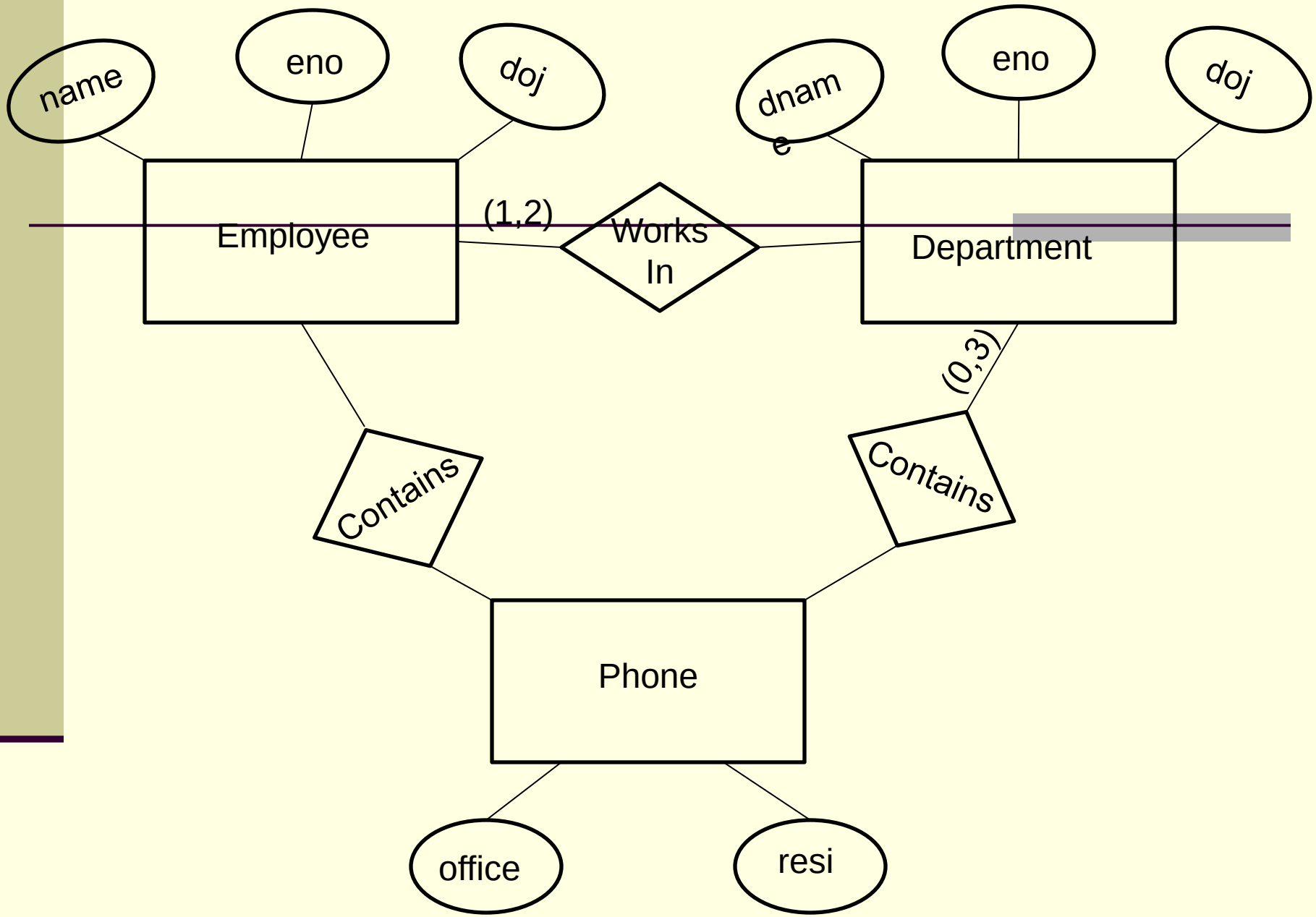


Weak Entity types (Cont.)

- Note: the primary key of the strong entity type is not explicitly stored with the weak entity type, since it is implicit in the identifying relationship.
- If *loan_number* were explicitly stored, *payment* could be made a strong entity, but then the relationship between *payment* and *loan* would be duplicated by an implicit relationship defined by the attribute *loan_number* common to *payment* and *loan*

Qu.

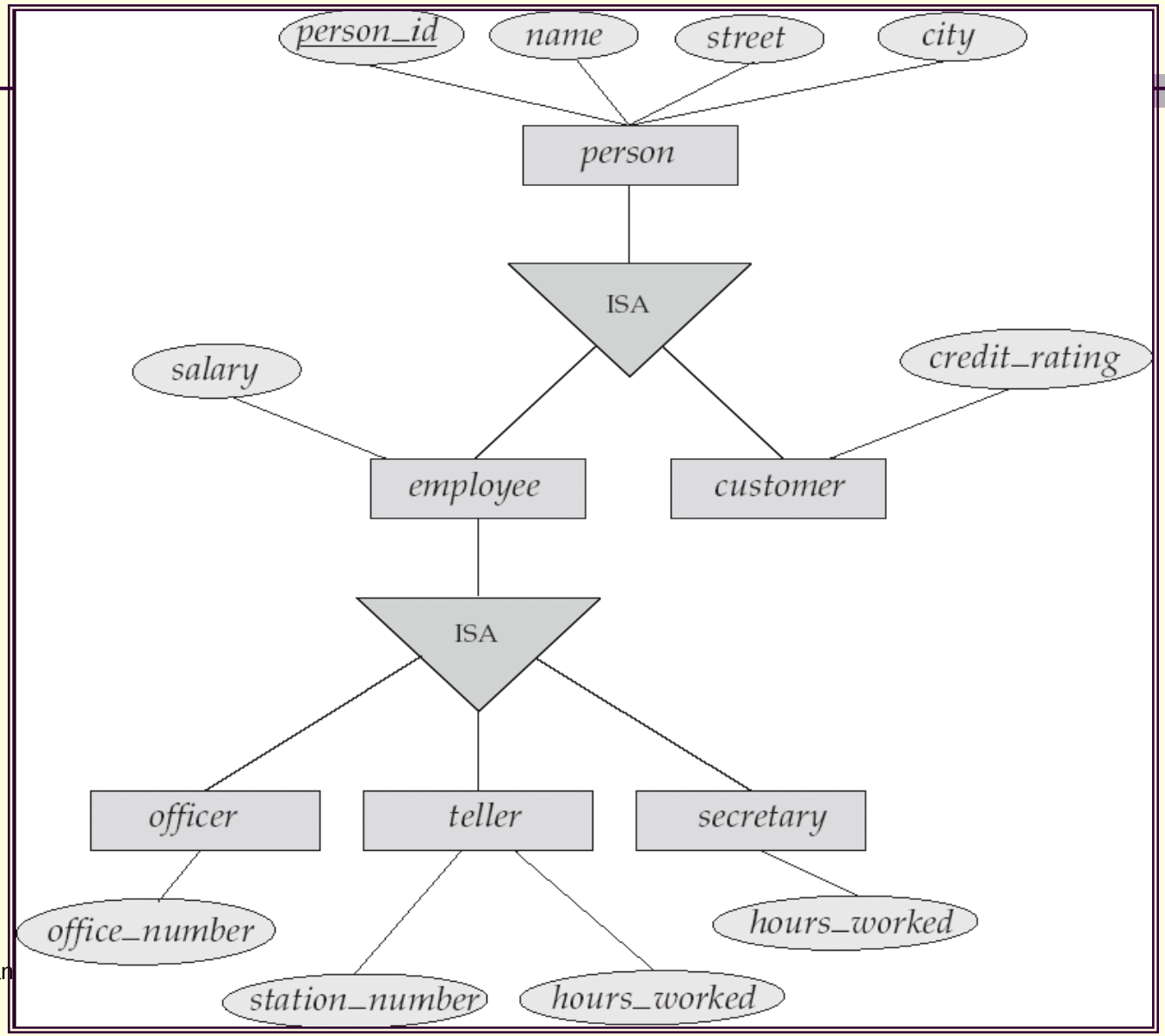
An employee works in one some department. The department contains phone, the employee also has phone. Assume that an employee works in maximum 2 departments or minimum one department. Each department must have maximum 3 phones or minimum zero phone. Design an E-R diagram for the above.


name
eno
doj
dnam
e
eno
doj
Employee
(1,2)
Works
In
Department
(0,3)
Contains
Contains
Phone
office
resi

Extended E-R Features: Specialization

- Top-down design process; we designate sub-groupings within an entity set that are distinctive from other entities in the set.
- These subgroupings become lower-level entity sets that have attributes or participate in relationships that do not apply to the higher-level entity set.
- Depicted by a *triangle* component labeled ISA (E.g. *customer* “is a” *person*).
- **Attribute inheritance** – a lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set to which it is linked.

Specialization Example



Extended ER Features: Generalization

- **A bottom-up design process** – combine a number of entity sets that share the same features into a higher-level entity set.
- Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way.
- The terms specialization and generalization are used interchangeably.

Specialization and Generalization (Cont.)

- Can have multiple specializations of an entity set based on different features.
- E.g. *permanent_employee* vs. *temporary_employee*, in addition to *officer* vs. *secretary* vs. *teller*
- Each particular employee would be
  - a member of one of *permanent_employee* or *temporary_employee*,
  - and also a member of one of *officer*, *secretary*, or *teller*
- The ISA relationship also referred to as **superclass - subclass** relationship

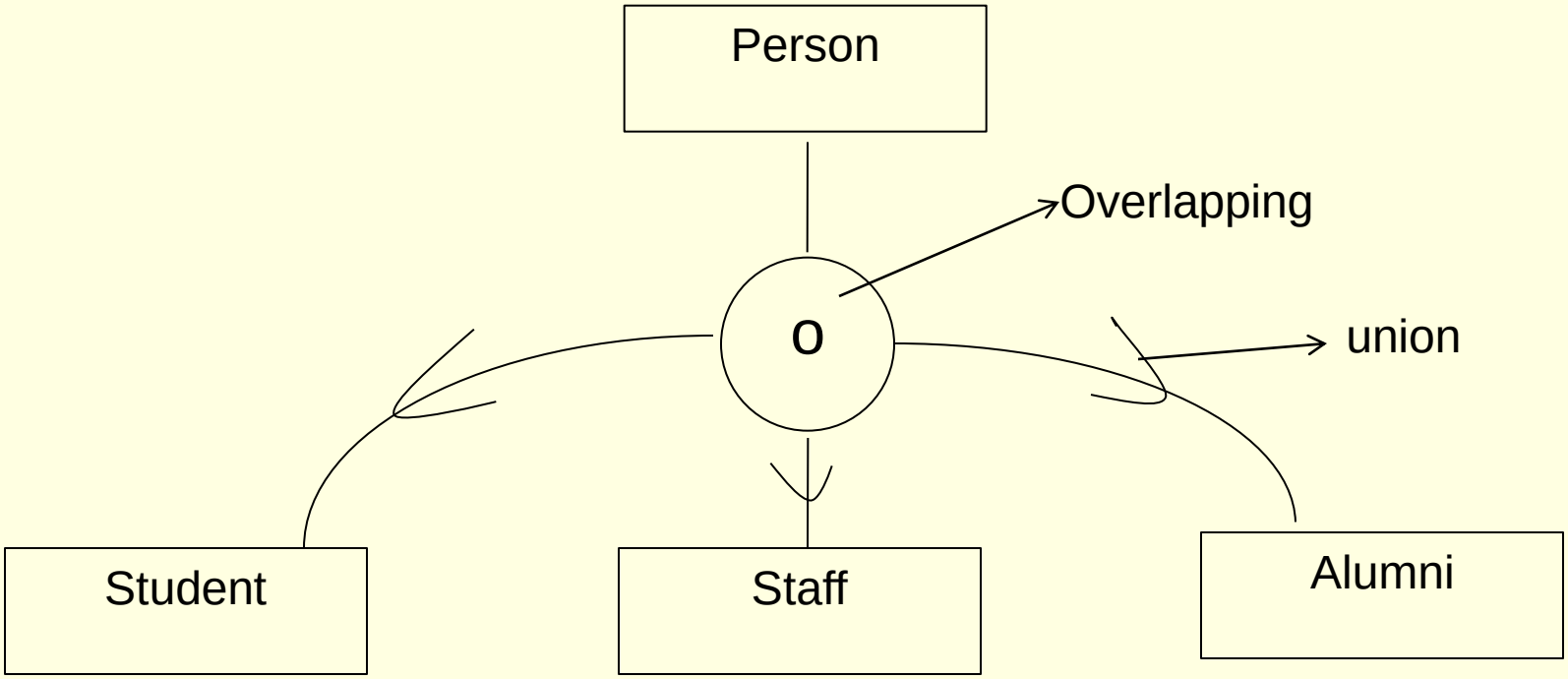
Design Constraints on a Specialization/Generalization

- Constraint on which entities can be members of a given lower-level entity set.
  - **Condition-defined**
    - Example: all customers over 65 years are members of *senior-citizen* entity set; *senior-citizen* ISA *person*.
  - **User-defined**
- Constraint on whether or not entities may belong to more than one lower-level entity set within a single generalization.
  - **Disjoint**
    - an entity can belong to only one lower-level entity set
    - Noted in E-R diagram by writing *disjoint* next to the ISA triangle
  - **Overlapping**
    - an entity can belong to more than one lower-level entity set

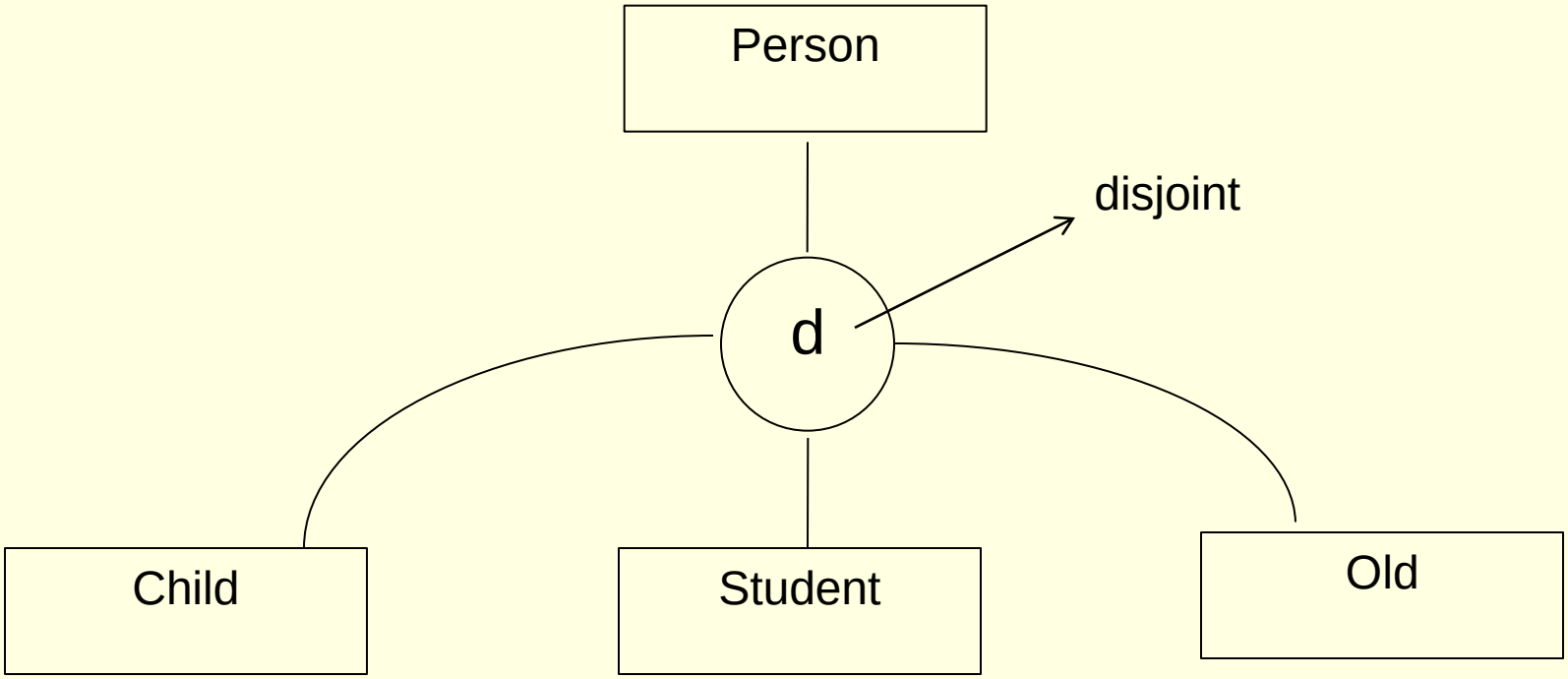
Design Constraints on a Specialization/Generalization (Cont.)

- **Completeness constraint** -- specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within a generalization.
  - **total** : an entity must belong to one of the lower-level entity sets
  - **partial**: an entity need not belong to one of the lower-level entity sets

Example of Disjoint & Overlapping Constraint

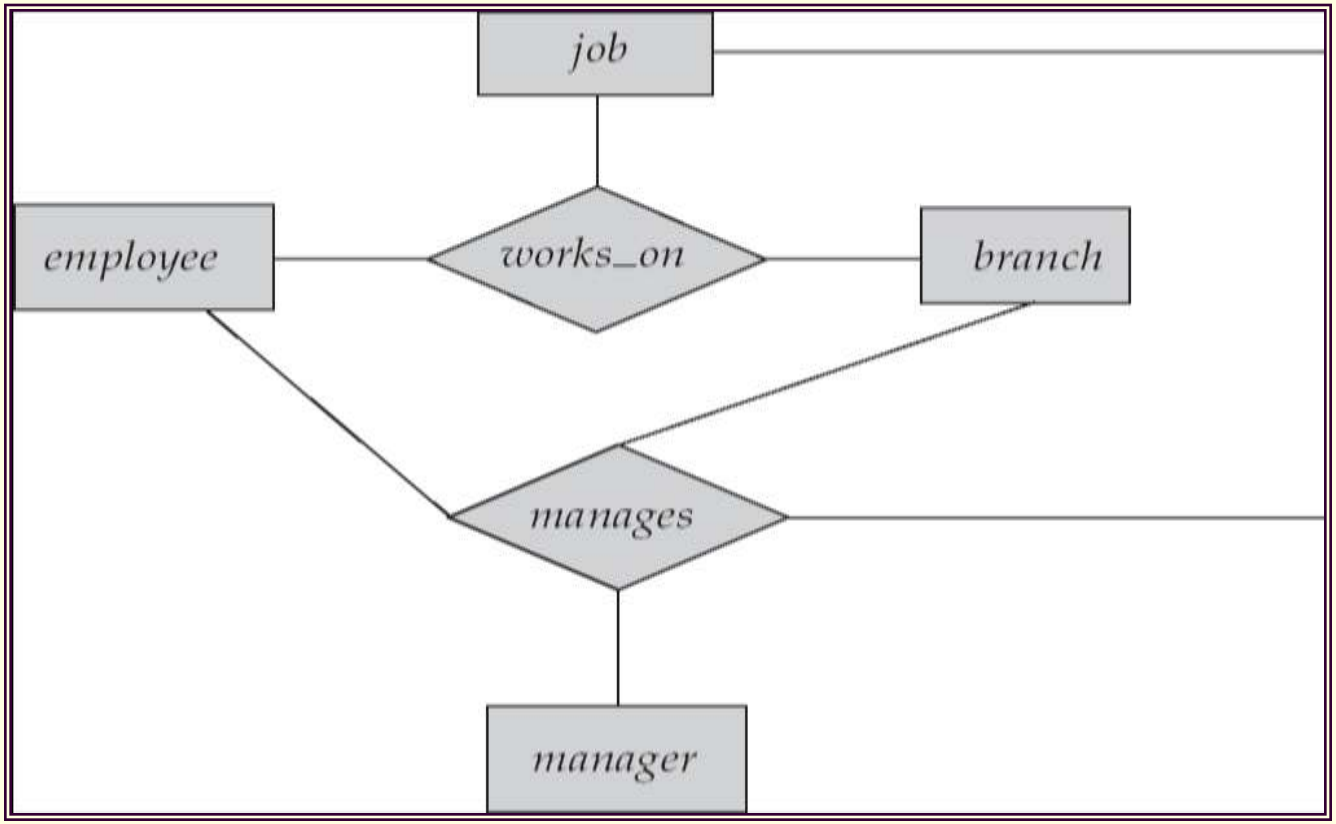


Example of Disjoint & Overlapping Constraint



Aggregation

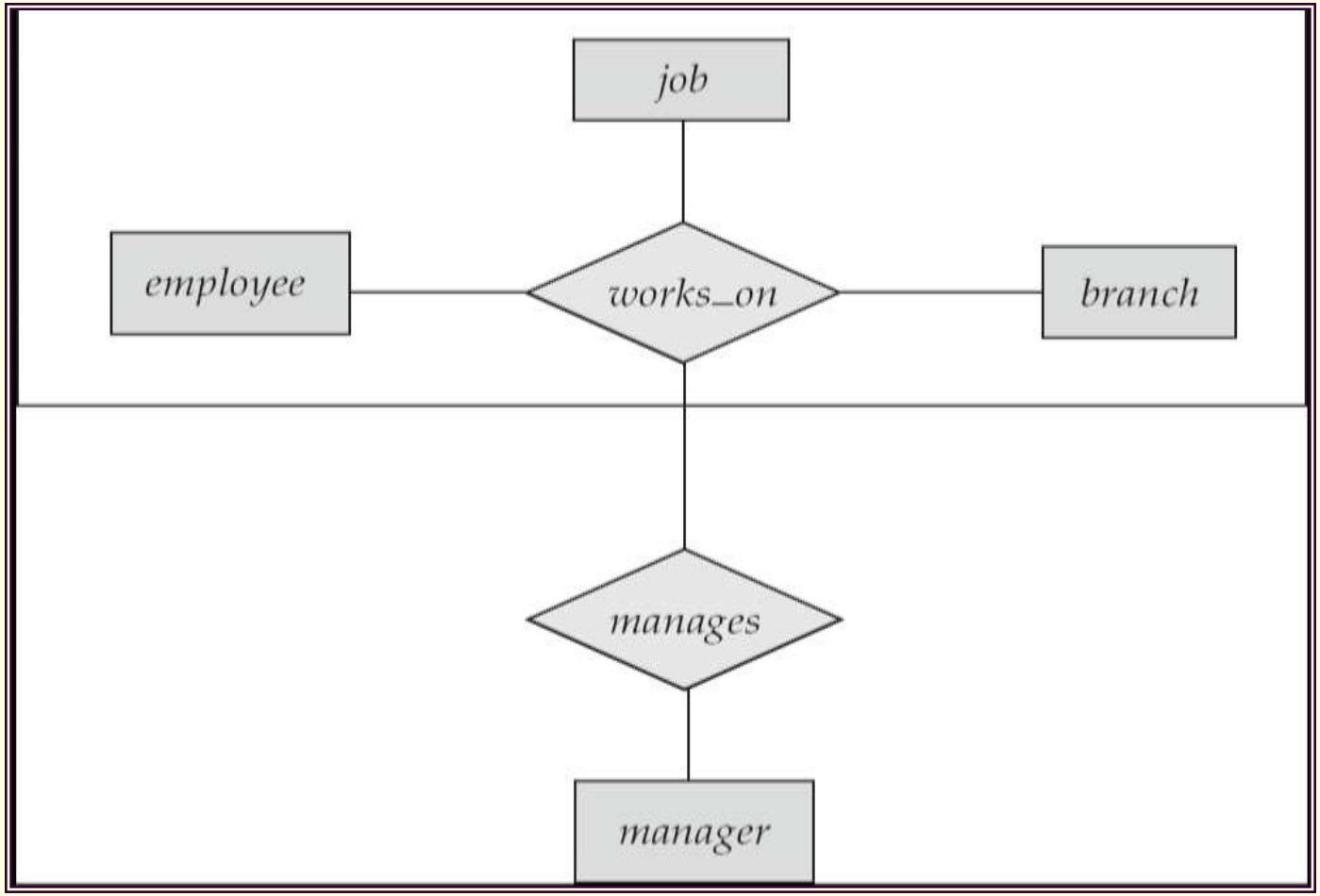
- Consider the ternary relationship *works_on.*
- Suppose we want to record managers for tasks performed by an employee at a branch

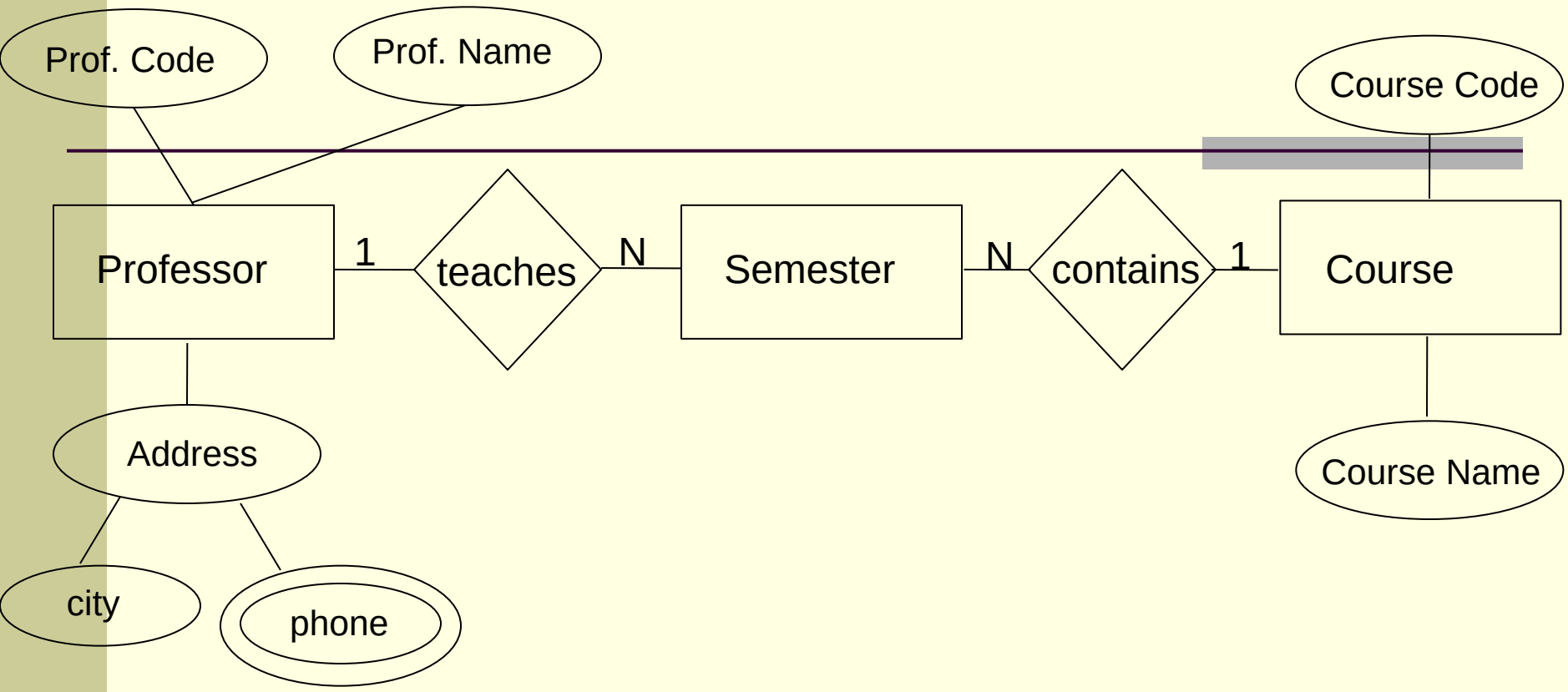


Aggregation (Cont.)

- Relationship sets *works_on* and *manages* represent overlapping information
  - Every *manages* relationship corresponds to a *works_on* relationship
  - However, some *works_on* relationships may not correspond to any *manages* relationships
    - So we can't discard the *works_on* relationship
- Eliminate this redundancy via *aggregation*
  - Treat relationship as an abstract entity
  - Allows relationships between relationships
  - Abstraction of relationship into new entity
- Without introducing redundancy, the following diagram represents:
  - An employee works on a particular job at a particular branch
  - An employee, branch, job combination may have an associated manager

E-R Diagram With Aggregation

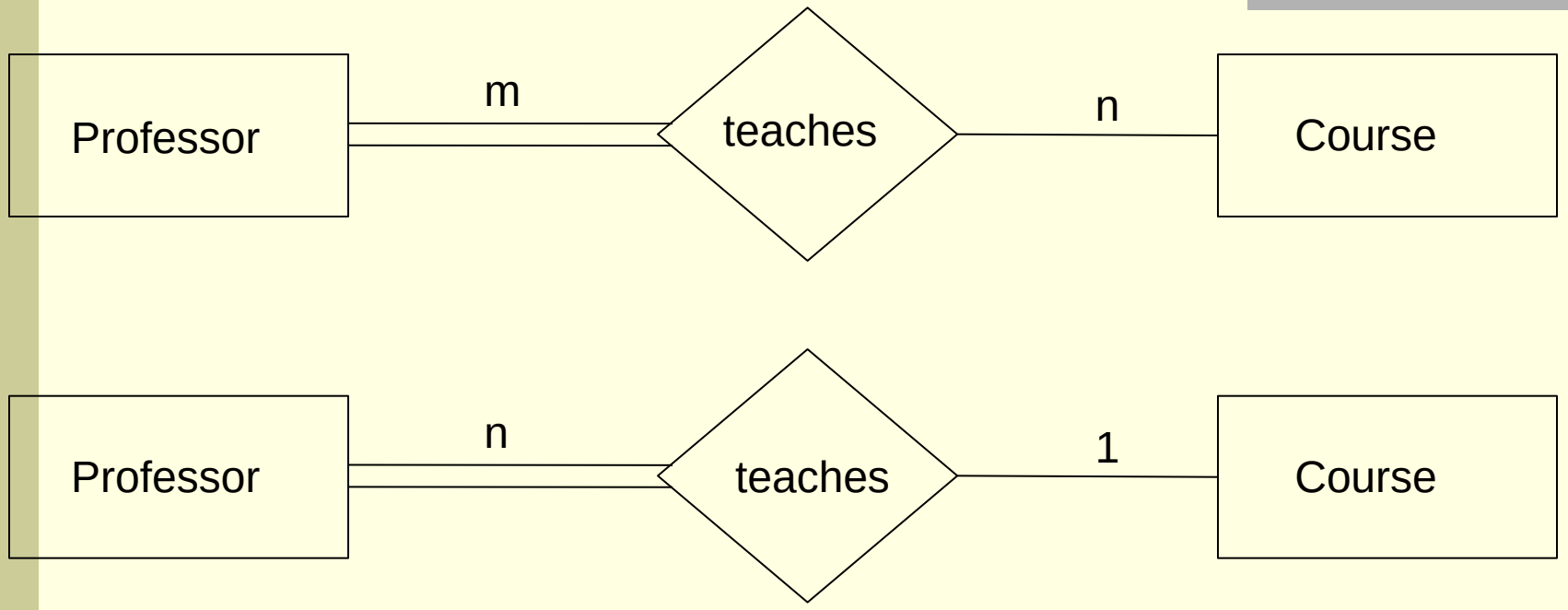




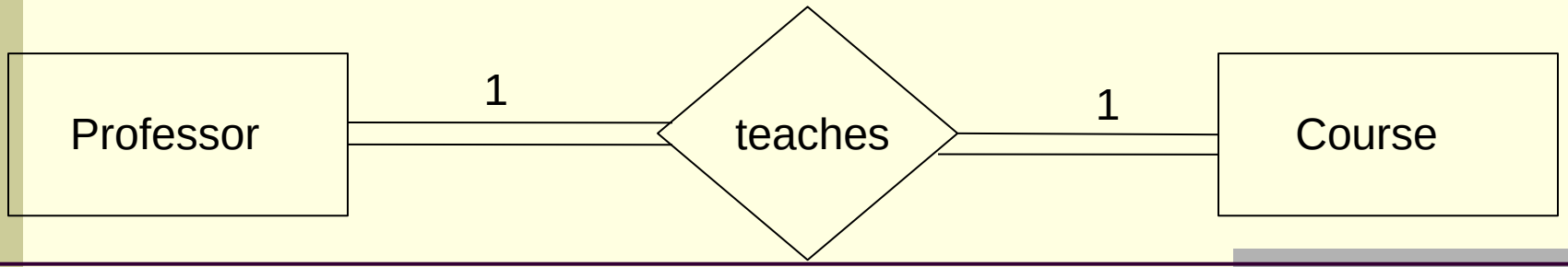
1. The Professor can teach the same course in several semesters and each offering must be recorded

2. Adding one more attribute in the ‘Professor’ entity named year/session

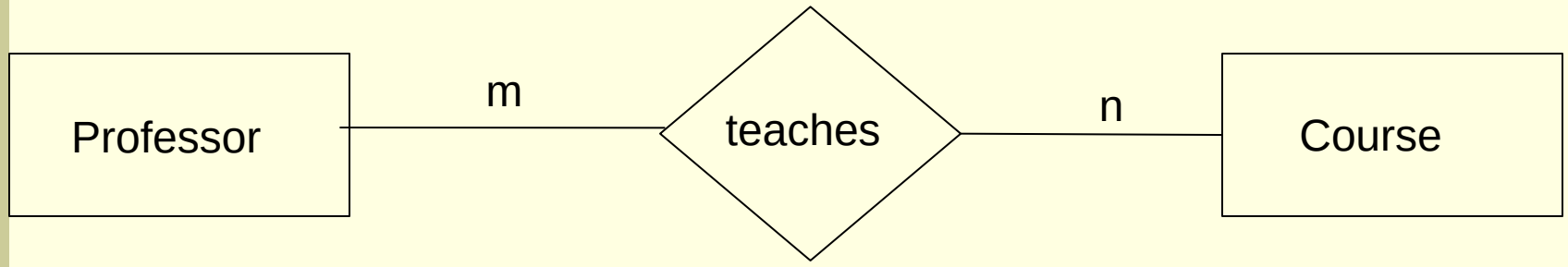
3. Every professor must teach some course



4. Every professor teaches exactly one course only.



5. Every professor teaches exactly one course and each course must be taught by a professor.



6. A team of professors can teach certain courses jointly but it is possible that no one professor can teach the whole course

Buffer Manager

- Programs call on the buffer manager when they need a block from disk.
  - If the block is already in the buffer, buffer manager returns the address of the block in main memory
  - If the block is not in the buffer, the buffer manager
    - Allocates space in the buffer for the block
      - Replacing (throwing out) some other block, if required, to make space for the new block.
      - Replaced block written back to disk only if it was modified since the most recent time that it was written to/fetched from the disk.
    - Reads the block from the disk to the buffer, and returns the address of the block in main memory to requester.

Buffer-Replacement Policies

- Most operating systems replace the block **least recently used** (LRU strategy)
- Idea behind LRU – use past pattern of block references as a predictor of future references
- Queries have well-defined access patterns (such as sequential scans), and a database system can use the information in a user's query to predict future references
  - LRU can be a bad strategy for certain access patterns involving repeated scans of data
    - For example: when computing the join of 2 relations r and s by a nested loops
      for each tuple *tr* of *r* do
        for each tuple *ts* of *s* do
          if the tuples *tr* and *ts* match …
  - Mixed strategy with hints on replacement strategy provided by the query optimizer is preferable

Buffer-Replacement Policies (Cont.)

- **Pinned block** – memory block that is not allowed to be written back to disk.
- **Toss-immediate** strategy – frees the space occupied by a block as soon as the final tuple of that block has been processed
- Most recently used (MRU) strategy – system must pin the block currently being processed. After the final tuple of that block has been processed, the block is unpinned, and it becomes the most recently used block.
- Buffer manager can use statistical information regarding the probability that a request will reference a particular relation
  - E.g., the data dictionary is frequently accessed. Heuristic: keep data-dictionary blocks in main memory buffer
- Buffer managers also support forced output of blocks for the purpose of recovery.

File Organization

- The database is stored as a collection of *files*. Each file is a sequence of *records.* A record is a sequence of fields.
- One approach:
  - assume record size is fixed
  - each file has records of one particular type only
  - different files are used for different relations

  This case is easiest to implement; will consider variable length records later.

Fixed-Length Records

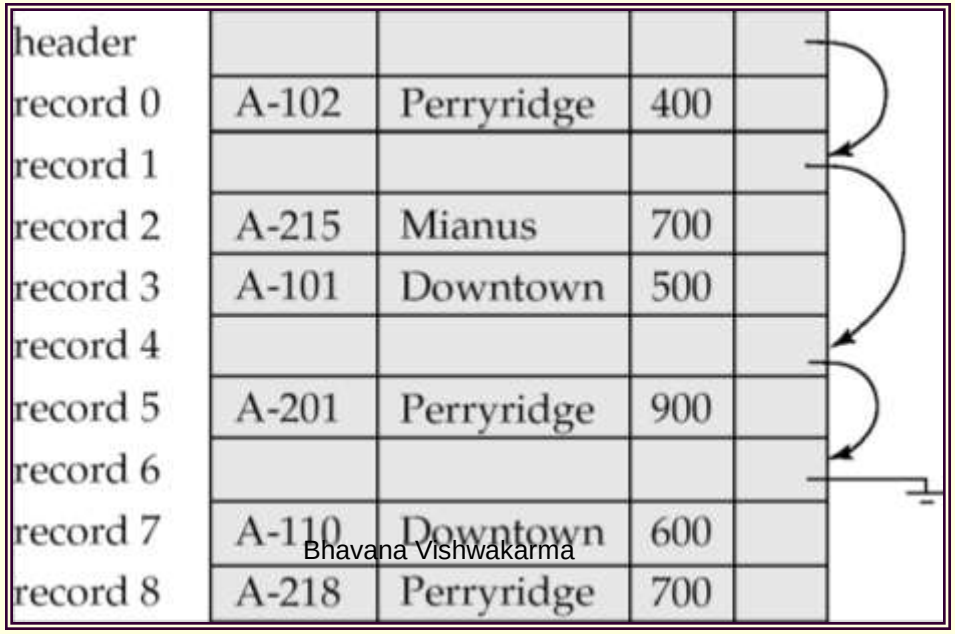
- Simple approach:
  - Store record *i* starting from byte $n * (i - 1)$, where *n* is the size of each record.
  - Record access is simple but records may cross blocks
    - Modification: do not allow records to cross block boundaries

- Deletion of record *i:* alternatives*:*
  - move records $i + 1, \ldots, n$ to $i, \ldots, n - 1$
  - move record *n* to *i*
  - do not move records, but link all free records on a *free list*

record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

Free Lists

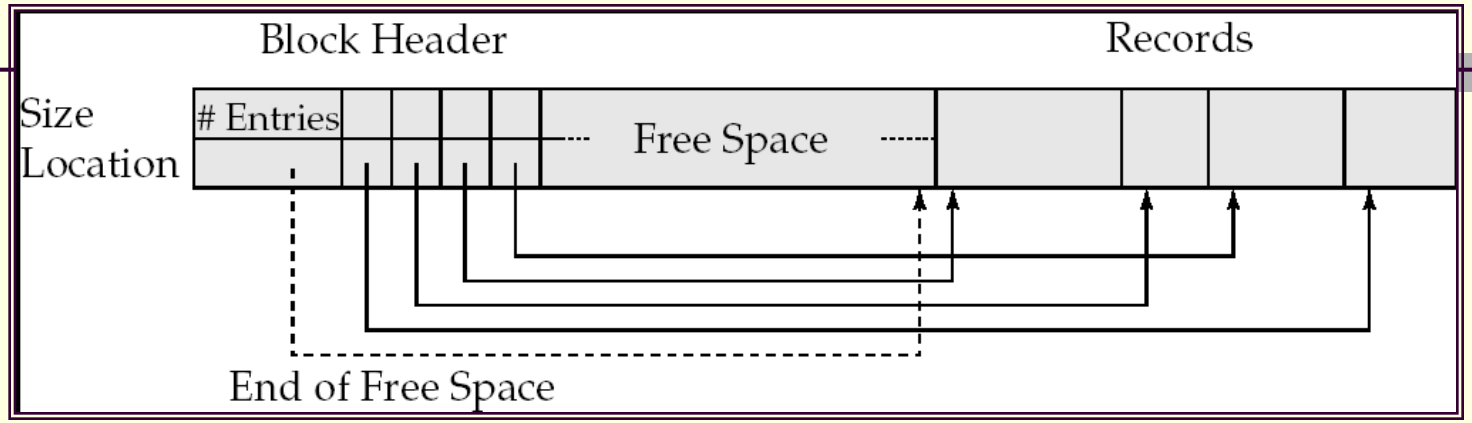
- Store the address of the first deleted record in the file header.
- Use this first record to store the address of the second deleted record, and so on
- Can think of these stored addresses as pointers since they "point" to the location of a record.
- More space efficient representation: reuse space for normal attributes of free records to store pointers. (No pointers stored in in-use records.)



Variable-Length Records

- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file.
  - Record types that allow variable lengths for one or more fields.
  - Record types that allow repeating fields (used in some older data models).

Variable-Length Records: Slotted Page Structure



- Slotted page header contains:
  - number of record entries
  - end of free space in the block
  - location and size of each record
- Records can be moved around within a page to keep them contiguous with no empty space between them; entry in the header must be updated.
- Pointers should not point directly to record — instead they should point to the entry for the record in header.

Organization of Records in Files

- **Heap** – a record can be placed anywhere in the file where there is space
- **Sequential** – store records in sequential order, based on the value of the search key of each record
- **Hashing** – a hash function computed on some attribute of each record; the result specifies in which block of the file the record should be placed
- Records of each relation may be stored in a separate file. In a **multitable clustering file organization** records of several different relations can be stored in the same file
  - Motivation: store related records on the same block to minimize I/O

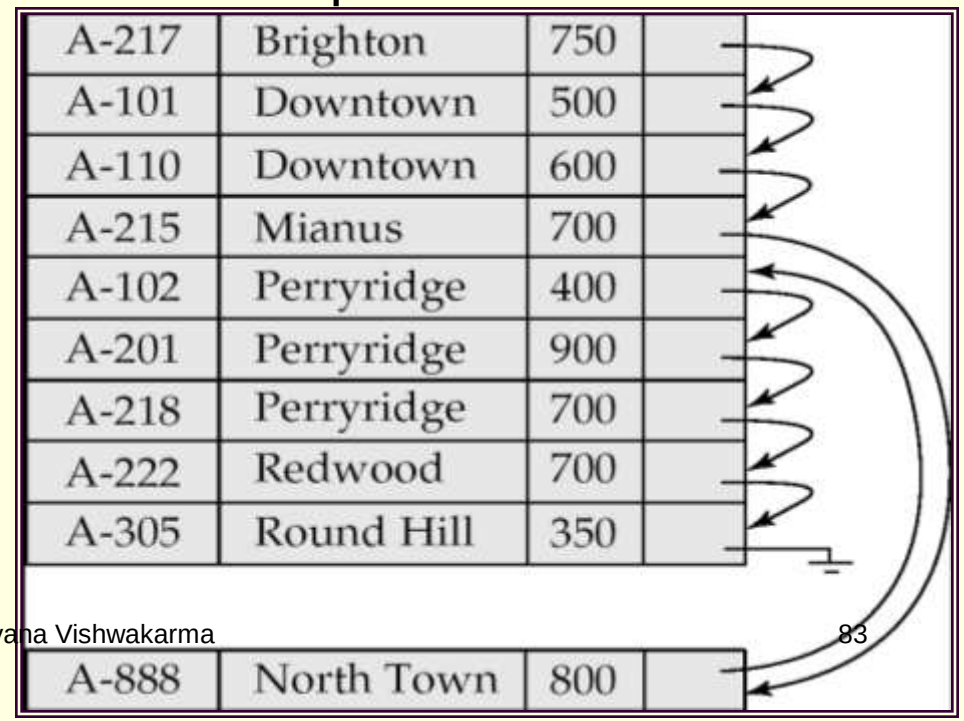
Sequential File Organization

- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key

A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	

Sequential File Organization (Cont.)

- Deletion – use pointer chains
- Insertion –locate the position where the record is to be inserted
  - if there is free space insert there
  - if no free space, insert the record in an overflow block
  - In either case, pointer chain must be updated
- Need to reorganize the file from time to time to restore sequential order



Multitable Clustering File Organization

Store several relations in one file using a **multitable clustering** file organization

customer_name	*account_number*
Hayes	A-102
Hayes	A-220
Hayes	A-503
Turner	A-305

customer_name	*customer_street*	*customer_city*
Hayes	Main	Brooklyn
Turner	Putnam	Stamford

Multitable Clustering File Organization (cont.)

Multitable clustering organization of *customer* and *depositor:*

Hayes	Main	Brooklyn
Hayes	A-102	
Hayes	A-220	
Hayes	A-503	
Turner	Putnam	Stamford
Turner	A-305	

- good for queries involving *depositor* $\bowtie$ *customer*, and for queries involving one single customer and his accounts
- bad for queries involving only customer
- results in variable size records
- Can add pointer chains to link records of a particular relation

Data Dictionary Storage

Data dictionary (also called system catalog) stores metadata; that is, data about data, such as

- Information about relations
  - names of relations
  - names and types of attributes of each relation
  - names and definitions of views
  - integrity constraints
- User and accounting information, including passwords
- Statistical and descriptive data
  - number of tuples in each relation
- Physical file organization information
  - How relation is stored (sequential/hash/…)
  - Physical location of relation
- Information about indices.

Data Dictionary Storage (Cont.)

- Catalog structure
  - Relational representation on disk
  - specialized data structures designed for efficient access, in memory
- A possible catalog representation:

Relation_metadata = (*relation_name, number_of_attributes, storage_organization, location*)

Attribute_metadata = (*attribute_name, relation_name, domain_type, position, length*)

User_metadata = (*user_name, encrypted_password, group*)

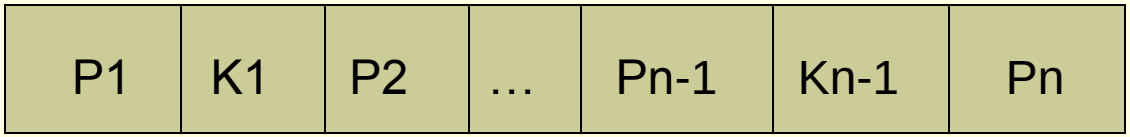
Index_metadata = (*index_name, relation_name, index_type, index_attributes*)

View_metadata = (*view_name, definition*)

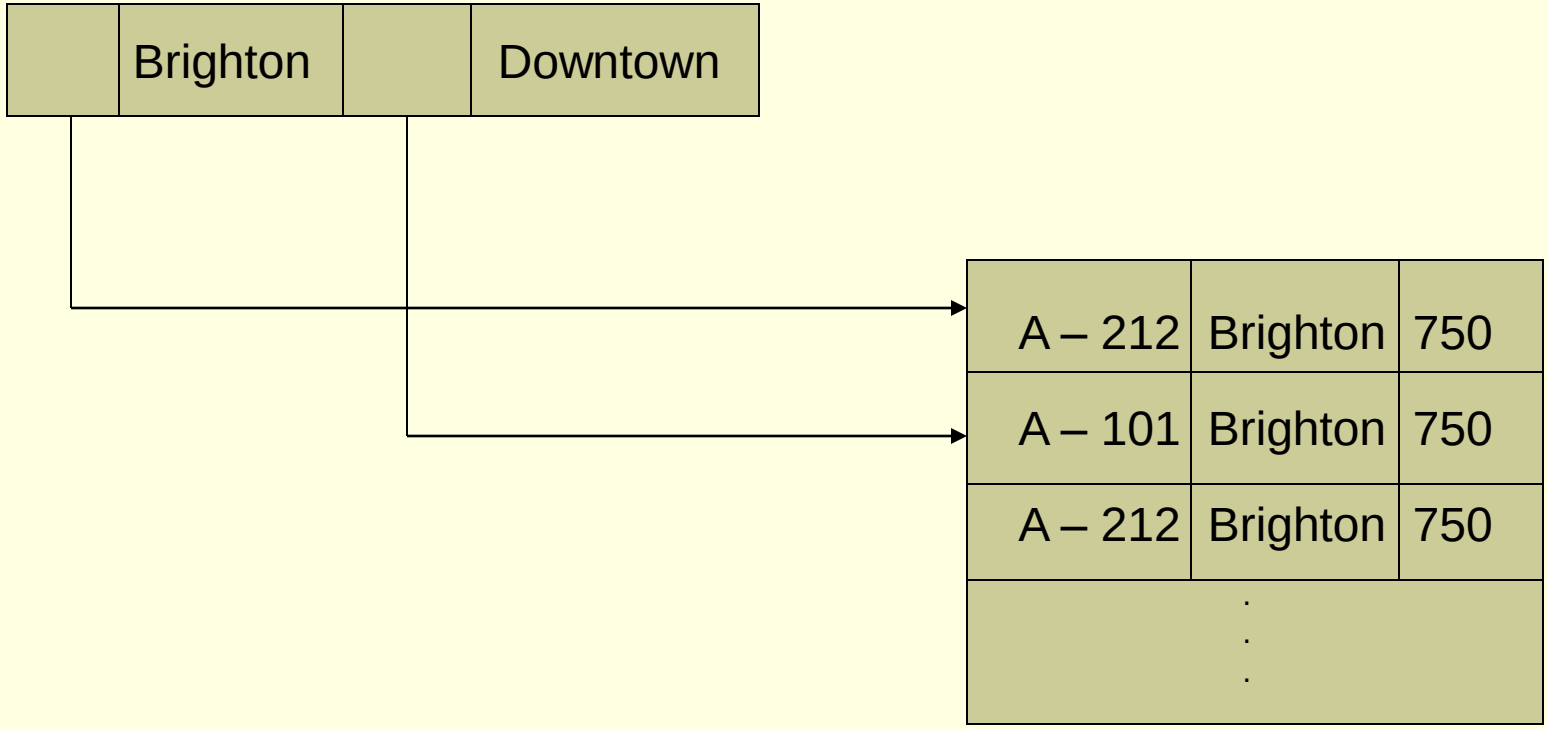
B+ - Tree Structure

- A B+ - Tree is in the form of a balanced tree in which every path from the root of the tree to a leaf of the tree is the same length.
- Each non-leaf node in the tree has between [n/2] and n children, where n is fixed.
- B+ - Trees are good for searches, but cause some overhead issues in wasted space.

- A typical node contains up to n – 1 search key values K1, K2,…, Kn-1, and n pointers P1, P2,…, Pn. The search key values are kept in sorted order.

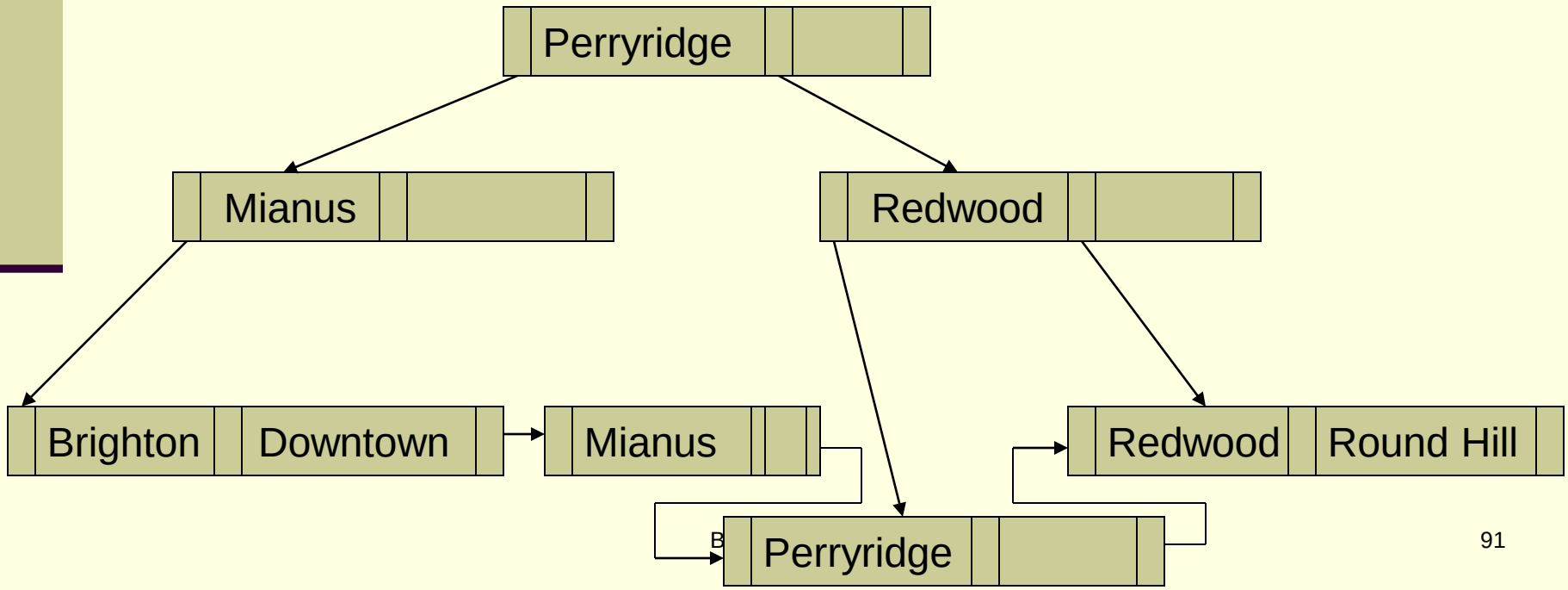


- The pointer Pi can point to either a file record or a bucket of pointers which each point to a file record.
- leaf node, n = 3



- Each leaf can hold up to n – 1 values and must contain at least [(n – 1) / 2] values.
- Nonleaf node pointers point to tree nodes (leaf nodes). Nonleaf nodes can hold up to n pointers and must hold at least [n/2] pointers.

  i.e. n = 3

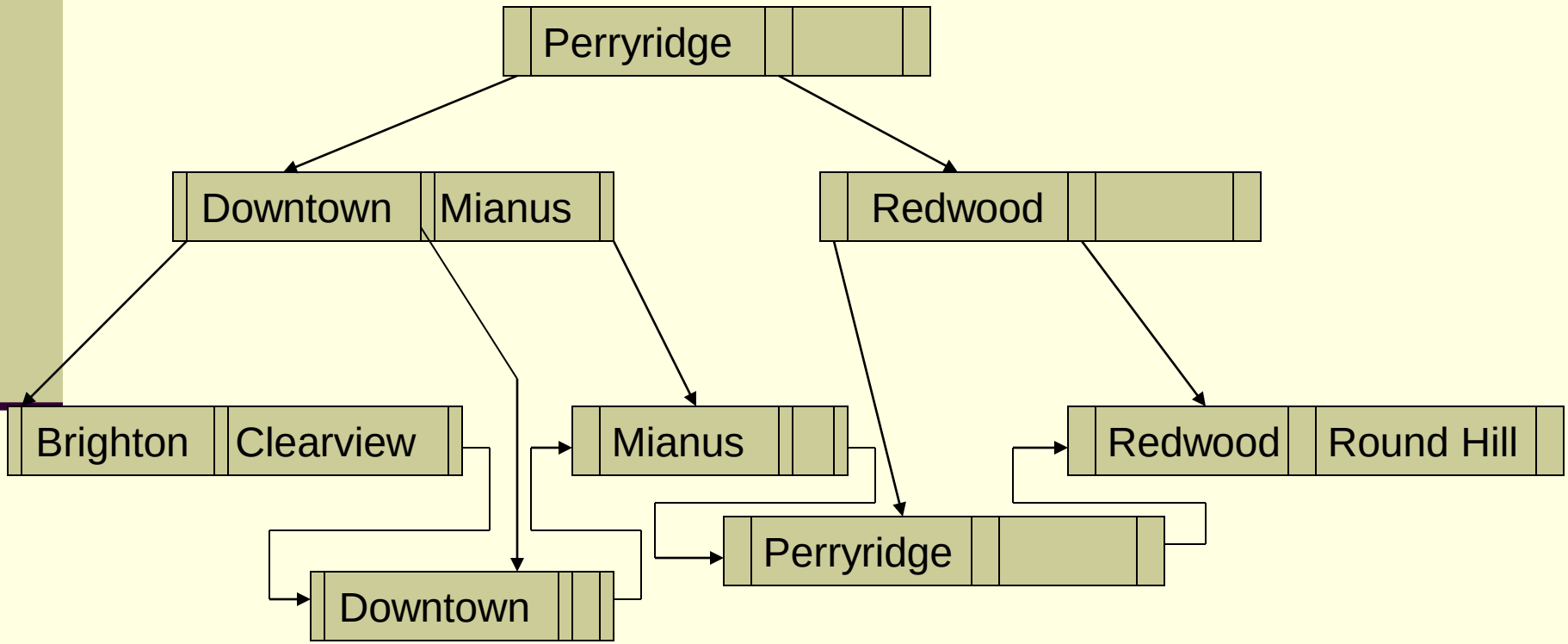


B+ - Tree Updates

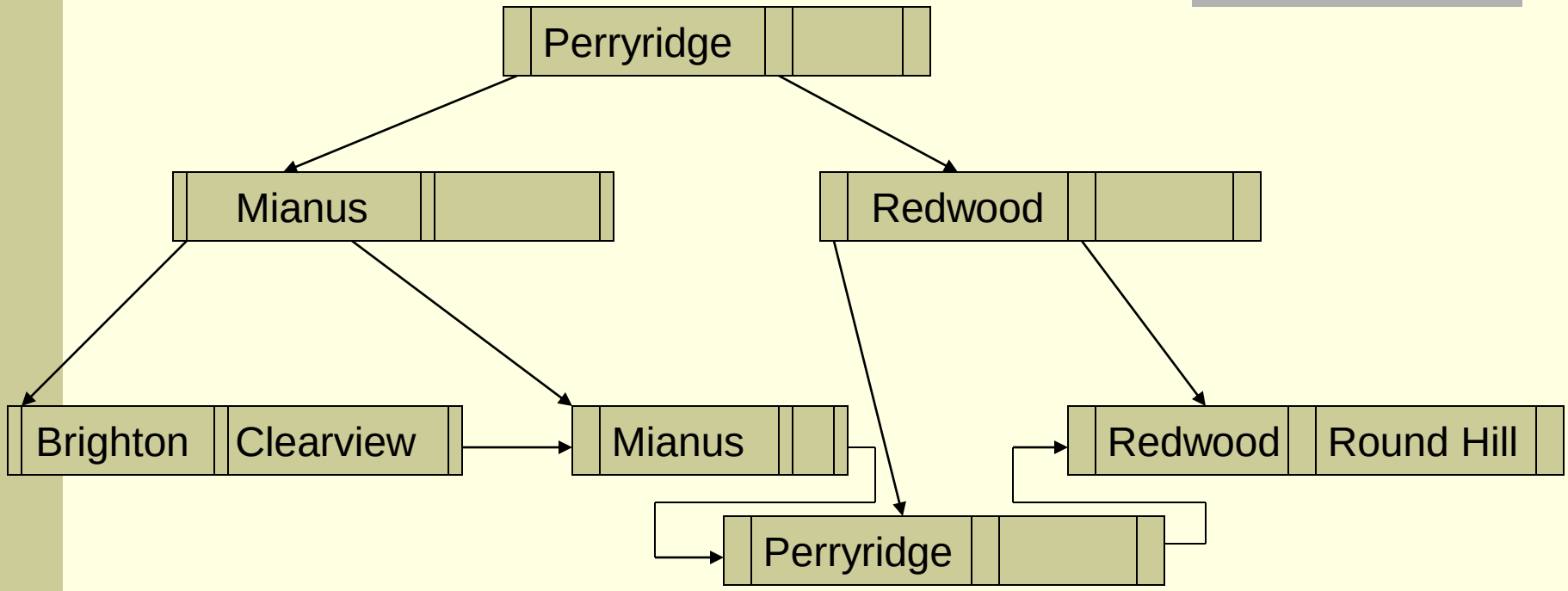
- Insertion – If the new node has a search key that already exists in another leaf node, then it adds the new record to the file and a pointer to the bucket of pointers. If the search key is different from all others, it is inserted in order.
- Deletion – It removes the search key value from the node.

- i.e. we are going to insert a node with a search key value "Clearview". We find that "Clearview" should be in the node with Brighton and Downtown, so we must split the node.

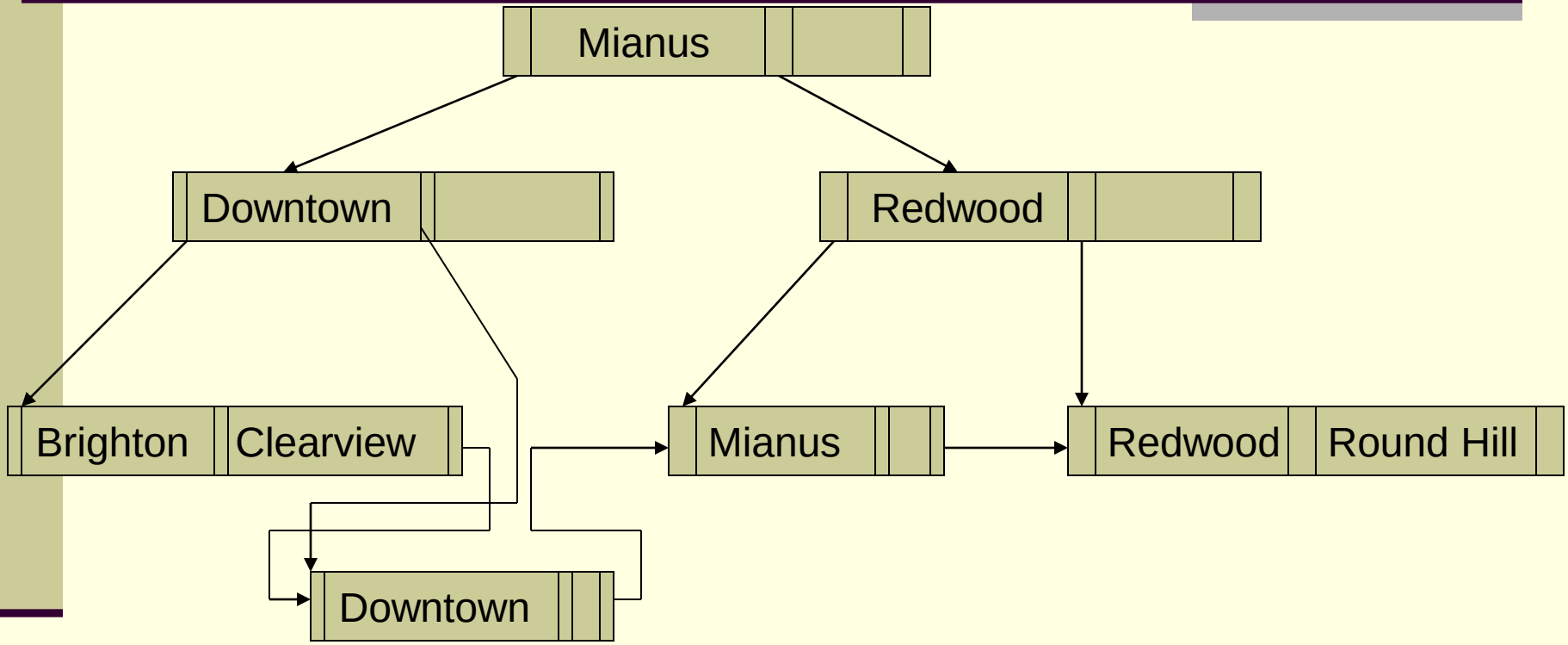
  i.e. n = 3



- i.e. Deletion of “Downtown” from slide #8.



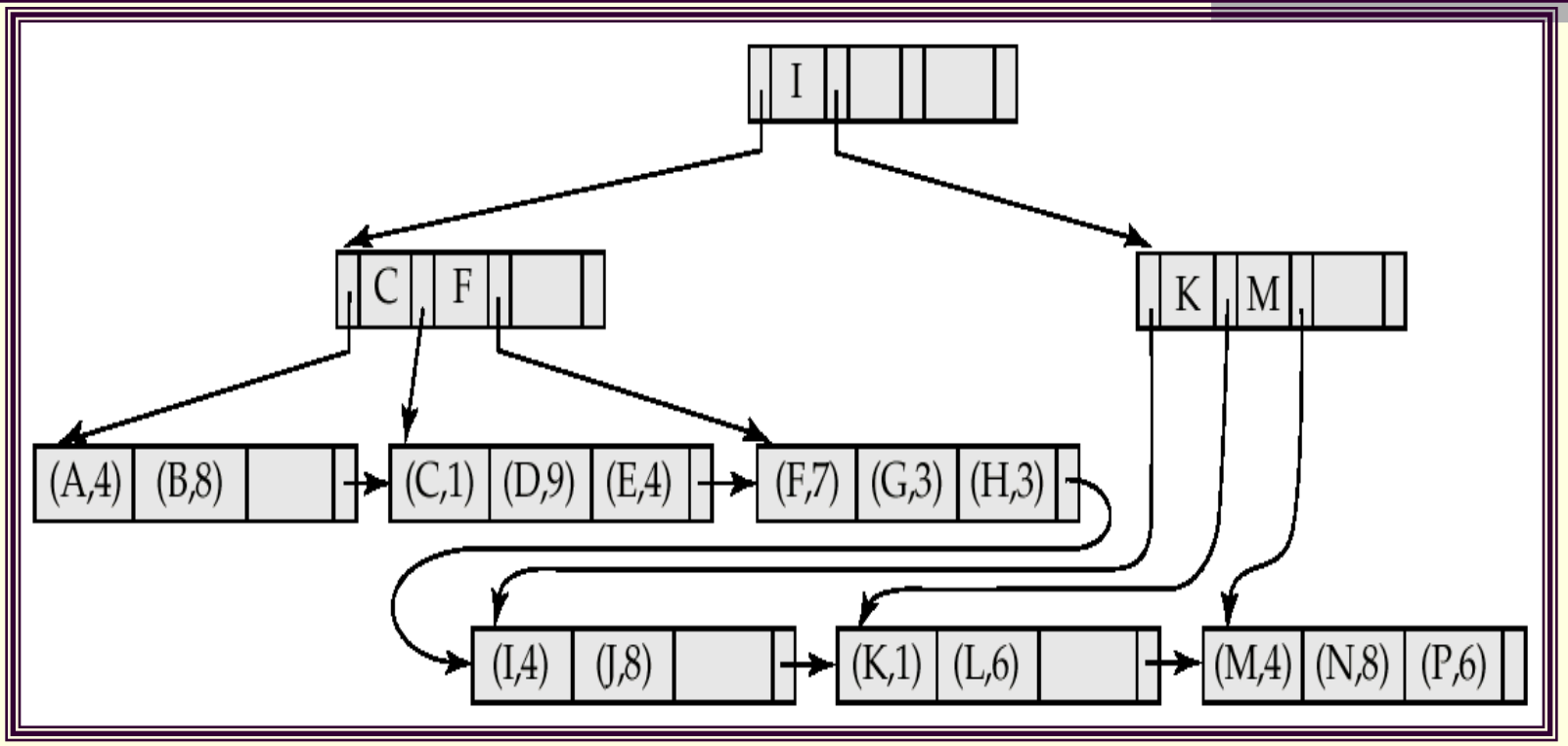
- i.e. Deletion of “Perryridge” from slide #8



B+ - Tree File Organization

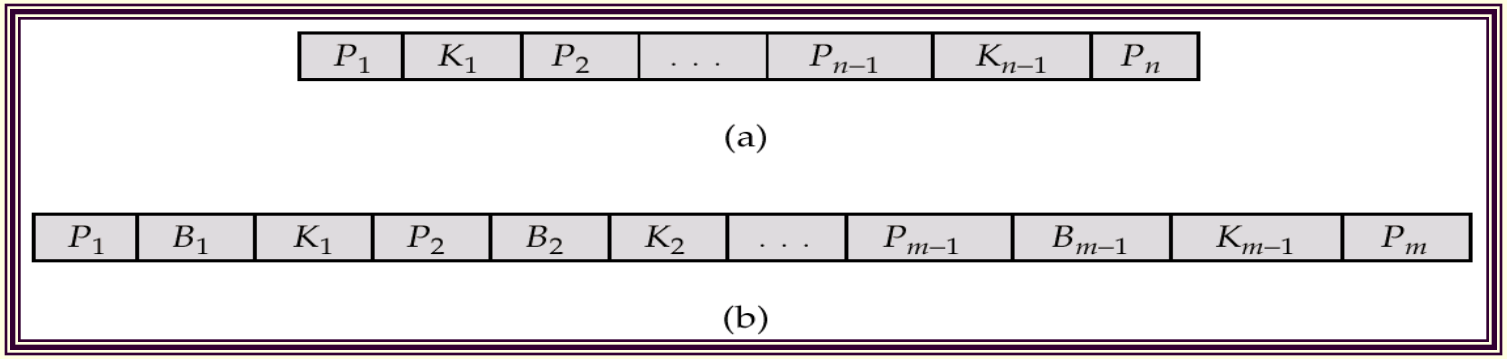
- In a B+ - Tree file organization, the leaf nodes of the tree stores the actual record rather than storing pointers to records.
- During insertion, the system locates the block that should contain the record. If there is enough free space in the node then the system stores it. Otherwise the system splits the record in two and distributes the records.
- During deletion, the system first removes the record from the block containing it. If the block becomes less than half full as a result, the records in the block are redistributed.

B+ - Tree File Organization



B - Tree Index Files

- Similar to B+-tree, but B-tree allows search-key values to appear only once; eliminates redundant storage of search keys.
- Search keys in nonleaf nodes appear nowhere else in the B-tree; an additional pointer field for each search key in a nonleaf node must be included.



B – Tree Index Files

- Advantages of B-Tree indices:
  - May use less tree nodes than a corresponding B^+-Tree.
  - Sometimes possible to find search-key value before reaching leaf node.
- Disadvantages of B-Tree indices:
  - Only small fraction of all search-key values are found early
  - Non-leaf nodes are larger, so fan-out is reduced. Thus B-Trees typically have greater depth than corresponding B^+-Tree
  - Insertion and deletion more complicated than in B^+-Trees
  - Implementation is harder than B^+-Trees.

Example of B – Tree