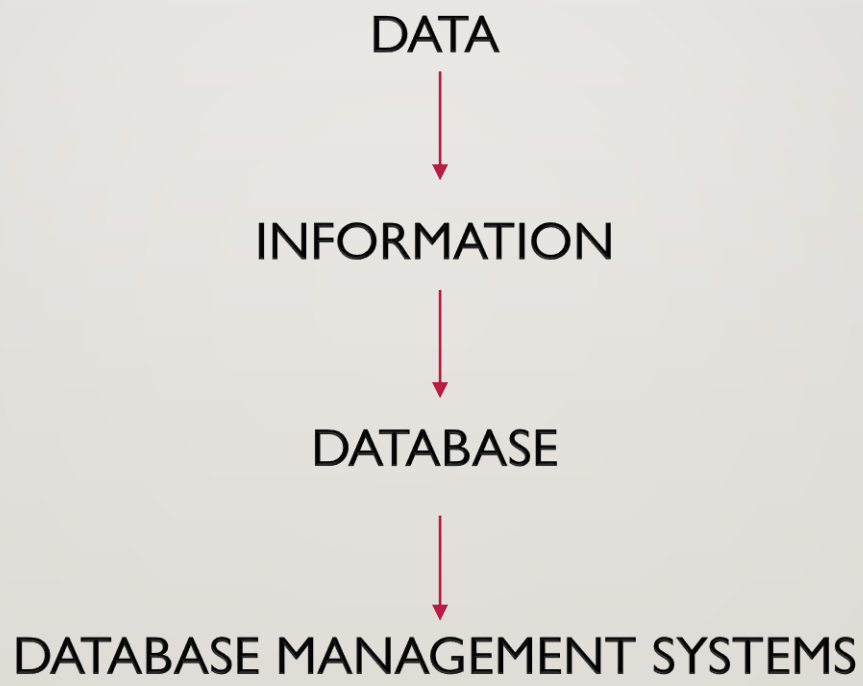
DATABASE MANAGEMENT SYSTEMS

MODULE I

INTRODUCTION



DATA BASE MANAGEMENT SYSTEMS

- **Collection of interrelated data**
- **Set of programs to access the data**
- **An environment that is both *convenient* and *efficient* to use**

APPLICATIONS

- **Banking:** all 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

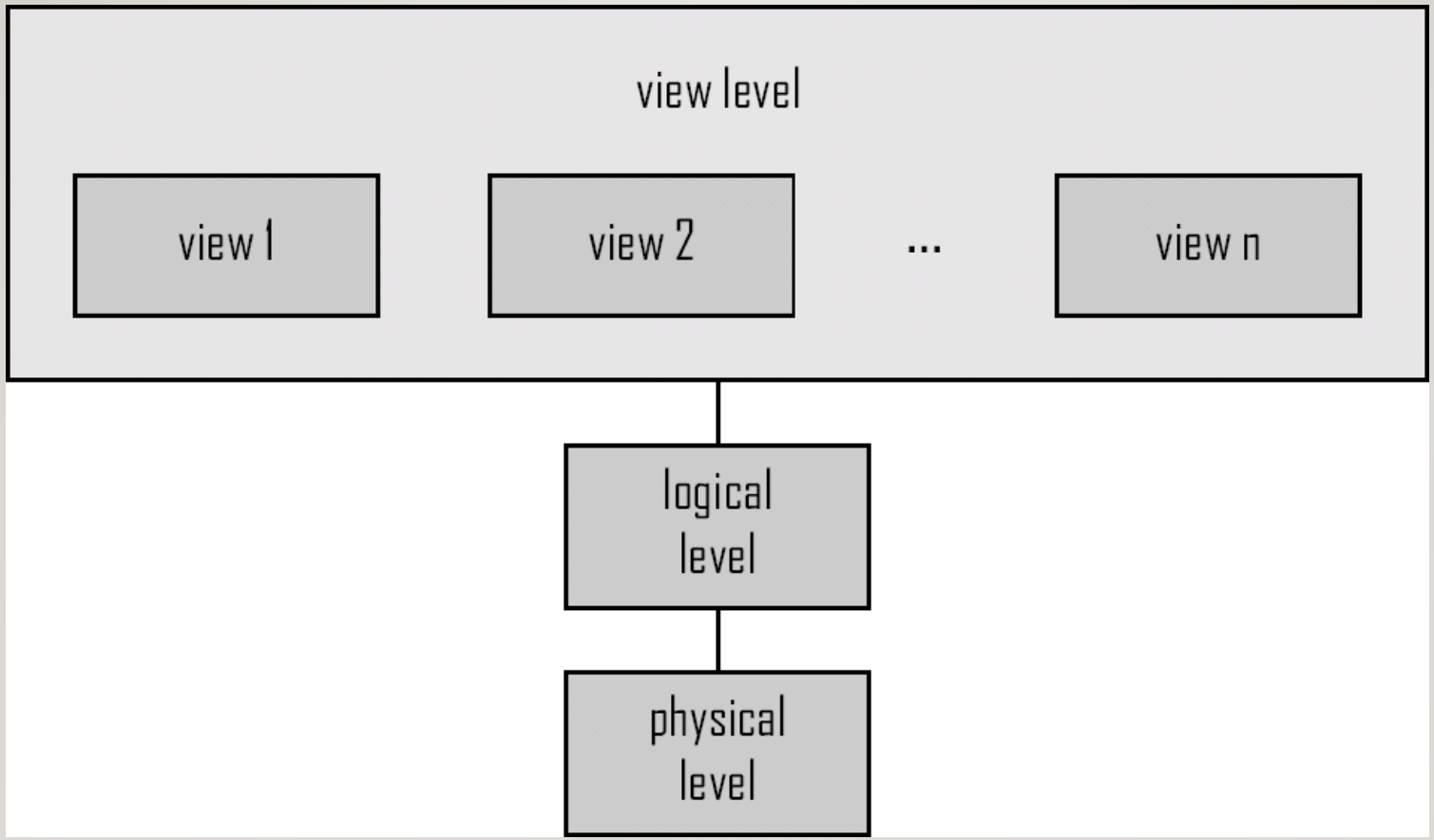
FILE SYSTEM VS DBMS

- **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**
  - **Difficulty in accessing data**
  - **Data isolation — multiple files and formats**
  - **Integrity problems**
  - **Atomicity of updates**
    - **Example: Transfer of funds from one account to another should either complete or not happen at all**
  - **Concurrent access by multiple users**
    - **Example: Two people reading a balance and updating it at the same time**
  - **Security problems**

ADVANTAGES OF DBMS

- Data Independence
- Efficient Data Access
- Data Administration
- Reduced application development time
- Data Integrity and Security
- Concurrent access and Crash recovery

View of Data



Degree/ Level of Data 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 customer = record
        customer_id : string;
        customer_name : string;
        customer_street : string;
        customer_city : string;
    end;
```

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

INSTANCES AND SCHEMAS

- **Instance** –the actual content of the database at a particular point in time.
- 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
  - **Physical schema**: database design at the physical level
  - **Logical schema**: database design at the logical level

  ex: int a=5

  A database schema corresponds to the declaration of a variable. And the database instance corresponds to the value of a variable at given instant.

Example:
The Schema for students relation appear as,
Students(sid:string, name:string, age:integer, gpa:real)
The instance of the students relation appear as,

Sid	Name	Age	Cgpa
50000	Dave	19	3.3
53666	Jones	18	3.4
53831	Smith	19	2.8

DATA INDEPENDENCE

- The ability to modify the schema in one level without affecting the schema in next higher level is called data independence.
- **Logical data independence**: The ability to modify the logical schema without affecting the schema in next higher level (external schema.)
- **Physical Data Independence** –the ability to modify the physical schema without changing the logical schema

DBMS structure

DBMS structure

Database Users

Users are differentiated by the way they expect to interact with the system

- **Application programmers** –are computer professionals who write appn prgms. They use RAD tools to construct forms and reports with minimum programming effect.
- **Sophisticated users** –interact with the system without writing programs, instead they form their requests in a database query language
- **Specialized users** –write specialized database applications that do not fit into the traditional data processing framework
- Ex:Computer aided design systems, knowledgebase expert systems.
- **Naïve users** –invoke one of the permanent application programs that have been written previously
  - Examples, people accessing database over the web, bank tellers, clerical staff

Database Administrator

- Has central control of both data and programs to access that data.
- Coordinates all the activities of the database system
  - has a good understanding of the enterprise's
    - information resources and needs.
- Database administrator's duties include:
  - Storage structure and access method definition
  - Schema and physical organization modification
  - Granting users authority to access the database
  - Backing up data
  - Monitoring performance and responding to changes
  - Periodically backing up the database,either on
    - tapes or onto remote servers.

Data storage and Querying

- Storage management
- Query processing
- Transaction processing

Storage Management

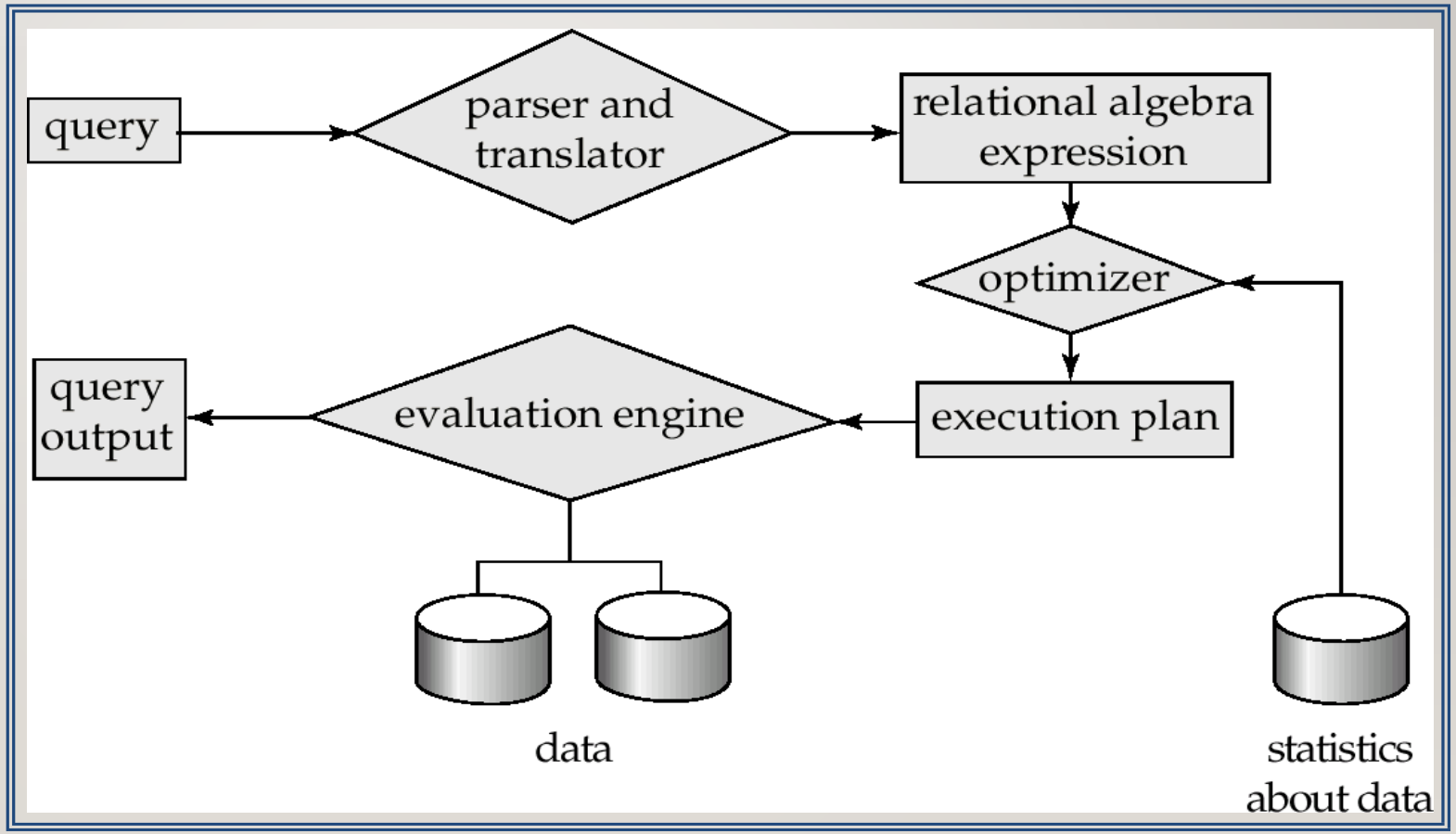
- **Storage manager is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.**
- **The storage manager is responsible to the following tasks:**
  - **Interaction with the file manager**
  - **Efficient storing, retrieving and updating of data**
- **Storage mngr implements several data structures**
  - **Data files**
  - **Data dictionary**
  - **Indices**

- Authorization and integrity mngr tests for satisfaction of integrity constraints and checks the authority of users to access the data
- Transaction mngr ensures databse remains in consistent state despite system failures and concurrent transaction executions proceed without conflicting
- File mngr manages allocation of space on disk storage and the data structures used to represent data on disk
- Buffer mngr which is responsible for fetching data from disk storage into main memory and deciding what data to cache in main memory

Query Processing

DDL interpreter interprets DDL stmts and records the defninitions in data dictionary

1.Parsing and translation
2.Optimization
3.Evaluation



Transaction Management

- 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.

DATA MODELS

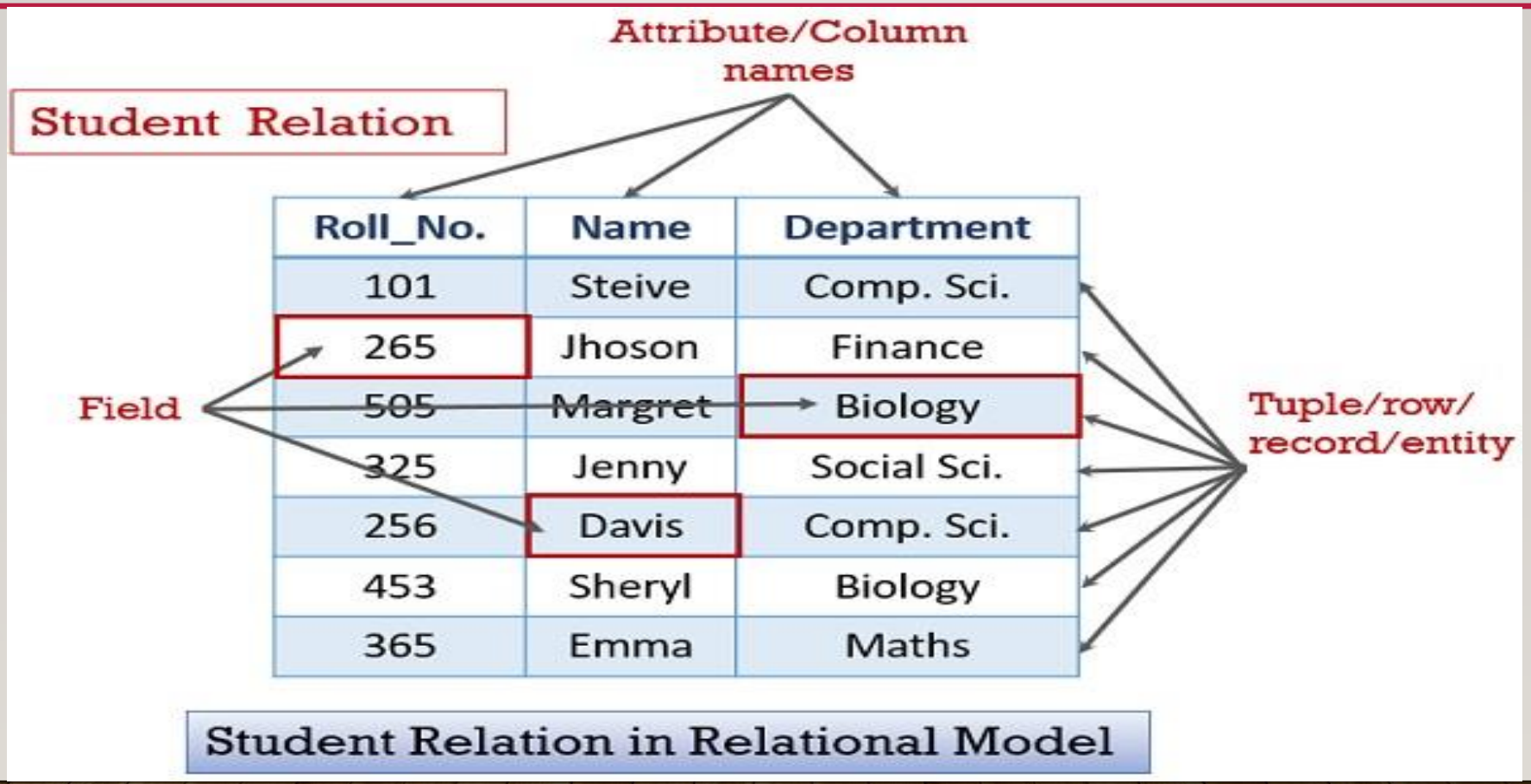
- Underlying the structure of database is data model.
- It is a collection of tools for describing
  - Data ,Data relationships,Data semantics & consistency constraints.
- A Data model provides a way to describe the design of a database at physical, logical and view level.

- **Data Modelling** is the process of creating a data modelfor the data to be stored in a database.

DATA MODEL TYPES

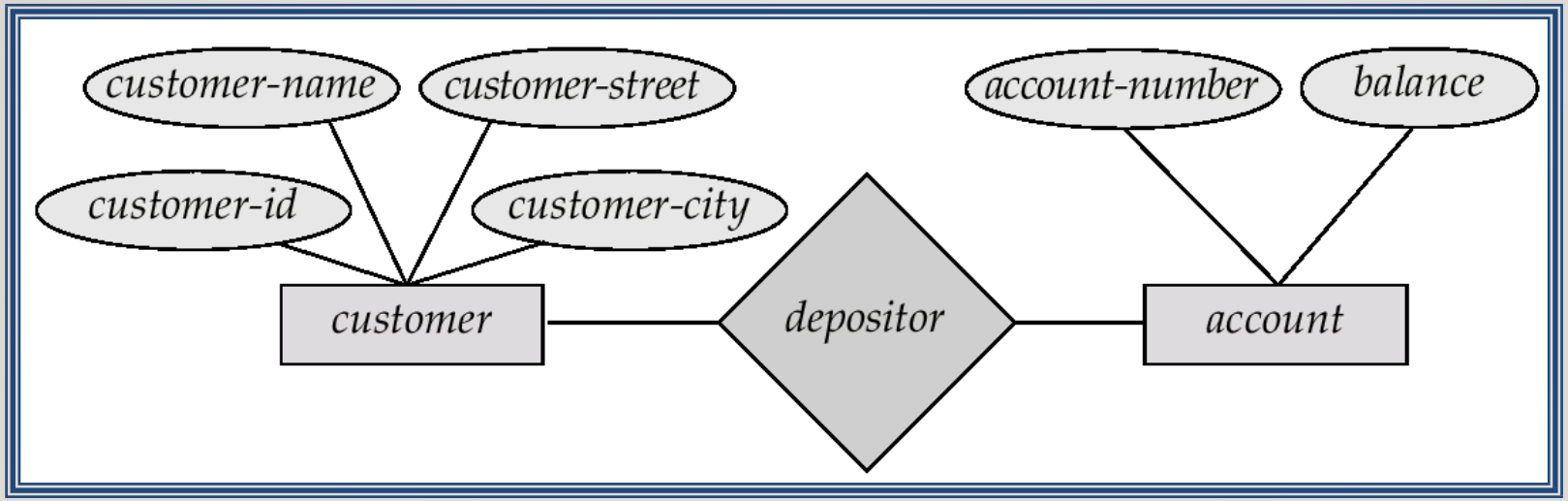
- Relational model
- Entity-Relationship data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Semi structured data model (XML)
- Other older models:
  - Network model
  - Hierarchical model

RELATIONAL DATA MODEL



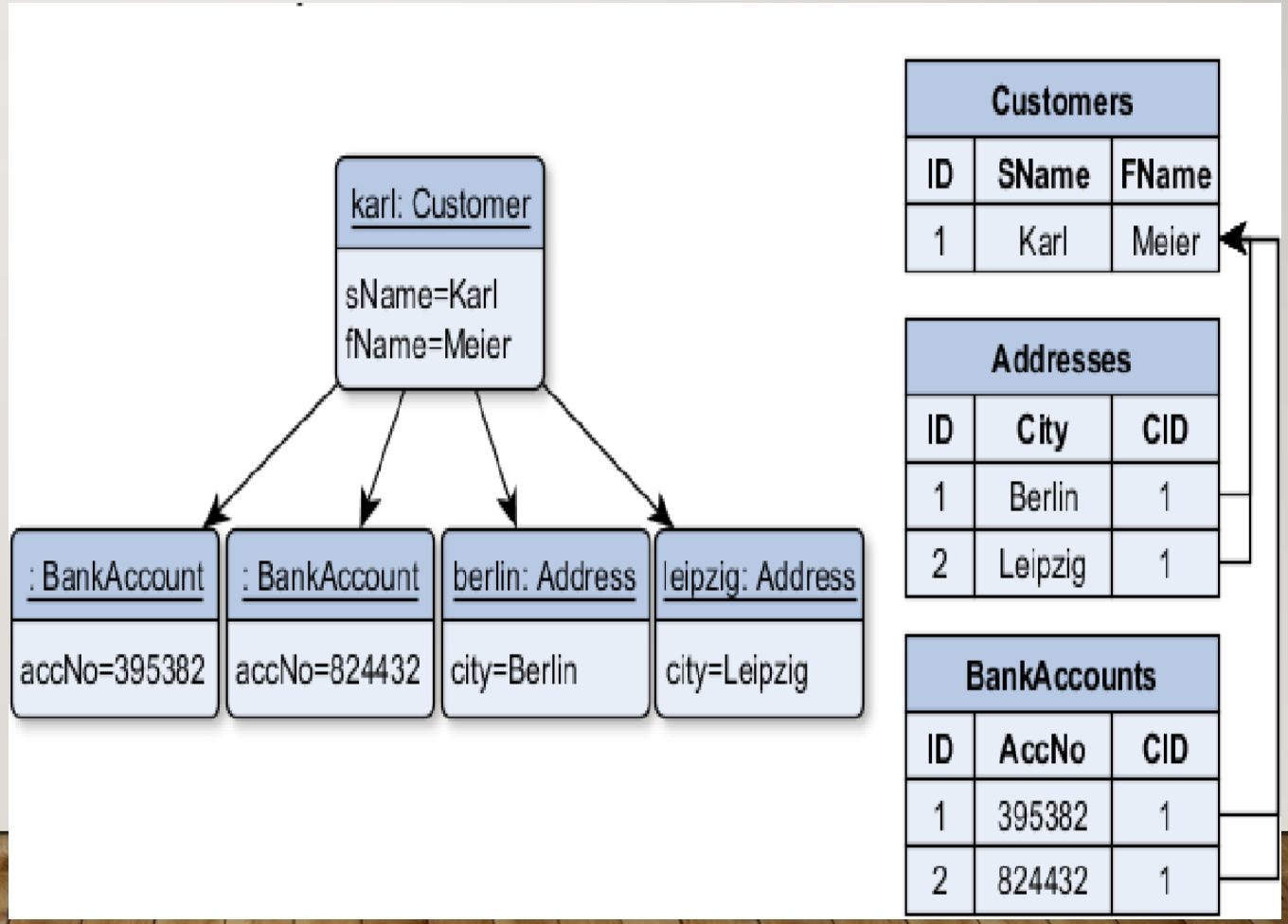
ENTITY- RELATIONSHIP DATA MODEL(ER MODEL)

- An entity is a thing or object in the real world that is distinguishable from other objects.
- ❖ Rectangles represent entities
- ❖ Diamonds represent relationship among entities.
- ❖ Ellipse represent attributes
- ❖ Lines represent link of attributes to entities to relationships.

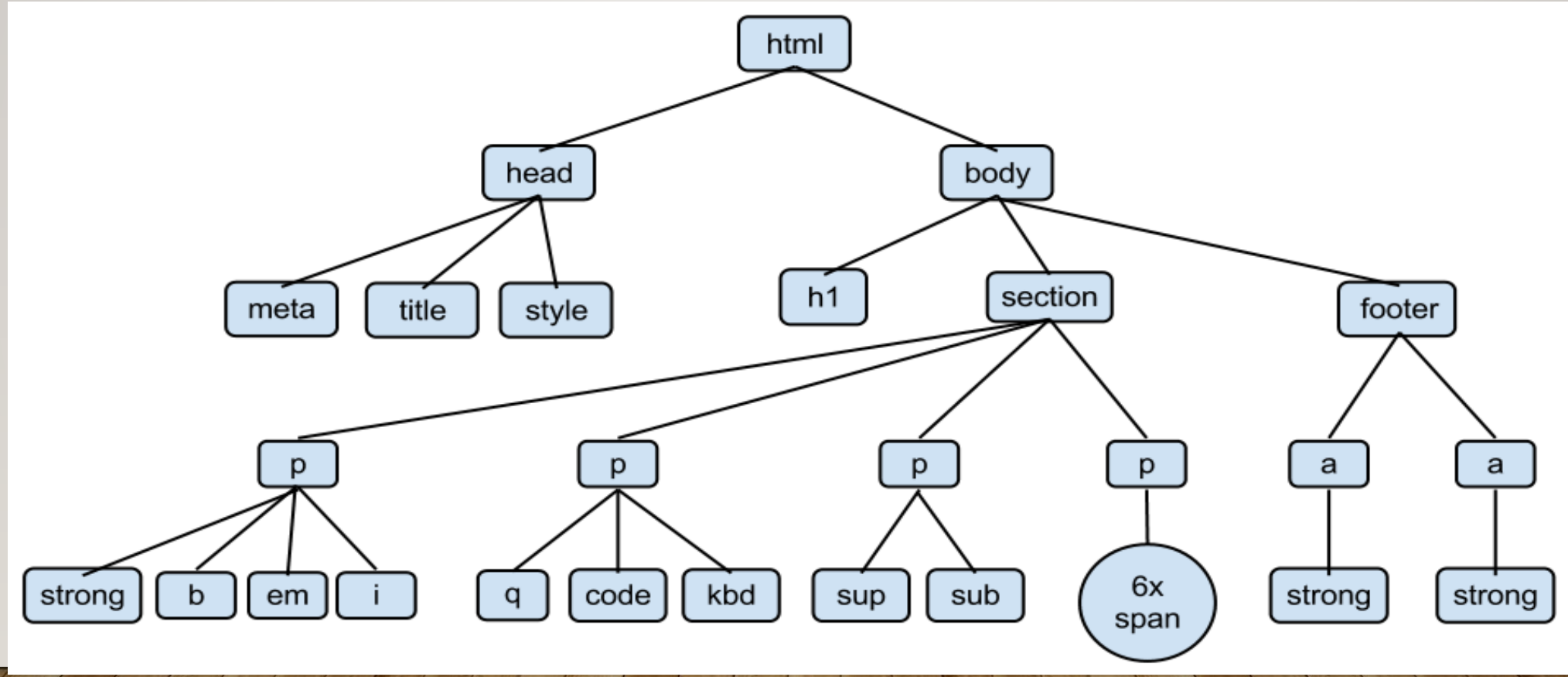


OBJECT BASED DATA MODEL

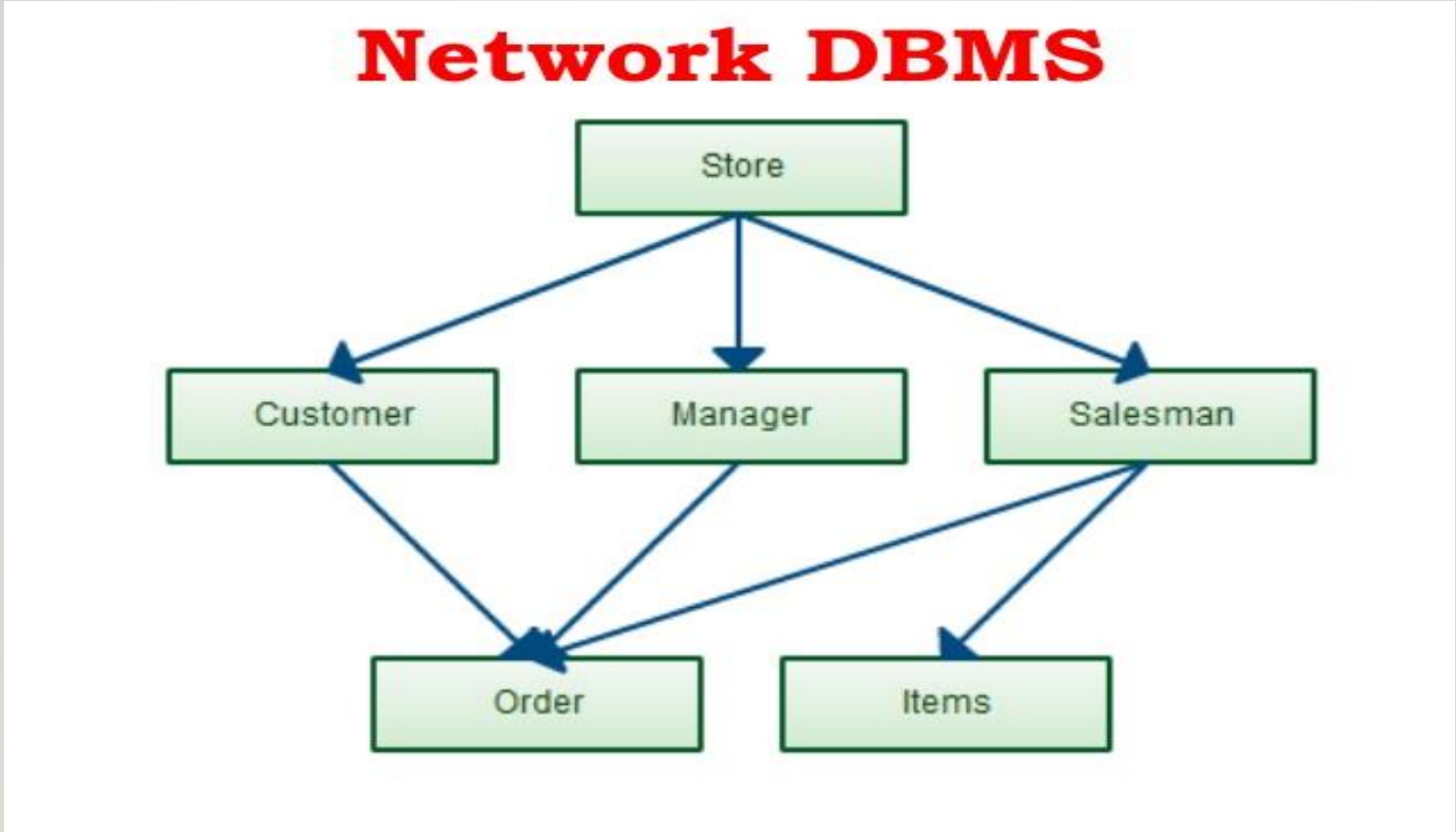
- It is based on object oriented programming language paradigm.
- Inheritance,object identity and encapsulations
- It can be seen as extending the E-R model with opps concepts.



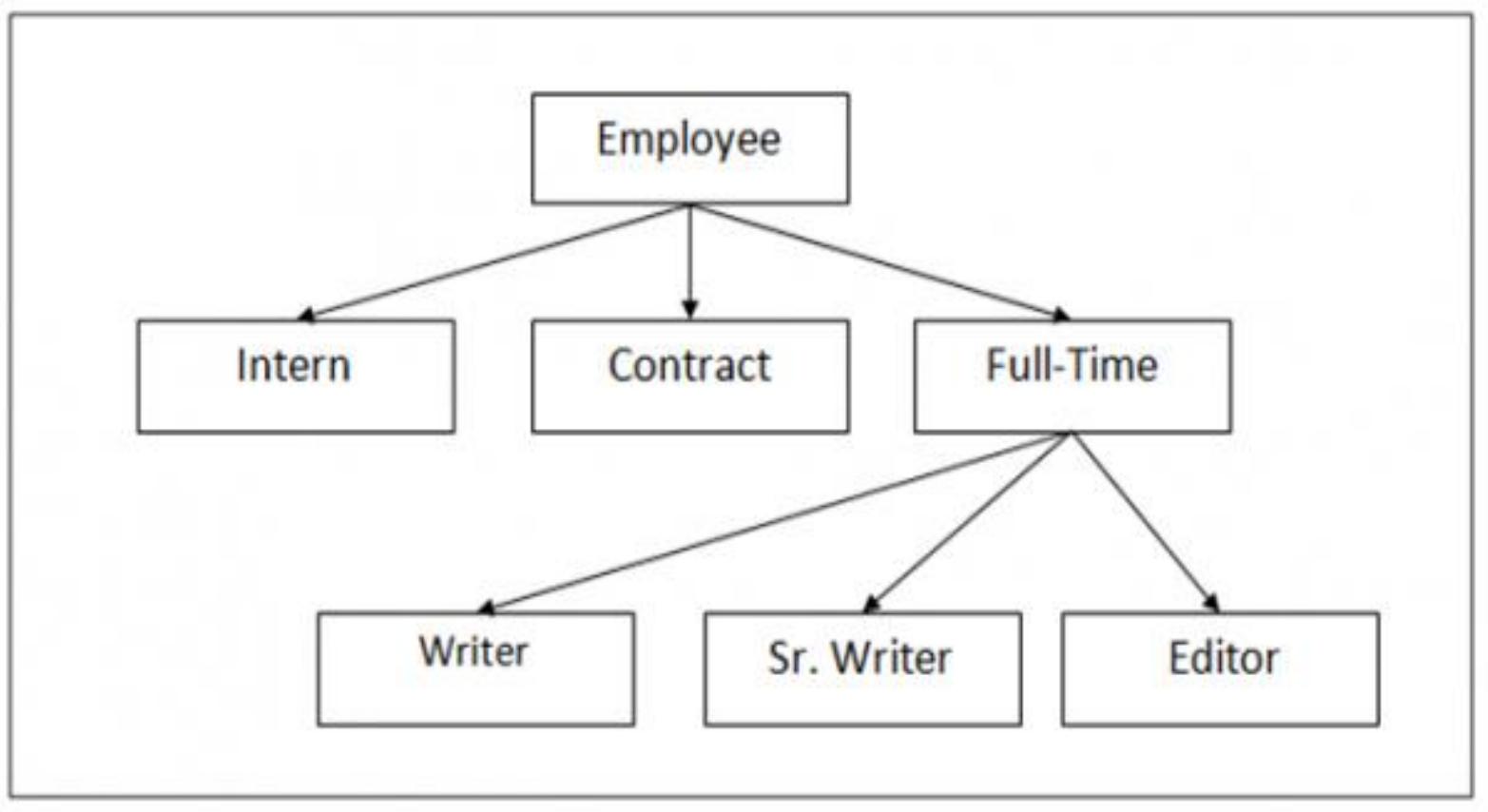
SEMI STRUCTURED DATA MODEL



NETWORK DATA MODEL



HIERARCHICAL MODEL



IMPORTANCE OF DATA MODELS

- Facilitate interaction among the designer, the applications programmer, and the end user.
- End users have different views and needs for data.
- Data model organizes data for various users.
- Data model is an abstraction
  - Cannot draw required data out of the data model.

DATA MODEL BASIC BUILDING BLOCKS

- Entity
- Attribute
- Relationship
  - One-to-many
  - Many-to-many
  - One-to-one
- Constraint

Evolution of Data Models:

TABLE 3.1 Evolution of Major Data Models

GENERATION	TIME	DATA MODEL	EXAMPLES	COMMENTS
First	1960s–1970s	File system	VMS/VSAM	Used mainly on IBM mainframe systems Managed records, not relationships
Second	1970s	Hierarchical and network	IMS, ADABAS, IDS-II	Early database systems Navigational access
Third	Mid-1970s	Relational	DB2 Oracle MS SQL Server MySQL	Conceptual simplicity Entity relationship (ER) modeling and support for relational data modeling
Fourth	Mid-1980s	Object-oriented Object/ relational (O/R)	Versant Objectivity/DB DB2 UDB Oracle 11g	Object/relational supports object data types Star Schema support for data warehousing Web databases become common
Fifth	Mid-1990s	XML Hybrid DBMS	dbXML Tamino DB2 UDB Oracle 11g MS SQL Server	Unstructured data support O/R model supports XML documents Hybrid DBMS adds object front end to relational databases Support large databases (terabyte size)
Emerging Models: NoSQL	Late 2000s to present	Key-value store Column store	SimpleDB (Amazon) BigTable (Google) Cassandra (Apache)	Distributed, highly scalable High performance, fault tolerant Very large storage (petabytes) Suited for sparse data Proprietary API

HISTORY OF DATABASE SYSTEMS

- **1950s and early 1960s:**

  First general purpose DBMS was designed by charles bachman at general electric was called Integrated data store. He is first to receive ACM'S turing award(1973).

  - Data processing using magnetic tapes for storage
    - Tapes provide only sequential access
  - Punched cards for input
- **Late 1960s and 1970s:**

  In late 1960's IBM developed information mangmt system(IMS) DBMS used even today in major installations.

  - Hard disks allow direct access to data
  - Network and hierarchical data models in widespread use
  - In 1970 Edgar Codd defined new data representation framework -relational data model.
  - ACM'S turing award(1981).

History (cont.)

- **1980s:**
  - Research relational prototypes evolve into commercial systems
    - SQL becomes industry 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
- **2000s:**
  - XML and XQuery standards
  - Automated database administration
  - Increasing use of highly parallel database systems
  - Web-scale distributed data storage systems

Database Management Systems (DBMS)

1960's	Hierarchical	Network
1970's	Relational	
1990's	Object-oriented	Object-relational
1995+	Java	XML
	CMDB	Mobile
	IMDB	Embedded

INTRODUCTION TO DATABASE DESIGNAND ER DIAGRAMS

- The database design can be divided into 6 steps.ER model is relevent to first 3steps

  1.Requirement analysis

  2.Conceptual database design

  3.Logical database design

  4.Schema refinement

  5.Physical database design:.Ex:Indexes

  6.Application and security design

DATABASE DESIGN

- *Conceptual design*:*(ER Model is used at this stage.)*
  - What are the *entities* and *relationships* in the enterprise?
  - What information about these entities and relationships should we store in the database?
  - What are the *integrity constraints* or *business rules* that hold?
  - A database `schema' in the ER Model can be represented pictorially (*ER diagrams*).
  - Can map an ER diagram into a relational schema.

ENTITY-RELATIONSHIP (E/R) MODEL

- Widely used conceptual level data model
  - proposed by Peter P Chen in 1970s
- Data model to describe the database system at the requirements collection stage
  - high level description.
  - easy to understand for the enterprise managers.
  - rigorous enough to be used for system building.
- Concepts available in the model
  - entities and attributes of entities.
  - relationships between entities.
  - diagrammatic notation.

ENTITIES

- *Entity* - a thing (animate or inanimate) of independent physical or conceptual existence and *distinguishable*.

  In the University database context, an individual *student*, *faculty member*, a *class room*, a *course* are entities.

- *Entity Set* or *Entity Type*-

  Collection of entities all having the same properties. *Student* entity set – collection of all *student* entities. *Course* entity set – collection of all *course* entities.

TYPES OF ATTRIBUTES

- Simple Attributes
  - having atomic or indivisible values.

    `example`: *Dept* – a string

    PhoneNumber – an eight digit number
- Composite Attributes
  - having several components in the value.

    `example`: *Qualification* with components

    (*DegreeName*, *Year*, *UniversityName*)
- Derived Attributes
  - Attribute value is dependent on some other attribute.

    `example`: *Age* depends on *DateOf Birth*.

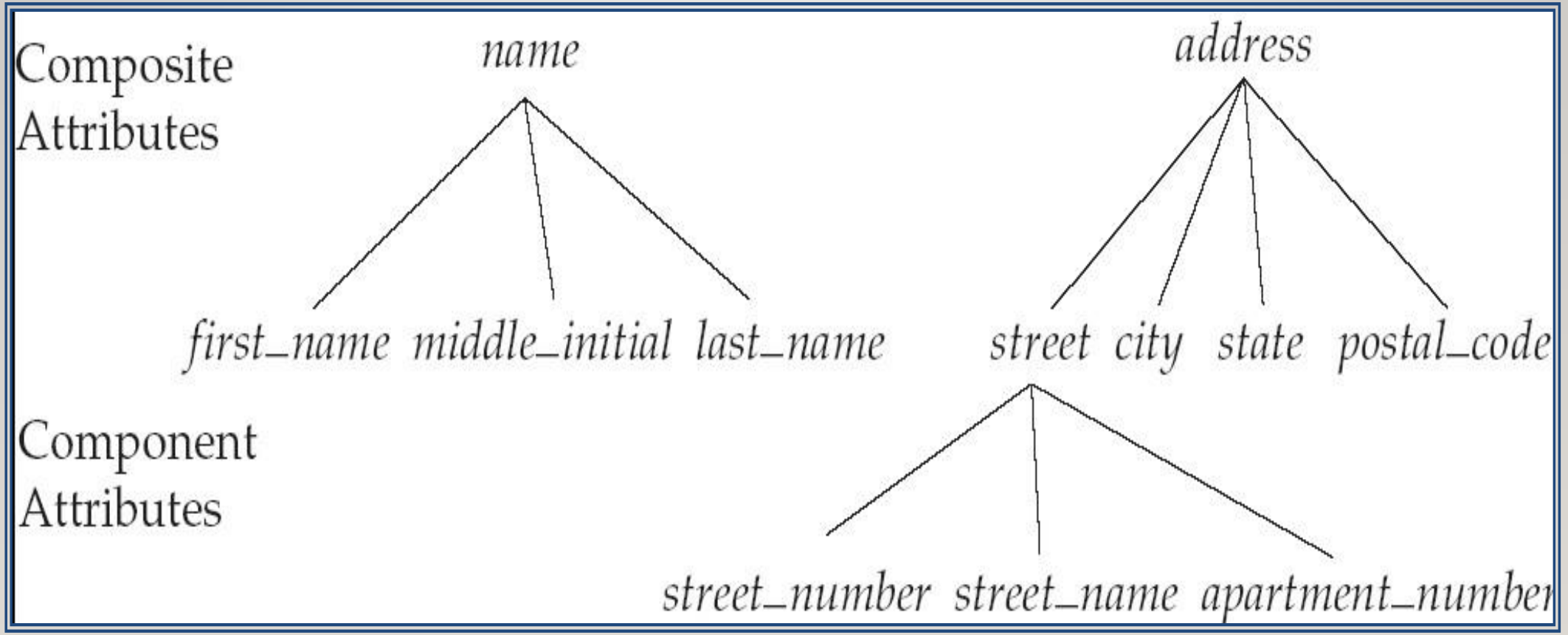
    So age is a derived attribute.

Types of Attributes

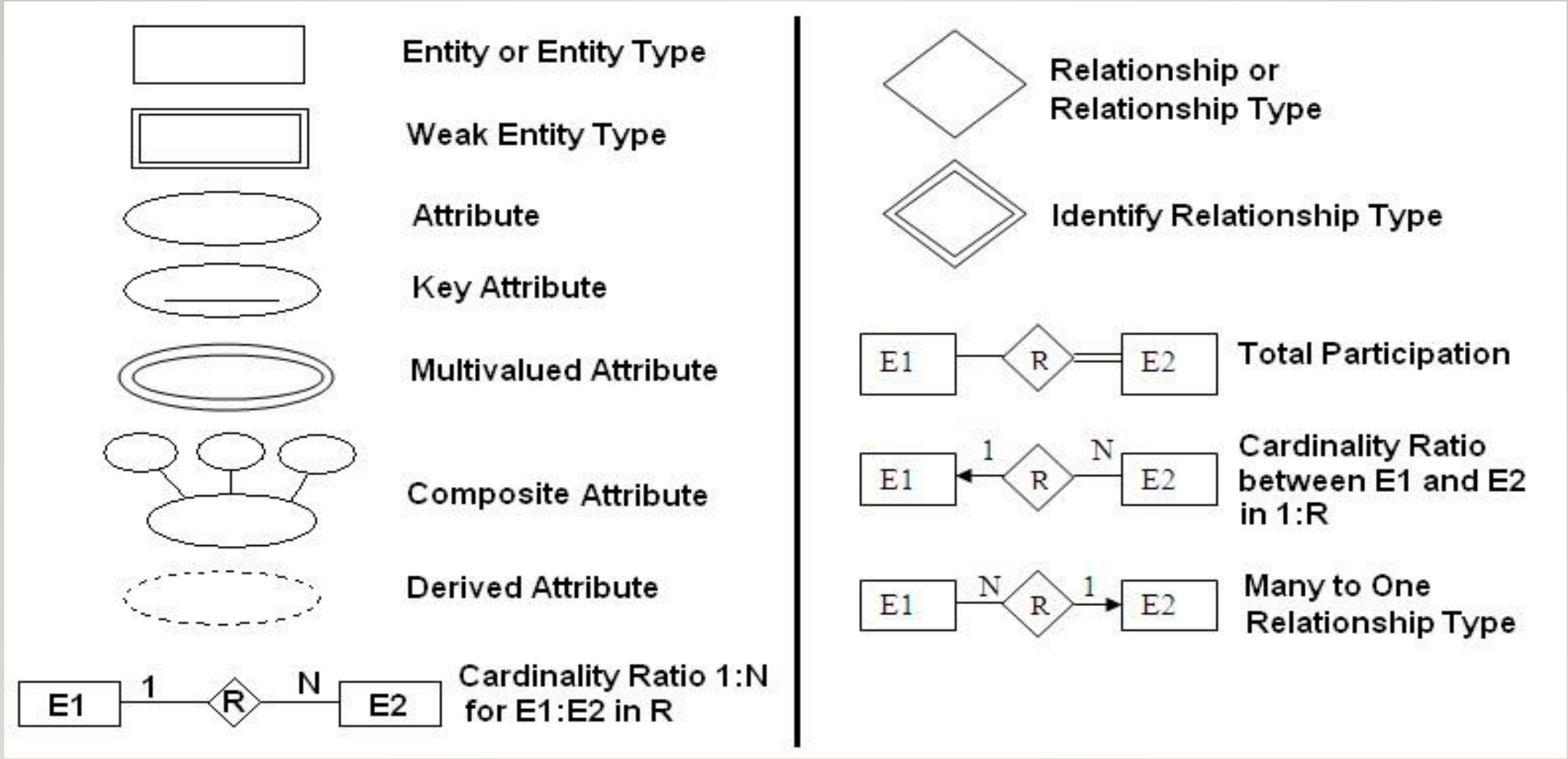
- Single-valued
  - having only one value rather than a set of values.
  - for instance, *PlaceOfBirth* – single string value.
- Multi-valued
  - having a set of values rather than a single value.

    for instance, *CoursesEnrolled* attribute for student *EmailAddress* attribute for student *PreviousDegree* attribute for student.
- Attributes can be:
  - simple single-valued, simple multi-valued,
  - composite single-valued or composite multi-valued.

Composite Attributes

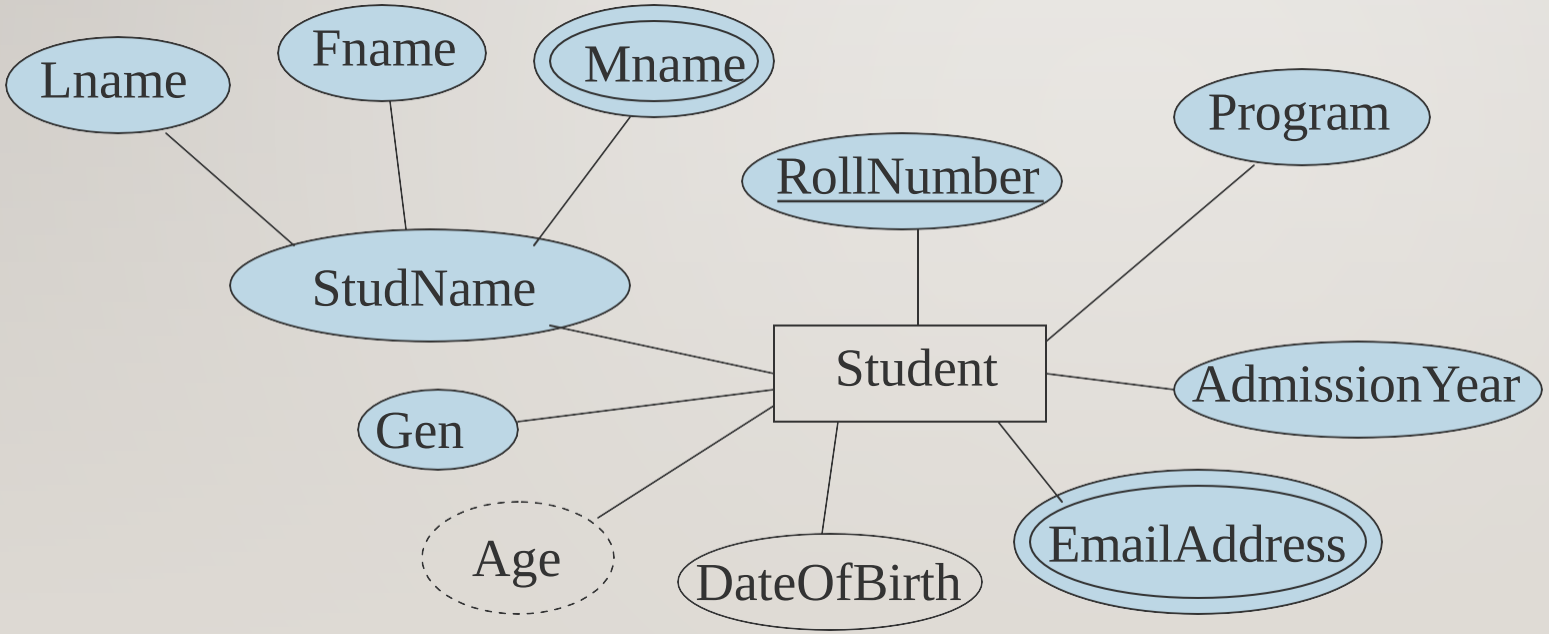


Symbols of ER Model



DIAGRAMMATIC NOTATION FOR ENTITIES

entity - rectangle
attribute - ellipse connected to rectangle
multi-valued attribute - double ellipse
composite attribute - ellipse connected to ellipse
derived attribute - dashed ellipse



DOMAINS OF ATTRIBUTES

- Each attribute takes values from a set called its *domain*
- For instance, *studentAge* – {17,18, …, 55}

  HomeAddress – character strings of length 35
- Domain of composite attributes –

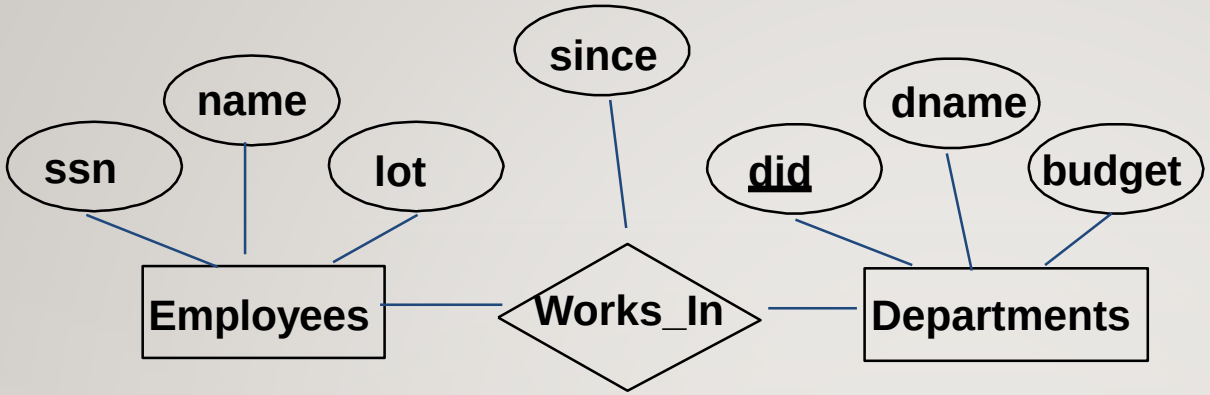
  cross product of domains of component attributes
- Domain of multi-valued attributes –

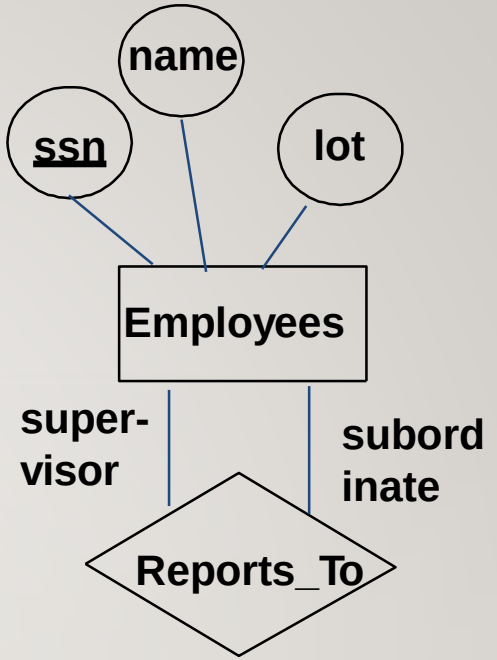
  set of subsets of values from the basic domain

ENTITY SETS AND KEY ATTRIBUTES

- *Key* – an attribute or a collection of attributes whose value(s) uniquely identify an entity in the entity set.
- For instance,
  - *RollNumber* - Key for *Student* entity set
  - *EmpID* - Key for *Faculty* entity set
  - *HostelName, RoomNo* - Key for *Student* entity set (assuming that each student gets to stay in a single room)
- A key for an entity set may have more than one attribute.
- An entity set may have more than one key.
- Keys can be determined only from the meaning of the attributes in the entity type.
  - Determined by the designers

RELATIONSHIPS





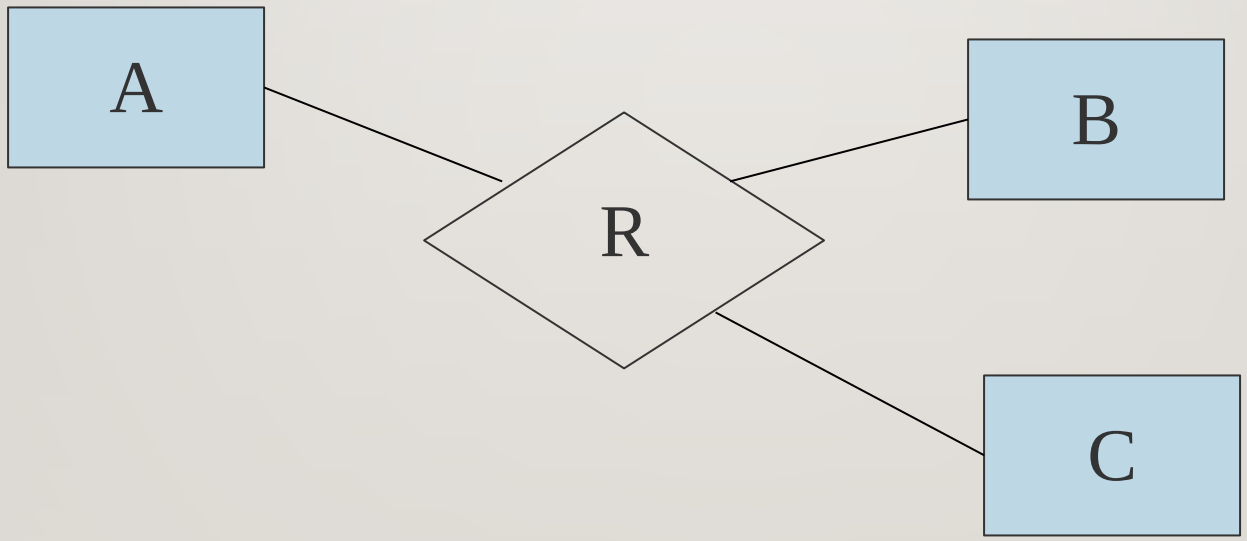
- ***Relationship***: **Association among two or more entities. E.g., Attishoo works in Pharmacy department.**
- ***Relationship Set***: **Collection of similar relationships.**
- **{(e1,…e2)|e1Ꞓ E1, e2Ꞓ E2….. enꞒ En}**

DEGREE OF A RELATIONSHIP

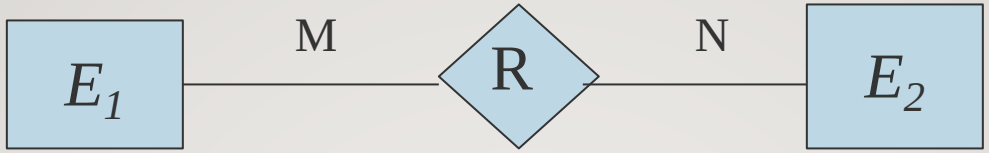
- *Degree*: the number of participating entities.
  - Degree 2: Binary
  - Degree 3: Ternary
  - Degree n: n- ary
- Binary relationships are very common and widely used.

DIAGRAMMATIC NOTATION FOR RELATIONSHIPS

- Relationship – diamond shaped box
  - Rectangle of each participating entity is connected by a line to this diamond. Name of the relationship is written in the box.



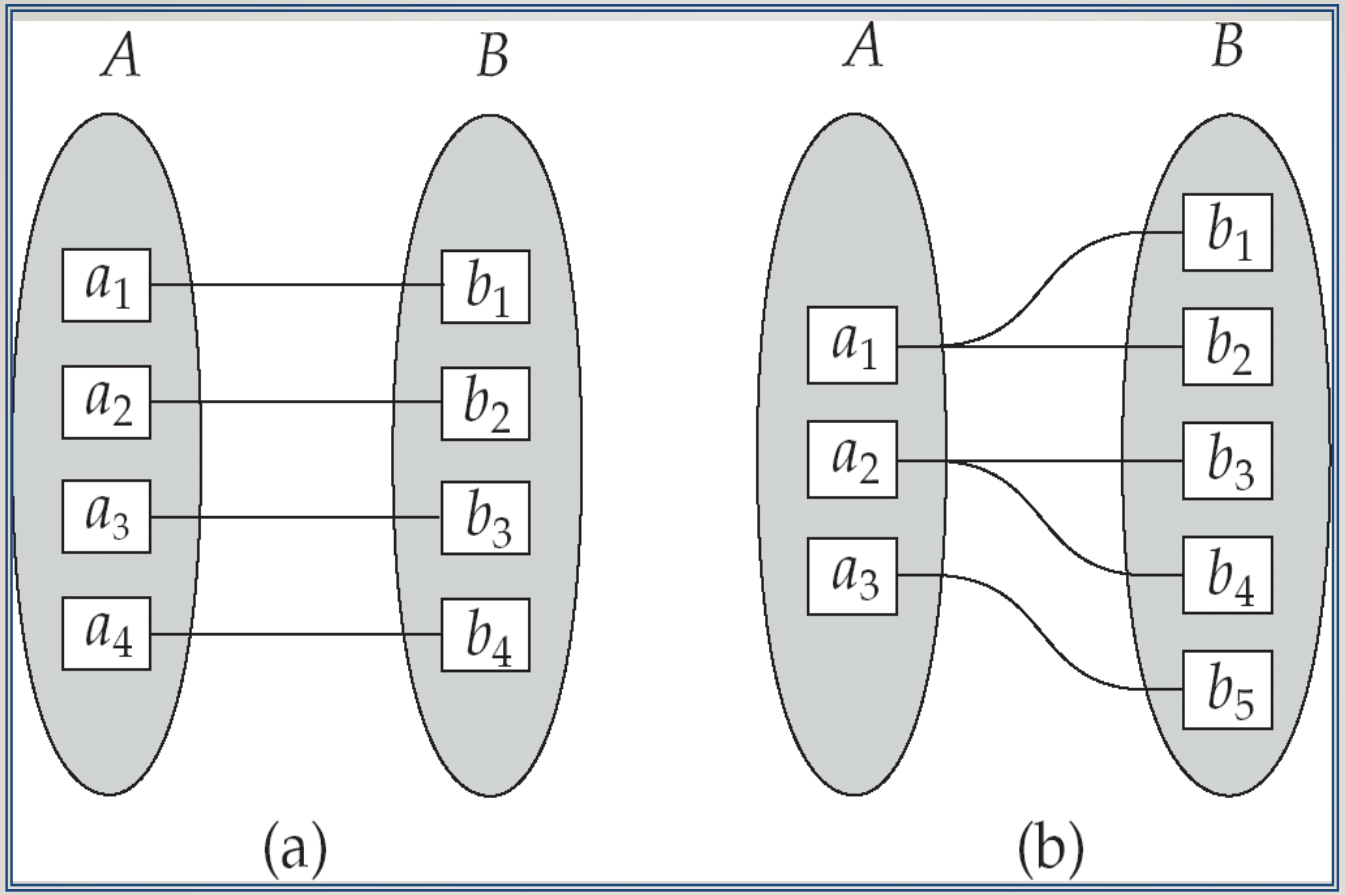
BINARY RELATIONSHIPS AND CARDINALITY RATIO



- The number of entities from E_2 that an entity from E_1 can possibly be associated thru R (and vice-versa) determines the *cardinality ratio* of R.

- Four possibilities are usually specified:
  - *one-to-one (1:1)*
  - *one-to-many (1:N)*
  - *many-to-one (N:1)*
  - *many-to-many (M:N)*

MAPPING CARDINALITIES

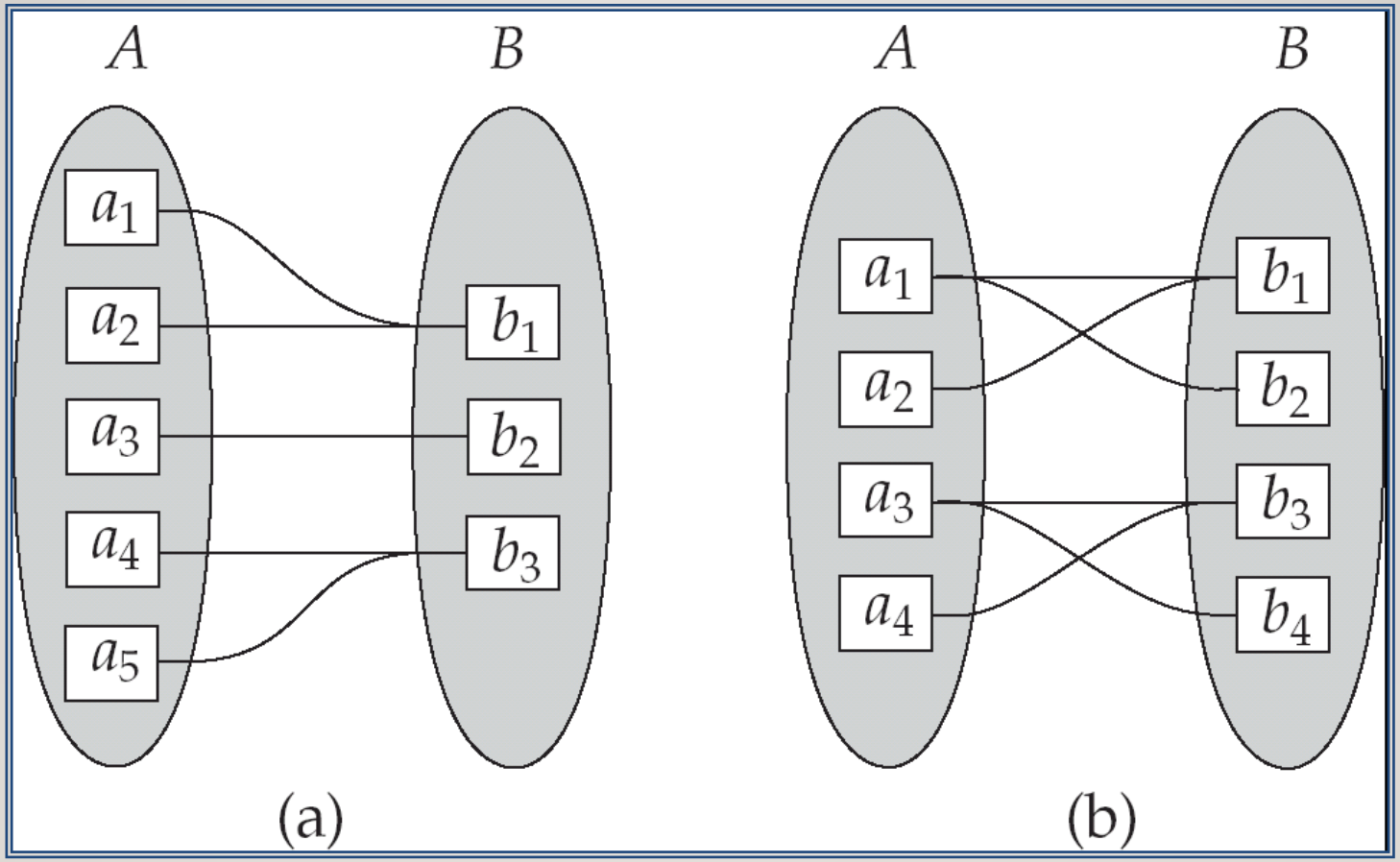


One to one

One to many

Note: Some elements in *A* and *B* may not be mapped to any elements in the other set

MAPPING CARDINALITIES

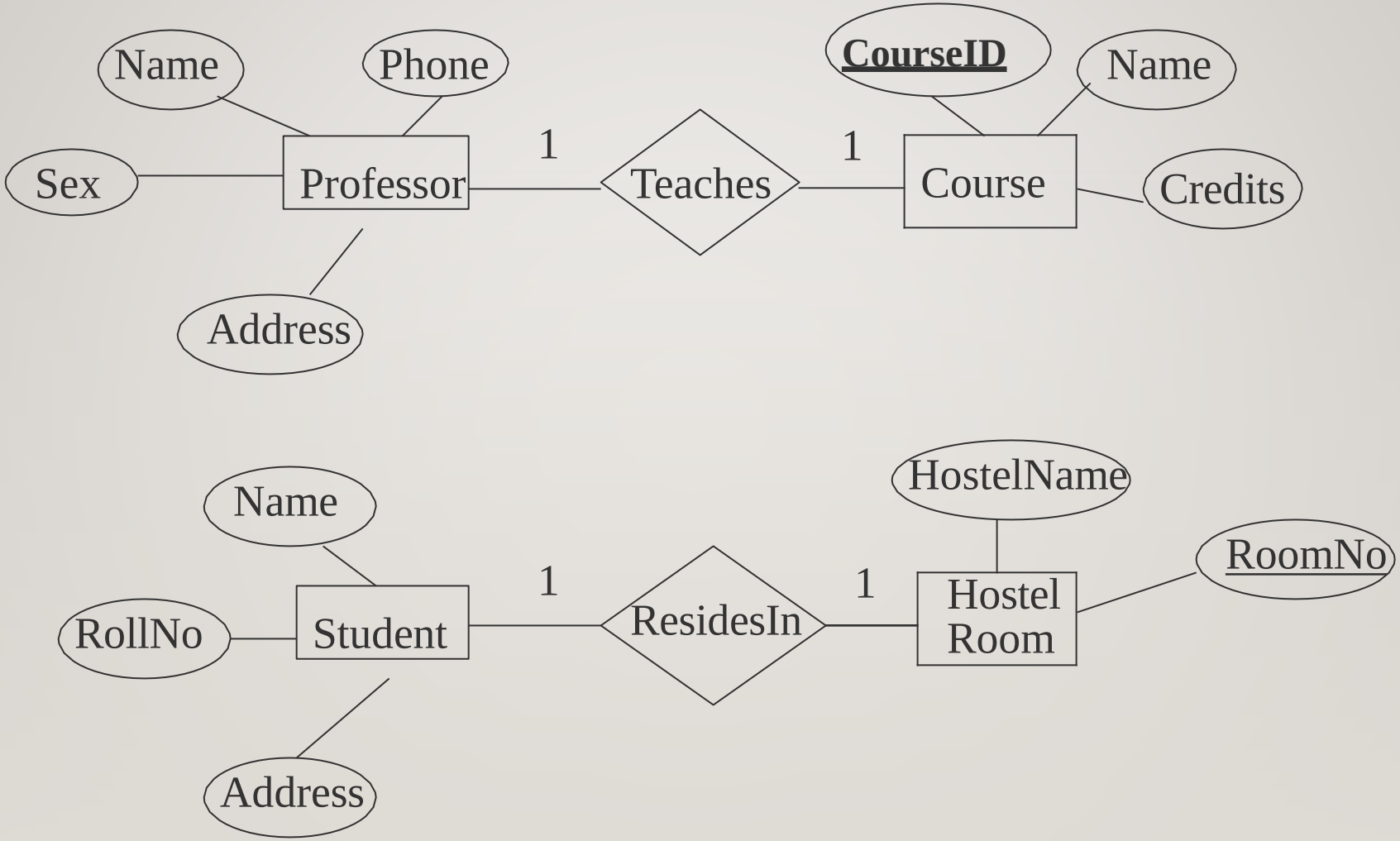


Many to one

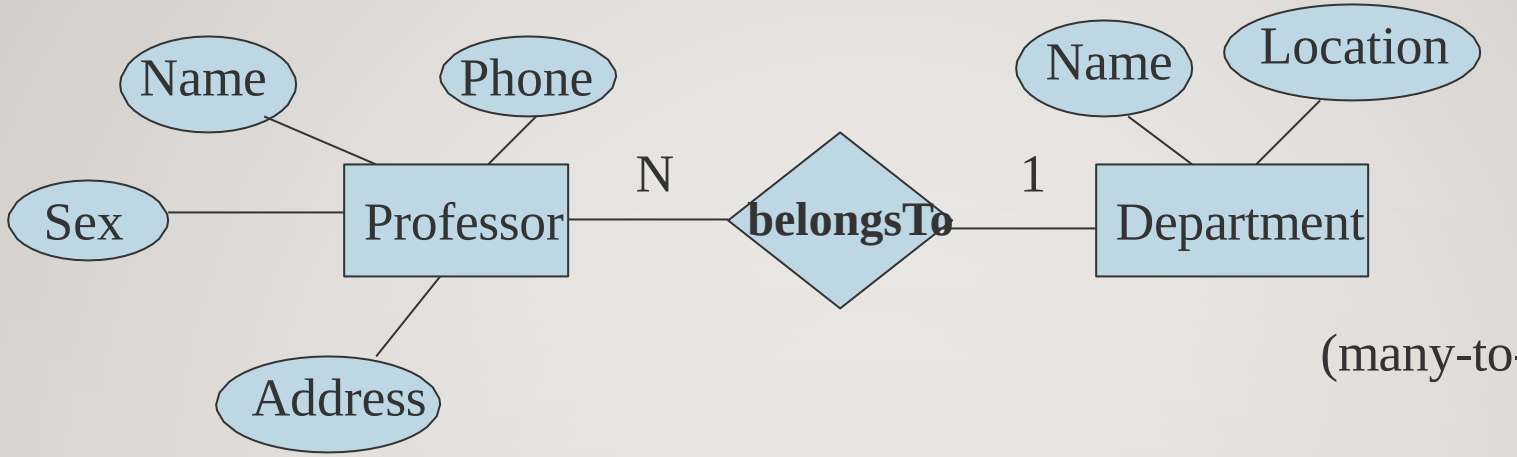
Many to many

Note: Some elements in A and B may not be mapped to any elements in the other set

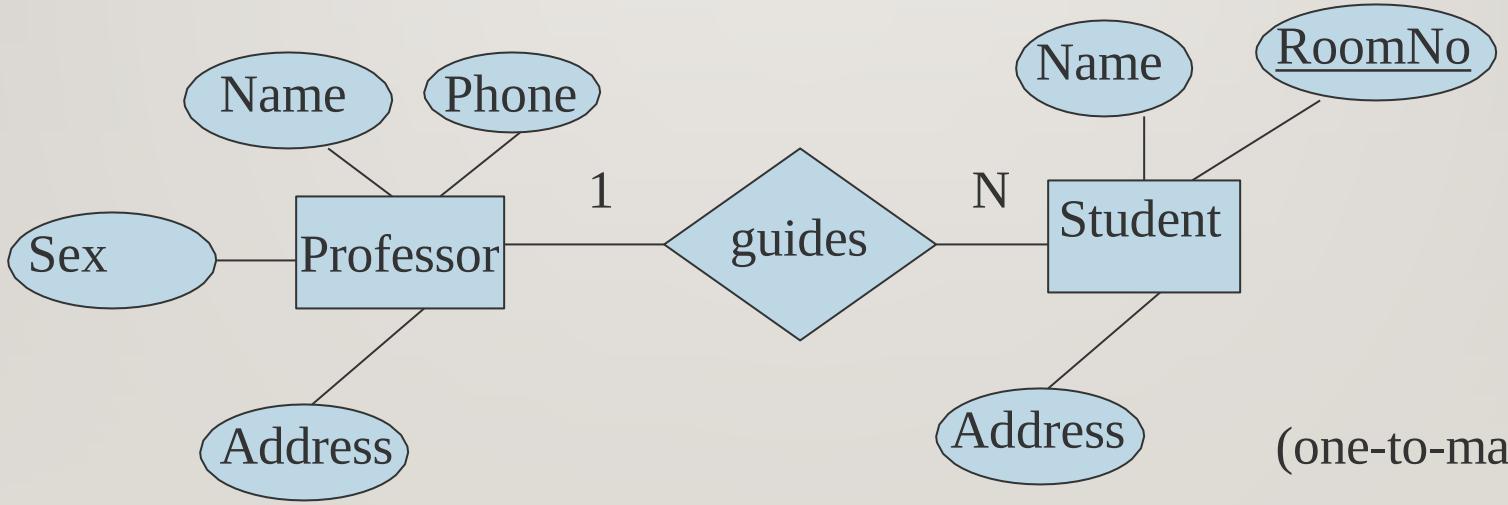
CARDINALITY RATIO – EXAMPLE (*ONE-TO-ONE*)



CARDINALITY RATIO – EXAMPLE (*MANY-TO-ONE/ONE-TO-MANY*)

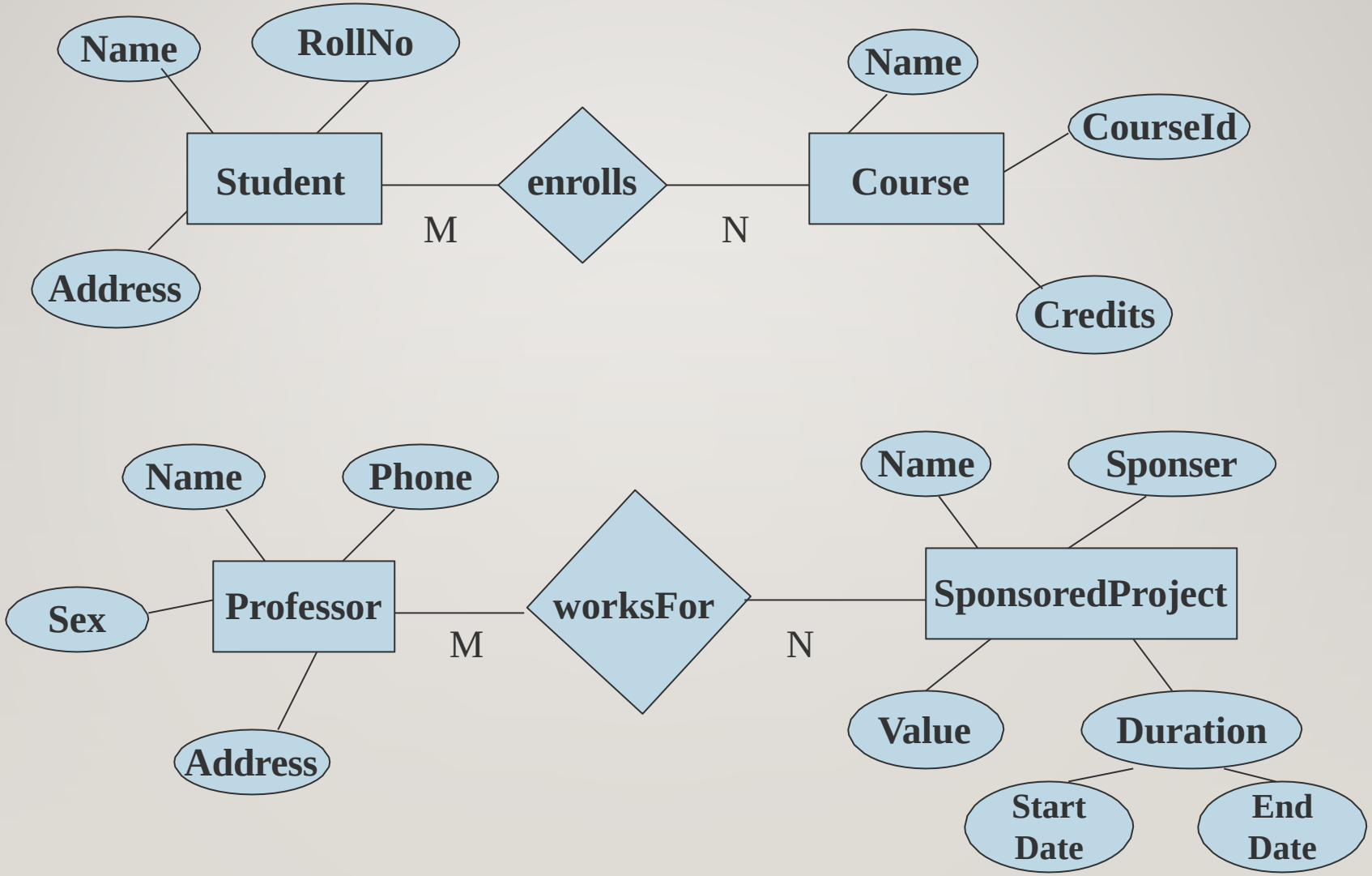


(many-to-one)



(one-to-many)

CARDINALITY RATIO – EXAMPLE (*MANY-TO-MANY*)

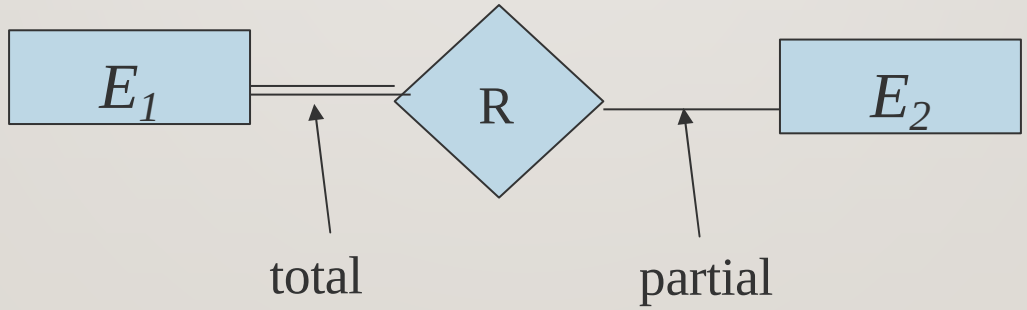


ADDITIONAL FEATURES OF THE ER MODEL

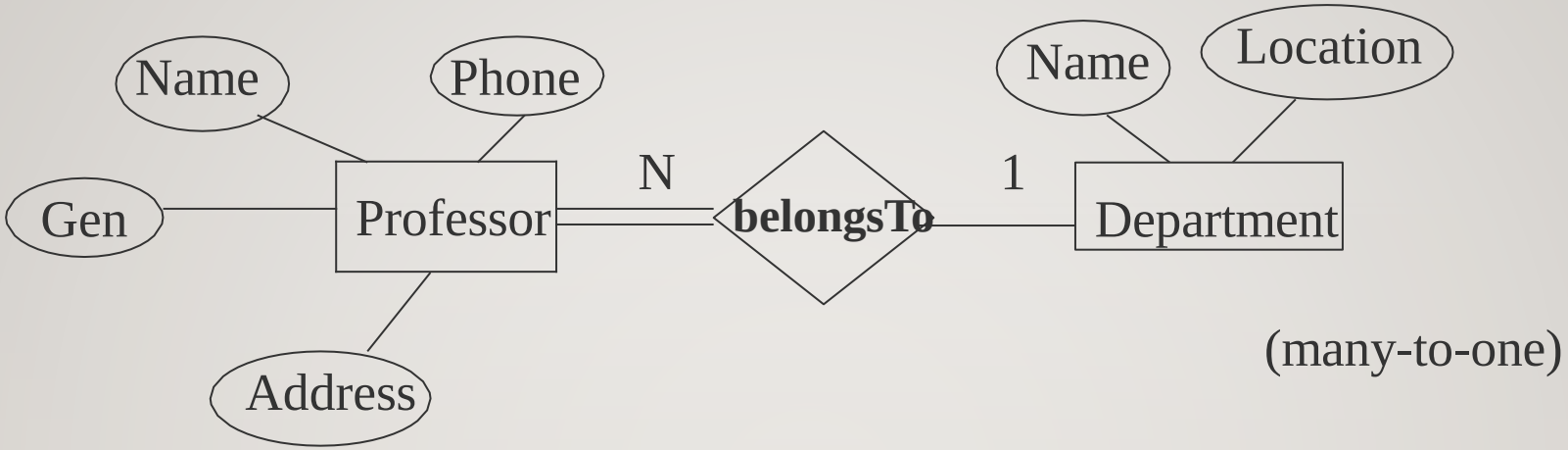
PARTICIPATION CONSTRAINTS

- An entity set may participate in a relation either *totally* or *partially*.
- *Total participation*: Every entity in the set is involved in some association (or tuple) of the relationship.
- *Partial participation*: Not all entities in the set are involved in association (or tuples) of the relationship.

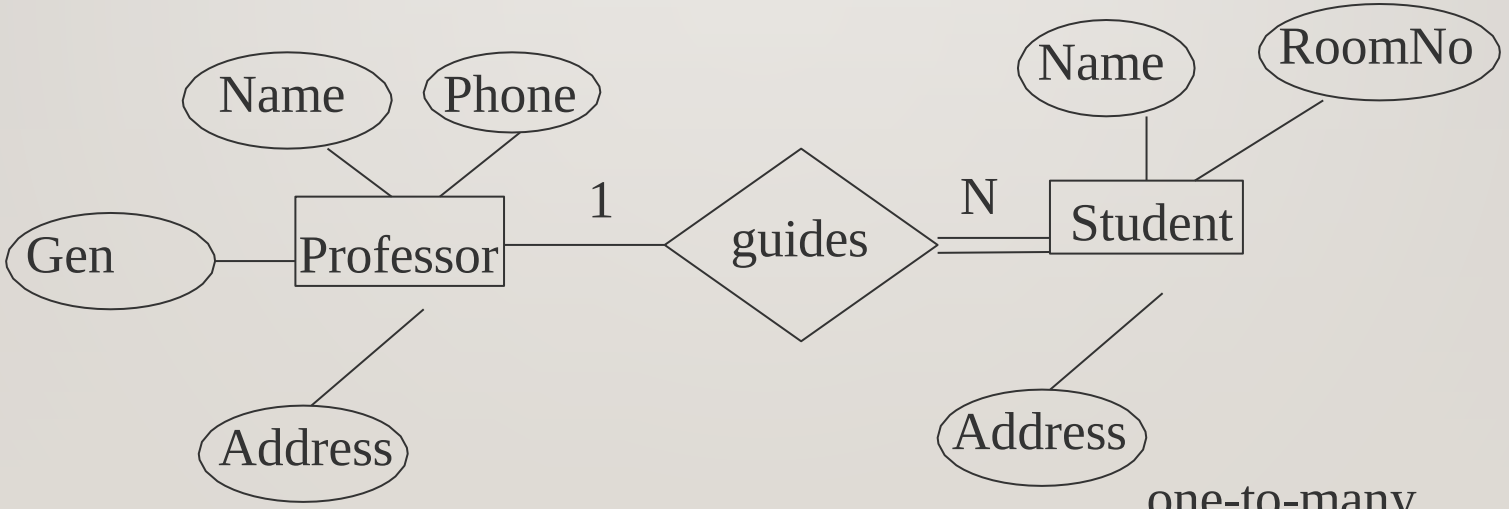
Notation:



EXAMPLE OF TOTAL/PARTIAL PARTICIPATION



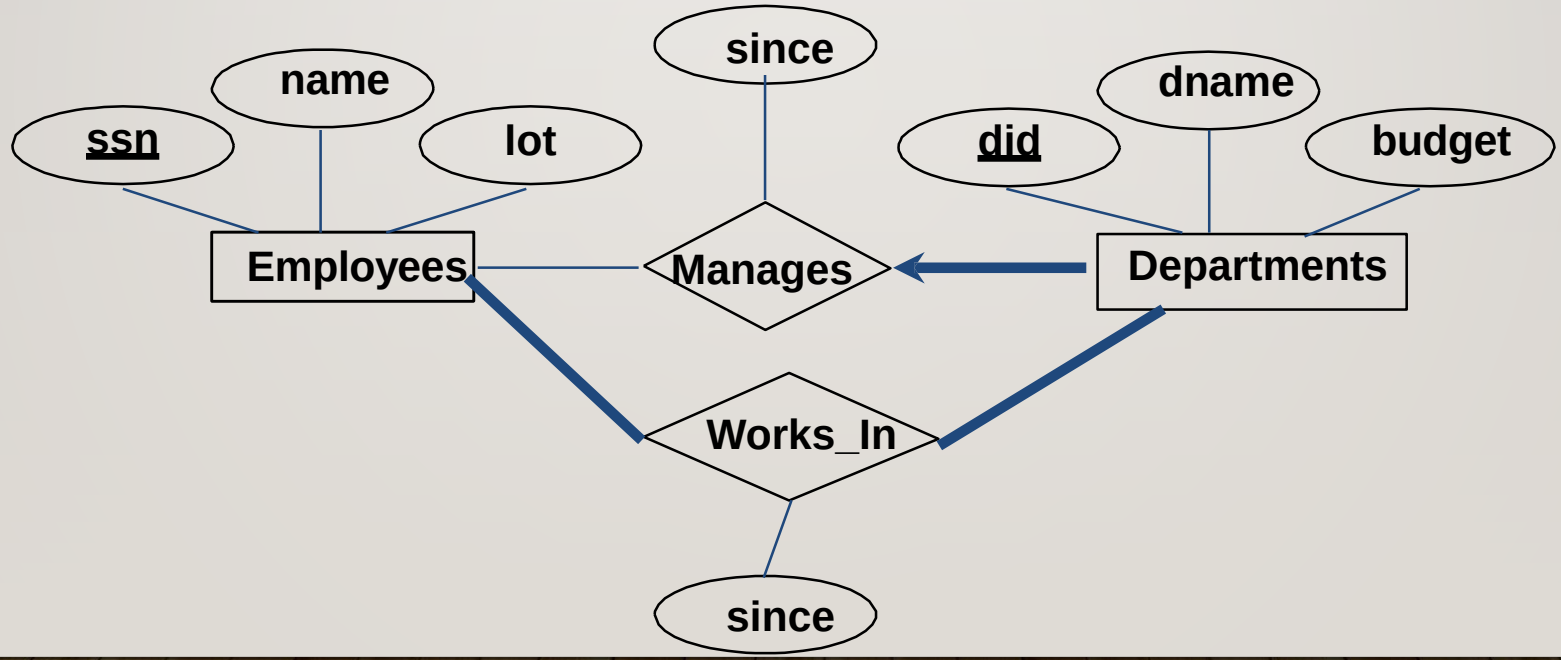
(many-to-one)



one-to-many

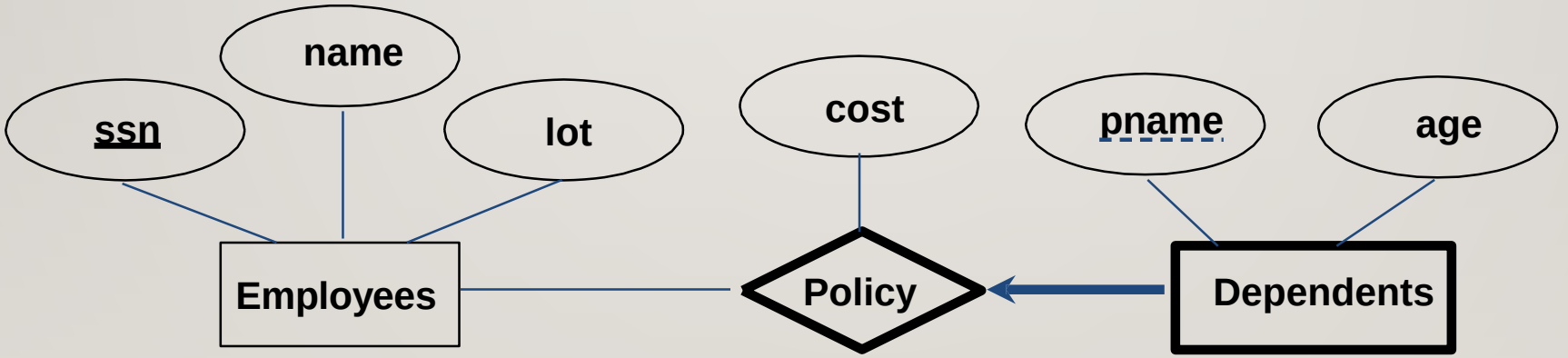
PARTICIPATION CONSTRAINTS

- **Does every department have a manager?**
  - **If so, this is a *participation constraint*: the participation of Departments in Manages is said to be *total* (vs. *partial*).**
    - **Every Departments entity must appear in an instance of the Manages relationship.**



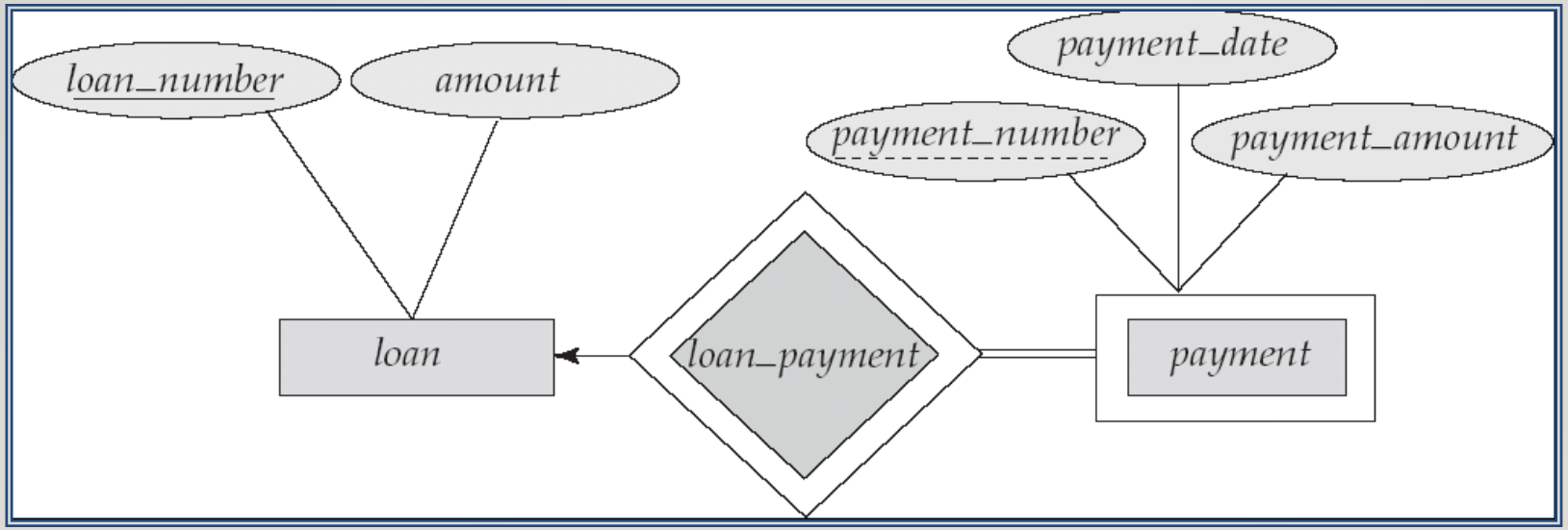
WEAK ENTITIES

- A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.
- **Restrictions**
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities).
  - Weak entity set must have total participation in this *identifying* relationship set.



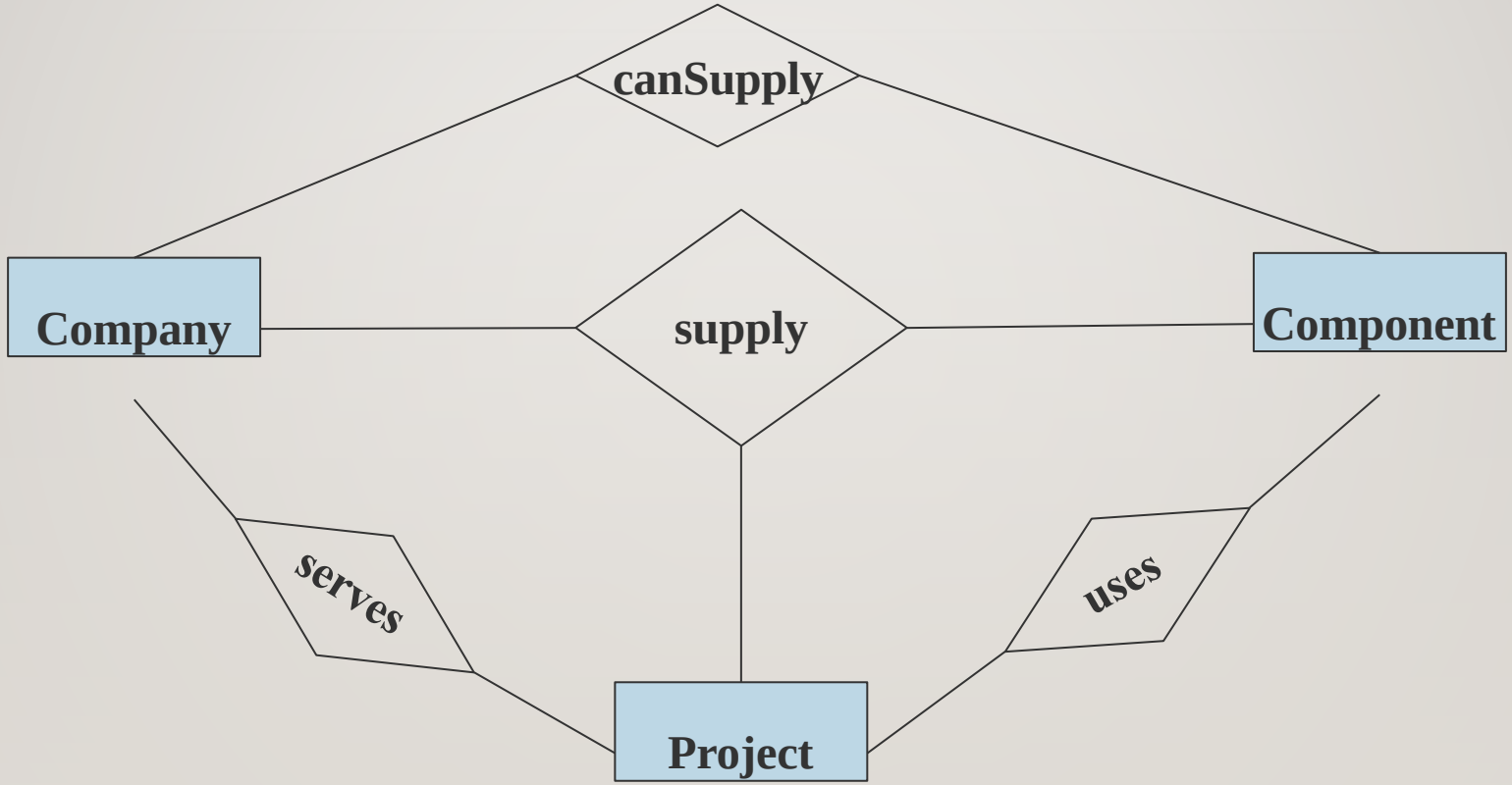
WEAK ENTITY SETS

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



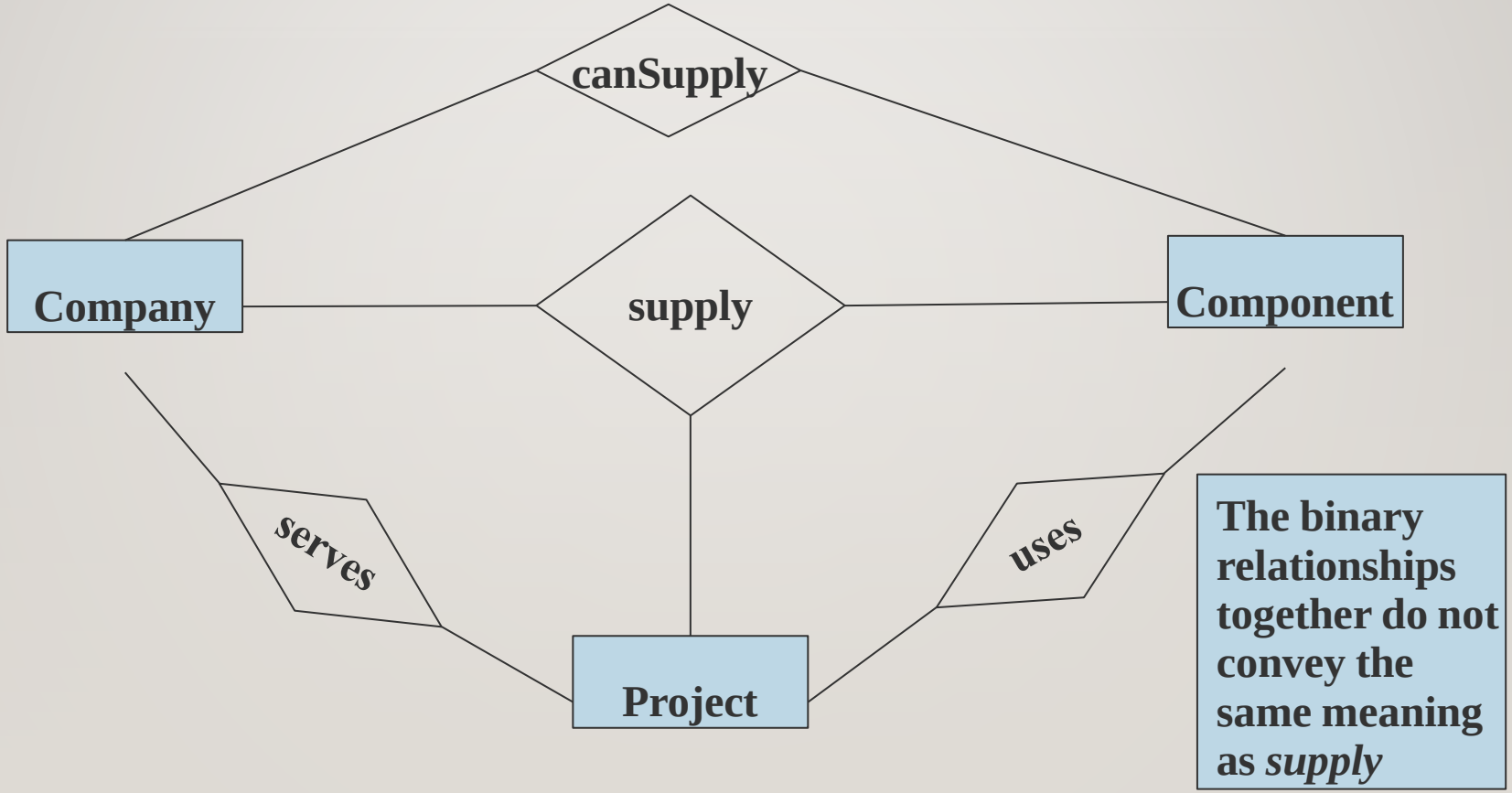
TERNARY RELATIONSHIPS

Relationship instance (c, p, j) indicates that
company c supplies a component p that is made use of by the project j



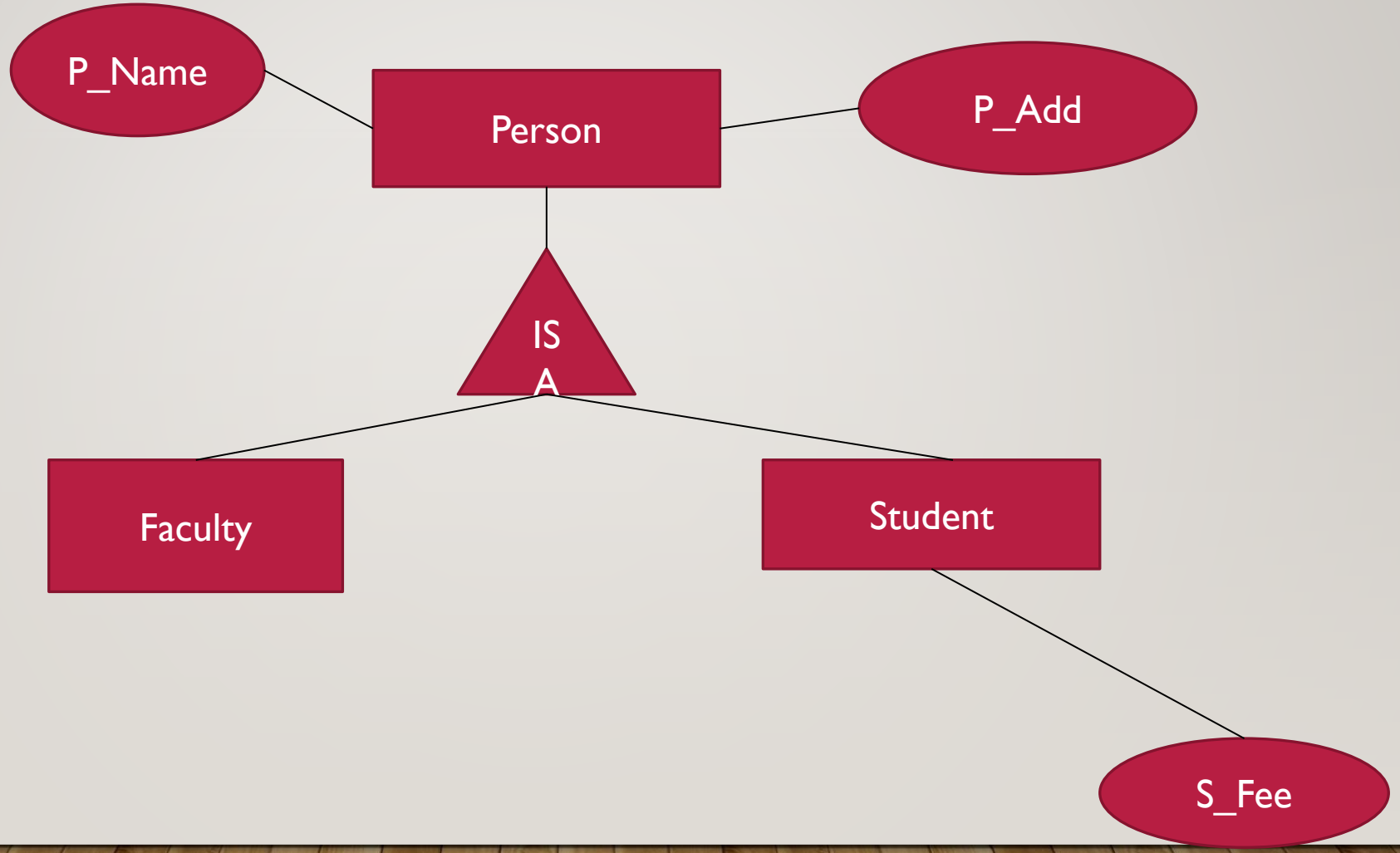
TERNARY RELATIONSHIPS

(c,p) in *canSupply*, (j,p) in *uses*, (c,j) in *serves* may not together imply (c,p,j) is in *supply*. Whereas the other way round is of course true.



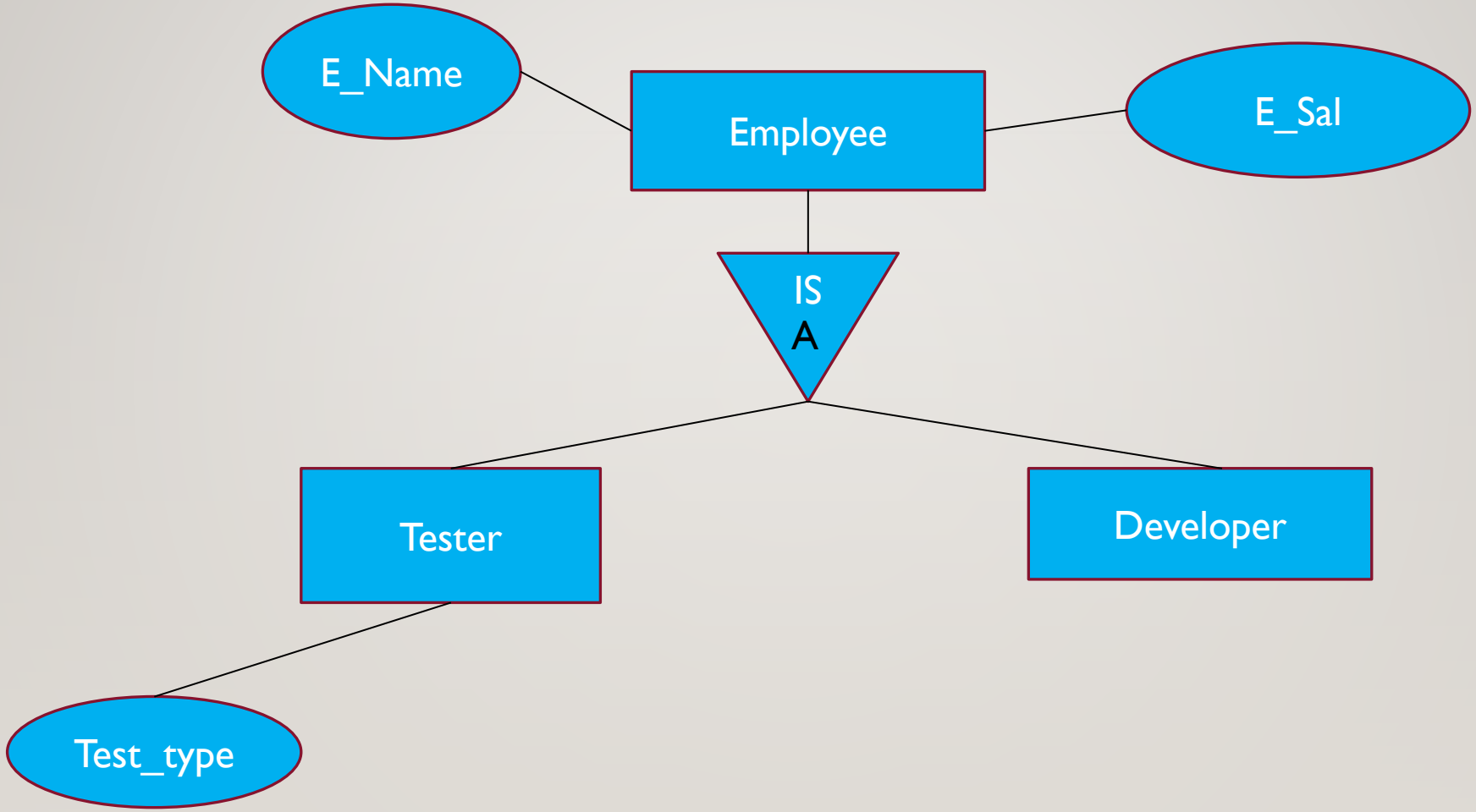
GENERALIZATION

- Generalization is the process of extracting common properties from a set of entities and create a generalized entity from it. It is a bottom-up approach in which two or more entities can be generalized to a higher level entity if they have some attributes in common. For Example, STUDENT and FACULTY can be generalized to a higher level entity called PERSON as shown in Figure 1. In this case, common attributes like P_NAME, P_ADD become part of higher entity (PERSON) and specialized attributes like S_FEE become part of specialized entity (STUDENT).


P_Name
Person
P_Add
IS
A
Faculty
Student
S_Fee

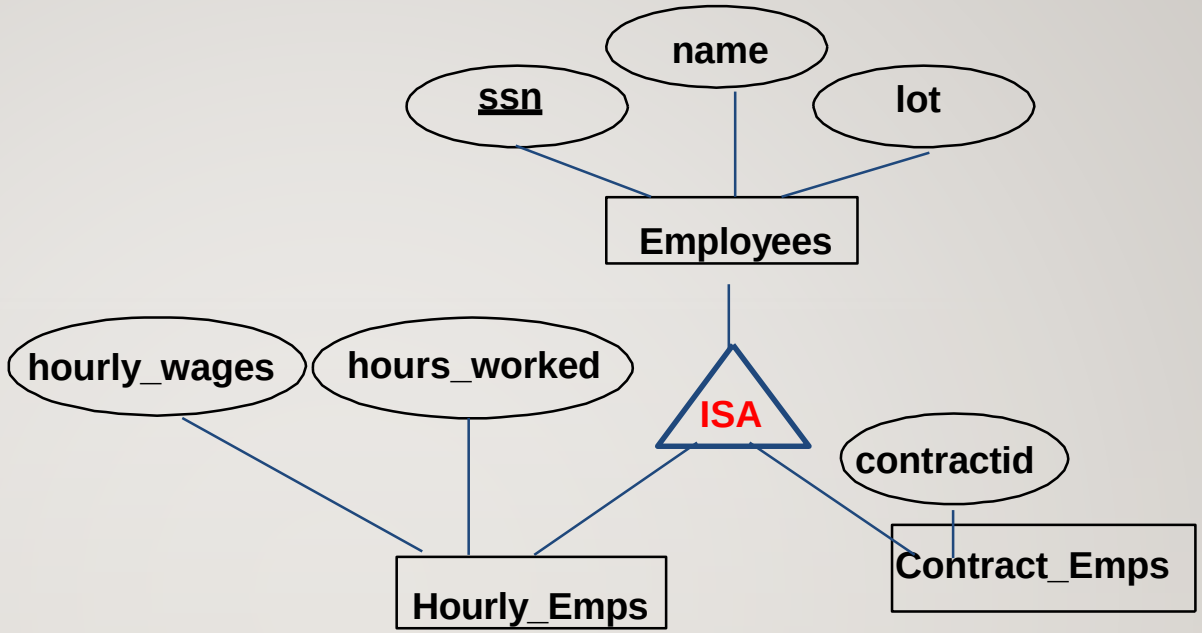
SPECIALIZATION

- In specialization, an entity is divided into sub-entities based on their characteristics. It is a top-down approach where higher level entity is specialized into two or more lower level entities. For Example, EMPLOYEE entity in an Employee management system can be specialized into DEVELOPER, TESTER etc. as shown in Figure 2. In this case, common attributes like E_NAME, E_SAL etc. become part of higher entity (EMPLOYEE) and specialized attributes like TES_TYPE become part of specialized entity (TESTER).


E_Name
Employee
E_Sal
IS
A
Tester
Developer
Test_type

ISA (`IS A') HIERARCHIES

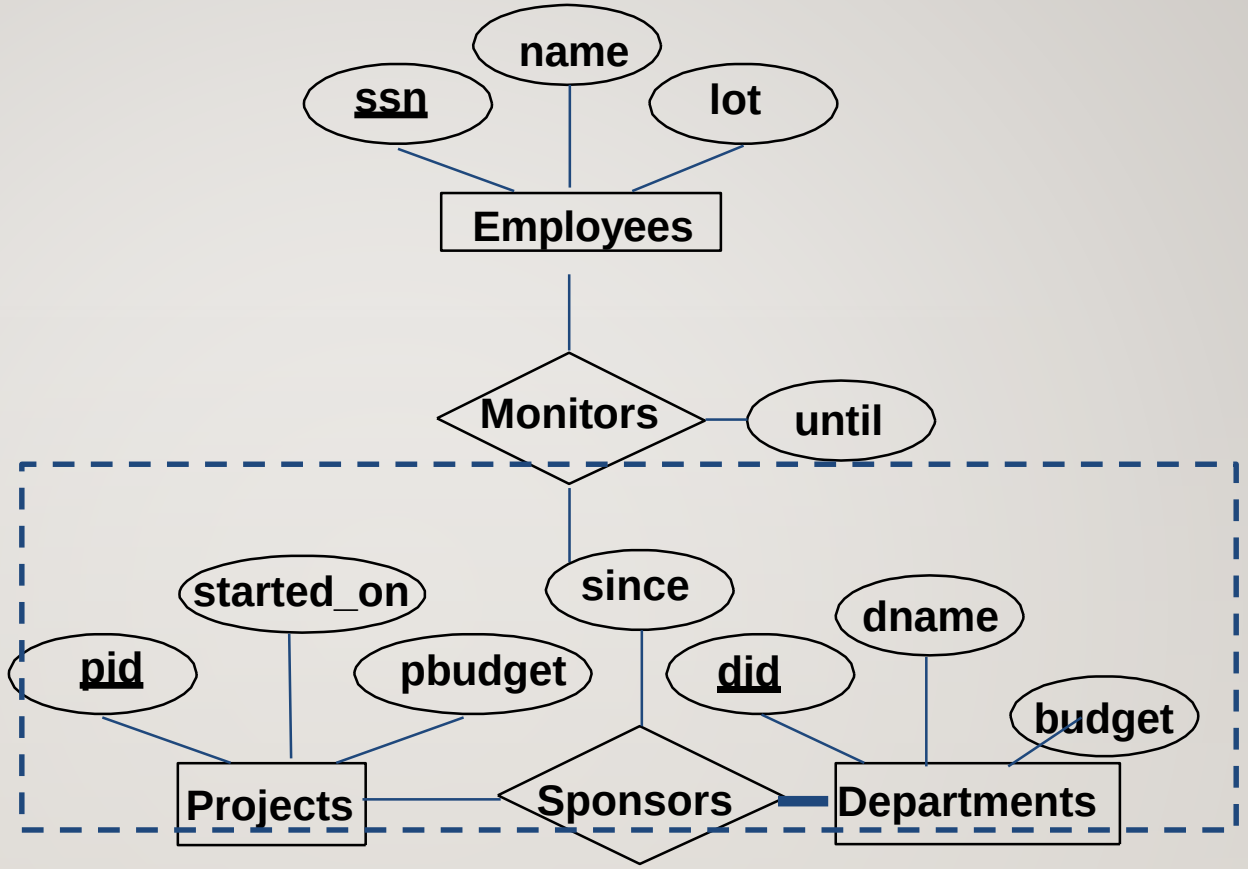
- **If we declare A ISA B, every A entity is also considered to be a B entity.**



- ***Overlap constraints*: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? *(Allowed/disallowed)***
- ***Covering constraints*: Does every Employees entity also have to be an Hourly_Emps or a Contract_Emps entity? *(Yes/no)***
- **Reasons for using ISA:**
  - **To add descriptive attributes specific to a subclass.**
  - **To identify entities that participate in a relationship.**

AGGREGATION

- **Used when we have to model a relationship involving (entity sets and) a *relationship set*.**
  - ***Aggregation*** **allows us to treat a relationship set as an entity set for purposes of participation in (other) relationships.**



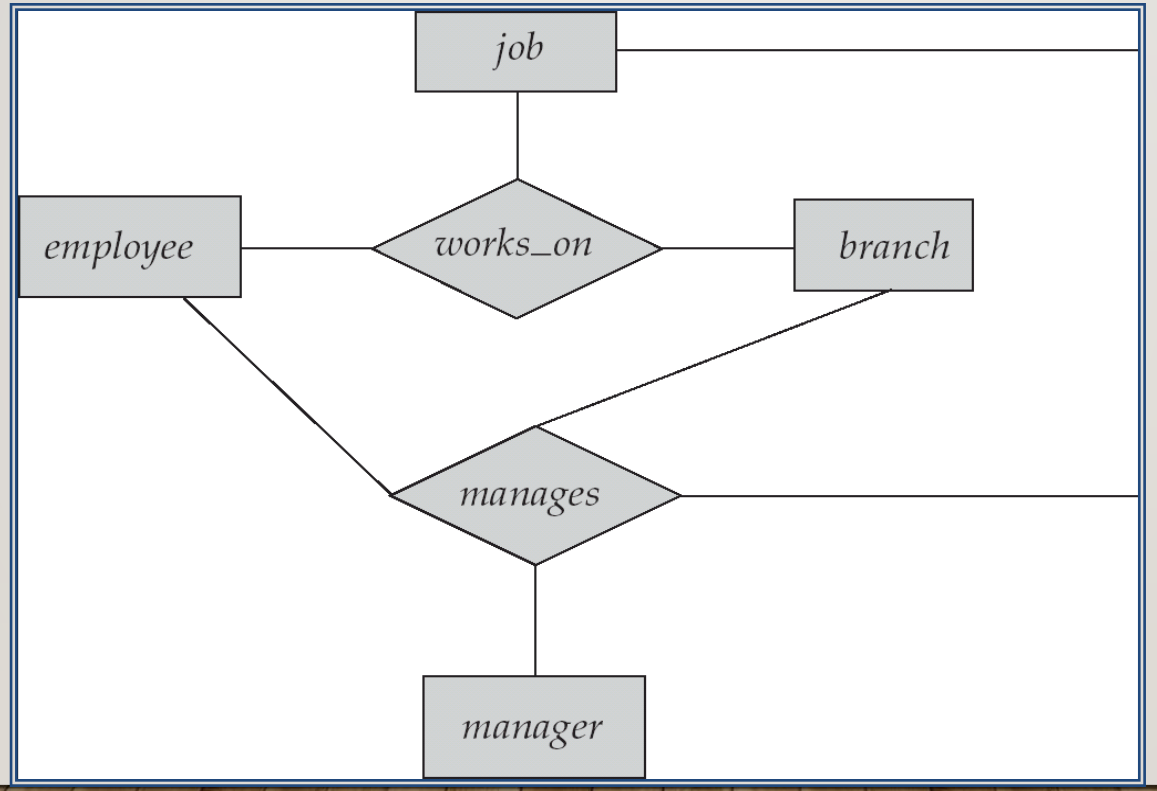
✉ ***Aggregation vs. ternary relationship*:**

❖ **Monitors is a distinct relationship, with a descriptive attribute.**

❖**Also, can say that each sponsorship is monitored by at most one employee.**

AGGREGATION

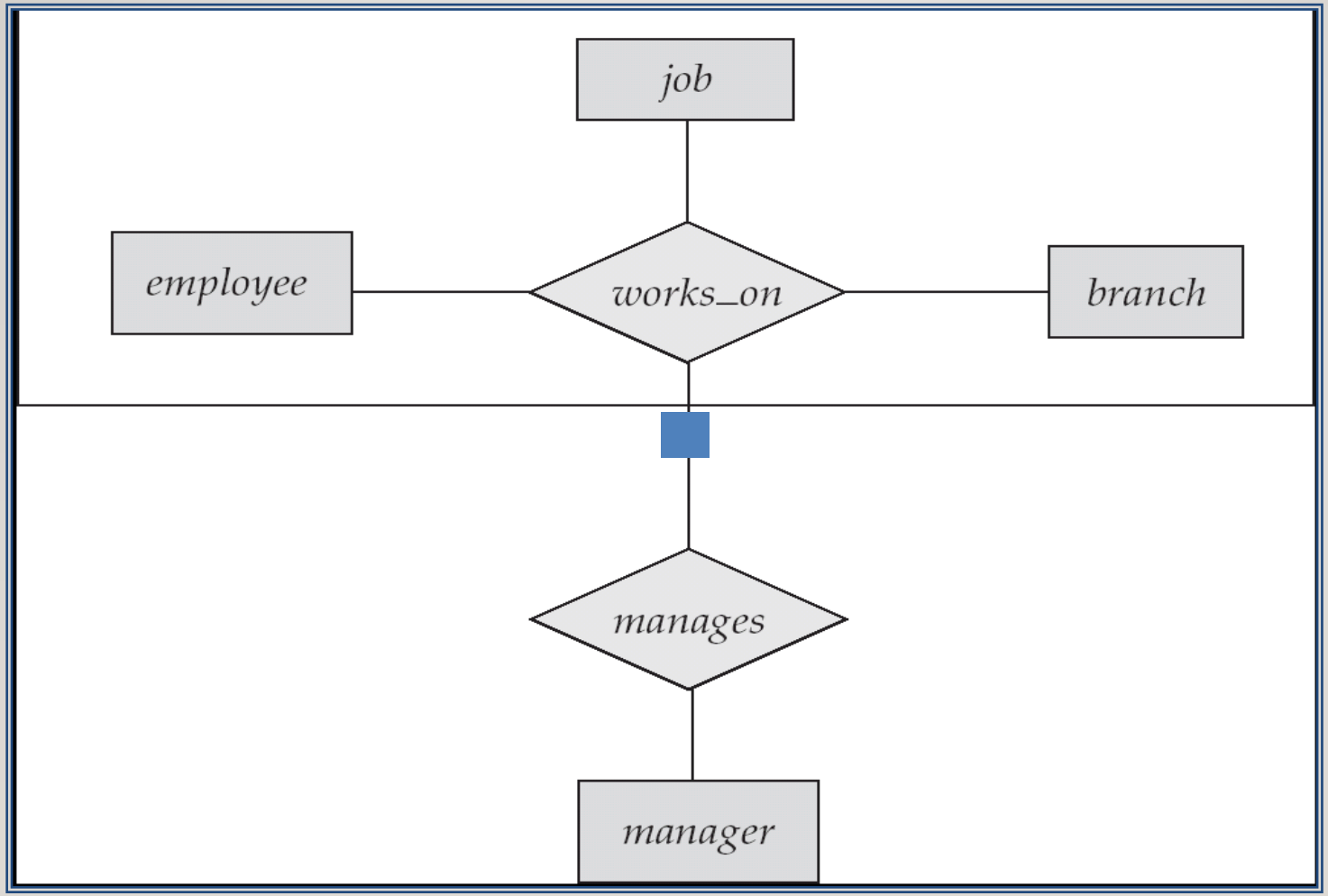
- **Consider the ternary relationship *works_on*, which we saw earlier**
- **Suppose we want to record managers for tasks performed by an employee at a branch**



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**

E-R DIAGRAM WITH AGGREGATION



CONCEPTUAL DESIGN USING THE ER MODEL

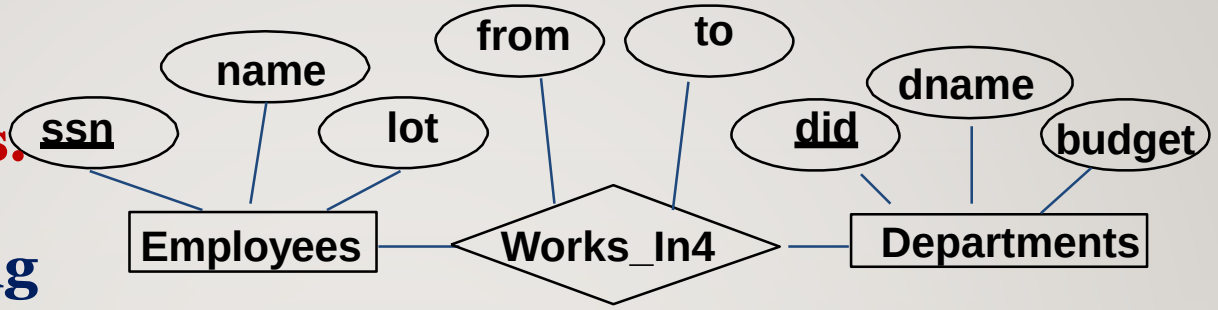
- **Design choices:**
  - **Should a concept be modeled as an entity or an attribute?**
  - **Should a concept be modeled as an entity or a relationship?**
  - **Identifying relationships: Binary or ternary? Aggregation?**
- **Constraints in the ER Model:**
  - **A lot of data semantics can (and should) be captured.**
  - **But some constraints cannot be captured in ER diagrams.**

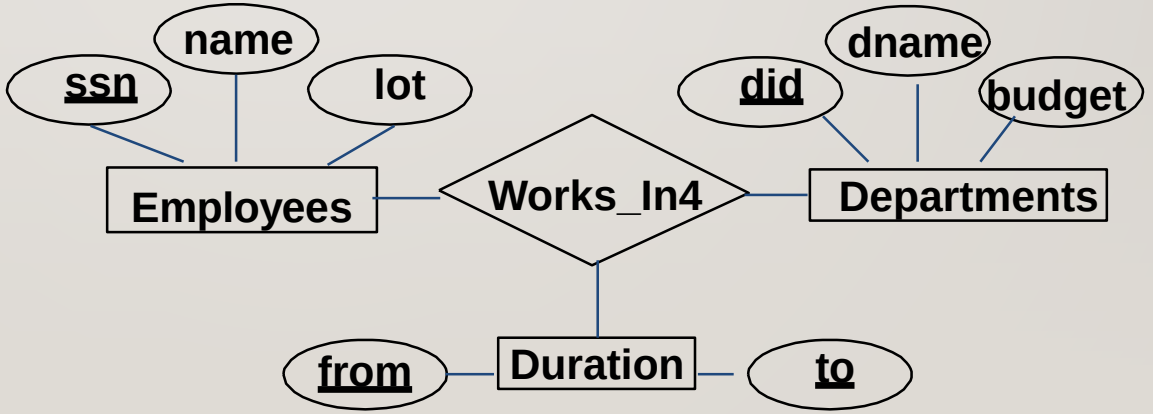
ENTITY VS. ATTRIBUTE

- **Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?**
- **Depends upon the use we want to make of address information, and the semantics of the data:**
  - **If we have several addresses per employee, *address* must be an entity (since attributes cannot be set-valued).**
  - **If the structure (city, street, etc.) is important, e.g., we want to retrieve employees in a given city, *address* must be modeled as an entity (since attribute values are atomic).**

ENTITY VS. ATTRIBUTE (CONTD.)

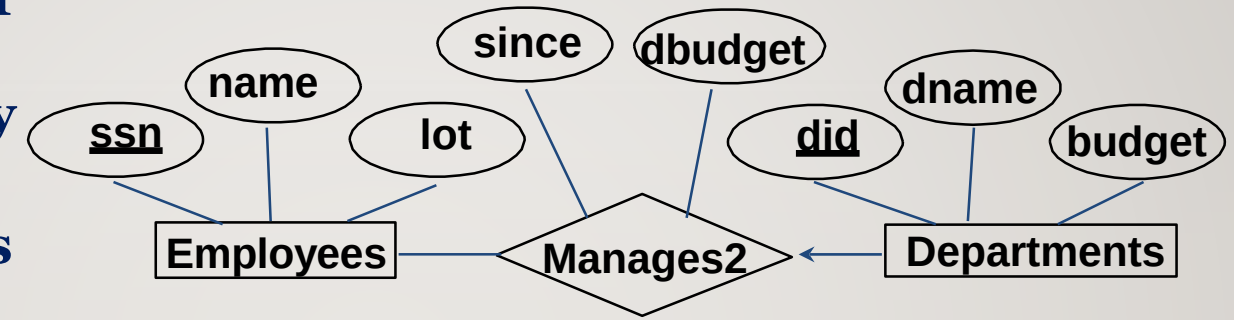
- **Works_In4 does not allow an employee to work in a department for two or more periods.**
- **Similar to the problem of wanting to record several addresses for an employee: We want to record *several values of the descriptive attributes for each instance of this relationship*. Accomplished by introducing new entity set, Duration.**

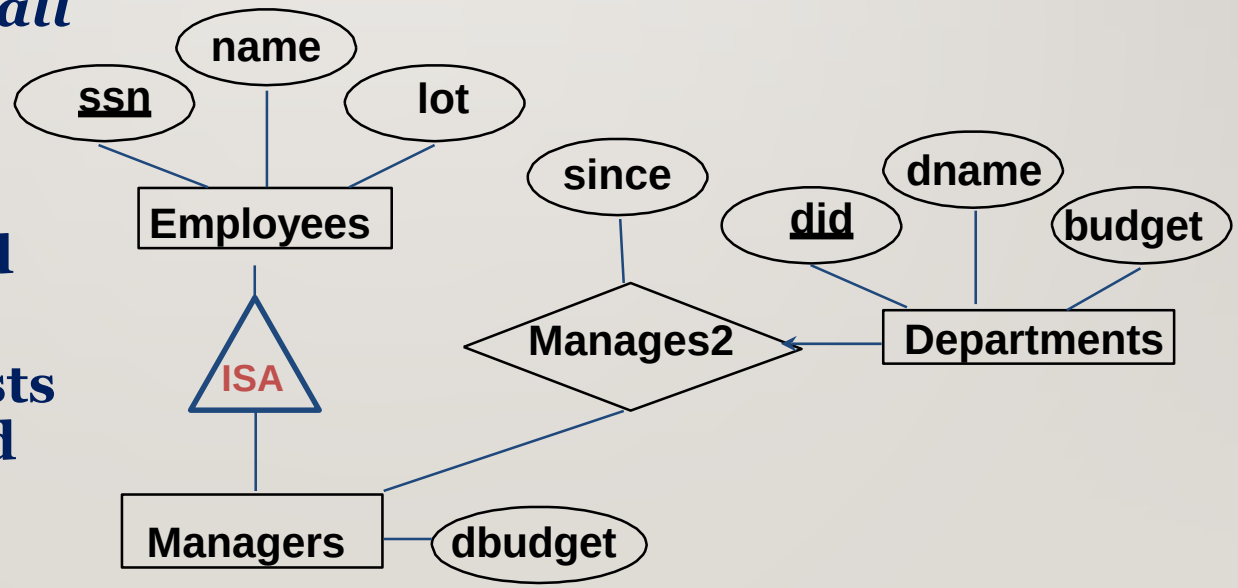




ENTITY VS. RELATIONSHIP

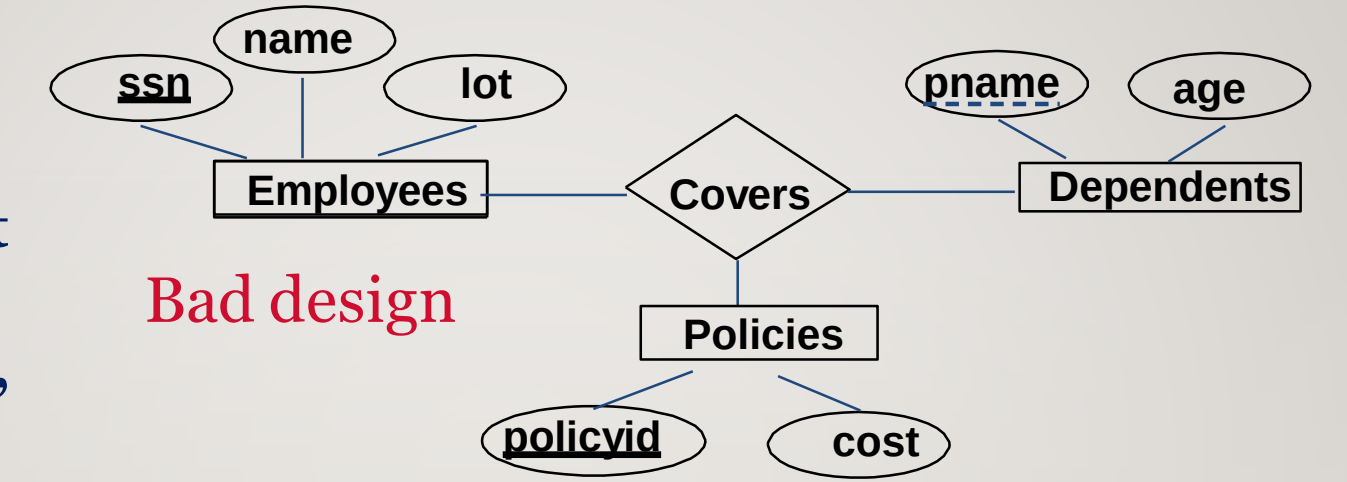
- **First ER diagram OK if a manager gets a separate discretionary budget for each dept.**
- **What if a manager gets a discretionary budget that covers *all* managed depts?**
  - **Redundancy: *dbudget* stored for each dept managed by manager.**
  - **Misleading: Suggests *dbudget* associated with department-mgr combination.**



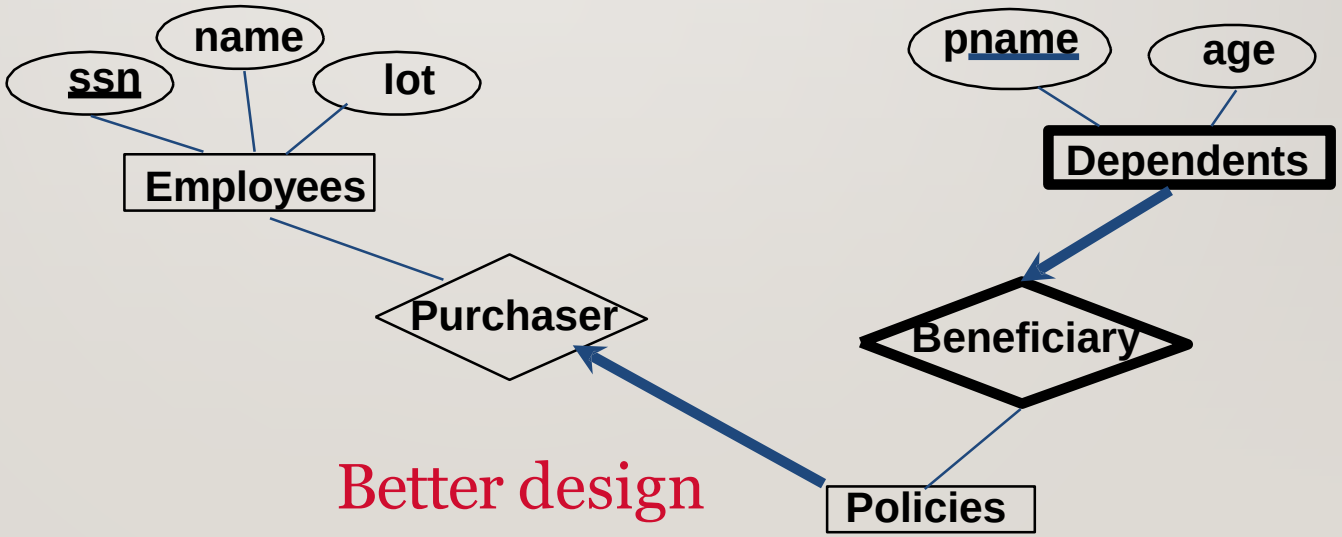


BINARY VS. TERNARY RELATIONSHIPS

- **If each policy is owned by just 1 employee, and each dependent is tied to the covering policy, first diagram is inaccurate.**
- **What are the additional constraints in the 2nd diagram?**



Bad design



Better design

BINARY VS. TERNARY RELATIONSHIPS (CONTD.)

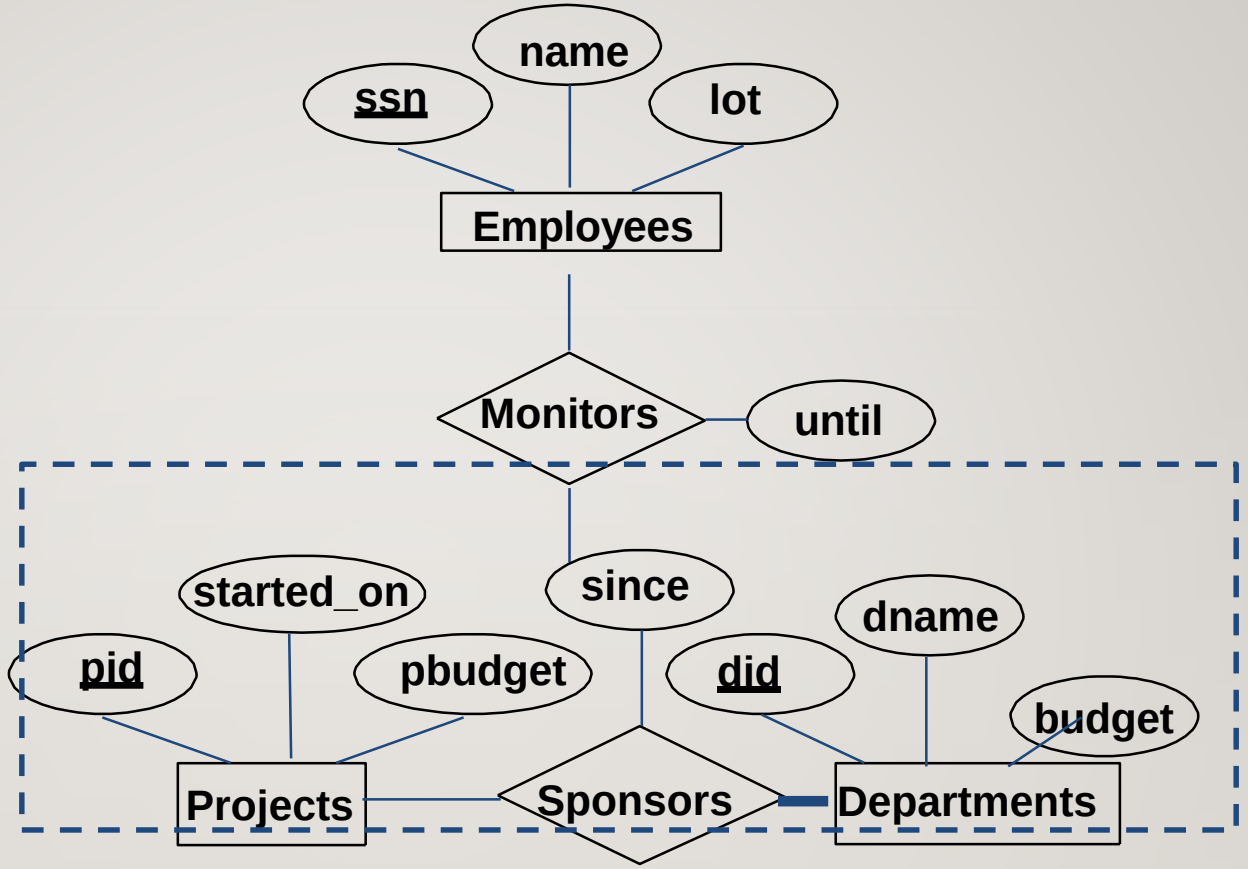
- **Previous example illustrated a case when two binary relationships were better than one ternary relationship.**

- **An example in the other direction: a ternary relation Contracts relates entity sets Parts, Departments and Suppliers, and has descriptive attribute *qty*.**

  - **S "can-supply" P, D "needs" P, and D "deals-with" S does not imply that D has agreed to buy P from S.**
  - **How do we record *qty*?**

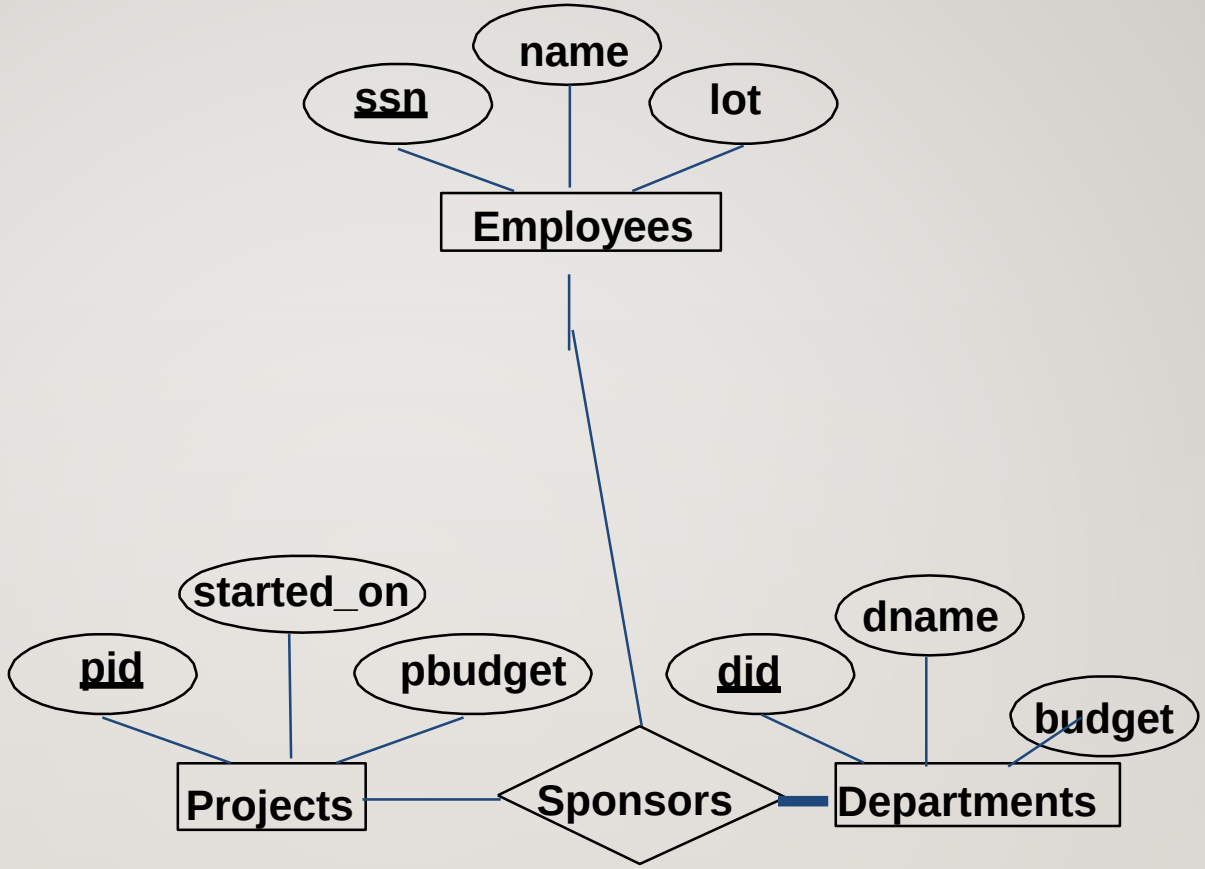
AGGREGATION V/S TERNARY RELATIONSHIP

- The choice between using aggregation or ternary relationship is mainly determined by existence of a relationship that relates relationship set to entity set.



The choice may also be guided by certain integrity constraints that we want to express.

AGGREGATION V/S TERNARY RELATIONSHIP



- **Using ternary relationship instead of aggregation**

CONCEPTUAL DESIGN FOR LARGE ENTERPRISES

- **For large enterprise the design may require efforts of more than one designer and span data and application code used by number of user groups.**
- **ER diagrams for Conceptual design offers additional advantage that high level design can be diagramatically represented and easily understood by many people.**

2 approaches:

- **Usual approach: requirements of various user groups are considered,any conflicting requirements are somehow resolved and single set of global requirements is generated at the end of requirements phase**
- **Alternative approach: is to develop separate conceptual schemas for different user groups and then integrate these conceptual schemas**

RELATIONAL DATABASE: DEFINITIONS

- ***Relational database*: a set of *relations***
- ***Relation:* made up of 2 parts:**
- ***Relation schema and relational instance*.**
  - ***Instance* : a *table*, with rows and columns.**
  - **Set of tuples also called as records**
  - **#Rows = *cardinality*, #fields = *degree / arity*.**
  - **A domain is referred by domain name consisting of set of associated values.**
  - ***Schema* : specifies name of relation, plus name and type of each column.**
    - **E.G. Students (*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real).**
- **Can think of a relation as a *set* of rows or *tuples* (i.e., all rows are distinct).**