# SE240: Introduction to Database Systems

Lecture 02:

ER Diagram

# Outline

- **Database Design**
- Entity
- Relationship
  - Binary relationship
  - Non-binary relationship
- Weak Entity/Strong Entity
- Class Hierarchy
- Aggregation
- Conceptual Design with the E-R Model

# Database Design

- The database design process can be divided into six steps.

1. Requirement Analysis
   - What data is to be stored?
   - What applications must be built.
   - What operations are the most frequent and subject to performance requirements.

# Database Design

2. Conceptual database design
   - High-level description of
     - Data to be stored
     - Constraints
   - Often carried out using the ER model
3. Logical database design
   - Choose a DBMS to implement our database design.
   - Convert the conceptual database design into a database schema in the data model of the chosen DBMS.
     - Convert ER schema into a relational database schema.

# Database Design

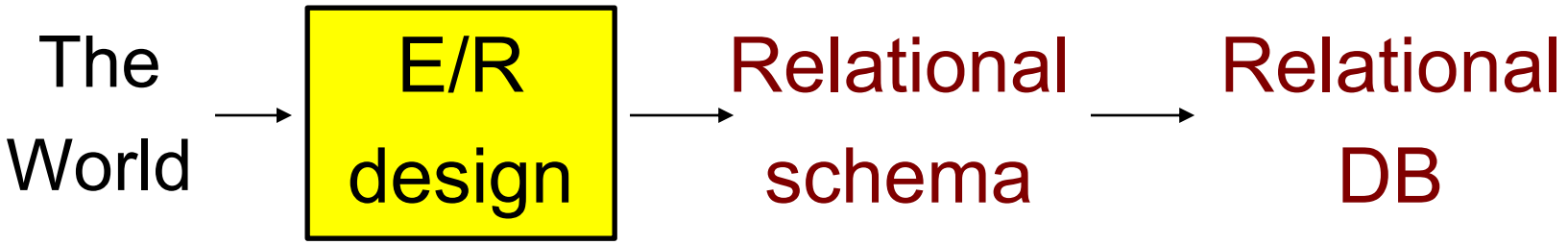
4. Schema Refinement
   - Analyze the collection of relations in our relational database.
   - Identify the potential problems.
   - Refine the schema

5. Physical database design
   - Ensure that the design meets the performance requirement.
   - Build index, clustering some tables etc.
   - Redesign parts of the database schema.

# Database Design

6. Application and security design
   - Write application programs.
   - Identify data that can be accessible by certain types of users.
   - Take steps to ensure that access rules are enforced.

# DB Development Path



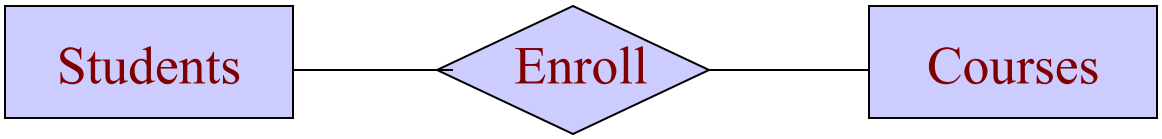

1. Requirement analysis
2. Conceptual database design – ER
3. Logical database design – relational schema
4. Schema refinement
5. Physical database design
6. Application and security design

# Entity-Relationship Model

- **Entity-Relationship (ER) model** is a popular conceptual data model.
- This model is used in the design of database applications.
- The model describes ***data*** to be stored and the ***constraints*** over the data.
- E-R model views the real world as a collection of **entities** and **relationships** among entities.

# Entity-Relationship Model

- **E/R design** translated to a relational design then implemented in an RDBMS
- Elements of model
  - Entities, Entity Sets, Attributes
  - Relationships **(!= relations!)**
- Graphical representation of miniworld
  - Entity: like an object, represented by a rectangle
  - Relationship: connect two or more entity sets, represented by diamonds



# Outline

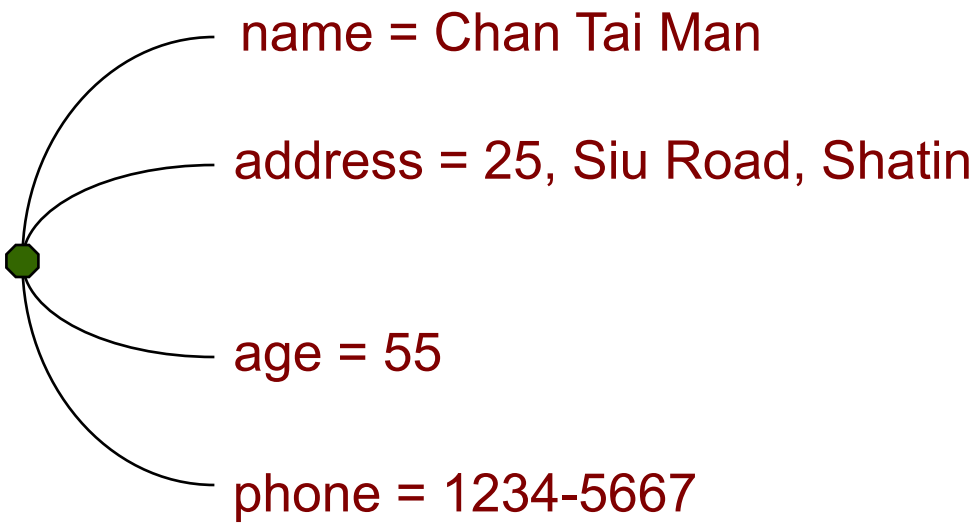
- Database Design
- **Entity**
- Relationship
  - Binary relationship
  - Non-binary relationship
- Weak Entity/Strong Entity
- Class Hierarchy
- Aggregation
- Conceptual Design with the E-R Model

# Entities and Attributes

- E-R model views the real world as a collection of entities and relationships among entities.
- An entity is an object in the real world that is distinguishable from other objects.
  - Examples
    - A classroom
    - A teacher
    - The address of the teacher

# Entities and Attributes

- An entity is described using a set of attributes whose values are used to distinguish one entity from another of the same type
- Entity - Employee



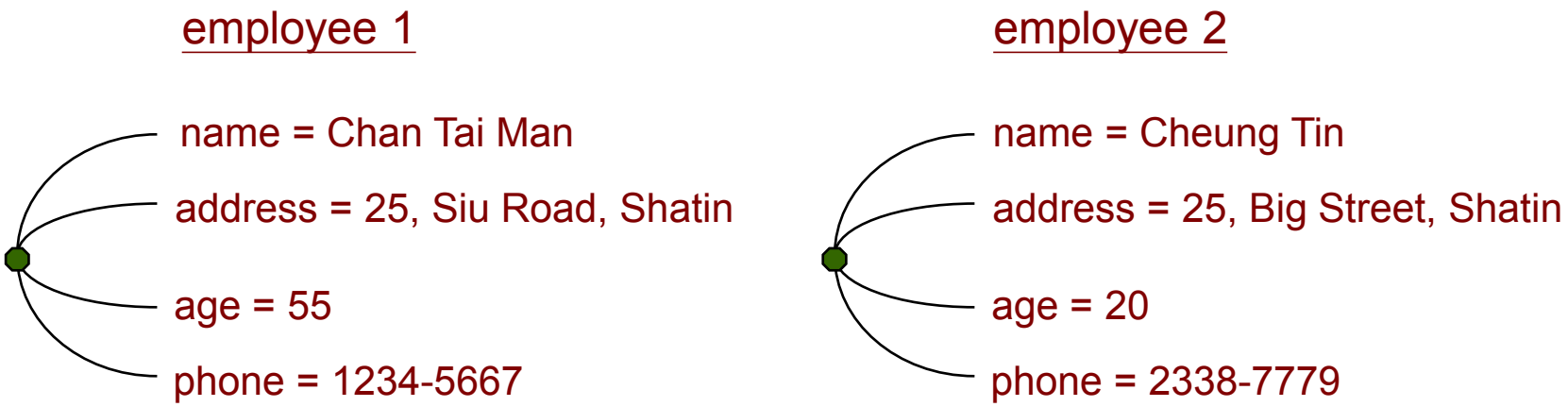
# Entities and Attributes

- An entity set is a collection of entities of the same type.

# Entities and Attributes

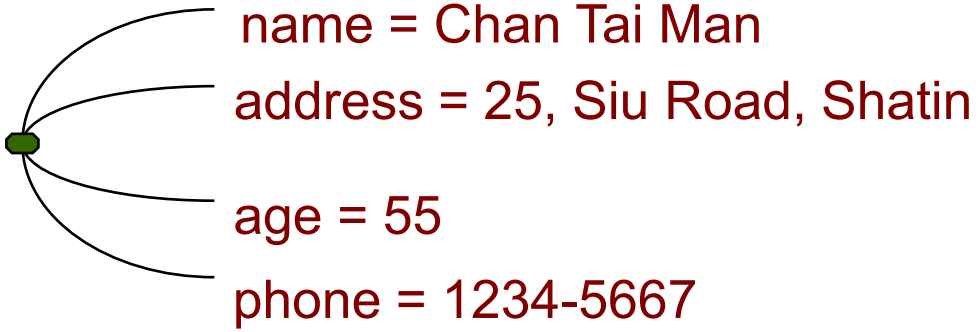
- All entities in a given entity set have the same attributes (the values may be different).

employee = (name, address, age, phone)



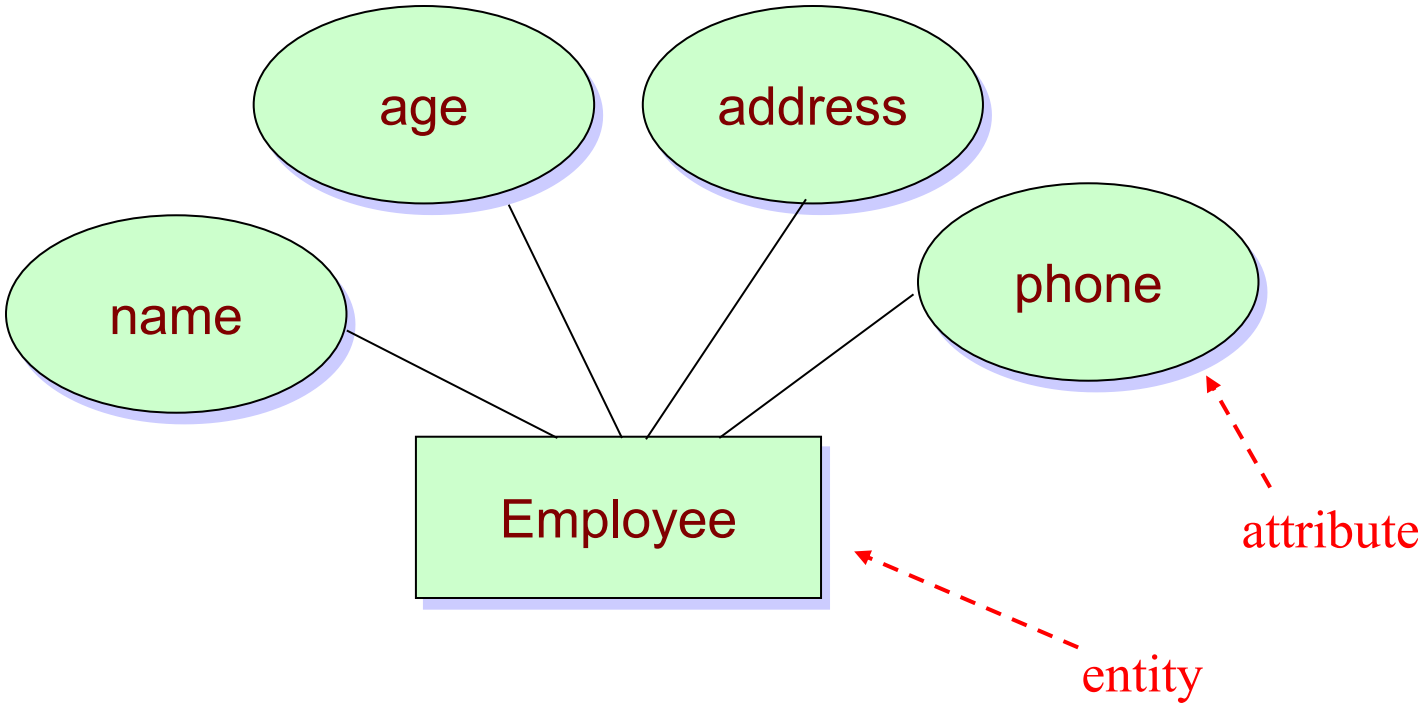
# Entities and Attributes

- For each attribute associated with an entity set, we identify a domain of possible values.
- Example
  - The domain associated with the attribute name might be the set of 20-character *strings*.
  - The domain associated with the attribute age might be an *integer*.



# ER Diagram

- The ER model can be presented graphically by an ER diagram

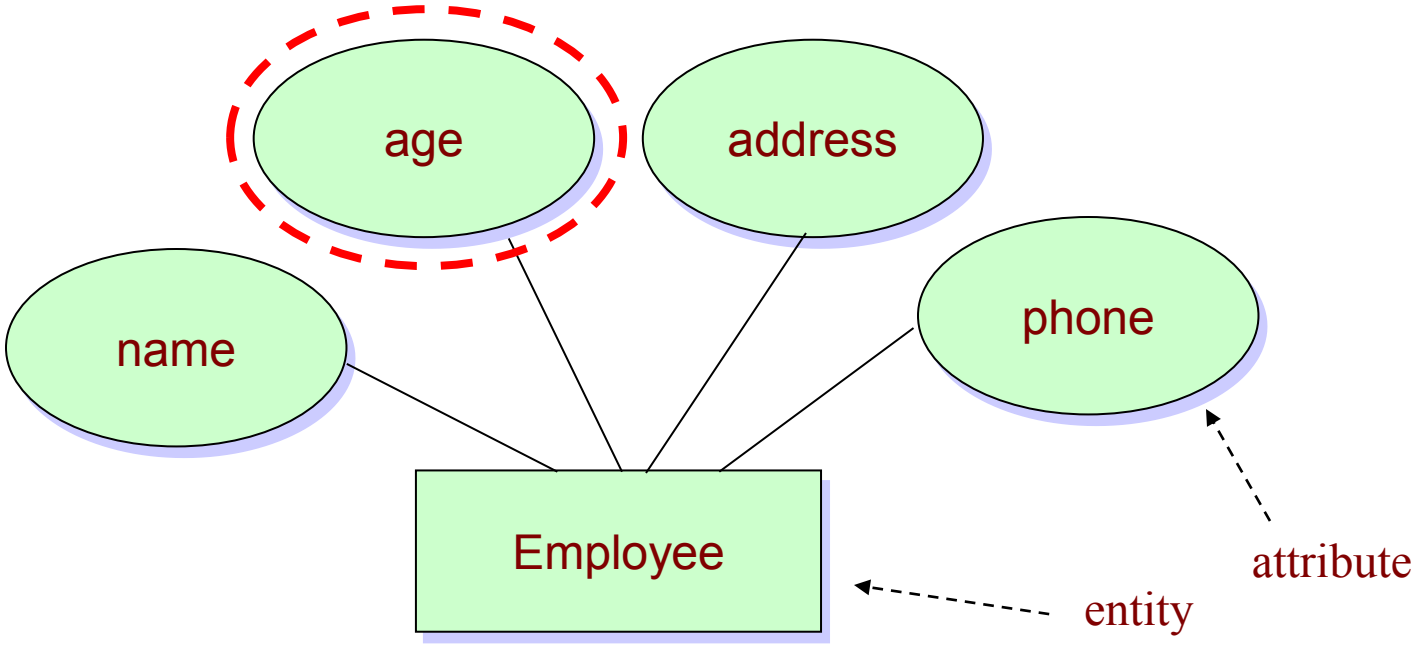


# Different Attribute Types

- Simple attribute
- Composite attribute
- Multi-valued attribute
- Derived attribute

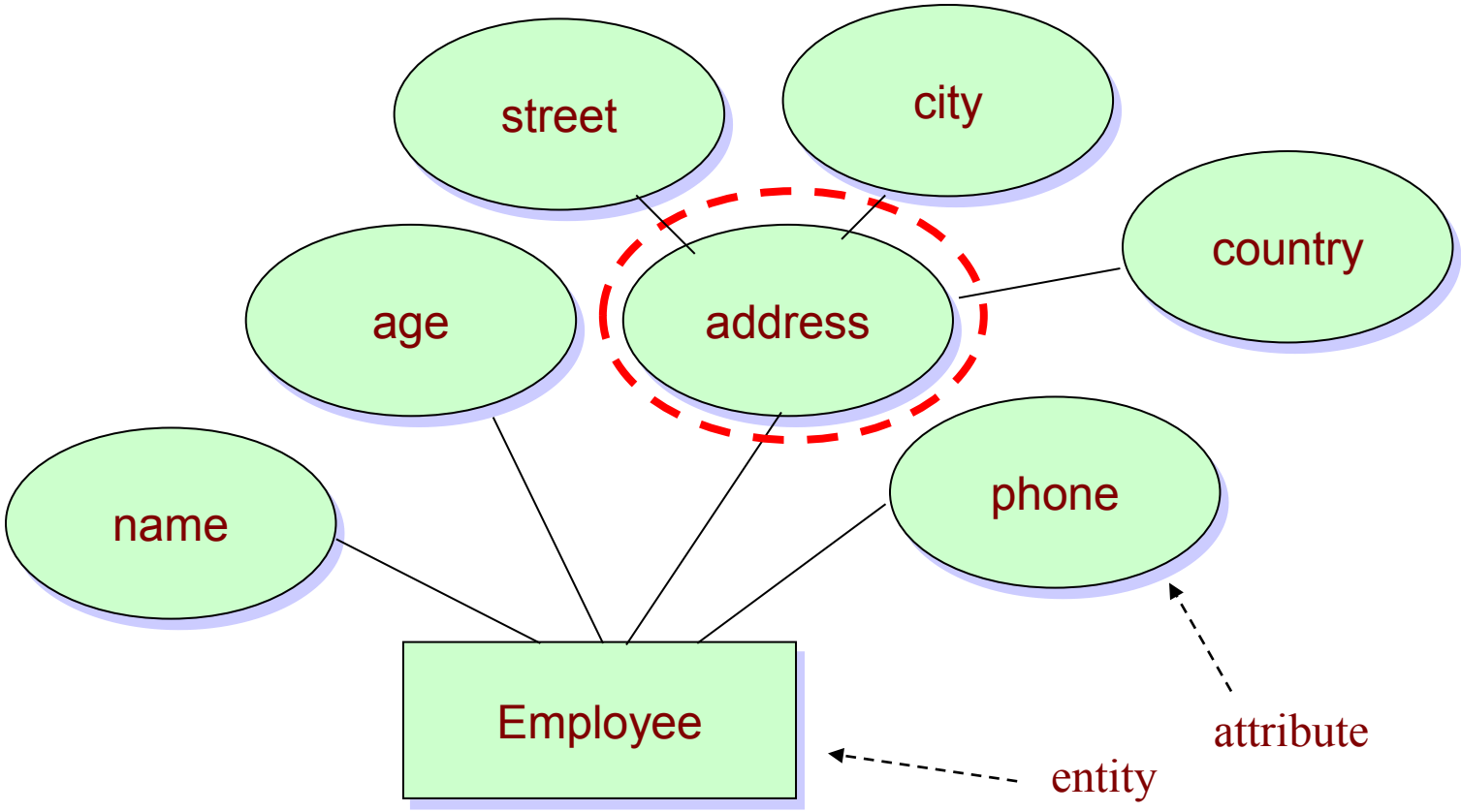
# Simple attribute

- **Simple attribute**
  - contains a single value



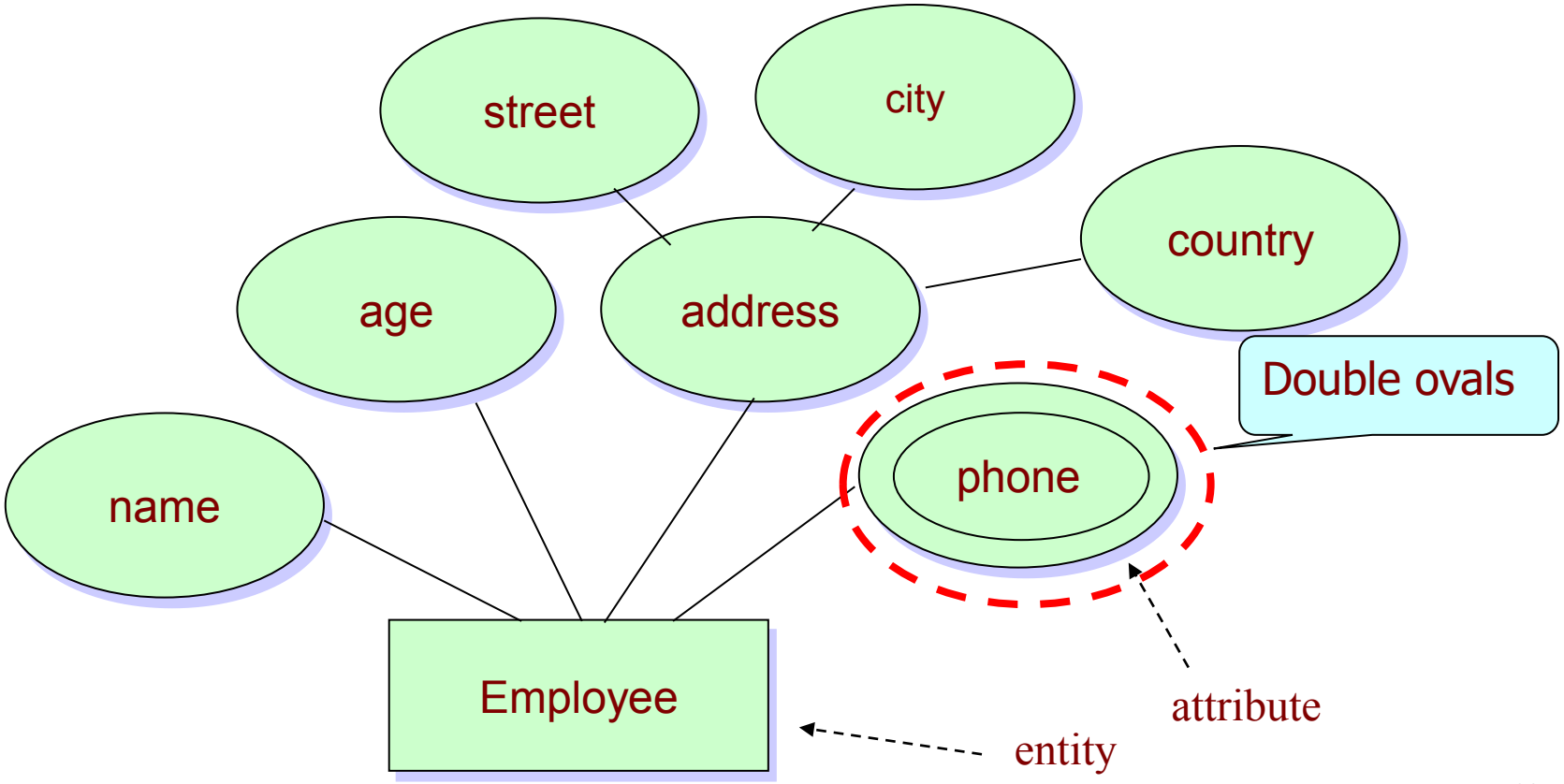
# Composite attribute

- **Composite attribute**
  - Contains several components



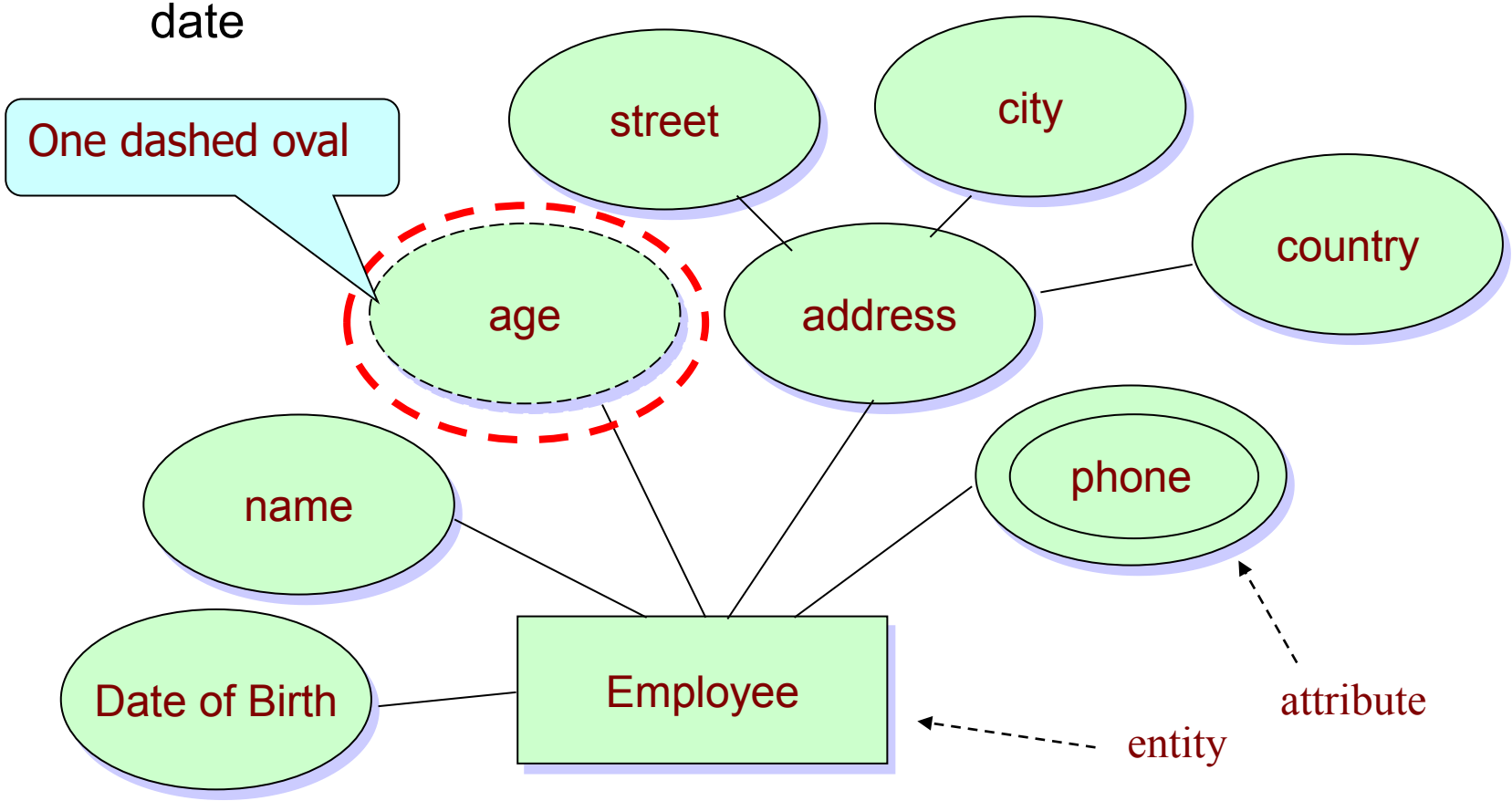
# Multi-valued attribute

- **Multi-valued attribute**
  - Contains more than one value



# Derived attribute

- **Derived attribute**
  - Computed from other attributes
  - e.g., age can be computed from date of birth and the current date



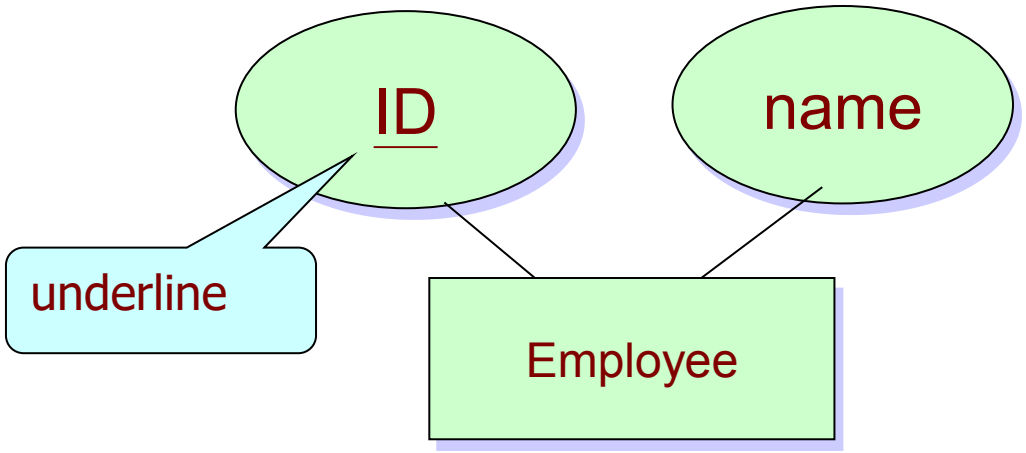
# Key Attributes

- A key has only one meaning
  - It is a set of one or more attributes whose combined values are unique among all occurrences in a given entity set.
  - A key means of specifying ***uniqueness***.

# Key Attributes

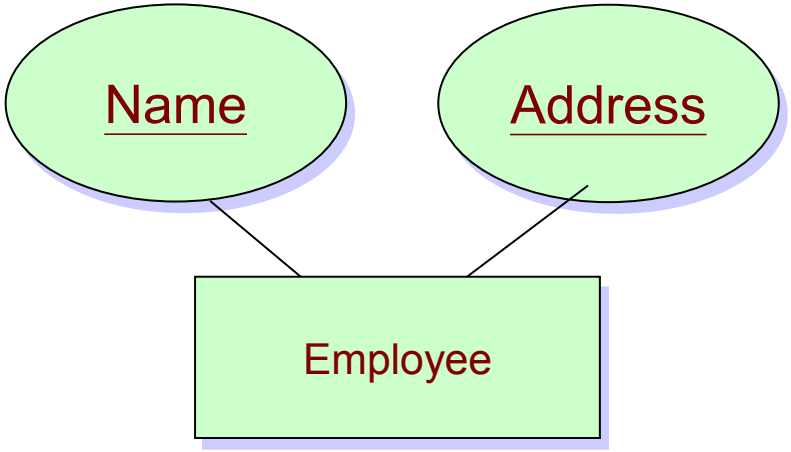
- **Key**
  - A set of attributes that can **uniquely** identity an entity
  - E.g., ID

A key is an attribute or a collection of attributes



# Key Attributes

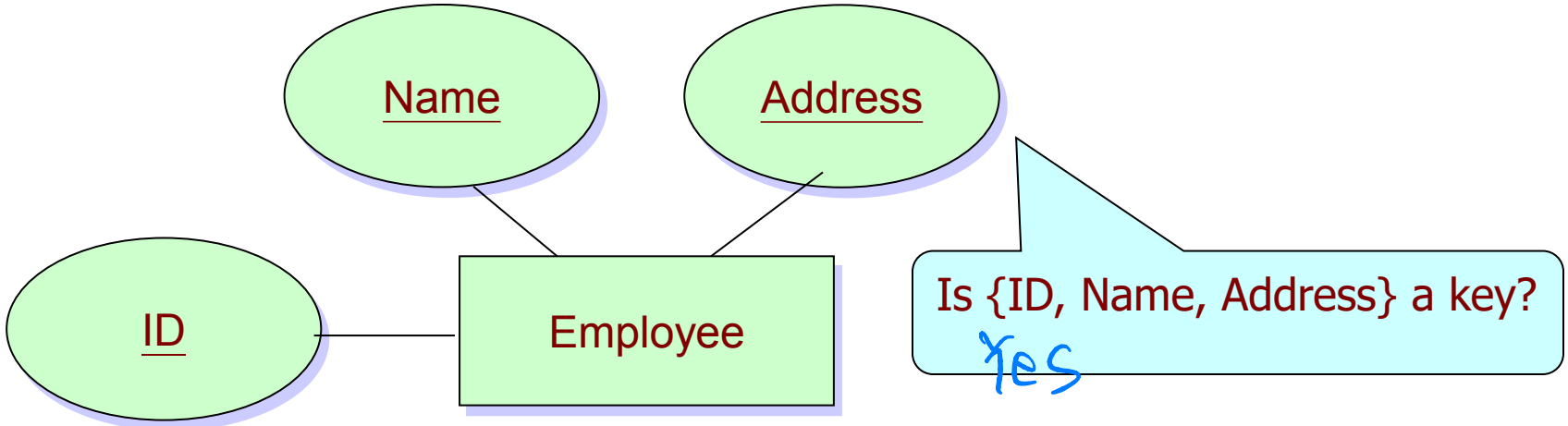
- **Composite Key**
  - Two or more attributes are used to serve as a key
  - E.g., Name or Address alone cannot uniquely identify an employee, but together they can



# Key Attributes

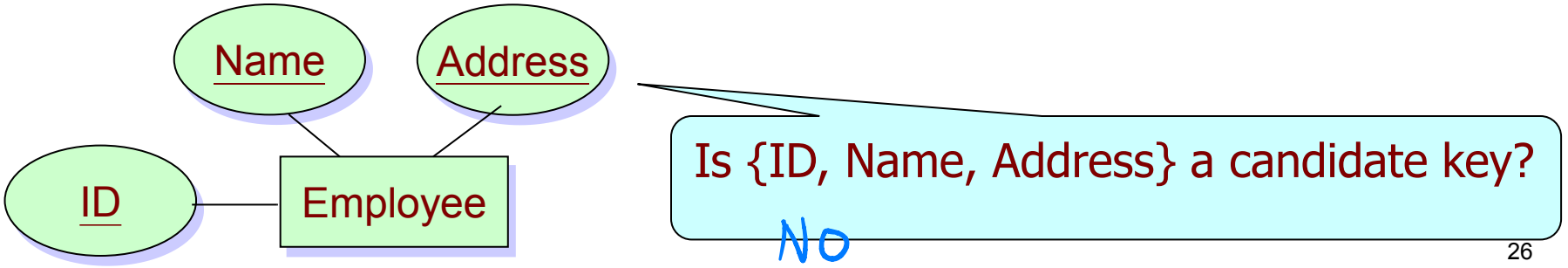
- An entity may have more than one key
- E.g., {ID} and {Name, Address} both are two keys

composite key



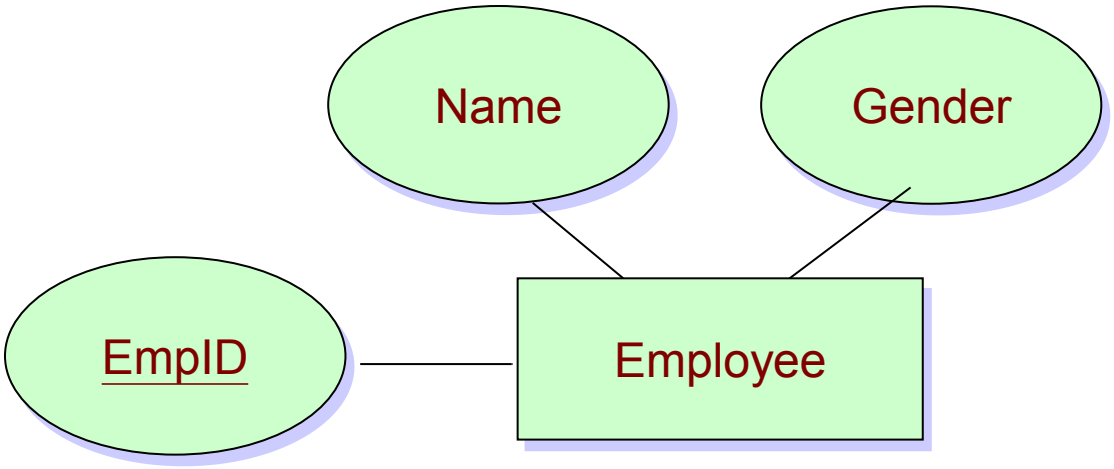
# Key Attributes

- A **minimal** set of attributes that uniquely identifies an entity is called a **candidate key**.
- E.g., both {ID} and {Name, Address} are two candidate keys
- If there are many candidate keys, we should choose one candidate key as the **primary key**.



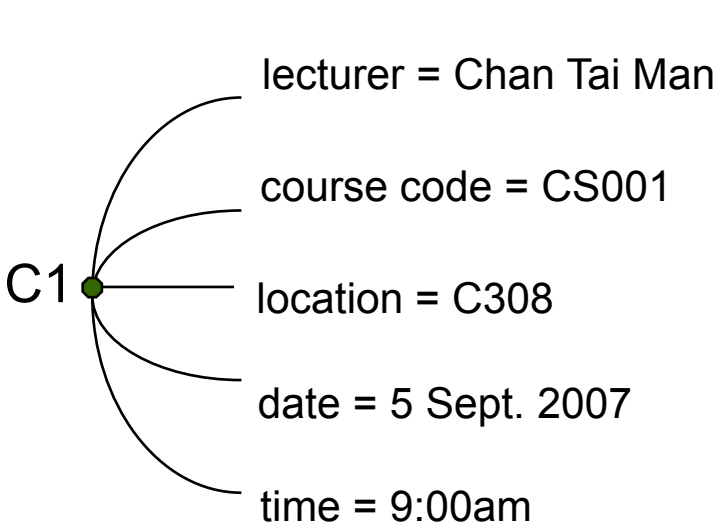
# Key Attributes

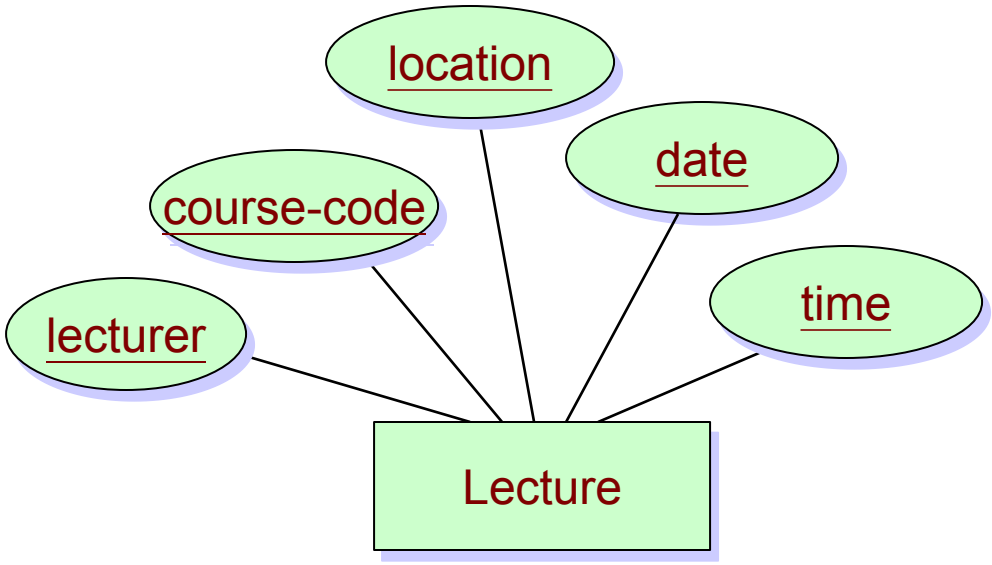
- Sometimes, artificial keys can be created
- E.g., if there is no ID stored in Employee, we can create a new attribute called “EmpID”



# Key Attributes

- The key should depend on the real life possibility rather than on the current set of the data.
- For example
  - In a database which contains only two employees aged 30 and 40, the age may distinguish each employee.
  - However, there can be in the future a new employee with the same age as an existing employee.
  - Therefore, {age} can not be a key.




- For example, {location, date, time} is a key.
- {lecturer, date, time} is also a key.
- These are candidate keys, we can choose one to be the primary key.
  - The same age as an existing employee.
  - Therefore, {age} can not be a key.

# Summary: Key

- ***Key***: An attribute or combination of attributes that uniquely identify an entity. A key means of specifying ***uniqueness***.
- ***Super Key***: A key that can be uniquely used to identify an entity, that may contain extra attributes that are not necessary to uniquely identify entities.
- ***Candidate Key***: A candidate key can be uniquely used to identify an entity without any extraneous data. It is a minimal super-key. Often abbreviate the ***Candidate Key*** to just ***Key***.

Candidate Key ⊆ Super Key

refers to the value, not the notation

not directly connected

contain unique values ↔ a super key with no repeated

keys

# Summary: Key

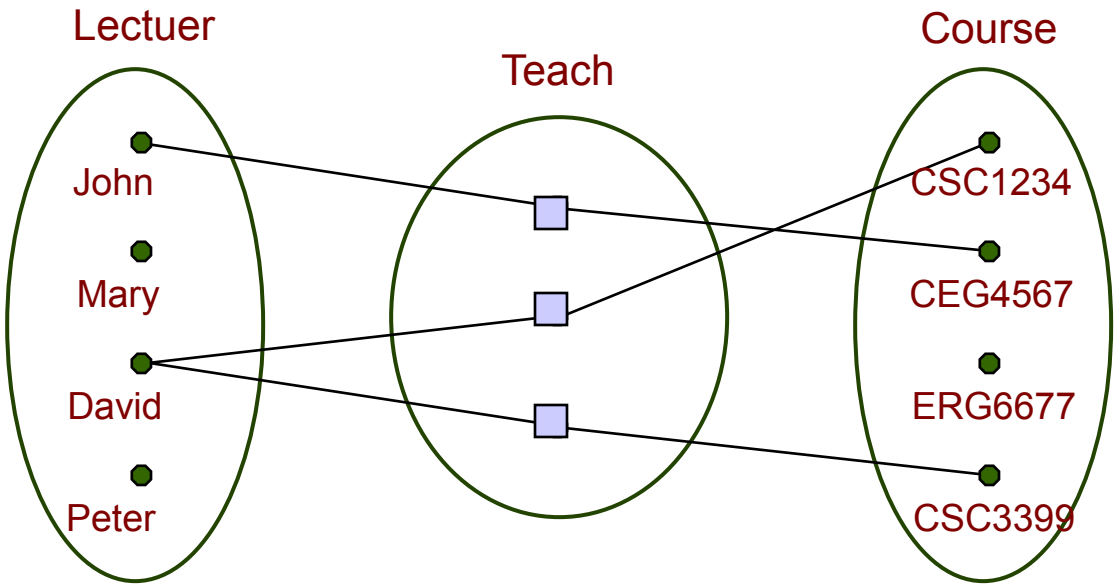
- ***Primary Key***: It is one of the candidate keys.
- ***Alternate Key***: A candidate key that is not the primary key is called an alternate key. A Alternate key depends on the primary key chosen
- ***Composite Key***: Key made up of multiple attributes.
- ***Surrogate Key***: Also called ***Artificial Key***. It is the Surrogate primary key acting as the substitute of the primary key. It is generated artificial internally.

# Outline

- Database Design
- Entity
- **Relationship**
  - **Binary relationship**
  - Non-binary relationship
- Weak Entity/Strong Entity
- Class Hierarchy
- Aggregation
- Conceptual Design with the E-R Model

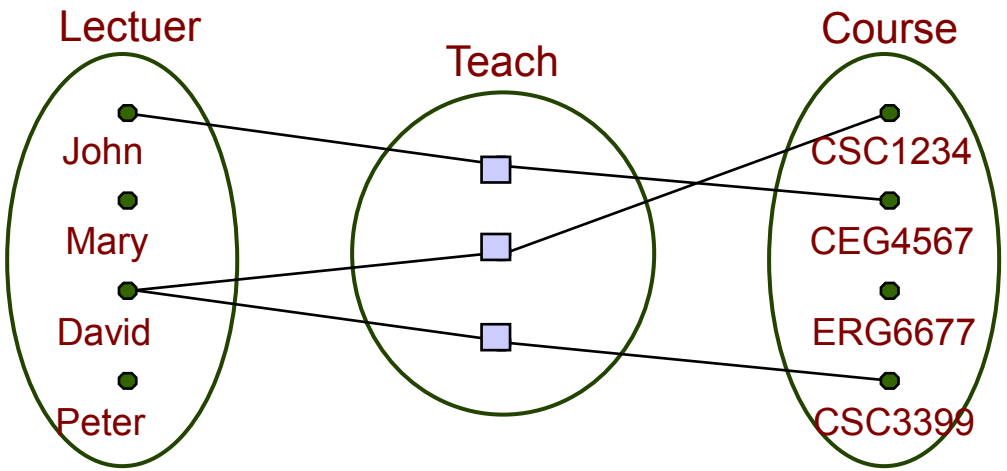
# Relationships

- A ***relationship*** is an association among two or more entities.
- Example:
  - E1 = ( John, Mary, David, Peter )
  - E2 = ( CSC1234, CEG4567, ERG6677, CSC3399 )



- As with entities, we may wish to collect a set of similar relationships into a ***relationship set***.
- A relationship set can be seen as a set of *n*-items (*n*-tuple):

$$\{(e_1,\cdots,e_n)\,|\,e_1 \in E_1,\cdots e_n \in E_n\}$$



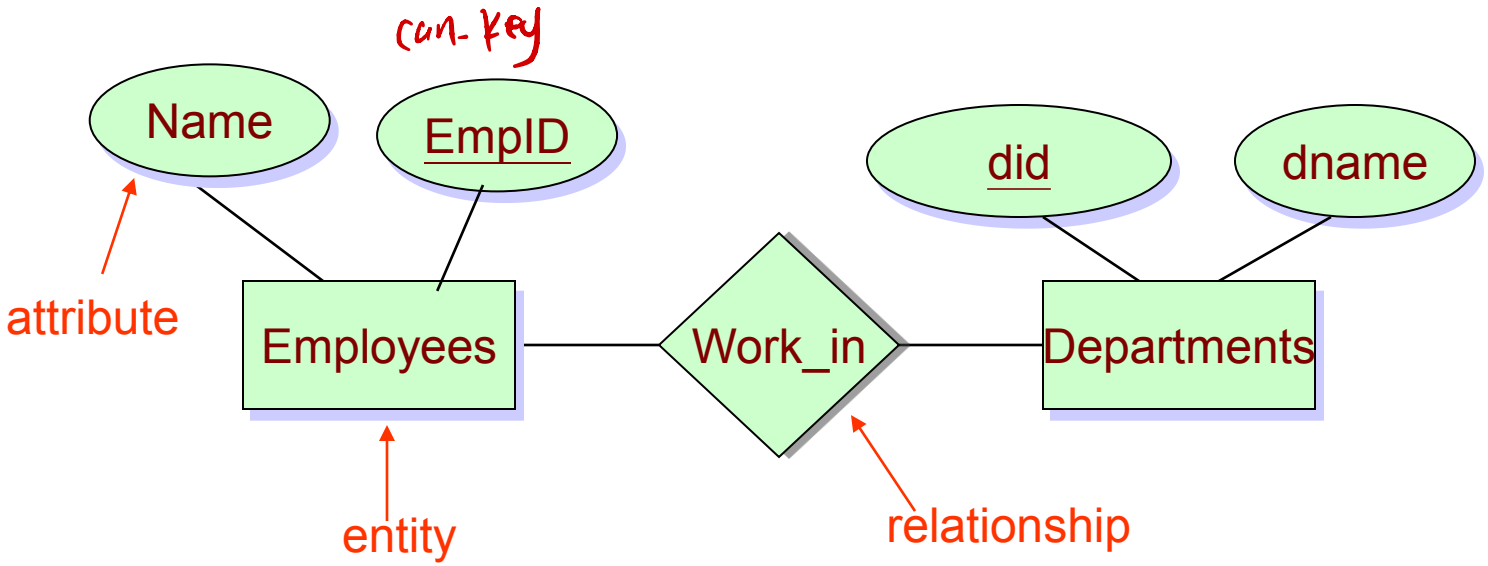

- The teach relationship can be represented by the set of ordered pairs: {(John,CEG4567), (David, CSC1234),(David,CSC3399)}

# Relationships

- The ***degree*** refers to the number of entity sets that participate in a relationship set.
- Relationship sets that involve two entity sets are ***binary*** (or degree two).
- Relationships among more than two entity sets are rare. (We will discuss later in details)

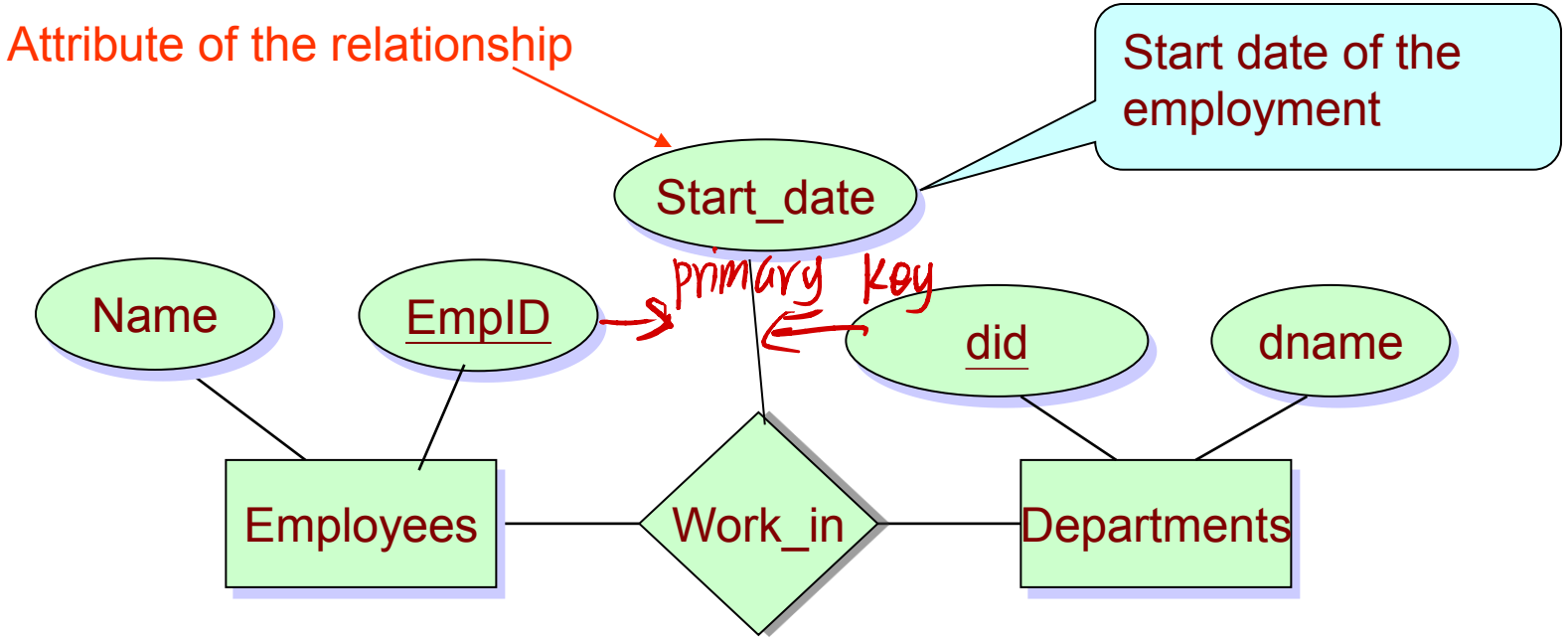
# Binary Relationship

- Employees work in departments
- “Work_in” is a relationship between Employees and Departments
- It means that an employee works in multiple departments and vice versa



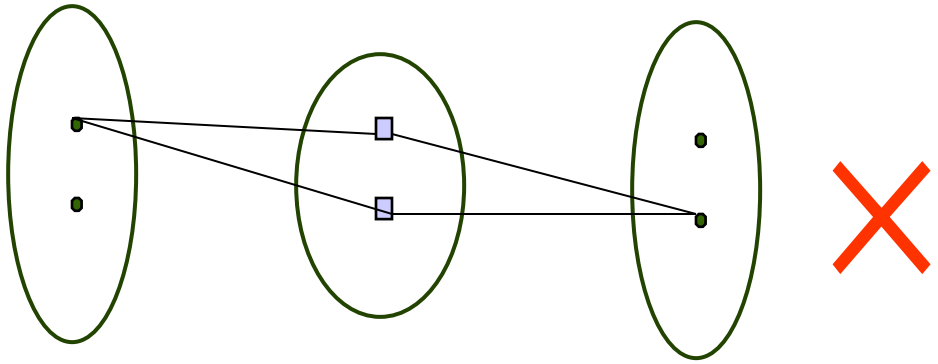
# Binary Relationship

- A relationship can also have attributes which are used to describe the record information about the relationship (instead of the information of each individual entity).



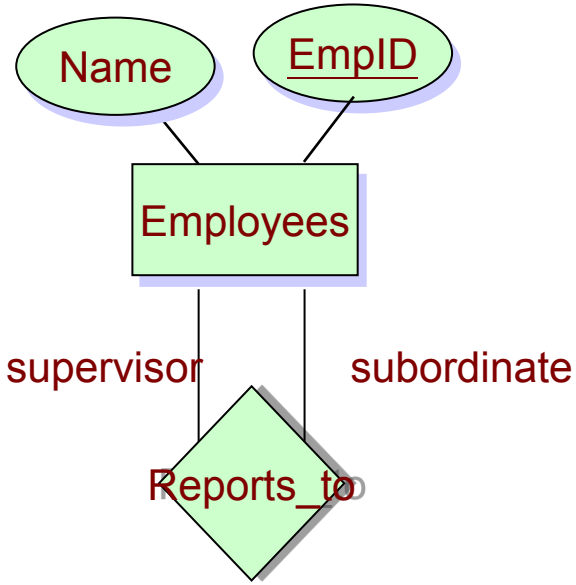
- A relationship must be uniquely identified by the participating entities, without reference to the descriptive attributes.
  - In the previous example, each relationship must be uniquely identified by the combination of the *employee id* and the *department did*.
- Thus, for each employee-department pair, we cannot have more than one associated "since" value.
  - (Mary-ID, Dept ABC ; 1-2003)
  - (Mary-ID, Dept ABC ; 2-2004)

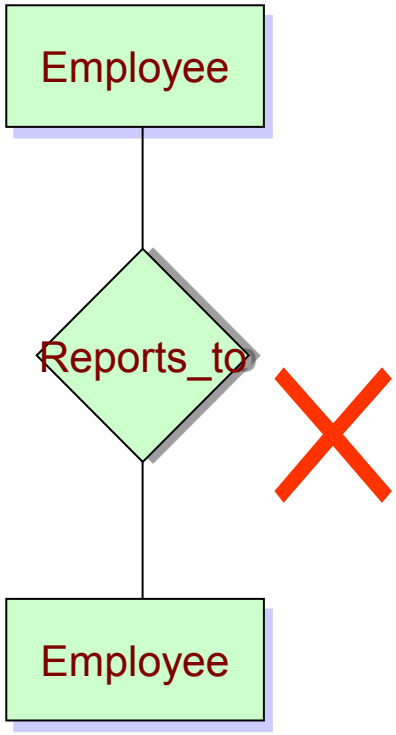
→ different ID in the same department



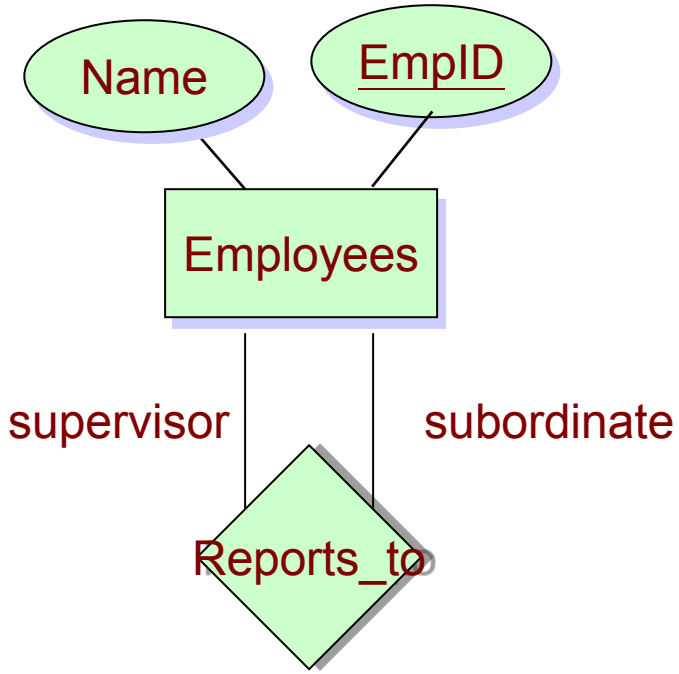
# Recursive Relationship

- Recursive Relationship
  - Entity sets of a relationship need not be distinct
  - Sometimes, a relationship might involve two entities in the same entity set
  - E.g., Employees related to employees



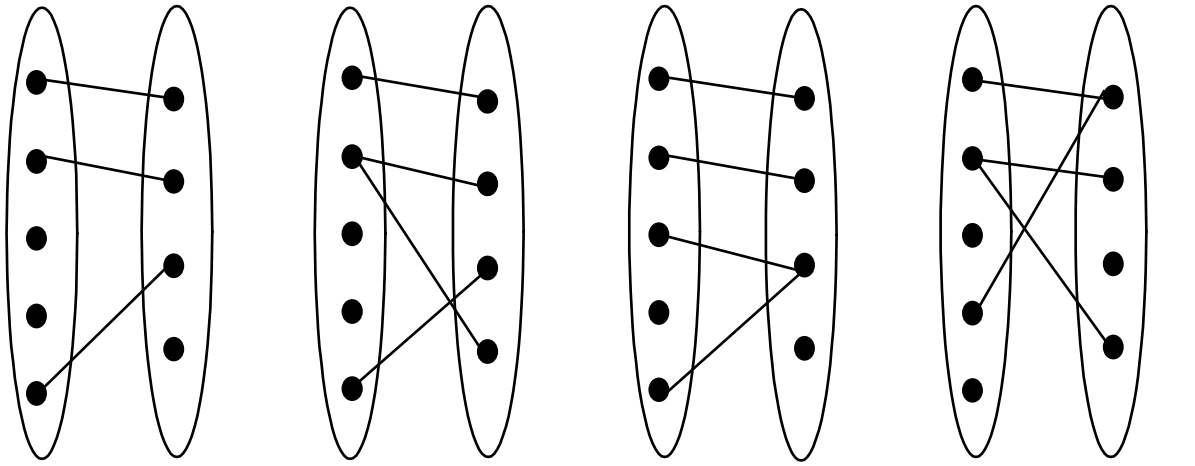
# Recursive Relationship




- Since employees report to other employees
  - Every relationship in “Reports_To” is of form (emp1, emp2) where both emp1 and emp2 are entities in employees.
  - However, they play different roles.
    - emp1 reports to emp2, which is reflect in the role indicators supervisor and subordinate in the diagram

# Constraints

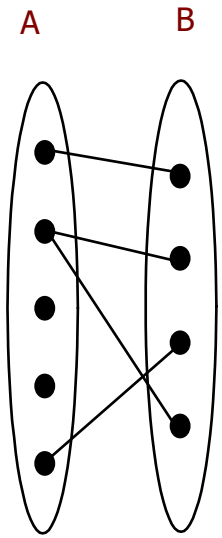
- The model describes data to be stored and the **constraints** over the data
- The type of association of a binary relationship can be classified into the following cases: ∃



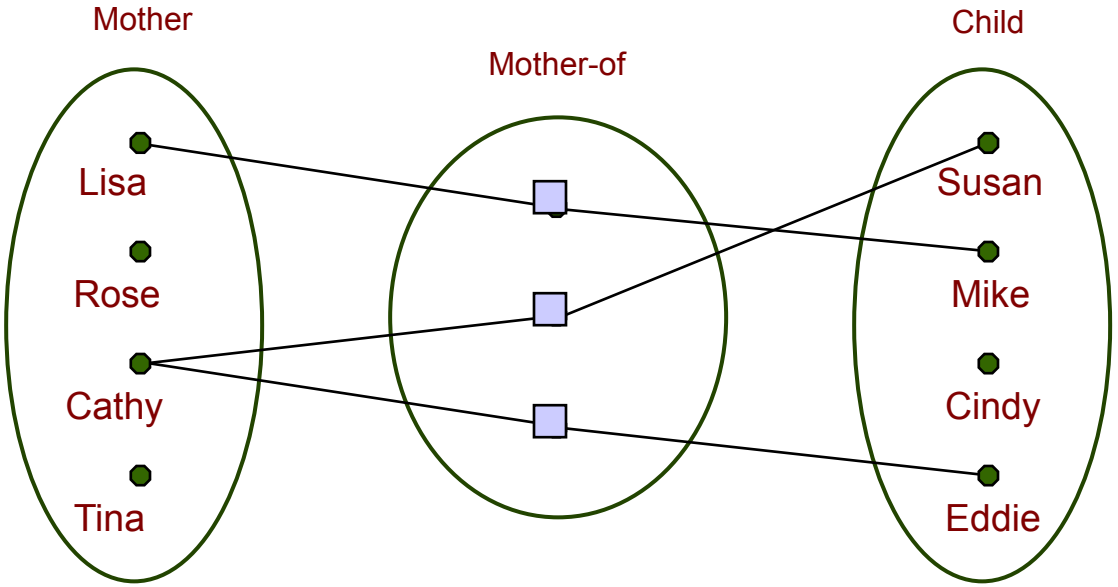

The preceding discussion identified each relationship in both directions; that is, relationship are ***bidirectional***.

# One-to-many relationship

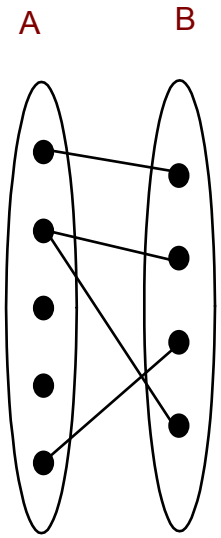
- An entity in B can be associated with at most one entity in A
- Example: each child can appear in at most one mother-child relationship.



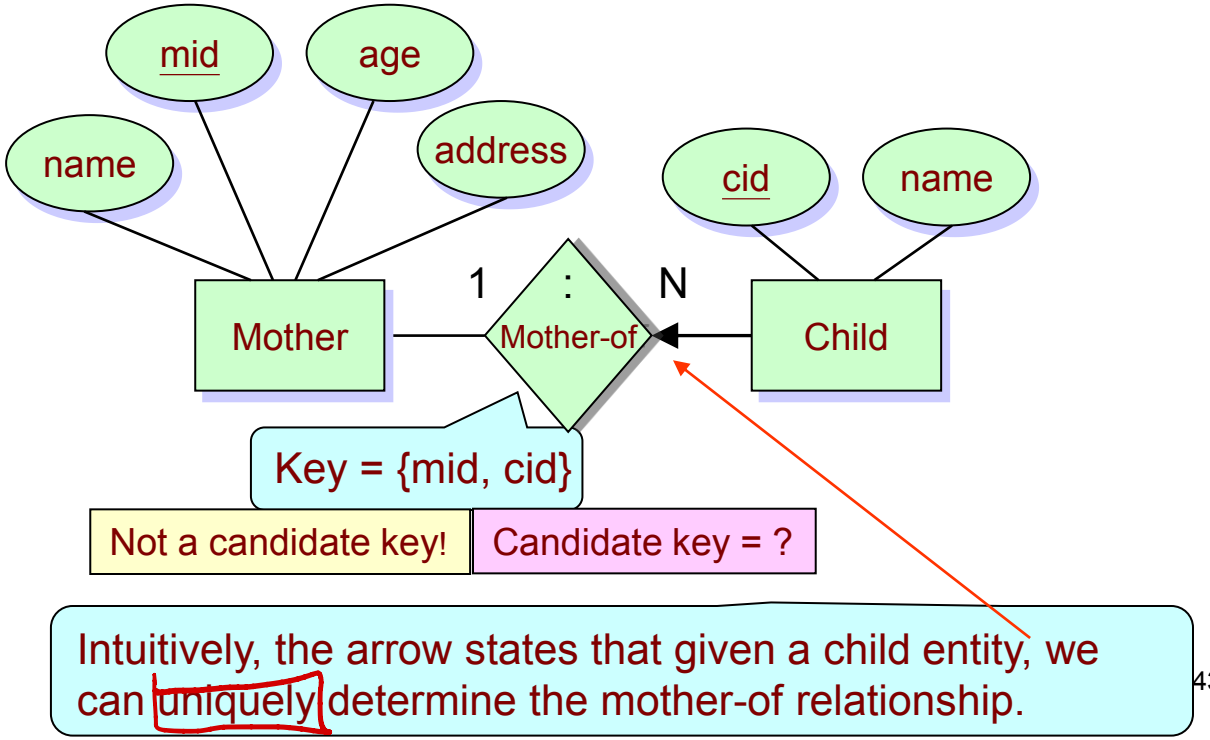

1-to-Many
1:N



- Child has a key constraint in the mother-of relationship set.
- This restriction can be indicated by an arrow in the E-R diagram.
  - **A** Child is associated with at most **one** Mother via Mother-of
  - **A** Mother is associated with **several** (including 0) Children via Mother-of

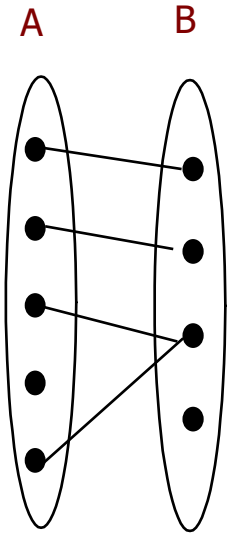



1-to-Many
1:N

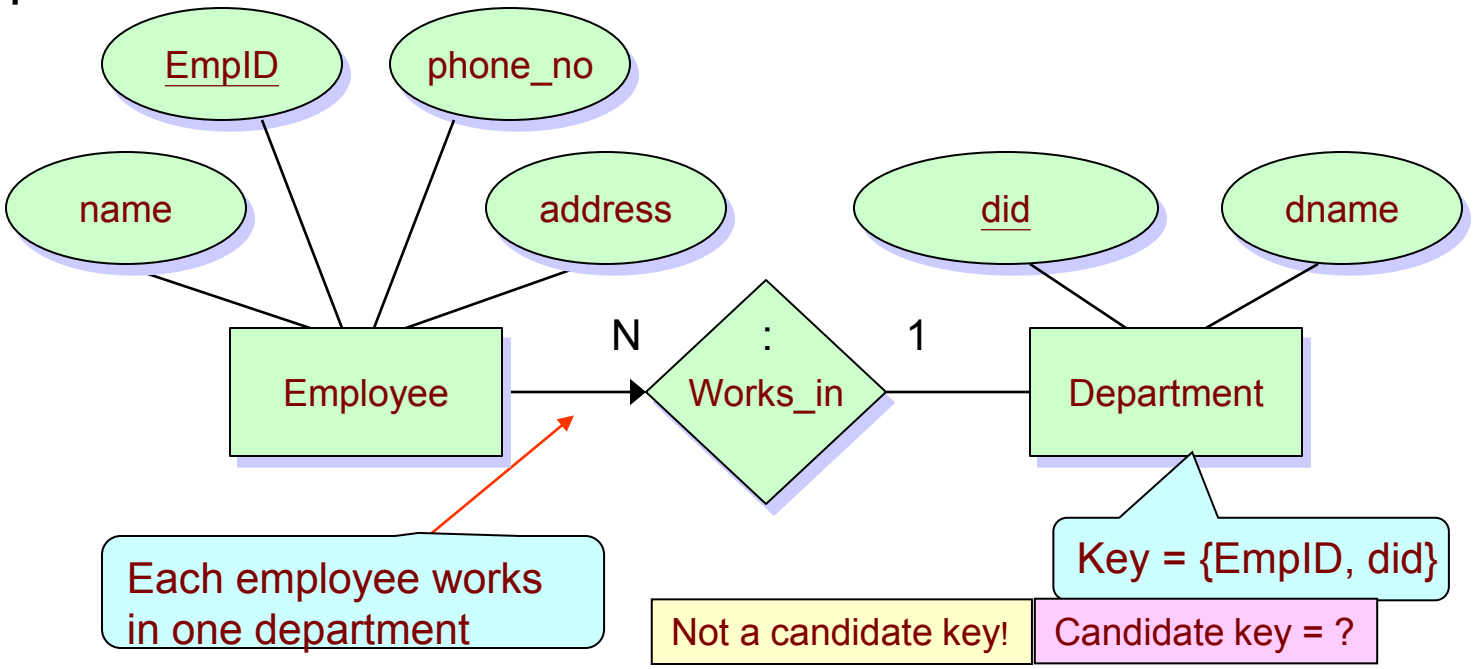


# Many-to-one relationship

- Similar to 1-to-many
- **A** Department is associated with **several** (including 0) Employees via Works_in,
- **A** Employee is associated with **at most one** Department via Works_in

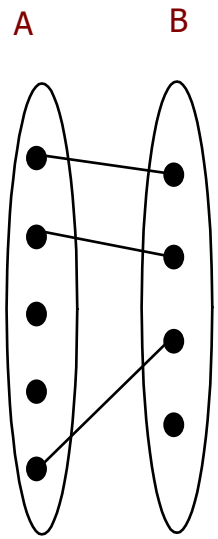



Many-to-1
N:1

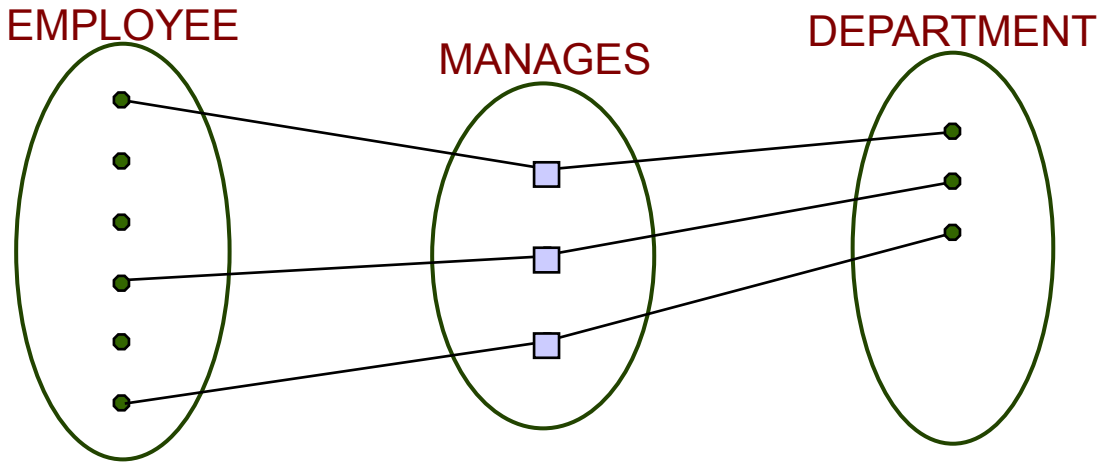


# One-to-one relationship

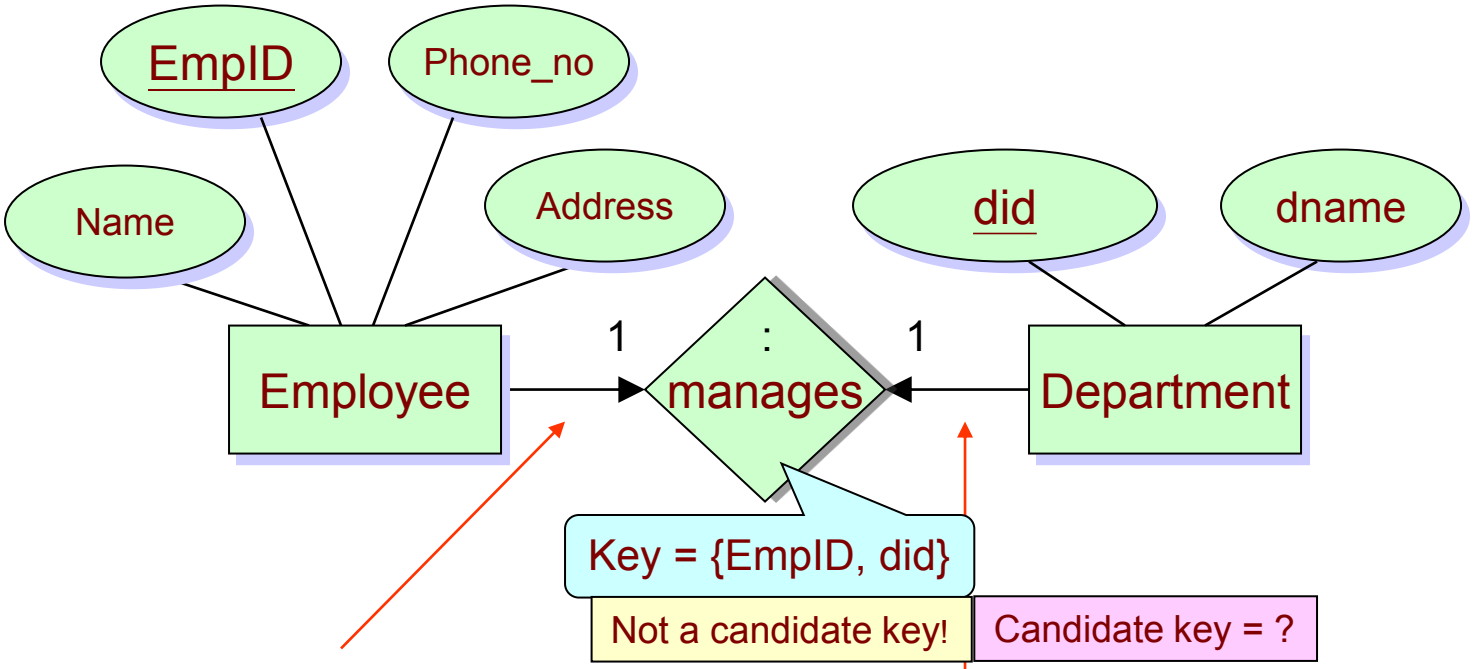
- If the relationship between A and B satisfies the one-to-one mapping constraint from A to B, then
- **An** entity in A is related to **at most one** entity in B, and
- **An** entity in B is related to **at most one** entity in A.

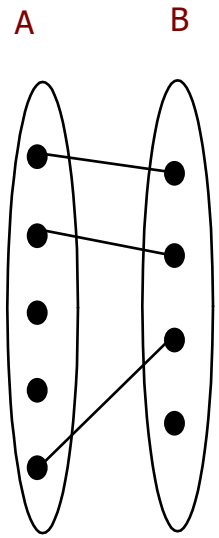



1-to-1
1:1



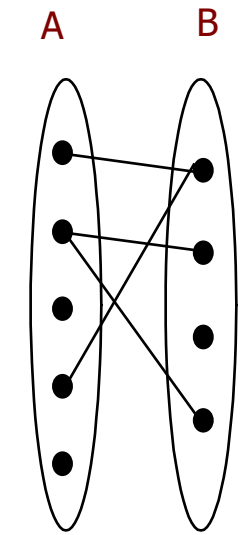
- A Employee is associated with at most one Department via the relationship manages
- A Department is associated with at most one Employee via manages



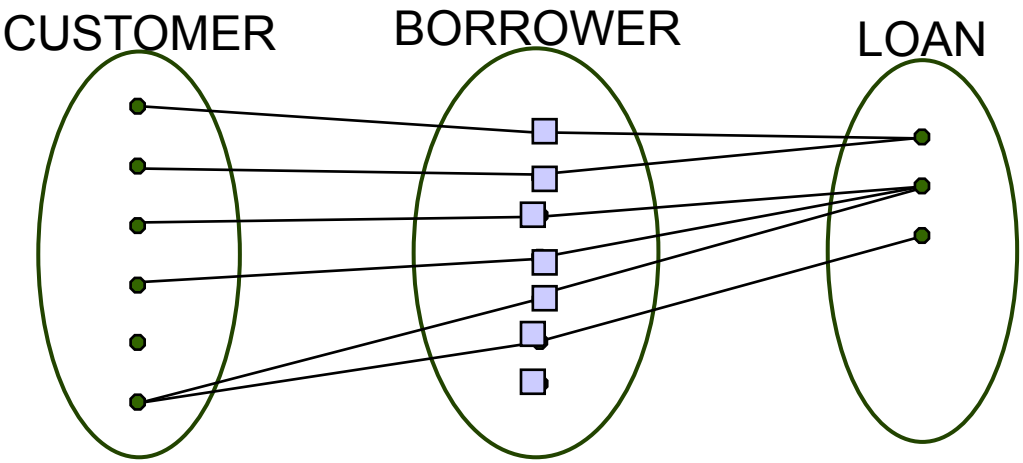



# Many-to-many Relationship

- **An** entity in A is associated with **any number** of entities in B
- **An** entity in B is associated with **any number** of entities in A
- In fact, there is no restriction in the mapping.

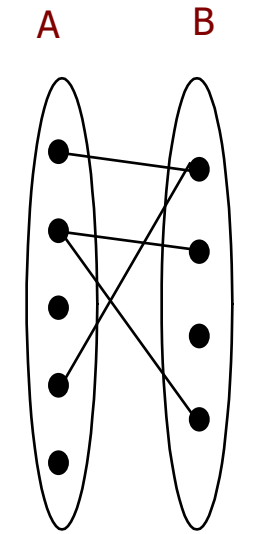



Many-to-Many
N:M

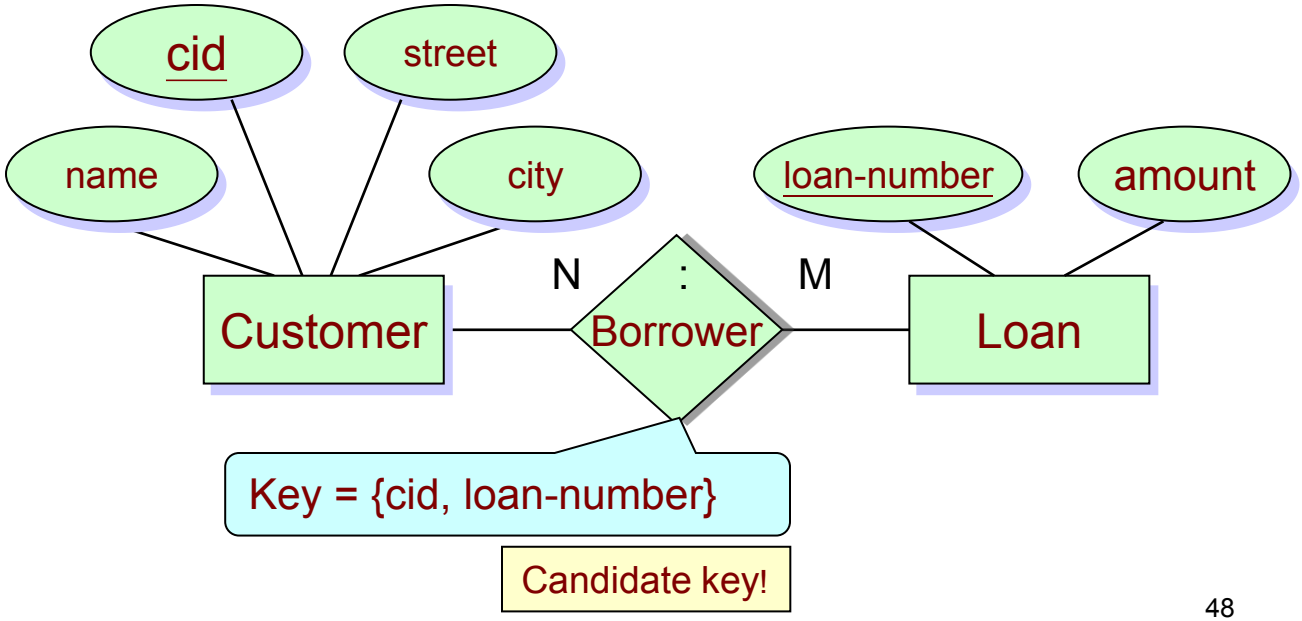


# Many-to-many Relationship

- A customer can associate with several loans (possibly 0) via Borrower
- A loan can associate with several customers (possibly 0) via Borrower




Many-to-Many
N:M



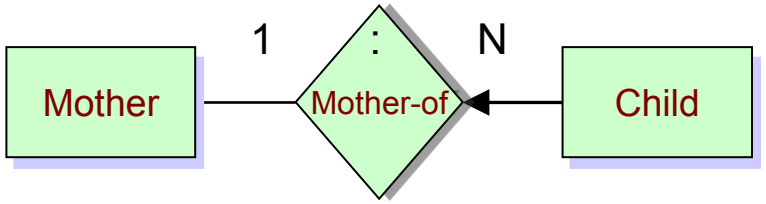
# Keys for a Relationship Set

- The concept of keys is also used to identify a relationship, as in entities.
- Key of relationship set R
  - R = a set of attributes that can ***uniquely*** identify a relationship in R.
- For example:
  - CID is the primary key of customer, and
  - account-number is the primary key of account.
  - the attributes of the relationship set ***custacct*** are then (account-number, CID).

  This is enough information to enable us to relate an account to a person.

# Key Constraints (1-to-many)

- An entity set E has a key constraint in a relationship set R
- The key of E can be used as the key in R.
- For example
  - child-mother: is a many-to-one relationship
  - entity Child has a key constraint
  - Child can be the primary key in the relationship set

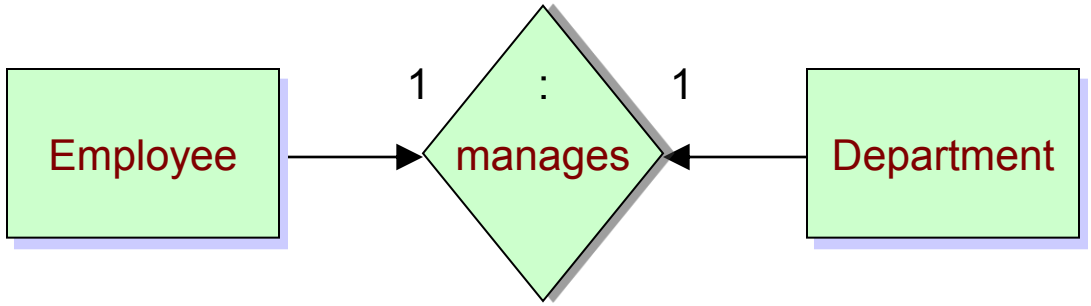



1-to-many or many-to-1

Key for Mother-of = (Child_id)

# Key Constraints (1-to-1)

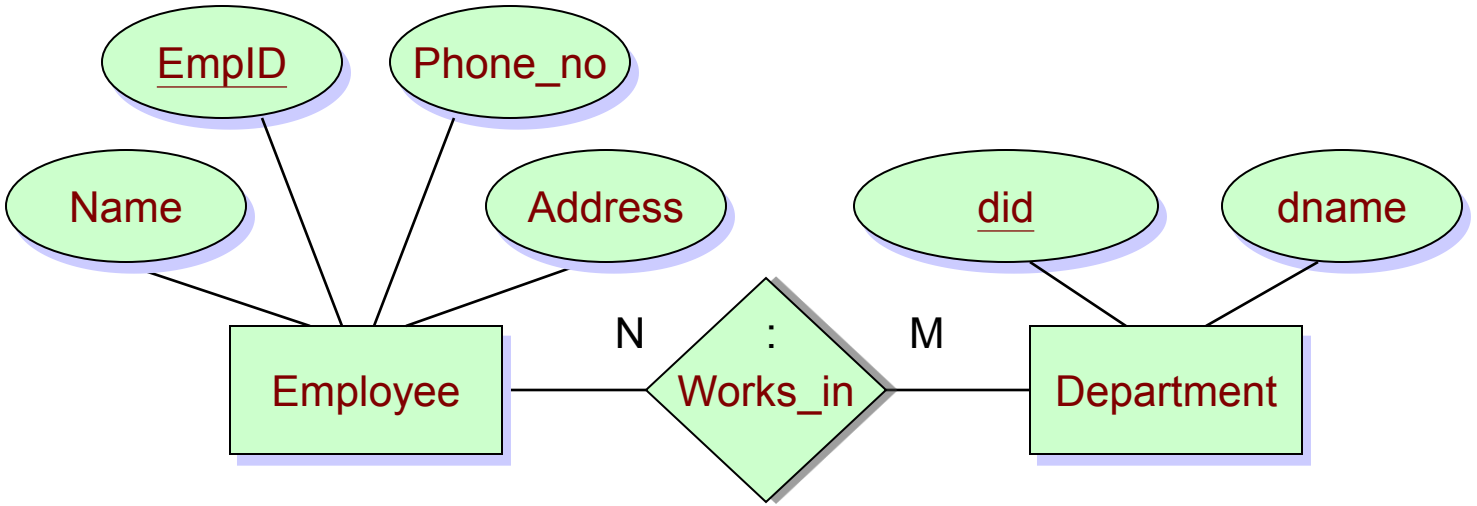
- One-to-one
  - For an one-to-one relationship between two entity sets E and F,
  - Either key(E) or key(F) is the key for the relationship set.
- For example, “manages” relation




Key for managers = (employee_id)
Or
Key for managers = (department_id)

# Key Constraints (many-to-many)

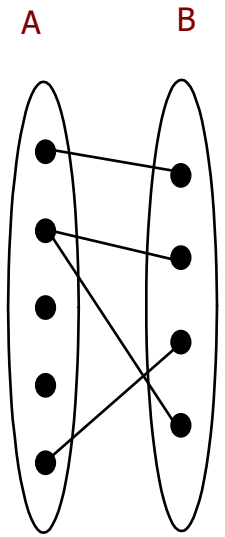
- Many-to-many
  - For a relationship among $E_1$, …, $E_k$, with no mapping constraint, the primary key is normally the ***union*** of the primary keys for $E_1$, …, $E_k$.
- For example, "Work_in" relation




Key for Works_in = (employee_id, department_id)

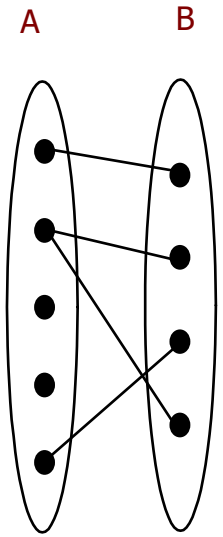
# Participation Constraint

- The above constraints (e.g., Works_in) tells us that a department have some employees.
- A natural question to ask is whether every department at least have one employee.
- Suppose that each department at least have one department. Such a constraint is called a **participation constraint.**



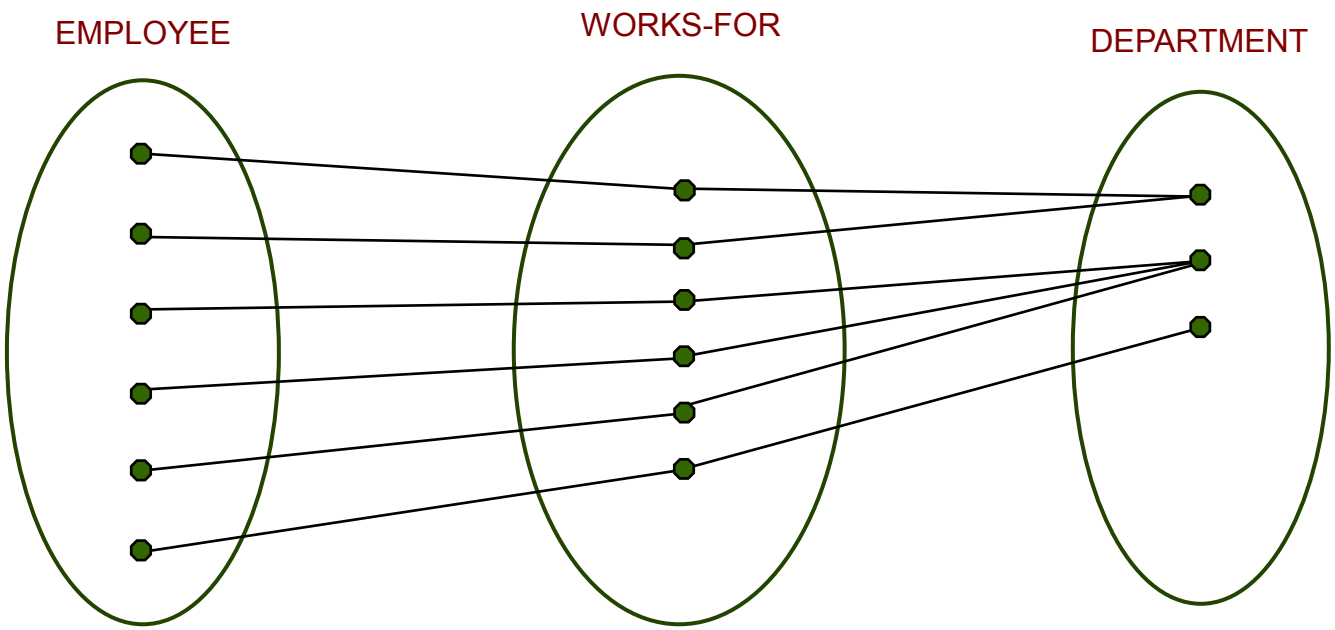
# Participation Constraint

- We can classify participation in relationships as follows.
- **Total**
  - Each entity in the entity set must be associated in at least one relationship
- **Partial**
  - Each entity in the entity set may (or may not) be associated in a relationship

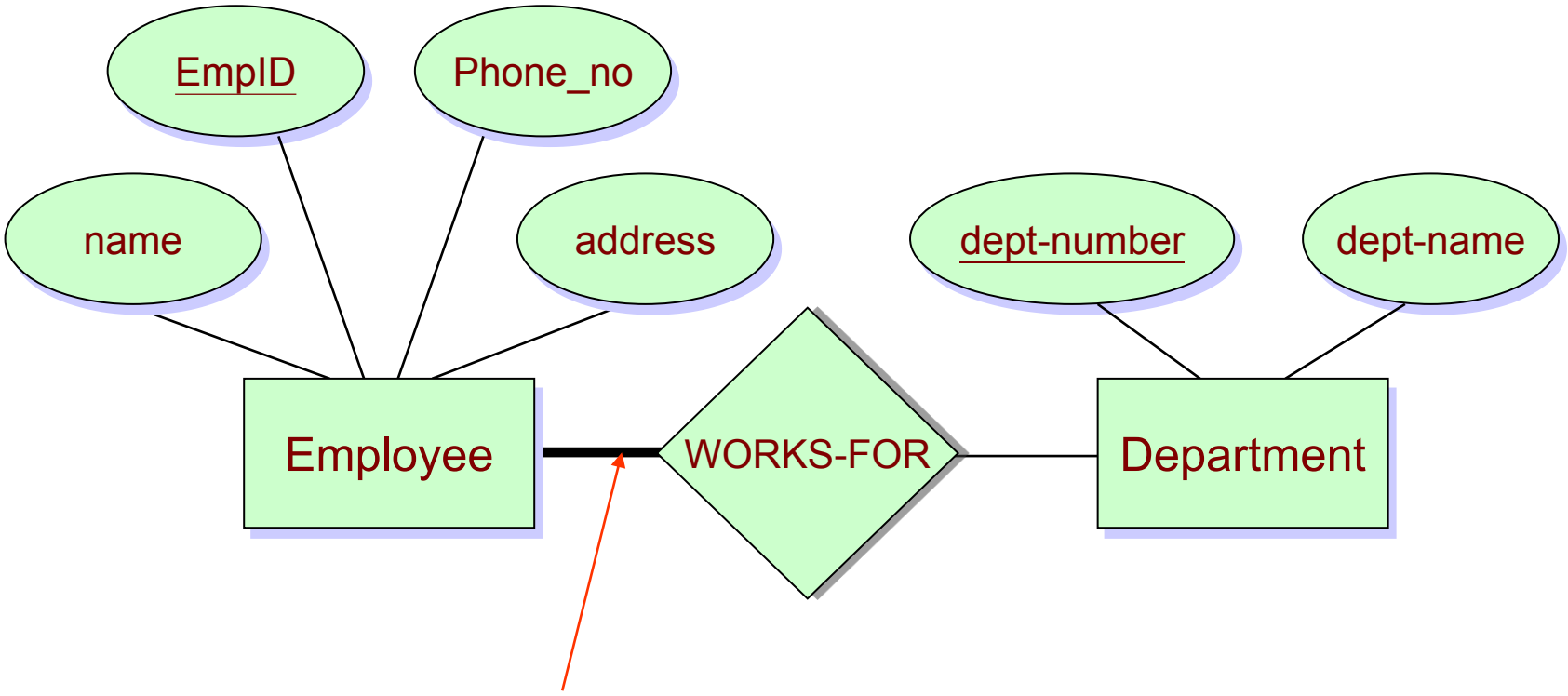


# Participation Constraint

- For example
  - ***Every employee*** must work for some department.
  - The participation of EMPLOYEE in WORKS-FOR is ***total participation***



# Participation Constraint




If the participation of an entity set in a relationship set is total, the two are connected by a thick link.
Independently, the presence of an arrow indicates a key constraint.

# Outline

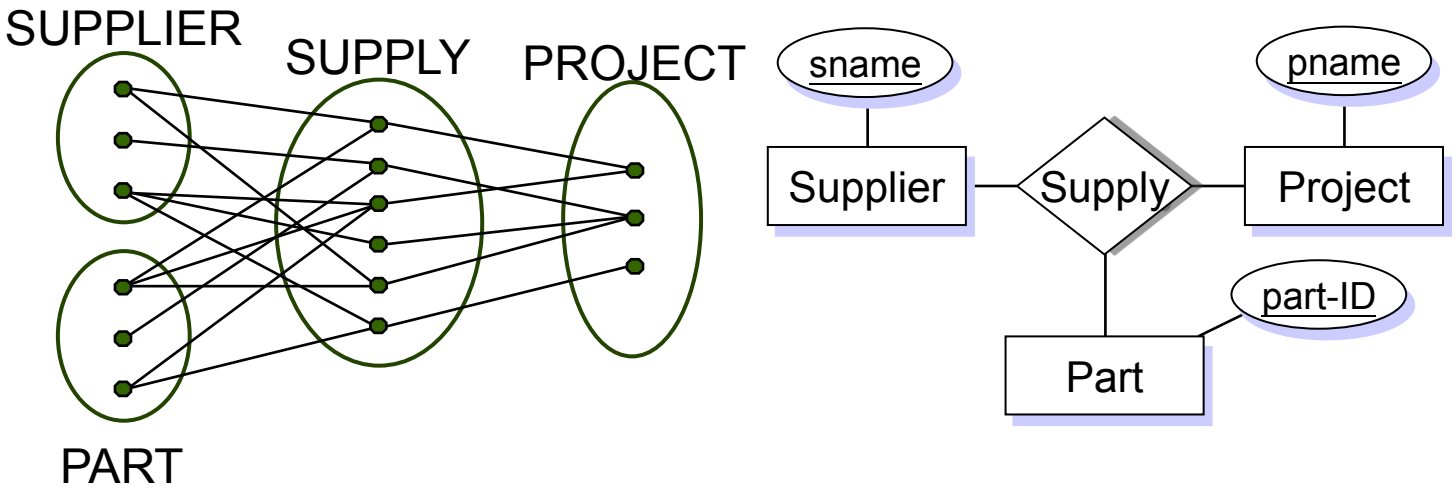
- Database Design
- Entity
- **Relationship**
  - Binary relationship
  - **Non-binary relationship**
- Weak Entity/Strong Entity
- Class Hierarchy
- Aggregation
- Conceptual Design with the E-R Model

# Non-Binary Relationship Set

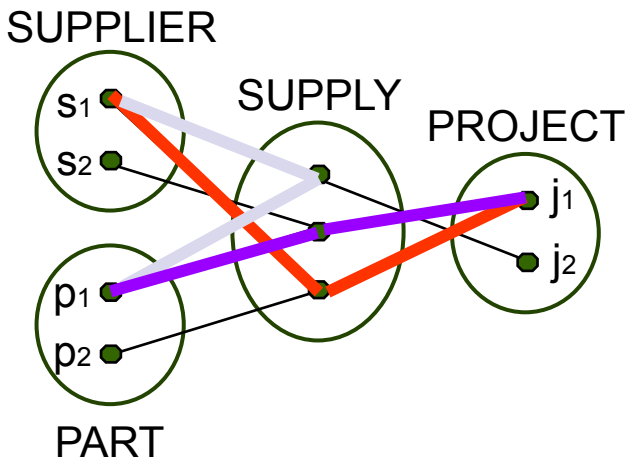
- A relationship set is a subset of the Cartesian product

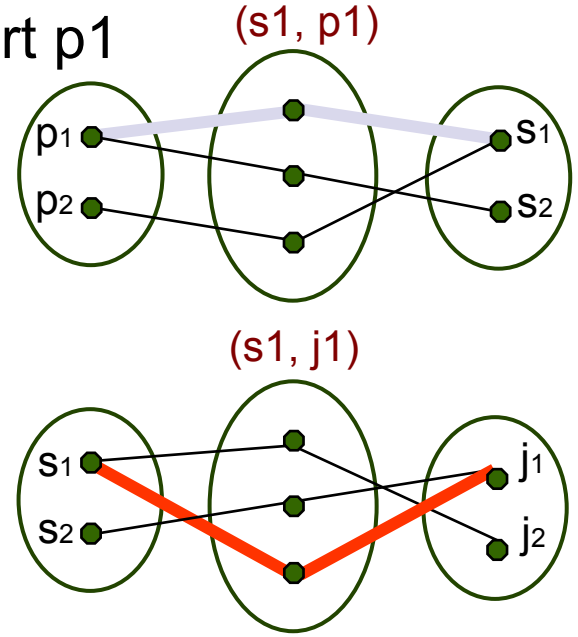
$$E_1 \times E_2 \times \ldots \times E_n$$

- *n* = the degree of the relationship set
- In general, a non-binary relationship set cannot be directly replaced by a number of binary relationship sets. For example,



- Consider supplier s1, project j1, and part p1




We have the following items in the SUPPLY:

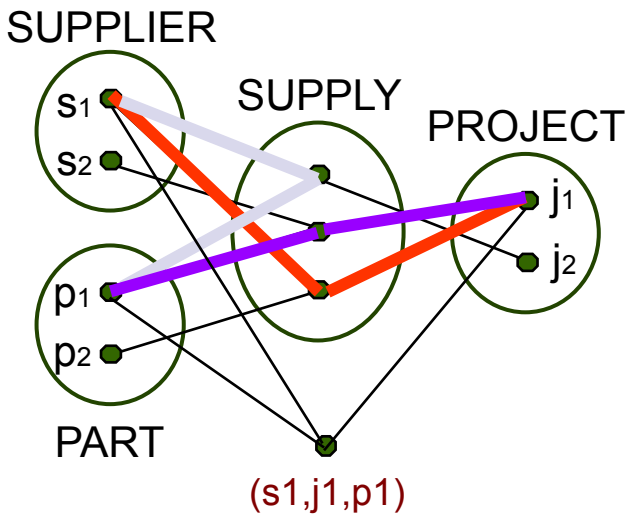
- (**s1, p1**, j2)
- (**s1**, p2, **j1**)
- (s2, **p1, j1**)

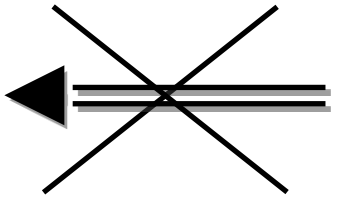
=>

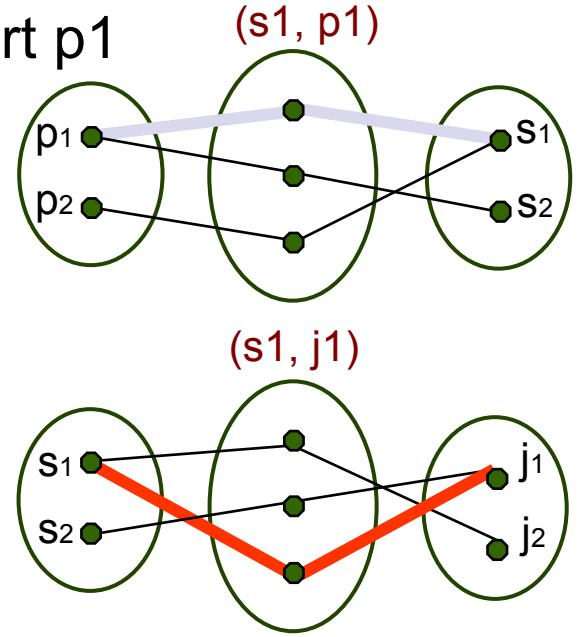
(s1, p1)
(s1, j1)
(j1, p1)

- Existence of (s1,p1), (j1,p1) and (s1,j1), does not imply existence of (s1,j1,p1) necessarily
  - This is known as the connection trap

- Consider supplier s1, project j1, and part p1

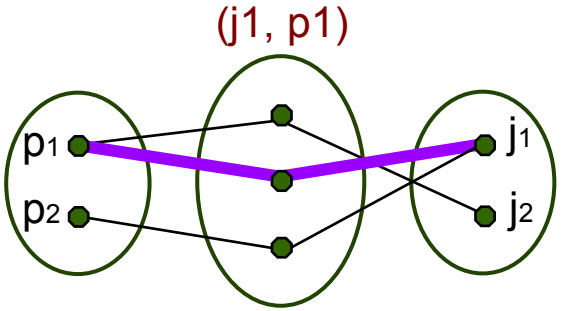





We have the following items in the SUPPLY:

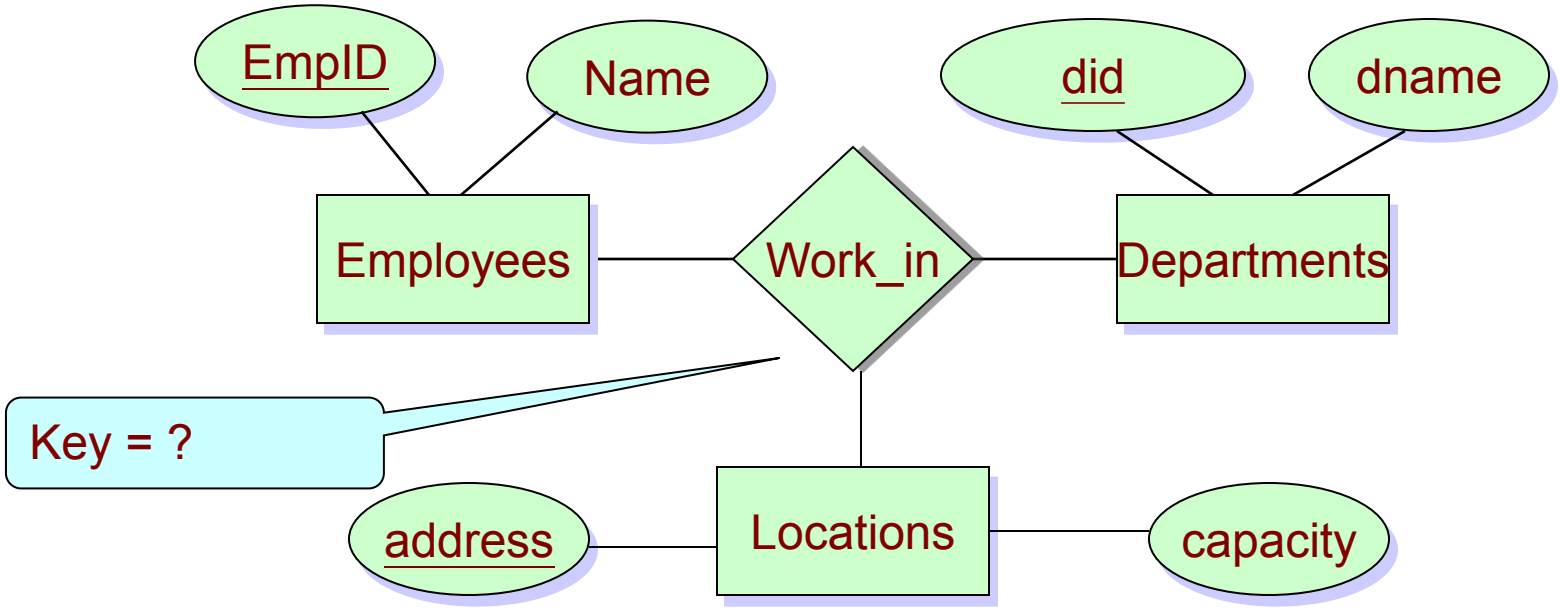
(s1,j1,p1) <= (s1, j1), (s1, p1), (j1, p1)



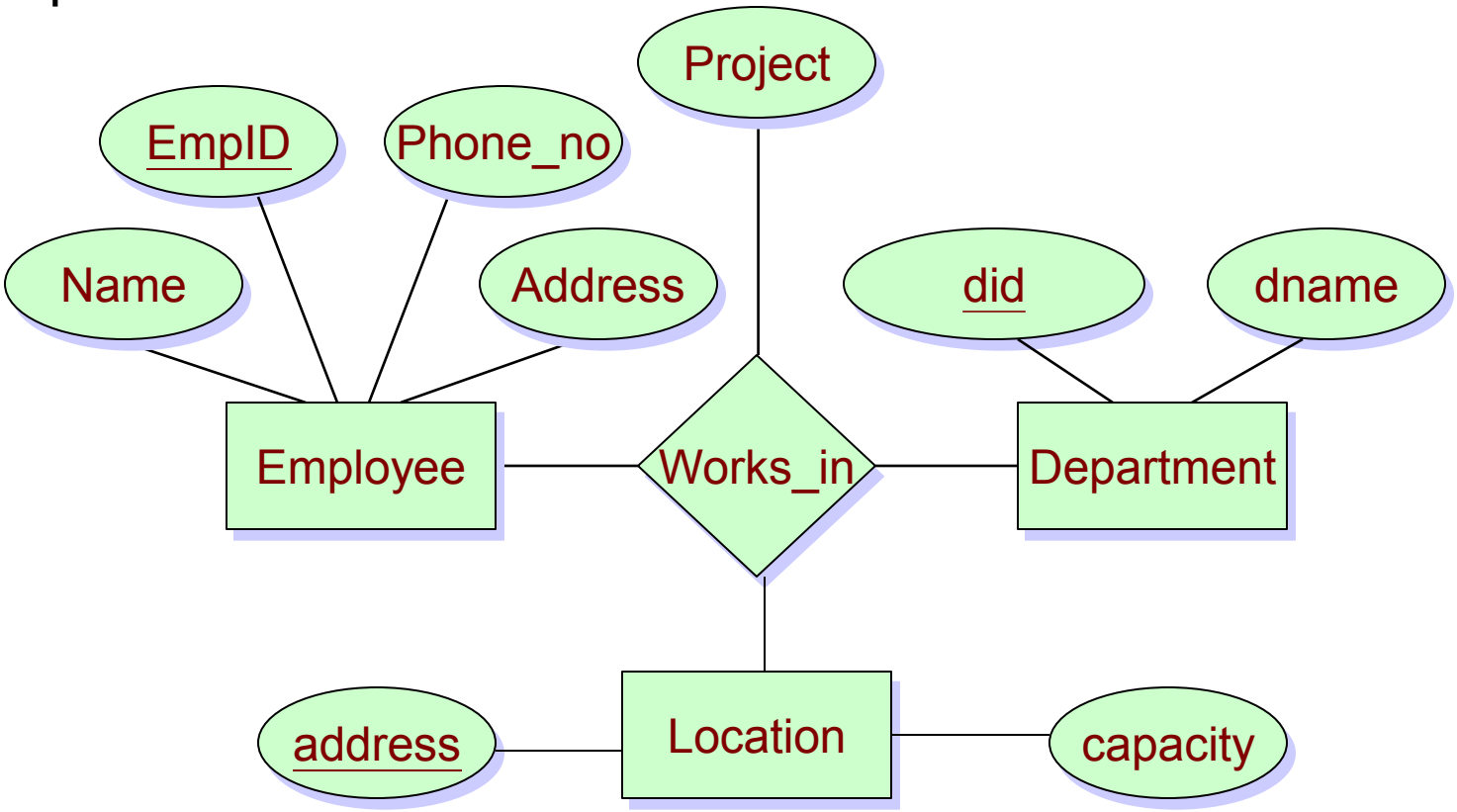

- Existence of (s1,p1), (j1,p1) and (s1,j1), does not imply existence of (s1,j1,p1) necessarily
  - This is known as the connection trap

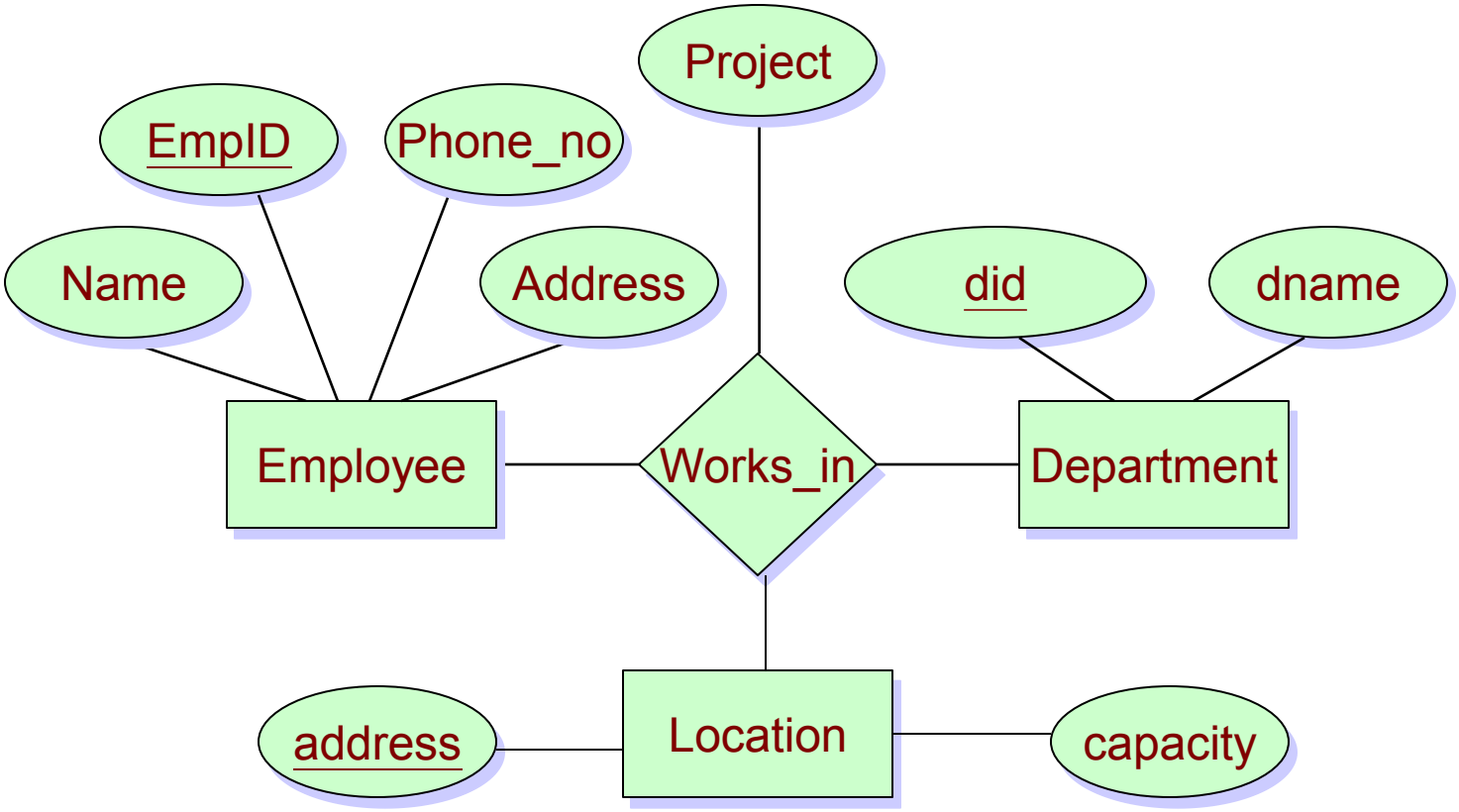
# Example: Ternary Relationship

- Suppose now each department has offices in several locations and we want to record the locations at which each employee works.



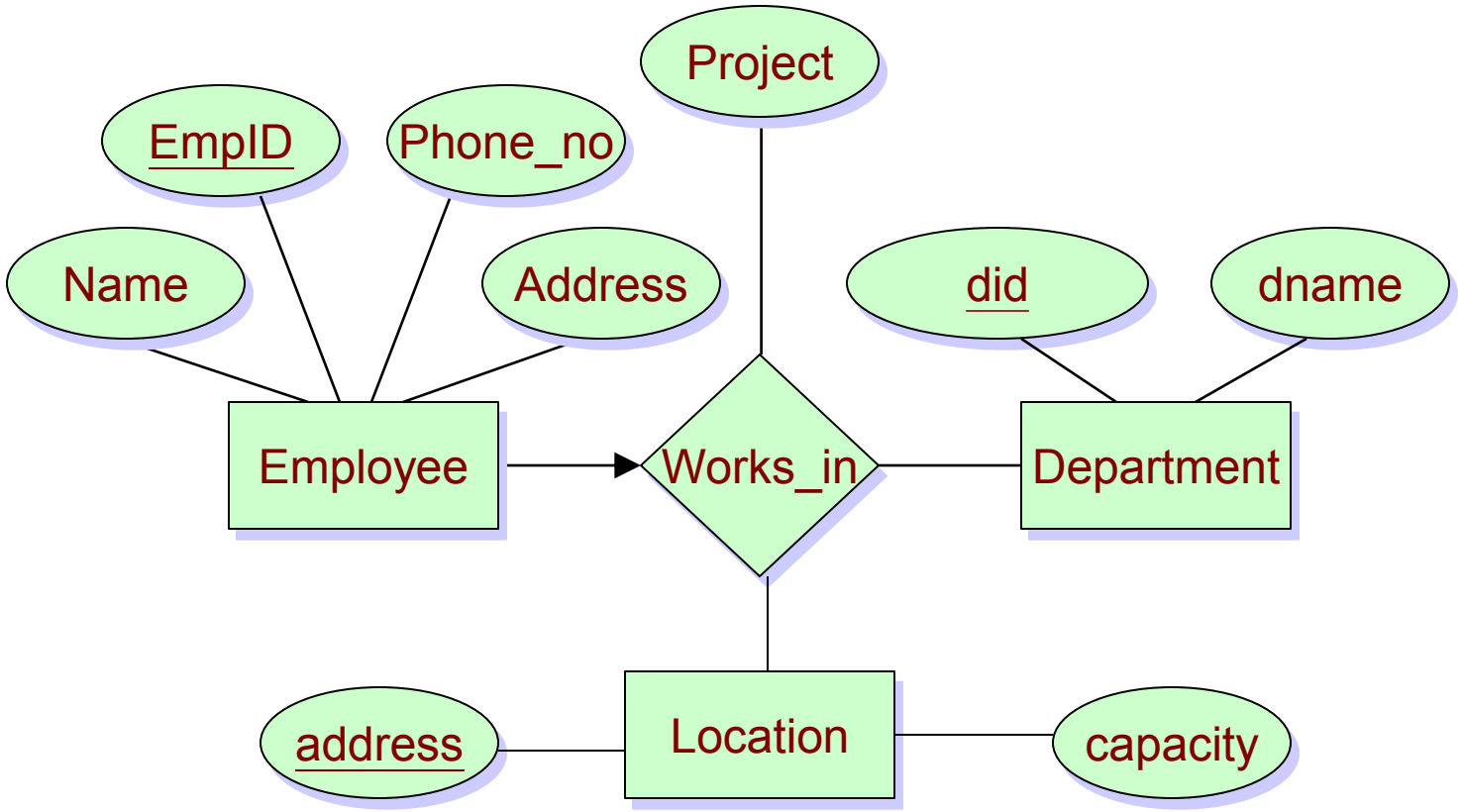
- Can an employee work in two locations for the same department ?
- Can an employee work in two projects for the same department ?



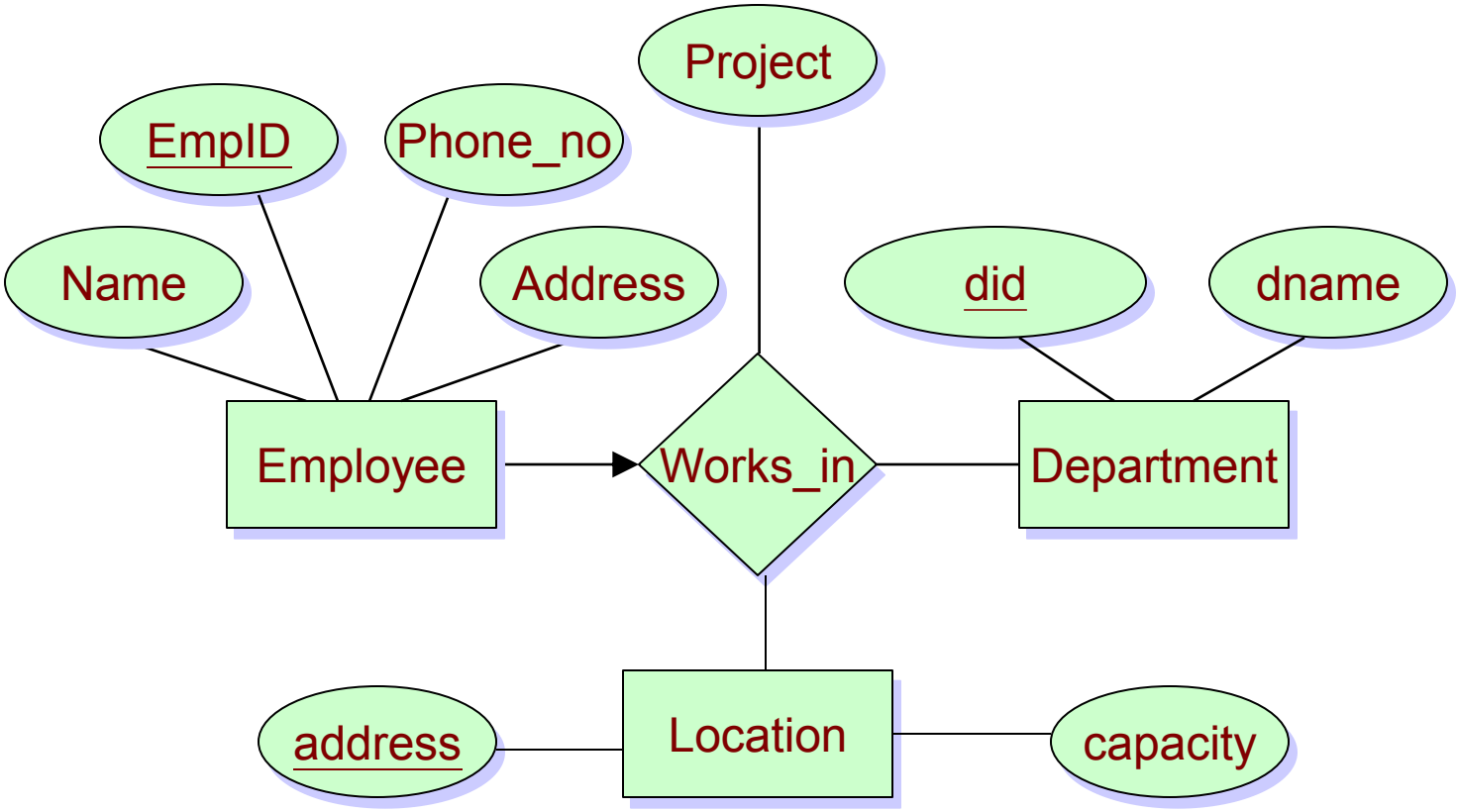



- (emp1, dept1, location1, proj1)
- (emp1, dept1, location1, proj2) ✗
- (emp1, dept1, location2, proj2)
- (emp1, dept2, location1, proj1)

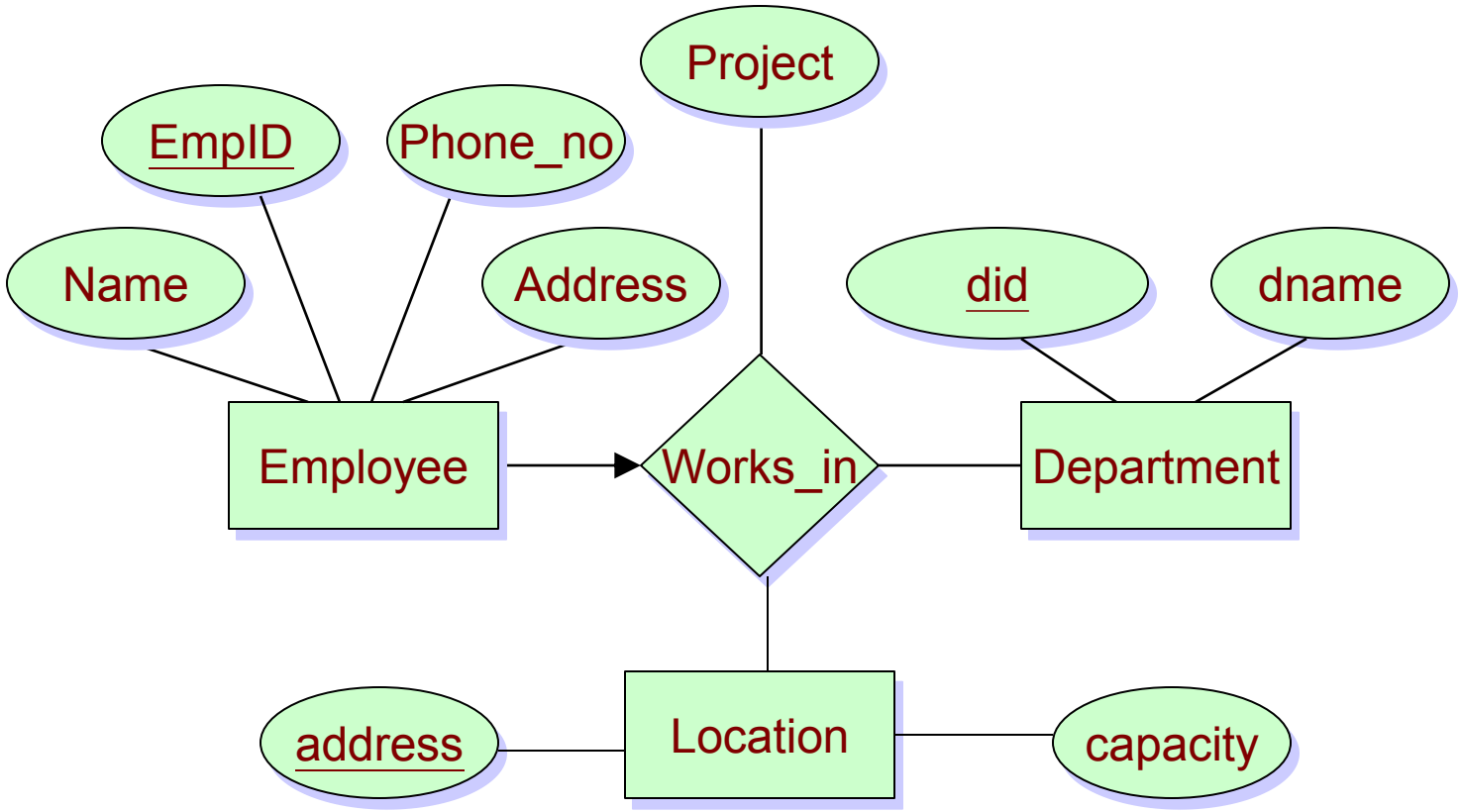
- The key constraint is not limited to binary relationships.
- For example, an employee works in at most one department at one location.
  - employee is a key in (employee, department, location)

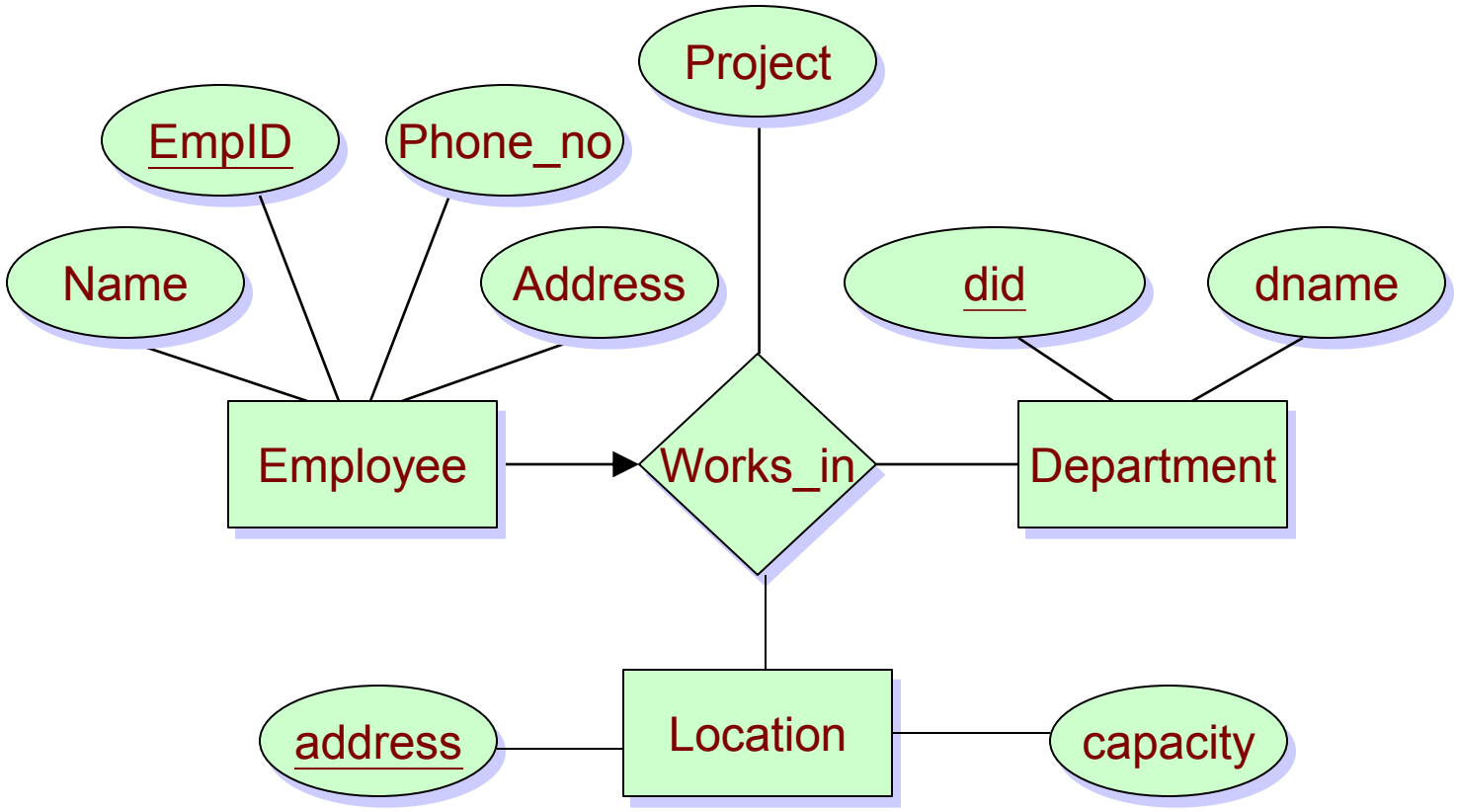


- What is the key for this relationship?



- Can an employee work in two locations for the same department? (Can she work in two locations?)
- Can an employee work in two projects for the same department? (Can he work in two projects?)






- Given (emp1, dept1, location1, proj1)
- (emp1, dept1, location2, proj1) ✗
- (emp1, dept2, location1, proj1) ✗
- (emp1, dept1, location1, proj2) ✗

# Outline

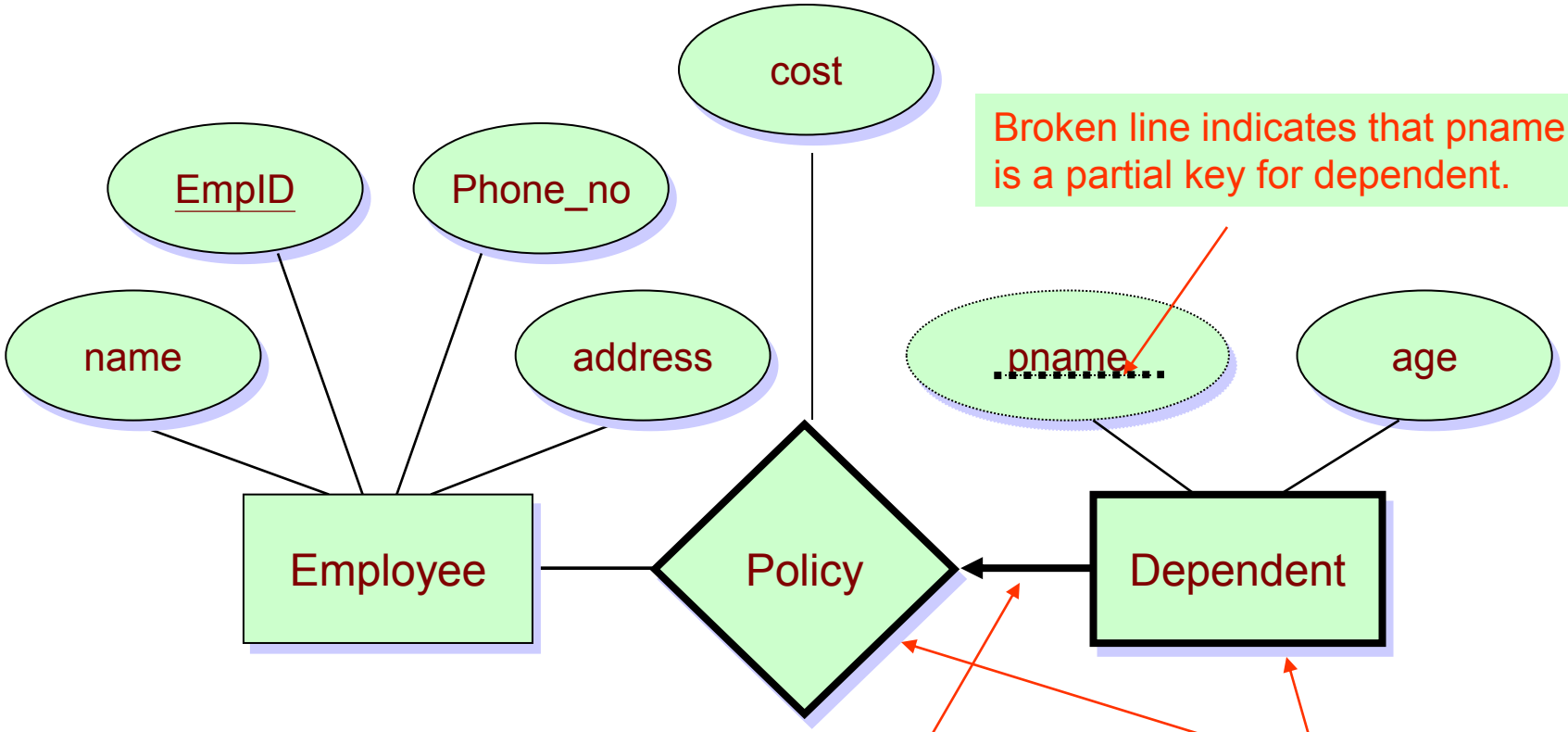
- Database Design
- Entity
- Relationship
  - Binary relationship
  - Non-binary relationship
- **Weak Entity/Strong Entity**
- Class Hierarchy
- Aggregation
- Conceptual Design with the E-R Model

# Weak Entities

- **Strong Entity**
  - An entity can be ***uniquely*** identified by some attributes related to this entity
  - E.g., Employee has an attribute EmpID (which can be used to uniquely identify each employee)
- **Weak Entity**
  - An entity ***cannot be uniquely*** identified by all attributes related to this entity

# Example: Weak Entities

- Suppose *employees* can purchase insurance policies to cover their *dependents*.
- The attribute of the *dependents* entity set are *pname* and *age*
- The attribute *pname* *cannot* uniquely identify a *dependent*
- *Dependent* is a weak entity set. since dependents come from employees
- A *dependent* can only be identified by considering some of its attributes in conjunction with the *primary key* of *employee* (***identifying entity set***).
- The set of attributes that uniquely identify a weak entity for ***a given*** owner entity is called a ***discriminator*** or ***partial key***.

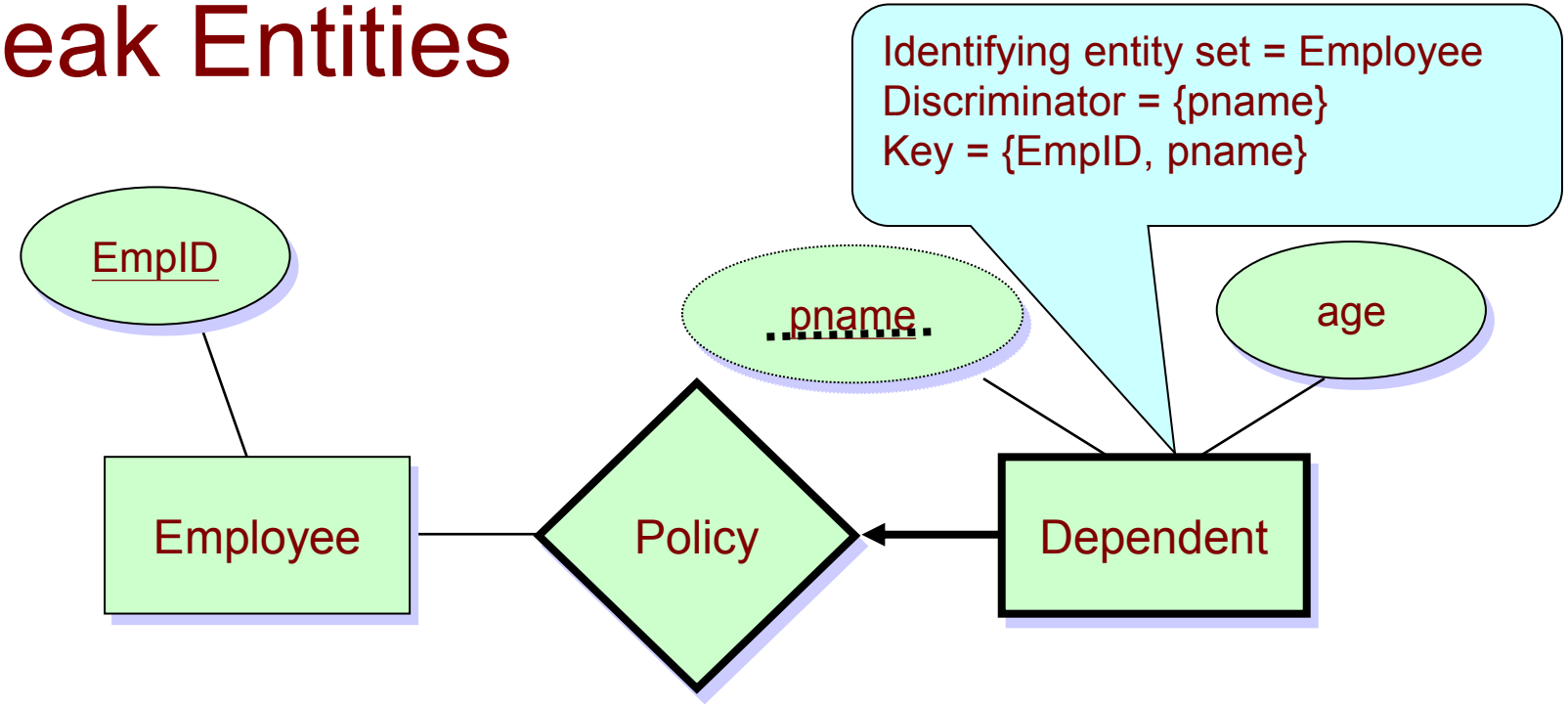



Broken line indicates that pname is a partial key for dependent.

The arrow from Dependent to Policy indicates that each Dependent entity appears in at most one Policy relationship. The arrow is thick because of the total participation constraint.

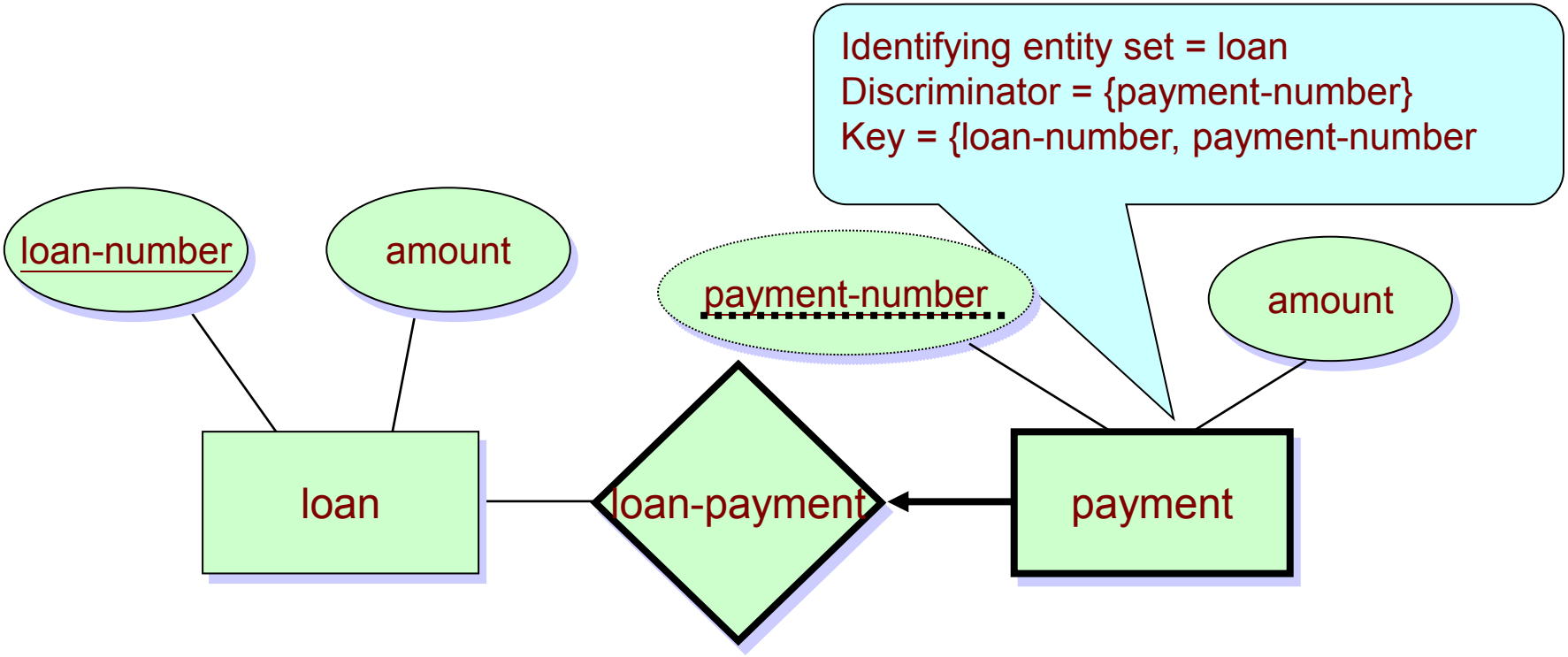
To underscore the fact that Dependent is a weak entity and Policy is its identifying relationship, we draw both with dark lines.

# Weak Entities




- Definition: If a weak entity set W is dependent on a strong entity set E, we say that E owns W.
- A dependent cannot be uniquely identified by “pname”.
  - E.g., Employee owns Dependent

# Example: Weak Entities




- A payment itself cannot be identified by “payment-number”
- loan owns payment

# More about Weak Entity Set

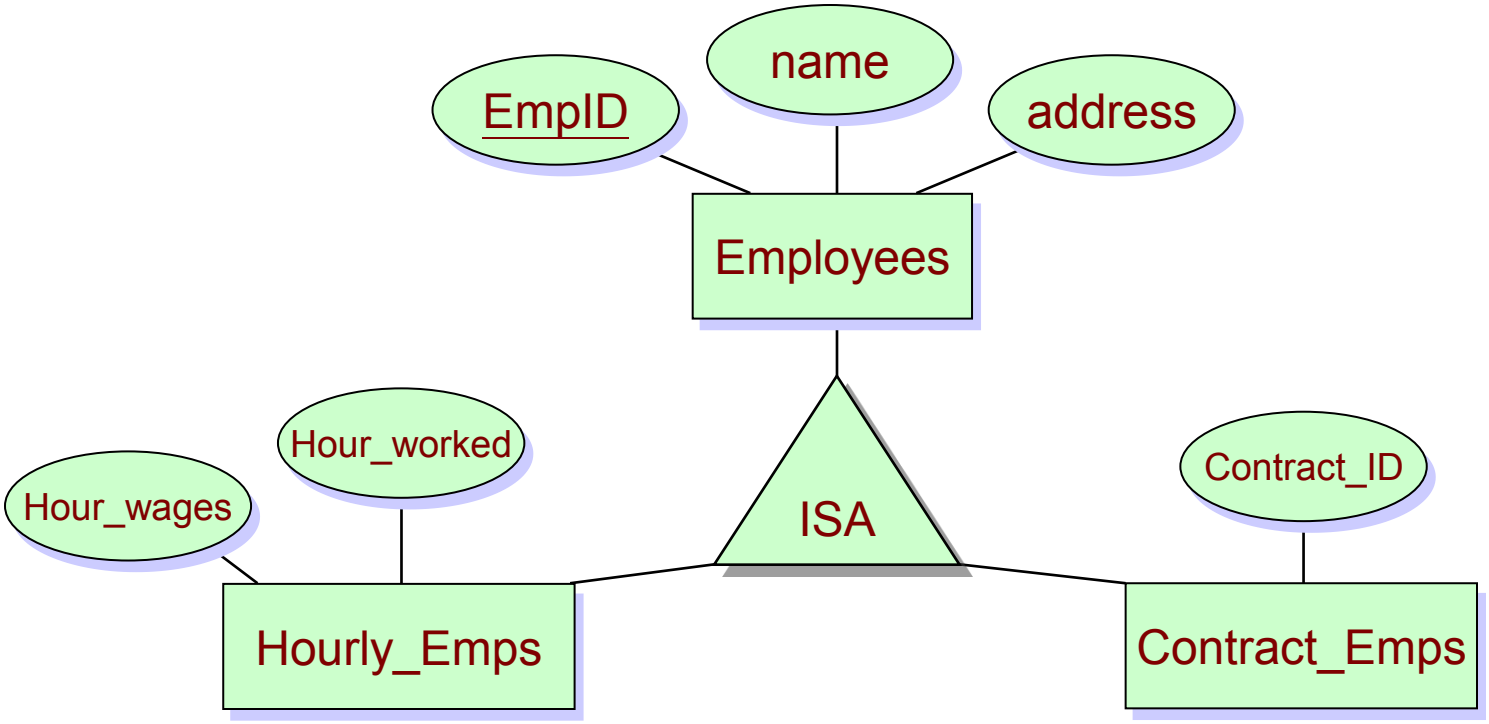
- A **weak entity set** can be identified uniquely only by considering the primary key of another (owner) entity set
  - Owner entity set and weak entity set must participate in ***one-to-many*** relationship set (one owner, many weak entities).
  - Weak entity set must have ***total participation***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

# Outline

- Database Design
- Entity
- Relationship
  - Binary relationship
  - Non-binary relationship
- Weak Entity/Strong Entity
- **Class Hierarchy**
- Aggregation
- Conceptual Design with the E-R Model

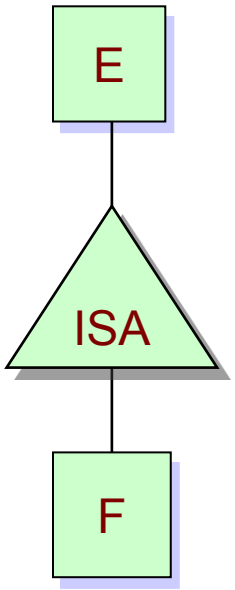
# Class Hierarchy

- Sometimes, it is natural to classify the entities in an entity set into subclasses



# Generalisation / Specialisation

- Arranging of entity types in a type hierarchy.
  - Determine entity types whose set of properties are actual a subset of another entity type.
- Definition Generalisation / Specialisation / Inheritance:
  - Two entity types E and F are in an ISA-relationship ("F is a E"), if
    - (1) the set of attributes of F is a superset of the set of attributes of E, and
    - (2) the entity set F is a subset of the entity set of E ("each f is an e")



# Generalisation / Specialisation

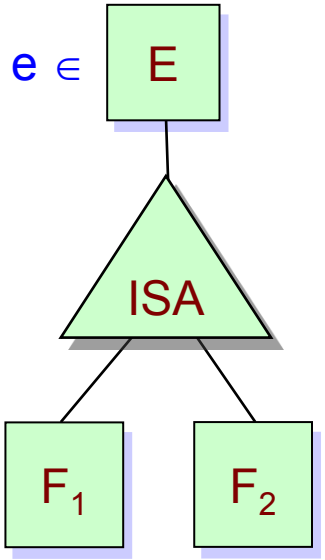
- One says that F is a specialisation of E (F is subclass) and E is a generalisation of F (E is superclass).
- Attribute inheritance – a lower-level entity type inherits all the attributes and relationship participations of its supertype.
- For example, the attributes defined for an Hourly_Emps entity are the attributes for Employees plus that of Hourly_Emps

# Constraints on ISA Hierarchies

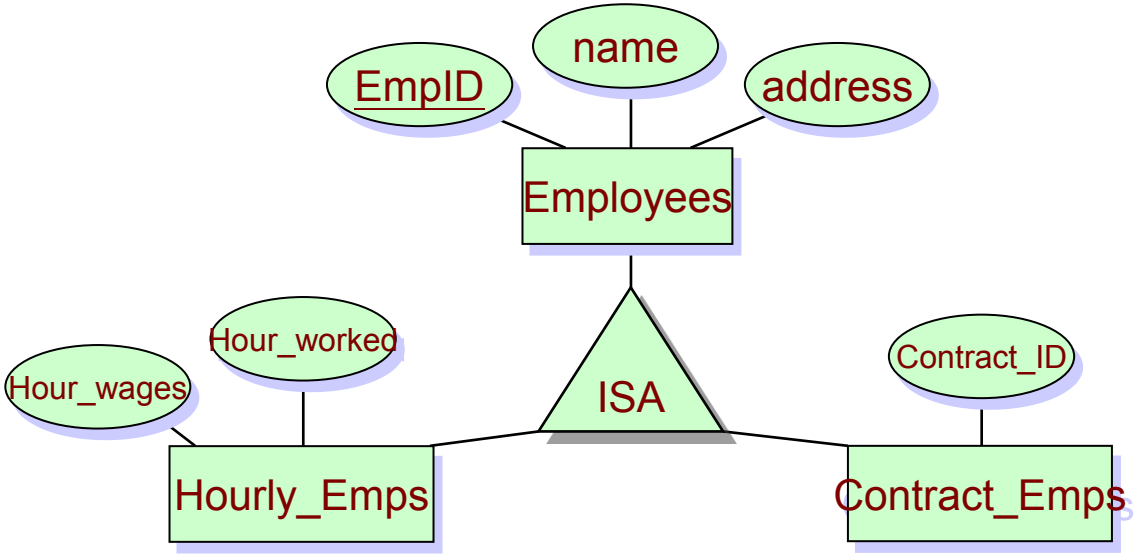
- ***Overlap Constraints*** - Constraint on whether or not entities may belong to more than one lower-level entity set
  - **Disjoint**
    - an entity can belong to **only one** lower-level entity set
  - **Overlapping**
    - an entity can belong to **more than one** lower-level entity set
- ***Covering Constraints*** - Whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets
  - **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

# Constraints on ISA Hierarchies

| Constraints | | Overlap: DISJOINT | Overlap: OVERLAPPING |
|---|---|---|---|
| Cover | *PARTIAL* | • Superclass has optional subclass, superclass may be a member of only one subclass<br>• Subclass sets are unique | • Superclass has optional subclass, superclass may be a member of at least one subclass<br>• Subclass sets are not unique |
| | *TOTAL* | • Every superclass occurrence is a member of only one subclass<br>• Subclass sets are unique | • Every superclass occurrence is a member of at least one subclass<br>• Subclass sets are not unique |




| Constraints | | Overlap: DISJOINT | Overlap: OVERLAPPING |
|---|---|---|---|
| Cover | *PARTIAL* | • e may be $\notin F_1 \cup F_2$<br>• $F_1 \cap F_2 = \emptyset$ | • e may be $\notin F_1 \cup F_2$<br>• $F_1 \cap F_2$ may be $\neq \emptyset$ |
| | *TOTAL* | • e must be $\in F_1 \cup F_2$<br>• $F_1 \cap F_2 = \emptyset$ | • e must be $\in F_1 \cup F_2$<br>• $F_1 \cap F_2$ may be $\neq \emptyset$ |




- We can specify two kinds of constraints with respect to ISA hierarchies
  - **Overlap constraints**: Can an employee be an Hourly_Emps as well as a Contract_emps entity? If so, specify => Hourly_Emps OVERLAPS Contract_Emps.
  - **Covering constraints**: Does every Employees' entity also have to be an Hourly_Emps or a Contract_Emps entity? If so, write Hourly_Emps AND Contract_Emps COVER Employees.

# Class Hierarchy

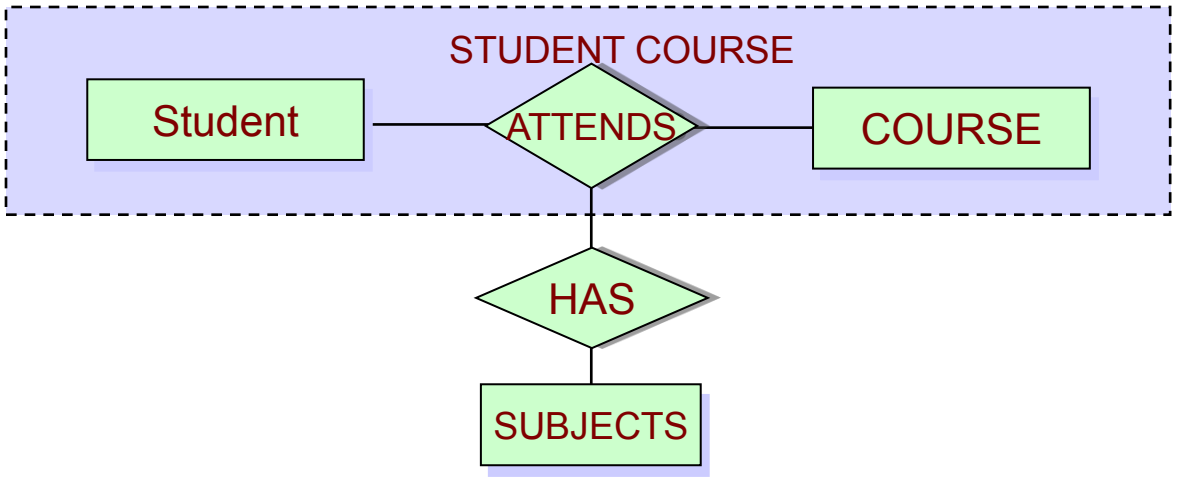
- Reason why we use class hierarchy
  - Add ***descriptive attribute*** that make sense only for the entities in a subclass
    - Hourly_wages does not make sense for a Contract_Emps entity.
  - Identify the set of entities that participate in some relationships
    - We may want to have a relationship called “Bonus” with Contract_Emps (not Hourly_Emps)

# Outline

- Database Design
- Entity
- Relationship
  - Binary relationship
  - Non-binary relationship
- Weak Entity/Strong Entity
- Class Hierarchy
- **Aggregation**
- Conceptual Design with the E-R Model

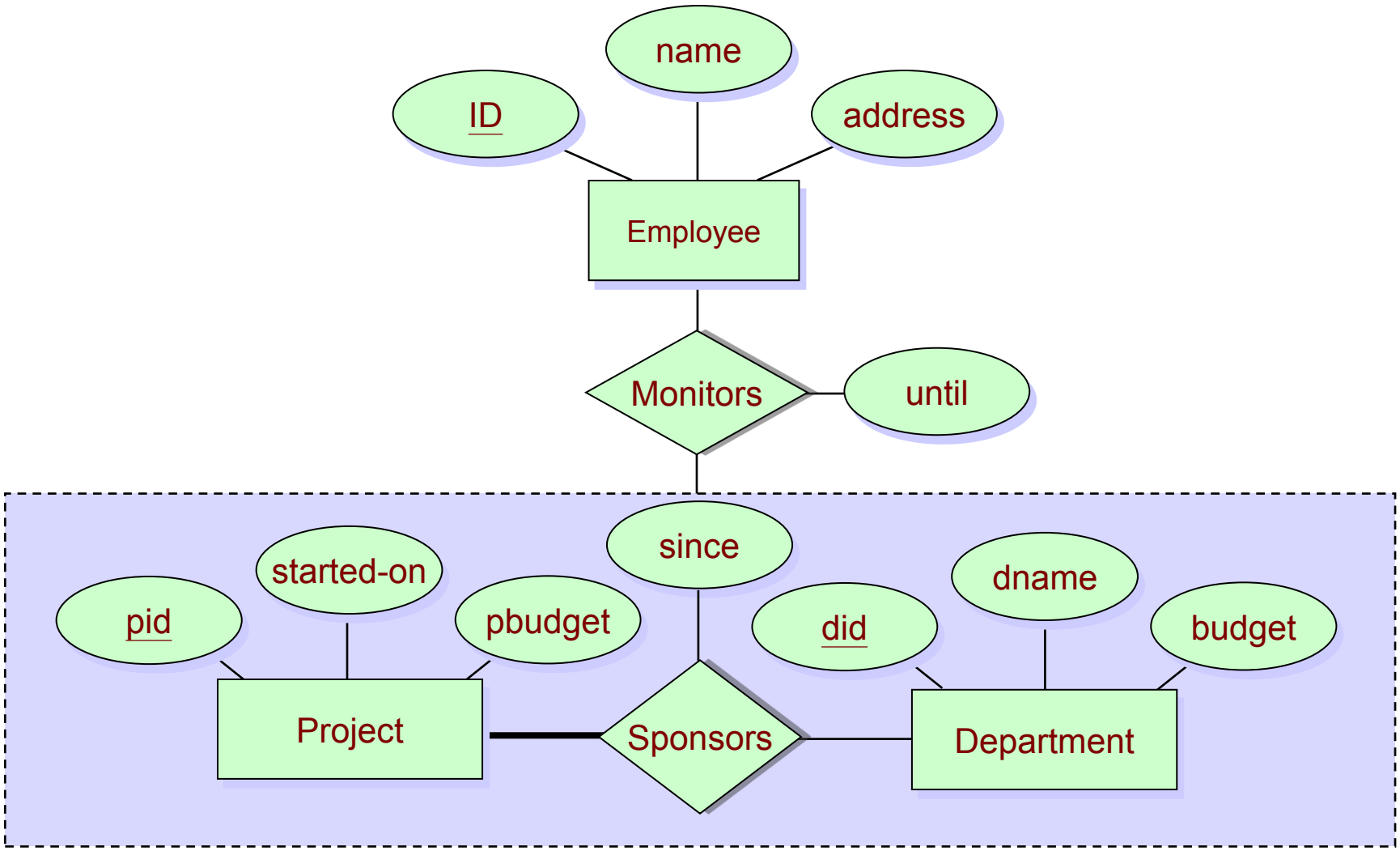
# Aggregation → represent the relationship between two relation-ships

- The E-R model cannot express relationships among relationships.
- An abstraction through which relationships are treated as higher-level entities.
- Aggregation is a process when relation between two entity is treated as a single entity.



# Example: Aggregation

- Consider an entity set called project.
- Each project entity is sponsored by one or more departments.
- A department that sponsors a project might assign employees to monitor the sponsorship.
- Intuitively, **monitors** should be a relationship set that associates a Sponsors relationship (rather than a project or department entity) with an Employee entity.


name
ID
address
Employee
Monitors
until
since
started-on
pid
pbudget
dname
did
budget
Project
Sponsors
Department

# Outline

- Database Design
- Entity
- Relationship
  - Binary relationship
  - Non-binary relationship
- Weak Entity/Strong Entity
- Class Hierarchy
- Aggregation
- **Conceptual Design with the E-R Model**

# Conceptual Design with the E-R Model

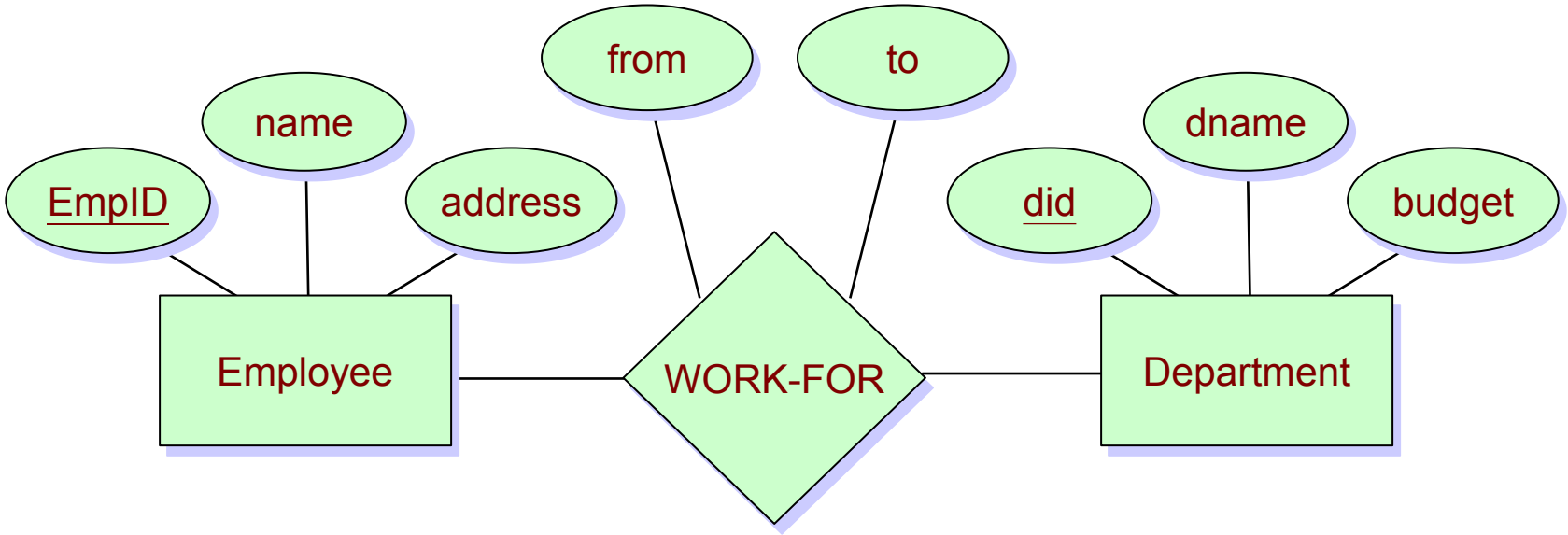
- Developing an ER diagram presents several choices, including the following:
  - Should a concept be modeled as an entity or an attribute?
  - Concept modeled as an entity or a relationship?
  - What are the relationship sets and their participating entity sets?
  - Should we use binary or ternary relationships?
  - Should we use aggregation?

# Entity versus Attribute

- Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?

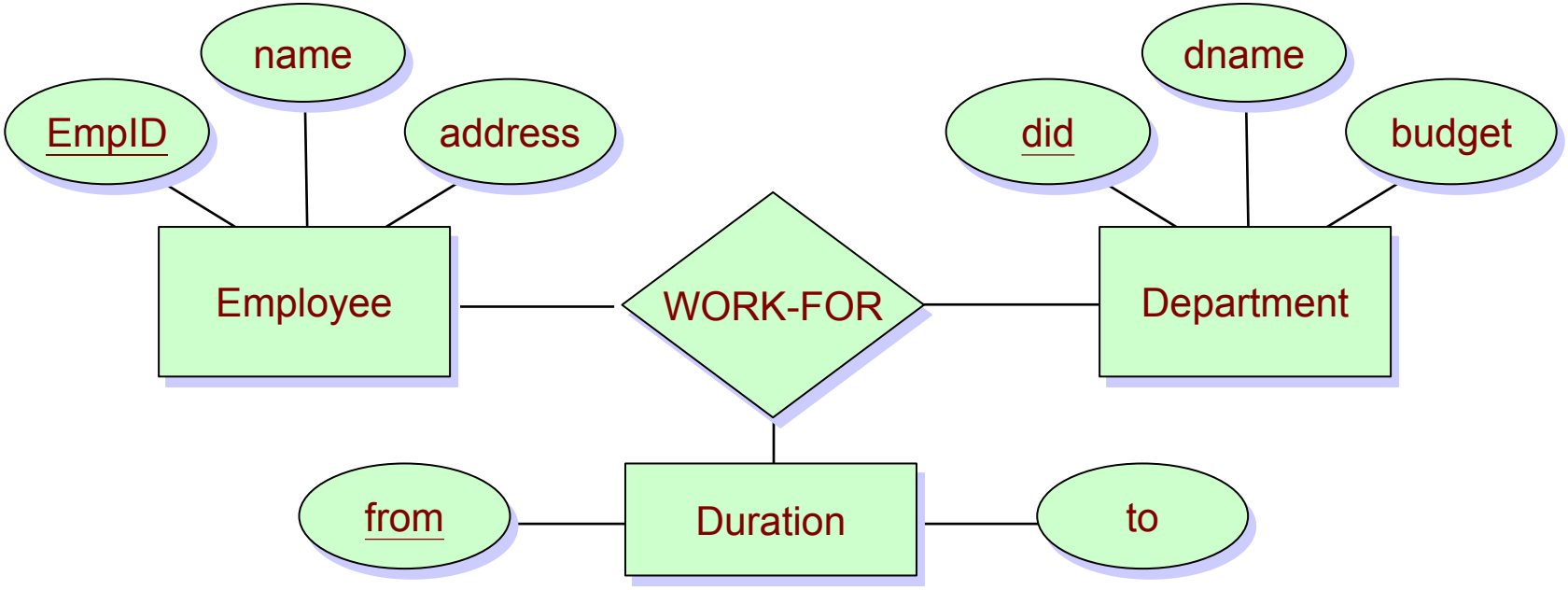
1. If we may have several addresses per employee, address can be an entity (if attributes cannot be allow multi-valued)
2. If the structure (city, street, etc) is important, e.g. we want to retrieve employees in a given city, address must be modeled as an entity (attribute values are atomic)

- Should duration of an employee working in a department be an attribute or an entity?



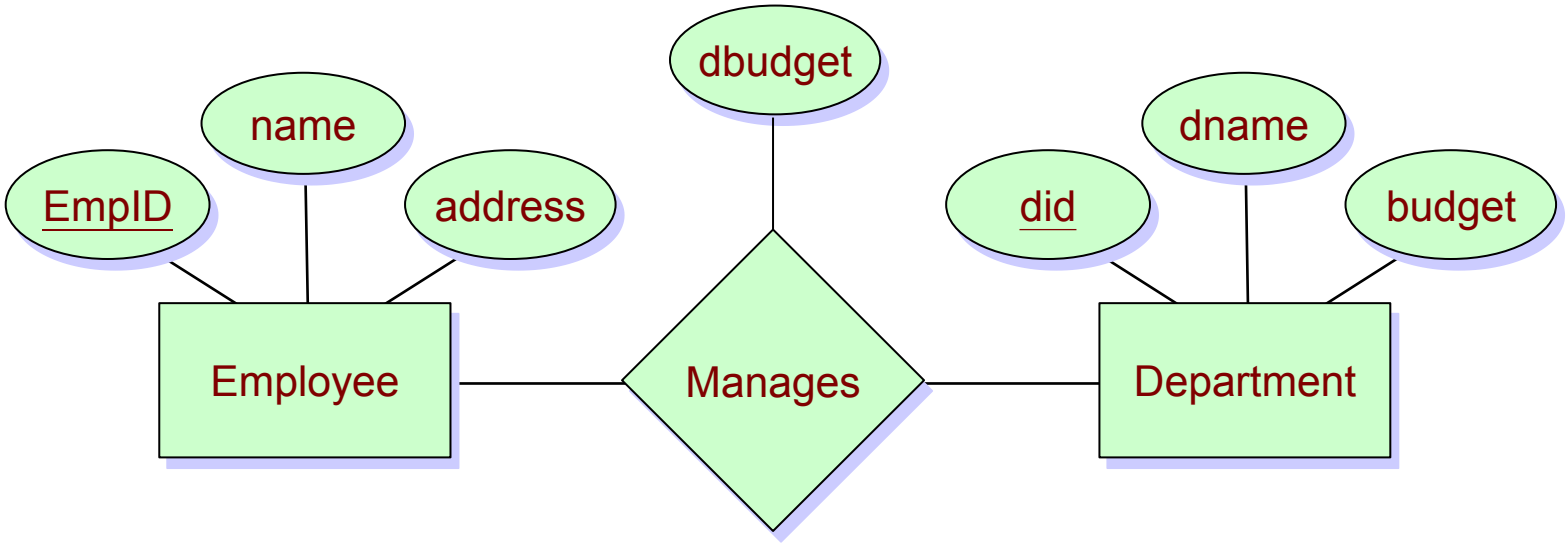

- This ER model does not allow an employee to work in a department for *two or more* periods.
  - Because a relationship is ***uniquely*** identified by the participating entities.

- The problem can be addressed by introducing an entity set called Duration.

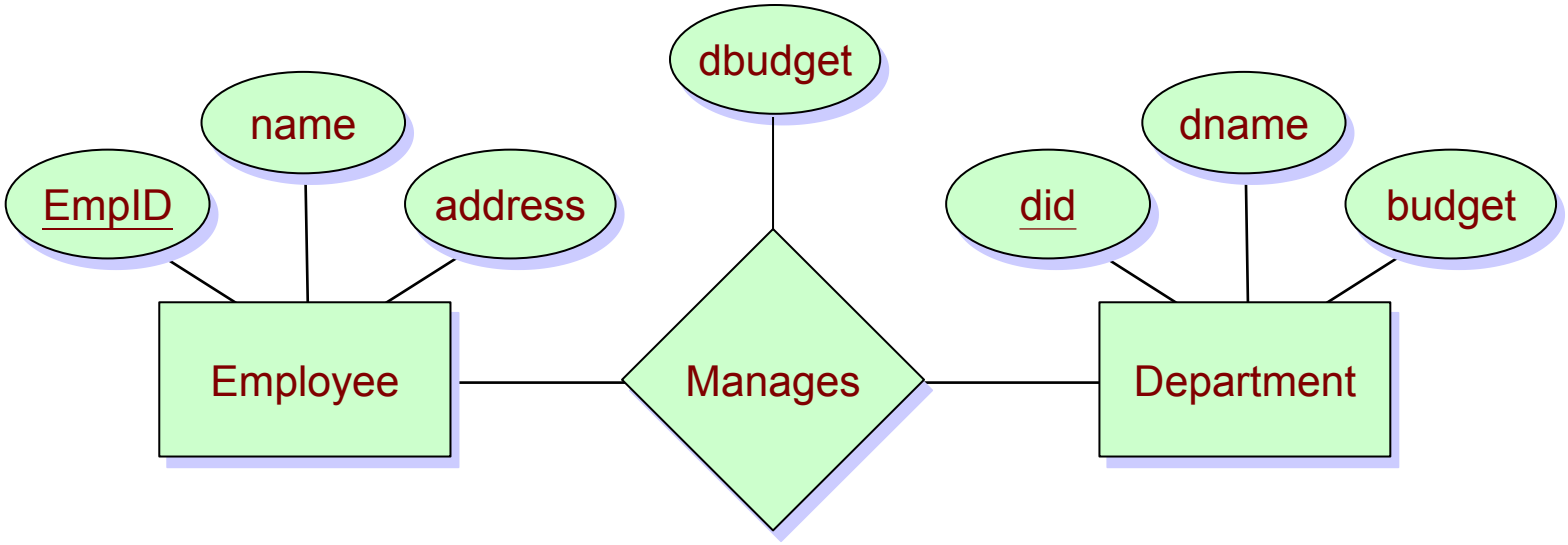


# Entity versus Relationship

- Suppose each department manager is given a budget (dbudget).

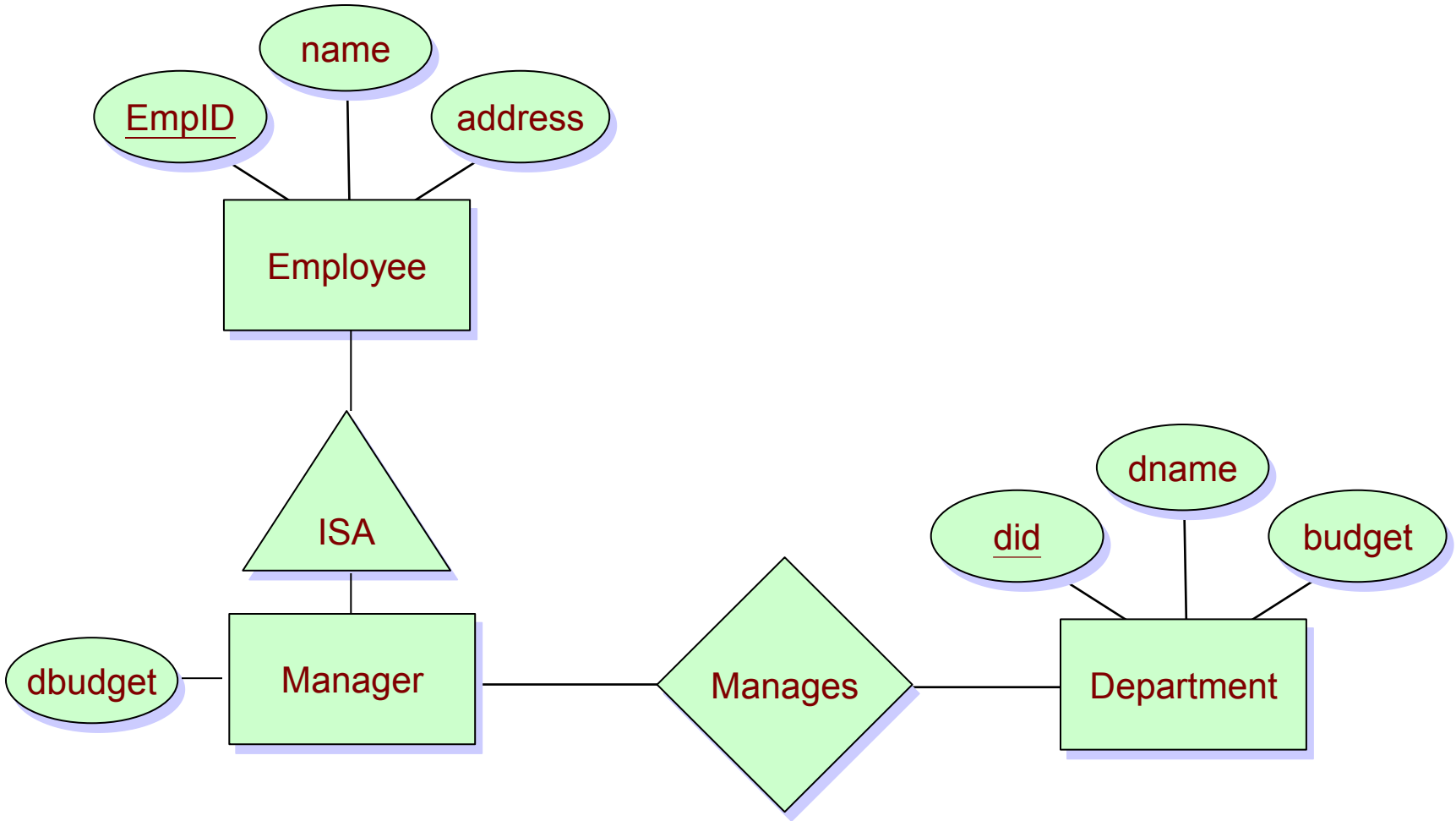



- This approach is natural if we assume that a manager receives a separate budget for each department.

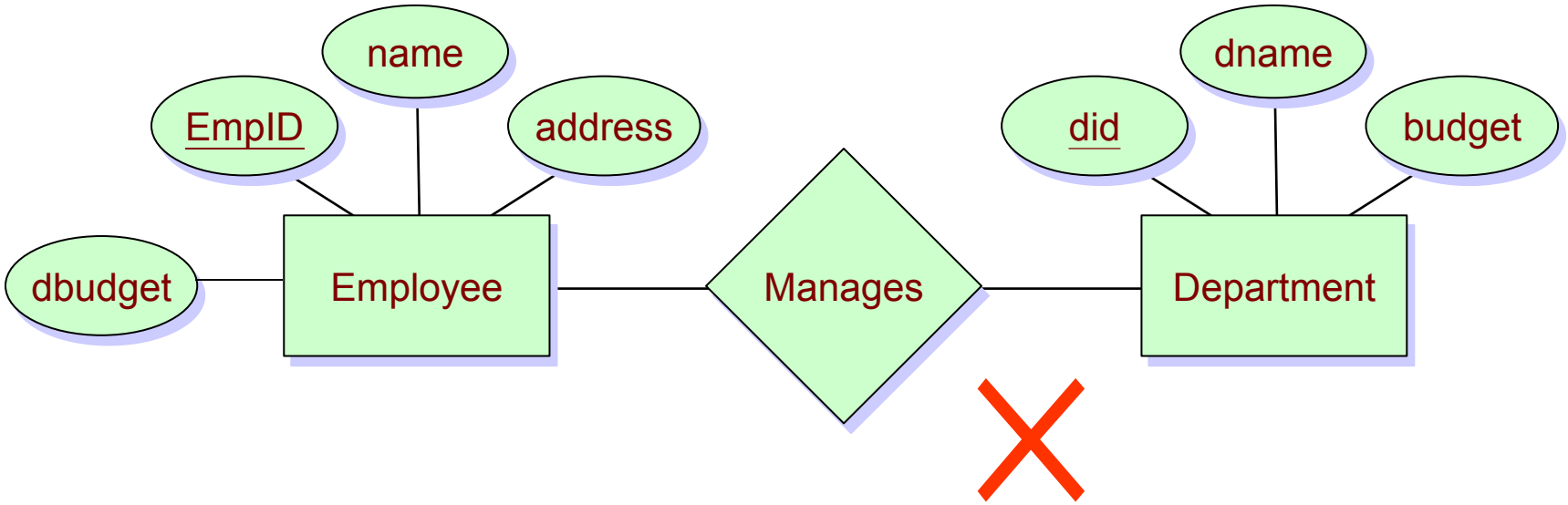



- What if the budget is a ***sum*** that covers all departments managed by that employee?
  - In this case, each manages relationship that involves a given employee will have the same value in the dbudget field, leading to redundant storage of the same information.
    - (Tim, dept A,  $200000 ) (Tim, dept B, $200000 )
  - It is also misleading; it suggests that the budget is associated with the relationship, when it is actually associated with the manager.

- We can address the problem by introducing a new entity set called manager with the ISA hierarchy.

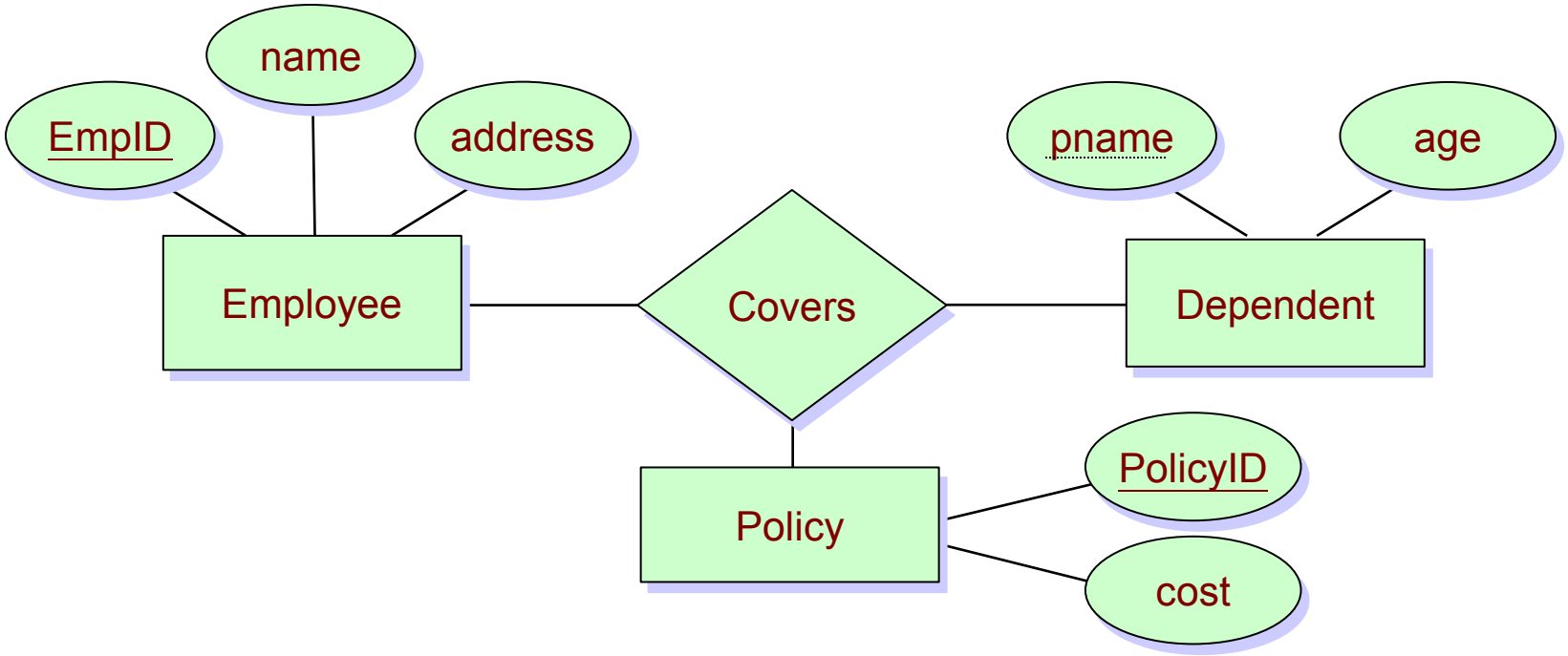


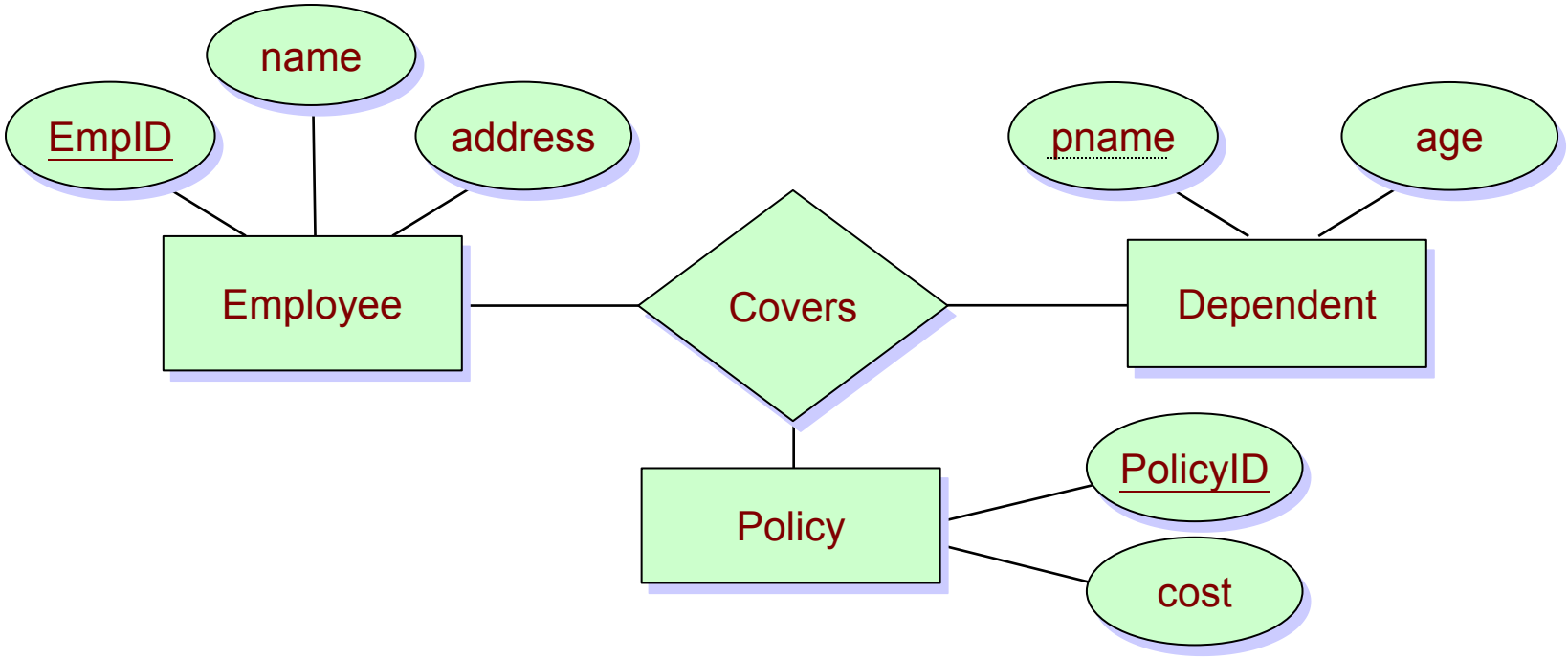
- Why not simply make the budget an attribute of Employee ?



# Binary versus Ternary Relationships

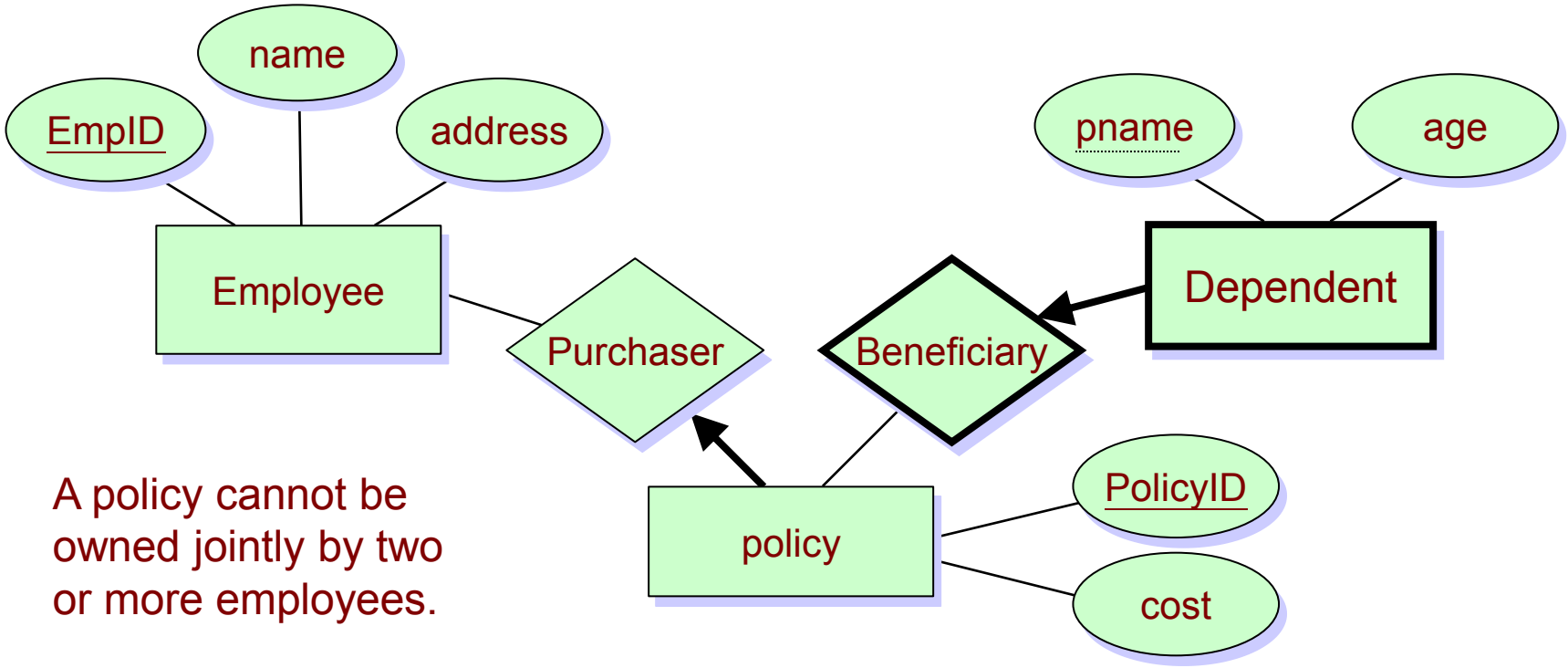
- The following ER diagram models the situation that an employee can own several policies, each policy can be owned by several employees, and each dependent can be covered by several policies.






- Suppose we have the following additional requirements:
  - Every policy **MUST BE** owned by some employee. (Impose a total participation constraint on Policy?).
  - Dependents is a **WEAK ENTITY SET**, and each dependent entity is uniquely identified by (pname, policyID).
  - A policy **CANNOT BE** owned by two or more employees. (Impose a key constraint on Policy?).

- Here is a solution

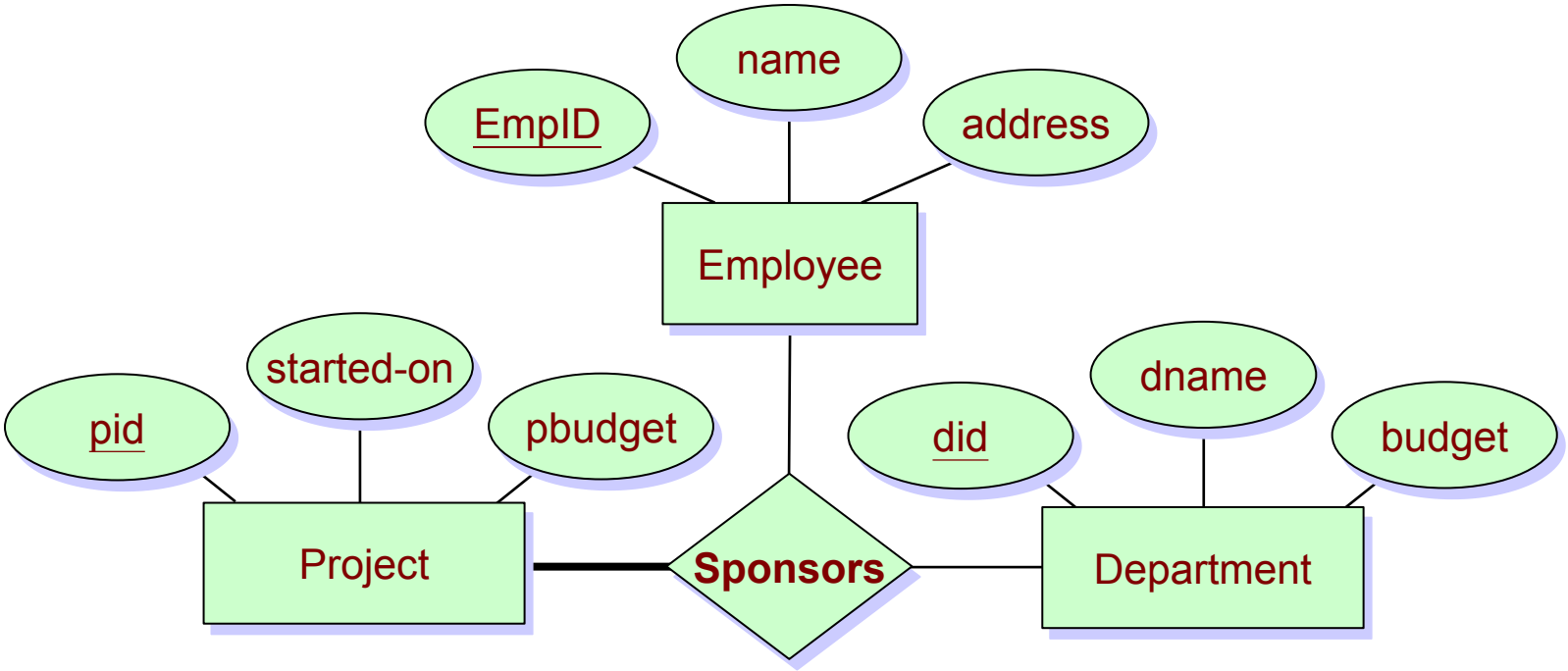



A policy cannot be owned jointly by two or more employees.

Every policy must be owned by some employee.

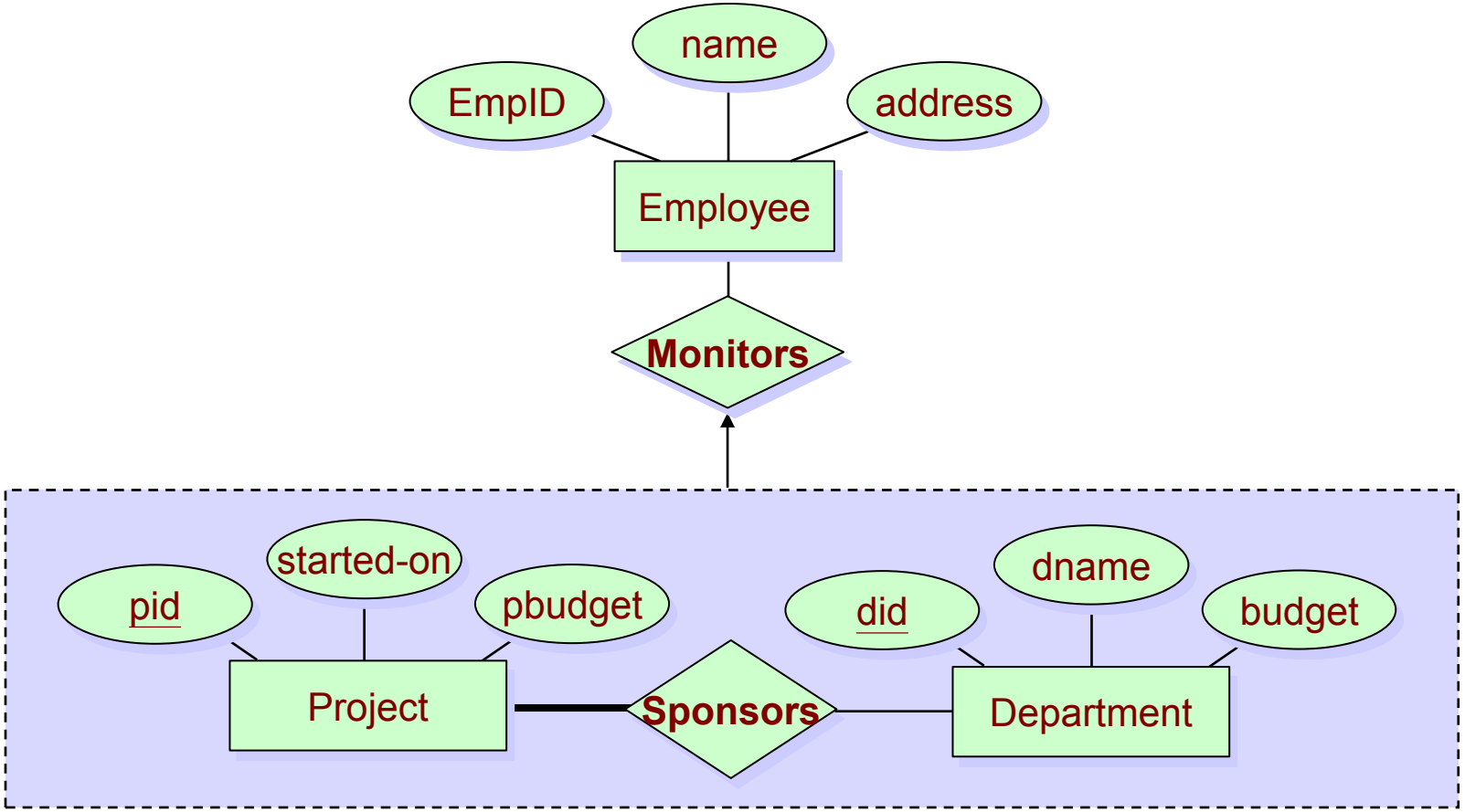
Dependent is a weak entity set, each dependent entity is uniquely identified by (pname, policyID).

# Aggregation versus Ternary Relationships




- However, suppose we have a constraint that each sponsorship can be monitored by at *most one* employee.

- If aggregation is used, we can draw an arrow from the aggregated relationship sponsors to the relationship monitors.



# Summary

- Entities
- Relationships
- Constraints
- Weak Entity/Strong Entity
- Class Hierarchy
- Aggregation
- Textbook Ch. 2