Database Design

Entity-relationship Model
w2_wed

Simple Attributes

- Attributes may be **simple (atomic).**
  - For example:
    - Entity = Student
    - Attributes = Student number, name…
  - Student A:
    - Student number = z12345678
    - Name = Bob
    - …

Entity-Relationship Model

- Attributes can also be **multi-valued**
  - ***Multivalued***: has more than one value
  - No longer a simple attribute with only one value.
- Example: not everything is a single solid color

Problem

- **Q: “Do we need multi-value attributes?”**
- The attributes you design should be able to describe, and the combinations of all its instances should be able to express the entity faithfully

Multivalued Attribute

Composite Attributes

- What is a simple attribute? Attributes that are not divisible are called *simple* or *atomic attributes*.
- **Composite attributes** can be divided into smaller subparts, which represent more basic attributes with independent meanings.

Question:

- Some semantics cannot be faithfully captured using atomic attributes
- Question: is *Address* a simple attribute/value?
  - *Address = 'Computer Science Building (K17), Engineering Rd, UNSW Sydney, Kensington NSW 2052"*
- How can should we model Addresses ?

Question:

- Question: is *Address* a simple attribute or a composite attribute?
  - *Address = "Computer Science Building (K17), Engineering Rd, UNSW Sydney, Kensington NSW 2052"*

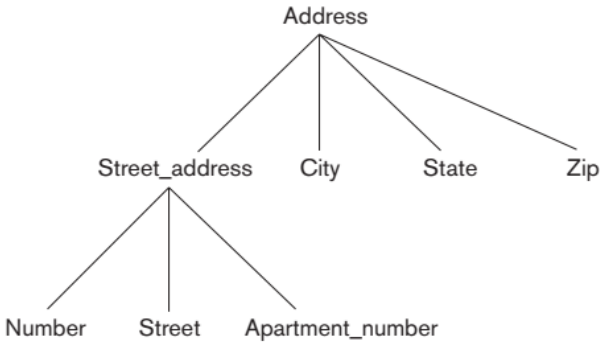


Figure 7.4
A hierarchy of composite attributes.

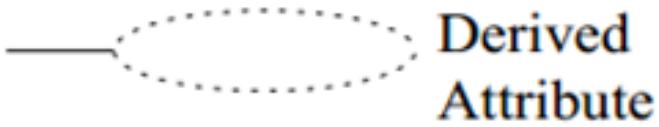
- We also avoided this possible situation:
  - Address = "that red house on parks street"

Question:

- Composite attributes are useful for situations when…
  - End-user sometimes refers to the composite attribute as a unit
  - But at other times refers specifically to its components.
- Question: Can't I just let my composite attributes, be simple attributes instead?

Derived Attributes

- Attributes that are problematic if modeled with a simple value.
- Scenario: modelling a person's age
  - This value can change and is dynamic.



Derived Attributes

- If we want to store *age* information, we model it as an **derived attribute,** we is said to be **derivable.**
  - We may store a separate attribute *Date-of-birth* attribute, called the **stored attribute** in rel. to *age*.
- Note: Where there is a derived attribute, there must also be an attribute where it's values can be derived from.
  - **Derived attribute** values are not stored
  - The **stored attribute** that derives the value is stored.

Take away

- Given an entity, the attributes you design should be able to describe, and the combinations of all its instances should be able to express the entity faithfully.
  - This is why attribute types such as multi-labelled attributes, composite attributes exist.
  - Can I have an entity to describe something abstract? Of course, entities don't have to concrete/ tangible objects. You will still need to find the right attributes to describe it.

Entity Instances/Facts

- A good entity should have good attributes that can be filled with good values.
- Car Entity:
  - ((Reg. Num, State), Make, Model, Year, {Colour})
- Car instance:
  - (CS9311, NSW), Toyota, Toyota Corolla, 1999, {White, Silver})

Take away

CAR

Registration (Number, State), Vehicle_id, Make, Model, Year, {Color}

CAR_1

((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})

CAR_2

((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})

CAR_3

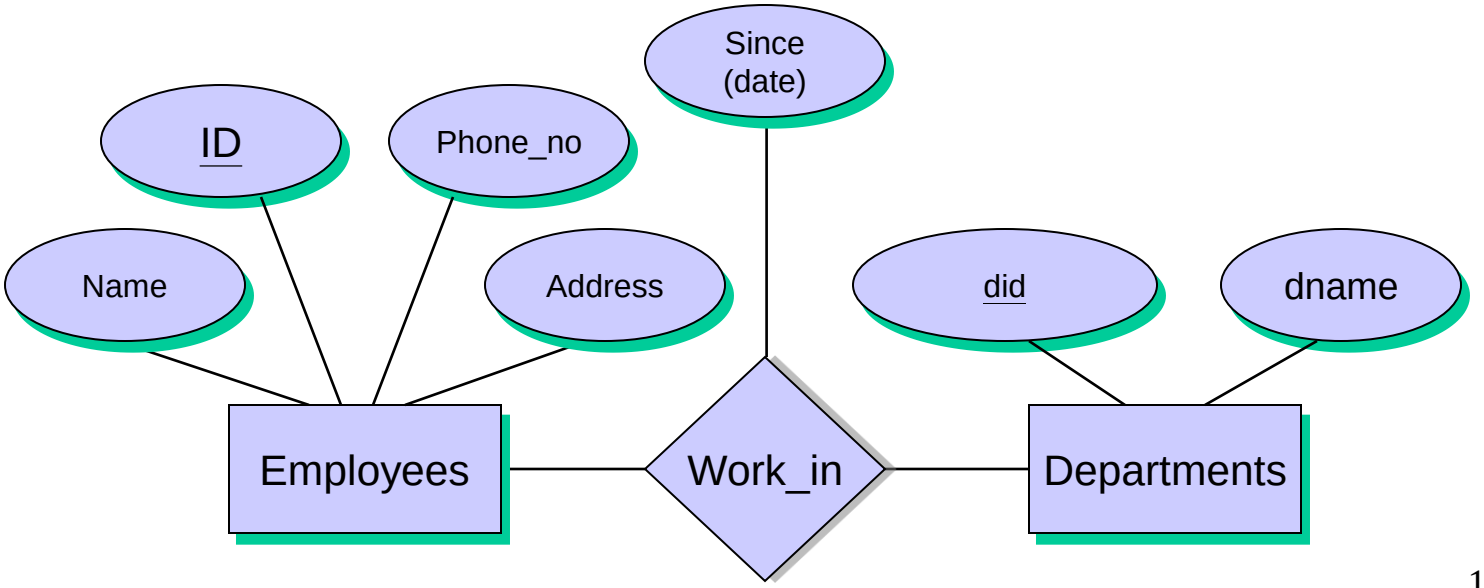
((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})

Take away

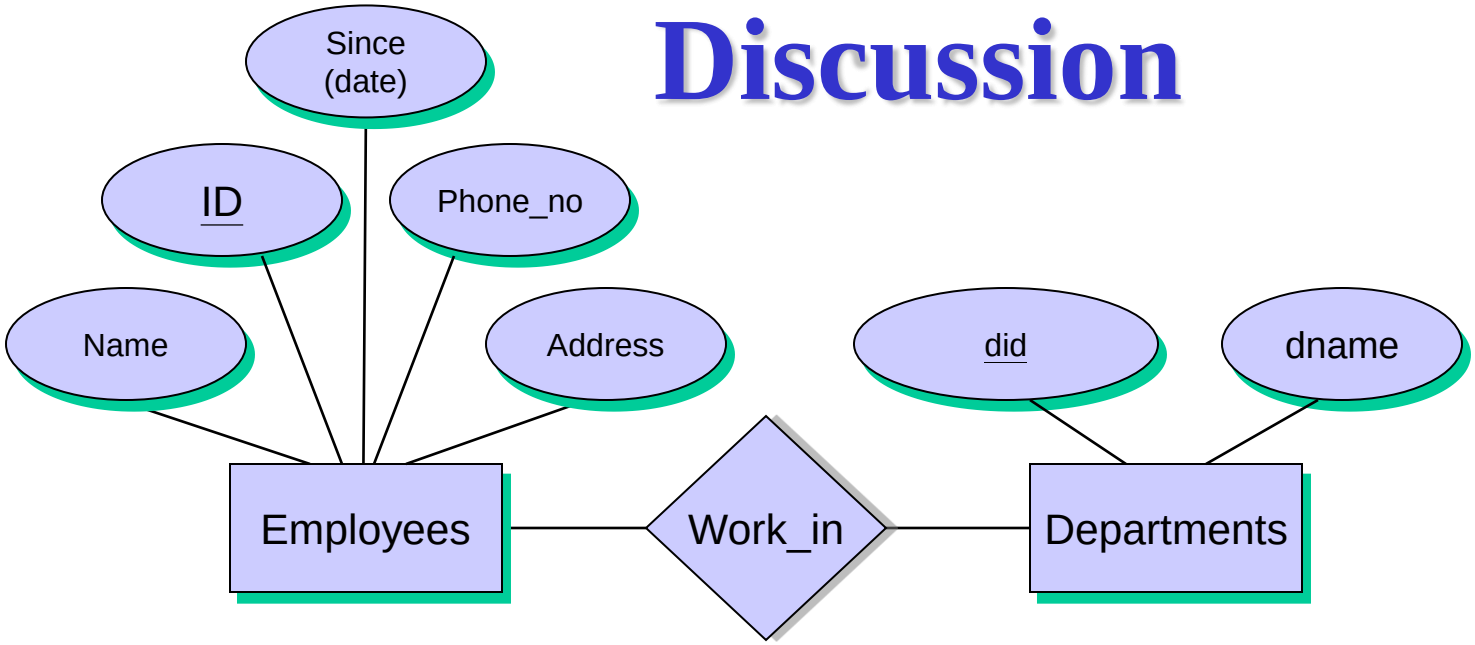
1. In the context of ER modelling, an entity type describes the **intension** for a *set of entities* that share the same structure.
2. An entity type is represented in ER diagrams as a rectangular box enclosing the entity type name.
3. The collection of entities of a particular entity type is grouped into an entity set, which is also called the **extension** of the entity type.

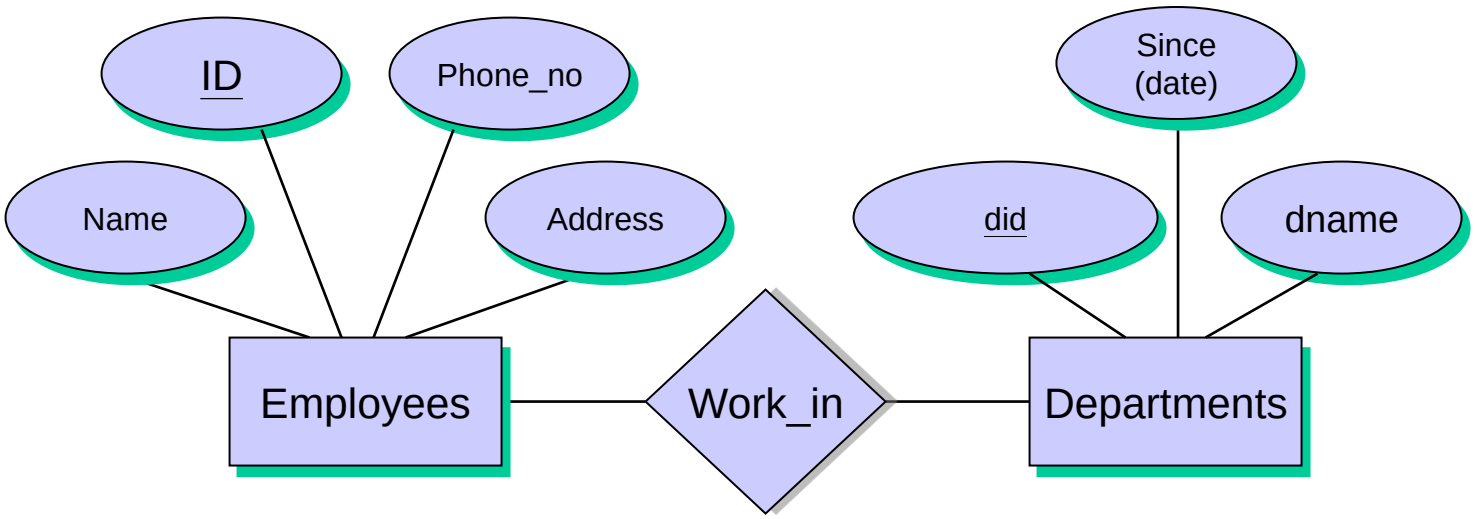
Relationship Attributes

- A relationship can also have descriptive attributes.
- Descriptive attributes are used to record information about the relationship, rather than about any one of the participating entities.



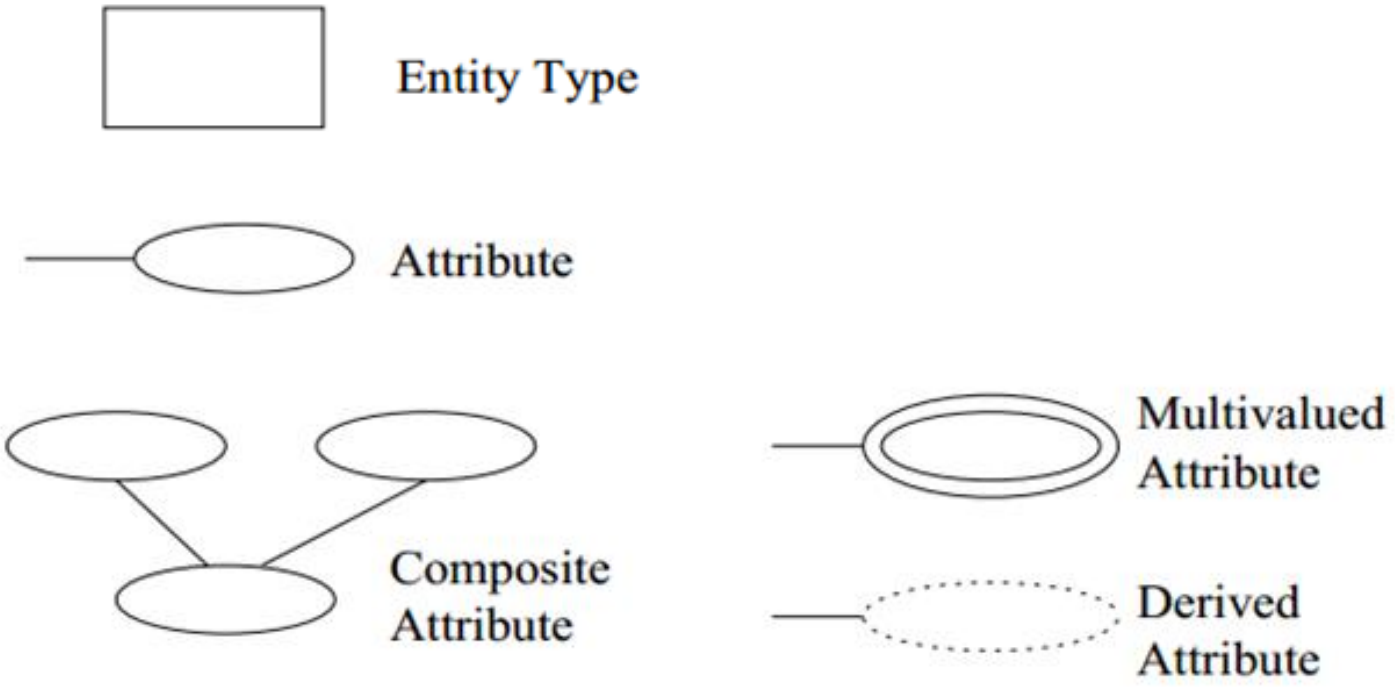
Discussion


Since
(date)
ID
Phone_no
Name
Address
did
dname
Employees
Work_in
Departments


ID
Phone_no
Since
(date)
Name
Address
did
dname
Employees
Work_in
Departments

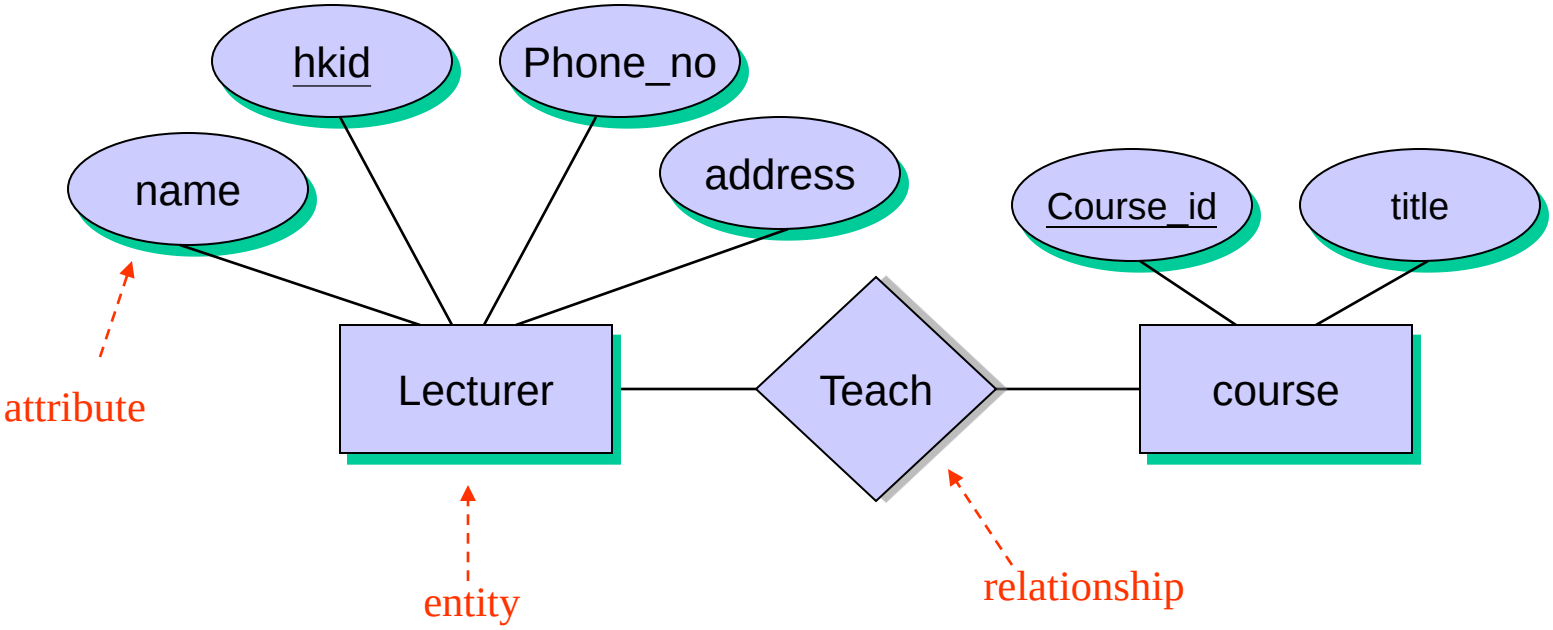
Visualizing an ER Data Model

- The new notations (so far):



Relationships

- A relationship set can also be represented by an E-R diagram.



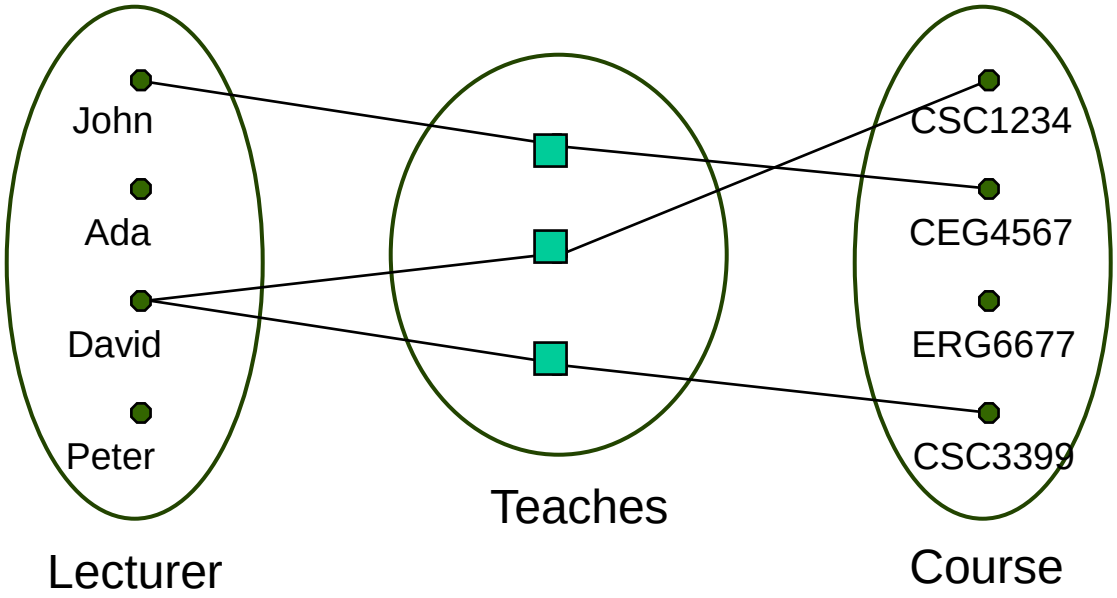
Relationships

- A relationship is an generally an natural association among two or more entity sets.

  Lecturer Set = {John, Ada, David, Peter}

  Course Set = {CSC1234, CEG4567, ERG6677, CSC3399}

  Teaches Set = {(John,CEG4567), (David, CSC1234),(David,CSC3399)}

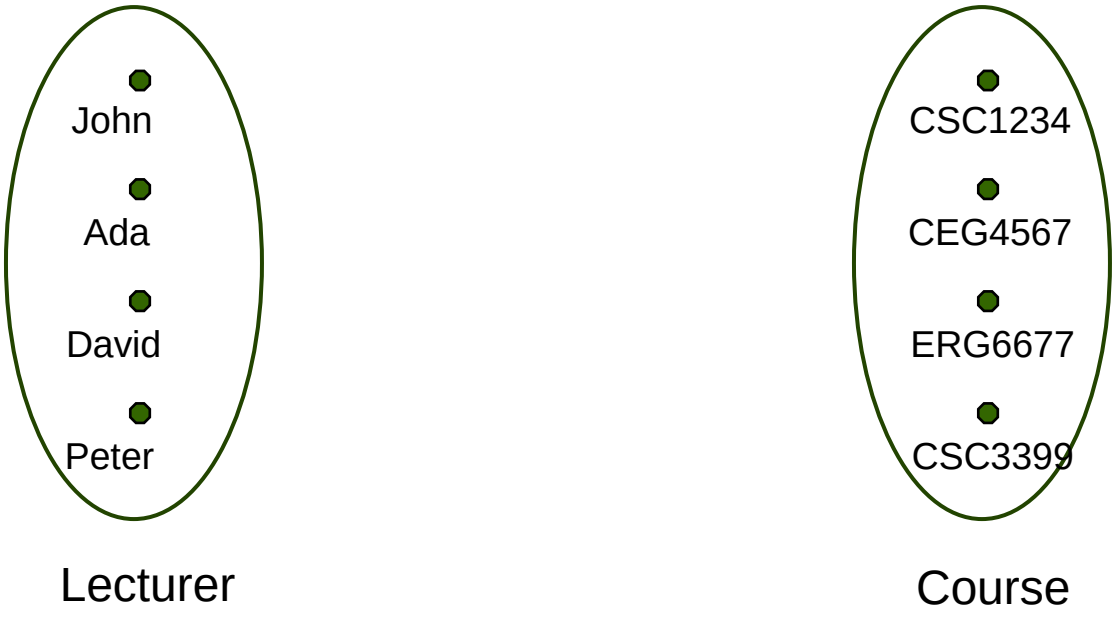


Relationships

- There is a need to storing this “association” between individual in relationships.

  Lecturer Set = {John, Ada, David, Peter}

  Course Set = {CSC1234, CEG4567, ERG6677, CSC3399}



A Relationship Set (0)

- The language is based around sets.
- **Entity Set :** An entity set is a collection or set of all entities of a particular entity type at any point in time. The type of all the entities should be the same.

A Relationship Set (1)

- The language is based around sets.
- The instances of the same type of relationship are described formally as a relationship **set**.
  - Teaches Set = {(John,CEG4567), (David, CSC1234),(David,CSC3399)}

- The instances of the same type of entity are described formally as an entity **set**.
  - *Lecturer Set* = {John, Ada, David, Peter}
  - *Course Set* = {CSC1234, CEG4567, ERG6677, CSC3399}

A Relationship Set (2)

- A relationship set can be thought of a set of **n-tuples**, defined below

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

- The teach relation can be represented by the set:
  - R1 = {(John,CEG4567), (David, CSC1234),(David,CSC3399)}

A **relationship set** is a subset of

$$E1 \times E2 \times \cdots \times En$$

- *n* = the **degree** of the relationship set

A Relationship Set (3)

- There is an underlying assumption is that all elements of a set are to be distinct.
- Meaning, each entity instances must be unique.
  - *How do we identify them?*
  - *There must be a key for all entities.*

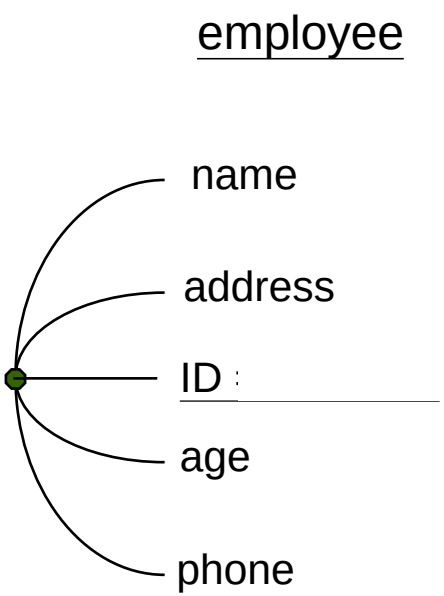
Keys

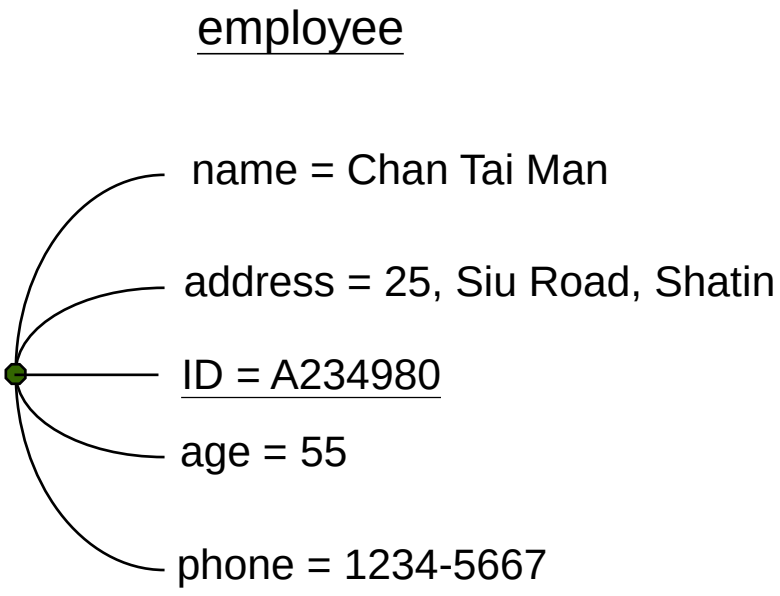
- ***Key constraint in ER Modelling:*** in any **extension** of the entity type, no two entities have the same values for their key attribute(s).

- For example:
  - {payroll number} is a key of RESEARCHER,
  - {car registration number} is a key of CAR.
  - Person (sex, name) will have instances: (Female, 23); (Male, 34); (Female, 23)

Keys

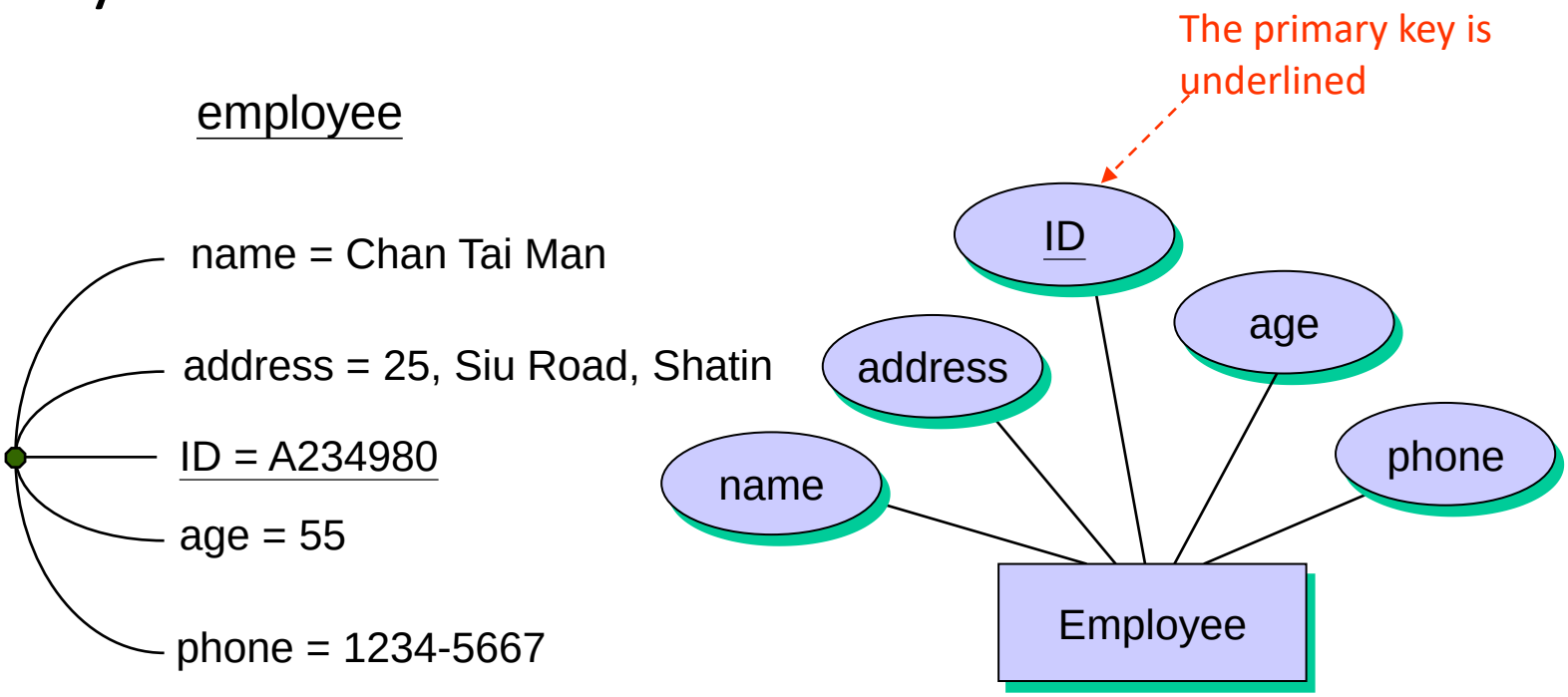
- Generally there includes an attribute as a key.



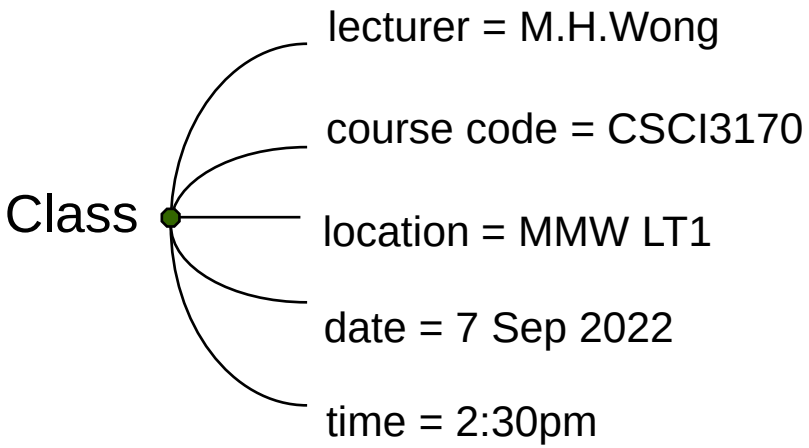


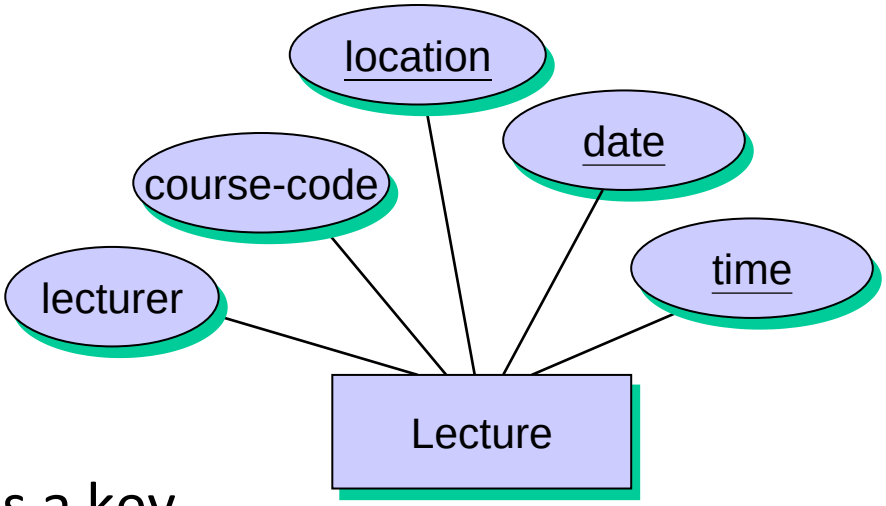
Keys

- Usually, we need to add an extra attribute as a key.



- A **key** may contain more than one attribute.





  - *{location, date, time}* is a key.
  - *{lecturer, date, time}* is also a key.
  - *{lecturer, location, date, time}* is not a key, but it is a super key.

- Keys containing multiple attributes as **composite keys** (aka **compound key**)

Check Semantics of Keys

- The semantics of the chosen key should take into realistic expectations of it being able to identify said entity across any possible extension.
- Possibility rather than on the current set of the data.
  - E.g., in the a database which contains only two employees, the age might be able to distinguish each employee. However, we may get a new employee with the same age as an existing employee.

Key

- A **key** may contain more than one attribute.
  - *{location, date, time}* is a key.
  - *{lecturer, date, time}* is also a key.
  - *{lecturer, location, date, time}* is not a key, but it is a super key.

- Keys containing multiple attributes as **composite keys** (aka **compound key**)
  - A ***super key*** is a non-minimal set of attributes that uniquely identify any entity instance of an entity type

Key

- In real-life, there is likely to very be than one valid possible keys, there can be many candidate keys.
  - See intension for *Person: (DNA Sequence, Passport Number)*

- A **candidate** key is selected by the designer to uniquely identify tuples in a table, that selected candidate key is the **primary** key.

Key

- *Q: Technically, can an key be all an entites attributes?*
- A: Since the extension is a **set**, in the worst case scenario, the set of all it's attributes together should able to uniquely identify its values (strictly speaking), and becomes as the natural composite key.
- **Every entity must have a final primary key: a minimal set of attributes that can uniquely identity its entity instances, and must not be NULL.**

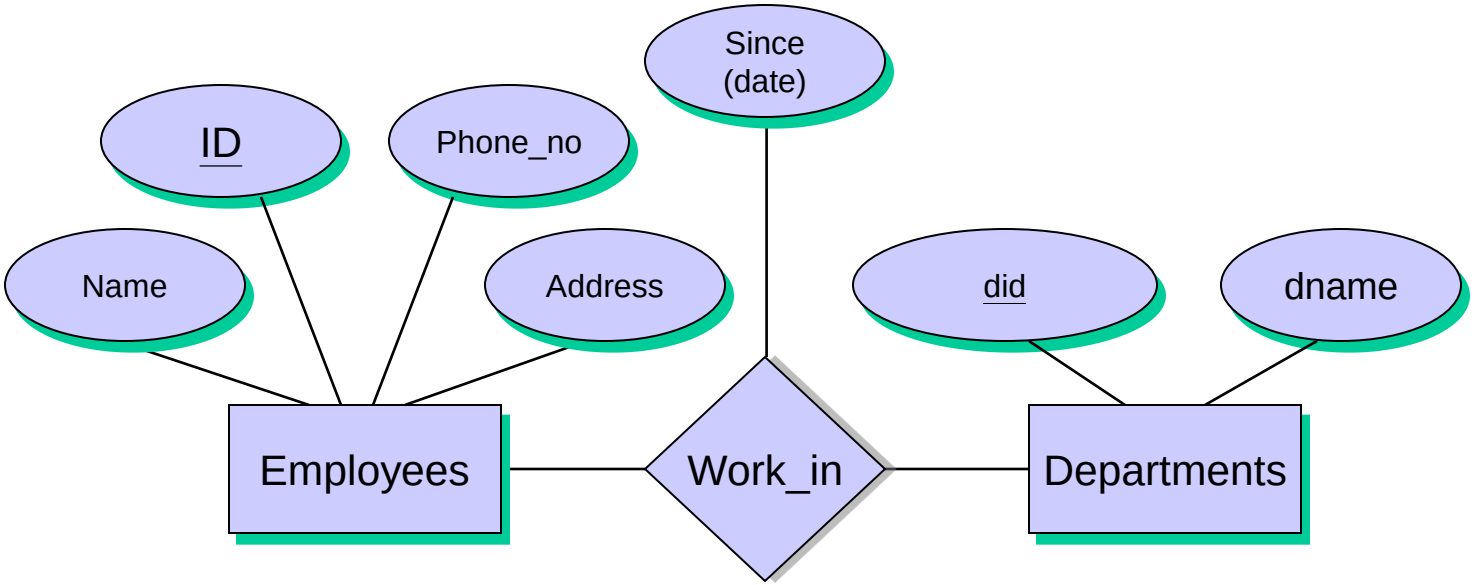
"Key" Summary

- A superkey is any set of attributes which can uniquely identify an entity.
- A key is a **minimal** set of attributes whose values uniquely identify an entity in the set, also called a candidate key.
- There can be more than one candidate key.
- A primary key is a candidate key chosen to serve as the key for the entity set.

Minimal: no proper subset of the attributes can be a superkey

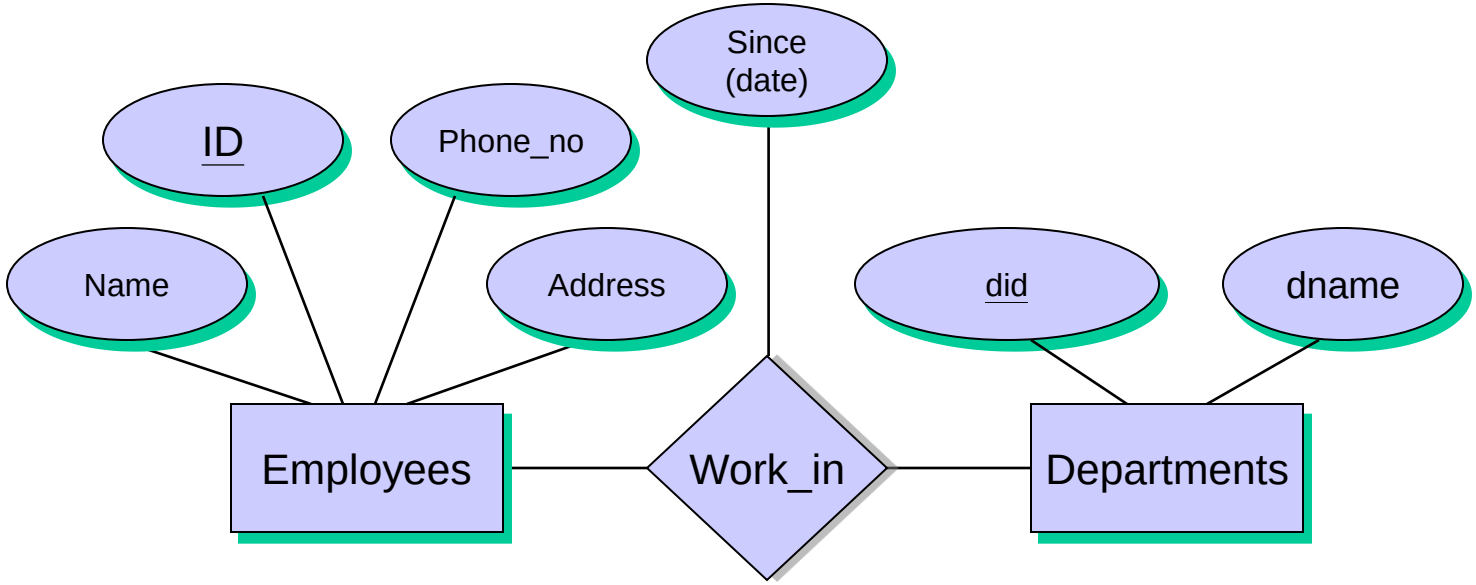
Keys Cont.

- There is also language relating to a relationship set.



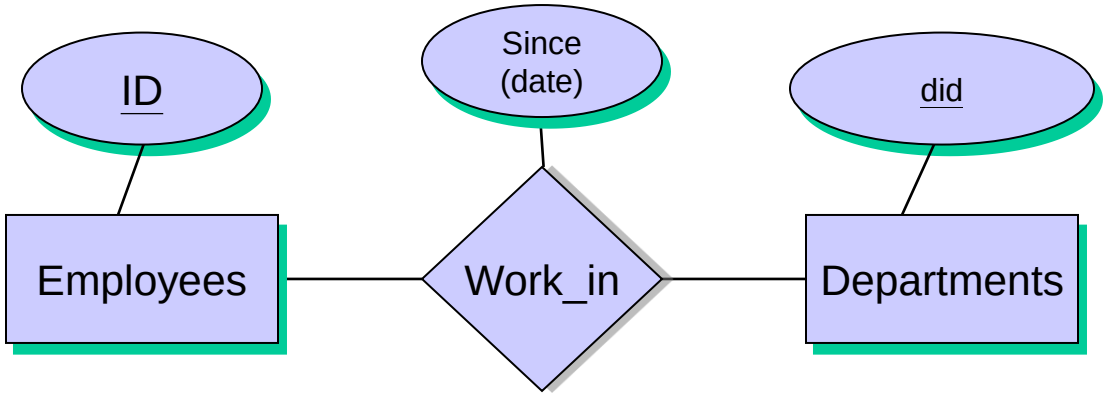
Keys in Relationships (0)

- A relationship must be uniquely identified. The participating entity instances contribute to the identification of the relationship instance .



Keys in Relationships (1)

- Here, each relationship must be uniquely identified by the combination of the employee *hkid* and the department *did*.
  - Meaning for each employee-department pair, we cannot have more than one associated “since” value.



Keys in Relationships (2)

- The concept of keys is also used to identify a relationship, as in entities.
- **Many-to-many**
  - For a relationship among *E1, …, Ek*, with no mapping constraint, the primary key is normally the union of the primary keys for *E1, …, Ek*.
  - *E.g., {ID, loan_number}* is the key for borrower.

- (For the time being, let's consider value of k = 2)

Keys in Relationships (3)

- The concept of keys is also used to identify a relationship, as in entities.
- **1-to-many** or **many-to-1**
  - An entity set *E* has a key constraint in a relationship set *R*, such that each entity in an instance of E appears in at most one relationship in the instance of *R*.
  - Hence an entity in *E* can uniquely identify a relationship in *R*. Thus, the key of *E* can be used as the key in *R*.
    - e.g. child-mother: is a many-to-one relationship, where entity Child has a key constraint. Child can be the primary key in the relationship set.

Keys in Relationships (4)

- The concept of keys is also used to identify a relationship, as in entities.
- **One-to-one**
  - For a one-to-one relationship between two entity sets *E* and *F*, key(*E*) and key(*F*) are both keys for the relationship set.
  - e.g. “manages” relation both {ID} and {did} are the keys of the relation.

Participation?

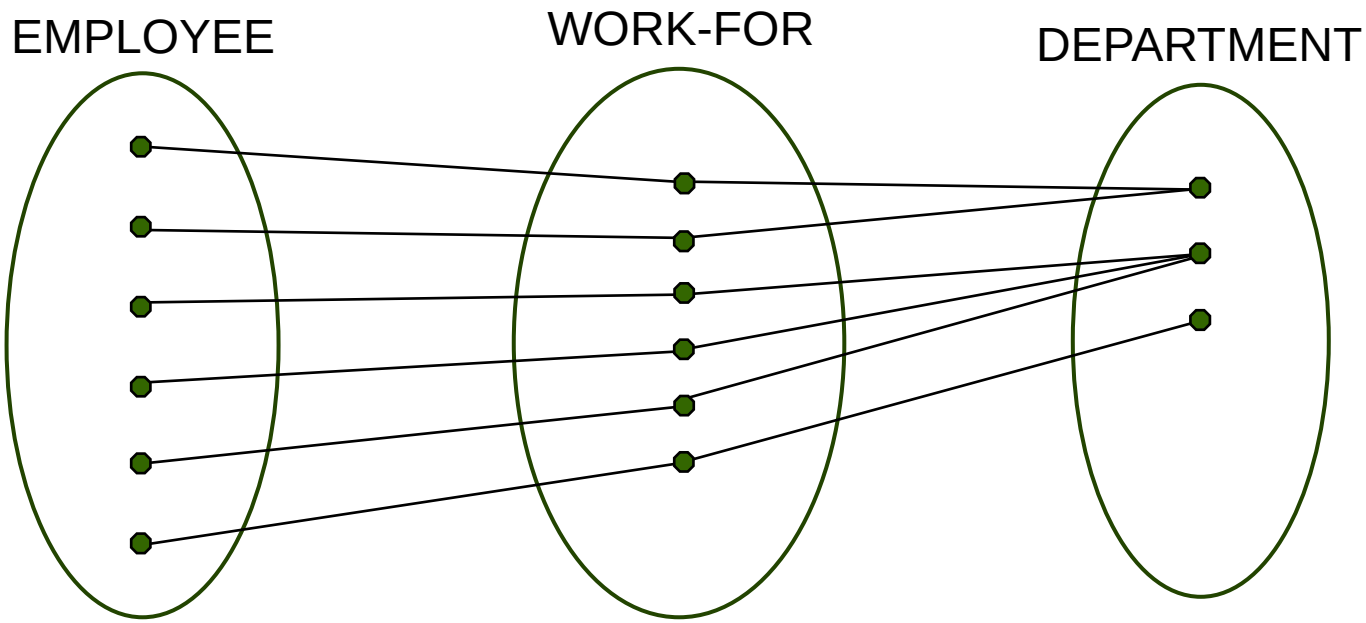
- The **key constraint** on *Manages* tells us that a department has at most one manager.
- A natural question to ask is whether every department has a manager.
- Suppose every department is required to have a manager. Such a constraint is an example of participation constraint.

Participation?

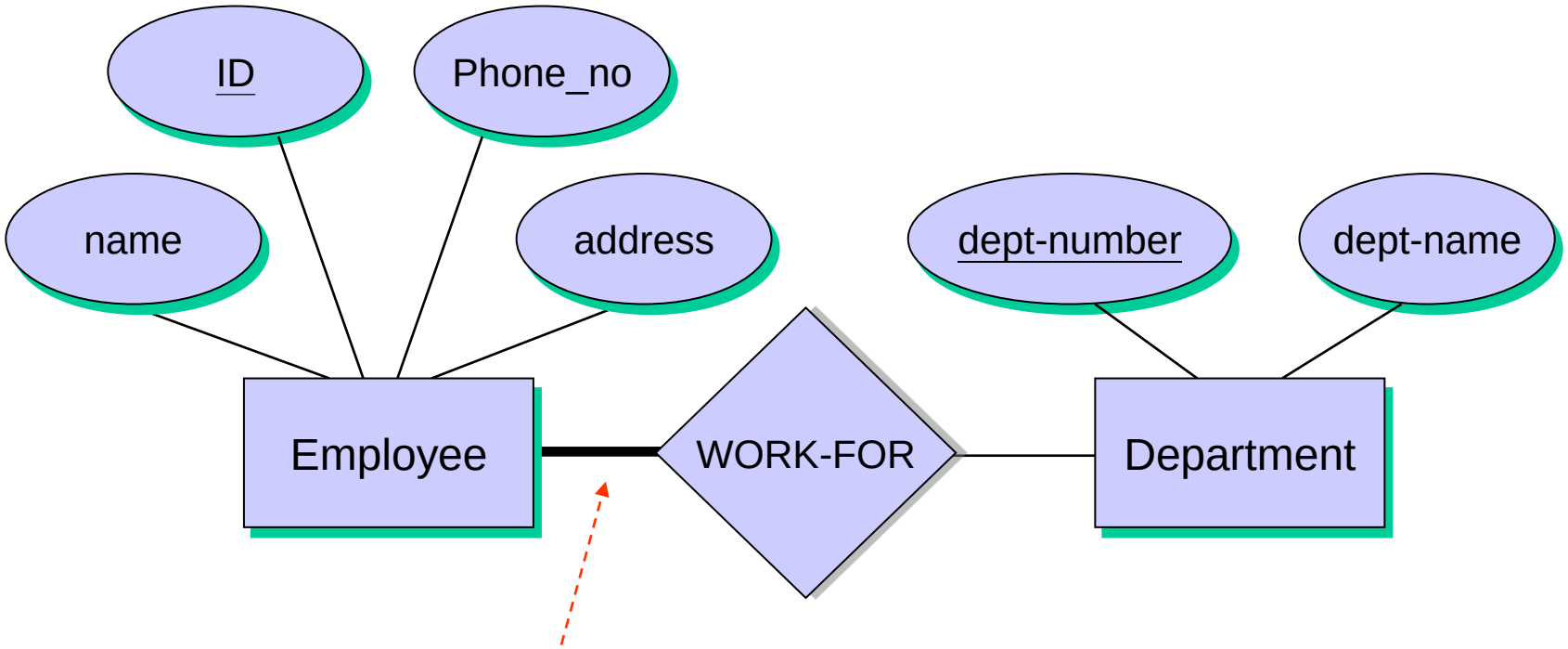
- In short, a participation constraint imposes some requirements on the number of times an entity participates in a relationship.
- We can classify participation in relationships as follows:
  - **Total** - each entity in the entity set must participate in at least one relationship.
  - **Partial** - an entity in the entity set may not participate in a relationship.

Participation?

- E.g. Every employee must work for some department.
  - What would you say about the participation of *EMPLOYEE* in *WORKS-FOR* relationship?



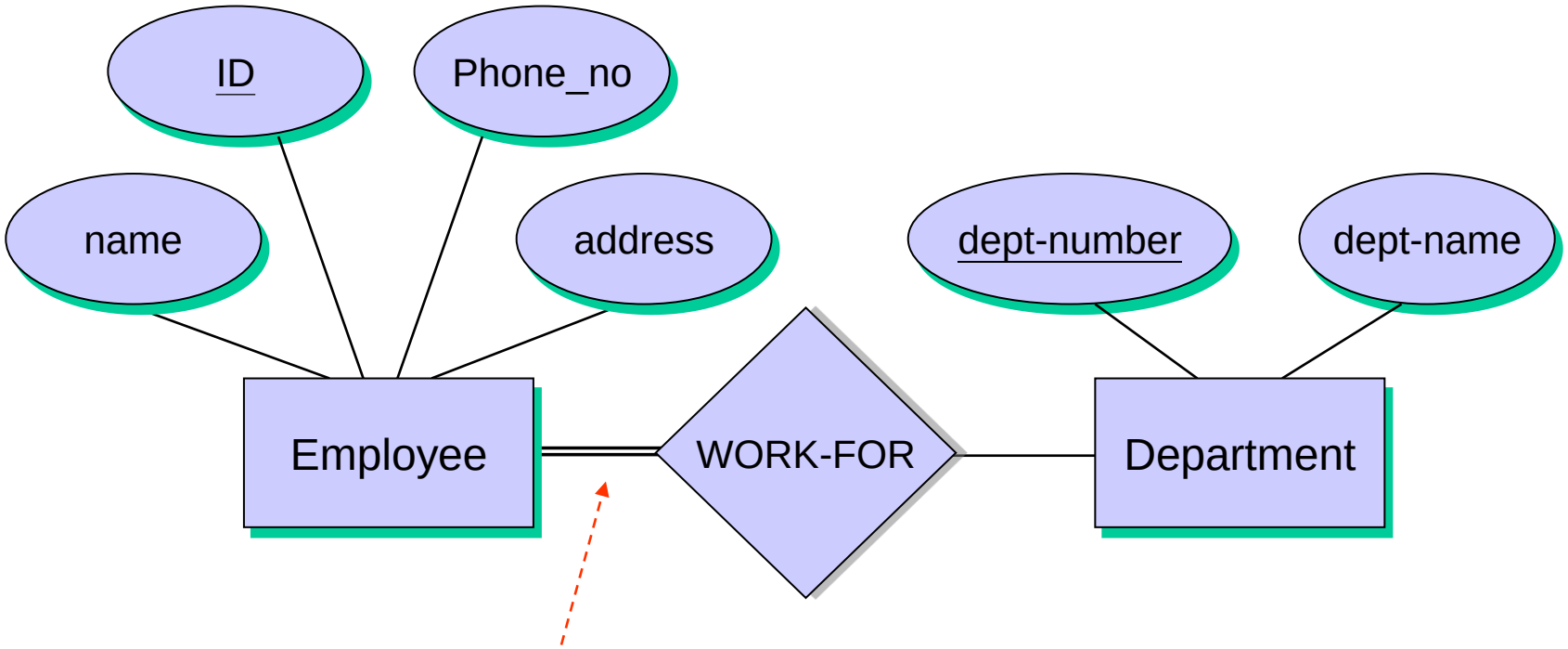
Lecture Notation



If the participation of an entity set in a relationship set is total, the two are connected by a **THICK** line; independently, the presence of an arrow indicates a key constraint.

In this case, the **THICK** line means every employee has to participate in the work-for relationship.
i.e., every employee has to work for some departments (at least one department).

(I'll allow this alt. notation)



If the participation of an entity set in a relationship set is total, the two are connected by a **DOUBLE** line; independently, the presence of an arrow indicates a key constraint.

In this case, the **DOUBLE** line means every employee has to participate in the work-for relationship.
i.e., every employee has to work for some departments (at least one department).

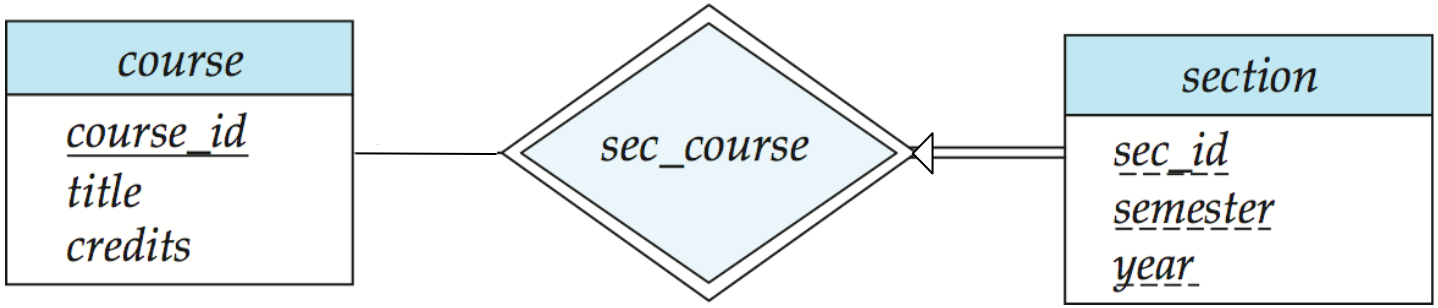
- 10 minute break

Weak Entity

- Weak entity, entities with the property that they can't have a "natural primary key" of their own.
- Scenario: Course and Section

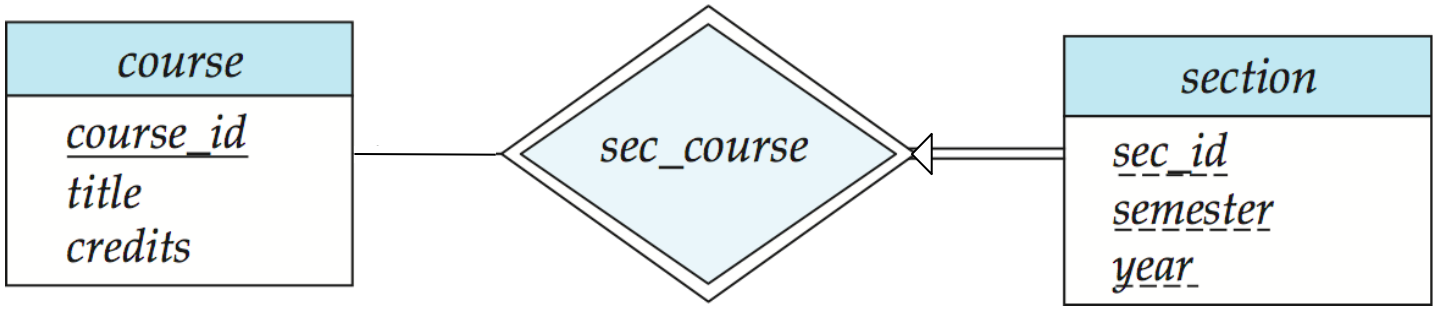
Weak Entity Sets?

- All entity sets have a primary key. They exist because they are independent.
- There does not exist a primary key in “section”.
  - There may be two, (1, Spring, 2010) and (1, Spring, 2010).
  - They are two distinct entities from different courses, but we cannot manage them.



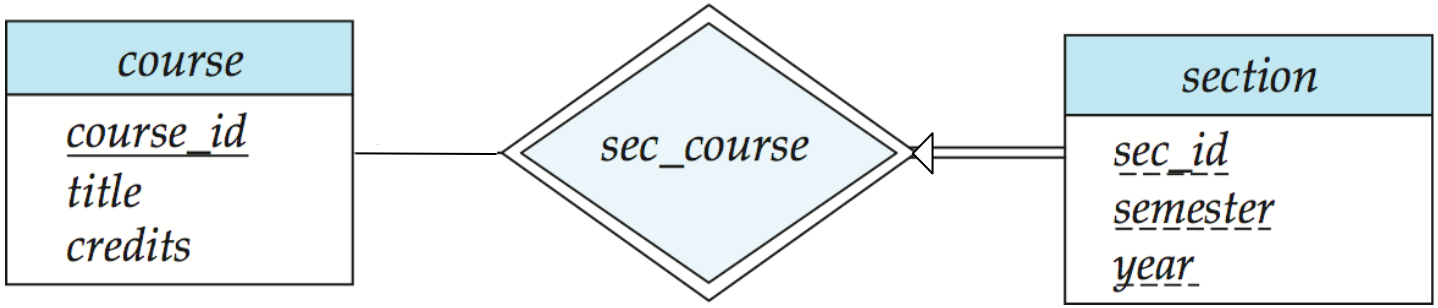
Weak Entity Sets?

- Consequently, the section is **not an independent entity set**, because we can find no effective primary key.
- But the section should be an entity (we need it to be)
  - In this sense, it should be something weaker.



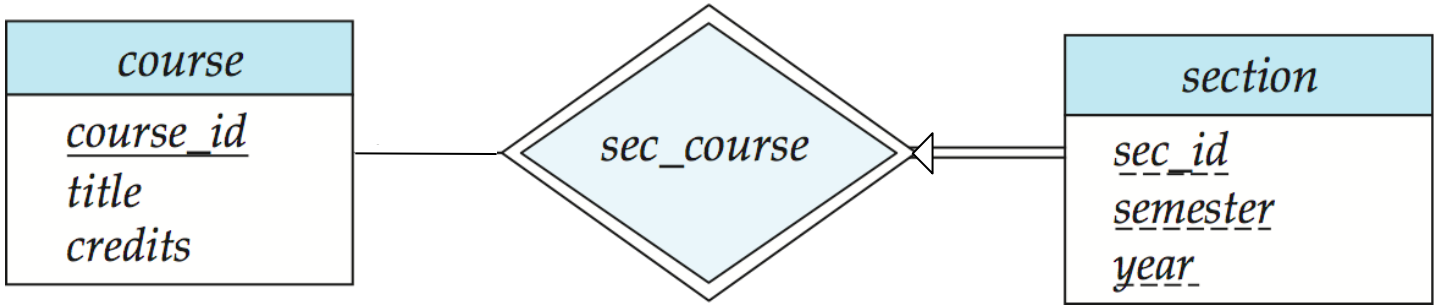
Make Weaker Stronger?

- What if… we make *sec_id* unique in the *section* entity?
  - Add an **artificial** primary key *sec_id* and section becomes an entity set or precisely a strong entity set.
  - Could this work?



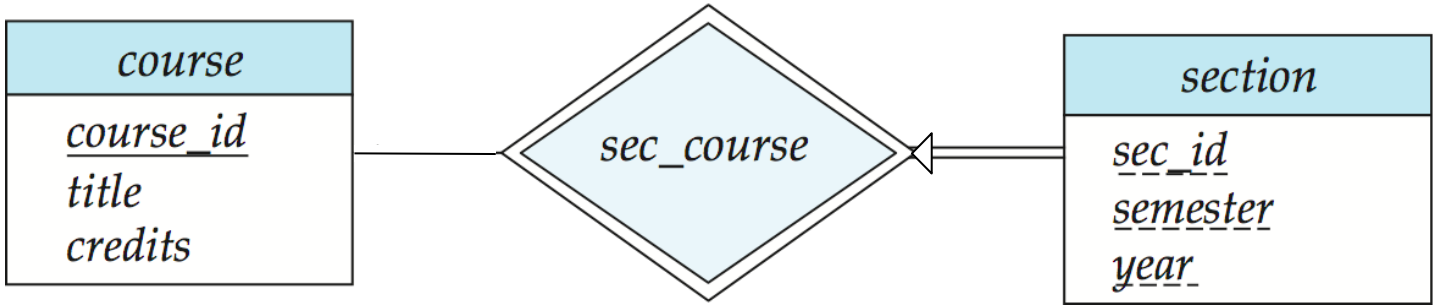
Make Weaker Stronger?

- Conceptually, a section is still dependent on a course for its existence.
  - How about we add a course_id into the section entity set?
    - Such that *Section* has primary key *(course_id, sec_id, semester, year)*.
    - *Section* becomes an entity set or precisely a strong entity set!



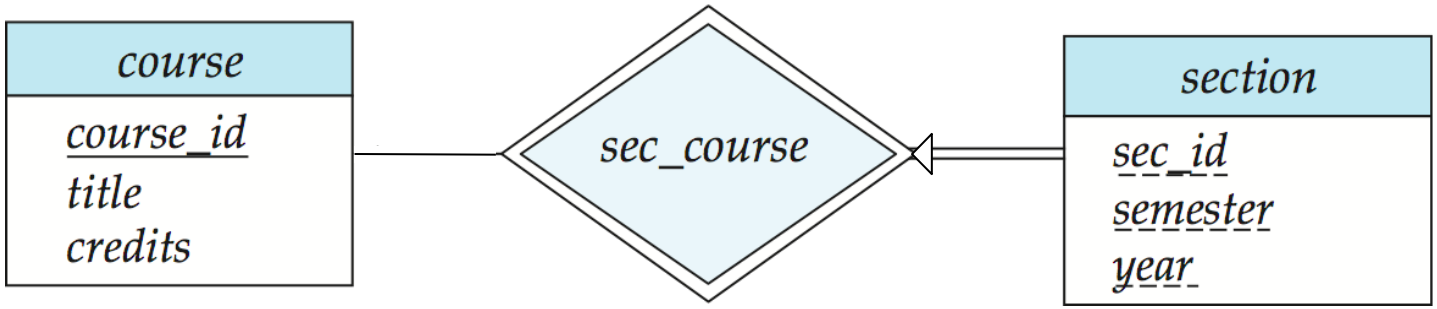
Make Weaker Stronger?

- There are a few **big** problems:
- (1) Semantically it doesn't fit, or is awkward
  - *course_id* is not a part of section, esp. since we treat section as an entity set.
  - *course_id* is to identify the course entity set not the section.



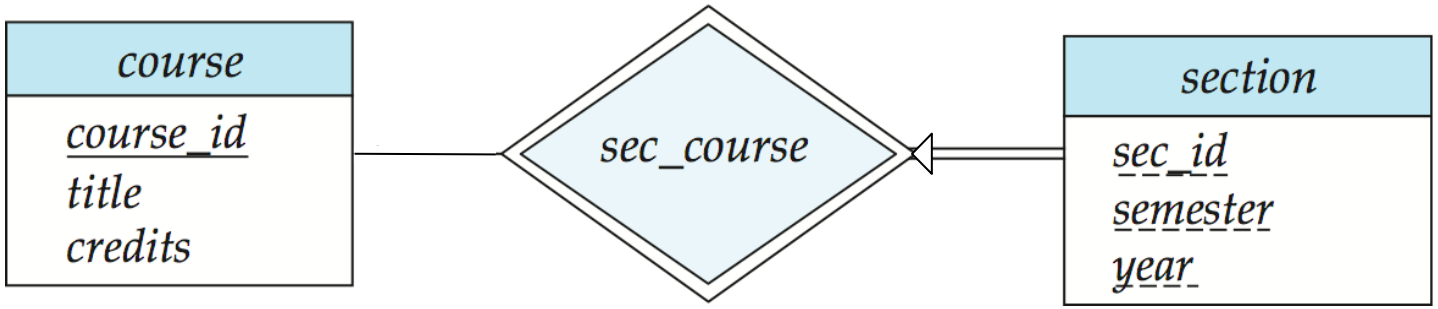
Make Weaker Stronger?

- There are a few **big** problems:
- (2) What about the preexisting relation?
  - (1) if we add *course_id* into the section entity, the *sec_course* relationship becomes redundant. (2) But, without *sec_course*, the relationship between course and section becomes implicit, which is not what we want.



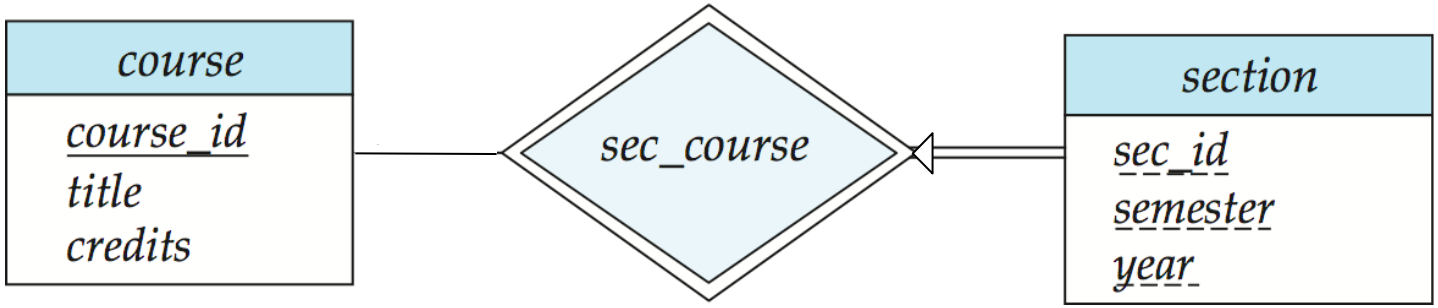
But we need a primary key!

- How about we "borrow" the *course_id* from the *course* entity set instead?
  - Meaning the **primary key** of the section is formed by (1) primary key of the course on which the section is existence dependent and (2) the weak entity's discriminator *sec_id*.
  - The discriminator (or partial key) of *section* is the set of attributes: *sec_id, semester, and year*.



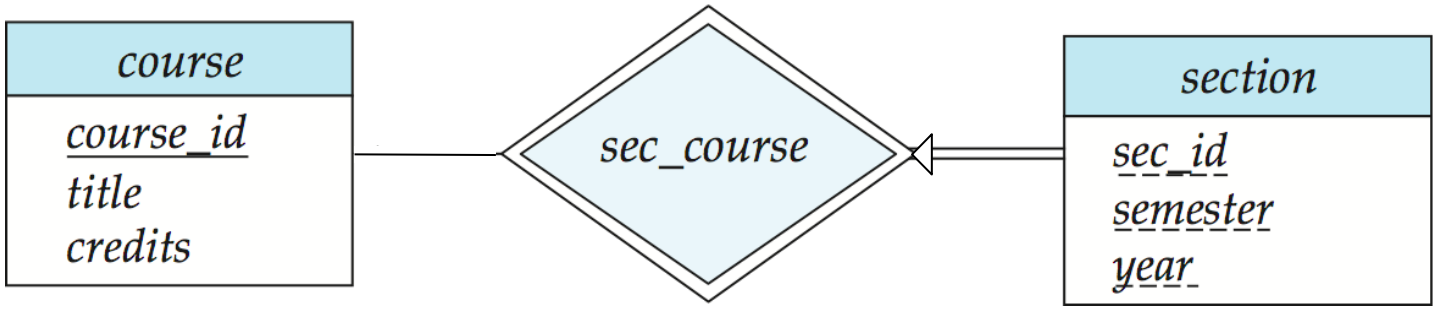
Weak Entity Sets (1)

- The **existence** of a weak entity now depends on the existence of an identifying entity set
  - It must relate to the identifying entity set via a total; one-to-many relationship set from the identifying to the weak entity set
  - Identifying relationship is depicted using a thick/double diamond



Weak Entity Sets (2)

- Primary key *section*: *(course_id, sec_id, semester, year)*
- Notation:
  - We **underline** the discriminator of a weak entity set with a dashed line.
  - We put the identifying relationship of a weak entity in a thick/double diamond.



Summary (1)

- An entity set that does not have a primary key is referred to as a **weak entity set**.
  - Weak entities have a partial discriminator key
- Terminology
  - ***Partial discriminator key:*** a key that is partially unique

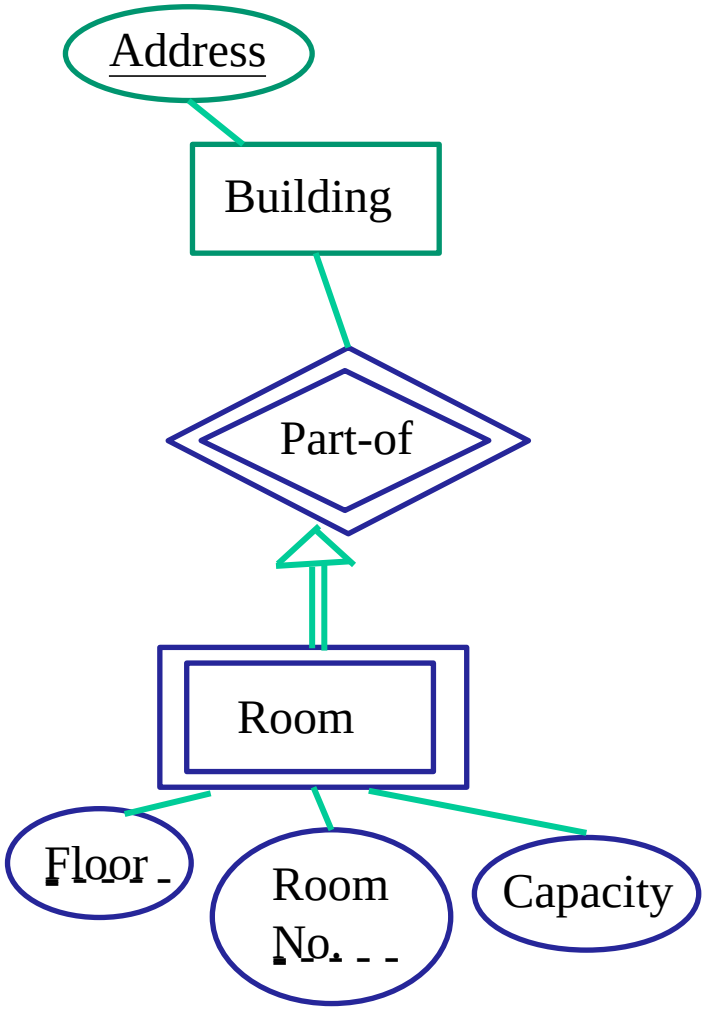
- ***Identifying relationship***
  - The relationship type between a weak entity type to the *owner* of the weak entity type

Summary

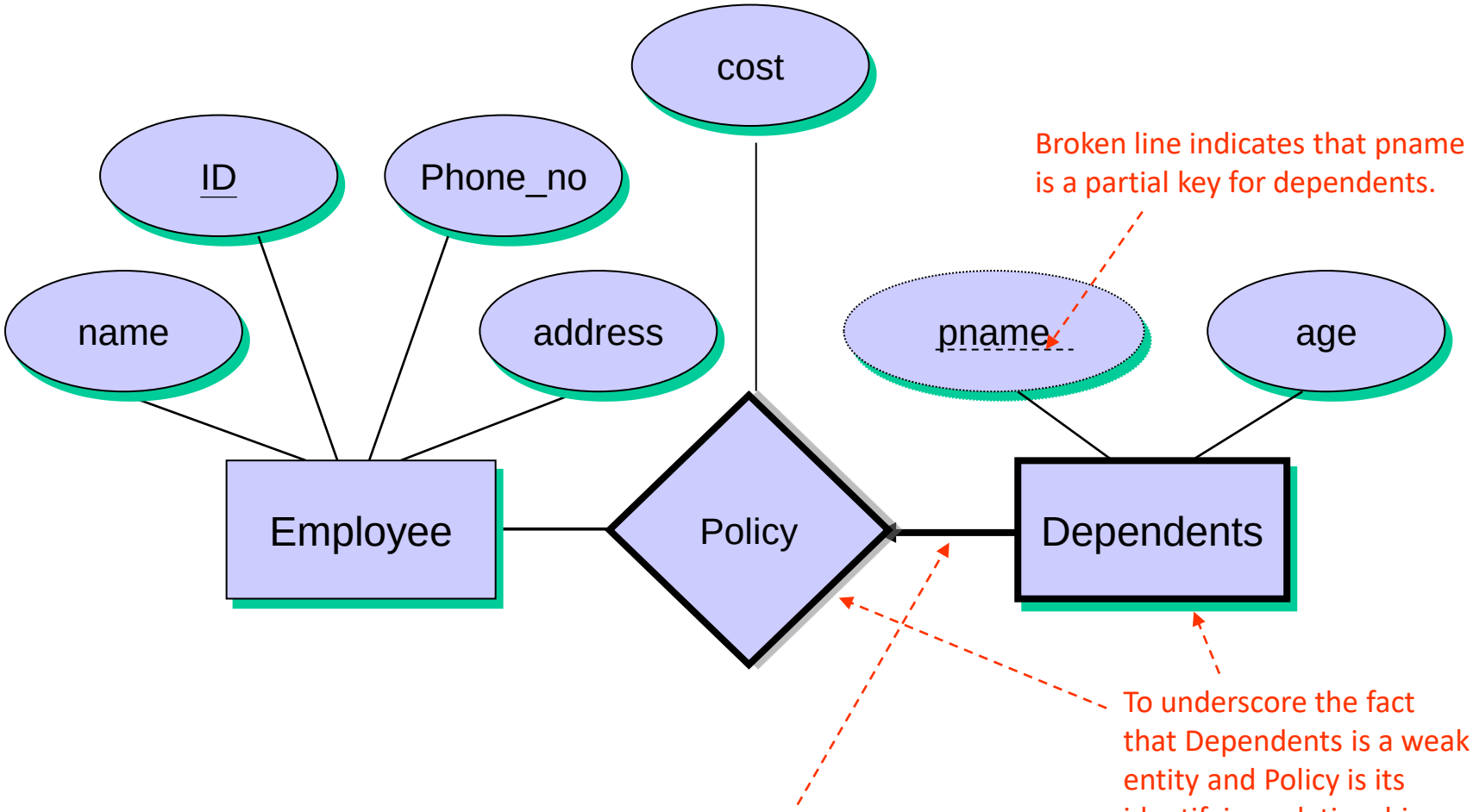
- ***Strong entity***
  - An entity which has it's primary key
  - Each entity can be distinguished from other entities in the same set.
- ***Weak entity***
  - An entity which does not have it's own primary key
  - May not be able to distinguish themselves from others without associations with entities in other entity sets.

Example (1)

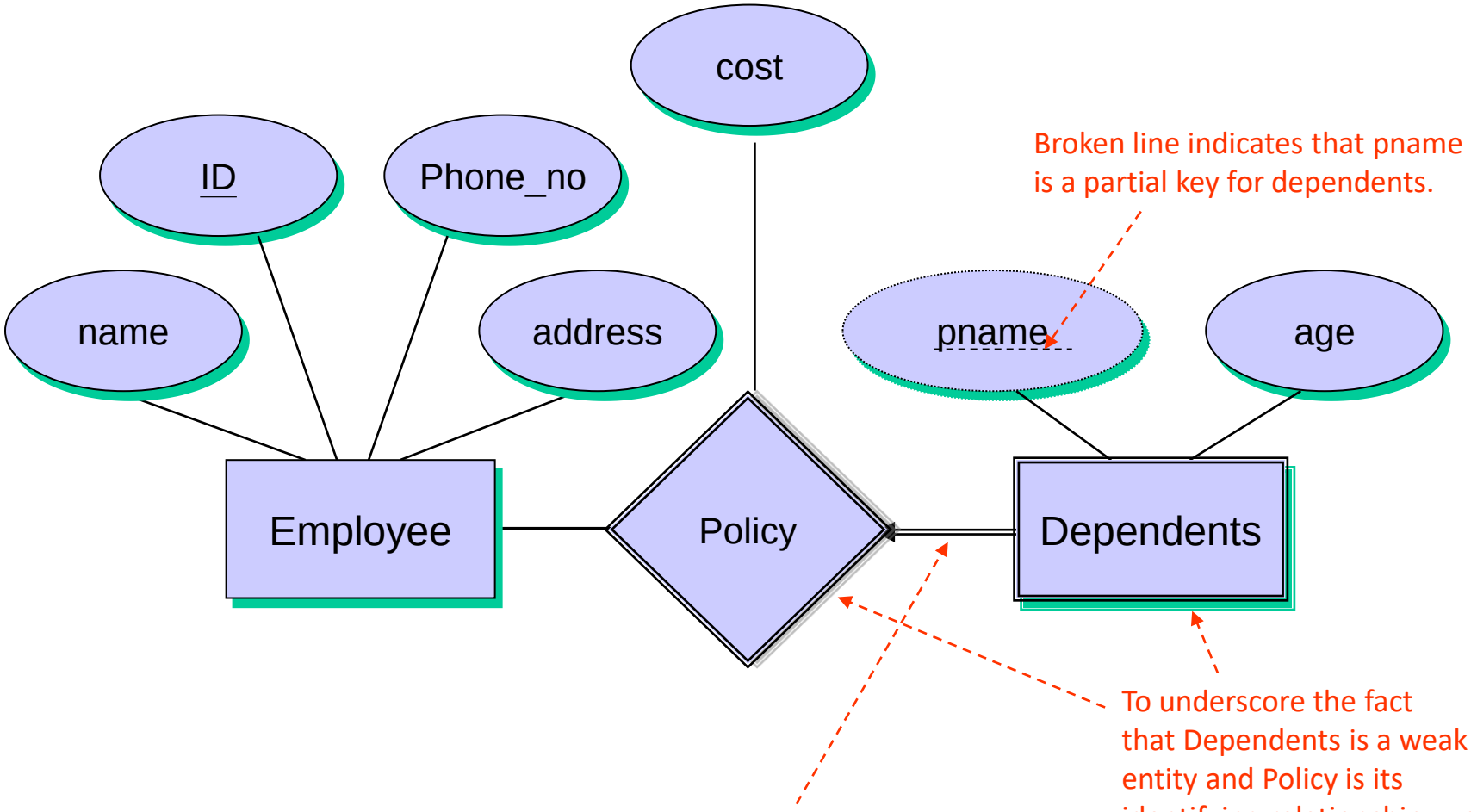
1. A *Building* entity may be related to several *Rooms*, identified by their *floor* and *room_number*.
2. Room is called a weak entity, {*floor, room_number*} is a partial key for it.
3. The **identifying relationship** between *Building* and *Room* is *part-of*.
4. *Building* is said to own *Room*.



Lecture notation

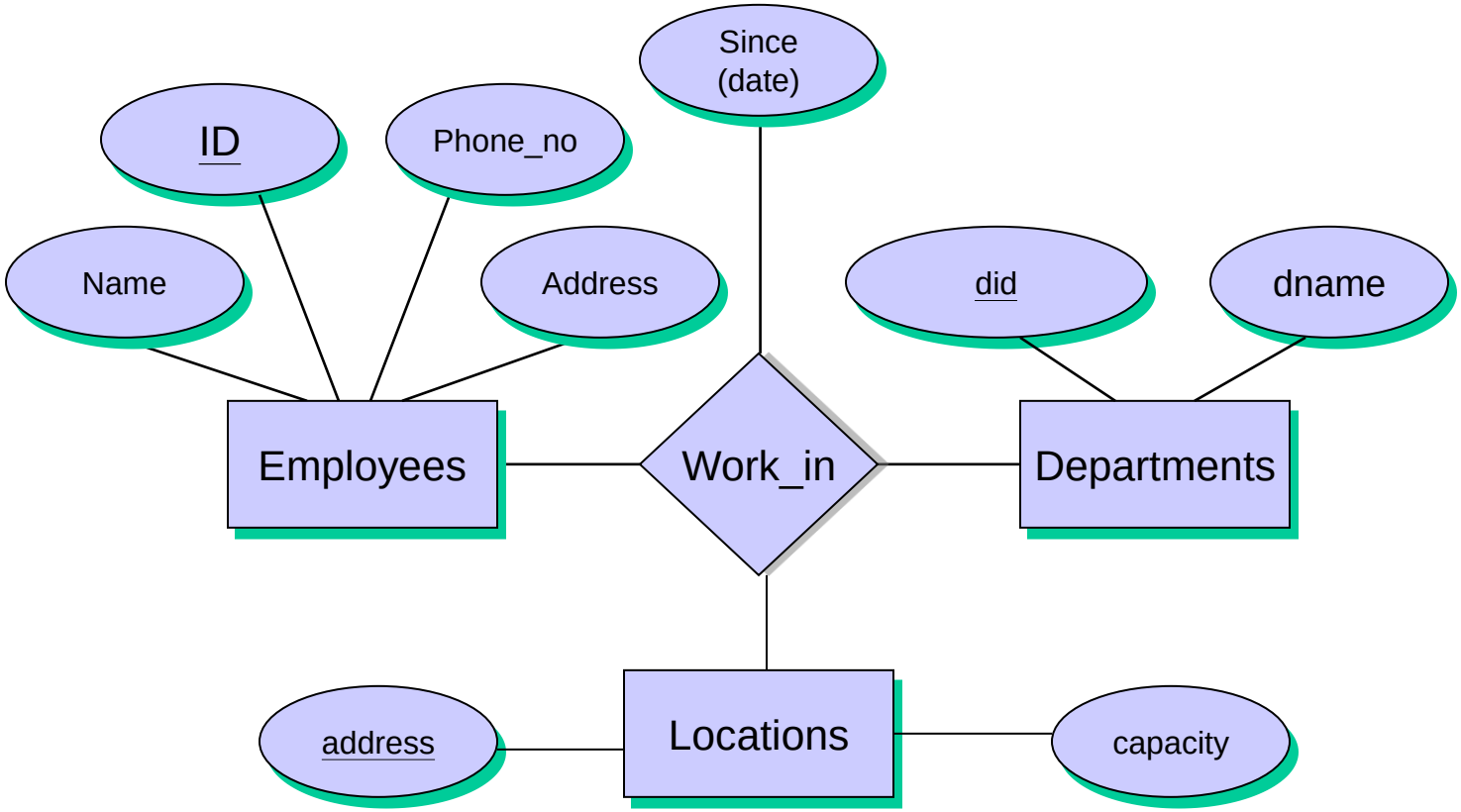


I'll allow this alt. notation



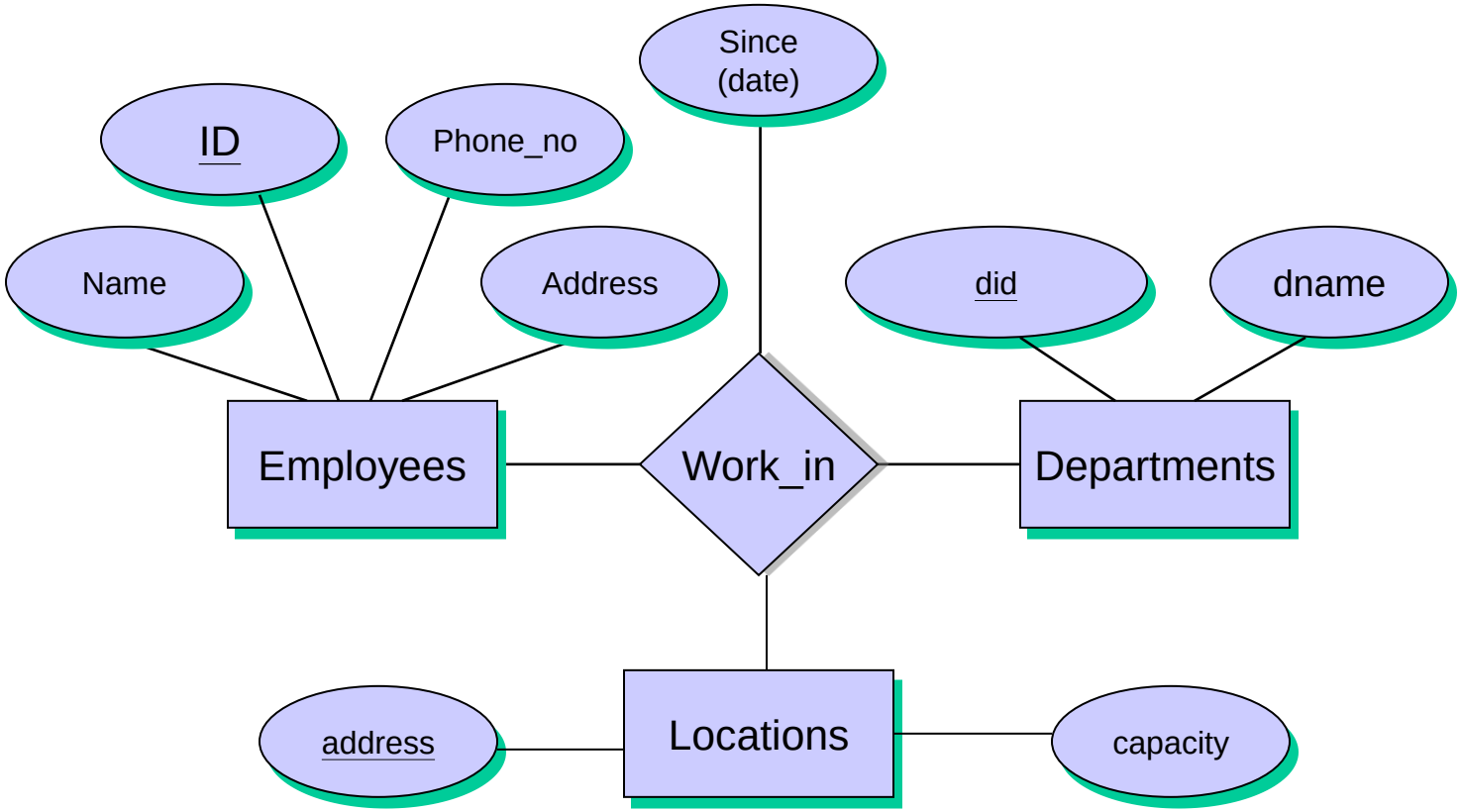
Exercise

- Notice anything about this ER diagram?



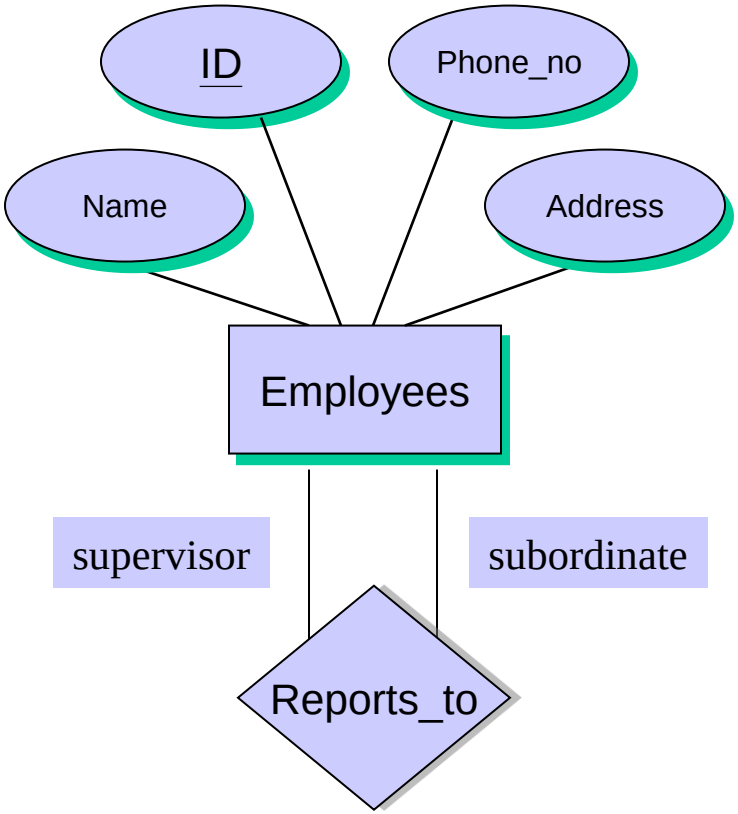
Non-Binary Relationships

- This relationship is ternary (degree=3).



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.

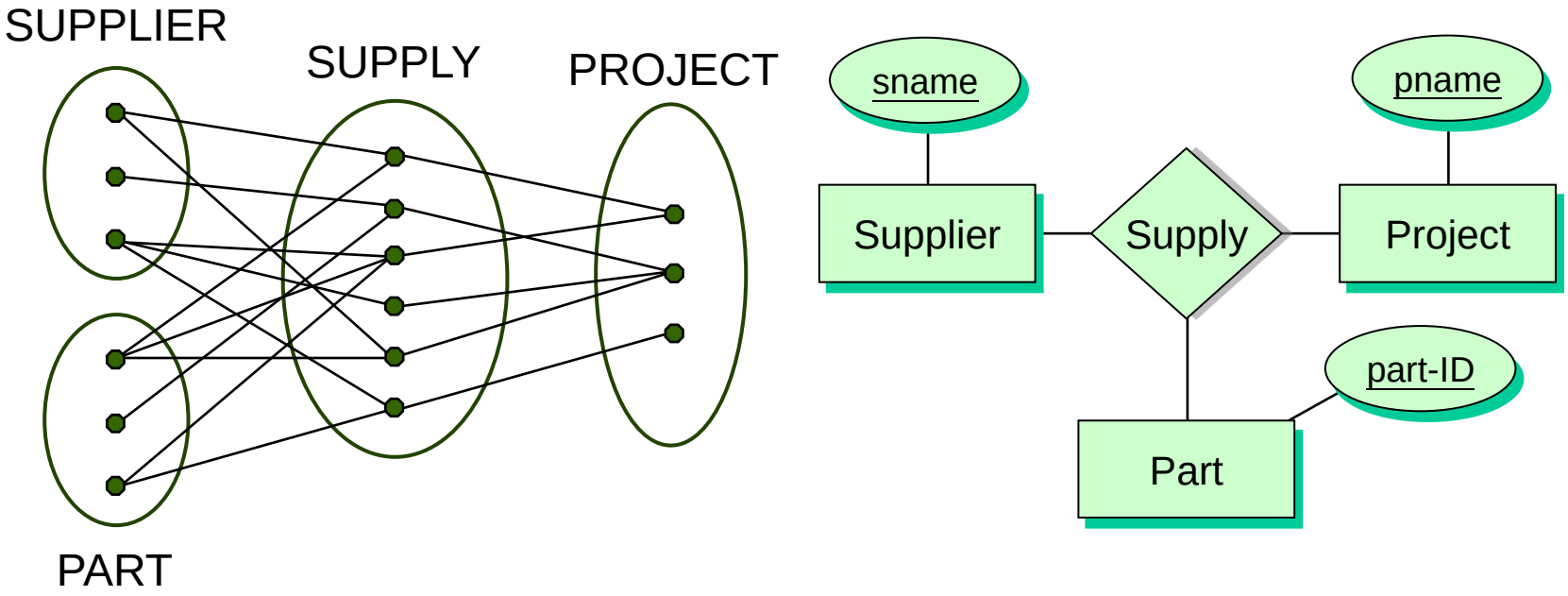


Recursive Relationships

- Suppose now each department has offices in several locations and we want to record the locations at which each employee works.
- Since employees report to other employees
  - Every relationship in Reports_To is of the form *(emp1,emp2),* where both *emp1* and *emp2* are entities in employees.
  - However, they play different roles.
    - emp1 reports to emp2, which is reflected in the role indicators *supervisor* and *subordinate* in the previous diagram.

Non-binary relationship set

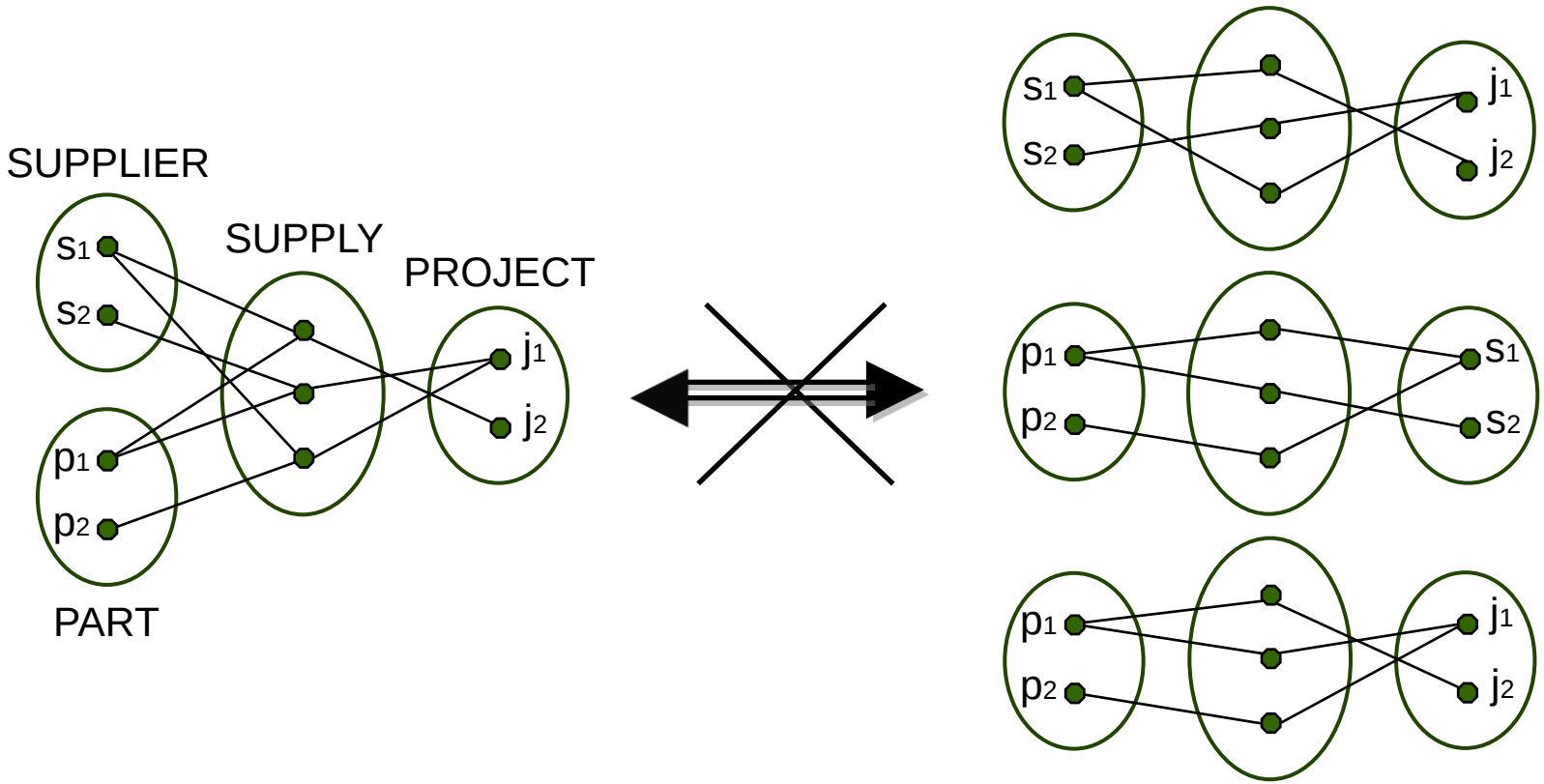
- Suppose $n \geq 2$, for *E1, E2, …, En*



Discussion

- In general, a non-binary relationship set cannot be replaced by a number of binary relationship sets.
- Consider supplier s, project j, and part p,
  - Existence of (s,p), (j,p) and (s,j), where s in SUPPLIER, p in PART, j in PROJECT, does not imply existence of (s,j,p) necessarily.
  - This is known as the connection trap

Discussion


SUPPLIER
SUPPLY
PROJECT
s1
s2
p1
p2
j1
j2
PART
s1
s2
j1
j2
p1
p2
s1
s2
p1
p2
j1
j2

Conceptual Design

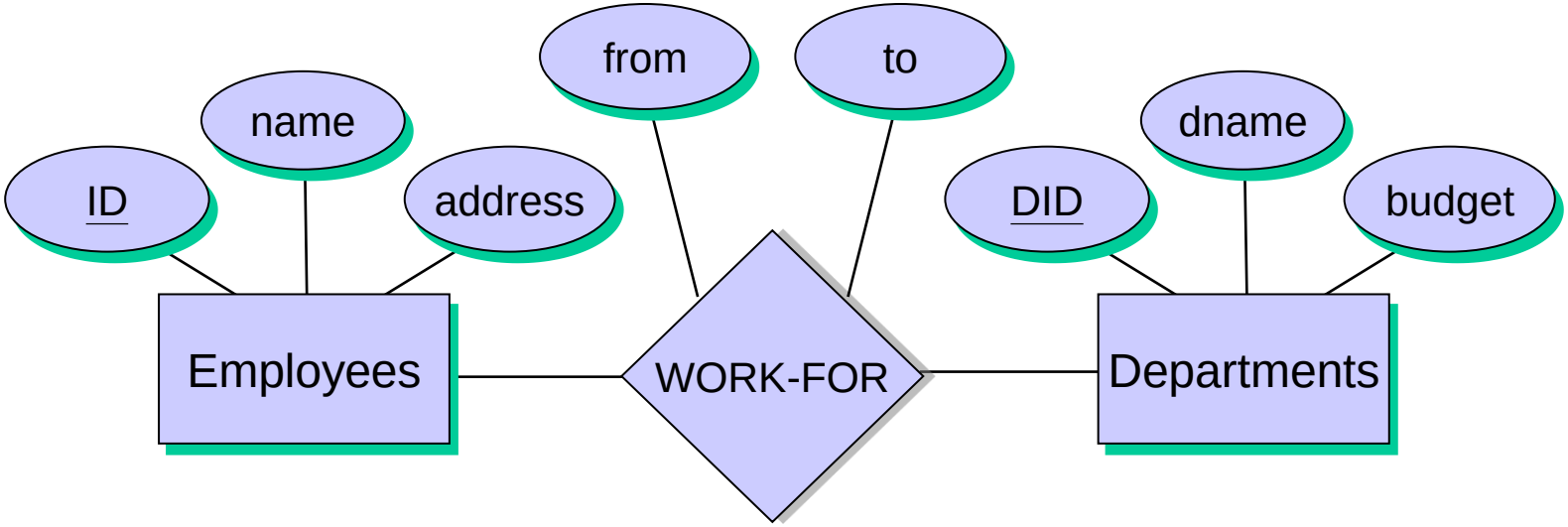
- Developing an ER model presents several questions.
  - Should a concept be modeled as an entity or an attribute?
  - Should a concept be modeled as an entity or a relationship?
  - What are the relationship sets and their participating entity sets? Should we use binary or ternary relationships?
  - Should we use aggregation?
  - Will ER models be in the exam?

Conceptual Design

- Entity versus Attribute
  - Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?
  - Considerations
    1. If we have several addresses per employee, *address* must be an entity (*attributes cannot be* ***set-valued***)
    2. If the structure (city, street, etc) is important, e.g. want to retrieve employees in a given city, address must be modeled as an entity (*attribute values are atomic*)

Conceptual Design

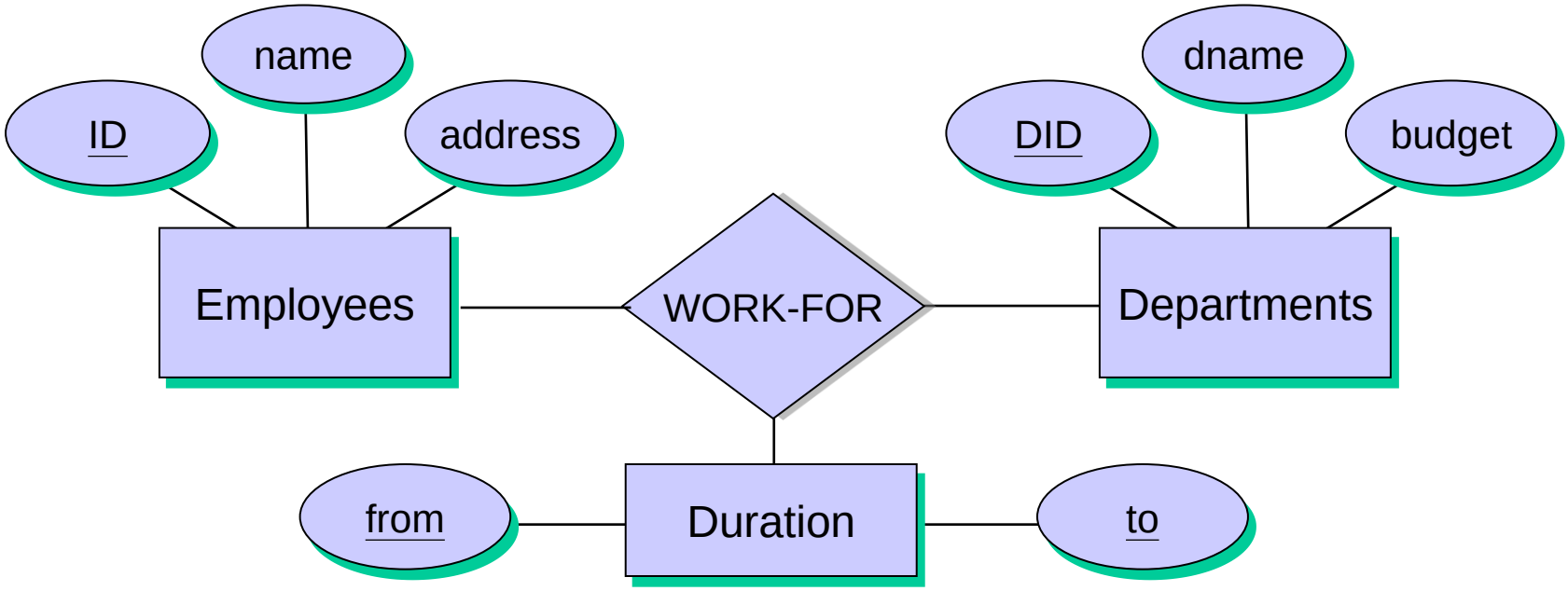
- Should duration of an employee working in a department be an attribute or an entity?



- 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.

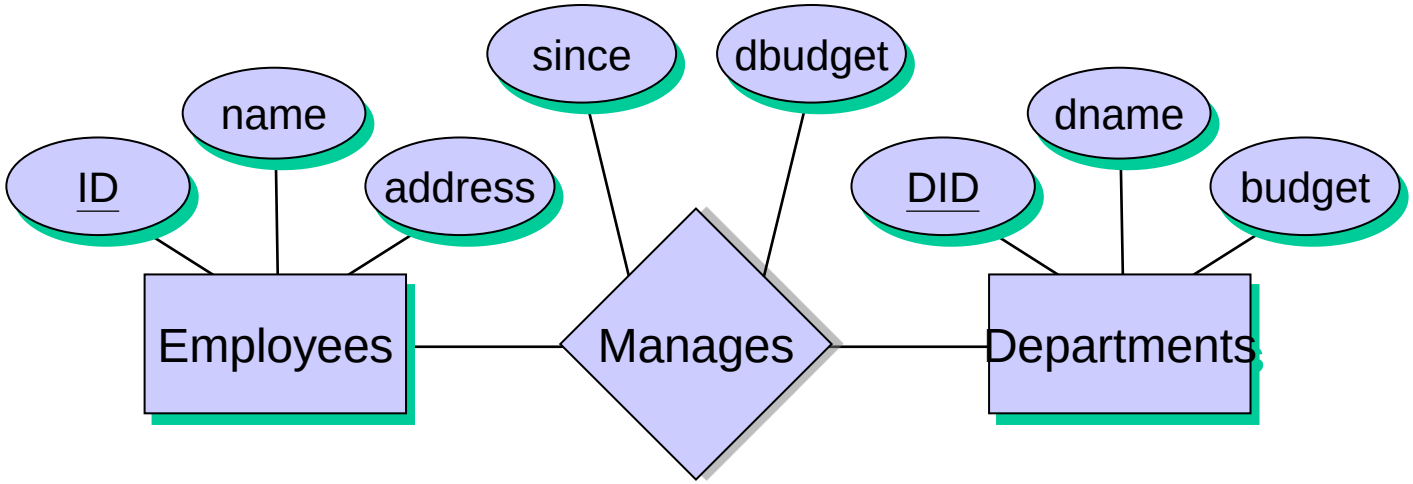
Conceptual Design

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



Conceptual Design

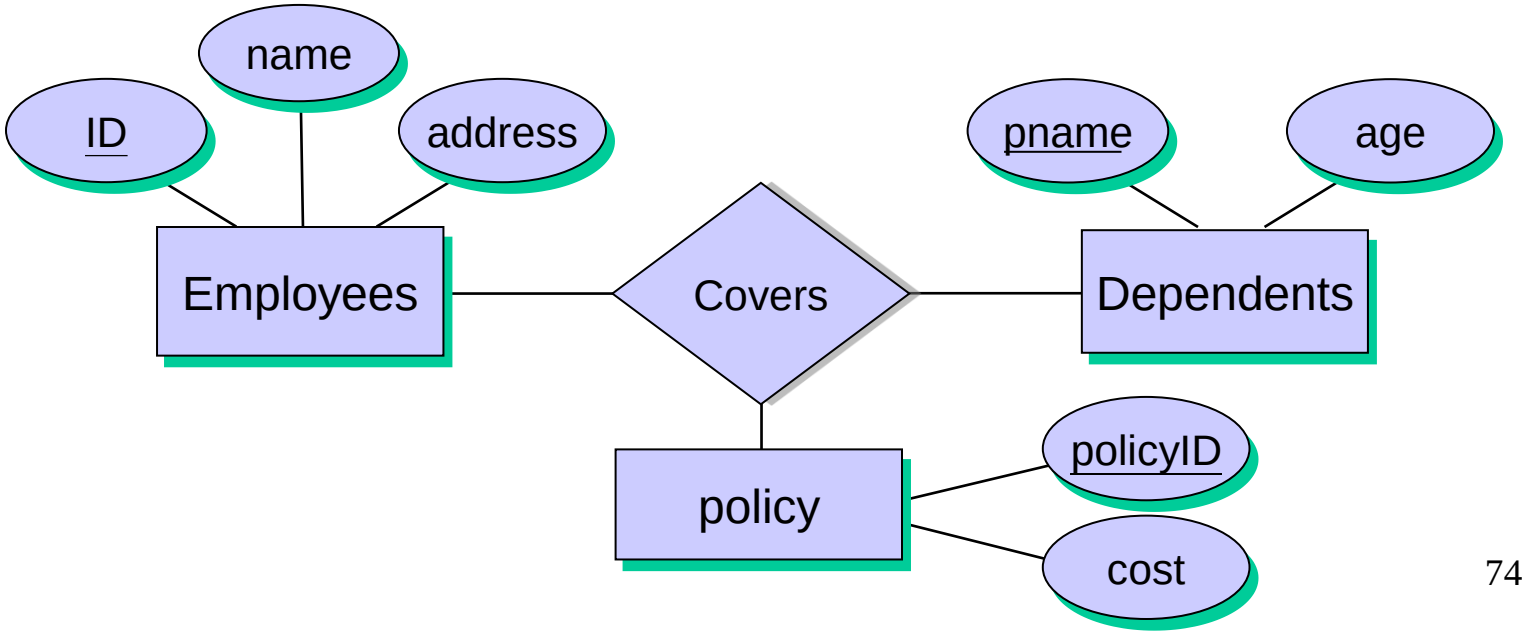
- Entity versus Relationship
  - Suppose each department manger is given a discretionary budget (dbudget).



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

Conceptual Design

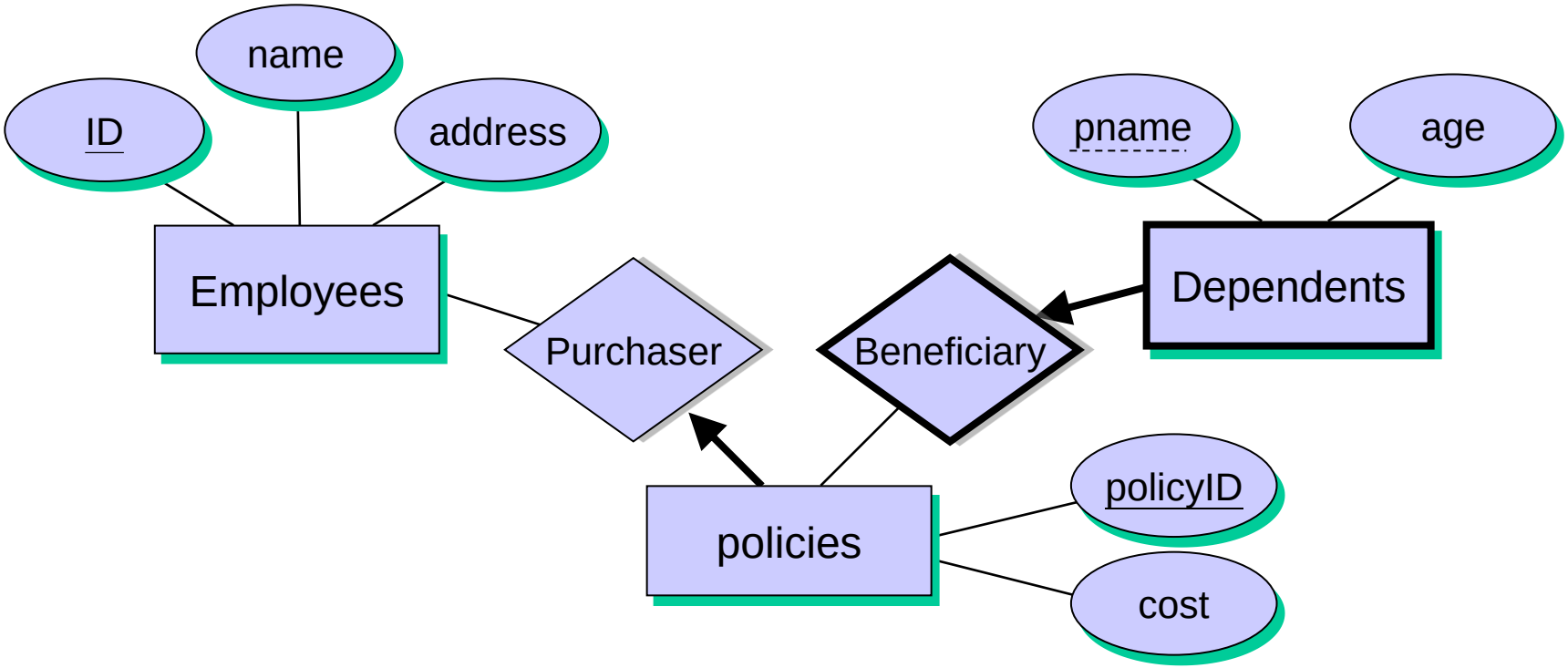
- Binary versus Ternary Relationships
  - The following ER diagram models the situation that an employee can own several policies, each policy can be owned by several employees, and each dependent can be covered by several policies.



- Suppose we have the following additional requirements:
  - A policy cannot be owned jointly by two or more employees. (Impose a key constraint on Policies with respect to Covers, but it introduces a side effect that each policy can cover only one dependent).
  - Every policy must be owned by some employee. (Impose a total participation constraint on Policies, it is acceptable if each policy covers at least one dependent).
  - Dependents is a weak entity set, and each dependent entity is uniquely identified by taking pname in conjunction with the policyid of a policy entity.

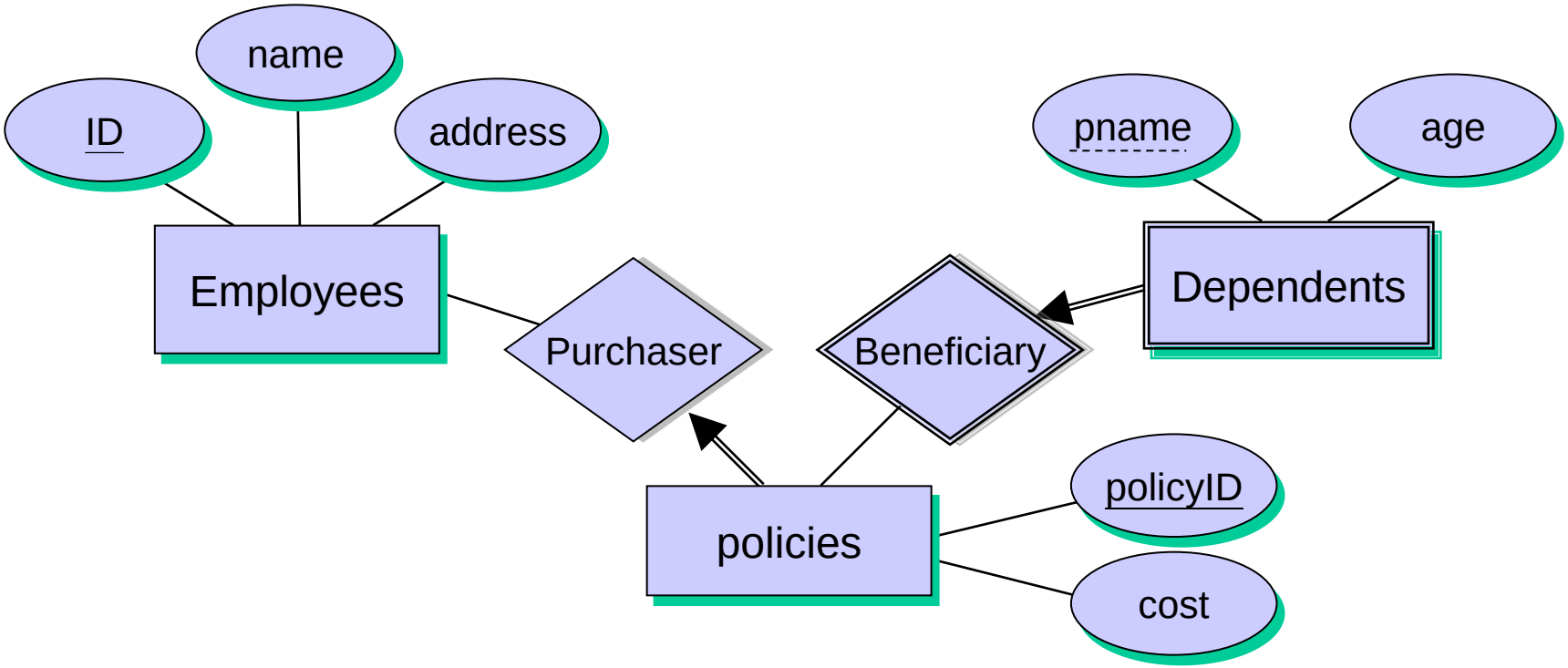
Lecture Notation

- Here is a solution



(I'll allow this alt. notation)

- Here is a solution

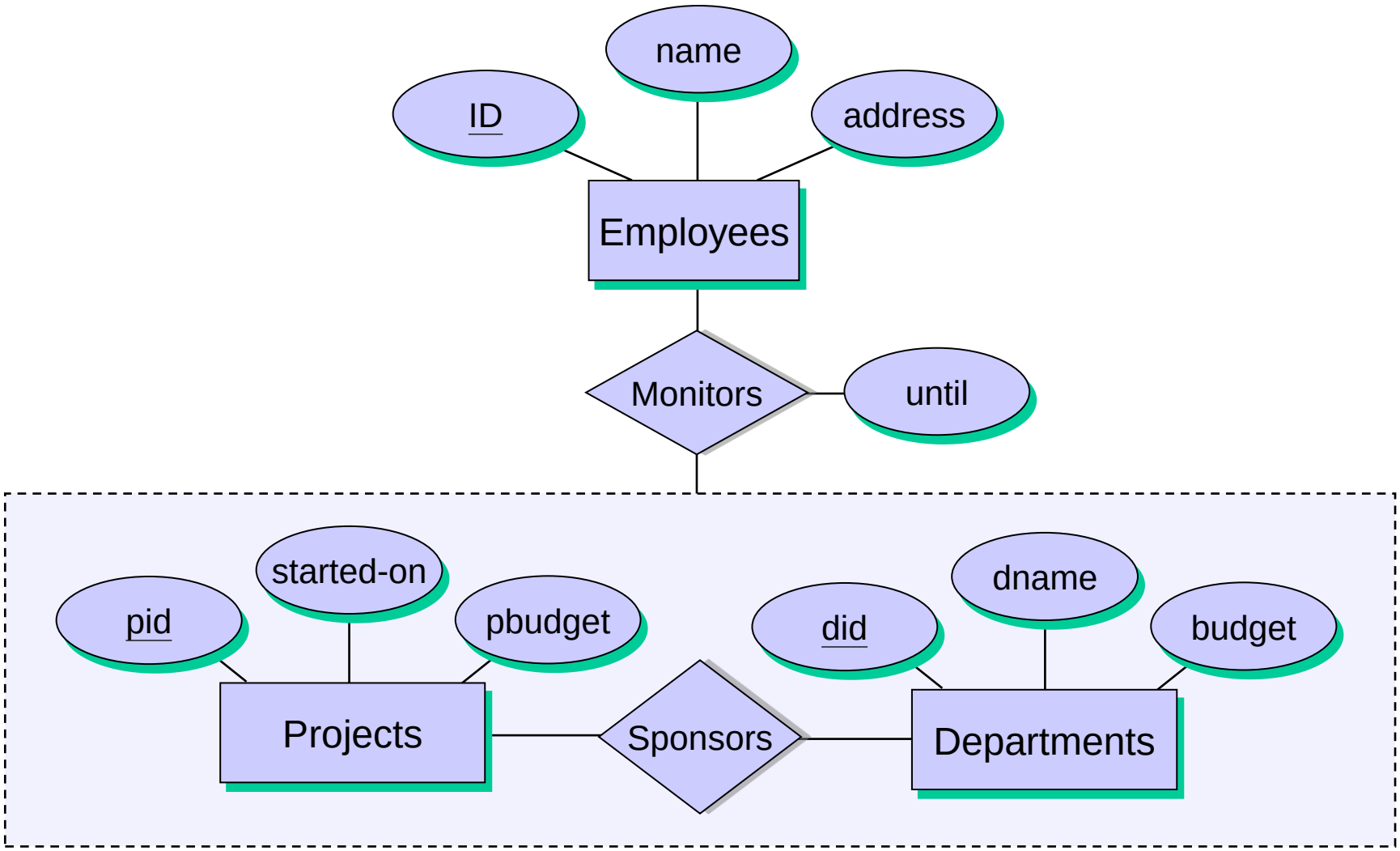


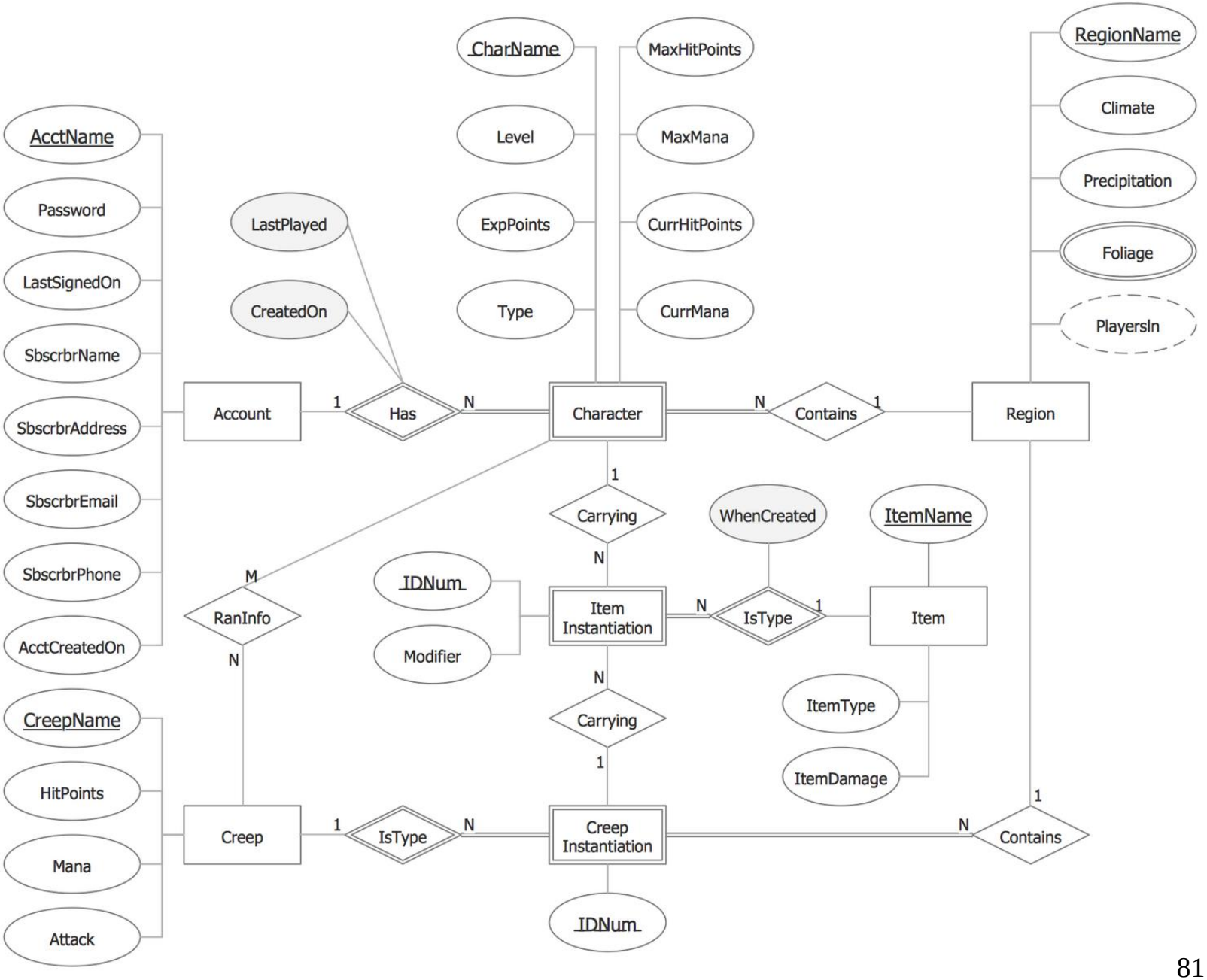
Aggregation

- Sometimes, we have to model a relationship between a collection of entities and *relationships*.
- Aggregation allows us to indicate that a relationship set (identified through a dashed box) participates in another relationship set.

Example

- Suppose we have an entity set called projects.
- Each projects 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 projects or departments entity) with an Employees entity.


name
ID
address
Employees
Monitors
until
started-on
pid
pbudget
dname
did
budget
Projects
Sponsors
Departments



https://i.pinimg.com/originals/f0/2e/61/f02e61606cf4f522087bb3940ed35a3b.png