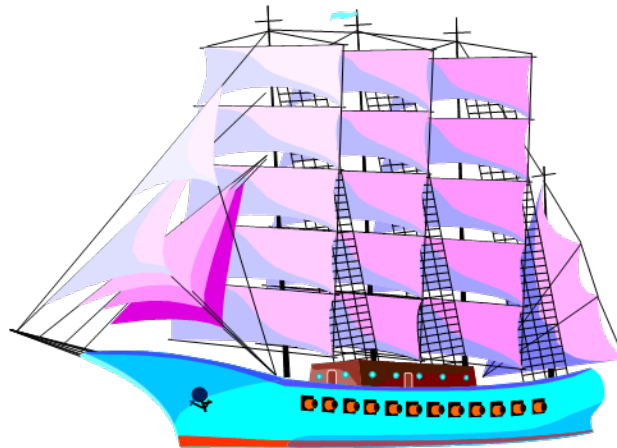


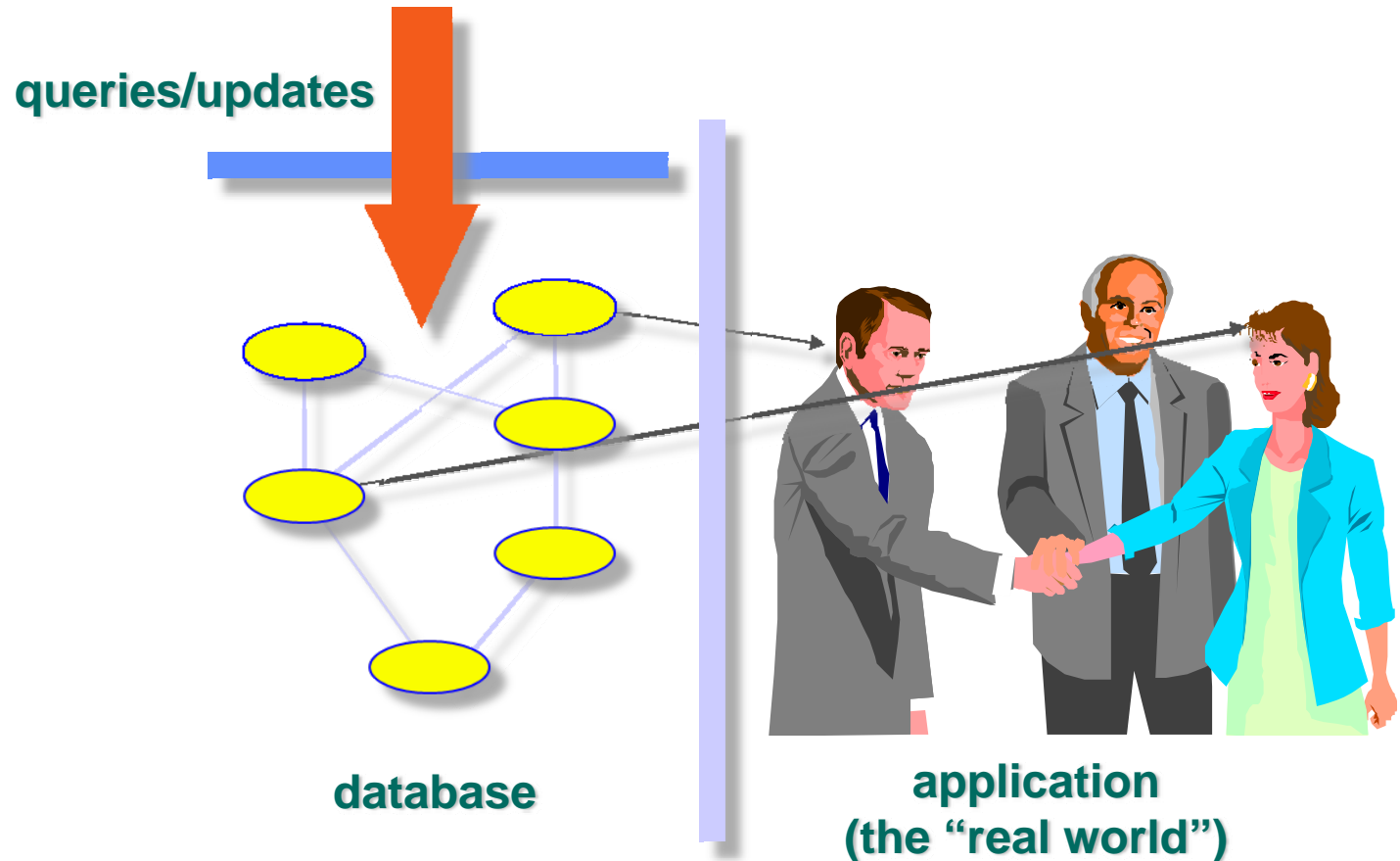
# CSCC43 Introduction to Databases

## *The Entity-Relationship Model*



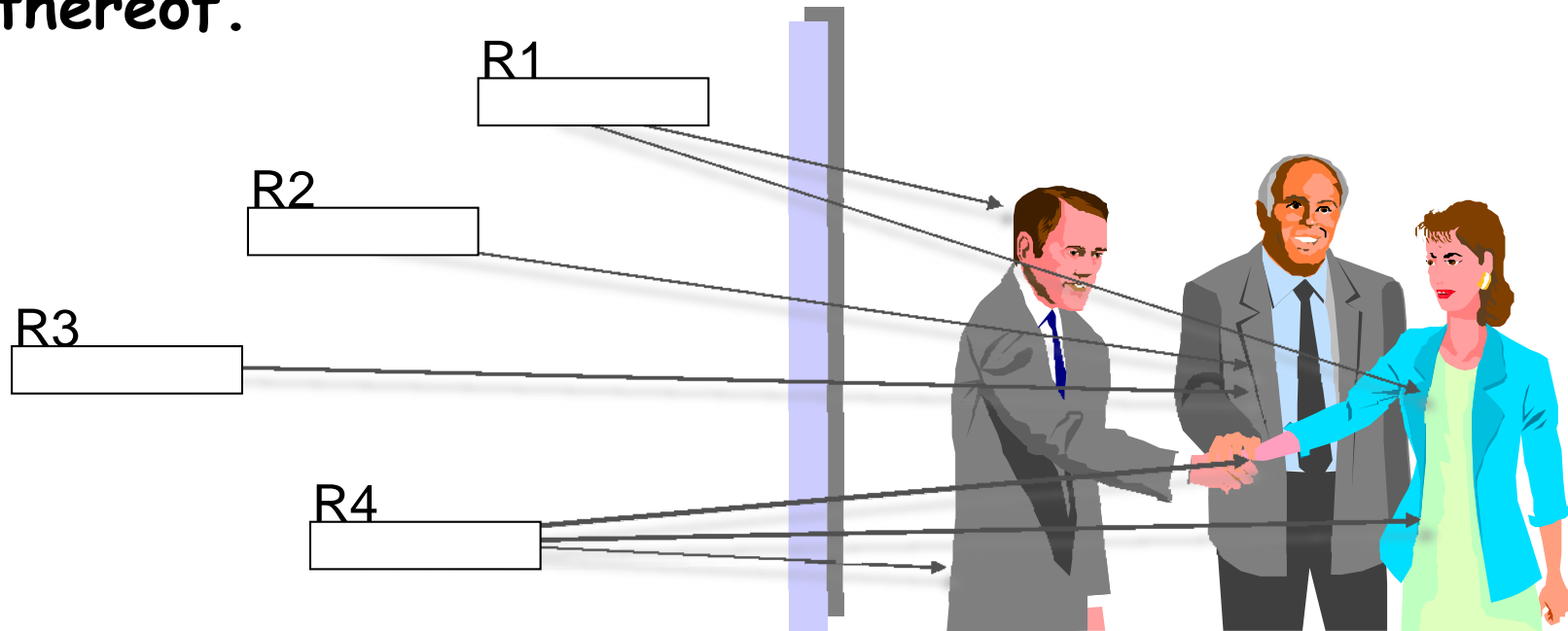
# Data Semantics

→ *Semantics* is about mapping the contents of a database into entities and relationships in the world.



## *The Bad News ...*

→ In the Relational Model, relations can be used to represent entities, relationships, or combinations thereof.



*The mapping between the contents of a database and the world is many-to-many*

# *The Entity Relationship Model*

- The *Entity-Relationship* (ER) *model* is a *conceptual* ("semantic") data model, capable of describing the meaning of the data in a database in an easy-to-understand graphical notation.
- The meaning is described in terms of a *conceptual* (or, *ER*) *schema*.
- ER schemas are comparable to class diagrams in UML.

# *Entities and Entities Sets*

- An *entity set* represents a class of objects (things, people,...) that have properties in common and an autonomous existence, e.g., City, Department, Employee, Purchase and Sale.
- An *entity* is an instance/member of an entity set, e.g., Stockholm, Helsinki, are examples of instances of the entity City; Peterson and Johanson are examples of instances of the Employee entity set.
- When meaning is unambiguous, we drop "set".

# *Examples of Entity Sets*

EMPLOYEE

DEPARTMENT

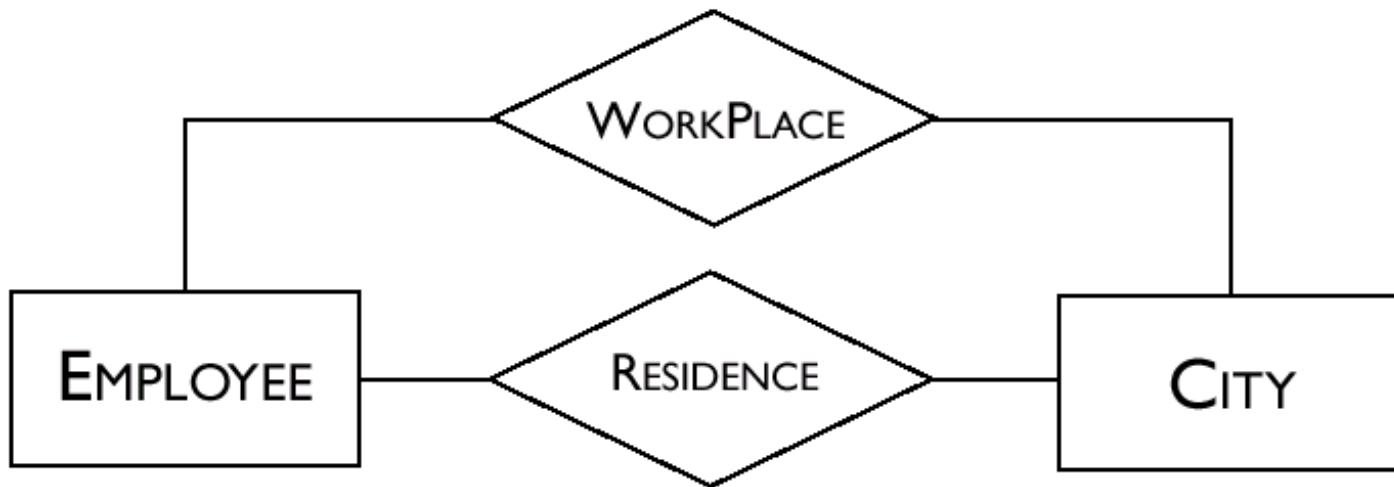
CITY

SALE

# Relationship Sets

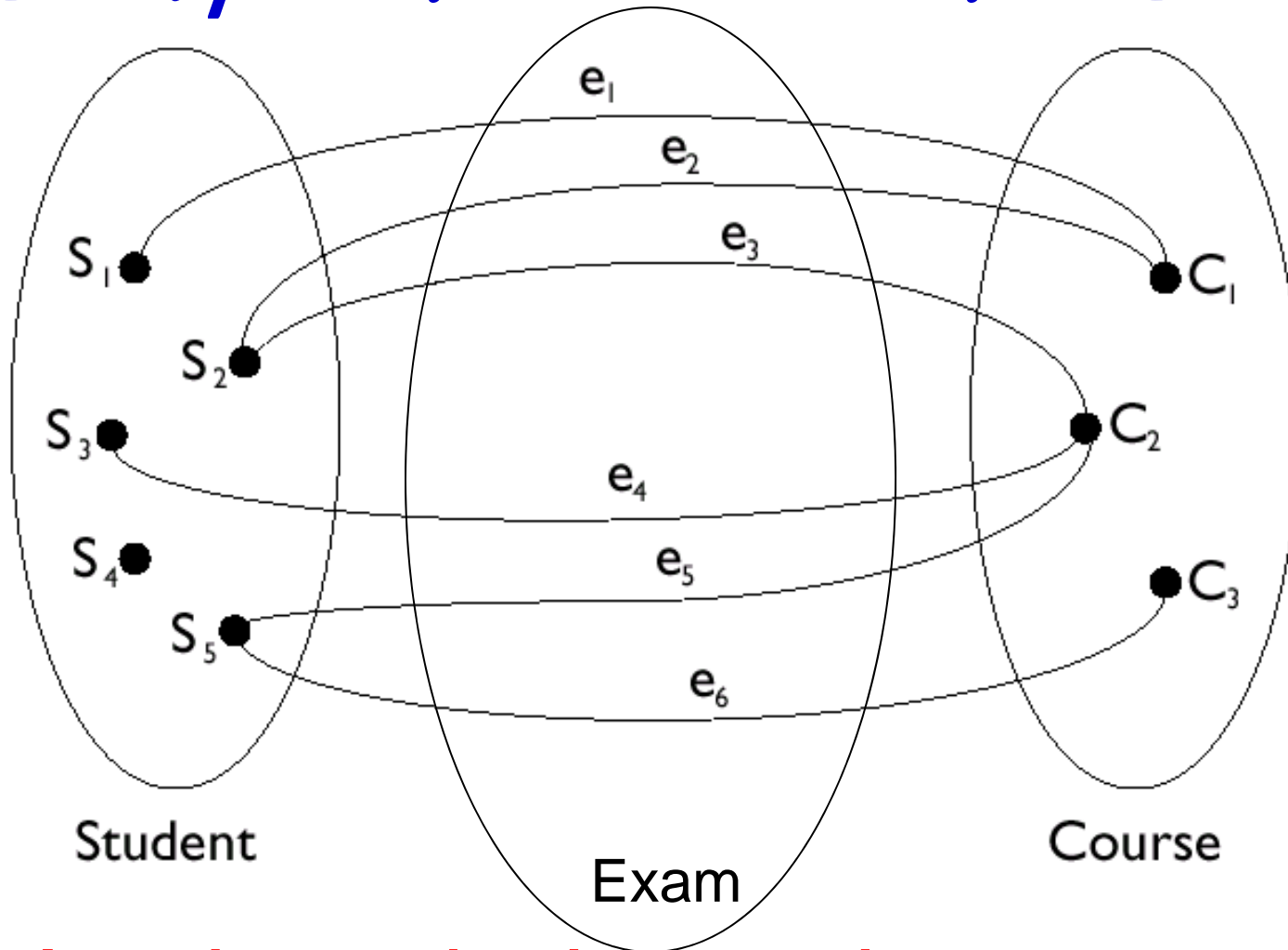
- A *relationship set* represents an association between 2 or more entity sets.
- *Residence* is a relationship set between entity sets *City* and *Employee*; *Exam* is a relationship set between the entity sets *Student* and *Course*.
- An instance of an n-ary relationship set is an n-tuple made up of entities, one for each of entity sets involved.
- The pair (Johanssen, Stockholm), or the pair (Peterson, Oslo), are relationship instances of *Residence*.
- *Note: The collection of instances of a relationship is by definition a set, not a bag; i.e., no duplicates!*

# *Examples of Relationship Sets*



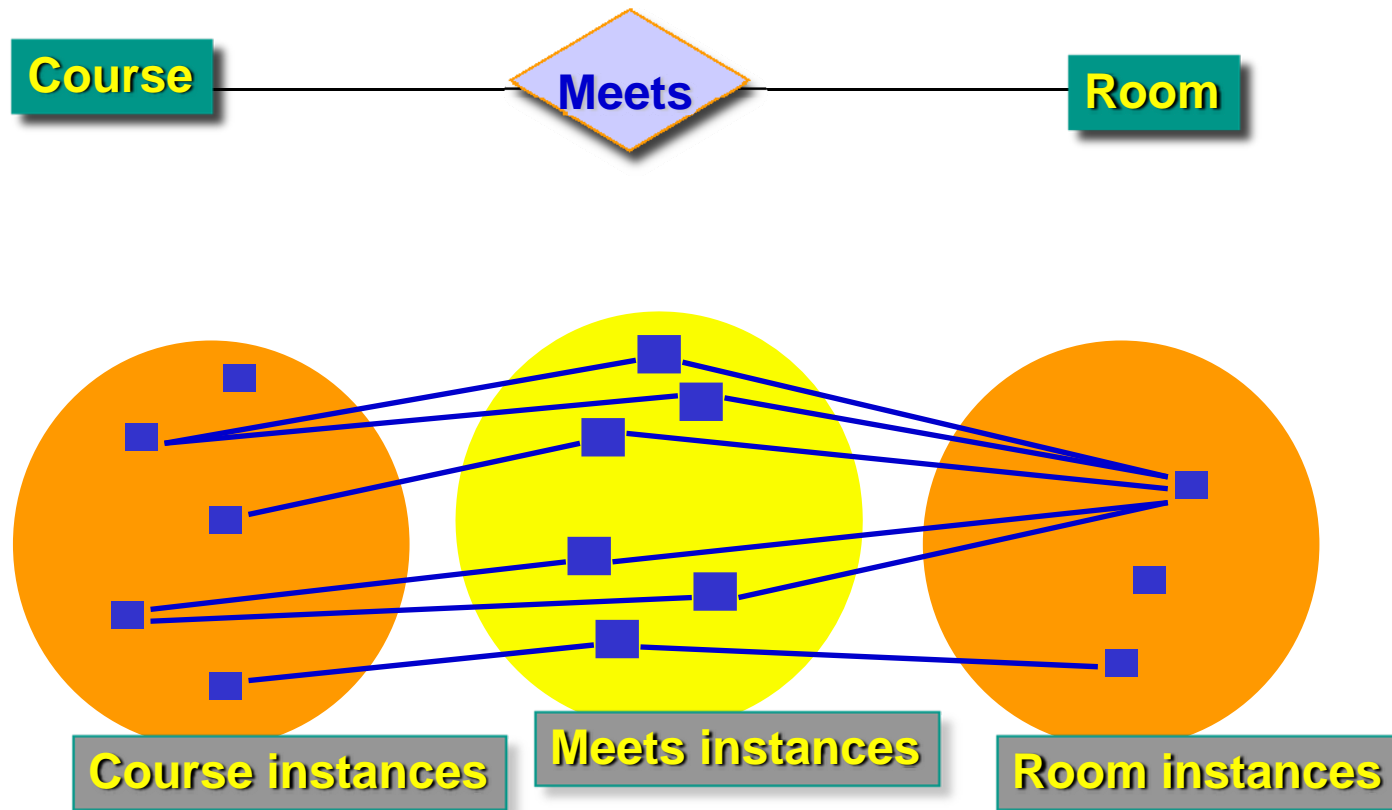


## *Example of Instances for Exam*



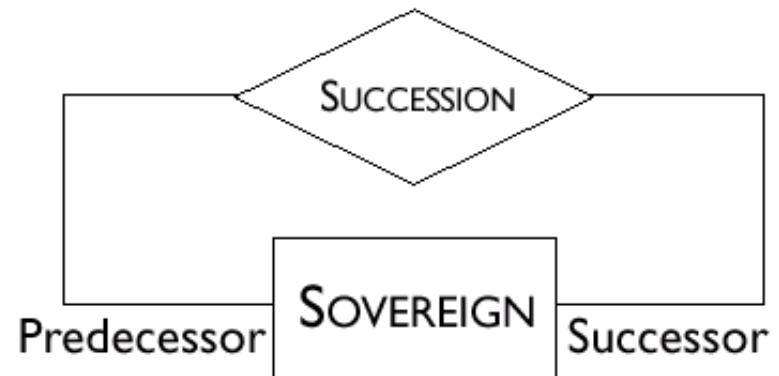
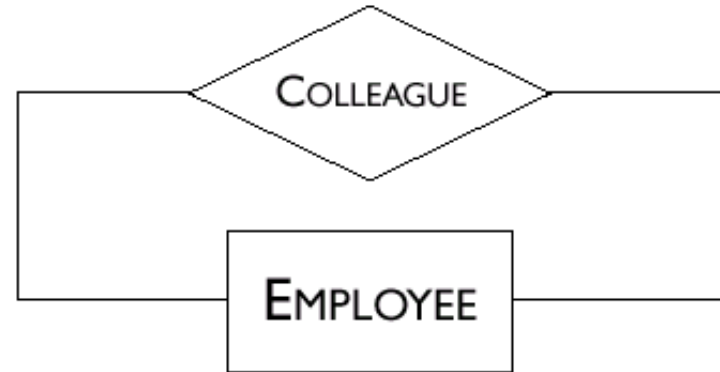
**Note: A student can't take more than one exam for a particular course!**

# *What Does A Schema Really Mean?*

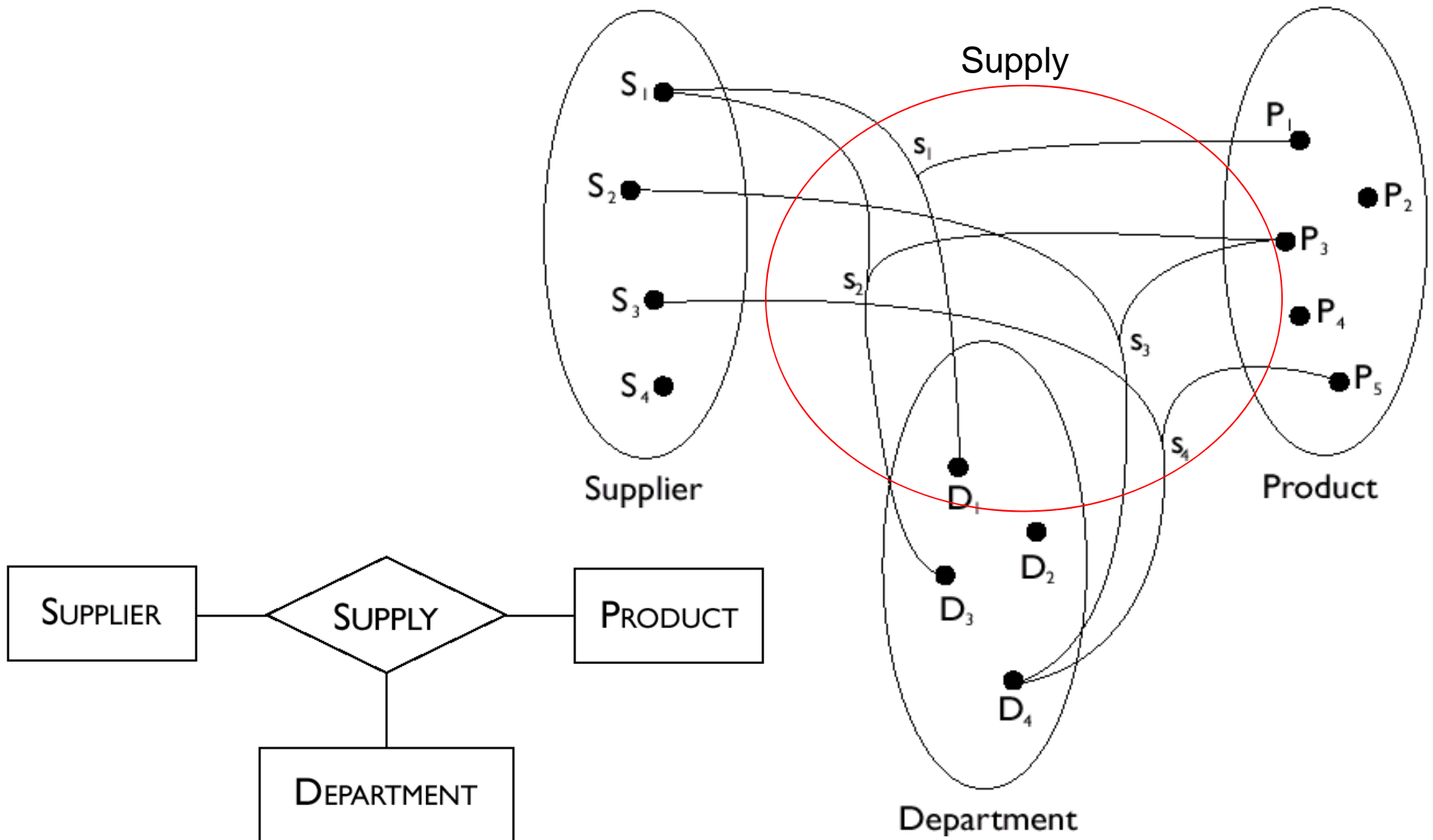


# *Recursive Relationships*

- Recursive relationships relate an entity to itself.
- Note in the second example that the relationship is not symmetric. In this case, it is necessary to indicate the two *roles* that the entity plays in the relationship.



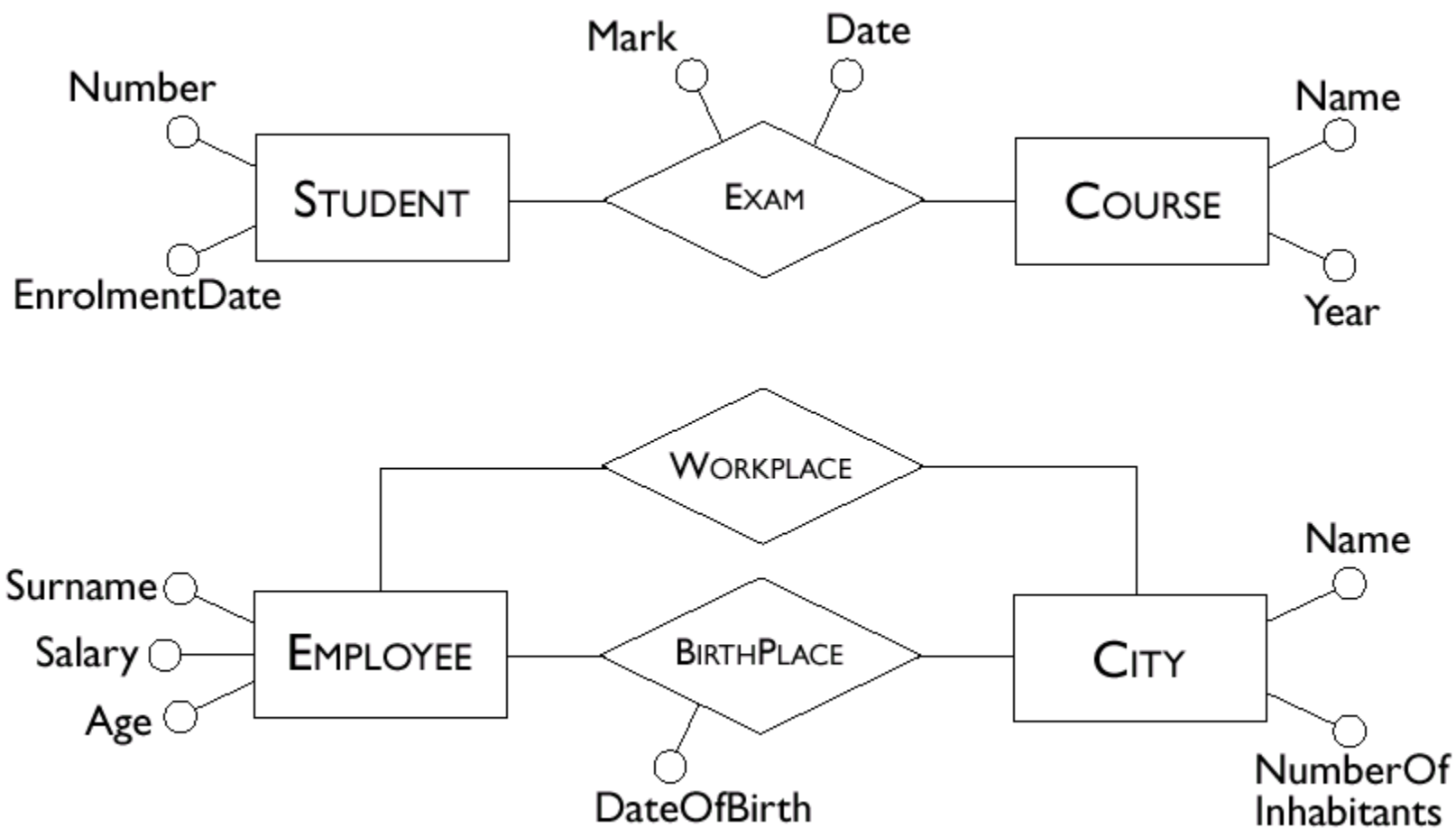
# *Ternary Relationships*



# Attributes

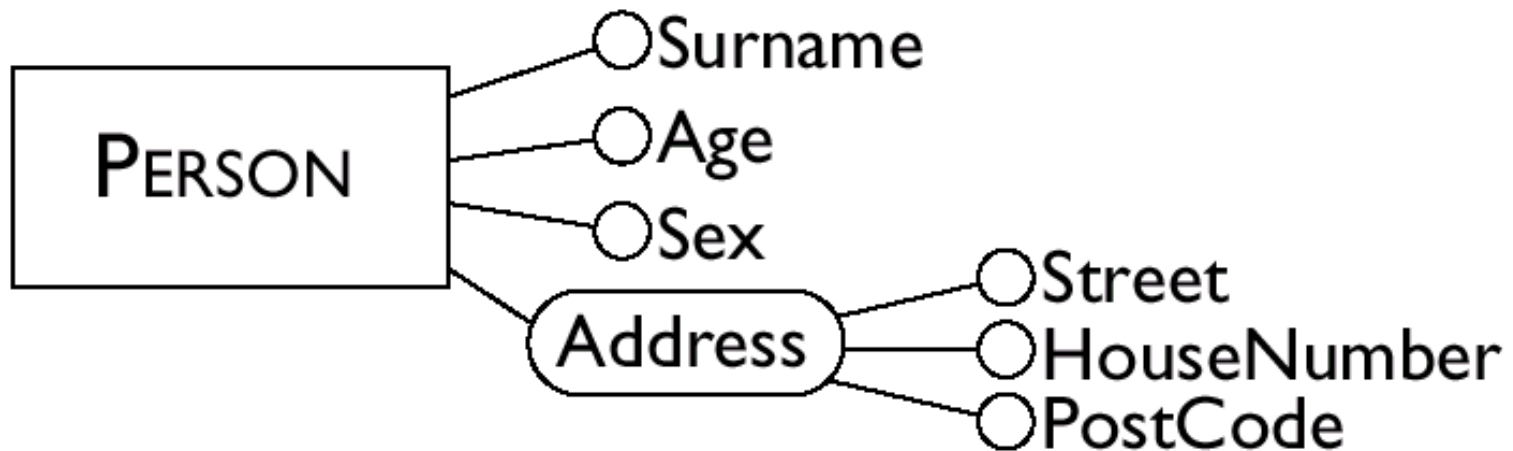
- Describe elementary properties of entities or relationships.
- For example, Surname, Salary and Age are attributes of Employee, while Date and Mark are attributes of relationship Exam between Student and Course.
- An attribute associates with each instance of an entity (or relationship) one or more values belonging to its *domain*.
- Attributes may be single-valued, or multi-valued.

# *Attribute Examples*

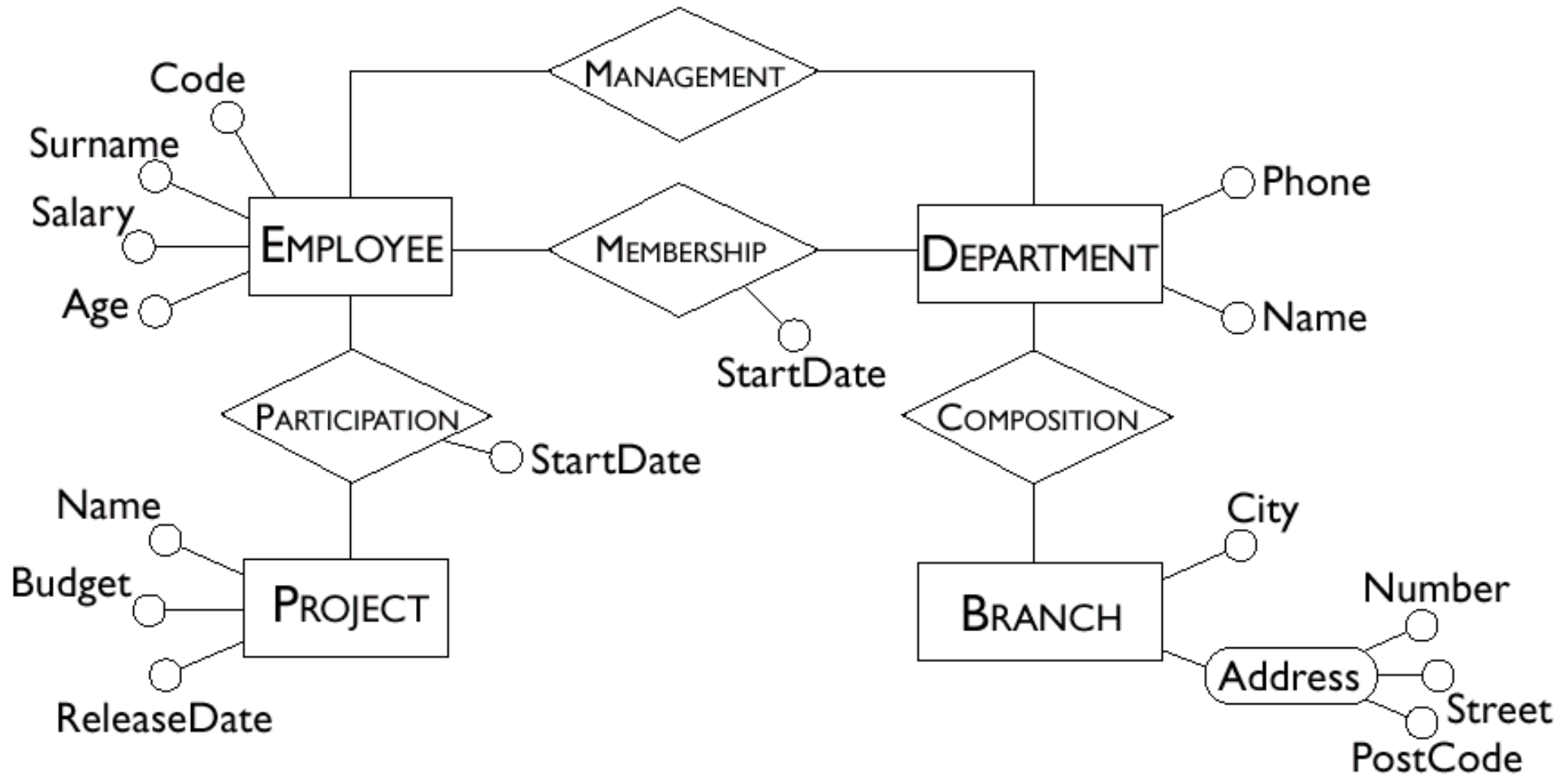


# *Composite Attributes*

→ It is sometimes convenient to group attributes of the same entity or relationship that have closely connected meaning or uses. Such groupings are called *composite attributes*.



# *Schema with Attributes*





# *Cardinalities*

- These are specified for each entity participating in a relationship and describe the maximum and minimum number of relationship instances that an entity instance can participate in.



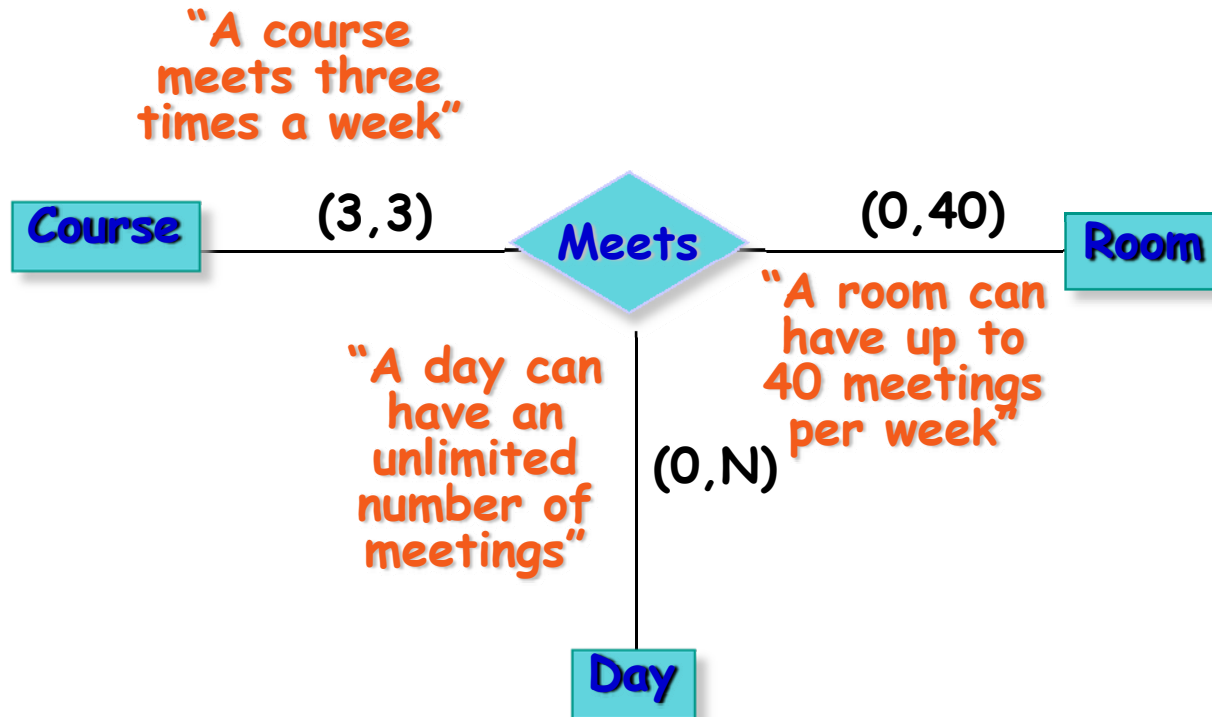
## *Cardinalities (cont'd)*

- In principle, a cardinality is any pair of non-negative integers  $(n,m)$  such that  $n \leq m$ . or a pair of the form  $(n,N)$  where  $N$  means "any number".
- If minimum cardinality is 0, entity participation in a relationship is **optional**. If minimum cardinality is 1, entity participation in a relationship is **mandatory**.
- If maximum cardinality is 1, each instance of the entity is associated at most with a single instance of the relationship; if maximum cardinality is  $N$ , then each instance of the entity is associated with an arbitrary number of instances of the relationship.

# *Cardinality Examples*

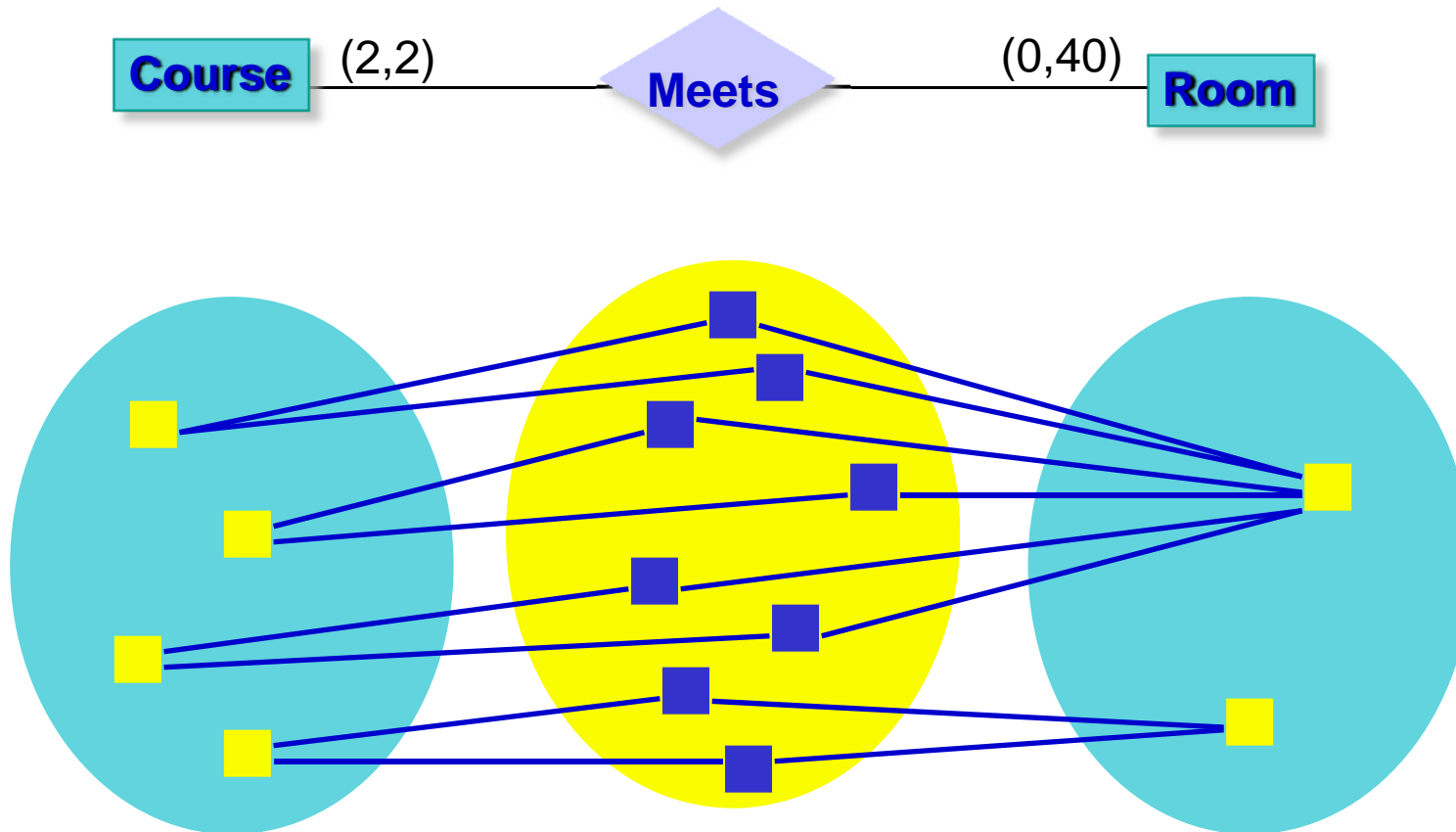


# Cardinality Example

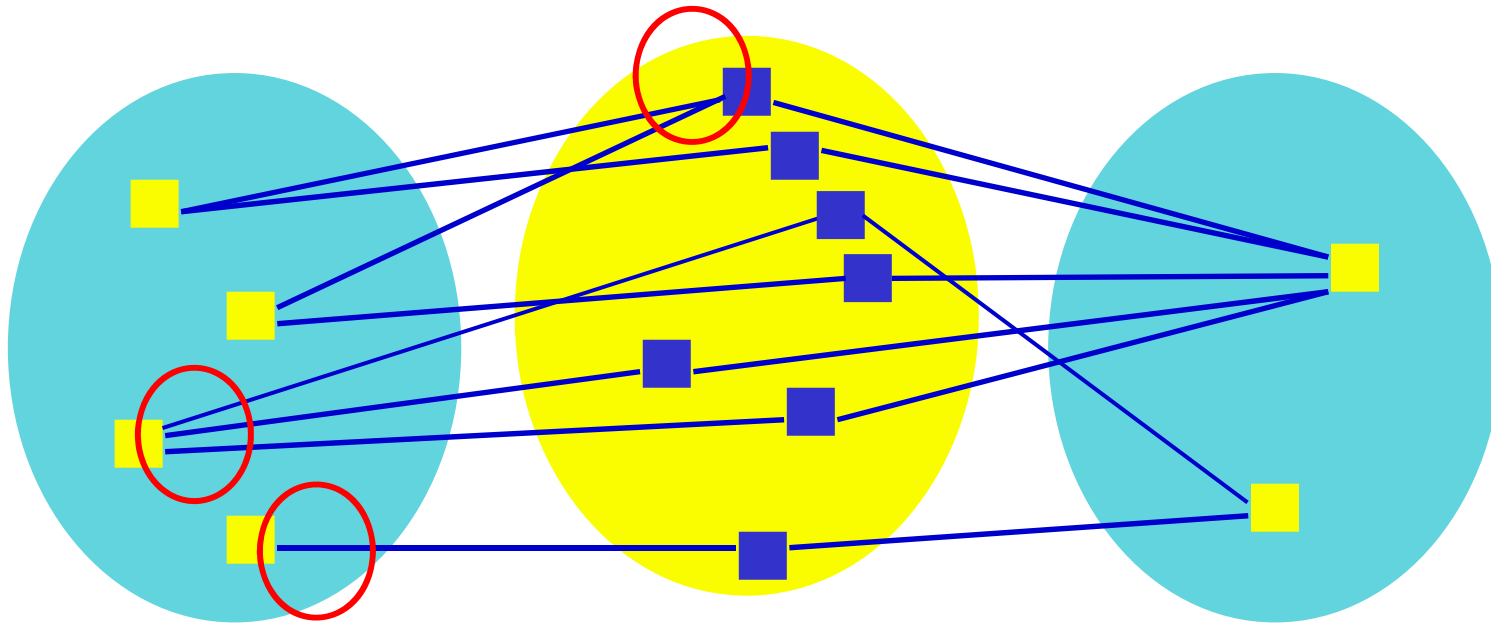


# *Instantiating ER schemas*

→ An ER schema specifies what states are possible in the world being modelled

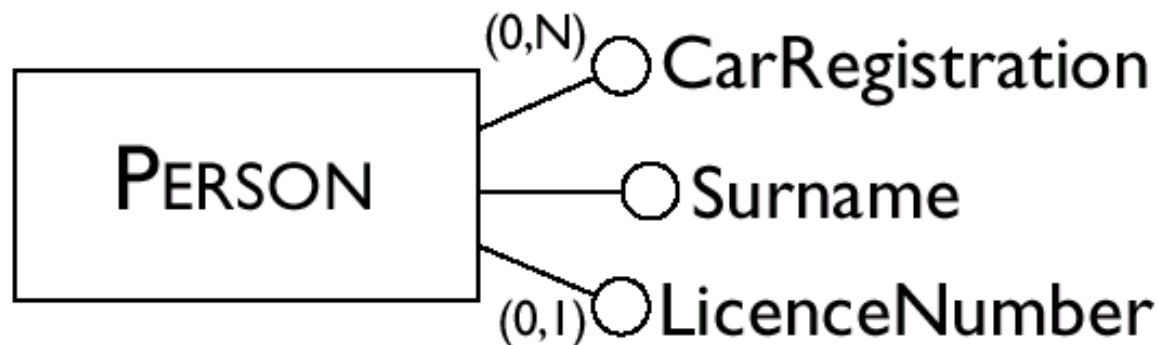


# *Illegal Instantiations*



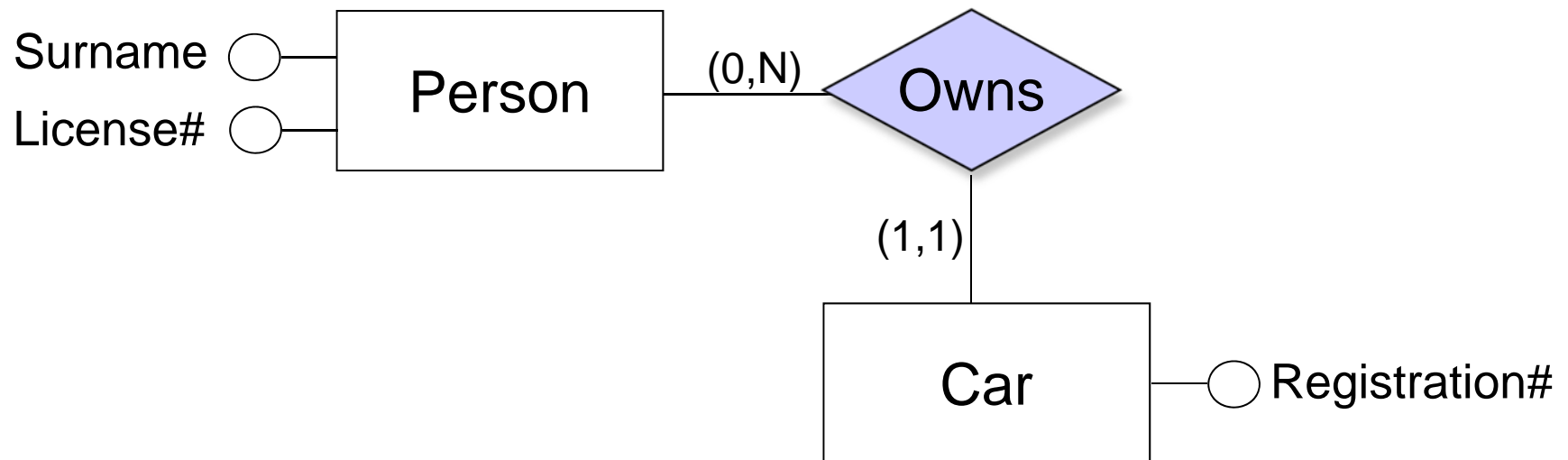
# Cardinalities of Attributes

- Describe min/max number of values an attribute can have.
- When the cardinality of an attribute is (1,1), it can be omitted (*single-valued* attributes)
- The value of an attribute, may also be null, or have several values (*multi-valued* attributes)



## *Cardinalities (cont'd)*

- Multi-valued attributes often represent situations that can be modelled with additional entities.
- For example, the ER schema of the previous slide can be revised into ( ... AND looks better as):



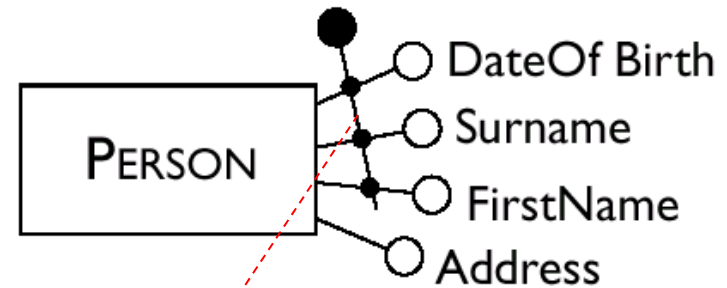
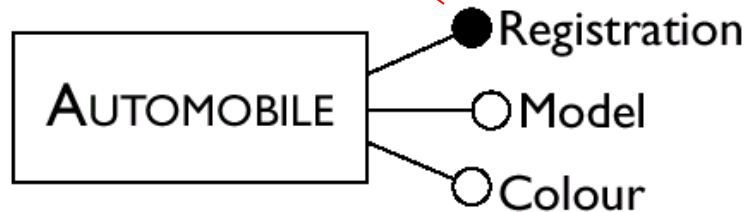


# Keys

- *Keys* consist of minimal sets of attributes which identify uniquely instances of an entity set.
- For example, socialInsurance# may be a key for Person. Alternatively, firstName, middleName, lastName, address may be a key.
- In most cases, a key is formed by one or more attributes of the entity itself (*internal* keys).
- Sometimes, other entities are involved in the identification (*external keys, weak entities*).
- A key for a relationship consists of keys of entities it relates.

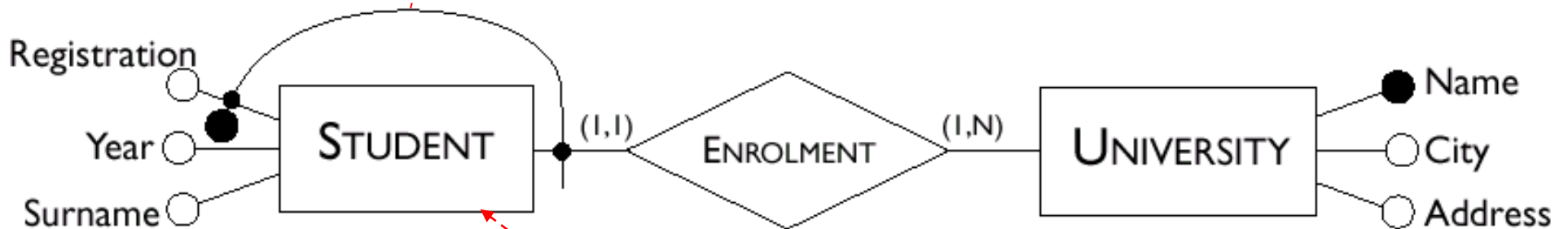
# Examples of Keys

*internal, single-attribute*



*internal, multi-attribute*

*external, multi-attribute*

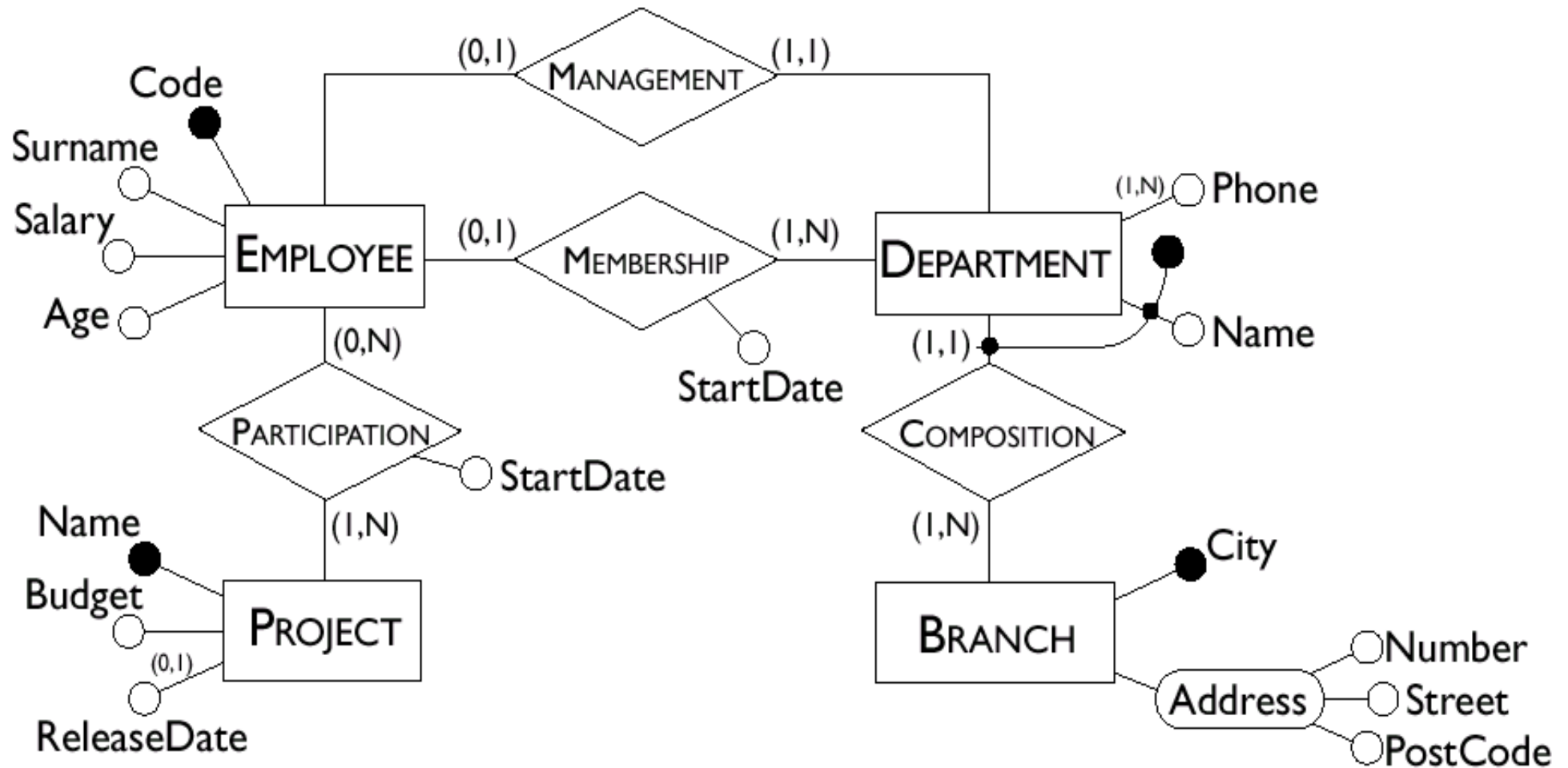


*Weak entity*

## *General Observations on Keys*

- A key may consist of one or more attributes, provided that each of them has (1,1) cardinality.
- A external key can involve one or more entities, provided that each of them is member of a relationship to which the entity to be identified participates with cardinality equal to (1,1).
- A external key may involve an entity that has itself a external key, as long as cycles are not generated.
- Each entity must have at least one (internal or external ) key. If there is more than one key, one of them is labelled as *primary*.

# *Schema with Keys*

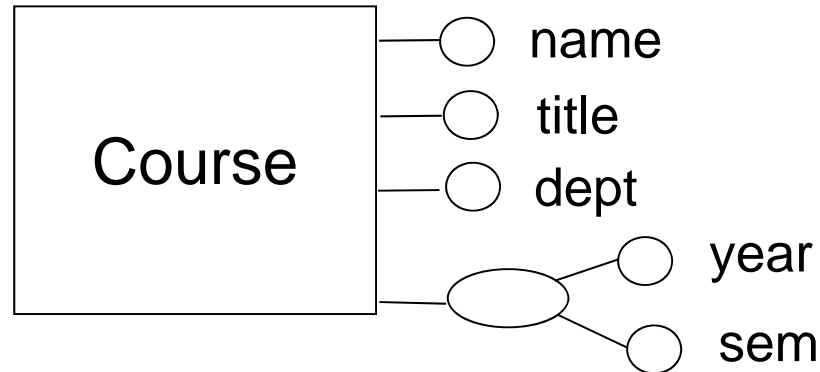


# *Modelling an Application with Keys*

Keys constitute a powerful mechanism for modelling an application. Assume we want a database storing information about course lectures for a semester, e.g., 2008F.

- Suppose first that we use the key `coursename, day, hour` for the `Lecture` entity. What does this say about the application?
- Suppose now we use only `coursename` as identifier for `Lecture`. What does this say about the application??

# *Consider...*

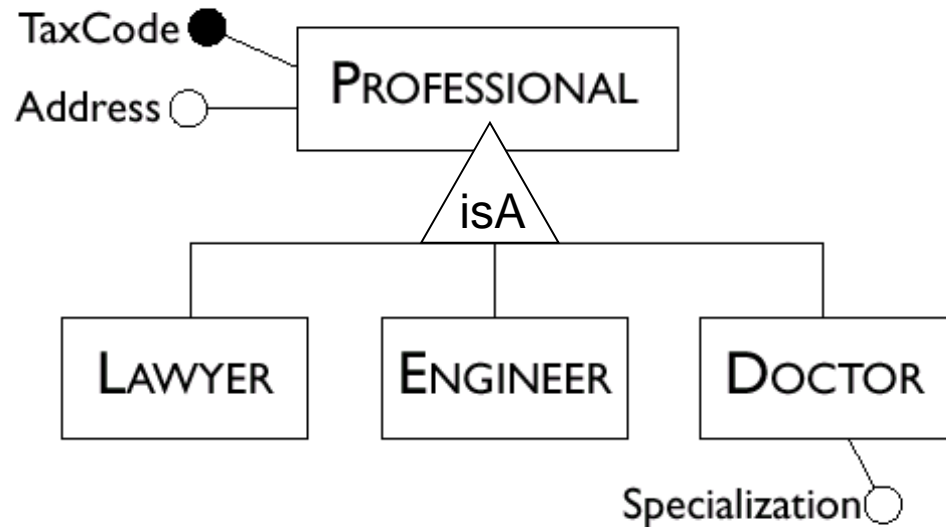
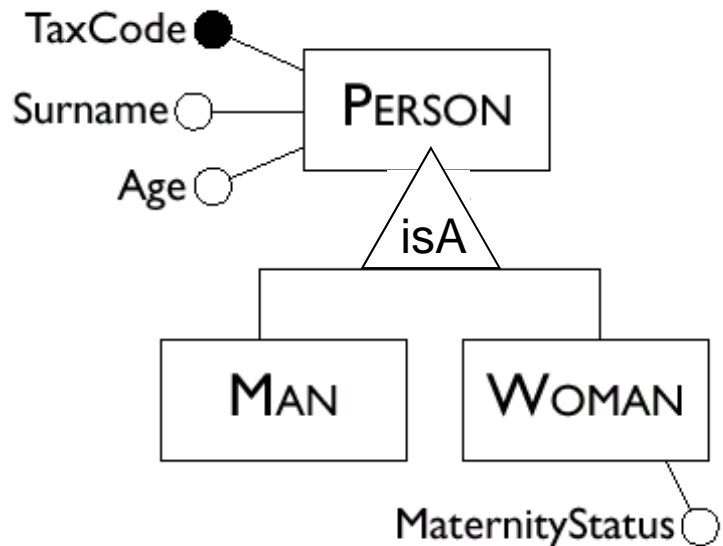


■ Suppose all attributes are single-valued, and we want information on course offerings over the years. What does each of the following identifiers say about the application?

- ✓ name;
- ✓ name, sem, year;
- ✓ sem, year;
- ✓ name, dept;
- ✓ dept, year.

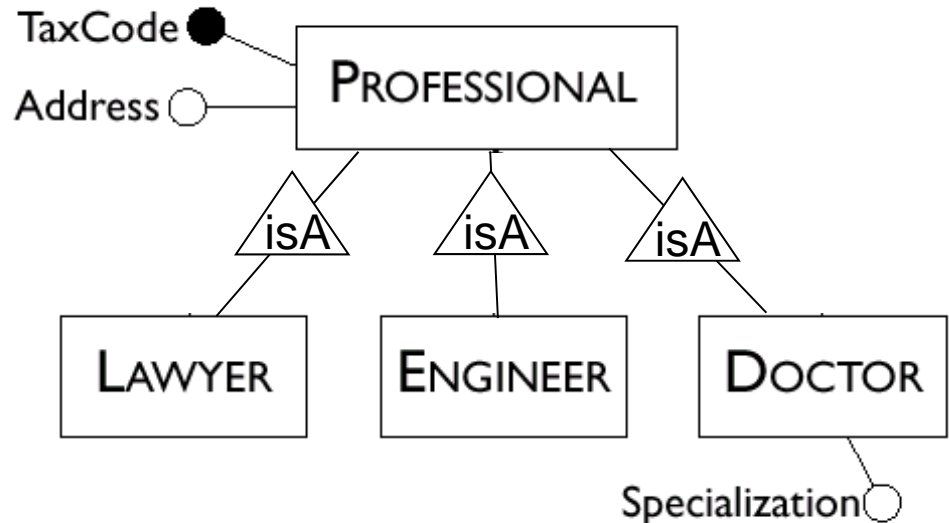
# Entity (Class) Hierarchies

- Entity hierarchies organize a group of entity sets according to their generality/specificity, e.g., Person is more general than Man, (Also called *generalization hierarchies*)



# Overlap Constraints

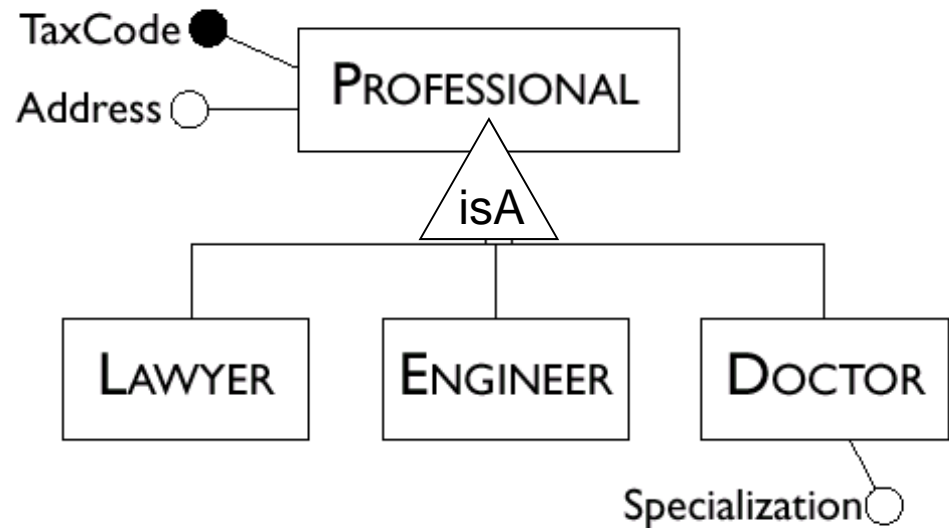
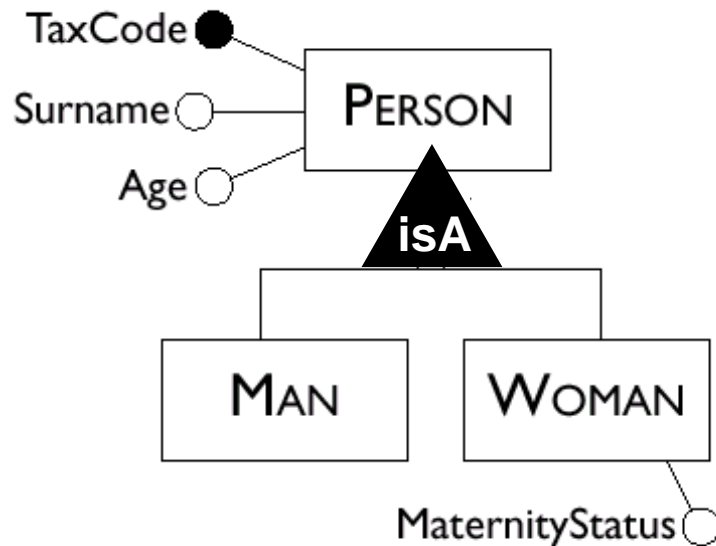
- Sometimes we want to specify that the children of an entity do/do not overlap.
- For example in the previous slide, Man/Woman don't overlap, neither do Lawyer/Engineer/Doctor.
- To indicate possible overlap, we use separate isA links. E.g.,





# Covering Constraints

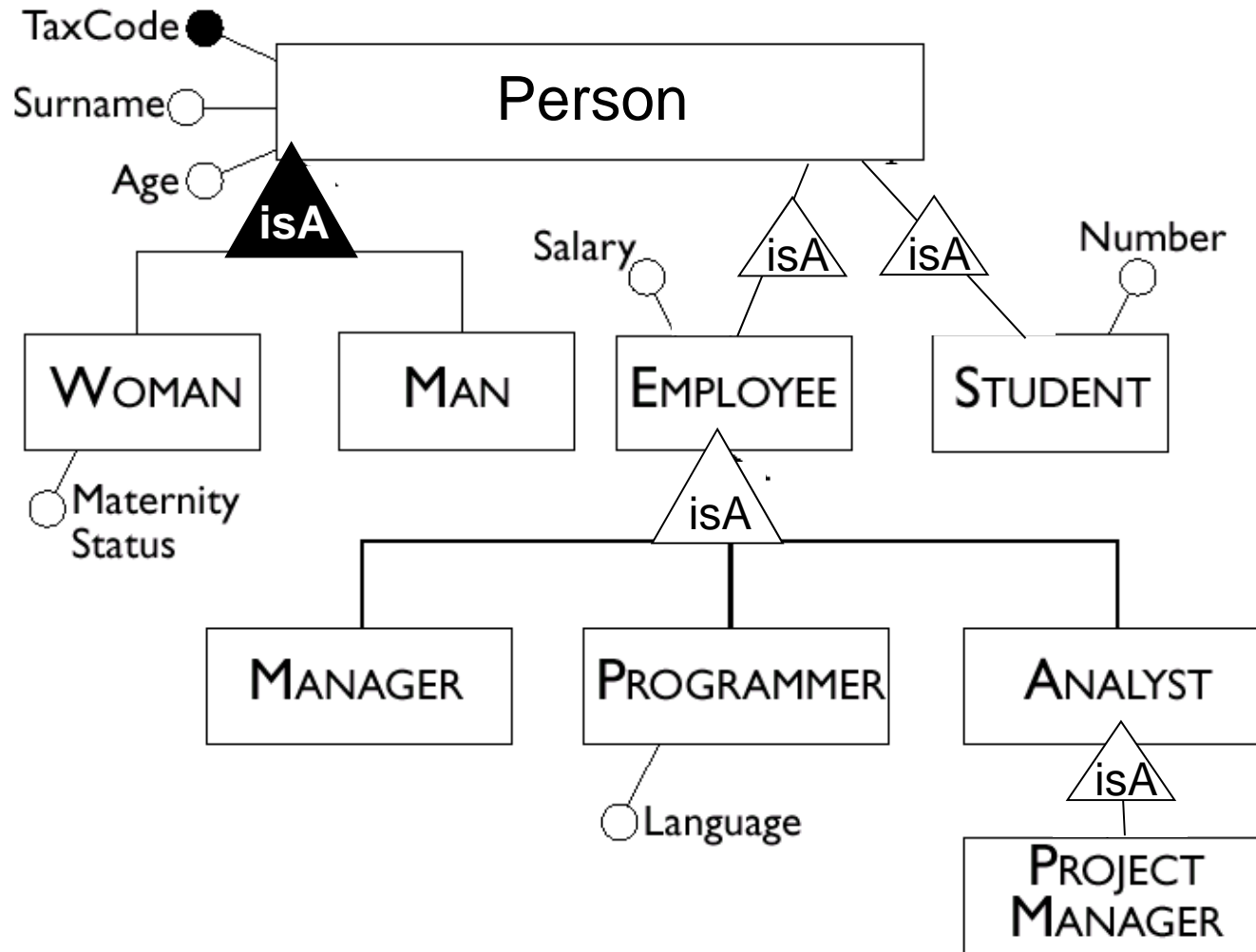
→ Now we want to be able to say that instances of the children of an entity include all instances of their parent (I.e., cover it). We use a dark triangle notation for this.



# *Properties of Generalization*

- Every instance of a child is also an instance of the parent.
- Every property of the parent entity (attribute, key, relationship) is also a property of a child entity. This property of generalizations is known as *inheritance*.

# *An Entity Hierarchy*

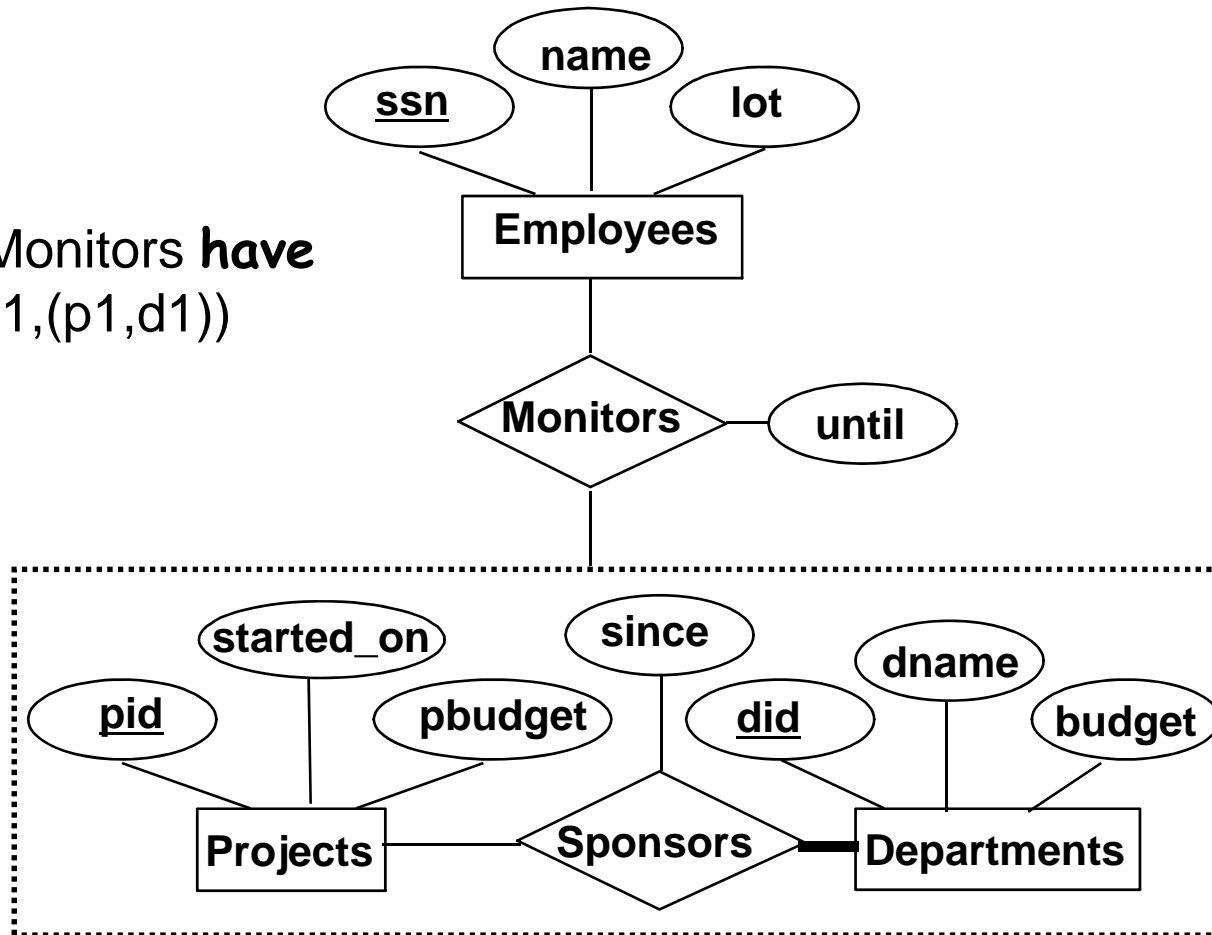


# *Aggregation*

- 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.

# *An Example*

Instances of Monitors have  
the form (e1,(p1,d1))



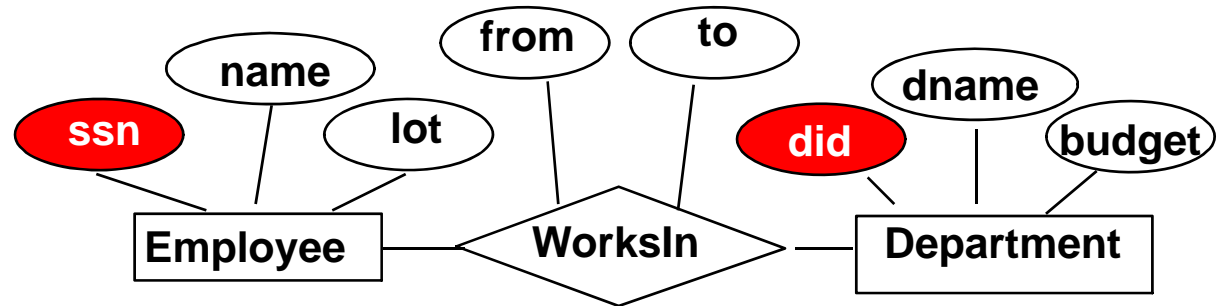
# *Conceptual Design with the ER Model*

- Design choices: Should a concept be modeled as an entity, an attribute, or a relationship?
- Constraints on the ER Model: A lot of data semantics can (and should) be captured; but some constraints cannot be captured by ER diagrams.

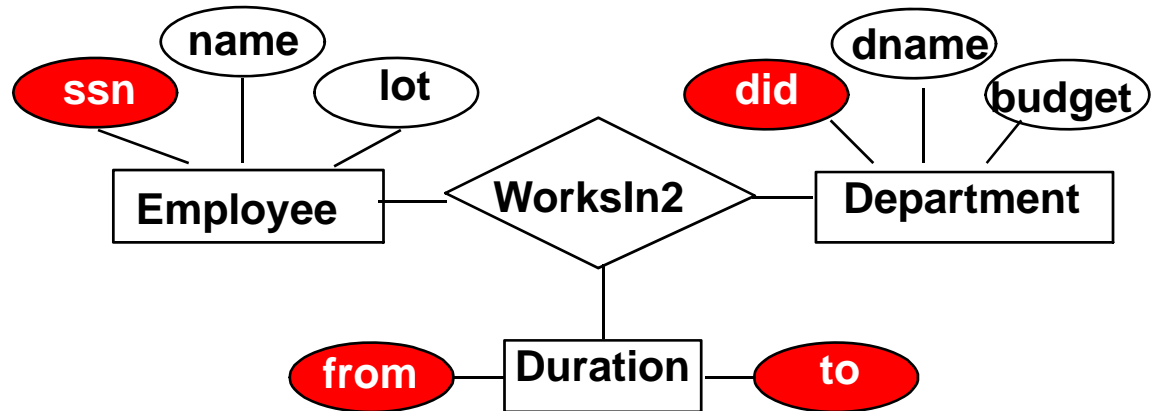
## *Some Rules of Thumb*

- If a concept has significant properties and/or describes classes of objects with an autonomous existence, it is appropriate to represent it as an entity.
- If a concept has a simple structure, and no relevant properties associated with it, it can be represented with attributes of other concepts to which it refers.
- If a concept provides a logical link between 2+ entities, it is convenient to represent it with a relationship.
- If 1+ concepts are particular cases of another concept, it is convenient to represent them in terms of a generalization relationship.

# Entity vs. Attribute



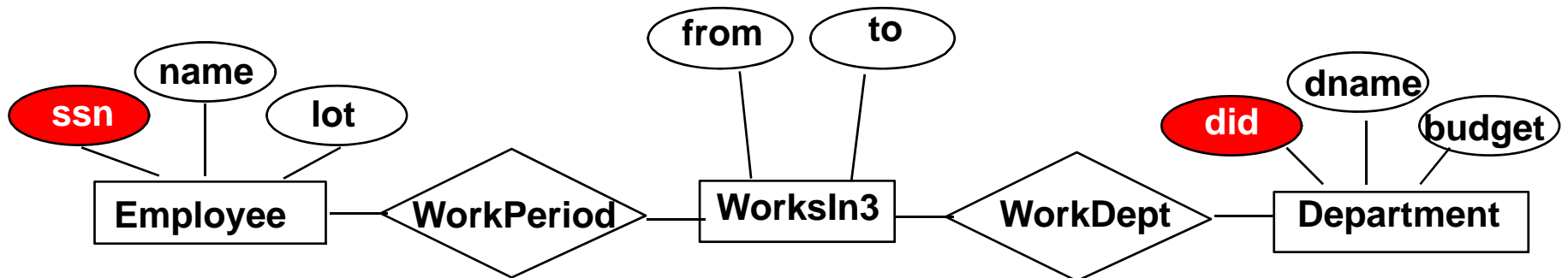
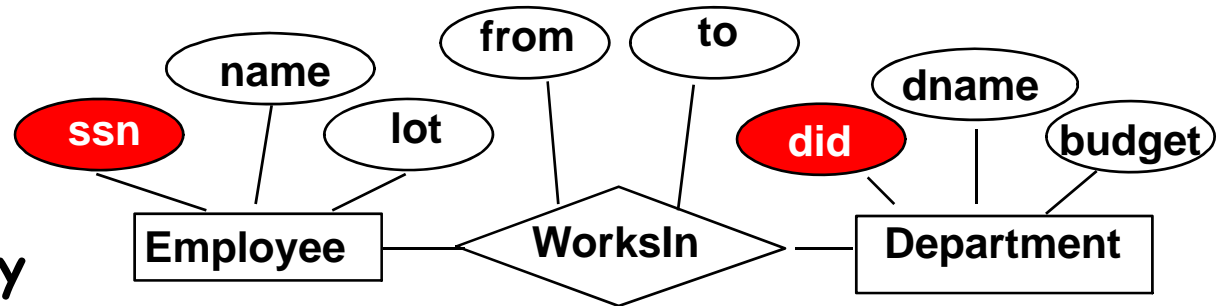
WorksIn does not allow an employee to work in a department for two or more periods.





# Entity vs. Relationship

WorksIn can also be turned into an entity to avoid the problem mentioned earlier



# *Database Design -- Early Phases*

- To design a database, we need to identify requirements, generate a conceptual design, using an ER schema, and from it generate a logical design using a relational schema.
- In the following we discuss how can one go from an informal description of requirements to a conceptual design.

# *From Text to an ER Schema*

We wish to create a database for a company that runs training courses. For this, we must store data about trainees and instructors. For each course participant (about 5,000 in all), identified by a code, we want to store her social security number, surname, age, sex, place of birth, employer's name, address and telephone number, previous employers (and periods employed), the courses attended (there are about 200 courses) and the final assessment for each course. We need also to represent the seminars that each participant is attending at present and, for each day, the places and times the classes are held.

Each course has a code and a title and any course can be given any number of times. Each time a particular course is given, we will call it an 'edition' of the course. For each edition, we represent the start date, the end date, and the number of participants. If a trainee is self-employed, we need to know her area of expertise, and, if appropriate, her title. For somebody who works for a company, we store the level and position held. For each instructor (about 300), we will show the surname, age, place of birth, the edition of the course taught, those taught in the past and the courses that the tutor is qualified to teach. All the instructors' telephone numbers are also stored. An instructor can be permanently employed by the training company or freelance.

# Example, Annotated

We wish to create a database for a company that runs training courses. For this, we must store data about the *trainees* and the *instructors*. For each *course participant* (about 5,000), identified by a code, we want to store her social security number, surname, age, sex, place of birth, employer's name, address and telephone number, previous employers (and periods employed), the courses attended (there are about 200 courses) and the final assessment for each course. We need also to represent the *seminars* that each participant is attending at present and, for each day, the places and times the classes are held.

Each *course* has a code and a title and any course can be given any number of times. Each time a particular course is given, we will call it an 'edition' of the course. For each edition, we represent the start date, the end date, and the number of participants. If a trainee is self-employed, we need to know her area of expertise, and, if appropriate, her title. For somebody who works for a company, we store the level and position held. For each *instructor* (about 300), we will show the surname, age, place of birth, the edition of the course taught, those taught in the past and the courses that the *tutor* is qualified to teach. All the instructors' telephone numbers are also stored. An instructor can be permanently employed by the training company or freelance.

# Example, Annotated

We wish to create a database for a company that runs training courses. For this, we must store data about the *trainees* and the *instructors*. For each *course participant* (about 5,000), identified by a code, we want to store her social security number, surname, age, sex, place of birth, employer's name, address and telephone number, previous employers (and periods employed), the courses attended (there are about 200 courses) and the final assessment for each course. We need also to represent the *seminars* that each participant is attending at present and, for each day, the places and times the classes are held.

Each *course* has a code and a title and any course can be given any number of times. Each time a particular course is given, we will call it an 'edition' of the course. For each edition, we represent the start date, the end date, and the number of participants. If a trainee is self-employed, we need to know her area of expertise, and, if appropriate, her title. For somebody who works for a company, we store the level and position held. For each *instructor* (about 300), we will show the surname, age, place of birth, the edition of the course taught, those taught in the past and the courses that the *tutor* is qualified to teach. All the instructors' telephone numbers are also stored. An instructor can be permanently employed by the training company or freelance.

# *Glossary*

<b>Term</b>	<b>Description</b>	<b>Synonym</b>	<b>Links</b>
Trainee	Participant in a course. Can be an employee or self-employed.	Participant	Course, Company
Instructor	Course tutor. Can be freelance.	Tutor	Course
Course	Course offered. Can have various editions.	Seminar	Instructor, Trainee
Company	Company by which a trainee is employed or has been employed.		Trainee

# *Structuring of Requirements (I)*

<b>Phrases of a general nature</b>
We wish to create a database for a company that runs training courses. We wish to maintain data for the trainees and the instructors.
<b>Phrases relating to trainees</b>
For each trainee (about 5000), identified by a code, the database will hold the social security number, surname, age, sex, town of birth, current employer, previous employers (along with the start date and the end date of the period employed), the editions of the courses the trainee is attending at present and those he or she has attended in the past, with the final marks out of ten.
<b>Phrases relating to the employers of trainees</b>
For each employer of a trainee the database will store name, address and telephone number.

# *Structuring of Requirements (II)*

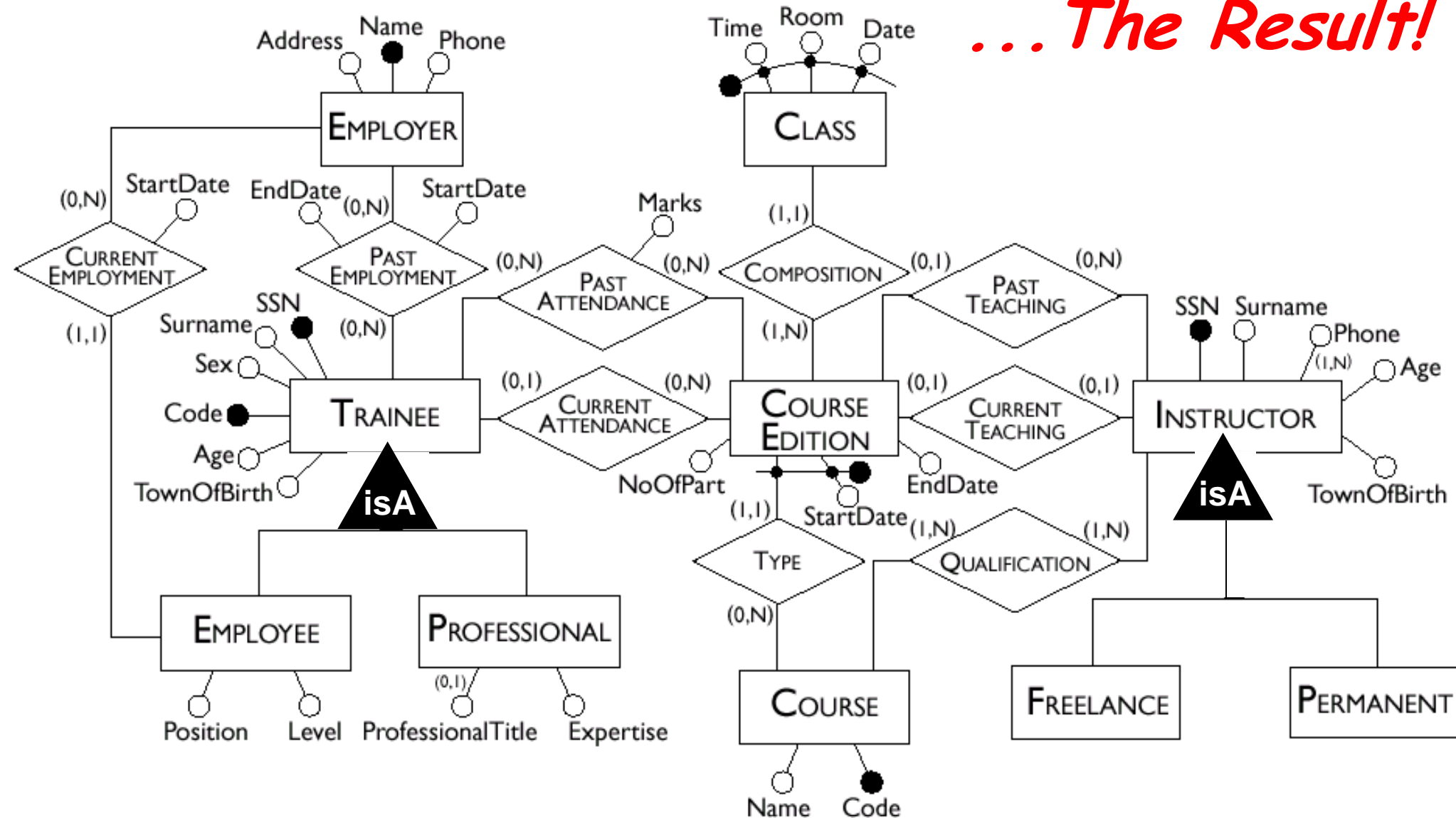
<b>Phrases relating to courses</b>
For each course (about 200), we will hold the name and code. Each time a particular course is given, we will call it an <i>édition</i> of the course. For each edition, we will hold the start date, the end date, and the number of participants. For the editions currently in progress, we will hold the dates, the classrooms and the times in which the classes are held.
<b>Phrases relating to specific types of trainee</b>
For a trainee who is a self-employed professional, we will hold the area of expertise and, if appropriate, the professional title. For a trainee who is an employee, we will hold the level and position held.
<b>Phrases relating to instructors</b>
For each instructor (about 300), we will hold surname, age, town of birth, all telephone numbers, the edition of courses taught, those taught in the past and the courses the instructor is qualified to teach. The instructors can be permanently employed by the training company or can be freelance.



# *Design Decisions*

- Consider *address* of a trainee. Is it an entity, an attribute or relationship?
- Consider *address* for a telephone company database, which has to keep track of how many and what type of phones are available in any one household, who lives there (there may be several phone bills going to the same address) etc.
- How do we represent employment of a trainee by a particular employer?
- How do we represent a course edition?

... The Result!



# *Documentation of an E-R Schema - - The Data Dictionary*

- An ER schema is rarely sufficient by itself to represent all aspects of an application in detail.
- It is therefore important to complement every ER schema with support documentation, that facilitates the interpretation of the schema and describes properties of the data that cannot be expressed directly by the constructs of the model.
- A widely-used documentation concept for conceptual schemas is the *business rule*.

# *Business Rules*

- Business rules are used to describe the properties of an application, e.g., the fact that an employee cannot earn more than his or her manager.
- A business rule can be:
  - ✓ the *description* of a concept relevant to the application (also known as a *business object*),
  - ✓ an *integrity constraint* on the data of the application,
  - ✓ or, a *derivation rule* whereby information can be derived from other information within a schema.

# *The Data Dictionary*

- Descriptive business rules can be organized as a *data dictionary*. This is made up of three tables: the first describes the entities of the schema, the second describes the relationships.
- Business rules are included in the third table;
  - ✓ constraints can be expressed as follows:  
    <concept> must/must not <expression on concepts>
  - ✓ derivations can be expressed as follows:  
    <concept> is obtained by <operations on concepts>

# *Example of a Data Dictionary*

<b>Entity</b>	<b>Description</b>	<b>Attributes</b>	<b>Identifier</b>
EMPLOYEE	Employee working in the company.	Code, Surname, Salary, Age	Code
PROJECT	Company project on which employees are working.	Name, Budget, ReleaseDate	Name
....	....	....	....

<b>Relationship</b>	<b>Description</b>	<b>Entities involved</b>	<b>Attributes</b>
MANAGEMENT	Associate a manager with a department.	Employee (0,1), Department (1,1)	
MEMBERSHIP	Associate an employee with a department.	Employee (0,1) Department (1,N)	StartDate
....	....	....	....

# *Examples of Business Rules*

## **Constraints**

**(BR1)** The manager of a department must belong to that department.

**(BR2)** An employee must not have a salary greater than that of the manager of the department to which he or she belongs.

**(BR3)** A department of the Rome branch must be managed by an employee with more than 10 years' employment with the company.

**(BR4)** An employee who does not belong to a particular department must not participate in any project.

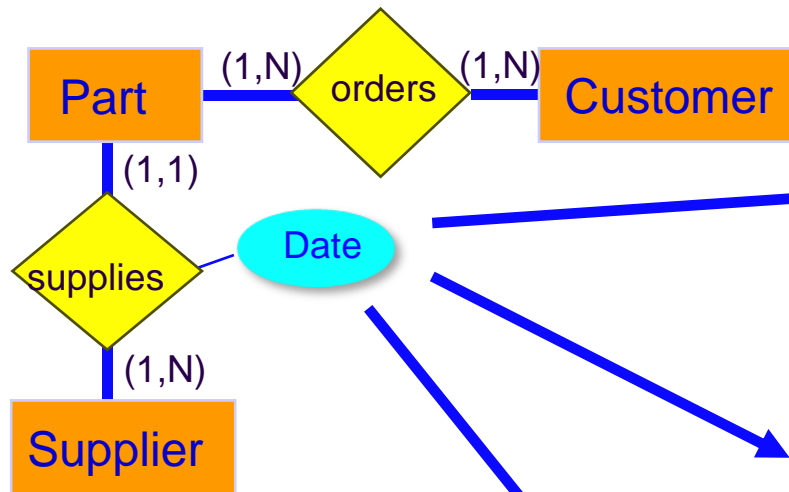
....

## **Derivations**

**(BR5)** The budget for a project is obtained by multiplying the sum of the salaries of the employees who are working on it by 3.

....

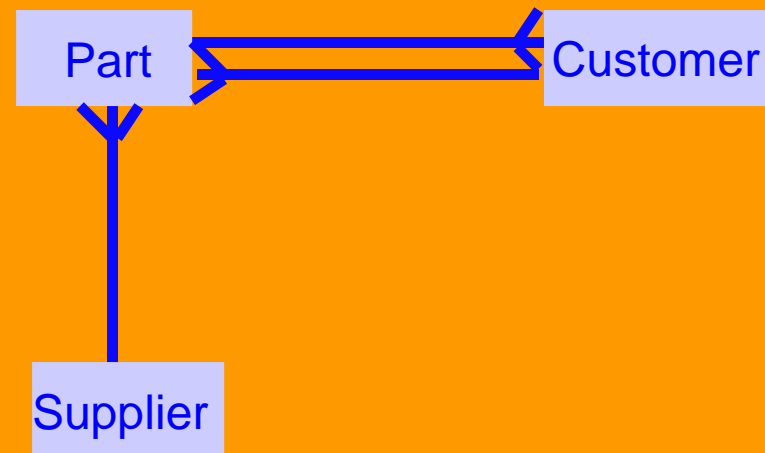
# Designing a Database Schema



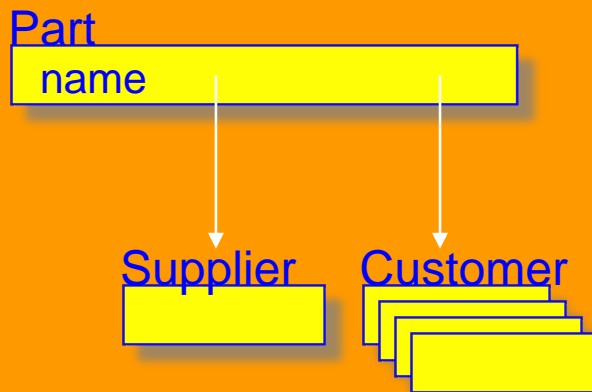
## Relational

Part(Name,Description,Part#)  
Supplier(Name, Addr)  
Customer(Name, Addr)  
Supplies(Name,Part#, Date)  
Orders(Name,Part#)

## Network



## Hierarchical



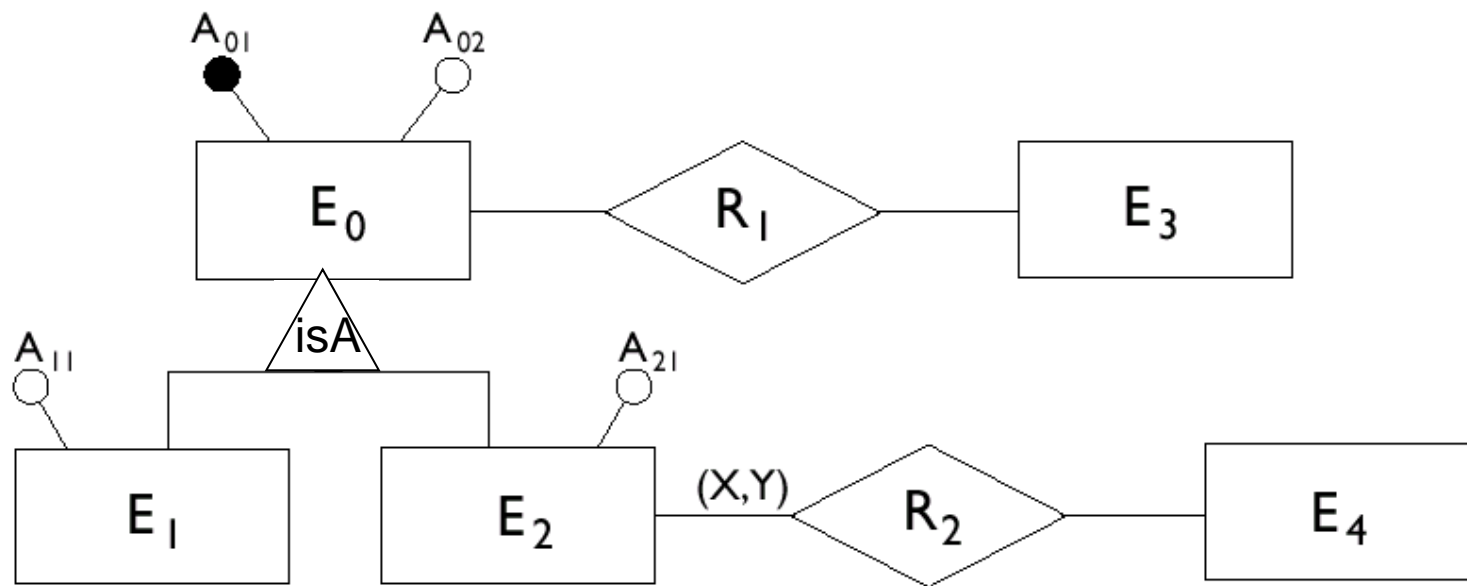


# *(Relational) Database Design*

- Given a conceptual schema (ER, but could also be UML), generate a logical (relational) schema.
- This is *not* just a simple translation from one model to another for two main reasons:
  - ✓ not all the constructs of the ER model can be translated naturally into the relational model;
  - ✓ the schema must be restructured in such a way as to make the execution of the projected operations as efficient as possible.

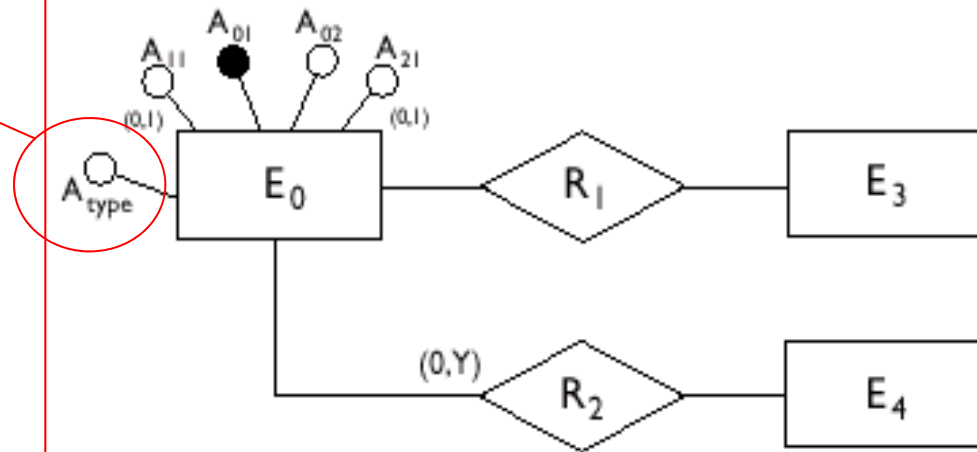
# Removing Generalizations

- The relational model does not allow direct representation of generalizations that may be present in an ER diagram.
- For example, here is an ER schema with generalizations:

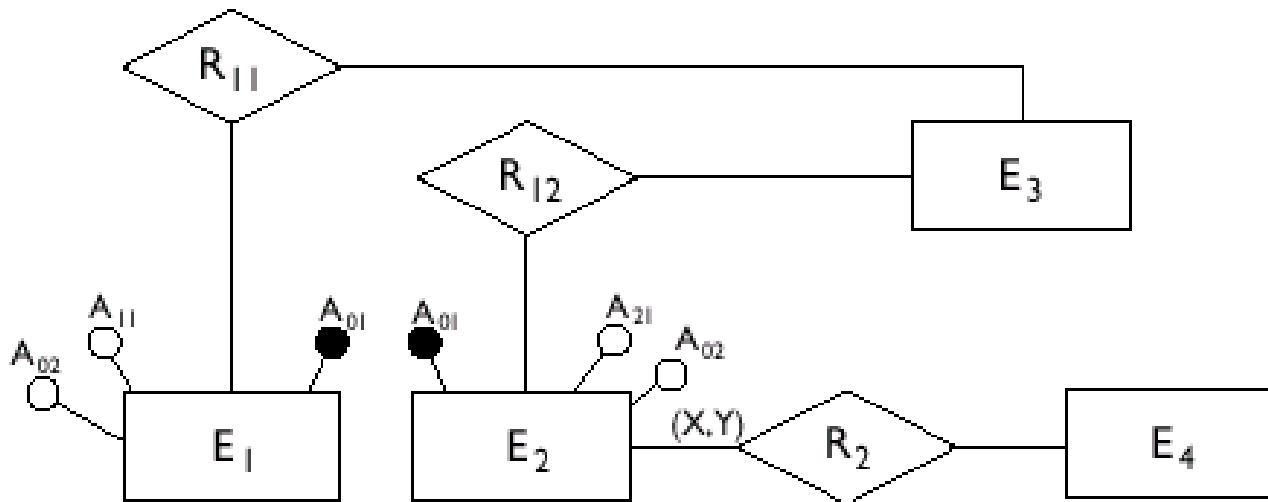


## Option 1

Note!



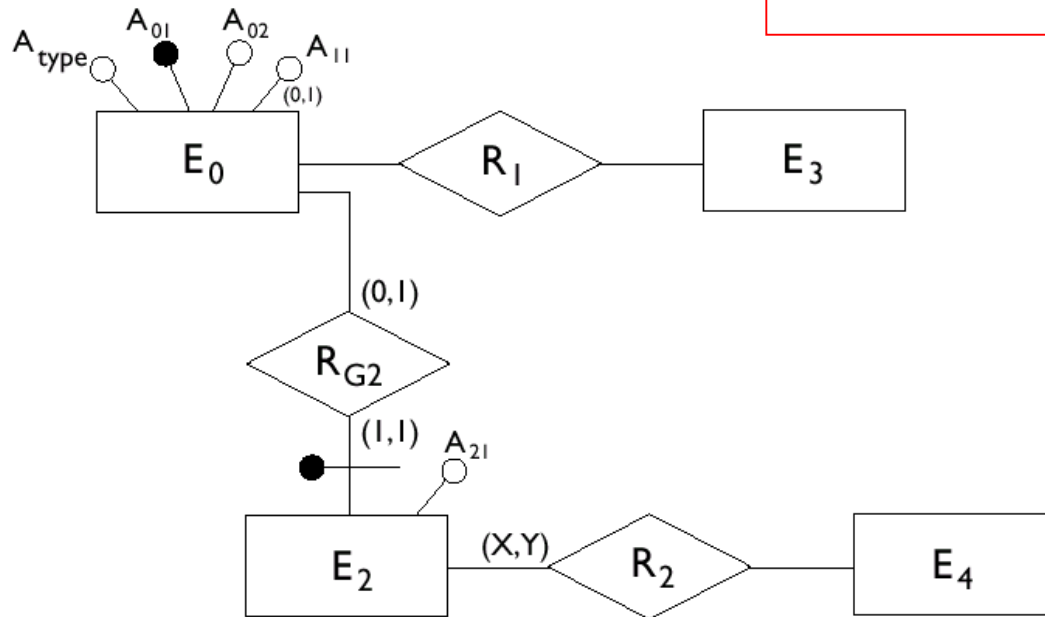
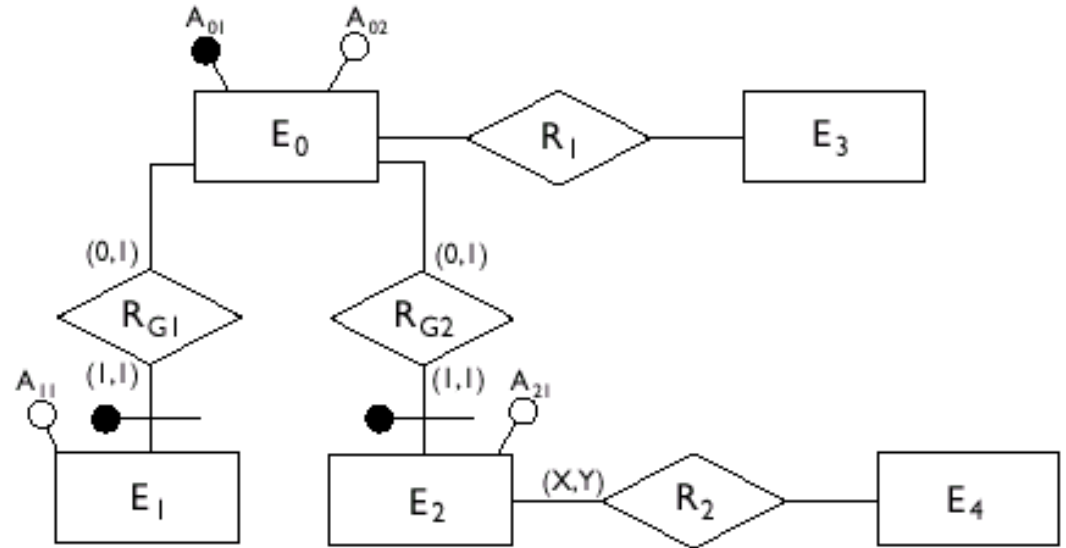
## Option 2



## Option 3

*...Two  
More...*

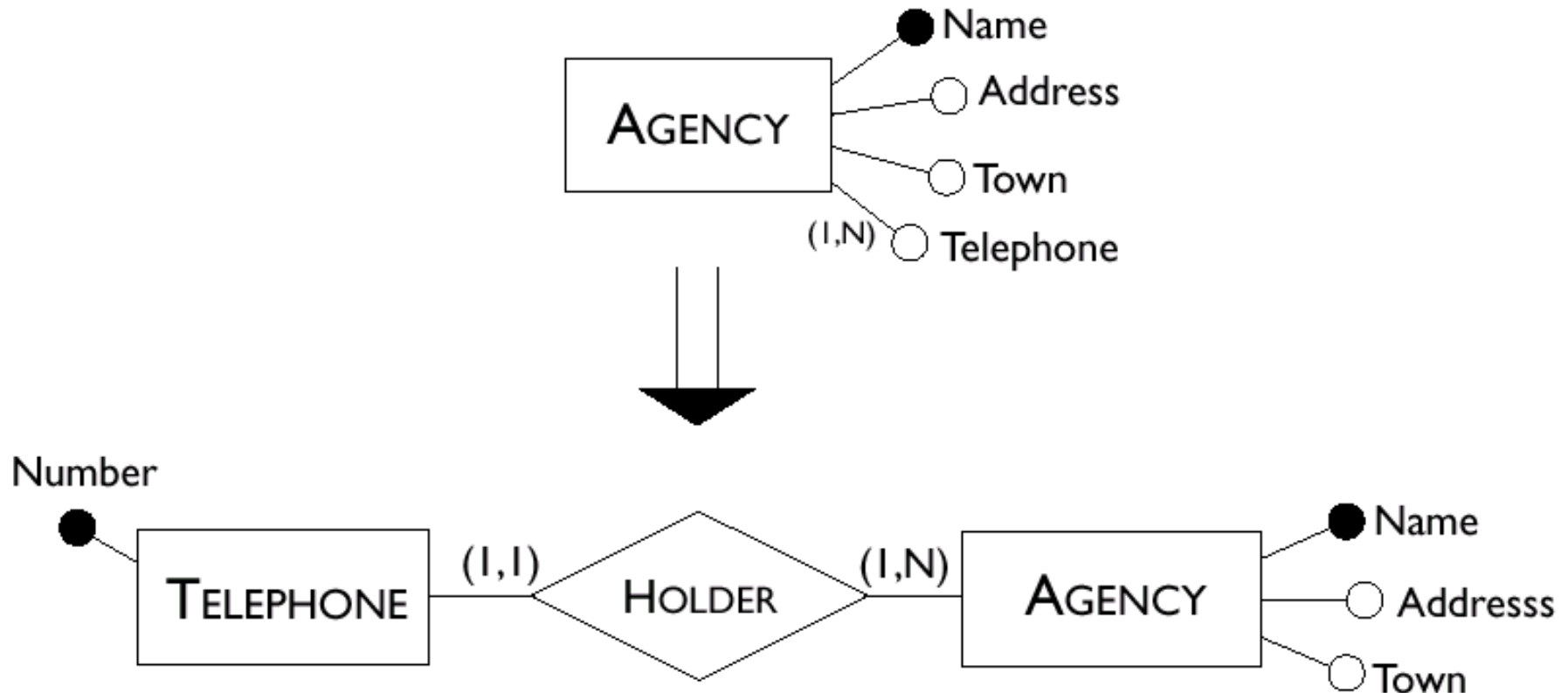
## Option 4 (combination)



# *General Rules For Removing Generalization*

- Option 1 is convenient when the operations involve the occurrences and the attributes of  $E_0$ ,  $E_1$  and  $E_2$  more or less in the same way.
- Option 2 is possible only if the generalization satisfies the coverage constraint (i.e., every instance of  $E_0$  is either an instance of  $E_1$  or  $E_2$ ) and is useful when there are operations that apply only to occurrences of  $E_1$  or  $E_2$ .
- Option 3 is useful when the generalization is not coverage-compliant and the operations refer to either occurrences and attributes of  $E_1$  ( $E_2$ ) or of  $E_0$ , and therefore make distinctions between child and parent entities.
- Available options can be combined (see option 4)

# *Elimination of Multi-Valued Attributes*



## *Selecting a Primary Key*

- Every relation must have a unique primary key.
- The criteria for this decision are as follows:
  - ✓ Attributes with null values cannot form primary keys;
  - ✓ One/few attributes is preferable to many attributes;
  - ✓ Internal keys preferable to external ones (weak entities depend for their existence on other entities);
  - ✓ A key that is used by many operations to access instances of an entity is preferable to others.
- At this stage, if none of the candidate keys satisfies the above requirements, it may be best to introduce a new attribute (e.g., social insurance #, student #,...)

# *Translation into a Logical Schema*

- Starting from an E-R schema, an equivalent relational schema is constructed. By “equivalent”, we mean a schema capable of representing the same information.
- We will deal with the translation problem systematically, beginning with the fundamental case, that of entities linked by many-to-many relationships.



# *Many-to-Many Relationships*

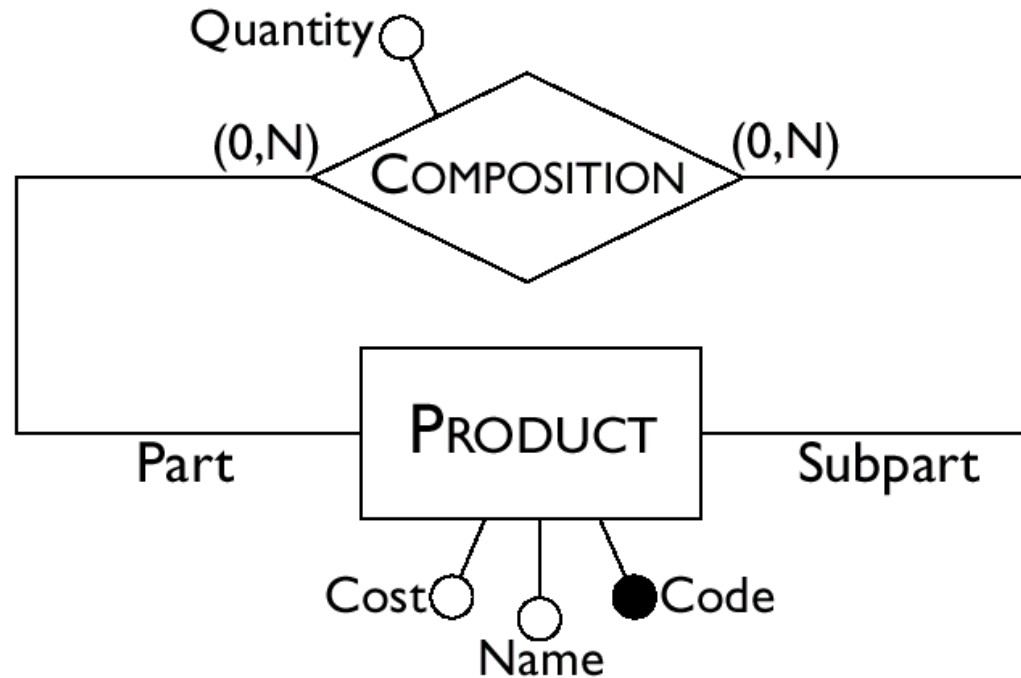


**Employee(Number, Surname, Salary)**

**Project(Code, Name, Budget)**

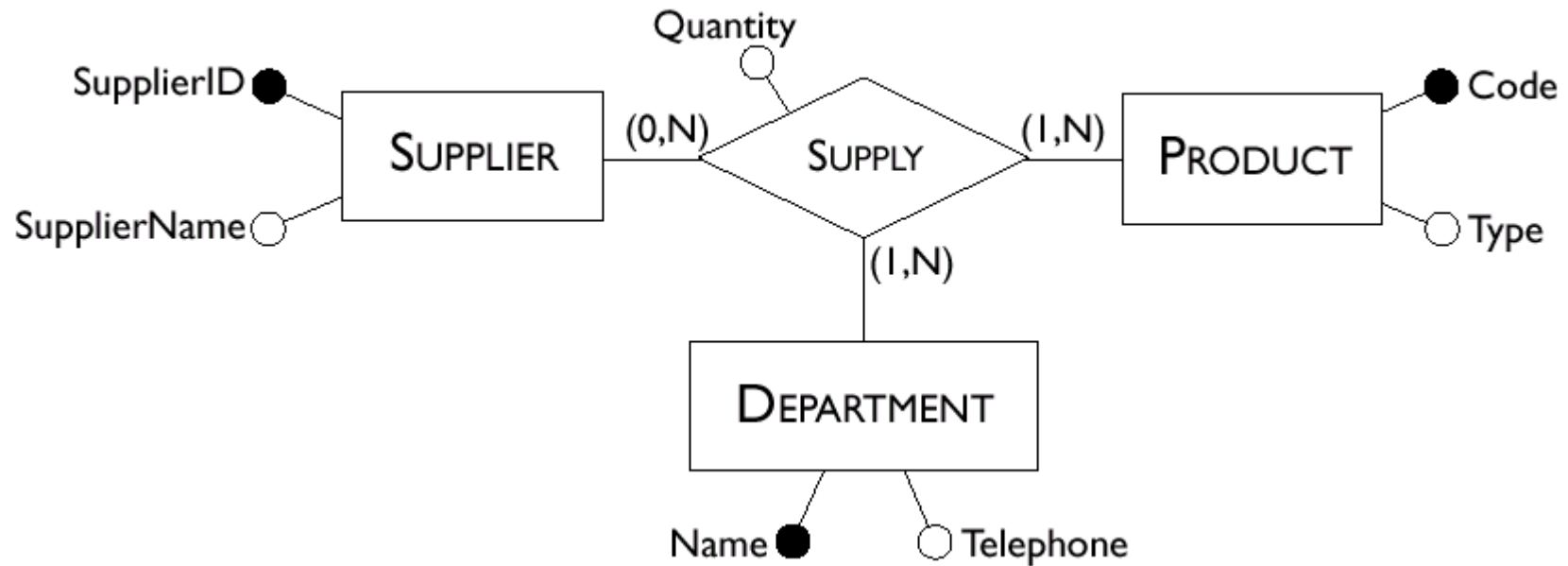
**Participation(Number, Code, StartDate)**

# *Many-to-Many Recursive Relationships*



**Product(Code, Name, Cost)**  
**Composition(Part, SubPart, Quantity)**

# *Ternary Relationships*



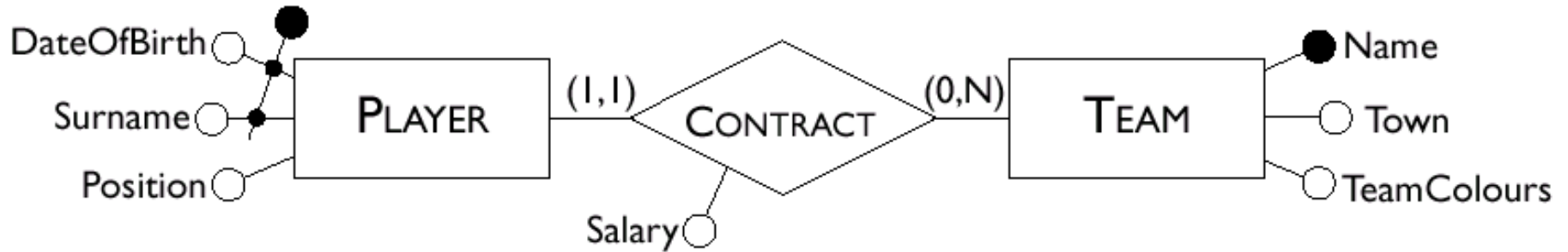
**Supplier(SupplierID, SupplierName)**

**Product(Code, Type)**

**Department(Name, Telephone)**

**Supply(Supplier, Product, Department, Quantity)**

# *One-to-Many Relationships*



**Player(Surname, DateOfBirth, Position)**

**Team(Name, Town, TeamColours)**

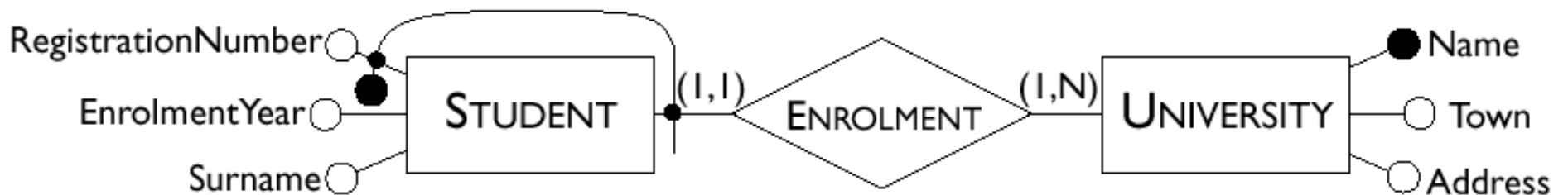
**Contract(PlayerSurname, PlayerDateOfBirth, Team, Salary)**

*Or*

**Player(Surname, DateOfBirth, Position, TeamName, Salary)**

**Team(Name, Town, TeamColours)**

# *Weak Entities*



**Student(RegistrationNumber, University, Surname,  
EnrolmentYear)**  
**University(Name, Town, Address)**

# *One-to-One Relationships*



Head(Number, Name, Salary, Department, StartDate)

Department(Name, Telephone, Branch)

*Or*

Head(Number, Name, Salary, StartDate)

Department(Name, Telephone, HeadNumber, Branch)

# *Optional One-to-One Relationships*

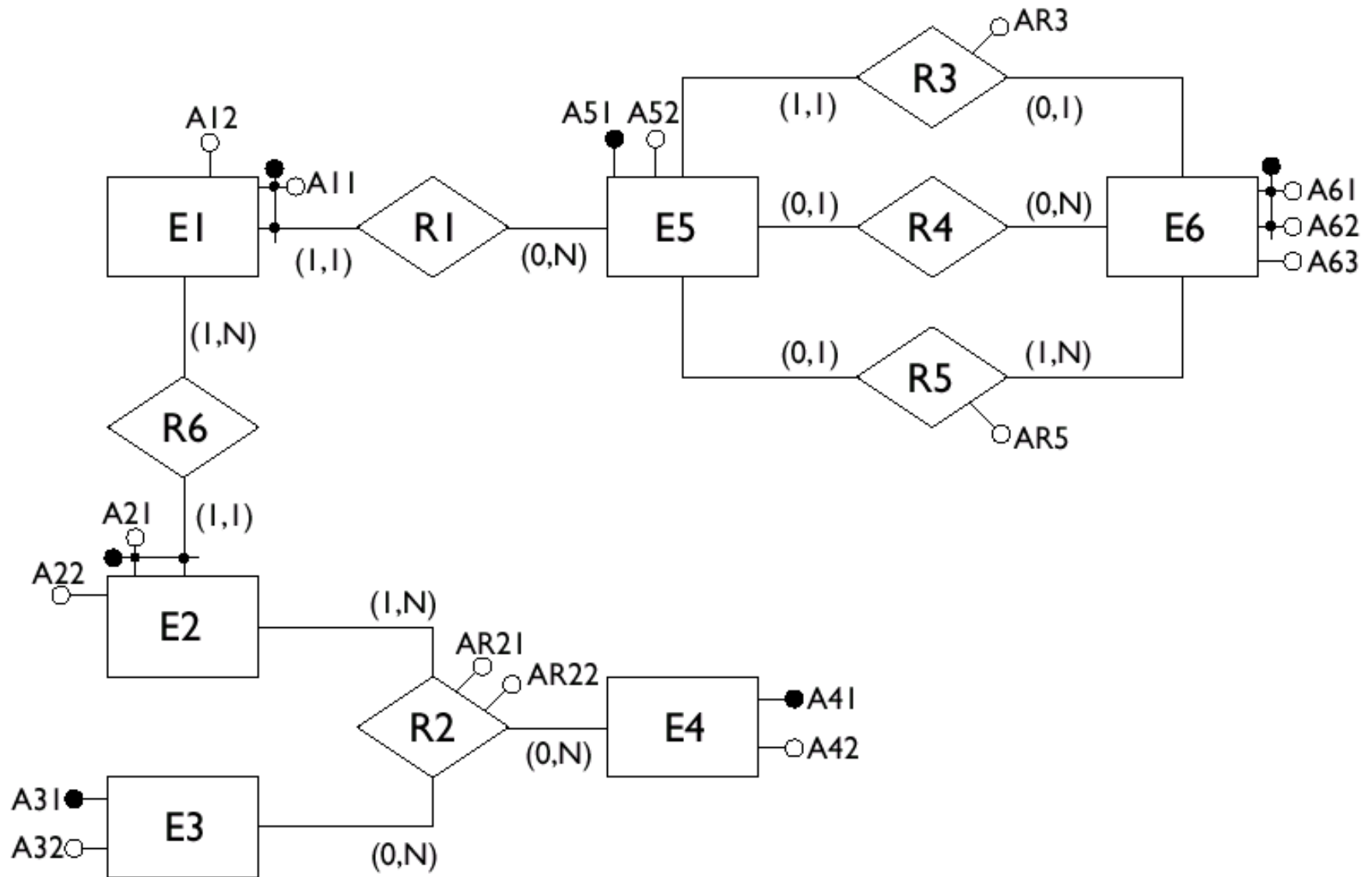


Employee(Number, Name, Salary)  
Department(Name, Telephone, Branch, Head,  
StartDate)

*Or, if both entities are optional*

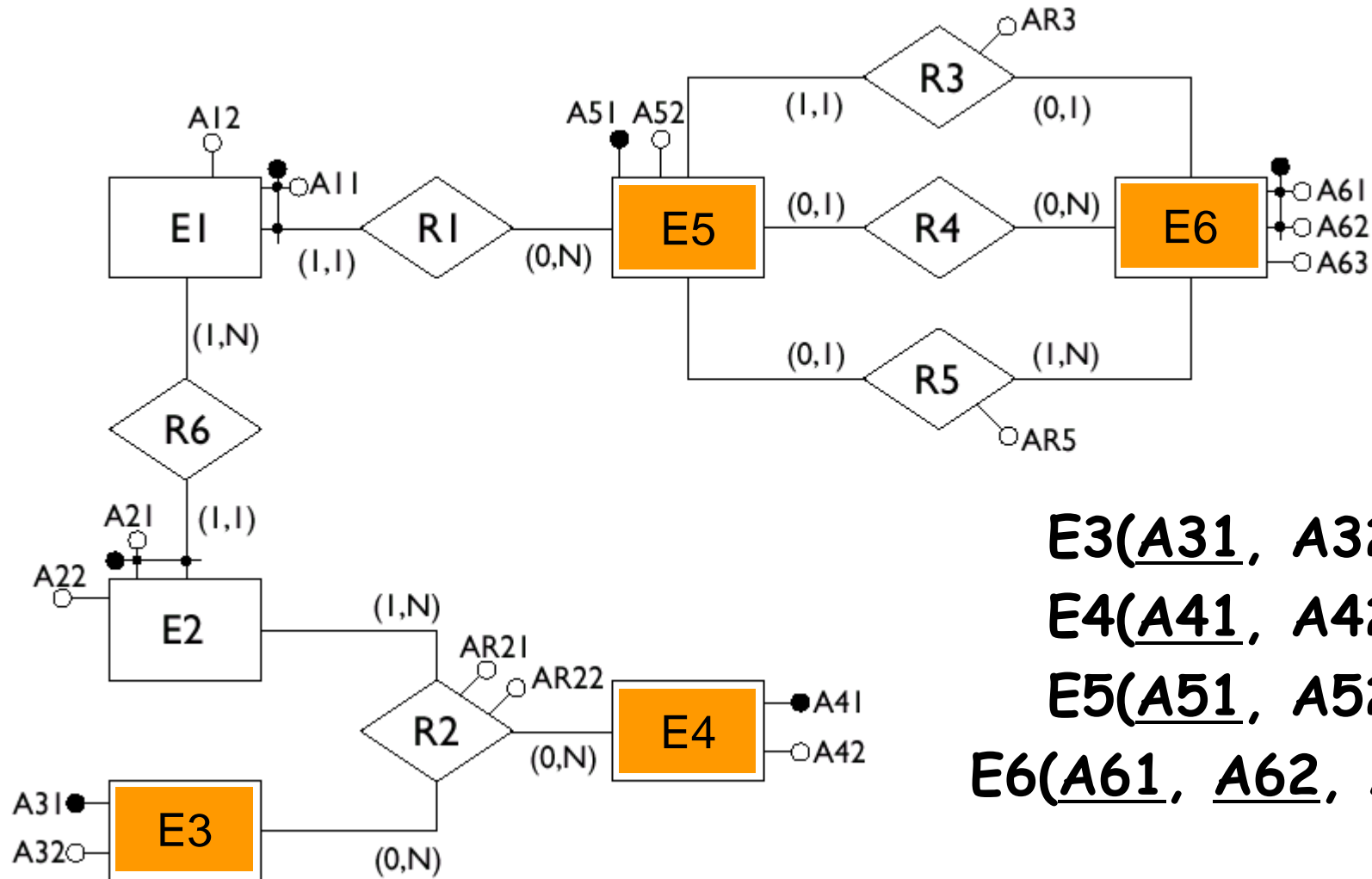
Employee(Number, Name, Salary)  
Department(Name, Telephone, Branch)  
Management(Head, Department, StartDate)

# *A Sample ER Schema*





# *Entities with Internal Identifiers*



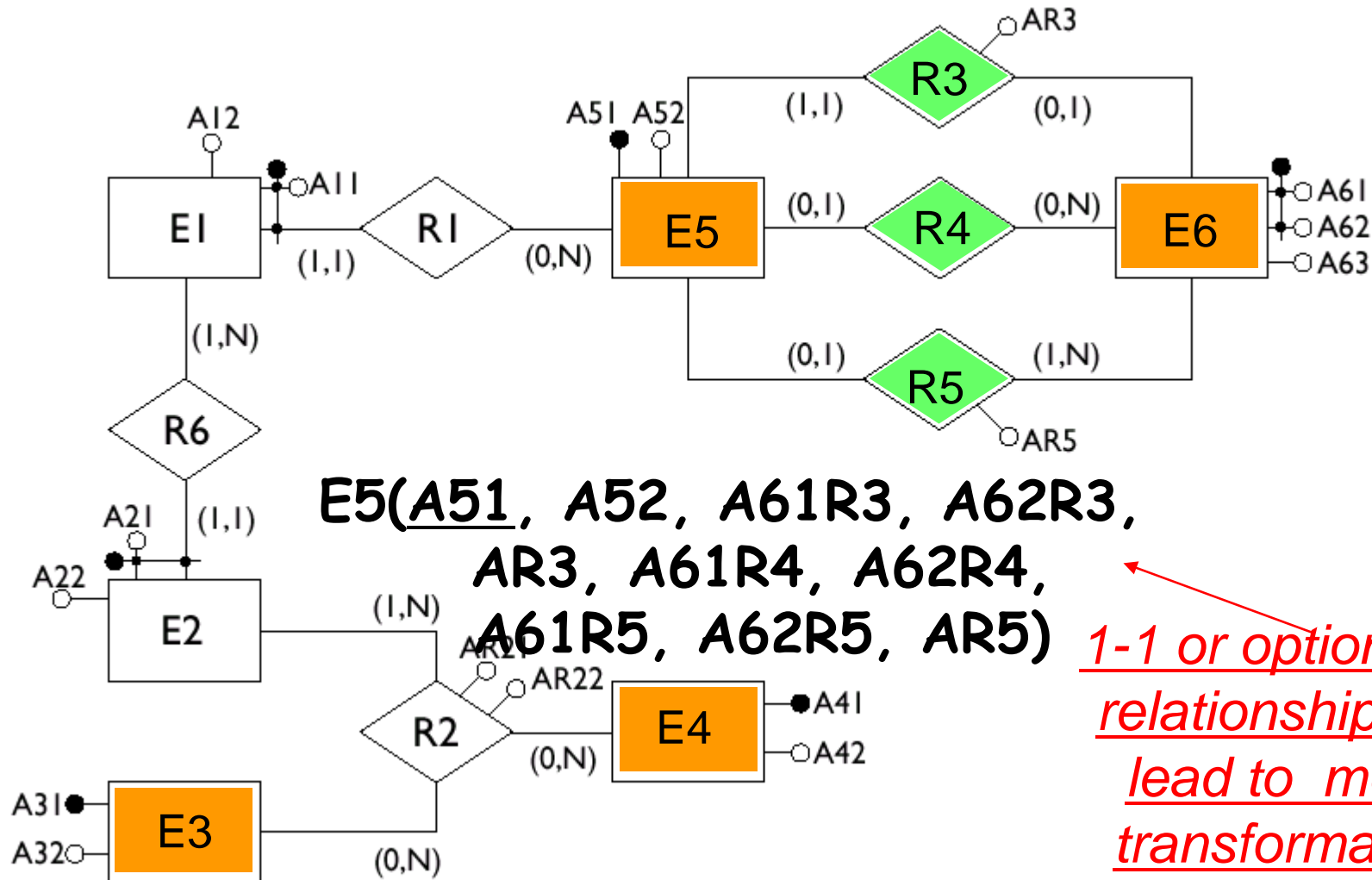
**E3(A31, A32)**

**E4(A41, A42)**

**E5(A51, A52)**

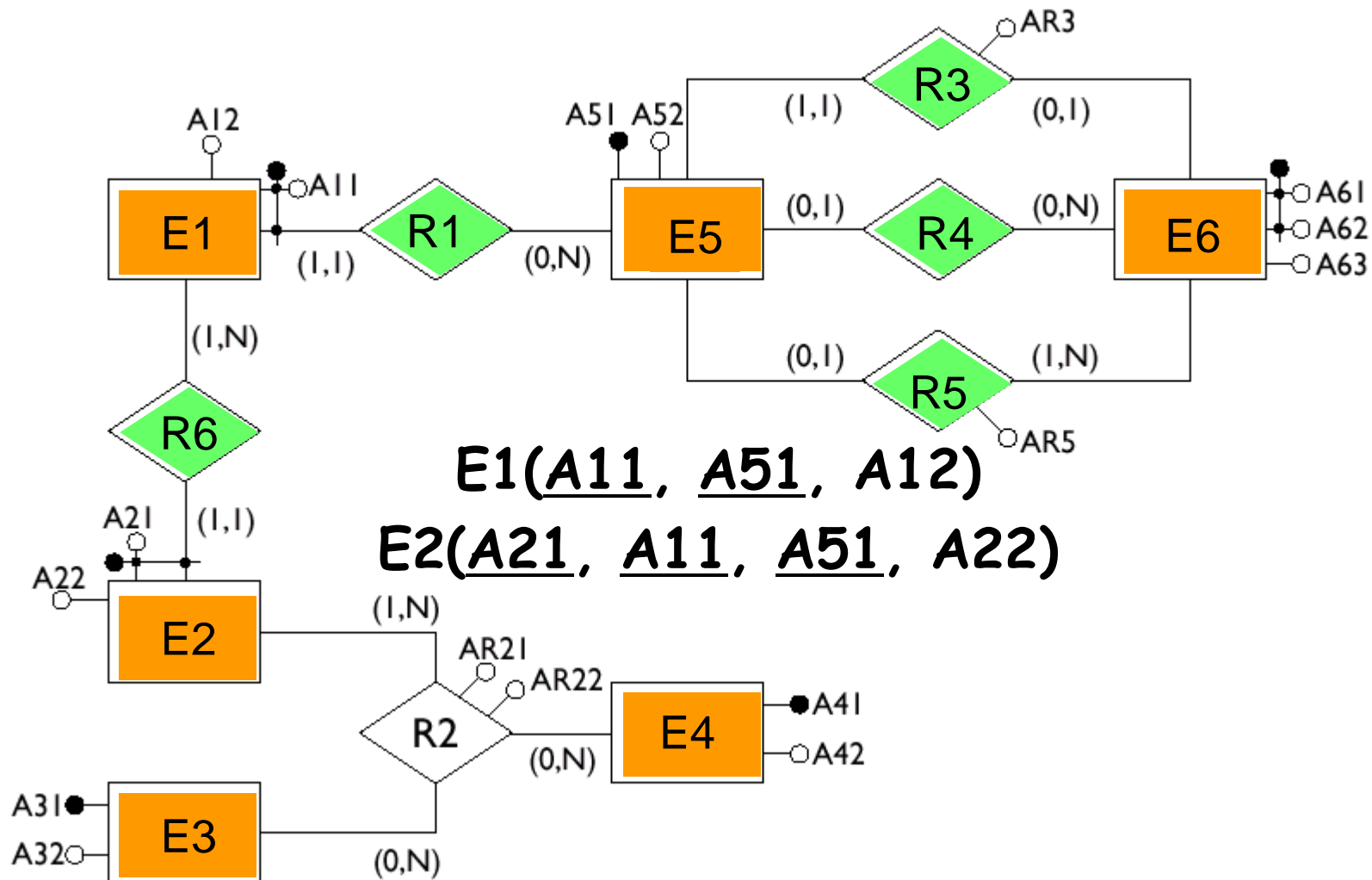
**E6(A61, A62, A63)**

# *1-1 and Optional 1-1 Relationships*

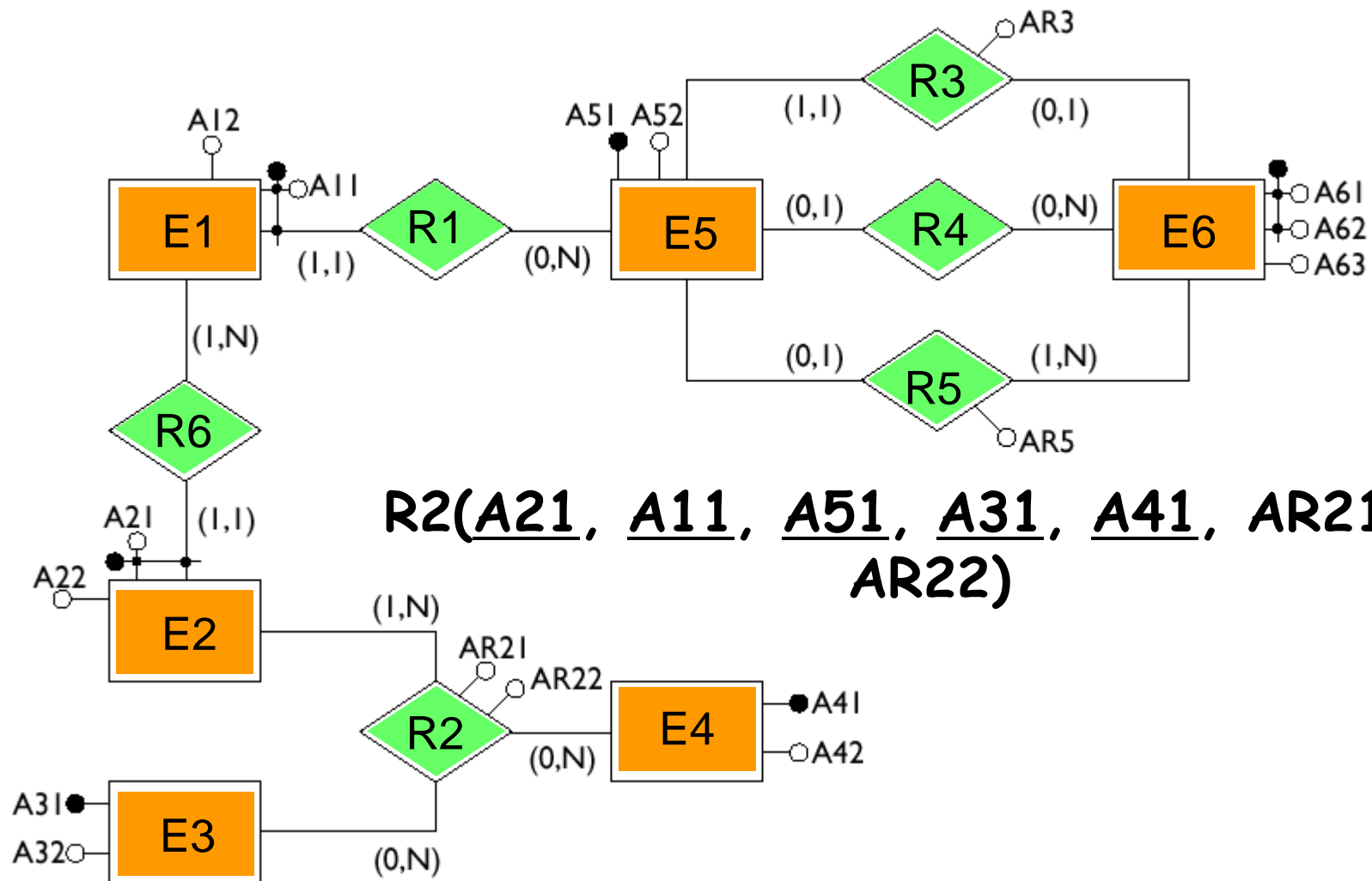


*1-1 or optional 1-1 relationships can lead to messy transformations*

# Weak Entities



# Many-to-Many Relationships



## *Result of the Translation*

E1(A11, A51, A12)

E2(A21, A11, A51, A22)

E3(A31, A32)

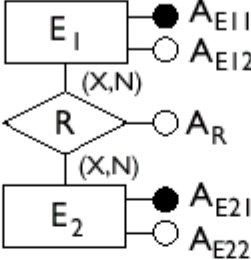
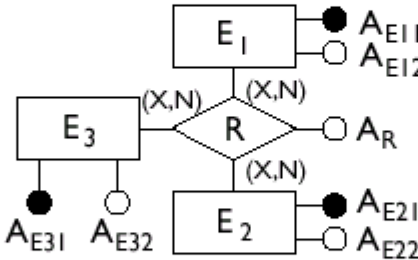
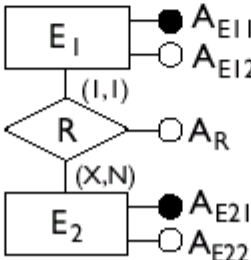
E4(A41, A42)

E5(A51, A52, A61R3, A62R3, AR3, A61R4, A62R4,  
A61R5, A62R5, AR5)

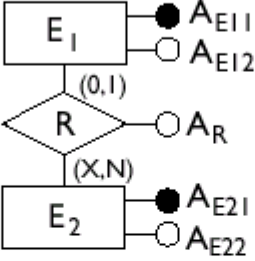
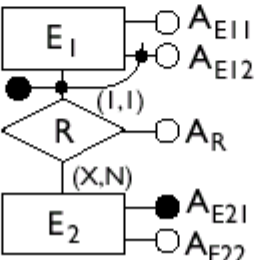
E6(A61, A62, A63)

R2(A21, A11, A51, A31, A41, AR21, AR22)

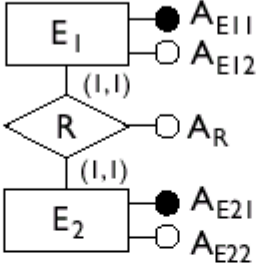
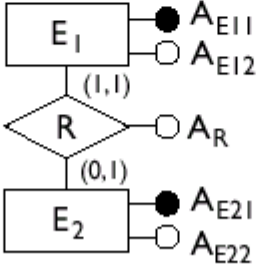
# Summary of Transformation Rules

Type	Initial schema	Possible translation
Binary many-to-many relationship		$E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, A_R)$
Ternary many-to-many relationship		$E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $E_3(\underline{A_{E31}}, A_{E32})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, \underline{A_{E31}}, A_R)$
One-to-many relationship with mandatory participation		$E_1(\underline{A_{E11}}, A_{E12}, A_{E21}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$

## ...More Rules...

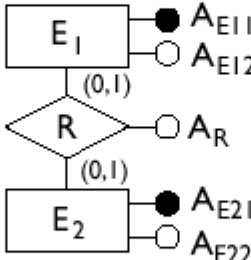
Type	Initial schema	Possible translation
One-to-many relationship with optional participation		$E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, A_R)$ <p>Alternatively:</p> $E_1(\underline{A_{E11}}, A_{E21}, A_{E21}^*, A_R^*)$ $E_2(\underline{A_{E21}}, A_{E22})$
Relationship with external identifiers		$E_1(\underline{A_{E12}}, \underline{A_{E21}}, A_{E11}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$

## *...Even More Rules...*

Type	Initial schema	Possible translation
One-to-one relationship with mandatory participation for both entities		$E_1(\underline{A_{E11}}, A_{E12}, \underline{A_{E21}}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$ <p>Alternatively:</p> $E_2(\underline{A_{E21}}, A_{E22}, \underline{A_{E11}}, A_R)$ $E_1(\underline{A_{E11}}, A_{E12})$
One-to-one relationship with optional participation for one entity		$E_1(\underline{A_{E11}}, A_{E12}, \underline{A_{E21}}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$



## *...and the Last One...*

Type	Initial schema	Possible translation
One-to-one relationship with optional participation for both entities		$\underline{E_1(A_{E11}, A_{E21})}$ $E_2(\underline{A_{E21}}, A_{E22}, A_{E11}^*, A_R^*)$ <p>Alternatively:</p> $\underline{E_1(A_{E11}, A_{E12}, A_{E21}^*, A_R^*)}$ $E_2(\underline{A_{E21}}, A_{E22})$ <p>Alternatively:</p> $\underline{E_1(A_{E11}, A_{E12})}$ $\underline{E_2(A_{E21}, A_{E22})}$ $R(\underline{A_{E11}}, \underline{A_{E21}}, A_R)$