

# The Entity/Relationship (E/R) Model & DB Design

Introduction to databases

CSC343, Fall 2017

Diane Horton, Michelle Craig, and Sina Meraji

Based on slides by Manos Papagelis

Thanks to Ryan Johnson, John Mylopoulos, Arnold Rosenbloom  
and Renee Miller for material in these slides

# Overview

- Using the Entity/Relationship (ER) Model to model the real world
- From there, designing a database schema
  - Restructuring of an E/R model
  - Translating an E/R model into a logical model (DB Schema)

# THE ENTITY/RELATIONSHIP (E/R) MODEL

# Conceptualizing the real-world

- DB design begins with a boss or client who wants a database.
- We must **map** the entities and relationships of the world into the concepts of a database. This is called *modeling*.
- Sketching the key components is an efficient way to develop a design.
  - Sketch out (and debug) schema designs
  - Express as many constraints as possible
  - Convert to relational DB once the client is happy

# Entity/Relationship Model

- Visual data model (diagram-based)
  - Quickly “chart out” a database design
  - Easier to “see” big picture
  - Comparable to class diagrams in UML
- Basic concepts:
  - *entities*
  - *relationships* among them
  - *attributes* describing the entities and relationships

# Entity Sets

- An **entity set** represents a *category of objects* that have properties in common and an autonomous existence (e.g., City, Department, Employee, Sale)
- An **entity** is an *instance of an entity set* (e.g., Stockholm is a City; Peterson is an Employee)

EMPLOYEE

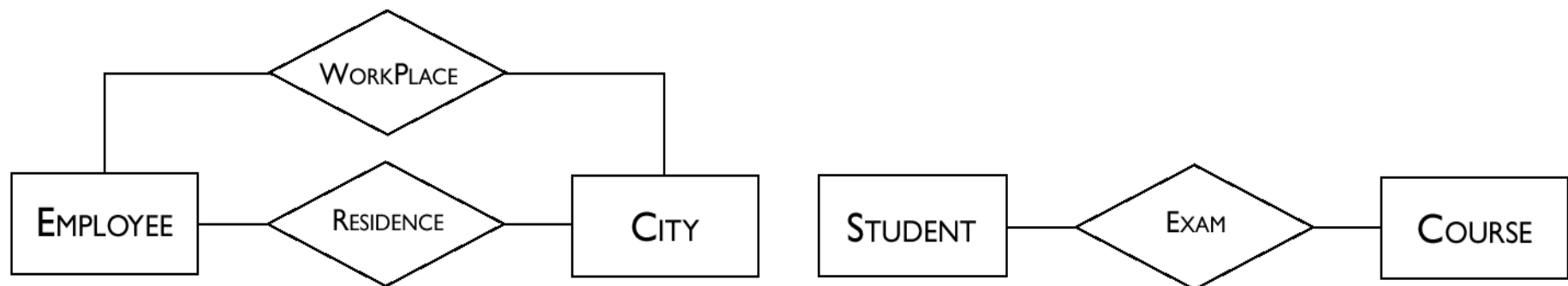
DEPARTMENT

CITY

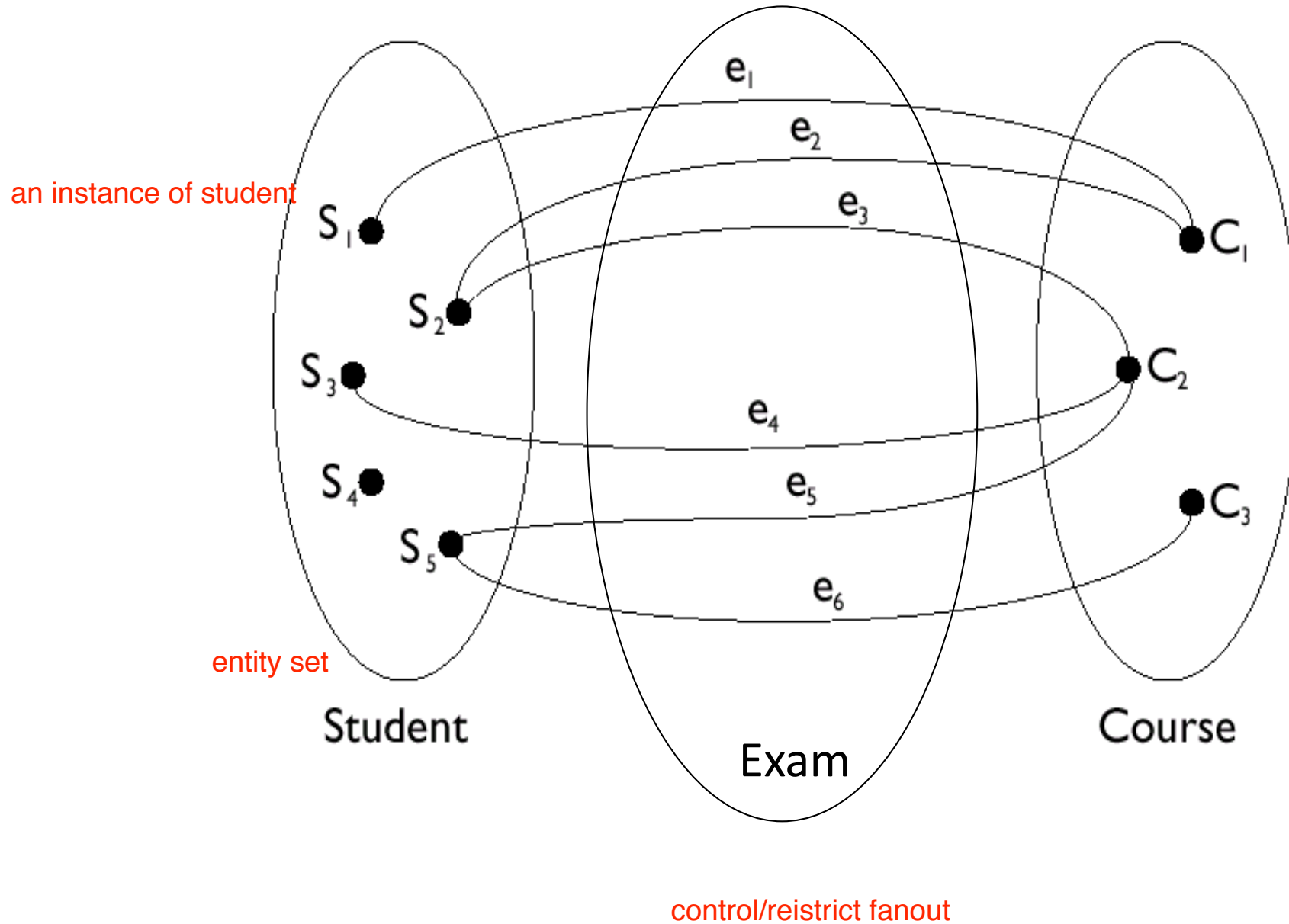
SALE

# Relationship Sets diamonds

- A relationship set is an association between 2+ entity sets (e.g., Residence is a relationship set between entity sets City and Employee)
- A relationship is an instance of a n-ary relationship set (e.g., the pair <Johanssen, Stockholm> is an instance of relationship Residence)



# Example of Instances for Exam

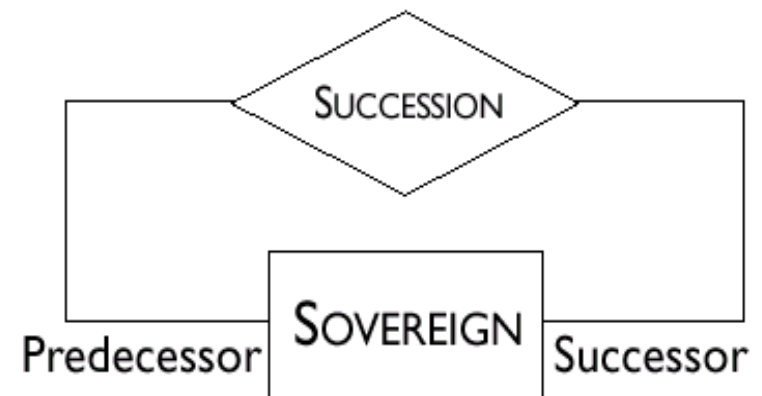
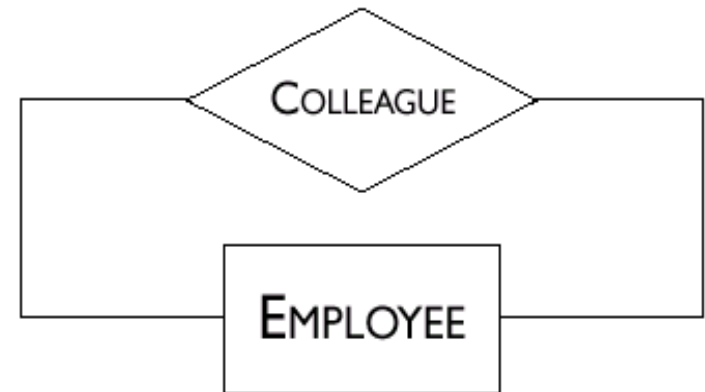




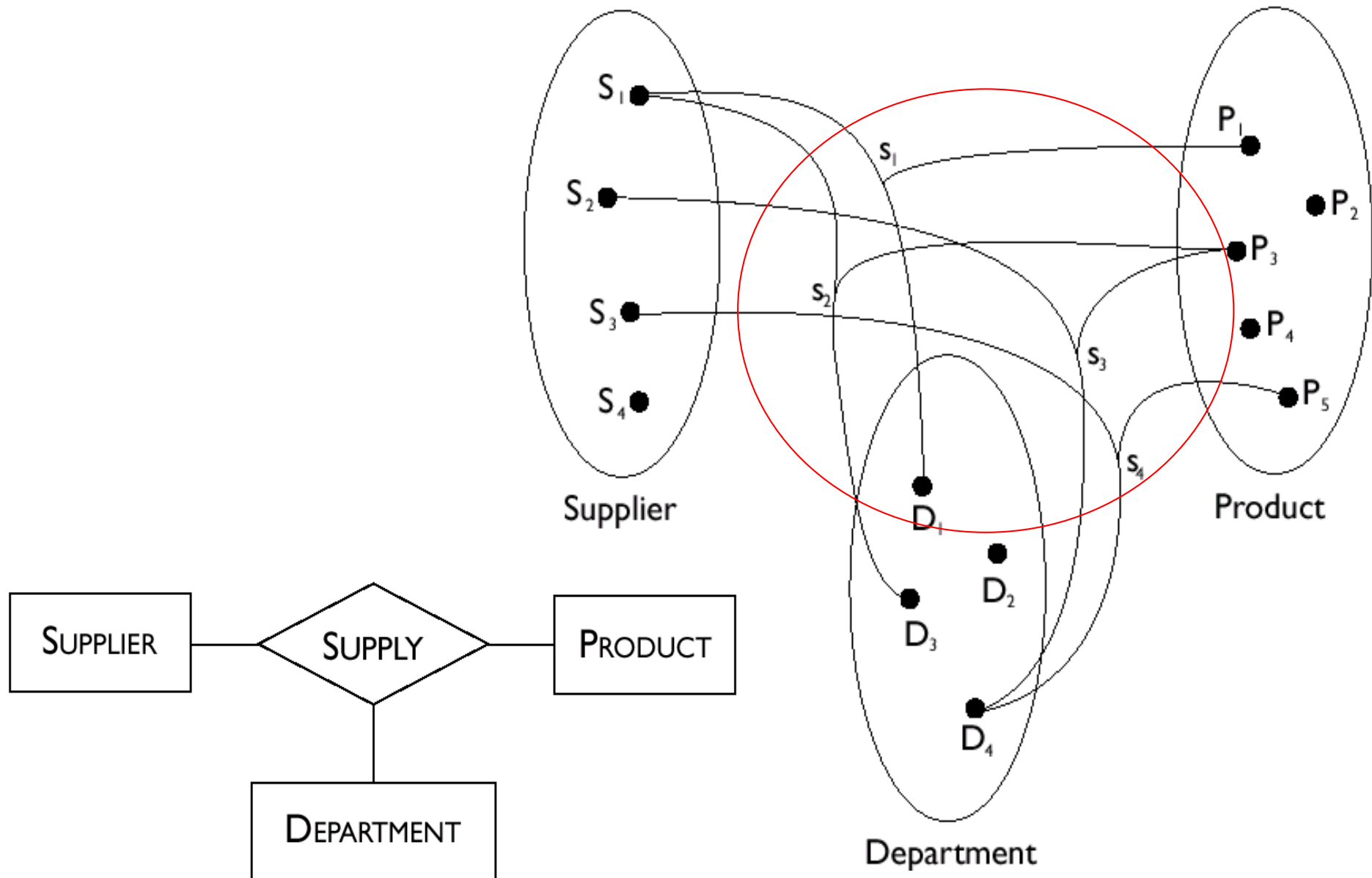
# Recursive Relationships

- Recursive relationships relate an entity set to itself
- The relationship may be asymmetric
  - If so, we indicate the two **roles** that the entity plays in the relationship

i.e. colleagues in a single entity set employee

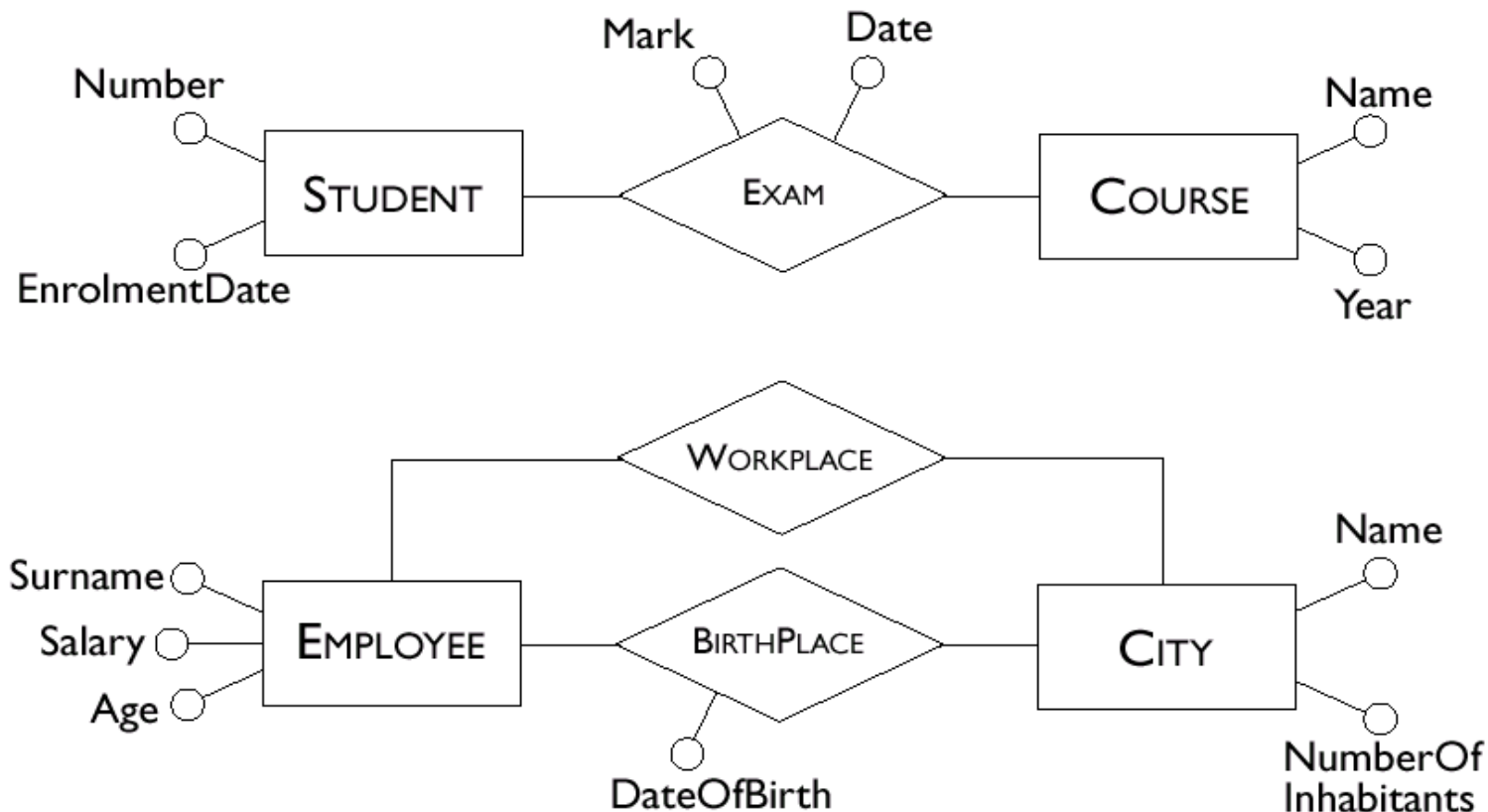


# Ternary Relationships



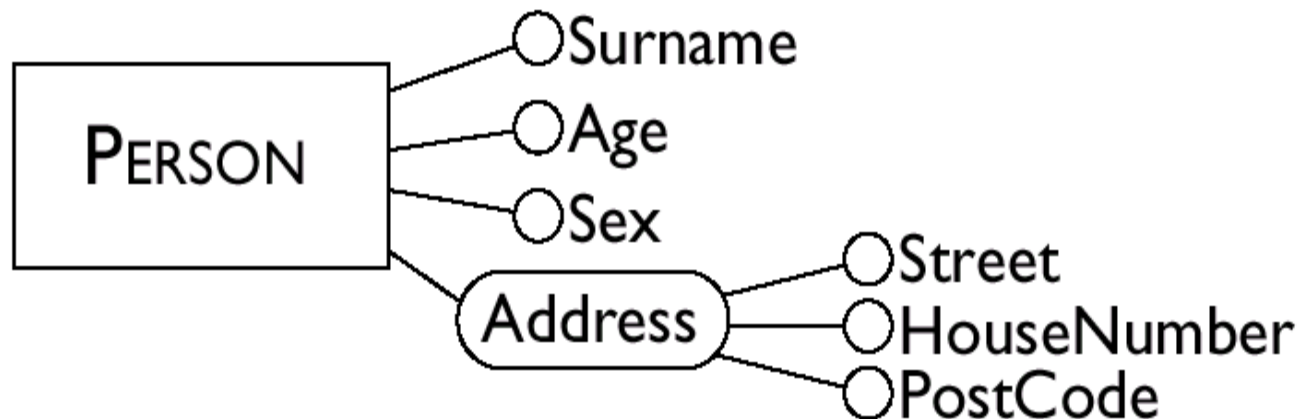
# Attributes small circle

- Describe elementary properties of entities or relationships (e.g., Surname, Salary and Age are attributes of Employee)
- May be **single-valued**, or **multi-valued**

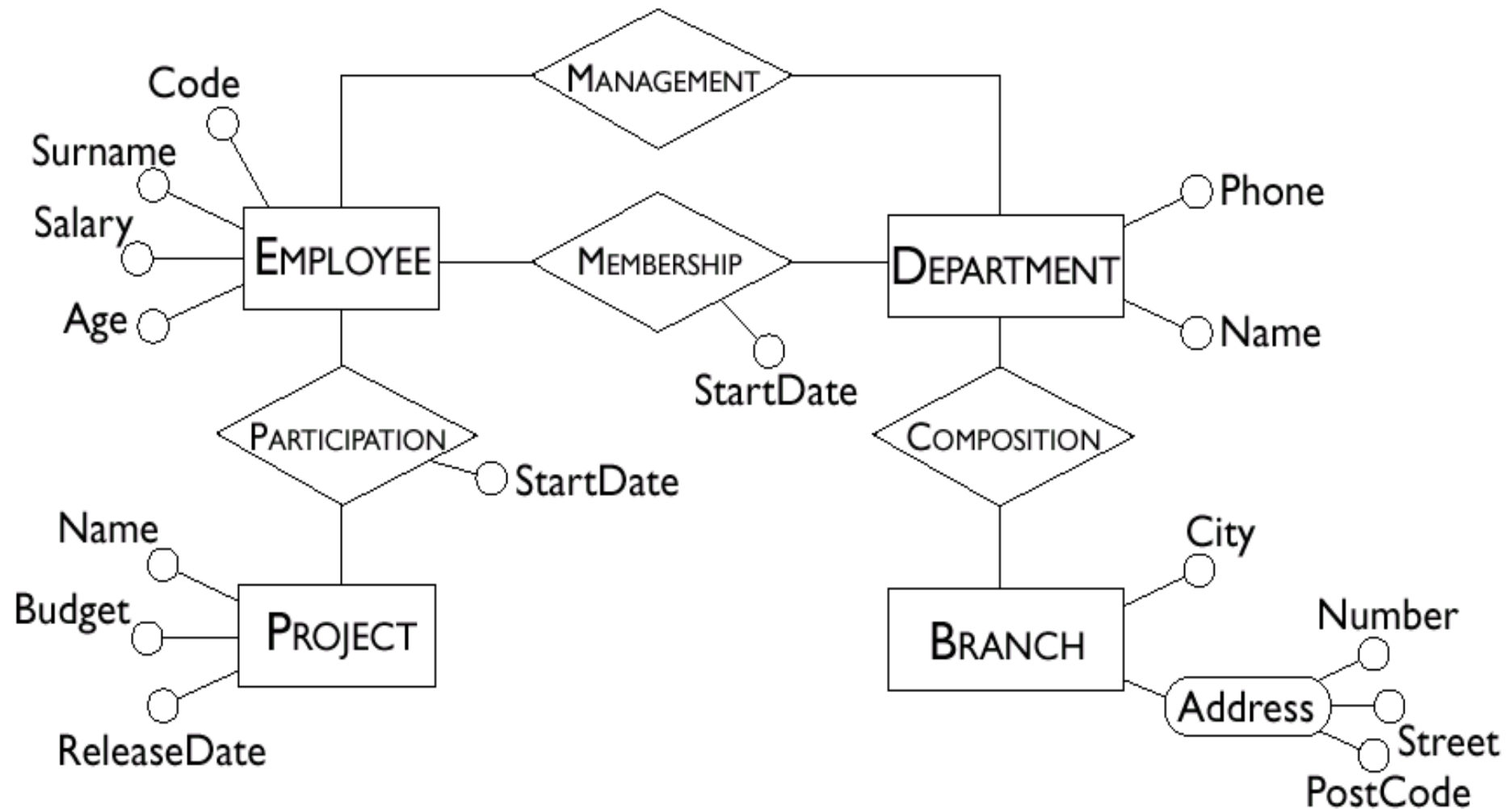


# Composite Attributes

- **composite attributes** are grouped attributes of the same entity or relationship that have closely connected meaning or uses

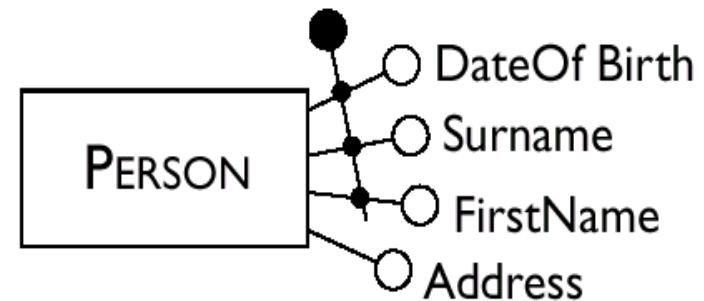
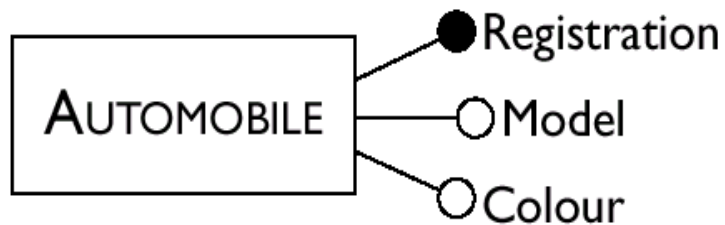


# Example Schema with Attributes



# Keys in E/R

- Notation: solid circle
- If multi-attribute, connect with a line and a “knob”



# Cardinalities

- Each entity set participates in a relationship set with a minimum (**min**) and a maximum (**max**) cardinality
- Cardinalities **constrain** how entity instances participate in relationship instances
- **Notation**: pairs of (**min**, **max**) values for each entity set



employee has at least 1 at most 5 tasks

has task with 0-50 person assigned

An entity might not participate in any relationship





# Cardinalities (cont.)

- In principle, cardinalities are pairs of non-negative integers **(min, max)** such that **min ≤ max**.
- minimum cardinality **min**:
  - If **0**, entity participation in a relationship is **optional**
  - If **1**, entity participation in a relationship is **mandatory**
  - Other values are possible
- maximum cardinality **max**:
  - If **1**, each instance of the entity is associated **at most with a single** instance of the relationship
  - If **> 1**, then each instance of the entity can be associated **multiple** instances of the relationship
  - We write **N** to indicate **no upper limit**
  - Other values are possible

# Cardinality Examples



each person must have a residence

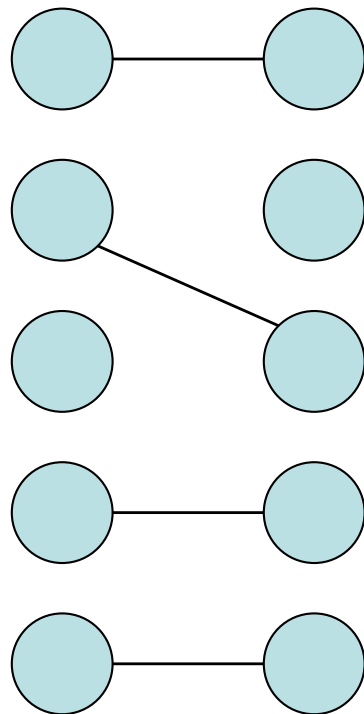




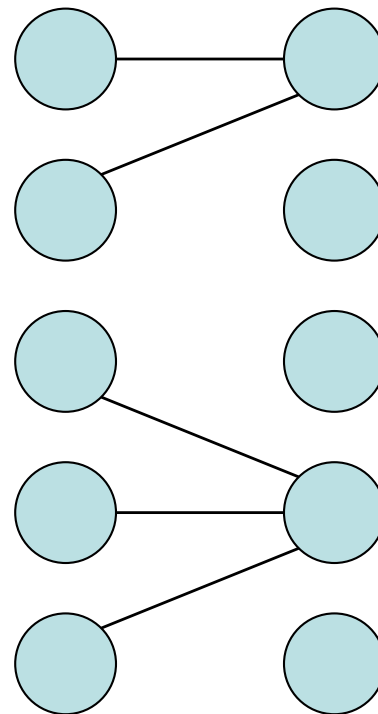


# Multiplicity of relationships

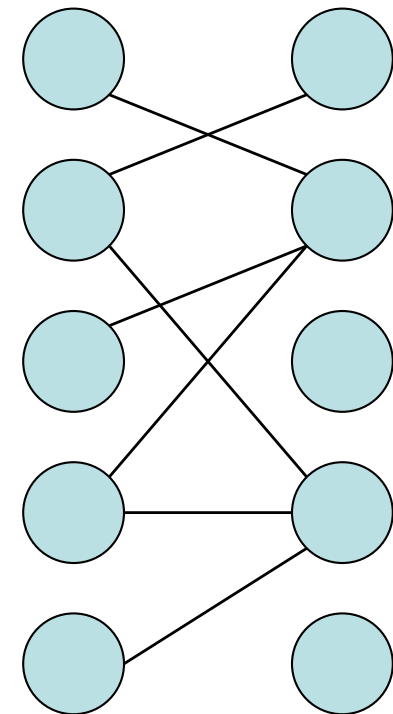
If entities **E1** and **E2** participate in relationship **R** with cardinalities **(n1, N1)** and **(n2, N2)** then the multiplicity of **R** is **N1-to-N2** (which is the same as saying **N2-to-N1**)



1-to-1



N-to-1 OR 1-to-N

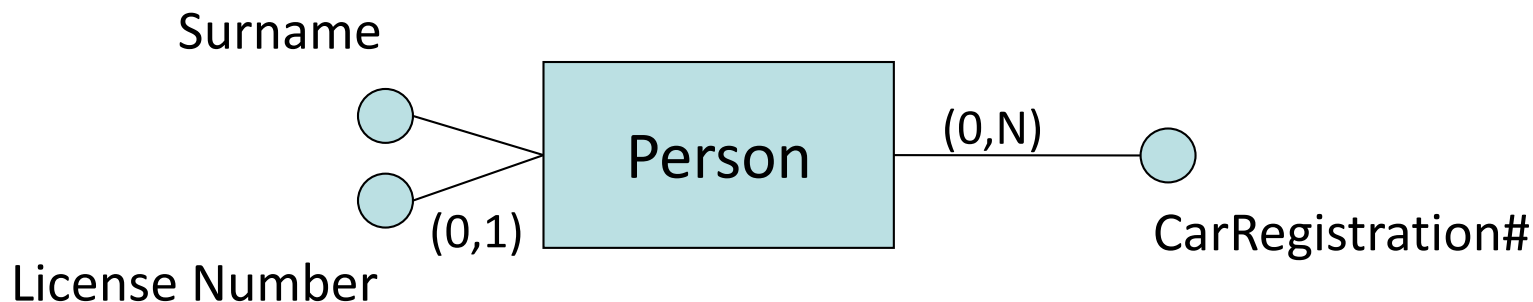


N-to-N

max has to be 1, says nothing about the min, could be (0,1)-(0,1) (1,1)-(0,1) (0,1)-(1,1) (1,1)-(1,1)

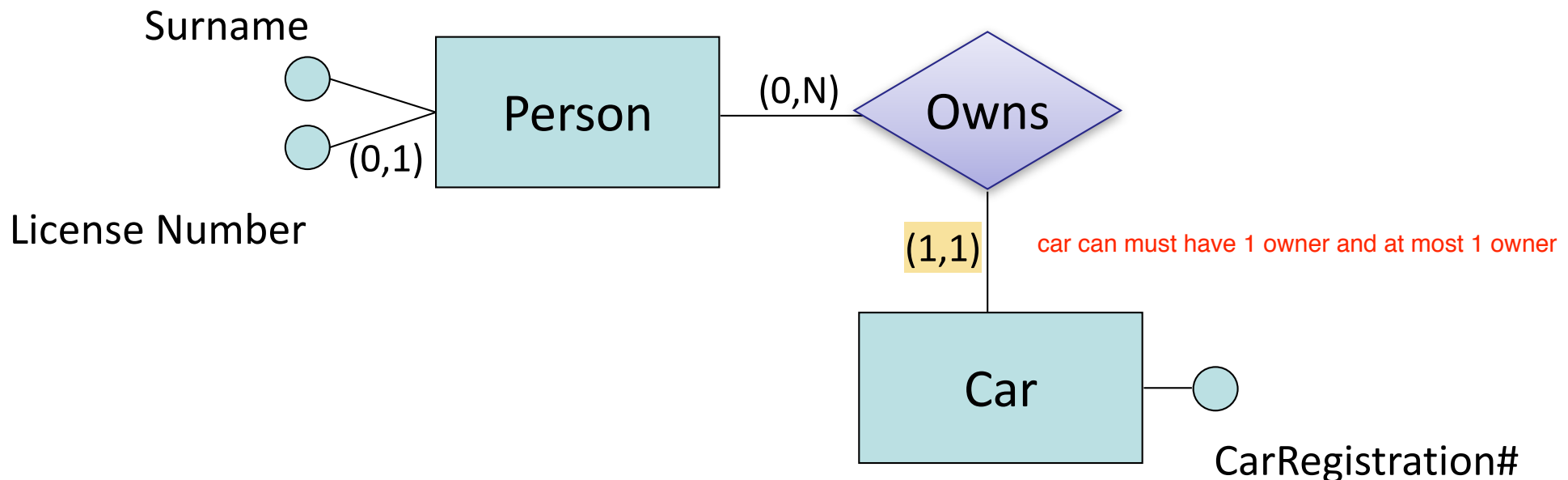
# Cardinalities of Attributes

- 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) default
- The value of an attribute may also be null, or have several values (multi-valued attributes)



# Cardinalities of Attributes (cont.)

- Multi-valued attributes often represent situations that can be modeled with additional entities. E.g., the ER schema of the previous slide can be revised into:



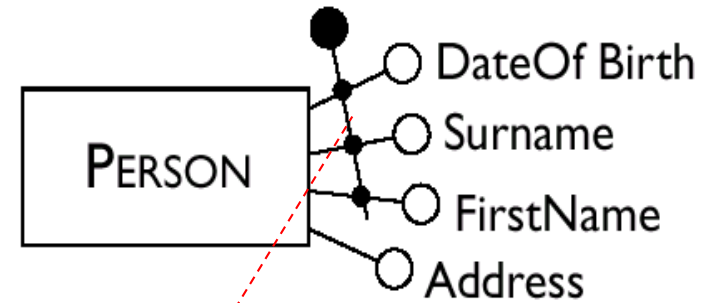
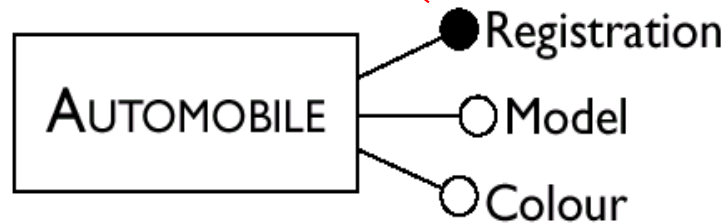
# Keys in E/R

- **Keys** consist of minimal sets of attributes which uniquely identify instances of an entity set.
- Usually, a key is formed by one or more attributes of the entity itself (*internal* keys).
- Sometimes, an entity doesn't have a key among its attributes. This is called a *weak entity*.  
Solution: the keys of related entities brought in to help with identification (becoming *foreign keys*).
- A key for a relationship consists of the keys of the entities it relates



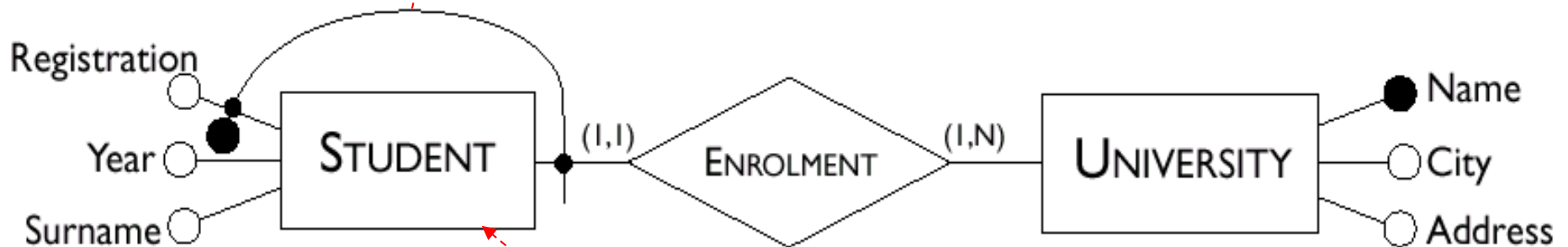
# Examples of Keys in E/R

*internal, single-attribute*



*internal, multi-attribute*

*foreign, multi-attribute*



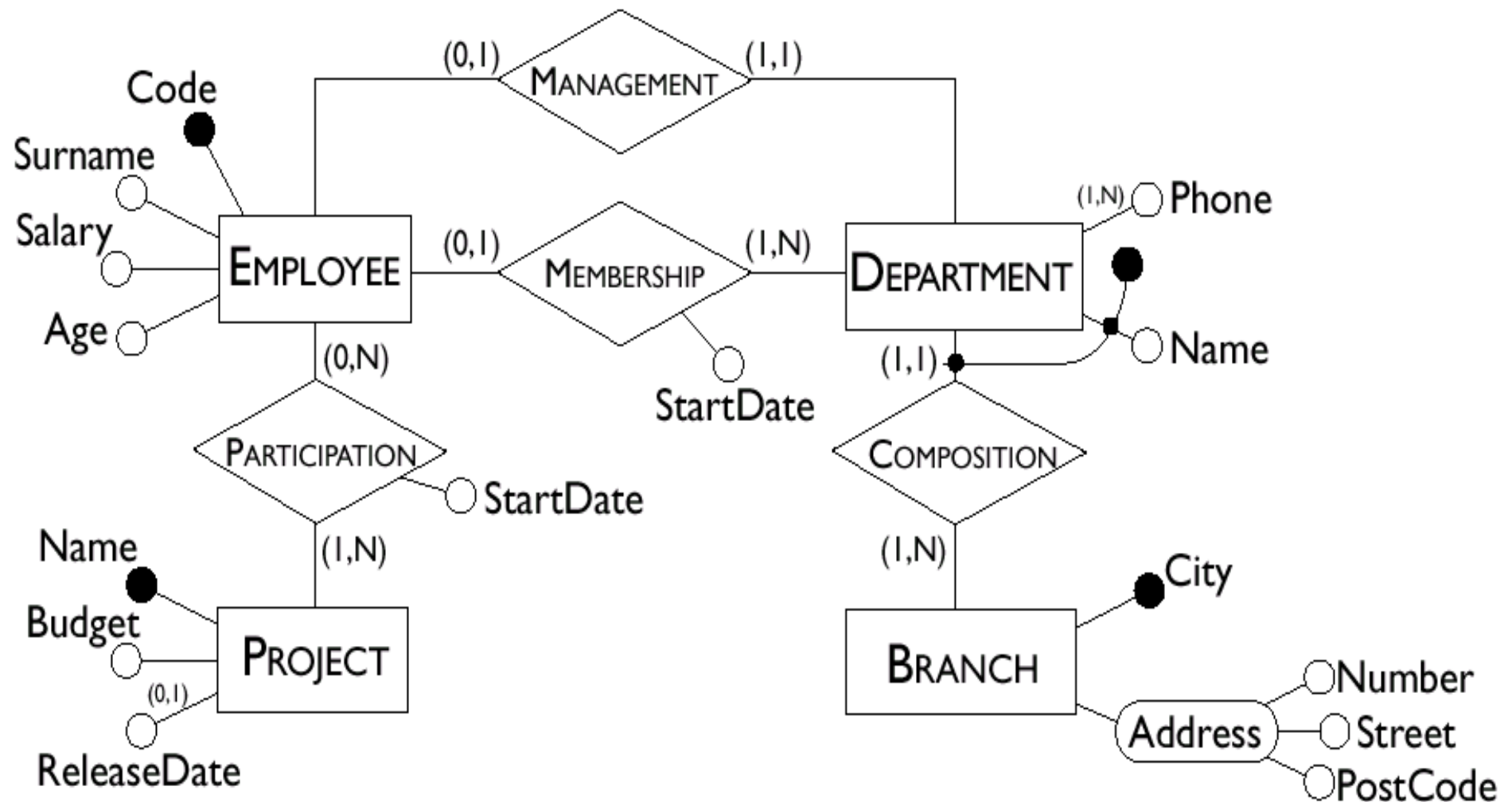
*Weak entity*

university should be weak entity too.. since name not unique, we can add a globally unique universityID

# General Observations about Keys

- A key may consist of one or more attributes, provided that each of the **attributes** has (1,1) cardinality
- A foreign 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 in the **relationship** with cardinality equal to (1,1)
- A foreign key may involve an entity that has itself a foreign key, as long as cycles are not generated
- Each entity set must have at least one (internal or foreign) key

# Schema with Keys



# Challenge: modeling the “real world”

- Life is arbitrarily complex
  - Directors who are also actors? Actors who play multiple roles in one movie? Animal actors?
- **Design choices**: Should a concept be modeled as an entity, an attribute, or a relationship?
- **Limitations** of the ER Model: A lot of data semantics can be captured but some cannot
- Key to successful model: **parsimony**
  - As complex as necessary, but no more
  - Choose to represent only “relevant” things

# EXAMPLE

# From real world to E/R Model

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.

# From real world to E/R Model

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

Term	Description	Synonym	Links
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

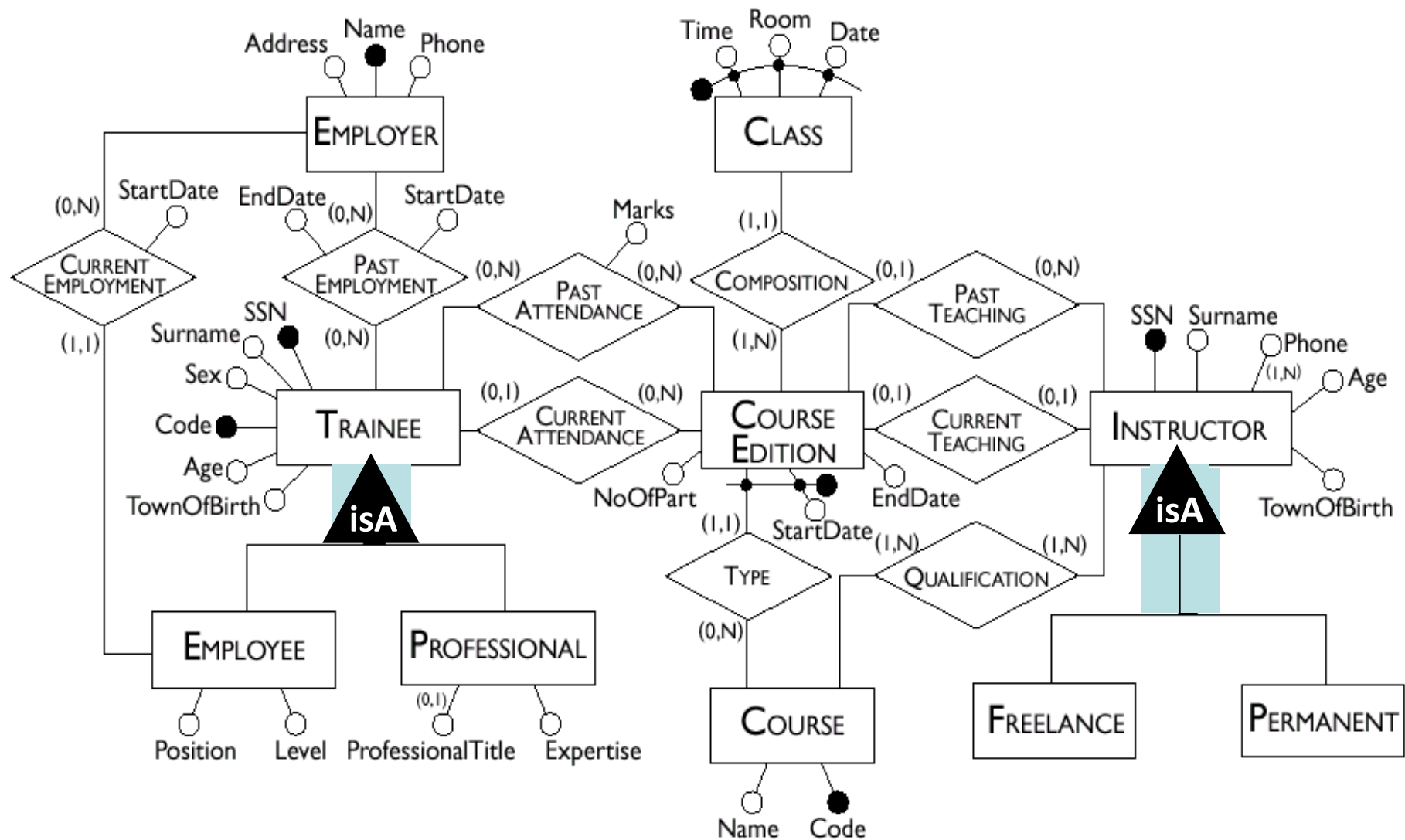


# More Annotations

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), 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)*, courses attended (there are about 200 courses) and the final assessment for each course. We need also to represent *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.

# ... the E/R model result



# FROM E/R MODEL TO DATABASE SCHEMA

# Two Steps

- *Restructure* the ER schema to improve it, based on criteria
- *Translate* the schema into the relational model

# **1. RESTRUCTURING AN E/R MODEL**

# Restructuring Overview

**Input:** E/R Schema

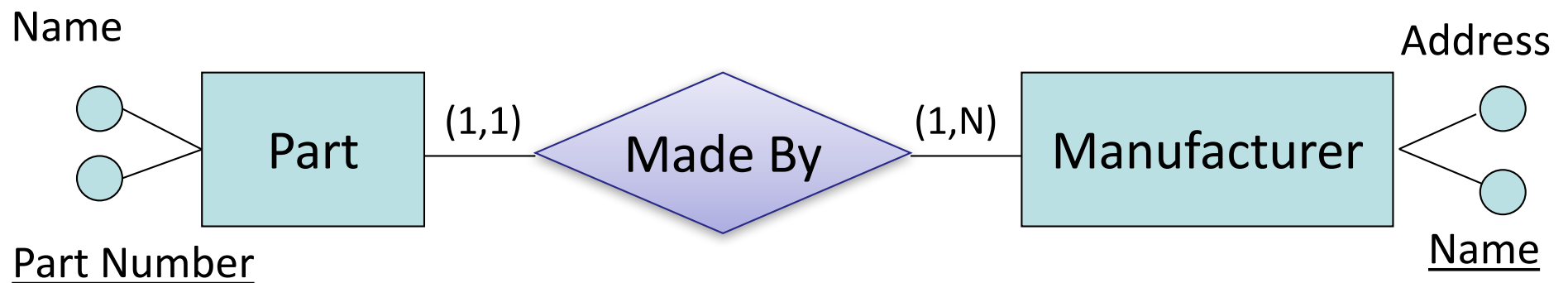
**Output:** Restructured E/R Schema

Restructuring includes:

- Analysis of redundancies
- Choosing entity set vs attribute
- Limiting the use of weak entity sets
- Selection of keys
- Creating entity sets to replace attributes with cardinality greater than one

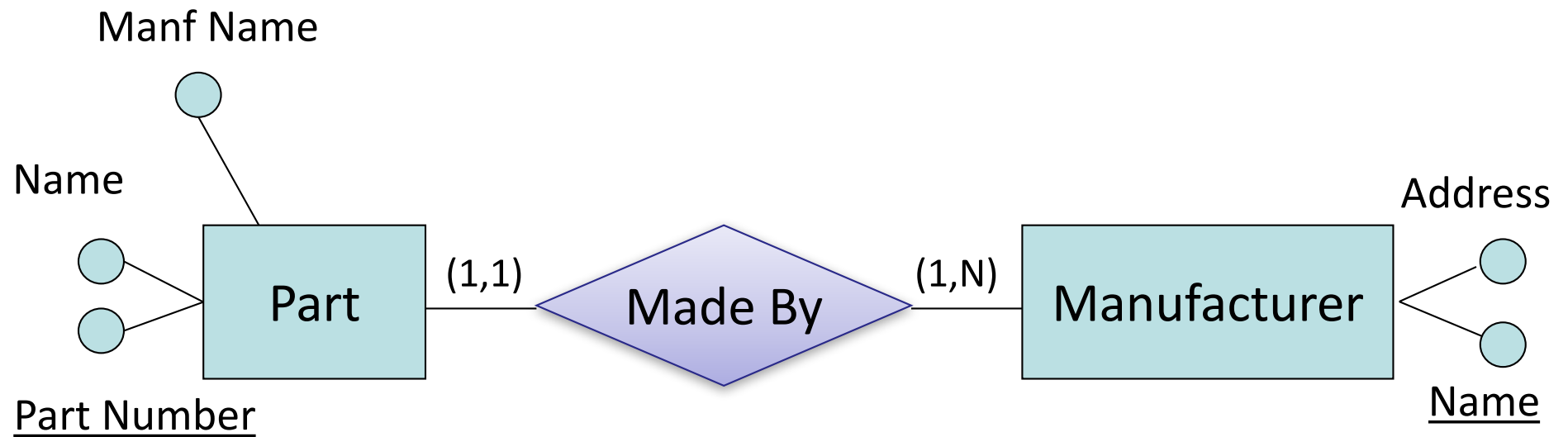
# Example: no redundancy

It is not redundant to have Name twice.



# Example: redundancy

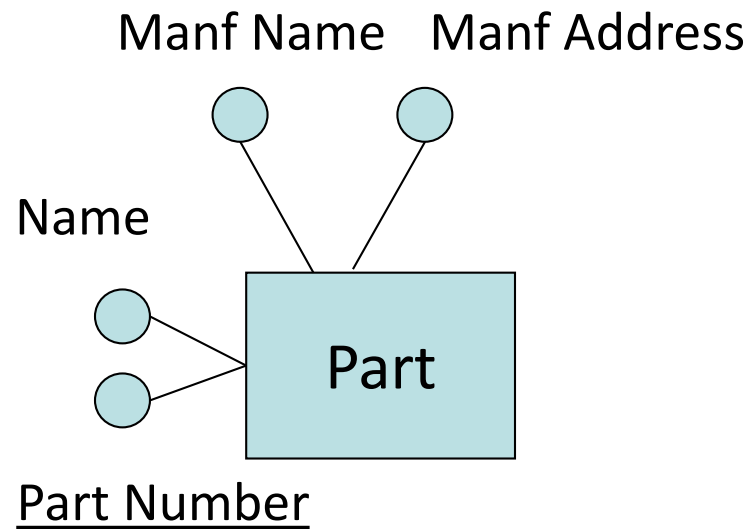
What is redundant here?





# Example: redundancy

What is redundant here?



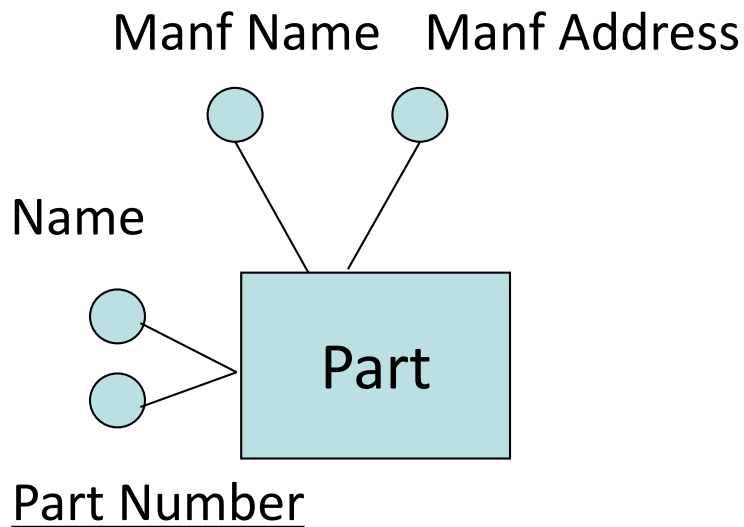
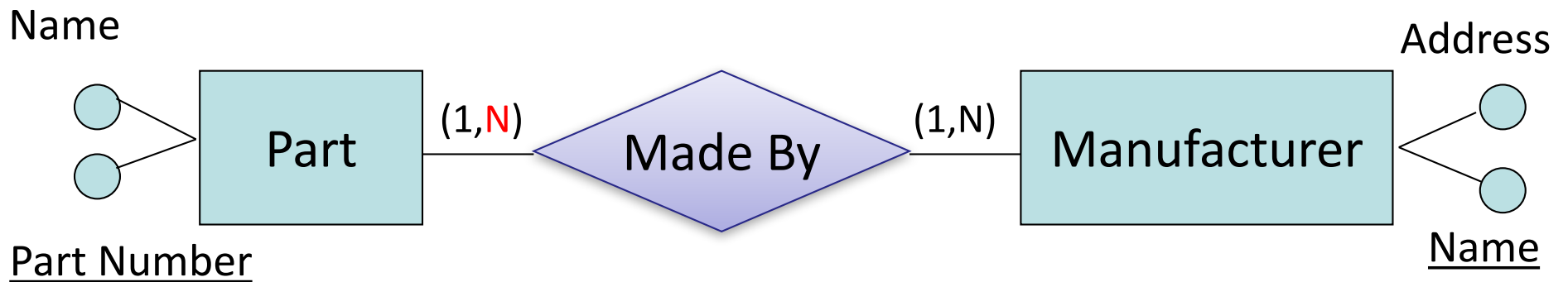
# Entity Sets Versus Attributes

- An entity set should satisfy at least one of the following conditions:
  - It is more than the name of something; it has at least one non-key attribute.
  - or
  - It is the “many” in a many-one or many-many relationship.
- Rules of thumb
  - A “thing” in its own right => Entity Set
  - A “detail” about some other “thing” => Attribute

*Really this is just about avoiding redundancy*

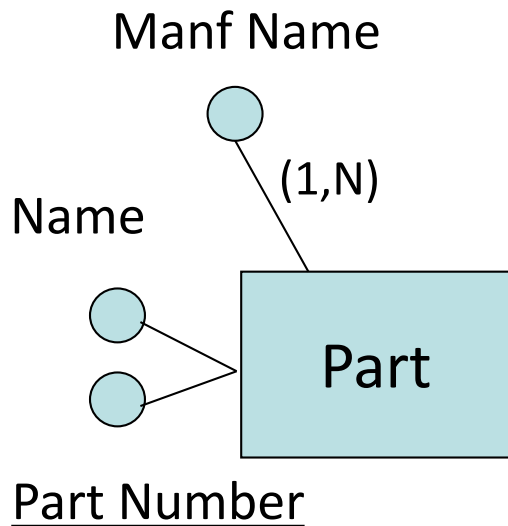
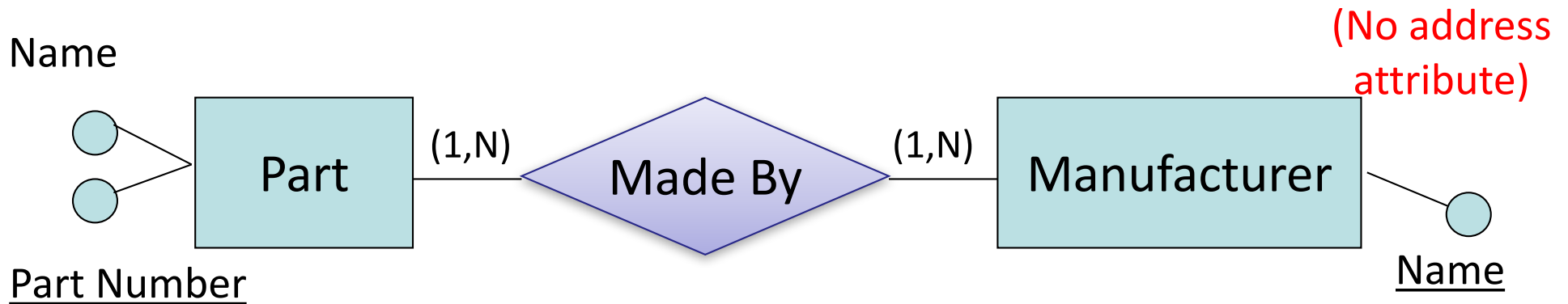
# E.S. vs. attributes: examples

Domain fact change: **A part can have more than one manufacturer ...**



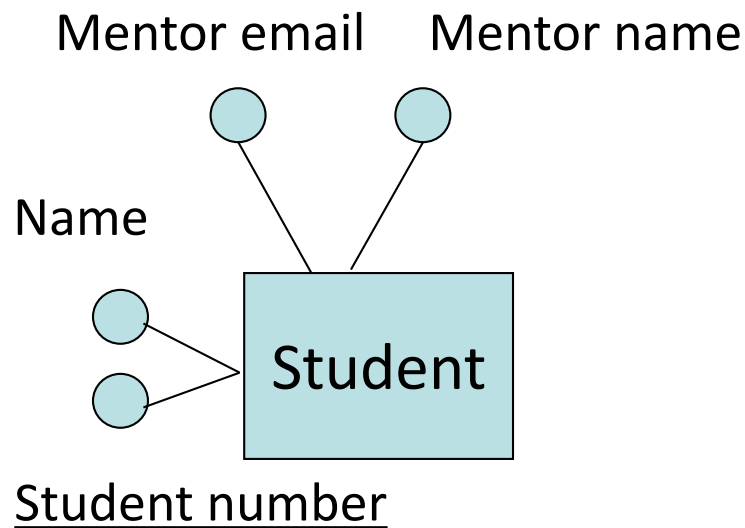
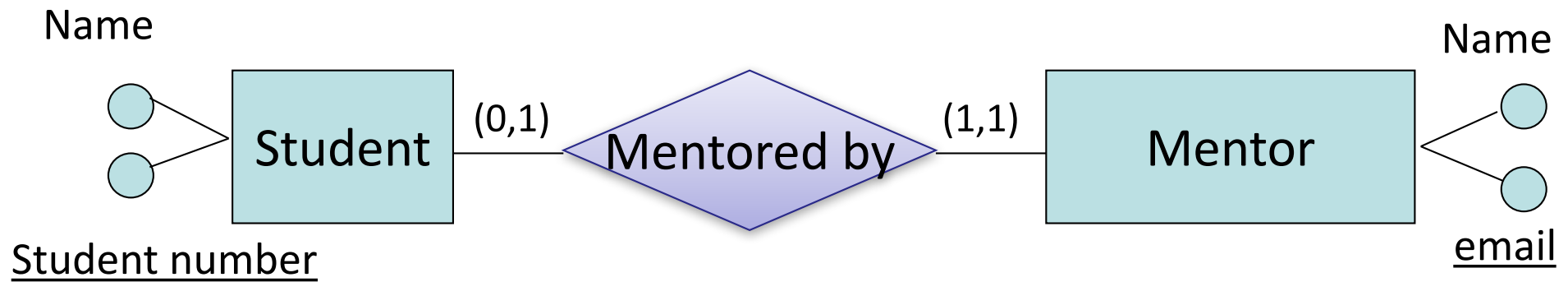
# E.S. vs. attributes: examples

Domain fact change: **Not representing Manufacturer address ...**



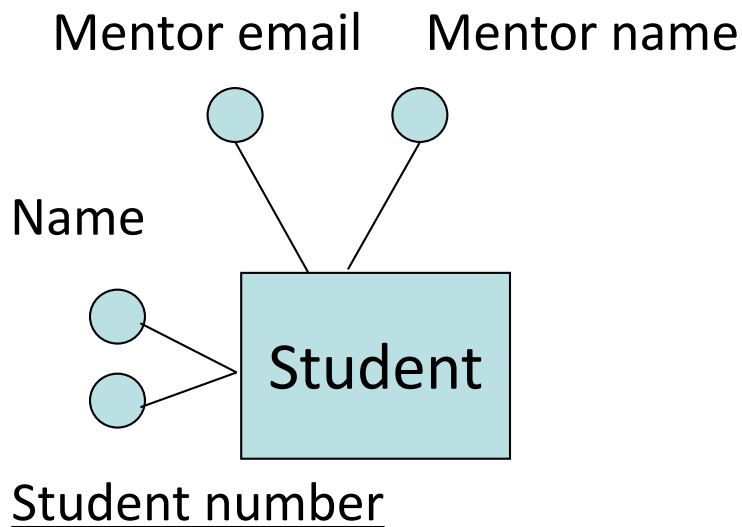
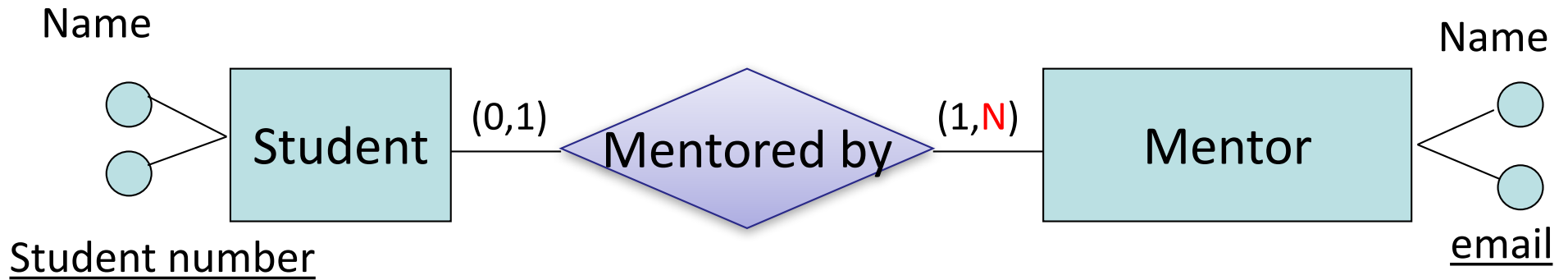
# E.S. vs. attributes: examples

## New domain



# E.S. vs. attributes: examples

Domain fact change: **A mentor can have more than one mentee ...**



# When to use weak entity sets?

- The usual reason is that there is no global authority capable of creating unique ID's
- **Example:** it is unlikely that there could be an agreement to assign unique student numbers across all students in the world

# Don't Overuse Weak Entity Sets

- Beginning database designers often doubt that anything could be a key by itself
  - They make all entity sets weak, supported by all other entity sets to which they are linked
- It is usually better to create unique IDs
  - Social insurance number, automobile VIN, etc.
  - Useful for many reasons (next slide)



# Selecting a Primary Key

- Every relation must have a 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

# Keeping keys simple

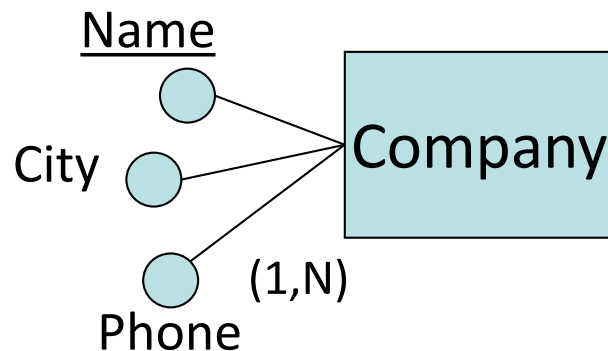
## Multi-attribute and/or string keys...

- ... are wasteful
  - e.g. Movies(title, year, ...): 2 attributes, ~16 bytes
  - Number of movies ever made  $\ll 2^{32}$  (4 bytes)
  - => Integer movieID key saves 75% space and a lot of typing
- ... break encapsulation
  - e.g. Patient(firstName, lastName, phone, ...)
  - Security/privacy hole
  - => Integer patientID prevents information leaks
- ... are brittle (nasty interaction of above two points)
  - Name or phone number change? Parent and child with same name?
  - Patient with no phone? Two movies with same title and year?
  - => Internal ID always exist, are immutable, unique

*Also: computers are really good at integers...*

# Attributes with cardinality $> 1$

- The relational model doesn't allow multi-valued attributes. We must convert these to entity sets.



## **2. TRANSLATING AN E/R MODEL INTO A DB SCHEMA**

# Translation into a Logical Schema

**Input:** E/R Schema

**Output:** Relational Schema

- Starting from an E/R schema, an equivalent relational schema is constructed
  - “**equivalent**”: a schema capable of representing the same information
- A good translation should also:
  - not allow redundancy
  - not invite unnecessary null values

# The general idea

- Each entity set becomes a relation.  
Its attributes are
  - the attributes of the entity set.
- Each relationship becomes a relation.  
It's attributes are
  - the keys of the entity sets that it connects, plus
  - the attributes of the relationship itself.
- We'll see opportunities to simplify.

# Many-to-Many Binary Relationships



# Many-to-Many Binary Relationships



Employee(Number, Surname, Salary)

Project(Code, Name, Budget)

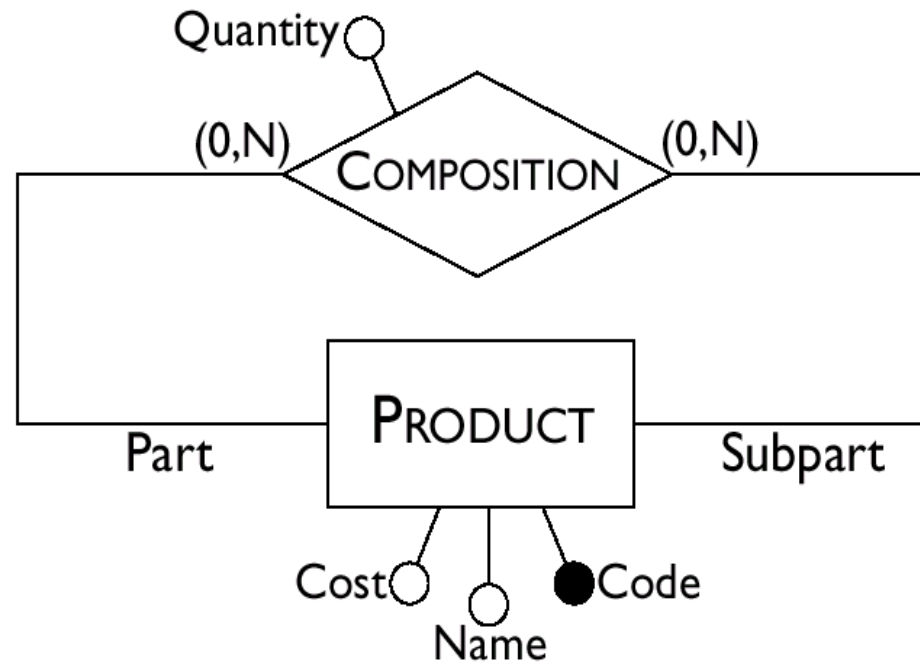
Participation(Number, Code, StartDate)

Participation[Number]  $\subseteq$  Employee[Number]

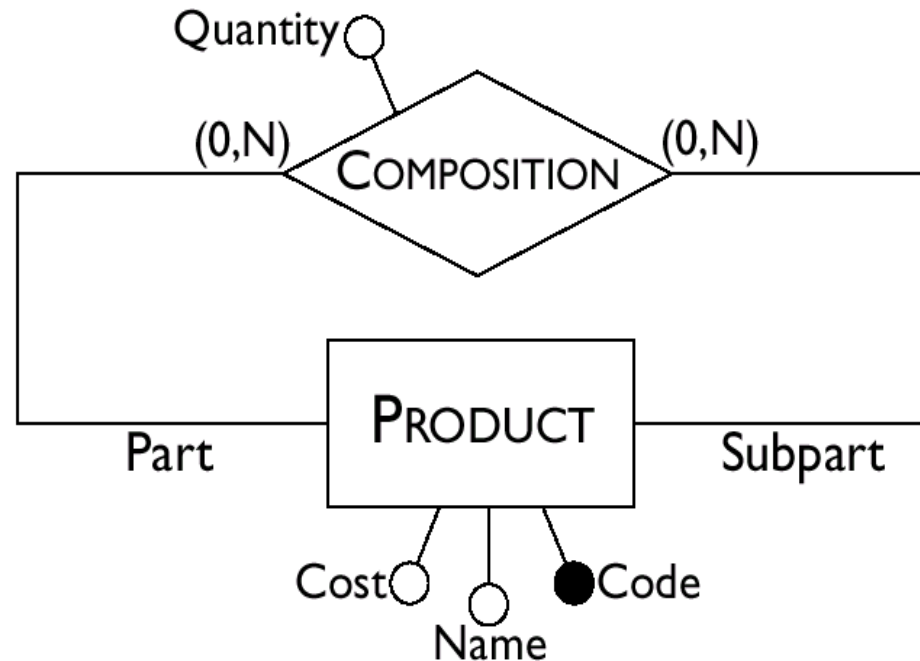
Participation[Code]  $\subseteq$  Project[Code]



# Many-to-Many Recursive Relationships



# Many-to-Many Recursive Relationships



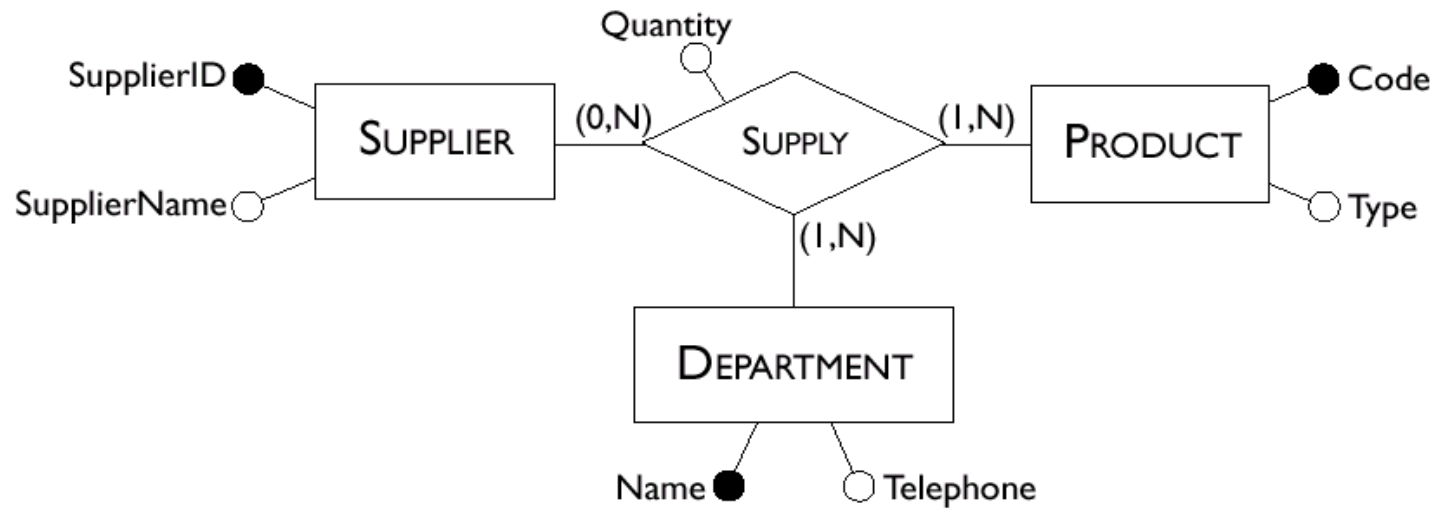
Product(Code, Name, Cost)

Composition(Part, SubPart, Quantity)

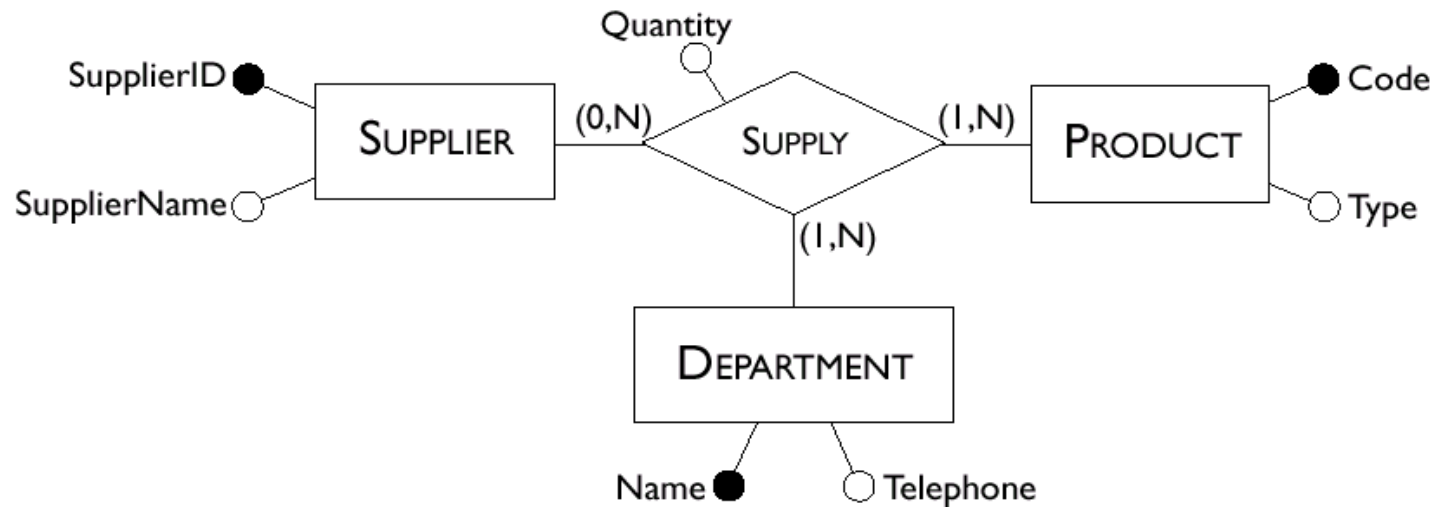
$\text{Composition}[\text{Part}] \subseteq \text{Product}[\text{Code}]$

$\text{Composition}[\text{SubPart}] \subseteq \text{Product}[\text{Code}]$

# Many-to-Many Ternary Relationships



# Many-to-Many Ternary Relationships



Supplier(SupplierID, SupplierName)

Product(Code, Type)

Department(Name, Telephone)

Supply(Supplier, Product, Department, Quantity)

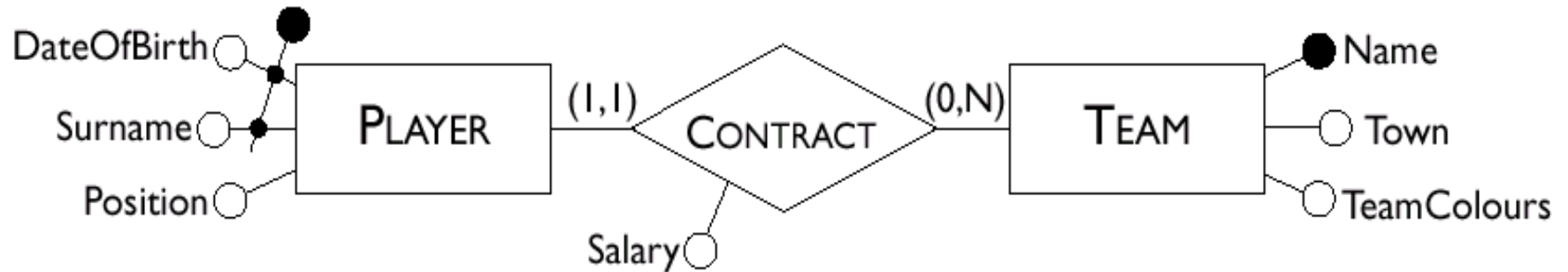
$\text{Supply}[\text{Supplier}] \subseteq \text{Supplier}[\text{SupplierID}]$

$\text{Supply}[\text{Product}] \subseteq \text{Product}[\text{Code}]$

$\text{Supply}[\text{Department}] \subseteq \text{Department}[\text{Name}]$

# One-to-Many Relationships

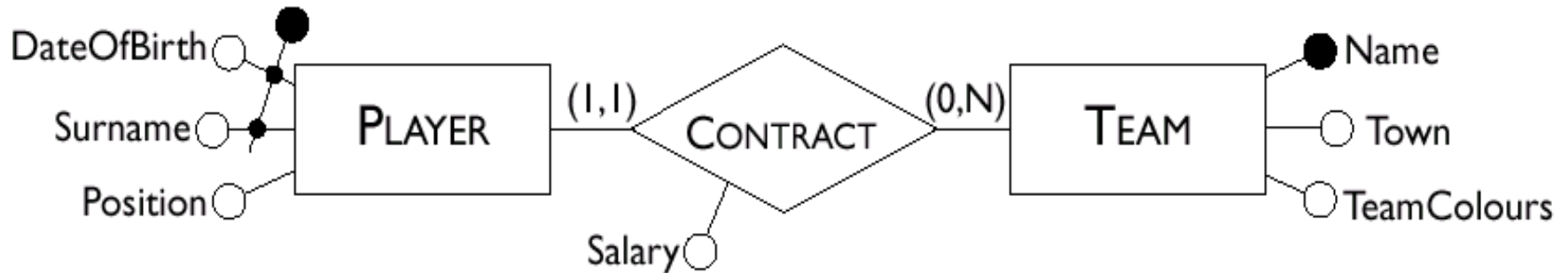
with mandatory participation on the “one” side





# One-to-Many Relationships

with mandatory participation on the “one” side



Player(Surname, DOB, Position)

Team(Name, Town, TeamColours)

Contract(PlayerSurname, PlayerDOB, Team, Salary)

$\text{Contract}[\text{PlayerSurname}, \text{PlayerDOB}] \subseteq \text{Player}[\text{Surname}, \text{DOB}]$

$\text{Contract}[\text{Team}] \subseteq \text{Team}[\text{Name}]$

*OR*

Player(Surname, DOB, Position, TeamName, Salary)

Team(Name, Town, TeamColours)

$\text{Player}[\text{TeamName}] \subseteq \text{Team}[\text{Name}]$

# One-to-One Relationships with mandatory participation for both





# One-to-One Relationships with mandatory participation for both



The standard translation, with 3 relations (what is key of management)?

*Or*

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

Department(Name, Telephone, Branch)

$\text{Head}[\text{Department}] \subseteq \text{Department}[\text{Name}]$

*Or*

Head(Number, Name, Salary)

Department(Name, Telephone, Branch, HeadNumber, StartDate)

$\text{Department}[\text{HeadNumber}] \subseteq \text{Head}[\text{Number}]$



# One-to-One Relationships with mandatory participation for both



*Or*

Head(Number, Name, Salary, StartDate)

Department(Name, Telephone, HeadNumber, Branch)

Department[HeadNumber]  $\subseteq$  Head[Number]

# One-to-One Relationships with optional participation for one





# One-to-One Relationships with optional participation for one



Employee(Number, Name, Salary)

Department(Name, Telephone, Branch)

Management(Head, Department, StartDate)

Management[Head]  $\subseteq$  Employee[Number]

Management[Department]  $\subseteq$  Department[Name]

*Or*

Employee(Number, Name, Salary)

Department(Name, Telephone, Branch, Head, StartDate)

Department[Head]  $\subseteq$  Employee[Number]

# Summary of Types of Relationship

- many-to-many (binary or ternary)
- one-to-many
  - mandatory: (1,1) on the “one” side
  - optional: (0,1) on the “one” side
- one-to-one
  - both mandatory: (1,1) on both sides
  - one mandatory, one optional:  
(1,1) on one side and (0,1) on other side
  - both optional: (0,1) on both sides

# Will the schema be “good”?

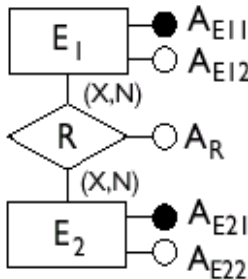
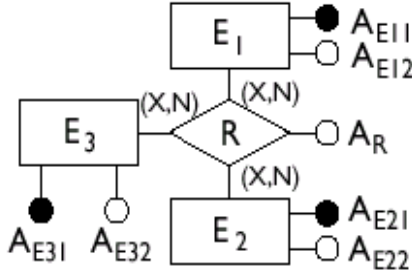
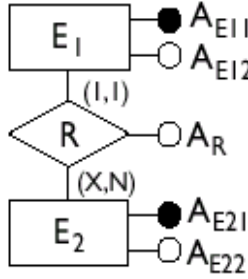
- If we use this process, will the schema we get be a good one?
- The process should ensure that there is no redundancy.
- But only with respect to what the E/R diagram represents.
- Crucial thing we are missing: functional dependencies. (We only have keys, not other FDs.)
- So we still need FD theory.

# Redundancy can be *desirable*

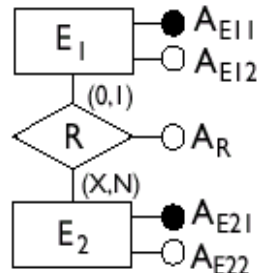
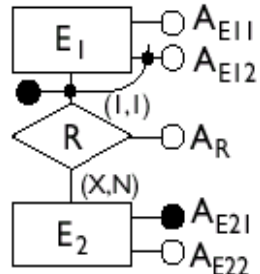
- **Disadvantages** of redundancy:
  - More storage (but usually at negligible cost)
  - Additional operations to keep the data consistent
- **Advantages** of redundancy:
  - Speed: Fewer accesses necessary to obtain information
- How to decide to maintain or eliminate a redundancy?  
Examine:
  - **the cost of operations** that involve the redundant information and
  - **the storage needed** for the redundant informationwith and without the redundancy.

*Performance analysis is required to decide about redundancy*

# 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		$E_1(\underline{A_{E11}}, A_{E12}, A_{E21}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$

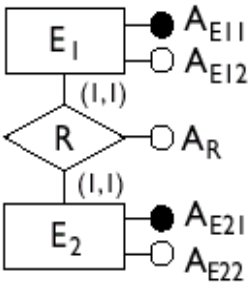
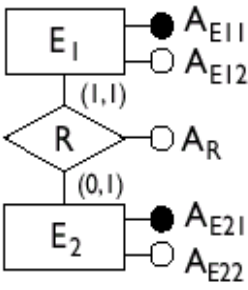
# ...More Rules...

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		$E_1(\underline{A_{E12}}, \underline{A_{E21}}, A_{E11}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$

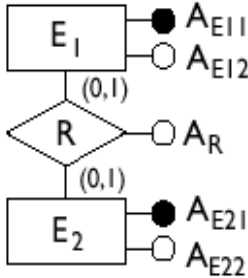
\* Indicates that nulls are allowed



# ...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		$E_1(\underline{A_{E11}}, A_{E21})$ $E_2(\underline{A_{E21}}, A_{E22}, A_{E11}^*, A_R^*)$ <p>Alternatively:</p> $E_1(\underline{A_{E11}}, A_{E12}, A_{E21}^*, A_R^*)$ $E_2(\underline{A_{E21}}, A_{E22})$ <p>Alternatively:</p> $E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, A_R)$