

# How do you like to see the Spy Database? As tables?

**Agent**(agent\_id, first, middle, last, address, city, country, salary, clearance\_id)

**Mission**(mission\_id, name, access\_id, team\_id, mission\_status)

**Affiliation**(aff\_id, title, description)

**AffiliationRel**(aff\_id, agent\_id, affiliation\_strength)

**LanguageRel**(lang\_id, agent\_id)

**Language**(lang\_id, language)

**SecurityClearance**(sc\_id, sc\_level, description)

**Skill**(skill\_id, skill)

**SkillRel**(skill\_id, agent\_id)

**Team**(team\_id, name, meeting\_frequency)

**TeamRel**(team\_id, agent\_id)

# How do you like to see the Spy Database? As tables with foreign keys?

**Agent**(agent\_id, first, middle, last, address, city, country, salary, clearance\_id)

**Mission**(mission\_id, name, access\_id, team\_id, mission\_status)

**Affiliation**(aff\_id, title, description)

**AffiliationRel**(aff\_id, agent\_id, affiliation\_strength)

**LanguageRel**(lang\_id, agent\_id)

**Language**(lang\_id, language)

**SecurityClearance**(sc\_id, sc\_level, description)

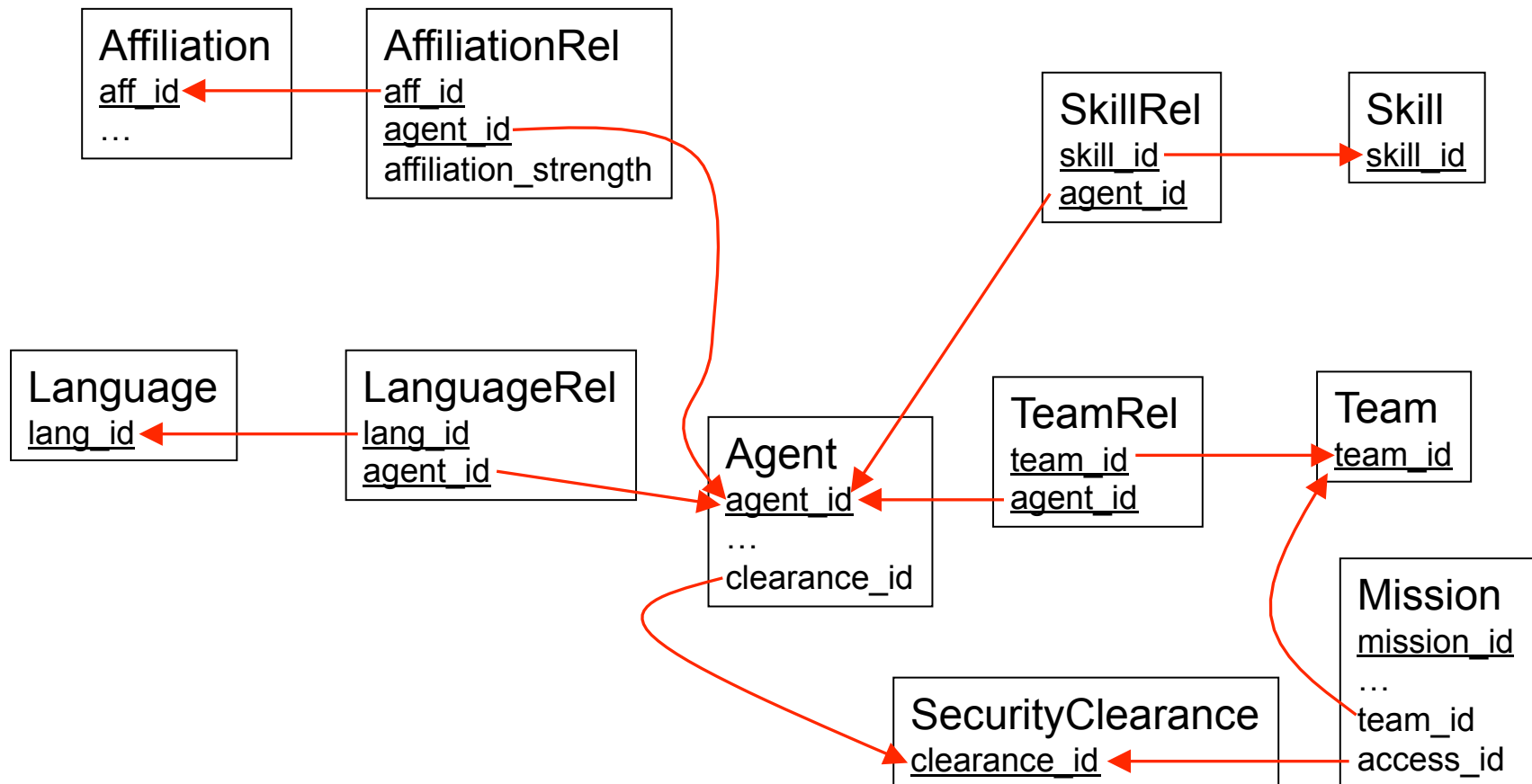
**Skill**(skill\_id, skill)

**SkillRel**(skill\_id, agent\_id)

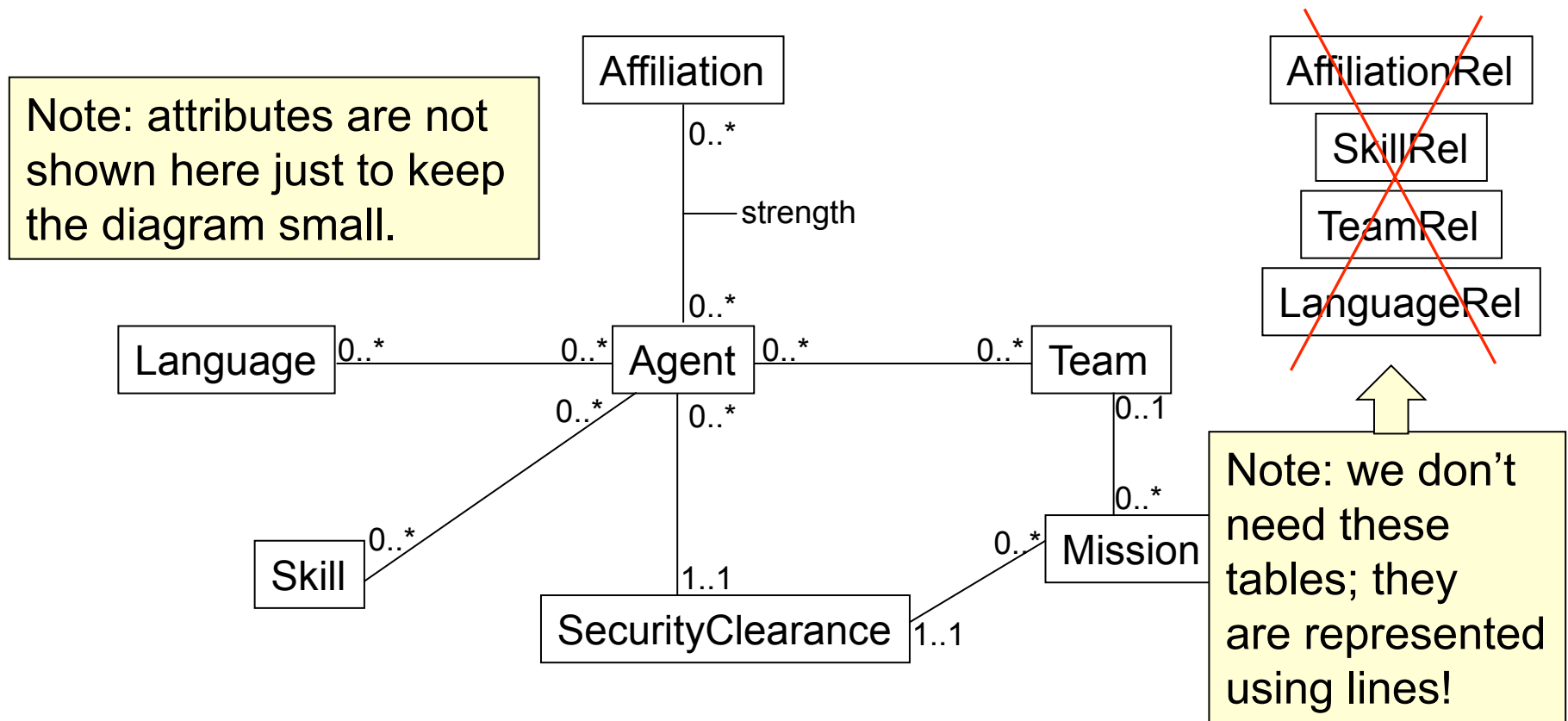
**Team**(team\_id, name, meeting\_frequency)

**TeamRel**(team\_id, agent\_id)

# How do you like to see the Spy Database? MS® Access "Relationship Diagram" - of tables?



# Another way to see the Spy Database: in an Entity-Relationship Diagram (ERD)

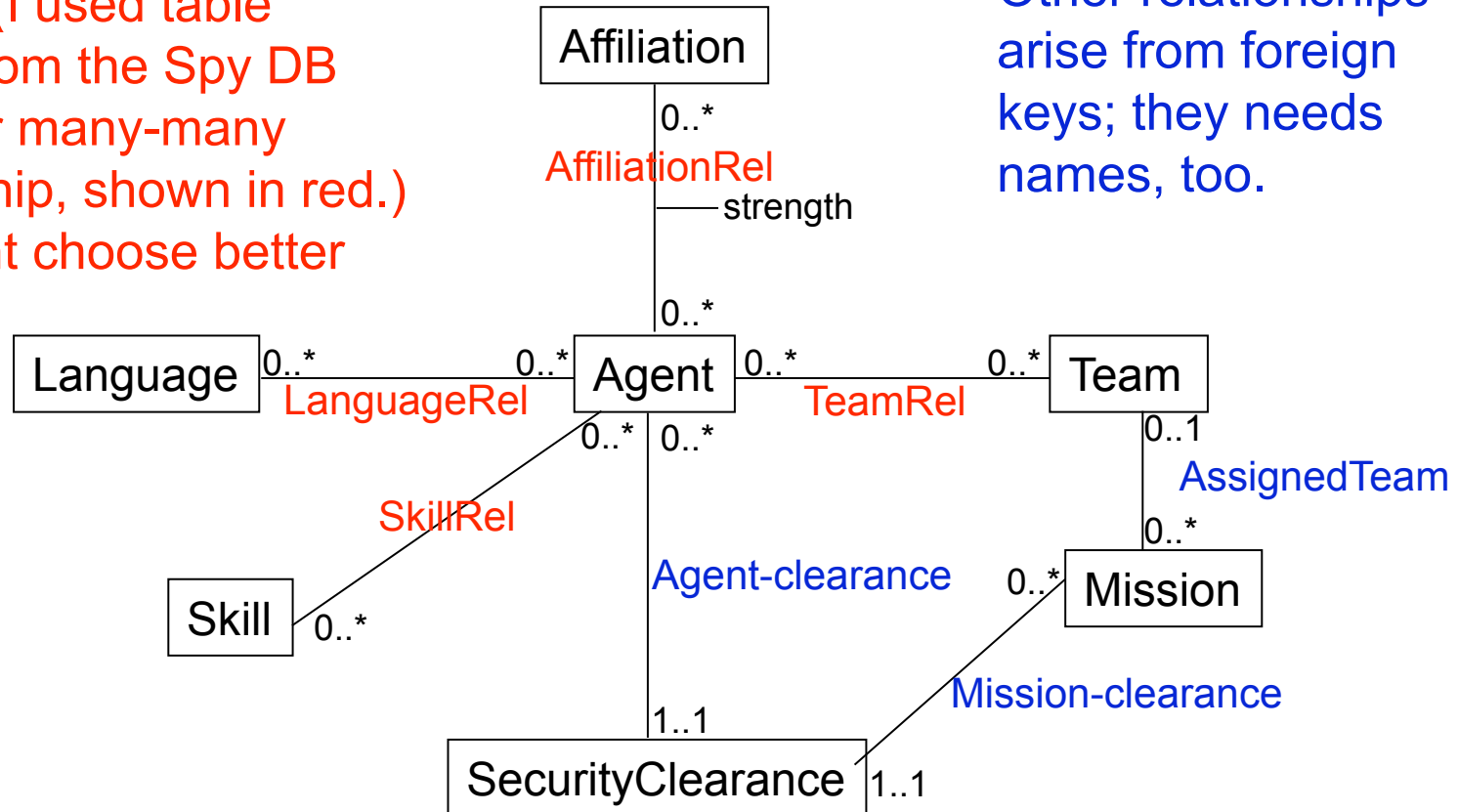


Rectangles: Entities      Lines: Relationships (with cardinalities)

# Entity-Relationship Diagram (ERD)

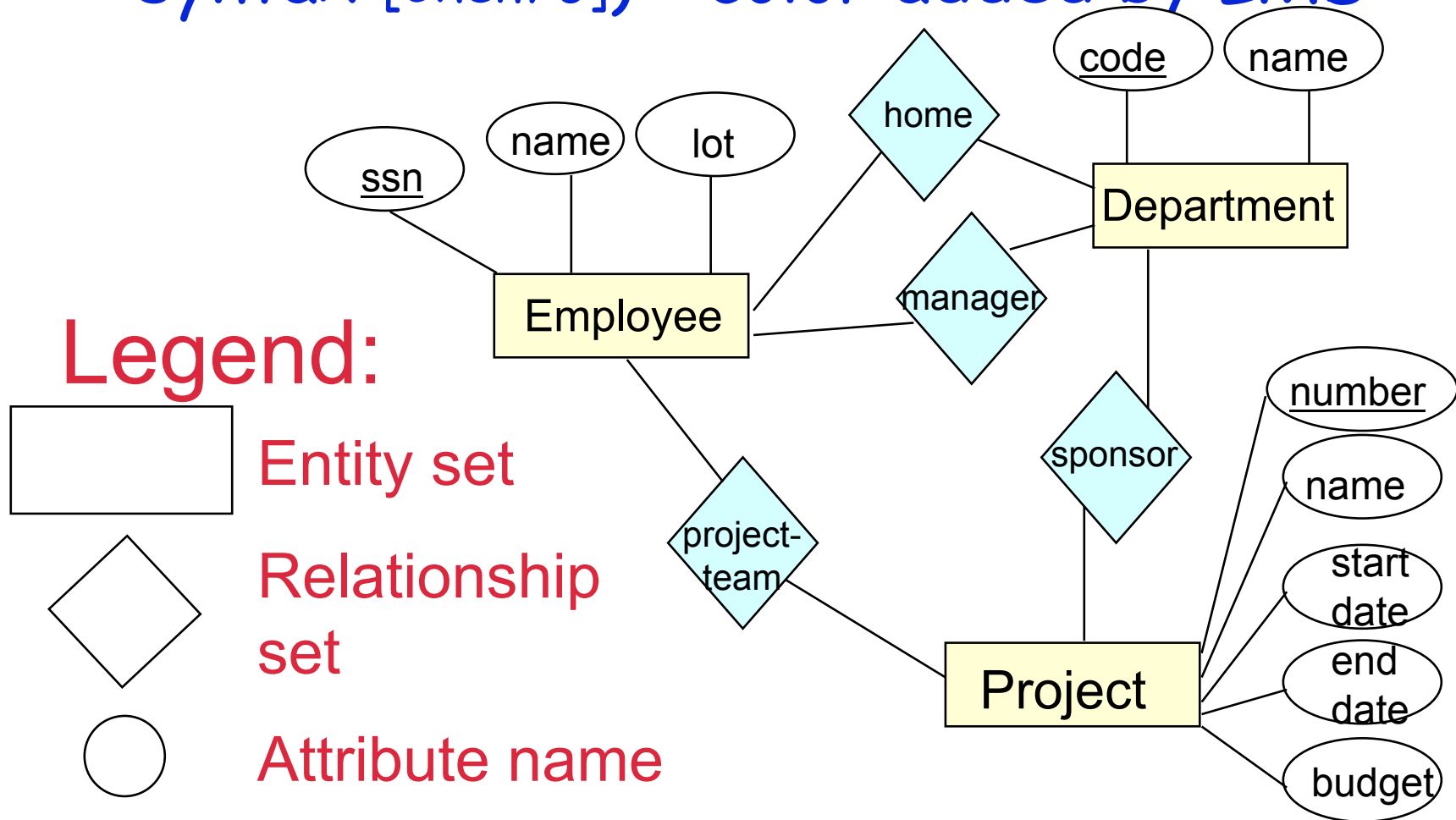
Note: Relationships need names. (I used table names from the Spy DB tables for many-many relationship, shown in red.) You might choose better names.

Other relationships arise from foreign keys; they need names, too.



# Conceptual Database Design Using the Entity-Relationship (ER) Model

# Entity-Relationship Diagram (original syntax [Chen76]) - color added by LMD



# Definitions

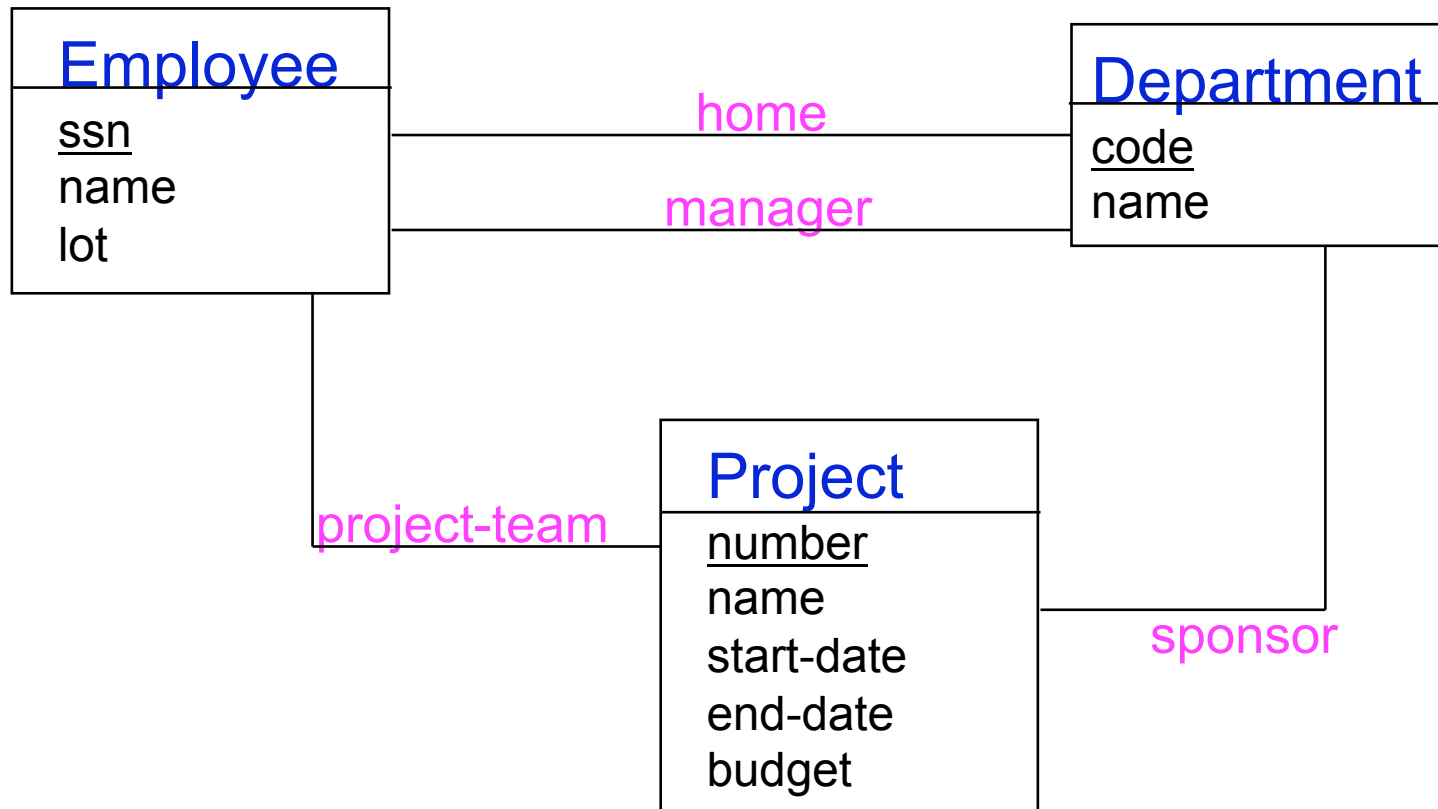
- *Entity*: Real-world object distinguishable from other objects. An employee named **John Smith**, for example. One instance of an entity set.
  - An entity is described using a set of *attributes*.
  - An entity has one or more *keys*.
- *Entity Set*: A collection of similar entities. (**All employees in the company**, for example.)
- An *entity set* is (informally) called an *entity* if everyone knows we're talking about the schema.
- An *entity* in an ERD may also be called an *entity type*. But there is a difference between the *entity definition/type* and the *instance* (the data) of the entity.

## Definitions (cont.)

- Relationship: Association among 2 or more entities. John's **home department** is Pharmacy, for example, where John and Pharmacy are both entities. One instance of a relationship set.
- Relationship Set: Collection of similar relationships. The **home department** relationship set consists of all indications of a home department for an employee.
- A **relationship set** is (informally) called a **relationship** if everyone knows we are talking about the schema. rather than individual rows.
- A **relationship set** may also be called a **relationship type**. But there is a difference between the **relationship definition/type** and the **instance** (the data) for the relationship definition.

# UML version of the same E-R Diagram

UML: Unified Modeling Language - for OO Design



# Equivalent Relational Schema

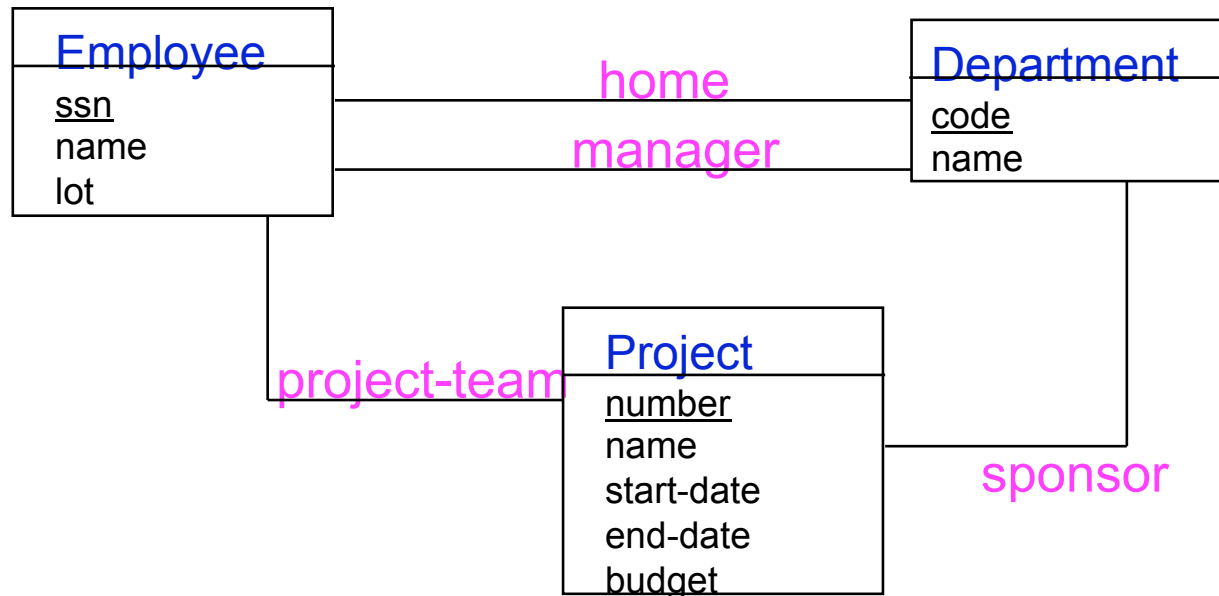
Employee (ssn, name, lot, home-dept)

Project-team(ssn, number)

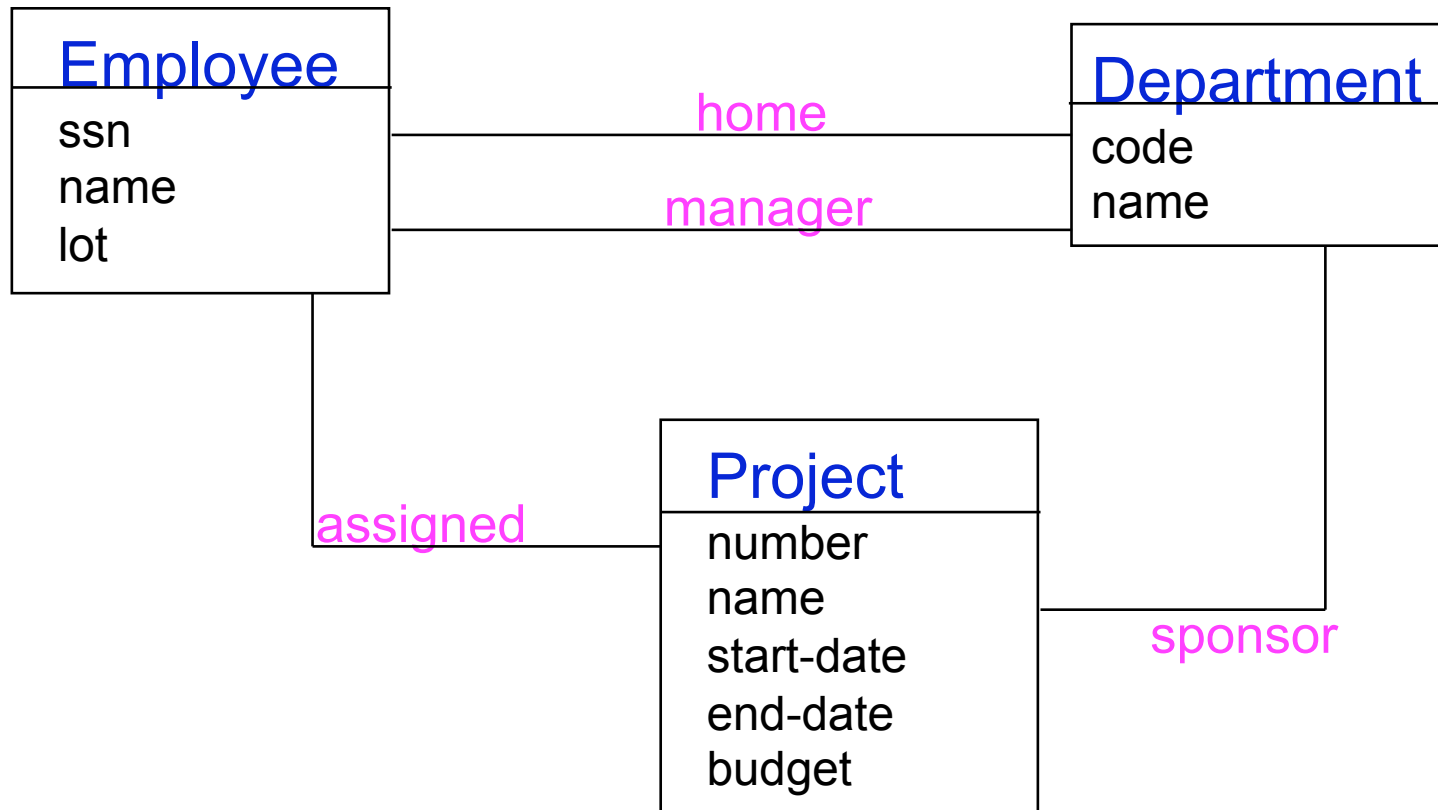
Department (id, name, manager)

Project (number, name, start-date, end-date, budget, sponsor)

extra table to represent a  
many-to-many relationship

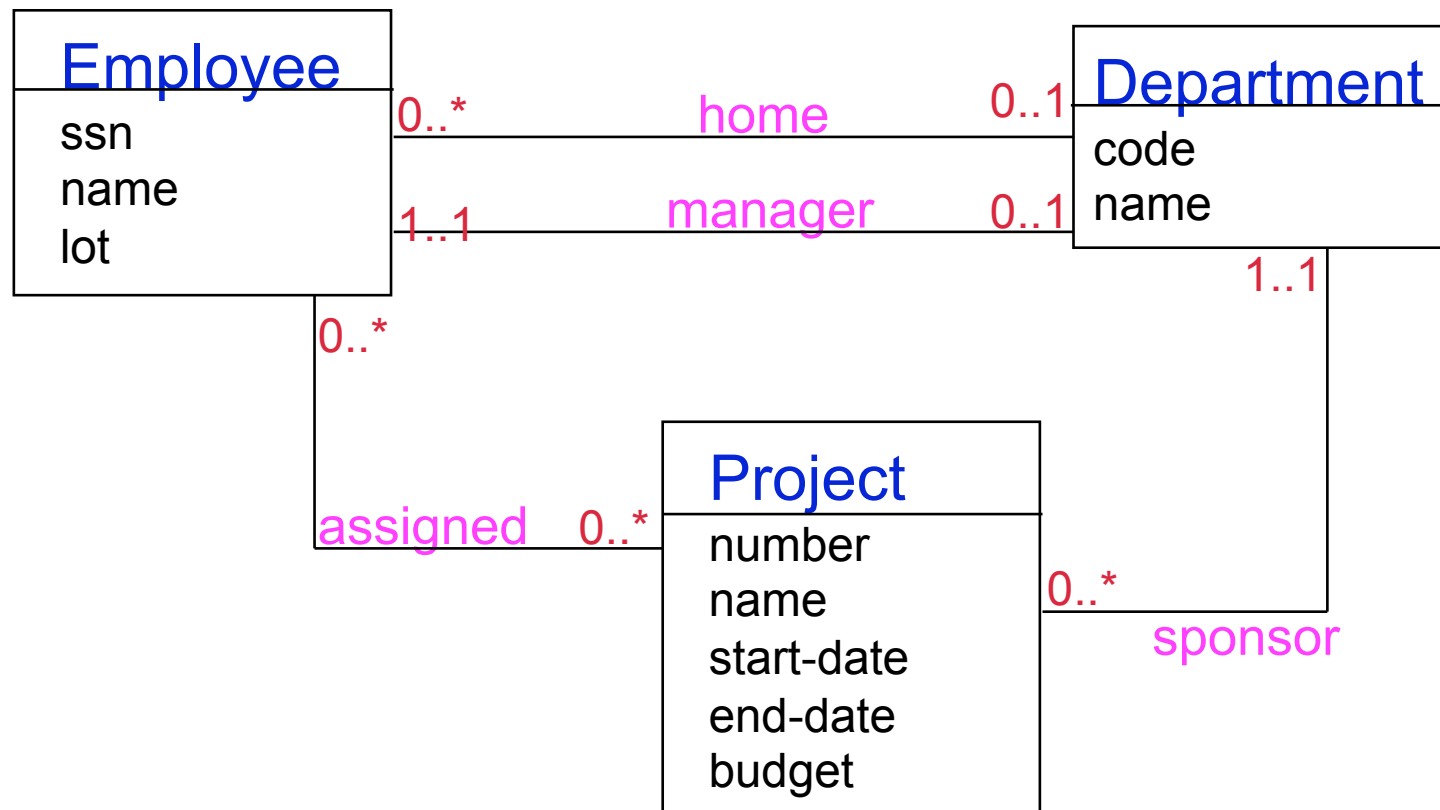


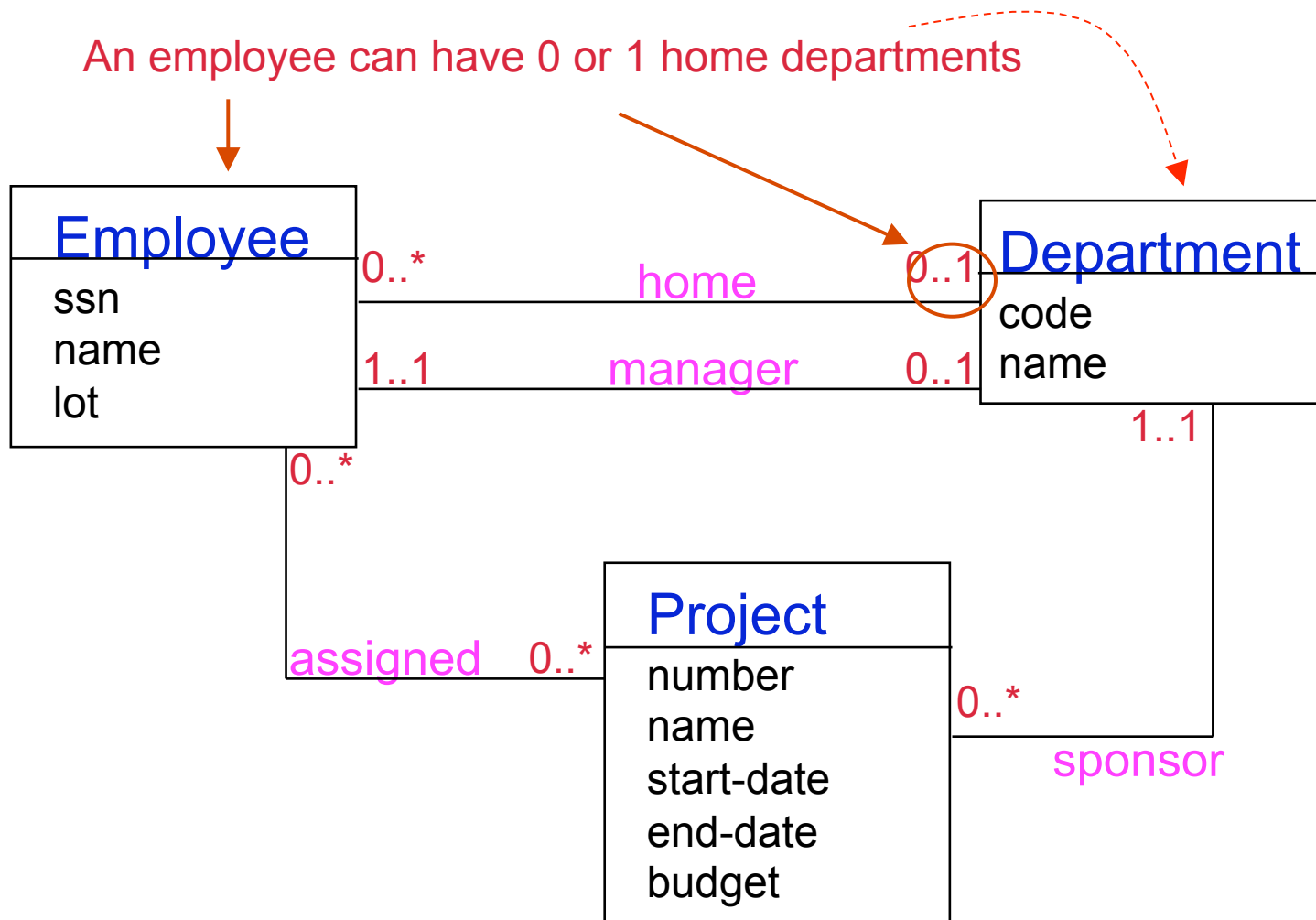
# Cardinality Constraints on Relationship sets: How many entities can participate?

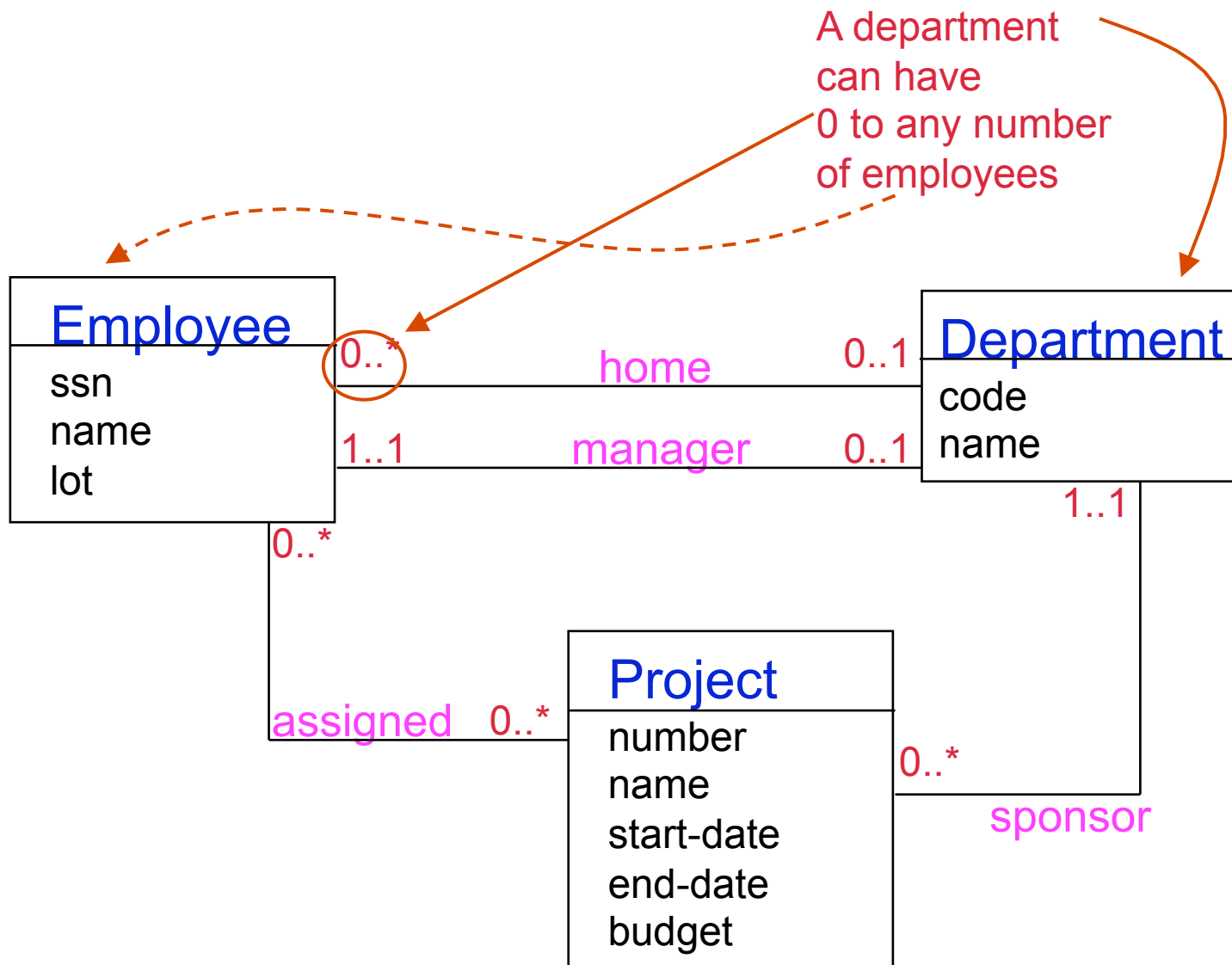


# Cardinality Constraints on Relationship sets

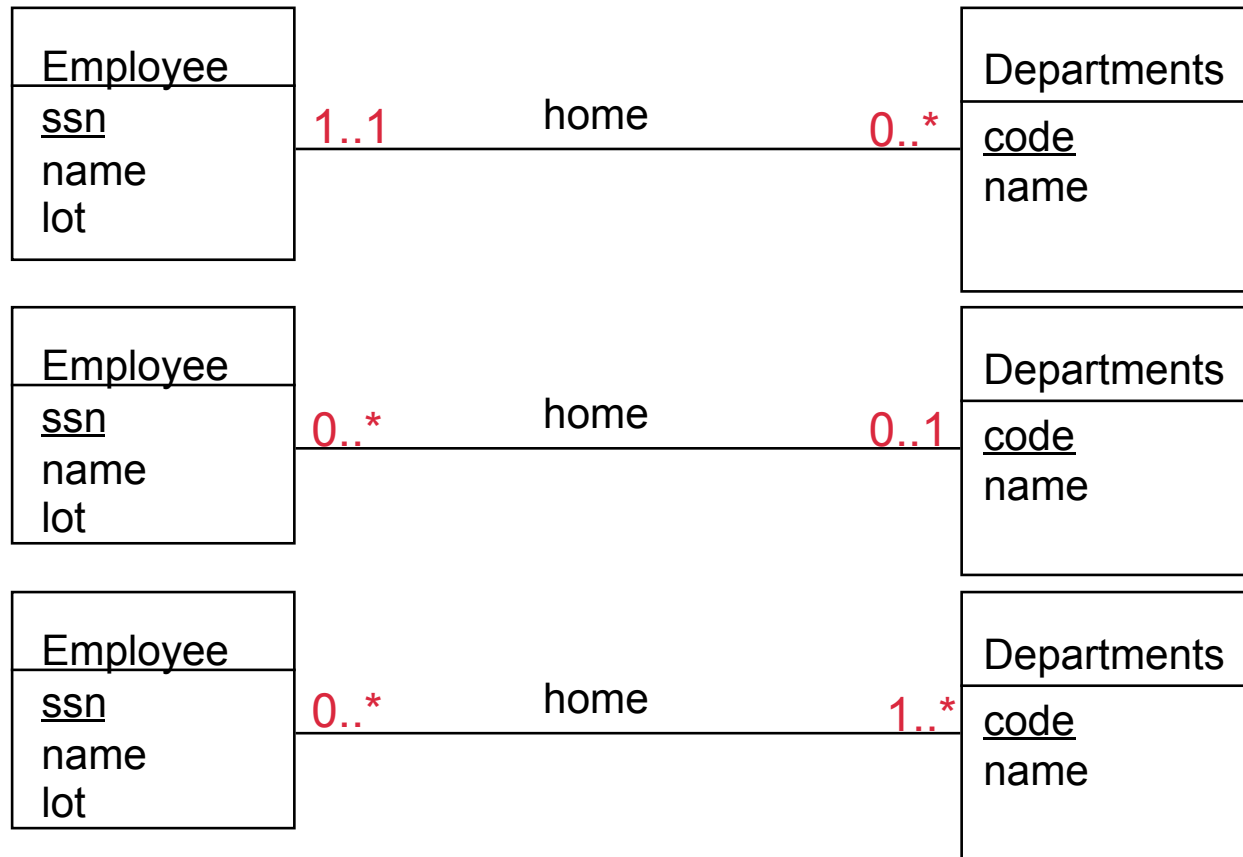
## How many entities can participate?





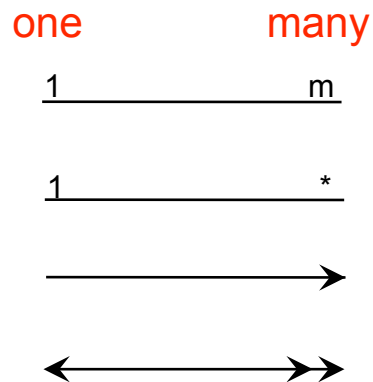


# Unified Modeling Language (UML): Class Diagram

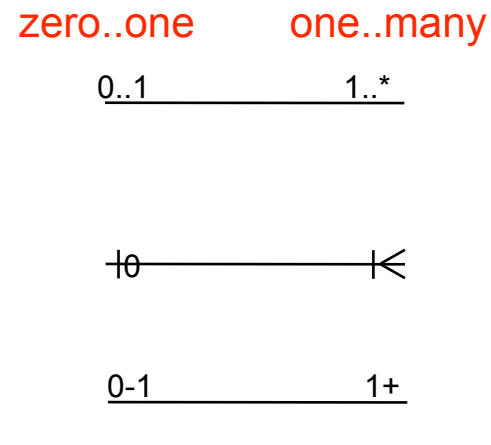


Which one is right? We must discover the semantics of the application!

# Various notation for "one-to-many"



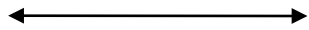
maximum cardinalities only



minimum and maximum cardinalities

# Various notations for "many-to-many"

many                  many



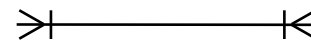
m                          n

\*                                  \*

maximum cardinalities only

one..many                  one..many

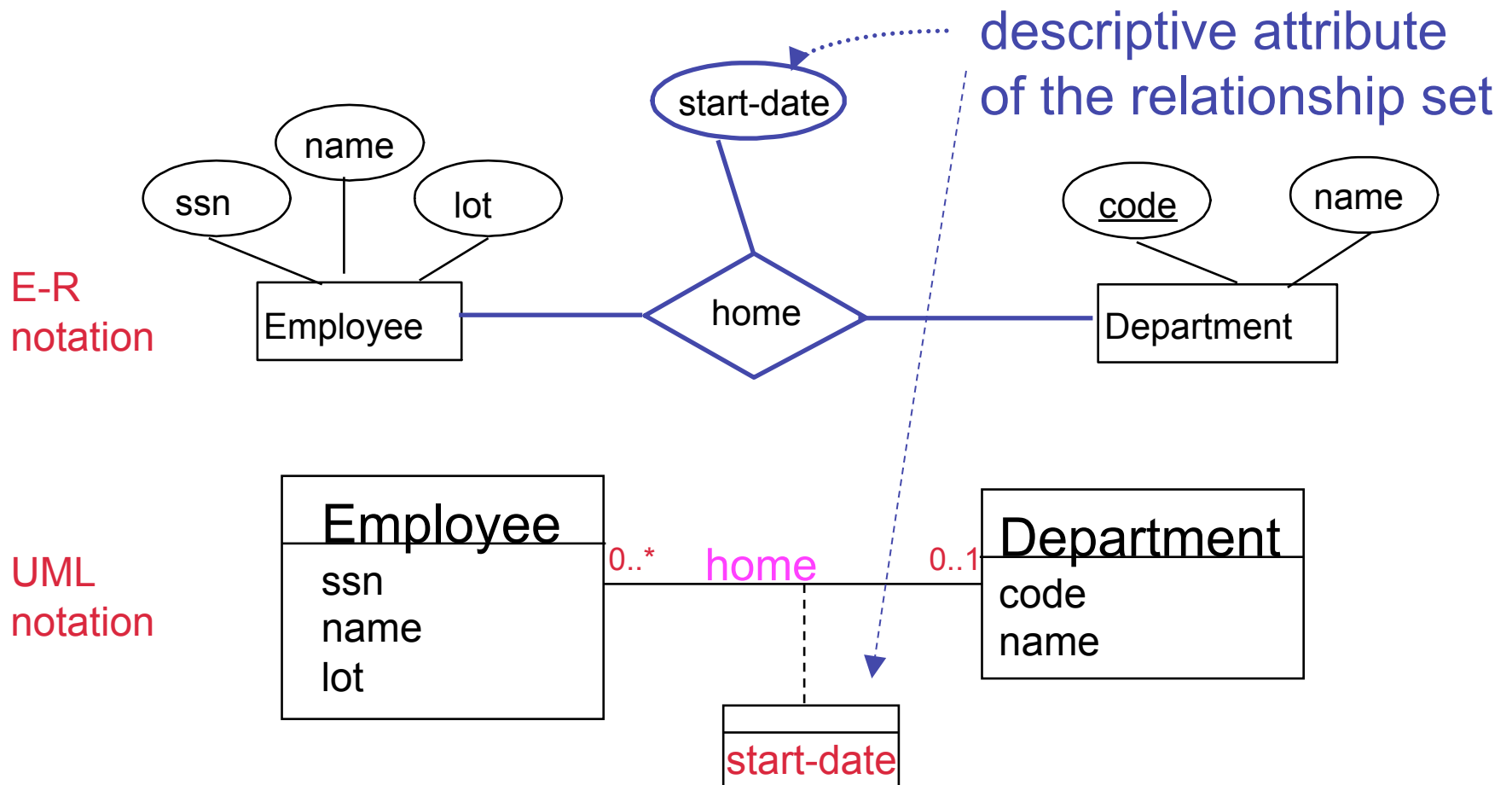
1..\*                                  1..\*



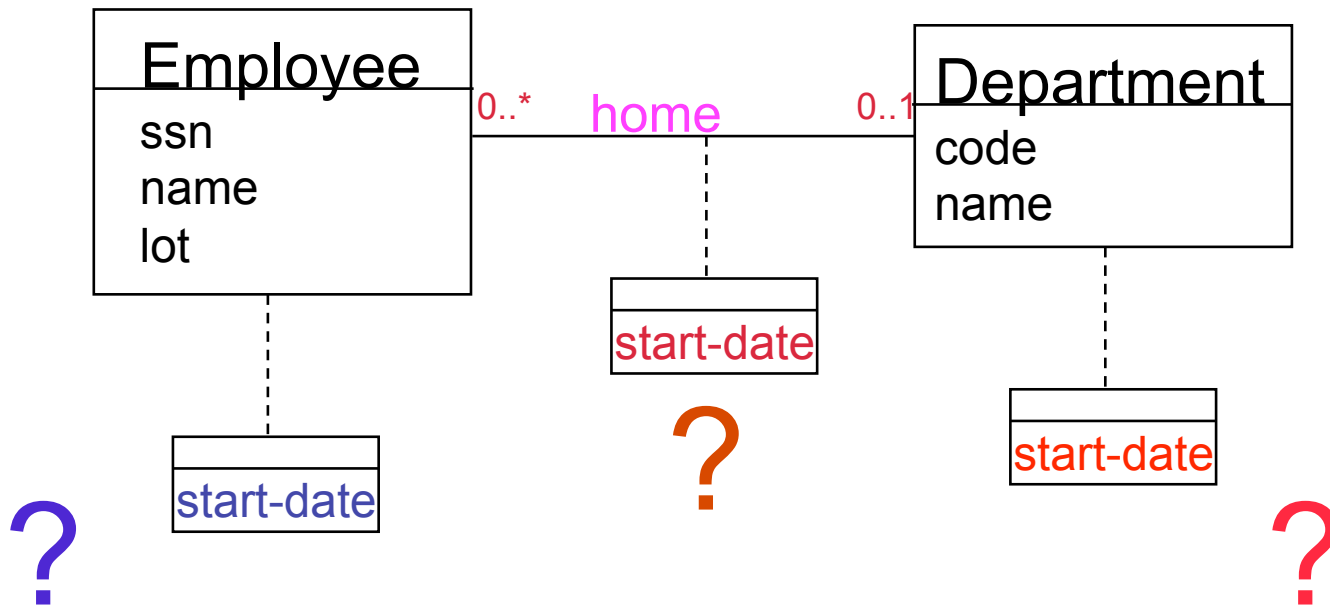
1+    1+

minimum and maximum cardinalities

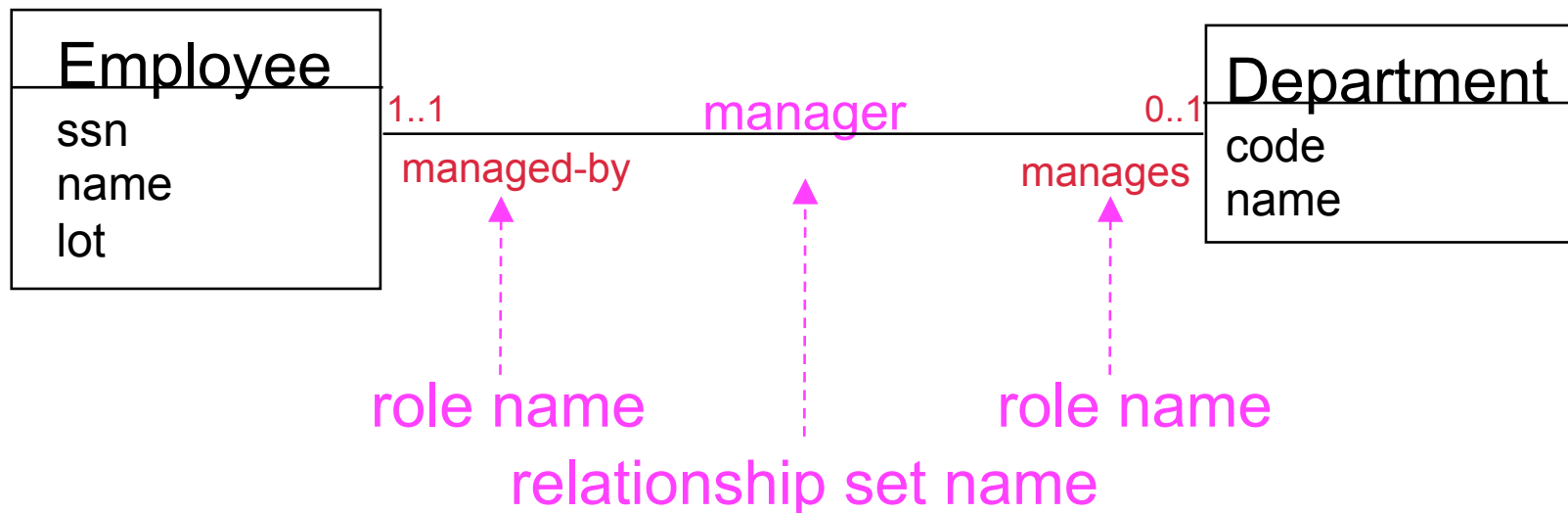
# Relationship sets can have attributes



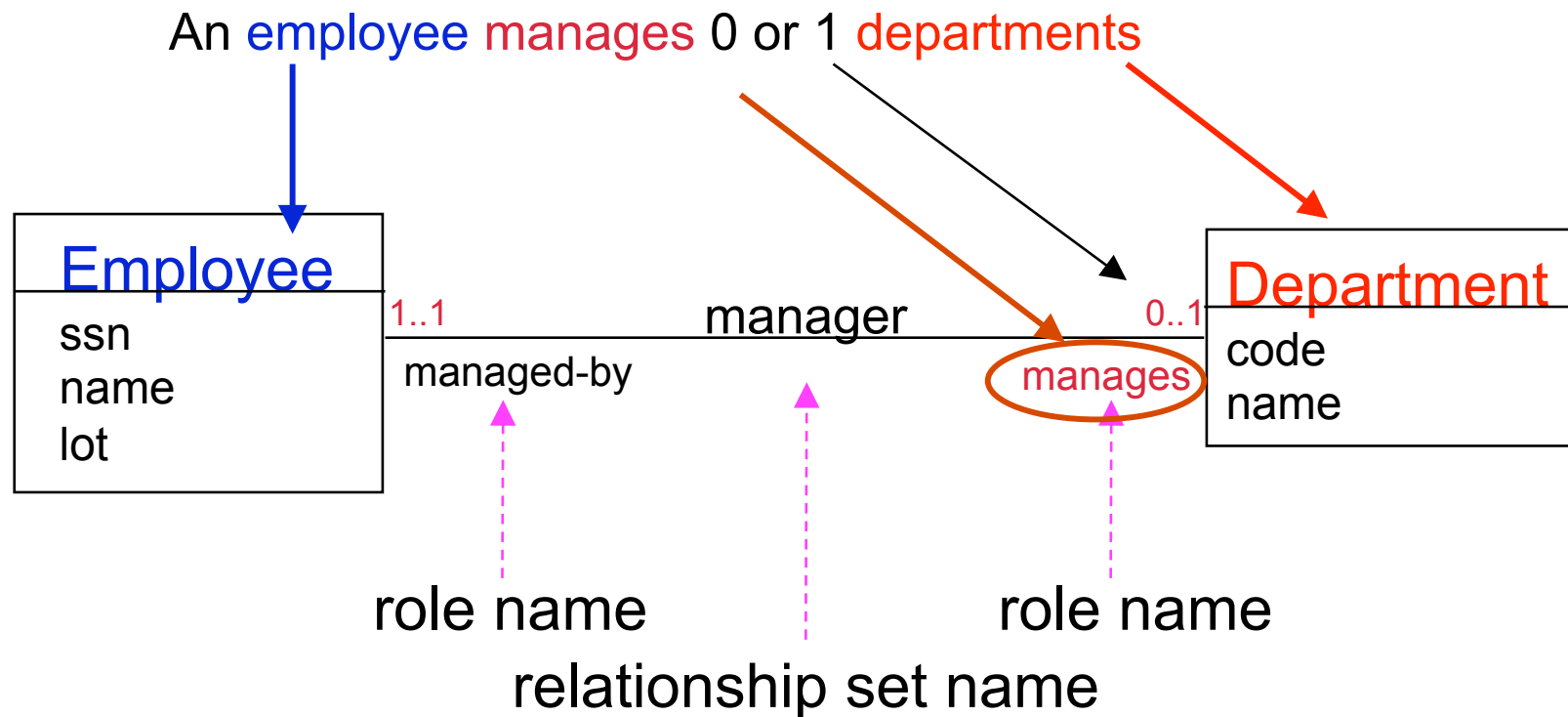
Try all three locations for the attributes:  
which one makes sense?



# Relationship sets can have **role** names (in addition to the name of the relationship set)

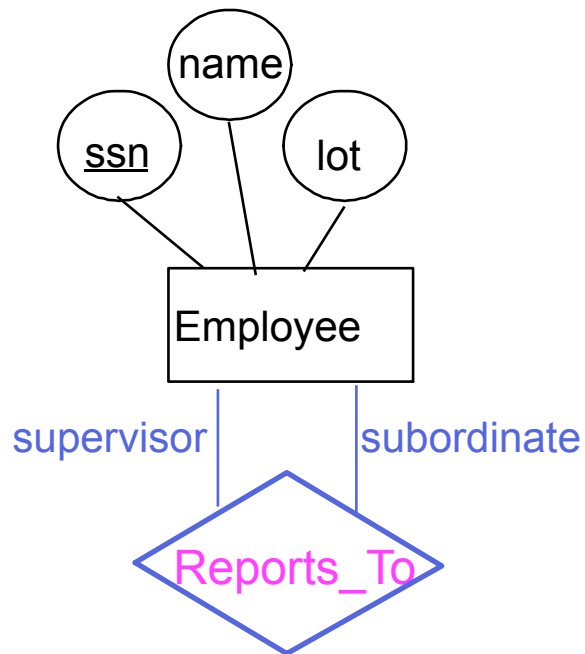


# Example: reading **role** names

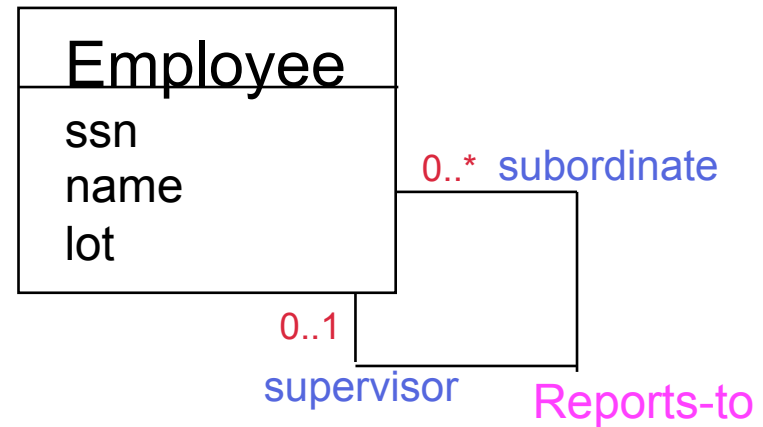


# Same entity sets can participate in different "roles" for the same relationship set

E-R notation



UML notation

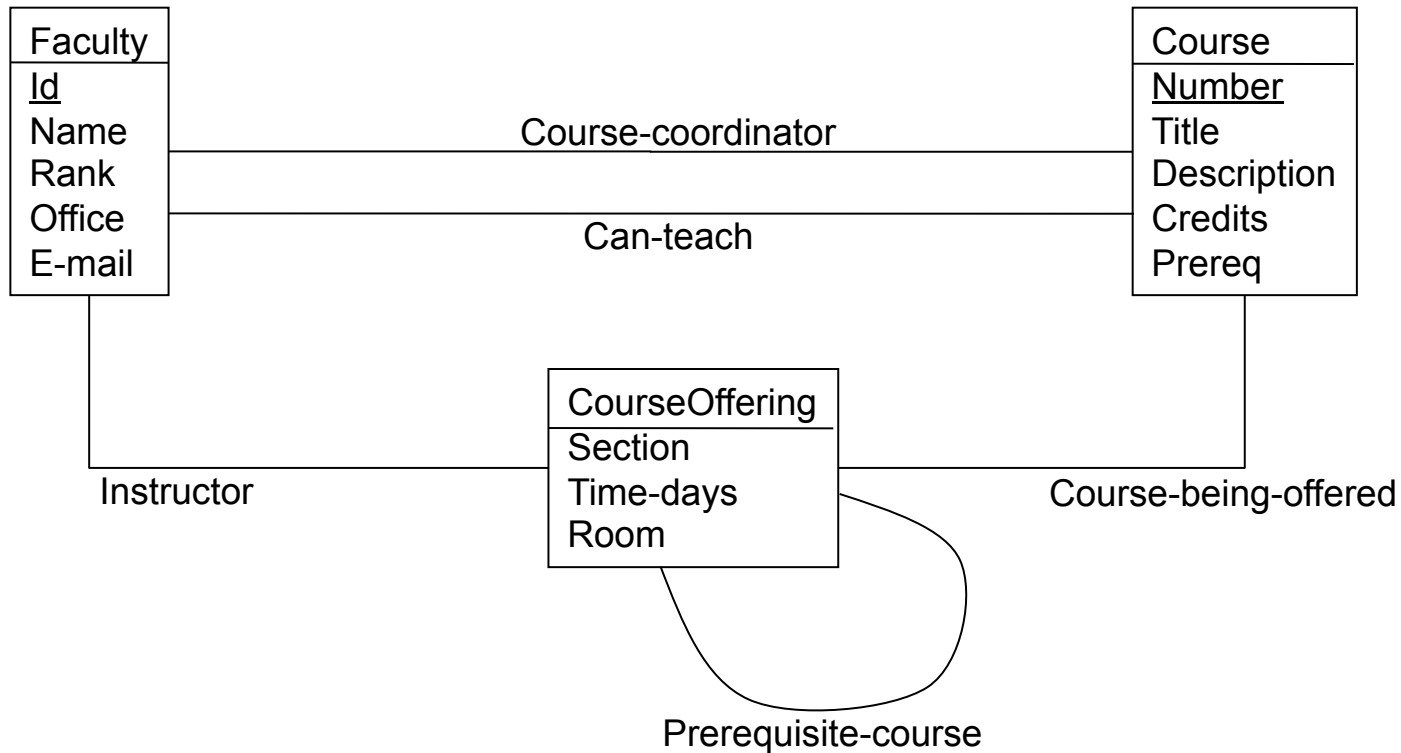


## Exercise: Draw an ERD

Let's draw an ERD for a database that describes faculty, courses, and course offerings. Faculty can serve as the "course coordinator" for a course. Faculty can be qualified to teach a course. Courses can have other courses as prerequisites. One or more faculty can teach each course offering.

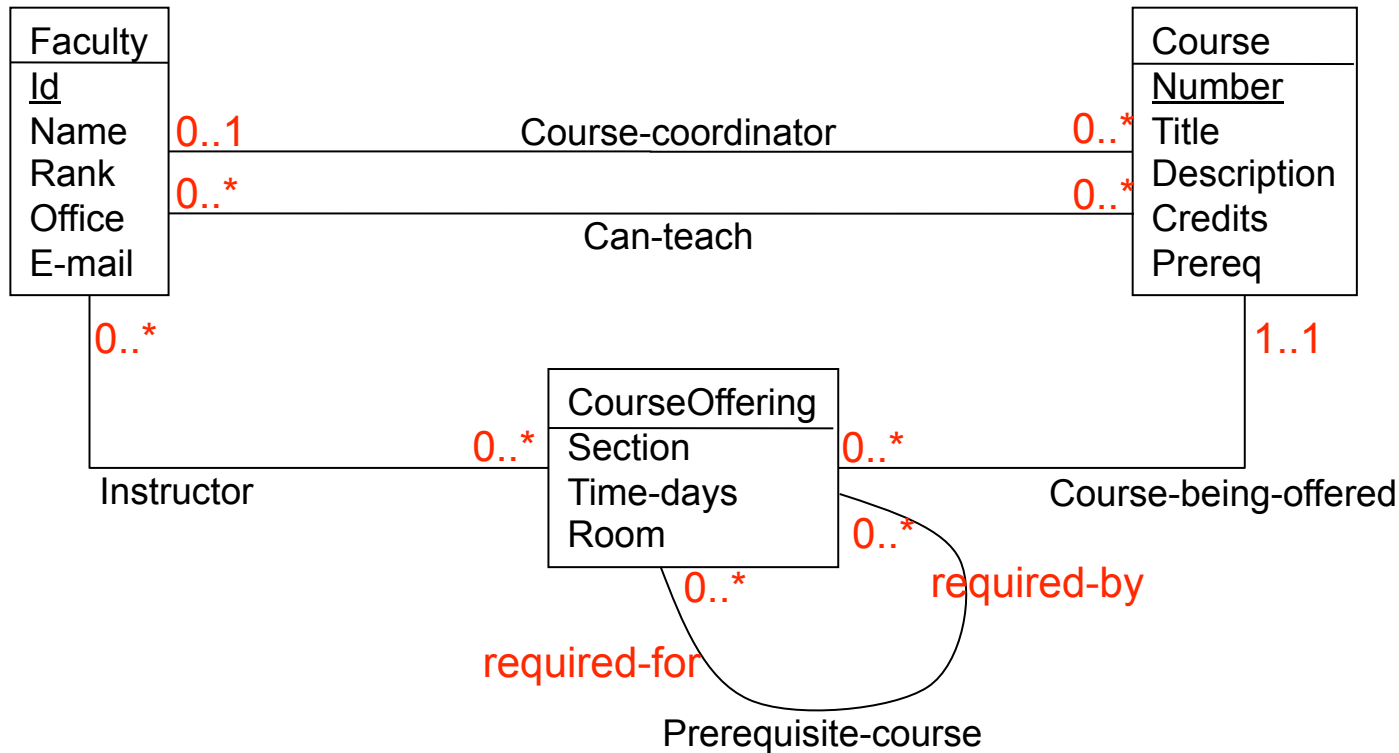
# One Possible ERD

(note: this ERD doesn't have all the details yet)

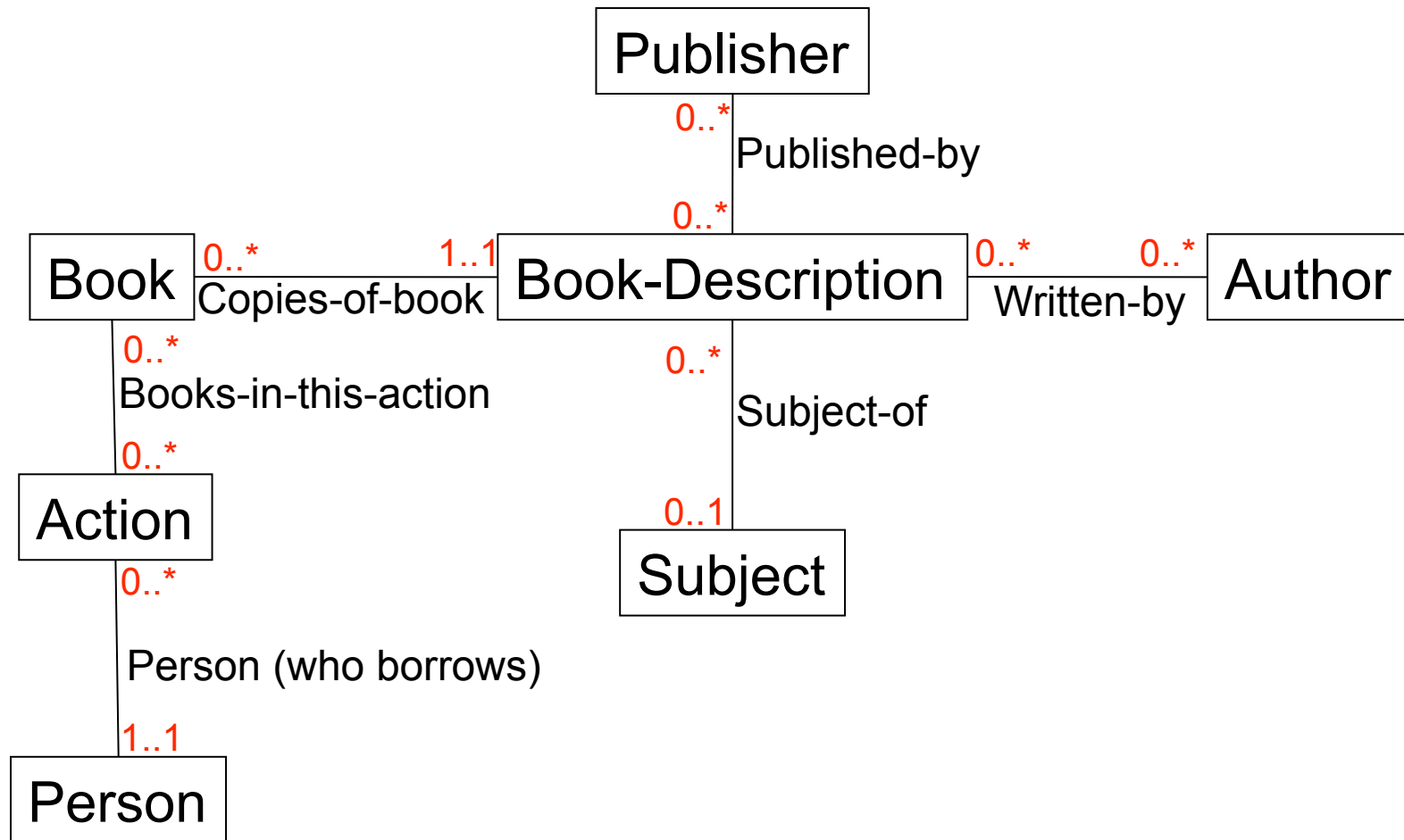


# Sample ERD

Let's fill in cardinalities and role names.



# Let's look at another ERD: Library DB



# Here's the schema for Library DB

Action (transactionid, **personid**, borrowdate, duedate, returndate)

Author(authorid, firstname, middlename, lastname)

Book (bookid, **bookdescid**)

Book\_description (bookdescid, title, subtitle, edition, voltitle, volnumber, language, place, year, isbn, dewey, **subjectid**)

Borrowrel (**transactionid**, bookid)

Person (personid, firstname, middlename, lastname, parentname, address, city, zipcode, phonenumber, emailaddress, property, studentno, idcardno)

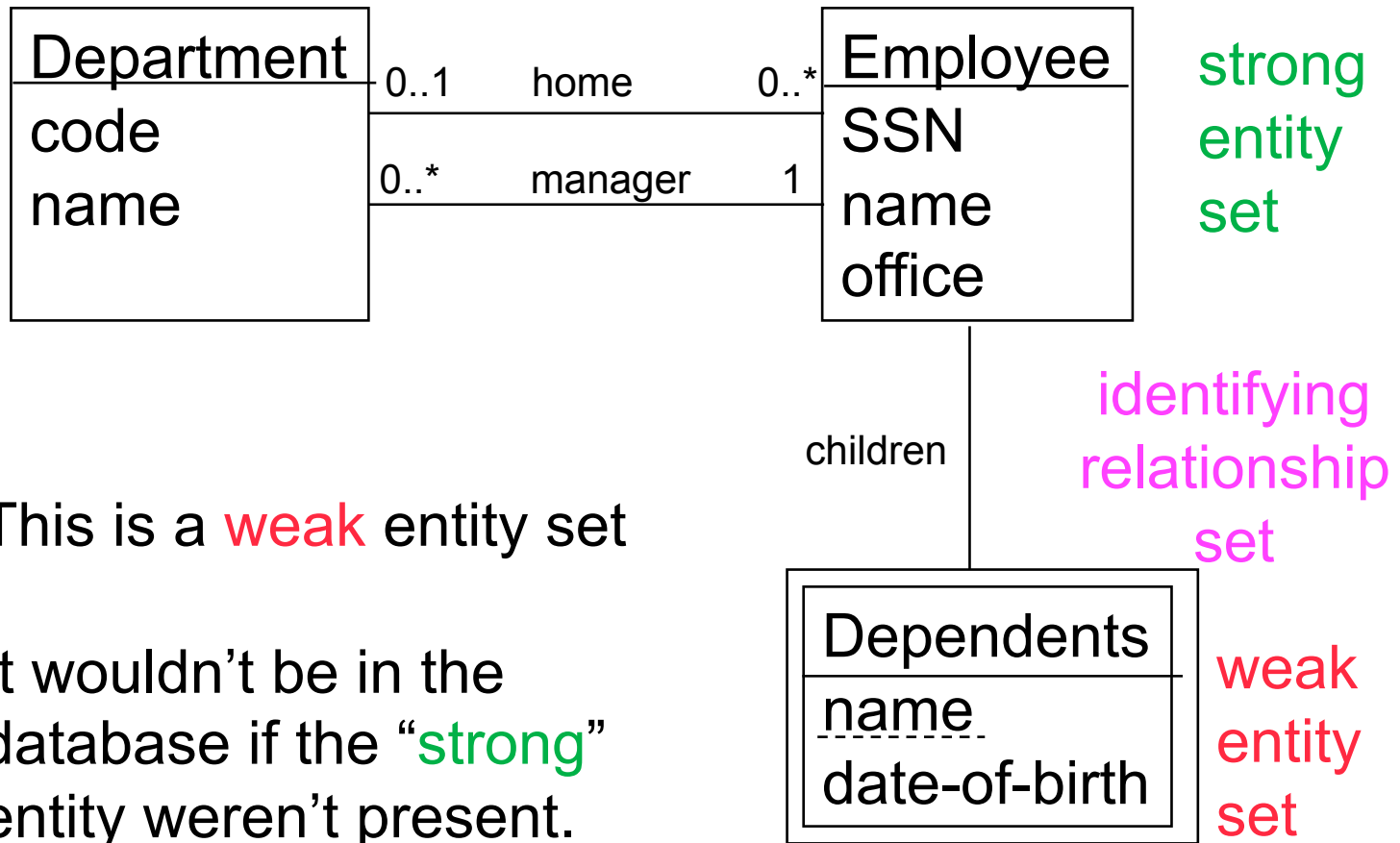
Publisher (publisherid, publisherfullname)

Relauth (**bookdescid**, authorid, role)

Relpub (**bookdescid**, publisherid, role)

Subject (subjectid, subjecttype)

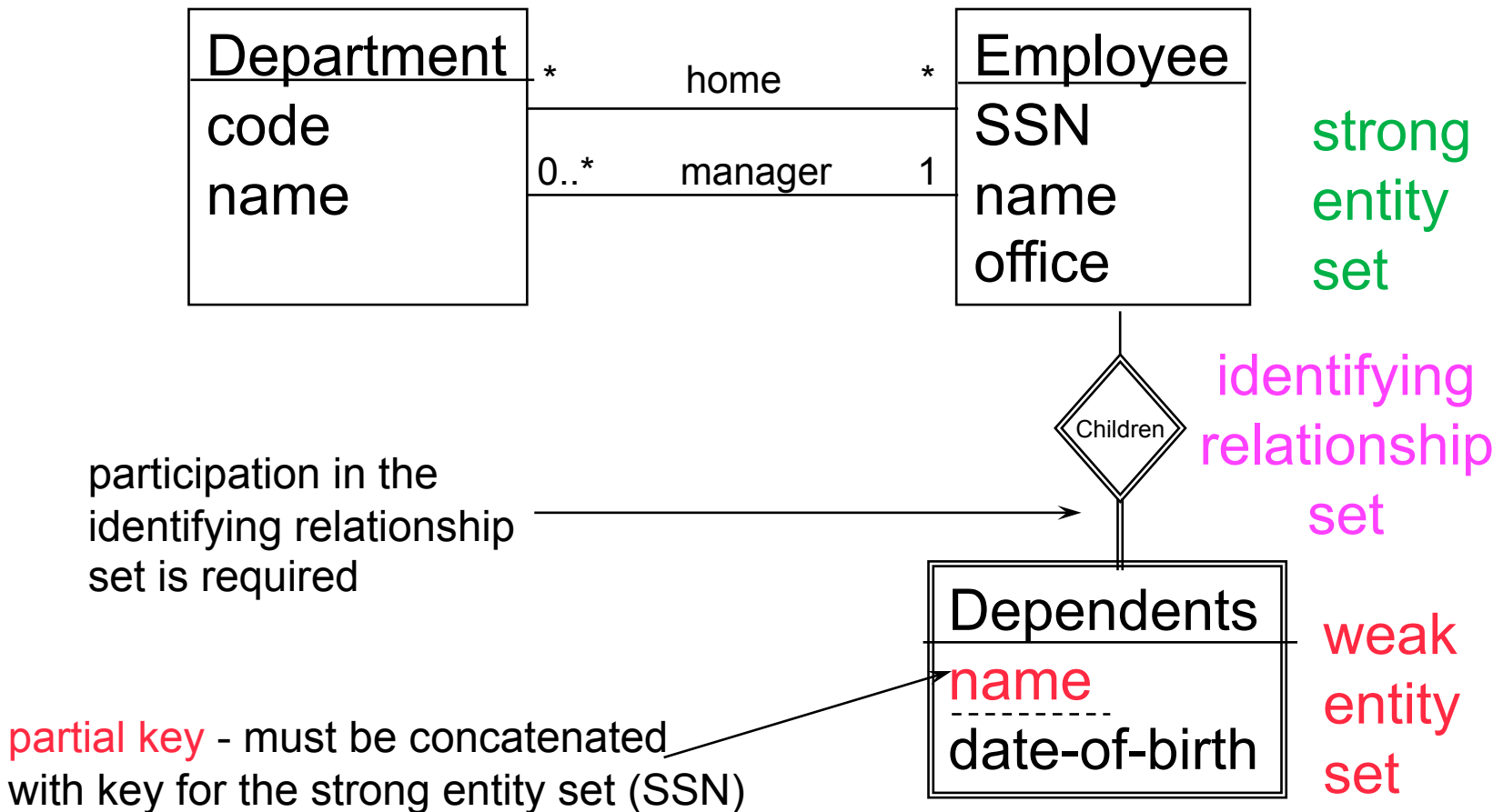
# Weak Entity Sets and Identifying Relationship sets



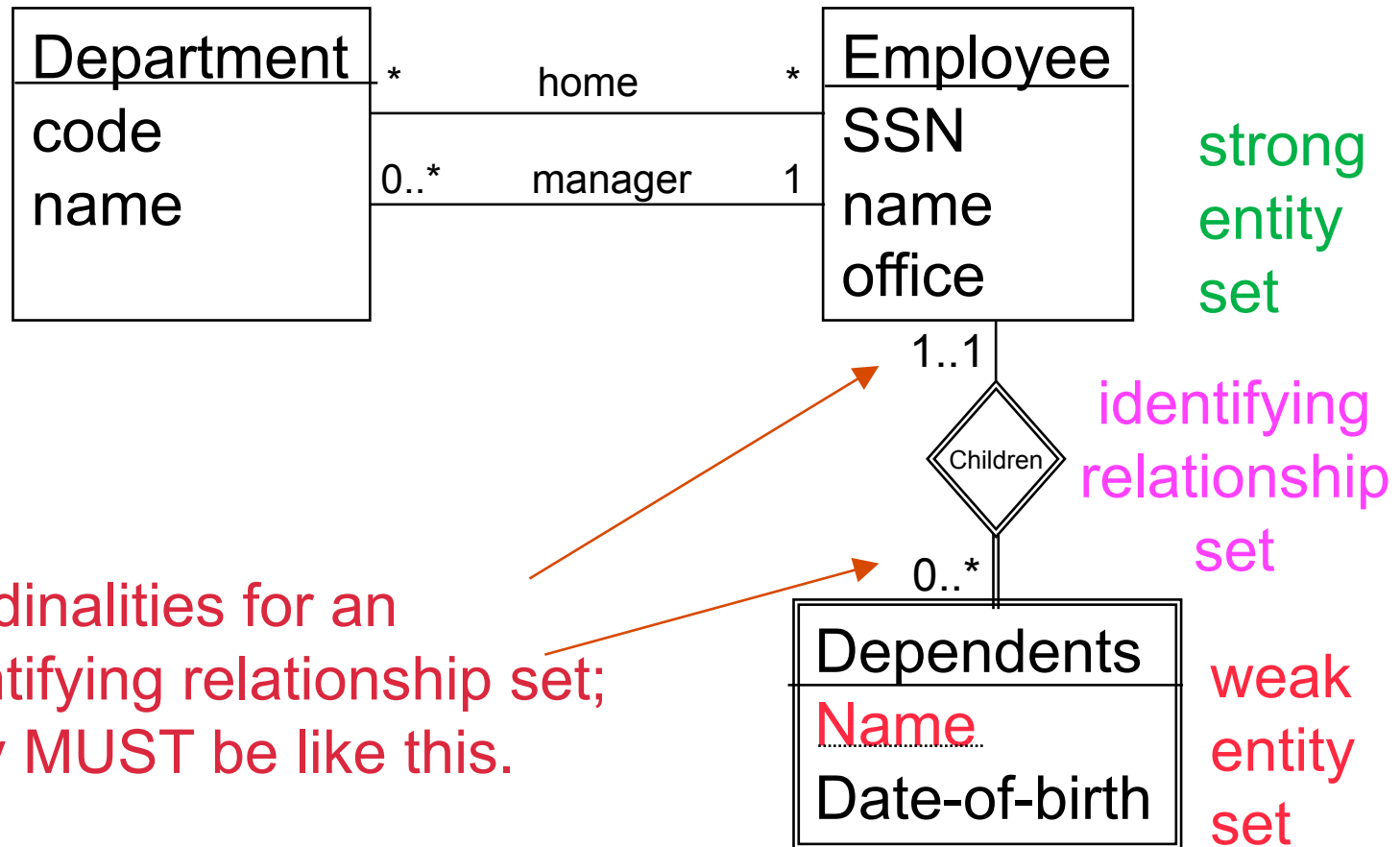
This is a **weak** entity set

It wouldn't be in the database if the "strong" entity weren't present.

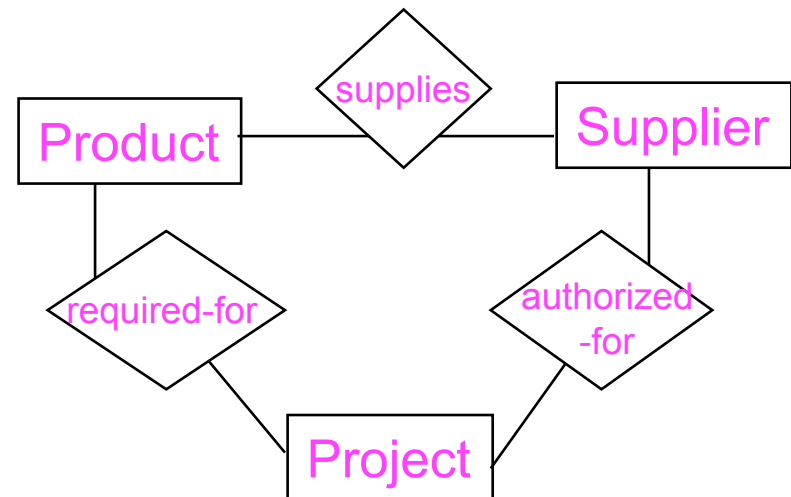
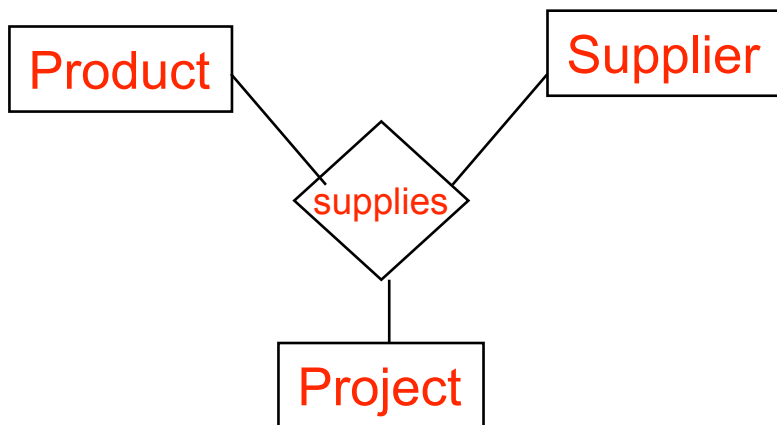
# Weak Entity Sets and Identifying Relationship sets: Alternative Notation



# Weak Entity Sets and Identifying Relationship sets: Alternative Notation (cont.)



# Ternary vs. Binary Relationship sets



These two schemas are not equivalent!

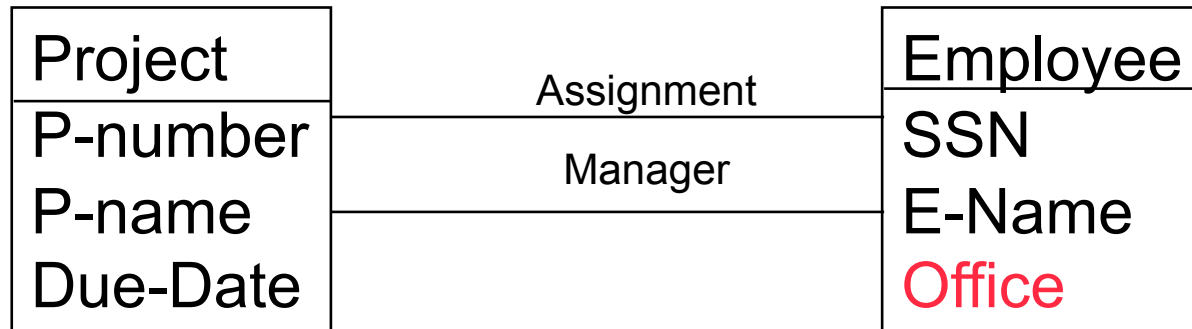
When would we use a ternary relationship set?

When would we use three binary relationship sets?

## Binary vs. Ternary Relationship sets (Cont.)

- The ternary relationship set means that a Supplier must be authorized to supply a particular part to a particular project. e.g., Office-Depot can supply laser printer paper to project 112. Office-Max can supply paper clips to Project 112. Office-Max can supply pencils to project 115. (But based on that much information, Office-Max can't supply pencils to 112.)
- The three binary relationship sets each represent something distinct. A Supplier can be authorized to supply certain products (Office-Max can supply pencils). A Project can require certain products (112 needs pencils). And a Supplier can be authorized to supply a certain project. (Office-Max supplies 112)  
Therefore: Office-Max can supply pencils to 112.

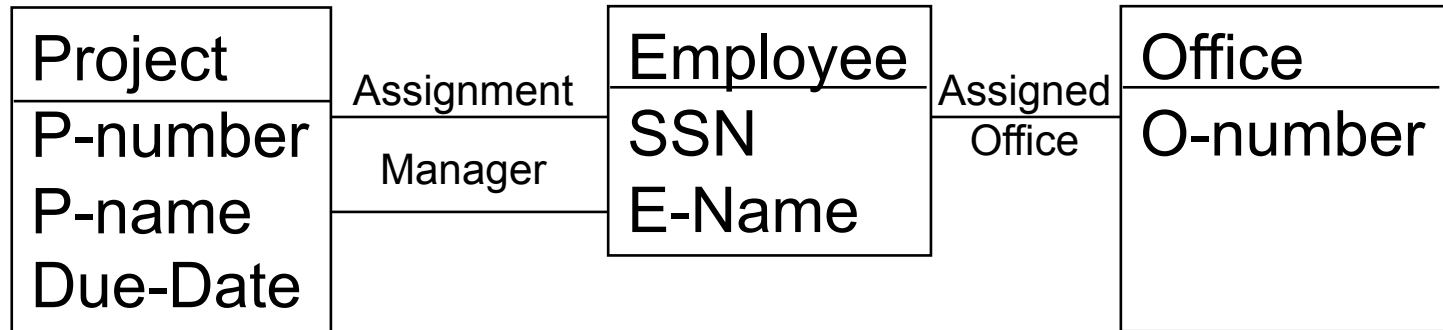
Duality: **entity** and **relationship** ↔ **attribute** ↔ **value**



Should **Office** be an **attribute of Employee**? or a **separate entity set**? Most attributes can be “promoted” to an entity set and some entities can be “demoted” to an attribute value.

This explains why there are so many different ways to design a schema.

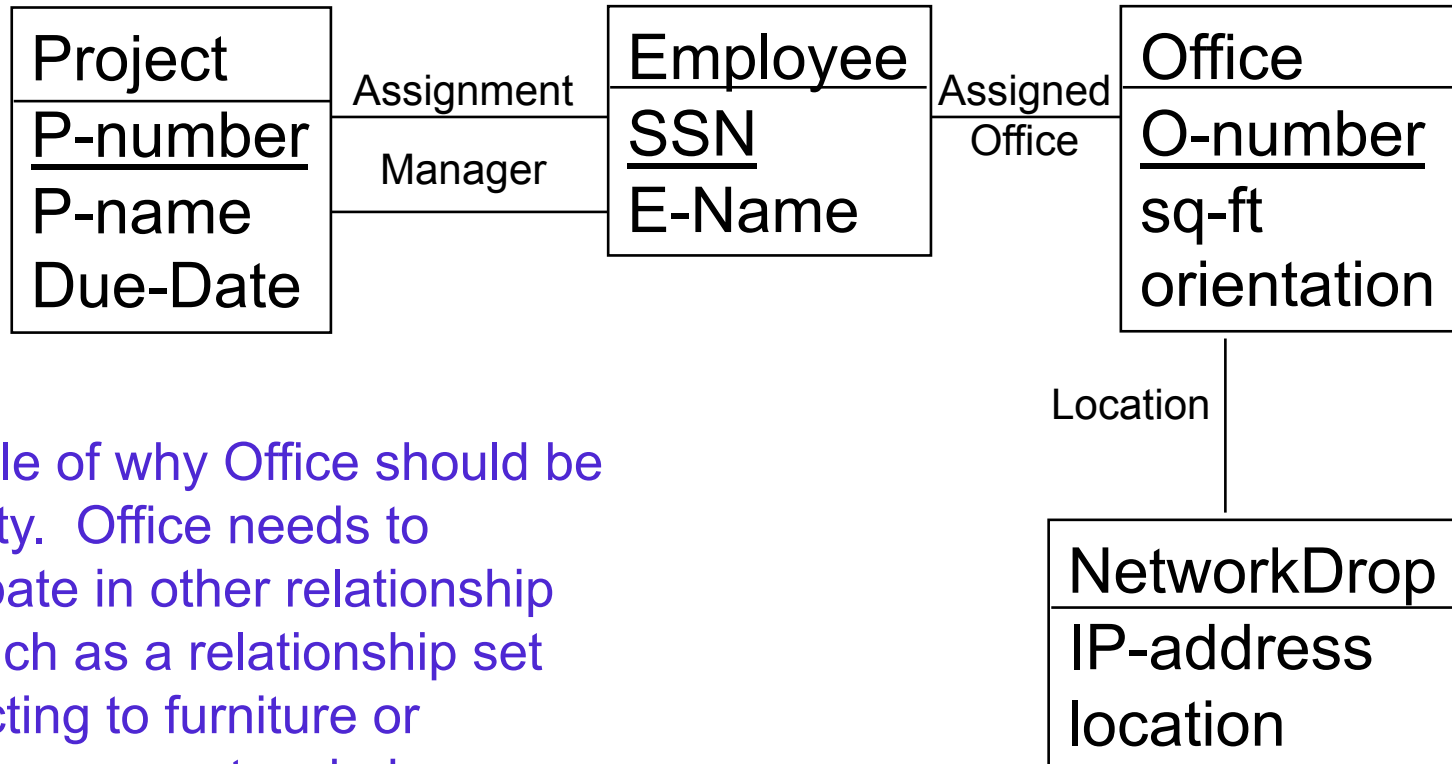
# Entity vs. Value of an Attribute



What are some reasons to model Office as an entity set?

- an employee can have more than one office
- there are other attributes of Office
- Office needs to participate in other relationship sets such as a relationship set connecting to furniture or telephones or network drops (located in the office)

# Entity vs. Value of an Attribute (cont.)

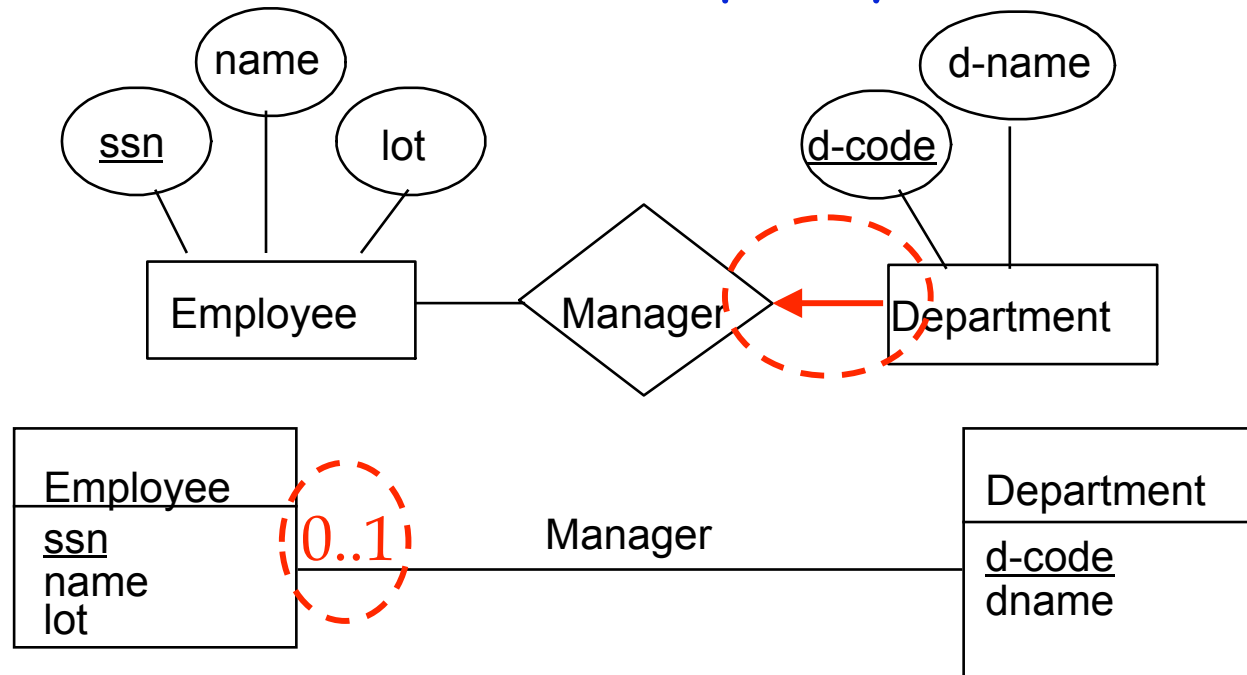


Example of why Office should be an entity. Office needs to participate in other relationship sets such as a relationship set connecting to furniture or telephones or network drops (located in the office)

# Key Constraints - as described in the book

(limiting participation in relationship set to at most 1 entity)

same as maximum multiplicity of 1 in UML



The red parts of these diagrams have the same meaning.

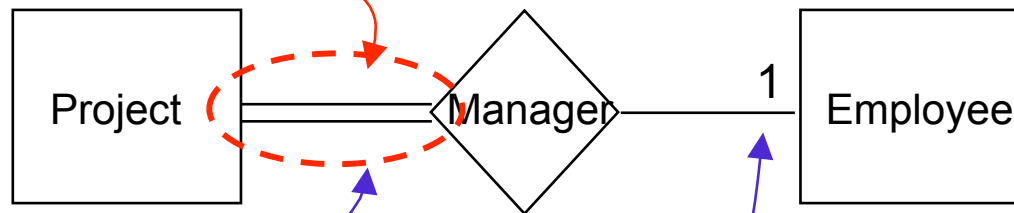
Each dept has at most one manager, according to the *key constraint* on Manages.

**Participation Constraint - as in text:** when every entity **MUST** participate in a relationship set

**a Project has exactly one manager**



The red parts of these diagrams have the same meaning.



**a Project MUST have a manager  
and there is at most 1 employee who is manager**

Note ERDs and UML Diagrams can be at two levels:

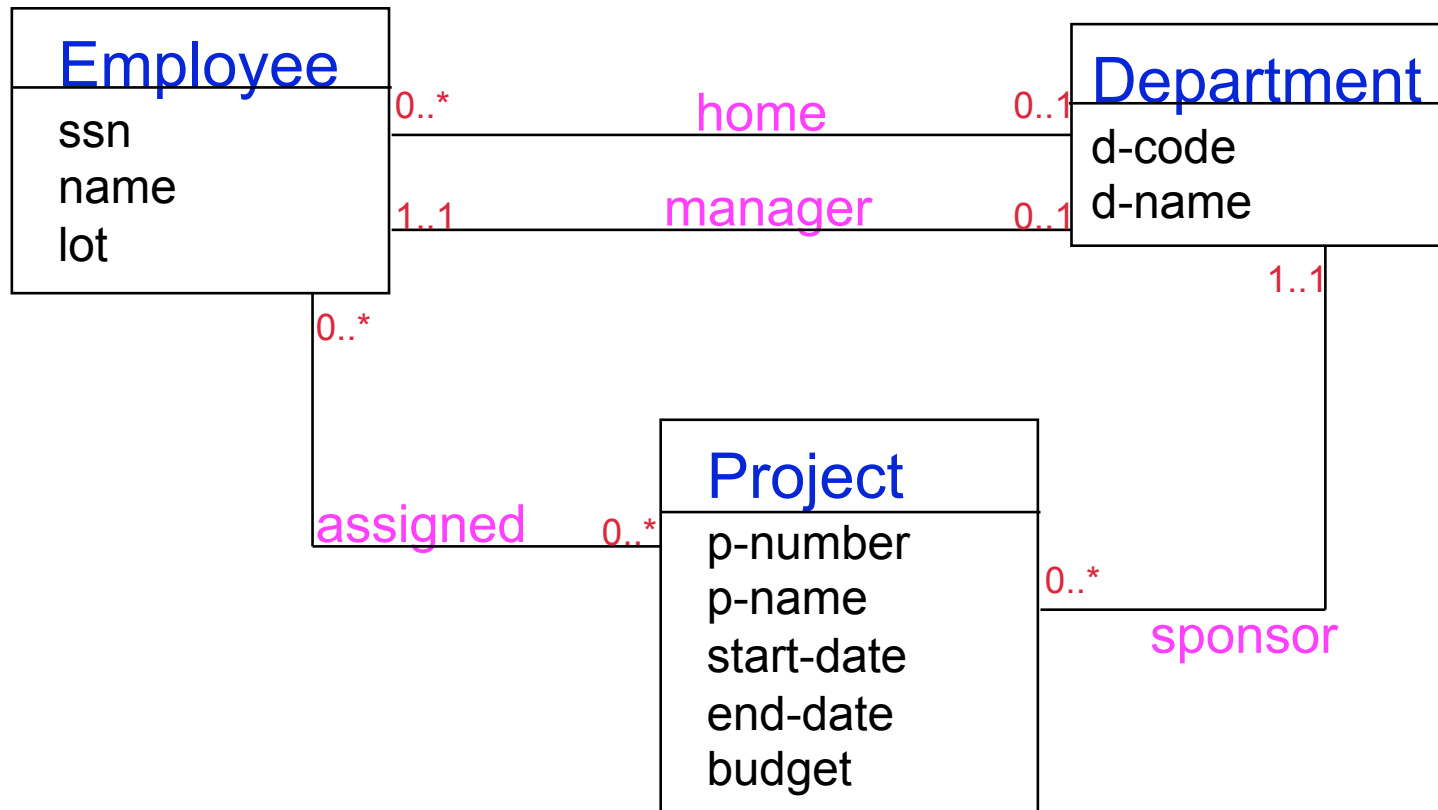
the ERD/UML level

and

the Relational Table level.

The difference is primarily with the many-to-many relationship sets.

# Entity-Relationship Diagram



# Equivalent Relational Schema - with foreign keys shown

Employee (ssn, name, lot, home-dept)

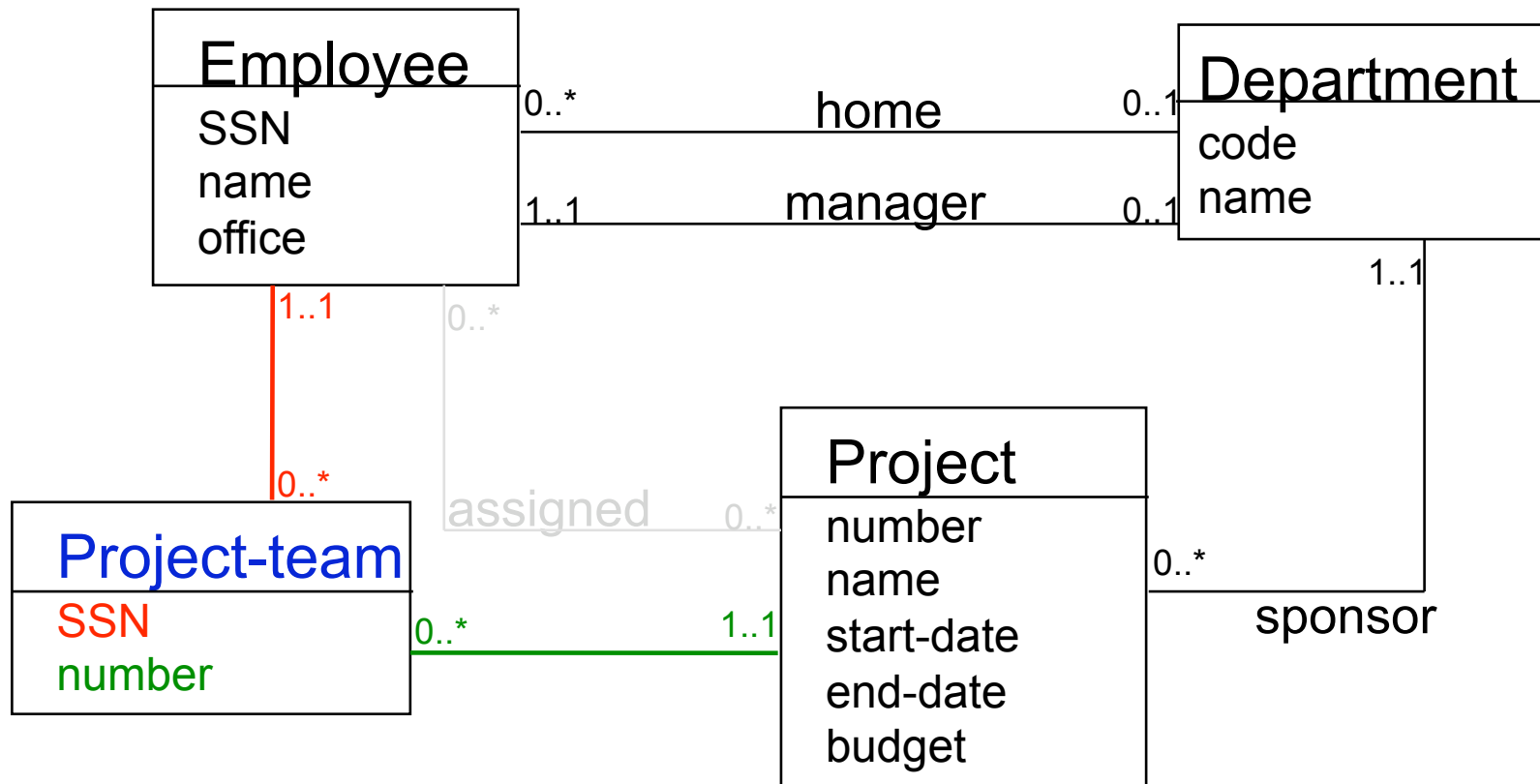
Project-team(ssn, number)

Department (id, name, manager)

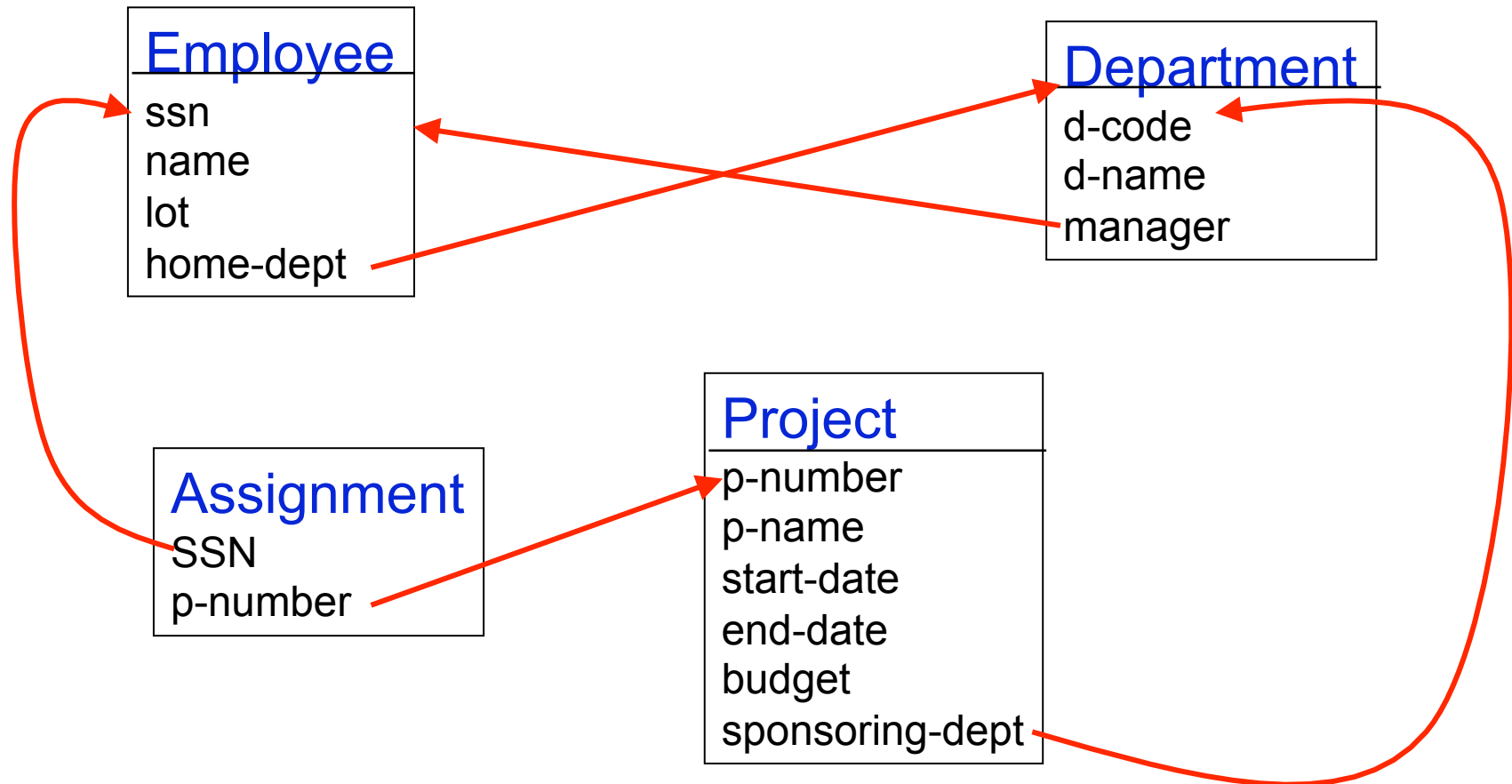
Project (number, name, start-date, end-date, budget, sponsor)

Notice that the many-to-many relationship set must be represented in a (new) table.

# Table-level Schema in ERD Syntax



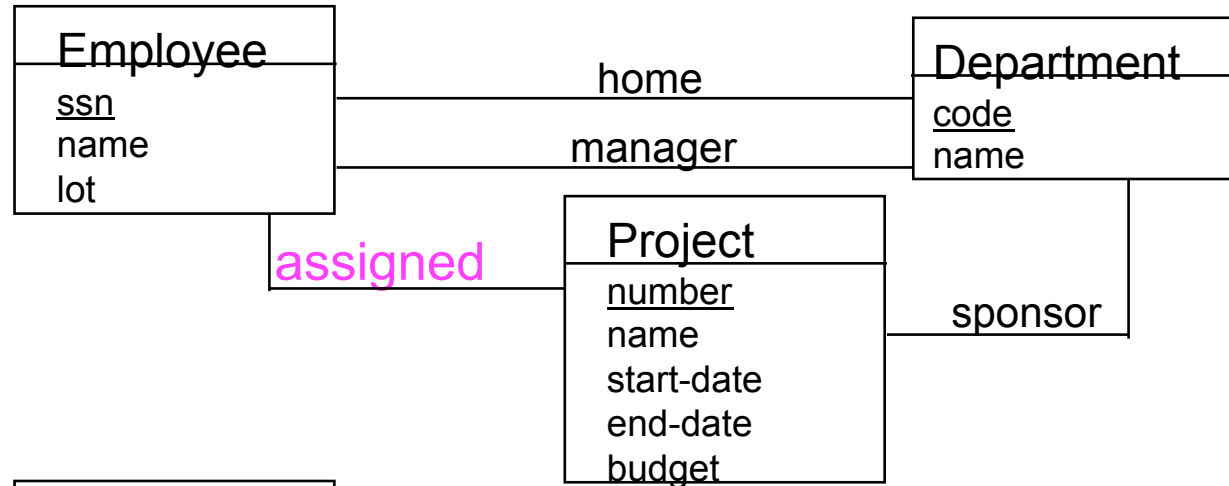
# Equivalent Relational Schema



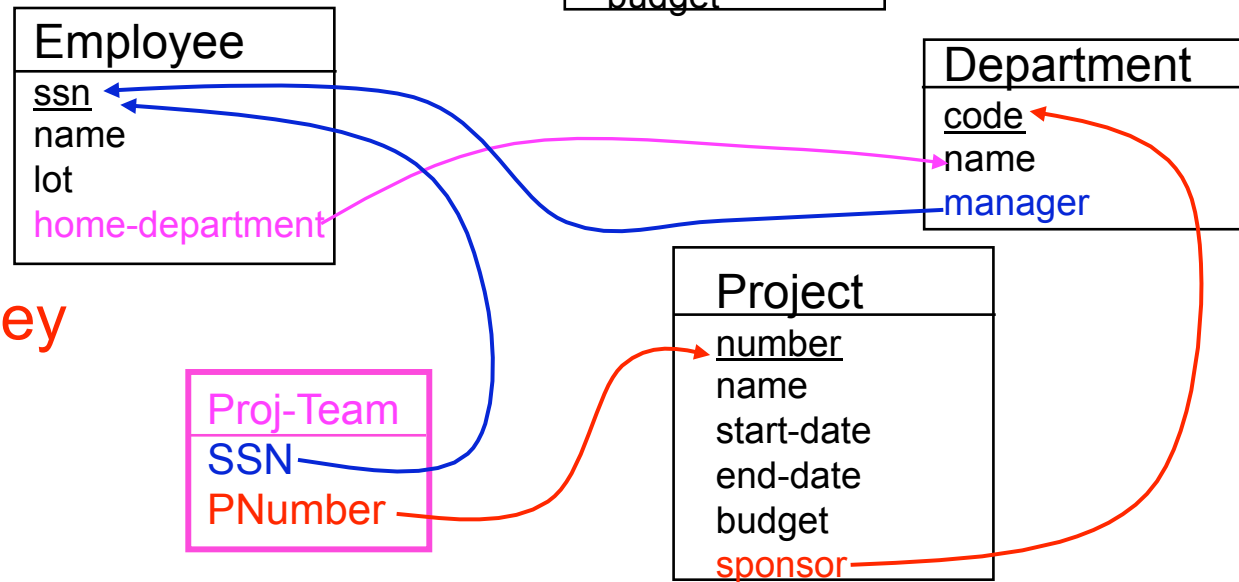
Notice that the relationship sets shown in this diagram aren't really needed. *foreign keys* reference other tables.

# ERD compared to Relationship Diagram

ERD

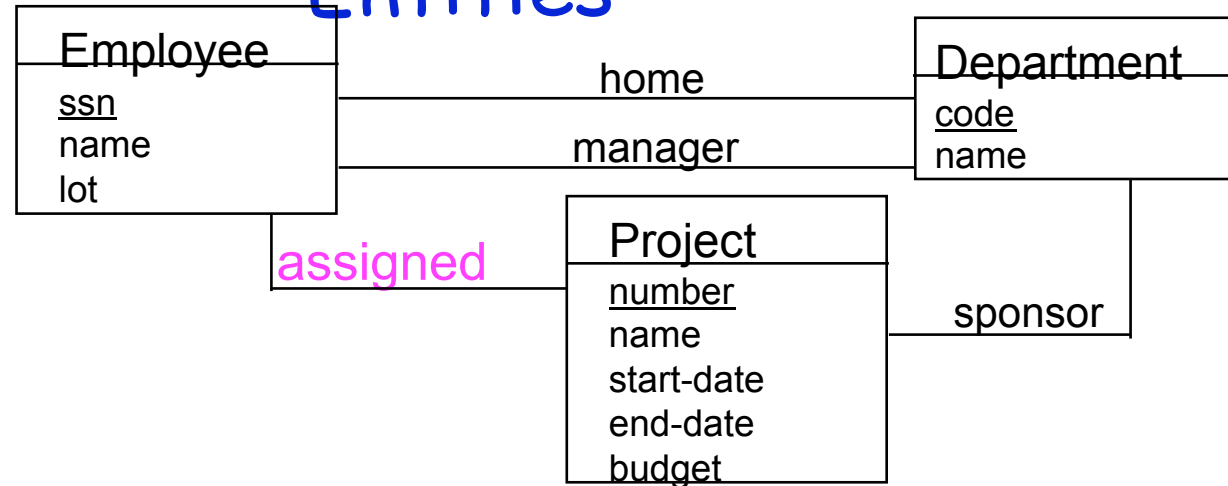


Relational  
DB Diagram:  
box = table  
arrow = foreign key  
colorful parts are  
new compared  
to ERD

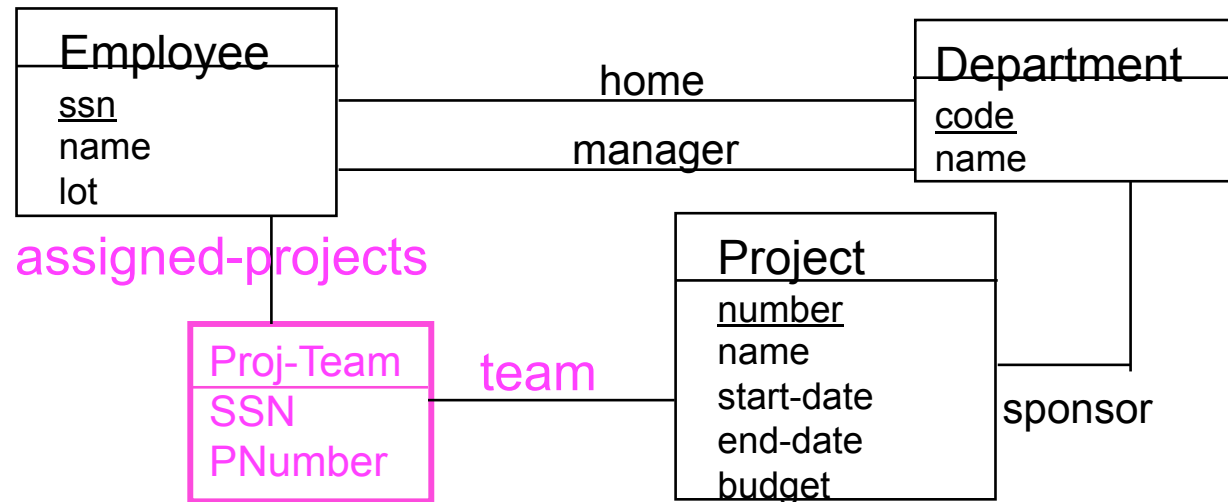


# Relationships CAN be represented as Entities

ERD

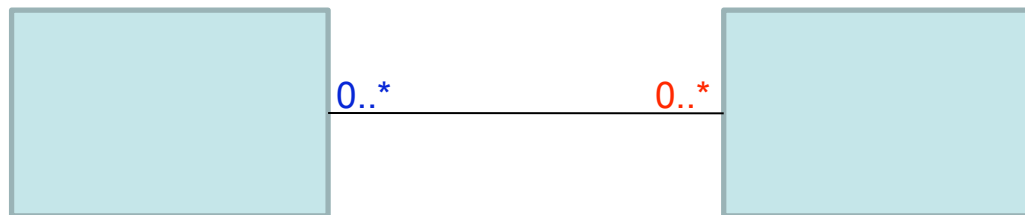


Project-Team is an entity with two new relationships



# Template for "many-to-many" relationships

A many-to-many relationship:



Becomes a new entity plus two relationships:



Notice the cardinalities on the two relationships.

# Overview of Database Design

- **Conceptual design:**
  - ER Model is used for conceptual design.
  - What are the **entities** and **relationships** we need?
- **Logical design:**
  - Transform ER diagram to Relational Schema
- **Schema Refinement: (Normalization)**
  - Check relational schema for redundancies and related anomalies. Modify schema accordingly.
- **Physical Database Design and Tuning:**
  - Consider typical workloads; (sometimes) modify the database design; select file types and indexes.

# Entity-Relationship Model is a different data model than the Relational Model

- **Relational model** has:
  - **tables** (relations) with attributes, keys, foreign keys, domain definitions for attributes
- **Entity-Relationship model** has:
  - **Entities and entity sets** with attributes, keys, and domain definitions for attributes
  - **relationships among entities and relationship sets** with cardinality constraints (in the book they describe key constraints and participation constraints for relationship sets)

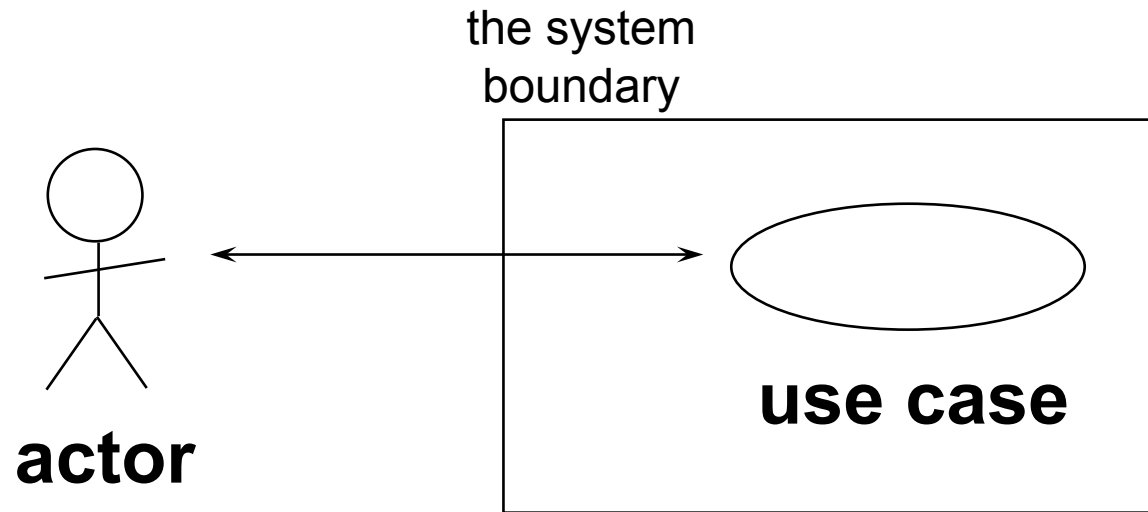
# Introduction to Use Cases

Results from collaboration with  
Drs. Earl Ecklund and Lois Delcambre

Use Cases invented by Dr. Ivar  
Jacobson

---

# Use Case Model



outside the system

the system

nondeterministic  
not described in detail

sequence of events  
that form a dialogue

---

# Setting the Scope of the “System”

Identify what’s inside the “System”

ATM  
machine for  
a bank

or is it a network of ATMs for a bank?

or is it a distributed, full-service bank system?

---

# Identify all Actors for Use Cases

**actor** - anything that needs to exchange information with the system. actor can be a system.

One user or user group may play multiple roles; thus you may need to define multiple actors.

Describe the purpose for each actor (role)

---

# Possible Kinds of Actors

Prototypical user (plays a certain role)

External systems

Scripted usage

- system startup

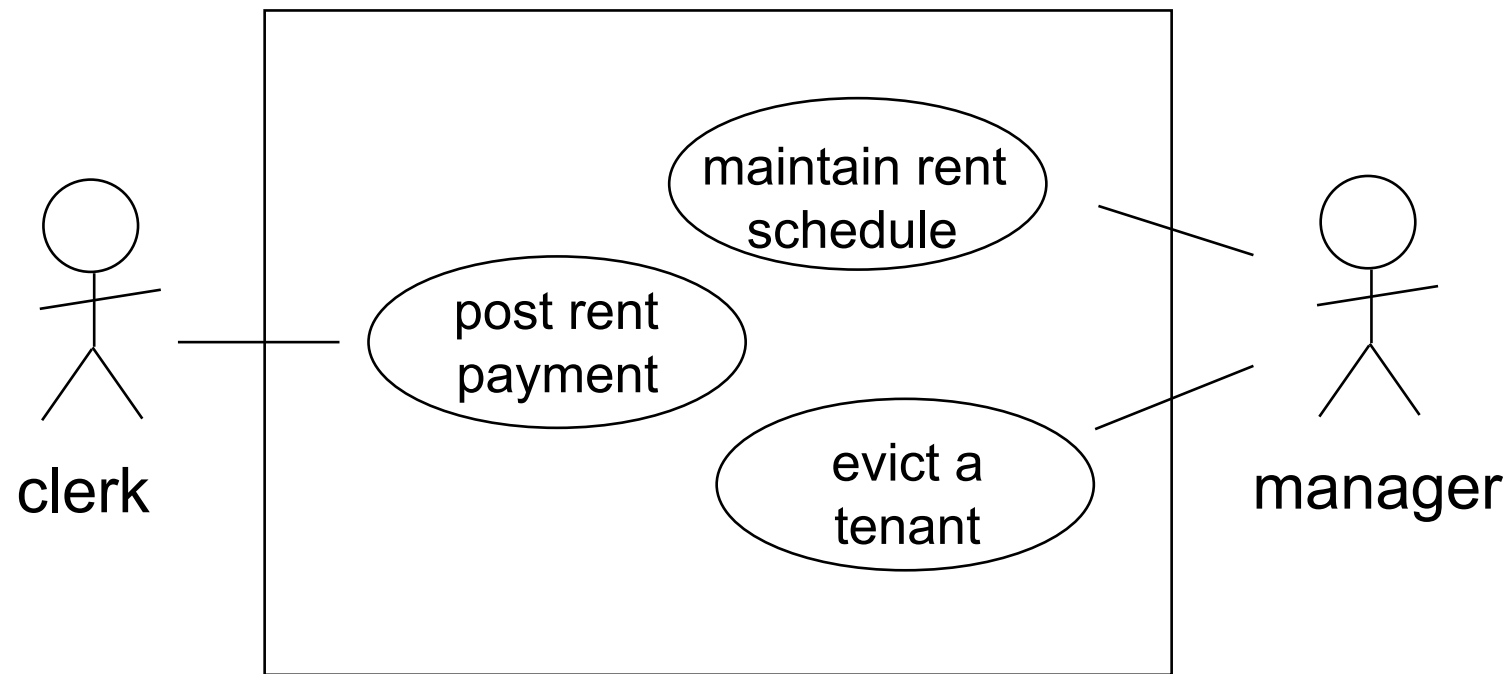
- operations

- systems maintenance

- termination

---

# Tenant System Use Case Model

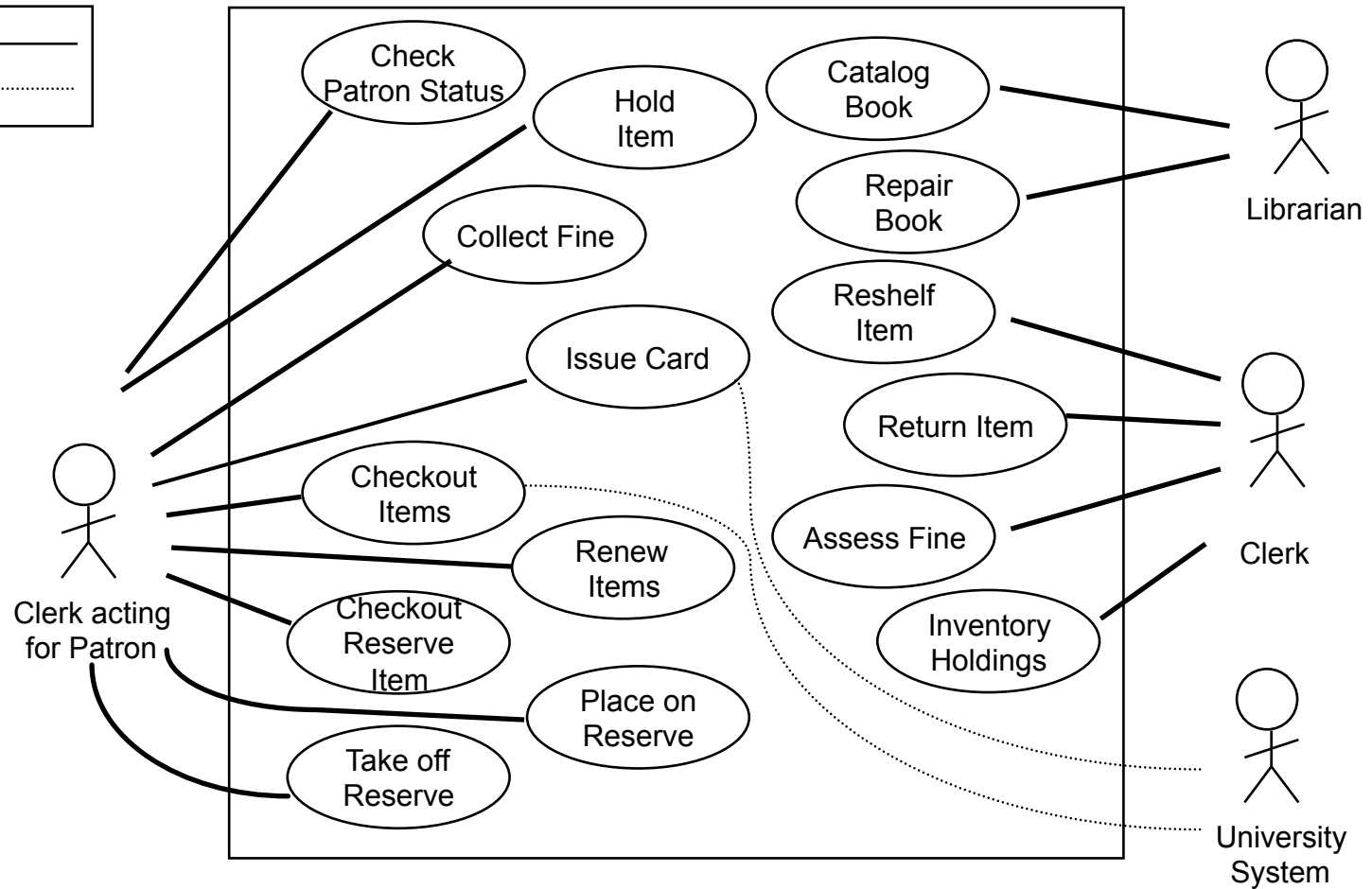
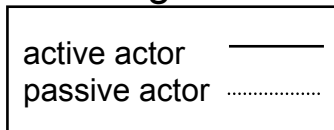


Two actors and three “use cases” have been identified so far, each use case just has a name

---

# Library Use Case Model

## Legend



## A Use Case is:

- user-visible
  - user-meaningful
  - easy for a user to confirm (or change)
-

# Questions to help discover use cases:

- what are the main tasks of each actor?
  - will the actor read/write/change system information?
  - does the actor inform the system of outside changes?
  - does the actor wish to be informed of unexpected changes?
  - is this an *active* or a *passive* actor?
-

# Example Use Cases

For the Grader System

Actor: Registrar

1. Define course offerings - Enter information (dept, number, name and instructor) for each course offering to be held in the current term.
  2. Manage course offering roster (initial/add/drop) - Initially load the list of all students enrolled in each course offering; then add or drop students, one at a time, to a course offering with existing enrollment.
-

# Sketch the User Interface

to encourage brainstorming about the system  
(should be user interface technology-neutral)

Delegate Functions to Grader

Course **CSE 504**      Instructor(s): **Delcambre/Ecklund**

Graders	Delegated Functions	Functions
Smith, J. <span style="background-color: yellow; border: 1px solid black; padding: 2px;">Jones, K.</span>	manage teams	define WI . . .

ok cancel

---

# How much detail should we put in a use case?

- *should we find all use cases?*  
80-20 rule.....try to get the most important use cases. try to get a range of use cases (e.g., several from each actor) initially
  - *should we write all exceptions?*  
not necessarily, especially early on
  - *should an exception be a use case in its own right?*  
if the behavior is significant - write another use case....otherwise simply list at the bottom
-

# Use Case Model in Two Styles

- Original use cases

each scenario = one meaningful interaction  
no concern about relationship among use cases

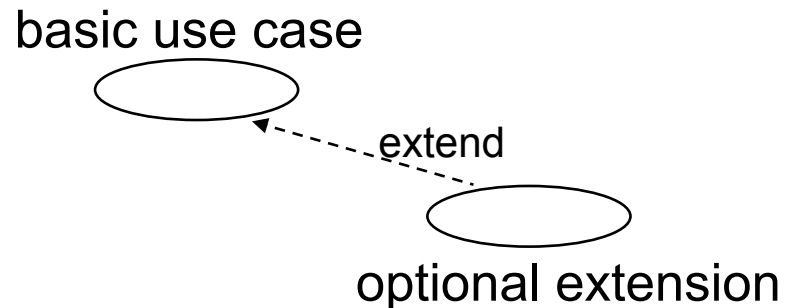
- Refined Use Case Model

identify common parts; connect the use cases with  
`<<include>>` and `<<extend>>` links

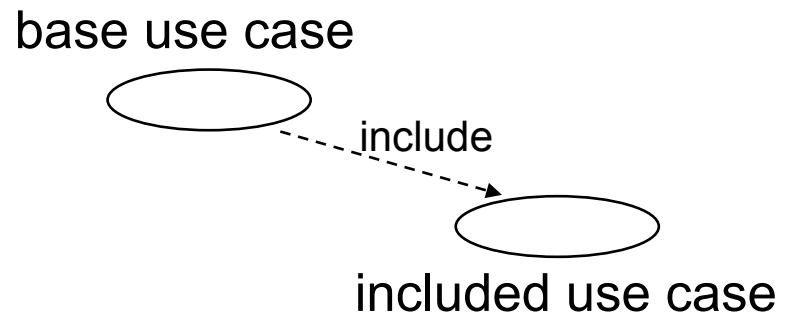
---

# Refined Use Case Diagram

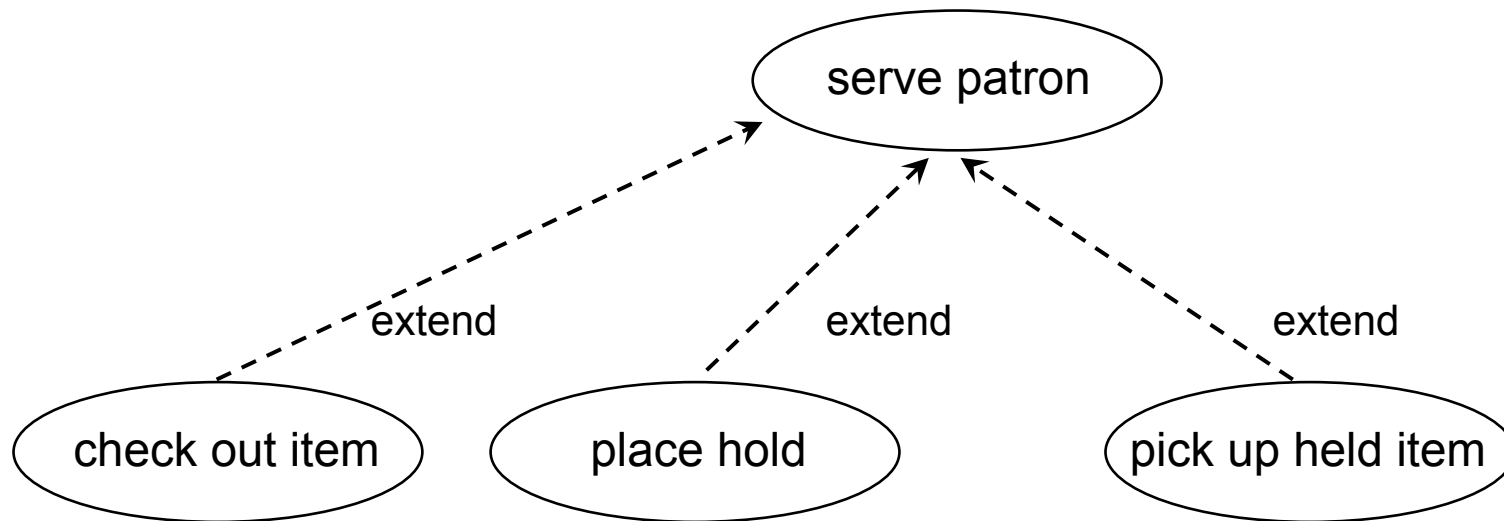
UML notation:  
when a use case is  
optionally invoked  
(a use case *extension*  
needs *insertion point*)



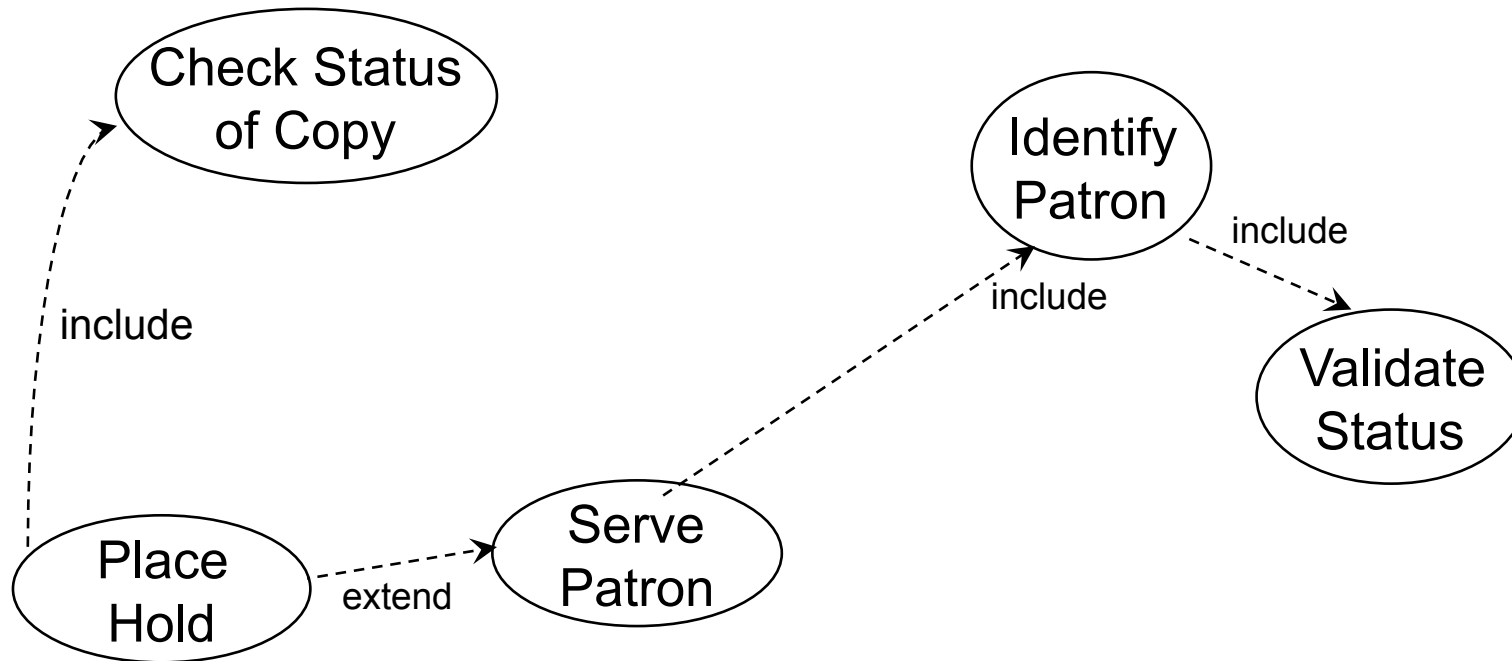
UML notation:  
when a use case is  
always invoked, like  
a procedure  
(an *included* use case)



# Example of Use Case Extension



# Example of Included Use Case





## Example Use Case:

## Find Course

### Supervisor

**Actor:** Student who wants to find course supervisor

1. Student logs into Banner
  - a. If login is invalid, issue **error** message and return to step 1
2. Student chooses "**find** course supervisor" from menu.
3. Student **inserts** quarter and CRN from dropdown list.
4. System **returns** course supervisor information.
5. Student chooses "**exit**" from menu.

## Notes about Use Cases

- Each use case **must have a name and an actor**. The actor may be another system, for example if another company is purchasing an item.
- Each step of the use case is **doable** by the actor or the system you have built.
- Error cases can be handled inline, as in the example, or at the end of the use case.
- In order to find uses cases, think of who users are, what they will do, and/or what screens will look like.
- Use the given format: start with the actor and number the steps.