Lecture 4

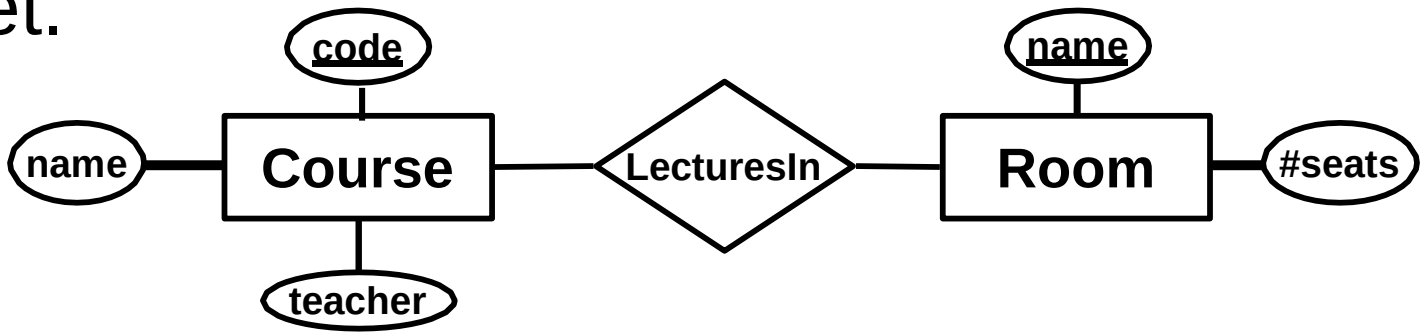
Database design

Cardinality

CARDINALITY

Many-to-many relationships

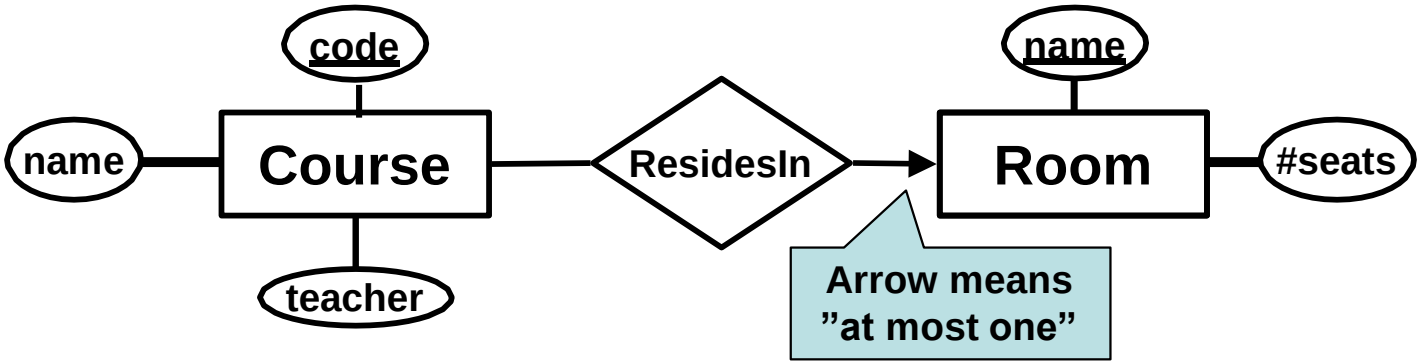
- Many-to-many (n-to-n, many-many) relationships
  - Each entity in either of the entity sets can be related to any number of entities of the other set.



  - A course can have lectures in many rooms.
  - Many courses can have lectures in the same room.

Many-to-one relationships

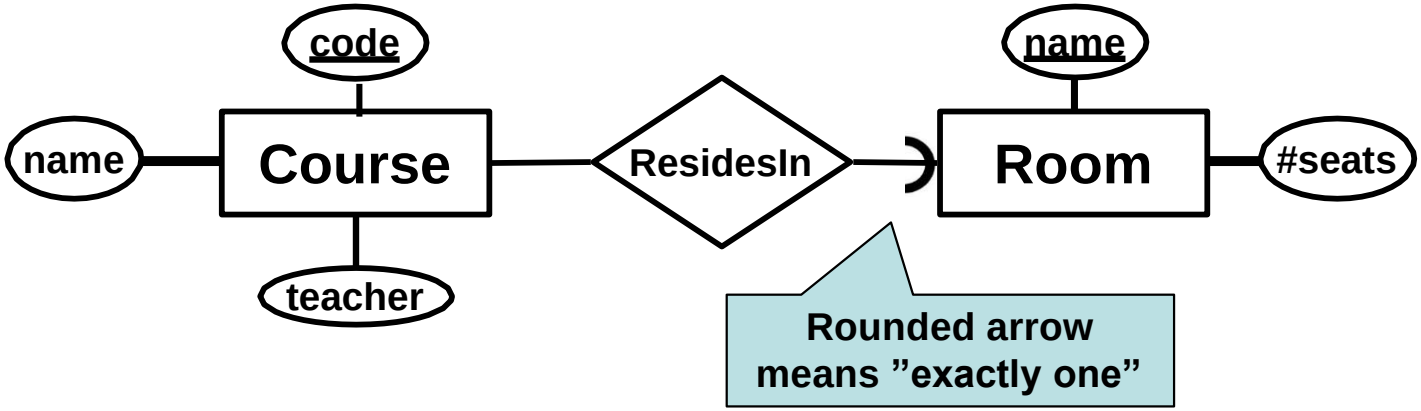
- Many-to-one (n-to-1, many-one) relationships
  - Each entity on the ”many” side can only be related to (at most) one entity on the ”one” side.



  - Courses have all their lectures in the same room.
  - Many courses can share the same room.

Many-to-”exactly one”

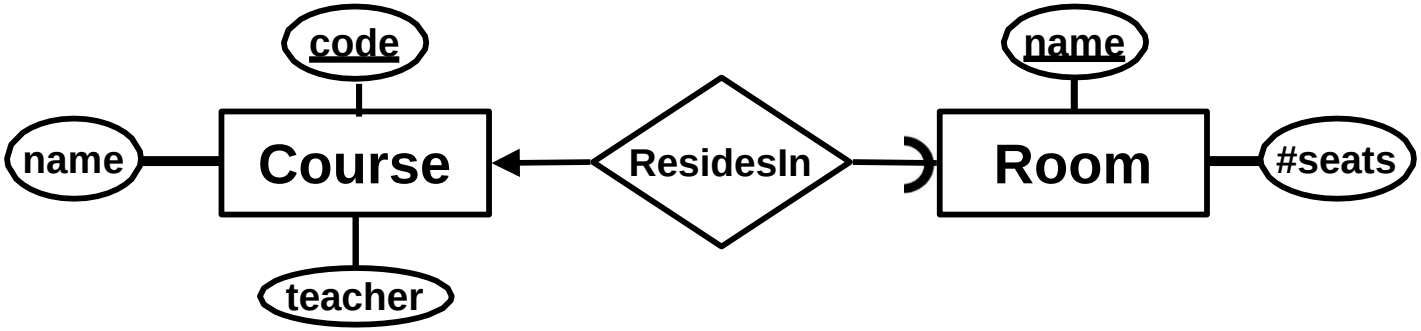
- All entities on the ”many” side *must* be related to one entity on the ”one” side.
  - This is also known as ***total participation***



  - All courses have all their lectures in some room.
  - Many courses can share the same room.

One-to-one relationships

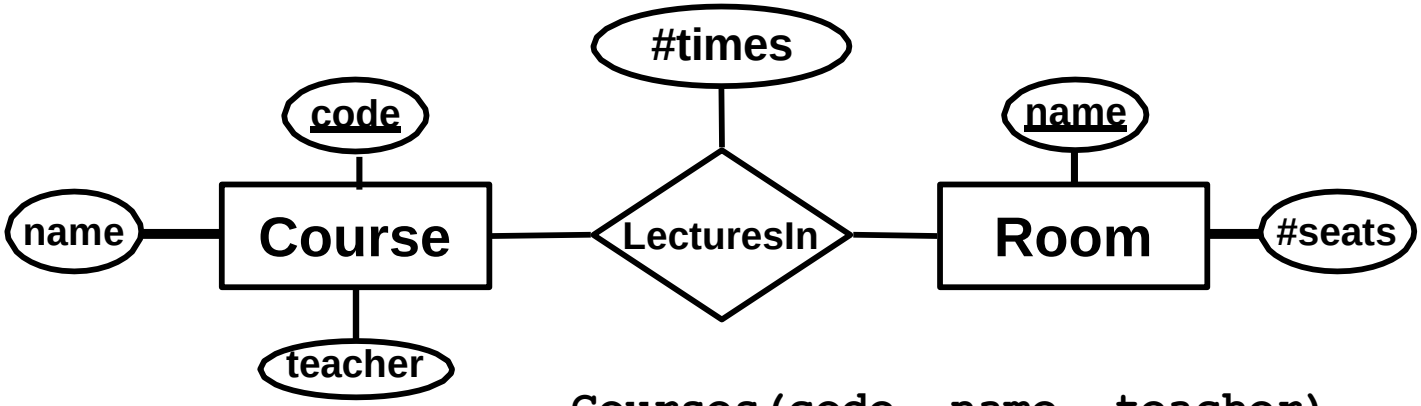
- One-to-one (1-to-1, one-one) relationships
  - Each entity on either side can only be related to (at most) one entity on the other side.



  - Courses have all their lectures in the same room.
  - Only one course in each room.
  - Not all rooms have courses in them.

Translating multiplicity

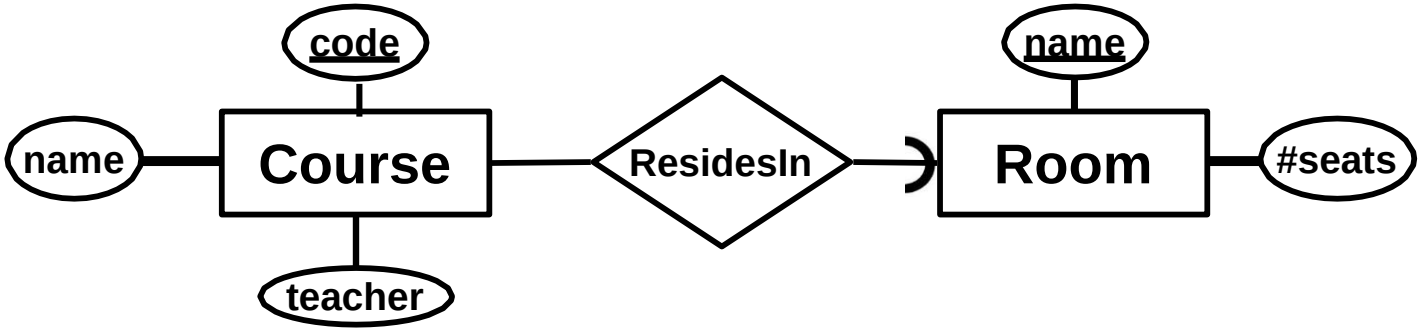
- A *many-to-many* relationship between two entities is translated into a relation, where the attributes are the *keys* of the related entities, and any attributes of the relation.

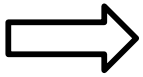



```
Courses(code, name, teacher)
Rooms(name, #seats)
LecturesIn(code, name, #times)
  code  -> Courses.code
  name -> Rooms.name
```

Translating multiplicity

- A *X-to-”exactly one”* relationship between two entities is translated as part of the ”many”-side entity.

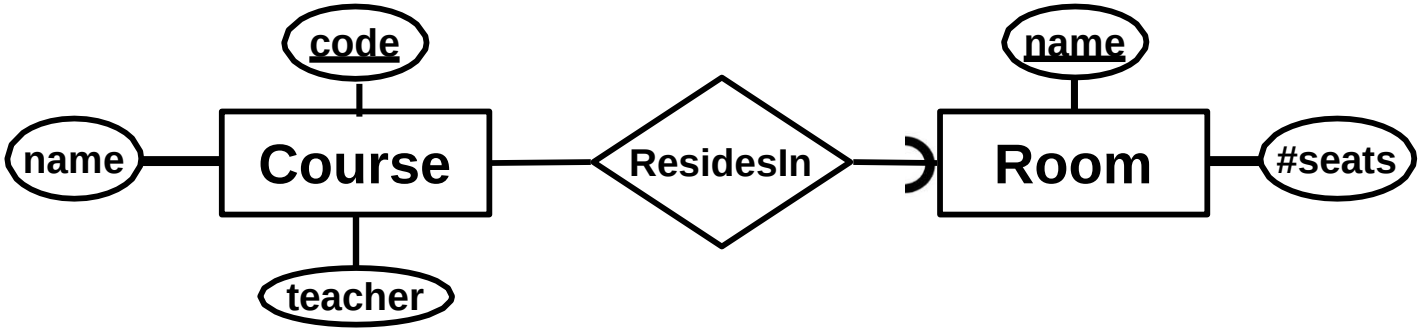




What?

Translating multiplicity

- A *X-to-”exactly one”* relationship between two entities is translated as part of the ”many”-side entity.

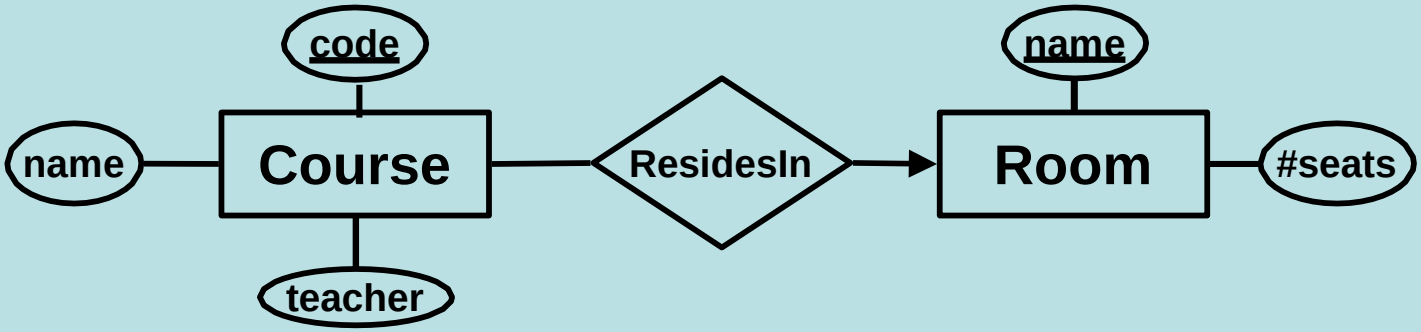


⇨

```
Courses(code, name, teacher, room)
  room -> Rooms.name
Rooms(name, #seats)
```

Quiz

How do we translate an *X-to-one* (meaning ”at most one”) relationship?




```
Courses(code, name, teacher, room)
Room(name, #seats)
```

or

```
Courses(code, name, teacher)
Room(name, #seats)
ResidesIn(code, room)
```

?

Aside: the NULL symbol

- Special symbol NULL means either
  - we have no value, or
  - we don’t know the value

- Use with care!
  - Comparisons and other operations won’t work.
  - May take up unnecessary space.

Translation comparison

```
Courses(code, name, teacher)
Rooms(name, #seats)
ResidesIn(code, room)
```

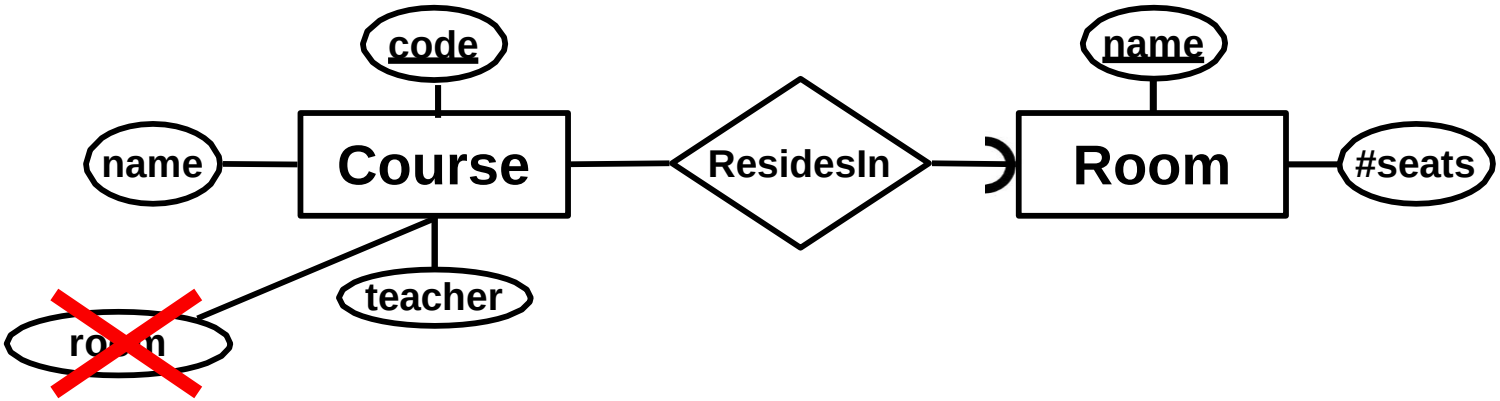
Note that ”room” is not a key here (why not?)

- Safe translation - no NULLs anywhere.
- May lead to duplication of the course code.
- May lead to more *joins*.
- Default translation rule, use unless you have a good reason not to.

```
Courses(code, name, teacher, room)
Rooms(name, #seats)
```

- Will lead to NULLs for courses that have no room.
- Can sometimes be preferred when *not* having a room is an uncommon exception to the rule.
- Reduces the need for *joins.*

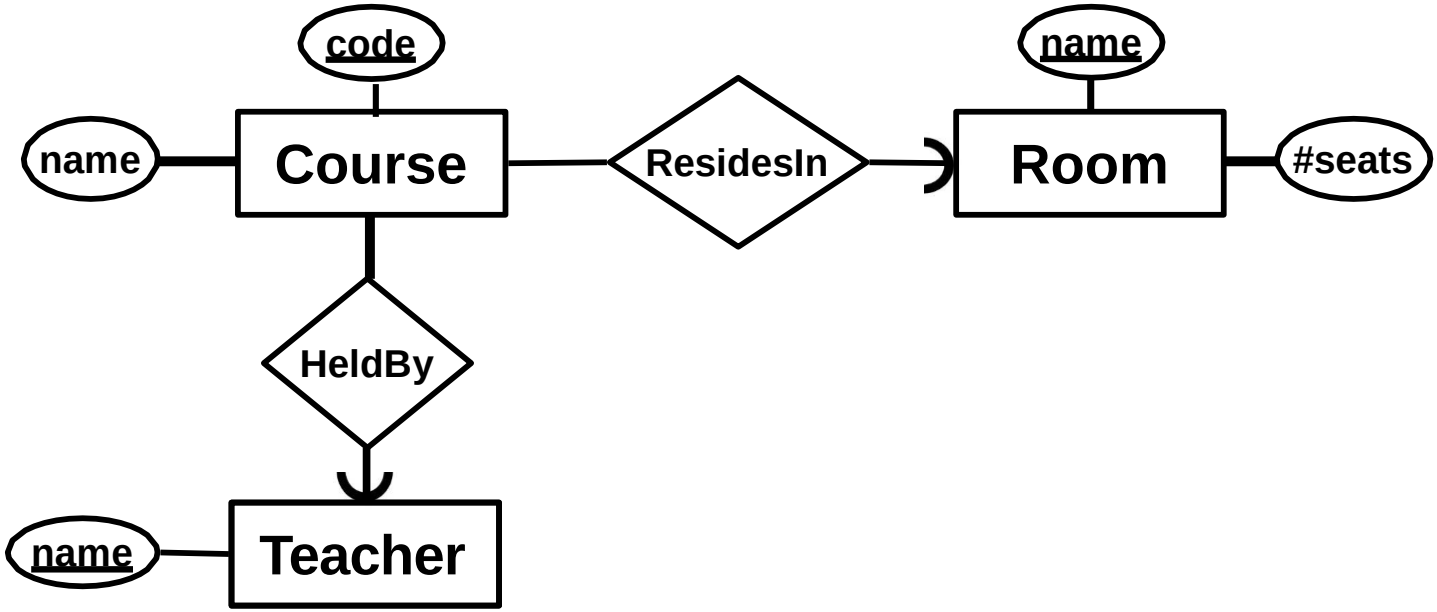
Bad E-R design



- Room is a related entity – not an attribute as well!
- E-R modelling error #1 – don’t do this!!

Attribute or related entity?

What about teacher? Isn’t that an entity?



Quiz!

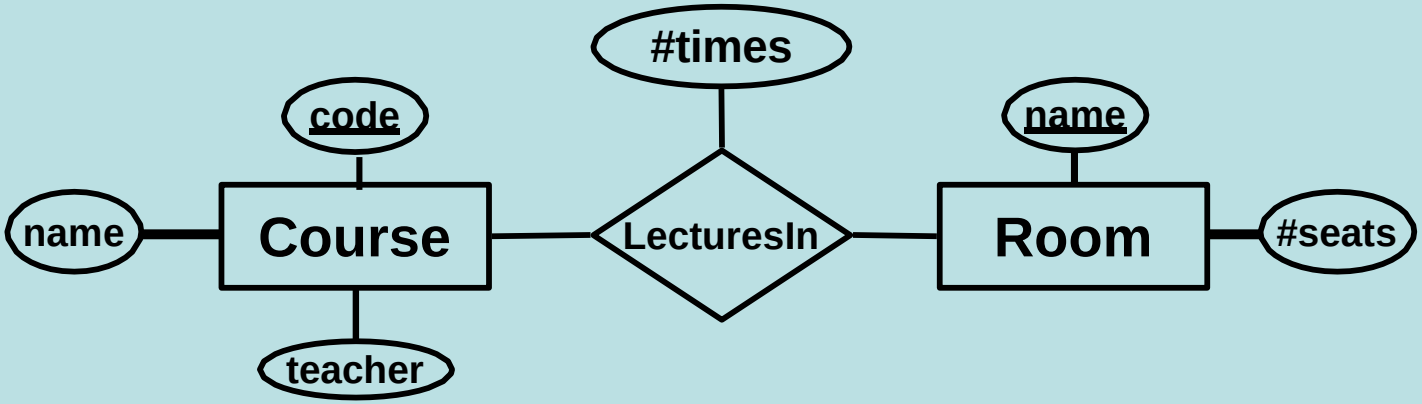
When should we model something as an entity in its own right (as opposed to an attribute of another entity)?

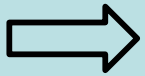
At least one of the following should hold:

- Consists of more than a single (key) attribute
- Used by more than one other entity
- Part of an X-to-many relation as the many side
- Generally entity-ish, is important on its own

Quiz!

- Translate this E-R diagram to relations

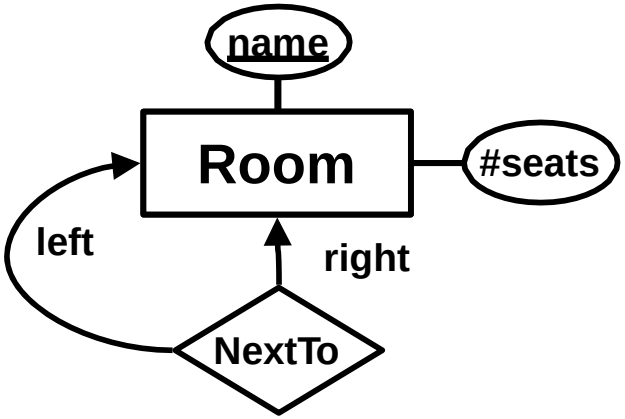


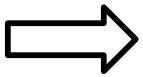


```
Courses(code, name, teacher)
Rooms(name, #seats)
LecturesIn(course, room, #times)
  course -> Courses.code
  room   -> Rooms.name
```

Relationships to ”self”

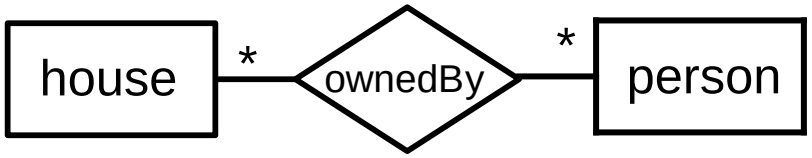
- A relationship can exist between entities of the same entity set.
- Use *role* annotations for attributes.





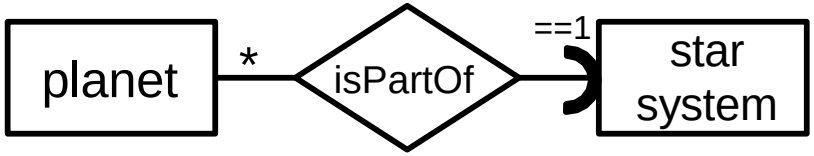
```
Rooms(name, #seats)
NextTo(left, right)
  left  -> Rooms.name
  right -> Rooms.name
```

ER Cheatsheet 2



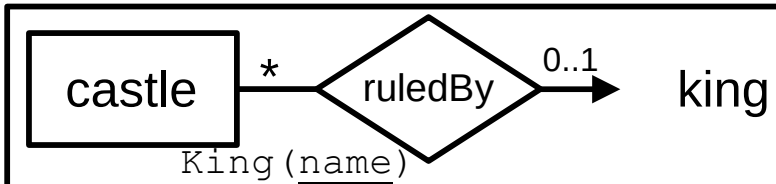
A house can be owned by several people
A person can own several houses

```
Person(name)
House(address)
OwnedBy(owner, property)
     owner -> Person.name
     property -> House.address
```



A planet is part of exactly 1 starsystem
A starsystem can have several planets

```
Starsystem(name)
Planet(name, system)
     system -> Starsystem.name
```



A castle is ruled by 0 or 1 king
A king can rule several castles

```
King(name)
Castle(name)
RuledBy(castle, king)
     castle -> Castle.name
     king -> King.name
```

… or …

```
King(name)
Castle(name, king)
     king -> King.name
```

(king can be NULL!)