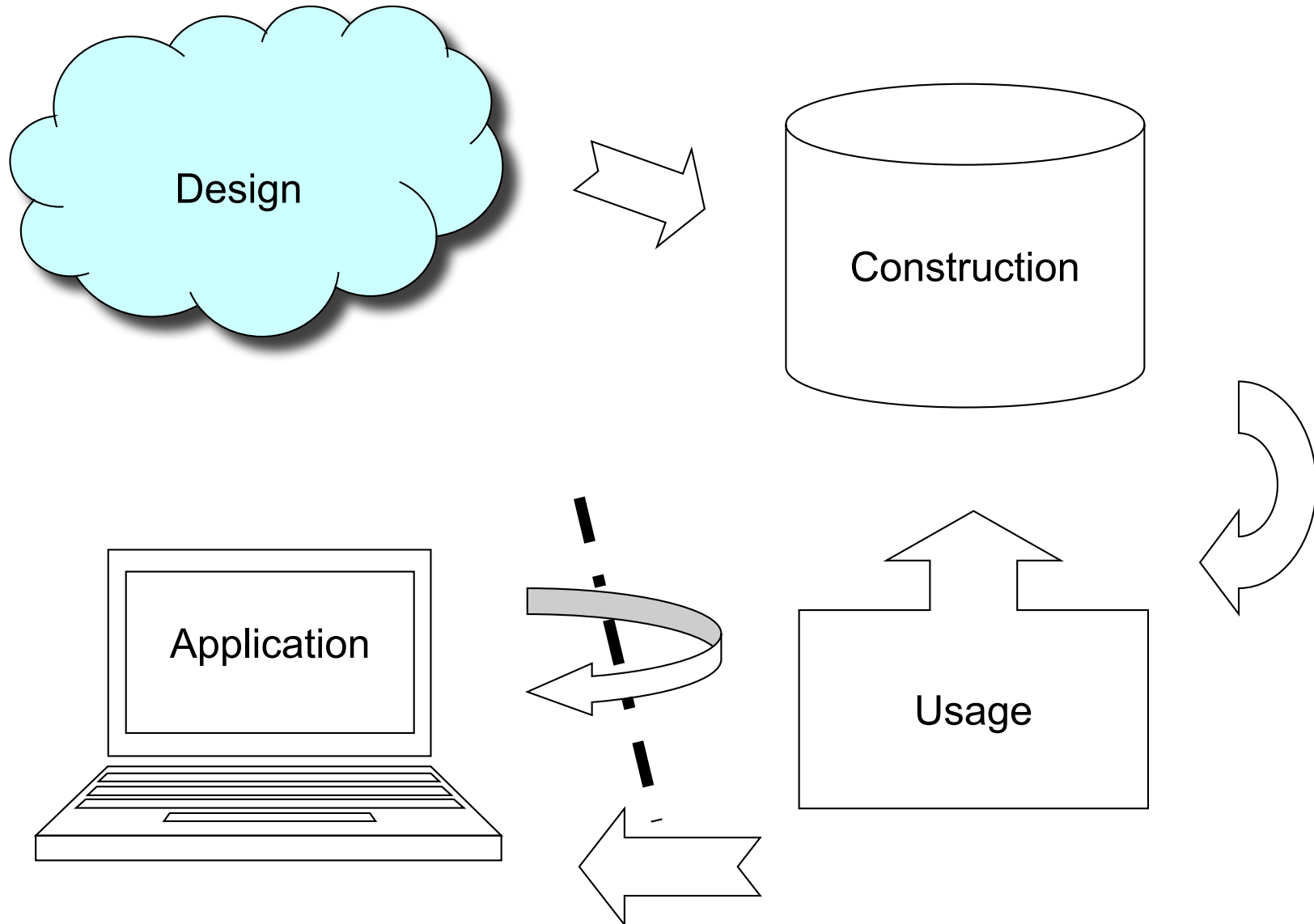


# Database design

## The Entity-Relationship model

# Course Objectives



# The Entity-Relationship approach

- Design your database by drawing a picture of it – an *Entity-Relationship diagram*
  - Allows us to sketch the design of a database informally (which is good when communicating with customers)
- Use (more or less) mechanical methods to convert your diagram to relations.
  - This means that the diagram can be a formal specification as well

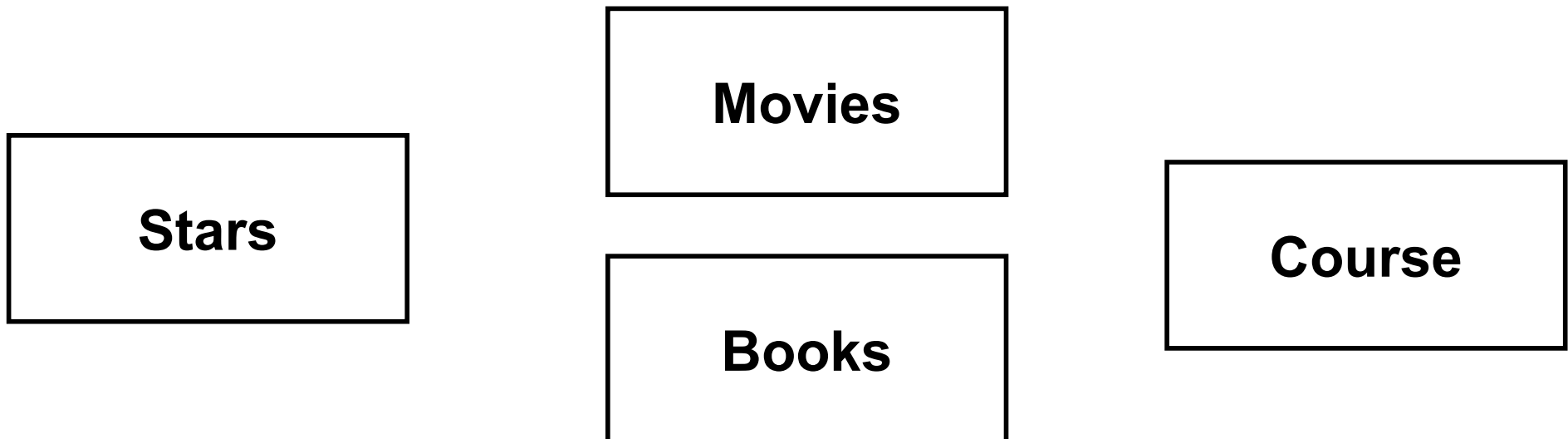
# ER BASICS

# E/R Model

- Three main element types:
  - Entity sets
  - Attributes, and
  - Relationships

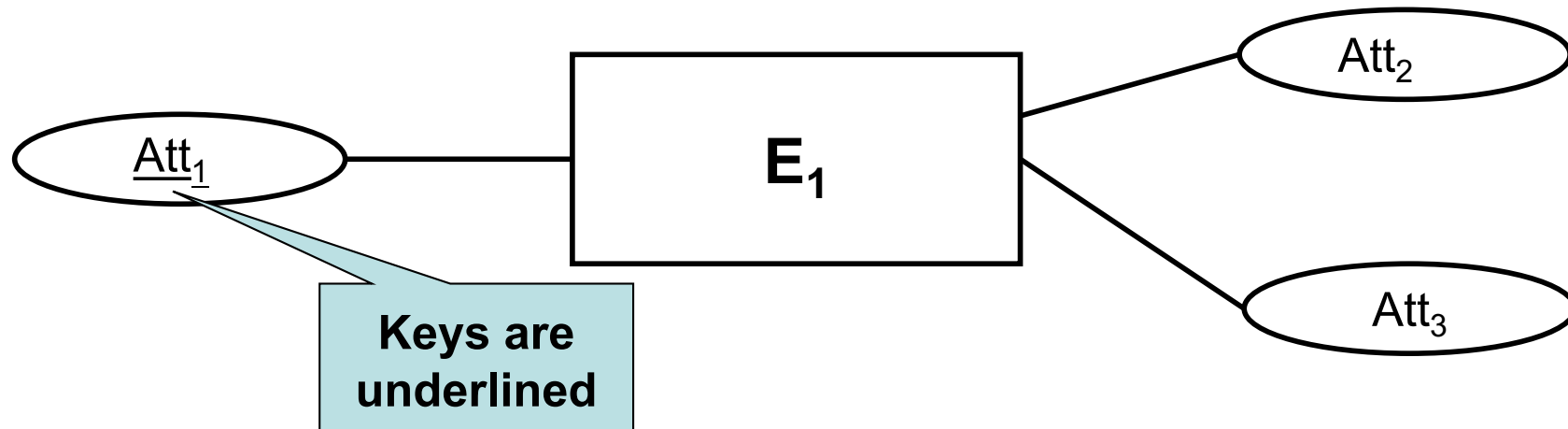
# Entity Sets

- *Entity* = object that exists and distinguishable from other entities
  - course, room, person, customers, books, etc.
- *Entity set* = collection of similar entities
  - all courses, all rooms etc.
- Entities are drawn as rectangles



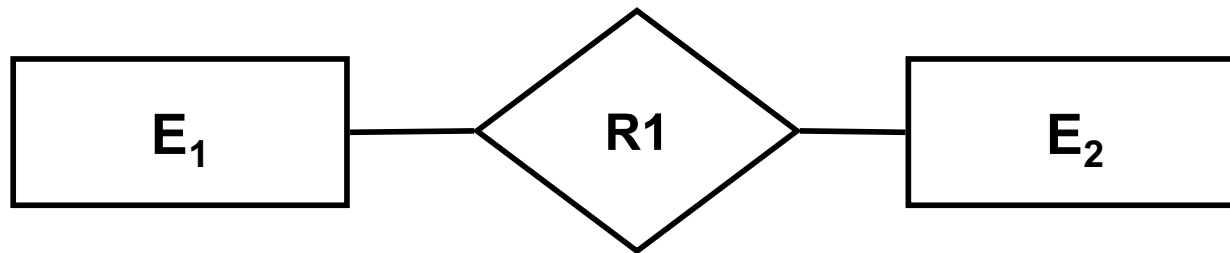
# Attributes

- Entity sets have the same attributes (though not the same values)
- Attributes are drawn as ovals connected to the entity by a line.



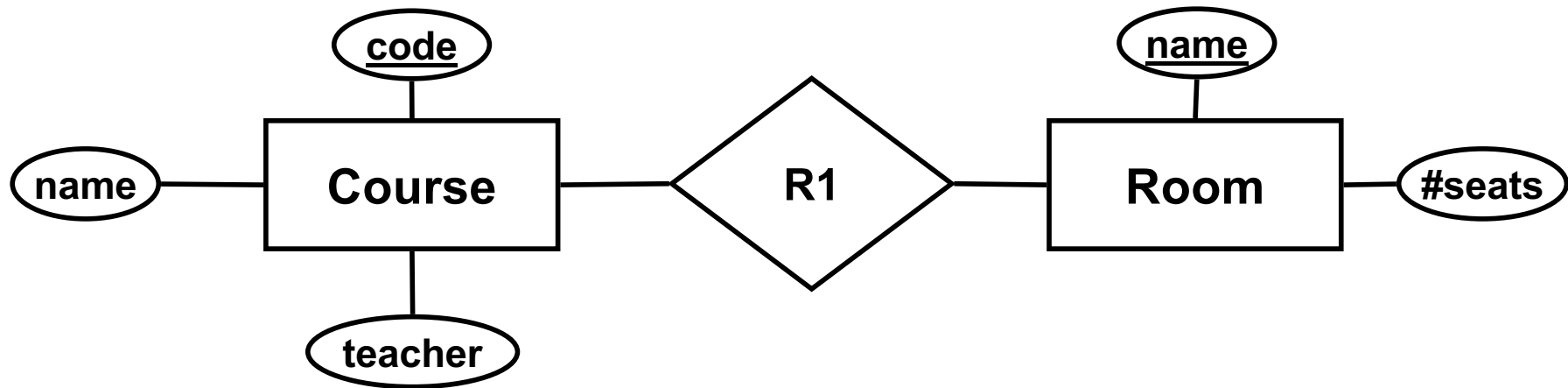
# Relationships

- A relationship is an association among several entities
- Drawn as a diamond between the related entities, connected to the entities by lines.
- Note: Relationship  $\neq$  Relation!!



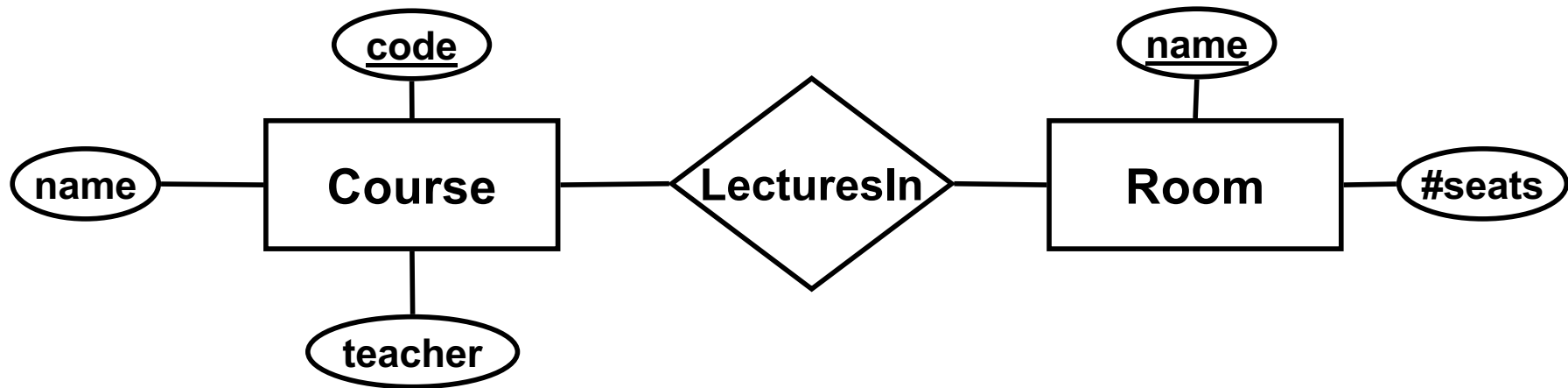


## Examples:



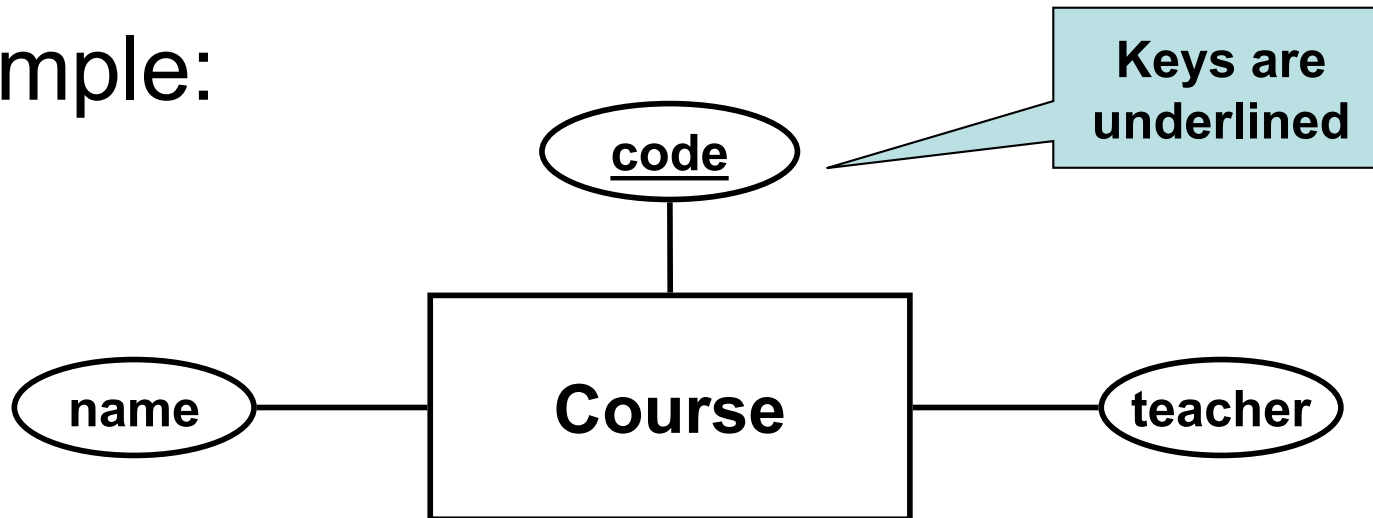
- A course has lectures in a room.
- A course is related to a room by the fact that the course has lectures in that room.
- Both entities are related through the relationship named "R1"

Example:



- A course has lectures in a room.
- A course is related to a room by the fact that the course has lectures in that room.
- A relationship is **often** named with a verb form (LecturesIn)

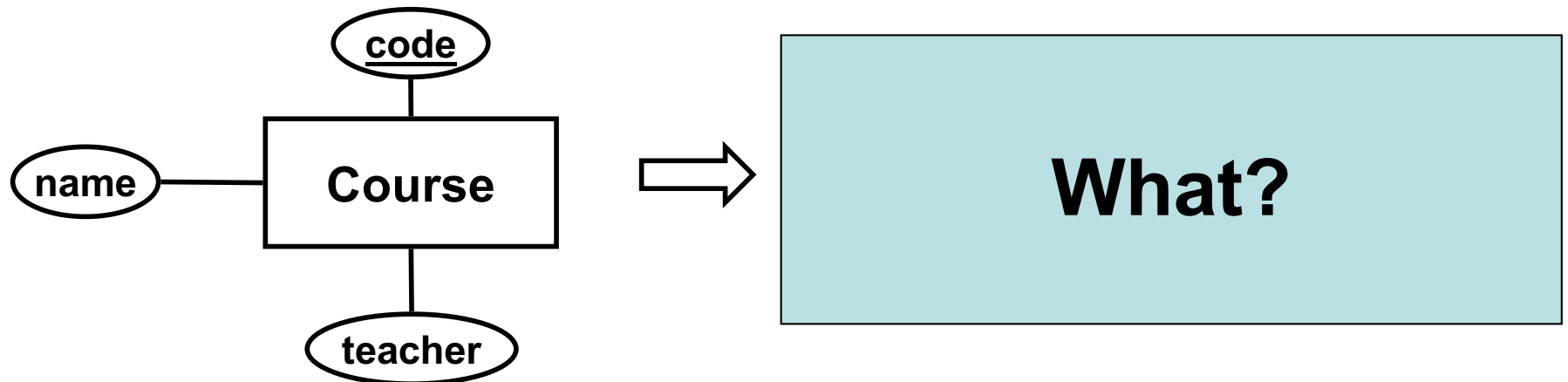
Example:



- A course has three attributes – the unique course code, a name and the name of the teacher.
- All course entities have values for these three attributes, e.g. (TDA357, Databases, Mickey).

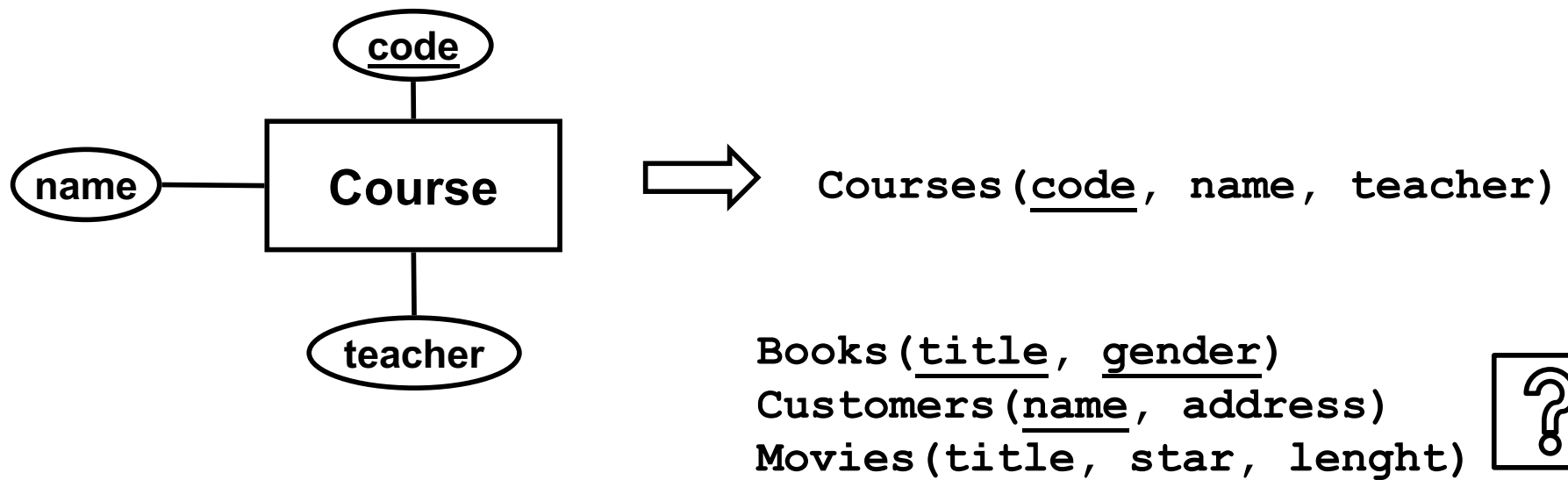
# Translation to relations

- An E-R diagram can be mechanically translated to a relational database schema.
- An entity becomes a relation, the attributes of the entity become the attributes of the relation, keys become keys.



# Translation to relations

- An E-R diagram can be mechanically translated to a relational database schema.
- An entity becomes a relation, the attributes of the entity become the attributes of the relation, keys become keys.

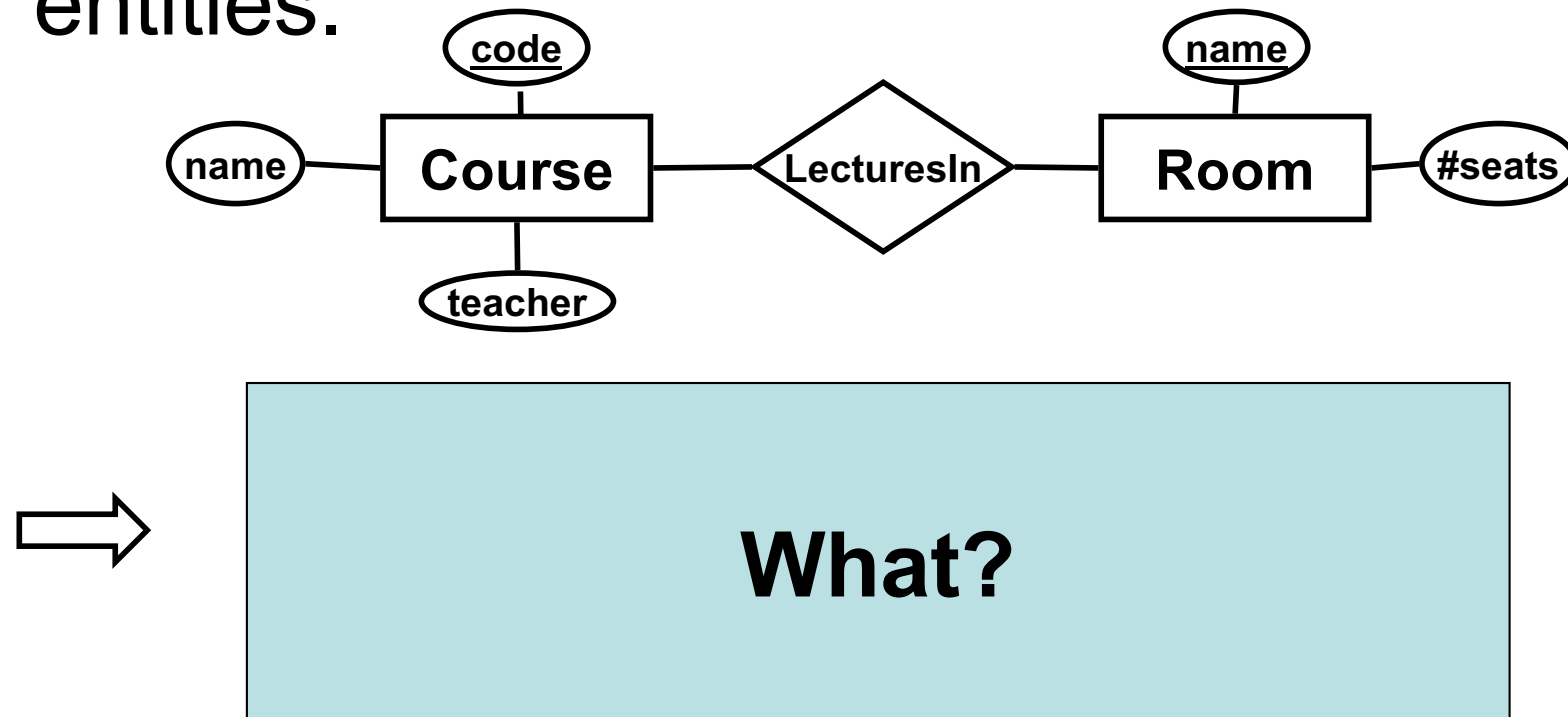


# A note on naming policies

- My view: A rectangle in an E-R diagram represents an entity, hence it is put in singular (e.g. Course).
  - Fits the intuition behind attributes and relationships better.
- The book: A rectangle represents an entity set, hence it is put in plural (e.g. Courses)
  - Easier to mechanically translate to relations.

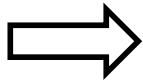
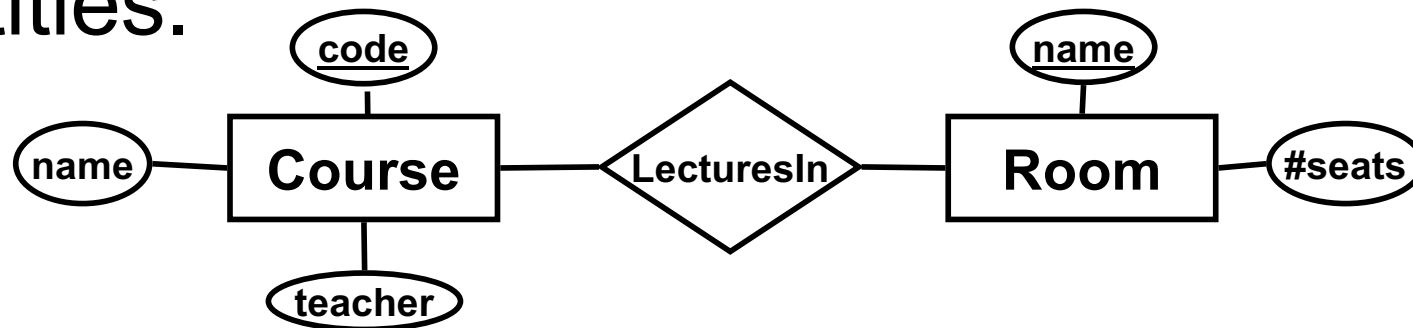
# Translation to relations

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



# Translation to relations

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



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



# References

```
Courses(code, name, teacher)
Teacher(name, #seats)
LecturesIn(code, name)
```

- We must ensure that the codes used in **LecturesIn** matches those in **Courses**.
  - Introduce *references* between relations.
  - e.g. the course codes used in **LecturesIn** *reference* those in **Courses**.

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



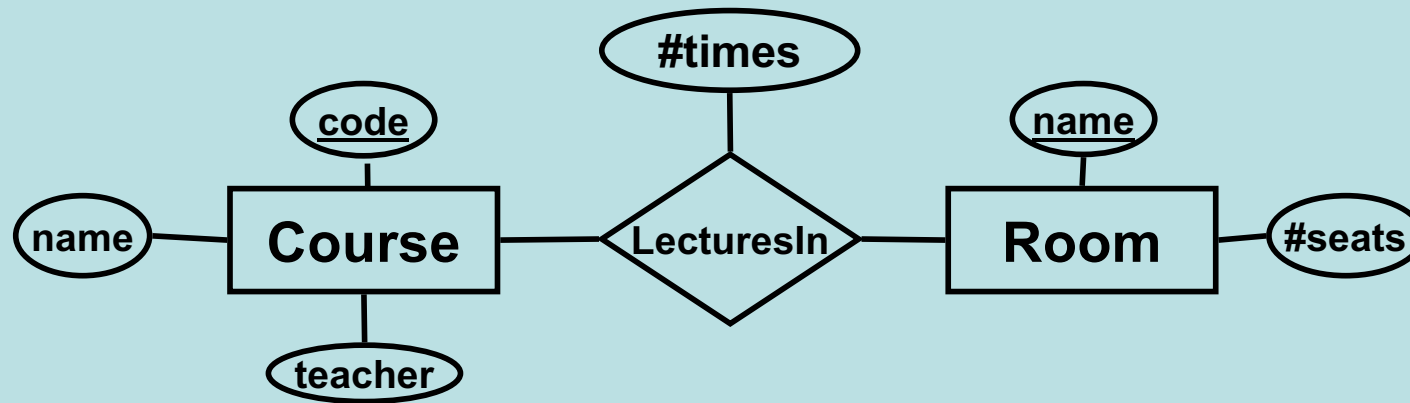
References

# ”Foreign” keys

- Usually, a reference points to the key of another relation.
  - E.g. **name** in **LecturesIn** references the key **name** in **Rooms**.
  - **name** is said to be a *foreign key* in **LecturesIn**.

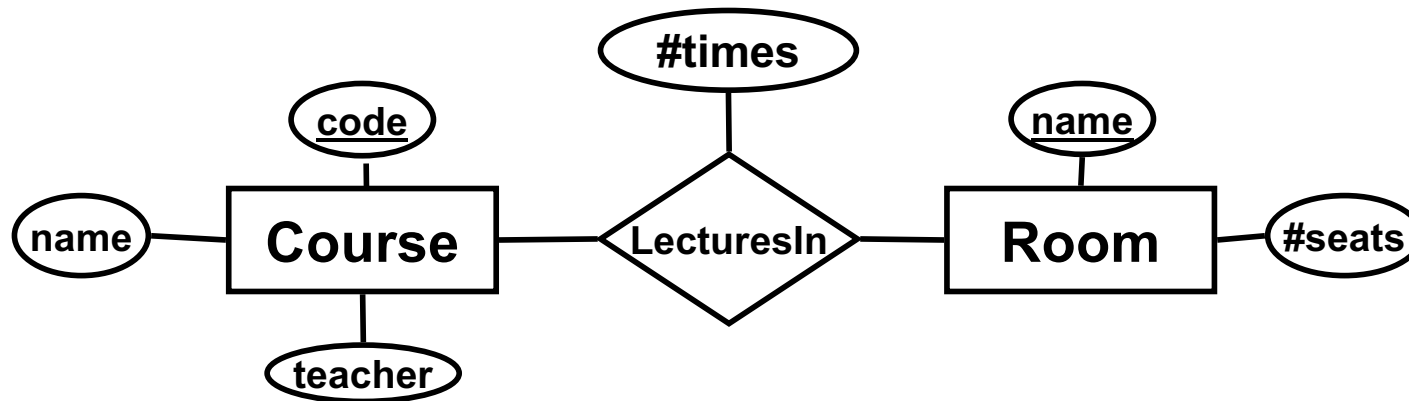
# Quiz

Suppose we want to store the number of times that each course has a lecture in a certain room. How do we model this?



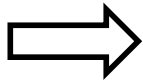
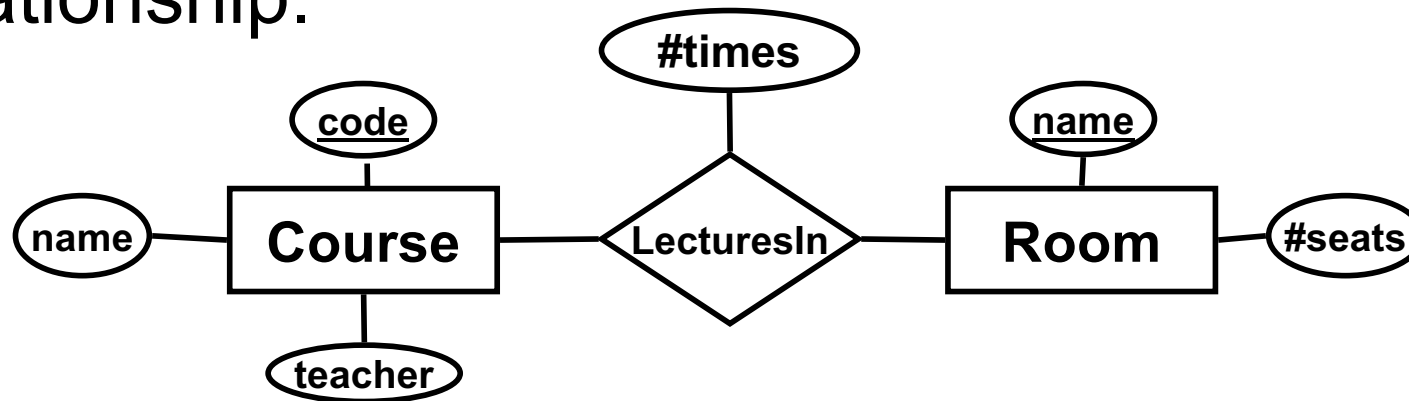
# Attributes on relationships

- Relationships can also have attributes.
- Represent a property of the relationship between the entities.
  - E.g. **#times** is a property of the relationship between a course and a room.



# Translation to relations

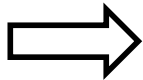
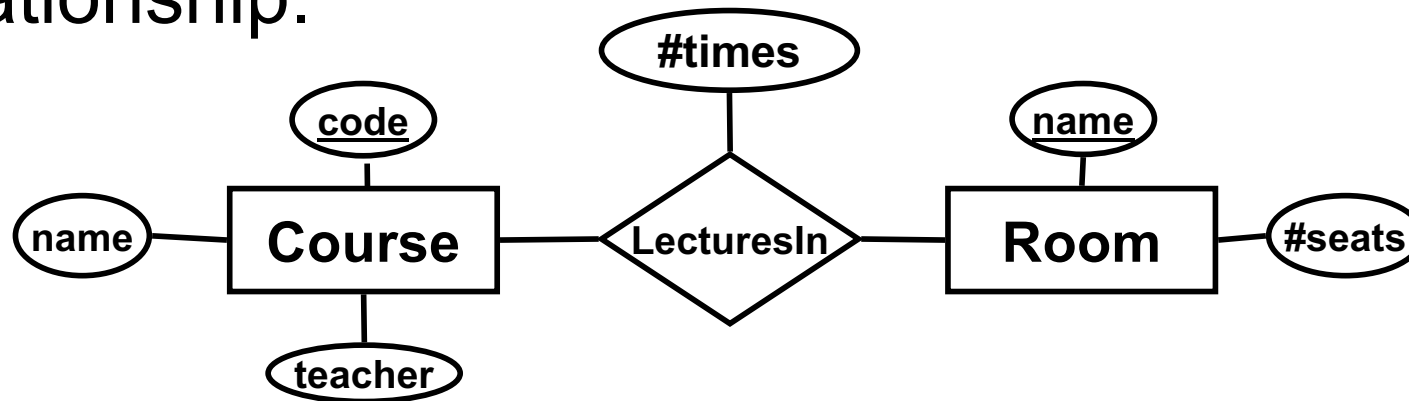
- A relationship between two entities is translated into a relation, where the attributes are the *keys* of the related entities, plus any attributes of the relationship.



**What?**

# Translation to relations

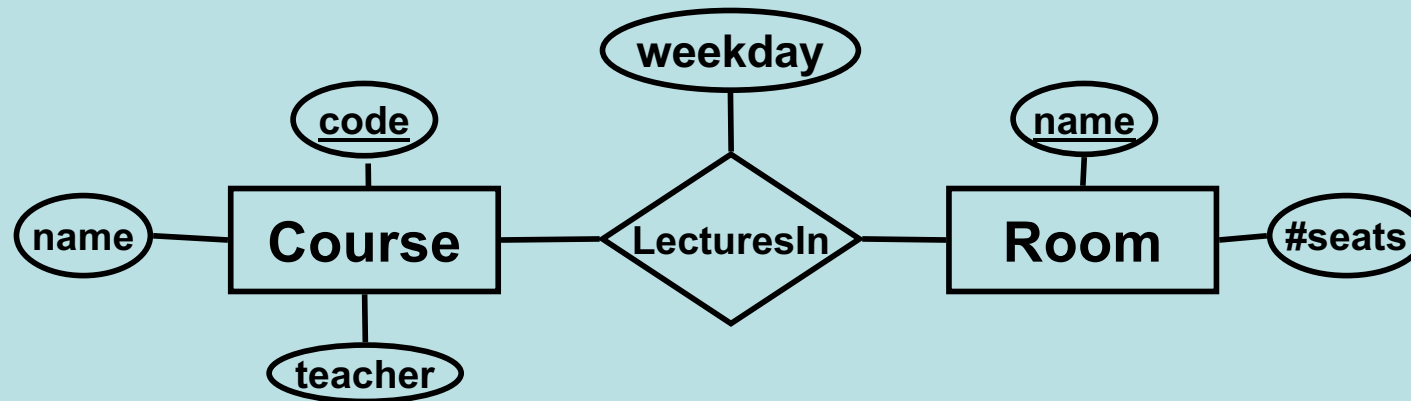
- A relationship between two entities is translated into a relation, where the attributes are the *keys* of the related entities, plus any attributes of the relationship.



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

# Quiz

Why could we not do the same for weekday?



- Not a property of the relationship – a course can have lectures in a given room on several weekdays!
- A pair of entities are either related or not.

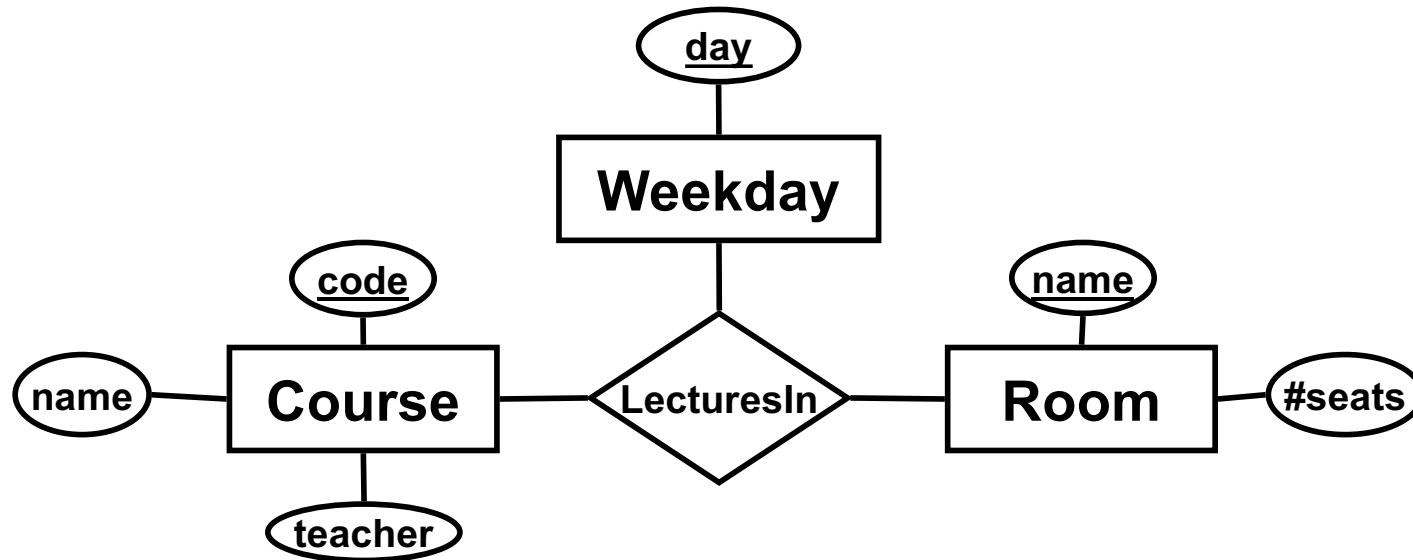
# Relationship (non-)keys

- Relationships have no keys of their own!
  - The "key" of a relationship is the combined keys of the related entities
  - Follows from the fact that entities are either related or not.
  - If you at some point think it makes sense to put a key on a relationship, it should probably be an entity instead.

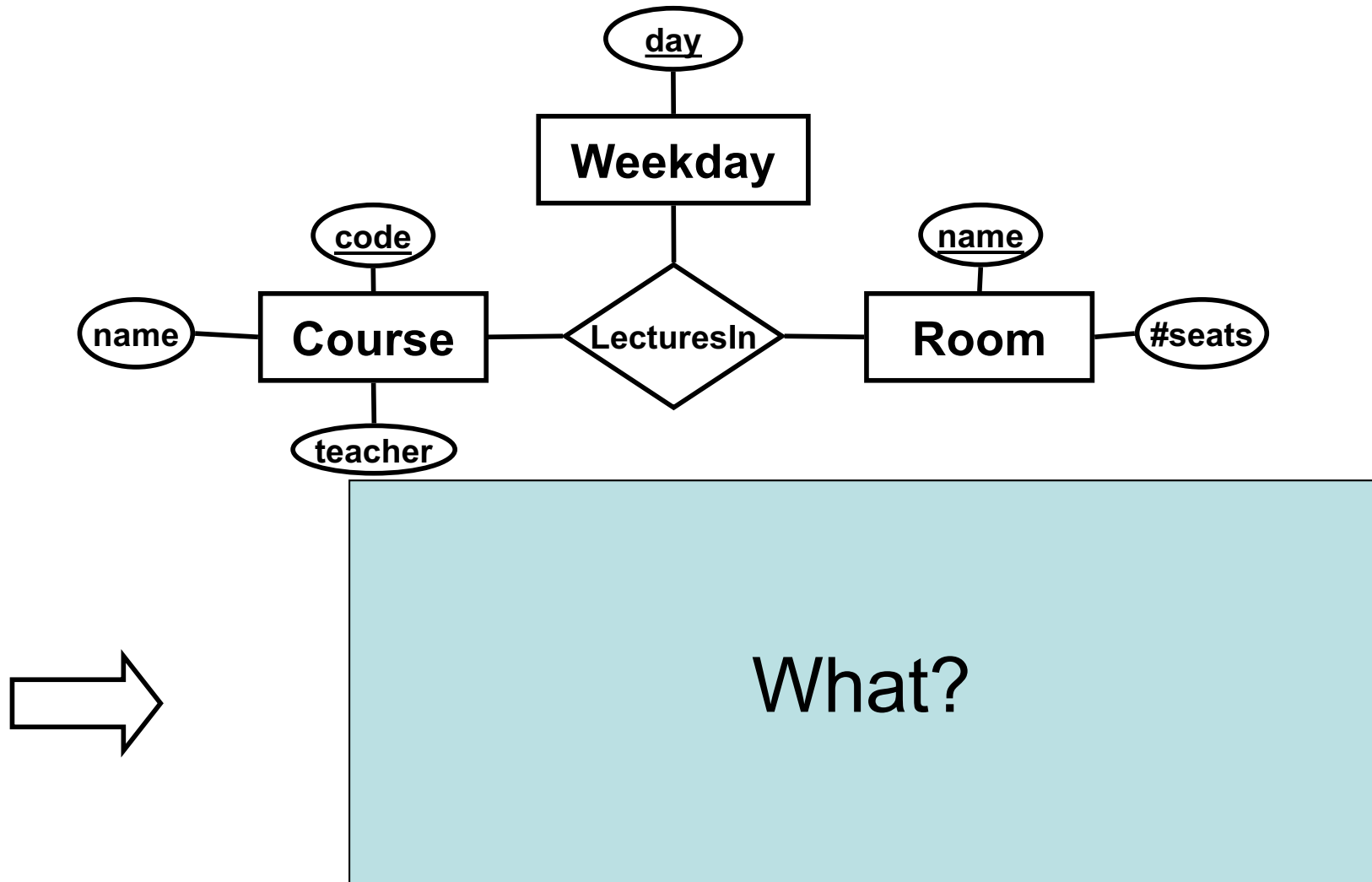


# Multiway relationships

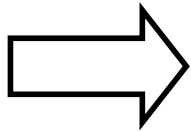
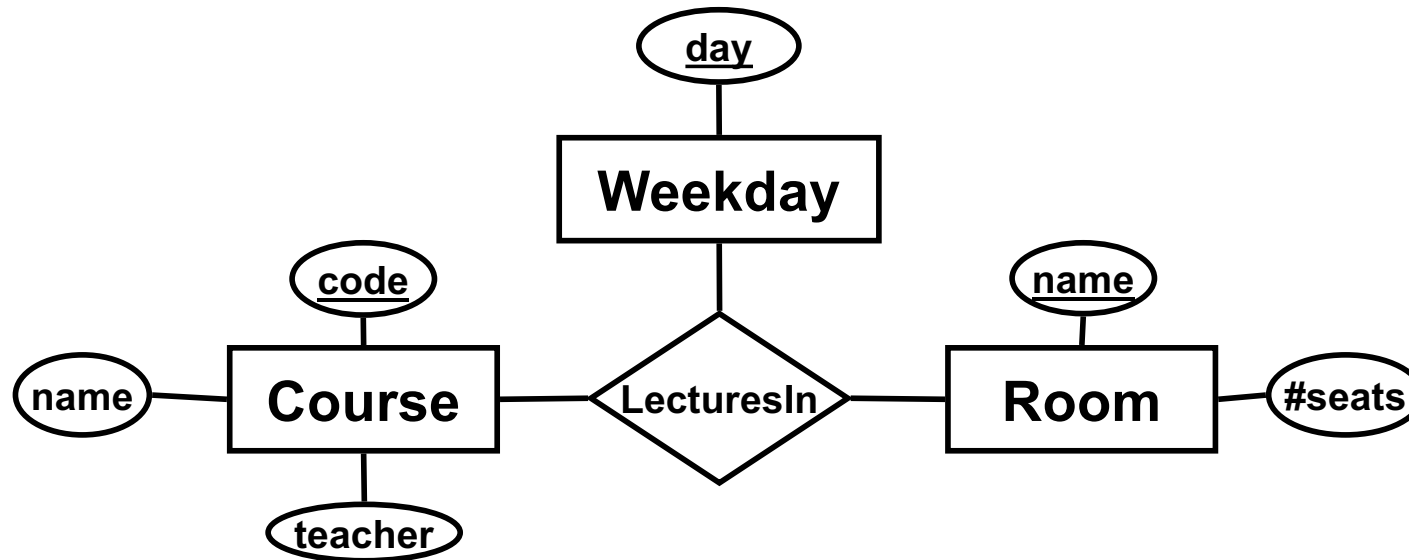
- A course has lectures in a given room on different weekdays.



- Translating to relations:



- Translating to relations:



`Courses (code, name, teacher)`

`Rooms (name, #seats)`

`Weekdays (day)`

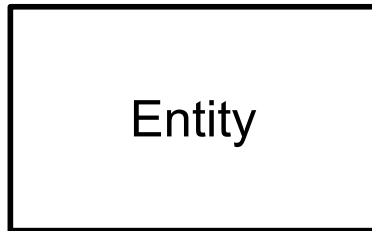
`LecturesIn (code, name, day)`

`code -> Courses.code`

`name -> Rooms.name`

`day -> Weekdays.day`

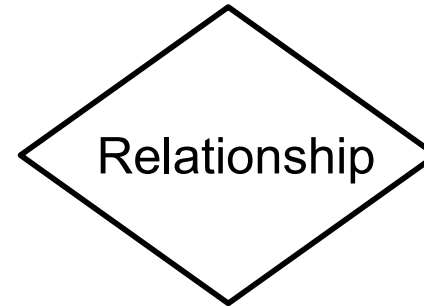
# ER Cheatsheet 1



ENTITY = noun/thing

- Exist on their own
- Have their own keys

```
Course(code, name, teacher)
Room(name, #seats)
Weekday(day)
```



RELATIONSHIP = verb

- Only exist in relation to an entity
- No own keys, only foreign keys
- Reference the entity keys with ->

```
HasLecturesIn(code, name, day, #times)
code -> Course.code
name -> Room.name
day -> Weekday.day
```

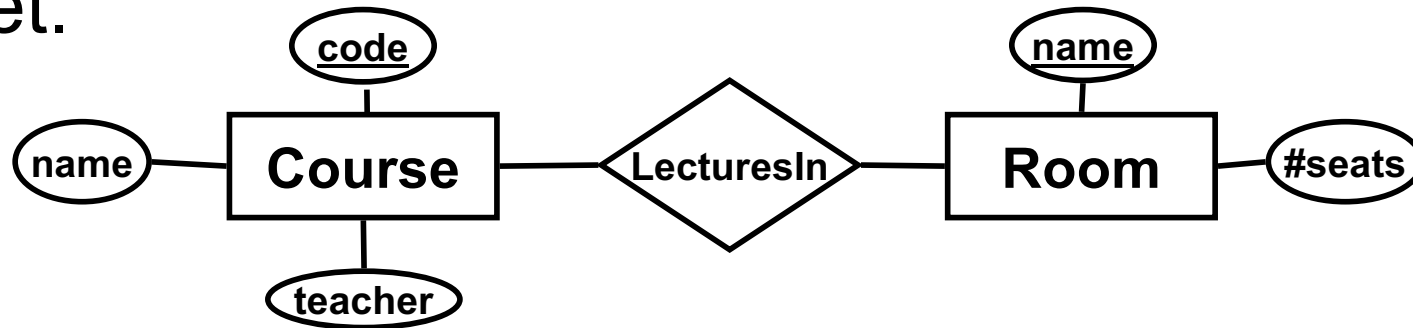
Both entities and relationships can have attributes!



**CARDINALITY**

# Many-to-many relationships

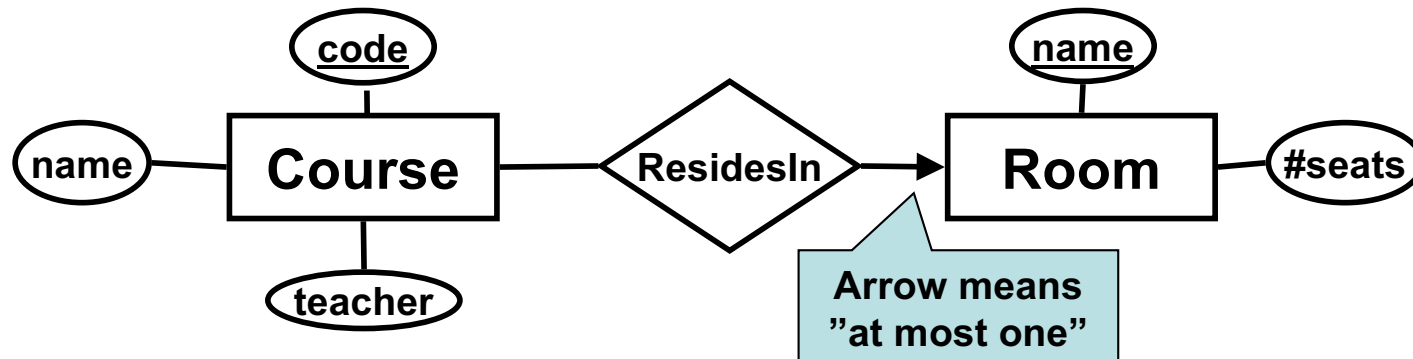
- Many-to-many (n-to-n, many-many) relationships
  - Each entity in either of the entity sets can be related to any number of entities of the other set.



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

# Many-to-one relationships

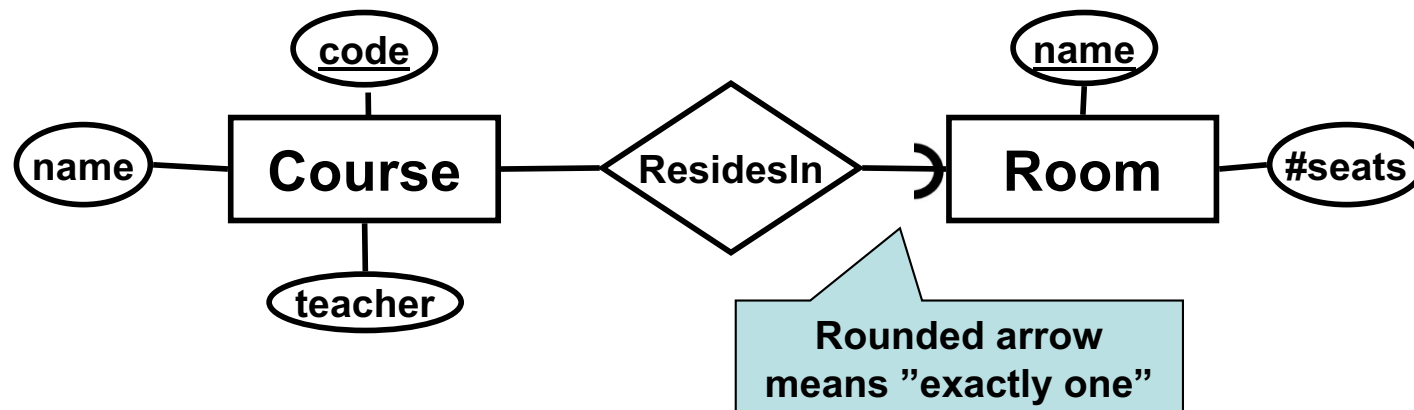
- Many-to-one (n-to-1, many-one) relationships
  - Each entity on the "many" side can only be related to (at most) one entity on the "one" side.



- 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***

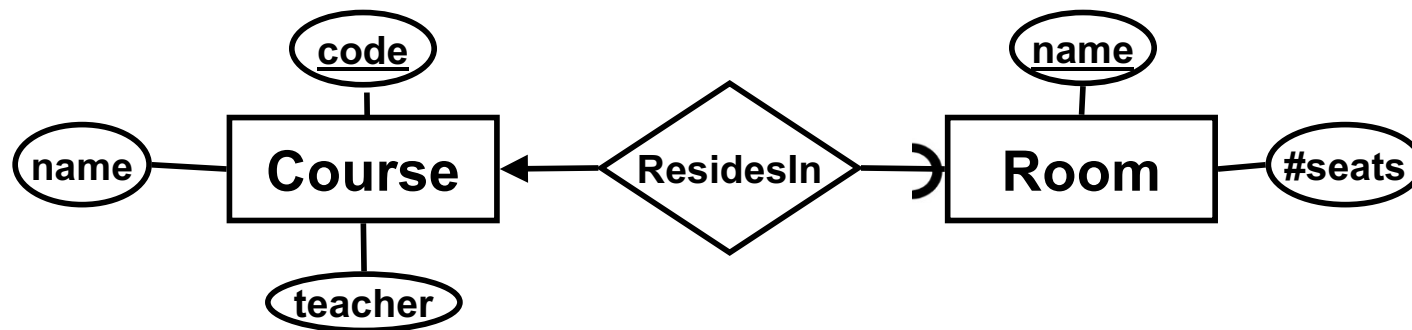


- All courses have all their lectures in some room.
- Many courses can share the same room.



# One-to-one relationships

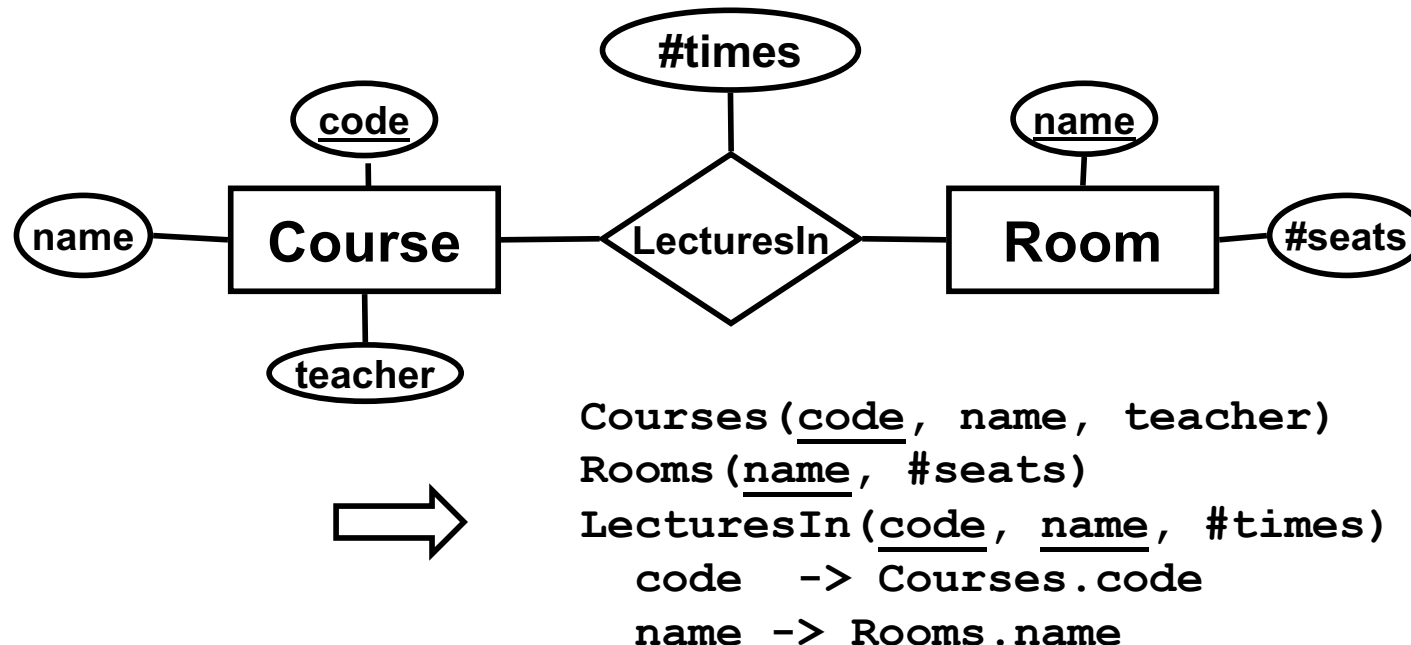
- One-to-one (1-to-1, one-one) relationships
  - Each entity on either side can only be related to (at most) one entity on the other side.



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

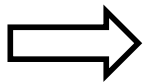
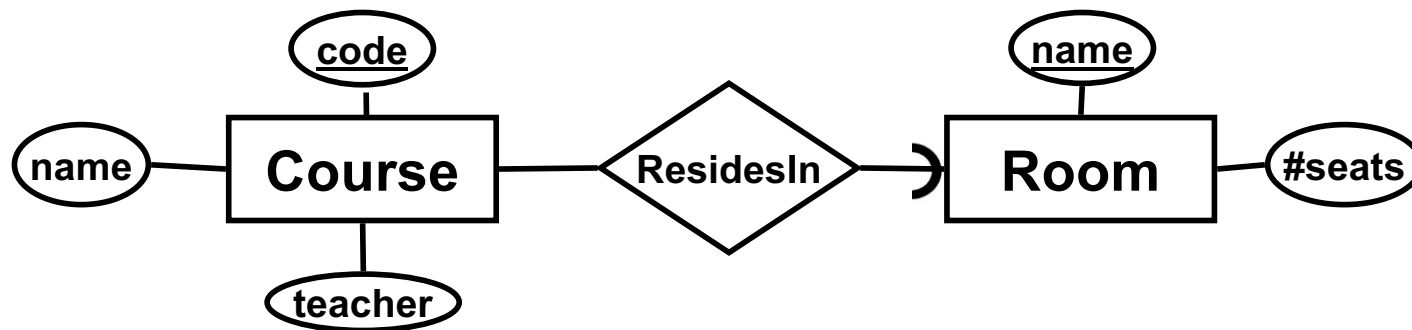
# Translating multiplicity

- 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.



# Translating multiplicity

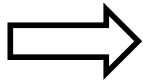
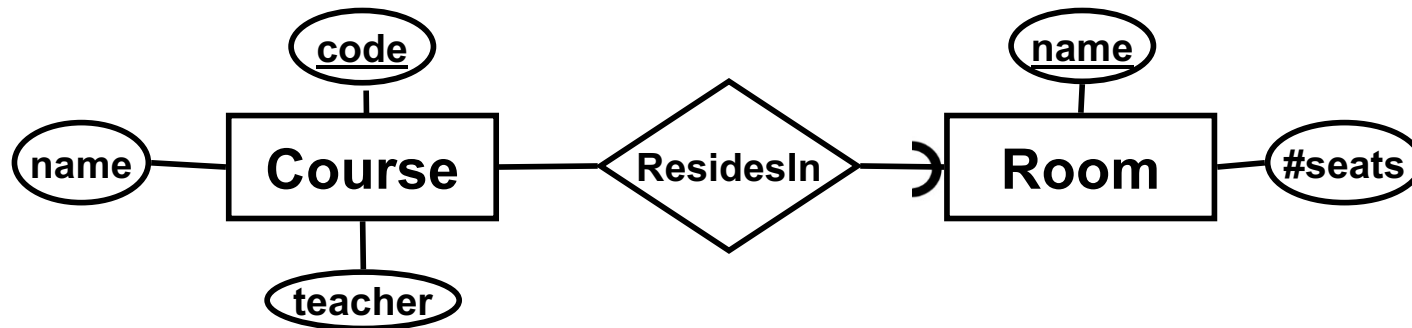
- A *X-to-“exactly one”* relationship between two entities is translated as part of the “many”-side entity.



What?

# Translating multiplicity

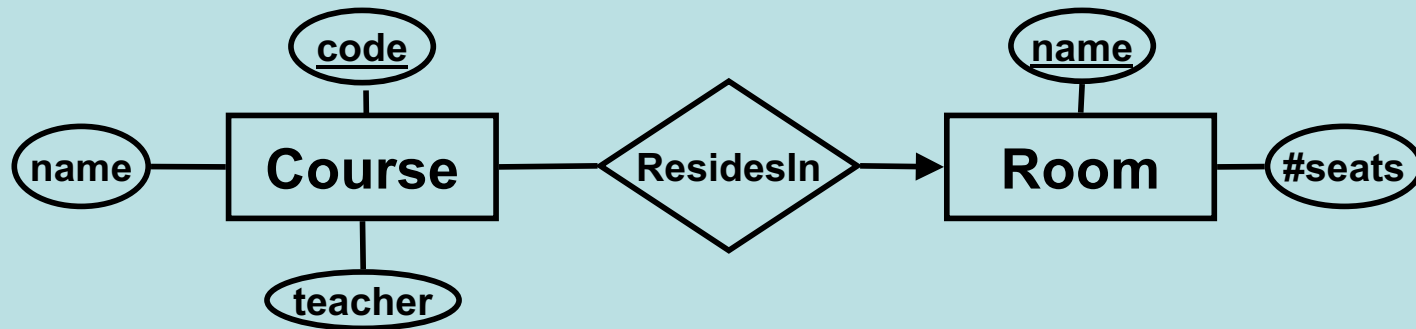
- A *X-to-“exactly one”* relationship between two entities is translated as part of the “many”-side entity.



```
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?)

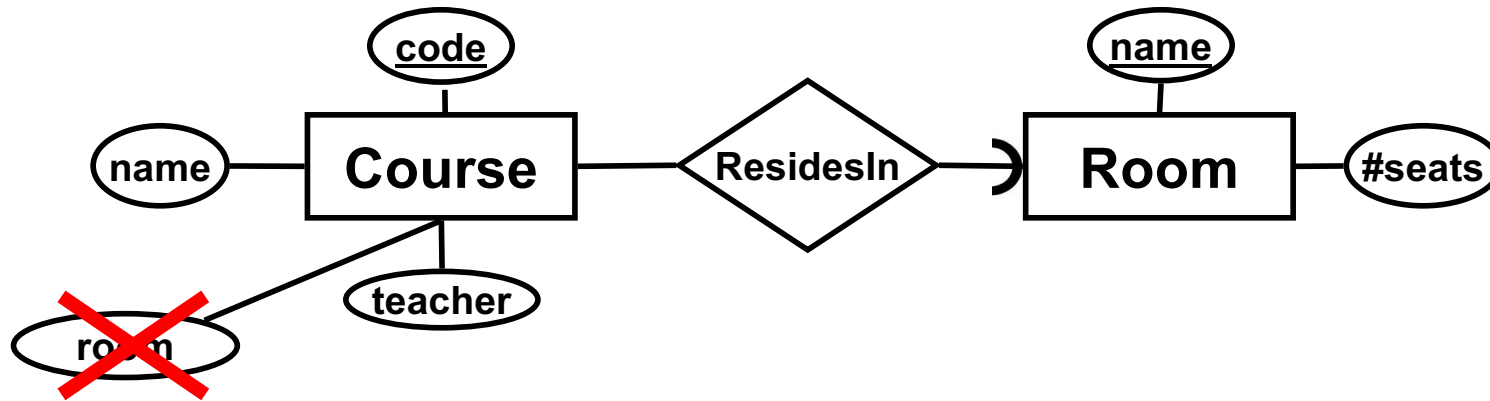
- Safe translation - no NULLs anywhere.
- May lead to duplication of the course code.
- May lead to more *joins*.
- Default translation rule, use unless you have a good reason not to.

**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

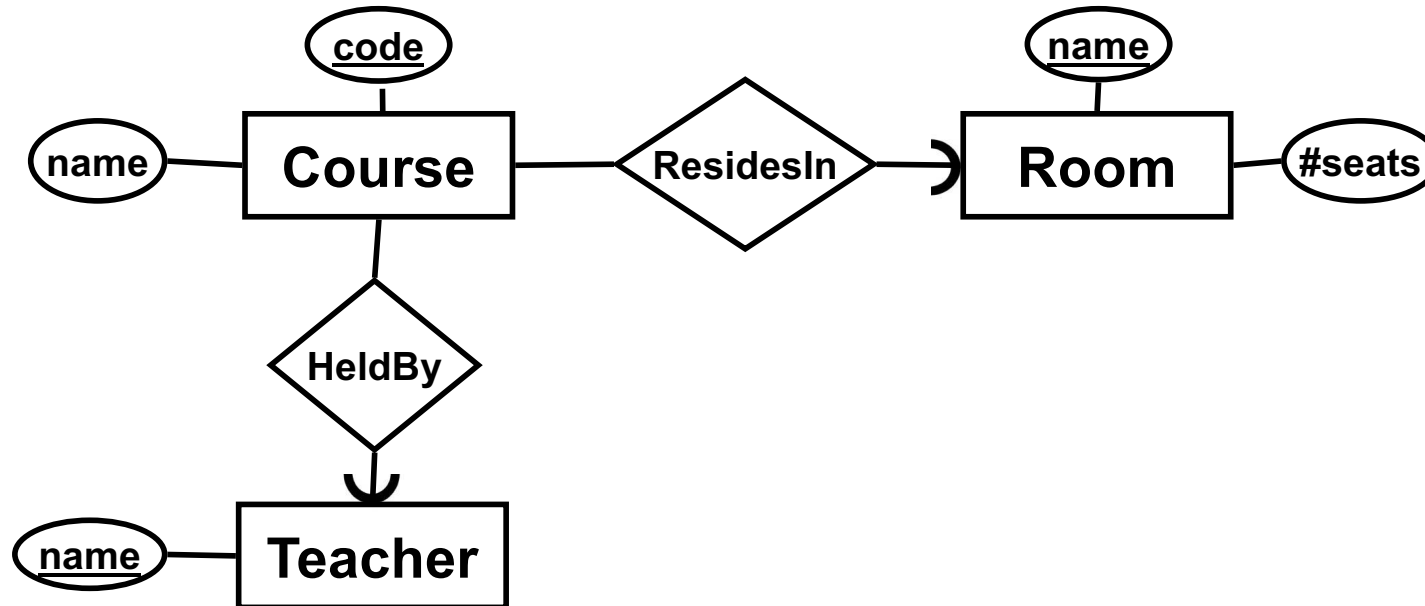


- Room is a related entity – not an attribute as well!
- E-R modelling error #1 – don't do this!!



# Attribute or related entity?

What about teacher? Isn't that an entity?



# Quiz!

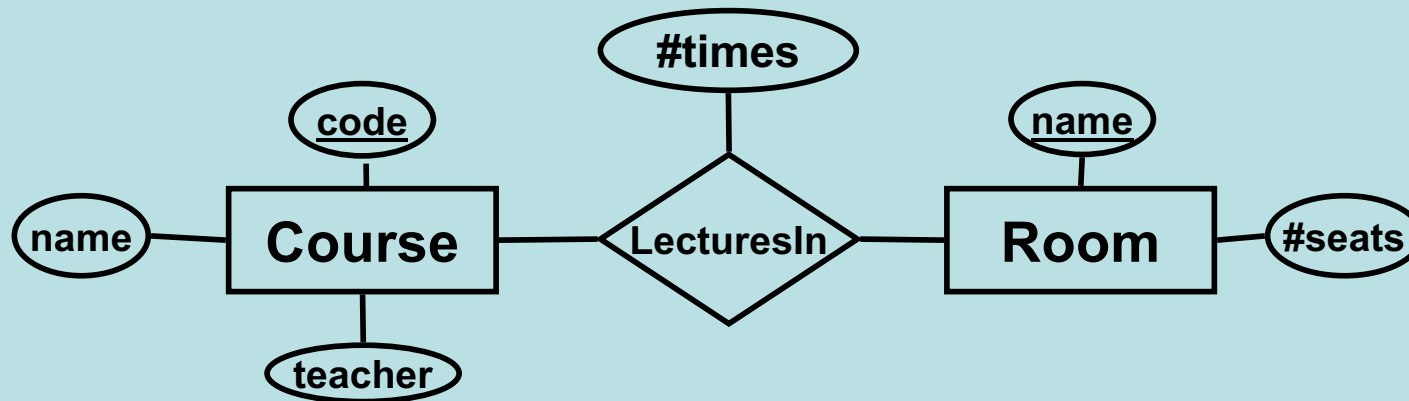
When should we model something as an entity in its own right (as opposed to an attribute of another entity)?

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!

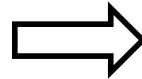
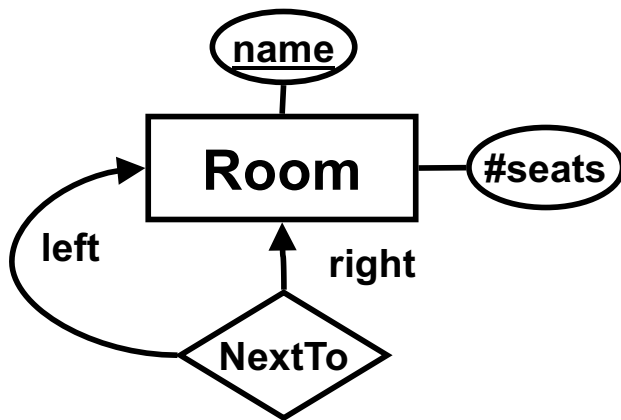
- Translate this E-R diagram to relations



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

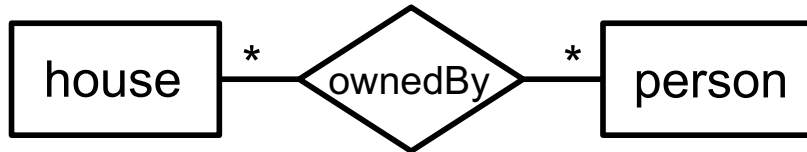
# Relationships to "self"

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



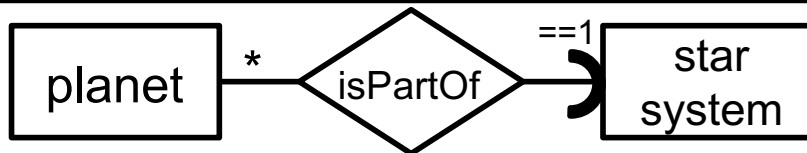
```
Rooms (name, #seats)
NextTo (left, right)
    left -> Rooms.name
    right -> Rooms.name
```

# ER Cheatsheet 2



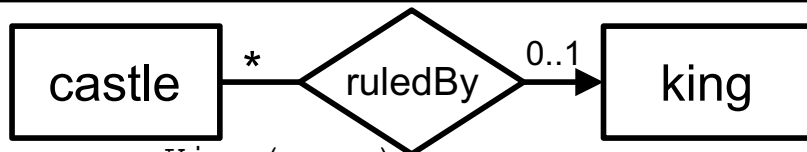
A house can be owned by several people  
A person can own several houses

```
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!)
```

# Break! In part 2:

weak entities, subclasses,  
"multivalued" and "flag" attributes

Subclassing and weak entities

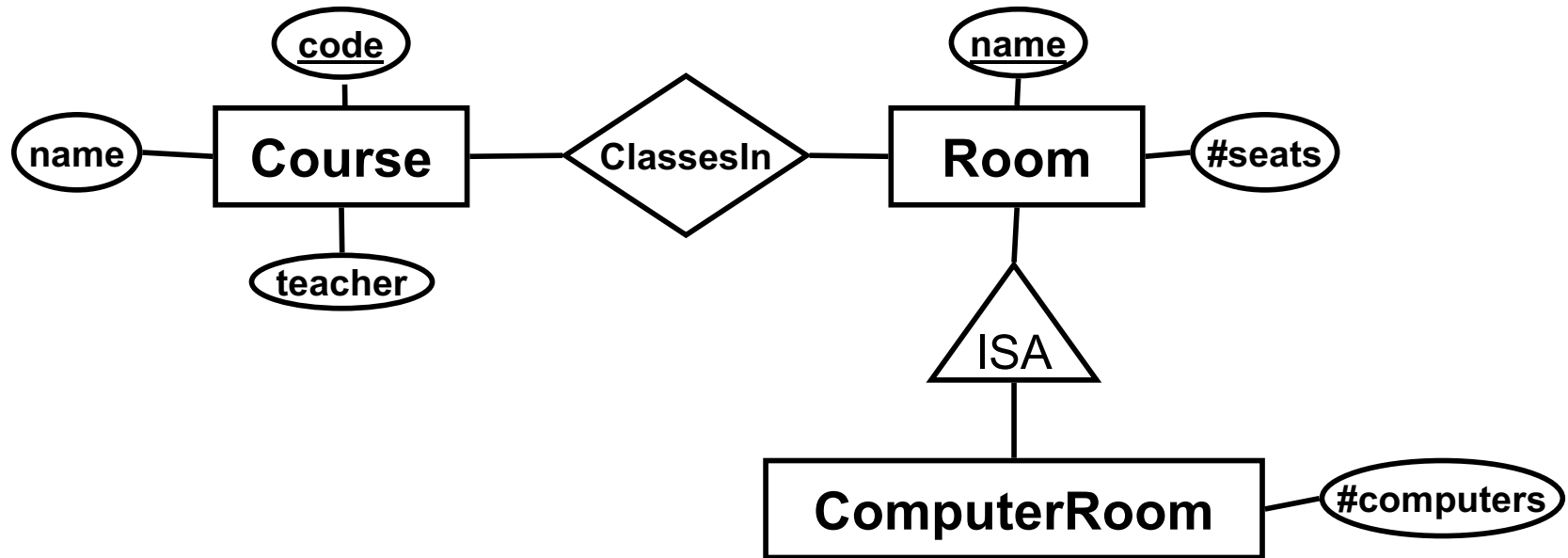
# **SPECIAL RELATIONSHIPS**

# Subclassing

- Subclass = sub-entity = special case.
- A subclass is a subset of an entity set.
- More attributes and/or relationships.
- A subclass shares the key of its parent.
  
- Drawn as an entity connected to the superclass by a special triangular relationship called *ISA*.  
Triangle points to superclass.
  - ISA = "is a"



# Example:



- A computer room *is a* room.
- Not all rooms are computer rooms.
- Computer rooms share the extra property that they have a number of computers.

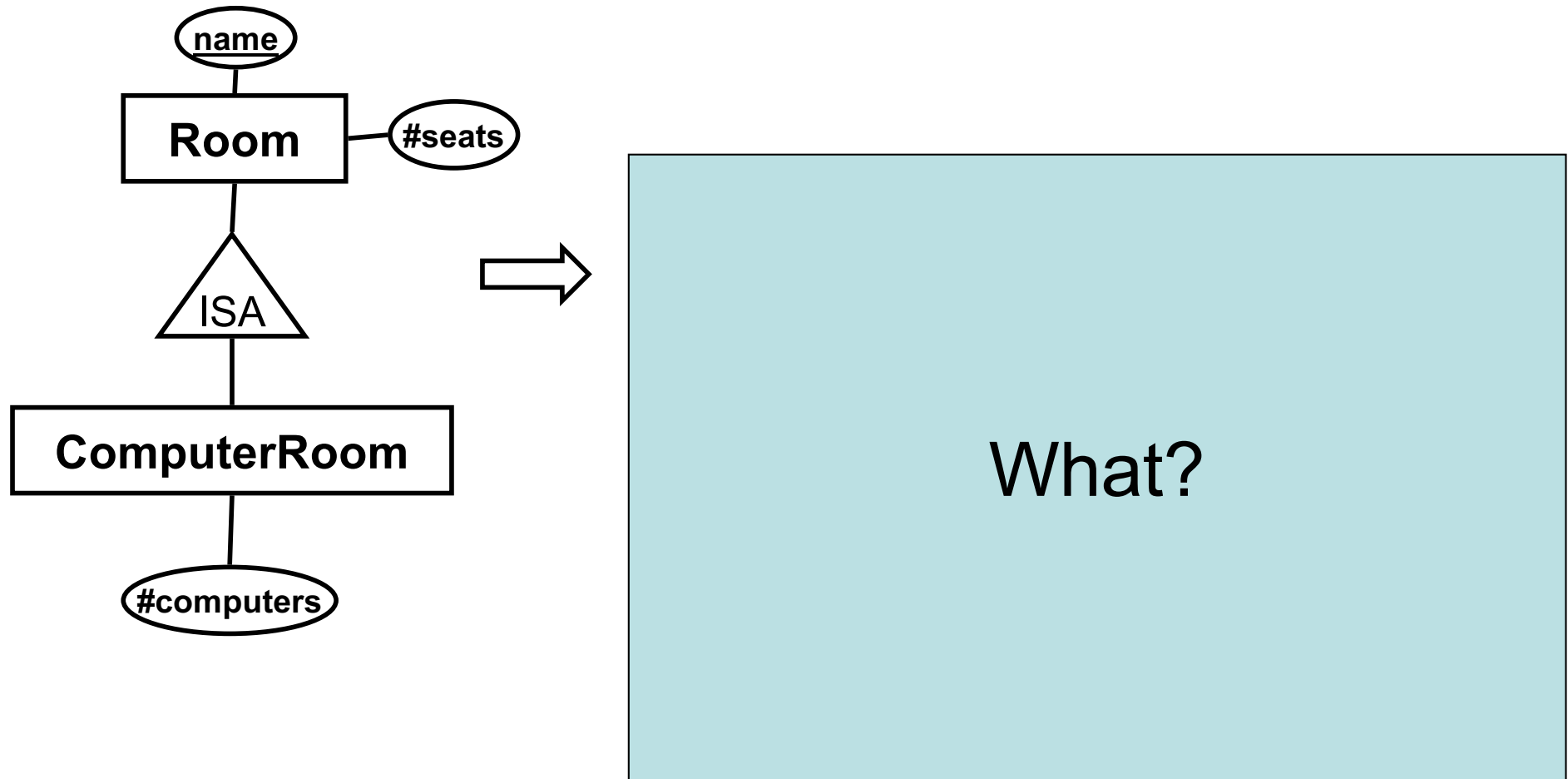
# Subclass/Superclass Hierarchy

- We assume that subclasses form a tree hierarchy.
  - A subclass has only one superclass.
  - Several subclasses can share the same superclass.
    - E.g. Computer rooms, lecture halls, chemistry labs etc. could all be subclasses of Room.

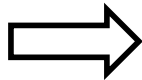
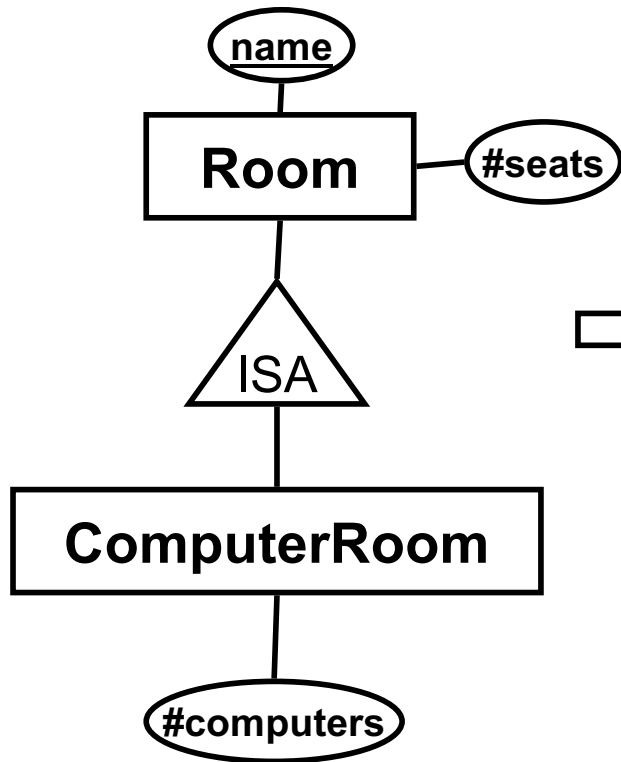
# Translating ISA to relations

- Standard approach:
  - An ISA relationship is a standard one-to-  
"exactly one" relationship. Each  
subclass becomes a relation with the  
key attributes of the superclass  
included.
  - Also known as the E-R approach.

# The E-R approach:



# The E-R approach:



Rooms (name, #seats)

ComputerRooms (name, #computers)

name -> Rooms.name

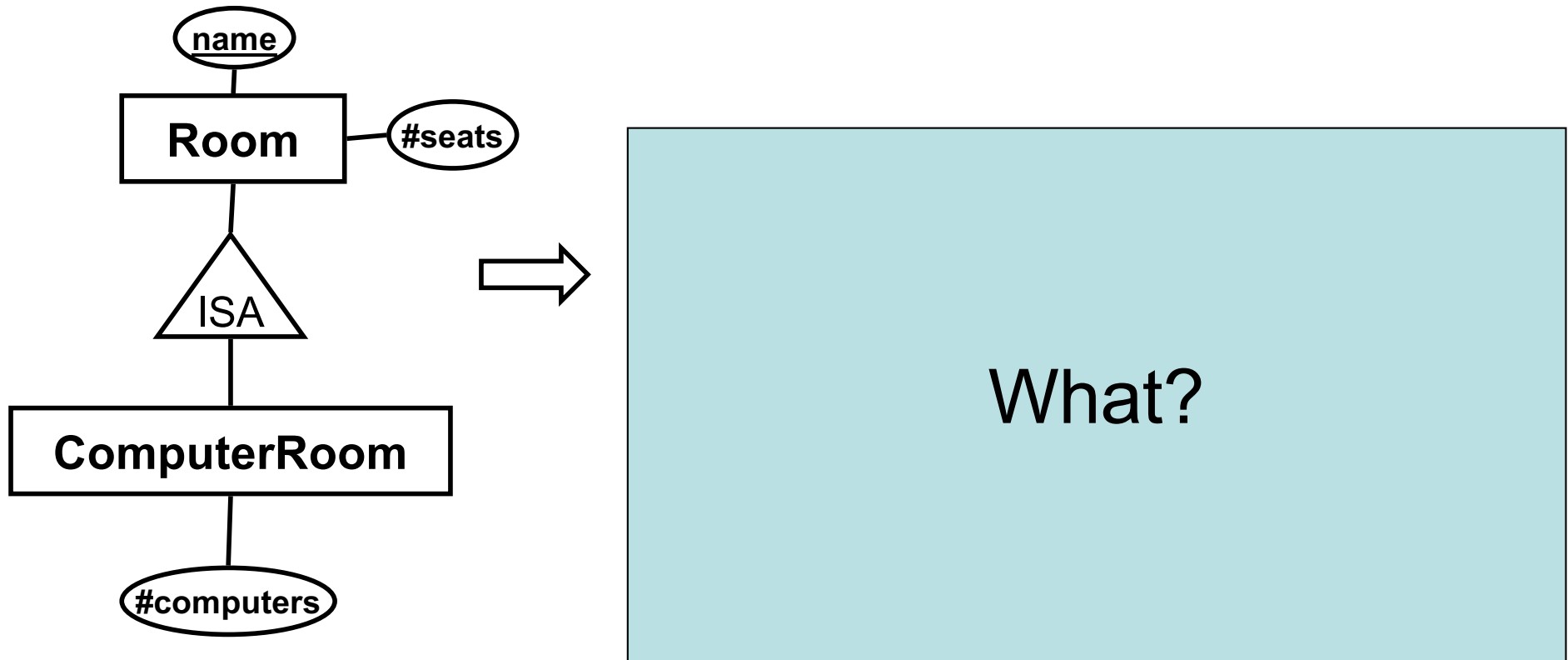
<u>name</u>	#seats
VR	216
ED6225	52

<u>name</u>	#computers
ED6225	26

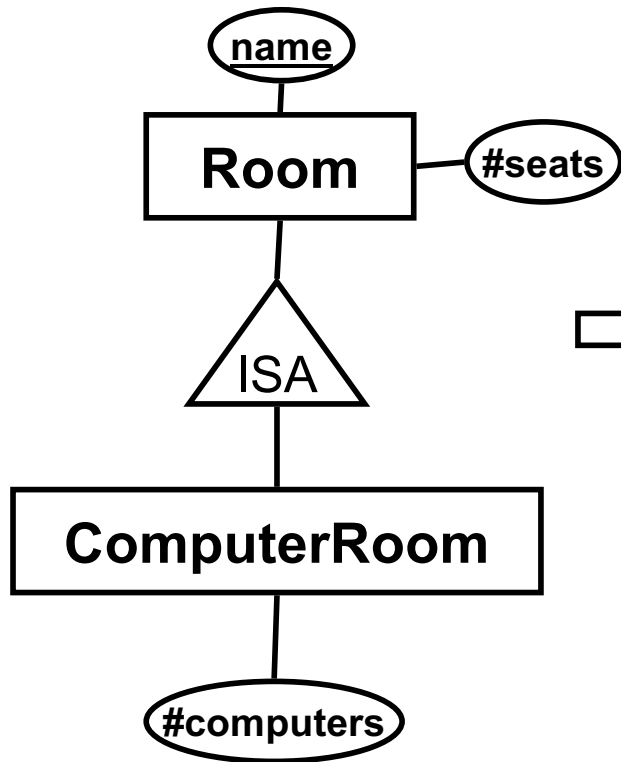
# Alternate ISA translations

- Two alternate approaches
  - *NULLs*: Join the subclass(es) with the superclass. Entities that are not part of the subclass use NULL for the attributes that come from the subclass.
  - *Object-oriented*: Each subclass becomes a relation with all the attributes of the superclass included. An entity belongs to either of the two, but not both.

# The NULLs approach:



# The NULLs approach:

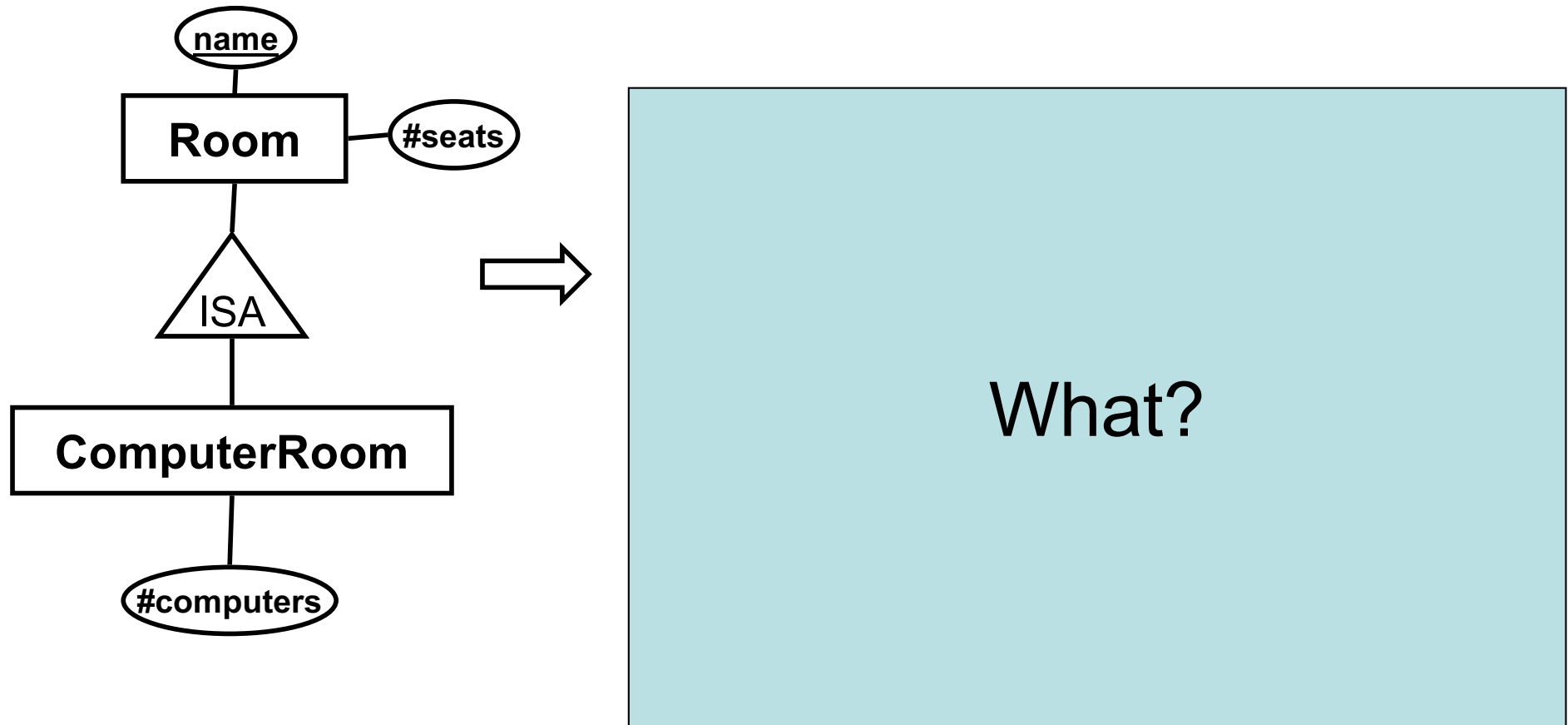


Rooms (name, #seats, #computers)

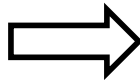
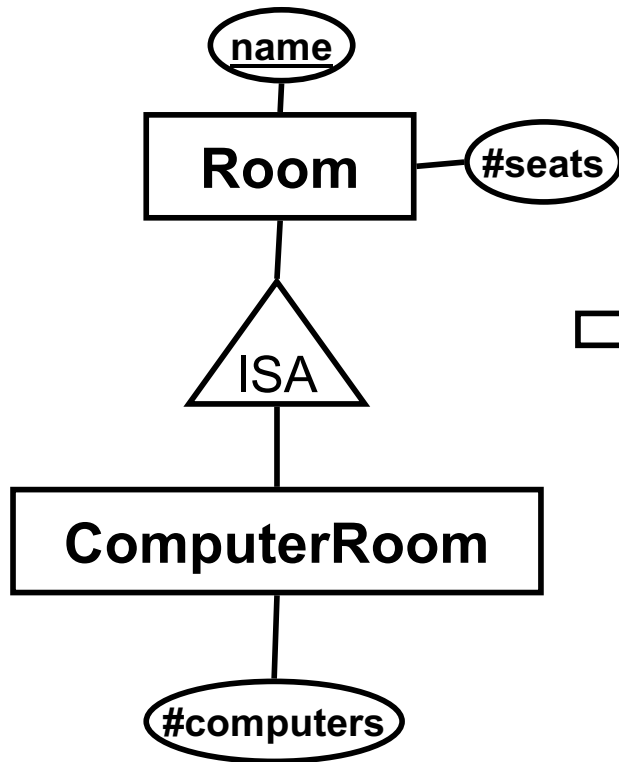
<u>name</u>	#seats	#computers
VR	216	NULL
ED6225	52	26



# The object-oriented (OO) approach:



# The object-oriented (OO) approach:



Rooms (name, #seats)  
ComputerRooms (name, #seats,  
#computers)

<u>name</u>	#seats
VR	216

<u>name</u>	#seats	#computers
ED6225	52	26

# Comparison – E-R

- E-R approach
  - Always works.
  - Use unless you have a good reason not to.

# Comparison – OO

- OO approach
  - Good when searching for general information about entities in a subclass only.
    - *"List the number of seats in all computer rooms"*
  - Does *not* work if superclass has any relationships.
    - An entity belonging to the subclass does not belong to the superclass as well, so foreign keys would have no single table to refer to.

# Comparison – NULLs

- NULLs approach
  - Could save space in situations where most entities in the hierarchy are part of the subclass (e.g. most rooms have computers in them).
  - Reduces the need for *joins*.
  - Not suited if subclass has any relationships.
    - Would lose the constraint that only the entities in the subclass can participate in the relationship.

# Weak entities

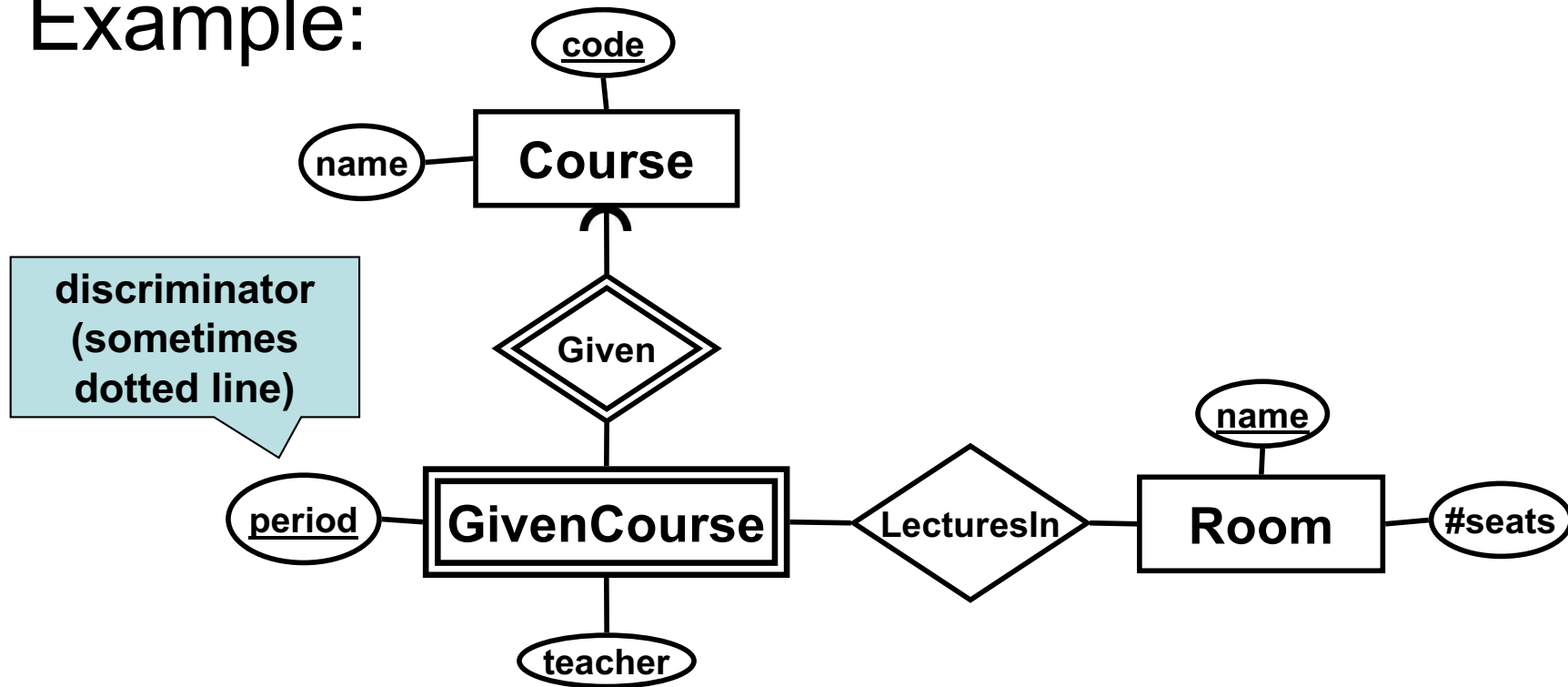
- Some entities depend on other entities.
  - A course is an entity with a code and a name.
  - A course does not have a teacher, rather it has a teacher for each time the course is given.
  - We introduce the concept of a given course, i.e. a course given in a particular period. A given course is a *weak entity*, dependent on the entity course. A given course has a teacher.

# Weak entities

- A *weak entity* is an entity that depends on another entity for help to be "uniquely" identified.
  - E.g. an airplane seat is identified by its number, but is not uniquely identified when we consider other aircraft. It depends on the airplane it is located in.
- Drawn as a rectangle with double borders.
- Related to its *supporting entity* by a *supporting relationship*, drawn as a diamond with double borders. This relationship is always many-to-"exactly one".

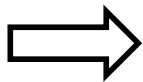
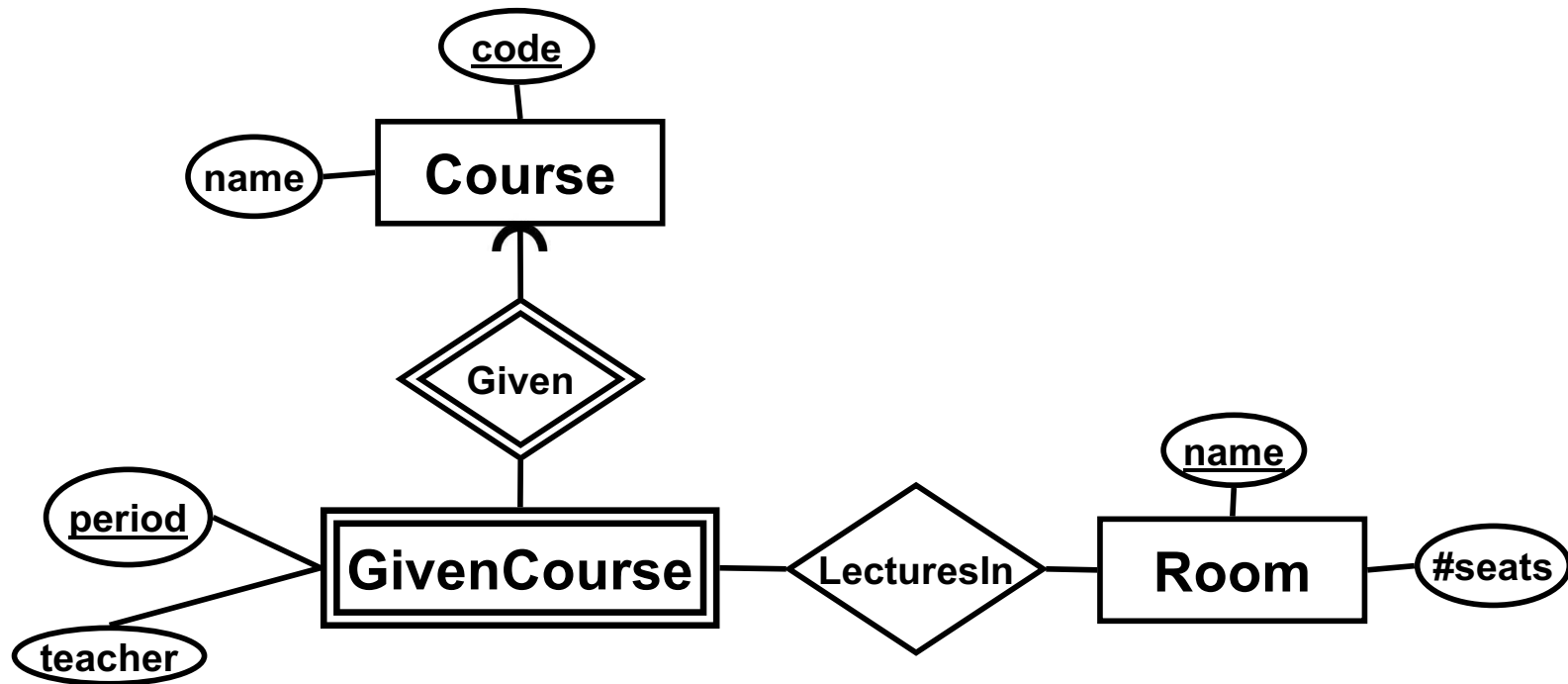
# Weak entities in E-R diagrams

Example:



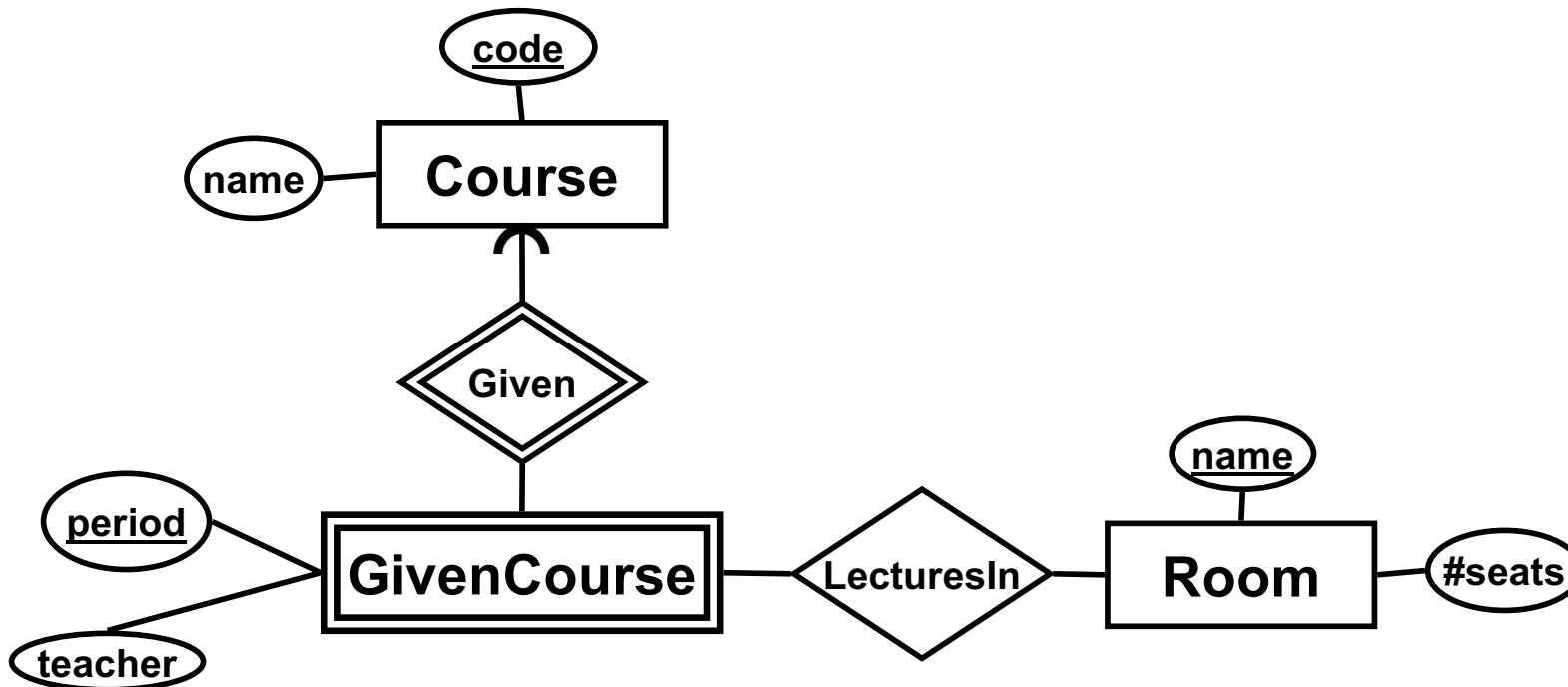


# Translating to relations:



What?

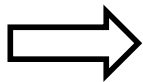
# Translating to relations:



`Courses (code, name)`

`GivenCourses (course, period, teacher)`

`course -> Courses.code`



`LecturesIn (course, period, room)`

`(course, period) -> GivenCourses.(course, period)`

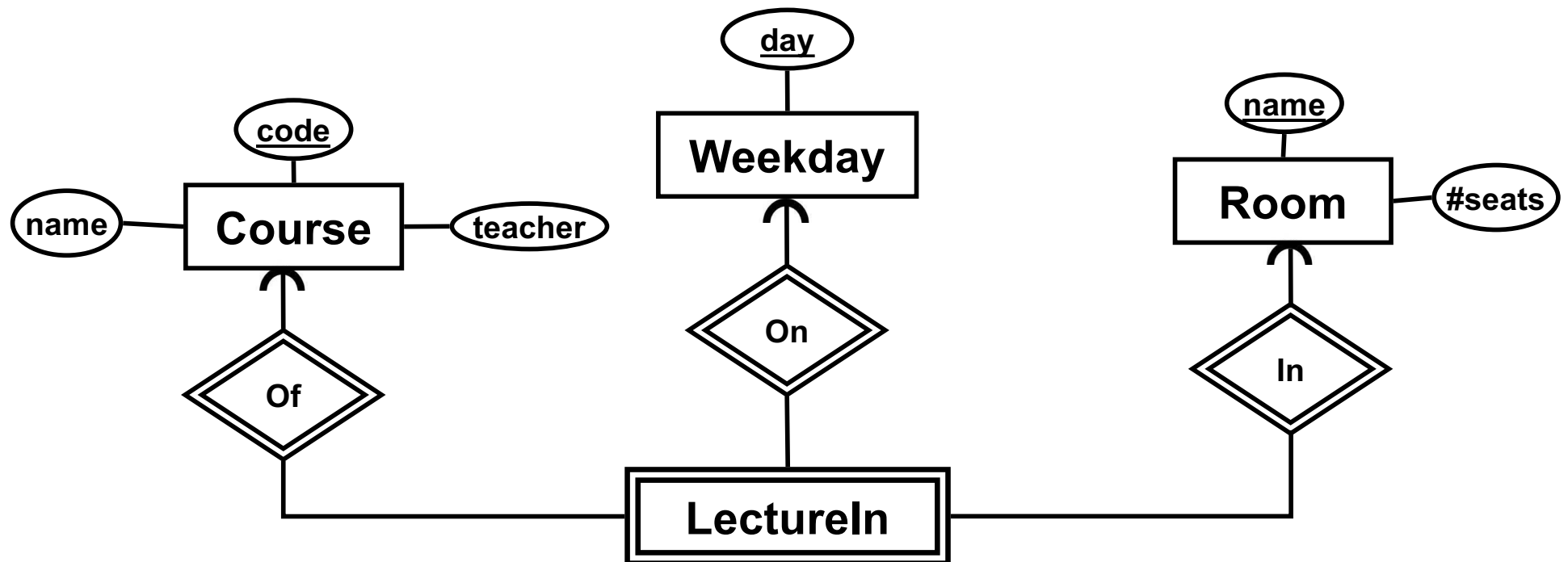
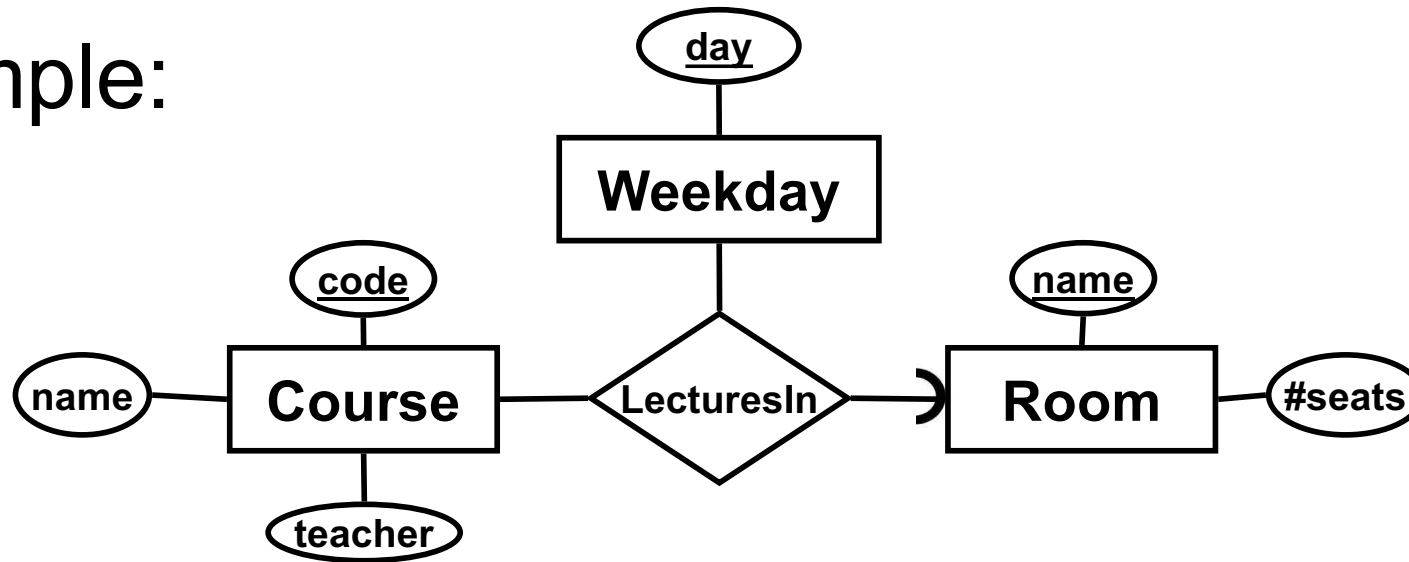
`room -> Rooms.name`

`Rooms (name, #seats)`

# Multiway relationships as WEs

- Multiway relationships can be transformed away using weak entities
  - Substitute the relationship with a weak entity.
  - Insert supporting relationships to all entities related as "many" by the original relationship.
  - Insert ordinary many-to-one relationships to all entities related as "one" by the original relationship.

Example:



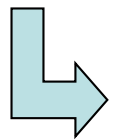
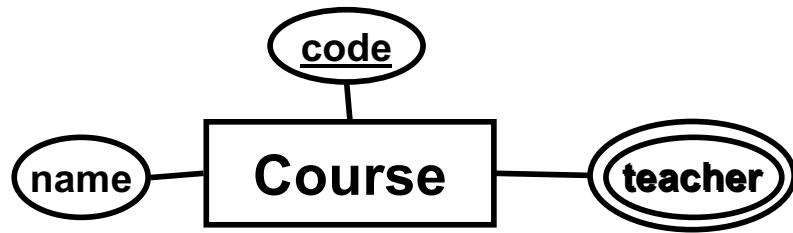
# What's the point?

- Usually, relationships work just fine, but in some special cases, you need a weak entity to express all multiplicity constraints correctly.
- A weak entity is needed when a **part** of an entity's key is a foreign key.

“Multivalued” attributes and “flag” attributes

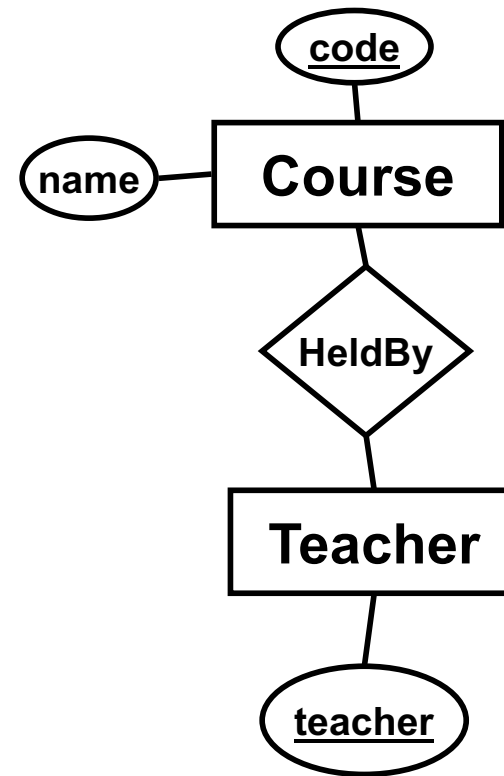
**THINGS NOT TO DO...**

# "Multivalued" attributes



```
Courses (code, name)  
HeldBy (code, teacher)  
code -> Courses.code
```

```
Courses (code, name)  
Teachers (teacher)  
HeldBy (code, teacher)  
code -> Courses.code  
teacher -> Teachers.teacher
```

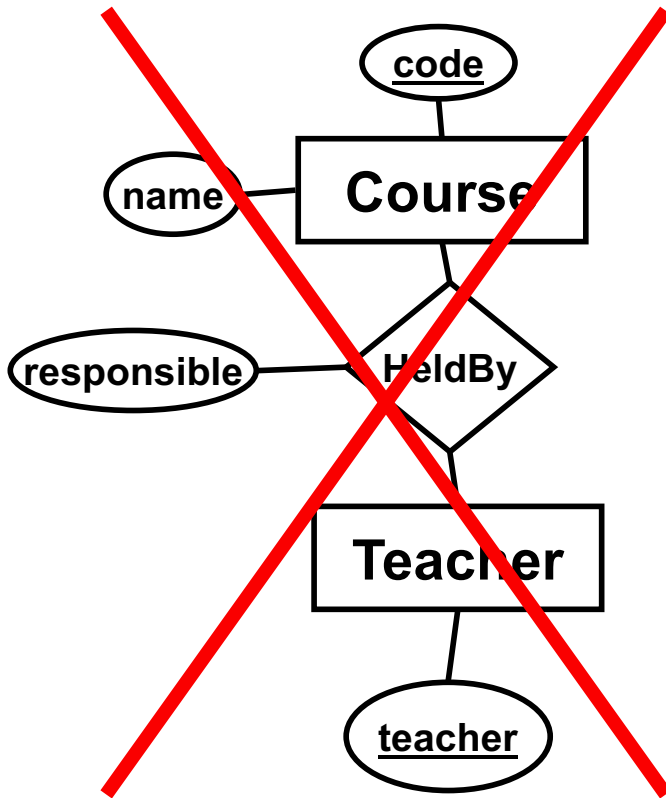


# ”Multivalued” attributes

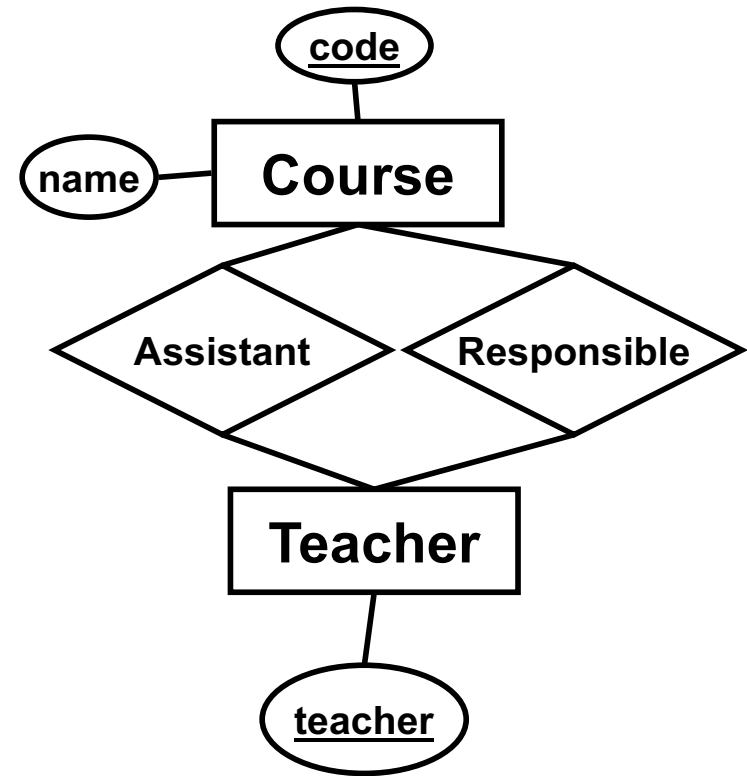
- Inflexible if you later want more attributes on teachers.
- No guarantees against e.g. spelling errors of teacher names.
  - less flexible to insert a constraint on what values are allowed than to use an extra table.
- Tables are cheap – references are cheap
  - No reason NOT to use an entity.
- Rule of thumb: Don't use multivalued attributes!!



# "Flag" attributes on relationships



VS.



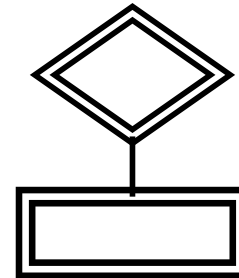
# ”Flag” attributes on relationships

- Less intuitively clear.
- Inflexible if later you need more roles.
- Tables are cheap, union of two tables is a cheap operation ( $O(1)$ ) – filtering can be expensive ( $O(n)$ )!
- Only benefit: automatic mutual exclusion (a teacher can only be *either* responsible *or* an assistant).
  - If important, can be recovered via assertions (costly).
- Rule of thumb: Don't use flag attributes on relationships!

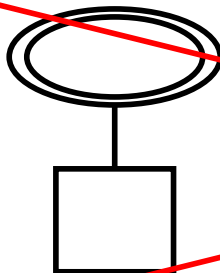
# ER cheatsheet 3



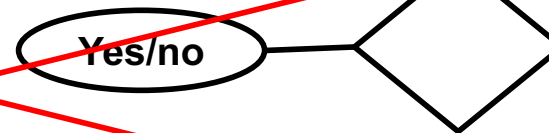
- Subclassing  
sub-entity extends super-entity
- ER-approach
  - NULL-approach
  - OO-approach



- Weak entities, identifying relationship  
Weak entity "is part of" entity
- Composite key with foreign key



"multivalued" attributes



"flag" attributes on relationships

Don't do this

Next time, lecture 3

Functional Dependencies

BCNF

3NF