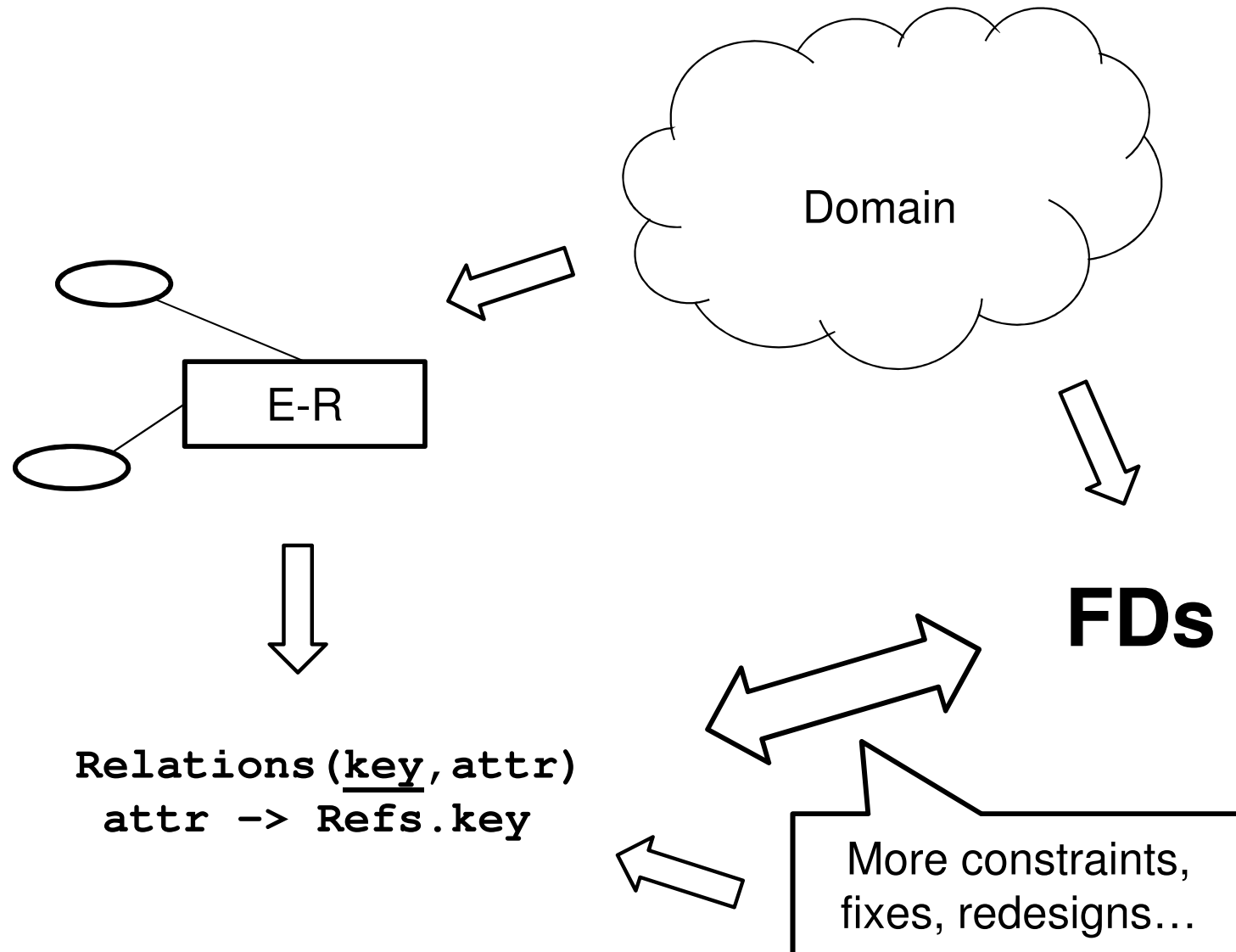


# Database design IV

INDs and 4NF  
Design wrapup

# The design process



# Work flow

- **DRAW** your diagram of the domain.
- **TRANSLATE** to relations forming a schema
- **IDENTIFY** dependencies from domain!
- **RELATE** the dependencies to the schema, to find more constraints, and to validate your design.

# How NOT to find FDs

- Do an E-R diagram, translate, look at the keys of relations, pick the proper FDs.

Quiz: Why not?

- Extra constraints: You can't find those "extras" in the core structure!
- Validation: If the FDs are taken from the diagram, it will contain the same errors!

# Work flow cont.

- Draw your diagram and translate to relations: You now have the basic structure for your database.
- Find extra constraints from the domain, via FDs, to complete the design.
- In simple cases there might be no extras to find, but in general there will be:

**Both steps are needed!**

# Lab Assignment

- Write a "student portal" application in Java
  - Part I: **Design**
    - Given a domain description, design a database schema using an E-R diagram.
  - Part II: **Design**
    - Given a domain description, find and act on the functional dependencies of the domain to fix the schema from Part I.
  - Part III: **Construction** and **Usage**
    - Implement the schema from Part II in Oracle.
    - Insert relevant data.
    - Create views to support key operations.
  - Part IV: **Construction**
    - Create triggers to support key operations.
  - Part V: Interfacing from external **Application**
    - Write a Java application that uses the database from Part III.

# Part II – Dependencies

- Identify the functional dependencies that you expect should hold for the domain.
- Identify any extra constraints required.
- Improve the schema from task 1 so that all constraints are captured (or argue why you cannot capture them).

# Part II – Dependencies

- Hand in:
  - a list of functional dependencies
  - a list of extra constraints, with motivation
  - an updated schema from task 1.
- Submission deadline: Fri, Feb 7 (23:59)



# Quiz time!

What's wrong with this schema?

`Courses (code, period, name, teacher)`

`code → name`

`code, period → teacher`

```
{ ('TDA357', 3, 'Databases', 'Niklas Broberg'),  
  ('TDA357', 2, 'Databases', 'Graham Kemp') }
```

**Redundancy!**

# Using FDs to detect anomalies

- Whenever  $X \rightarrow A$  holds for a relation  $R$ , but  $X$  is not a key for  $R$ , then values of  $A$  will be redundantly repeated!

**Courses (code, period, name, teacher)**

```
{ ('TDA357', 3, 'Databases', 'Niklas Broberg'),  
  ('TDA357', 2, 'Databases', 'Graham Kemp') }
```

`code`  $\rightarrow$  `name`

`code, period`  $\rightarrow$  `teacher`

# Decomposition

**Courses (code, period, name, teacher)**

**code → name**

**code, period → teacher**

- Fix the problem by decomposing Courses:
  - Create one relation with the attributes from the offending FD, in this case **code** and **name**.
  - Keep the original relation, but remove all attributes from the RHS of the FD. Insert a reference from the LHS in this relation, to the key in the first.

**Courses (code, name)**

**GivenCourses (code, period, teacher)**

**code → Courses.code**

# Boyce-Codd Normal Form

- A relation  $R$  is in Boyce-Codd Normal Form (BCNF) if, whenever a nontrivial FD  $X \rightarrow A$  holds on  $R$ ,  $X$  is a superkey of  $R$ .
  - Remember: nontrivial means  $A$  is not part of  $X$
  - Remember: a superkey is any superset of a key (including the keys themselves).

**Courses** (code, name)

**GivenCourses** (code, period, teacher)

# 3NF vs BCNF

- Three important properties of decomposition:
  1. Recovery (loss-less join)
  2. No redundancy
  3. Dependency preservation
- 3NF guarantees 1 and 3, but not 2.
- BCNF guarantees 1 and (almost) 2, but not 3.

# Almost?

Example:

Courses (code, name, room, teacher)

code → name

<u>code</u>	name
TDA357	Databases

<u>code</u>	<u>room</u>	<u>teacher</u>
TDA357	VR	Niklas Broberg
TDA357	VR	Graham Kemp
TDA357	HC1	Niklas Broberg
TDA357	HC1	Graham Kemp

These two relations are in BCNF, but there's lots of redundancy!

# Quiz time!

What's wrong with this schema?

CourseInfo (code, room, teacher)

(No FDs!)

<u>code</u>	<u>room</u>	<u>teacher</u>
TDA357	VR	Niklas Broberg
TDA357	VR	Graham Kemp
TDA357	HC1	Niklas Broberg
TDA357	HC1	Graham Kemp

**Redundancy!**

# Let's start from the bottom...

<u>code</u>	<u>room</u>
TDA357	HC1
TDA357	VR

<u>code</u>	<u>teacher</u>
TDA357	Niklas Broberg
TDA357	Graham Kemp



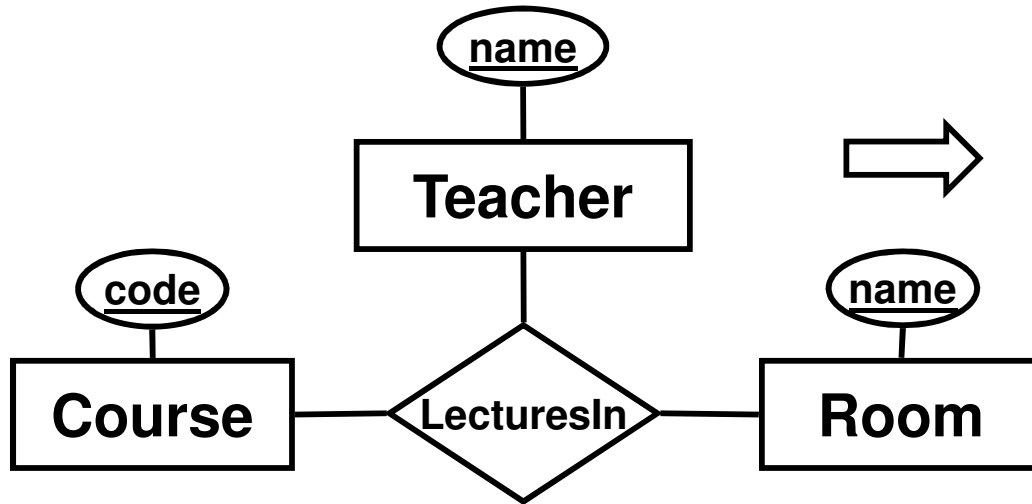
<u>code</u>	<u>room</u>	<u>teacher</u>
TDA357	VR	Niklas Broberg
TDA357	VR	Graham Kemp
TDA357	HC1	Niklas Broberg
TDA357	HC1	Graham Kemp



- No redundancy before join
- The two starting tables are what we really want to have

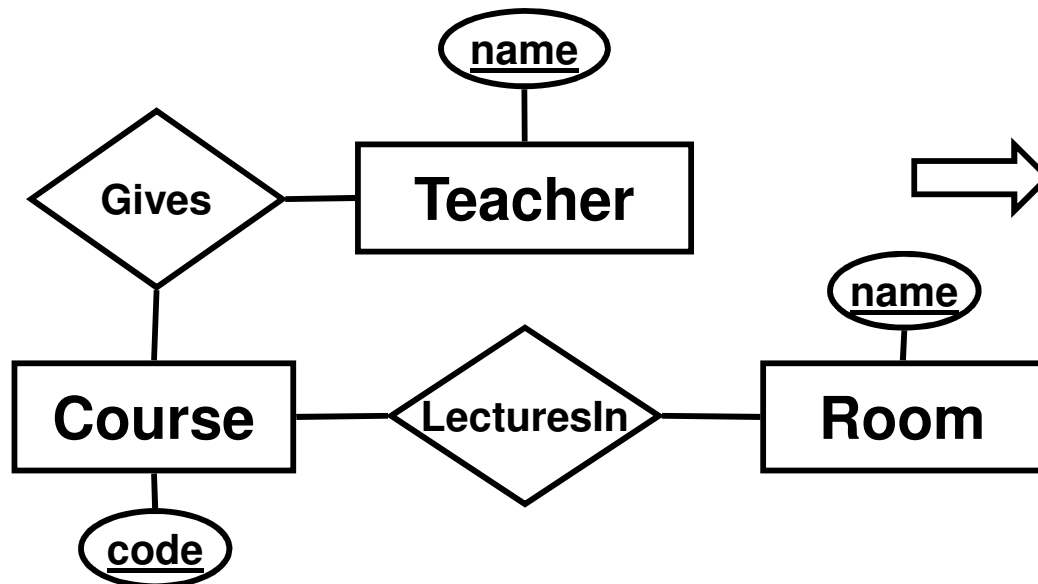


# Compare with E/R



→

```
LecturesIn(code, teacher, room)  
code -> Courses.code  
room -> Rooms.name  
teacher -> Teachers.name
```



→

```
LecturesIn(code, room)  
code -> Courses.code  
room -> Rooms.name  
Gives(code, teacher)  
code -> Courses.code  
teacher -> Teachers.name
```

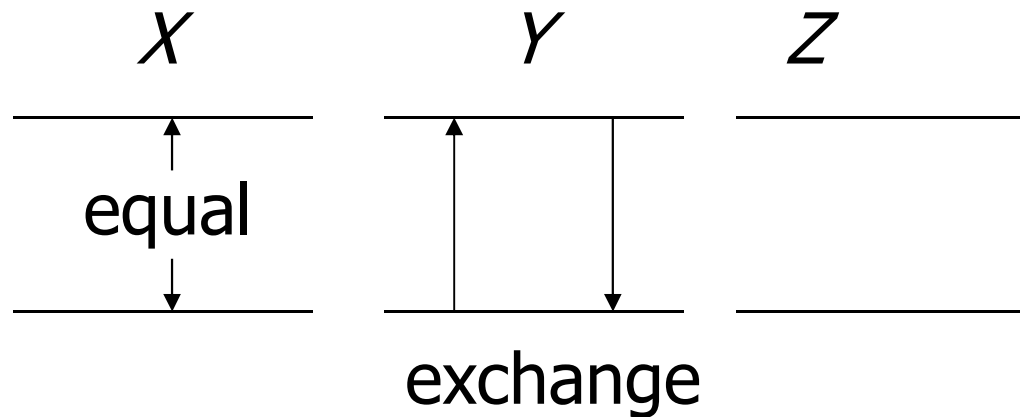
# Independencies (INDs)

- Some attributes are not uniquely defined (as with FDs), but are still independent of the values of other attributes.
  - In our example: code does not determine room, there can be several rooms for a course. But the rooms a course uses is *independent* of the teachers on the course.
- $X \twoheadrightarrow Y \mid Z$  states that from the point of view of X, Y and Z are independent.
  - Just  $X \twoheadrightarrow Y$  means that X's relationship to Y is independent of all other attributes.

# Independent how?

- An IND  $X \twoheadrightarrow Y$  is an assertion that if two tuples of a relation agree on all the attributes of  $X$ , then their components in the set of attributes  $Y$  may be swapped, and the result will be two tuples that are also in the relation.
- If (for some  $X$ ) all values of  $Y$  (for that  $X$ ) can be combined with all values of  $Z$  (for that  $X$ ), then (from  $X$ )  $Y$  and  $Z$  are independent.

# Picture of IND $X \twoheadrightarrow Y / Z$



If two tuples have the same value for  $X$ , different values for  $Y$  and different values for the  $Z$  attributes, then there must also exist tuples where the values of  $Y$  are exchanged, otherwise  $Y$  and  $Z$  are not independent!

# Implied tuples

**Courses (code, name, room, teacher)**

**code  $\rightarrow$  name**

**code  $\twoheadrightarrow$  room | teacher**

If we have:

<u>code</u>	name	<u>room</u>	<u>teacher</u>
TDA357	Databases	VR	Niklas Broberg
TDA357	Databases	HC1	Graham Kemp

we must also have:

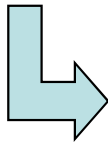
TDA357	Databases	VR	Graham Kemp
TDA357	Databases	HC1	Niklas Broberg

otherwise room and teacher would not be independent!

# Compare with joining

<u>code</u>	<u>room</u>
TDA357	HC1
TDA357	VR

<u>code</u>	<u>teacher</u>
TDA357	Niklas Broberg
TDA357	Graham Kemp



<u>code</u>	<u>room</u>	<u>teacher</u>
TDA357	VR	Niklas Broberg
TDA357	VR	Graham Kemp
TDA357	HC1	Niklas Broberg
TDA357	HC1	Graham Kemp



- Joining two independent relations yields a relation with all combinations of values!

# FDs are INDs

- Every FD is an IND (but of course not the other way around). Compare the following cases:
  - If  $X \twoheadrightarrow Y$  holds for a relation, then all possible values of  $Y$  for that  $X$  must be combined with all possible combinations of values for "all other attributes" for that  $X$ .
  - If  $X \rightarrow A$ , there is only one possible value of  $A$  for that  $X$ , and it will appear in all tuples where  $X$  appears. Thus it will be combined with all combinations of values that exist for that  $X$  for the rest of the attributes.

# Example:

<u>code</u>	<i>name</i>	<u>room</u>	<u>teacher</u>
TDA357	Databases	VR	Niklas Broberg
TDA357	Databases	VR	Graham Kemp
TDA357	Databases	HC1	Niklas Broberg
TDA357	Databases	HC1	Graham Kemp

**code** → **name**

There are four possible combinations of values for the attributes **room** and **teacher**, and the only possible value for the **name** attribute, "Databases", appears in combination with all of them.

**code** → **teacher**

There are two possible combinations of values for the attributes **name** and **room**, and all possible values of the attribute **teacher** appear with both of these combinations.

**code** → **room**

There are two possible combinations of values for the attributes **name** and **teacher**, and all possible values of the attribute **room** appear with both of these combinations.



# IND rules $\neq$ FD rules

- Complementmentation
  - If  $X \twoheadrightarrow Y$ , and  $Z$  is all other attributes, then  $X \twoheadrightarrow Z$ .
- Splitting doesn't hold!!
  - **code  $\twoheadrightarrow$  room, #seats**
    - **code  $\twoheadrightarrow$  room** does not hold, since **room** and **#seats** are not independent!
- None of the other rules for FDs hold either.

# Example:

<u>code</u>	<i>name</i>	<u>room</u>	<i>#seats</i>	<u>teacher</u>
TDA357	Databases	VR	216	Niklas Broberg
TDA357	Databases	VR	216	Graham Kemp
TDA357	Databases	HC1	126	Niklas Broberg
TDA357	Databases	HC1	126	Graham Kemp

**code**  $\rightarrow$  **room, #seats**

We cannot freely swap values in the **#seats** and **room** columns,  
so neither

**code**  $\rightarrow$  **room**

or

**code**  $\rightarrow$  **#seats**

holds.

# Fourth Normal Form

- The redundancy that comes from IND's is not removable by putting the database schema in BCNF.
- There is a stronger normal form, called 4NF, that (intuitively) treats IND's as FD's when it comes to decomposition, but not when determining keys of the relation.

# Fourth Normal Form (4NF)

- 4NF is a strengthening of BCNF to handle redundancy that comes from independence.
  - An IND  $X \twoheadrightarrow Y$  is trivial for R if
    - Y is a subset of X
    - X and Y together = R
  - Non-trivial  $X \rightarrow A$  violates BCNF for a relation R if X is not a superkey.
  - Non-trivial  $X \twoheadrightarrow Y$  violates 4NF for a relation R if X is not a superkey.
    - Note that what is a superkey or not is still determined by FDs only.

# BCNF Versus 4NF

- Remember that every FD  $X \rightarrow Y$  is also a IND,  $X \twoheadrightarrow Y$ .
- Thus, if R is in 4NF, it is certainly in BCNF.
  - Because any BCNF violation is a 4NF violation.
- But R could be in BCNF and not 4NF, because IND's are “invisible” to BCNF.

# INDs for validation

- Remember that FDs can:
  - Allow you to validate your schema.
  - Find "extra" constraints that the basic structure doesn't capture.
- INDs ONLY validate your schema.
  - No extra dependencies to be found.
  - If your E-R diagram and translation are correct, INDs don't matter.

# The whole truth(?)...

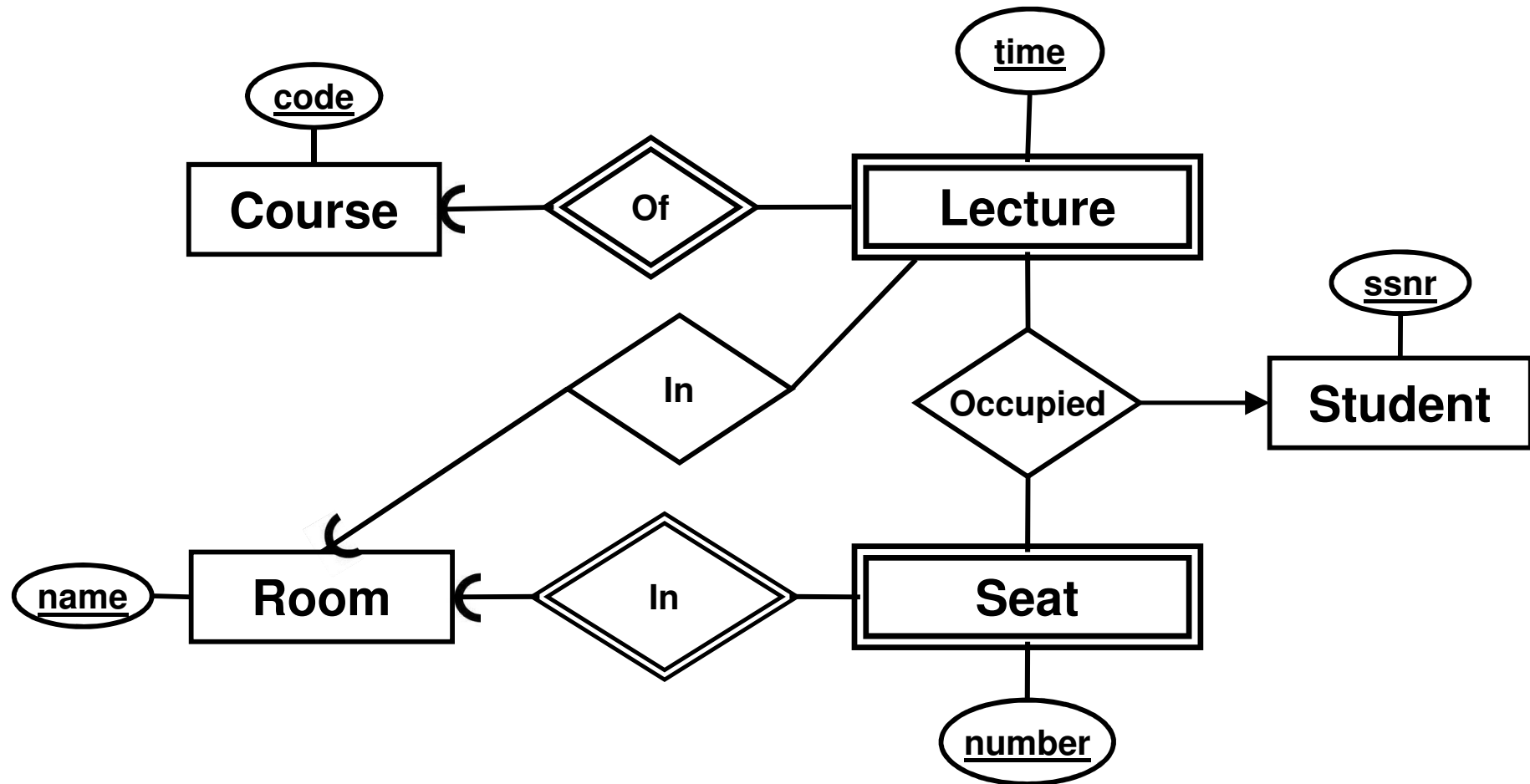
- Independencies are called Multi-valued dependencies (??!) in literature (including the course book).
- ... technically they *are* dependencies of a sorts, but that's really only relevant for abstract mathematics, not databases. The important thing about them is the independence between attributes.
- So, forget you ever heard the name MVDs. Repeat after me, INDependencies...

# First and second?

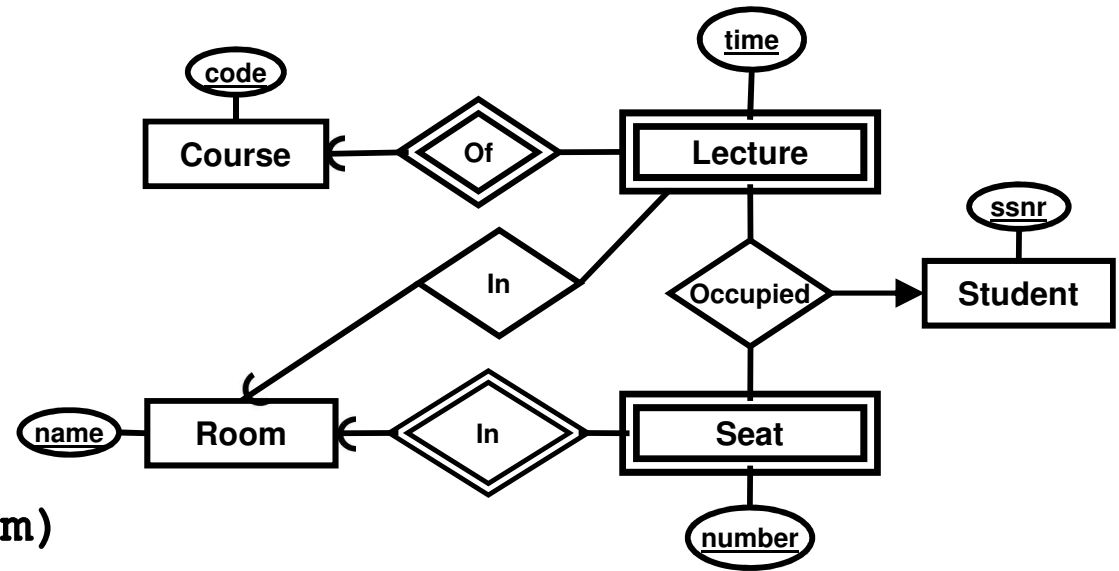
- Only interesting as a theoretical foundation, never used in practice.
  - First Normal Form (1NF): All attributes are atomic (no attributes are sets of values).
  - Second Normal Form (2NF): Some simple forms of redundancy removed.
- There are even more restrictive forms than 4NF, but these are rarely (if ever) used.



# Example: E-R does not imply BCNF



Students (ssnr)  
 Courses (code)  
 Rooms (name)  
 Lectures (course, time, room)  
     course -> Courses.code  
     room -> Rooms.name  
 Seats (room, number)  
     room -> Rooms.name  
 Occupied (course, time, room, number, student)  
     (course, time) -> Lectures.(course, time)  
     (room, number) -> Seats.(room, number)  
     student -> Students.ssnr



**Redundancy!**

Quiz: What just went wrong?

# Fix attempt #1

Students (ssnr)

Courses (code)

Rooms (name)

Lectures (course, time, room)

course -> Courses.code

room -> Rooms.name

Seats (room, number)

room -> Rooms.name

Occupied (course, time, number, student)

(course, time) -> Lectures.(course, time)

student -> Students.ssnr

(room, number) -> Seats.(room, number) ??

We broke the reference! Now we could (in theory) book seats that don't exist in the room where the lecture is given!

# Fix attempt #2

Students (ssnr)

Courses (code)

Rooms (name)

Lectures (course, time, room)

course -> Courses.code

room -> Rooms.name

Seats (room, number)

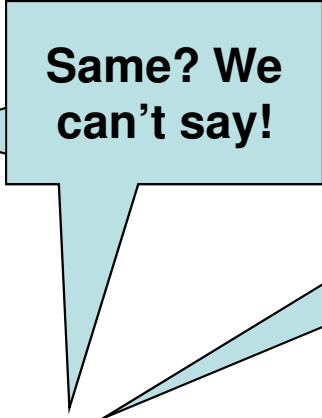
room -> Rooms.name

Occupied (course, time, number, room, student)

(course, time) -> Lectures.(course, time)

(room, number) -> Seats.(room, number)

student -> Students.ssnr



Same? We can't say!



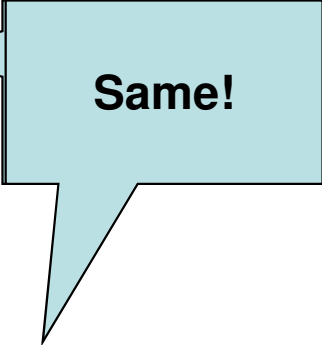
No longer part of key!

No guarantee that the hall where the seat is booked is the same hall that the movie is shown in!

... and redundancy (3NF solution)

# Fix attempt #3

```
Students (ssnr)
Courses (code)
Rooms (name)
Lectures (course, time, room)
    course -> Courses.code
    room   -> Rooms.name
Seats (room, number)
    room   -> Rooms.name
Occupied (course, time, number, room, student)
    (course, time, room) ->
        Lectures.(course, time, room)
    (room, number) -> Seats.(room, number)
    student       -> Students.ssnr
```

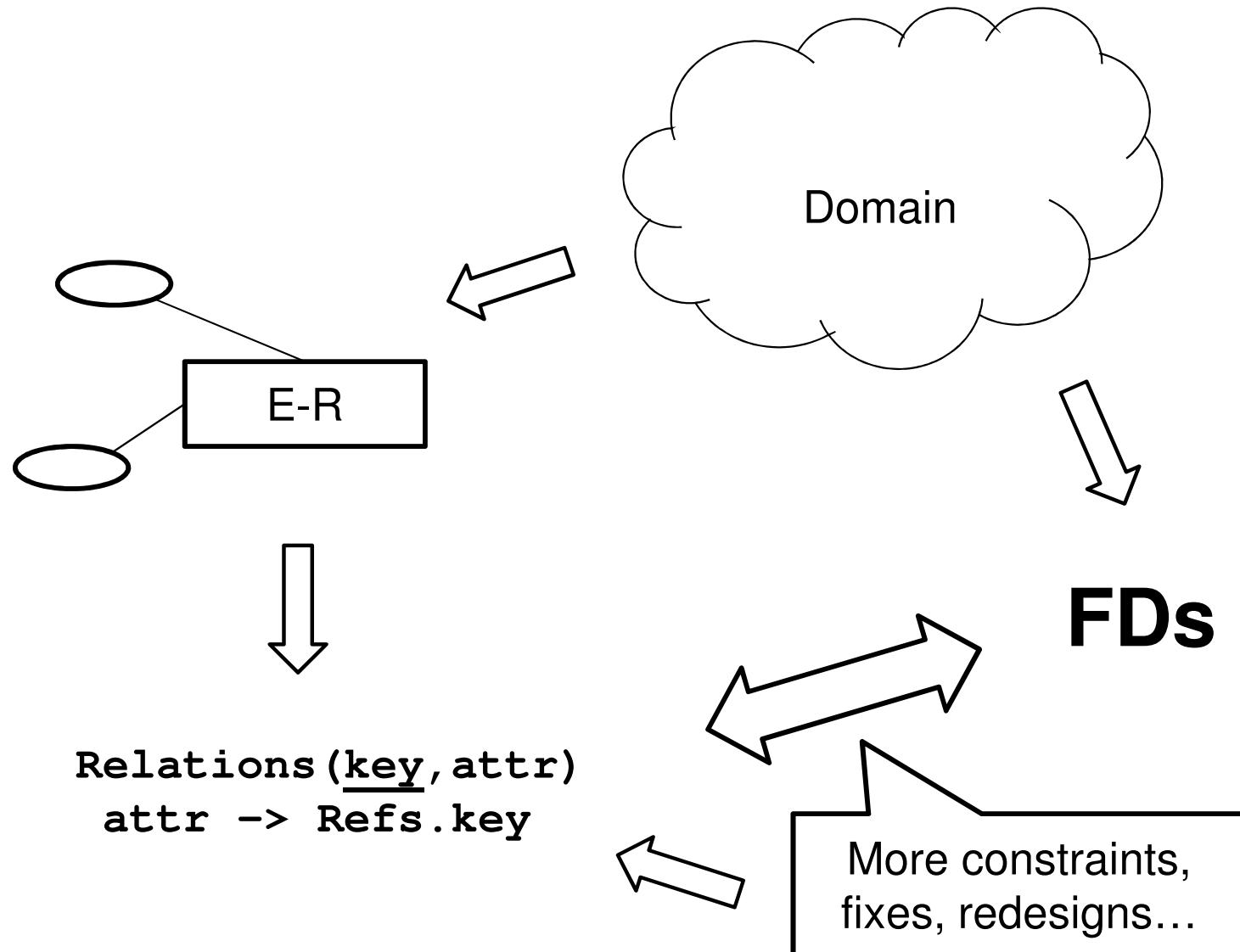


Still redundancy though (3NF solution). Possibly the best we can do though.

# Moral of the story

- E-R diagrams don't always yield a schema that captures even the "core" constraints, or guarantee any particular normal form.
- FDs must be taken into account!

# The design process revisited



# Course Objectives – Design

When the course is through, you should

- Given a domain, know how to design a database that correctly models the domain and its constraints.

*”We want a database that we can use for scheduling courses and lectures. This is how it’s supposed to work: ...”*



# Exam – FDs and NFs (12)

*“A car rental company has the following, not very successful, database. They want your help to improve it. ...”*

- Identify all functional dependencies you expect to hold in the domain.
- Indicate which of those dependencies violate BCNF with respect to the relations in the database.
- Do a complete decomposition of the database so that the resulting relations are in BCNF.

# Next Lecture

Database Construction –  
SQL Data Definition Language