HALMSTAD UNIVERSITY

DS4001 Databases (7.5 credits)

# Lecture 9 – Schema & FDs

Yuantao Fan
yuantao.fan@hh.se

Halmstad University

# Overview

- Recall ER-models
- ER-diagrams and Schema
- Functional dependencies
  - Indicate redundancy
  - Identifying FDs
  - Inferring FDs

# Recall ER Model

- Entity sets
  - The “things/objects/concept” that possess information
  - Exist independent from other entities (generaly speaking), except for weak entities
  - E.g. Students, Teachers, Courses, Employees, Companies etc.

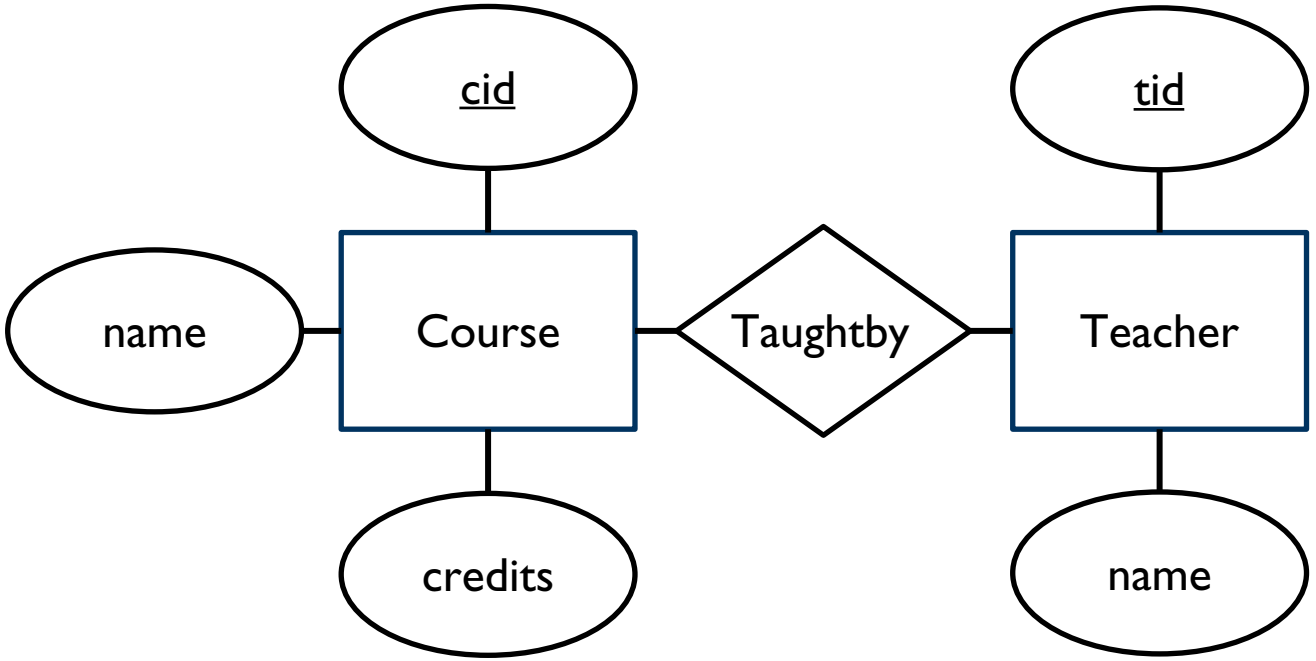
- Relationships
  - How entities are related to each other

- Attributes
  - The information, values that will be stored into tables in the database
  - Both relationships and entities can have attributes
  - E.g. Teachers/Students/Employees’s name, age nationality…

# ER Diagrams and Schemas

- Entities are named singular, while relations are in plural

- Database schema
  - “blueprint” of a database which describes how the data may relate to other tables or other data models
  - Composed of entities, relationships, and their attributes, including keys

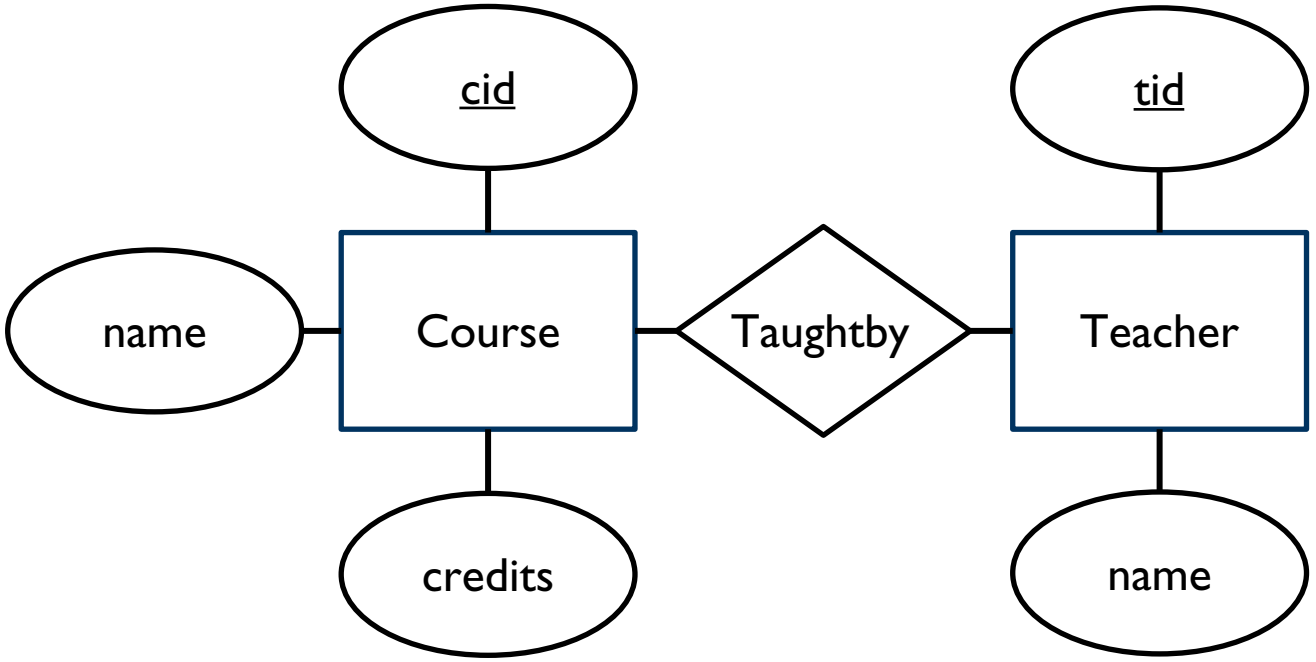


# ER Diagrams and Schemas

- Entities are named singular, while relations are in plural

- Database schema
  - "blueprint" of a database which describes how the data may relate to other tables or other data models
  - Composed of entities, relationships, and their attributes, including keys

*TaughtBy(course, teacher)*
*course -> Courses.cid*
*teacher -> Teachers.tid*




*Courses(cid, name, credits)*

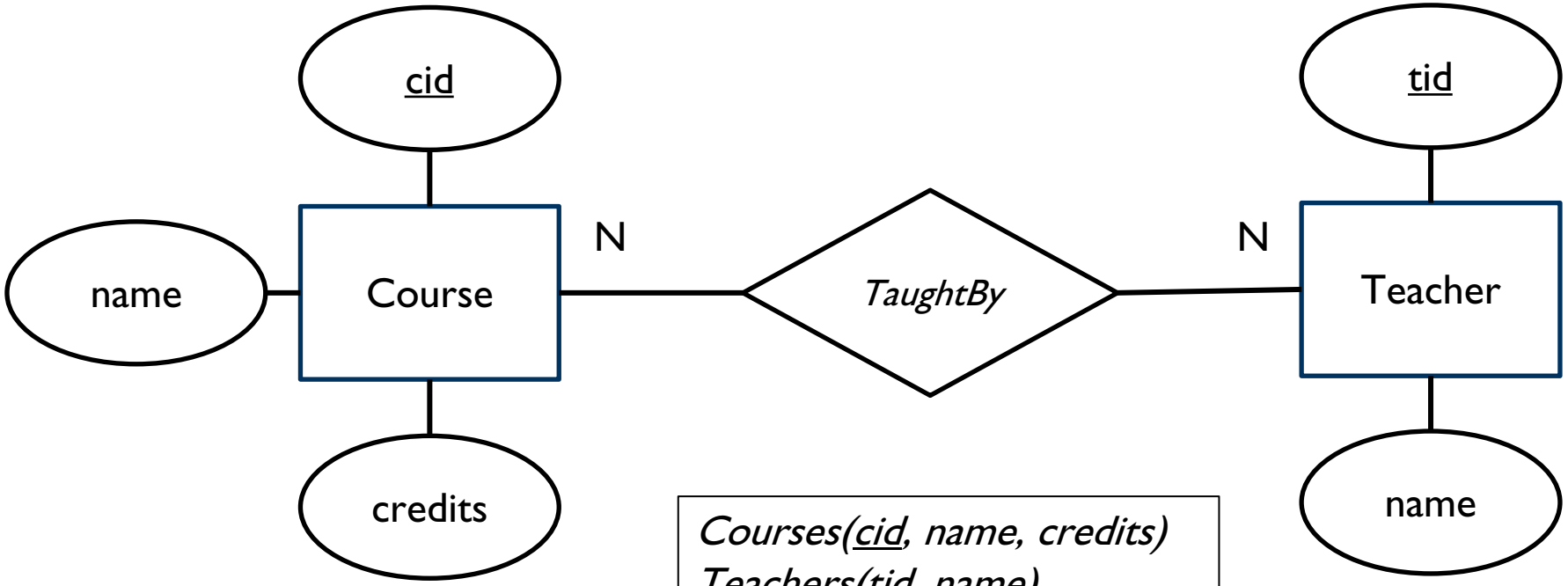
*Teachers(tid, name)*

```
CREATE TABLE Courses (
        cid CHAR(6) PRIMARY KEY,
        name TEXT NOT NULL,
        credits FLOAT NOT NULL );
```

```
CREATE TABLE Teachers (
        tid CHAR(6) PRIMARY KEY,
        name TEXT NOT NULL);
```

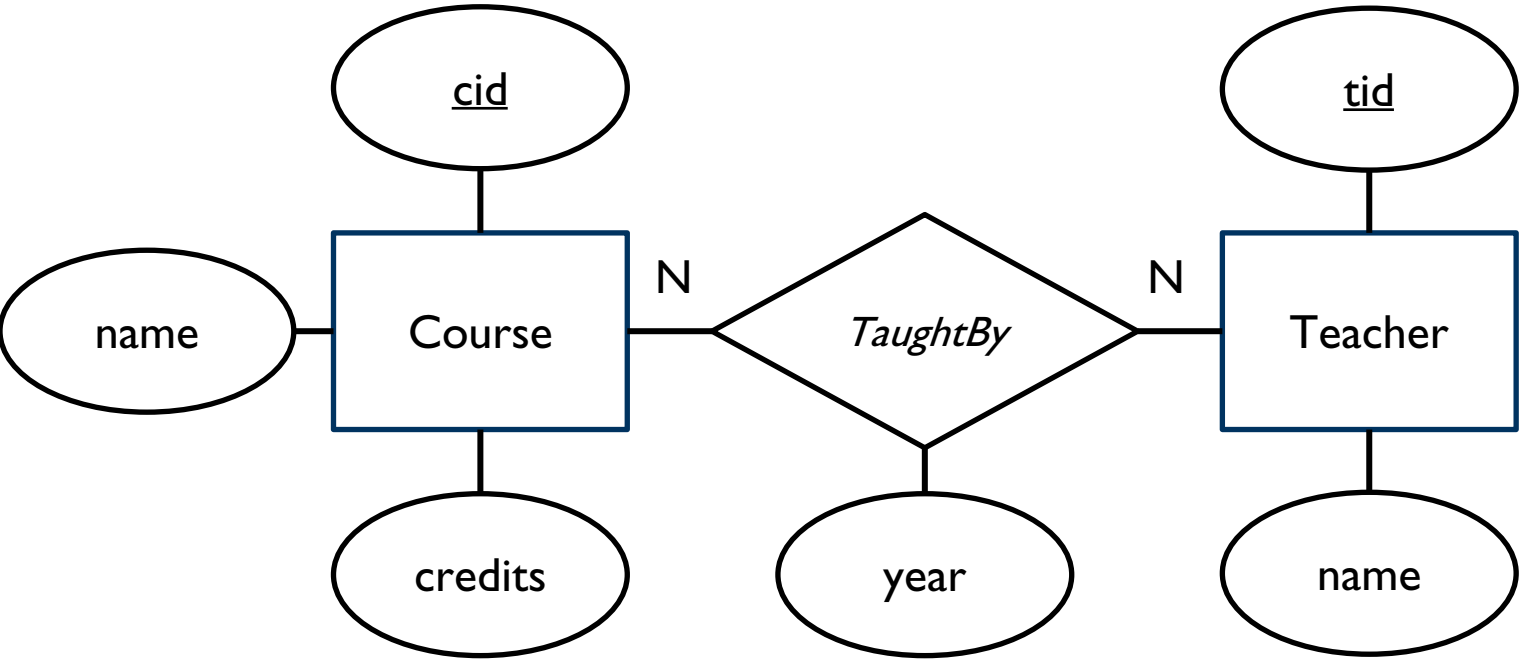
# ER Diagrams and Schemas

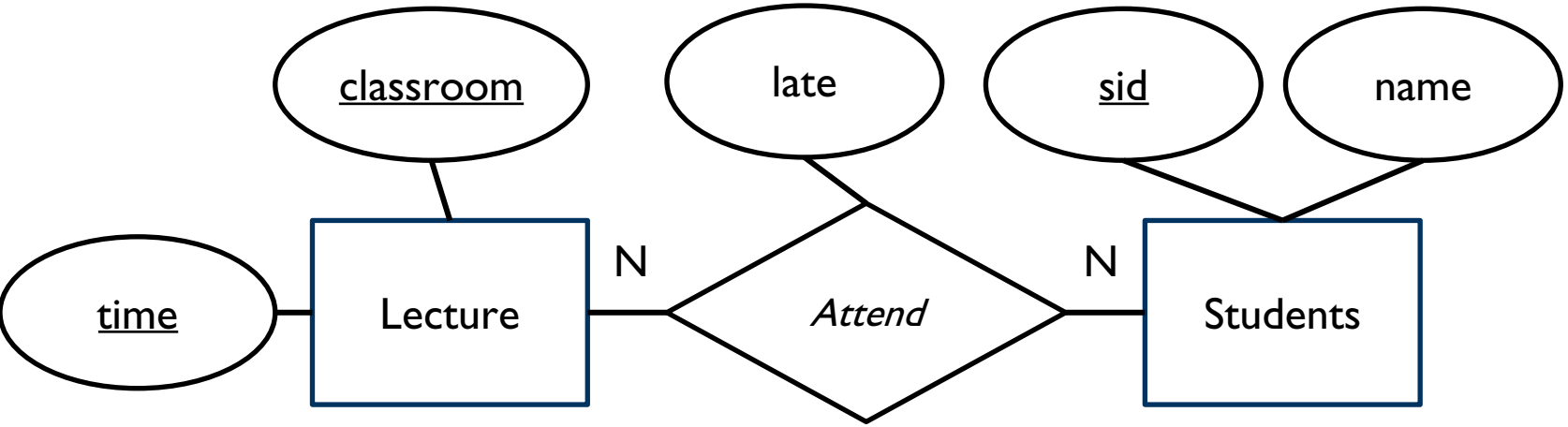
Entities are named singular, while relations are in plural
Relationships in diamond-shpes



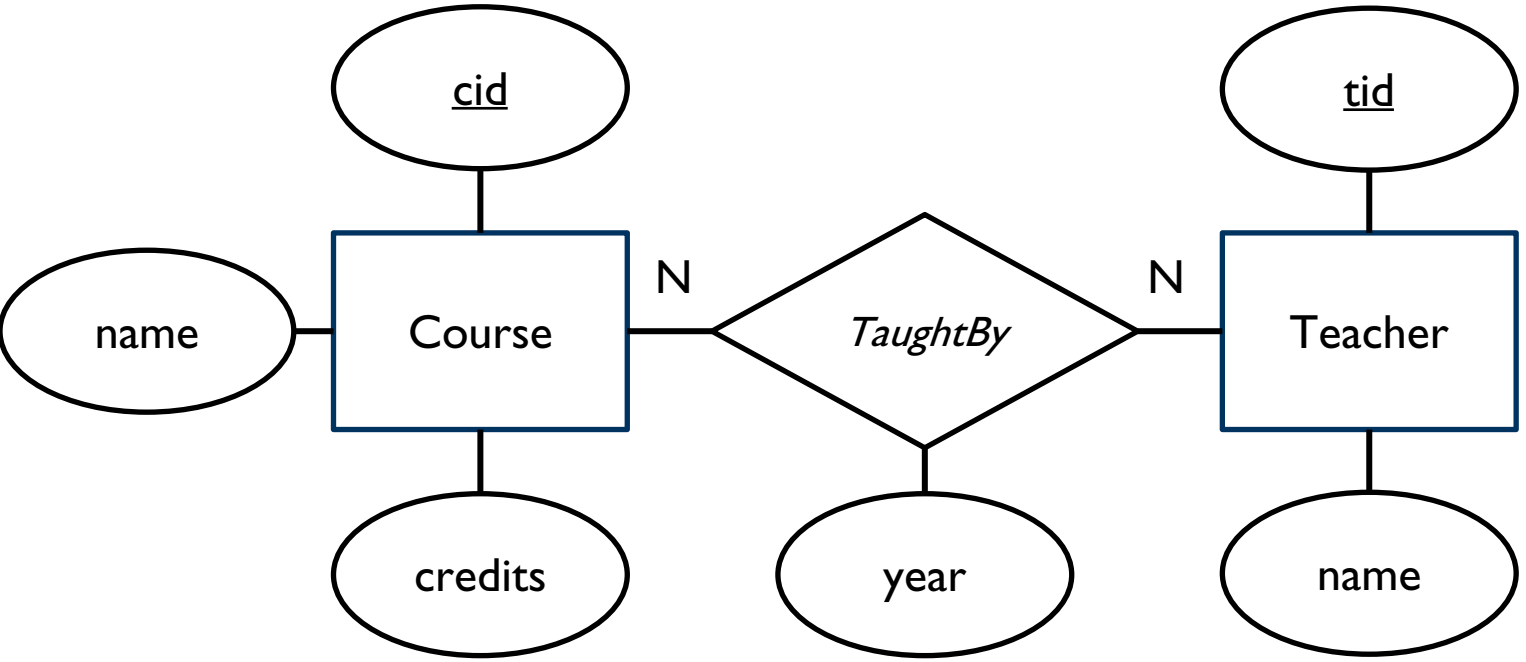

*Courses(<u>cid</u>, name, credits)*
*Teachers(<u>tid</u>, name)*
*TaughtBy(<u>course</u>, <u>teacher</u>)*
*course -> Courses.cid*
*teacher -> Teachers.tid*

# Many-to-many Relationships Examples

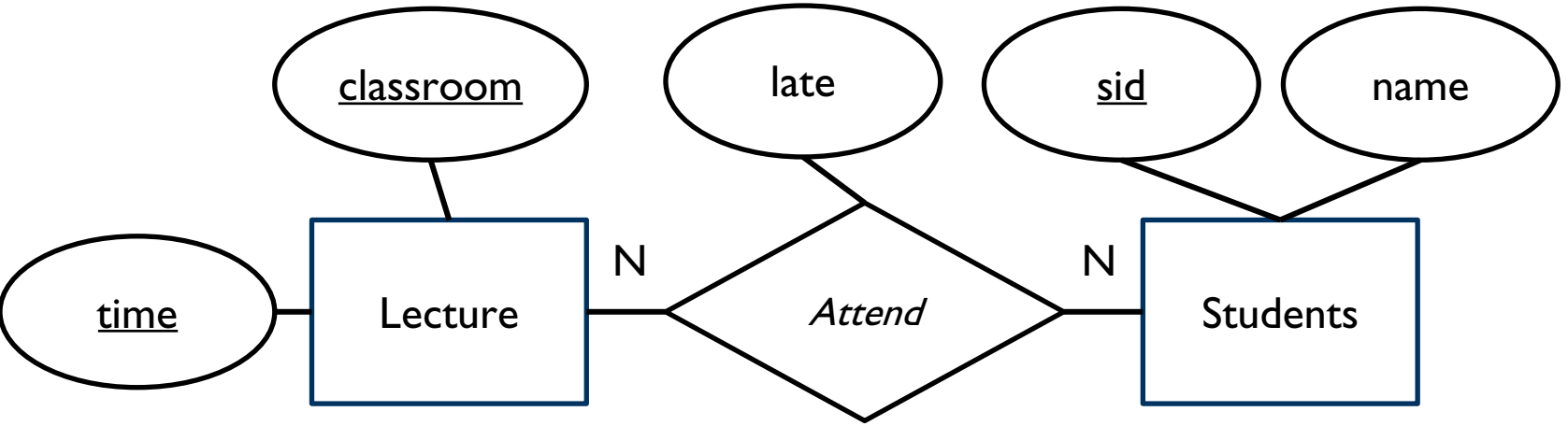


# Many-to-many Relationships Examples

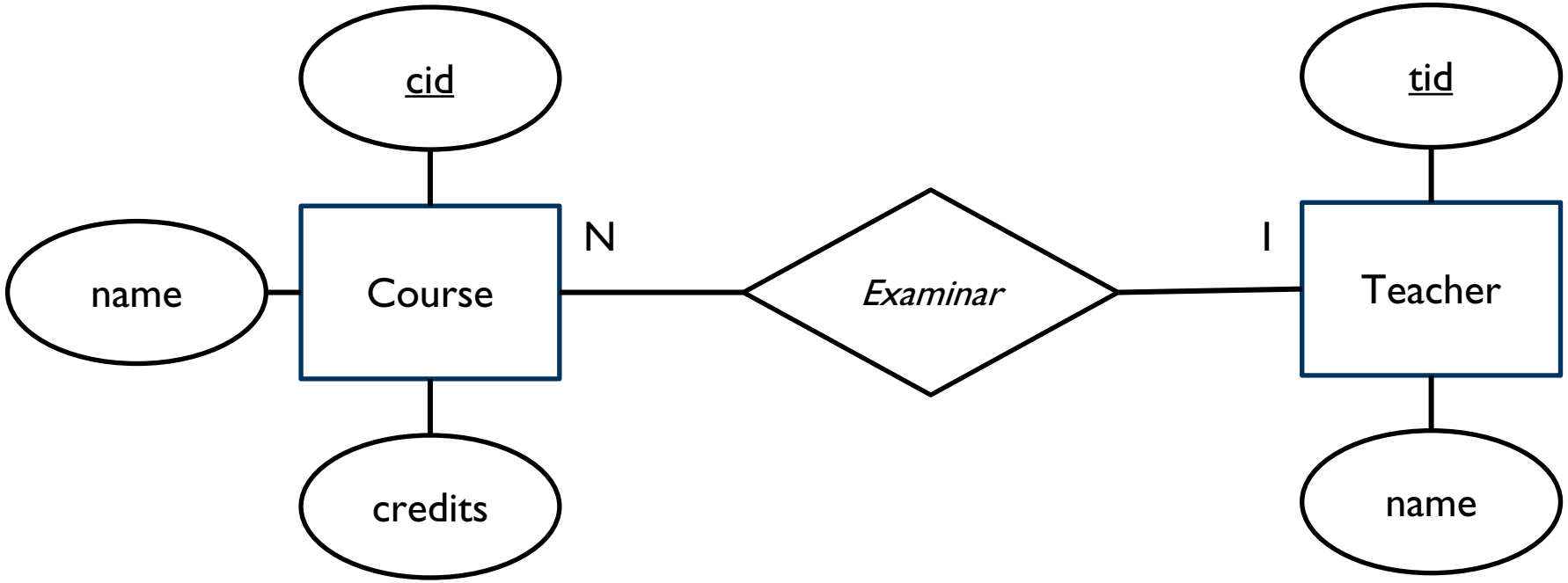



*Courses(<u>cid</u>, name, credits)*
*Teachers(<u>tid</u>, name)*
*TaughtBy(<u>course</u>, <u>teacher</u>, year)*
*course -> Courses.cid*
*teacher -> Teachers.tid*

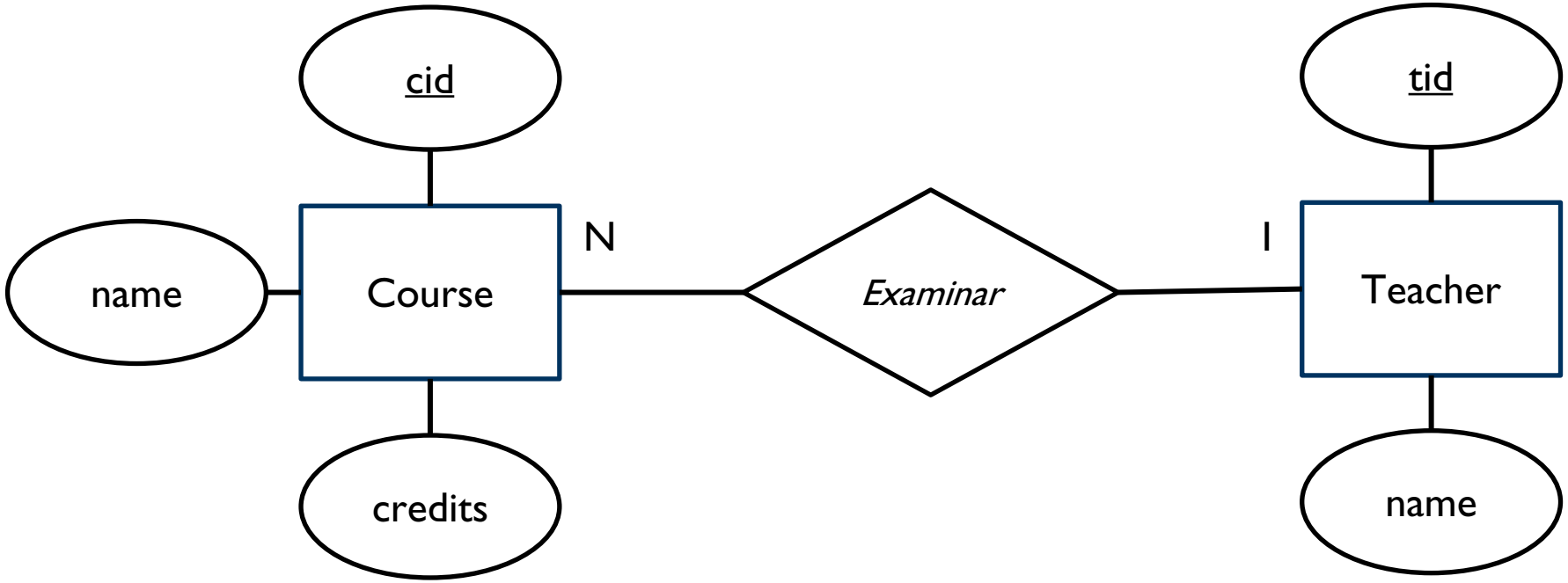



*Lecture(<u>classroom</u>, <u>time</u>)*
*Students(<u>sid</u>, name)*
*Attend(<u>room</u>, <u>time</u>, <u>std</u>, late)*
*room -> Lecture.classroom*
*time -> Lecture.time*
*std -> students.sid*

# Many-to-exactly-one Relationships

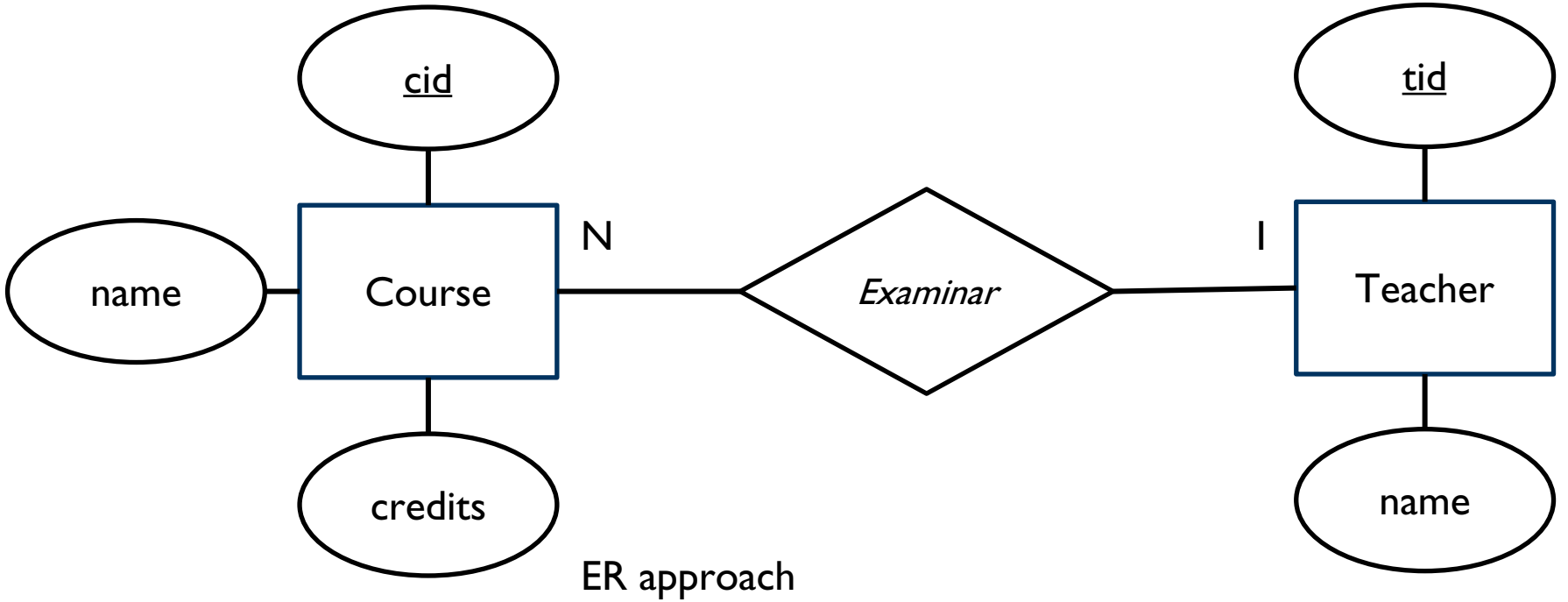


# Many-to-exactly-one Relationships




*Courses(<u>cid</u>, name, credits)*
*Teachers(<u>tid</u>, name)*
*Examinar(<u>course</u>, teacher)*
*course -> Courses.cid*
*teacher -> Teachers.tid*

# Many-to-exactly-one Relationships

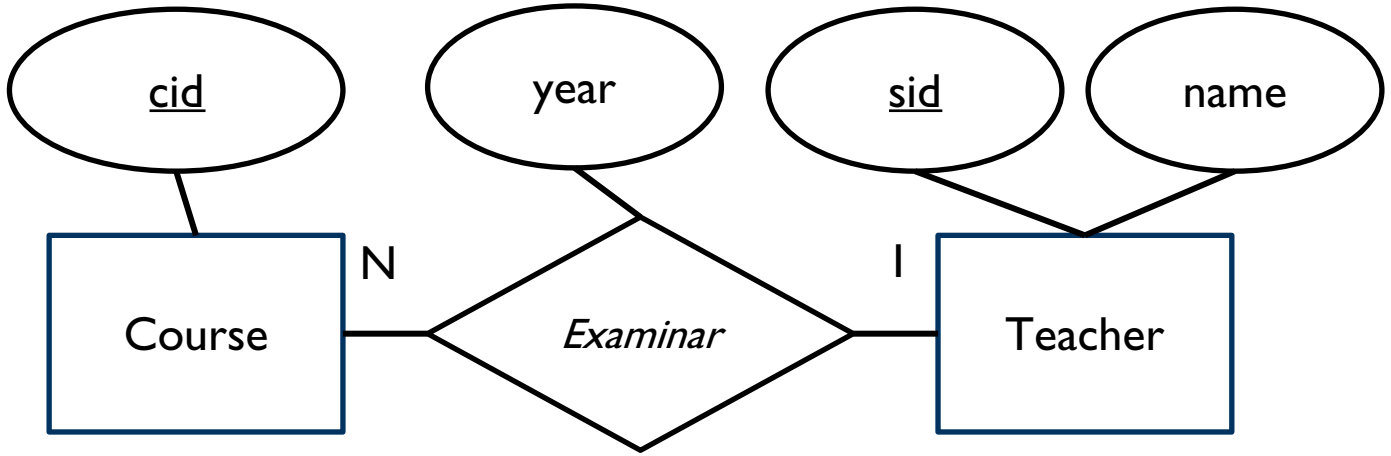



ER approach

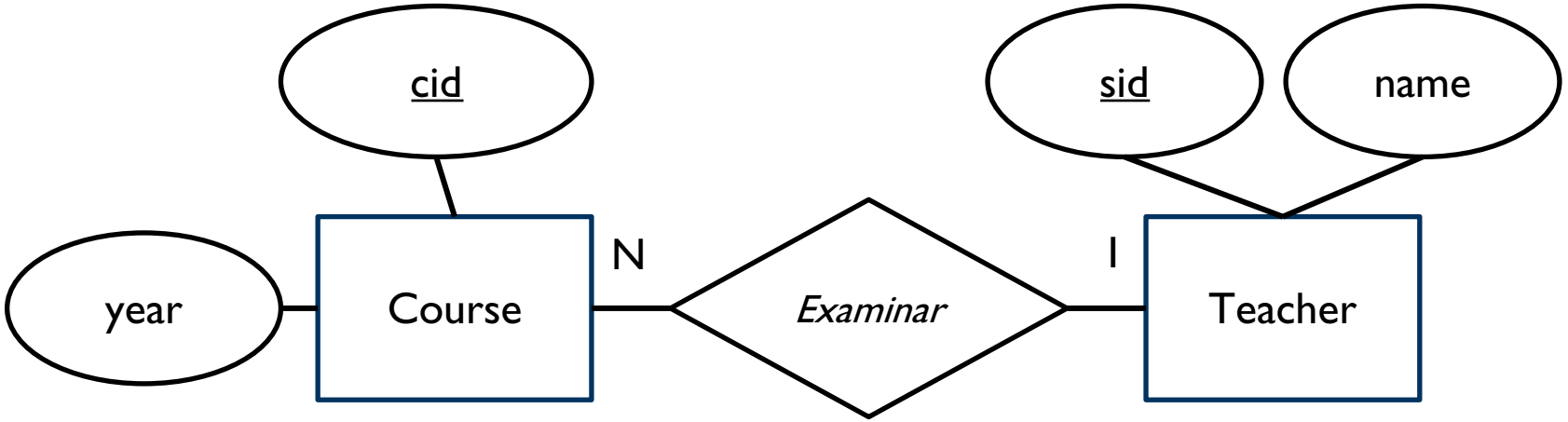
*Courses(<u>cid</u>, name, credits)*
*Teachers(<u>tid</u>, name)*
*Examinar(<u>course</u>, teacher)*
*course -> Courses.cid*
*teacher -> Teachers.tid*

*Courses(<u>cid</u>, name, credits, examinar)*
*examinar -> Teachers.tid*
*Teachers(<u>tid</u>, name)*

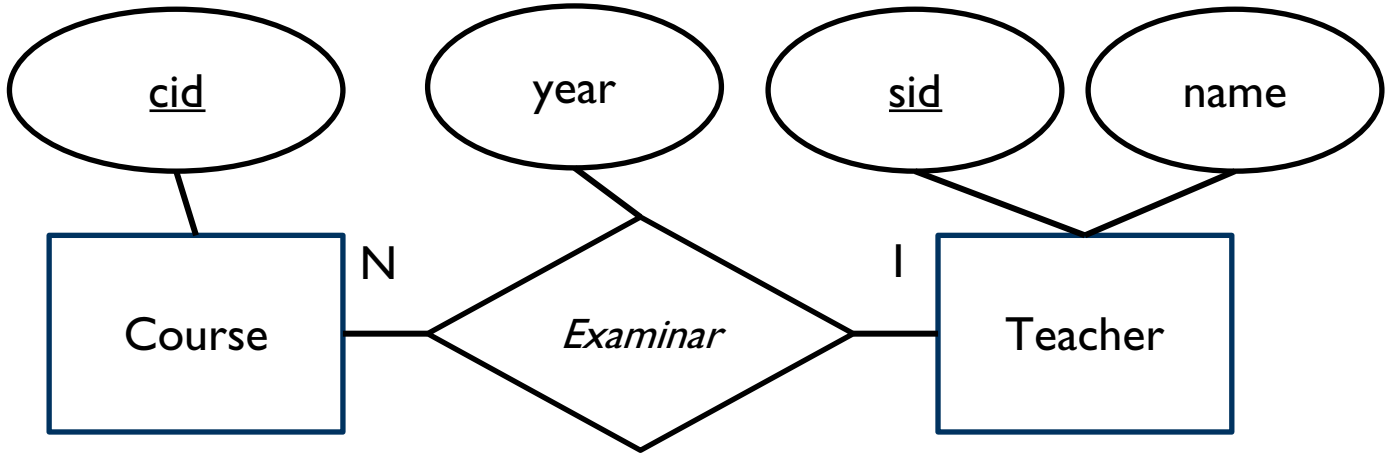
# Many-to-exactly-one Relationships




Vs.



# Many-to-exactly-one Relationships

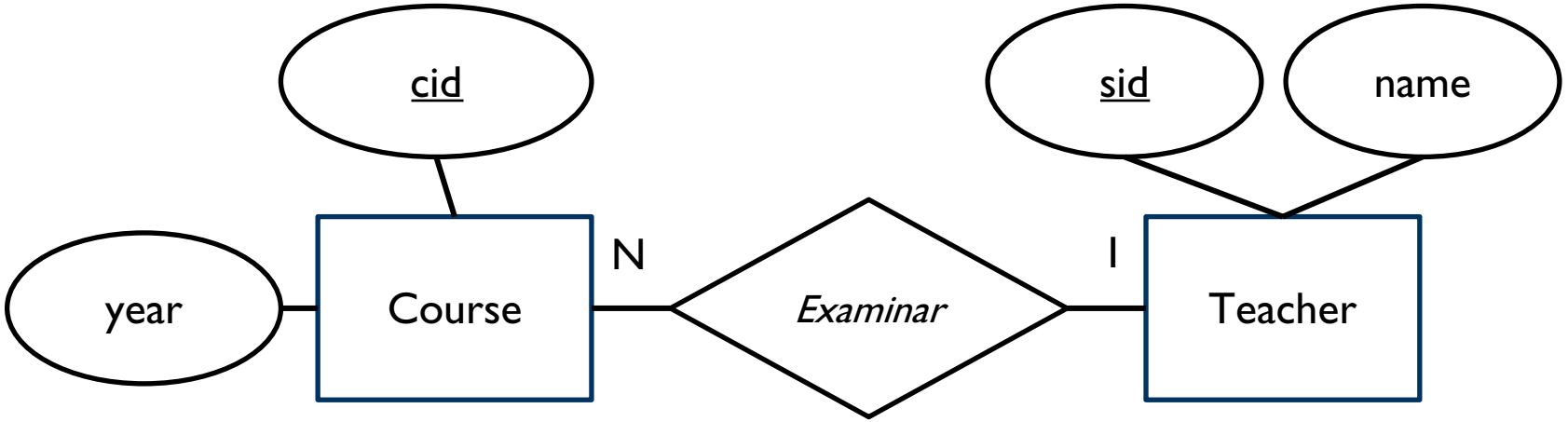



*Courses(<u>cid</u>, examinar, year)*
*examinar -> Teacher.sid*

*Teachers(<u>sid</u>, name)*

Vs.

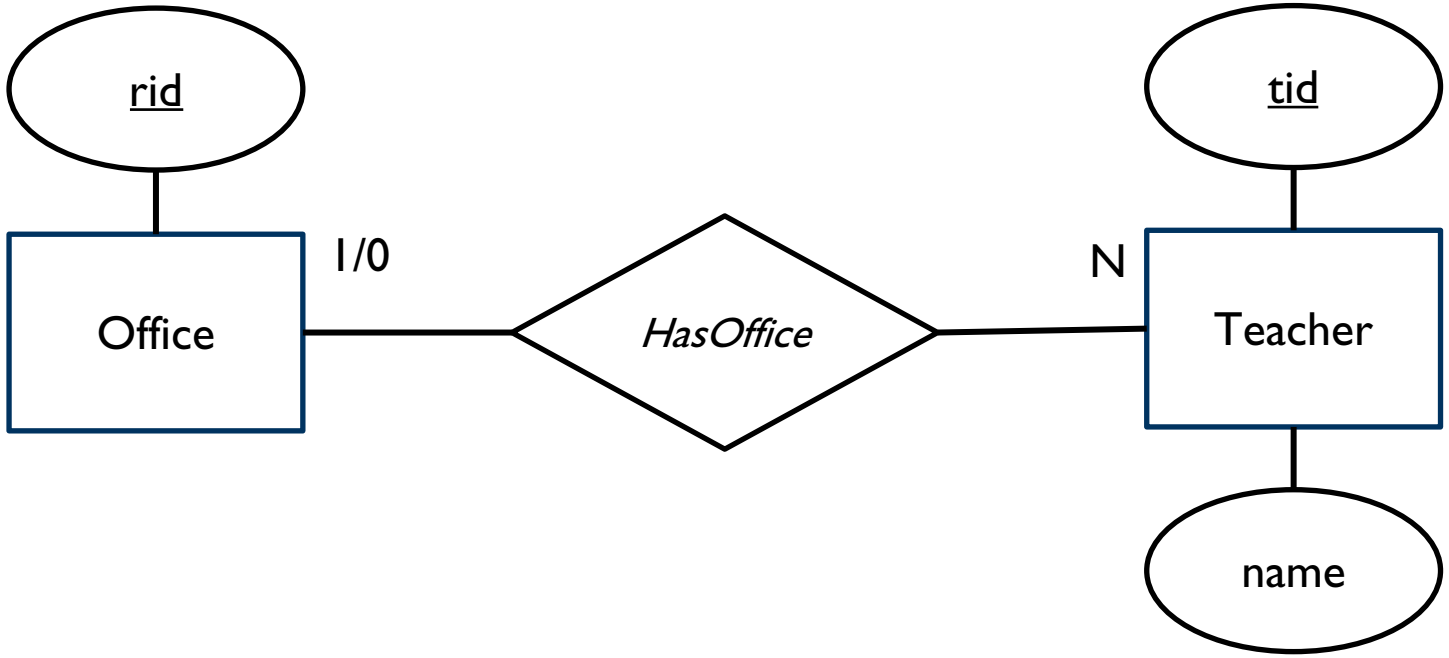
Same schema?



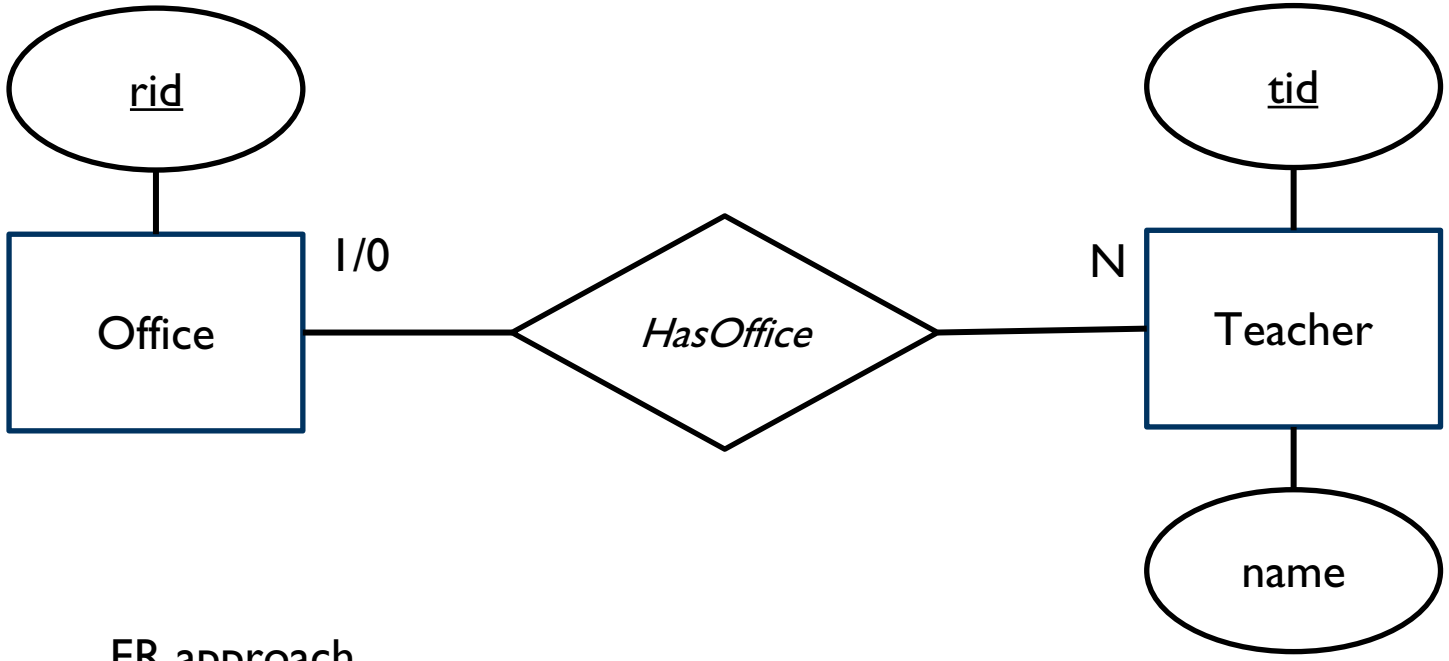

*Courses(<u>cid</u>, year, examinar)*
*examinar -> Teacher.sid*

*Teachers(<u>sid</u>, name)*

# Many-to-at-most-one Relationships



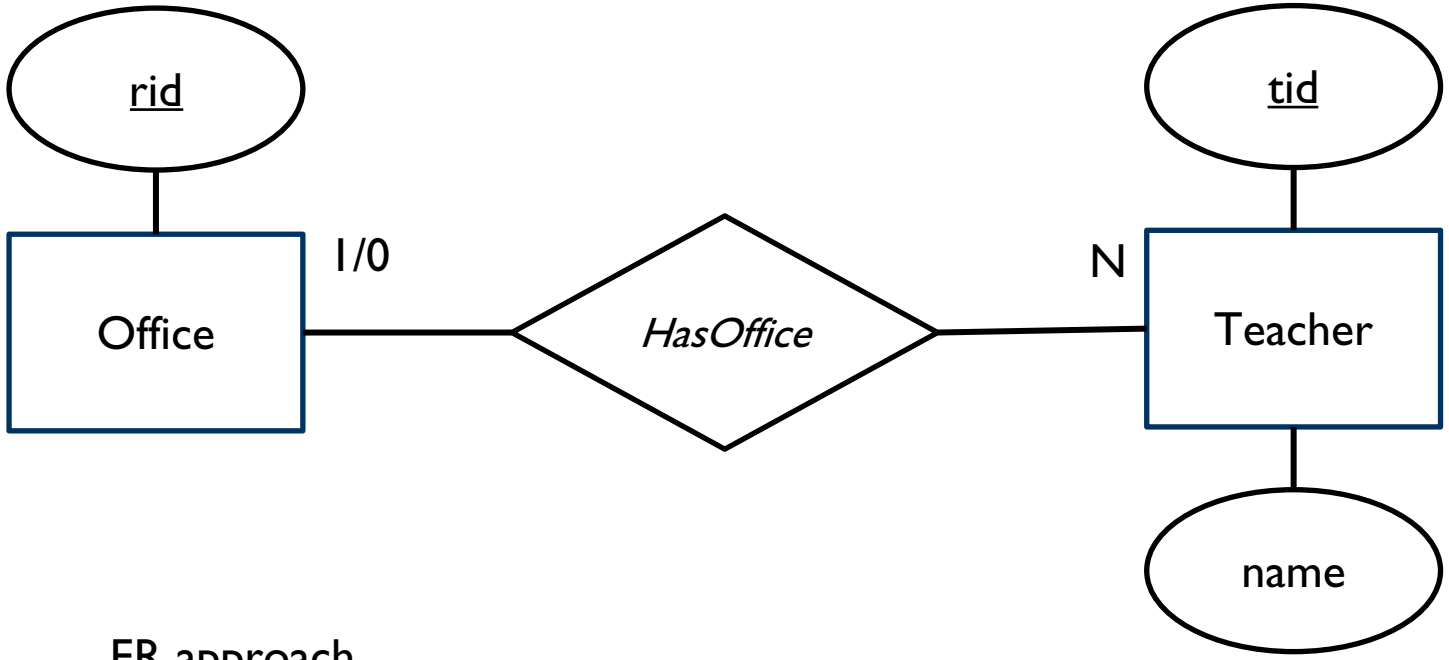
# Many-to-at-most-one Relationships




ER approach

*Offices(rid)*
*Teachers(tid, name)*
*HasOffice(teacher, office)*
*teacher -> Teachers.tid*
*office -> Offices.rid*

# Many-to-at-most-one Relationships



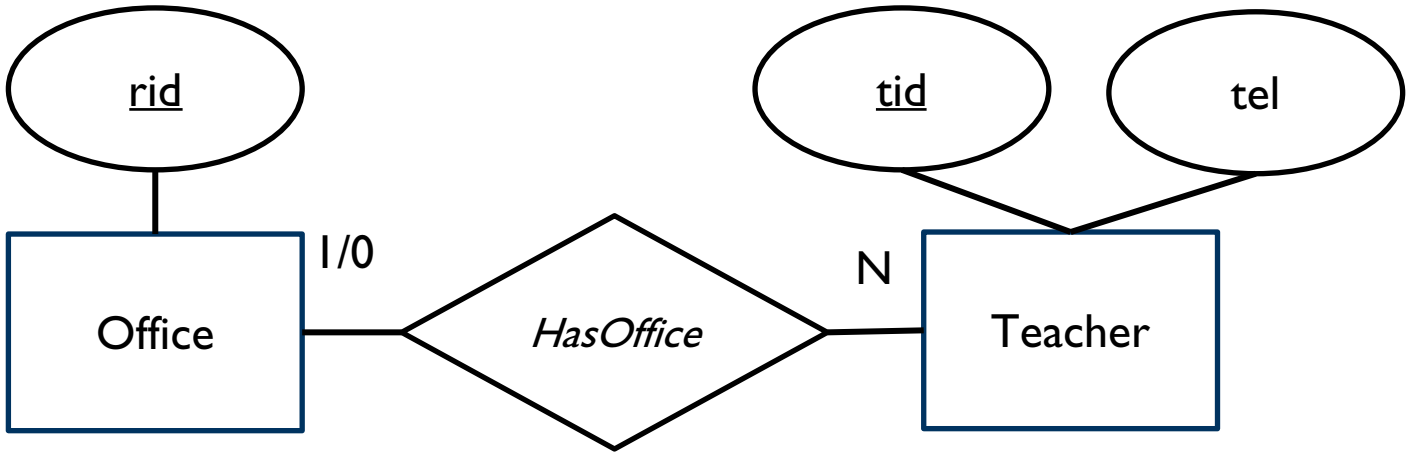

ER approach

*Offices(rid)*
*Teachers(tid, name)*
*HasOffice(teacher, office)*
*teacher -> Teachers.tid*
*office -> Offices.rid*

Null approach

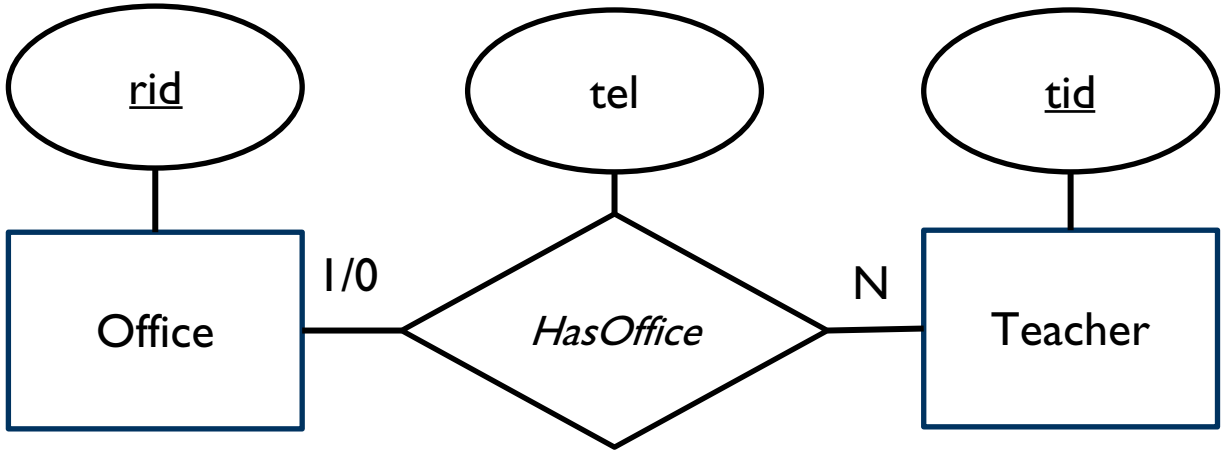
*Offices(rid)*
*Teachers(tid, name, office (or null))*
*office -> Offices.rid*

# Many-to-at-most-one Relationships



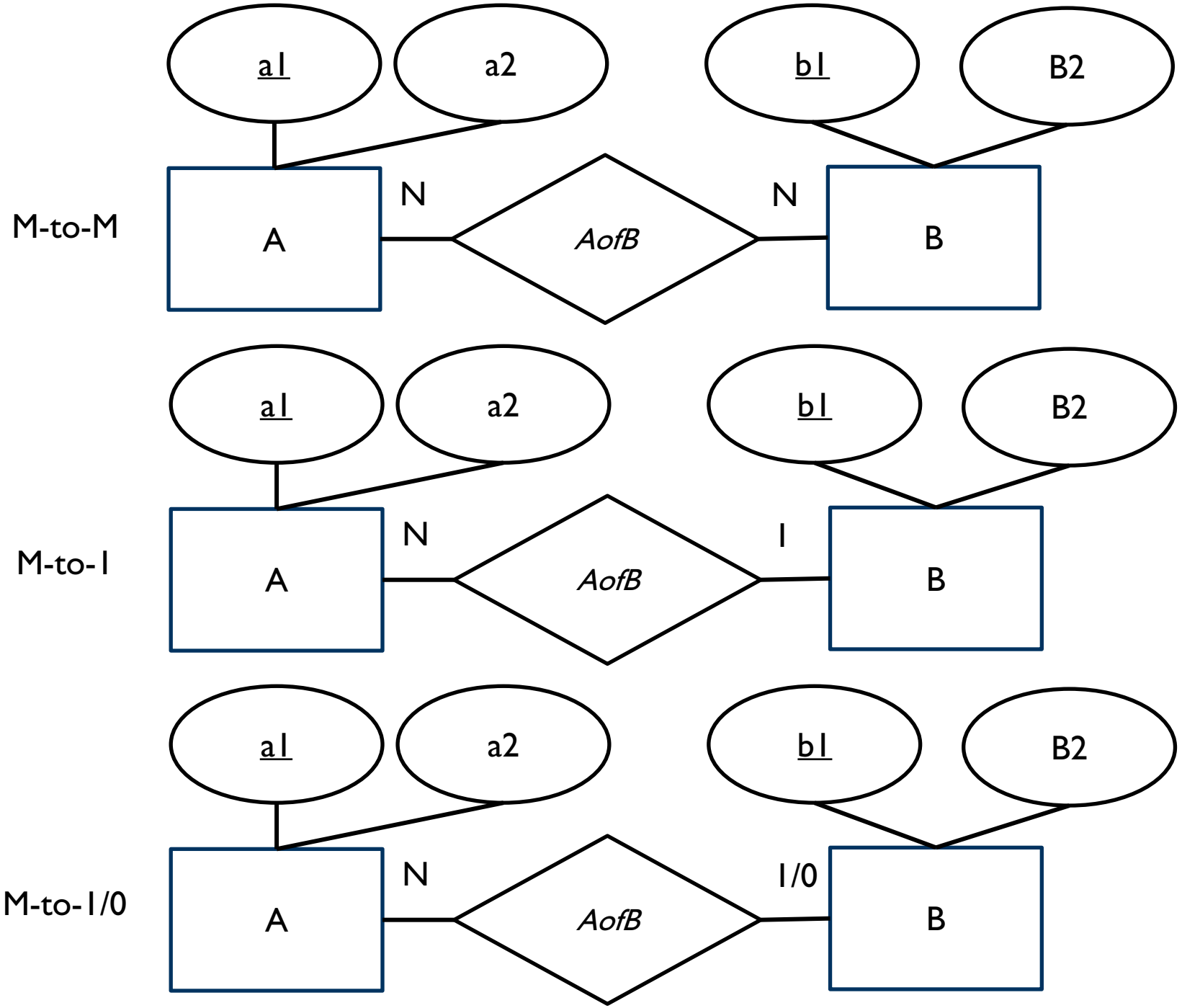

*Offices(<u>rid</u>)*
*Teachers(<u>tid</u>, tel)*
*HasOffice(<u>teacher</u>, office)*
*teacher -> Teachers.tid*
*office -> Offices.rid*

Vs.



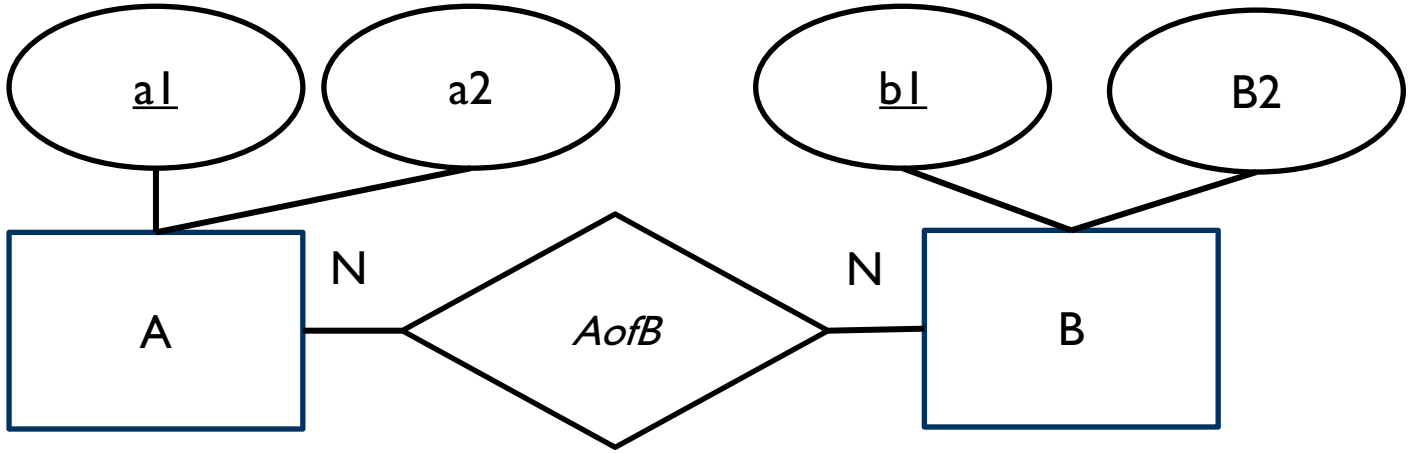

*Offices(<u>rid</u>)*
*Teachers(<u>tid</u>)*
*HasOffice(<u>teacher</u>, office, tel)*
*teacher -> Teachers.tid*
*office -> Offices.rid*

# Multiplicity Summary



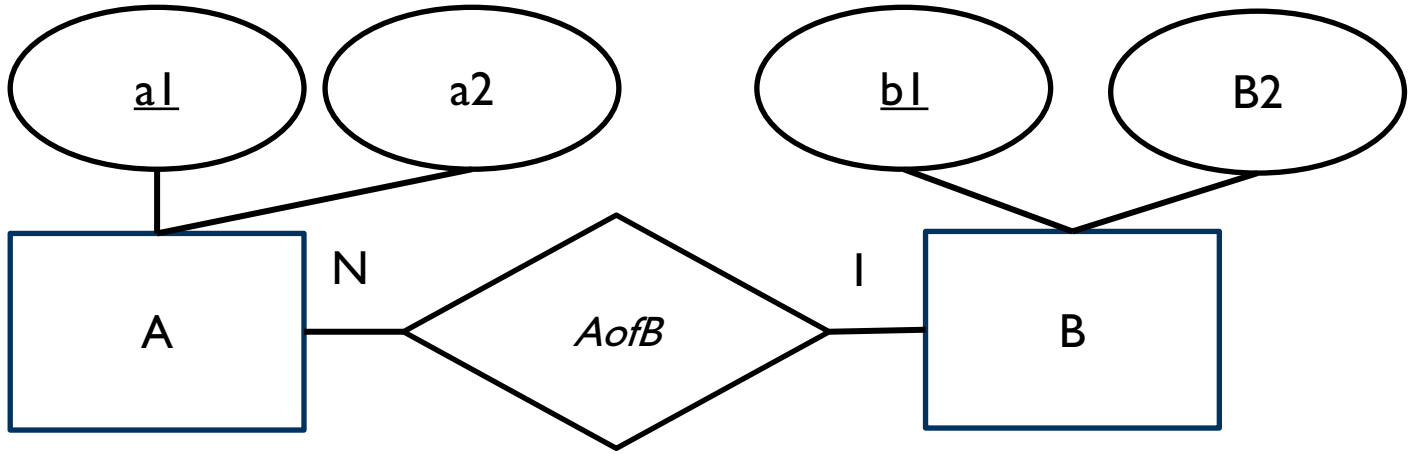
# Multiplicity Summary

M-to-M




*As(a1, a2)*
*Bs(b1, b2)*
*AsofBs(a, b)*
*a1 -> As.a1*
*b1 -> Bs.b1*

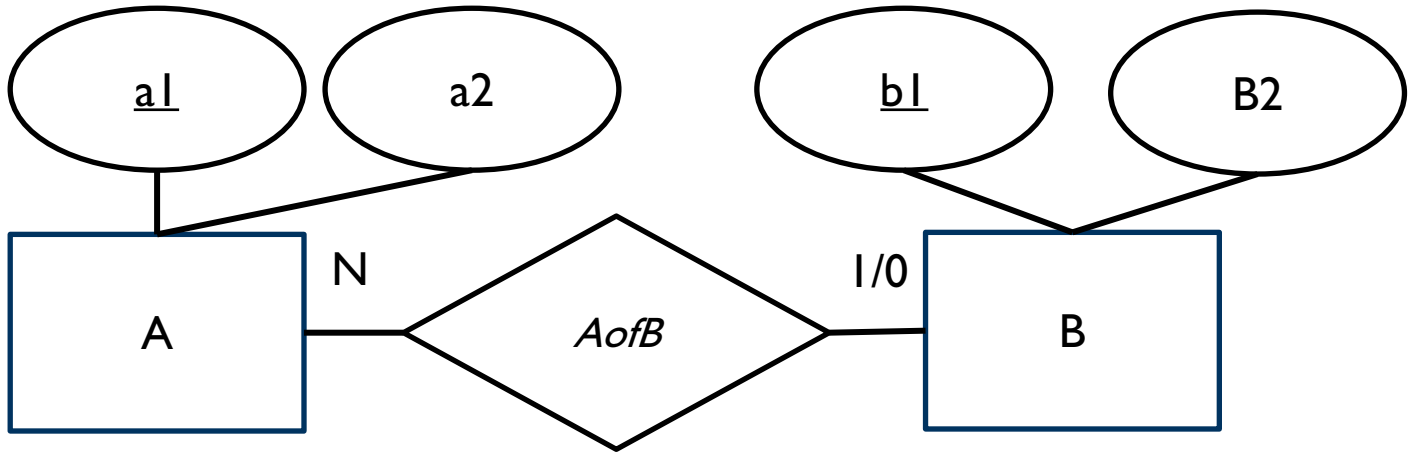
M-to-1




*As(a1, a2, b1)*
*b1 -> Bs.b1*

*Bs(b1, b2)*

M-to-1/0

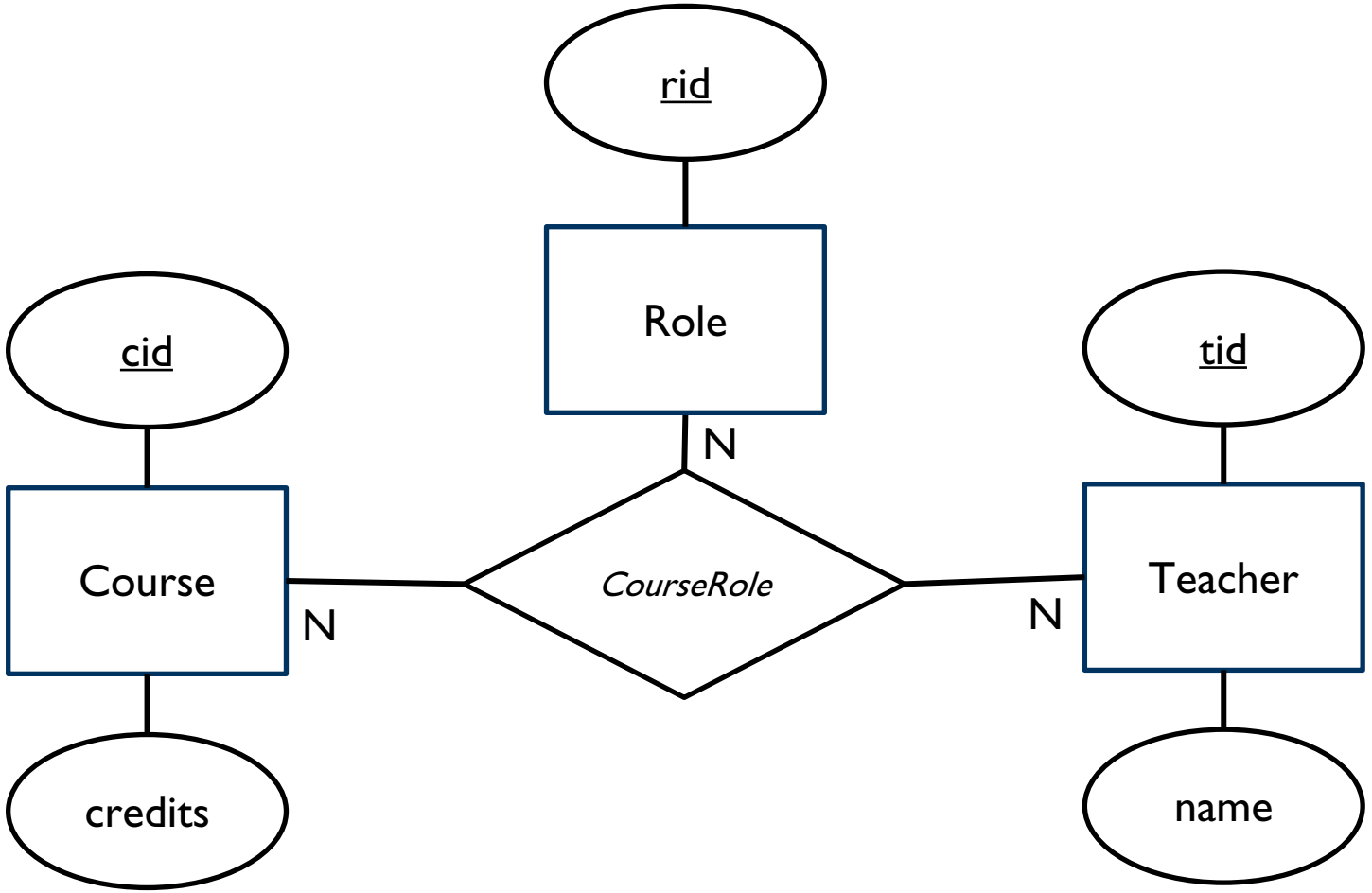



*As(a1, a2)*
*Bs(b1, b2)*
*AsofBs(a, b)*
*a1 -> As.a1*
*b1 -> Bs.b1*

or

*As(a1, a2, b1 (or null))*
*b1 -> Bs.b1*
*Bs(b1, b2)*

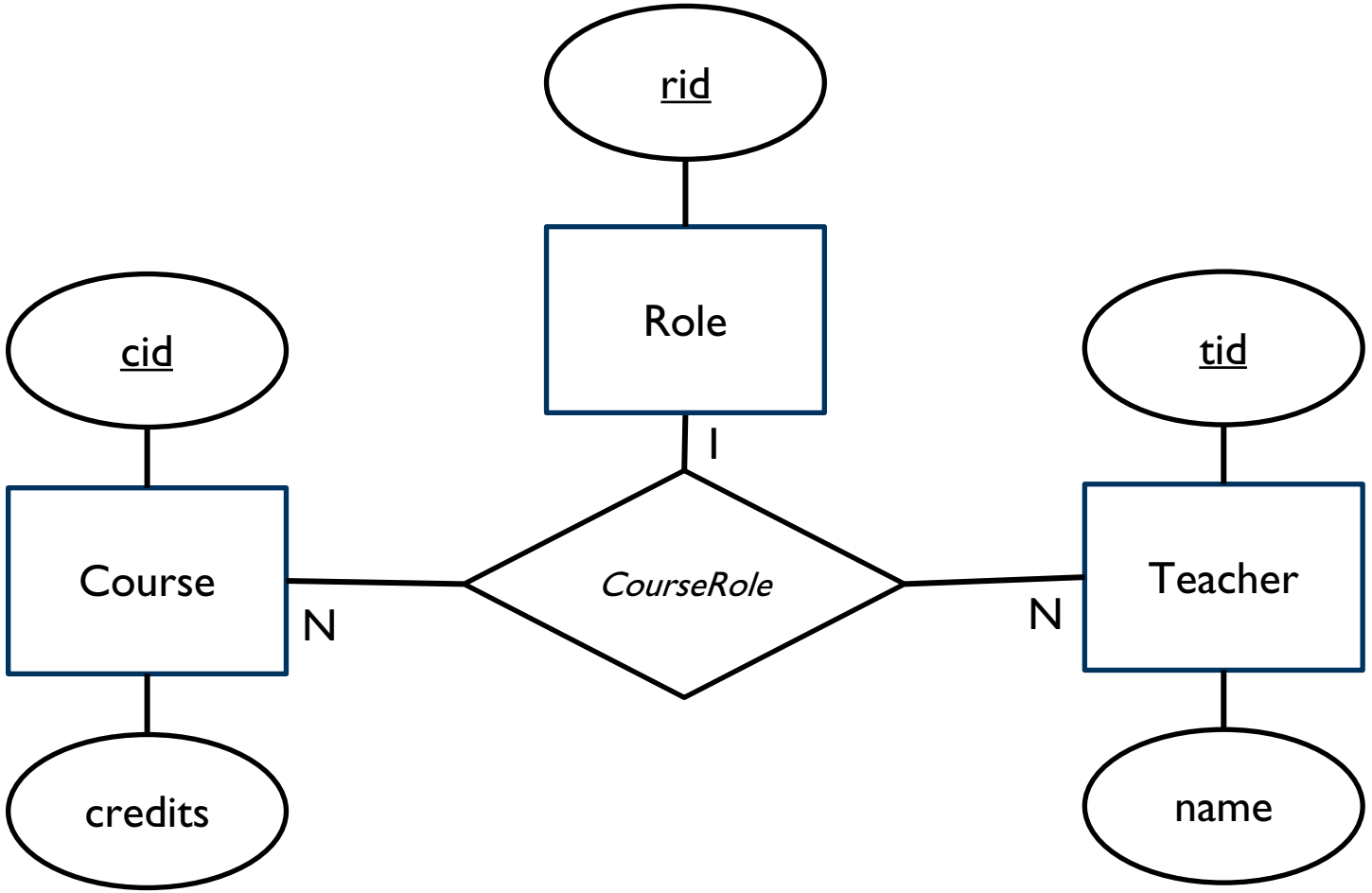
# Multiway relationship




*Courses(<u>cid</u>, credits)*
*Teachers(<u>tid</u>, name)*
*Role(<u>rid</u>)*
*CourseRole(<u>course</u>, <u>teacher</u>, <u>role</u>)*
*course -> Courses.cid*
*teacher -> Teachers.tid*
*role→ Roles.rid*

Key pairs (*<u>course, teacher</u>*) ensures assignment of any number teachers with any number of courses, for each association we need to select a valid role

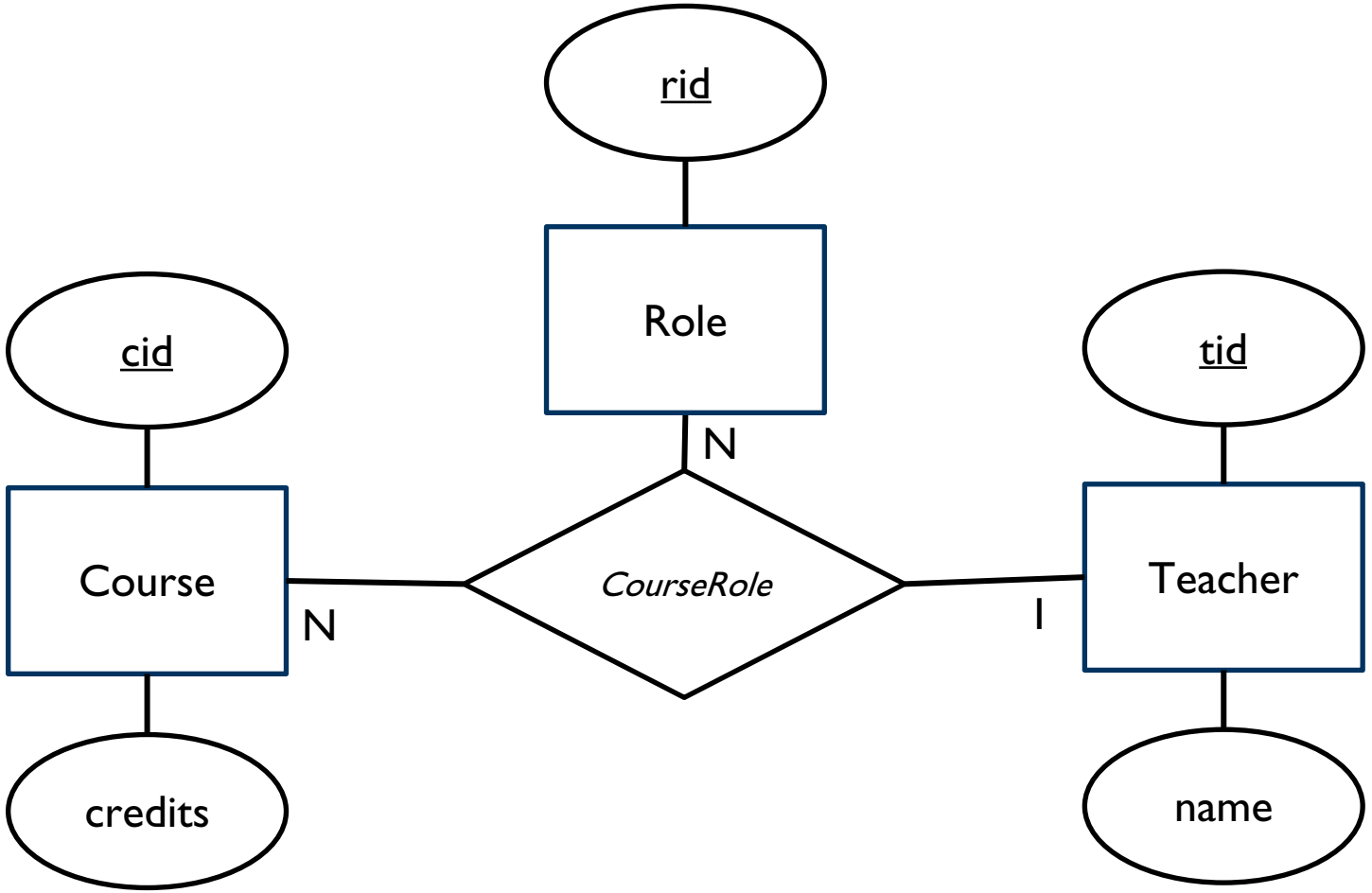
# Multiway relationship




*Courses(<u>cid</u>, credits)*
*Teachers(<u>tid</u>, name)*
*Role(<u>rid</u>)*
*CourseRole(<u>course</u>, <u>teacher</u>, role)*
*course -> Courses.cid*
*teacher -> Teachers.tid*
*role→ Roles.rid*

Key pairs (*<u>course, teacher</u>*) ensures assignment of any number teachers with any number of courses, for each association we need to select a valid role

# Multiway relationship

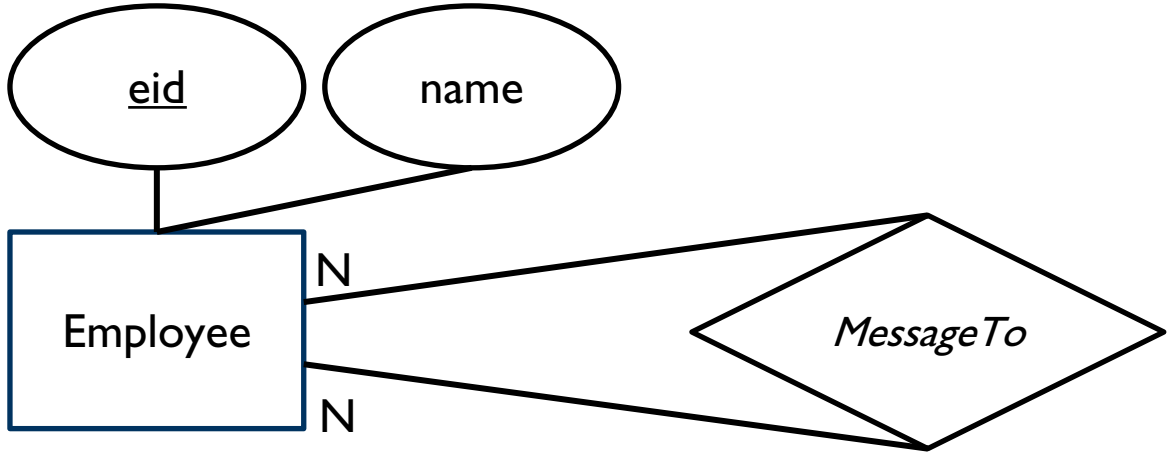



*Courses(<u>cid</u>, credits)*
*Teachers(<u>tid</u>, name)*
*Role(<u>rid</u>)*
*CourseRole(<u>course</u>, teacher, <u>role</u>)*
*course -> Courses.cid*
*teacher -> Teachers.tid*
*role→ Roles.rid*

Key pairs (*<u>course, teacher</u>*) ensures assignment of any number teachers with any number of courses, for each association we need to select a valid role

# Self-relationships

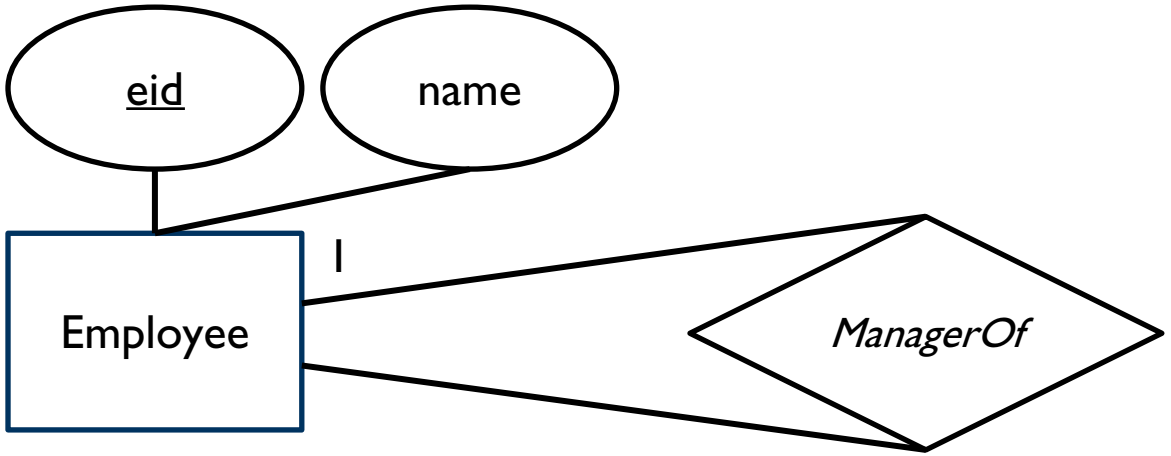
M-to-M




*Employee(<u>eid</u>, name)*

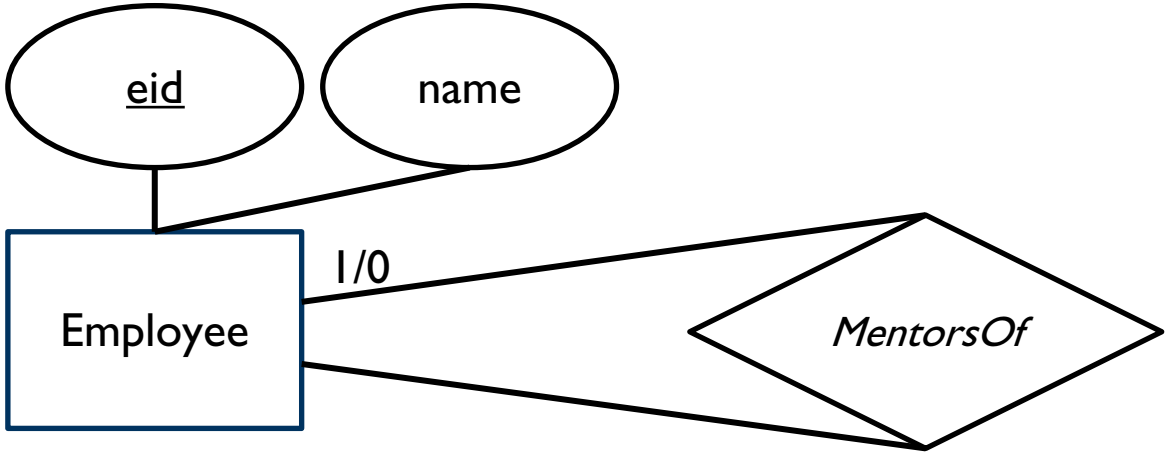
*MessageTo(<u>sender</u>, <u>receiver</u>)*
*sender -> Employee.eid*
*receiver -> Employee.eid*

M-to-1




*Employee(<u>eid</u>, name, Manager)*
*Manager -> Employee.eid*

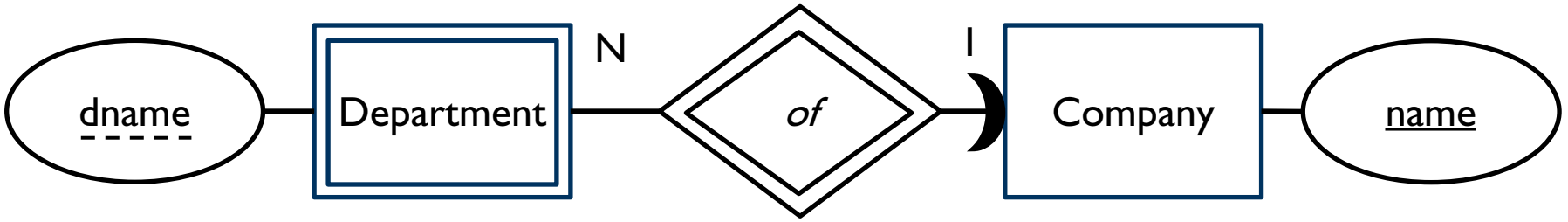
M-to-1/0




*Employee(<u>eid</u>, name)*

*MentorsOf(<u>mentee</u>, mentor)*
*mentee -> Employee.eid*
*Mentor -> Employee.eid*

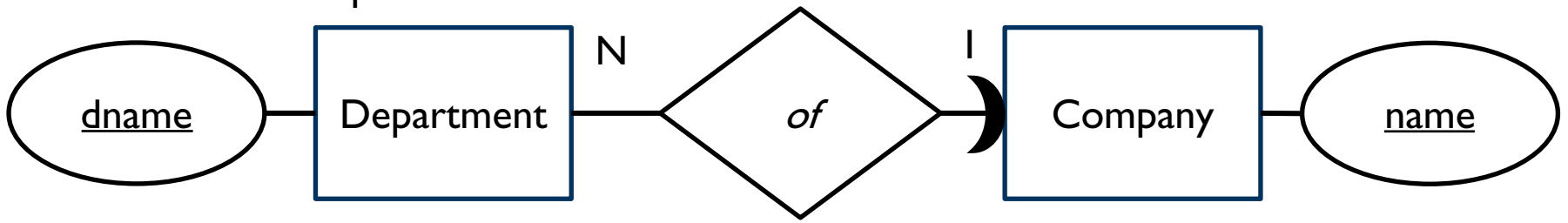
# Weak Entities




*Companies(<u>name</u>)*
*department(<u>dname</u>, <u>company</u>)*
*company->companies.name*

Vs.

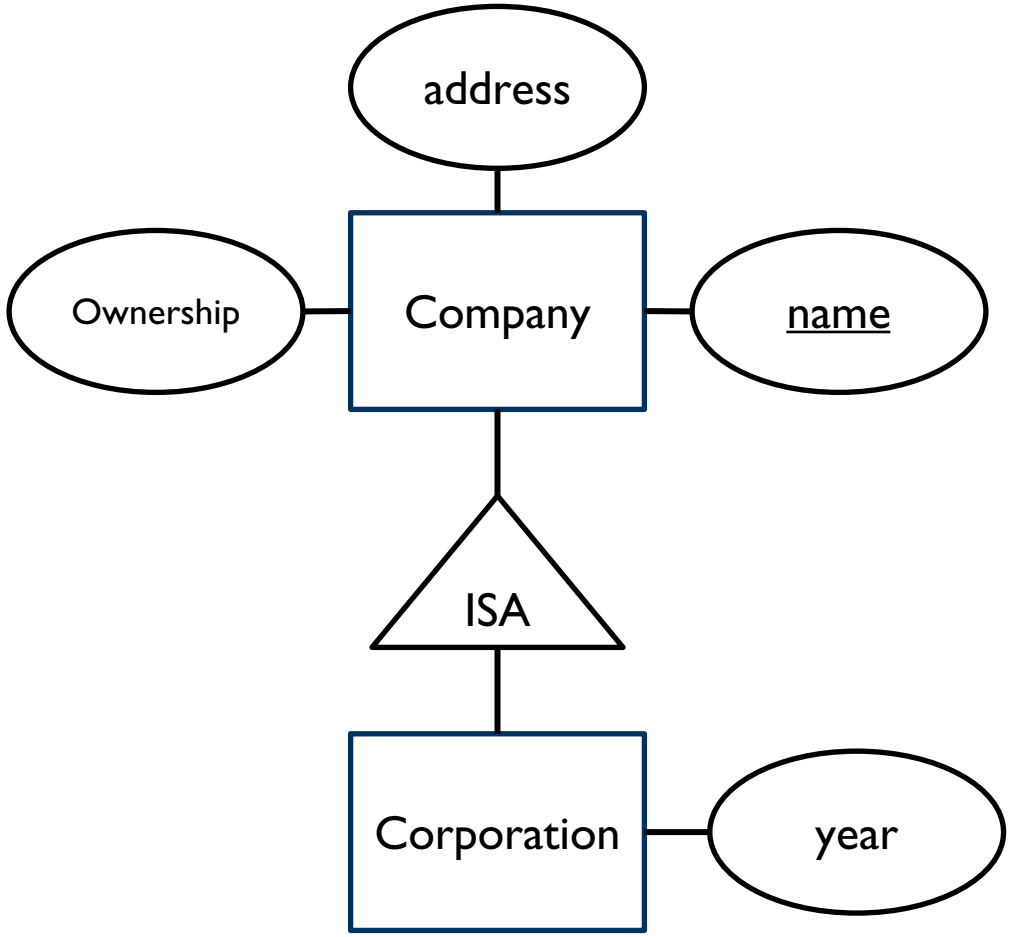
M-to-1 Relationships




*Companies(<u>name</u>)*
*department(<u>dname</u>, company)*
*company->companies.name*

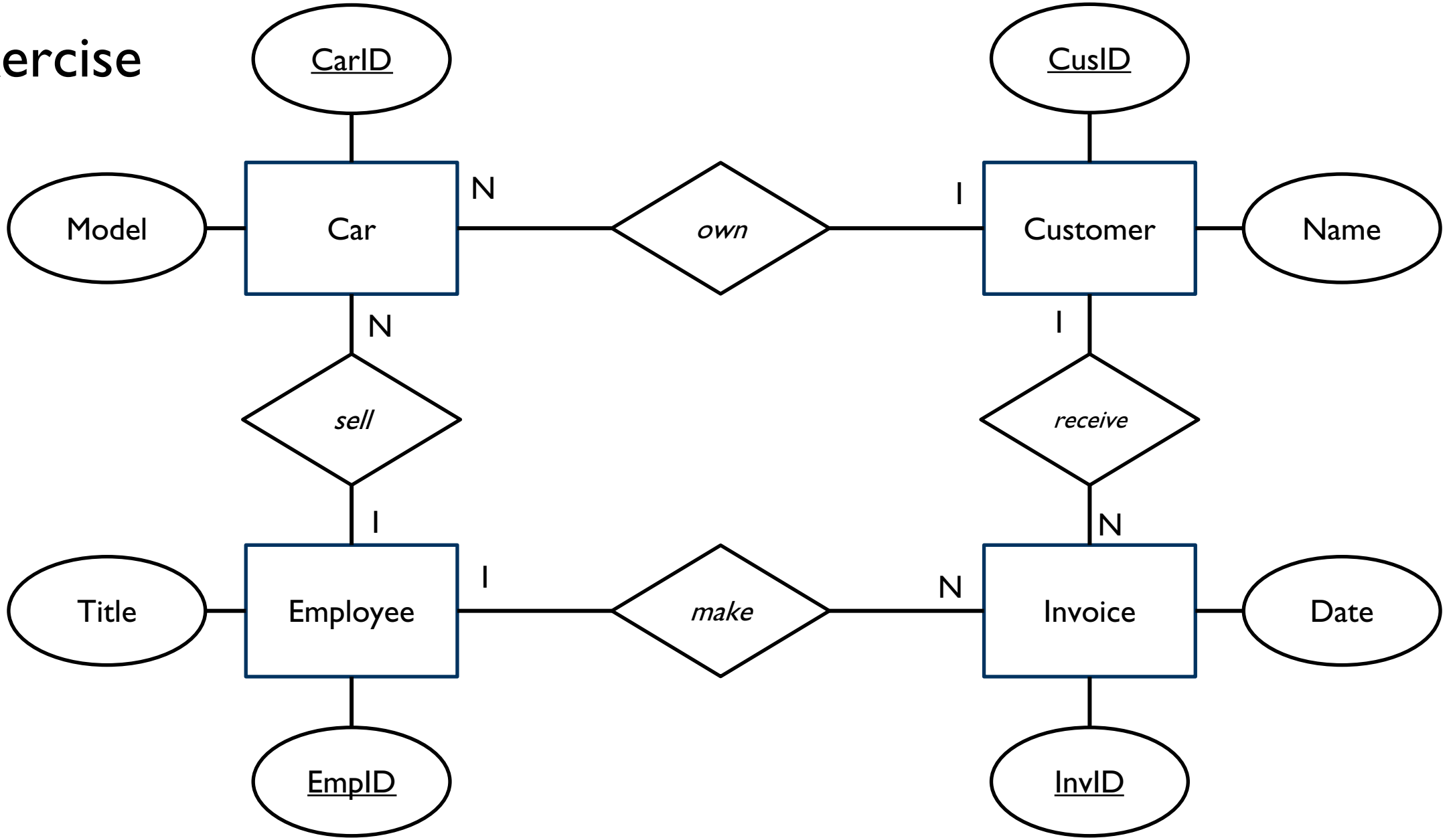
# Inheritance in EER

- ISA relationships, ISA stands for “is a”
- Example
  - Corporations are a special kind of companies, they have a year in addition to all properties of other companies
- Corporation is a subentity
- Campany is its superentity
- Note that corporation do not has key attributes
- Subentities can never have key attributes of their own




*Companies(<u>name</u>, address)*
*Corporations(<u>name</u>, year)*
*name>companies.name*

# Exercise

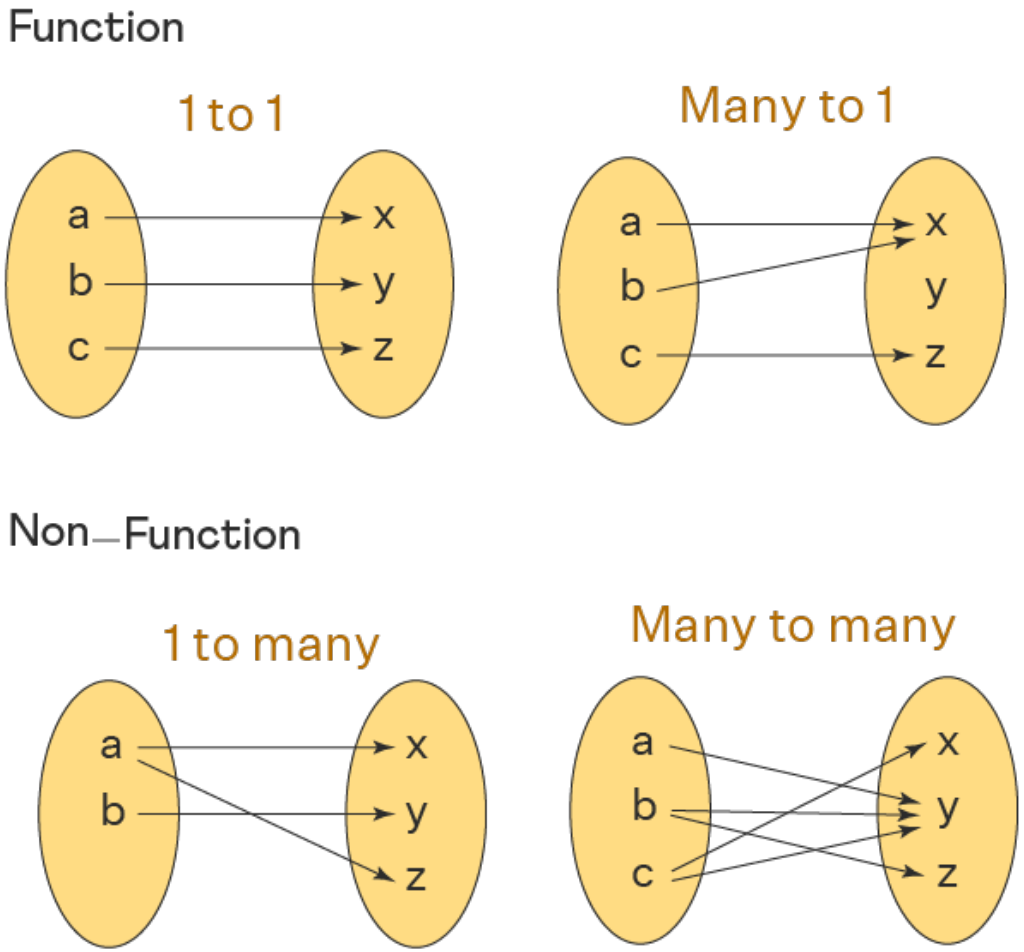

CarID
CusID
Model
Car
N
own
1
Customer
Name
N
1
sell
receive
1
N
Title
Employee
1
make
N
Invoice
Date
EmpID
InvID

# Functional Dependency

- Utilized to detect data redundancies
  - Removing data redundancies
  - Avoiding anomlies (insertion, deletion, update)
- Values in some columns uniquely decide values in others
- X→ Y
  - X decide values in Y

# Functional Dependency

- Utilized to detect data redundancies
  - Removing data redundancies
  - Avoiding anomlies (insertion, deletion, update)
- Values in some columns uniquely decide values in others
- X→ Y
  - X decide values in Y



# Functional Dependency

- Why the following databse design is bad?

  Schedule

| tid | cid | c_name | date | time | room | nn_seats |
|---|---|---|---|---|---|---|
| 33 | 4 | Databases | 2030-01-23 | 10:15 - 12:00 | D415 | 50 |
| 33 | 4 | Databases | 2030-01-24 | 08:15 - 10:00 | D415 | 50 |
| 11 | 3 | Mathematics | 2030-01-24 | 13:15 - 15:00 | D208 | 30 |
| 11 | 3 | Mathematics | 2030-01-25 | 13:15 - 15:00 | D415 | 50 |

- Redundancy
  - Duplicates in the number of seats for room D415
- Update anomaly & delete anomaly
  - Change nn_seats in one row but not others
  - nn_seats information is gone if all bookings of D415 is removed

# Functional Dependency

- Why the following databse design is bad?

  Schedule

| tid | cid | c_name | date | time | room | nn_seats |
|---|---|---|---|---|---|---|
| 33 | 4 | Databases | 2030-01-23 | 10:15 - 12:00 | D415 | 50 |
| 33 | 4 | Databases | 2030-01-24 | 08:15 - 10:00 | D415 | 50 |
| 11 | 3 | Mathematics | 2030-01-24 | 13:15 - 15:00 | D208 | 30 |
| 11 | 3 | Mathematics | 2030-01-25 | 13:15 - 15:00 | D415 | 50 |

- Identifying Functional Dependencies

# Functional Dependency

- Why the following databse design is bad?

  Schedule

| tid | cid | c_name | date | time | room | nn_seats |
|---|---|---|---|---|---|---|
| 33 | 4 | Databases | 2030-01-23 | 10:15 - 12:00 | D415 | 50 |
| 33 | 4 | Databases | 2030-01-24 | 08:15 - 10:00 | D415 | 50 |
| 11 | 3 | Mathematics | 2030-01-24 | 13:15 - 15:00 | D208 | 30 |
| 11 | 3 | Mathematics | 2030-01-25 | 13:15 - 15:00 | D415 | 50 |

- Identifying Functional Dependencies
  - room -> nn_seats
  - cid -> course_name

# Decomposing the table

Schedule

| tid | cid | date | time | room |
|---|---|---|---|---|
| 33 | 4 | 2030-01-23 | 10:15 - 12:00 | D415 |
| 33 | 4 | 2030-01-24 | 08:15 - 10:00 | D415 |
| 11 | 3 | 2030-01-24 | 13:15 - 15:00 | D208 |
| 11 | 3 | 2030-01-25 | 13:15 - 15:00 | D415 |

Nice and simple when tables are small

Room

| room | nn_seats |
|---|---|
| D415 | 50 |
| D208 | 30 |

Course

| cid | course_name |
|---|---|
| 3 | Discrete Mathematics |
| 4 | Databases |

# Solution via FD?

- Decomposing tables to removed redundancies
  - No functional dependenies between attributes in the same table
    - i.e. FD (X –> Y) does not connect columns in the same table
    - Avoid anomalies previously mentioned
- Note that data loss must be avoided via decomposition
  - Reconstruction based on FD
  - Recompose by looking up X for Y values
- Decomposing table may require additional joins for querying the data

# Finding Functional Dependencies (FDs)

- Common mistake
  - Try finding FDs by looking at data
- Data only captures current state of the database
  - Not all functional dependencies may appear
  - Data may suggest misleading “pseudo FDs”
- Two valid sources for mining FDs:
  - Domain knowledge
  - Inferring new FDs from given FDs