# Database Systems

## The Entity Relationship Model

Dr. Rudra Pratap Deb Nath

Department of Computer Science
and Engineering

University of Chittagong
rudra@cu.ac.bd

4th Semester 2022

# Learning goals

## Goals

- Create non-trivial ER diagrams
- Assess the quality of an ER diagram
- Perform and explain the mapping of ER diagrams to relations
- Use a particular ER notation properly

## Motivation

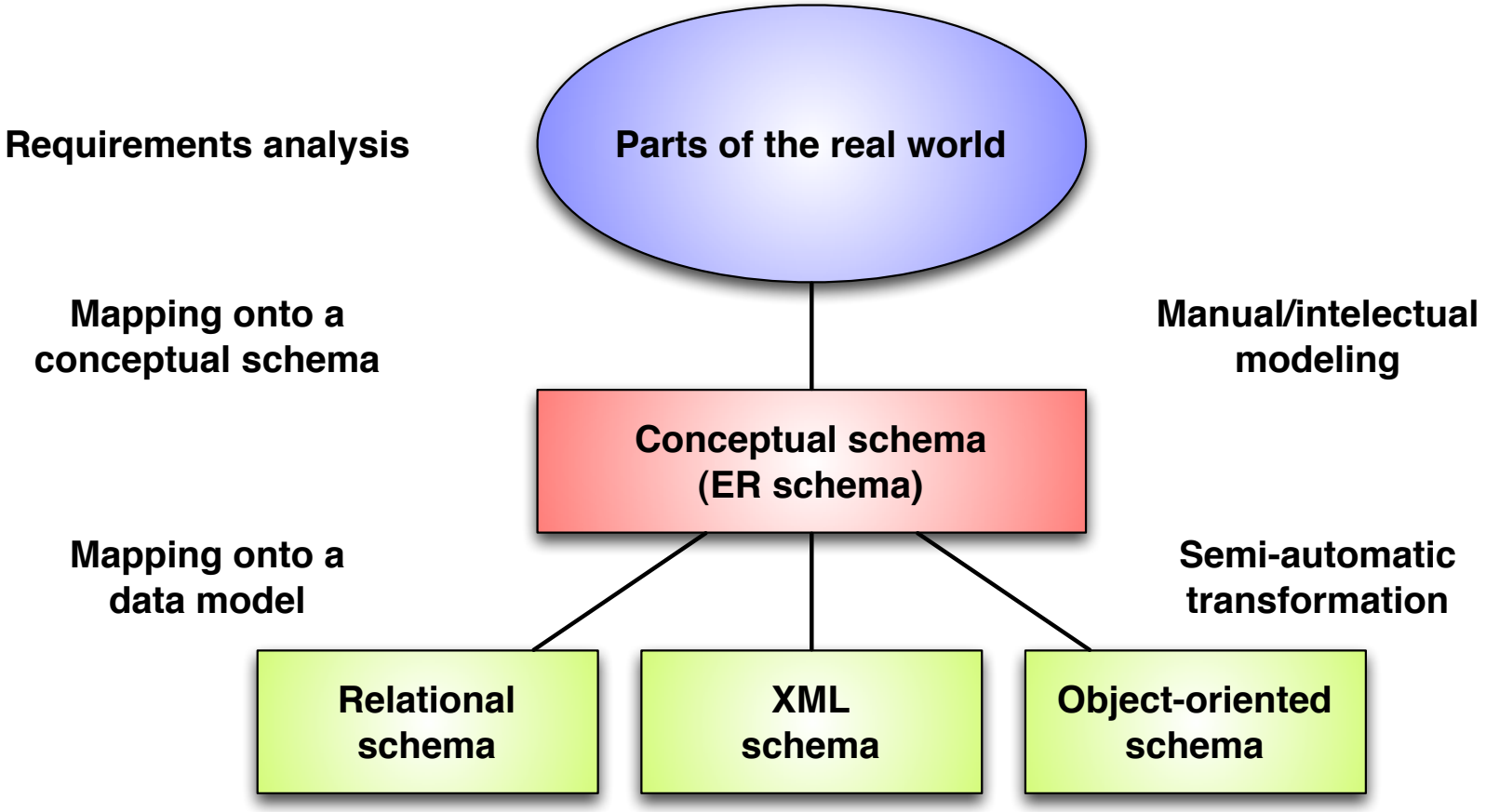
- ER diagrams are used widely
- ER model is easy to learn
  *Much simpler than UML*
- An ER diagram is a good communication tool
  *Talking the same language*

# Outline I

# Outline II

# Outline III

# Steps of database design



# Step 1: Requirements analysis


http://www.cu.ac.bd/
https://cu.ac.bd/dept/facultyprofile.php?secno=2&menumapno=130

Processes to model

- "Students take courses"
- "Instructors offer courses"
- "The student ID unambiguously identifies a student"
- ...

# Step 1: Requirements analysis – object specification

**Employees**

Attributes: EmpNo, salary, rank

**EmpNo**
- Type: char
- Length: 9
- Domain: 0...999.999.99
- Degree of availability: 100%
- Uniqueness: true

**Salary**
- Type: decimal
- Length: (8,2)
- Degree of availability: 10%
- Uniqueness: no

**Rank**
- Type: String
- Length: 4
- Degree of availability: 100%
- Uniqueness: no

# Step 1: Requirements analysis – relationship specification

Relationship: "grades"

## Participating objects

- Instructor as examiner
- Student as examinee
- Course as topic

## Attributes of relationship "grades"

- Date
- Time
- Grade

# Step 2: Mapping onto a conceptual model

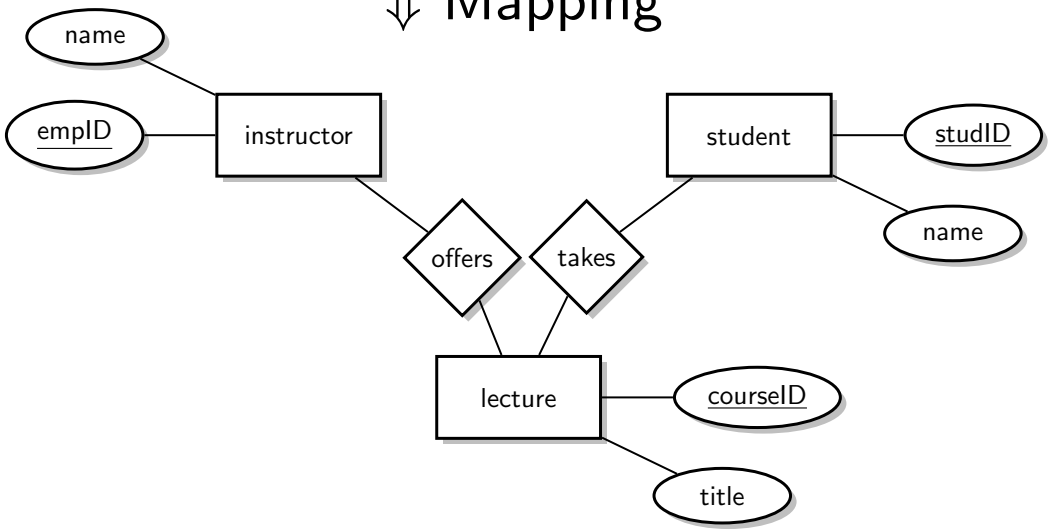
Requirements

- "Students take courses"
- "Instructors offer courses"
- "The student ID unambiguously identifies a student"
- ...

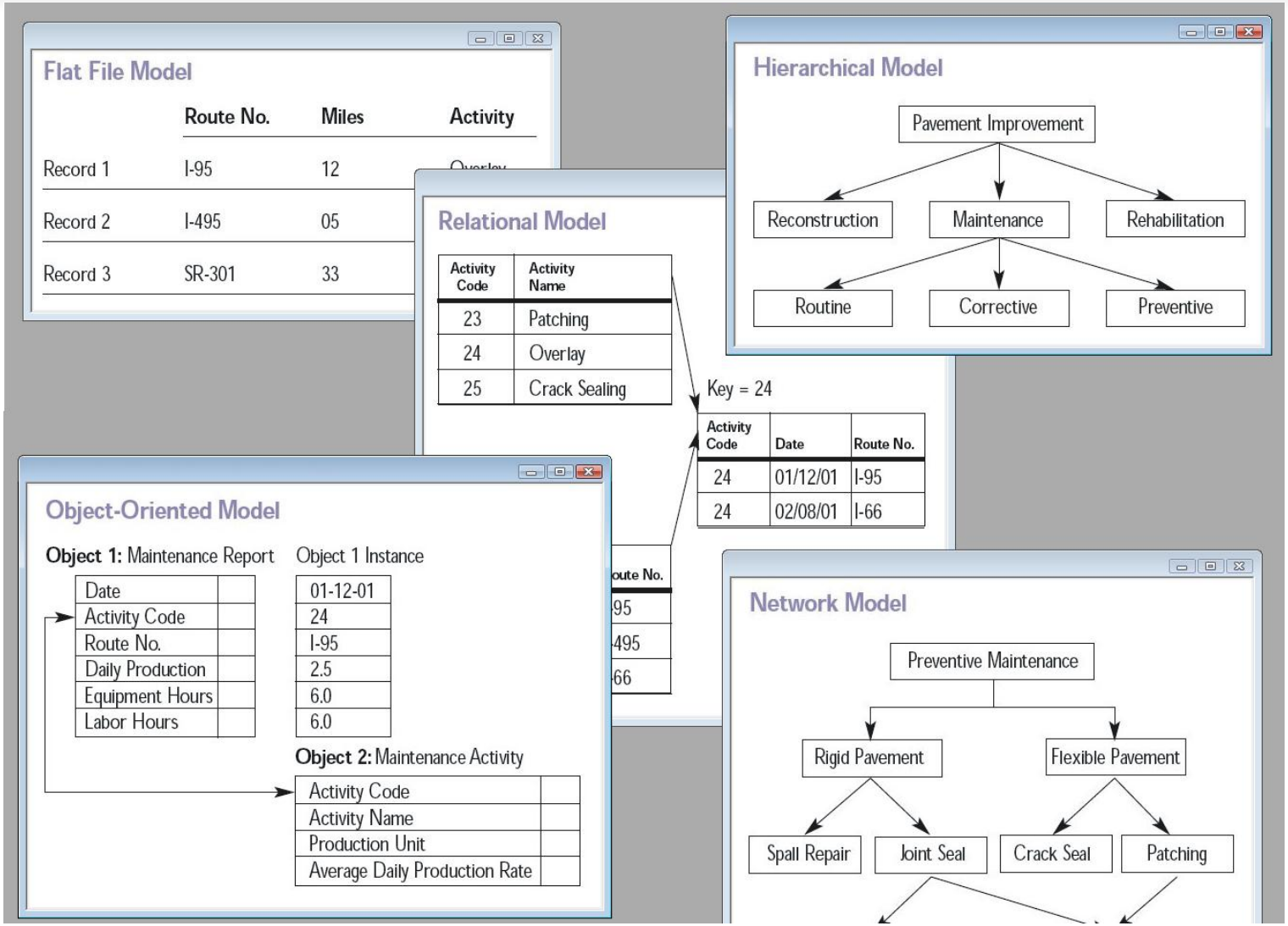
Functional requirements

- Secretary needs to feed in the grades
- ...

⇓ Mapping

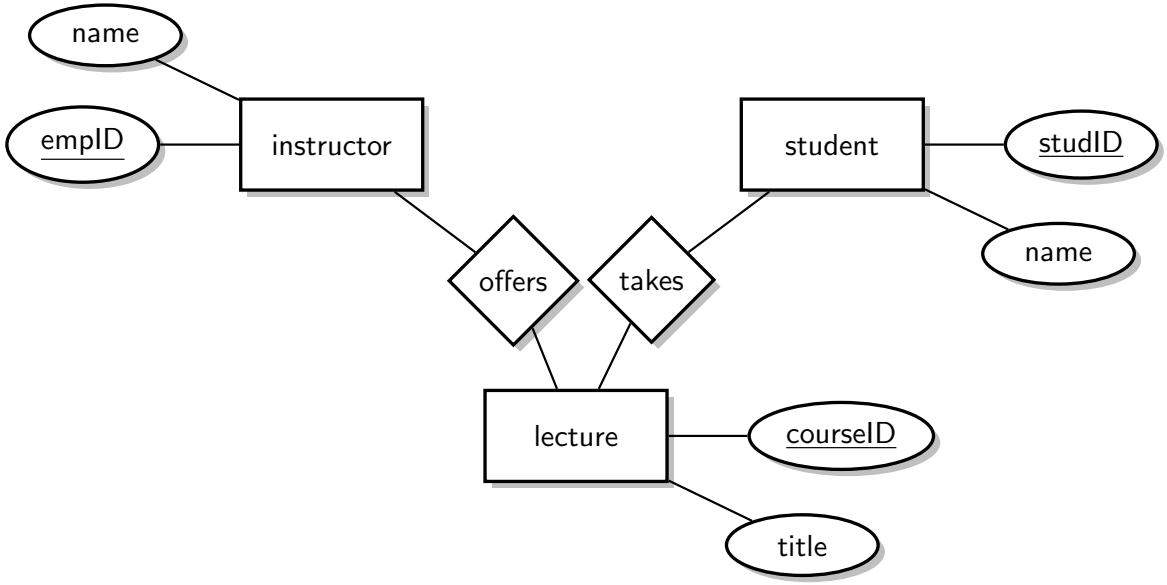


# Step 3: Mapping onto a data model




http://en.wikipedia.org/wiki/Database_model

# Step 3: Mapping onto the relational model




⇓ Mapping

Relational model

- student (studID: integer, name: string)
- takes (studID: integer, courseID: integer)
- lecture (courseID: integer, title: string)

# Step 4: Realization and implementation

Relational model

- student (studID: integer, name: string)
- takes (studID: integer, courseID: integer)
- lecture (courseID: integer, title: string)

⇓ Mapping

Tables in a DB

| student | |
|---|---|
| studID | name |
| 26120 | Pedersen |
| 25403 | Hansen |
| ... | ... |

| takes | |
|---|---|
| studID | courseID |
| 25403 | 5022 |
| 26120 | 5001 |
| ... | ... |

| lecture | |
|---|---|
| courseID | title |
| 5001 | DBS |
| 5022 | Belief and Knowledge |
| ... | ... |

# Step 4: Realization and implementation

Tables in a DB

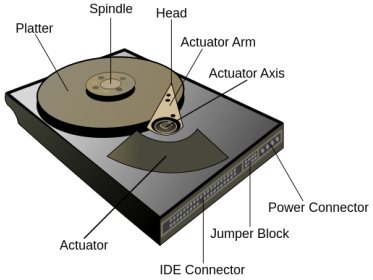
| student | |
|---|---|
| studID | name |
| 26120 | Pedersen |
| 25403 | Hansen |
| ... | ... |

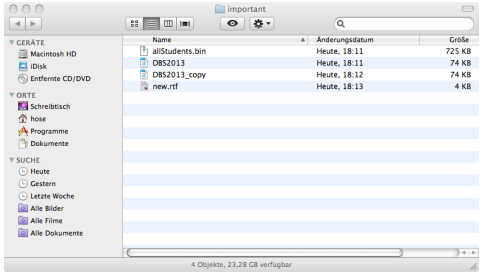
| takes | |
|---|---|
| studID | courseID |
| 25403 | 5022 |
| 26120 | 5001 |
| ... | ... |

| lecture | |
|---|---|
| courseID | title |
| 5001 | DBS |
| 5022 | Belief and Knowledge |
| ... | ... |

⇓ Mapping

Memory, pages, data structures, indexes, files, devices



http://en.wikipedia.org

# Steps of database design

1. Requirements analysis
   *What are we dealing with?*
2. Mapping onto a conceptual model (conceptual design)
   *What data and relationships have to be captured?*
3. Mapping onto a data model (logical design)
   *How to structure data in a specific model (here: the relational model)?*
4. Realization and implementation (physical design)
   *Which adaptations and optimizations does a specific DBMS require?*

# Steps of database design

1. Requirements analysis
   *What are we dealing with?*
2. Mapping onto a conceptual model (conceptual design)
   *What data and relationships have to be captured?*
3. Mapping onto a data model (logical design)
   *How to structure data in a specific model (here: the relational model)?*
4. Realization and implementation (physical design)
   *Which adaptations and optimizations does a specific DBMS require?*

A good design avoids redundancy and incompleteness.

# Outline

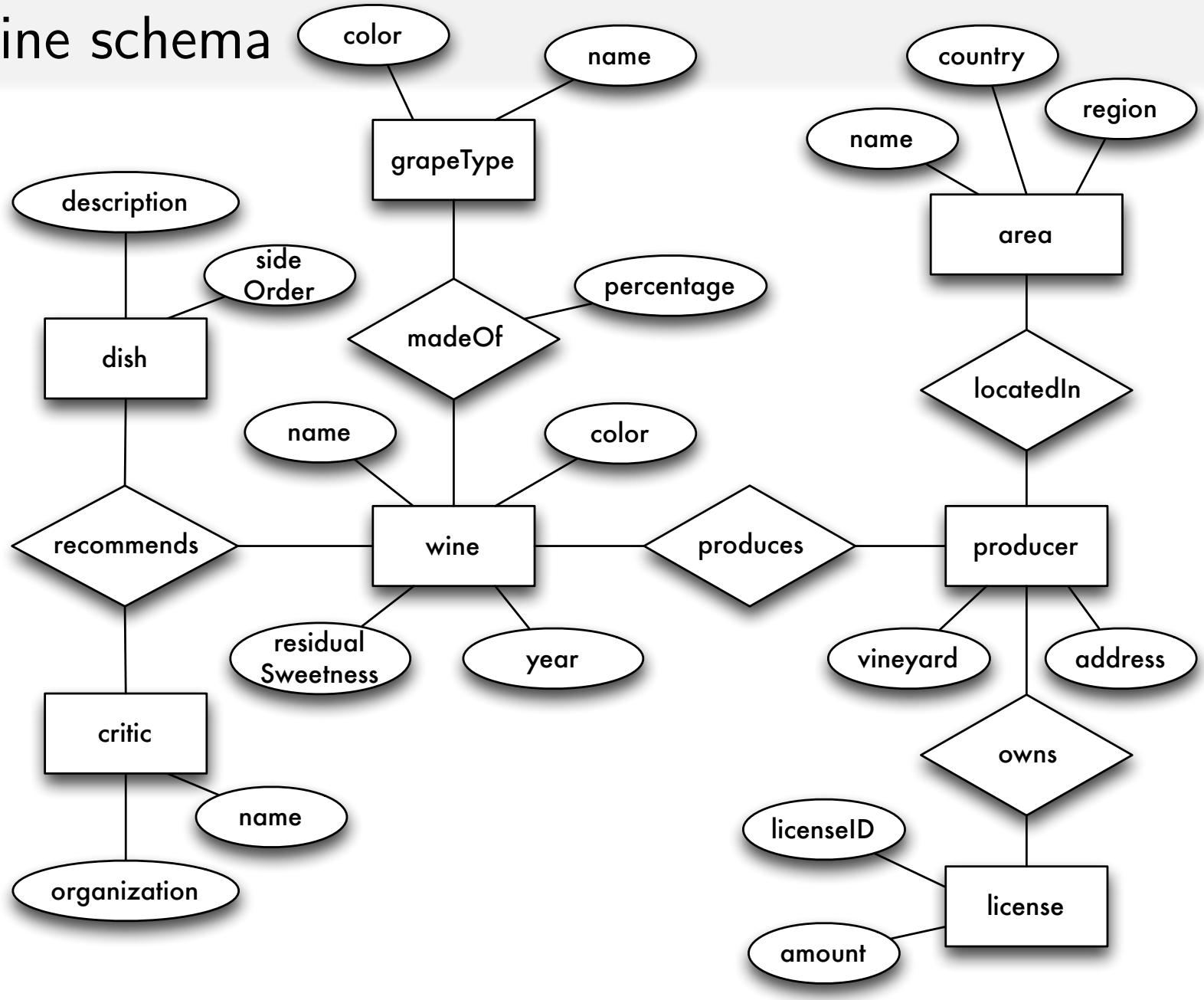
# Entity Relationship Model (ERM)

# Example scenarios

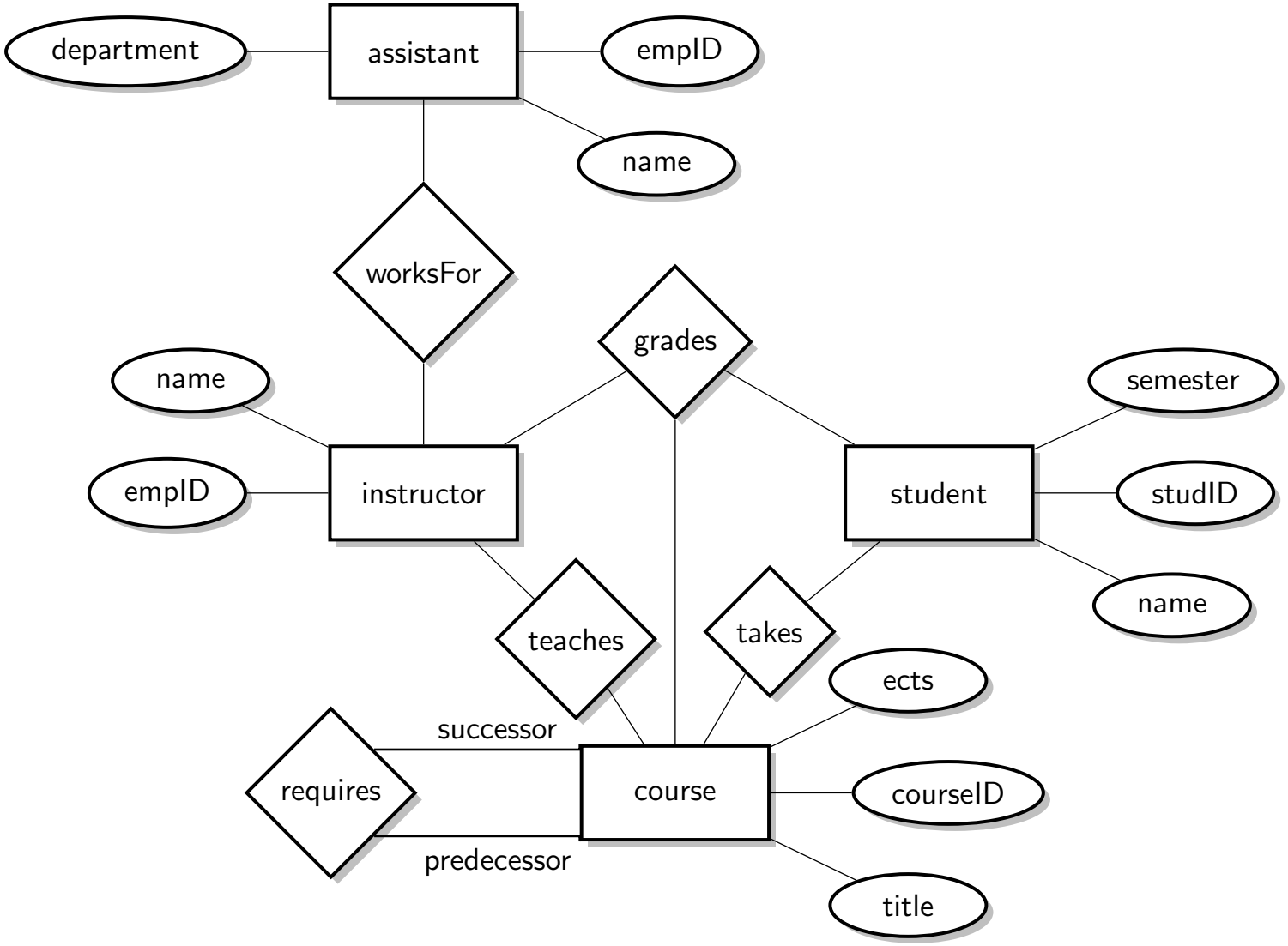
Example scenarios used on the slides

- University: students, instructors, courses,...
- Wine: wine, producers, regions,...

# Wine schema


color
name
grapeType
description
side
Order
dish
madeOf
percentage
name
color
wine
recommends
produces
residual
Sweetness
year
critic
name
organization
country
region
name
area
locatedIn
producer
vineyard
address
owns
licenseID
license
amount

# University schema (different from the book!)



# Entities and entity types

- **Entities** are objects of the real world about which we want to store information
- Only characteristics of entities can be stored in a database (description), not the entity itself!

# Entities and entity types

- **Entities** are objects of the real world about which we want to store information
- Only characteristics of entities can be stored in a database (description), not the entity itself!

Entities are grouped into **entity types**.

# Entities and entity types

- **Entities** are objects of the real world about which we want to store information
- Only characteristics of entities can be stored in a database (description), not the entity itself!

Entities are grouped into **entity types**.


The **extension** of an entity type (**entity set**) is a particular collection of entities.

# Entities and entity types

- **Entities** are objects of the real world about which we want to store information
- Only characteristics of entities can be stored in a database (description), not the entity itself!

Entities are grouped into **entity types**.

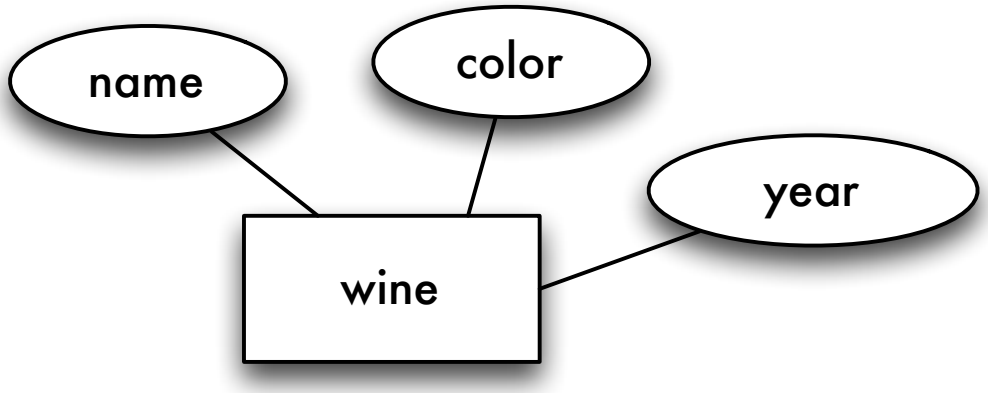

The **extension** of an entity type (**entity set**) is a particular collection of entities.

Often the two terms entity set and entity type are used as synonyms (also in the book).

# Attributes

**Attributes** model characteristics of entities or relationships.

- All entities of an entity type have the same characteristics
- Attributes are declared for entity types
- Attributes have a domain or value set



# Attributes

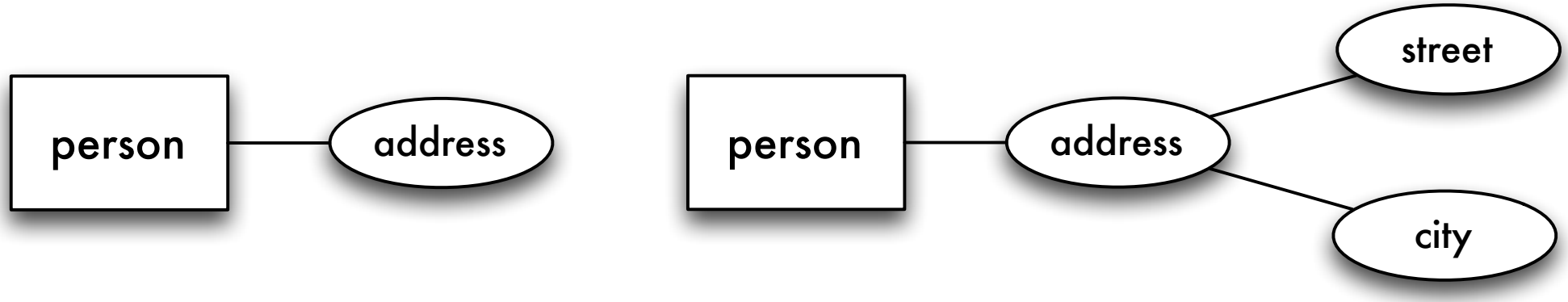
**Single-valued** vs. **multi-valued** attributes

- A person might have multiple phone numbers (or a single one)

# Attributes

**Simple** attributes vs. **composite** attributes

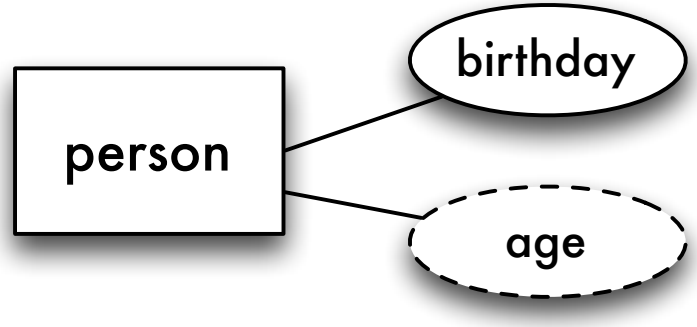
- An address can be modeled as a string or composed of street and city



# Attributes

Stored attributes vs. **derived** attributes

- E.g.: birthdate and age



# Keys

A **(super) key** consists of a subset of an entity type's attributes $E(A_1, \ldots, A_m)$

$$\{S_1, \ldots, S_k\} \subseteq \{A_1, \ldots, A_m\}$$

The attributes $S_1, \ldots, S_k$ of the key are called **key attributes**.

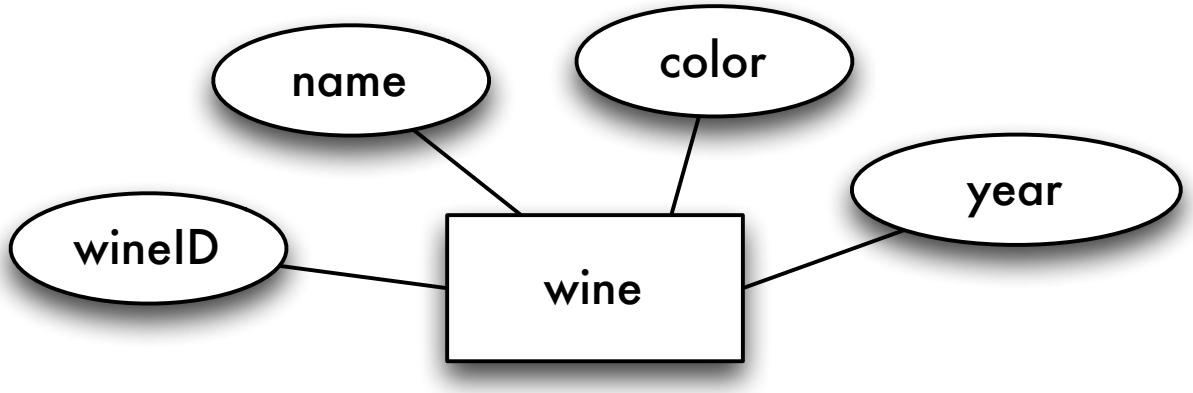
The key attribute's values uniquely identify an individual entity.

A **candidate key** corresponds to a *minimal* subset of attributes that fulfills the above condition.

If there are multiple **candidate keys**, one is chosen as **primary key**.

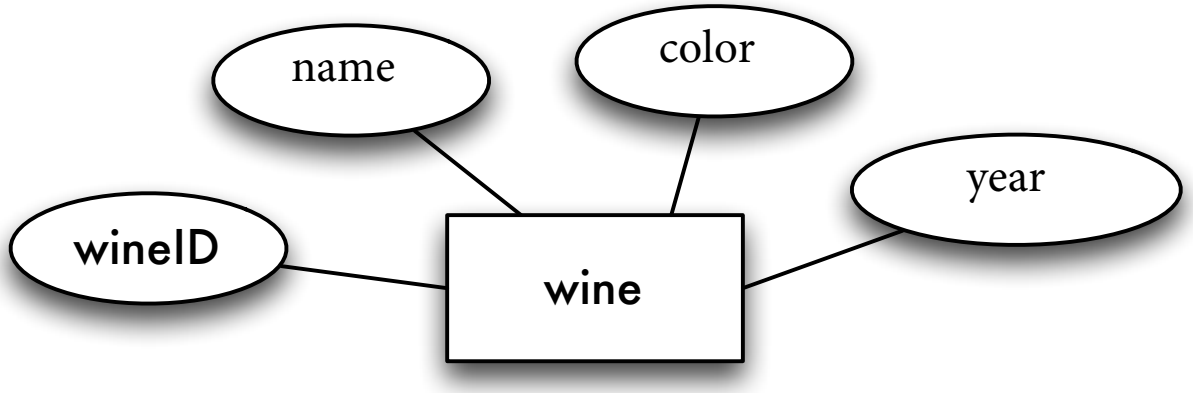
# Primary keys

Primary key attributes are marked by underlining.



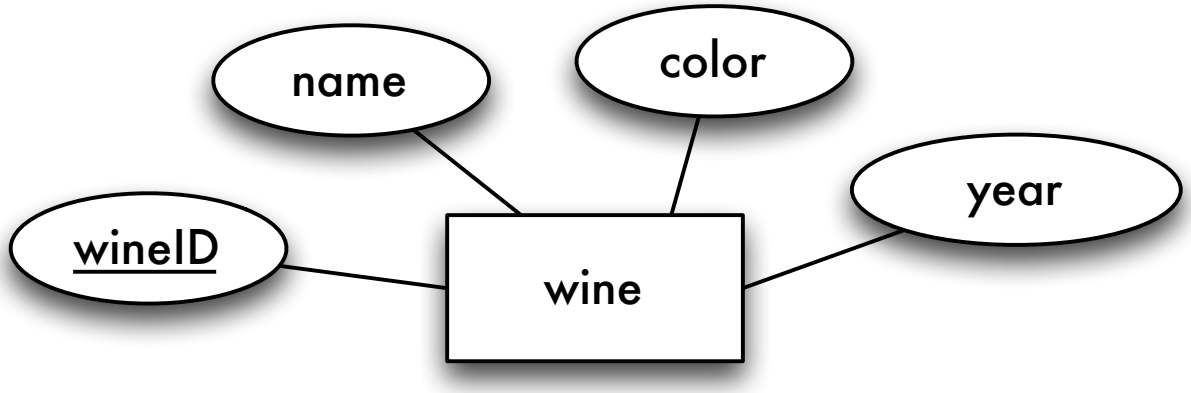
# Primary keys

Primary key attributes are marked by underlining.



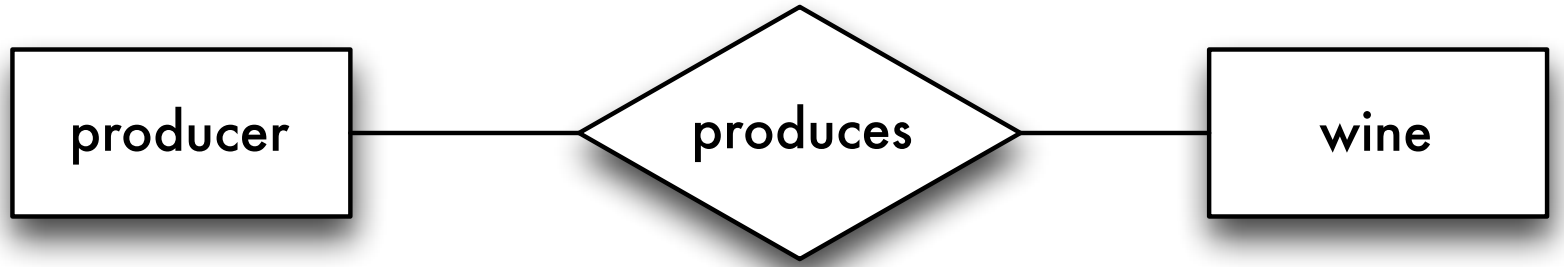
# Primary keys

Primary key attributes are marked by underlining.



# Relationships and relationship types

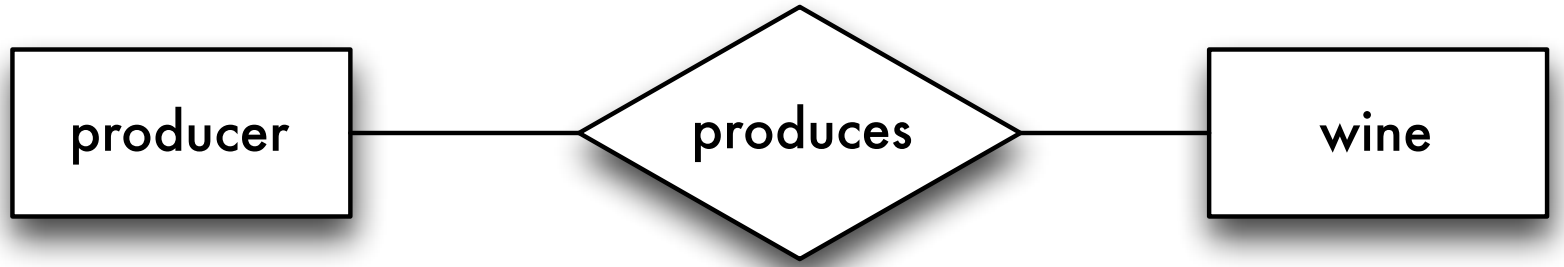
- **Relationships** describe connections between entities.
- Relationships between entities are grouped into **relationship types**.




An association between two or more entities is called **relationship (instance)**. A **relationship set** is a collection of relationship instances.

# Relationships and relationship types

- **Relationships** describe connections between entities.
- Relationships between entities are grouped into **relationship types**.




An association between two or more entities is called **relationship (instance)**. A **relationship set** is a collection of relationship instances.

Often the two terms relationship set and relationship type are used as synonyms (also in the book).

# Mathematical understanding of relationship types

A relationship type $R$ between entity types $E_1, E_2, \ldots, E_n$ can be considered a mathematical **relation**.

**Instance** of a relationship type $R$:

$$R \subseteq E_1 \times E_2 \times \cdots \times E_n$$

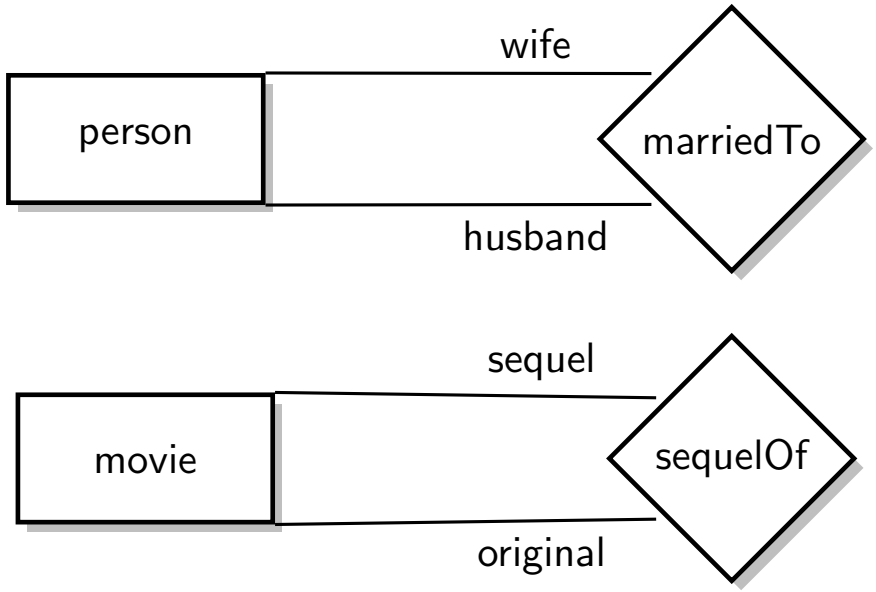
A particular element $(e_1, e_2, \ldots, e_n) \in R$ is called an **instance** of the relationship type with $e_i \in E_i$ for all $1 \leq i \leq n$.

Hint: This notation does not cover attributes of relationship types.

# Recursive relationship types and role names

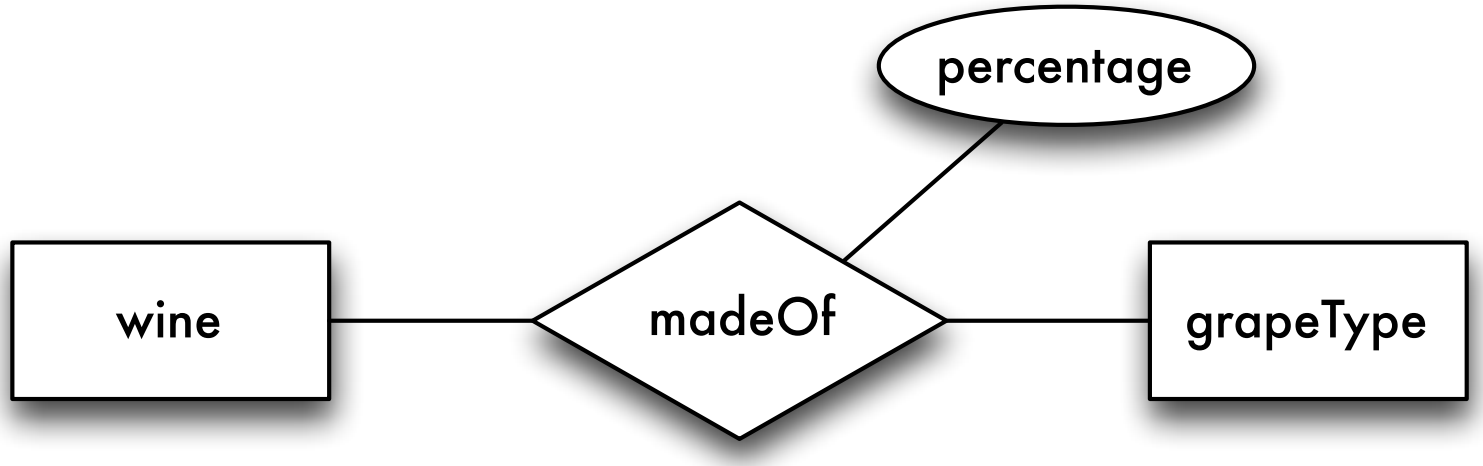
**Role names** are optional and used to characterize a relationship type.

- Especially useful for recursive relationship types, i.e., an entity type is participating multiple times in a relationship type.



# Attributes of relationship types

Relationship types can also have (descriptive) attributes.



# Summary: basic concepts

*students take courses*

# Summary: basic concepts

*students take courses*

1. Entity

# Summary: basic concepts

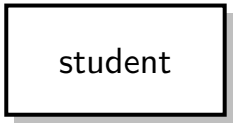
*students take courses*

1. Entity → Entity type

# Summary: basic concepts

*students take courses*

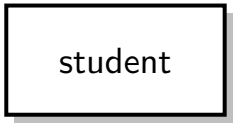
1. Entity → Entity type




# Summary: basic concepts

*students take courses*

1. Entity → Entity type

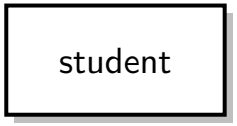



2. Relationship

# Summary: basic concepts

*students take courses*

1. Entity → Entity type

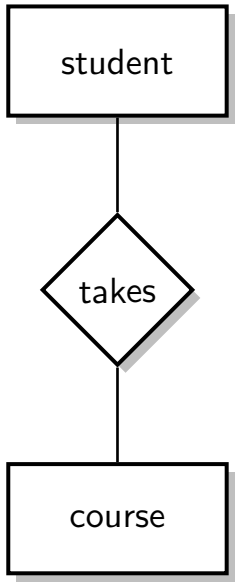



2. Relationship → Relationship type

# Summary: basic concepts

*students take courses*

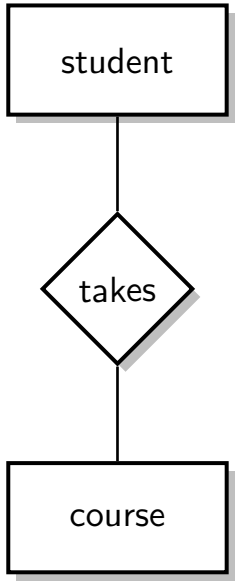
1. Entity → Entity type
2. Relationship → Relationship type



# Summary: basic concepts

*students take courses*

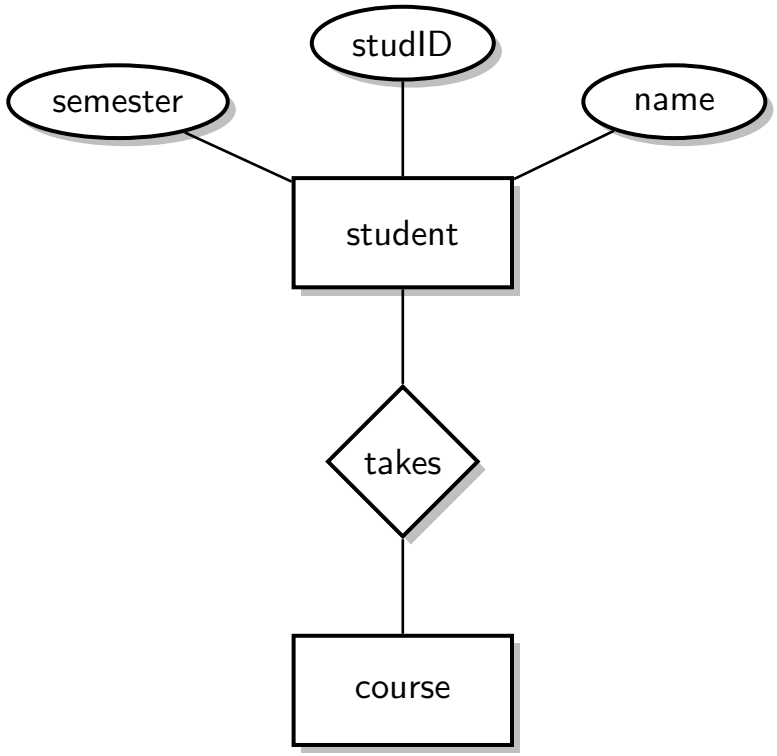
1. Entity → Entity type
2. Relationship → Relationship type
3. Attribute



# Summary: basic concepts

*students take courses*

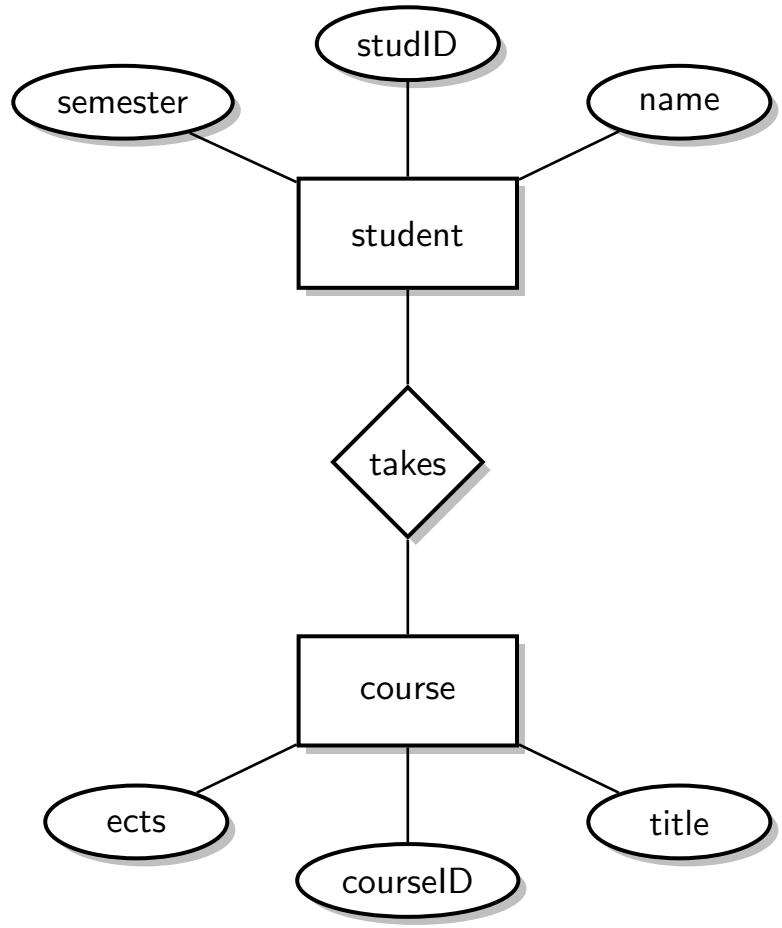
1. Entity → Entity type
2. Relationship → Relationship type
3. Attribute



# Summary: basic concepts

*students take courses*

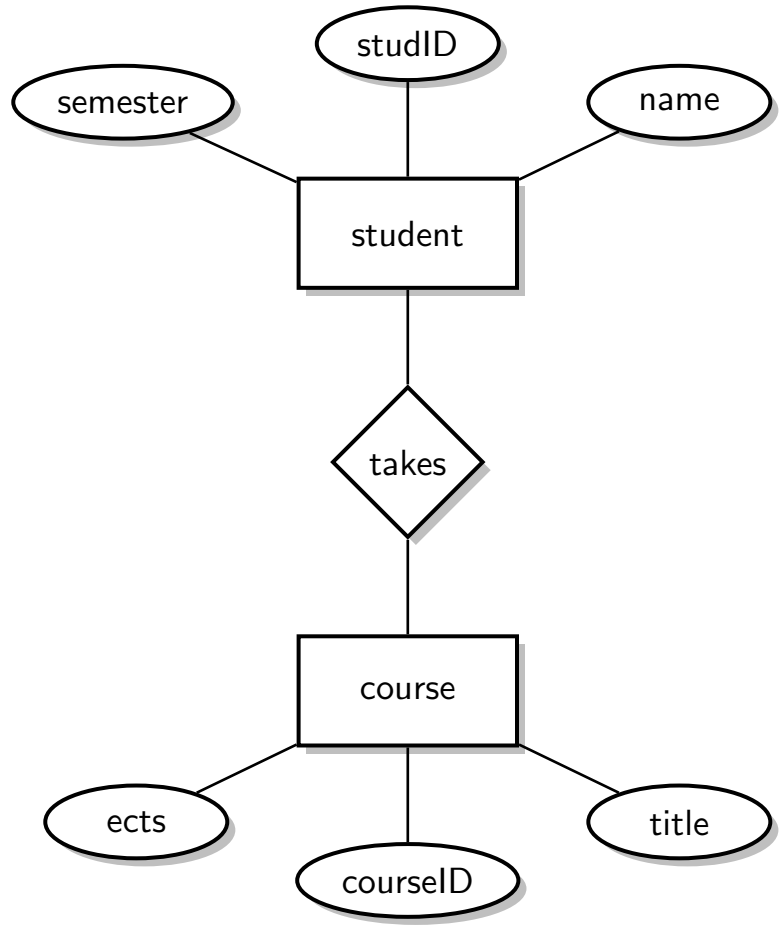
1. Entity → Entity type
2. Relationship → Relationship type
3. Attribute



# Summary: basic concepts

*students take courses*

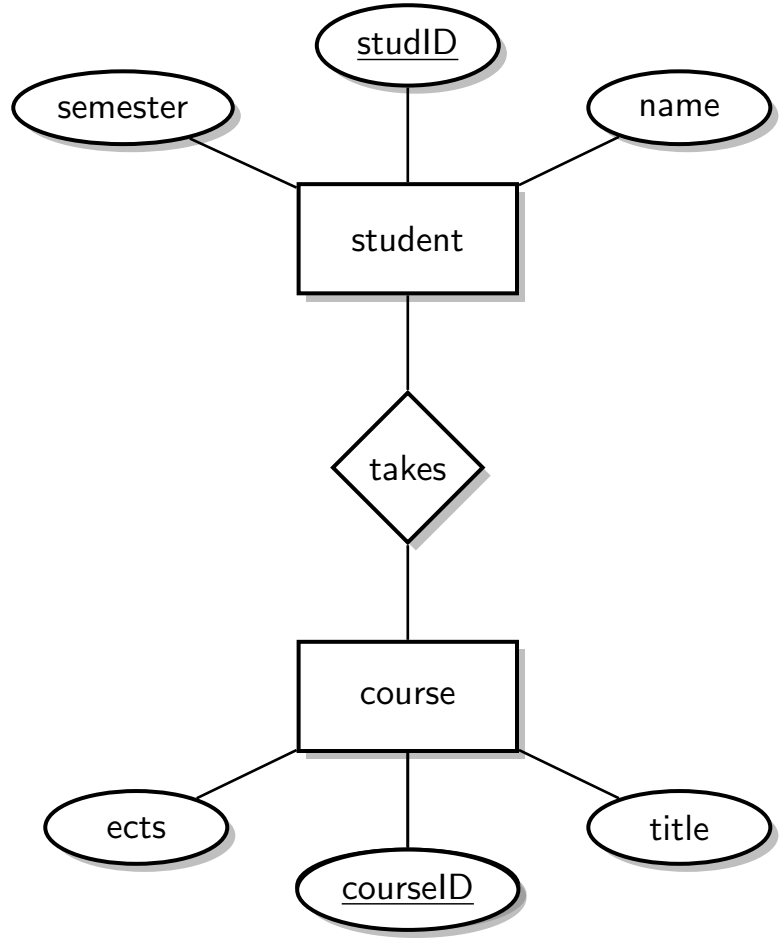
1. Entity → Entity type
2. Relationship → Relationship type
3. Attribute
4. Primary key



# Summary: basic concepts

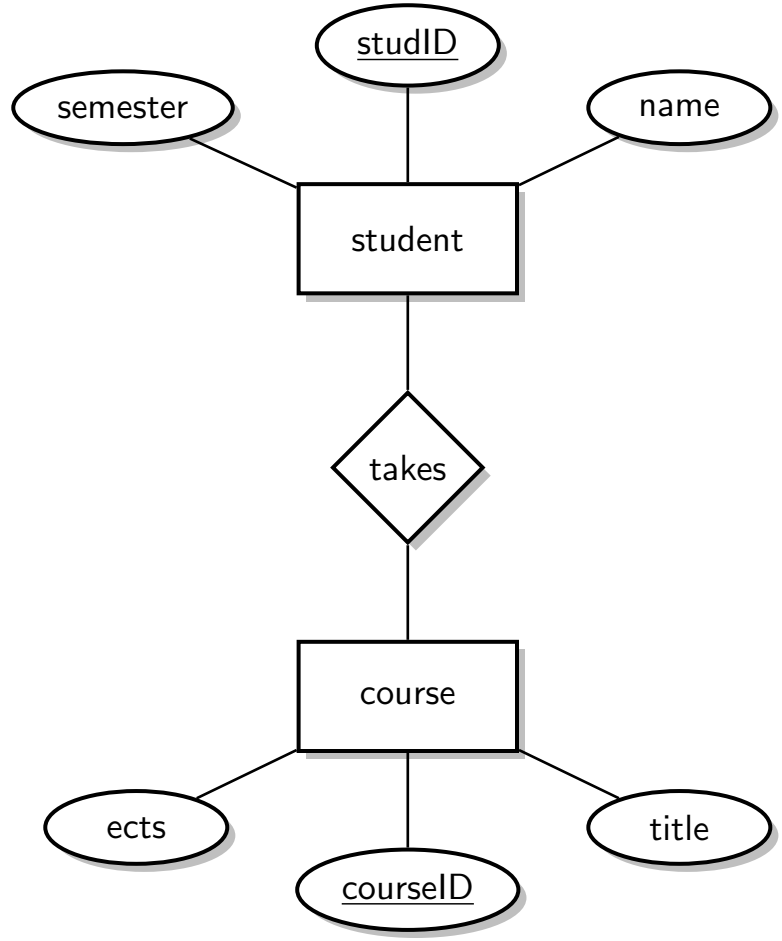
*students take courses*

1. Entity → Entity type
2. Relationship → Relationship type
3. Attribute
4. Primary key



# Summary: basic concepts

*students take courses*

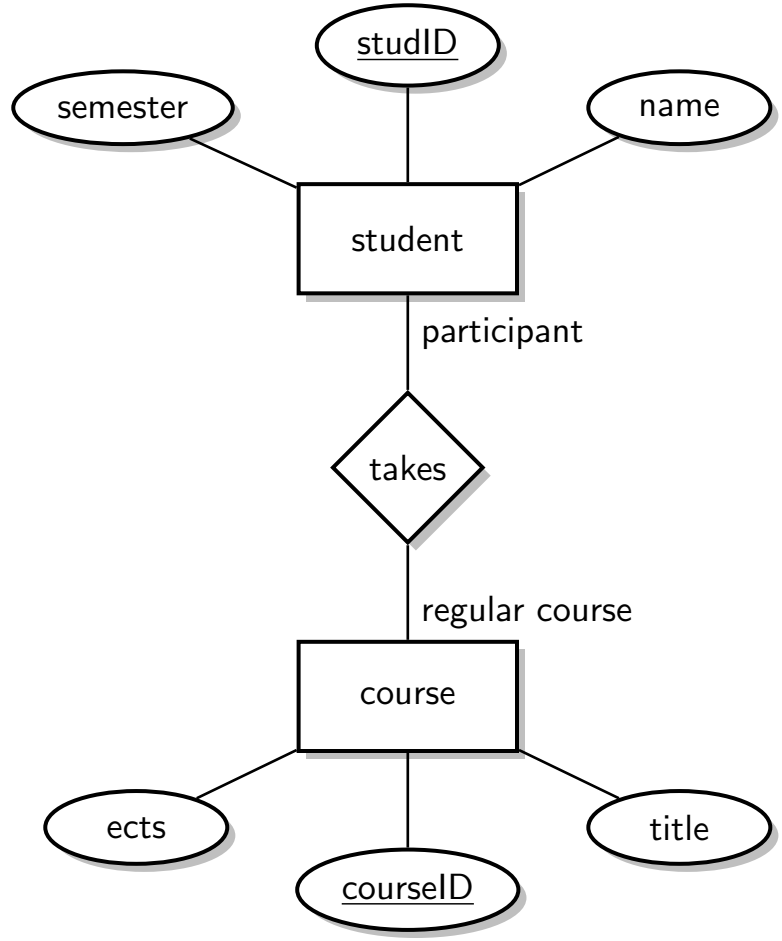



1. Entity → Entity type
2. Relationship → Relationship type
3. Attribute
4. Primary key
5. Role

# Summary: basic concepts

*students take courses*

1. Entity → Entity type
2. Relationship → Relationship type
3. Attribute
4. Primary key
5. Role



# Outline

# Characteristics of relationship types

Degree

- Number of participating entity types
- Mostly: binary
- Rarely: ternary
- In general: n-ary or n-way (multiway relationship types)

# Characteristics of relationship types

Degree

- Number of participating entity types
- Mostly: binary
- Rarely: ternary
- In general: n-ary or n-way (multiway relationship types)

Cardinality ratio / cardinality limits / participation constraint

- Number of times entities are involved in relationship instances
- Cardinality ratio (Chen notation): 1:1, 1:N, N:M
- Participation constraint: partial or total
- Cardinality limits ([min,max] notation): [min,max]

# Characteristics of relationship types

**Degree**

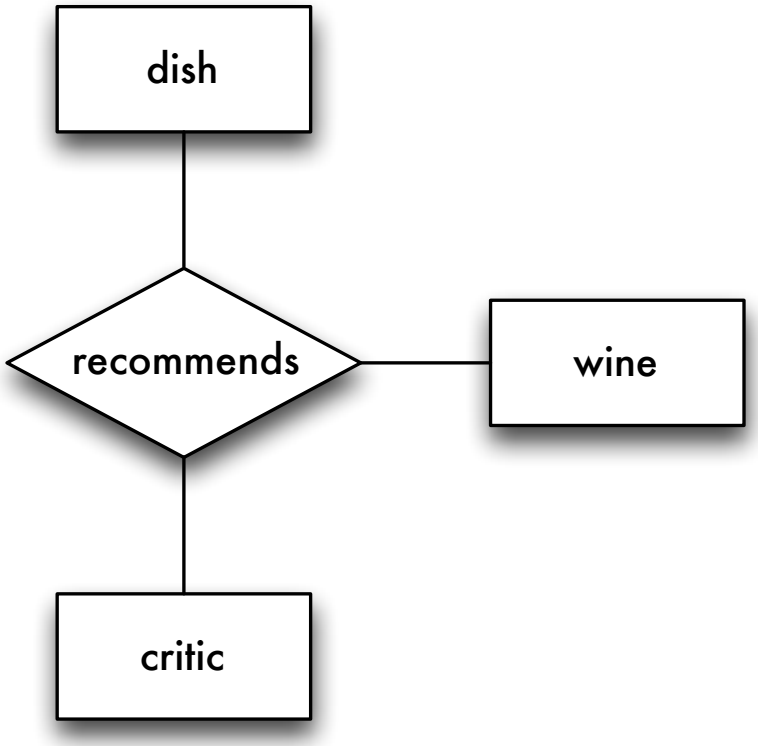
- Number of participating entity types
- Mostly: binary
- Rarely: ternary
- In general: n-ary or n-way (multiway relationship types)

Cardinality ratio / cardinality limits / participation constraint

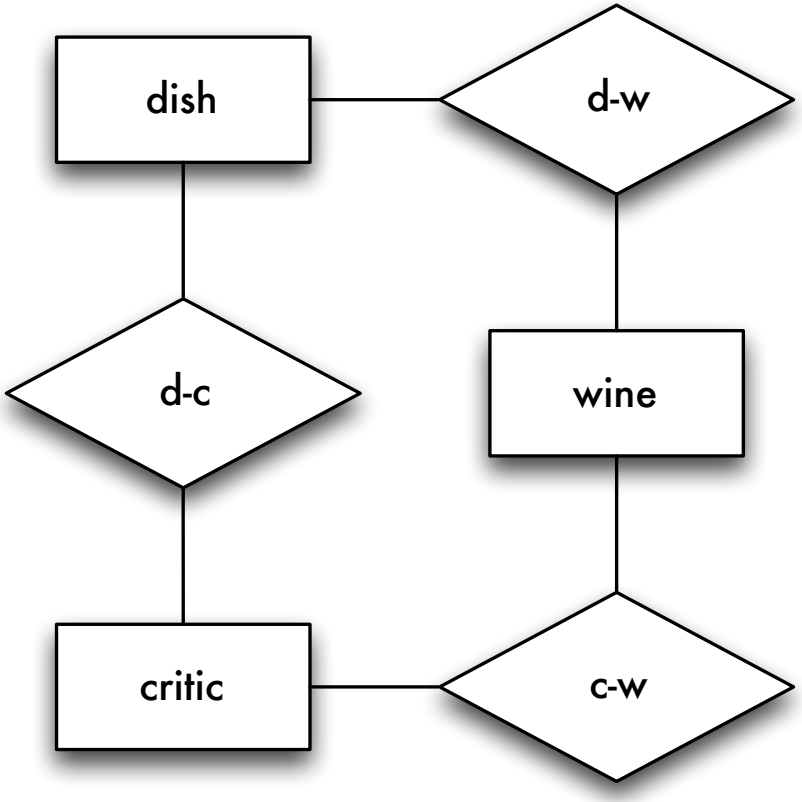
- Number of times entities are involved in relationship instances
- Cardinality ratio (Chen notation): 1:1, 1:N, N:M
- Participation constraint: partial or total
- Cardinality limits ([min,max] notation): [min,max]

# Multiway relationship types

Ternary relationship type

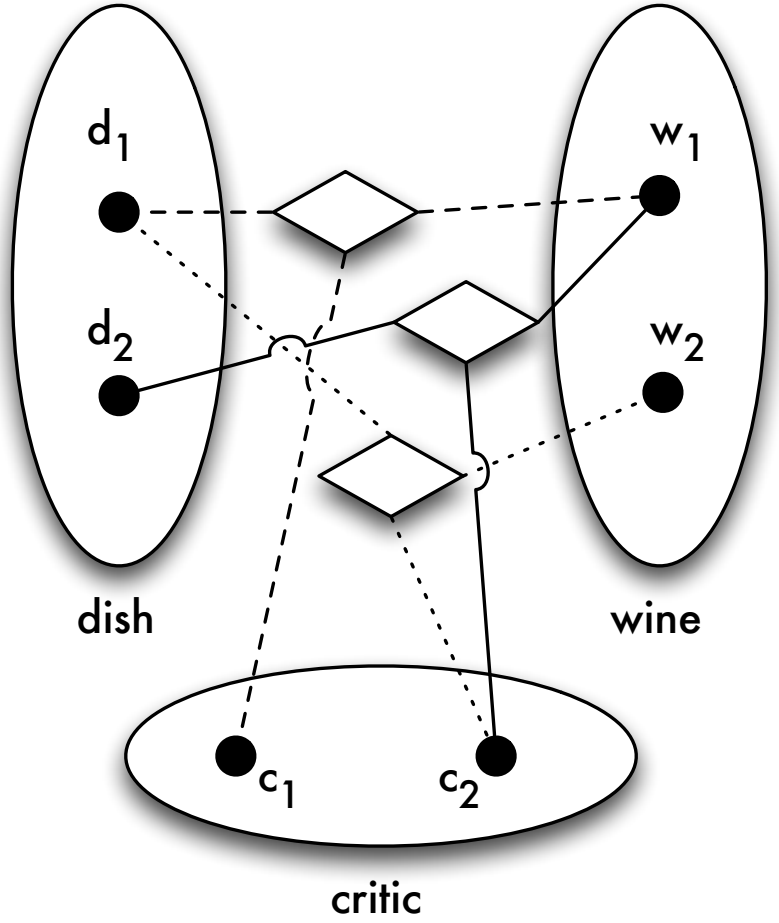



Three binary relationship types

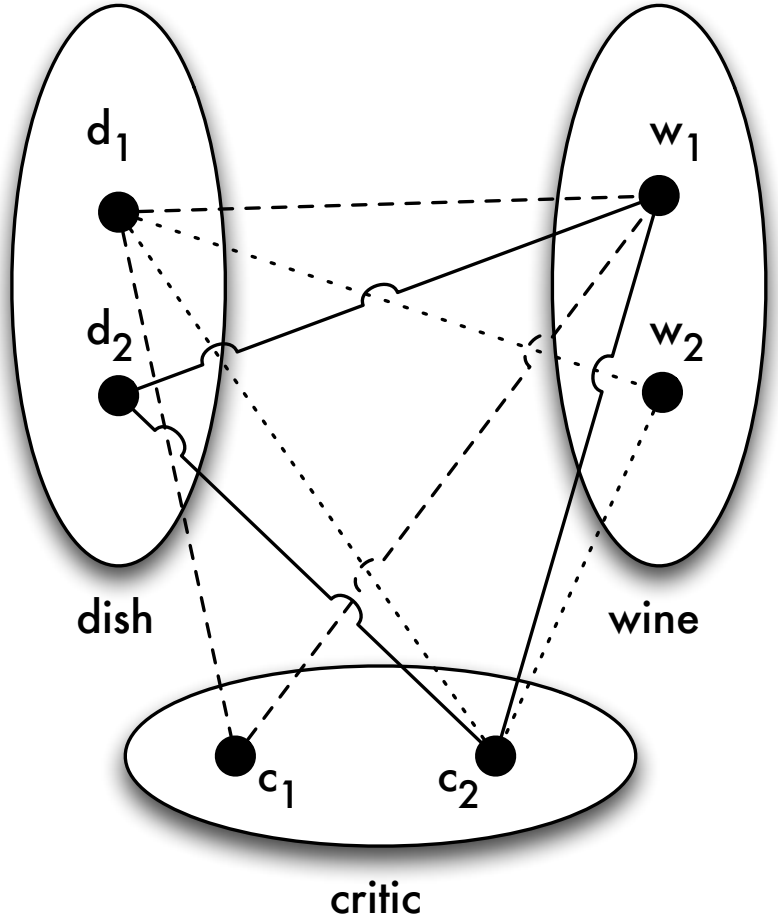


# Multiway relationship types

## Ternary relationship type

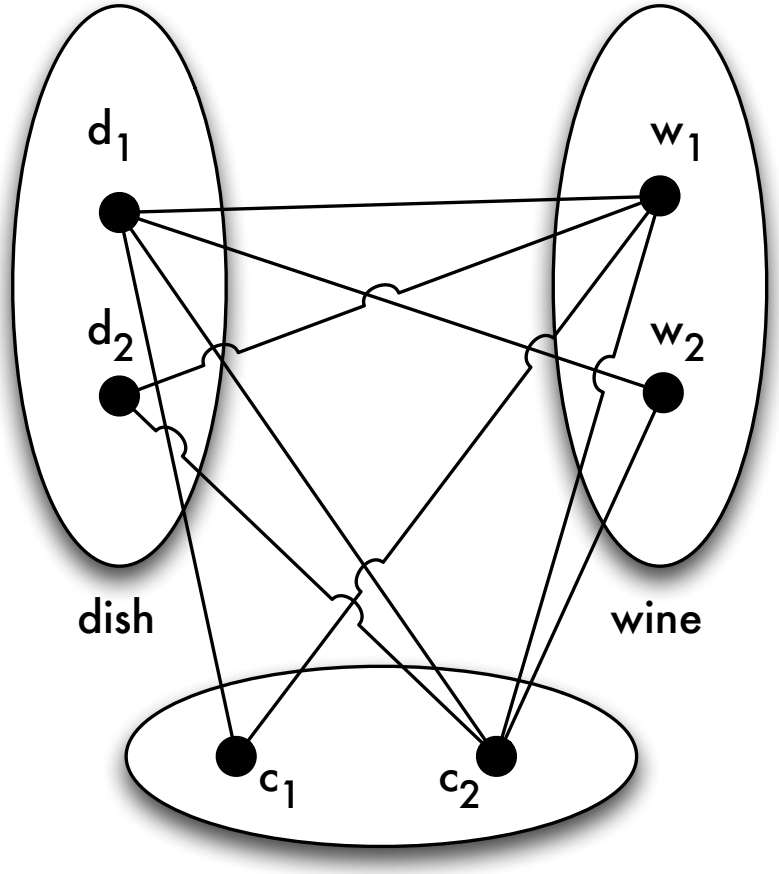



## Three binary relationship types



# Reconstruction of relationship instances

## Binary relationship types

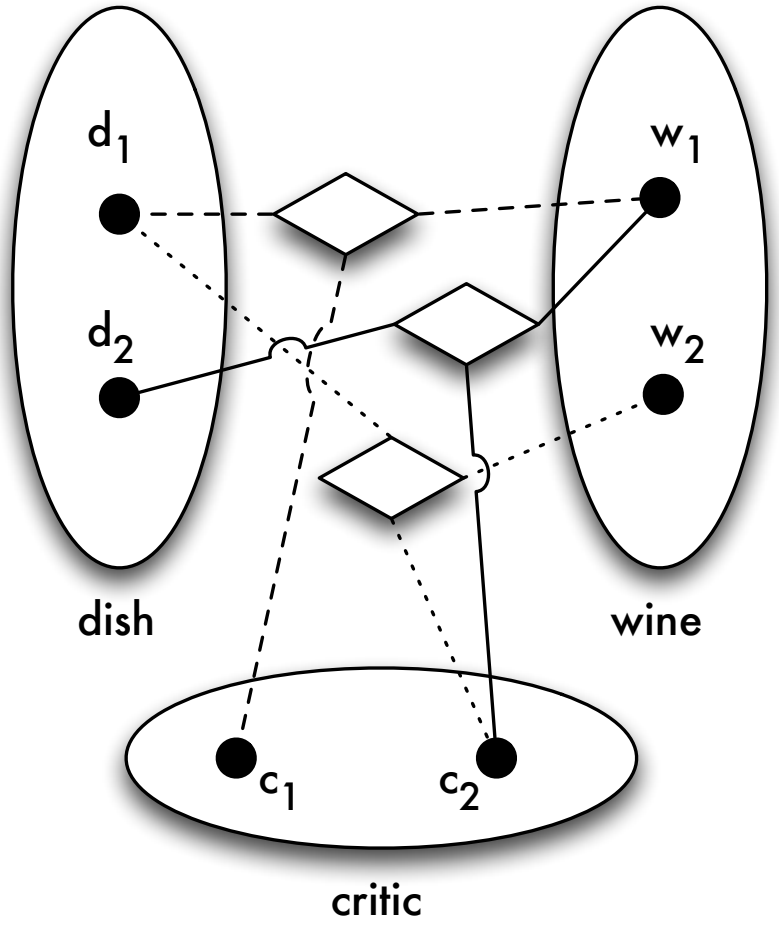



## Reconstructible relationship instances

- $d_1 - c_1 - w_1$
- $d_1 - c_2 - w_2$
- $d_2 - c_2 - w_1$
- but also: $d_1 - c_2 - w_1$

# Binary vs. n-ary relationship types

Ternary relationship type




Using binary relationships we can reconstruct the relationship instance

$$d_1 - c_2 - w_1$$

which is not contained in the ternary relationship type!

# Characteristics of relationship types

Degree

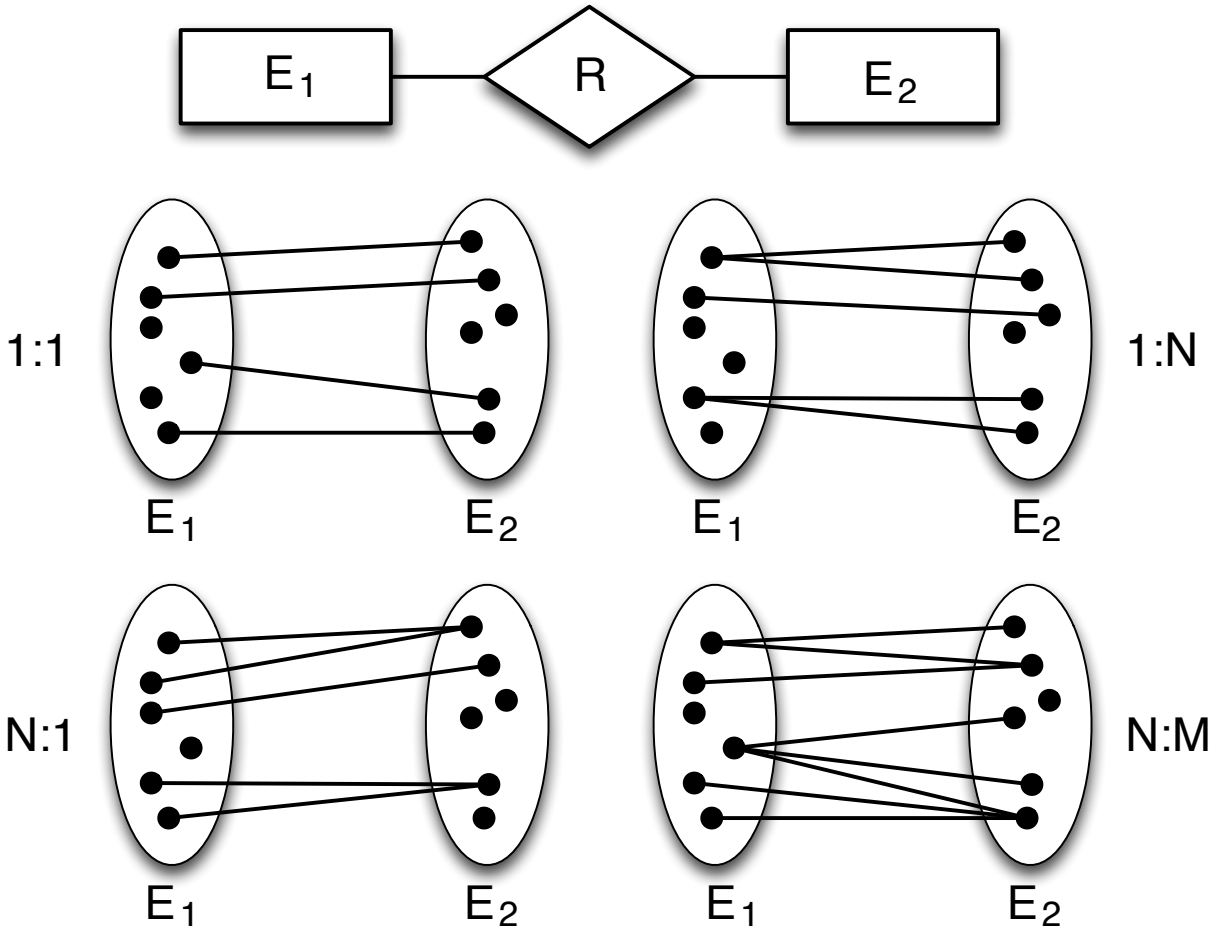
- Number of participating entity types
- Mostly: binary
- Rarely: ternary
- In general: n-ary or n-way (multiway relationship types)

Cardinality ratio / cardinality limits / participation constraint

- Number of entities involved in a relationship instance
- **Cardinality ratio (Chen notation): 1:1, 1:N, N:M**
- **Participation constraint: partial or total**
- Cardinality limits ([min,max] notation): [min,max]

# Chen notation (cardinality ratio)

$$R \subseteq E_1 \times E_2$$




- 1: at most one
- N: arbitrary number

# Functional relationships

1:1, 1:N, and N:1 can be considered a **partial functions** (often also a total function)

1:1 relationship: $R : E_1 \nrightarrow E_2$ and $R^{-1} : E_2 \nrightarrow E_1$

1:N relationship: $R^{-1} : E_2 \nrightarrow E_1$

N:1 relationship: $R : E_1 \nrightarrow E_2$

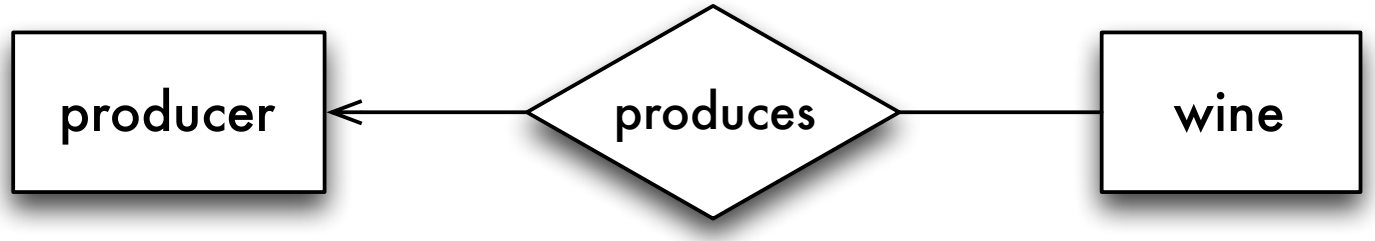
also referred to as **functional relationship**.

The "direction" is important!
The function always leads from the "N" entity type to the "1" entity type.

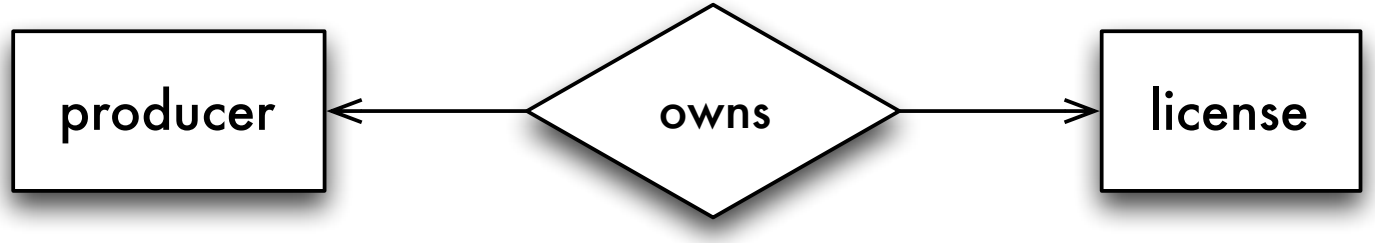
In the context of this lecture, we do not distinguish between partial ($\nrightarrow$) and total functions ($\rightarrow$). Hence, we simply write $\rightarrow$.

# Graphical notation

1:N relationship type



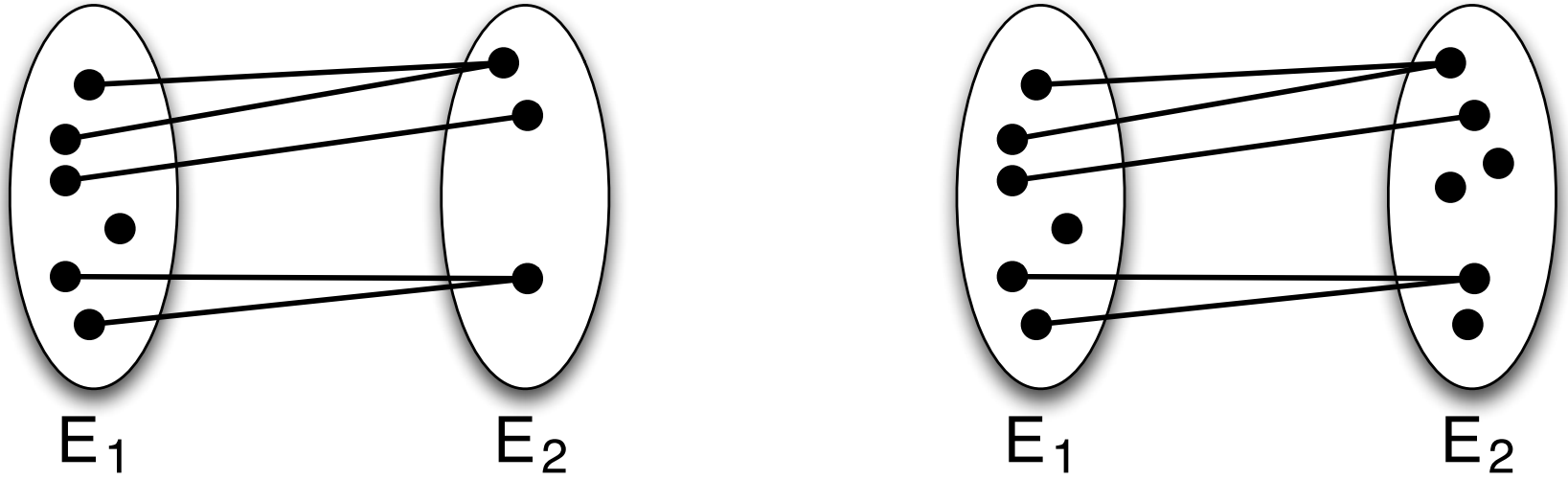

1:1 relationship type



# Participation constraint

## Total

Each entity of an entity type **must** participate in a relationship, i.e., it cannot exist without any participation ($E_2$ in the left example).

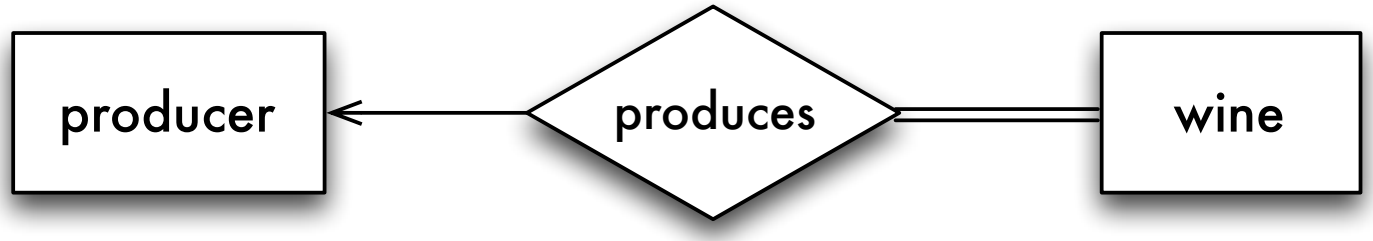



## Partial

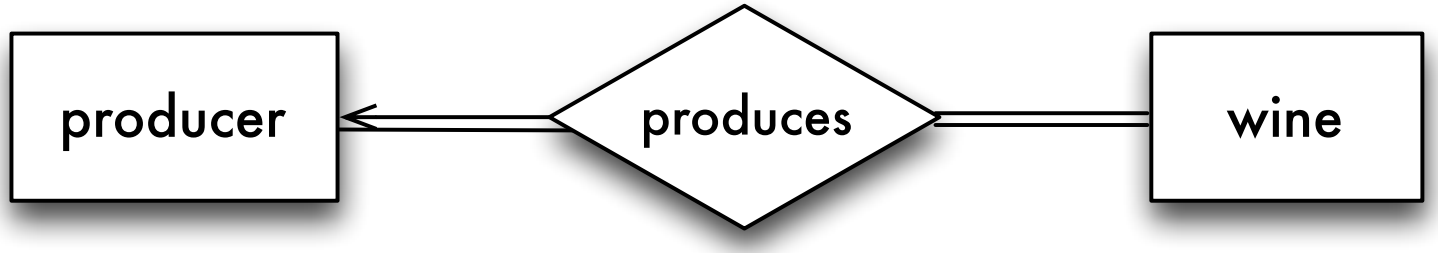
Each entity of an entity type **can** participate in a relationship, i.e., it can exist without any participation.

# Graphical notation

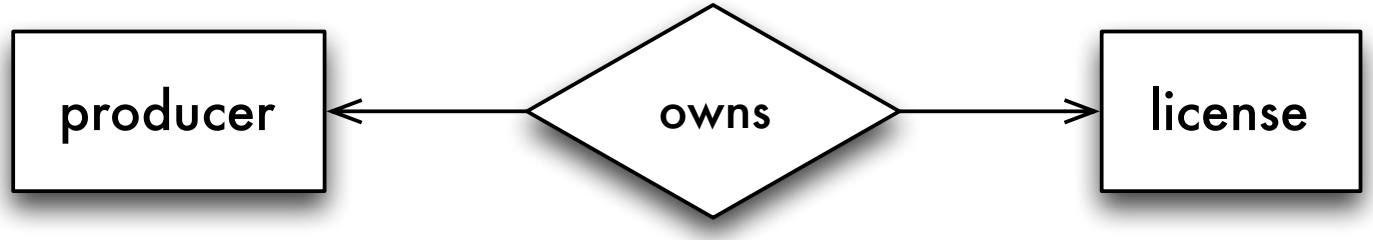
1:N relationship type with total participation of entity type wine



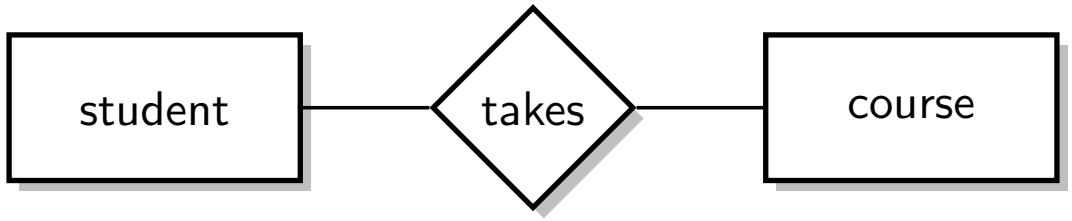

1:N relationship type with total participation of both involved entity types




1:1 relationship type with partial participation

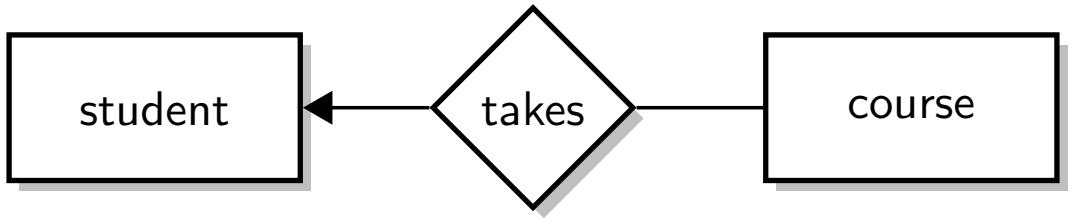


# Overview cardinality ratios




| | |
|---|---|
| Which relationship type is it? | N:M |
| How many students are there in a course? | arbitrary |
| How many courses does a student take? | arbitrary |

# Overview cardinality ratios

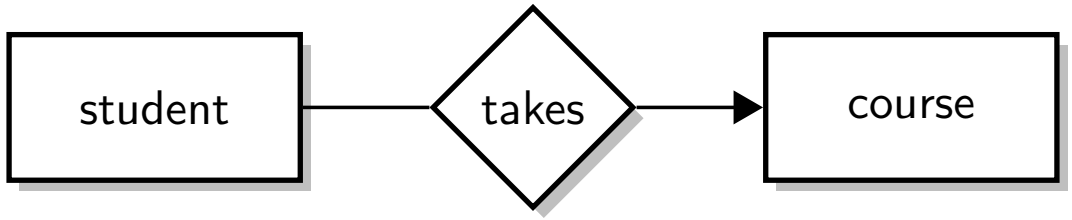



Which relationship type is it?

How many students are there in a course?

How many courses does a student take?

# Overview cardinality ratios

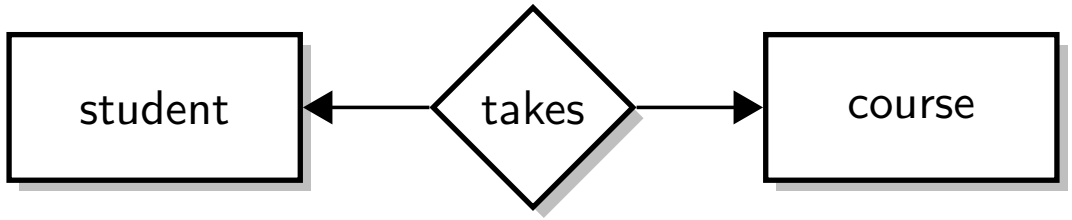



Which relationship type is it?

How many students are there in a course?

How many courses does a student take?

# Overview cardinality ratios

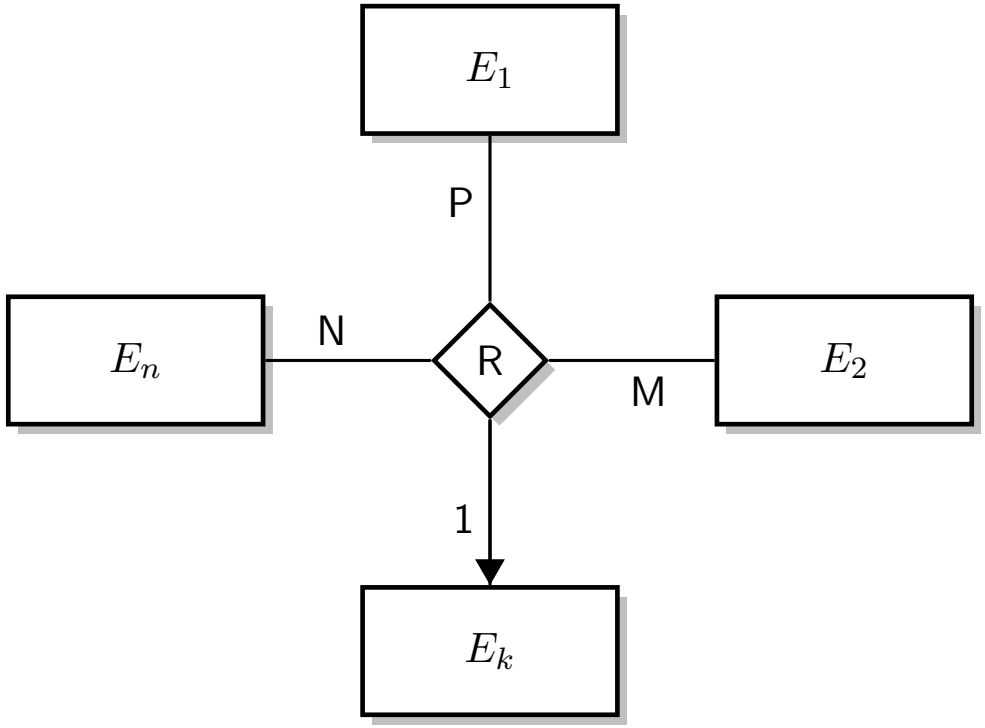



Which relationship type is it?

How many students are there in a course?

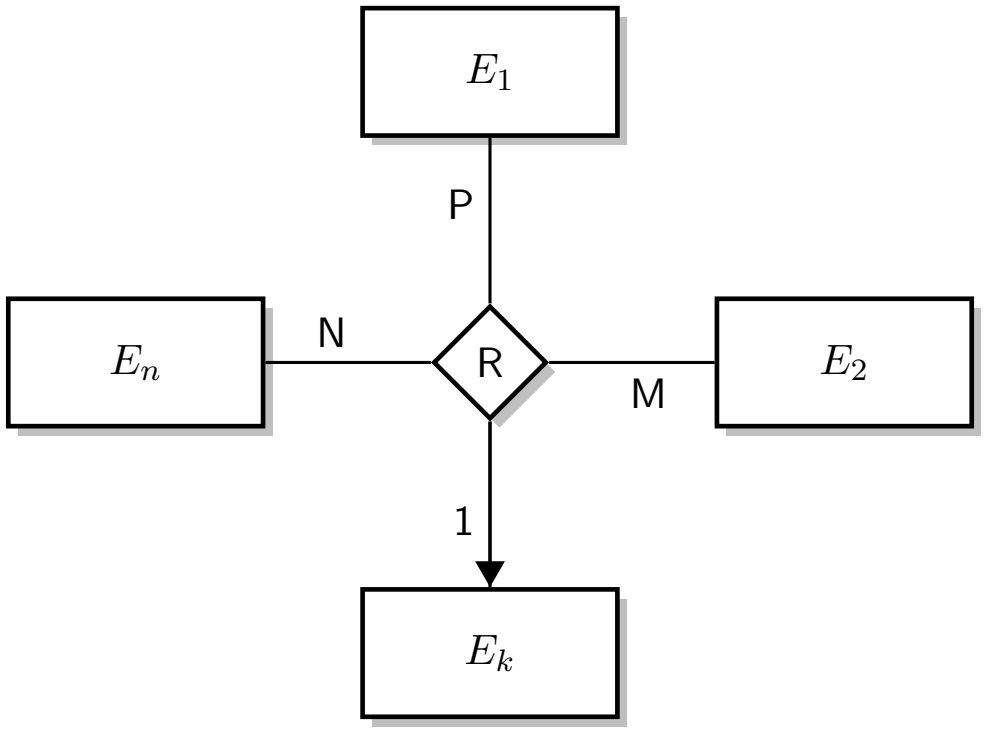
How many courses does a student take?

# Cardinality ratios for n-ary relationship types




$$R : E_1 \times E_2 \times ... \times E_{k-1} \times E_{k+1} \times ... \times E_n \rightarrow E_k$$

# Cardinality ratios for n-ary relationship types

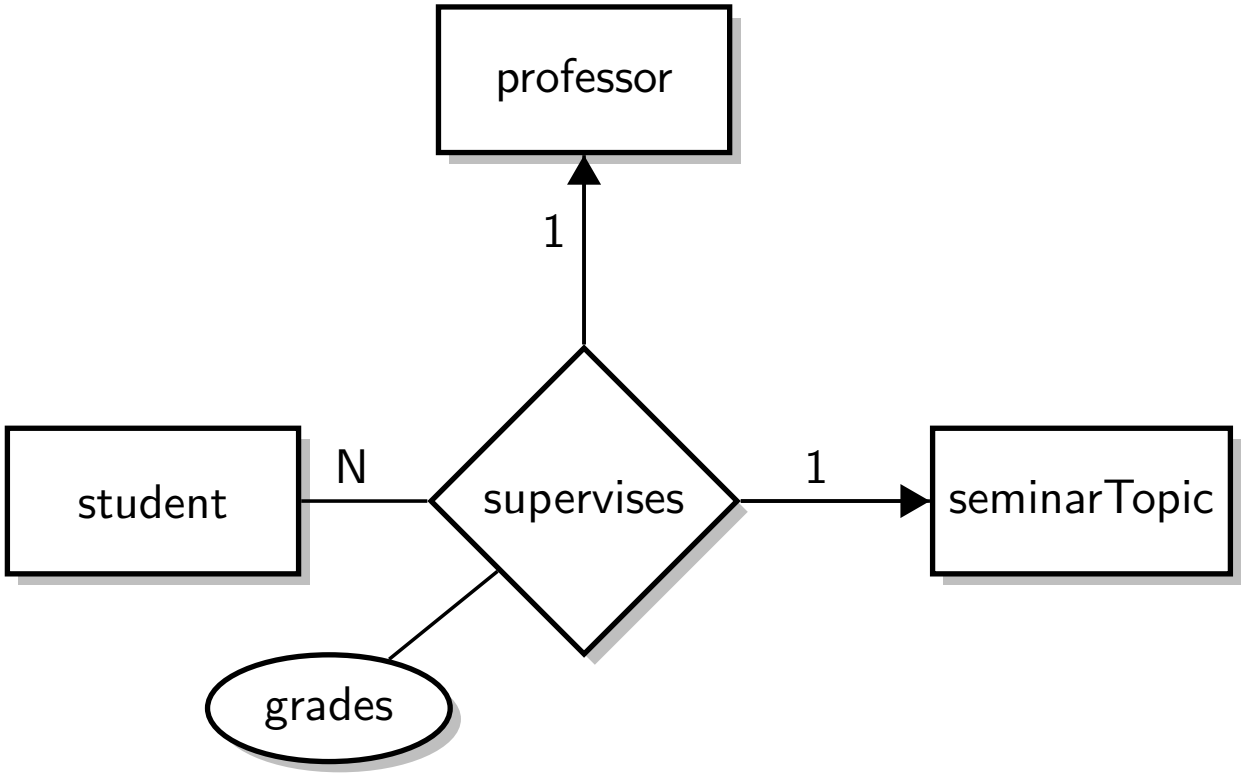



$$R : E_1 \times E_2 \times ... \times E_{k-1} \times E_{k+1} \times ... \times E_n \rightarrow E_k$$

## Remark on notation in general

Using arrows or annotating lines with 1, N, M, etc. is equivalent.
Having both is not necessary but sometimes useful for clarification.

# Example relationship: supervises




supervises: professor $\times$ student $\rightarrow$ seminarTopic

supervises: seminarTopic $\times$ student $\rightarrow$ professor

# Characteristics of relationship types

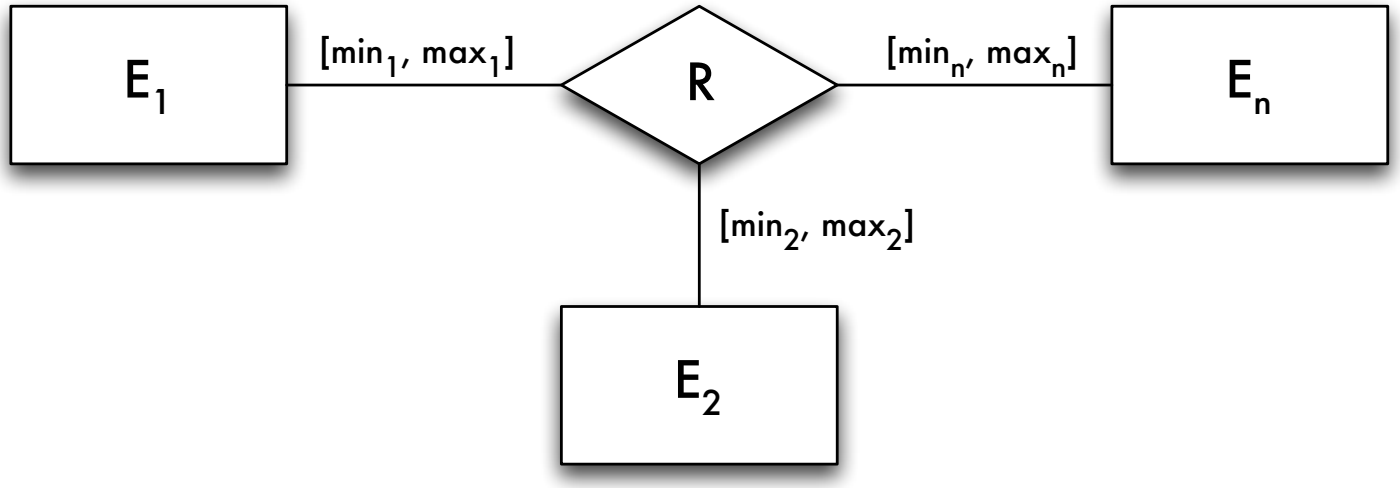
Degree

- Number of participating entity types
- Mostly: binary
- Rarely: ternary
- In general: n-ary or n-way (multiway relationship types)

Cardinality ratio / cardinality limits / participation constraint

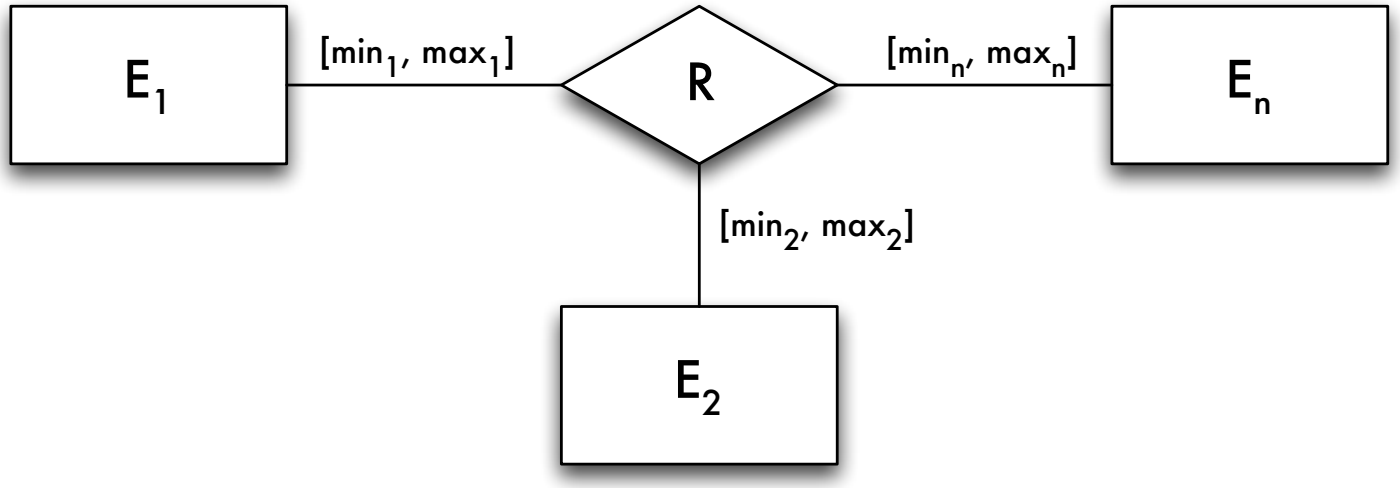
- Number of entities involved in a relationship instance
- Cardinality ratio (Chen notation): 1:1, 1:N, N:M
- Participation constraint: partial or total
- **Cardinality limits ([min,max] notation): [min,max]**

# [min,max] notation (cardinality limits)




Restricts the number of times an entity can participate in a relationship.

# [min,max] notation (cardinality limits)



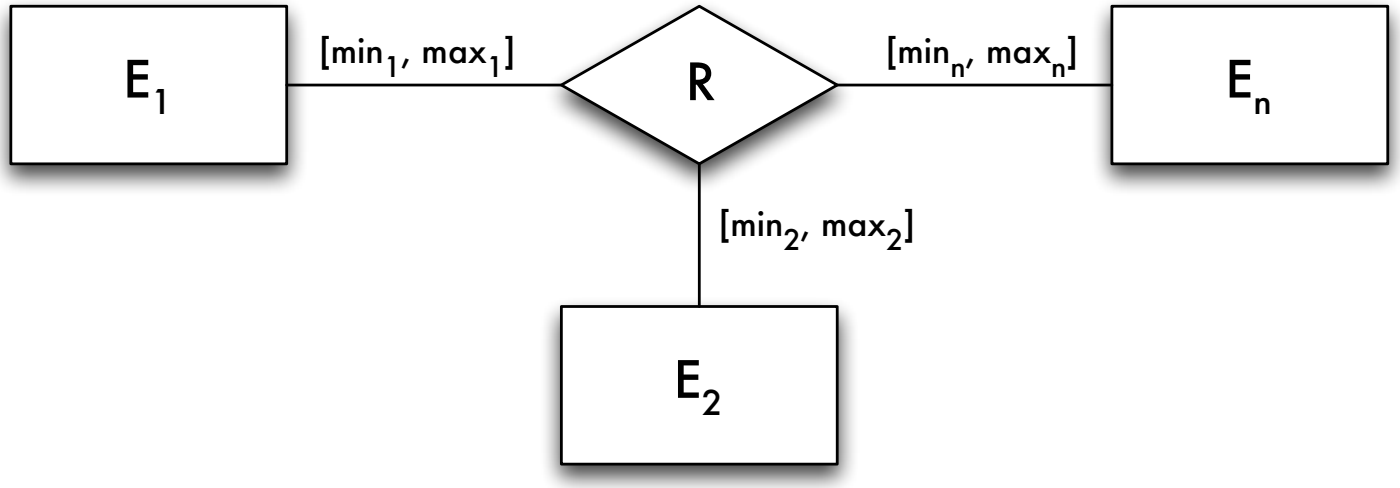

$$R \subseteq E_1 \times E_2 \times ... \times E_i \times ... \times E_n$$

For each $e_i \in E_i$ there exists

- at least $min_i$ instances of relationship type $R$ involving $e_i$ and
- at most $max_i$ instances of relationship type $R$ involving $e_i$

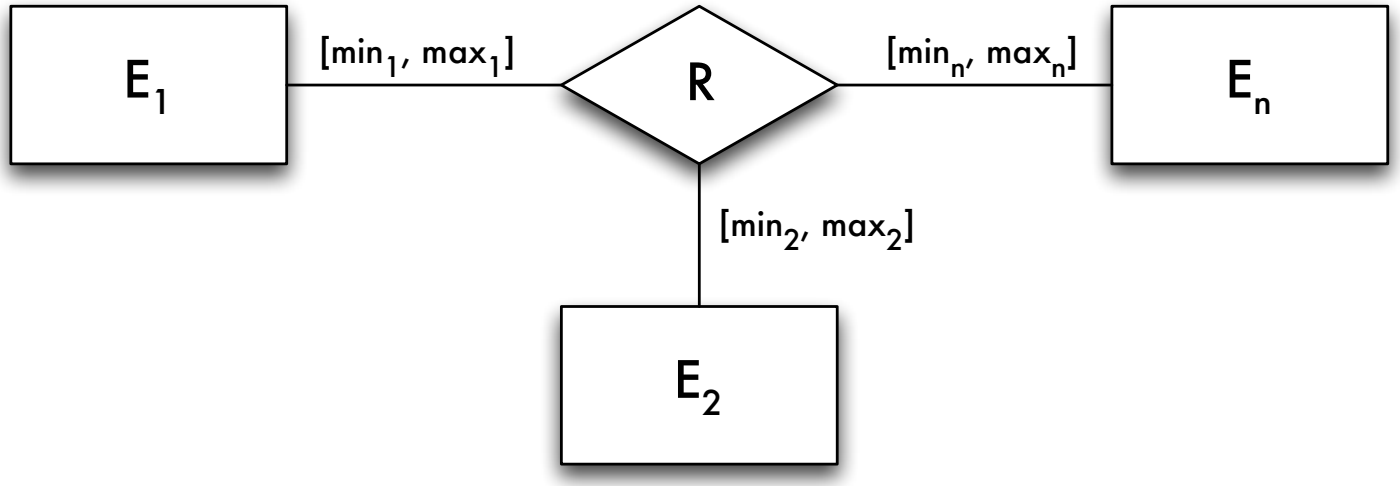
Cardinality constraint: $min_i \leq |\{r \mid r \in R \wedge r.E_i = e_i\}| \leq max_i$

# [min,max] notation (cardinality limits)




Special values for $min_i$: $0$
Special values for $max_i$: $*$

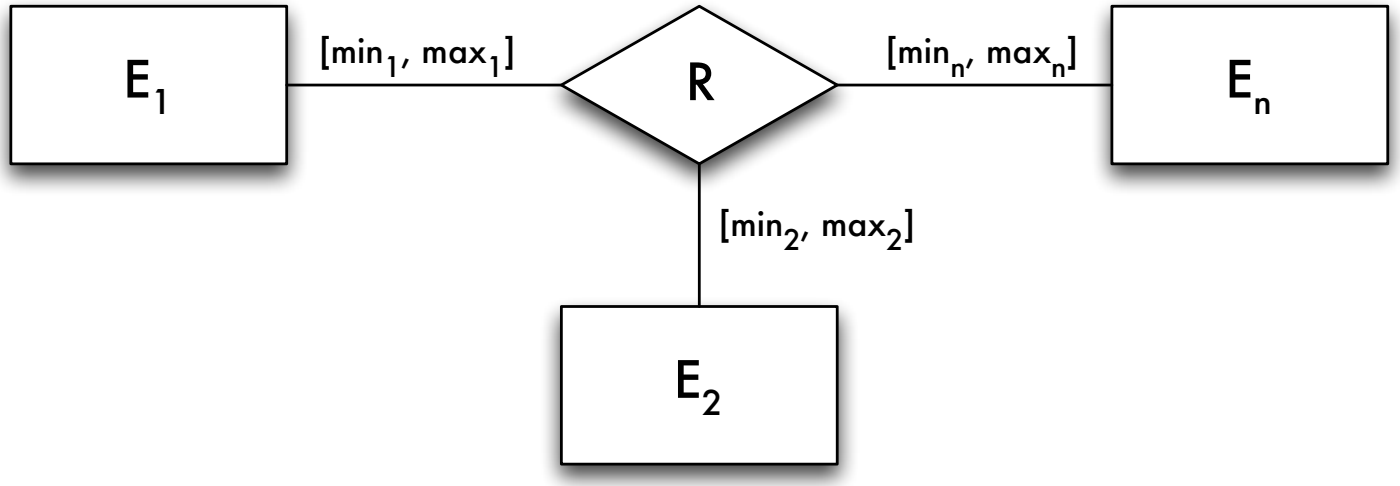
# [min,max] notation (cardinality limits)




Special values for $min_i$: $0$
Special values for $max_i$: $*$

$[0, *]$ represents no restrictions $\rightarrow$ default

# [min,max] notation (cardinality limits)



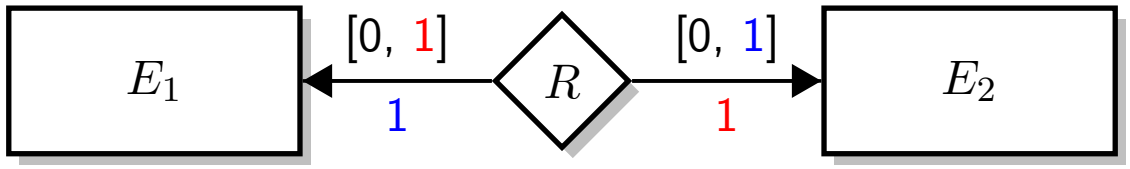

Special values for $min_i$: $0$
Special values for $max_i$: $*$

$[0, *]$ represents no restrictions $\rightarrow$ default

The book uses a slightly different notation: 1..* instead of $[1, *]$

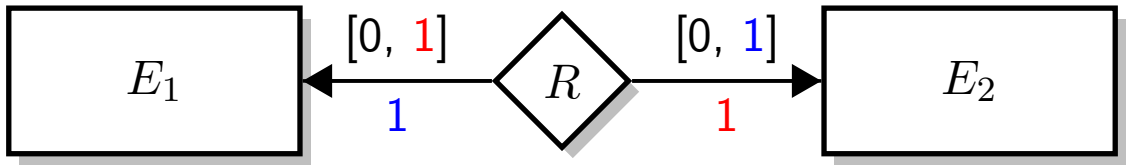
# Chen notation vs [min,max] notation

1:1 relationship type

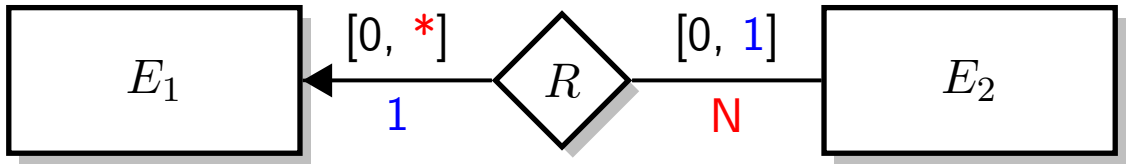


# Chen notation vs [min,max] notation

1:1 relationship type

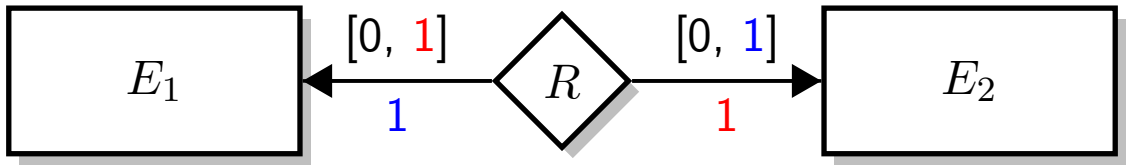



1:N relationship type

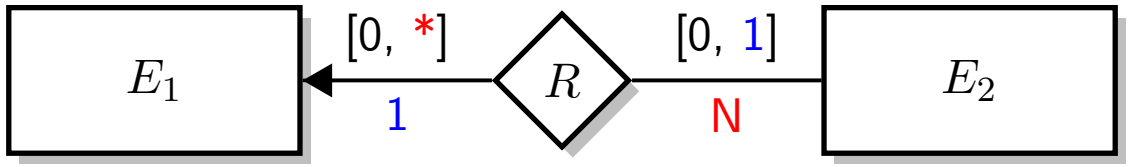


# Chen notation vs [min,max] notation

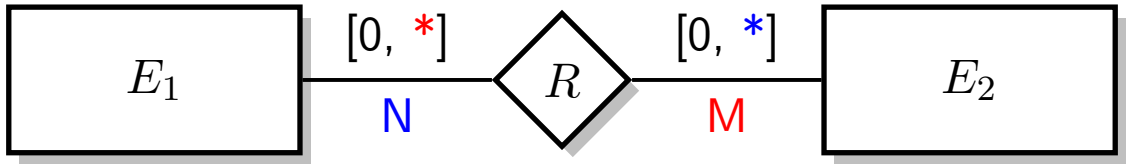
1:1 relationship type




1:N relationship type

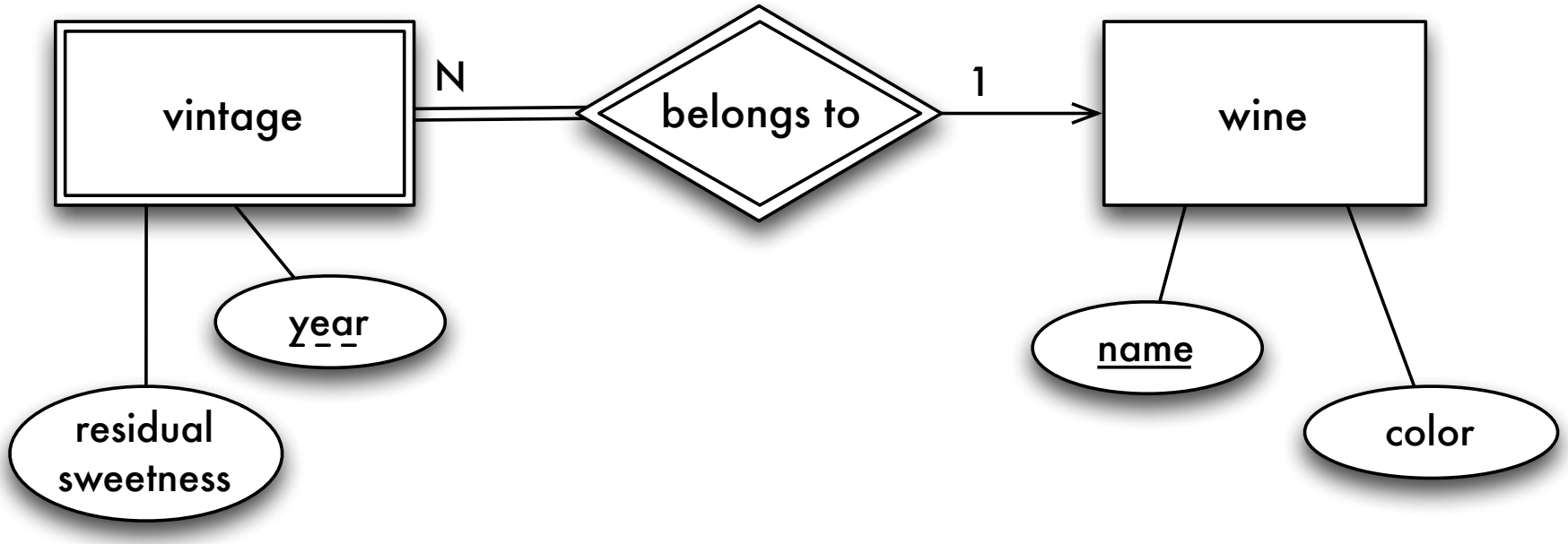



N:M relationship type



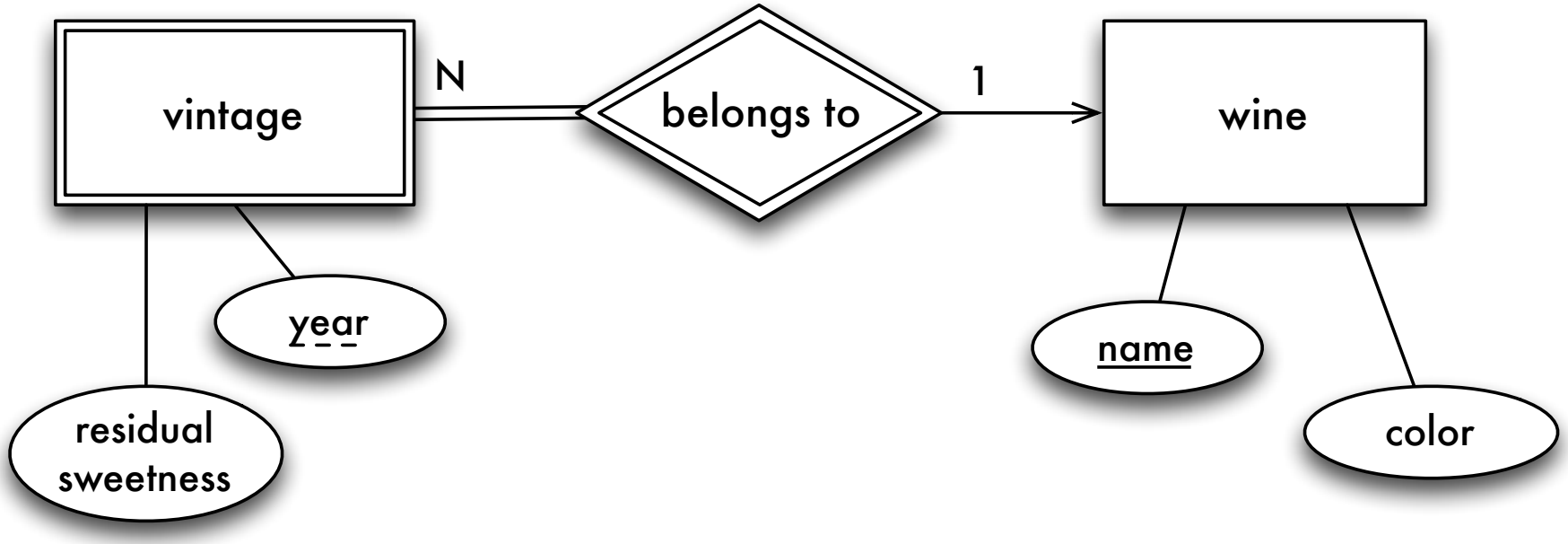
# Outline

# Weak entity types



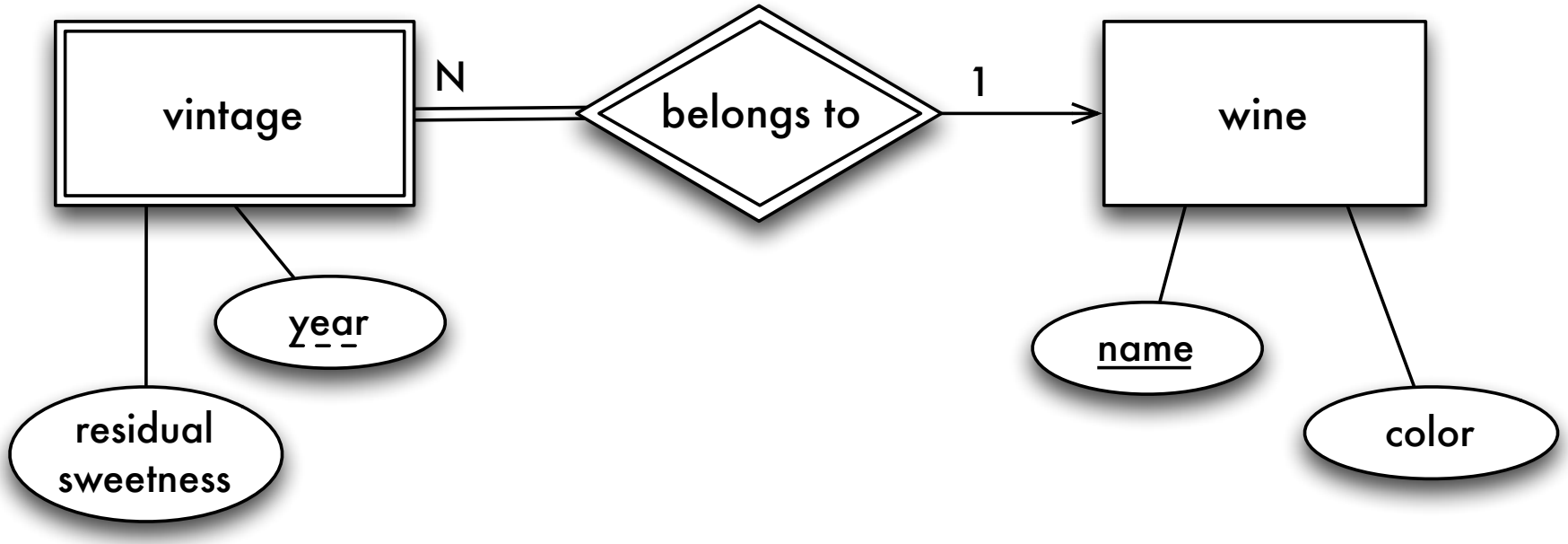

The existence of a **weak entity** depends on the existence of a **strong entity** (aka. identifying or owning entity) associated by an **identifying relationship**.

# Weak entity types




- Total participation of the weak entity type.
- Only in combination with 1:N (N:1) (or rarely also 1:1) relationship types
  The strong entity type is always on the “1”-side

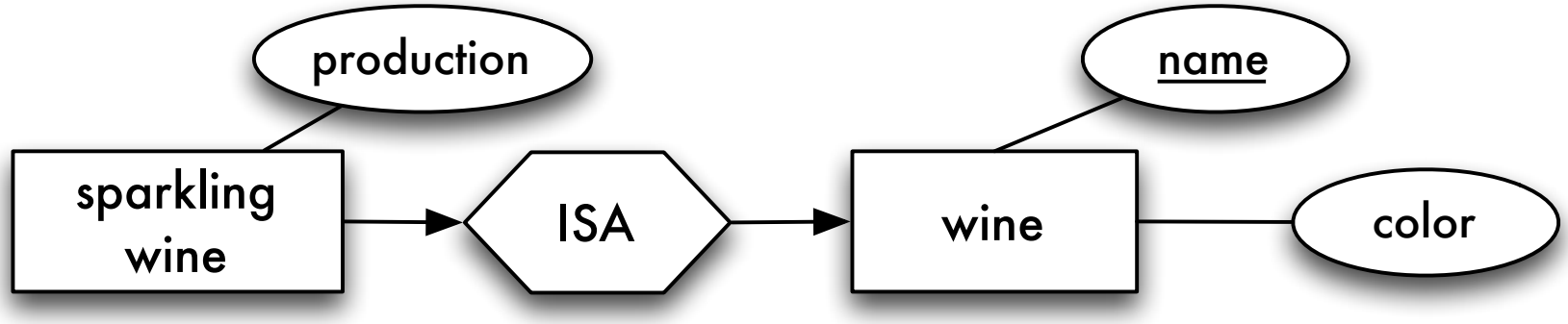
# Weak entity types



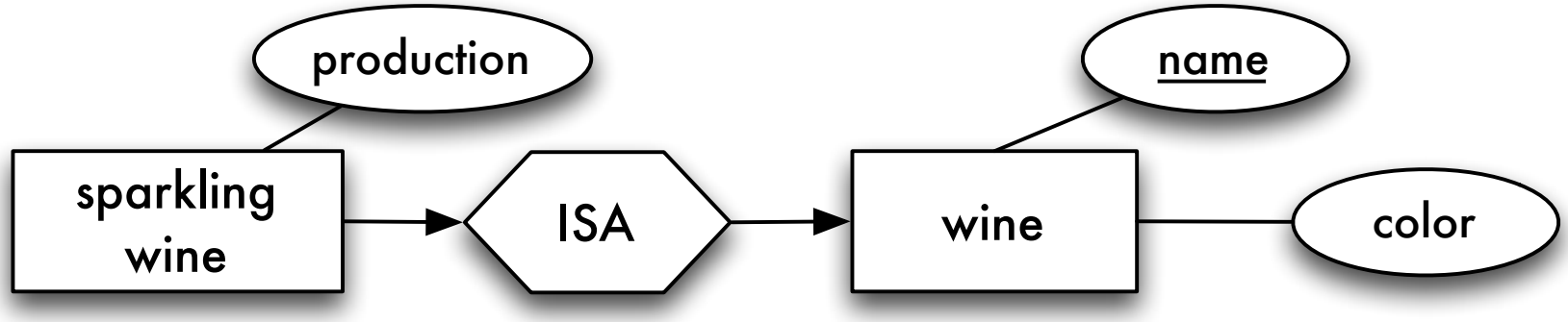

- Weak entities are uniquely identifiable in combination with the corresponding strong entity's key.
- The weak entity type's key attributes are marked by underlining with a dashed line (**partial key**, **discriminator**).

# The ISA relationship type

**Specialization and generalization** is expressed by the ISA relationship type (inheritance).



# Characteristics

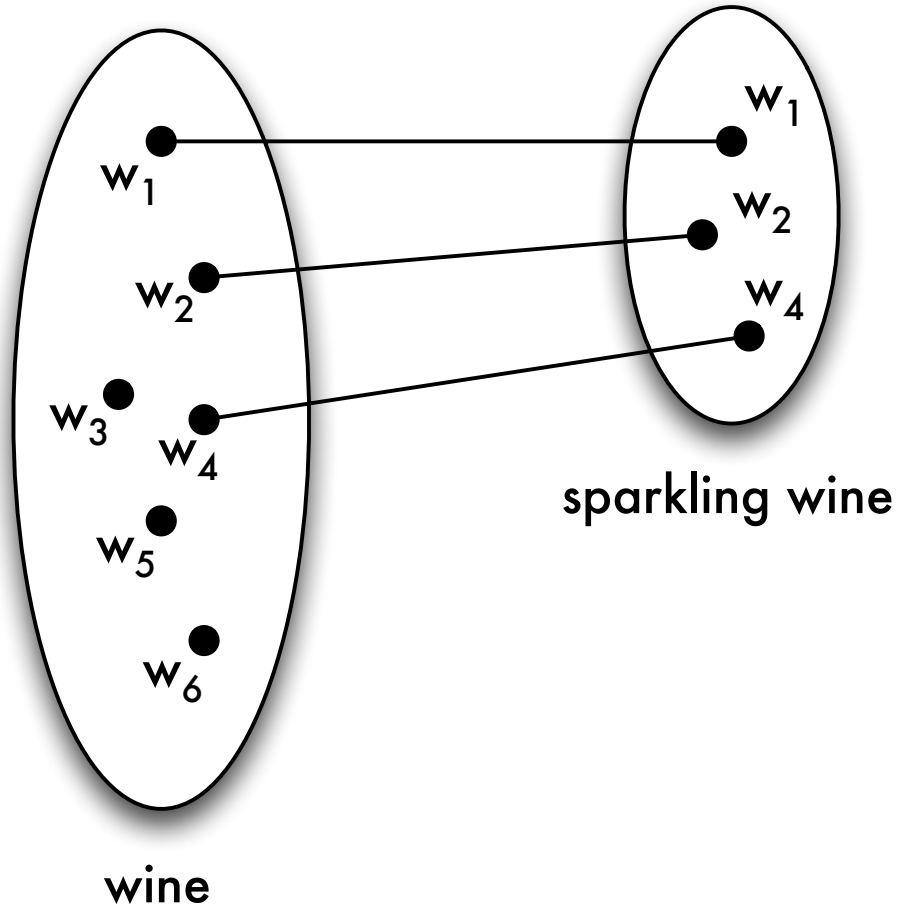



Each sparkling wine entity is associated with exactly one wine entity
$\rightsquigarrow$ sparkling wine entities are identified by the functional ISA relationship

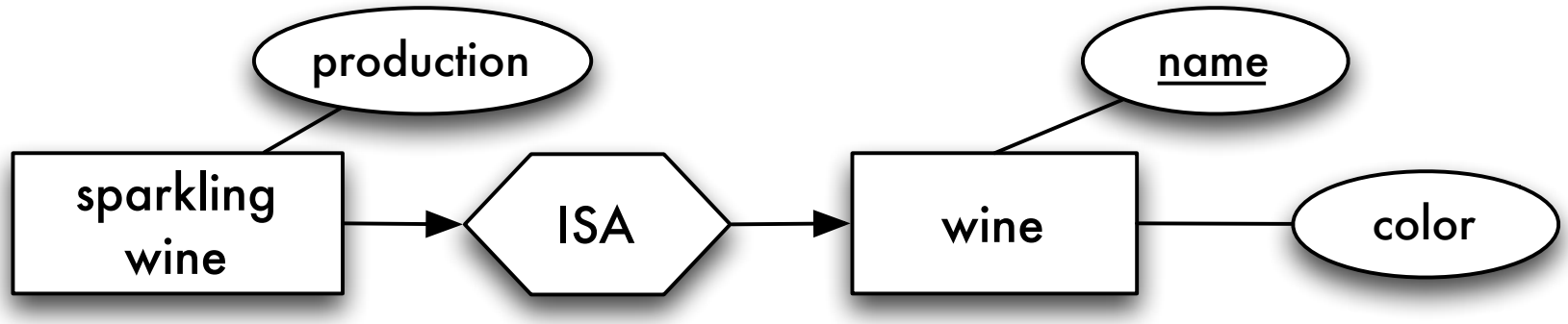
Not every wine is also a sparkling wine

Attributes of entity type wine are inherited by entity type sparkling wine

# Characteristics

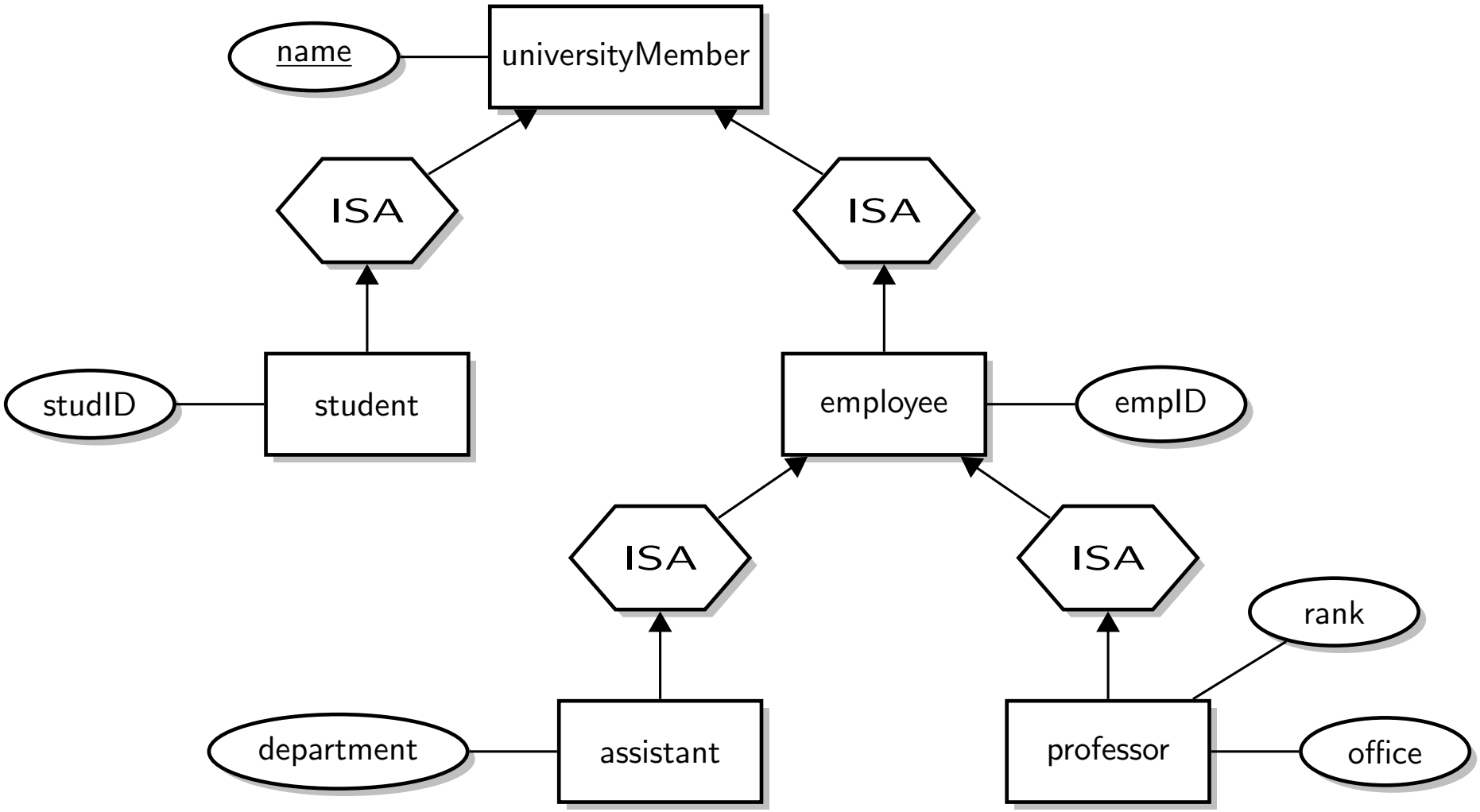


# Cardinality




- The cardinalities are always
$$\text{ISA}(E_1[1,1], E_2[0,1])$$
- Each entity of entity type $E_1$ (sparkling wine) participates exactly once, entities of entity type $E_2$ (wine) participate at most once.

# University example (overlapping specialization)



# Special characteristics

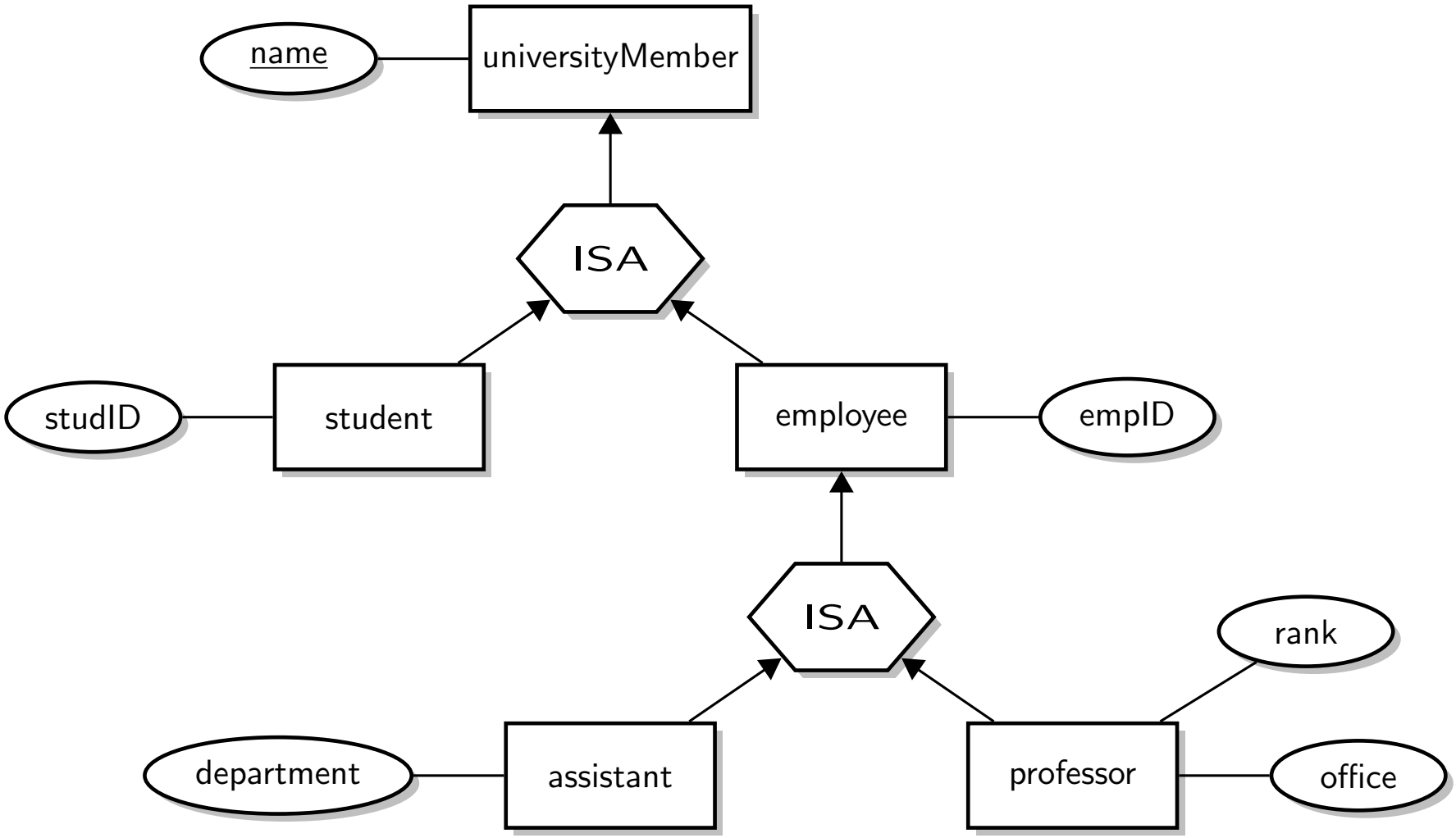
## Overlapping specialization

An entity may belong to multiple specialized entity sets.
→ separate ISA symbols are used

## Disjoint specialization

An entity may belong to at most one specialized entity set.
→ arrows to a shared ISA symbol in the diagram

# University example



# Attributes and relationship types

Lower-level entity types inherit

- attributes of the higher-level entity type
- participation in relationship types of the higher-level entity type

Lower-level entity types can

- have attributes
- participate in relationship types that the higher-level entity type does not participate in

# Participation constraints

## Total generalization/specialization

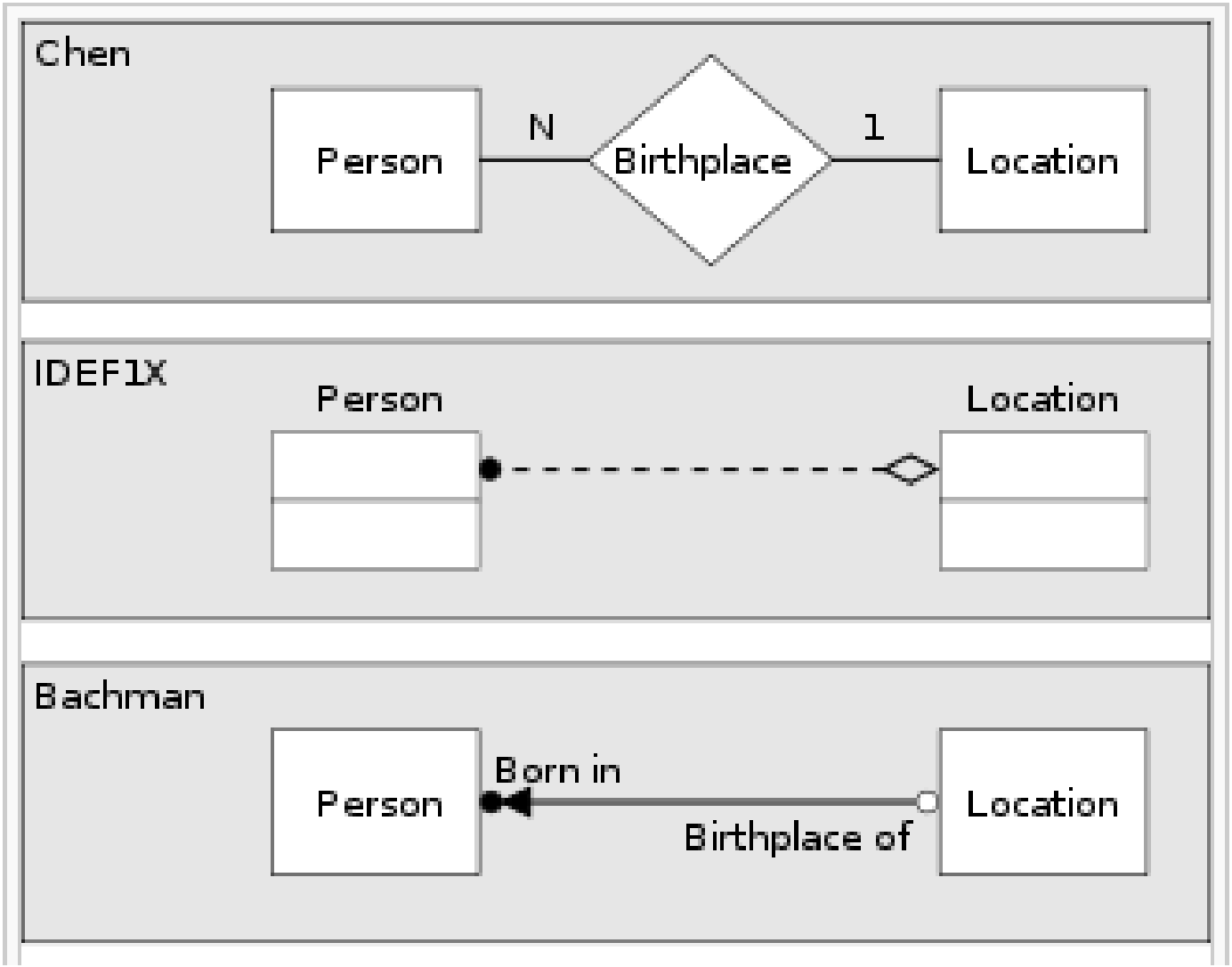
Each higher-level entity must belong to a lower-level entity type.
Notation: double line

## Partial generalization/specialization (default)

Each higher-level entity can (may or may not) belong to a lower-level entity type.

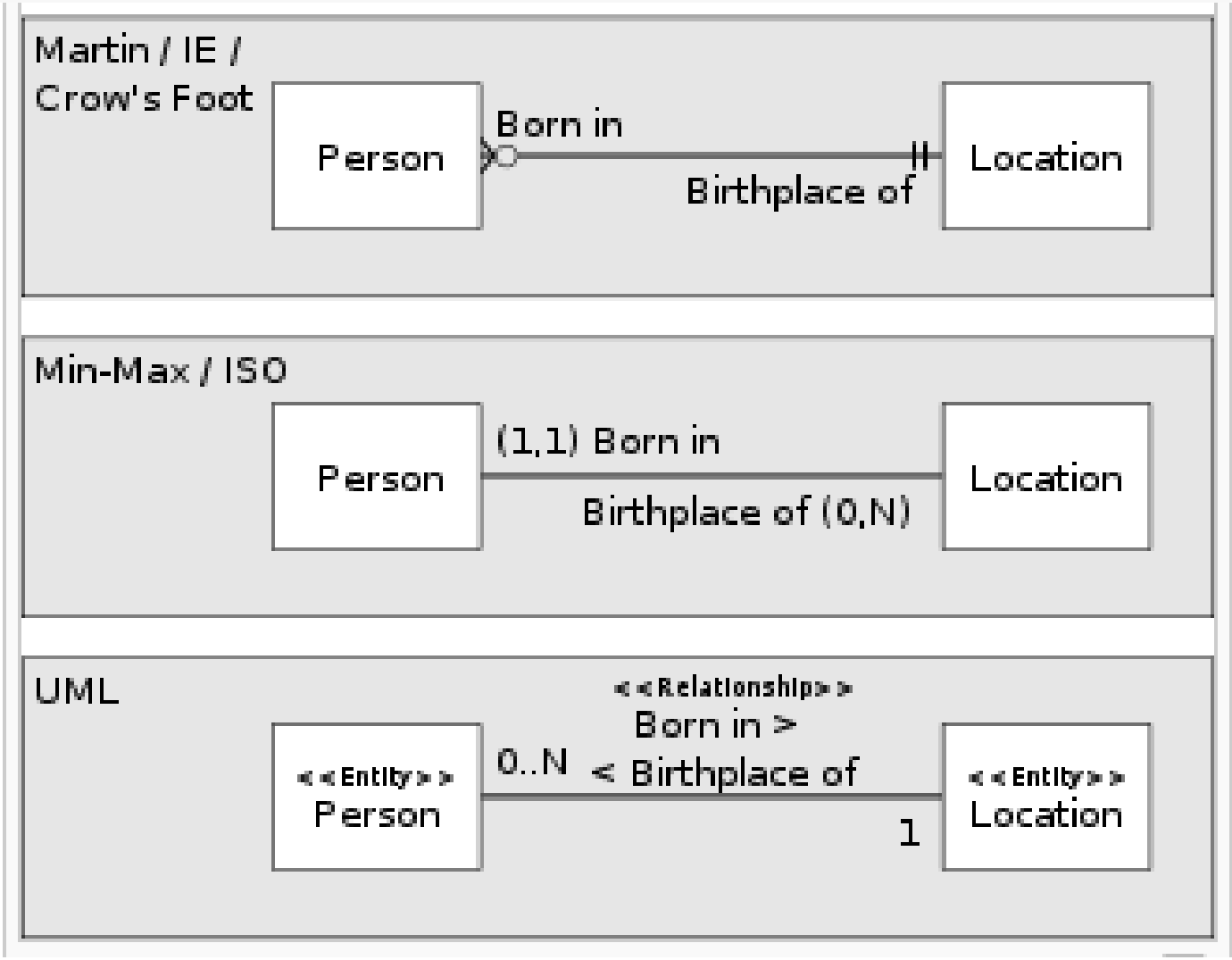
# Outline

# Alternative notations




http://en.wikipedia.org/wiki/Entity-relationship_model

# Alternative notations




http://en.wikipedia.org/wiki/Entity-relationship_model

# Summary

- Entity relationship diagrams (ERDs) describe the conceptual schema of a database
- There is also an extended ER model
- Basic ER concepts (Entity types, relationship types, attributes)
- Degree of relationship types
- Cardinalities (Chen, [min,max], total/partial participation)
- Weak entity types
- ISA relationship type

# Outline I

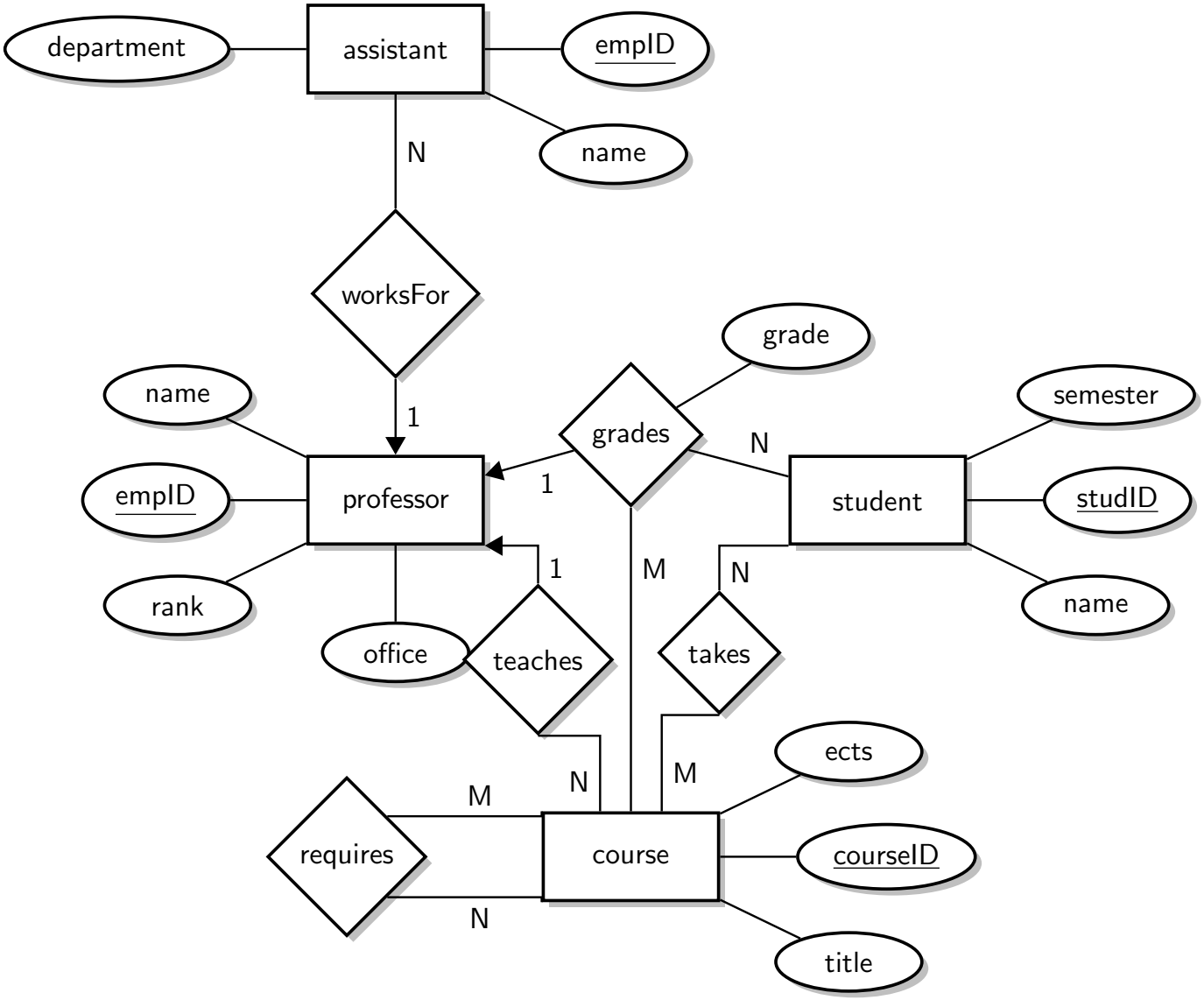
# Outline II

# Outline III

# Design notes

- Entities correspond to nouns, relationships to verbs.
- Each statement in the requirement specification should be reflected somewhere in the ER schema.
- Each ER diagram (ERD) should be located somewhere in the requirement specification.
- Conceptual design often reveals inconsistencies and ambiguities in the requirement specification, which must be first resolved.

# University schema with cardinality ratios




How to create relations representing all information of this ER diagram?

# Entity types

- **student**
  $\{[\ \underline{\text{studID: integer}},\ \text{name: string},\ \text{semester: integer}\ ]\}$
- **course**
  $\{[\ \underline{\text{courseID: integer}},\ \text{title: string},\ \text{ects: integer}\ ]\}$
- **professor**
  $\{[\ \underline{\text{empID: integer}},\ \text{name: string},\ \text{rank: string},\ \text{office: integer}\ ]\}$
- **assistant**
  $\{[\ \underline{\text{empID: integer}},\ \text{name: string},\ \text{department: string}\ ]\}$

Basic approach

- For each entity type $\rightarrow$ relation
- Name of the entity type $\rightarrow$ name of the relation
- Attributes of the entity type $\rightarrow$ Attributes of the relation
- Primary key of the entity type $\rightarrow$ Primary key of the relation

# Entity types

- **student**
  $\{[\ \underline{\text{studID: integer}},\ \text{name: string},\ \text{semester: integer}\ ]\}$
- **course**
  $\{[\ \underline{\text{courseID: integer}},\ \text{title: string},\ \text{ects: integer}\ ]\}$
- **professor**
  $\{[\ \underline{\text{empID: integer}},\ \text{name: string},\ \text{rank: string},\ \text{office: integer}\ ]\}$
- **assistant**
  $\{[\ \underline{\text{empID: integer}},\ \text{name: string},\ \text{department: string}\ ]\}$

Notation of relational schemas

student ($\underline{\text{studID: integer}}$, name: string, semester: integer)

student: $\{[\ \underline{\text{studID: integer}},\ \text{name: string},\ \text{semester: integer}\ ]\}$

We do not care about the order of attributes **in this context**!

# Entity types

- **student**
  $\{[\ \underline{\text{studID: integer}},\ \text{name: string},\ \text{semester: integer}\ ]\}$
- **course**
  $\{[\ \underline{\text{courseID: integer}},\ \text{title: string},\ \text{ects: integer}\ ]\}$
- **professor**
  $\{[\ \underline{\text{empID: integer}},\ \text{name: string},\ \text{rank: string},\ \text{office: integer}\ ]\}$
- **assistant**
  $\{[\ \underline{\text{empID: integer}},\ \text{name: string},\ \text{department: string}\ ]\}$

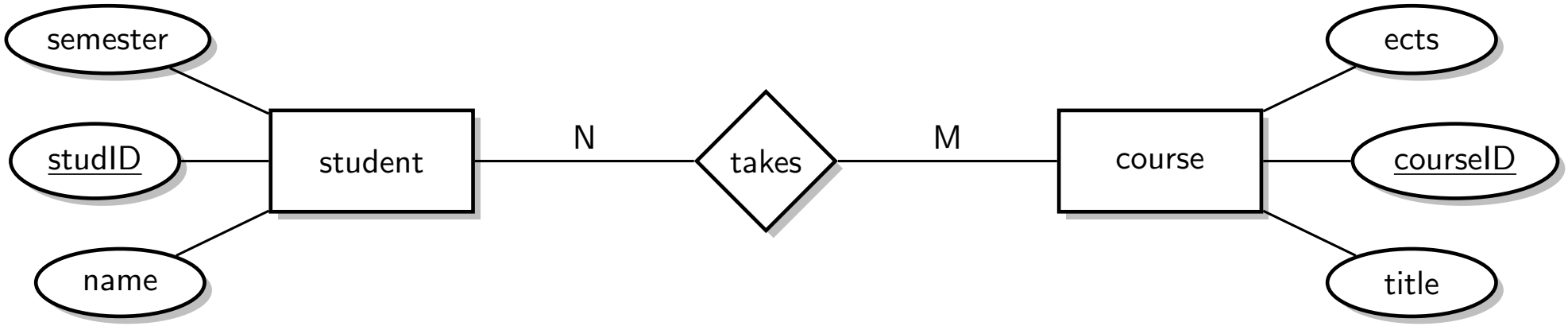
Notation of relational schemas

student ($\underline{\text{studID}}$, name, semester)

student: $\{[\ \underline{\text{studID}},\ \text{name, semester}\ ]\}$

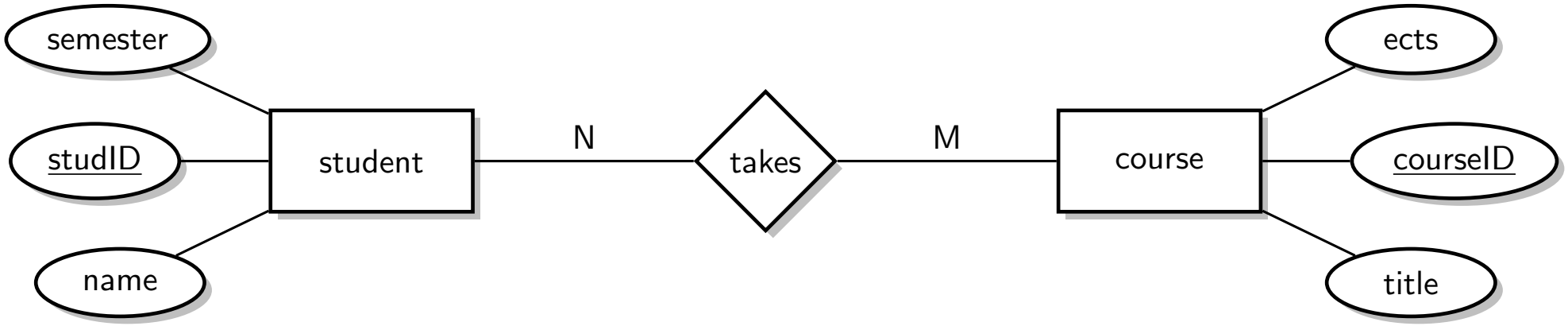
And for the moment we also do not care about attribute domains.

# Mapping of N:M relationship types




How to map this information to relations?

# Mapping of N:M relationship types

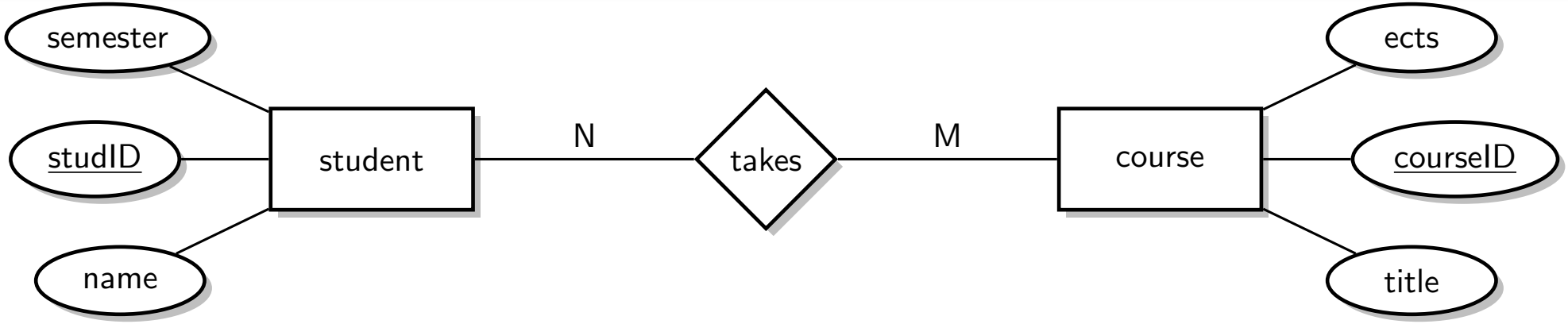



Basic approach

- New relation with all attributes of the relationship type
- Add the primary key attributes of all involved entity types
- Primary keys of involved entity types together become the key of the new relation

**takes:** $\{[\ \underline{\text{studID}} \rightarrow \text{student}, \underline{\text{courseID}} \rightarrow \text{course}\ ]\}$

# Mapping of N:M relationship types




Basic approach

- New relation with all attributes of the relationship type
- Add the primary key attributes of all involved entity types
- Primary keys of involved entity types together become the key of the new relation

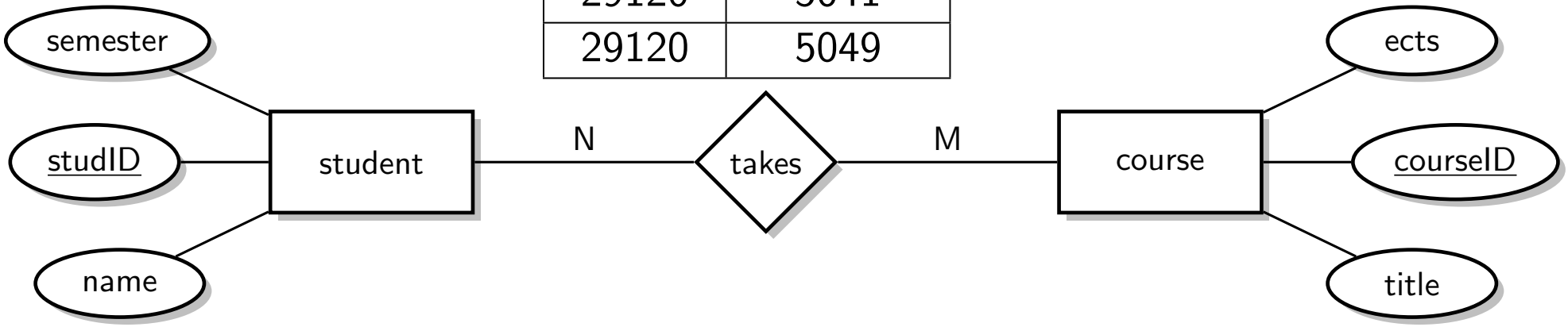
Key attributes "imported" from involved entity types (relations) are called **foreign keys**.

# Mapping of N:M relationship types

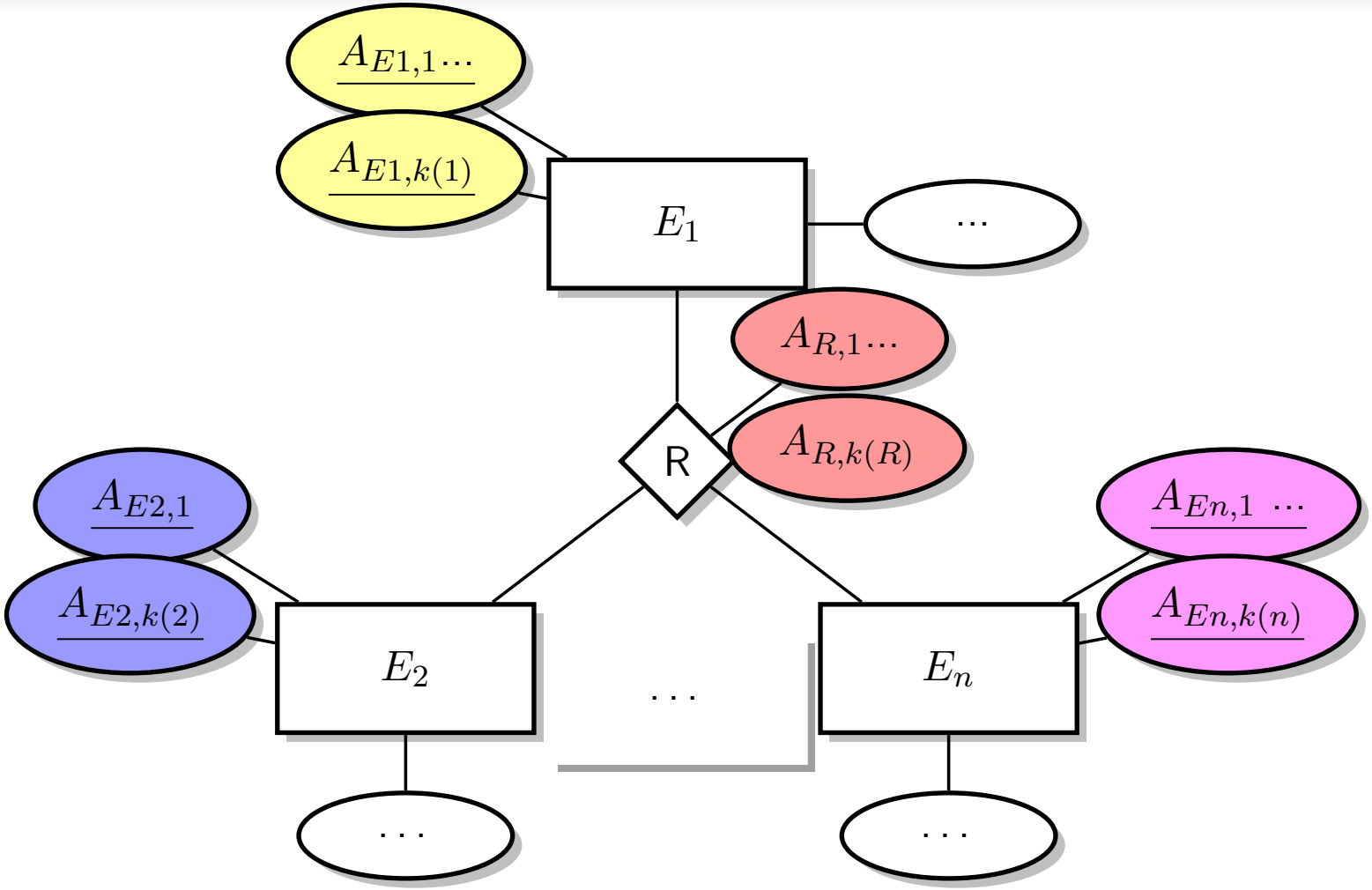
| student | |
|---|---|
| **studID** | **...** |
| 26120 | ... |
| 27550 | ... |
| ... | ... |

| takes | |
|---|---|
| **studID** | **courseID** |
| 26120 | 5001 |
| 27550 | 5001 |
| 27550 | 4052 |
| 28106 | 5041 |
| 28106 | 5052 |
| 28106 | 5216 |
| 28106 | 5259 |
| 29120 | 5001 |
| 29120 | 5041 |
| 29120 | 5049 |

| course | |
|---|---|
| **courseID** | **...** |
| 5001 | ... |
| 4052 | ... |
| ... | ... |



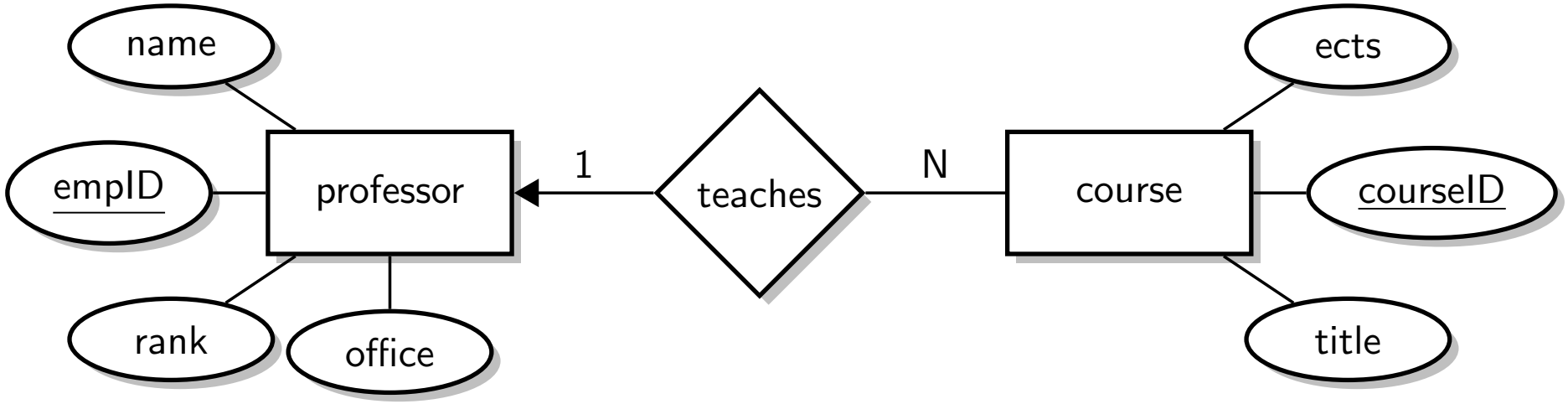
# Mapping of N:M relationship types in general




R: {[ $\underline{A_{E1,1}, \ldots, A_{E1,k(1)}}$ , $\underline{A_{E2,1}, \ldots, A_{E2,k(2)}}$ , ..., $\underline{A_{En,1}, \ldots, A_{En,k(n)}}$ , $A_{R,1}, \ldots, A_{R2,k(R)}$ ]}

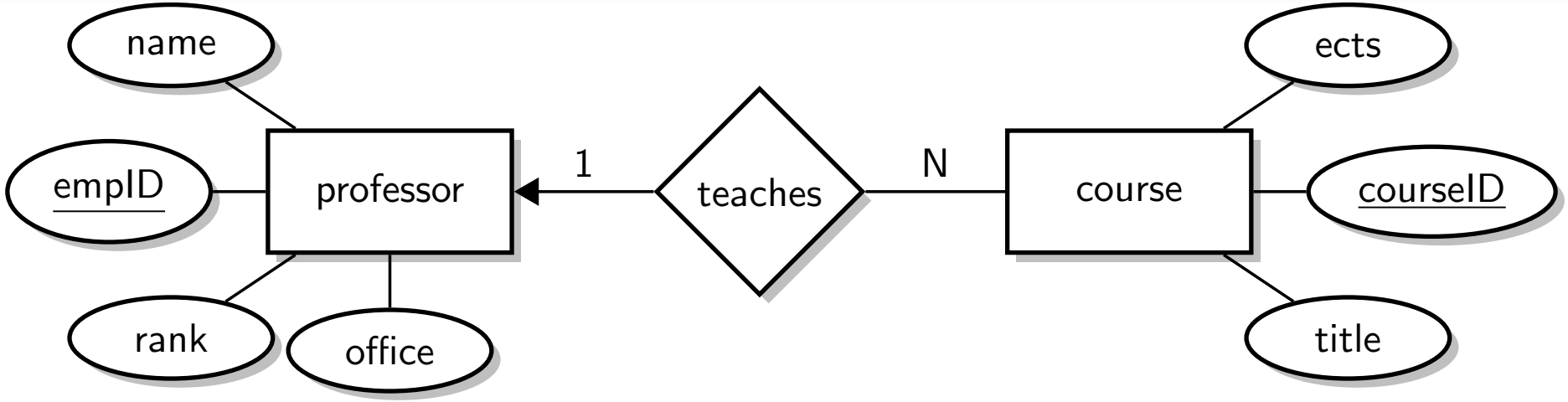
key of $E_1$ | key of $E_2$ | key of $E_n$ | attributes of $R$

# Mapping of 1:N relationship types




Is this different from N:M relationship types?

# 1:N relationship types

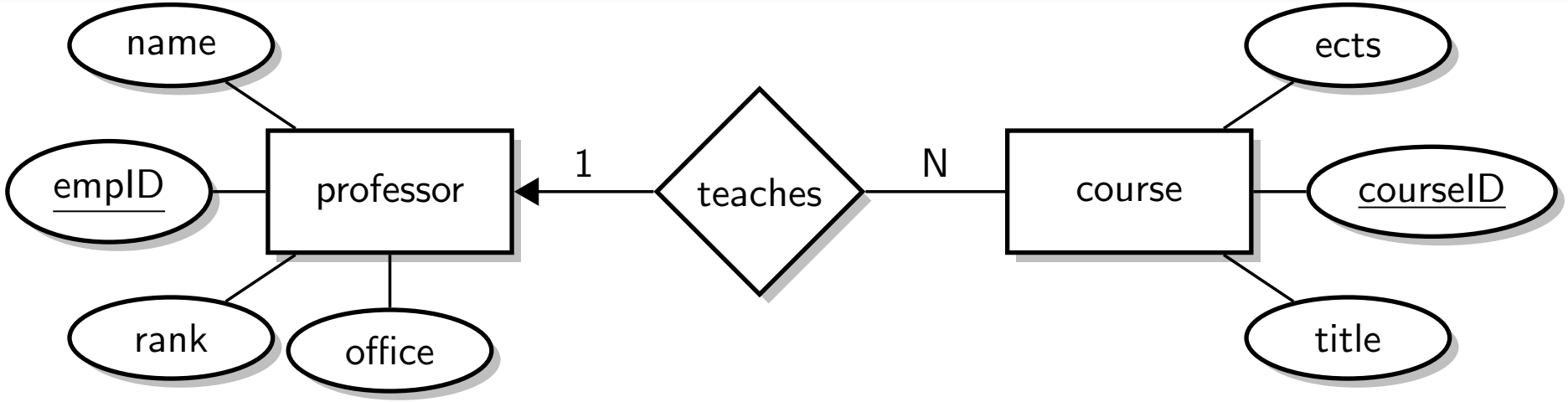



- New relation with all attributes of the relationship type
- Add primary key attributes of all involved entity types

**Initially!**

- **course:** {[ courseID, title, ects ]}
- **professor:** {[ empID, name, rank, office ]}
- **teaches:** {[ courseID → course, empID → professor ]}

# 1:N relationship types

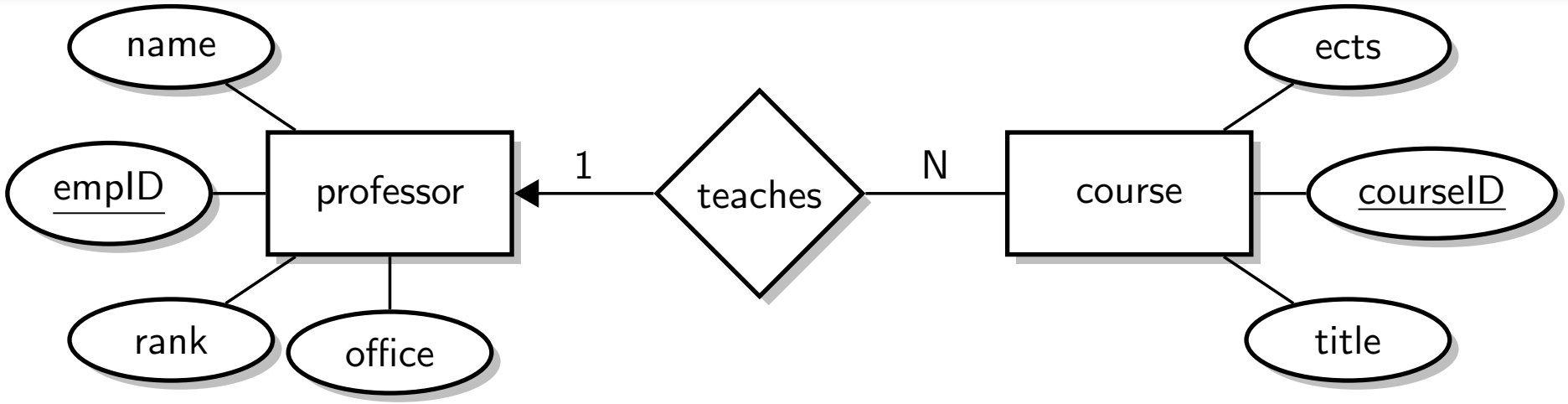



- New relation with all attributes of the relationship type
- Add primary key attributes of all involved entity types
- Primary key ...

**Initially!**

- **course:** {[ courseID, title, ects ]}
- **professor:** {[ empID, name, rank, office ]}
- **teaches:** {[ courseID → course, empID → professor ]}

# 1:N relationship types




- New relation with all attributes of the relationship type
- Add primary key attributes of all involved entity types
- Primary key of the "N"-side becomes the key in the new relation

**Initially!**

- **course:** {[ courseID, title, ects ]}
- **professor:** {[ empID, name, rank, office ]}
- **teaches:** {[ courseID → course, empID → professor ]}

# Mapping of 1:N relationship types

Initially

- **course:** {[ courseID, title, ects ]}
- **professor:** {[ empID, name, rank, office ]}
- **teaches:** {[ courseID → course, empID → professor}

# Mapping of 1:N relationship types

Initially

- **course:** {[ courseID, title, ects ]}
- **professor:** {[ empID, name, rank, office ]}
- **teaches:** {[ courseID → course, empID → professor}

Improvement by merging

- **course:** {[ courseID, title, ects, taughtBy → professor ]}
- **professor:** {[ empID, name, rank, office ]}

**taughtBy** is a **foreign key** and references the primary key of relation professor.
Values of **taughtBy** correspond to values of *empID* in relation professor.

# Mapping of 1:N relationship types

Initially

- **course:** {[ courseID, title, ects ]}
- **professor:** {[ empID, name, rank, office ]}
- **teaches:** {[ courseID → course, empID → professor}

Improvement by merging

- **course:** {[ courseID, title, ects, taughtBy → professor ]}
- **professor:** {[ empID, name, rank, office ]}

**taughtBy** is a **foreign key** and references the primary key of relation professor.
Values of **taughtBy** correspond to values of *empID* in relation professor.

Relations with the same key can be combined...
**but only these and no others!**

# Mapping of 1:N relationship types

Initially

- **course:** {[ courseID, title, ects ]}
- **professor:** {[ empID, name, rank, office ]}
- **teaches:** {[ courseID → course, empID → professor}

Improvement by merging

- **course:** {[ courseID, title, ects, taughtBy → professor ]}
- **professor:** {[ empID, name, rank, office ]}

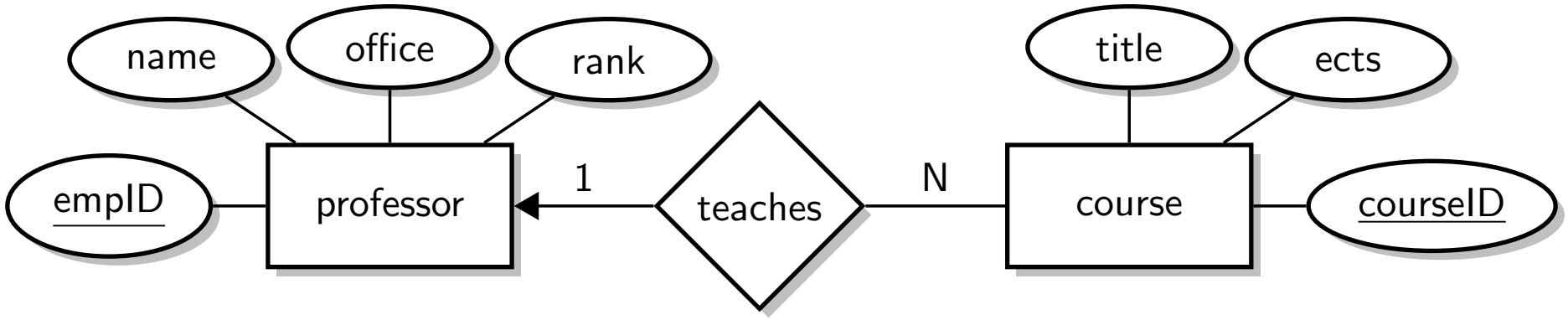
**taughtBy** is a **foreign key** and references the primary key of relation professor.
Values of **taughtBy** correspond to values of *empID* in relation professor.

If the **participation** is **not total**, merging requires null values for the foreign key. In such cases, it might be preferable for some applications to have a separate relation.

# Professor and course

| professor | | | |
|---|---|---|---|
| **empID** | **name** | **rank** | **office** |
| 2125 | Socrates | C4 | 226 |
| 2126 | Russel | C4 | 232 |
| 2127 | Kopernikus | C3 | 310 |
| 2133 | Popper | C3 | 52 |
| 2134 | Augustinus | C3 | 309 |
| 2136 | Curie | C4 | 36 |
| 2137 | Kant | C4 | 7 |

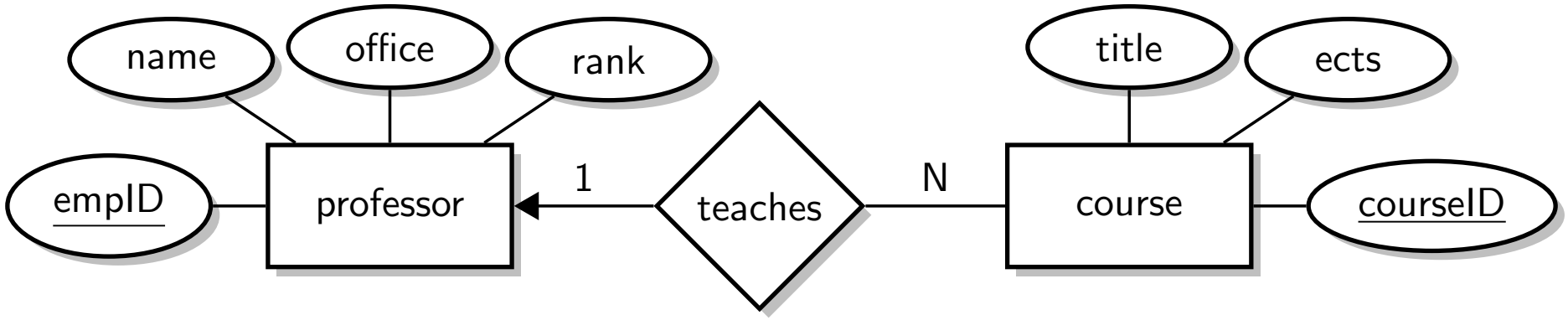
| course | | | |
|---|---|---|---|
| **courseID** | **title** | **ects** | **taughtBy** |
| 5001 | DBS | 4 | 2137 |
| 5041 | Robotics | 4 | 2125 |
| 5043 | Software Engineering | 3 | 2126 |
| 5049 | Ethics | 2 | 2125 |
| 4052 | Logic | 4 | 2125 |
| 5052 | Theory of Science | 3 | 2126 |
| 5216 | Bioethics | 2 | 2126 |
| 5259 | Chemistry | 2 | 2133 |
| 5022 | Belief and Knowledge | 2 | 2134 |
| 4630 | Physics | 4 | 2137 |



# Attention: this does **not** work

| professor | | | | |
|---|---|---|---|---|
| **empID** | **name** | **rank** | **office** | **teaches** |
| 2125 | Socrates | C4 | 226 | 5041 |
| **2125** | **Socrates** | **C4** | **226** | **5049** |
| **2125** | **Socrates** | **C4** | **226** | **4052** |
| **...** | **...** | **...** | **...** | **...** |
| 2134 | Augustinus | C3 | 309 | **5022** |
| 2136 | Curie | C4 | 36 | **??** |
| ... | ... | ... | ... | ... |

| course | | |
|---|---|---|
| **courseID** | **title** | **ects** |
| 5001 | DBS | 4 |
| 5041 | Robotics | 4 |
| 5043 | Software Engineering | 3 |
| 5049 | Ethics | 2 |
| 4052 | Logic | 4 |
| 5052 | Theory of Science | 3 |
| 5216 | Bioethics | 2 |
| 5259 | Chemistry | 2 |
| 5022 | Belief and Knowledge | 2 |
| 4630 | Physics | 4 |



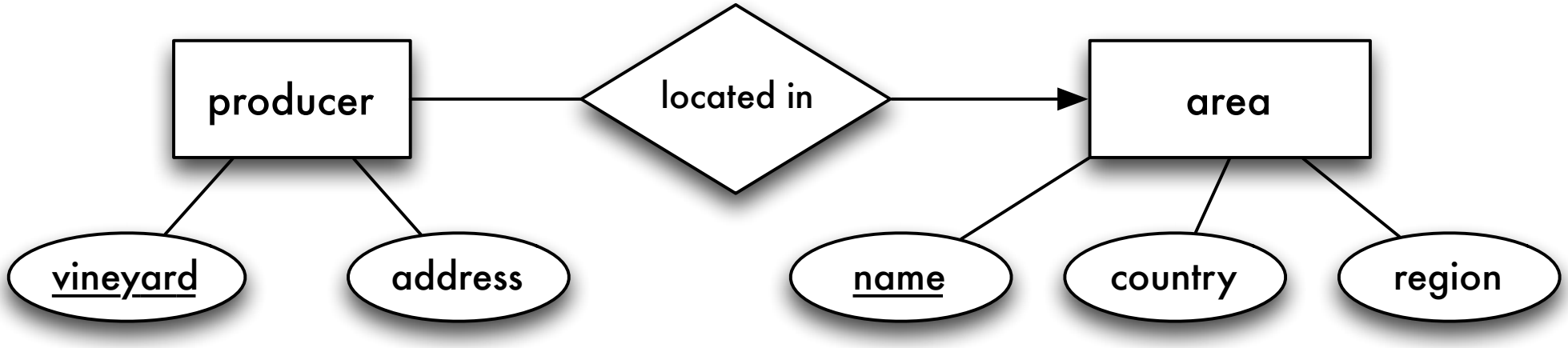
# Why can/will there be problems?

| professor | | | | |
|---|---|---|---|---|
| **empID** | **name** | **rank** | **office** | **teaches** |
| 2125 | Socrates | C4 | 226 | 5041 |
| **2125** | **Socrates** | **C4** | **226** | **5049** |
| **2125** | **Socrates** | **C4** | **226** | **4052** |
| **...** | **...** | **...** | **...** | **...** |
| 2134 | Augustinus | C3 | 309 | **5022** |
| 2136 | Curie | C4 | 36 | **??** |
| ... | ... | ... | ... | ... |

| course | | |
|---|---|---|
| **courseID** | **title** | **ects** |
| 5001 | DBS | 4 |
| 5041 | Robotics | 4 |
| 5043 | Software Engineering | 3 |
| 5049 | Ethics | 2 |
| 4052 | Logic | 4 |
| 5052 | Theory of Science | 3 |
| 5216 | Bioethics | 2 |
| 5259 | Chemistry | 2 |
| 5022 | Belief and Knowledge | 2 |
| 4630 | Physics | 4 |

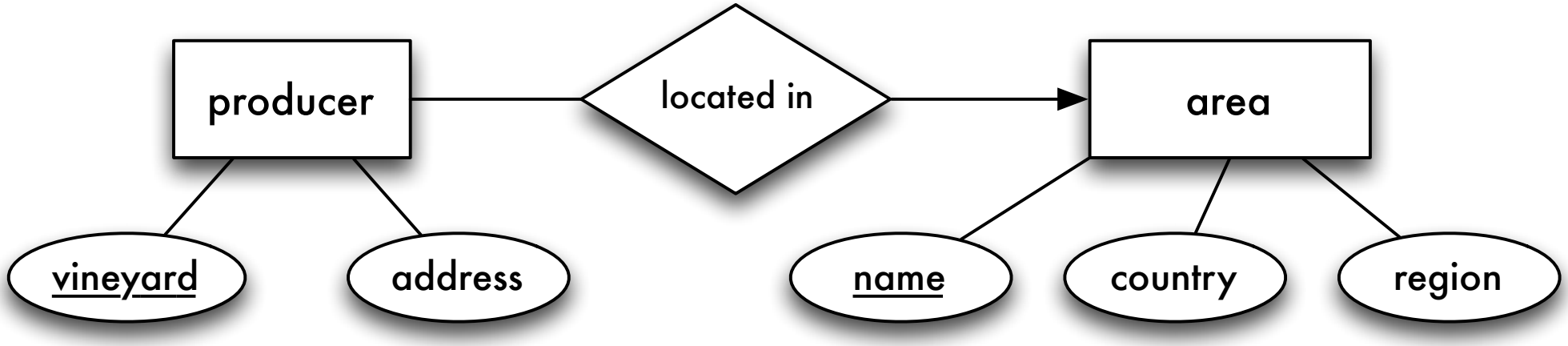
- **Update anomaly:** What happens when Socrates moves?
- **Deletion anomaly:** What happens if "Belief and Knowledge" is no longer taught?
- **Insert anomaly:** Curie is new and does not yet teach any lectures

# Summary: N:1 relationship types




How to map this ERD to relations?

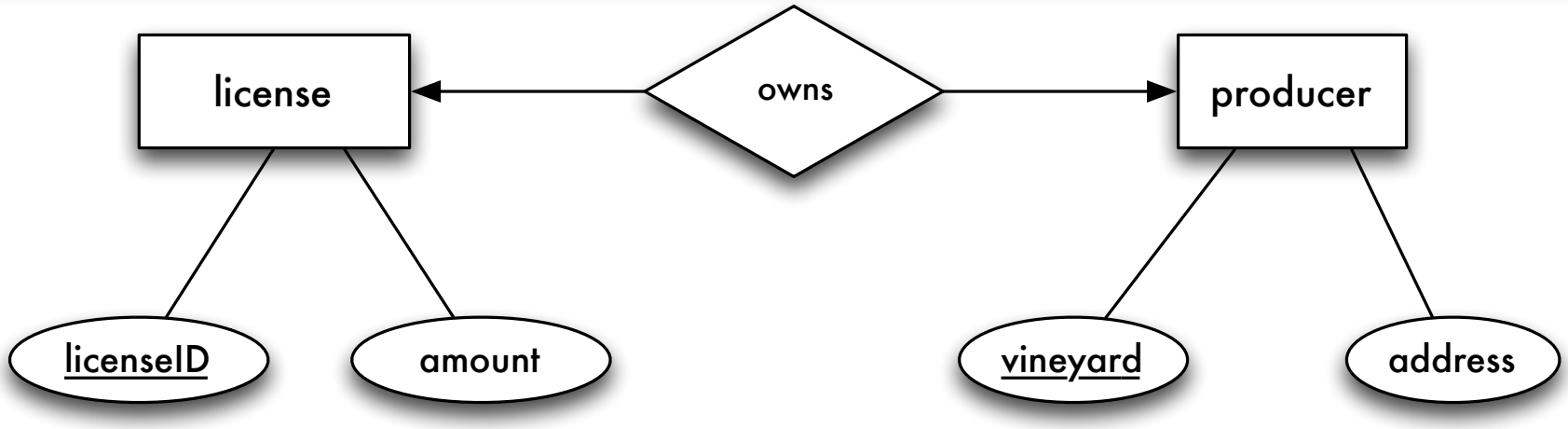
# Summary: N:1 relationship types




How to map this ERD to relations?

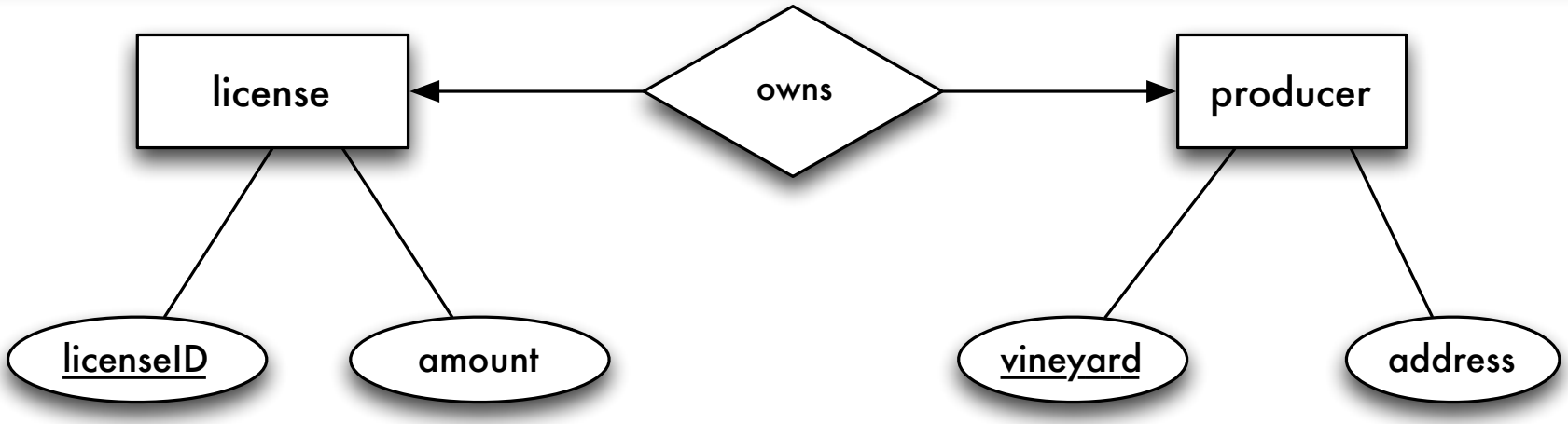
- **producer:** {[ vineyard, address, locatedIn → area ]}
- **area:** {[ name, country, region ]}

# 1:1 relationship types




Is this different from 1:N relationship types?

# 1:1 relationship types




- New relation with all attributes of the relationship type
- Add primary key attributes of all involved entity types
- Primary key of any of the involved entity types can become the key in the new relation

**Initially!**

- **license:** {[ licenseID, amount ]}
- **producer:** {[ vineyard, address ]}
- **owns:** {[ licenseID → license, vineyard → producer ]} or
  **owns:** {[ licenseID → license, vineyard → producer ]}

# 1:1 relationship types

Initially

- **license:** {[ licenseID, amount ]}
- **producer:** {[ vineyard, address ]}
- **owns:** {[ licenseID → license, vineyard → producer ]} or
  **owns:** {[ licenseID → license, vineyard → producer ]}

Improvement by merging

- **license:** {[ licenseID, amount, ownedBy → producer ]}
- **producer:** {[ vineyard, address ]}

or

- **license:** {[ licenseID, amount ]}
- **producer:** {[ vineyard, address, ownsLicense → license ]}

# 1:1 relationship types

Initially

- **license:** {[ licenseID, amount ]}
- **producer:** {[ vineyard, address ]}
- **owns:** {[ licenseID → license, vineyard → producer ]} or
  **owns:** {[ licenseID → license, vineyard → producer ]}

Improvement by merging

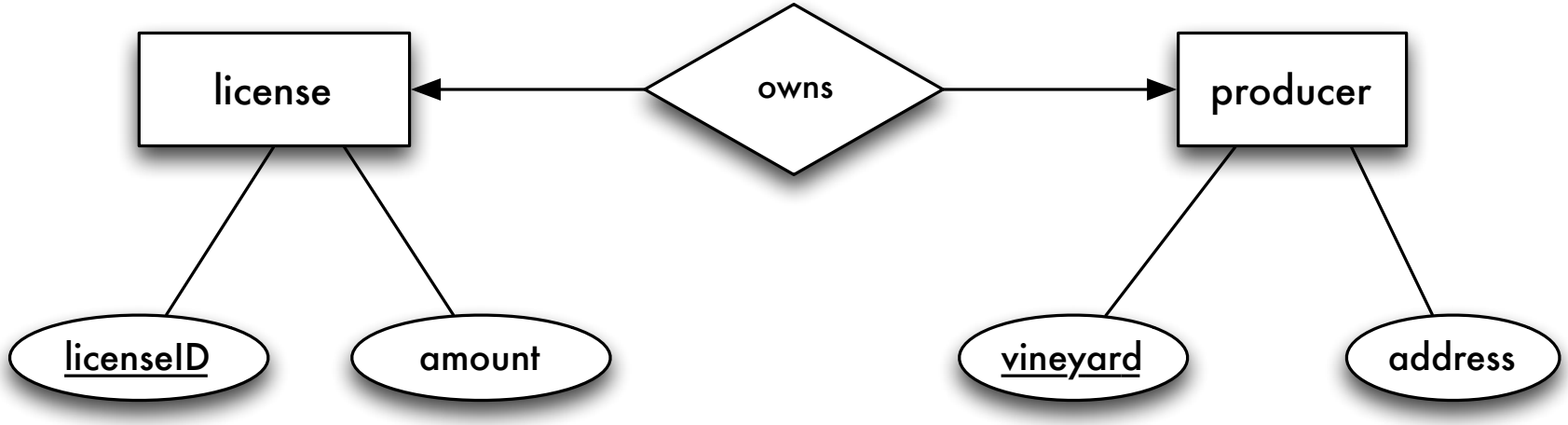
- **license:** {[ licenseID, amount, ownedBy → producer ]}
- **producer:** {[ vineyard, address ]}

or

- **license:** {[ licenseID, amount ]}
- **producer:** {[ vineyard, address, ownsLicense → license ]}

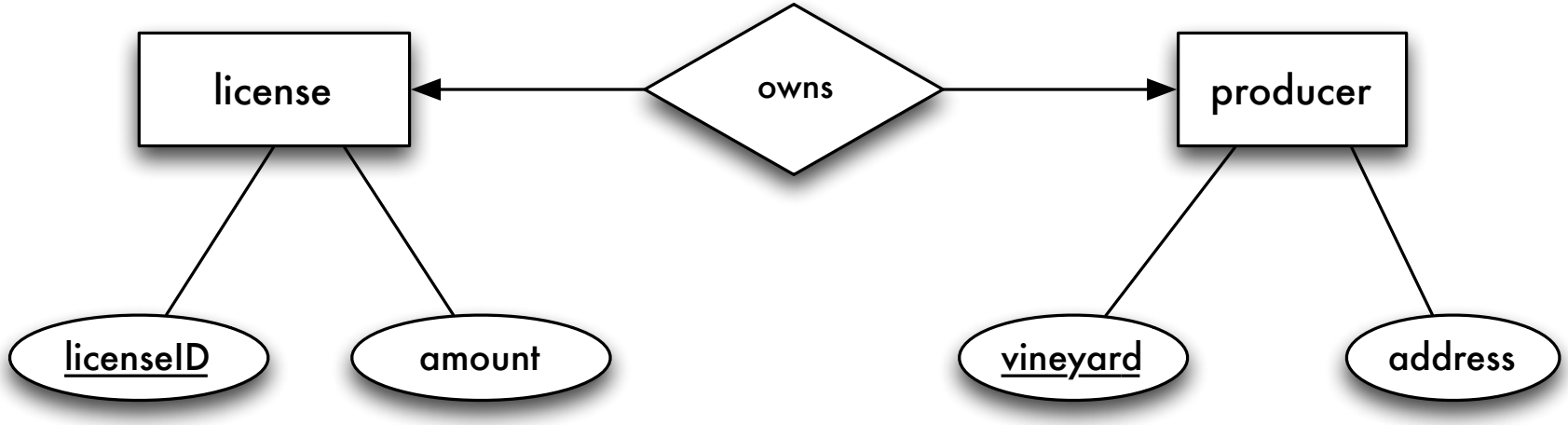
It is best to extend a relation of an entity type with **total participation**.

# Why not a single relation?




| producer | vineyard | address | licenseID | amount |
|---|---|---|---|---|
| | Rotkäppchen | Freiberg | 42-007 | 10.000 |
| | Weingut Müller | Dagstuhl | 42-009 | 250 |

# Why not a single relation?

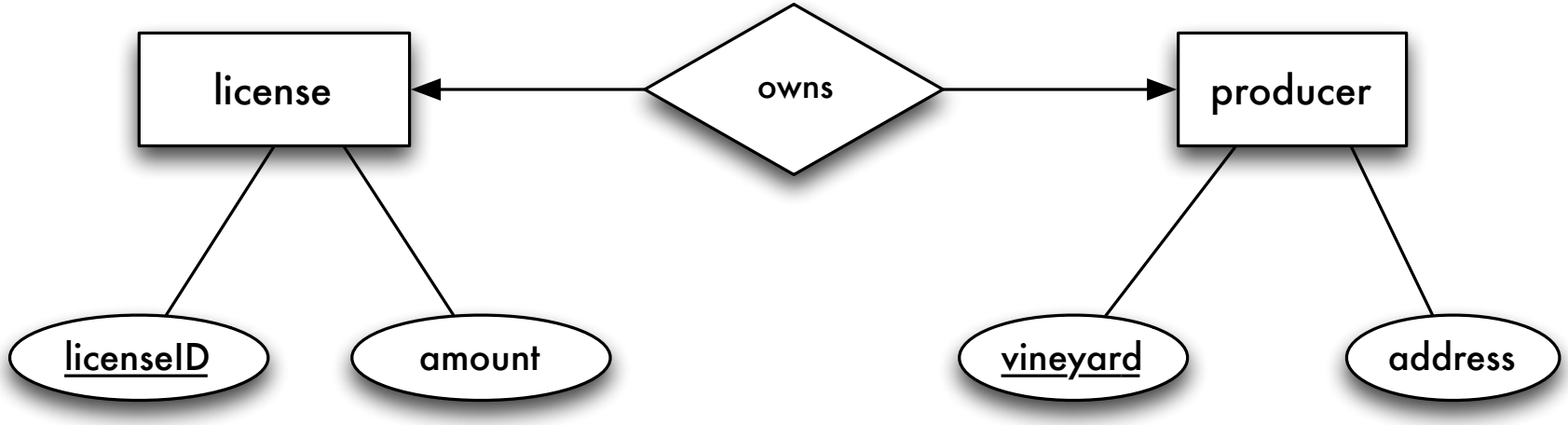



producer

| vineyard | address | licenseID | amount |
|---|---|---|---|
| Rotkäppchen | Freiberg | 42-007 | 10.000 |
| Weingut Müller | Dagstuhl | 42-009 | 250 |

Only correct in case of **total participation** ([1,1]) of both involved entity types

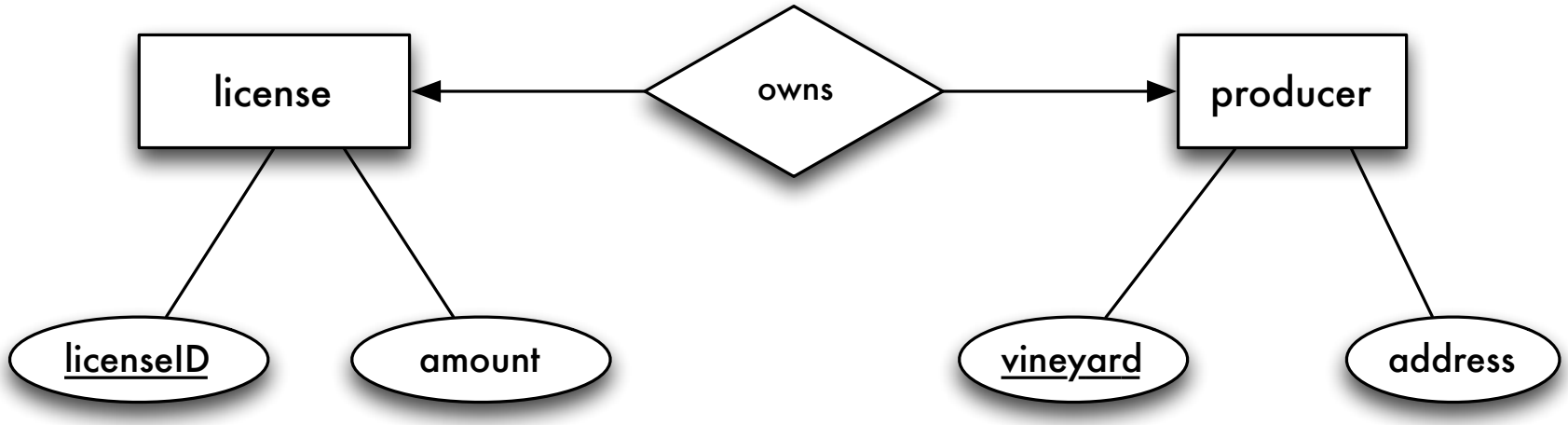
# Why not a single relation?




Producers without licenses require null values

| producer | vineyard | address | licenseID | amount |
|---|---|---|---|---|
| | Rotkäppchen | Freiberg | 42-007 | 10.000 |
| | Weingut Müller | Dagstuhl | ⊥ | ⊥ |

# Why not a single relation?



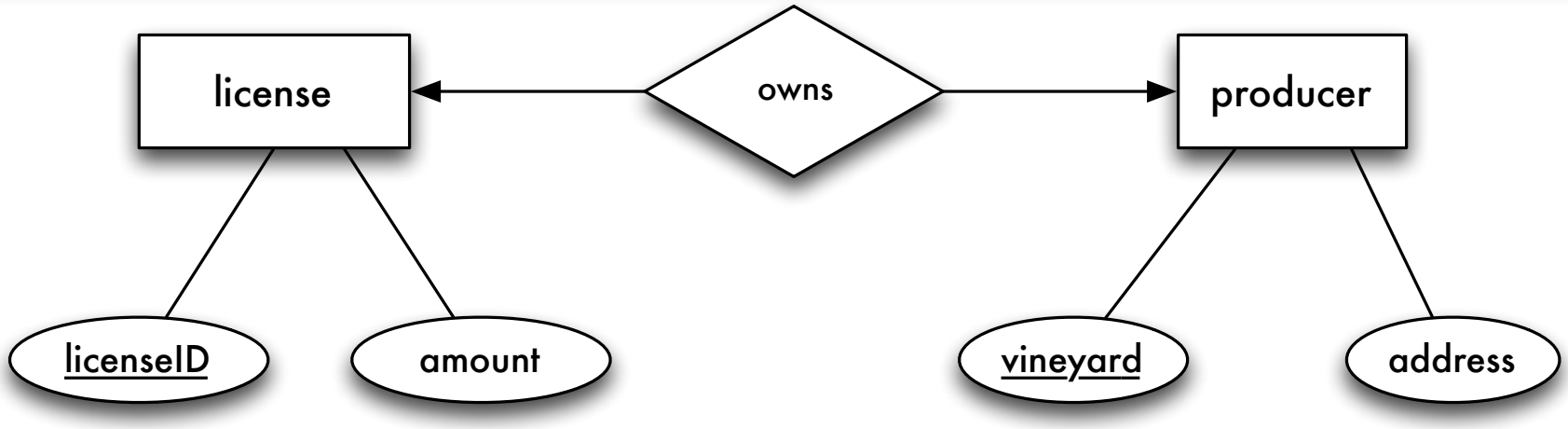

Producers without licenses require null values

producer

| vineyard | address | licenseID | amount |
|---|---|---|---|
| Rotkäppchen | Freiberg | 42-007 | 10.000 |
| Weingut Müller | Dagstuhl | ⊥ | ⊥ |

Possible in case of **partial participation** ([0,1]) of one involved entity type, **total participation** of the other one
→ leads to null values

# Why not a single relation?

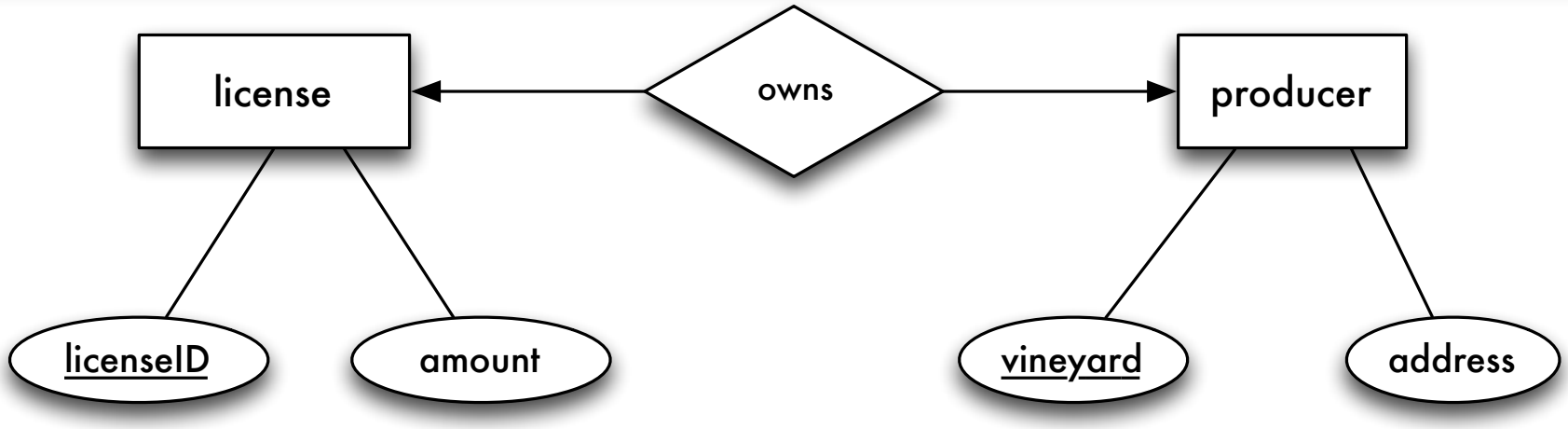



Free licenses lead to more null values

`producer`

| `vineyard` | `address` | `licenseID` | `amount` |
|---|---|---|---|
| Rotkäppchen | Freiberg | 42-007 | 10.000 |
| Weingut Müller | Dagstuhl | $\perp$ | $\perp$ |
| $\perp$ | $\perp$ | 42-003 | 100.000 |

# Why not a single relation?



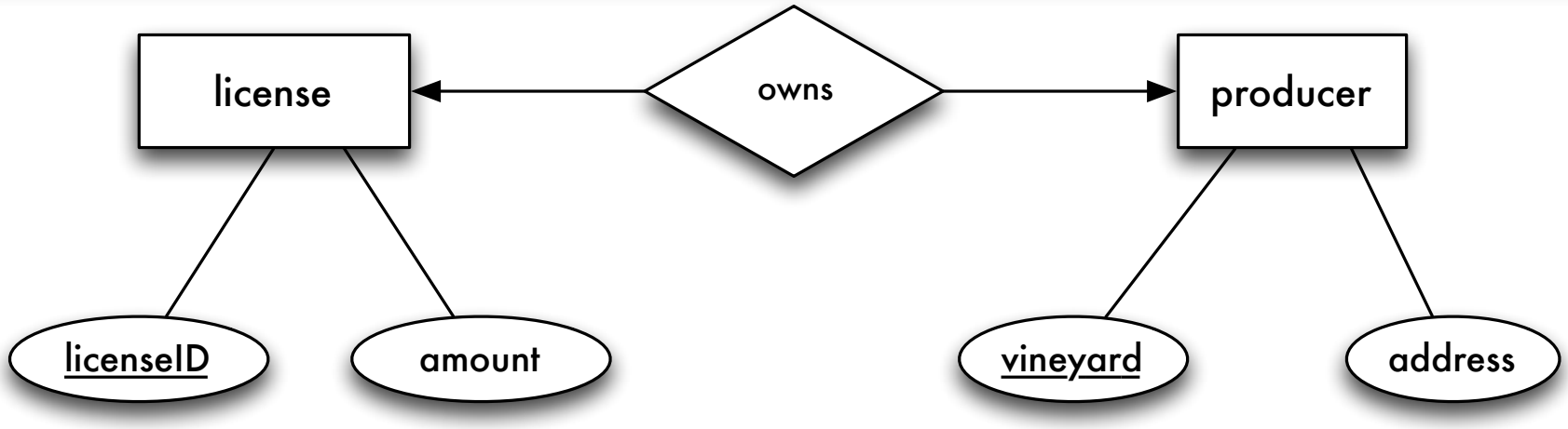

Free licenses lead to more null values

`producer`

| `vineyard` | `address` | `licenseID` | `amount` |
|---|---|---|---|
| Rotkäppchen | Freiberg | 42-007 | 10.000 |
| Weingut Müller | Dagstuhl | ⊥ | ⊥ |
| ⊥ | ⊥ | 42-003 | 100.000 |

**Partial participation** of both involved entity types
→ leads to null values in all attributes
→ difficult to determine a primary key, waste of memory

# Why not a single relation?




Free licenses lead to more null values

`producer`

| `vineyard` | `address` | `licenseID` | `amount` |
|---|---|---|---|
| Rotkäppchen | Freiberg | 42-007 | 10.000 |
| Weingut Müller | Dagstuhl | $\perp$ | $\perp$ |
| $\perp$ | $\perp$ | 42-003 | 100.000 |

- In general: no merging into a single relation!
- Standard approach: 2 relations (some null values are tolerable in most applications)

# Summary: mapping relationship types to relations

## M:N relationship type

- New relation with relationship type's attributes
- Add attributes referencing the primary keys of the involved entity type relations
- Primary key: set of foreign keys

## 1:N relationship type

- Add information to the entity type relation of the "N"-side:
  - Add foreign key referencing the primary key of the "1"-side entity type relation
  - Add attributes of the relationship type

## 1:1 relationship type

- Add information to one of the involved entity type relations:
  - Add foreign key referencing the primary key of the other entity type relation
  - Add attributes of the relationship type

# Foreign keys

A **foreign key** is an attribute (or a combination of attributes) of a relation that references the primary key (or candidate key) of another relation.

Example

- **course:** {[ courseID, title, ects, **taughtBy** ]}
- **professor:** {[ empID, name, rank, office ]}

taughtBy is a foreign key referencing relation professor

Notation

- **course:** {[ courseID, title, ects, **taughtBy** $\rightarrow$ **professor** ]}
- **professor:** {[ empID, name, rank, office ]}

# Foreign keys

A **foreign key** is an attribute (or a combination of attributes) of a relation that references the primary key (or candidate key) of another relation.

Notation

- **course:** {[ courseID, title, ects, **taughtBy** $\rightarrow$ **professor** ]}
- **professor:** {[ empID, name, rank, office ]}

Alternative notation

- **course:** {[ courseID, title, ects, **taughtBy** ]}
- **professor:** {[ empID, name, rank, office ]}

**Foreign key: course.taughtBy $\rightarrow$ professor.empID**

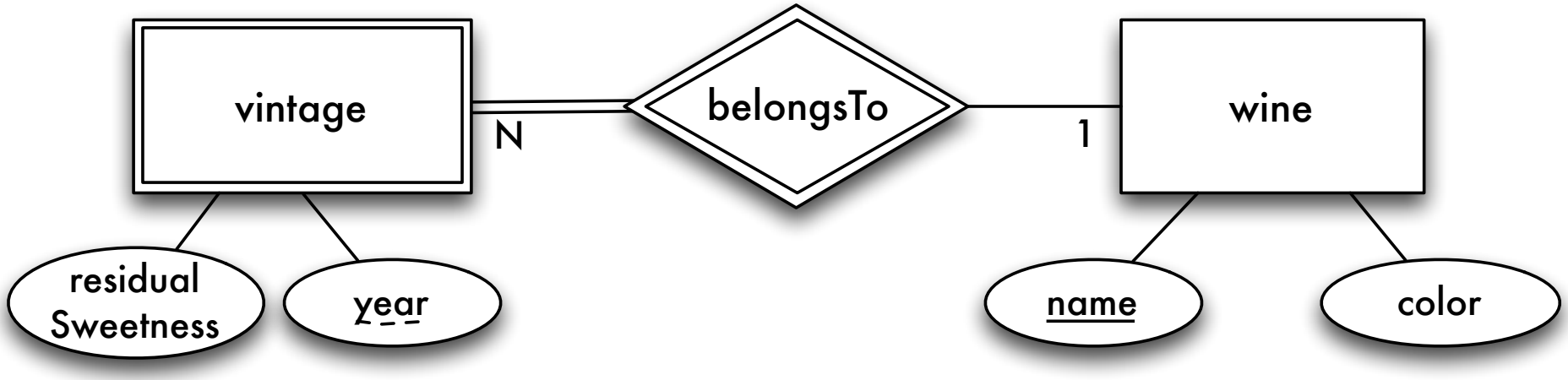
Notation for composite keys: $\{ R.A_1, R.A_2 \} \rightarrow \{ S.B_1, S.B_2 \}$

# Outline I

# Outline II

# Outline III

# Weak entity types

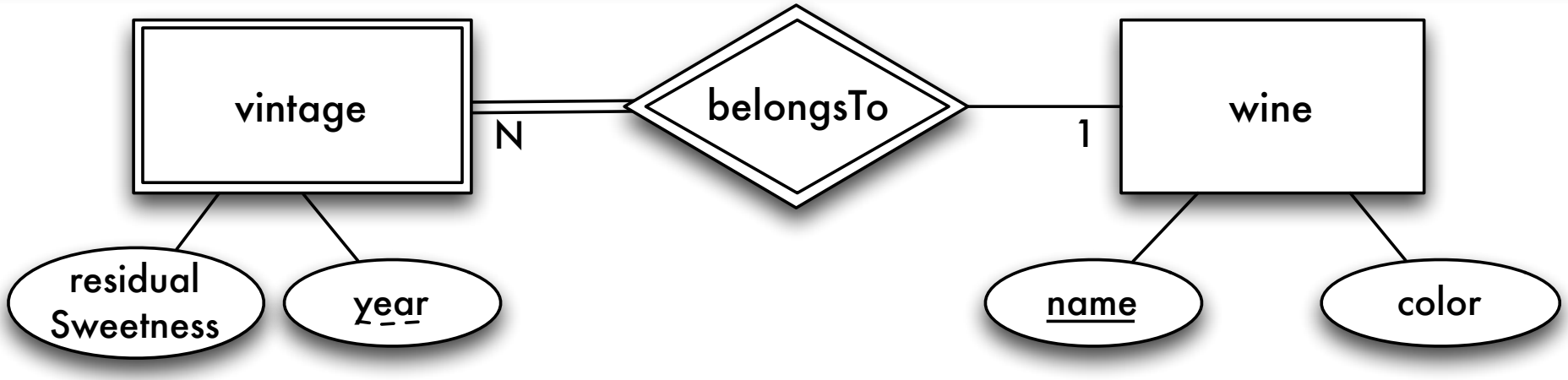



Entities of a weak entity type are

- existentially dependent on a strong entity type
- uniquely identifiable in combination with the strong entity type's key

How to map weak entity types to relations?

# Weak entity types



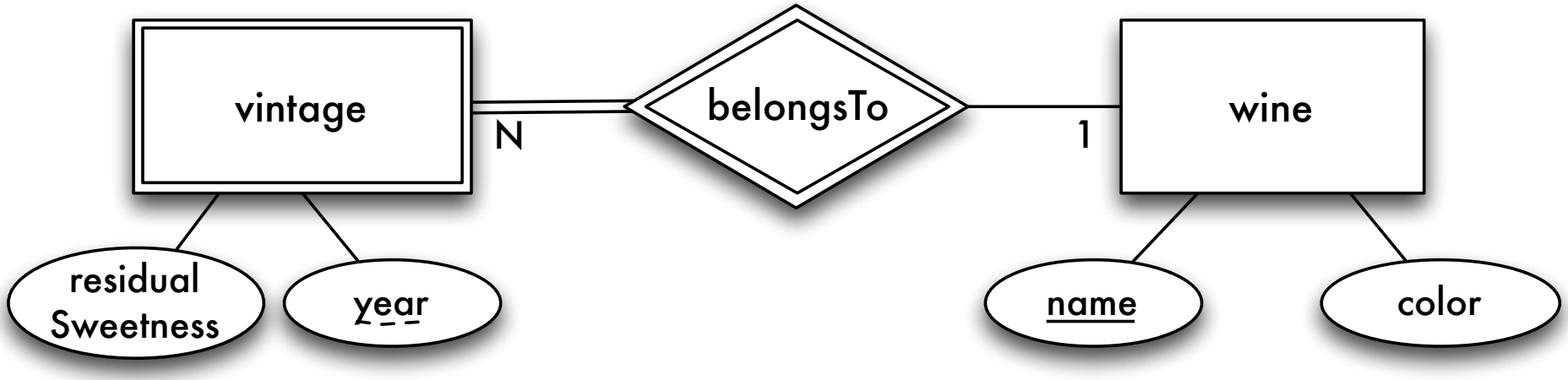

Mapping the identifying relationship type

- New relation with all attributes of the relationship type
- Add primary key attributes of all involved entity types
- Foreign key of the "N"-side becomes the key in the new relation

Initially!

- wine: {[ color, <u>name</u> ]}
- vintage: {[ <u>name → wine, year</u>, residualSweetness ]}
- belongsTo: {[ <u>name → wine, year → vintage</u> ]}

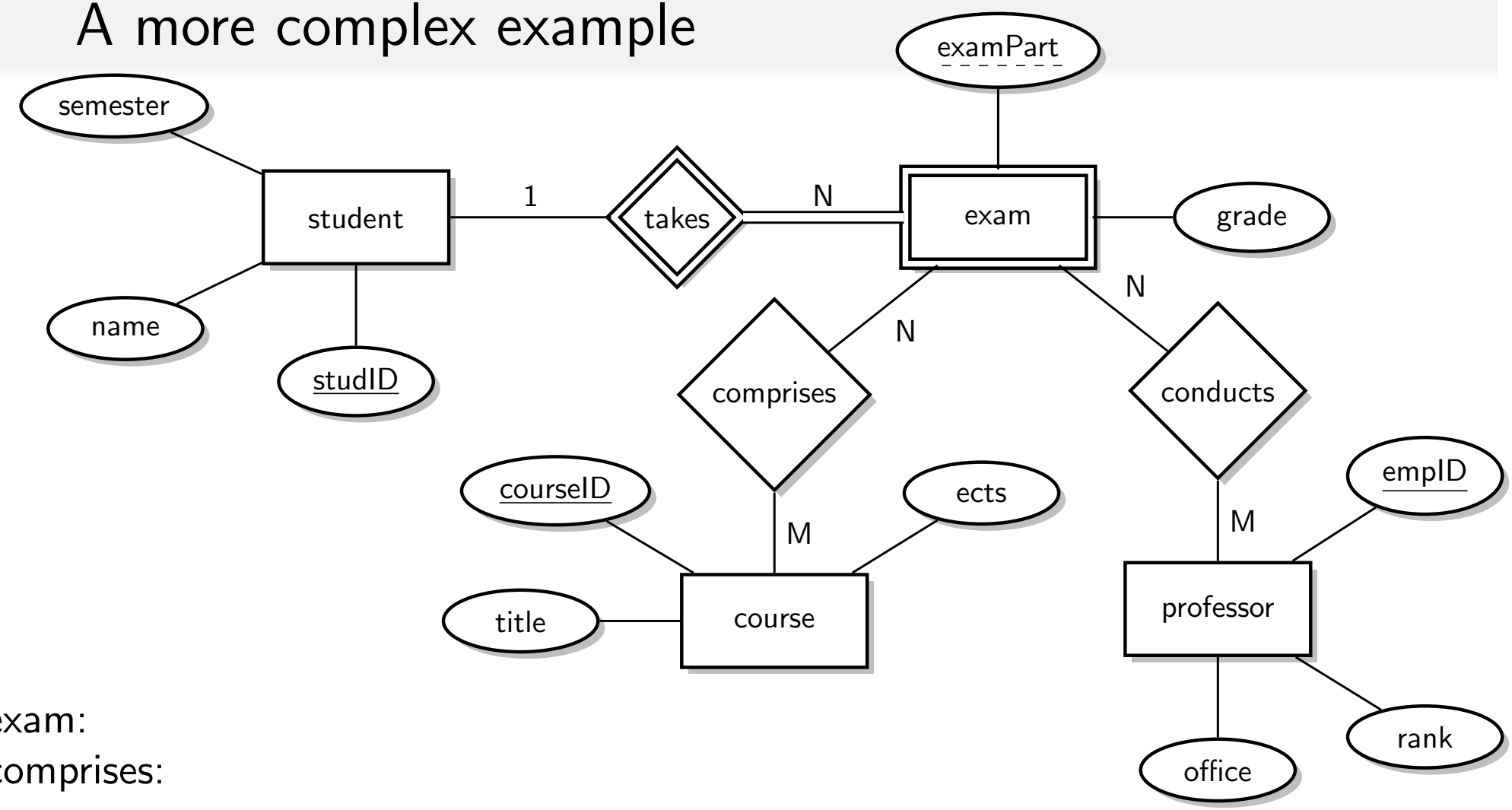
# Weak entity types




Weak entity types and their identifying relationship types can **always** be merged.

- wine: {[ color, <u>name</u> ]}
- vintage: {[ <u>name → wine, year</u>, residualSweetness ]}

# A more complex example

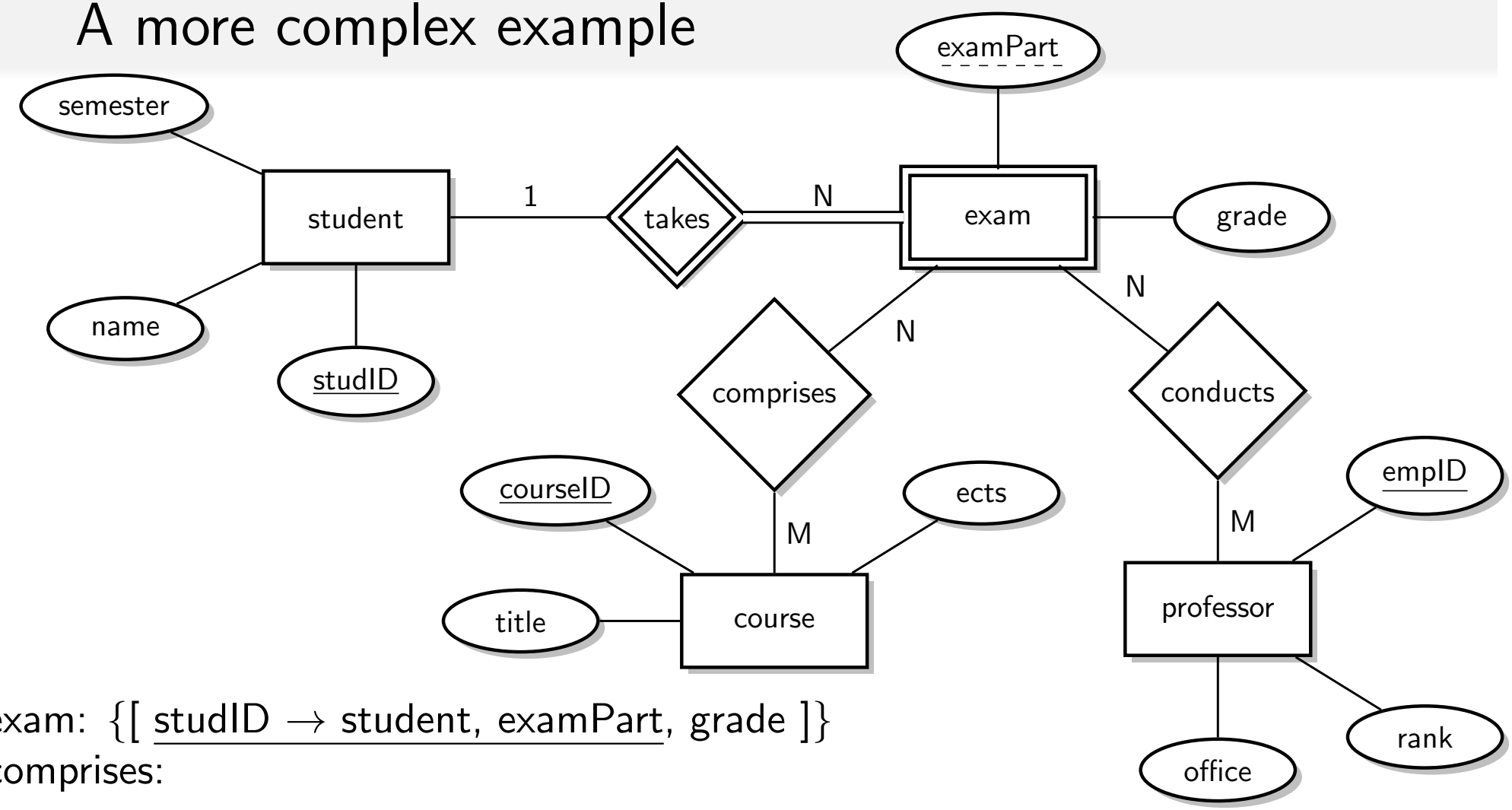



exam:

comprises:

conducts:

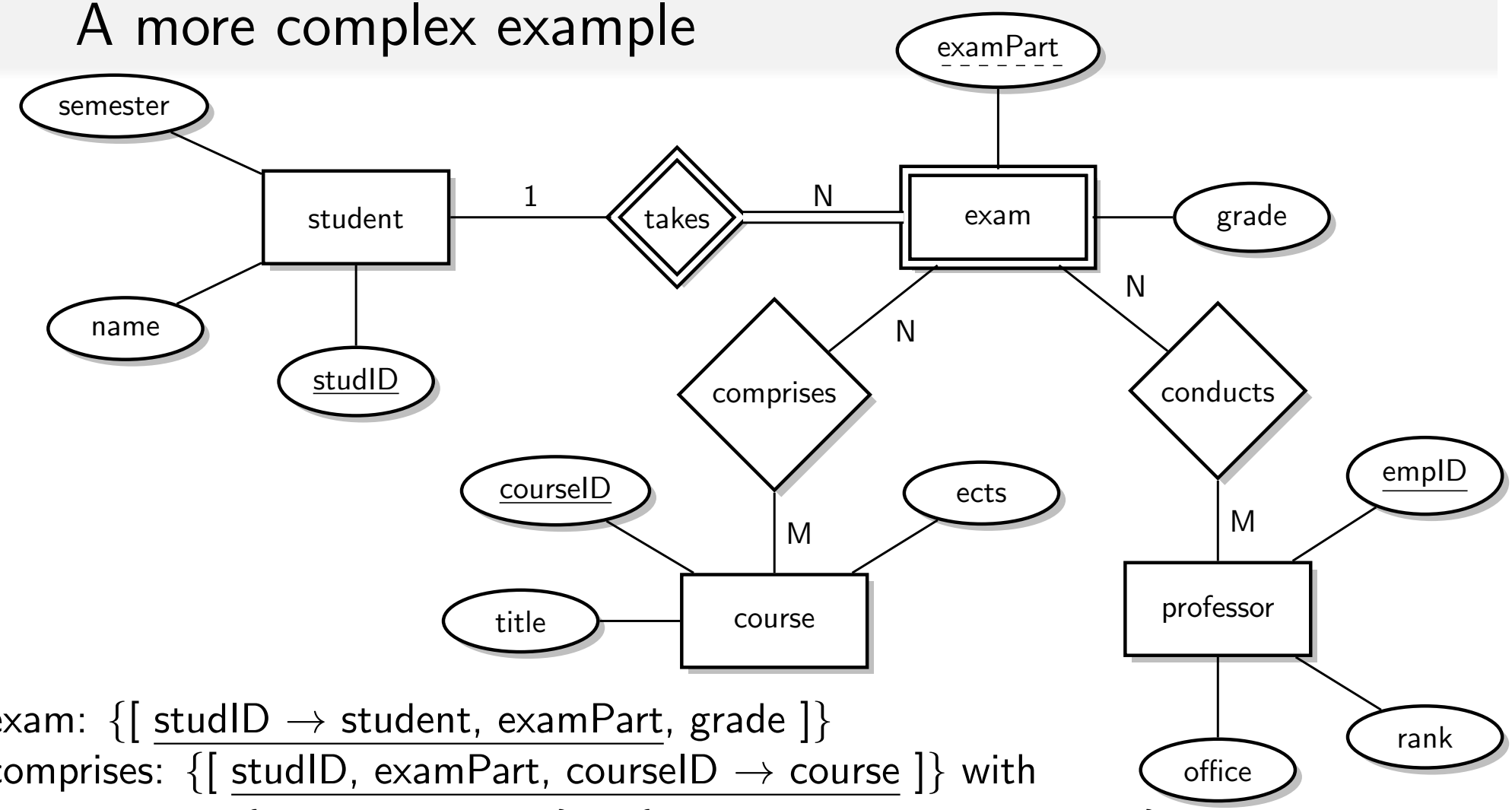
# A more complex example




exam: {[ studID → student, examPart, grade ]}
comprises:

conducts:

# A more complex example



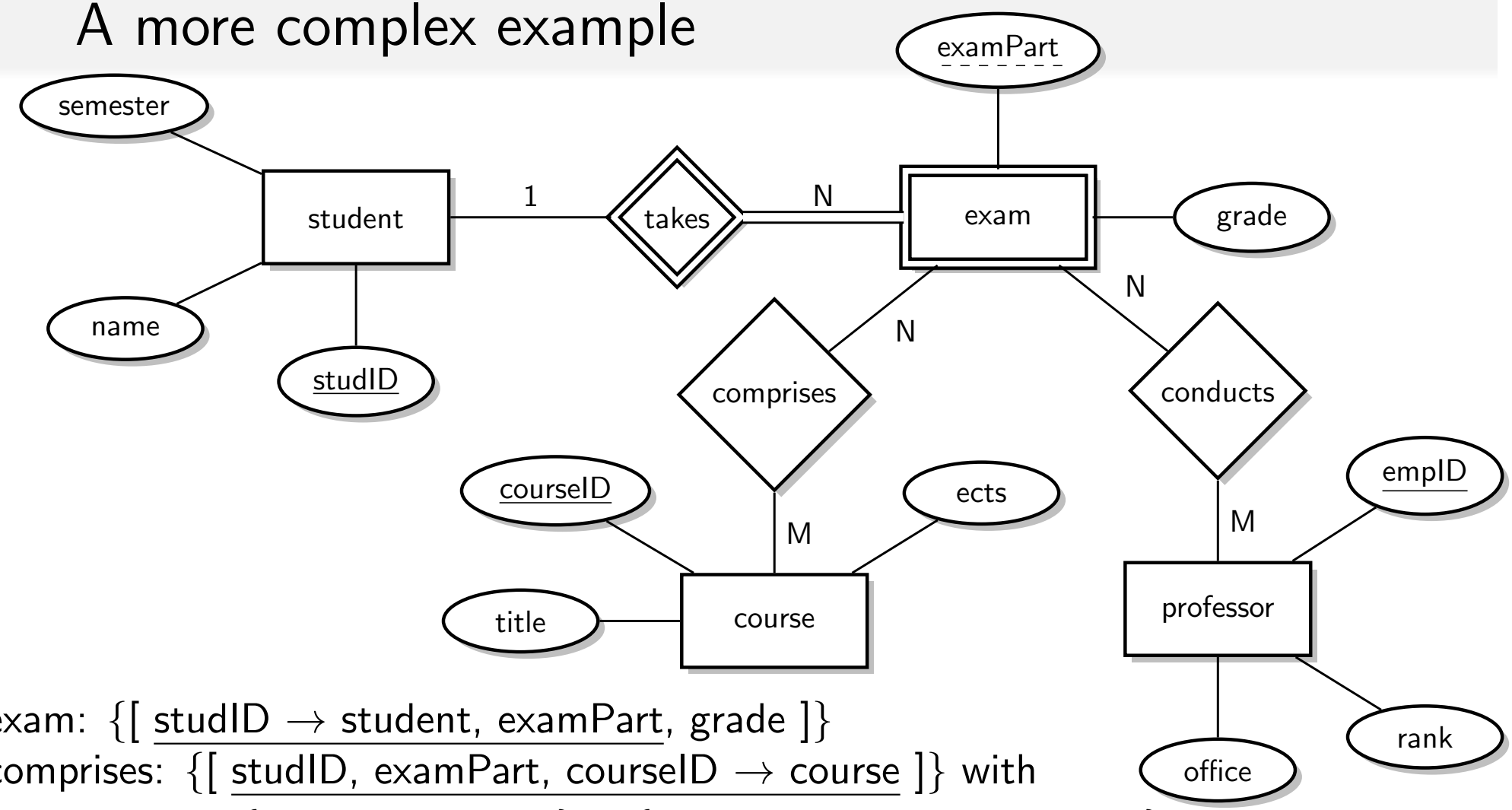

exam: {[ studID → student, examPart, grade ]}

comprises: {[ studID, examPart, courseID → course ]} with

Foreign key: $\{studID, examPart\} \rightarrow \{exam.studID, exam.examPart\}$

conducts:

# A more complex example




exam: {[ studID → student, examPart, grade ]}

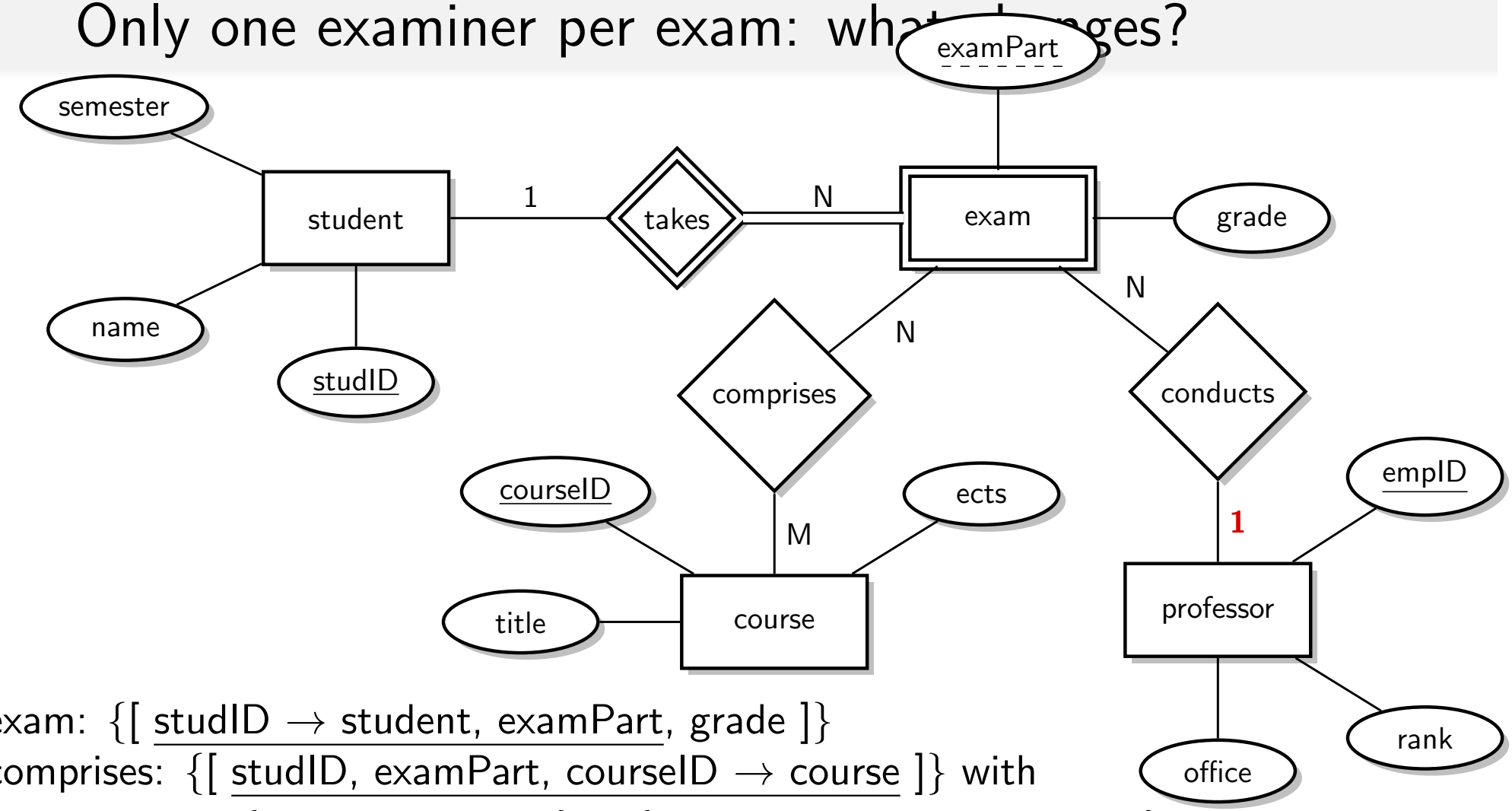
comprises: {[ studID, examPart, courseID → course ]} with

Foreign key: $\{studID, examPart\} \rightarrow \{exam.studID, exam.examPart\}$

conducts: {[ studID, examPart, empID → professor ]} with

Foreign key: $\{studID, examPart\} \rightarrow \{exam.studID, exam.examPart\}$

# Only one examiner per exam: what changes?




exam: {[ studID → student, examPart, grade ]}

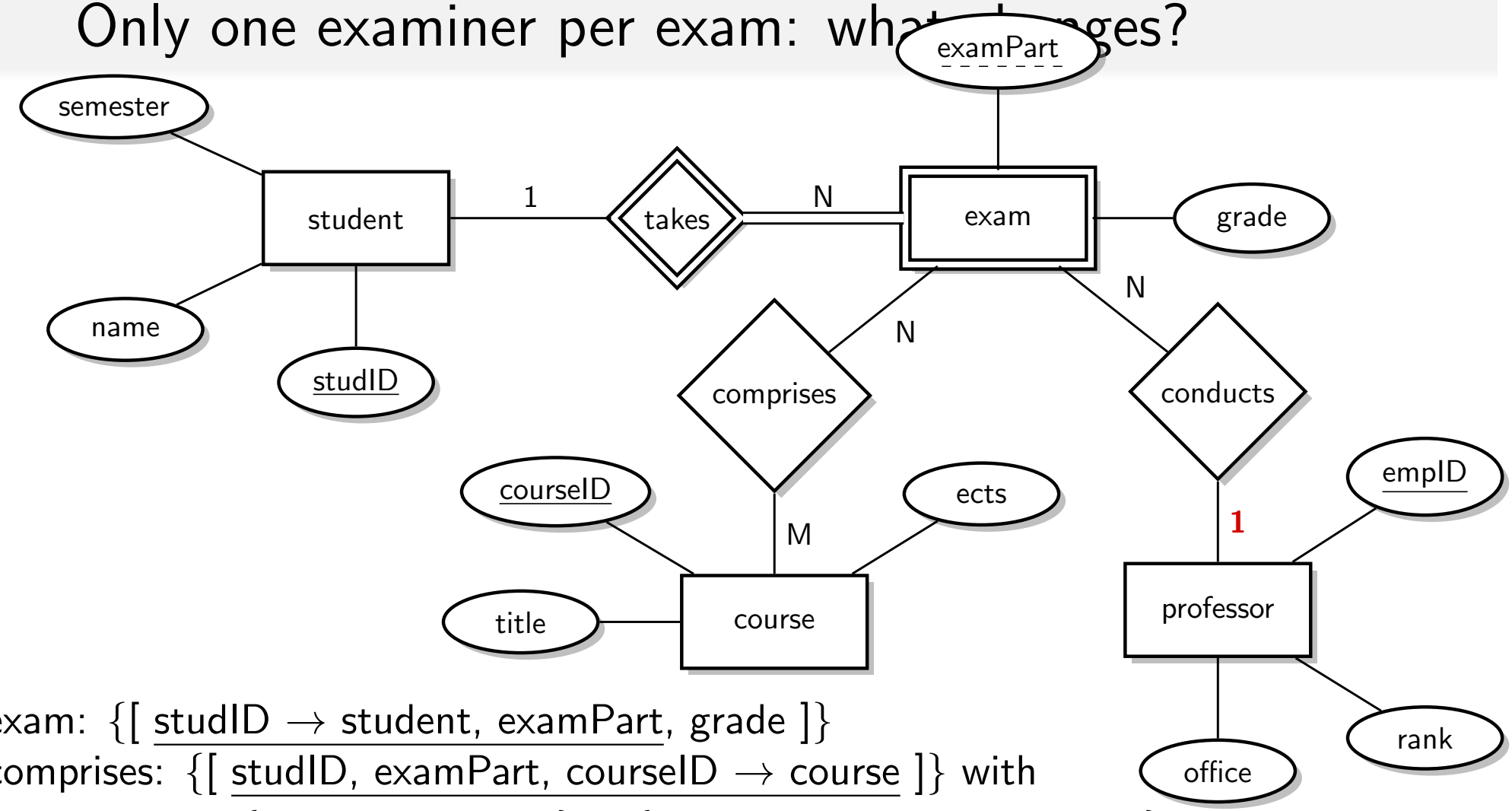
comprises: {[ studID, examPart, courseID → course ]} with

Foreign key: $\{studID, examPart\} \rightarrow \{exam.studID, exam.examPart\}$

conducts: {[ studID, examPart, empID → professor ]} with

Foreign key: $\{studID, examPart\} \rightarrow \{exam.studID, exam.examPart\}$

# Only one examiner per exam: what changes?




exam: {[ studID → student, examPart, grade ]}

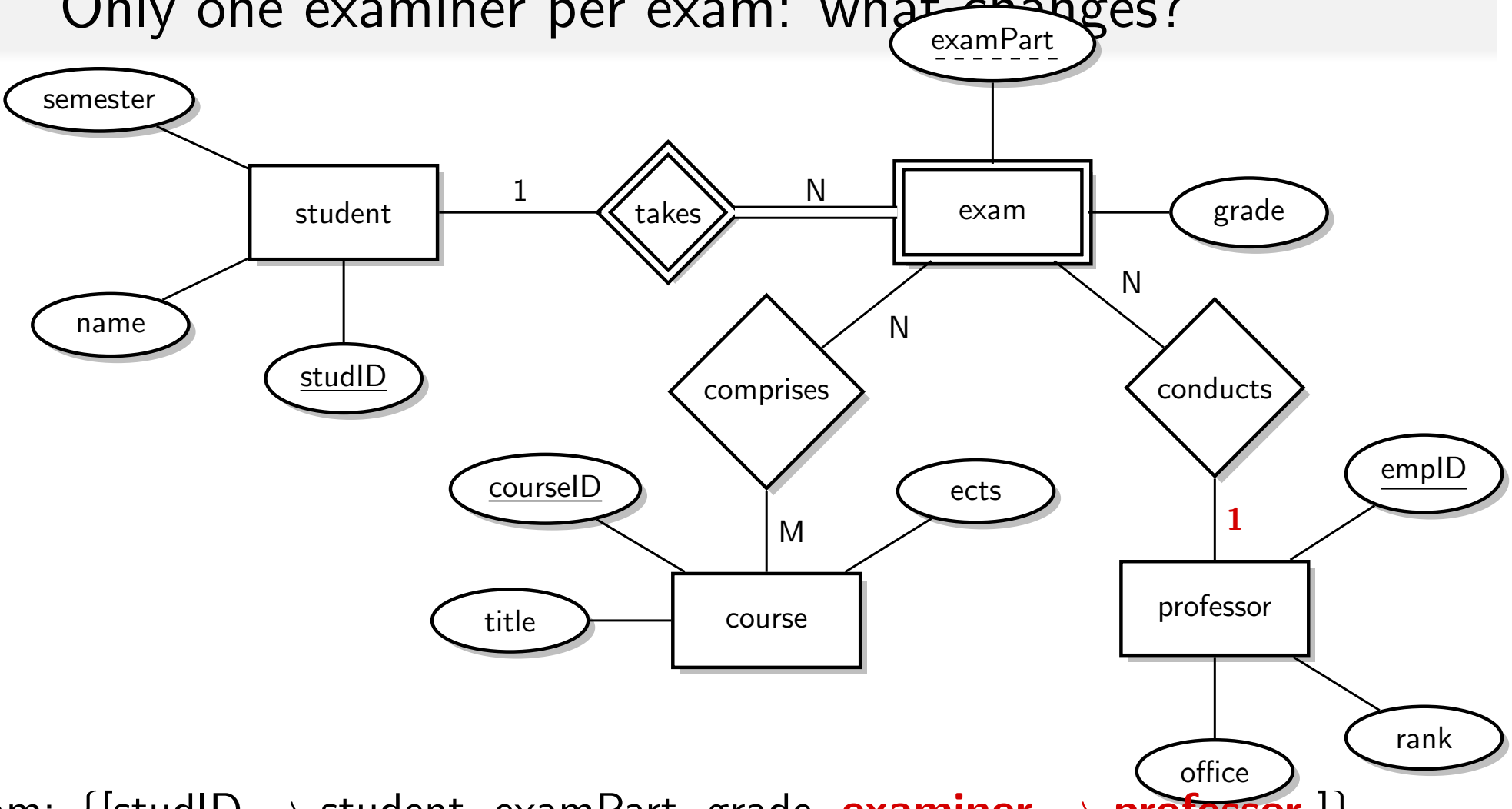
comprises: {[ studID, examPart, courseID → course ]} with

Foreign key: $\{studID, examPart\} \rightarrow \{exam.studID, exam.examPart\}$

conducts: {[ studID, examPart, empID → professor ]} with

Foreign key: $\{studID, examPart\} \rightarrow \{exam.studID, exam.examPart\}$

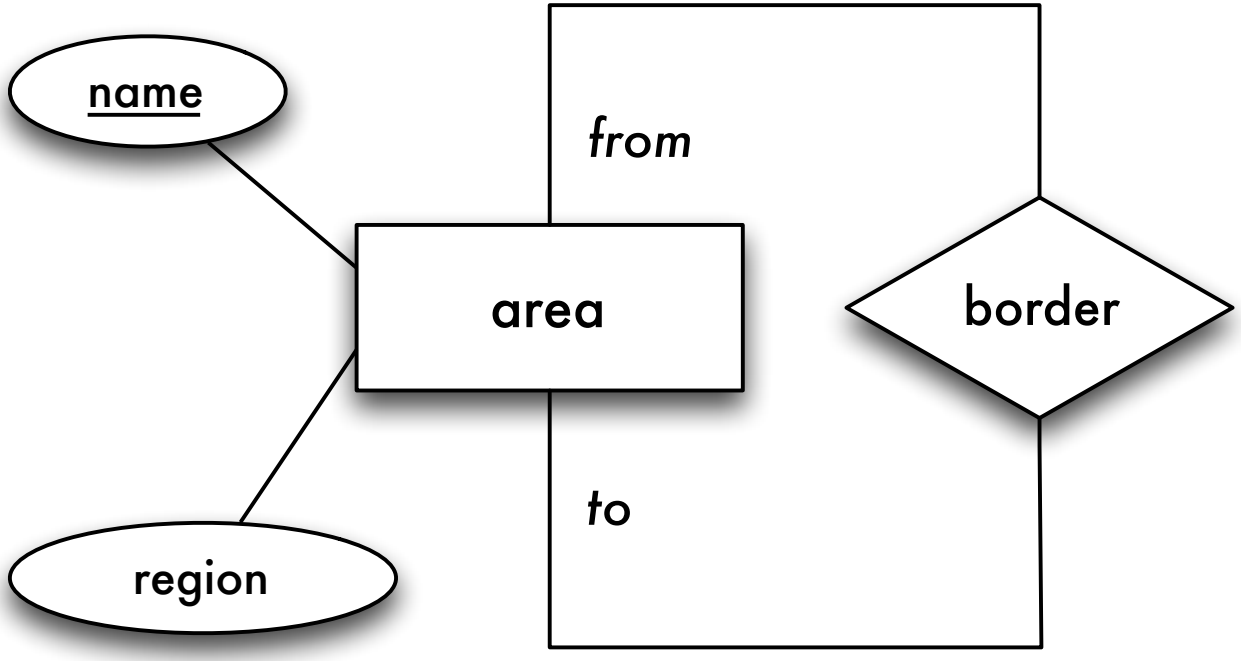
# Only one examiner per exam: what changes?




exam: {[studID → student, examPart, grade, **examiner → professor** ]}
comprises: {[studID, examPart, courseID → course ]} with

Foreign key: $\{studID, examPart\} \rightarrow \{exam.studID, exam.examPart\}$

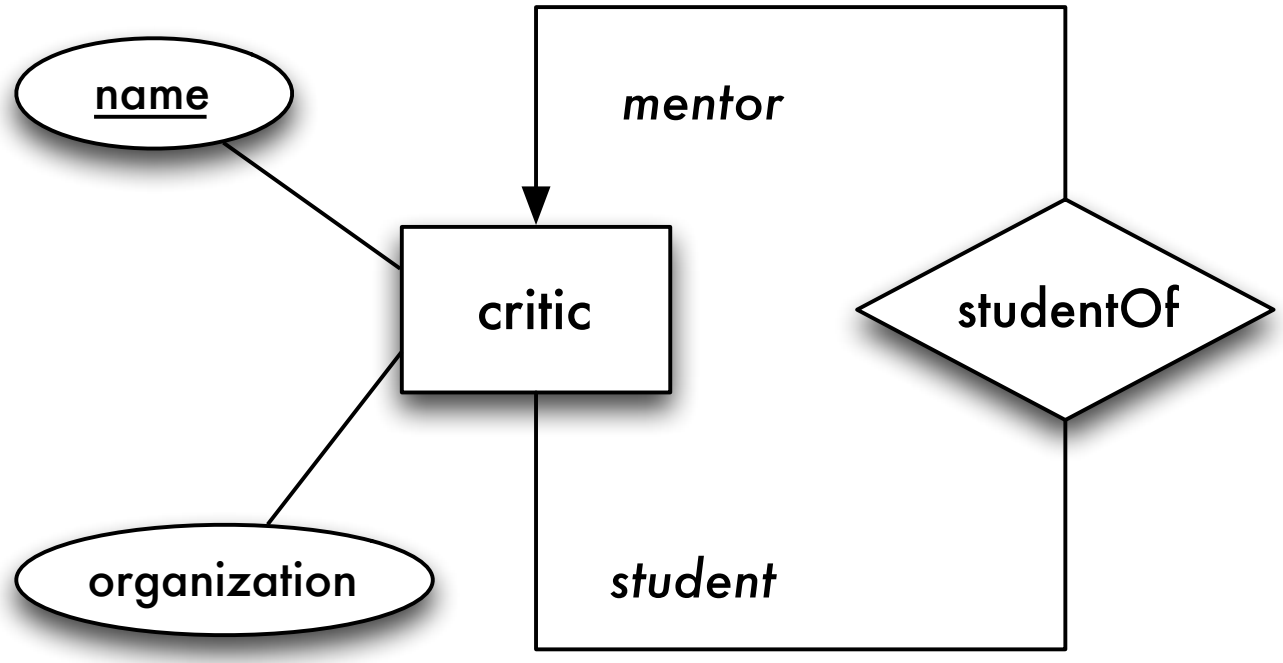
# Recursive relationship types




Mapping just like standard N:M relationship types and renaming of foreign keys

- area: {[ name, region ]}
- border: {[ from → area, to → area ]}

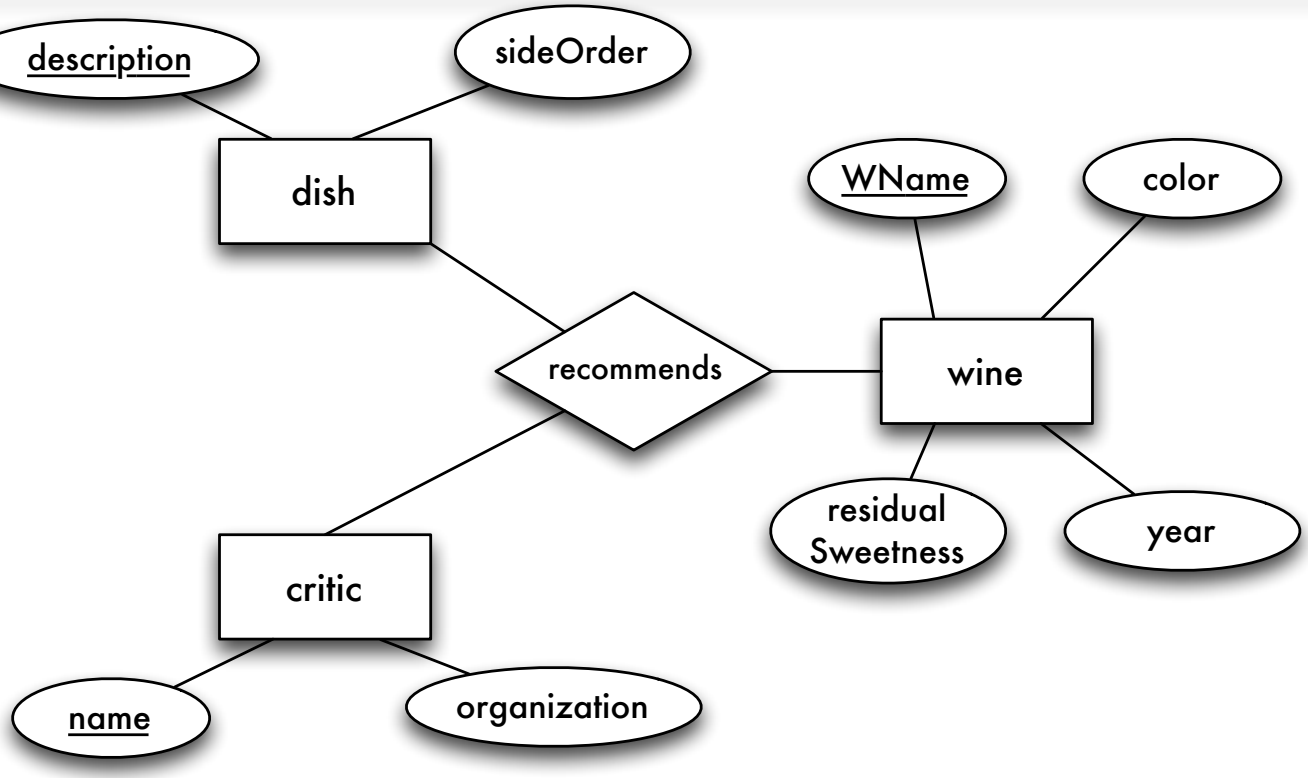
# Recursive functional relationship types




Mapping just like standard 1:N relationship types and merging

- critic: {[ name, organization, mentor → critic ]}

# N-ary relationship types

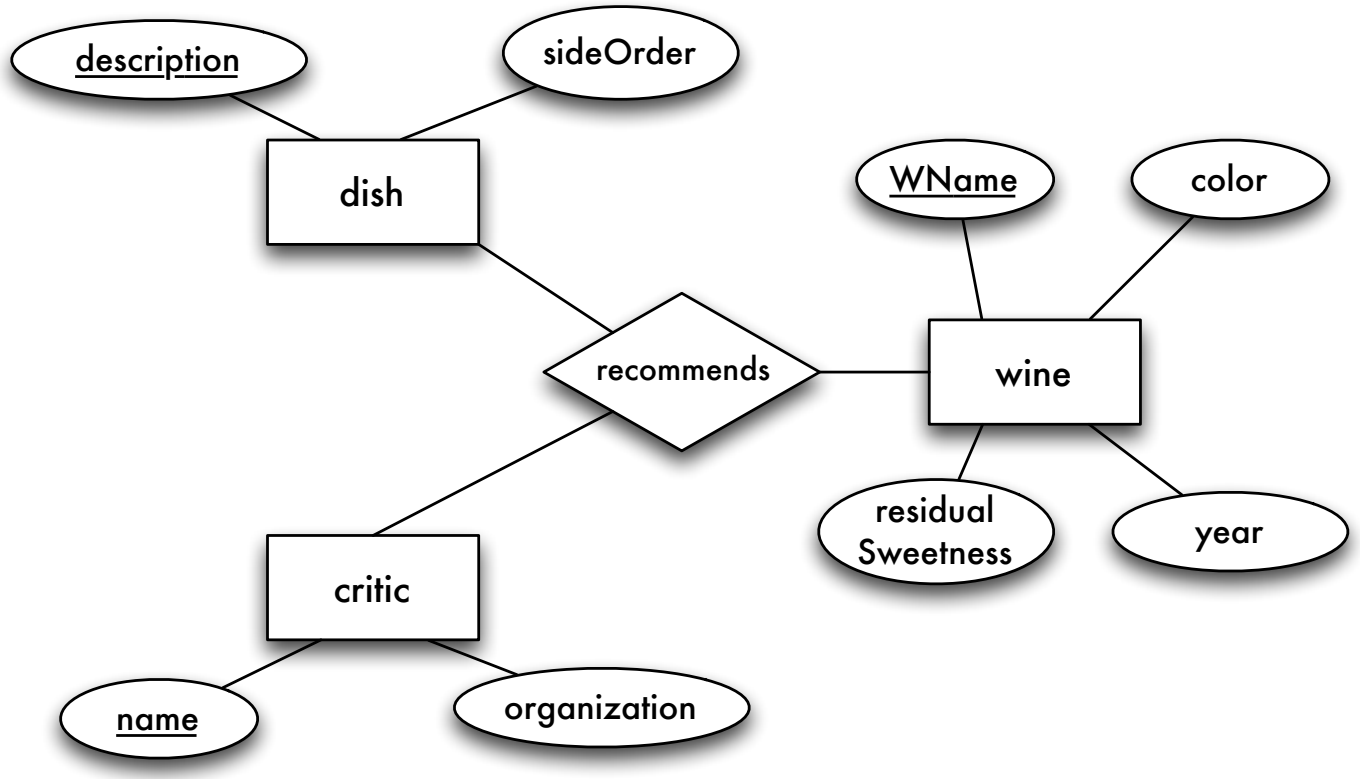



Entity types:

All participating entity types are mapped according to the standard rules.

- critic: {[ name, organization ]}
- dish: {[ description, sideOrder ]}
- wine: {[ color, WName, year, residualSweetness ]}

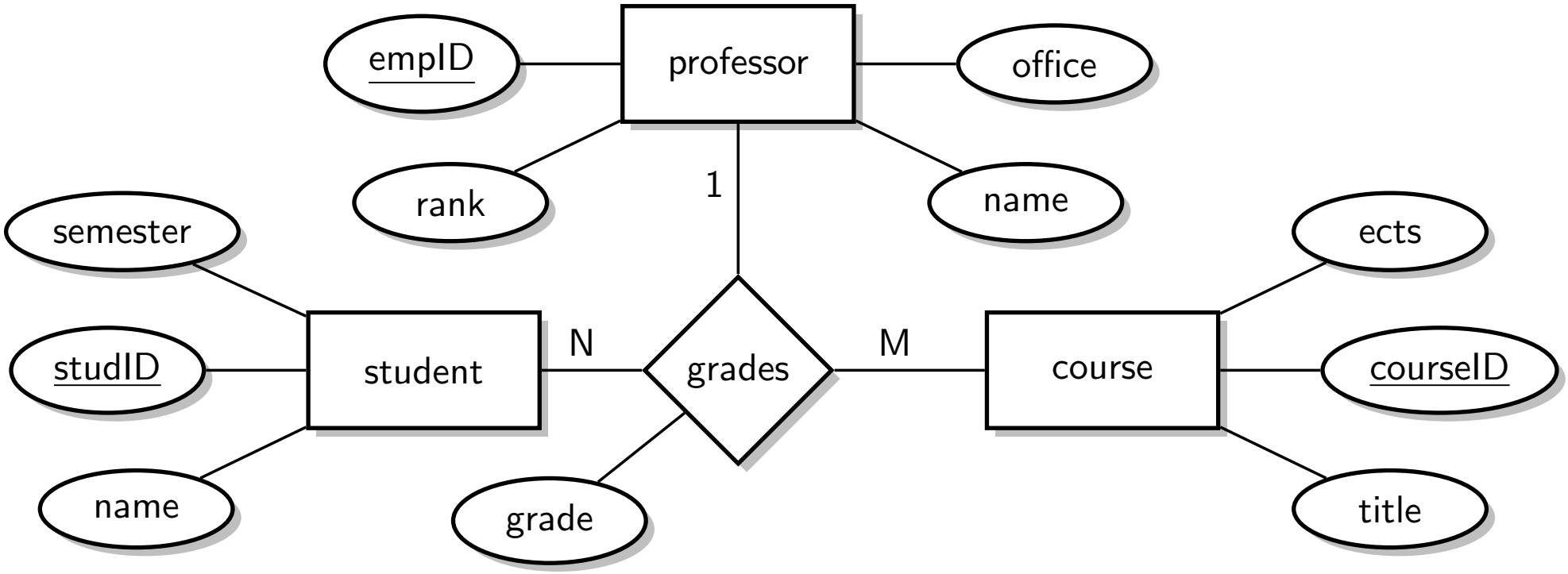
# N-ary relationship types




N-ary relationship types (N:M:P)

recommends: {[ WName → wine, description → dish, name → critic ]}

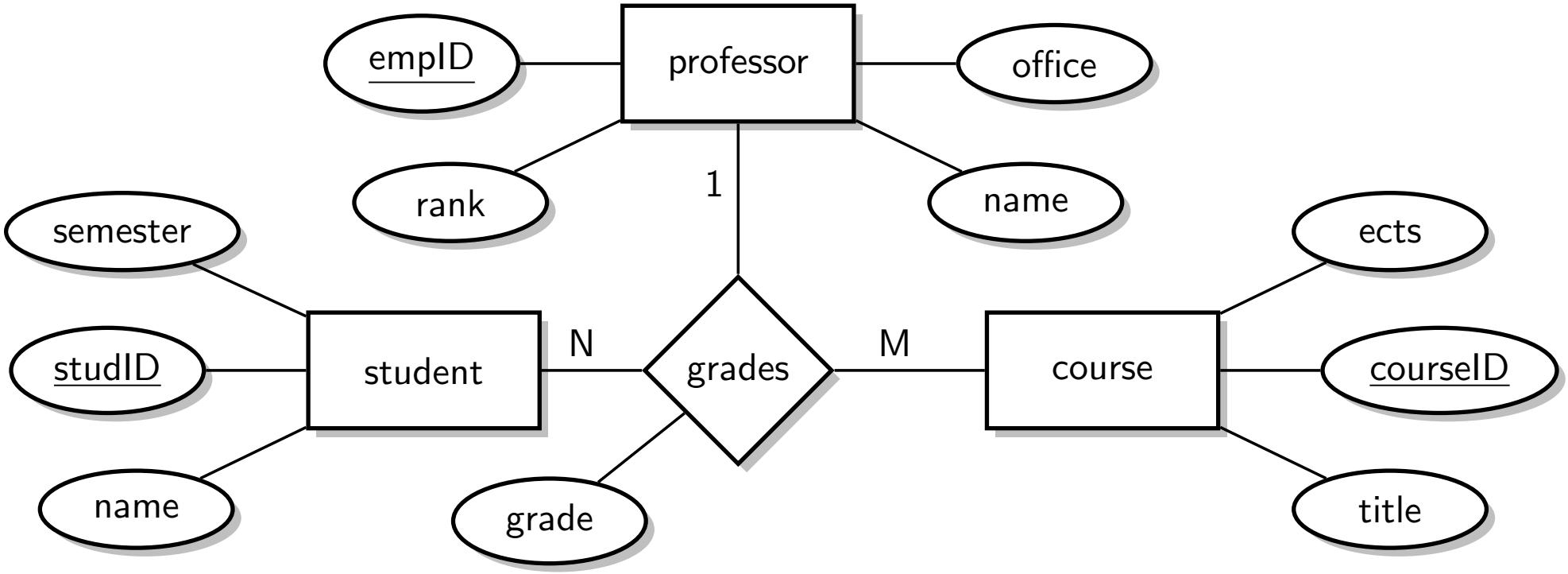
# N:M:1 relationship type




Relations

- student: {[ studID, name, semester ]}
- course: {[ courseID, title, ects ]}
- professor: {[ empID, name, rank, office ]}
- grades:

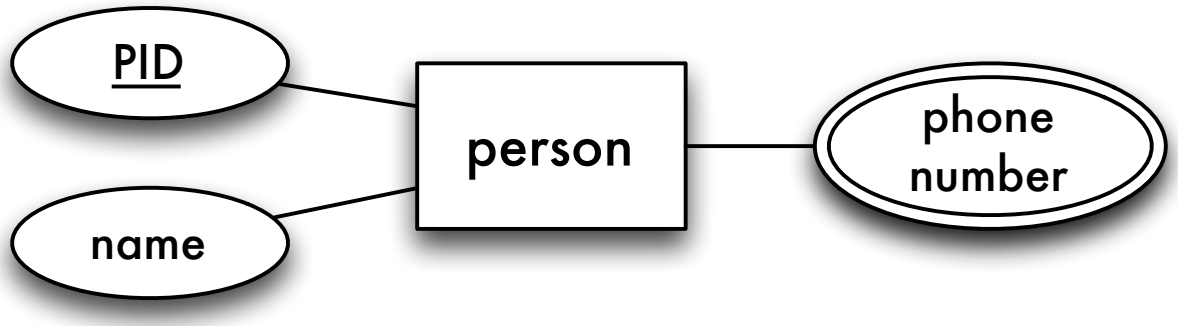
# N:M:1 relationship type




Relations

- student: {[ studID, name, semester ]}
- course: {[ courseID, title, ects ]}
- professor: {[ empID, name, rank, office ]}
- grades: {[ studID → student, courseID → course, empID → professor, grade ]}

# Multi-valued attributes

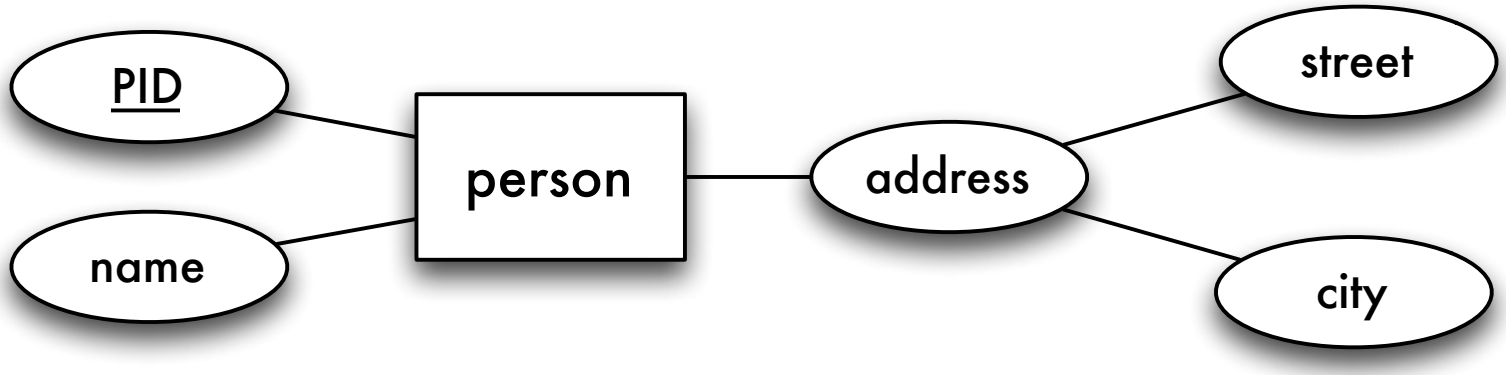



Create a separate relation for each multi-valued attribute

Relations

- person: {[ PID, name ]}
- phoneNumber: {[ PID → person, number ]}

# Composite attributes

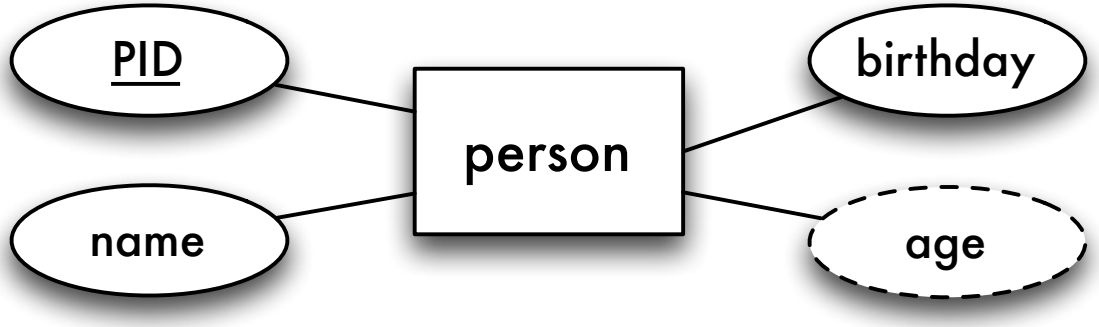



**Include the component attributes in the relation**

Relation

- person: {[ PID, name, street, city ]}

# Derived attributes

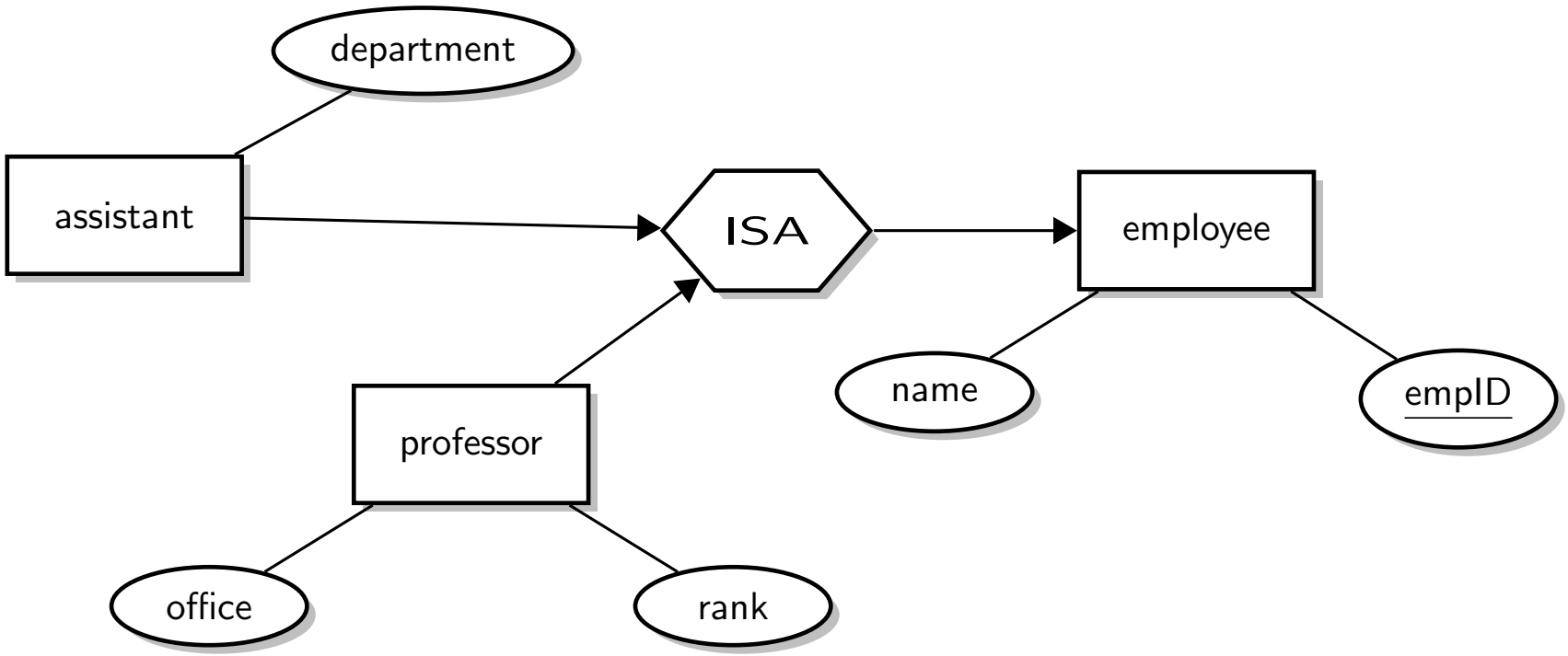



Ignored during mapping to relations, can be added later by using views.

# Overview of the steps

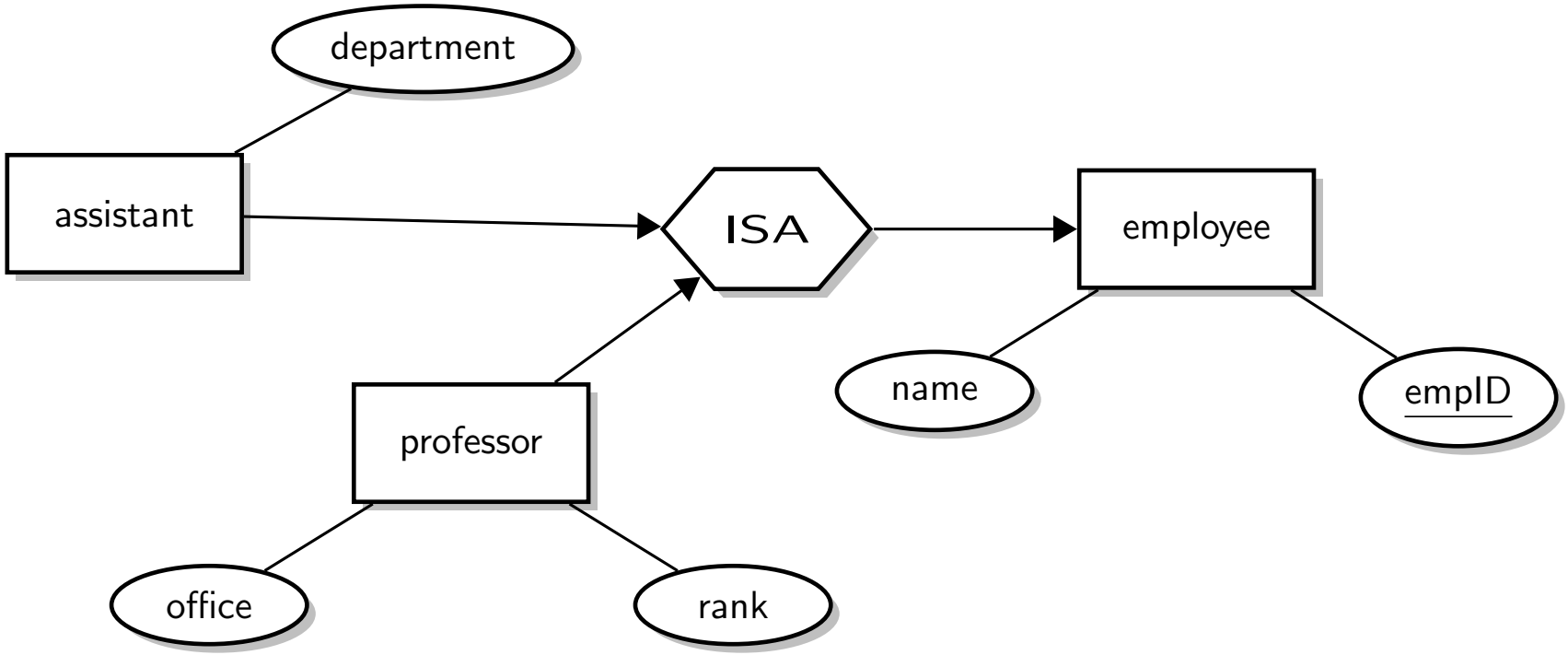
1. Regular entity type
   Create a relation, consider special attribute types
2. Weak entity type
   Create a relation
3. 1:1 binary relationship type
   Extend a relation with foreign key
4. 1:N binary relationship type
   Extend a relation with foreign key
5. N:M relationship type
   Create a relation
6. N-ary relationship type
   Create a relation

# Relational modeling of generalization



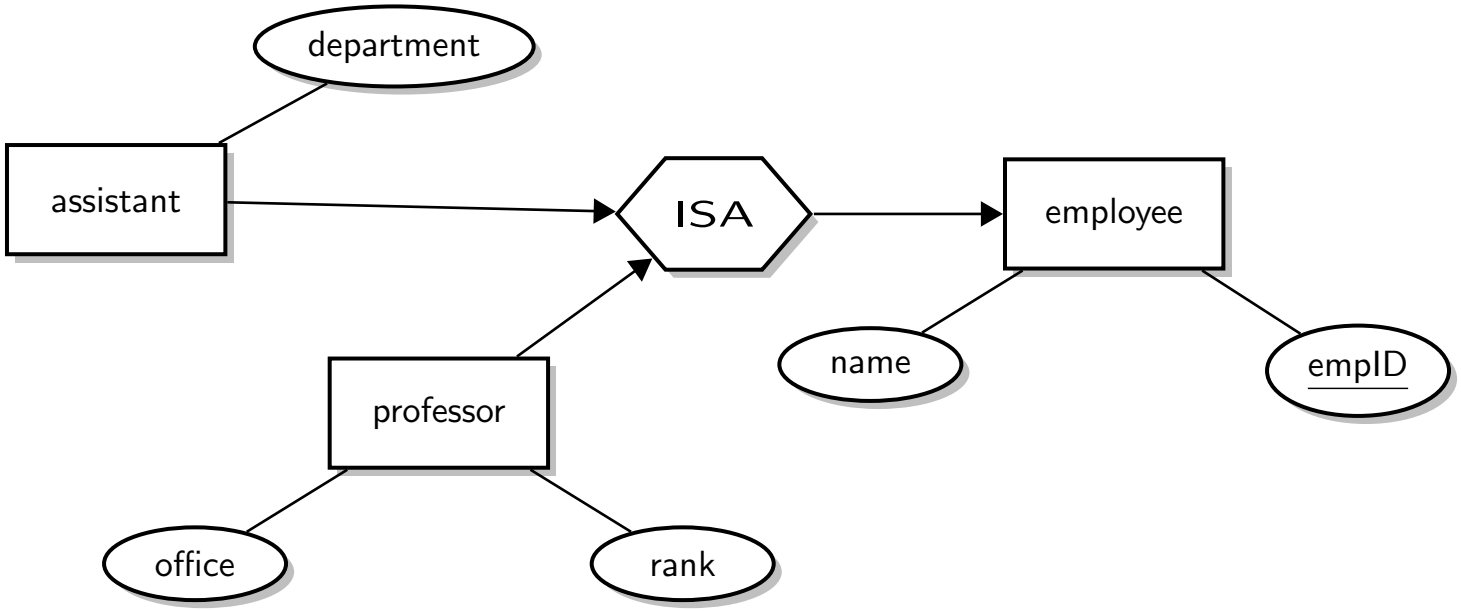

The relational model does not support generalization and cannot express inheritance.
→ Generalization is **simulated**.

# Relational modeling of generalization




How to map this information to relations?

# Alternative 1: main classes




A particular entity is mapped to **a single tuple** in a single relation (to its main class).

- employee: {[ empID, name ]}
- professor: {[ empID, name, rank, office ]}
- assistant: {[ empID, name, department ]}

# Alternative 1: main classes

| employee | |
|---|---|
| empID | name |
| 2123 | P. Müller |
| 2124 | A. Schmidt |

employee:
{[ empID, name ]}

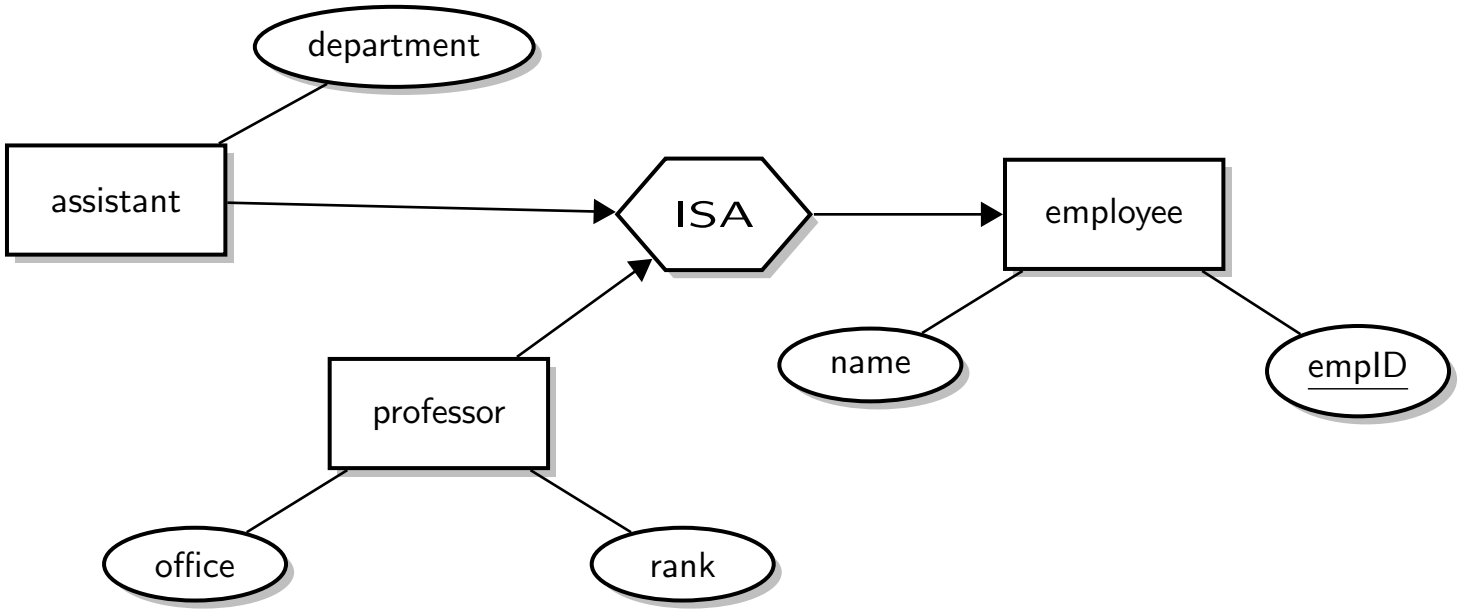
| professor | | | |
|---|---|---|---|
| empID | name | rank | office |
| 2125 | Socrates | C4 | 226 |
| 2126 | Russel | C3 | 232 |
| 2127 | Kopernikus | C3 | 310 |
| 2128 | Curie | C4 | 36 |

professor:
{[ empID, name, rank, office ]}

| assistant | | |
|---|---|---|
| empID | name | department |
| 2150 | C. Meyer | DBS |
| 2151 | B. Fischer | Physics |

assistant:
{[ empID, name, department ]}

# Alternative 2: partitioning




**Parts of** a particular entity are mapped to multiple relations, the key is duplicated.

- employee: {[ empID, name ]}
- professor: {[ empID → employee, rank, office ]}
- assistant: {[ empID → employee, department ]}

# Alternative 2: partitioning

| employee | |
|---|---|
| empID | name |
| 2123 | P. Müller |
| 2124 | A. Schmidt |
| 2125 | Socrates |
| ... | ... |
| 2150 | C. Meyer |
| 2151 | B. Fischer |

employee:
{[ empID, name ]}

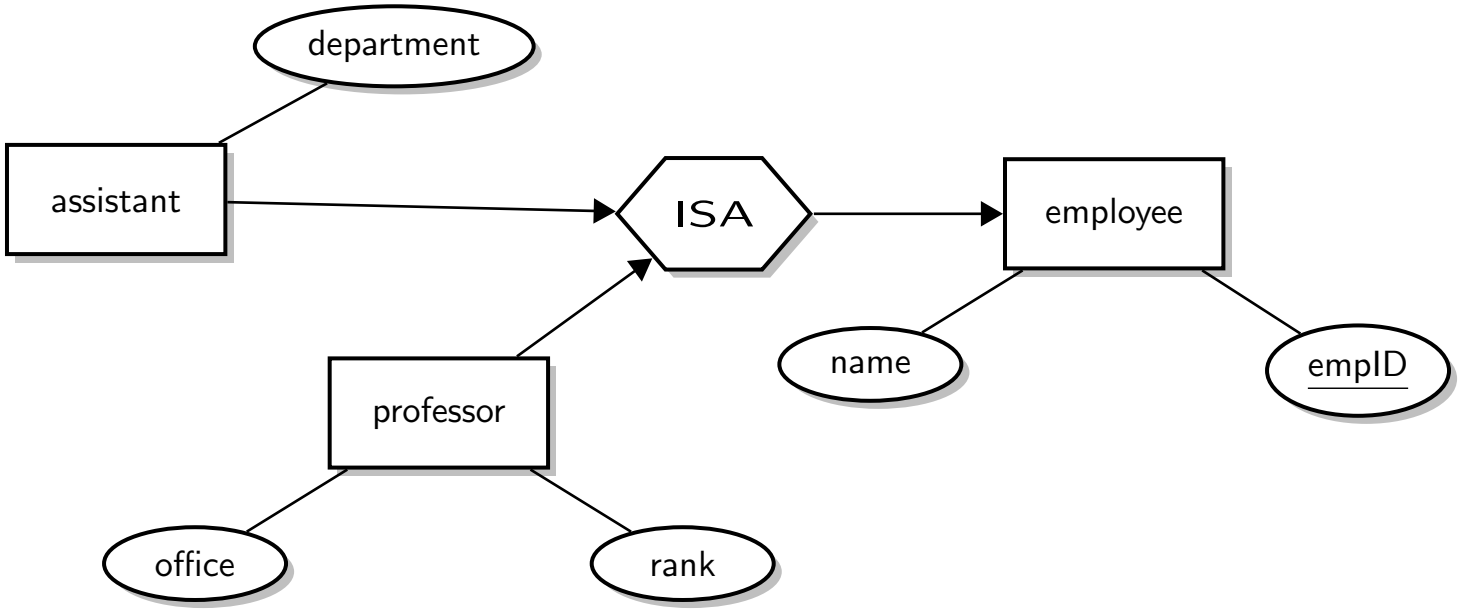
| professor | | |
|---|---|---|
| empID | rank | office |
| 2125 | C4 | 226 |
| ... | ... | ... |

professor:
{[ empID → employee, rank, office ]}

| assistant | |
|---|---|
| empID | department |
| 2150 | DBS |
| 2151 | Physics |

assistant:
{[ empID → employee, department ]}

# Alternative 3: full redundancy




A particular entity is stored **redundantly** in the relations with all its inherited attributes.

- employee: {[ empID, name ]}
- professor: {[ empID, name, rank, office ]}
- assistant: {[ empID, name, department ]}

# Alternative 3: full redundancy

| employee | |
|---|---|
| empID | name |
| 2123 | P. Müller |
| 2124 | A. Schmidt |
| 2125 | Socrates |
| ... | ... |
| 2150 | C. Meyer |
| 2151 | B. Fischer |

employee:
{[ empID, name ]}

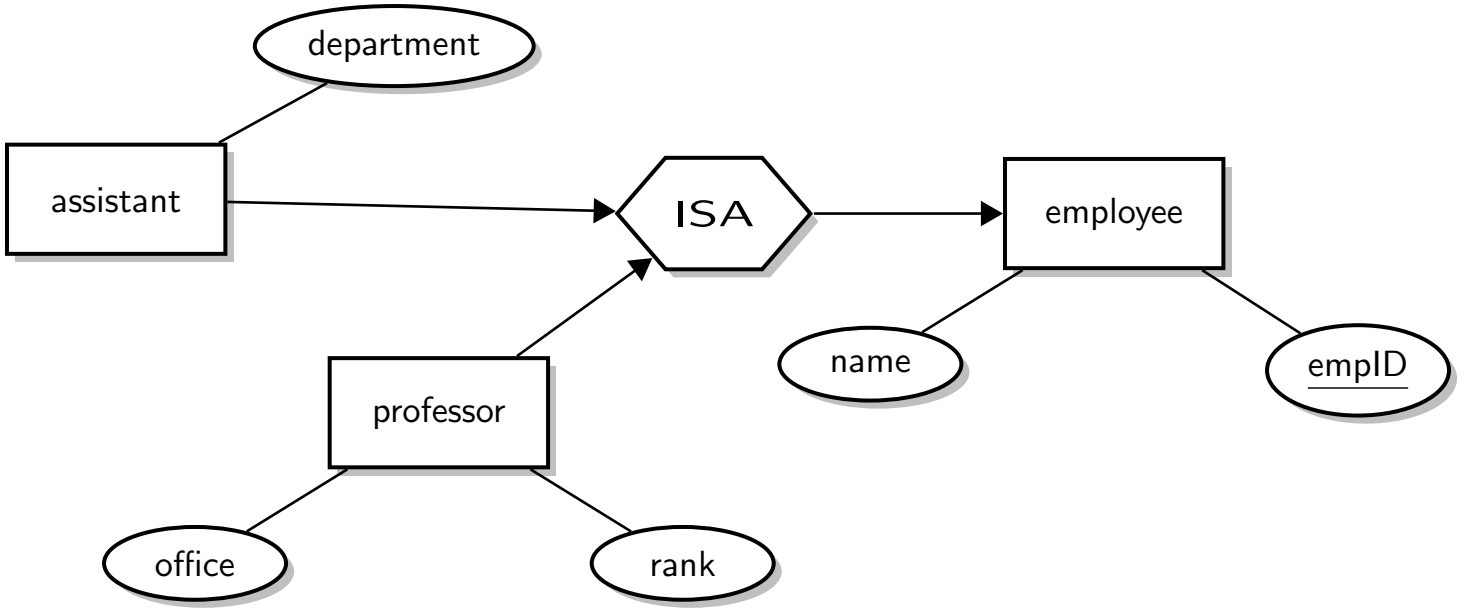
| professor | | | |
|---|---|---|---|
| empID | name | rank | office |
| 2125 | Socrates | C4 | 226 |
| ... | ... | ... | ... |

professor:
{[ empID, name, rank, office ]}

| assistant | | |
|---|---|---|
| empID | name | department |
| 2150 | C. Meyer | DBS |
| 2151 | B. Fischer | Physics |

assistant:
{[ empID, name, department ]}

# Alternative 4: single relation




All entities are stored in **a single relation**. An additional attribute encodes the membership in a particular entity type.

- employee: {[ empID, name, **type**, rank, office, department ]}

# Alternative 4: single relation

employee: {[ empID, name, type, rank, office, department ]}

| employee | | | | | |
|---|---|---|---|---|---|
| empID | name | type | rank | office | department |
| 2123 | P. Müller | employee | ⊥ | ⊥ | ⊥ |
| 2124 | A. Schmidt | employee | ⊥ | ⊥ | ⊥ |
| 2125 | Socrates | professor | C4 | 226 | ⊥ |
| 2126 | Russel | professor | C3 | 232 | ⊥ |
| 2127 | Kopernikus | professor | C3 | 310 | ⊥ |
| 2128 | Curie | professor | C4 | 36 | ⊥ |
| 2150 | C. Meyer | assistant | ⊥ | ⊥ | DBS |
| 2151 | B. Fischer | assistant | ⊥ | ⊥ | Physics |

# Summary

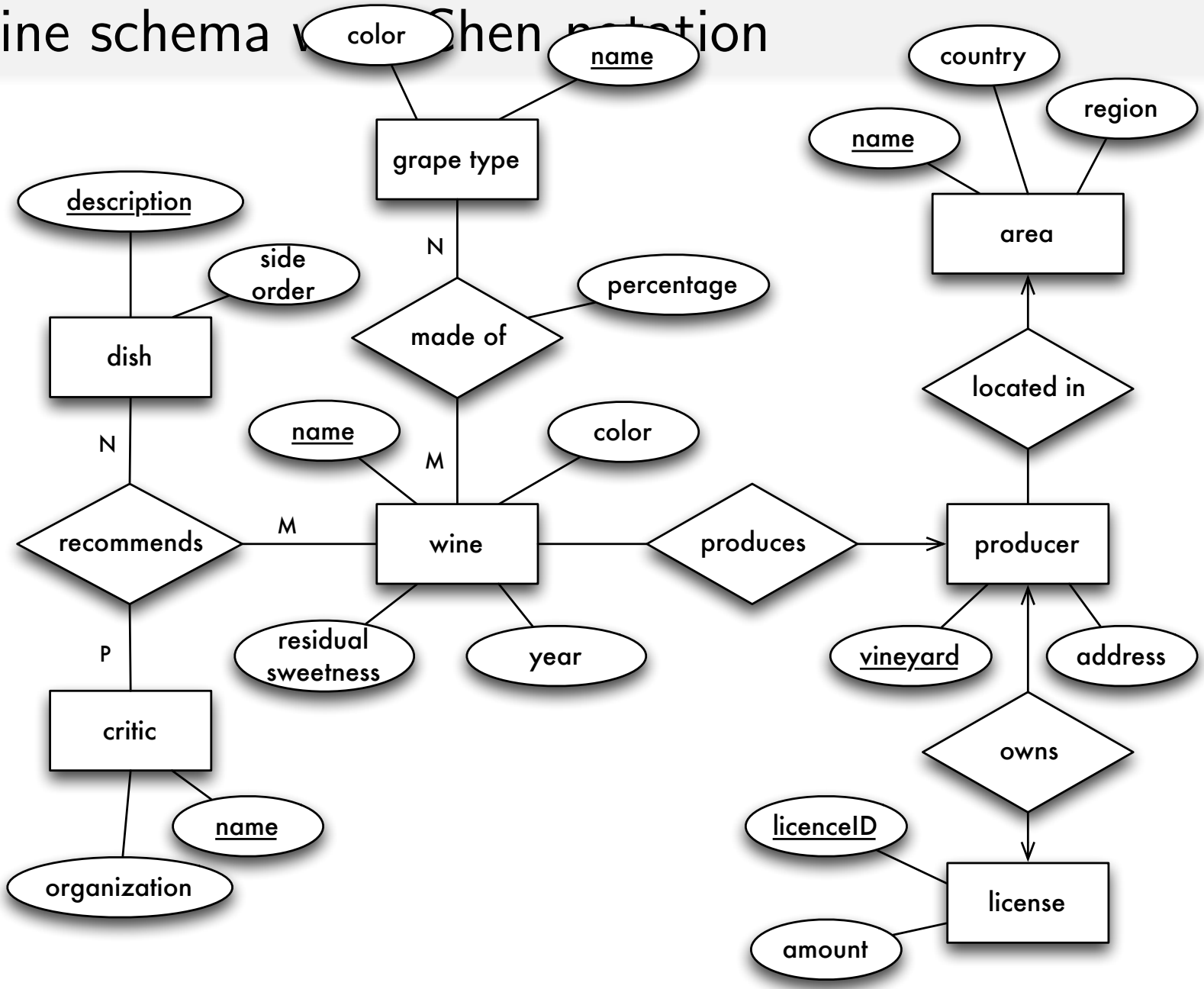
Mapping ER diagrams to relations

- Entity types
- Binary relationship types
- N-ary relationship types
- Weak entity types
- Recursive relationship types
- Generalization
  - The "partitioning" alternative is preferred in most applications.
- The discussed "mapping" rules aim for a minimum number of relations, not necessarily minimum null values and redundancy!

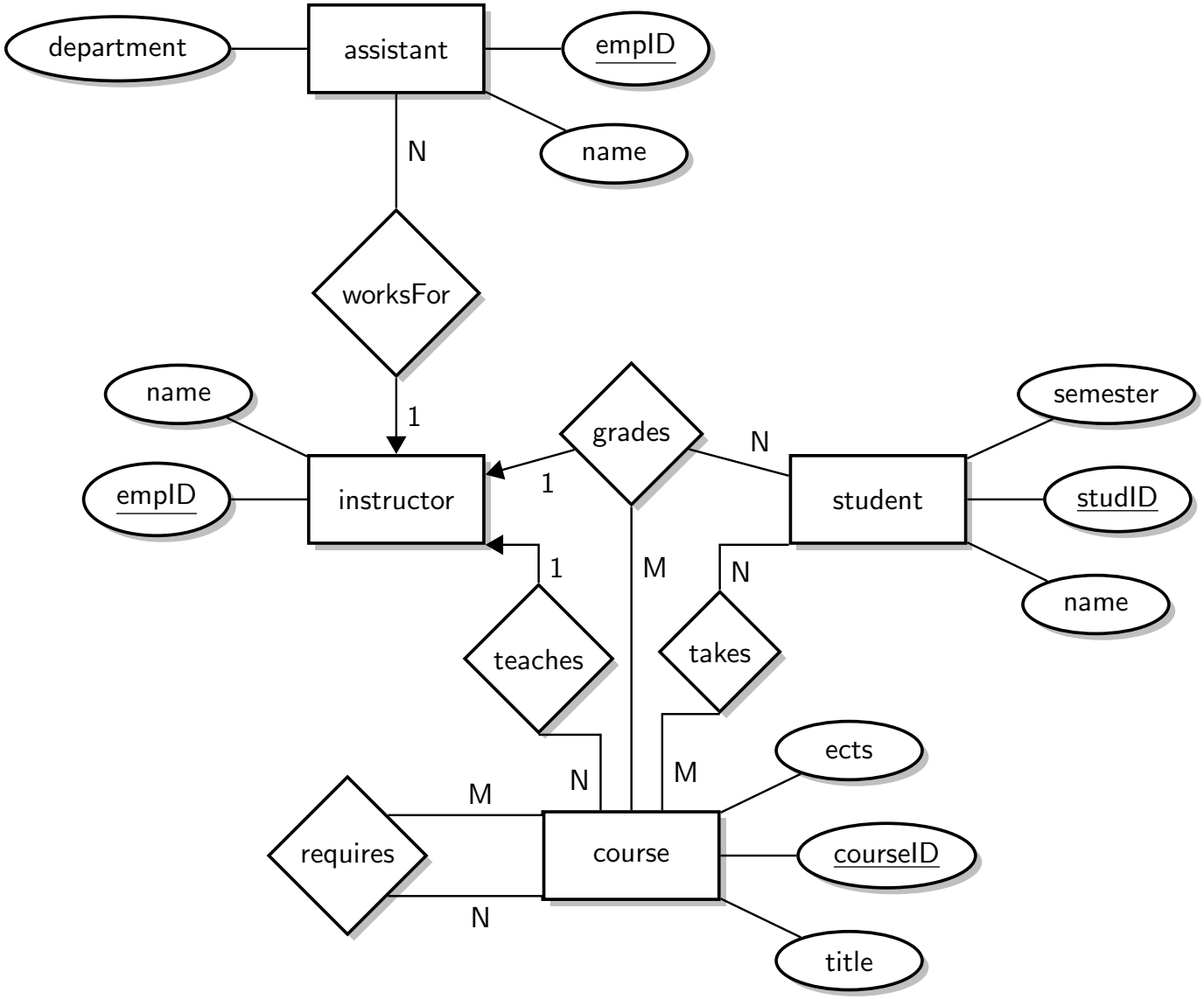
# Appendix

1. Database design
   - Steps of database design
   - Example design
2. Basic concepts
   - Example scenarios
   - Entity types
   - Attributes
   - Relationship types
3. Characteristics of relationship types
   - Degree
   - Chen notation (cardinality ratio)
   - Participation constraint
   - Chen notation (cardinality ratios) for nary relationship types
   - $[min, max]$ notation (cardinality limits)
4. Additional concepts
   - Weak entity types
   - The ISA relationship type
5. Alternative notations
6. Mapping basic concepts to relations

# Wine schema v Chen notation



# University schema with cardinality ratios



# Acknowledgement


- Christian S. Jensen, Aalborg University
- slides of Database System concept book