Databases

Conceptual Data Modelling

Danilo Montesi

danilo.montesi@unibo.it

Conceptual Models, Why? (1)

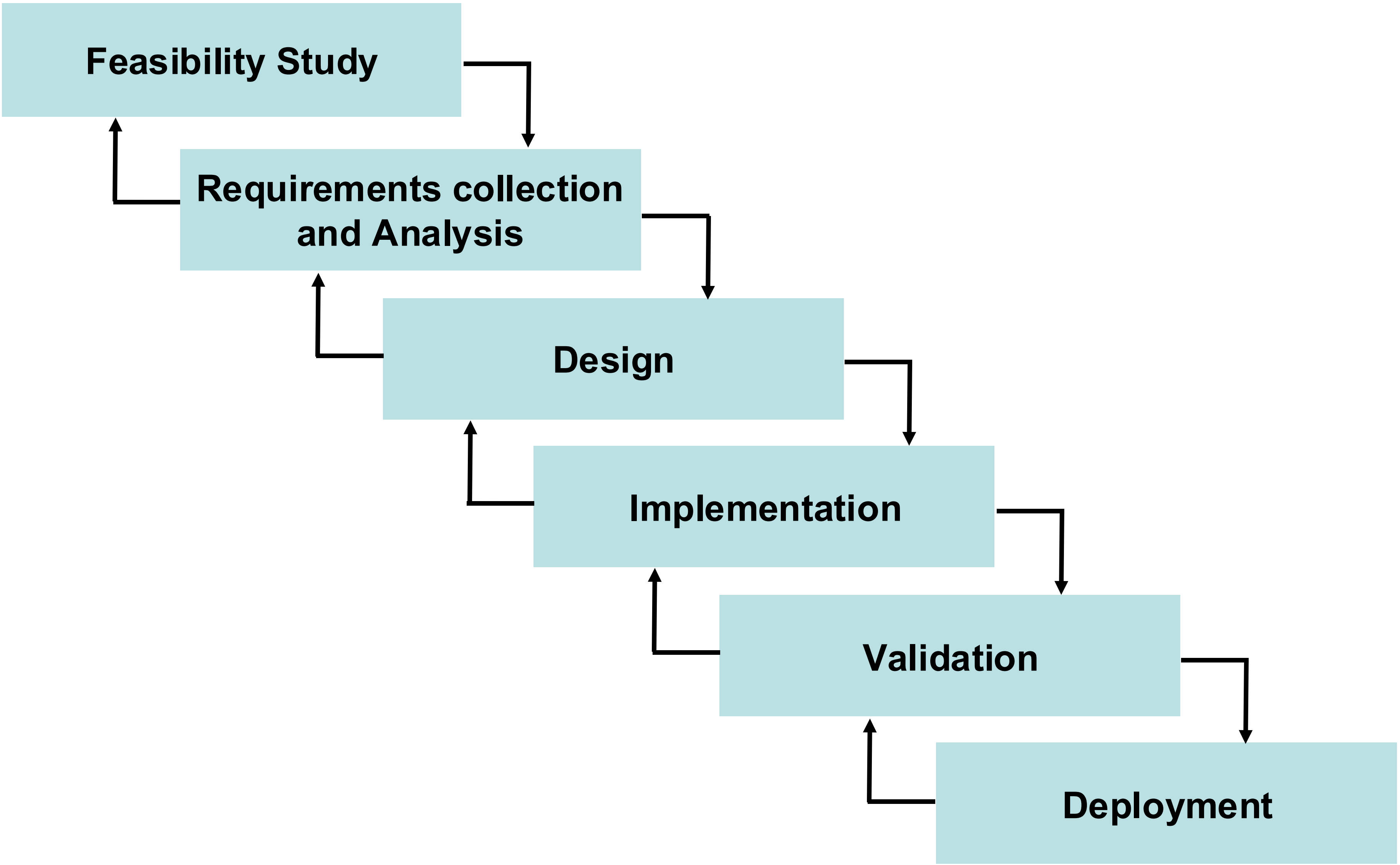
Let’s try to build a relational Database directly from the logical model:

- Where shall we start from?
- There’s a risk to go into any further detail
- We need to define relations between the tables
- The relational model is too “rigid” for modelling

Database Design

- Design is one of the tasks belonging to the Information System's development process
- It must be seen against a more general context:
  - Information Systems' (IS) life cycle:
    - The set of the tasks sequentially performed by business analysts, software architects, users, within both the develop and usage of information systems
    - Iterative task ("cycles")

Life Cycle


Feasibility Study
Requirements collection and Analysis
Design
Implementation
Validation
Deployment

Software Life Cycle: Phases

- **Feasibility Study**: costs and priorities definition
- **Requirement collection and Analysis**: system’s properties analysis
- **Design**: data and methods
- **Implementation**
- **Validation**: model checking, debugging
- **Deployment**: system running

Design

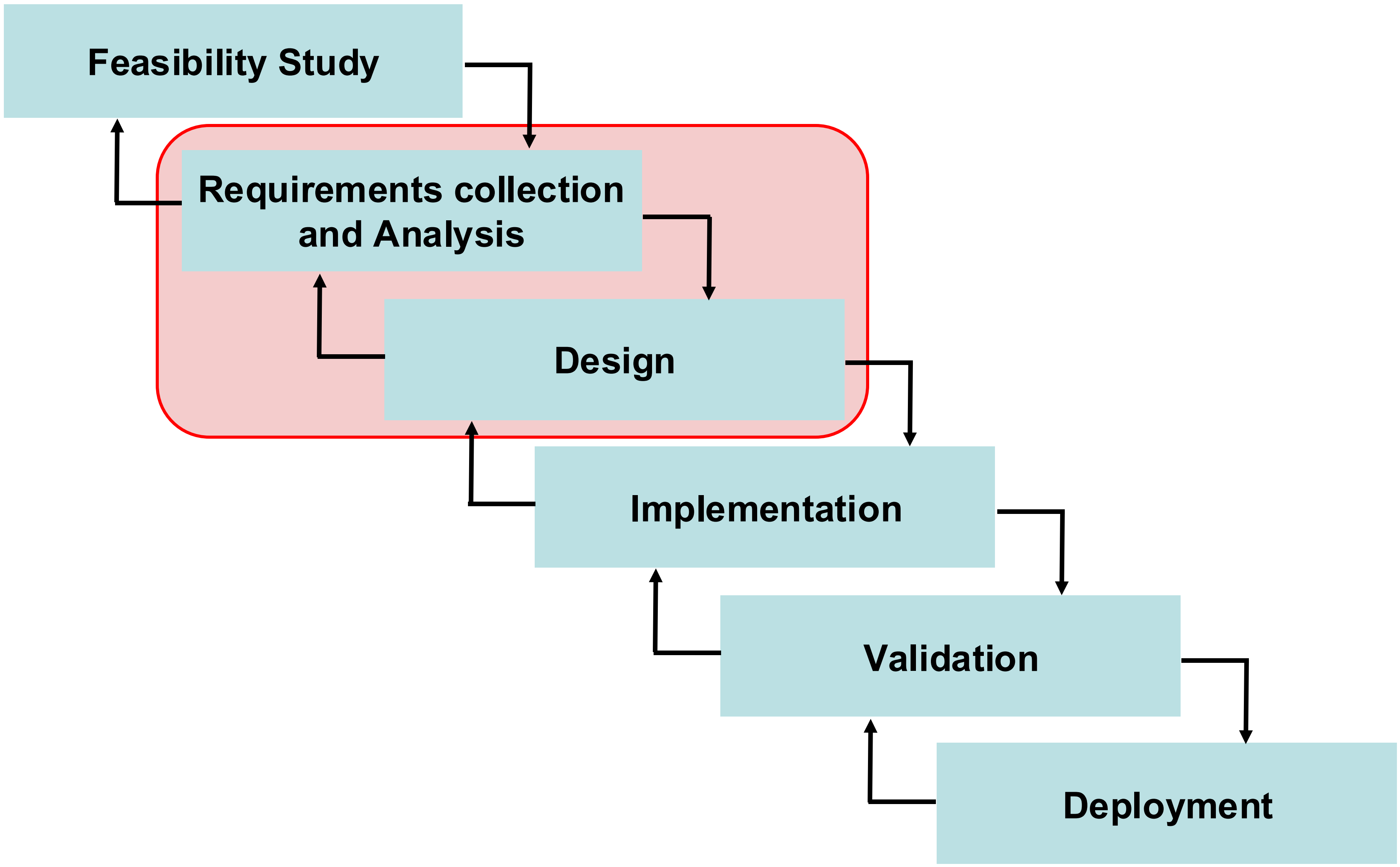
The design of an IS involves two aspects:

- **Data Design**
- **Application Program Design**

But:

- Data have a key role
  - They are more stable

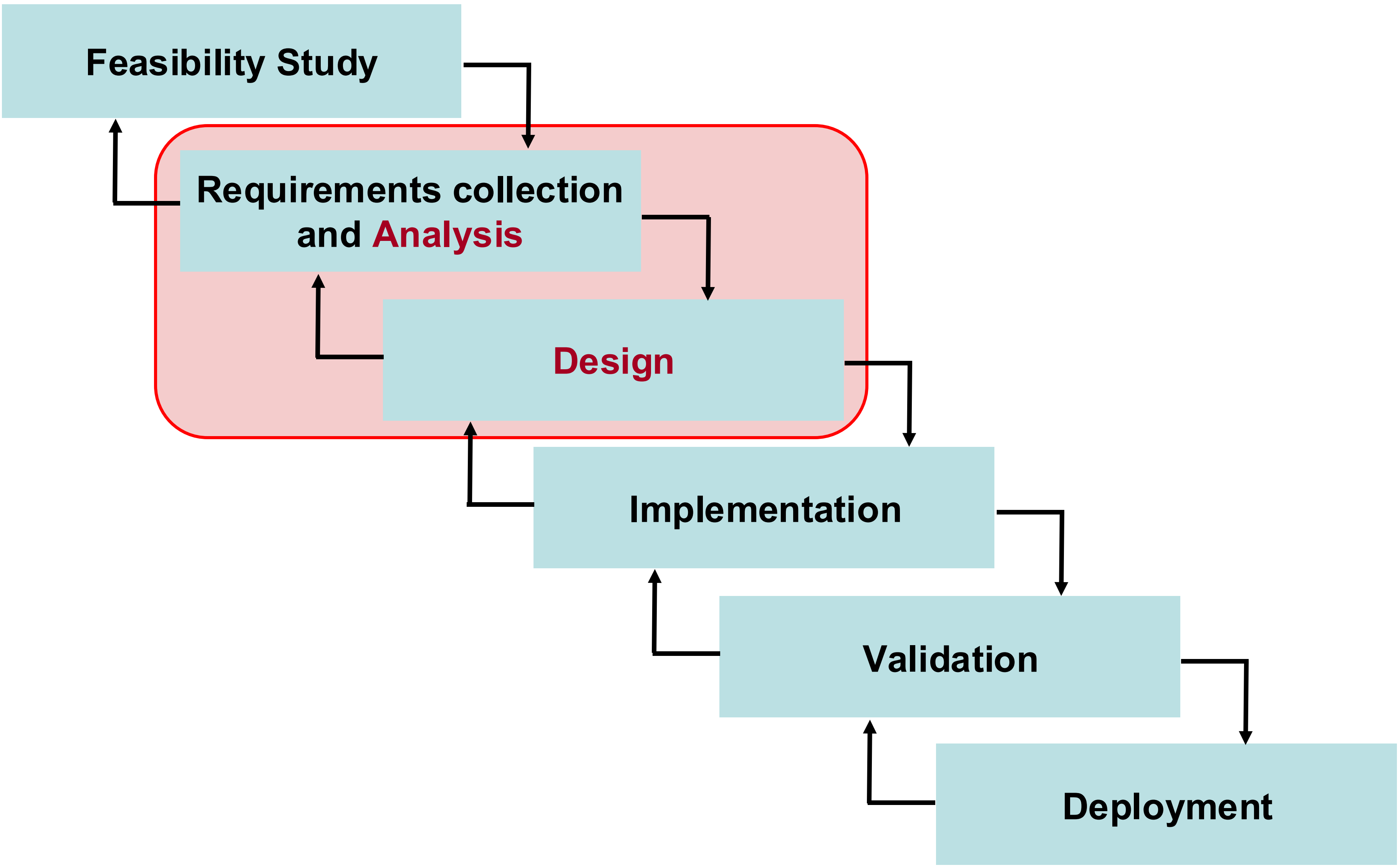
Requirements & Design



Design Methodology

- Good quality projects systematically follow a design methodology, consisting of:
  - Define separate tasks
  - Selection criteria
  - Representation Models
  - General solutions and User-friendliness

Requirements & Design



Requirements & Design: Details

DB requirements

Conceptual Design

“WHAT?”: analysis

Conceptual Schema

Logical Design

Logical Schema

“HOW?”: design

Physical Design

Physical Schema

Data models: Conceptual, Logical & Physical

The result of each design phase corresponds to a scheme that is used in the following phase:

- Conceptual Schema
- Logical Schema
- Physical Schema

Data Model

- A collection of components used to categorize relevant data and describe operations over them
- Crucial component: constructors
- Constructors play the same roles of data types definitions within programming languages
  - For example, the relational model defines the relation constructor for uniform sets of tuples (records)

Schemas and Instances

- For each Database there are:
  - a schema, time invariant, describing the data structure (*intensional aspect*)
    - relational data model: relations' attributes
  - an instance, the current values, that could change over time, also very quickly (*extensional aspect*)
    - relational data model: the set of uniform tuples

Two Kinds of Models

- **Logical models**: organize data within DBMS
  - data abstraction level used by software (*persistency frameworks*)
  - Independent from the Physical Design

  e.g.: **relational**, graph, hierarchy, object
- **Conceptual models**: allow the data representation independently of the system
  - They try to describe real world concepts
  - They are used in the preliminary phases of software design

  the most used is the **Entity-Relationship** model

Conceptual Models, Why? (1)

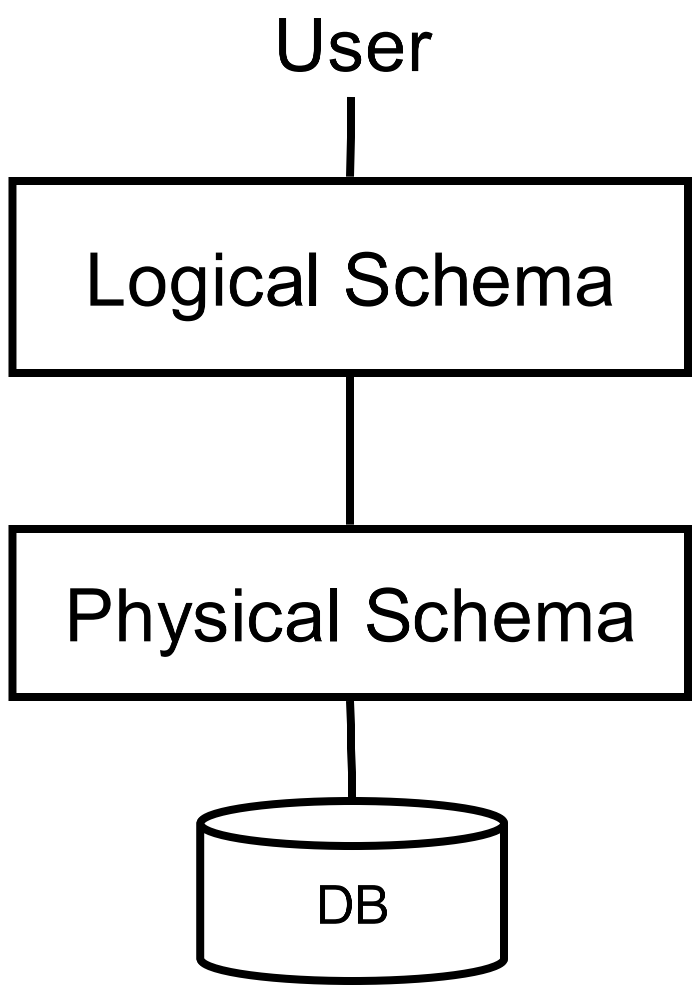
Let’s try to build a relational Database directly from the logical model:

- Where shall we start from?
- There’s a risk to go into any further detail.
- We need to define relations between the tables
- The relational model is too “rigid” for modelling

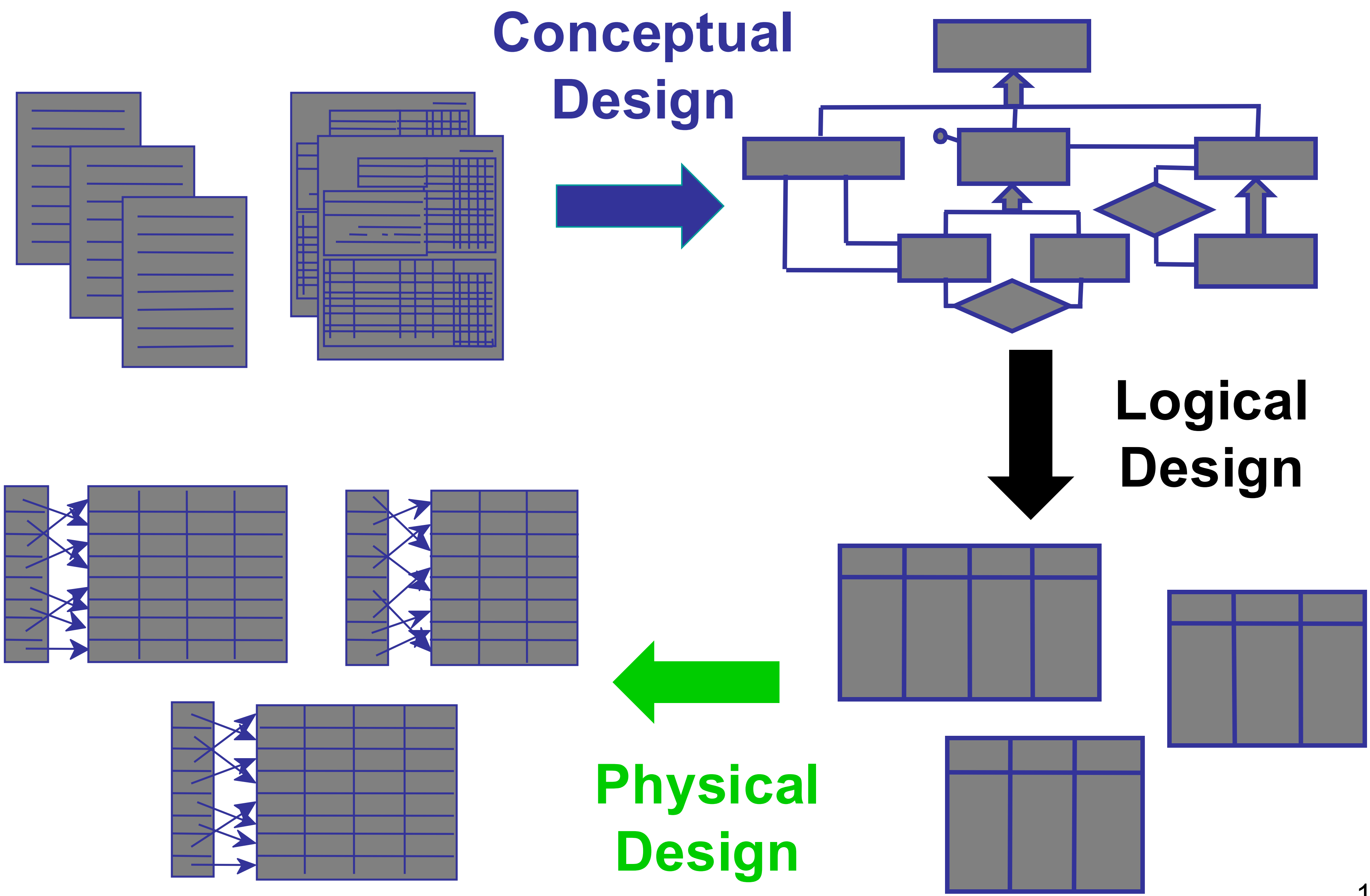
Conceptual Models, Why? (2)

- Allow to reason about reality of interest by defining a “model” which is independent of the implementation
- Allow to define object classes of interest and their relationships
- Provide efficient visual representations (useful for documentation purposes)

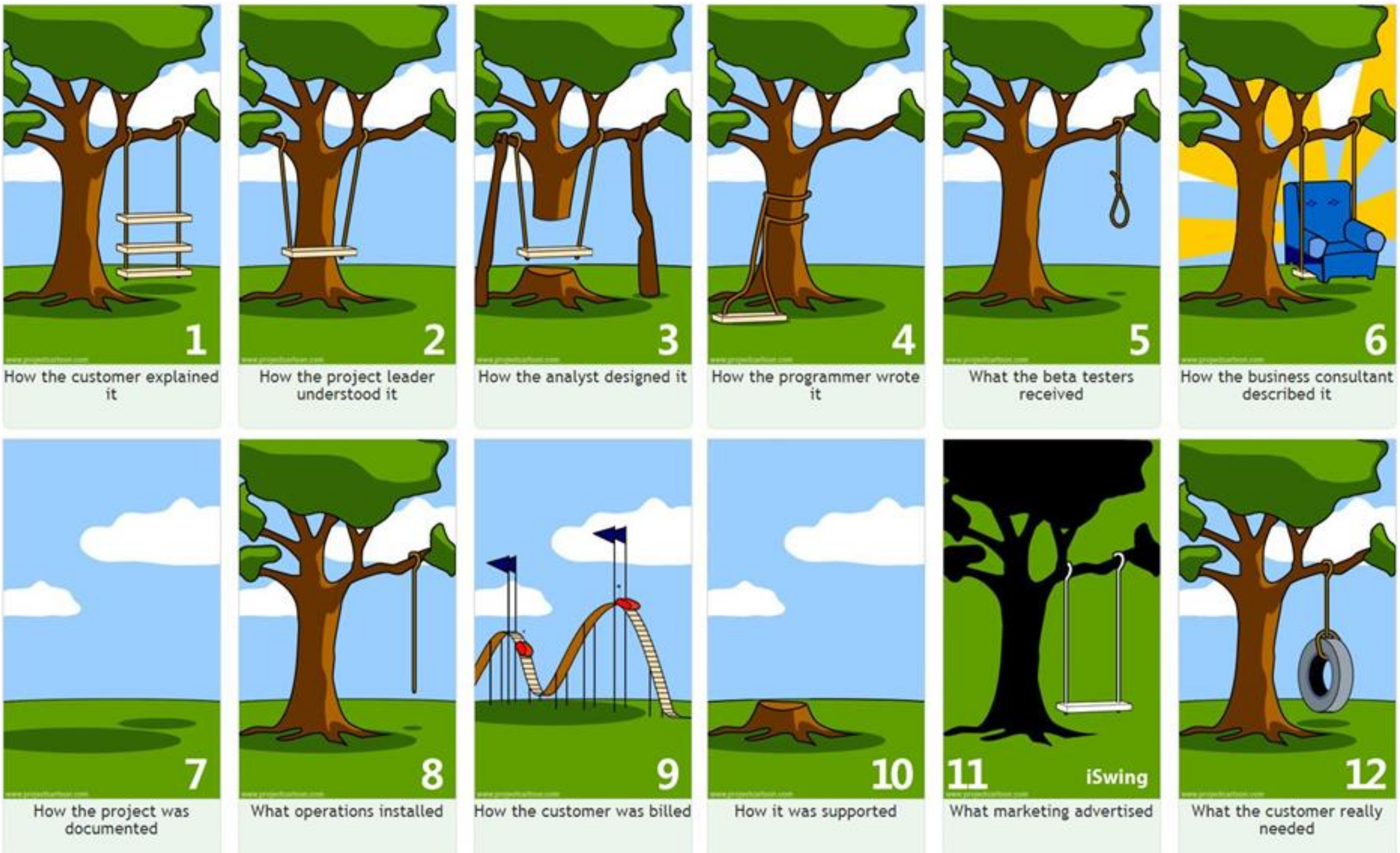
Coarse Grained DBMS Architecture



Design: Phases



Tree Swing Analogy since the 1970s



Entity-Relationship Model (ER)

- The most used conceptual data model
  - There are many roughly similar versions of it

ER Constructors

- Entity
- Relationship
- Attribute
- Key attribute
- Generalization
- ….

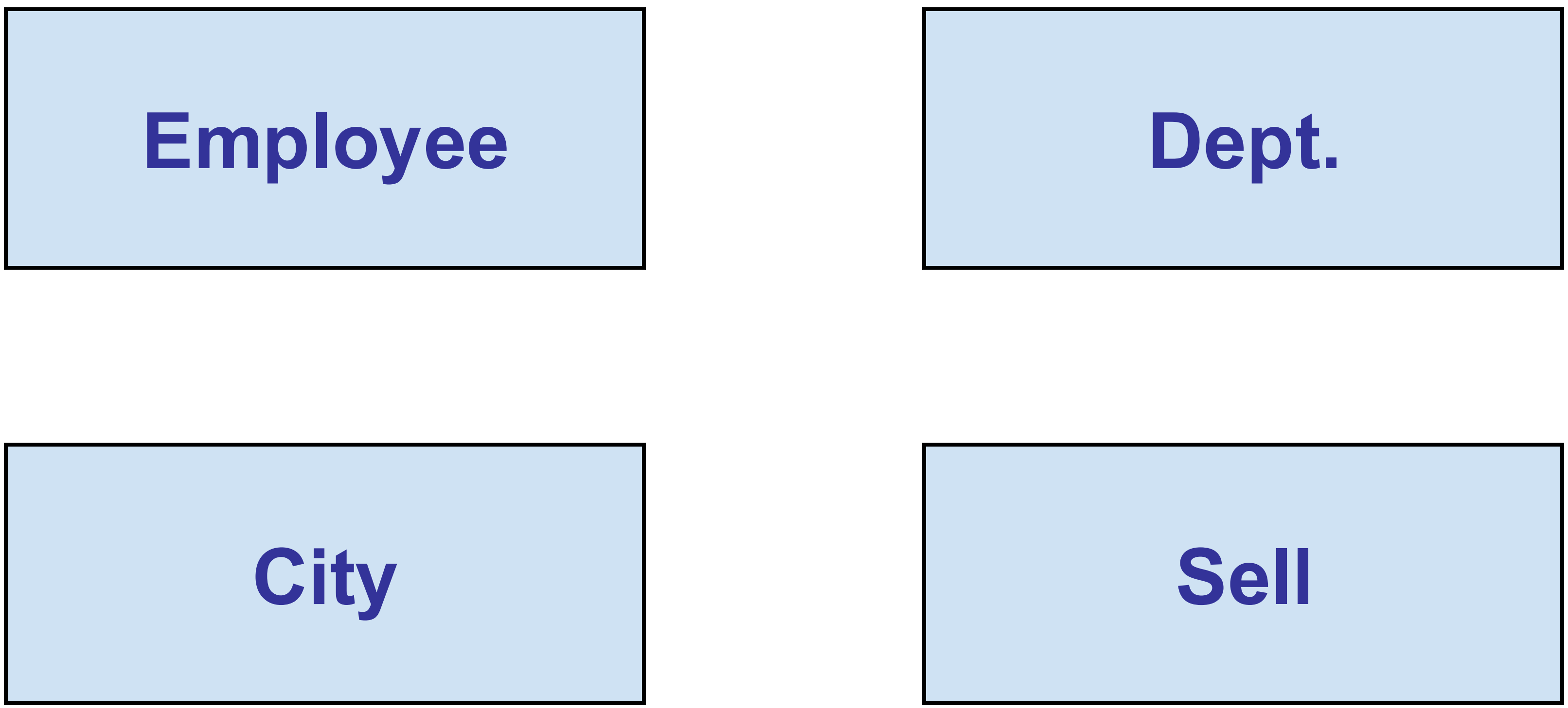
Entity

- **Class of “objects”** (things, people, places) belonging to reality of interest, sharing some common properties and having an autonomous existence
  - Example: Employee, City, BankAccount, SalesOrder, Bill

Entity: Schema and Instance

- **Schema**: uniform class of objects
- **Instance**: an element within the class (an instance, not the single data)
- ER schema entities do not represent each possible instance ("abstraction")

Entity: Visual Representation



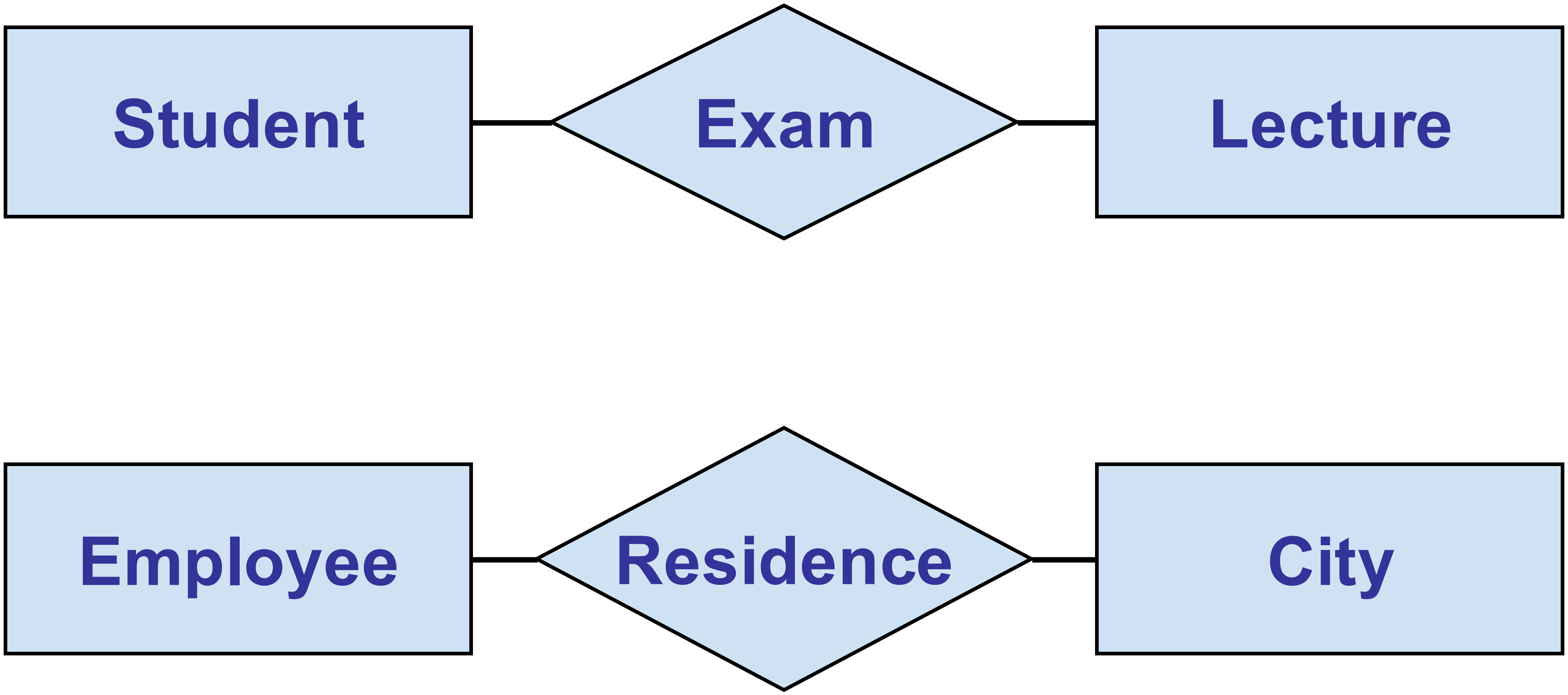
Entity: Comments

- Each entity has an unique name within the schema:
  - Meaningful names
  - Pragmatics: singular nouns

Relationship

- Establish **associations** between two or more entity types within the domain model
  - Example:
    - Residence (between People and City)
    - Exam (between Student and Lecture)
- It is also called association, correlation

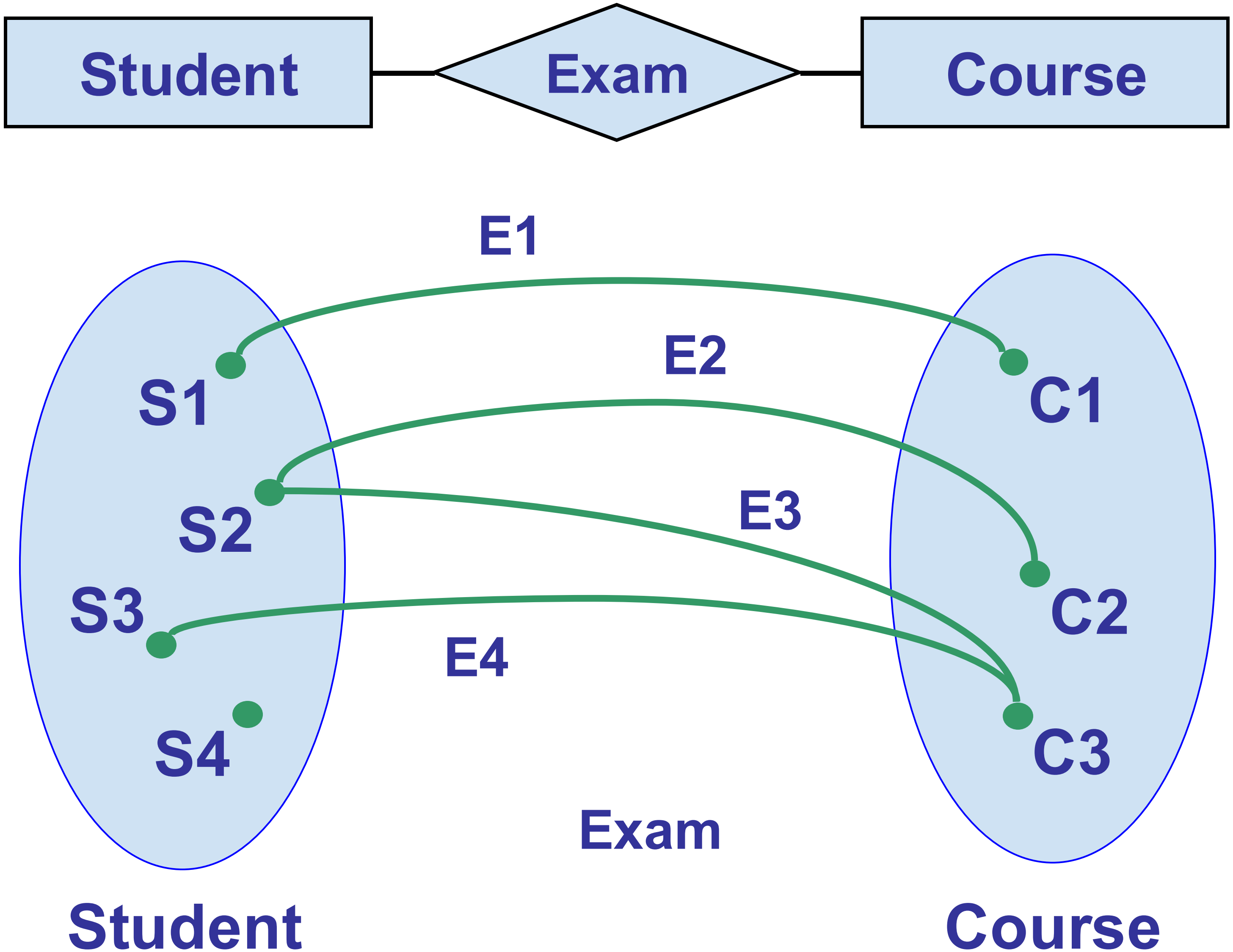
Relationship: Visual Representation



Relationship: Comments

- Each relationship has an unique name within the schema:
  - Meaningful names
  - Pragmatics:
    - singular nouns instead of verbs (if possible)

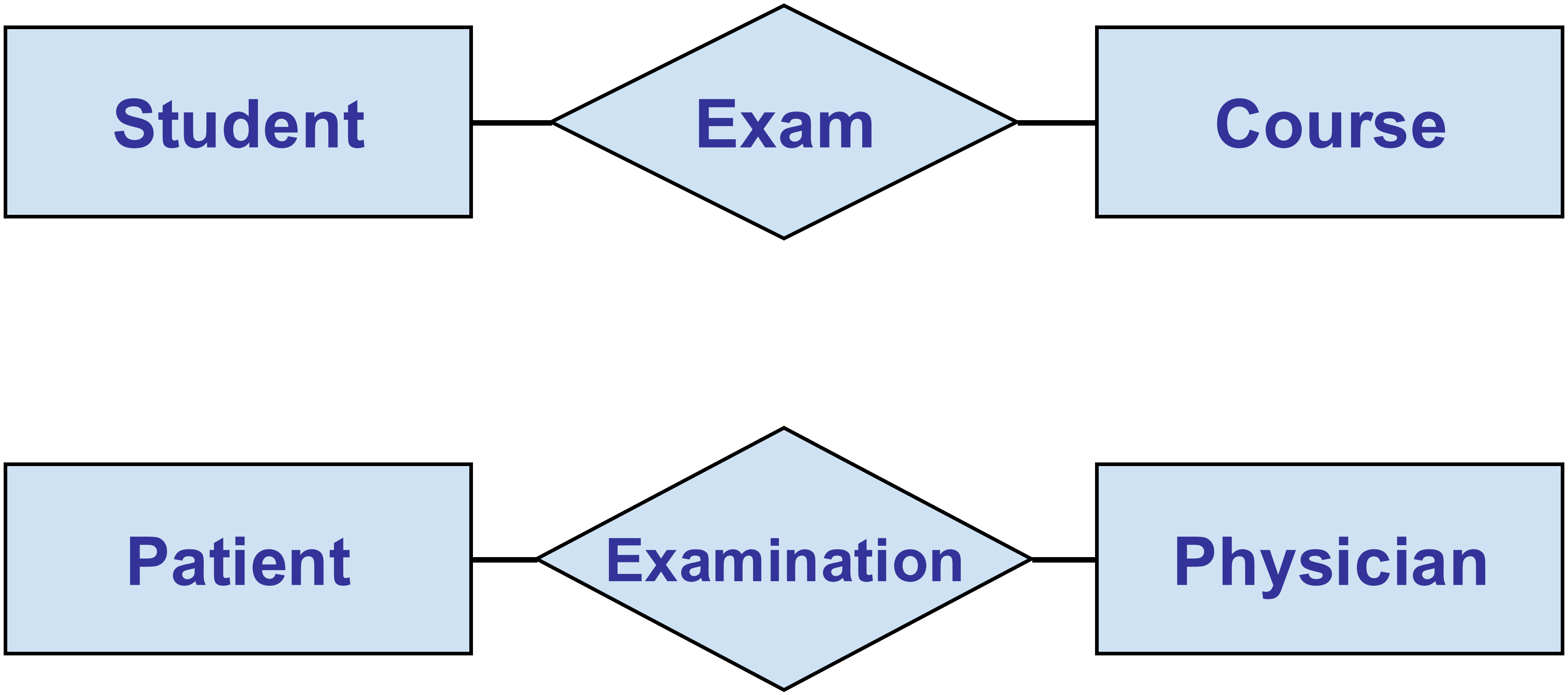
Schema vs Instance: an Example (1)



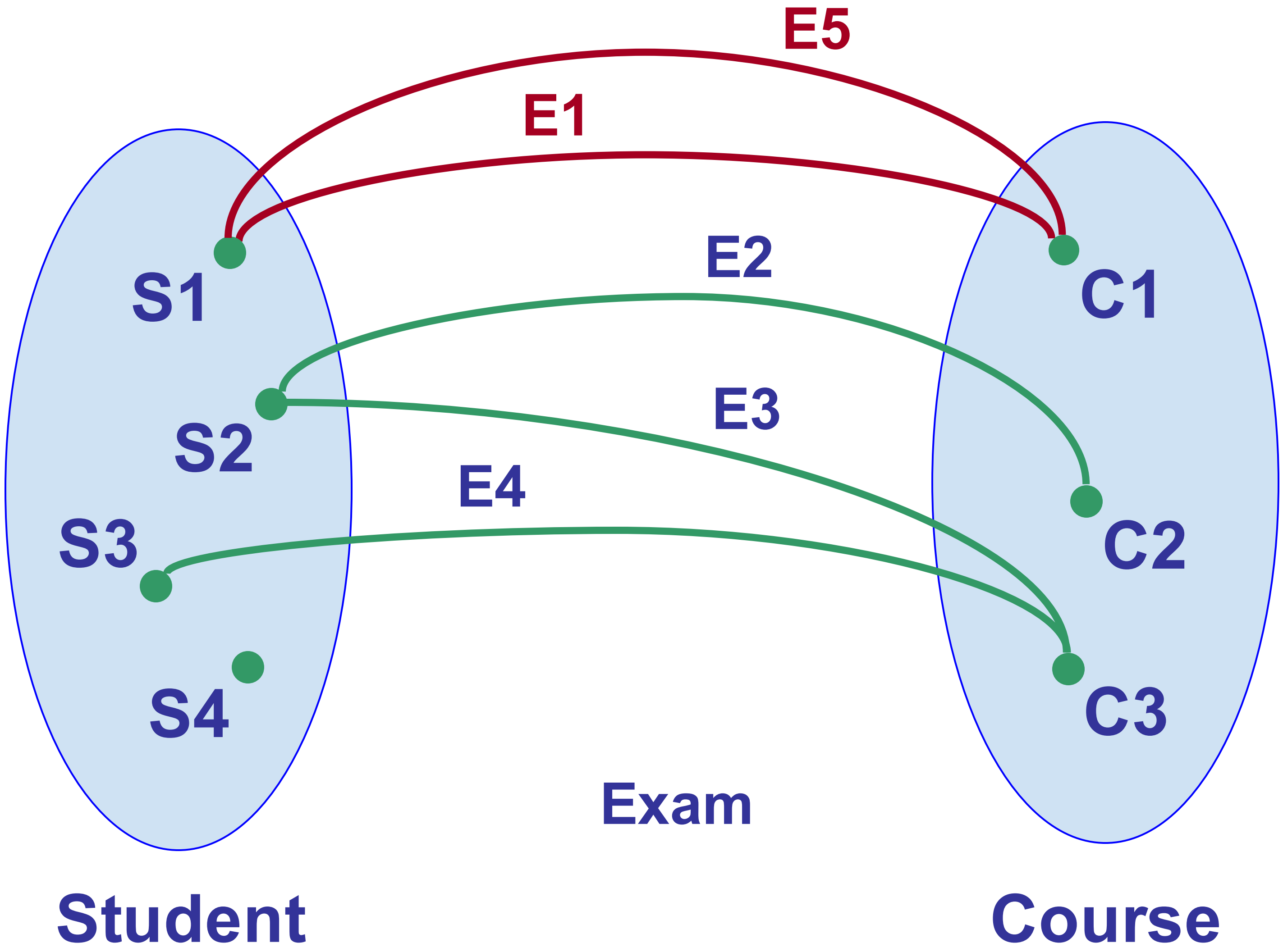
Types of Relationships

- A binary relationship instance is a pair of entity instances, one for each entity involved
- A n-ary relationship instance is a tuple of entity instances, one for each entity involved
- **There can be no repeated instances (pairs or tuple) within a Relationship**

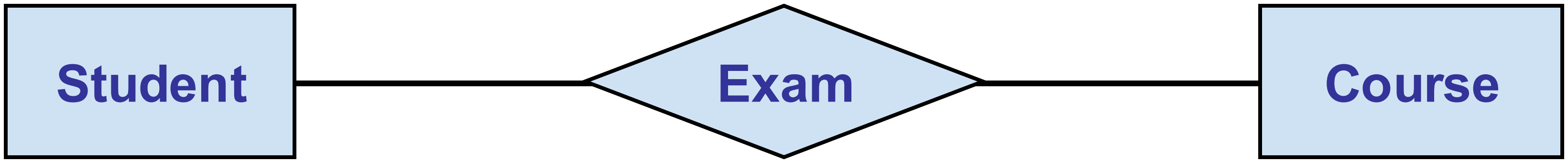
Are Those Relationships Correct?



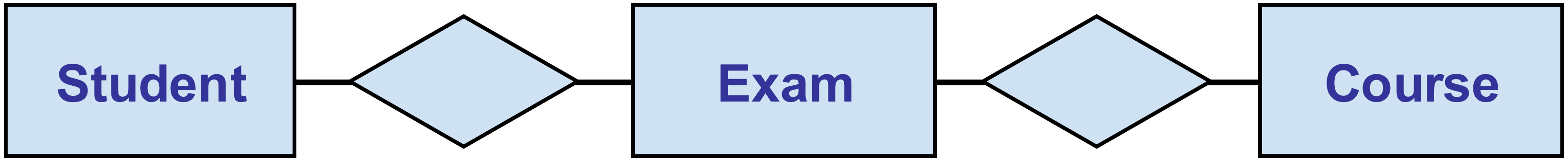
Warning!



Promoting Relationships to Entities

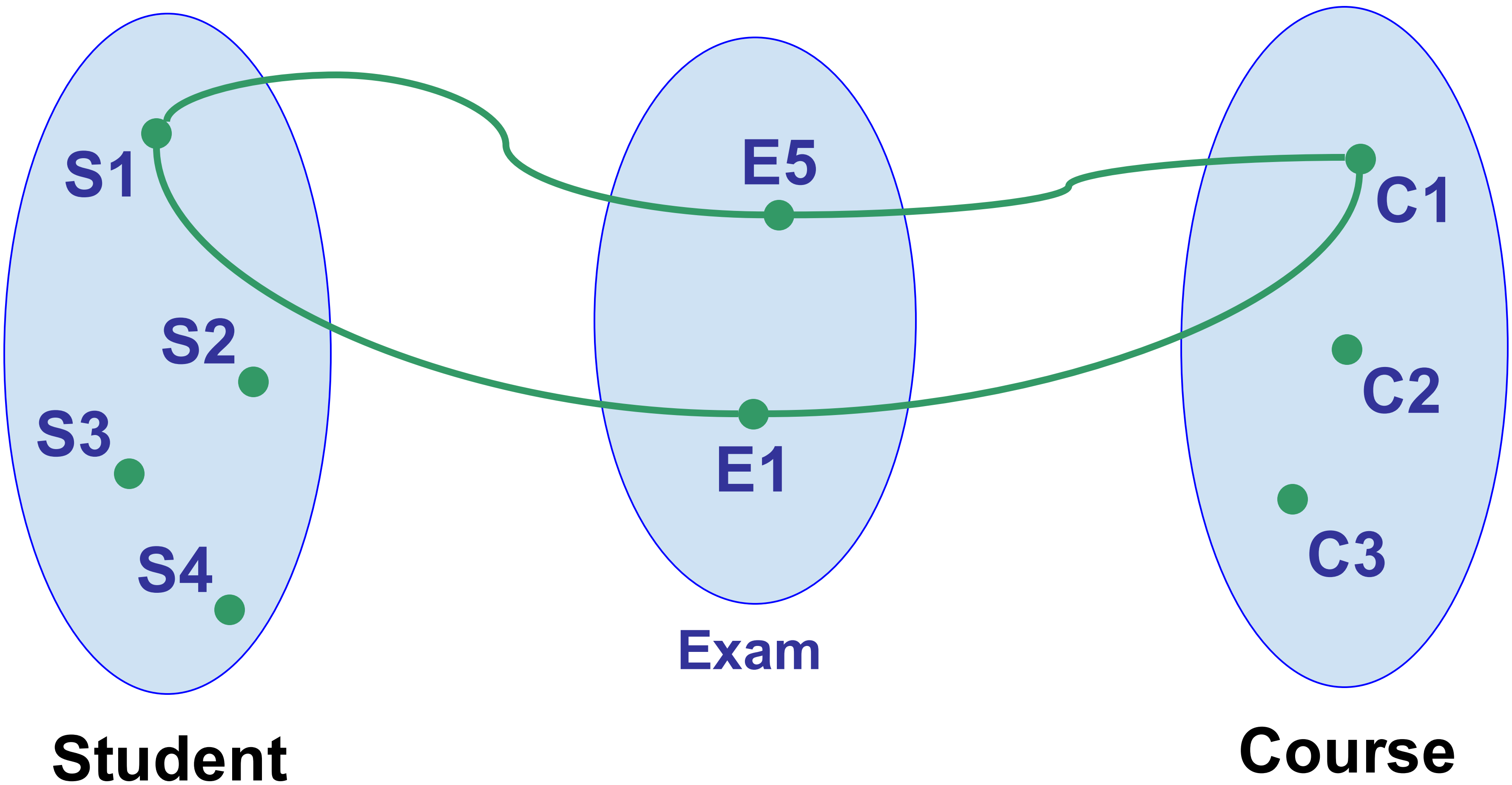


Exam does not capture the requirement "a Student took the same Exam more times for a given Course"

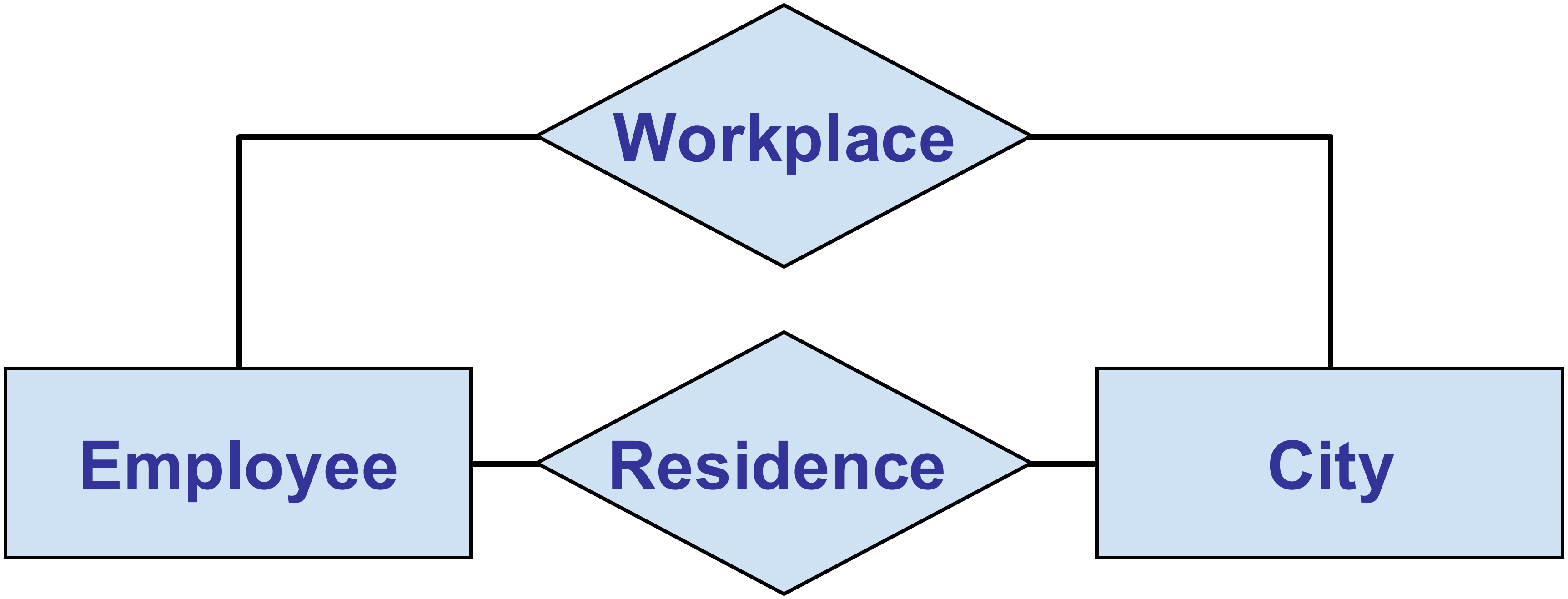


The solution is to represent Exam as an entity, connected by two relationship, one to Student and the other to Course

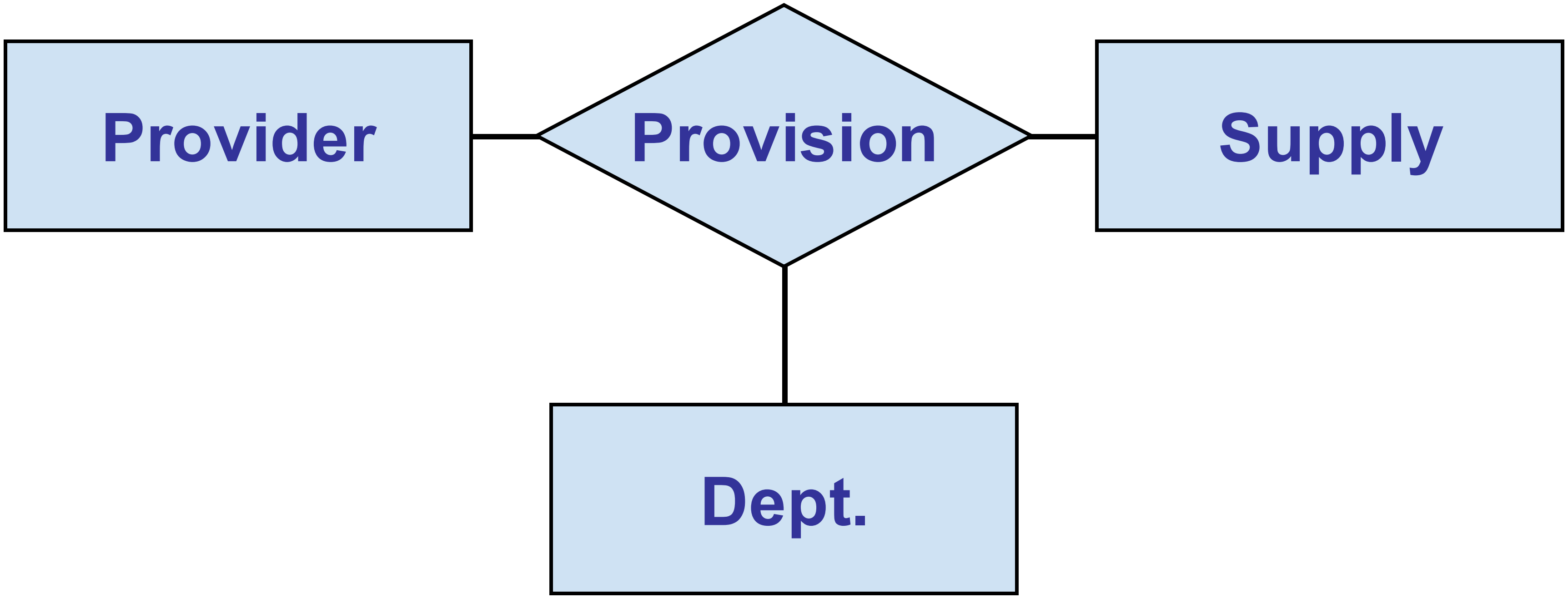
Exam as an Entity



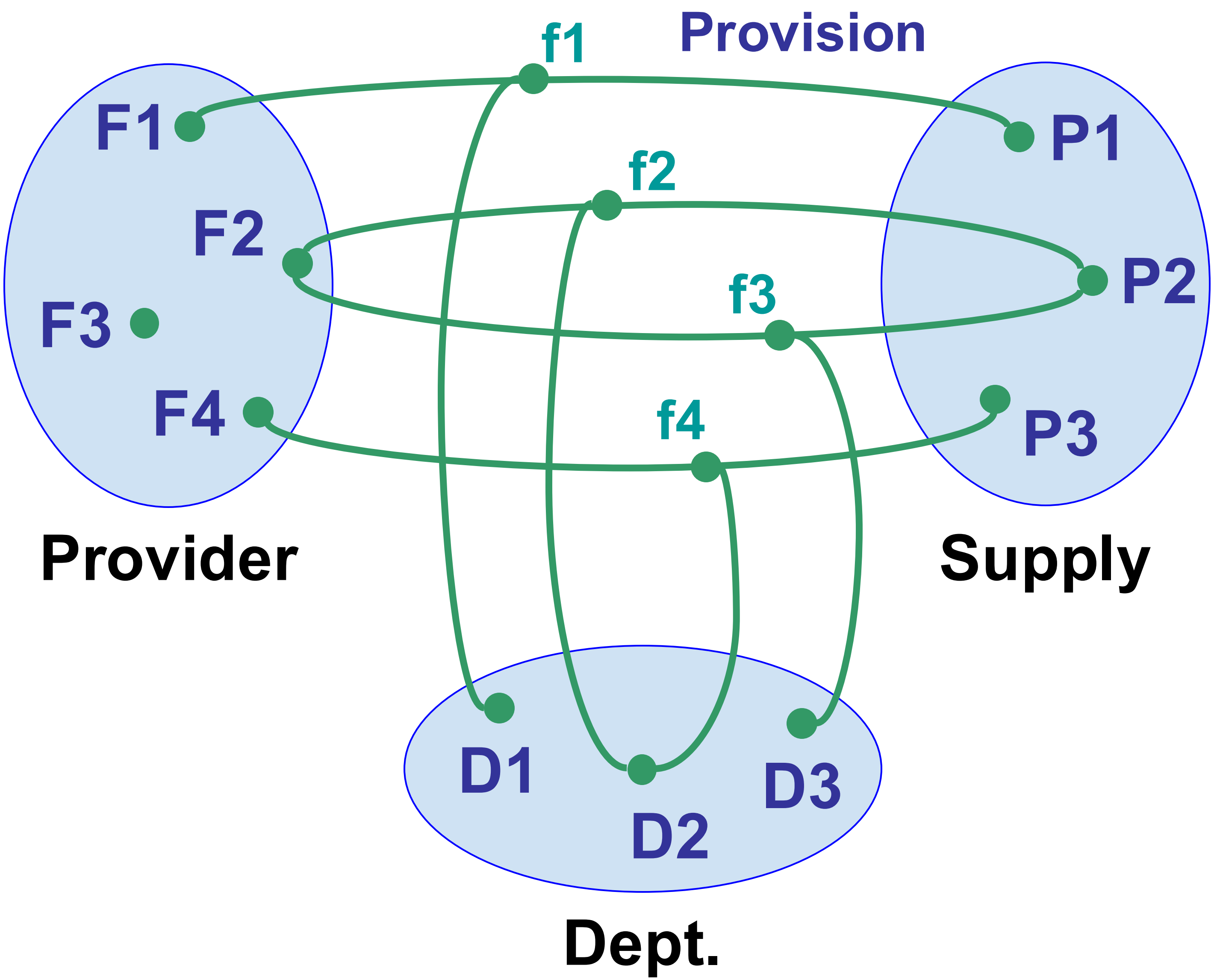
Two Relationships Between the Same two Entities



N-ary Relationship

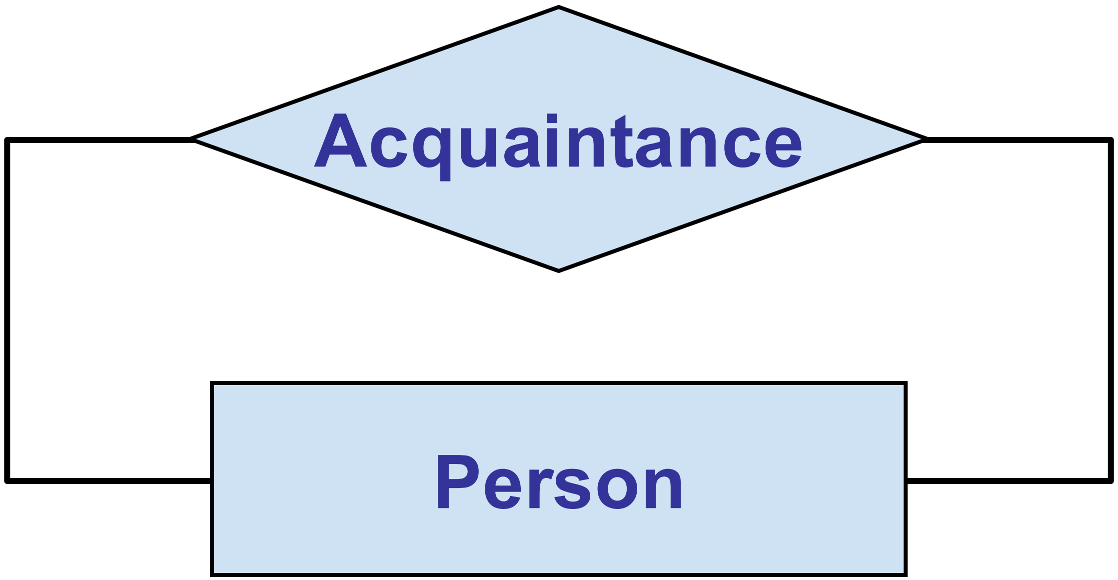


Example of Instances (1)

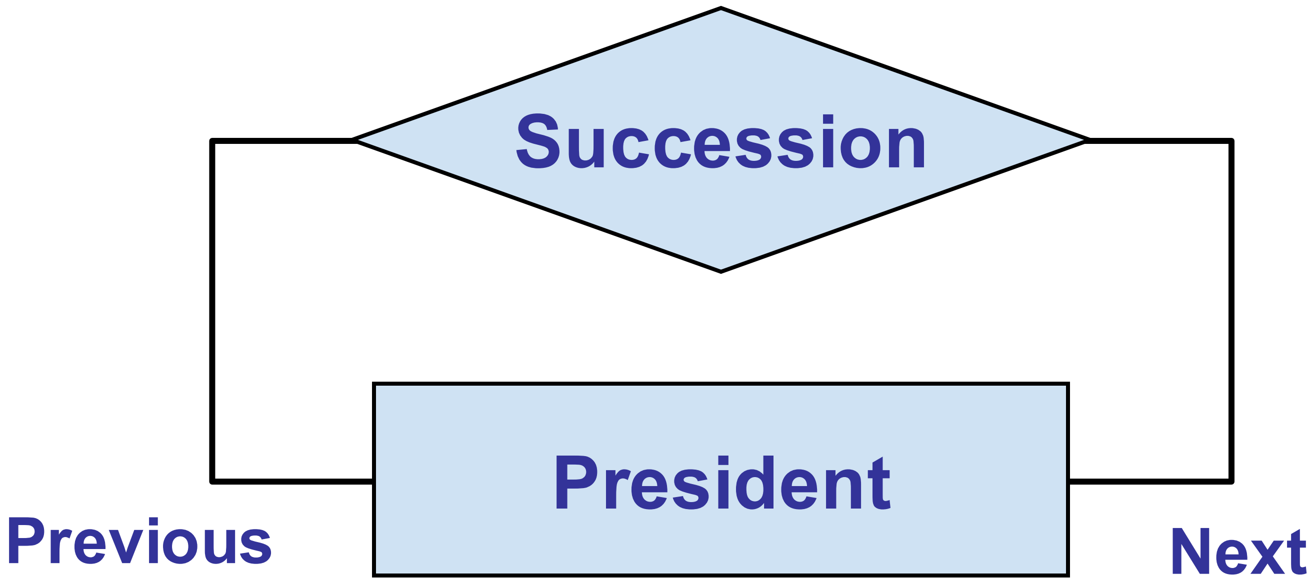


Recursive Relationship

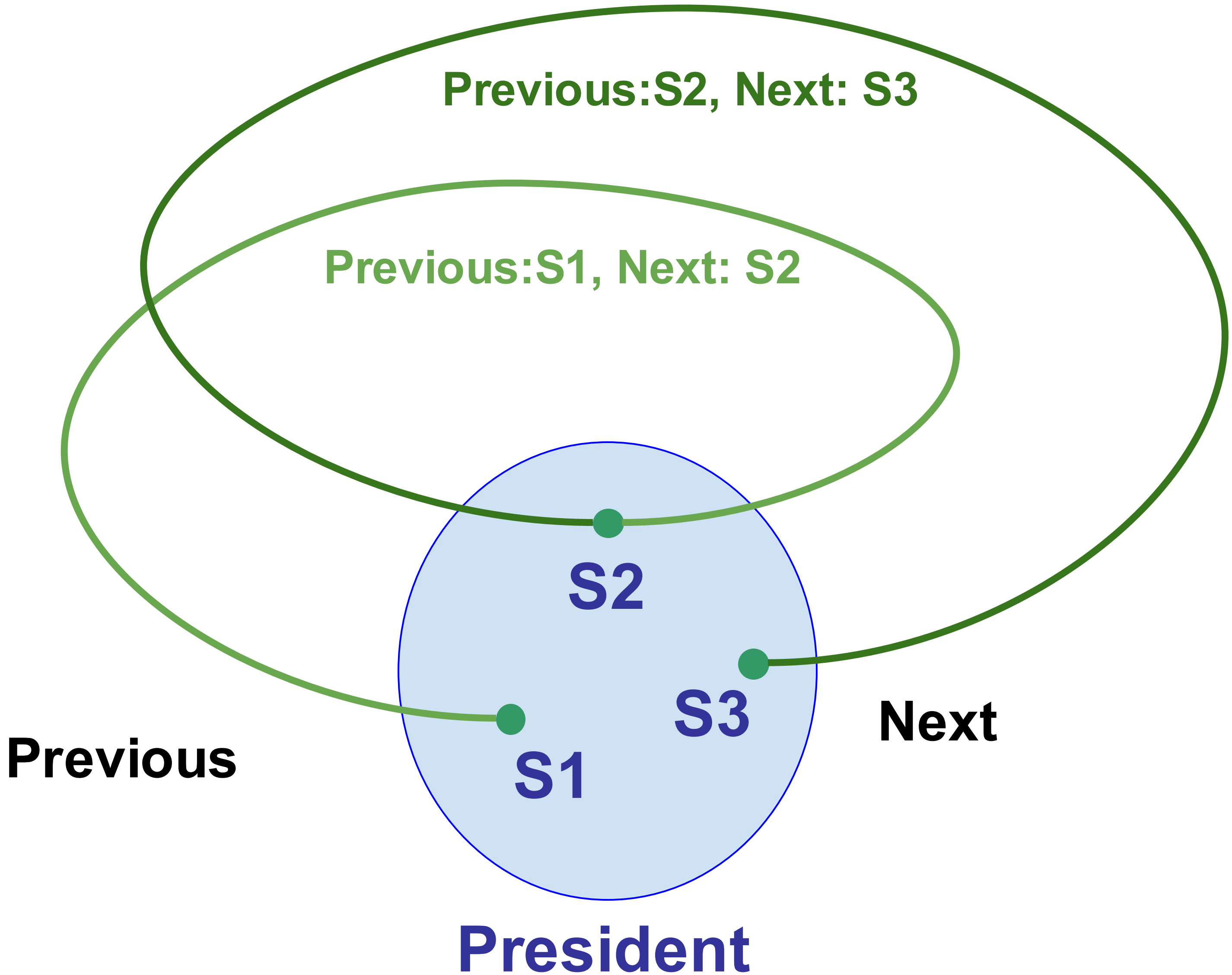
- It involves the same entity twice



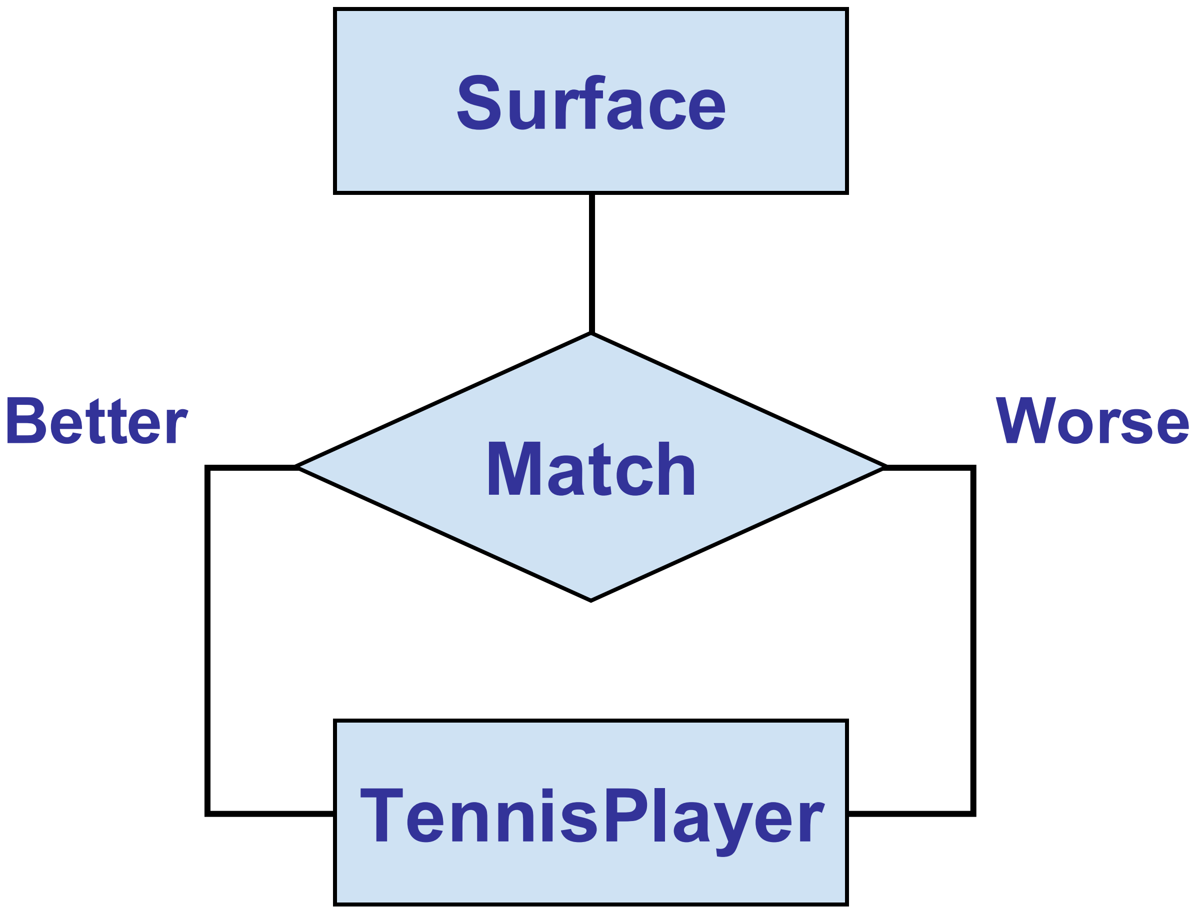
Relationship with “Role Names”



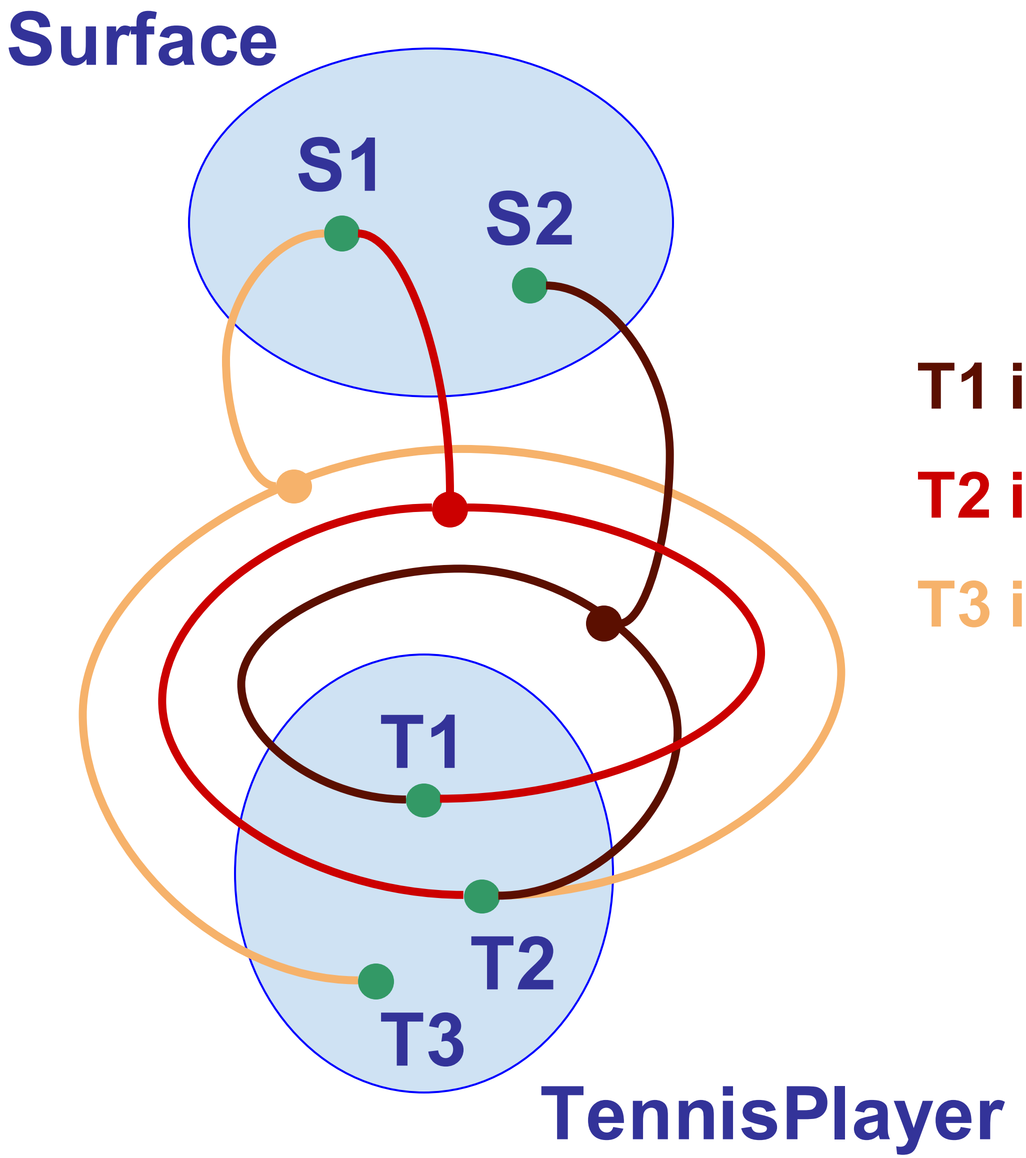
Example of Instances (2)



Trinary Recursive Relationship



Example of Instances (3)



T1 is better than T2 on S2

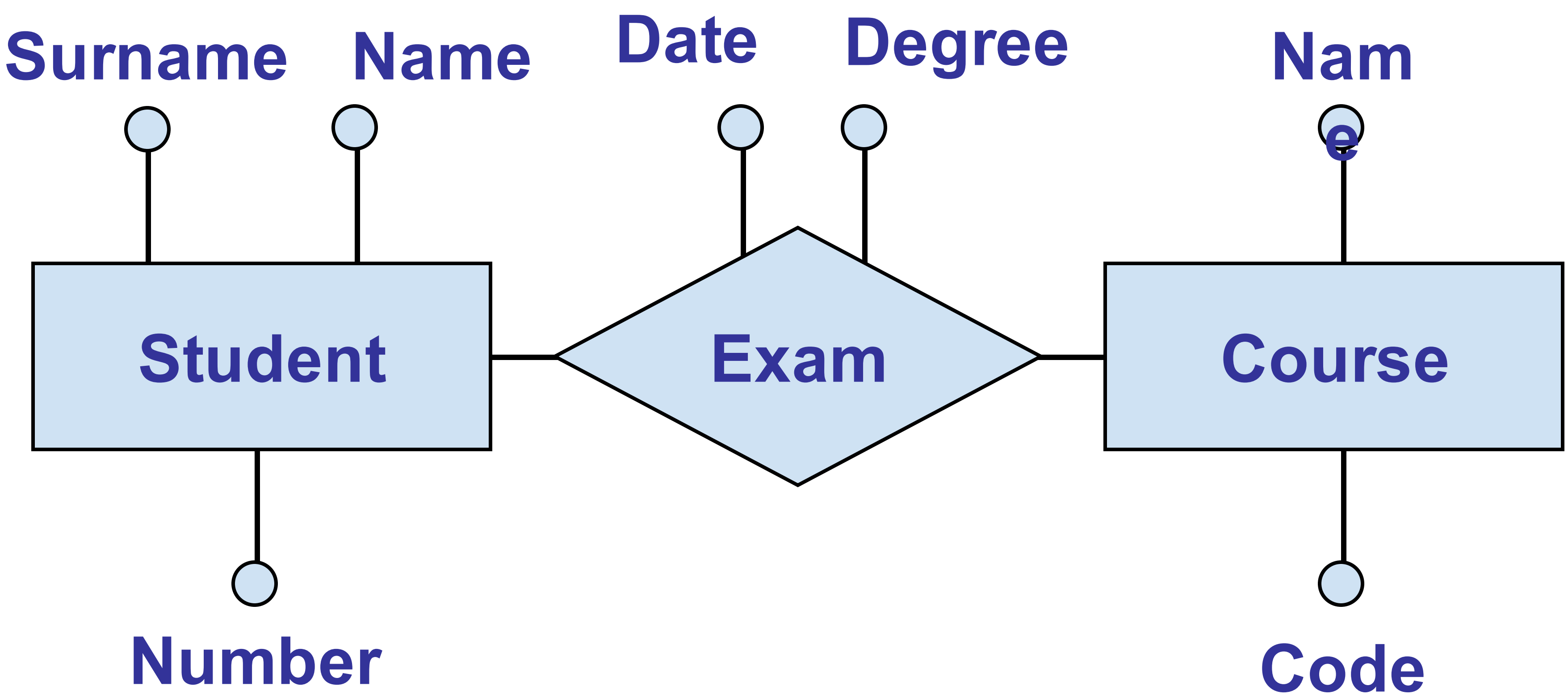
T2 is better than T1 on S1

T3 is better than T2 on S1

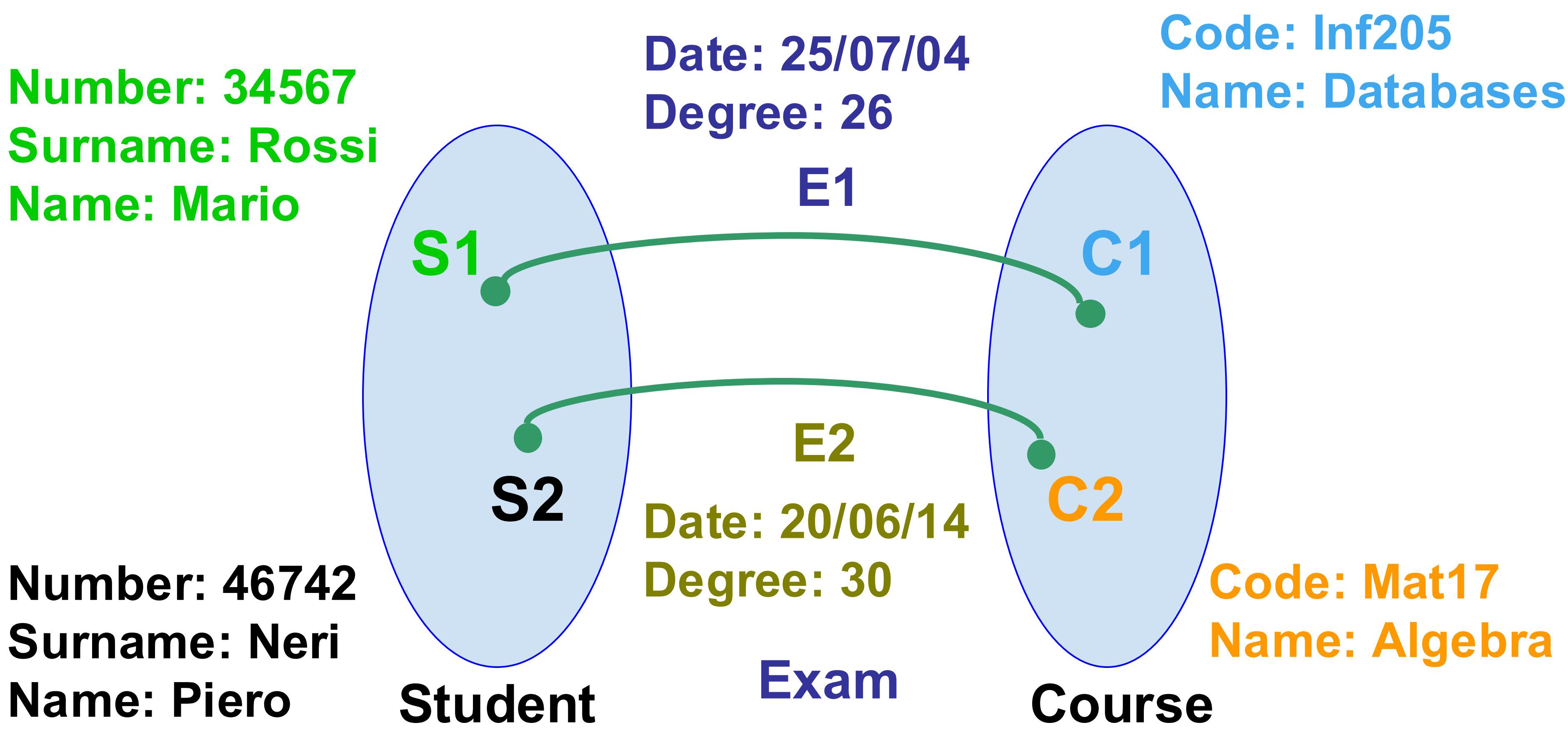
Attribute

- Attribute is a **property** belonging to either entities or relations
- Relates each entity or relationship to a value belonging to a set (value set) called domain

Attributes: Visual Representation



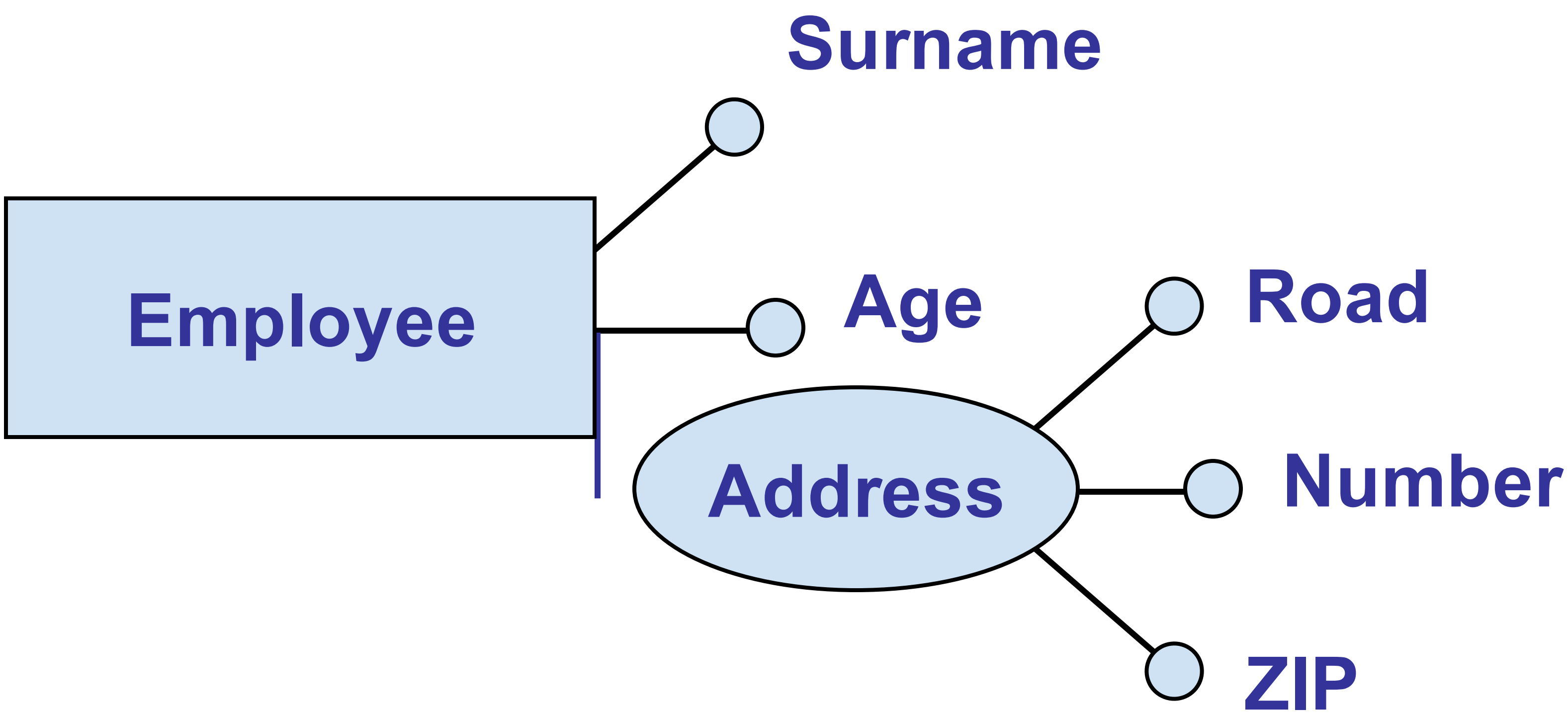
Example of Instances (4)



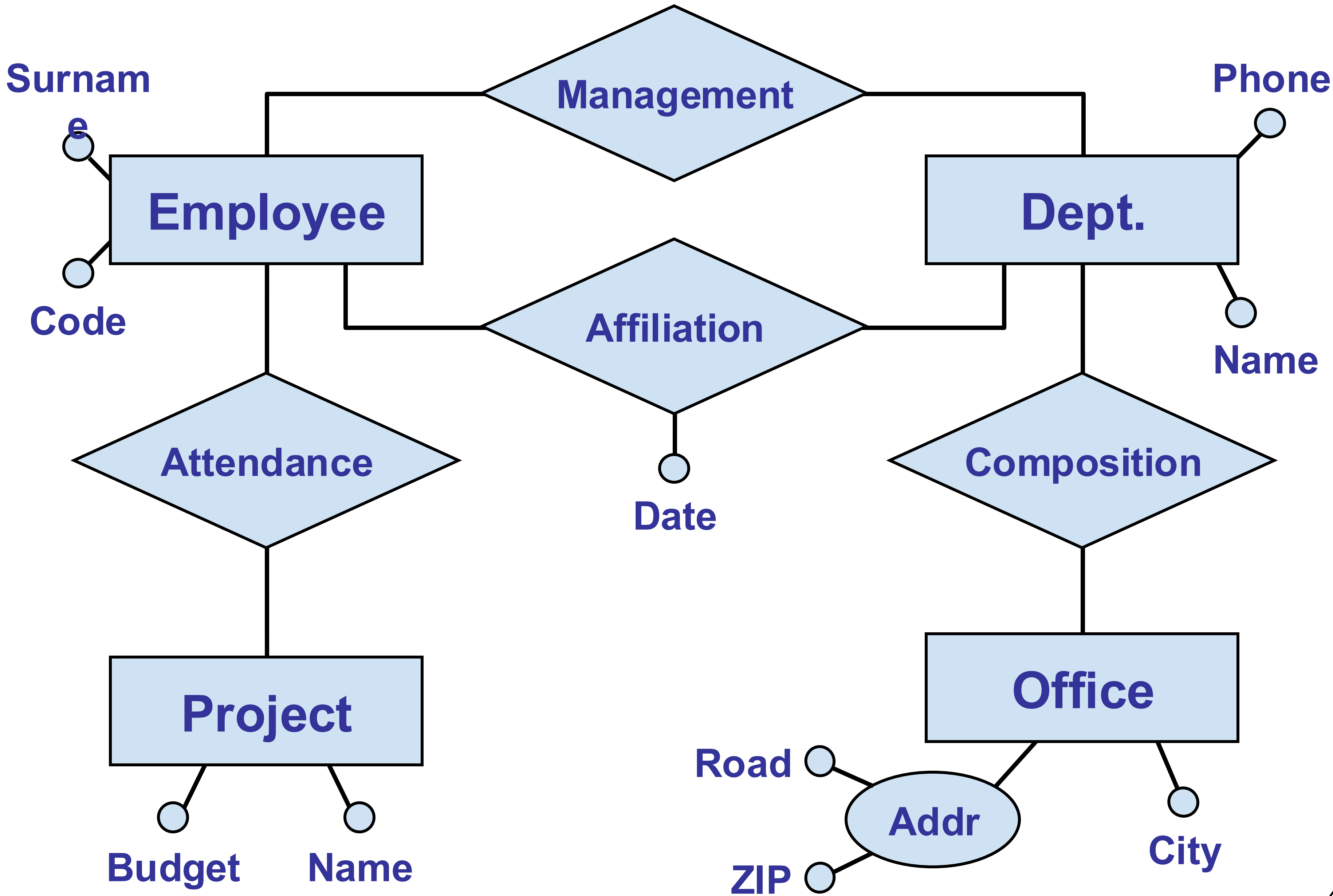
Composite Attributes

- Composite attributes group together attributes of the same entity or relationship which show similarities in their meaning or use
  - Example:
    - Road, Number and ZIP define Address

Graphical Representation



ER Schema: an Example (1)



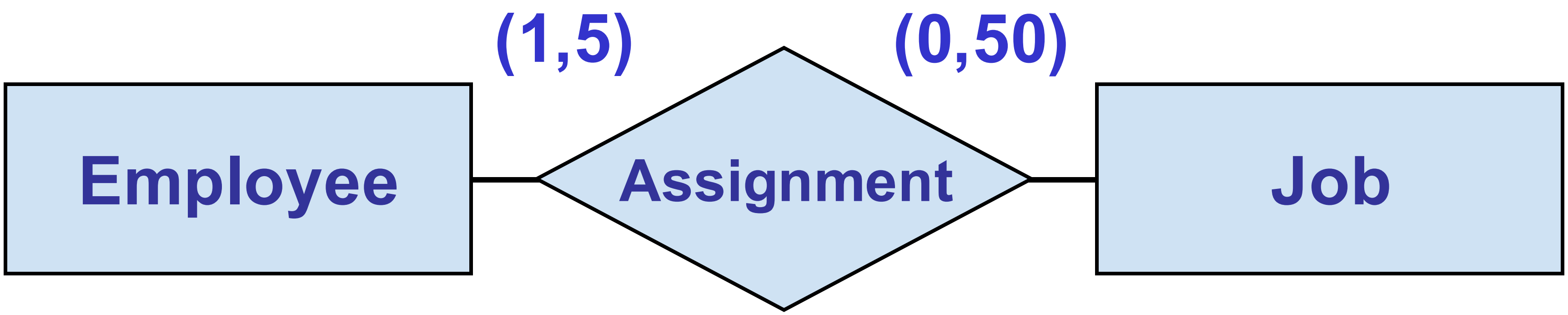
Other ER Constructors

- Cardinality
  - for relationships
  - for attributes
- Key
  - internal
  - external
- Generalization

Relationships’ Cardinality

- It is a pair of values related to each entity involved in the relationship
- Specify the minimum and the maximum number of occurrences of the relationships to which each entity instance can participate

Cardinality: an Example

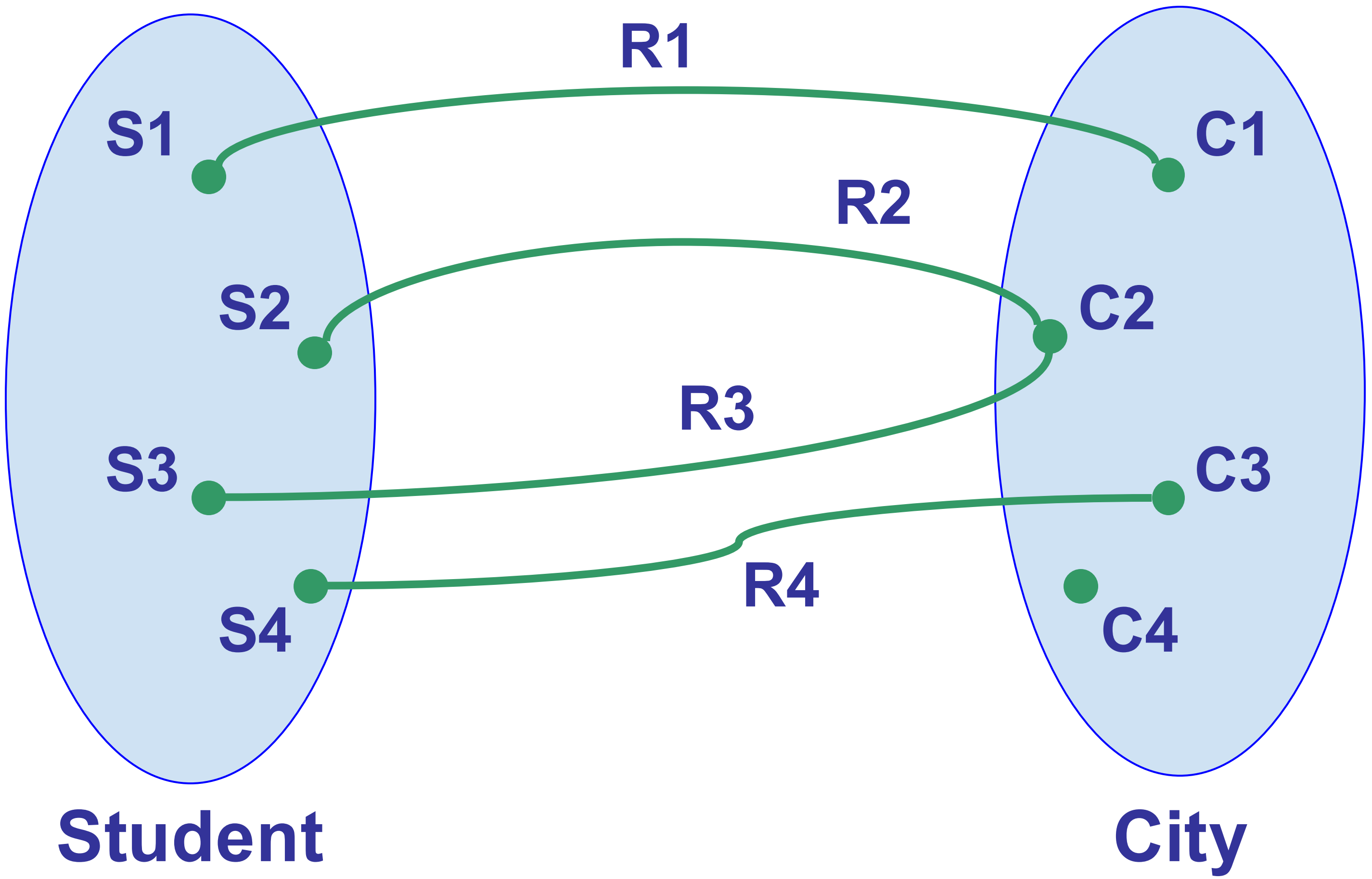


- An Employee has at least one job and has up to 5 Jobs
- Each Job could be assigned to at most 50 Employees but could be also assigned to no Employees

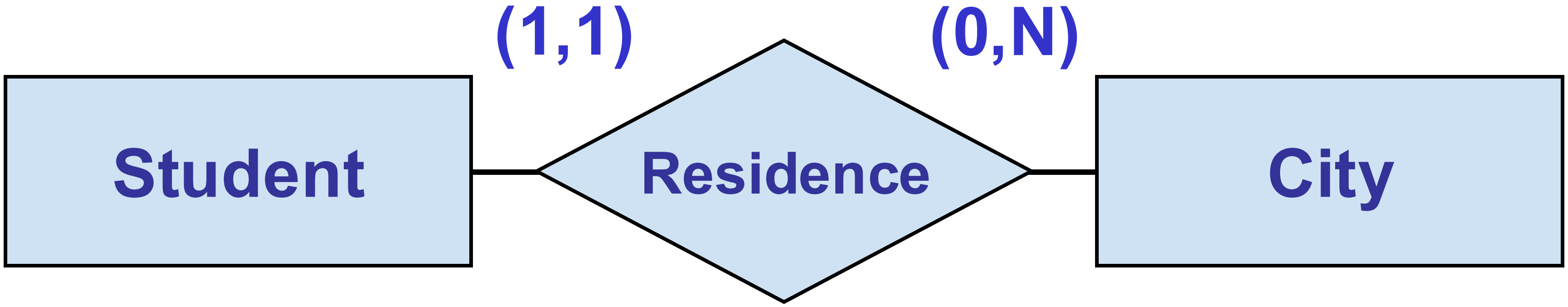
Cardinality

- For simplicity, we use three symbols only:
  - 0 and 1 for minimum cardinality:
    - 0 = "optional" participation
    - 1 = "mandatory" participation
  - 1 and "N" for maximum cardinality:
    - "N" does not impose any restriction

Example: Residence (1)



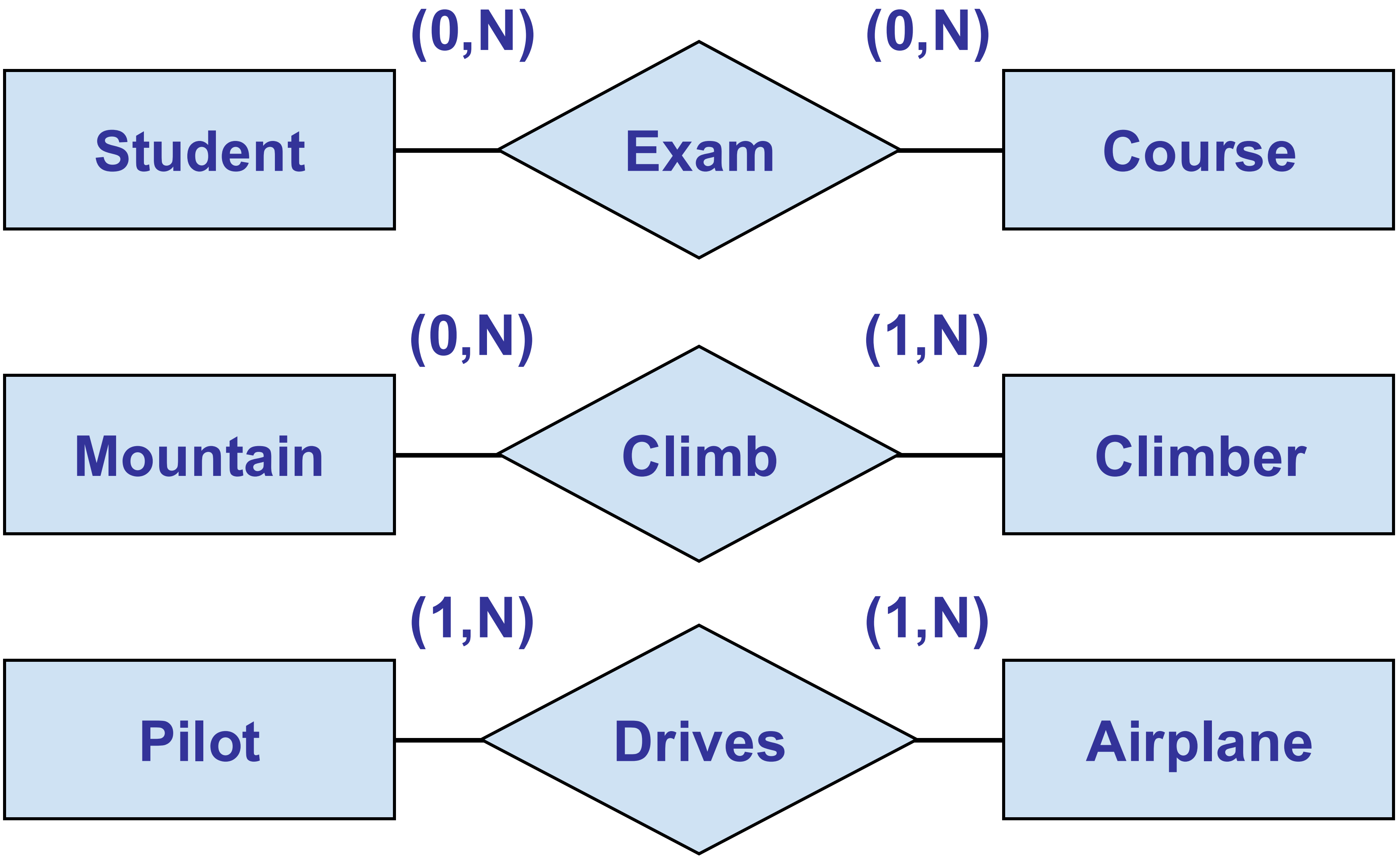
Example: Residence (2)



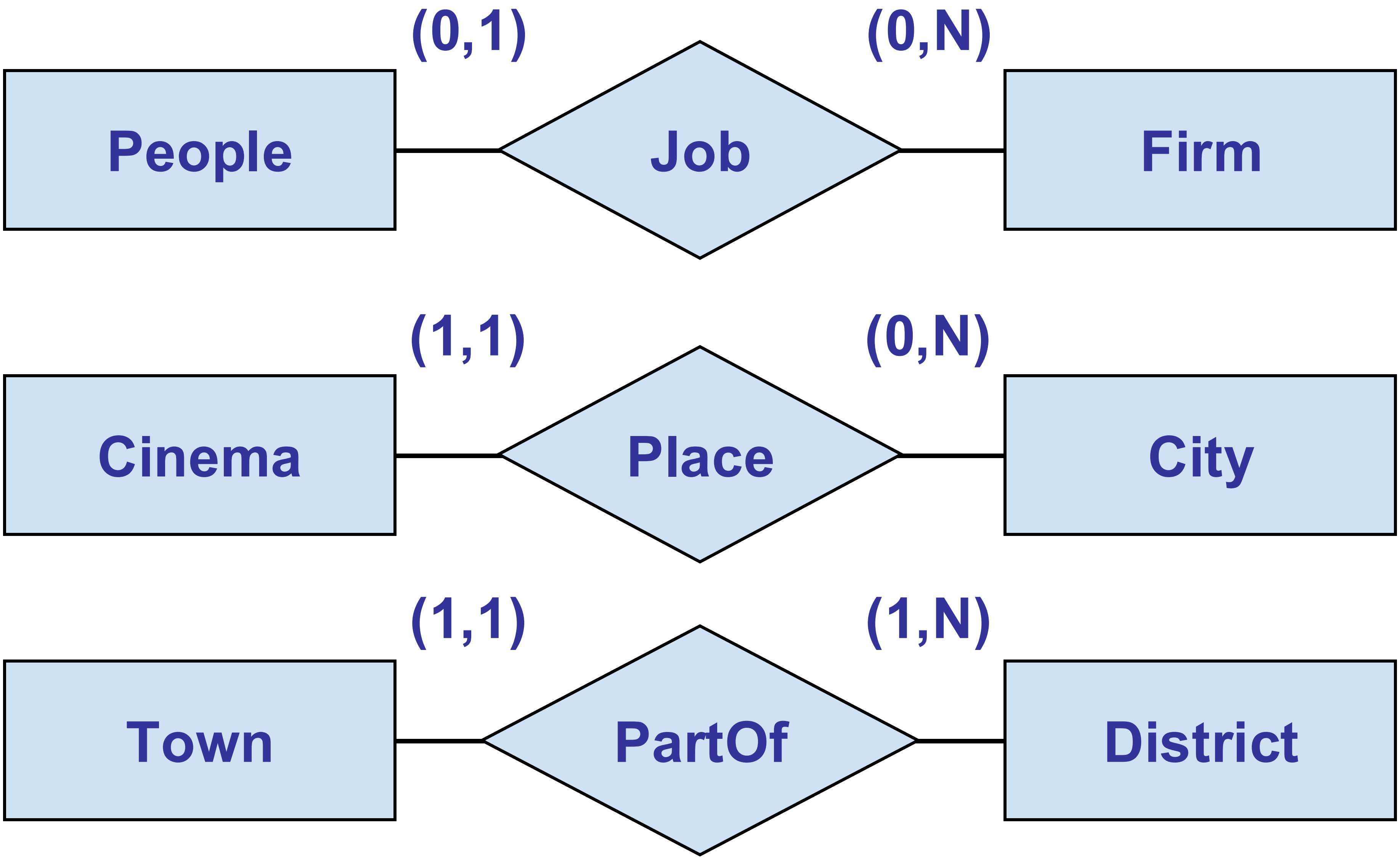
Types of Relationships

- With respect to maximum cardinalities, we have:
  - One-to-One
  - One-to-Many
  - Many-to-Many

Relationship “Many-to-Many”



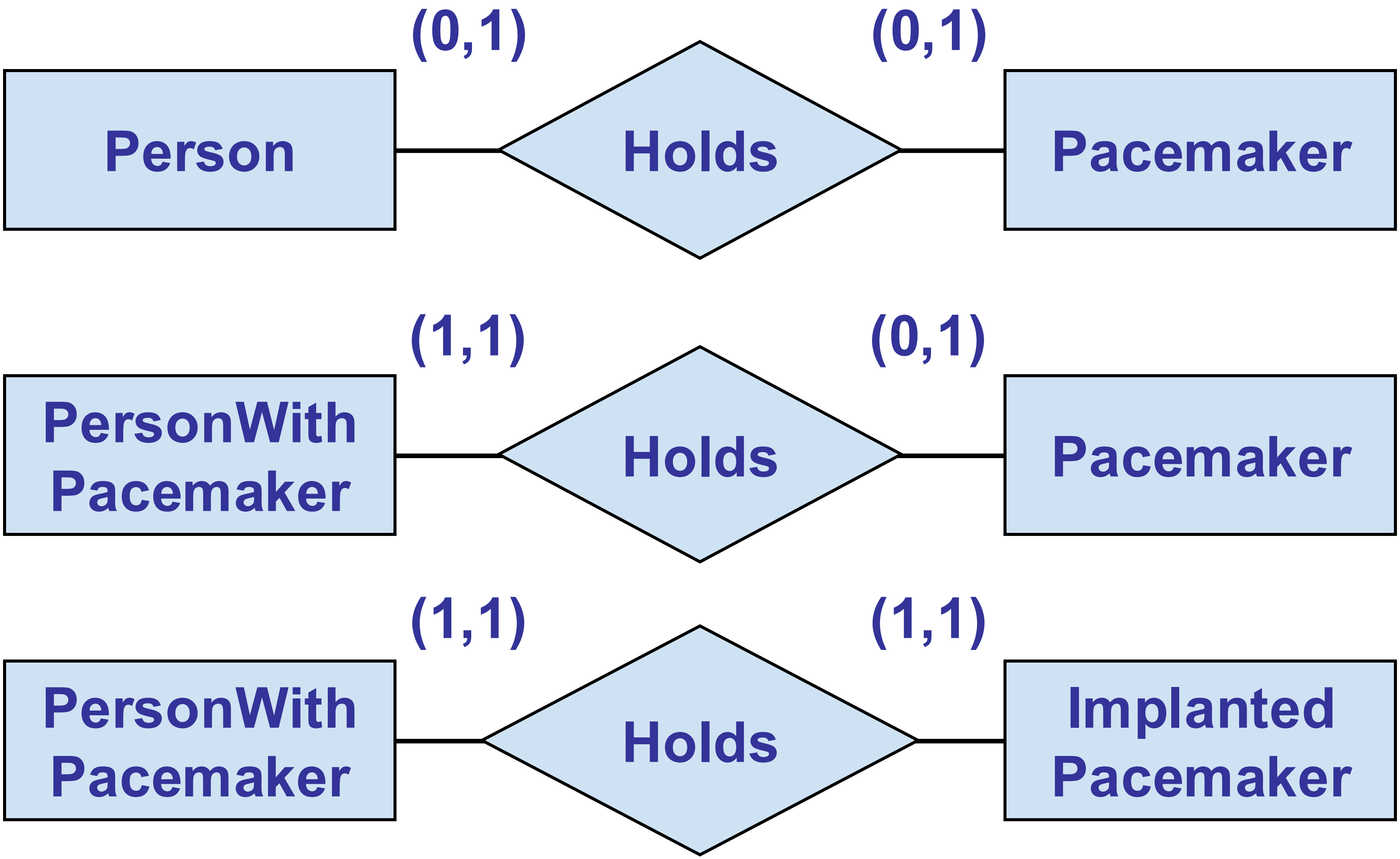
Relationship “One-to-Many”



Some Warnings

- Be careful to the “One-to-Many” relationships’ direction
- Mandatory-to-Mandatory relationships are very infrequent

Relationship “One-to-One”

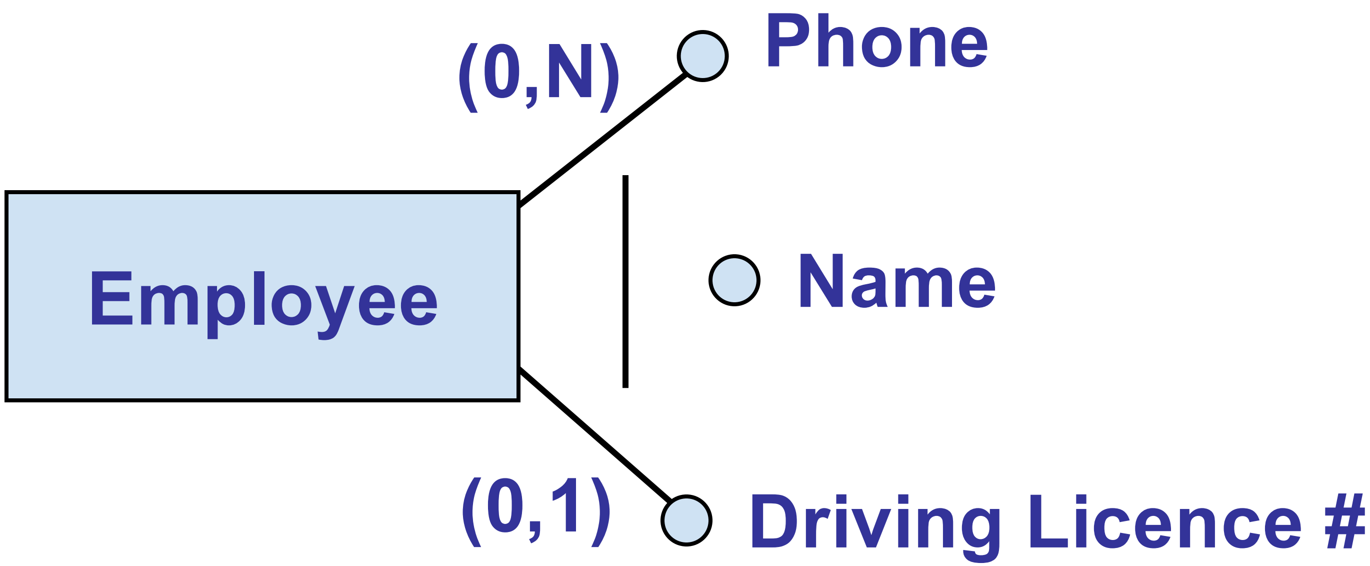


Attributes' Cardinality

It is possible to relate cardinalities also to attributes, with two purposes:

- indicate **optionality** (partial information)
- indicate **multivalued attributes**

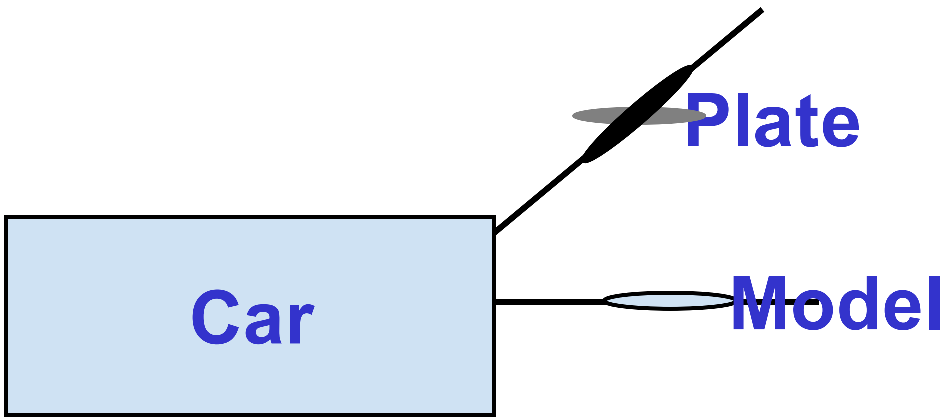
Graphical Representation



Entity Identifier

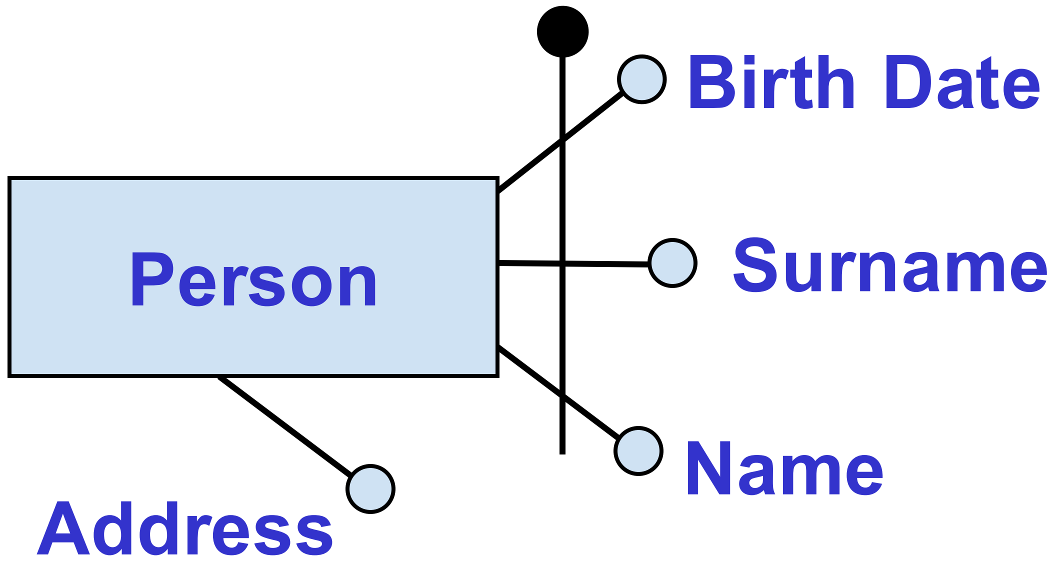
- A way to unique identify the occurrences of an entity
- Formed by:
  - Entity's attributes (internal identifier)
  - (attributes +) external entities reachable through relationships (external identifier)

Internal Identifiers (1)



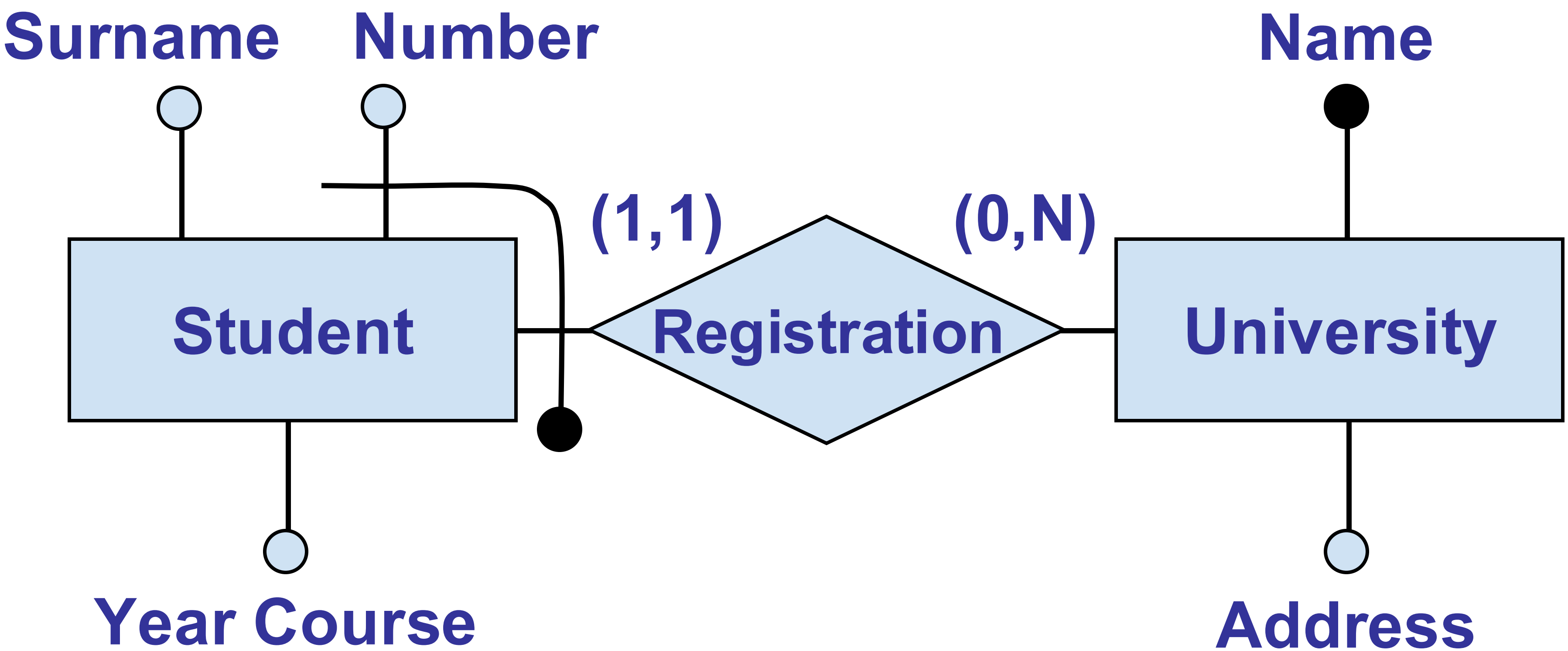
Attribute Plate is the internal identifier for Car because there are no two cars with the same plate

Internal Identifiers (2)



Attributes Name, Surname e Birth Date define Person's internal identifier, because (we assume that) no two persons with the same name, surname and birth date can exist

External Identifier



Attribute Number and entity University form the external identifier because there could be students with the same number but belonging to different universities

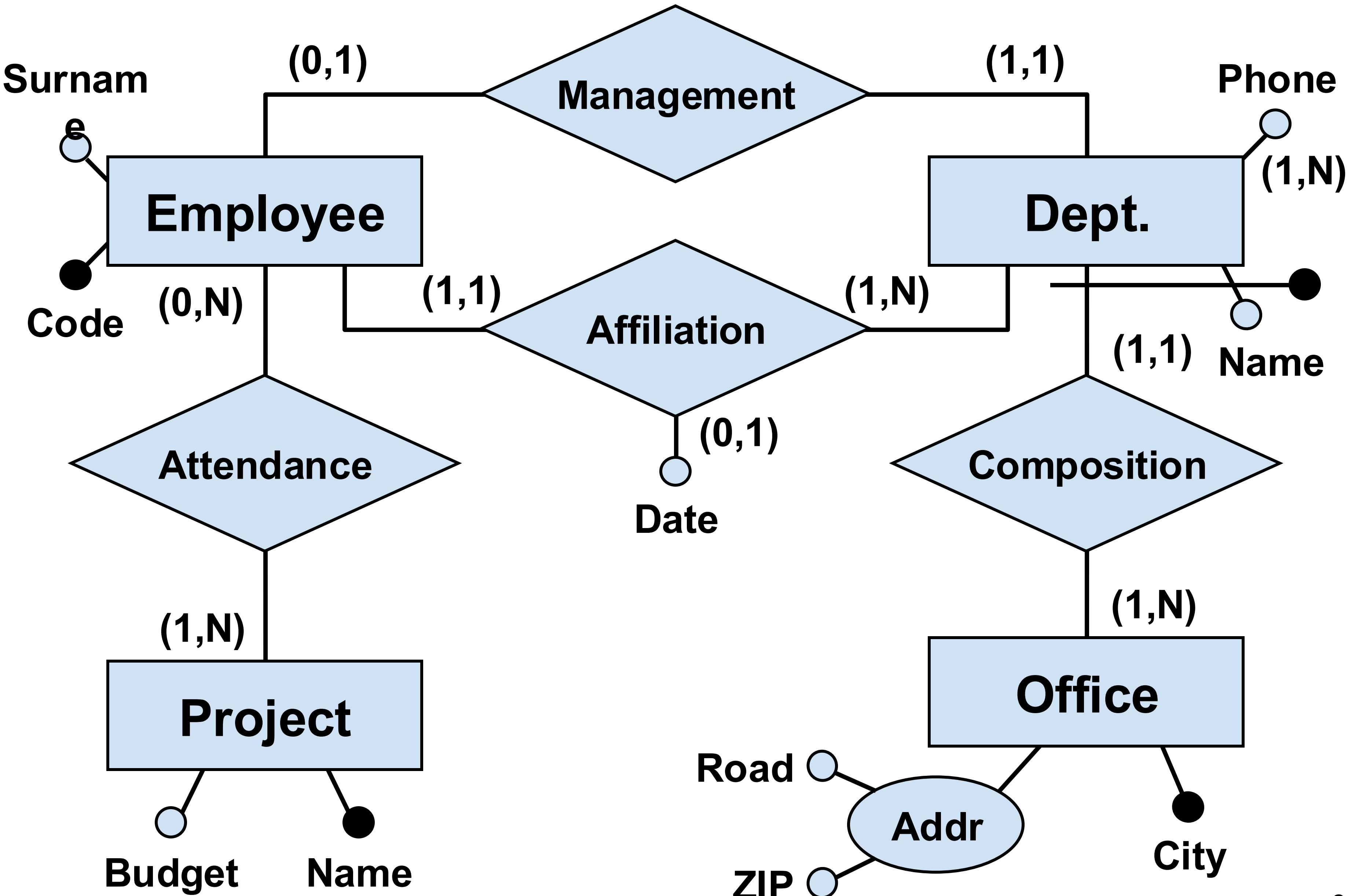
Remarks

- Each entity must have at least one identifier, but it can generally have more than one identifier
- An external identify is possible only through a relationship to which the identity to be identified participates with a (1,1 cardinality)
- **Why do not relationships have identifiers?**
  - It's better to associate attributes to entities and not to relations

Memories from SQL Exercises

```
ACTOR(Id, Name, Year, Nationality)
RECITAL(Actor, Film)
FILM(Code, Title, Year, Director,
     Country, Genre)
SCREENING(Code, Film, Room, Date,
          Profits)
ROOM(Code, Name, Seats, City)
```

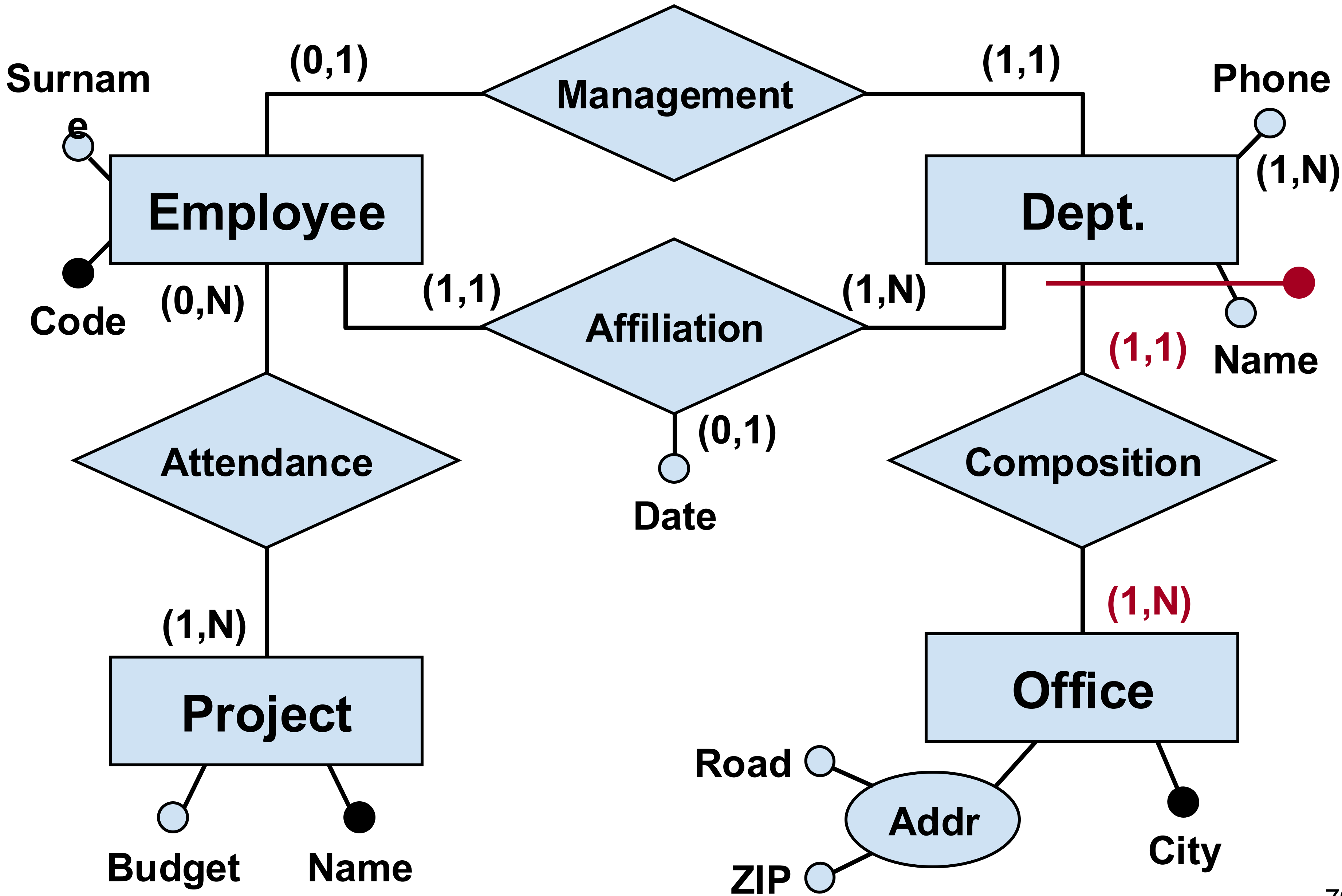
ER Schema: an Example (2)



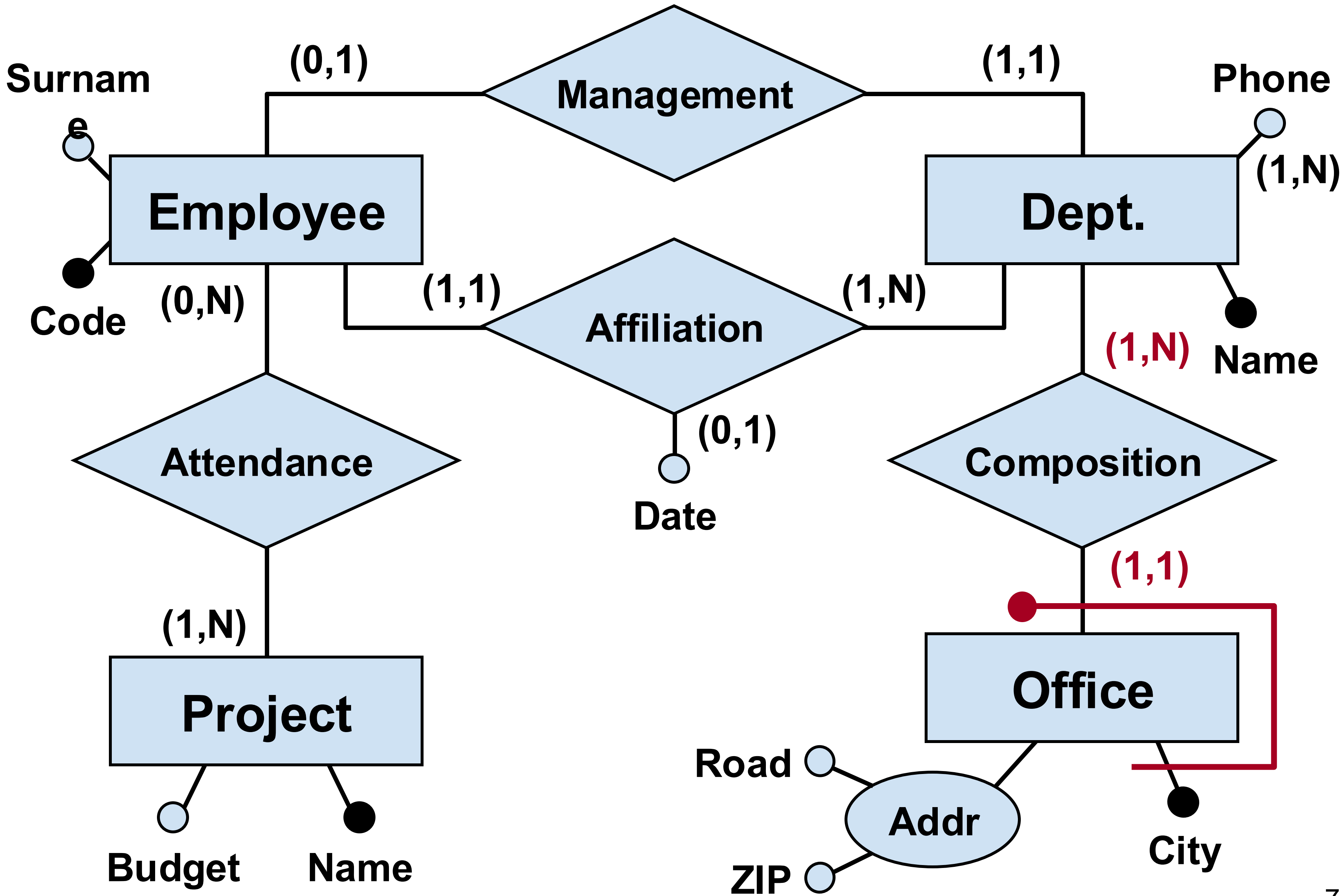
Warning

- Slight changes in cardinalities and identifiers could alter the whole meaning of the diagram

ER Schema: an Example (3)



ER Schema: an Example (4)



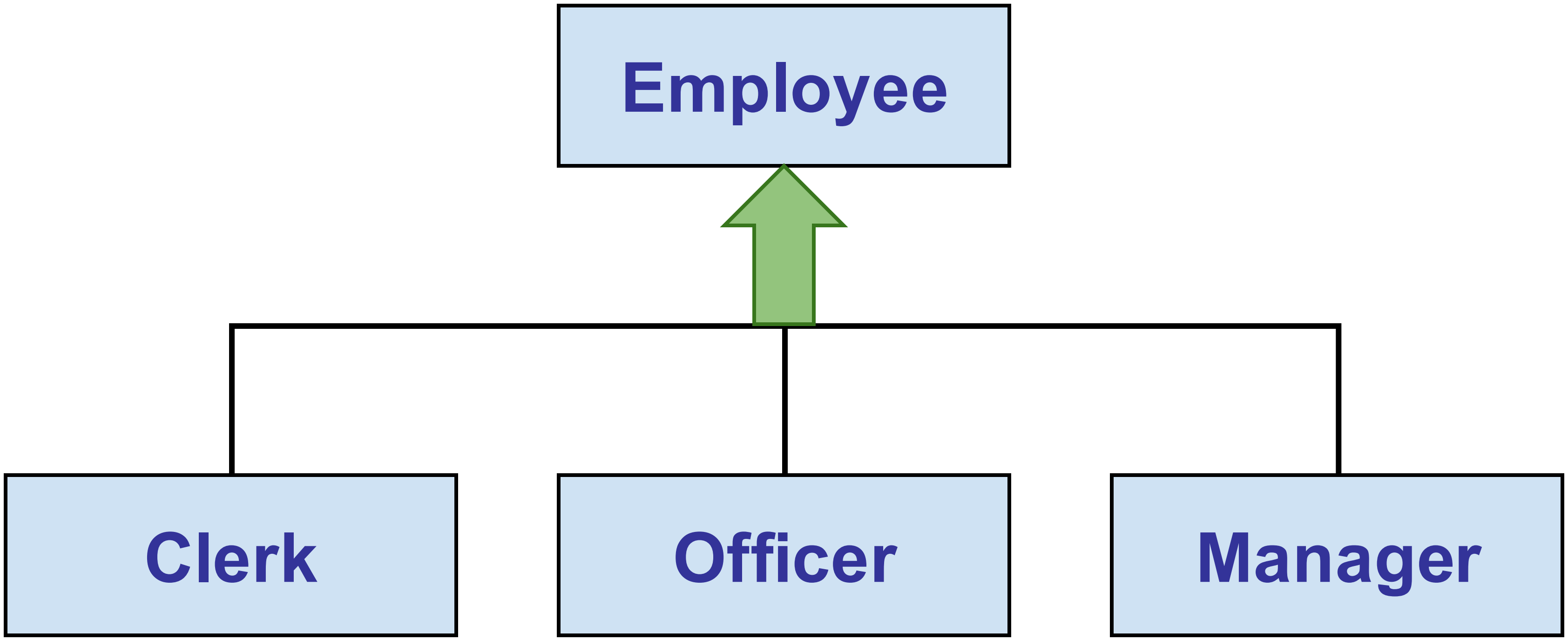
Extern Key and Cardinality

- In the first diagram for each Dept. there is only one Office: we could have Dept.s with the same Name, the only constraint is that Dept.'s City has to be different
- Viceversa in the second diagram each Office belongs to only one Dept.: hence in the same City we could have many different Dept. having different Names

Generalization

- Relates entities one or more entities E_1, E_2, ..., E_n with one other entity E having E_i-s as specific cases
  - E is a **generalization** of E_1, E_2, ..., E_n
  - E_1, E_2, ..., E_n are **specialization** of E

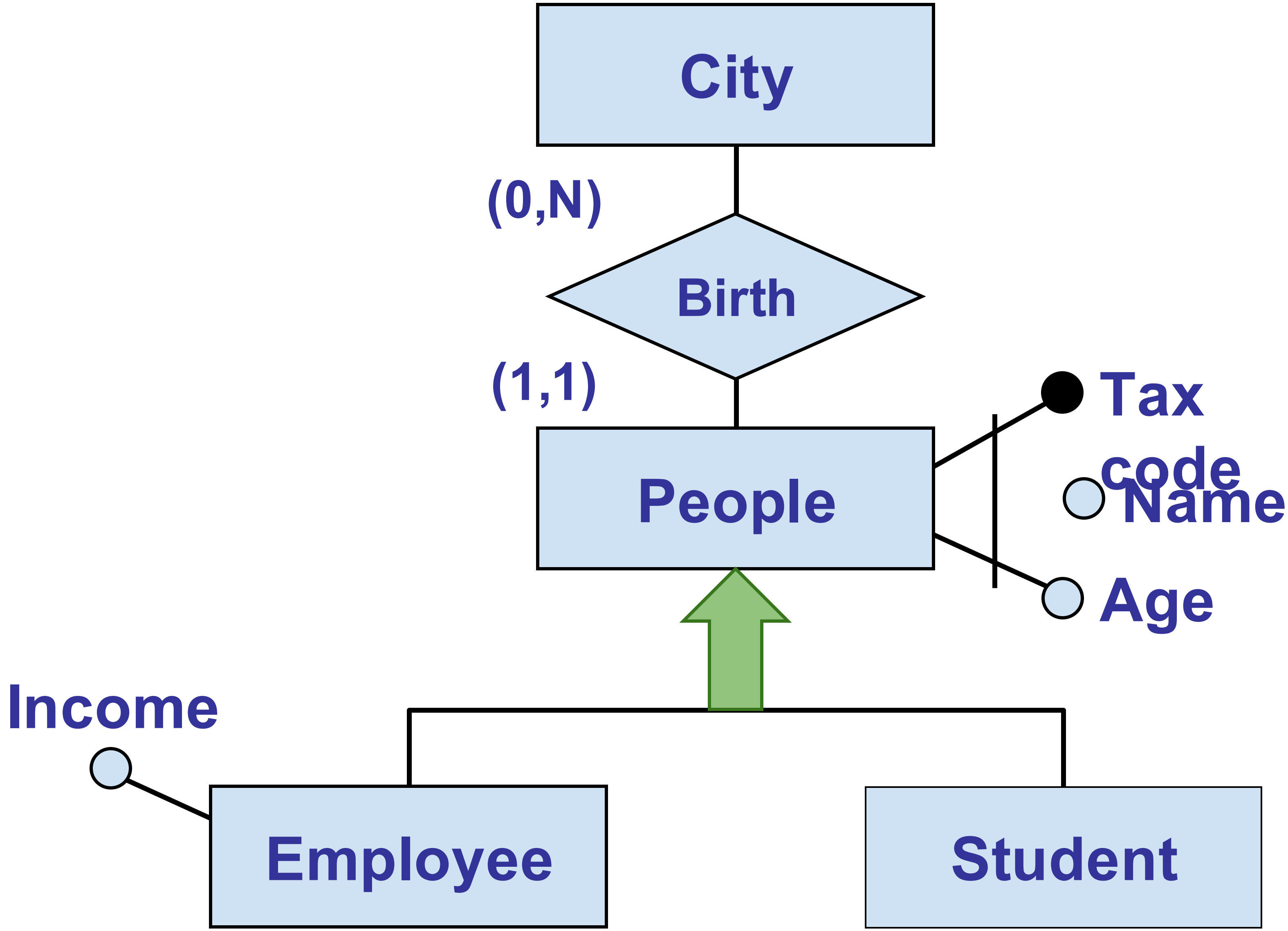
Graphical Representation



Generalization Properties

- If E (parent) is a generalization of E_1, E_2, ..., E_n (children):
  - Each property in E is significant for E_1, E_2, ..., E_n
  - Each occurrence of E_1, E_2, ..., E_n is occurrence of E, too

Generalization Properties: an Example



Inheritance

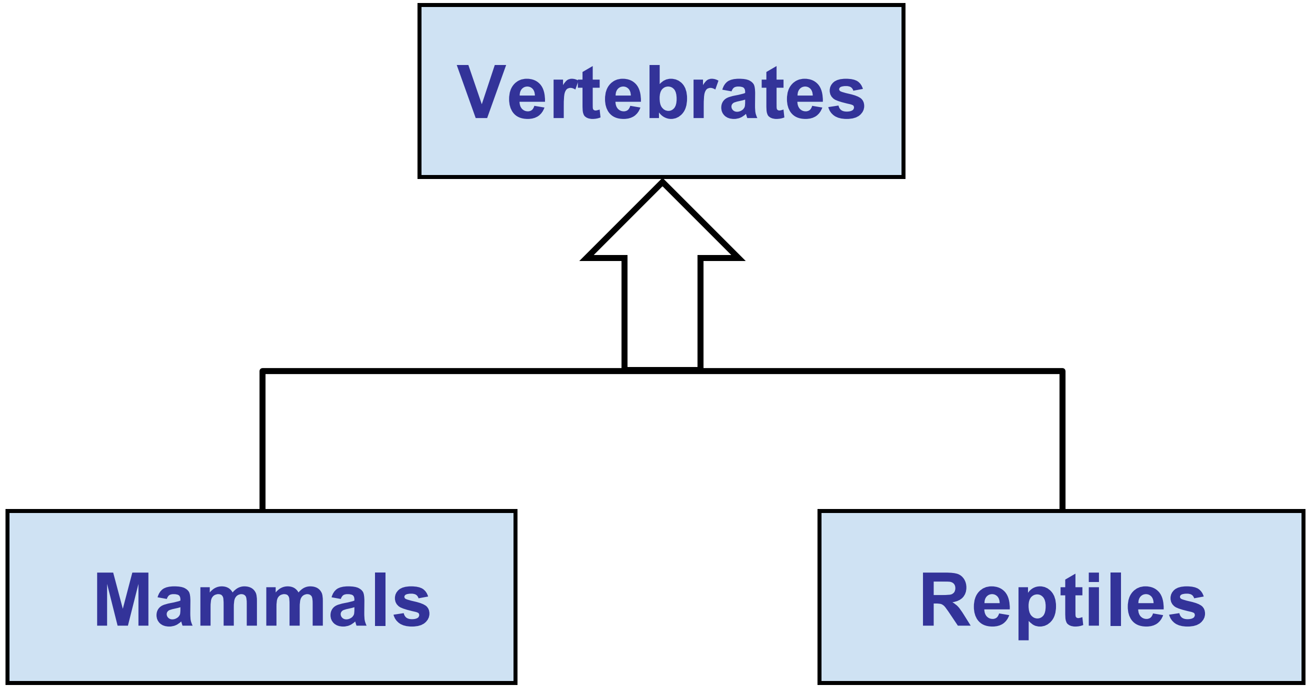
- All the parent entities' properties (attributes, relationships, generalizations) are inherited by the child entities and not explicitly represented

Types of Generalization

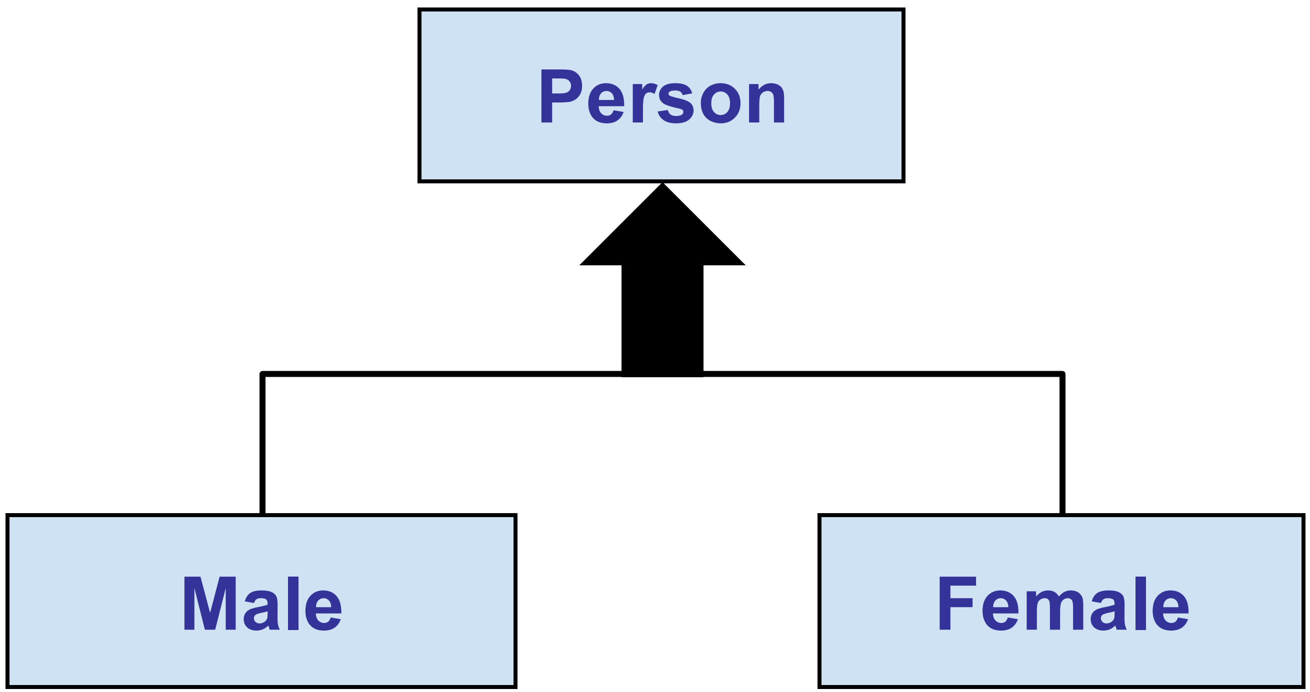
- Total if each occurrence of the parent entity is occurrence of at least one of the child entities, partial otherwise
- Disjoint if each occurrence of the parent entity is occurrence of at most one of the child entities, overlapping otherwise

- However, we only consider (without loss of generality) disjoint generalizations and distinguish between total and partial ones

Generalization: Disjoint Partial



Generalization: Disjoint Total



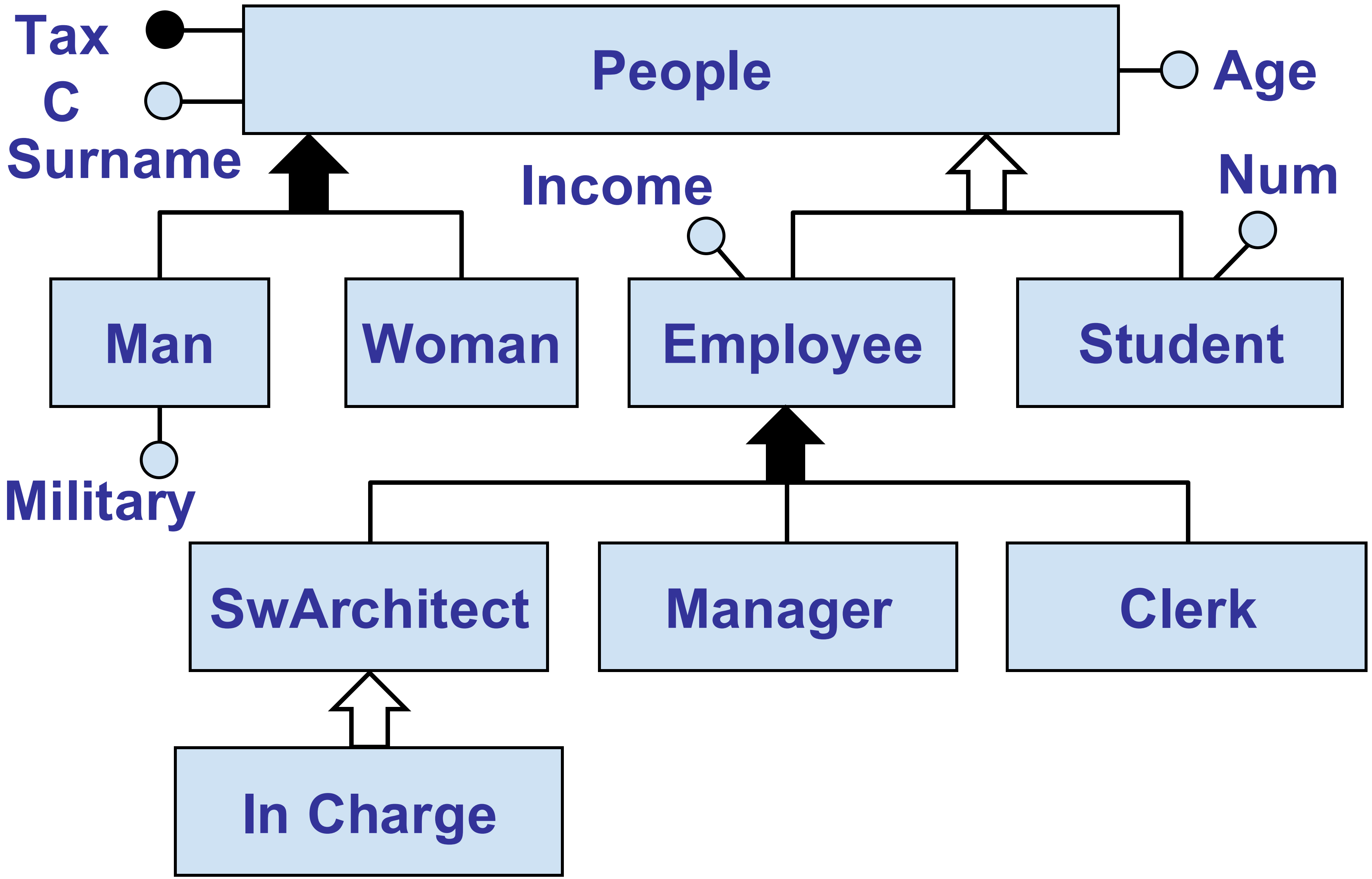
Other Properties

- We could define multi-level hierarchies and generalization at the same level
- Each entity could be included in more different hierarchies, either as a parent and as a child
- A generalization which has only one child entity is called **subset**
- Some configurations do not make any sense
- The parent in a generalization could have no identifier, as long as …

Exercise

People have Tax Code, Surname and Age; for each man we want to know whether he did military service; employees have an income and could be either managers, clerks or (software) architect (some of whom are in charge of a project); Students (that cannot be employees) have a (registration) Number; some people are neither Employees nor Students (and we don't need further details).

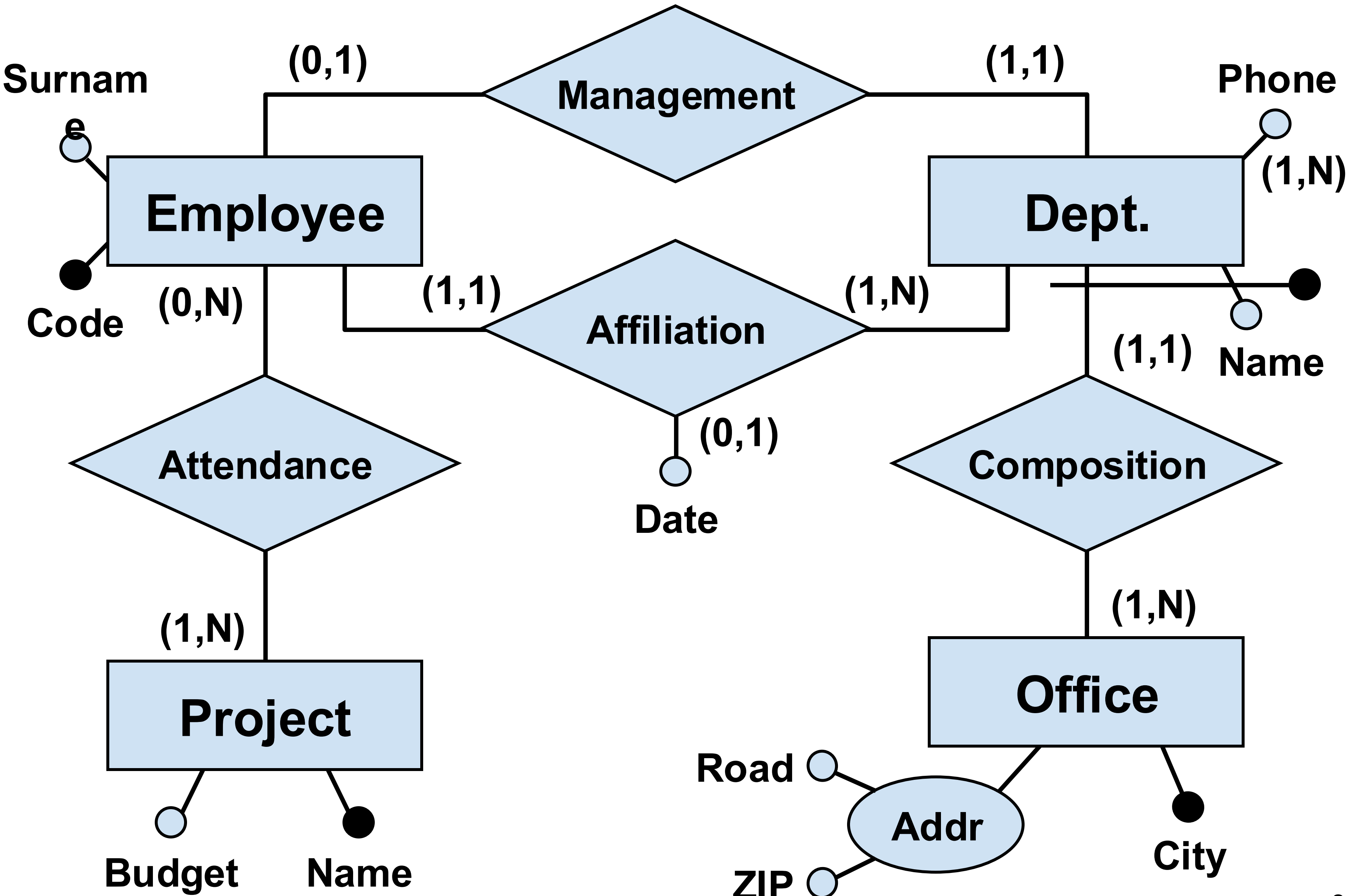
Conceptual Schema



Conceptual Modelling Documentation

- Data dictionary
  - Entity
  - Relationship
- Not-expressible Constraints

Conceptual Schema Final



Data Dictionary (Entities)

Entity	Description	Attributes	Identifier
Employee	Employee in a Dept.	Code, Name, Surname	Code
Project	Projects of a Dept.	Name, Budget	Name
Department	Structure of a Dept.	Name, Phone	Name, Office
Office	Office’s location	City, Address	City

Data Dictionary (Relationships)

Relationships	**Description**	**Components**	**Attributes**
Management	Management of a Dept.	Employee, Dept.	
Affiliation	Affiliation to a Dept.	Employee, Dept.	Date
Attendance	Attendance for a project	Employee, Project	
Composition	Composition of Departments	Dept., Office	

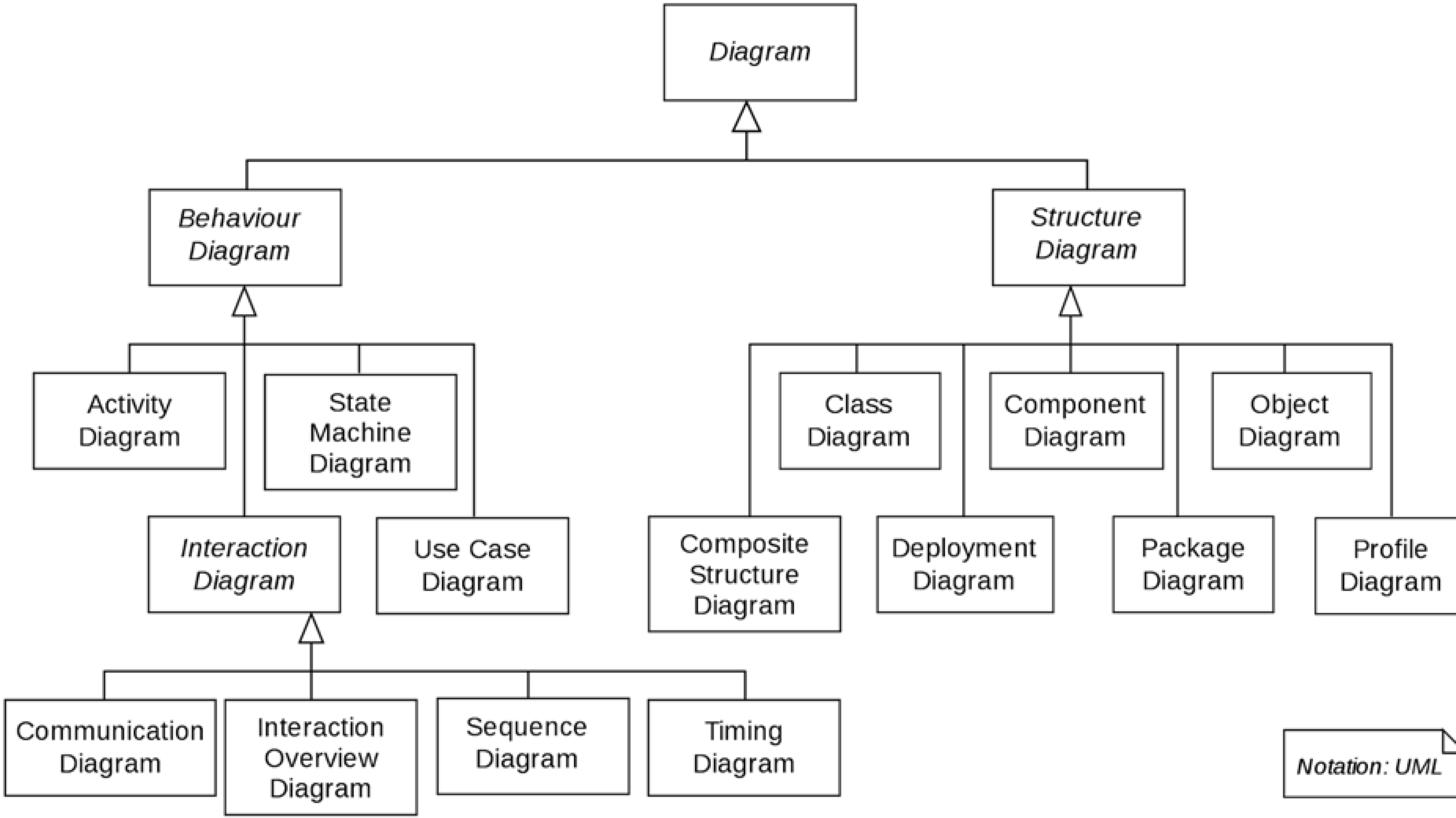
Not-expressible Constraints

Integrity Constraints on Data
1. A department director must belong to that department
2. An employee must not have an income greater than the director of the department which he is affiliated
3. A department placed in Rome must be directed by an employee with at least 10 years of service
4. An employee that isn't affiliated to any department must not attend to any project

Unified Modeling Language

- A **modeling language** allows the specification, the visualization, and the documentation of the development of a software system
- The **models** are descriptions which users and developers can use to communicate ideas about the software
  - UML 1.* is a modeling language
  - UML 2.* is still a modeling language, but it is so “detailed” that can be used also as a programming language

UML2 Types of Diagrams

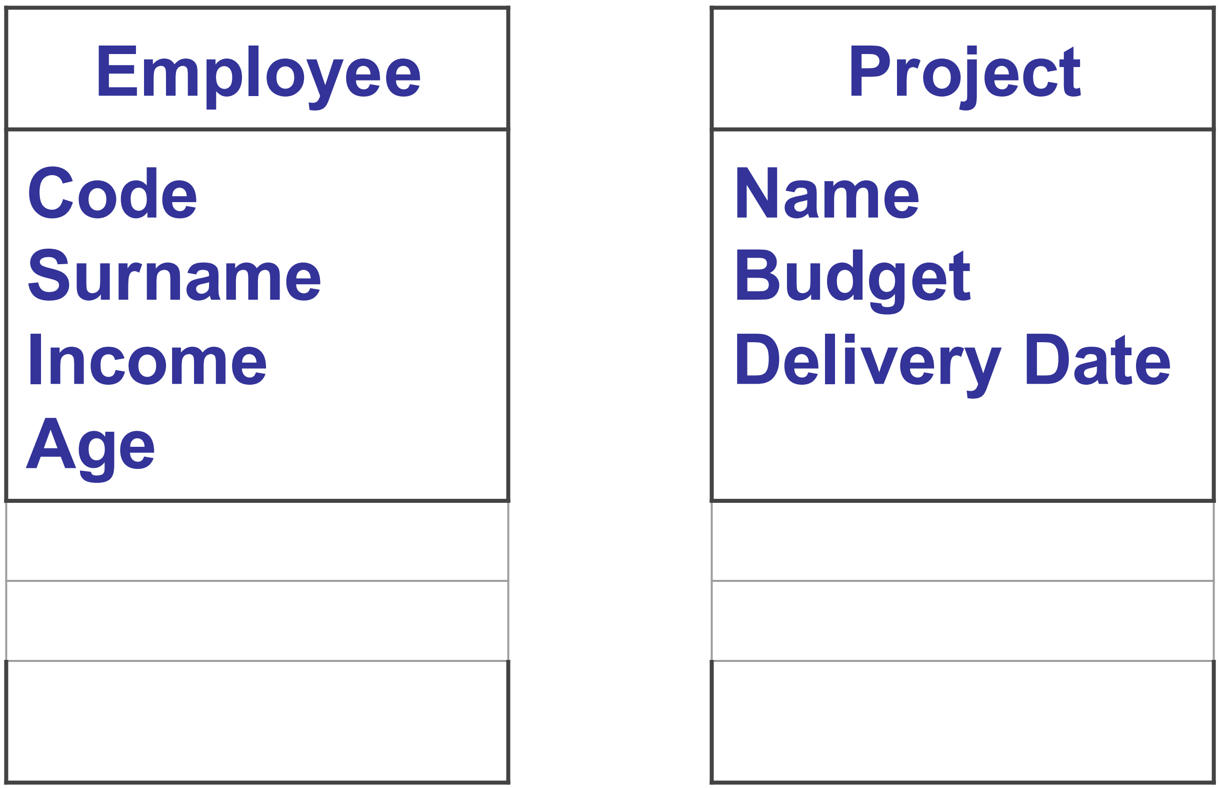


[Source: Wikipedia]

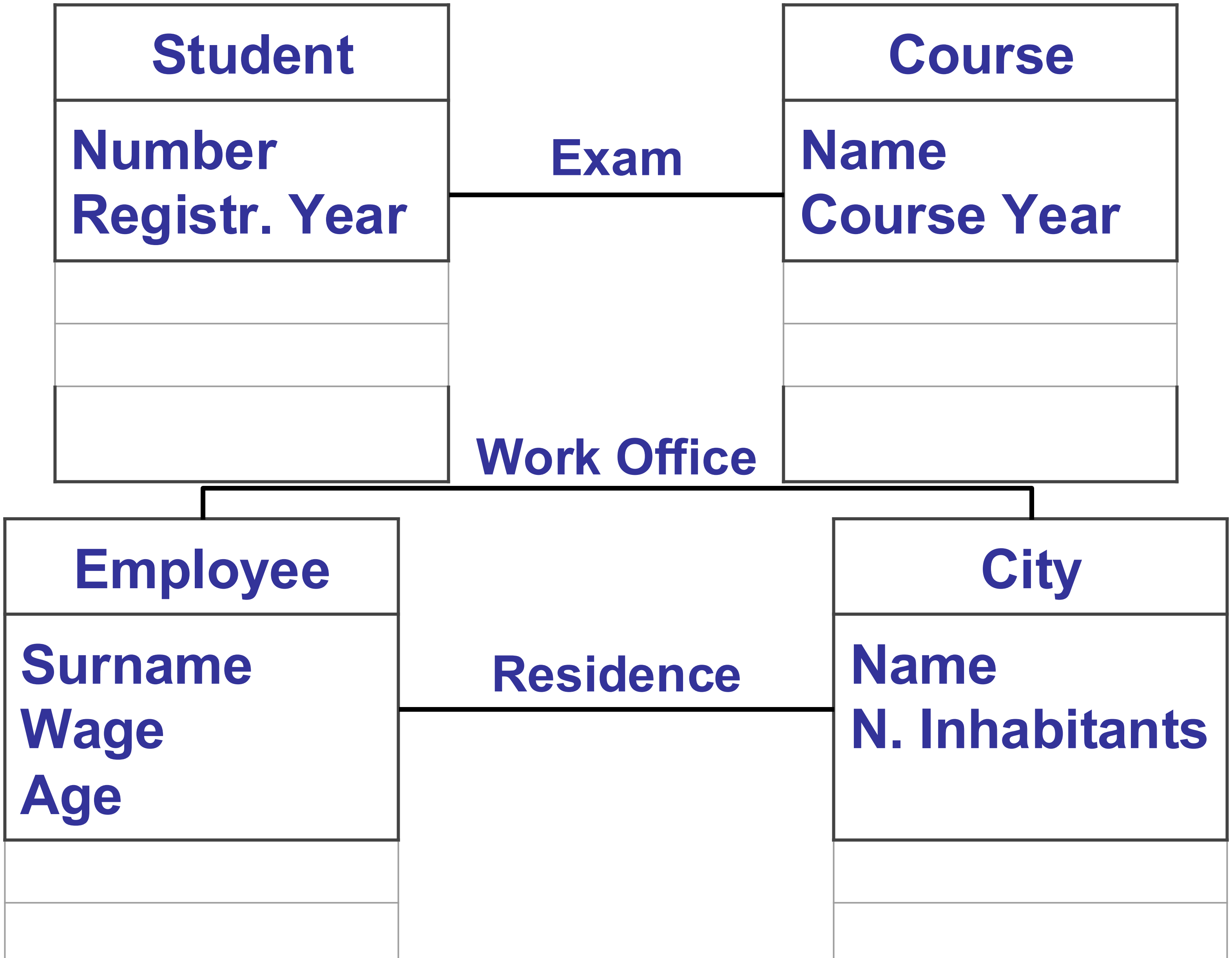
Data Modelling in UML2

- UML2 could be used in place of ER for data modelling
  - In UML2 we use Class Diagrams
- The graphical representation changes, but the design approach is the same
- Let’s see how to represent Conceptual Models with UML2

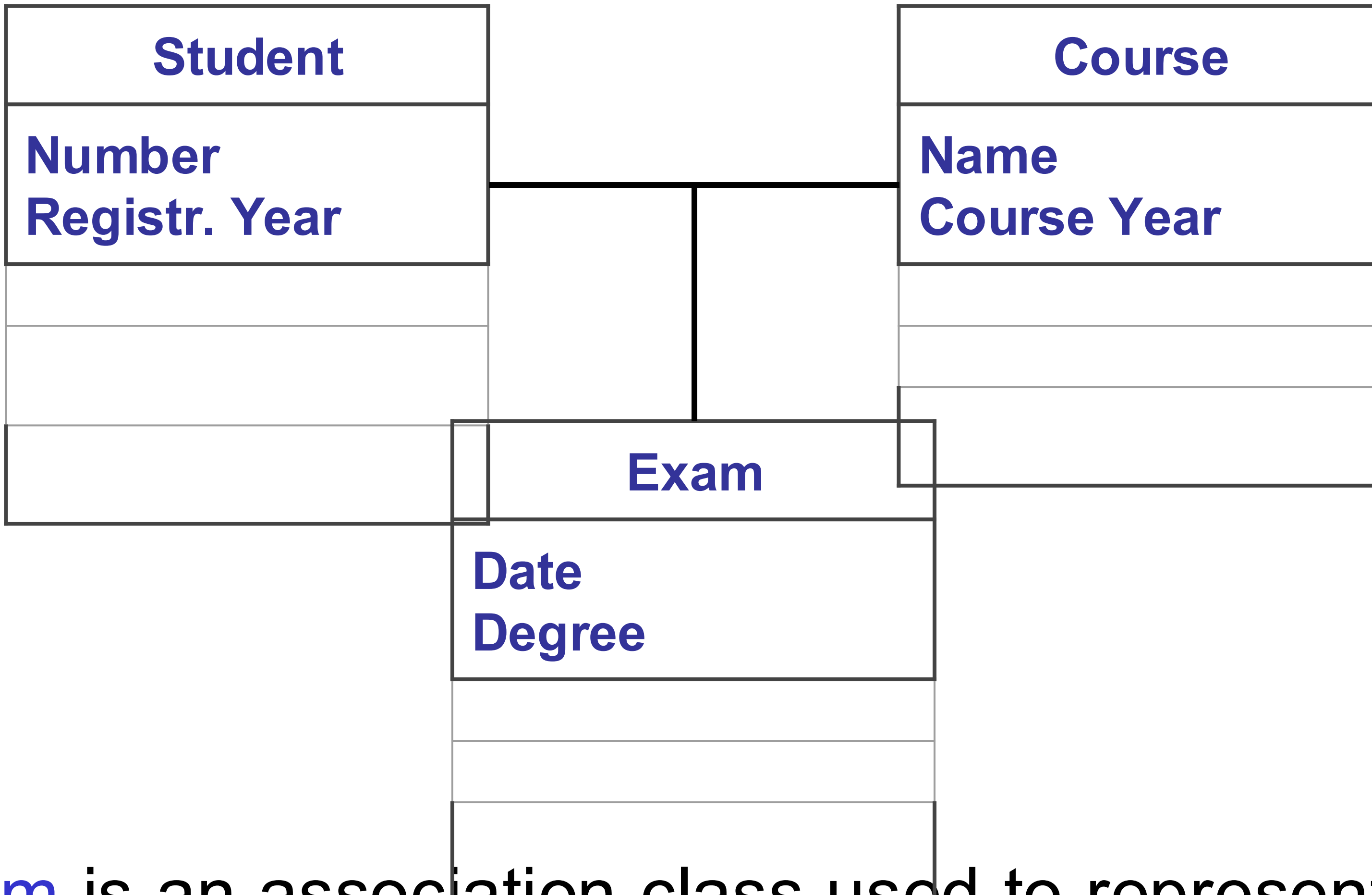
Classes


Employee
Code
Surname
Income
Age
Project
Name
Budget
Delivery Date

Associations

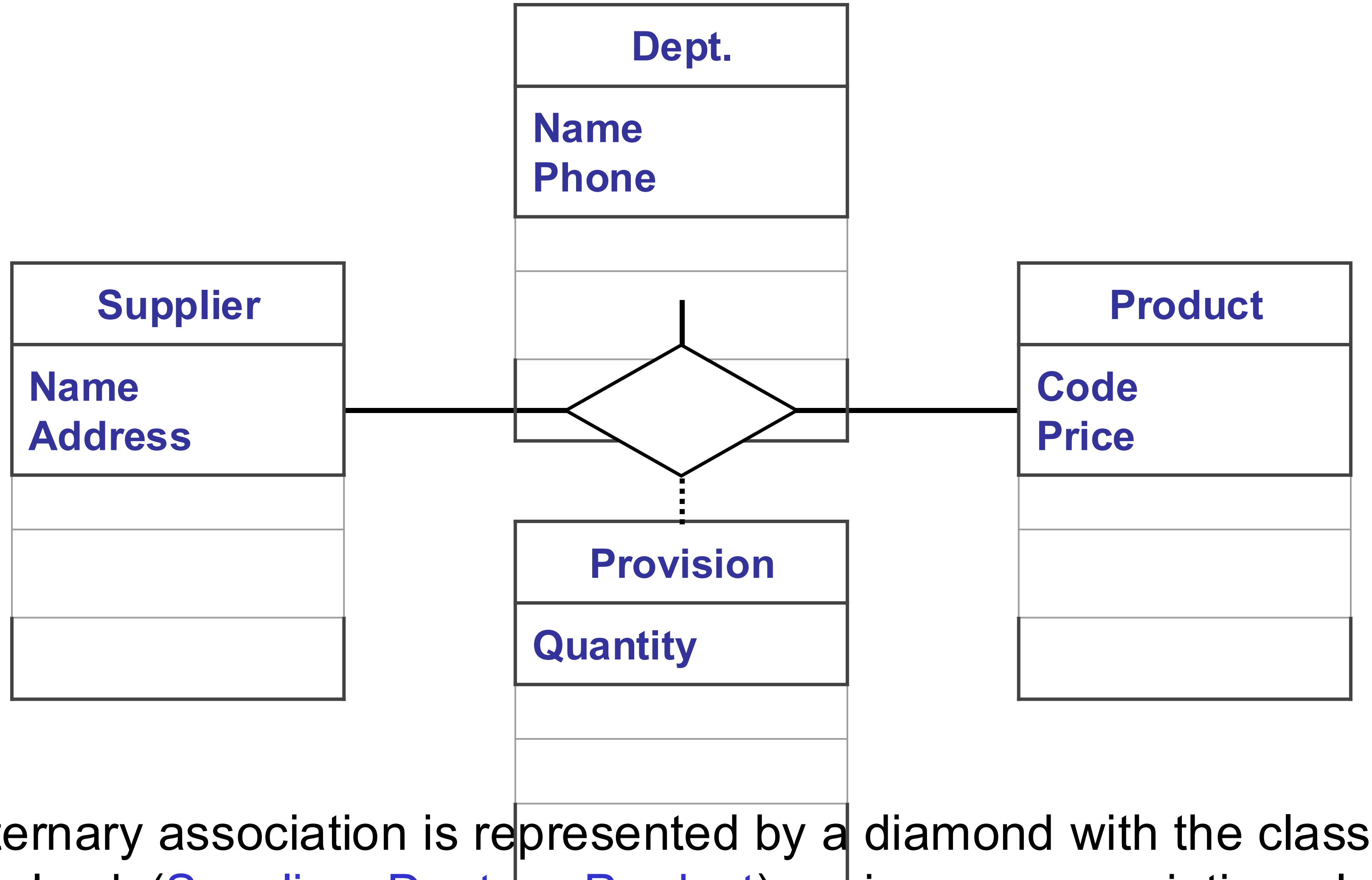


Association Class



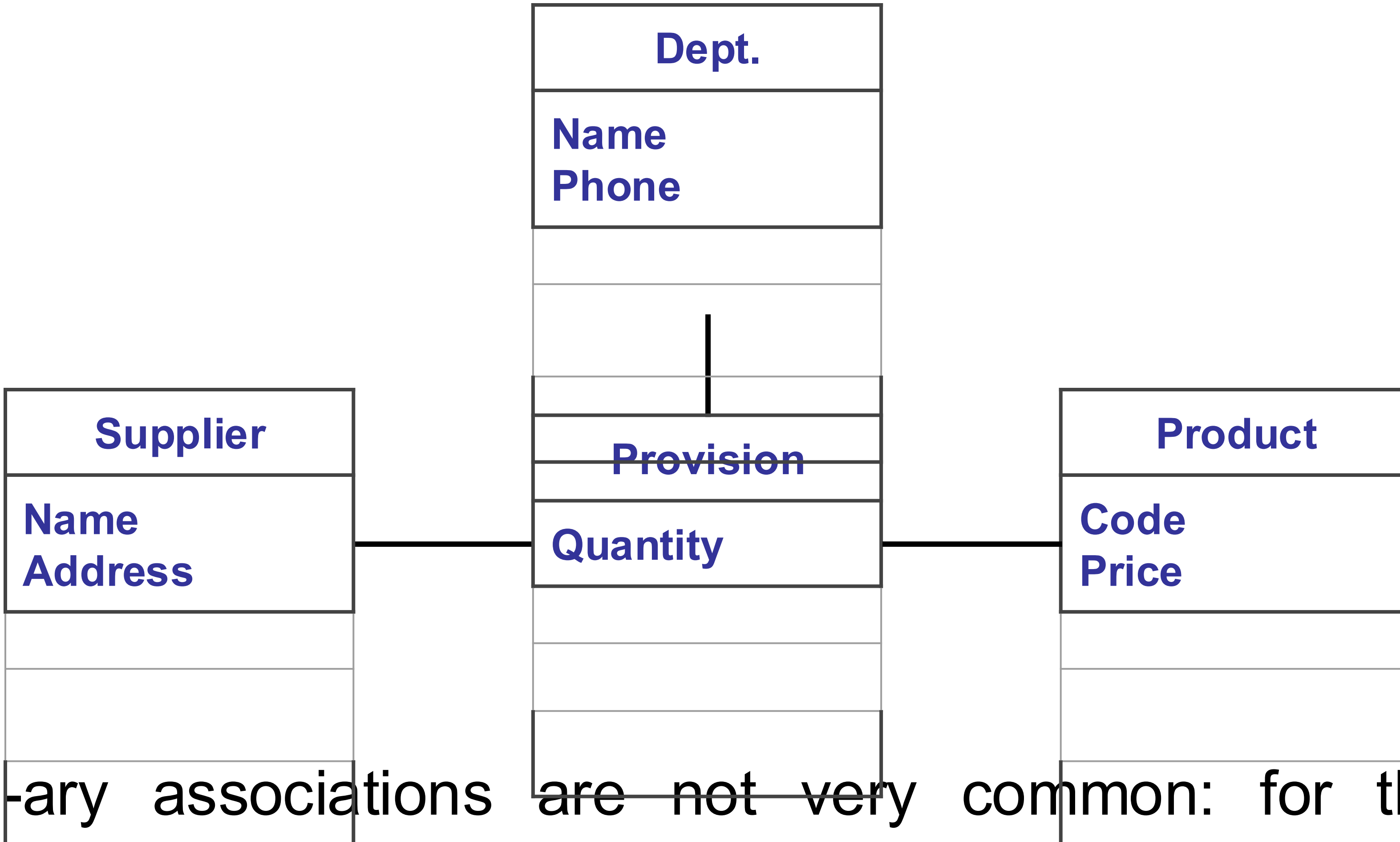
Exam is an association class used to represent the Degree and Date attributes of the association between Student and Course

Ternary Association



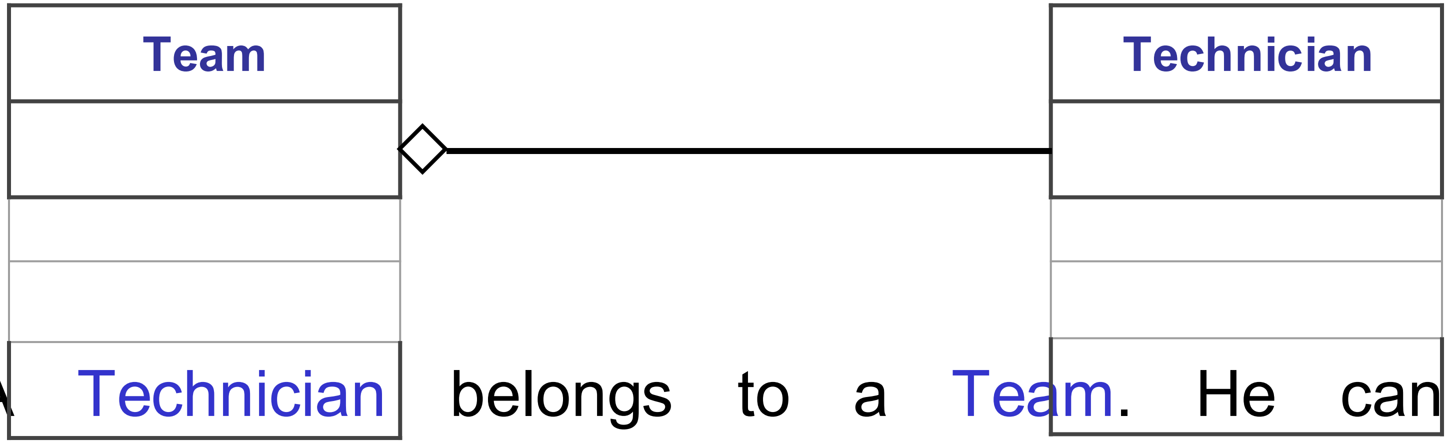
A ternary association is represented by a diamond with the classes involved (Supplier, Dept. e Product), using an association class (Provision) in order to assign attributes to the association among these classes

Association's Reification

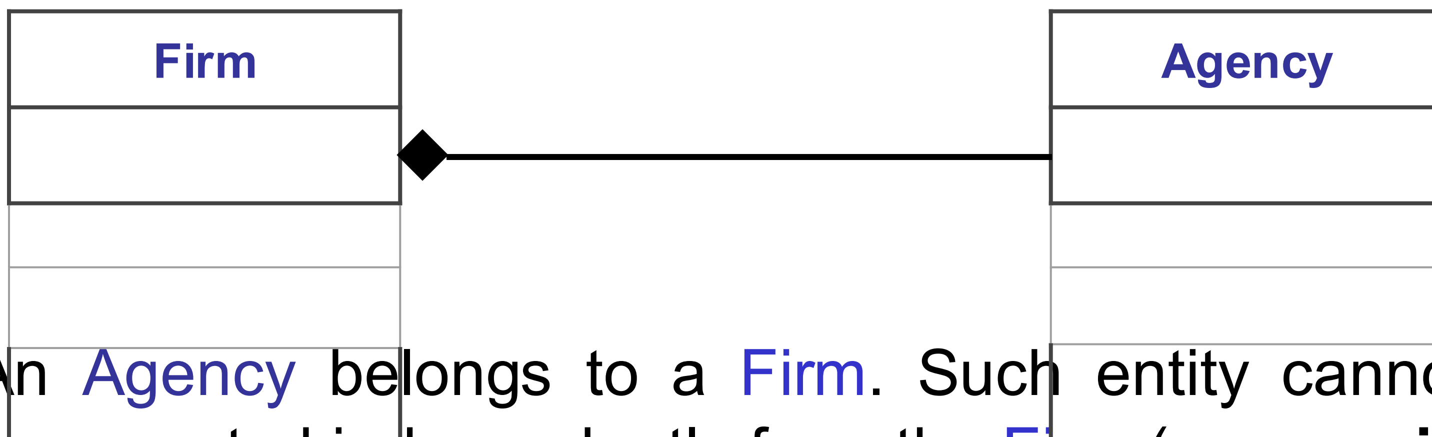


N-ary associations are not very common: for this reason they should be reified into one single class (Provision) linked to the original classes with a binary relation

Aggregation and Composition

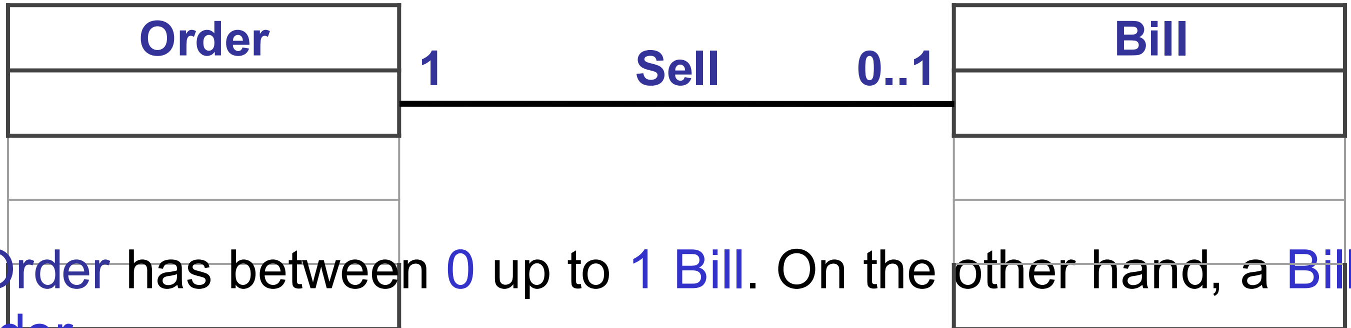


- A Technician belongs to a Team. He can be represented independently from the Team where he belongs (**aggregation**)

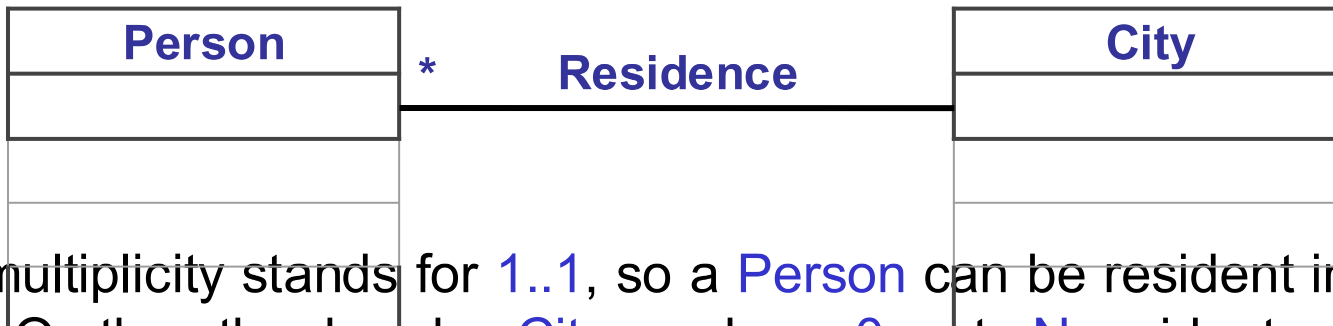


- An Agency belongs to a Firm. Such entity cannot be represented independently from the Firm (**composition**)

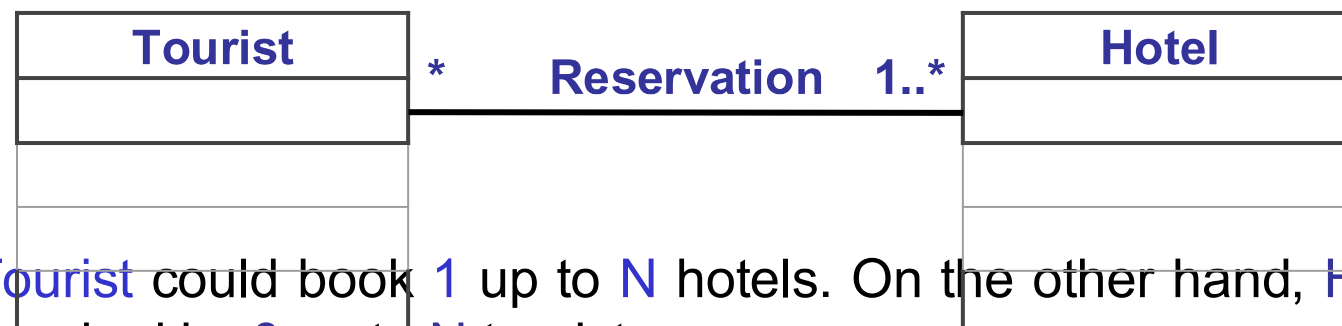
Associations’ Cardinality



- An Order has between 0 up to 1 Bill. On the other hand, a Bill has just 1 Order

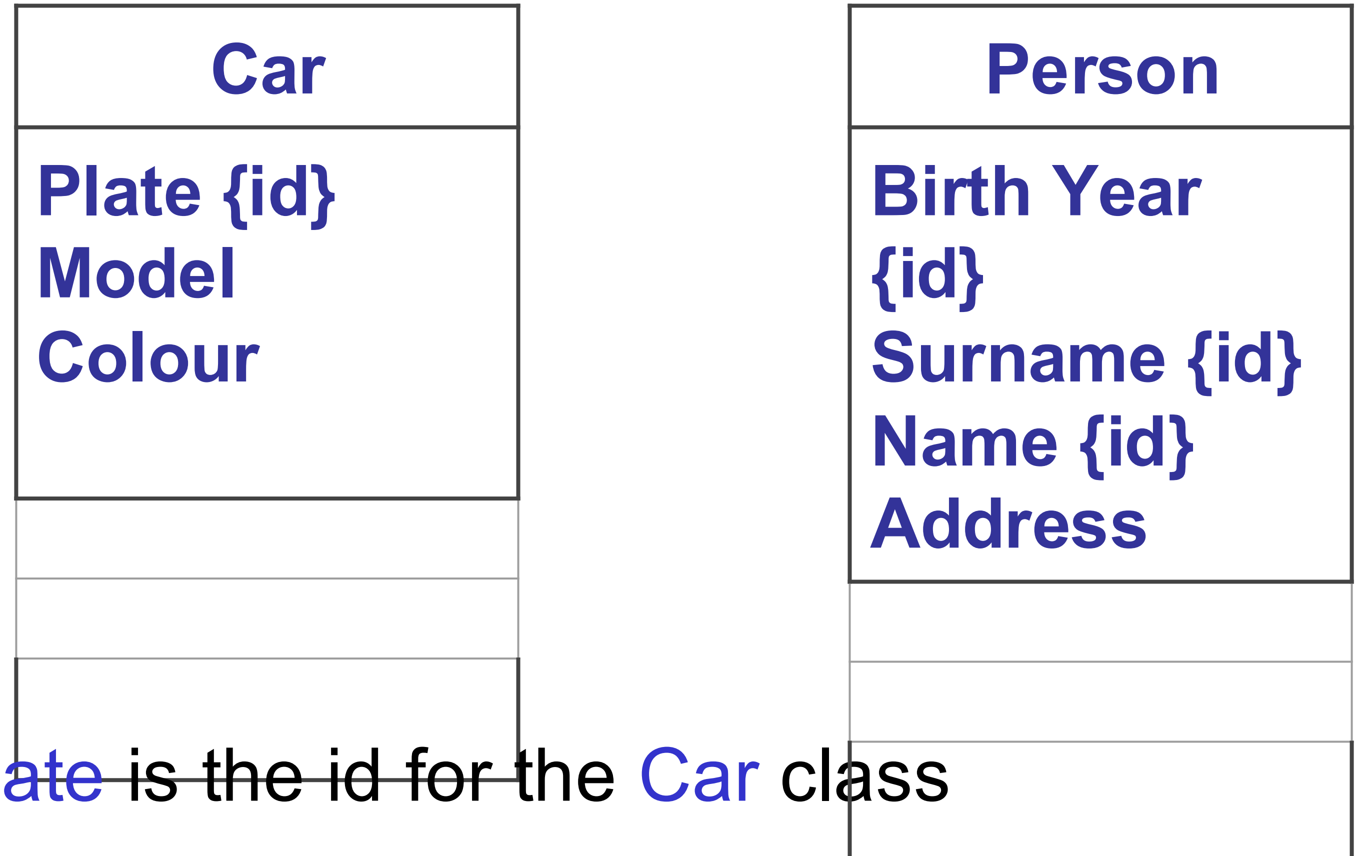


- No multiplicity stands for 1..1, so a Person can be resident in just one City. On the other hand, a City can have 0 up to N resident people



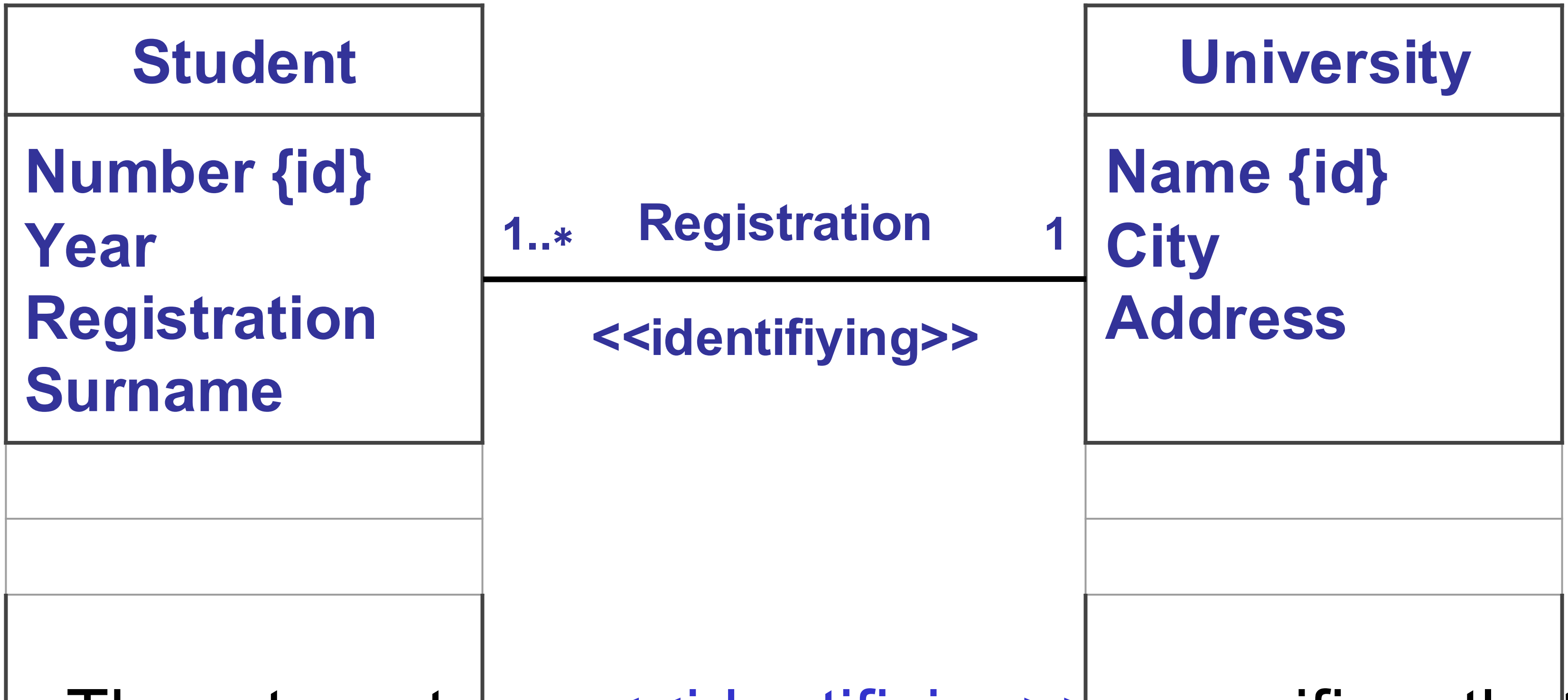
- A Tourist could book 1 up to N hotels. On the other hand, Hotel can be booked by 0 up to N tourists

Identifiers



- Plate is the id for the Car class
- Attributes Date Nascita, Surname and Name are identifiers for the Person class

External Identifier

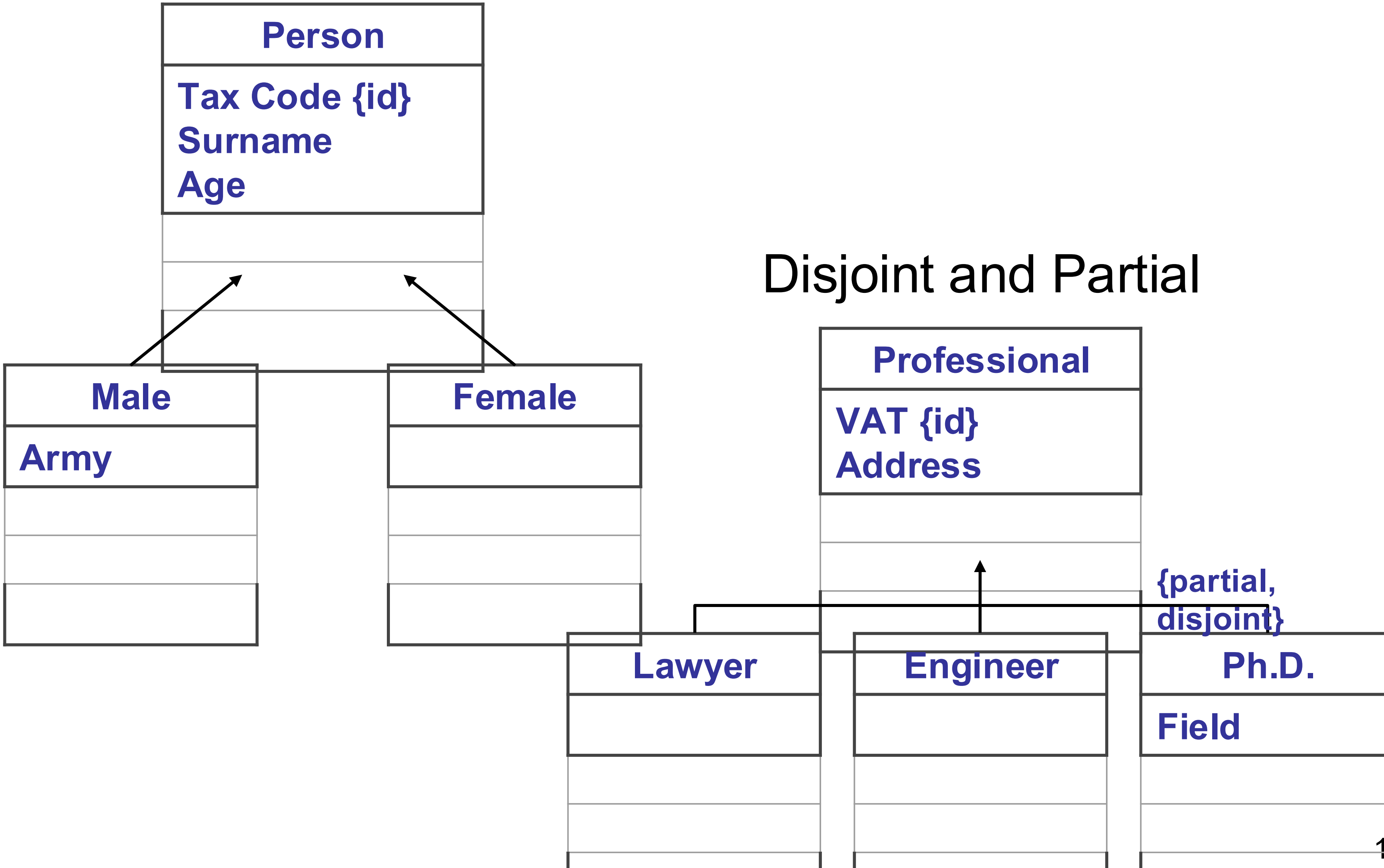


- The stereotype <<identifiying>> specifies that the association between Student and University together with the Number id is precisely identifying a given student

Generalization

Disjoint and Total

Disjoint and Partial



Conceptual Modelling in UML