# *ICT & Business Intelligence & CRM*
# Databases Design: Conceptual Model

Andrea Vandin
Scuola Superiore Sant'Anna
Paolo Ferragina
Scuola Superiore Sant'Anna
Anna Monreale
Università di Pisa

*Reference: Chapter 2 of Databases Essentials, Antonio Albano*

# Need to design

- Database are often born designless,
  - from a huge spreadsheet
- Anomalies arise, because of redundancy
- Redundancy generate errors
- Design must involve the user

# Anomalies

| Name | Surname | Address | StudId | Subject | Date | Grade |
|---|---|---|---|---|---|---|
| Mario | Addis | Via Roma | 354765 | BD | 1/1/13 | 28 |
| Luca | Bini | Via Pola | 354234 | BD | 2/3/12 | 18 |
| Mario | Addi | Via Roma | 354765 | Alg | 1/1/13 | 27 |
| Luca | Bini | Via Pola | 354234 | Pro | 2/5/12 | 30 |
| Luca | Bini | Via Bari | 354234 | Lab | 3/4/12 | 24 |

# Phases for DB realization

- User requirements analysis & specification
  - collecting **user needs** and normalizing them according to standards
- **Conceptual design (TODAY-NOW)**
  - Focus is more on how tables are related with each other
    - e.g., we do not need to consider all attributes/domains of tables
  - This is the phase in which requirements are formalized and integrated into a **global conceptual schema**
    - (Global because it considers all tables)
  - We use a DBMS-independent (conceptual) language
- Logical design (NEXT CLASS)
  - The conceptual schema is mapped into a **logical schema**
    - We use the data model supported by the chosen DBMS
    - We get closer to the actual DB creation. It depends on the chosen DBMS
- Physical design
  - concerns the selection of the **data structures** used to store and retrieve the data.

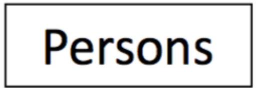
# User Requirements Analysis

- Difficult activity because hard to standardize

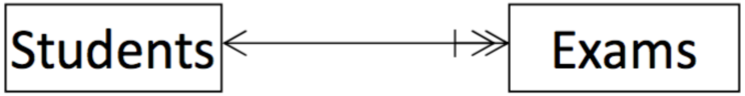
- Suggestions
  - Involve the users many times for iterative checks
  - Consider the point of view of the applications users
  - Be sure that you are using a common language
  - Identify case studies that you can discuss in detail
    - to identify the properties to be captured by the model

# An Object Oriented Language for data design

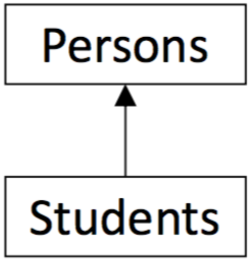
- Realization of a **diagram** representing the conceptual model of the database
- Components:
  - **Classes** (collections – the tables)



  - **Relationships** among classes



  - **Sub-collections** links



# Class Diagram

- Phase of Analysis
  - Need to adopt the right level of abstraction

- In particular
  - We do not need all attributes
  - Type (numeric, string) of attributes is not necessary

# Example: University DB

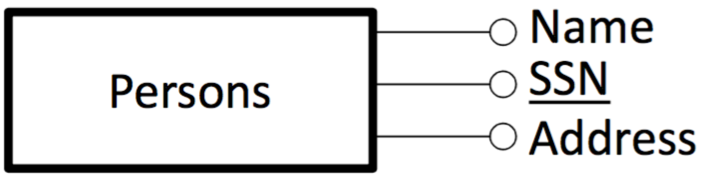
- We need to design the database for managing data about **courses** of the computer science degree at the University of Pisa
- The system must manage data about **students** of both the master and bachelor programs. For each program, we need to maintain data related to the students' **exams**.
- We need to record data about the **courses** and the **students' exams** for each course.
- We want to record teachers for each course, who may be more than one. Moreover, a **teacher** may be **internal** or **external**.
- For each teacher, we have one or more **phone numbers**.
- We need to record each student's supervisor (a **teacher**). Bachelor students may ask for a supervisor only when they are attending 3rd year.
- Lastly, master students help (tutor) bachelor ones. The system must maintain information about such tutoring activities.

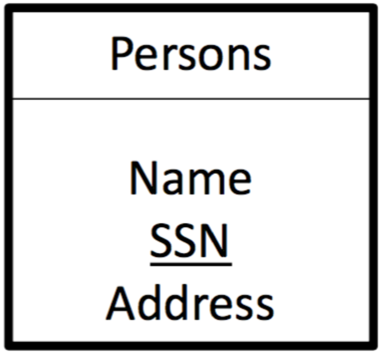
# Classes

- **“Concepts” of the reality to be modelled**
  - facts, people, things,
  - examples: student, course, exam, teacher
- **Instances of a class**
  - entities, objects of the reality to be modelled
    - Student Pinco Pallino, course ICT, teacher Andrea Vandin
- **Classes have attributes**
  - Properties relevant for the application

# Class with attributes

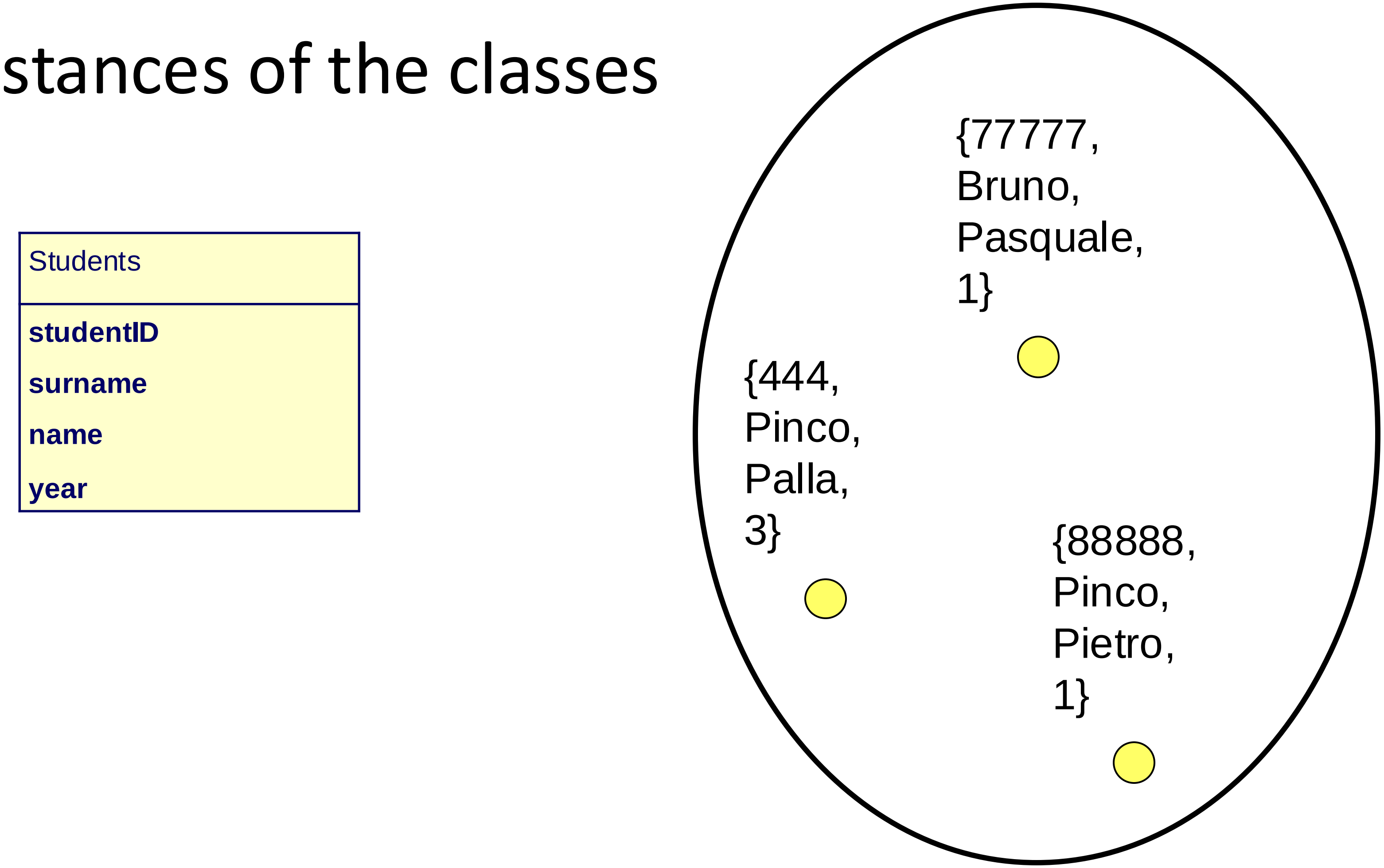
- A person class, with attributes:
  - Name
  - SSN (key)
  - Address





# Classes

- Instances of the classes



Students

studentID

surname

name

year

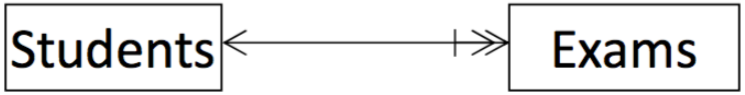
{77777,
Bruno,
Pasquale,
1}

{444,
Pinco,
Palla,
3}

{88888,
Pinco,
Pietro,
1}

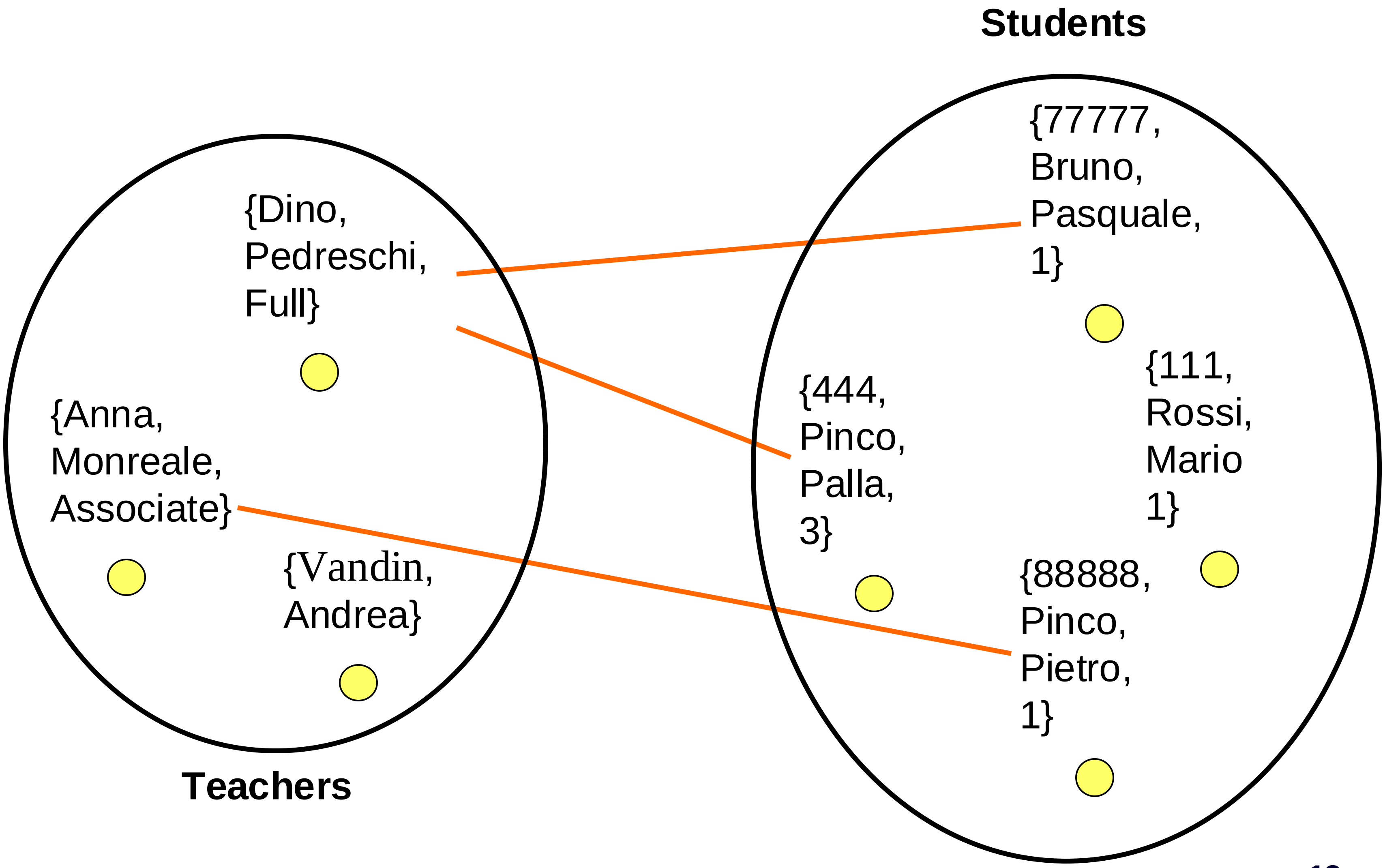
# Relationship

- **Relationship between classes**
  - Logic link relevant for the application
  - ex: **teaching** between teacher and course
  - ex: student **passes** an exam



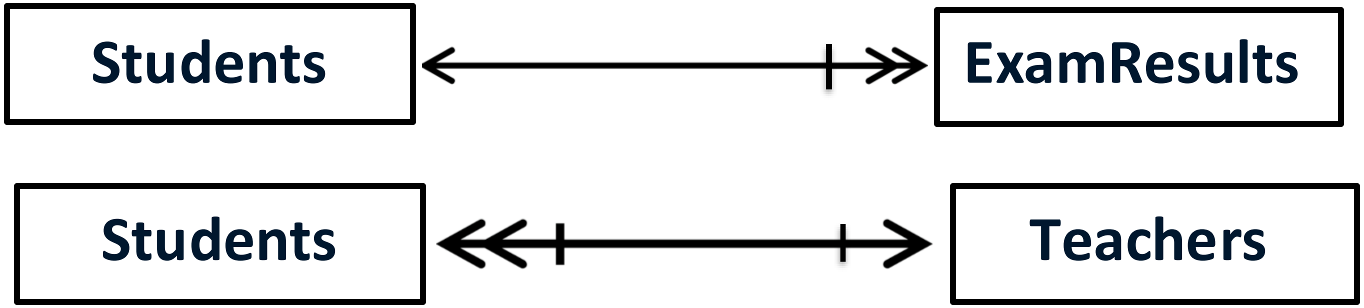
- **Instance of a relationship**
  - A set of edges between instances belonging to the involved classes

# Relationship: Instances



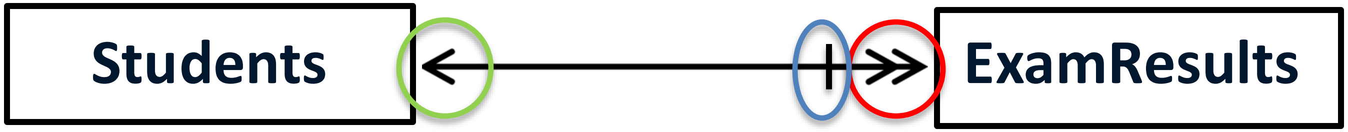
# Cardinality

- **Constraints on relationships**
  - Constraints on the number of edges between instances of classes
    - How many supervisors can a specific student have?
- **Minimal Cardinality:** 0 or 1
- **Maximal Cardinality:** 1 or many



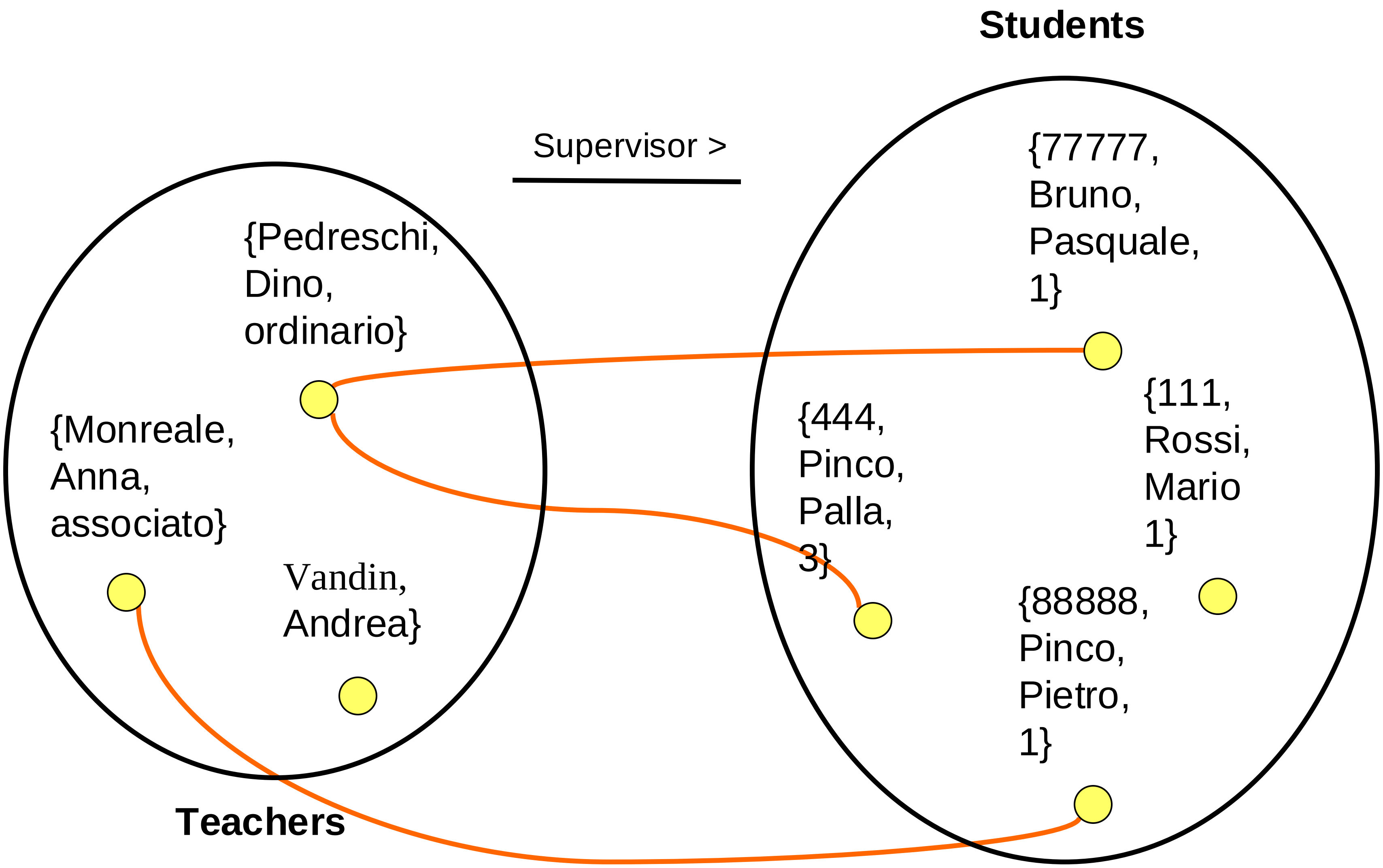
# Cardinality

- **Constraints on relationships**
  - Constraints on the number of edges between instances of classes
    - How many supervisors can a specific student have?
- **Minimal Cardinality:** 0 or 1
- **Maximal Cardinality:** 1 or many



A student can pass zero or several exams
An exam result is precisely for one student only

# Cardinality



Students

Supervisor >

{Pedreschi, Dino, ordinario}

{Monreale, Anna, associato}

Vandin, Andrea}

{77777, Bruno, Pasquale, 1}

{444, Pinco, Palla, 3}

{111, Rossi, Mario 1}

{88888, Pinco, Pietro, 1}

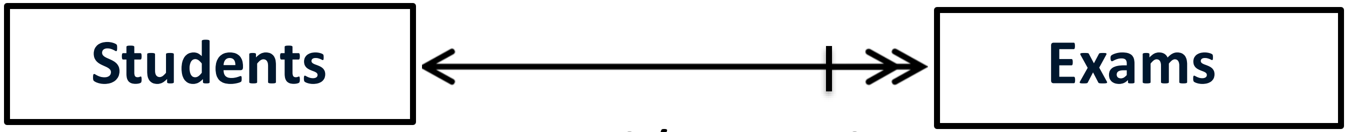
Teachers

# Cardinality (upper bound)

- Classification of the relationships wrt the cardinality
  - **One to One**: maximal cardinality equal to **1** for both classes
    - Manages[Managers, Departments]
      - Each manager must manage precisely one dept
      - Each dept has precisely one manager
  - **One to Many**: maximal cardinality equal to 1 for a class and **many (N)** for the other one
    - Owns[Persons, Cars]
      - A person might own more cars
      - A car has 1 owner
  - **Many to Many**: maximal cardinality equal to **N** for both classes
    - Teaching[Course, Teacher]
      - A teacher teaches more courses
      - A course can be taught by more professors (like this! :D)

# Cardinality (lower bound)

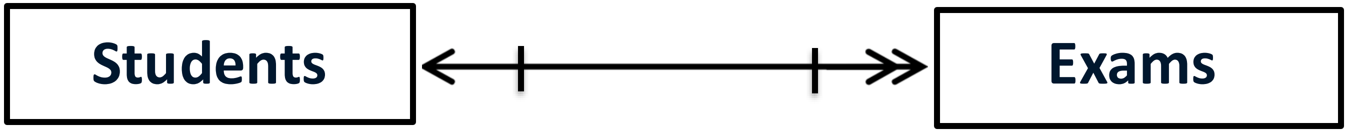
- Sixteen combinations:
  - One to many total/partial



Students

Exams

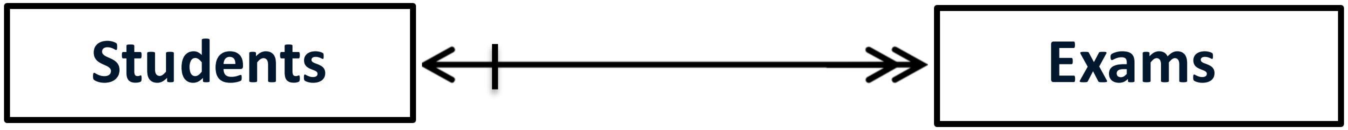
  - One to many partial/partial



Students

Exams

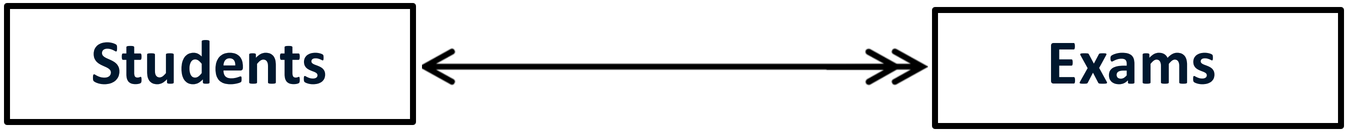
  - One to many partial/total



Students

Exams

  - One to many total/total



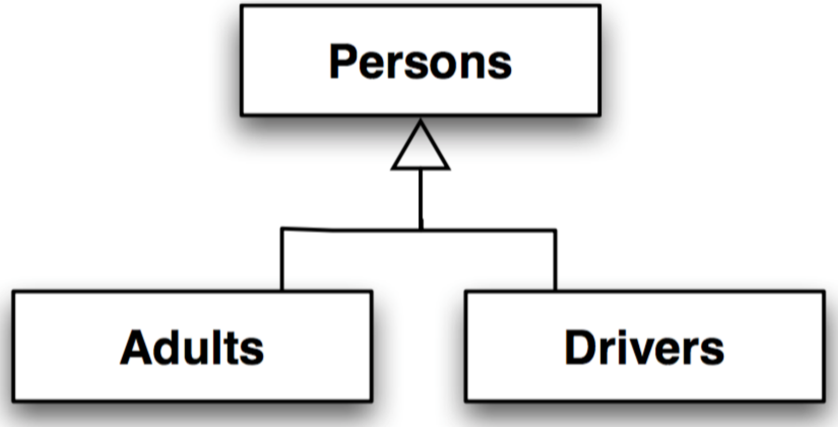
Students

Exams

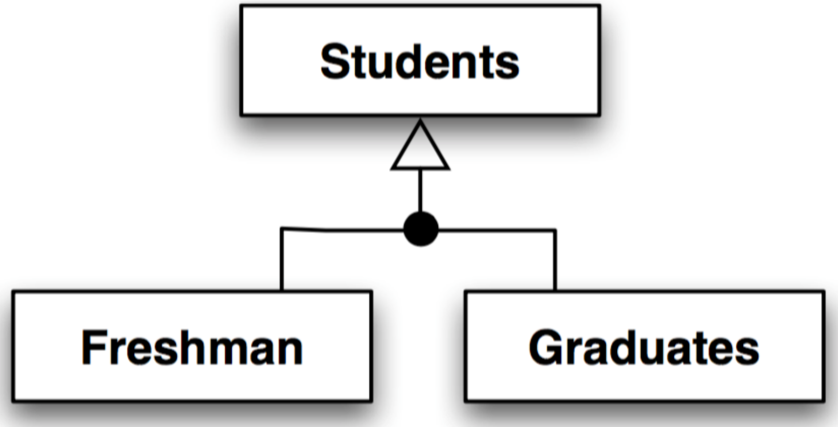
# Class Hierarchy

- A subclass:
  - a subset of class elements, for which we plan to collect more information:
  - ex: Students  is subclass of Persons
  - ex: **Internal** and **external** teachers are subclasses of the generic concept “**teacher**”

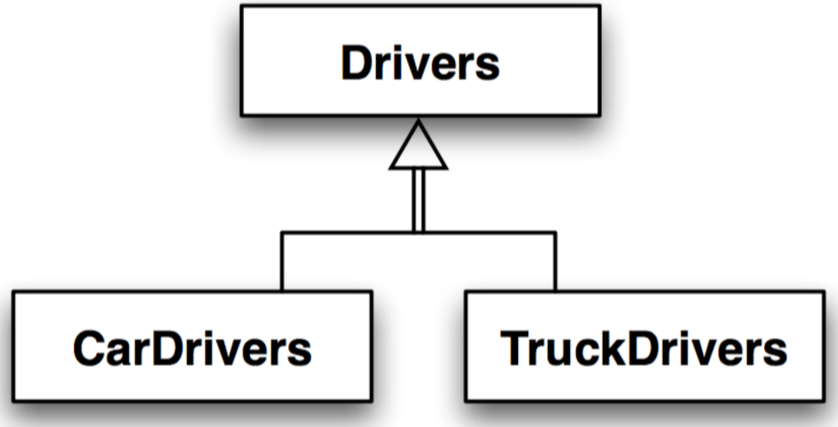
# Class Hierarchy



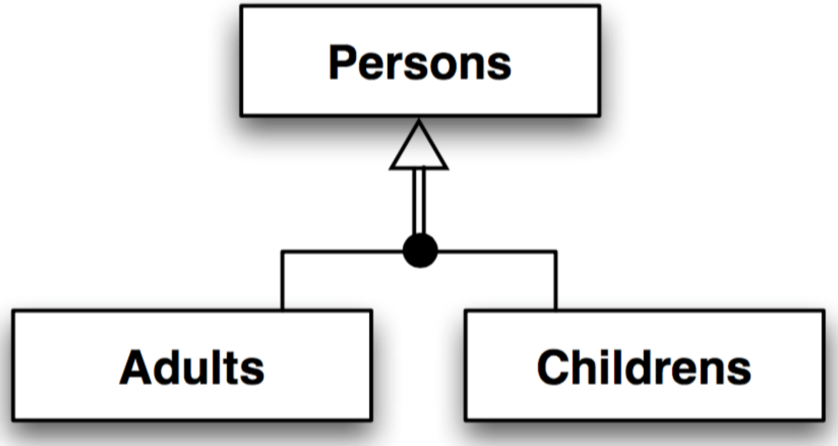
*Overlapping subsets*



*Non overlapping subsets*

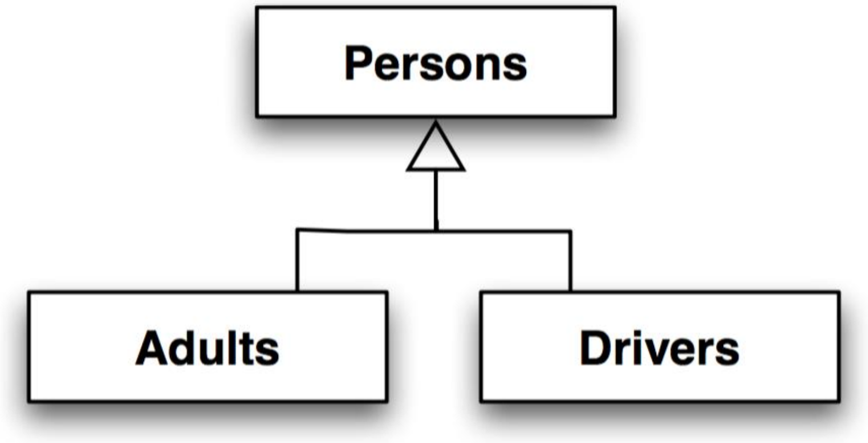


*Overlapping cover*

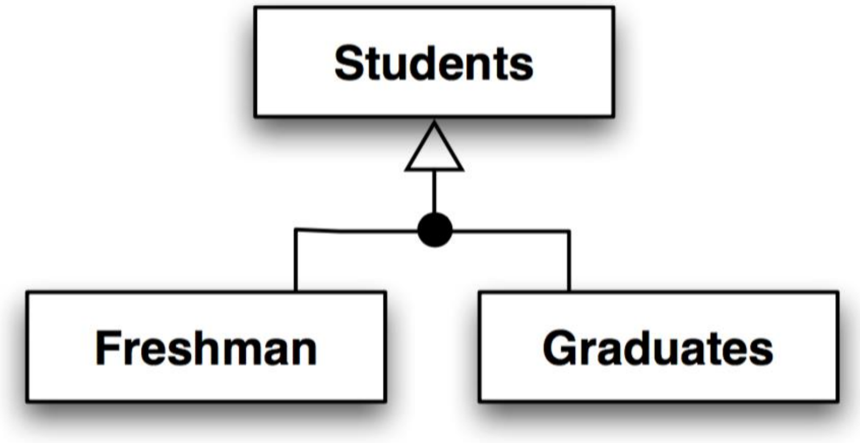


*Non overlapping cover*

# Class Hierarchy



*Overlapping subsets*



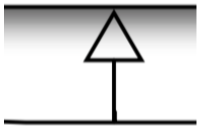
*Non overlapping subsets*

Adults and Drivers are **not disjoint** sets
*An adult can be a driver*

Freshman and graduates are **disjoint** sets
*A freshman cannot be a graduate*

Both refinements are **subsets**.
I.e., the sub-classes **do not include all** persons/students
- A Person can be a teenager
- A Student can be a second-year student

This is denoted by a single-line below the triangle



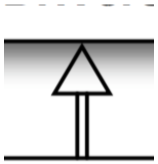
# Class Hierarchy

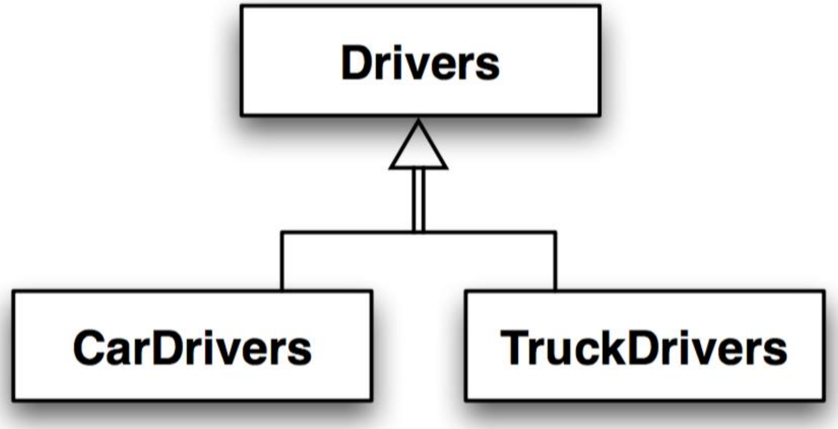
CarDrivers and TruckDrivers are **not disjoint** sets

Adults and Childrens are **disjoint** sets

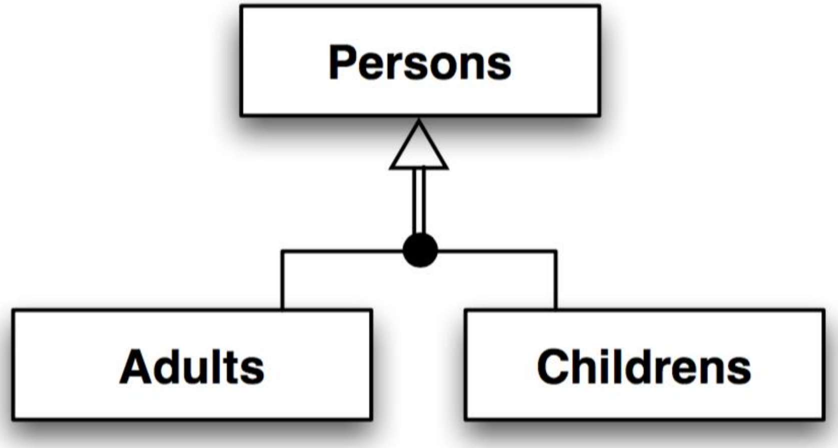
Both refinements are **coverings**
I.e., the sub-classes **do include all** drivers/students
- A Driver can only be a Car- or Truck-driver
- A Person can only be an Adult or Child

This is denoted by a double-line below the triangle





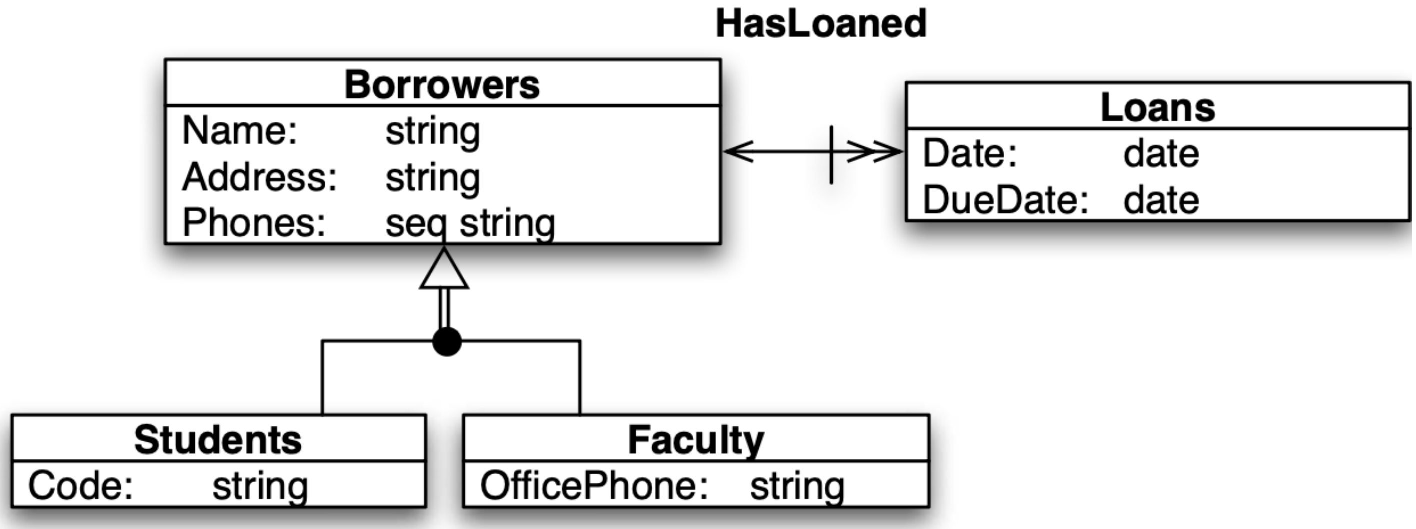
*Overlapping cover*



*Non overlapping cover*

# Subclassing + attributes!?

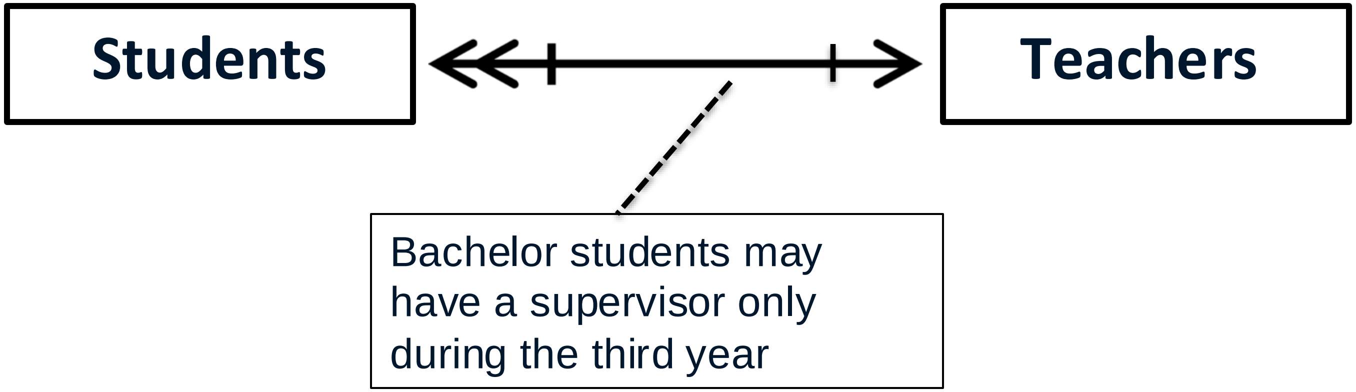
**Is it useful?**



Loans (of books) are done by Borrowers.
A Borrower can be either a Student, or a Faculty

# Notes

- Sometime it is necesary to add notes in the diagram to express some constraints
  - Ex: Bachelor students may ask a supervisor only when they are attending the third year.



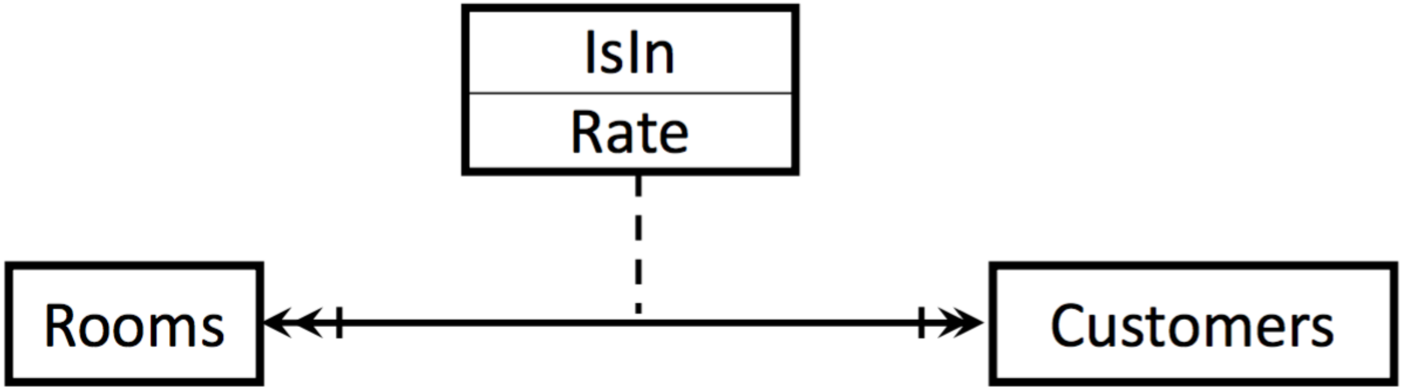
Students

Teachers

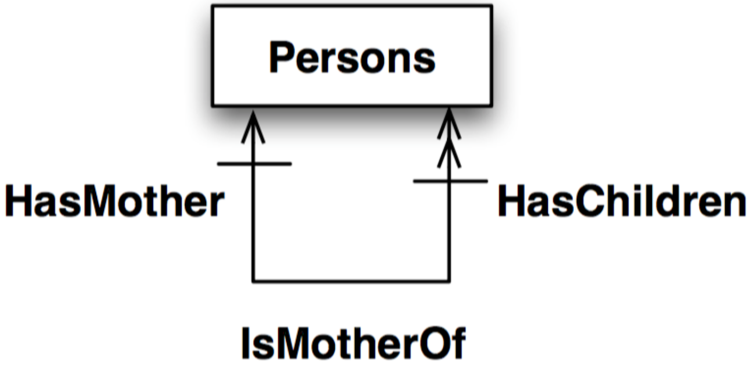
Bachelor students may have a supervisor only during the third year

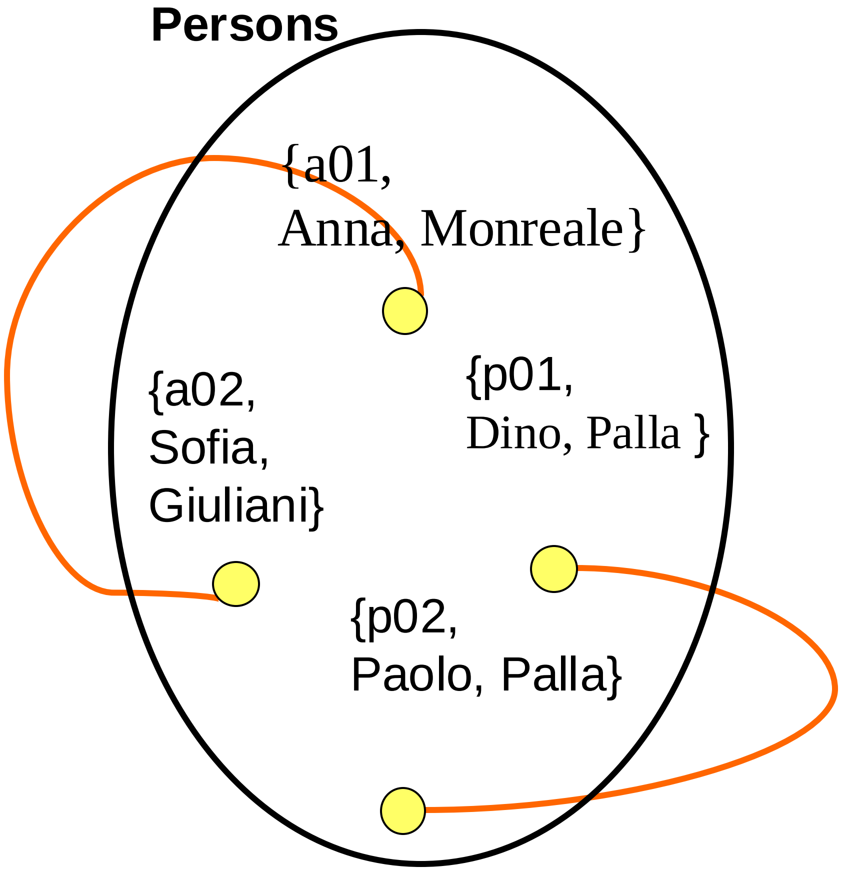
# Relationship with attributes

- Sometimes a relationship may have some properties that characterize each instance of the relationship
- **“John is occupying the room 105 at Le Meridien - Houston, at a $145 rate”**
- This is a relationship instance between persons and rooms, with a rate attribute



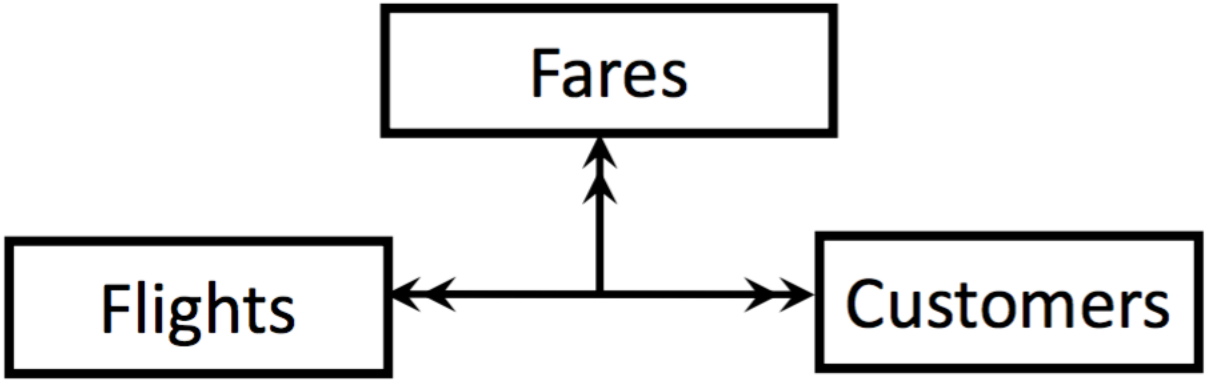
# Recursive Relationships





# Ternary Relationship

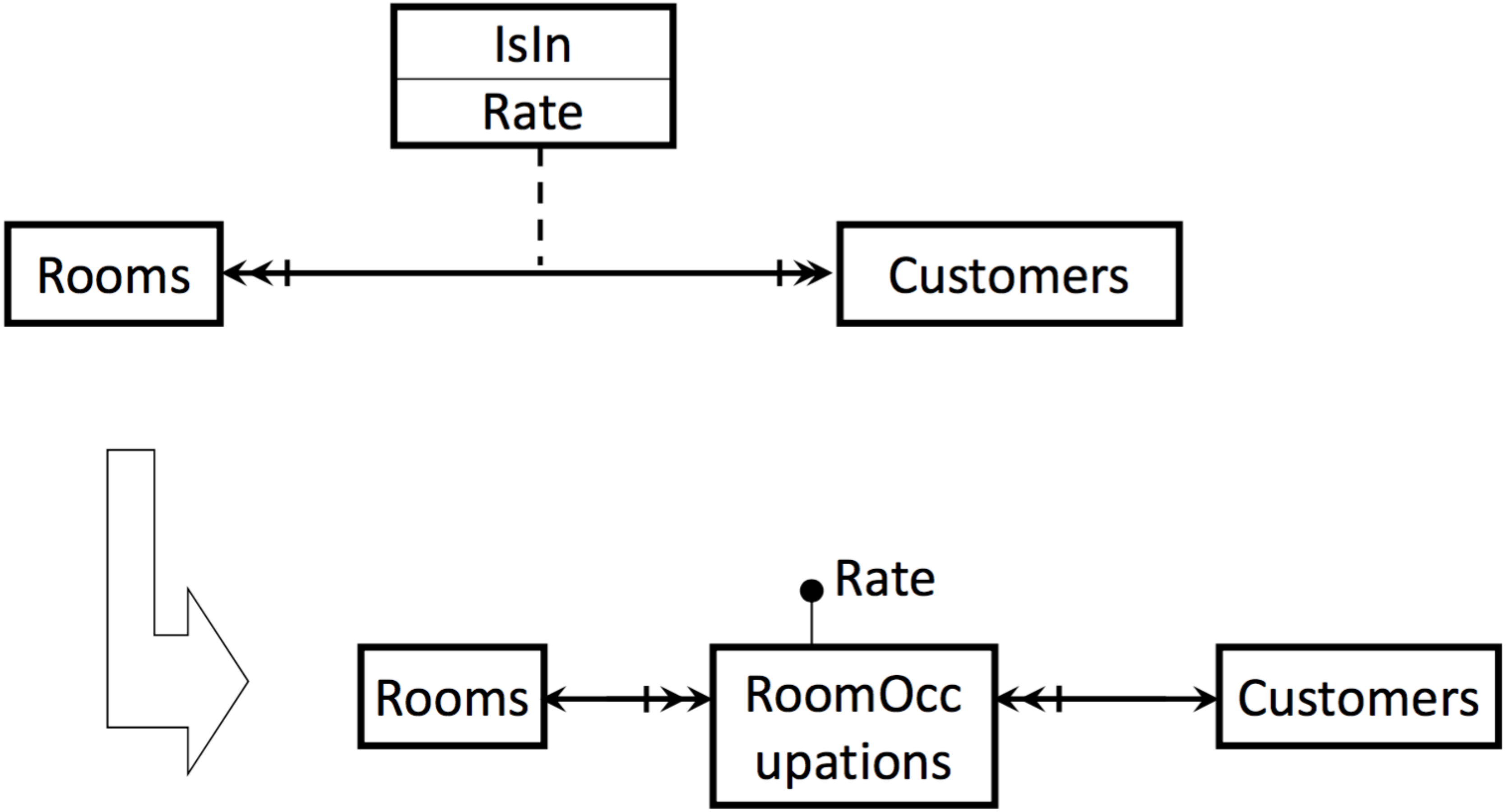
- Ternary facts exist also
- “John booked flight FK354/13-6-2000 with Y2 fare”



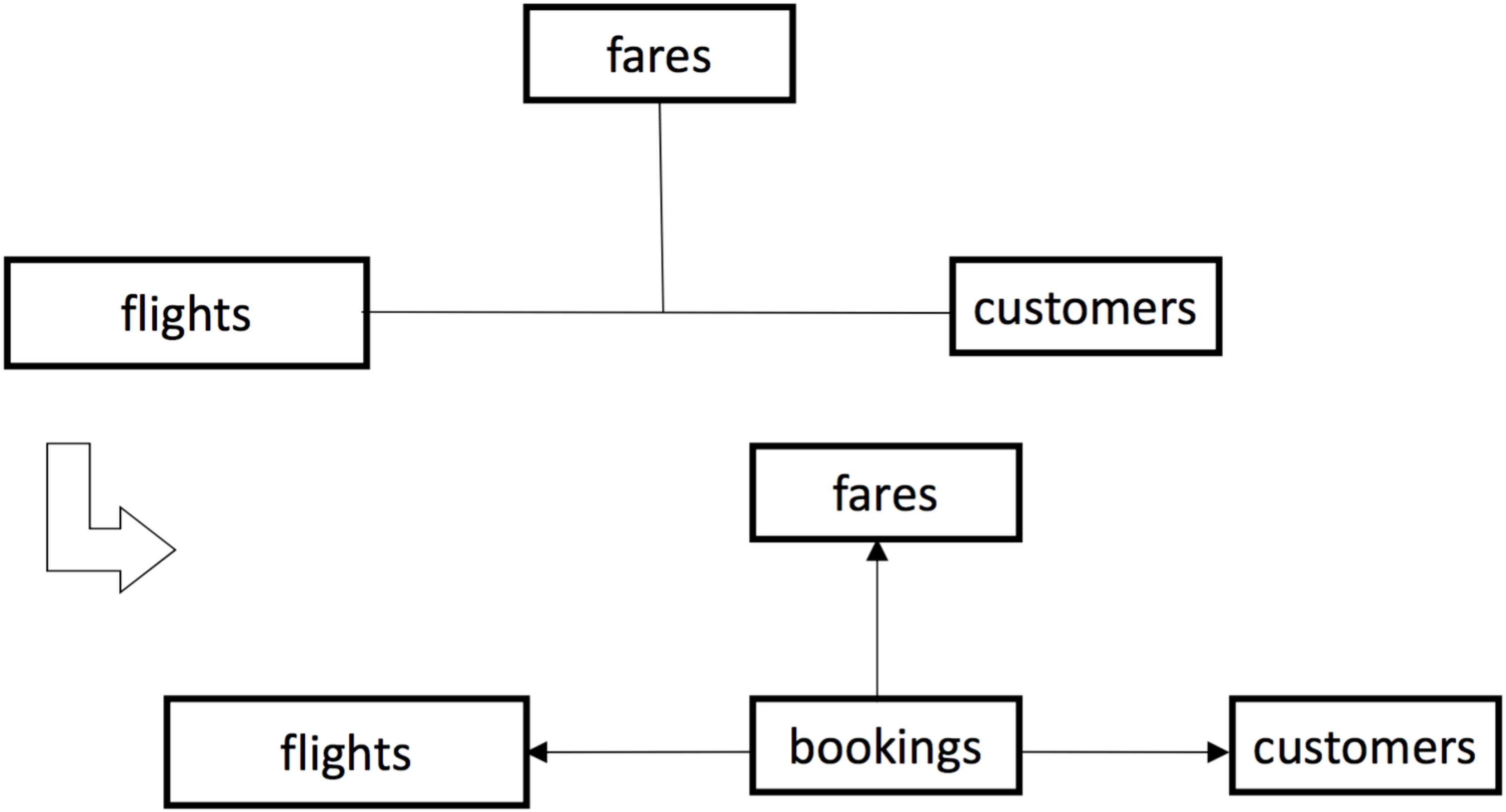
# Keep it simple
# KISS: Keep It Simple *Student*

- Whenever it makes sense, upgrade a **relationship with attributes**, or a **ternary** one, to a collection

# From Attributes to classes



# From ternary to new class


fares
flights
customers
fares
flights
bookings
customers