Basi di Dati

Lecturer: Prof. Gabriel Kuper

Assistent: Matteo Lissandrini

Hours: Tuesday (14:00-16:00) (Lectures: First two weeks)
Tuesday (14:00-16:00) (Exercises in Aula PC: From third week)
Thursday (14:00-16:00) (Lectures - all weeks)

Book: Database Systems: The Complete Book, by Garcia-Molina, Ullman and Widow (Pearson International). 2nd edition

Note: First edition is fine for this course, but not for Sistemi Informativi

Chapters: 1, 2.1-2.4, 3.1-3.4.1, 4.1-4.5, 5.1-5.2, 6, 7, 8, 9.3-9.4, 10.2, 14.3.1-14.3.3., 15.1-15.6, 16.2-16.7, 18.1-18.7

Outline of the course

Sep 17 Introduction to databases and the Entity-Relationship (ER) model

Sep 19 Relational model, conversion ER to relational and The relational algebra

Sep 24 Exercises: ER model

Sep 26 The extended relational algebra and introduction to SQL

Oct 1 Exercises: Algebra

Oct 3 SQL: Nested queries

Oct 8 Exercises SQL (1)

Oct 10 SQL: Schema definition and update

Oct 15 Exercises SQL (2)

Oct 17 SQL: Constraints and triggers

Oct 22 Exercises SQL (3)

Oct 24 SQL: Views, NULL values

Oct 29 Exercises SQL (4)

Oct 31 Recursion and FDs

Nov 5 Exercises SQL (5)

Nov 7 FD inference, Normalization and BCNF

Nov 12 Exercises on FDs and BCNF

Nov 14 Embedded SQL

Nov 19 (Lecture) Conversion of SQL to algebra and optimization

Nov 21 Optimization

Nov 26 Exercises on optimization

Nov 28 Transactions and serializability

Dec 3 Exercises on serializability

Dec 5 Locking and 2PL

Dec 10 Exercises on locking

Databases

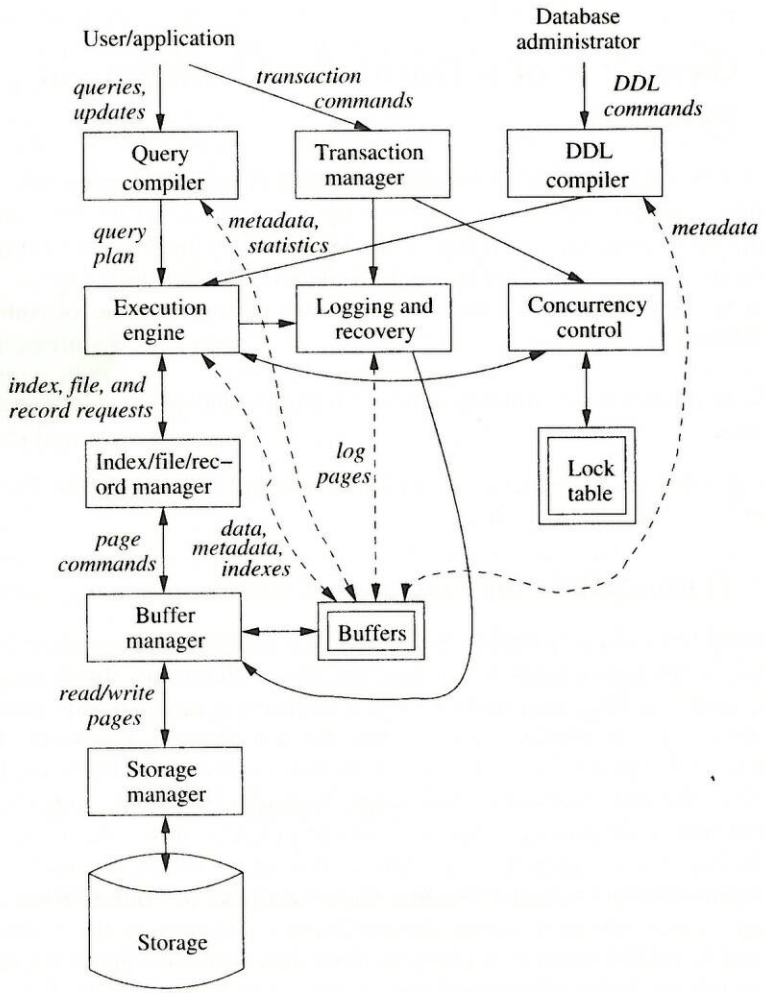
- Database: A large, integrated, collection of data
- Examples: Clients, products etc.
- Some databases (satellite data, scientific data, films) can be enormous
- A database (DB) normally models part of an organization using
  - Entities (e.g., students, courses)
  - Relations between entities (e.g.: Paolo takes the DB course, etc)
- A DBMS (Database management system) is a software system for representing and manipulating (querying, modifying) databases

Why use a DBMS?

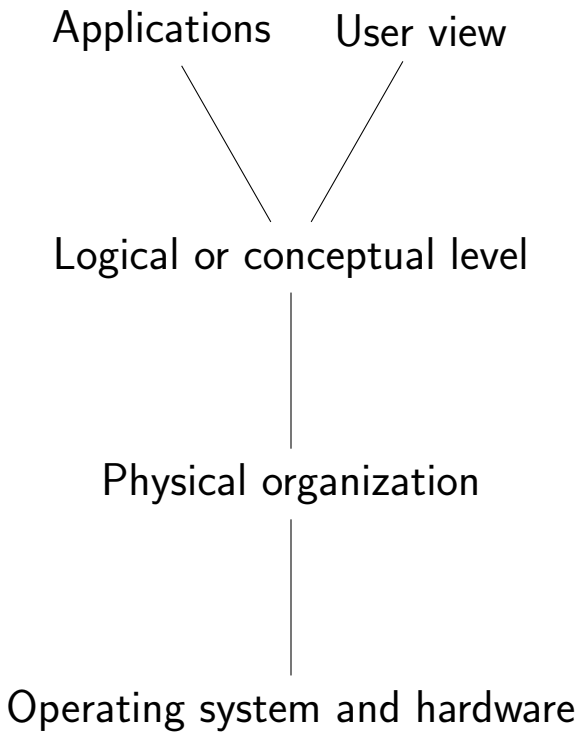
- **Data Independence and efficiency**. The user does not need to know how the database is physically organized. Queries are optimized automatically
- **Reduction of time need to develop applications.** Queries are written in a declarative language, and the user does not need to specify how they should be evaluated
- **Data integrity and security**. Constraints on data are enforced automatically
- **Concurrent access, recovery from errors**. Many user may access the system at the same time, with no problems arising from their interaction

Data models

- A **data model** is a collection of concepts for describing the data
- A **database schema** is a description of the data that will be stored in a specific data model
- The **relational data model** is the most commonly used data model. The basic concepts are:
  - A **relation**: A table with rows (called "tuples") and columns (called "attributes")
  - A **relational schema** describes the structure of the tables in a specific database.


User/application
Database administrator
queries, updates
transaction commands
DDL commands
Query compiler
Transaction manager
DDL compiler
query plan
metadata, statistics
metadata
Execution engine
Logging and recovery
Concurrency control
index, file, and record requests
log pages
Lock table
Index/file/record manager
page commands
data, metadata, indexes
Buffer manager
Buffers
read/write pages
Storage manager
Storage

Abstraction levels



Logical level

- Data models. For example
  - **Relational**
  - Object-oriented
  - Others: Hierachical (ancient), semi-structured (XML) etc.
- Data Definition Language (DDL): Language used by the administrator to describe the structure of the data
- Data Manipulation Language (DML): Language used by users to access the data
- In modern relational systems these are both part of SQL, but access to the DDL may be restricted to administrators

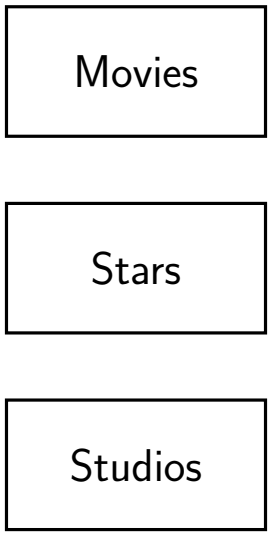
ER (Entity-Relational) model

- **Not** part of the relational model
- Used as a tool for designing a relational schema for an application
- After the design stage, the ER diagram is converted into a representation as a relational database
- Possible linguistic confusion: "Relation" in the relational model vs "Relationship" in the ER model (same word in Italian...)
- ER model: Graphical representaion of the database part of an application (also part of UML)

ER model

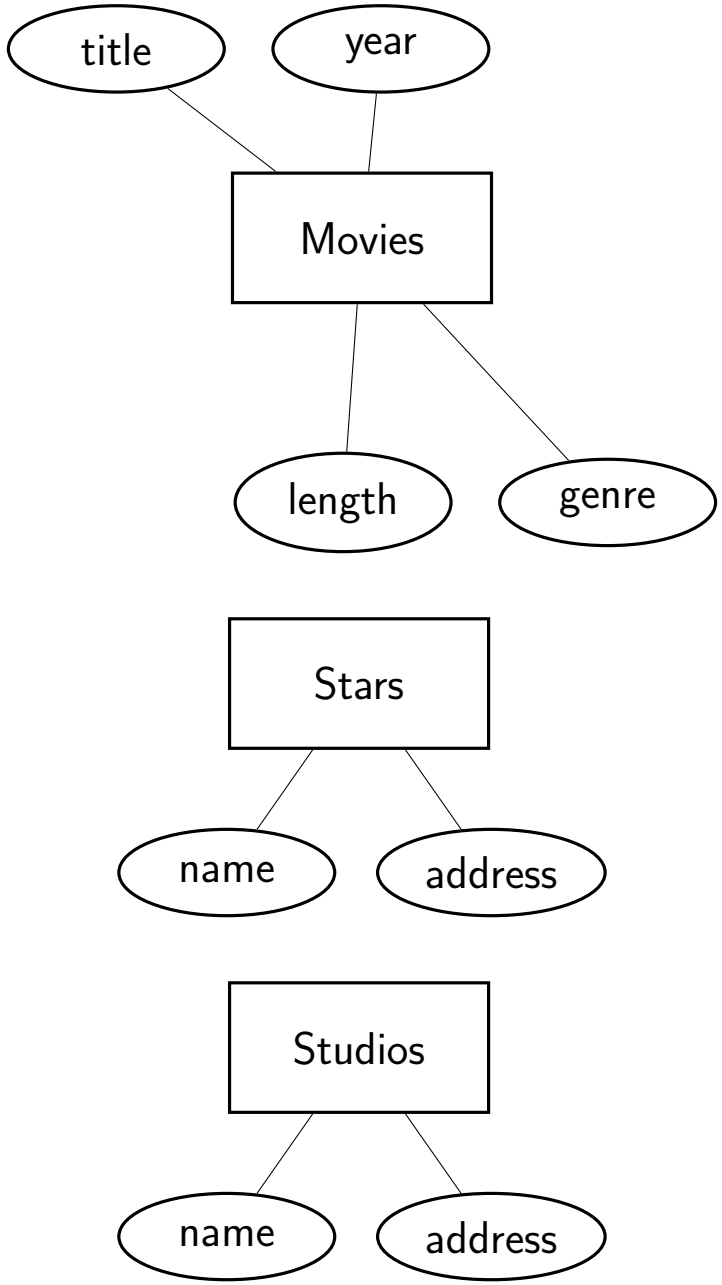
- Entities: Collections of objects of a "similar" nature
- Attributes. Properties of entities
- Relationships. Relations between entities
- Concept of entity is a bit vague. What is an entity depends on the application

Examples of Entities



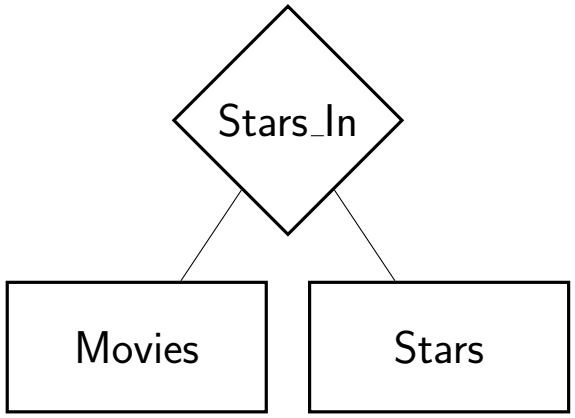
Attributes

- Attribute: A property of an entity
- Examples:
  - Movies: *title*, *year*, *length*, *genre*, ...
  - Stars: *name*, *address*
  - Studios: *name*, *address*
- Attributes will be drawn inside ovals, with lines connecting them to the relevant entity
- Note on notation: ER model is not standardized. Other symbols (such as those in UML) are acceptable for this course

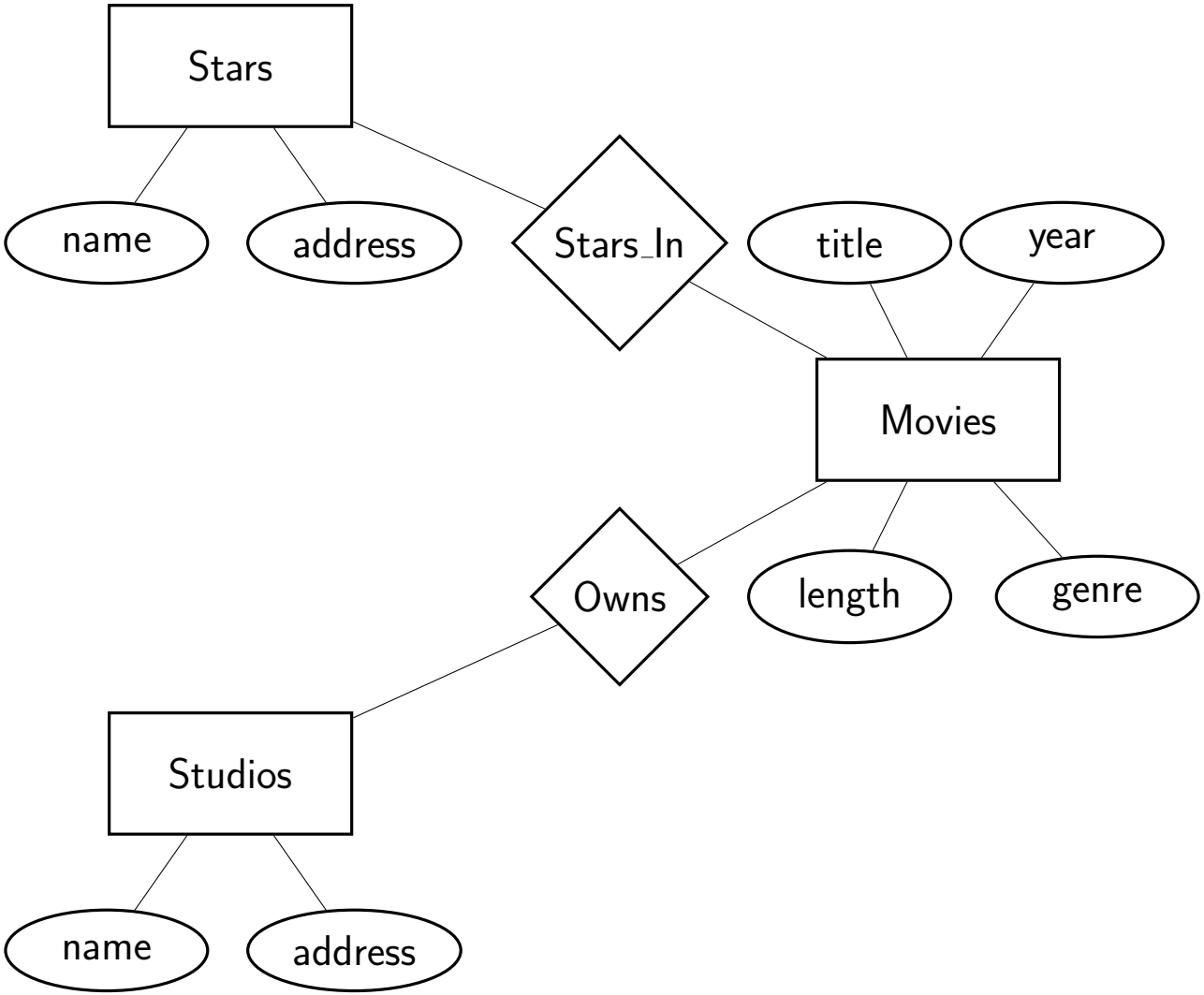

title
year
Movies
length
genre
Stars
name
address
Studios
name
address

Relationship

- A relationship links two or more entites.
- For example `Stars_In` relates movies to stars that appear in them

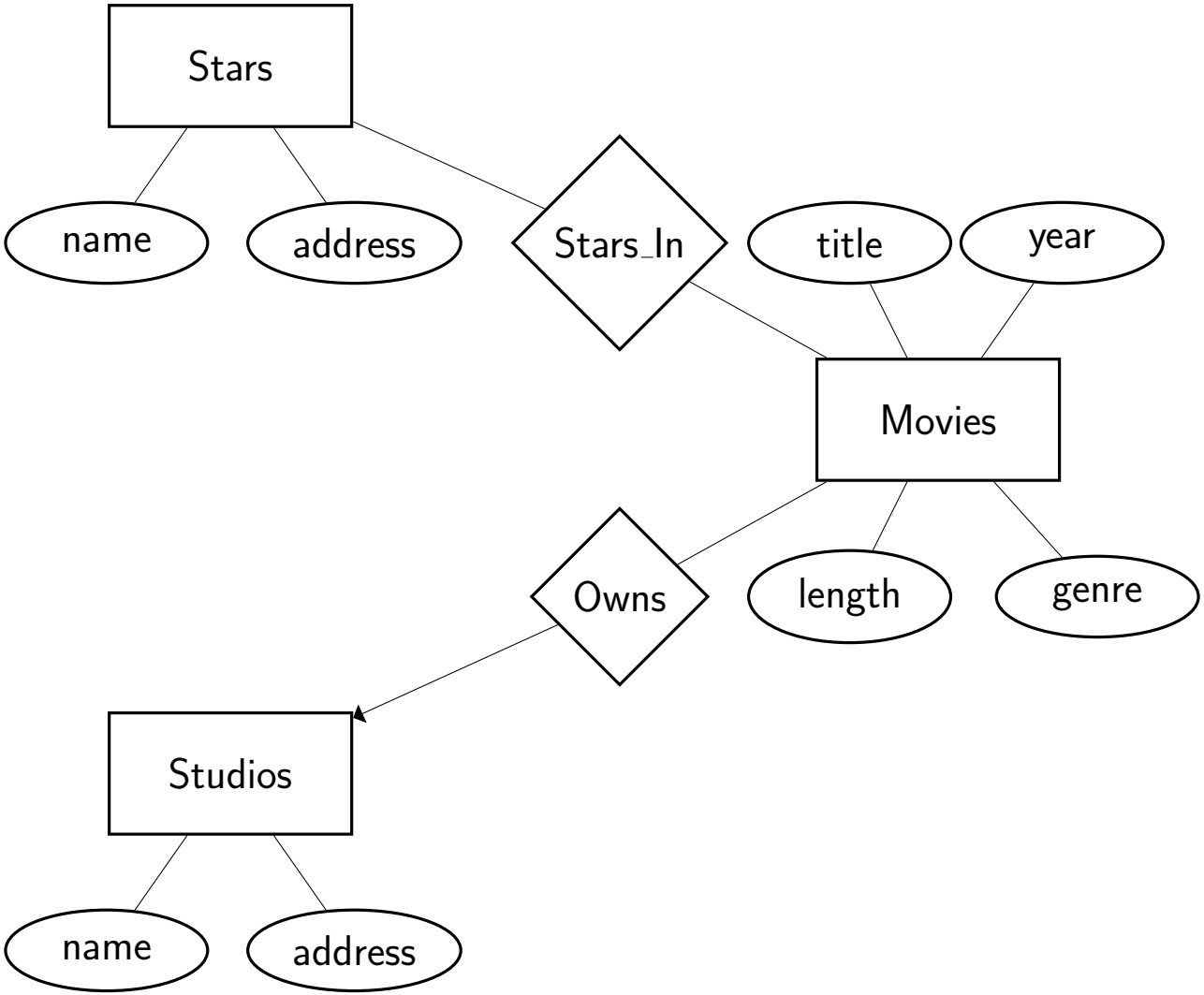


- In the next slide, we introduce a new entity `Studio` and a relationship `Owns` between films and studios

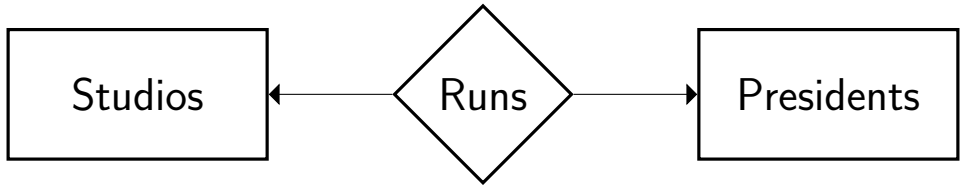

Stars
name
address
Stars_In
title
year
Movies
Owns
length
genre
Studios
name
address

Instances

- Instance of an ER diagram
  - Assign to each entity E a set of objects $\{e_1, \ldots, e_n\}$
  - Assign to each entity a list of values for all the attributes
  - Assign to each relationship R between entities E and F a set of pairs of objects in the corresponding entities $\{(e_1, f_1), (e_2, f_2), \ldots\}$
- If each member of E can only be related via R to a single member of F, R is called "many-to-one" from E to F.
- Note that this does not mean that every object in E is related to exactly one object in F, only that it is not related to **more** than one
- If R is many-to-one from E to F **and** from F to E, R is called "one-to-one"
- The ER model also lets us specify "exactly one", cardinality constraints and other properties of the model, which we will not discuss here
- In the next slide, we add the fact that every film is owned by (at most) one studio


Stars
name
address
Stars_In
title
year
Movies
Owns
length
genre
Studios
name
address

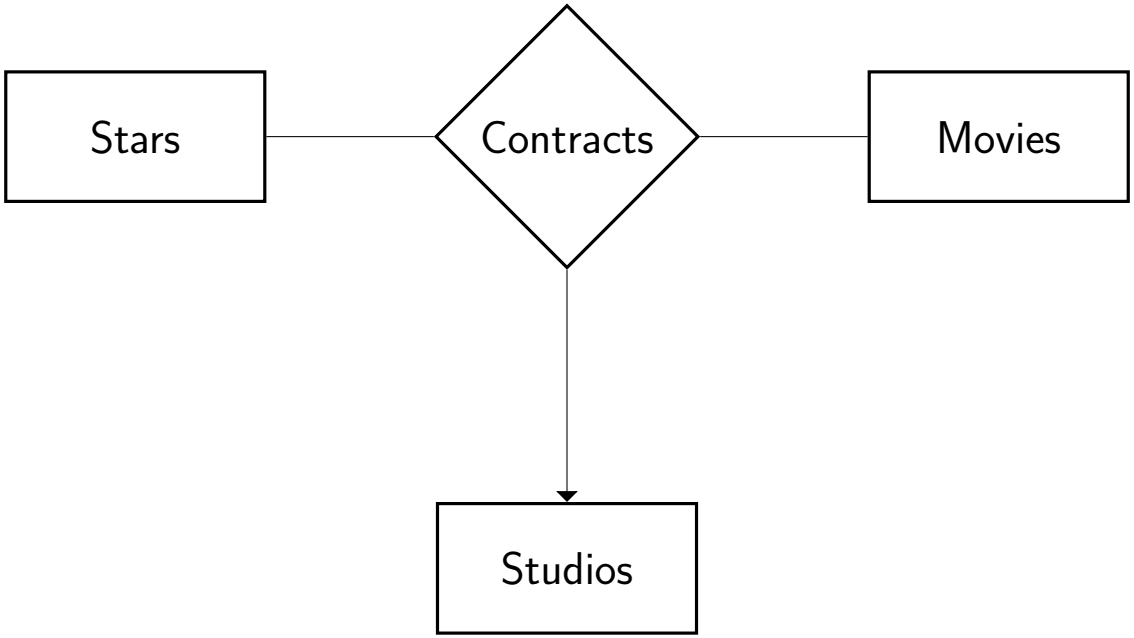
One-to-one Relationships



- If we have a one-to-one relationship, one should consider whether it makes sense to combine the two entities (in this case, probably not)
- Combining the entities saves space and complexity, since there is no need to represent the relationship
- Note that in this example, a studio is allowed not to have a president, and there may be a president who does not have a studio. If we want to disallow this, one must use additional constraints

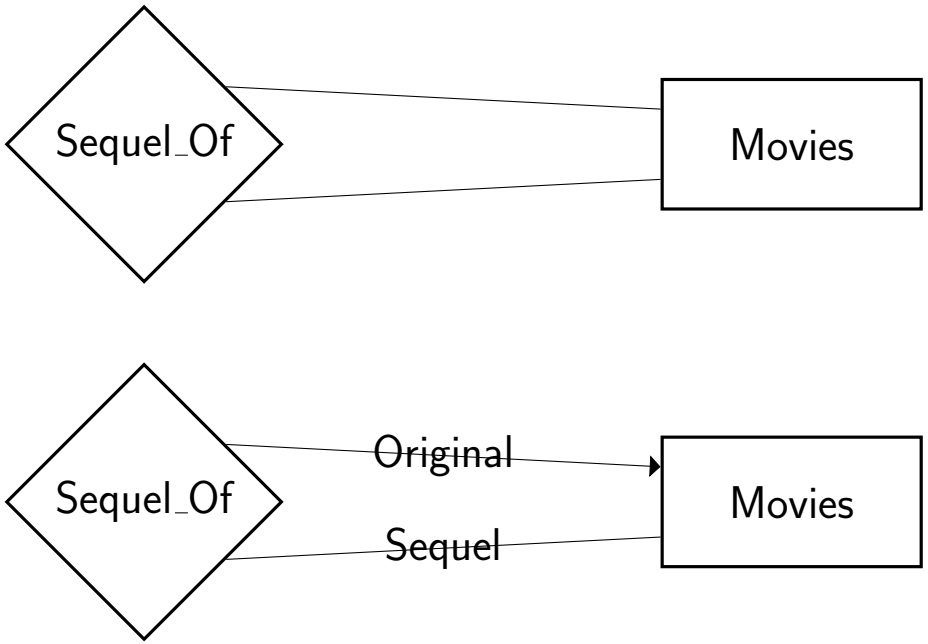
Relationships between 3 or more entities

- Contracts. Each contract is between a "Star" and a "Studio" to make a "Movie"
- How do we model this?
- Relationship between "Movie" and "Studio"? No, as there is a separate contract for each "Star"
- Relationship betweem "Star" and "Studio"? No, since the "Star" may participate in more than one film
- Note however that all contracts for a film are with the same studio


Stars
Contracts
Movies
Studios

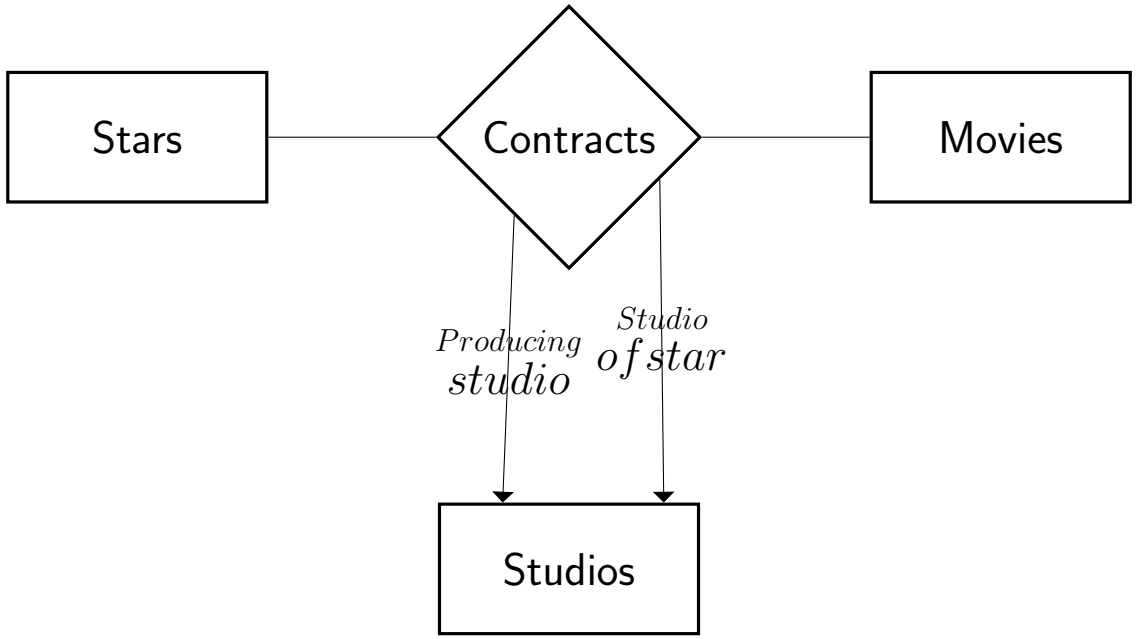
Roles in relationships

- Entity: Movies.
- Relationship: Sequel_Of



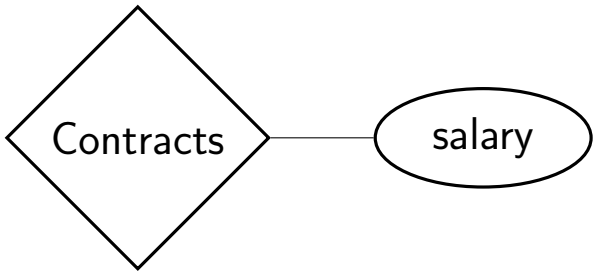
Another example: Four entities

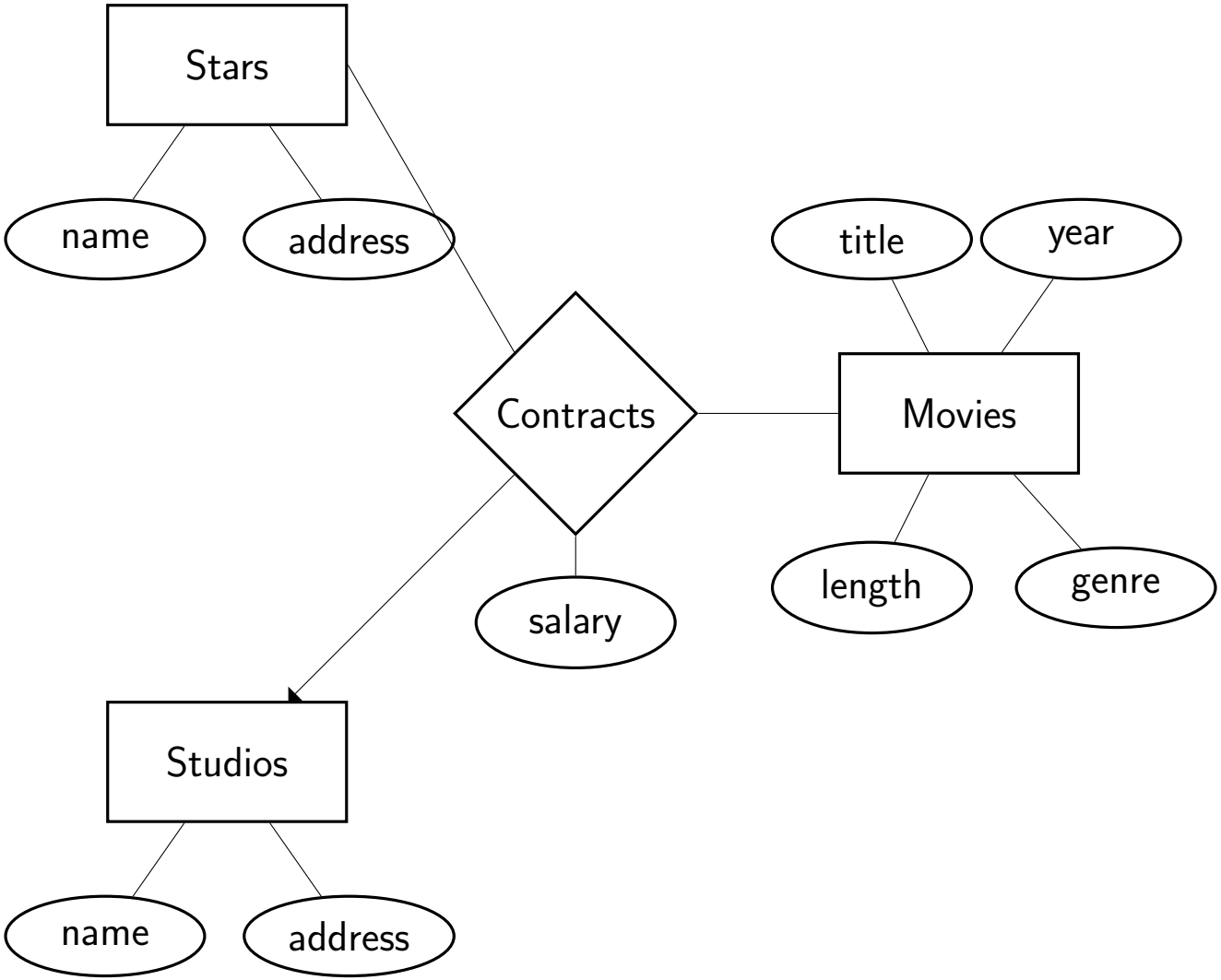
- Assume that each "Star" is represented by a "Studio" (which has nothing to do with the "Movie"
- A contract is between a (1) Movie (2) Star (3) Studio producing the Movie and (4) Studio representing the Star



Relationships can have attributes

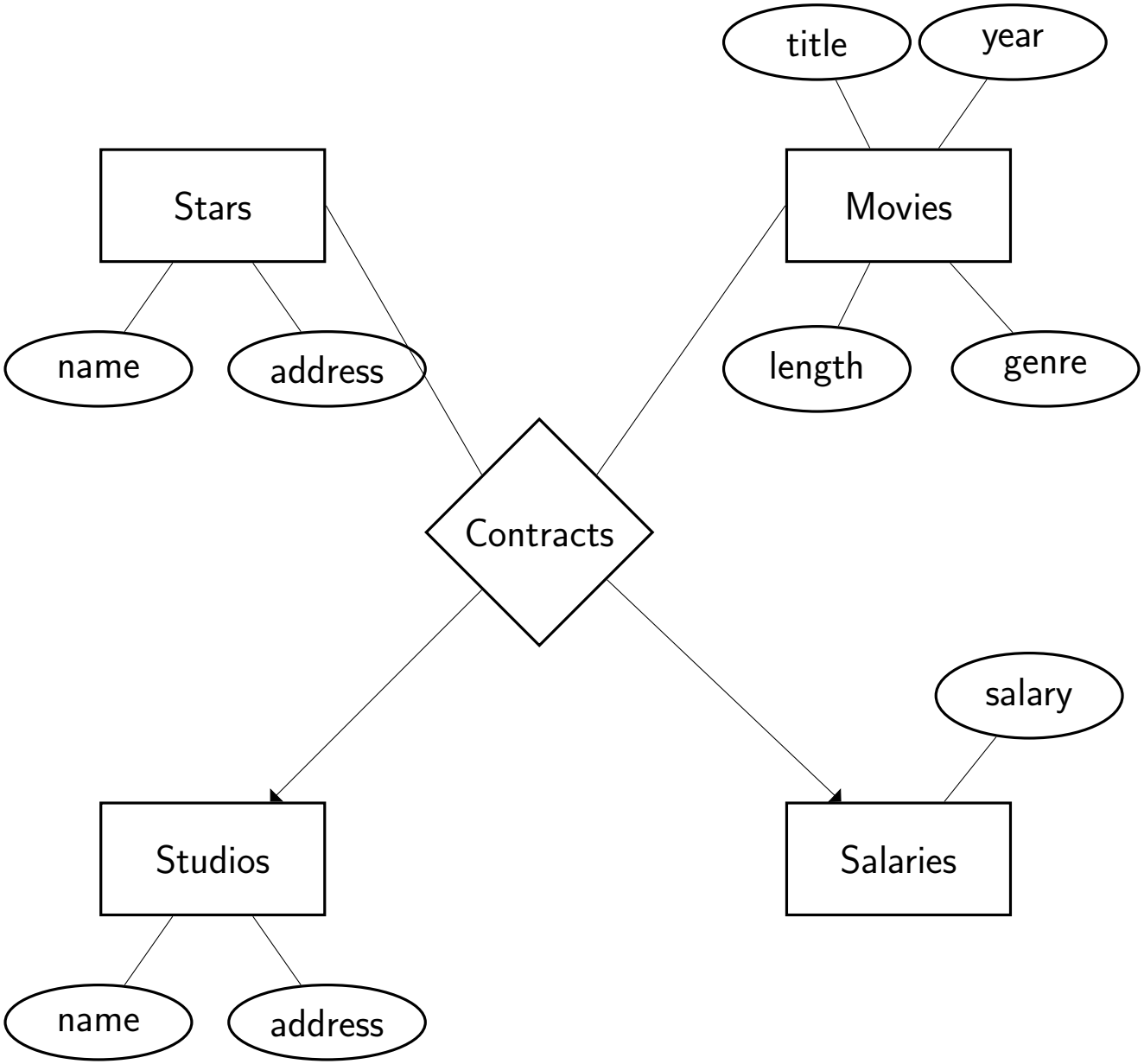
- Salary: Payment to "Star" for a film
- Is part of the contract
- Cannot be modelled as an attribute of any entity
- It is an attribute **of the relationship** contracts




Stars
name
address
Contracts
salary
Studios
name
address
title
year
Movies
length
genre

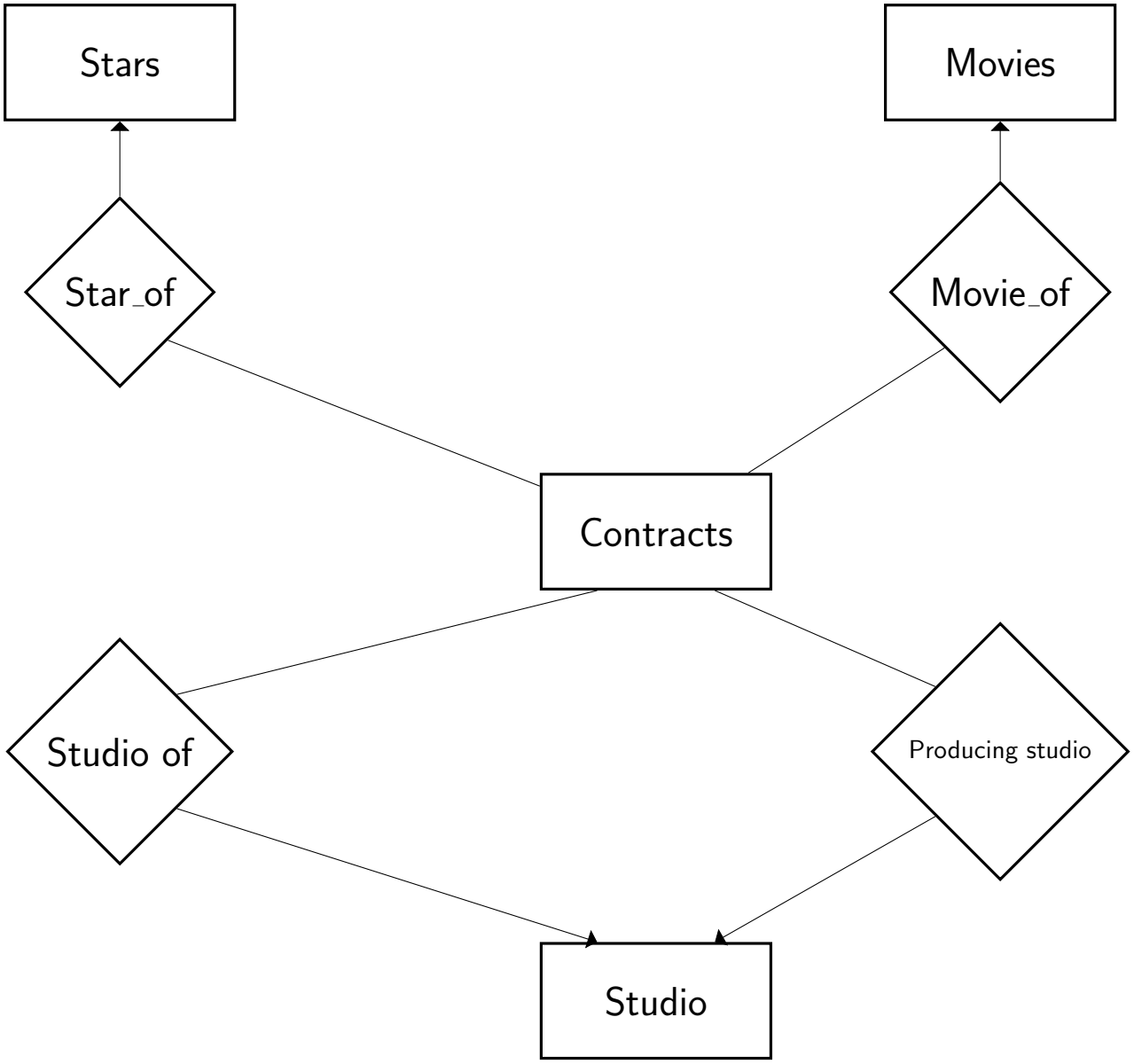
Elimination of attributes on relationships

- These attributes can always be eliminated
- In order to eliminate them, one must introduce new entities
- These entities my be rather artifical
- In our example, we introduce an entity “Salaries” with the attribute “salary”.


title
year
Stars
Movies
name
address
length
genre
Contracts
salary
Studios
Salaries
name
address

Elimination of multi-way relationships

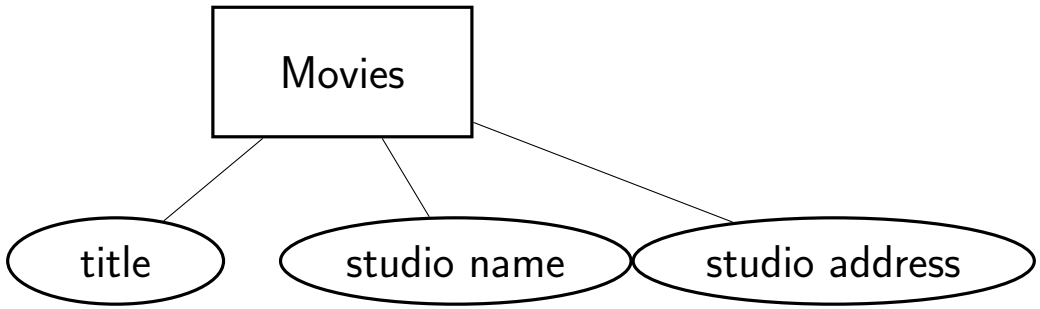
- We study this for two reasons
  - For better understanding of the ER model
  - There are implementations of the ER model that do not allow such relationships
- To eliminate them: Introduce a new entity for each such relationship
- Then introduce new relationships (many-to-one) between this entity and all original ones
- Our example: Introduce a new entity "Contracts" (which might have been a reasonable choice from the start)


Stars
Movies
Star_of
Movie_of
Contracts
Studio of
Producing studio
Studio

Design principles

- Faithfulness to the application
  - Attributes and relationships that make sense
  - Don't combine entities that represent different concepts or use seperate entities for part of the same thing
  - Example: University. Professors and Students. Are they different entities?
    - If the data is significantly different: students have courses and grades, professors have publications, use separate entities
    - If the data is similar: name, address, phone, etc, use one entity
  - Non always clear how to model relationship: Is a relationship between courses and teachers many-to-one? This may depend on the university rules, i.e., on the application
- Minimimize the information stored, but not excessively. From the Contracts relationship we can determine that certain films are owned by certain studios. Does that mean that we can eliminate relationship "Owns"? Non neccessarily. This depends on whether we are sure that we will have contract information for every Film in the database

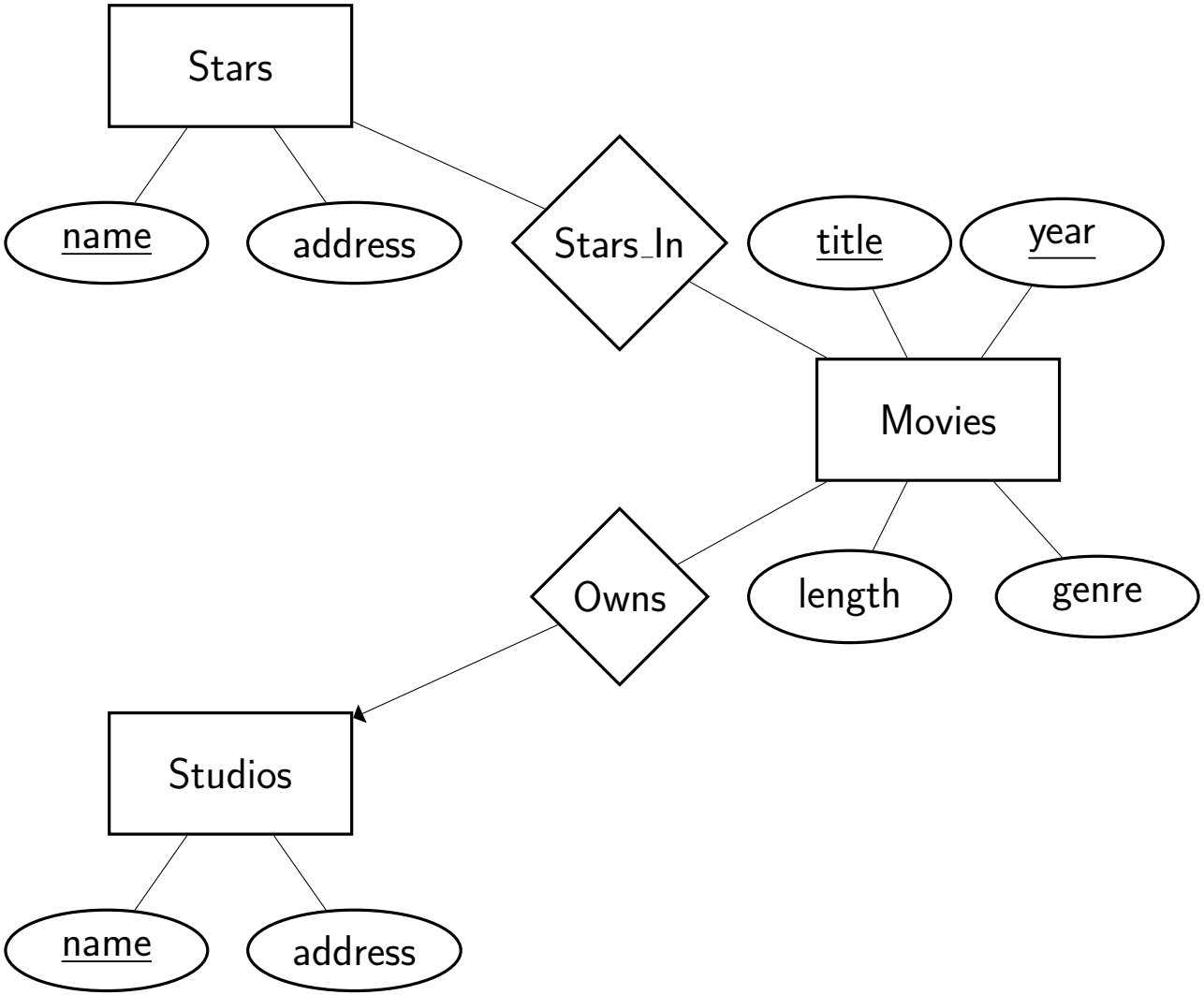
Do not repeat information



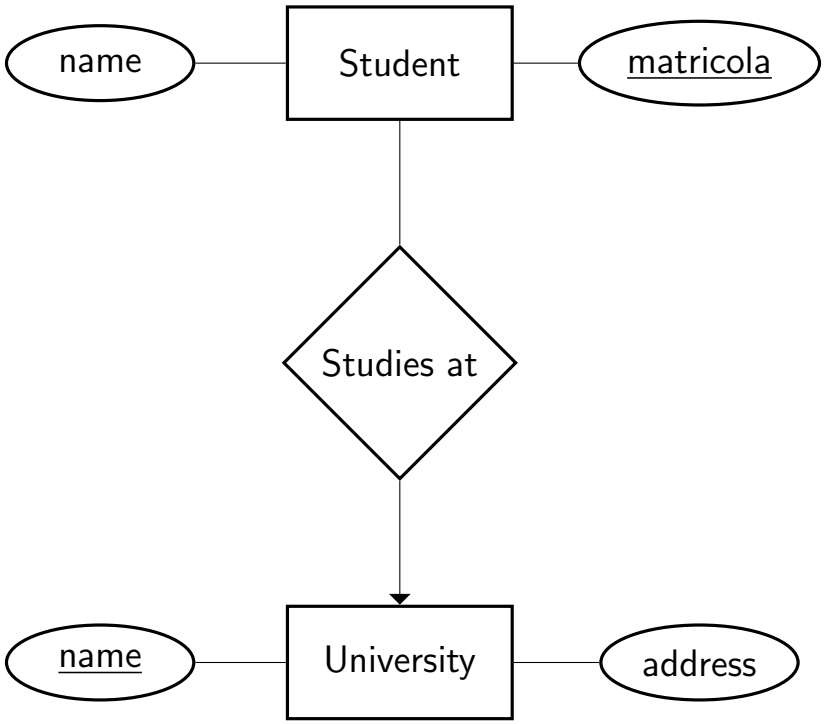
- In this example, we have eliminated the entity "Studio"
- When a relationship is many-to-one, we can always do this
- But: We must repeat the name, and all other attributes, of the Studio for every object in the class "Movies"
- Even worse: If the studio address changes, we must modify many entities in "Movies"
- In practice, one often forgets to update all of them, and we get an inconsistent database (called "update anomally").

Constraints: Keys

- Constraints: Conditions that the data must **always** satisfy
- The most important class: Key constraints
- A **Key** is an attribute, or a list of attributes that identify a unique entity
- For example: An entity "Students" for University of Trento) has 2 possibile keys, matricola number or codice fiscale
- In our example
  - Studios: Key "name"
  - Stars: Key "name"
  - Film: The key will be the pair of"title" **and** "year", since we assume that the title is not sufficient to identify a film: (King Kong, 1933) and (King Kong, 2007)
- Every entity **must have a key**
- Only one key is allowed for each entity: We must chose between matricola and codice fiscale above
- In ER diagrams, keys are underlined


Stars
name
address
Stars_In
title
year
Movies
Owns
length
genre
Studios
name
address

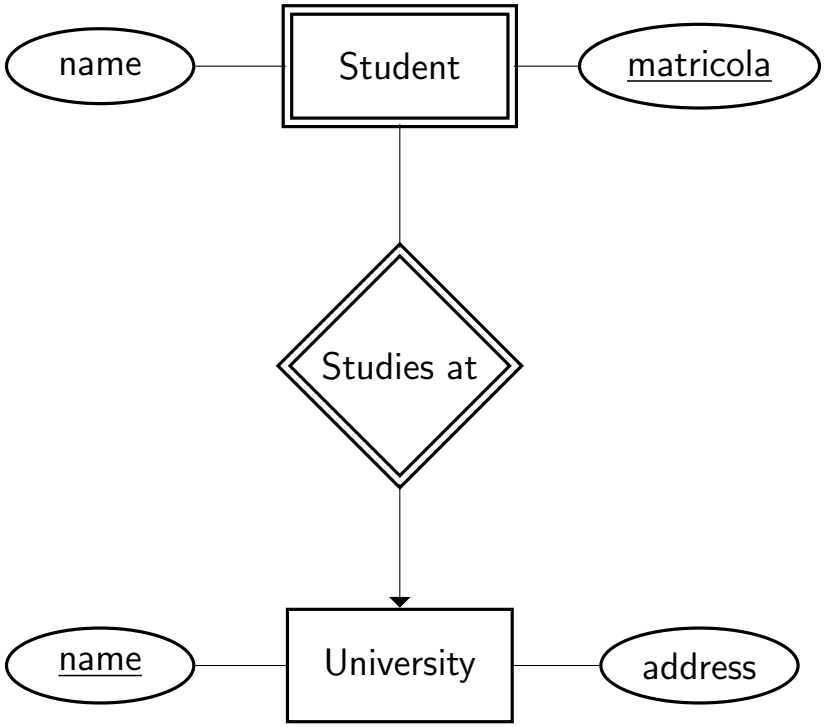
Weak entities



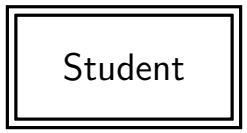
- What is a key for "Student"?

Keys for weak entities

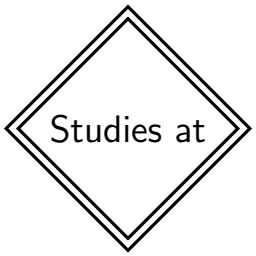
- Matricola identifies a student at the University of Trento.
- But without knowing the university, the matricola is not sufficient to identify a student
- A student can be identified by
  1. A key for "Student" relative to the university: "matricola"
  2. The identity of the university, which can be done via a key ("name") for this entity
- Weak entity: Part of the key identifying the entity is in another entity class

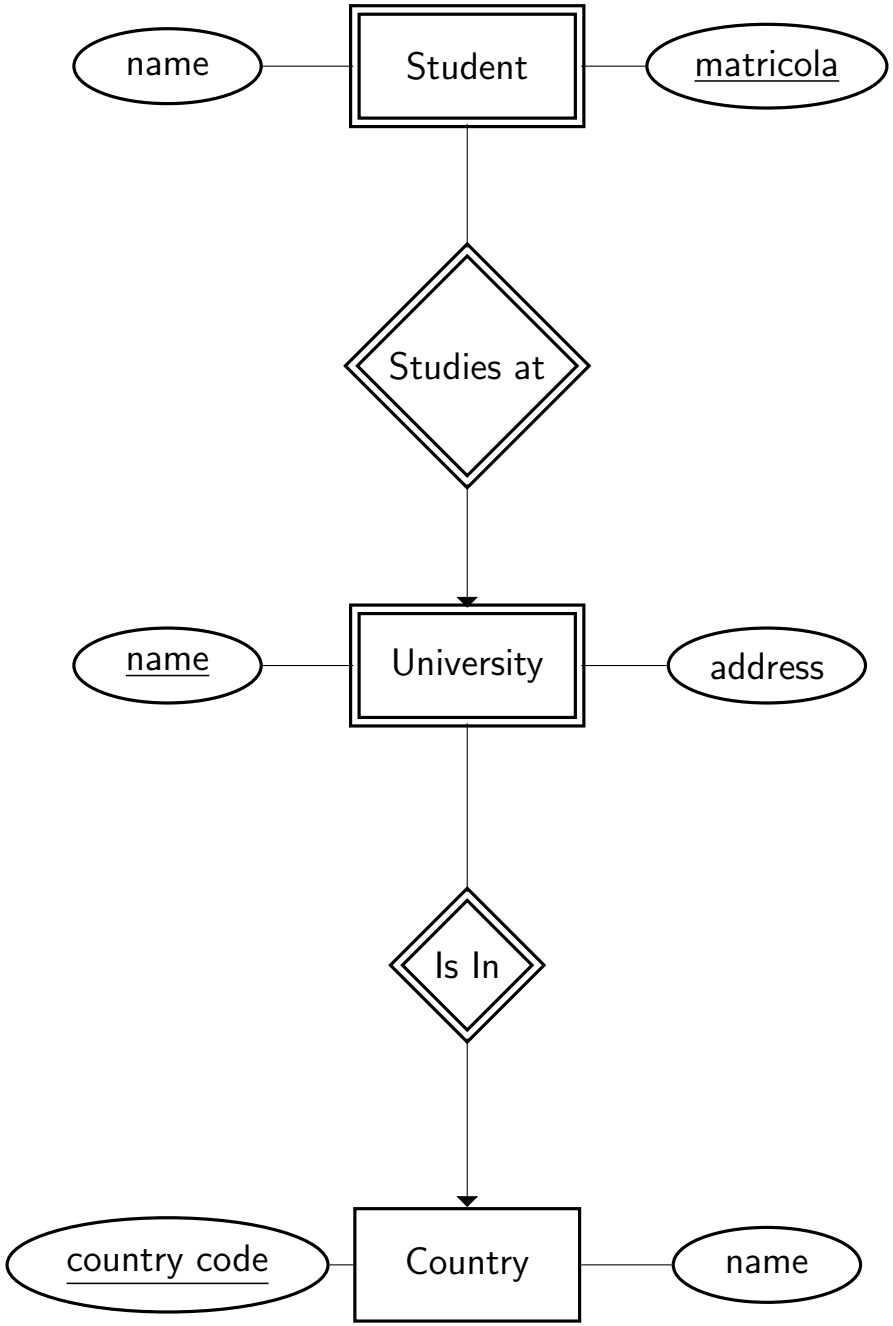

name
Student
matricola
Studies at
name
University
address

Weak entity



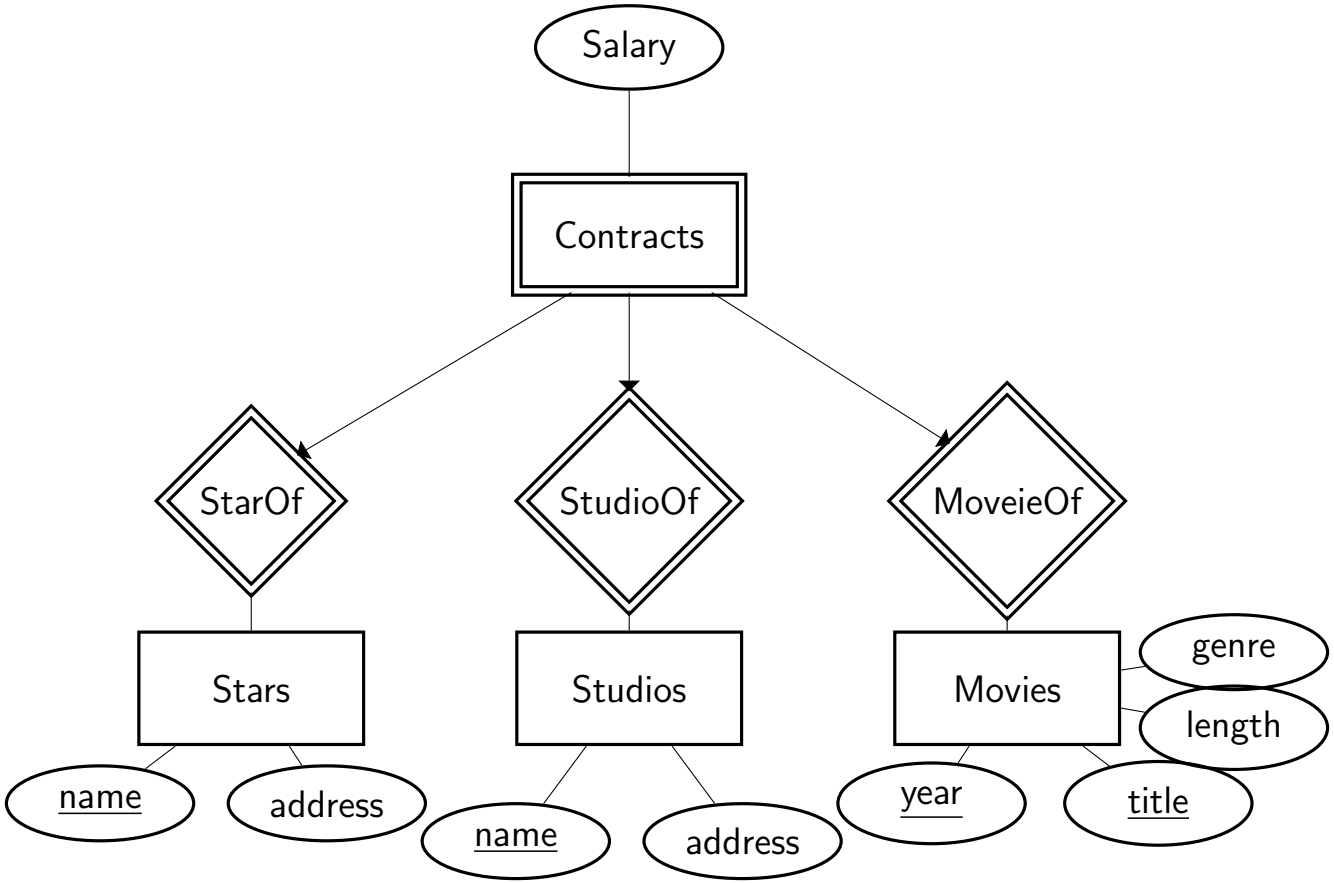
Supporting relationship




name
Student
matricola
Studies at
name
University
address
Is In
country code
Country
name

Another example

- We made “contacts” into an entity in an example above
- But we didn't mention what the keys where. What are the possibilities?
- One possibility: Introduce an artificial attribute for the contract number
- May even be a natural choice, such as the number on the form
- Alternative: Make “contracts” into a weak entity


Salary
Contracts
StarOf
StudioOf
MoveieOf
Stars
Studios
Movies
genre
length
name
address
name
address
year
title

Introduction to the Relational Model

- What is the relational model?
- Basic concepts
- Brief history
- How to convert an ER diagram to a relational schema

The Relational model

- Data is stored in tables, called **relations**
- Each table has a pre-defined structure, with a fixed number of columns, each labelled by a different **attribute**
- Each attribute is declared to be from a specific domain (**string, integer, boolean, etc.**) and all values in this column must be of that type
- Columns are, in principle, unordered, and can be referred to only by their attribute (minor exception: commands for inserting data)
- The number of columns/attributes in a relation is called its **arity**. (Binary, ternary relations)
- A row in a relation is called a **tuple**
- Relations are **sets** of tuples, i.e., unordered without duplicates. Again, this is the abstract principle; we shall see exceptions later

Introduction of the relational model

- Introduced by Codd at IBM (*A Relational Model of Data for Large Shared Data Banks*, 1970)
- Basic idea is to represent data by tables
- Codd describes two languages, (1) a **procedural** language, called the relational algebra, it which you can describe how the computer can evaluate a query and (2) a **declarative** language, called the tuple calculus (which eventually became SQL), in which the user can specify what the result of the query should be, without saying how it should be computed
- We will study one of each: the relational algebra and SQL
- Main result of his paper: Both languages are equivalent, and there are effective algorithms to convert from one to another
- How does this work? You may have seen other "declarative" languages, and they don't have nice properties like this
- Key idea: Restrict what the language can express. This is one of the few languages that you will see, that is not Turing complete
- In fact, we also want very low complexity. Relational languages are usually linear (for those of you who have studied complexity, you might have seen the class called LOGSPACE)