Conceptual Database Design

Entity/Relationship (E/R) Model

Database Design

- Requirements collection and analysis
  - Determine what information the database must hold
  - Determine the relationships among the components of that information

  ➔ *Conceptual database design* using some data model
- A **data model** is a collection of concepts for describing:
  1. data and relationships among data
  2. data semantics and constraints

Design Approaches & Notations

- Entity-Relationship Model (E/R)
- Object-Oriented (e.g., Object Definition Language -- ODL)
- Semi-structured data (e.g., XML)

A Design Process

Entity-Relationship (E/R) Model

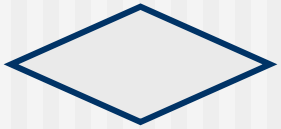
- E/R model (Chen 1976) is a *graphical* language/notation to present a database model
- It grew out of modeling applications
- Widely used in conceptual database design
- No single standard for the E/R language/notation

Entity-Relationship (E/R) Model

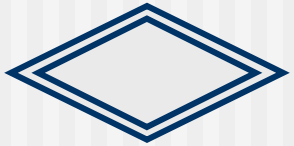
- Collection of abstraction / modeling primitives

Entity set

Weak entity set



Relationship set



Weak relationship set

Attribute

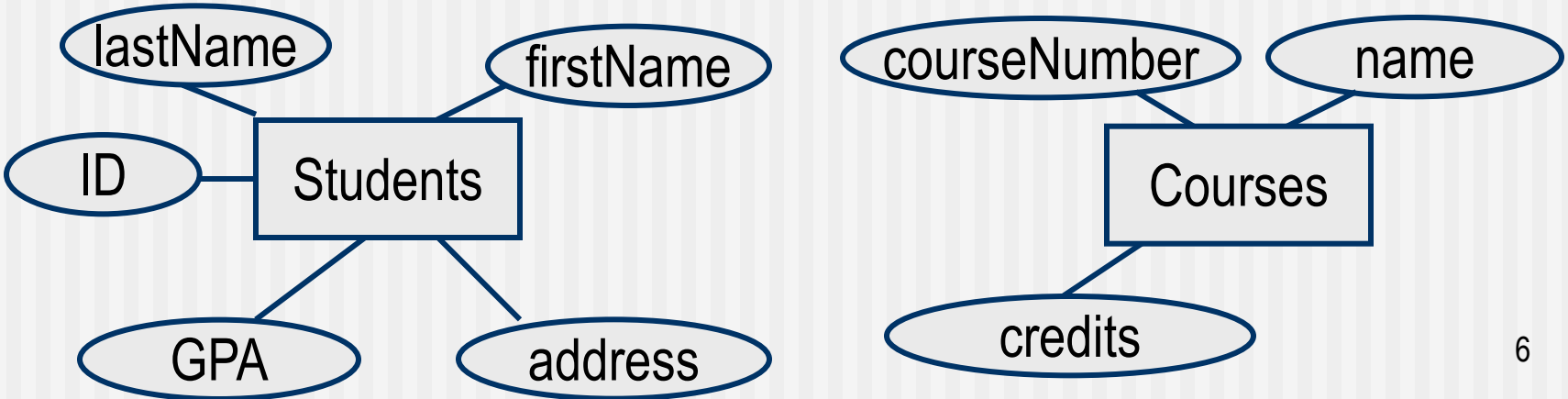
Inheritance

Multiplicity of relationships

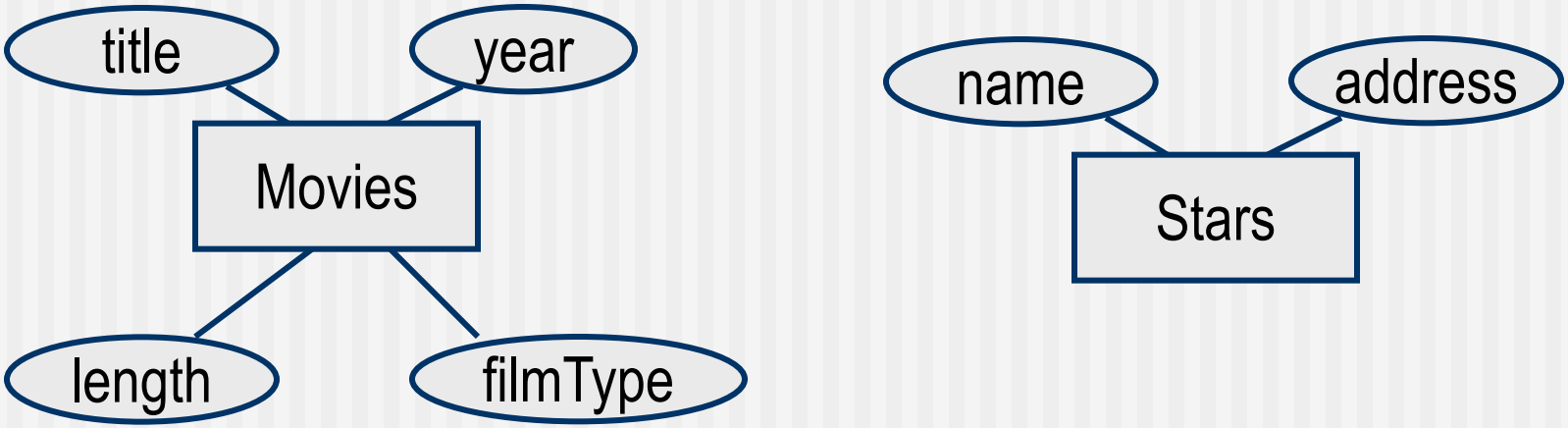
Referential integrity

Entities and entity sets

- **Entity** – A real world object that is "distinguishable" from other objects
- **Entity Set** -- A collection of similar entities
  - All entities in an entity set have the same set of **attributes**
- In ODL:
  - **Object** corresponds to entity
  - **Class** corresponds to entity set

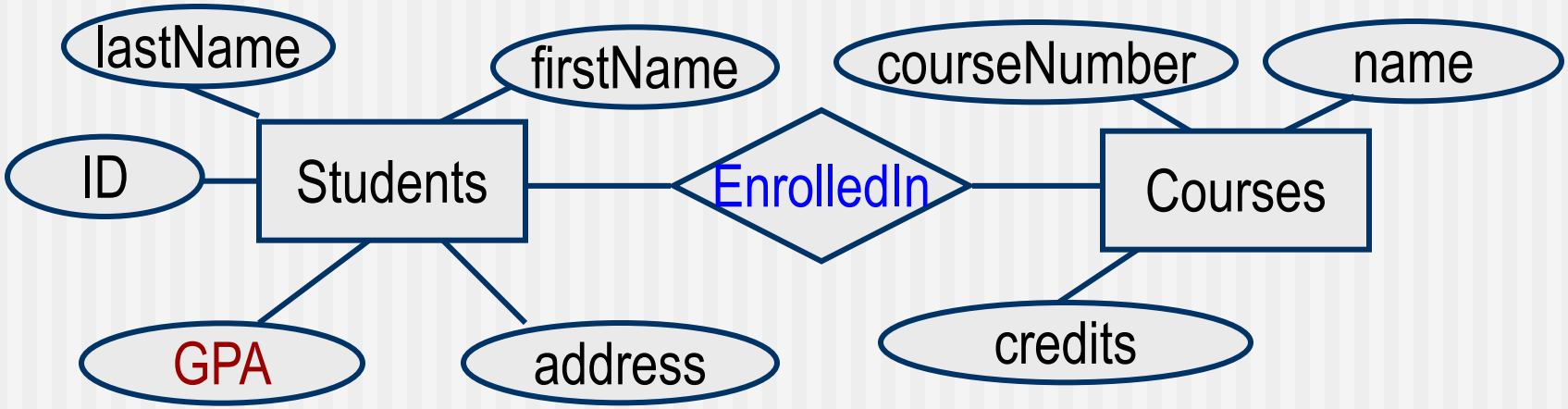


Entities and entity sets

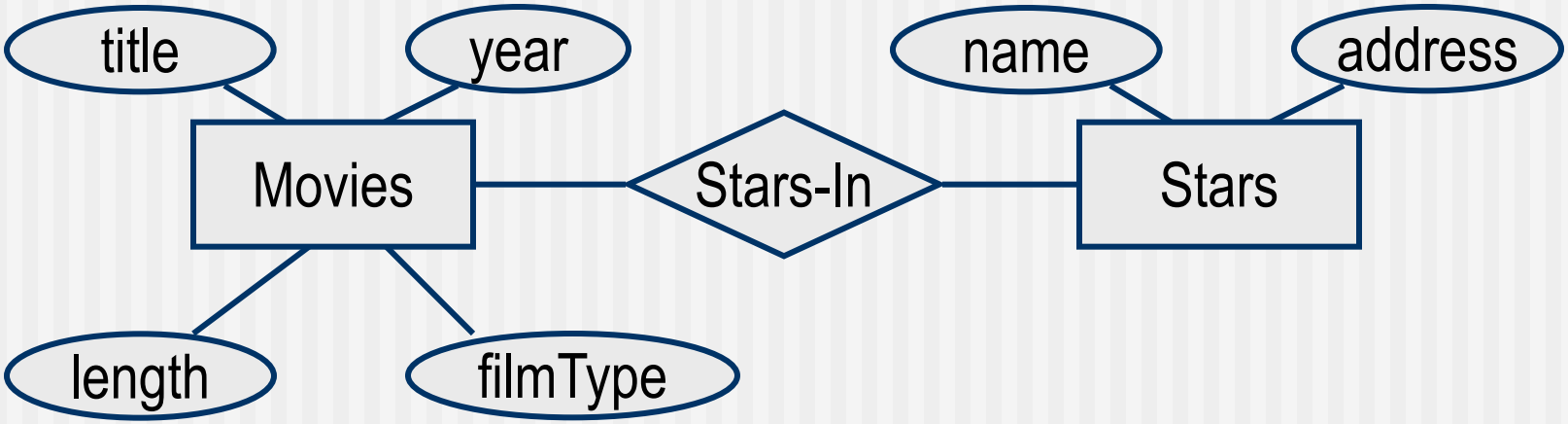


Relationships and Relationship Sets

- **Relationships** are association among entities
- **Relationship set** is a set of relationships of the same type
  - If $E_1, \ldots, E_n$ are entity sets, a relationship R on these sets is defined as: $R \subseteq E_1 \times \ldots \times E_n$
    - In general, relationships are n-ary, where $n \geq 2$
    - Many database relationships are binary

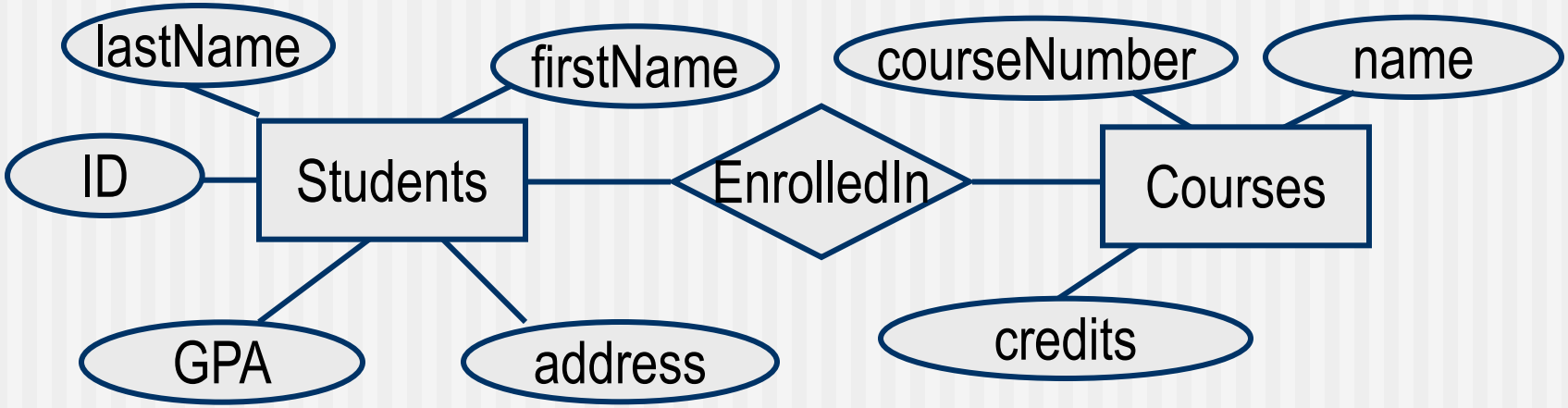


Relationships and Relationship Sets



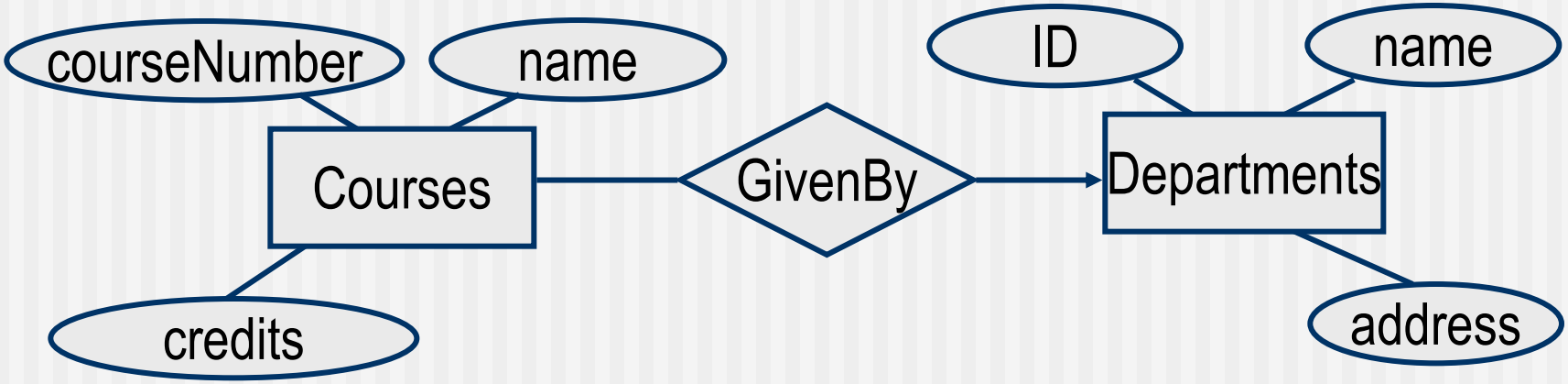
Multiplicity of Binary Relationships

- **EnrolledIn** relationship between entity sets **Student** and **Course** is **many-many**



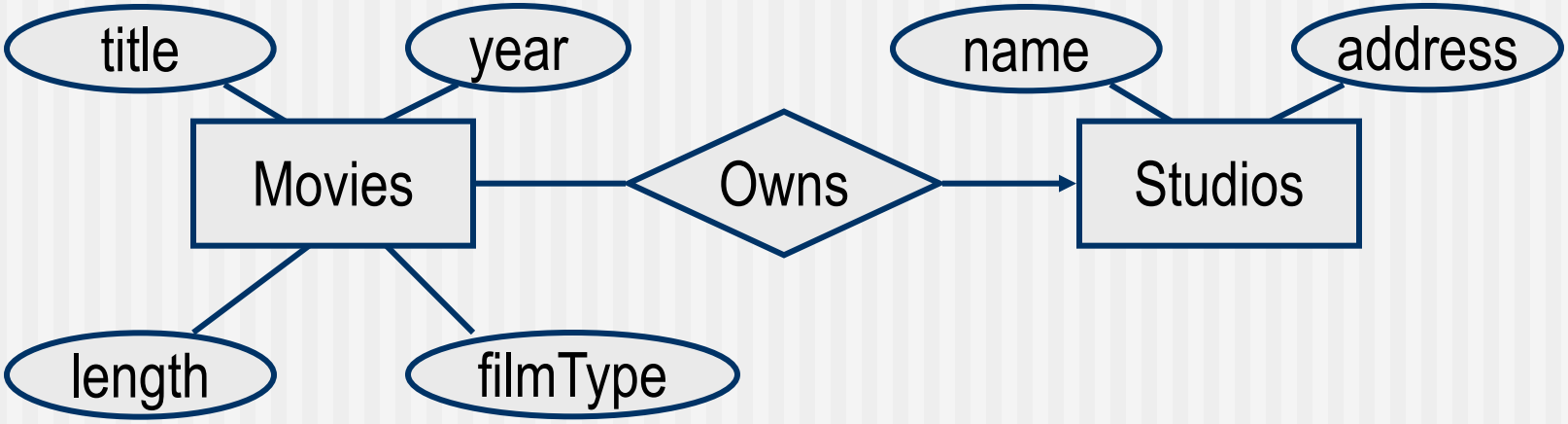
Multiplicity of Binary Relationships

- In E/R diagrams, arrows can be used to indicate the multiplicity of a relationship



- If a relationship is **many-one** from entity set **Course** to **Department,** we place an **arrow entering Department**
- The arrow indicates that each entity in the entity set **Course** is related to **at most one** entity in the entity set **Department**
- In this case, we may also say that the relationship from **Department** to **Course** is one-many (also shown as 1-M).

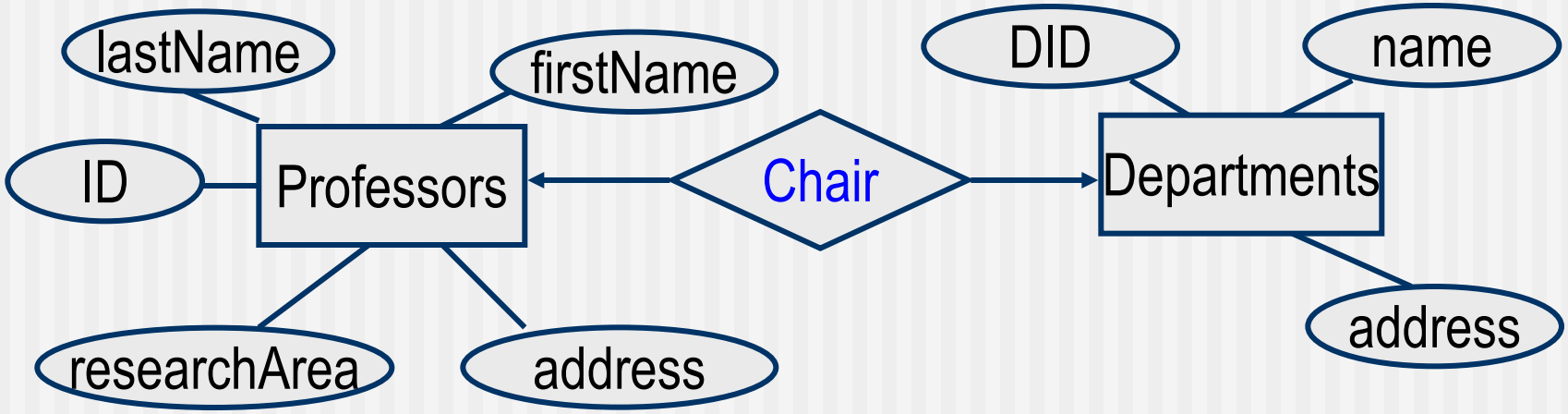
Multiplicity of Binary Relationships



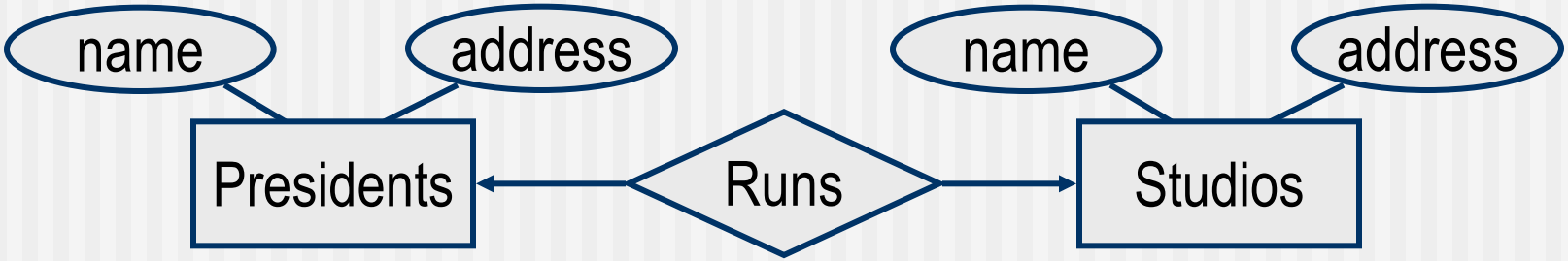
- The arrow indicates that each entity/tuple in relation **Movie** is "related to" **at most one** entity in **Studio**
- **Owns** relationship is **one-many** from **Studio** to **Movie**

Multiplicity of Binary Relationships

- The relationship Chair between the entity sets **Department** and **Professor** is **one-one,** indicating which professor is the chair of which department. Note that this represents also the situation where a department is assigned no chair.



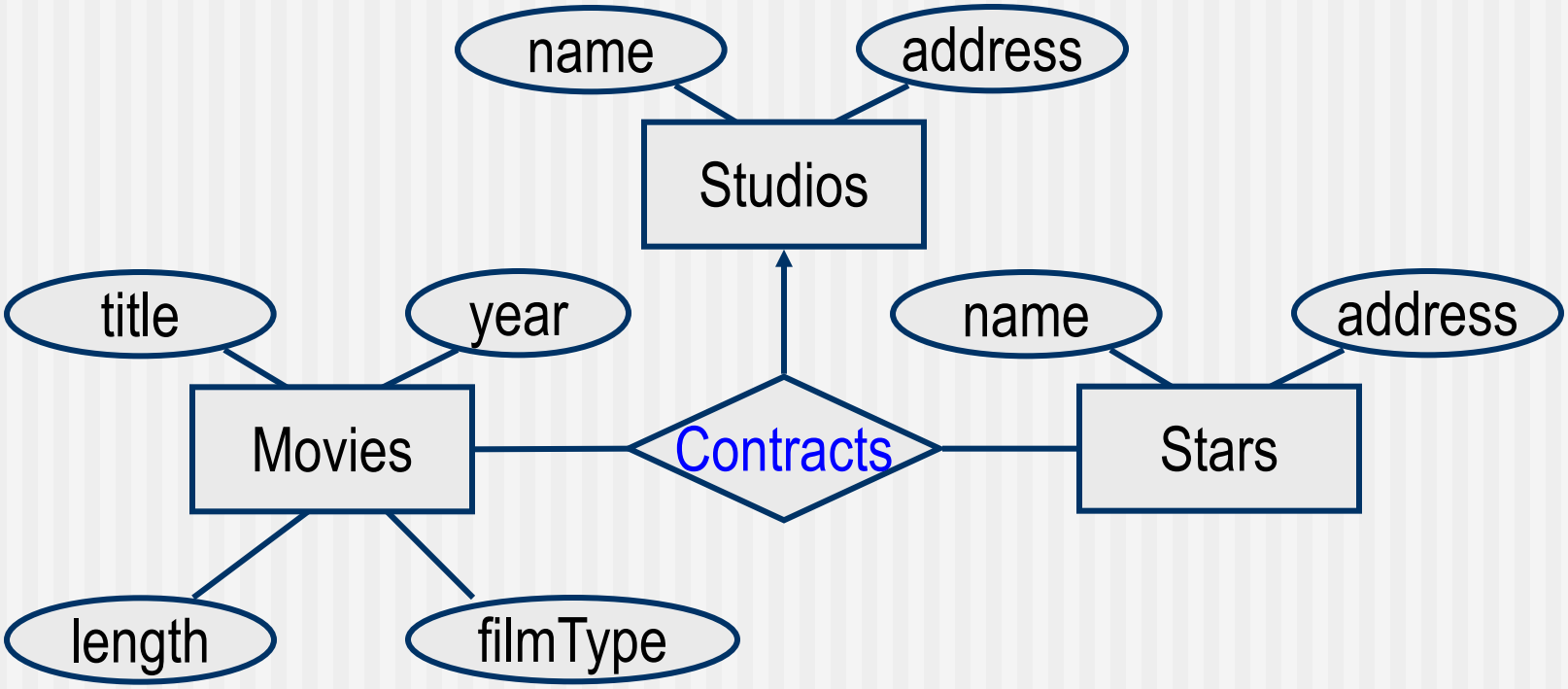
Multiplicity of Binary Relationships



Multiway Relationships

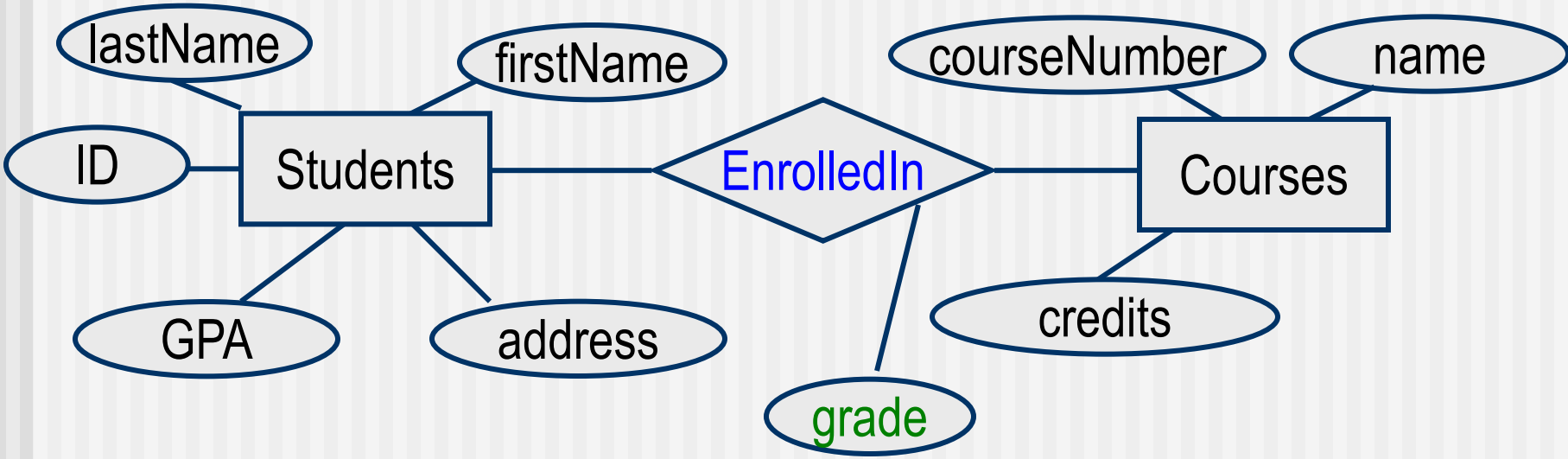
- **ODL** (an OO notation which we will introduce later) allows defining only **binary** relationships, i.e., relationships involving two classes.
- In general, we need to represent **n-ary** (multi-way**)** relationships, i.e., relationships involving more than two entity sets
- **E/R** model allows defining n-ary relationships *conveniently*.
- An n-ary relationship in **E/R** is represented by a diamond connecting all entity sets involved.

Multiway Relationships



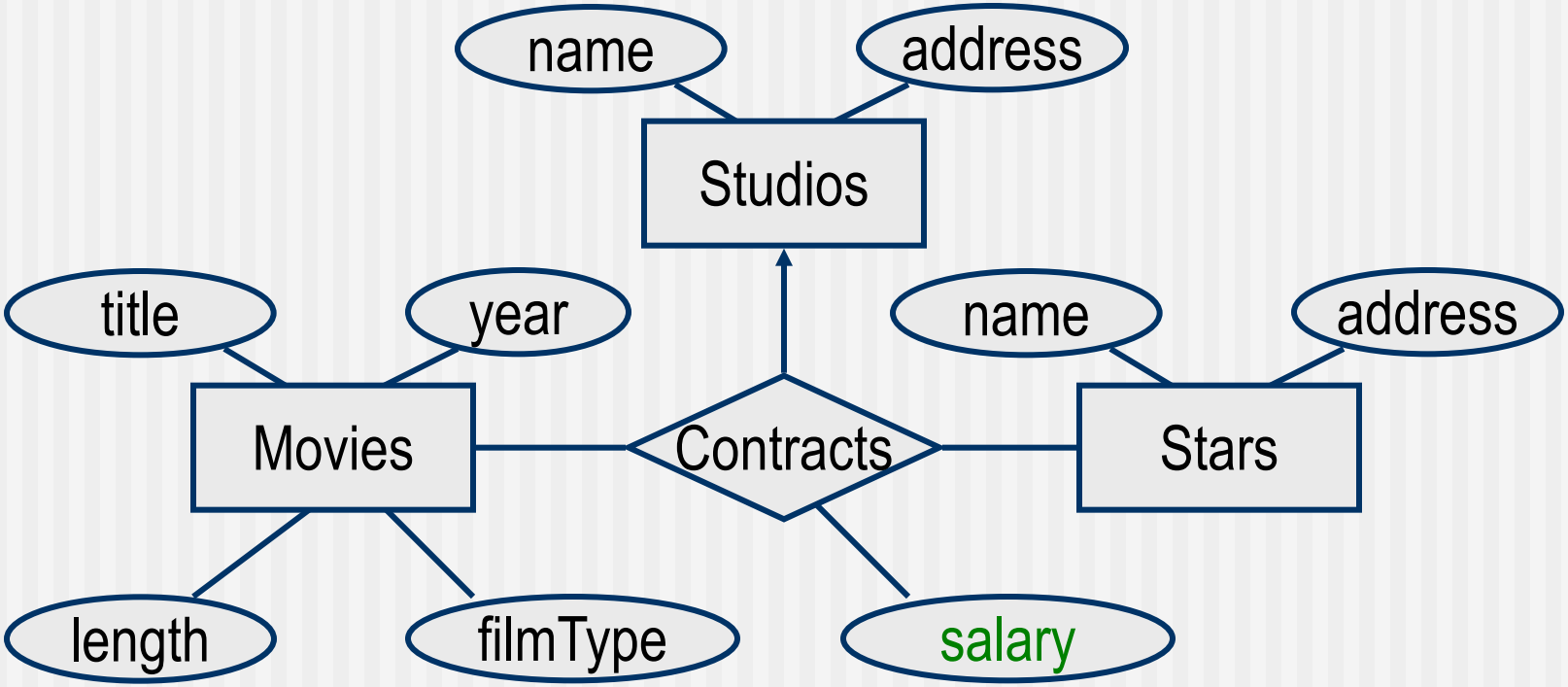
Attributes on Relationships

- Consider **EnrolledIn** relationship between **Student** and **Course**



- We may wish to record student's **grade** for this course
- Where should it go?
- Sometimes, it is more appropriate to associate **attributes** with a **relationship** rather than with the entity sets involved

Attributes on Relationships



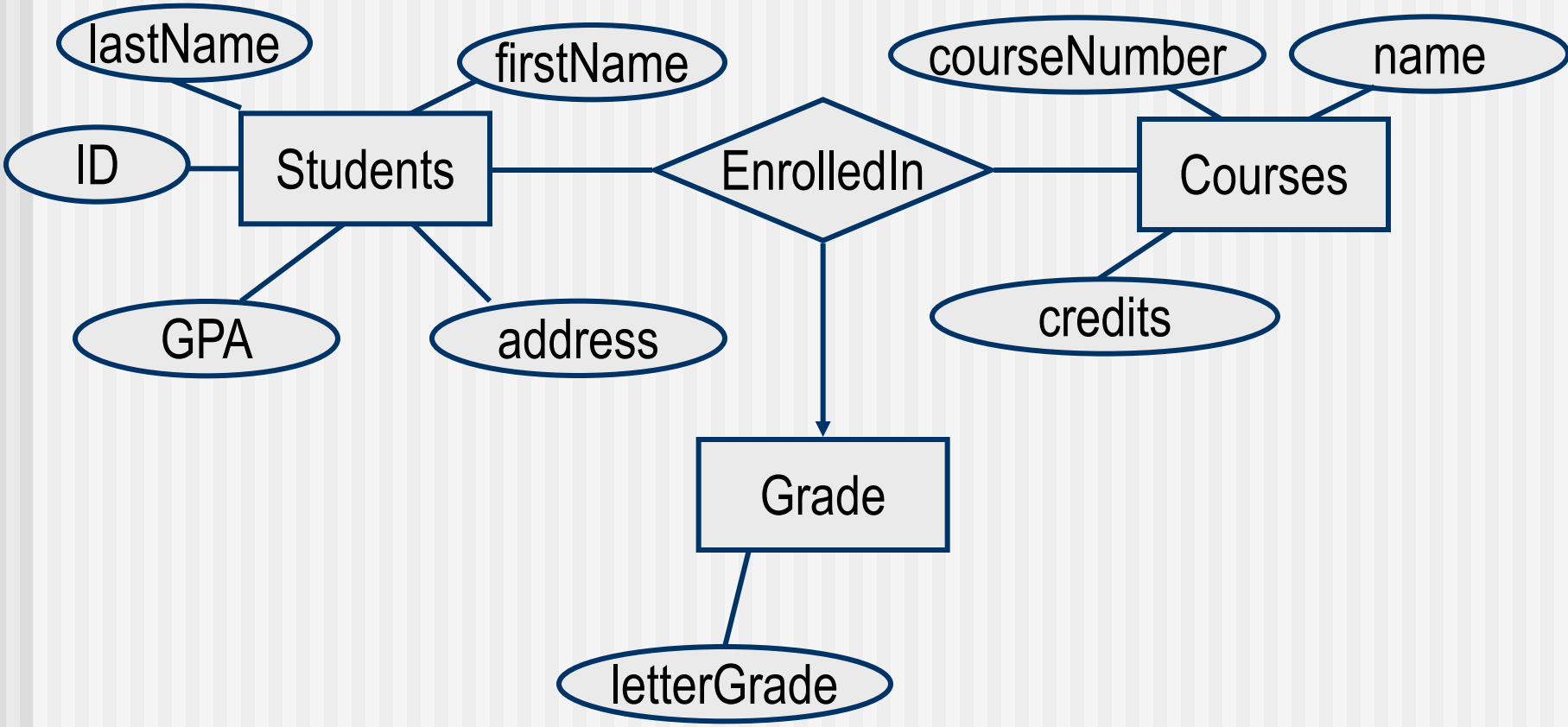
- Suppose we want to record the star's **salary** associated with this contract
- Where should it go?

Moving an attribute to an entity set

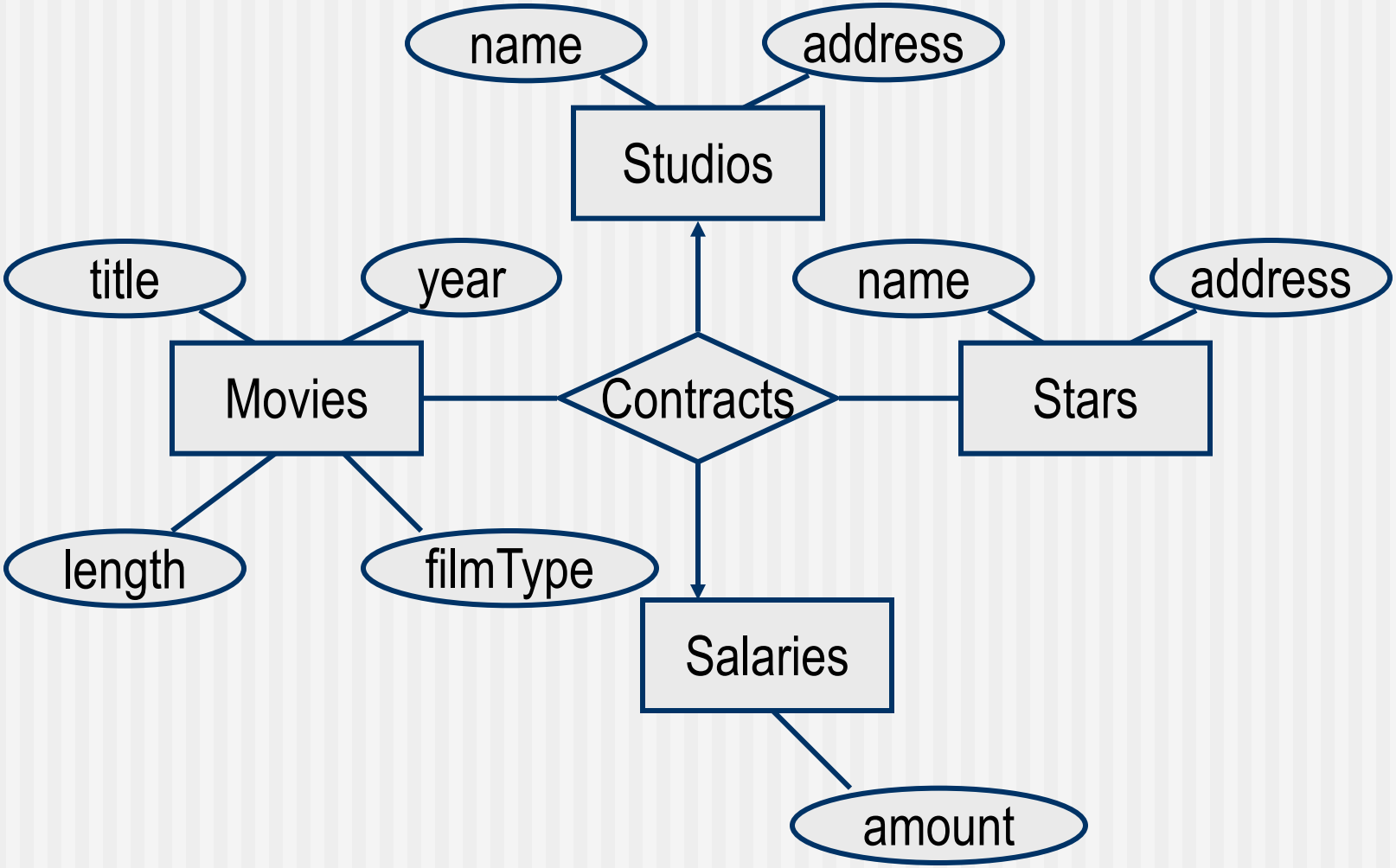
- We could add an attribute(s) to a **relationship**
- Alternatively, we could do the following:
  - invent a **new** entity set, whose entities have the attributes ascribed to the relationship
  - "connect/include" this entity set in the relationship
  - omit the attribute from the relationship itself

  - Consider again the "**salary**" for the "**Contract**" example:

Moving an attribute to an entity set



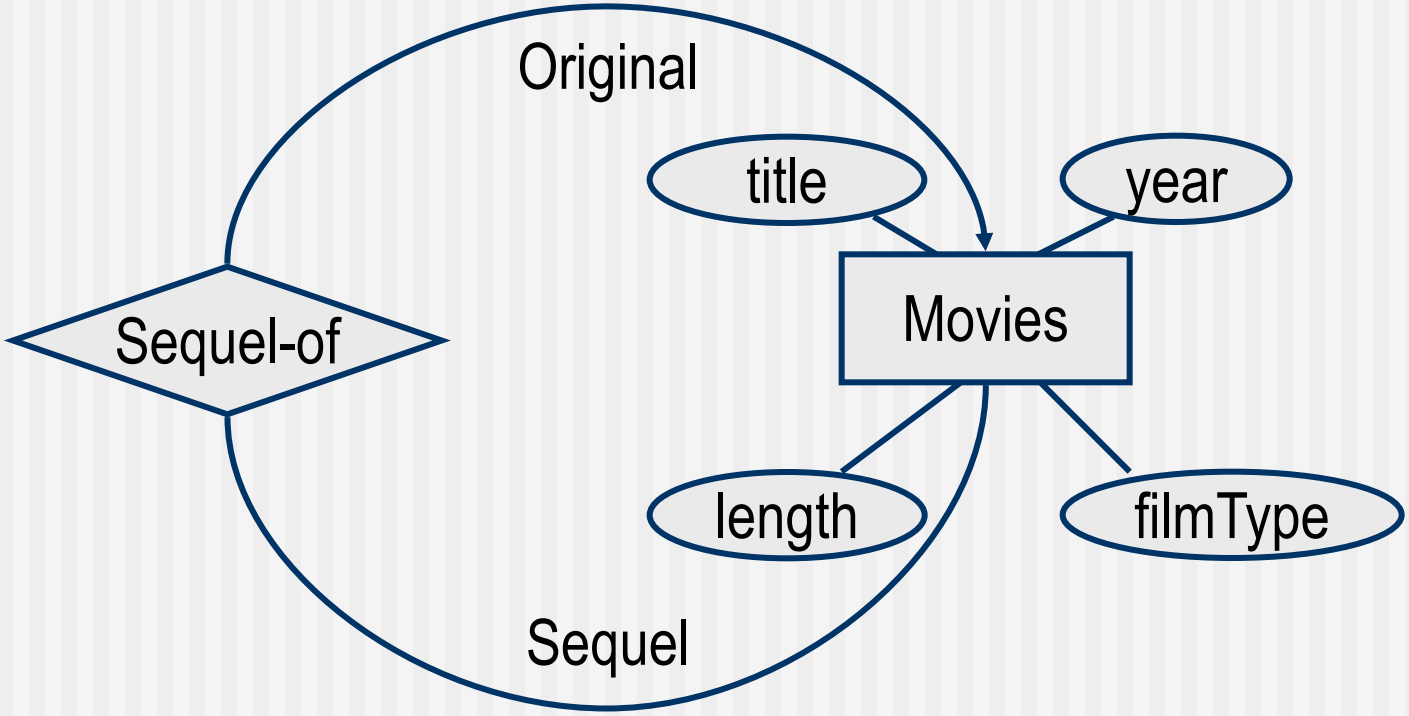
Moving an attribute to an entity set



Roles in Relationships

- It is possible that one entity set appears **two** or **more** times in a relationship
- Suppose, we want to capture relationship between two **movies**, one of which is the **sequel** of the other

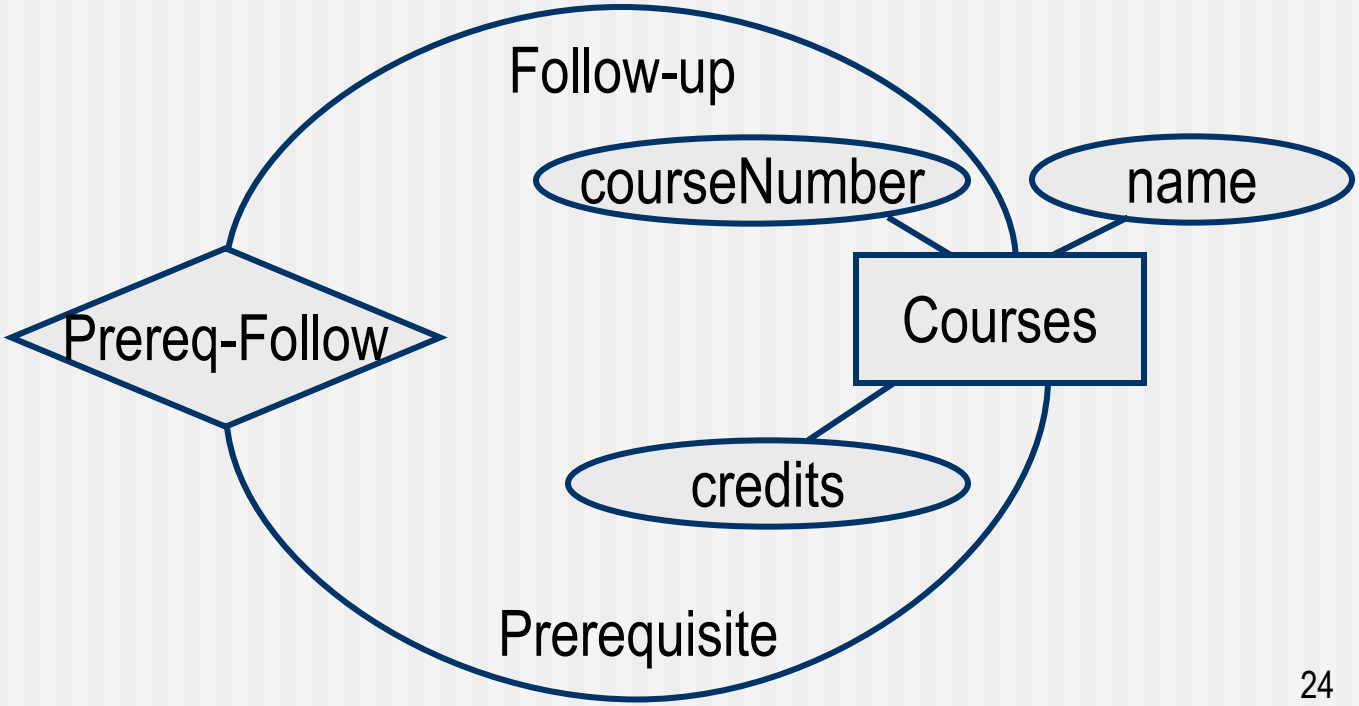
Roles in Relationships



Each line to the entity set represents a different **role** that the entity set plays in the relationship

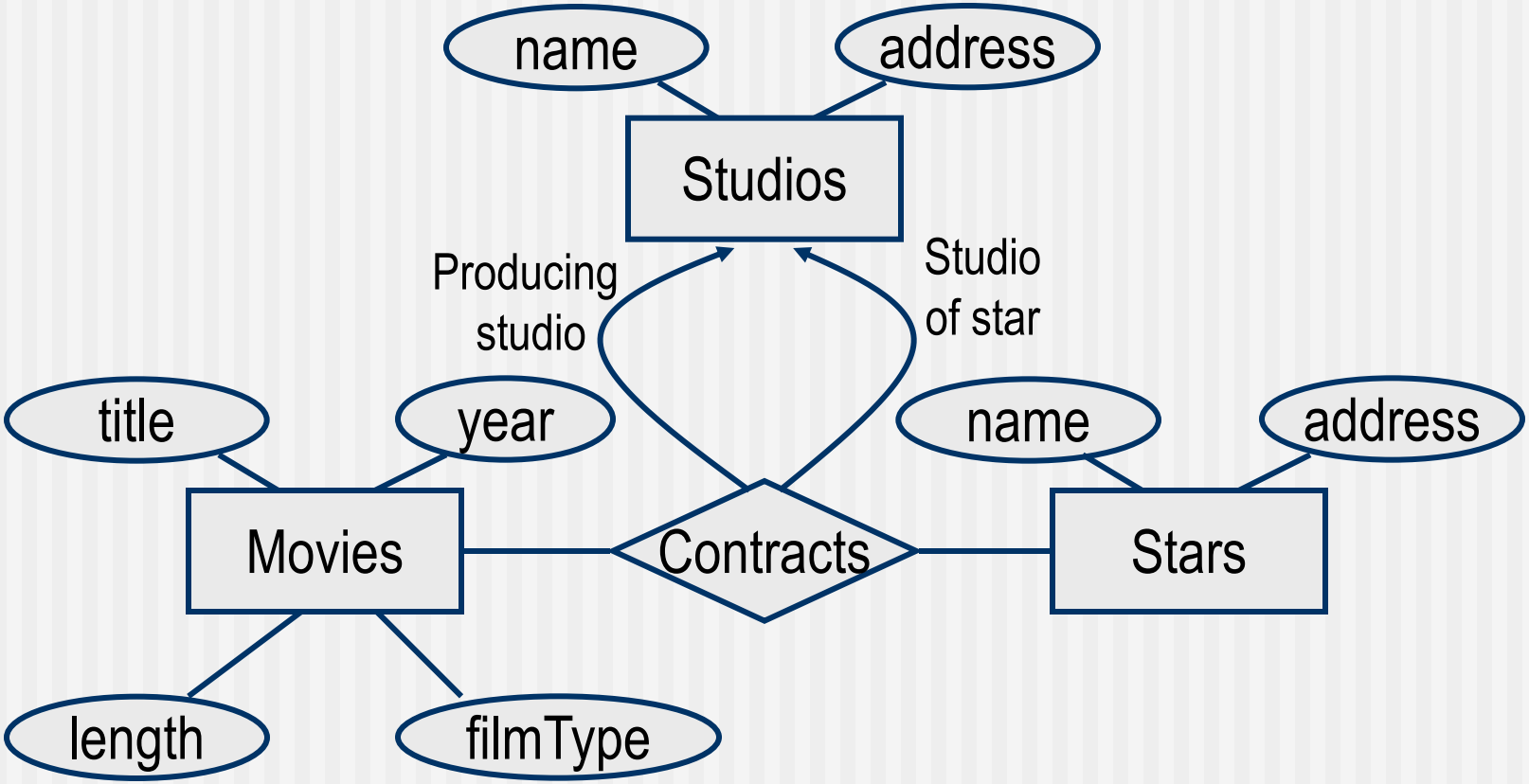
Roles in Relationships

Each line to the entity set represents a different **role** that the entity set plays in the relationship



A more complex example

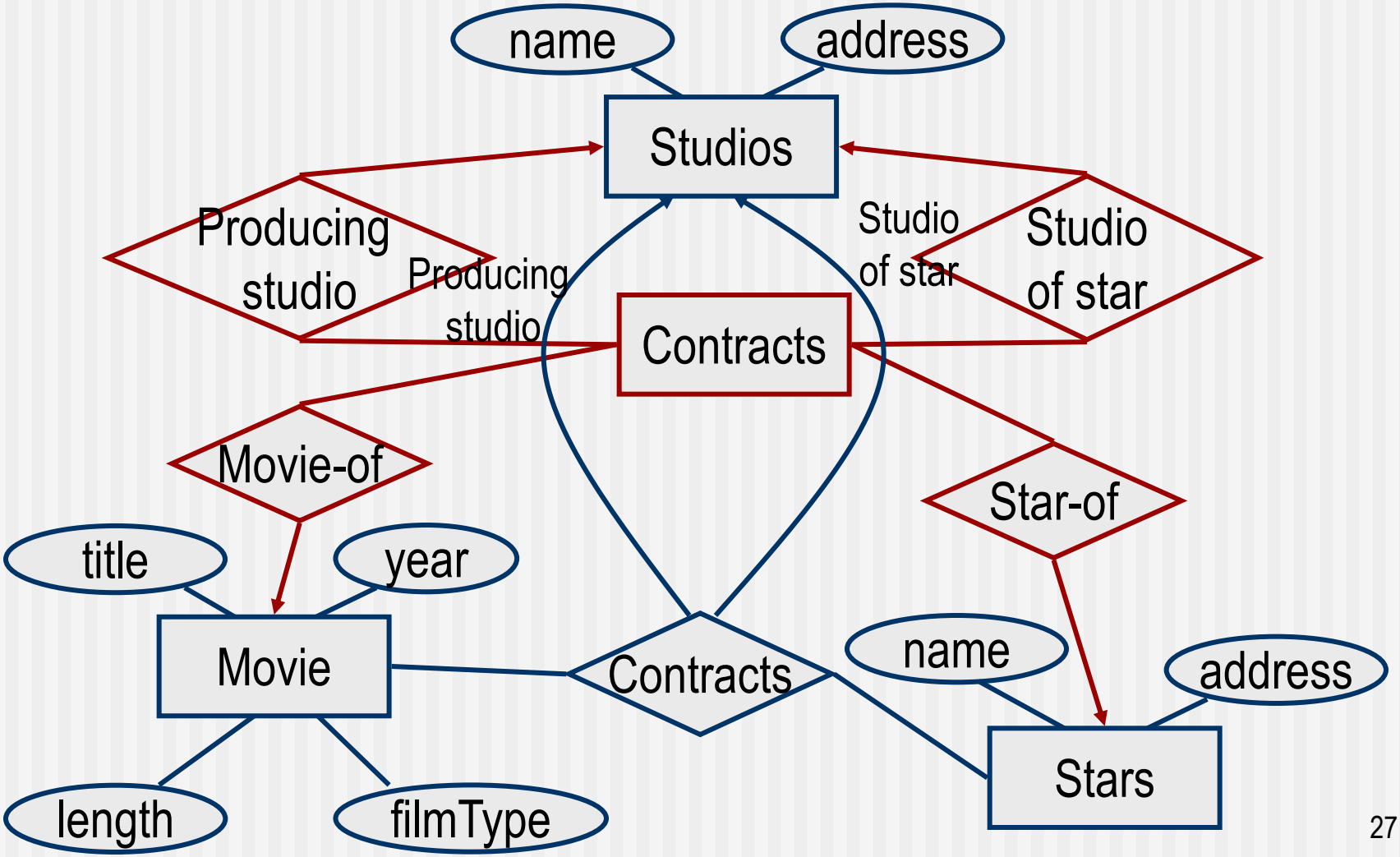
- Suppose, each star is associated with exactly one studio
- Supp. studio s1 of star a1 may further contract with another studio s2 to allow a1 to play in movie m2 made by studio s2



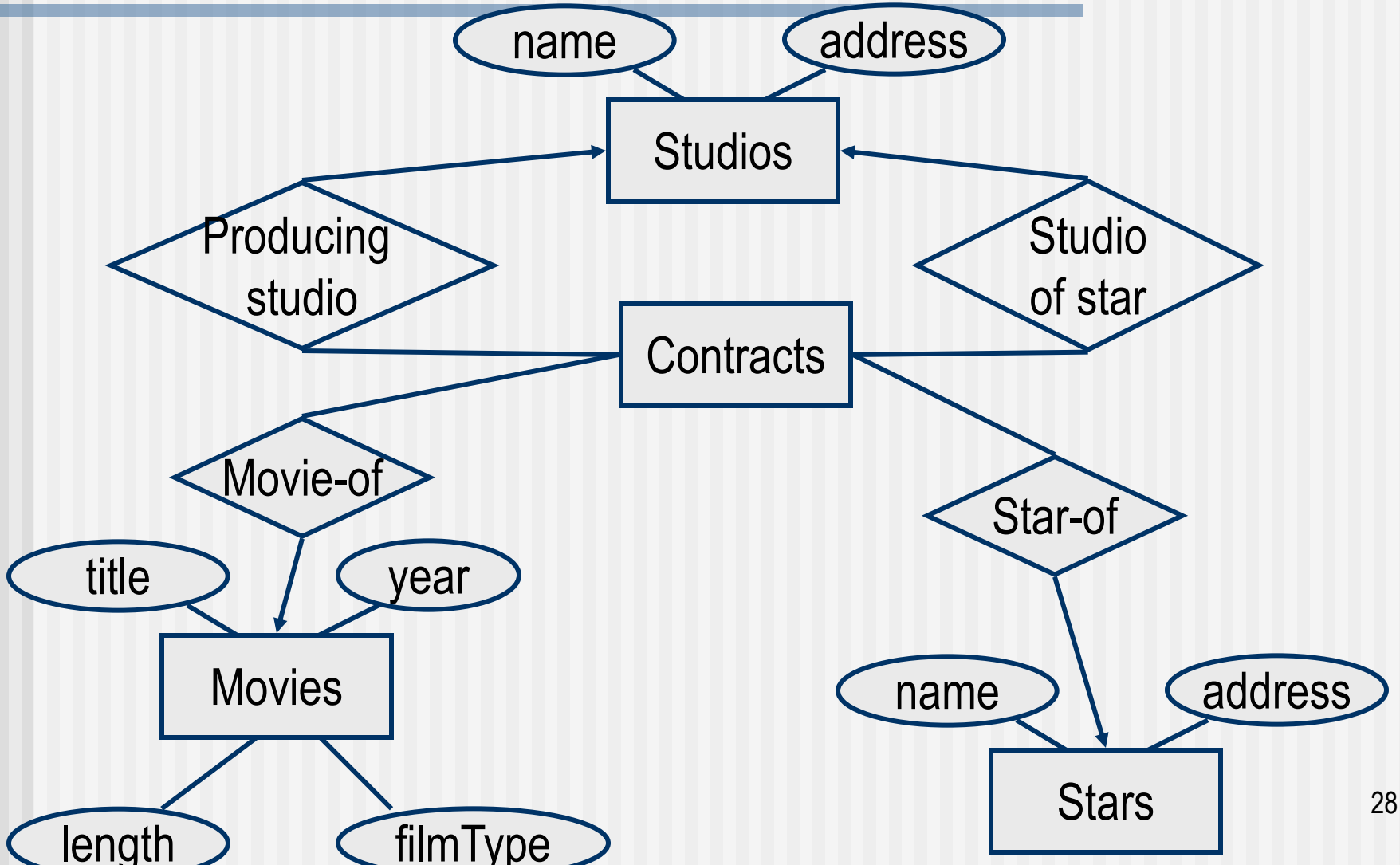
Converting n-ary relationship to binary

- Any n-ary relationship *R* can be converted into a **collection** of M-1 **binary** relationships **without loosing** the information represented by *R*
- To do this:
  - Introduce a **new** entity set E, called **connecting** entity set, whose entities might be thought of as tuples in *R* (the original n-ary relationship)
  - Introduce M-1 relationships from E to each one of the entity sets involved in *R*
  - If an entity set E plays **more than one** role, then E is the target of one relationship for **each role**

Converting n-ary relationship to binary

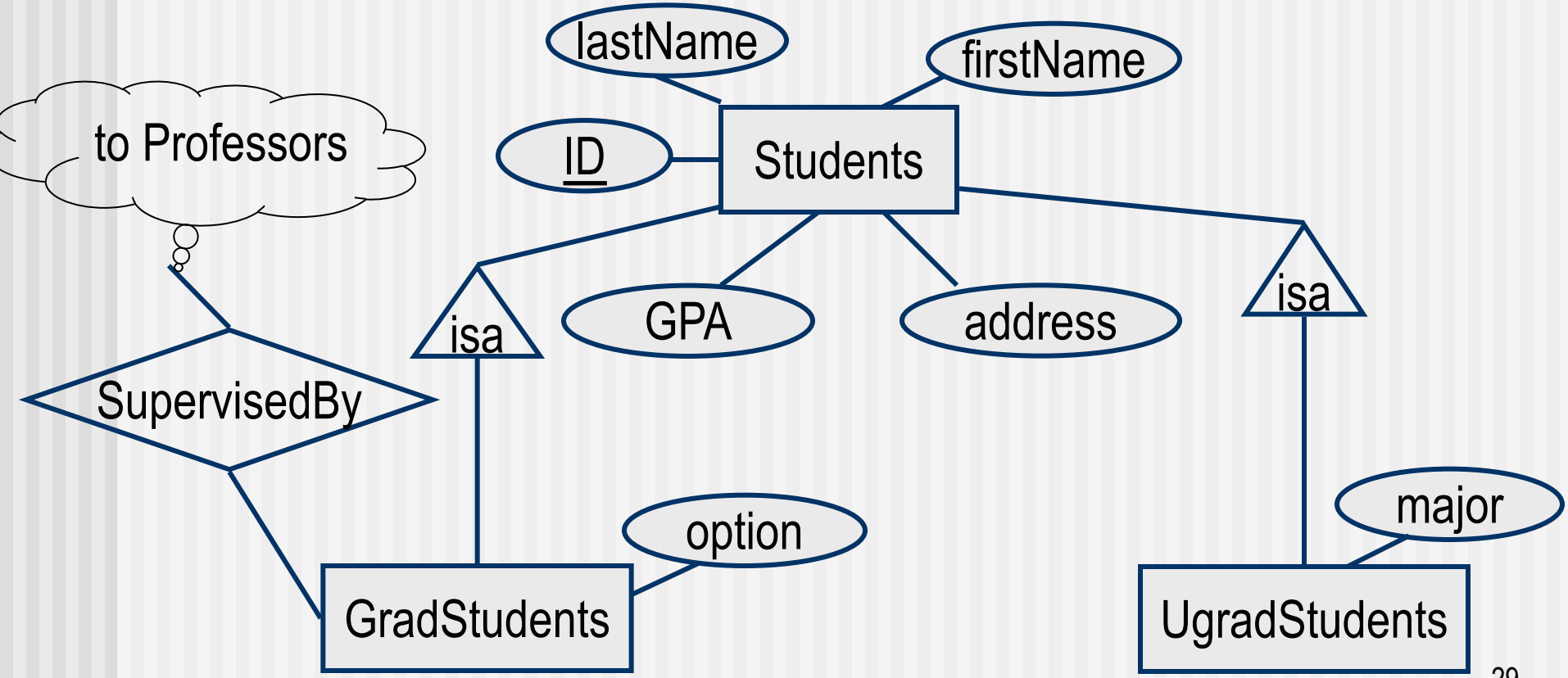


Converting n-ary relationship to binary

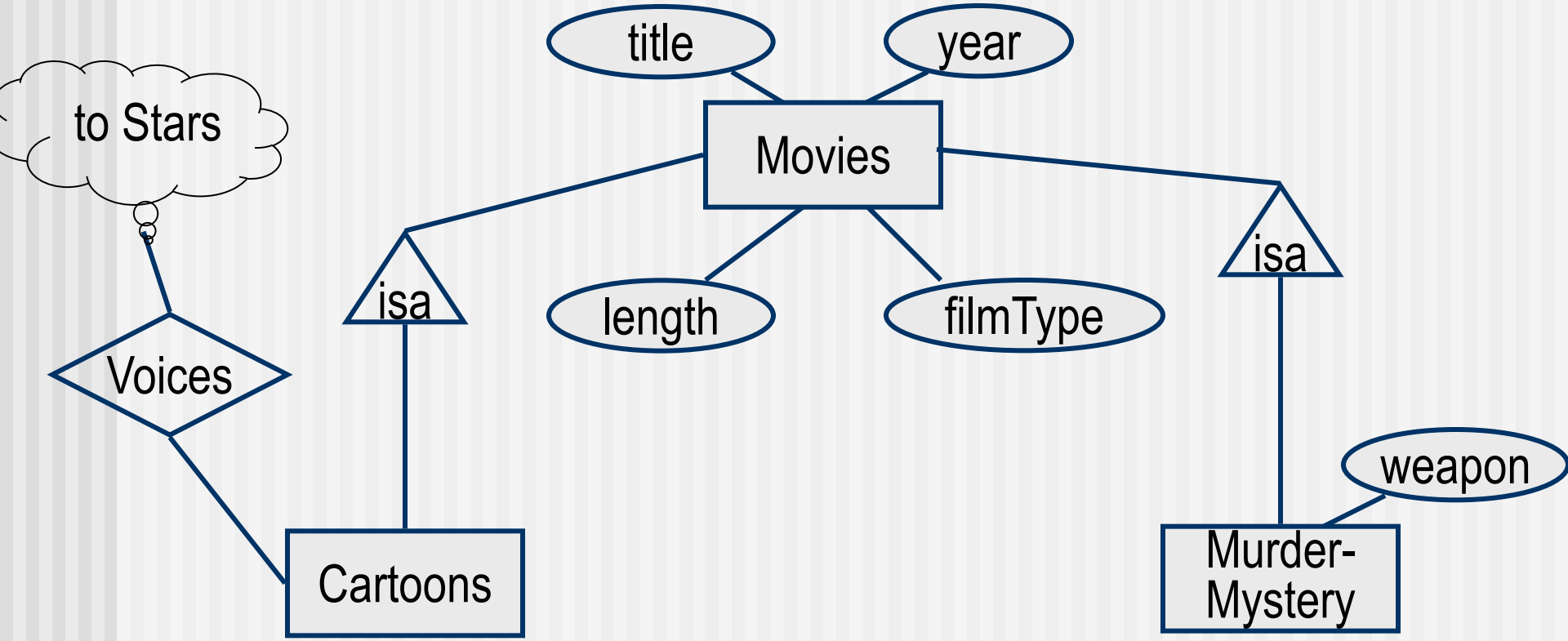


Inheritance in E/R

- Inheritance in E/R is expressed by ***isa*** relationship



Inheritance in E/R



Inheritance in E/R

- There is a subtle difference between the concept of inheritance in an OO approach and in E/R
- In OO, an object must be a member of exactly one class
- In E/R
  - We consider an entity as having "components" belonging to several entity sets that are "part of" a single **isa**-hierarchy
  - The "components" are connected into a single entity by the **isa** relationships
  - The entity has whatever attributes any of its components has, and participates in whatever relationships its components participate in

    ➔ **In an E/R diagram, we represent an entity set (e.g., CartoonMurderMystery) only if it has attributes and/or relationships of its own**

Constraints

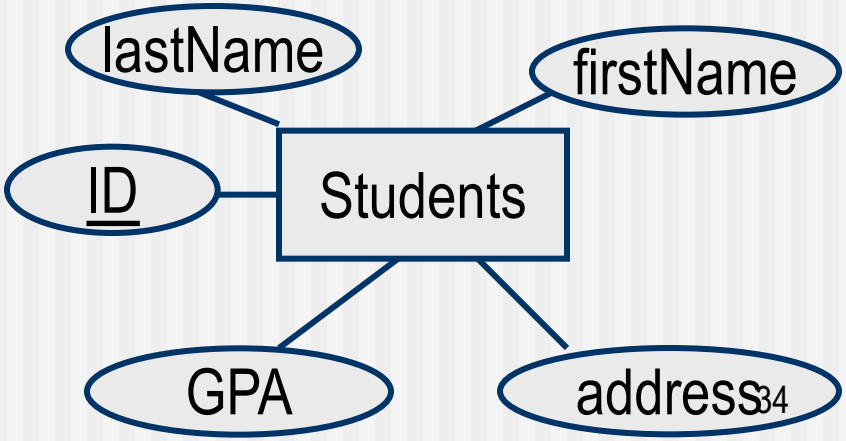
- There are some important **aspects** of the **real world** that **cannot** be represented using the **ODL** or **E/R** model introduced so far
- The additional information about these aspects often takes the form of **constraints** on the data
- Sometimes modeling this additional information goes beyond the **structural** and **type** constraints imposed by **classes**, **entity sets**, **attributes,** and **relationships**

A Classification of Constraints

- A **Key K** is a set of attribute(s) that **uniquely** identifies an object within its class or an entity within its entity set **R**; Defn: **K** $\subseteq$ **R**.
  - That is, no two entities may agree in all their **key values**
- **Single-value constraints** are requirements that the value in a certain context/role be unique. In addition to *key constraints*, there are other sources of single-value constraints (e.g., M:1 or 1:1 relationships, like a department's chair)
- **Referential integrity constraints** are requirements that a value referred to by some object/entity must actually exist in the database; This means, no dangling pointers
- **Domain constraints** require that the value of an attribute must be drawn from a specific set of values (called attribute domain), or lies within a specific range
- **General constraints –** arbitrary assertions that must hold on the db

Keys

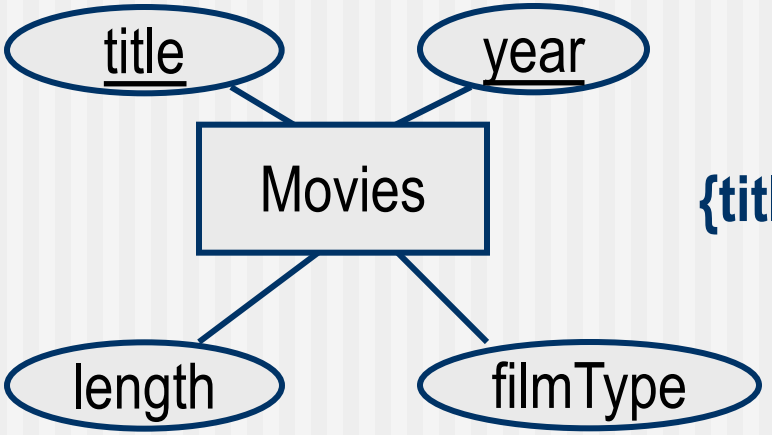
- A *superkey* is a set of attributes whose values uniquely identify an entity (object) in the entity set (class); this set may not be minimal.
- A **minimal superkey** is called a **(***candidate***) key.**
- An entity set may have more than one candidate **key**. One of them is picked (by ?) as the primary key**;** others may be called *alternates*
- In **E/R,** we underline the attributes forming a key of an entity set
- No notation in E/R for alternate keys



Here, **ID** is the key of **Student**

Example

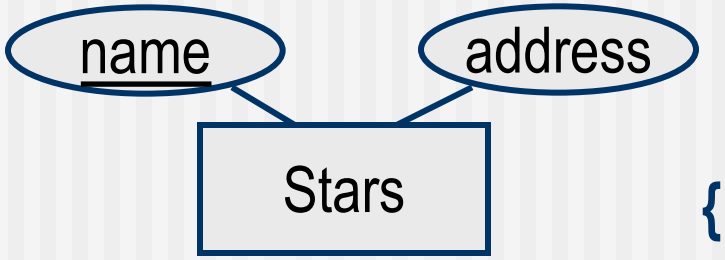
- What should we consider as a key for **Movie** ?
- **title**?
  - there could be different movies with the same name
- **{title, year}**?
  - there still could be two movies with the same title made in the same year, but that's very unlikely as we understand!



{title, year} is a key for **Movie**

Example

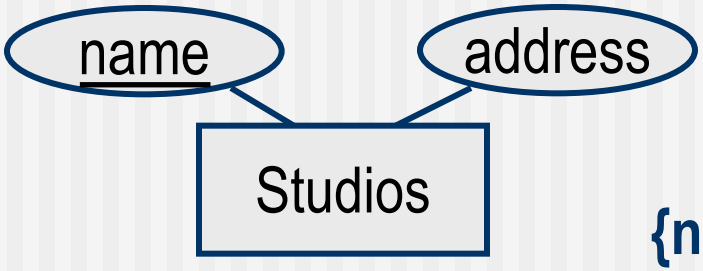
- What should be the key for **Star = {name, address}**?
- **name**?
  - We may think that the name cannot serve to distinguish two people, but for stars, the names distinguish them, since traditionally they choose “stage names” as names



{name} is the key for **Star**

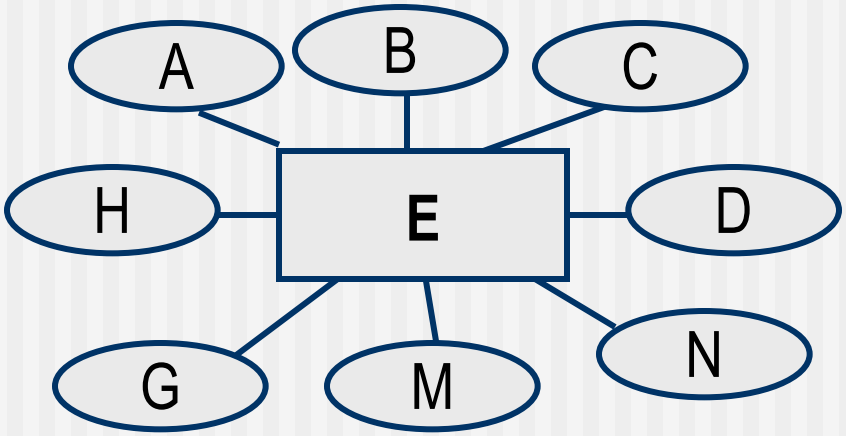
Example

- What should be considered as the key for **Studio**?
- **name**?
  - It is “reasonable” to assume that there are **no** studios with the **same names**



{name} is the key for **Studio**

Selecting Primary Key



Suppose candidate keys for **E** are :

1. **{A, B}**
2. **{D, N}**
3. **{G, M, N}**

- Which of the three should be pick as the **primary key**?

Selecting Primary Key

- Some criteria to choose a **primary key** when there are alternate keys (i.e., more than one candidate):
  - Total size of the attributes forming a key
  - Number of attributes forming a key
  - Convenience/Natural choice
  - A combination of the above

Single-Value Constraints

- In **E/R:**
  - attributes are **atomic** (first normal form, 1NF)
  - an arrow ($\rightarrow$) can be used to express the multiplicity
  - What about multi-valued or structured in E/R?

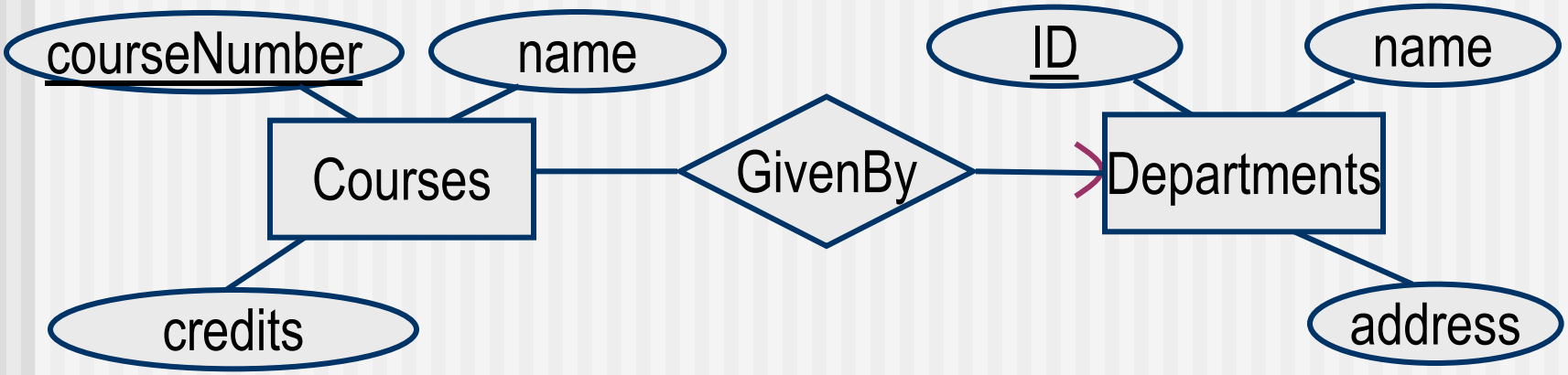
    No for attributes but Yes for relationships

Single-Value Constraints

- Suppose A is a single-valued attribute.
- Using the E/R model introduced so far, we can't:
  - Express that the value for A *must* be present (e.g., when A is the key or part of a key), or
  - Express that the value for A *may* be present (e.g., when A's value is optional, for which we use NULL)
- If the choice is not explicit, we use the following "defaults":
  - The value for A must exist if A is part of the key
  - The value for A is optional, otherwise

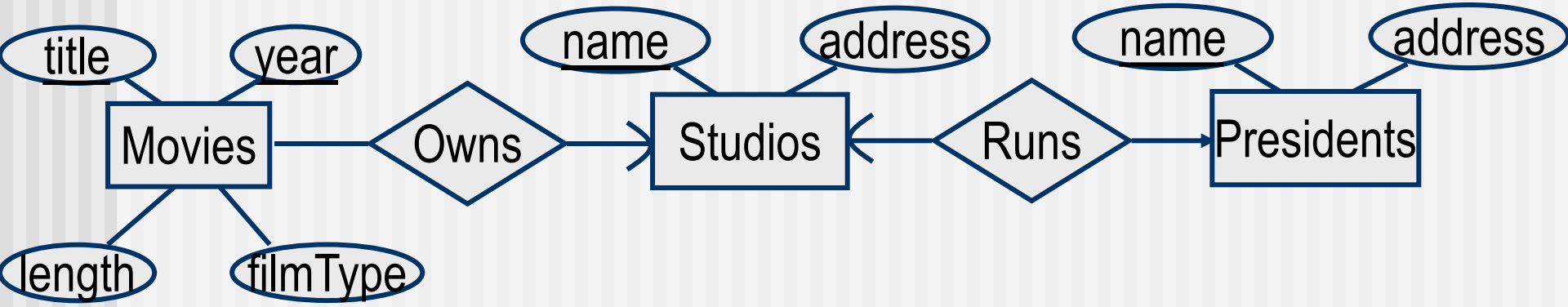
Referential Integrity Constraints

- For relationships:
  - **Single-value** + **Existence** = **Referential Integrity Constraint**
- We extend the **arrow** notation to indicate a reference is mandatory (to support referential integrity)



- This means, there is no course listed in the database unless there is a department giving the course.

Referential Integrity Constraints



- The studio for a movie must always be present in **Studio**
- If someone is a president, then the studio that he/she runs must exist in **Studio**
- However, the model allows studios without presidents (temporarily)

Domain constraints

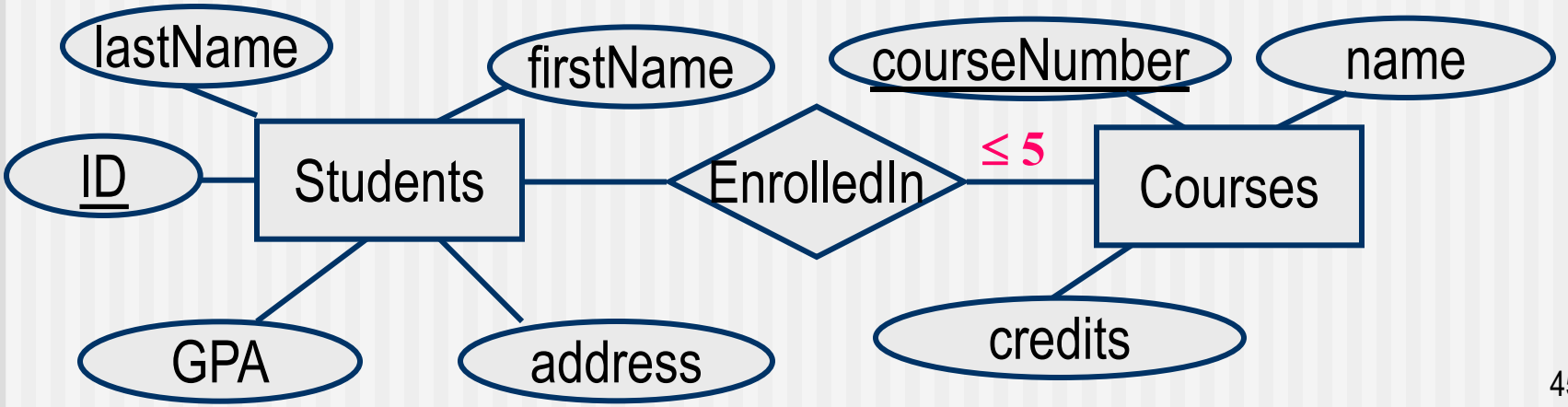
- **Domain constraints** restrict the values of an attribute to be drawn from a set
  - E/R does **not** support imposing domain constraints
  - ODL allows using types to limit/control possible values of the attributes
  - ODL does not support restricting the “range” of values allowed for an attribute

Relationship degree constraints

- **Relationship degree constraints (**multiplicity**)** restrict the number of entities in the entity sets involved in a relationship
- For example, we can impose a constraint that:

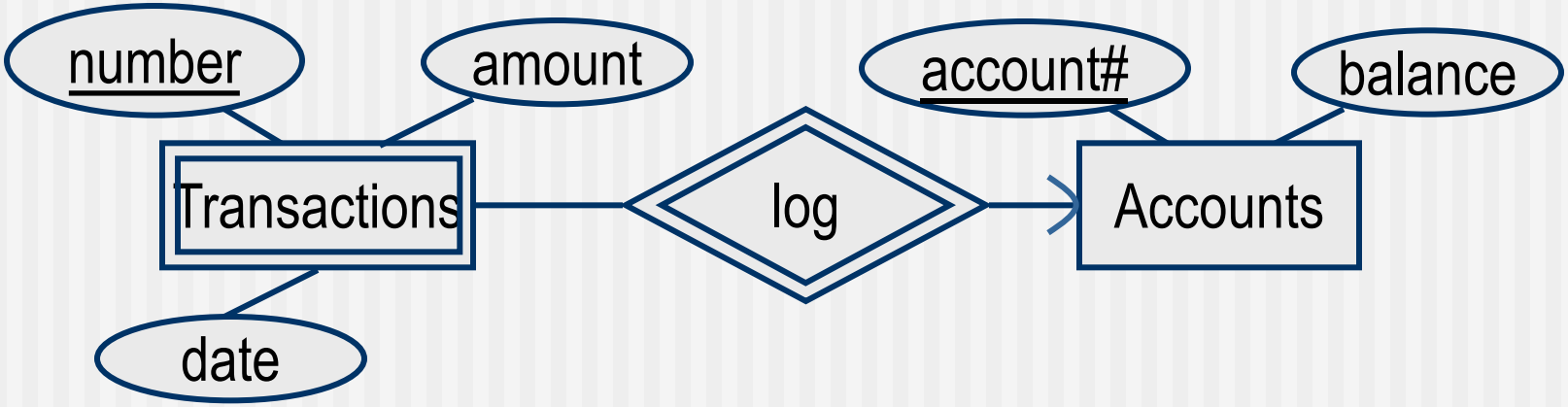
  a student cannot be enrolled in more that 5 courses

  - In E/R, we may attach a "bounding number" to the corresponding link
  - In ODL, a set of references or an **array** of size 5 (of reference type) will do



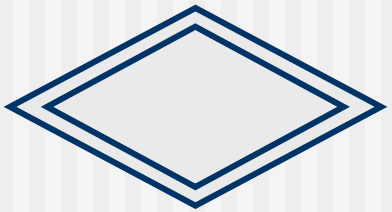
Weak Entity / Relationship Sets: Example

- Log records the transactions initiated through an ATM
- Each transaction has a number, a date, and an amount
- Different accounts might have transactions by the same number, on the same date, and for the same amount



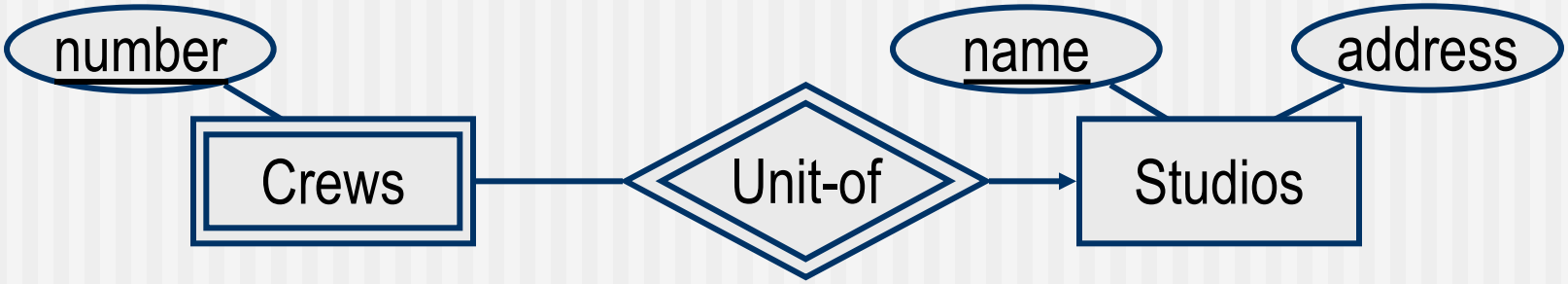
Weak Entity / Relationship Sets

- A **strong** entity set has a key
- A **weak** entity set does not have sufficient attributes of its own to form a key. It participates in a M-1 relationship (with no descriptive attributes) with a strong entity set
- **Discriminator** of a weak entity set is a set of attributes that distinguishes among the entities corresponding to a strong entity
- **key** of a weak entity set = key of the strong entity together with the discriminator of the weak entity
- In E/R, these are represented by:



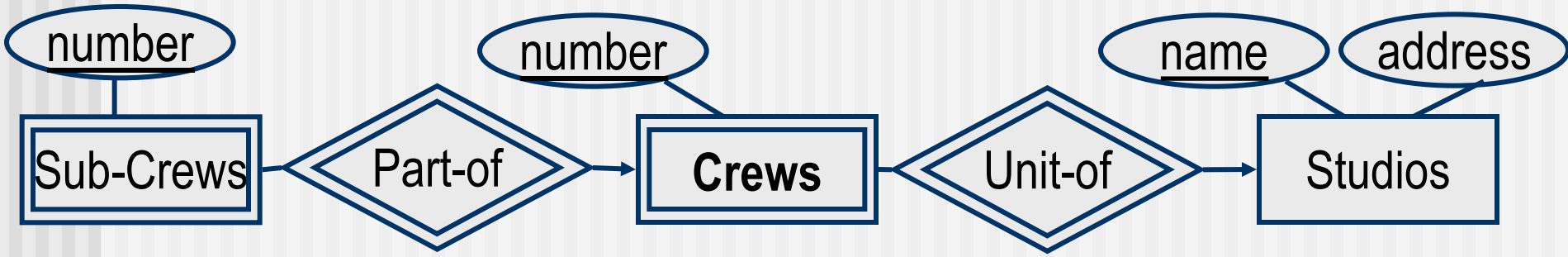
Another example

- A movie studio may have several film crews
- The crews in a studio may be designated as crew 1, crew 2, and so on
- All studios use the same designations for crews



Sources of Weak Entity Sets

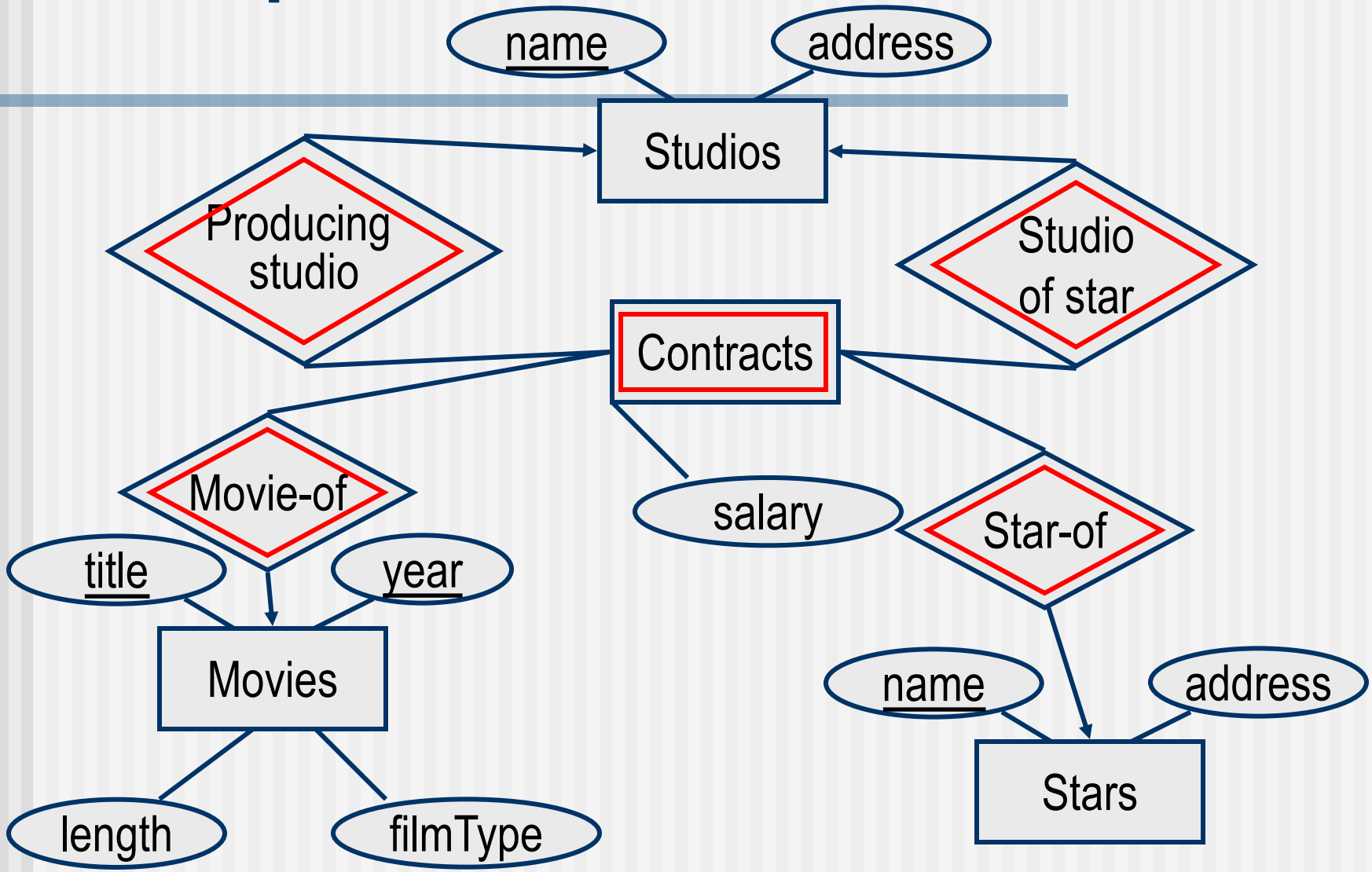
- Entity sets fall into a hierarchy
  - The entities in set **Crew** are sub-units of entities in set Studio
    - The "numbers" used for crew entities are not unique until we take into account the name of Studio with which she/he is associated (subordinate)
  - It is possible to have a chain of "weak" entity sets/relationships



Sources of Weak Entity Sets

- Connecting entity sets (CES):
  - CES's often have no attributes of their own
  - Their key is formed from the key attributes of the entity sets which they connect
- A connecting entity set is always weak

Example


name
address
Studios
Producing studio
Studio of star
Contracts
Movie-of
salary
Star-of
title
year
Movies
name
address
Stars
length
filmType

Design Principles

- Design should
  - Reflect the "reality" we are trying to model – faithful/realistic
  - Avoid redundancy -- minimal
    - Redundant information takes space
    - Could cause inconsistency
  - Be as simple as possible
- Be careful when choosing between using attributes and using classes or entity sets. Remember that:
  - An attribute is **simpler** to implement than a class/entity set or a relationship
  - If something has more information associated with it than just its name, it should probably be modeled as an entity set or a class

A quick test!

- Which of the following statements is **NOT** correct?
  - **A.** A data model is a collection of concepts for describing data and their relationships.
  - **B.** A data model is a language for describing the data semantics and the constraints.
  - **C.** A data model is a language for expressing queries and transactions over a database.
  - **D.** A data model is an abstract representation of the structure of the data.