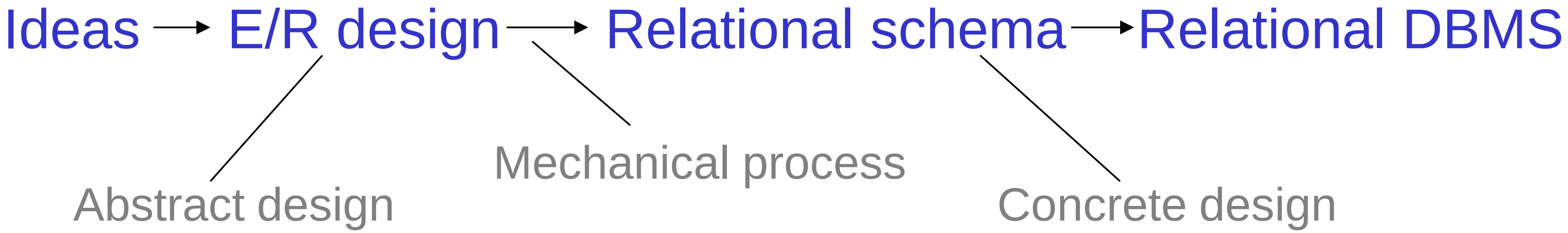
From E/R Diagrams to Relations



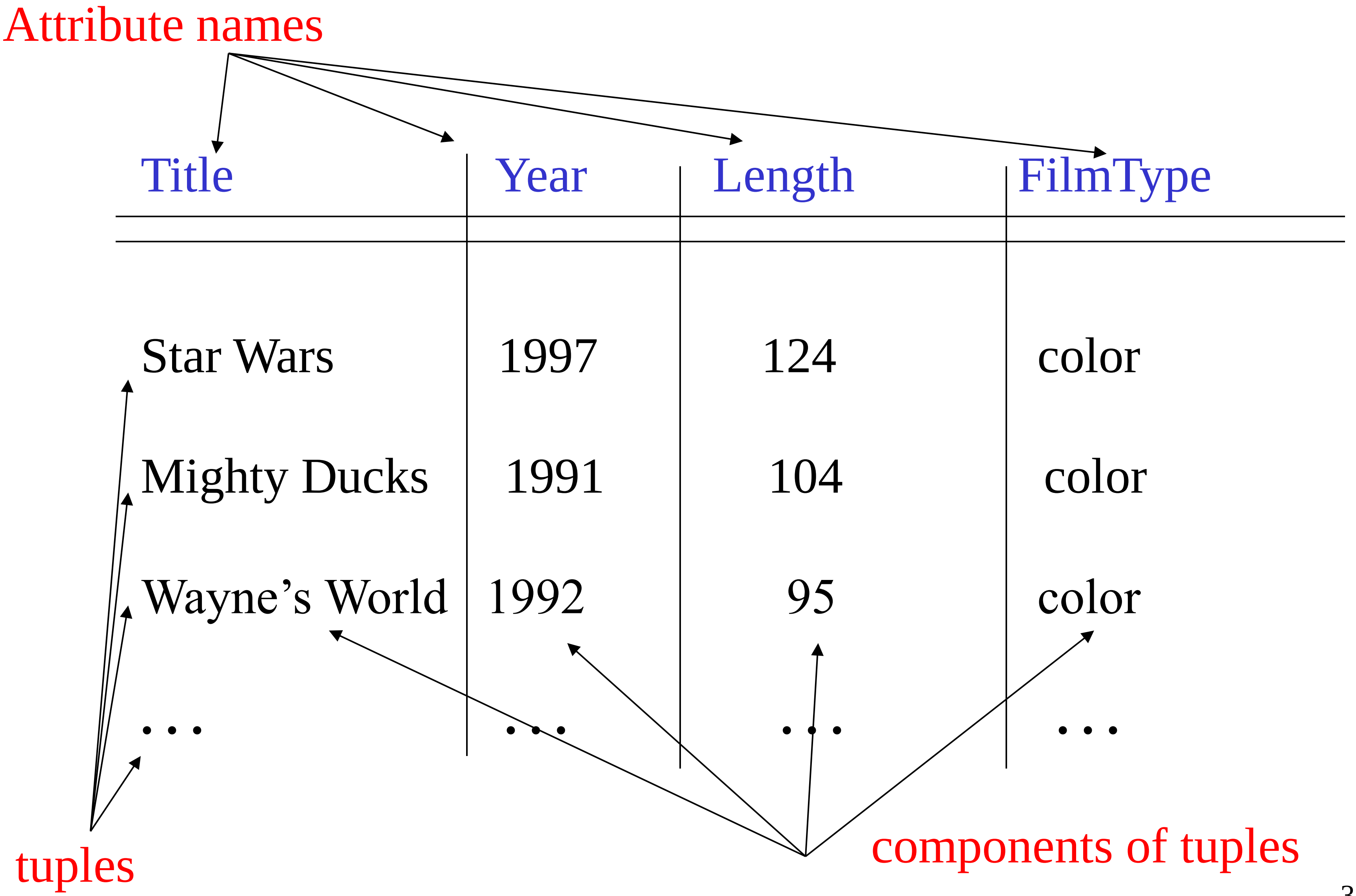
Ideas → E/R design → Relational schema → Relational DBMS

Abstract design

Mechanical process

Concrete design

Relations (or Tables) Terminology



Attribute names

Title	Year	Length	FilmType
Star Wars	1997	124	color
Mighty Ducks	1991	104	color
Wayne’s World	1992	95	color
. . .	. . .	. . .	. . .

tuples

components of tuples

More Terminology

Every attribute has an atomic type.

Relation Schema: relation name + attribute names + attribute types

Relation instance: a set of tuples. Only one copy of any tuple!

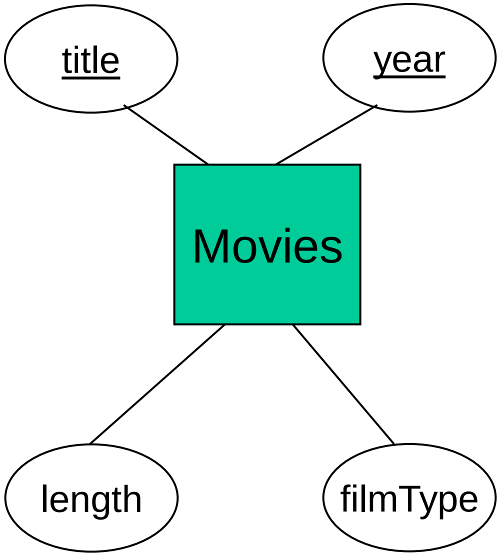
Database Schema: a set of relation schemas.

Database instance: a relation instance for every relation in the schema.

From E/R Diagrams to Relations

- **Entity sets** become relations with the same set of attributes.
- **Many-Many Relationships** become relations whose attributes are only:
  - The keys of the connected entity sets.
  - Attributes of the relationship itself.
    - Sometimes attribute renaming needed to avoid name clashes.
- **Many-One Relationships** usually don’t need separate tables.
  - The key of the “one” side is included in the relation of the “many” side
- **One-One Relationships** are similar.
- **Ternary (or higher) relationships** need separate tables with keys of the participating entity sets.
  - The key is the union of keys of the “many” sides.

Example: Entity Sets to Relations

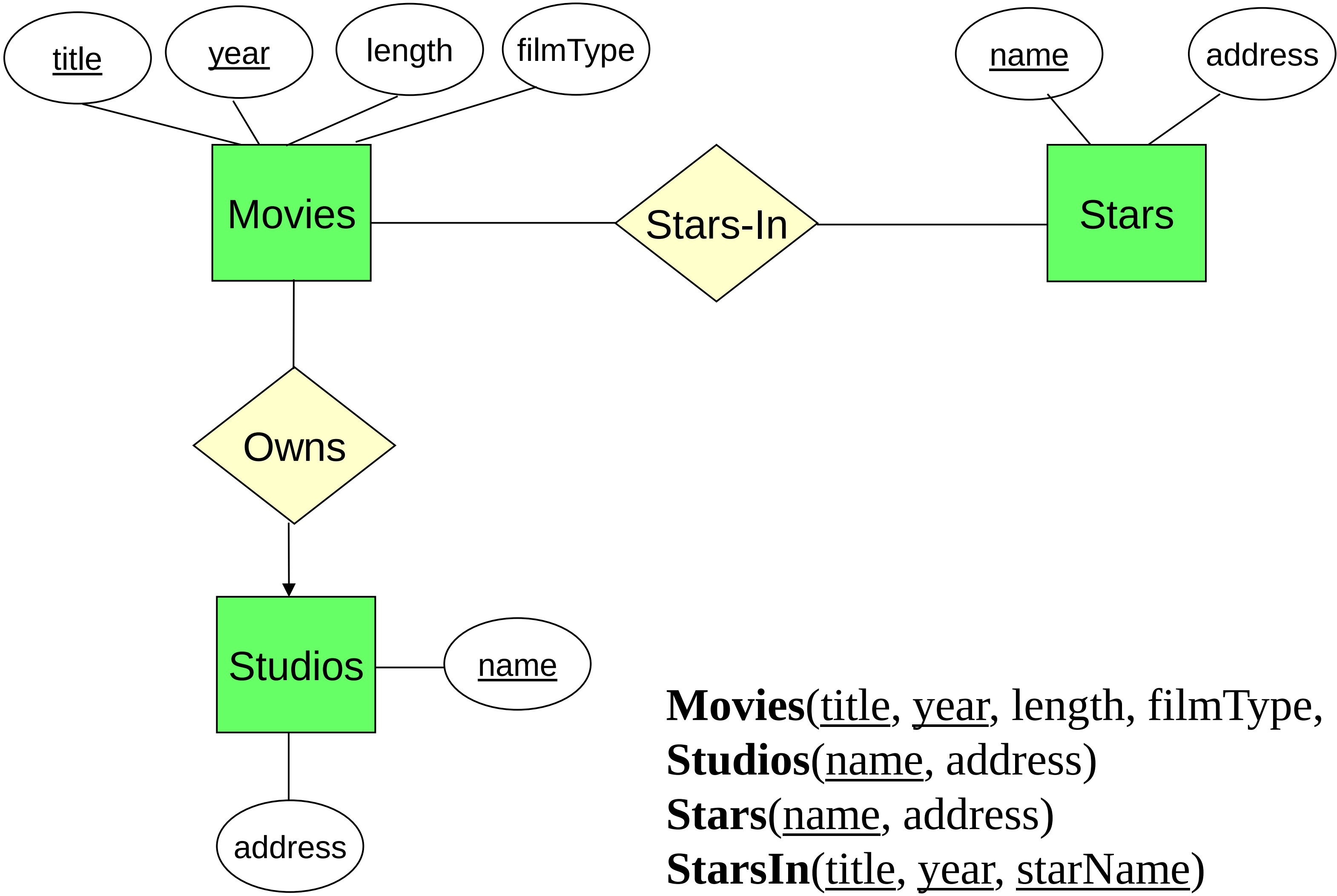


Relation schema:
Movies(title, year, length, filmtype)

A relation instance:

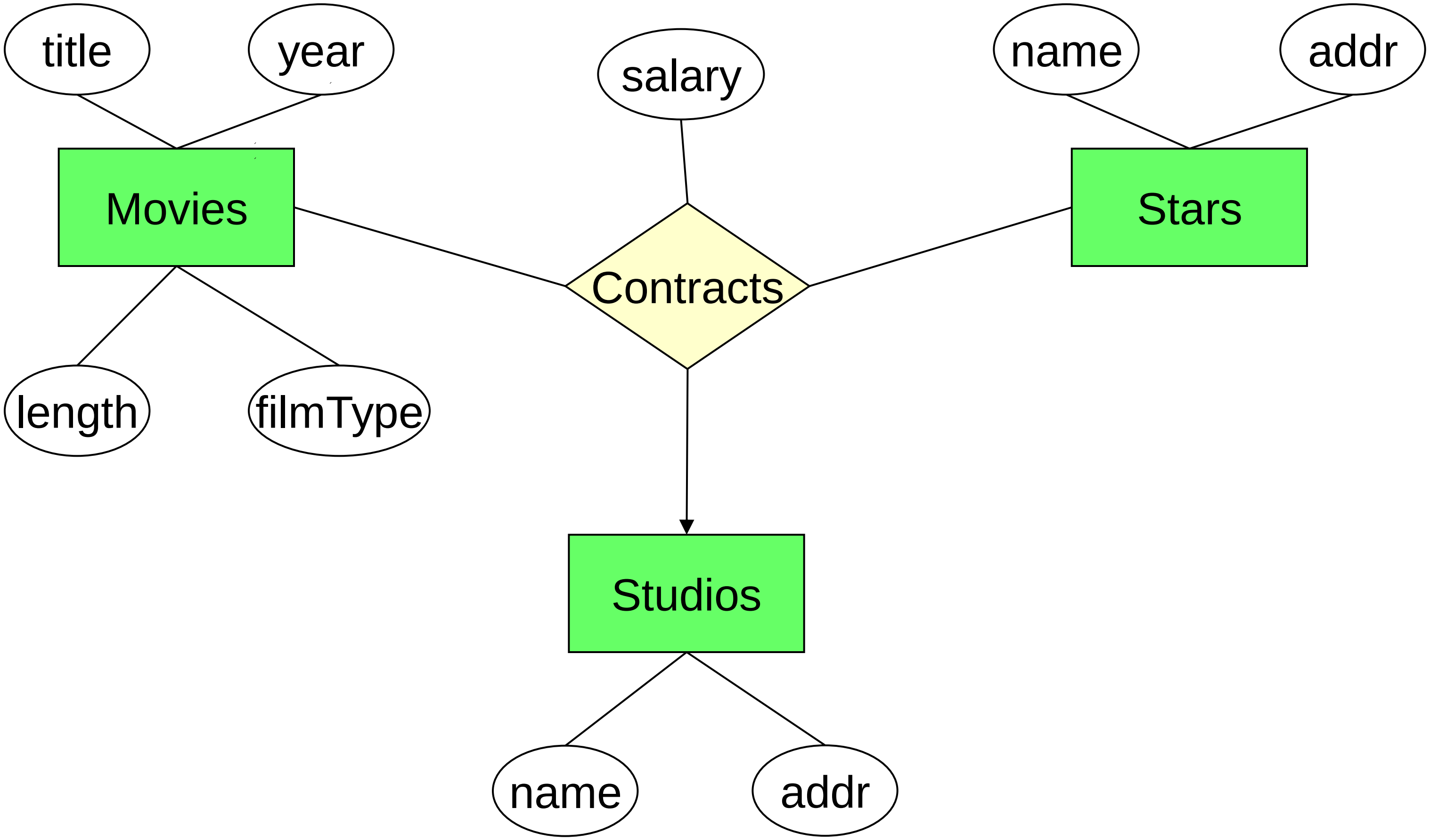
title	**year**	**length**	**filmtype**
Star Wars	1977	124	Color
Mighty Ducks	1991	104	Color
Wayne’s World	1992	95	Color

Example (with attrib. renaming)



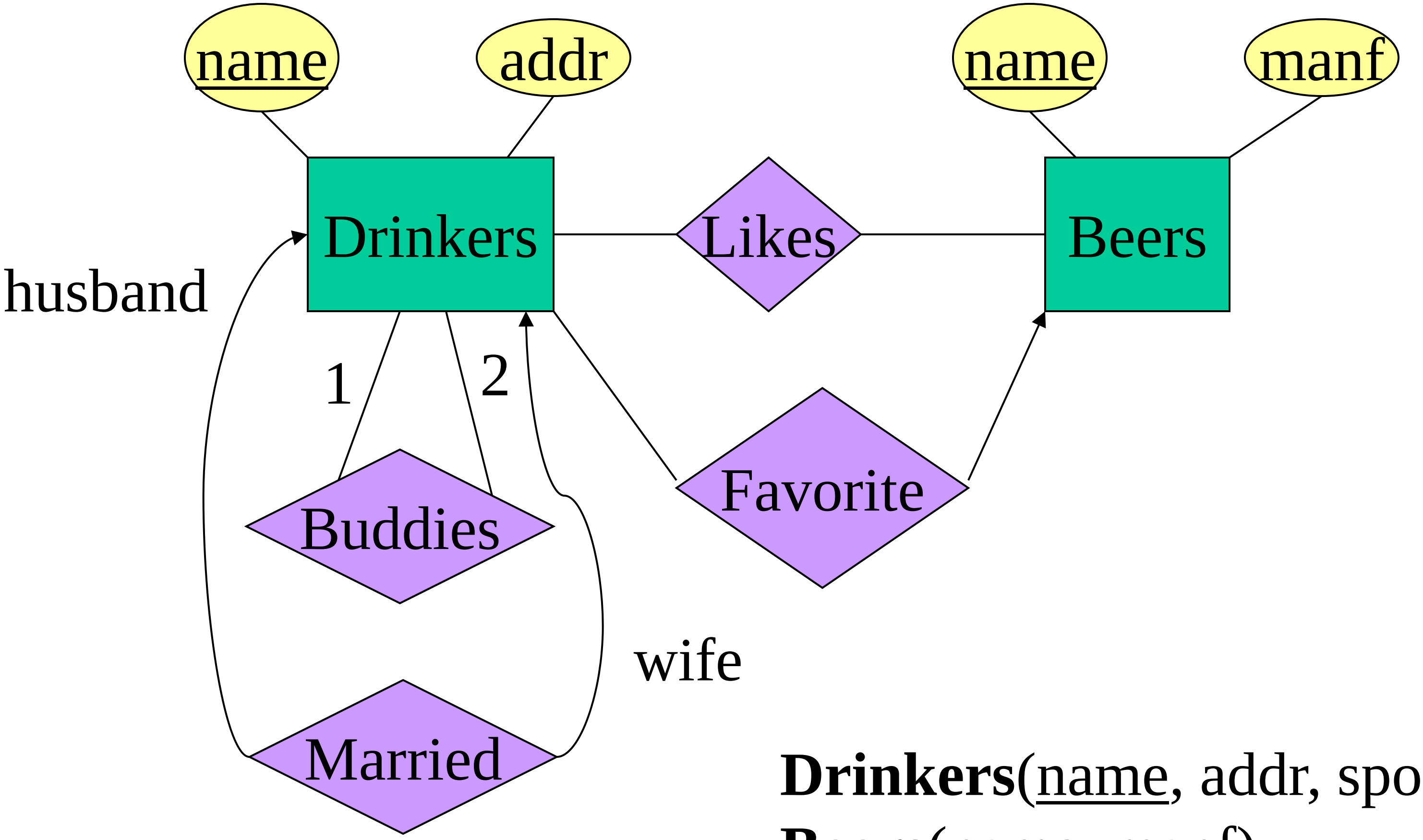
Movies(title, year, length, filmType, studioName)
Studios(name, address)
Stars(name, address)
StarsIn(title, year, starName)

Example



Movies(title, year, length, filmType)
Studios(name, address)
Stars(name, address)
Contracts(title, year, starName, studioName, salary)

Example

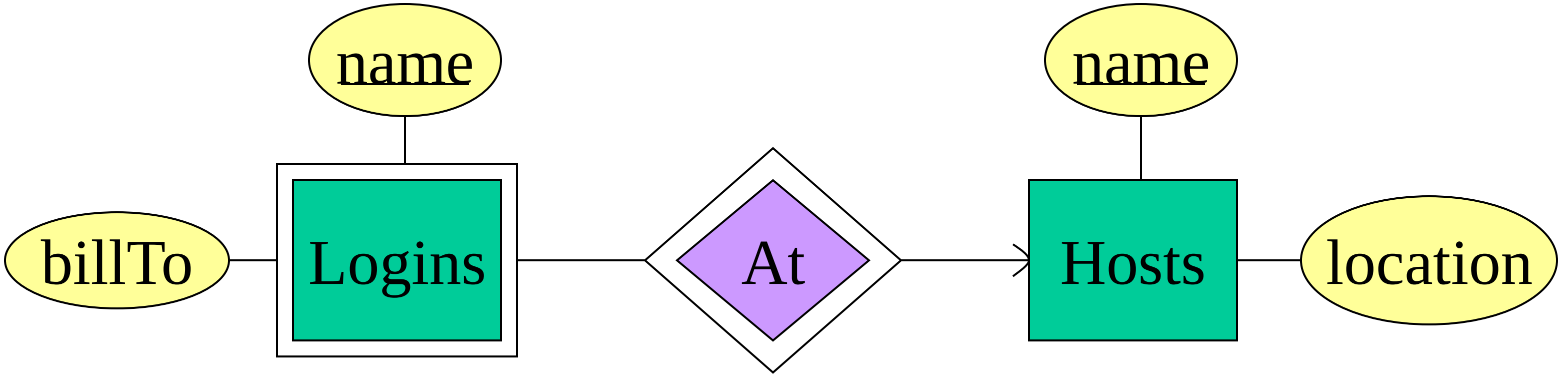


Drinkers(name, addr, spouse, favBeer)
Beers(name, manf)
Likes(drinker, beer)
Buddies(name1, name2)

Handling Weak Entity Sets

- Relation for a weak entity set must include attributes for its complete key (including those belonging to other entity sets), as well as its own, nonkey attributes.
- A supporting (double-diamond) relationship is redundant and yields no relation.

Example



Hosts(hostName, location)

Logins(loginName, hostName, billTo)

~~At(loginName, hostName, hostName2)~~

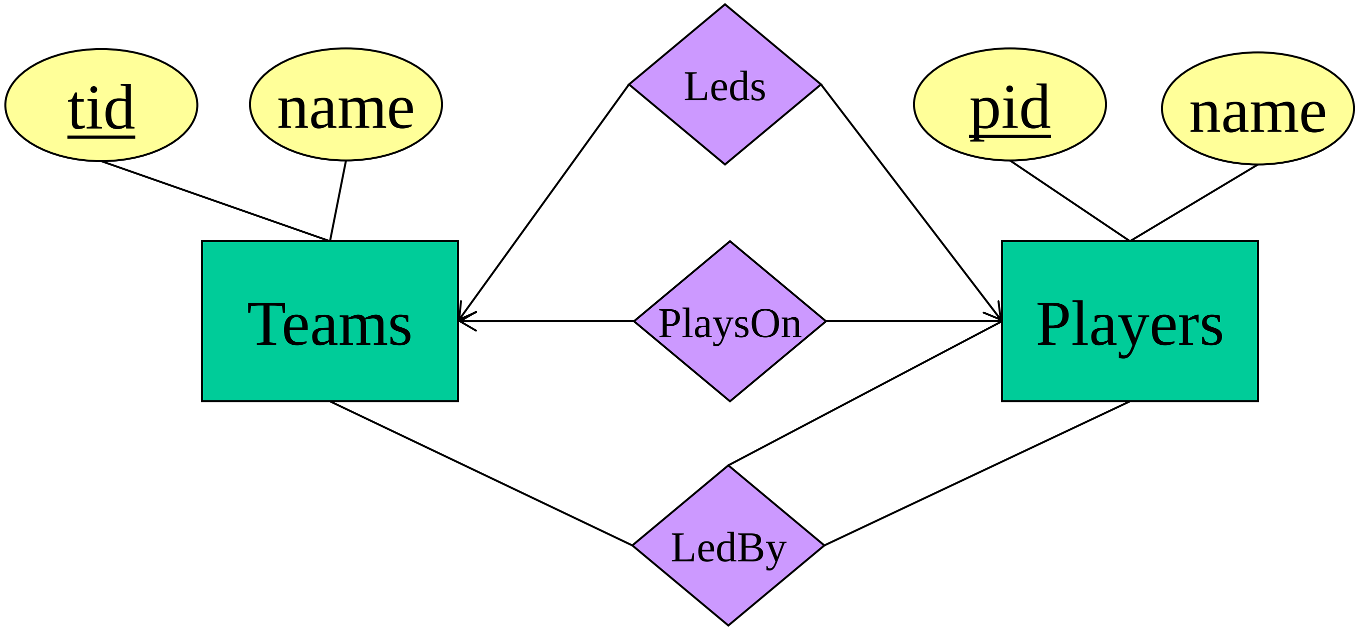
Must be the same

At becomes part of Logins

Example

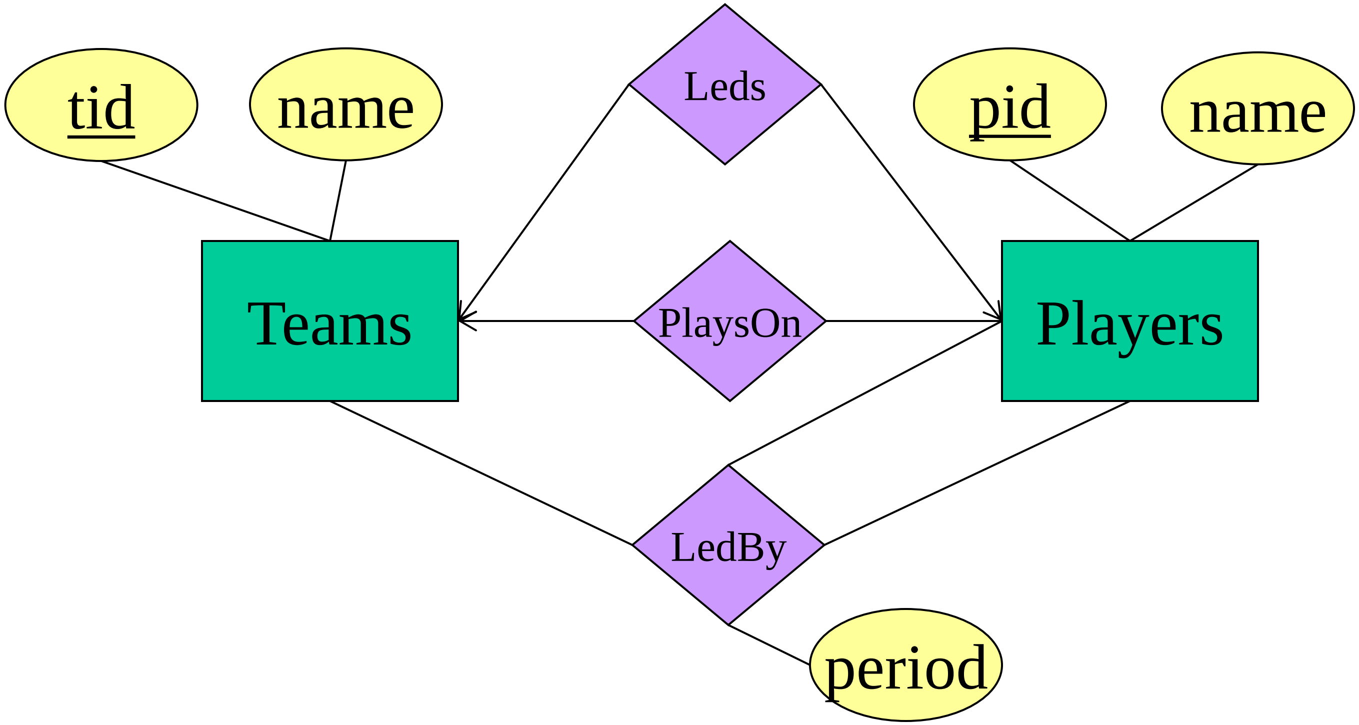
Teams, players, fans, lead by, etc…

Example (Fragment)



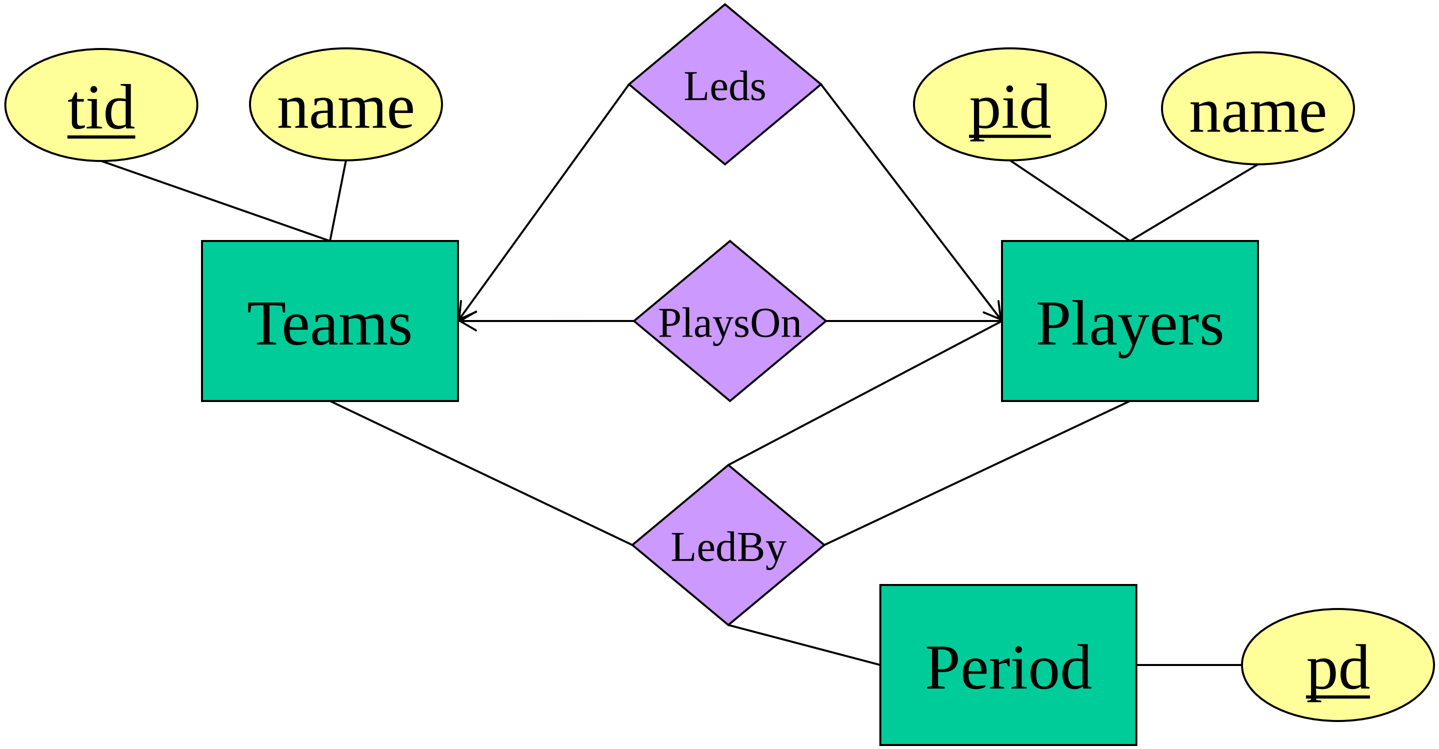
Teams(tid, name, pidCapt)
Players(pid, name, tid)
LedBy(pid1, pid2, tid)

Example (Fragment)



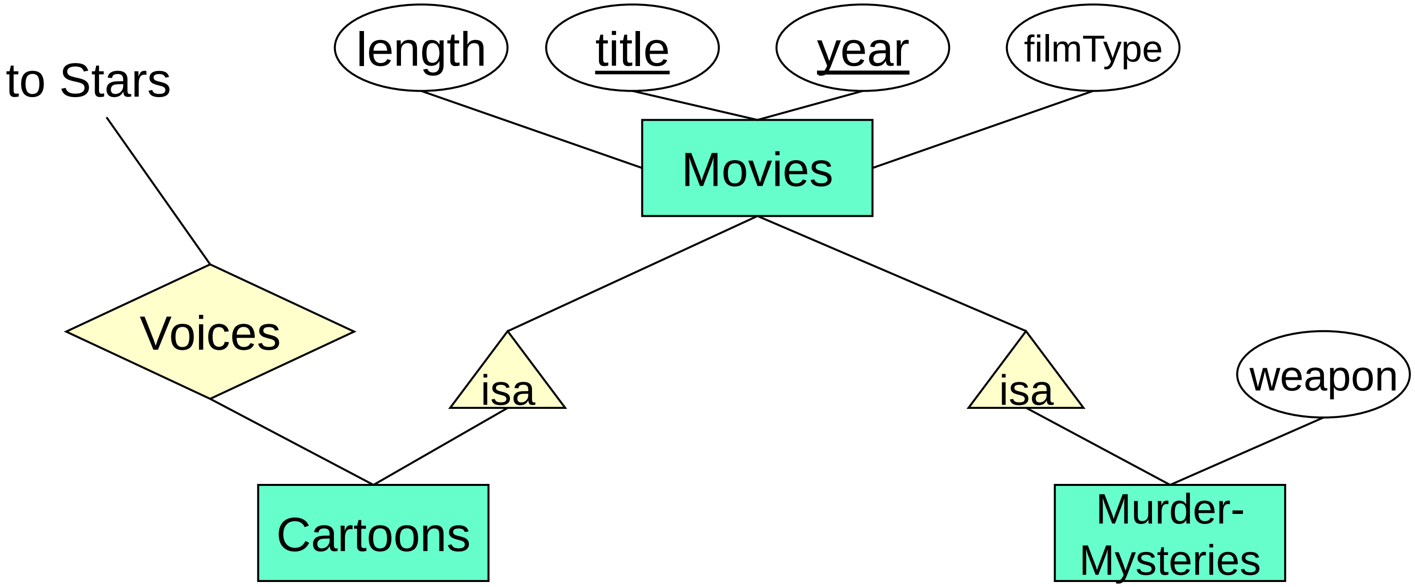
Teams(tid, name, pidCapt)
Players(pid, name, tid)
LedBy(pid1, pid2, tid, period)

Example (Fragment)



Teams(<u>tid</u>, name, pidCapt)
Players(<u>pid</u>, name, tid)
LedBy(<u>pid1</u>, <u>pid2</u>, <u>tid</u>, <u>pd</u>)

ISA


to Stars
length
title
year
filmType
Movies
Voices
isa
isa
weapon
Cartoons
Murder-
Mysteries

OO approach

- Every subclass has its own relation.
  - All the properties of that subclass, including all its inherited properties, are represented in this relation.

- **Example:**

Movies(*title, year, length, filmType* **)**

Cartoons(*title, year, length, filmType* **)**

MurderMysteries(*title, year, length, filmType, weapon***)**

Cartoon-MurderMysteries(*title, year, length, filmType, weapon***)**

Voices(title, year, starName **)**

- Can we merge **Cartoons** with **Movies**?
  - If we do, we lose information about which moves are cartoons.

E/R Approach

- We will have the following relations:
  - **Movies(***title, year, length, filmType***)**.
  - **MurderMystery(***title, year, weapon***)**.
  - **Cartoons(***title, year***)**.
  - **Voices(***title, year, name***)**.

E/R approach - Remarks

- No relation for class **Cartoon-MurderMystery***.*
- For a movie that is both, we obtain:
  - its voices from the **Voices** relation,
  - its weapon from the **MurderMystery** relation,
  - and all other information from the **Movies** relation.
- Relation **Cartoons** has a schema that is a **subset** of the schema for the relation **Voices***.* Should we eliminate the relation **Cartoons**?
- However there may be **silent** cartoons in our database. Those cartoons would have no voices and we would lose them.

Comparison of Approaches

OO translation **drawback**:

- Too many tables! Why?
  - In the OO approach if we have a root and **n** children we need **2^n** different tables!!!

E/R translation **drawback**:

- We may have to look in several relations to gather information about a single object.
  - For example, if we want the length and weapon used for a murder mystery film, we have to look at **Movies** and **MurderMysteries** relations.

Comparison of Approaches (Continued)

OO translation **advantage**:

- The **OO** translation keeps **all** properties of an object together in **one** relation.

E/R translation **advantage**:

- The **E/R** translation allows us to find in one relation tuples from all classes in the hierarchy.

Examples

- What movies of 2009 were longer than 150 minutes?
  - Can be answered directly in the E/R approach.
  - In the OO approach we have to examine all the relations.

- What weapons were used in cartoons of over 150 minutes in length?
  - More difficult in the E/R approach.
    - We should access **Movies** to find those of over 150 mins.
    - Then, we have to access **Cartoons** to see if they are cartoons.
    - Then we should access **MurderMysteries** to find the weapon.
  - In OO approach we need only access the **Cartoon-MyrderMysteries** table.

Null Values to Combine Relations

- If we are **allowed** to use **NULL** in tuples, we can handle a hierarchy of classes with a single relation.
  - This relation has attributes for all the properties possessed by objects in any of the classes of the hierarchy.
  - An object is represented by a single tuple. This tuple has NULL in each attribute corresponding to a property that does not belong to the object's class.

- If we apply this approach to the *Movie* hierarchy, we would create a single relation whose schema is:
  - **Movie(***title, year, length, filmType, studioName, starName, voice, weapon***)**
    - "*Who Framed Roger Rabbit?"*, being both a cartoon and a murder-mystery, would be represented by several tuples that had no NULL's.
    - *The Little Mermaid,* being a cartoon but not a murder-mystery, would have NULL in the *weapon* component.

- This approach allows us to find **all** the information about an object in one relation. Drawback?
  - Depending on the data, there could be too many nulls.