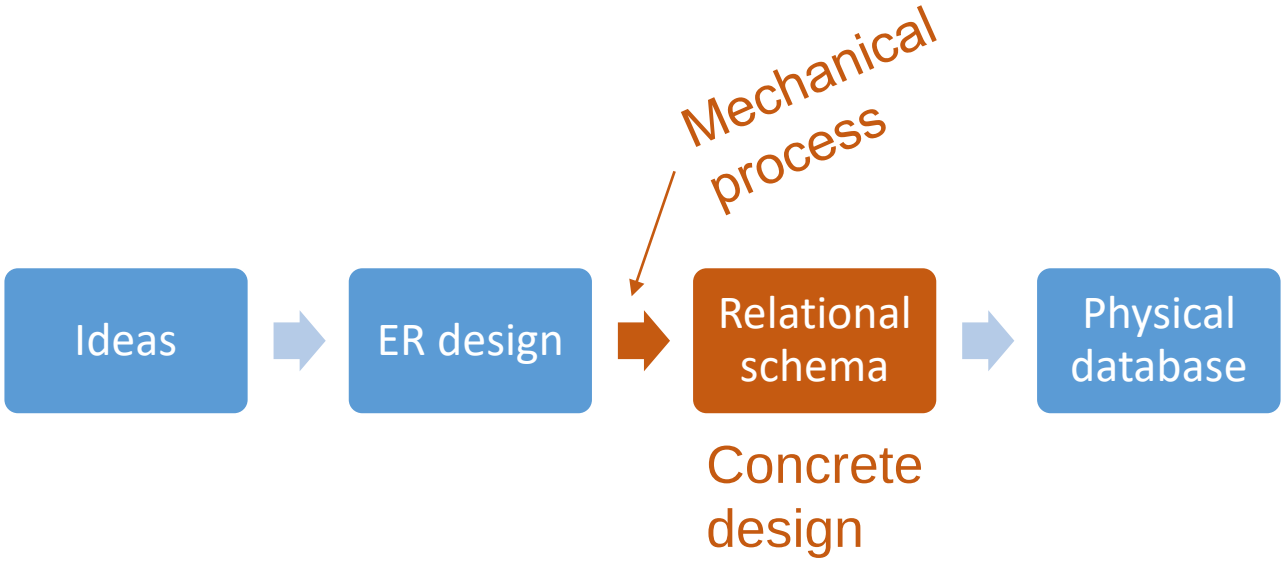
CMPT 321, Fall 2017

From E/R Diagrams to Relations (tables)

Lecture 01.03

By *Marina Barsky*

Rules of converting E/R diagrams to relations (tables)



Process of database design

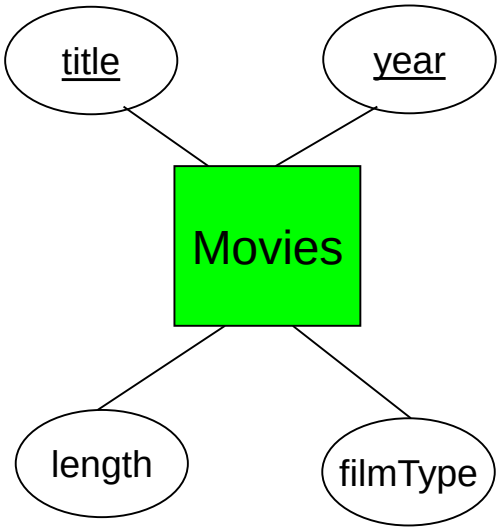
Terminology

- Every attribute of an entity has an *atomic* type

- Relation Schema: relation name + attribute names + attribute types
- Relation instance: set of tuples

- Database Schema: set of relation schemas
- Database instance: relation instance for every relation in the schema

Example



Attribute
Attribute type
Tuple: (Star Wars, 1977, 124, Color)

Relation Schema:
Movies (title:*string*, year:*int*, length:*int*, filmtype:*string*)

Relation **instance**:

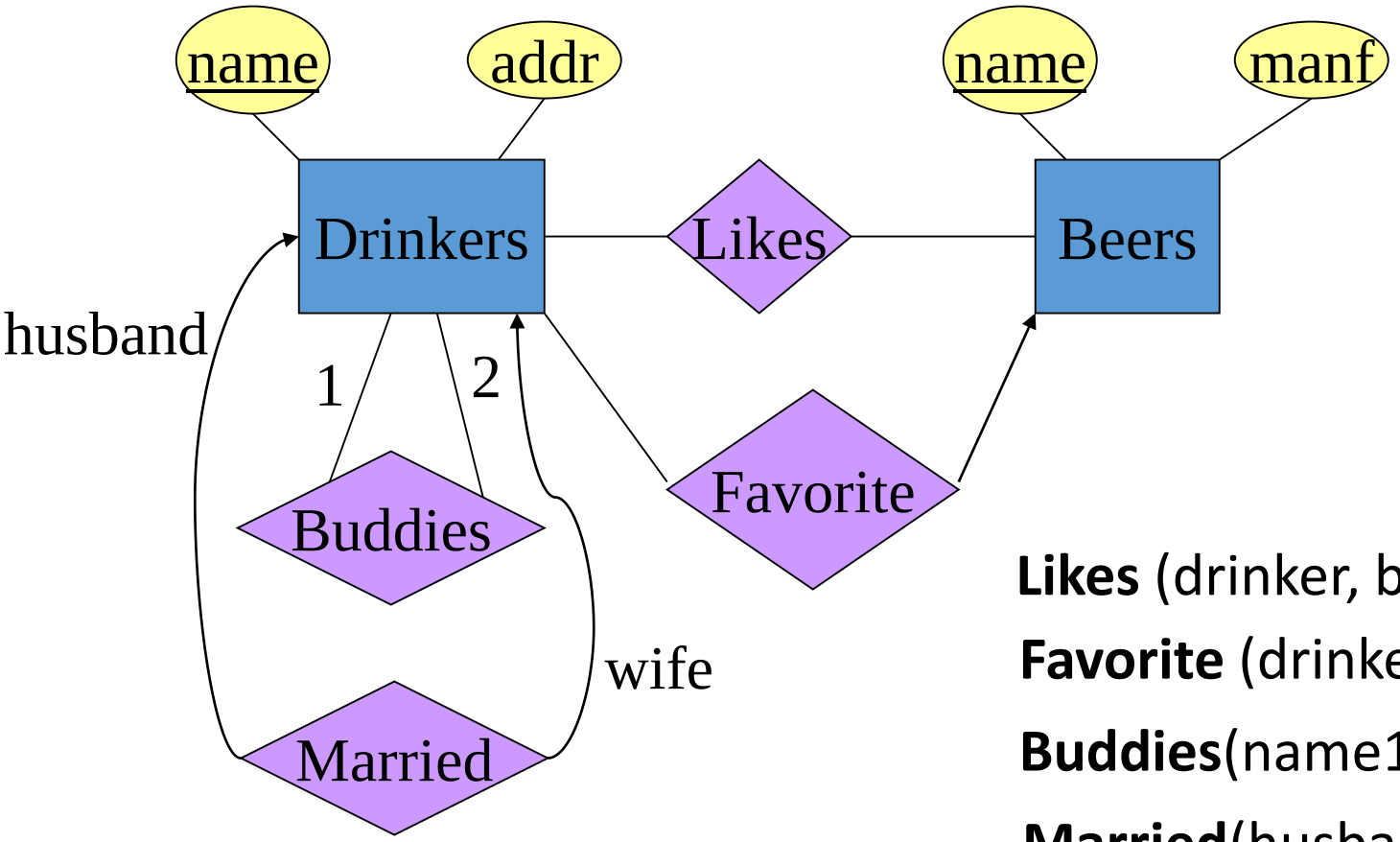
title	**year**	**length**	**filmtype**
Star Wars	1977	124	Color
Mighty Ducks	1991	104	Color
Wayne's World	1992	95	Color

From E/R to relational schema

- Each entity set becomes a relation.
  Its attributes are
  - the attributes of the entity set

- Each relationship becomes a relation
  It's attributes are
  - the keys of the entity sets that it connects, plus
  - the attributes of the relationship itself.

BBD to relations



Likes (drinker, beer)

Favorite (drinker, beer)

Buddies(name1, name2)

Married(husband, wife)

Example: relationship to relation (with Renaming)

Relationship **Stars-In** between entity sets **Movies** and **Stars** is represented by a relation with schema:

Stars-In(title, year, starName)

A sample instance is:

We rename here just for clarity.

title	*year*	***starName***
Star Wars	1977	Carrie Fisher
Star Wars	1977	Mark Hamill
Star Wars	1977	Harrison Ford
Mighty Ducks	1991	Emilio Estevez
Wayne's World	1992	Dana Carvey
Wayne's World	1992	Mike Meyers

Redundancy?
No, just multi-attribute keys

Relationship **Stars-In** between entity sets **Movies** and **Stars** is represented by a relation with schema:

Stars-In(title, year, starName)

A sample instance is:

title	*year*	*starName*
Star Wars	1977	Carrie Fisher
Star Wars	1977	Mark Hamill
Star Wars	1977	Harrison Ford
Mighty Ducks	1991	Emilio Estevez
Wayne's World	1992	Dana Carvey
Wayne's World	1992	Mike Meyers

Why not surrogate key for identifying each movie entity

- If we want to combine data from IMDB, MovieLens, Netflix – can only identify movies by name, year
- No globally accepted movie identifier exists
- Movies, video games vs. books (International Standard Book Number)

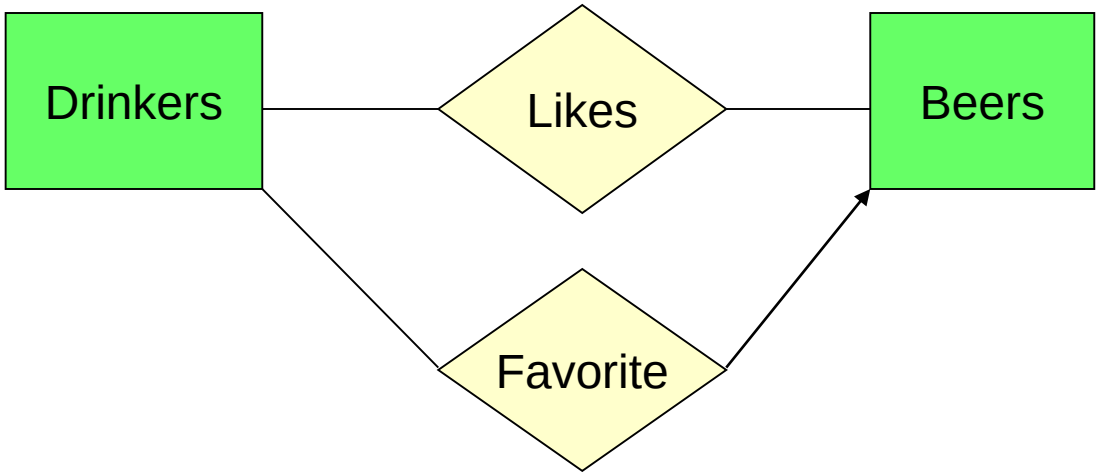
Redundancy? Yes

Stars-In(title, year, duration, starName)

A sample instance is:

title	*year*	*duration*	*starName*
Star Wars	1977	120	Carrie Fisher
Star Wars	1977	120	Mark Hamill
Star Wars	1977	120	Harrison Ford
Mighty Ducks	1991	130	Emilio Estevez
Wayne's World	1992	90	Dana Carvey
Wayne's World	1992	90	Mike Meyers

Many-One Relationships



- We not always have a separate relation for them.

Instead of having

Drinkers(name, addr) and

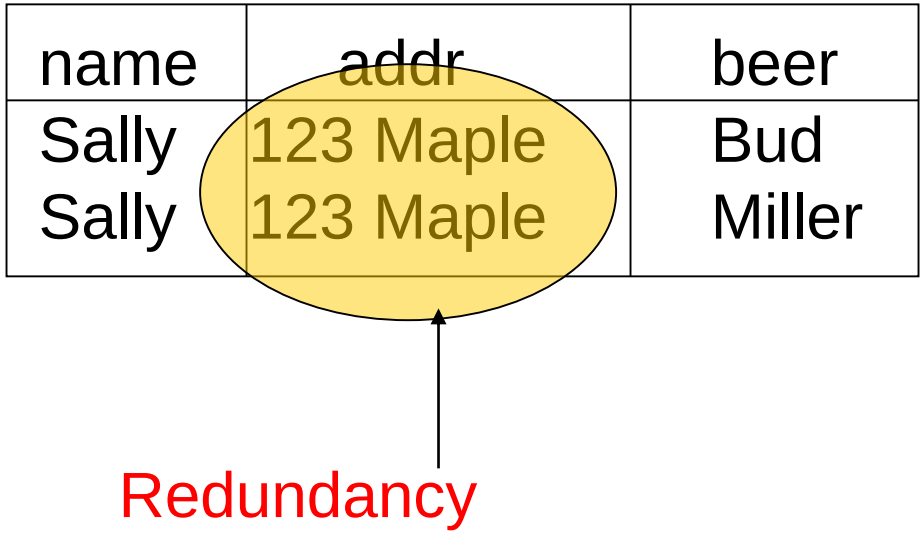
Favorite(drinker, beer)

have

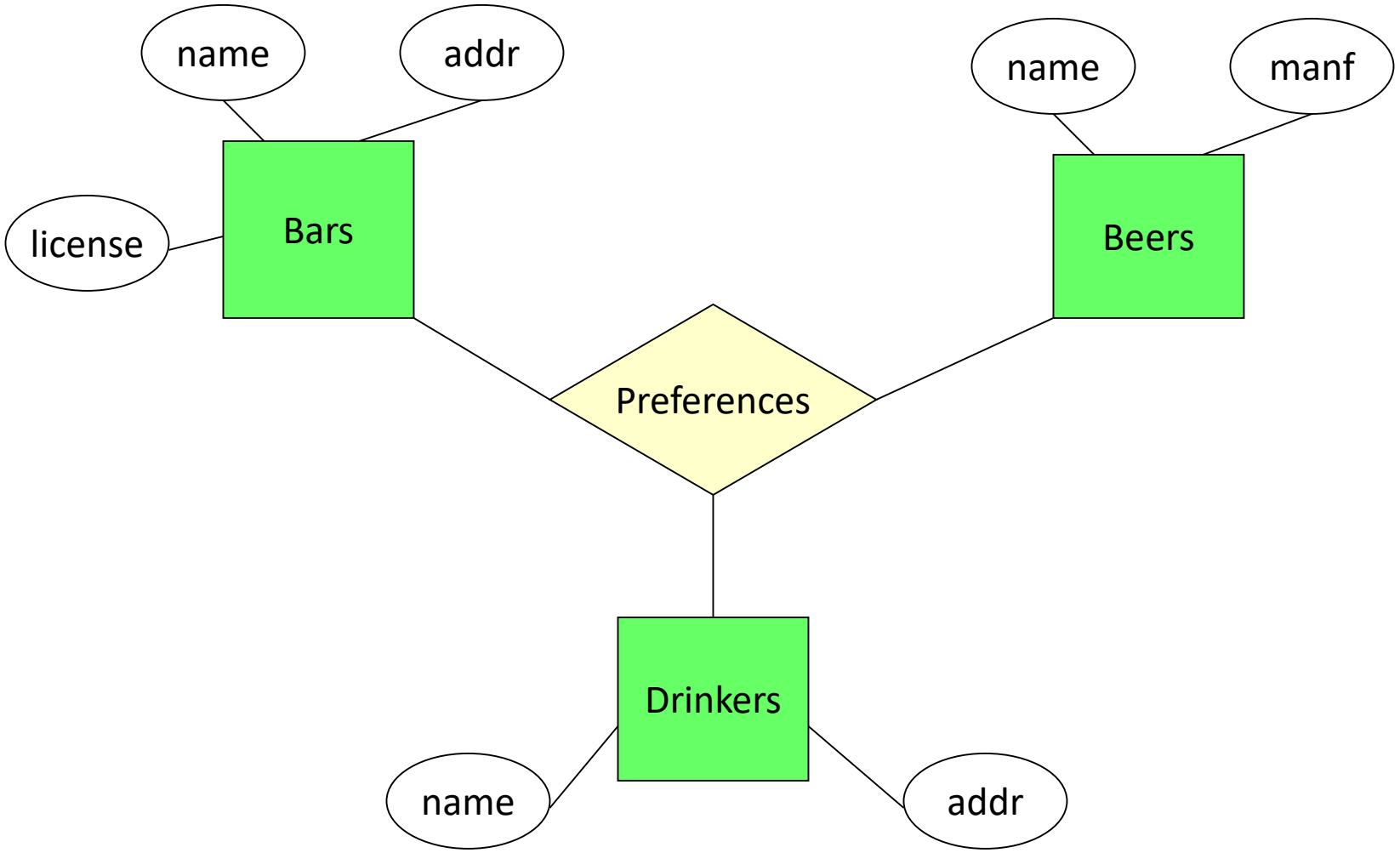
Drinkers(name, addr, favBeer)

Risk with Many-Many Relationships

- Combining **Drinkers** with **Likes** would be a mistake. Why?
- It leads to harmful redundancy, as:

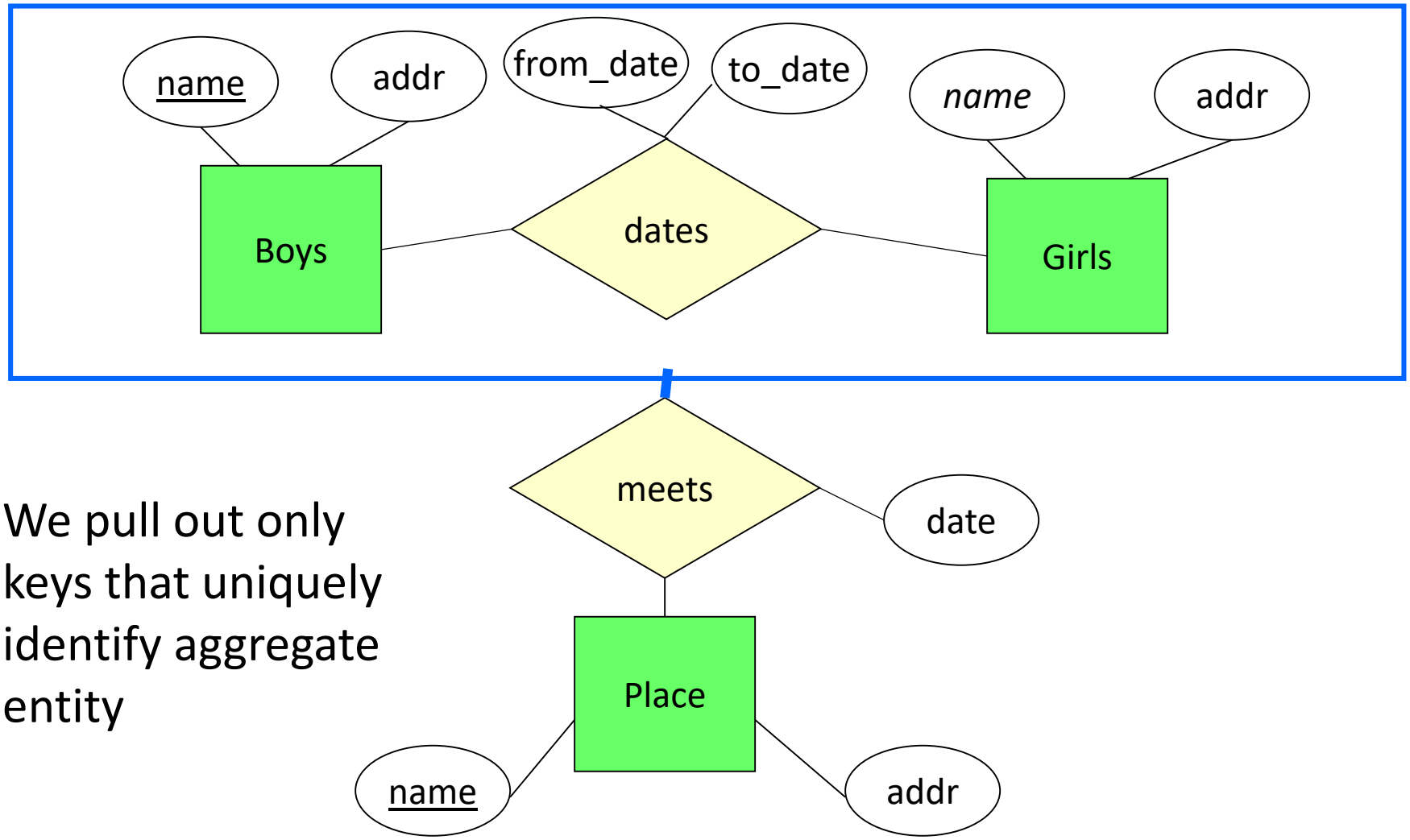


Many-to-Many Ternary Relationships



Preferences(drinker_name, beer_name, bar_name)

Aggregate entities



We pull out only keys that uniquely identify aggregate entity

Meets (boy_name, girl_name, place_name)

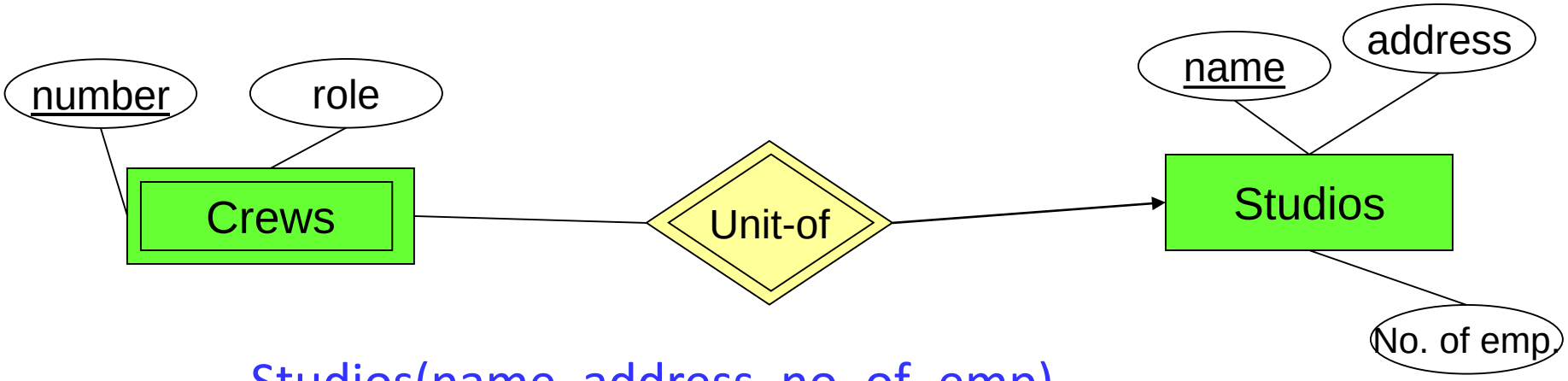
Handling weak entity sets

- We use week entity sets to identify sub-units of the main entity, rather than sub-classes

- 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, non-key attributes.

- A supporting (double-diamond) relationship is redundant and yields no relation.

Example 1: movies



Studios(name, address, no_of_emp)

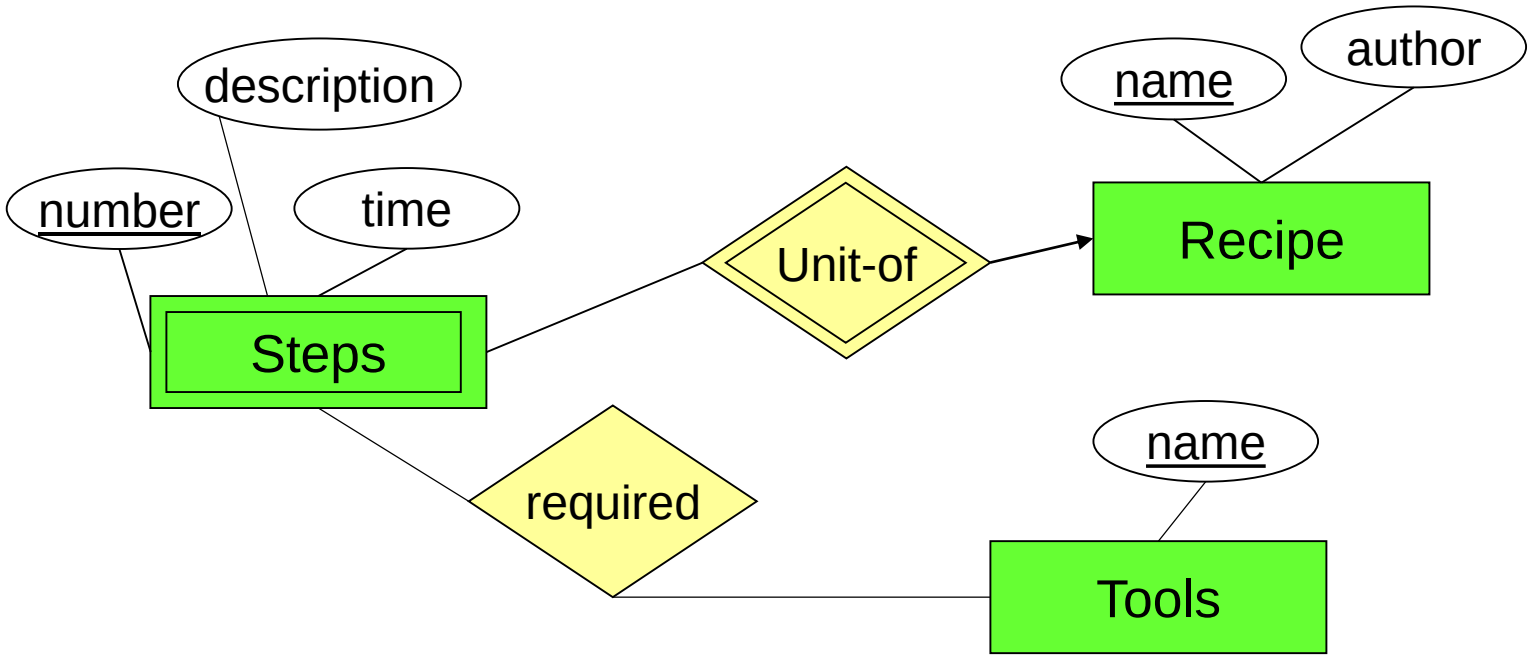
Crews(number, studioName, role)

~~Unit-of(number, studioName, studioName2)~~

Must be the same

Unit-of becomes part of Crews

Example 2: recipe



Recipe(name, author)
Steps(recipe_name, recipe_author, step_number, descr, time)
Tools(name,…)
Tools_required (recipe_name, recipe_author, step_number, tool_name)

Subclass Structures to Relations

Two main approaches

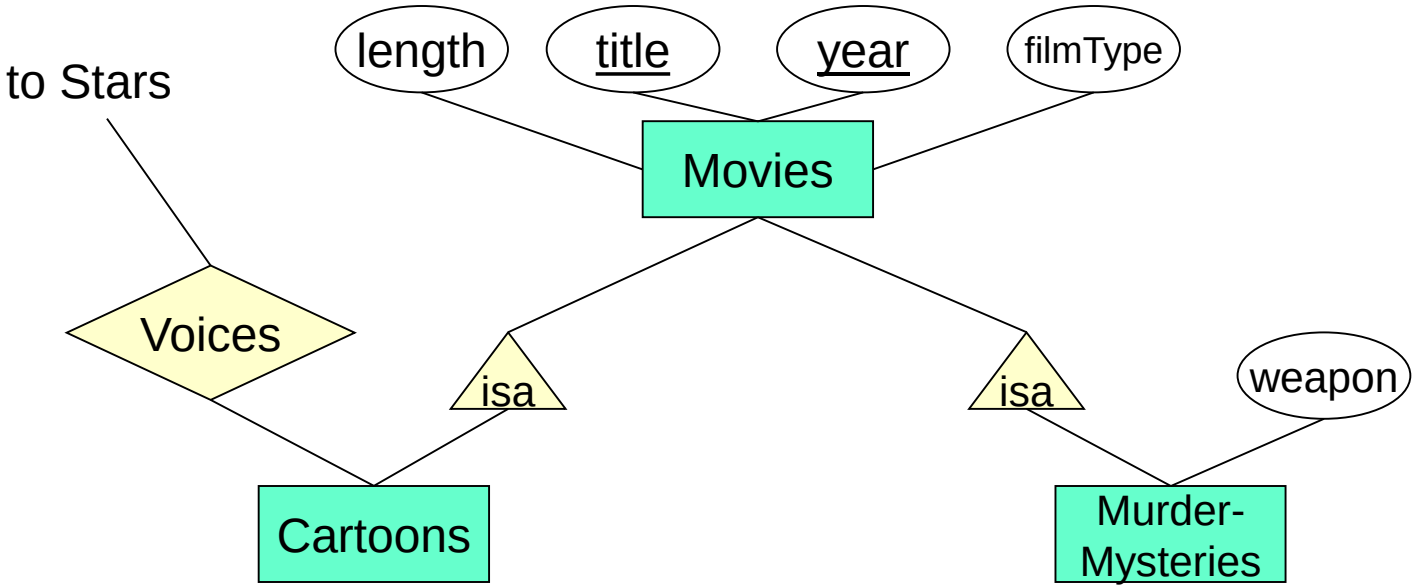
OO Approach

- An object belongs to **exactly one** class.
  - An object inherits properties from all its super-classes but it is **not** a member of them.

E/R Approach

- An "object" can be represented by entities belonging to several entity sets that are related by **isa** relationships.
  - The linked entities together represent the object and give that object all its properties (attributes and relationships).

Subclasses example



How to convert to relations?

OO approach: example

- 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 movies are cartoons.

- Is it necessary to create two relations ***voices:*** one connecting **cartoons** with **stars**, and one connecting **cartoon-murder-mysteries** with **stars**?
  - Not, really. We can use the same relation (table).

E/R Approach: example

- We will have the following relations:

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

  MurderMystery (*title, year, weapon***)**

  Cartoons (*title, year***)**

  Voices (*title, year, name***)**

- **Remarks:**
  - There is 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 basic 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 **advantage**:

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

OO translation **drawback**:

- Too many tables!
  - If we have a root and ***n*** children we need $\mathbf{2}^{\boldsymbol{n}}$ different tables!!!

Comparison of Approaches

E/R translation **advantage**:

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

E/R translation **drawback**:

- We may have to look in several relations to gather information about a single object

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-MurderMysteries** table.

Null Values to implement subclasses

- If we are **allowed** to use **NULL** in tuples, we can handle a hierarchy of classes with a single relation.

- For the *Movie* hierarchy, we would create a single relation:
  - **Movie (**title, year, length, filmType, studioName, starName, voice, weapon**)**
  - "*Who Framed Roger Rabbit?*", being both a cartoon and a murder-mystery, is represented by a tuple that had no NULL's.
  - *The "Little Mermaid,"* being a cartoon but not a murder-mystery, has NULL in the *weapon* component.

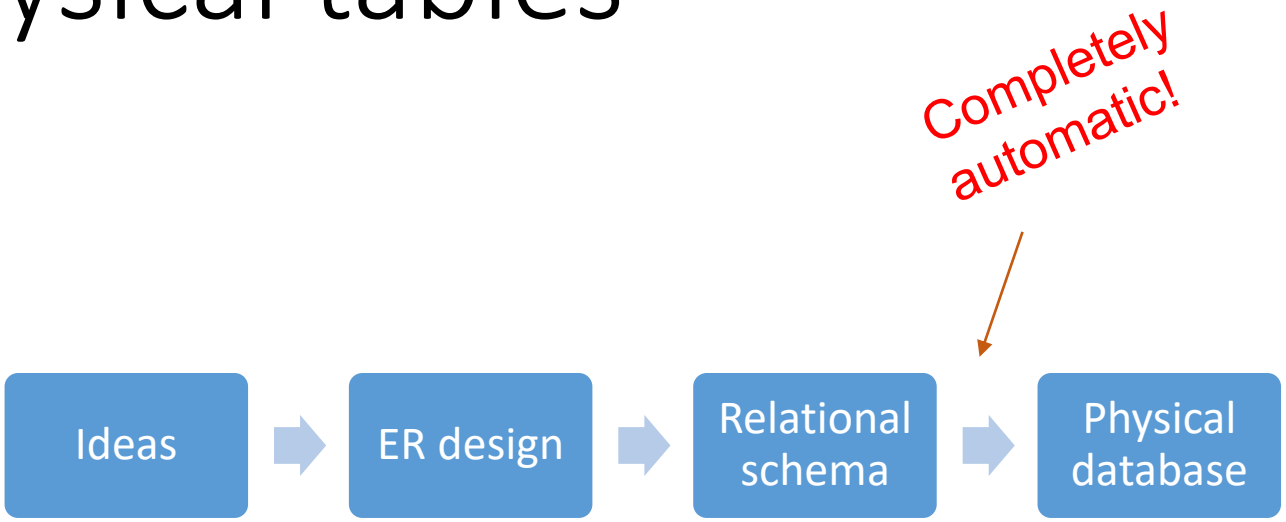
- This approach allows us to find **all** the information about an object in one relation. Drawback?

How to ensure that the schema is "good"?

- The process of translation should ensure that there is no redundancy.
  - But only with respect to what the E/R diagram represents.

- Crucial thing we are missing: functional dependencies (We only have keys, not other FDs.)
- So we still need to learn the design theory to fully eliminate redundancy

Converting logical schema into physical tables



Details slightly differ for each DBMS

Data Definition Language (DDL): converitng Schema into physical tables

CREATE TABLE **table_name**
(
    column_name1 *data_type*,
    column_name2 *data_type*,
    column_name3 *data_type*,
    ....
)

SQLite3

- Fast, small-footprint, installation-free database, well suited for data analysis.

https://www.sqlite.org/whentouse.html

- Just download sqlite3 and start creating databases and querying them

http://www.sqlite.org/download.html

Creating tables in SQLite (in file *movie_tables.sql*)

```
DROP TABLE if exists MovieStar;
/* Delete table if it already exists */


CREATE TABLE MovieStar(
        name VARCHAR (50) PRIMARY KEY,
        address text,
        gender char(1),
        birthdate char(20)
);
```

Creating tables in SQLite (in file *movie_tables.sql*)

```
DROP TABLE if exists Movie;
/* Delete table if it already exists */

CREATE TABLE Movie (
        title varchar(30),
        year int,
        length int,
        inColor int,
        studioName varchar(20),
        producerC varchar(3),
        primary key (title, year)
 );
```

Creating database in SQLite

- Launch sqlite in the terminal or command prompt:
  sqlite3

```
SQLite version 3.13.0 2016-05-18 10:57:30
sqlite> .open movies
sqlite> .read movie_tables.sql

sqlite> SELECT name FROM sqlite_master WHERE type='table';
```

Creates database named movies

Runs sql script to create empty tables

To see all the tables

SQLite data types

- TEXT
- NUMERIC
- INTEGER
- REAL
- BLOB

Date and time

- SQLite does not support date and time storage classes. You can use the TEXT, INT, or REAL to store date and time values:

1. TEXT: A date in a format like "YYYY-MM-DD HH:MM:SS.SSS"
2. REAL: The number of days since noon in Greenwich on November 24, 4714 B.C.
3. INTEGER: The number of seconds since 1970-01-01 00:00:00 UTC

- You can choose to store dates and times in any of these formats and freely convert between formats using the built-in date and time functions.

Populate tables with data

```
INSERT INTO Movie (title, year, length, inColor, studioName, producerC)
VALUES('Godzilla', 1998, 120, 1 , 'Paramount', 123);
```