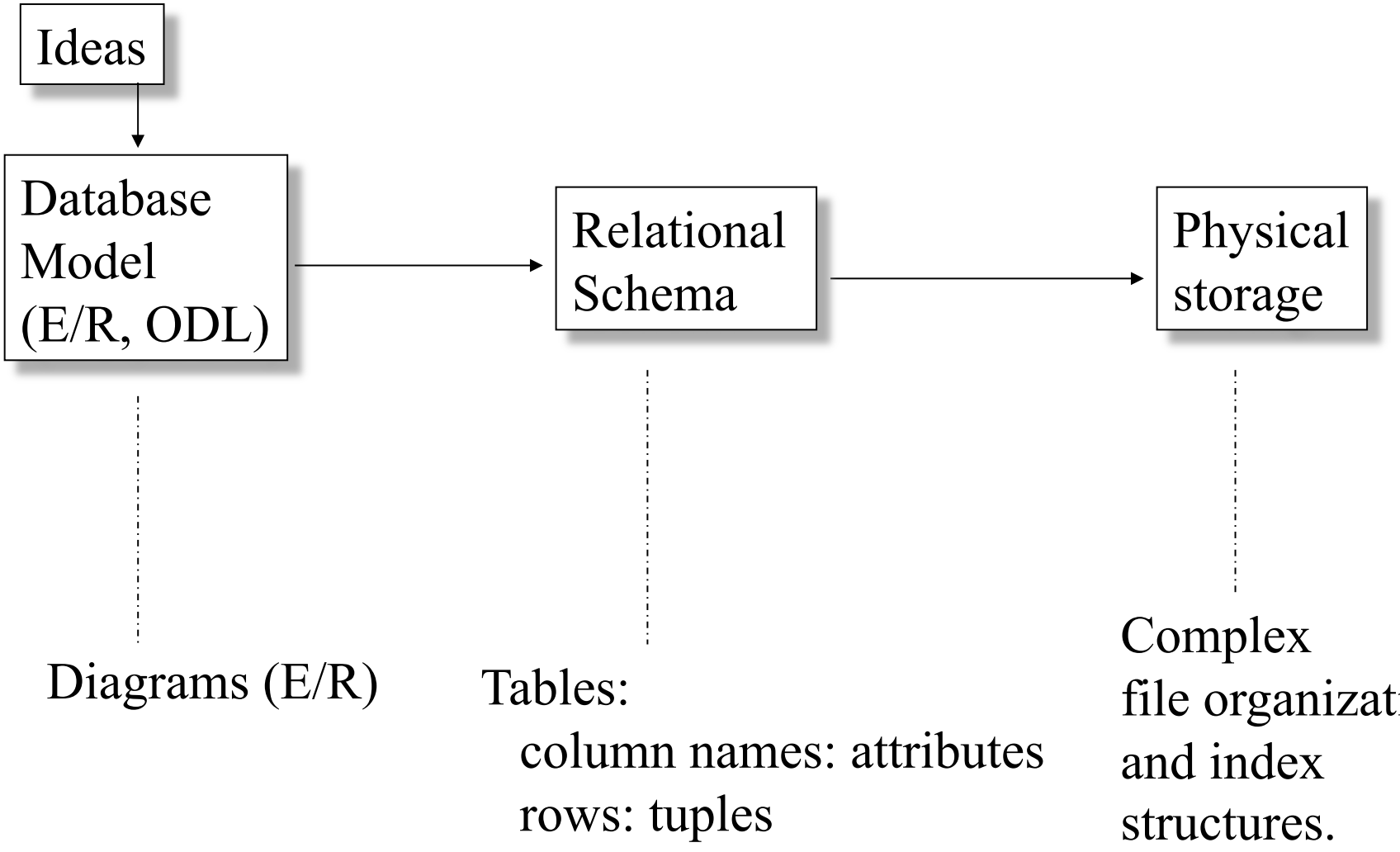
CS411
Database Systems

03: Relational Model

Text: 2.2-2.3, 4.5-4.6

Why Do We Learn This?

Database Modeling & Implementation

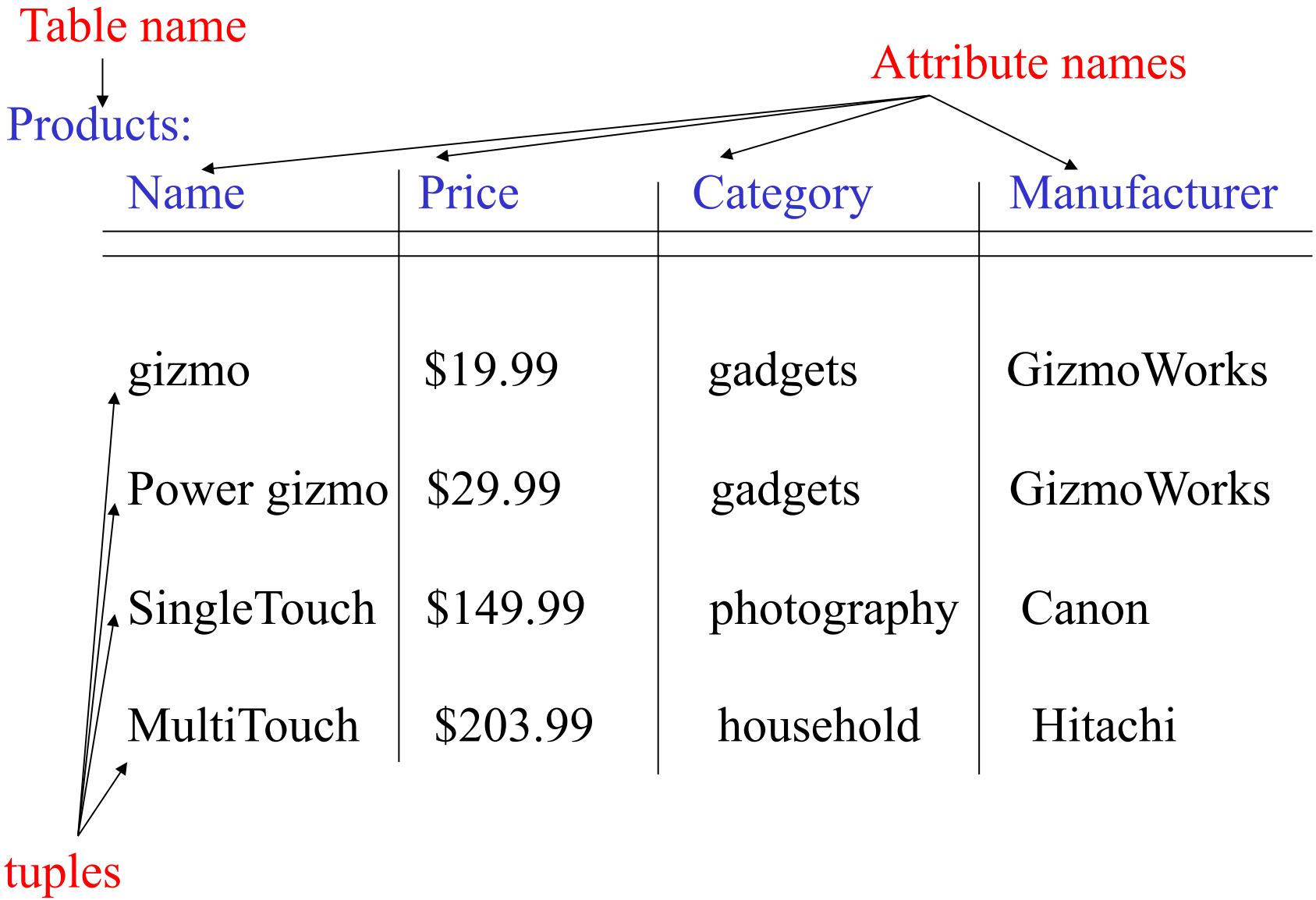


ER Model vs. Relational Model

- Both are used to model data
- ER model has many concepts
  - entities, relations, attributes, etc.
  - well-suited for capturing the app. requirements
  - not well-suited for computer implementation
  - (does not even have operations on its structures)
- Relational model
  - has just a single concept: relation
  - world is represented with a collection of tables
  - well-suited for efficient manipulations on computers

The Basics

An Example of a Relation



Domains

- Each attribute has a type
- Must be atomic type
- Called *domain*
- Examples:
  - Integer
  - String
  - Real
  - …

Equivalent representations of a relation

- A relation is a set of tuples, not a list of tuples
- Reordering the rows in a table does not change the relation.
- Each tuple is an assignment of values to each attribute of the relation.
- Reordering the columns in a table does not change the relation.

Example

Name	Price	Category	Manufacturer
gizmo	$19.99	gadgets	GizmoWorks
Power gizmo	$29.99	gadgets	GizmoWorks
SingleTouch	$149.99	photography	Canon
MultiTouch	$203.99	household	Hitachi

Same as …

Name	Price	Category	Manufacturer
gizmo	$19.99	gadgets	GizmoWorks
SingleTouch	$149.99	photography	Canon
Power gizmo	$29.99	gadgets	GizmoWorks
MultiTouch	$203.99	household	Hitachi

Same as …

Name	Category	Manufacturer	Price
gizmo	gadgets	GizmoWorks	$19.99
Power gizmo	gadgets	GizmoWorks	$29.99
SingleTouch	photography	Canon	$149.99
MultiTouch	household	Hitachi	$203.99

Schemas vs. Instances

Schemas

The Schema of a Relation:

- Relation name plus attribute names
- E.g. Product(Name, Price, Category, Manufacturer)
- In practice we add the domain for each attribute

The Schema of a Database

- A set of relation schemas
- E.g. Product(Name, Price, Category, Manufacturer),
  Vendor(Name, Address, Phone),
  . . . . . .

Instances

Relational schema: Product(Name, Price, Category, Manufacturer)

Instance:

Name	Price	Category	Manufacturer
gizmo	$19.99	gadgets	GizmoWorks
Power gizmo	$29.99	gadgets	GizmoWorks
SingleTouch	$149.99	photography	Canon
MultiTouch	$203.99	household	Hitachi

Instances

- **Relational schema** = R(A1,…,Ak):
  Instance = Set of tuples, with each tuple being the values of corresponding domains

- **Database schema** = R1(…), R2(…), …, Rn(…)
  Instance = n relation instances, of types R1, R2, ..., Rn

Updates

The database maintains a current database state.

Updates to the data:

1) add a tuple
2) delete a tuple
3) modify an attribute in a tuple

Updates to the data happen very frequently.

Updates to the schema: relatively rare. Rather painful. Why**?**

Keys

- A set of attributes forms a *key* for a relation if no two tuples in a relation instance can have the same values for all those attributes.

- Similar to the *key* in the ER model.

Defining a Database Schema in SQL: The *Data Definition Language* (DDL)

Defining a Database Schema

- A database schema comprises declarations for the relations (“tables”) of the database.
- Many other kinds of elements may also appear in the database schema, including views, indexes, and triggers, which we’ll introduce later.

Declaring a Relation

- Simplest form is:

  CREATE TABLE <name> (

  <list of elements>

  );

- And you may remove a relation from the database schema by:

  DROP TABLE <name>;

Elements of Table Declarations

- The principal element is a pair consisting of an attribute and a type.
- The most common types are:
  - INT or INTEGER (synonyms).
  - REAL or FLOAT (synonyms).
  - CHAR(*n*) = fixed-length string of *n* characters.
  - VARCHAR(*n*) = variable-length string of up to *n* characters.
  - DATE = ‘yyyy-mm-dd’
  - TIME = ‘hr:min:sec” (e.g., 15:30:02.5’ = two and a half seconds after 3:30PM.)

Example: Create Table

```
CREATE TABLE Sells (
    bar     CHAR(20),
    beer        VARCHAR(20),
    price       REAL
);
```

Declaring Keys

- An attribute or list of attributes may be declared PRIMARY KEY or UNIQUE.

- Implies that no two tuples may agree on all those attributes.

Declaring Single-Attribute Keys

- Place PRIMARY KEY or UNIQUE after the type in the declaration of the attribute.
- Example:

```
CREATE TABLE Beers (
    name    CHAR(20) UNIQUE,
    manf    CHAR(20)
);
```

Declaring Multiattribute Keys

- A key declaration can also be another element in the list of elements of a CREATE TABLE statement.

Example: Multiattribute Key

- The bar and beer together are the key for Sells:

```
CREATE TABLE Sells (
    bar     CHAR(20),
    beer    VARCHAR(20),
    price   REAL,
    PRIMARY KEY (bar, beer)
);
```

- This form is essential if the key consists of more than one attribute.
  - May be used even for one-attribute keys.

PRIMARY KEY Versus UNIQUE

- The SQL standard allows DBMS implementers to make their own distinctions between PRIMARY KEY and UNIQUE.
  - Example: some DBMS might automatically create an *index* (data structure to speed search) in response to PRIMARY KEY, but not UNIQUE.

Required Distinctions

- However, standard SQL requires these distinctions:
  1. There can be only one PRIMARY KEY for a relation, but several UNIQUE attributes.
  2. No attribute of a PRIMARY KEY can ever be NULL in any tuple. But attributes declared UNIQUE may have NULL's, and there may be several tuples with NULL.

Other Declarations for Attributes

- Two other declarations we can make for an attribute are:
  1. NOT NULL means that the value for this attribute may never be NULL.
  2. DEFAULT <value> says that if there is no specific value known for this attribute's component in some tuple, use the stated <value>.

Example: Default Values

```
CREATE TABLE Drinkers (
   name CHAR(30) PRIMARY KEY,
   addr CHAR(50)
      DEFAULT '123 Sesame St.',
   phone CHAR(16)
);
```

•Default value is "NULL" unless explicitly specified as above.

Effect of Defaults

- Suppose we insert the fact that Sally is a drinker, but we know neither her address nor her phone.
- But what tuple appears in Drinkers?

name	addr	phone
'Sally'	'123 Sesame St'	NULL

- If we had declared phone NOT NULL, this insertion would have been rejected.

Adding Attributes

- We may change a relation schema by adding a new attribute ("column") by:

  ALTER TABLE <name> ADD

  <attribute declaration>;

- Example:

```
ALTER TABLE Bars ADD
phone CHAR(16)DEFAULT 'unlisted';
```

Deleting Attributes

- Remove an attribute from a relation schema by:

  ALTER TABLE <name>

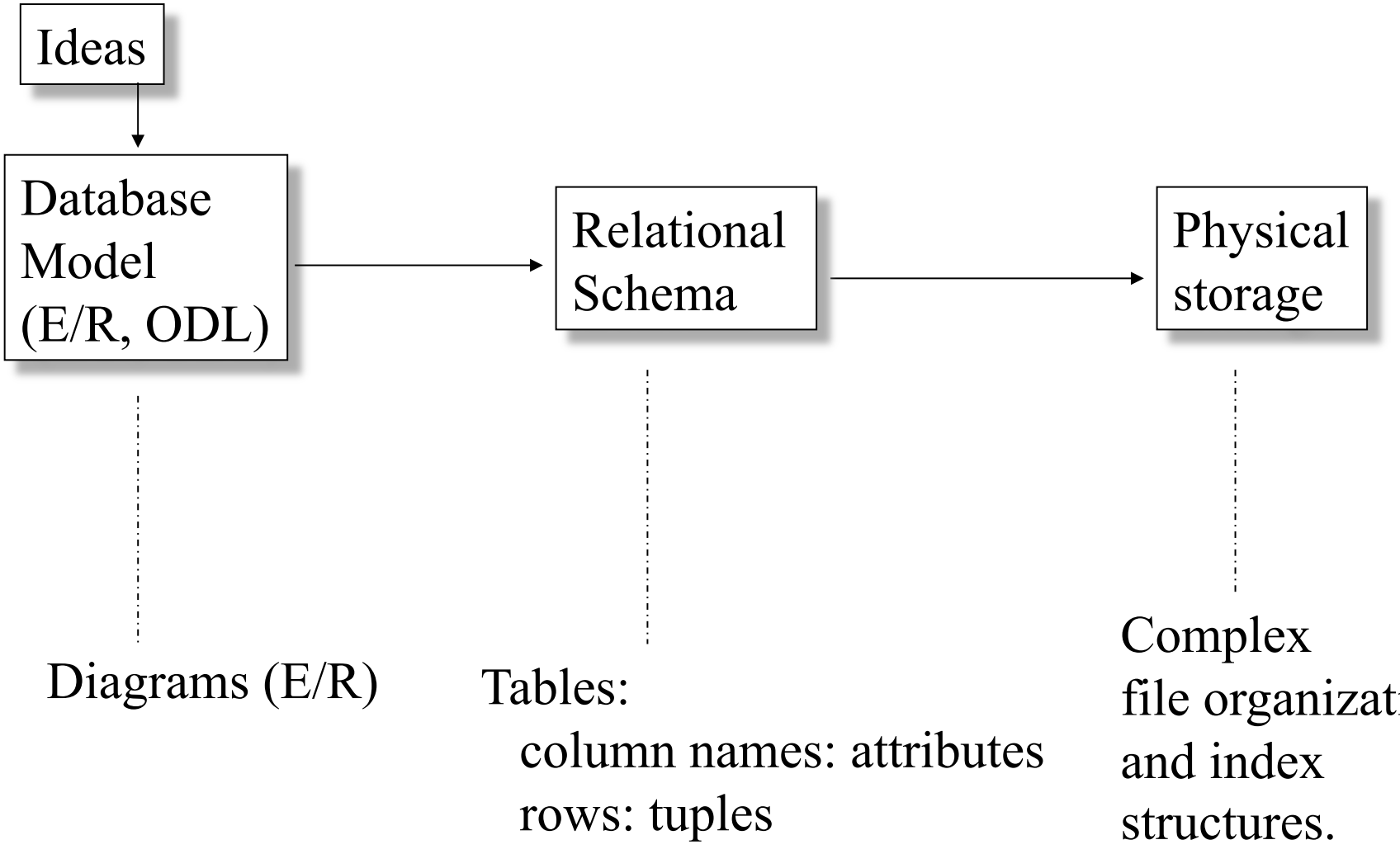
  DROP <attribute>;

- Example: we don't really need the license attribute for bars:

```
ALTER TABLE Bars DROP license;
```

ER Model → Relational Model

Database Modeling & Implementation

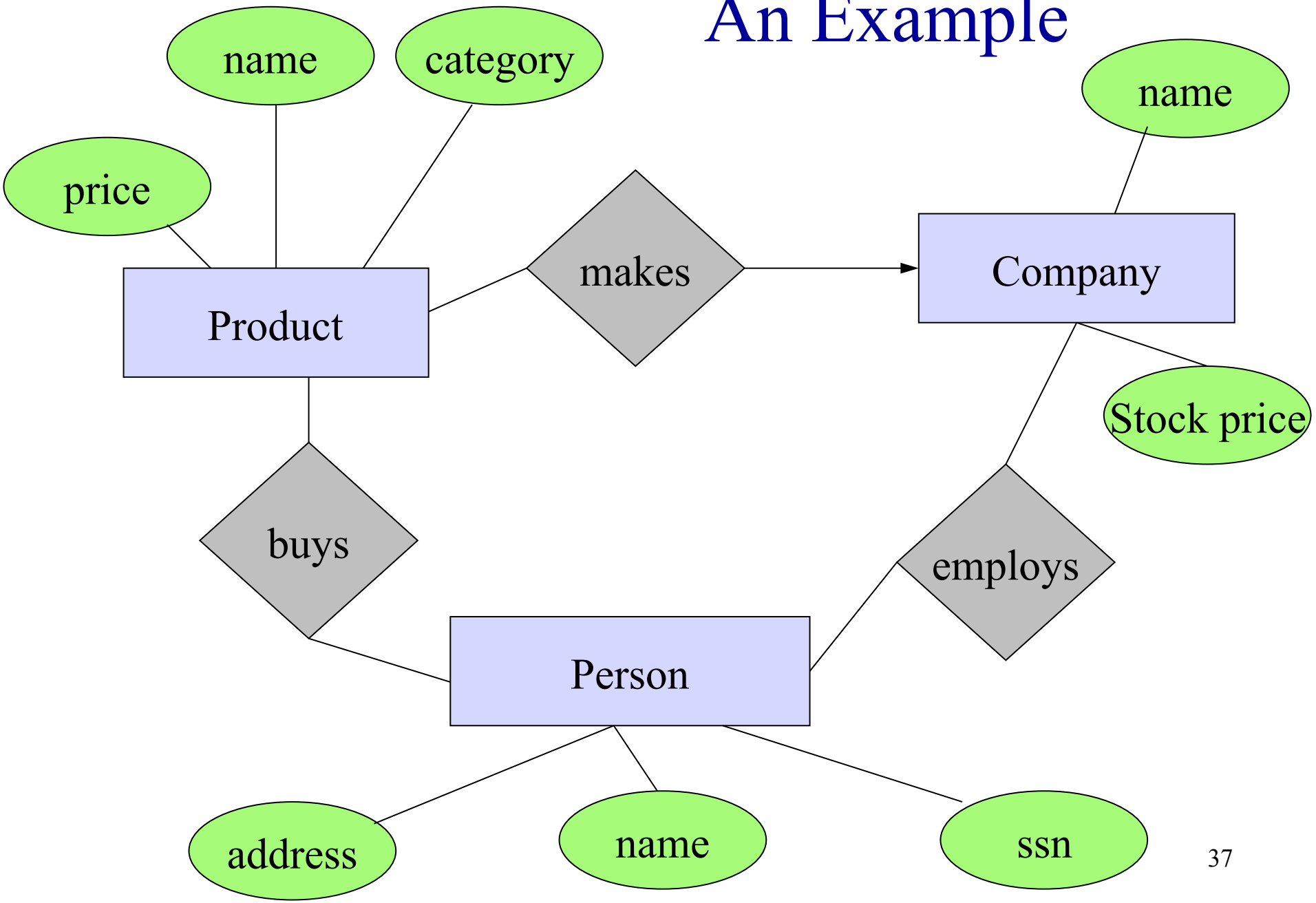


Translating ER Diagram to Rel. Design

- Basic cases
  - entity set E = relation with attributes of E
  - relationship R = relation with attributes being keys of related entity sets + attributes of R

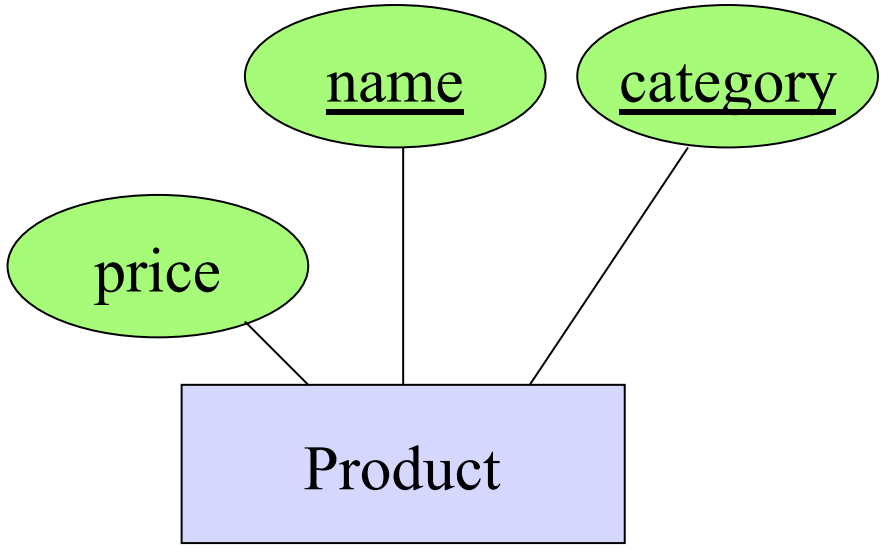
- And some special cases we'll cover afterwards

An Example


name
category
price
Product
makes
Company
name
Stock price
buys
employs
Person
address
name
ssn

Basic Cases

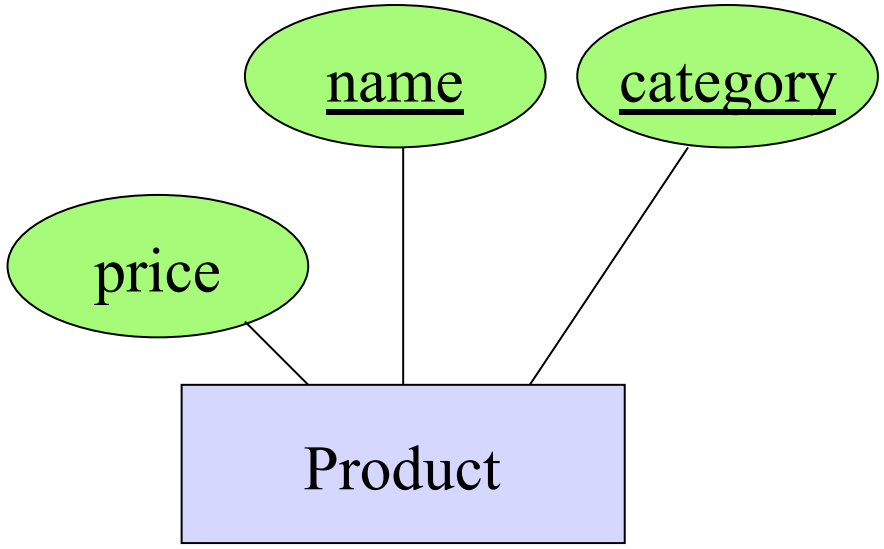
Entity Set to Relation



Product:

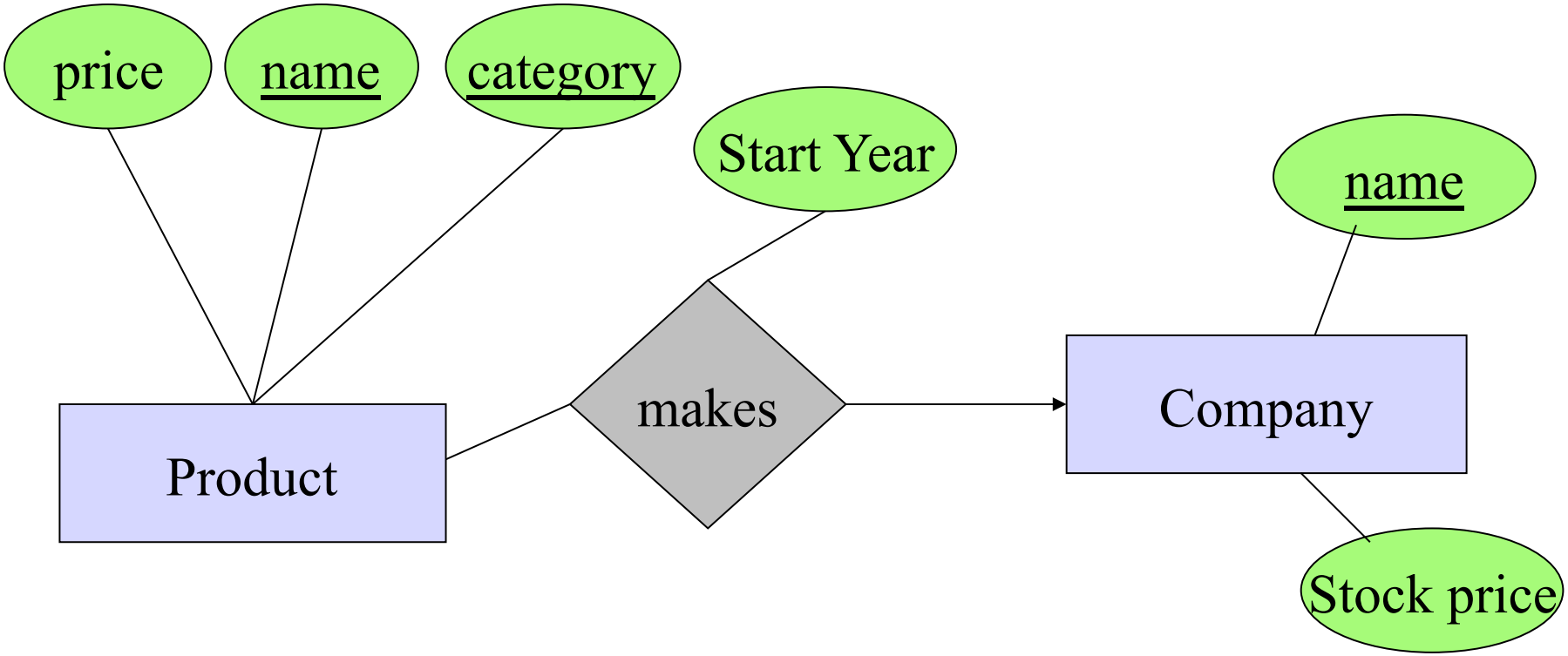
Name	Category	Price
gizmo	gadgets	$19.99

Entity Set to Relation



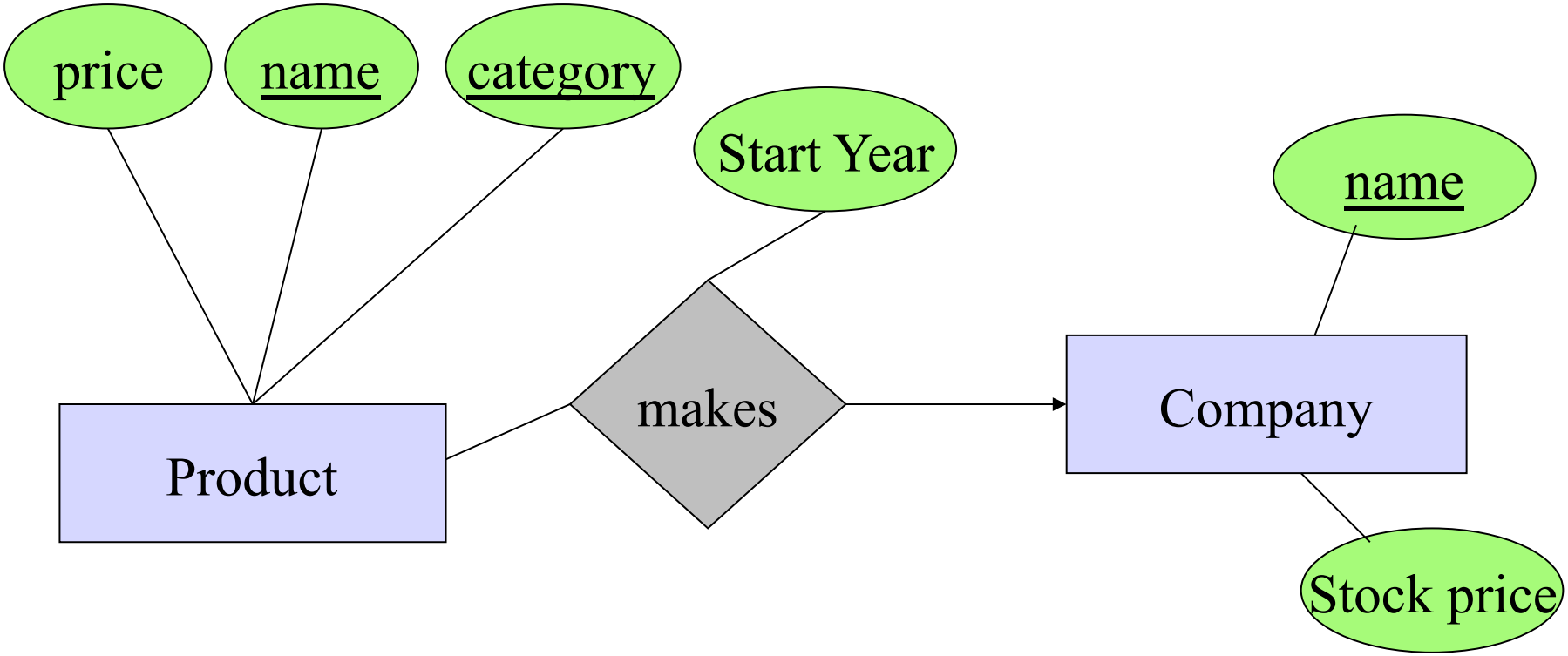
• CREATE TABLE PRODUCT(name, category, price,
PRIMARY KEY (name, category))

Q: Relationship → Relation?



How to capture "Make" in a relation?

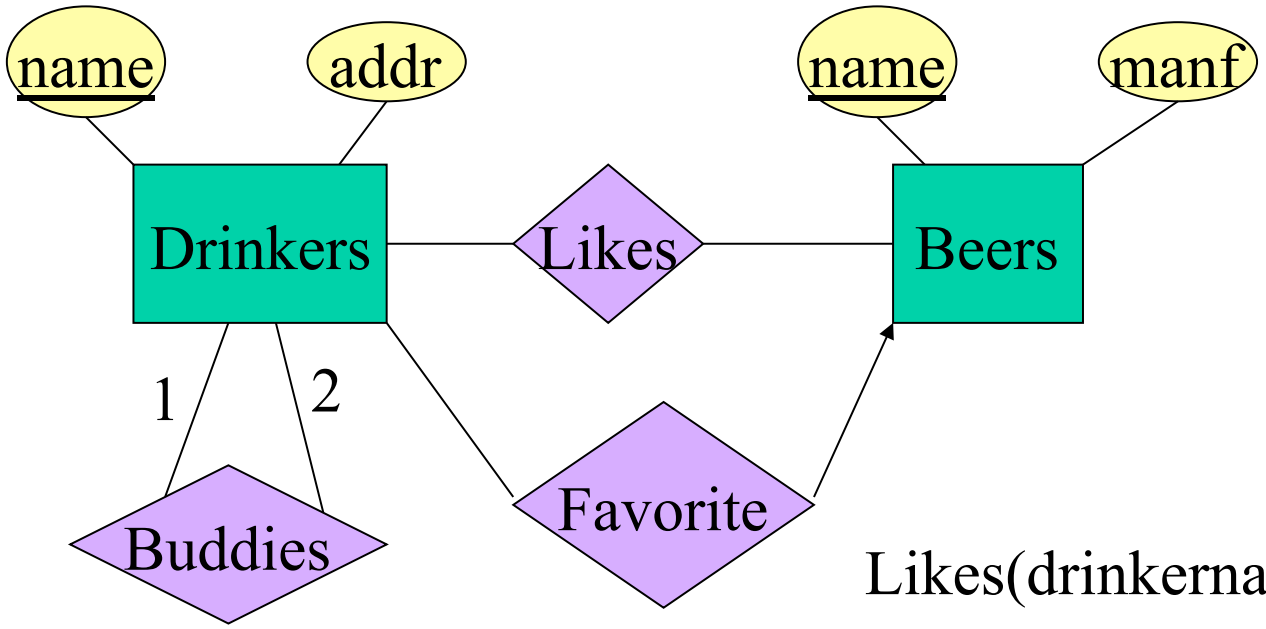
Q: Relationship → Relation?



How to capture "Make" in a relation?

- Key attributes of each Entity Set become attributes of the Relation
- Attributes of the relationship also become attributes of the Relation
- Makes(pname, pcategory, cname, start_year)

Relationship to Relation: Another Example



Likes(drinkername, beername)

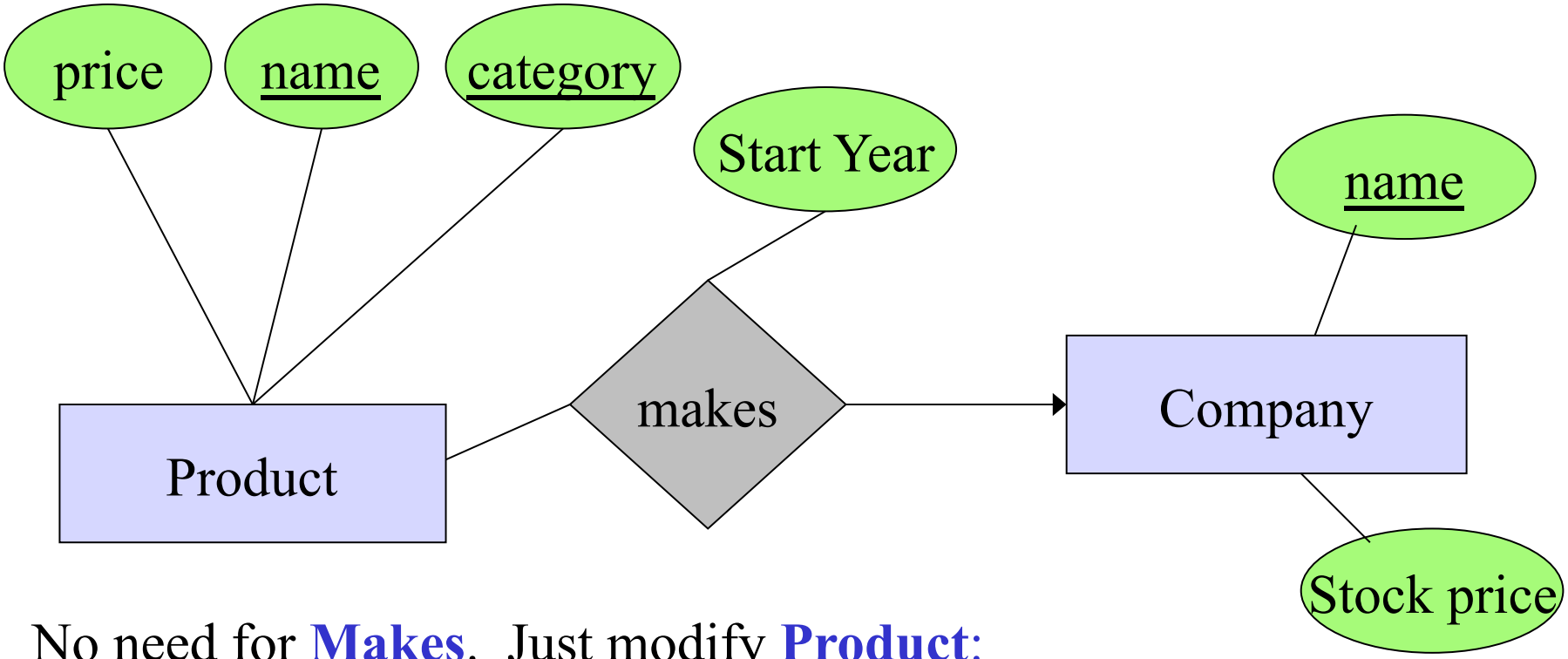
Favorite(drinkername, beername)

Buddies(name1, name2)

Special Cases of translating ER Diagram to Rel. Design:

1) many one relations
2) weak entity sets
3) “isa” cases

Combining Two Relations



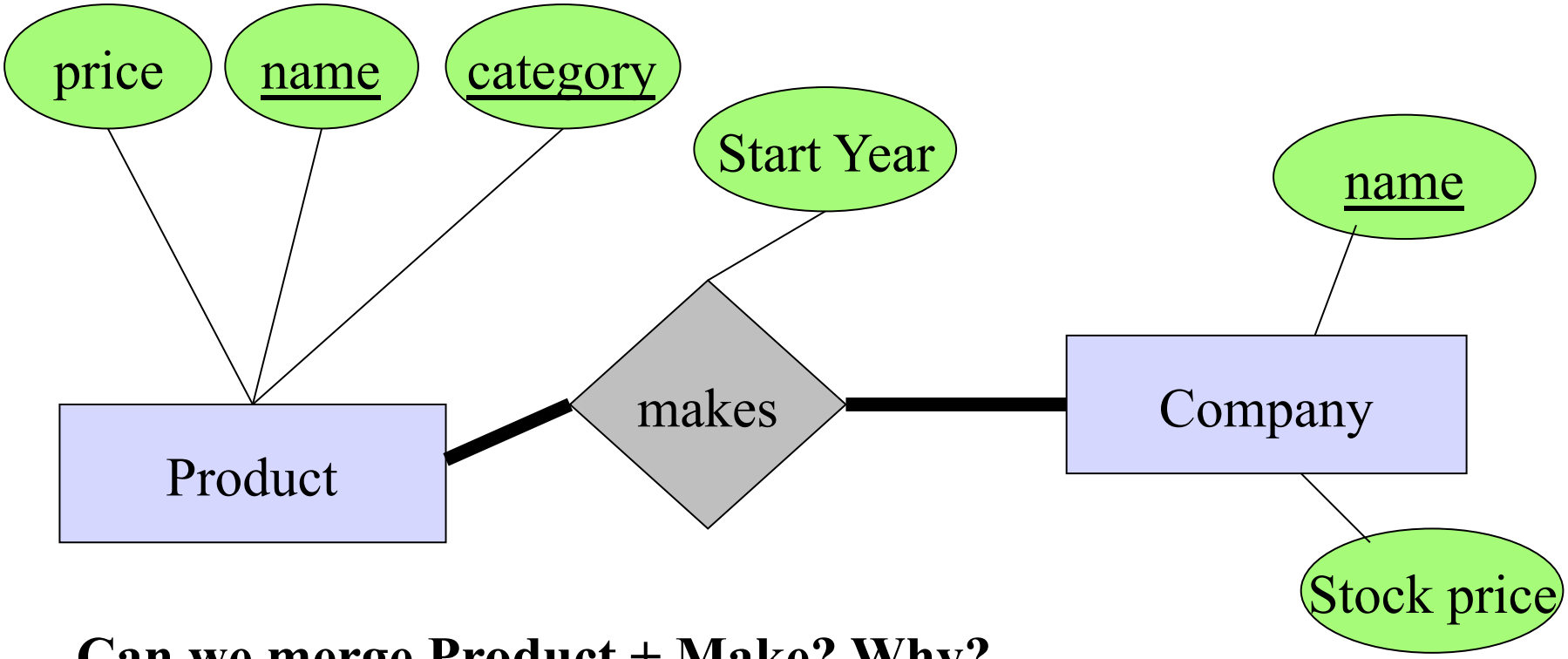
No need for **Makes**. Just modify **Product**:

name	category	price	StartYear	companyName
gizmo	gadgets	19.99	1963	gizmoWorks

Combining Relations

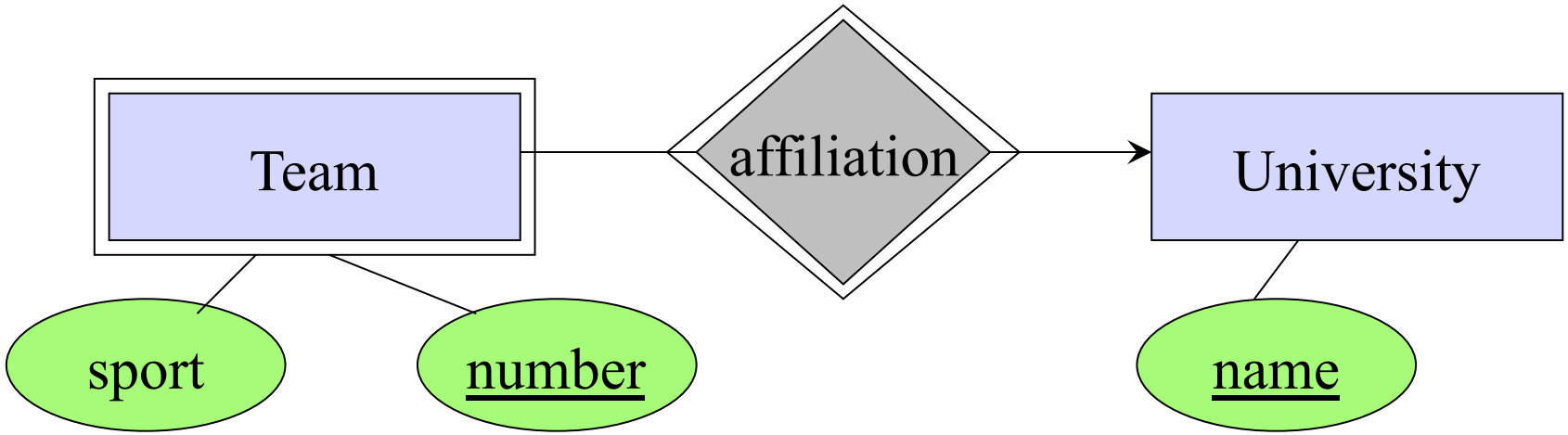
- It is OK to combine the relation for an entity-set *E* with the relation *R* for a many-one relationship from *E* to another entity set.

Q: What happen if Many-to-Many?

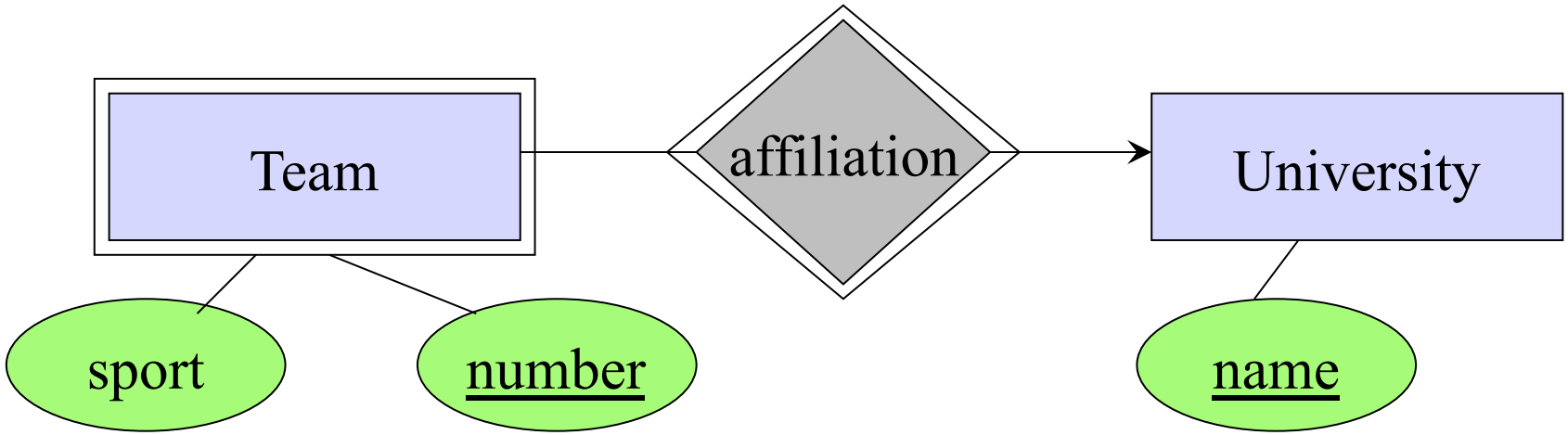


Can we merge Product + Make? Why?

Handling Weak Entity Sets



Handling Weak Entity Sets



Relation Team:

Sport	Number	Affiliated University
mud wrestling	15	Montezuma State U.

- **need all the attributes that contribute to the key of Team**
- don't need a separate relation for Affiliation. (why ?)

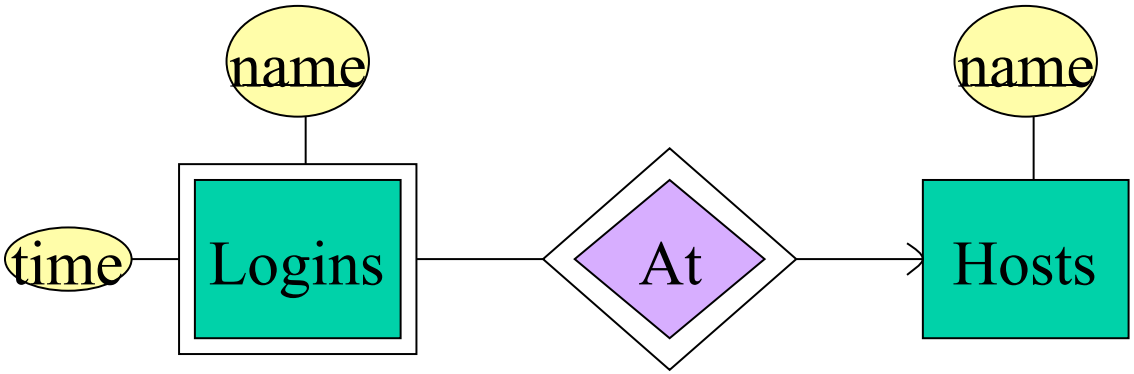
Handling Weak Entity Sets

- 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.

- Relation for a relationship in which a weak entiry set appears must use include attributes for the complete key of that entity set

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

Another Example



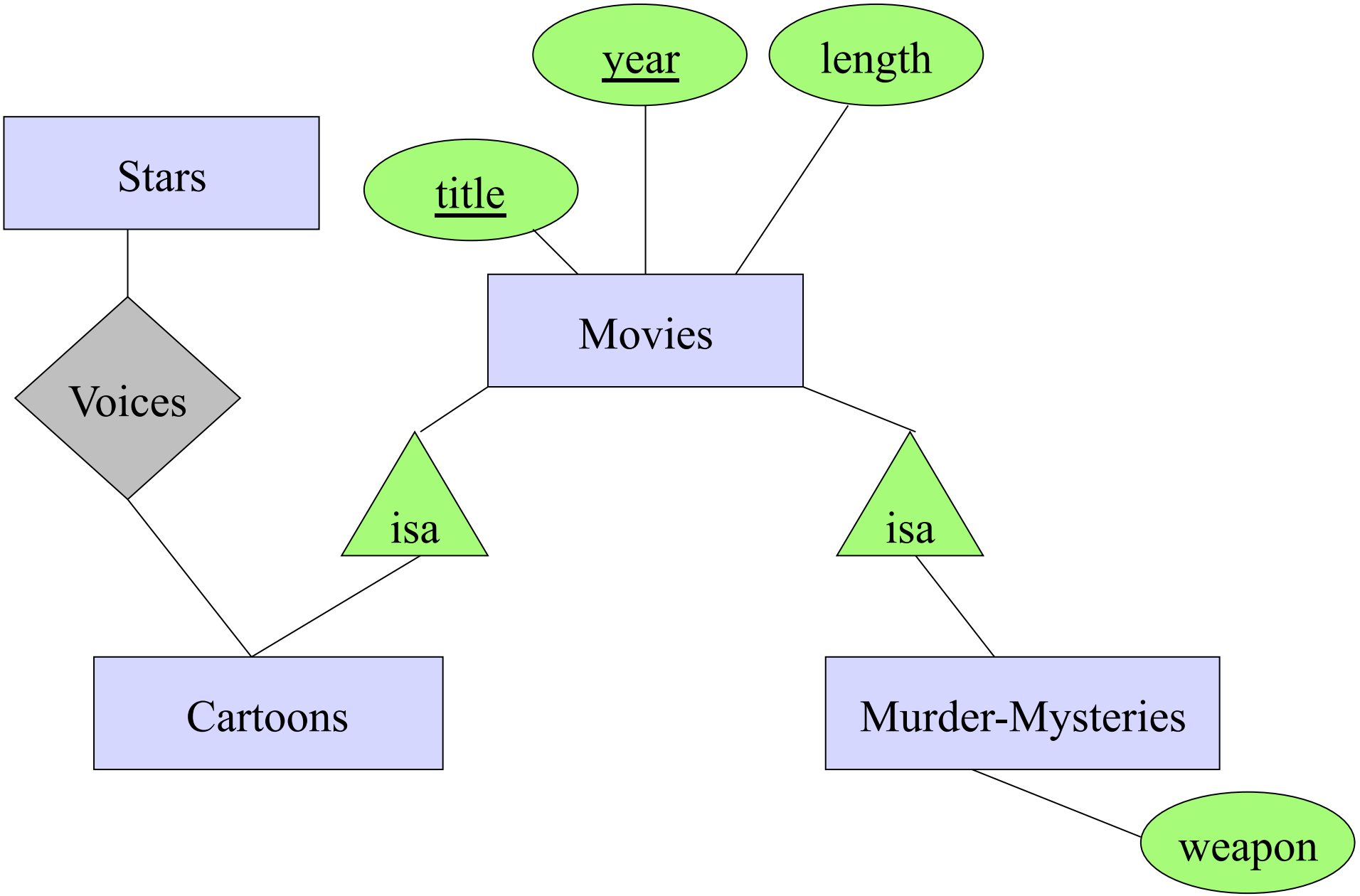
Hosts(hostName)

Logins(loginName, hostName, time)

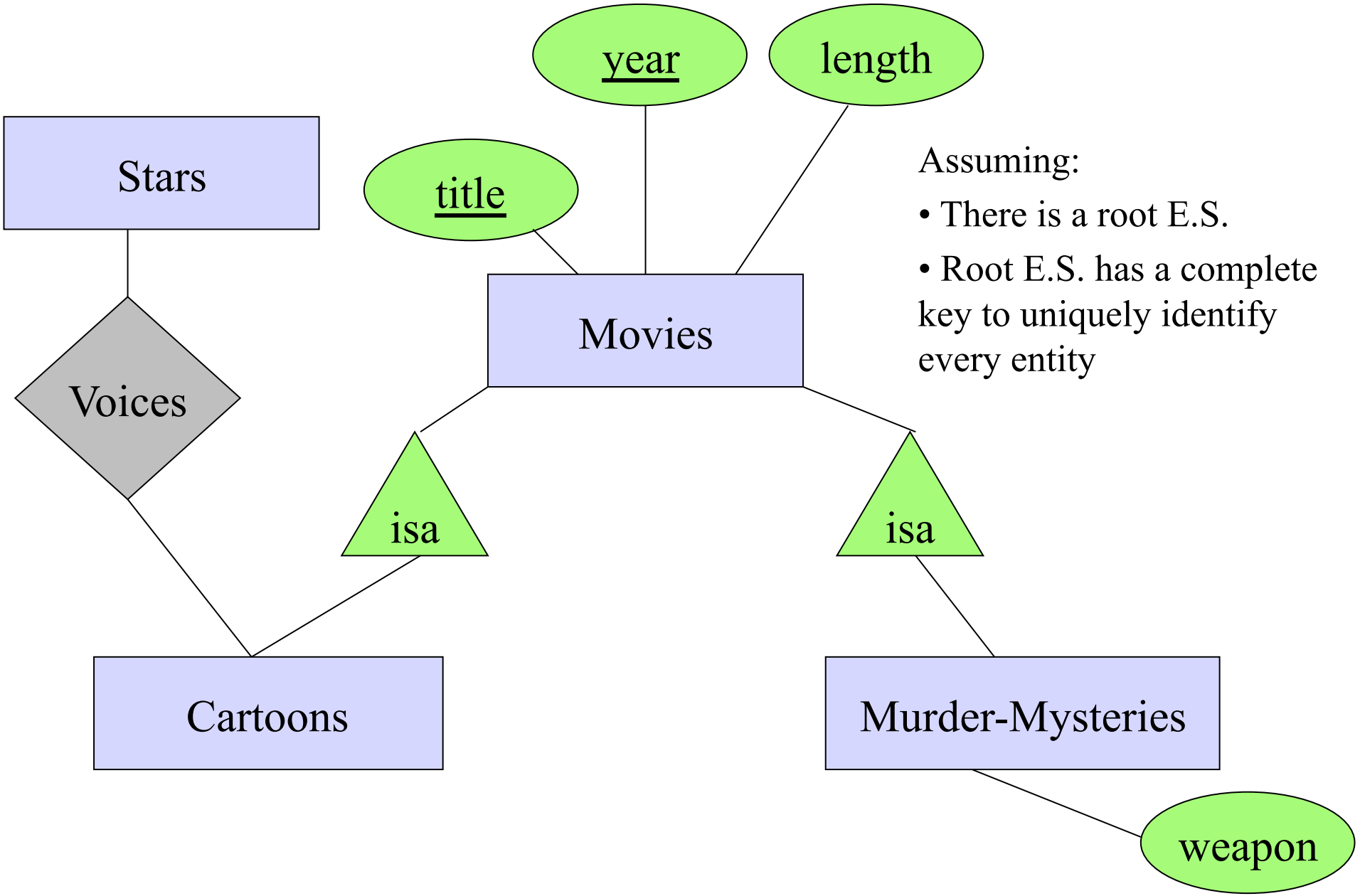
~~At(loginName, hostName)~~

At becomes part of Logins

Translating Subclass Entity Sets



Translating Subclass Entity Sets



Assuming:

- There is a root E.S.
- Root E.S. has a complete key to uniquely identify every entity

Option #1: the E/R Approach

• For each entity set *E*, create a relation with the key attributes from the root e.s. and any attributes of *E* itself.

Movies(title, year, length)

Cartoons(title, year)

MurderMysteries(title, year, weapon)

Voices(title, year, star-name)

Option #1: the E/R Approach

• For each entity set *E*, create a relation with the key attributes from the root e.s. and any attributes of *E* itself.

Movies(title, year, length)

Every movie has a tuple here

Cartoons(title, year)

MurderMysteries(title, year, weapon)

Voices(title, year, star-name)

Option #1: the E/R Approach

• For each entity set *E*, create a relation with the key attributes from the root e.s. and any attributes of *E* itself.

Movies(title, year, length)

Every movie has a tuple here

Cartoons(title, year)

Cartoon movies have a tuple here and in "Movies"

MurderMysteries(title, year, weapon)

Voices(title, year, star-name)

Option #1: the E/R Approach

• For each entity set *E*, create a relation with the key attributes from the root e.s. and any attributes of *E* itself.

Movies(title, year, length)

Every movie has a tuple here

Cartoons(title, year)

Cartoon movies have a tuple here and in "Movies"

MurderMysteries(title, year, weapon)

M.M. movies have a tuple here and in "Movies"

Voices(title, year, star-name)

Option #1: the E/R Approach

• For each entity set *E*, create a relation with the key attributes from the root e.s. and any attributes of *E* itself.

Movies(title, year, length)

Every movie has a tuple here

Cartoons(title, year)

Cartoon movies have a tuple here and in "Movies"

MurderMysteries(title, year, weapon)

M.M. movies have a tuple here and in "Movies"

Voices(title, year, star-name)

For every cartoon movie that has at least one voice star, there is at least one tuple here with the title and year of that movie.

How is “Roger Rabbit” stored?

Option #2: the OO Approach

Basic idea: each object can only belong to a single table.

Each entity can be either a

- Movie but neither cartoon nor murder mystery
- Movie and cartoon but not murder mystery
- Movie and murder mystery but not cartoon
- Movie and cartoon and murder mystery

We've thus enumerated all possible subtrees of the hierarchy.

Option #2: the OO Approach

Basic idea: Enumerate all possible subtrees of the hierarchy. Create a relation (table) for each, with appropriate attributes:

Movies
Movies-C
Movies-MM
Movies-C-MM

Option #2: the OO Approach

Basic idea: Enumerate all possible subtrees of the hierarchy. Create a relation (table) for each, with appropriate attributes:

Movies(title, year, length)
Movies-C(title, year, length)
Movies-MM(title, year, length, weapon)
Movies-C-MM(title, year, length, weapon)

Option #2: the OO Approach

Basic idea: Enumerate all possible subtrees of the hierarchy. Create a relation (table) for each, with appropriate attributes:

Movies(title, year, length)
Movies-C(title, year, length)
Movies-MM(title, year, length, weapon)
Movies-C-MM(title, year, length, weapon)

Each movie will have a tuple in only one of these four relations. (Unlike the E/R approach.)

Option #2: the OO Approach

Basic idea: Enumerate all possible subtrees of the hierarchy. Create a relation (table) for each, with appropriate attributes:

Movies(title, year, length)
Movies-C(title, year, length)
Movies-MM(title, year, length, weapon)
Movies-C-MM(title, year, length, weapon)

What about “Voices”?

Option #2: the OO Approach

Basic idea: Enumerate all possible subtrees of the hierarchy. Create a relation (table) for each, with appropriate attributes:

Movies(title, year, length)
Movies-C(title, year, length)
Movies-MM(title, year, length, weapon)
Movies-C-MM(title, year, length, weapon)

What about "Voices"?

Voices(title, year, star-name)

(i.e., key attributes of the entity sets connected by the relationship)

Option #3: The Null Value Approach

Have one table:

Movie(title, year, length, weapon)

Some values in the table will be NULL, meaning that the attribute not make sense for the specific movie.

Option #3: The Null Value Approach

Have one table:

Movie(title, year, length, weapon)

Some values in the table will be NULL, meaning that the attribute not make sense for the specific movie.

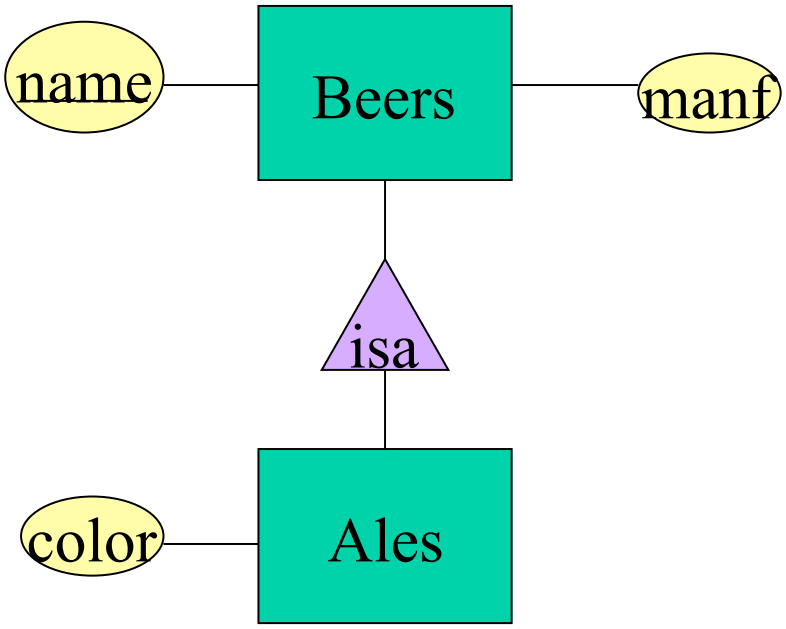
Too many meanings for NULL ("don't know the value" versus "attribute doesn't make sense for this tuple").

Translating Subclass Entities: The Rules

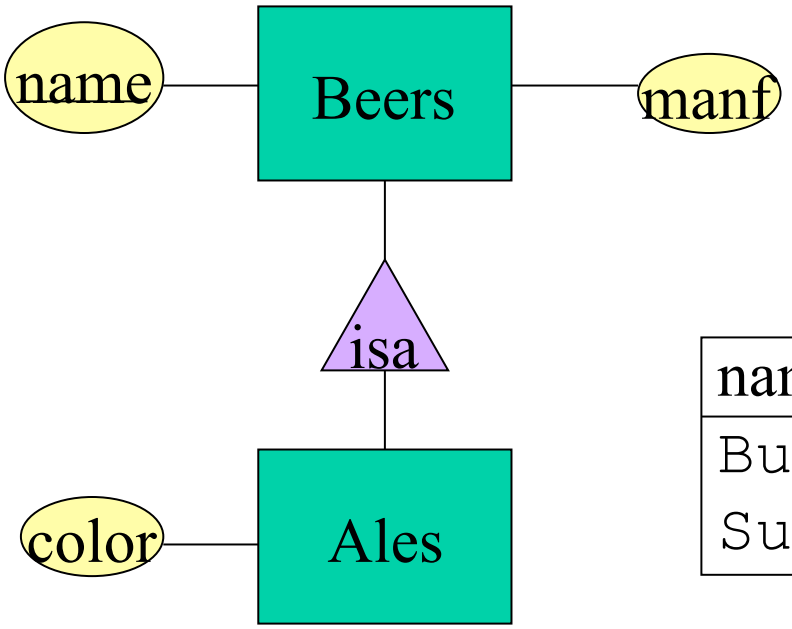
Three approaches:

1. *E/R style* : create one relation for each subclass, with only the key attribute(s) and attributes attached to that E.S.; entity has a tuple in all appropriate relations.
2. *Object-oriented* : create a relation for each possible subtree of the hierarchy, with all its attributes; each entity belongs to exactly one relation;
3. *Use nulls* : create one relation; entities have null in attributes that don't belong to them.

Example



E/R Style



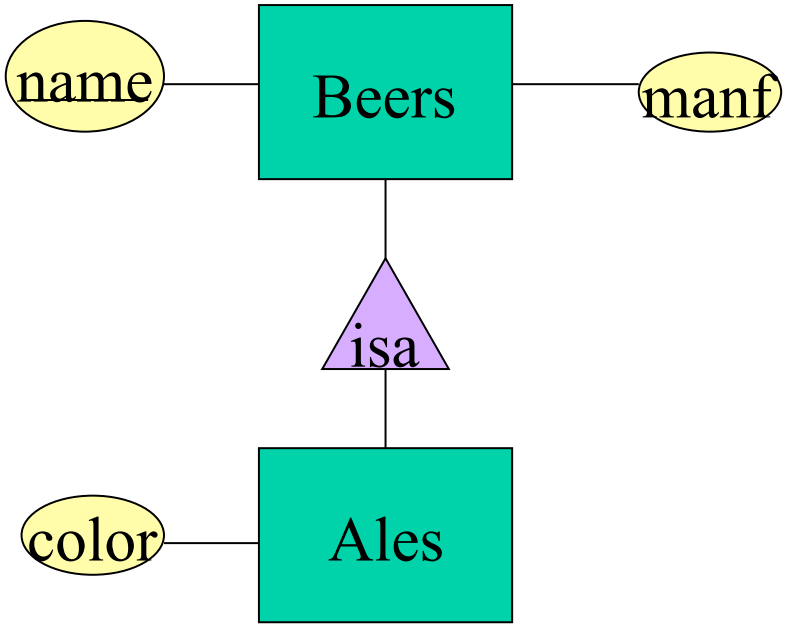
name	manf
Bud	Busch
Summerbrew	Pete's

Beers

name	color
Summerbrew	dark

Ales

Object Oriented



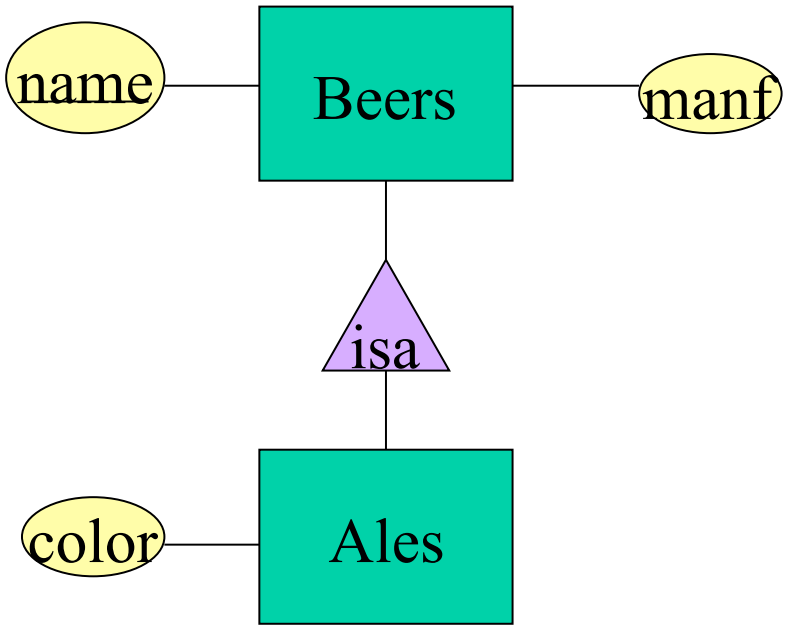
name	manf
Bud	Busch

Beers

name	manf	color
Summerbrew	Pete's	dark

Ales

Using Nulls



name	manf	color
Bud	Busch	NULL
Summerbrew	Pete's	dark

Beers

Q: What would work faster?

- Query 1: Find the colors of ales made by some company, say, Pete's.

- Query 2: Find all beers made by Pete's.

Comparisons

- O-O approach good for queries like "find the color of ales made by Pete's."
  - Just look in Ales relation.
- E/R approach good for queries like "find all beers (including ales) made by Pete's."
  - Just look in Beers relation.
- Using nulls saves space unless there are *lots* of attributes that are usually null.

Comparisons

- In answering a query, we prefer to have all necessary attributes in one place (table).
  - The Nulls approach has the advantage here.
  - Between E/R and OO, it's a mixed verdict, as we saw.

- We prefer not to use too many relations.
  - The Nulls method is best, followed by E/R, then by OO.
  - Null: 1. E/R: n+1. OO: 2^n.

- We want to minimize space consumption.
  - OO is best.
  - Null may be bad if too many NULLs needed.