Recap

- Data-savviness is the future!
- Notion of a DBMS
- The relational data model and algebra: bags and sets
- SQL Queries
- SQL Modifications
- SQL DDL
- Next: Database Design

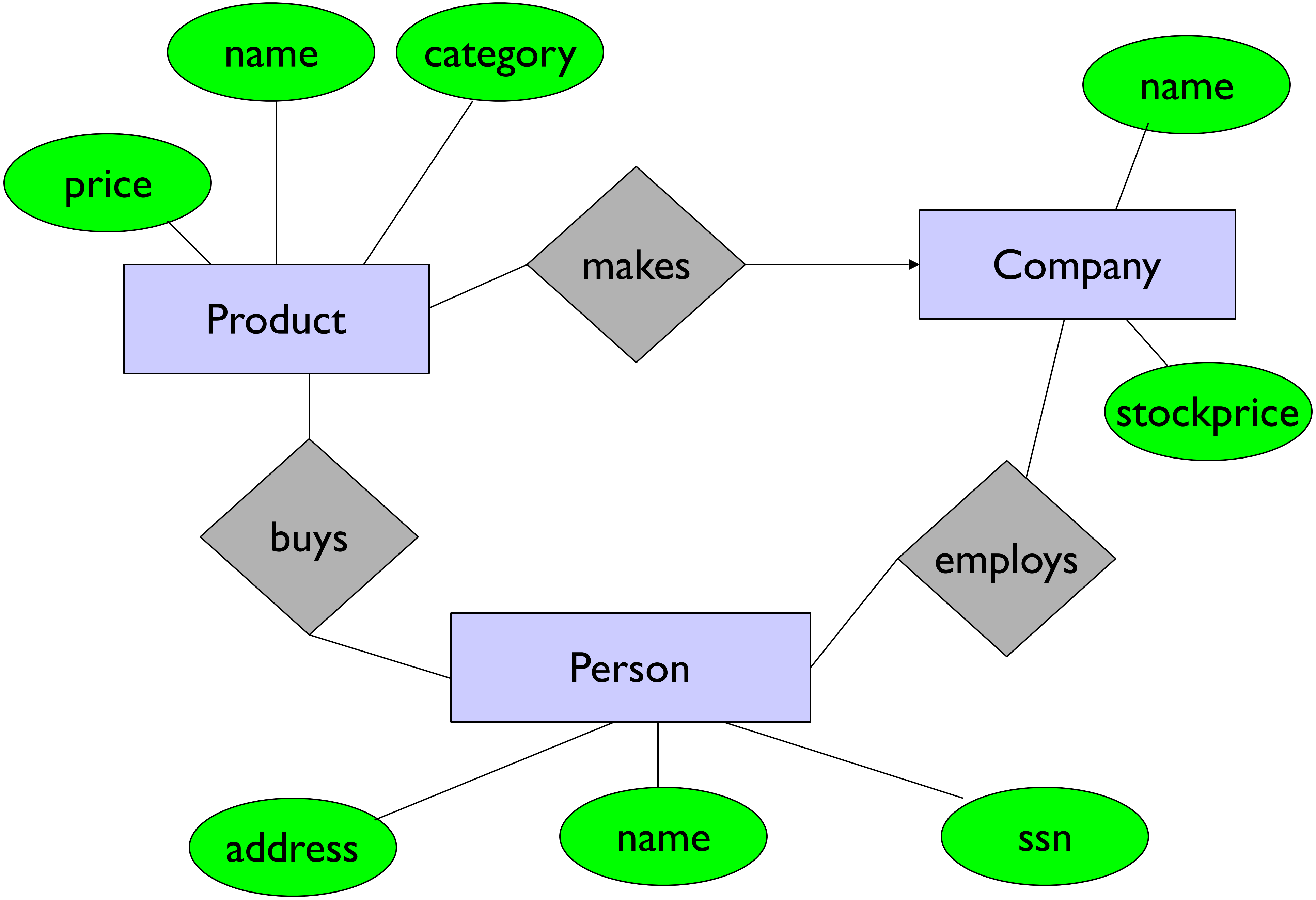
So far…

- We've seen the DDL syntax of SQL to define relations, alter them, and then drop them
- But we haven't figured out which relations to use in the first place!

- That is the focus of this set of slides

To Decide on the Relational Schema…

- You need to faithfully capture the requirements of the application
    - What information needs to be represented and how they relate with each other
- A step towards that is a conceptual model, or a diagram of the entities and relationships
    - This is the so-called ER diagram or Entity-Relationship diagram
- Essentially "doodles about data"

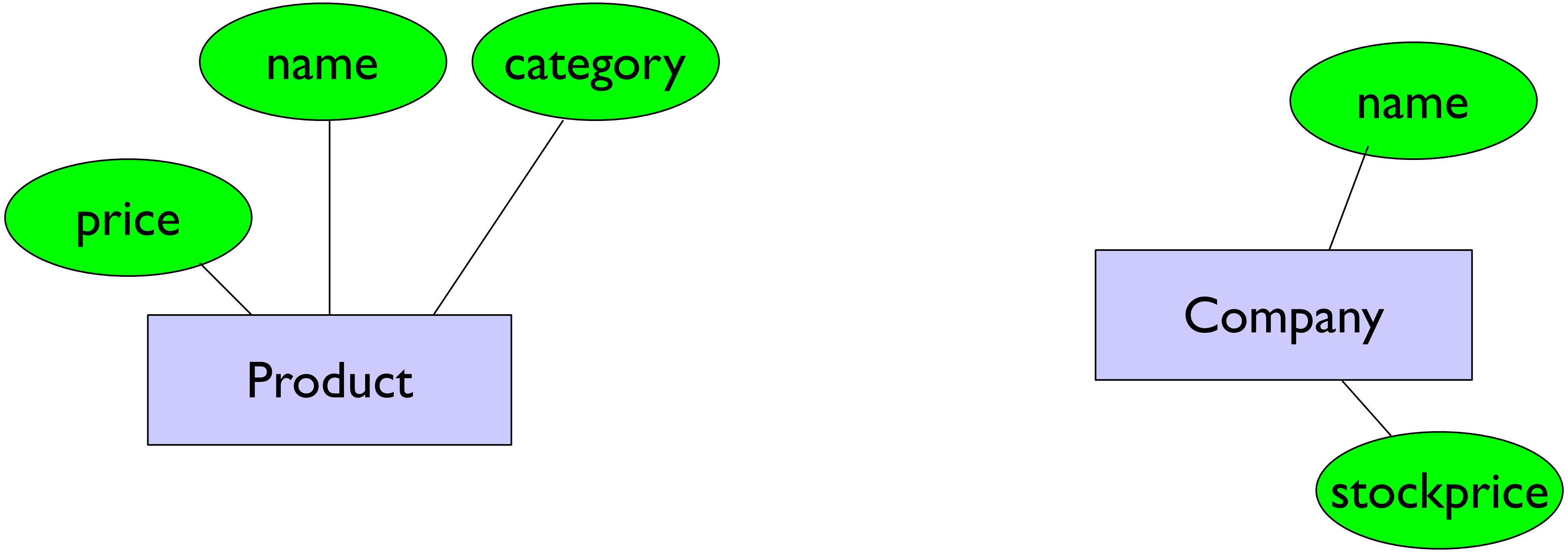

name
category
price
name
makes
Company
Product
stockprice
buys
employs
Person
address
name
ssn

The ER diagram

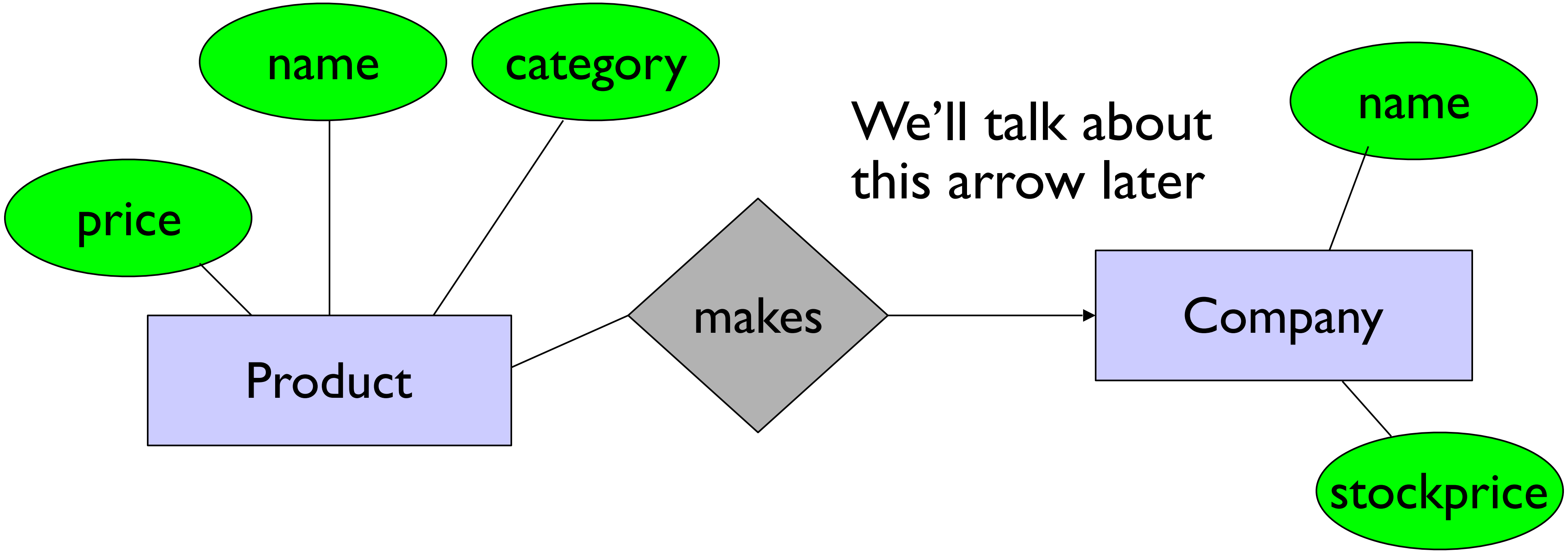
- Proposed by Peter Chen in 1976
- A very rich modeling language: we're going to study a tiny subset

- Two basic primitives in an ER diagram:
  - **Entities**: things, objects, …
    - Described using a set of attributes (all atomic)
    - A set of entities is called an **Entity set**
      - Represented via a rectangle
  - **Relationships**: connections between entities
    - Represented via diamonds

Entity Sets



Relationships

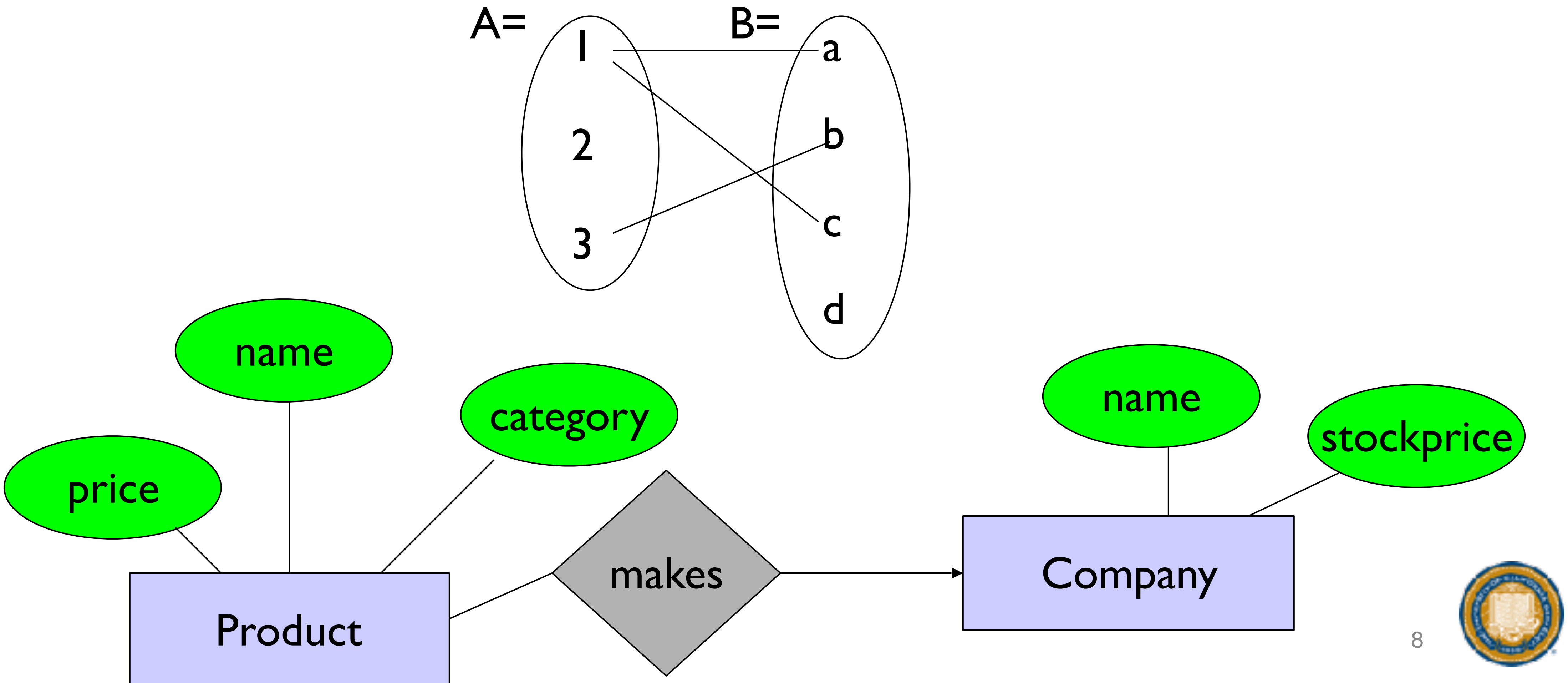


We'll talk about
this arrow later

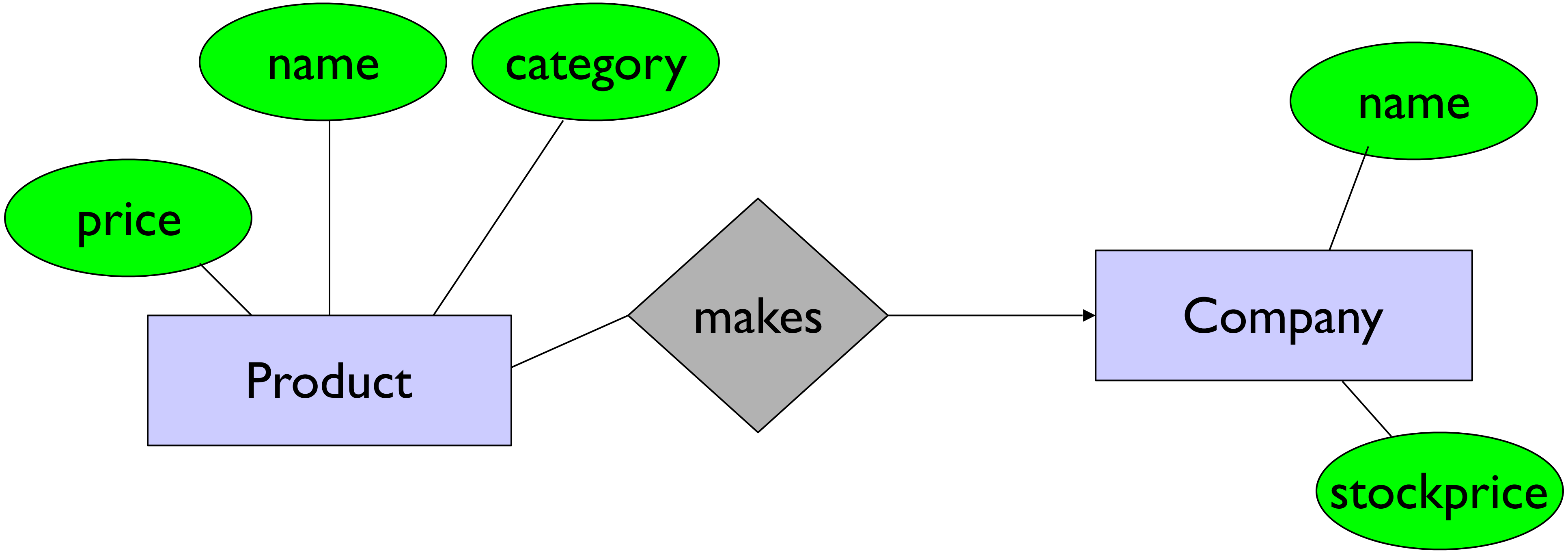
Goal: Connect Entity Sets

Relationships

- A and B are Entity Sets, a Relationship between A and B is a subset of A X B
- Makes is a subset of Product X Company



Back to the Diagram

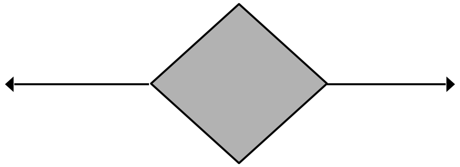


- Entity Sets: Rectangles
- Attributes: Ovals attached to rectangles or diamonds
- Relationships: Diamonds connecting rectangles

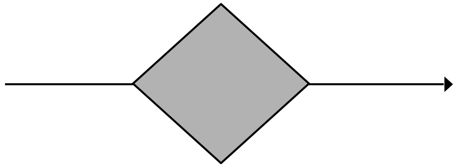
The Arrows in Relationships

- Arrow = **at most one**
- Another interpretation: *determines*

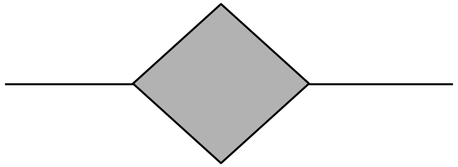
- One-one: one on LHS/RHS connected to at most one of RHS/LHS



- Many-one: one on LHS connected to at most one on RHS

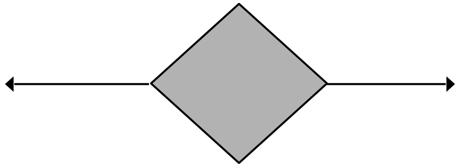


- Many-many: no constraints

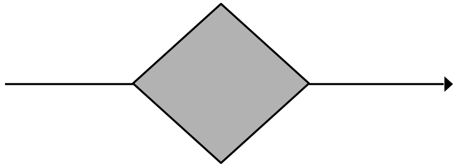


Example Scenarios for Each Case

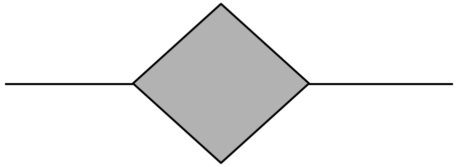
- one-one:



- many-one



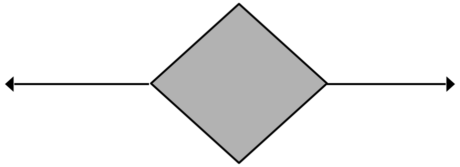
- many-many



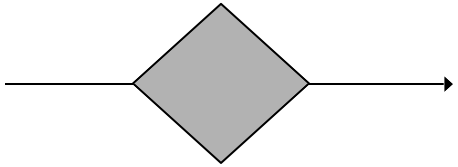
Department – head ??
Actor – play ??
Employee – company ??

Example Scenarios for Each Case

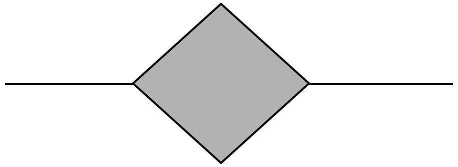
- one-one:



- many-one



- many-many



Department – head:

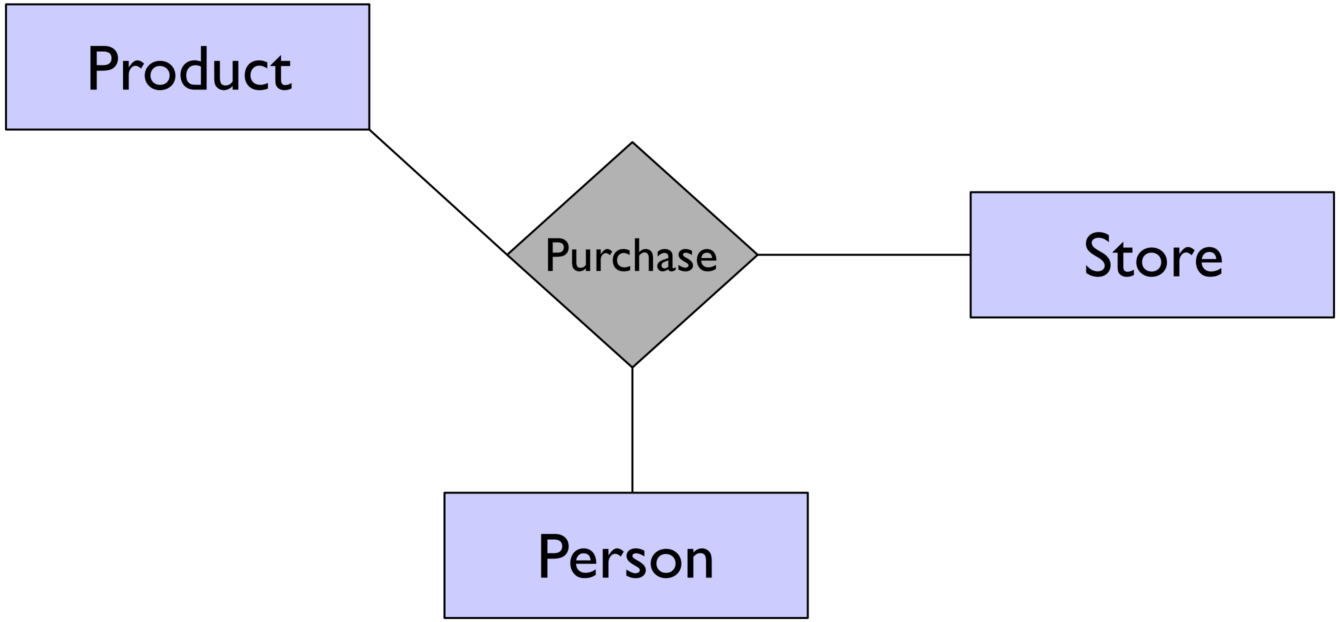
one – one. (each dep has 1 head, each head is head of 1 dep)

Actor – play: many – many. (each play has many actors, and vice versa)

Employee – company:

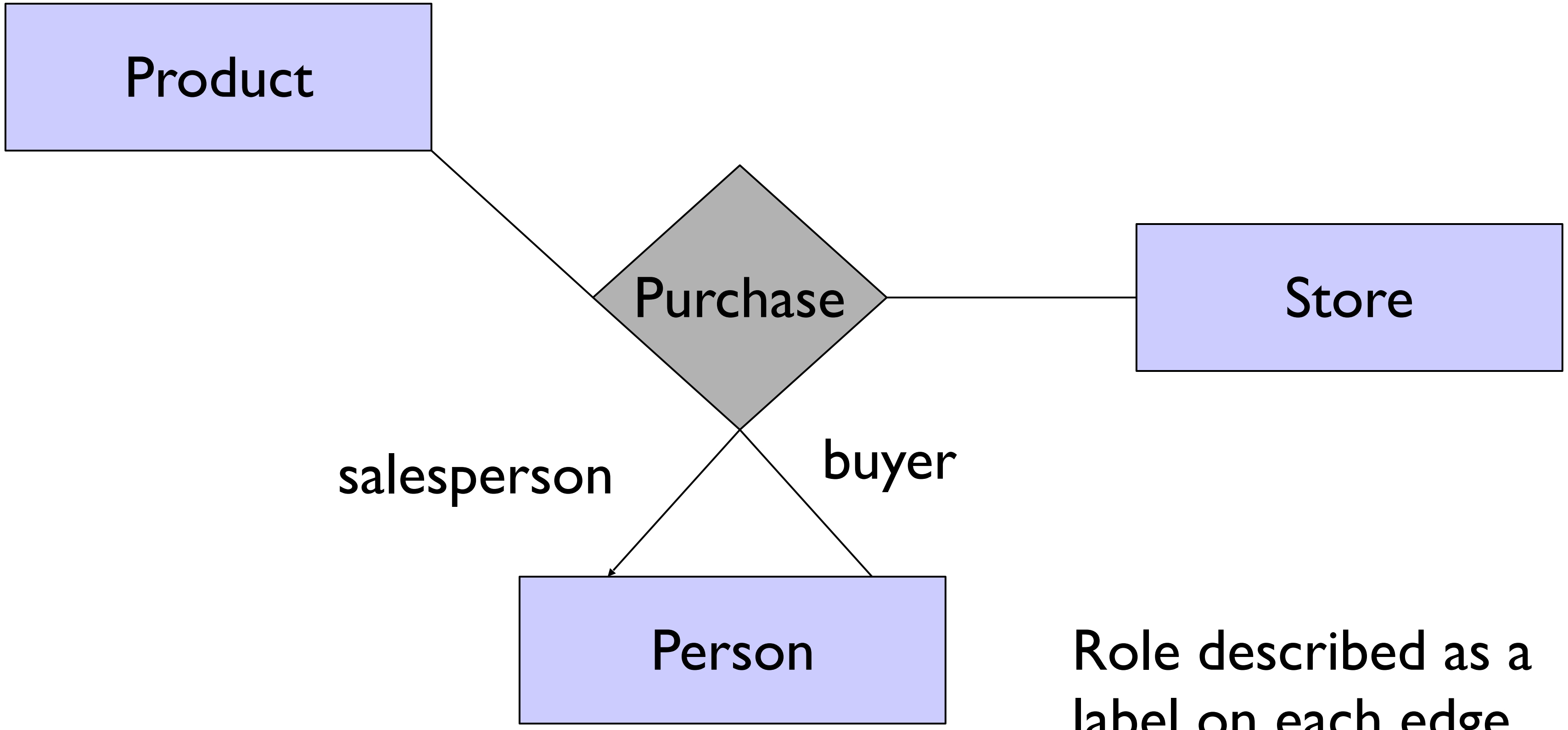
many – one. (each company has many employees, each employee works for one company)

Multi-way Relationships



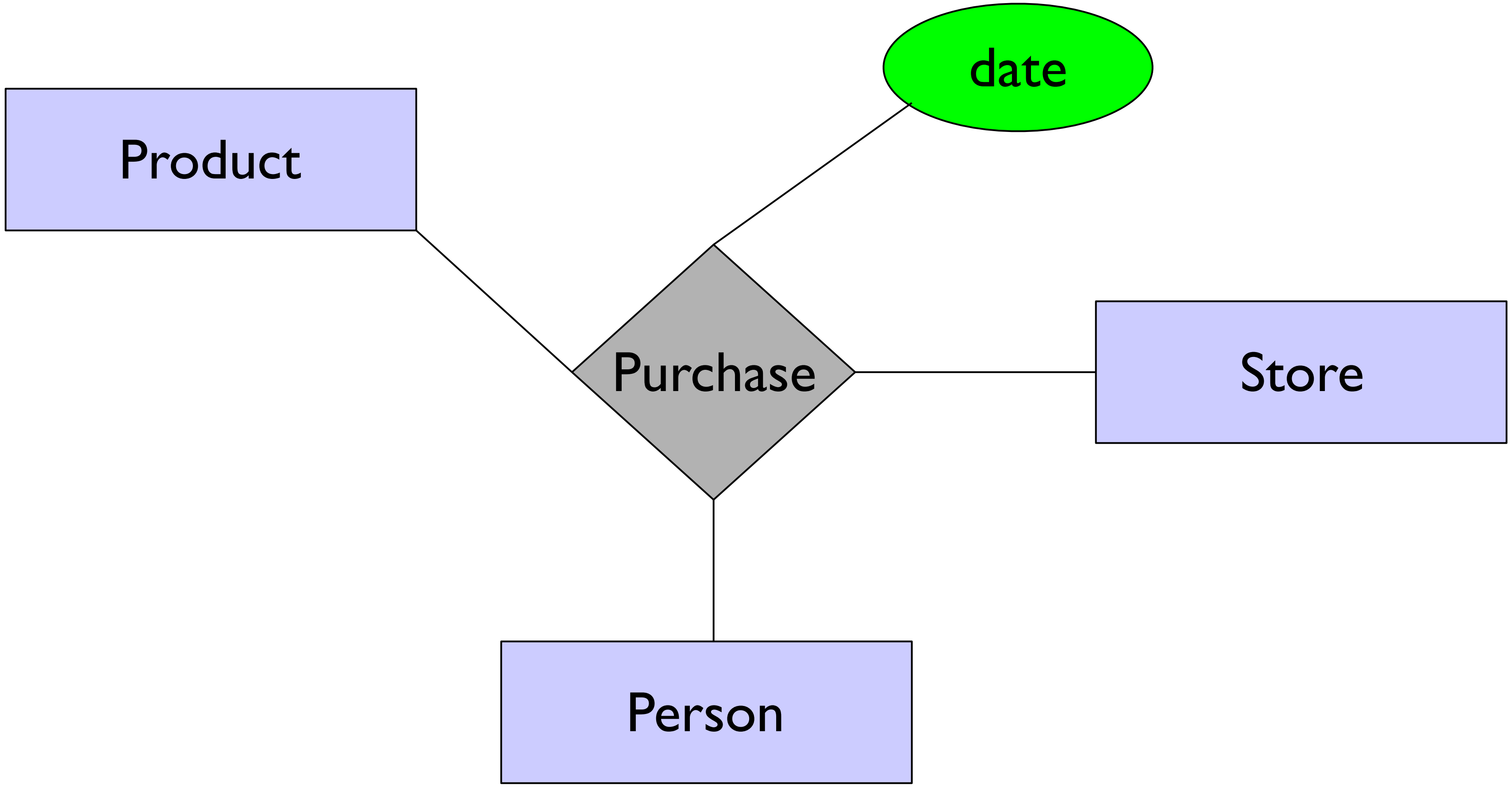
- Relationship between person, store, purchase
  - Purchase is a subset of Product X Person X Store
- Arrow still means *at most one:*
  - **combination of the other entities *determines* or is *connected to at most one entity* of inbound entity set**

Roles in Relationships



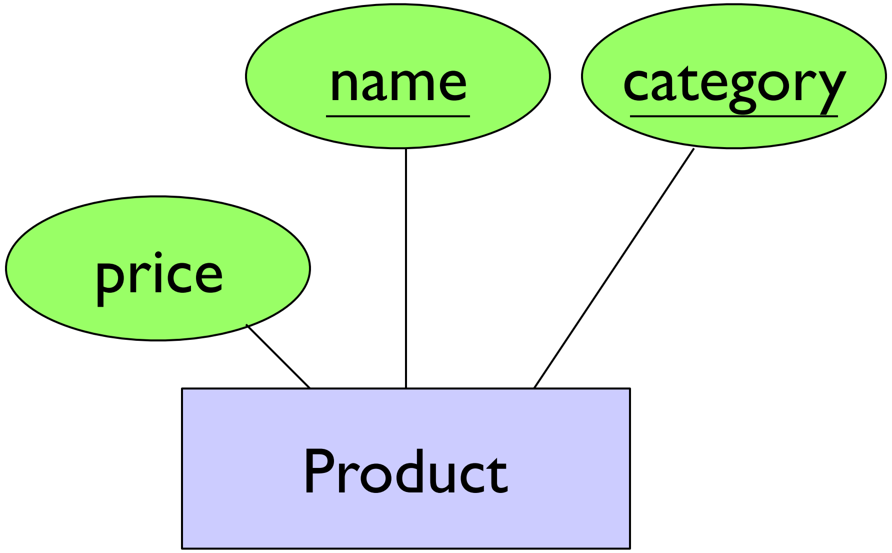
Role described as a label on each edge.

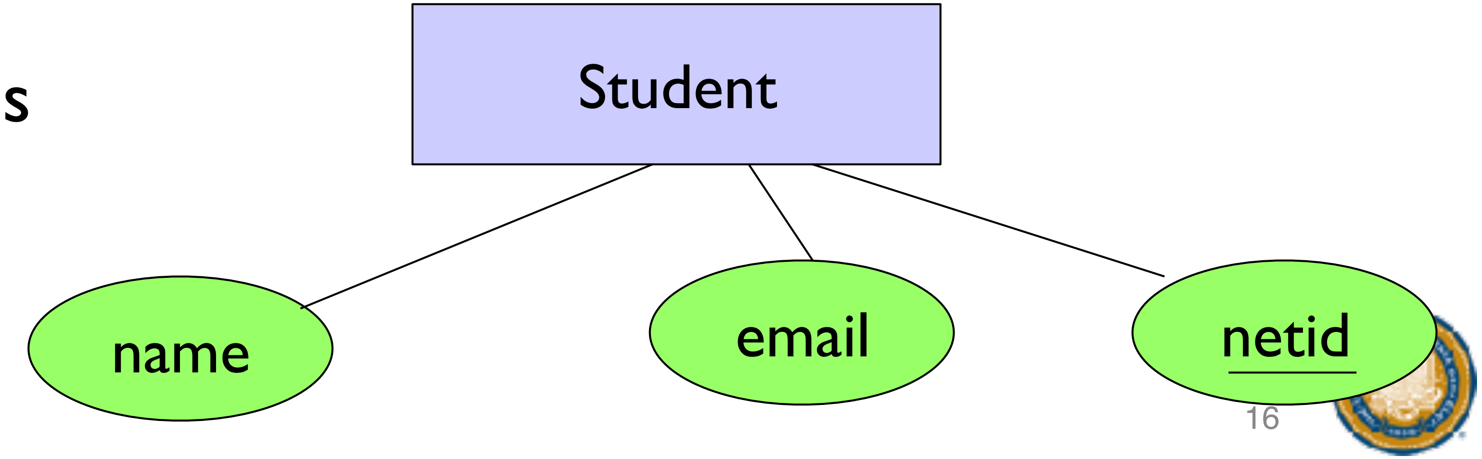
Attributes of Relationships



Remember Primary Keys?

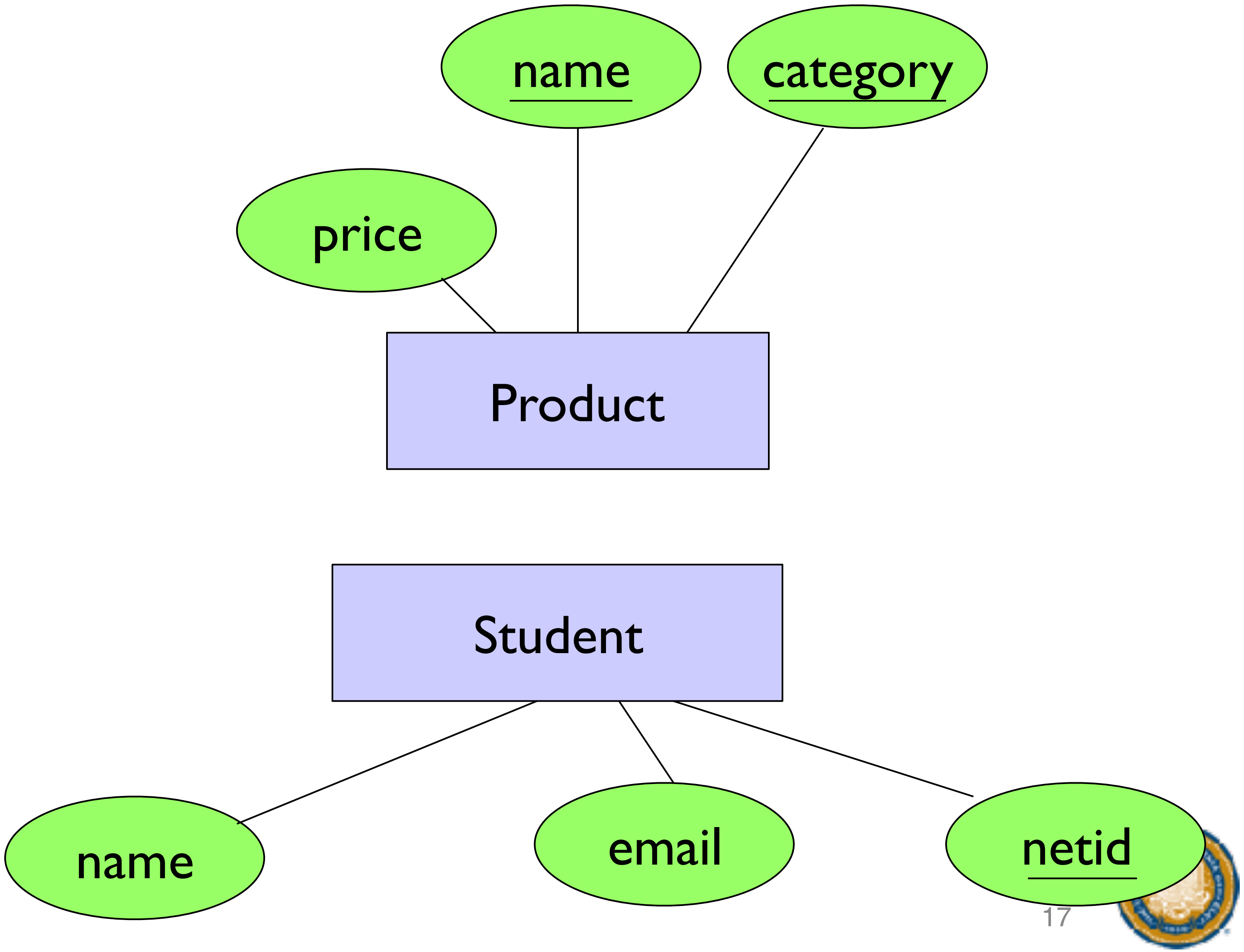
- A form of constraint on the data
- We can capture primary keys by underlining them
- Name and category together determine the Product entity
- No formal way to capture multiple keys in ER diagrams





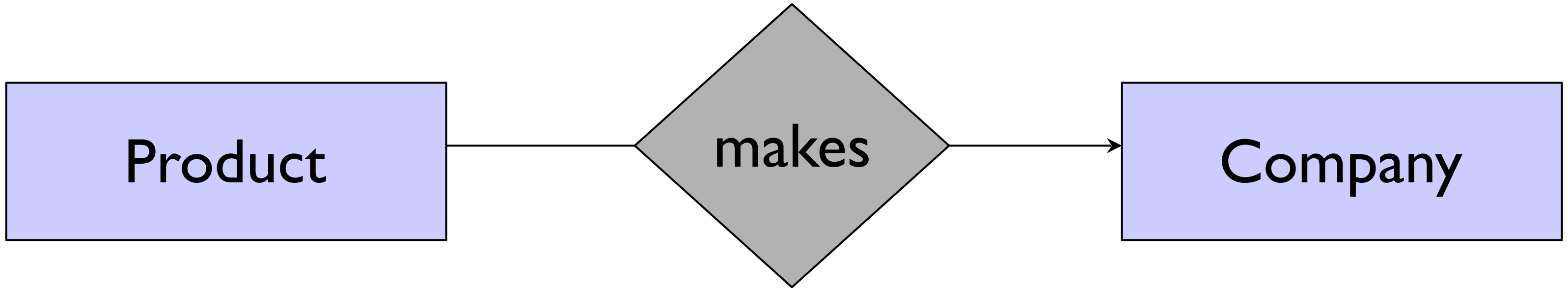
Remember Primary Keys?

- In ER diagrams, we require every entity set to have a primary key
- In the worst case it is the entire set of attributes!

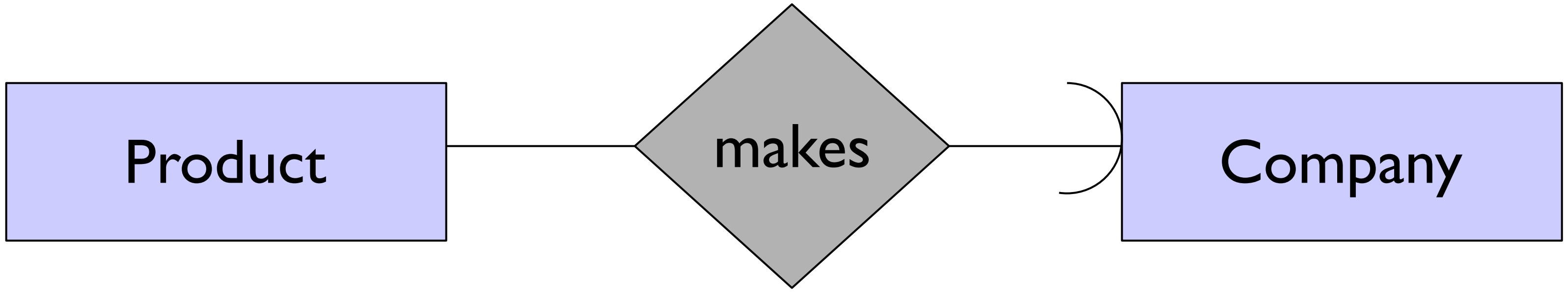


Another Constraint: Referential Integrity

- Recall: arrow meant "at most one"
- Each product made by (related to) at most one company

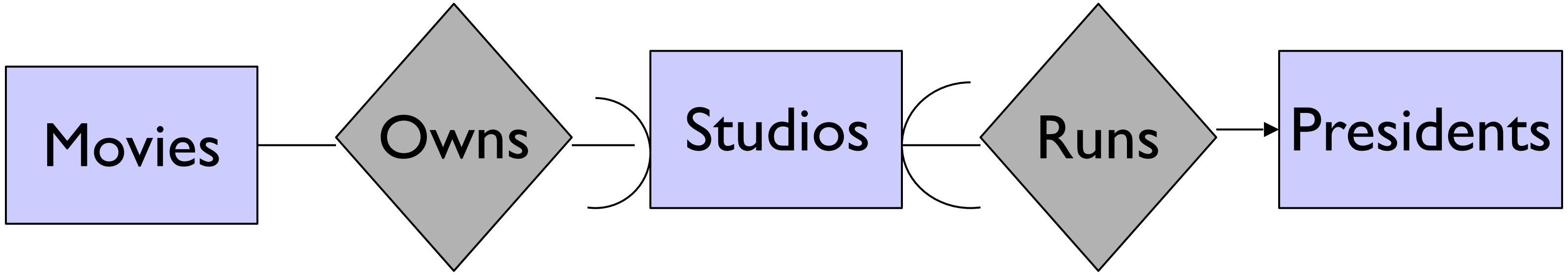


- But wouldn't it be weird if a product wasn't made by any company?



- Encode that with a semi-circle: **exactly one**, as opposed to arrow: **at most one**
- **"No dangling pointers"**

Another Example



What do these two semi-circles mean?

Each movie is owned by precisely one studio, and
Each president runs exactly one studio

Each studio has up to one president

Lots More!

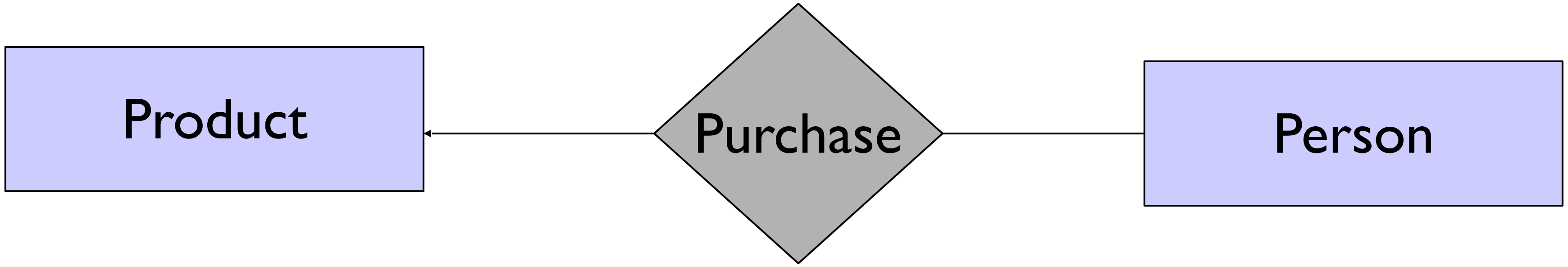
There's a lot more that ER diagrams can capture…

- Subclasses
- "Weak" entity sets
- Other constraints
- …

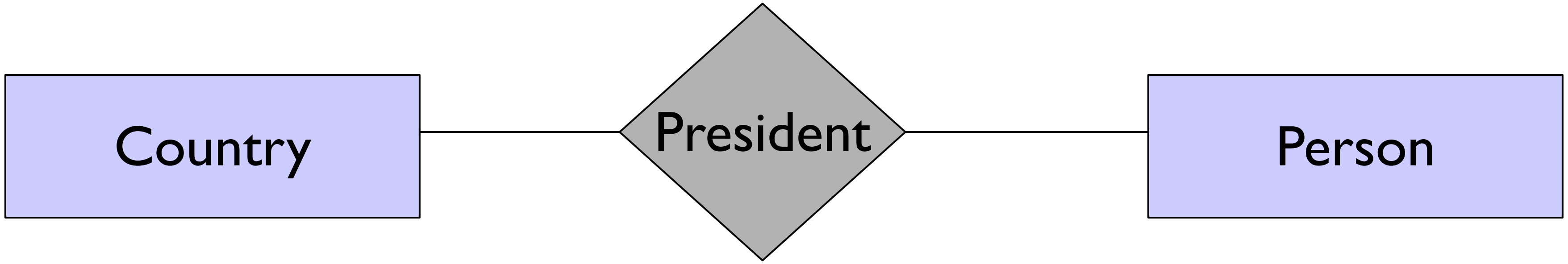
Design Principles

- Be faithful to reality
- Avoid redundancy
- Pick the right kind of element

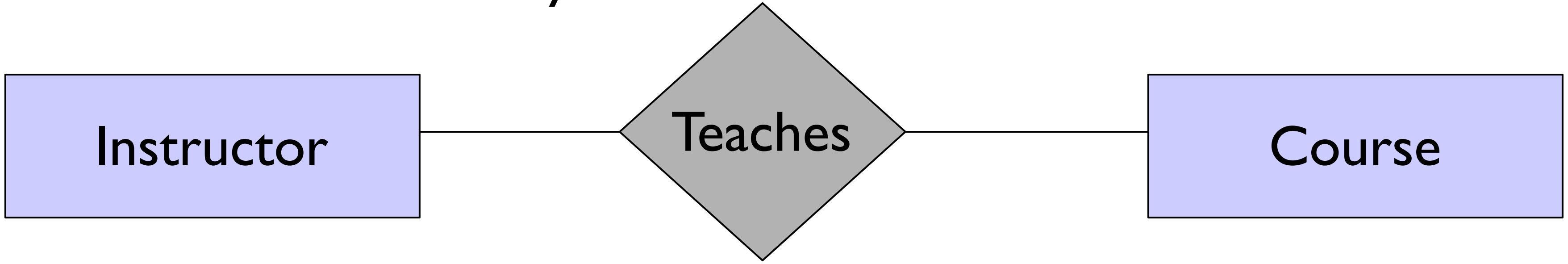
Being Faithful to Reality



No: A Person may purchase multiple Products



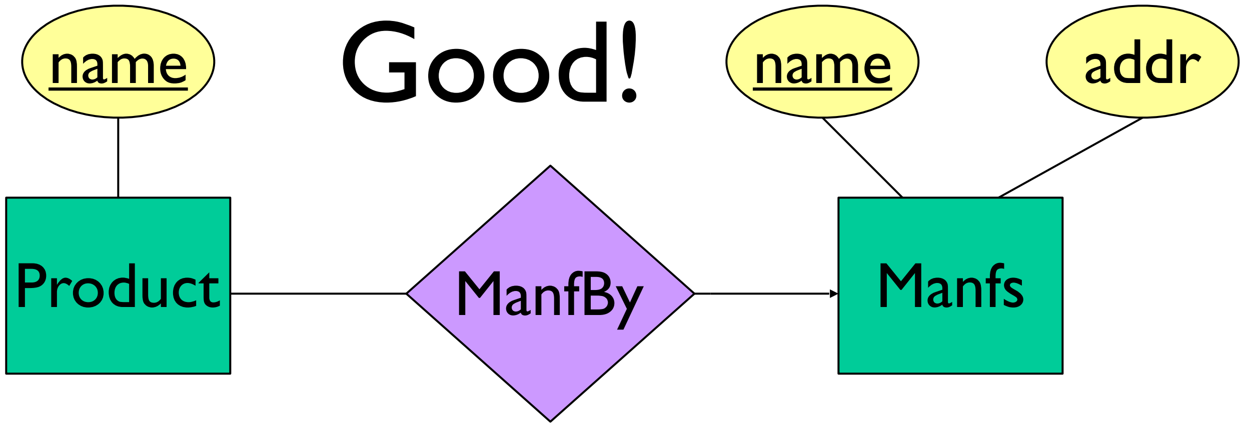
No: A Country can have at most one President



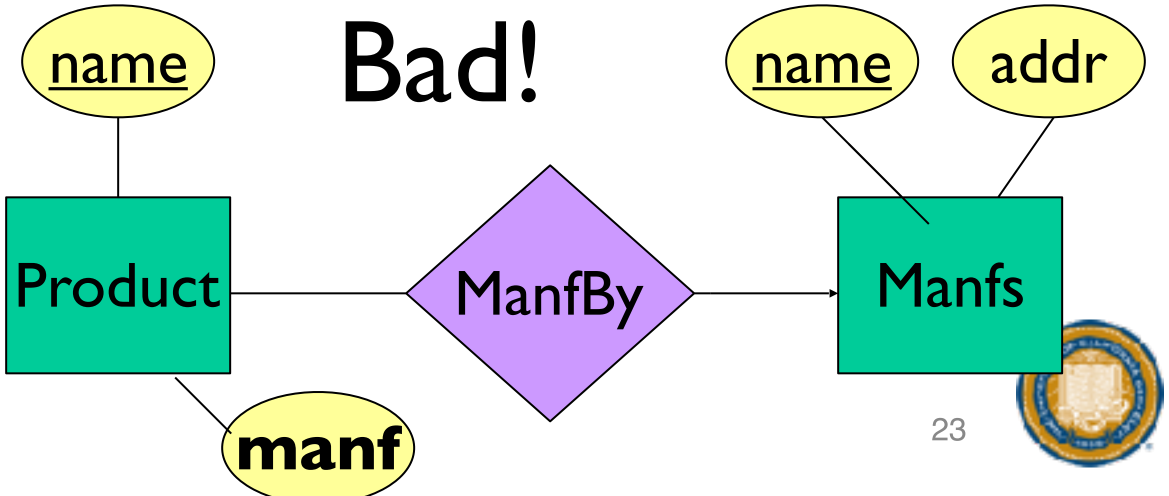
Yes if multiple instructors, No if not.

Avoiding Redundancy

- Redundancy: saying the same thing in multiple ways or places
- Redundancy
  - Wastes space
  - Allows inconsistencies to pop up (**update anomalies**)
    - If you edit one copy but not others



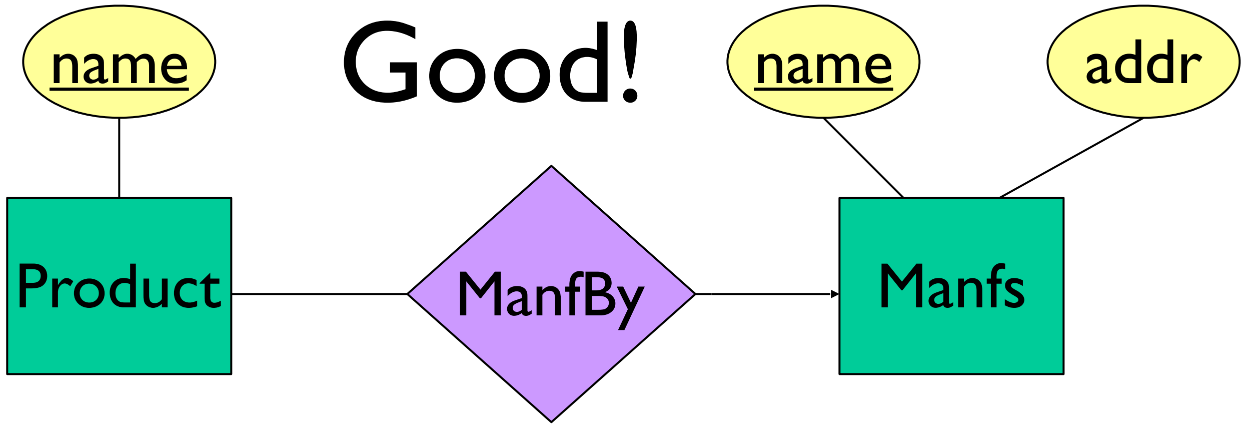
Good!

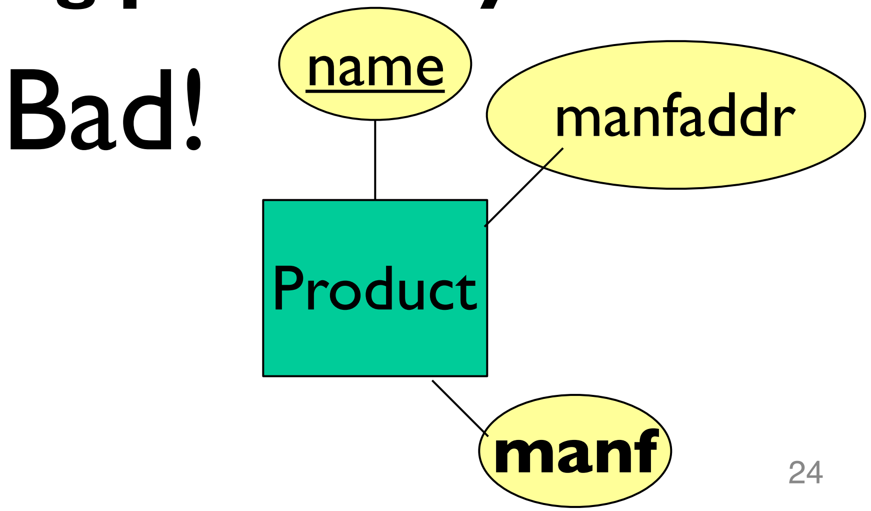


Bad!

Avoiding Redundancy

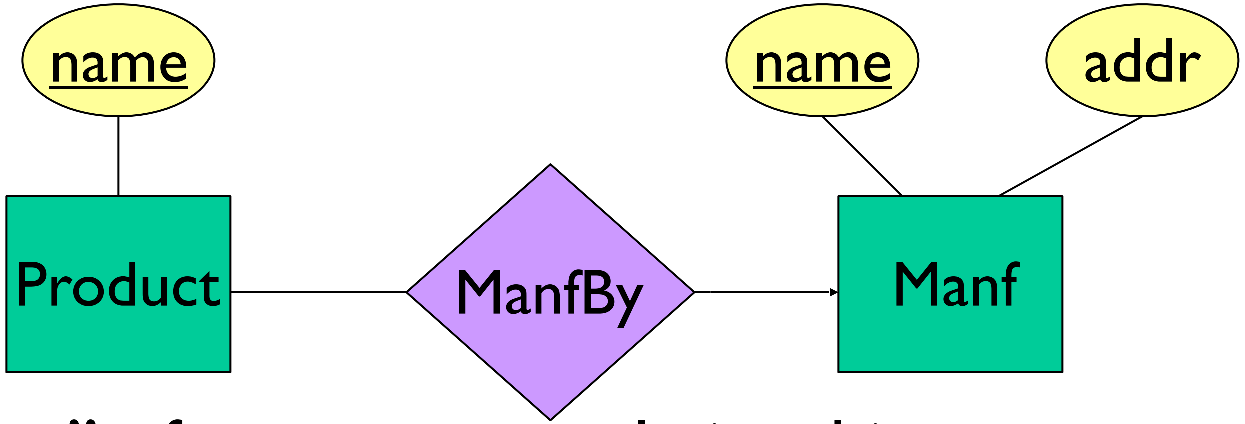
- Redundancy: saying the same thing in multiple ways or places
- Redundancy
  - Wastes space
  - Allows inconsistencies to pop up
  - **Another issue with this one: you lose addresses for manf who haven’t started making products yet**





Picking the Right Kind of Element

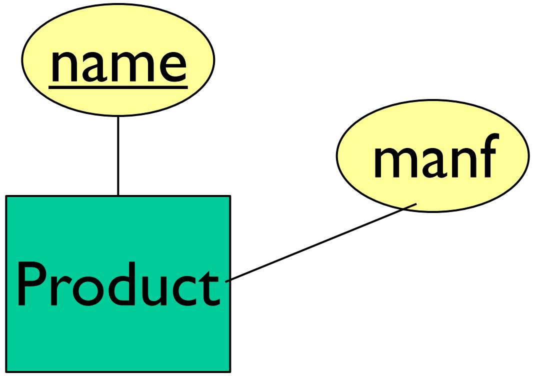
- An Entity Set should satisfy one of the following conditions:
  - It is more than just the name, i.e., it has at least one non-key attrib
  - OR, it is the "many" in a many-one or many-many relationship



- Product: "many" of a many-one relationship
- Manf: has a non-key attrib: addr

Picking the Right Kind of Element

- An Entity Set should satisfy one of the following conditions:
  - It is more than just the name, i.e., it has at least one non-key attrib
  - OR, it is the "many" in a many-one or many-many relationship



- If we had no manf address info, might as well merge into product

Next: Translation into a Relational Schema

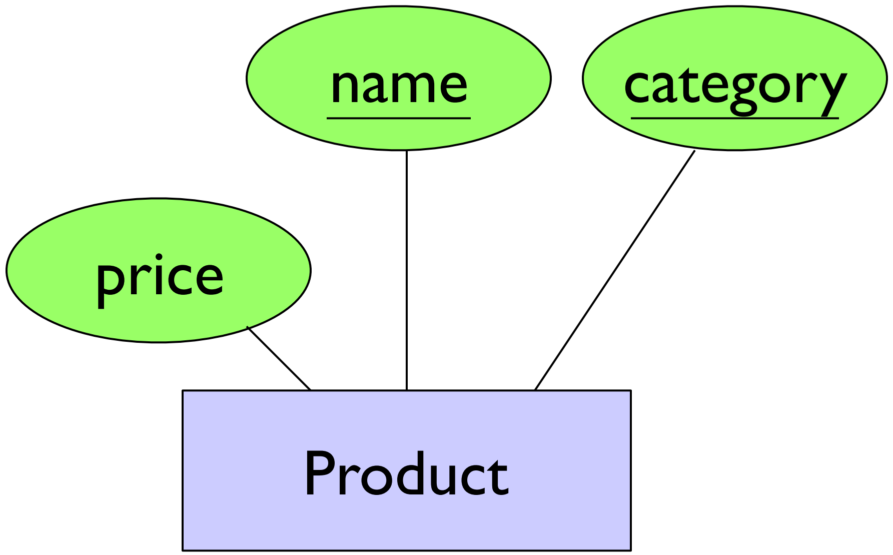
- Now, let’s say we have an ER diagram
- How do we convert it into a relational schema?

- Turns out many database (ER) design software will do this automatically for you!

Entity Set to Relation

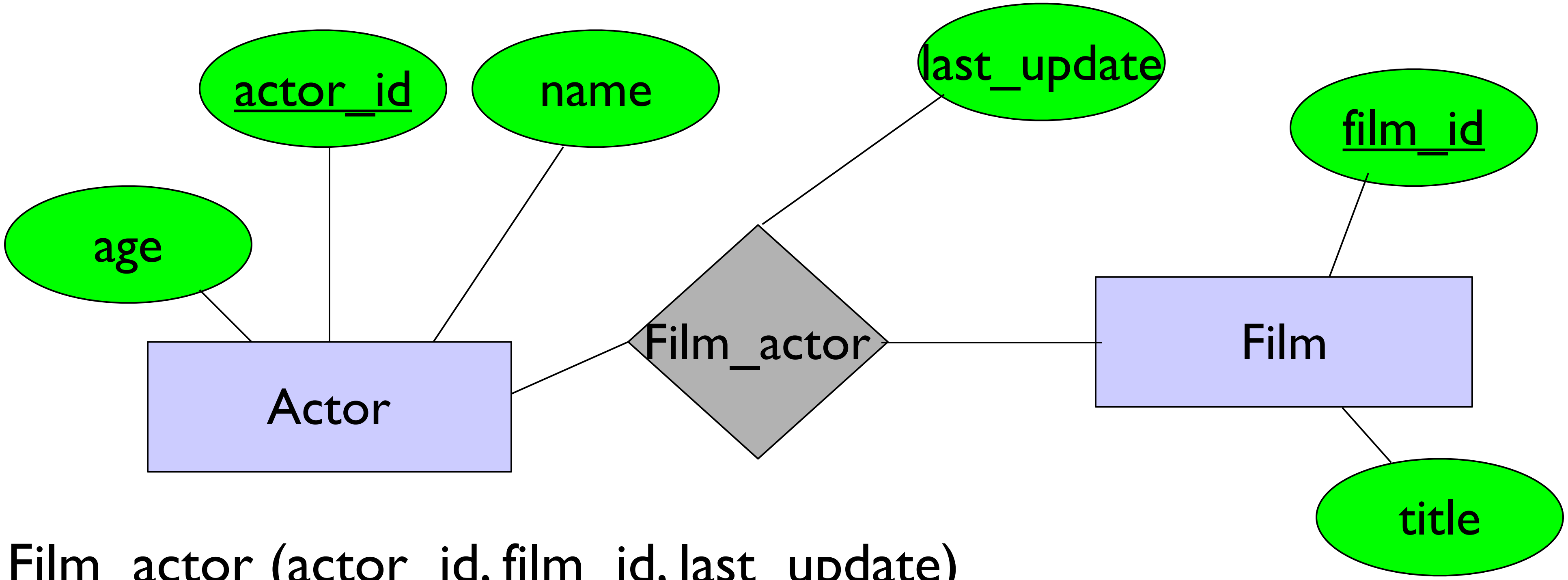
- Relation:

Product (name, category, price)



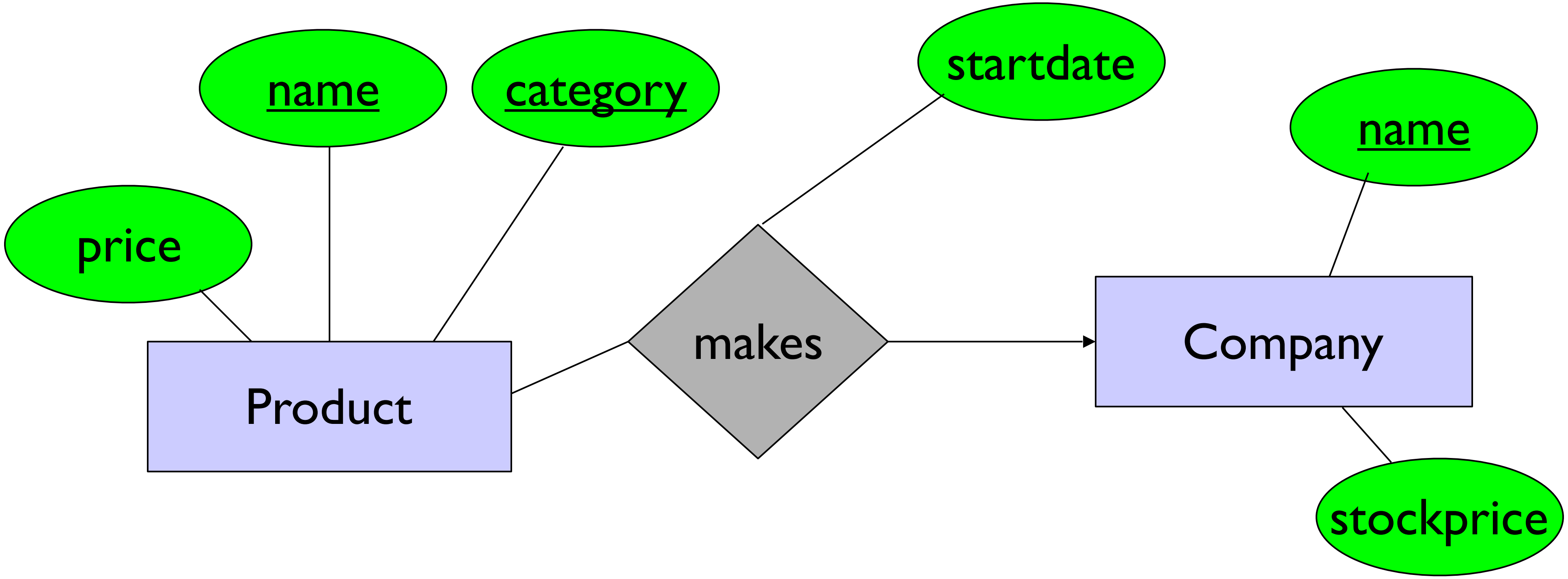
```
CREATE TABLE Product
  (name VARCHAR (25), price REAL, category CHAR (4),
   PRIMARY KEY (name, category));
```

Relationship to Relation: Our Familiar Movie Database



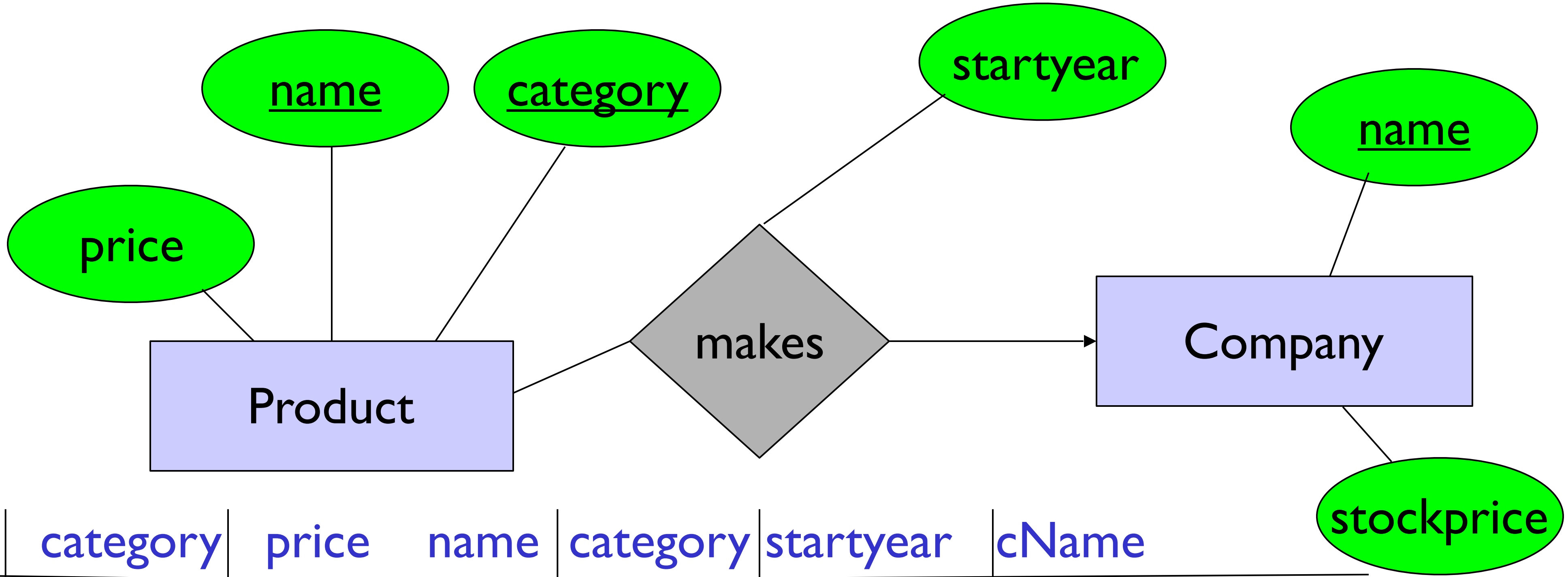
- Film_actor (actor_id, film_id, last_update)
- Include:
  - Key attributes of each Entity Set
  - Attributes of the relationship
- Three tables: Actor, Film_actor, Film

Relationship to Relation: Many to one



- Can improve the representation if it is many-to-one (or one-to-one)
  - Product (price, name, category, startdate, companyname)
  - Instead of:
    - Product (price, name, category),
    - Makes (name, category, startdate, companyname)

Relationship to Relation: Many to one



name	category	price
gizmo	gadgets	19.99
kphone	phone	200.00

name	category	startyear	cName
gizmo	gadgets	1963	gizmoWorks

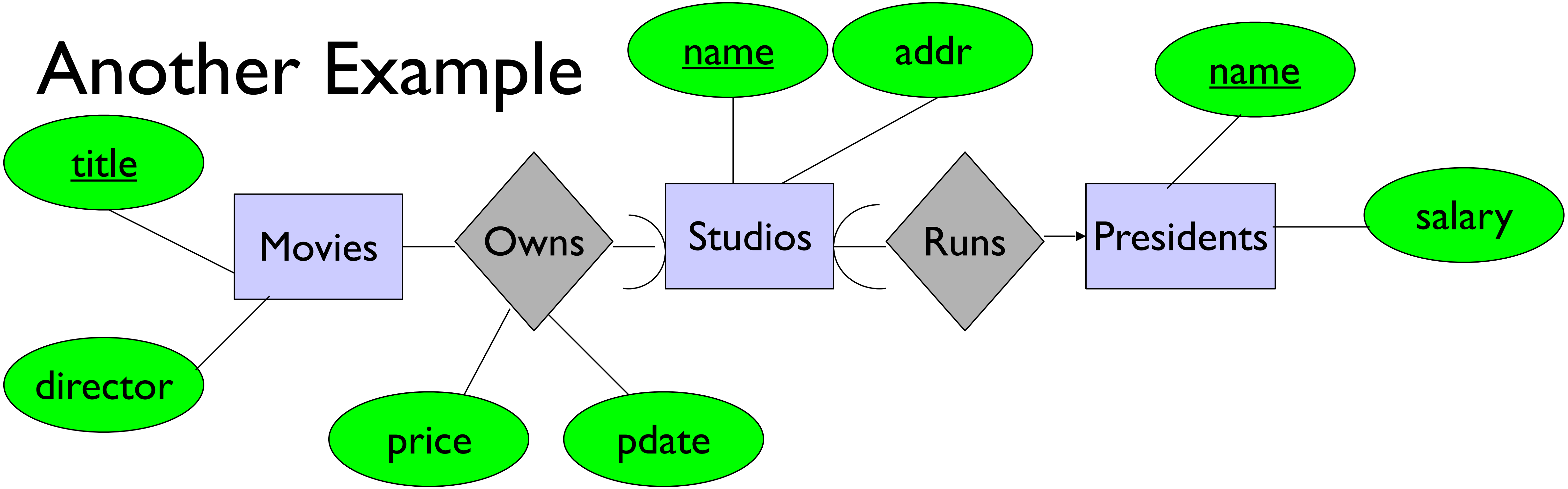
name	category	price	startyear	companyName
gizmo	gadgets	19.99	1963	gizmoWorks
kphone	gadgets	200.0	NULL	NULL

Combining Relations

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

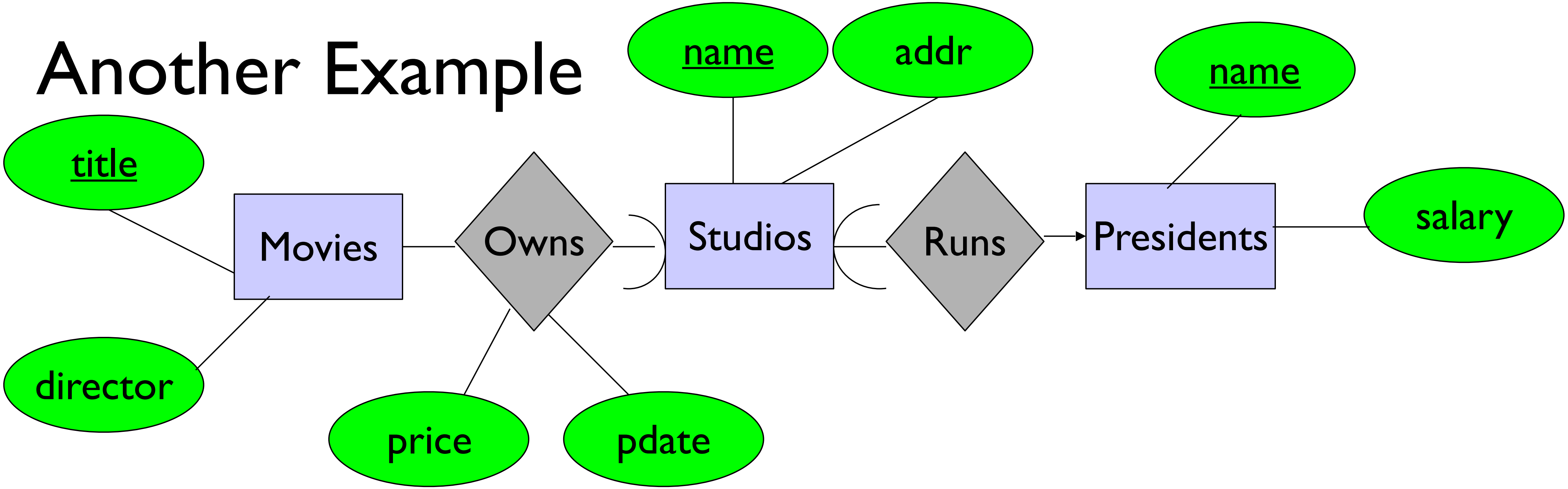
- Typically, when combining two tables into one, there is always a danger of redundant information: the same information represented multiple times
  - Why does the redundancy argument not apply here?
    - One single tuple "smushed" together

Another Example



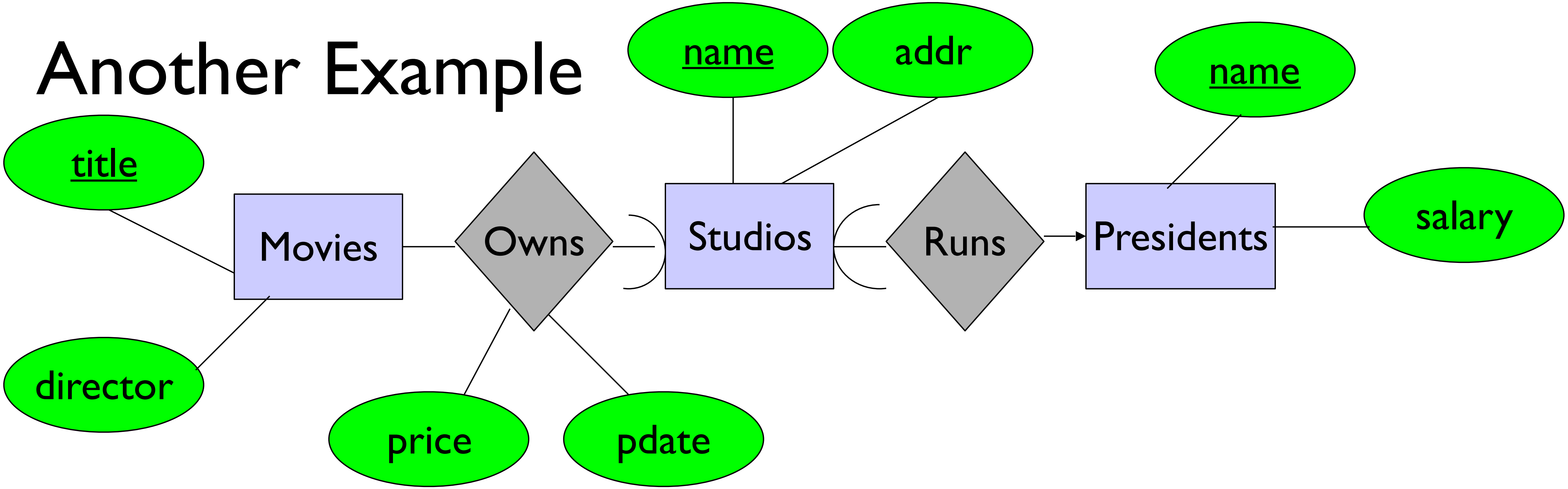
- What relations would we have if we had one relation for each Entity Set and Relationship?

Another Example



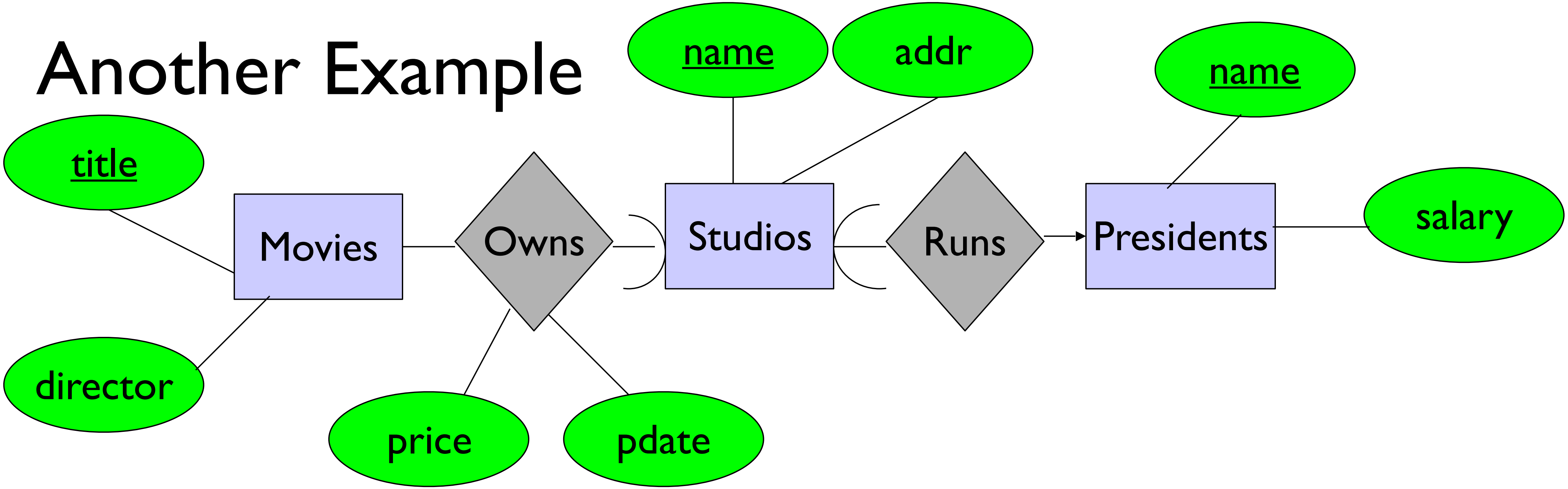
- Movies (title, director)
- Owns (movie_title, price, pdate, studio_name)
- Studios (name, addr)
- Runs (studio_name, president_name)
- Presidents (name, salary)

Another Example



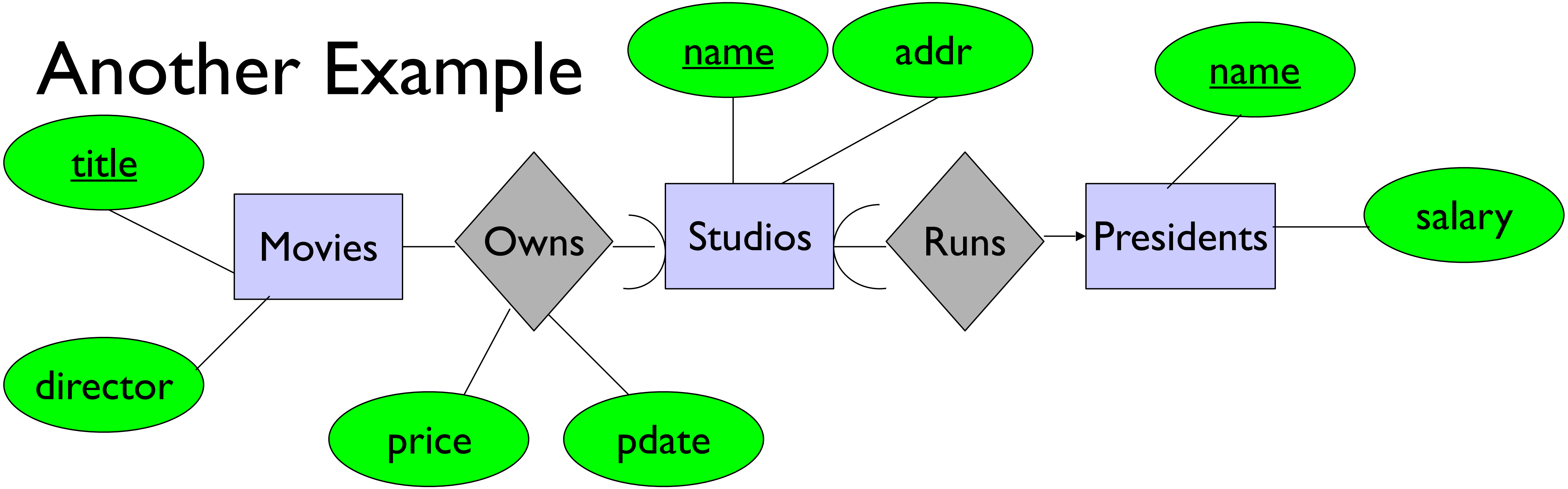
- Movies (title, director, **price, pdate, studio_name**)
- ~~Owns (movie_title, price, pdate, studio_name)~~
- Studios (name, addr)
- Runs (studio_name, president_name)
- Presidents (name, salary)

Another Example



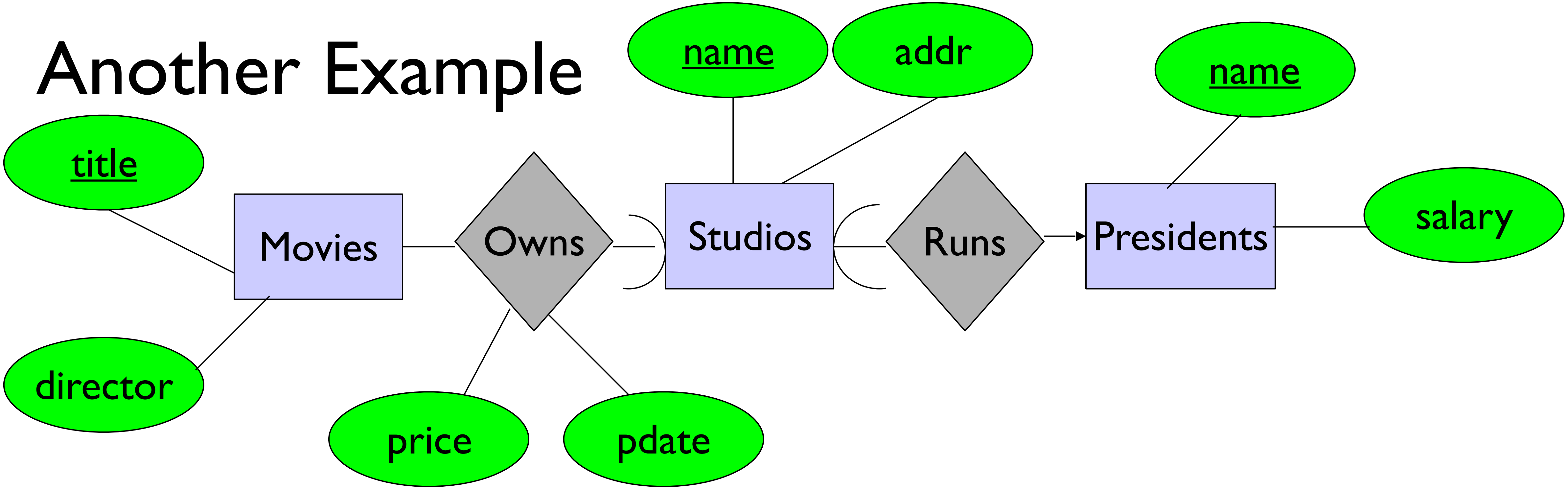
- Movies (title, director, **price, pdate, studio_name**)
- ~~Owns (movie_title, price, pdate, studio_name)~~
- Studios (name, addr, **president_name**)
- ~~Runs (studio_name, president_name)~~
- Presidents (name, salary)

Another Example



- Movies (title, director, **price, pdate, studio_name**)
- ~~Owns (movie_title, price, pdate, studio_name)~~
- Studios (name, addr, **president_name, salary**)
- ~~Runs (studio_name, president_name)~~
- ~~Presidents (name, salary)~~

Another Example



- Movies (title, director, **price, pdate, studio_name**)
- Studios (name, addr, **president_name, salary**)

- Why not go all the way and have all the information in one table?
- Massive redundancy!

One Final Piece: Functional Dependencies and Normalization

- Start with requirements, then abstract them out into an ER diagram, convert them into relations
  - Are we done?
  - **Not quite! There can still be issues with the relations thus designed, causing us problems**
  - Or, we may start with a set of relations (not so carefully designed without ER diagrams) and have to "fix it"

- The concepts of "functional dependencies" and "normalization" help us find our way to good designs in such cases
  - We'll do a very quick intuitive overview… but will skip detailed discussions

- Let's consider an example

Is this a good design?

- Individuals with several phones

Address	SSN	Phone Number
10 Green	123-456-789	(201) 233-1456
10 Green	123-456-789	(201) 123-3439
431 Purple	987-654-321	(145) 241-2131
431 Purple	987-654-321	(312) 123-1287

Is this a good design?

Address	SSN	Phone Number
10 Green	123-456-789	(201) 233-1456
10 Green	123-456-789	(201) 123-3439
431 Purple	987-654-321	(145) 241-2131
431 Purple	987-654-321	(312) 123-1287

- Individuals with several phones
- Redundancy:
  - Address repeated multiple times
- Update anomalies:
  - If we update address of person with phone number 201-233-1456, there will be two addresses of that person (wrong!)
- Deletion anomalies:
  - If we delete the phone number for a person, we need to check if there is another phone number for said person, else may lose address info.

Better Designs Exist!

Address	SSN
10 Green	123-456-789
431 Purple	987-654-321

SSN	Phone Number
123-456-789	(201) 233-1456
123-456-789	(201) 123-3439
987-654-321	(145) 241-2131
987-654-321	(312) 123-1287

- Each bit of info only exists “once”

- How would you recover the original relation via these two relations?
- A: a natural join

- Unfortunately, will not detect this even with principled ER design and translation

Better Designs Exist!

Address	SSN
10 Green	123-456-789
431 Purple	987-654-321

SSN	Phone Number
123-456-789	(201) 233-1456
123-456-789	(201) 123-3439
987-654-321	(145) 241-2131
987-654-321	(312) 123-1287

- Each bit of info only exists "once"
- Splitting relations into multiple relations that minimizes redundancy is called **normalization**
  - We use functional dependencies to guide us
- There can be other objectives (in additional to minimizing redundancy)

Better Designs Exist!

Address	SSN
10 Green	123-456-789
431 Purple	987-654-321

SSN	Phone Number
123-456-789	(201) 233-1456
123-456-789	(201) 123-3439
987-654-321	(145) 241-2131
987-654-321	(312) 123-1287

- Each bit of info only exists "once"

- Benefits of normalization:
  - Removing redundancy, minimizes update, and delete anomalies
- Downsides of normalization:
  - Joins are costly sometimes; want to avoid them
  - This is the main reason why people avoid normalization

Functional Dependencies

- How do perform normalization? use **functional dependencies!**
- A form of constraint on your data

$$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$$

- If two tuples agree on values of the LHS, they must agree on the values of the RHS: **holds for all instances!**
- Q: Have we discussed any examples of FDs so far?

- Special case: LHS is the primary key (or any key)
  - Movies (<u>title</u>, director, price, pdate, studio_name)
    - title ⟶ director, price, pdate, studio_name

Functional Dependencies improve Designs

- Functional dependencies explicitly capture almost (see MVDs!) all interesting redundancy-oriented constraints in your data
- Provided by the user as "semantics"
- Returning to our example
  - SSN → Address [not just here]
  - SSN ↛ Phone Number
- This causes us to "factor out" SSN and Address into a separate relation
  - "decompose" or "normalize"

Address	SSN	Phone Number
10 Green	123-456-789	(201) 233-1456
10 Green	123-456-789	(201) 123-3439
431 Purple	987-654-321	(145) 241-2131
431 Purple	987-654-321	(312) 123-1287

Lots more to say about FDs and Normalization!

- Various types of "normal forms": BCNF, 1NF, 2NF, 3NF & trade offs!
- Calculus of FDs: closure, splitting/combining, basis
- But we won't spend more time on this

- Rule: "decompose/normalize" your relations to minimize redundancy, while ensuring that you can correctly reconstruct the original relation.
  - But beware that normalization can lead to performance overheads due to joins!