Conceptual Design with ER Model

DSCI 551

Wensheng Wu

Lecture Outline

- Steps in building a database application
- Conceptual design with ER model

Steps in Building a DB Application

- Step 0: pick an application domain
  - E.g., course management

ER => relational model (tables)
RDBMS

- Step 1: conceptual design
  - Decide on what to model in the application domain
    - E.g., instructors, students, courses, etc.
  - need a modeling language to express what you want
  - ER model is the most popular such language
  - output: an ER diagram of the app. domain

Steps in Building a DB Application

- Step 2: pick a type of DBMS
  - Here we use relational DBMS
- Step 3: translate ER design to a relational schema
  - use a set of rules to translate ER to rel. schema
  - use a set of schema refinement rules to transform the above rel. schema into a good rel. schema
  - (3NF, BCNF, 4NF)
- At this point
  - you have a good relational schema on paper

Steps in Building a DB Application

- Subsequent steps include
  - implement your relational DBMS using a "database programming language" called SQL
  - ordinary users cannot interact with the database directly
  - and the database also cannot do everything you want
  - hence write your application program in Php, C++, Java, Python, etc. to handle the interaction and take care of things that the database cannot do
- So, the first thing we should start with is to learn ER model ...

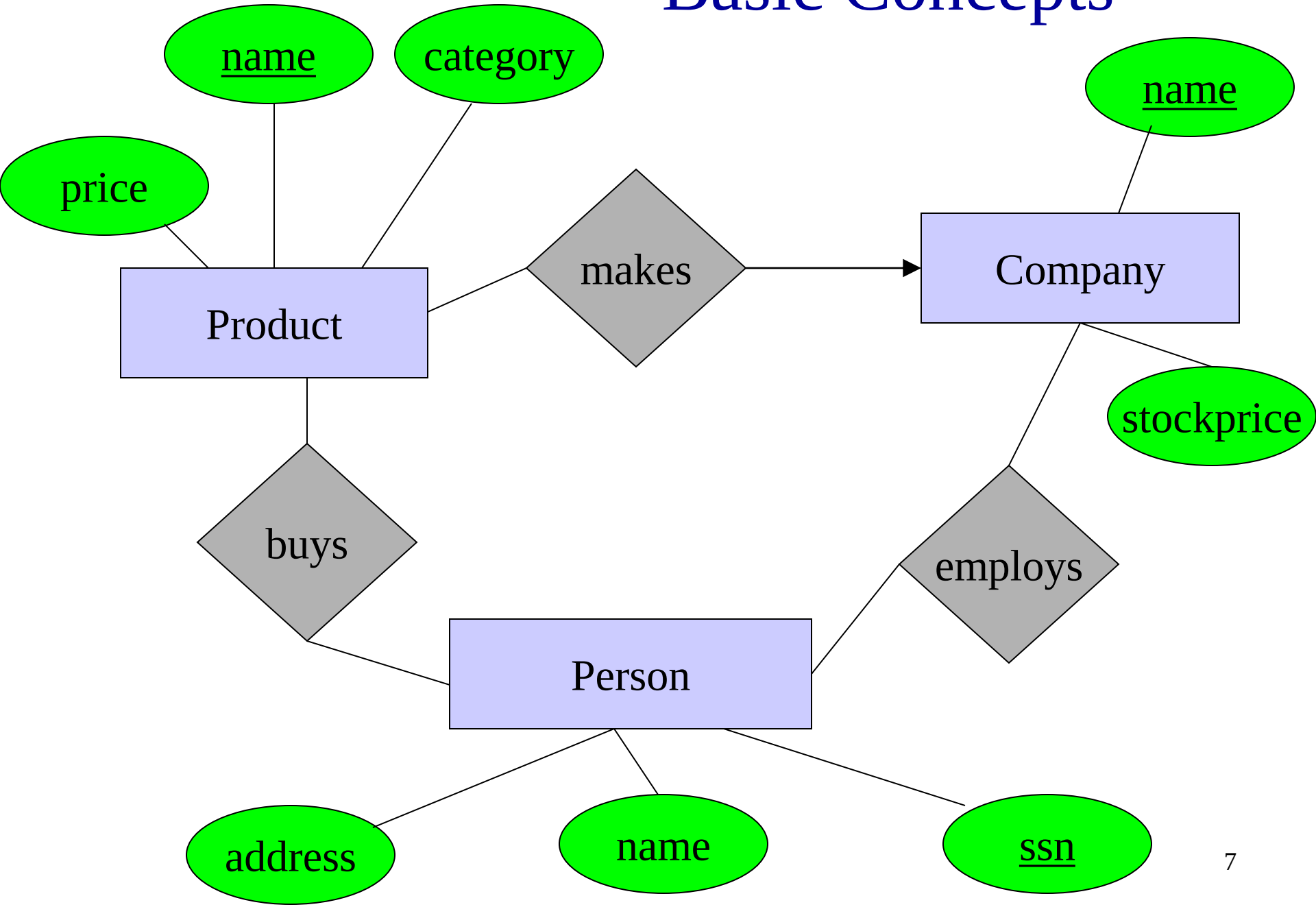
ER Model

- Gives us a language to specify
  - what information the db must hold
  - what are the relationships among components of that information
- Proposed by Peter Chen in 1976
- What we will cover
  - basic stuff
  - subclasses (is-a): person, student
  - constraints (PK, FK: emp(did), dept(id))
  - weak entity sets
  - design principles

Multiset/bag

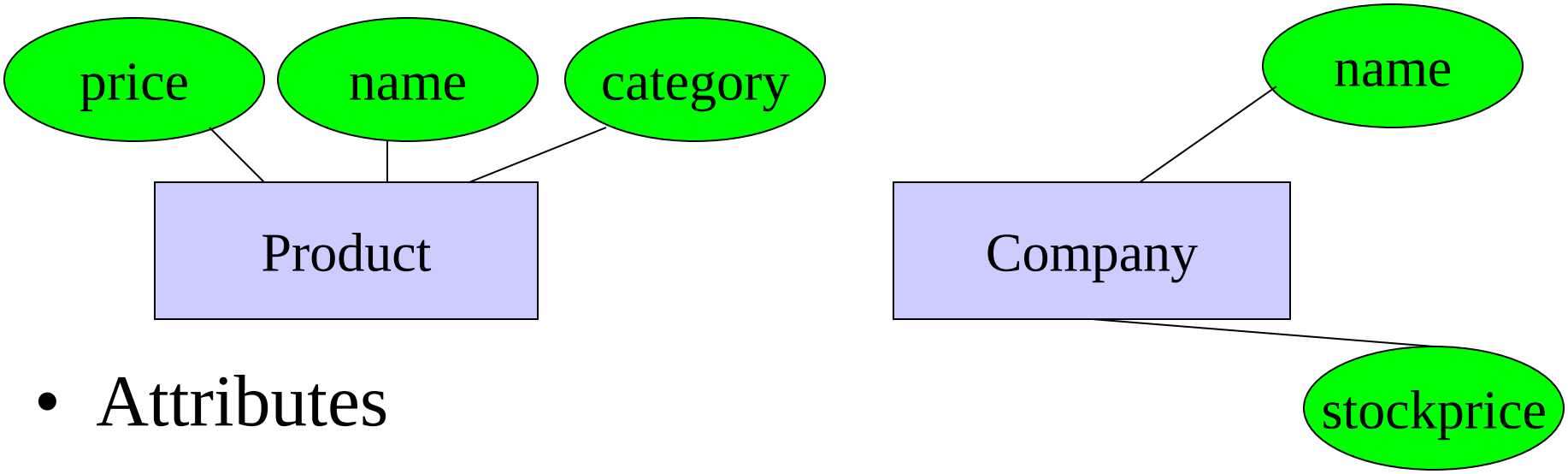
http://www.csc.lsu.edu/~chen/

Basic Concepts



Entities and Attributes

- Entities
  - real-world objects distinguishable from other objects
  - described using a set of attributes

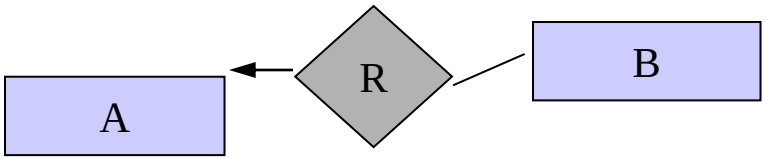


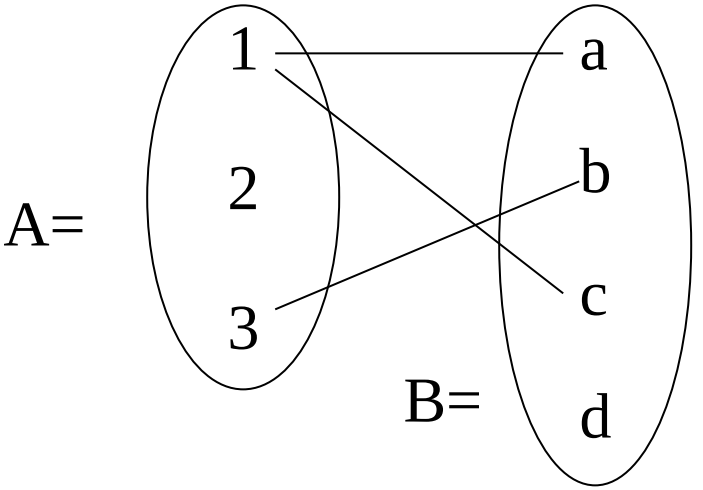
- Attributes
  - each has an atomic domain: string, integers, reals, etc.
- Entity set: a collection of similar entities

Relationships

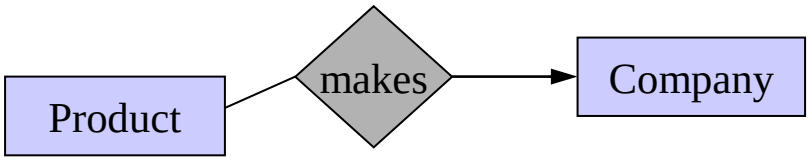
- A mathematical definition:
  - if A, B are sets, then a relationship R is a subset of A x B
- A={1,2,3}, B={a,b,c,d},

  R = {(1,a), (1,c), (3,b)}





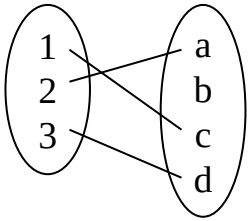
makes is a subset of **Product** x **Company**:

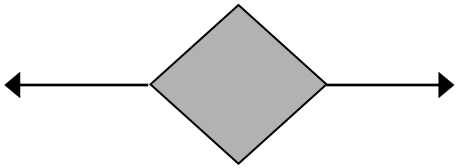


More about relationships ...

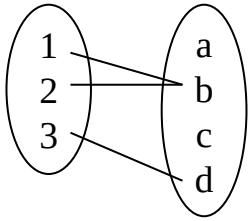
Multiplicity of E/R Relationships

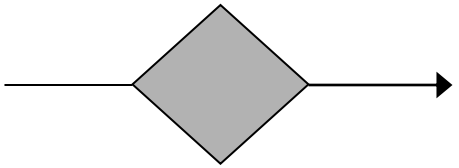
- one-one:
  - One = at most one



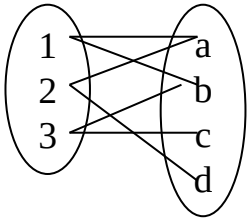


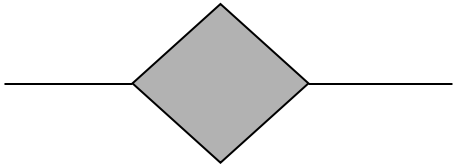
- many-one/one-many
  - Here left side = many

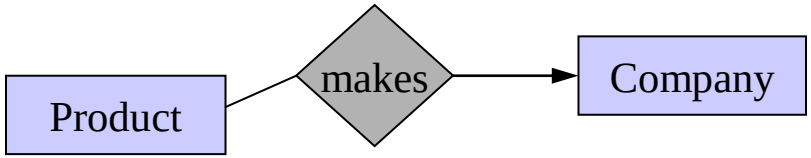




- many-many

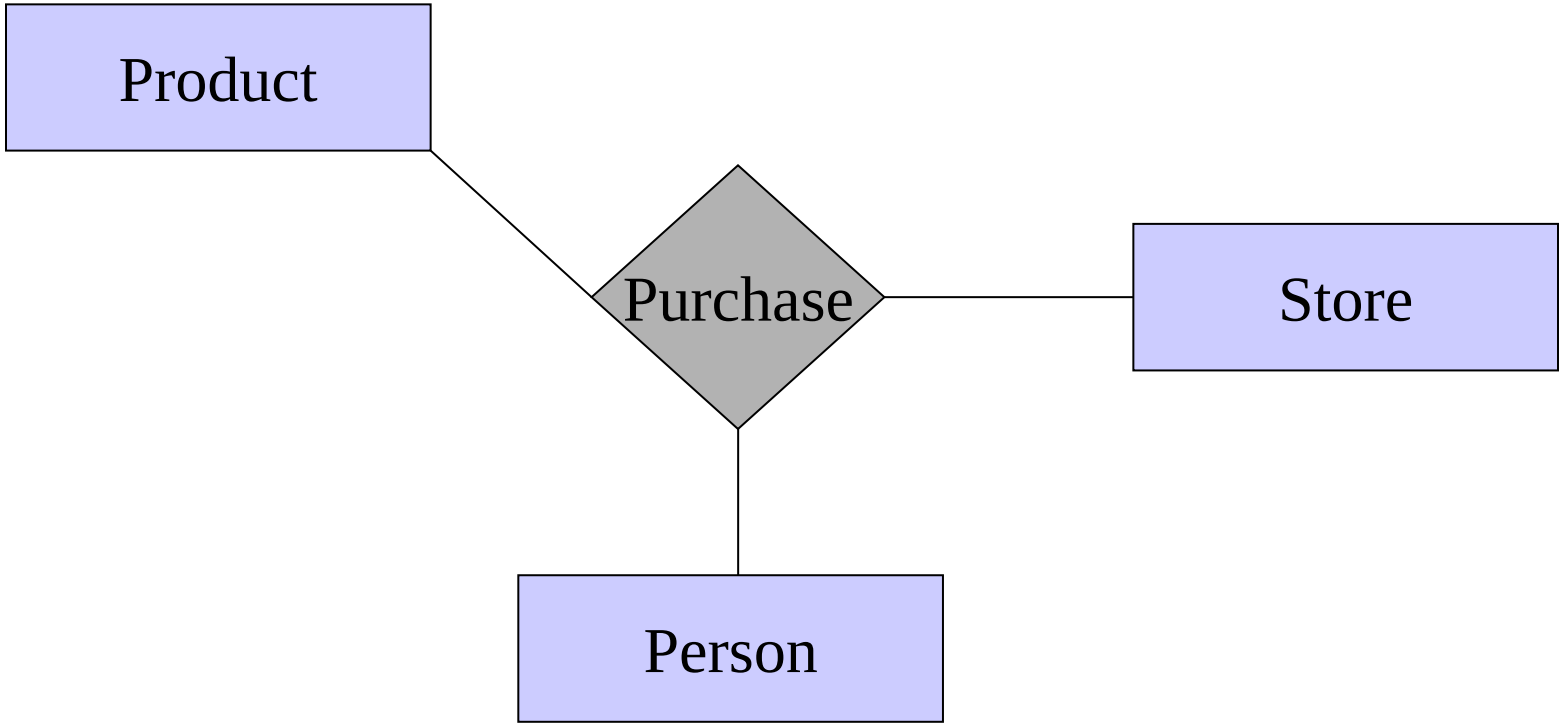






Multiway Relationships

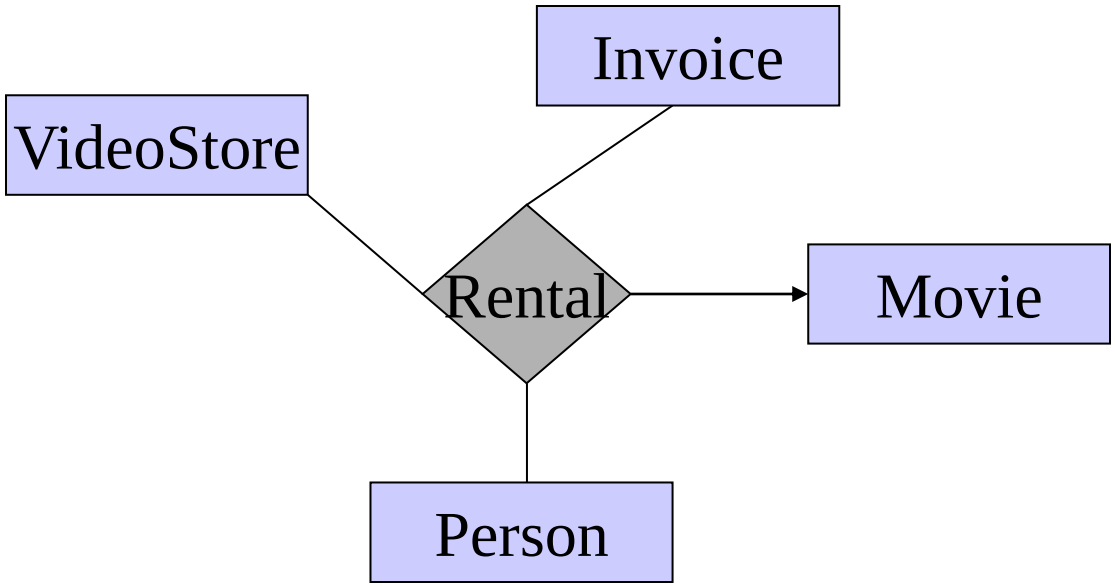
How do we model a purchase relationship between buyers, products and stores?



Can still model as a mathematical set (how ?)

Arrows in Multiway Relationships

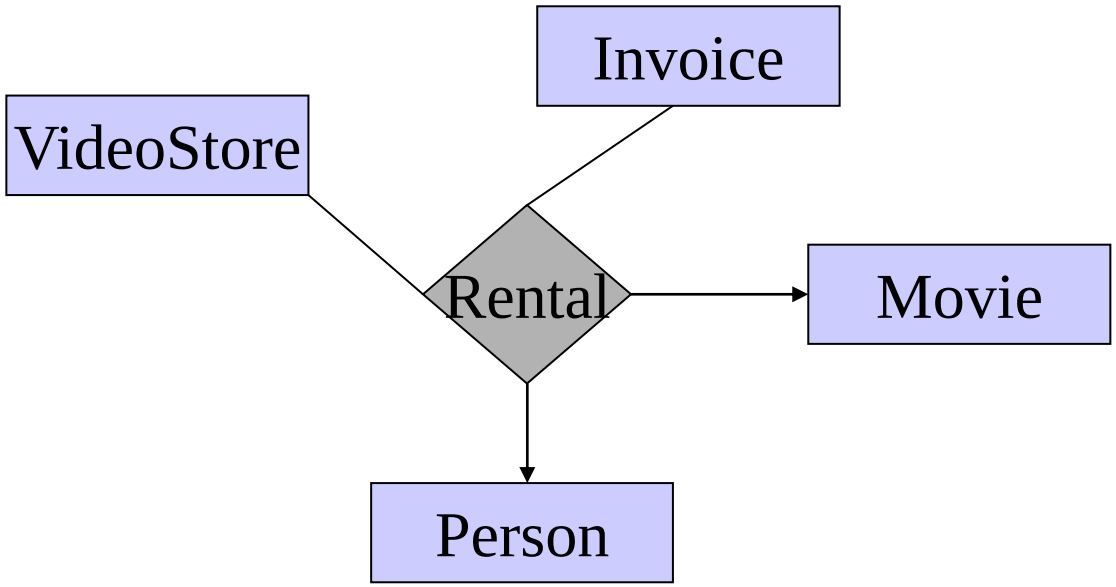
Q: what does the arrow mean?



A: if I know the store, person, invoice, I know the movie too

Arrows in Multiway Relationships

Q: what do these arrows mean?

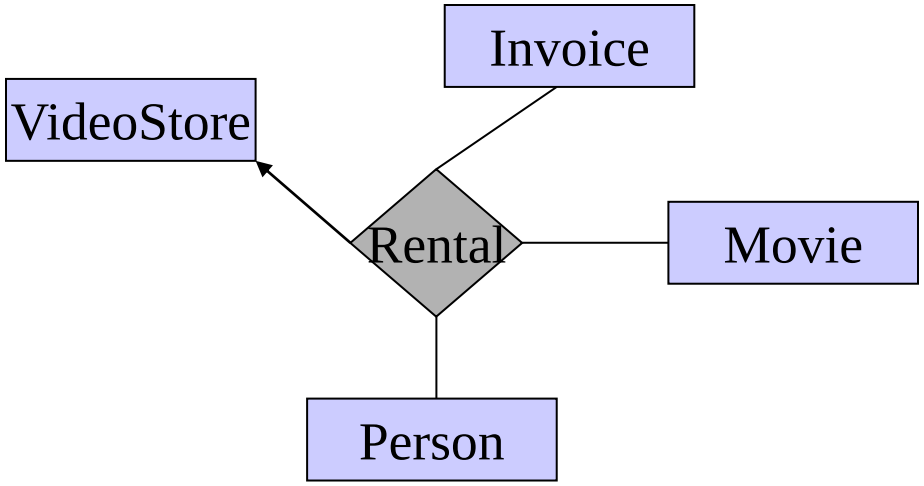


A: store, person, invoice determines movie
and store, invoice, movie determines person

Arrows in Multiway Relationships

Q: how do I say: “invoice determines store” ?

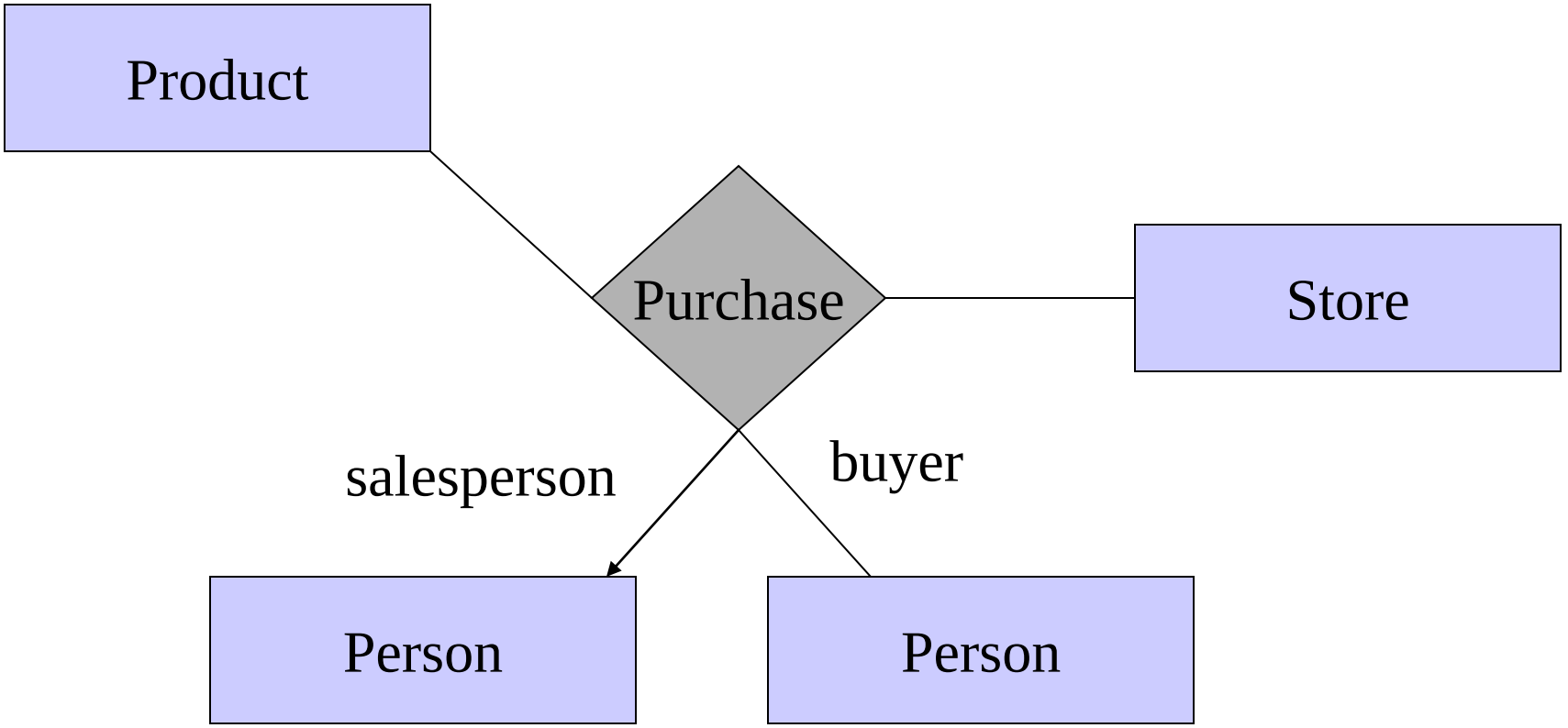
A: no good way; best approximation:



Relational model captures many-one relationships in functional dependencies, e.g., invoice → store

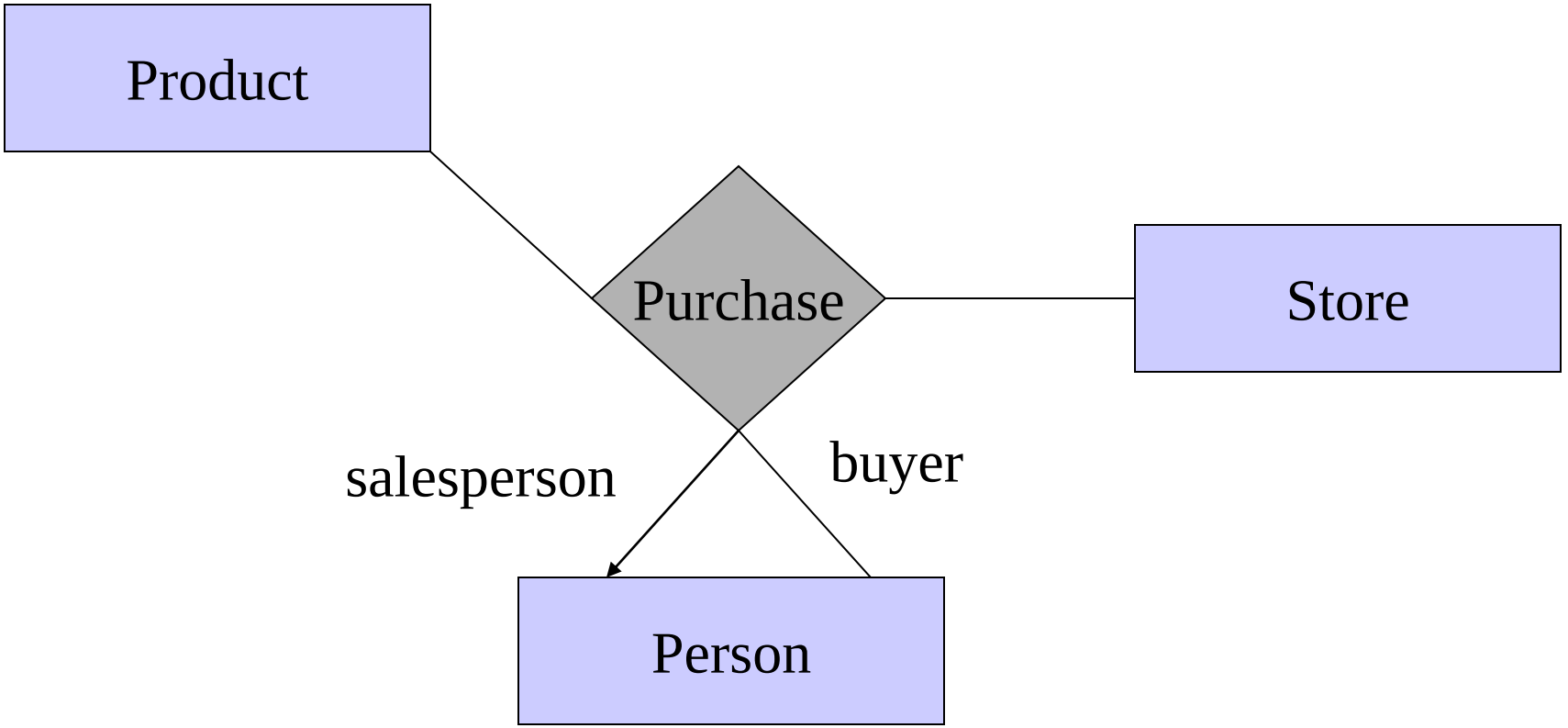
Roles in Relationships

What if we need an entity set twice in one relationship?

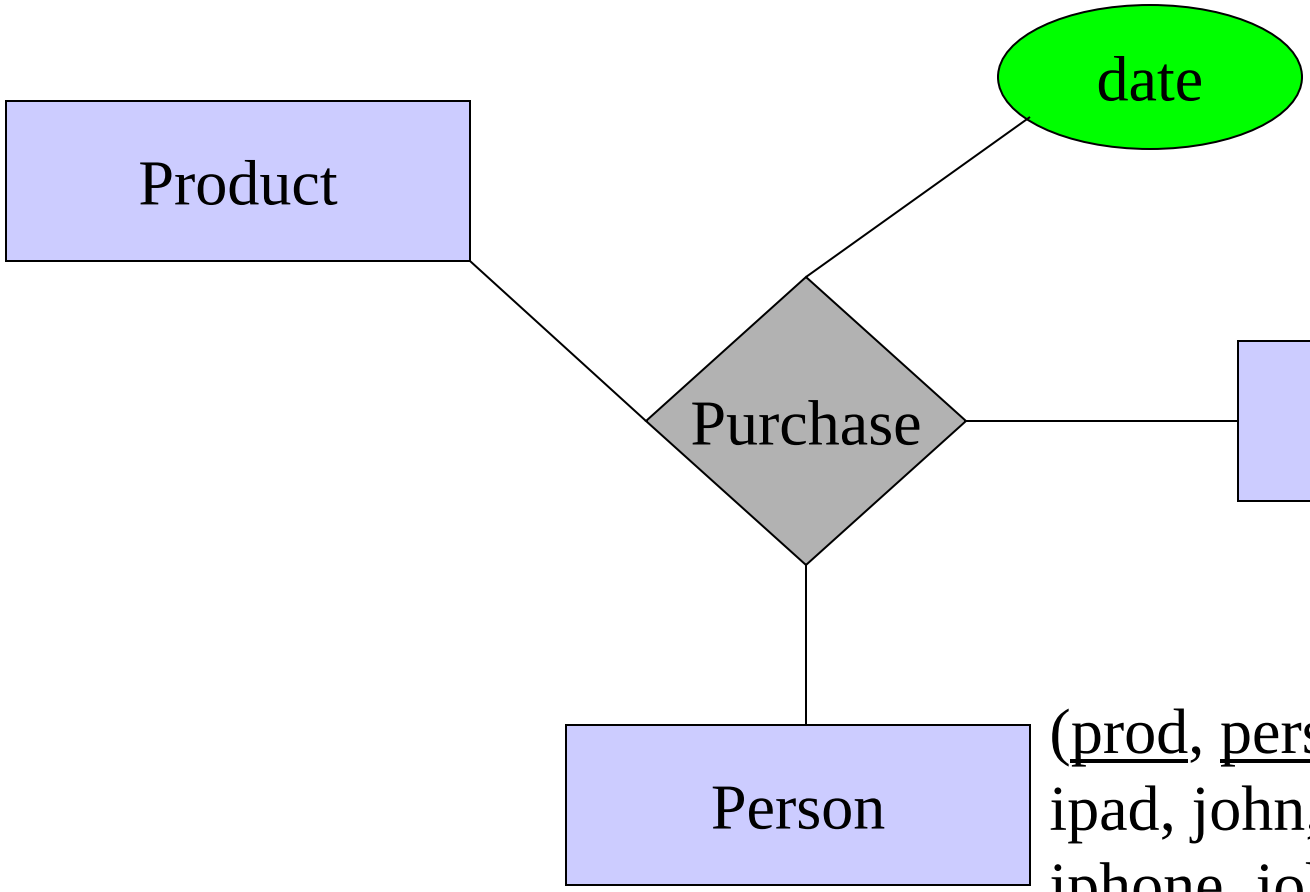


Roles in Relationships

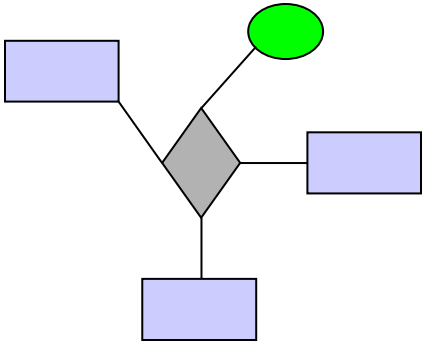
What if we need an entity set twice in one relationship?



Attributes on Relationships



(<u>prod</u>, <u>person</u>, <u>storeID</u>, date)
ipad, john, s1, 9/30
iphone, john, s1, 9/30
ipad, john, s2, 10/1



Converting Multiway Relationships to Binary

date

ProductOf

Product

Purchase

StoreOf

Store

(<u>prod</u>, <u>person</u>, <u>storeID</u>, date)
ipad, john, s1, 9/30
iphone john, s1, 9/30
ipad, john, s2, 10/1

BuyerOf

Person

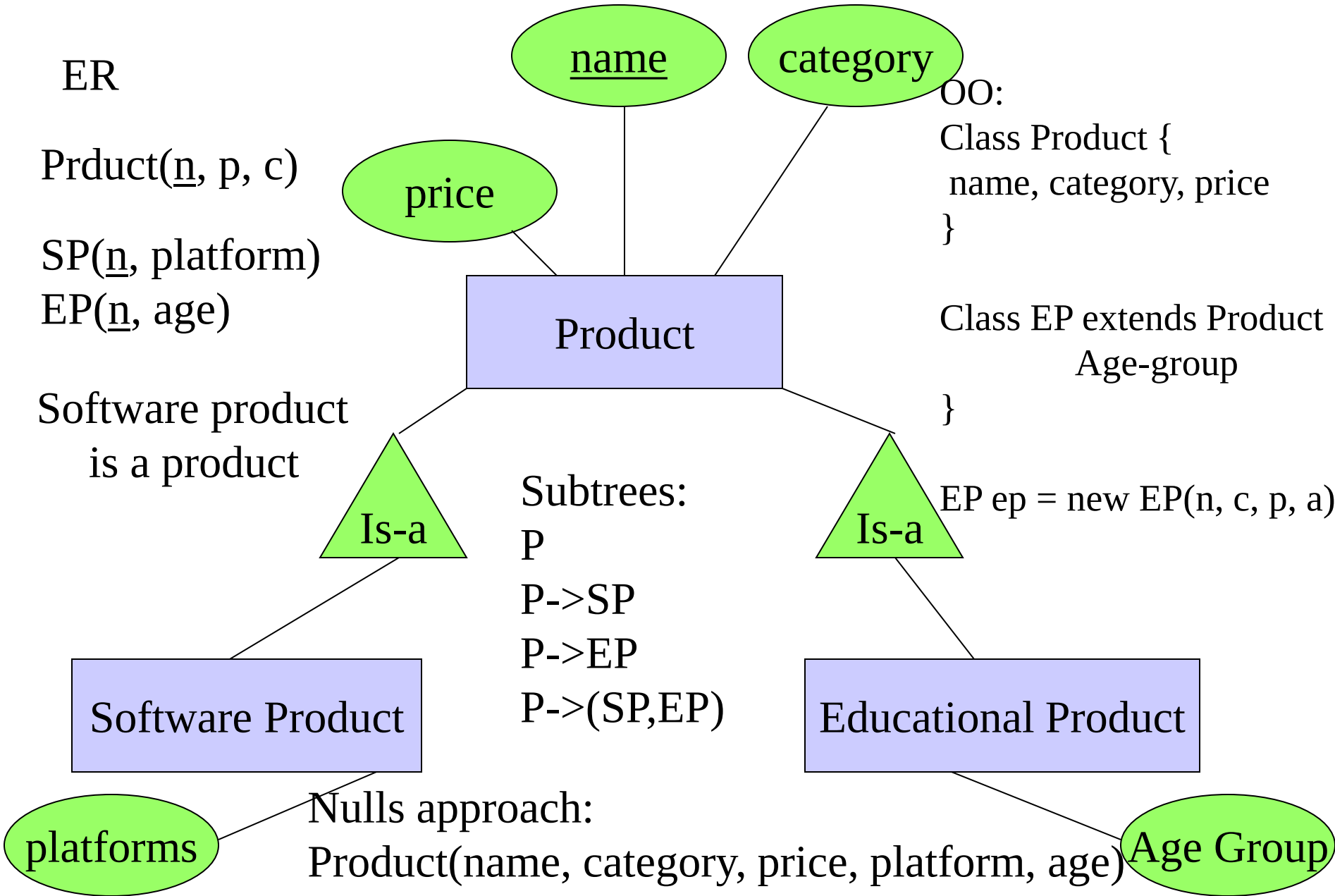
Relationships: Summary

- Modeled as a mathematical set
- Binary and multiway relationships
- Converting a multiway one into many binary ones
- Constraints on the degree/multiplicity of the relationship
  - many-one, one-one, many-many
  - limitations of arrows
- Attributes of relationships
  - not necessary, but useful

Roadmap

- What we will cover
  - basic stuff
  - subclasses 
  - constraints
  - weak entity sets
  - design principles

Subclasses in ER Diagrams



Subclasses in ER Diagrams

- Assume subclasses form a tree.
  - I.e., no multiple inheritance.
- Isa triangles indicate the subclass relationship.
  - Point to the superclass.

Subclasses

- Subclass = special case = fewer entities = more properties.
- Example: Ales are a kind of beer.
  - Not every beer is an ale, but some are.
  - Let us suppose that in addition to all the *properties* (attributes and relationships) of beers, ales also have the attribute *color*.

Example

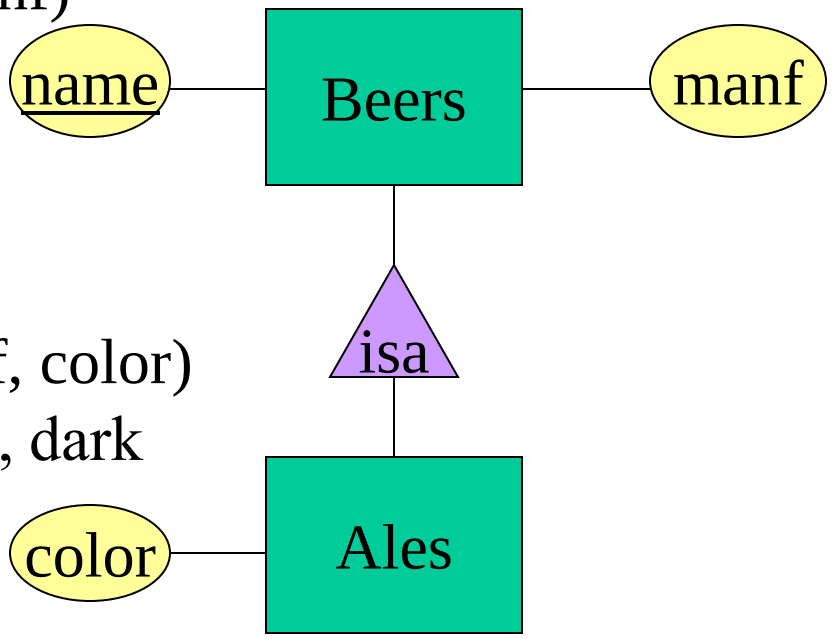
OO:

Think about Java:

Ales a = new Ales(name, manf, color)

ER

Beers(name, manf)
Budweiser, A-B



Beers(name, manf)
BudLight, Pete's
Budweiser, A-B

Ales(name, manf, color)
BudLight, Pete's, dark

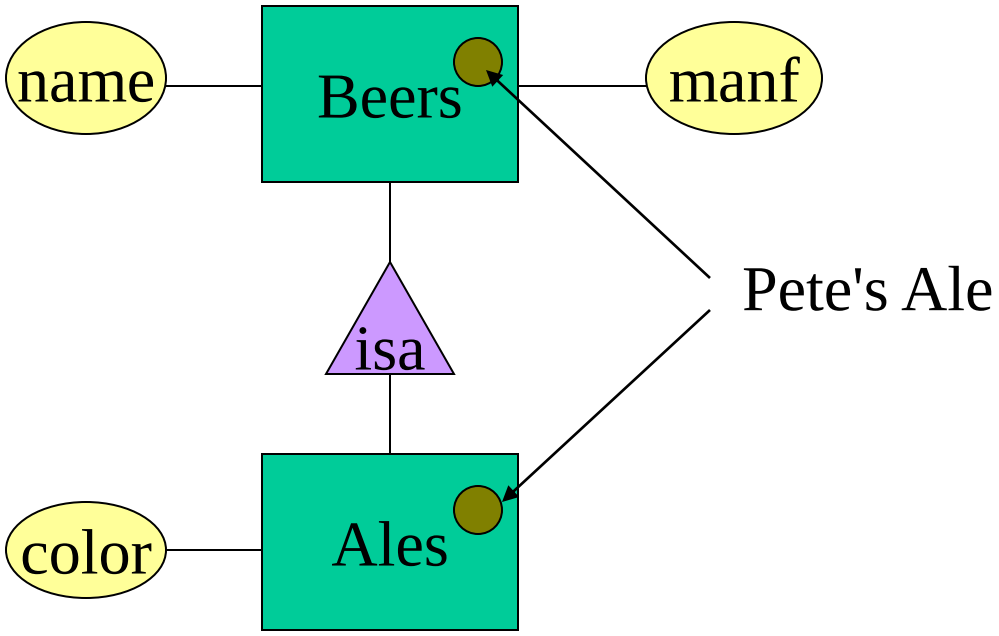
Ales(name, color)
BudLight, dark

Nulls

ER vs. Object Oriented Subclasses

- In the object-oriented world, objects are stored in one class only.
  - Subclasses inherit all properties from superclasses.
  - All properties of the object are stored together.
- In contrast, in the E/R view, entities may have components in all subclasses to which they belong.
  - Matters when we convert to relations.

Example



Roadmap

- What we will cover
  - basic stuff
  - subclasses
  - constraints 
  - weak entity sets
  - design principles

Constraints

- A constraint = an assertion about the data in the database that must be true at all times
- Part of the database schema
- Very important in database design
  - To ensure data integrity

Modeling Constraints

Finding constraints is part of the modeling process.
Commonly used constraints:

Keys: social security number uniquely identifies a person.

Single-value constraints: a person can have only one spouse.
create table person(ssn, name …, age int check(age <= 150))

Referential integrity constraints: if you work for a company, it must exist in the database.
(exactly one: age not null)

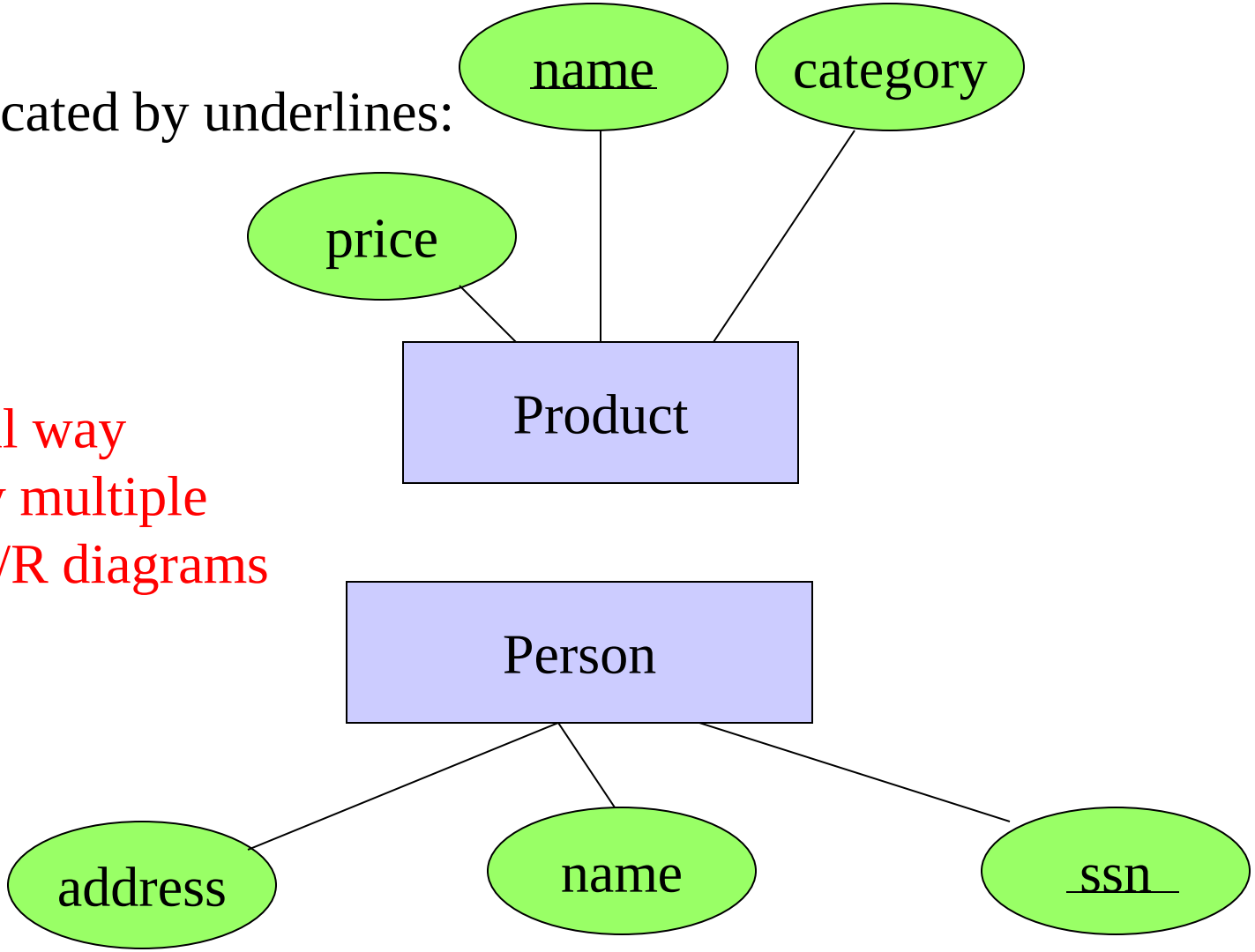
Domain constraints: peoples' ages are between 0 and 150. (check)
General constraints: all others (e.g., at most 50 students can enroll in a class) // create assertion as SQL query

Why Constraints are Important?

- Give more semantics to the data
  - help us better understand it
- Allow us to refer to entities (e.g., using keys)
- Enable efficient storage
  - E.g., store ages as tiny integer (1 byte for example)
  - tinyint in mysql
- Enable efficient lookup
  - E.g., creating an index on key

Keys in E/R Diagrams

Indicated by underlines:

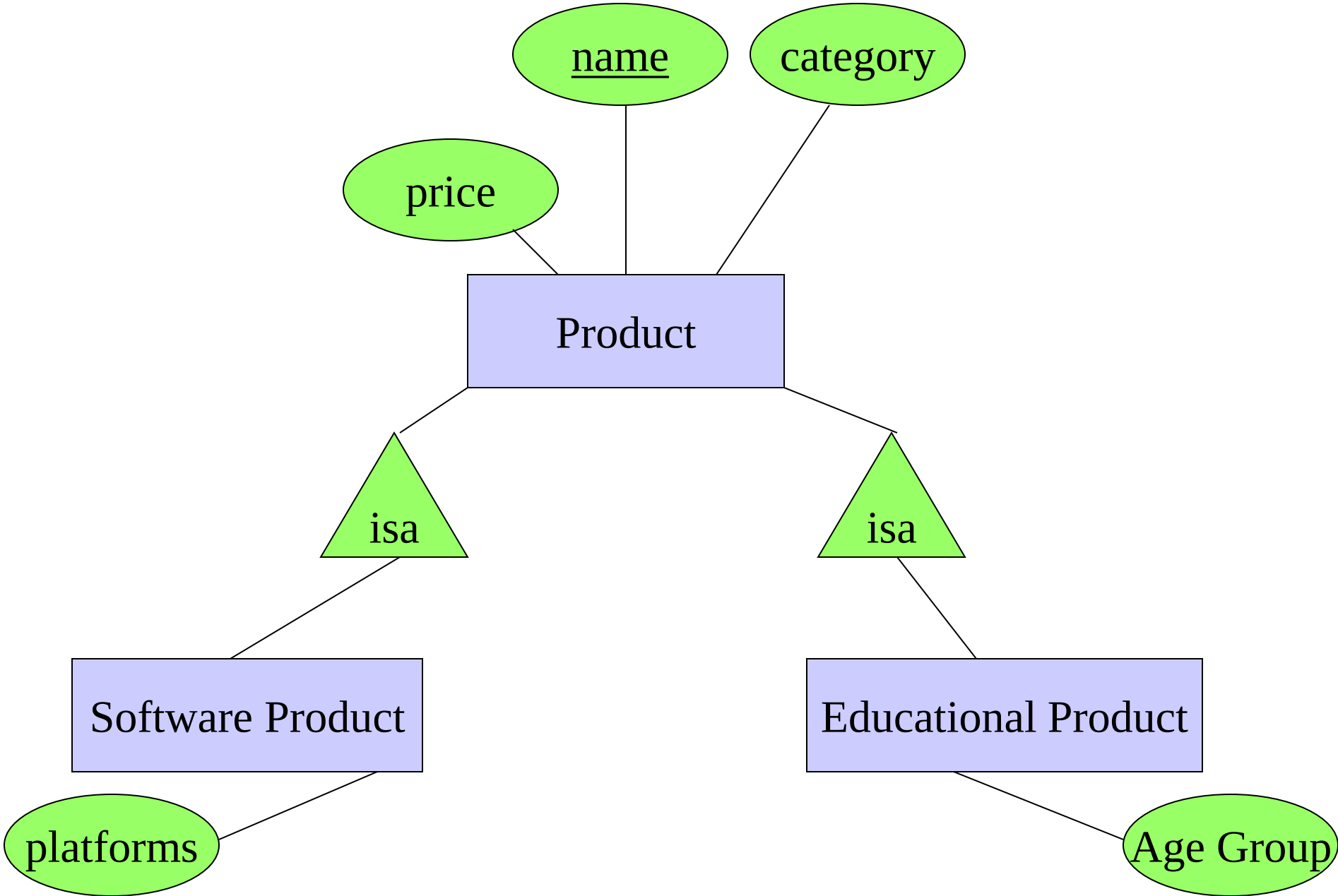


No formal way
to specify multiple
keys in E/R diagrams

More about Keys

- Every entity set must have a key
  - why?
- A key can consist of more than one attribute
- There can be more than one key for an entity set
  - one key will be designated as primary key
- Requirement for key in an isa hierarchy
  - Root entity set has all attributes needed for a key

Subclasses in ER Diagrams

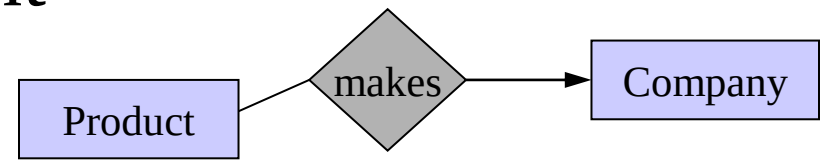


Single Value Constraint

- An entity has at most one value for a given attribute or relationship

- An attribute (age) of an entity set has a single value (25) or NULL
  - i.e., the value may be missing

- A many-one relationship also implies a single value constraint



Referential Integrity Constraint

- Ref. int. constraint: exactly one value exists in a given role

- An attribute has a non-null, single value
  - this can be considered a kind of ref. int. constraint

- However, we more commonly use such constraints to refer to relationships (FK)

Referential Integrity Constraints

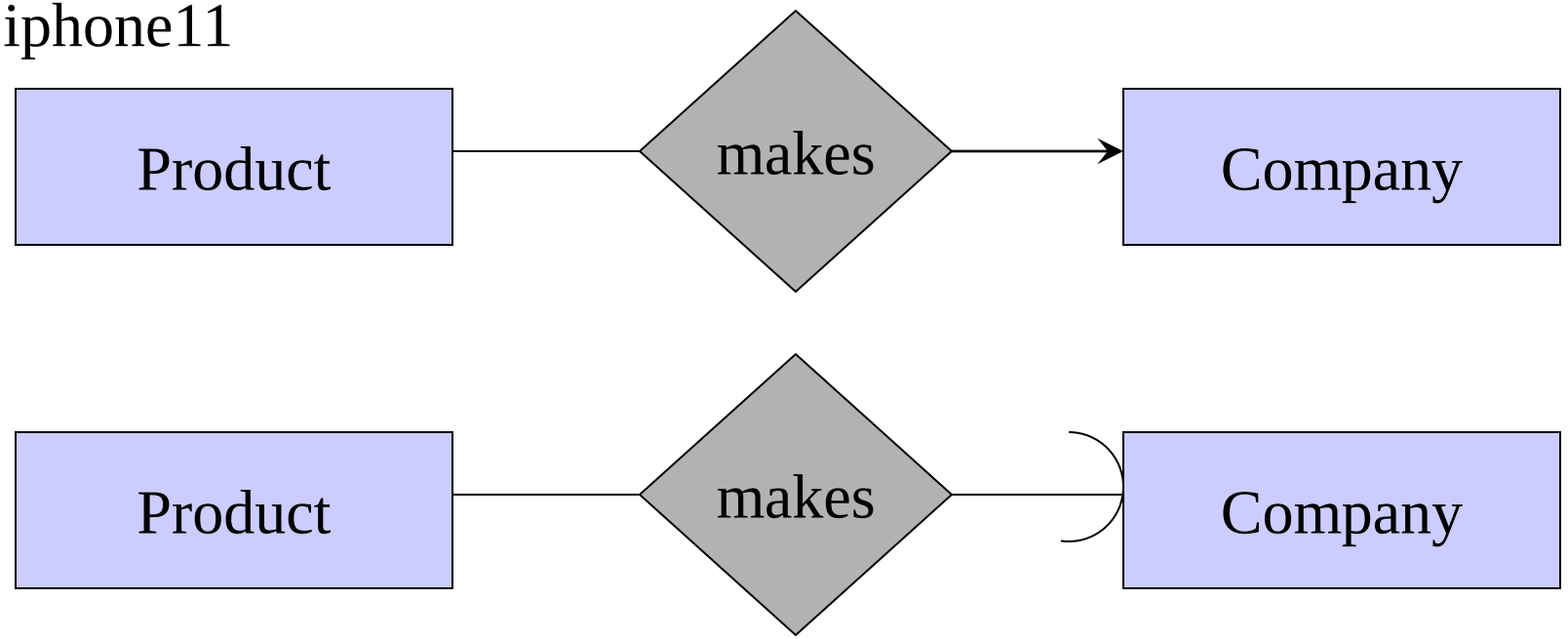
- In some formalisms we may refer to other object but get garbage instead
  - e.g. a dangling pointer in C/C++
- The Referential Integrity Constraint on relationships explicitly requires a reference to exist

Referential Integrity Constraints

Product(id, tite)
100, iphone11

Makes(pid, cname)

Company(name, addr)



- This will be even clearer once we get to relational databases

Roadmap

- What we will cover
  - basic stuff
  - subclasses
  - constraints
  - weak entity sets 
  - design principles

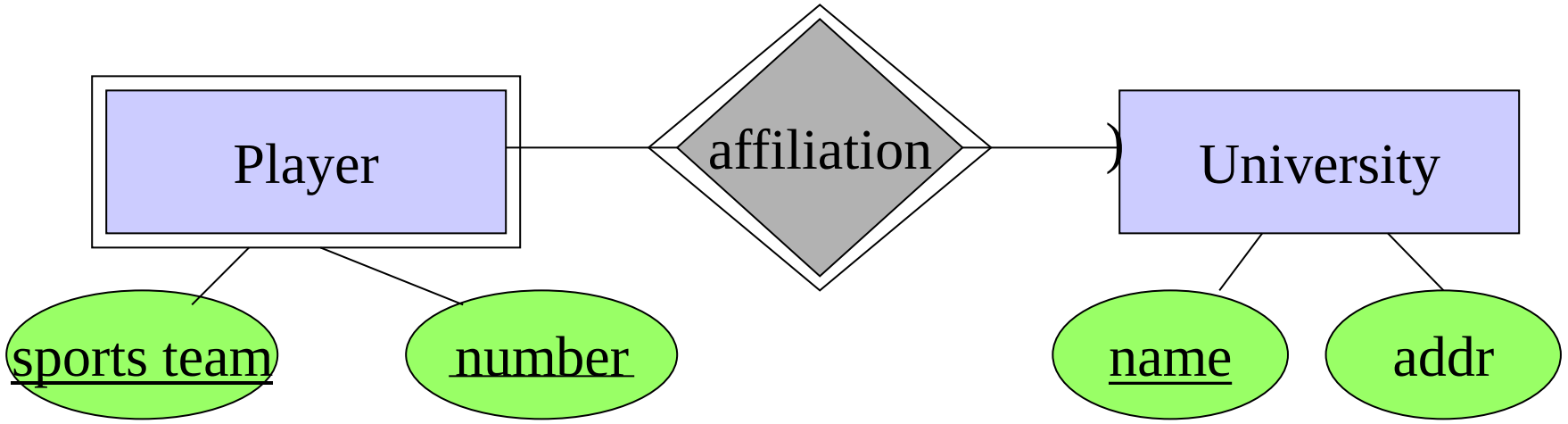
Weak Entity Sets

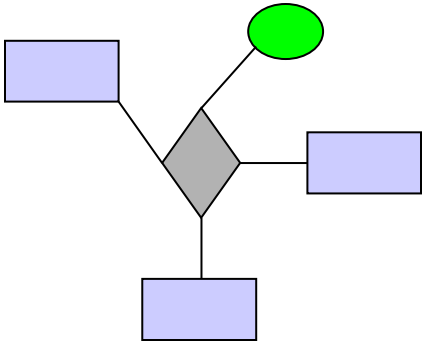
Entity sets are weak when (some or all of) their key attributes come from other entity sets to which they are related.

This happens when:

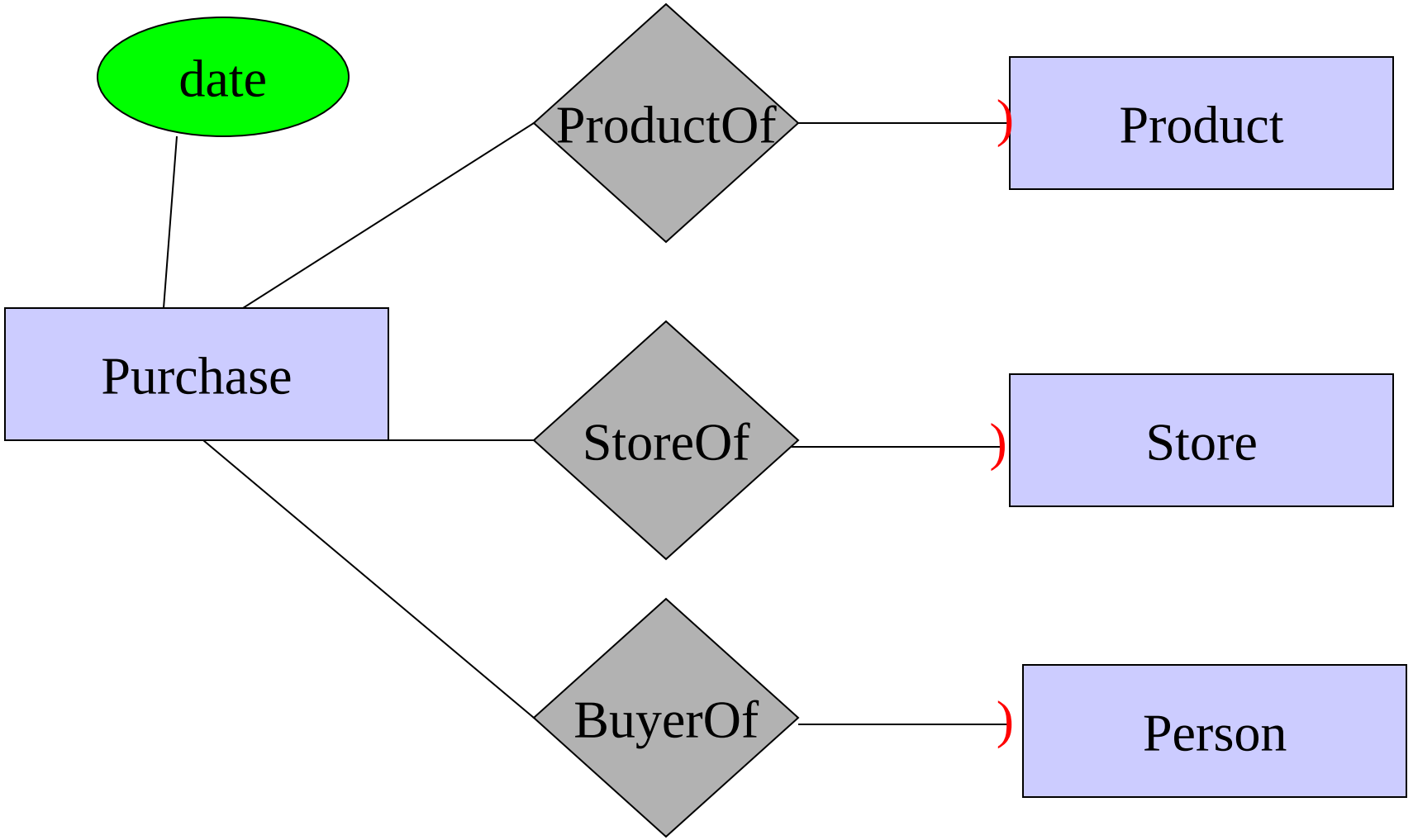
Player(univ, team, number)
University(name, addr)

- part-of relationships
- splitting n-ary relationships to binary.



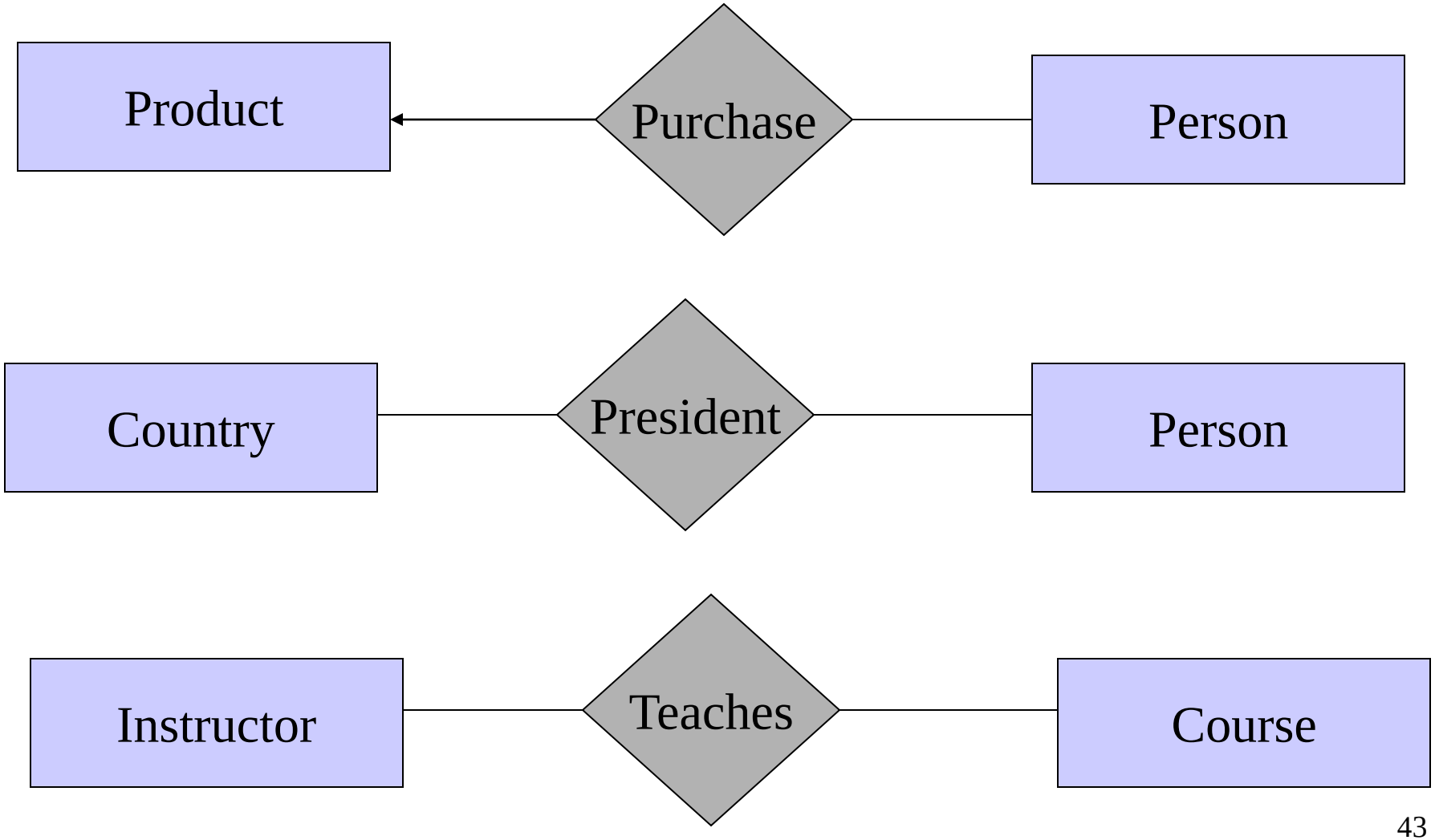


Converting Multiway Relationships to Binary

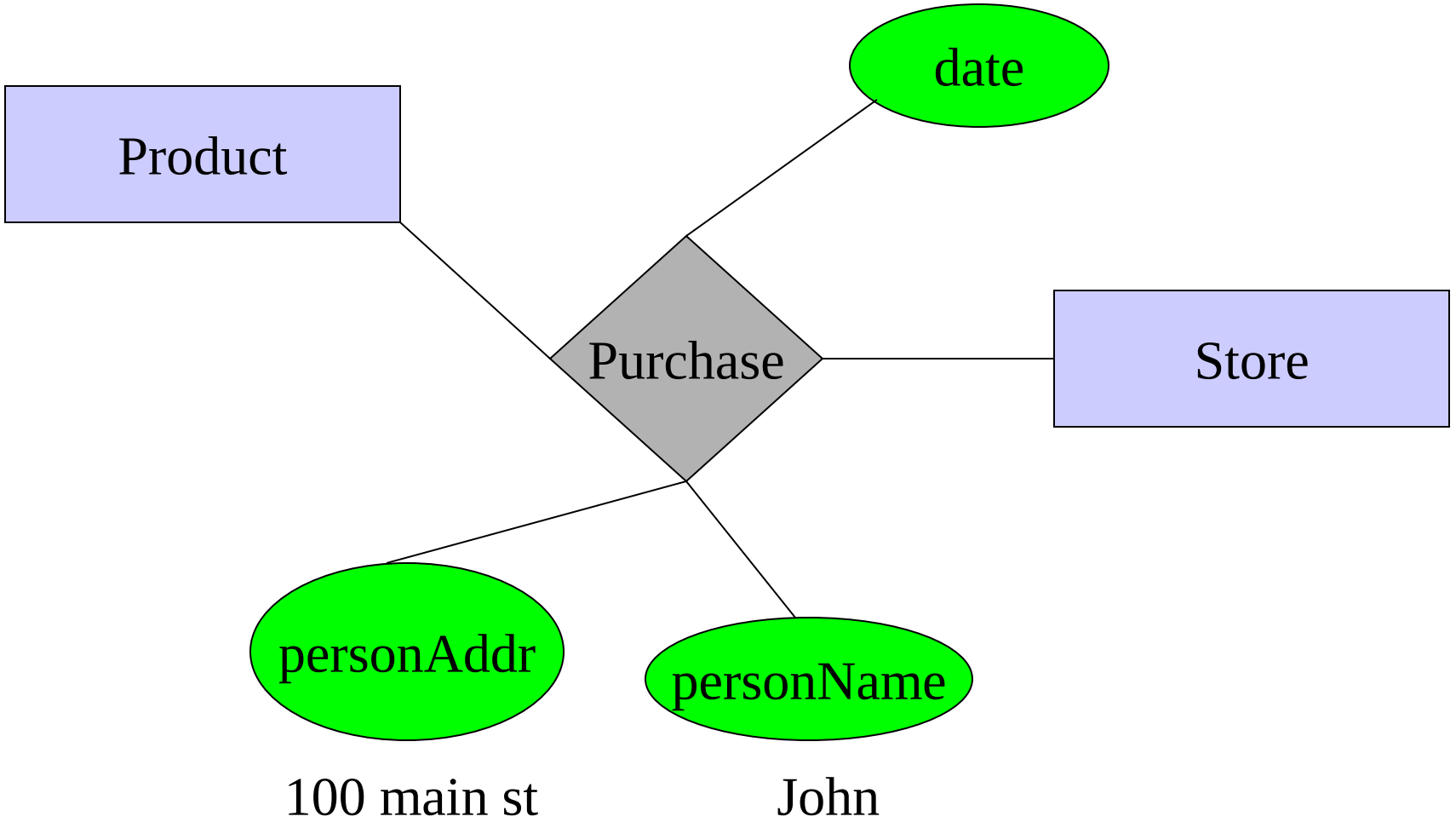


Now, about design techniques ...

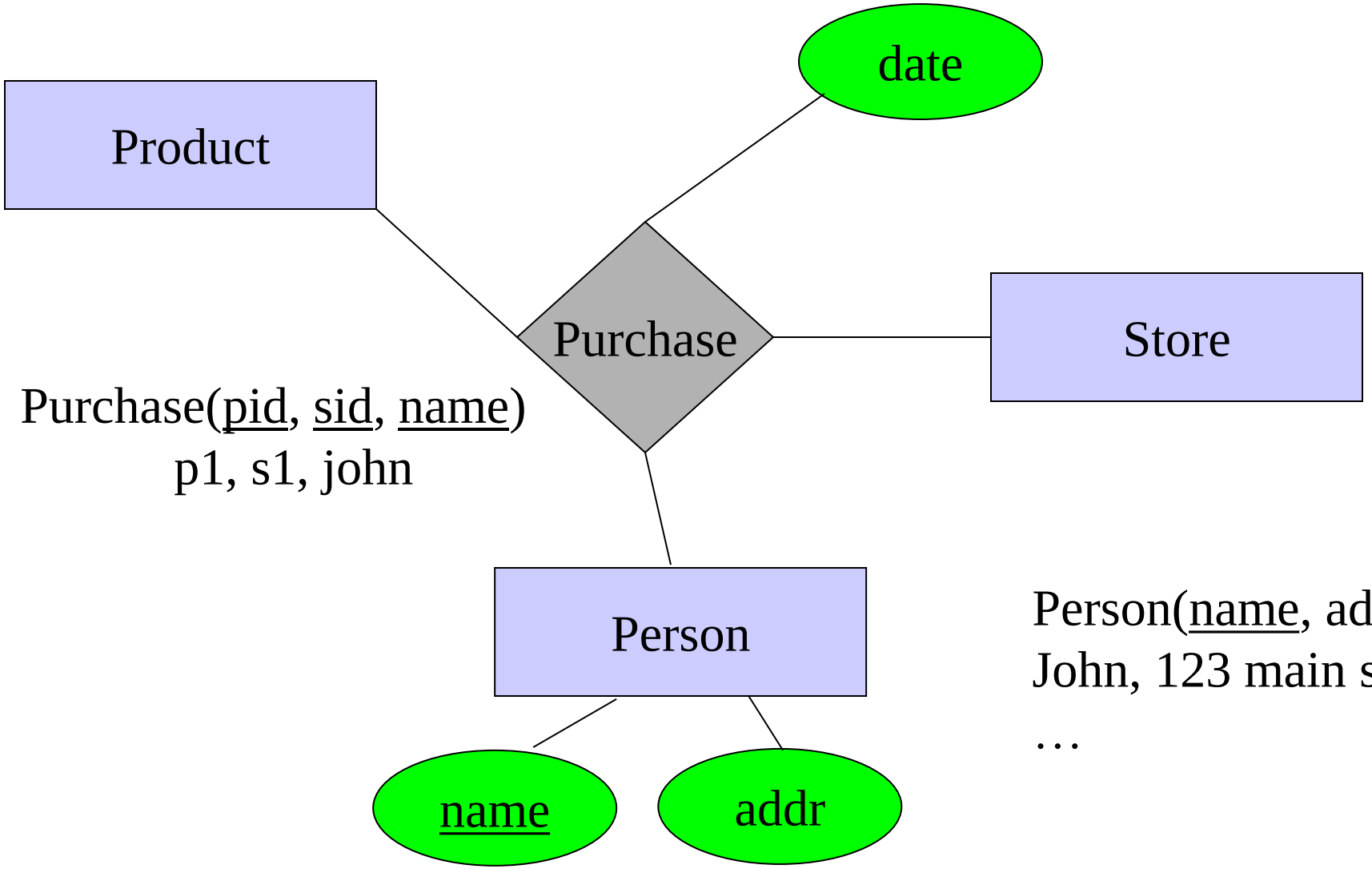
Design Principle 1: Be Faithful



Design Principle 2: Avoid Redundancy



Modified

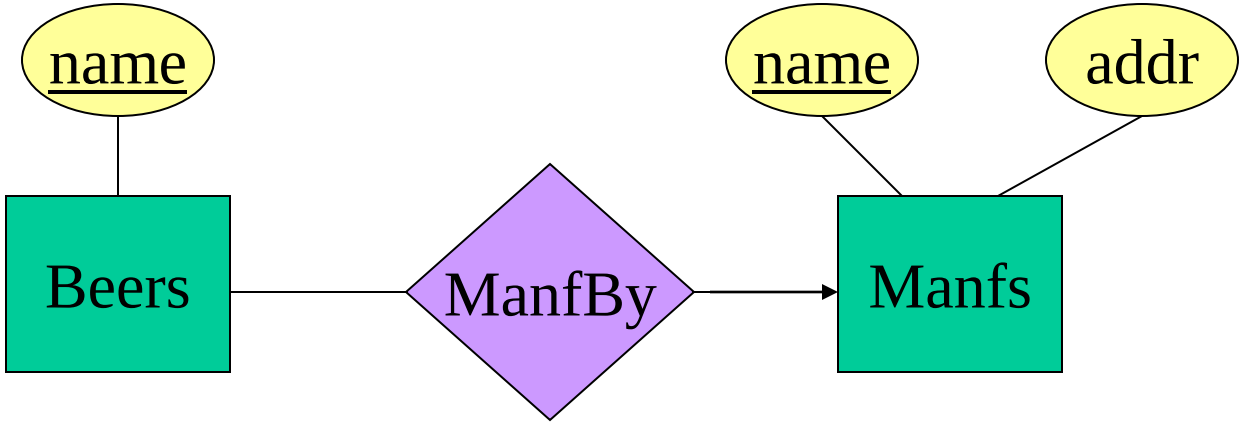

date
Product
Purchase
Store
Purchase(pid, sid, name)
p1, s1, john
Person
Person(name, addr)
John, 123 main st.
…
name
addr

Avoiding Redundancy

- Redundancy occurs when we say the same thing in more than one way.

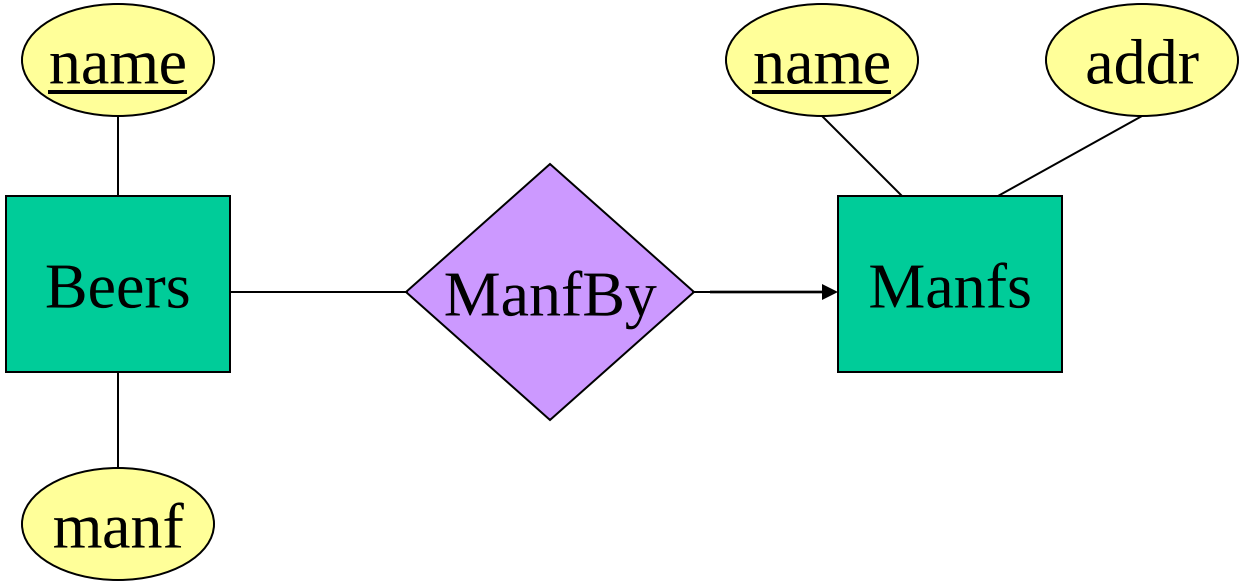
- Redundancy wastes space and (more importantly) encourages inconsistency.
  - Multiple instances of the same fact may become inconsistent if we change one and forget to change the other, related version.

Example: Good



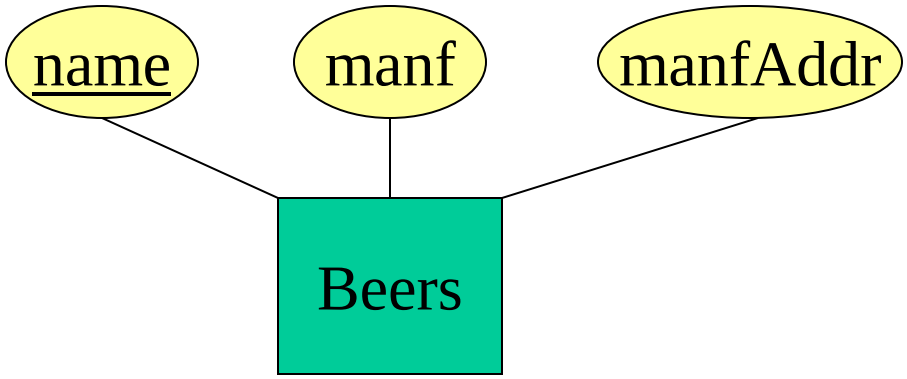
This design gives the address of each manufacturer exactly once.

Example: Bad



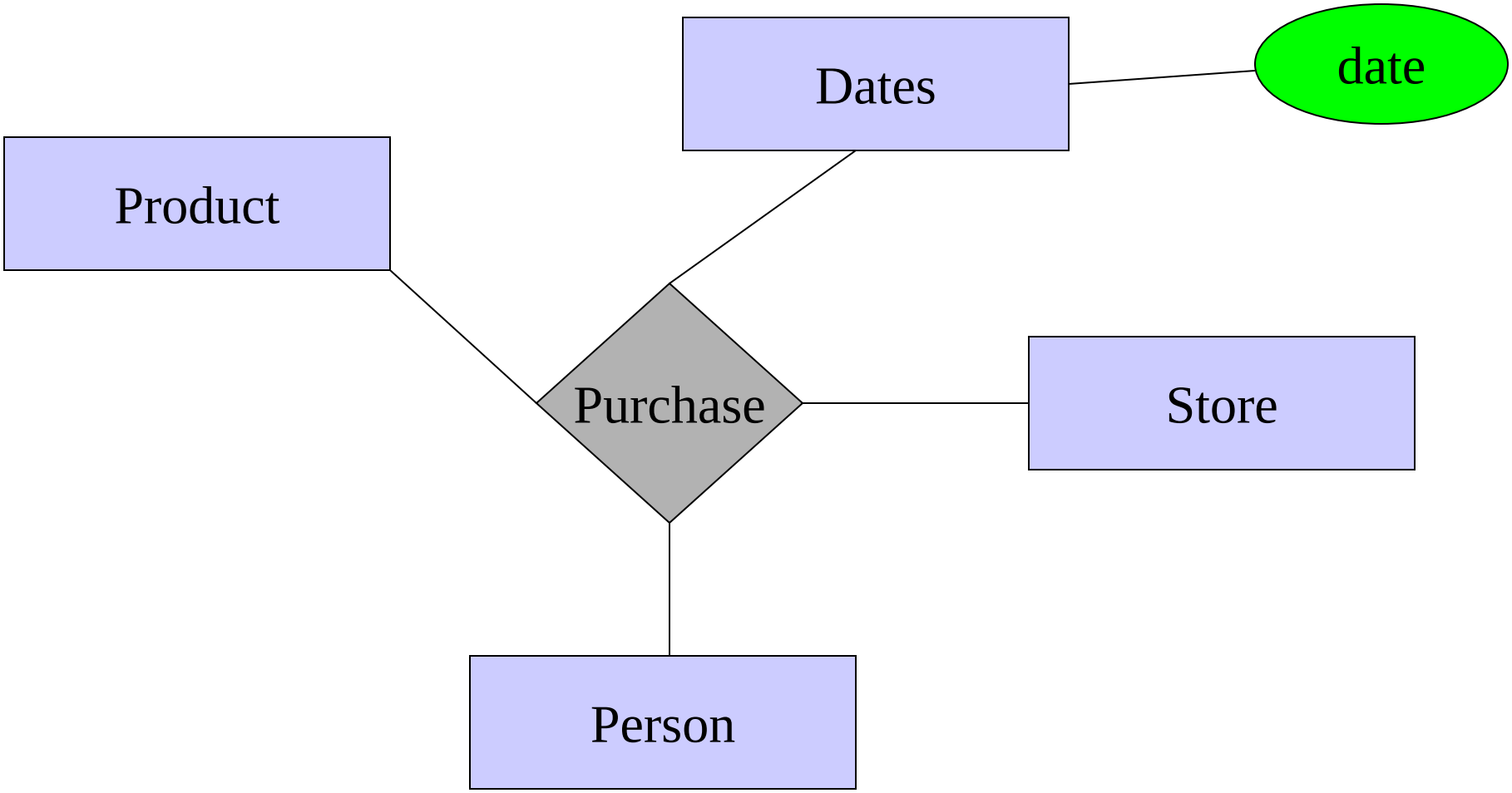
This design states the manufacturer of a beer twice: as an attribute and as a related entity.

Example: Bad



This design repeats the manufacturer's address once for each beer; loses the address if there are temporarily no beers for a manufacturer.

Design Principle 3: KISS



More on Design Techniques

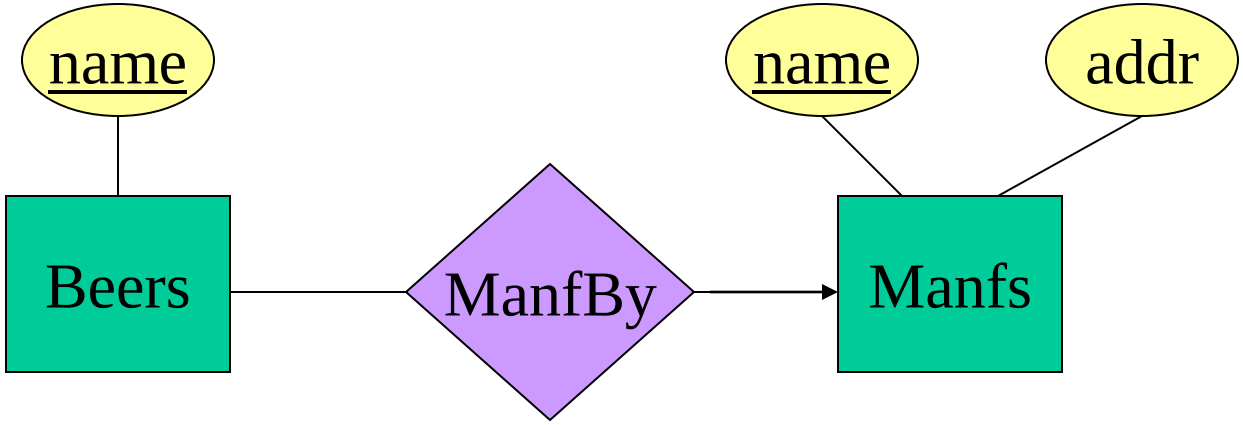
1. Don't use an entity set when an attribute will do.
2. Limit the use of weak entity sets.
    1. Vin for cars

Entity Sets Versus Attributes

- An entity set should satisfy at least one of the following conditions:
  - It is more than the name of something; it has at least one nonkey attribute.

    or
  - It is the "many" in a many-one or many-many relationship. (why?)

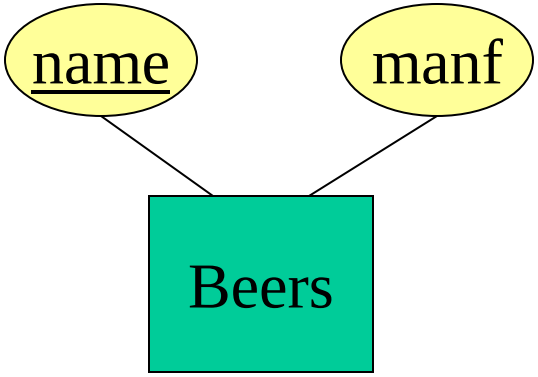
Example: Good



- *Manfs* deserves to be an entity set because of the nonkey attribute *addr*.

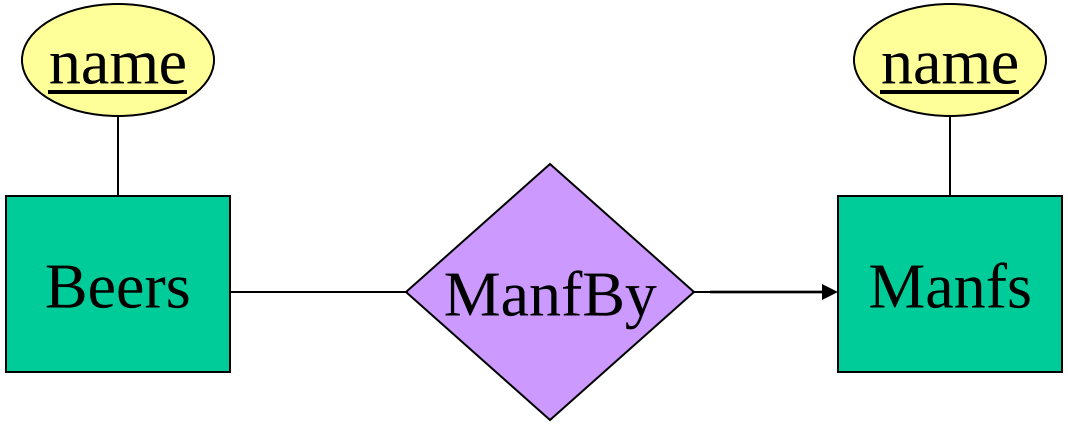
- *Beers* deserves to be an entity set because it is the "many" of the many-one relationship *ManfBy*.

Example: Good



There is no need to make the manufacturer an entity set, if we record nothing about manufacturers besides their name.

Example: Bad



Since the manufacturer is nothing but a name, and is not at the "many" end of any relationship, it should not be an entity set.

Don't Overuse Weak Entity Sets

- Beginning database designers often doubt that anything could be a key by itself.
  - They make all entity sets weak, supported by all other entity sets to which they are linked.
- In reality, we usually can create unique ID's for entity sets.
  - Examples include social-security numbers, automobile VIN's etc.

When Do We Need Weak Entity Sets?

- The usual reason is that there is no global authority capable of creating unique ID's.
- Example: it is unlikely that there could be an agreement to assign unique player numbers across all football teams in the world.

ER Review

- Basic stuff
  - entity, attribute, entity set
  - relationship: binary, multiway, converting from multiway
  - relationship roles, attributes on relationships
- Subclasses (is-a)
- Constraints
  - multiplicity of relationships
    - many-one, one-one, many-many
    - limitations of arrows
  - keys, single-valued, ref integrity, domain & general constraints

ER Review

- Weak entity set

- Design principles
  - be faithful
  - avoid redundancy (entity set vs attributes)
  - KISS