Lecture 02: Conceptual Design, Normal Forms

Tuesday, April 7, 2009

Outline

- Chapter 2: Database design
- Chapter 19: Normal forms

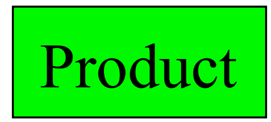
Note: slides for Lecture 1 have been updated. Please reprint.

Database Design

- Requirements analysis
  - Discussions with user groups
- Conceptual database design
  - E/R model
- Logical Database design
  - Database normalization

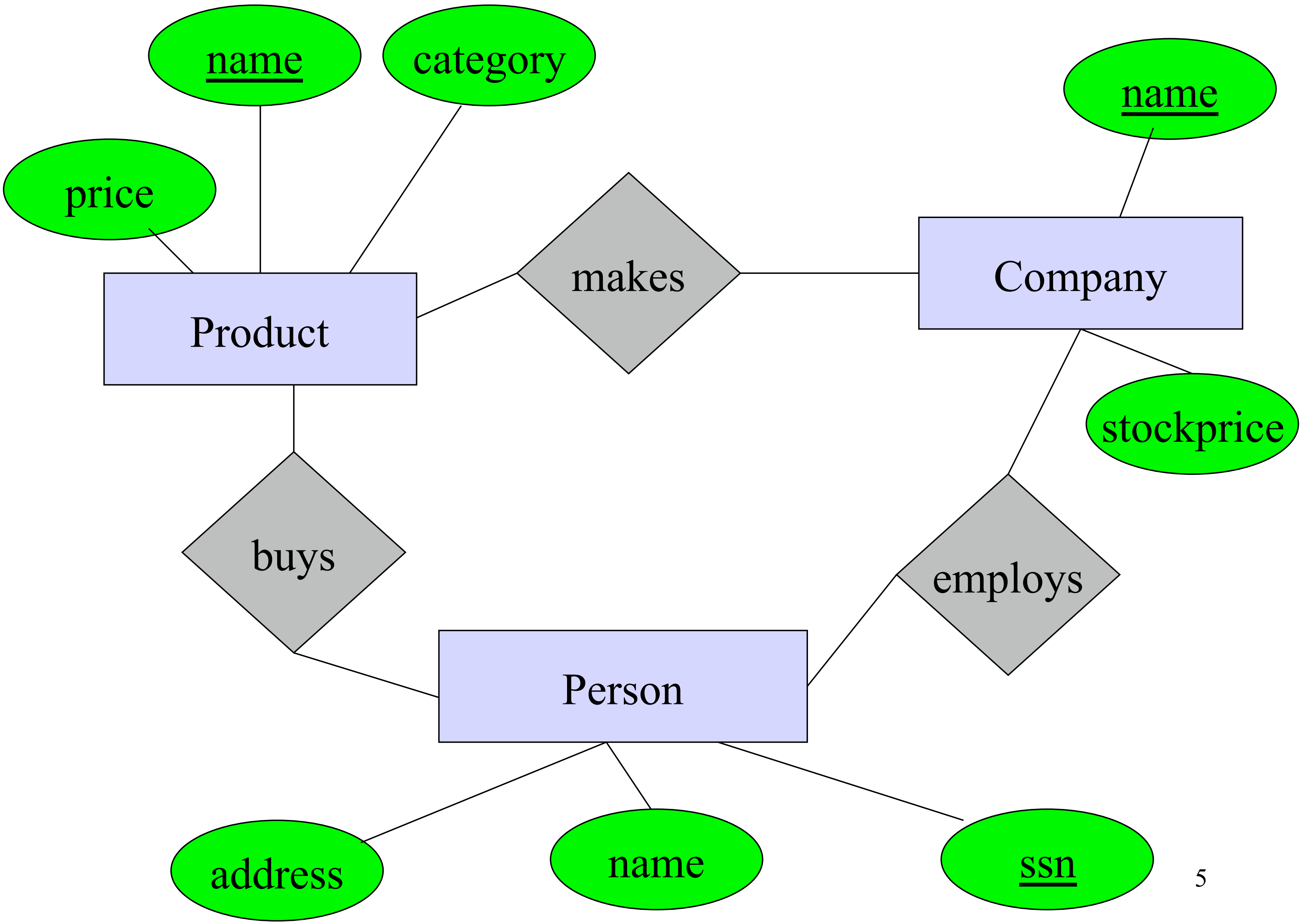
Entity / Relationship Diagrams

- Entities:



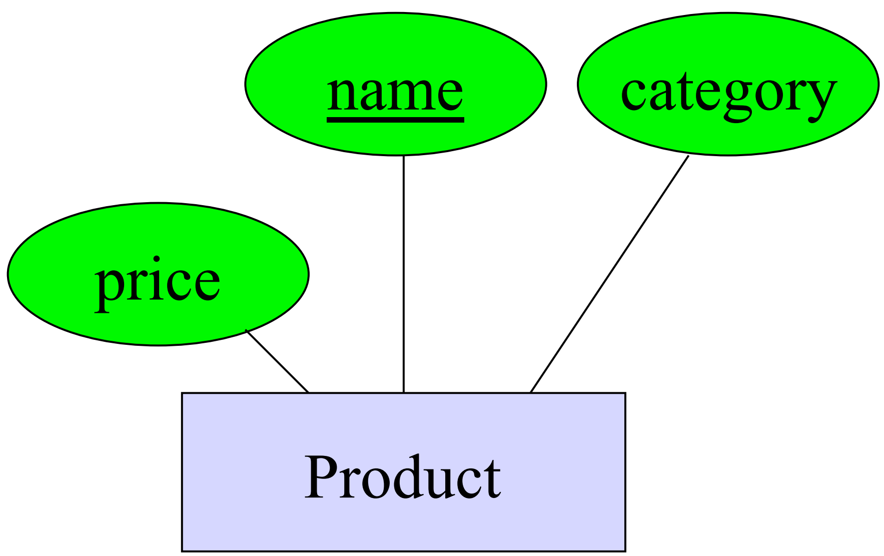
- Attributes:

- Relationships:


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

Keys in E/R Diagrams

- Every entity set must have a key

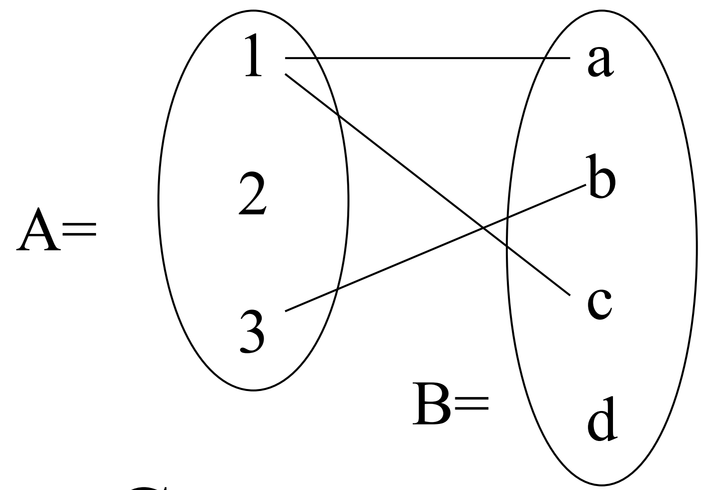


What is a Relation ?

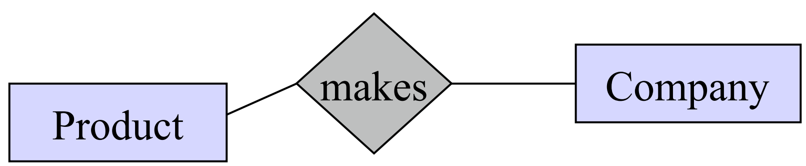
- A mathematical definition:
  - if A, B are sets, then a relation R is a subset of $A \times B$
- $A=\{1,2,3\}$, $B=\{a,b,c,d\}$,

  $A \times B = \{(1,a),(1,b), \ldots, (3,d)\}$

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

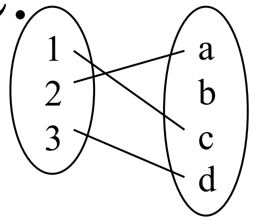


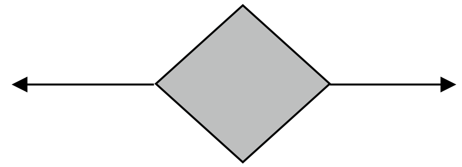
\- **makes** is a subset of **Product × Company**:



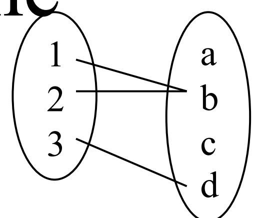
Multiplicity of E/R Relations

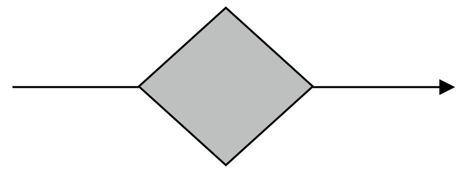
- one-one:



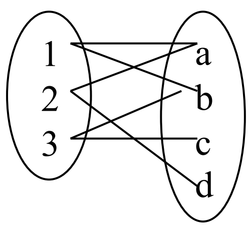


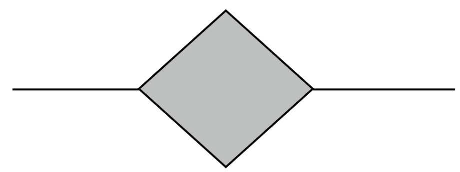
- many-one





- many-many

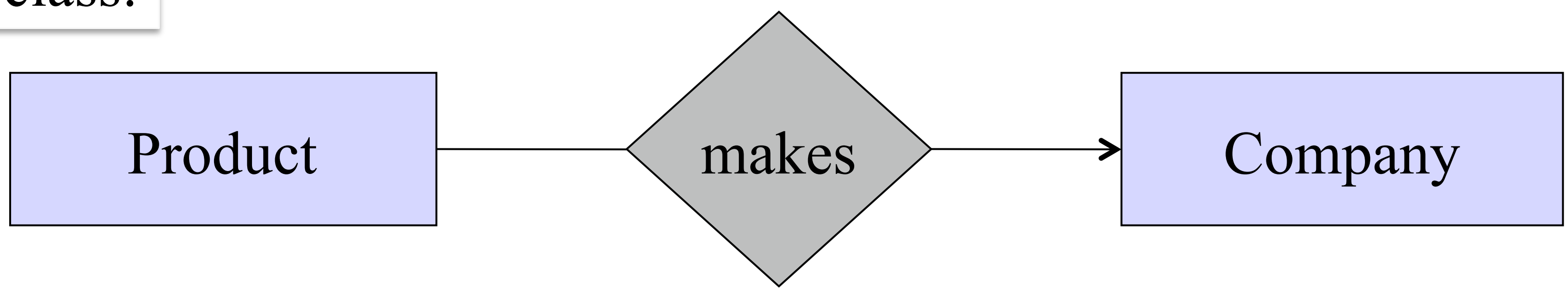




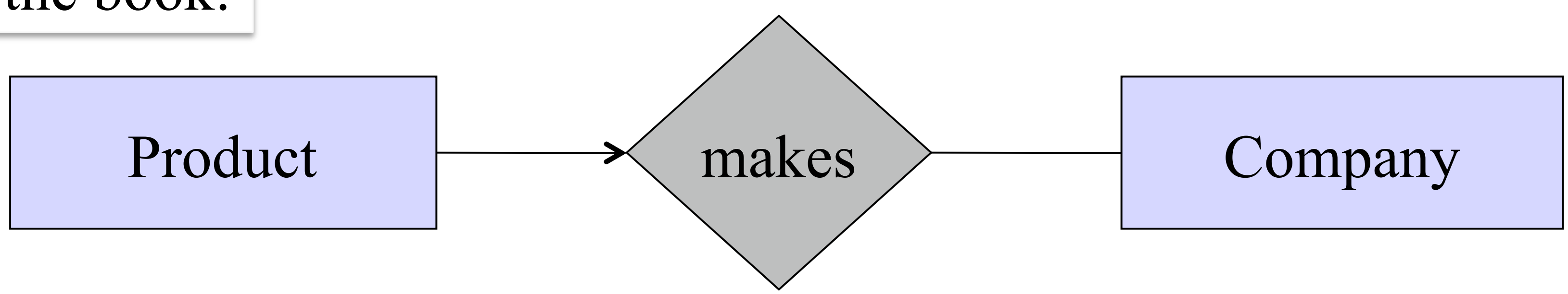
Note: “many-one” actually means “many-[zero-or-one]”

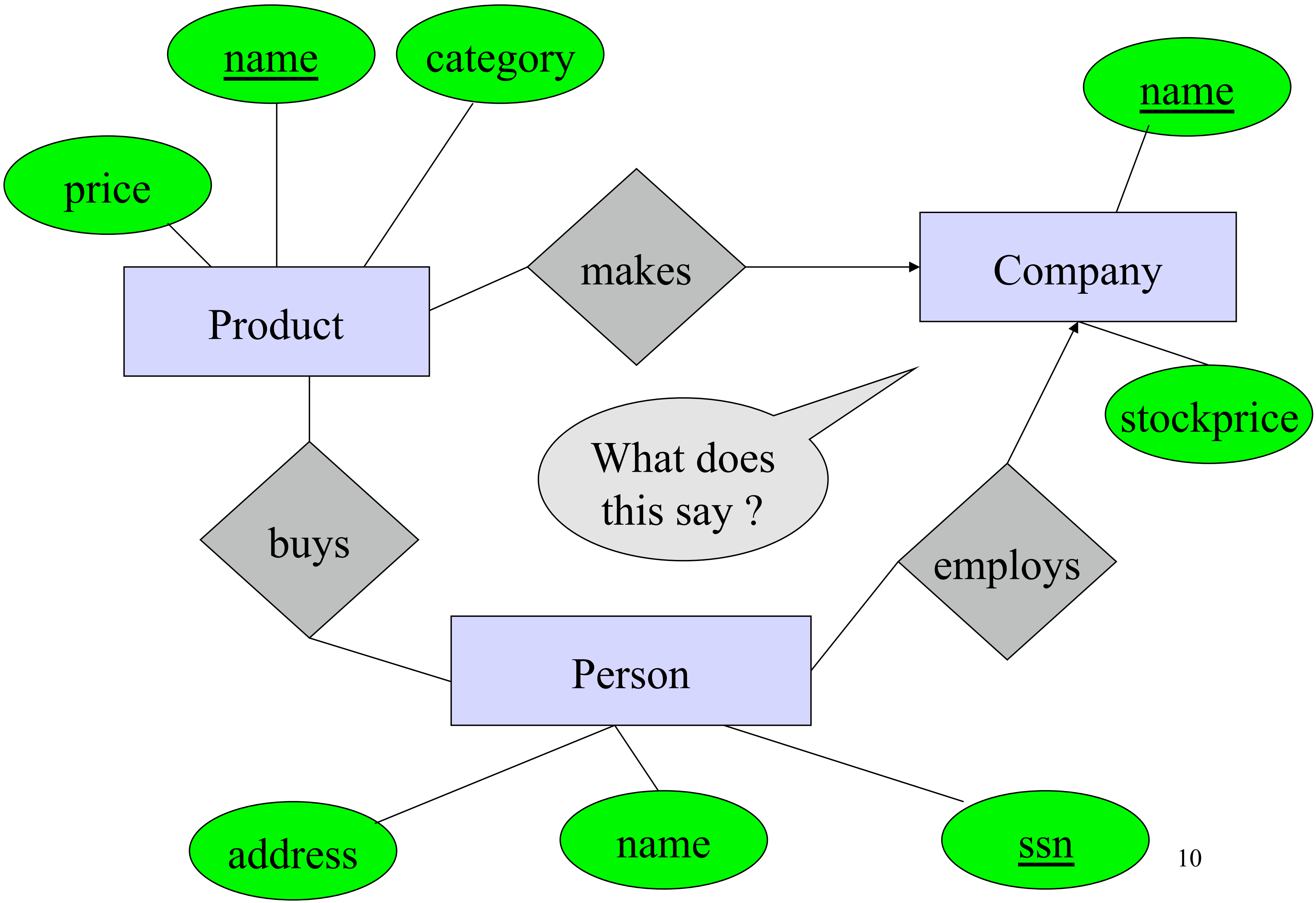
Notation in Class v.s. the Book

In class:



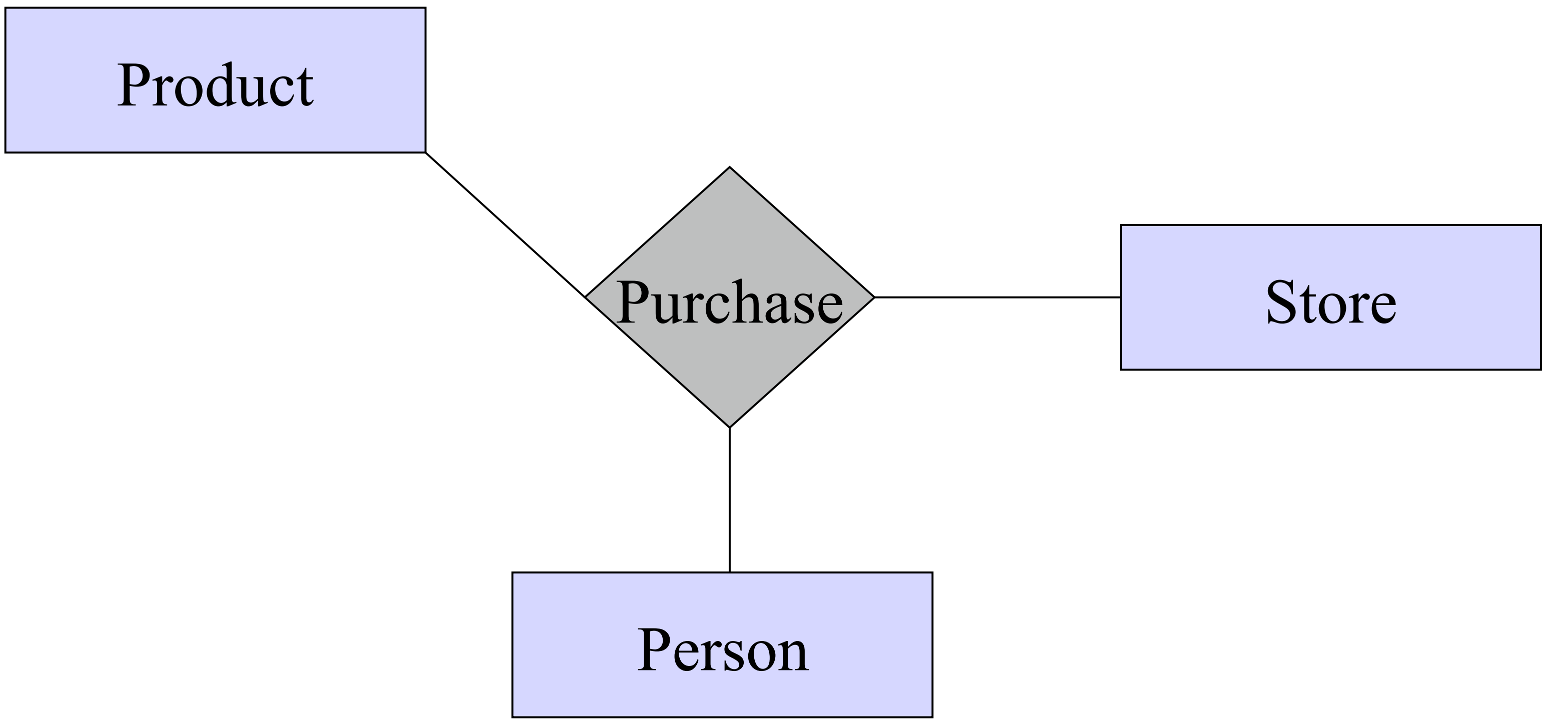
In the book:




name
category
price
name
makes
Company
Product
What does
this say ?
stockprice
buys
employs
Person
address
name
ssn

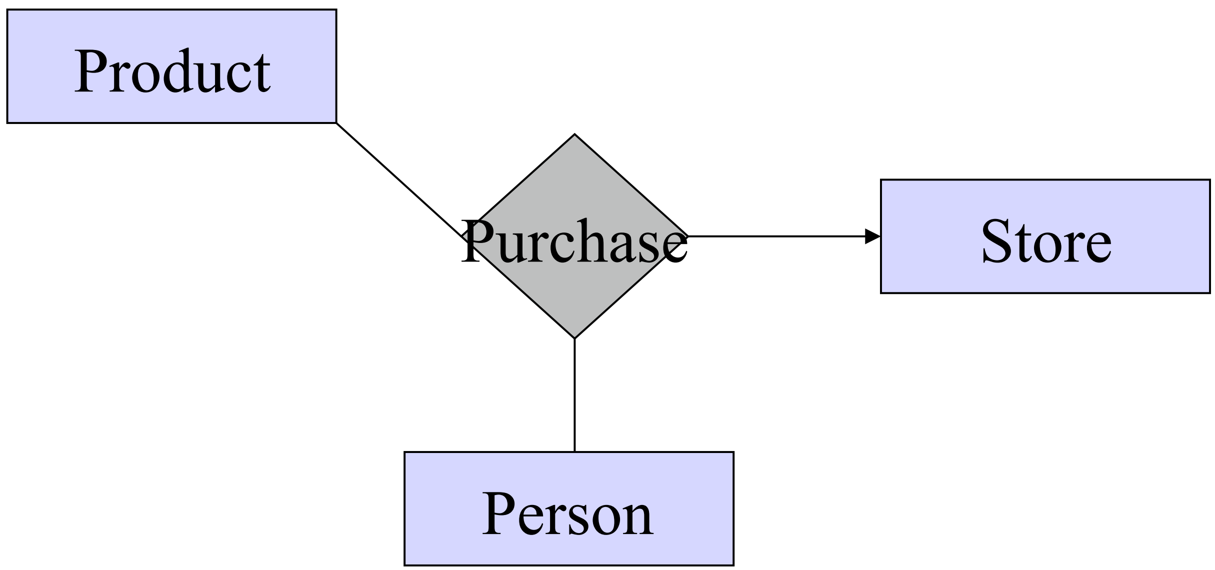
Multi-way Relationships

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



Arrows in Multiway Relationships

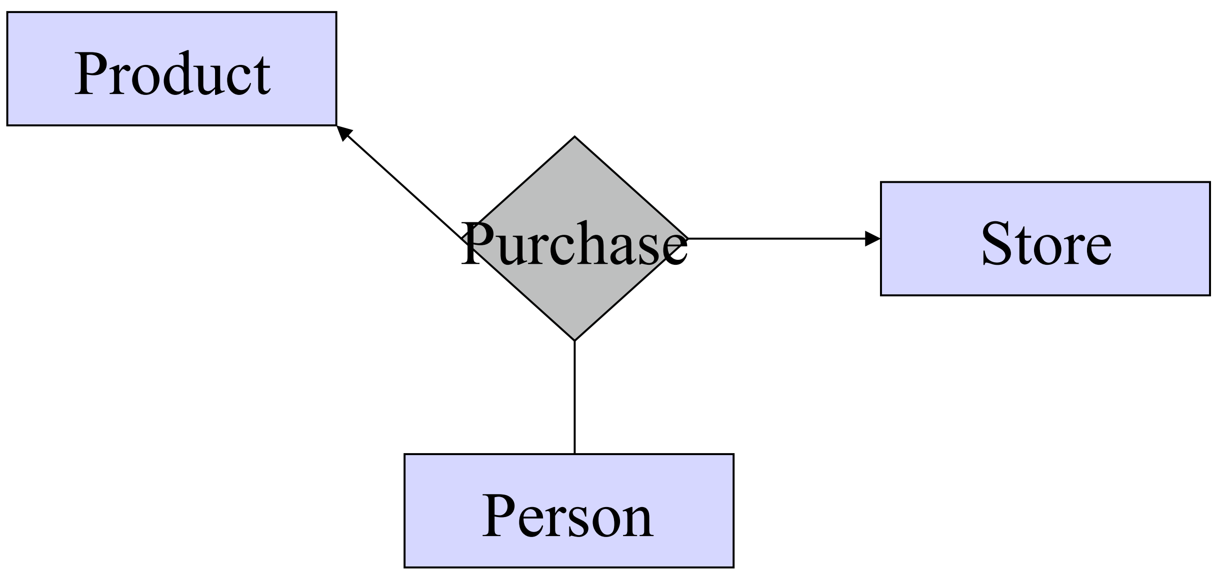
Q: what does the arrow mean ?



A: a given person buys a given product from at most one store

Arrows in Multiway Relationships

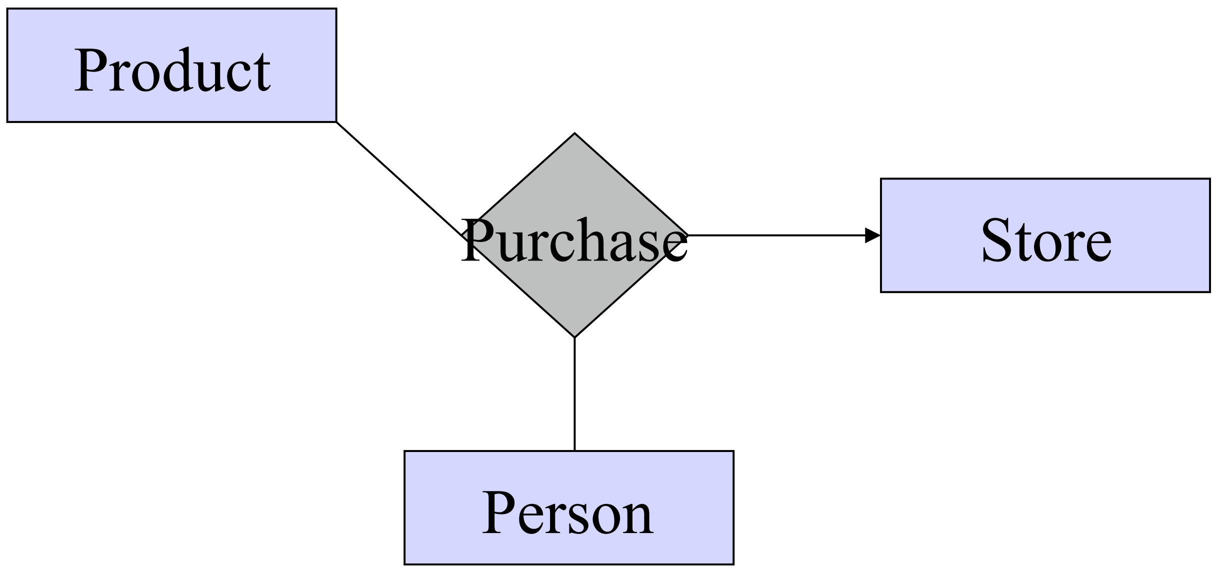
Q: what does the arrow mean ?



A: a given person buys a given product from at most one store AND every store sells to every person at most one product

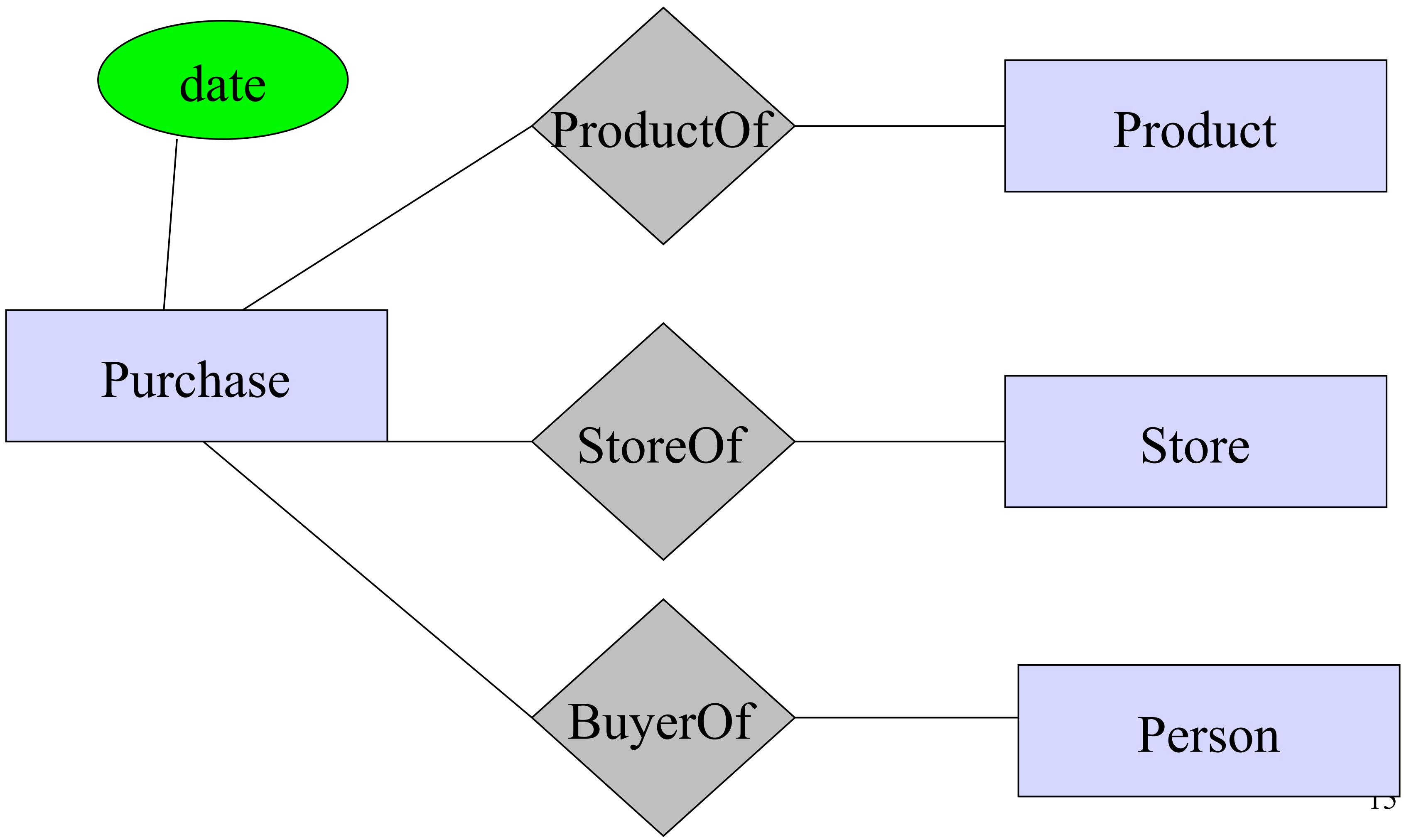
Arrows in Multiway Relationships

Q: How do we say that every person shops at at most one store ?



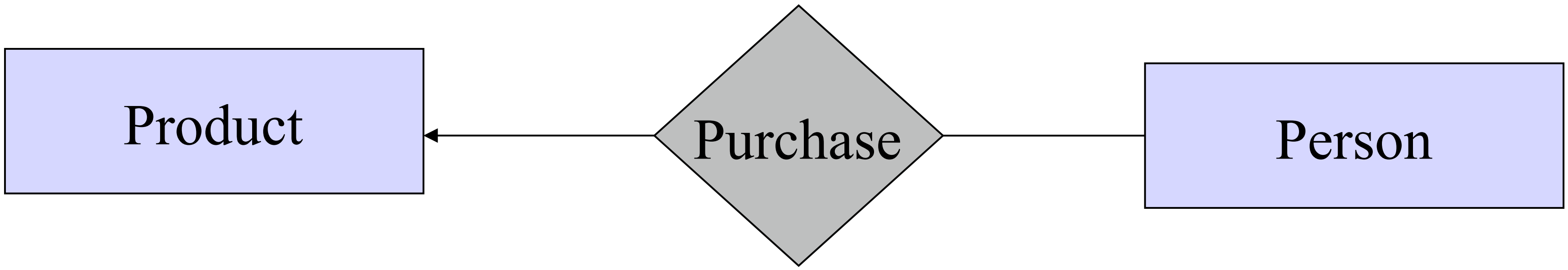
A: cannot. This is the best approximation.
(Why only approximation ?)

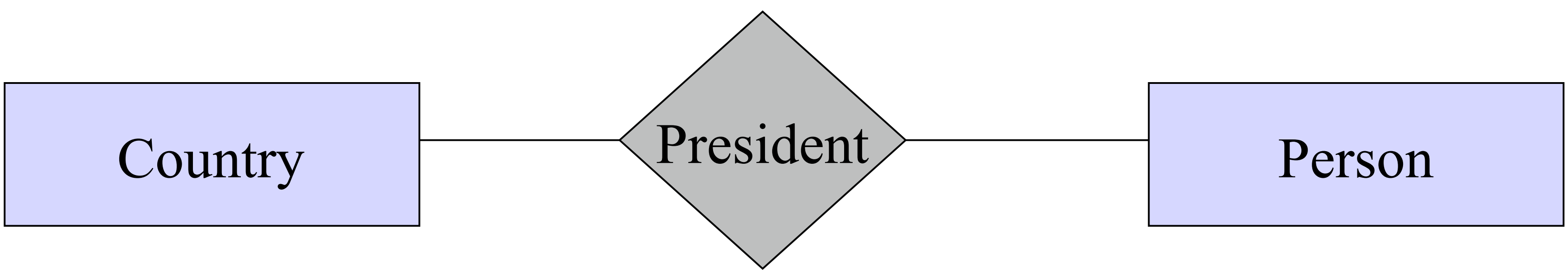
Reification: Multi-way to Binary



3. Design Principles

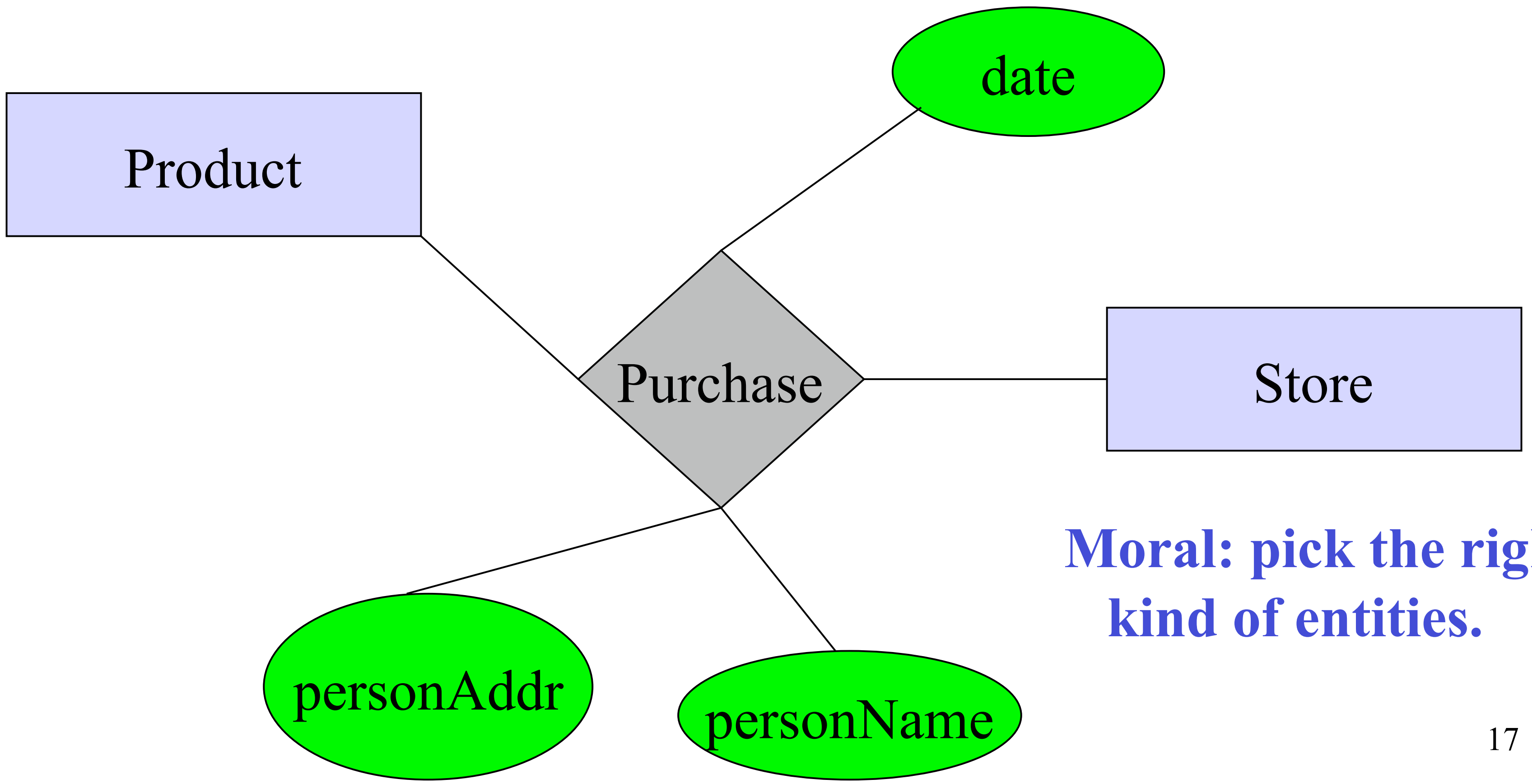
What's wrong?





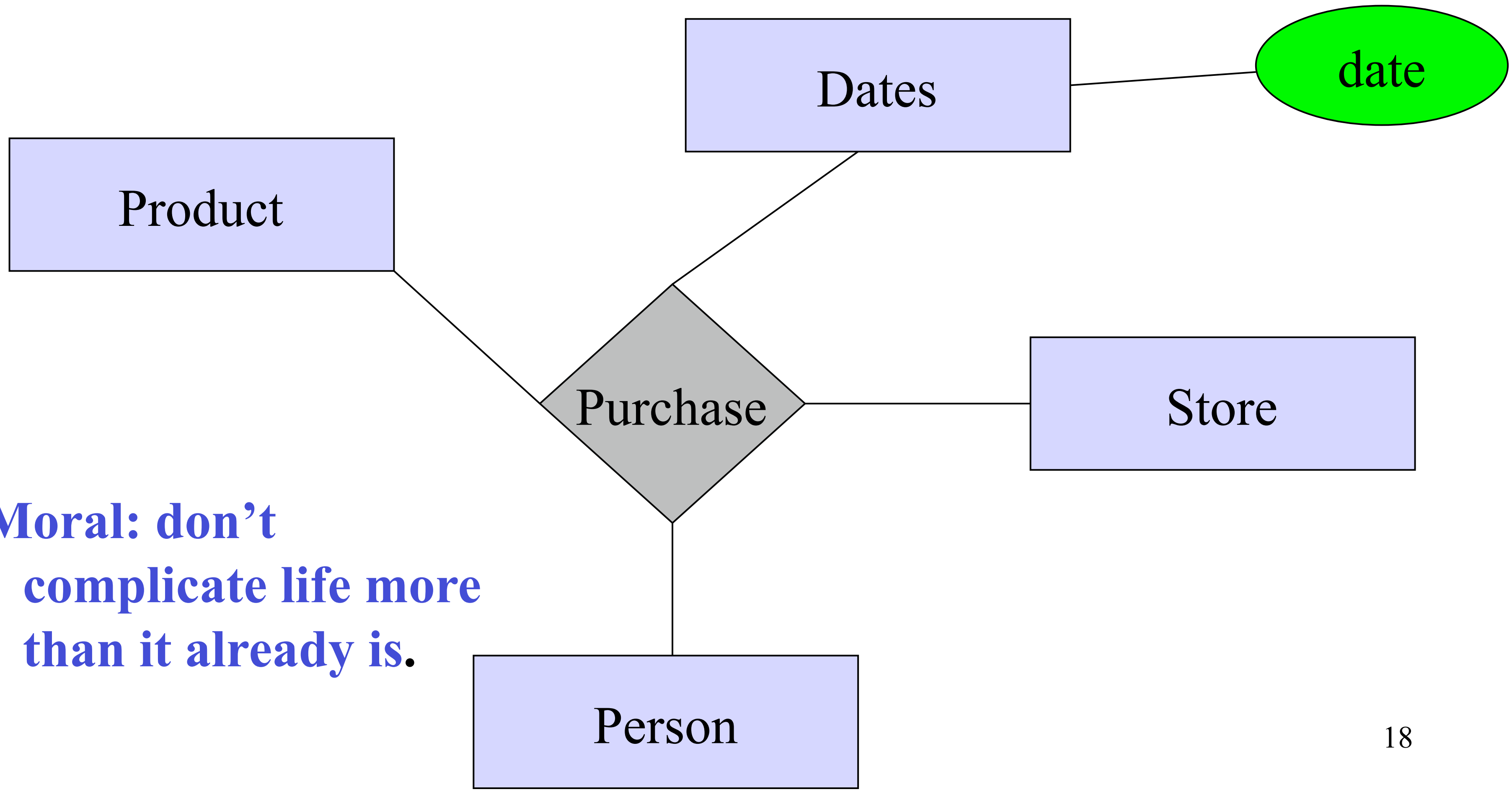
Moral: be faithful!

Design Principles: What's Wrong?



Moral: pick the right kind of entities.

Design Principles: What's Wrong?

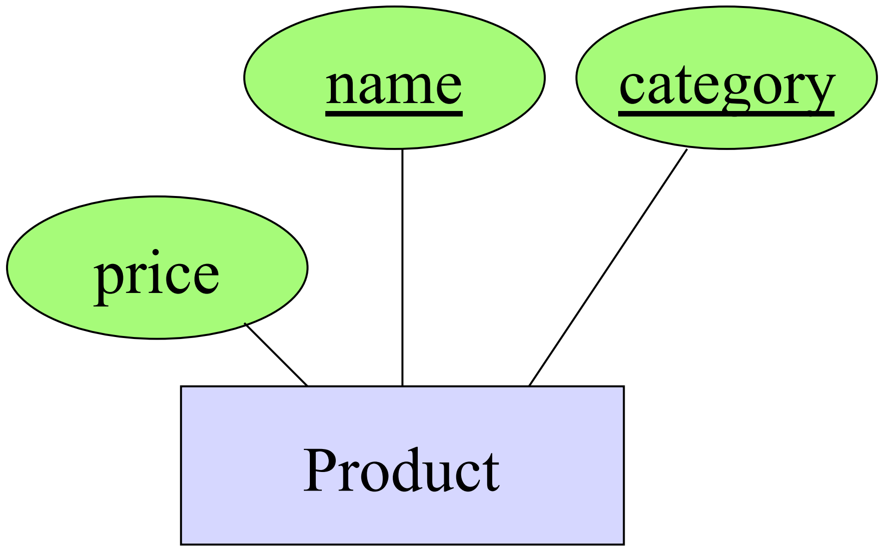


Moral: don't complicate life more than it already is.

From E/R Diagrams to Relational Schema

- Entity set → relation
- Relationship → relation

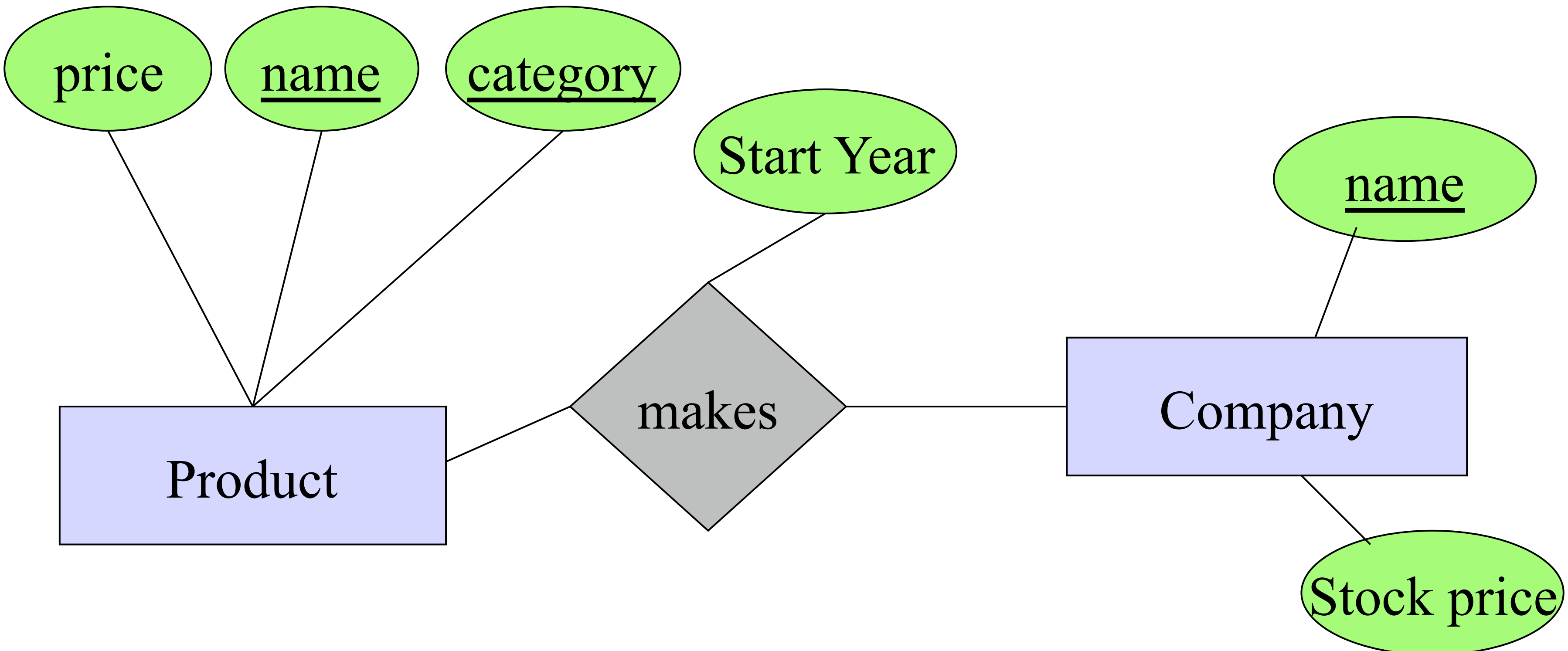
Entity Set to Relation



Product

<u>Name</u>	<u>Category</u>	Price
Gizmo	Gadgets	$19.99

Relationships to Relations

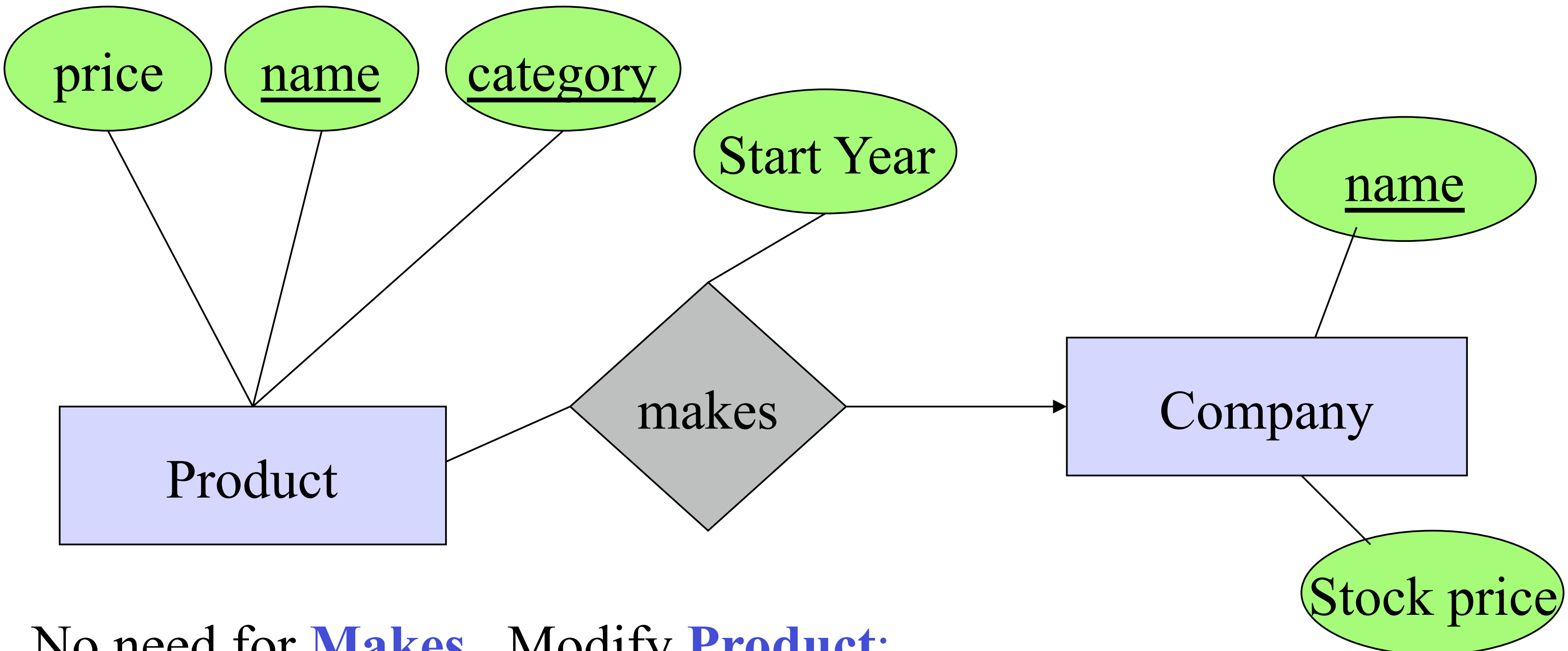


Makes

ProdName	ProdCategory	CompanyName	StartYear
Gizmo	Gadgets	gizmoWorks	1963

(watch out for attribute name conflicts)

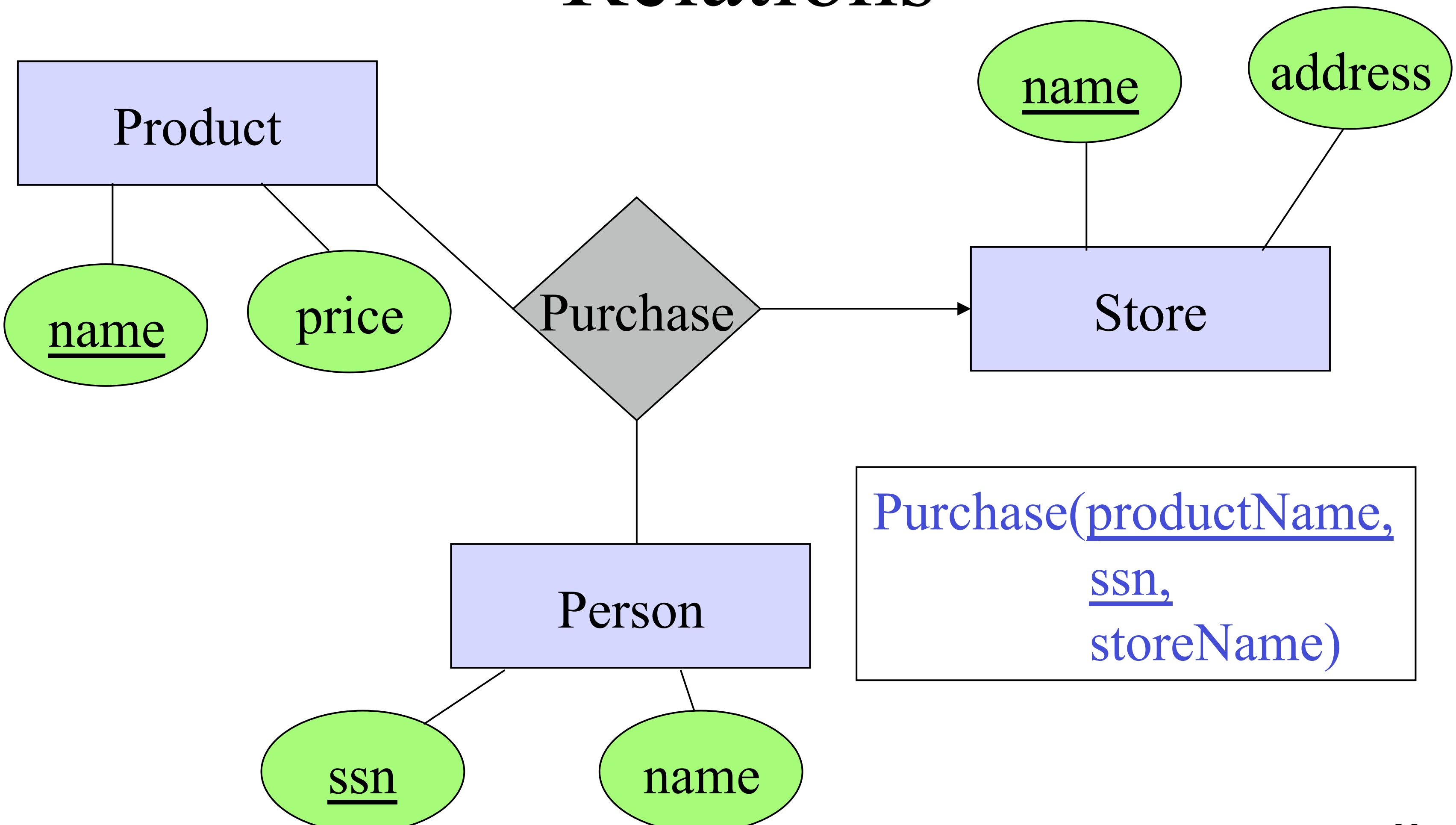
Relationships to Relations



No need for **Makes**. Modify **Product**:

Name	Category	Price	CompanyName	StartYear
Gizmo	Gadgets	$19.99	gizmoWorks	1963

Multi-way Relationships to Relations

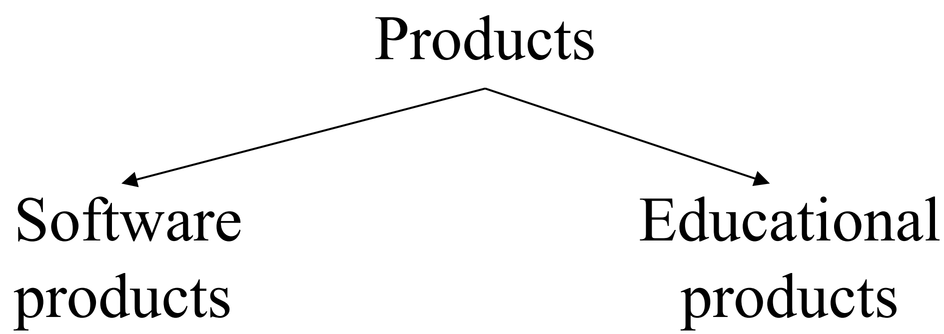


Purchase(productName,
ssn,
storeName)

Modeling Subclasses

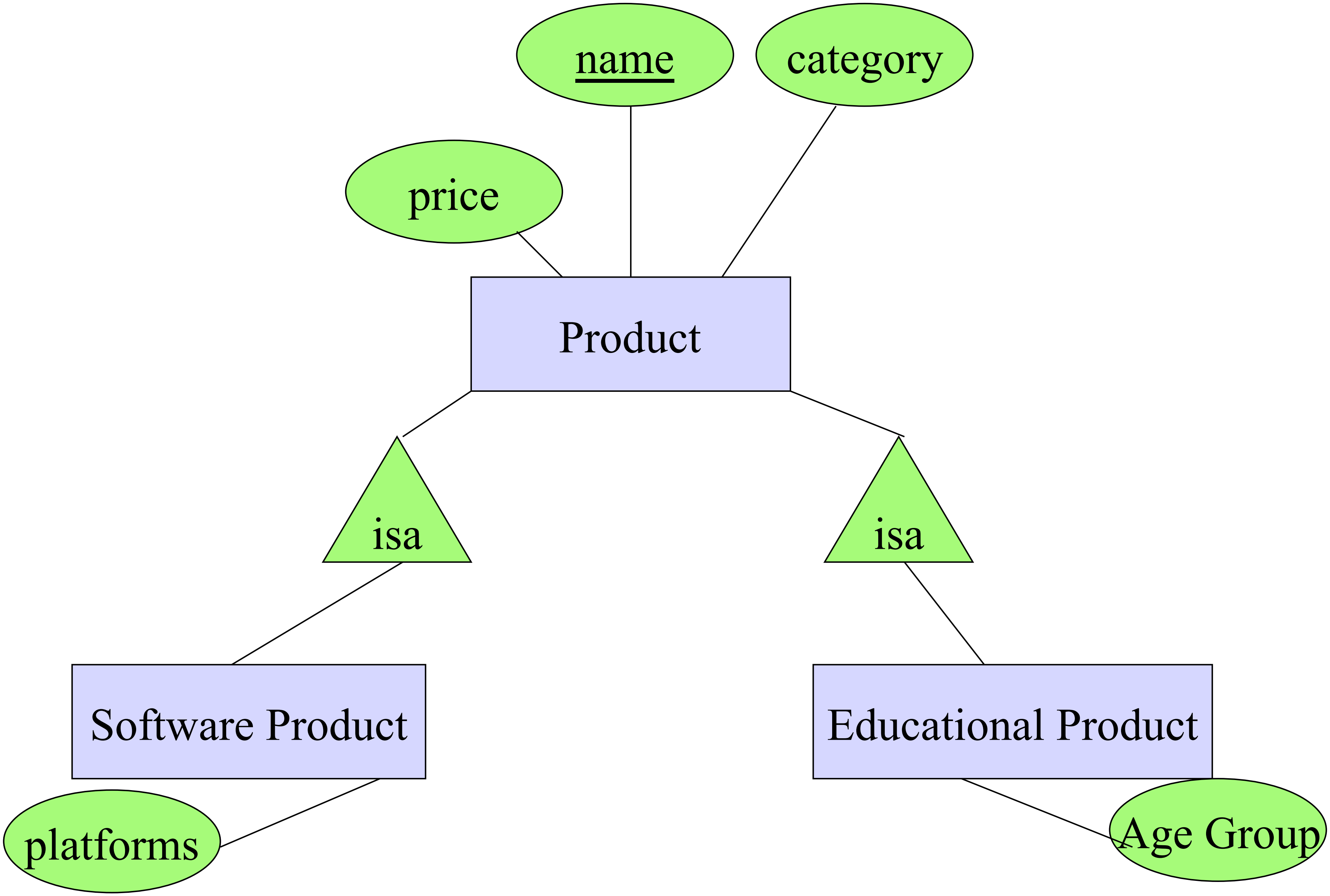
Some objects in a class may be special

- define a new class
- better: define a *subclass*



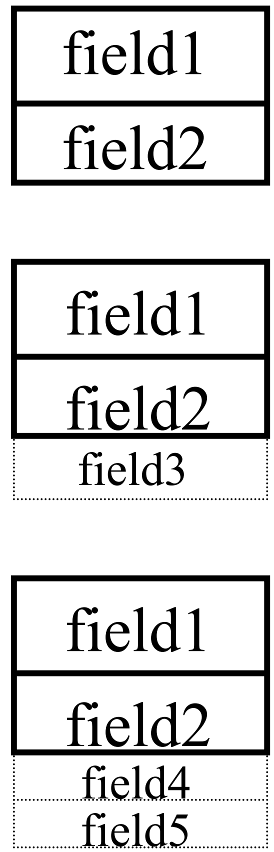
So --- we define subclasses in E/R

Subclasses

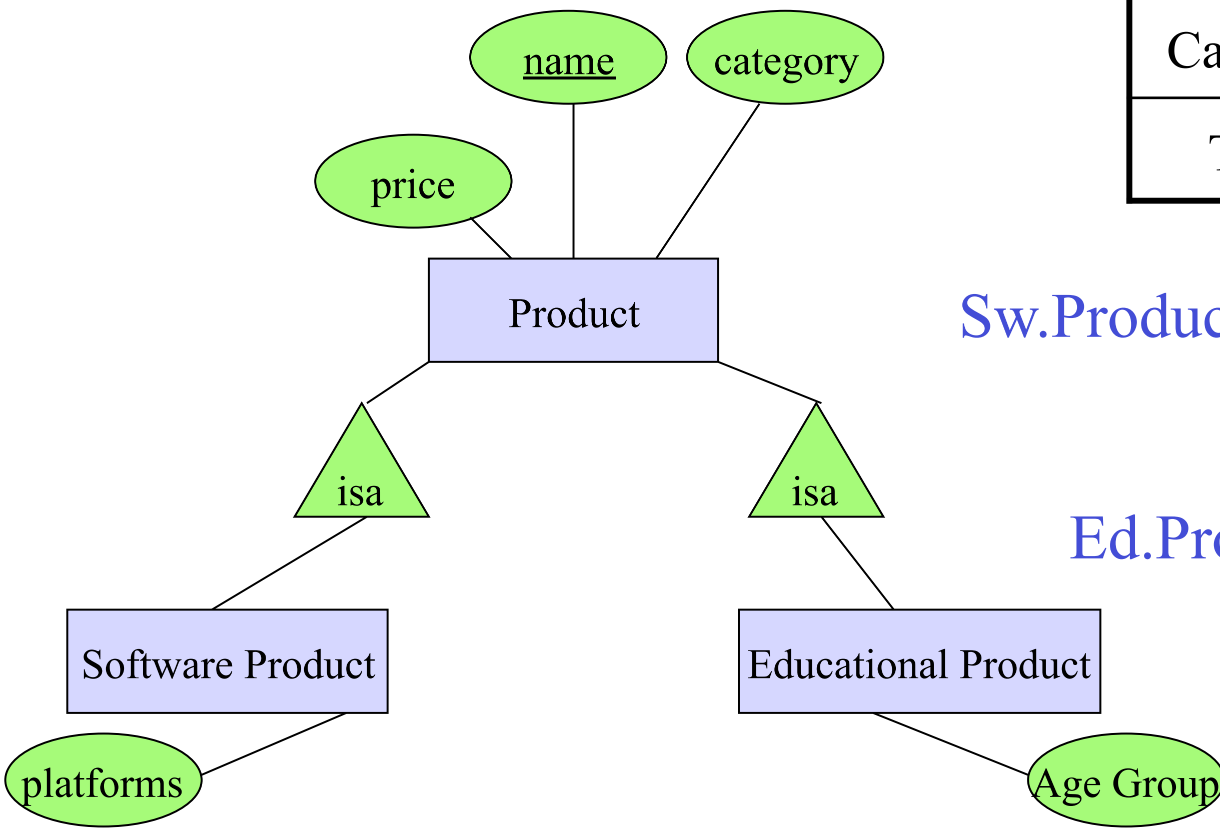

name
category
price
Product
isa
isa
Software Product
Educational Product
platforms
Age Group

Understanding Subclasses

- Think in terms of records:
  - Product
  - SoftwareProduct
  - EducationalProduct



Subclasses to Relations



Product

Name	Price	Category
Gizmo	99	gadget
Camera	49	photo
Toy	39	gadget

Sw.Product

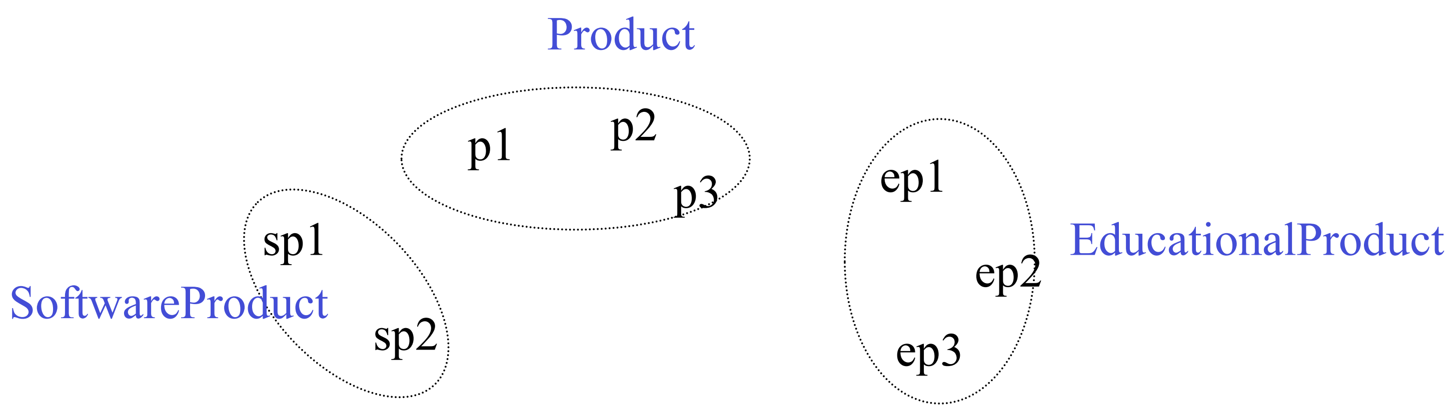
Name	platforms
Gizmo	unix

Ed.Product

Name	Age Group
Gizmo	todler
Toy	retired

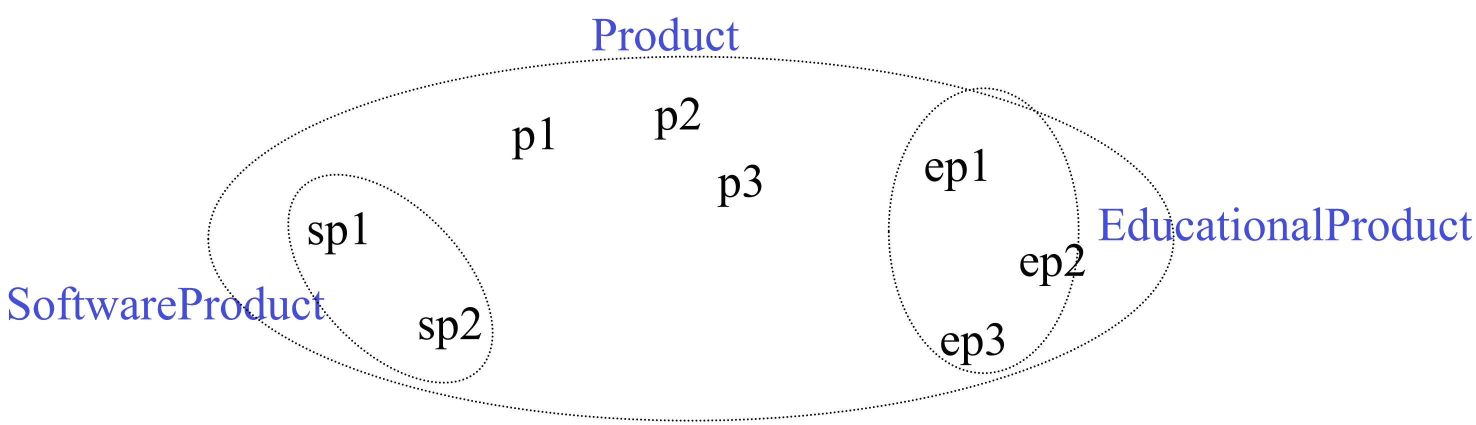
Difference between OO and E/R inheritance

- OO: classes are disjoint (same for Java, C++)

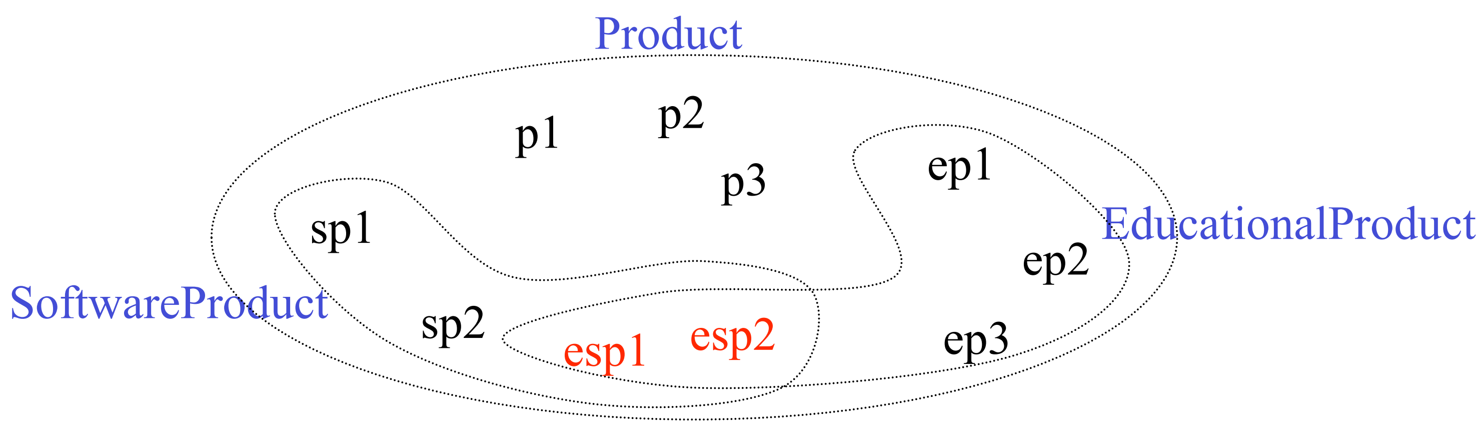


Difference between OO and E/R inheritance

- E/R: entity sets overlap

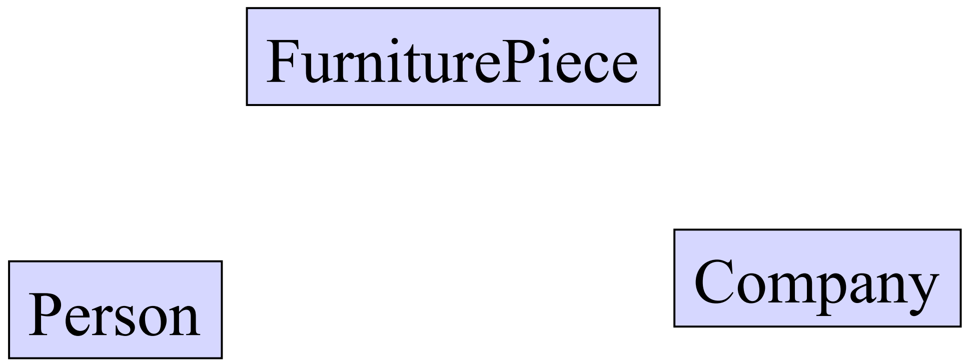


No need for multiple inheritance in E/R



We have three entity sets, but four different kinds of objects.

Modeling UnionTypes With Subclasses

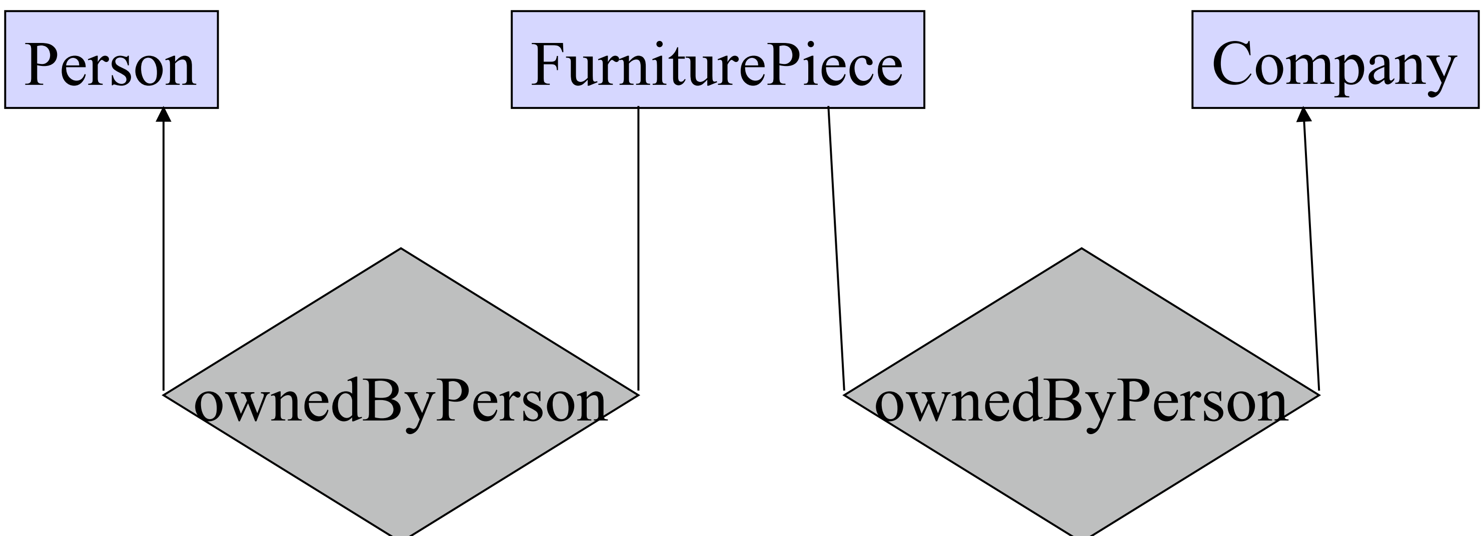


Say: each piece of furniture is owned either by a person, or by a company

Modeling Union Types with Subclasses

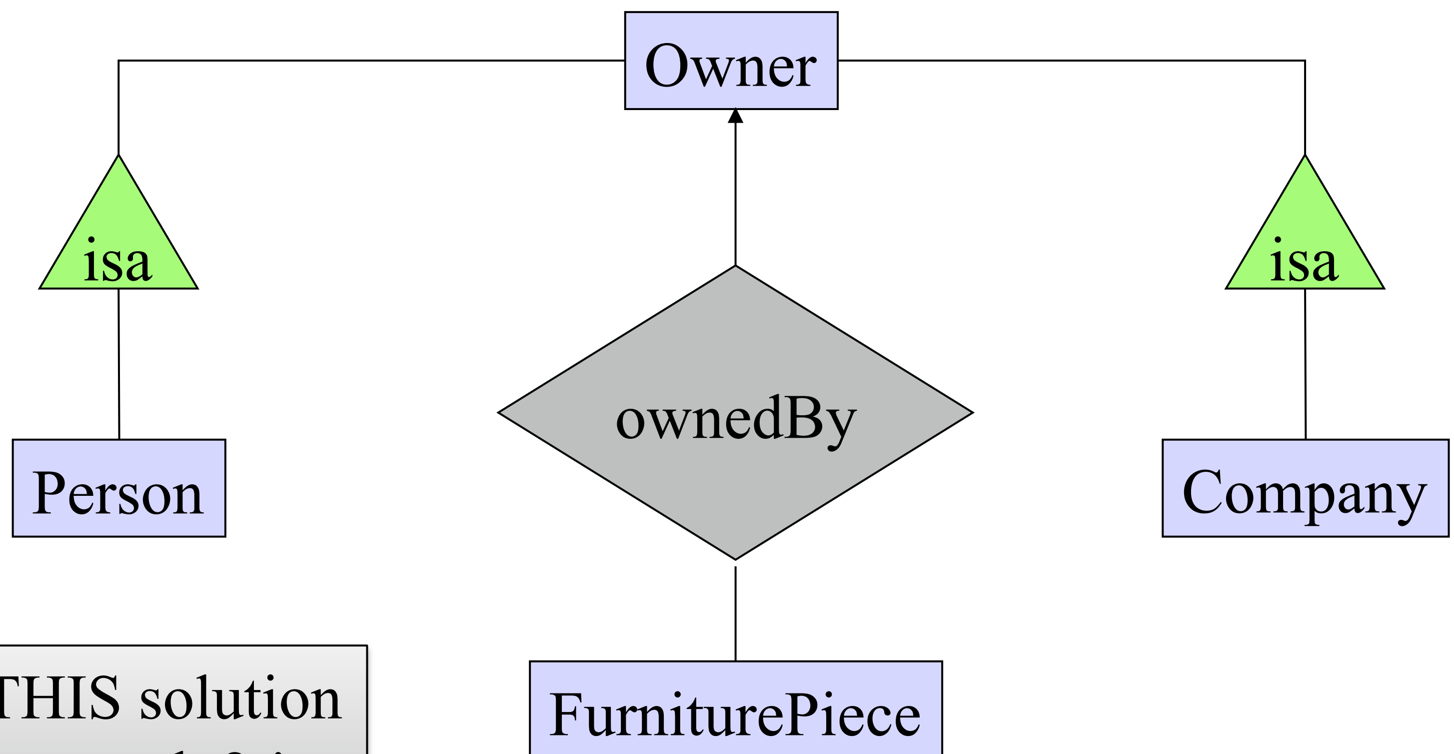
Say: each piece of furniture is owned either by a person, or by a company

Solution 1. Acceptable, imperfect (What's wrong ?)



Modeling Union Types with Subclasses

Solution 2: better, more laborious



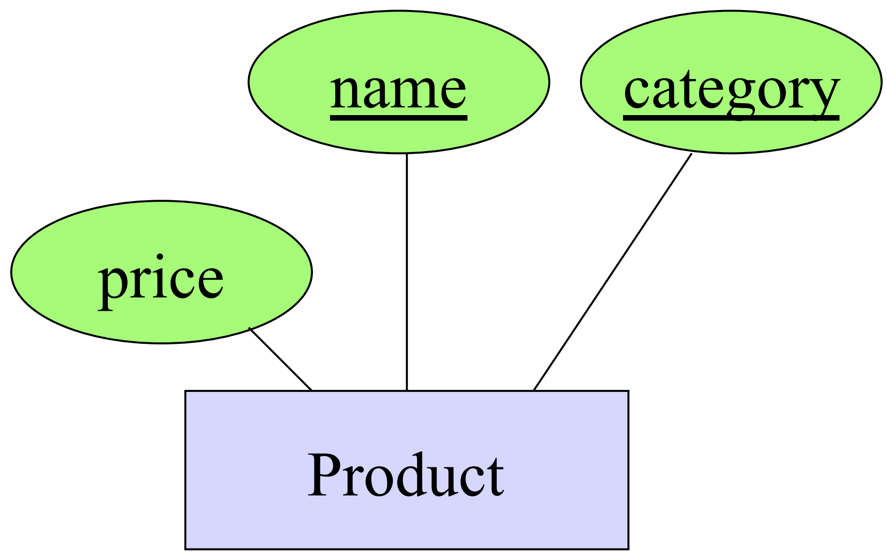
Use THIS solution in homework 2 !

Constraints in E/R Diagrams

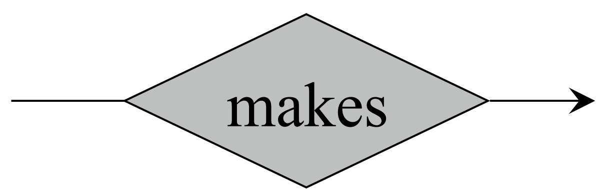
- Key constraints
- Single value constraints
- Referential integrity constraints
- Cardinality constraints

Keys in E/R Diagrams

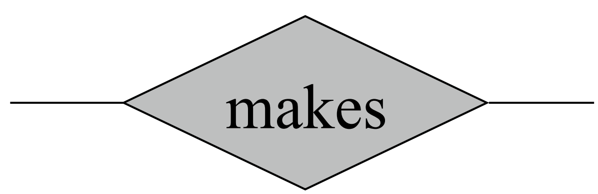
In E/R diagrams each entity set must have exactly one key (consisting of one or more attributes)



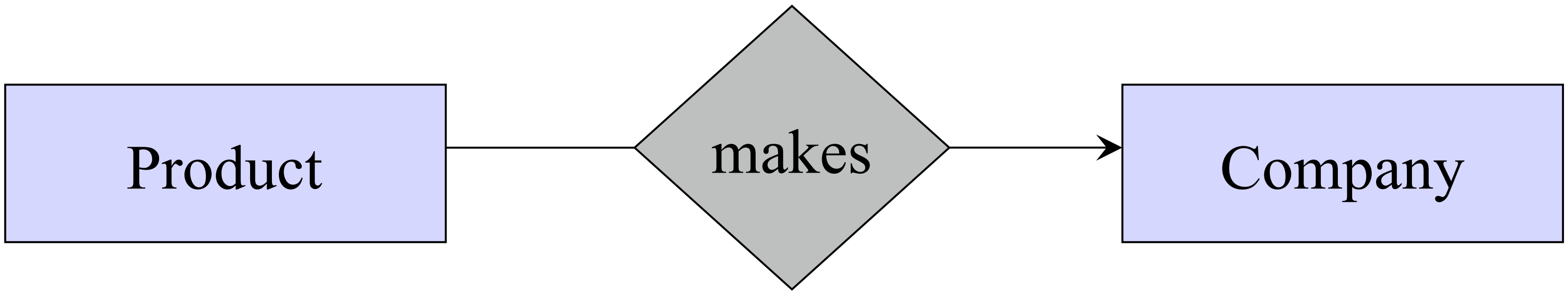
Single Value Constraints



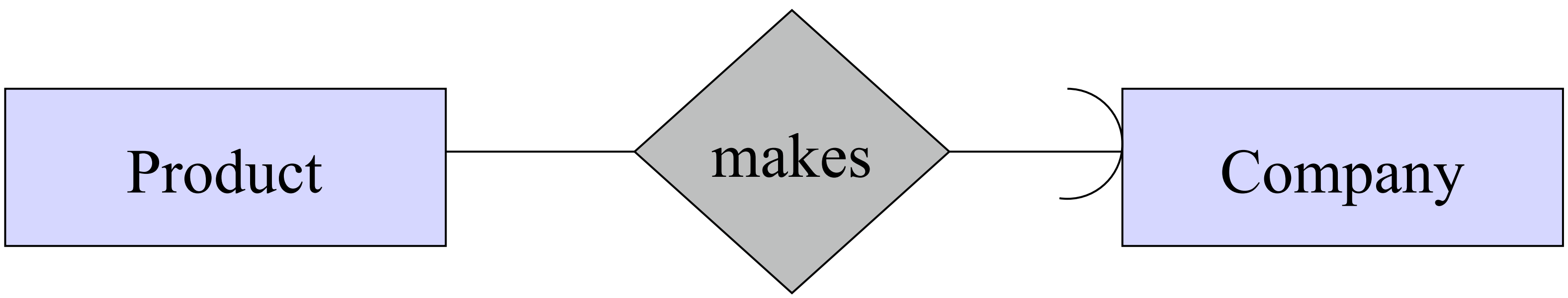
v. s.



Referential Integrity Constraints

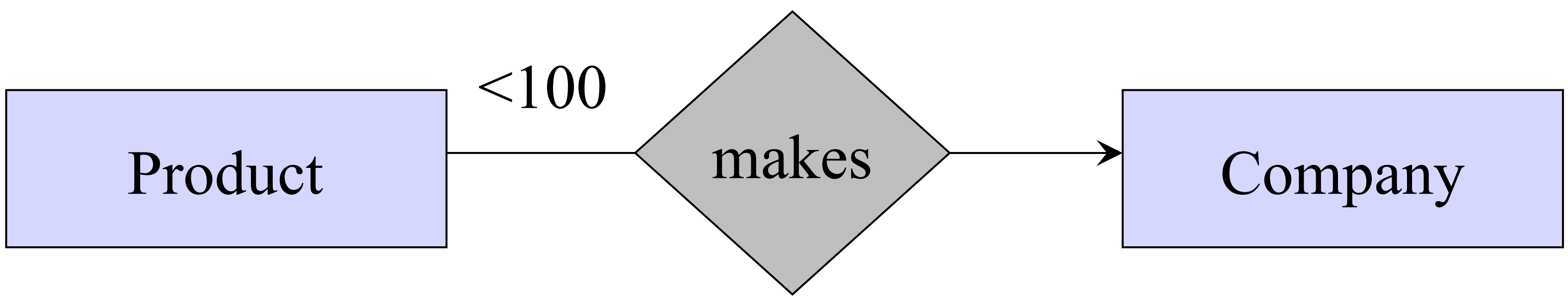


Each product made by at most one company.
Some products made by no company



Each product made by *exactly* one company.

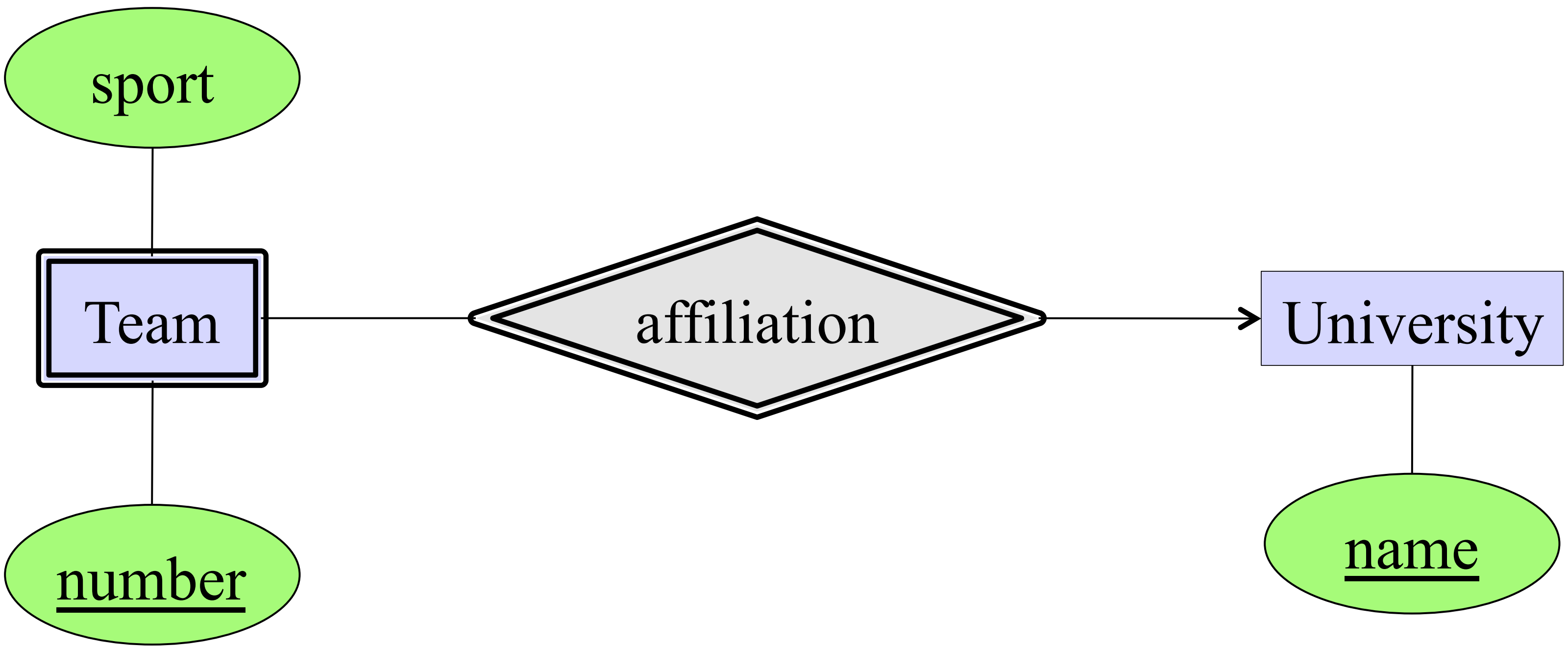
Cardinality Constraints



What does this mean ?

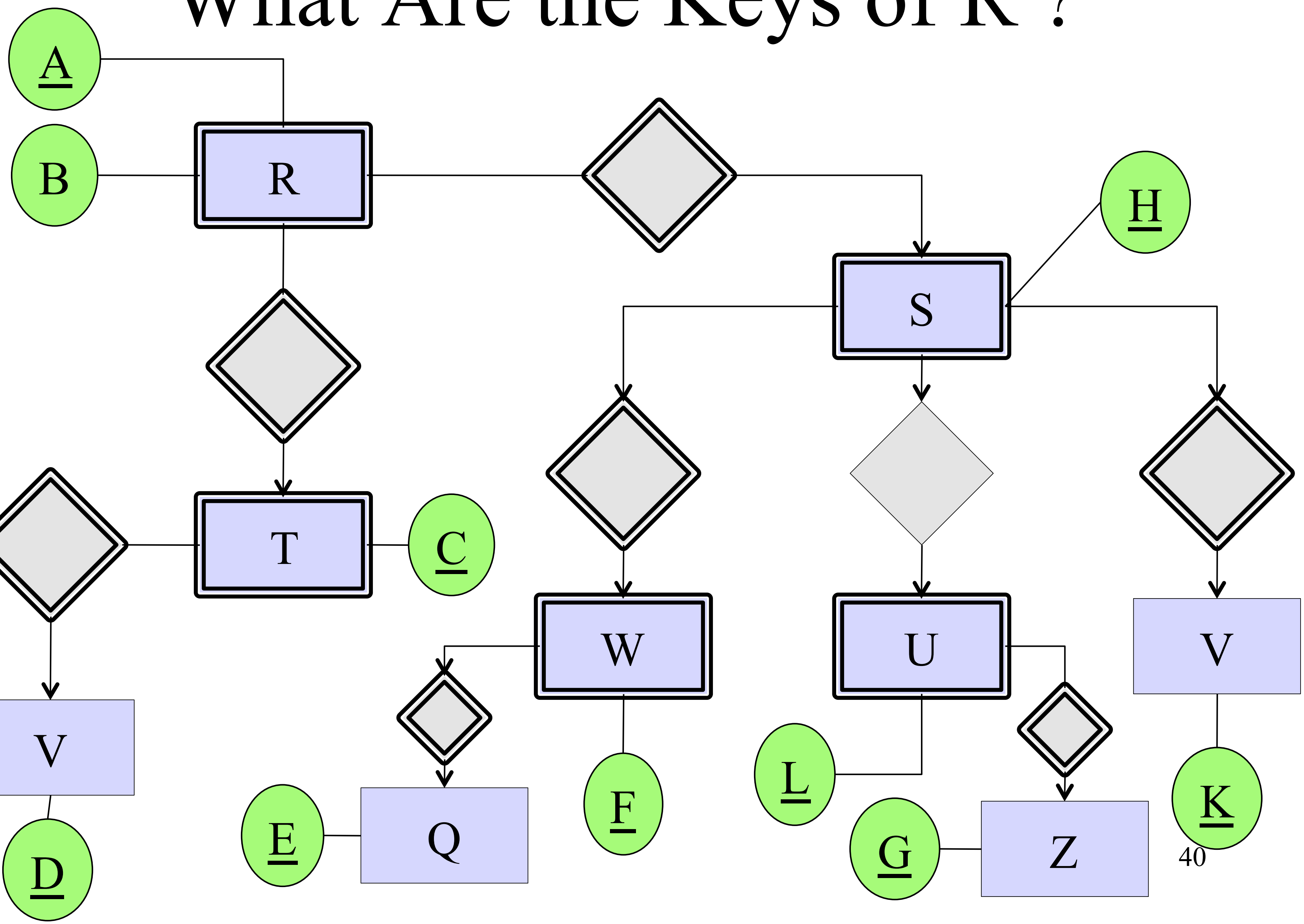
Weak Entity Sets

Weak entity set = entity where part of the key comes from another



Convert to a relational schema (in class)

What Are the Keys of R ?



Schema Refinements = Normal Forms

- 1st Normal Form = all tables are flat
- 2nd Normal Form = obsolete
- Boyce Codd Normal Form = will study
- 3rd Normal Form = see book

First Normal Form (1NF)

- A database schema is in First Normal Form if all tables are flat

Student

Name	GPA	Courses
Alice	3.8	Math DB OS
Bob	3.7	DB OS
Carol	3.9	Math OS

May need to add keys

Student

Name	GPA
Alice	3.8
Bob	3.7
Carol	3.9

Takes

Student	Course
Alice	Math
Carol	Math
Alice	DB
Bob	DB
Alice	OS
Carol	OS

Course

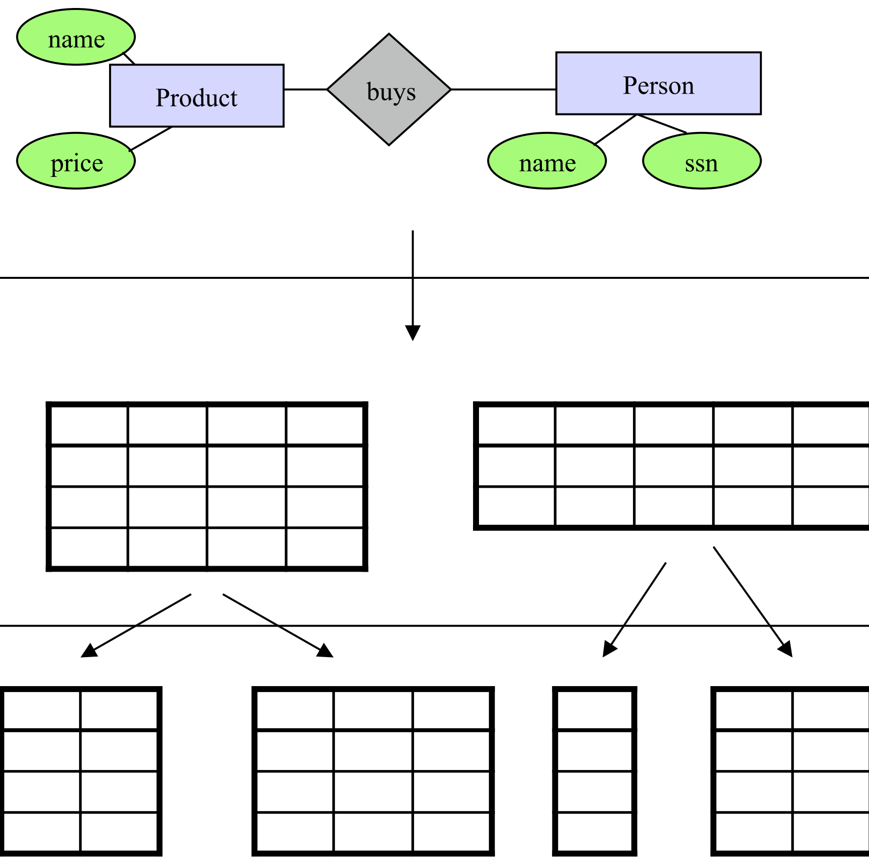
Course
Math
DB
OS

Relational Schema Design

Conceptual Model:

Relational Model:
plus FD's

Normalization:
Eliminates ***anomalies***



Data Anomalies

When a database is poorly designed we get anomalies:

Redundancy: data is repeated

Updated anomalies: need to change in several places

Delete anomalies: may lose data when we don't want

Relational Schema Design

Recall set attributes (persons with several phones):

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield

One person may have multiple phones, but lives in only one city

Anomalies:

- Redundancy = repeat data
- Update anomalies = Fred moves to "Bellevue"
- Deletion anomalies = Joe deletes his phone number: what is his city ?

Relation Decomposition

Break the relation into two:

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield

Name	SSN	City
Fred	123-45-6789	Seattle
Joe	987-65-4321	Westfield

SSN	PhoneNumber
123-45-6789	206-555-1234
123-45-6789	206-555-6543
987-65-4321	908-555-2121

Anomalies have gone:

- No more repeated data
- Easy to move Fred to "Bellevue" (how ?)
- Easy to delete all Joe's phone number (how ?)

Relational Schema Design (or Logical Design)

Main idea:

- Start with some relational schema
- Find out its ***functional dependencies***
- Use them to design a better relational schema

Functional Dependencies

- A form of constraint
  - hence, part of the schema
- Finding them is part of the database design
- Also used in normalizing the relations

Functional Dependencies

Definition:

If two tuples agree on the attributes

$$\boxed{A_1, A_2, \ldots, A_n}$$

then they must also agree on the attributes

$$\boxed{B_1, B_2, \ldots, B_m}$$

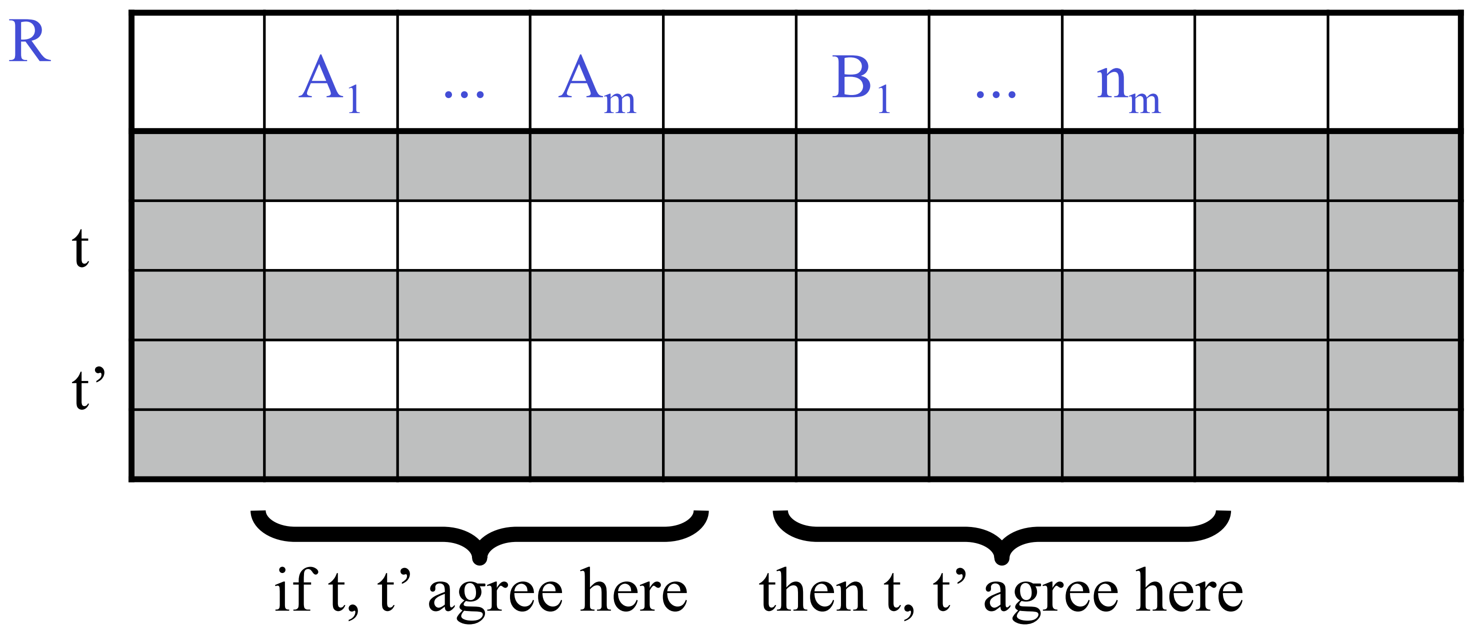
Formally:

$$\boxed{A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m}$$

When Does an FD Hold

Definition: $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ holds in R if:

$$\forall t, t' \in R, (t.A_1 = t'.A_1 \wedge \ldots \wedge t.A_m = t'.A_m \Rightarrow t.B_1 = t'.B_1 \wedge \ldots \wedge t.B_n = t'.B_n)$$



Examples

An FD holds, or does not hold on an instance:

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

EmpID → Name, Phone, Position

Position → Phone

but not Phone → Position

Example

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876 ←	Salesrep
E1111	Smith	9876 ←	Salesrep
E9999	Mary	1234	Lawyer

Position → Phone

Example

EmpID	Name	Phone	Position
E0045	Smith	1234 →	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234 →	Lawyer

but not Phone → Position

Example

FD's are constraints:
• On some instances they hold
• On others they don't

name → color
category → department
color, category → price

name	category	color	department	price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Green	Toys	99

Does this instance satisfy all the FDs ?

Example

name → color
category → department
color, category → price

name	category	color	department	price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Black	Toys	99
Gizmo	Stationary	Green	Office-supp.	59

What about this one ? (At home…)

An Interesting Observation

If all these FDs are true:

name → color
category → department
color, category → price

Then this FD also holds:

name, category → price

Why ??

Goal: Find ALL Functional Dependencies

- Anomalies occur when certain “bad” FDs hold

- We know some of the FDs

- Need to find *all* FDs, then look for the bad ones

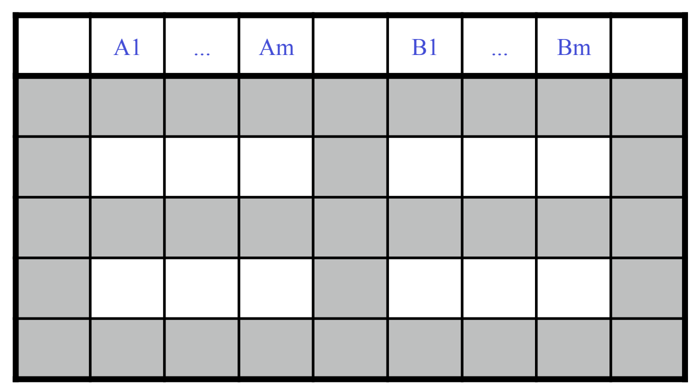
Armstrong's Rules (1/3)

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

Splitting rule
and
Combing rule

Is equivalent to

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



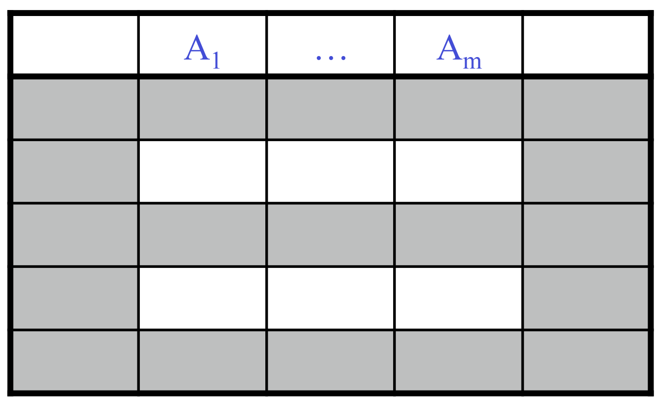
Armstrong's Rules (2/3)

$$\boxed{A_1, A_2, \ldots, A_n \rightarrow A_i}$$

Trivial Rule

where i = 1, 2, ..., n

Why ?



Armstrong's Rules (3/3)

Transitive Closure Rule

If $\boxed{A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m}$

and $\boxed{B_1, B_2, \ldots, B_m \rightarrow C_1, C_2, \ldots, C_p}$

then $\boxed{A_1, A_2, \ldots, A_n \rightarrow C_1, C_2, \ldots, C_p}$

Why ?

	A_1	…	A_m		B_1	…	B_m		C_1	…	C_p	

Example (continued)

Start from the following FDs:

1. name → color
2. category → department
3. color, category → price

Infer the following FDs:

Inferred FD	Which Rule did we apply ?
4. name, category → name	
5. name, category → color	
6. name, category → category	
7. name, category → color, category	
8. name, category → price	

Example (continued)

1. name → color
2. category → department
3. color, category → price

Answers:

Inferred FD	Which Rule did we apply ?
4. name, category → name	Trivial rule
5. name, category → color	Transitivity on 4, 1
6. name, category → category	Trivial rule
7. name, category → color, category	Split/combine on 5, 6
8. name, category → price	Transitivity on 3, 7

THIS IS TOO HARD ! Let's see an easier way.

Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$

The **closure**, $\{A_1, \ldots, A_n\}^+$ = the set of attributes B s.t. $A_1, \ldots, A_n \rightarrow B$

Example:

name → color
category → department
color, category → price

Closures:

$\text{name}^+ = \{\text{name, color}\}$
$\{\text{name, category}\}^+ = \{\text{name, category, color, department, price}\}$
$\text{color}^+ = \{\text{color}\}$

Closure Algorithm

X={A1, …, An}.

Repeat until X doesn't change **do**:

if $B_1, \ldots, B_n \rightarrow C$ is a FD **and**
$B_1, \ldots, B_n$ are all in X
then add C to X.

Example:

name → color
category → department
color, category → price

{name, category}$^+$ =
{ name, category, color, department, price }

Hence: name, category → color, department, price

Example

In class:

R(A,B,C,D,E,F)

A, B	$\rightarrow$	C
A, D	$\rightarrow$	E
B	$\rightarrow$	D
A, F	$\rightarrow$	B

Compute $\{A,B\}^+$ $X = \{A, B,$ $\}$

Compute $\{A, F\}^+$ $X = \{A, F,$ $\}$

Why Do We Need Closure

- With closure we can find all FD's easily

- To check if $X \rightarrow A$
  - Compute X^+
  - Check if $A \in X^+$

Using Closure to Infer ALL FDs

Example:

$A, B \rightarrow C$
$A, D \rightarrow B$
$B \rightarrow D$

Step 1: Compute X^+, for every X:

$A+ = A$, $B+ = BD$, $C+ = C$, $D+ = D$
$AB+ = ABCD$, $AC+ = AC$, $AD+ = ABCD$,
$BC+ = BCD$, $BD+ = BD$, $CD+ = CD$
$ABC+ = ABD+ = ACD^+ = ABCD$ (no need to compute– why ?)
$BCD^+ = BCD$, $ABCD+ = ABCD$

Step 2: Enumerate all FD's $X \rightarrow Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

$AB \rightarrow CD$, $AD \rightarrow BC$, $BC \rightarrow D$

Another Example

- Enrollment(student, major, course, room, time)

  student → major

  major, course → room

  course → time

  What else can we infer ? [in class, or at home]

Keys

- A **superkey** is a set of attributes $A_1, ..., A_n$ s.t. for any other attribute B, we have $A_1, ..., A_n \rightarrow B$

- A **key** is a minimal superkey
  – I.e. set of attributes which is a superkey and for which no subset is a superkey

Computing (Super)Keys

- Compute X^+ for all sets X
- If X^+ = all attributes, then X is a key
- List only the minimal X's

Example

Product(name, price, category, color)

name, category $\rightarrow$ price
category $\rightarrow$ color

What is the key ?

Example

Product(name, price, category, color)

name, category → price
category → color

What is the key ?

(name, category) + = name, category, price, color

Hence (name, category) is a key

Examples of Keys

Enrollment(student, address, course, room, time)

student → address
room, time → course
student, course → room, time

(find keys at home)

Eliminating Anomalies

Main idea:

- $X \rightarrow A$ is OK if X is a (super)key
- $X \rightarrow A$ is not OK otherwise

Example

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield
Joe	987-65-4321	908-555-1234	Westfield

SSN → Name, City

What the key?

{SSN, PhoneNumber}

Hence SSN → Name, City
is a "bad" dependency

Key or Keys ?

Can we have more than one key ?

Given R(A,B,C) define FD’s s.t. there are two or more keys

Key or Keys ?

Can we have more than one key ?

Given R(A,B,C) define FD's s.t. there are two or more keys

$AB \rightarrow C$ $BC \rightarrow A$	or	$A \rightarrow BC$ $B \rightarrow AC$

what are the keys here ?

Can you design FDs such that there are *three* keys ?

Boyce-Codd Normal Form

A simple condition for removing anomalies from relations:

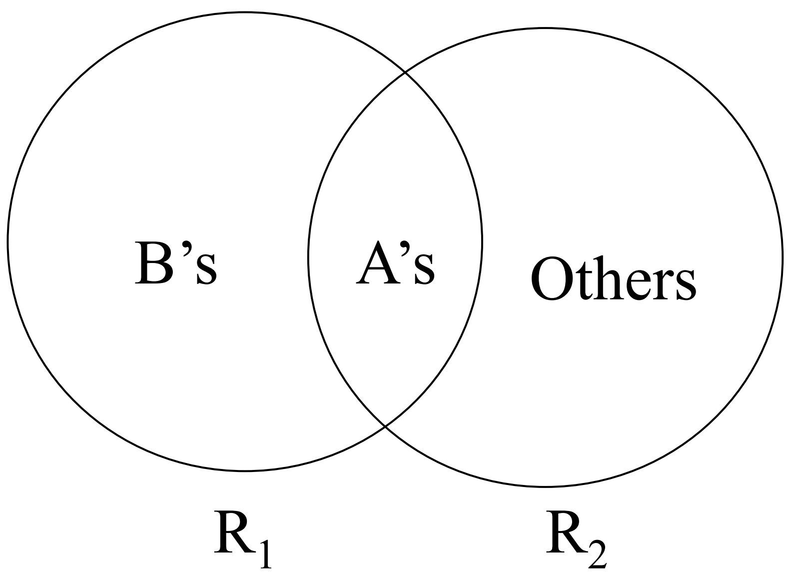
> A relation R is in BCNF if:
>
> If $A_1, \ldots, A_n \rightarrow B$ is a non-trivial dependency in R, then $\{A_1, \ldots, A_n\}$ is a superkey for R

In other words: there are no "bad" FDs

Equivalently:
$\forall$ X, either ($X^+ = X$) or ($X^+ =$ all attributes)

BCNF Decomposition Algorithm

<u>repeat</u>
  choose $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ that violates BNCF
  split R into $R_1(A_1, \ldots, A_m, B_1, \ldots, B_n)$ and $R_2(A_1, \ldots, A_m, [\text{others}])$
  continue with both R_1 and R_2
<u>until</u> no more violations



Is there a 2-attribute relation that is not in BCNF ?

In practice, we have a better algorithm (coming up)

Example

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield
Joe	987-65-4321	908-555-1234	Westfield

$\boxed{SSN \rightarrow Name, City}$

What the key?

{SSN, PhoneNumber}

use SSN → Name, City to split

Example

Name	SSN	City
Fred	123-45-6789	Seattle
Joe	987-65-4321	Westfield

SSN → Name, City

SSN	PhoneNumber
123-45-6789	206-555-1234
123-45-6789	206-555-6543
987-65-4321	908-555-2121
987-65-4321	908-555-1234

Let's check anomalies:

- Redundancy ?
- Update ?
- Delete ?

BCNF Decomposition Algorithm

BCNF_Decompose(R)

find X s.t.: $X \neq X^+ \neq$ [all attributes]

<u>**if**</u> (not found) <u>**then**</u> "R is in BCNF"

<u>**let**</u> $Y = X^+ - X$
<u>**let**</u> $Z = [\text{all attributes}] - X^+$
decompose R into $R1(X \cup Y)$ and $R2(X \cup Z)$
continue to decompose recursively R1 and R2

Find X s.t.: $X \neq X^+ \neq$ [all attributes]

Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

SSN $\rightarrow$ name, age

age $\rightarrow$ hairColor

In class….

Find X s.t.: $X \neq X^+ \neq$ [all attributes]

Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

SSN → name, age

age → hairColor

Iteration 1: Person

SSN+ = SSN, name, age, hairColor

Decompose into: P(<u>SSN</u>, name, age, hairColor)

Phone(SSN, phoneNumber)

Iteration 2: P

age+ = age, hairColor

Decompose: People(<u>SSN</u>, name, age)

Hair(<u>age</u>, hairColor)

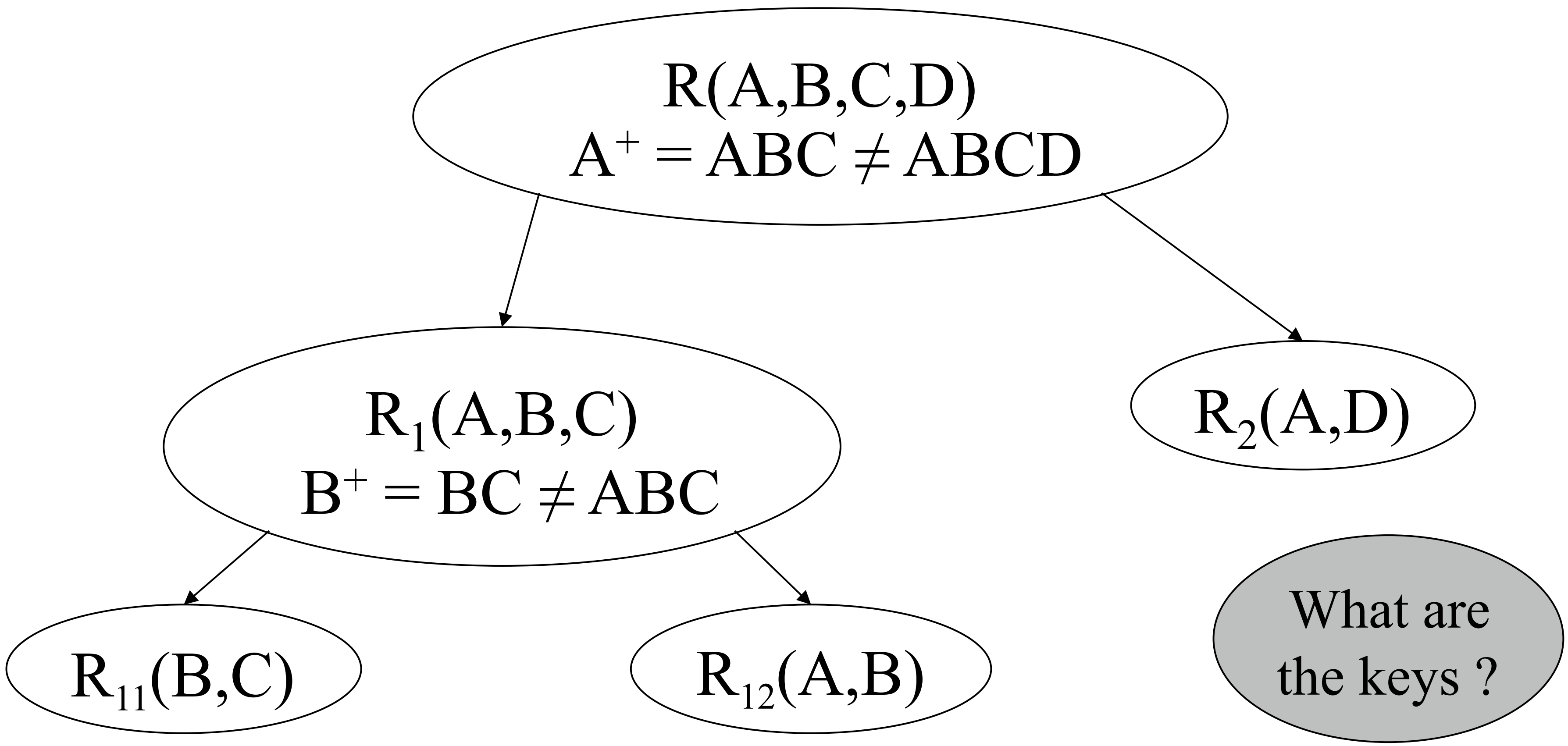
Phone(SSN, phoneNumber)

What are the keys ?

$R(A,B,C,D)$

$A \rightarrow B$
$B \rightarrow C$

Example



What happens if in R we first pick B^+ ? Or AB^+ ?

Decompositions in General

$R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_p)$

$R_1(A_1, \ldots, A_n, B_1, \ldots, B_m)$

$R_2(A_1, \ldots, A_n, C_1, \ldots, C_p)$

R_1 = projection of R on $A_1, \ldots, A_n, B_1, \ldots, B_m$

R_2 = projection of R on $A_1, \ldots, A_n, C_1, \ldots, C_p$

Theory of Decomposition

- Sometimes it is correct:

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Price
Gizmo	19.99
OneClick	24.99
~~Gizmo~~	~~19.99~~

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Lossless decomposition

Incorrect Decomposition

- Sometimes it is not:

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

What's incorrect ??

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Price	Category
19.99	Gadget
24.99	Camera
19.99	Camera

Lossy decomposition

Decompositions in General

$R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_p)$

$R_1(A_1, \ldots, A_n, B_1, \ldots, B_m)$

$R_2(A_1, \ldots, A_n, C_1, \ldots, C_p)$

If $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$
Then the decomposition is lossless

Note: don't need $A_1, \ldots, A_n \rightarrow C_1, \ldots, C_p$

BCNF decomposition is always lossless. WHY ?