Logical Database Design: Entity-Relation Models

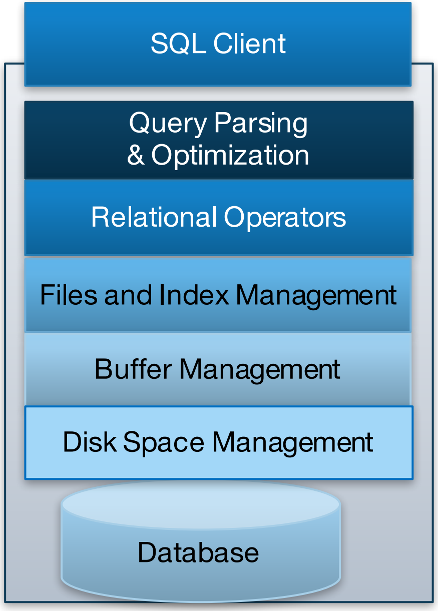
Alvin Cheung

Aditya Parameswaran

R&G 2

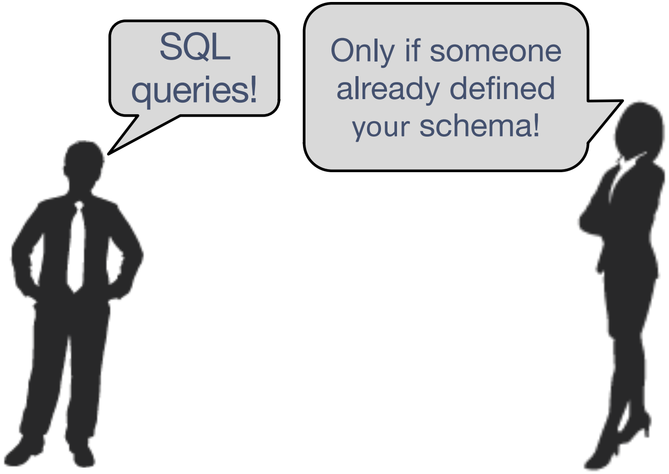
Architecture of a DBMS

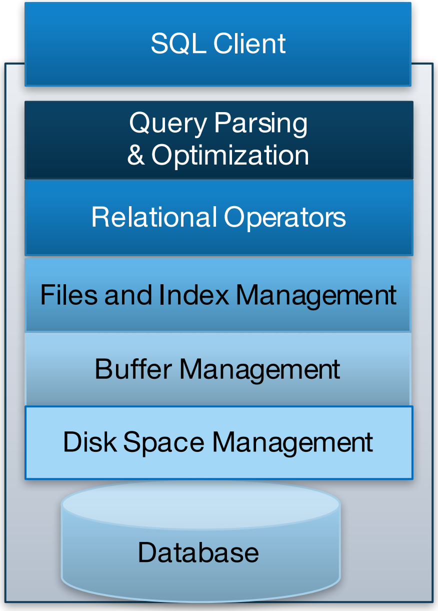
- Gives us a good sense of how to build a DBMS
- How about using one?



Architecture of a DBMS, Pt 2

- Gives us a good sense of how to build a DBMS
- How about using one?




SQL Client

Query Parsing & Optimization

Relational Operators

Files and Index Management

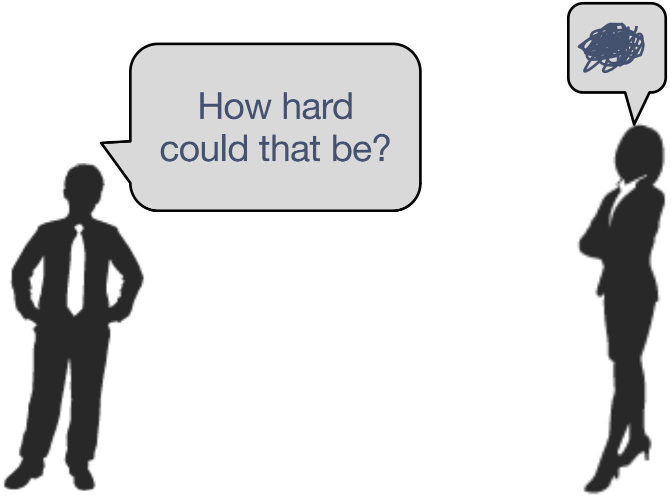
Buffer Management

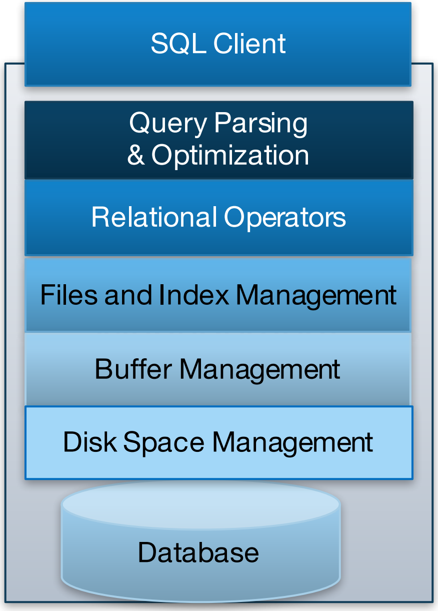
Disk Space Management

Database

Architecture of a DBMS, Pt 3

- Gives us a good sense of how to build a DBMS
- How about using one?




SQL Client

Query Parsing & Optimization

Relational Operators

Files and Index Management

Buffer Management

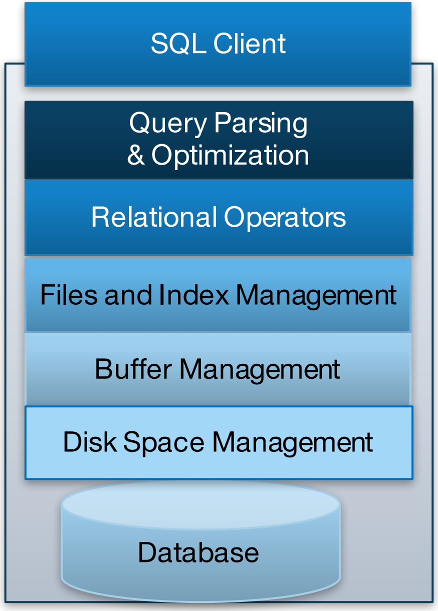
Disk Space Management

Database

Design of a Database

- Gives us a good sense of how to build a DBMS
- How about using one?

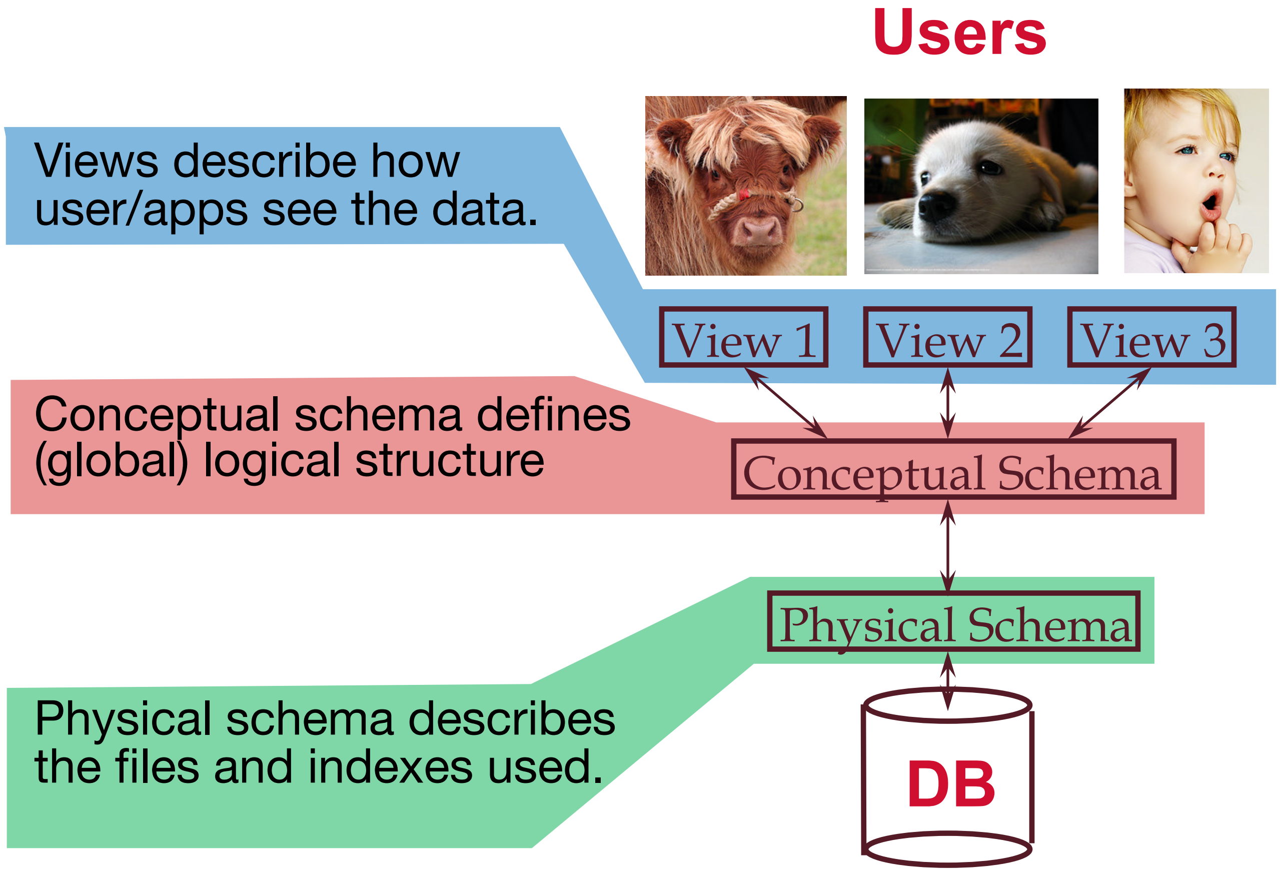
- Today let's talk about how to design a database
  - Not a database system
  - Let's start with what we know… data models



Describing Data: Data Models

- **Data model :** collection of concepts for describing data.
- **Schema:** description of a particular collection of data, using a given data model.
- **Relational model of data**
  - Main concept: relation (table), rows and columns
  - Every relation has a schema
    - describes the columns
    - column names and domains

Levels of Abstraction: Various Schemas



Users

Views describe how user/apps see the data.

View 1

View 2

View 3

Conceptual schema defines (global) logical structure

Conceptual Schema

Physical Schema

Physical schema describes the files and indexes used.

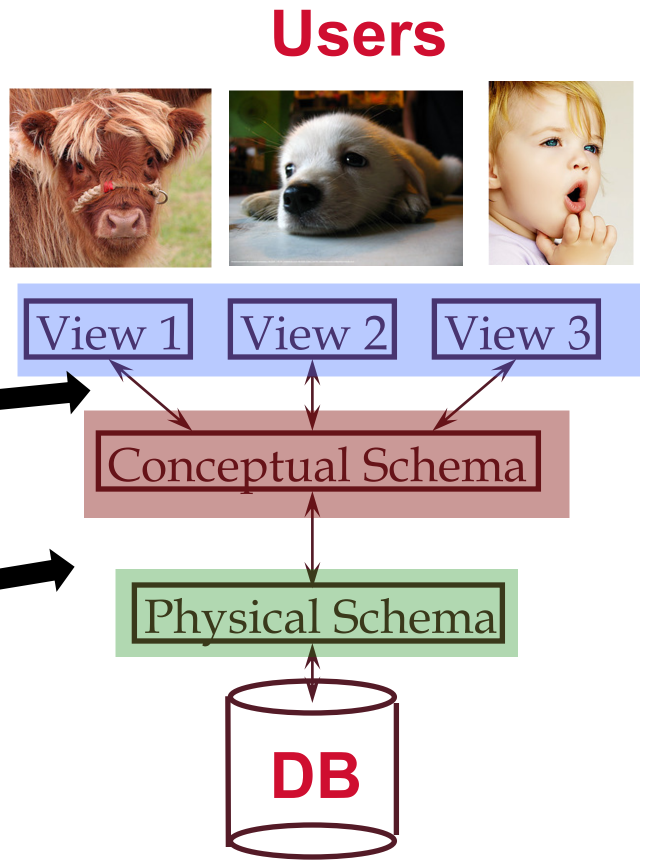
DB

Example: University Database

- **Conceptual schema:**
  - `Students(sid text, name text, login text,`
    `age integer, gpa float)`
  - `Courses(cid text, cname text,`
    `credits integer)`
  - `Enrolled(sid text, cid text, grade text)`
- **Physical schema:**
  - Relations stored as unordered files.
  - Index on first column of Students.
- **External/View schema**:
  - `Course_info (cid text, enrollment integer)`
    - Group by query on `Enrolled`

Data Independence

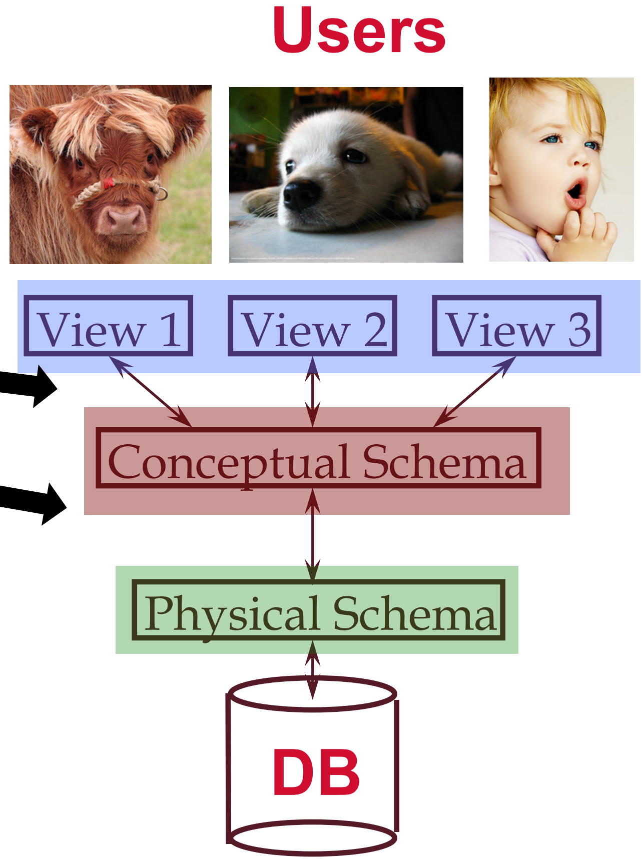
- Insulate apps from structure of data
- **Logical data independence:**
  - Maintain views when logical structure [schema] changes
  - E.g., if we split a relation into two, can simply change view query, apps that use views don’t need to be rewritten
- **Physical data independence:**
  - Maintain logical structure when physical structure changes
  - E.g., if we add an index, queries to conceptual schema don’t need to be rewritten



Data Independence

- Insulate apps from structure of data

- **Logical data independence:**
- **Physical data independence:**
- Q: Why important for DBMS?
- Because databases and their associated apps persist over long periods of time!
    - Applications, hardware may change, …
    - E.g., Many banks are still running old database systems
    - Modularity is key



Data Models

- Relational Model:
  - A collection of relations
  - Easy to implement in a database
  - Easy to provide both logical and physical data independence
  - Harder to reason about
- Today: Entity-Relational (ER) Model
  - A collection of entities and relations
  - Harder to implement in a database directly
  - But easier to reason about
- So we will talk about the ER Model and how to translate it into the Relational Model
- Let’s talk about the workflow of database design…

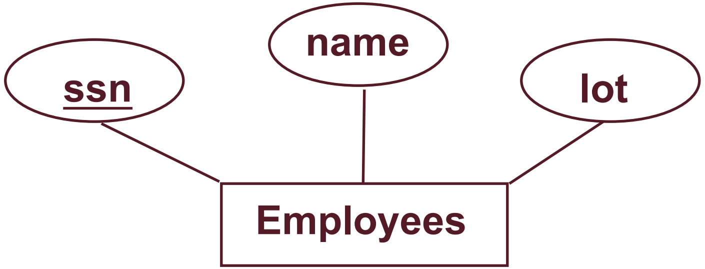
Many Steps in Database Design!

- **Requirements Analysis**
  - Translating user needs; what must database do? what must it capture?
- **Conceptual Design [Needs => ER Diagram]**
  - *high level visual description of data(often done w/ER model)* ⇦ You are here
  - Object-Relational Mappings (ORMs: Hibernate, Rails, Django, etc) encourage you to program here [essentially ER models]
- **Logical Design [ER Diagram => Relations]**
  - translate ER into DBMS data model
  - ORMs often require you to help here too
  - can be partially automated
- **Schema Refinement [Relations => Better Relations]**
  - consistency, normalization [add constraints, break or merge relations]
- **Physical Design [Storing Relations]**
  - indexes, disk layout

- **Orthogonal: Security Design [Relational Access Control]**
  - who accesses what, and how

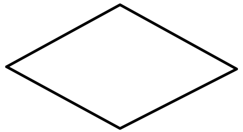
ER Model Basics: Entities

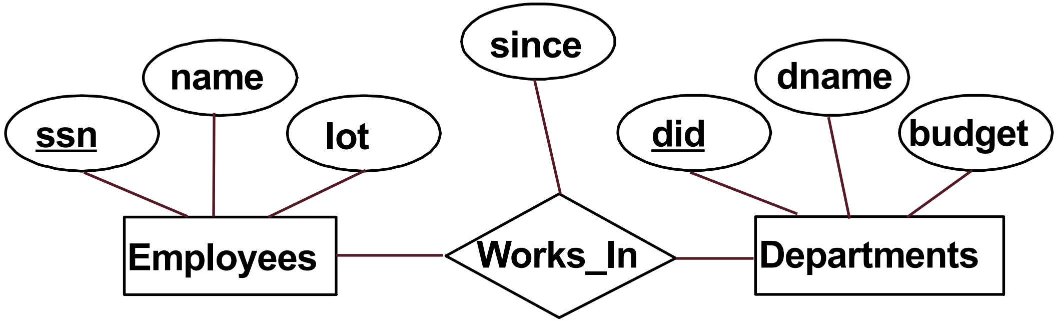
- **Entity**:
  - A real-world object described by a set of attribute values.



- **Entity Set:** A collection of similar entities.
  - E.g., all employees.
  - All entities in an entity set have the same attributes.
  - Each entity set has a primary key (underlined)
  - Each attribute has a domain

ER Model Basics: Relationships





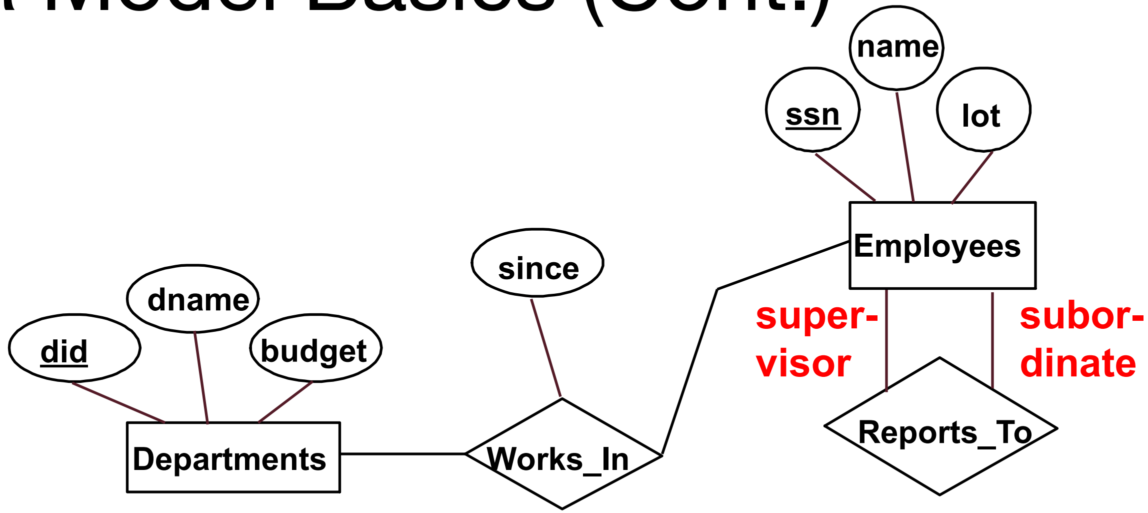
Relationship: Association among two or more entities.

- E.g., Murtaza works in Pharmacy department.
- Relationships can have their own attributes.

Relationship Set: Collection of similar relationships.

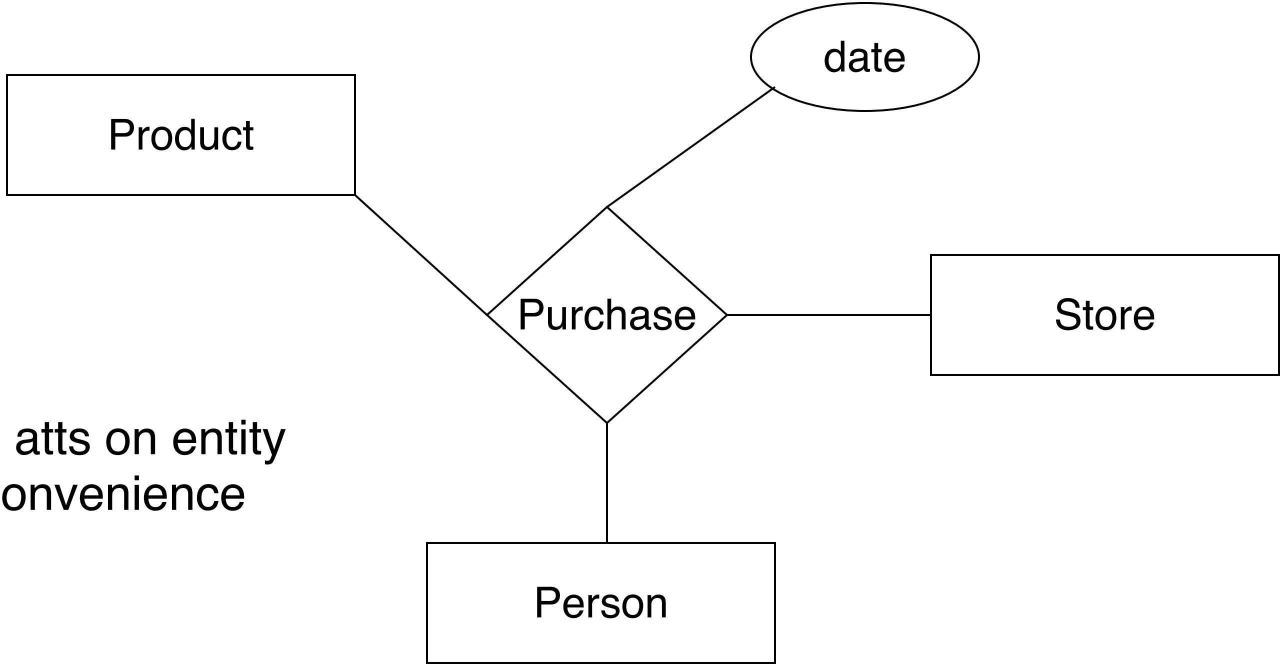
- An n-ary relationship set R relates n entity sets E1 ... En ; each relationship in R involves entities e1 $\in$ E1, ..., en $\in$ En

ER Model Basics (Cont.)



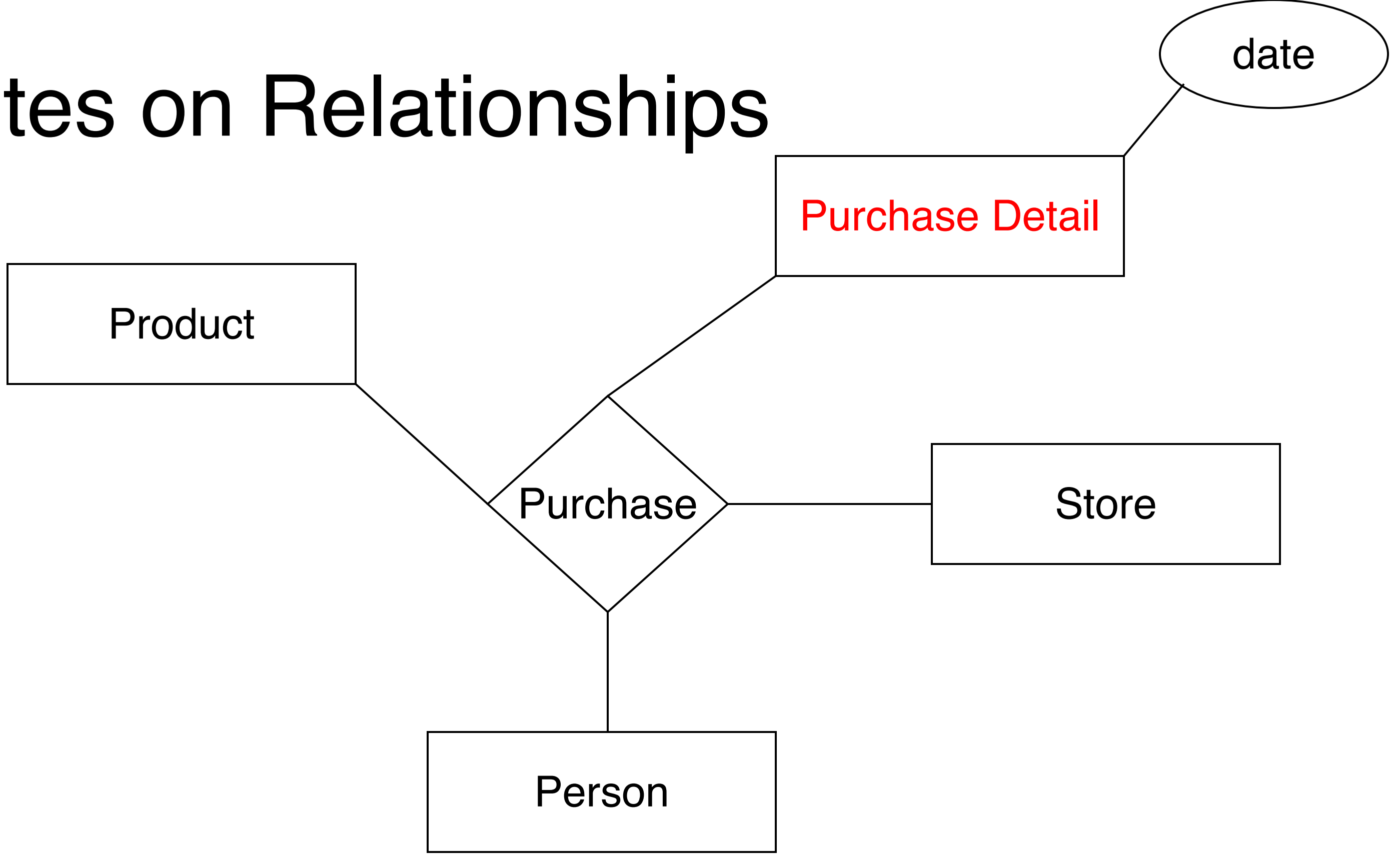
Same entity set can participate in different relationship sets, or in different “roles” in the same relationship set.

Attributes on Relationships



Dropping atts on entity sets for convenience

Attributes on Relationships



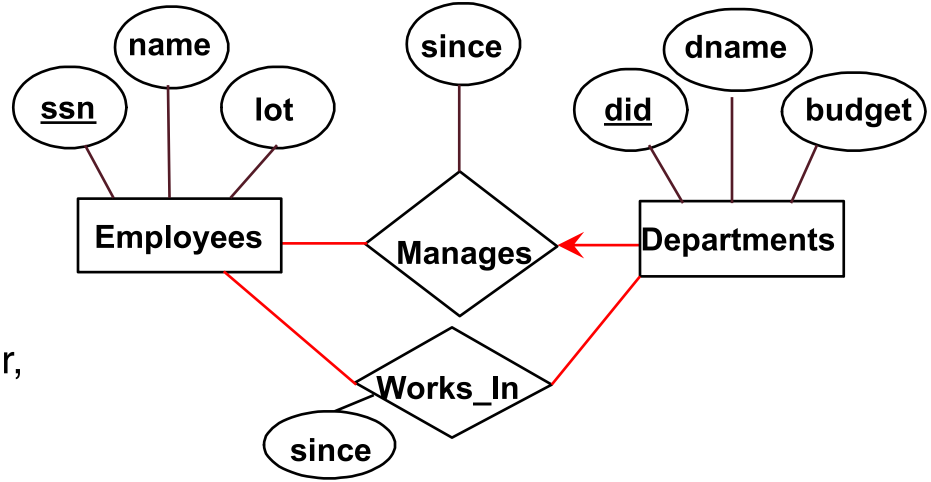
Thus, not necessary to add attributes to relationships.
But, this is overkill; simpler to have date as an attribute of Purchase

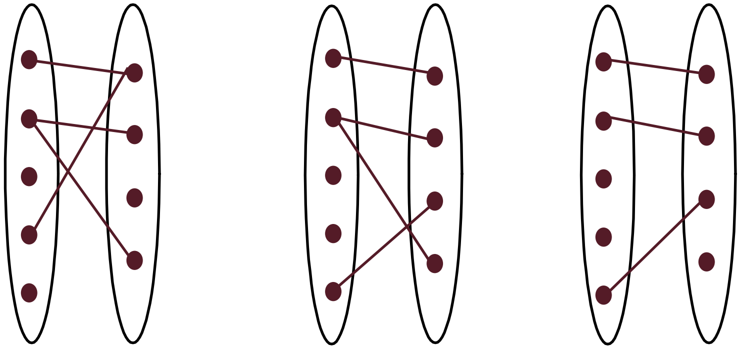
Key Constraints

- An employee can work in **many** departments; a dept can have **many** employees.

- In contrast, each dept has **at most one** manager, according to the *key constraint* on **Dept** in the **Manages** relationship set. Equivalently:
  - Each dept participates at most once in this relationship
  - Each dept has at most one emp. managing it

- A **key constraint** gives a 1-to-many/many-to-1 relationship.



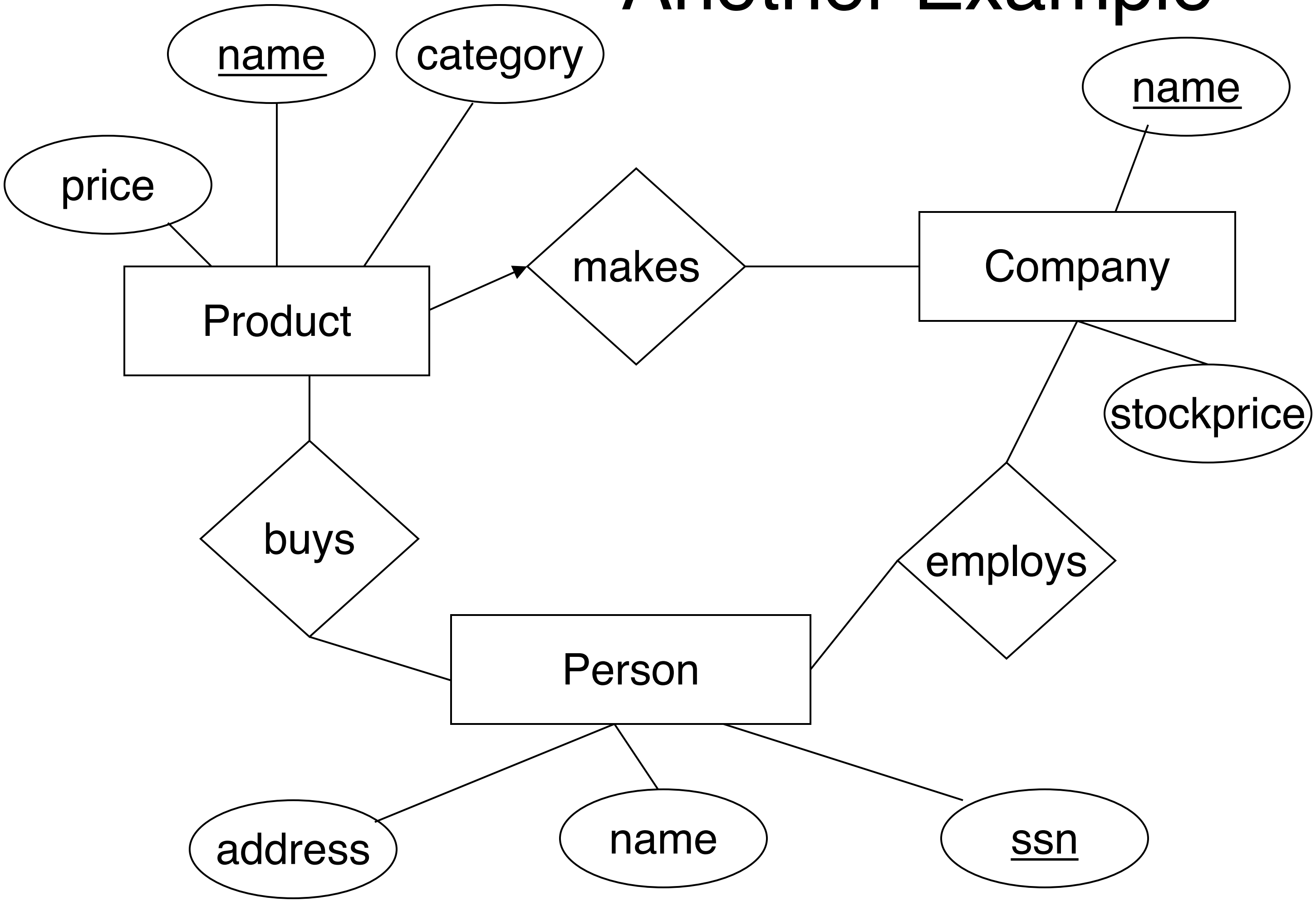


Many-to-Many

1-to-Many | Many-to-1

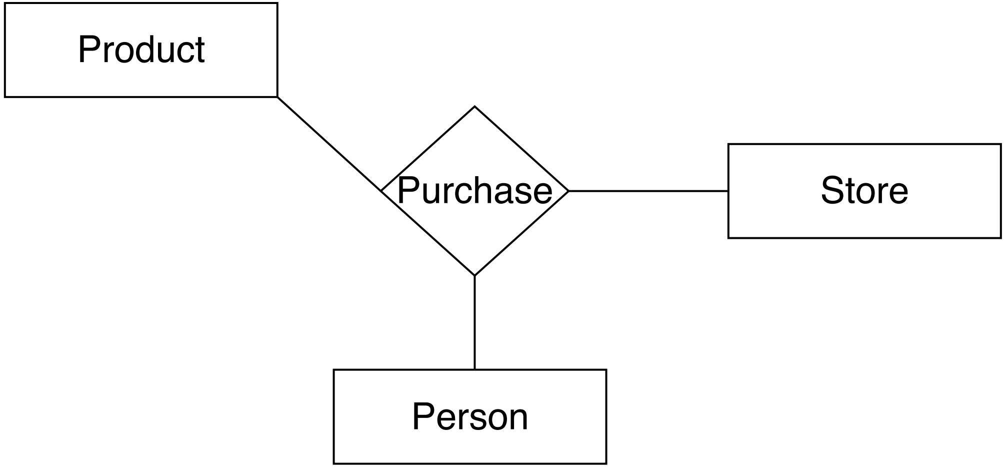
1-to-1

Another Example



Recall: K-ary Relationships

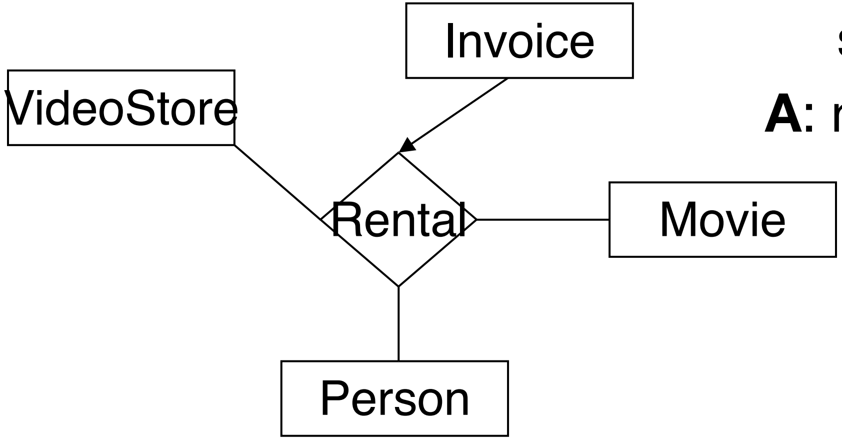
How do we model a purchase relationship between buyers, products and stores?



Essentially capturing a subset of product x store x person

Arrows in Multiway Relationships

Q: what does the arrow mean ?



A: “At most one”. Each invoice can be with at most one movie, person, and video store combo

Q: What if I had an arrow from Person?

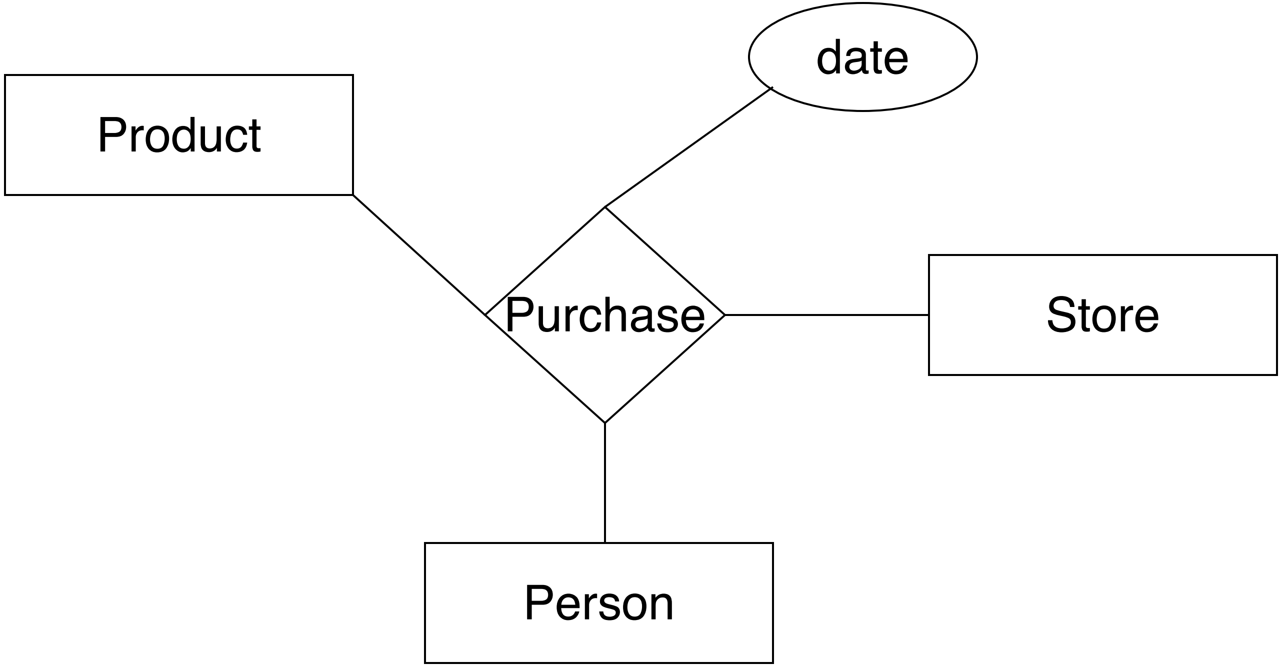
Q: how do I say: “each person only shops at most one store” ?

A: no good way ☹

Some ER Modeling Tools Require 2-way Relationships

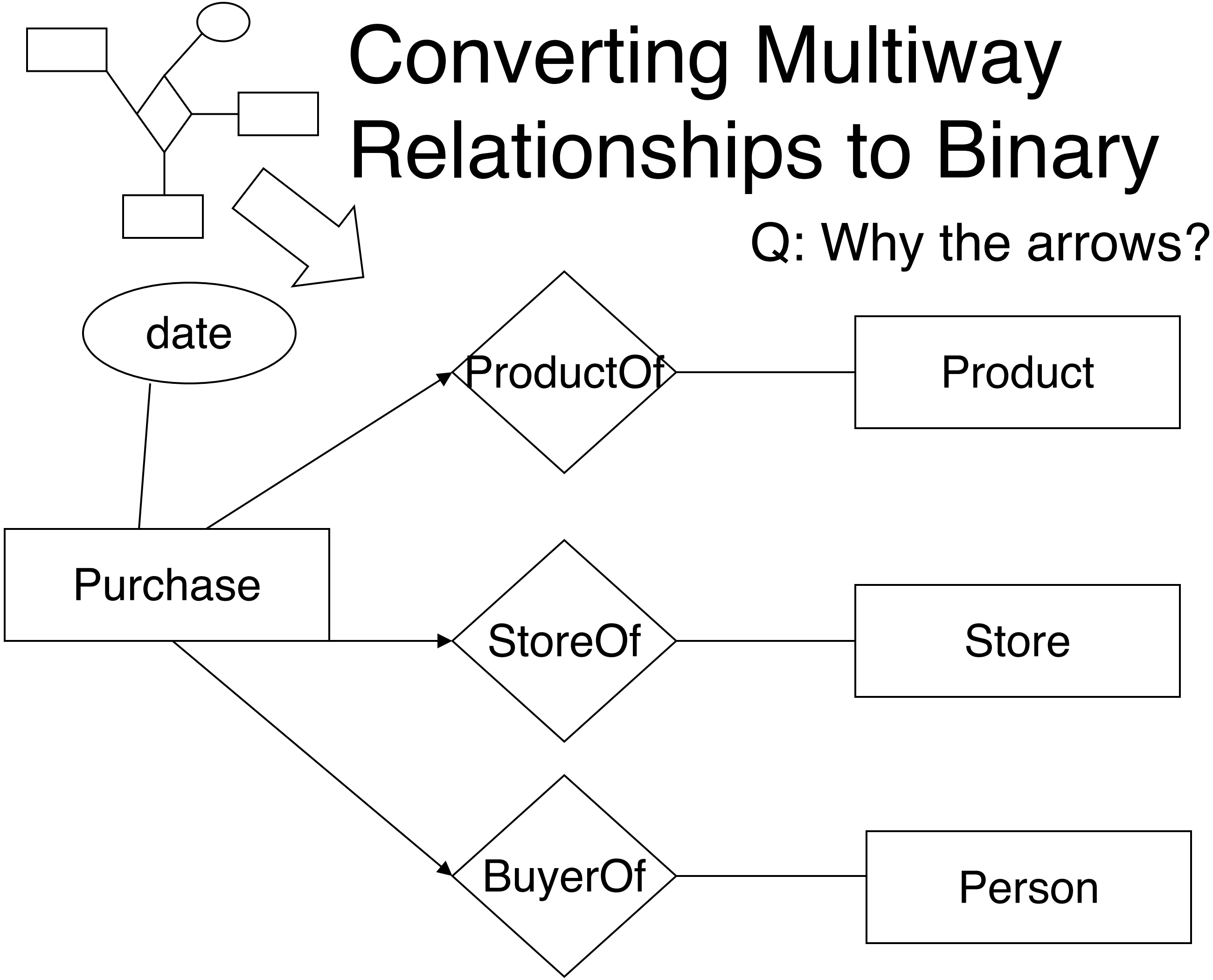
Do we need multi-way relationships or do 2-way (binary) relationships suffice?

How would you convert this into binary?



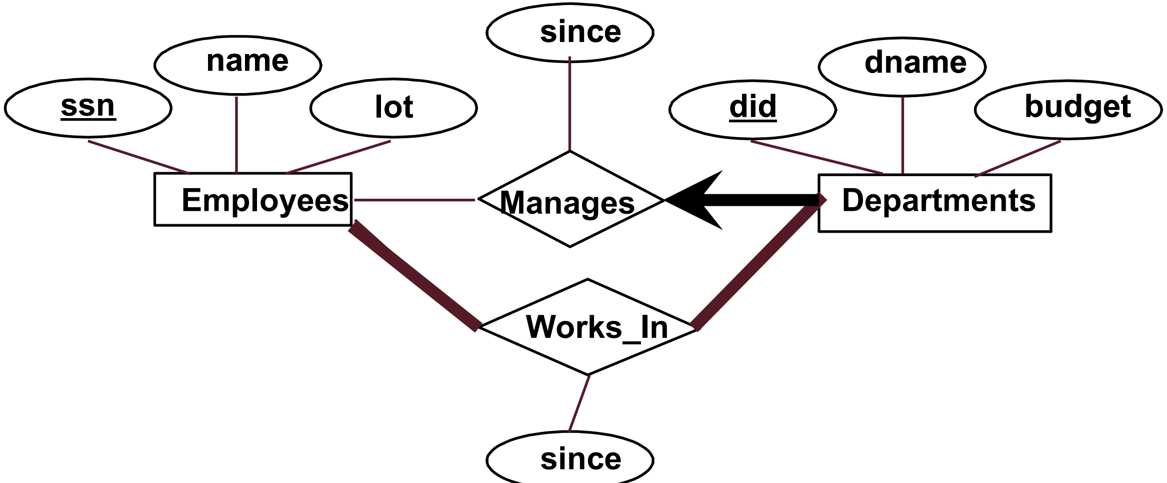
Converting Multiway Relationships to Binary

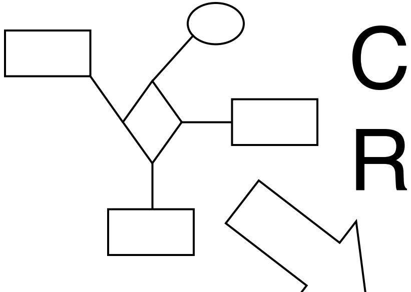
Q: Why the arrows?



Participation Constraints

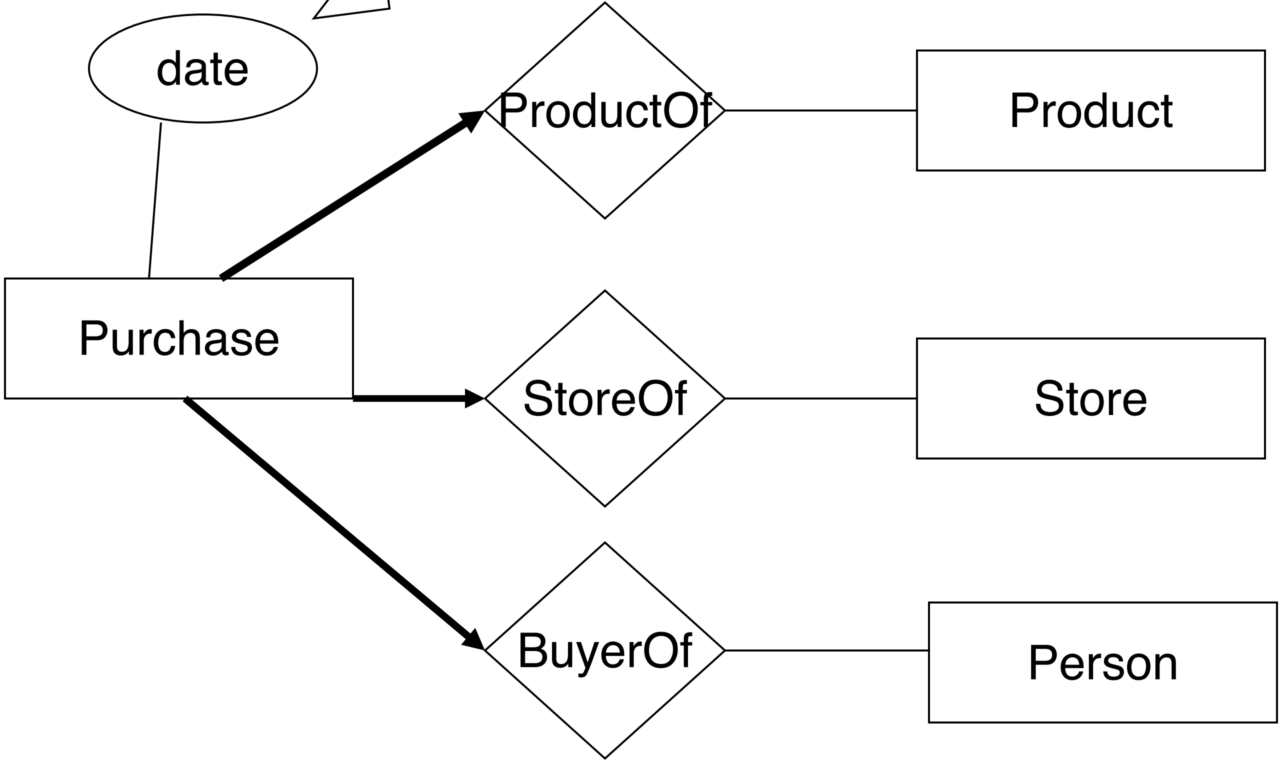
- Does every employee work in a department?
  - If so: a **participation constraint**
  - participation of Employees in Works_In is **total** (vs. partial)
  - Basically means that every employee participates in “**at least one**”.
- Likewise, what if every department has an employee working in it?
- Likewise, what if every department has a manager?
  - Along with the arrow (at most one), this means exactly one





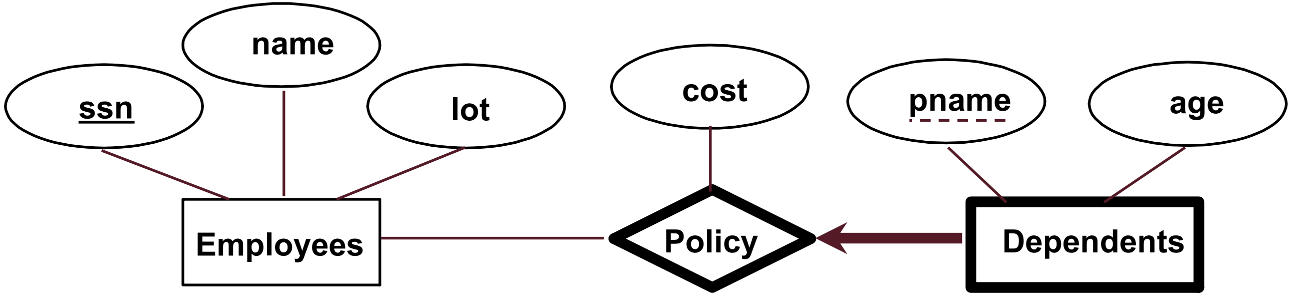
Converting Multiway Relationships to Binary II

At most 1 + At least 1 = Exactly 1!



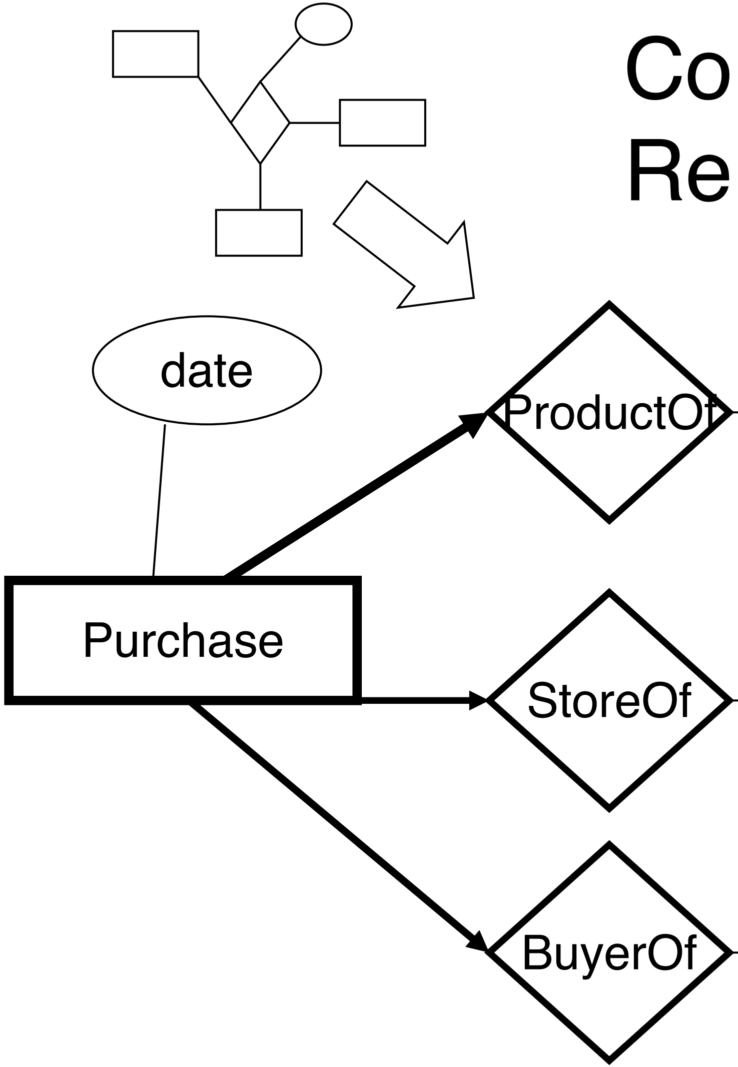
Weak Entities

- A **weak entity** can be identified uniquely only by considering the primary key of other (owner) entities.
  - Owner entity set(s) and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities)
    - Each empl. can have multiple dep. Each dep is associated w. at most 1 empl.
  - Weak entity set must have total participation in this relationship set.
    - Each dep is associated with at least one empl.



- Weak entities have only a “partial key” (dashed underline) [pname] (+ [ssn] = full key)

Converting Multiway Relationships to Binary III



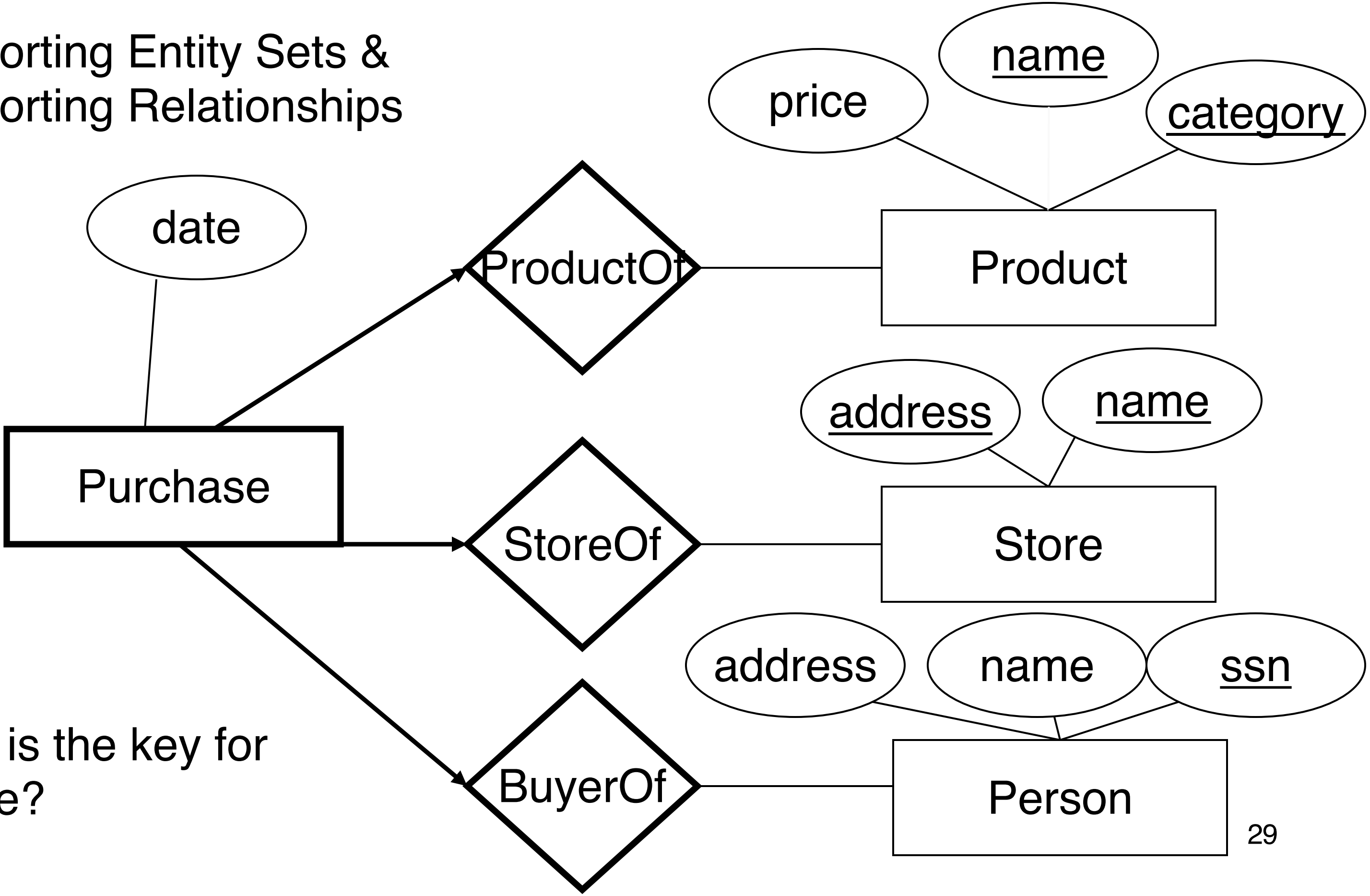
Purchase is in fact, a weak entity set

Purchase requires key attr from other relations to help identify it

No partial key in this case

Multi-Way Relationship Transformation

3 Supporting Entity Sets &
3 Supporting Relationships

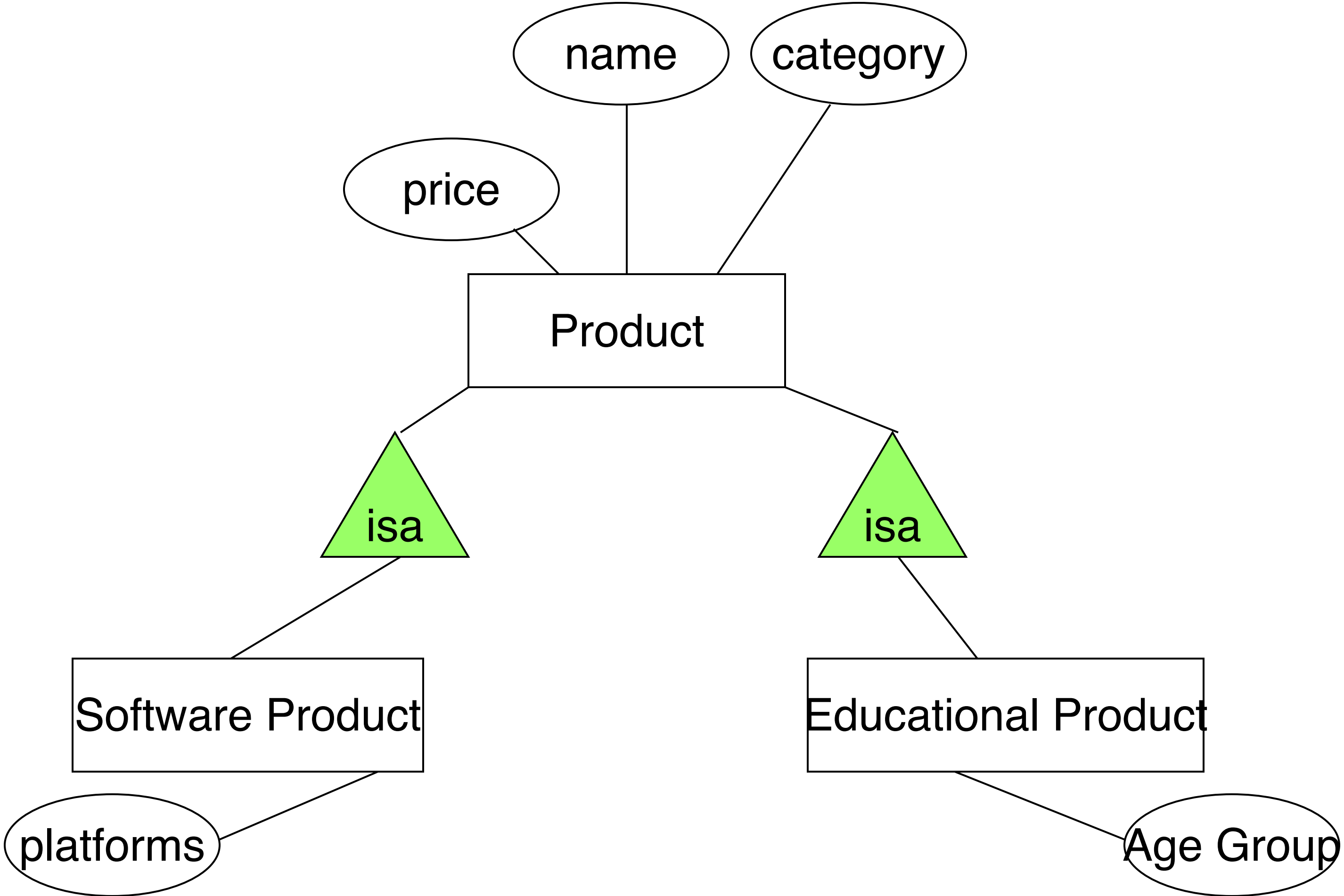


Q: What is the key for Purchase?

Relationships: Summary

- Modeled as a mathematical set
- Binary and multi-way relationships
- Converting a multi-way one into many binary ones
- Constraints on the degree of the relationship
  - many-one, one-one, many-many
  - participation constraints
  - limitations of arrows
- Attributes of relationships
  - not necessary, but useful
- Weak entity sets & supporting relationships

Subclasses in ER Diagrams



Subclasses

- “Isa” triangles indicate the subclass relationship.
  - Point to the superclass.

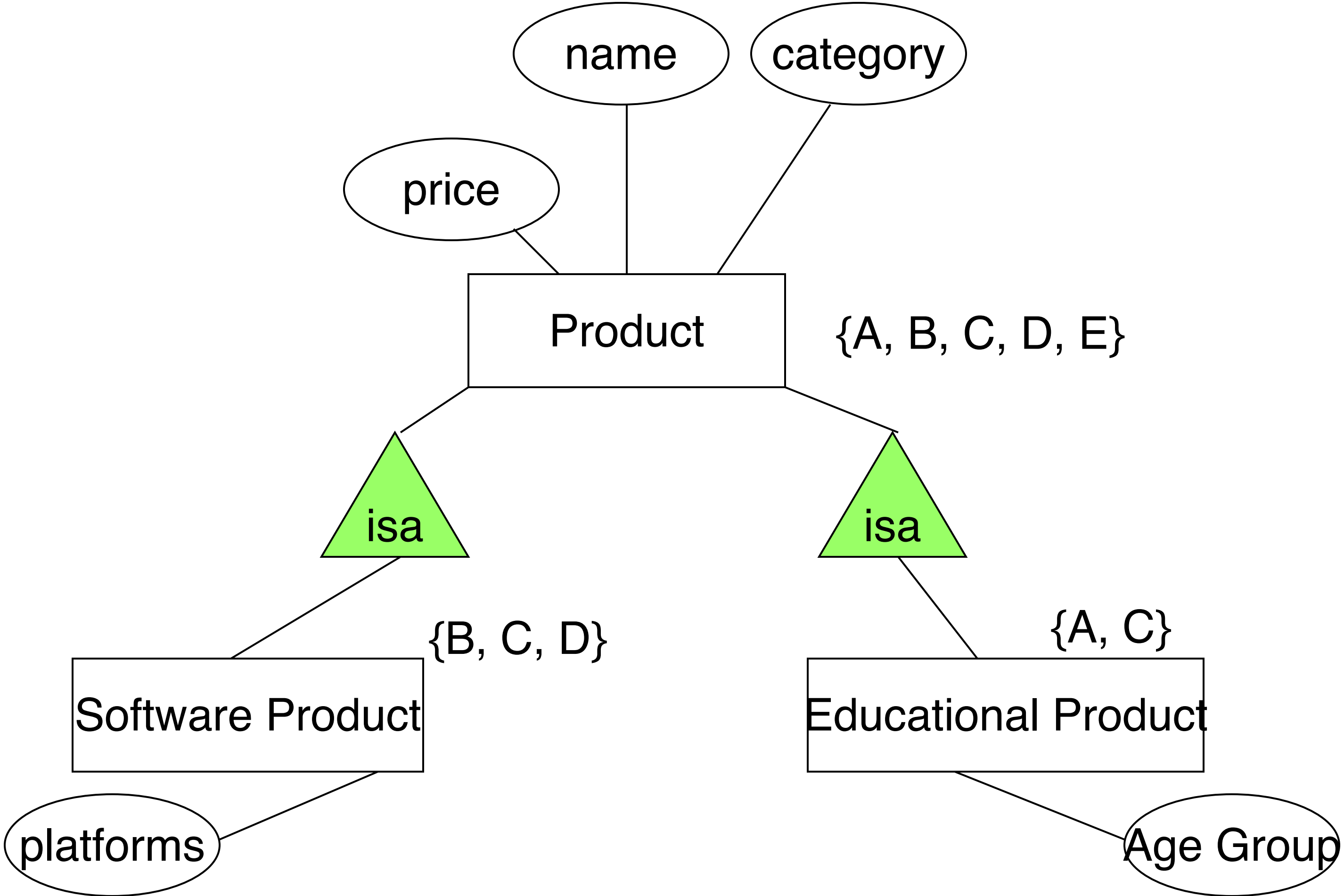
- Subclasses form a tree.
  - I.e., no “multiple inheritance”.

- Why subclasses?
  - Unnecessary to add redundant properties to the root entity set that don’t apply to many of the entities

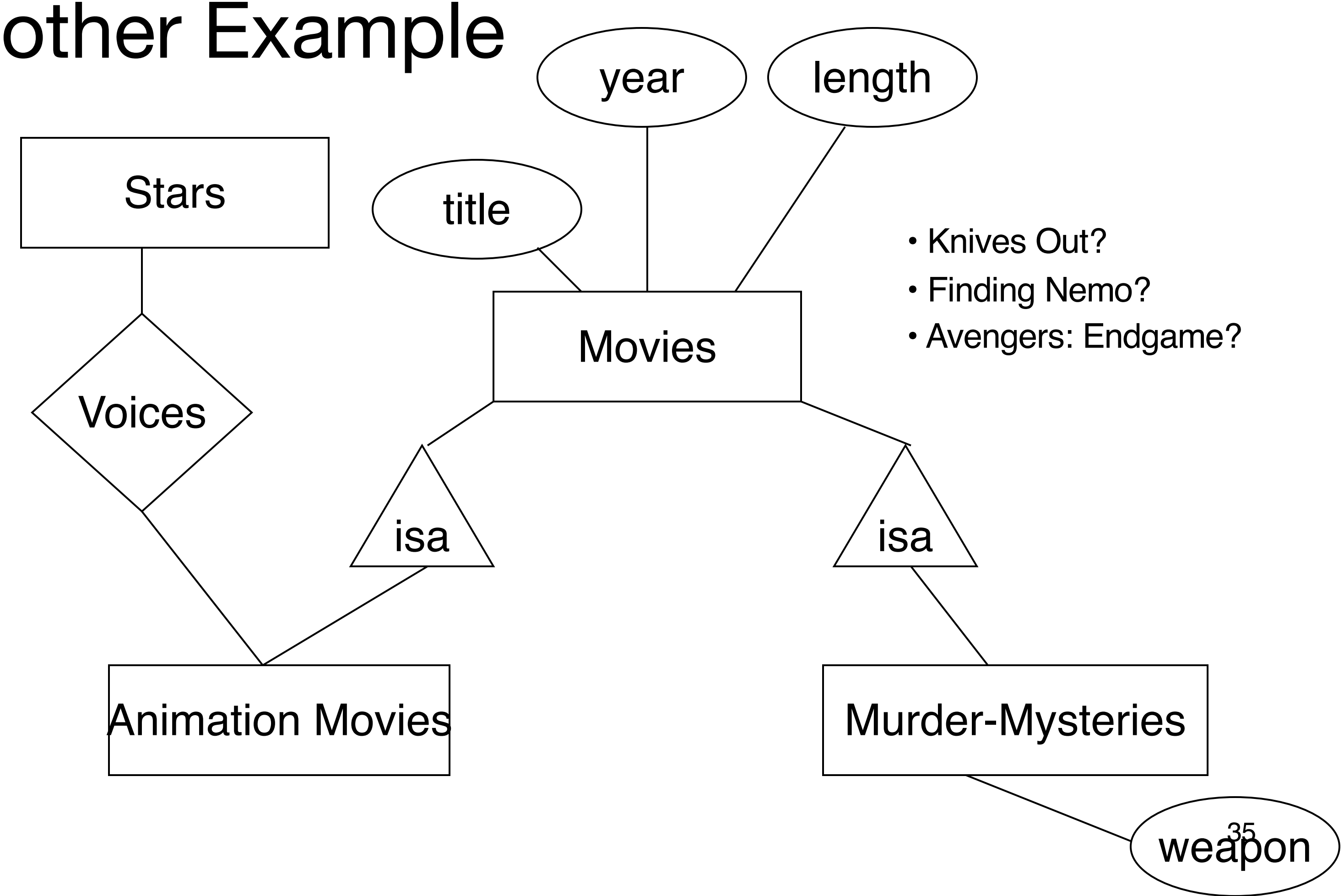
ER Vs. Object Oriented Subclasses

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

Subclasses in ER Diagrams



Another Example



- Knives Out?
- Finding Nemo?
- Avengers: Endgame?

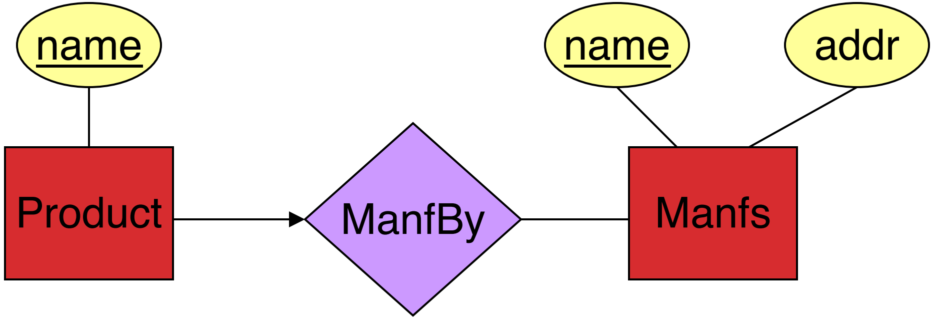
Conceptual Design Using the ER Model

- ER modeling can get tricky!
- Design choices:
    - **Entity** or **attribute**?
    - **Entity** or **relationship**?
    - Relationships: **Binary or ternary?**
- ER Model goals and limitations:
    - Lots of semantics can (and should) be captured.
    - Some constraints cannot be captured in ER.
        - (E.g., partial key constraints in k-ary relationships)
        - We'll refine things in our logical (relational) design

Principle 1: Avoiding Redundancy

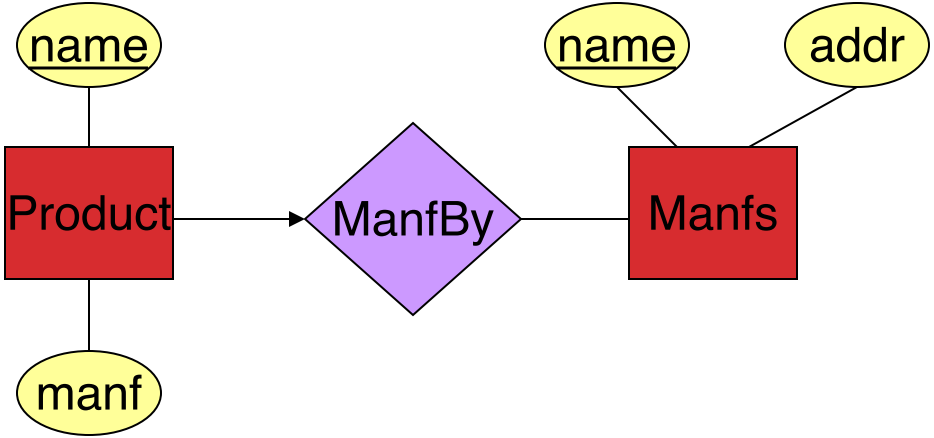
- Redundancy occurs when we say the same thing in two different ways.
- Redundancy wastes space (as we will see) and (more importantly) encourages inconsistency.
  - The two instances of the same fact may become inconsistent if we change one & forget to change the other.

Example: Good



This design gives the address of each manufacturer exactly once.

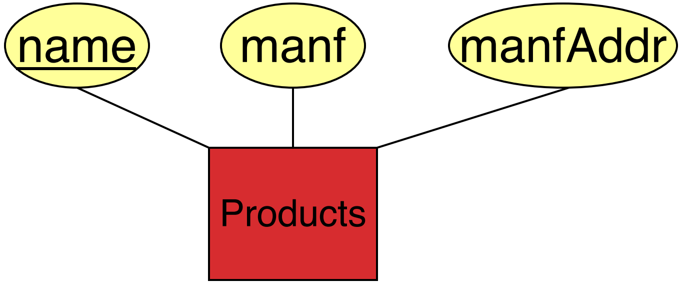
Example: Bad



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

Update issues, Wasteful, …

Example: Bad



This design repeats the manufacturer’s address once for each product (wasteful, update anomalies);

Also loses the address if there are temporarily no products for a manufacturer.

Principle 2: Entity vs. Attribute

- “Address”:
  - attribute of Employees?
  - Entity of its own?
- It depends! Semantics and usage.
  - Several addresses per employee?
    - must be an entity
    - atomic attribute types (no set-valued attributes!)
  - Care about structure? (city, street, etc.)
    - must be an entity! (or at least multiple attributes)
    - atomic attribute types (no tuple-valued attributes!)

Entity Sets Versus Attributes

- Rule: An entity set should satisfy at least one of the following conditions:
  - It is more than the name of something;
    - i.e., it has at least one non-key attribute.

  or

  - It is the “many” in a many-one or many-many relationship.

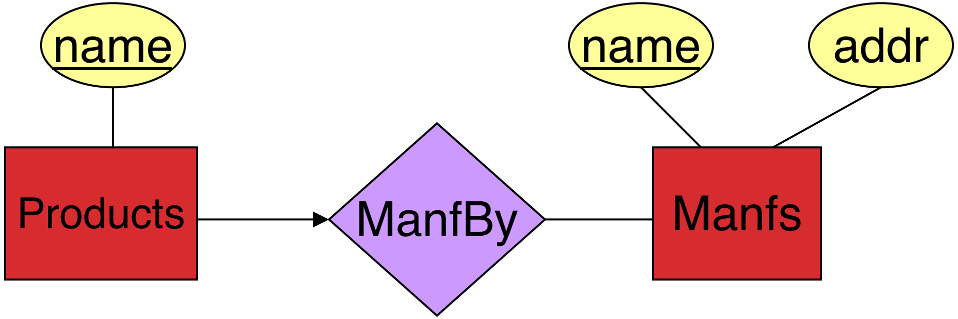
Examples will illustrate why, but also think why these rules make sense.

Example: Good

An E.S. is more than the name of something;

i.e., it has at least one non-key attribute. OR

An E.S is the “many” in a many-one or many-many relationship.



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

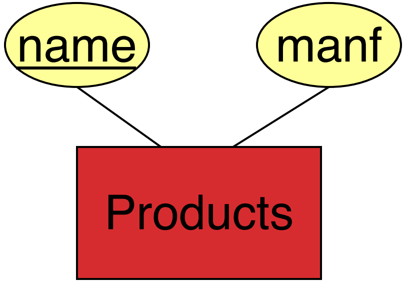
- *Products* deserves to be an entity set because it is the “many” of the many-one relationship *ManfBy*. Can you see why?

Example: Good

An E.S. is more than the name of something;

i.e., it has at least one non-key attribute. OR

An E.S is the "many" in a many-one or many-many relationship.



If we had no manufacturer address information…

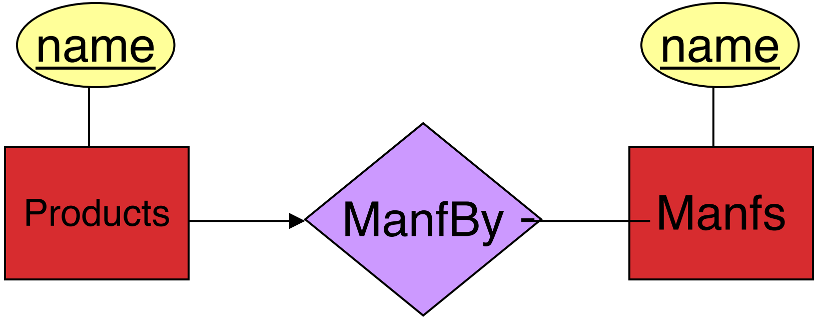
There is no need to make the manufacturer an entity set, because we record nothing about manufacturers besides their name.

Example: Bad

An E.S. is more than the name of something;

i.e., it has at least one non-key attribute. OR

An E.S is the “many” in a many-one or many-many relationship.



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.

E-R Diagram as Wallpaper

- Very common for them to be wall-sized