More Design Issues

1. Subclasses.
2. Keys.
3. Weak entity sets.

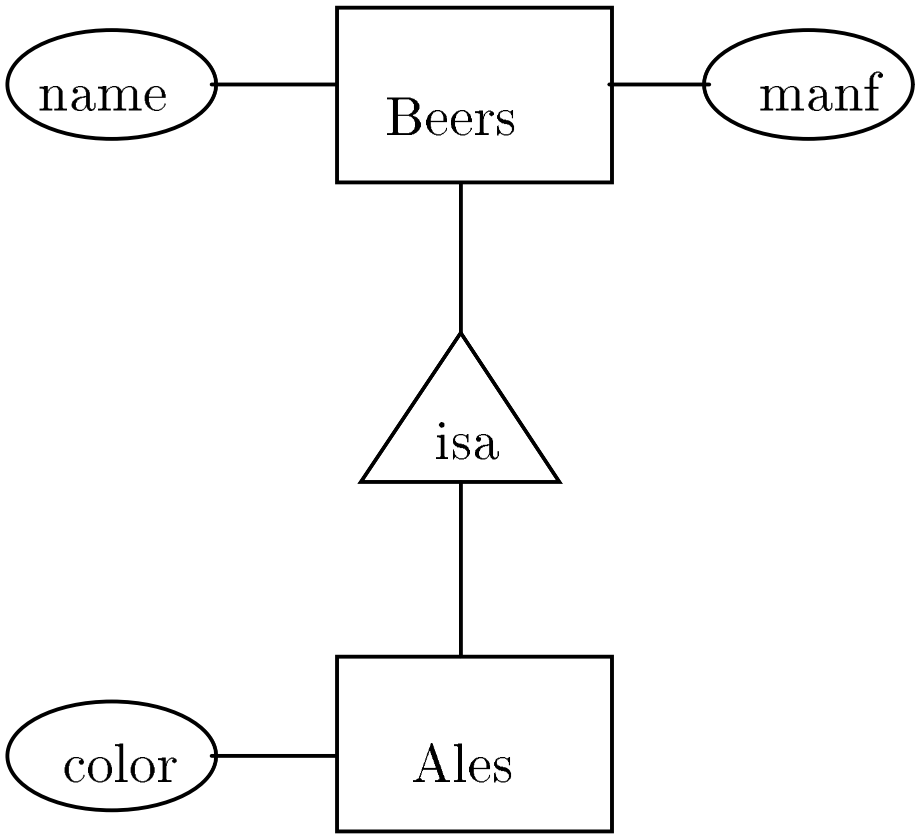
Subclasses

Subclass = special case = fewer entities = more properties.

- Example: Ales are a kind of beer. In addition to the *properties* (= attributes and relationships) of beers, there is a "color" attribute for ales.

E/R Subclasses

- *isa* triangles indicate the subclass relation.



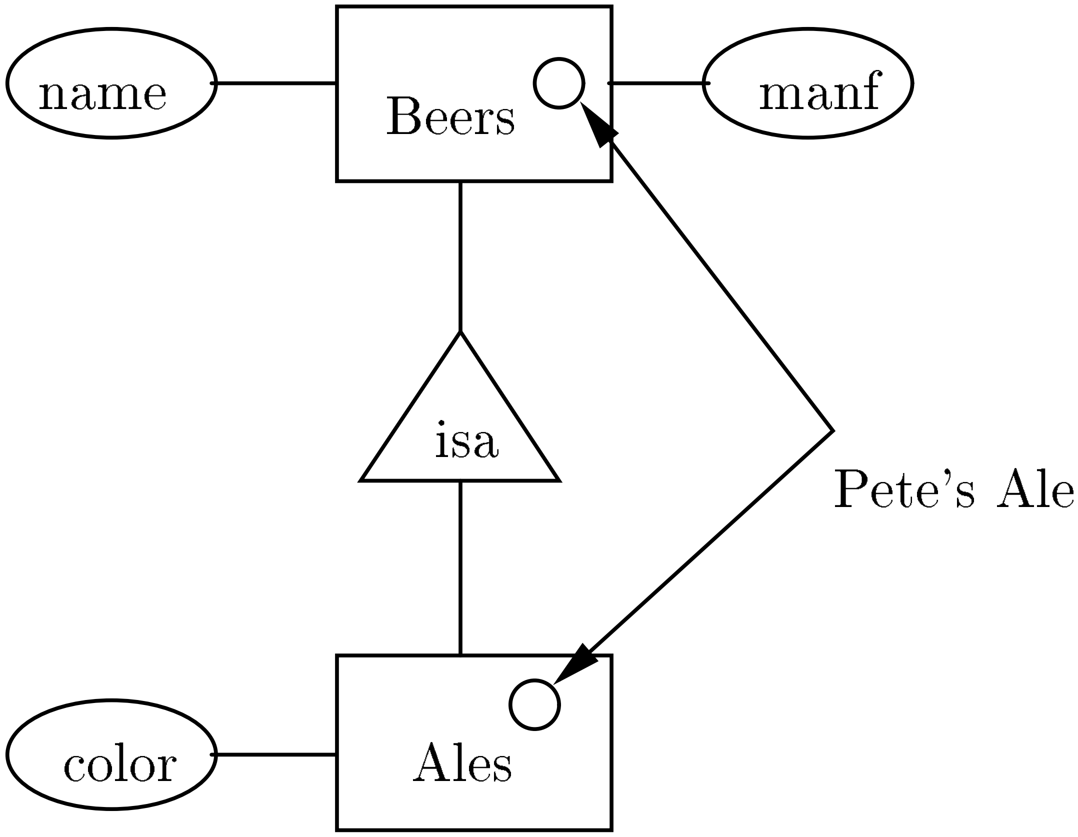
Different Subclass Viewpoints

1. *E/R viewpoint*: An entity has a *component* in each entity set to which it logically belongs.

   - Its properties are the union of the properties of these E.S.

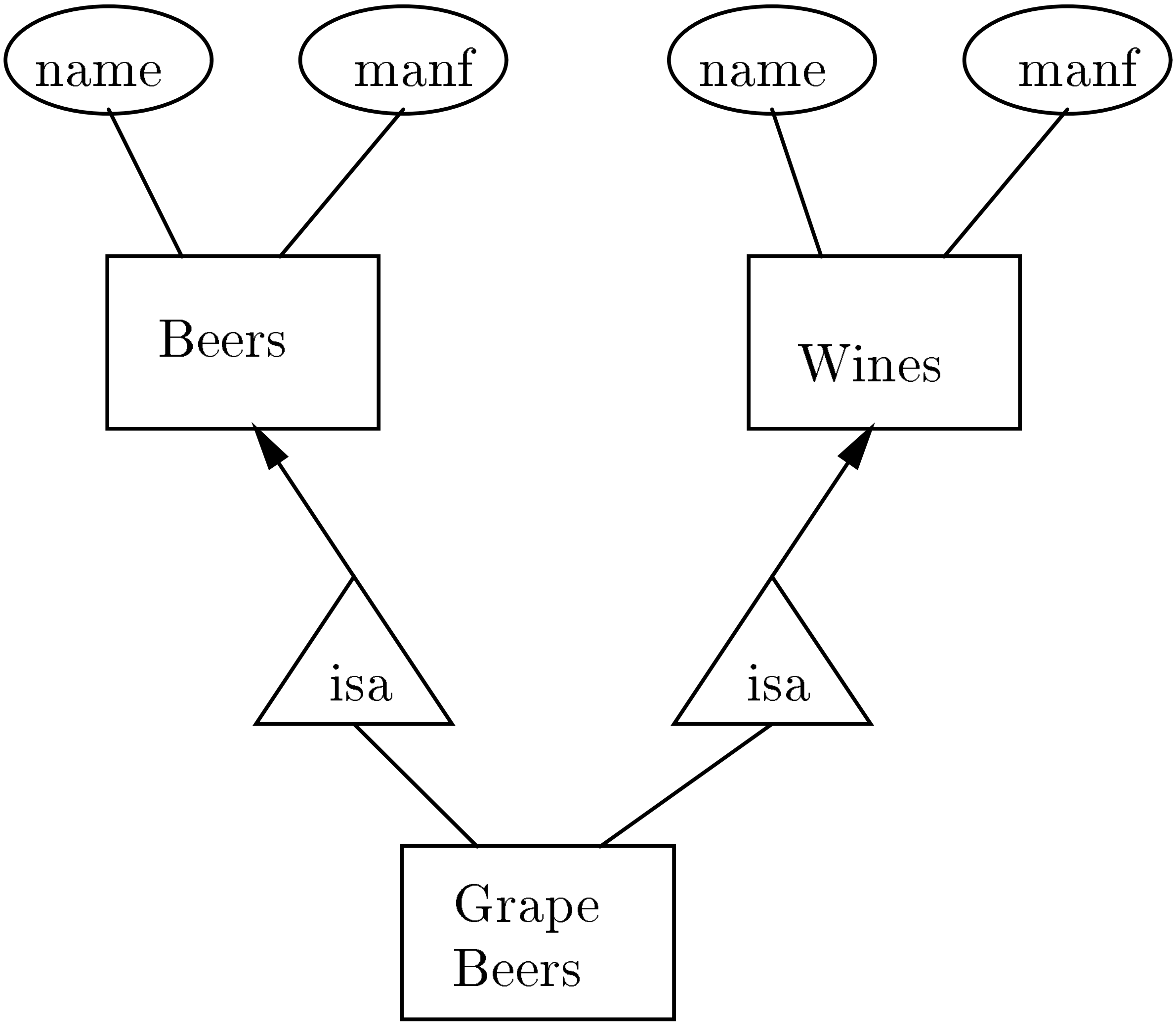
2. Contrasts with *object-oriented viewpoint*: An object (entity) belongs to exactly one class.

   - It *inherits* properties of its superclasses.



Multiple Inheritance

Theoretically, an E.S. could be a subclass of several other entity sets.



Problems

How should conflicts be resolved?

- Example: `manf` means grower for wines, bottler for beers. What does `manf` mean for "grape beers"?
- Need ad-hoc notation to resolve meanings.
- In practice, we shall assume a tree of entity sets connected by *isa*, with all "isas" pointing from child to parent.

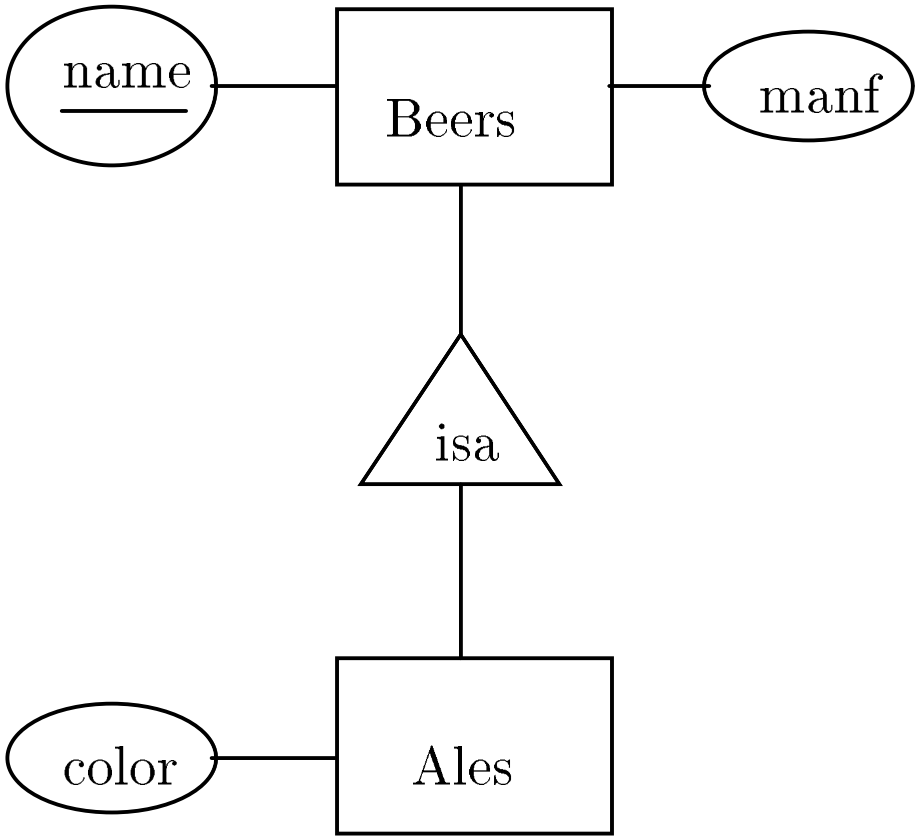
Keys

A *key* is a set of attributes whose values can belong to at most one entity.

- In E/R model, every E.S. must have a key.
  - ❖ It could have more than one key, but one set of attributes is the "designated" key.
- In E/R diagrams, you should underline all attributes of the designated key.

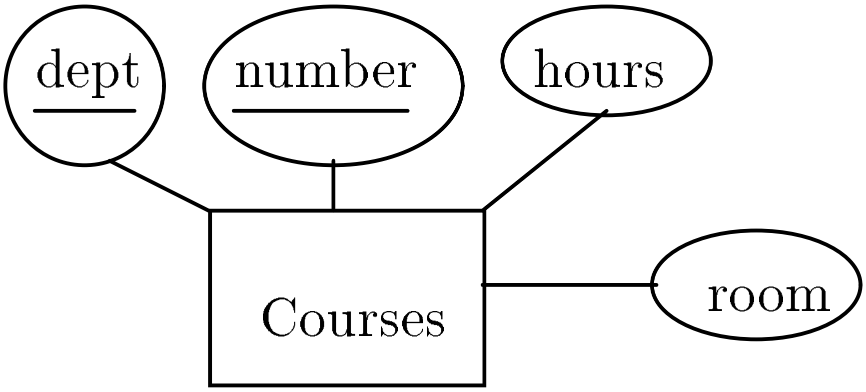
Example

Suppose `name` is key for *Beers*.



- Beer name is also key for ales.
  - In general, key at root is key for all.

Example: A Multiattribute Key



- Possibly, hours + room also forms a key, but we have not designated it as such.

Weak Entity Sets

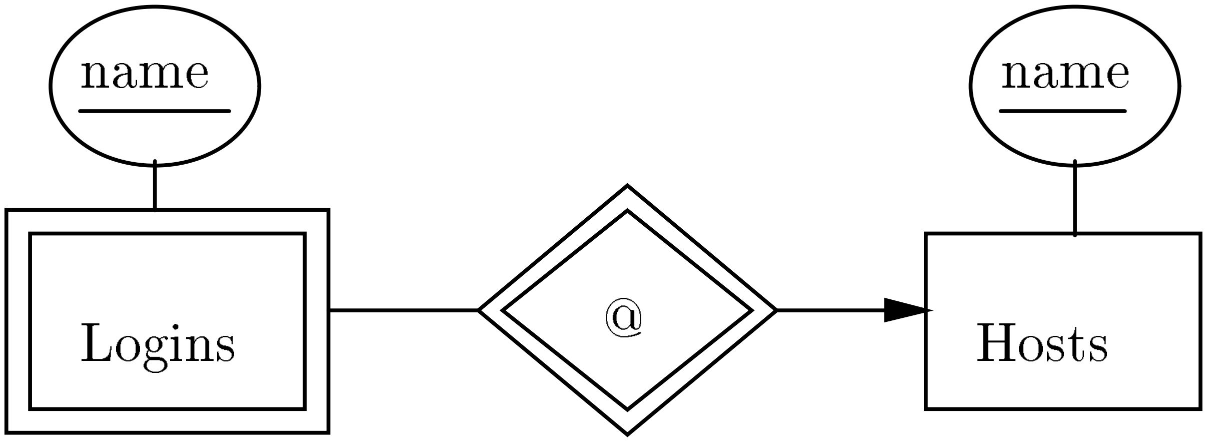
Sometimes an E.S. E's key comes not (completely) from its own attributes, but from the keys of one or more E.S's to which E is linked by a *supporting* many-one relationship.

- Called a *weak* E.S.
- Represented by putting double rectangle around E and a double diamond around each supporting relationship.
- Many-one-ness of supporting relationship (includes 1-1) essential.
  - With many-many, we wouldn't know which entity provided the key value.

Example: Logins (Email Addresses)

Login name = user name + host name, e.g., `ullman@shalmaneser.stanford.edu`.

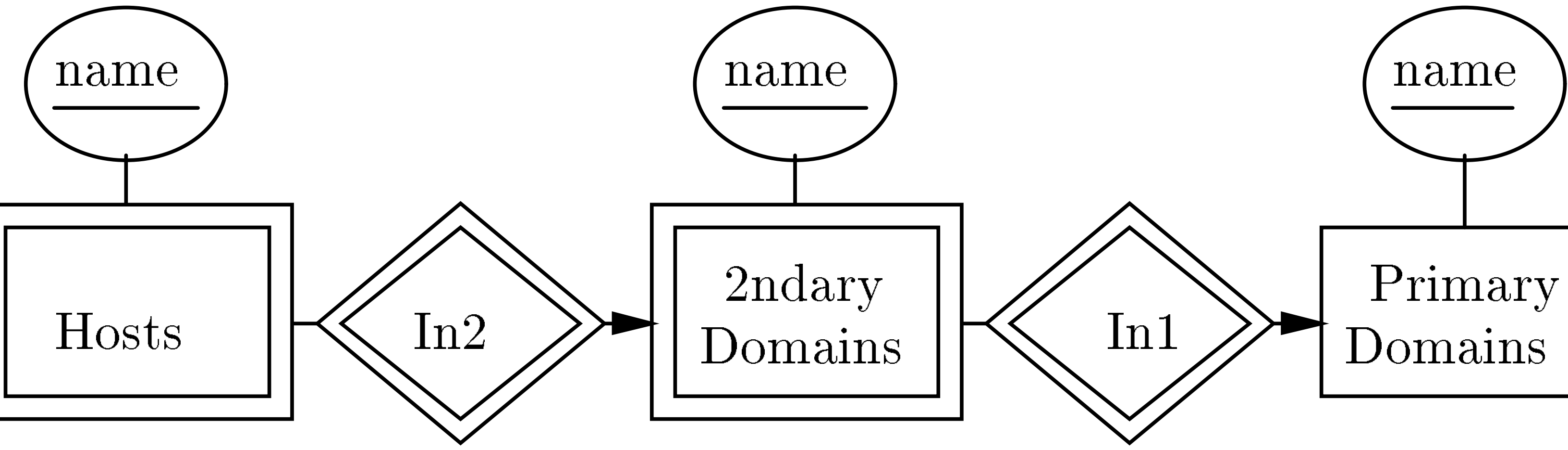
- A "login" entity corresponds to a user name on a particular host, but the passwd table doesn't record the host, just the user name, e.g. `ullman`.
- Key for a login = the user name at the host (which is unique for that host only) + the IP address of the host (which is unique globally).



- Design issue: Under what circumstances could we simply make login-name and host-name be attributes of logins, and dispense with the weak E.S.?

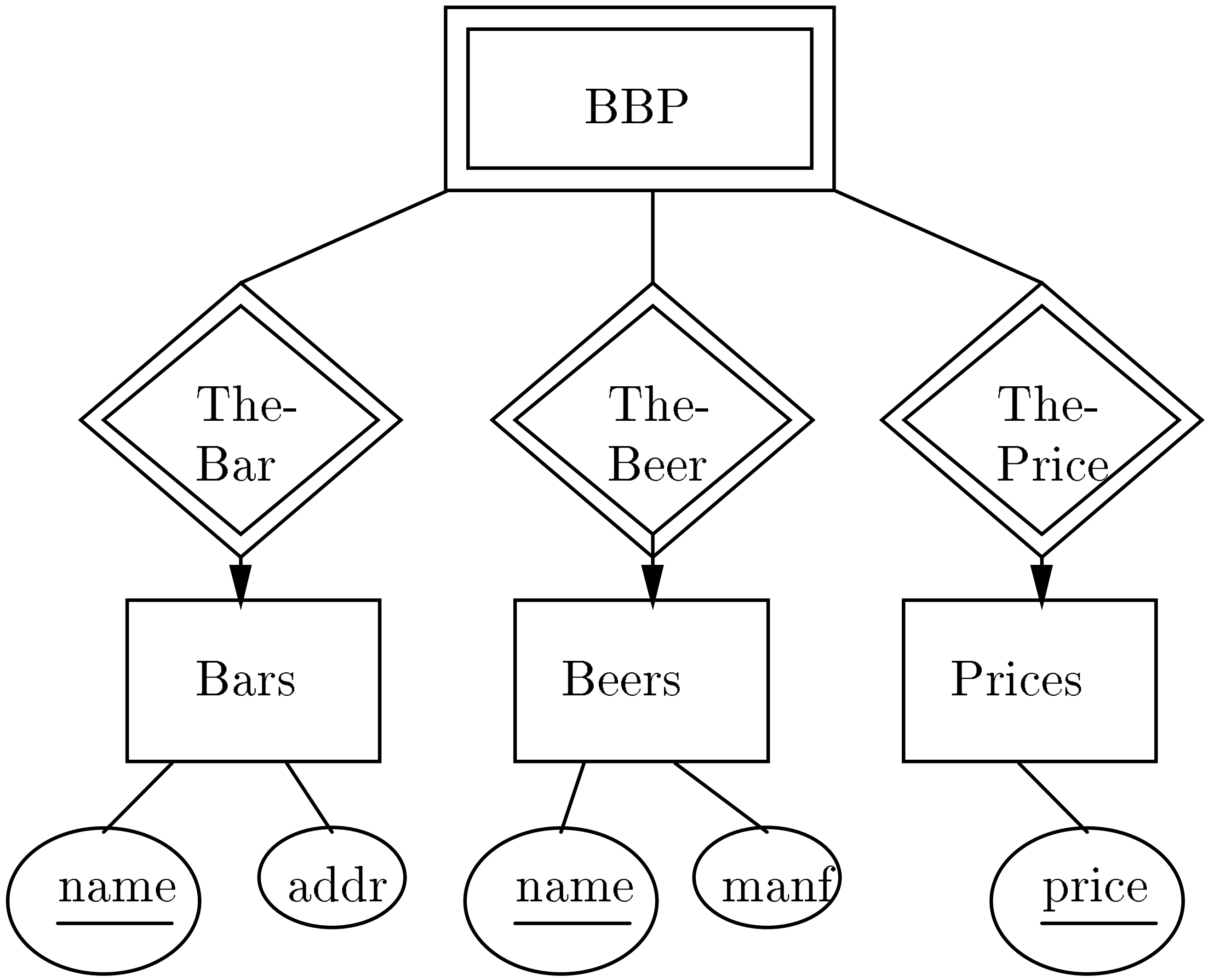
Example: Chain of "Weakness"

Consider IP addresses consisting of a primary domain (e.g., `edu`) subdomain (e.g., `stanford`), and host (e.g. `shalmaneser`).



- Key for primary domain = its name.
- Key for secondary domain = its name + name of primary domain.
- Key for host = its name + key of secondary domain = its name + name of secondary domain + name of primary domain.

All "Connecting" Entity Sets Are Weak



- In this special case, where bar and beer determine a price, we can omit `price` from the key, and remove the double diamond from `ThePrice`.
- Better: `price` is attribute of `BBP`.

Design Principles

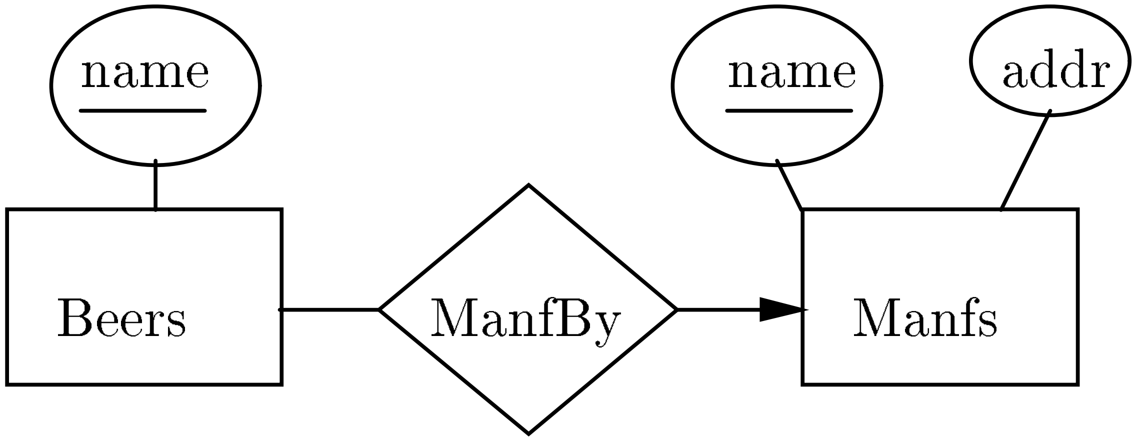
Setting: client has (possibly vague) idea of what he/she wants. You must design a database that represents these thoughts and only these thoughts.

- Avoid redundancy.
    - Wastes space and encourages inconsistency.
    - Intuition: something is redundant if it could be hidden from view, and you could still figure out what it is from the other data.
- KISS = keep it simple, students.
    - Avoid intermediate concepts.

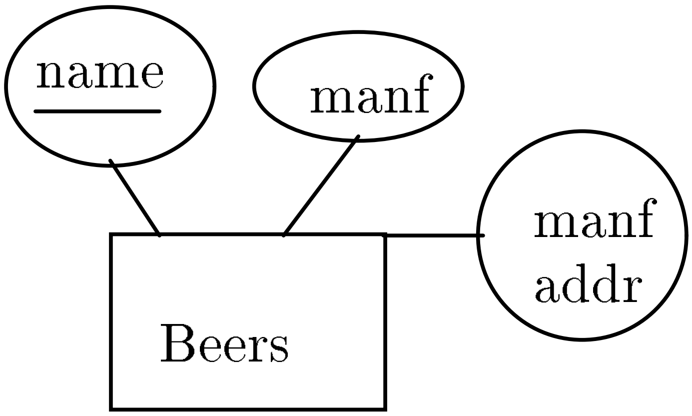
- Faithfulness to requirements.
  - Remember the design *schema* should enforce as many constraints as possible. Don't rely on future data to follow assumptions.
  - Example: If registrar wants to associate only one instructor with a course, don't allow sets of instructors and count on departments to enter only one instructor per course.

Example

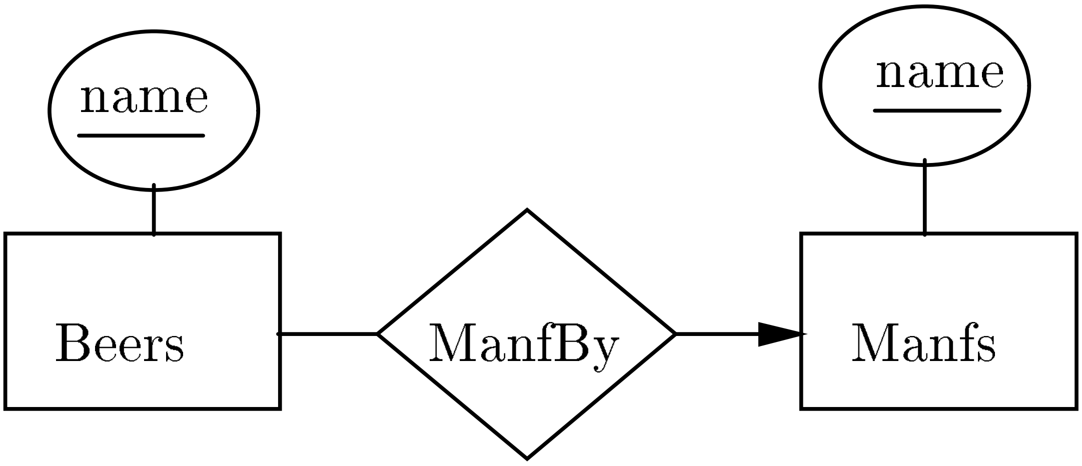
Good:



Bad (redundancy): repeats manufacturer address for each beer they manufacture.



Bad (needless intermediate):



- Question: Why is it OK to have *Beers* with just its key as attribute? Why not make set of beers an attribute of manufacturers?