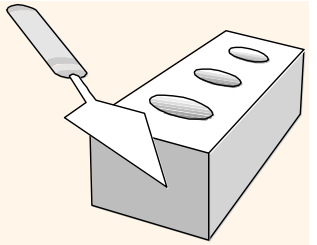
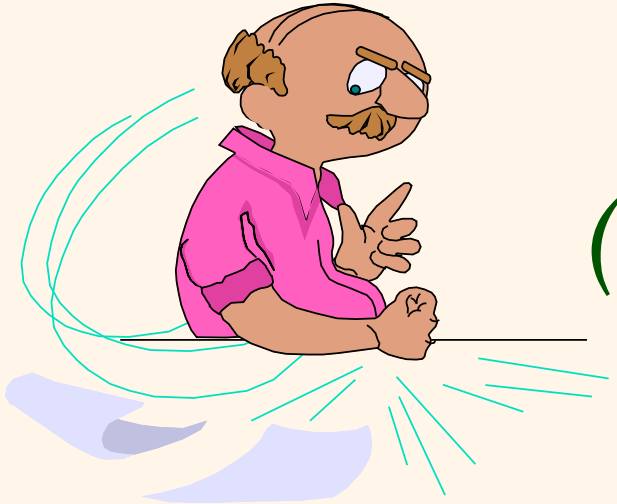


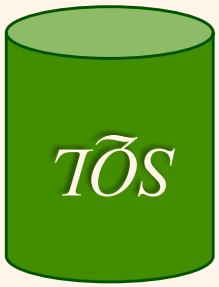
Introduction to Data Management

**** The "Flipped" Edition ****

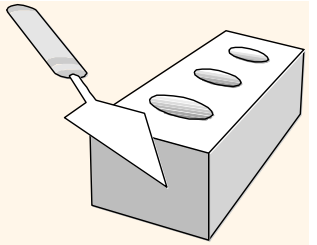


Lecture #3
(ER Model, cont.)

Instructor: Mike Carey
mjcarey@ics.uci.edu

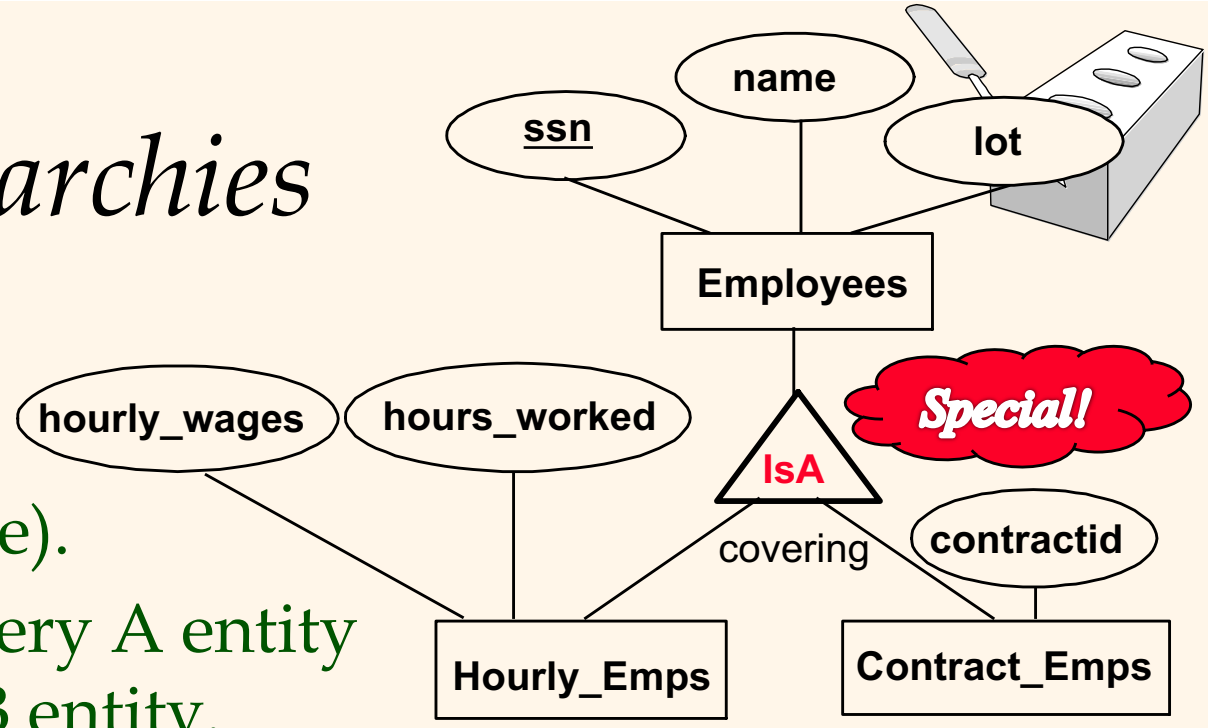


Today's Notices



- Keep watching the course wiki page:
  - http://www.ics.uci.edu/~cs122a/
- And camping out on the Piazza page:
  - http://piazza.com/uci/fall2021/cs122aeecs116/home
- Partnering: Pick a "brainstorming buddy"!
  - Individual HW submissions (***not*** team submissions)
  - See partner-related portion(s) of the first HW assignment
- HW#1 will appear on the wiki by 6PM "today" (Wed)
  - ***Theme:*** **SWOOSH.com** (Simple Web-Optimized Office for Students at Home)
- Complete Quiz 1 on Gradescope before Friday at 4PM
  - Quizzes are free discussion-participation points...!

IsA ("is a") Hierarchies

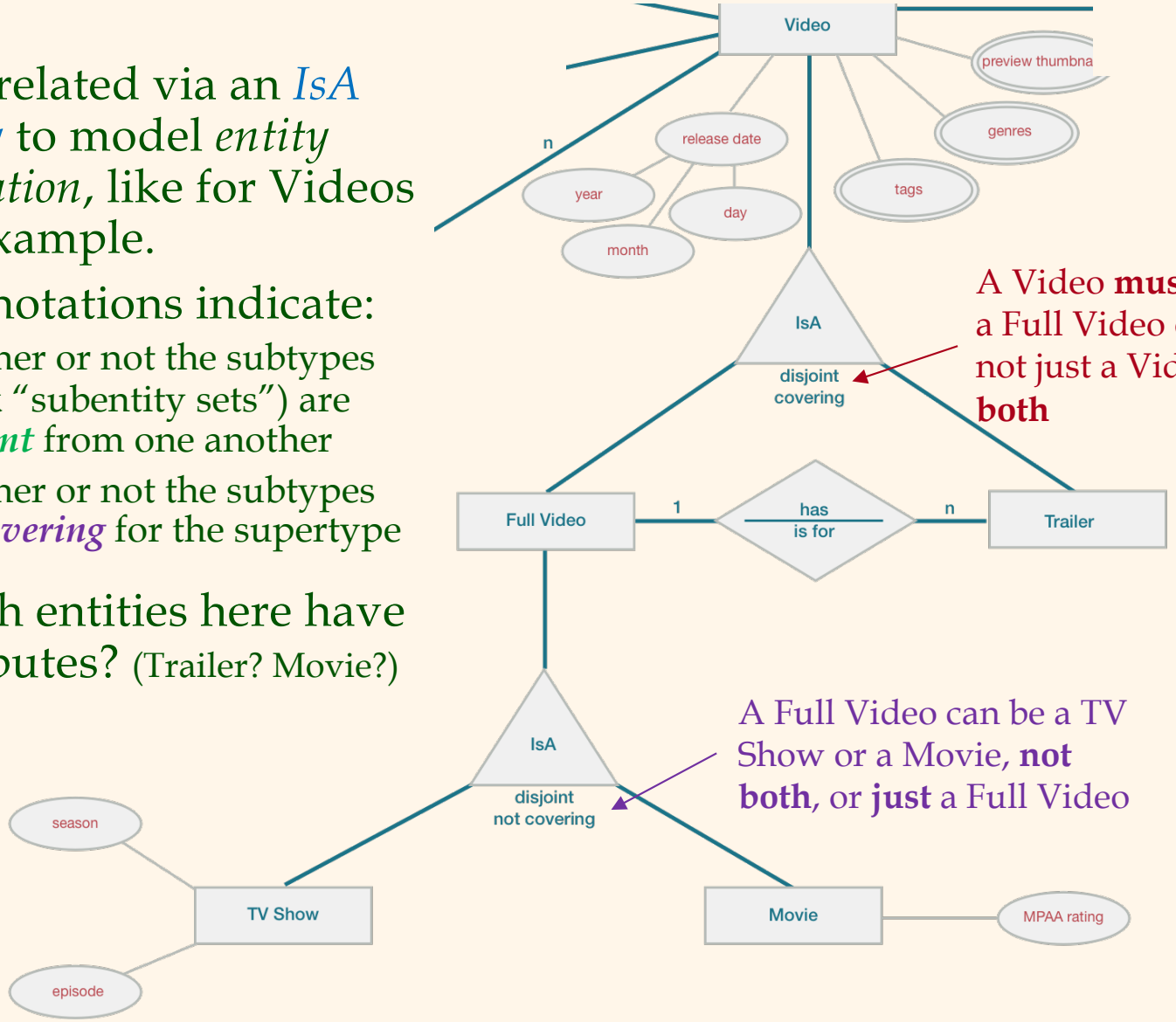


- As in Java or other PLs, ER attributes are inherited (including the key attribute).
- If we declare A **IsA** B, every A entity is also considered to be a B entity.
  - *Covering constraints*: *Must every Employees entity be either an Hourly_Emps or a Contract_Emps entity? (**covering** if so)*
    - ***Ex:*** Hourly_Emps and Contract_Emps **cover** Employees (pick 1 of **2** vs. 1 of 3)
  - *Overlap constraints: Can some Employees entity be an Hourly_Emps **as well as** a Contract_Emps entity? (**disjoint** if not)*
    - ***Ex:*** Hourly_Emps **overlaps** (not disjoint from) Contract_Emps (else pick 1 of the **3**)
  - Reasons for using IsA:
    - To add descriptive attributes specific to a subclass.
    - To identify subclasses that participate in a relationship.
  - Design: specialization (top-down), generalization (bottom-up)

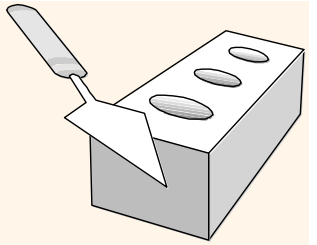
Another IsA Example

- Entities related via an *IsA hierarchy* to model *entity specialization*, like for Videos in this example.
- *IsA's* annotations indicate:
  - Whether or not the subtypes (think "subentity sets") are ***disjoint*** from one another
  - Whether or not the subtypes are ***covering*** for the supertype

Q: So which entities here have which attributes? (Trailer? Movie?)



Aggregation

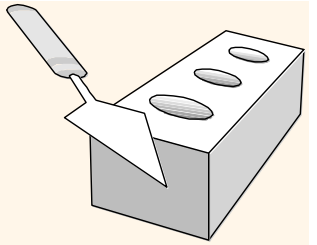


- Used when we have to model a relationship involving (entitity sets and) a *relationship set*.
  - *Aggregation* allows us to treat a relationship set as an entity set for purposes of participating in (other) relationships.

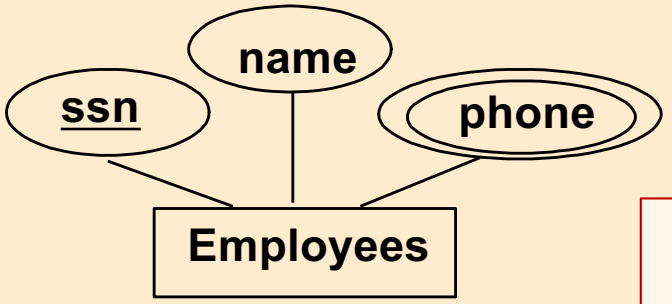
☛ *Aggregation vs. ternary relationship*:

- Monitors is a distinct relationship; it even has its own attribute here.
- Each sponsorship can monitored by zero or more employees (as above).

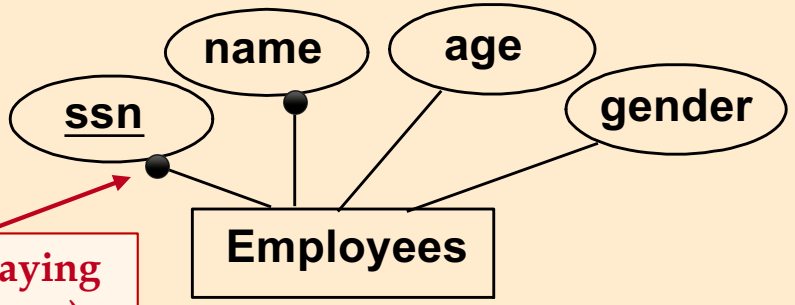
Additional Advanced ER Features



- Multi-valued (vs. single-valued) attributes

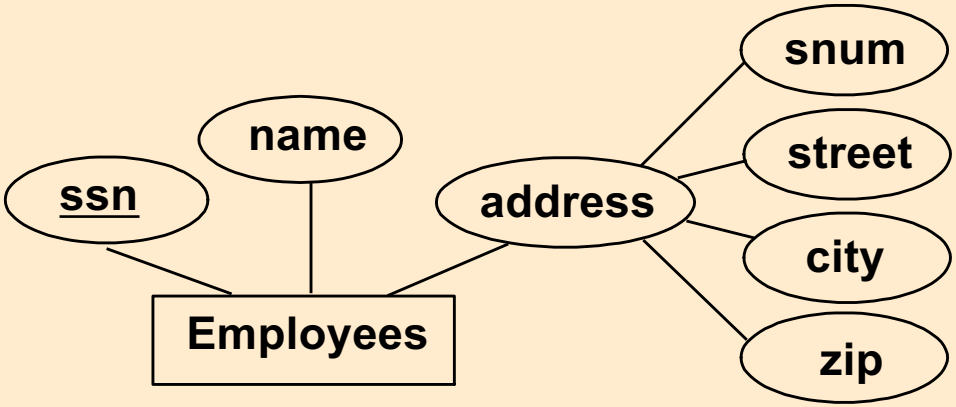


- Mandatory attributes

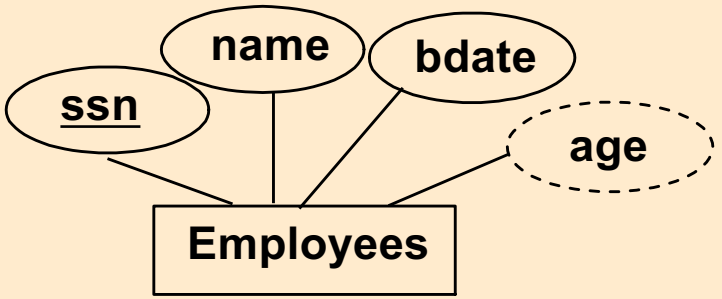


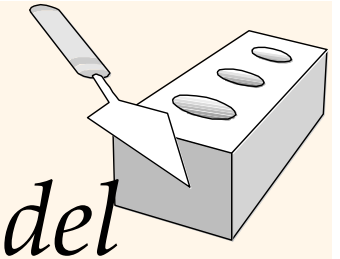
(Goes *w/o* saying for entity keys)

- Composite (vs. atomic) attributes



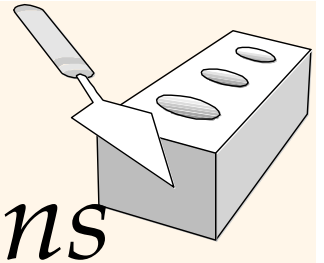
- Derived (vs. base/stored) attributes





Conceptual Design Using the ER Model

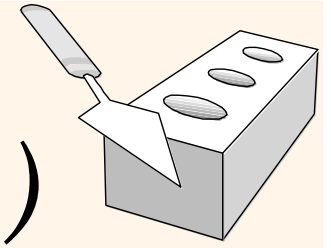
- Design choices:
  - Should a given concept be modeled as an *entity* or as an *attribute*?
  - Should a given concept be modeled as an *entity* or as a *relationship*?
  - Characterizing relationships: Binary or ternary? Aggregation? …
- Constraints in the ER Model:
  - Many of the semantics can (and should) be captured
  - But – not every constraint can be captured by ER diagrams. (*Ex:* Department heads from earlier…!)



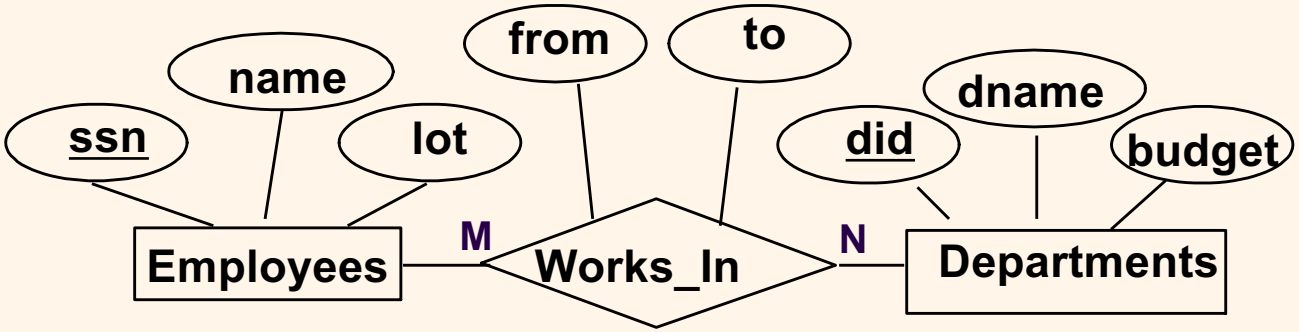
Advanced Attribute Considerations

- Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?
- Depends on how we want to use addresses, on the data semantics, and the available model features:
  - If we have *several* addresses per employee, *address* would be a multivalued attribute – or an entity, if we stick just to basic E-R concepts (which can't be set-valued w/o advanced modeling).
  - If address *structure* (city, street, etc.) is important, e.g., to query for employees in a given city, *address* should be modeled as a composite attribute – or an entity (as attribute values must be *atomic* otherwise) – i.e., it shouldn't be an address string.
  - If the address itself is *logically separate* (e.g., representing a property) and *"refer-able"*, it's rightly an entity in any case!

Attribute Considerations (Cont'd.)

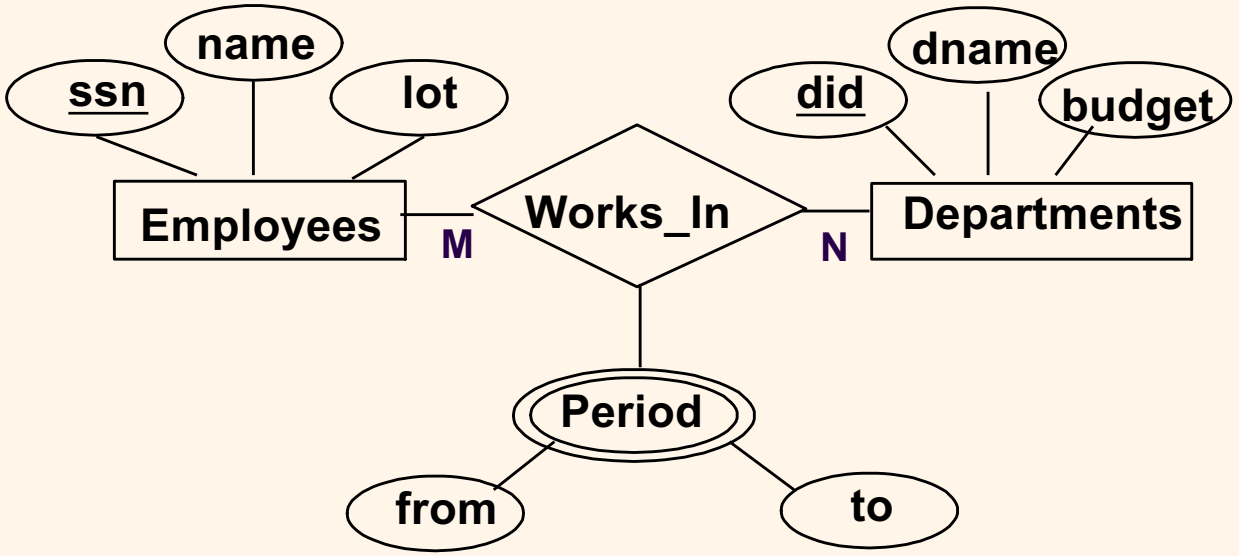


- Works_In here does not allow an employee to work in a department for two or more periods. ***(Q: Why...?)***

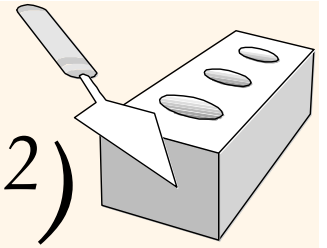


Pause the video and think ... then wait until I tell you start it again. ☺

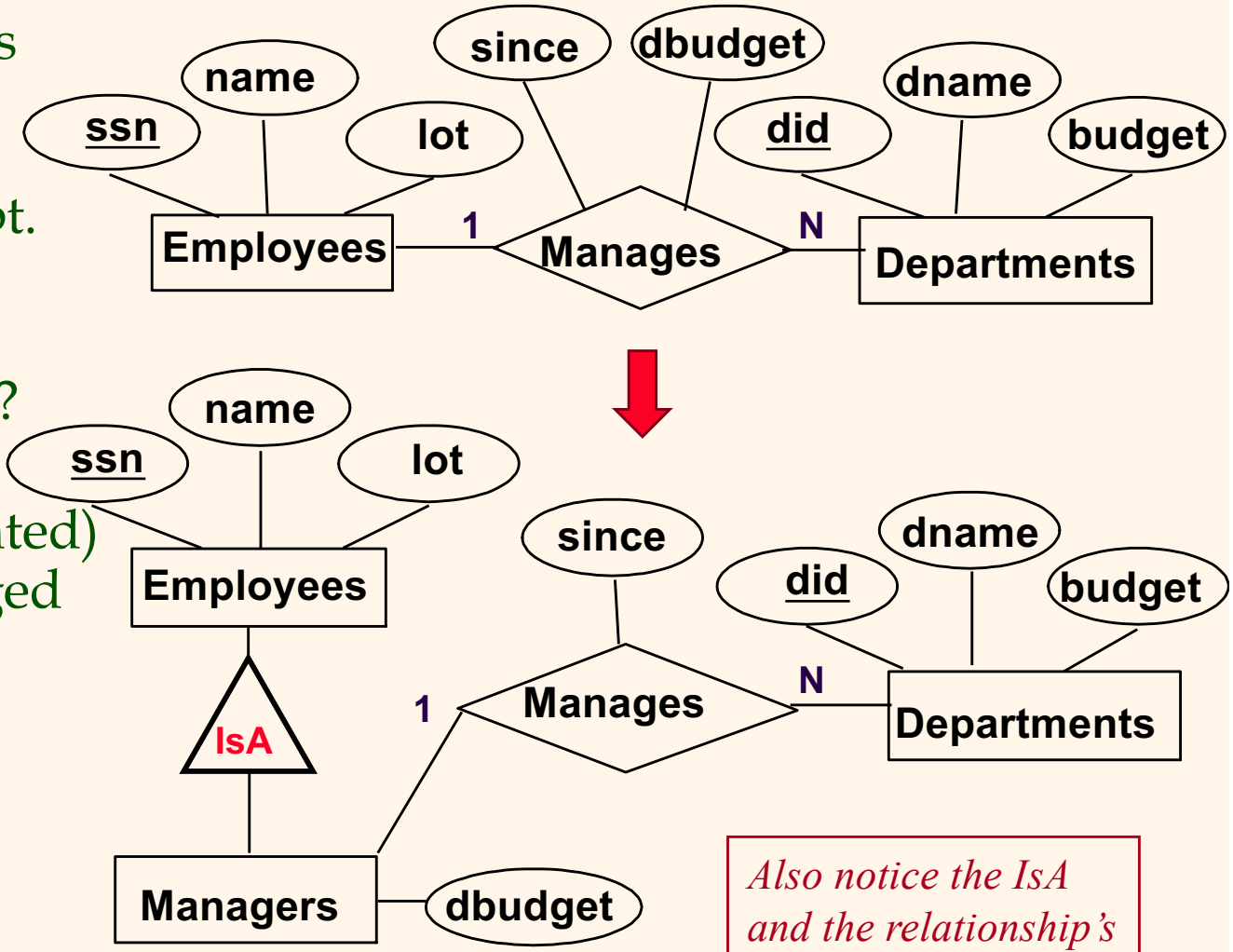
- Similar to wanting several addresses for an employee, here we want to record *several values of the descriptive attributes for each instance of a "relationship".* This can be achieved by using a *multivalued* relationship attribute.



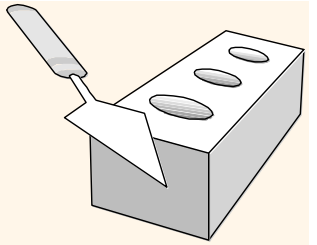
Attribute Considerations (Cont'd.[2])



- ER diagram on the right is okay if a manager gets a separate discretionary budget (dbudget) per dept.
- What if a manager gets a discretionary budget that covers all managed depts?
  - Redundancy: *dbudget* could be stored (repeated) with each dept managed by the manager.
  - Misleading: Suggests *dbudget* is associated with department-mgr combination.

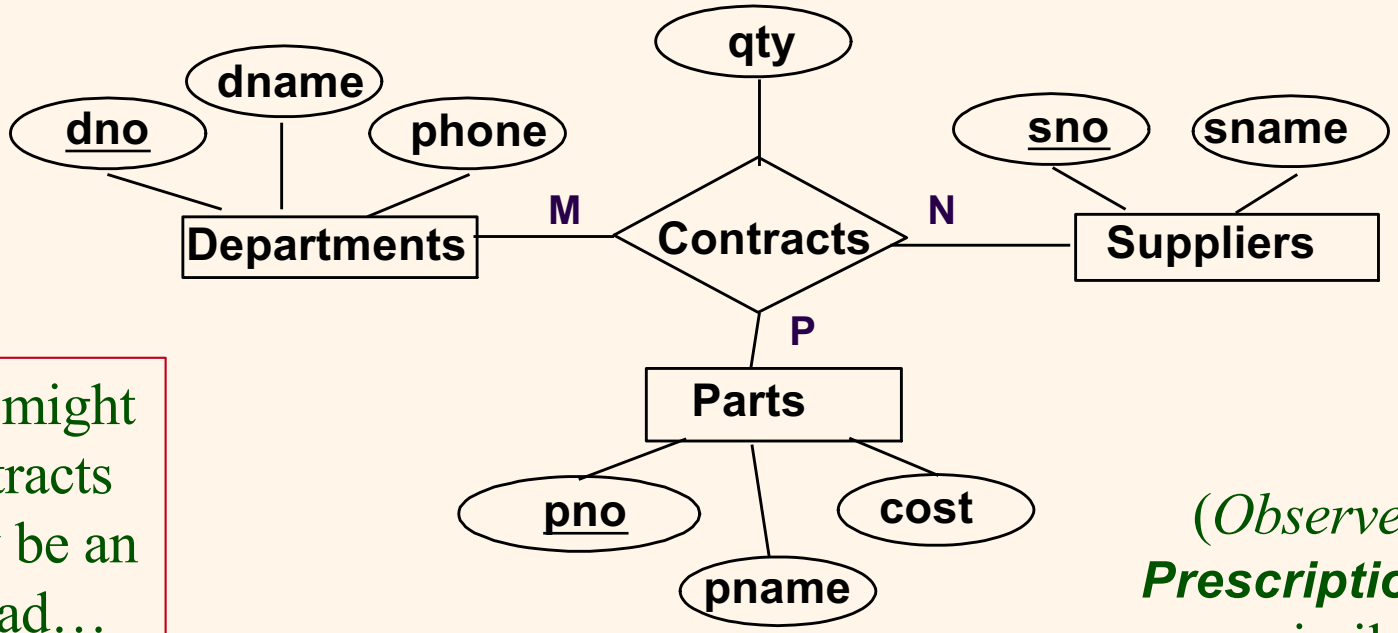


Also notice the IsA and the relationship's placement...



Binary vs. Ternary Relationships

- An example where ternary is needed: a ternary relation Contracts relates the entity sets Parts, Departments and Suppliers, and has descriptive attribute *qty*:

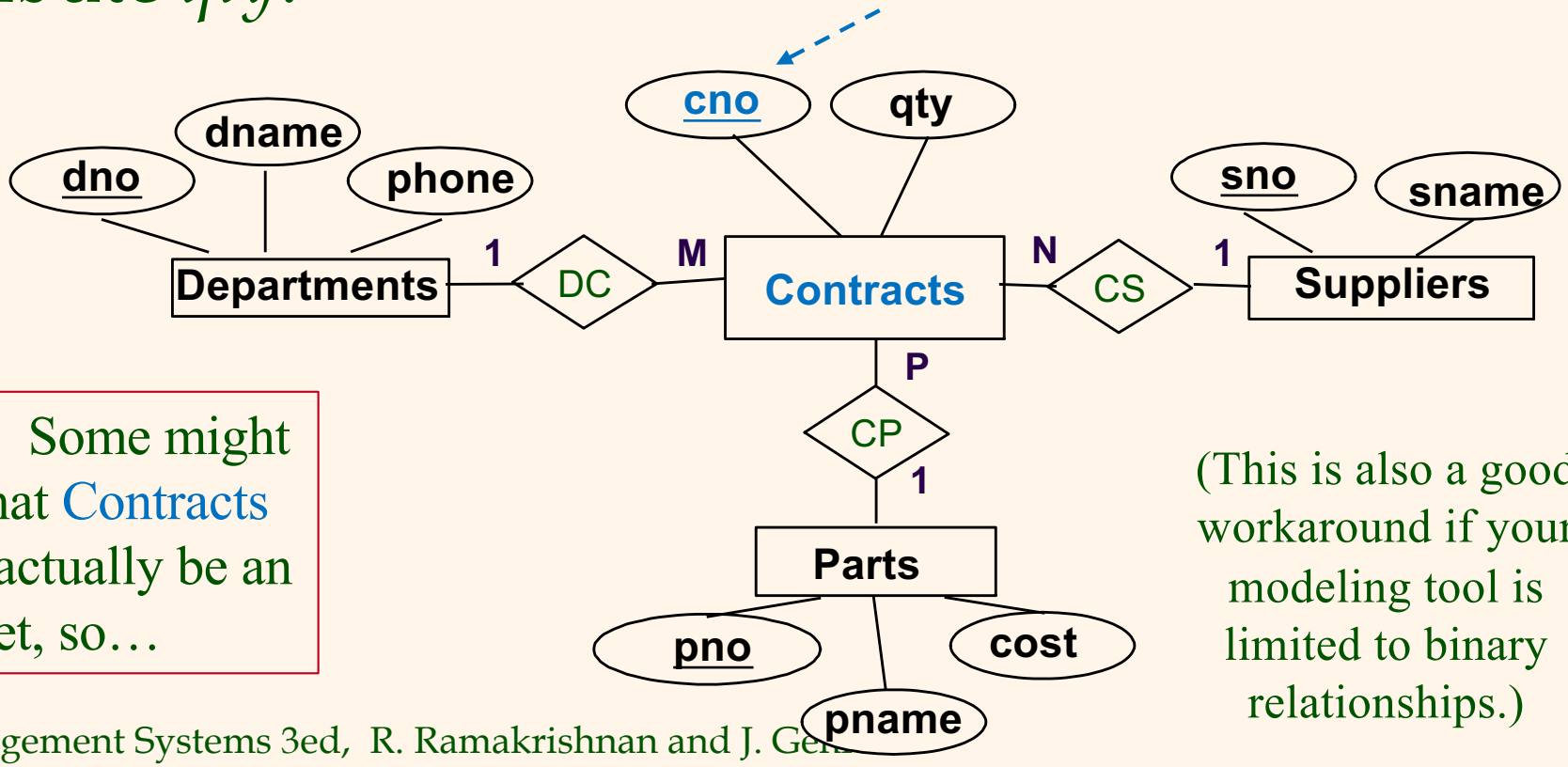


NOTE: Some might argue that Contracts should actually be an **entity set** instead…

(*Observe*: ***Prescriptions*** was similar)

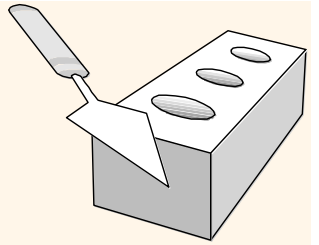
Binary vs. Ternary Relationships (Cont'd.)

- What the entity set perspective would lead to: an entity set Contracts related to the entity sets Parts, Departments and Suppliers, with the descriptive attribute *qty*:



NOTE: Some might argue that Contracts should actually be an entity set, so…

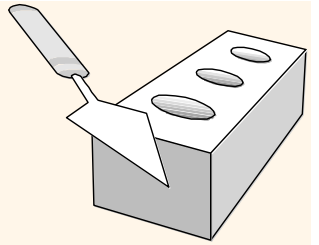
(This is also a good workaround if your modeling tool is limited to binary relationships.)



Database Design Process (Flow)

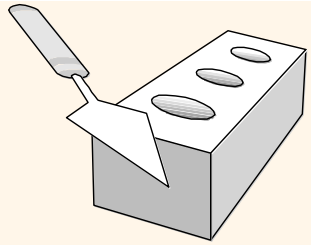
- Requirements Gathering (*interviews*)
- Conceptual Design (*using ER*)
- Platform Choice (*which DBMS?)*
- Logical Design (*for target data model*)
- Physical Design (*for target DBMS & workload*)
- Implement (*and test, of course* ☺)

(Expect backtracking, iteration, and also incremental adjustments – and, we will actually be giving you a bit of practice with that last one in the next few HW assignments...! ☺)



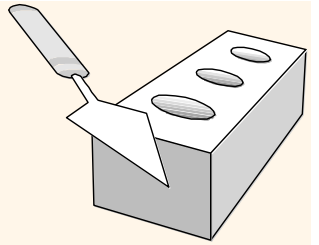
Summary of Conceptual Design

- *Conceptual design* follows *requirements analysis*
  - Yields a high-level description of data to be stored.
- *ER model* popular for conceptual design
  - Constructs are expressive, close to the way people think about their applications.
- Basic constructs: *entities*, *relationships*, and *attributes* (of entities and/or relationships).
- Additionally: *weak entities*, *ISA hierarchies*, *aggregation*, and *multi-valued, composite* and/or *derived* attributes.
- Note: Many variations on the ER model (and many ER notations in use as well) – and also, UML...



Summary of ER (Cont'd.)

- Several kinds of integrity constraints can be expressed in the ER model: *cardinality constraints*, *participation constraints*, also *disjoint/covering constraints* for IsA hierarchies. Some *foreign key constraints* are implicit in the definition of a relationship set (more about key constraints soon).
  - Some constraints (notably, *functional dependencies*) cannot be expressed in the ER model.
  - Constraints play an important role in determining the best database design for an enterprise!



Summary of ER (Cont'd.)

- ER design is *subjective*. There are often many ways to model a given scenario! Analyzing alternatives can be tricky, especially for a large enterprise. Common choices include:
  - Entity vs. attribute, entity vs. relationship, binary or *n*-ary relationship, whether or not to use an IsA hierarchy, and whether or not to use aggregation.
- Ensuring good database design: The resulting relational schema should be analyzed and refined further. For this, FD information and normalization techniques are especially useful (coming soon).