COMP-421
The Entity-Relationship Model

Steps in Database Design

1) **Requirement Analysis**
   - Identify the data that needs to be stored
     - data requirements
   - Identify the operations that need to be executed on the data
     - functional requirements
2) **Conceptual Database Design**
   - **Use a semantic data model to develop semantic schema**
3) Map semantic schema to relational schema

Requirement Analysis

- Similar to first phase of general software design
- focus: data & functional requirements
- Questions to ask:
  - What are the **entities** and **relationships** in the application?
  - What information about these entities and relationships should we store in the database?
  - What are the integrity constraints or business rules that hold?
  - What operations do we want to execute on the entities and relationships?
- Tools known from SE
  - data-flow diagrams, sequence diagrams…
- In this course:
  - half-formal, structured specification in plain English

Requirement Analysis: Minerva/Banner University Database

- The database captures crucial information about university objects and subjects such as students, employees, financial data, courses and much more.
- The data can be accessed via interfaces that allow
  - students access to registration, address changes, fee assessment, transcripts;
  - Instructors access to enrollment information, grade input
  - Principal researchers access to grant information, travel reimbursements, …

Requirement Analysis: Student Database

- The database contains information about *persons:*
  - Data:
    - each person has an identifying ID
    - each person has a PIN, name, permanent address + phone number, 0 or more emergency contacts
  - Functionality
    - all data can be viewed and changed
- A person can be a *student.* A student has
  - Data:
    - …
  - Functionality:
    - Register for course…
- A person can be an instructor.
  - Functionality:
    - Can teach a course
    - Can enter grades

Student Database (contd)

- A course
  - Data
    - Term: the term the course takes place
    - course id (special format)
    - CRN
    - type
    - Credits: how many credits; typically 1-4
    - title
    - days
    - time
    - capacity
    - enrollment
    - one or more *instructors*
    - *room*
    - …
  - Some properties are specific for one term; others always hold for this course

Student Database (contd)

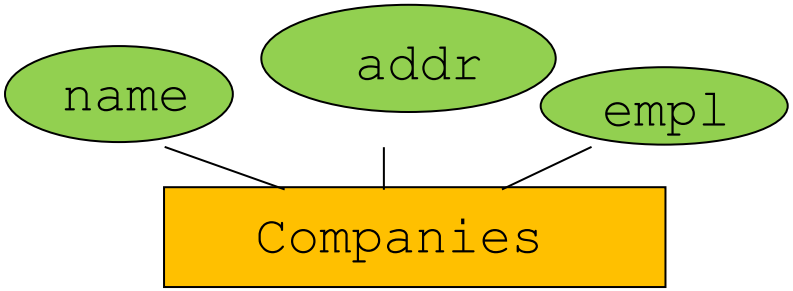
- A course
  - Functionality:
    - Students can register
    - Course can be cancelled
    - Course can be assigned instructor

Semantic Data Model: Entity-Relationship Model (ER)

- The *E/R model* is a language that allows for a pictorially description of the data determined through the requirement analysis
  - Conceptually similar to UML class diagrams
- An *E/R schema* is a representation of the structure of the data of an application using the E/R model
- An ER schema should be understandable by non-computer scientists.
- The main concepts of the E/R model are entity sets that group entities and relationship sets that group relationships
- An ER schema can be translated into the relational schema in a quite straightforward way.
- Many dialects
  - We use the one from the book….

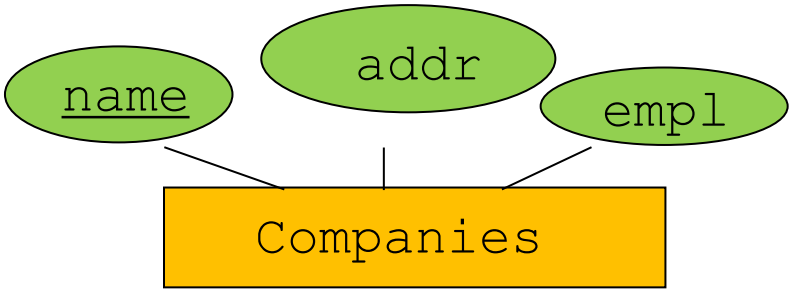
Entity

- **Entity**: Real-world object distinguishable from other objects. An entity is described using a set of **attributes.**
  - similar to an object/instance in OO
- **Entity Set:** A collection of similar entities, e.g., all companies registered in Quebec (similarity to a "class" in OO)
  - All entities in an entity set have the same attributes (until we consider ISA hierarchies later…). An attribute describes a property of the entity set.
  - An entity set must have a **key:** (underlined)
    - Minimum set of attributes whose values *uniquely identify* an entity in the set
    - What would be a good key for Companies
    - Often: artificial key



Entity set = rectangle, Attribute = oval

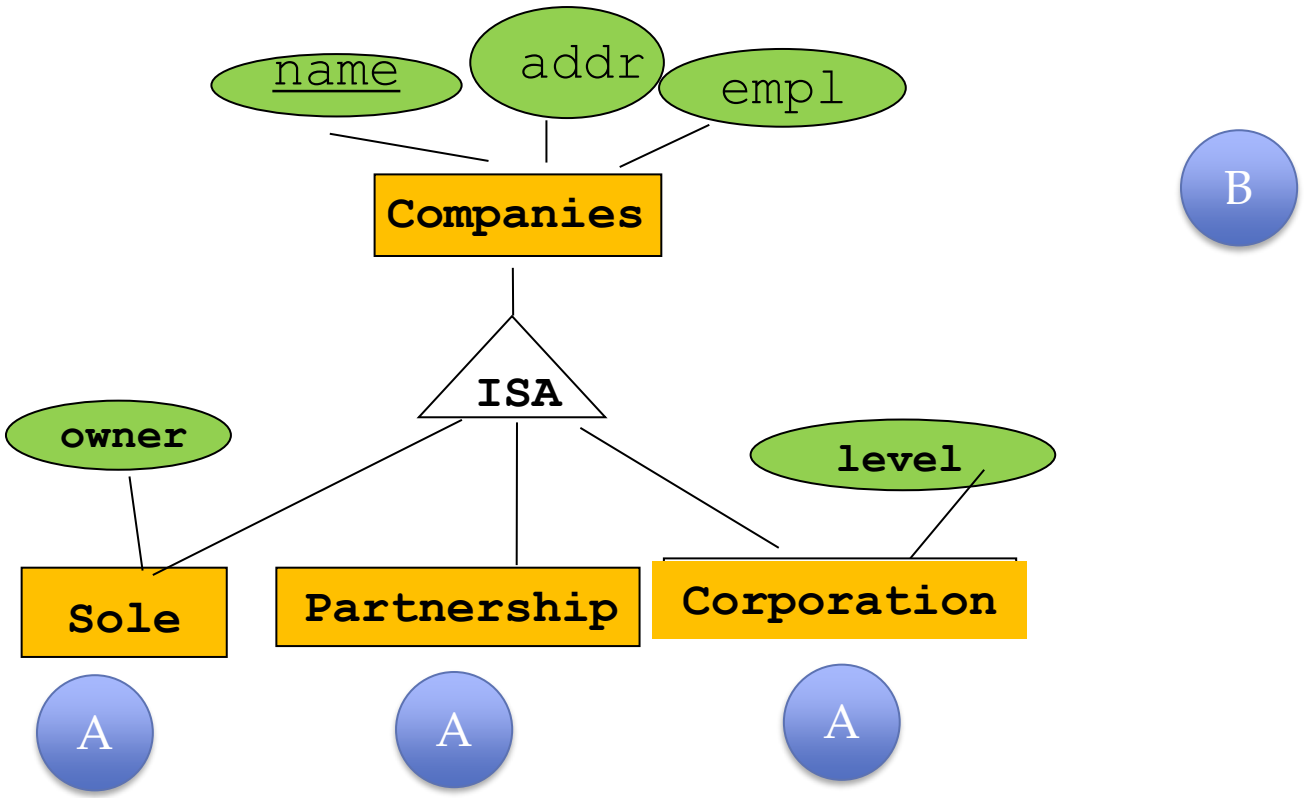
Basic Entity Sets → Tables



name	addr	empl
BiggestEngCompanyEver	Eng. Av., H3X…	25,000
BiggestConstCommpanyEver	Constr. St. H4E…	47,000
NoNameCompany	Whatever St., …	200
…		

ISA (“is a”) Hierarchies: Subclasses

- Subclass = special case = fewer entities = more properties
- “A ISA B”, then every A entity is also a B entity
  - Key only in B.

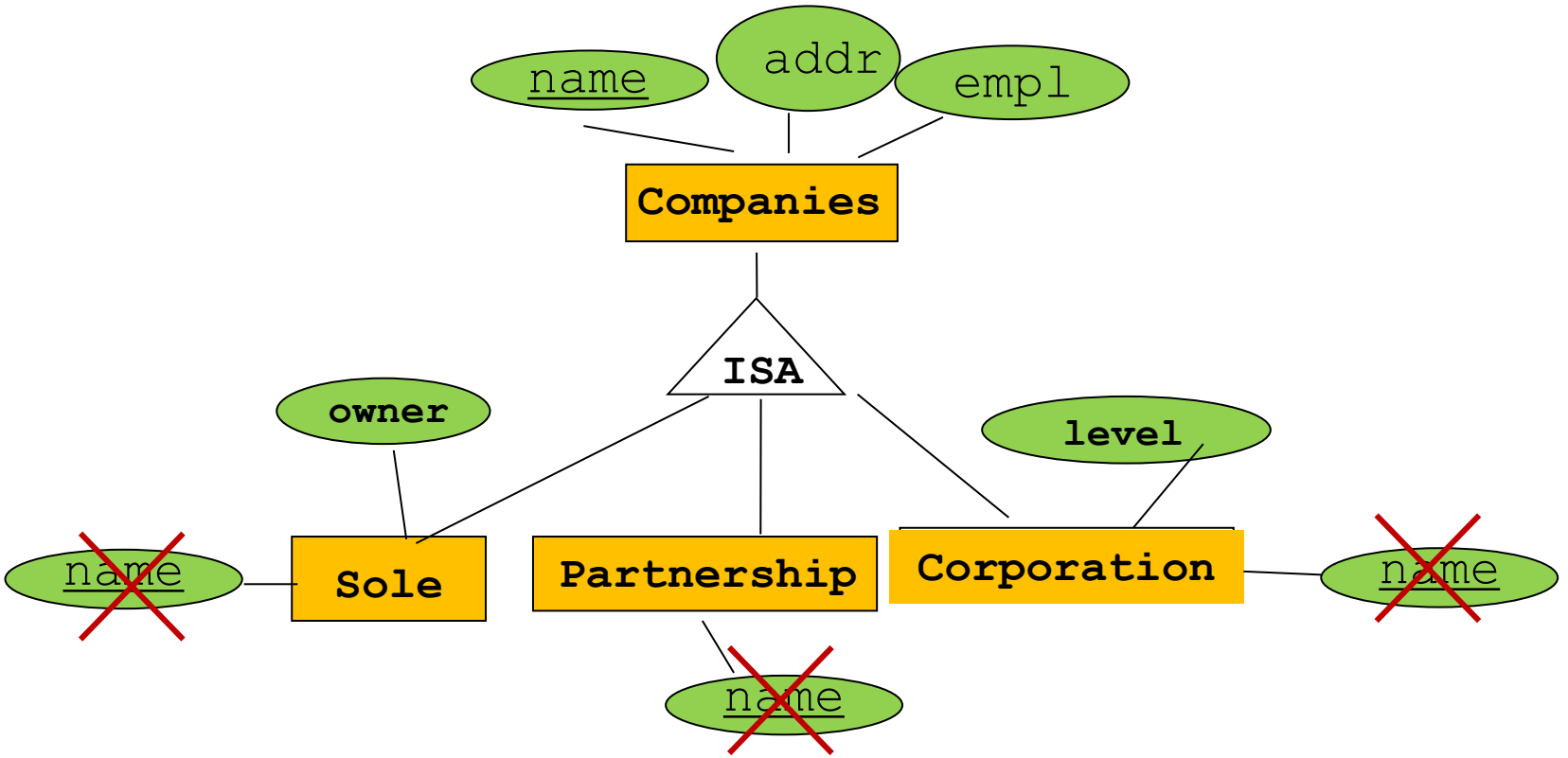


ISA (“is a”) Hierarchies: Subclasses

- Reasons for using Subclasses:
  - Additional descriptive attributes specific for a subclass
  - Identification of a subclass that participates in a relationship (will see later)
- Where do the entities reside:
  - E/R-viewpoint:: An entity has a component in each entity set to which it logically belongs. Its attributes are the union of these entity sets.
  - Contrast OO-viewpoint: An object belongs to exactly one class. The subclass inherits the attributes of its superclass.

ISA ("is a") Hierarchies: Keys

- Only the highest entity sets has the key attribute!!

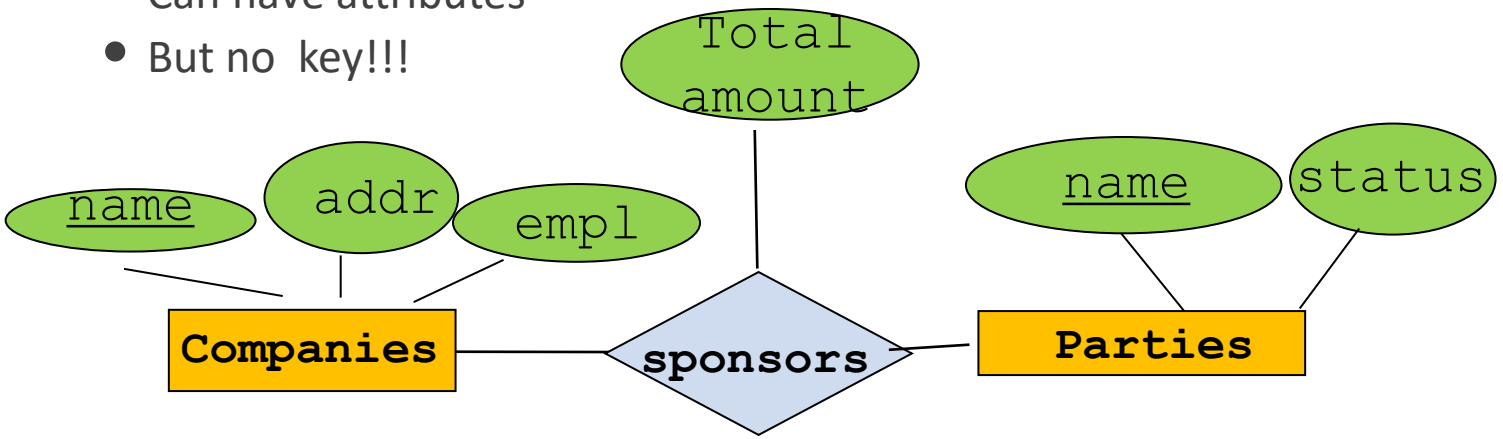


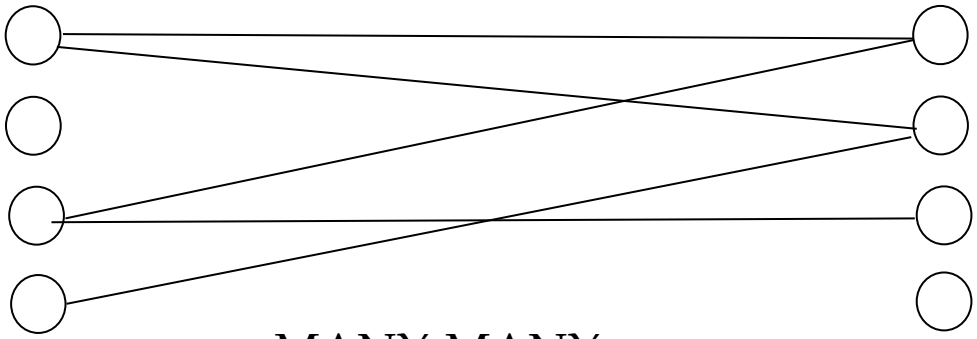
ISA Hierarchies (Contd.)

- **Overlap Constraint**: Can an entity be in more than one subclass? (allowed/disallowed) → *Default is non-overlapping*
- **Covering Constraint**: Must every entity of the superclass be in one of the subclasses? (yes/no) → *Default is non-covering*
- Developing Class Hierarchies
  - **Specialization**: Top-down approach
    - The superclass is there first
    - Then identify subsets of an entity set (the superclass) that share some distinguishing characteristic (special attributes / relationship).
  - **Generalization**: Bottom-up approach
    - Several entity sets are generalized into a common entity set
    - common characteristics of a collection of entity sets are identified, a superclass entity set is built with these characteristics as attributes.
- Theoretically multiple inheritance possible, in practice not used

Relationship

- **Relationship:** Association among two or more entities. E.g. Company X has sponsored the Liberal Party (with a total amount of ….)
- **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 ∈ E1, …. en ∈ En
    - Can have attributes
    - But no key!!!





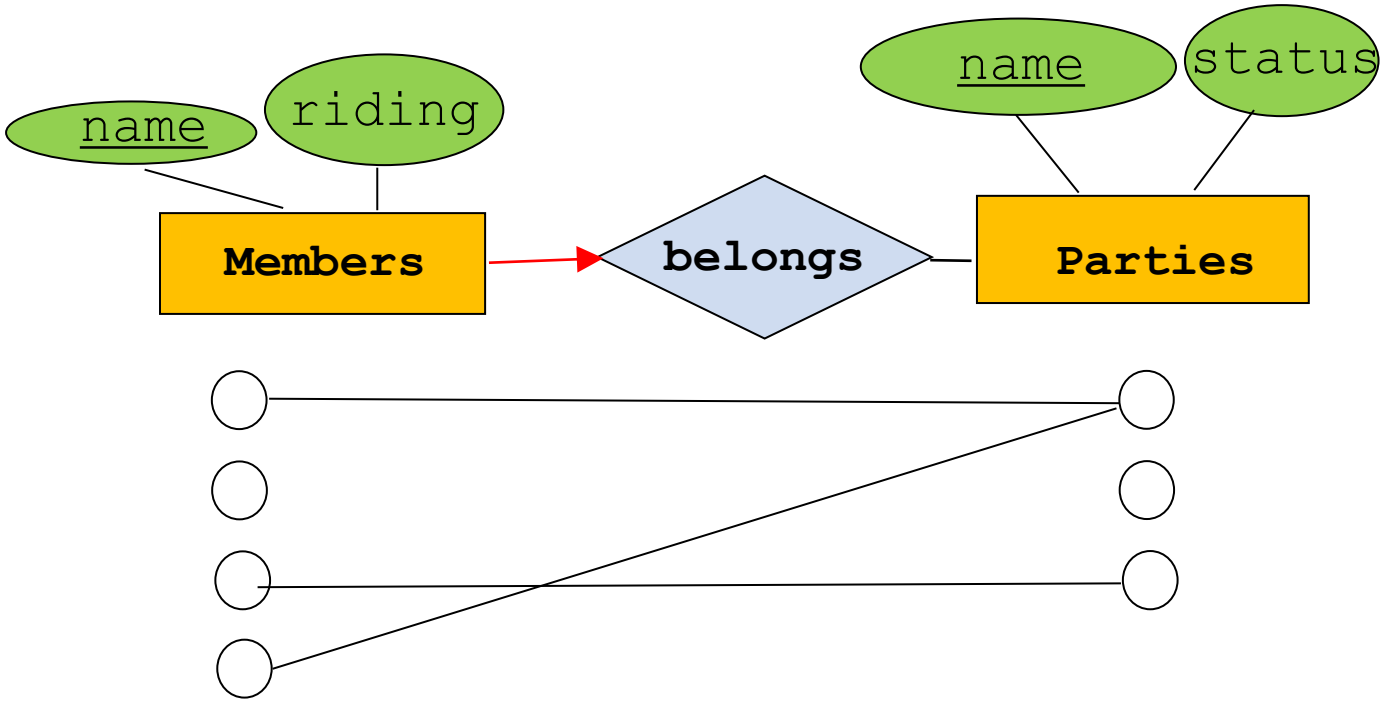
MANY-MANY

Many-many

- Each entity of `Companies` can have relationships with many entities from `Parties`
  - Company *BiggestEngCompanyEver* sponsors *Liberals* and *PQ*
- Each entity of `Parties` can have relationships with many entities from `Companies`
  - The *Liberals* are sponsored by *BiggestEngCompanyEver* and *BiggestConstCompanyEver*
- Each relationship is uniquely defined by the primary keys of participating entities
  - *BiggestEngCompanyEver* cannot sponsor twice the *Liberals*
  - If we wanted to keep track of each sponsorship transaction, we would have to model this differently….

Key-constraints: one-many, many-one

- Belongs Relationship Set:
  - Members (of the National Assembly) and (Political) Parties
  - A member of the national assembly can belong to at most one party
  - One party can have several members in the National Assembly
  - The condition "each member belongs to only one party" is called a **key constraint** on the `belong` relationship set
    - depicted with arrow from `Members` to `Parties`

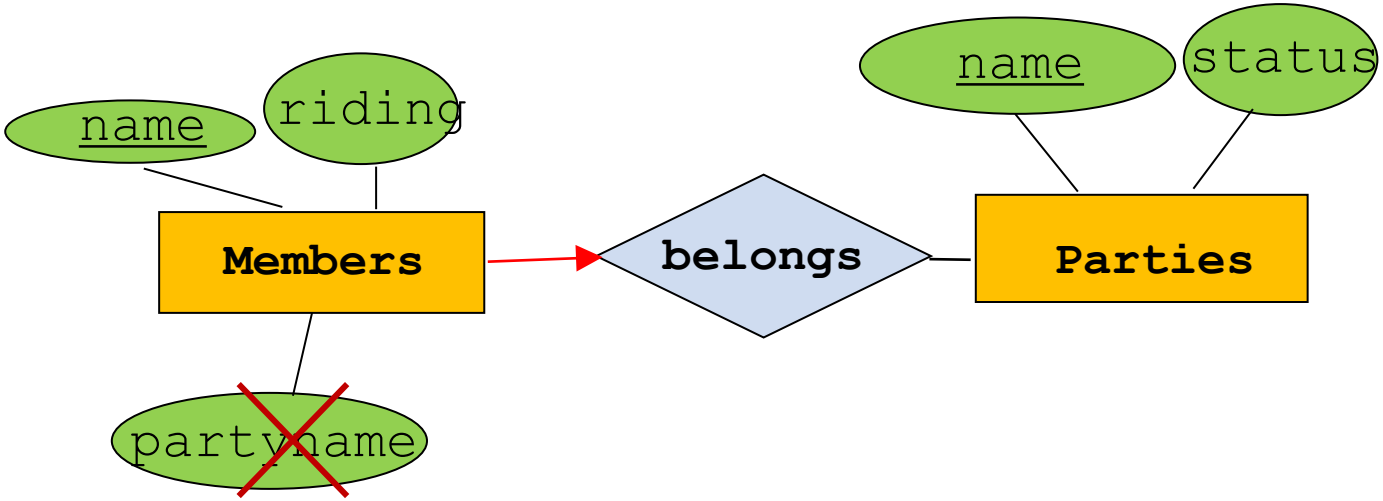


MANY-ONE

Some important notes

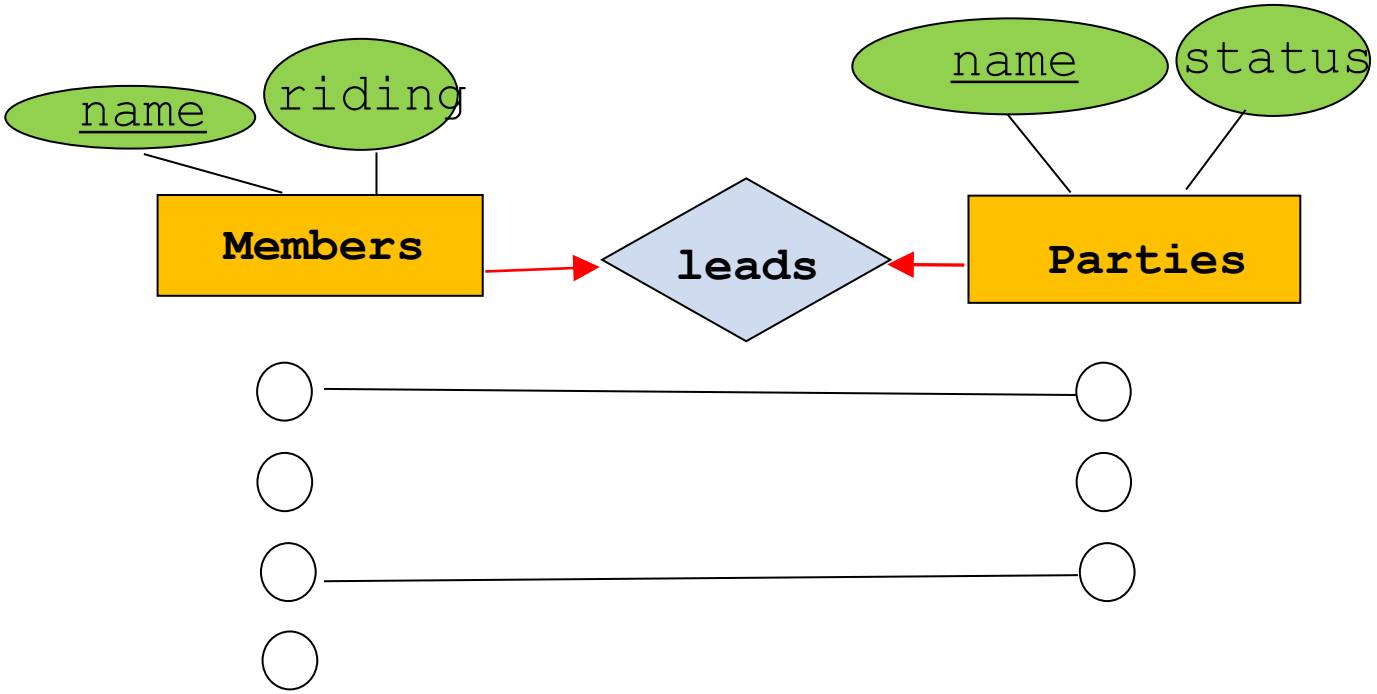
- The term *key constraint* is orthogonal to *key attribute*
  - Don't confuse the two

- Do not repeat the key of Parties as attribute in Members;
  - The connection is already given through the relationship
- We will see that in the relational model the relation Members will have partyname as attribute – but not in the ER schema!



one-one

- Lead
  - Each member of the National Assembly can be leader of at most one party
  - Each party can have at most one leader
  - One-to-one relationship set between `Members` and `Parties`
  - Key constraints in both directions



Note on Key Constraints

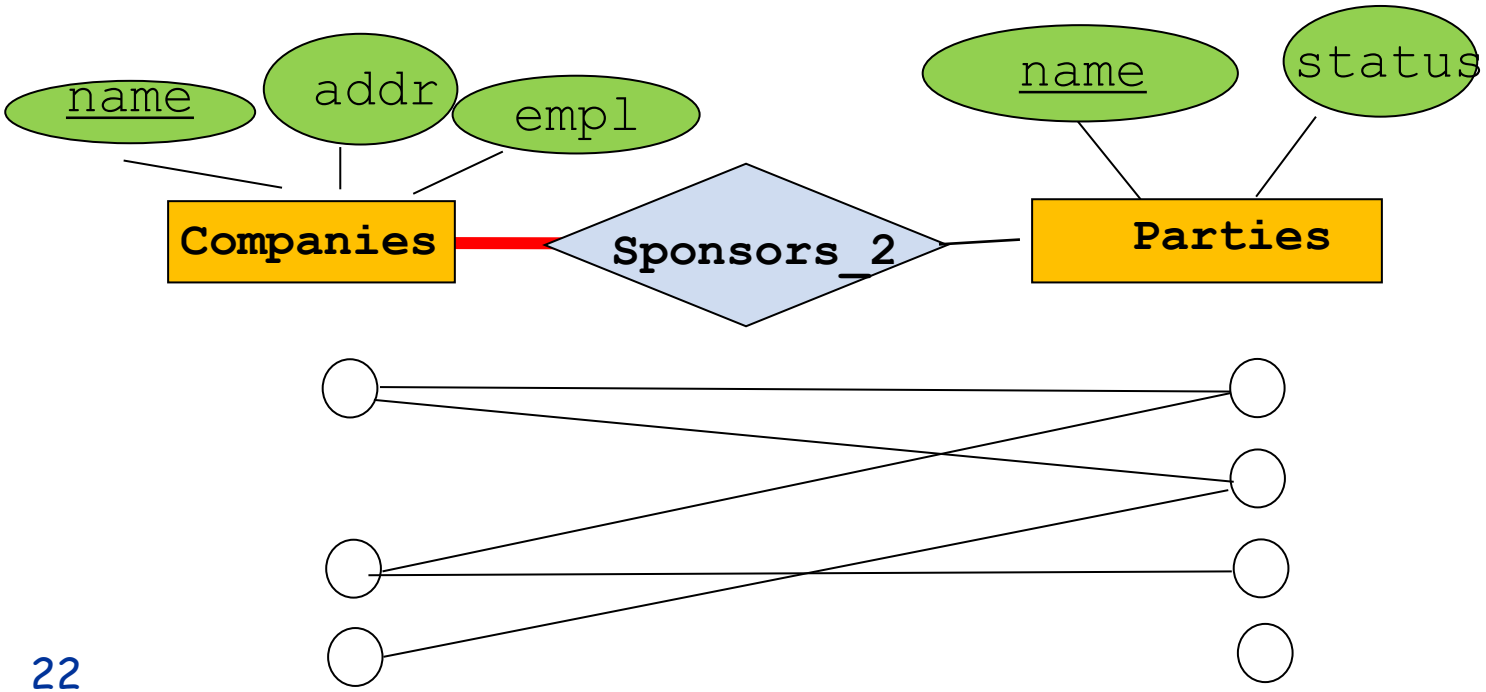
- The existence (or non-existence) of key constraints has a large influence on the final database design
  - Key constraints: less tables in the relational schema
  - No key constraints: more tables in the relational schema
- Before indicating key constraints, make sure that they really hold in all circumstances
  - Invariants of the application
  - Local example, Quebec Solidaire has two leaders.
  - in Germany, the Green party had for a long time always two leaders → key constraint of last slide does not hold
- If it later turns out that a key constraint does not hold, then your database design might have problems to capture all data

Specific cardinalities

- Some E/R languages allow you to specify a many-to-many with a specific number of relations.
- For instance:
  - A party has at most 2 leaders or
  - A party has exactly 2 leaders
- In our language
  - Option to have 2 relationship sets with key constraints
    - Leads1 and leads2 from Party to Member

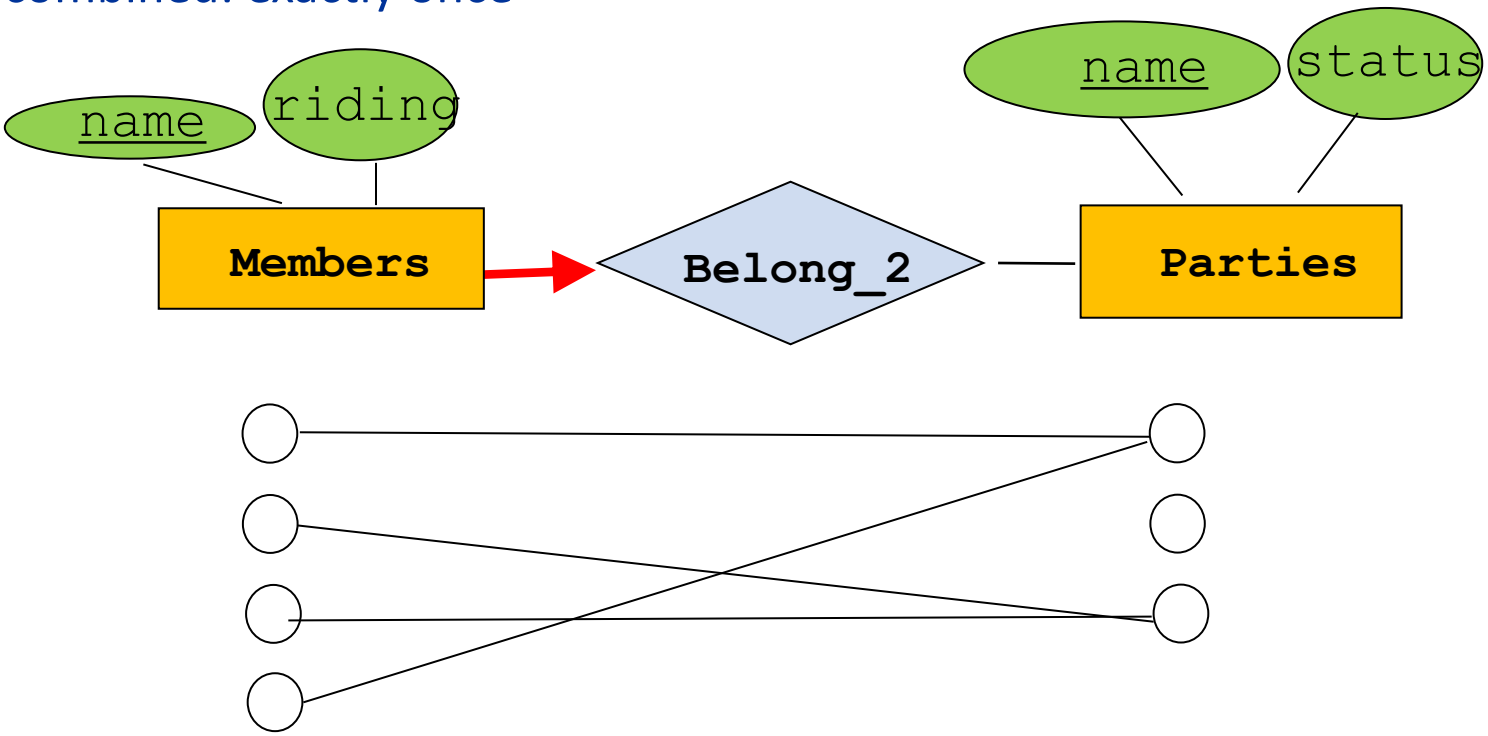
Participation Constraints

- Sponsor_2 participation:
  - Each company must sponsor at least one party
    - (we don't keep track of companies that don't provide sponsorship….)
  - the **participation constraint** requires that the participation of `Company` in `Sponsor` is *total* (vs. *partial*) (depicted with a **thick** line)



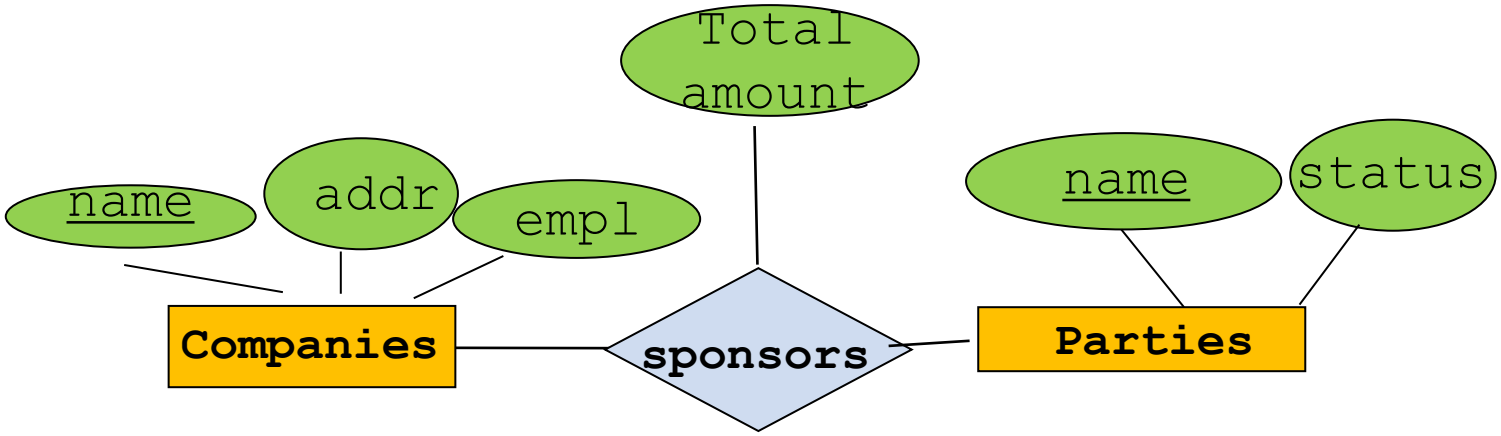
Participation Constraints

- belongs_2
  - Each member must belong to exactly one party
  - key constraint: at most once (at most one party)
  - participation constraint: at least one
  - combined: exactly once



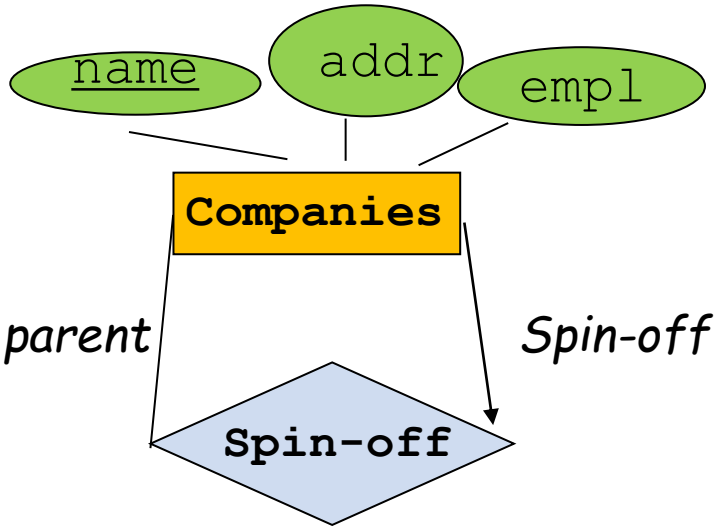
Model Task

- A company can make several sponsorship payments to a party. Each payment has a date and an amount.
- Redesign below to allow for above situation



Relationships contd.

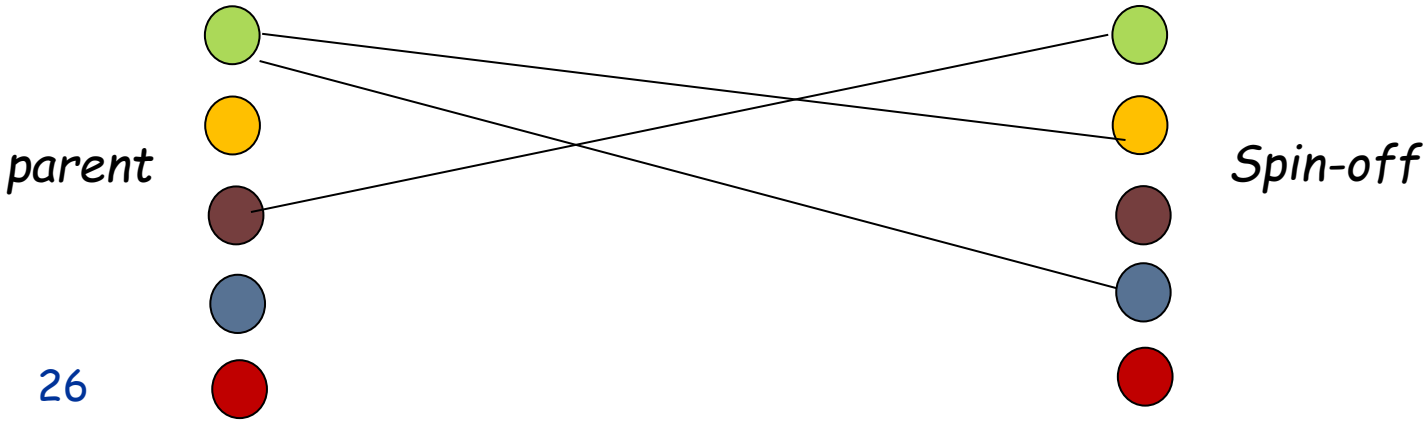
Relationship between companies



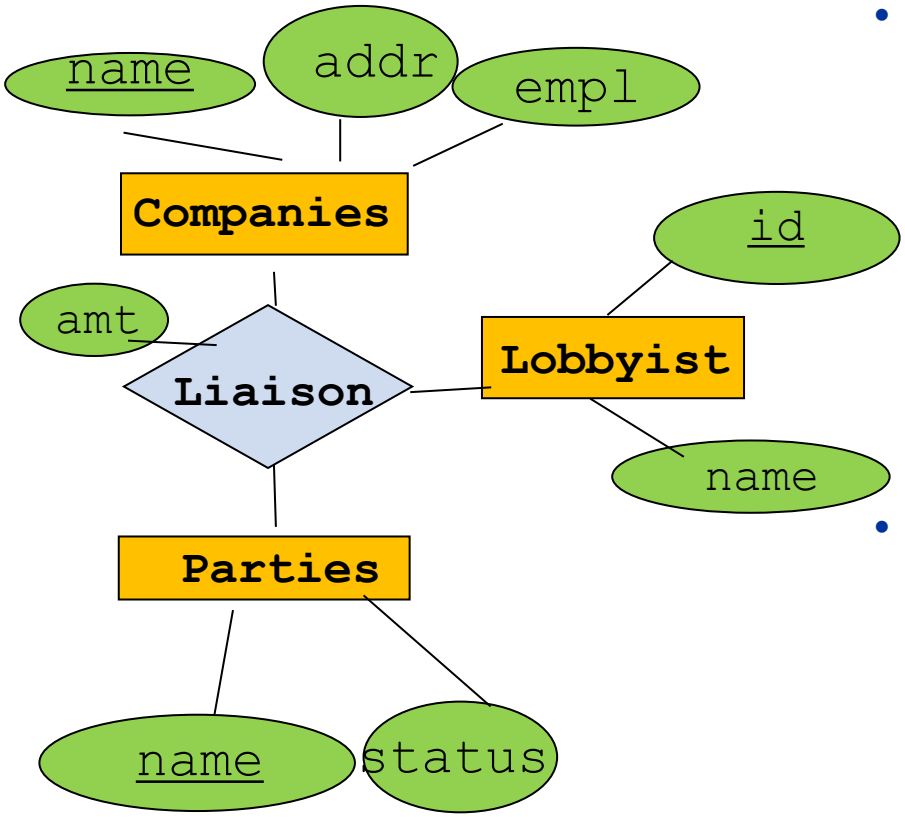
Ex: Spin-off

Alphabet ====== Google

Role indicator:
Parent company / spin-off



Ternary Relationship

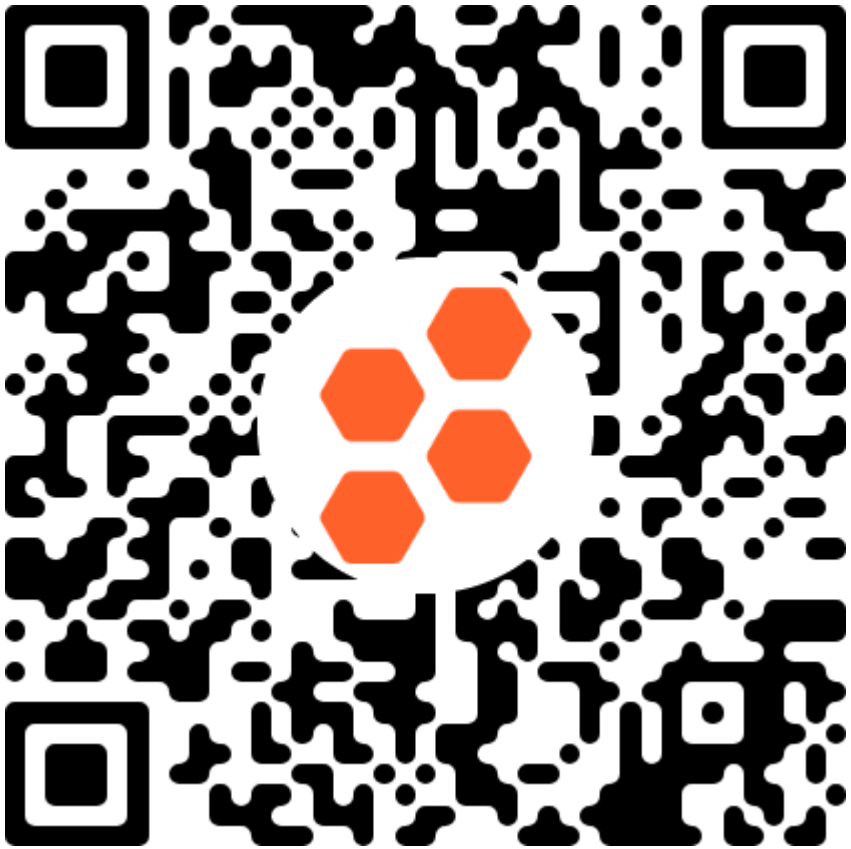


Examples

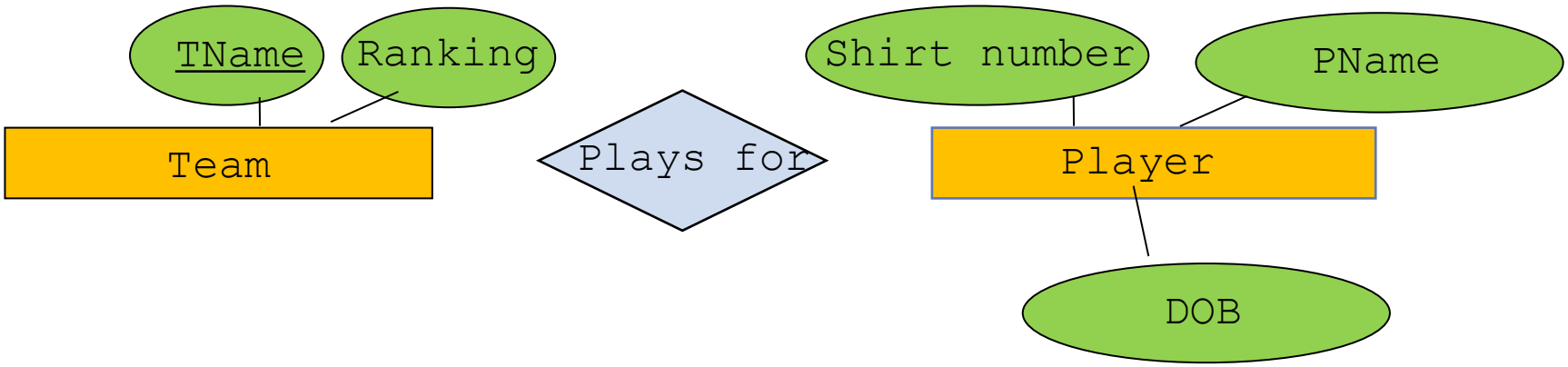
- Lobbyist Miller with id 007 builds the liaison for
  - BiggestEngFirmEver and the Liberal Party;
  - BiggestConsFirmEver and the Liberals
  - BiggestConsFirmEver and the CAQ
- Lobbyist Max with id 008 builds the liaison for
  - BiggestEngFirmEver and the Liberal Party;

- Note that given relationship set is many-many-many

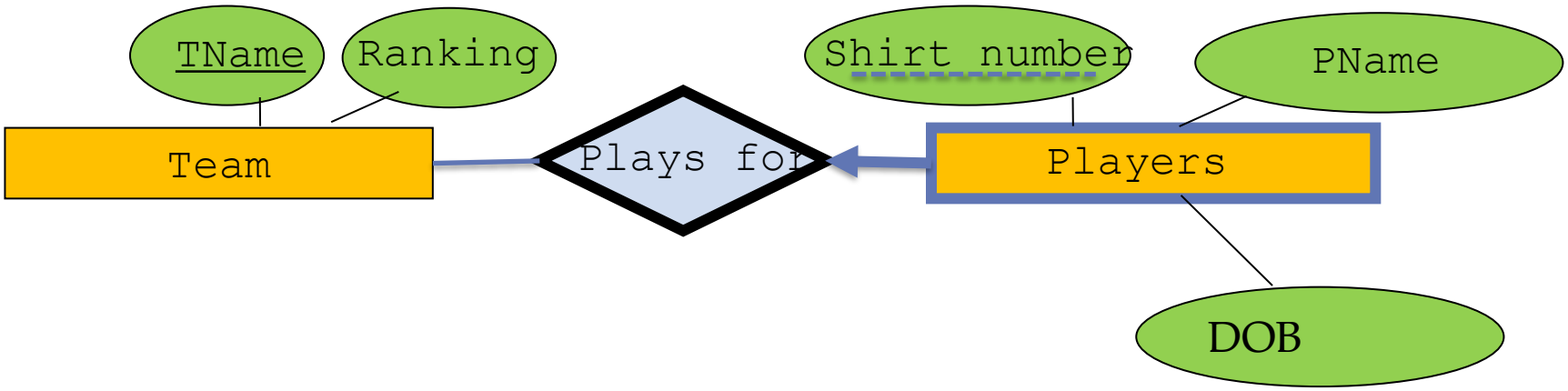
Requirements Quiz



Weak Entities



Weak Entities

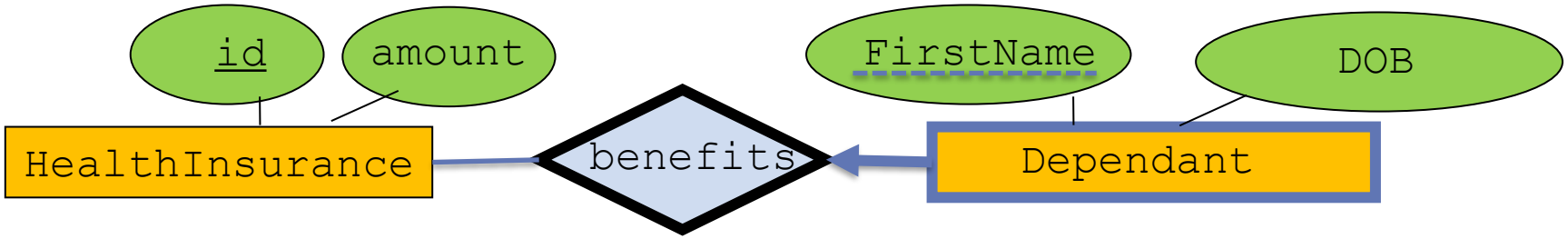


Weak Entities

- A **weak** entity "needs help" from another entity to be uniquely identifiable
- That is, a weak entity can be identified uniquely only by considering the primary key of another (owner) entity.
  - The key in weak entity set is the union of the key of the owner entity set and a set of its own attributes (underlined with dashes)
  - Owner entity set and weak entity set must participate in a *supporting one-to-many* relationship set (one owner, many weak entities).
  - Weak entity set must have total participation in this *identifying* set.

Example: Dependants covered

- First names are not unique
- But the first names among siblings are unique
  - Combination of life insurance id and first name are unique
- E/R:
  - Weak entity set in **bold**
  - Relationship set to supporting entity set with key and participating constraint (**bold and arrowed**)
  - Relationship set in **bold**
  - Partial key in weak entity set with **dashed line**



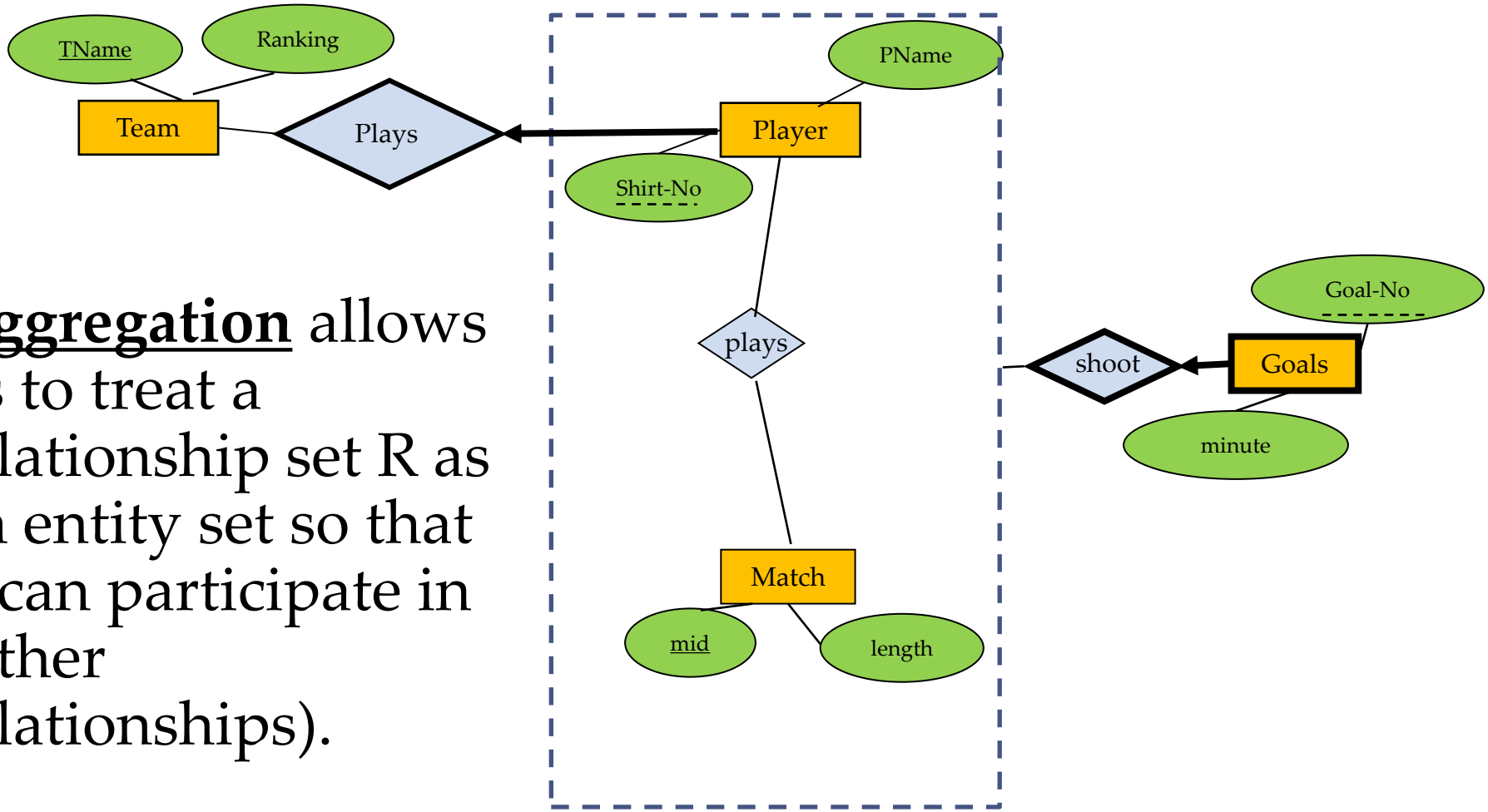
Don't overdo Weak Entities

- Many things appear to depend on entities of other entity sets and need their keys to identify.
- In reality:
  - Examples
    - Car and Owners
      - Make License plate key
    - Online users and their purchases
      - Generate unique purchase order id

When to use weak entities

- If the partial key within the weak entity set has some “meaning”
  - A name, an order (1st, 2nd, …)
- If there is no global authority to create global identifiers

Aggregation

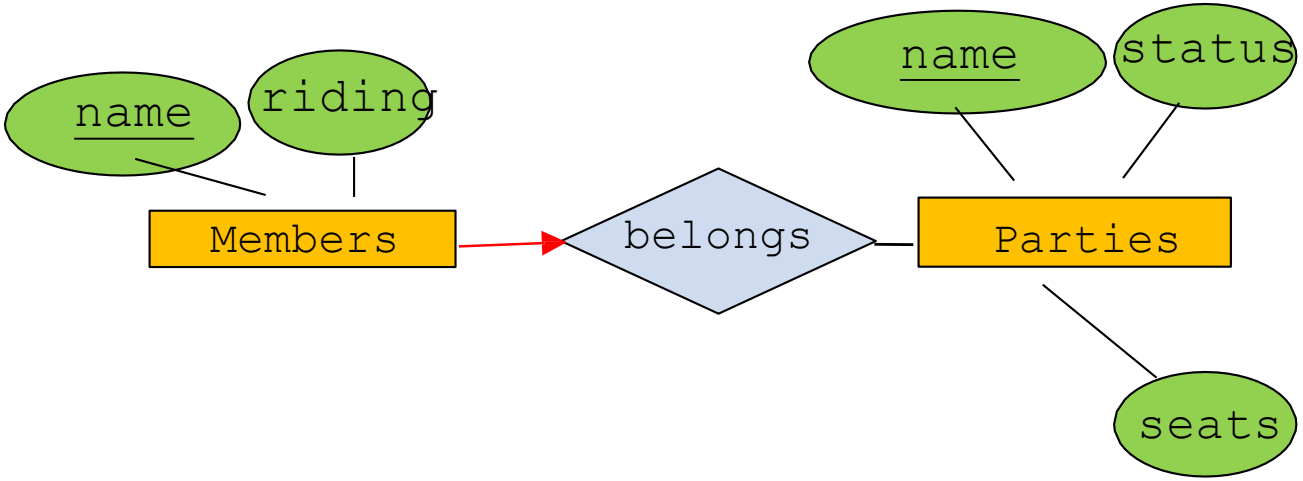


Aggregation allows us to treat a relationship set R as an entity set so that R can participate in (other relationships).

Conceptual Design with ER

- Design Principles:
  - Keep it simple
    - Don’t use an entity set when an attribute is sufficient
  - Avoid redundancies (or not)
  - Capture as many constraints as possible
- Design Choices
  - Entity or attribute?
  - Attributes and relationships
  - Identifying relationships: Binary or ternary?

Redundant Attributes



- Attributes `seats` carries redundant information
- Why?
  - The information about many seats a party has can be queried by counting the number of relationships the Party has in the relationship set "belongs"
- Keep it nevertheless
  - If it is something that will be queried often; it might be useful
  - One has to be aware that it is redundant and ensure consistency

Entity Set vs. Attribute

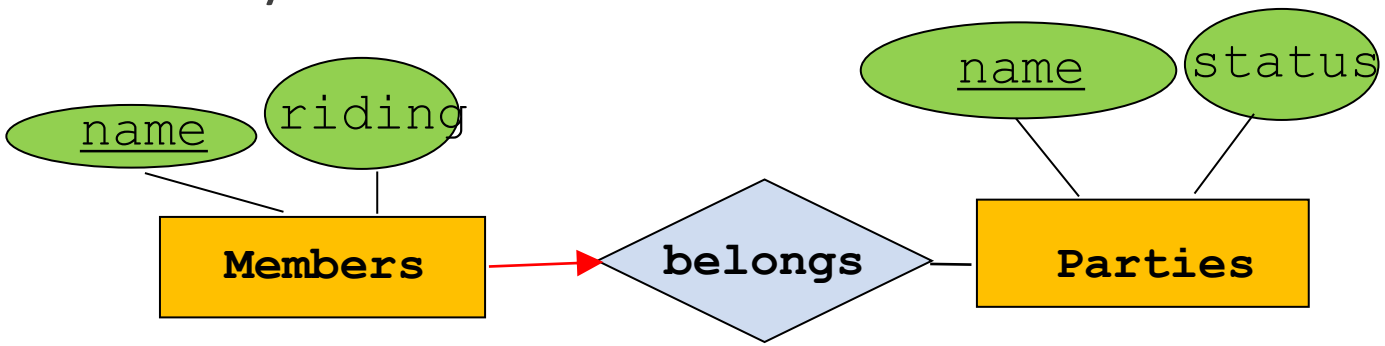
- A “thing” should satisfy at least one of the following to be represented as an entity set
  1. It is more than the name of something; at least one non-key attribute
  2. It is the many in a many-many or many-one relationship set
     - Example: A product can belong to many categories

Entity Set vs. Attribute

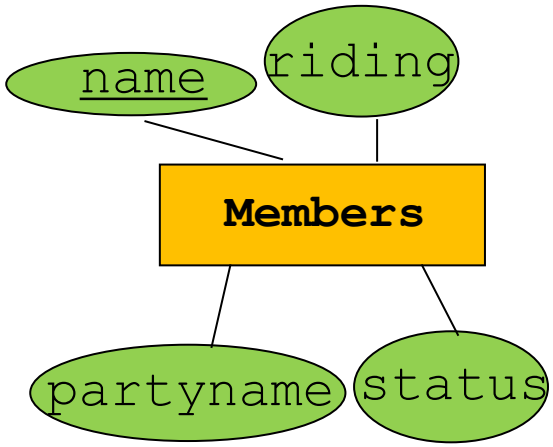
– It is more than the name of something;

– at least one non-key attribute → Entity Set

- Party has additional attribute status



– Bad Design

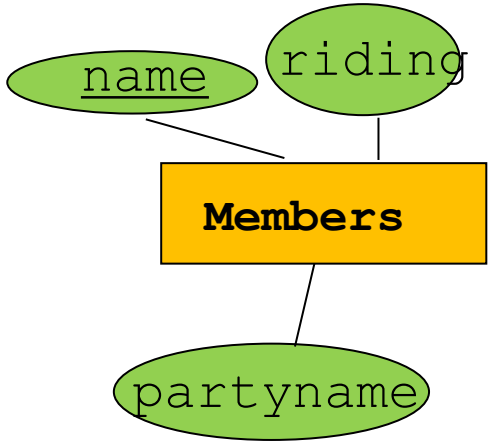


Redundancy: This design indicates the status of a party for each member of the party

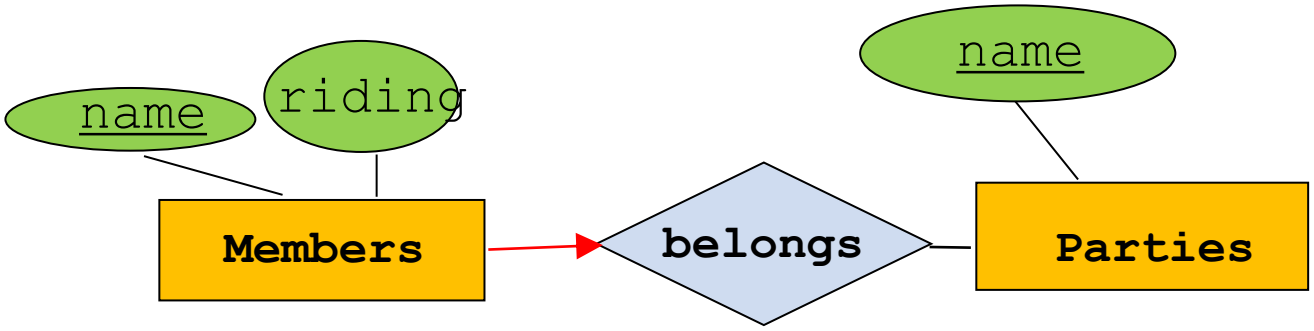
Information loss: It loses the information for a party that temporarily does not have members.

Entity Set vs. Attribute

– Attribute: No extra information about the party except the name (and a member belongs to at most one party)

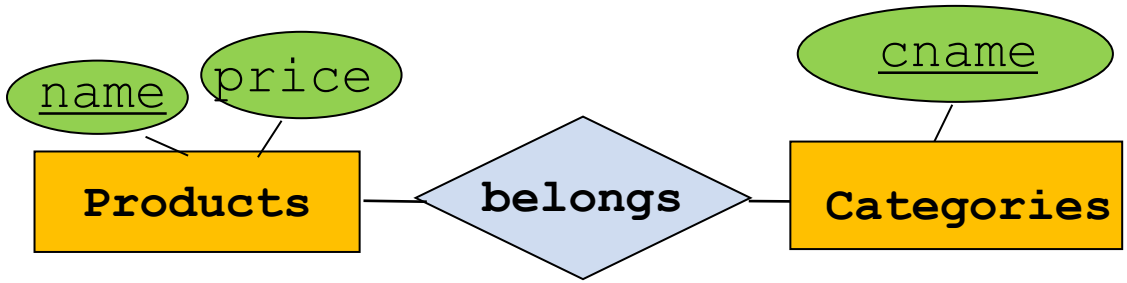


– Not so great design (still correct…)

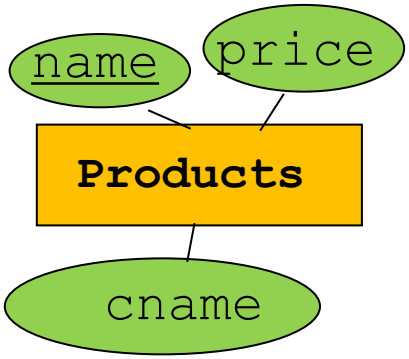


Entity Set vs. Attribute

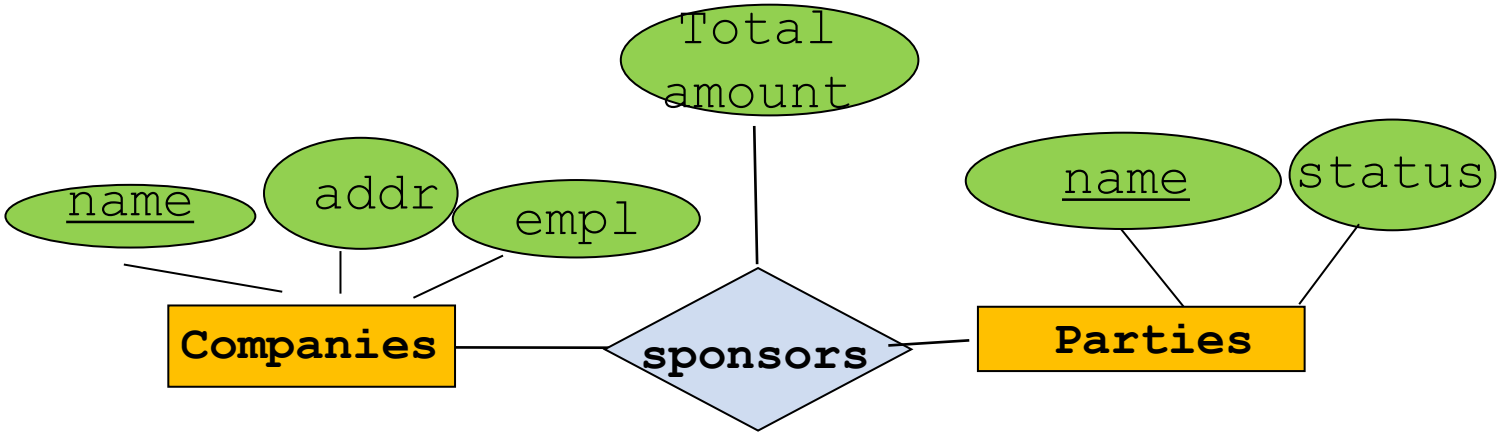
- It is the many in a many-many or many-one relationship set → Entity Set
  - Example: *Category* and a product can belong to many categories

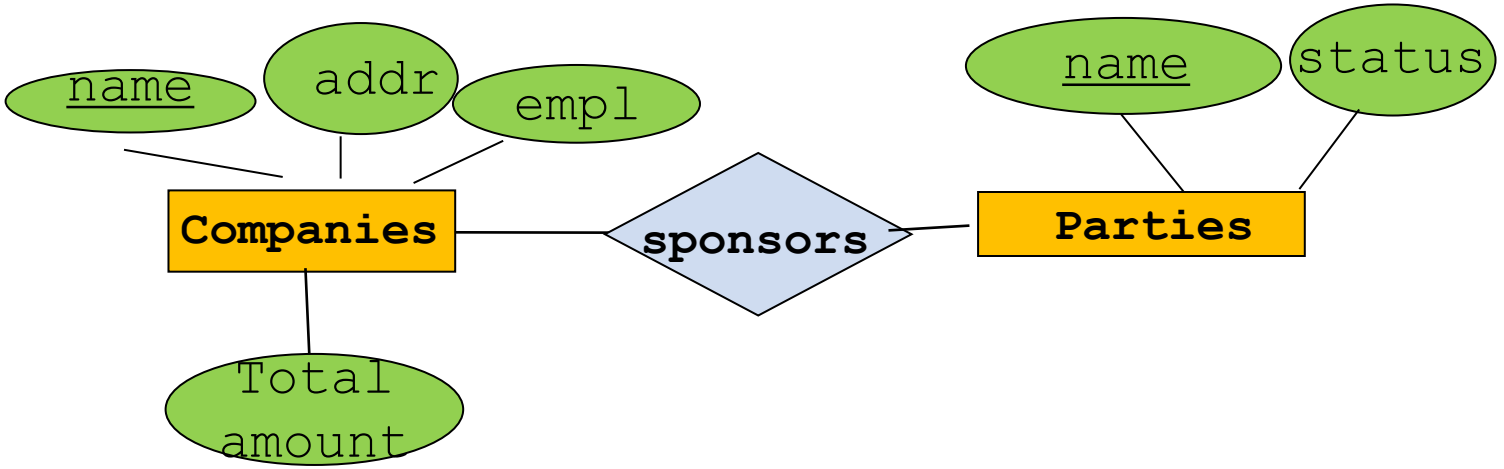


- It is in one in a many-one or one-one relationship set and has no special attributes → Attribute
  - Example*: Category* and a product belongs to only one category (and category is simply a name)
  - Only possible because ONE category value (attributes cannot have multiple values)

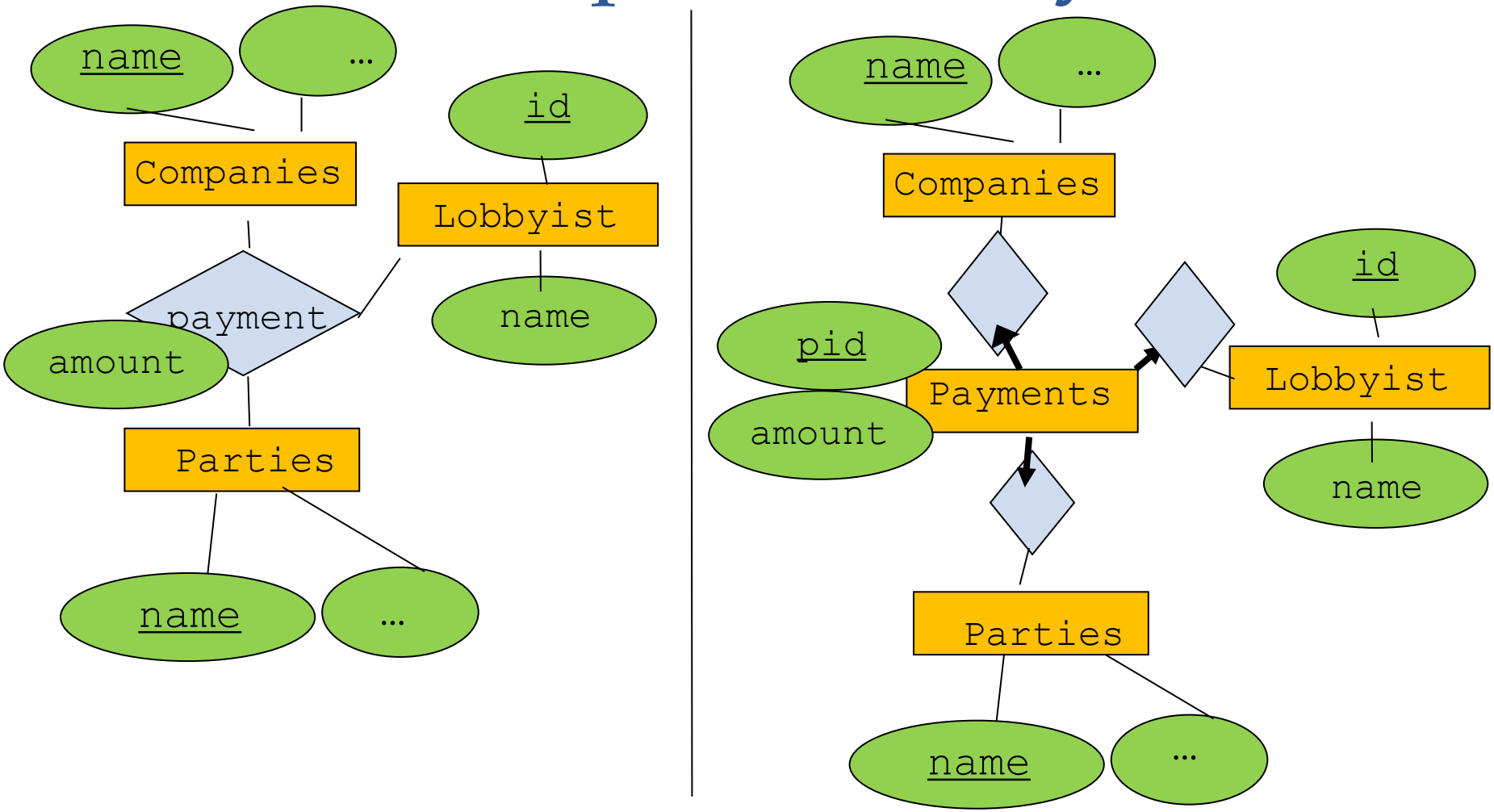


Relationships and Attributes



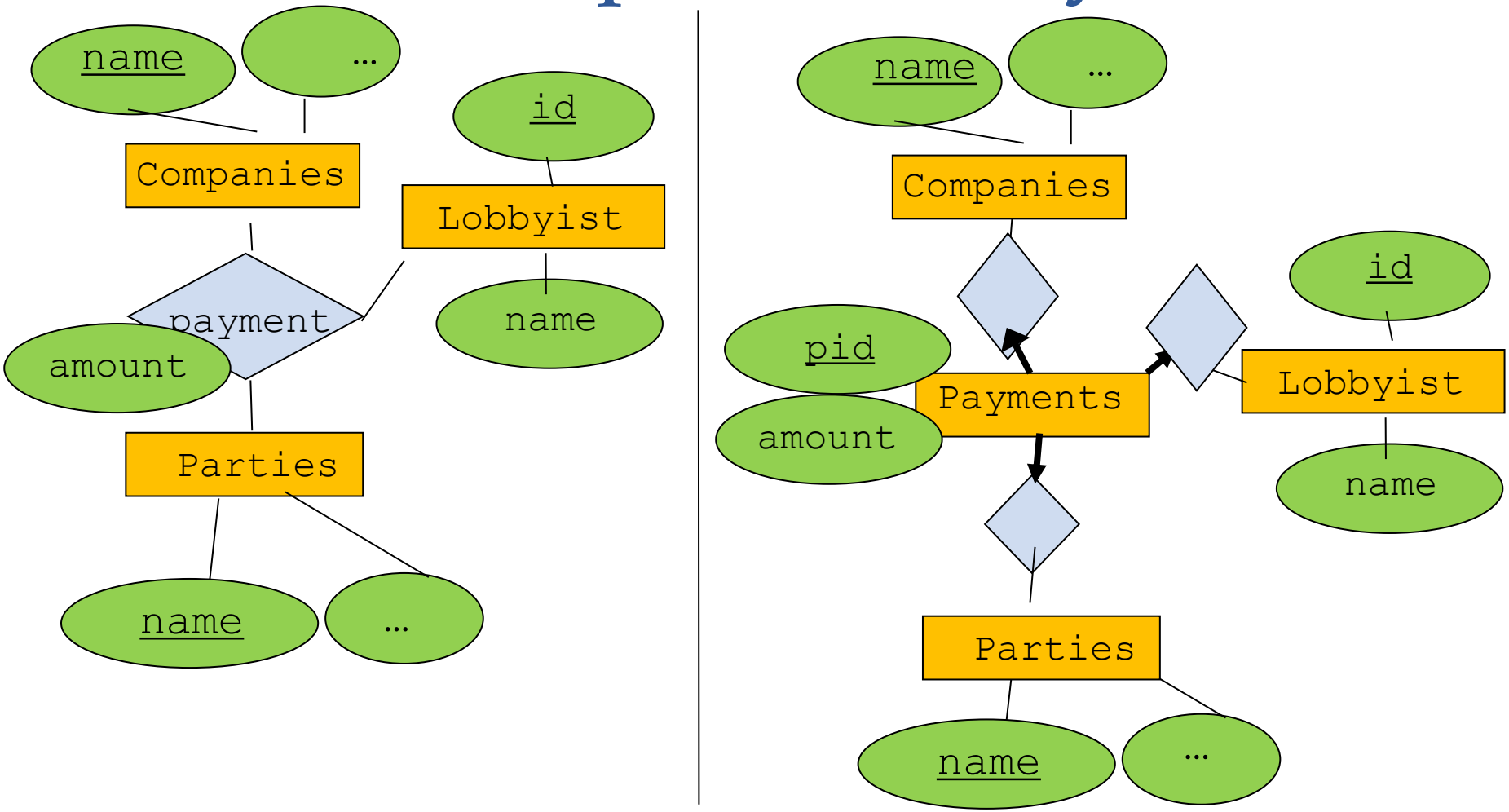


Relationships vs. Entity Sets



Ternary (payment as relationship set) vs. 3 binary relationship sets (payment as entity set)

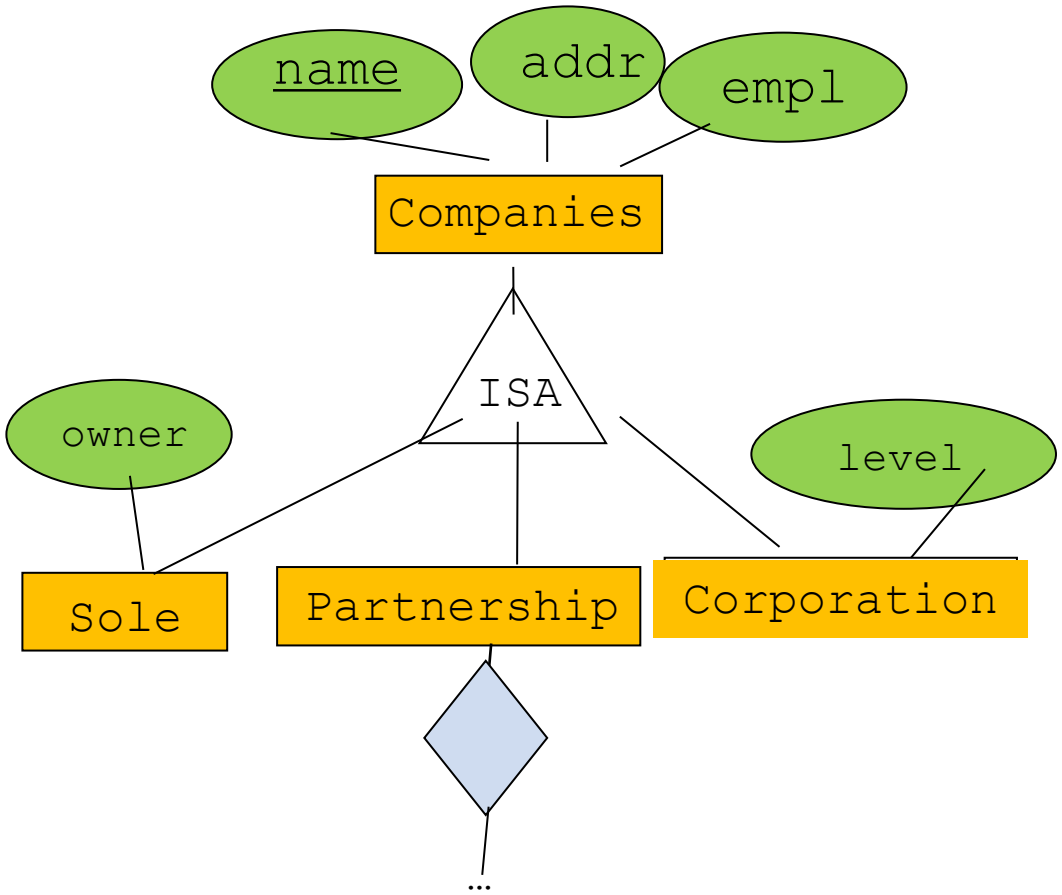
Relationships vs. Entity Sets



payment as entity set allows several payments of the same company to the same party by the same connection

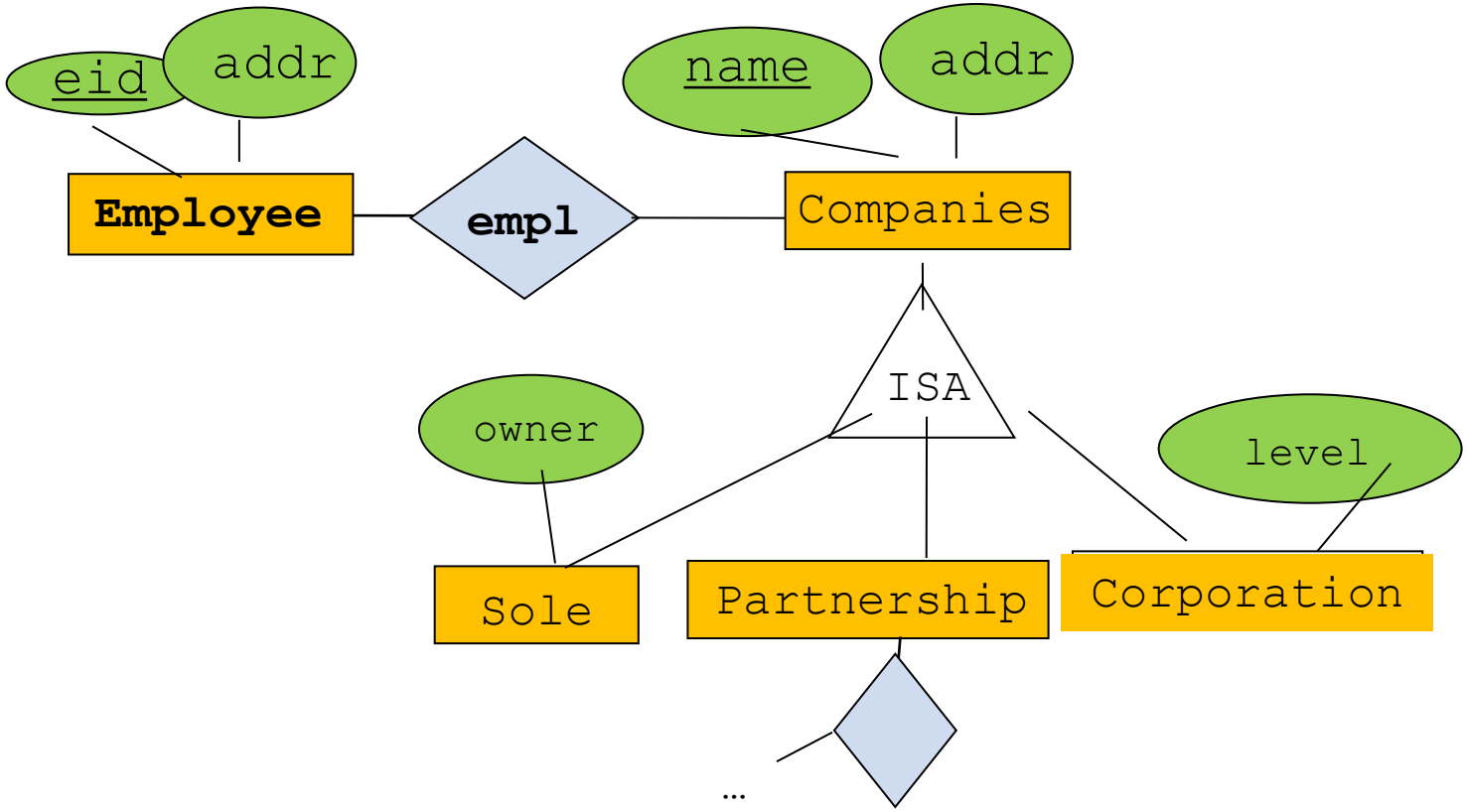
ISA ("is a") Hierarchies

- Subclasses
  - Have extra attributes that only belong to the subclass OR
  - Have a relationship set that only holds for the subclass



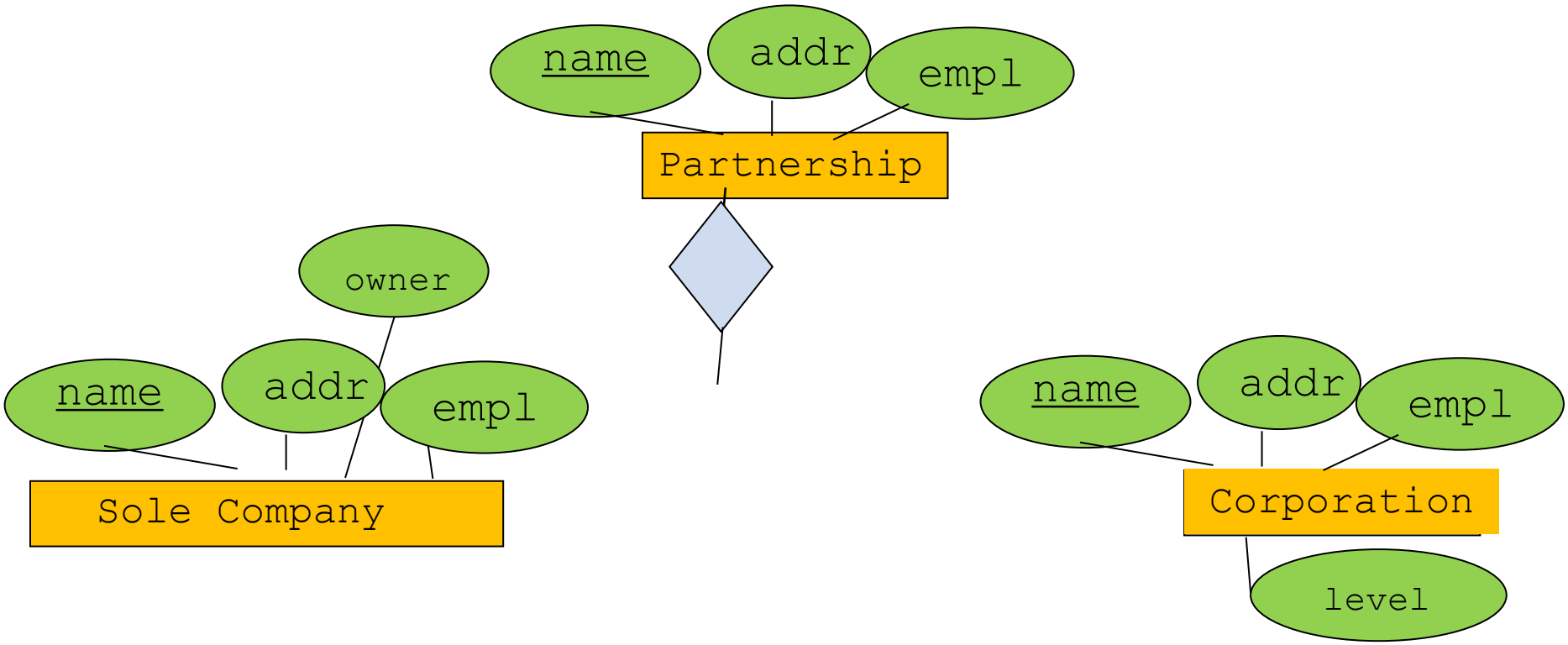
ISA (“is a”) Hierarchies

- Usefulness of a Super Classes
  - Superclass has relationship sets that hold for all entities
  - Importance of unified key attribute across all subclasses
  - There are many queries that are interested only in the attributes of the superclass

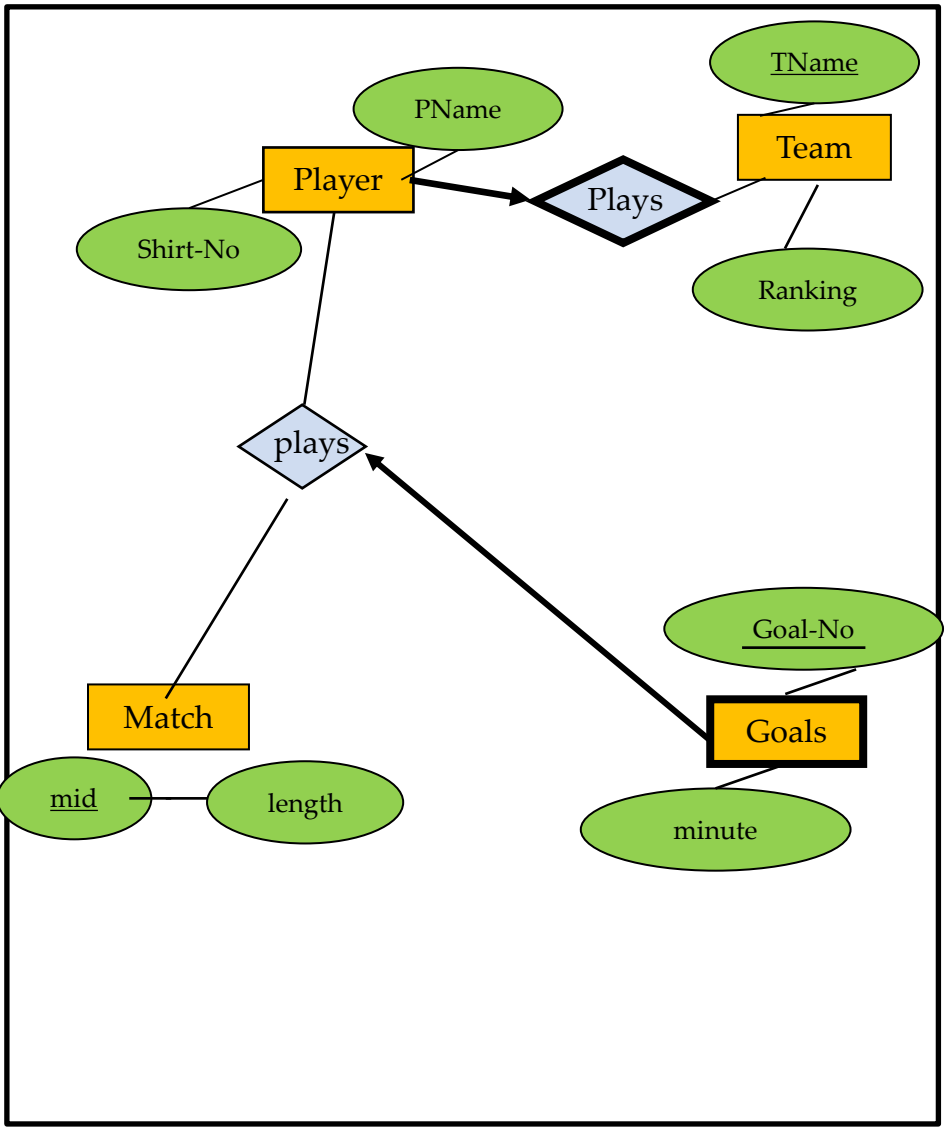


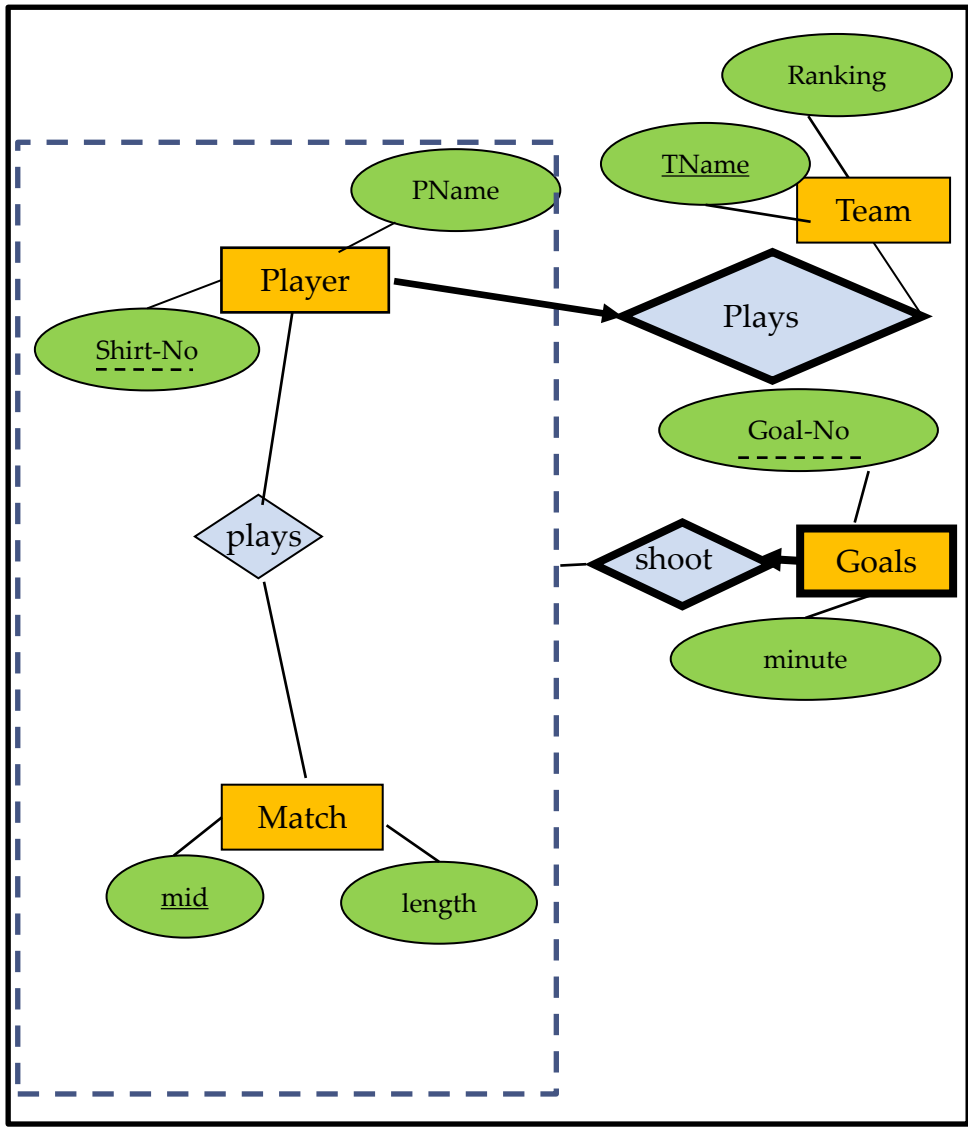
ISA (“is a”) Hierarchies

- Usefulness of a Super Class
  - If superclass not useful



Tenary Vs Aggregation





Binary Vs Aggregation