University of Glasgow

Data Storage and Retrieval
Lecture 3
Data Modelling: ER Diagrams

Dr. Graham McDonald

Graham.McDonald@glasgow.ac.uk

University of Glasgow

What is an ER Model?

- Entity Relationship Model: A **conceptual data model**
  - later mapped to a logical data model or **schema** (i.e. definitions of TABLES)
  - this in turn is mapped to a physical model by the DBMS
- Usually described using ***Entity-Relationship Diagrams***
  - Describes type of information to be stored in a database
  - Provides a pictorial overview and classifications of used terms and their relationships
- The most common method for modelling of a DB

University of Glasgow

The Entity-Relationship Model

- Data in an ER Model is described in terms of three key concepts:
  - Entities
  - Attributes
  - Relationships

Entities

- An **entity** is a uniquely identifiable object in the real world about which we wish to store data
    - For example: The Bank of Scotland, The University of Aberdeen, Tony Blair, Celtic Football Club, BBC, my car…….
- A thing which is recognised as being capable of an *independent existence* and which can be ***uniquely identified***

University of Glasgow

Entities

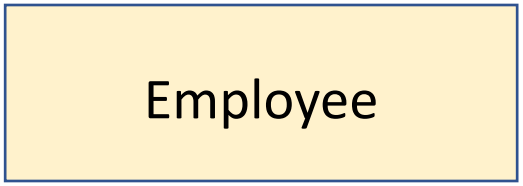
- Entities are grouped together into ‘categories’ called **entity types** or **entity sets**
  - *Employee*, *Department*, *Project*

- An **entity** is an instance of a given entity-type
- There are usually *many* instances of an entity-type

Entity Types

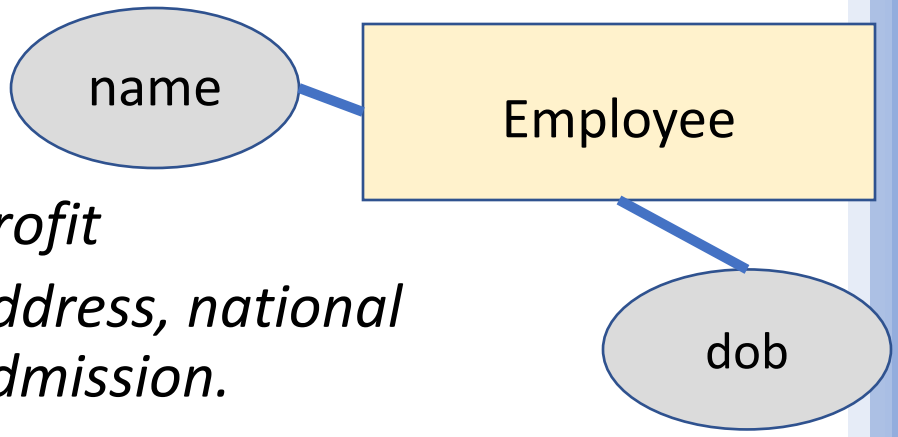
Entity types can be thought of as (common) nouns

- Can be a **physical object** such as a house or a car
- Can be an **event** such as a house sale or a car service
- Can be a **concept** such as a customer transaction



- Proper nouns indicate instances of entities
  - Joe Bloggs is a Customer

Attributes



Attributes are properties

that describe an entity (type)

- BT: *name, address, annual profit*
- Patient John Smith: *name, address, national insurance number, date of admission.*

It is expected that all instances of a given entity type will have the same attributes

- an **entity type** defines a set of entities that have the same attributes
- i.e. We record same details for employees Jane Black and Gregory White

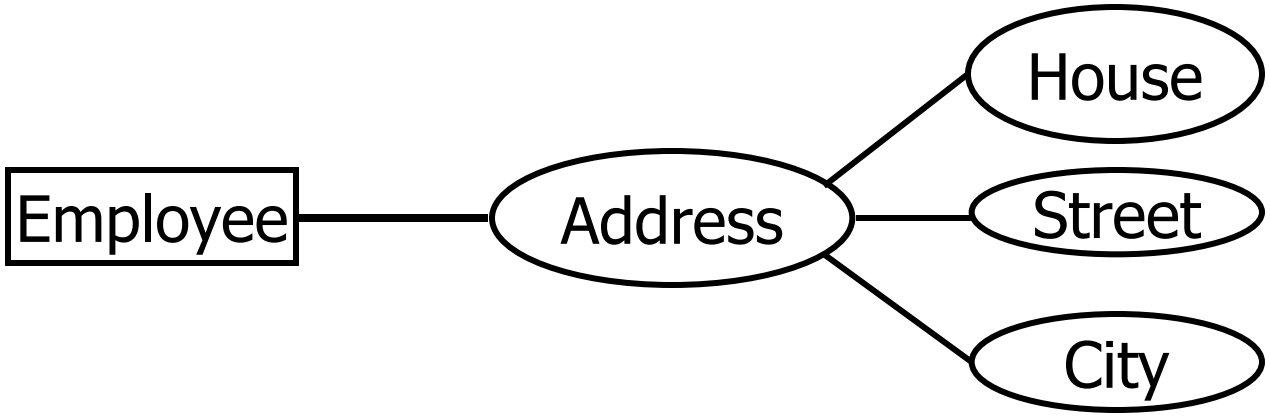
Attributes are drawn as ovals, and attached to the boxes representing entity types with lines

Attributes: **Simple** (atomic) vs composite

- Simple
  - indivisible value
  - age, gender

Attributes: Simple (atomic) vs **composite**

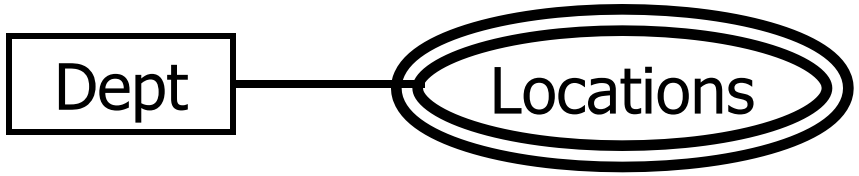
- Composite
  - composed of a set of component values
  - address, date of birth



Other Kinds of Attribute

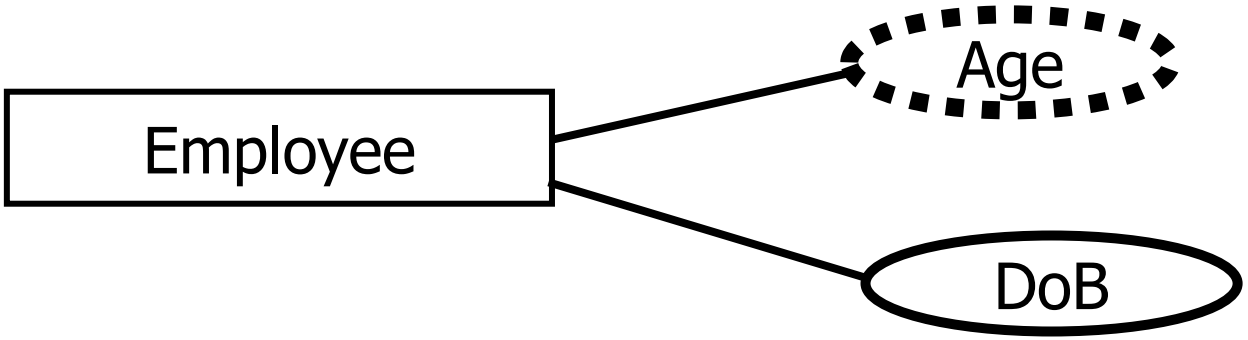
- Single-valued vs multi-valued
  - multi-valued stores a set of values
  - Indicated by double-lined attribute oval
- Examples:

  locations for a department; hobbies for a person



Other Kinds of Attribute

- Derived vs base
  - base are explicitly stored
  - derived can be calculated by a procedure
    - (e.g., age from birthday)



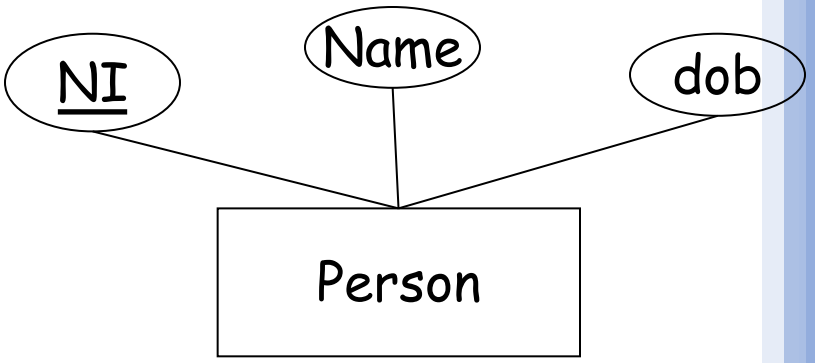
University of Glasgow

Primary Key

- An Entity type will usually have **key attribute(s)**:
  - one (**or possibly more**) of the attributes which are unique for all entity instances
  - for example
    - A book's ISBN
    - A date (composite attribute M/D/Y)

Key Attributes

- The **primary key** attributes of an entity type is an attribute whose values are distinct for each entity
- We underline key attributes



- Sometimes several attributes (a composite attribute) together form a key
  - NB: Such a composite should be **minimal**
  - E.g. The combination of account number AND sort code are unique in UK banking

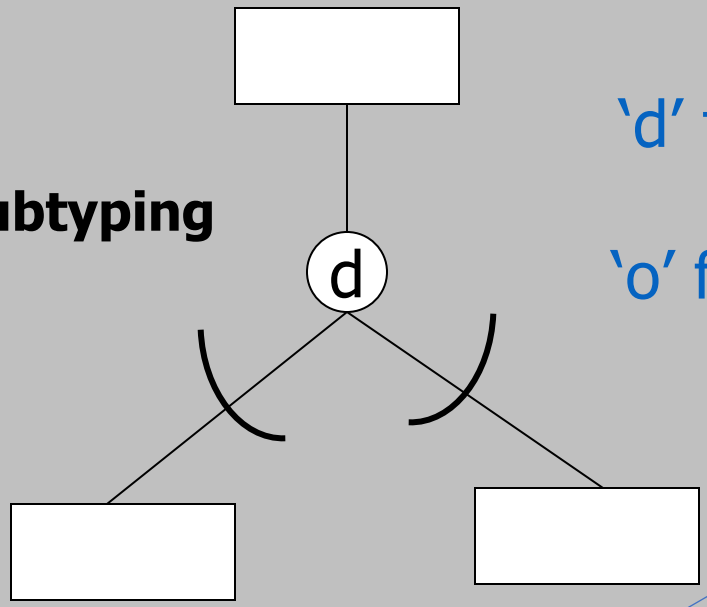
University of Glasgow

Subtyping

- A subtype is an entity type that *inherits* the properties of its parent type
  - e.g. programmer & manager can be represented as subtypes of employee
- Employee attributes (name, NIN, etc) belong to programmer and manager by virtue of being subtypes of employee
- Subtypes may be
  - disjoint - must belong to exactly one subtype
  - inclusive - may belong to either or both

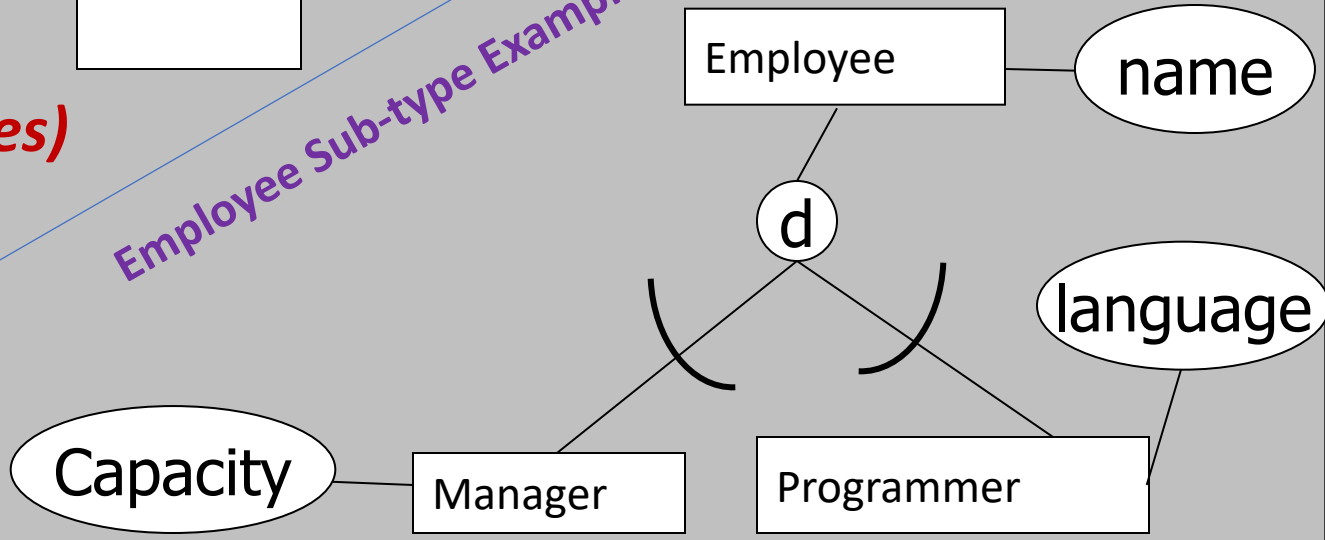
Subtyping Notation

(Supertype)



'd' for disjoint
or
'o' for inclusive

Employee Sub-type Example



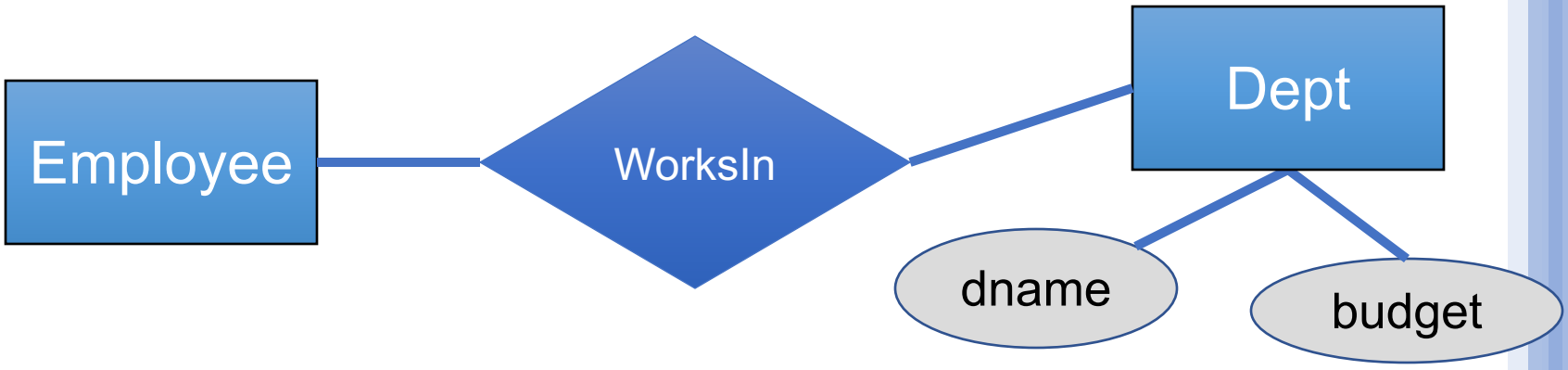
University of Glasgow

Relationships

- Captures how two or more entity types are related
- Can be thought of as verbs, linking two or more nouns
- Examples:
  - an *owns* relationship between a company and a computer
  - a *supervises* relationship between an employee and a department
  - a *performs* relationship between an artist and a song
  - a *proved* relationship between a mathematician and a theorem

Relationships

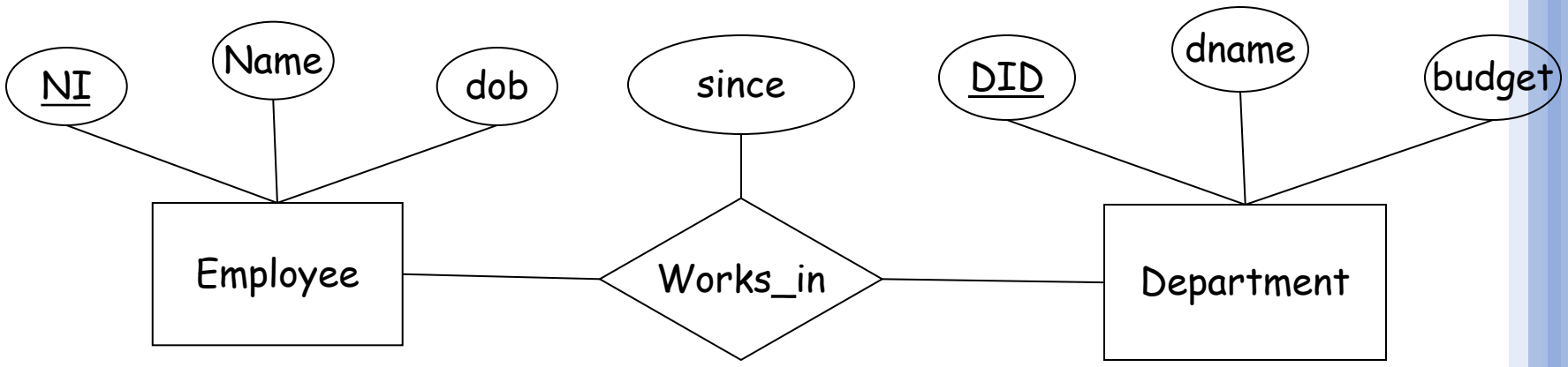
- **Relationships types** represent the *interaction* between entity types
  - For example the entities in types “employee” and “dept” can interact through the relationship “worksIn”



- Relationship types are represented by diamonds
- They connect the participating entity types with straight lines

Relationship attributes

- Relationships can also have **attributes**
  - NB: A relationship must be uniquely determined by the participating entities, without reference to the relationship attributes



  - E.g. Mark works_In Computing Science, since 2009
  - E.g. John works_In Computing Science, since 2009

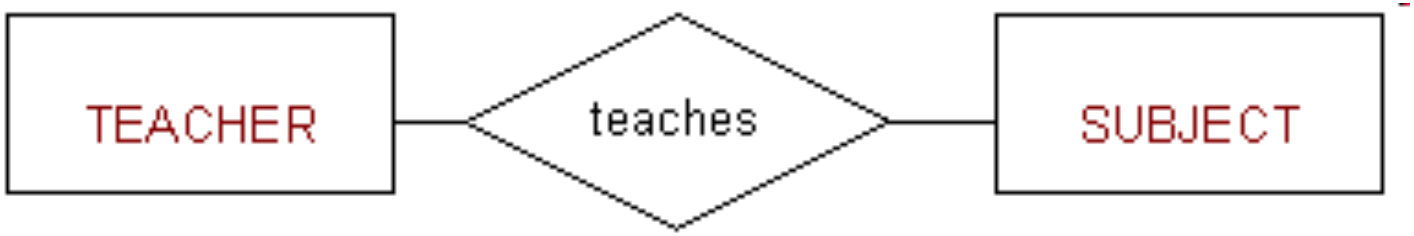
University of Glasgow

Relationship Degrees

- The ***Degree*** of a relationship is the number of entity types participating
  - Binary relationships
    - 2 participating entity types
    - Employee works for Department
  - N-ary (e.g. Ternary) relationships
    - >=3 participating entity types
    - a Manager manages a Project in a Department

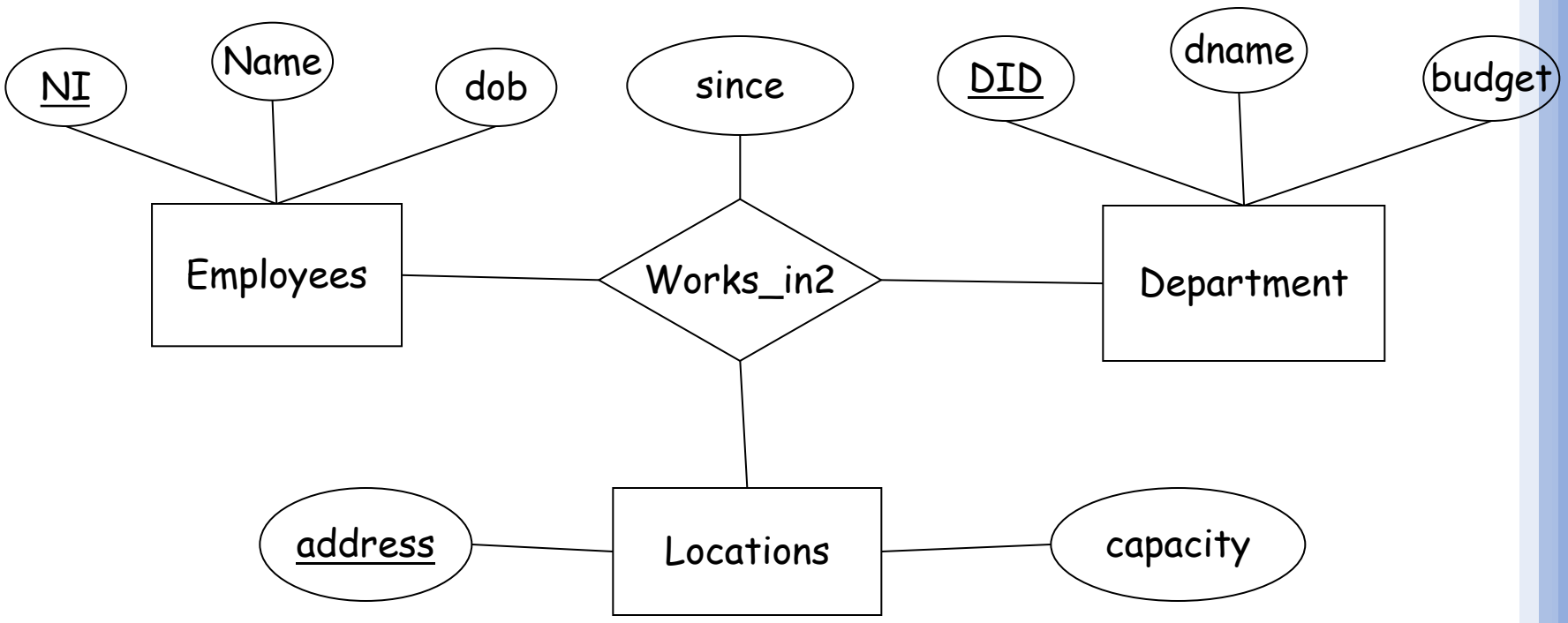
Binary Relationships

- 2 participating entity types
  - Teacher teaches Subject



N-ary Relationships

- Although relatively rare, n-ary relationships can exist
  - e.g. ternary (degree 3):



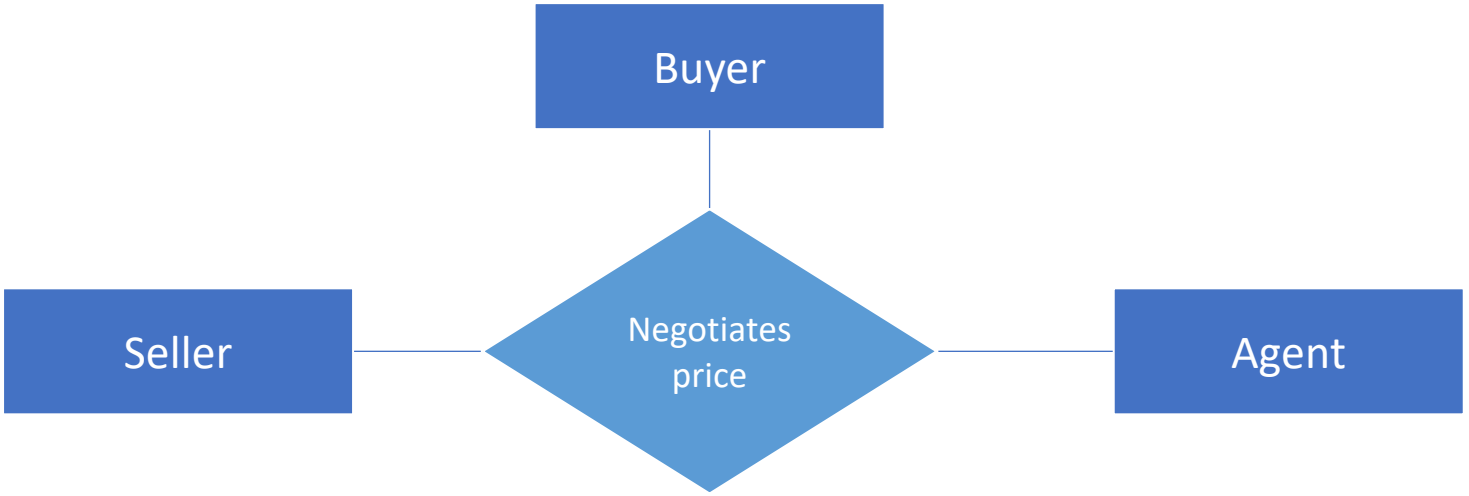
Remember: These are RARE!

Example: Ternary Relationships

3 participating entity types

- An agent negotiates the price between a seller and buyer

Example Ternary Relationship

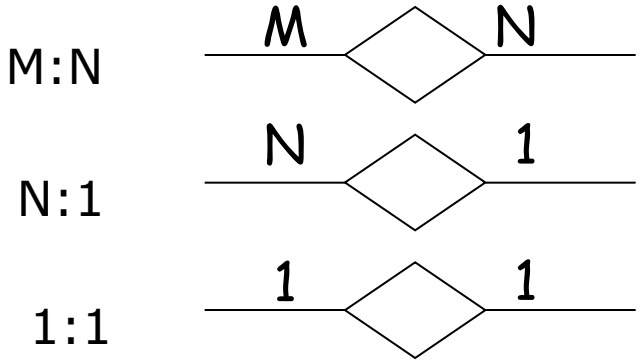


Remember: These are RARE!

Cardinality Constraints on Relationship Types

- For example:
  - An employee can work in many departments; a department can have many employees
  - In contrast, each department has at most one manager

- The cardinality specifies the number of entity instances that can participate from each side of the relationship of a binary relationship
  - One to one (1:1)
  - One to many (1:N)
  - Many to Many (N:M)



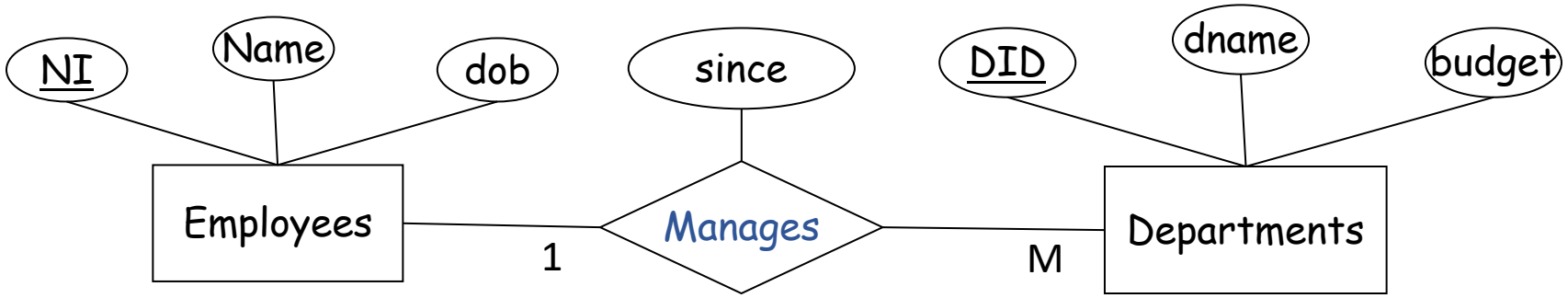
Note: Sometimes this is denoted using different arrowheads

Cardinality – 1:1

- One-to-one (1-1)

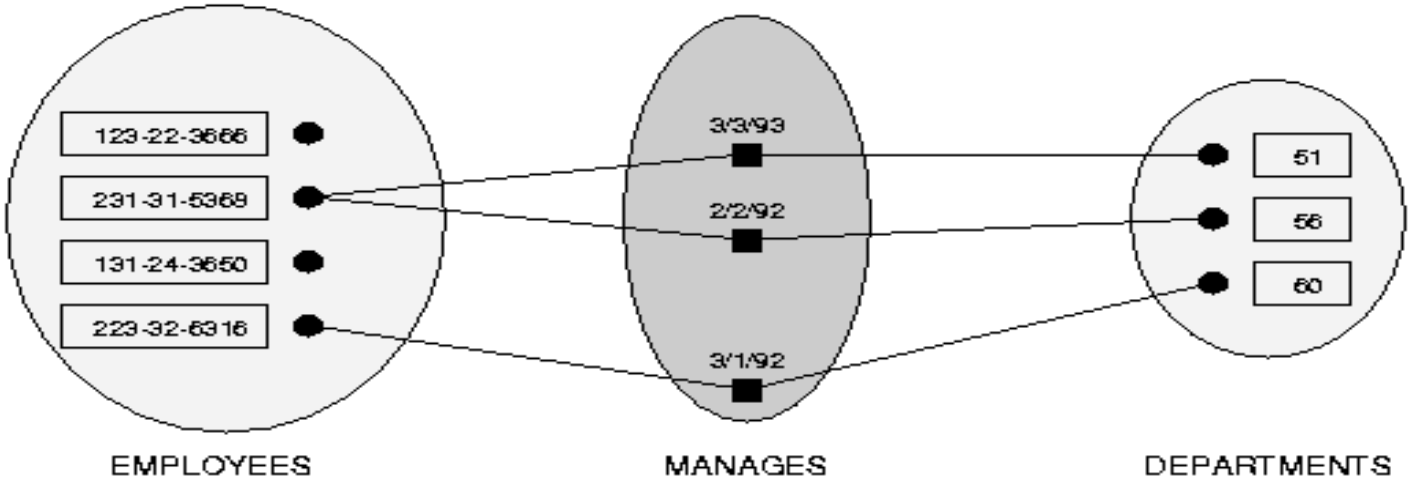
- Each manager manages ONLY one project
- Each project is managed by ONLY one manager

Cardinality – 1:N

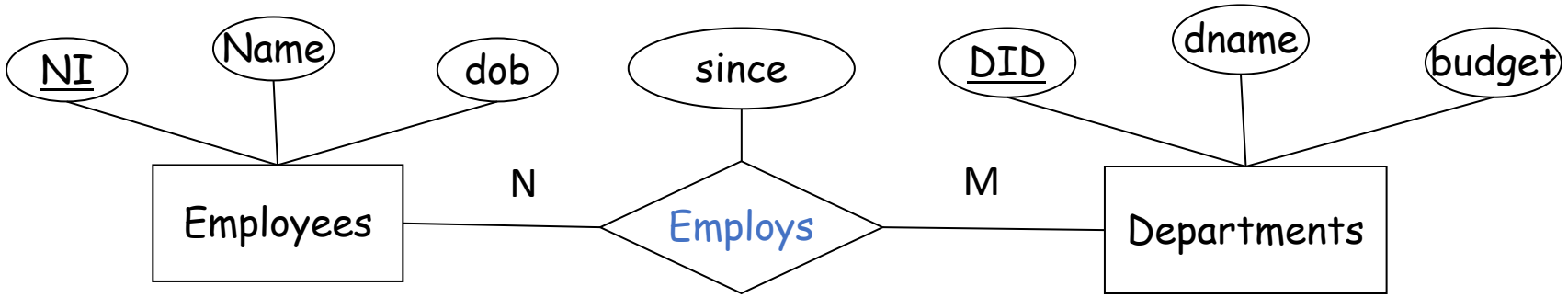


1:N --- One to many

A department cannot have more than one manager
(but it may be that an individual manages multiple departments)

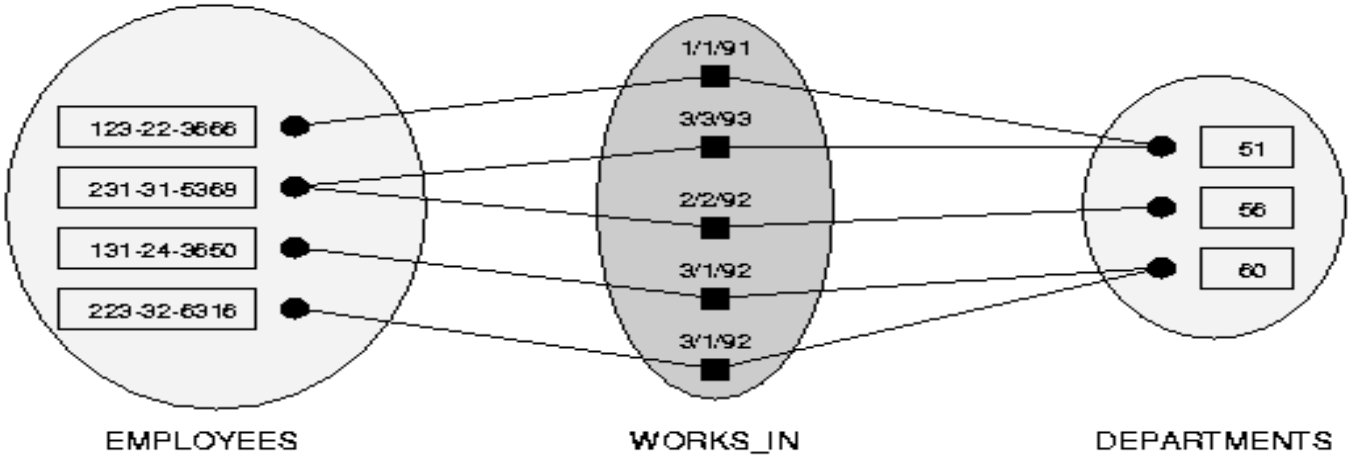


Cardinality – N:M



N:M -- Many to many

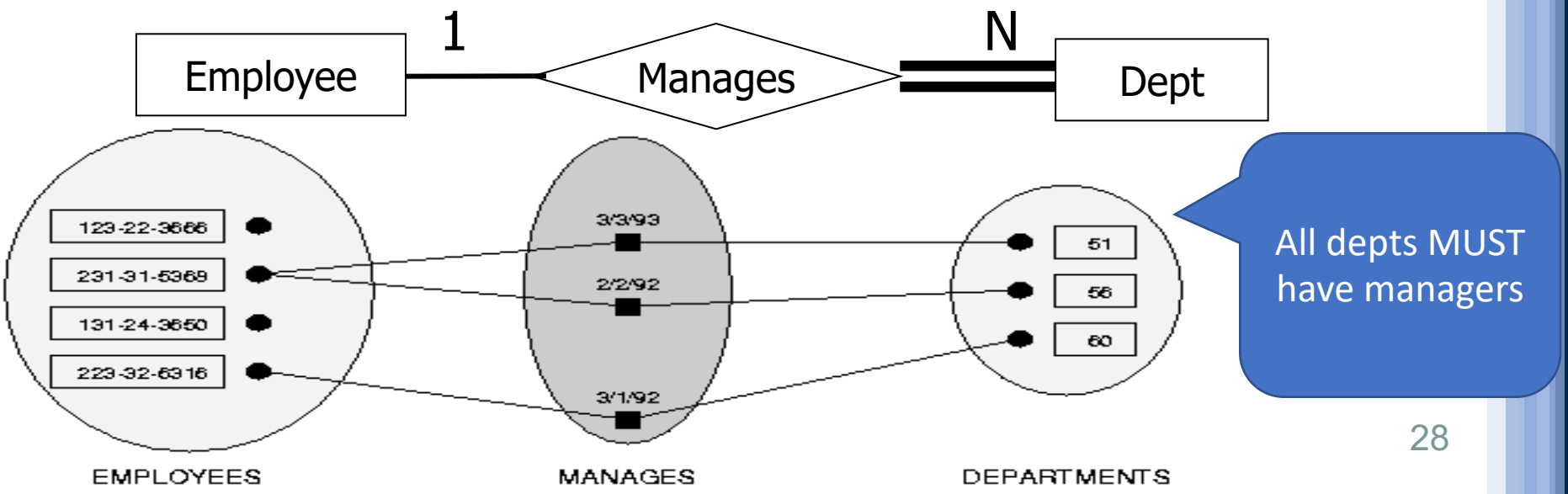
Departments may employ more than one person at a time, and an individual person may be employed by more than one department



Participation Constraints on Relationships

Every department must have a manager

- A double line indicates a *participation constraint* - totality
  - all entities in the entity set must participate in *at least one* relationship in the relationship set;



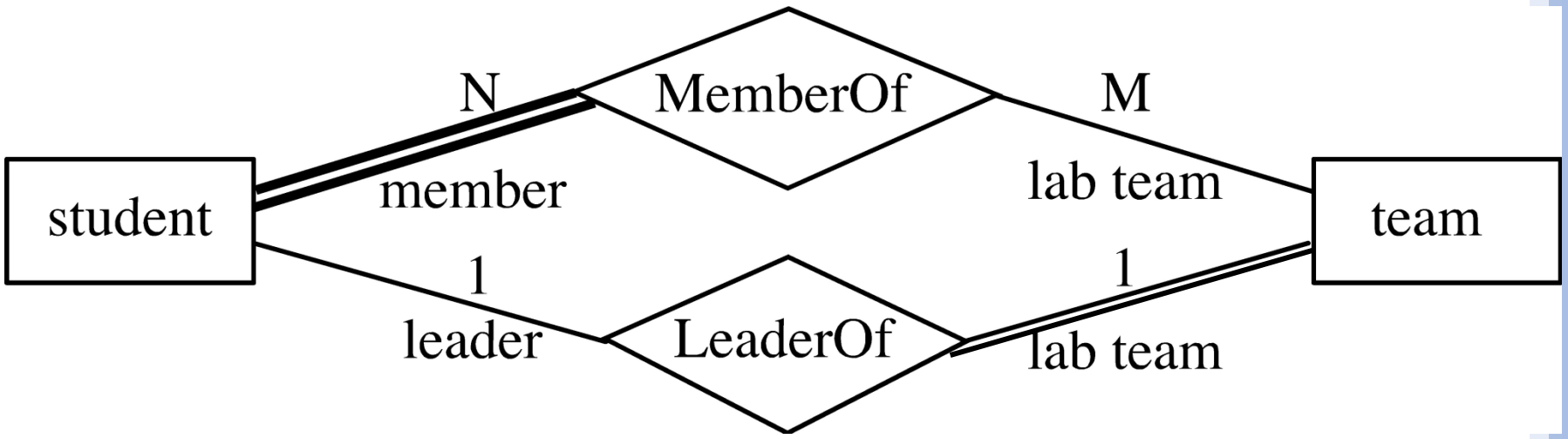
Participation Constraints on Relationships

Every department must have a manager

- A double line indicates a *participation constraint* - totality
  - all entities in the entity set must participate in *at least one* relationship in the relationship set;

Cardinality + Participation Constraints = Structural Constraints

Total Participation



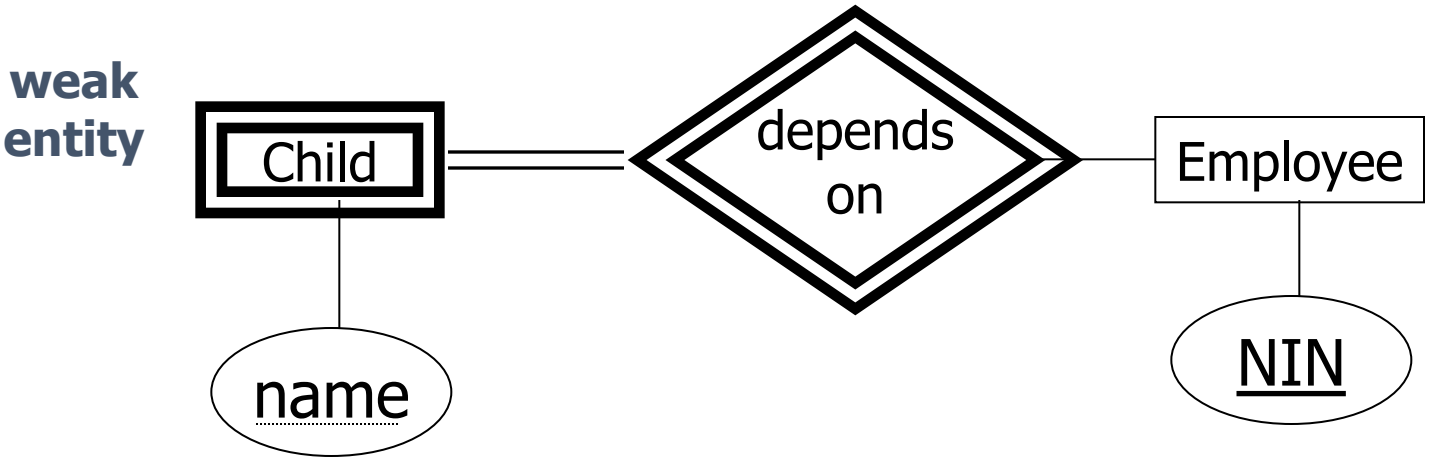
Every student **must** be a member of a team
Every team **needs** to have a leader

A double line indicates the total participation constraint in an ER model

Note - the participation of *student* in *LeaderOf* is **partial**,
because a student ***might*** be a team leader

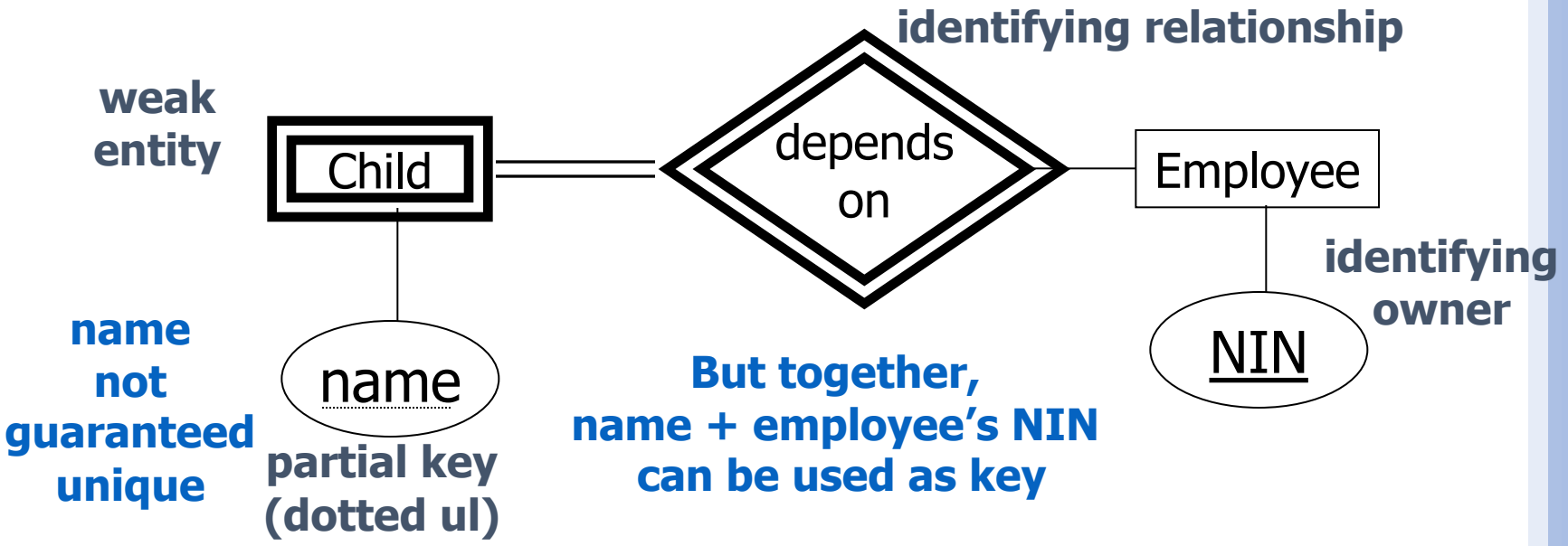
Weak Entity Types

- Do not have primary key (attributes) of their own
- Depend on other entities to guarantee uniqueness



Weak Entity Types

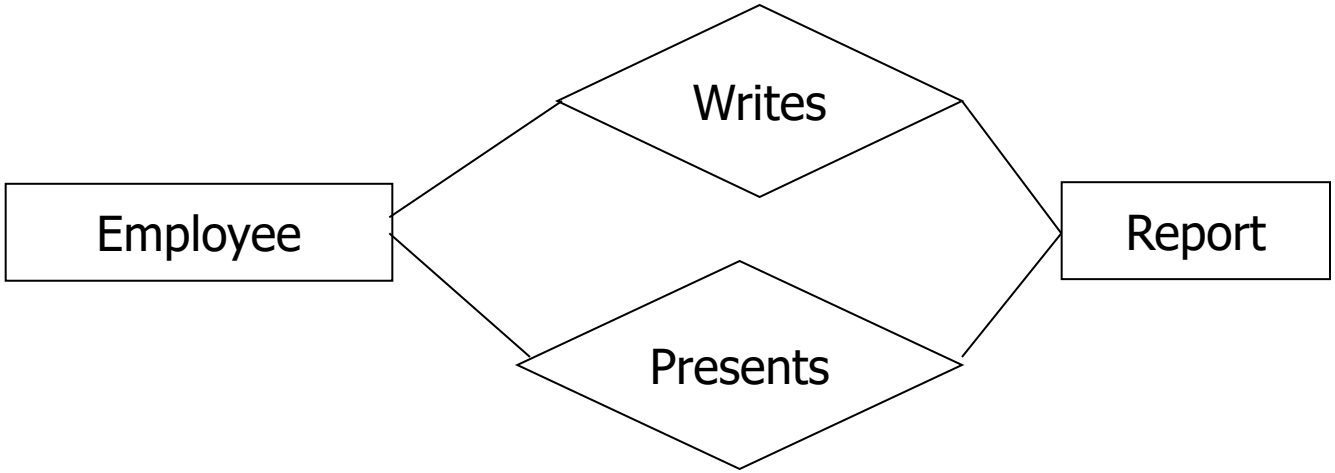
- Depend on other entities to guarantee uniqueness
- Do not have sufficient attribute(s) to form a primary key of their own



- A weak entity type must have **total participation** in this identifying relationship

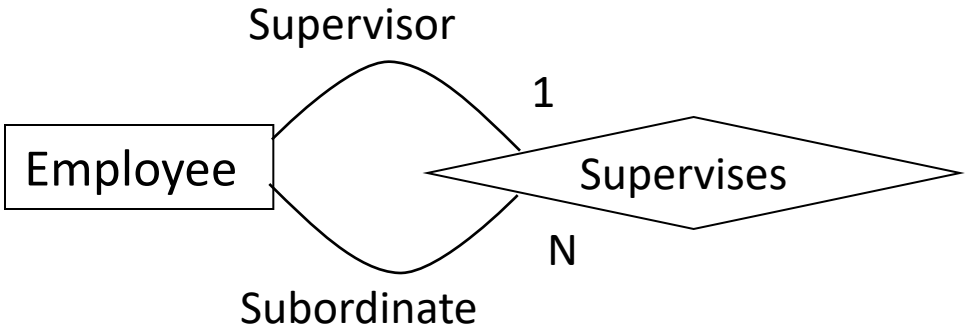
More on Relationships - 1

- There may be more than one relationship between entity types



More on Relationships - 2

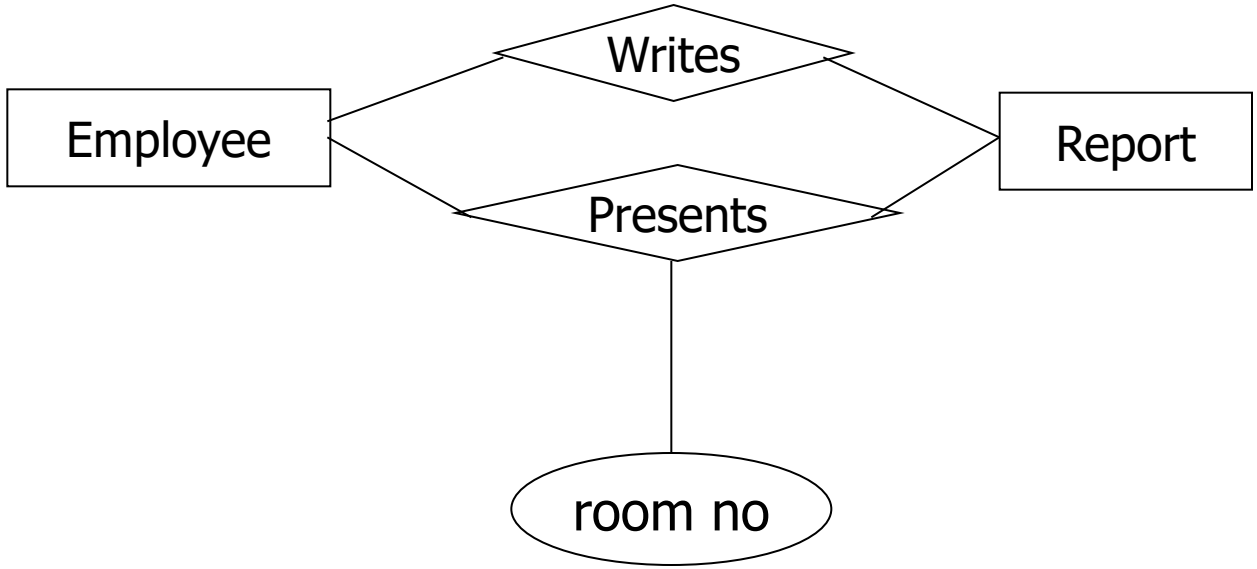
- An entity type may be in a relationship with itself
  - this is a recursive relationship



- We *name* the 'roles' of each side of the relationship

More on Relationships - 3

Recall: relationships may themselves have attributes



University of Glasgow

From written Scenario to an ER Model

- Identify the **Entities**, their **Attributes**, and all **Relationships** involved in any given scenario
- Represent this in an Entity-Relationship Diagram
- ER Diagram (and model) can then be used to implement the actual relationship tables in the database itself.

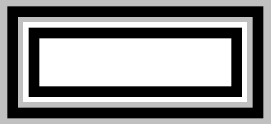
University of Glasgow

Constructing an ER diagram

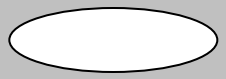
1. Identify the entity types (in boxes)
2. Identify each entity types' properties
3. Decide which properties are attributes (connected to entity in oval)
4. Decide which attributes could be keys
5. Select primary key (underlined attribute)
6. Determine which properties infer relationships (labelled diamond between the participating entities)
7. Decide on the cardinality and participation of the relationship (numbers at entities involved in relationship; single line Vs double line at entity)

ER Diagram Notation

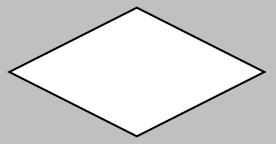
strong entity type



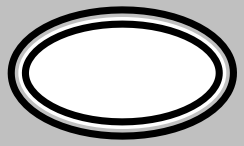
weak entity type



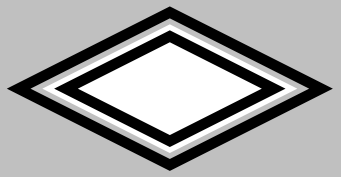
attribute



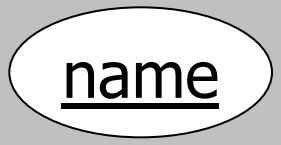
relationship



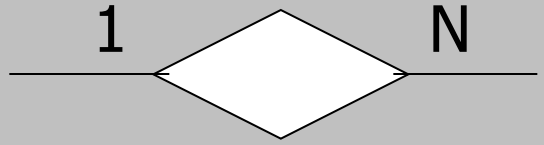
multi-valued attribute



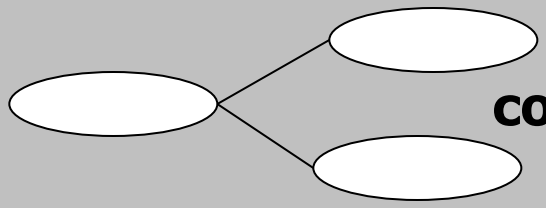
Identifying relationship



key attribute



relationship cardinality



composite attribute

total participation

partial participation