The University of Manchester

H4

Designing Databases:
The Entity-Relationship Approach

Fundamentals of Databases

Alvaro A A Fernandes, SCS, UoM

[COMP23111 Handout 04 of 12]

Acknowledgements

- These slides are adaptations (mostly minor, but some major) of material authored and made available to instructors by **Ramez Elmasri and Shamkant B. Navathe** to accompany their textbook **Database Systems: Models, Languages, Design, and Application Programming, 6th (Global) Edition, Addison-Wesley Pearson, 2011, 978-0-13-214498-8**
- Copyright remains with them and the publishers, whom I thank.
- All errors are my responsibility.

In Previous Handouts

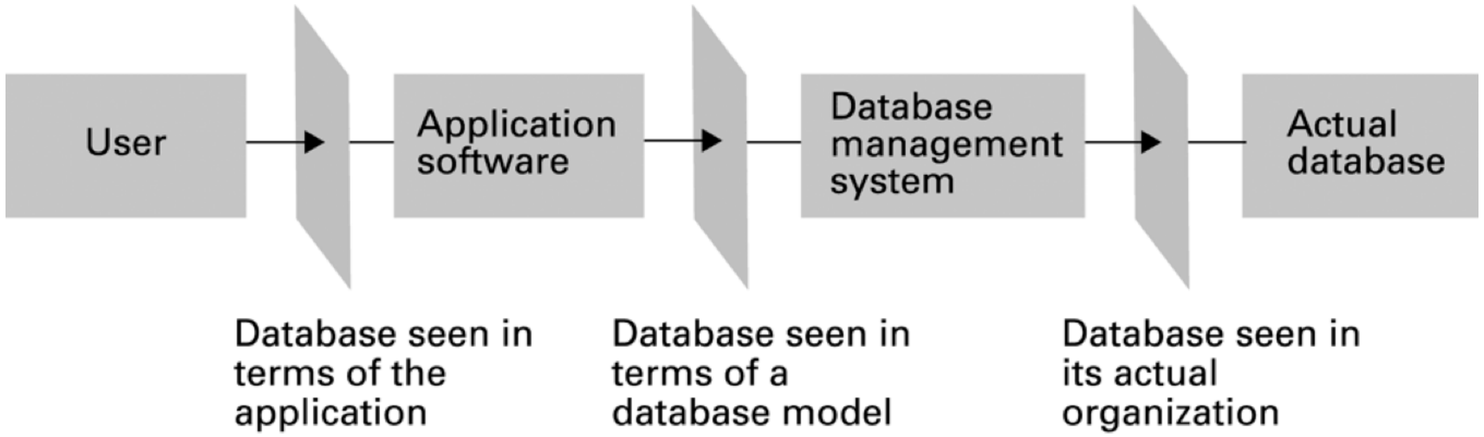
- We learned that data is an enterprise asset and that DBMSs are crucial to manage it well.
- We learned the importance of adopting different levels of abstraction in designing and implementing databases.
- We learned how data models lead to a distinction between schemas and instances that enables a logical view of the data.
- We learned about the main components of DBMSs and the various architectures used to deploy them.
- We learned about the relational approach to logical data modelling.
- We learned about the relational algebra and SQL, both its DDL and DML capabilities and its querying constructs.

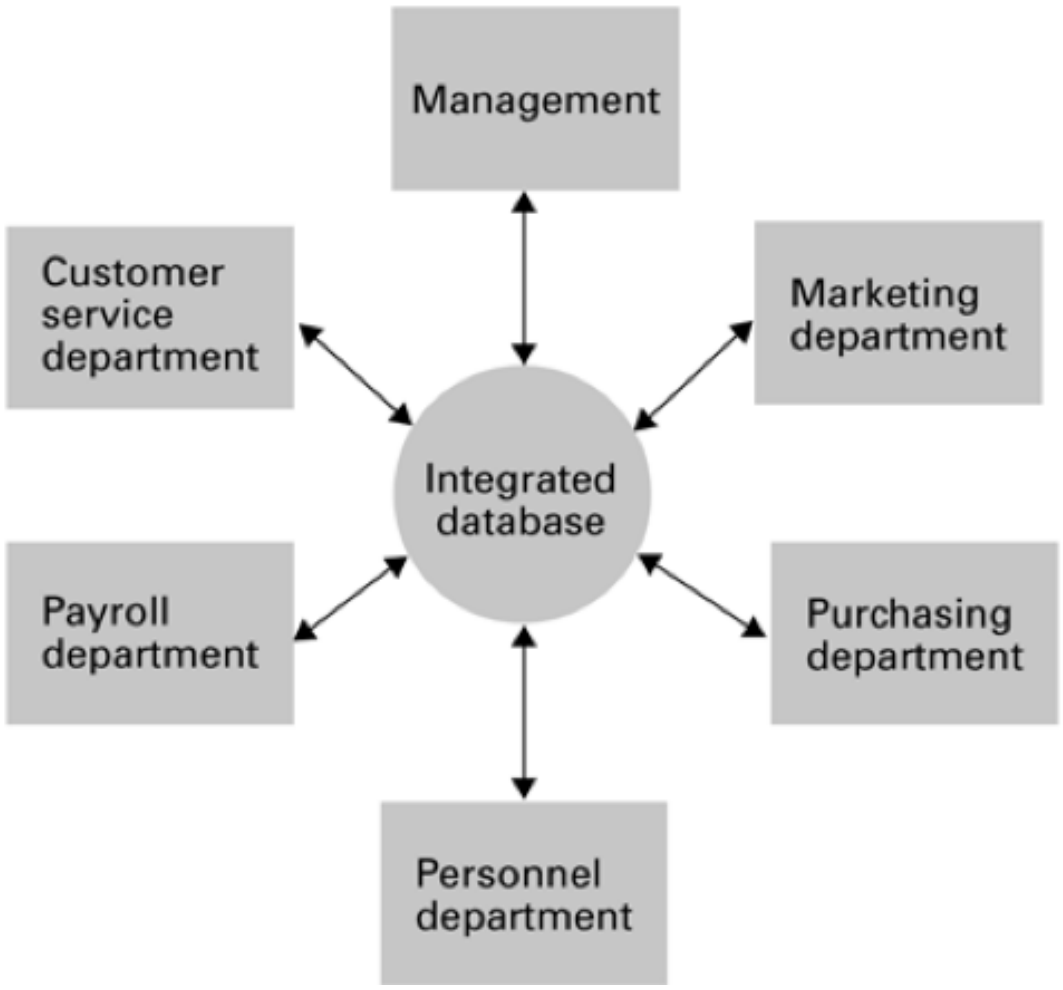
In This Handout

- What are database applications?
- Why conceptual modelling?
- What are the phases in the design of database applications?
- What is entity-relationship (ER) modelling?
- What are entity types and entity sets, attributes and value sets?
- What are relationship types, relationship sets and roles?
- What structural constraints are used in ER modelling?
- What guidelines are used for ER modelling?

Conceptual Design of Database Applications

- **Database applications** comprise:
  - an integrated database
  - plus application programs that
  - implement queries and updates to that database





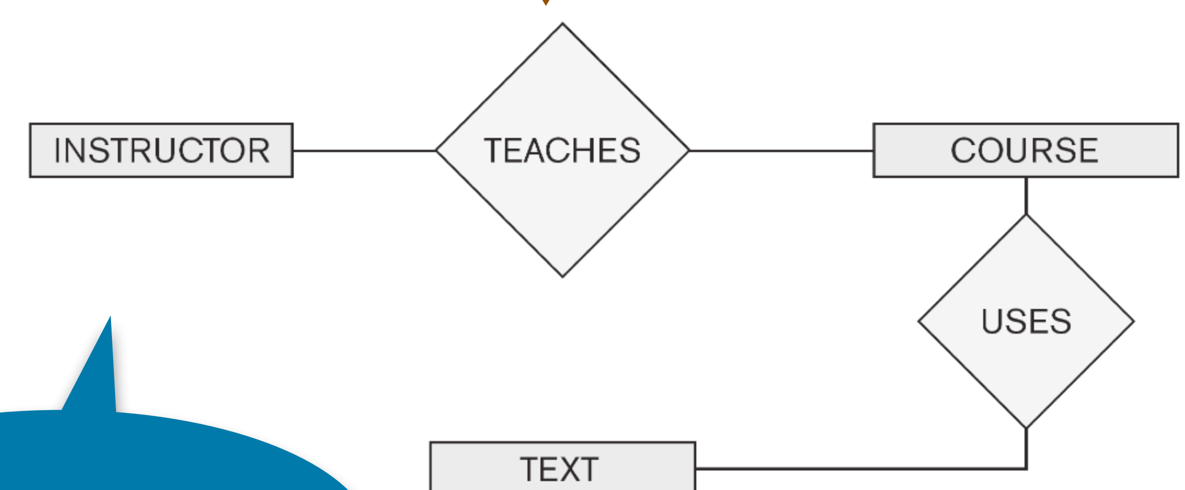
Conceptual Design of Database Applications

- **Entity-Relationship (ER) Modelling** is a popular, high-level conceptual data modelling approach
- Its simplicity makes it a good meeting point between end users and database designers
- It focuses on data requirements, not overall business logic

data requirements collection

Instructors teach courses.
Courses use texts.

conceptual design



conceptual model

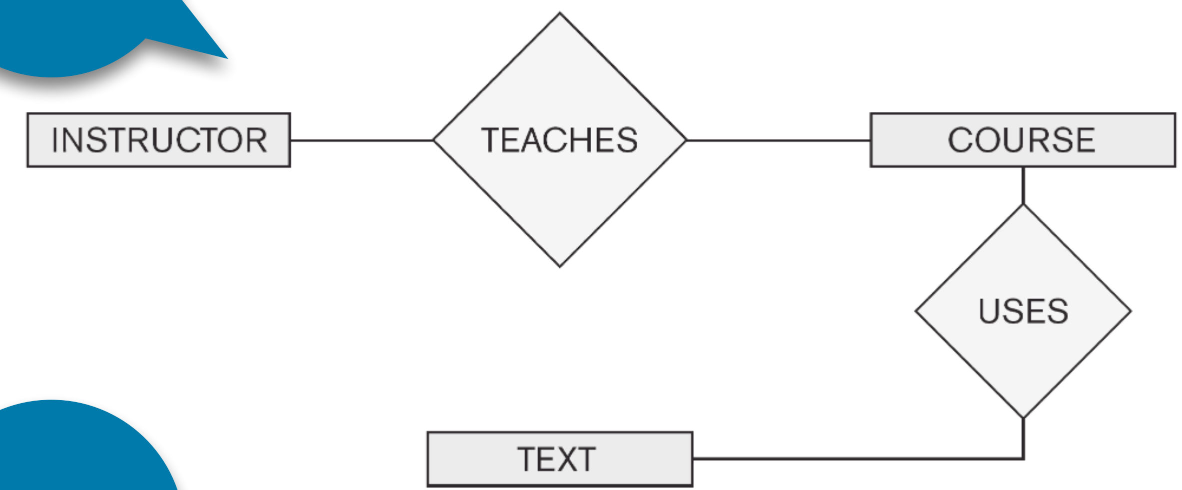
Conceptual Design of Database Applications

- An **ER Diagram (ERD)** is a diagrammatic representation for an ER model

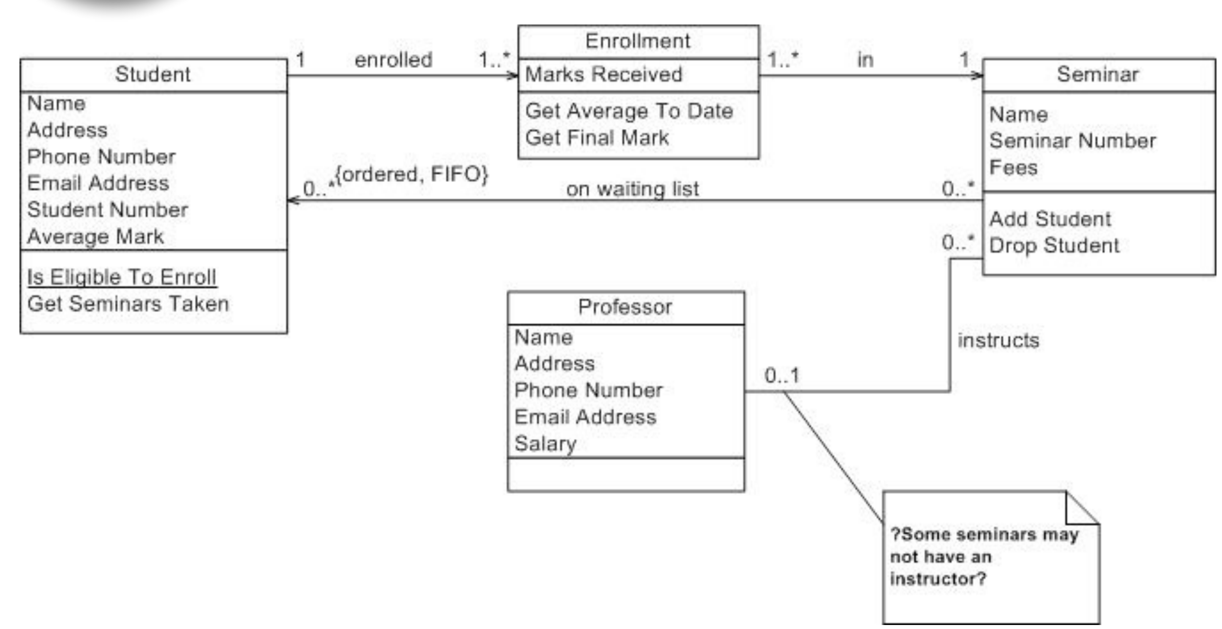
- The **Unified Modeling Language (UML)** supersumes the ERD language as it captures functional requirements too

- UML *class diagrams* are roughly equivalent to ERDs

ERD

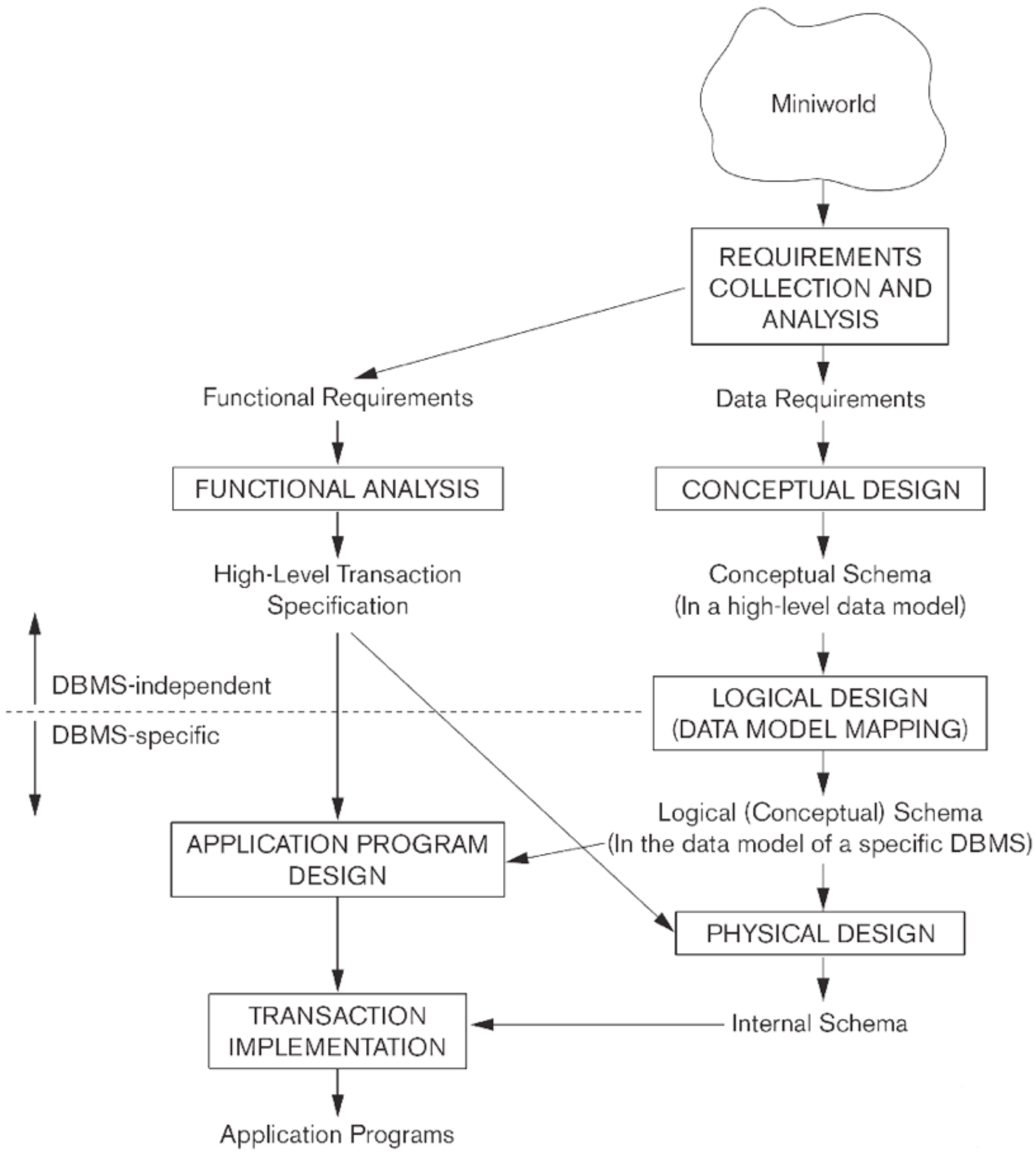


UML



http://www.agilemodeling.com/artifacts/classDiagram.htm

Database Application Design Phases



The Notion of a Miniworld

- A **miniworld** (or **universe of discourse**) comprises what lies in scope for the design activity
- It consists of the specific activities and processes in an organization that the envisaged database application is meant to support

Miniworld

- For example, assume the organization is a university.
- Assume the database application is to monitor programmes, student enrolment, course delivery, etc.
- The activities in this miniworld include:
  - Course units having a syllabus and designated support textbooks.
  - Course units having coursework (with activities, tasks, deadlines) and exams, and both have marks and earn credits.
  - Course units are (mandatory or optional) part of a degree programme.
  - Course units are taught by instructors in lectures that are held at particular locations at particular times.
  - etc.

Requirements Collection and Analysis

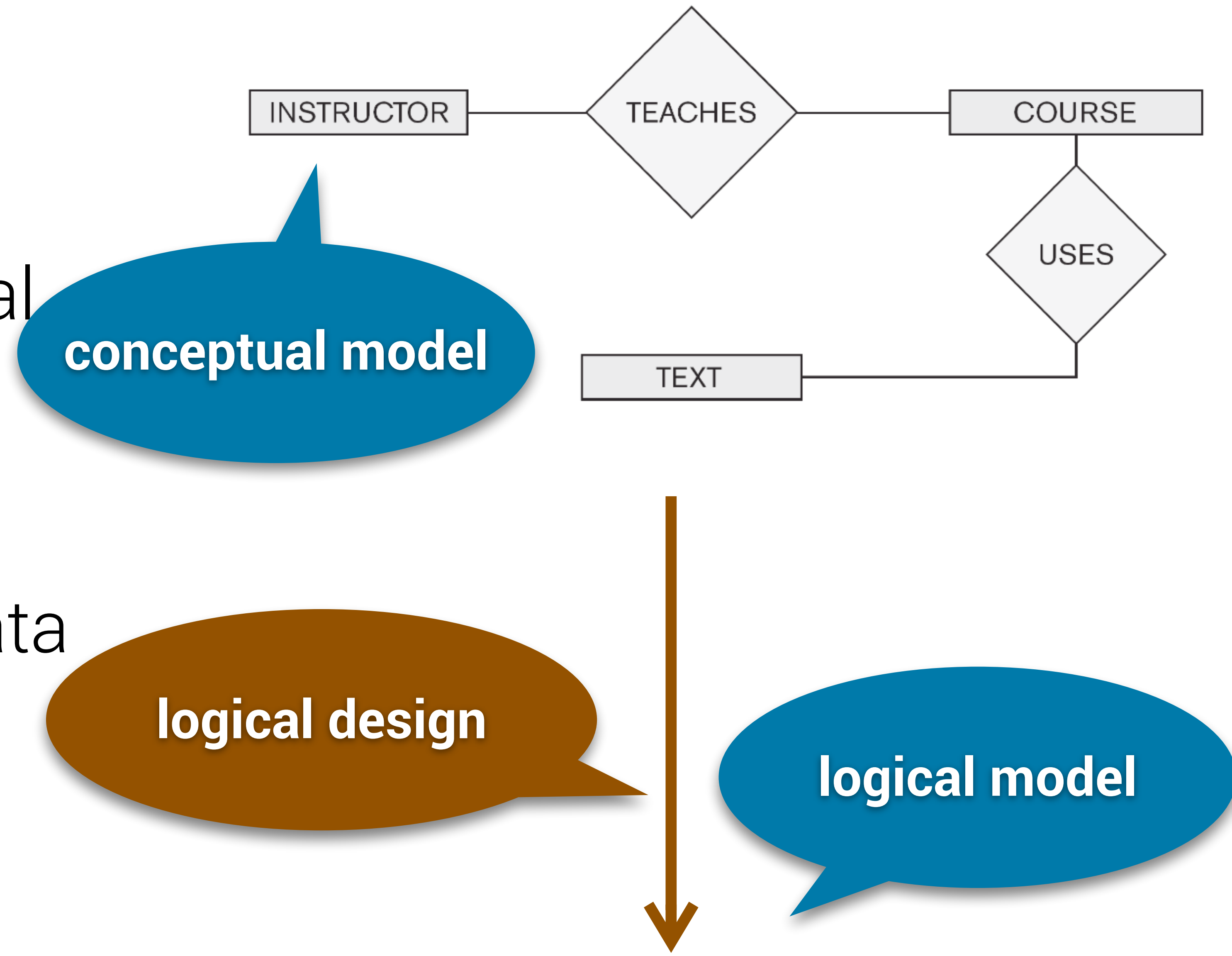
- The database application designers interview the prospective database users
- The common goal is to understand and document data and (related) functional requirements

Requirements Collection and Analysis

- The desired outcomes are:
  - **Data requirements**, which are input to conceptual design
  - **Functional requirements**, which are input to functional analysis (but also informs physical design, later on)

From Conceptual to Logical Models

- A **conceptual model** (also called a **conceptual schema**) is the main outcome of the conceptual design phase.
- It is a rigorous, but high-level, description of the data requirements
- It includes detailed descriptions of the entity types, relationship types, and of the various constraints on the latter

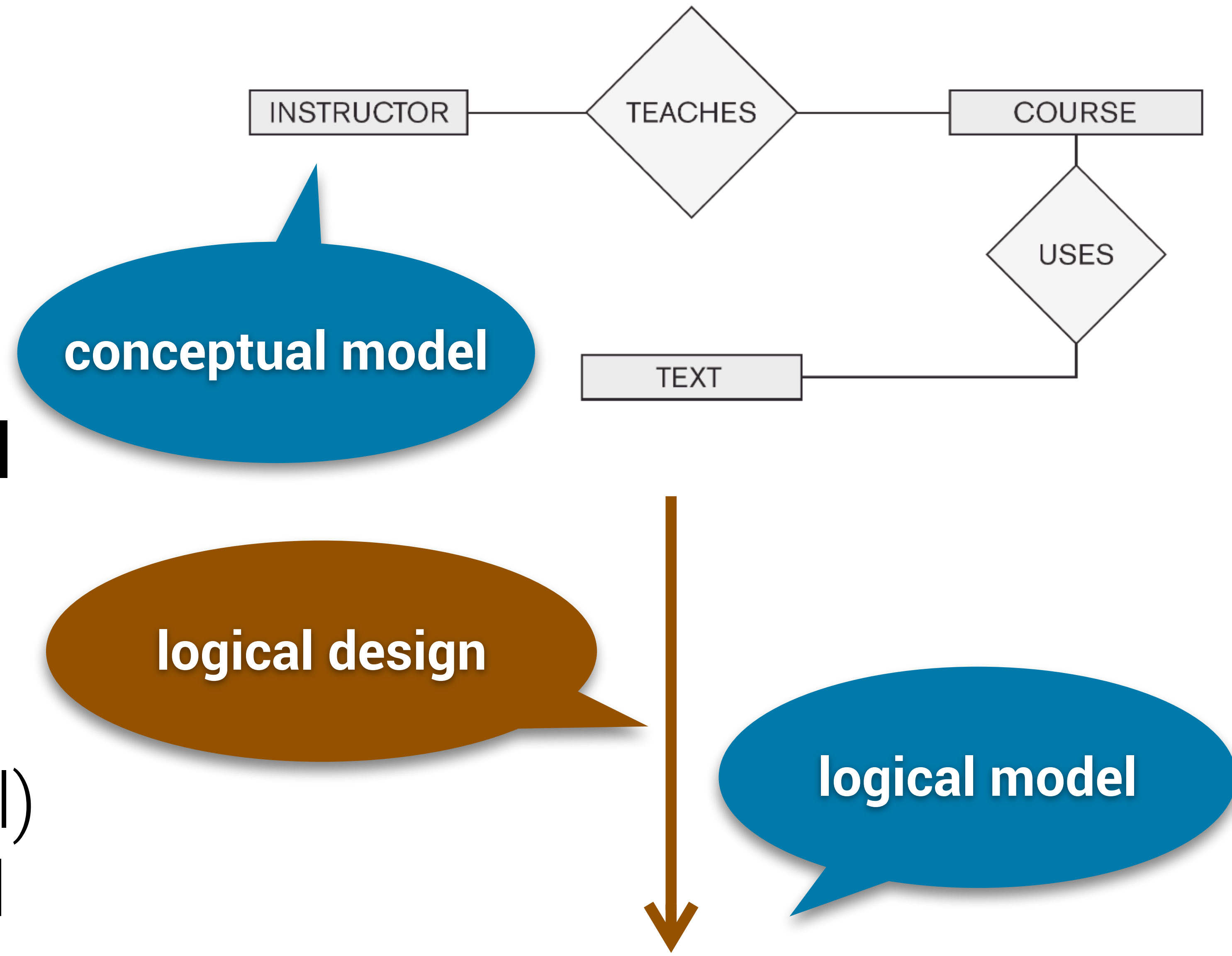



```
instructors(i_id, i_name, ...)
teaches(i_id, c_id, ...)
course(c_id, c_name, ...)
use(c_id, t_id, ...)
text(t_id, t_name, ...)
```

From Conceptual to Logical Models

- A conceptual model is suitable to being (reasonably formally) mapped into an (**implementation**) **logical model** or **schema**

- By a **logical schema** we mean one (e.g., relational) that is directly supported by the DBMS we intend to implement the database application on




```
instructors(i_id, i_name, ...)
teaches(i_id, c_id, ...)
course(c_id, c_name, ...)
use(c_id, t_id, ...)
text(t_id, t_name, ...)
```

From Logical to Physical Models

- Logical design (e.g., as achieved by conceptual-to-logical model-mapping) is the phase that transforms the conceptual schema into a logical schema.

- It need not be informed by functional analysis but can benefit from knowledge of information about likely sizes and loads, access patterns, etc.

From Logical to Physical Models

- The physical design phase considers in detail the internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files specified
- It is where/when performance implications are considered in light of detailed knowledge of sizes, loads, etc.

A Sample Database Application

- Assume our miniworld encompasses the employees, departments and projects of a company.
- The company is organized into departments, which have a name, a number and a location (possibly more than one).
- Each department is managed by a manager, who is an employee, and we wish to store the date in which the latter started managing the department.
- Each department controls various projects.

A Sample Database Application

- An employee works for only one department but may work on more than one project (not necessarily controlled by the same department).
- We wish to store how many hours an employee works on each project and which employee supervises which employee.
- For each employee, we wish to store the employee's name, social security number, address, salary, gender, and birth date
- We also wish to store information about the dependents of each employee, viz., their name, gender, birth date, and relationship to the employee.

ER Modelling: A Word of Caution

- Note that, in terms of the phases in database design, we are now taking a step back from the notions in logical modelling we have been studying through the relational model.
- There are very obvious similarities but beware the differences too

ER Modelling: A Word of Caution

- Some of the differences are:
  - ER models are conceptual models
  - ER models are not logical models like relational ones
  - Conceptual models (like ER models) are more abstract (i.e., contain less directly-implementable detail) than logical models (like relational ones)
  - Conceptual models are not directly supported by concrete DBMSs
  - Conceptual models have no associated languages (like RA or SQL)
  - In conceptual modelling, we mostly dwell at the schema level, whereas instances are only considered in very abstract terms

ER Modelling: Basic Constructs

- The basic constructs in ER modelling capture the data requirements in the form of:
  - Entity types
  - Attribute types
  - Relationship types

ER Modelling: Entities and Attributes

- An **entity** is a notion or concept in the real world that has an independent existence

- It can be *concrete* (e.g., a car, a person) or *abstract* (e.g., a course, a project)

- An **attribute** is a particular property value that contributes to characterize a particular entity

- Both entities and attributes are captured as types in a conceptual model/schema.

ER Modelling: Attribute Types

- Attribute types can be:
  - **Composite** v. **simple** (or **atomic**)
  - **Single-valued** v. **multivalued**
  - **Stored** v. **derived**
  - **NULL-valued**
  - **Complex-valued** (i.e., arbitrarily-nested composite/ multivalued)
- Try not to confuse the notion of a '*composite*' attribute type that of a '*complex*' one

ER Modelling: Example Attribute Types

- Simple
  - *height*
- Composite
  - *(given-name, family-name)*
- Single-Valued
  - *birth-date*
- Multivalued
  - *degrees-obtained*

ER Modelling: Attribute Types

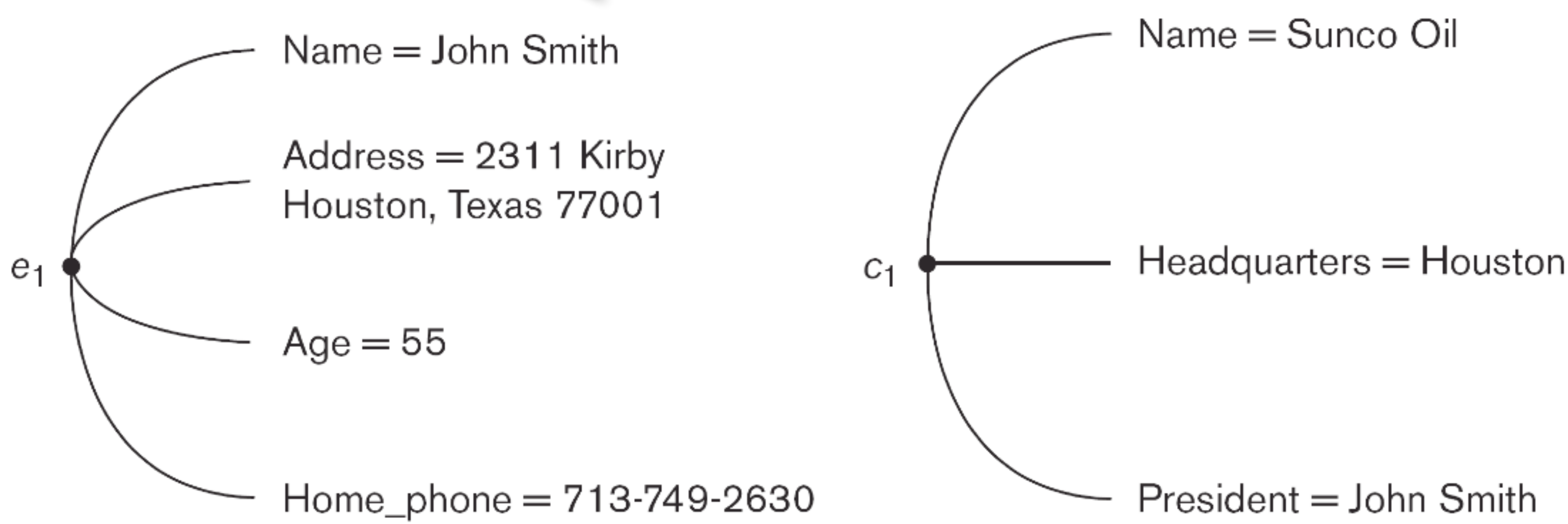
- Stored
  - *birth-date*
- Derived
  - *current-age*
- NULL-valued
  - Not applicable: *flat-number* (if a single-household house)
  - Missing: *height* (it must exist, but we may not know its value)
  - Not known: *phone-number* (there may or may not be one, we don't know)

ER Modelling: Attribute Types

- Complex-valued
  - {*x*}: multivalued, i.e., a collection, e.g., a set of *x* values
  - (*t*,*u*,*v*): composite, i.e., a non-atomic structure, e.g, a tuple of *t*, *u*, and *v* values
  - {*contact-det* ({*phone* (*area*, *phone#*)}, *address* (*street*, *building#*, *city*, *zip*))}: a combination of many attribute types

ER Modelling: Example Entities and Attributes

Two entities, EMPLOYEE e_1 and COMPANY c_1, with their attributes



not ERD notation!

ER Modelling: Entity Types

- An **entity type** is a collection of all entities that have the same attributes (i.e., share the same structure)
- It is a schema-level notion, analogous to a class in object-oriented approaches

ER Modelling: Entity Types and Entity Sets

- An **entity set** is a collection of specific, individual entities of a particular entity type that belong to the miniworld at a given point in time

- It is an instance-level notion, analogous to the extension (or extent) of an object-oriented class (i.e., the collection of objects that belong to the class at a given point).

ER Modelling: Example Entity Types and Sets

Two entity types, EMPLOYEE and COMPANY, with some member entities

Entity Type Name:	EMPLOYEE	COMPANY
	Name, Age, Salary	Name, Headquarters, President
Entity Set: (Extension)	e_1 • (John Smith, 55, 80k)	c_1 • (Sunco Oil, Houston, John Smith)
	e_2 • (Fred Brown, 40, 30K)	c_2 • (Fast Computer, Dallas, Bob King)
	e_3 • (Judy Clark, 25, 20K)	⋮
	⋮	

not ERD notation!

ER Modelling: Keys

- In ER modelling, a **key** (or **uniqueness**) **constraint** holds.
- Entities must be uniquely identifiable in every possible entity set of an entity type.
- Keys are attributes whose values are distinct for each individual entity in any entity set.
- There can be more than one key in an entity type.
- An entity type that has no natural or assigned key is said to be **weak**.

ER Modelling: A Word of Caution on Keys

- Try not to confuse the ER notion of 'key' with the relational ones of 'primary key' and 'foreign key'.

- The notion of key in ER modelling is related (but not identical) to the notion of a primary/foreign key in a relational database.

- It is best compared to the relational notion of a candidate key, which a designer may choose to make primary or not.

- Also, foreign keys model relationships implicitly in the relational model, but the 'R' in 'ER' tells you that relationships are modelled explicitly in the ER model.

ER Modelling: Attribute Types and Value Sets

- A **value set** (or **domain of values**) specifies the set of values that may be assigned to a given attribute for each individual entity of a given entity type.

- In ER modelling, we assume that value sets are implicitly grounded on mappings to classical data types (e.g., integers, floats, strings, etc.).

- A attribute A of an entity set E whose value set is V is a function from E to the power set $P(V)$ of V, i.e.,

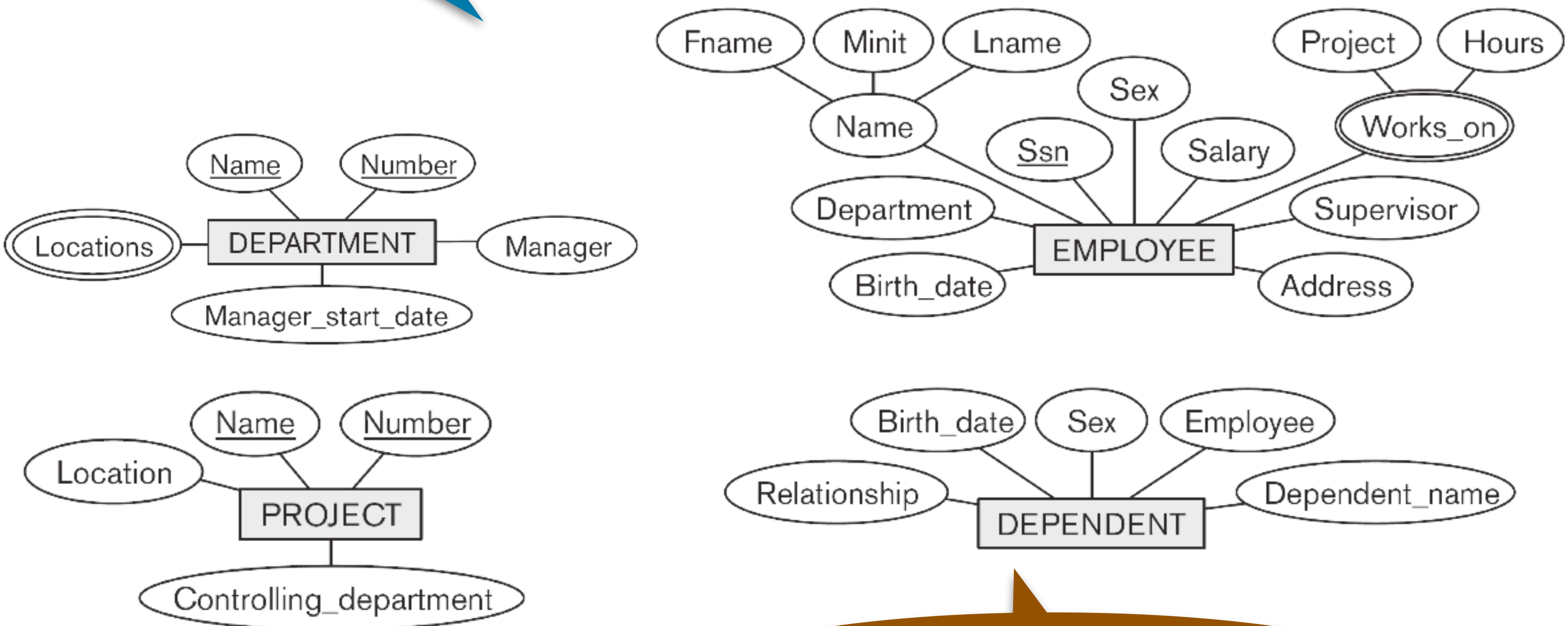
  $$A: E \rightarrow P(V)$$

- We denote the value of the attribute A of an entity e by $A(e)$.

Conceptual Design: COMPANY Database Version 0

Preliminary design of entity types for the COMPANY database.

ERD notation!



Some attributes will be reconceptualized as relationship types later

ER Modelling: Relationship Types

- A **relationship type** is often *implicitly* captured in verbal specifications when an attribute of one entity type seems also (or refers to) an attribute of another entity type.

- The relational approach to logical modelling adopts and formalizes this view

- However, in the ER model, we represent such references *explicitly* as relationship types, rather than as attributes that have shared domains (as the relational model does).

ER Modelling: Relationship Types

- A relationship type R among n entity types E_1, E_2, ..., E_n defines a set of possible associations among the participating entity types.
- This is, like entity type, a schema-level notion.
- The corresponding instance-level notion is that of a relationship set.

ER Modelling: Relationship Sets

- A **relationship set** r contains a set of associations, or **relationship instances** (i.e., n-tuples), r_i
  - Each r_i associates n individual entities (e_1, e_2, ..., e_n)
  - Each entity e_j in r_i is a member of the entity set E_j
- The relationship set r is a subset of the Cartesian product of the entity sets of the participating entities, i.e.

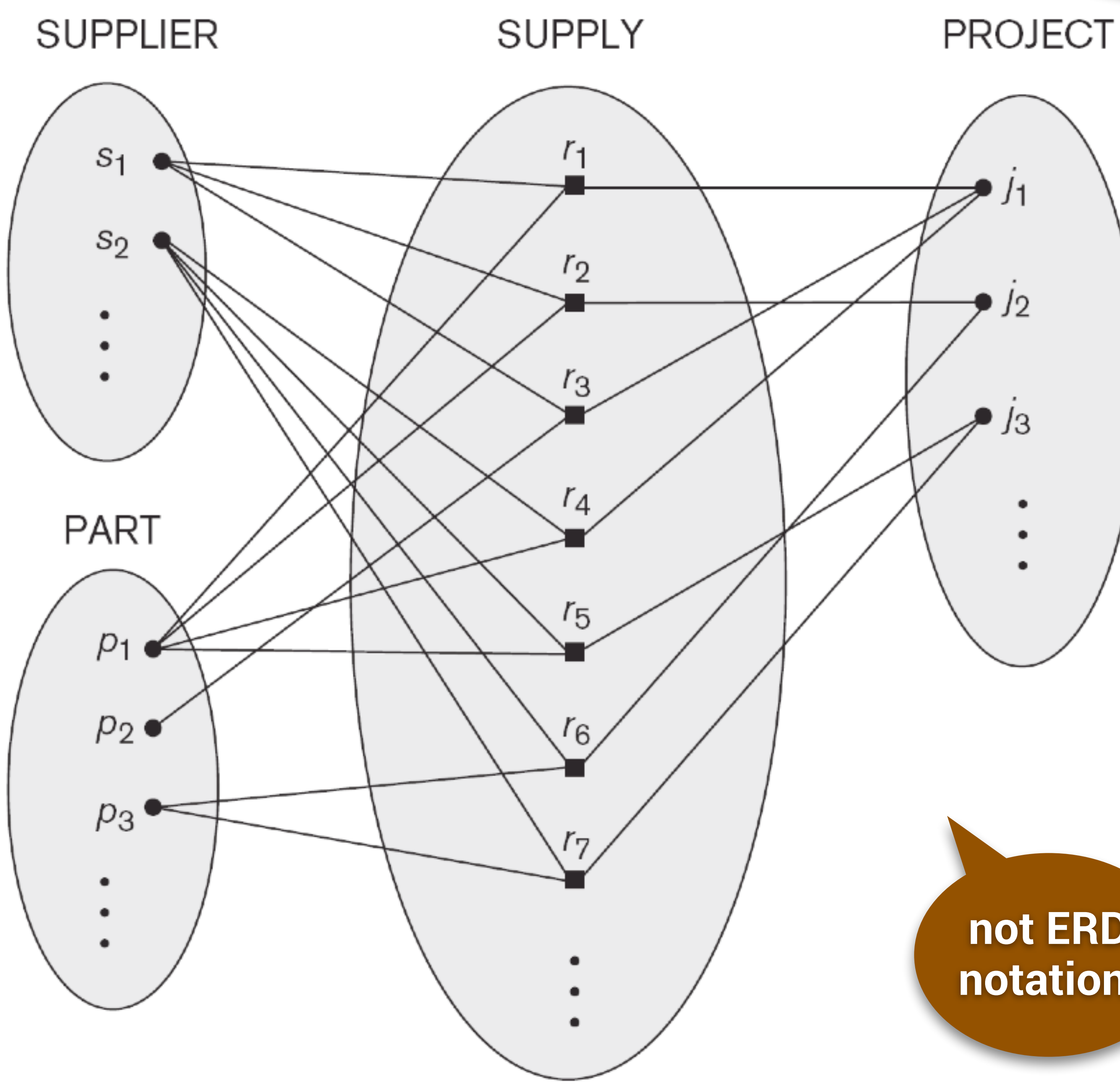
$$r \subseteq E_1 \times E_2 \times \ldots \times E_n$$

ER Modelling: Relationship Degree

- The **degree** of a relationship type is the number of participating entity types.
- So, a binary relationship has two participating entities; a ternary relationship, three; etc.

ER Modelling: Example Relationship Set

Some relationship instances in the SUPPLY ternary relationship set

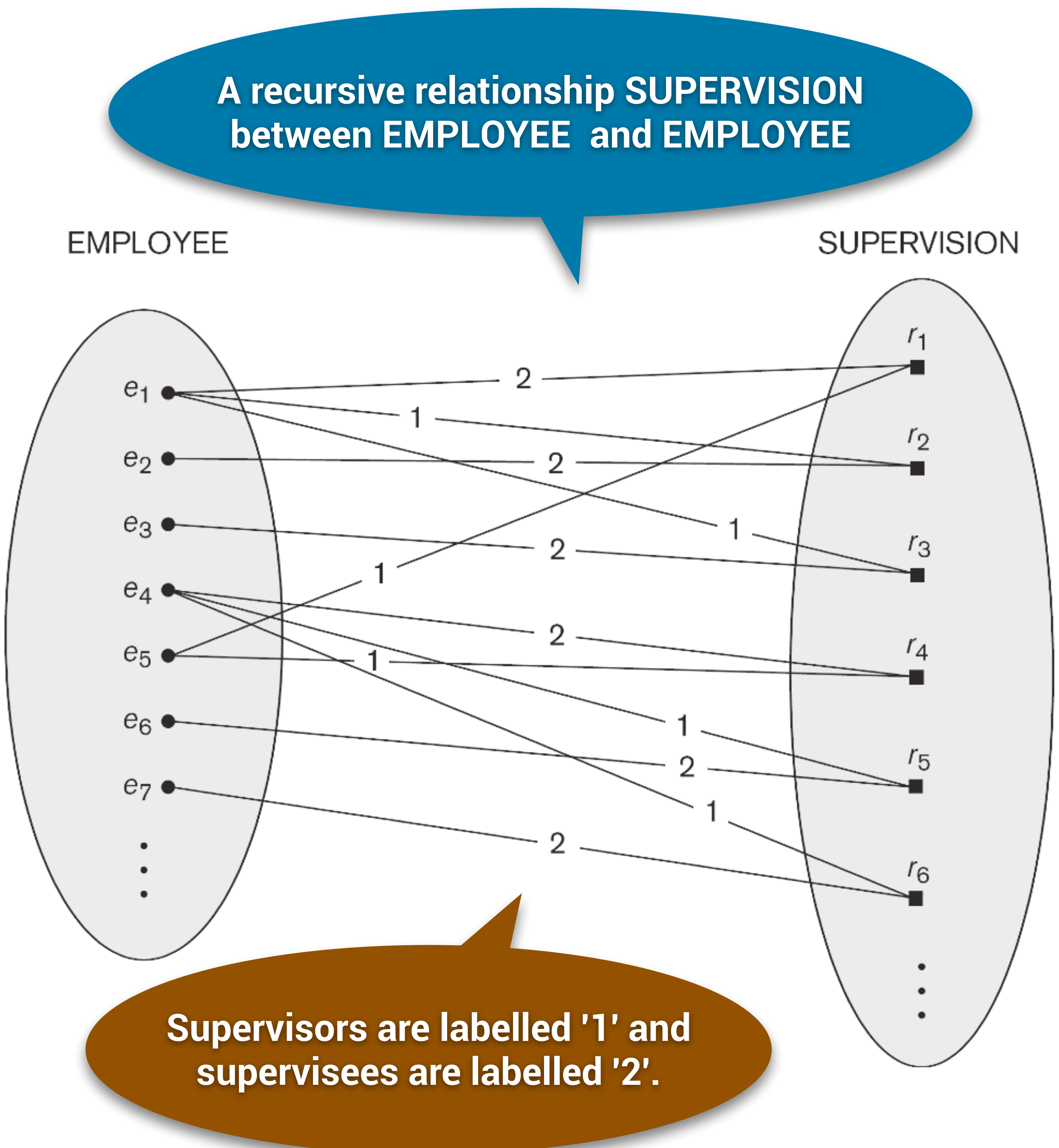


not ERD notation!

ER Modelling: Recursive Relationships and Role Names

- In a **recursive relationship**, the same entity type participates more than once in a relationship type in different roles.
- For example, an employee may supervise many employees.

ER Modelling: Example Roles in Recursive Relationships



- A **role name** can be used to disambiguate the part that a participating entity plays in a relationship.
- When the same entity type appears more than once in a relationship type, role names are very useful.
- When all the entity types in a relationship type are distinct, roles are not as useful.

ER Modelling: Constraints on Relationship Types

- A **cardinality ratio constraint** specifies the maximum number of relationship instances that an entity can participate in

- It can be **1:1**, **1:N**, **N:1**, **N:M** (or, equivalently, **M:N**)

- In this notation, we can only specify no-specific-maximum (i.e., *M*, and *N*) or a maximum of *one*.

- Other notations allow *M*, and *N* to be fixed to a particular number.

ER Modelling: Constraints on Relationship Types

- A **participation constraint** specifies whether the existence of the entity depends on its being related to another entity
- In other words, it states the minimum number of relationship instances in which an entity must participate in that relationship type.
- A participation constraint is sometimes called a **minimum cardinality constraint**.
- If participation is **total**, then <u>every</u> entity in the total entity set <u>must</u> participate in some relationship instance
- Total participation is also called an **existence dependency**
- If participation is **partial**, then <u>some</u> entities <u>may not</u> participate in any relationship instance
- Cardinality ratio and participation constraints are called **structural constraints** on relationship types.

ER Modelling: Attributes of Relationship Types

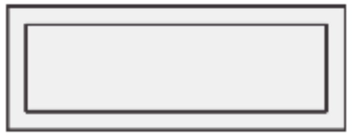
- Relationship attributes involved in 1:1 or 1:N relationship types can be migrated to one of the entity types.
  - For a **1:1** relationship type, the relationship attribute can be migrated to *either of the entity types* in the relationship.
  - For a **1:N** or a **N:1** relationship type, the relationship attribute can be migrated *only to the entity type on the N-side* of the relationship.
- For an **N:M** relationship type, both the participating entity types contribute attributes that determine the relationship, i.e., is not possible to do away with a relationship type by migrating the relationship attributes on the participating entity types.

ER Modelling: Weak Entity Types

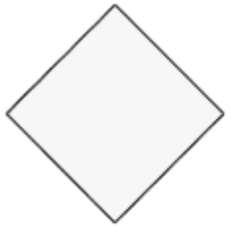
- A **weak entity type** does not have key attributes of their own
- It can only be properly identified (i.e., retain its conceptual status and have an associated entity set) by being related to specific entities of some other entity type, called the **owner**
- An **identifying relationship** is one that relates a weak entity type to its owner, and **always** has a *total* participation constraint

Summary of ER Diagram Notation

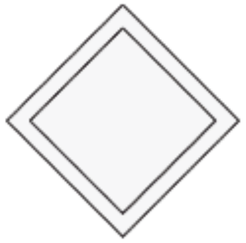
Entity



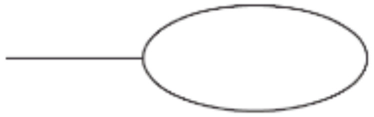
Weak Entity



Relationship

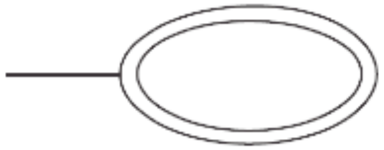


Identifying Relationship



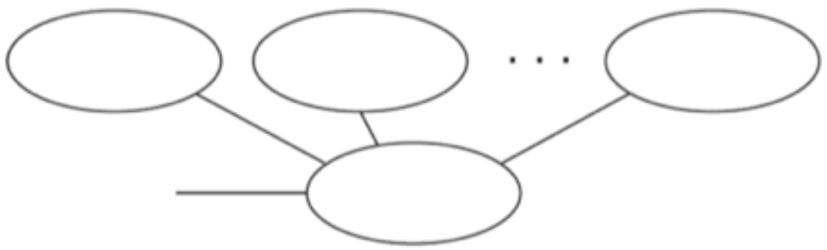
Attribute

Key Attribute



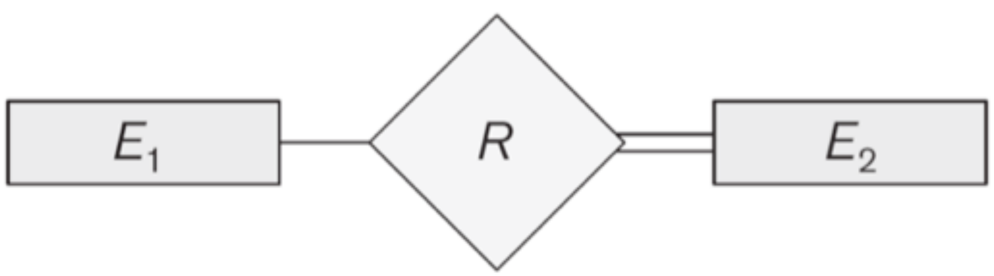
Multivalued Attribute

Summary of ER Diagram Notation

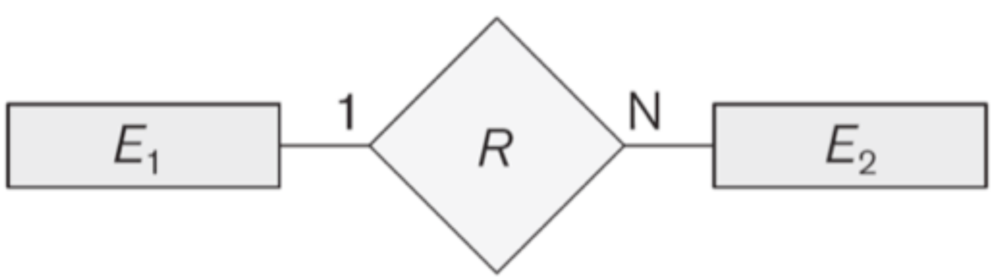


Composite Attribute

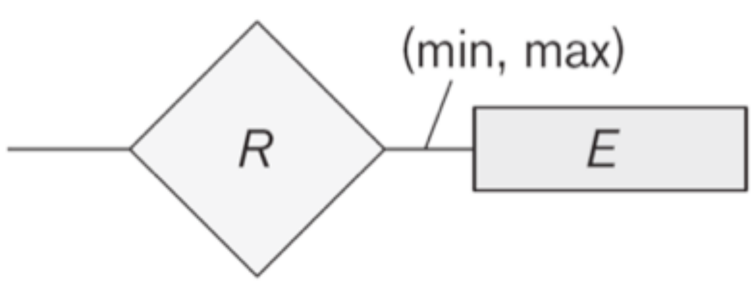
Derived Attribute



Total Participation of E_2 in R



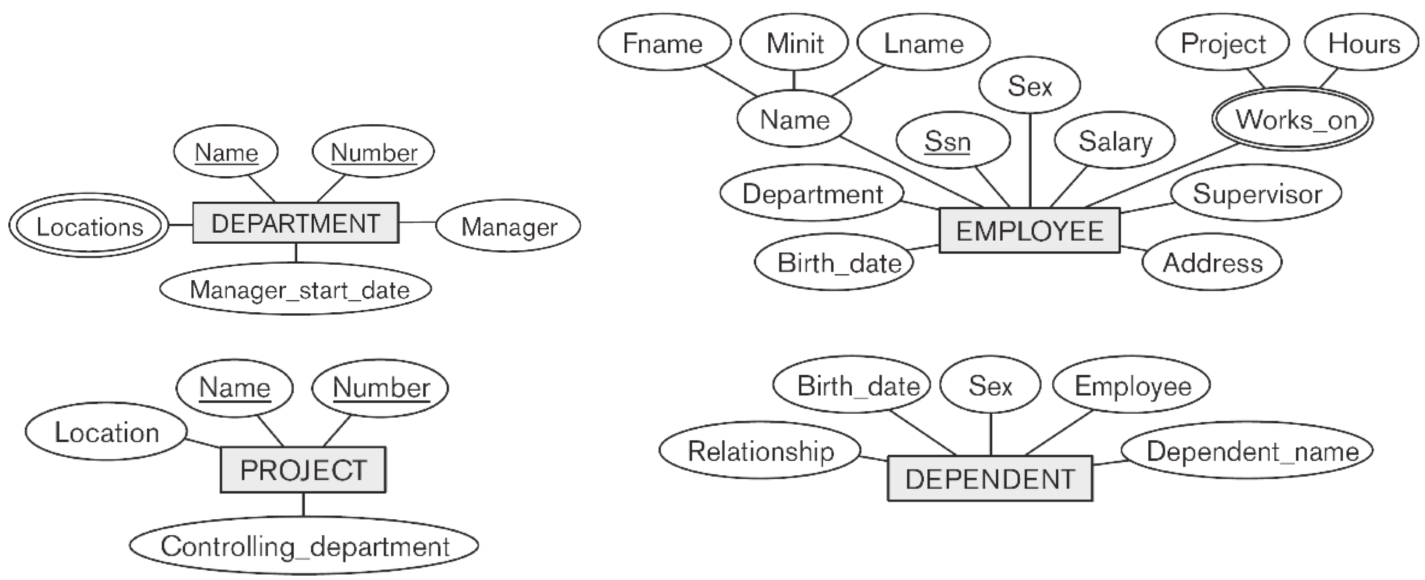
Cardinality Ratio 1: N for E_1:E_2 in R



Structural Constraint (min, max) on Participation of E in R

ER Modelling: From Implicit to Explicit Relationships

- Consider again the initial ERD (Version 0) for the COMPANY database we saw before
- It models relationships implicitly
- To make relationships explicit we:
  - Change attributes that represent relationships into relationship types
  - Determine the cardinality ratio and participation constraints of each relationship type



Annotating a Requirements Specification

bold denotes entity types

italic denotes attribute types

<u>underline</u> denotes relationship types

- Assume our miniworld encompasses the **employees**, **departments** and **projects** of a company.
- The company is organized into departments, which have a *name*, a *number* and a *location* (possibly more than one).
- Each department <u>is managed by a</u> **manager**, who is an employee, and we wish to store the *date* in which the latter started managing the department.
- Each department <u>controls various</u> projects.

Annotating a Requirements Specification

- An employee works for only one department but may work on more than one project (not necessarily controlled by the same department).
- We wish to store how many *hours* an employee works on each project and which employee supervises which employee.
- For each employee, we wish to store the employee's *name*, *social security number*, *address*, *salary*, *gender*, and *birth date*.
- We also wish to store information about the **dependents** of each employee, viz., their *name*, *gender*, *birth date*, and *relationship to the employee*.

ER Modelling: Design Choices

- Often, it is unclear, at first, whether it is best to model a concept as an entity type or an attribute.
- If so, model it first as an attribute.
- If, later, it cannot be construed as a property of any entity type, then consider recasting it as an entity type.

ER Modelling: Design Choices

- Always check that the uniqueness constraint comes naturally (or else, aim to be sure that the entity is weak).

- If you find that an attribute occurs in several entity types, then consider factoring it out as an independent entity type.

- Conversely, an entity type could be recast as an attribute if it only has a single attribute.

- If an attribute is found to be a reference to another entity type, recast it as a relationship.

ER Modelling: Guidelines for Naming/Diagramming

- Always choose a name/label (e.g., EMPLOYEE) that conveys the meaning of the concept (i.e., of a company employee) represented by the chosen construct (i.e., an entity type) in the model/diagram.

- In a text/narrative that states the requirements, look for:

  - **Nouns**: they normally suggest names of either an entity type (e.g., DEPARTMENT) or an attribute (e.g., Location)

  - **Verbs**: they normally suggest names of relationship types (e.g., WORKS_FOR, WORKS_ON, MANAGES).

ER Modelling: Guidelines for Naming/Diagramming

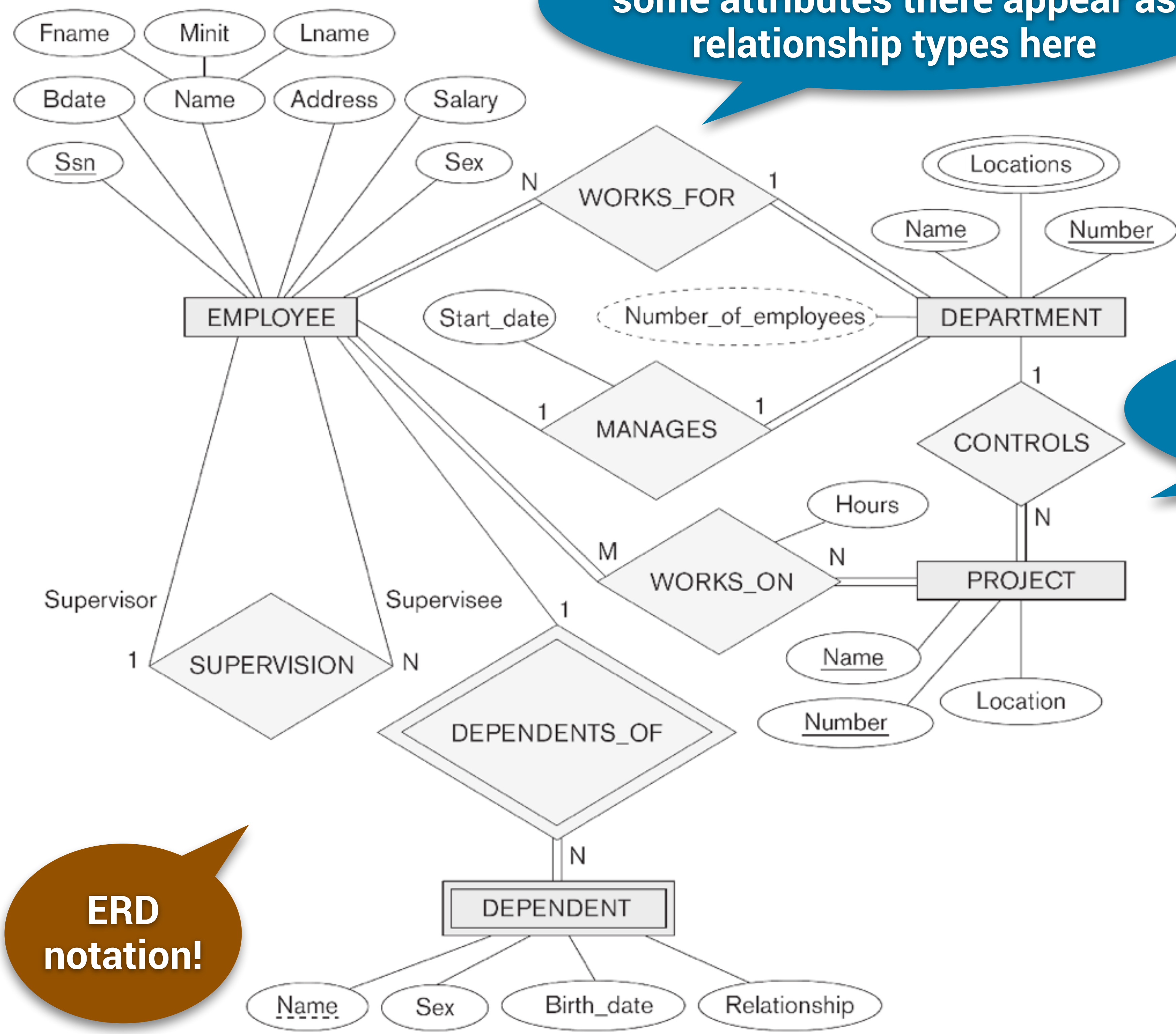
- Choose binary relationship names under the assumption that the ER diagram is read from left to right and from top to bottom (if possible).

- Although not impossible, it is improbable that there will be islands, i.e., parts that are not connected to others.

- In practice, it is imperative to clarify doubts by going back to the users: never assume you know better.

Conceptual Design: COMPANY Database Version 1

compared to V0, some attributes there appear as relationship types here



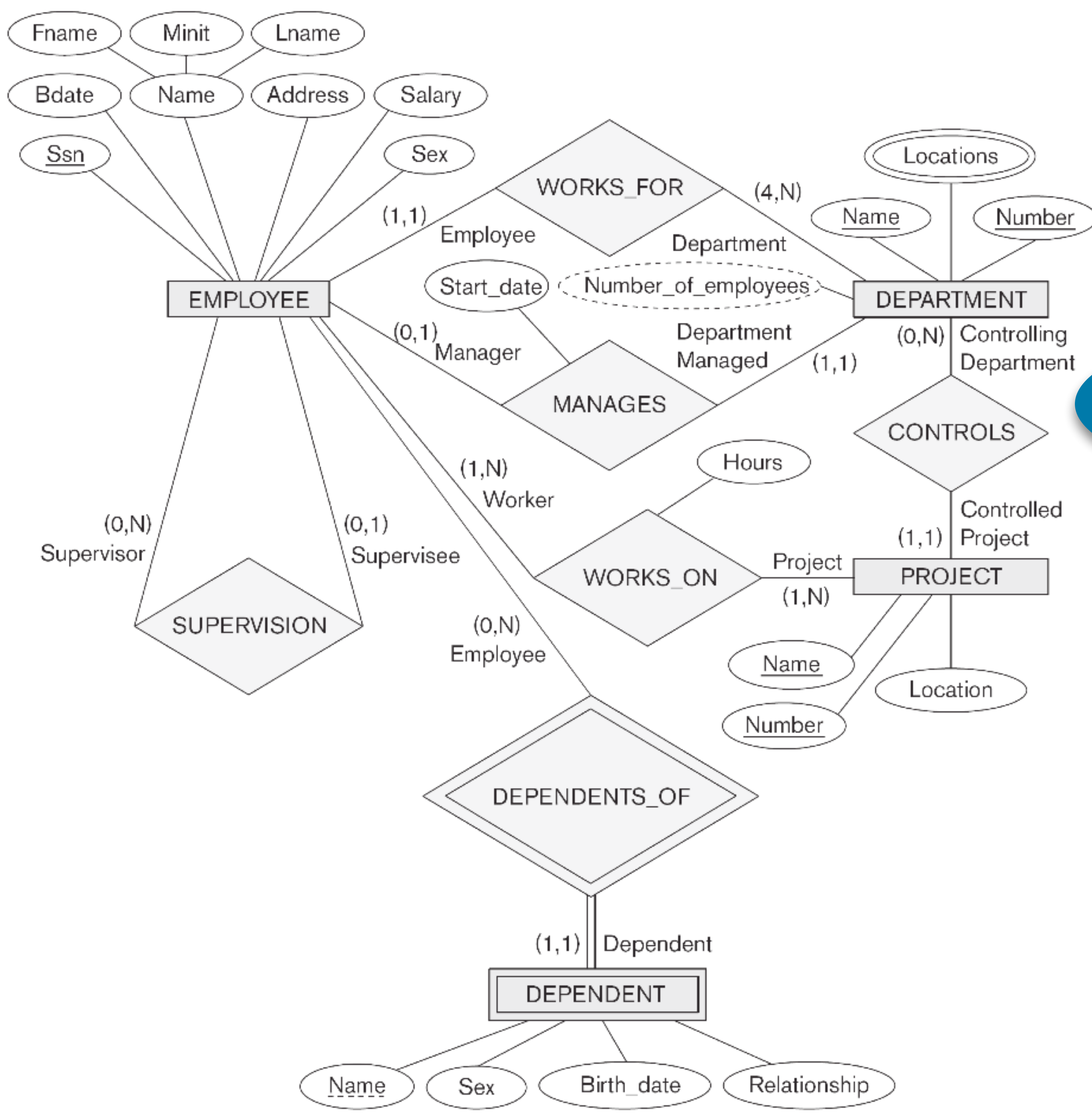
cardinality ratio and participation constraints made explicit

ERD notation!

ER Modelling: Alternative Notation for Constraints

- There exists an alternative notation for specifying structural constraints more precisely
- It replaces the (1:1, 1:N, N:1, N:M) cardinality ratio notation and the single/double line notation for participation constraints
- Instead, it associates a pair of integer numbers (min, max) with each participation of an entity type *E* in a relationship type *R*, where $0 \leq min \leq max \geq 1$

Conceptual Design: COMPANY Database Version 1'



alternative ERD notation!

structural constraints expressed as (min,max) pairs

ER Modelling: Higher-Degree Relationships

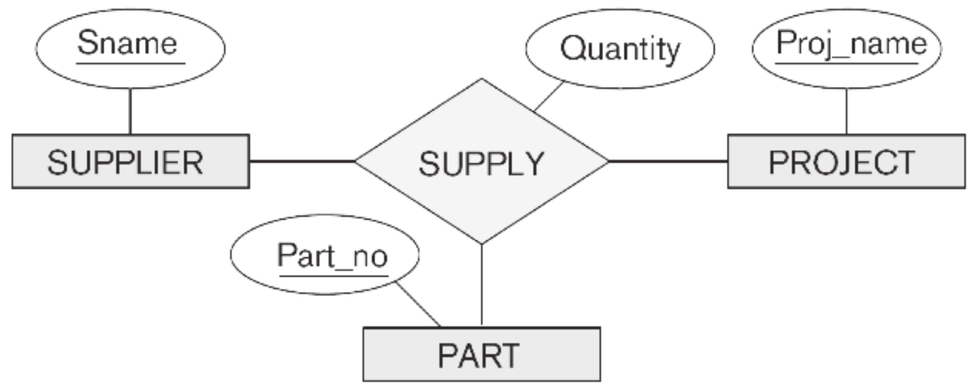
- Some database design tools only cater for binary relationships.
- Note that three binary relationships may fail to capture the semantics of a single ternary relationship.

ER Modelling: Higher-Degree Relationships

- Possible ways around such tool limitations are:
  - Model the ternary relationship as a weak entity type, without a partial key and with three identifying relationships.
  - Model the ternary relationship as a regular entity type introducing an artificial or surrogate key for that purpose.

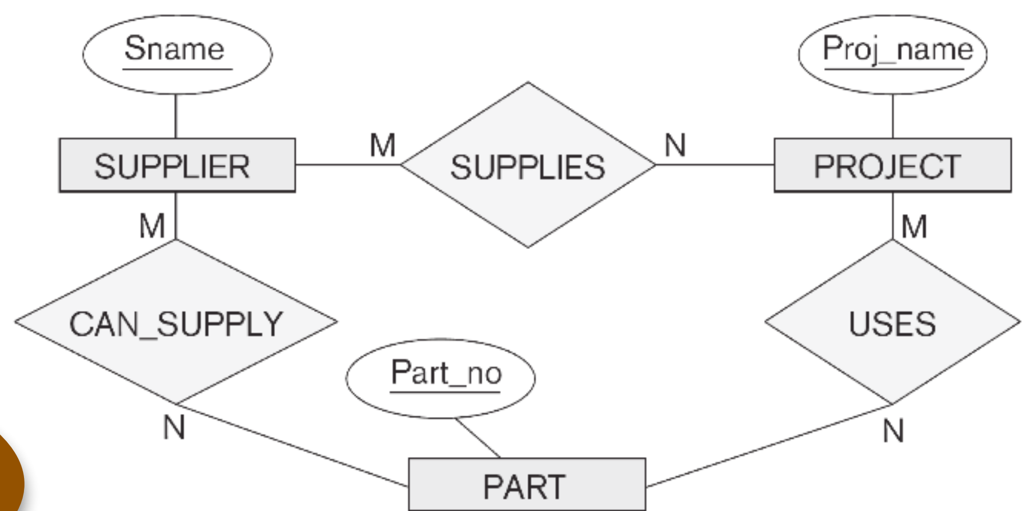
Example: Binary v. Ternary Relationships

(a)



3-ary: best solution

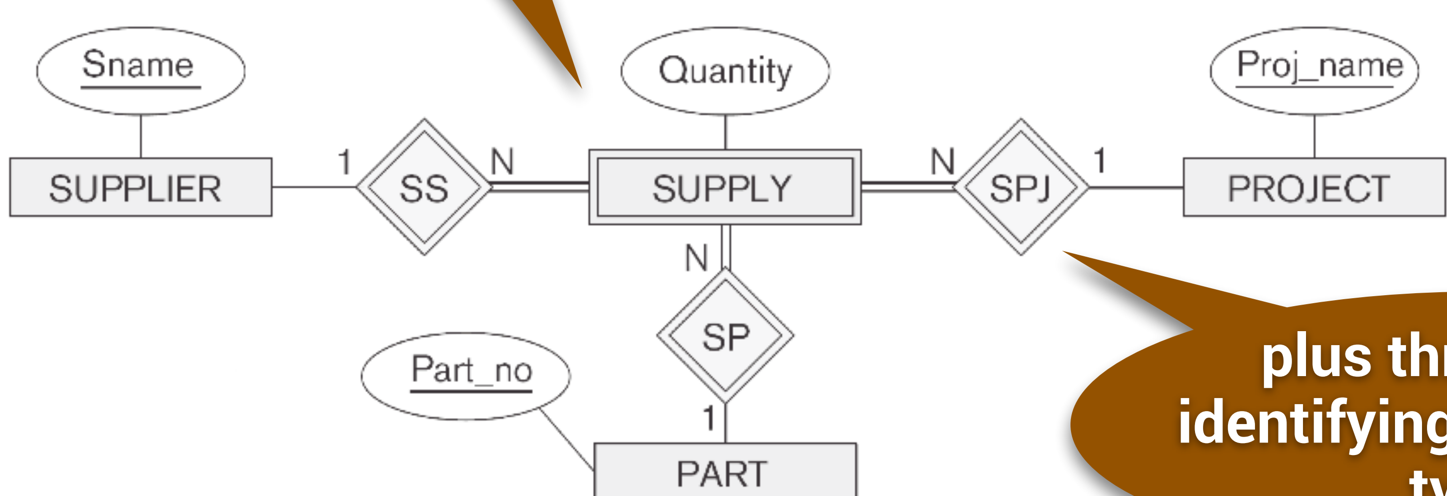
(b)



not equivalent to 3-ary SUPPLY

relationship-as-weak-entity

(c)



plus three binary identifying relationship types

Summary

The ER Approach to Database Design

- Database applications comprise a database and the queries and updates used to retrieve and manipulate the data held in it.

- Conceptual modelling is the use of high-level abstractions for capturing data requirements in a way that facilitates communication between designers and users.

- The design of database applications starts with requirements gathering, then proceeds by stepwise refinement from conceptual, through logical, to physical design.

- It also includes the design of queries and transactions that occur in the functional requirements.

The ER Approach to Database Design

- Entity-relationship (ER) modelling is a popular approach to the conceptual design phase of database applications.
- ER modelling involves identifying entity types and the attributes that describe them, as well as relationship types, at schema level.
- Entity sets, value sets and relationship sets are the corresponding notions at instance level.
- ER models are expressed as diagrams (ERDs).
- ER models can make use of simple and composite, single- and multivalued, stored and derived attributes.

The ER Approach to Database Design

- ER models use uniqueness, cardinality ratio and participation constraints, as well as weak entity types, to capture expressive data requirements.
- ER modelling uses some informal and intuitive guidelines and proceeds by refinement.
- Ultimately, a conceptual model must be capable of being mapped into a target implementation model, referred to as a logical model (e.g., relational).
- Further down, the logical model is associated with a physical model, where more attention is given to performance requirements.

In The Next Handout

- We'll continue our exploration of conceptual modelling.
- We'll see that how the ER conceptual model can be enhanced to capture more application semantics.