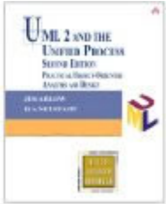
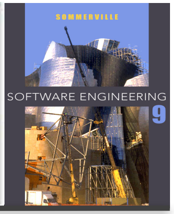




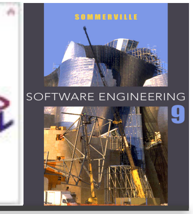
Lecture 5

STRUCTURED ANALYSIS

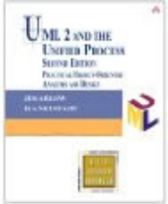
PB007 Software Engineering I
Faculty of Informatics, Masaryk University

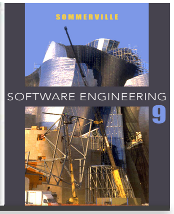
Fall 2016

Outline



- ✧ Yourdon Modern Structured Analysis (YMSA)
  - Context diagram (CD)
  - Data flow diagram (DFD)
- ✧ Data modelling
  - Entity relationship diagram (ERD)
- ✧ Relational database design
  - Normalization

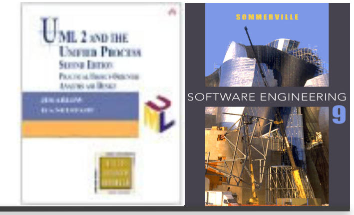




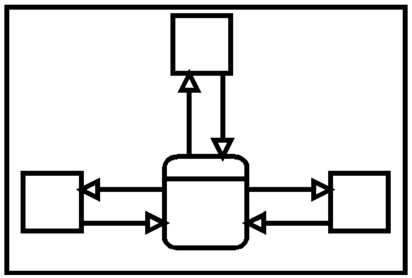
Yourdon Modern Structured Analysis (YMSA)

Lecture 5/Part 1

E. Yourdon: Modern structured analysis



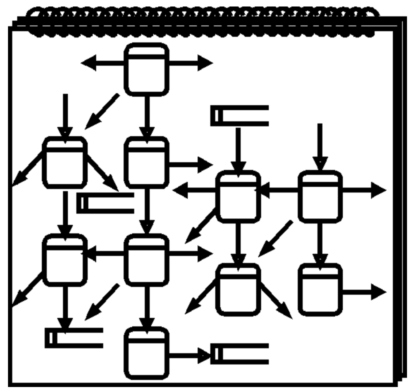
Environment model



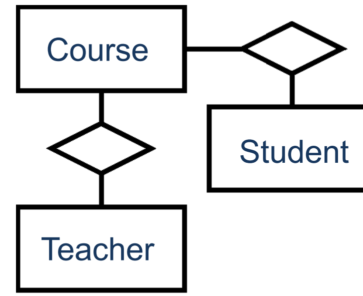
Events:

E1: registered
E2: rolled in
E3: rolled out
E4: started
E5: ended

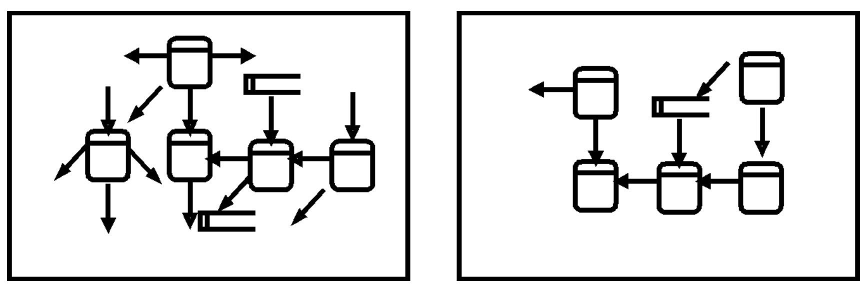
Behavioral model



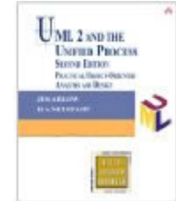
Data model



Functional decomposition

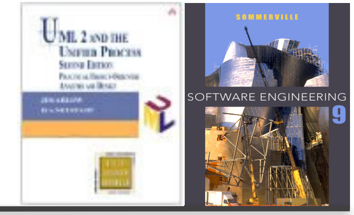


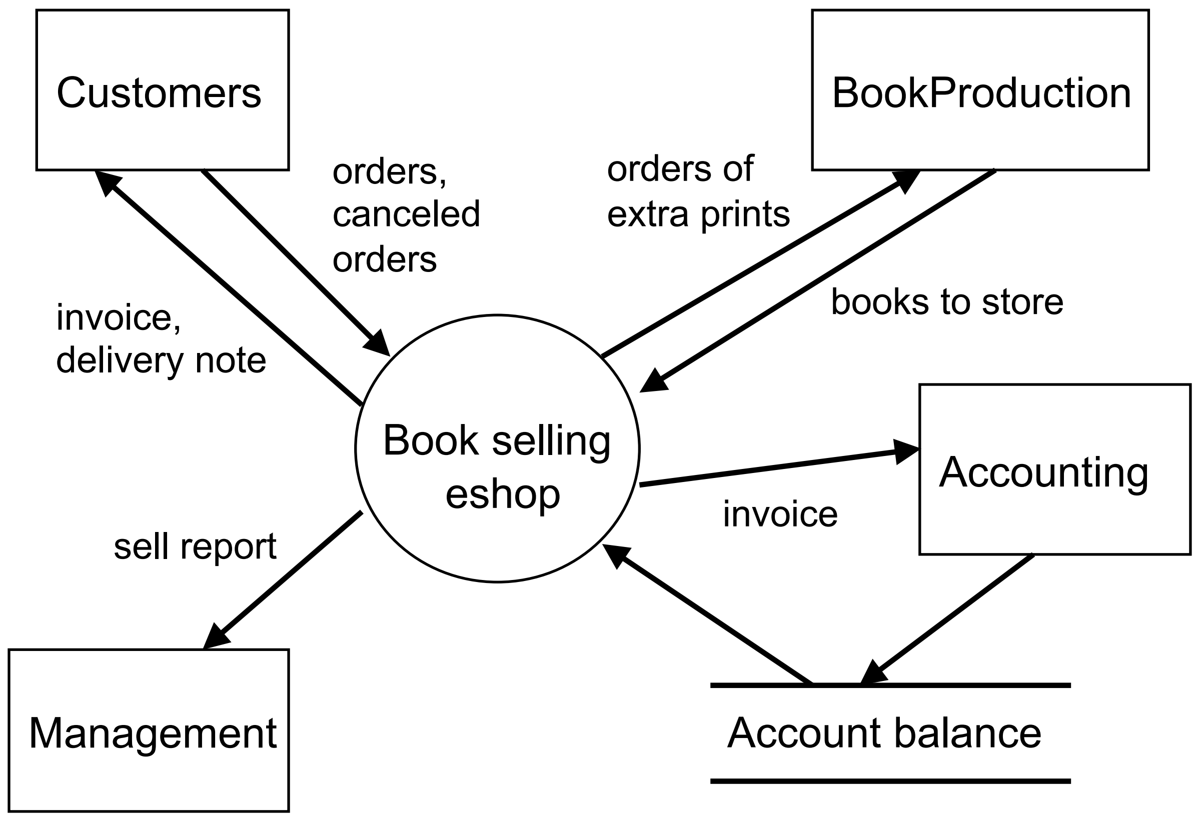
Environment model



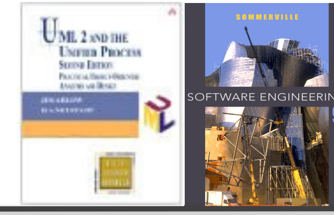
- **Context diagram** is a special case of a data flow diagram, containing a single process representing the whole system. It emphasizes:
  - **Terminators** – people and systems communicating with the system
  - **Data received** from the environment that shall be processed
  - **Data produced** by the system and sent to the environment
  - **Data stores** shared by the system and its terminators
  - **System boundary**
- **Event list** is a textual list of stimuli coming from the environment that must be responded by the system.

Context diagram example



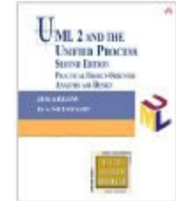


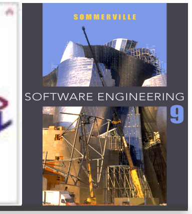
Behavioral model



- Behavioral model specifies the flow of data through the modeled information system, modeling its process aspects.
  - It shows what kinds of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored.
  - It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel.
- **Data flow diagram (DFD)** is a graphical representation of the system as a network of processes that fulfill system functions and communicate through system data.

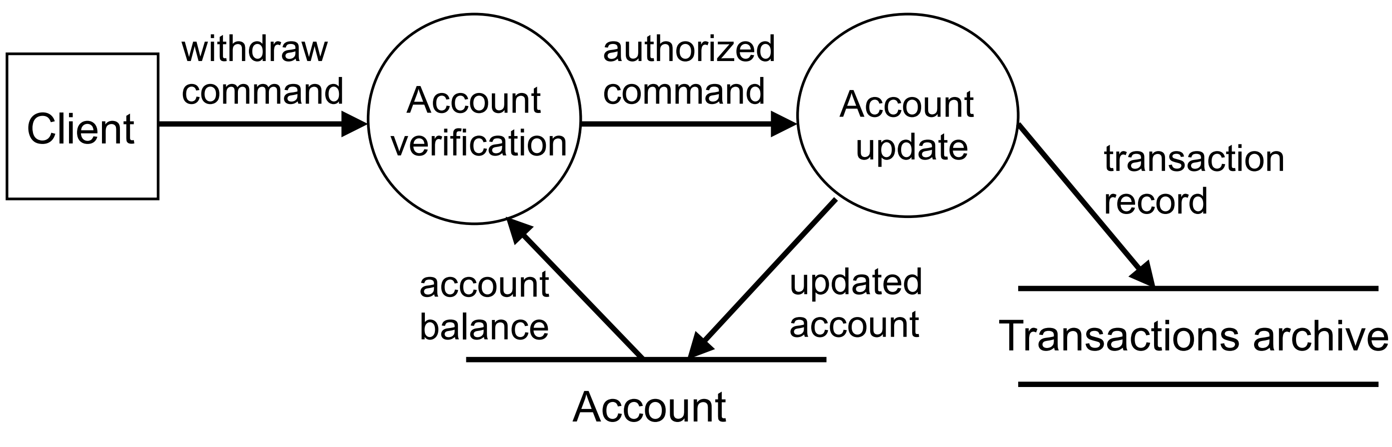
Data flow diagram (DFD)



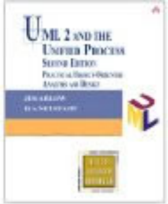


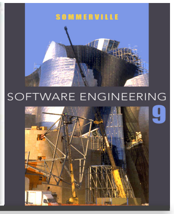
✧ DFD consists of four types of elements:

- Processes
- Data flows
- Data stores
- Terminators



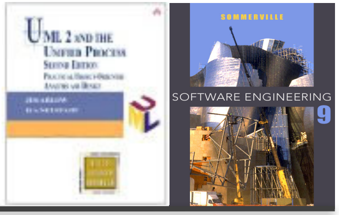
Processes and Data flows



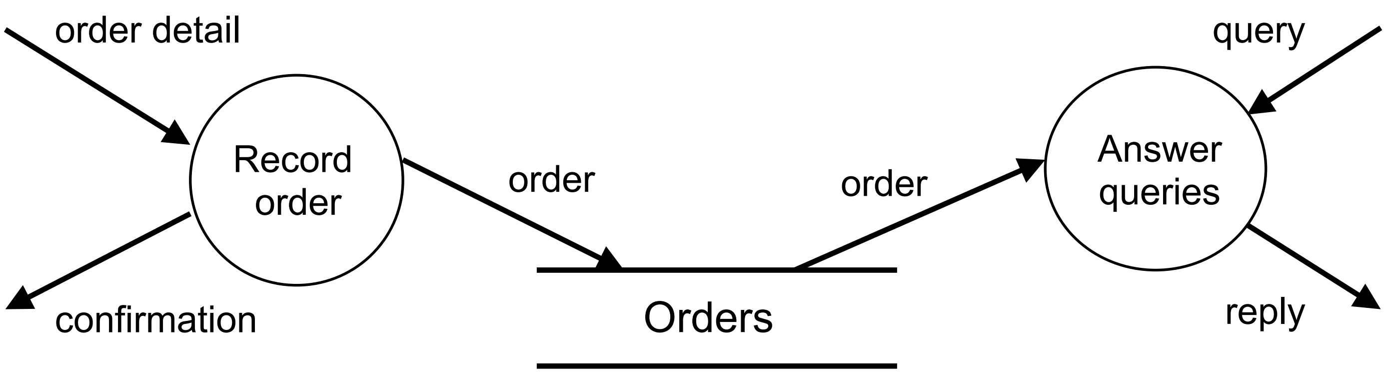


- ✧ A **Process** models a part of the system that transforms specific inputs to outputs.
- ✧ Name of a process is a single word, phrase or simple sentence, e.g. “User authentication”.
  - The process name sometimes contains the name of a person, group of people, department or device – specifying also the actor or tool of the process.
- ✧ A **Data flow** models a way for data transfer from one part of the system to another.
  - Flows can also model the transfer of physical materials.

Data stores

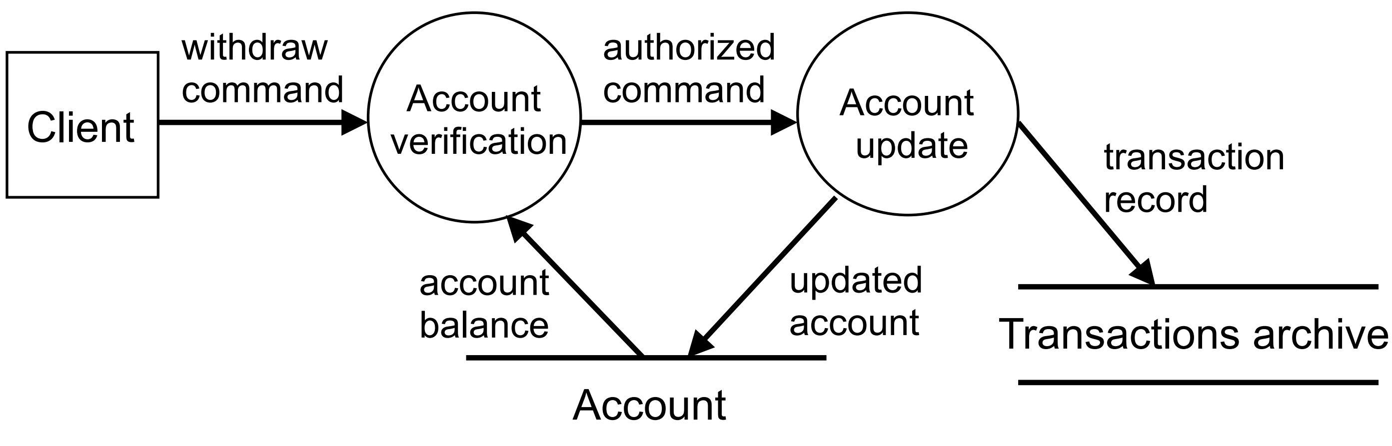


- **Data store** models a static collection of data that are shared by two or more processes operating in different time.
  - Name is a plural of the data name going to and coming from the data store.

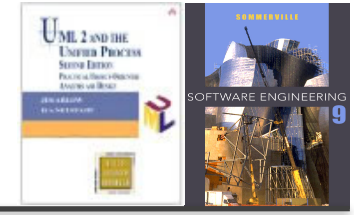


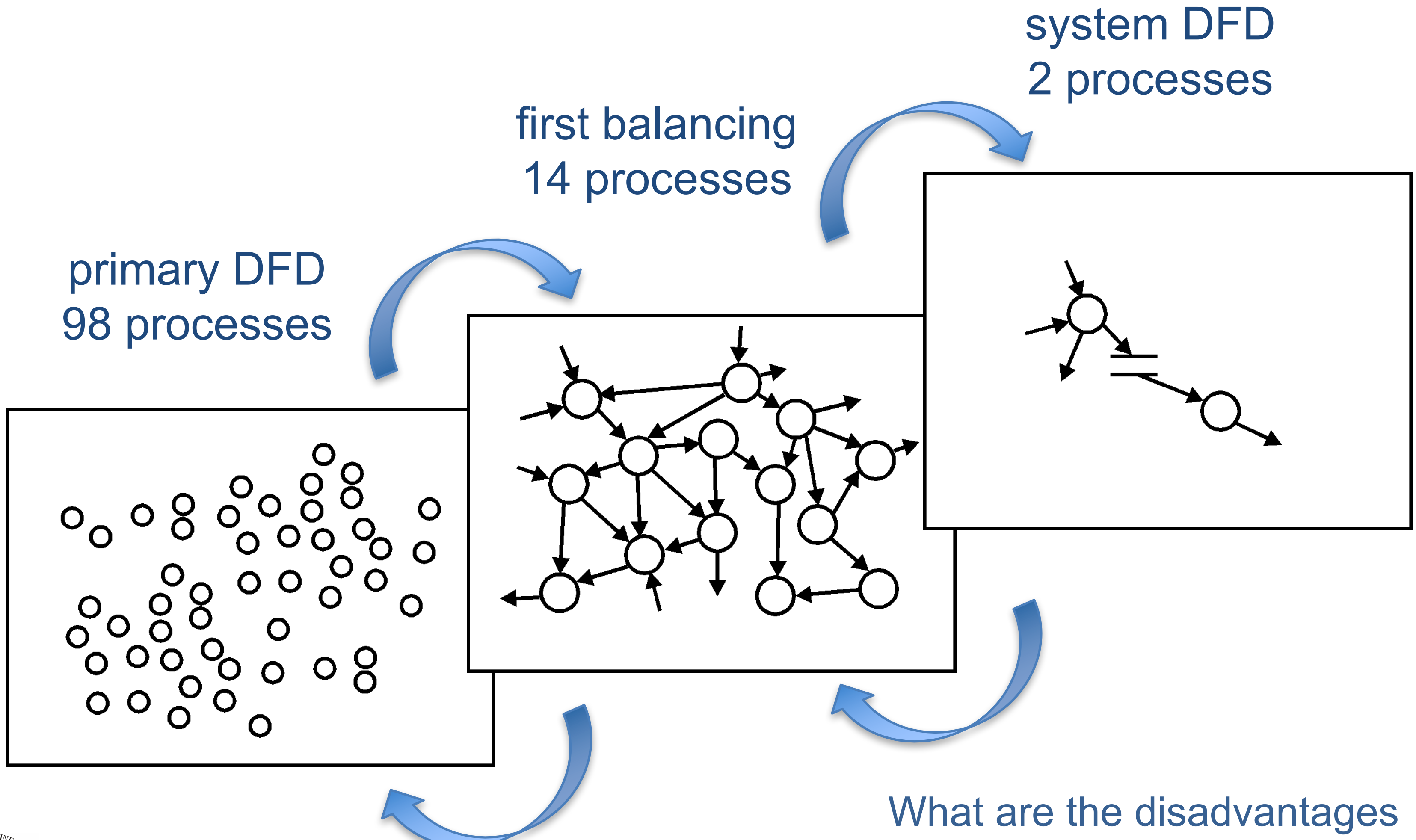
Terminators

- A **Terminator** represents an external entity communicating with the system.
- The flows connecting terminators with the processes or data stores inside the system represent the interfaces between the system and its environment.

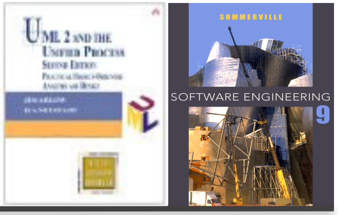


Top-down and bottom-up DFD balancing





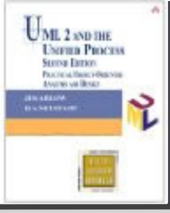
What are the disadvantages of the two approaches?

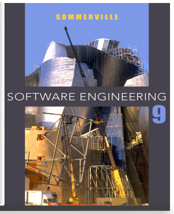


Data modelling

Lecture 5/Part 2

Data modeling

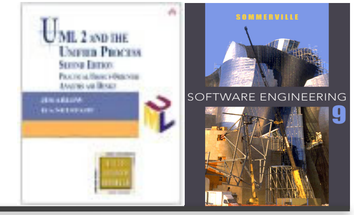




- Defines static data structure, relationships and attributes
- Complementary to the behavior model in structured analysis; models information not covered by DFDs
- More stable and essential information comparing to DFD

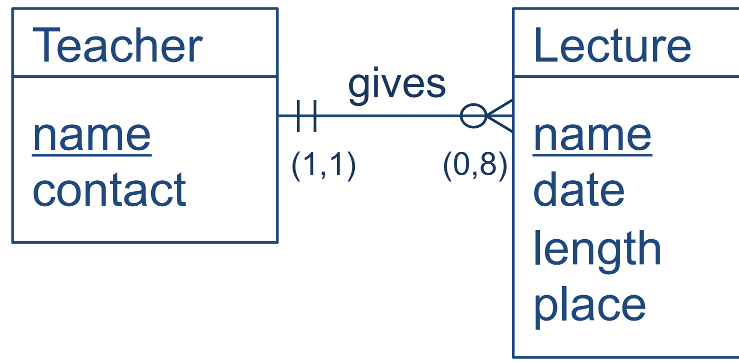
- **Entity-Relationship modeling**
  - Identify system entities – both abstract (lecture) and concrete (student)
  - For each entity examine – the purpose of the entity, its constituents (attributes) and relationships among entities
  - Check model consistency and include data details

Entity Relationship Diagram (ERD)

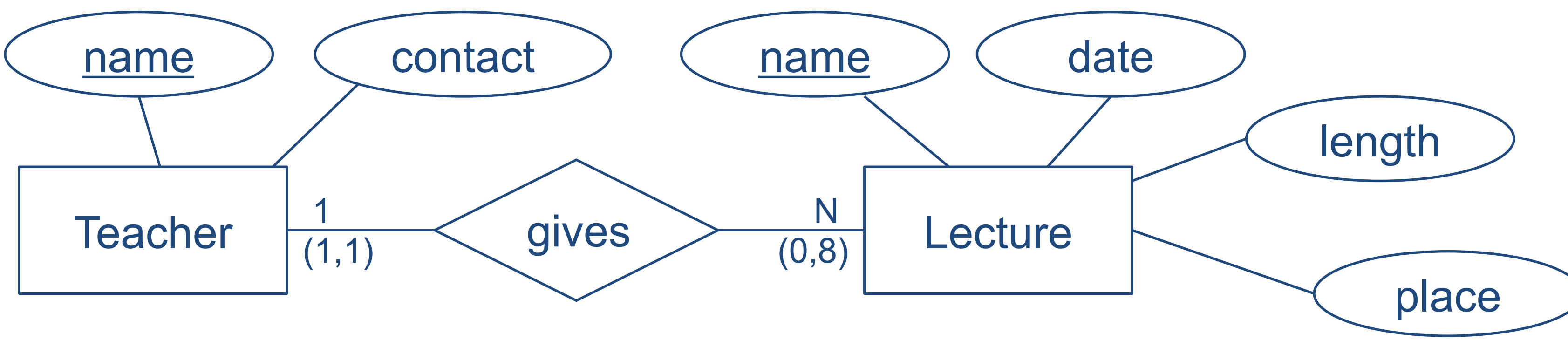


- **Entities** and their types
- **Relationships** and their types
- **Attributes** and their domains

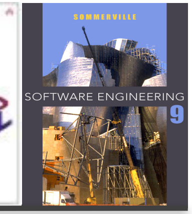
Crow's Foot notation
(implementation level descript.)



Chen's notation
(concept level description)



Entities and Entity types



✧ An **Entity** is anything about which we want to store data

- Identifiable – entities can be distinguished by their identity
- Needed – has significant role in the designed system
- Described by attributes shared by all entities of the same type

✧ An **Entity set** is a set of entities of the same **Entity type**.

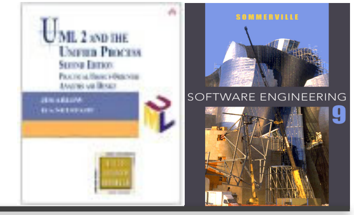
Entity	Entity type
You	Student
Your neighbor	Student
Me	Teacher
This PB007 lecture	Lecture

Student

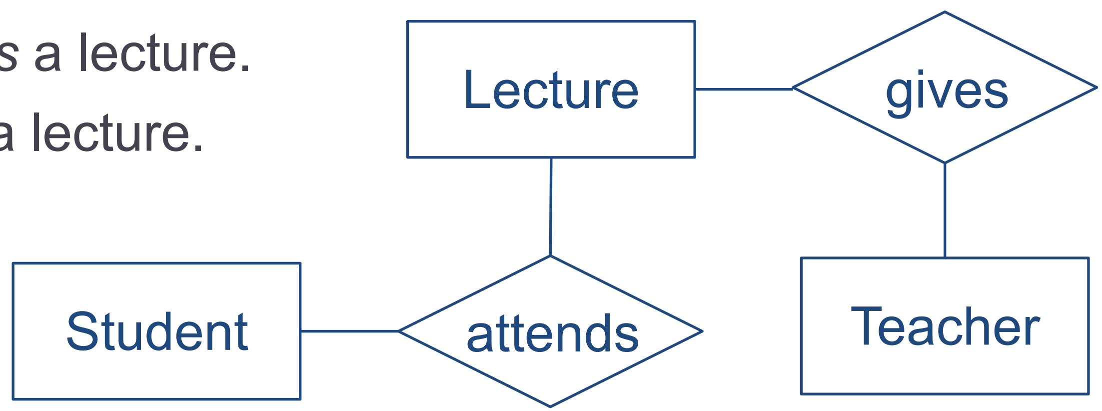
Teacher

Lecture

Relationships and Relationship types

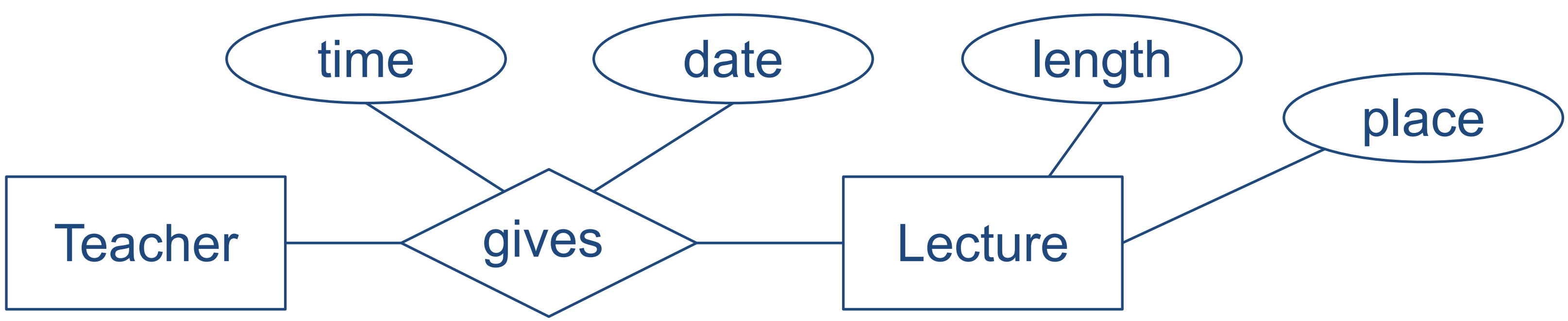


- ✧ Entities take part in **Relationships** (among possibly more than two entities), that can often be identified from verbs or verb phrases.
  - You are *attending* this PB007 lecture.
  - I am *giving* this PB007 lecture.
- ✧ A **Relationship set** is a set of relationships of the same **Relationship type**.
  - A student *attends* a lecture.
  - A teacher *gives* a lecture.

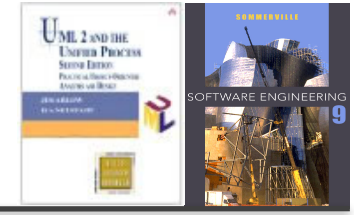


Attributes and Attribute domains

- An **Attribute** is a fact, aspect, property, or detail about either an entity type or a relationship type.
  - E.g. a lecture might have attributes: time, date, length, place.
- An **Attribute type** is a type domain of the attribute. If the domain is complex (domain of an attribute *address*), the attribute may be an entity type instead.

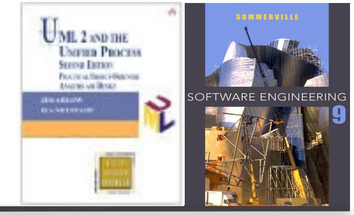


Attributes or entities?



- To decide whether a concept be modeled as an attribute or an entity type:
  - Do we wish to store any information about this concept (other than an identifying name)?
  - Is it single-valued?
  - E.g. *objectives* of a *course* – are they more than one? If just one, how complex information do we want to store about it?
- General guidelines:
  - Entities can have attributes but attributes have no smaller parts.
  - Entities can have relationships between them, but an attribute belongs to a single entity.

Relationship-type degree

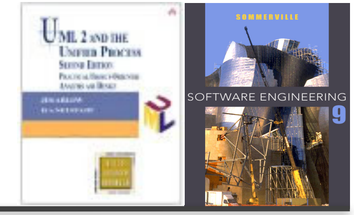


Every manager leads exactly one department.
Every department is led by exactly one manager.

Every edition plan contains one or more book titles.
Every book title is part of exactly one edition plan.

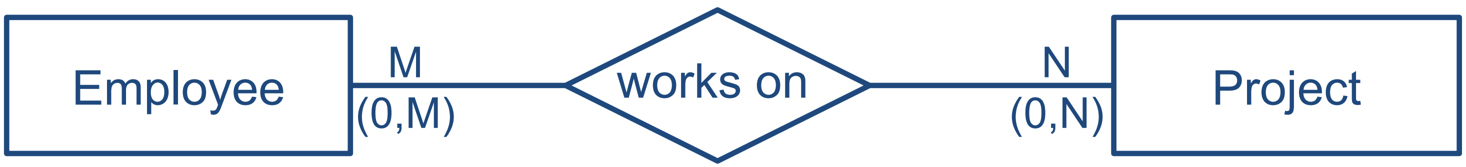
Every producer produces one or more products.
Every product is produced by one or more producers.

Relationship-type degree

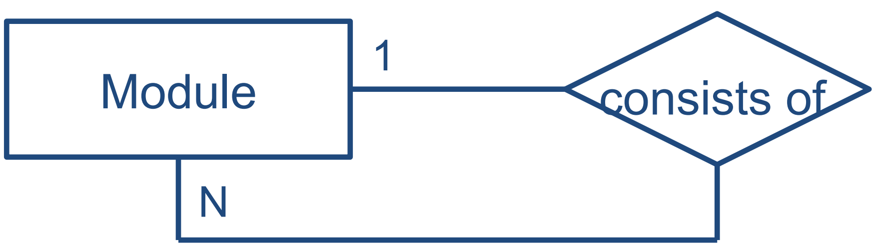


Mandatory relationship

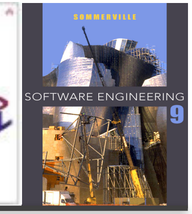
Optional relationship



Recursive relationship

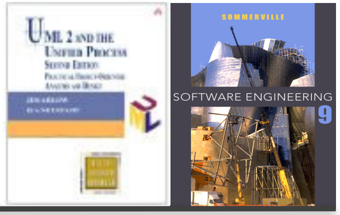


Cardinality ratio

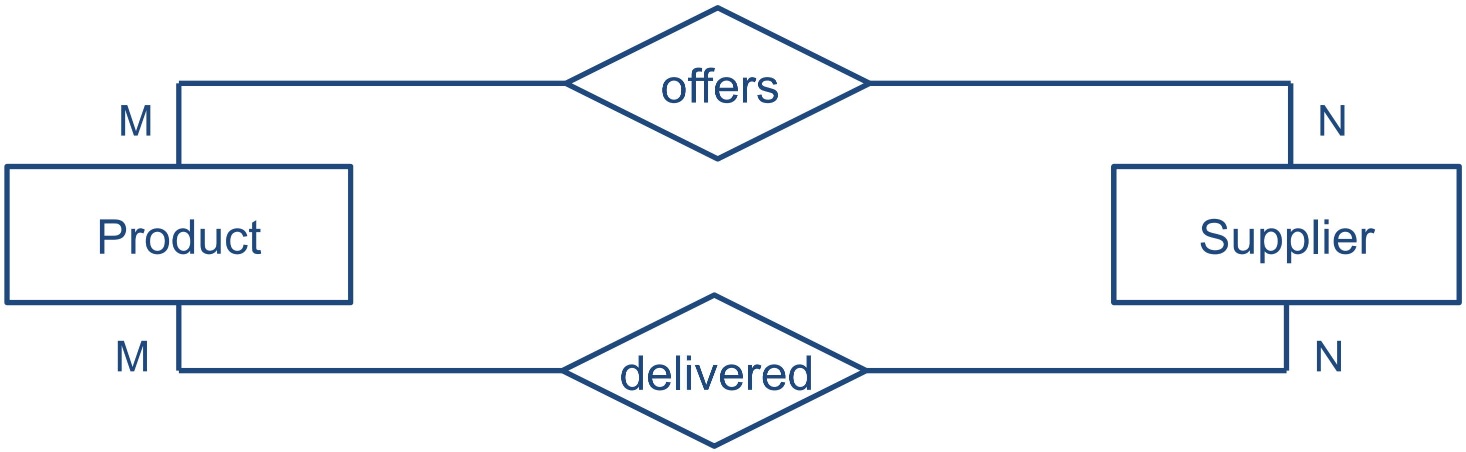



- **Cardinality ratio** of a relationship type describes the number of entities that can participate in the relationship.

- One to one 1:1
  - Each lecturer has a unique office.
- One to many 1:N
  - A lecturer may tutor many students, but each student has just one tutor.
- Many to many M:N
  - Each student takes several modules, and each module is taken by several students.

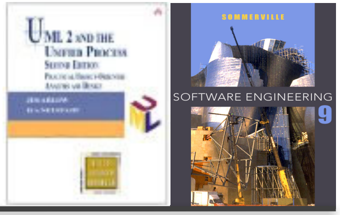


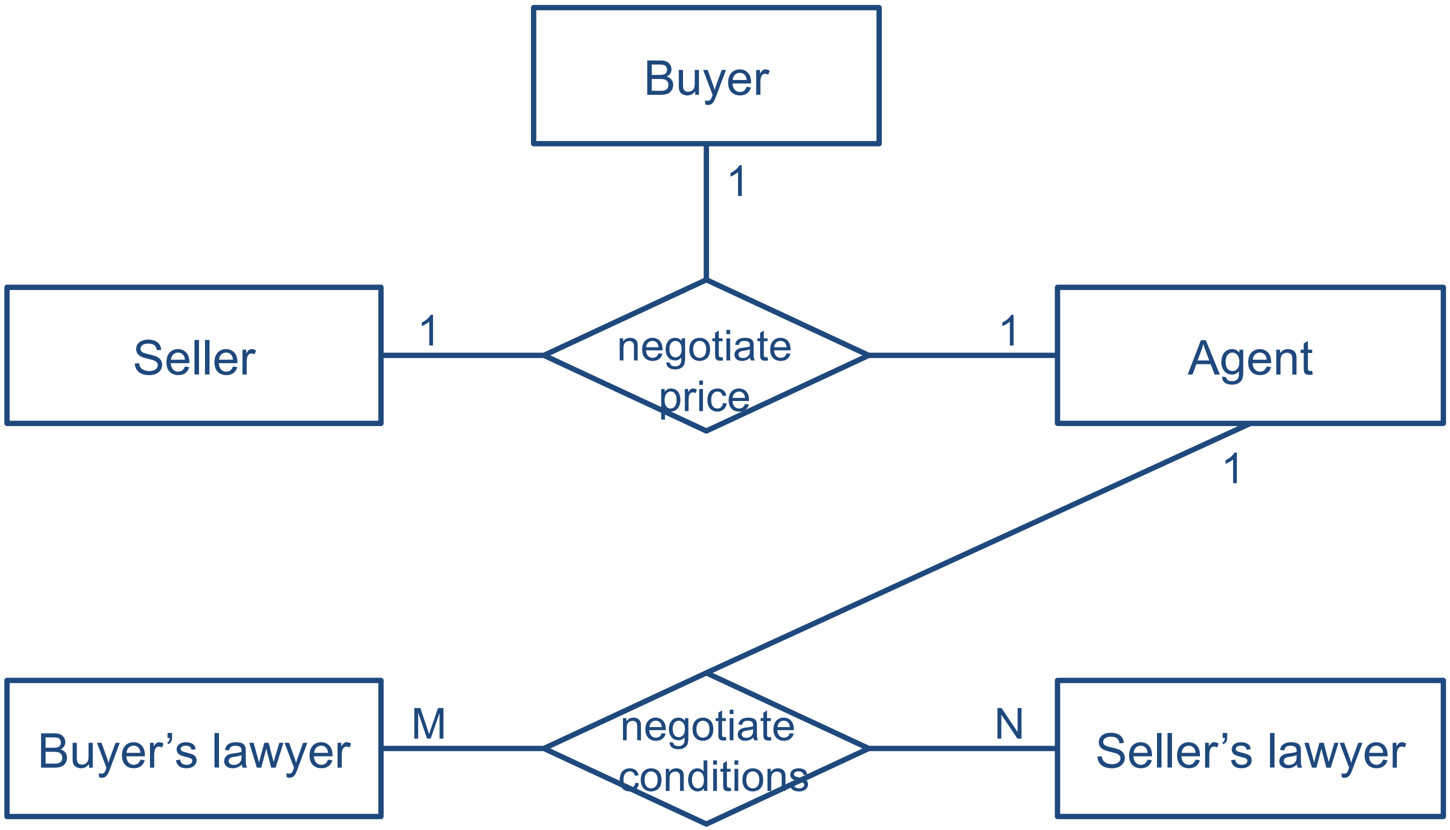
More relationships between two entities



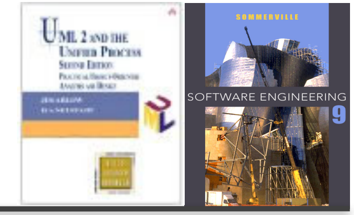
- Relationship *offers* has attributes:
  - *payment conditions, due date*.
- Relationship *delivered* has attributes:
  - *delivery note details*.

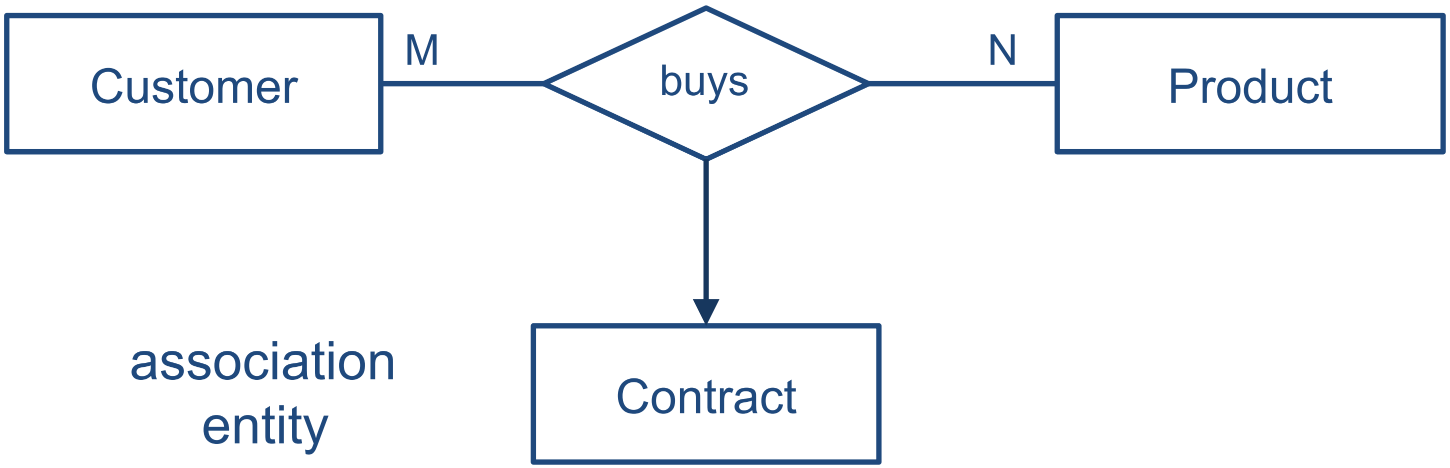
Relationships among more than two entities





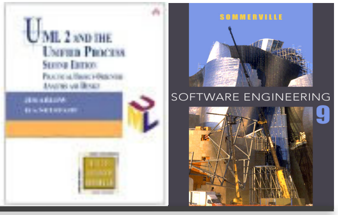
Association entity

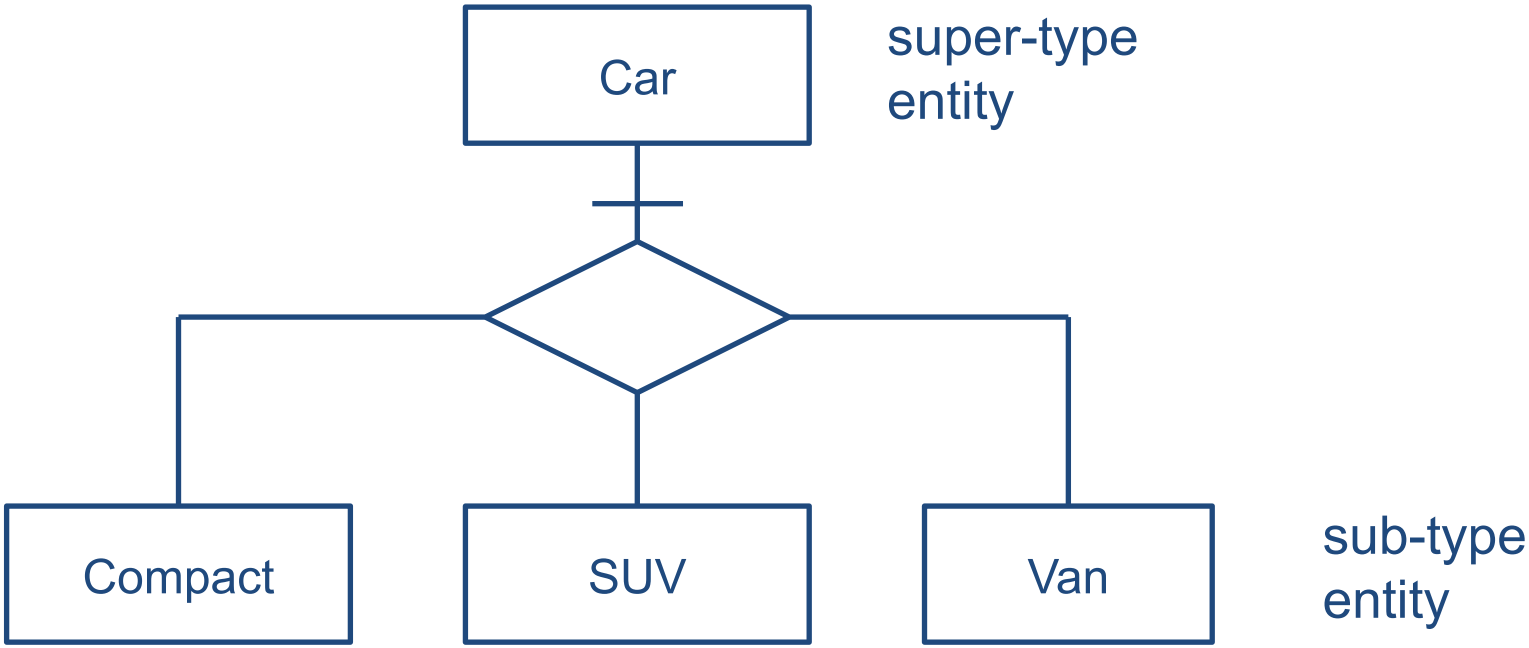




✧ The *Contract* exists just as a result of the relationship between the *Customer* and *Product* entity.

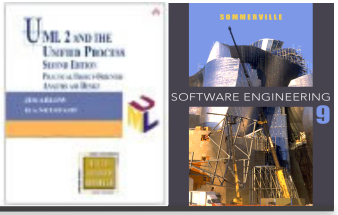
Super-type and sub-type entities





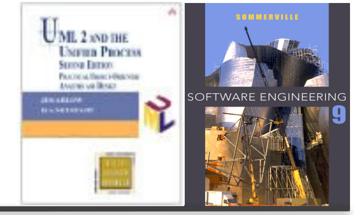
- Extended ERDs model also inheritance, i.e. the relationship of specialization–generalization

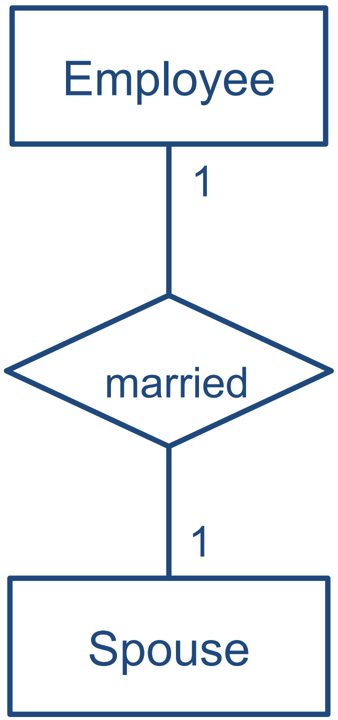
ERD modeling in structured analysis



- Iterative development in structured analysis
  - Entities identification -> initial ERD
  - Attributes identification -> detailed ERD
  - Identification of missing and redundant entities
    - Entities constituting of only one attribute (identifier)
    - Entity sets consisting of a single entity
    - Derived entities and relationships
    - Association entities
    - ERD-DFD consistency and completeness checking
- Modeled in parallel with DFD

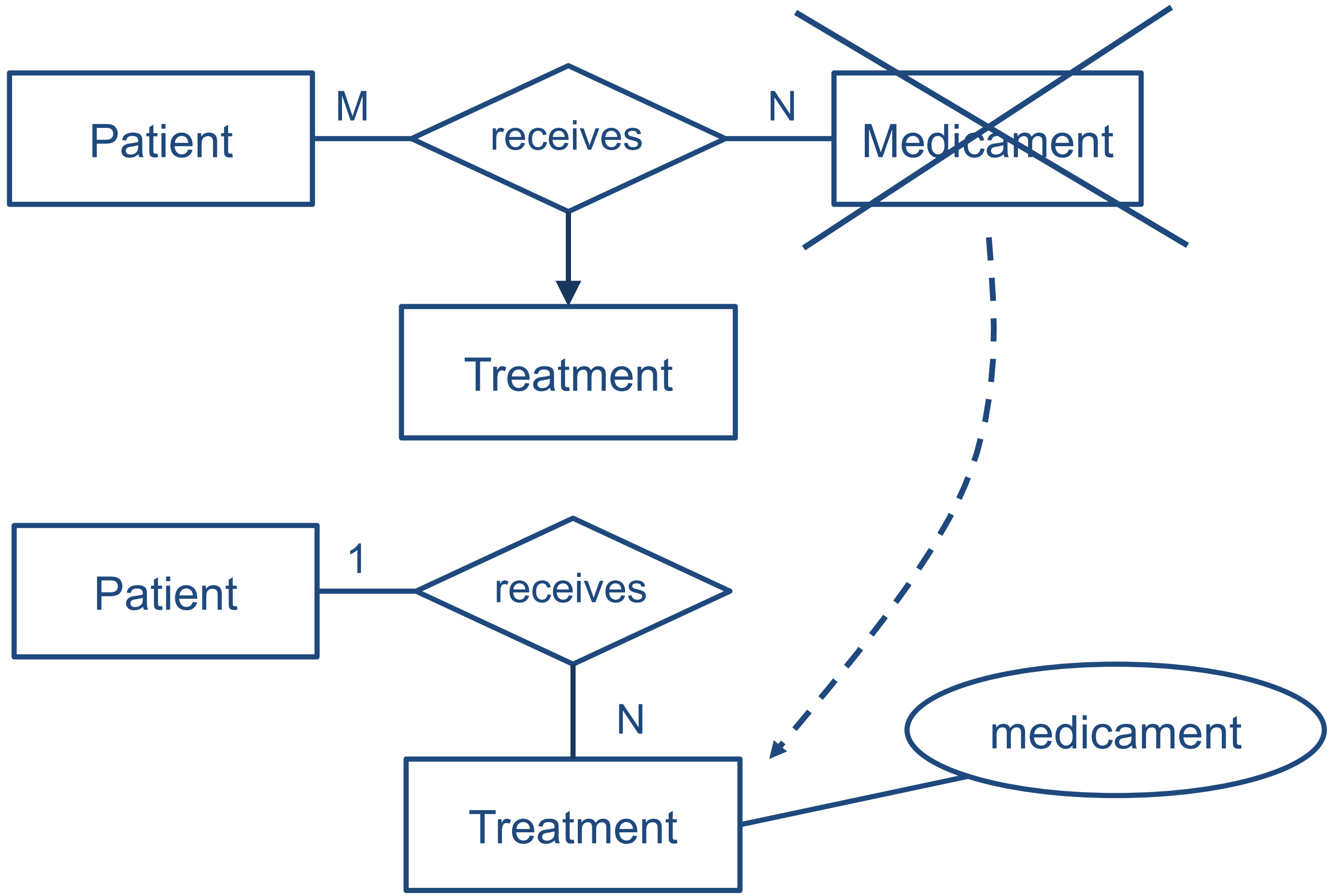
Removal of unneeded (redundant) entities



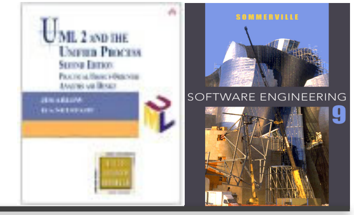


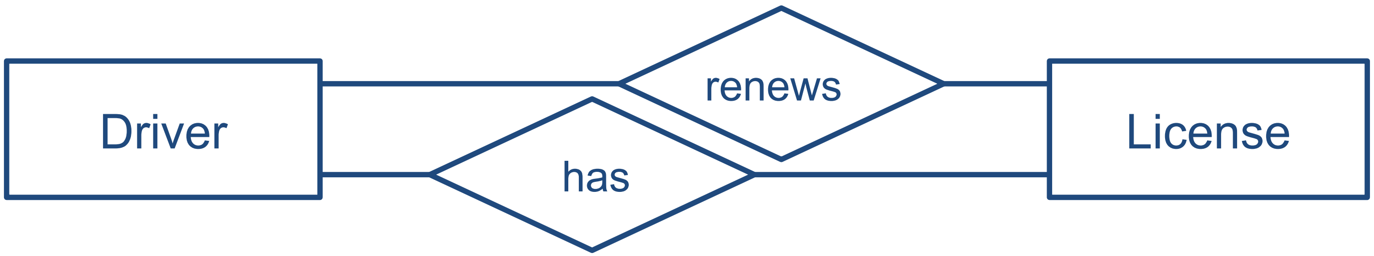
The *Spouse* entity is better suited as *Employee*’s attribute.

Removal of unneeded (redundant) entities

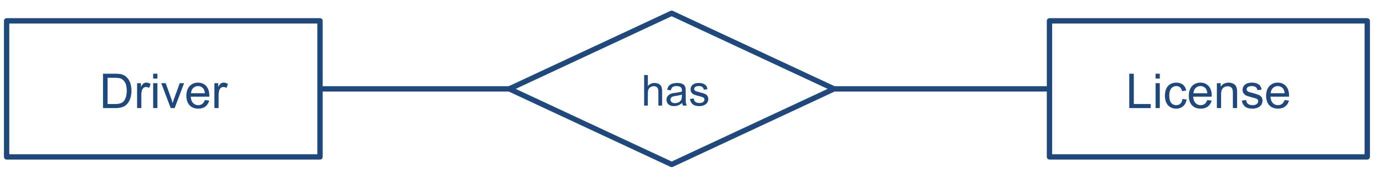


Removal of unneeded relationships

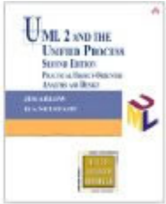


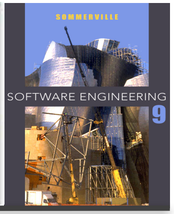


The duty to *renew* the license can be derived from the entities



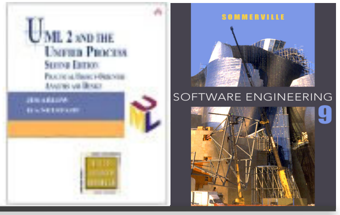
Data dictionary





- Used for documentation of complex ERD models
- Symbols:
  - = consists of
  - + and
  - ( ) optional part (0 or 1)
  - [|] alternative choice
  - { } iteration (1 or more) $a={}_1\{b\}_{15}$
  - * * comment
  - @ identifier (key)

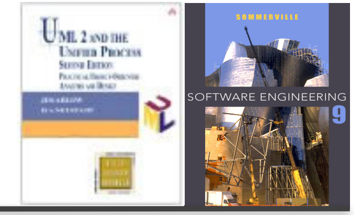
Example – Order



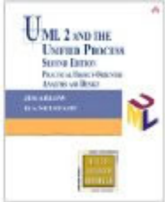
- **Order** **no. 2012-007-24**
- **Issue date:** 23.4.2012
  Delivery date: 30.4.2012
- **Customer:** no. 007
  Dr. John Smith
- **Goods:**

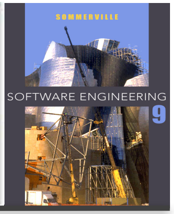
Number	**Name**	**Pieces**	**Price/piece**
P3876	Software engineering	6	135
H4681	UML2 and the UP	4	52
X6574	SA in practice	3	50

Example – Order



- ORDER = @number + issue date + (delivery date) +
  customer number + customer name +
  { product number + product name +
  ordered pieces + (product price per piece) }
- customer name = (title) + first name + surname
- title = [Mr. | Mrs. | Miss. | Dr. | Prof.]
- first name = { allowed symbol }
- allowed symbol = [A - Z | a - z |]

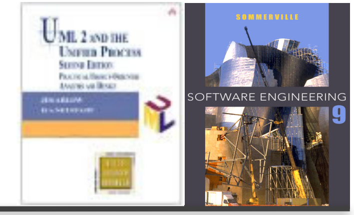


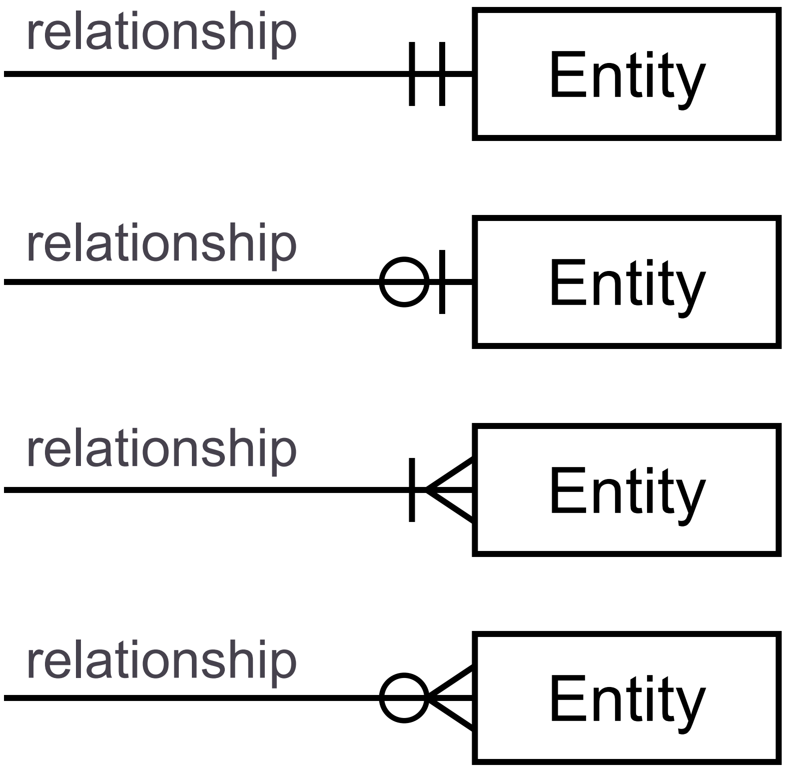


Relational Database Design

Lecture 5/Part 3

Crow's Foot notation





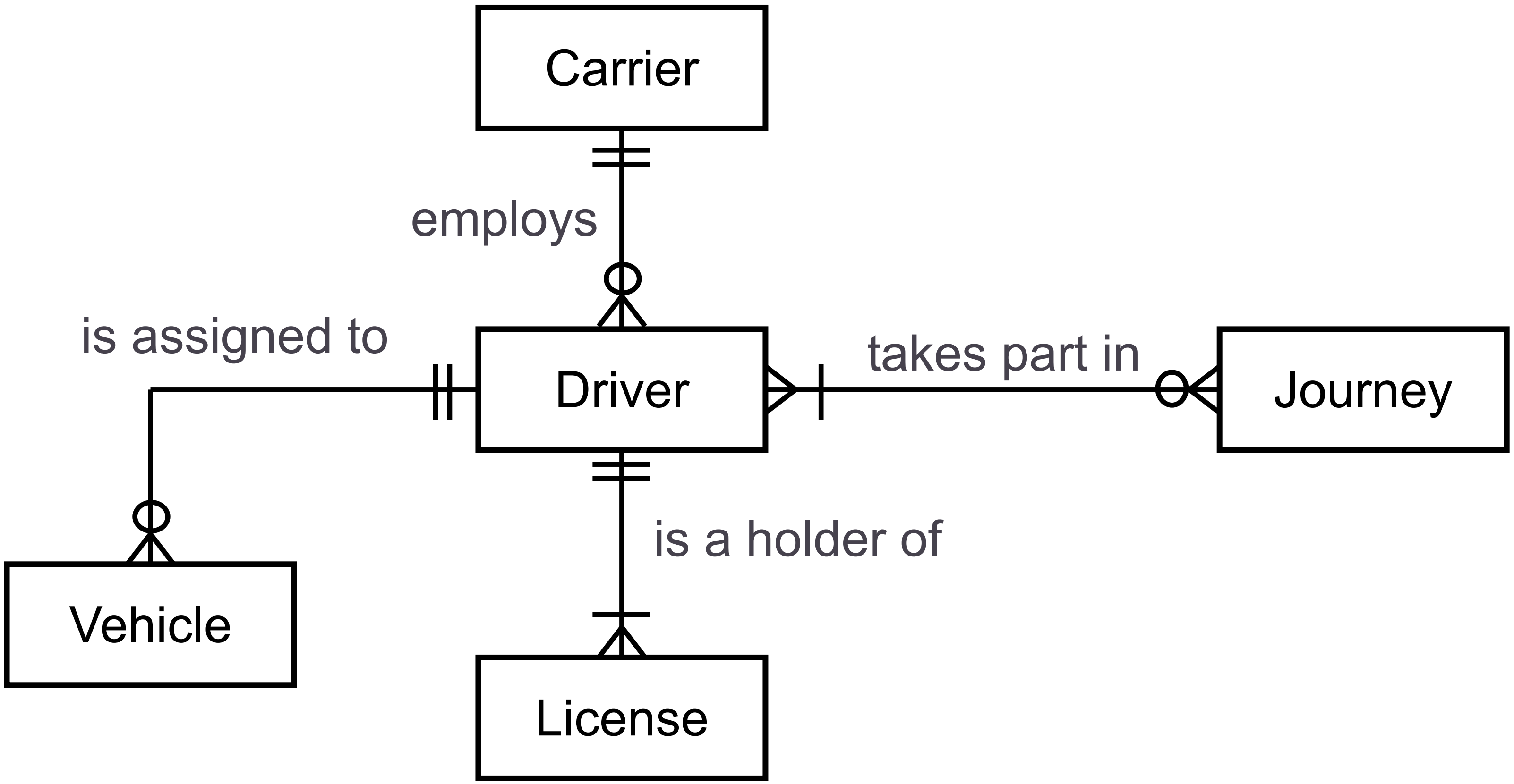
Exactly one occurrence

None or one occurrence

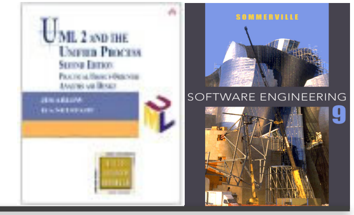
One or more occurrence

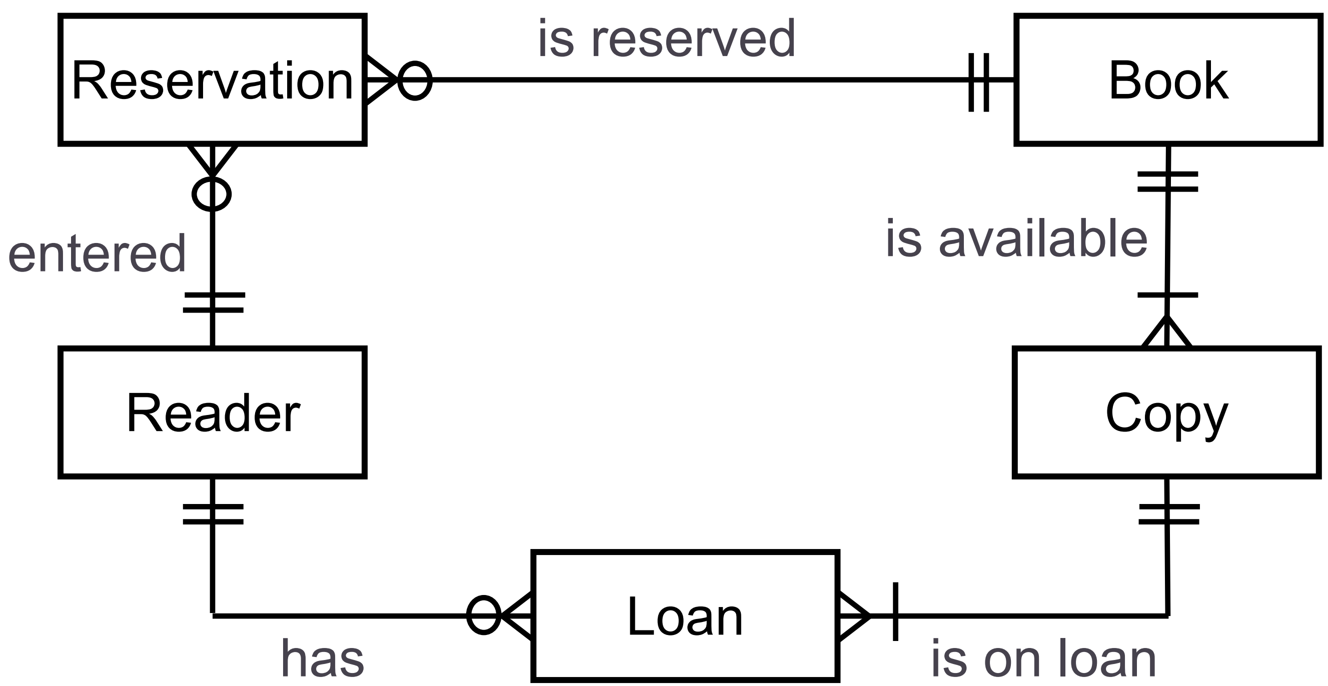
None or more occurrences

ERD example – Transport

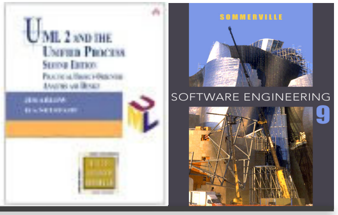


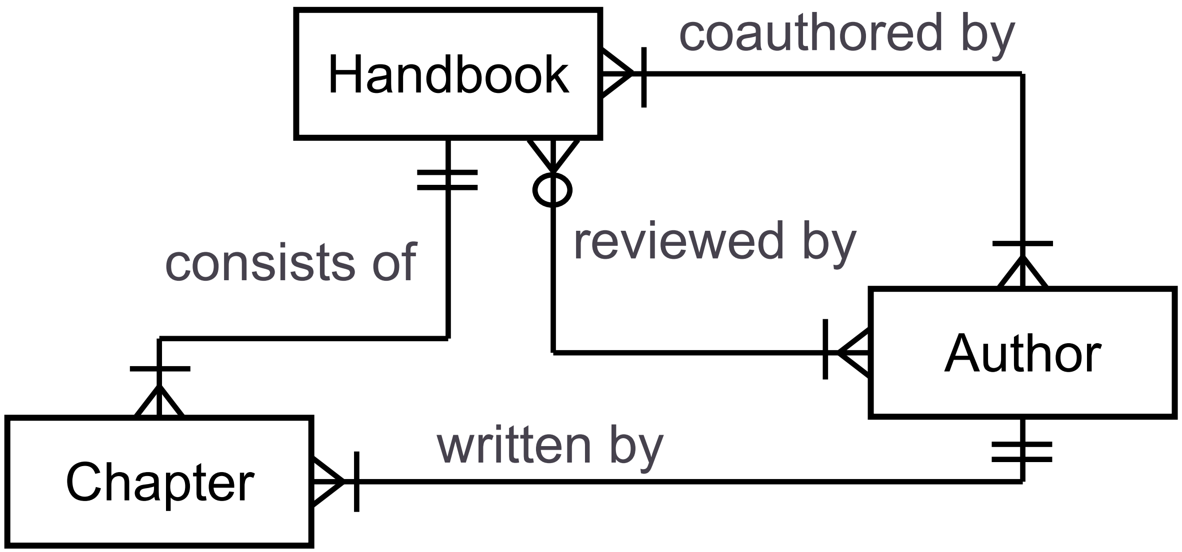
ERD example – Library



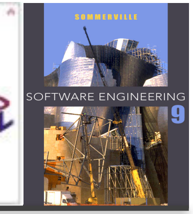


ERD example – Book editing





Relational database design based on ERDs

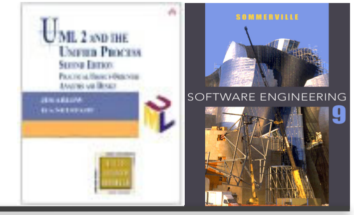


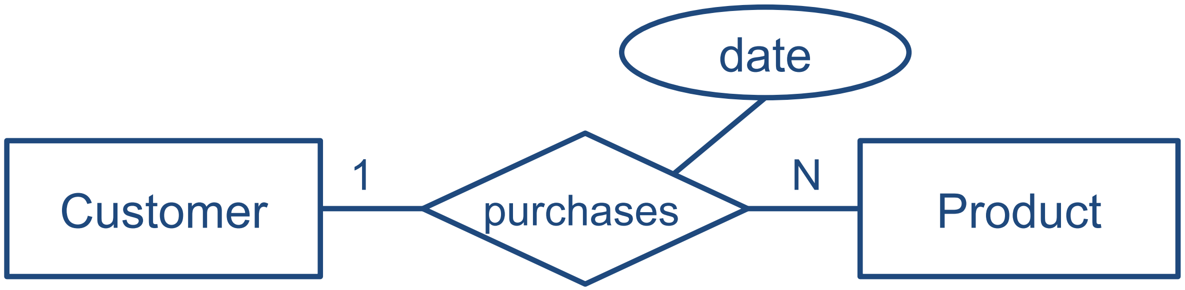
- Entity-relationship modeling is a first step towards database design.

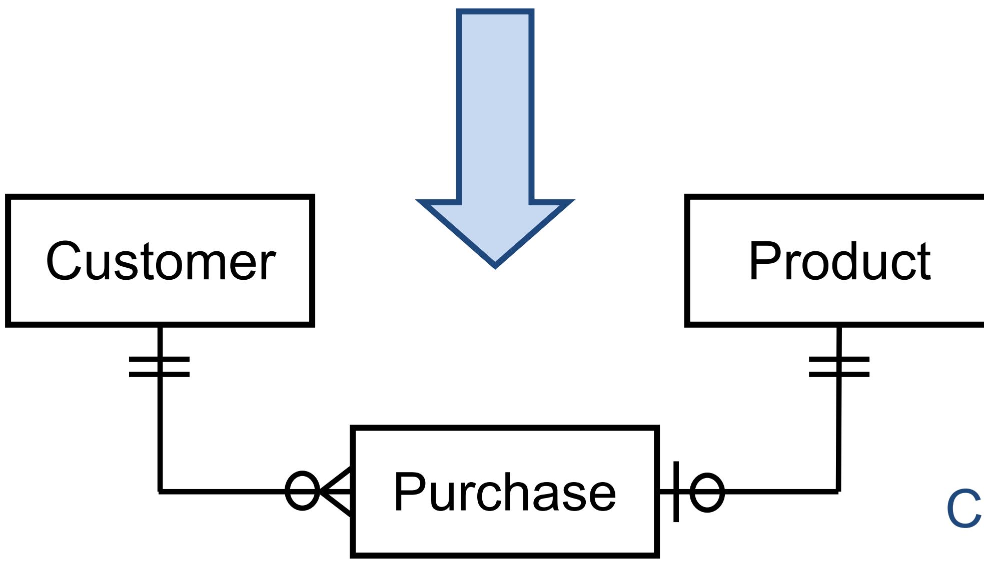
Database design process:

1. **Determine the purpose of the database.**
2. **Find and organize the information required** - Create ERD model of the system. Each entity type becomes a table, attribute becomes a column, entity becomes a row in the table. Handle relationships with attributes, association entities and M:N relationships.

Relationships to entities

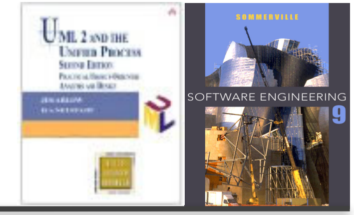


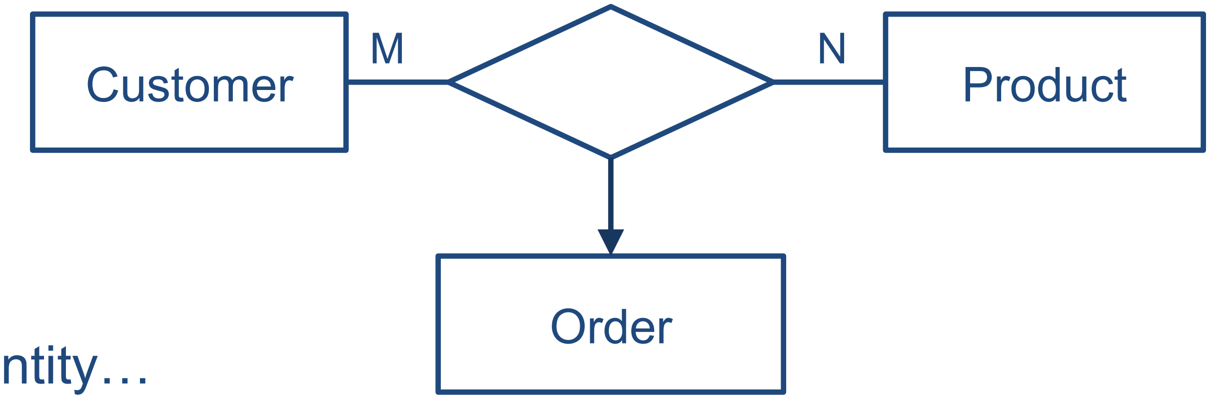




Can the purchase entity be omitted?

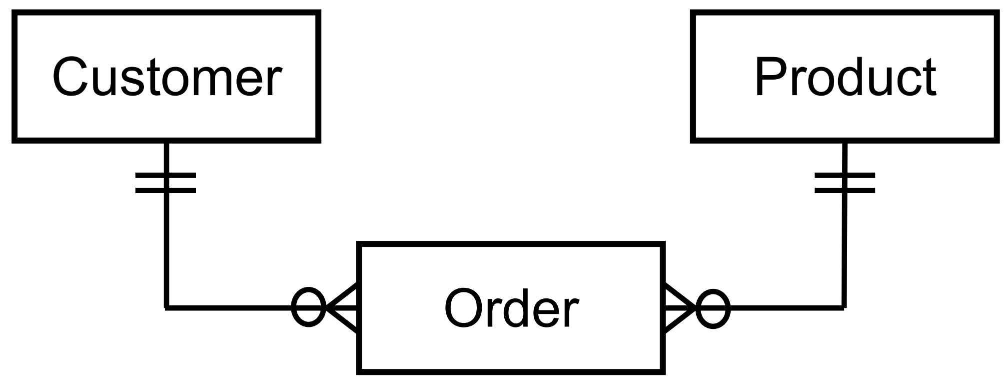
Association entities



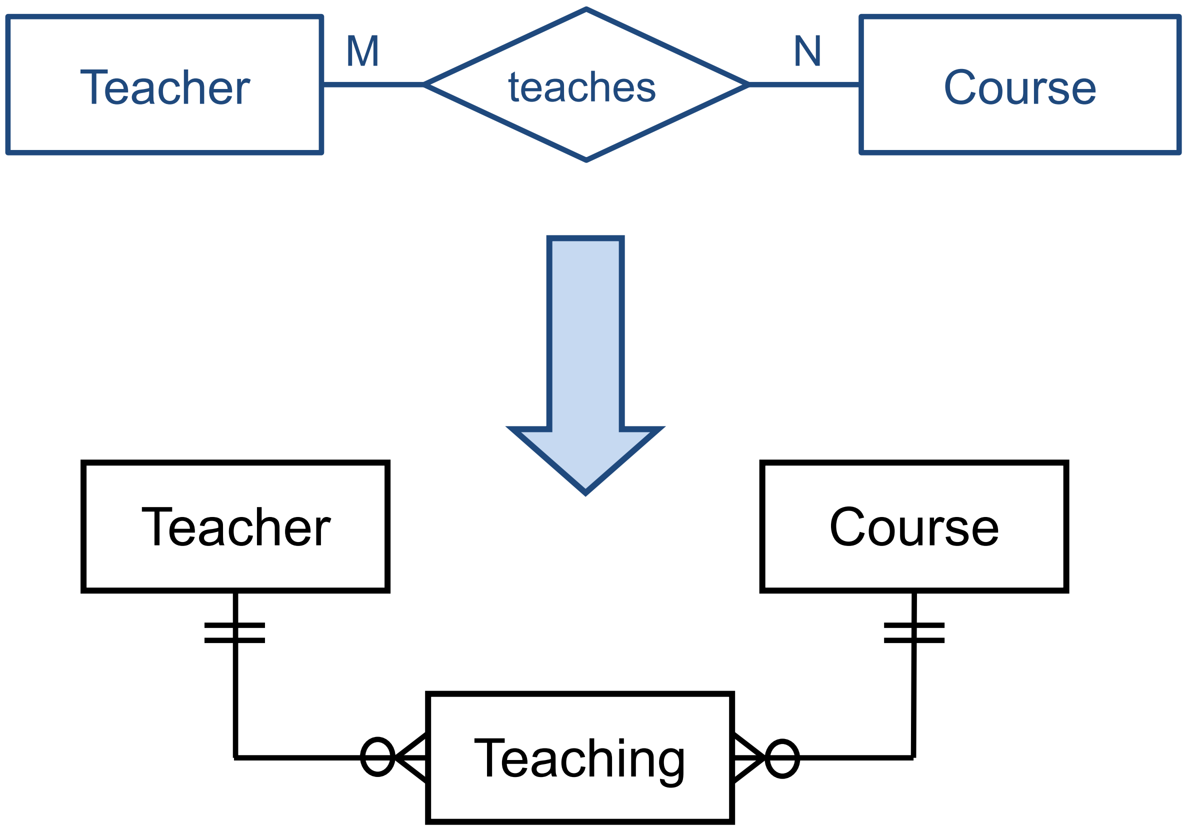


Association entity…

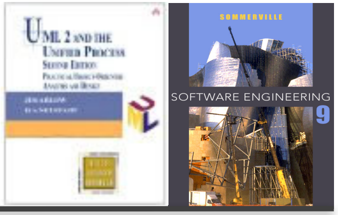
… can become
an entity on its own

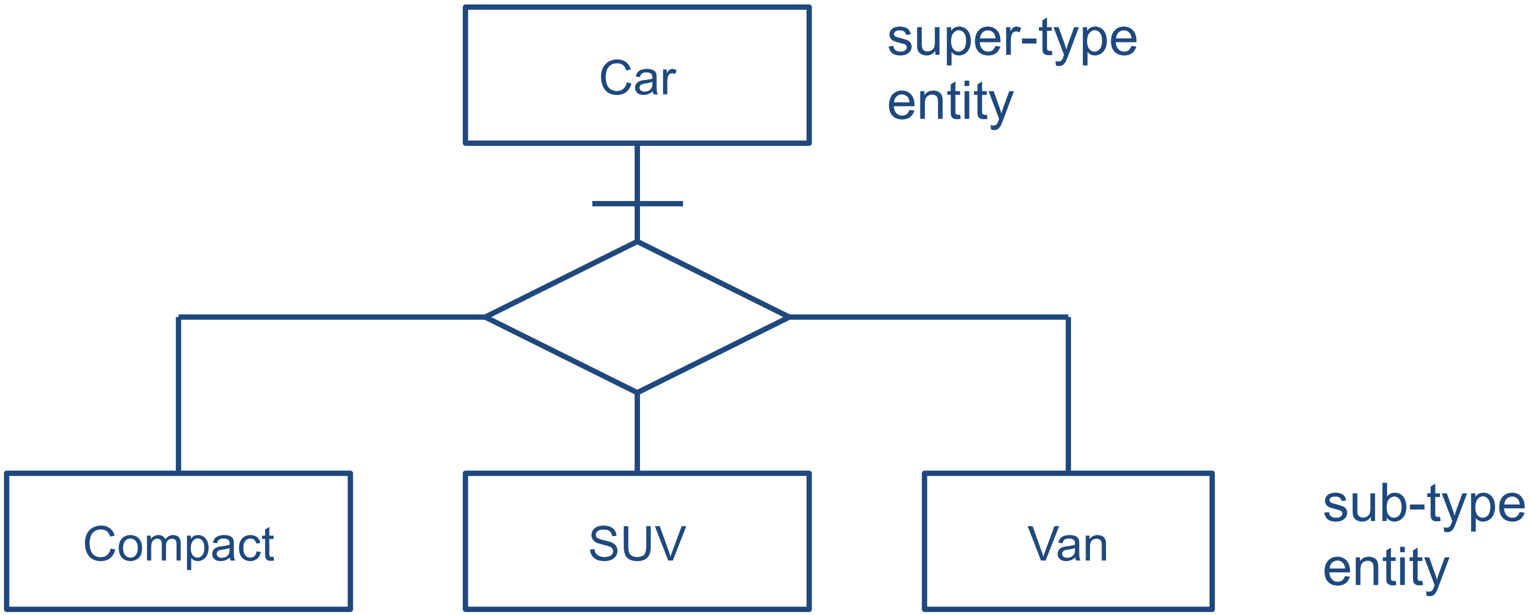


M:N relationships



Sub-types and super-types

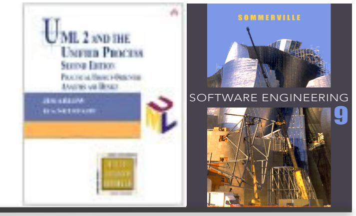




✧ Three options:

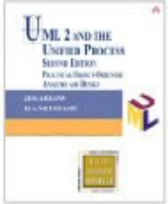
- One big Car entity with all attributes
- Three smaller Compact, SUV and Van entities
- Four entities with relationship between sub-type and super-type entity

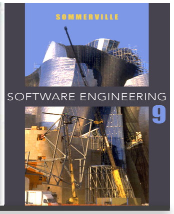
Database design process (continued)



3. **Specify primary keys** - Choose each table's primary key. The primary key is a column that is used to uniquely identify each row. An example might be Product ID or Order ID.
4. **Apply the normalization rules** - Apply the data normalization rules to see if tables are structured correctly. Make adjustments to the tables.
5. **Refine the design** - Analyze the design for errors. Create tables and add a few records of sample data. Check if results come from the tables as expected. Make adjustments to the design, as needed.

Entities and keys





✧ Superkey

- A set of attributes that **uniquely identifies** each entity.

✧ Candidate key

- A **non-redundant** superkey, i.e. all items of a candidate key are necessary to identify an entity, no key attribute can be removed.
- There can be more combinations of entity attributes that can be used as candidate keys.

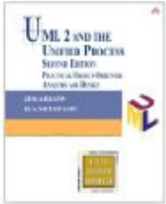
✧ Primary key

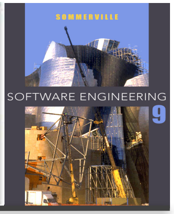
- The **selected candidate key**, marked with # symbol.

✧ Foreign key

- A set of attributes in one entity that **uniquely identifies** (i.e. is a primary key in) **another entity**.

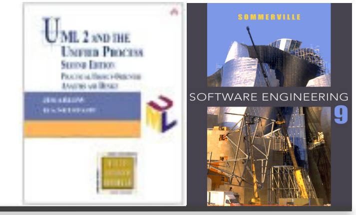
Data normalization goals by E.F. Codd



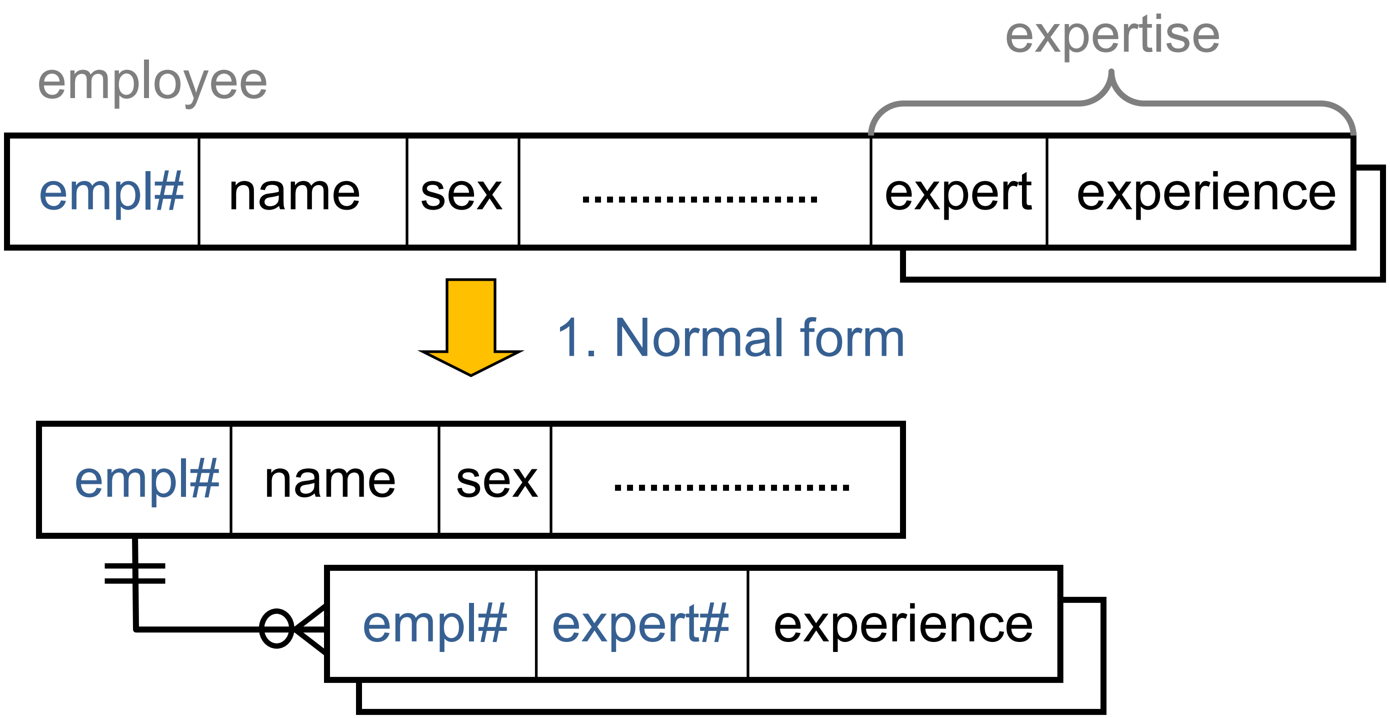


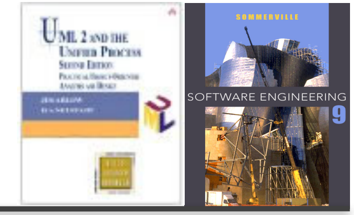
- Minimize **redundancy** and **dependency**
  - Minimize redesign when extending database structure
  - Make the data model more informative to users
- Free the database of modification anomalies
  - **Update anomaly** – the same information expressed on multiple rows → update resulting in logical inconsistencies.
  - **Insertion anomaly** – certain facts cannot be recorded, because of their binding with another information into one record.
  - **Deletion anomaly** – deletion of data representing certain facts necessitating deletion of unrelated data.
- Avoid bias towards any particular **pattern of querying**

1. Normal form – no repeating groups

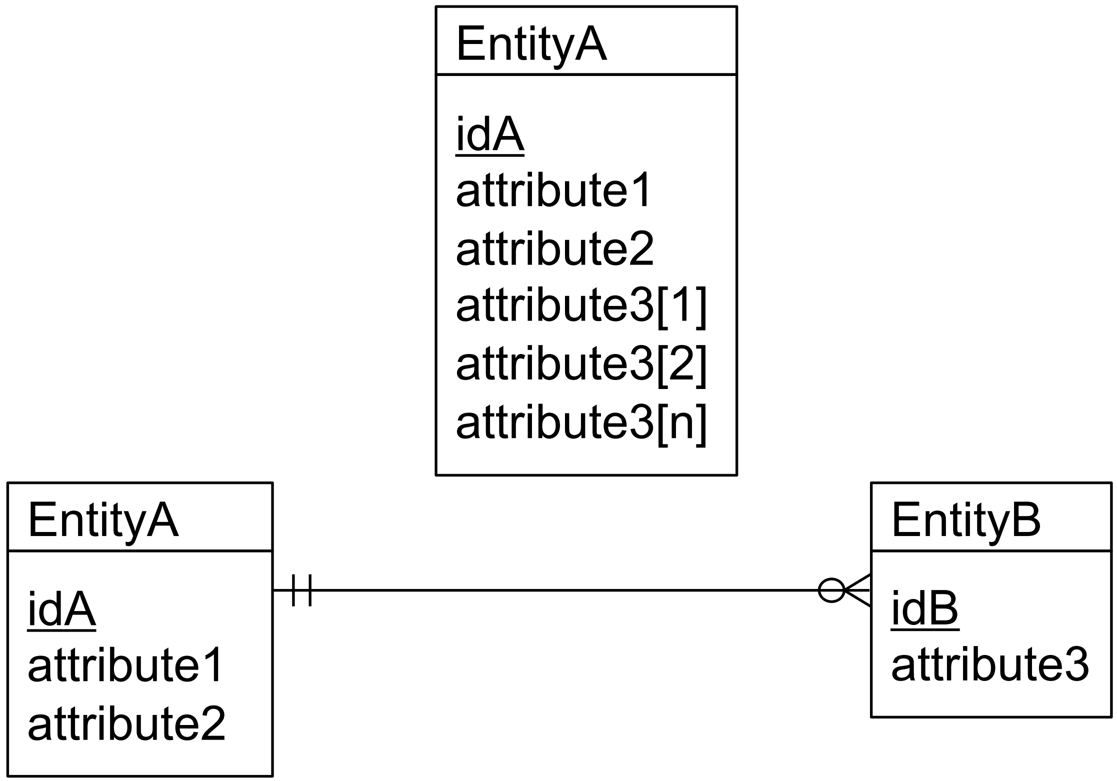


Def.1NF: A relation is in 1NF if the domain of each attribute contains only **atomic values**, and the value of each attribute contains only a **single value** from that domain.

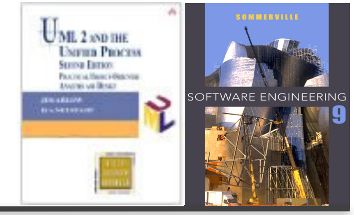




1. Normal form – normalization example

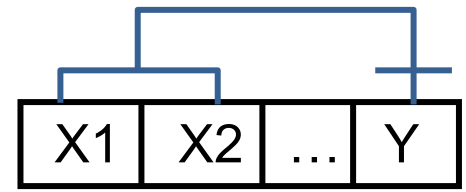


Functional dependency

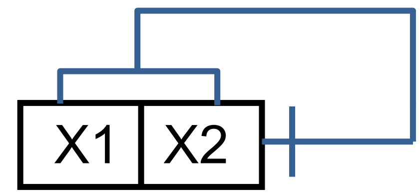


✧ Functional dependency

- In a given table, an attribute Y is said to have a functional dependency on a set of attributes X if and only if each X value is associated with precisely one Y value.

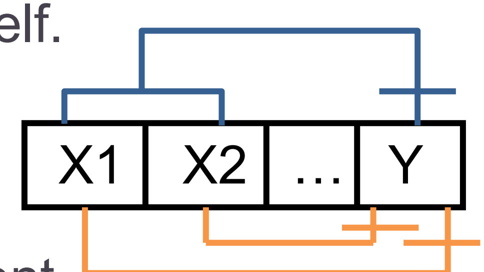


✧ Trivial functional dependency



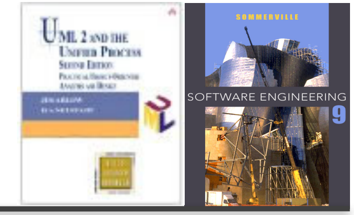
- A trivial functional dependency is a functional dependency of an attribute on a superset of itself.

✧ Full functional dependency



- An attribute is fully functionally dependent on a set of attributes X if it is: functionally dependent on X, and not functionally dependent on any proper subset of X.

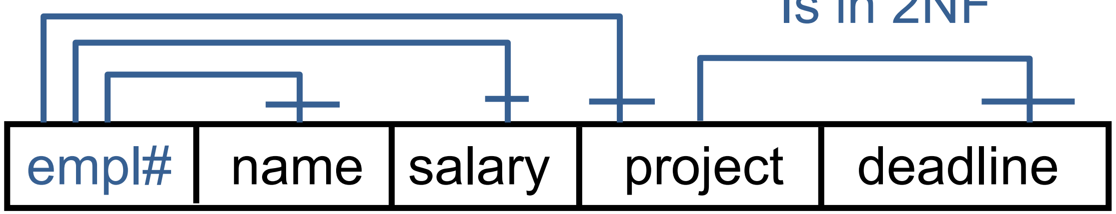
2. Normal form – no partial dependency



Def. 2NF: In 1NF and no non-prime attribute in the table is functionally dependent on a proper subset of any candidate key.

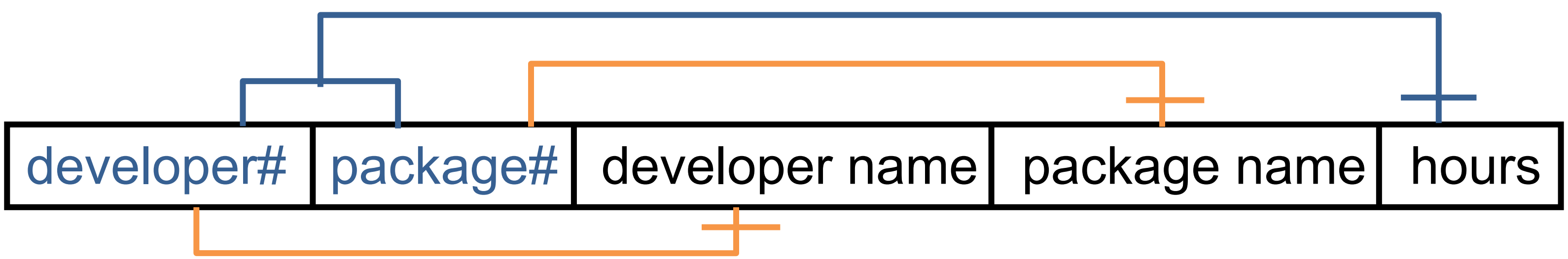
Example EMPLOYEE

Is in 2NF



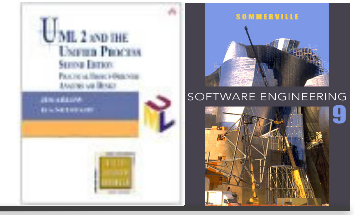
Example PROGRAMING

Not in 2NF



What anomalies can you identify in this example?

2. Normal form – no partial dependency



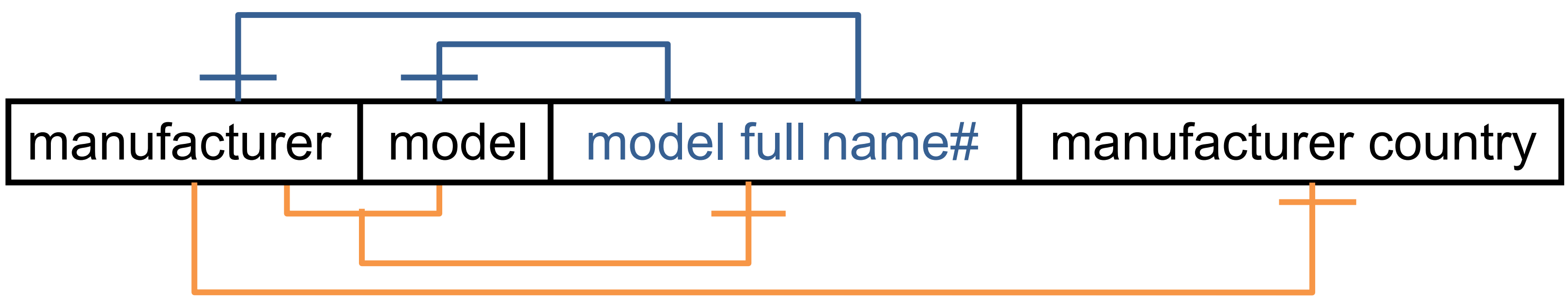
not part of any candidate key

Def. 2NF: In 1NF and no non-prime attribute in the table is functionally dependent on a proper subset of any candidate key.

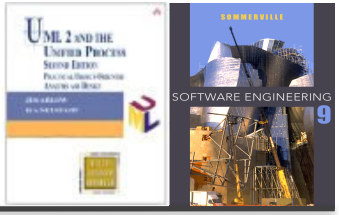
- Does the “candidate key” part of the definition make difference?
- When there is only one-item primary key, is 2NF guaranteed?

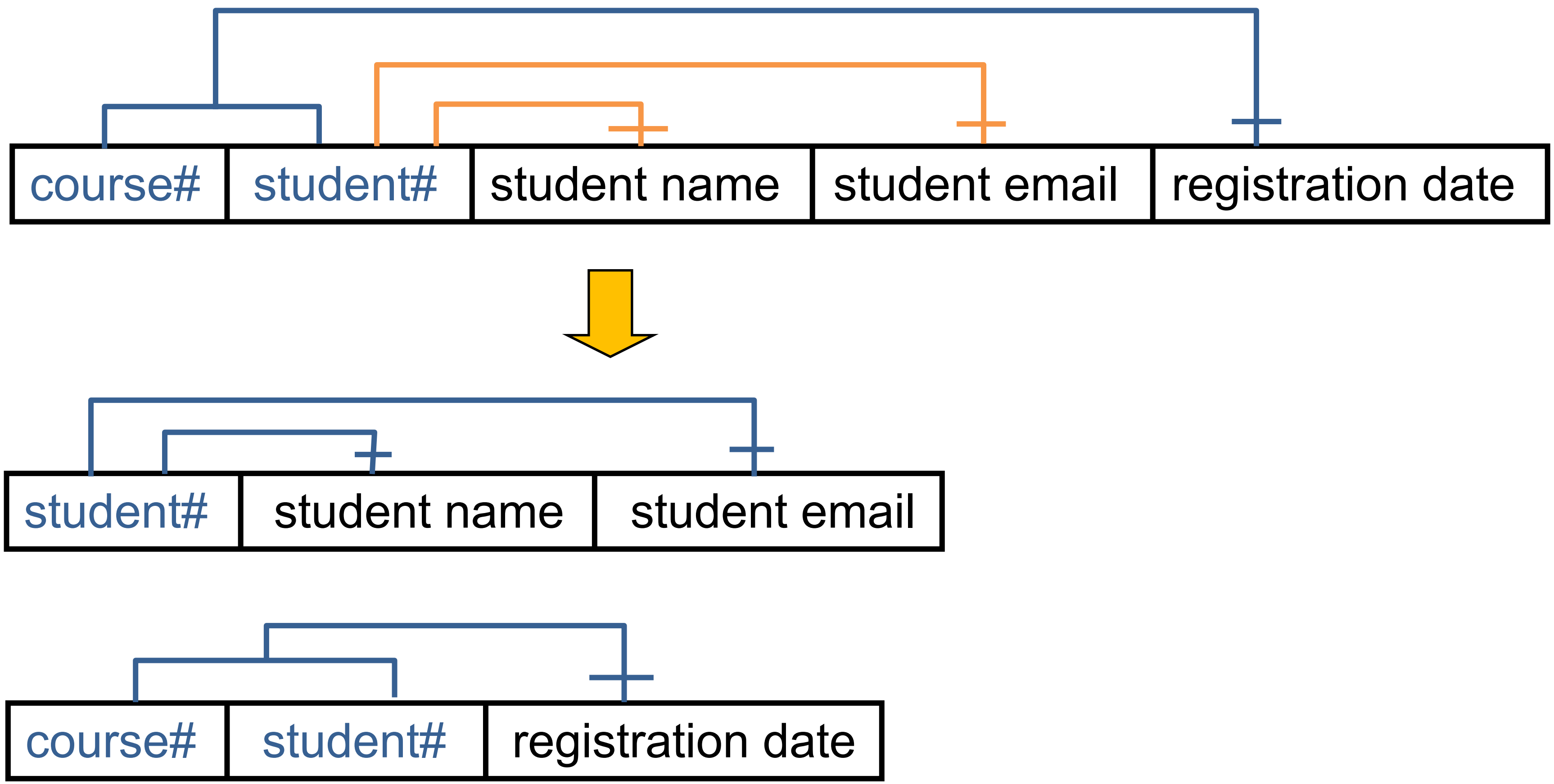
Example DISHWASHER MODELS

Not in 2NF

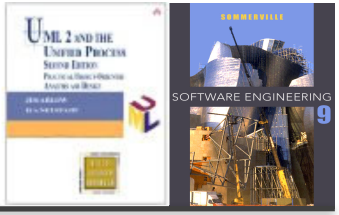


2. Normal form – normalization example



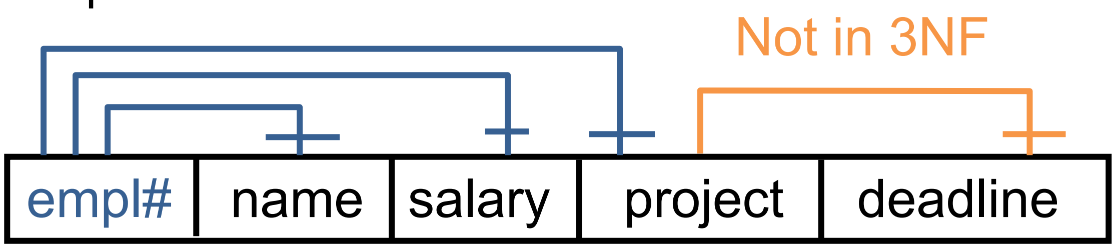


3. Normal form – no transitive dependency



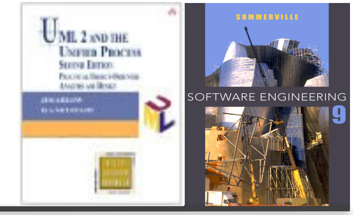
Def. 3NF: In 2NF and every non-prime attribute is non-transitively (i.e. only directly) dependent on every candidate key.

Example EMPLOYEE

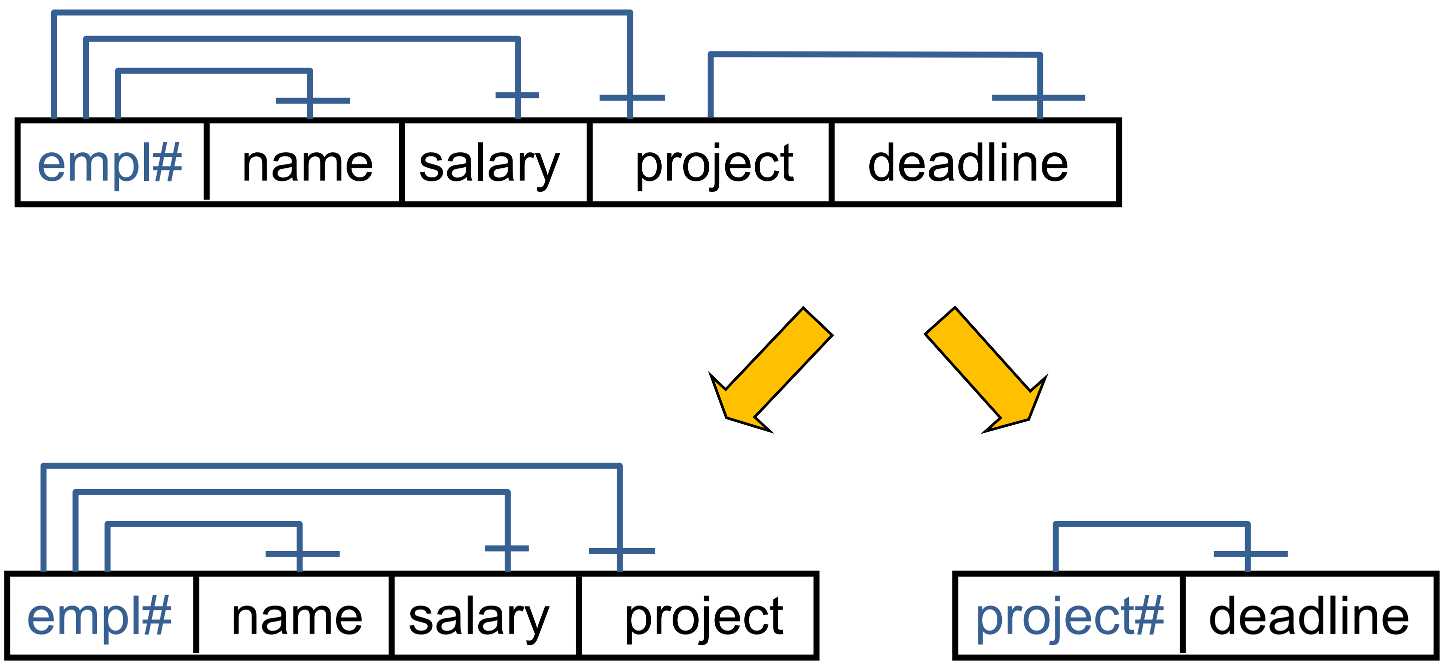


What anomalies can you identify in this example?

3. Normal form – normalization example



deadline is transitively dependent on empl#



ERD vs. UML Class Diagram

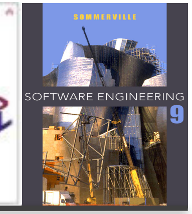

✧ Class diagrams

- model both **structural and behavior features** of a system (attribute and operations),
- contain **many different types of relationships** (association, aggregation, composition, dependency, generalization), and
- are more likely to **map into real-world objects**.

✧ Entity relationship models

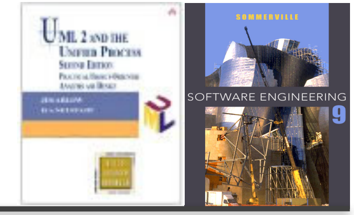
- model only **structural data view** with a low variety of relationships (simple relations and rarely generalization), and
- are more likely to **map into database tables** (repetitive records).
- They allow us to design **primary and foreign entity keys**, and used to be normalized to simplify data manipulation.

ERD vs. UML Class Diagram



- Although there can be one to one mapping between ERD and Class diagram, it is very common that
  - one class is mapped to more than one entity, or
  - more classes are mapped to a single entity.
- Furthermore, not all classes need to be persistent and hence reflected in the ERD model, which uses to be driven by the database design.
- **Summary:**
  - ERD is **data-oriented** and **persistence-specific**
  - Class diagram targets also operations and is persistence independent

Key points



- ✧ Structured analysis, and YMSA in particular, models systems from the perspectives of:
  - system interaction with its environment (CD), and
  - hierarchy of system processes and data flows (DFD).
- ✧ Data modeling, and ERD in particular, focuses on modeling data **entities, relationships and attributes**.
- ✧ Data normalization focuses on reducing **redundancy and dependency** in database design, and on avoiding bias towards a particular **pattern of querying**.
  - 1NF: no repeating groups
  - 2NF: no partial dependency
  - 3NF: no transitive dependency