ifis

Institut für Informationssysteme
Technische Universität Braunschweig

Relational Database Systems I

Wolf-Tilo Balke
Simon Barthel

Institut für Informationssysteme
Technische Universität Braunschweig
www.ifis.cs.tu-bs.de

An example

Detour

- We want to model a simple university database
  - In our database, we have students. They have a name, a registration number, and a course of study.
  - The university offers lectures. Each lecture may be part of some course of study in a certain semester. Lectures may have other lectures as prerequisites. They have a title, provide a specific number of credits and have an unique ID
  - Each year, some of these lectures are offered by a professor at a certain day at a fixed time in a specific room. Students may register for that lecture.
  - Professors have a name and are member of a specific department.

An example

Detour

- Which are our entity types?
  - In our database, we have **students**. They have a name, a registration number and a course of study.
  - The university offers **lectures**. Each lecture may be part of some course of study in a certain semester. Lectures may have other lectures as prerequisites. They have a title, provide a specific number of credits and have unique ID
  - Each year, some of these lectures are offered by a **professor** at a certain day at a fixed time in a specific room. Students may attend that lecture.
  - Professors have a name and are member of a specific department.

An example

Detour

Student

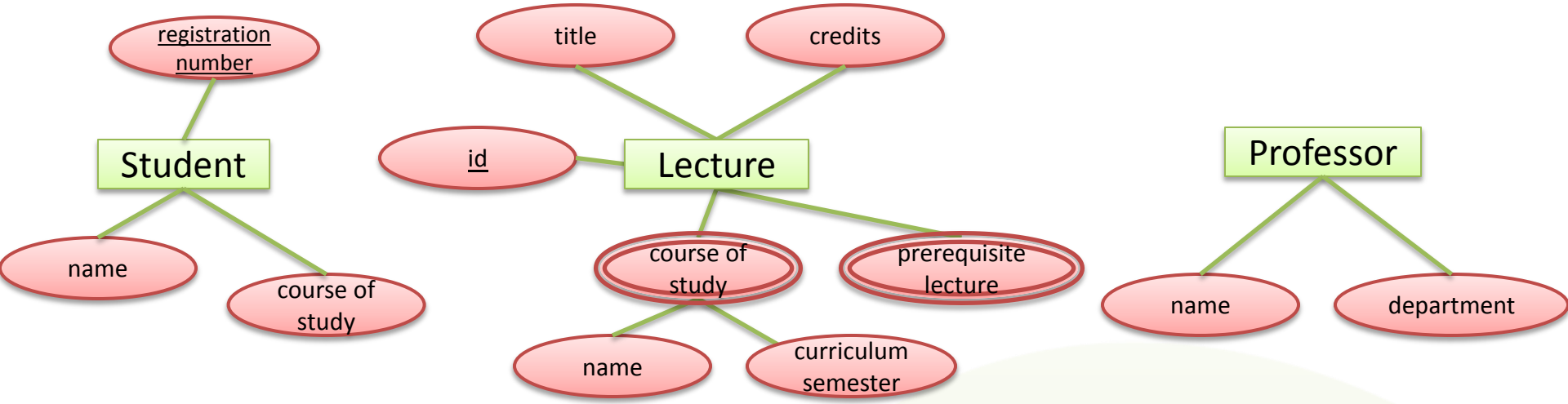
Lecture

Professor

- What attributes are there?
  - In our database, we have students. They have a **name**, a **registration number** and a **course of study**.
  - The university offers lectures. Each lecture may be part of some **course of study** in a certain **semester**. Lectures may have other lectures as **prerequisites**. They have a **title**, provide a specific number of **credits** and have **unique ID**
  - Professors have a **name** and are member of a specific **department**.

An example

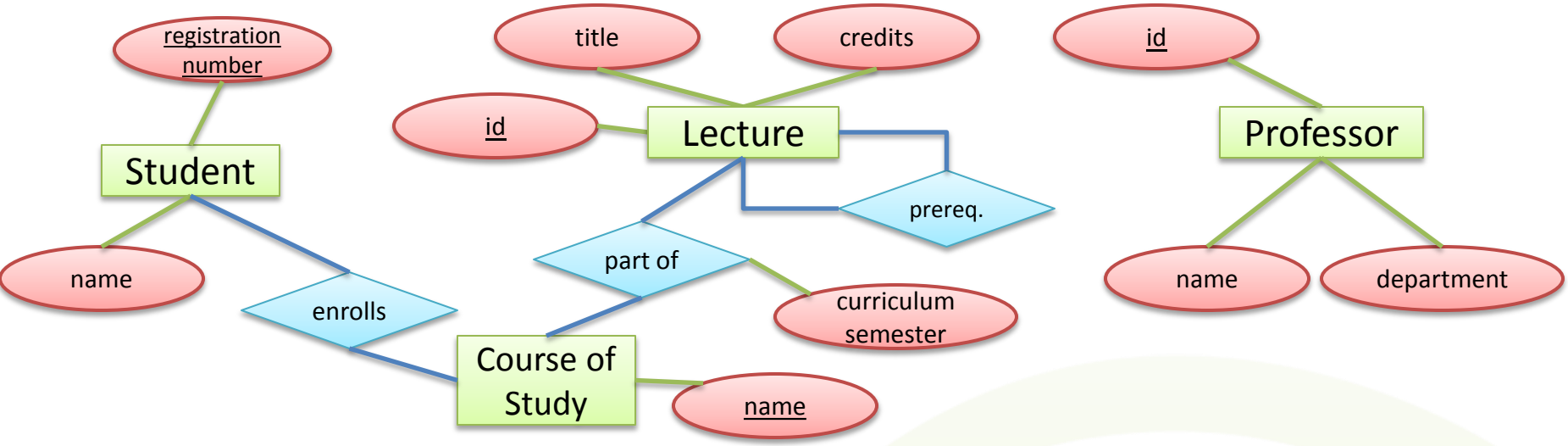
Detour



- First try…
  - This model is **really crappy**!
  - "Course of study" does not seem to be an attribute
    - Used by student and lecture. Even worse, lecture refers to a course of study in a specific curriculum semester.
    - Use additional entity type with relationships!
  - "Prerequisite lecture" also is not a good attribute
    - Prerequisite lectures are also lectures. Use a relationship instead!
  - "Professor" does not have key attributes

An example

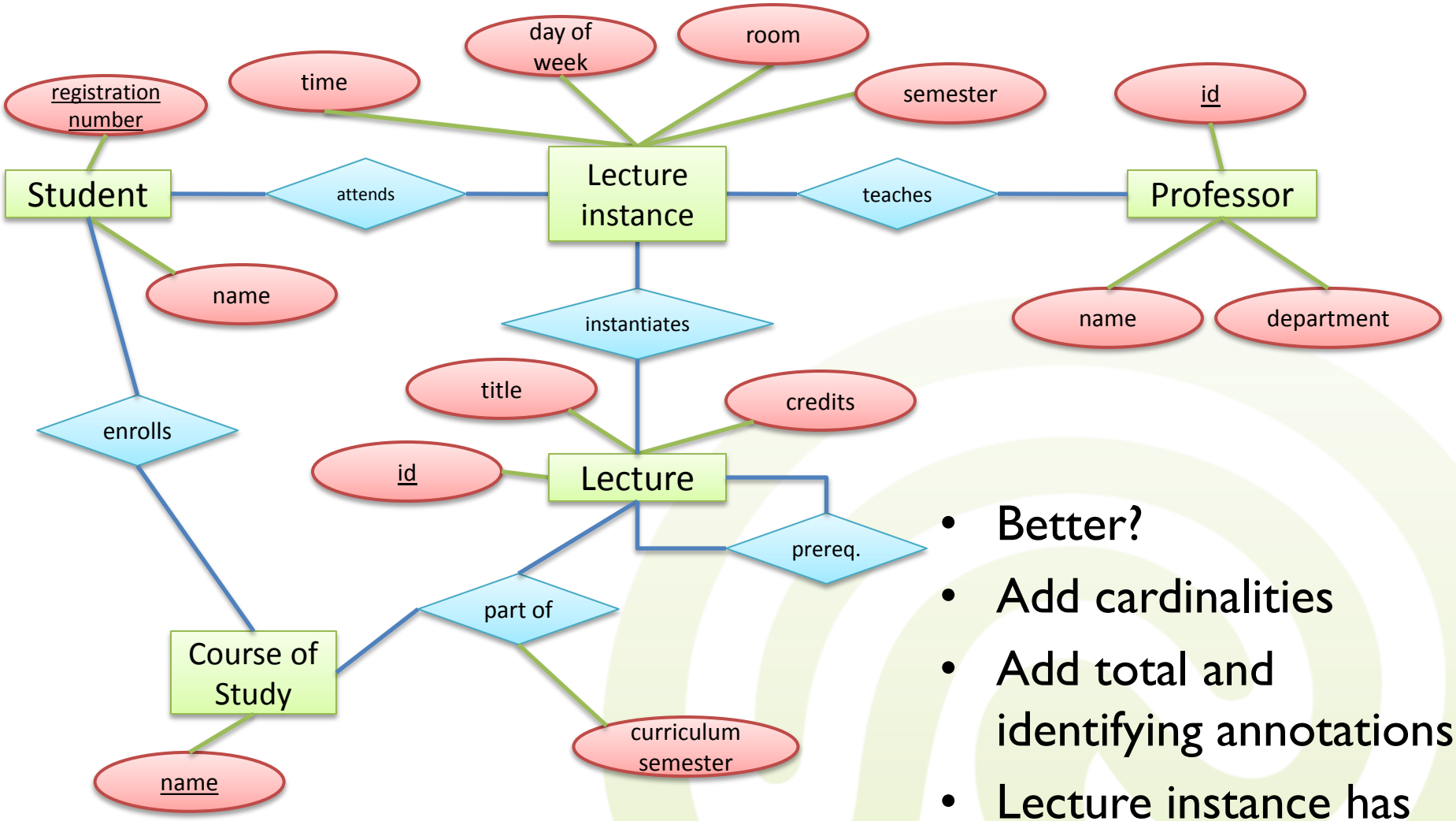
Detour



- Second try
  - Professors use a **surrogate** key now
    - Key is automatically generated and has no meaning beside unique identification
  - Course of study is an entity type now
- Which entity types are additionally related?
  - "Each year, some lectures of the **pool of all lectures** are offered by a professor at a certain day at a fixed time in a specific room. Students may attend that lecture."

An example

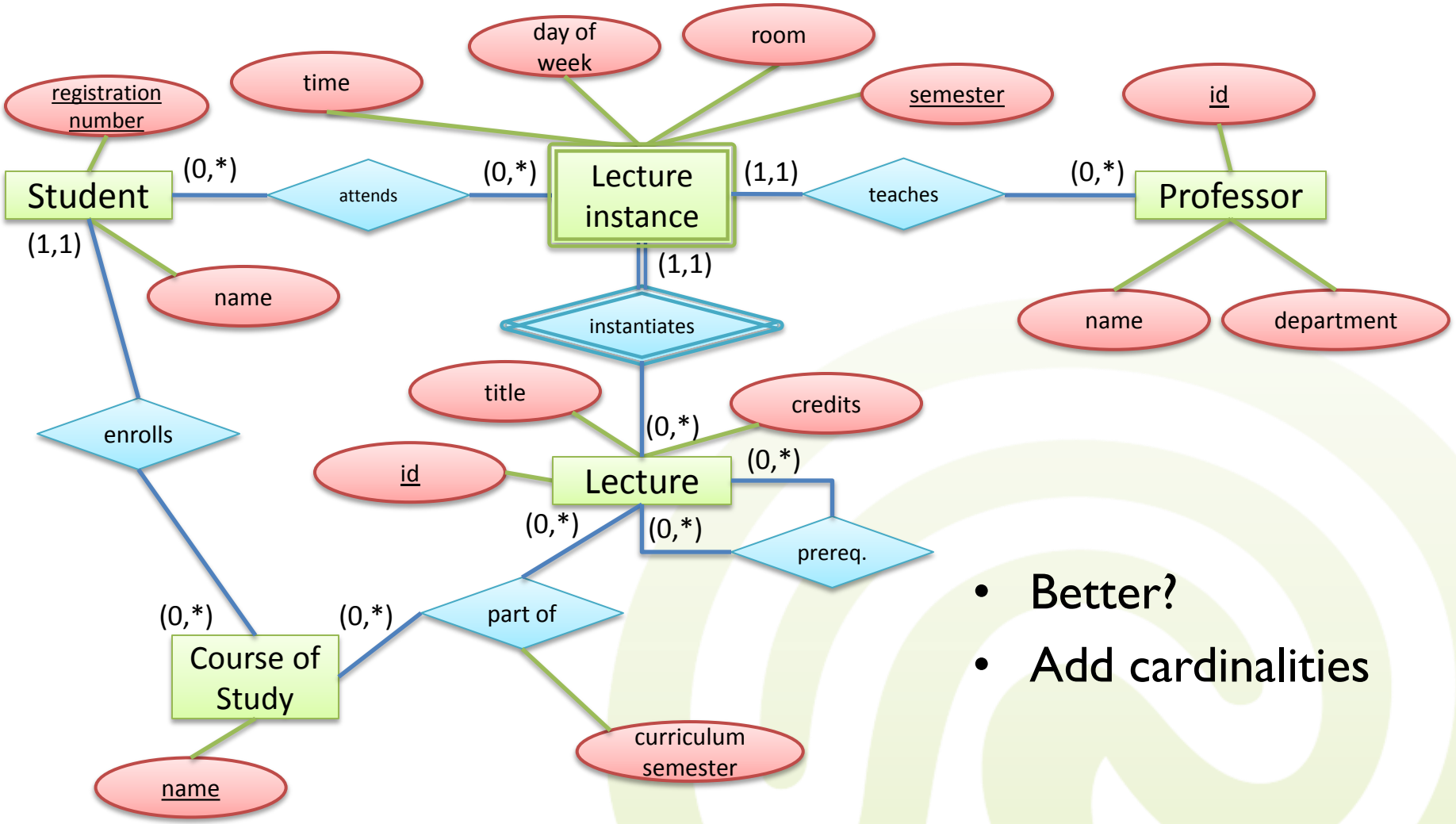
Detour



- Better?
- Add cardinalities
- Add total and identifying annotations
- Lecture instance has no key

An example

Detour



- Better?
- Add cardinalities

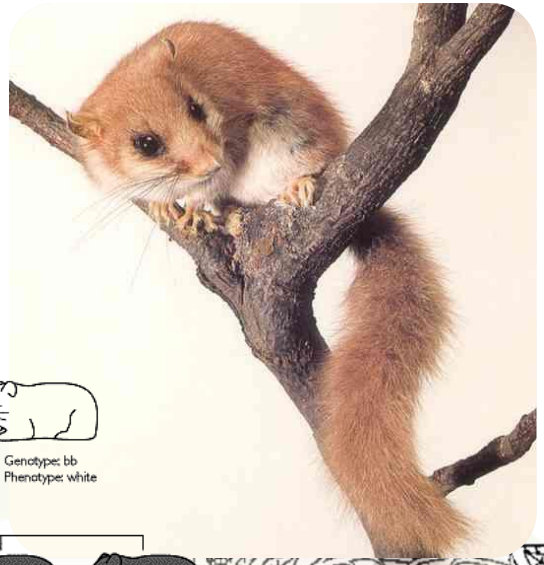
An example

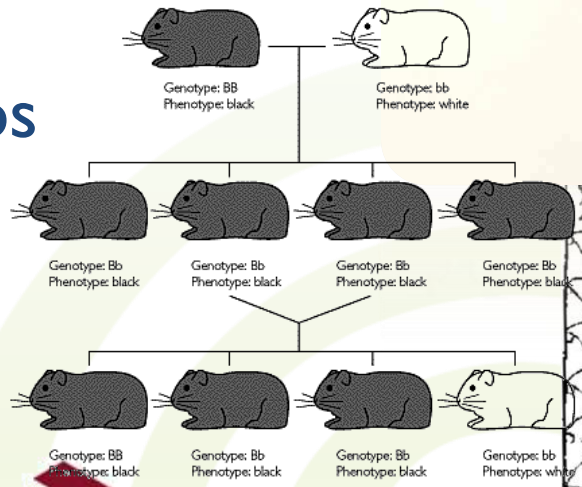
Detour

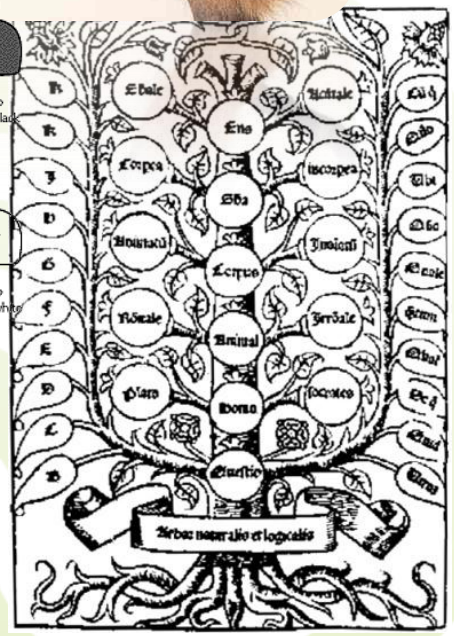
- Modeling is not that simple
- Many possible (and also correct) ways of modeling the same miniworld
  - Some are more elegant, some are less elegant
- Models alone are not enough, they need to be documented
  - What are the meanings of the attributes? The meanings of the relationships?

3 Extended Data Modeling

- Alternative ER Notations
- Extended ER
  - Inheritance
  - Complex Relationships
- UML







3.1 ER – Alternative Notations

- There is a plethora of alternative notations for ER diagrams
  - Different styles for entities, relationships and attributes
  - No standardization among them
  - Also, notations are often freely mixed
    - ER diagrams can look completely different depending on the used tool / book

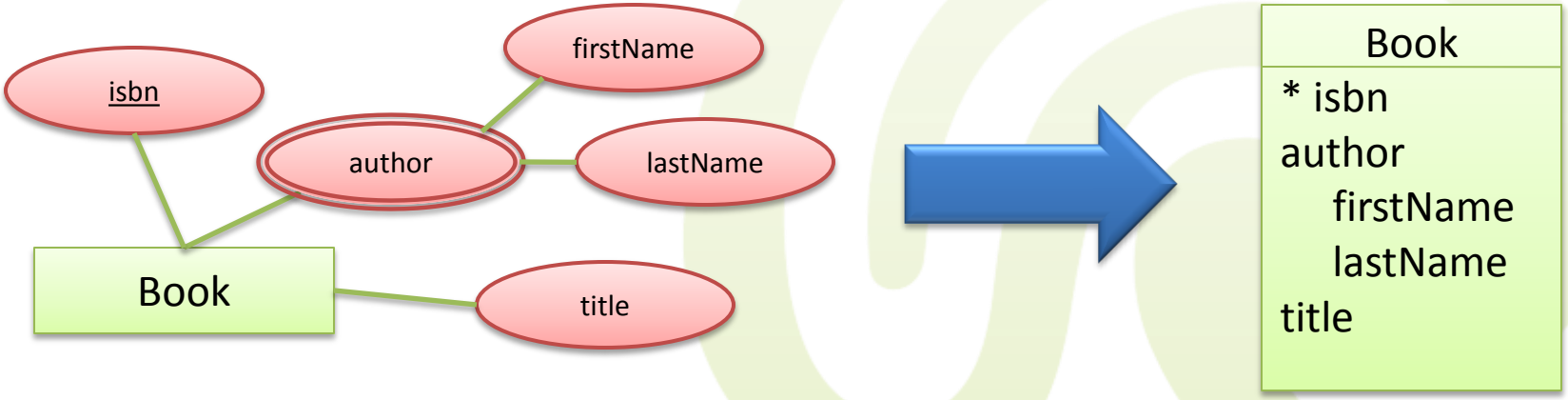
- In the following, we introduce the (somewhat popular) crow's foot notation

3.1 ER – Crow's Foot Notation

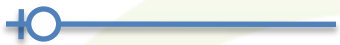
- **Crow's foot** notation was initially developed by Gordon Everest
  - Derivate of 3.1 ERD notation
  - Main Goal
    - Consolidate graphical representation
    - Provide a better and faster overview
    - Allow for easier layouting
  - Widespread use in many current tools and documentations

3.1 ER – Crow's Foot Notation

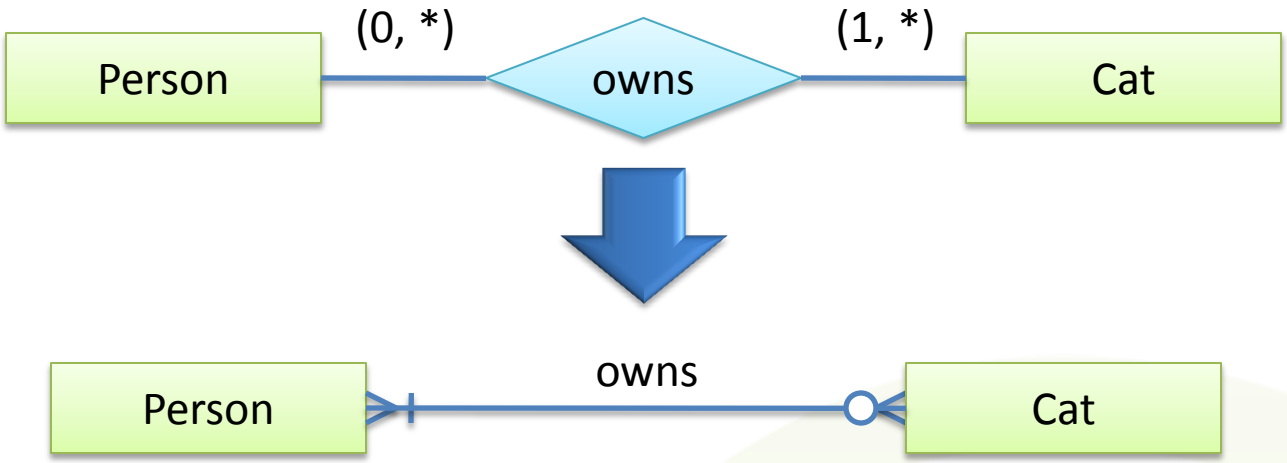
- **Entity Types**
  - Entity Types are modeled with a named box
  - Attribute names are written inside the box separated by a line
    - Key attributes are marked with a leading asterisk
    - Composite attributes are represented with indentation



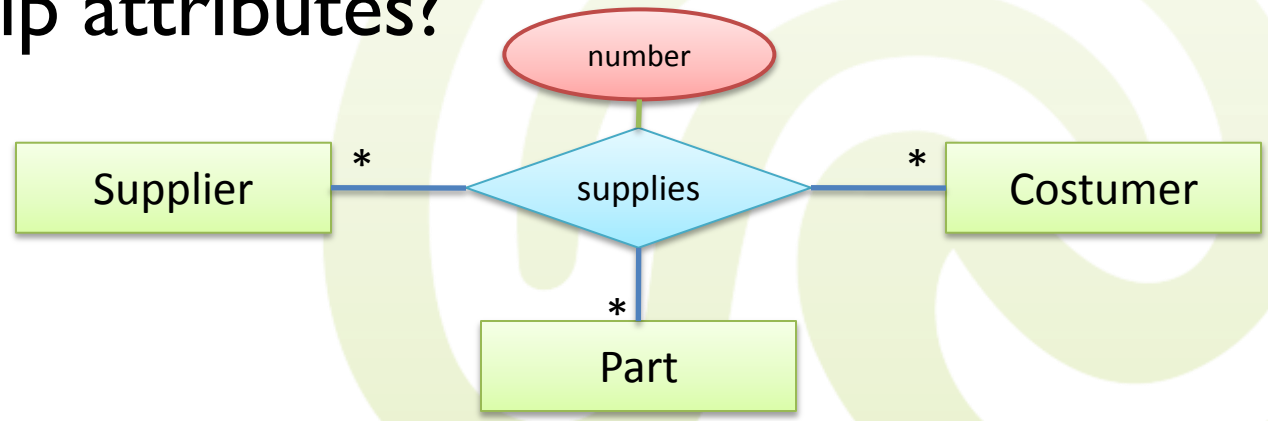
3.1 ER – Crow's Foot Notation

- **Relationship Types**
  - Relationship types are modeled by lines connecting the entities
  - Line is annotated with the name of the relationship which is a verb
  - Cardinalities are represented graphically
    - **(0, 1)** : Zero or one 
    - **(1, 1)** : Exactly one 
    - **(0, *)** : Zero or more 
    - **(1, *)** : one or more 
    - ATTENTION: Cardinalities are written on the **opposite side** of the relationship (in contrast to "classic ER")

3.1 ER – Crow's Foot Notation

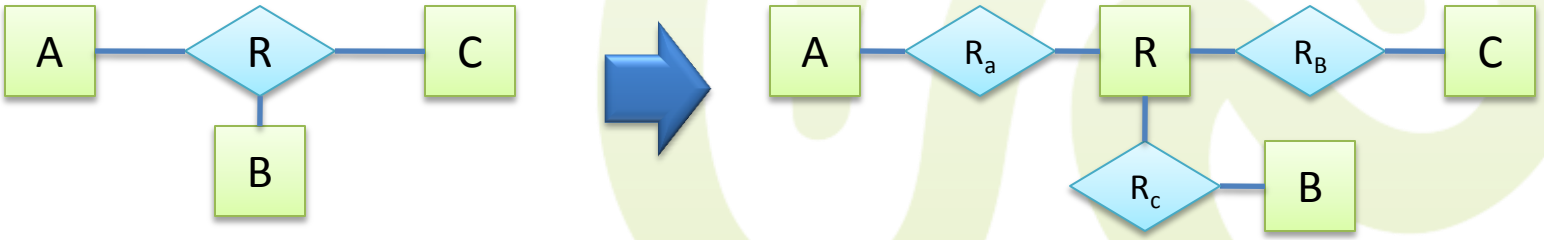


- What happens to n-ary relationships or relationship attributes?

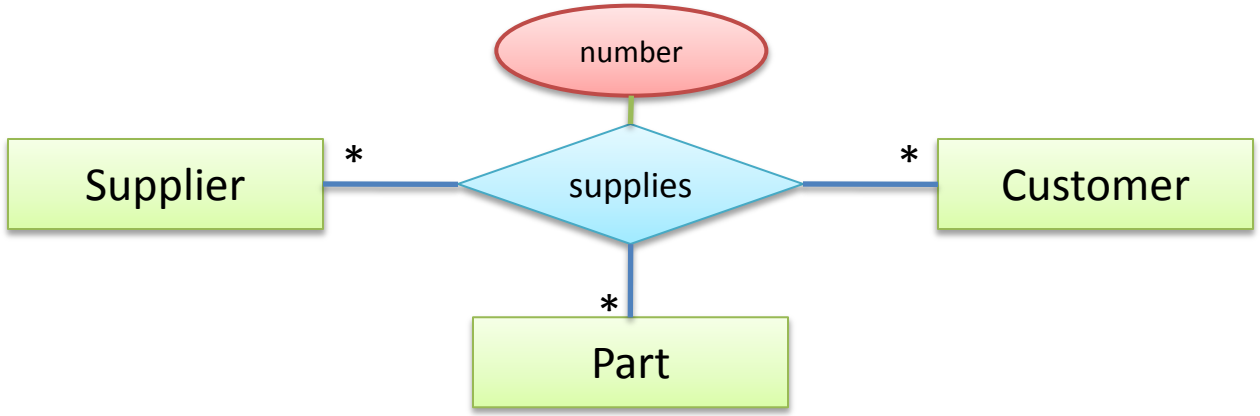


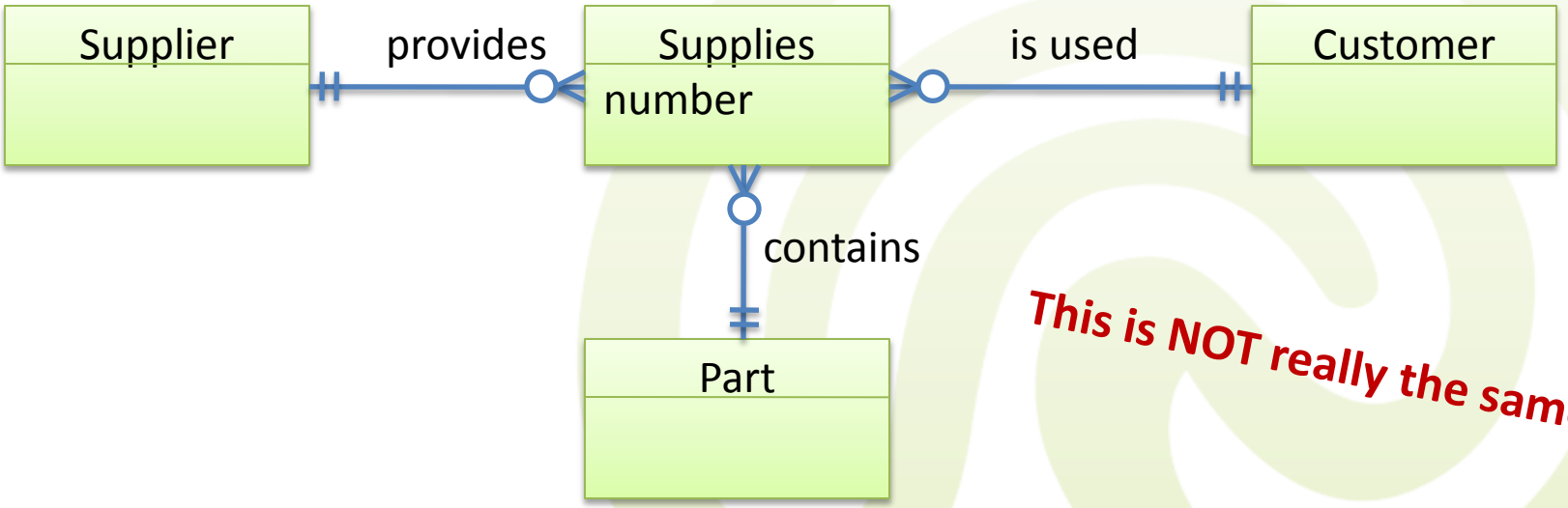
3.1 ER – Crow's Foot Notation

- **Problem**
  - N-ary relationship types **are not supported** by crow's foot notation, neither are relationship attributes
- **Workaround solution**:
  - **Intermediate entities** must be used
    - N-ary relationships are broken down in a series of **binary** relationship types anchoring on the intermediate entity



3.1 ER – Crow's Foot Notation



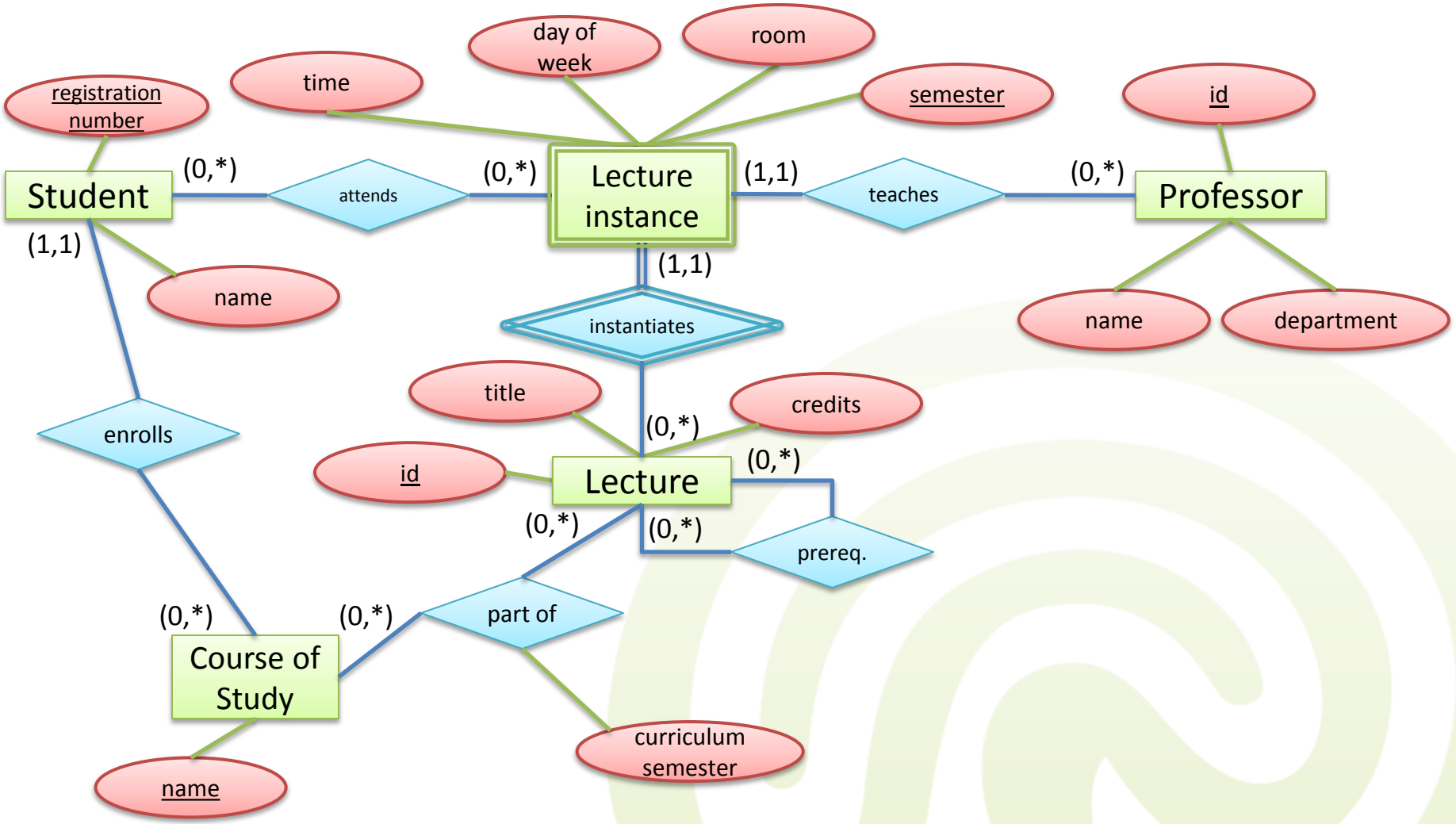


This is NOT really the same!

3.1 ER – Crow's Foot Notation

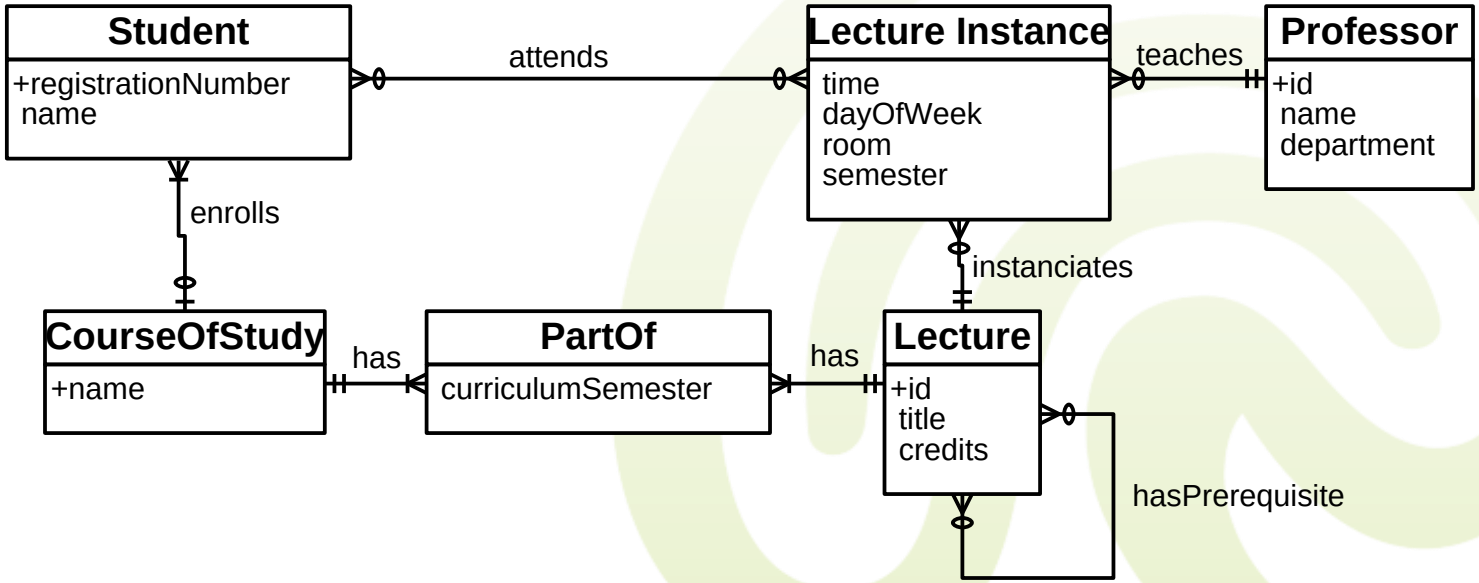
- Originally, ER diagrams were intended to be used on a **conceptual** level
  - Model data in an abstract fashion **independent** of implementation
- Crow's foot notation sacrifices some conceptual expressiveness
  - Model is closer to the **logical** model (i.e. the way the data is later really stored in a system)
  - This is **not** always **desirable** and may obfuscate the intended semantics of the model

3.1 An example



3.1 An example

- The same example using Crow's Foot Notation
  - Note that the "part of" relationship had to use an intermediate entity



3.1 ER – Even more notations…

- **Barker's notation**
  - Based on Crow's Foot Notation
  - Developed by Richard Barker for **Oracle**'s CASE modeling books and tools in1986
  - Cardinalities are represented differently
    - **(0, 1)** : Zero or one 
    - **(1, 1)** : Exactly one 
    - **(0, N)** : Zero or more 
    - **(1, N)** : one or more 
    - Cardinalities position similar to Crow's Foot notation and opposite to classic ER
  - Different notation of subtypes

3.1 ER – Even more notations…

- **Black Diamond Notation**
  - Cardinalities are represented differently
    - Cardinality annotation per relationship, not per relationship end
    - 1:1
    - 1:N
    - N:M

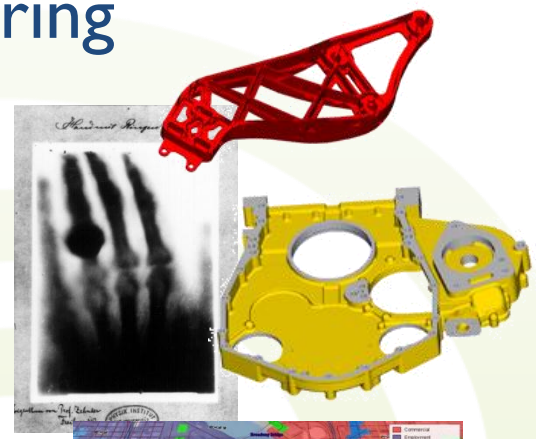
  - Also, N-ary relationships possible
    - ternary

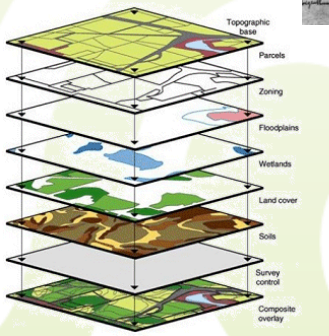
3.2 Extended Data Modeling

- Traditional **ER modeling** proved to be very **successful** in classic "DB" domains:
  - Accounting
  - Banking
  - Airlines
  - Business and industry applications in general
  - …

3.2 Extended Data Modeling

- However, in the late 70s, popularity of DBs extended into fields with more **complicated data formats**
  - Computer-aided design and manufacturing (CAD/CAM)
  - Geographic information systems (GIS)
  - Medical information systems
  - …
- Expressiveness of ERD is **not sufficient** here



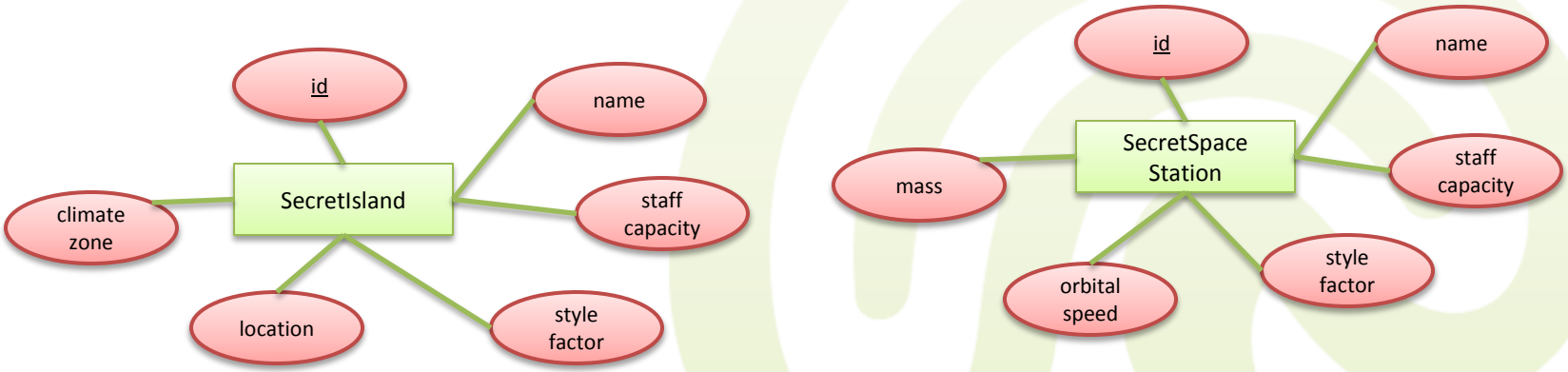


3.2 Extended Data Modeling

- Extended entity relationship (**EER**) models provide many additional **features** for more accurate **conceptual modeling**
  - Refinement of relationship types
    - Specialization and generalization
    - Class, subclass, and inheritance
  - Entity sets with existence dependencies
  - Extended modeling of domains and constraints
- Extended ER contains all features of "classic" ER

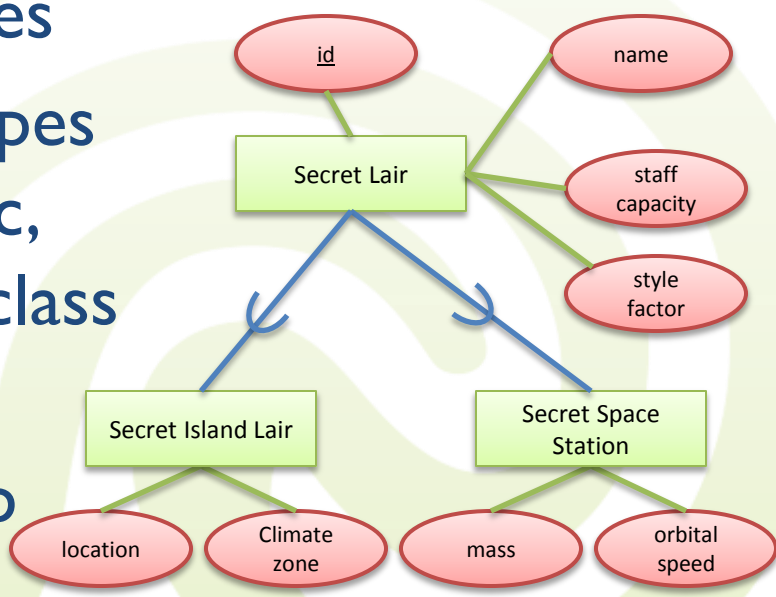
3.2 Subclasses / Superclasses

- Problem:
  - Model **secret lairs** to base highly secret research activities
  - **Secret island** and **secret space station** are special kinds of secret lairs, **share many attributes,** but still need some **unique attributes**



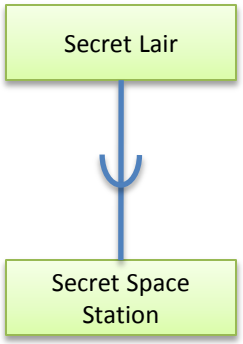
3.2 Subclasses / Superclasses

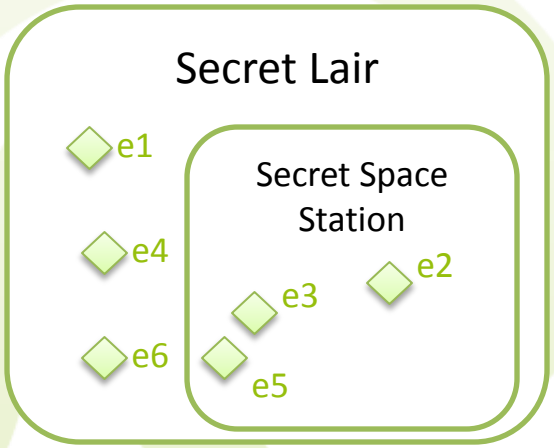
- Solution: **Subclasses** and **superclasses**
- A **subclass** entity type **inherits** all attributes and constraints from its **superclass** entity type
  - Subclasses may add additional attributes, constraints or relationship types
  - In EER, subclass relationship types are annotated with an open arc, which is opened to the super class (think of set inclusion)
  - Describes an "is-a" relationship



3.2 Subclasses / Superclasses

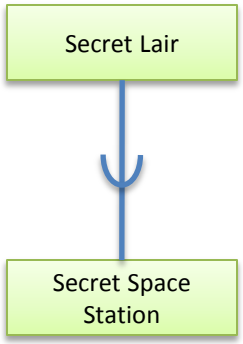
- **Subclass entity types** represent subsets of the entity set of the superclass' entity type
  - That is, an entity which is contained in the subclass is also contained in the superclass
  - In particular, no entity can **only** exist in a subclass set

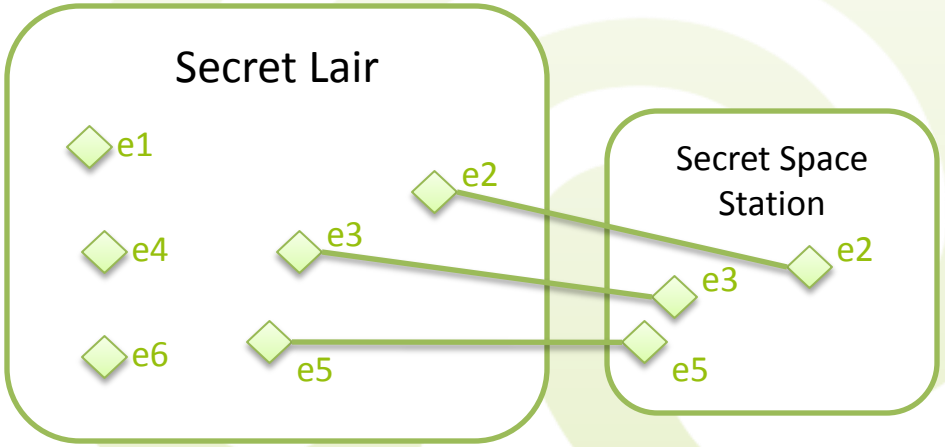




3.2 Subclasses / Superclasses

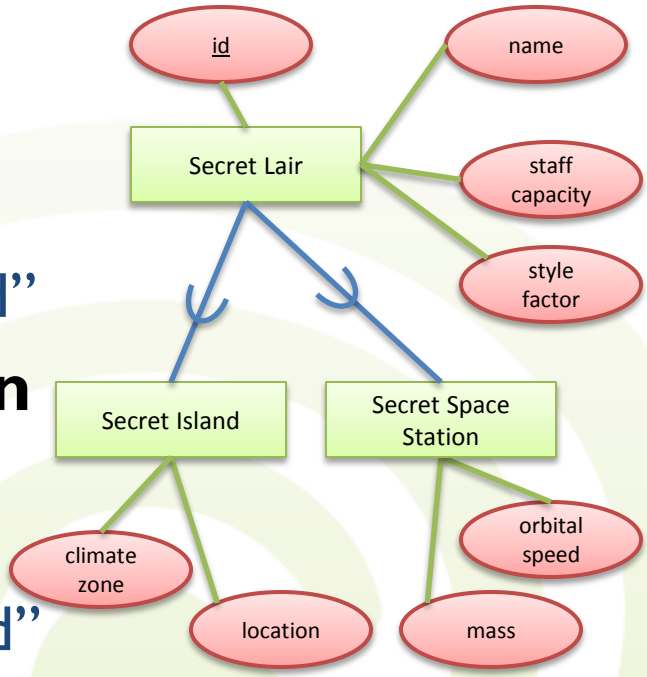
- Possible **implementation: Two distinct** database entries that represent the same instance
  - The same instance appears as a database entry in the superclass and subclass sets, and they are related to each other
  - 1:1 relationship on **entity level**
    - Linking two database entries of the **same entity** in a specialized **role**
  - Often, this solution is easier and more flexible to implement





3.2 Specialization / Generalization

- The process of defining a set of **subclasses** for a superclass is called **specialization**
  - Specialized entity types supplement additional attributes and relationships
  - "Secret lair" can be specialized into "secret space station" and "secret island"
- The inverse process is **generalization**
  - Generalization suppresses differences among specialized subclasses
  - "Secret space station" and "secret island" are generalized to "secret lair"



3.2 Specialization / Generalization

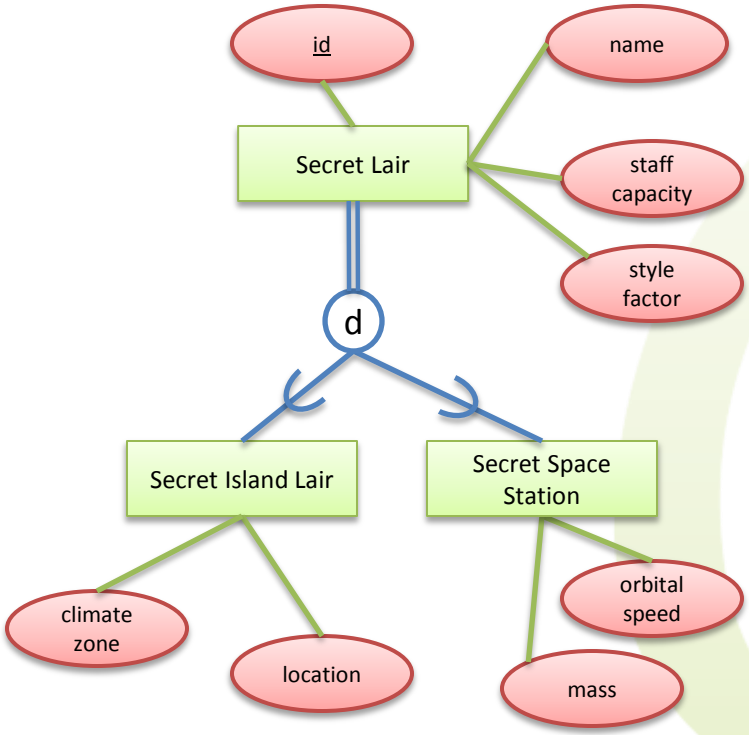
- Specialization and generalization usually result in the **same model**
  - However, the process of how to reach the model is different
  - **Specialization**: **top-down** conceptual refinement
    - Start with super classes, find suitable subclasses
  - **Generalization**: **bottom-up** conceptual synthesis
    - Model sub classes, find proper generalized super class

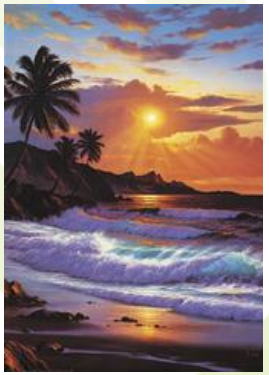
3.2 Constraints on Specialization

- Specializations can be constrained and modeled in further detail regarding two properties
  - **Exclusiveness** (indicated by a labeled circle)
    - **Disjoint:** Subclasses are mutually exclusive (default, label **d**)
    - **Overlapping:** Each entity may be contained in more than one subclass (label **o**)
  - **Completeness**
    - **Total:** No entity is member of the superclass without being member of a subclass (denoted by double line)
    - **Partial:** There are entities that are not contained in any subclass (default)

3.2 Constraints on Specialization

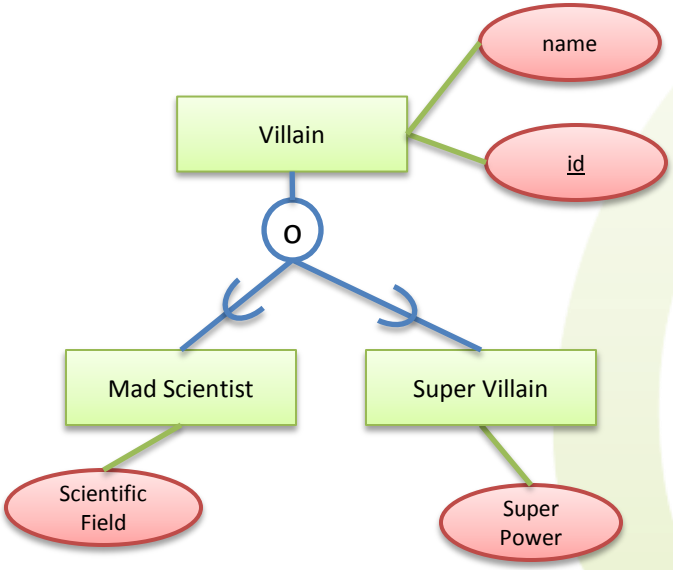
- Examples
  - **Disjoint** and **total:**
    "A secret lair may either be a secret island or a secret space station (but nothing else)."





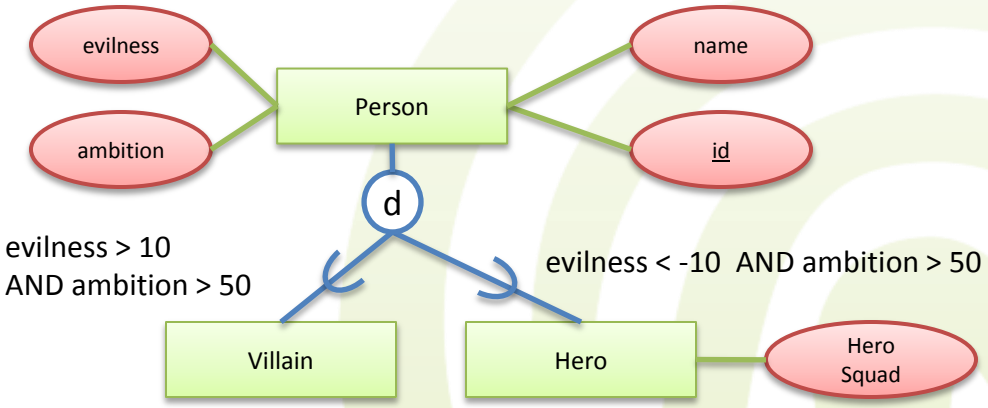
3.2 Constraints on Specialization

- Examples
  - **Overlapping** and **partial:**
    "A villain is a mad scientist, or a super villain, any combination of both, or something else (just a villain)."



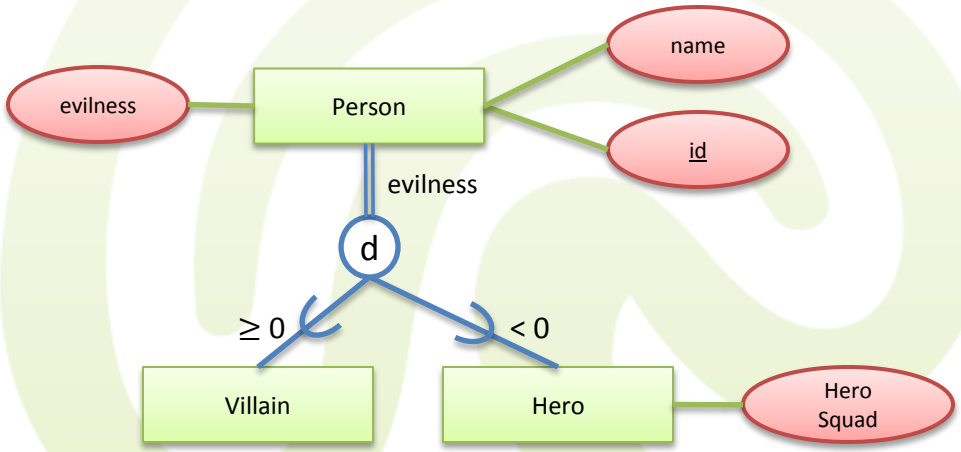
3.2 Constraints on Specialization

- Specializations may be **predicate-defined**
  - A subclass is predicate-defined if there is a predicate (condition) that implies an entity's membership
  - Condition is added to the specialization line
  - Predicate-defined specialization are not necessarily total



3.2 Constraints on Specialization

- Specializations may be **attribute-defined**
  - Attribute-defined is a special case of predicate-defined, where the membership in subclasses depends on a **single attribute value**
  - Attribute is added to line connecting circle and superclass, condition added to lines connecting circle and subclasses



3.2 Constraints on Specialization

- Consequences of specialization
  - **Deleting** an entity from the superclass also deletes it from all subclasses
  - **Inserting** an entity in a superclass automatically inserts it into all matching **predicate-defined** subclasses
  - In a **total** specialization, inserting one entity into a superclass implies that it has to be inserted into **at least one** subclass, too

3.2 Hierarchies and Lattices

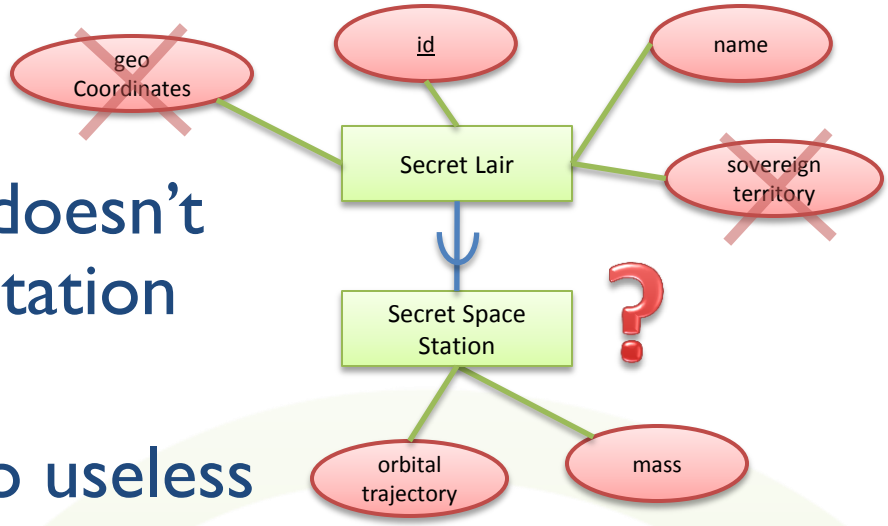
- A subclass may be further specialized
- If every subclass has just **one superclass,** the inheritance structure is a **specialization hierarchy**
- If there are subclasses having **more than one superclass** at the same time, the structure is a **specialization lattice**
  - Shared subclasses possible with multiple inheritance
- Subclasses recursively inherit all attributes and relationships of their superclasses up to the root

3.2 Polymorphism

- **Inheritance** may lead to two special problems
  - Polymorphism
  - Multiple inheritance
- **Polymorphism**
  - Usually, subclasses inherit all attributes and relationships of their supertypes
  - Subtypes may define additional attributes/relationships
  - What happens if an attribute in the subtype means something different?
  - What happens if an attribute is not needed at all?
  - What if some attribute should have a different name?

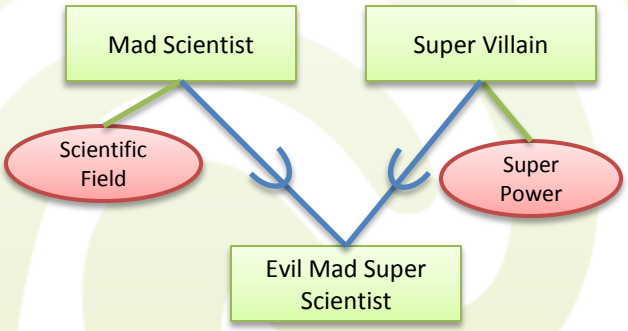
3.2 Polymorphism

- Example:
  - Sovereign territory just doesn't make sense for a space station
    - Should be removed
  - Geo coordinates are also useless
    - But: Orbital trajectory somehow represents the same concept (location)
  - Unfortunately, relational databases and ER don't provide any useful support for polymorphism
    - **Avoid** models where you need it!
    - If it is really necessary, **constraints** and **null-values** may be used to help out…



3.2 Multiple Inheritance

- **Multiple inheritance**
  - A subclass may have multiple superclasses
    - Inheritance lattice instead of inheritance hierarchy
  - But: What happens if superclasses define the same attribute/relationship differently
    - Which one should be inherited?
    - Are both needed?
    - ER provides no support for conflicting multi-inheritance
      - Avoid models with such conflicts

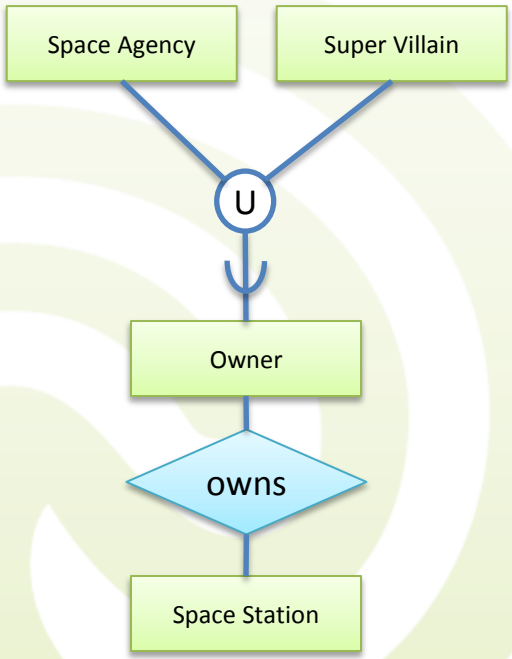


3.2 Union Types

- In a superclass–subclass relationship, the **subclass inherits all** attributes and relationships of the superclass(es)
- However, sometimes it is beneficial that a subclass inherits from only **one superclass** (chosen from a **set** of potential distinct superclasses)
  - Every space station has an owner
  - A space station owner is either a space agency or a super villain

3.2 Union Types

- Solution: **Union types**
  - Denoted by a "u" in a circle
  - Space agency and Super villain are neither related, nor of the same type
  - An owner is **either** a space agency **or** a super villain



3.2 Taxonomies & Ontologies

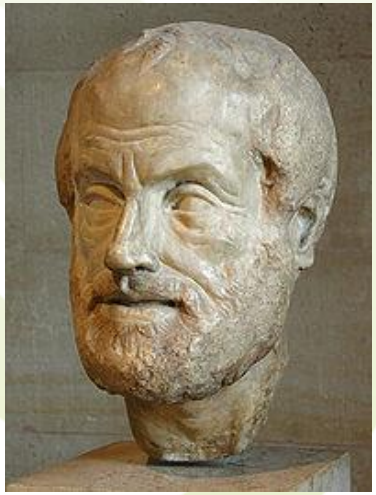
Detour

- Science and philosophy always strived to **explain** the world and the nature of being
  - First formal school of studies: **Aristotle's metaphysics** ("beyond the physical," around 360 BC)
  - Traditional branches of metaphysics
    - **Ontology**
      - Study of being and existence
    - Natural theology
      - Study of God, nature and creation
    - Universal science
      - "First Principles" and logics

3.2 Taxonomies & Ontologies

Detour

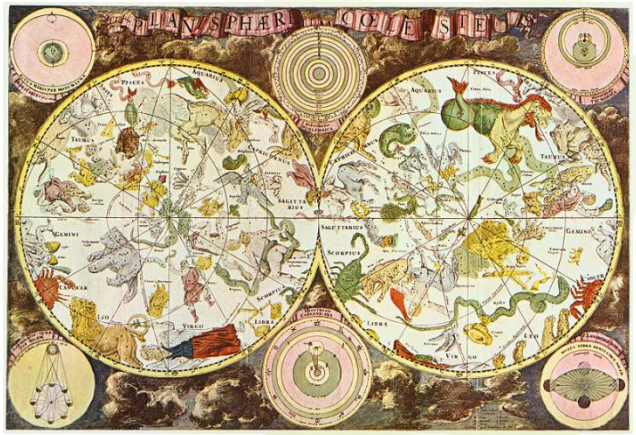
- **Ontology** tries to describe everything which **is** (exists), and its relation and categorization with respect to other things in existence
  - What is **existence**? Which **things** exists? Which are **entities**?
  - Is existence a property?
  - Which **entities** are fundamental?
  - What is a **physical object**?
  - How do the **properties** of an object relate to the object itself? What features are the **essence**?
  - What does it means when a physical object exists?
  - What constitutes the **identity** of an object?
  - When does an object **go out of existence**, as opposed to merely **change**?
  - Why does anything exist rather than nothing?



3.2 Taxonomies & Ontologies

Detour

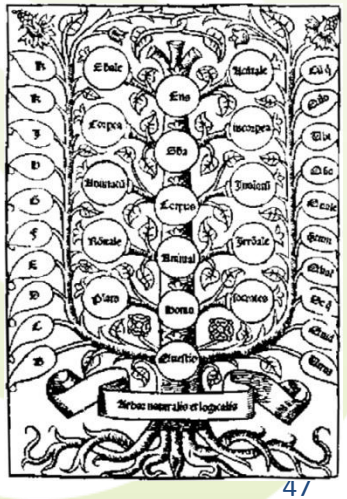
- Parts of metaphysics evolved into **natural philosophy**
  - Study of **nature** and the **physical universe**
  - In the late 18th century, it became just "**science**"
  - Ontology is still a dominant concept in science
    - Representation of all knowledge about things



3.2 Taxonomies & Ontologies

Detour

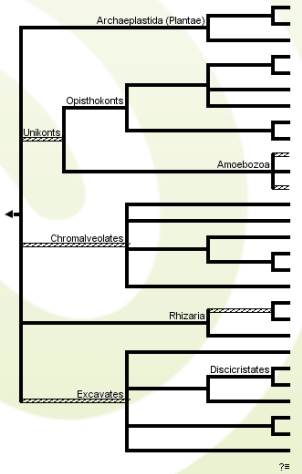
- **Ars Generalis Ultima**
  - Created in 1305 by Ramon Llull
  - "Ultimate" solution for the **Ars Magna** (Great Art)
    - Mechanical combination of terms to **create knowledge**
    - Base hope: all facts and truths can be created in such a way
  - Heavy use of Arbor Scientiae (**Tree of Knowledge**)
    - Tree structure showing an hierarchy of philosophical concepts
    - Together with various "machines" (paper circles, charts, etc.) **reasoning** was possible



3.2 Taxonomies & Ontologies

Detour

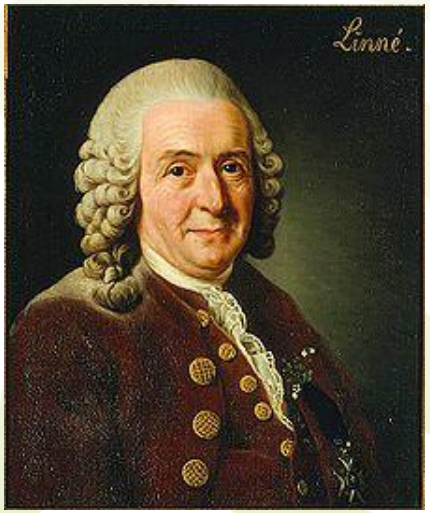
- **Taxonomies** (τάξις : arrangement) are part of ontology
  - Groups things with similar properties into **taxa**
  - Taxa are put into an **hierarchical structure**
    - Hierarchy represents supertype–subtype relationships
    - Represents a **specialization** of taxa, starting with the most general one
  - Taxonomies can be modeled with ER using specialization hierarchies
    - Taxa are represented by entity types

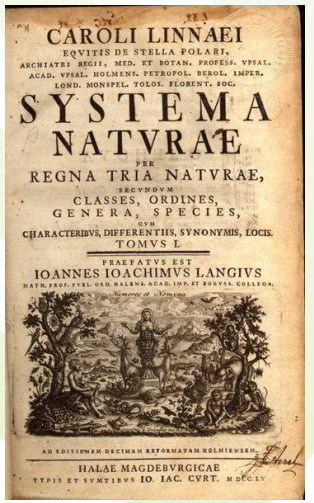


3.2 Taxonomies

Detour

- Example: **Linnaean Taxonomy**
  - Classification of all living things by Carl von Linné in 1738
  - Classification into multiple hierarchy layers
    - Domain, Kingdom, Phylum, Subphylum, Class, Cohort, Order, Suborder, Infraorder, Superfamily, Family, Genus, Species
  - Each layer adds additional properties and restrictions

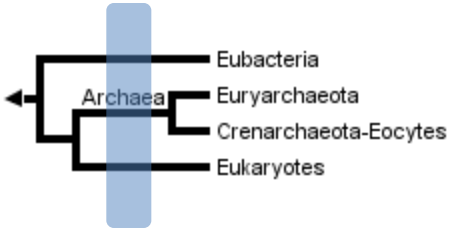




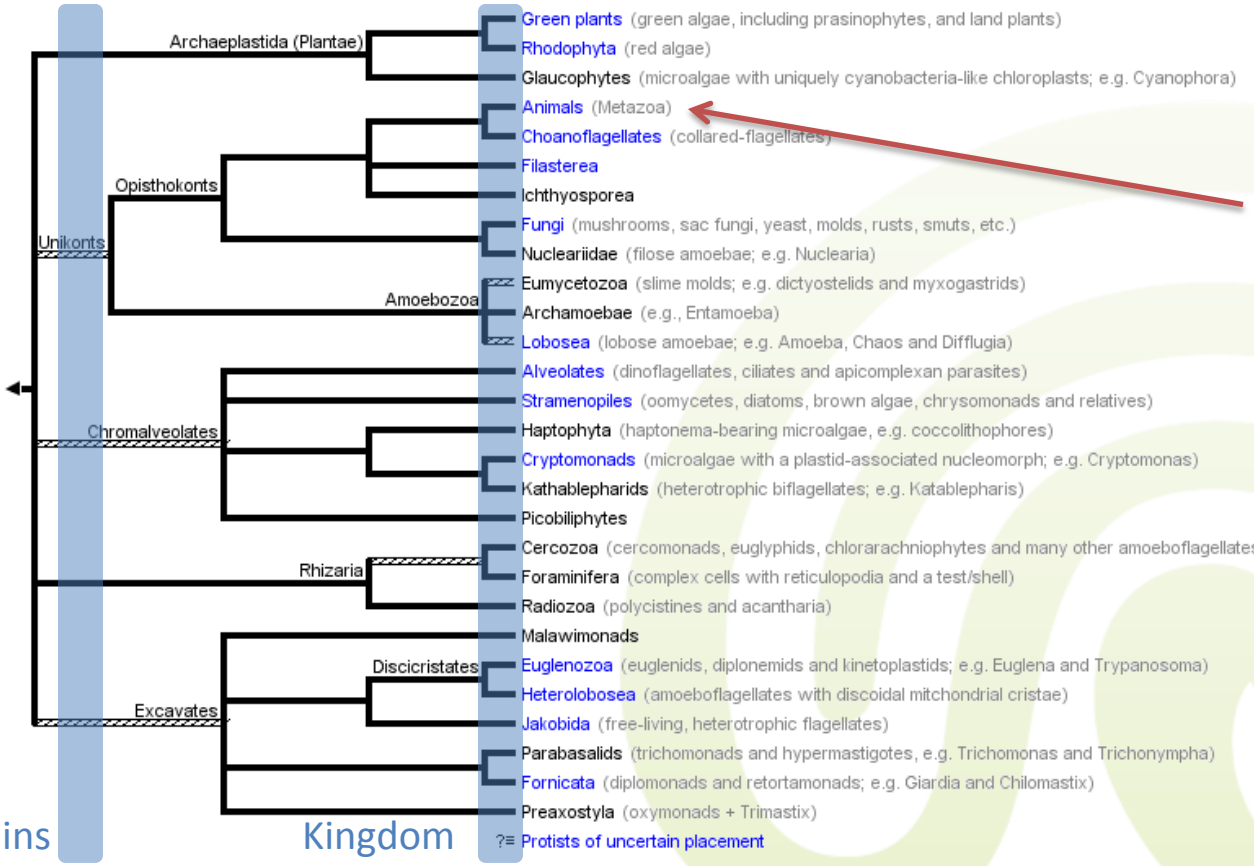
3.2 Taxonomies

Detour

- Domain: **Eukaryotes**
  - Organisms having cell membranes



Domain



3.2 Taxonomies

- Example: **Red Squirrel**
  (Binomial Name: Tamiasciurus hudsonicus)
  - Kingdom**: Animals**
  - Phylum**: Chordata** (with ***backbone***)
  - Class: **Mammalia** (with backbone, ***nursing its young***)
  - Order: **Rodentia** (backbone, nursing its young, ***sharp front teeth***)
  - Suborder: **Scriuomorpha (**backbone, nursing its young, sharp front teeth, ***like squirrel***)
  - Family: **Scriudae** (backbone, nursing its young, sharp front teeth, like squirrel, ***bushy tail & lives on trees (i.e. real squirrel)***)
  - Genus: **Tamiasciurus** (backbone, nursing its young, sharp front teeth, like squirrel, bushy tail & trees, ***from N-America***)
  - Species: **Hudsonicus** (backbone, nursing its young, sharp front teeth, like squirrel, bushy tail & trees, from N-America, ***brown fur with white belly***)

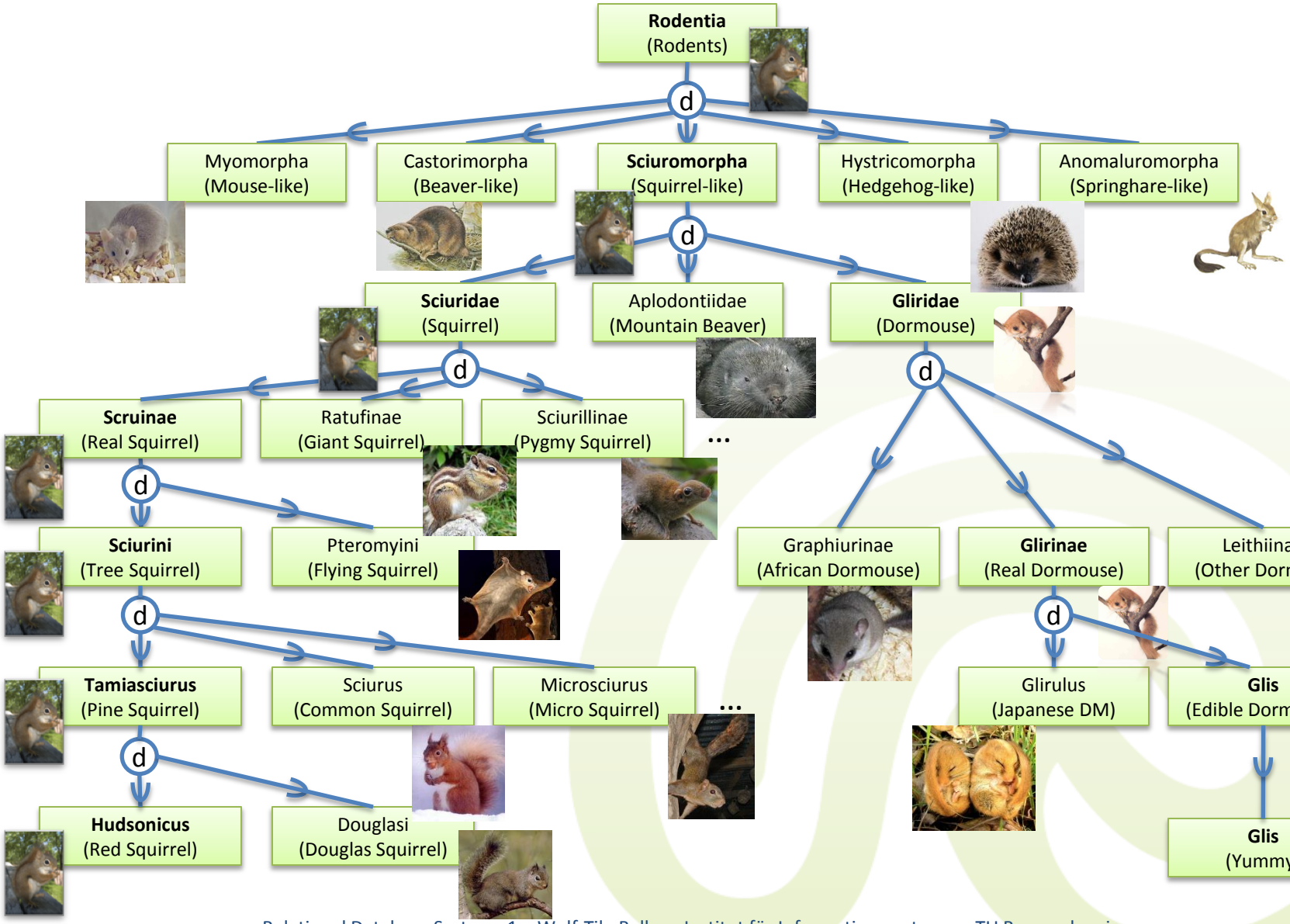
3.2 Taxonomies

Detour

- Example: **Edible Dormouse**
  (Binomial Name: Glis Glis)
  - Kingdom**: Animals**
  - Phylum**: Chordata** (with ***backbone***)
  - Class: **Mammalia** (with backbone, ***nursing its young***)
  - Order: **Rodentia** (backbone, nursing its young, ***sharp front teeth***)
  - Suborder: **Scriuomorpha (**backbone, nursing its young, sharp front teeth, ***like squirrel***)
  - Family: **Gliradae** (backbone, nursing its young, sharp front teeth, like squirrel, ***sleeps long***)
  - Genus: **Glis** (backbone, nursing its young, sharp front teeth, bushy tail, like squirrel, ***eaten by Romans***)
  - Species: **Glis** (backbone, nursing its young, sharp front teeth, bushy tail, climbs trees, ***nothing more to classify***)

3.2 Taxonomies

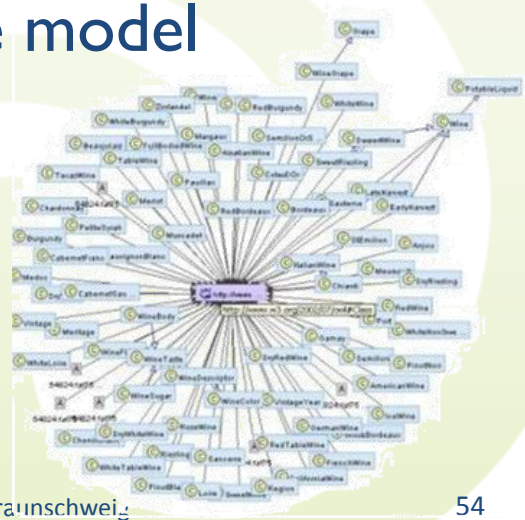
Detour



3.2 Ontologies in CS

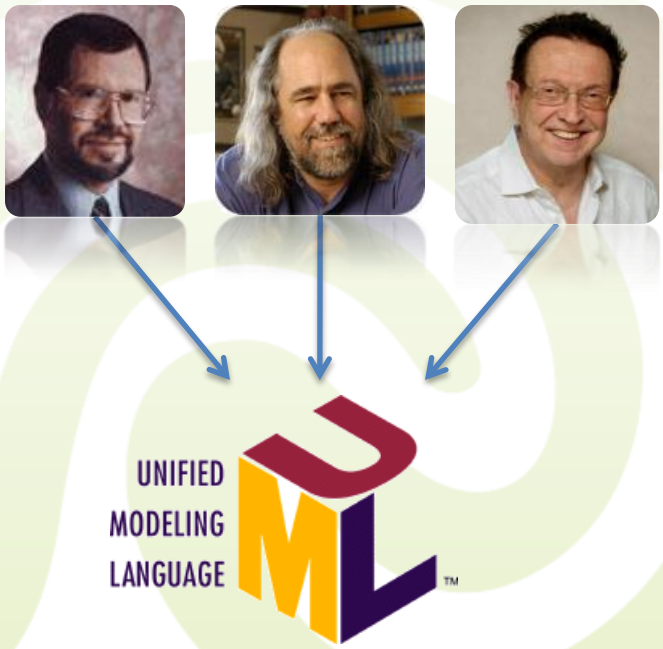
Detour

- Recently, creating **ontological models** became fashionable in CS
  - So called **ontologies**
  - Widely used in medical informatics, bio-informatics, Semantic Web, etc.
- In addition to "normal" data models, ontologies offer **reasoning capabilities**
  - Allow to classify instances automatically
  - Allow to extract additional facts from the model
- In CS, ontologies are usually modeled using **special languages**
  - OWL, DAML+OIL, IDEF, …



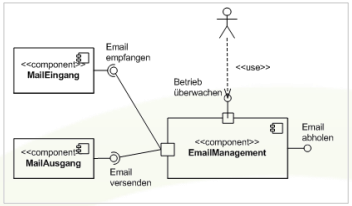
3.3 UML

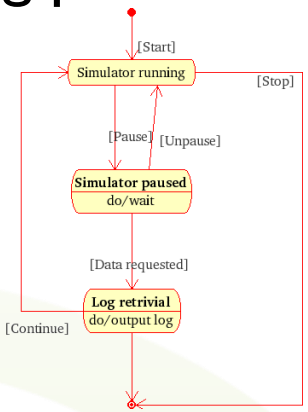
- UML (**Unified Modeling Language**) is a set of multiple modeling languages and diagram types
  - First standardized in 1997
  - Unification of dominating object-oriented software design methods
    - James Rumbaugh: OMT
    - Grady Booch: Boochs Method
    - Ivar Jacobsen: OOSE

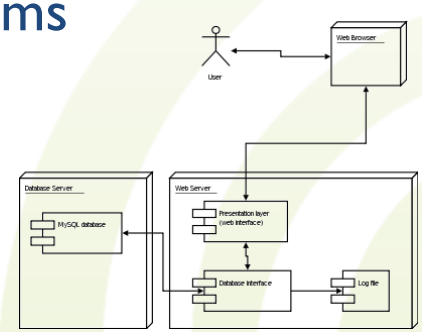


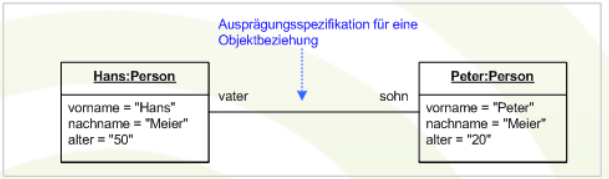
3.3 UML

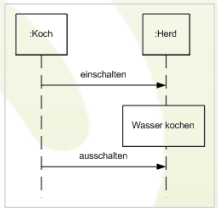
- UML provides support for various software modeling problems
  - Static structural diagrams
    - **Class diagram**
    - Component diagram
    - Deployment diagram
    - Composite structure diagram
    - Object diagram
    - Package diagram
  - Dynamic behavior diagrams
    - Activity diagram
    - State diagram
    - Use-case diagram
  - Interaction diagrams
    - Communication diagram
    - Sequence diagram
    - Timing diagram
    - Interaction overview diagram

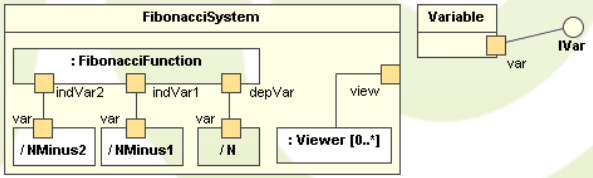






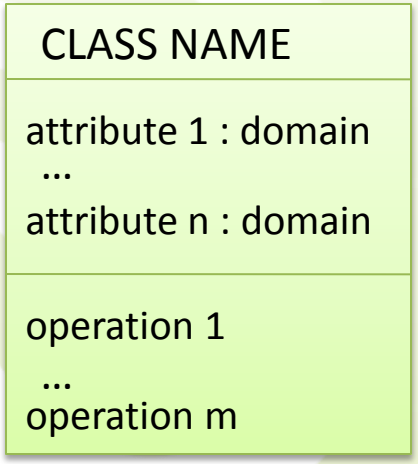






3.3 UML

- For data modeling, only **class diagrams** are used
  - Closely related to ER diagrams in crow's foot notation
    - Additional notations for logical design and operations
- Entity type becomes **class**
  - Attributes written as in crow's foot notation
    - Usually, also domains are modeled
  - Operations are usually not needed in pure data models
  - Entity type instances are called **objects**

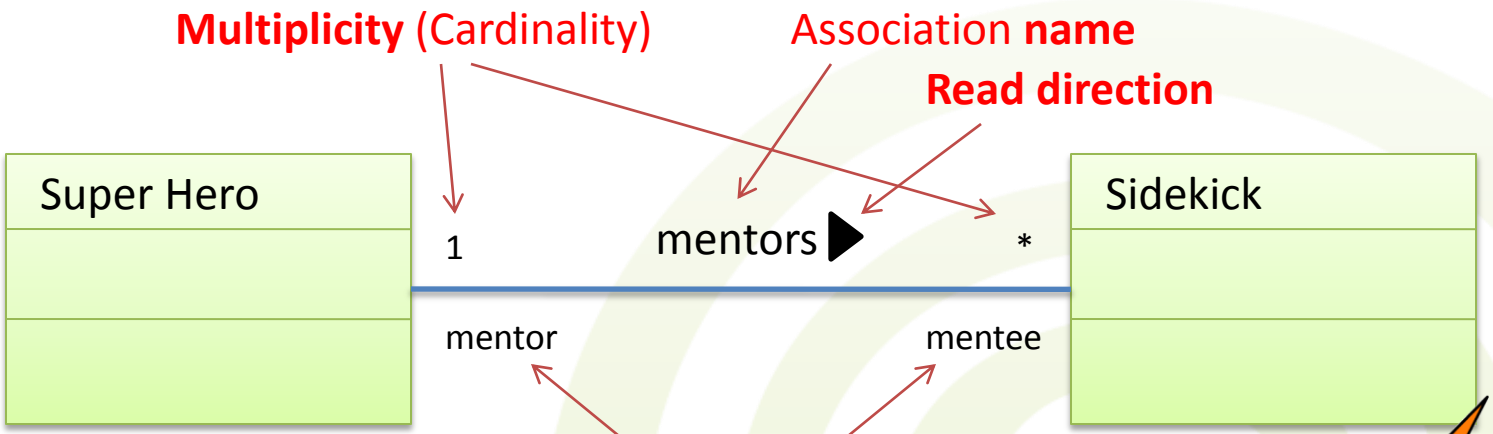


3.3 UML

- In UML, relationship types are called **associations**
- Simplest case: just a plain **line**
  - Although using just a line is valid, a good model should provide additional information
    - Name
    - Direction
    - Multiplicity
    - Order
    - Navigability
    - Special Aggregation Types

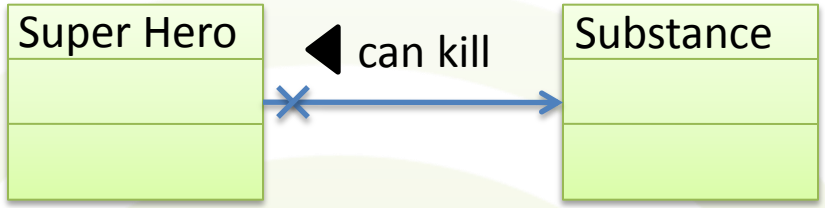
3.3 UML

- Better:
  "A super hero may mentor multiple sidekicks"
  - Careful: Multiplicity in opposite direction to Chen ER



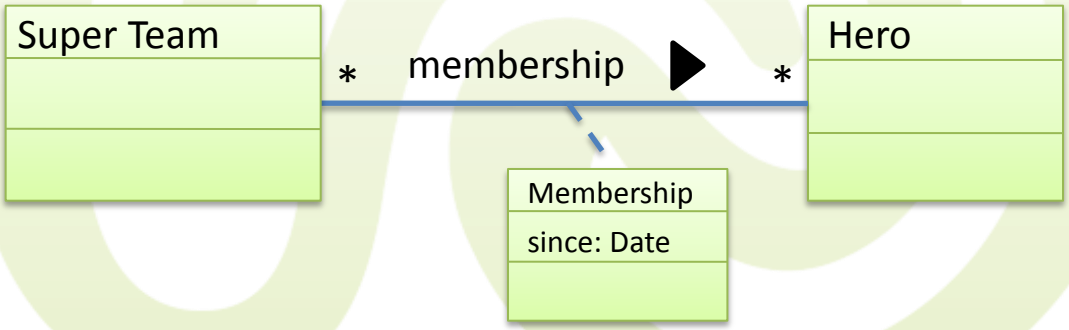
3.3 UML

- Association **navigability**
  - Denoted by an arrowhead and small cross
  - Models how you can navigate among objects involved in the association
  - One-way association
  - Example:
    - For each hero, you can navigate to the substances which may kill him
    - You cannot natively navigate from a substance to a hero



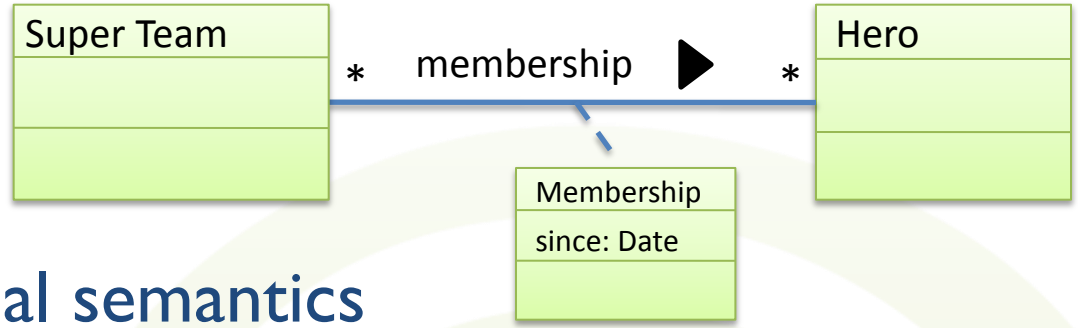
3.3 UML

- UML does not allow to add attributes to associations directly
- Workaround: **Association classes**
  - Association classes belong to an association (indicated by dashed line)
  - They share the association name
  - Each instance of the association creates an according class object

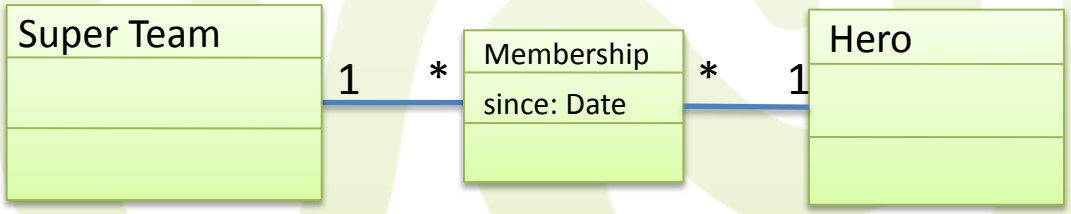


3.3 UML

- Association classes cannot directly be replaced by a “normal” class

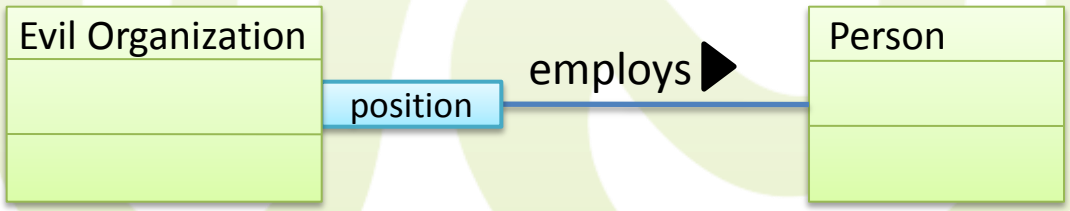


  - Introduces additional semantics
  - The replacement model allows that a hero is assigned **twice** to the same super team!



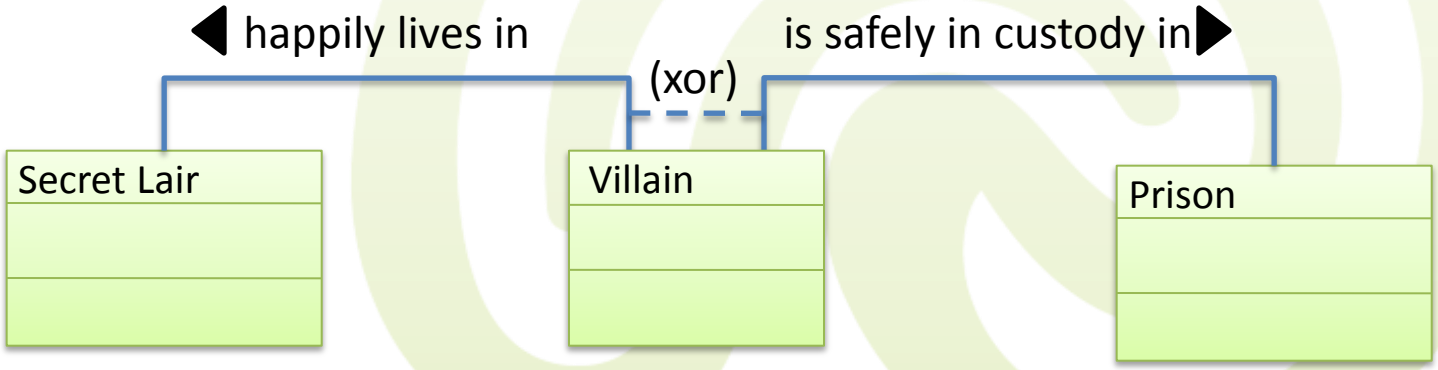
3.3 UML

- **Qualified associations**
  - Associations may be qualified by an additional attribute
    - Each association instance between objects is **classified** by this attribute
    - Semantically stronger as a classifier than an attribute of an association class
      - Within a programming language, the qualification attribute would end up as a key of a (hash) map
  - Example: "Von Doom Industries employs Victor von Doom as CEO"



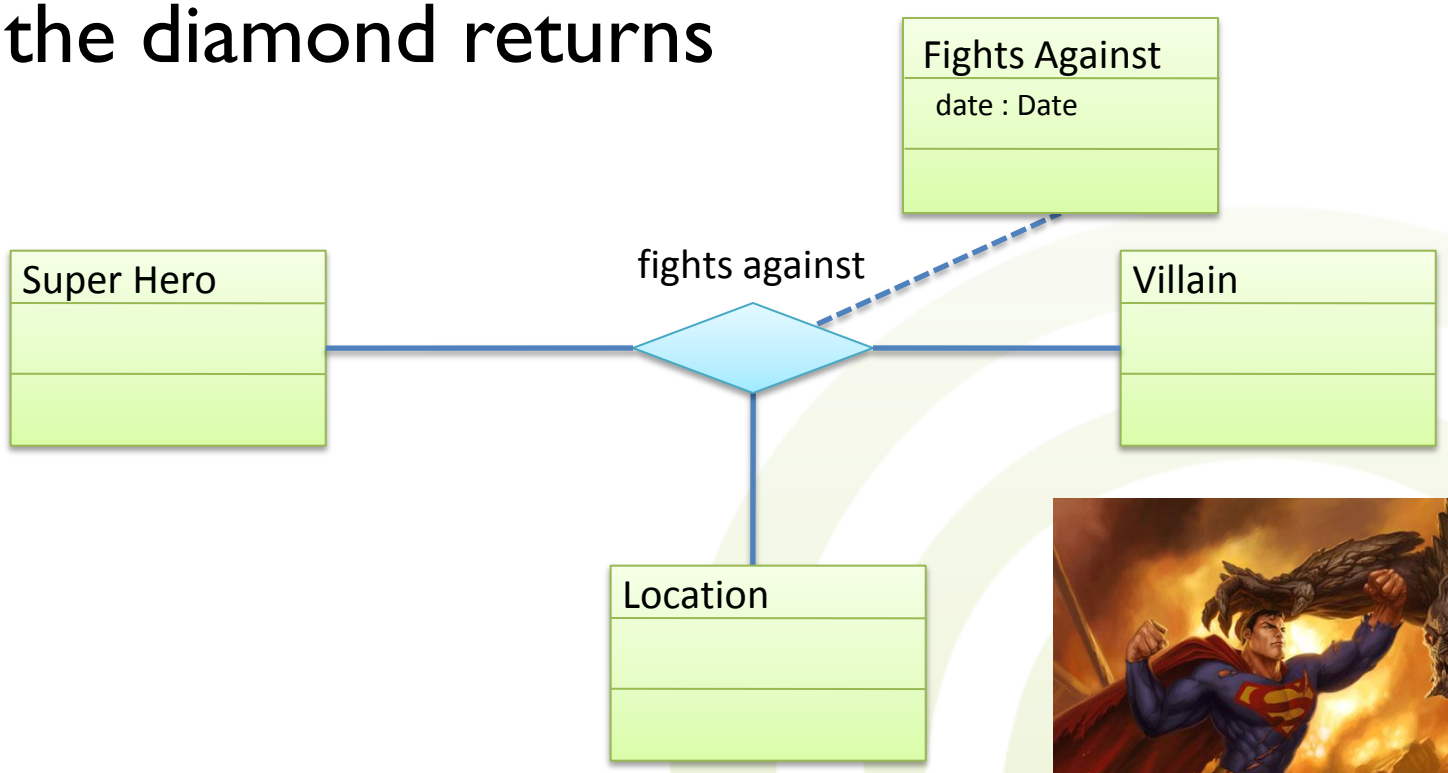
3.3 UML

- **XOR restrictions** on associations
  - A class having multiple associations can be modeled in such a way that **only one** of these **associations** can be active **at a time**
  - Example: "A villain lives either in a secret lair, or in a prison (but not in both)."



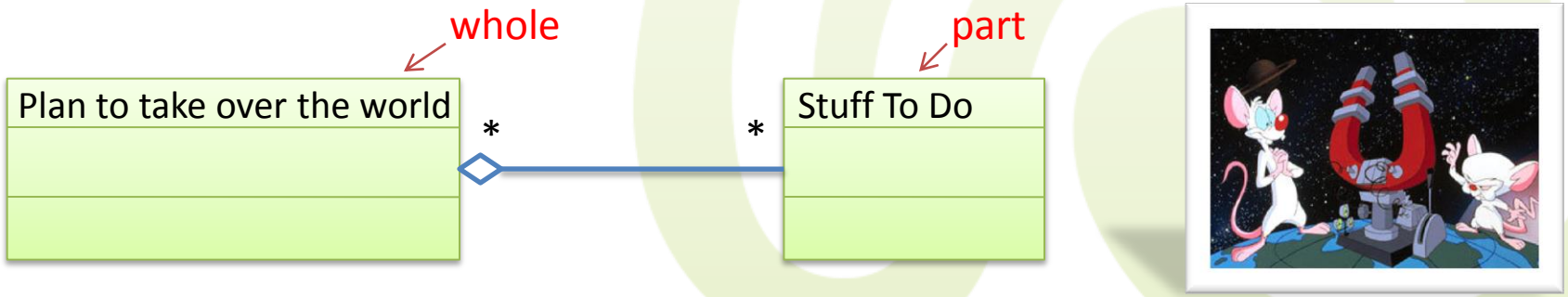
3.3 UML

- For ***n*-ary associations** ($n > 2$), the diamond returns



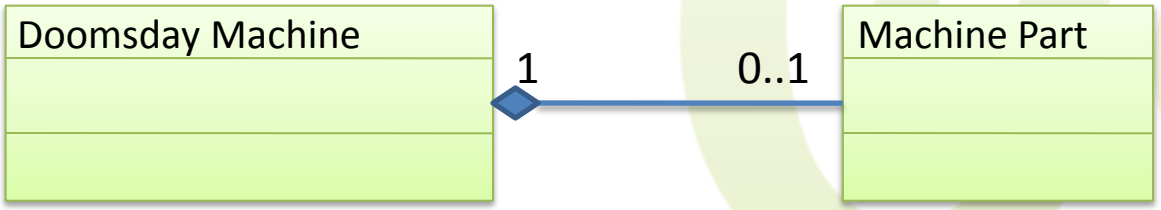
3.3 UML

- **Aggregation**
  - The aggregation is a special association within UML
  - Colloquial: "**is part of**" or "consist of"
  - Denoted by a small, empty diamond
  - Aggregation just states that one class is part of another; it poses no further restrictions
    - Objects may still exist independently of each other
    - Objects may be part of several other objects
  - Example: "A plan to take over the world consists of several things that need to be done."



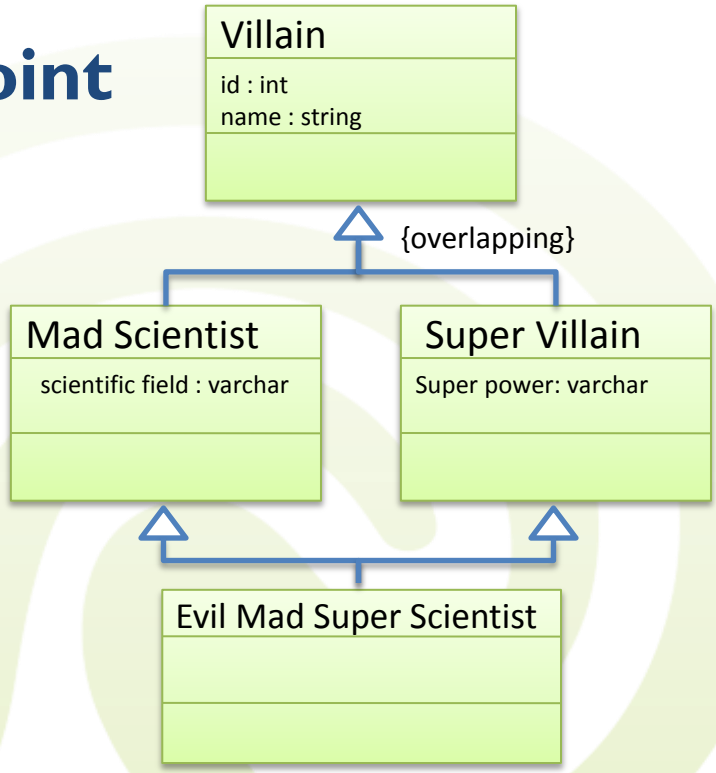
3.3 UML

- **Composition** (also called strong aggregation)
  - **Stricter** version of aggregation
    - Diagrammed by solid diamond
  - Based on multiplicity of the part-side
    - **1**: An object is **always part** of just **one** other object.
      If the "main" object is deleted, the part needs to be assigned to another "master" or is deleted.
    - **0..1**: An object may be part of **at most one** other object.
      May also exist alone.
    - * : Not allowed. Part of one object max.
  - Example: "A doomsday machine is made of multiple parts"



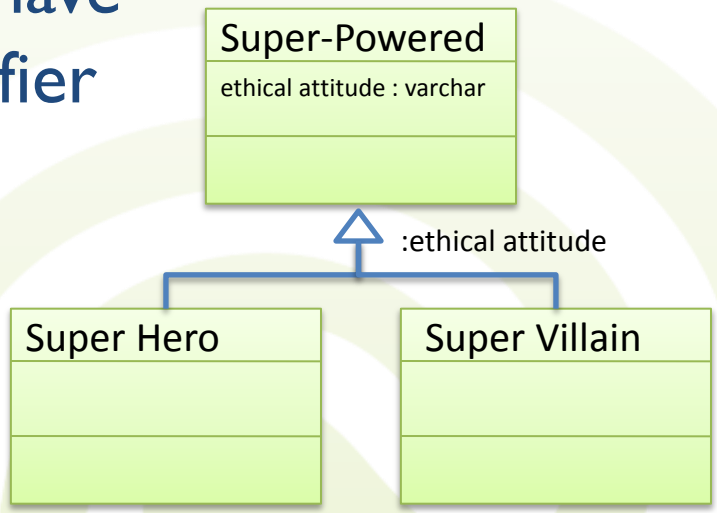
3.3 UML

- **Generalization**
  - Induces a class-subclass relationship (**"is-a"**)
    - Diagrammed with an hollow arrow
  - By default, generalization is **disjoint**
    - **Overlapping** is additionally annotated in curly brackets
  - By default, generalization is partial (**incomplete** in UML)
    - Total (**complete**) is also annotated in curly brackets



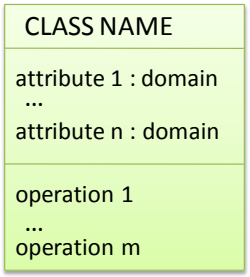
3.3 UML

- **Classification attributes**
  - Similar to EER's **attribute-defined** relationship types
  - Denoted by ":attributeName"
  - All objects of a given subtype have the **same value** for the classifier attribute

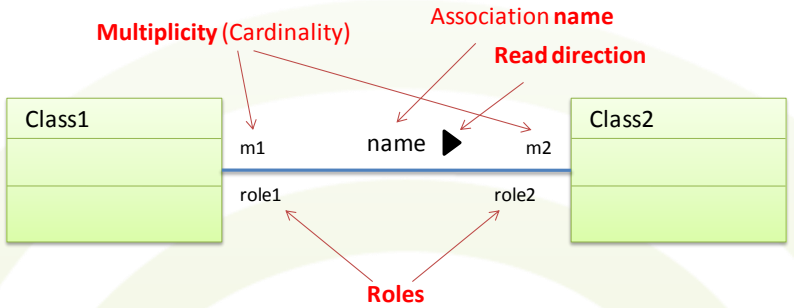


3.3 UML: Summary

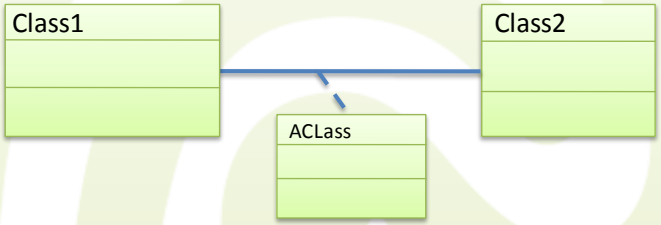
- **Class** (entity type)



- **Associations** (relationship types)

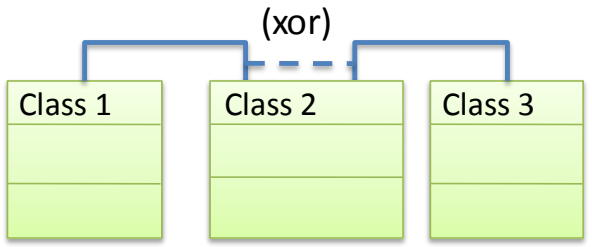


- **Association class**

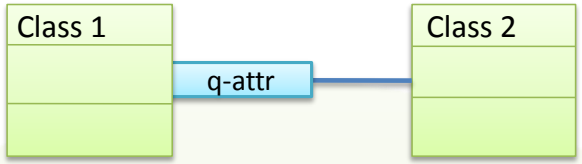


3.3 UML: Summary

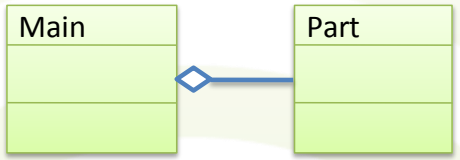
- **XOR restriction**



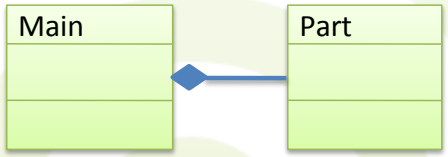
- **Qualifying attribute**



- **Aggregation**



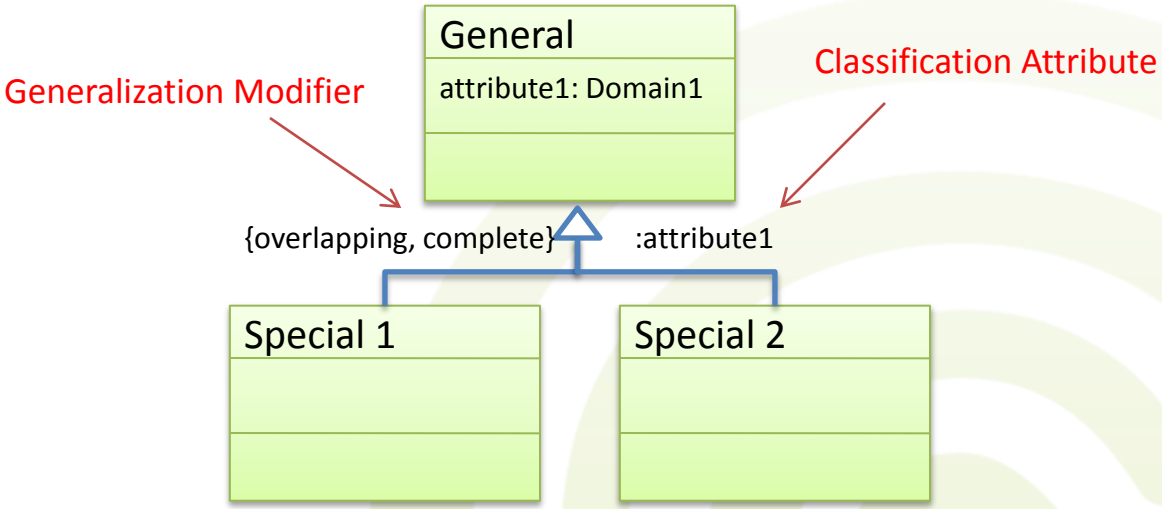
- **Composition**



- **Navigable association**

3.3 UML: Summary

- **Generalization / Specialization**



3.4 Next Week

- View integration
- Resolving conceptual incompatibility
- Entity clustering for ER models
- Commercial dimension:
  The BEA story