ifis

Institut für Informationssysteme
Technische Universität Braunschweig

Relational Database Systems I

Wolf-Tilo Balke
Niklas Kiehne, Enrique Pinto Dominguez
Institut für Informationssysteme
Technische Universität Braunschweig
http://www.ifis.cs.tu-bs.de

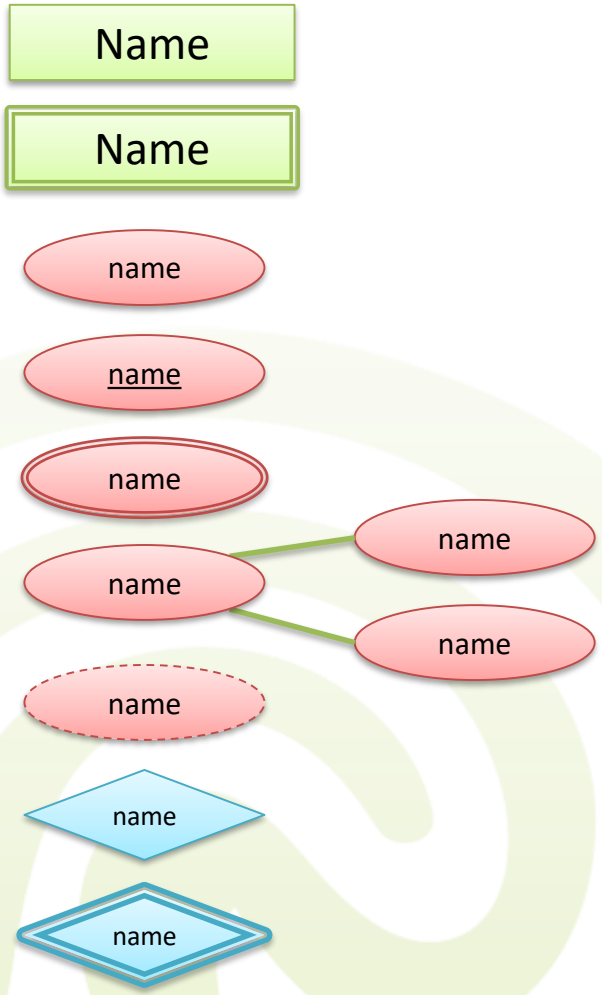
Summary last week

- **Data models** define the structural constrains and possible manipulations of data
  - Examples of Data Models:
    - Relational Model, Network Model, Object Model, etc.
  - Instances of data models are called **schemas**
    - Careful: Often, sloppy language is used where people call a schema also a model
- We have three types of schemas:
  - **Conceptual Schemas**
  - **Logical Schemas**
  - **Physical Schemas**
- We can use ER modeling for conceptual and logical schemas

Summary last week

Summary

- Entity Type
- Weak Entity Type
- Attribute
- Key Attribute
- Multi-valued Attribute
- Composite Attribute
- Derived Attribute
- Relationship Type
- Identifying Relationship Type



Summary last week

Summary

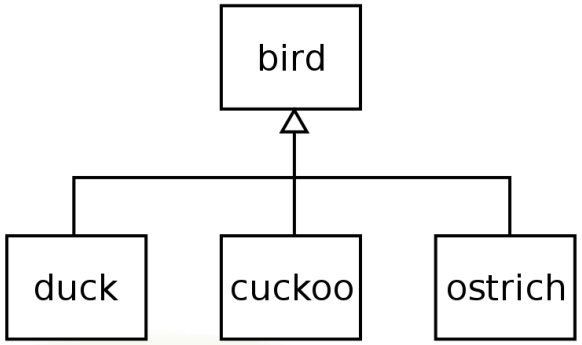
- Total participation of E2 in R

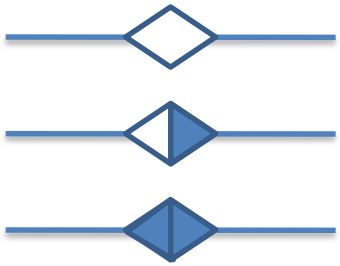
- Cardinality
  - an instance of E1 may relate to multiple instances of E2

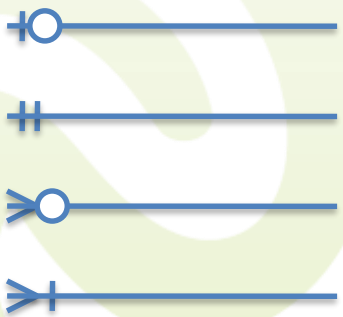
- Specific cardinality with min and max
  - an instance of E1 may relate to multiple instances of E2

3 Extended Data Modeling

- **Alternative ER Notations**
- Extended ER
  - Inheritance
  - Complex Relationships
- Taxonomies & Ontologies
- 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

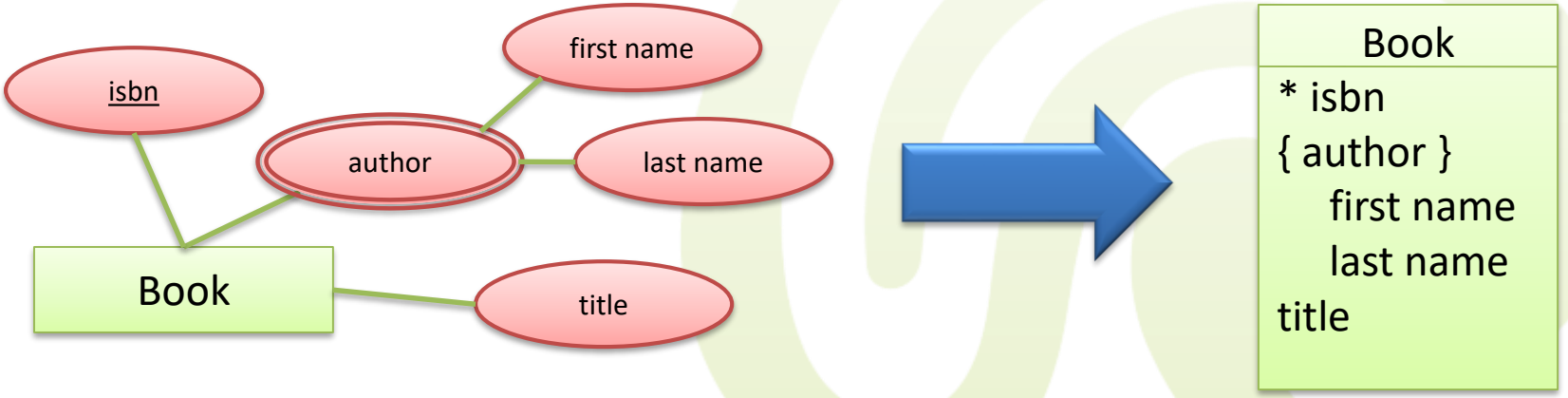
- In the following, we introduce the (somewhat popular) crow's foot notation

3.1 ER – Crow's Foot Notation

- **Crow's foot** notation was initially developed by Gordon Everest
  - derivate of extended entity relationship notation
  - main goal
    - consolidate graphical representation
    - provide a better and faster overview
    - allow for easier layouting
  - widespread use in many current tools and documentations

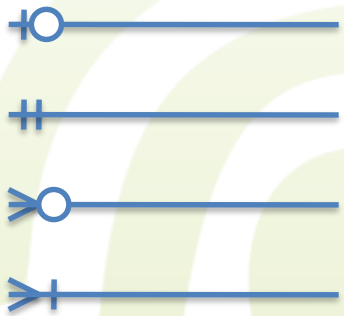
3.1 ER – Crow's Foot Notation

- **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



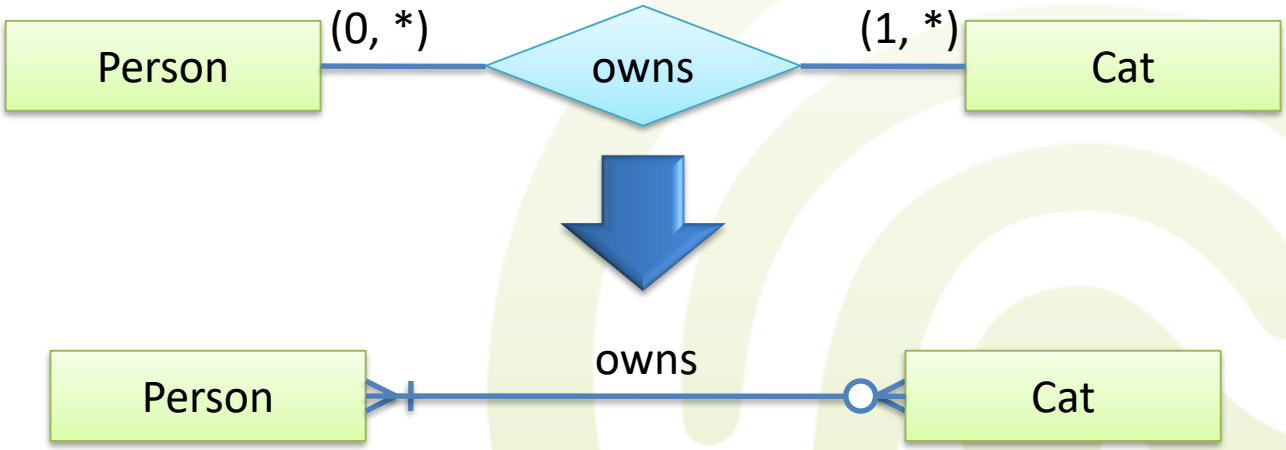
3.1 ER – Crow's Foot Notation

- **Relationship Types**
  - relationship types are modeled by lines connecting the entities (no explicit symbol for relationships)
  - 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



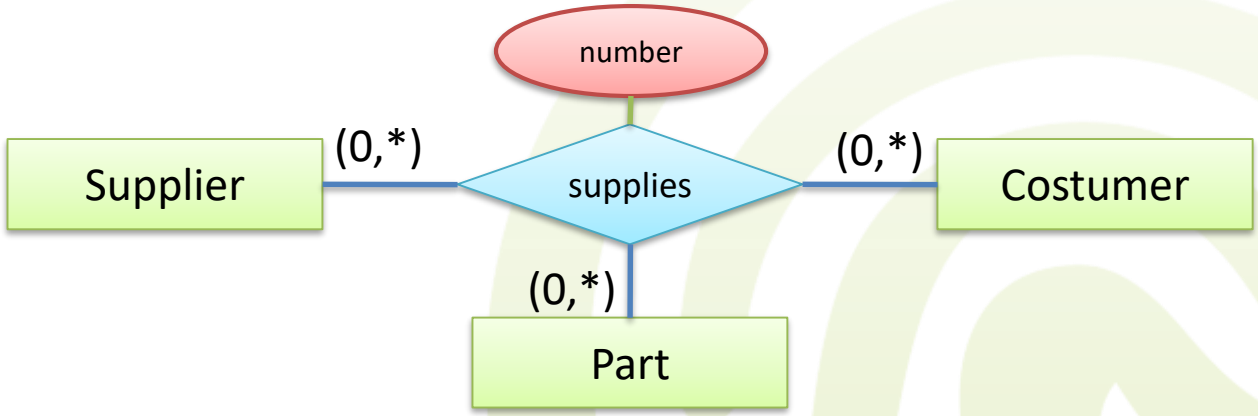
3.1 ER – Crow's Foot Notation

- **Attention:**
  - Cardinalities are written on the **opposite side** of the relationship (in contrast to *Chen notation*)



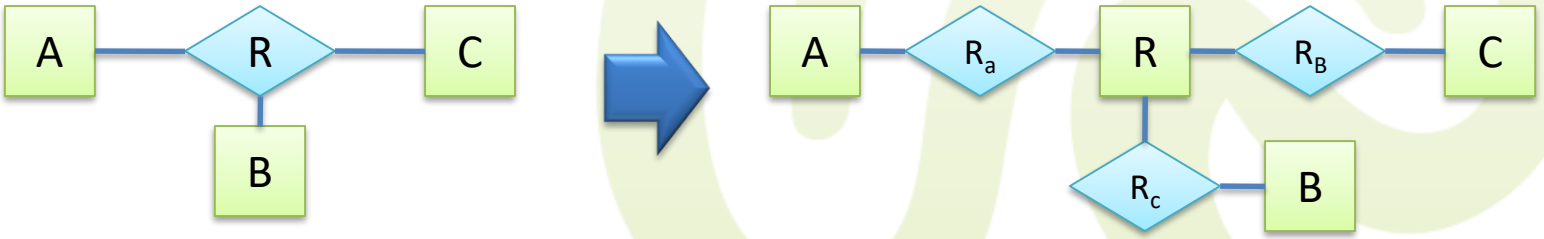
3.1 ER – Crow's Foot Notation

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

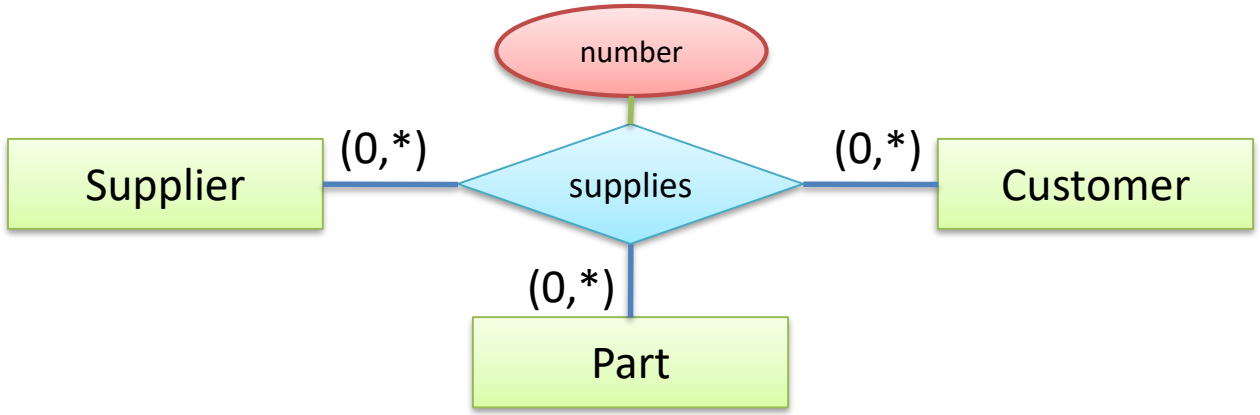


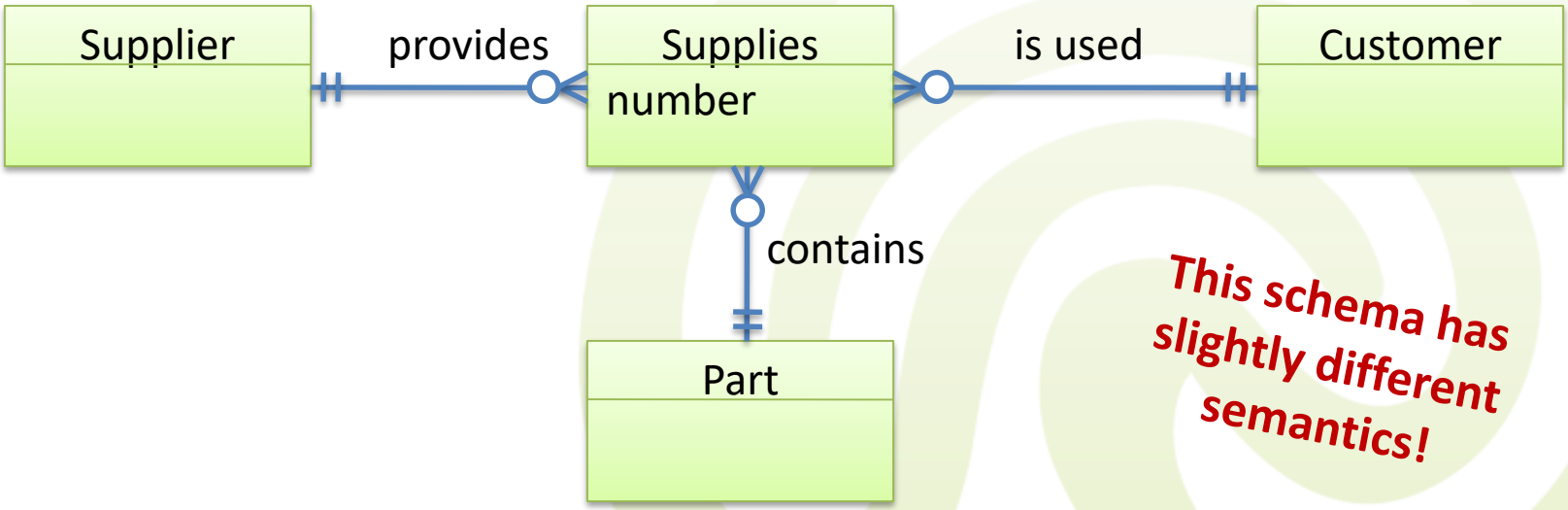
3.1 ER – Crow's Foot Notation

- **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





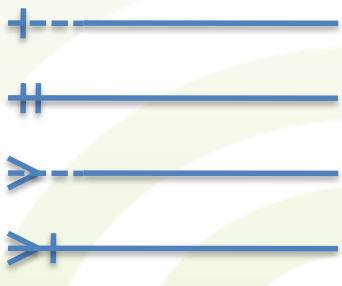
This schema has slightly different semantics!

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 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 in 1986
  - 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

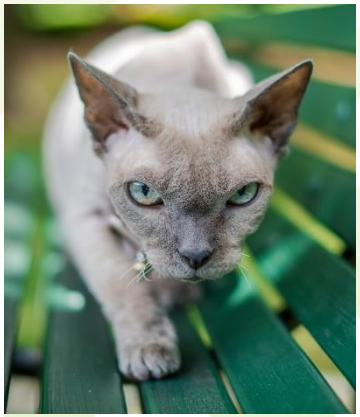
3.1 ER – Even more notations…

- **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 Extended Data Modeling

- Alternative ER Notations
- **Extended ER**
  - Inheritance
  - Complex Relationships
- Taxonomies & Ontologies
- UML

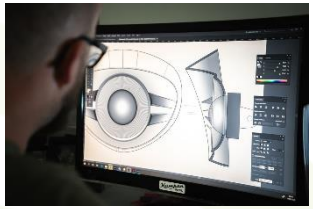


3.2 Extended Data Modeling

- 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

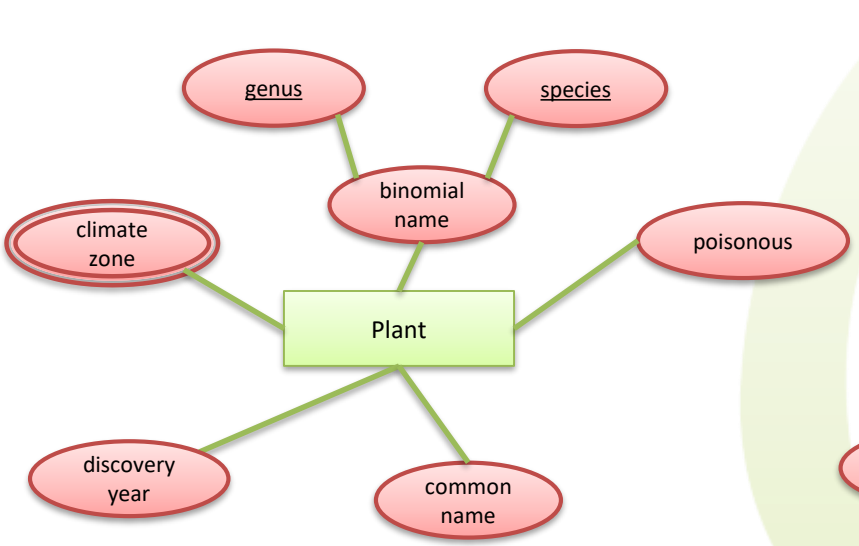


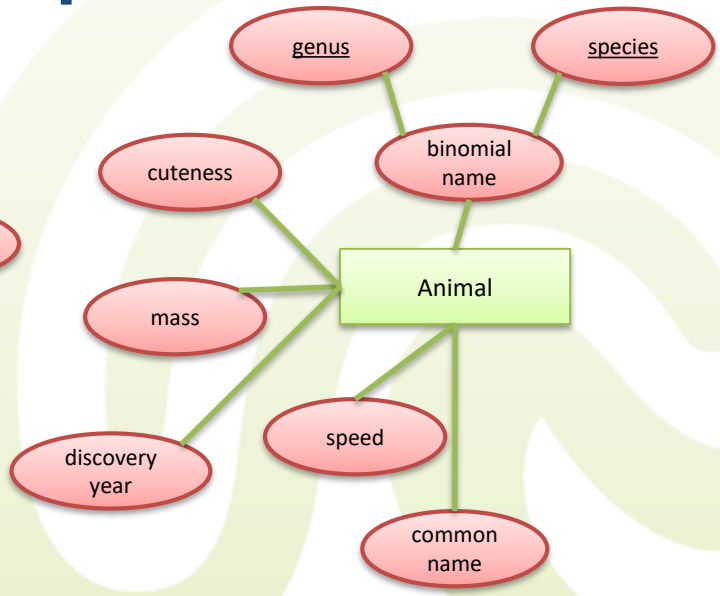
3.2 Extended Data Modeling

- 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
  - **also, most modern object-oriented programming languages use similar modelling semantics**
- Extended ER contains all features of *classic* ER

3.2 Extended Data Modeling

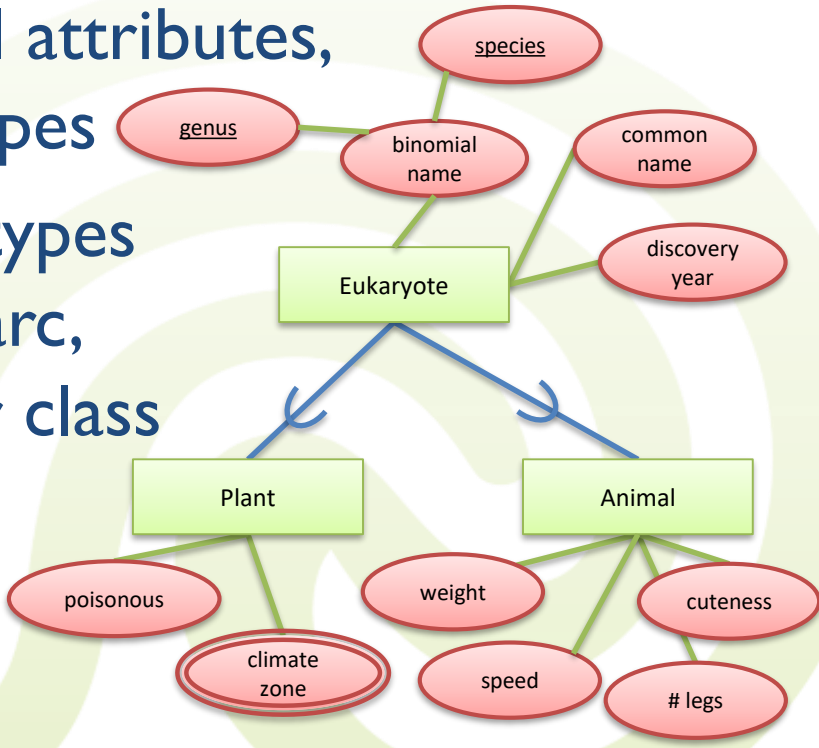
- Problem
  - model **eukaryotes** living in allotment patches
  - **plants** and **animals** are special kinds of eukaryotes, **share many attributes,** but still need some **unique attributes**





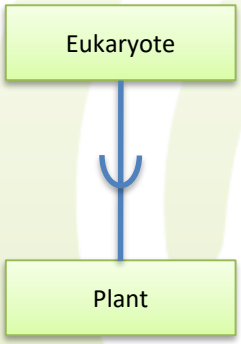
3.2 Subclasses / Superclasses

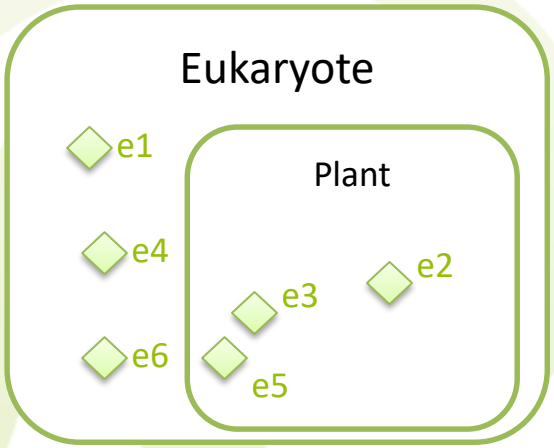
- Solution: **subclasses** and **superclasses**
- A **subclass** entity type **inherits** all attributes and constraints from its **superclass** entity type
  - subclasses may add additional attributes, constraints or relationship types
  - in EER, subclass relationship types are annotated with an open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



3.2 Subclasses / Superclasses

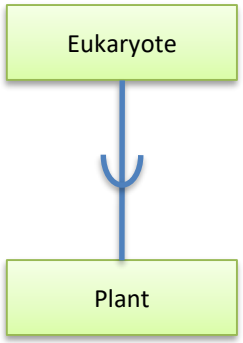
- **Subclass entity types** represent subsets of the entity set of the superclass's entity type
  - i.e. an entity which is contained in the subclass is also contained in the superclass
  - In particular, no entity can **only** exist in a subclass set

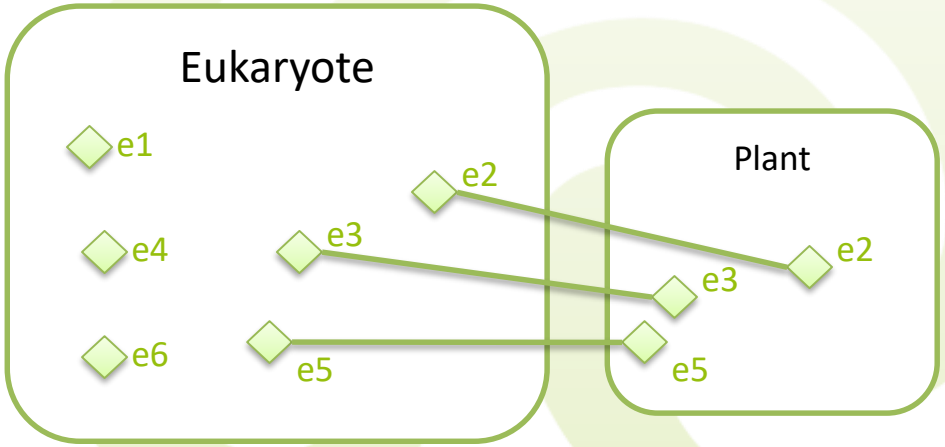




3.2 Subclasses / Superclasses

- Possible **implementation: two distinct** database entries that represent the same entity
  - Implementation ≡ logical schema
  - 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
  - *Eukaryote* can be specialized into *Animal* and *Plant*
- The inverse process is **generalization**
  - generalization suppresses differences among specialized subclasses
  - *Animal* and *Plant* are generalized to *Eukaryote*

3.2 Specialization / Generalization

- Specialization and generalization may result in the **same model**
  - however, the process of how to reach the model is different
  - **specialization: top-down** conceptual refinement
    - start with superclasses, find suitable subclasses
  - **generalization: bottom-up** conceptual synthesis
    - model subclasses, find proper generalized superclass

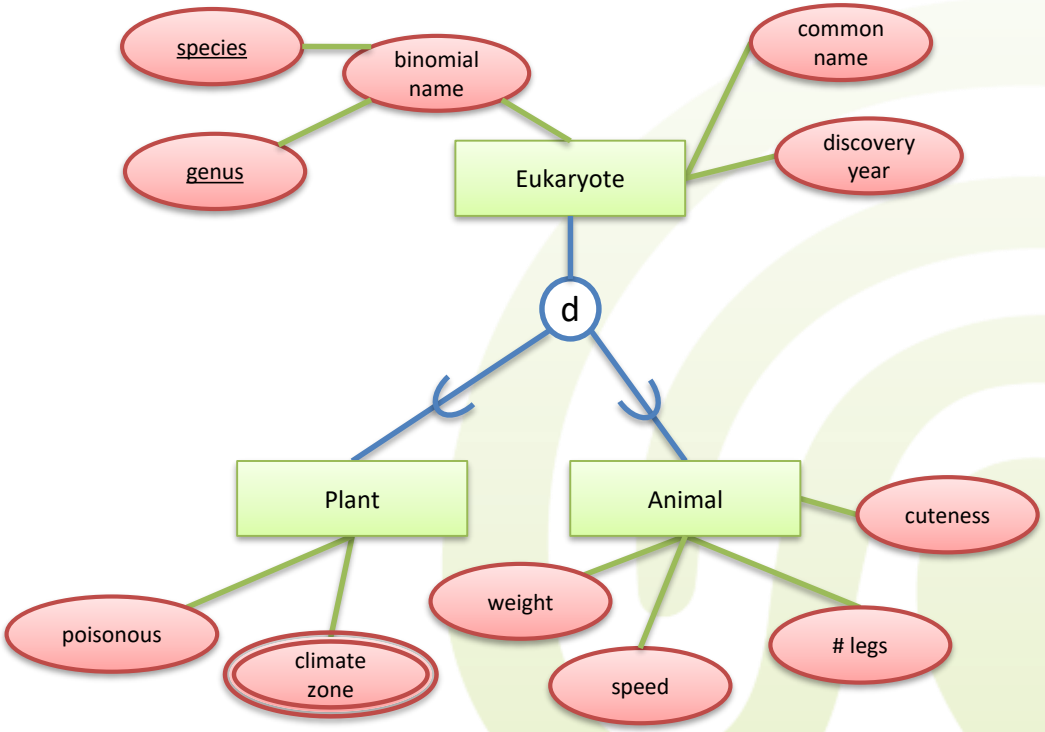
3.2 Constraints on Specialization

- 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

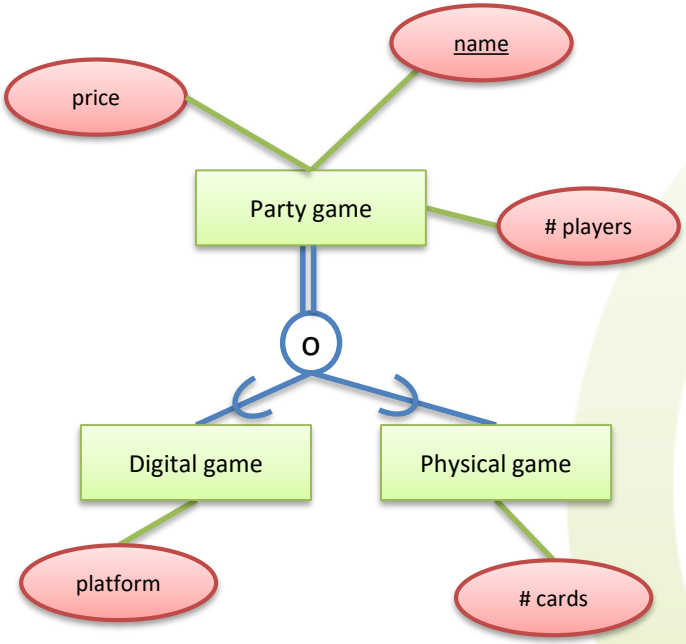
- Examples
  - **disjoint** and **partial:**

    An eukaryote may be an animal, a plant or neither (there are other eukaryotes that are not modelled), but not both.



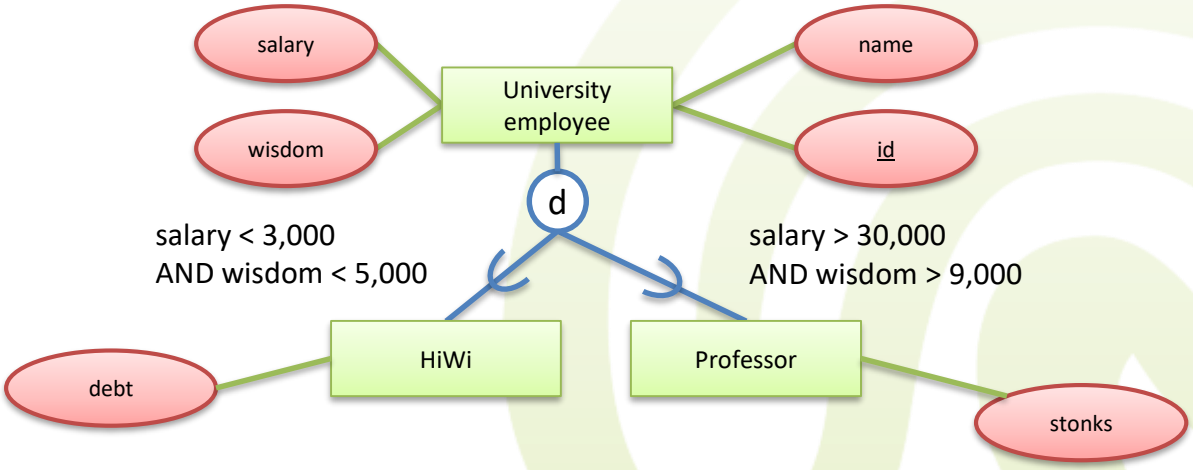
3.2 Constraints on Specialization

- Examples
  - **overlapping** and **total:**
    A party game like UNO must be digital, physical or both



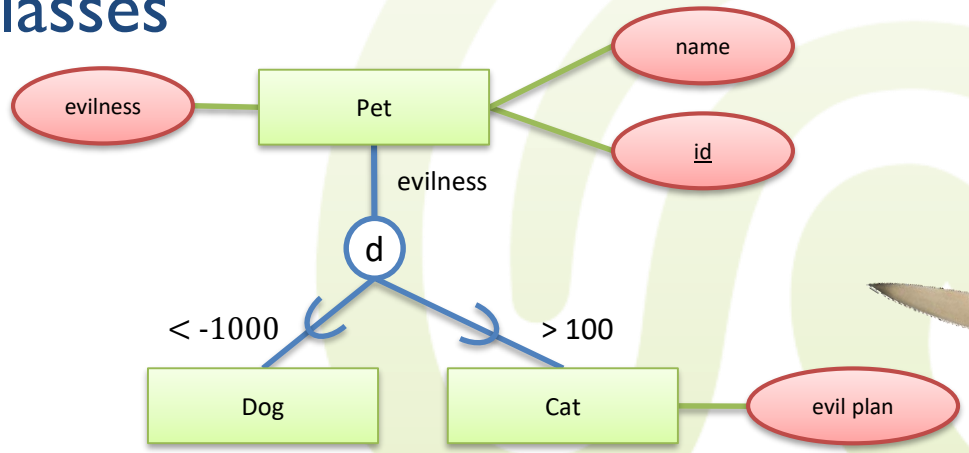
3.2 Constraints on Specialization

- 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
    - Deleting only from subclass has no clear semantics
  - **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

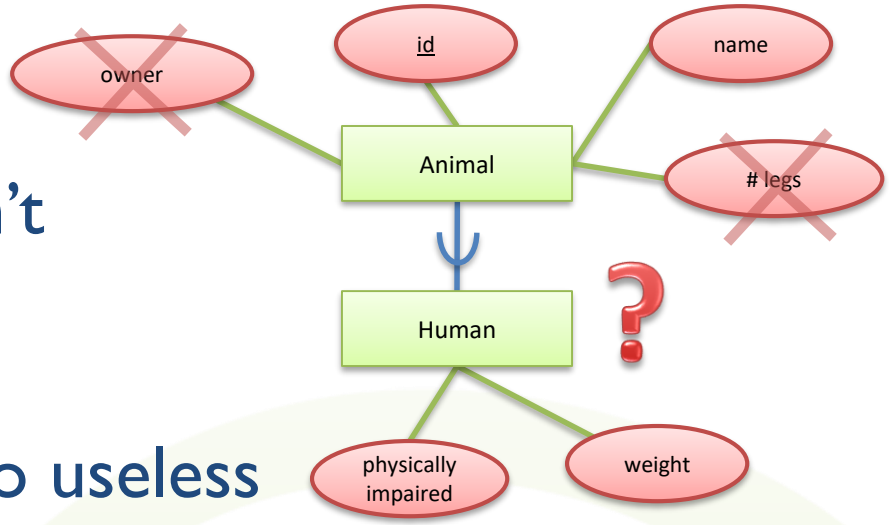
3.2 Polymorphism

- **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?

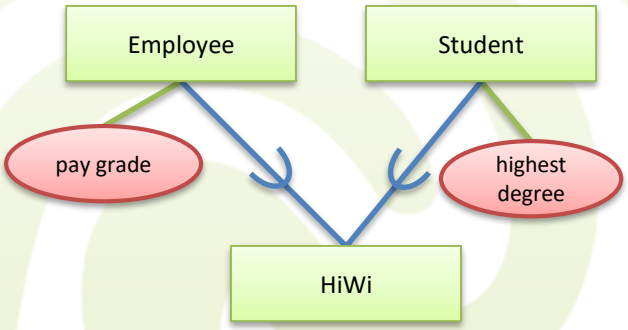
3.2 Polymorphism

- Example
  - storing the owner doesn't make sense for a human
    - should be removed
  - the number of legs is also useless
    - **but:** knowing about a physical impairment (including e.g. a missing leg) would aid in ensuring accessibility and fairness
  - unfortunately, relational databases and ER don't provide any useful support for polymorphism
    - **avoid** schemas 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

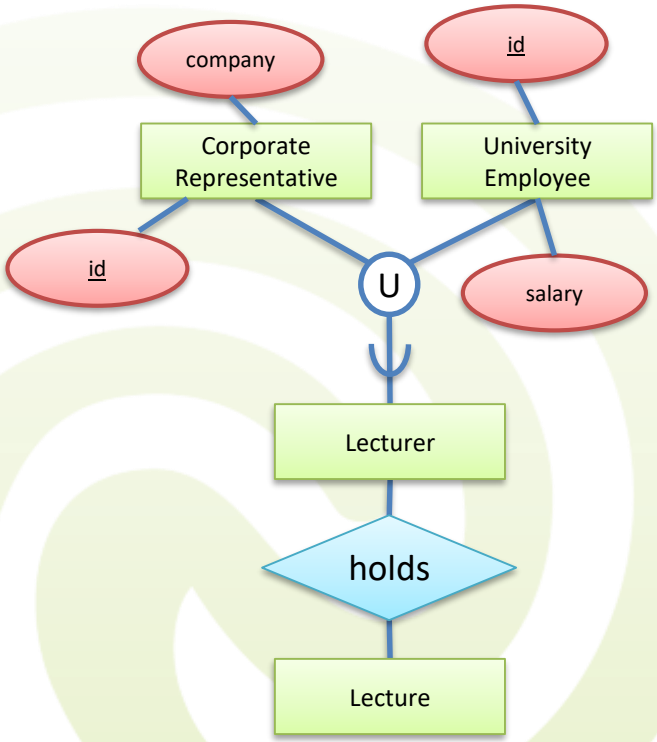


3.2 Union Types

- 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)
  - each lecture is held by a lecturer
  - the lecturer is either a university employee or a corporate representative

3.2 Union Types

- Solution: **union types**
  - Denoted by a *u* in a circle
  - *Guest Lecturer* and *University Employee* are not related
  - A Lecturer is **either** a *Corporate Representative* **or** a *University Employee*



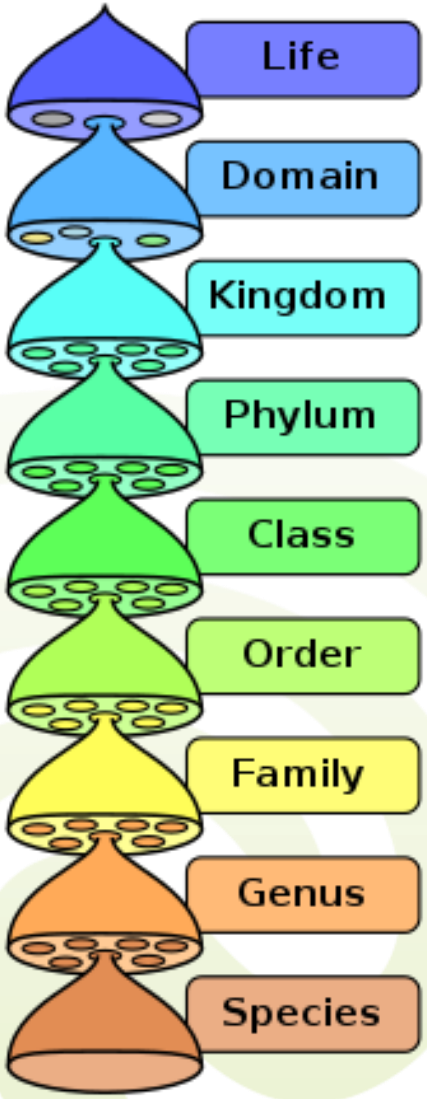
Quick Exercise

- Another database about persons and HiWis
  - We have persons with a first name and last name
  - People can also be HiWis, which have any number of nicknames and any number of expertises
  - Expertises have a name and can be technological, pedagocical or magical
    - …and any combination of it

Quick Exercise

3 Extended Data Modeling

- Alternative ER Notations
- Extended ER
  - Inheritance
  - Complex Relationships
- **Taxonomies & Ontologies**
- UML



3.3 Taxonomies & Ontologies

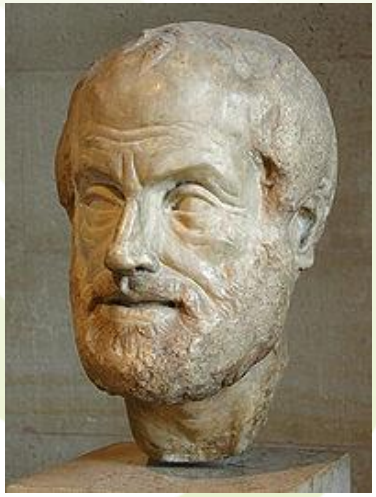
Detour

- 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

3.3 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 mean 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?



3.3 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

3.3 Taxonomies & Ontologies

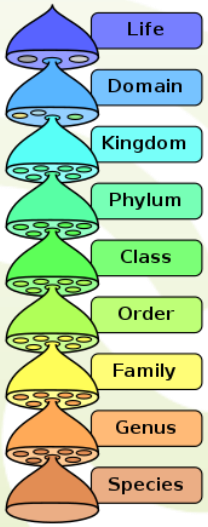
Detour

- **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 a hierarchy of philosophical concepts
    - together with various *machines* (paper circles, charts, etc.) **reasoning** was possible

3.3 Taxonomies & Ontologies

Detour

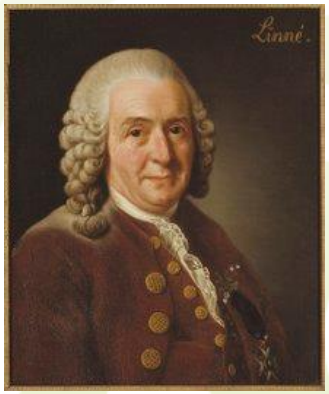
- **Taxonomies** (τάξις : arrangement) are part of ontology
  - groups things with similar properties into **taxa**
  - taxa are put into an **hierarchical structure**
    - hierarchy represents supertype-subtype relationships
    - represents a **specialization** of taxa, starting with the most general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

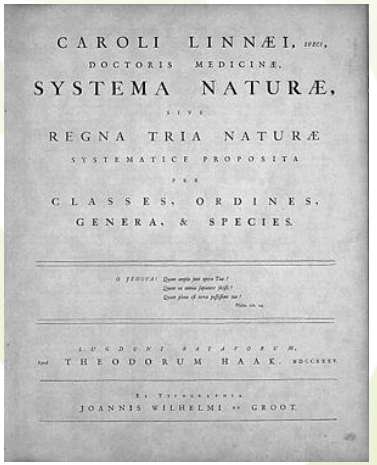


3.3 Taxonomies

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

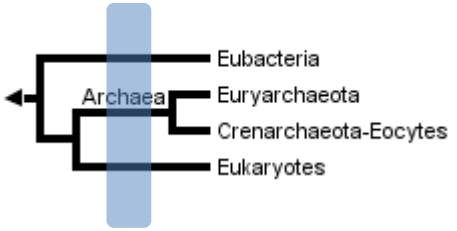



CAROLI LINNÆI,
DOCTORIS MEDICINÆ,
SYSTEMA NATURÆ,
REGNA TRIA NATURÆ
SYSTEMATICE PROPOSITA
CLASSES, ORDINES,
GENERA, & SPECIES.
LUGDUNI BATAVORUM,
THEODORUM HAAK.
JOANNIS WILHELMI DE GROOT.

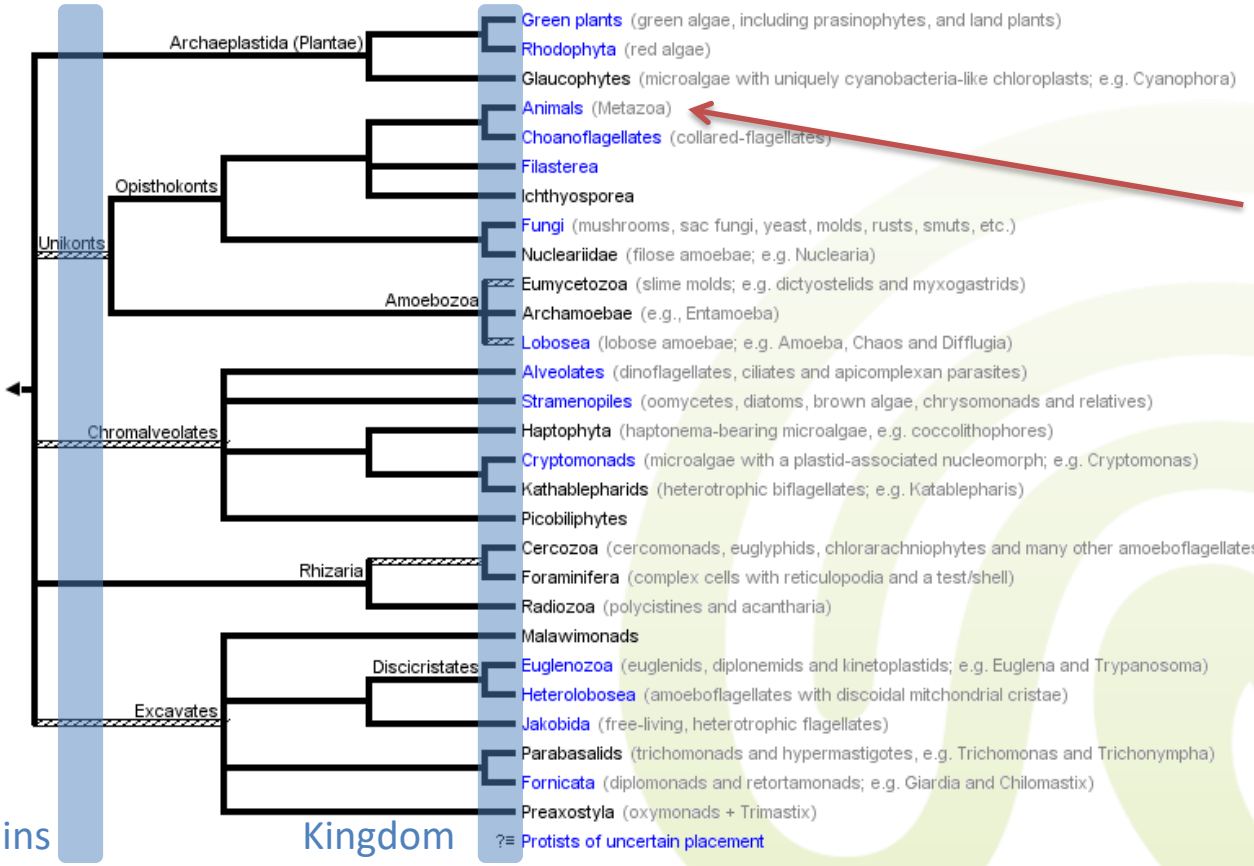
3.3 Taxonomies

Detour

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



Domain



3.3 Taxonomies

- Example: **American 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 North America***)
  - species: **Hudsonicus** (backbone, nursing its young, sharp front teeth, like squirrel, bushy tail & trees, from N-America, ***brown fur with white belly***)

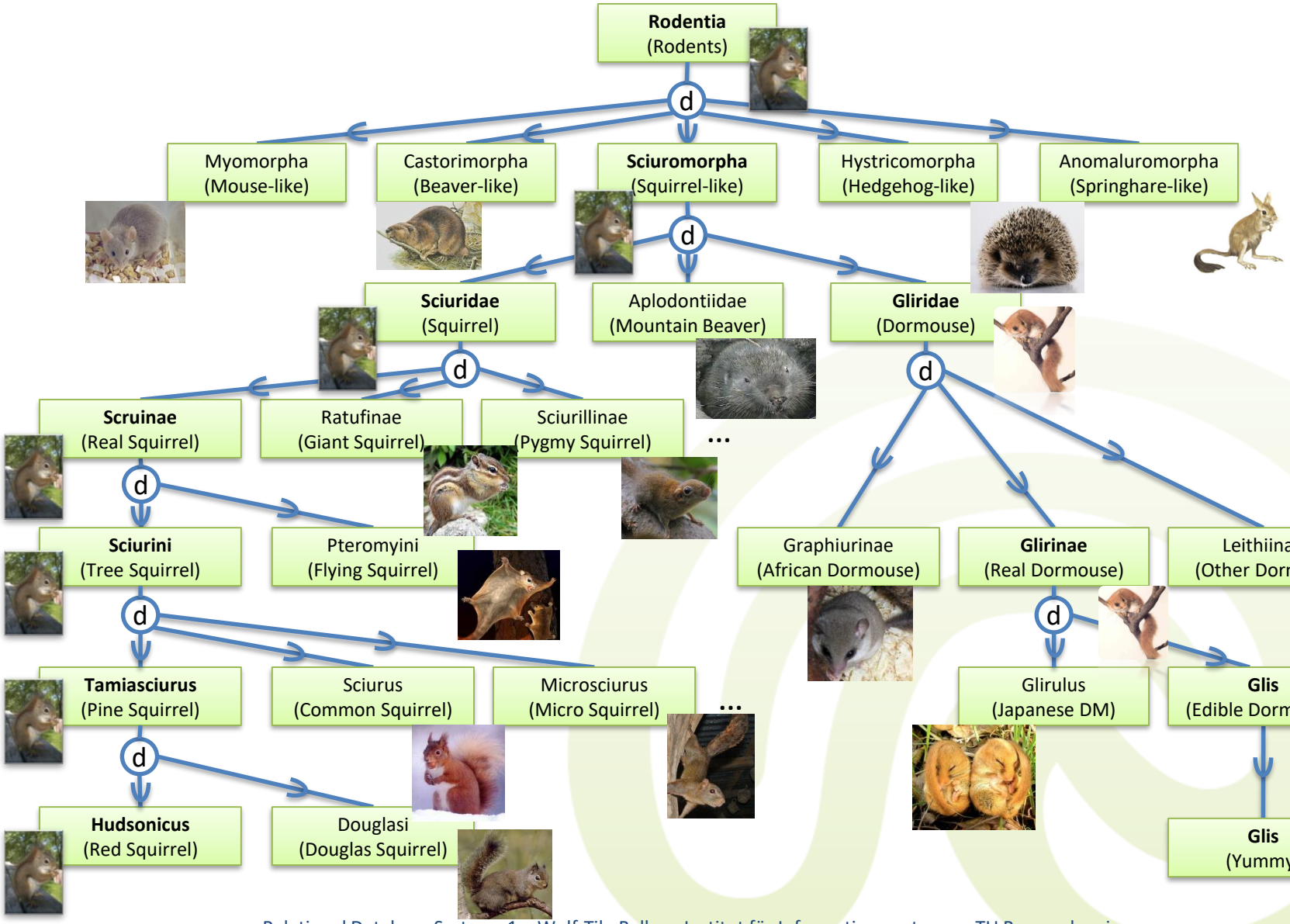
3.3 Taxonomies

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***)

3.3 Taxonomies

Detour



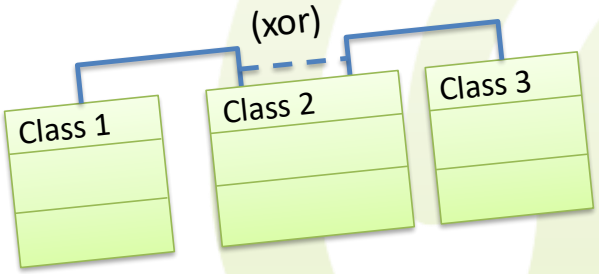
3.3 Ontologies in CS

Detour

- Recently, creating **ontological models** became fashionable in CS
  - so called **ontologies**
  - widely used in e.g. medical informatics, bio-informatics, Semantic Web
- 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**
  - e.g. OWL, DAML+OIL, IDEF

3 Extended Data Modeling

- Alternative ER Notations
- Extended ER
  - Inheritance
  - Complex Relationships
- Taxonomies & Ontologies
- **UML**

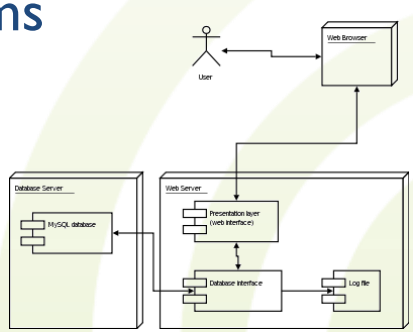


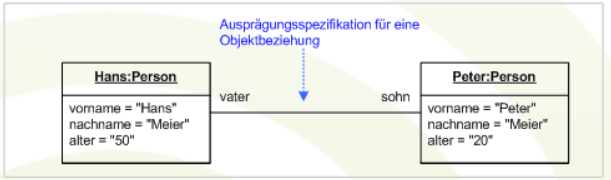
3.4 UML

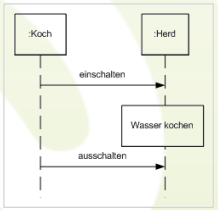
- UML (**Unified Modeling Language**) is a set of multiple modeling languages and diagram types
  - first standardized in 1997
  - unification of dominating object-oriented software design methods
    - James Rumbaugh: OMT
    - Grady Booch: Boochs Method
    - Ivar Jacobsen: OOSE

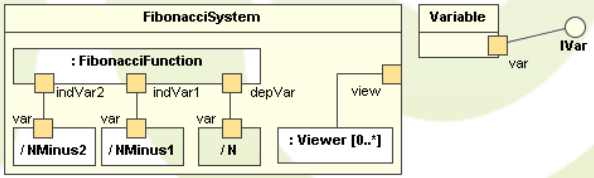
3.4 UML

- 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



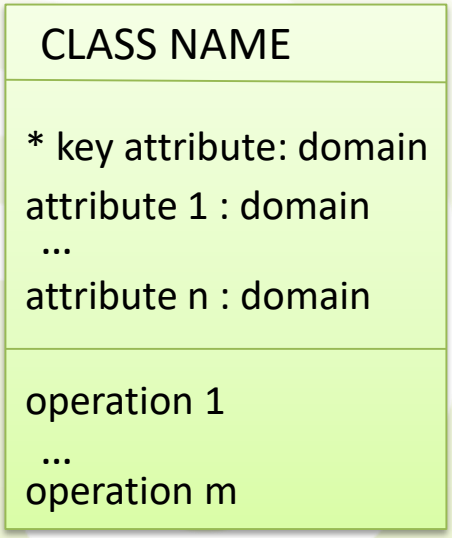






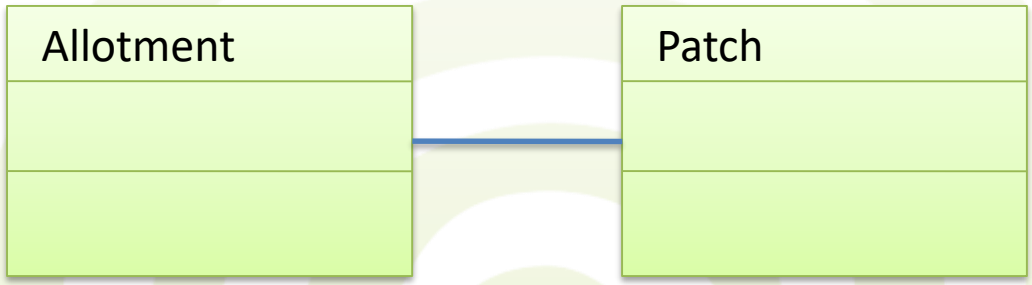
3.4 UML

- 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
    - no composite or multivalued attributes
    - derived attributes are modeled as operations
    - key attributes are marked with a *
  - operations are only needed for derived attributes in pure data models
  - entity type instances are called **objects**



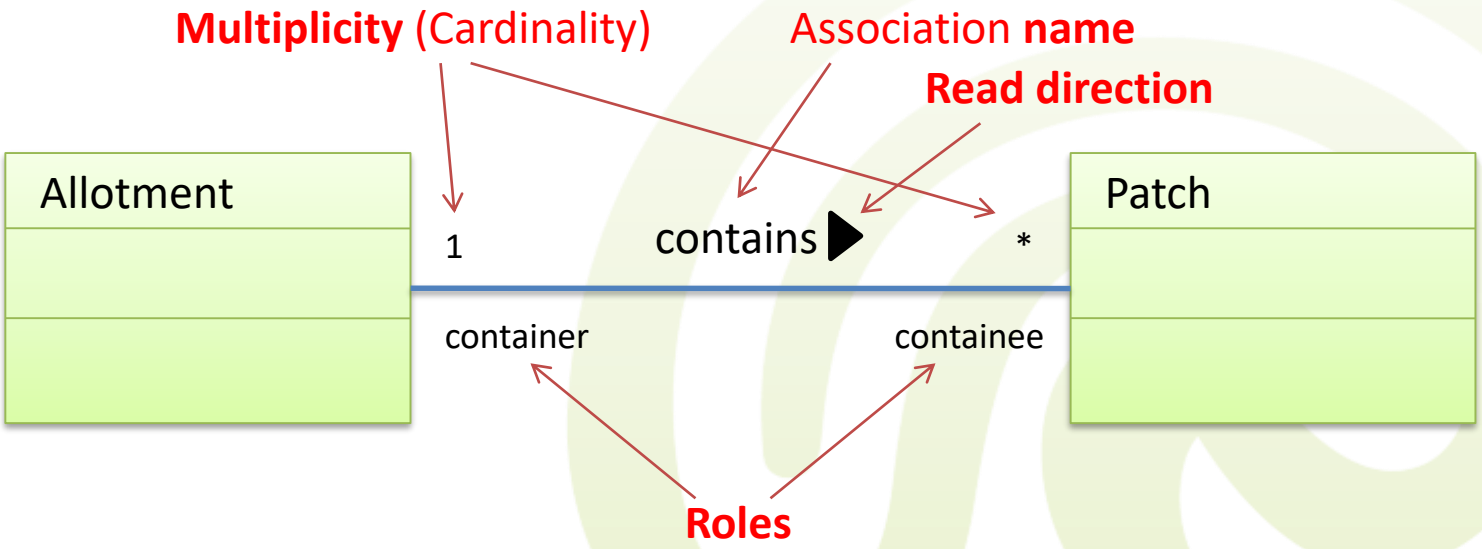
3.4 UML

- 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



3.4 UML

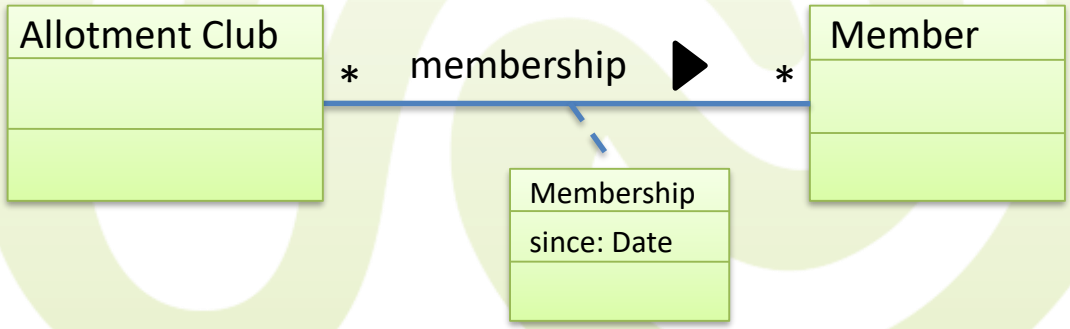
- Example
  Allotments may contain multiple patches (dt.: Beete). Each patch is contained in exactly one allotment.
  - **careful:** multiplicity in opposite direction to Chen ER



3.4 UML

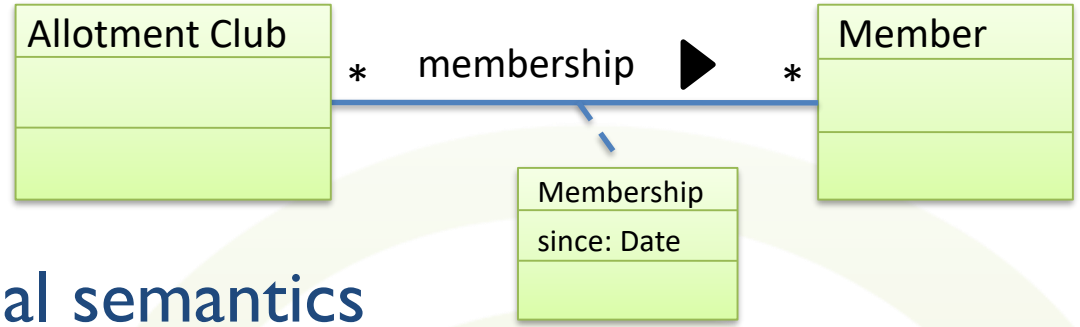
- UML does not allow to add attributes to associations directly
- Workaround: **association classes**
  - association classes belong to an association (indicated by dashed line)
  - they share the association name
  - each instance of the association creates an according class object

Modelling this entity usually only makes sense when your Universe of Discourse contains multiple allotment clubs!

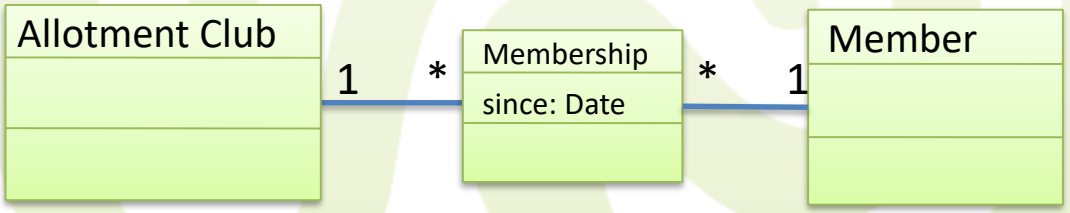


3.4 UML

- Association classes cannot directly be replaced by a *normal* class

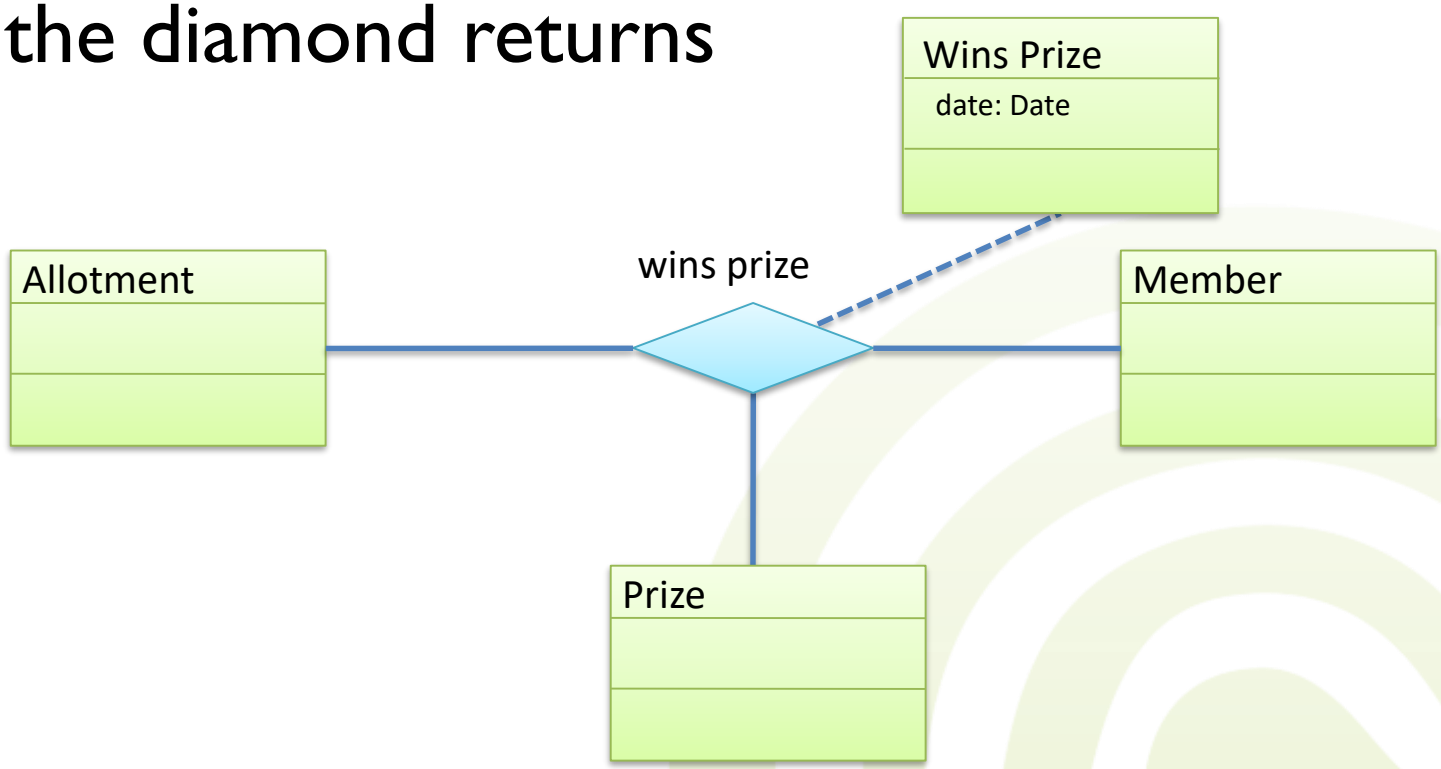


  - introduces additional semantics
  - the replacement model allows that a person (member) joins the same allotment club **twice**!



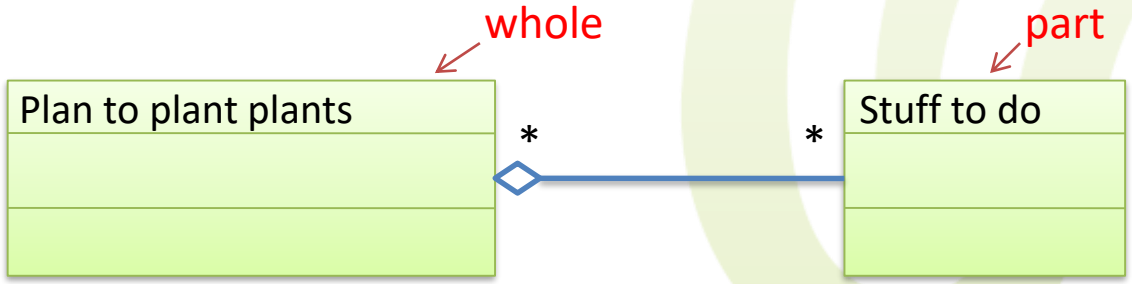
3.4 UML

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



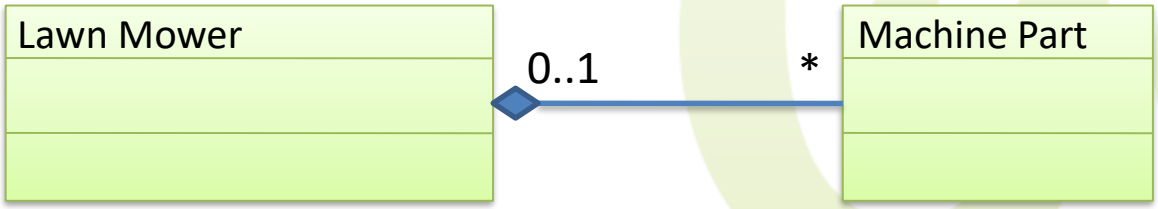
3.4 UML

- **Aggregation**
  - the aggregation is a special association within UML
  - colloquial: ***is_part_of*** or *consists_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 plant plants consists of several things that need to be done.*



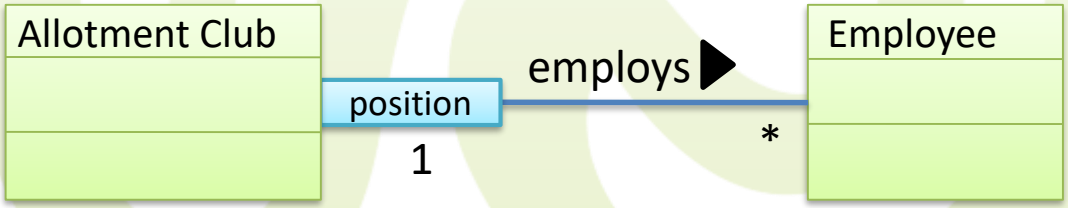
3.4 UML

- **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.
      It may also exist alone.
    - *: not allowed. Part of one object max.
  - Example
    - *A lawn mower is made of multiple parts.*



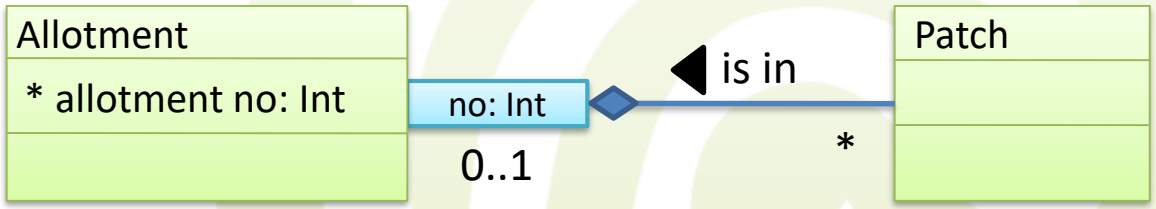
3.4 UML

- **Qualified associations**
  - associations may be qualified by an additional attribute
    - each association instance between objects is **classified** by this attribute
  - Example
    - *Garden Corp. employs Gnarden Gome as CEO*
    - *Garden Corp. hires Moritz Mower to look after the pathways*



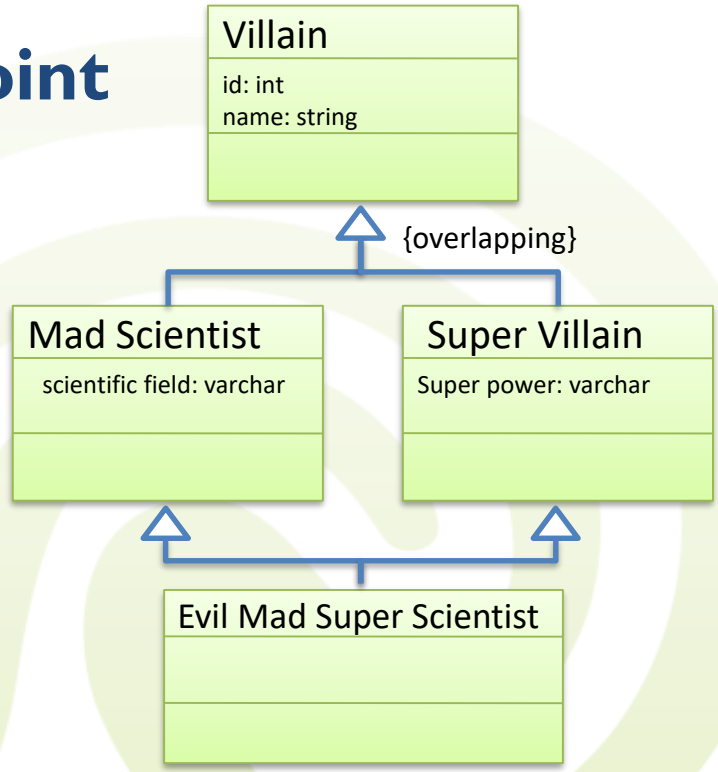
3.4 UML

- **Weak entities** through qualified associations
  - a weak entity's partial key is modeled by the classifying attribute of a qualified association
  - Example
    - *An allotment possibly contains many patches. An allotment patch is identified by a number and the allotment number of its containing allotment.*



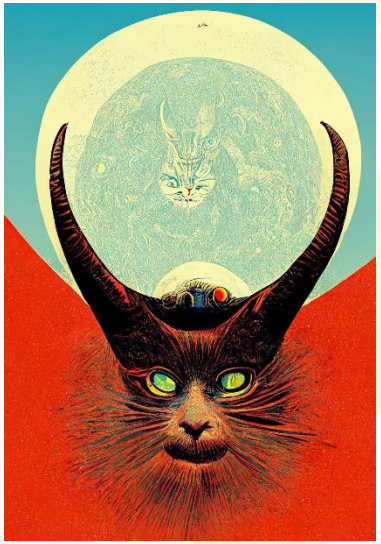
3.4 UML

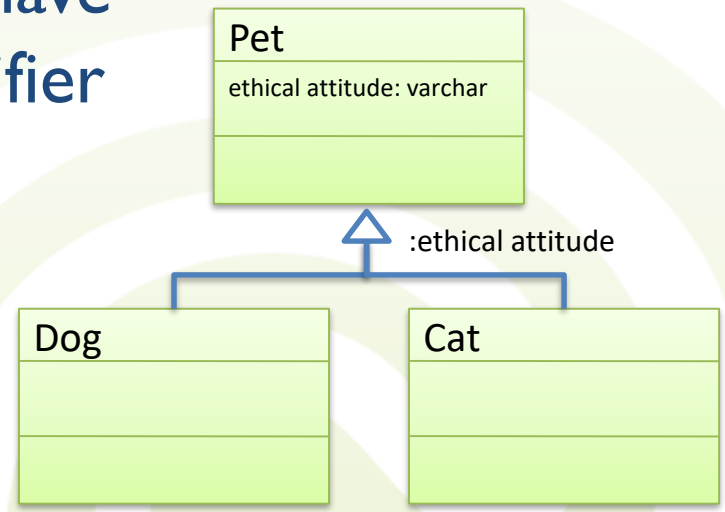
- **Generalization**
  - induces a class-subclass relationship (***is_a***)
    - diagrammed with an hollow arrow
  - by default, generalization is **disjoint**
    - **overlapping** is additionally annotated in curly brackets
  - by default, generalization is partial (**incomplete** in UML)
    - total (**complete**) is also annotated in curly brackets



3.4 UML

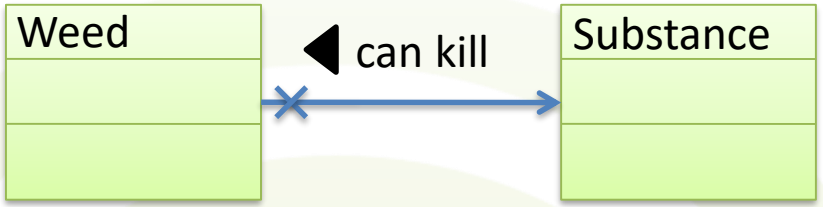
- **Classification attributes**
  - similar to EER's **attribute-defined** relationship types
  - denoted by *:attribute_name*
  - all objects of a given subtype have the **same value** for the classifier attribute





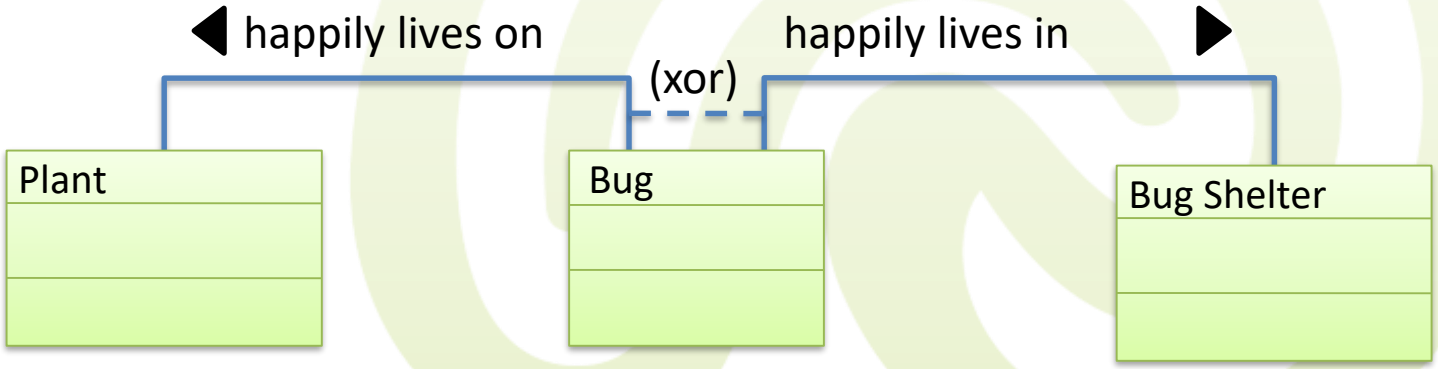
3.4 UML

- 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 type of weed (dt.: Unkraut), you can navigate to the substances which may kill it
    - you cannot natively navigate from a substance to weeds
      - This may modify what the actual data structures implementing the model may look like



3.4 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 bug lives either on a plant or in a bug shelter, but not both.*



3 Next Week

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