Conceptual Modelling Languages and ORM

Enrico Franconi

http://krdb.eu
KRDB Research Centre for Knowledge and Data
Free University of Bozen-Bolzano, Italy

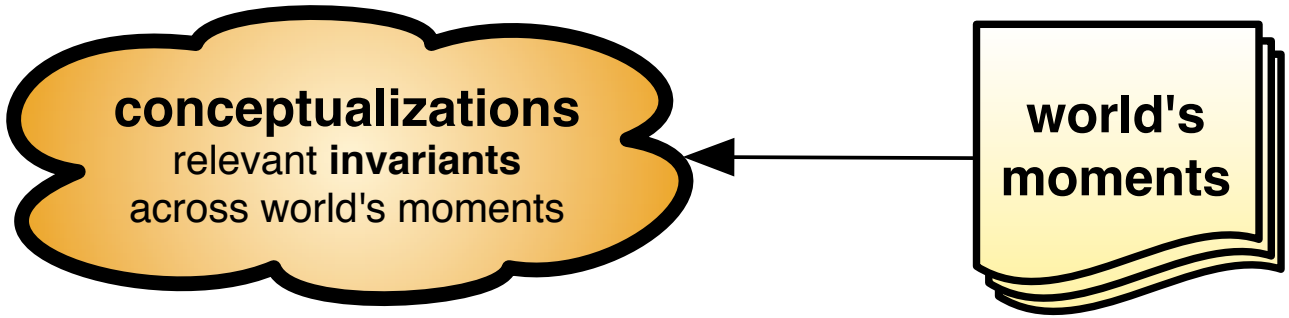
#SSOW2019

What is an Ontology

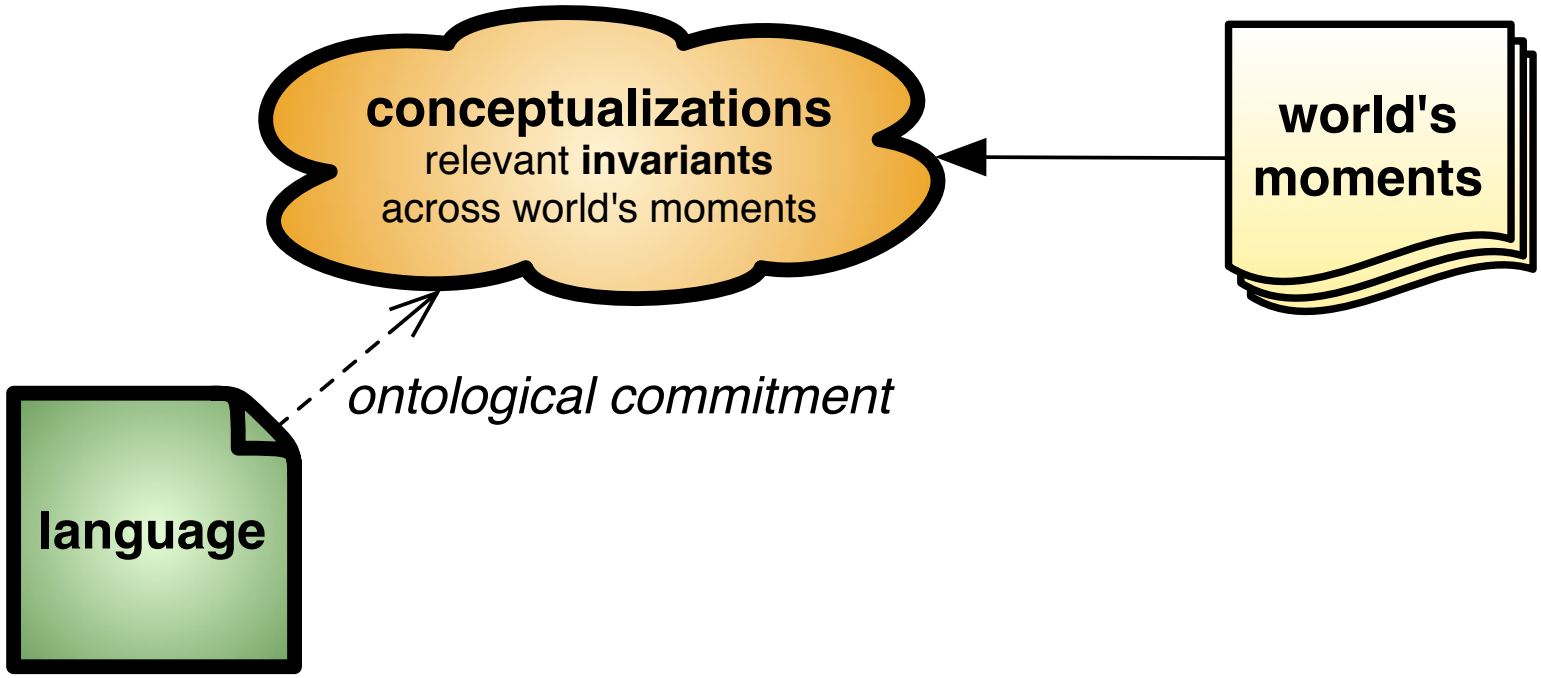
- An ontology specifies a set of constraints, which declare what should necessarily hold in any possible world.
- Any possible world should conform to the constraints expressed by the ontology.
- Given an ontology, a legal world description (called also model) is a finite possible world satisfying the constraints.

Conceptual Schema and Intended Meaning

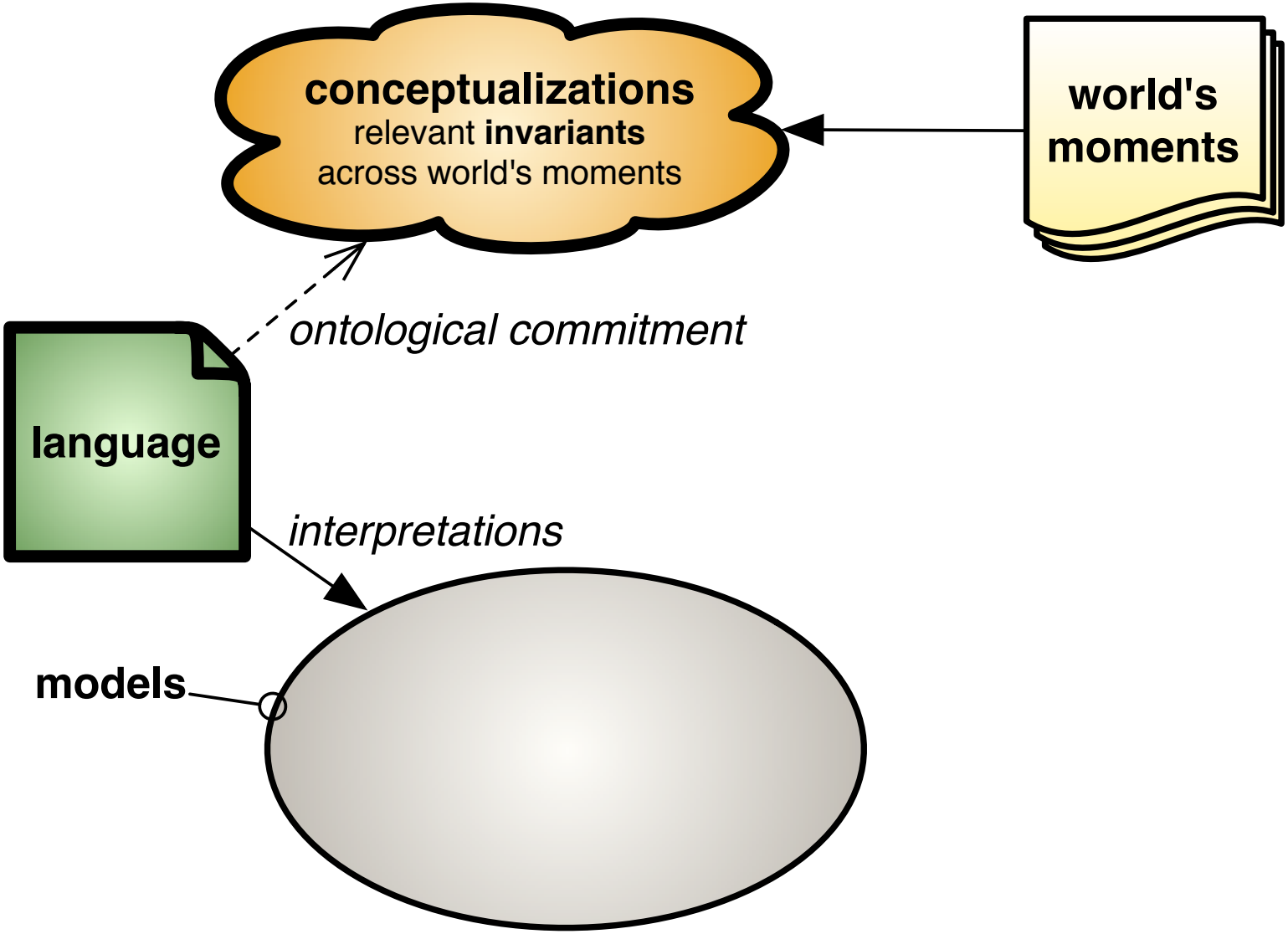
Conceptual Schema and Intended Meaning



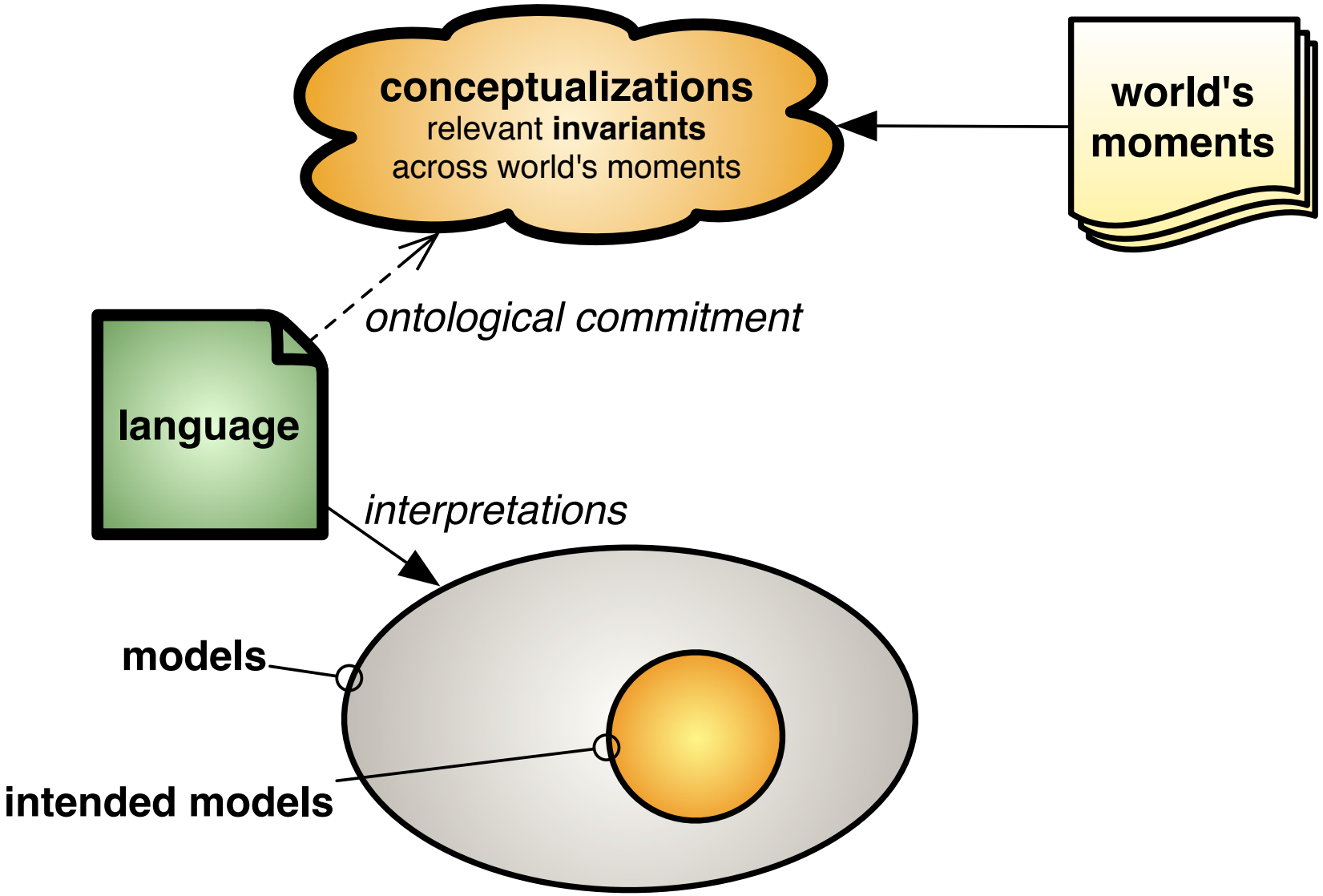
Conceptual Schema and Intended Meaning



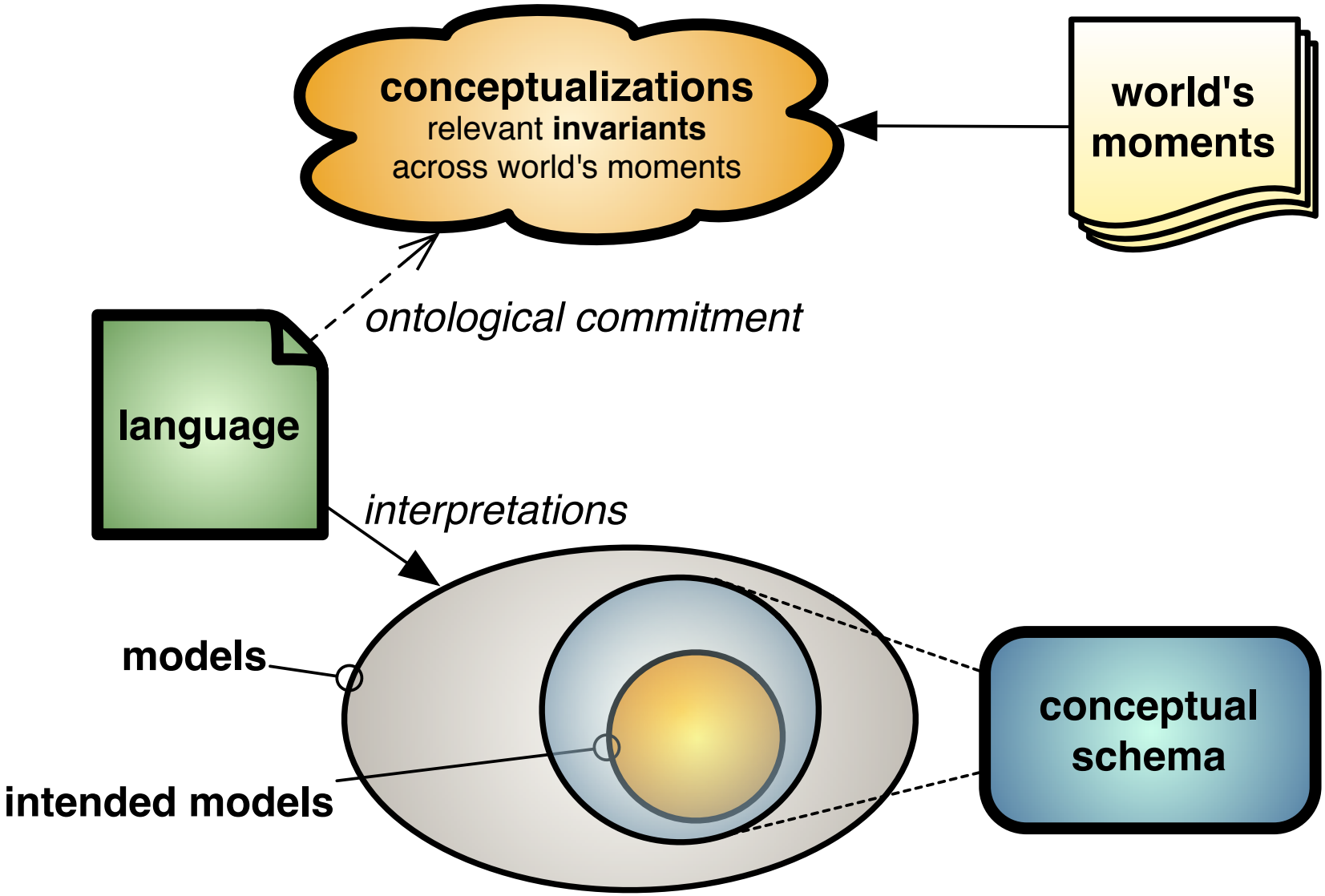
Conceptual Schema and Intended Meaning



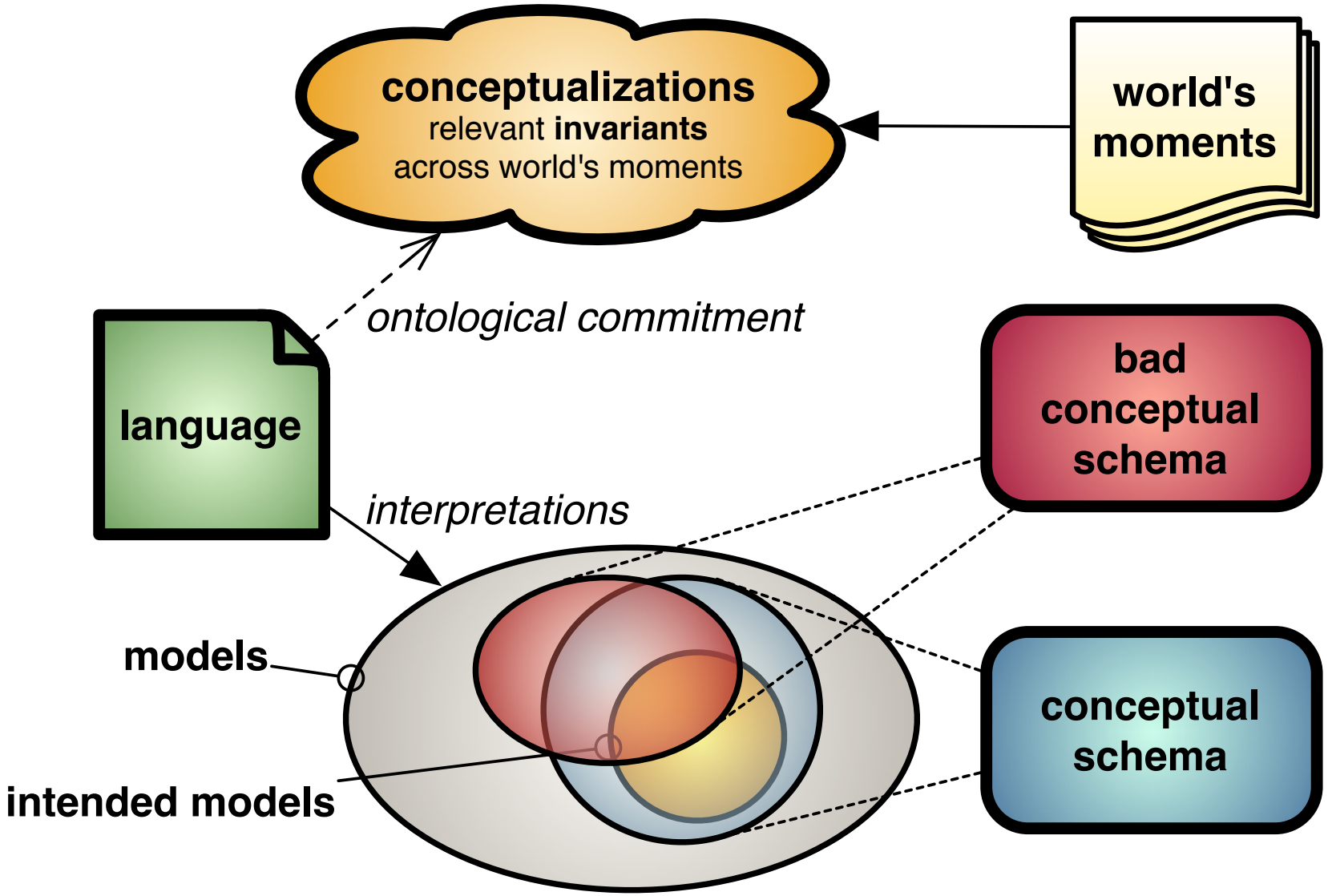
Conceptual Schema and Intended Meaning



Conceptual Schema and Intended Meaning

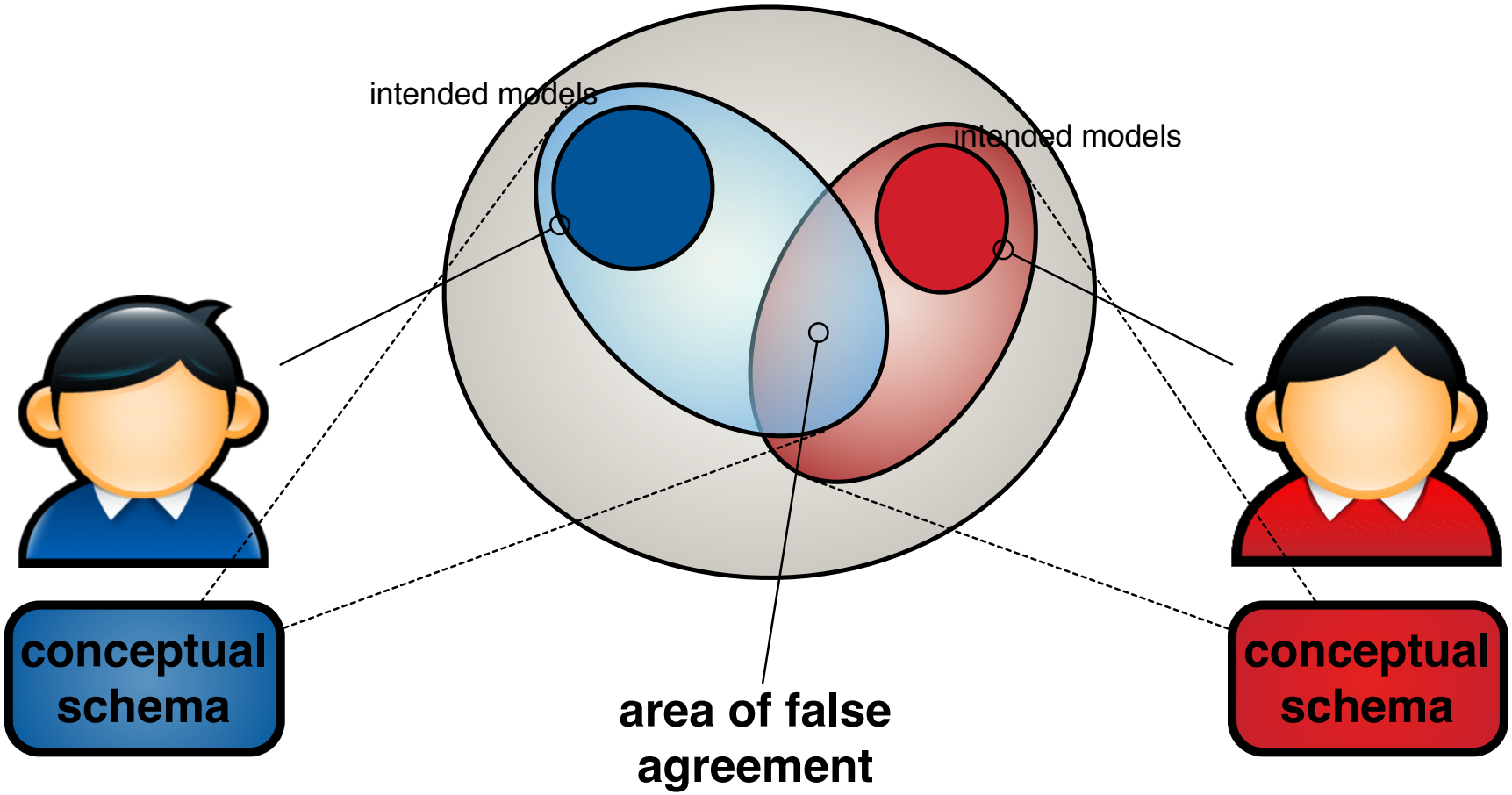


Conceptual Schema and Intended Meaning



Direct link to logical theories!

The Need for Validation, and the Importance of Precision



N.B.: Precision can only be defined w.r.t. a language. Whether the language is good or not is another story!

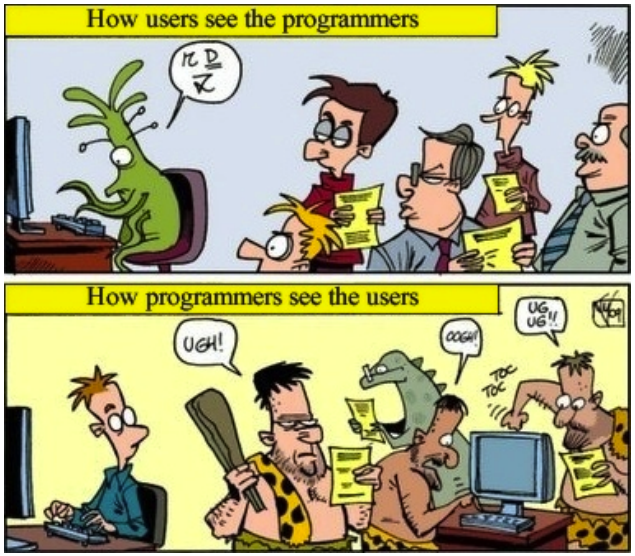
Conceptual Layer

Governs the information system at the conceptual level.

- It is completely *independent* from user interfaces, storage and data access techniques: stability.
- Conceptual information processor: mediates the communication between the external users and the internal data structures (e.g., a database).
- The conceptual layer models a virtual information base.

Conceptual Modeling Languages: Criteria

Clarity: how easy the language can be understood and used (by different stakeholders).



- Graphical vs textual notations.
- Unambiguous language: formal foundation.
- The more expressive the language, the more difficult is to retain clarity.
- Less expressive languages require complex combinations of their few constructs.

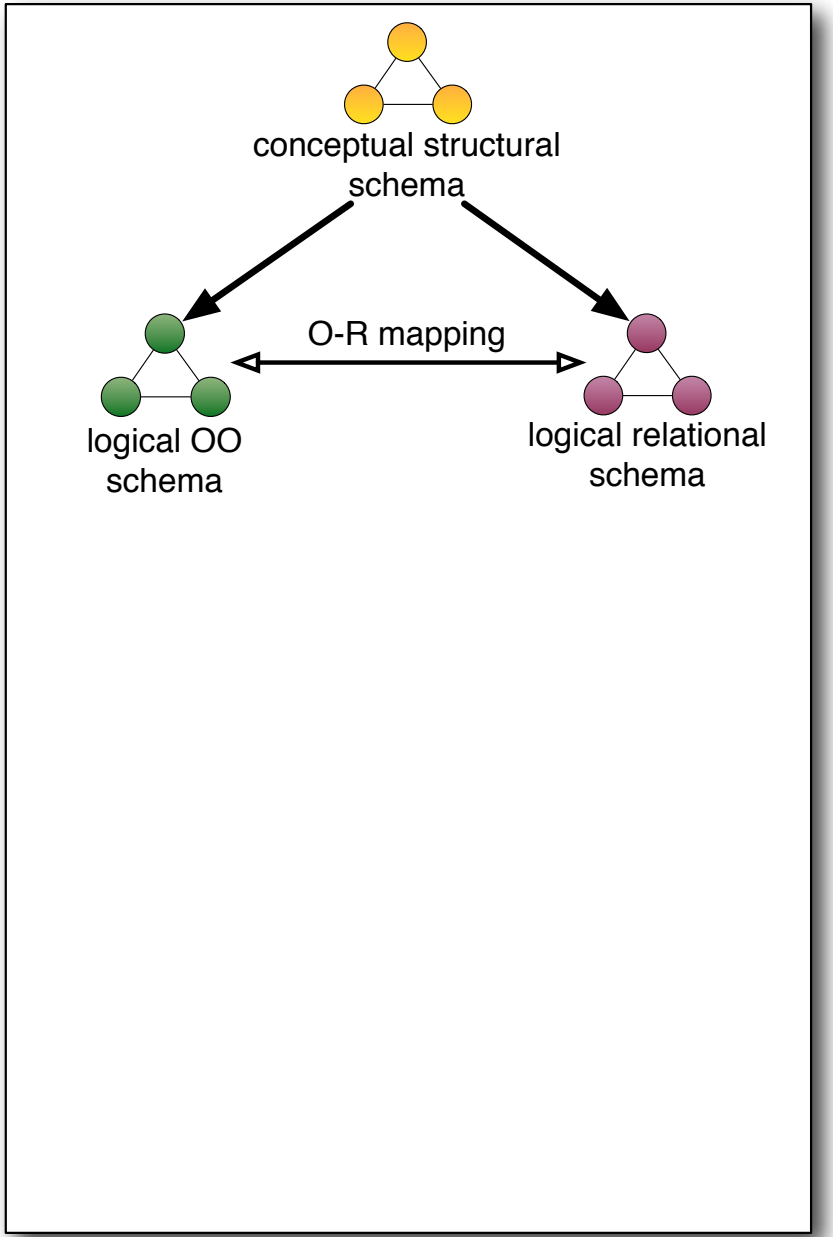
- Abstraction: remove unnecessary details. Use requirements to drive abstraction.
- Simplicity: Prefer simple schemas. Follow *Occam's razor* with a critical approach.
- Orthogonality: minimization of the overlapping of language constructs. Their (in)dependence must reflect the one of the corresponding domain aspects.

Trade-Offs

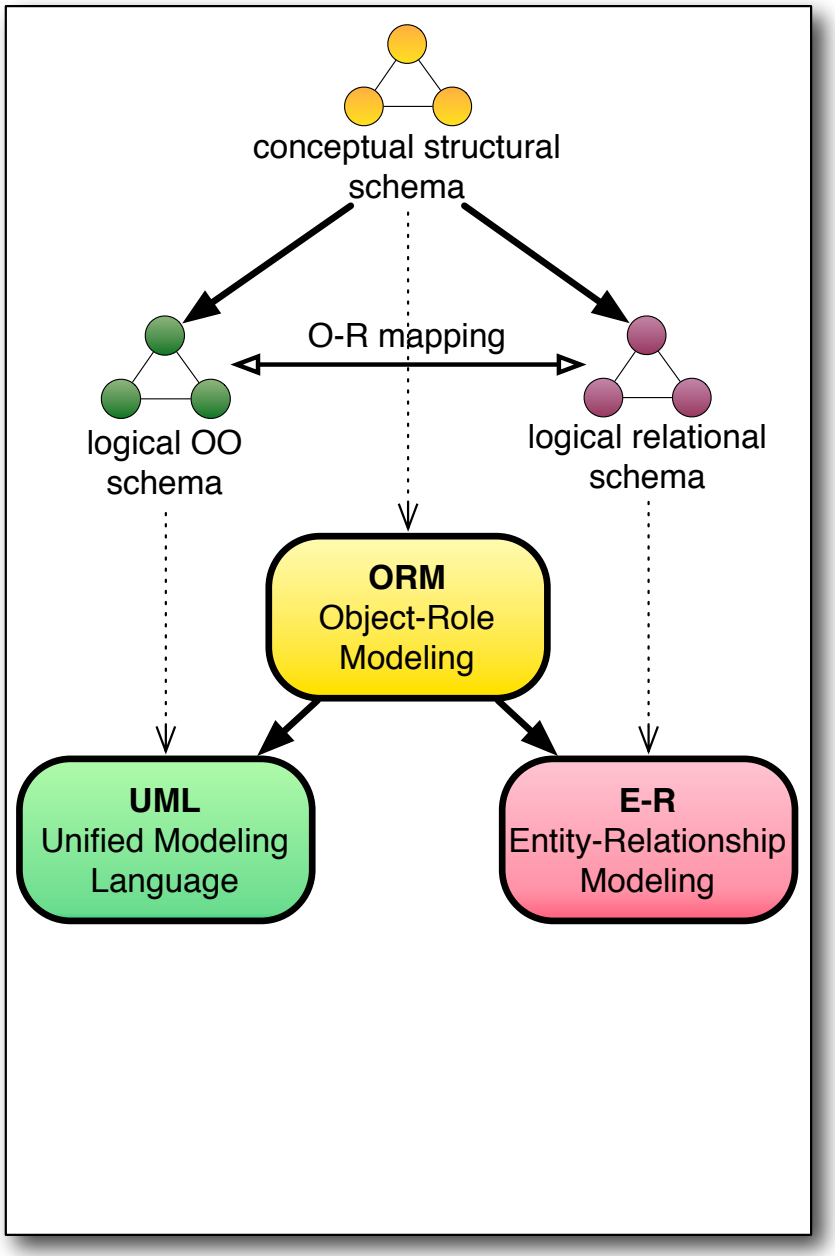
Trade-offs between contrasting desiderata.

- Expressivity vs tractability: the more expressive the language, the harder it is to *compute* with it.
- Parsimony vs convenience: fewer concepts vs compact models.
  - Would you use `Assembler` to implement a web server?

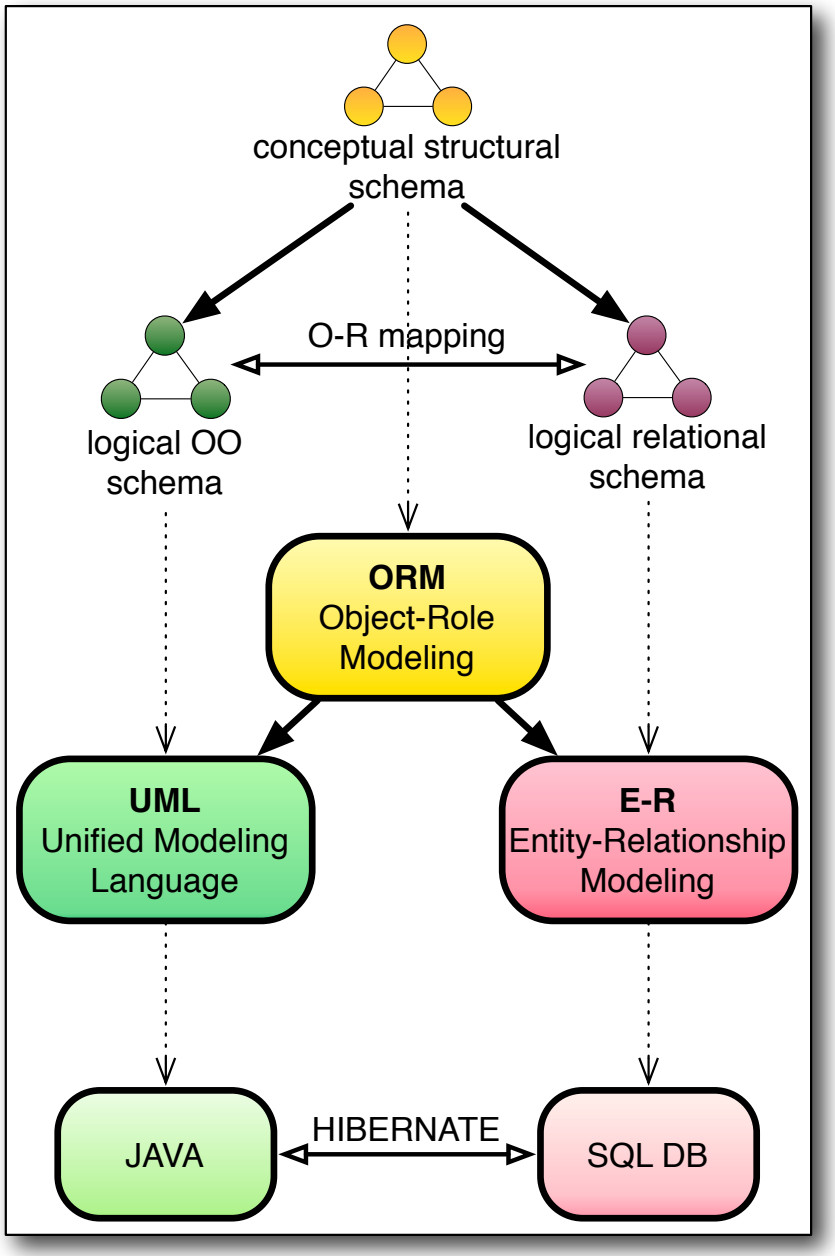
Modeling Languages and Frameworks



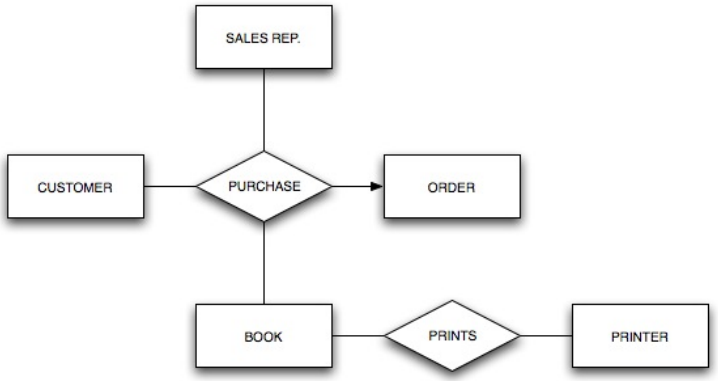
Modeling Languages and Frameworks

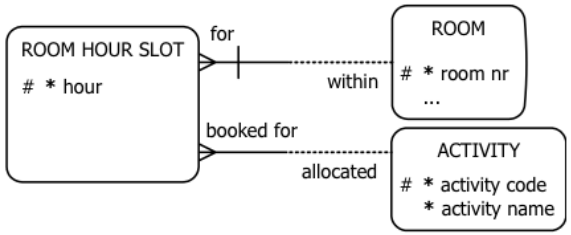


Modeling Languages and Frameworks



E-R: Abstract Representation of Data



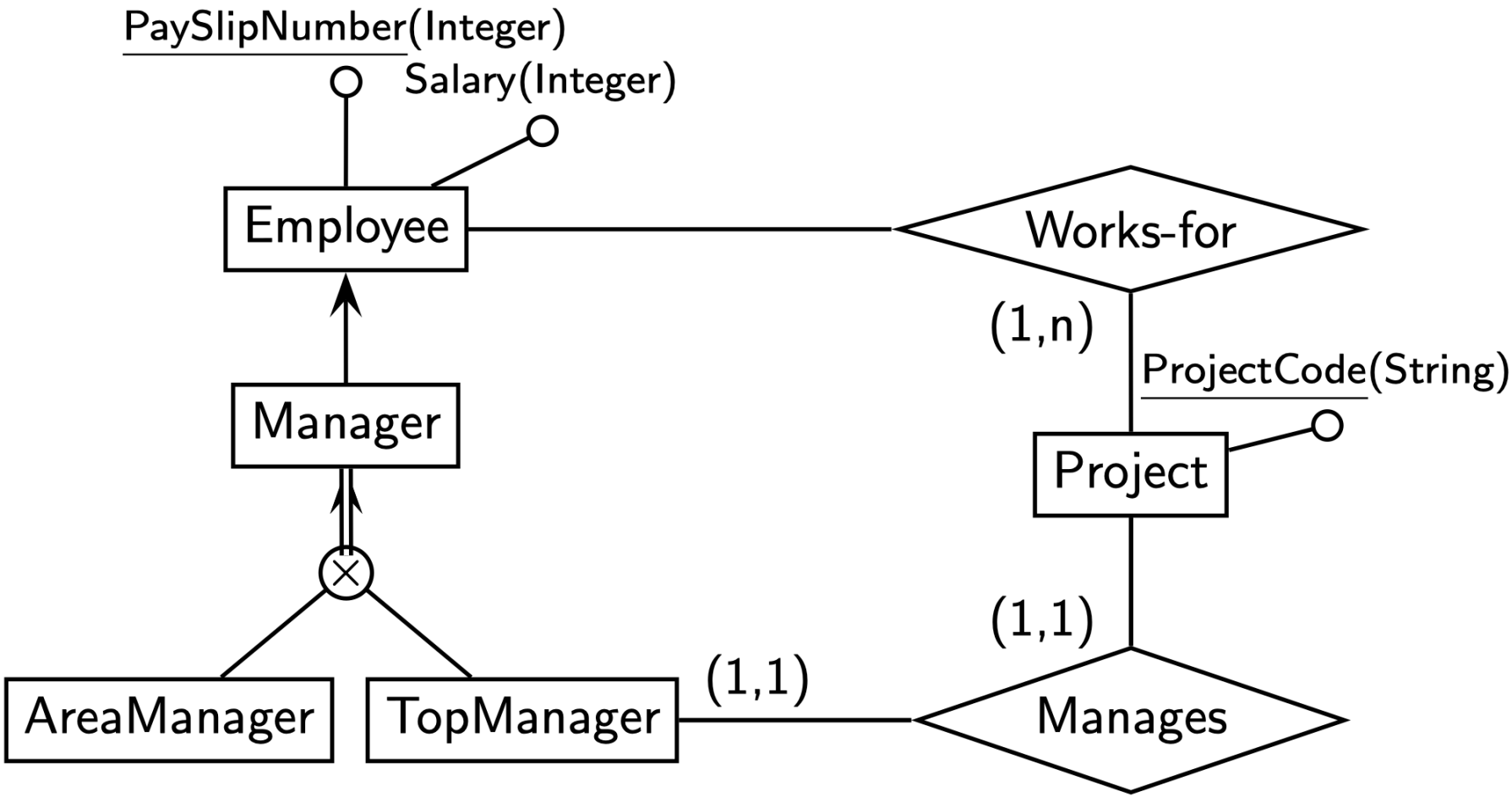


- The most widely used approach to data modeling.
- Key notions:
  - entities, relationships, attributes;
  - identification and multiplicity constraints.
- Lack of dynamic modeling.
- Close to relational database schemas $\rightarrow$ logical relational modeling.
- Different dialects: Chen, Barker, IE, IDEF1X, EXPRESS ...

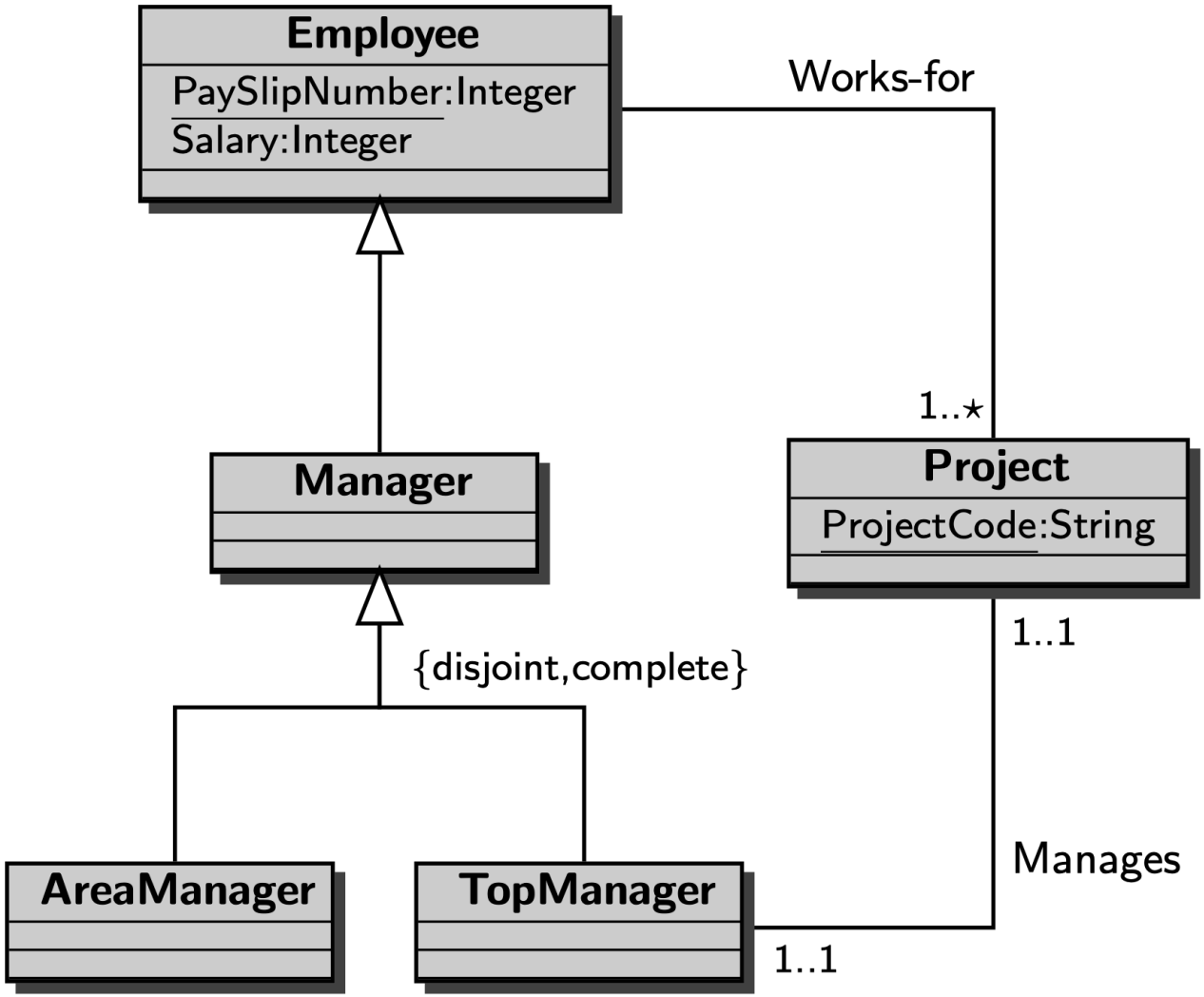
UML: Modeling for OO Software Engineering

- Family of notations:
  - Structure diagrams: class/object diagram, component, composite structure, deployment, package, profile.
  - Dynamic diagrams:
    - ★ Behavior: use case, state machine, activity.
    - ★ Interaction: communication, interaction overview, sequence, timing.
- Key notions:
  - object (class) as entity (type);
  - attributes (with visibility), relationships (basic, generalization, aggregation, composition);
  - multiplicity constraints, OCL;
  - behavioral aspects (operations, parameters, visibility);
  - no mandatory identification for objects (implicit reference, object ids).
- OMG standard.

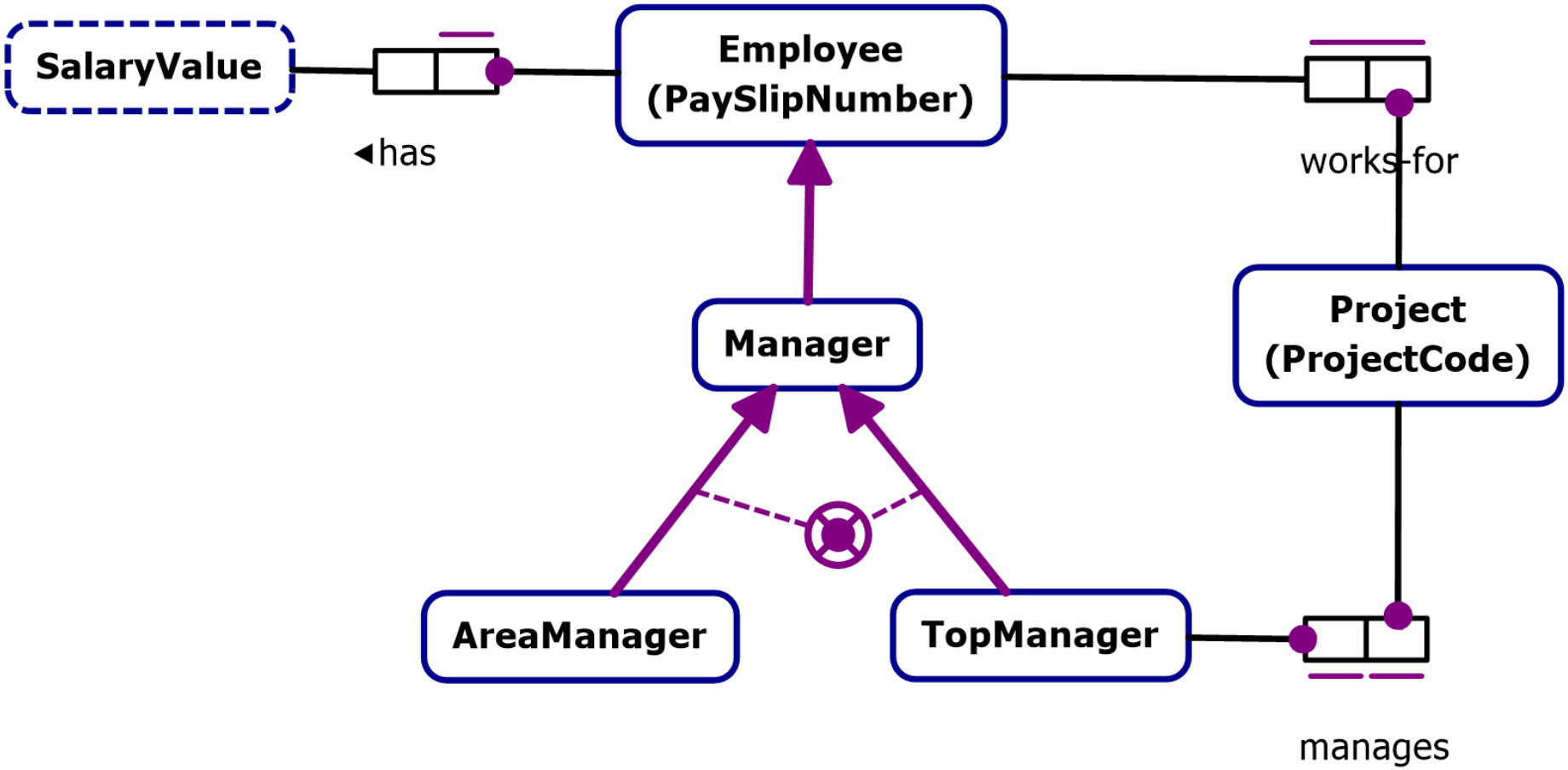
ER Diagram


PaySlipNumber(Integer)
Salary(Integer)
Employee
Works-for
(1,n)
ProjectCode(String)
Manager
Project
(1,1)
(1,1)
AreaManager
TopManager
Manages

UML Diagram

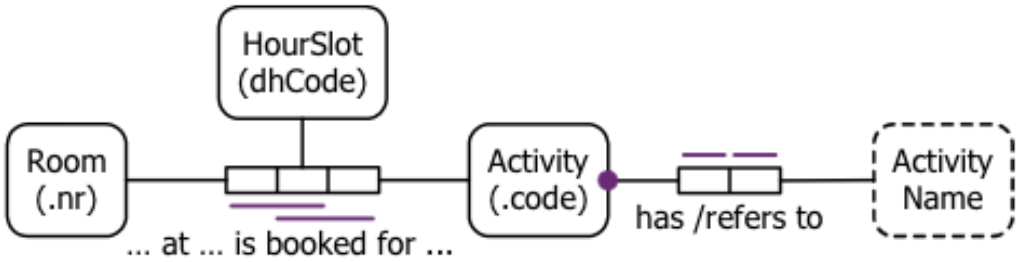

Employee
PaySlipNumber:Integer
Salary:Integer
Works-for
1..⋆
Project
ProjectCode:String
Manager
1..1
{disjoint,complete}
Manages
AreaManager
TopManager
1..1

ORM Diagram


SalaryValue
has
Employee
(PaySlipNumber)
works-for
Manager
Project
(ProjectCode)
AreaManager
TopManager
manages

ORM

Fact-oriented conceptual approach to modeling and querying the information semantics of a UoD.



- Falkenberg in 1973; formalised and enhanced by Halpin ($\rightarrow$ ORM 2).
- Starts from elementary facts.
- Key notions:
  - objects (relevant entities) playing roles (parts in relationship types);
  - intuitive treatment of n-ary (ordered) roles;
  - rich business constraints (subsumes UML and E-R);
  - no use of attributes!
- Diagrammatic + textual form (controlled natural language).
- Two forms of validation with domain experts:
  - verbalization - natural language description of the diagrams;
  - population - sample prototypical instances and counterexamples.

Fact Types in ORM

Having attributes in the conceptual design is a premature commitment.

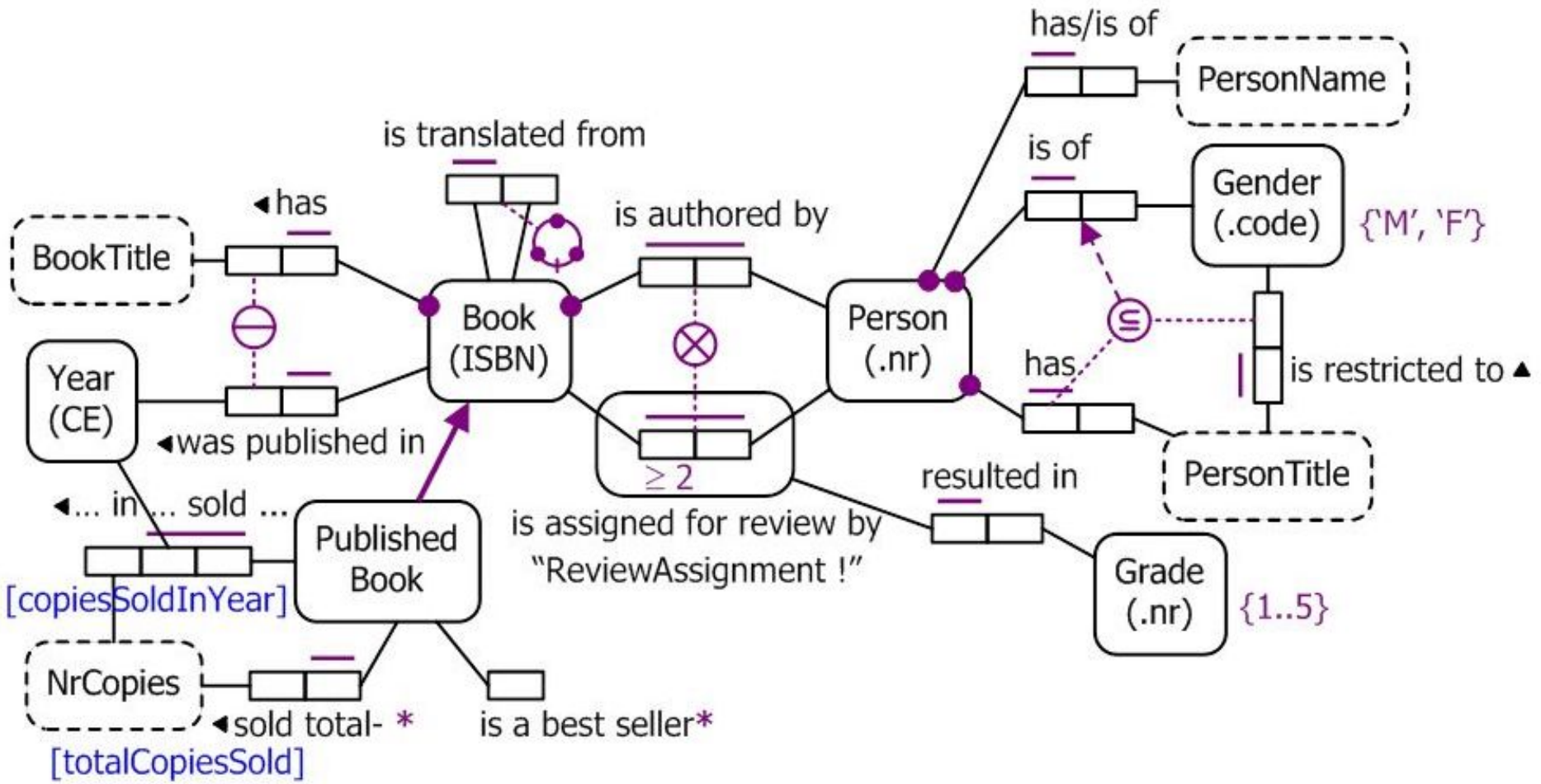
- In ORM fact structures are expressed as fact types:
  - unary (e.g. Person *smokes*);
  - binary (e.g. Person *was born* on Date);
  - ternary (e.g. Person *visited* Country in Year);
  - ...
- Advantages:
  - semantic stability (minimize the impact of change caused by the need to record something about an attribute);
  - natural verbalization (all facts and rules may be easily verbalized in sentences understandable to the domain expert);
  - populatability (sample fact populations may be conveniently provided in fact tables);
  - null avoidance (no nulls occur in populations of base fact types, which must be elementary or existential).
- Attributes can be obtained through views over the ORM schema.

Roles

Modeled by logical predicates: sentences containing "object holes".

- Object hole: placeholder for an object designator (object term).
  The person with firstname 'Ann' *smokes* $\rightarrow$ ... *smokes* (unary).
- Most predicates: binary.
  The person with firstname 'Ann' *employs* the person with firstname 'Bob' $\rightarrow$ ... *employs* ...
- Extension to arbitrary n-ary predicates.
- Principles:
  - Order matters.
  - The n object terms must not be necessarily distinct.
  - The obtained proposition must not be expressible as a conjunction of simpler independent propositions.

ORM with an example

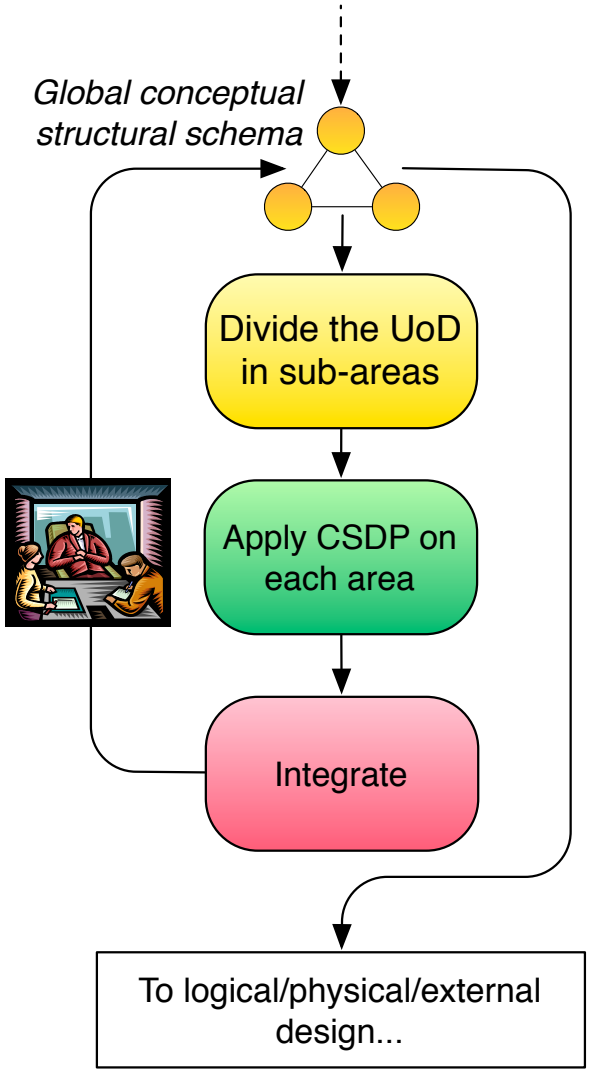


Each PublishedBook **is a** Book that was published in **some** Year.

* **For each** PublishedBook, totalCopiesSold= **sum**(copiesSoldInYear).
* PublishedBook is a best seller **iff** PublishedBook sold total NrCopies >= 10000.

ORM Design Methodology

ORM provides a Conceptual Schema Design Procedure.

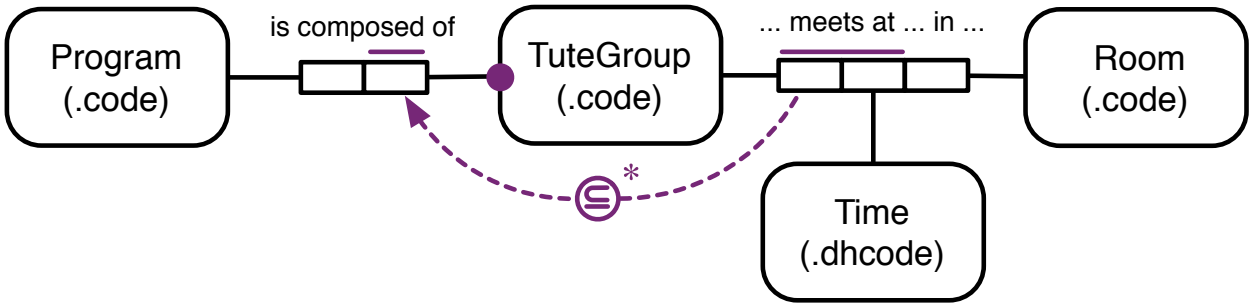


1. Transform familiar examples into elementary facts.
2. Draw the fact types, and apply a population check.
3. Check for entity types to be combined, and note any arithmetic derivations.
4. Add uniqueness constraints, and check the arity of fact types.
5. Add mandatory role constraints, and check for logical derivations.
6. Add value, set-comparison, and subtyping constraints.
7. Add further constraints, do final checks.

From Design to Implementation

- A conceptual schema is designed to ultimately store, update and query the relevant information of a domain.
- Concrete typical outcomes:
  - Development of a *physical database schema*.
  - Development of the *model layer* of an (object-oriented) software application.
- In both cases, we need to *map* the conceptual schema to a corresponding logical, and then physical, schema.
- Main methodological steps:
  1. Design the conceptual schema.
  2. Annotate the conceptual schema with mapping choices.
  3. Mapping the conceptual schema to a logical schema (relational, object-oriented, ...).
  4. Manipulate the logical schema.
  5. Generate the corresponding physical schema (MySQL DB, Java classes, ...).

Relational Translation



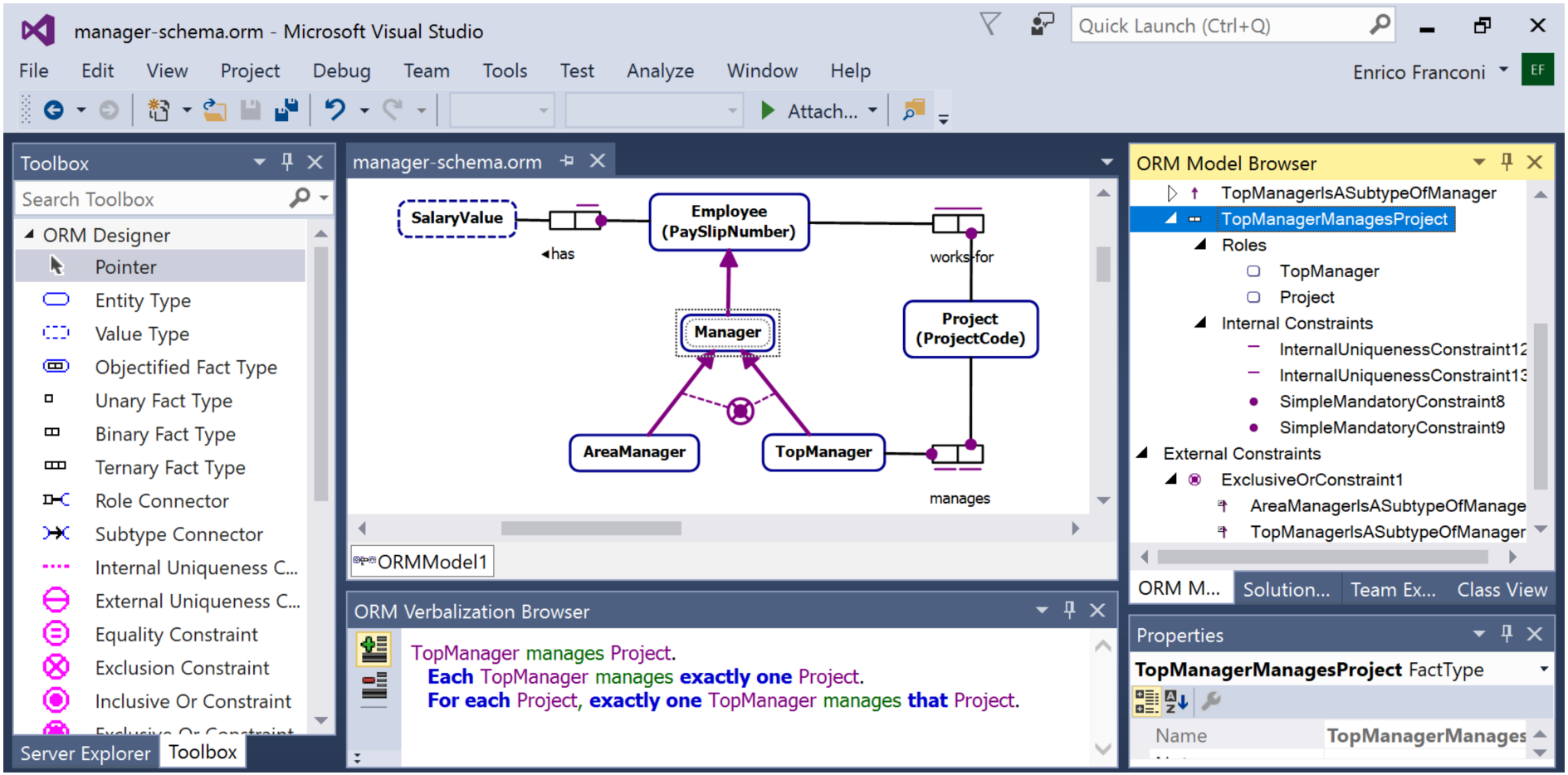
TuteGroup(tuteCode, progCode)

Meets(tuteCode, timeDHCode, roomCode)

TuteGroup	
PK	**tuteCode**
	progCode

Meets	
PK, FK1	**tuteCode**
PK	**timeDHCode**
	roomCode

NORMA in Microsoft Visual Studio

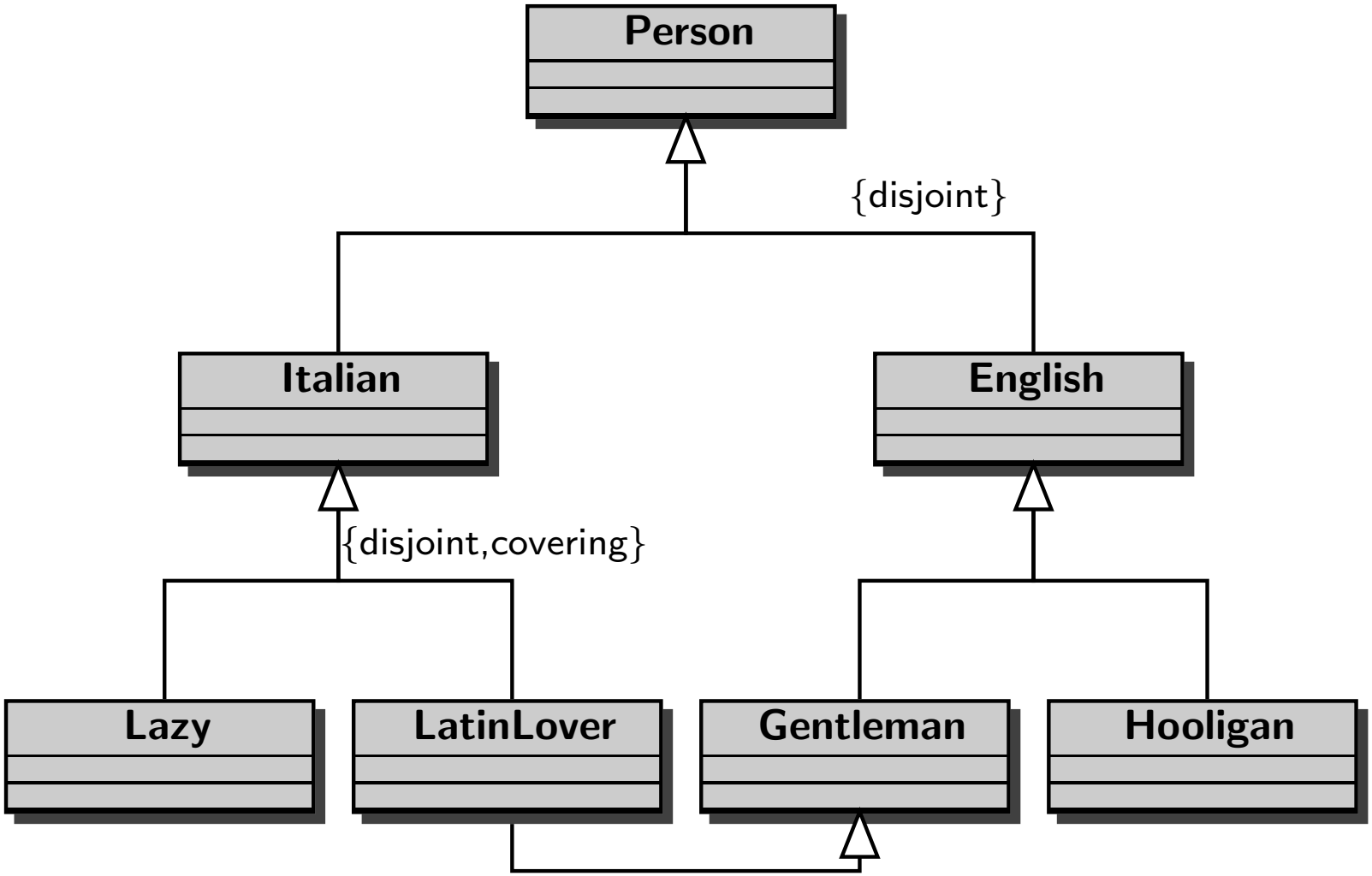


Reasoning

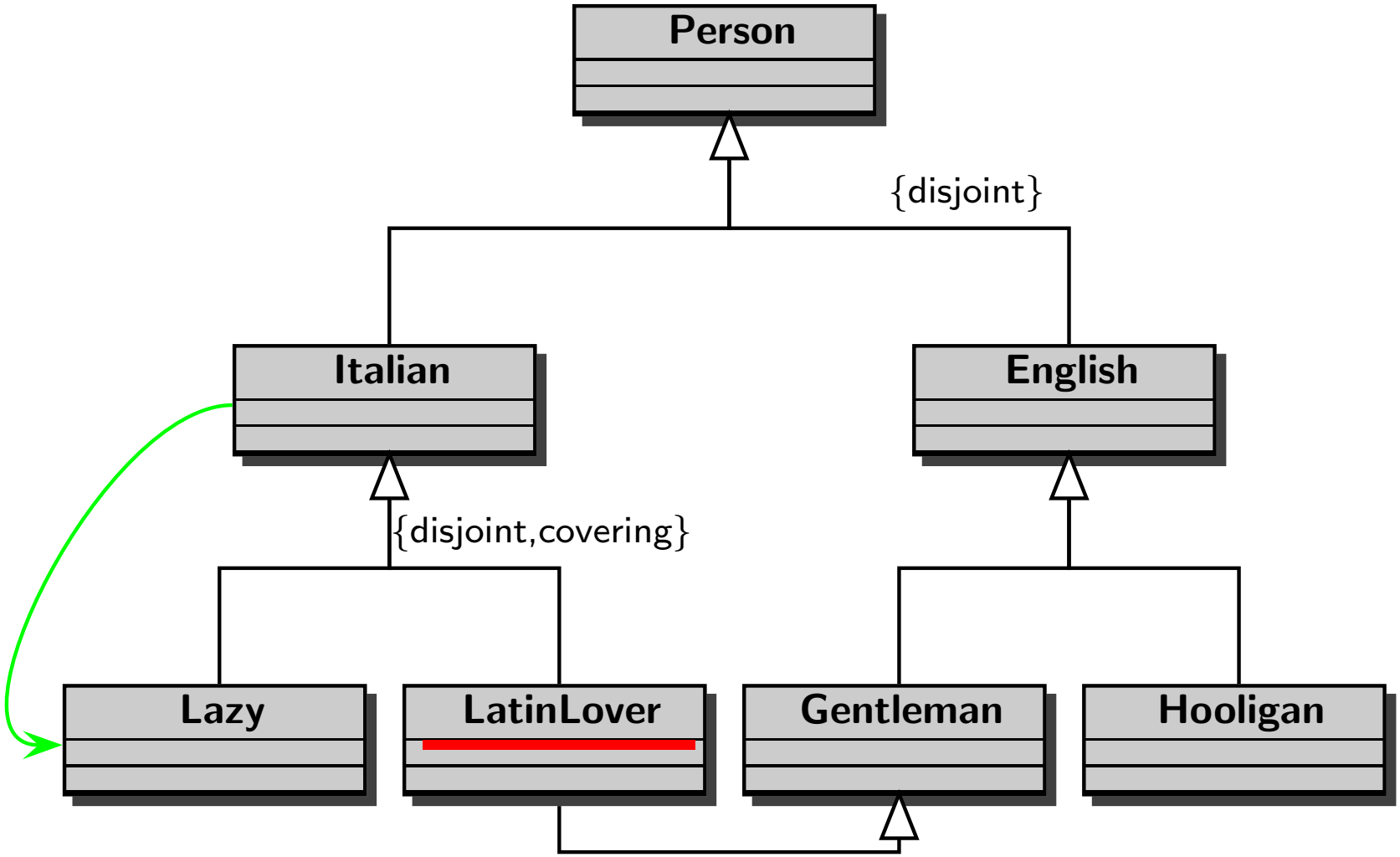
Given an ontology – seen as a collection of constraints – it is possible that additional constraints can be inferred.

- A class is inconsistent if it denotes the empty set in any legal world description.
- A class is a subclass of another class if the former denotes a subset of the set denoted by the latter in any legal world description.
- Two classes are equivalent if they denote the same set in any legal world description.
- A stricter constraint is inferred – e.g., a cardinality constraint – if it holds in in any legal world description.
- ...

Simple reasoning example



Simple reasoning example

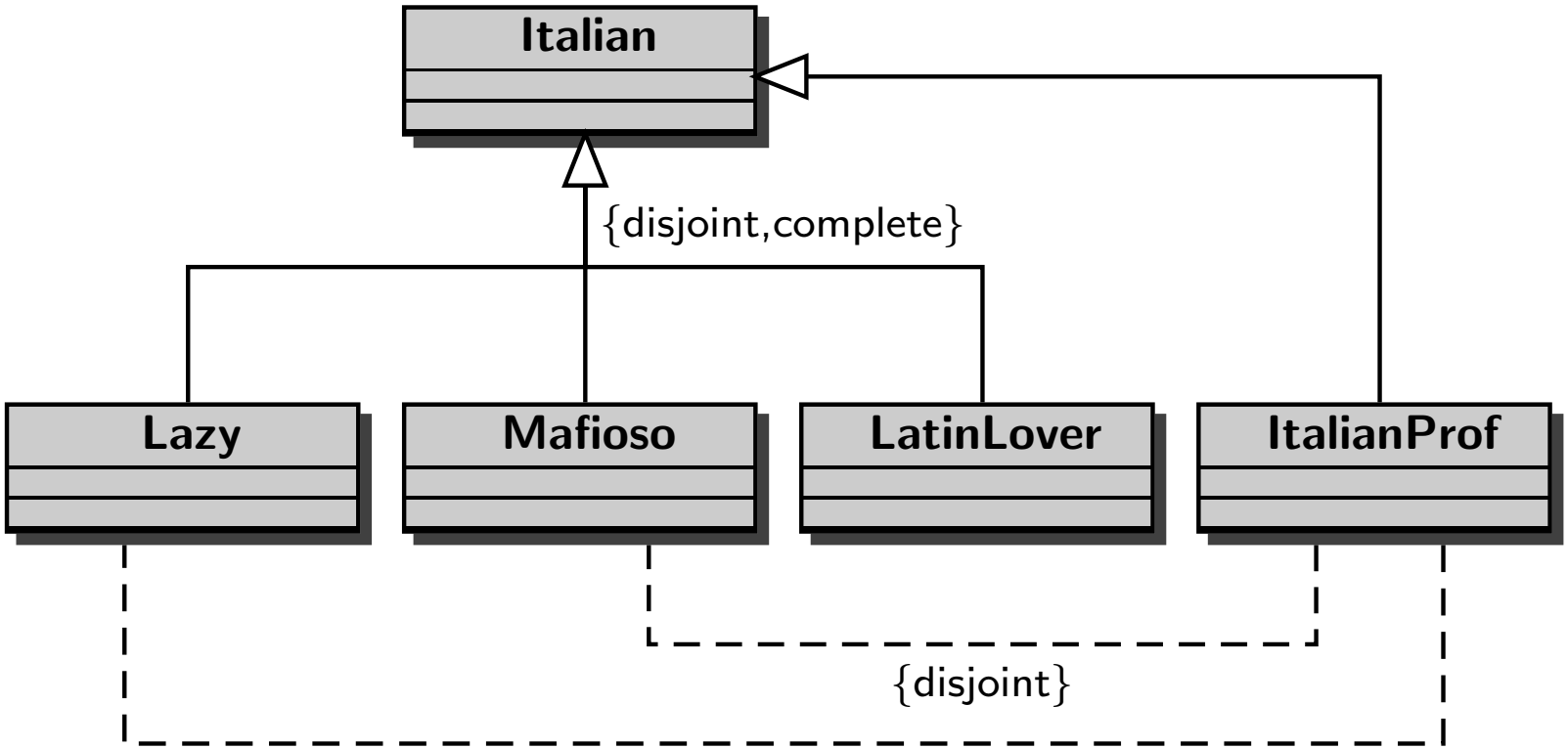


$\text{LatinLover} = \emptyset$

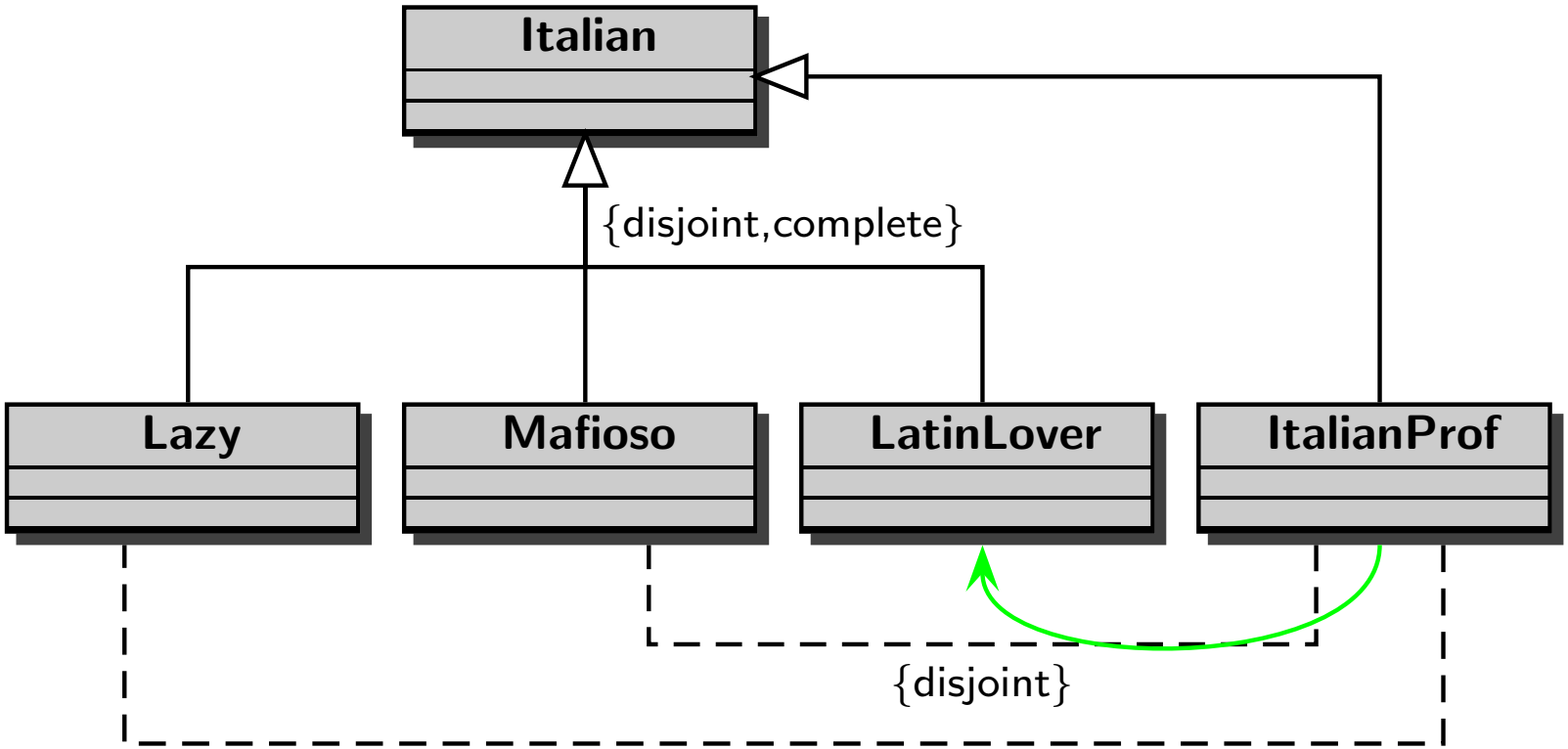
$\text{Italian} \subseteq \text{Lazy}$

$\text{Italian} \equiv \text{Lazy}$

Reasoning: cute professors



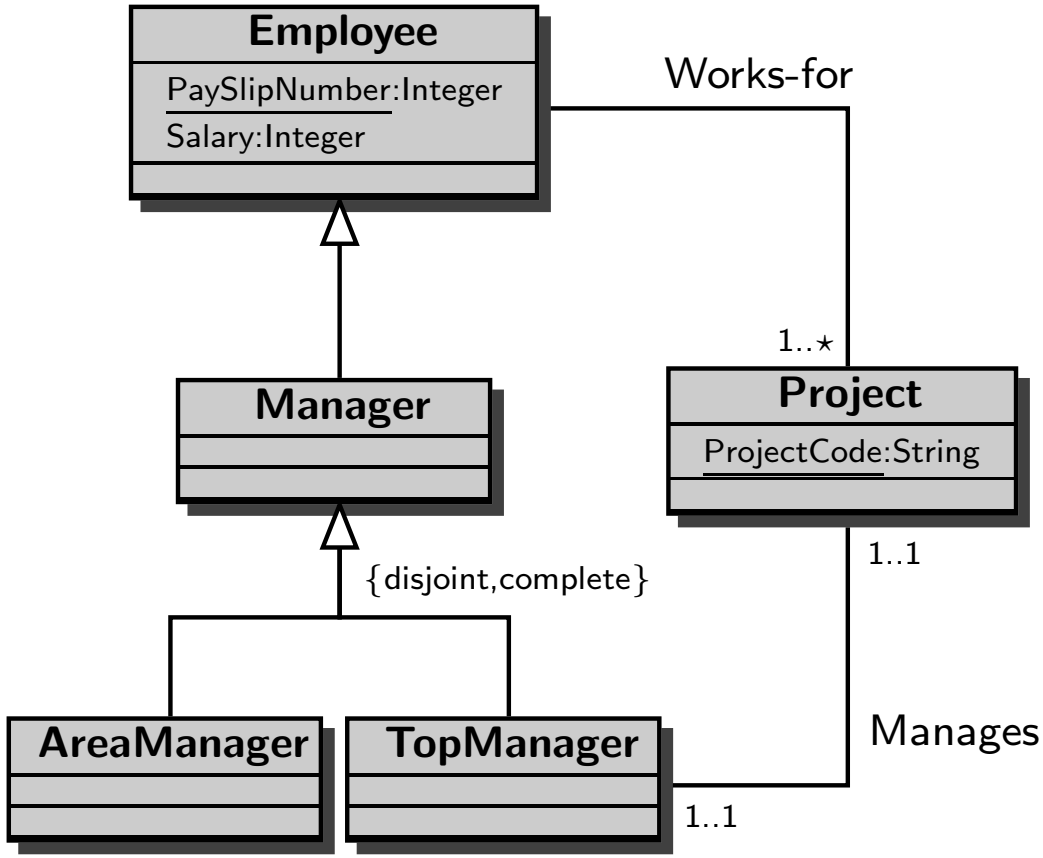
Reasoning: cute professors



implies

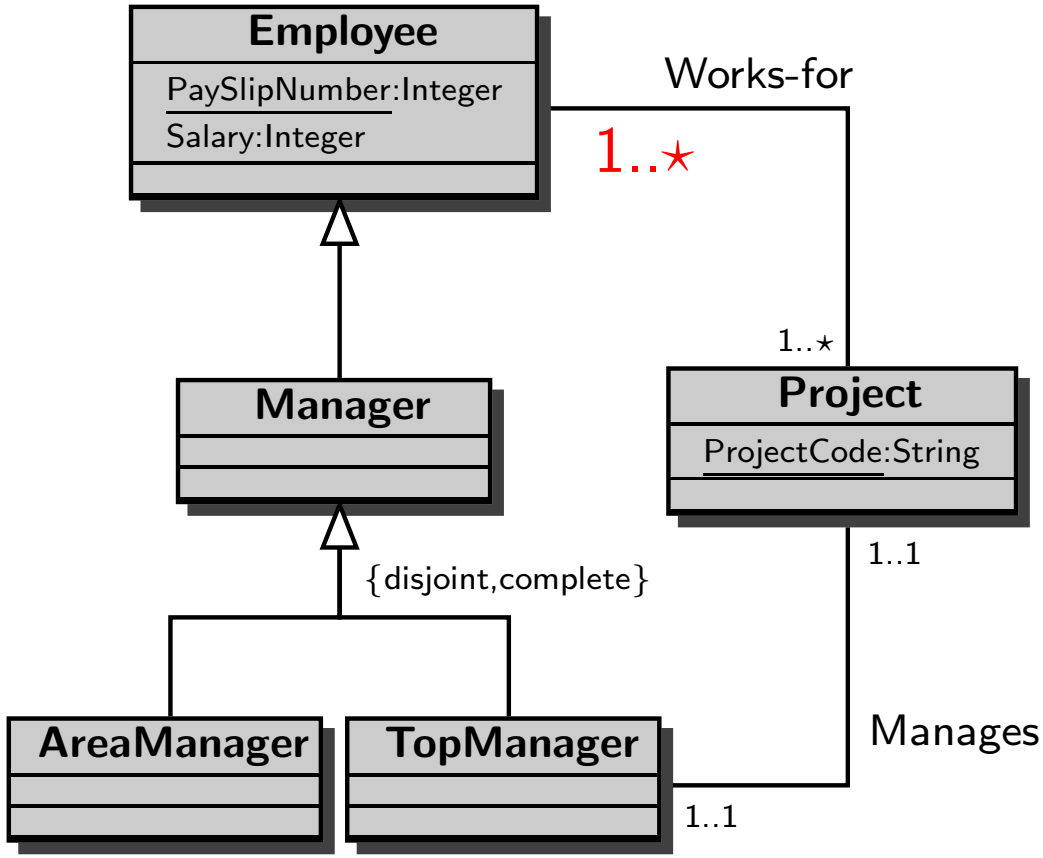
ItalianProf $\subseteq$ LatinLover

Reasoning with Conceptual Schemas



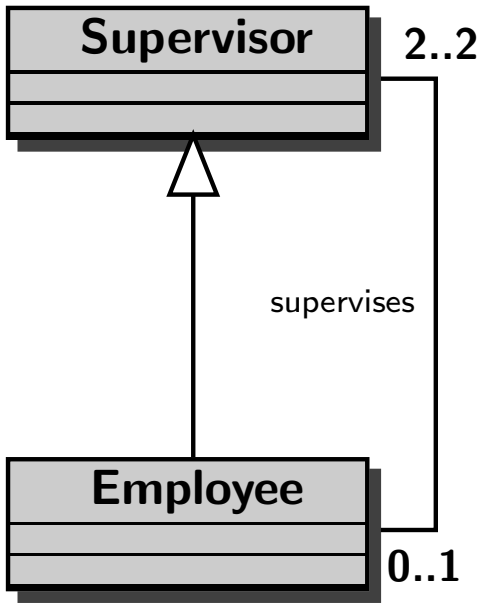
- Managers do not work for a project (she/he just manages it):
  $\forall x.\texttt{Manager}(x) \rightarrow \forall y.\neg\texttt{WORKS-FOR}(x, y)$

Reasoning with Conceptual Schemas

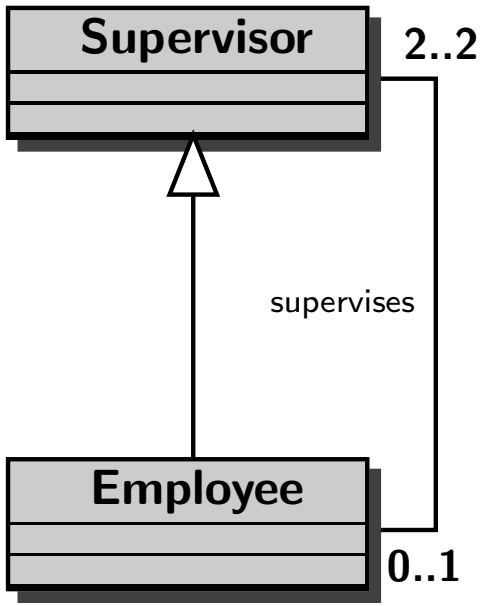


- Managers do not work for a project (she/he just manages it):
  $\forall x.\texttt{Manager}(x) \rightarrow \forall y.\neg\texttt{WORKS-FOR}(x, y)$
- If the minimum cardinality for the participation of employees to the *works-for* relationship is increased, then ...

Infinite Domain: the democratic company



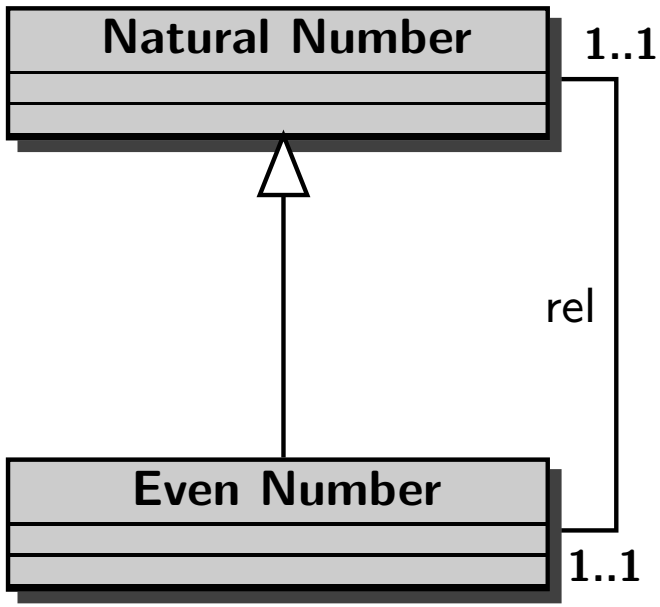
Infinite Domain: the democratic company



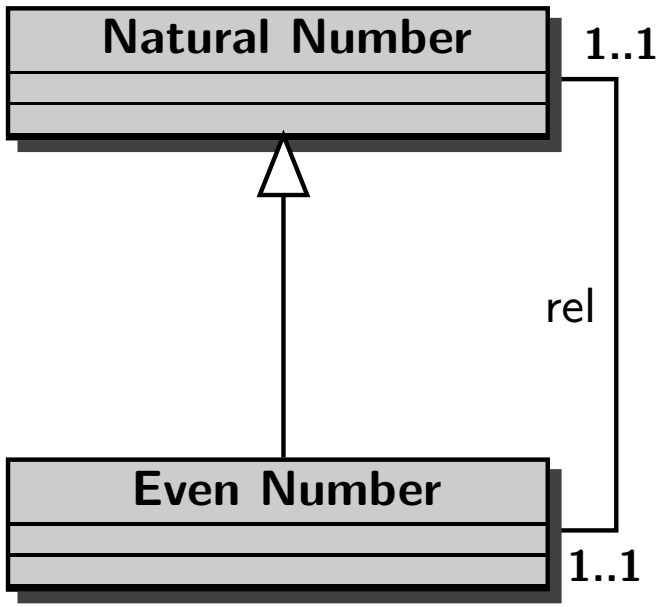
implies
"the classes Employee and Supervisor necessarily contain an infinite number of instances".

Since legal world descriptions are *finite* possible worlds satisfying the constraints imposed by the conceptual schema, the schema is inconsistent.

Bijection: how many numbers



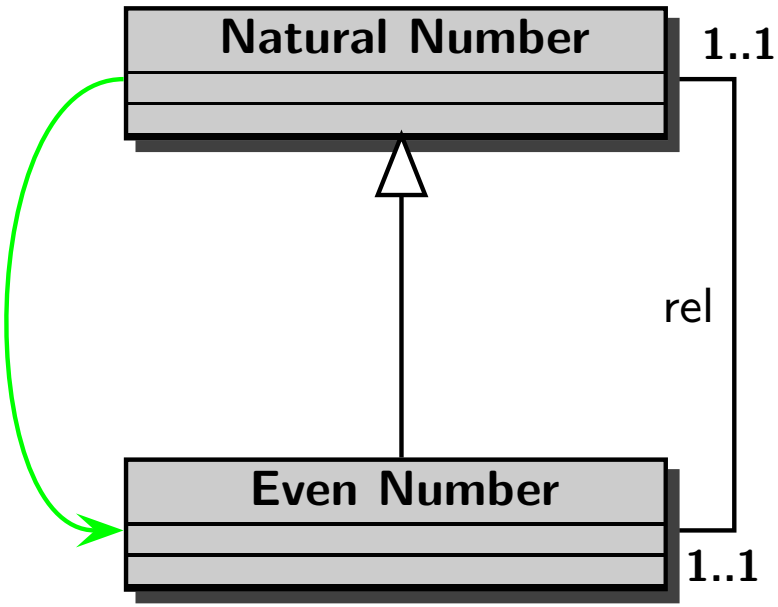
Bijection: how many numbers



implies

"the classes Natural Number and Even Number contain the same number of instances".

Bijection: how many numbers



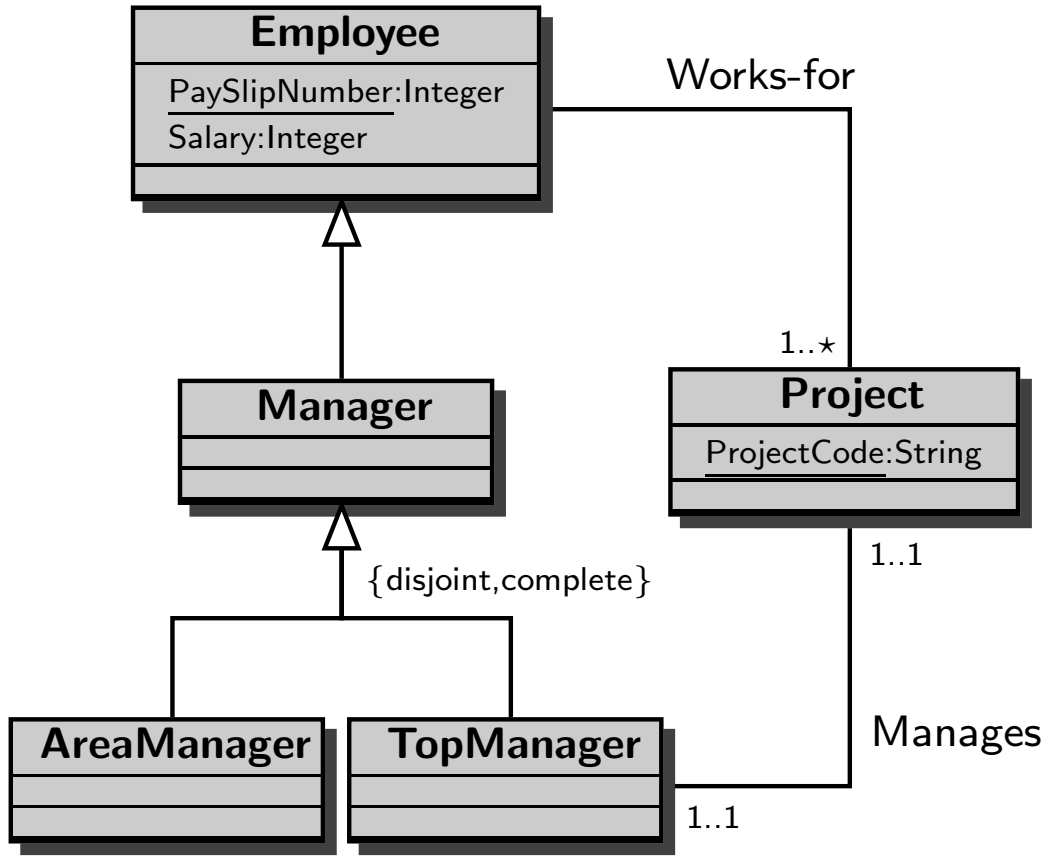
implies

"the classes Natural Number and Even Number contain the same number of instances".

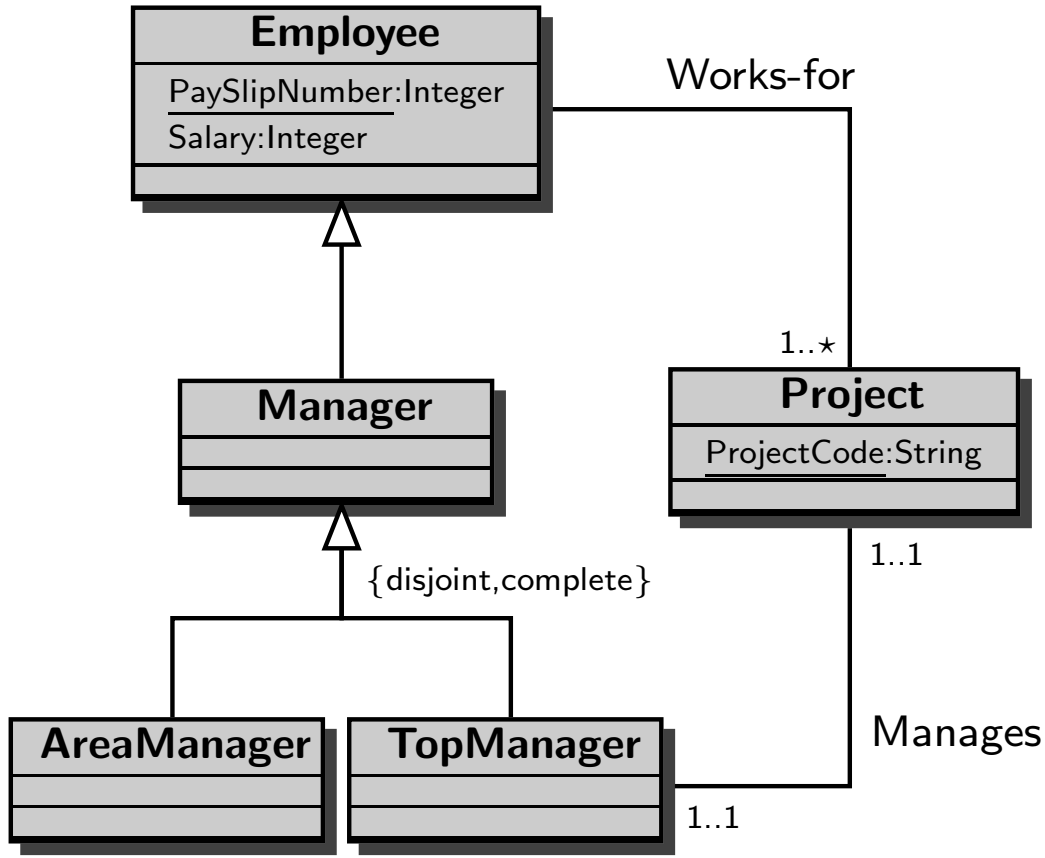
Only if the domain is finite: Natural Number $\equiv$ Even Number

Encoding Conceptual Schemas in Description Logics (DL) / OWL

- Object-oriented data models (e.g., UML and ODMG)
- Semantic data models (e.g., EER and ORM)
- Knowledge Graphs and
  web ontology languages (e.g., RDF/S and OWL)
- Theorems **prove** that a conceptual schema and its encoding as DL knowledge bases constrain every world description in the same way – i.e., the models of the DL theory correspond to the legal world descriptions of the conceptual schema, and vice-versa.



$$
\begin{array}{lcl}
\texttt{Works-for} & \sqsubseteq & \texttt{emp/2} : \texttt{Employee} \sqcap \texttt{act/2} : \texttt{Project} \\
\texttt{Manages} & \sqsubseteq & \texttt{man/2} : \texttt{TopManager} \sqcap \texttt{prj/2} : \texttt{Project} \\
\texttt{Employee} & \sqsubseteq & \exists^{=1}[\texttt{worker}](\texttt{PaySlipNumber} \sqcap \texttt{num/2} : \texttt{Integer}) \sqcap \\
 & & \exists^{=1}[\texttt{payee}](\texttt{Salary} \sqcap \texttt{amount/2} : \texttt{Integer}) \\
\top & \sqsubseteq & \exists^{\leq 1}[\texttt{num}](\texttt{PaySlipNumber} \sqcap \texttt{worker/2} : \texttt{Employee}) \\
\texttt{Manager} & \sqsubseteq & \texttt{Employee} \sqcap (\texttt{AreaManager} \sqcup \texttt{TopManager}) \\
\texttt{AreaManager} & \sqsubseteq & \texttt{Manager} \sqcap \neg\texttt{TopManager} \\
\texttt{TopManager} & \sqsubseteq & \texttt{Manager} \sqcap \exists^{=1}[\texttt{man}]\texttt{Manages} \\
\texttt{Project} & \sqsubseteq & \exists^{\geq 1}[\texttt{act}]\texttt{Works-for} \sqcap \exists^{=1}[\texttt{prj}]\texttt{Manages} \\
\ldots & &
\end{array}
$$



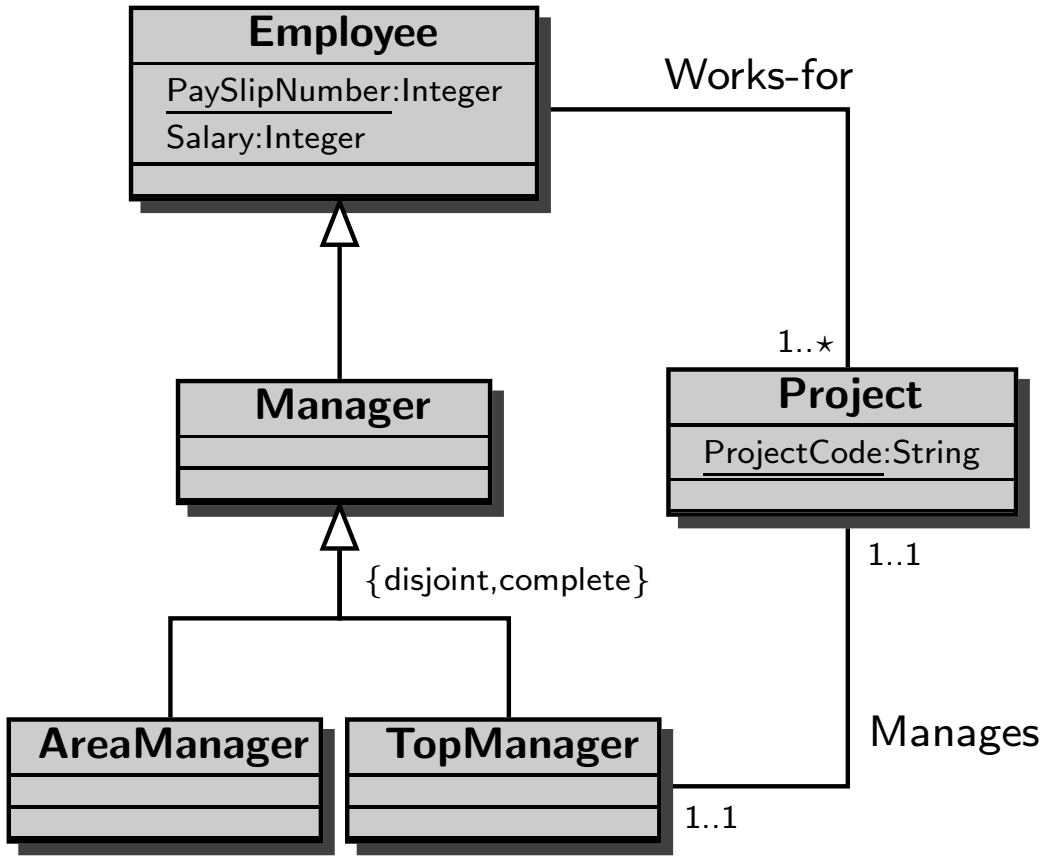
this is not OWL

$$
\begin{array}{lcl}
\texttt{Works-for} & \sqsubseteq & \texttt{emp}/2 : \texttt{Employee} \sqcap \texttt{act}/2 : \texttt{Project} \\
\texttt{Manages} & \sqsubseteq & \texttt{man}/2 : \texttt{TopManager} \sqcap \texttt{prj}/2 : \texttt{Project} \\
\texttt{Employee} & \sqsubseteq & \exists^{=1}[\texttt{worker}](\texttt{PaySlipNumber} \sqcap \texttt{num}/2 : \texttt{Integer}) \sqcap \\
 & & \exists^{=1}[\texttt{payee}](\texttt{Salary} \sqcap \texttt{amount}/2 : \texttt{Integer}) \\
\top & \sqsubseteq & \exists^{\leq 1}[\texttt{num}](\texttt{PaySlipNumber} \sqcap \texttt{worker}/2 : \texttt{Employee}) \\
\texttt{Manager} & \sqsubseteq & \texttt{Employee} \sqcap (\texttt{AreaManager} \sqcup \texttt{TopManager}) \\
\texttt{AreaManager} & \sqsubseteq & \texttt{Manager} \sqcap \neg\texttt{TopManager} \\
\texttt{TopManager} & \sqsubseteq & \texttt{Manager} \sqcap \exists^{=1}[\texttt{man}]\texttt{Manages} \\
\texttt{Project} & \sqsubseteq & \exists^{\geq 1}[\texttt{act}]\texttt{Works-for} \sqcap \exists^{=1}[\texttt{prj}]\texttt{Manages} \\
\ldots & &
\end{array}
$$

Set-based Constraints

$\text{Works-for} \subseteq \text{Employee} \times \text{Project}$
$\text{Manages} \subseteq \text{TopManager} \times \text{Project}$
$\text{Employee} \subseteq \{e \mid \sharp(\text{PaySlipNumber} \cap (\{e\} \times \texttt{Integer})) \geq 1\}$
$\text{Employee} \subseteq \{e \mid \sharp(\text{Salary} \cap (\{e\} \times \texttt{Integer})) \geq 1\}$
$\text{Project} \subseteq \{p \mid \sharp(\text{ProjectCode} \cap (\{p\} \times \texttt{String})) \geq 1\}$
$\text{TopManager} \subseteq \{m \mid 1 \geq \sharp(\text{Manages} \cap (\{m\} \times \Omega)) \geq 1\}$
$\text{Project} \subseteq \{p \mid 1 \geq \sharp(\text{Manages} \cap (\Omega \times \{p\})) \geq 1\}$
$\text{Project} \subseteq \{p \mid \sharp(\text{Works-for} \cap (\Omega \times \{p\})) \geq 1\}$
$\text{Manager} \subseteq \text{Employee}$
$\text{AreaManager} \subseteq \text{Manager}$
$\text{TopManager} \subseteq \text{Manager}$
$\text{AreaManager} \cap \text{TopManager} = \emptyset$
$\text{Manager} \subseteq \text{AreaManager} \cup \text{TopManager}$

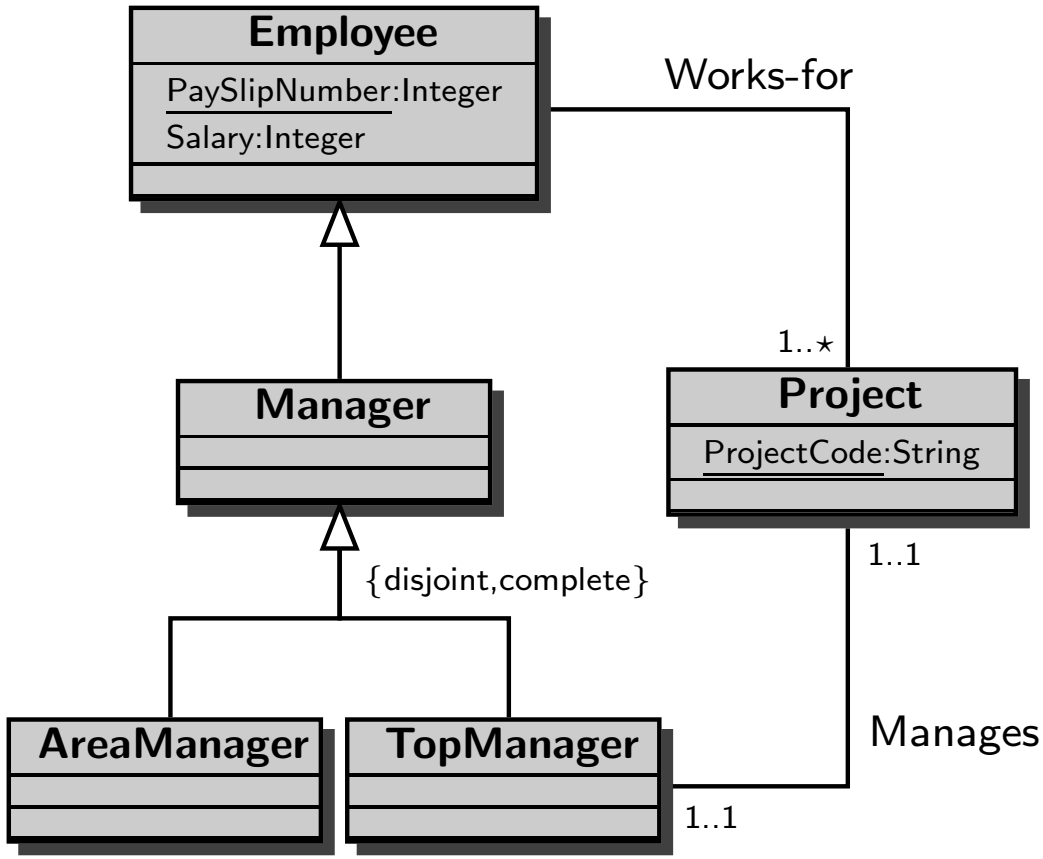
Deducing constraints



Managers are employees who do not work for a project (she/he just manages it):

$\texttt{Employee} \sqcap \neg(\exists^{\geq 1}[\texttt{emp}]\texttt{Works-for}) \sqsubseteq \texttt{Manager}, \quad \texttt{Manager} \sqsubseteq \neg(\exists^{\geq 1}[\texttt{emp}]\texttt{Works-for})$

Deducing constraints



Managers are employees who do not work for a project (she/he just manages it):

$\texttt{Employee} \sqcap \neg(\exists^{\geq 1}[\texttt{emp}]\texttt{Works-for}) \sqsubseteq \texttt{Manager}, \quad \texttt{Manager} \sqsubseteq \neg(\exists^{\geq 1}[\texttt{emp}]\texttt{Works-for})$

$\models$ For every project, there is at least one employee who is not a manager:

$\texttt{Project} \sqsubseteq \exists^{\geq 1}[\texttt{act}](\texttt{Works-for} \sqcap \texttt{emp} : \neg\texttt{Manager})$

i•com: Intelligent Conceptual Modelling

- i•com allows for the specification of multiple EER (or UML) diagrams and inter- and intra-schema constraints;
- Complete logical reasoning is employed by the tool using a hidden underlying $\mathcal{DLR}$ inference engine;
- i•com verifies the specification, infers implicit facts and stricter constraints, and manifests any inconsistencies during the conceptual modelling phase.

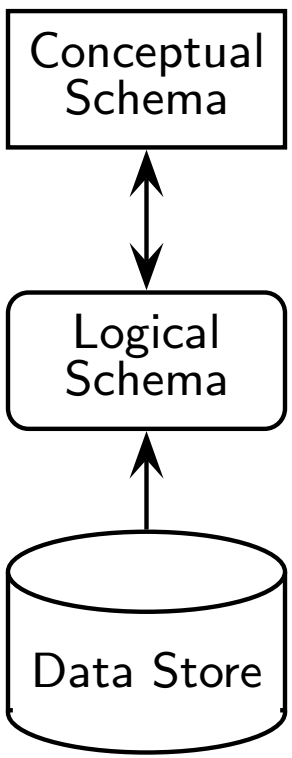
The agenda

- Description Logics are the backbone of semantic technologies
- They form the basis of trust
- But knowledge engineers should use conceptual modelling languages to design, manage, and use ontologies
- These languages are closer to the way we abstract the world in our cognition
- They are equipped with decades of experience of modelling: methodologies

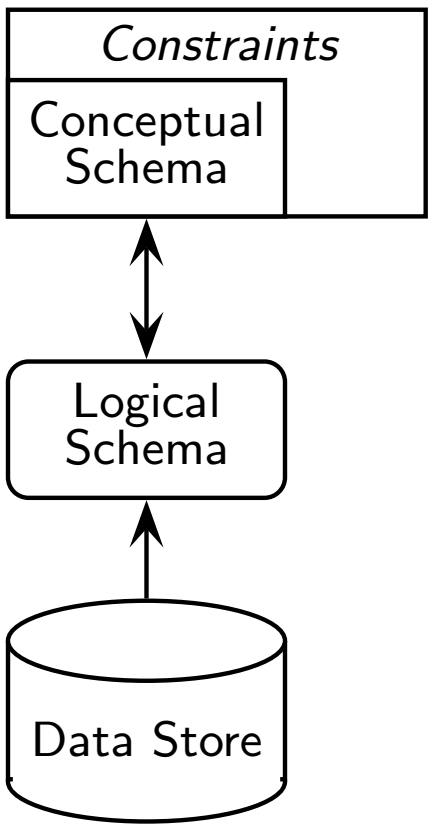
Summary

- Logic and Conceptual Modelling
- Description Logics for Conceptual Modelling
- Queries via a Conceptual Schema

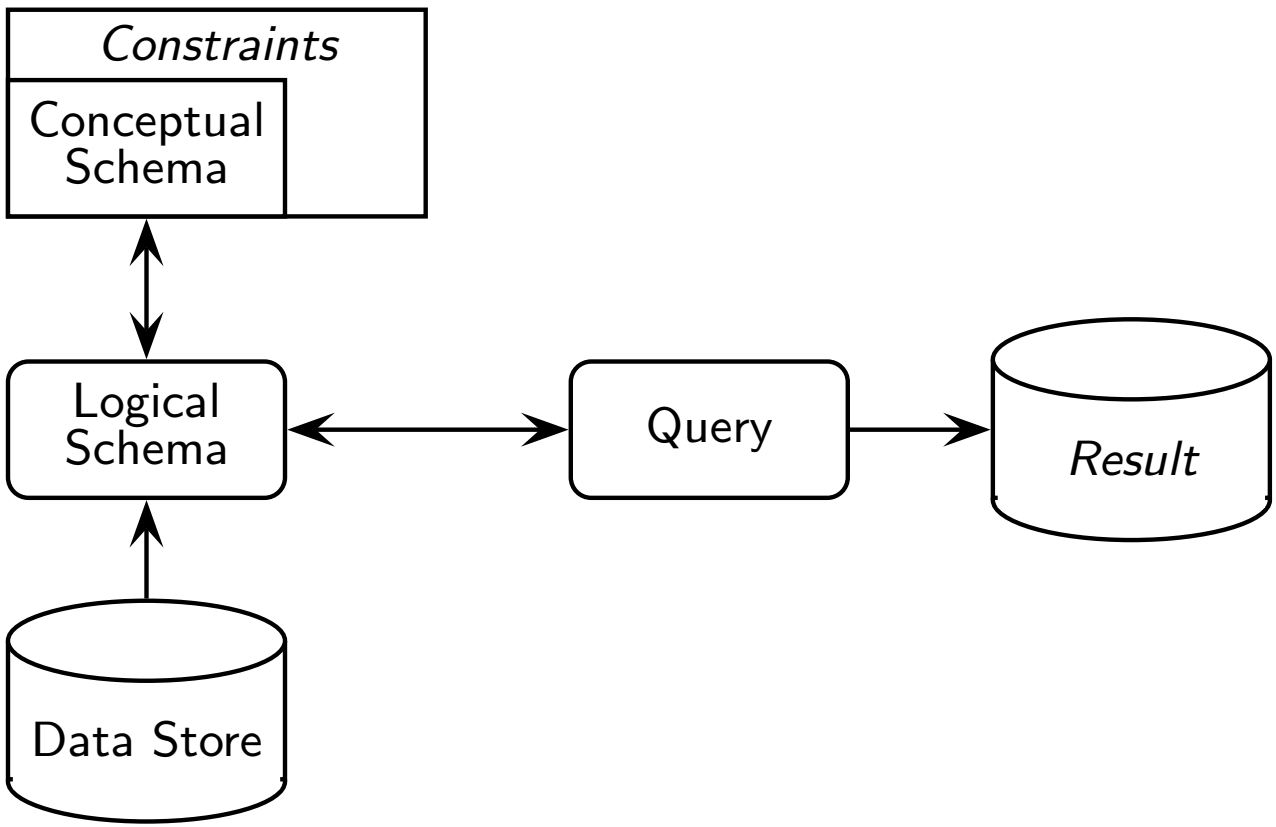
The role of a Conceptual Schema – revisited



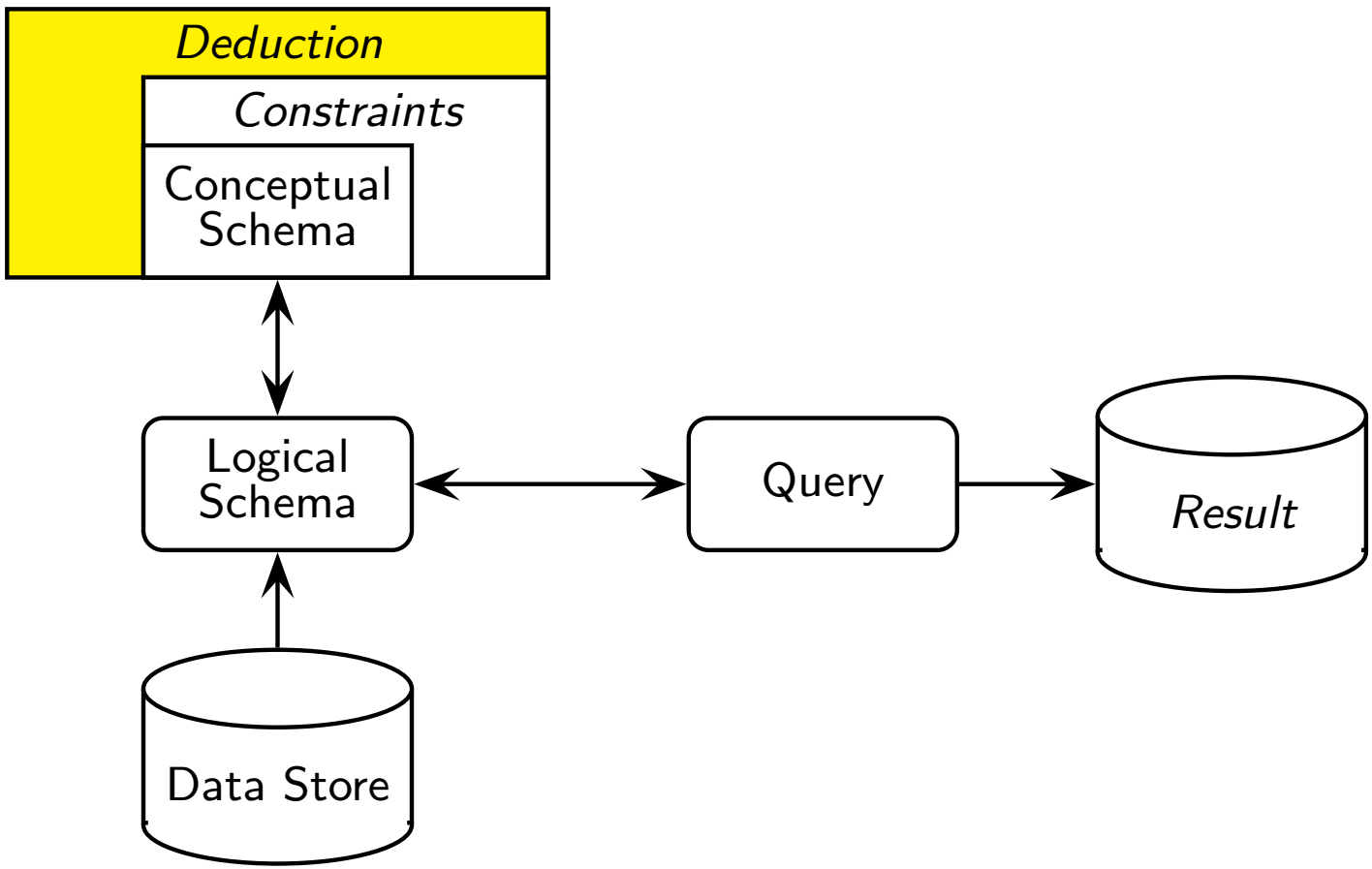
The role of a Conceptual Schema – revisited



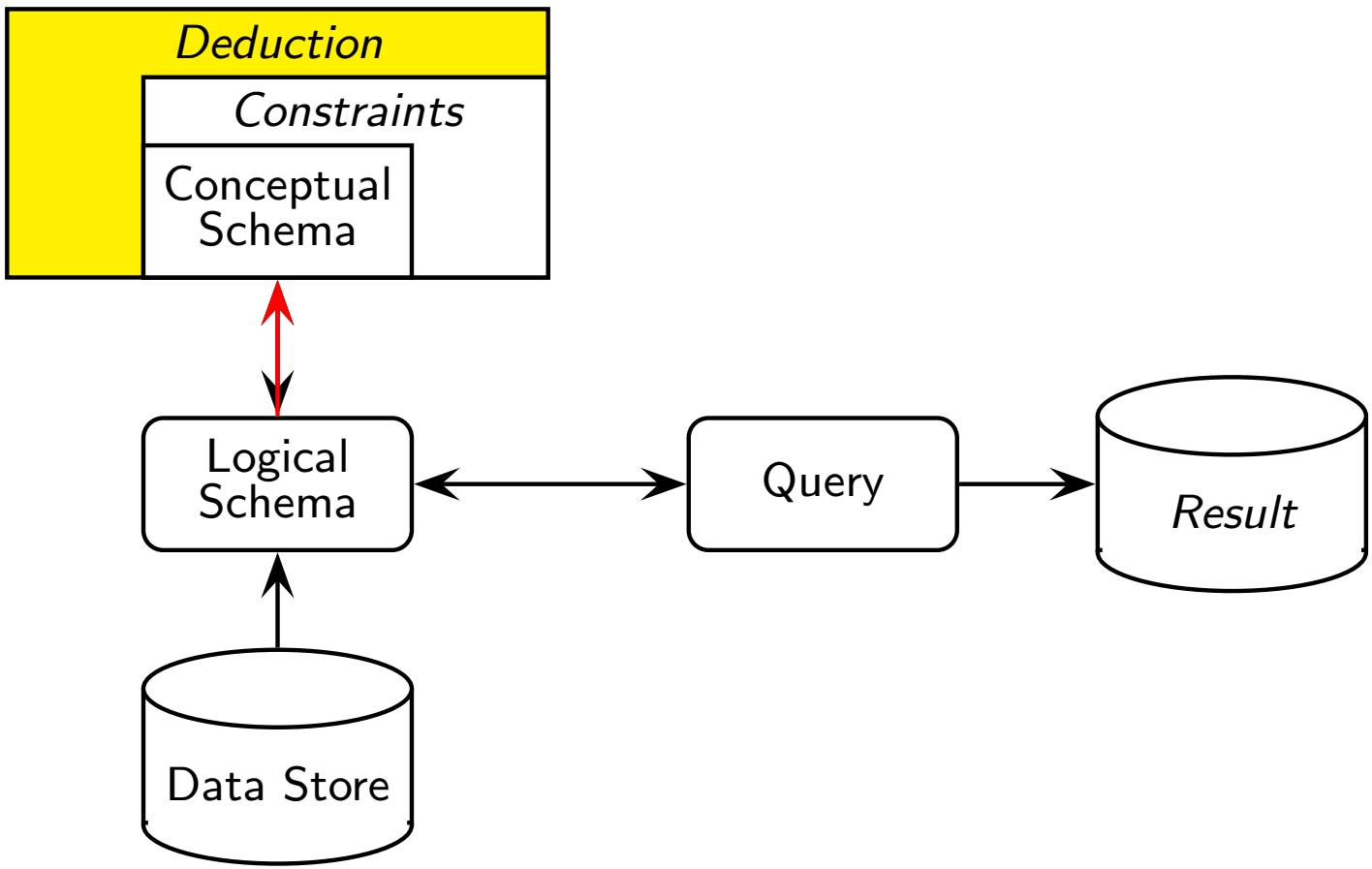
The role of a Conceptual Schema – revisited



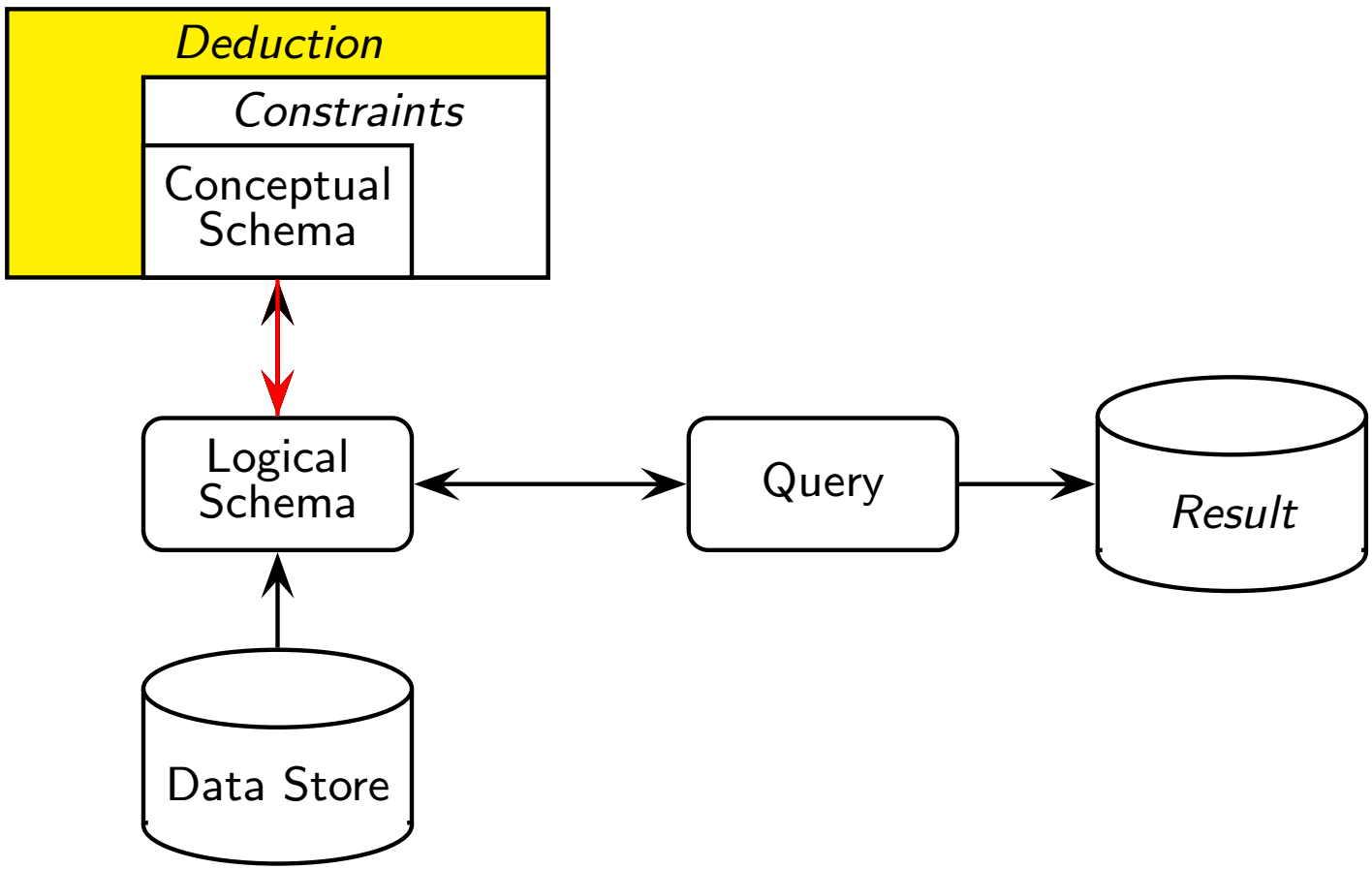
The role of a Conceptual Schema – revisited



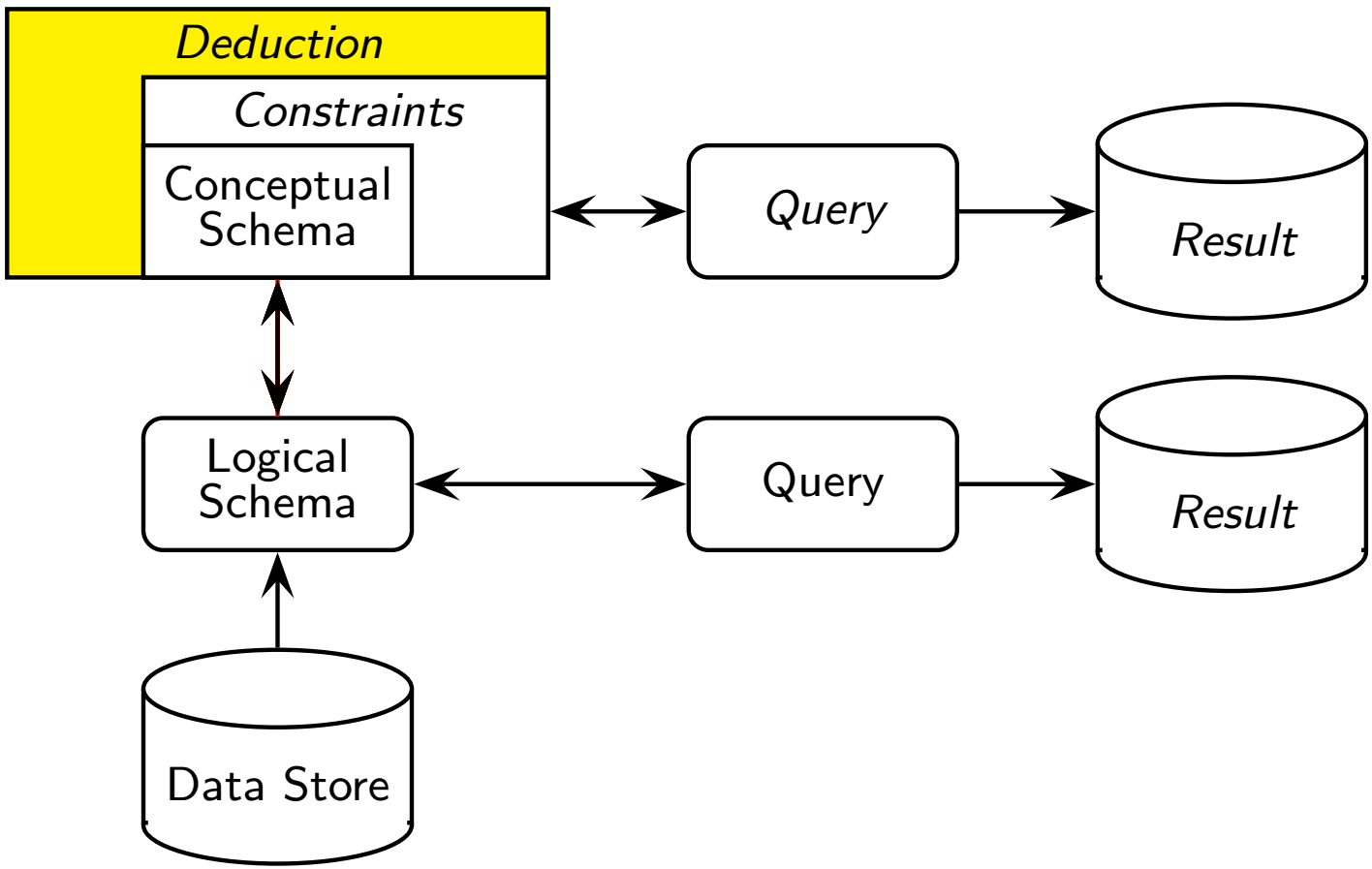
The role of a Conceptual Schema – revisited



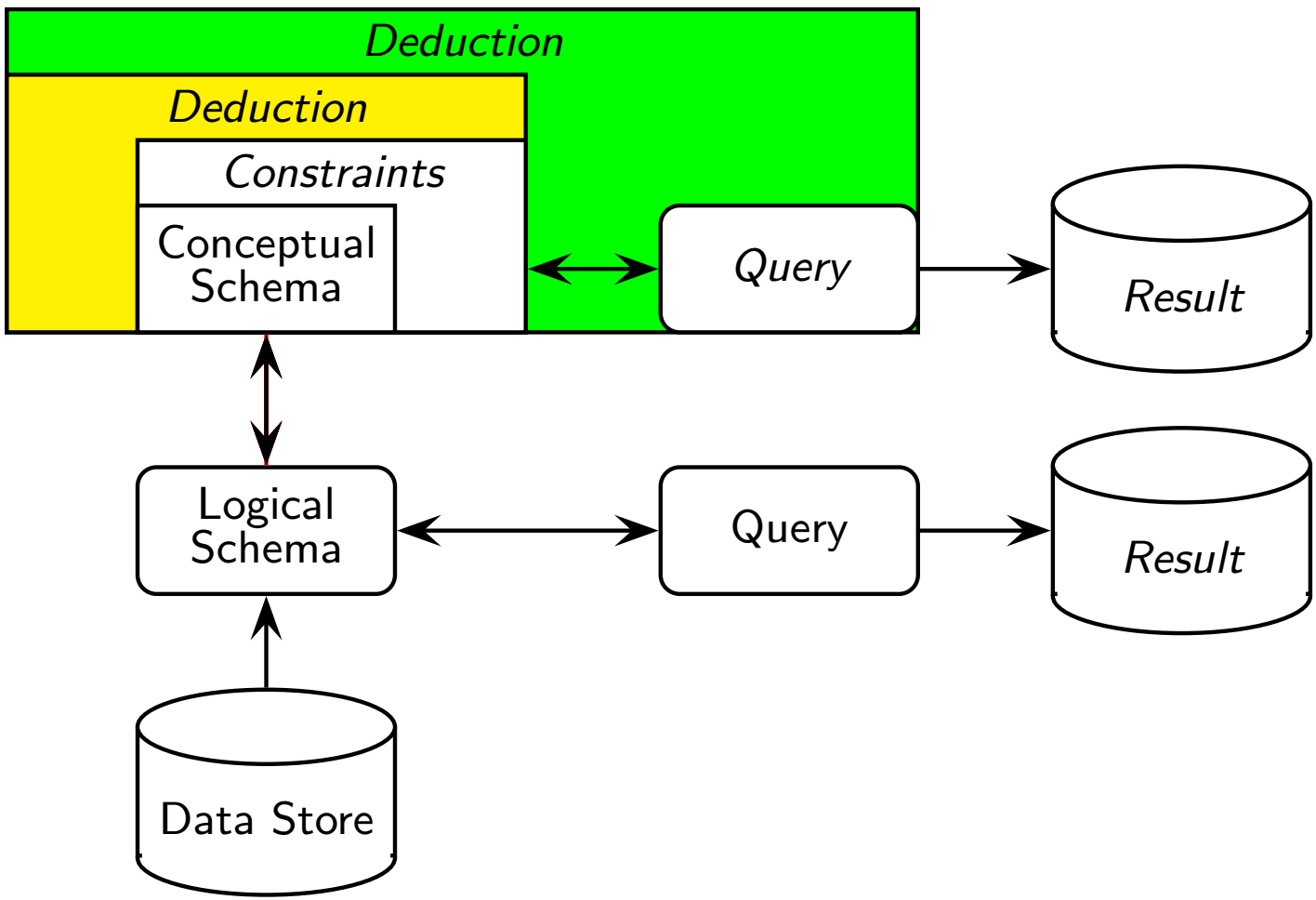
The role of a Conceptual Schema – revisited



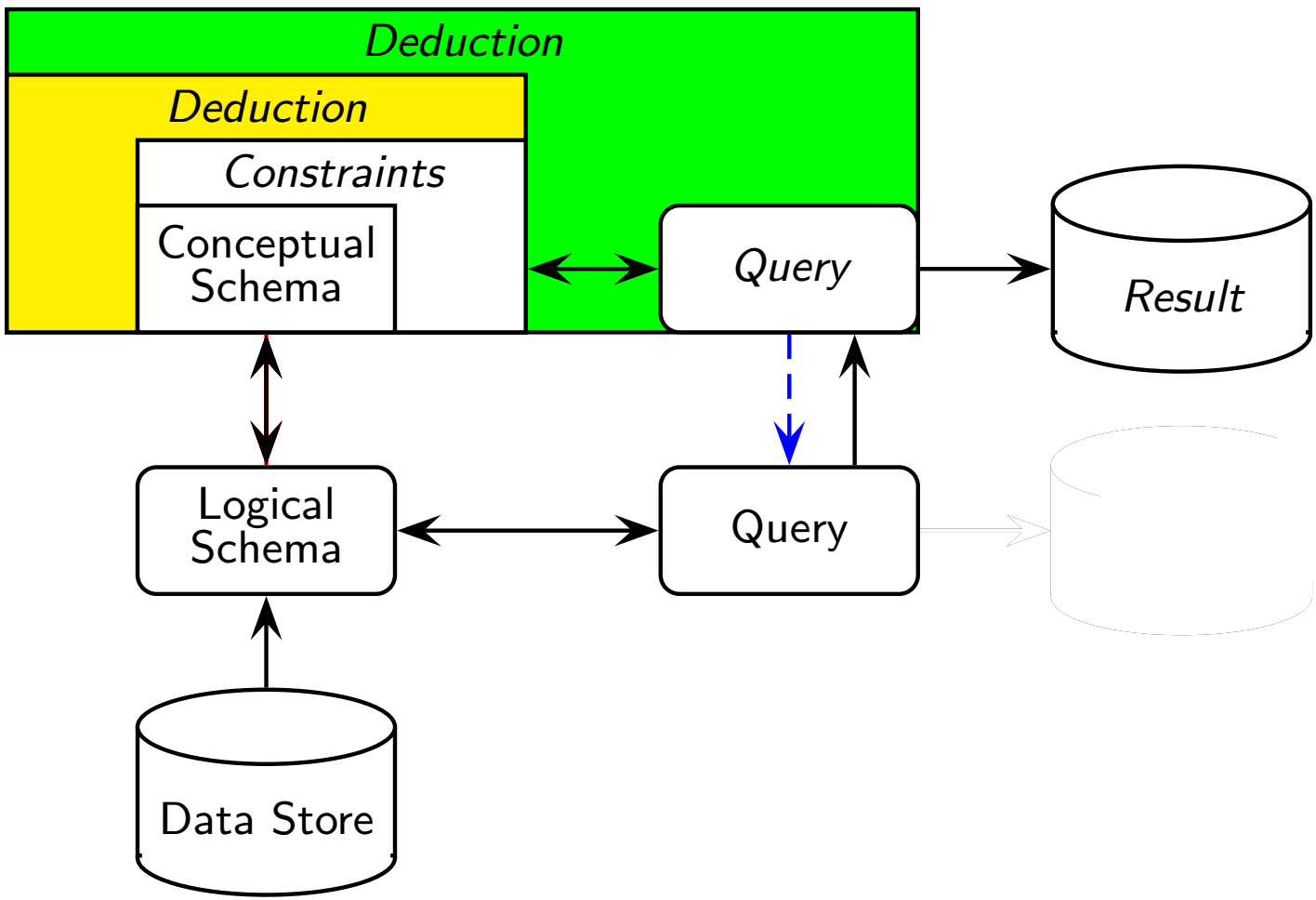
The role of a Conceptual Schema – revisited



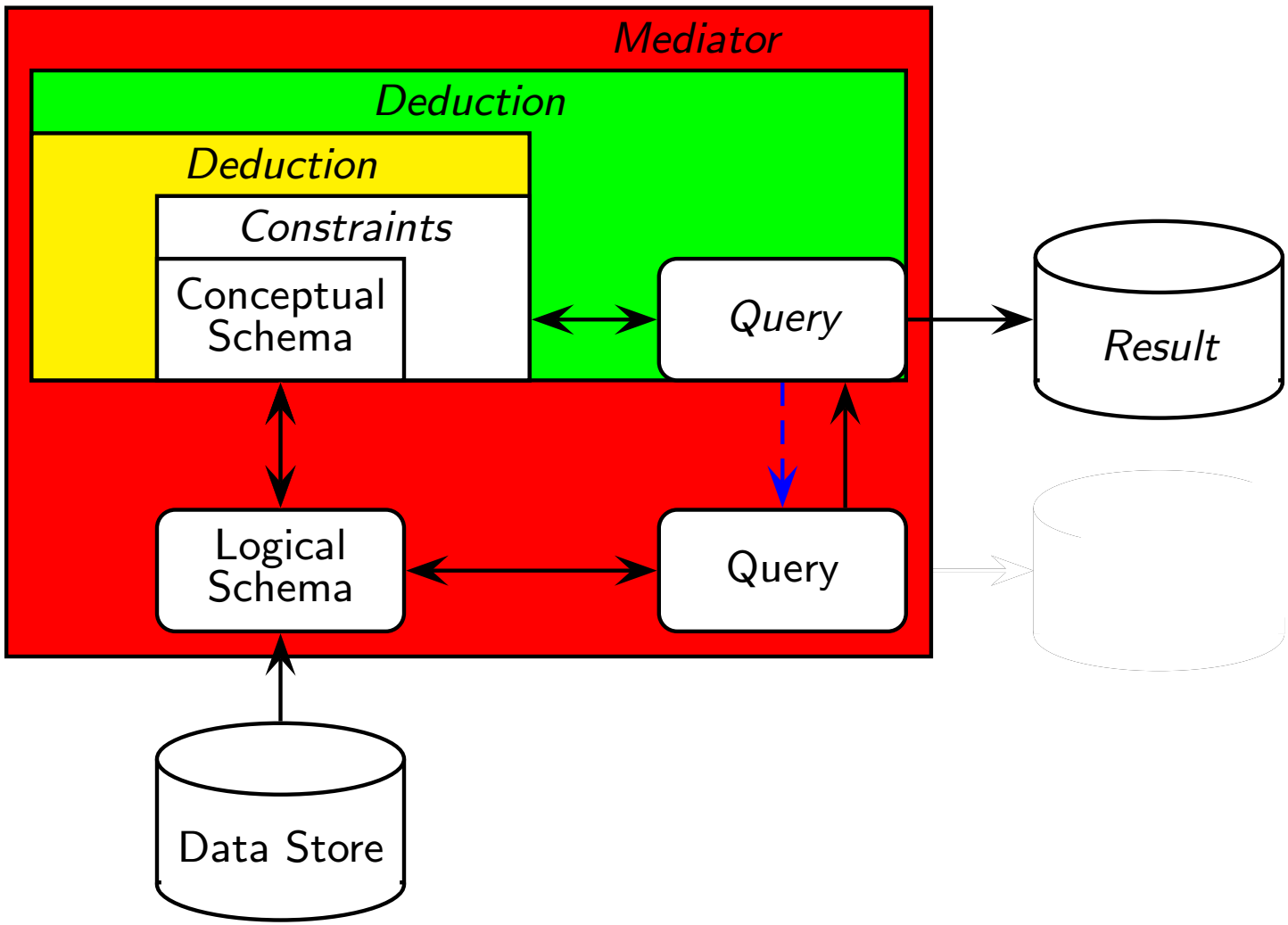
The role of a Conceptual Schema – revisited



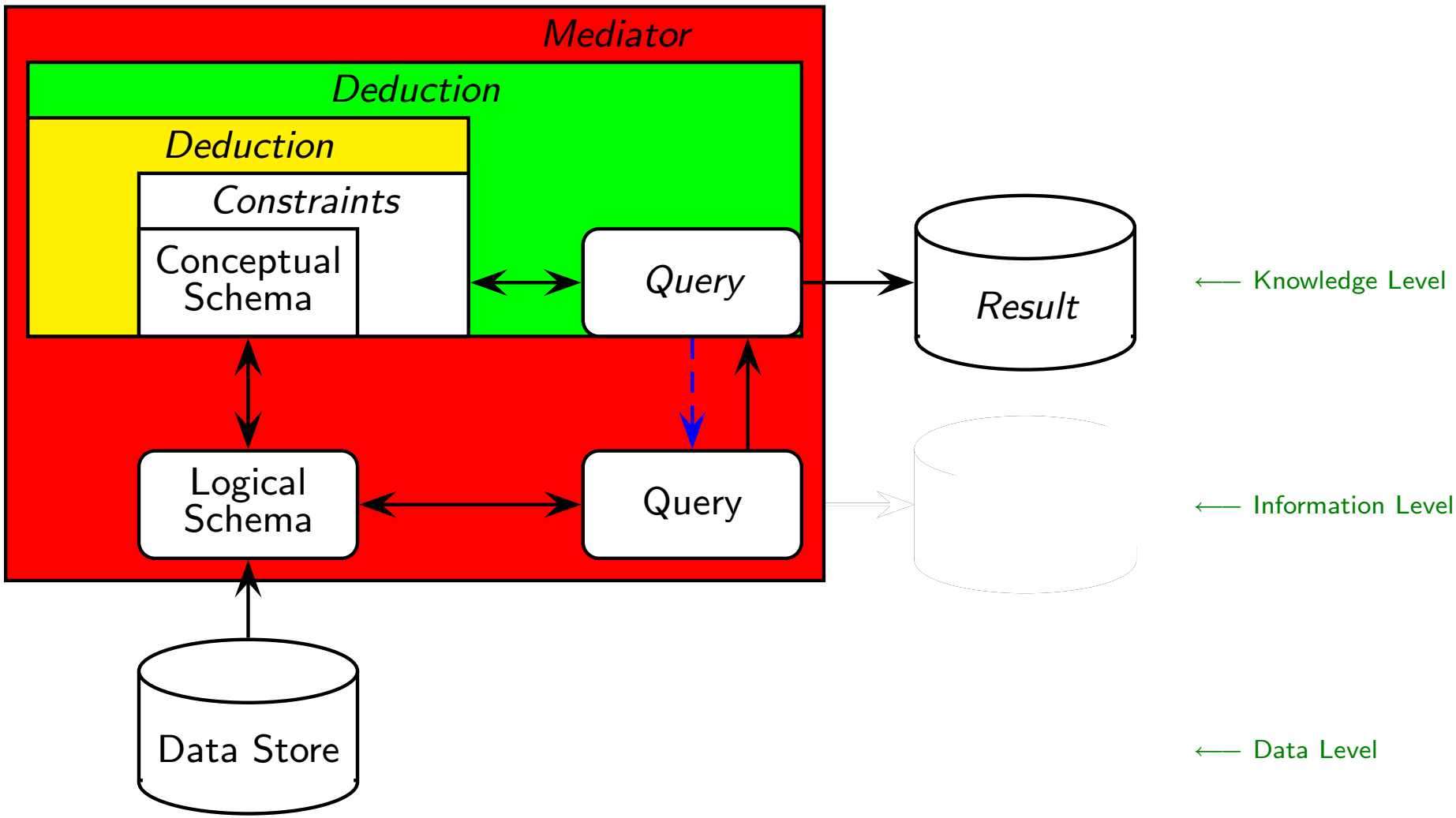
The role of a Conceptual Schema – revisited



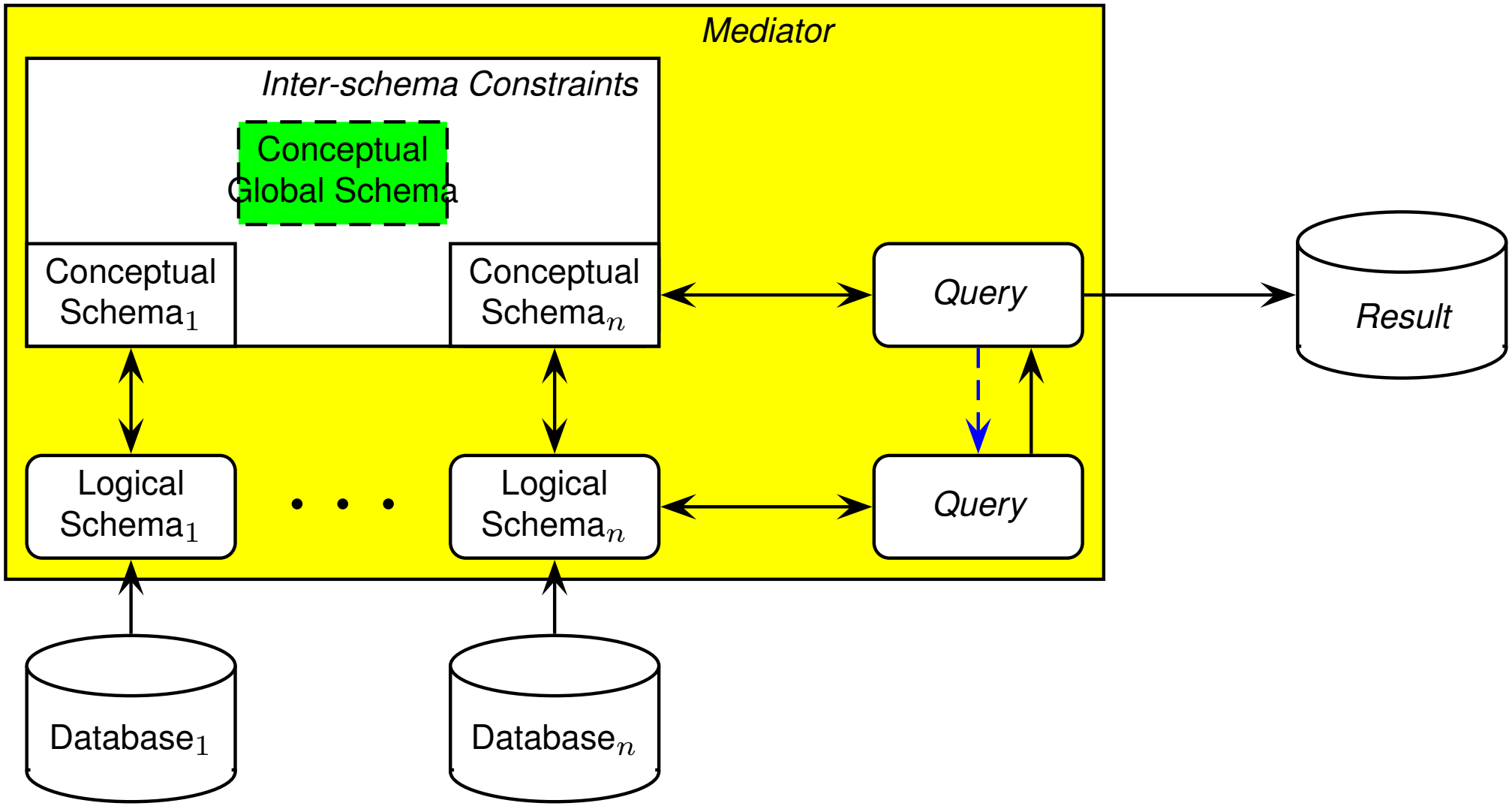
The role of a Conceptual Schema – revisited



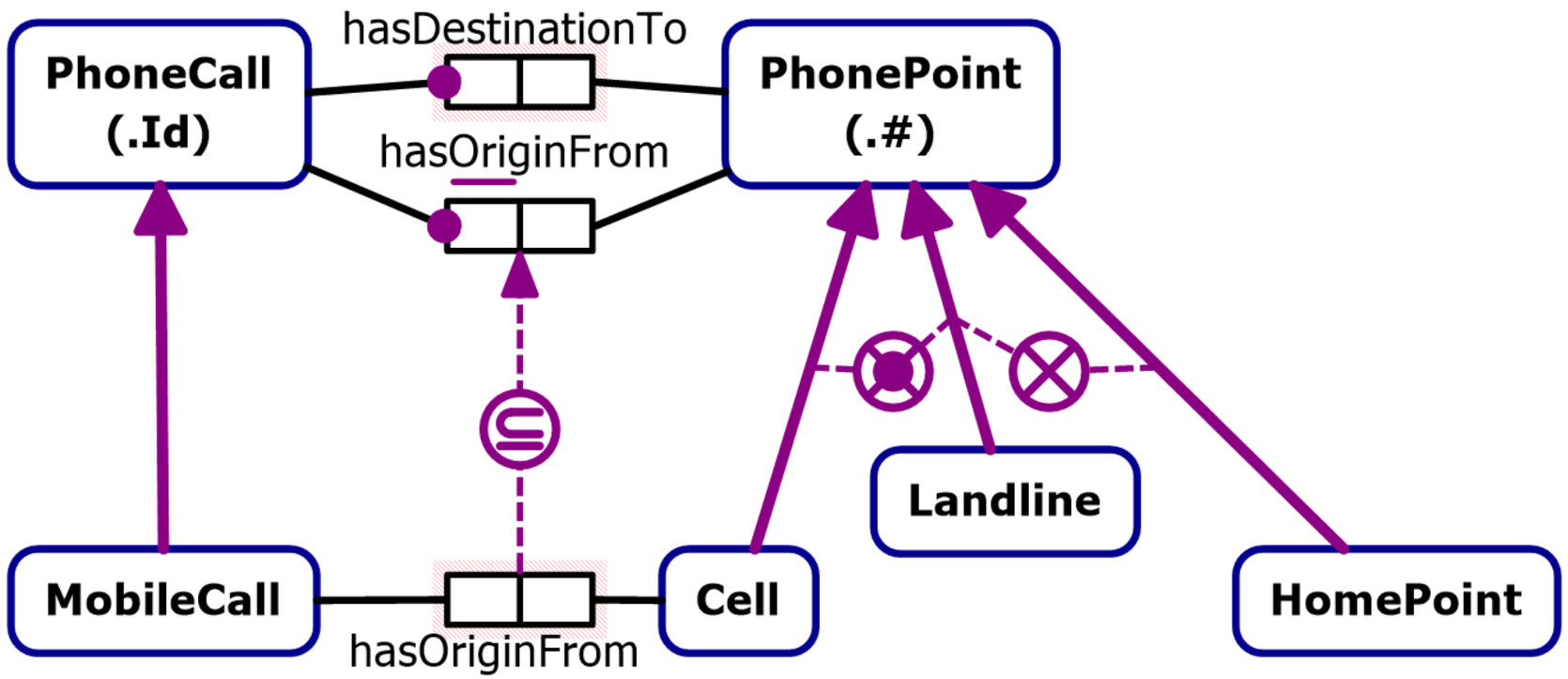
The role of a Conceptual Schema – revisited



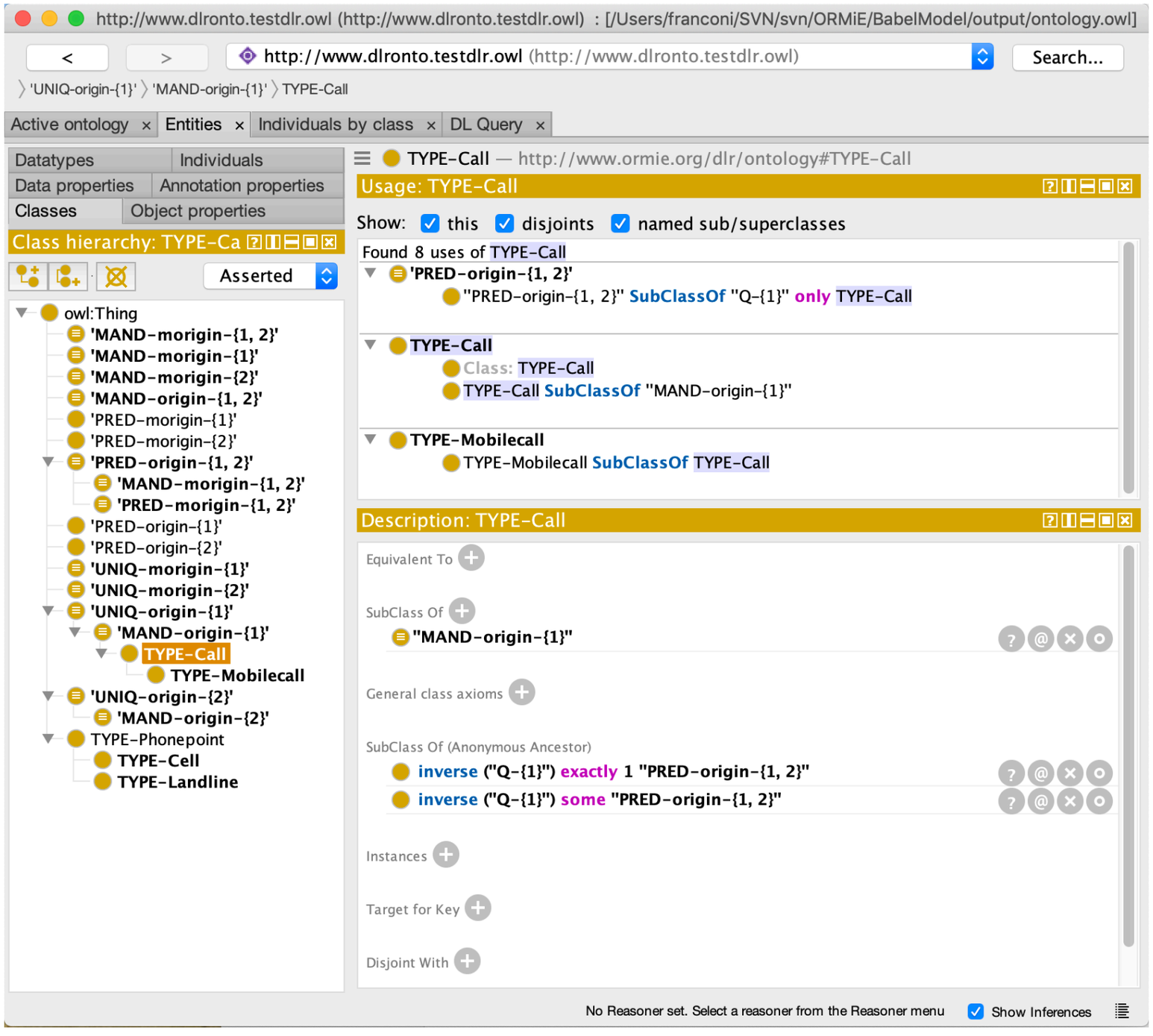
Mediator Architecture for Ontology Integration



ORM in NORMA


PhoneCall
(.Id)
hasDestinationTo
hasOriginFrom
PhonePoint
(.#)
Landline
MobileCall
Cell
hasOriginFrom
HomePoint

OWL in Protégé



```
ObjectProperty: <http://www.ormie.org/dlr/ontology#Q-{1}>
    Characteristics: Functional

ObjectProperty: <http://www.ormie.org/dlr/ontology#Q-{2}>
    Characteristics: Functional

Class: <http://www.ormie.org/dlr/ontology#MAND-morigin-{1, 2}>
    EquivalentTo:
        <http://www.ormie.org/dlr/ontology#PRED-morigin-{1, 2}>

Class: <http://www.ormie.org/dlr/ontology#MAND-morigin-{1}>
    EquivalentTo:
         inverse (<http://www.ormie.org/dlr/ontology#Q-{1}>) some <http://www.ormie.org/dlr/ontology#PRED-morigin-{1, 2}>

Class: <http://www.ormie.org/dlr/ontology#MAND-morigin-{2}>
    EquivalentTo:
         inverse (<http://www.ormie.org/dlr/ontology#Q-{2}>) some <http://www.ormie.org/dlr/ontology#PRED-morigin-{1, 2}>

Class: <http://www.ormie.org/dlr/ontology#MAND-origin-{1, 2}>
    EquivalentTo:
        <http://www.ormie.org/dlr/ontology#PRED-origin-{1, 2}>

Class: <http://www.ormie.org/dlr/ontology#MAND-origin-{1}>
    EquivalentTo:
         inverse (<http://www.ormie.org/dlr/ontology#Q-{1}>) some <http://www.ormie.org/dlr/ontology#PRED-origin-{1, 2}>
    SubClassOf:
        <http://www.ormie.org/dlr/ontology#UNIQ-origin-{1}>

Class: <http://www.ormie.org/dlr/ontology#MAND-origin-{2}>
    EquivalentTo:
         inverse (<http://www.ormie.org/dlr/ontology#Q-{2}>) some <http://www.ormie.org/dlr/ontology#PRED-origin-{1, 2}>
    SubClassOf:
        <http://www.ormie.org/dlr/ontology#UNIQ-origin-{2}>

Class: <http://www.ormie.org/dlr/ontology#PRED-morigin-{1, 2}>
    EquivalentTo:
        <http://www.ormie.org/dlr/ontology#MAND-morigin-{1, 2}>
    SubClassOf:
        <http://www.ormie.org/dlr/ontology#PRED-origin-{1, 2}>,
        <http://www.ormie.org/dlr/ontology#Q-{1}> some <http://www.ormie.org/dlr/ontology#PRED-morigin-{1}>,
        <http://www.ormie.org/dlr/ontology#Q-{2}> some <http://www.ormie.org/dlr/ontology#PRED-morigin-{2}>,
        <http://www.ormie.org/dlr/ontology#Q-{1}> only <http://www.ormie.org/dlr/ontology#TYPE-Mobilecall>,
        <http://www.ormie.org/dlr/ontology#Q-{2}> only <http://www.ormie.org/dlr/ontology#TYPE-Cell>

Class: <http://www.ormie.org/dlr/ontology#PRED-morigin-{1}>

Class: <http://www.ormie.org/dlr/ontology#PRED-morigin-{2}>

Class: <http://www.ormie.org/dlr/ontology#PRED-origin-{1, 2}>
    EquivalentTo:
        <http://www.ormie.org/dlr/ontology#MAND-origin-{1, 2}>
    SubClassOf:
        <http://www.ormie.org/dlr/ontology#Q-{1}> some <http://www.ormie.org/dlr/ontology#PRED-origin-{1}>,
        <http://www.ormie.org/dlr/ontology#Q-{2}> some <http://www.ormie.org/dlr/ontology#PRED-origin-{2}>,
        <http://www.ormie.org/dlr/ontology#Q-{1}> only <http://www.ormie.org/dlr/ontology#TYPE-Call>,
        <http://www.ormie.org/dlr/ontology#Q-{2}> only <http://www.ormie.org/dlr/ontology#TYPE-Phonepoint>

Class: <http://www.ormie.org/dlr/ontology#PRED-origin-{1}>

Class: <http://www.ormie.org/dlr/ontology#PRED-origin-{2}>

Class: <http://www.ormie.org/dlr/ontology#TYPE-Call>
    SubClassOf:
        <http://www.ormie.org/dlr/ontology#MAND-origin-{1}>

Class: <http://www.ormie.org/dlr/ontology#TYPE-Cell>
    SubClassOf:
        <http://www.ormie.org/dlr/ontology#TYPE-Phonepoint>
    DisjointWith:
        <http://www.ormie.org/dlr/ontology#TYPE-Landline>

Class: <http://www.ormie.org/dlr/ontology#TYPE-Landline>
    SubClassOf:
        <http://www.ormie.org/dlr/ontology#TYPE-Phonepoint>
    DisjointWith:
        <http://www.ormie.org/dlr/ontology#TYPE-Cell>

Class: <http://www.ormie.org/dlr/ontology#TYPE-Mobilecall>
    SubClassOf:
        <http://www.ormie.org/dlr/ontology#TYPE-Call>

Class: <http://www.ormie.org/dlr/ontology#TYPE-Phonepoint>

Class: <http://www.ormie.org/dlr/ontology#UNIQ-morigin-{1}>
    EquivalentTo:
         inverse (<http://www.ormie.org/dlr/ontology#Q-{1}>) exactly 1 <http://www.ormie.org/dlr/ontology#PRED-morigin-{1, 2}>

Class: <http://www.ormie.org/dlr/ontology#UNIQ-morigin-{2}>
    EquivalentTo:
         inverse (<http://www.ormie.org/dlr/ontology#Q-{2}>) exactly 1 <http://www.ormie.org/dlr/ontology#PRED-morigin-{1, 2}>

Class: <http://www.ormie.org/dlr/ontology#UNIQ-origin-{1}>
    EquivalentTo:
         inverse (<http://www.ormie.org/dlr/ontology#Q-{1}>) exactly 1 <http://www.ormie.org/dlr/ontology#PRED-origin-{1, 2}>

Class: <http://www.ormie.org/dlr/ontology#UNIQ-origin-{2}>
    EquivalentTo:
         inverse (<http://www.ormie.org/dlr/ontology#Q-{2}>) exactly 1 <http://www.ormie.org/dlr/ontology#PRED-origin-{1, 2}>
```

OWL Manchester syntax

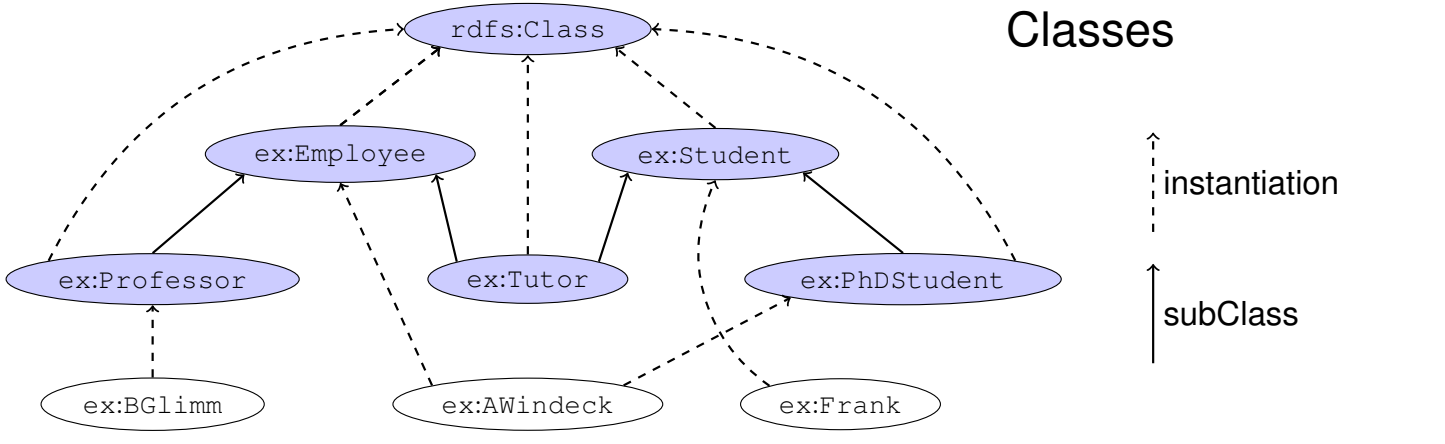
telecom-orm-xml.xml

Ontology

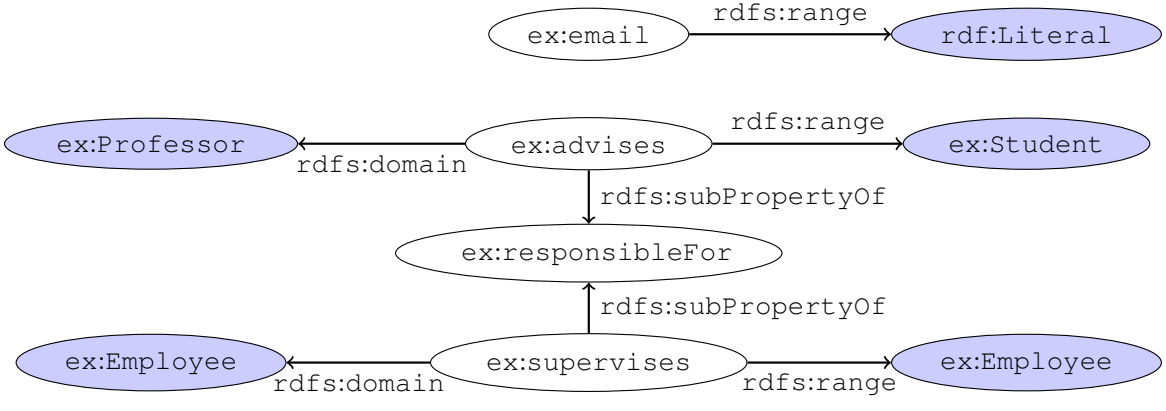
```xml
<?xml version="1.0"?>
<Ontology xmlns="http://www.w3.org/2002/07/owl#"
     xml:base="http://www.dlronto.testdlr.owl"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     ontologyIRI="http://www.dlronto.testdlr.owl">
    <Prefix name="" IRI="http://www.dlronto.testdlr.owl#"/>
    <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
    <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
    <Prefix name="xml" IRI="http://www.w3.org/XML/1998/namespace"/>
    <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
    <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>
    <Declaration>
        <Class IRI="http://www.ormie.org/dlr/ontology#TYPE-Call"/>
    </Declaration>
    <Declaration>
        <Class IRI="http://www.ormie.org/dlr/ontology#TYPE-Cell"/>
    </Declaration>
    <Declaration>
        <Class IRI="http://www.ormie.org/dlr/ontology#TYPE-Landline"/>
    </Declaration>
    <Declaration>
        <Class IRI="http://www.ormie.org/dlr/ontology#TYPE-Mobilecall"/>
    </Declaration>
    <Declaration>
        <Class IRI="http://www.ormie.org/dlr/ontology#TYPE-Phonepoint"/>
    </Declaration>
    <Declaration>
        <Class IRI="http://www.ormie.org/dlr/ontology#MAND-morigin-{1, 2}"/>
    </Declaration>
    <Declaration>
        <Class IRI="http://www.ormie.org/dlr/ontology#MAND-morigin-{1}"/>
    </Declaration>
    <Declaration>
        <Class IRI="http://www.ormie.org/dlr/ontology#MAND-morigin-{2}"/>
    </Declaration>
    <Declaration>
        <Class IRI="http://www.ormie.org/dlr/ontology#MAND-origin-{1, 2}"/>
    </Declaration>
    <Declaration>
        <Class IRI="http://www.ormie.org/dlr/ontology#MAND-origin-{1}"/>
    </Declaration>
    <Declaration>
        <Class IRI="http://www.ormie.org/dlr/ontology#MAND-origin-{2}"/>
    </Declaration>
    <Declaration>
        <Class IRI="http://www.ormie.org/dlr/ontology#PRED-morigin-{1, 2}"/>
    </Declaration>
    <Declaration>
```

OWL - XML

RDFS – Simple Ontologies



Relations



Classes, Roles and Individuals

Three building blocks of ontology axioms

- classes
  -
- individuals
  -
- roles
  -

Classes

Definition

- `<owl:Class rdf:about ="Professor"/>`
- equivalent to

```
<rdf:Description rdf:about="Professor">
  <rdf:type
    rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
</rdf:Description>
```

Pre-defined

- `owl:Thing`
- `owl:Nothing`

Individuals

Definition via class membership

```
<rdf:Description rdf:about="francescoRicci">
  <rdf:type rdf:resource="Professor"/>
</rdf:Description>
```

equivalent:

```
<Professor rdf:about="francescoRicci"/>
```

Abstract Roles (= Object Properties)

Abstract roles are defined in a way similar to classes

```
<owl:ObjectProperty rdf:about="hasAffiliation" />
```

Abstract roles connect individuals

Domain and range of abstract roles

```
<owl:ObjectProperty rdf:about="hasAffiliation">
  <rdfs:domain rdf:resource="Person" />
  <rdfs:range rdf:resource="Organization" />
</owl:ObjectProperty>
```

Concrete Roles (= Datatype Properties)

Concrete roles have datatypes as range

```
<owl:DatatypeProperty rdf:about="firstName" />
```

Concrete roles connect individuals with data values

Domain and range of concrete roles

```
<owl:DatatypeProperty rdf:about="firstName">
  <rdfs:domain rdf:resource="Person" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
```

Many XML datatypes can be used

Individuals and Roles

```
<Person rdf:about="francescoRicci">
   <hasAffiliation rdf:resource="unibz" />
   <hasAffiliation rdf:resource="facultyCS" />
   <firstName rdf:datatype="&xsd;string">
     Francesco
   </firstName>
</Person>
```

In general roles are not functional, that is, one individual can be connected to more than one individual (or value)

Agenda

- Motivation
- OWL – General Remarks
- Classes, Roles and Individuals
- **Class Relationships**
- Complex Classes
- Role Characteristics
- OWL Variants
- OWL Ontologies: Reasoning Tasks

Simple Class Relationships: Subclasses

```
<owl:Class rdf:about="Professor">
 <rdfs:subClassOf rdf:resource="FacultyMember" />
</owl:Class>
<owl:Class rdf:about="FacultyMember">
 <rdfs:subClassOf rdf:resource="Person" />
</owl:Class>
```

It logically follows that `Professor` is a subclass of `Person`

Simple Class Relationships: Disjointness

```
<owl:Class rdf:about="Professor">
  <rdfs:subClassOf rdf:resource="FacultyMember" />
</owl:Class>
<owl:Class rdf:about="Book">
  <rdfs:subClassOf rdf:resource="Publication" />
</owl:Class>
<owl:Class rdf:about="FacultyMember">
  <owl:disjointWith rdf:resource="Publication" />
</owl:Class>
```

It logically follows that `Professor` and `Book` are also disjoint classes

Simple Class Relationships: Class Equivalence

```
<owl:Class rdf:about="Man">
  <rdfs:subClassOf rdf:resource="Person" />
</owl:Class>
<owl:Class rdf:about="Person">
  <owl:equivalentClass rdf:resource="Human" />
</owl:Class>
```

It logically follows that `Man` is a subclass of `Human`

Individuals and Class Relationships

```
<Book rdf:about="http://semantic-web-book.org/uri">
   <author rdf:resource="pascalHitzler" />
   <author rdf:resource="markusKroetzsch" />
   <author rdf:resource="sebastianRudolph" />
</Book>
<owl:Class rdf:about="Book">
   <rdfs:subClassOf rdf:resource="Publication" />
</owl:Class>
```

It logically follows that

Foundations of Semantic Web Technologies

is a Publication.

Relationships between Individuals (sameAs)

```
<Professor rdf:about="francescoRicci" />
  <rdf:Description rdf:about="francescoRicci">
   <owl:sameAs rdf:resource="professorRicci" />
  </rdf:Description>
```

It logically follows that `professorRicci` is a `Professor`

Distinctness of individuals is expressed via `owl:differentFrom`.

Relationships between Individuals

```
<owl:AllDifferent>
<owl:distinctMembers rdf:parseType="Collection">
   <Person rdf:about="francescoRicci" />
   <Person rdf:about="diegoCalvanese" />
   <Person rdf:about="wernerNutt" />
</owl:distinctMembers>
</owl:AllDifferent>
```

This is an abbreviated notation instead of using several `owl:differentFrom`

Usage of `owl:AllDifferent` and `owl:distinctMembers` exclusively for this purpose

Closed Classes

```
<owl:Class rdf:about="TechniciansOfCS">
  <owl:oneOf rdf:parseType="Collection">
    <Person rdf:about="amantiaPano" />
    <Person rdf:about="konradHofer" />
  </owl:oneOf>
</owl:Class>
```

tells that there are only *exactly these two* `TechniciansOfCS`

Agenda

- Motivation
- OWL – General Remarks
- Classes, Roles and Individuals
- Class Relationships
- **Complex Classes**
- Role Characteristics
- OWL Variants
- OWL Ontologies: Reasoning Tasks

Logical Class Constructors

- logical `and` (conjunction): `owl:intersectionOf`
- logical `or` (disjunction): `owl:unionOf`
- logical `not` (negation): `owl:complementOf`

. . . used to construct complex classes from simple classesp

Conjunction

```
<owl:Class rdf:about="TechniciansOfCS">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="Technicians" />
    <owl:Class rdf:about="StaffOfCS" />
  </owl:intersectionOf>
</owl:Class>
```

it logically follows that all `TechniciansOfCS` are also `Technicians`

Disjunction

```
<owl:Class rdf:about="Professor">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="ActivelyTeaching" />
        <owl:Class rdf:about="Retired" />
      </owl:unionOf>
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

Negation

```
<owl:Class rdf:about="FacultyMember">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:complementOf rdf:resource="Publication" />
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

Semantically equivalent:

```
<owl:Class rdf:about="FacultyMember">
  <owl:disjointWith rdf:resource="Publication" />
</owl:Class>
```

Role Restrictions (allValuesFrom)

Used to define complex classes via roles

```
<owl:Class rdf:about="Exam">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="hasExaminer" />
      <owl:allValuesFrom rdf:resource="Professor" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

I.e., *all* examiners of an exam have to be professors

Role Restrictions (someValuesFrom)

```
<owl:Class rdf:about="Exam">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="hasExaminer" />
      <owl:someValuesFrom rdf:resource="Person" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

I.e., every exam must have *at least one* examiner

Role Restrictions (Cardinalities)

```
<owl:Class rdf:about="Exam">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="hasExaminer"/>
      <owl:maxCardinality
        rdf:datatype="&xsd;nonNegativeInteger">
        2
      </owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

I.e., an exam may have *at most two* examiners

Role Restrictions (Cardinalities)

```
<owl:Class rdf:about="Exam">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="hasTopic"/>
      <owl:minCardinality
        rdf:datatype="&xsd;nonNegativeInteger">3
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

I.e., an exam must cover *at least three* topics

Role Restrictions (Cardinalities)

```
<owl:Class rdf:about="Exam">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="hasTopic"/>
      <owl:cardinality
        rdf:datatype="&xsd;nonNegativeInteger">3
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

An exam must cover *exactly three* topics

Role Restrictions (hasValue)

```
<owl:Class rdf:about="ExamRicci">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="hasExaminer" />
      <owl:hasValue rdf:resource="francescoRicci" />
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

`owl:hasValue` always refers to one singular individual

The above is equivalent to the example on the next slide

Role Restrictions (hasValue)

```
<owl:Class rdf:about="ExamRicci">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="hasExaminer" />
      <owl:someValuesFrom>
        <owl:oneOf rdf:parseType="Collection">
          <owl:Thing rdf:about="francescoRicci" />
        </owl:oneOf>
      </owl:someValuesFrom>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

Agenda

- Motivation
- OWL – General Remarks
- Classes, Roles and Individuals
- Class Relationships
- Complex Classes
- **Role Characteristics**
- OWL Variants
- OWL Ontologies: Reasoning Tasks

Role Relationships

```
<owl:ObjectProperty rdf:about="hasExaminer">
  <rdfs:subPropertyOf rdf:resource="hasParticipant" />
</owl:ObjectProperty>
```

Likewise: `owl:equivalentProperty`

Roles can be inverses of each other:

```
<owl:ObjectProperty rdf:about="hasExaminer">
  <owl:inverseOf rdf:resource="examinerOf"/>
</owl:ObjectProperty>
```

Role Characteristics

- domain
- range
- transitivity, i.e.
  `r(a, b)` and `r(b, c)` imply `r(a, c)`
- symmetry, i.e.
  `r(a, b)` implies `r(b, a)`
- functionality
  `r(a, b)` and `r(a, c)` imply `b = c`
- inverse functionality
  `r(a, b)` and `r(c, b)` imply `a = c`

Domain and Range

```
<owl:ObjectProperty rdf:about="isMemberOf">
  <rdfs:range rdf:resource="Organization" />
</owl:ObjectProperty>
```

equivalent to:

```
<owl:Class rdf:about="&owl;Thing">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="isMemberOf" />
      <owl:allValuesFrom rdf:resource="Organization" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Domain and Range: Caution!

```
<owl:ObjectProperty rdf:about="isMemberOf">
  <rdfs:range rdf:resource="Organization" />
</owl:ObjectProperty>
<number rdf:about="five">
  <isMemberOf rdf:resource="PrimeNumbers" />
</number>
```

It follows that `PrimeNumbers` is an Organization!

Role Characteristics

```
<owl:ObjectProperty rdf:about="hasColleague">
  <rdf:type rdf:resource="&owl;TransitiveProperty" />
  <rdf:type rdf:resource="&owl;SymmetricProperty" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="hasProjectLeader">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="isProjectLeaderFor">
  <rdf:type rdf:resource="&owl;InverseFunctionalProperty" />
</owl:ObjectProperty>
<Person rdf:about="francescoRicci">
  <hasColleague rdf:resource="diegoCalvanese" />
  <hasColleague rdf:resource="wernerNutt" />
  <isProjectLeaderFor rdf:resource="bzTraffic" />
</Person>
<Project rdf:about="optique">
  <hasProjectLeader rdf:resource="diegoCalvanese" />
  <hasProjectLeader rdf:resource="calvaneseDiego" />
</Project>
```

Consequences from the Example

- `diegoCalvanese hasColleague francescoRicci`
- `diegoCalvanese hasColleague wernerNutt`
- `diegoCalvanese owl:sameAs calvaneseDiego`

Assertional Queries to OWL Ontologies

- Instance checking: does a given individual belong to a given class?
- Search for all individuals that are members of a given class
- Are two given individuals linked by a role?
- Search for all individual pairs that are linked by a certain role
- . . . caution: only “provable” answers will be given!

OWL 1 Language Elements

Head

- `rdfs:comment`
- `rdfs:label`
- `rdfs:seeAlso`
- `rdfs:isDefinedBy`
- `owl:versionInfo`
- `owl:priorVersion`
- `owl:backwardCompatibleWith`
- `owl:incompatibleWith`
- `owl:DeprecatedClass`
- `owl:DeprecatedProperty`
- `owl:imports`

Relationships between individuals

- `owl:sameAs`
- `owl:differentFrom`
- `owl:AllDifferent`
- `owl:distinctMembers`

Pre-defined datatypes (OWL 1)

- `xsd:string`
- `xsd:integer`

OWL Language Elements

Class constructors and relationships

- `owl:Class`
- `owl:Thing`
- `owl:Nothing`
- `rdfs:subClassOf`
- `owl:disjointWith`
- `owl:equivalentClass`
- `owl:intersectionOf`
- `owl:unionOf`
- `owl:complementOf`

Role restrictions

- `owl:allValuesFrom`
- `owl:someValuesFrom`
- `owl:hasValue`
- `owl:cardinality`
- `owl:minCardinality`
- `owl:maxCardinality`
- `owl:oneOf`

OWL Language Elements

Role constructors, relationships and characteristics

- `owl:ObjectProperty`
- `owl:DatatypeProperty`
- `rdfs:subPropertyOf`
- `owl:equivalentProperty`
- `owl:inverseOf`
- `rdfs:domain`
- `rdfs:range`
- `owl:TransitiveProperty`
- `owl:SymmetricProperty`
- `owl:FunctionalProperty`
- `owl:InverseFunctionalProperty`

Further Literature

- `http://www.w3.org/2004/OWL/`
  central W3C web page for OWL
- `http://www.w3.org/TR/owl-features/`
  overview over OWL
- `http://www.w3.org/TR/owl-ref/`
  comprehensive description of the OWL language components
- `http://www.w3.org/TR/owl-guide/`
  introduction into OWL knowledge modeling
- `http://www.w3.org/TR/owl-semantics/`
  describes the semantics of OWL and the abstract syntax for OWL DL
  ($\leadsto$ later lecture)