

RDF Schema (RDFS)

IE650 Knowledge Graphs



Previously on “Knowledge Graphs”

- Is RDF more powerful than XML?
- XML is a markup language for information
- In XML, arbitrary elements and attributes can be defined
- XML tag names are meaningless for a computer
- RDF is a markup language for information
- In RDF, arbitrary classes and predicates can be defined
- RDF class and predicate names are meaningless for a computer

Today: Schemas and Ontologies

- Last week's slides:

- Node types (“classes”) and edge types (“properties”)
 - Are also referred to the “schema” of the graph (aka “ontology”)
 - Can be defined with further restrictions
 - e.g., an edge of type “author” links a publication to a person

- Schemas and ontologies bring semantics to knowledge graphs
- Today:
 - Building simple ontologies with RDF Schema
 - Elements of RDF Schema
 - Automatic deduction with RDF Schema

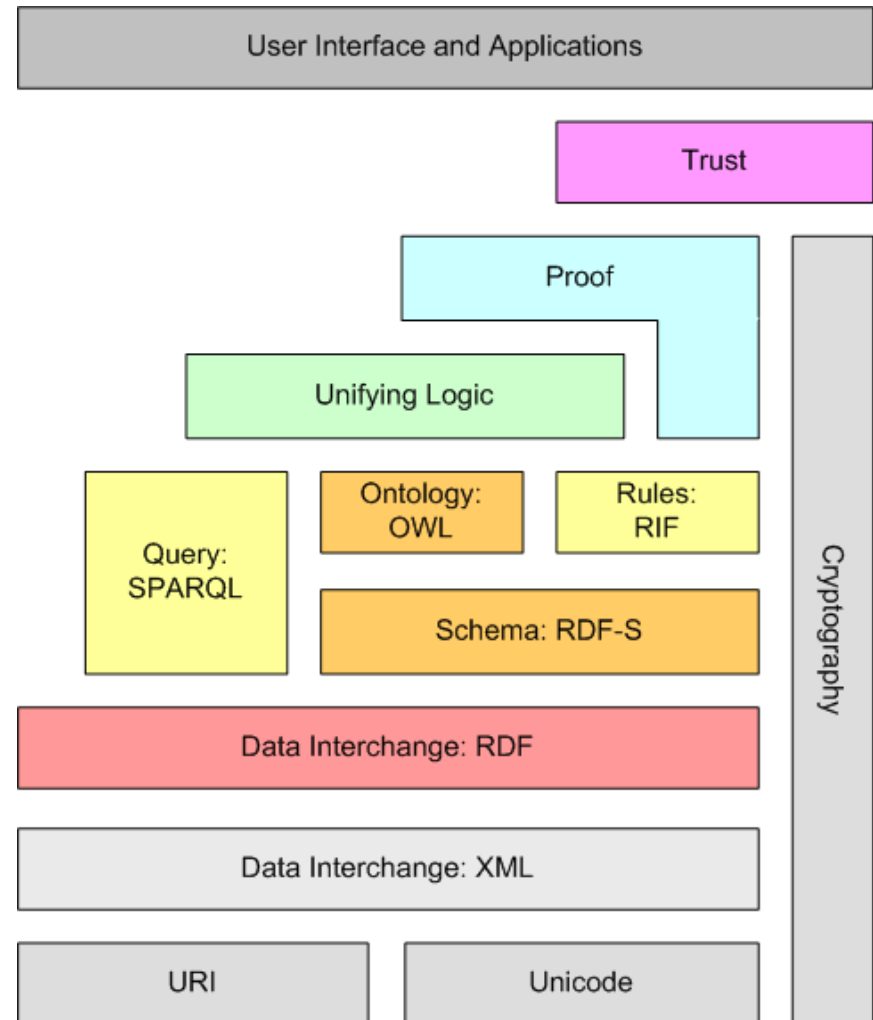
The Semantic Web Stack



here be dragons...

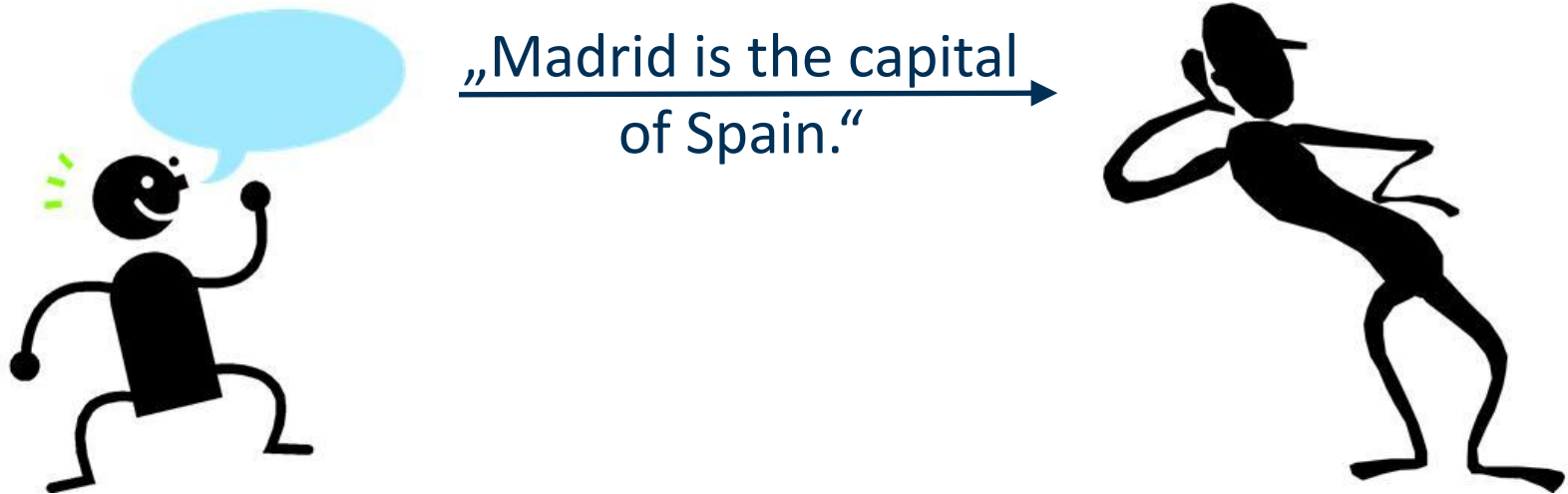
Knowledge Graph Technologies
(This lecture)

Technical
Foundations



What is Missing up to Now?

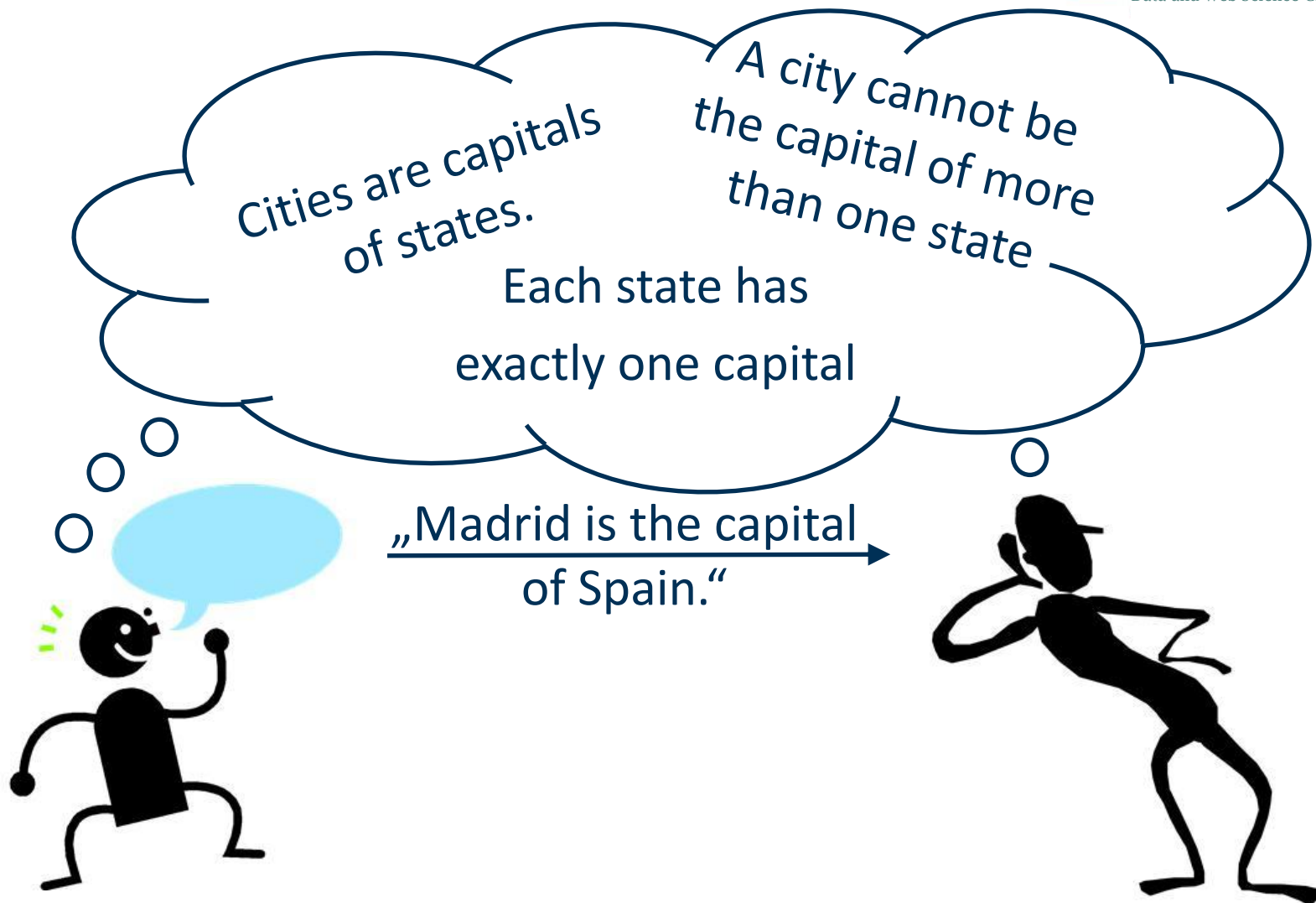
- Basic premise: knowledge graphs should encode information so that humans and computers can understand it
- But what does understand actually mean?



- Let's look at that sentence:
 - "Madrid is the capital of Spain."
- Published in a knowledge graph (i.e., using RDF):
 - `:Madrid :capitalOf :Spain .`
- How many pieces of information can we (i.e., humans) derive from that sentence?
 - (1 piece of information = 1 statement $\langle S, P, O \rangle$)
 - Estimations? Opinions?

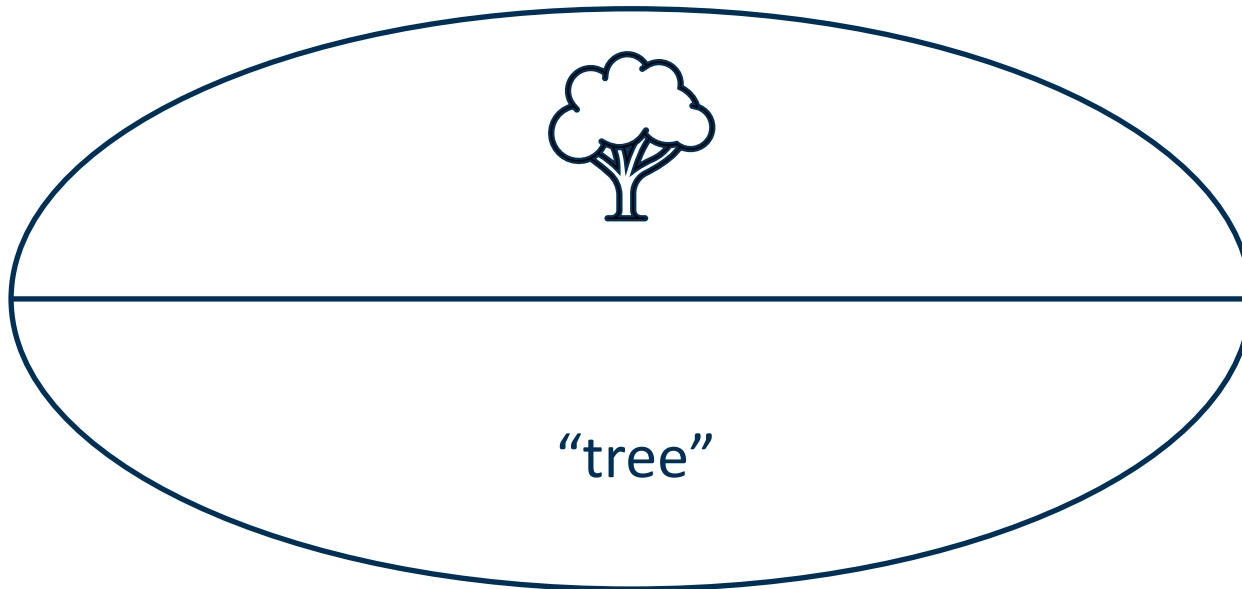
- Let's look at that sentence:
 - "Madrid is the capital of Spain."
- We can get the following information:
 - "Madrid is the capital of Spain."
 - "Spain is a state."
 - "Madrid is a city."
 - "Madrid is located in Spain."
 - "Barcelona is not the capital of Spain."
 - "Madrid is not the capital of France."
 - "Madrid is not a state."
 - ...

How do Semantics Work?



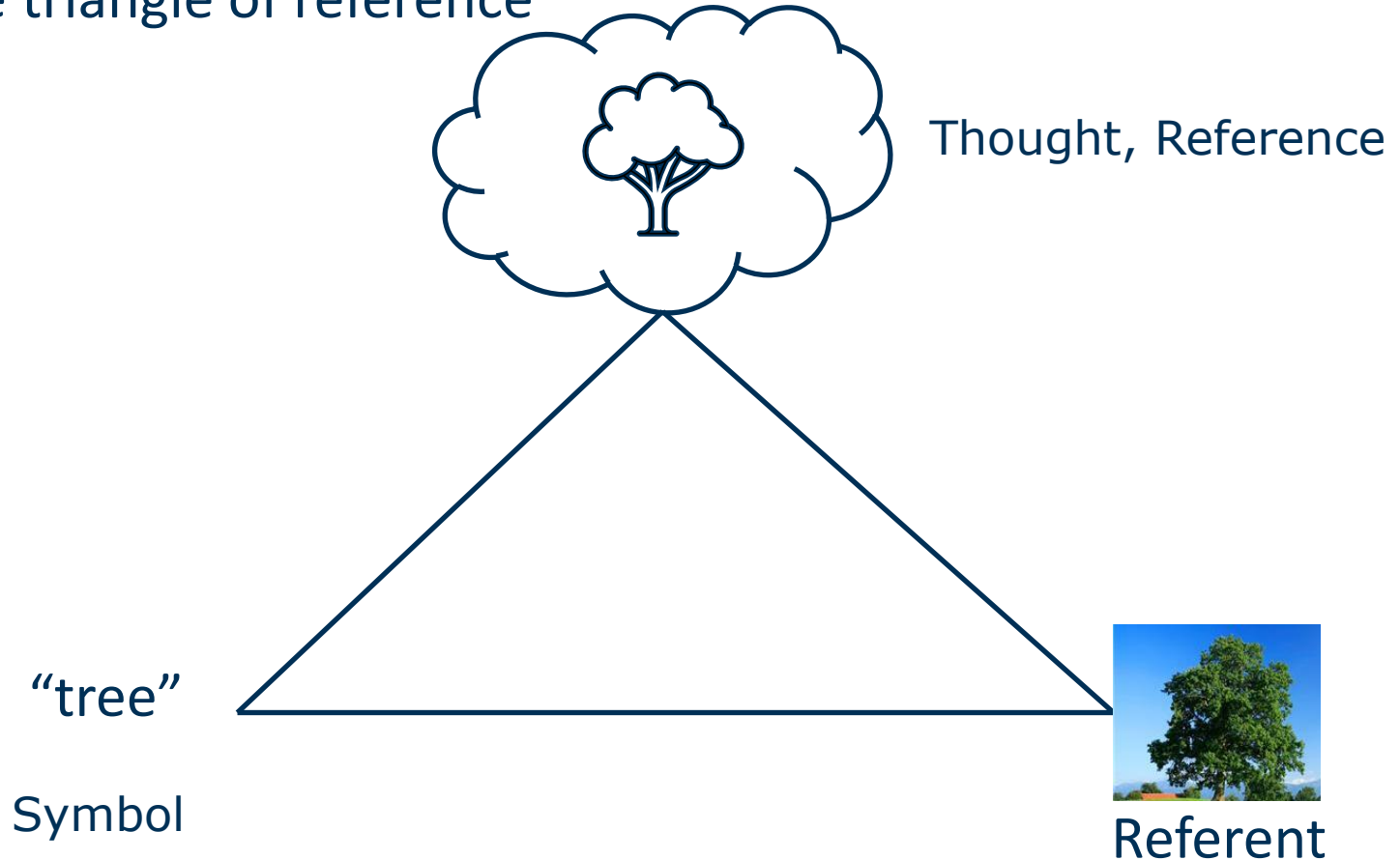
An Excursion to Linguistics

- Saussure's idea of a linguistic sign
- Ferdinand de Saussure (1857-1913):
 - Signifier (signifiant) and signified (signifié) cannot be separated from each other



An Excursion to Linguistics

- The triangle of reference

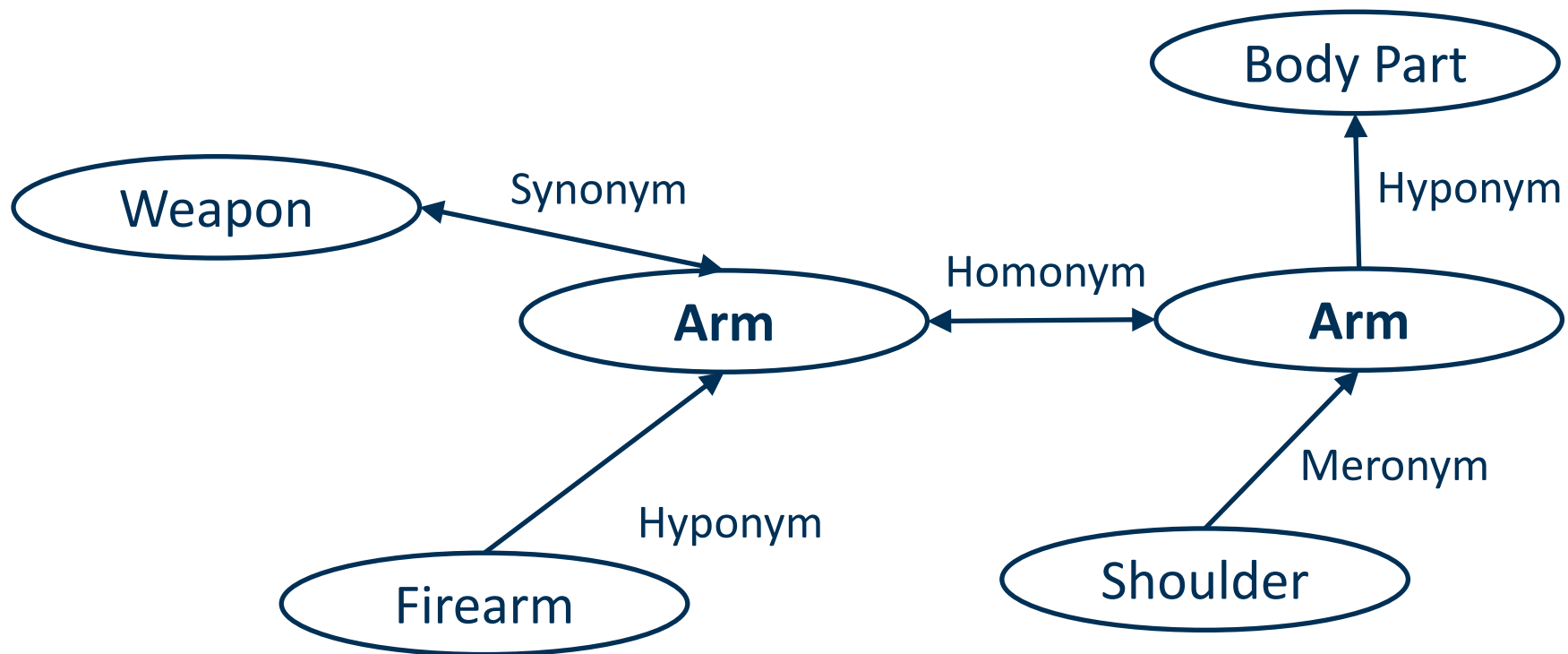


So, how do Semantics Work?

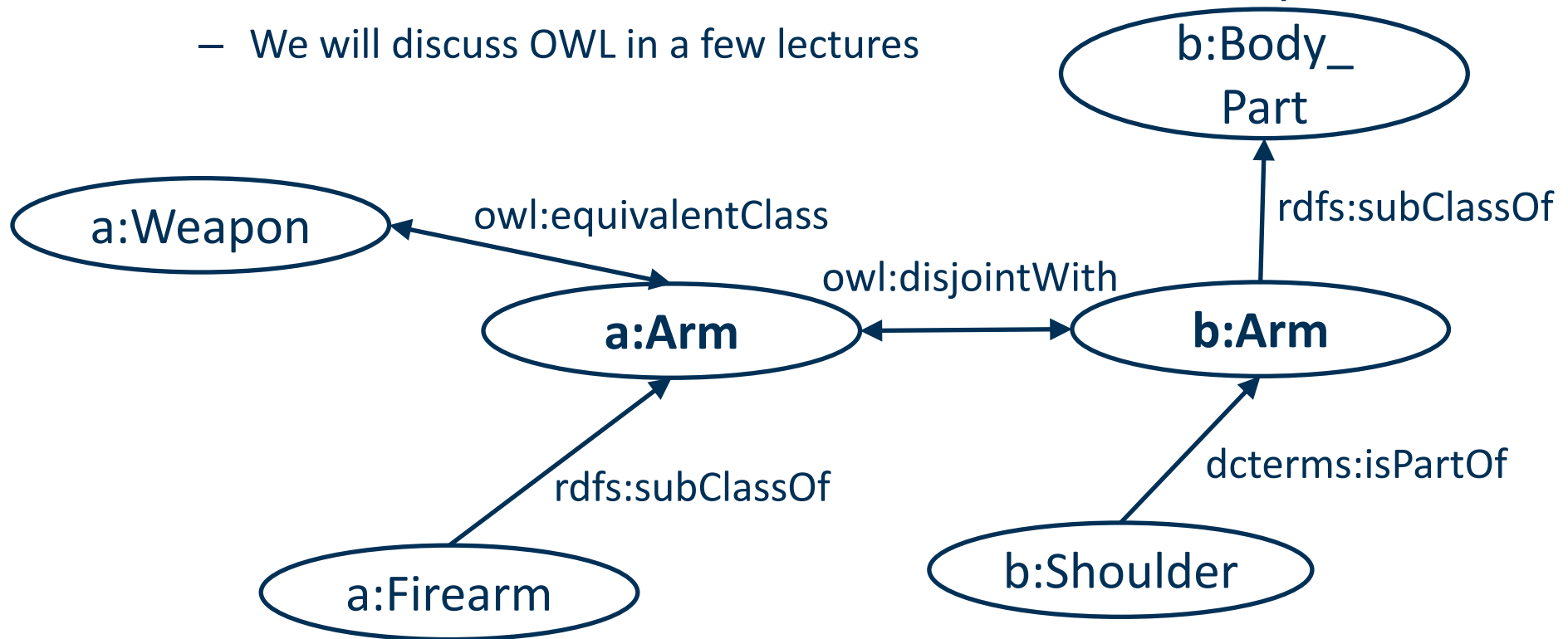
- Lexical semantics
 - Meaning of a word is defined by relations to other words
- Extensional semantics
 - Meaning of a word is defined by the set of its instances
- Intensional semantics, e.g., feature-based semantics
 - Meaning of a word is defined by features of the instances
- Prototype semantics
 - Meaning of a word is defined by proximity to a prototypical instance
- ...
- Each type of semantic is discussed in the following slides

Lexical Semantics

- Defining semantics by establishing relations between words

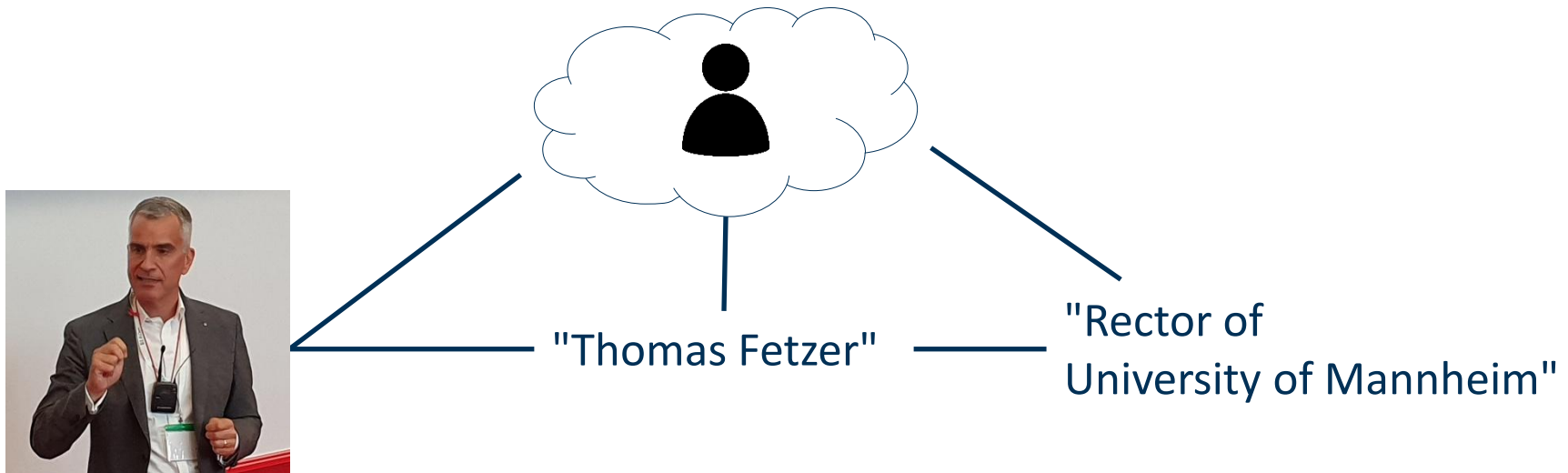


- This can also be translated to RDFS/OWL vocabulary
 - We will discuss OWL in a few lectures



Extensional Semantics

- Listing instances
 - EU members are Austria, Belgium, Bulgaria, ..., Sweden.
- *Thomas Fetzer == Rector of University of Mannheim*
 - both terms have the same extension



Intensional Semantics

- Describes features of things, i.e., *semes*
- A seme is a feature that distinguishes the meaning of two words

Word	has wings	can swim	has fur	can fly
Duck	+	+	-	+
Bird	+	0	-	0
Bee	+	-	-	+
Dolphin	-	+	-	-
...				

Intensional vs. Extensional Semantics

- Intensionally different things can have the same extension
- Classic example: morning star and evening star

Word	Celestial body	bright	visible in the morning	visible in the evening
Morning star	+	+	+	-
Evening star	+	+	-	+
...				

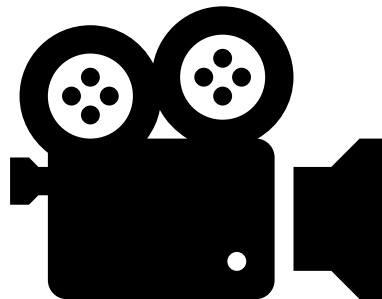
- both have the same extension (i.e., Venus)

Intensional vs. Extensional Semantics

- The extension can change over time without the intension changing
 - e.g., “student”
 - does that change the semantics?
- Intension may also change over time
 - technological achievements (e.g., intension of *ship*)
 - changes in moral values (e.g., intension of *marriage*)
- Extension may also be empty, e.g.
 - Unicorn
 - Martian
 - Yeti (?)

Intensional vs. Extensional Semantics

- ...explained by two well-known experts in the field :-)

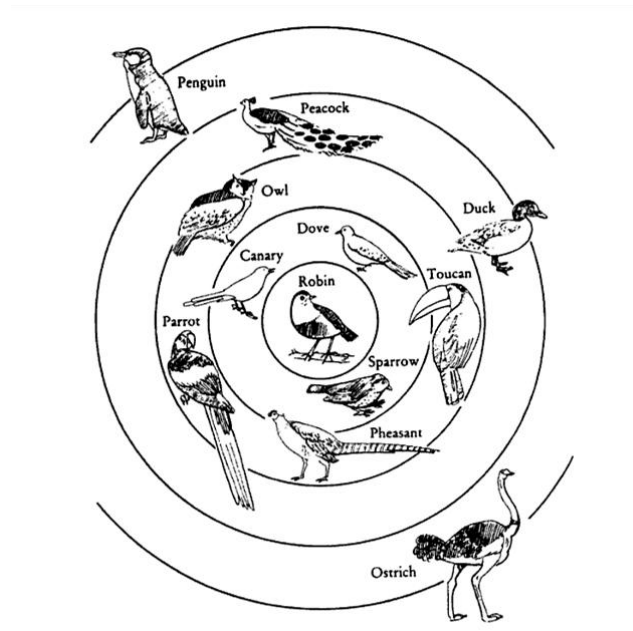


<https://www.youtube.com/watch?v=o8TVAhMQQZg>

<https://www.youtube.com/watch?v=dvTFv5xJUQc>

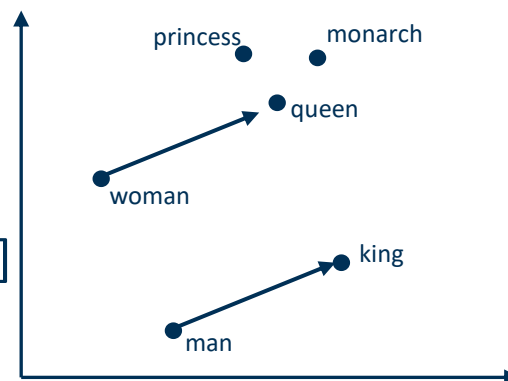
Prototype Semantics

- So far, intensional and extensional semantics are based on boolean logics (i.e., there's only “true” and “false”)
- Prototype Semantics: a more fuzzy variant



Numerical Semantics

- Word embeddings
 - Each word is assigned a static vector
e.g. woman=[0.156, 0.548, 0.982,]
 - Google word2vec google news:
 - 300 dimensions
 - 3 million words
 - Position of the word partially relevant during training (windowing)
- Transformers
 - Each word gets a different embedding vector
dependend of the context it appears
 - Positions are encoded
- Both not part of the lecture but often used for e.g.
matching of concepts



Semantic Shift



“Mask” 2019



“Mask” 2022

How do Semantics Work?

- We have learned:
Semantics define
the meaning of words
- That is what we
do with ontologies
 - Using methods from
lexical, intensional,
and extensional semantics

© Original Artist
Reproduction rights obtainable from
www.CartoonStock.com



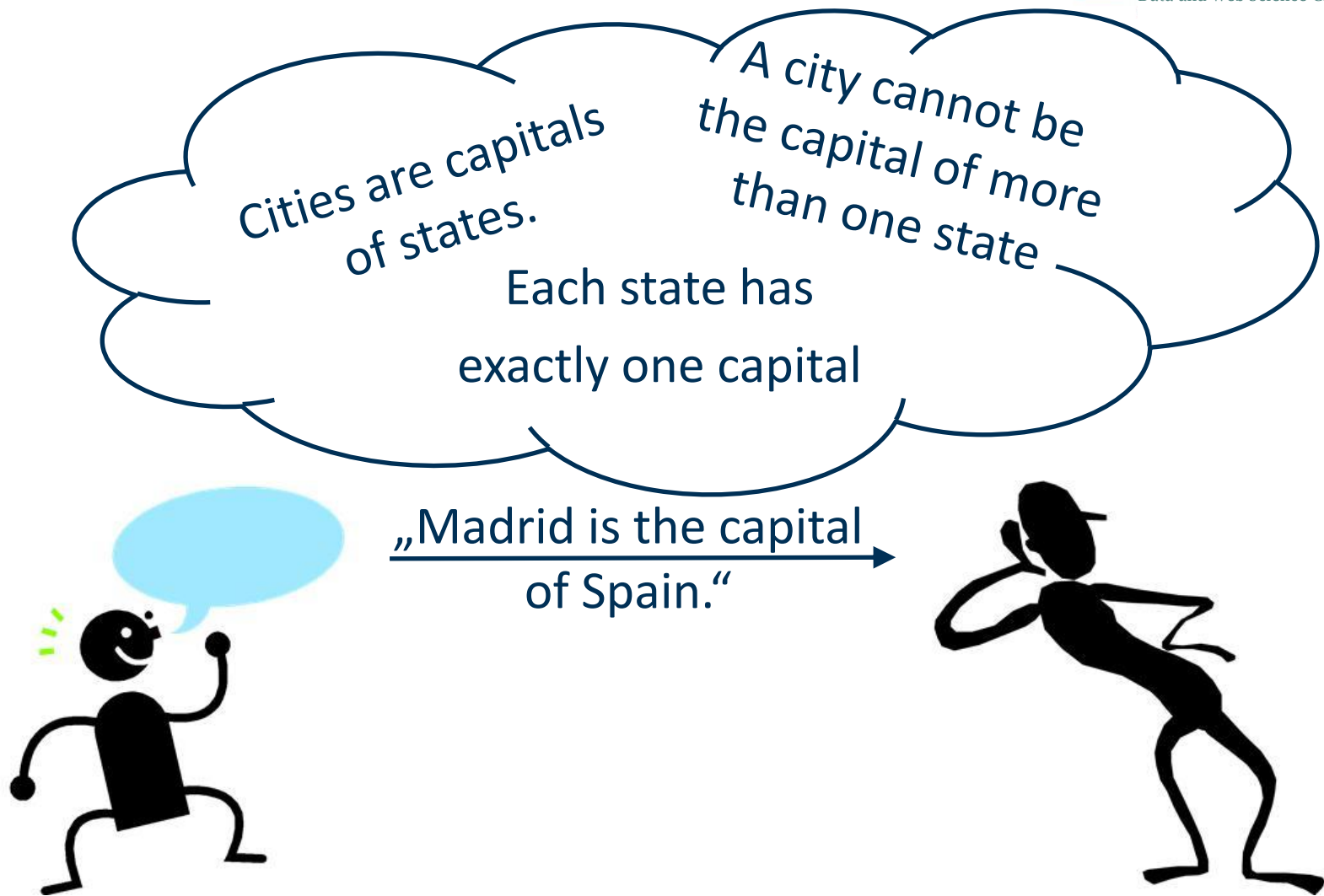
How do Semantics Work?

- We have learned:
Semantics define
the meaning of words
- That is what we
do with ontologies
 - Using methods from
lexical, intensional,
and extensional semantics

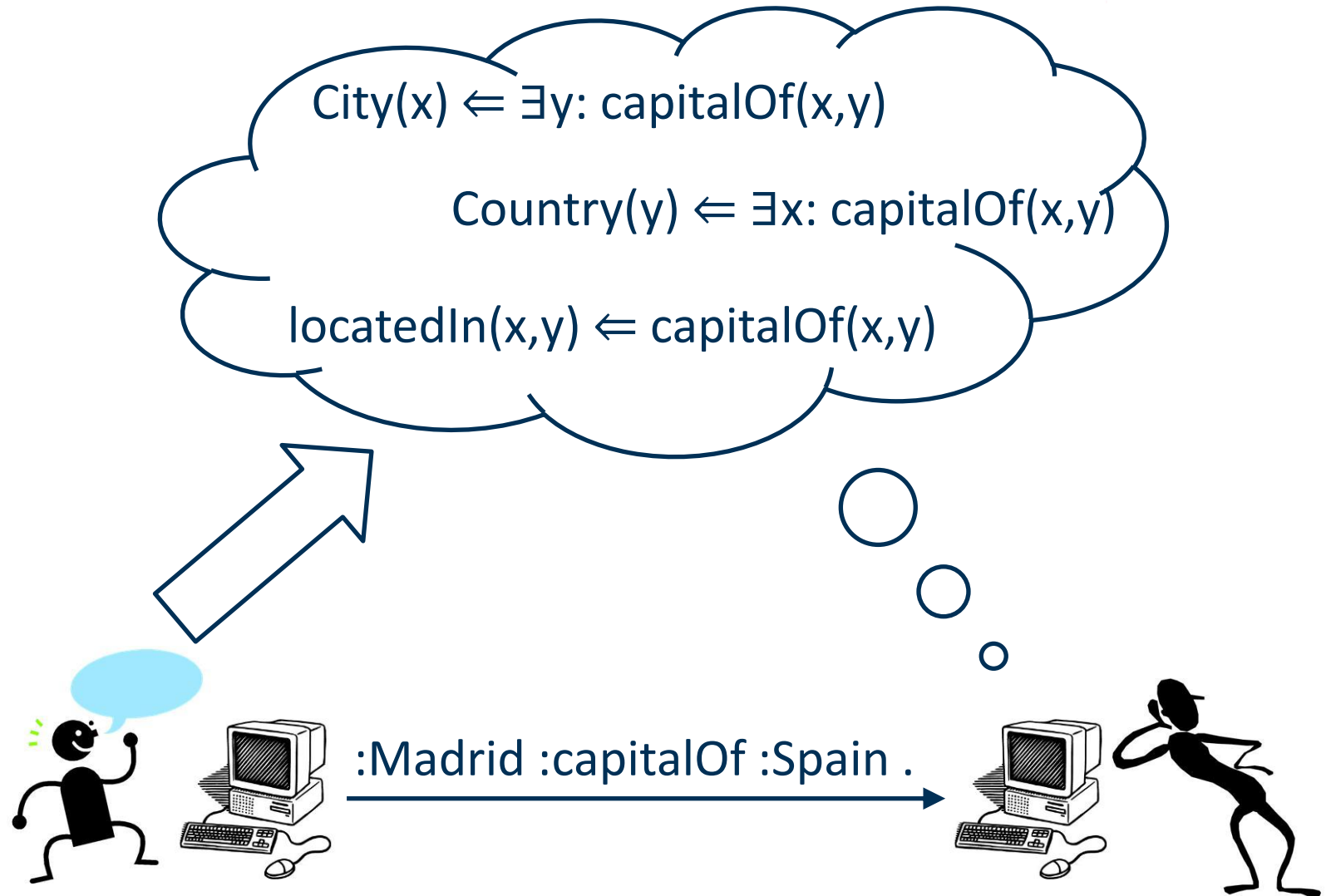


The Treachery of Images (1929) by René Magritte

How do Semantics Work?



Semantics Formalized



$\text{City}(x) \Leftarrow \exists y: \text{capitalOf}(x,y)$

$\text{Country}(y) \Leftarrow \exists x: \text{capitalOf}(x,y)$

$\text{locatedIn}(x,y) \Leftarrow \text{capitalOf}(x,y)$

- "An ontology is an explicit specification of a conceptualization." Gruber (1993): Toward Principles for the Design of Ontologies Used for Knowledge Sharing. In: International Journal Human-Computer Studies Vol. 43, Issues 5-6, pp. 907-928.
- Ontologies encode the knowledge about a domain
- They form a common vocabulary
 - and describe the semantics of its terms

What is an Ontology?

- Ontology (without *a* or *the*) is the philosophical study of being
 - greek: *όντος* (things that are), *λόγος* (the study)
 - A sub discipline of philosophy
- In computer science (with *a* or *the*)
 - A formalized description of a domain
 - A shared vocabulary
 - A logical theory

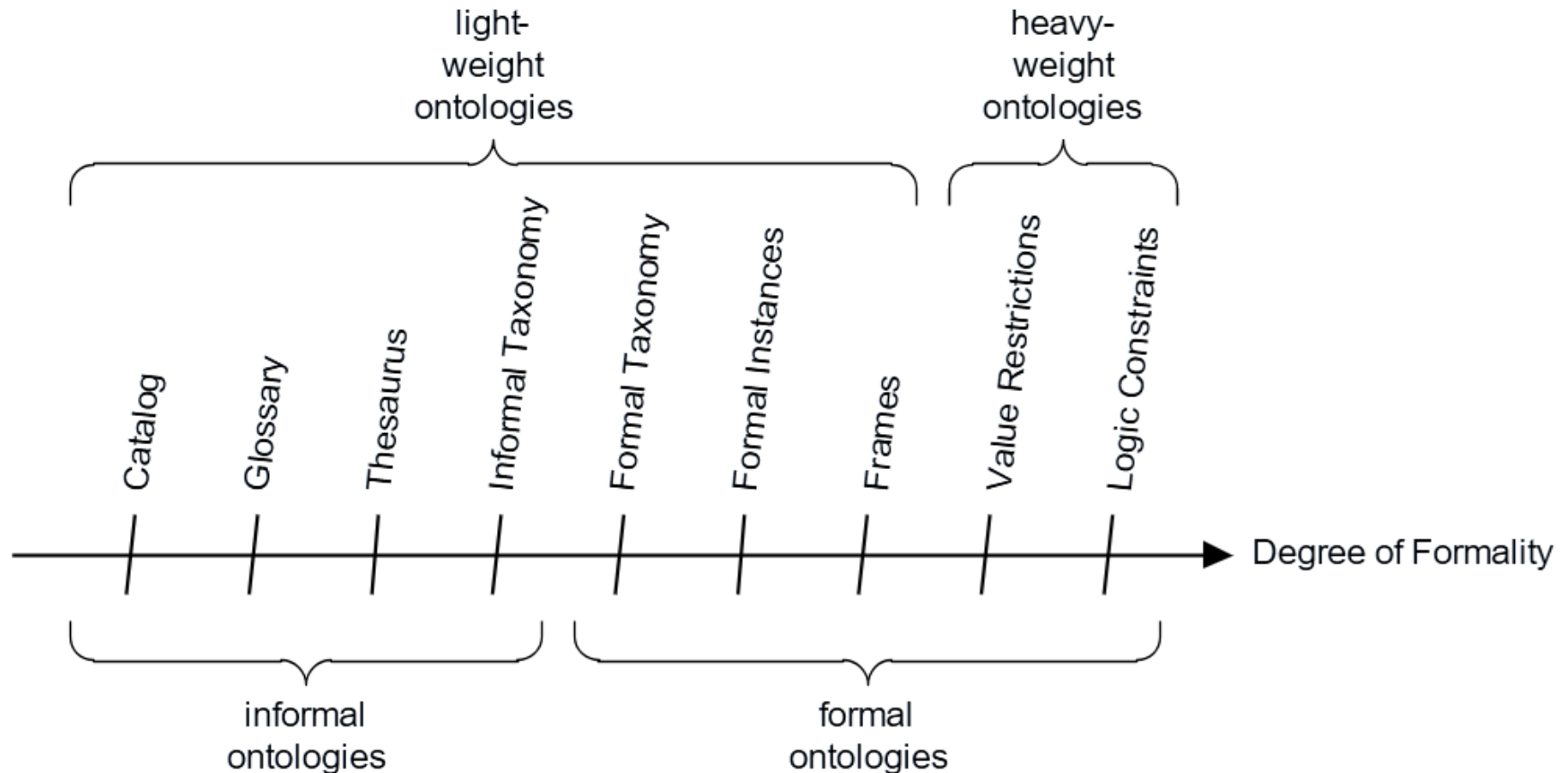
Ontologies – Further Definitions

- Guarino und Giaretta (1995):
"a logical theory which gives an **explicit, partial** account of a conceptualization"
- Uschold und Gruninger (1996):
"**shared** understanding of some domain of interest"
"an **explicit** account or representation of some **part of** a conceptualisation"
- Guarino (1998):
"a set of **logical axioms** designed to account for the intended meaning of a vocabulary"

Essential Properties of Ontologies

- Explicit
 - Meaning is not “hidden” between the lines
- Formal
 - e.g., using logic or rule languages
- Shared
 - An ontology just for one person does not make much sense
- Partial
 - There will (probably) never be a full ontology of everything in the world

Classifications of Ontologies



Lassila & McGuinness (2001): The Role of Frame-Based Representation on the Semantic Web.
In: Linköping Electronic Articles in Computer and Information Science 6(5).

The Oldest Ontology

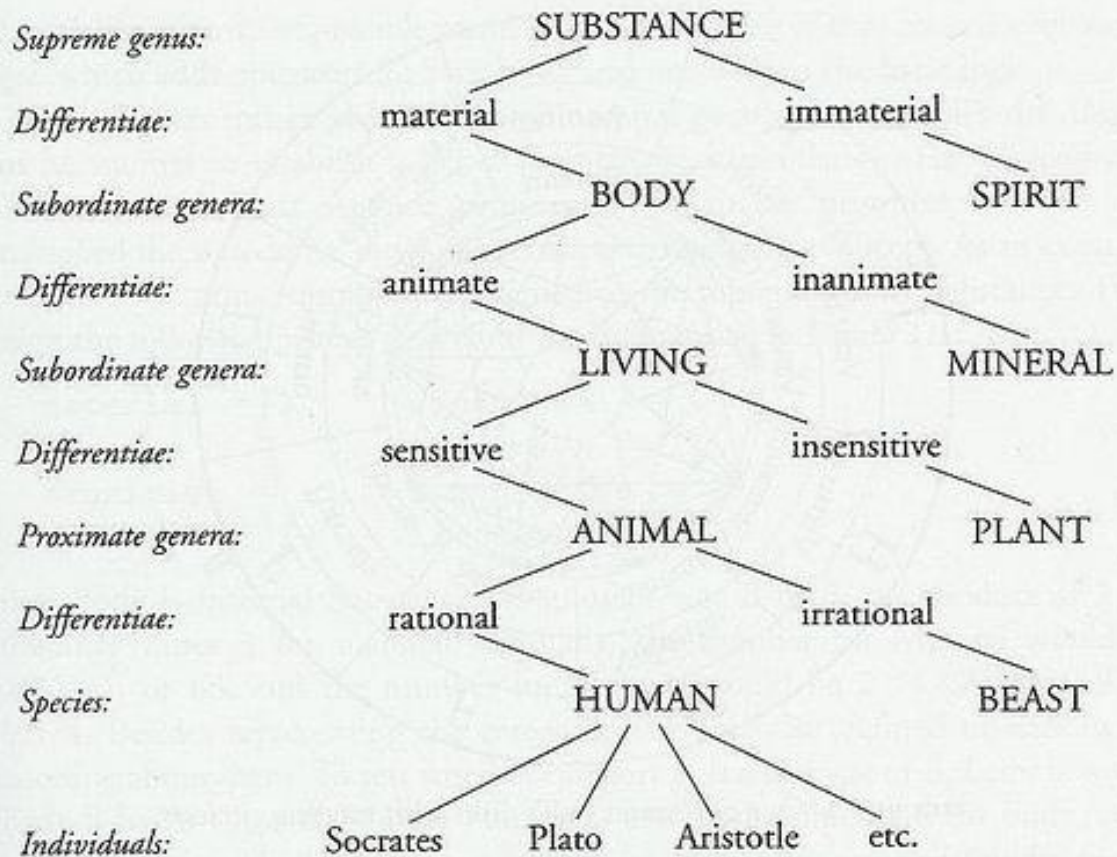


FIGURE 1.1 Tree of Porphyry, translated from a version by Peter of Spain (1239)



Encoding Simple Ontologies: RDFS

- A W3C Standard since 2004
- Most important element: classes



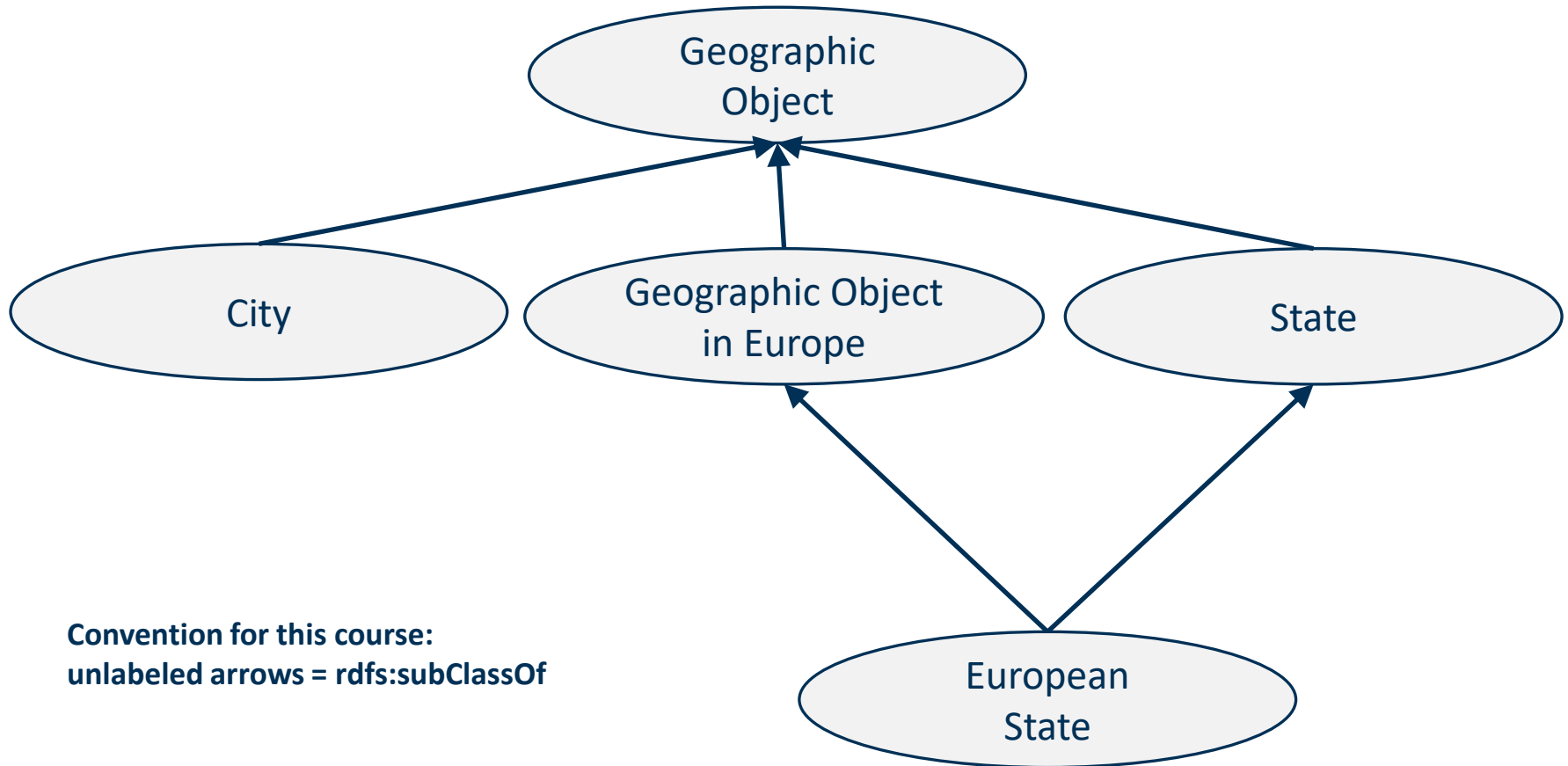
```
:State a rdfs:Class .
```

- Classes form hierarchies

```
:EuropeanState rdfs:subClassOf :State .
```

Class Hierarchies in RDF Schema

- Multiple inheritance is possible



Convention for this course:
unlabeled arrows = `rdfs:subClassOf`

Properties in RDF Schema

- Properties are the other important element
- Resemble two-valued predicates in predicate logic

```
:Madrid :capitalOf :Spain .  
:capitalOf a rdf:Property .
```

- Properties also form hierarchies

```
:capitalOf rdfs:subPropertyOf :locatedIn .
```

Domains and Ranges of Properties

- In general, properties exist independently from classes
 - i.e., they are *first class citizens*
 - this is different than OOP or ERM
- Defining the domain and range of a property:
`:capitalOf rdfs:domain :City .`
`:capitalOf rdfs:range :Country .`
- Domain and range are inherited by sub properties
 - They can also be further restricted

Predefined Properties

- We have already seen
 - `rdf:type`
 - `rdfs:subClassOf`
 - `rdfs:subPropertyOf`
 - `rdfs:domain`
 - `rdfs:range`

Further Predefined Properties

- Labels:

```
:Germany rdfs:label "Deutschland"@de .
```

```
:Germany rdfs:label "Germany"@en .
```

- Comments:

```
:Germany rdfs:comment "Germany as a political  
entity."@en .
```

- Links to other resources:

```
:Germany rdfs:seeAlso <http://www.deutschland.de/> .
```

- Link to defining schema:

```
:Country rdfs:isDefinedBy  
<http://foo.bar/countries.rdfs> .
```

URIs vs. Labels

- A URI is only a unique identifier
 - It does not need to be interpretable
 - `http://www.countries.org/4327893`
- Labels are made for human interpretation
- ...and can come in different languages:
 - `countries:4327893 rdfs:label "Deutschland"@de .`
 - `countries:4327893 rdfs:label "Germany"@en .`
 - `countries:4327893 rdfs:label "Tyskland"@sv .`
 - `...`

URIs vs. Labels

- Labels and comments can also be assigned to RDFS elements:

```
:Country a rdfs:Class.
```

```
:Country rdfs:label "Land"@de.
```

```
:Country rdfs:label "Country"@en.
```

```
:locatedIn a rdf:Property .
```

```
:locatedIn rdfs:label "liegt in"@de.
```

```
:locatedIn rdfs:label "is located in"@en.
```

```
:locatedIn rdfs:comment "refers to geography"@en.
```

RDF Schema and RDF

- Every RDF Schema document is also an RDF document
- This means: all properties of RDF also hold for RDFS!

- Non-unique Naming Assumption

```
schema1:Country a rdfs:Class .
```

```
schema2:State a rdfs:Class .
```

- Open World Assumption

```
:Country rdfs:subClassOf :GeographicObject .
```

```
:City rdfs:subClassOf :GeographicObject .
```

Our First Ontology

- States, cities, and capitals

```
:State a rdfs:Class .
```

```
:City a rdfs:Class .
```

```
:locatedIn a rdf:Property .
```

```
:capitalOf rdfs:subPropertyOf :locatedIn .
```

```
:capitalOf rdfs:domain :City .
```

```
:capitalOf rdfs:range :State .
```

Definition
of the
Terminology
(T-Box)

```
:Madrid :capitalOf :Spain .
```

Definition
of the
Assertions
(A-box)

What do We Gain Now?

:Country a rdfs:Class .
:City a rdfs:Class .
:locatedIn a rdfs:Property .
:capitalOf rdfs:subPropertyOf :locatedIn .
:capitalOf rdfs:domain :City .
:capitalOf rdfs:range :Country .



What do We Gain Now?

```
:Madrid :capitalOf :Spain .  
+ :capitalOf rdfs:domain :City  
→ :Madrid a :City .
```

```
:Madrid :capitalOf :Spain .  
+ :capitalOf rdfs:range:Country  
→ :Spain a :Country .
```

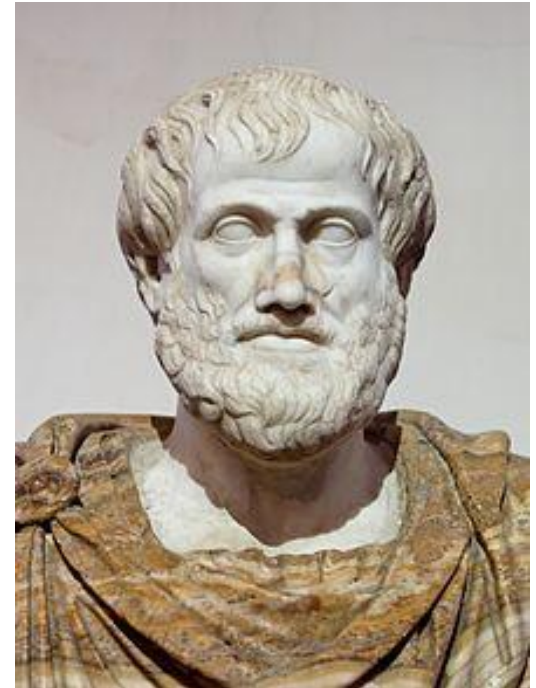
```
:Madrid :capitalOf :Spain .  
+ :capitalOf rdfs:subPropertyOf :locatedIn .  
→ :Madrid :locatedIn :Spain .
```

Reasoning with RDF

- RDF Schema allows for *deductive* reasoning on RDF
- This means:
 - given facts and rules,
 - we can derive new facts
- The corresponding tools are called *reasoner*
- Opposite of deduction: *induction*
 - deriving models from facts
 - see, e.g., lectures on data mining and machine learning

A Bit of History

- Aristotle (384 – 322 BC)
- Syllogisms
 - Deriving facts using rules
- Example:
 - All men are mortal.
 - Socrates is a man.
 - Socrates is mortal.



A Bit of History



Interpretation and Entailment

- Entailment
 - The set of all consequences of a graph
- Mapping a graph to an entailment is called *interpretation*
- Simplest Interpretation:
 - $\langle s, p, o \rangle \in G \rightarrow \langle s, p, o \rangle \in \text{Entailment}$
 - This interpretation creates all statements explicitly contained in the graph.
 - But the *implicit* statements are the interesting ones!

Interpretation using Deduction Rules

- RDF interpretation can be done using RDFS deduction rules
- Those create an entailment
 - using existing resources, literals, and properties
 - creating additional triples like `<s,p,o>`
 - e.g.,
 - `<Madrid, rdf:type, City>`
 - `<Madrid, located_in, Spain>`
- Note:
 - no *new* resources, literals, or properties are created!

Reasoning with Deduction Rules

- Deduction rules are an interpretation function
- Simple reasoning algorithm (a.k.a. *forward chaining*):

```
Given: an RDF Graph G and
       a set of deduction rules R
Entailment E = G
Repeat
    M := { }
    For all rules in R
        For each statement S in E
            Apply R to S
            If E does not contain consequence
                Add consequence to M
    Add all elements in M to E
until M = { }
```

Deduction Rules RDF Schema (Selection)

ID	Condition	Consequence
rdfs2	<code>s p o .</code> <code>p rdfs:domain c .</code>	<code>s rdf:type c .</code>
rdfs3	<code>s p o .</code> <code>p rdfs:range c .</code>	<code>o rdf:type c .</code>
rdfs5	<code>p1 rdfs:subPropertyOf p2 .</code> <code>p2 rdfs:subPropertyOf p3 .</code>	<code>p1 rdfs:subPropertyOf p3 .</code>
rdfs7	<code>p1 rdfs:subPropertyOf p2 .</code> <code>s p1 o .</code>	<code>s p2 o .</code>
rdfs9	<code>s rdf:type c1 .</code> <code>c1 rdfs:subClassOf c2 .</code>	<code>s rdf:type c2 .</code>
rdfs10	<code>c rdf:type rdfs:Class .</code>	<code>c rdfs:subClassOf c .</code>
rdfs11	<code>c1 rdfs:subClassOf c2 .</code> <code>c2 rdfs:subClassOf c3 .</code>	<code>c1 rdfs:subClassOf c3 .</code>

**rdfs:subClassOf is reflexive and transitive
(same for rdfs:subPropertyOf)**

Applying Deduction Rules

- Example:

```
:Employee a rdfs:Class .  
:Employee rdfs:subClassOf :Human .  
:Room a rdfs:Class .  
:worksIn rdfs:subPropertyOf :hasOffice .  
:hasOffice rdfs:domain :Employee .  
:hasOffice rdfs:range :Room .  
  
:Tim :worksIn :B626B01 .
```

Applying Deduction Rules

- Example:

s= :Tim
p1= :worksIn
o= :B626B01
p2= :hasOffice

`:worksIn rdfs:subPropertyOf :hasOffice .`

`:Tim :worksIn :B626B01 .`

ID	Condition	Consequence
rdfs7	<code>p1 rdfs:subPropertyOf p2 .</code> <code>s p1 o .</code>	<code>s p2 o .</code>

→ `:Tim :hasOffice :B626B01 .`

Applying Deduction Rules

- Example:

```
s= :Tim  
p= :hasOffice  
o= :B626B01  
c= :Employee
```

```
:Tim :hasOffice :B626B01 .
```

```
:hasOffice rdfs:domain :Employee .
```

ID	Condition	Consequence
rdfs2	<pre>s p o . p rdfs:domain c .</pre>	<pre>s rdf:type c .</pre>

```
→ :Tim rdf:type :Employee .
```

Applying Deduction Rules

- Example:

s = :Tim
c1 = :Employee
c2 = :Human

:Tim rdf:type :Employee.

:Employee rdfs:subClassOf :Human .

ID	Condition	Consequence
rdfs9	s rdf:type c1 . c1 rdfs:subClassOf c2 .	s rdf:type c2 .

→ :Tim rdf:type :Human .

Forward Chaining

- Example revisited:

```
:Employee a rdfs:Class .  
:Employee rdfs:subClassOf :Human .  
:Room a rdfs:Class .  
:worksIn rdfs:subPropertyOf :hasOffice .  
:hasOffice rdfs:domain :Employee .  
:hasOffice rdfs:range :Room .
```

```
:Tim :worksIn :B626B01 .
```

```
→ :Tim hasOffice :B626B01 .  
→ :Tim rdf:type Employee .  
→ :Tim rdf:type Human .
```



What if there are Multiple Domains/ Ranges?

- Example for social networks:

```
:knows rdfs:domain :Person .  
:knows rdfs:domain :MemberOfSocialNetwork .
```
- What should be the semantics here?
 - Everybody who knows someone
is a person **and** a member of a social network
 - Everybody who knows someone
is a person **or** a member of a social network

The Rules will Tell Us

```
      :knows rdfs:domain :Person.                                (a0)
      :knows rdfs:domain :MemberOfSocialNetwork . (a1)
      :Peter :knows :Stephen .                                    (a2)
(rdfs2+a0+a2) :Peter rdf:type :Person .                          (a3)
(rdfs2+a1+a2) :Peter rdf:type :MemberOfSocialNetwork .          (a4)
      ...
```

- This chain works for each object
 - it is always contained in both classes
 - i.e., the intersection semantics hold

What have We Gained?

- Let's look at that sentence:
 - "Madrid is the capital of Spain."
- We can get the following information:
 - "Madrid is the capital of Spain." ✓
 - "Spain is a state." ✓
 - "Madrid is a city." ✓
 - "Madrid is located in Spain." ✓
 - "Barcelona is not the capital of Spain." ✗
 - "Madrid is not the capital of France." ✗
 - "Madrid is not a state." ✗
 - ...

What we Cannot Express (up to Now)

- "Every state has exactly one capital"
 - Property cardinalities
- "Every city can only be the capital of one state."
 - Functional properties
- "A city cannot be a state at the same time."
 - Class disjointness
- ...
- For those, we need more expressive languages than RDFS!

What we Cannot Express (up to Now)

- "Every state has exactly one capital"
 - i.e., "A state cannot have more than one capital."
- "Every city can only be the capital of one state."
 - i.e., "A city cannot be the capital of two different states."
- "A city cannot be a state at the same time."

What we Cannot Express (up to Now)

- Note: there is no negation in RDF and RDFS
- This means, we cannot produce any contradictions
 - This makes reasoning easy
 - But it also restricts the utility
 - Example:
Mammals do not lay eggs
Penguins lay eggs
→ Penguins are not mammals
- We will get to know formalisms that support negation
 - and learn how to do reasoning with them

What we Cannot Express (up to Now)

- The missing negation perfectly fits the AAA principle
 - Anybody can say anything about anything
- ...and the Open World Assumption
- Any new knowledge will always fit to the knowledge that is already there
 - This principle is called “monotonicity”

What we Cannot Express (up to Now)

- Kurt Gödel (1906-1978)
- Logic systems are either
 - not very powerful or
 - not free of contradictions
- RDF Schema belongs to the first class



What we Cannot Express (up to Now)

- Jim Hendler (*1957)
- "A little semantics goes a long way."



Just a moment

- "We cannot produce any contradictions"
- so what about
 - :Peter a :Baby .
 - :Peter a :Adult .
- That is a contradiction!
- Well, it is – for us human beings
- But a computer will not know
 - Non-unique name assumption!

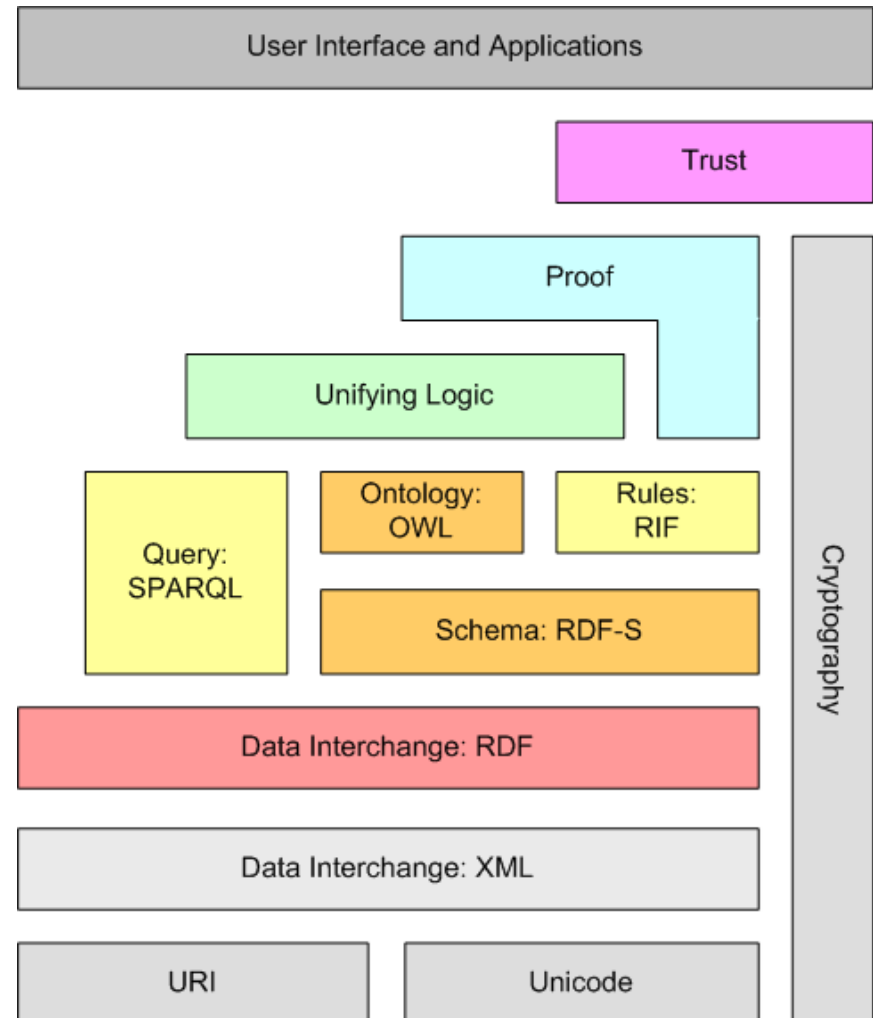
The Semantic Web Stack



here be dragons...

Knowledge Graph Technologies
(This lecture)

Technical
Foundations



Questions?

