

COMP318

Ontologies and Semantic Web

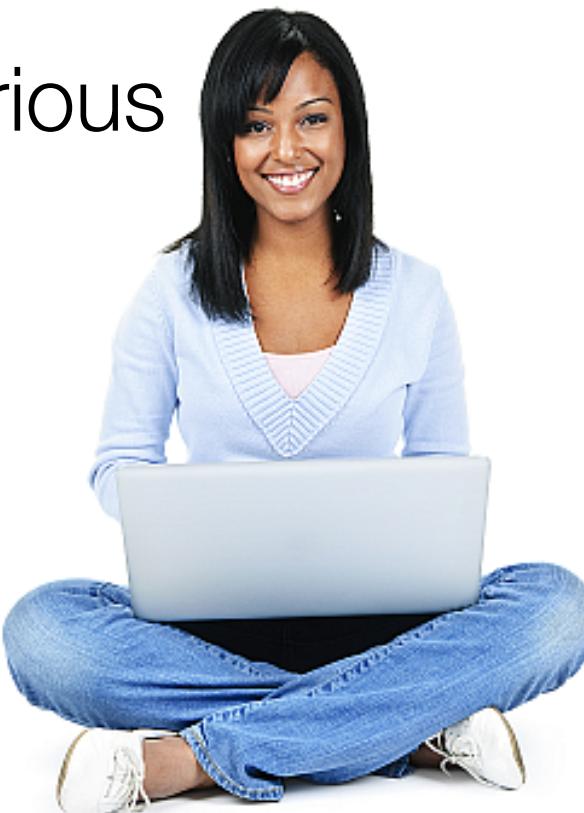
<http://www.csc.liv.ac.uk/~valli/Comp318.html>





Motivation

- The current Web is made of text, pictures, music, movies
 - Great for People!
 - We can make sense of what we see, and select, combine and integrate information from these various sources
 - Useless for Computer Automation
 - Computers can't easily do "intelligent" tasks, which requires understanding this information
- But what if the Web was made of text, pictures, music, movies etc that were also readable by machines?
 - What would be the impact on tasks like search, query answering and knowledge aggregation?



Representing facts

- The success of the WWW is based on the use of standard mechanisms to exchange and communicate data, information and knowledge to different stakeholders:
 - HTML: primarily for human consumption
 - XML: data exchange mechanism
 - ...



```
<html>
<head><title>John Smith</title></head>
<body bgcolor="white">
<b>John Smith</b><br>
is a Lecturer at the
<i>University of Liverpool</i><br>
John teaches <i>Semantic Web
technologies</i><br>
...
</body>
</html>
```

XML: data exchange mechanism

- Exchange languages components

- Syntax: how to write the data
- Data model: how to structure or organise the data
- Semantics: how to interpret (meaning) this data

```
<lecturer name John Smith>
<university> University of Liverpool</university>
    <course>SemWeb Technologies</course>
    <url>http://www.csc.liv.ac.uk/~jsmth</url>
</lecturer>
```

- Semantics clarifies the meaning, specifies assumptions and allows deductions

```
<course name SemWeb Technologies>
<university> University of Liverpool</university>
    <lecturer>John Smith</lecturer>
    <url>http://www.csc.liv.ac.uk/~Comp356</url>
</course>
```

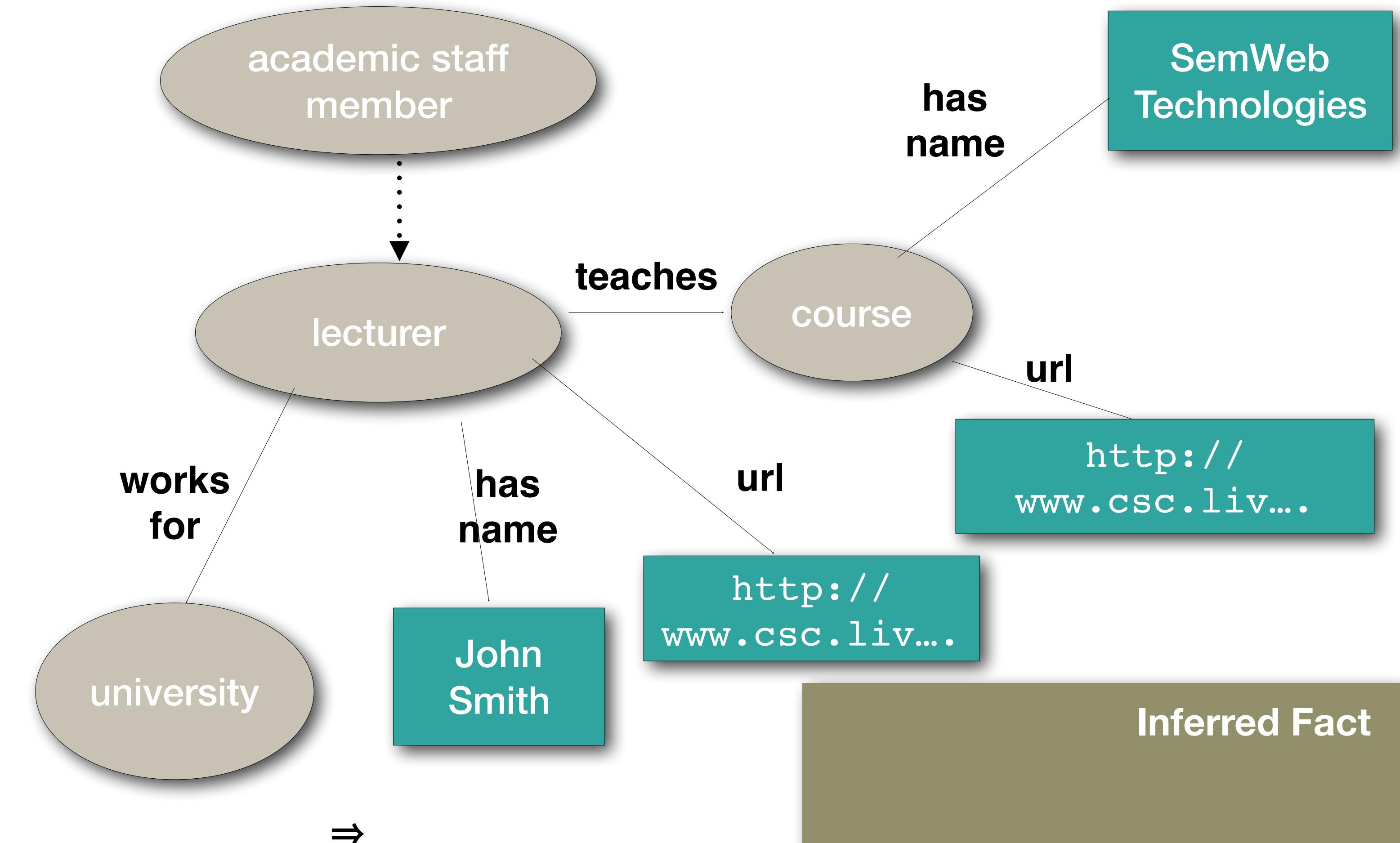
Adding Semantics

```
ex:Lecturer  
rdfs:subClassOf  
ex: AcademicStaff  
ex: hasName  
ex:teaches  
ex:url  
...
```

Schema

```
ex:john_smith  
rdf:type  
ex: Lecturer  
ex: hasName value "John  
Smith"  
ex: url value "http://www..."  
ex: teaches comp_356  
ex: comp_356 hasName  
"SemWeb Technologies"  
...
```

Assertion



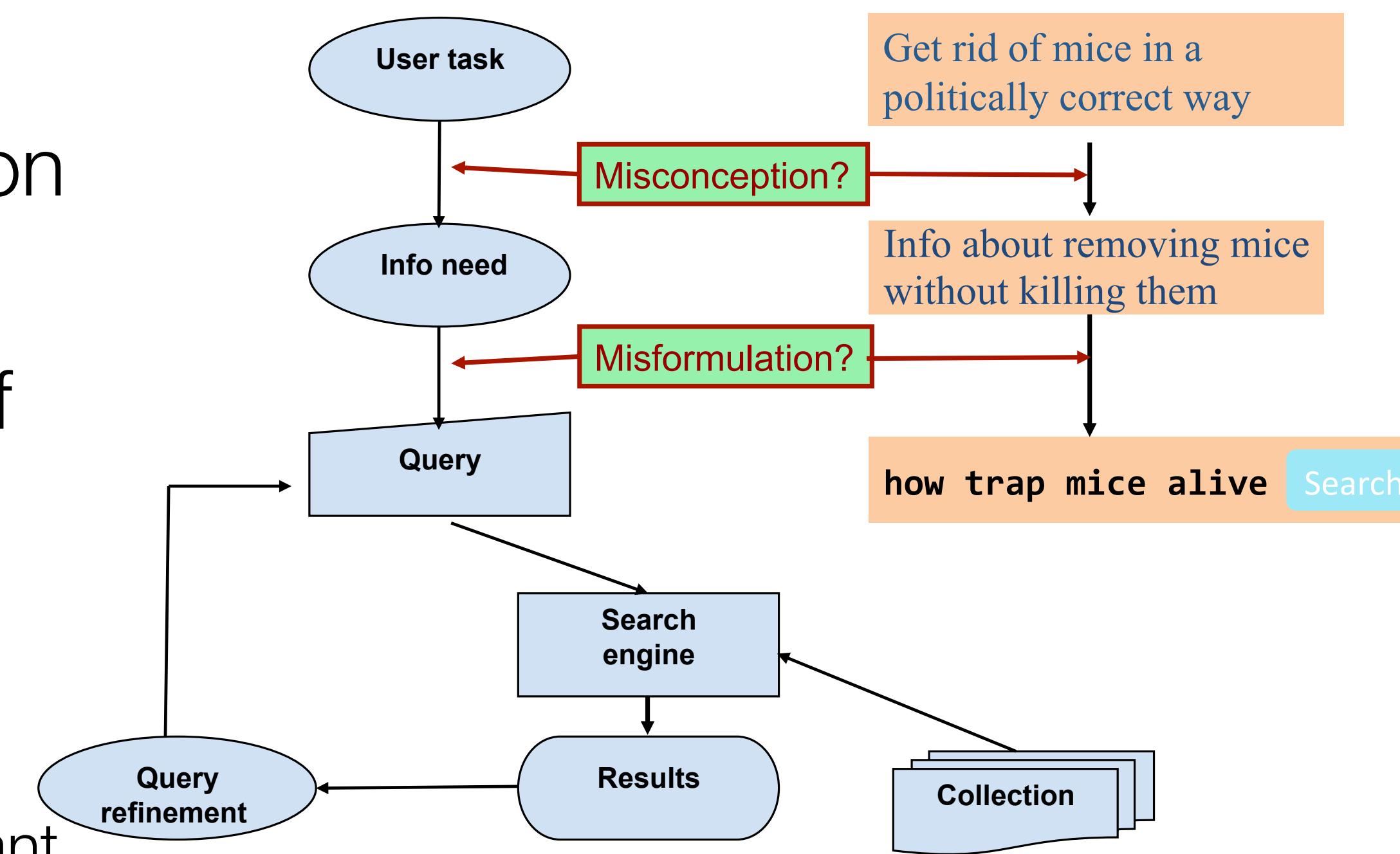
Semantics clarifies the meaning, specifies assumptions and allows deductions.

Search engines allow the web to scale

- The success of the web is due to effective search engines
 - No incentive in creating content unless it can be easily found
 - other methods for locating content haven't kept pace (taxonomies, bookmarks, etc)
- The web is both a technological artefact and a social environment
 - Search engines make aggregation of interests possible:
 - Create incentives for very specialised niche players:
 - Economical: specialised stores, providers, etc
 - Social: narrow interests, specialised communities, etc

Is it all working smoothly?

- The simple fact is that a generic web search returns too many pages!
 - Fine grained interpretation and / or aggregation of results is left to the user
 - Search engines try to contextualise the aim of the query;
 - It is simply difficult to distinguish the meaning between these two sentences:
 - “I am a lecturer of computer science” vs “I am an assistant professor of computer science”
 - Jaguar vs jaguar



The problem's with today's web

Human Centered:

- Information has to be processed by users;
- Direct questions are not understood:
 - Ask: “*How large is the web?*”
 - Answer:

About 8,040,000,000 results (0.68 sec)

Requires active involvement:

- Browsing and searching is keyword based;
- Laborious and often ineffective:
 - Ask: “*How large is the web today?*”
 - Answer:

About 2 results (0.27 sec)

Do you *trust* what you find?

- Where is the information coming from?
- Is it up to date?

The problem's with today's web

Information needs to be integrated manually:

- Users have to read a number of pages / documents;
- Each of these can be relevant / contain part of the answer;
- Users need to search and integrate information that is:
- Disperse;
- Heterogeneous;

Test:
Cost of a camera Canon EOS T3i

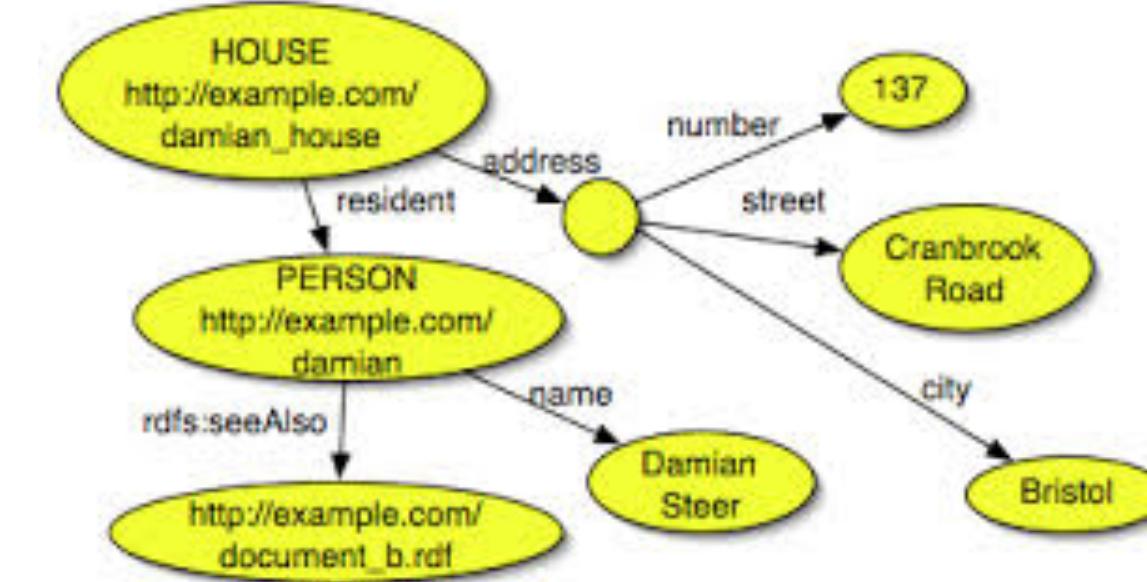
Results:

- Price VAT ex;
- Price VAT inc;
- Shipping included in the price;
- Price in Euros;
- ...

Syllabus

- In this module we:

- Introduce the next generation web (Semantic Web): separation between presentation and content
- Introduce the notion of explicitly representing content through models:
 - different markup languages: RDF and OWL, and query languages: SPARQL
- Study the principles for modelling expressive schemas (Ontologies) to be used to support different applications requiring knowledge sharing
- Look at various applications of these principles and technologies:
 - Knowledge graphs
 - Linked open data
- Assignments use industry tools, i.e. Jena java API and Protege ontology editor



Learning outcomes

- Have an understanding of the basic formal methods and techniques for designing and implementing advanced web applications
- Have an appreciation for Artificial Intelligence and Semantic Web research related to advanced web technology applications
- Be able to apply specific methods and techniques in the design and development of an application of advanced web technology for a case study

Lecturer and demonstrators

Dr Valentina Tamma
Room: Ashton 2.12
Ashton Bldg
v.Tamma@liv.ac.uk

Office hours
Tue 14 – 15
Fri 12 – 13

Joshua Alcock
sgjalcoc@student.liverpool.ac.uk

Reham Alharbi
R.Alharbi@liverpool.ac.uk

Make an appointment by email!

What to study

- Lecture slides:

- Slides are available on VITAL;

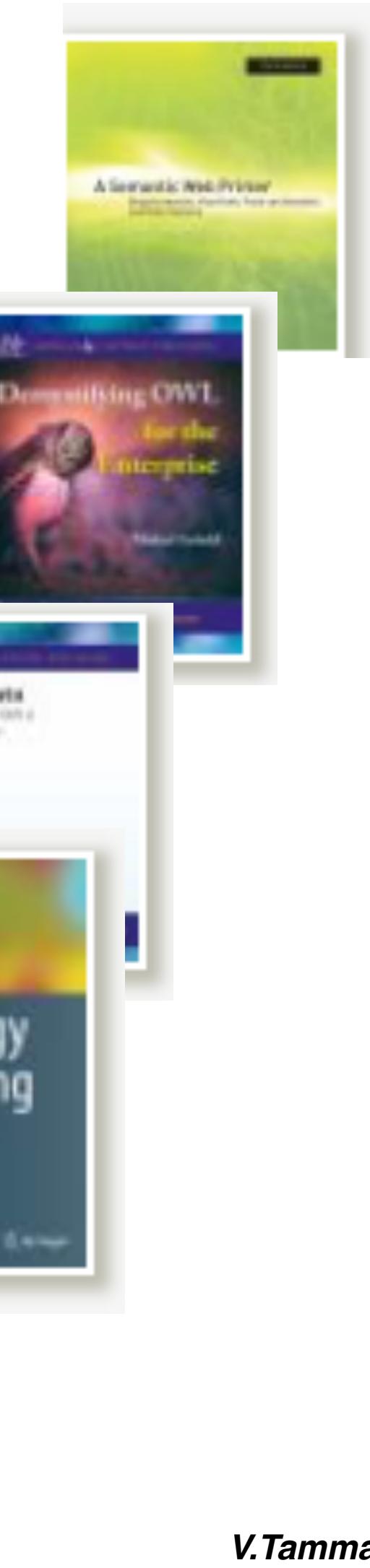
- Text books:

- There isn't a single book, but rather a collection of book chapters. All of these are available from the library:

- Antoniou, Groth, Hoekstra, van Harmelen: A Semantic Web Primer. MIT Press, 2012
 - Uschold: Demistifying OWL for the enterprise. Morgan & Claypool 2018.
 - Heath, Bizer: Linked Data - Evolving the Web into a Global Data Space. Morgan & Claypool 2011.
 - Euzenat, Shvaiko: Ontology matching. Springer, 2012
 - Gomez-Perez, Fernandez-Lopez, Corcho: Ontological Engineering. Springer-Verlag, 2003

- Useful readings:

- Papers of interest and web resources will be indicated during the lectures, and are available on the course web pages. They will allow you to get a better understanding of the topic, and ultimately better marks!



A word on self study

- Part of the course consists in becoming familiar with markup languages: (XML), RDF, OWL
- There is so much one can explain during lectures, but...
 - You need to study their syntax in your own time, and practice in the lab
 - Use the lab sessions to check your progress!
- XML is assumed, but some notes are available for you to refresh your memory...

Web page

- All the material (slides, appendices, exercises, etc) are published on VITAL;
- All announcements are made during lectures and on VITAL;
- First point of reference on the course
- Consult the course page on VITAL!

Assessment

- 80% Exam:
 - Past exam papers available at:
 - <http://www.csc.liv.ac.uk/student/exampapers/>
 - Careful, though, every year I refresh the content to keep into account the latest advances in the area!
- 20% Coursework (1 assignment, 1 class test):
 - You need to work on your own, this course is about learning the fundamental principles!

Organisation of the course

- 30 lectures (3 lectures a week)
 - But extra “surgery” lectures from week 11 if needed
- Lectures
 - Tue 11 - 12 (NICH-LT)
 - Thu 11 - 12 (CHEM-GOS)
 - Fri 11 - 12 (NICH-LT)
- Labs - two groups:
 - Wed 10 - 11 (GHolt H116/117 - Lab2)
 - Fri 09 - 10 (GHolt H105 - Lab 3)
 - Check on Orbit which group you belong to
- First Lab on Wed 05/02/2020

Practicals

- A set of exercises to be completed in the lab in preparation for the assignment and the class test.
- The Programming assignment has to be completed in JAVA
 - Use the industry standard API Apache Jena
 - Assignments marked on the grounds of problem solving ability, rather than mere JAVA skills
 - ***The aim of the assignment is to understand the principles rather than demonstrate programming ability***

Tips for passing!

- Attend the lectures, actively!
 - Ask questions, stop me if I'm not clear;
 - Warning: If some topics are to be read in your own time, that doesn't mean they won't appear in the exam;
 - We'll have class exercises, answers will be given in groups, you have no excuse not to participate!



Semantic Web and Linked (Open) data

- Evolution of the current web towards a web of data:
 - **From information to knowledge based**
 - Semantic web
 - www.semanticweb.org
- Delegate machines to “understand” what is the need behind a query...
 - **Represent knowledge on the Web in a form that is more easily machine-processable:**
 - Improve retrieval;
 - **Use intelligent techniques to take advantage of these representations:**
 - Delegate autonomous software components (agents)

But what is the Semantic Web?

- It NOT just another centralised Knowledge Base framework !!!
 - The Semantic Web is a web of (partially connected) ontologies and knowledge bases
- Facts scattered in many locations...
 - Breaking the “file” paradigm
 - Related instances & ontology fragments can be asserted anywhere
 - Scope & association is managed through namespaces
 - References to ontologies and instances managed through URI
 - Dynamic knowledge source that is typically:
 - Inconsistent / chaotic
 - Incomplete (open-world)
 - Composed of Several Ontologies
 - Generated by novices and machines, as well as experts

A richer data model

- Semantic Web: **beyond machine readable to machine understandable.**

- data model:
 - **data model that can be used by multiple applications**
 - not only for describing documents
 - for people to describe application-specific information
 - **data model that is domain independent**
 - any application can use it to describe information
- semantics:
 - mechanism to interpret the data model
 - describes the interpretations of the data items wrt the domain
- syntax:
 - standardised exchange mechanism

schema.org

Joint effort to define structured data markup schema supported by the major search engines: Google, bing, Yandex, Yahoo!, W3C...

schema.org = ontology + syntax

Hierarchy of types,
each with its own
properties

Microdata or
RDFa in
HTML

Recap

- Problems with today's web
- Introduction to the Semantic Web

COMP318: RDFS entailment

www.csc.liv.ac.uk/~valli/Comp318



Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

V.Tamma@liverpool.ac.uk

based on material by M. Rodriguez Muro, P. Hitzler and S. Rudolph

Where were we

- RDF entailment rules
- RDFS entailment rules

Reasoning engines

- Systems that perform inference are often called reasoning engines or reasoners.
 - **Reasoner engine**: a system that infers new information based on the contents of a knowledgebase. This can be accomplished using rules and a rule engine, triggers on a database or RDF store, decision trees, tableau algorithms, or even programmatically using hard-coded business logic
 - A reasoner must be compliant to the semantics of the ontology language it supports
 - Hence, an ontology language must state its semantics in a formal way
- The RDFS reasoner uses **entailment rules** that are supposed to capture the intended semantics

Soundness and Completeness

- Theorem. A graph G1 RDFS-entails a graph G2 if there is a graph G1' which has been derived from G1 via the rules lg, gl, rdfax, rdf1, rdf2, rdfsax and rdfs1 ... rdfs13 such that:
 - G1' simply entails G2 or
 - G1' contains an XML clash.
- The inference rules for RDFS-entailment we presented previously are sound but not complete (*ter Horst, 2005*).

Example

- The following graph:

```
ex:isHappilyMarriedTo rdfs:subPropertyOf _:bnode.  
_:bnode rdfs:domain ex:Person.  
ex:markus ex:isHappilyMarriedTo ex:anja .
```

- The triple `ex:markus rdf:type ex:Person .` is a semantic consequence of the graph above, but this cannot be derived from the inference rules

Decidability and complexity

- RDFS entailment is decidable, even though one has to deal with the infinite number of axiomatic triples:
 - due to the fact that the RDF vocabulary for encoding lists includes property names `rdf:_i` for all $i \geq 1$, with several RDFS axiomatic triples for each `rdf:_i`
- The problem of deciding whether a graph G_1 RDFS-entails another graph G_2 is NP-complete. The problem becomes polynomial if G_2 contains no blank nodes

RDFS entailment in state of the art systems

- Existing RDF stores (Jena, Sesame, Virtuoso, Oracle, etc) offer implementations of RDFS entailment together with ways of querying the stored graphs through SPARQL
- Implementations may be based on applying the rules in a backward chaining or a forward chaining fashion

RDFS entailment cheatsheet

RDFS entailment patterns.

	If S contains:	then S RDFS entails recognizing D:
rdfs1	any IRI aaa in D	aaa <code>rdf:type rdfs:Datatype .</code>
rdfs2	aaa <code>rdfs:domain XXX .</code> yyy aaa zzz .	yyy <code>rdf:type XXX .</code>
rdfs3	aaa <code>rdfs:range XXX .</code> yyy aaa zzz .	zzz <code>rdf:type XXX .</code>
rdfs4a	xxx aaa yyy .	XXX <code>rdf:type rdfs:Resource .</code>
rdfs4b	xxx aaa yyy .	yyy <code>rdf:type rdfs:Resource .</code>
rdfs5	XXX <code>rdfs:subPropertyOf yyy .</code> yyy <code>rdfs:subPropertyOf zzz .</code>	XXX <code>rdfs:subPropertyOf zzz .</code>
rdfs6	XXX <code>rdf:type rdf:Property .</code>	XXX <code>rdfs:subPropertyOf XXX .</code>
rdfs7	aaa <code>rdfs:subPropertyOf bbb .</code> xxx aaa yyy .	xxx bbb yyy .
rdfs8	XXX <code>rdf:type rdfs:Class .</code>	XXX <code>rdfs:subClassOf rdfs:Resource .</code>
rdfs9	XXX <code>rdfs:subClassOf yyy .</code> zzz <code>rdf:type XXX .</code>	zzz <code>rdf:type yyy .</code>
rdfs10	XXX <code>rdf:type rdfs:Class .</code>	XXX <code>rdfs:subClassOf XXX .</code>
rdfs11	XXX <code>rdfs:subClassOf yyy .</code> yyy <code>rdfs:subClassOf zzz .</code>	XXX <code>rdfs:subClassOf zzz .</code>
rdfs12	XXX <code>rdf:type rdfs:ContainerMembershipProperty .</code>	XXX <code>rdfs:subPropertyOf rdfs:member .</code>
rdfs13	XXX <code>rdf:type rdfs:Datatype .</code>	XXX <code>rdfs:subClassOf rdfs:Literal .</code>

Example

Given the RDF graph S:

:e rdfs:subClassOf :d .

:c rdf:type owl:Class .

:d rdfs:domain _:y .

:a rdfs:comment "string" .

:g :d :f.

Is the following graph **RDFS-entailed** by S? Explain the answer

:d rdf:type rdfs:Resource .

Example

Given the RDF graph S:

:e rdfs:subClassOf :d .

:c rdf:type owl:Class .

:d rdfs:domain _:y .

:a rdfs:comment "string" .

:g :d :f.

Is the following graph **RDFS-entailed** by S? Explain the answer

:d rdf:type rdfs:Resource .

Yes, this triple is entailed because it can be inferred from the axiomatic triples (remember in RDFS everything is a resource).

Exercise

Given the RDF graph S:

```
rdfs:range rdfs:range rdfs:Class .  
  
:s rdfs:domain :t .  
  
:u rdfs:subPropertyOf :s .  
  
:a :s :b .  
  
:a rdf:type :u .  
  
:u rdfs:subClassOf :y .  
  
:t rdfs:subClassOf :s .  
  
:t rdfs:comment "'bla'" .
```

Is the following graph RDFS-entailed by S? Explain the answer

```
:a rdf:type :t .
```

Exercise

rdfs:range rdfs:range rdfs:Class .

:s rdfs:domain :t .

Is the following graph RDFS-entailed by S?

Explain the answer

:u rdfs:subPropertyOf :s .

:a rdf:type :t .

:a :s :b .

Yes, because the following triples hold

:a rdf:type :u .

:a :s :b .

:u rdfs:subClassOf :y .

:s rdfs:domain :t and because of
entailment rule rdfs2 (cheat sheet)

:t rdfs:subClassOf :s .

:t rdfs:comment "bla" .

RDFS entailment

- Given the graph G below,

```
<d:Poe, o:wrote, d:TheGoldBug .>  
<d:TheGoldBug, rdf:type, o:Novel .>  
<d:Baudelaire, o:translated, d:TheGoldBug .>
```

```
<d:Poe, o:wrote, d:TheRaven .>  
<d:TheRaven, rdf:type, o:Poem .>  
<d:Mallarme', o:translated, d:TheRaven .>
```

```
<d:Mallarme', o:wrote, _:b .>  
  <_:b, rdf:type, o:Poem .>  
  
<o:Poem rdfs:subClassOf ex:Literature .>  
<o:Novel rdfs:subClassOf ex:Literature .>
```

- And the following graph S, determine if G entails (using simple and RDFS entailment) S, and explain why.

S= <d:Poe wrote _:c .> <_:c rdf:type ex:Literature .>

RDFS entailment

1. <**d:Poe**, o:wrote, d:TheGoldBug .>
2. <**d:TheGoldBug**, rdf:type, o:Novel .>
3. <**d:Baudelaire**, o:translated,
d:TheGoldBug .>
4. <**d:Poe**, o:wrote, d:TheRaven .>
5. <**d:TheRaven**, rdf:type, o:Poem .>
6. <**d:Mallarme`**, o:translated, d:TheRaven .>
7. <**d:Mallarme`**, o:wrote, _:b .>
 <_:b, rdf:type, o:Poem .>
8. <o:Poem rdfs:subClassOf ex:Literature .>
9. <o:Novel rdfs:subClassOf
ex:Literature .>

From RDFS 9 applied to 9 and 2

10. <**d:TheGoldBug**, rdf:type,
o:Literature .>

Apply SE1 to 1 and we obtain
<**d:Poe** o:wrote _:c>
with _:c -> d:TheGoldBug

Apply SE2 to 10, and we obtain
11. <_:c, rdf:type, o:Literature .>

So, the graph S is RDFS entailed

S= <**d:Poe** wrote _:c .>
<_:c rdf:type ex:Literature .>

Brief recap of RDF/RDFS

- RDF/RDFS describe subject predicate object triples and basic subsumption relations.
 - Very efficient deduction/querying
 - SPARQL based on simple entailment, but...
 - Very poor expressivity...
 - Describe data in terms of triples (RDF)
 - Describe the model behind the data (RDFS) by:
 - Declare the “types” and properties/relationships of the things we want to make assertions about
 - Allowing to infer new assertions implicitly stated from a set of given facts
- But sometimes we need to express more advanced notions, e. g.:
 - A person has only one birth date
 - No person can be male and female at the same time

What can you represent with RDFS

- **RDFS provides:**
 - Classes
 - Lecturer rdf:type rdfs:Class
 - Class hierarchies
 - Lecturer rdfs:subClassOf AcademicStaff
 - Properties
 - teachesModule rdf:type rdf:Property
 - Property hierarchies
 - teachesModule rdfs:subPropertyOf coordinatesModule
 - Domain and range declarations
 - teachesModule rdfs:domain Lecturer
 - teachesModule rdfs:range Module
 - They infer information rather than checking data

What can't you represent in RDFS

- **RDFS does NOT provide:**
 - Disjointness of Classes
 - *Male and Female are disjoint*
 - *they cannot have any shared instances*
 - Property characteristics (inverse, transitive, ...)
 - *Lecturer teachesModule Module and Module isTaughtByLecturer Lecturer are not explicitly related*
 - Local scope of properties
 - *Lecturer has a property hasTitle whose values (range) is restricted to all values of the class PhD*
 - *only people who hold a PhD can be lecturers*
 - but other AcademicStaffMembers can hold a BSc
 - Complex concept definitions (Boolean combination of classes)
 - *Person = Man ∪ Woman*
 - *Mother = Woman ∩ Parent*

What can't you represent in RDFS

- **RDFS does NOT provide:**
 - Cardinality restrictions
 - Person may have at most 1 name
 - Lecturer *teaches exactly two modules*
 - A way to distinguish between classes and instances:
 - Feline rdf:type rdfs:Class
 - Cat rdf:type Feline
 - felix rdf:type Cat
 - :felix is an instance of an instance (Cat)
 - A way to distinguish between language constructors and ontology vocabulary:
 - rdf:type rdfs:range rdfs:Class
 - rdfs:Property rdfs:type rdfs:Class
 - rdf:type rdfs:subPropertyOf rdfs:subClassOf
 - Reasoning for these non-standard semantics

What can you infer in RDFS

Schema

```
ex:Lecturer  
rdfs:subClassOf  
ex: AcademicStaff
```

RDFS Entailment ⇒

Inferred Fact

```
staffUniv:john_smith  
rdf:type  
ex: AcademicStaff
```

Assertion

```
staffUniv:john_smith  
rdf:type  
ex: Lecturer
```

What can't you infer in RDFS

- However, not all types of inferences are possible in RDFS

Schema

```
:wife_of rdfs:subPropertyOf married_to.  
:married_to rdfs:domain :Spouse;  
           rdfs:range  :Spouse.  
:wife_of rdfs:domain :Wife;  
           rdfs:range  :Husband.
```

Assertion

```
:juliet :wife_of :romeo.
```

Facts Inferred

```
:juliet rdf:type    :Wife;  
        rdf:type    :Spouse;  
        married_to :romeo;  
:romeo  rdf:type   :Spouse;  
        rdf:type   :Husband.
```

What can't you infer in RDFS

- What about if we want to model symmetry,
i.e. `:x :married_to :y`

~~implies `--> :married_to -->`?~~

Schema

```
:wife_of rdfs:subPropertyOf married_to.  
:married_to rdfs:domain :Spouse;  
           rdfs:range  :Spouse.  
:wife_of rdfs:domain :Wife;  
           rdfs:range  :Husband.  
:husband_of rdfs:domain :Husband;  
           rdfs:range  :Wife.
```

Assertion

```
:juliet :wife_of :romeo.
```

Facts Inferred

```
:juliet rdf:type  :Wife;  
        rdf:type  :Spouse;  
        married_to :romeo;  
:romeo rdf:type  :Spouse;  
        rdf:type  :Husband.
```

Facts *NOT Inferred*

```
:romeo :married_to :juliet.  
:romeo :husband_of :juliet.
```

Too much representational freedom is not good!

- We might want to be able to detect what might seem inconsistent facts, but RDFS is not able to constrain models through consistency and axioms:

Schema

```
:wife_of rdfs:subPropertyOf married_to.  
:married_to rdfs:domain :Spouse;  
           rdfs:range  :Spouse.  
:wife_of rdfs:domain :Wife;  
           rdfs:range  :Husband.  
:husband_of rdfs:domain :Husband;  
           rdfs:range  :Wife.
```

Facts *INCORRECTLY Inferred*

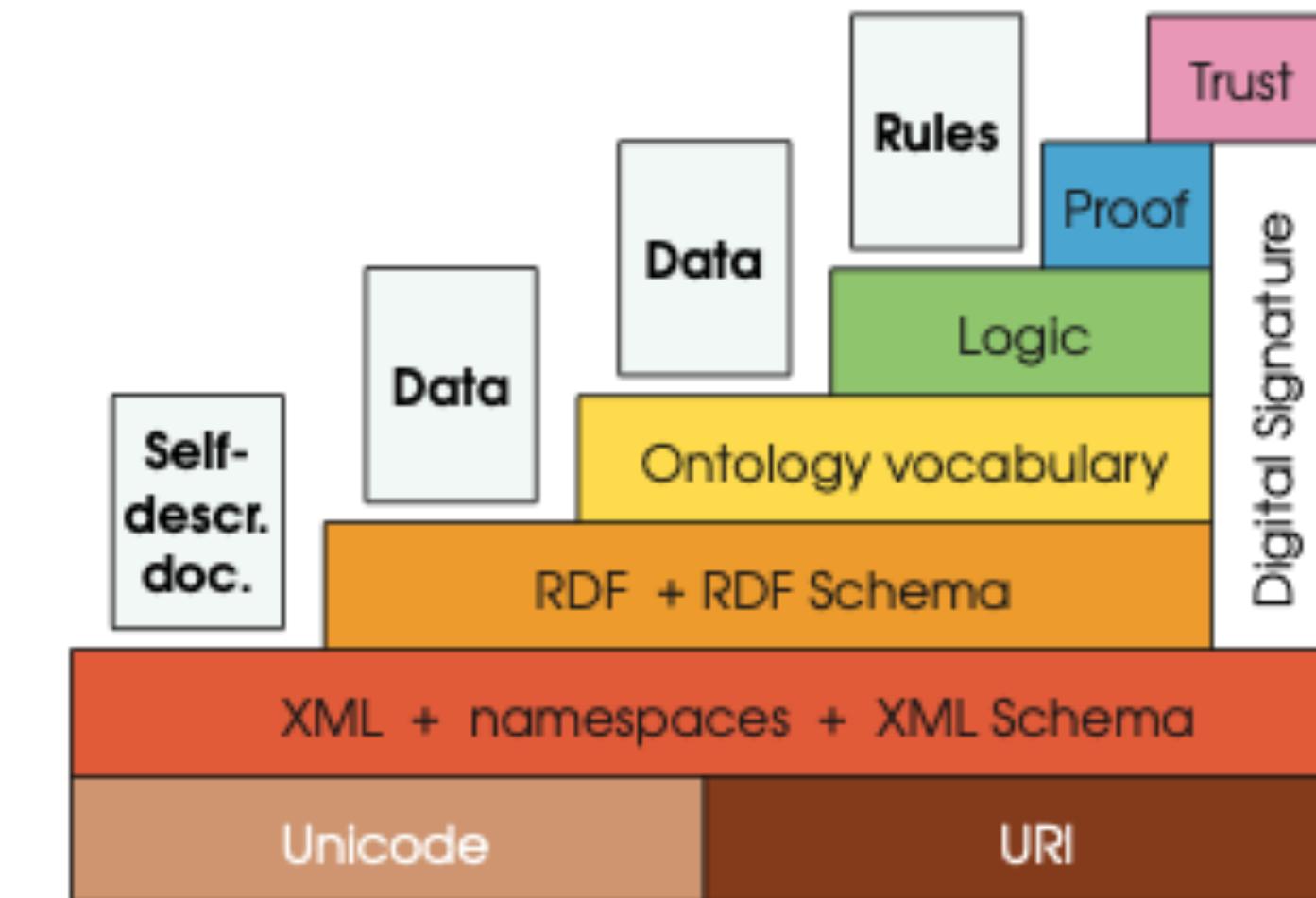
```
:romeo rdf:type :Wife.
```

Assertions

```
:romeo rdf:type :Husband.  
:romeo :wife_of : juliet.
```

There is no contradiction,
and the mis-modelling is
not diagnosed
automatically!

Layering of SW languages



T. Berners-Lee

Semantic + Web = Semantic Web

Represent Web content in a form that is more easily machine-processable:

describe meta-data about resources on the Web

i.e. descriptions about the data being represented, the model and constraints used to represent them.

Use intelligent techniques to take advantage of these representations:

process meta-data in a way that is similar to human reasoning and inference

thus information gathering can be done by a machine in a similar way to how humans currently gather information on the web..

Recap

- RDFS entailment
- Limitation of RDFS

COMP318: RDFS vs OWL

www.csc.liv.ac.uk/~valli/Comp318



Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

V.Tamma@liverpool.ac.uk

based on material by M. Rodriguez Muro, P. Hitzler and S. Rudolph

Where were we

- RDF entailment rules
- RDFS entailment rules

Brief recap of RDF/RDFS

- RDF/RDFS describe subject predicate object triples and basic subsumption relations.
 - Very efficient deduction/querying
 - SPARQL based on simple entailment, but...
 - Very poor expressivity...
 - Describe data in terms of triples (RDF)
 - Describe the model behind the data (RDFS) by:
 - Declare the “types” and properties/relationships of the things we want to make assertions about
 - Allowing to infer new assertions implicitly stated from a set of given facts
- But sometimes we need to express more advanced notions, e. g.:
 - A person has only one birth date
 - No person can be male and female at the same time

What can you represent with RDFS

- **RDFS provides:**
 - Classes
 - Lecturer rdf:type rdfs:Class
 - Class hierarchies
 - Lecturer rdfs:subClassOf AcademicStaff
 - Properties
 - teachesModule rdf:type rdf:Property
 - Property hierarchies
 - teachesModule rdfs:subPropertyOf coordinatesModule
 - Domain and range declarations
 - teachesModule rdfs:domain Lecturer
 - teachesModule rdfs:range Module
 - They infer information rather than checking data

What can't you represent in RDFS

- **RDFS does NOT provide:**
 - Disjointness of Classes
 - *Male and Female are disjoint*
 - *they cannot have any shared instances*
 - Property characteristics (inverse, transitive, ...)
 - *Lecturer teachesModule Module and Module isTaughtByLecturer Lecturer are not explicitly related*
 - Local scope of properties
 - *Lecturer has a property hasTitle whose values (range) is restricted to all values of the class PhD*
 - *only people who hold a PhD can be lecturers*
 - *but other AcademicStaffMembers can hold a BSc*
 - Complex concept definitions (Boolean combination of classes)
 - *Person = Man ∪ Woman*
 - *Mother = Woman ∩ Parent*

What can't you represent in RDFS

- **RDFS does NOT provide:**
 - Cardinality restrictions
 - Person may have at most 1 name
 - Lecturer *teaches exactly two modules*
 - A way to distinguish between classes and instances:
 - Feline rdf:type rdfs:Class
 - Cat rdf:type Feline
 - felix rdf:type Cat
 - :felix is an instance of an instance (Cat)
 - A way to distinguish between language constructors and ontology vocabulary:
 - rdf:type rdfs:range rdfs:Class
 - rdfs:Property rdfs:type rdfs:Class
 - rdf:type rdfs:subPropertyOf rdfs:subClassOf
 - Reasoning for these non-standard semantics

What can you infer in RDFS

Schema

```
ex:Lecturer  
rdfs:subClassOf  
ex: AcademicStaff
```

RDFS Entailment ⇒

Inferred Fact

```
staffUniv:john_smith  
rdf:type  
ex: AcademicStaff
```

Assertion

```
staffUniv:john_smith  
rdf:type  
ex: Lecturer
```

What can't you infer in RDFS

- However, not all types of inferences are possible in RDFS

Schema

```
:wife_of rdfs:subPropertyOf married_to.  
:married_to rdfs:domain :Spouse;  
           rdfs:range  :Spouse.  
:wife_of  rdfs:domain :Wife;  
           rdfs:range   :Husband.
```

Assertion

```
:juliet :wife_of :romeo.
```

Facts Inferred

```
:juliet rdf:type    :Wife;  
        rdf:type    :Spouse;  
        married_to :romeo;  
:romeo  rdf:type   :Spouse;  
        rdf:type   :Husband.
```

What can't you infer in RDFS

- What about if we want to model symmetry,
i.e. `:x :married_to :y`

~~implies `:x :married_to :y` \iff `:y :married_to :x`~~?

Schema

```
:wife_of rdfs:subPropertyOf married_to.  
:married_to rdfs:domain :Spouse;  
           rdfs:range  :Spouse.  
:wife_of rdfs:domain :Wife;  
           rdfs:range  :Husband.  
:husband_of rdfs:domain :Husband;  
           rdfs:range  :Wife.
```

Assertion

```
:juliet :wife_of :romeo.
```

Facts Inferred

```
:juliet rdf:type  :Wife;  
        rdf:type  :Spouse;  
        married_to :romeo;  
:romeo rdf:type  :Spouse;  
        rdf:type  :Husband.
```

Facts *NOT Inferred*

```
:romeo :married_to :juliet.  
:romeo :husband_of :juliet.
```

Too much representational freedom is not good!

- We might want to be able to detect what might seem inconsistent facts, but RDFS is not able to constrain models through consistency and axioms:

Schema

```
:wife_of rdfs:subPropertyOf married_to.  
:married_to rdfs:domain :Spouse;  
           rdfs:range  :Spouse.  
:wife_of rdfs:domain :Wife;  
           rdfs:range  :Husband.  
:husband_of rdfs:domain :Husband;  
           rdfs:range  :Wife.
```

Facts *INCORRECTLY Inferred*

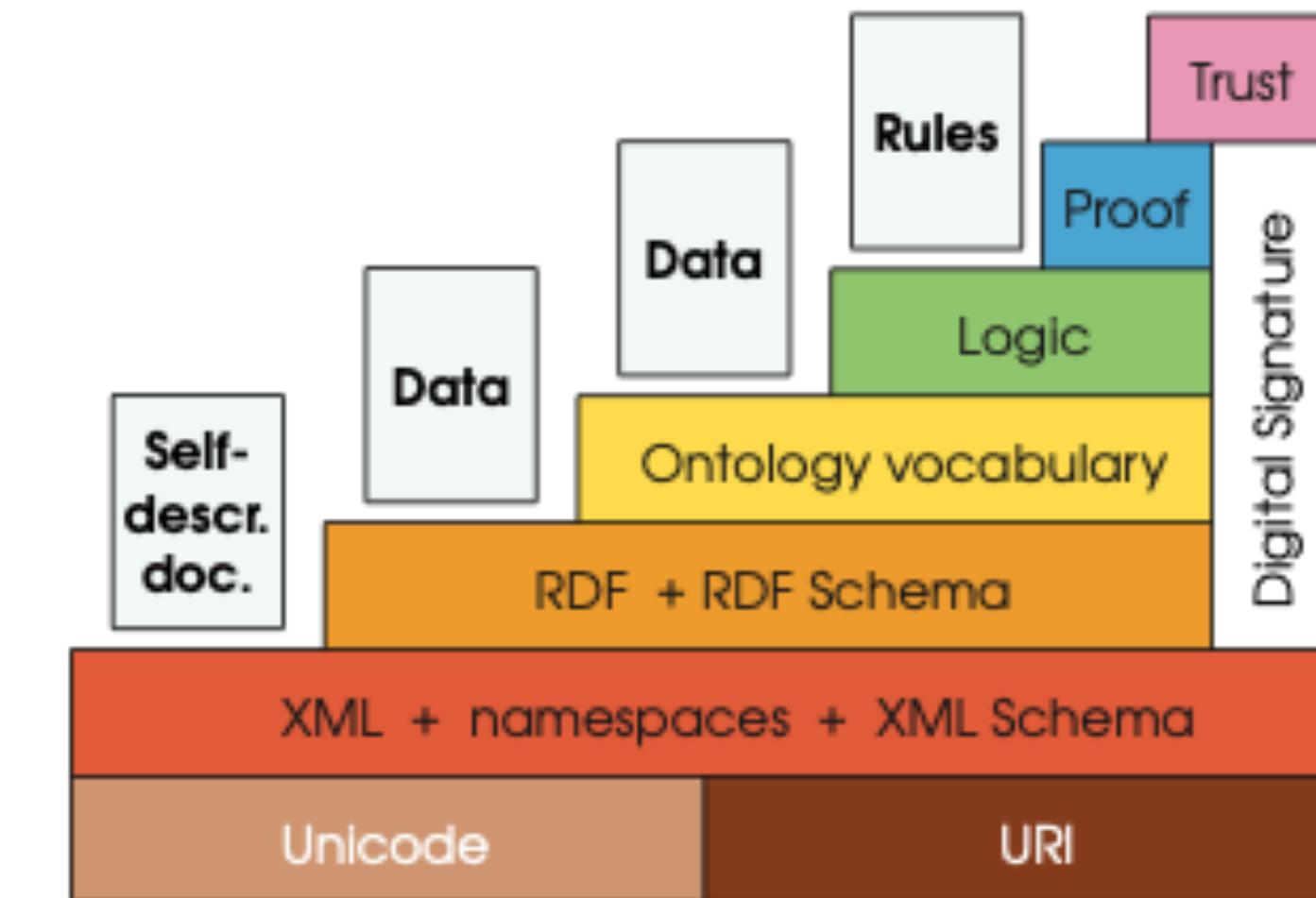
```
:romeo rdf:type :Wife.
```

Assertions

```
:romeo rdf:type :Husband.  
:romeo :wife_of : juliet.
```

There is no contradiction,
and the mis-modelling is
not diagnosed
automatically!

Layering of SW languages



T. Berners-Lee

Semantic + Web = Semantic Web

Represent Web content in a form that is more easily machine-processable:

describe meta-data about resources on the Web

i.e. descriptions about the data being represented, the model and constraints used to represent them.

Use intelligent techniques to take advantage of these representations:

process meta-data in a way that is similar to human reasoning and inference

thus information gathering can be done by a machine in a similar way to how humans currently gather information on the web..

Conflicting aims

- Ontologies provide the structured vocabulary for describing meta-data
 - Languages to represent ontologies need to be as expressive as possible whilst permitting automated deduction:
 - To describe meta-data, we want a (logic-based) language that is as expressive as possible.
 - To simulate human deduction in an efficient way, we want a logic that permits efficient automated deduction.
 - The logic of choice is a compromise between expressiveness and complexity of deduction.

Wish list for ontology languages....

- Compromise between expressivity and scalability
 - Well defined syntax
 - necessary for automatic machine-processing of information;
 - Formal semantics
 - describe the meaning of knowledge precisely, it allows computers to reason precisely about knowledge
 - class membership: if $x \in A$ and $A \subseteq B$ then we can infer that $x \in B$
 - consistency: $x \in A$, $A \subseteq B \cap C$, $A \subseteq D$ and $B \cap D = \emptyset$ then we have an inconsistency!
 - classification: if some property-value pairs are a sufficient condition for membership in A, and x satisfies them, then infer $x \in A$
 - Efficient reasoning
 - derivations can be computed mechanically
 - consistency, classification, detection of unintended relationships between classes;
 - Sufficient expressive power
 - Compatibility with RDF and RDFS;

Requirements for Ontology Languages

- The main requirements are:

- a well-defined syntax
- a formal semantics
- convenience of expression
- efficient (complex) reasoning support
- sufficient expressive power

RDFS

OWL

Extending RDFS

- OWL extends RDF Schema to a knowledge representation language for the Web
 - Logical expressions (and, or, not)
 - Woman = Human and Female
 - Person = Man or Woman
 - Man = not (Woman)
 - (in)equality
 - john differentFrom mary
 - There is no assumption that individuals have only one name, thus the need to explicitly state inequality
 - local properties
 - Lecturer only teaches modules

Extending RDFS

- OWL extends RDF Schema to a knowledge representation language for the Web
 - required/optional properties
 - teaches Modules is a required property for Lecturer
 - required values
 - BritishCitizen hasNationality = “British”
 - the value for the hasNationality property for the class BritishCitizen can only be “British”
 - enumerated classes
 - DaysOfTheWeek = {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday}
 - symmetry, inverse

Recap

- Limitation of RDFS
- Introduction to OWL

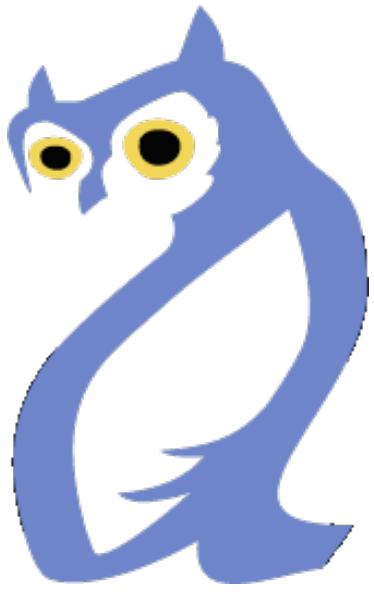
Reasoning support

- Formal semantics allows the automatic deduction of new facts and possible conflicts between class definitions (consistency):

An ontology language can be provided with formal semantics and reasoning support by mapping it to a known logical formalism and by using the reasoning tools developed for the chosen formalism.

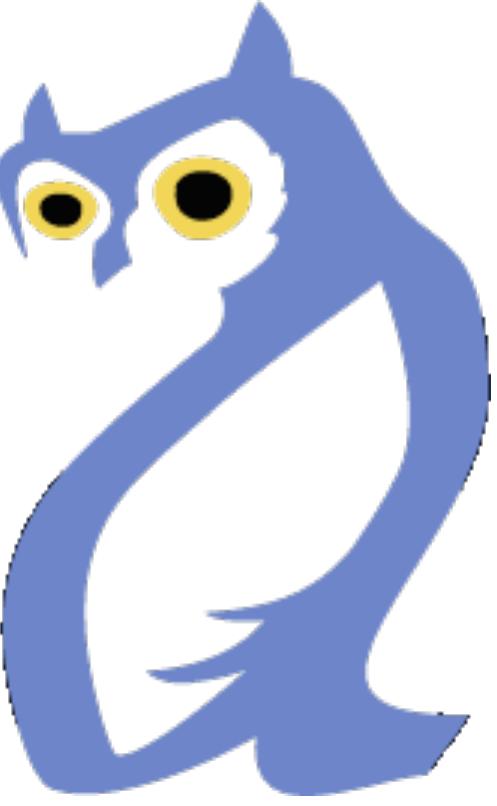
- These checks are extremely valuable for designing large ontologies, for collaborative ontology design and for sharing and integrating ontologies from various sources:
 - check the consistency of the ontology;
 - check for unintended relations between classes;
 - check for the unintended classification of instances.

OWL 1 and OWL 2



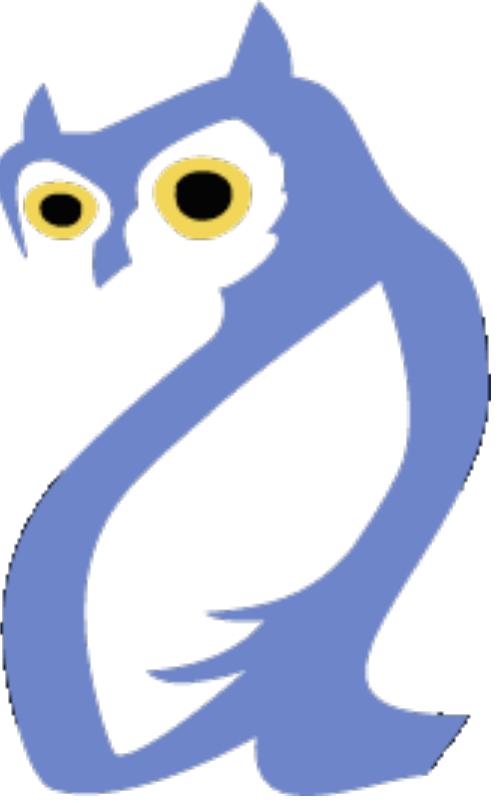
- OWL: OWL 1 (<http://www.w3.org/TR/owl-features/>) and OWL 2 (<http://www.w3.org/TR/owl2-overview/>)
 - Thus, from the absence of a statement alone, a deductive reasoner cannot infer that the statement is false.
 - Reasonable trade-off between expressivity and scalability
 - Fully declarative semantics.
- OWL 2 DL
 - Fragment of first order predicate logic, decidable
 - Known complexity classes
 - Reasonably efficient for real ontologies + instances
- Rationale for OWL
 - Open world assumption:
 - The absence of a particular statement means that the statement has not been made explicitly yet.
 - Whether the statement is true or not, and whether it is believed that it is (or would be) true or not is irrelevant.

OWL 1: Three species of OWL

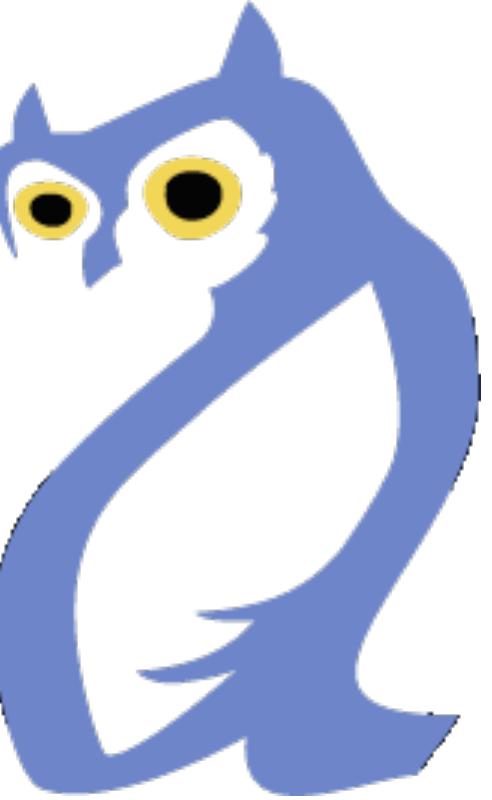


- OWL Lite:
 - Sublanguage of OWL DL but without nominals and XML datatypes:
 - Classification hierarchy
 - Simple constraints
 - It excludes enumerated classes, disjointness statements, and arbitrary cardinality.
 - Reasoning still not tractable.
- OWL DL
 - Sublanguage of OWL Full, it imposes restrictions on the use of OWL/RDFS constructors
 - Application of OWL's constructors to each other not permitted
 - Provides reasonably efficient reasoning support.

OWL 1: Three species of OWL



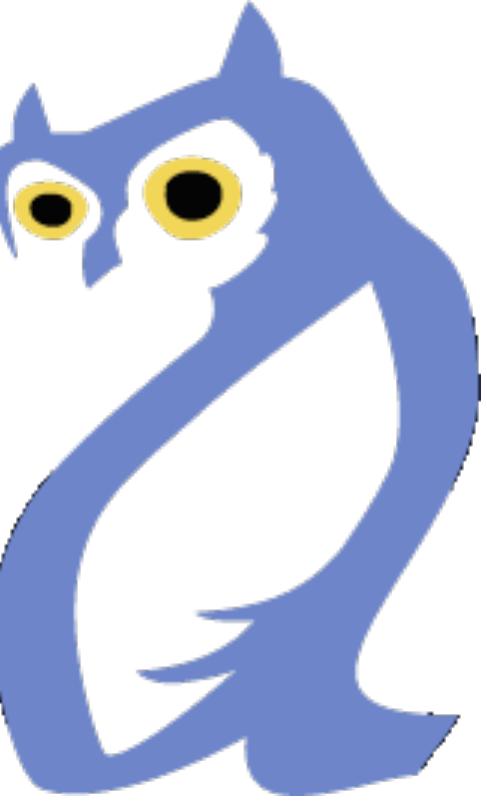
- OWL Full
 - Very high expressiveness, uses all of the OWL primitives
 - Fully upward compatible with RDF
 - Losing decidability: no complete or efficient reasoning support
 - All syntactic freedom of RDF (self-modifying):
 - primitives can be combined in arbitrary ways with RDF(S)



OWL 2 - Profile

- Sublanguages of OWL2 trading expressive power for efficient reasoning
 - Each supports different application scenarios
- OWL 2 EL
 - very large ontologies, efficient reasoning performance guaranteed at the expenses of expressive power;
- OWL 2 RL
 - subclass axioms understood as rule like implication, with head - superclass and body - subclass
 - different restrictions on subclasses and superclasses
 - allows the integration of OWL with rules

OWL 2 - Profile



- Sublanguages of OWL2 trading expressive power for efficient reasoning
 - Each supports different application scenarios
- OWL 2 QL
 - useful to query data rich applications
- different restrictions on subclasses and superclasses
- suitable for simple, lightweight ontologies with a large number of individuals and it is necessary to access the data directly via SQL queries
- fast implementation on top of legacy DB systems, relational or RDF

COMP318: Introduction to OWL

www.csc.liv.ac.uk/~valli/Comp318



Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

V.Tamma@liverpool.ac.uk

Where were we

- Limitations of RDF(S) as a modelling language
 - Characteristics
 - Wishlist for a Web ontology language

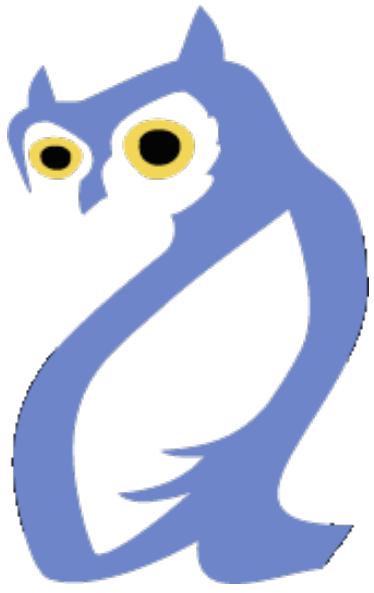
Reasoning support

- Formal semantics allows the automatic deduction of new facts and possible conflicts between class definitions (consistency):

An ontology language can be provided with formal semantics and reasoning support by mapping it to a known logical formalism and by using the reasoning tools developed for the chosen formalism.

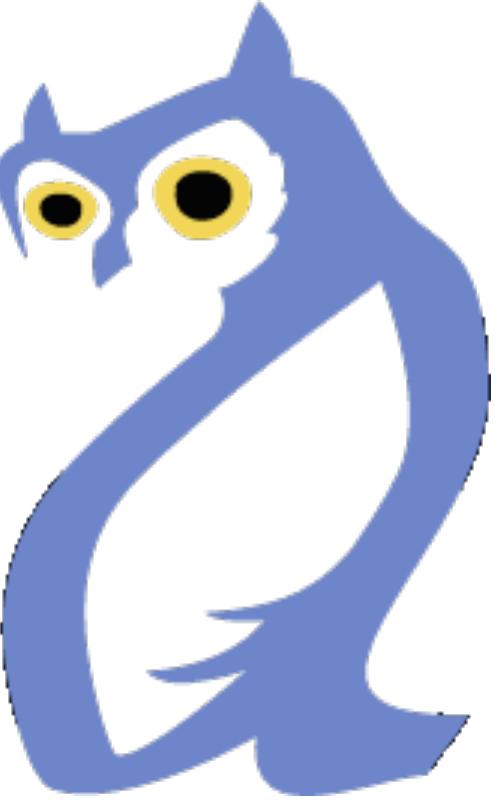
- These checks are extremely valuable for designing large ontologies, for collaborative ontology design and for sharing and integrating ontologies from various sources:
 - check the consistency of the ontology;
 - check for unintended relations between classes;
 - check for the unintended classification of instances.

OWL 1 and OWL 2



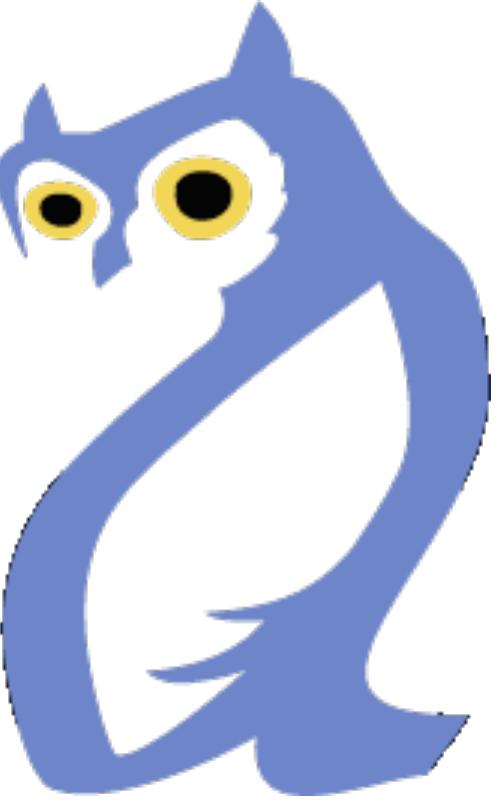
- OWL: OWL 1 (<http://www.w3.org/TR/owl-features/>) and OWL 2 (<http://www.w3.org/TR/owl2-overview/>)
 - Thus, from the absence of a statement alone, a deductive reasoner cannot infer that the statement is false.
 - Reasonable trade-off between expressivity and scalability
 - Fully declarative semantics.
- Rationale for OWL
 - **Open world assumption:** the absence of a particular statement means that the statement has not been made explicitly yet.
 - Whether the statement is true or not, and whether it is believed that it is (or would be) true or not is irrelevant.
- OWL 2 DL
 - Fragment of first order predicate logic, decidable
 - Known complexity classes
 - Reasonably efficient for real ontologies + instances

OWL 1: Three species of OWL

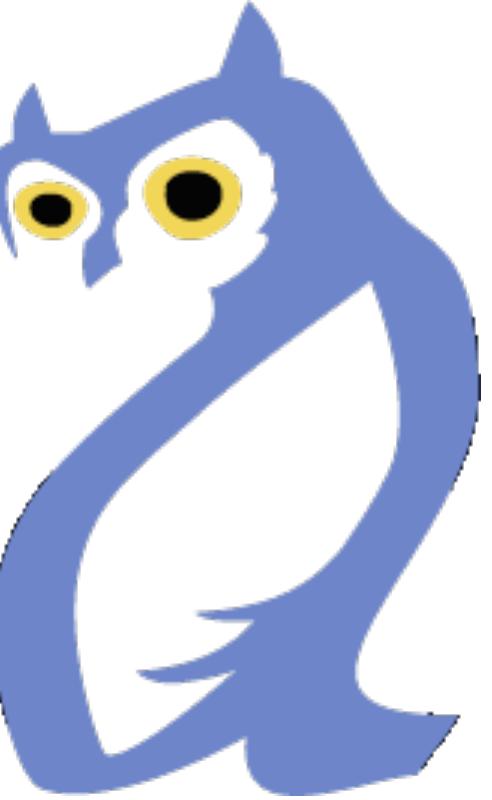


- OWL Lite:
 - Sublanguage of OWL DL but without nominals and XML datatypes:
 - Classification hierarchy
 - Simple constraints
 - It excludes enumerated classes, disjointness statements, and arbitrary cardinality.
 - Reasoning still not tractable.
- OWL DL
 - Sublanguage of OWL Full, it imposes restrictions on the use of OWL/RDFS constructors
 - Application of OWL's constructors to each other not permitted
 - Provides reasonably efficient reasoning support.

OWL 1: Three species of OWL



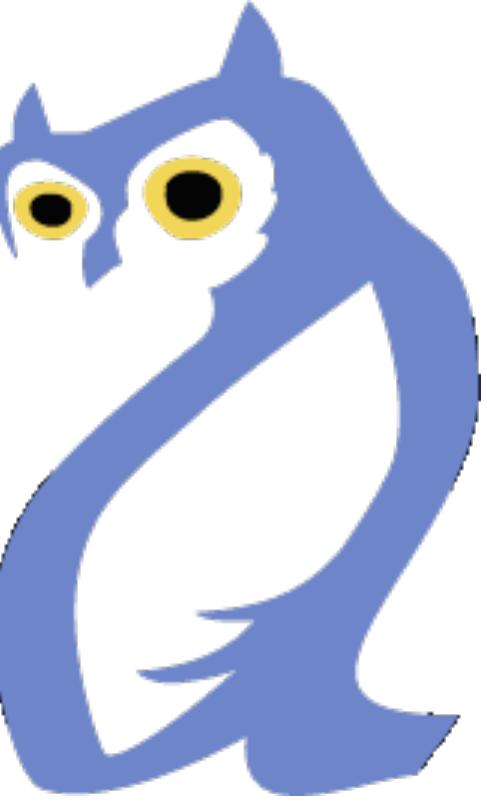
- OWL Full
 - Very high expressiveness, uses all of the OWL primitives
 - Fully upward compatible with RDF
 - Losing decidability: no complete or efficient reasoning support
 - All syntactic freedom of RDF (self-modifying):
 - primitives can be combined in arbitrary ways with RDF(S)



OWL 2 - Profile

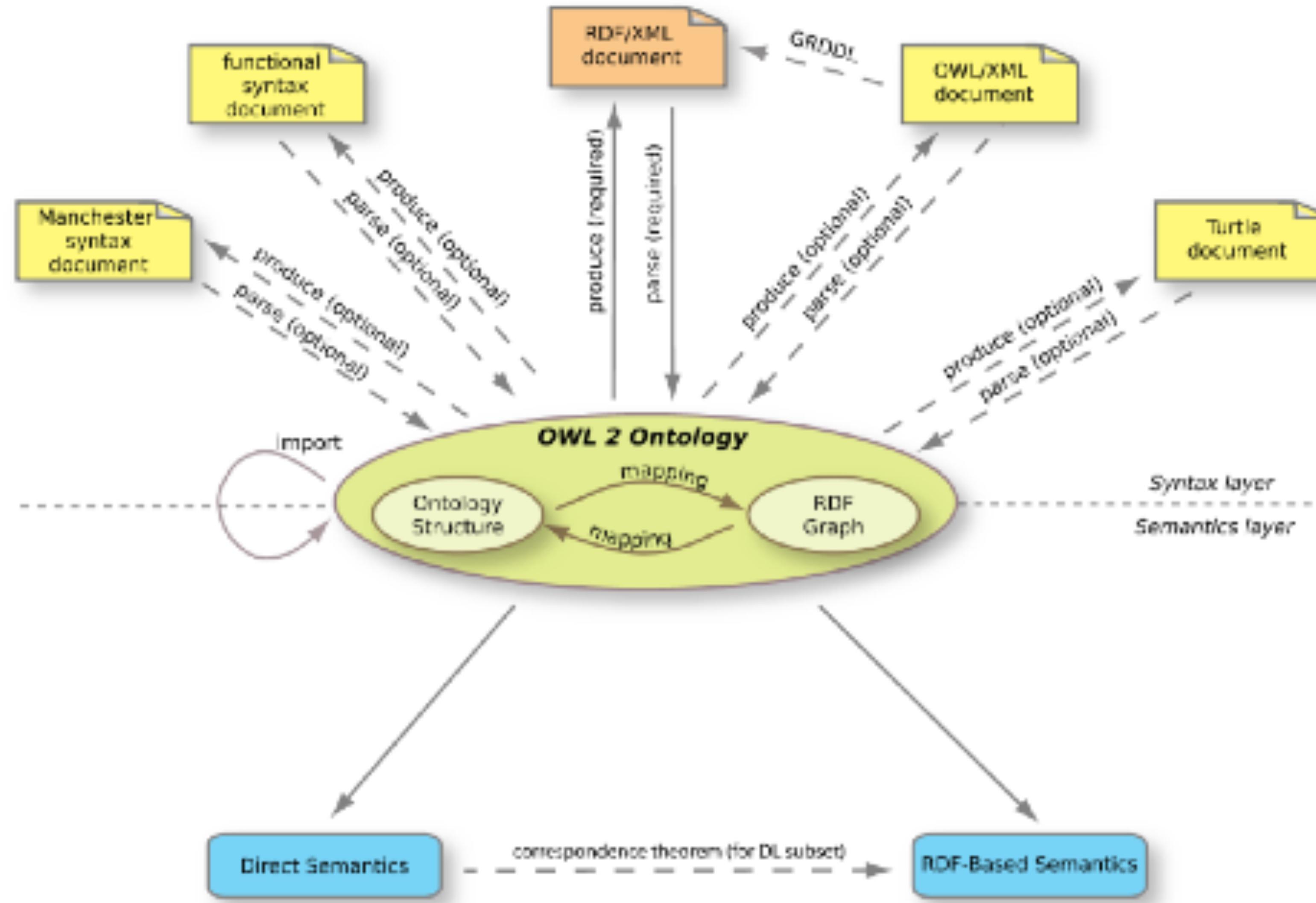
- Sublanguages of OWL2 trading expressive power for efficient reasoning
 - Each supports different application scenarios
- OWL 2 EL
 - very large ontologies, efficient reasoning performance guaranteed at the expenses of expressive power;
- OWL 2 RL
 - subclass axioms understood as rule like implication, with head - superclass and body - subclass
 - different restrictions on subclasses and superclasses
 - allows the integration of OWL with rules

OWL 2 - Profile



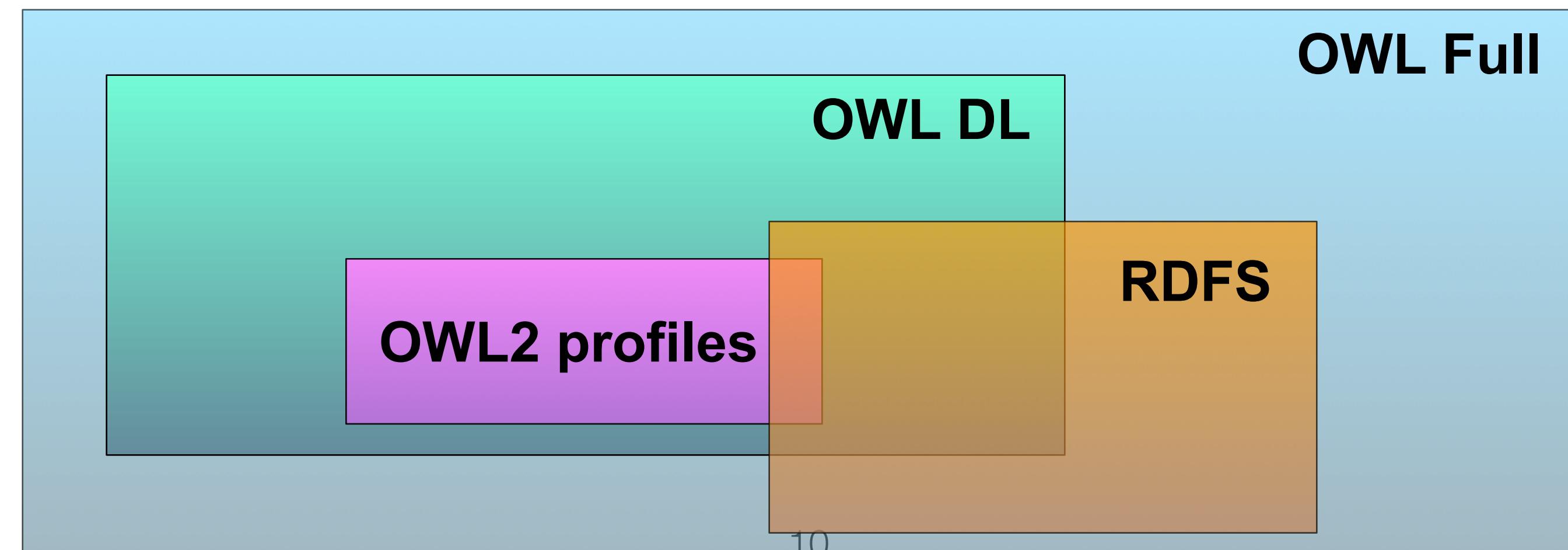
- Sublanguages of OWL2 trading expressive power for efficient reasoning
 - Each supports different application scenarios
- OWL 2 QL
 - useful to query data rich applications
- different restrictions on subclasses and superclasses
- suitable for simple, lightweight ontologies with a large number of individuals and it is necessary to access the data directly via SQL queries
- fast implementation on top of legacy DB systems, relational or RDF

OWL 2 structure



Compatibility between OWL and RDF(S)

- OWL uses to a great extent RDF(S)
 - One of the possible syntax formats for OWL is in RDF/XML
 - instances are declared in RDF
 - using `rdf:Description` and `rdf:type`
 - The OWL constructs `owl:Class`, `owl:DatatypeProperty` and `owl:ObjectProperty` are specialisations of the corresponding RDFS constructs



OWL syntax

- RDF
 - Official exchange syntax
 - Hard for humans
 - RDF parsers are hard to write!
 - More XML than RDF tools available
 - <http://www.w3.org/TR/owl-xmlsyntax/>
 - <http://www.w3.org/TR/owl2-mapping-to-rdf>
- UML
 - Large user base
- Human readable syntax
 - Manchester syntax, used by Protege
 - Functional syntax, more compact and readable
 - <http://www.w3.org/2007/OWL/wiki/ManchesterSyntax>
 - <http://www.w3.org/TR/owl2-manchester-syntax/>
 - Turtle syntax for OWL 2
 - <http://www.w3.org/TR/owl2-primer/>
- XML
 - Not the RDF syntax, it is not based on RDF conventions
 - Better for humans

OWL ontology header

```
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"  
          xmlns:xsd = "http://www.w3.org/2001/XMLSchema#"  
          xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
          xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"  
          xml:base="http://www.dcs.bbk.ac.uk/">  
  
<owl:Ontology rdf:about=""/>  
  <rdfs:comment>An example OWL ontology</rdfs:comment>  
  <owl:priorVersion rdf:resource="http://www.dcs.bbk.ac.uk/uni-old-ns"/>  
  <owl:imports rdf:resource="http://www.dcs.bbk.ac.uk/person"/>  
  <rdfs:label>SCSIS Ontology</rdfs:label>  
</owl:Ontology>  
  
...  
</rdf:RDF>
```

OWL namespace

Assertions for housekeeping purposes

Namespaces vs import

```
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"  
          xmlns:xsd = "http://www.w3.org/2001/XMLSchema#" ...>  
  
<owl:Ontology rdf:about=""">  
  <rdfs:comment>An example OWL ontology</rdfs:comment>  
  <owl:priorVersion rdf:resource="http://www.dcs.bbk.ac.uk/uni-old-ns"/>  
  <owl:imports rdf:resource="http://www.dcs.bbk.ac.uk/person"/>  
  <rdfs:label>SCSIS Ontology</rdfs:label>  
</owl:Ontology> ...</rdf:RDF>
```

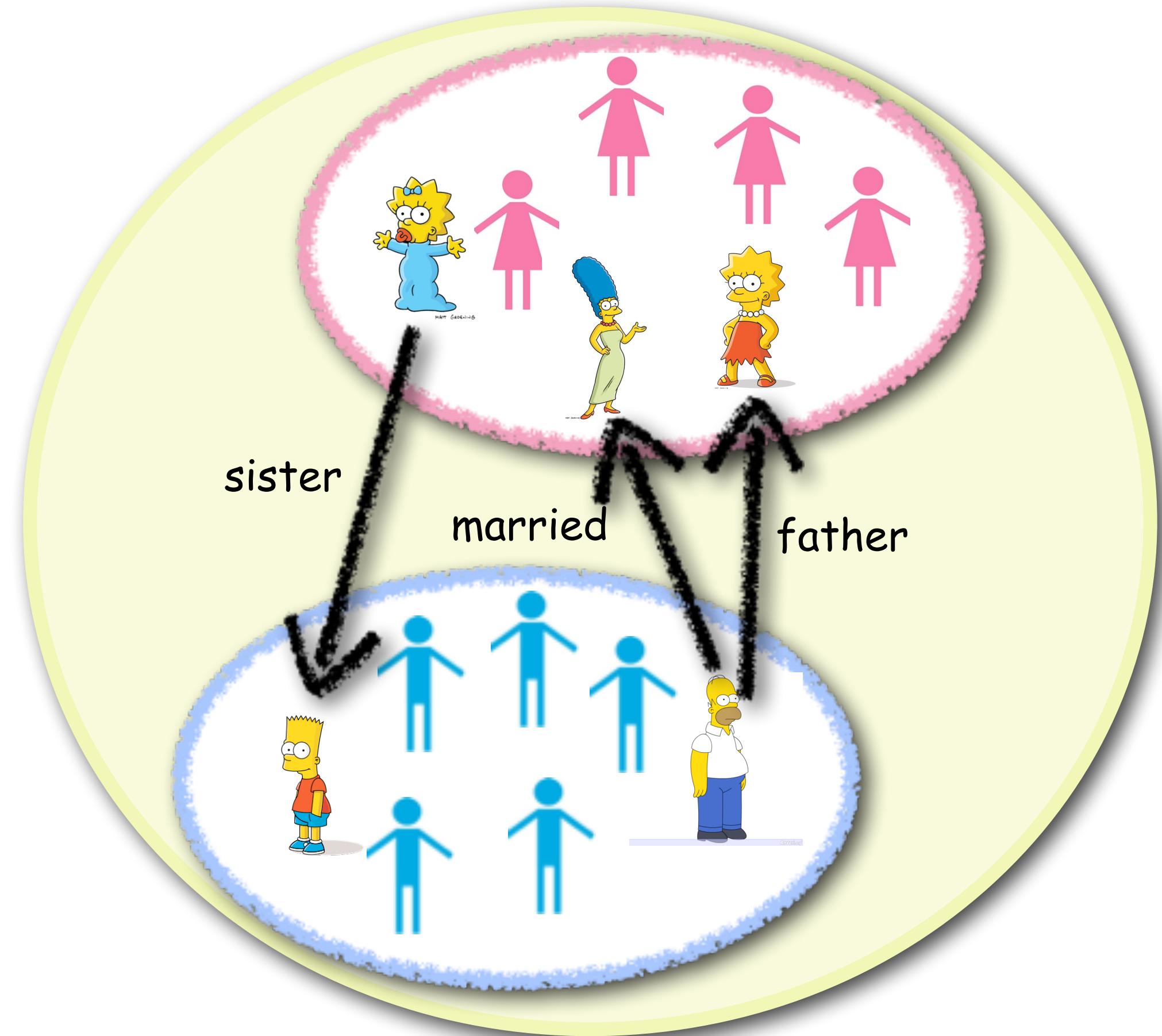
OWL namespace

Import of an ontology

Definitions from imported ontologies can be used as if they were asserted in the ontology where they are imported;
Namespaces allow users to disambiguate between similar class or property names defined in different ontologies.

What do we describe with OWL

- OWL (we assume DL) ontologies describe a world in terms of:
 - individuals (constants): *homer*, *lisa*, ...
 - classes (unary predicates): *man(x)*, *woman(x)*, *lazy(x)*, *clever(x)*, ...
 - properties/roles (binary predicates): *sister_of(x,y)*, *works_for(x,y)*...



Assertional knowledge (instances)

- Instances assert information about named individuals
- It is restricted to what can be stated in RDF
 - class membership: *female(marge)*
 - property membership: *married(marge, homer)*
 - use `rdf:ID` and `rdf:about` **almost** interchangeably

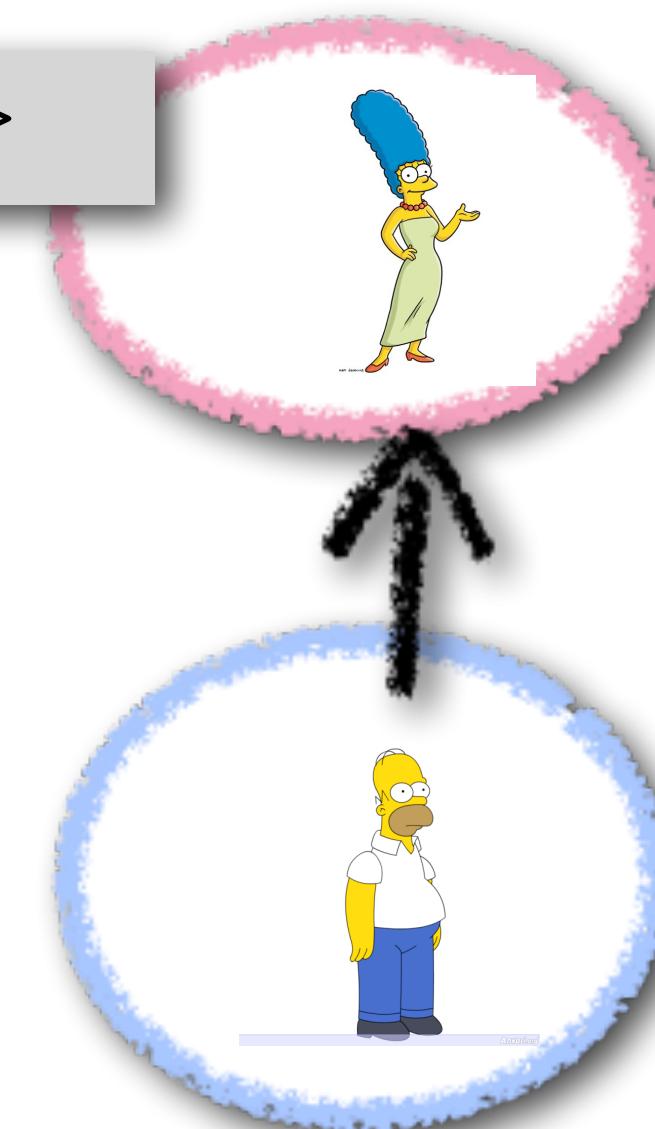
```
<rdf:Description rdf:ID="marge">  
  <rdf:type rdf:resource="#woman"/>  
</rdf:Description>
```

It declares the individual,
it can be used only once
in the document

```
<woman rdf:about="#marge"/>
```

It references the
individual, it can be used
as many times as needed

```
<rdf:Description rdf:about="marge">  
  <married rdf:resource="homer"/>  
</rdf:Description>
```



Unique Name Assumption

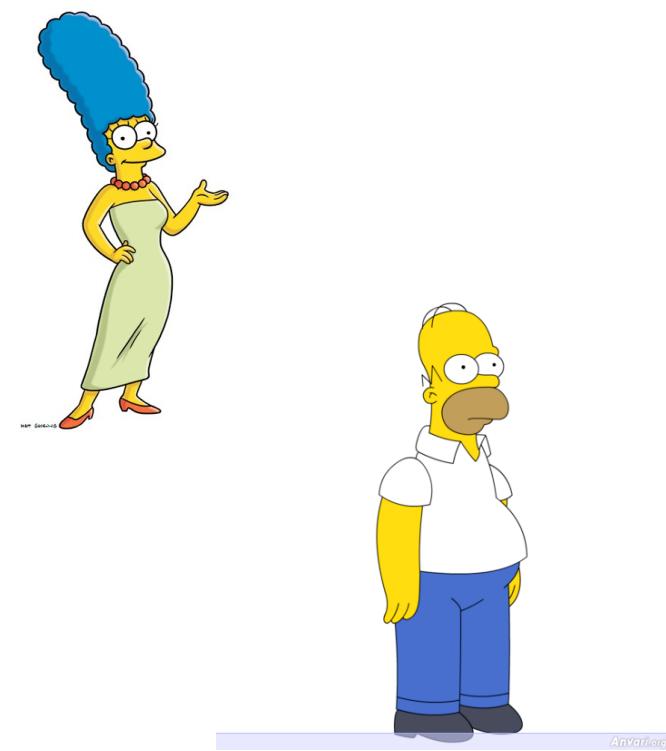
- In logics with the unique name assumption, different names always refer to different entities in the world.

- Despite being based on description logic, for which UNA holds, OWL does not make this assumption

- explicit constructs are used to express whether two names refer to the same or different entities
 - `owl:sameAs` - URIs refer to the same entity or individual
 - `owl:differentFrom` - URIs refer to different entities or individual

```
<rdf:Description rdf:about="#marge">
  <owl:sameAs rdf:resource="#margeSimpson">
</rdf:Description>

<rdf:Description rdf:about="#homer">
  <owl:differentFrom rdf:resource="#marge" />
</rdf:Description>
```



Terminological knowledge: classes and subclasses

- Classes are defined using owl:Class
 - subclass of rdfs:Class

```
<owl:Class rdf:ID="parents">
  <rdfs:subClassOf rdf:resource="#people" />
</owl:Class>
```

```
<owl:Class rdf:about="#children">
  <owl:disjointWith rdf:resource="#parents" />
</owl:Class>
<owl:Class rdf:ID="offspring">
  <owl:equivalentClass rdf:resource="#children" />
</owl:Class>
```



`owl:Thing` and `owl:Nothing`

- OWL has two predefined classes
 - `owl:Thing`
 - \top (in DL formalism)
 - class containing all individuals
 - `owl:Nothing`
 - \perp (in DL formalism)
 - “empty” class containing no individuals
- For every class C
 - `owl:Nothing` is a subclass of C
 - C is a subclass of `owl:Thing`

OWL class constructors

- Boolean combinations are used to define classes
 - married and single are disjoint but parent and single are not!

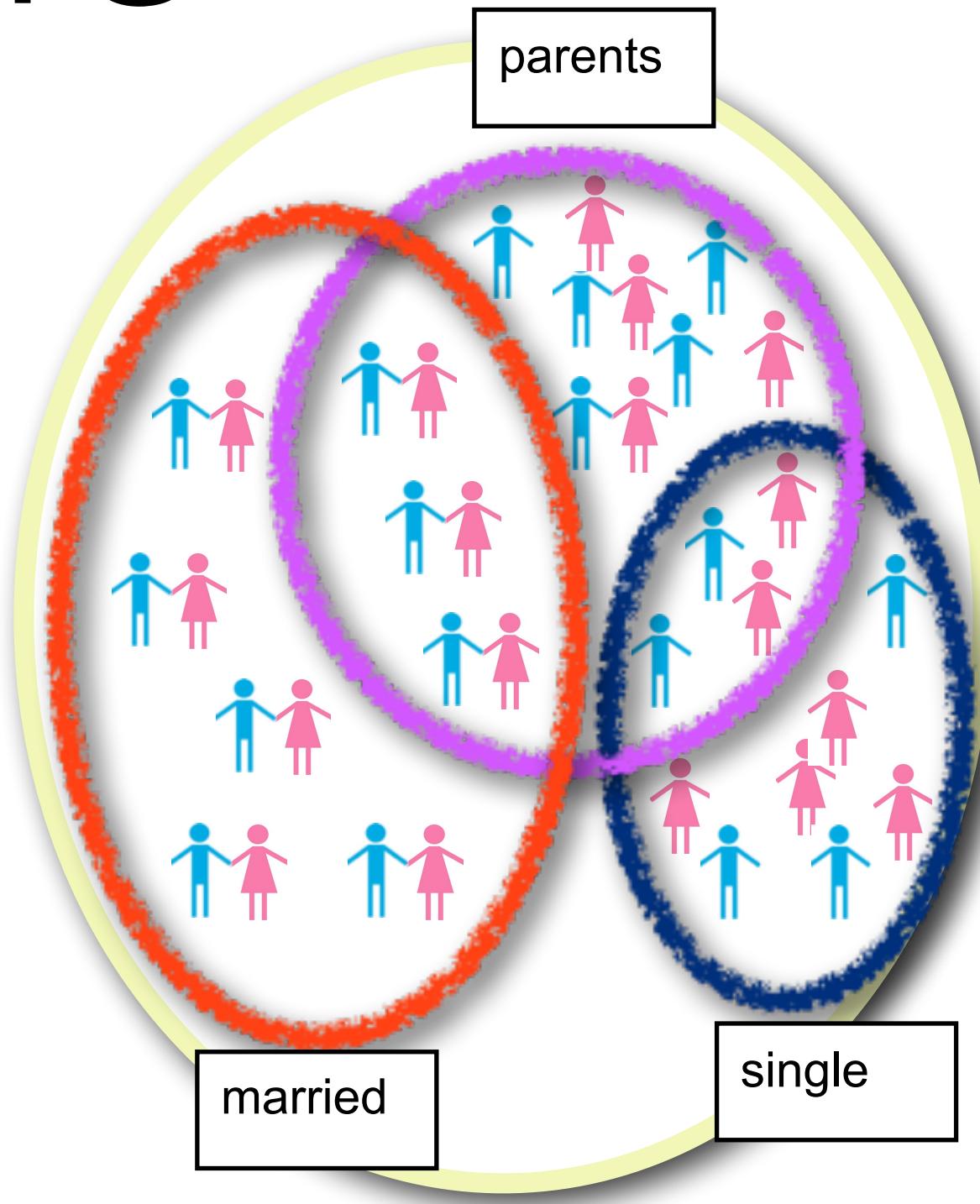
```
Class: Married  
EquivalentTo: not Single
```

Manchester syntax

```
:married owl:subClassOf [  
    rdf:type owl:Class ;  
    owl:intersectionOf (  
        [ rdf:type owl:Class ;  
        owl:complementOf :single ] )  
] .
```

turtle syntax

```
<owl:Class rdf:about="#married">  
    <owl:subClassOf>  
        <owl:Class>  
            <owl:complementOf rdf:resource="#single"/>  
        </owl:Class>  
    </owl:subClassOf>  
</owl:Class>
```



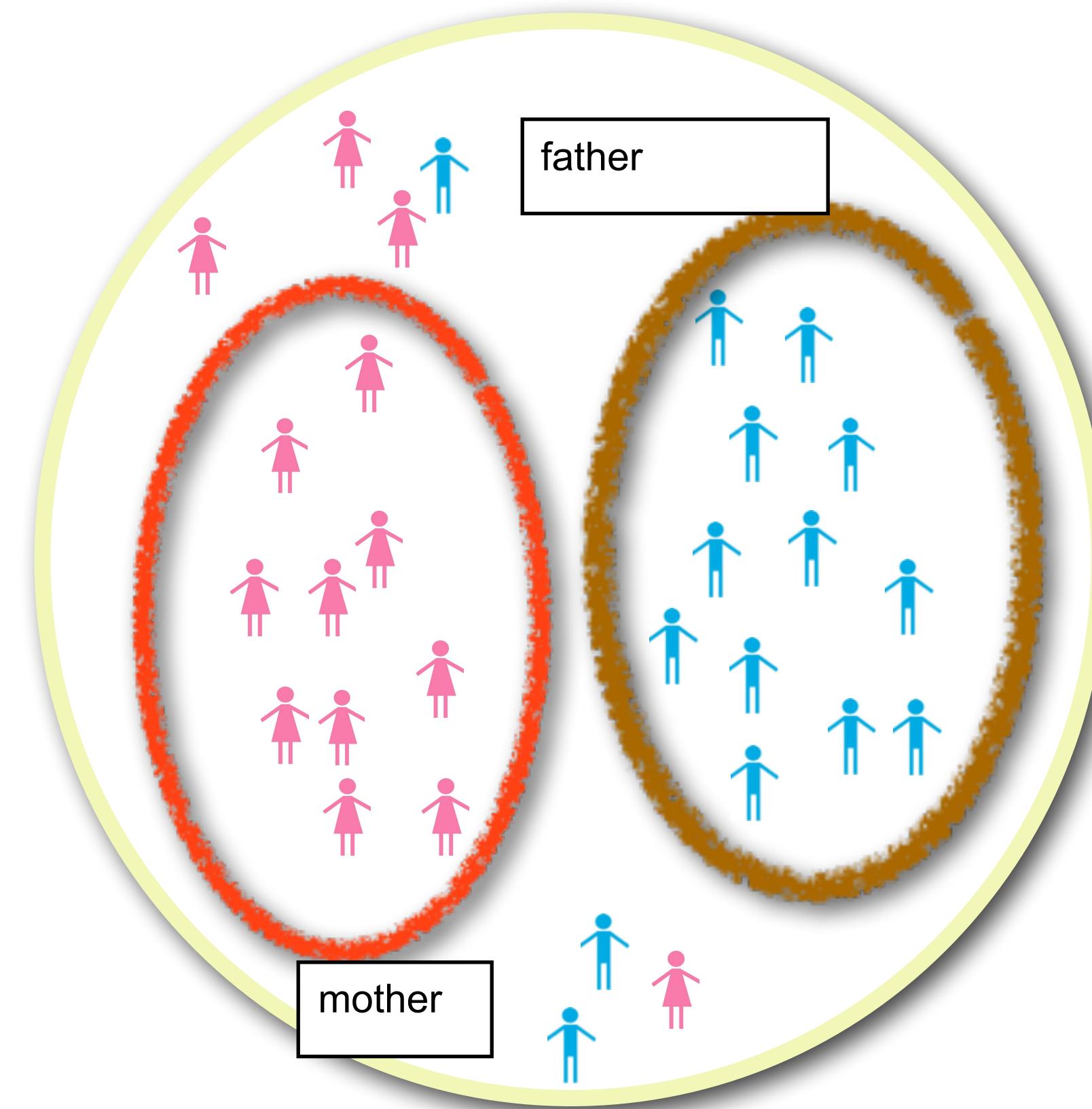
OWL class constructors

- parents are mothers and father

```
<owl:Class rdf:about="Parent">
  <owl:equivalentClass>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="Mother"/>
        <owl:Class rdf:about="Father"/>
      </owl:unionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

Class: Parent
EquivalentTo: Mother or Father

```
:Parent owl:equivalentClass [
  rdf:type owl:Class ;
  owl:unionOf ( :Mother :Father )
] .
```



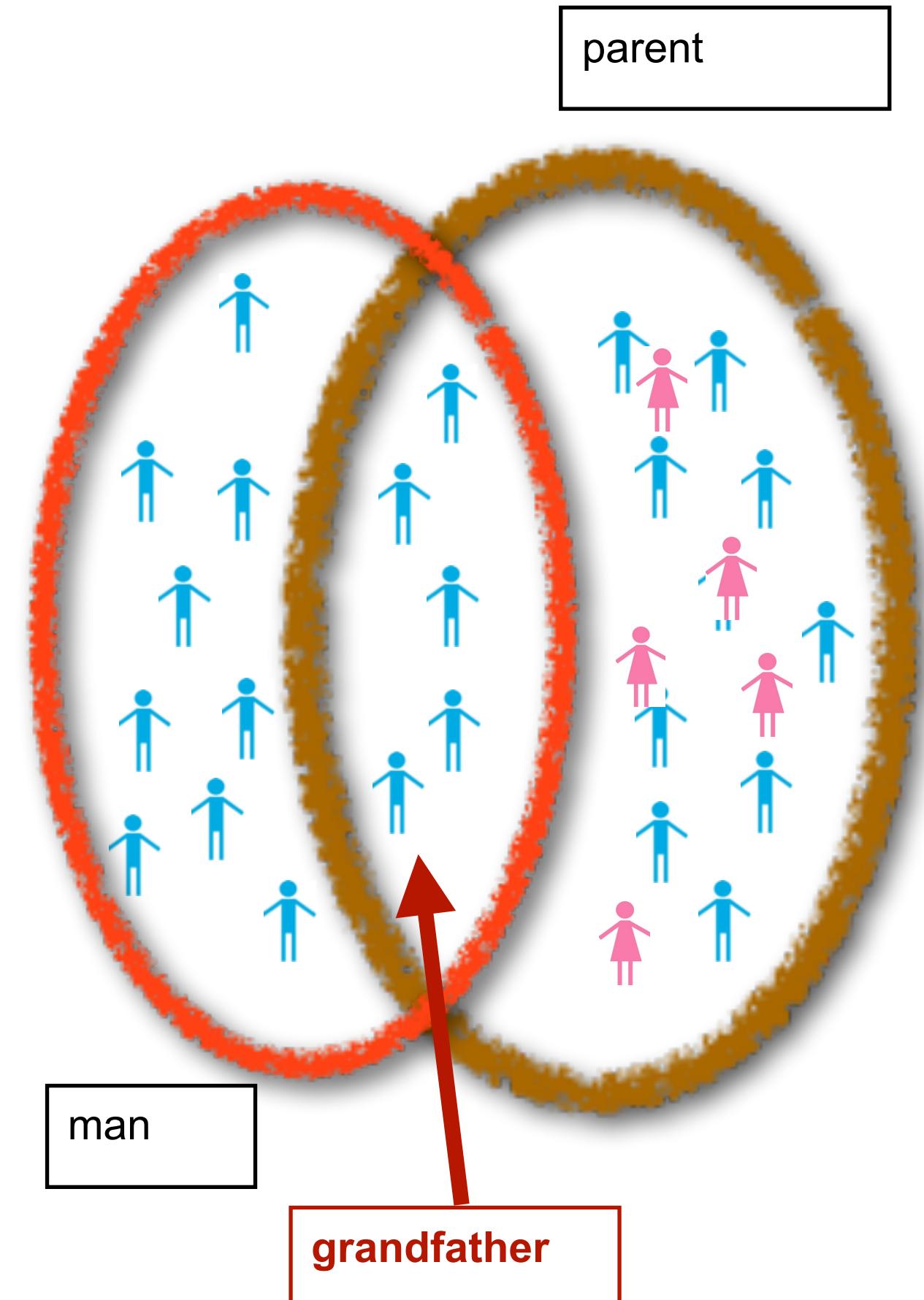
OWL class constructors

- Grandfather is both a man and a father

```
<owl:Class rdf:about="Grandfather">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="Man"/>
        <owl:Class rdf:about="Parent"/>
      </owl:intersectionOf>
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

Class: Grandfather
SubClassOf: Man and Parent

```
:Grandfather rdfs:subClassOf [
  rdf:type owl:Class ;
  owl:intersectionOf ( :Man :Parent )
] .
```

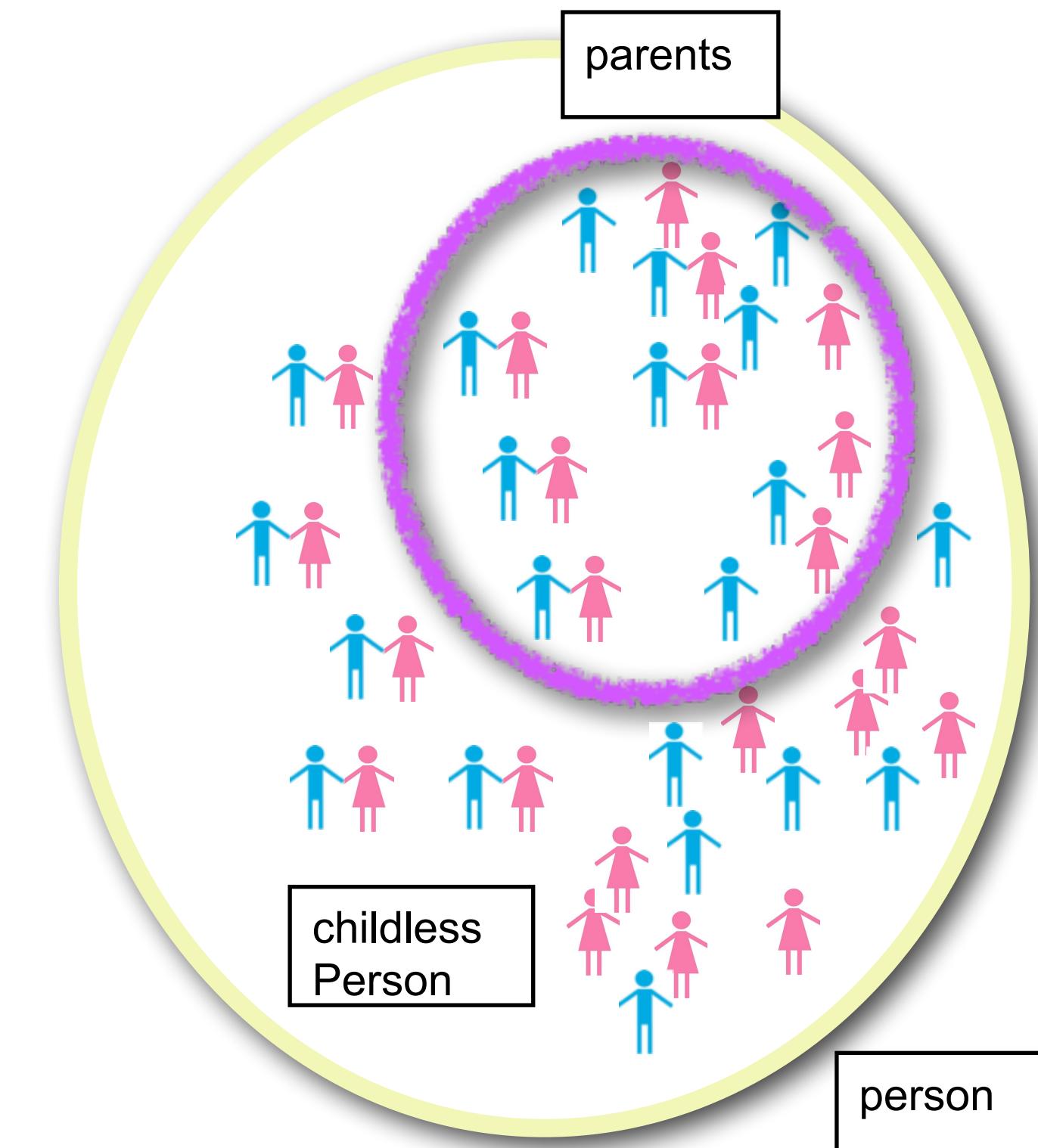


OWL class constructors

- a childless person is someone who is a person but not a parent

```
<owl:Class rdf:about="ChildlessPerson">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="Person"/>
        <owl:Class>
          <owl:complementOf rdf:resource="Parent"/>
        </owl:Class>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

```
:ChildlessPerson owl:equivalentClass [
  rdf:type owl:Class ;
  owl:intersectionOf ( :Person
    [ rdf:type owl:Class ;
      owl:complementOf :Parent ] ) ]
].
```



```
Class: ChildlessPerson
EquivalentTo: Person and not Parent
```

OWL class constructors

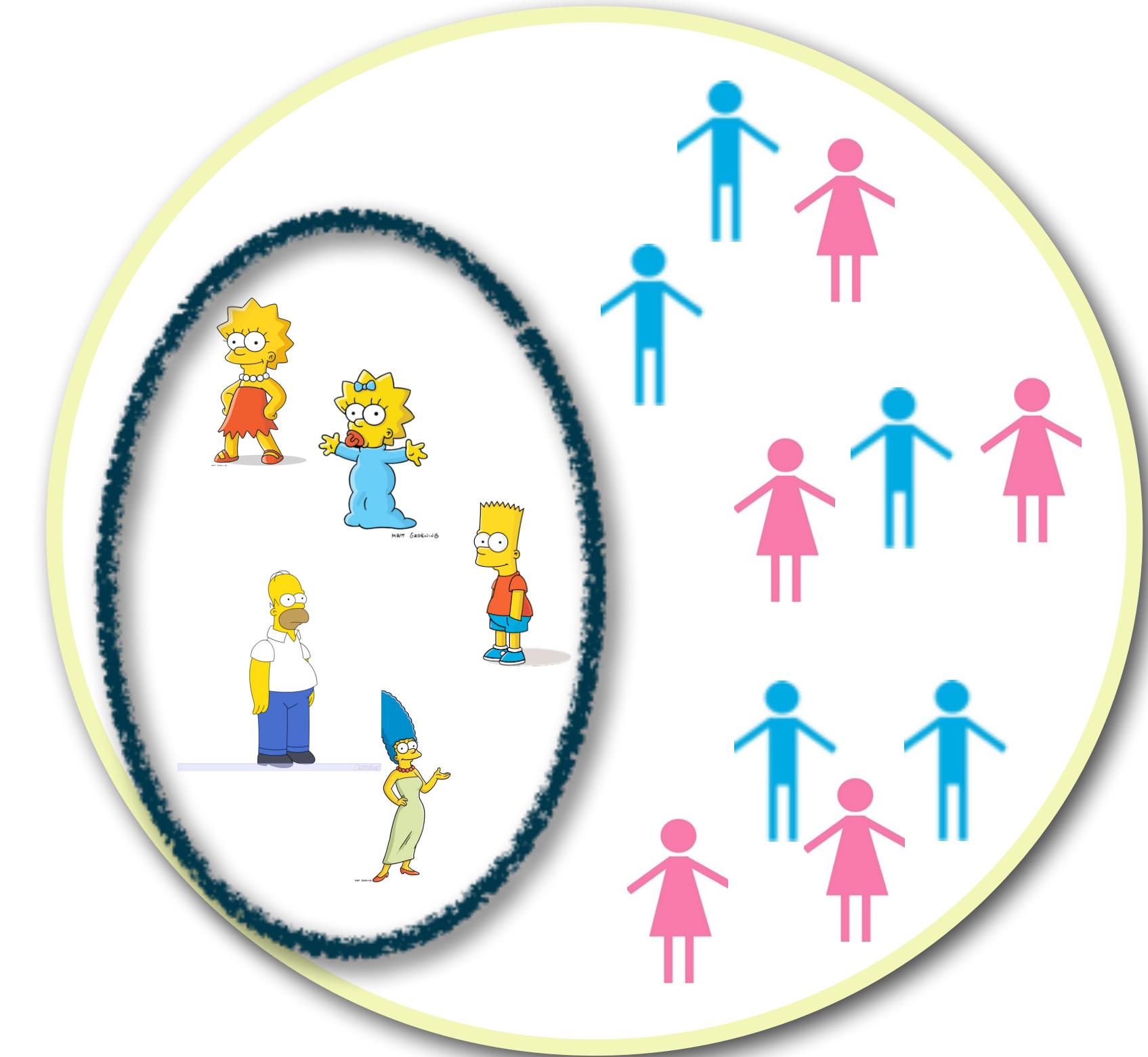
- Classes can also be defined through enumeration using **owl:oneOf**

- allows a class to be defined extensionally,
 - with exactly the enumerated individual

```
<owl:Class rdf:about="#simpsonFamily">
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#marge"/>
    <owl:Thing rdf:about="#homer"/>
    <owl:Thing rdf:about="#lisa"/>
    <owl:Thing rdf:about="#maggie"/>
    <owl:Thing rdf:about="#bart"/>
  </owl:oneOf>
</owl:Class>
```

```
Class: simpsonFamily
EquivalentTo: { marge, homer, lisa, maggie, bart }
```

```
:simpsonFamily owl:equivalentClass [
  rdf:type owl:Class ;
  owl:oneOf ( :marge, :homer, :lisa, :maggie, ;bart)
] .
```



Recap

- OWL preliminaries
- OWL class constructors
- <https://www.w3.org/TR/owl2-primer/>

COMP318: Introduction to OWL

www.csc.liv.ac.uk/~valli/Comp318



Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

V.Tamma@liverpool.ac.uk

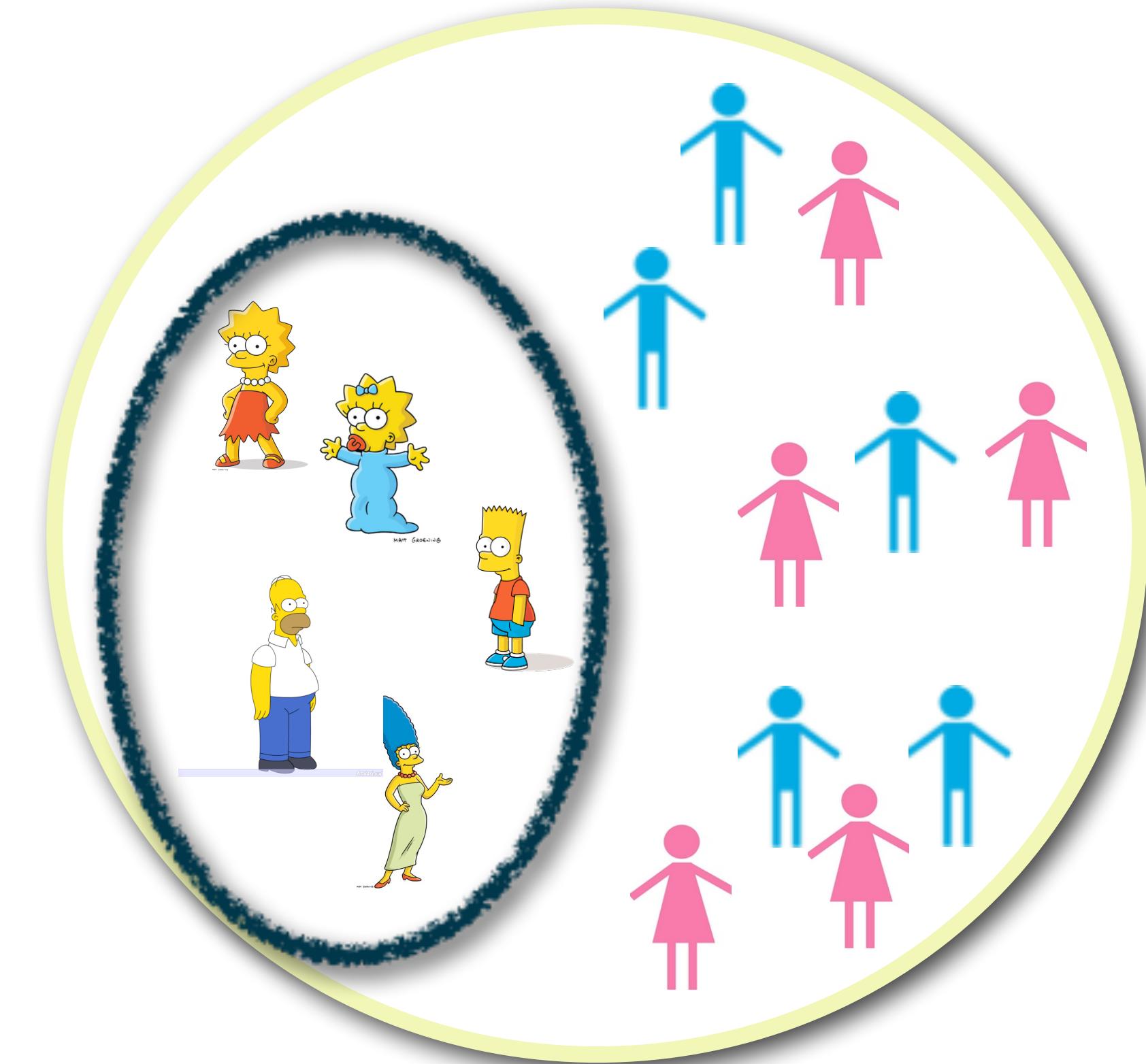
Where were we

- OWL preliminaries
- OWL class constructors
- <https://www.w3.org/TR/owl2-primer/>

OWL class constructors

- Classes can also be defined through enumeration using `owl:oneOf`
 - allows a class to be defined extensionally,
 - with exactly the enumerated individual

```
<owl:Class rdf:about="#simpsonFamily">
  <owl:oneOf rdf:type="Collection">
    <owl:Thing rdf:about="#marge"/>
    <owl:Thing rdf:about="#homer"/>
    <owl:Thing rdf:about="#lisa"/>
    <owl:Thing rdf:about="#maggie"/>
    <owl:Thing rdf:about="#bart"/>
  </owl:oneOf>
</owl:Class>
```



```
:simpsonFamily owl:equivalentClass [
  rdf:type owl:Class ;
  owl:oneOf ( :marge, :homer, :lisa, :maggie, ;bart)
].
```

```
Class: simpsonFamily
EquivalentTo: { marge, homer, lisa, maggie, bart }
```

OWL properties

- As in RDFS, there are two types of properties in OWL datatypes and object properties.
- datatype properties relate objects to datatype values:
 - name, phoneNumber, age...
- OWL does not have any predefined datatypes
 - but it allows users to use XML Schema data types
 - &xsd;nonNegativeInteger is an abbreviation for “<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>”

```
<owl:DatatypeProperty rdf:ID="age">
  <rdfs:range rdf:resource=""
    &xsd;nonNegativeInteger"/>
</owl:DatatypeProperty>
```

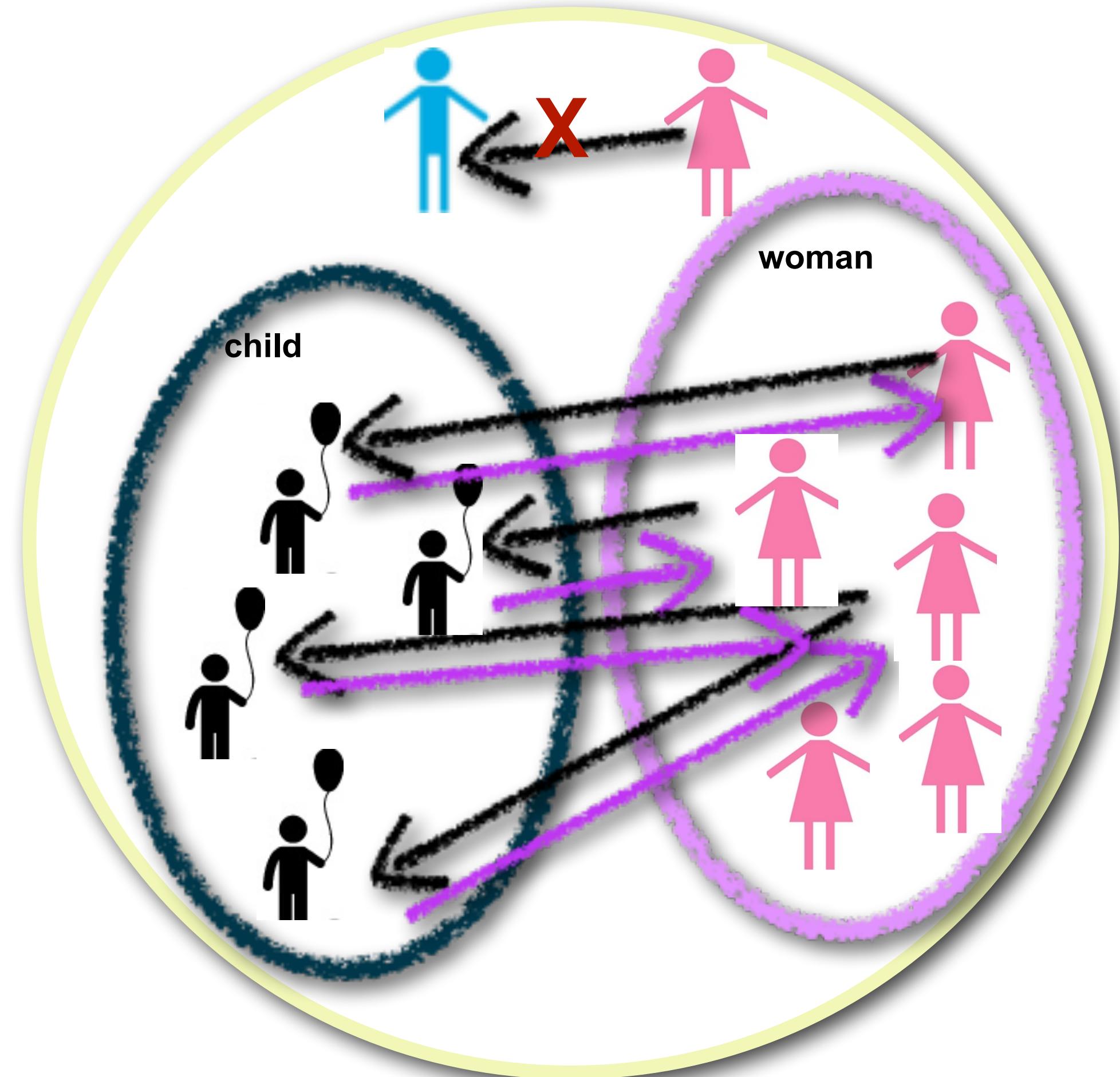
OWL properties

- **Object** properties.
 - relate objects to other objects
 - *marriedTo, father, spouse...*

```
<owl:ObjectProperty rdf:about="#motherOf">
  <rdfs:domain rdf:resource="#woman"/>
  <rdfs:range rdf:resource="#person"/>
  <rdfs:subPropertyOf rdf:resource="#parentOf"/>
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:about="#childOf">
  <rdfs:domain rdf:resource="#person"/>
  <rdfs:range rdf:resource="#woman"/>
  <owl:inverseOf rdf:resource="#motherOf"/>
</owl:ObjectProperty>
```

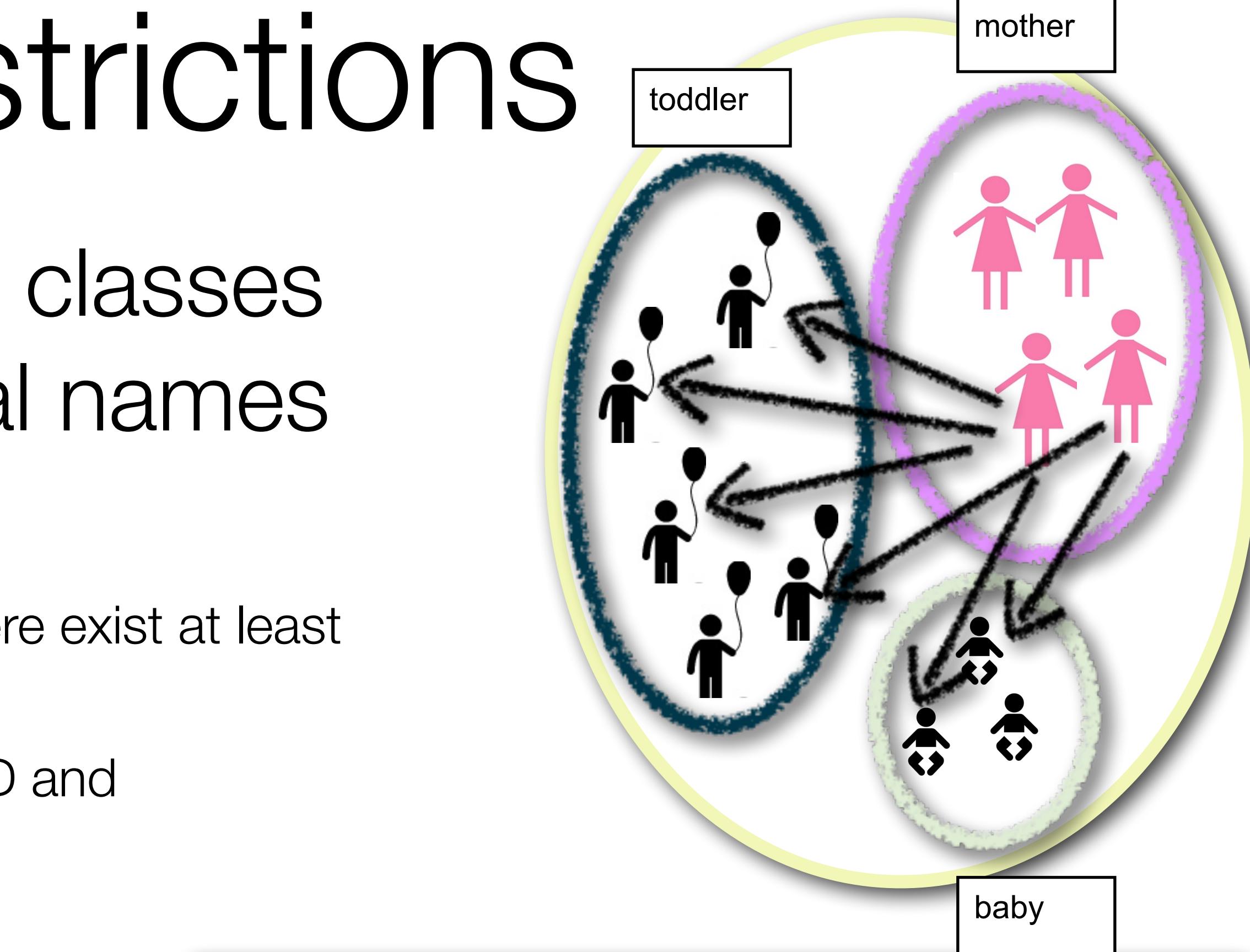
```
<owl:ObjectProperty rdf:about="#offspringOf">
  <owl:equivalentProperty rdf:resource="#childOf"/>
</owl:ObjectProperty>
```



Property restrictions

- Restrictions allow us to build new classes from class, property and individual names
 - existential quantification:
 - define a class that consists of all objects for which there exist at least **one** toddler among the values of motherOf
 - the restriction defines an **anonymous** class with no ID and only local scope - it can only be used in the place the restriction appears

```
<owl:Class rdf:about="#motherOfToddler">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:Property rdf:resource="#motherof"/>
      <owl:someValuesFrom rdf:resource="#toddler"/>
    </owl:Restriction>
  </<rdfs:subClassOf>
</owl:Class>
```



```
:motherOfToddler rdf:type owl:Class ;
  rdfs:subClassOf [
    rdf:type owl:Restriction ;
    owl:onProperty :motherOf ;
    owl:someValuesFrom :toddler .]
```

Class: motherOfToddler SubClassOf: motherOf [some toddler]

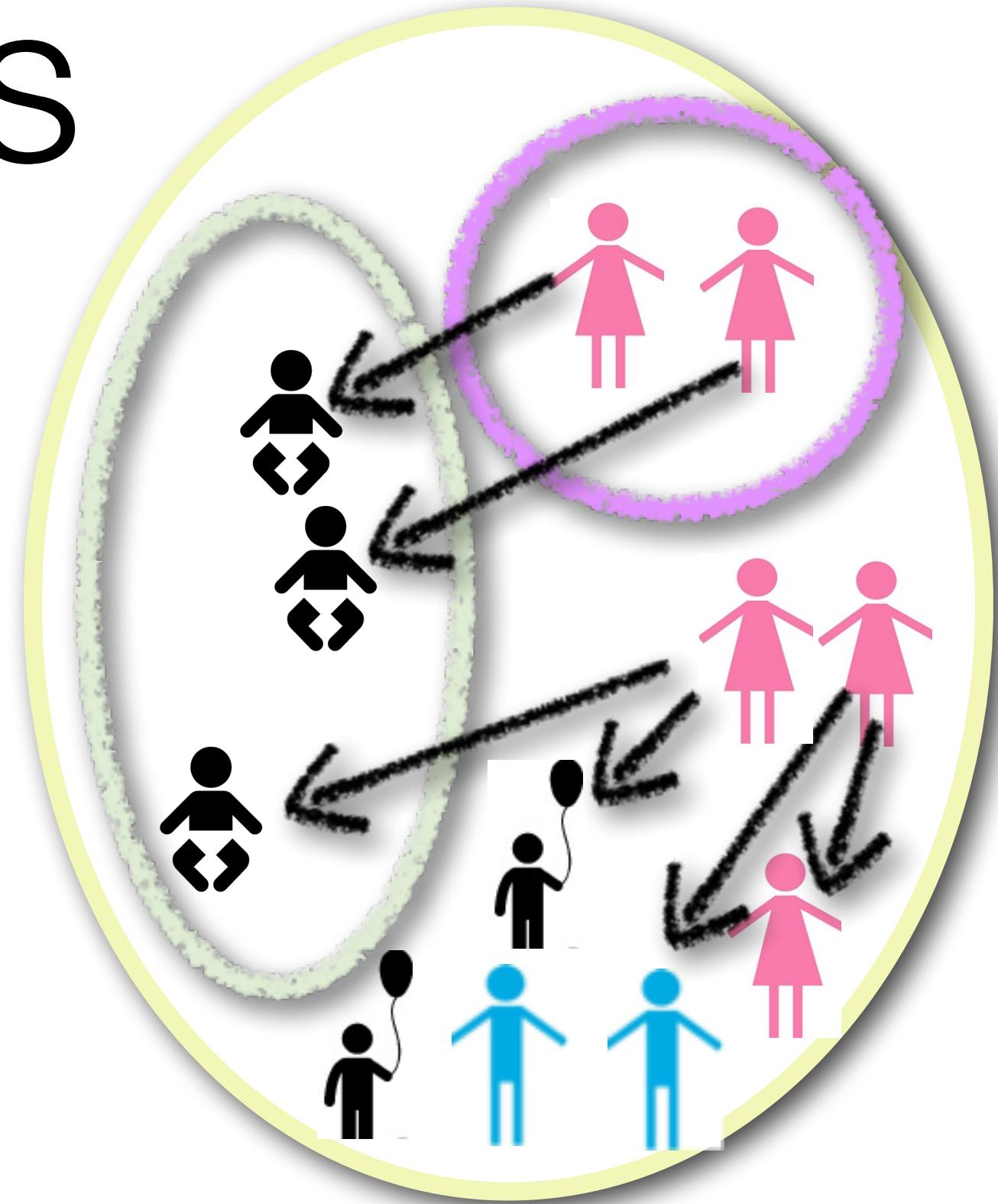
Property restrictions

- Restrictions allow us to build new classes from class, property and individual names
- universal quantification:
 - define a class that consists of all objects for which **all** values of motherOf are babies

```
<owl:Class rdf:about="#motherOfBaby">
  <owl:subClassOf>
    <owl:Restriction>
      <owl:Property rdf:resource="#motherOf"/>
      <owl:allValuesFrom rdf:resource="#baby"/>
    </owl:Restriction>
  </owl:subClassOf>
</owl:Class>
```

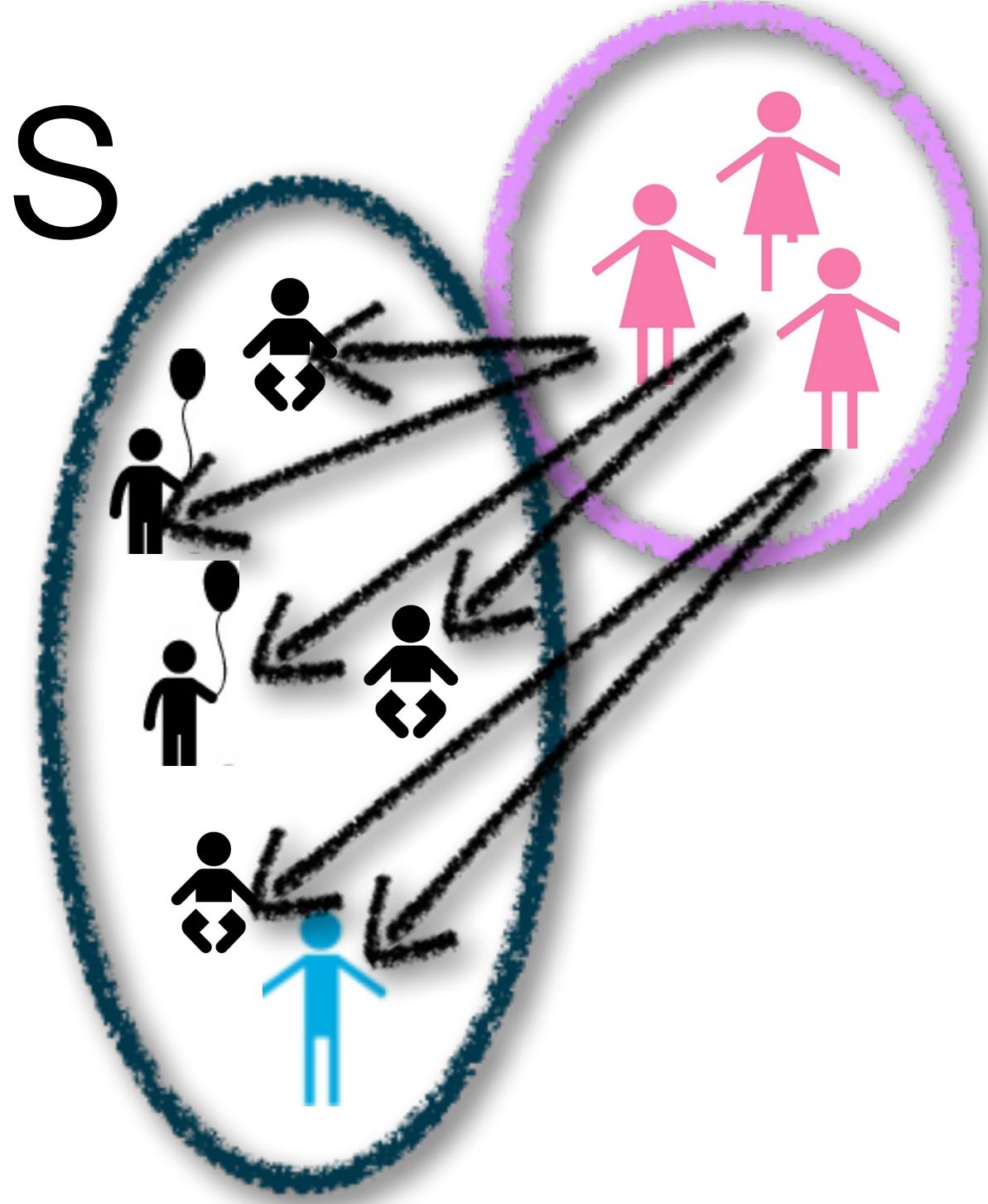
```
:motherOfBaby rdf:type owl:Class ;
  rdfs:subClassOf [
    rdf:type owl:Restriction ;
    owl:onProperty :motherOf ;
    owl:allValuesFrom :Baby . ]
```

Class: motherOfBaby SubClassOf: motherOf [all baby]



Cardinality restrictions

- Restrictions allow us to build new classes from class, property and individual names
- *min*, *max* and *exactly* cardinality restrictions



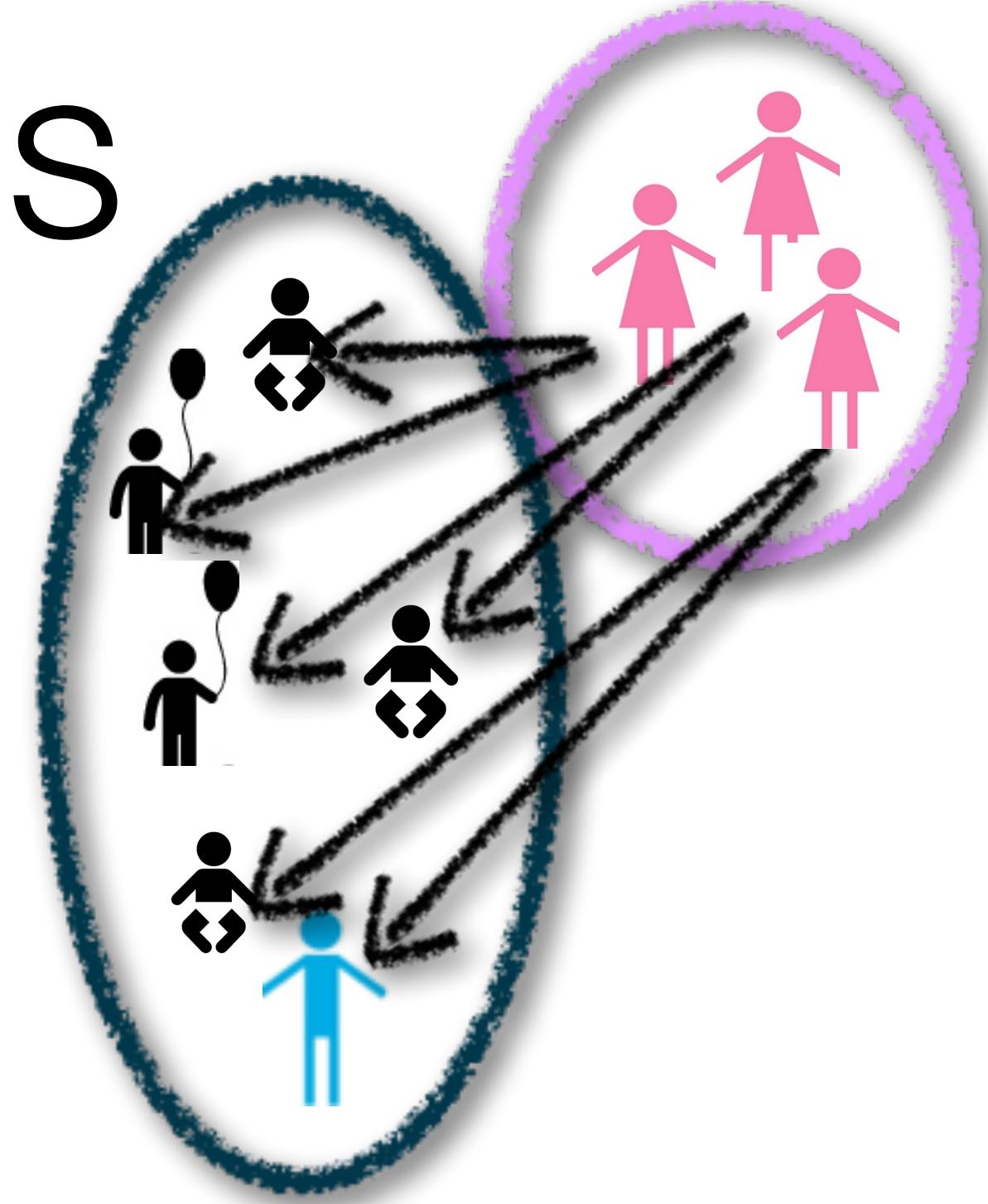
```
<owl:Class rdf:about="#motherOfChildren">
  <owl:subClassOf>
    <owl:Restriction>
      <owl:Property rdf:resource="#motherOf"/>
      <owl:minQualifiedCardinality
        rdf:datatype="&xsd;nonNegativeInteger"/> 2
      </owl:minQualifiedCardinality >
      <owl:Class rdf:resource="#offspring"/>
    </owl:Restriction>
  </owl:subClassOf>
</owl:Class>
```

Class: motherOfChildren SubClassOf: motherOf min 2 Offspring

```
:motherOfChildren rdf:type owl:Class ;
  rdfs:subClassOf [
    rdf:type owl:Class ;
    rdf:type owl:Restriction ;
    owl:minQualifiedCardinality "2"^^&xsd:nonNegativeInteger ;
    owl:onProperty :motherOf ;
    owl:onClass :Offspring . ]
```

Cardinality restrictions

- Restrictions allow us to build new classes from class, property and individual names
- *min*, *max* and *exactly* cardinality restrictions



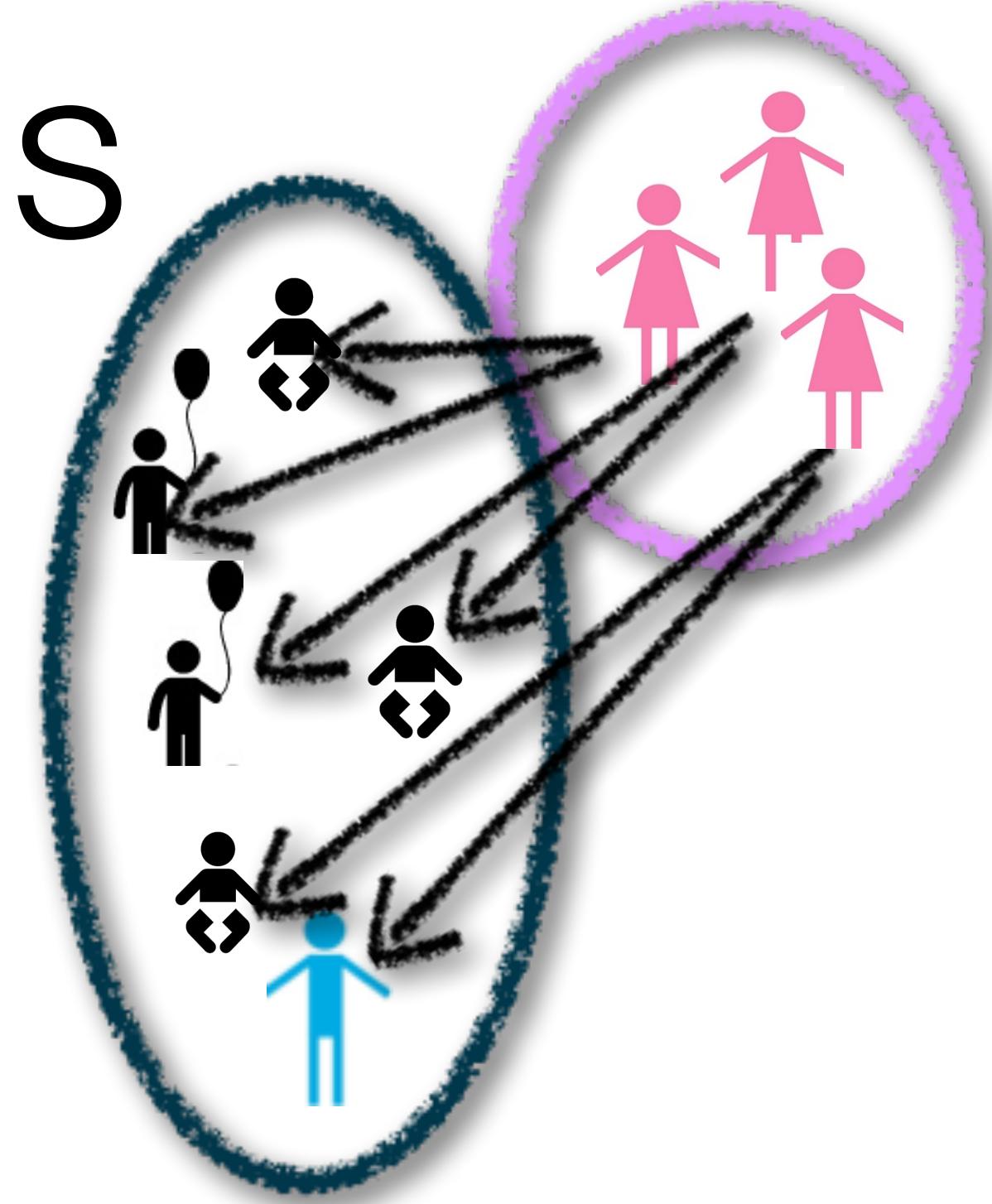
```
<owl:Class rdf:about="#motherOfChildren">
  <owl:subClassOf>
    <owl:Restriction>
      <owl:Property rdf:resource="#motherOf"/>
      <owl:maxQualifiedCardinality
        rdf:datatype="&xsd;nonNegativeInteger"/> 4
      </owl:maxQualifiedCardinality >
      <owl:Class rdf:resource="#offspring"/>
    </owl:Restriction>
  </owl:subClassOf>
</owl:Class>
```

Class: motherOfChildren SubClassOf: motherOf max 4 Offspring

```
:motherOfChildren rdf:type owl:Class ;
  rdfs:subClassOf [
    rdf:type owl:Restriction ;
    owl:maxQualifiedCardinality "4"^^&xsd:nonNegativeInteger;
    owl:onProperty :motherOf ;
    owl:onClass :Offspring . ]
```

Cardinality restrictions

- Restrictions allow us to build new classes from class, property and individual names
- *min*, *max* and *exactly* cardinality restrictions



```
<owl:Class rdf:about="#motherOfTwo">
  <owl:subClassOf>
    <owl:Restriction>
      <owl:Property rdf:resource="#motherOf"/>
      <b><owl:qualifiedCardinality>
        <rdf:datatype="&xsd;nonNegativeInteger"/> 2
      </owl:qualifiedCardinality>>
      <owl:Class rdf:resource="#offspring"/>
    </owl:Restriction>
  </owl:subClassOf>
</owl:Class>
```

```
Class: motherOfChildren SubClassOf: motherOf exactly 2 Offspring
```

```
:motherOfChildren rdf:type owl:Class ;
  rdfs:subClassOf [
    rdf:type owl:Class ;
    owl:intersectionOf
      ( [ rdf:type owl:Restriction ;
        owl:Cardinality "2"^^&xsd:nonNegativeInteger ;
        owl:onProperty :motherOf . ] )
```

hasValue restriction

- Restrictions allow us to build new classes from class, property and individual names
 - Simpsons children are children of Marge

```
<owl:Class rdf:about="#motherOfSimpsons">
  <owl:subClassOf>
    <owl:Restriction>
      <owl:Property rdf:resource="#offspringOf"/>
      <owl:hasValue rdf:resource="#marge"/>
    </owl:Restriction>
  </owl:subClassOf>
</owl:Class>
```

```
:motherOfSimpsons rdf:type owl:Class ;
  owl:EquivalentClass [
    rdf:type owl:Restriction ;
    owl:onProperty :offspringOf ;
    owl:hasValue :Marge
```

Class: motherOfSimpsons SubClassOf: offspringOf value Marge



OWL 1.0: characterising properties

- We can explicitly state the characteristics of object properties, and use these characteristics to refine reasoning:
 - owl:TransitiveProperty;
 - owl:SymmetricProperty;
 - owl:InverseOf;
 - owl:FunctionalProperty and owl:InverseFunctionalProperty.

OWL 1.0: characterising properties

- OWL:TransitiveProperty

- for all x, y, z if $R(x, y)$ and $R(y, z)$ then $R(x, z)$
 - isTallerThan, hasSameGradeAs, isSiblingOf, ...

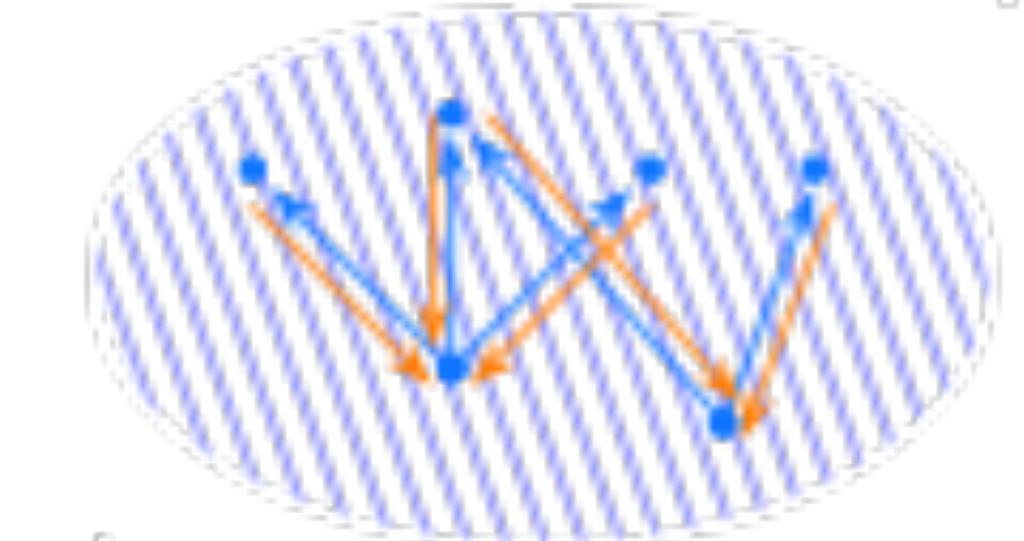
- OWL:SymmetricProperty

- for all x, y if $R(x, y)$ then $R(y, x)$
 - isSiblingOf, hasSameGradeAs, isFriendOf ...

- OWL:InverseProperty

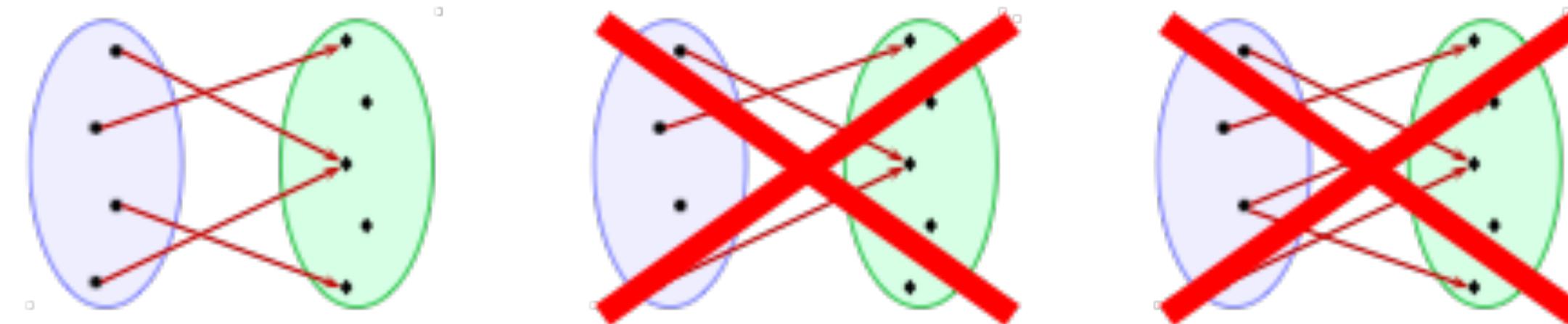
- for all x, y if $R(x, y)$ then $R(y, x) \equiv R^-(x, y)$
- hasParent
 - isParentOf

property R and its inverse R^-



Functions

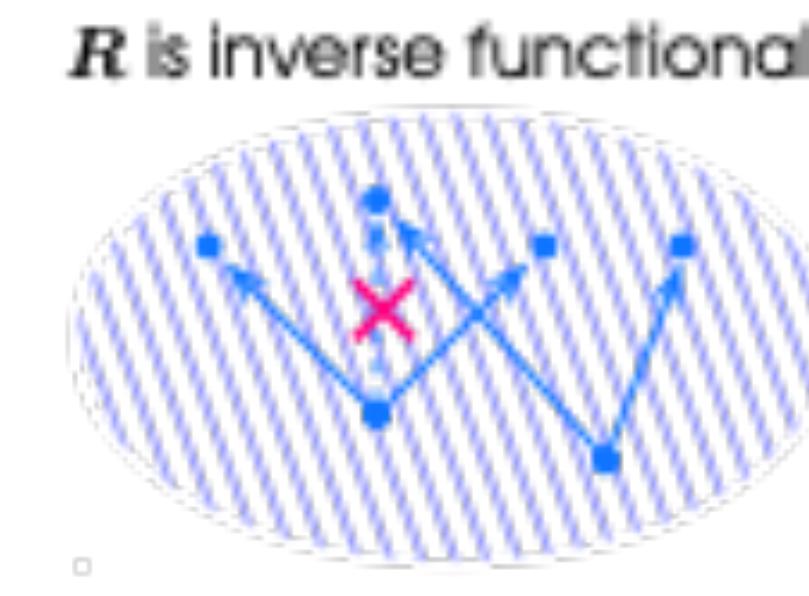
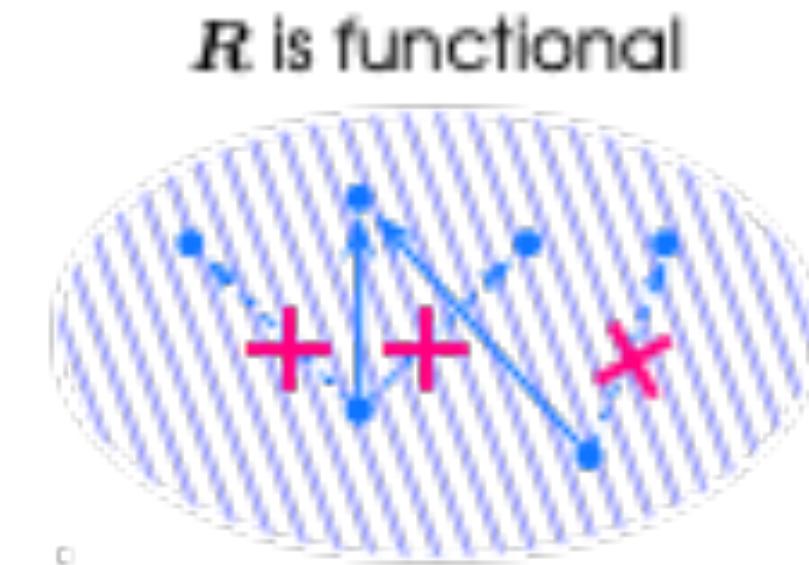
- A function from a set A to a set B is a binary relation $R \subseteq A \times B$ in which every element of A is R -related to a unique element of B
- in other words for each $a \in A$, there is precisely one pair (a, b) in R



OWL 1.0: characterising properties

- OWL:FunctionalProperty

- for every x there is at most one y with $R(x, y)$
 - at most one value is associated to each object
 - directSupervisor



- OWL:inverseFunctionalProperty

- for every y there is at most one x with $R(x, y)$
- two different objects cannot have the same value associated to them
 - hasStudentNumber
 - for each StudentNumber, there can only be one student associated to that number.

Example

- Translate in turtle syntax the following statements, and add any axiom you think appropriate:
 - john is a lecturer
 - mary is an academic staff member
 - mary is 39 years old
 - COMP1111 is a course
 - each course is taught by at most one staff member
 - john teaches COMP1111
 - mary teaches COMP1111

Example (ctd)

- Is the model we obtain correct, or does it contain contradictory information?
 - If so, what are the statements that cause a contradiction?
 - how would you solve it?

Example

- Translate in turtle syntax the following statements:
 - first year courses are courses taught only by professors

Recap

- OWL class constructors
- OWL properties
- Restrictions
- <https://www.w3.org/TR/owl2-primer/>

COMP318: OWL

www.csc.liv.ac.uk/~valli/Comp318



Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

v.Tamma@liverpool.ac.uk

Recap

- OWL ontology language
- OWL class constructors
- OWL property restrictions

Three species of OWL 1.0

- OWL Lite
 - Classification hierarchy
 - Simple constraints
- OWL DL
 - Maximal expressiveness while maintaining decidability
 - Standard formalisation in a DL
- OWL Full
 - Very high expressiveness
 - Losing decidability
 - All syntactic freedom of RDF (self-modifying)

Comp 318

OWL Lite vs OWL DL vs OWL Full

- OWL Lite
 - (sub)classes, individuals
 - (sub)properties, domain, range
 - conjunction
 - (in)equality
 - cardinality 0/1
 - datatypes
 - inverse, transitive, symmetric properties
 - **someValuesFrom**
- **allValuesFrom**
- OWL DL
 - Negation
 - Disjunction
 - Full cardinality
 - Enumerated types
 - **hasValue**
- OWL Full
 - Meta-classes
 - Modify language

OWL 2 profiles

- Sublanguages of OWL2 trading expressive power for efficient reasoning
 - Each supports different application scenarios
- OWL 2 EL
 - very large ontologies, efficient reasoning performance guaranteed at the expenses of expressive power;
- OWL 2 RL
 - subclass axioms understood as rule like implication, with head - superclass and body - subclass
- different restrictions on subclasses and superclasses
 - allows the integration of OWL with rules
- OWL 2 QL
 - useful to query data rich applications
 - different restrictions on subclasses and superclasses
 - suitable for simple, lightweight ontologies with a large number of individuals and it is necessary to access the data directly via SQL queries
 - fast implementation on top of legacy DB systems, relational or RDF

Example

- Translate in turtle syntax the following statements, and add any axiom you think appropriate:
 - john is a lecturer
 - mary is an academic staff member
 - mary is 39 years old
 - COMP1111 is a course
 - each course is taught by at most one staff member
 - john teaches COMP1111
 - mary teaches COMP1111

Example 1 in Manchester syntax

```
ObjectProperty: TaughtBy
    Characteristics: Functional
    Domain: Course
    Range: AcademicStaffMember

DataProperty: age
    Range: xsd:nonNegativeInteger

Class: AcademicStaffMember
    SubClassOf: Person

Class: Course
    DisjointWith: Person

Class: Lecturer
    SubClassOf: AcademicStaffMember

Individual: comp111
    Types: Course
    Facts: isTaughtBy john
           isTaughtBy mary

Individual: john
    Types:Lecturer
    DifferentFrom: mary
```

```
Individual: mary
    Types: AcademicStaffMember
    Facts: age "39"^^xsd:nonNegativeInteger
    DifferentFrom: john
```

Example (ctd)

- Is the model we obtain correct, or does it contain contradictory information?
 - If so, what are the statements that cause a contradiction?
 - how would you solve it?

Example in Manchester Syntax

ObjectProperty: TaughtBy

~~Characteristics: Functional~~

Domain: Course

Range: AcademicStaffMember

DataProperty: age

Range: xsd:nonNegativeInteger

Class: AcademicStaffMember

SubClassOf: Person

Class: Course

DisjointWith: Person

Class: Lecturer

SubClassOf: AcademicStaffMember

Individual: comp1111

Types: Course

Facts: ~~isTaughtBy john~~

isTaughtBy mary

Individual: john

Types: Lecturer

DifferentFrom: mary

Individual: mary

Types: AcademicStaffMember

Facts: age

"39"^^xsd:nonNegativeInteger

DifferentFrom: john

Example

- Translate in turtle syntax the following statements:
 - first year courses are courses taught only by professors
 - maths courses are taught by mary
 - all academic staff members must teach at least one undergraduate course
 - an undergraduate course is taught by someone

Example

```
FirstYearCourse
    rdf:type owl:Class
    rdfs:subClassOf [ rdf:type owl:Restriction;
                      owl:onProperty :isTaughtBy;
                      owl:AllValuesFrom
                        :Professors ] .
```

Example

- Maths courses are taught by Mary

```
:mathCourse
    rdf:type owl:Class
    rdfs:subClassOf ([ rdf:type owl:Restriction;
                      owl:onProperty :isTaughtBy;
                      owl:hasValue :mary ] ).

:mary rdf:type :Professor .

:mathsCourse rdf:type :FirstYearCourse .
```

Not necessary, should be inferred

Example

- all academic staff members must teach at least one undergraduate course

```
:AcademicStaffMember
  rdf:type owl:Class
  rdfs:subClassOf ([ rdf:type owl:Restriction;
                     owl:onProperty :teaches;
                     owl:someValuesFrom
                     :UndergraduateCourses ] ).
```

Example

- an undergraduate course is taught by someone (Academic staff member)

```
:Course
  rdf:type owl:Class
  rdfs:subClassOf ([ rdf:type owl:Restriction;
    owl:onProperty :isTaughtBy;
    owl:minQualifiedCardinality 1;
    owl:onClass :AcademicStaffMember] ).
```

Example

- an undergraduate course is taught by someone

```
:Course
  rdf:type owl:Class
  rdfs:subClassOf ( [ rdf:type owl:Restriction;
    owl:onProperty :isTaughtBy;
    owl:someValuesFrom
      :AcademicStaffMember] ).
```

Exercise

- a department has at least 10 members and at most 30 members

Exercise

- a department has at least 10 members and at most 30 members

```
:Department
  rdf:type owl:Class
  rdfs:subClassOf [ rdf:type owl:Restriction;
    owl:onProperty :hasMember;
    owl:minQualifiedCardinality 10;
    owl:onClass :Member]
  [ rdf:type owl:Restriction;
    owl:onProperty :hasMember;
    owl:maxQualifiedCardinality 30;
    owl:onClass :Member]
```

Example 3

- Translate in turtle or Manchester syntax the following statements:
 - *no course is an academic staff member*
 - *define peopleAtUni as the union of staffMember and student*
 - *a faculty in CS is a faculty member who works in the CS department*
 - *adminStaff are those department staff members that are neither faculty nor technical support staff*

Example 3 in turtle syntax

```
[ ] rdf:type owl:AllDisjointClasses ;
owl:members ( :Course :AcademicStaffMember ) .

PeopleAtUni owl:equivalentClass [
rdf:type owl:Class ;
owl:unionOf ( :AcademicStaffMember :Student ) ] .

FacultyInCS owl:equivalentClass [
rdf:type owl:Class ;
owl:intersectionOf ( Faculty
[ rdf:type owl:Restriction;
owl:onProperty :belongsTo;
owl:someValuesFrom :CSDepartment] .

AdminStaff owl:equivalentClass [ rdf:type owl:Class ;
owl:intersectionOf ( DepartmentMember
[ rdf:type owl:Class;
owl:complementOf [
rdf:type owl:Class ;
owl:unionOf ( Faculty TeachingSupportStaff ] ] )
```

Exercise

- Translate the following statements in Turtle syntax:
 - The class AcademicStaffMember does not share any element with the class TechnicalStaffMember
 - The class StaffMember includes elements that are in the class AcademicStaffMember or in the class TechnicalStaffMember
 - The role isResponsibleForTask is used to relate elements of the class StaffMember to elements of the class MgmtTasks

Sample solution

1. `rdf:type owl:AllDisjointClasses ;
owl:members (:AcademicStaffMember :TechnicalStaffMember)`
2. `:StaffMember owl:equivalentClass [
rdf:type owl:Class ;
owl:unionOf (:AcademicStaffMember :TechnicalStaffMember)
] .`
3. `:isResponsibleForTask rdfs:domain :StaffMember
rdfs:range :MgmtTasks`

Exercise

- Given the Knowledge Base described before, decide whether the following statements are reasonable and motivate your answer
 - teachesModule is functional
 - teachesModule is inverseFunctional

Sample solution

- `teachesModule` should not be functional
 - an `AcademicStaffMember` can teach more than one module
- `teachesModule` is not `inverseFunctional`,
 - since a specific instance of a `Module` can be taught by two different `AcademicStaffMembers`

COMP318: RDF serialisations

www.csc.liv.ac.uk/~valli/Comp318



Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

V.Tamma@liverpool.ac.uk

Where were we

- RDF:
 - data model for sharing information on the Web
 - syntax
 - triples
- RDFS
 - language for defining the schema representing a domain of interest
 - classes and properties
- RDF vocabulary
 - the constructs we use to represent data and knowledge in RDF/S
- RDFS semantics for inferring new information from the graph
- OWL syntax and modelling primitives

Why serialisation?

- RDF is not a data format, but a **data model** for describing resources in the forms of triples
 - *subject, predicate, object*
- One of the use cases underlying RDF's design is to describe (**mark up**) the content of HTML web pages.
 - RDFa is a syntactic variant introduced to help with that use case
 - RDFa embeds RDF within the attributes of HTML tags

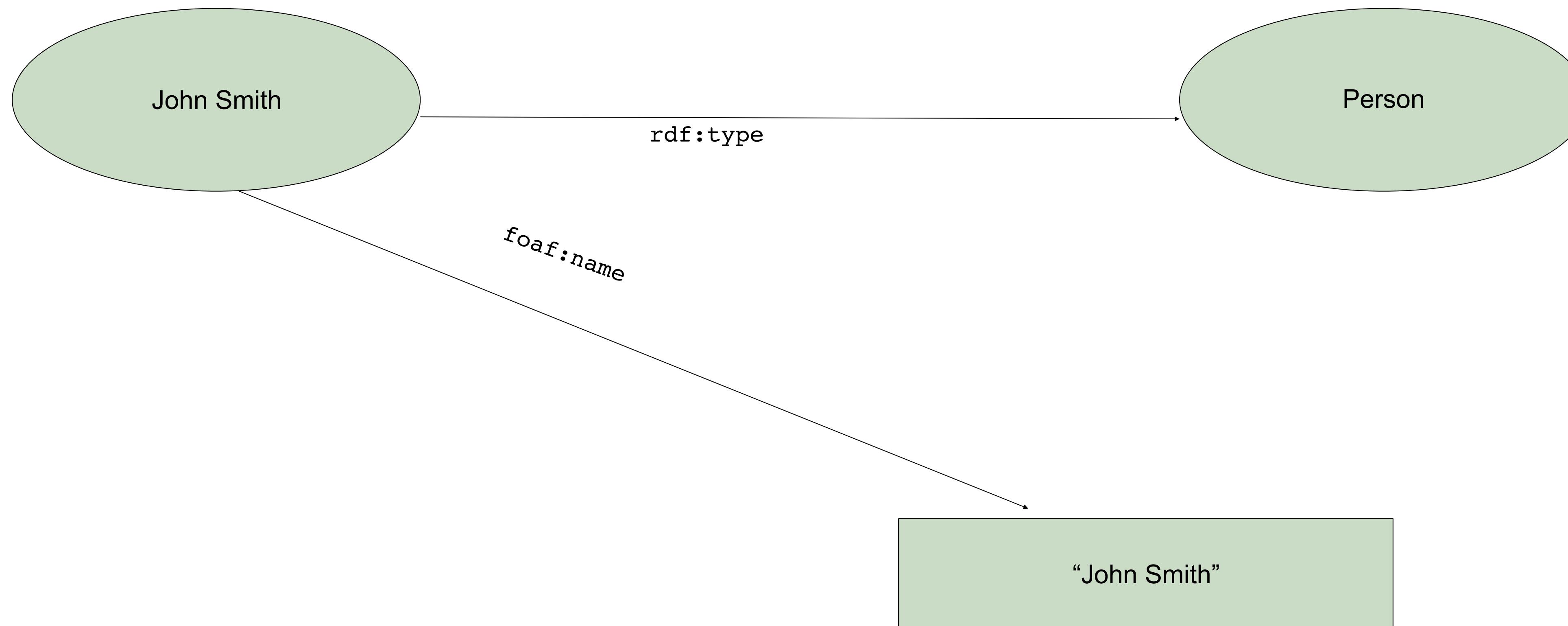
Why serialisation?

- RDF is not a data format, but a **data model** for describing resources in the forms of triples
 - *subject, predicate, object*
- To publish a document on the Web it must be serialised using one of the RDF syntax formats
 - the triples forming the graph representing the RDF document need to be written in a file
 - conforming to the syntactic constraints deriving from the chosen syntax
 - in advance if dealing with static data set
 - on demand if dealing with dynamic data sets

Serialisation formats

- RDF serialisation formalisms
 - RDF/XML
 - RDFa
 - Turtle
 - N-Triples
 - RDF/JSON

A simple example



RDF/XML

- The **MIME** type for the document
 - indicates the Internet media type and indicates the format of files on the Internet
 - for RDF/XML documents is **application/rdf+xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/">

  <rdf:Description rdf:about="http://www.liv.ac.uk/staff/john-smith">
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
      <foaf:name>John Smith</foaf:name>
  </rdf:Description>
</rdf:RDF>
```

RDF/XML explained

- The example shows two triples
 - The resource identified by the URI
`http://www.liv.ac.uk/staff/john-smith` is of type **Person**
 - **Person** is defined in the FOAF vocabulary
 - the URI identifies a document that is a reference to John Smith
 - The **name** of the resource is “John Smith”
 - **name** is defined in the FOAF vocabulary

Turtle

- Turtle is the plain text serialisation format for RDF
 - typically used to edit RDF documents by hand
 - the **MIME** type for Turtle is
text/turtle; charset=utf-8.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@ prefix foaf: <http://xmlns.com/foaf/0.1/> .  
  
<http://www.liv.ac.uk/staff/john-smith>  
    rdf:type foaf:Person;  
    foaf:name "John Smith" .
```

N-Triples

```
<http://www.liv.ac.uk/staff/john-smith>
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://xmlns.com/foaf/0.1/Person> .

<http://www.liv.ac.uk/staff/john-smith>
  <http://xmlns.com/foaf/0.1/name>
  "John Smith" .
```

N-Triples

- N-Triples is a subset of Turtle
 - it does not allow namespace prefixes and shorthand
 - introduces redundancy in the serialisation format
 - allows N-Triples files to be parsed one line at the time
 - no memory constraints over large files
 - N-Triples files can be easily compressed
 - reduction in network traffic when exchanging files
 - currently accepted standard for exchanging large dumps of Linked Data

RDF/JSON

- JSON (JavaScript Object Notation) serialisation for RDF
 - highly desirable as many programming languages provide JSON support
 - triples as nested data structures
 - it represents simple data structures and associative arrays:
 - **Number**
 - **String**
 - **Boolean**
 - **Array**: an ordered sequence of values, comma-separated and enclosed in square brackets; the values do not need to be of the same type.
 - **Object**: an unordered collection of key:value pairs with the ':' character separating the key and the value, comma-separated and enclosed in curly braces;
 - **null**
 - can facilitate uptake of mobile applications consuming semantic data
 - **MIME** content-type is **application/json**

RDF/JSON

```
{  
  "prefixes" : {  
    "rdf" : http://www.w3.org/1999/02/22-rdf-syntax-ns#  
    "foaf" : "http://xmlns.com/foaf/0.1/" ,  
  } ,  
  <http://www.liv.ac.uk/staff/john-smith> : {  
    "rdf:type" : [  
      { "value" : "foaf:Person", "type" : "rdf:Resource" }  
    ] ,  
    "http://xmlns.com/foaf/0.1/name" : [  
      { "value" : "John Smith", "type" : "literal" }  
    ]  
  }  
}
```

The diagram illustrates the mapping of RDF triples to JSON-LD. It shows three main components: **Subject**, **Predicate**, and **Object**. The Subject is represented by a grey box with an arrow pointing to the subject of the first triple. The Object is represented by a grey box with an arrow pointing to the object of the second triple. The Predicate is represented by a grey box with an arrow pointing to the predicate of the first triple.

RDF/JSON

- Example explained:
 - data represented in dictionaries:
 - two elements in a dictionary, with the keys: "**prefixes**" and "**http://www.liv.ac.uk/staff/john_smith**"
 - Their values are both dictionaries

RDFa

- RDFa is a W3C recommendation that adds a set of attribute level extensions to X(HTML)/XML for embedding rich metadata.
 - it supports the notions of namespaces and URIs
 - it allows the mixing of vocabularies as in RDF
 - it offers a flexible framework for using Resources of type URI or Literal
 - it is a complete serialisation of RDF
 - <http://www.w3.org/TR/rdfa-syntax/>

RDFa: embedding in HTML

- RDFa (RDF attribute) allows the embedding of semantic information in existing (X)HTML documents
 - extends (X)HTML a bit by:
 - defining general attributes to add metadata to any elements
 - provides an almost complete “serialisation” of RDF in XHTM
 - designed to enrich existing pages that have already limited semantics based on hyperlinks and tag layout
 - `div` and `span`
 - the RDF data is embedded within the HTML DOM
 - existing content can be annotated with RDFa by simply modifying the HTML document

Main principles of RDFa

- RDFa means “RDF in attributes”:
 - all RDF contents are defined through XML **attributes**
 - no elements
 - it uses the XML/HTML tree structure
 - many of the attributes are defined by RDFa
- some attributes (`@href`, `@rel`) are also reused
- `@href` and `@rel` represent objects that are URI references
- the text content (`@content`) is also reused for literals as well as `@href` values

RDFa

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML+RDFa 1.0//EN"
  "http://www.w3.org/MarkUp/DTD/xhtml-rdfa-1.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/">

<head>
  <meta http-equiv="Content-Type" content="application/xhtml+xml;
    charset=UTF-8" />
  <title>John Smith's personal page</title>
</head>

<body>
  <div about="http://liv.ac.uk/staff#john-smith" typeof="foaf:Person">
    <span property="foaf:name">John Smith</span>
  </div>
</body>

</html>
```

Main principles of RDFa

**RDFa is a serialization of RDF embedded in XHTML,
HTML, or XML in general**

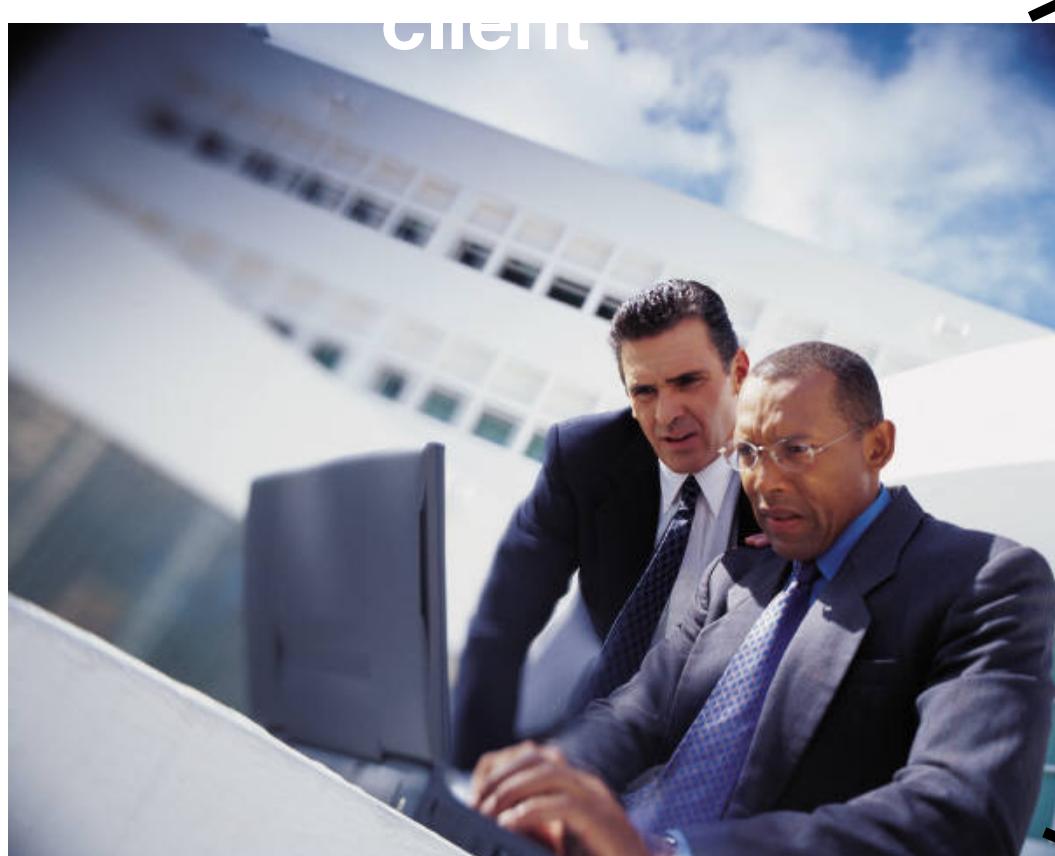
- Most of the data on the web are in (X)HTML:
 - new content generated every day
 - how do we get structured data from that info?
- Especially when authors of the “traditional web” don’t like to generate RDF/XML files separately
 - RDF/XML is complex
 - it requires a separate storage, generation, etc. mechanism
 - that is also valid for, e.g., Turtle
 - but even when authoring with a text editor, creating an extra file is a load

What does this mean in practice?

- The same (X)HTML file:
 - is used, unchanged, by browsers
 - they ignore attributes they do not know
 - can be used by specialised processors (or APIs) to extract RDF triples

Typical usage pattern

Request for
<http://www.w3.org/ns/entailment/data/RDFS>



Request for
<http://www.w3.org/ns/entailment/data/RDFS.ttl>

http://www.w3.org/ns/entailment/data/RDFS.html

W3C Semantic Web

Unique identifier for *RDFS Entailment*.

“<http://www.w3.org/ns/entailment/RDFS>” is the URI. The [specification for the RDFS entailment](#) is part of the [RDF Semantics](#) W3C Recommendation.

For more information about RDF, please refer to the [the RDF Concepts and Abstract Syntax Recommendation](#).

Ivan Herman, ivan@w3.org, W3C, Semantic Web Activity Lead, 2009-05-03

http://www.w3.org/ns/entailment/data/RDFS.ttl

```
ent:RDFS a ent:Entailment ;
  dc:creator <http://www.ivan-herman.net/foaf#me> ;
  dc:date "2009-05-03" ;
  dc:description "Unique identifier for RDFS Entailment" ;
  rdfs:comment "The specification for the RDFS entailment is part of the RDF Semantics W3C Recommendation." ;
  rdfs:isDefinedBy <http://www.w3.org/TR/2004/REC-rdf-mt-20040210/#rdfs\_entailment> ;
  rdfs:seeAlso <http://www.w3.org/TR/2004/REC-rdf-mt-20040210/> .

<http://www.w3.org/ns/entailment/data/RDFS.html> dc:title "Information Resource RDFS Entailment" ;
  xv:stylesheet <http://www.w3.org/StyleSheets/TR/base> .
```

RDFa

- adds new (X)HTML/XML attributes
- has namespaces and URI-s at its core; i.e., mixing vocabulary is just as easy as in RDF
- complete flexibility for using Literals or URI Resources
 - is a complete serialization of RDF
- RDFa is a bridge between the Web of Documents and the Web of Data

Where does the Turtle content come from?

- The triples are embedded in the HTML file
 - a client may know how to extract RDF triples directly from that file; or
 - an online “distiller” service is used; or
 - the server is set up to generate the Turtle file automatically
- However... the content is created only once!

Recap

- Serialisation:
 - RDF document published in a chosen syntax
 - XML/RDF, Turtle and N-triples, **RDFa**, RDF/JSON

COMP318: RDF serialisations

www.csc.liv.ac.uk/~valli/Comp318



Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

V.Tamma@liverpool.ac.uk

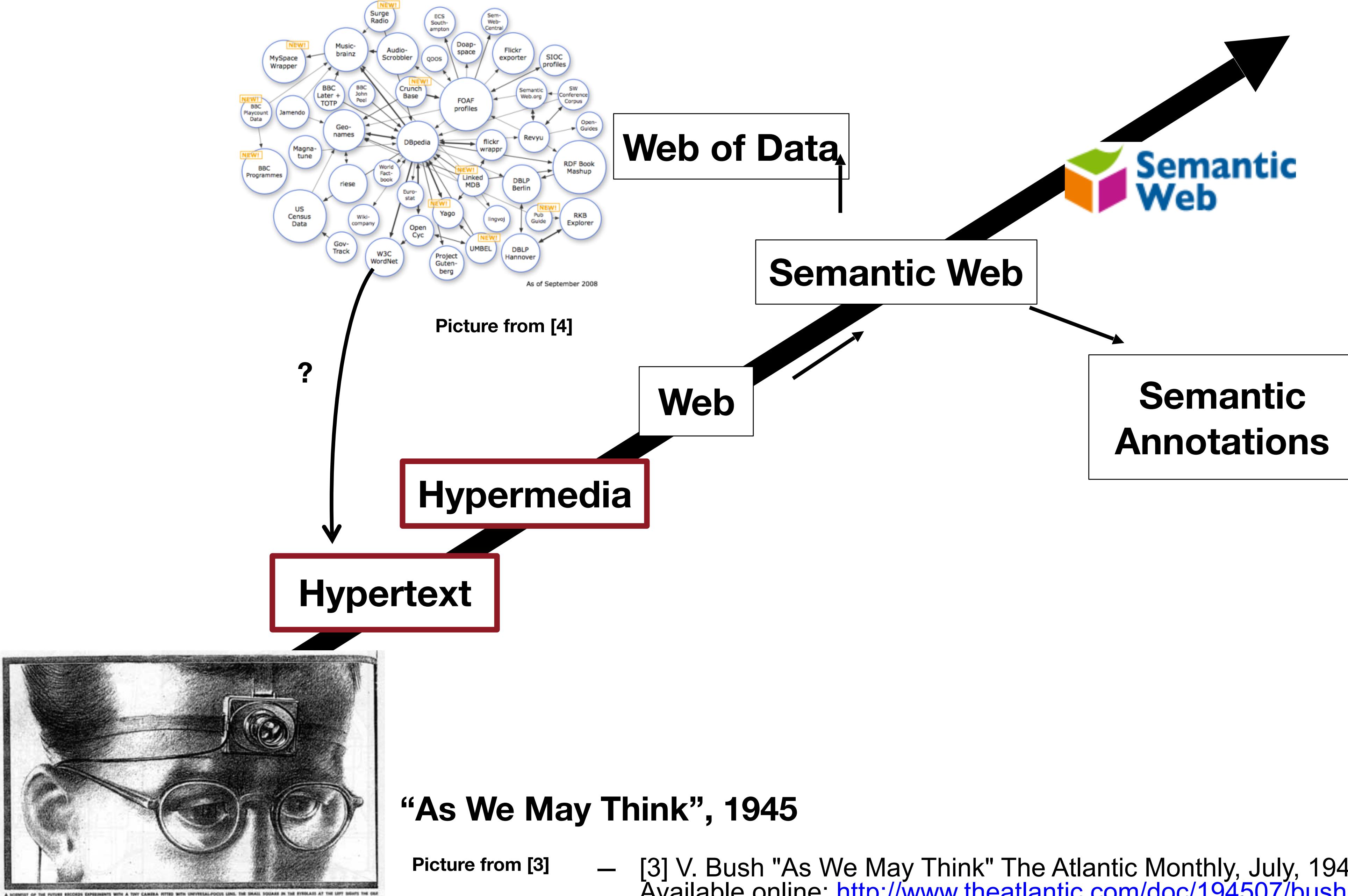
Where were we

- Serialization:
 - RDF document published in a chosen syntax
 - XML/RDF, Turtle and N-triples, **RDFa**, RDF/JSON
- Introduction to RDFa

Main principles of RDFa

**RDFa is a serialization of RDF embedded in XHTML,
HTML, or XML in general**

- Most of the data on the web are in (X)HTML:
 - new content generated every day
 - how do we get structured data from that info?
- Especially when authors of the “traditional web” don’t like to generate RDF/XML files separately
 - RDF/XML is complex
 - it requires a separate storage, generation, etc. mechanism
 - that is also valid for, e.g., Turtle
 - but even when authoring with a text editor, creating an extra file is a load



Microformats

- An approach to add meaning to HTML elements and to make data structures in HTML pages explicit.
- “*Designed for **humans first** and **machines second**, microformats are a set of simple, open data formats built upon existing and widely adopted standards. Instead of throwing away what works today, microformats intend to solve simpler problems first by adapting to current behaviours and usage patterns (e.g. XHTML, blogging).*”

Microformats

- Are highly correlated with semantic (X)HTML / “Real world semantics” / “Lowercase Semantic Web”
- Real world semantics (or the Lowercase Semantic Web) is based on three notions:
 - Adding of simple semantics with microformats (small pieces)
 - Adding semantics to the today’s Web instead of creating a new one (evolutionary not revolutionary)
- Design for humans first and machines second (user centric design)
- A way to combine human with machine-readable information.
- Provide means to embed structured data in HTML pages.
- Build upon existing standards.

Microformats

- Solve a single, specific problem (e.g. representation of geographical information, calendaring information, etc.).
- Provide an “API” for your website.
- Build on existing (X)HTML and reuse existing elements.
- Work in current browsers.
- Follow the DRY principle (“Don’t Repeat Yourself”).
- Compatible with the idea of the Web as a single information space.

Information Resource RDFS Entailment

http://www.w3.org/ns/entailment/data/RDFS.html iswc 2010

Hirek ▾ Social ▾ Private ▾ Mailing lists ▾ SW ▾ Python ▾ RDFa it! ▾ Bookmarklets ▾ To Mendeley To Faviki bit.ly Shorten with bit.ly MID Save Video

W3C Semantic Web

Unique identifier for *RDFS Entailment*.

“<http://www.w3.org/ns/entailment/RDFS>” is the URI. The [specification for the RDFS entailment](#) is part of the [RDF Semantics](#) W3C Recommendation.

For more information about RDF, please refer to the [the RDF Concepts and Abstract Syntax Recommendation](#).

Ivan Herman, ivan@w3.org, W3C, Semantic Web Activity Lead, 2009-05-03

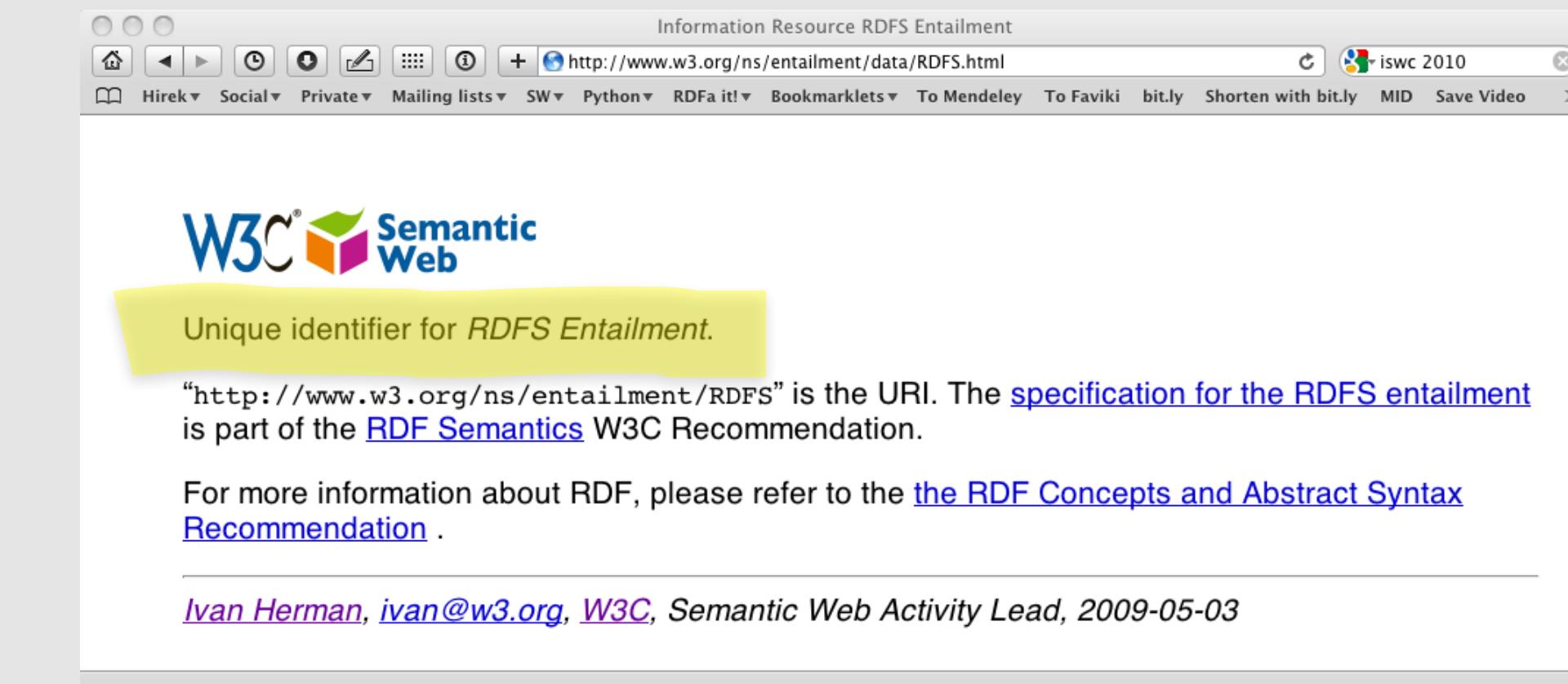
The source and generated RDF...

```
<p about="http://www.w3.org/ns/entailment/RDFS"  
    property="http://purl.org/dc/terms/description">
```

Unique identifier for

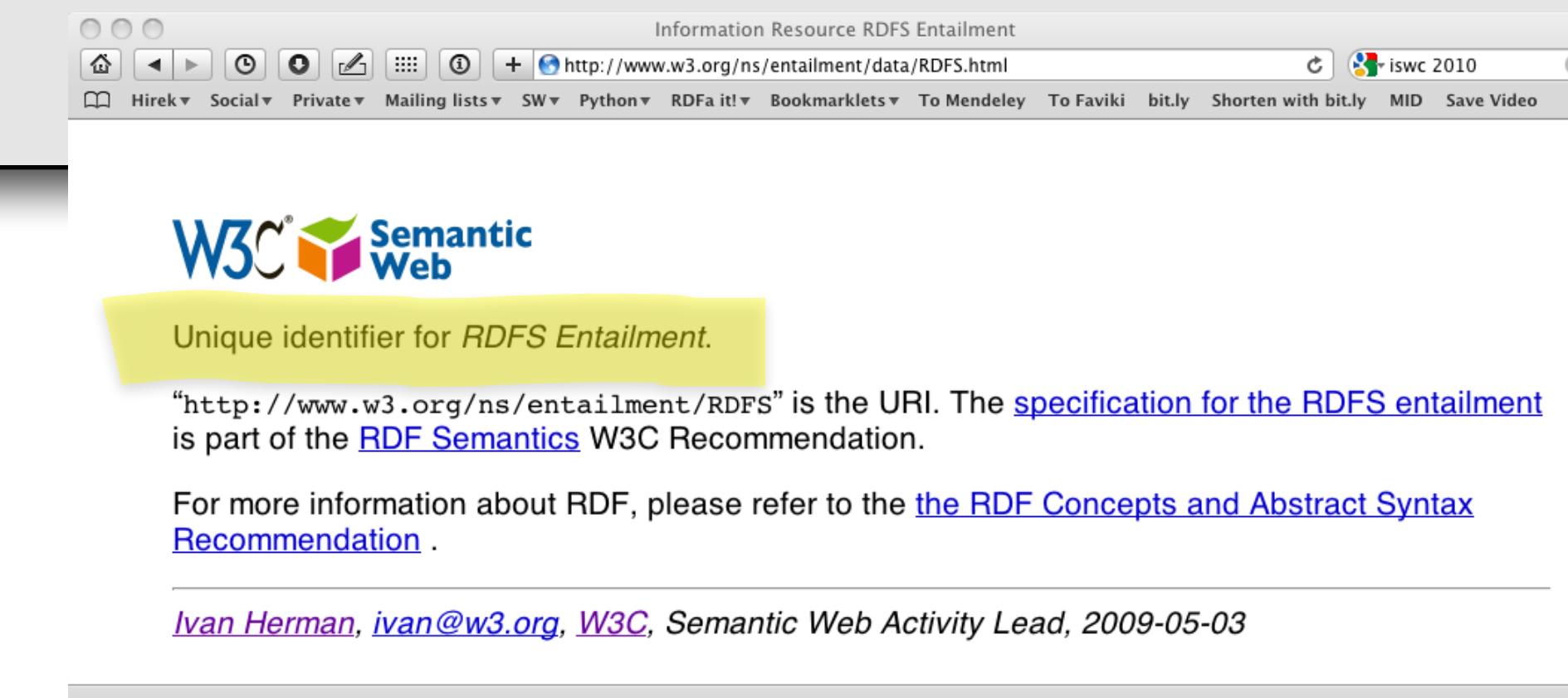
RDFS Entailment.

</p>



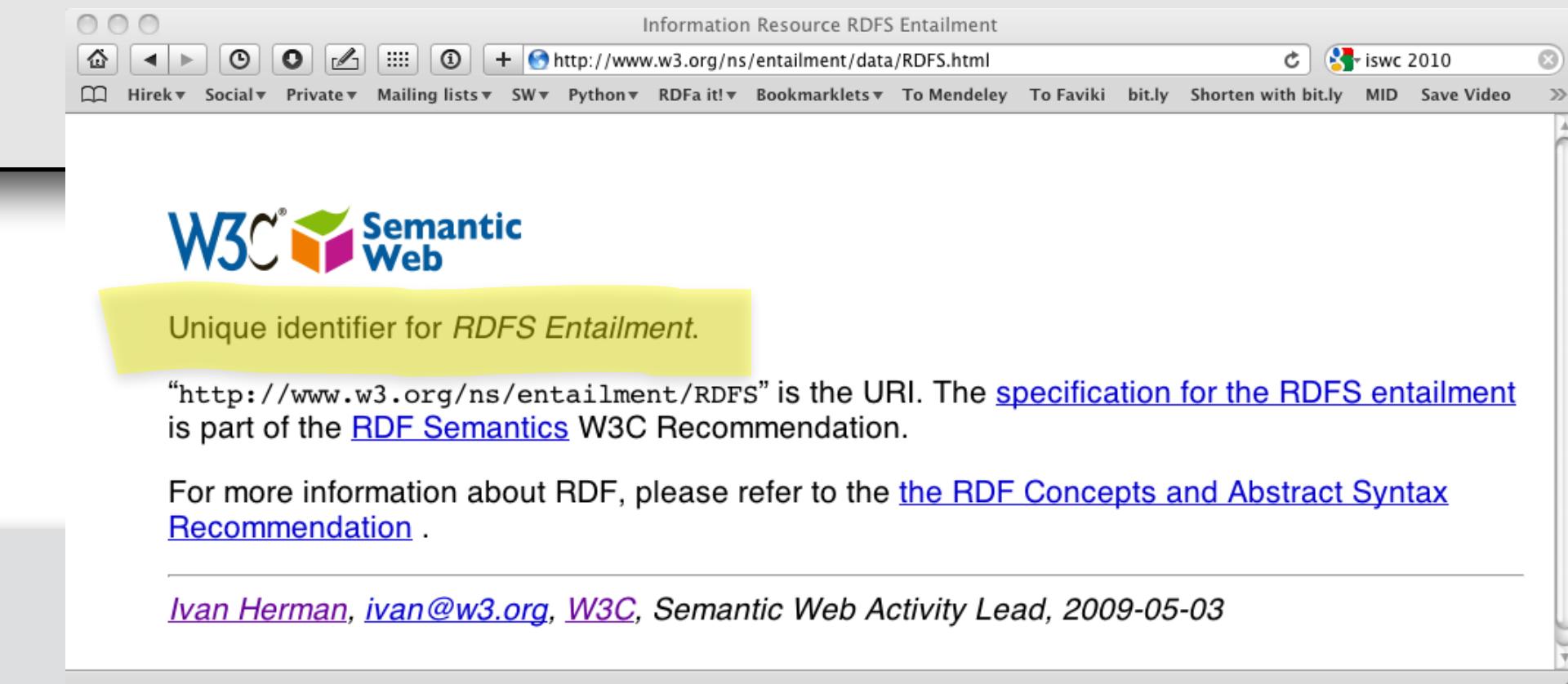
The source and generated RDF...

```
<p about="http://www.w3.org/ns/entailment/RDFS"
    property="http://purl.org/dc/terms/description">
    Unique identifier for
    <em>RDFS Entailment</em>.
</p>
```



The source and generated RDF...

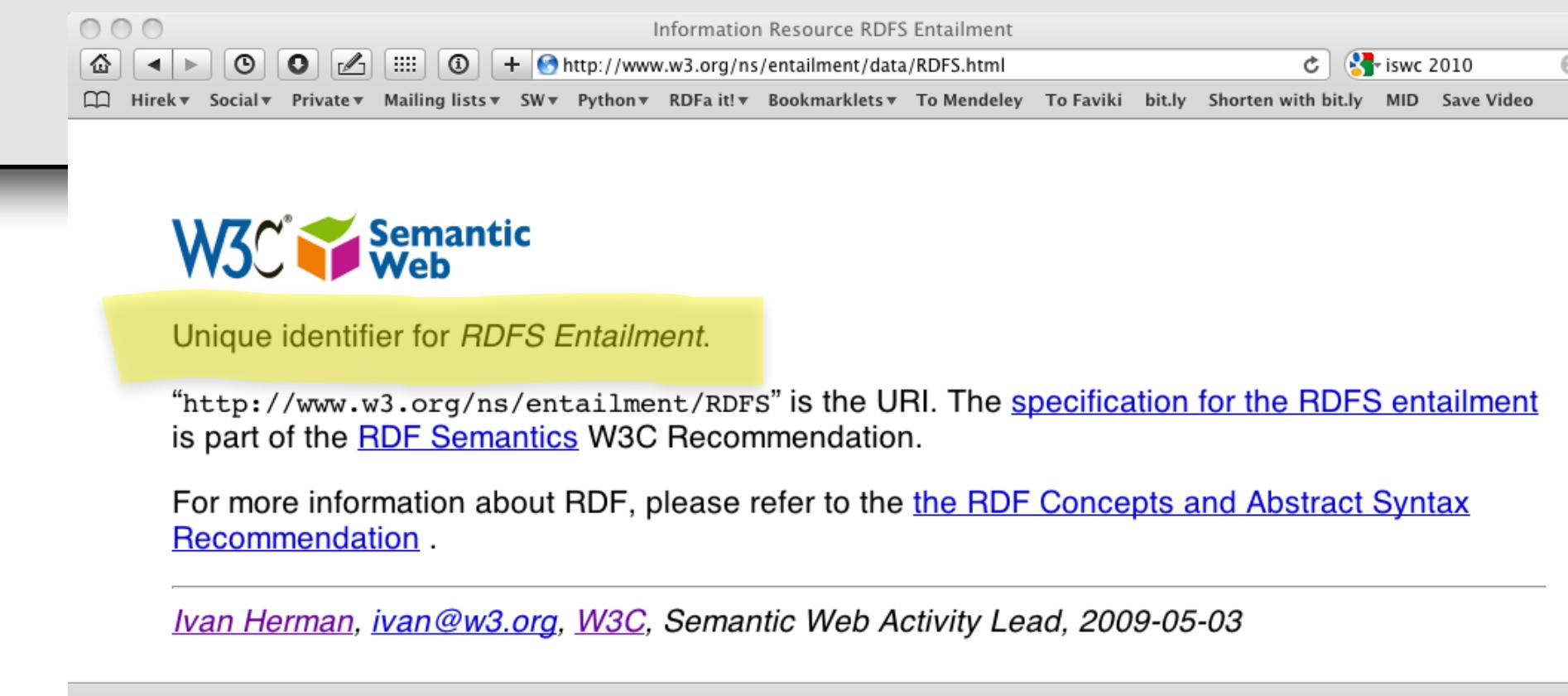
```
<p about="http://www.w3.org/ns/entailment/RDFS"
    property="http://purl.org/dc/terms/description">
    Unique identifier for
    <em>RDFS Entailment</em>.
</p>
```



<<http://www.w3.org/ns/entailment/RDFS>>

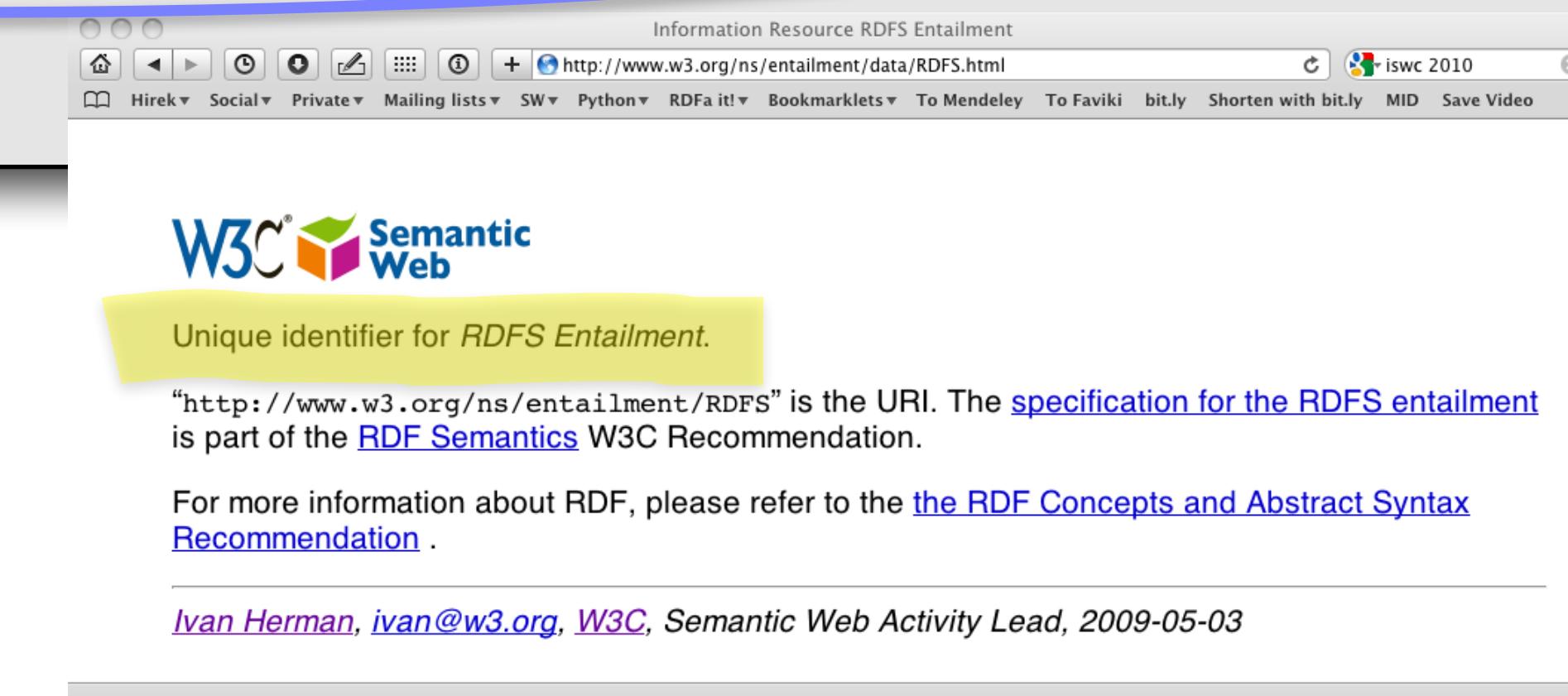
The source and generated RDF...

```
<p about="http://www.w3.org/ns/entailment/RDFS"
    property="http://purl.org/dc/terms/description">
  Unique identifier for
  <em>RDFS Entailment</em>.
</p>
```



The source and generated RDF...

```
<p about="http://www.w3.org/ns/entailment/RDFS"
    property="http://purl.org/dc/terms/description">
    Unique identifier for
    <em>RDFS Entailment</em>.
</p>
```



The source and generated RDF...

```
<p about="http://www.w3.org/ns/entailment/RDFS"
    property="http://purl.org/dc/terms/description">
    Unique identifier for
    <em>RDFS Entailment</em>.
</p>
```

```
<http://www.w3.org/ns/entailment/RDFS>
<http://purl.org/dc/terms/description>
"Unique identifier for RDFS Entailment ."
```

The source and generated RDF...

```
<a about="http://www.w3.org/ns/entailment/RDFS"  
rel="http://www.w3.org/2000/01/rdf-schema#seeAlso">  
  
href="http://www.w3.org/TR/2004/REC-rdf-mt-20040210/">  
  
</a>
```

The source and generated RDF...

```
<a about="http://www.w3.org/ns/entailment/RDFS"
  rel="http://www.w3.org/2000/01/rdf-schema#seeAlso">
  href="http://www.w3.org/TR/2004/REC-rdf-mt-20040210/">
</a>
```

```
<http://www.w3.org/ns/entailment/RDFS>
<http://www.w3.org/2000/01/rdf-schema#seeAlso>
```

The source and generated RDF...

```
<a about="http://www.w3.org/ns/entailment/RDFS"
    rel="http://www.w3.org/2000/01/rdf-schema#seeAlso">
    href="http://www.w3.org/TR/2004/REC-rdf-mt-20040210/">
</a>
```

```
<http://www.w3.org/ns/entailment/RDFS>
  <http://www.w3.org/2000/01/rdf-schema#seeAlso>
  <http://www.w3.org/TR/2004/REC-rdf-mt-20040210/>
```

Just compare

```
<http://www.w3.org/ns/entailment/RDFS>
  <http://purl.org/dc/terms/description>
    "Unique identifier for RDFS Entailment ."
<http://www.w3.org/ns/entailment/RDFS>
  <http://www.w3.org/2000/01/rdf-schema#seeAlso>
    <http://www.w3.org/TR/2004/REC-rdf-mt-20040210/> .
```

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dcterms: <http://purl.org/dc/terms/> .

<http://www.w3.org/ns/entailment/RDFS>
  rdfs:seeAlso <http://www.w3.org/TR/2004/REC-rdf-mt-20040210/> ;
  dcterms: description "Unique identifier for RDFS Entailment ."
```

The Turtle in RDFa

- Is that it?
 - The combination of **@about** with **@rel** / **@property** and possibly **@href** covers most of what we need...
 - but this is too complex for authors
- Go Turtle:
 - Use compact URIs when possible
 - Make use of the natural structure for
 - shared subjects
 - shared predicates
 - create blank nodes
 - ...

Compact URIs = CURIE's

- Just like in Turtle
 - define a prefix via @prefix
 - use prefix:reference to abbreviate a URI

Compact URIs = CURIE's

```
<html>
...
<p about="http://www.w3.org/ns/entailment/RDFS"
   property="http://purl.org/dc/terms/description">
    Unique identifier for
    <em>RDFS Entailment</em>.
</p></html>
```

```
<html prefix = "dcterms:http://purl.org/dc/terms/">
...
<p about="http://www.w3.org/ns/entailment/RDFS"
   property="http://purl.org/dc/terms/description">
    Unique identifier for
    <em>RDFS Entailment</em>.
</p></html>
```

RDFa supported attributes

- **xmlns**: a prefix and qualified URL defining a namespace for a document;
- **rel**: a white-space separated list of reserved keywords or CURIEs (**Compact URIs**) that details predicates between resources
 - No literals!
 - CURIES can be considered a datatype found both in XML and non-XML grammars
 - syntax: [isbn:0393315703]
 - the [] prevent ambiguity between CURIEs and regular URIs.
 - QNames can be considered a type of CURIES
- **rev**: similar to rel, but traverses the predicate in the opposite direction wrt **rel**

RDFa supported attributes

- **about**: a resource URI or CURIE used to represent the subject in an RDF triple
- **property**: a white-space separated list of CURIEs representing predicates between a subject and a plain literal.
- **rel**: represents predicates between a subject and another resource
 - ... and many more
- <http://www.w3.org/TR/rdfa-syntax/>
- <http://www.w3.org/TR/xhtml-rdfa-primer/>

Consuming RDFa

- Various search engines begin to consume RDFa
 - Google, Yahoo, ...
 - they may specify which vocabularies they “understand”
 - this is still an evolving area
- Facebook’s “social graph” is based on RDFa

Google's rich snippet

- Embedded metadata (microformat or RDFa) is used to improve search result page
- at the moment only a few vocabularies are recognised, but that will evolve over the years

gle

chicken noodle soup recipes

About 1,150,000 results (0.26 seconds)

Cooks.com - Recipes - Homemade Chicken Noodle SOUP

Enter your email to signup for the Cooks.com Recipe Newsletter.

NOODLE SOUP: Put chicken and all seasonings in a ... are soft.

www.cooks.com > Recipes - Cached - Similar

tools

Grandma's Chicken Noodle Soup Recipe - Allrecipes

★★★★★ 561 reviews - Prep time: 20 mins - Cook time: 25 mins

This is a recipe that was given to me by my grandmother. It is a ve and I believe that all will like it.

Who uses it

- A number of popular sites publish RDFa as part of their normal pages:
 - Google.com
 - Youtube.com
 - Facebook.com
 - Wikipedia.org
 - Yahoo.com
 - Amazon.com
 - Reddit.com
 - Netflix.com
 - Creative Commons snippets are in RDFa

COMP318: Ontology based Information Systems

www.csc.liv.ac.uk/~valli/Comp318



Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

v.Tamma@liverpool.ac.uk

Why do we need ontologies

- Ontologies provide a common vocabulary and definition of rules defining the use of the terms by independently developed resources, processes, services
- That means....



An example

42

An example

42

The password for a Microsoft Windows domain will expire by default after 42 days

Answer to the Ultimate Question of Life, the Universe, and Everything

A track in the 2008 Coldplay album Viva la vida

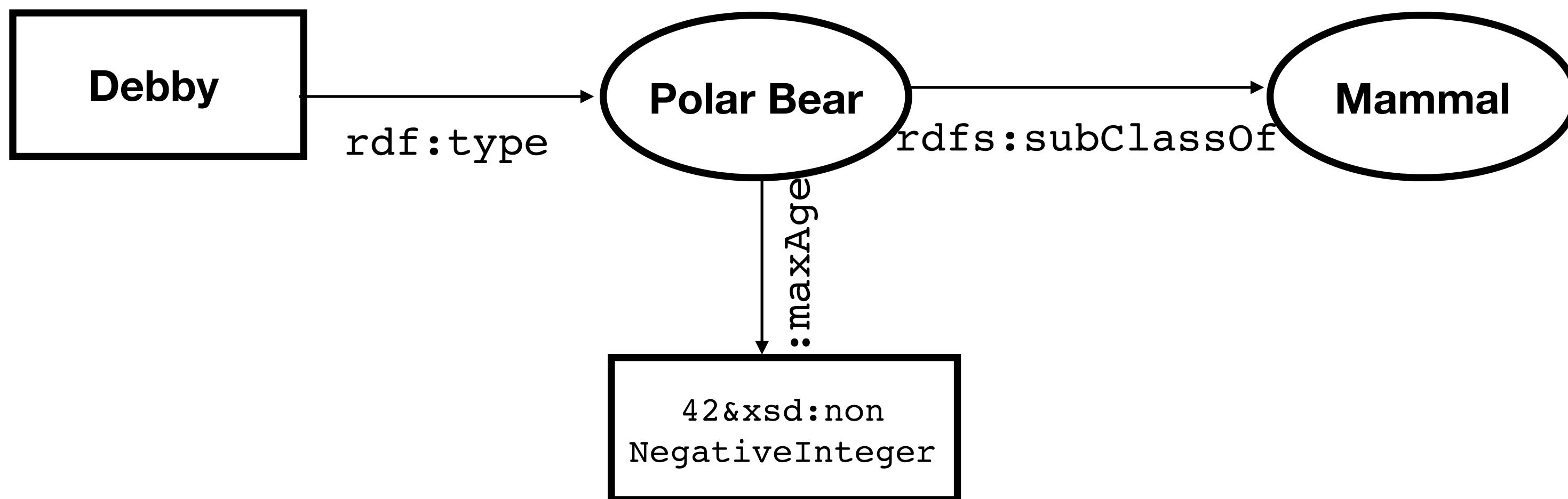
The angle (rounded to whole degrees) for which a rainbow appears

An example

42 years

An example

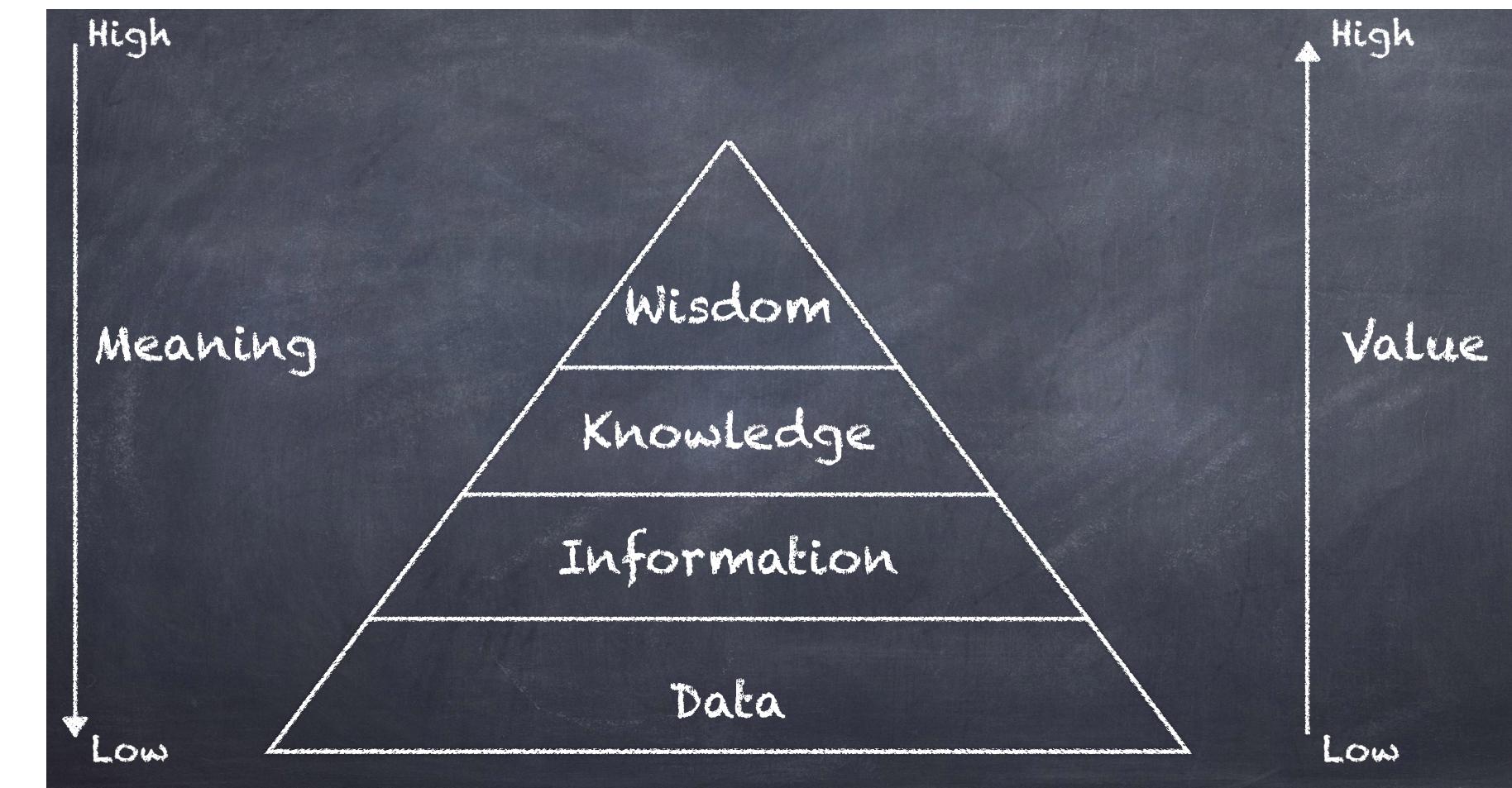
- Max recorded age for a polar bear in captivity
 - The oldest polar bear on record was Debby, who died at Assiniboine Park Zoo, Canada, in 2008 aged 42



Class: PolarBear SubClassOf: Mammal and maxAge only 42

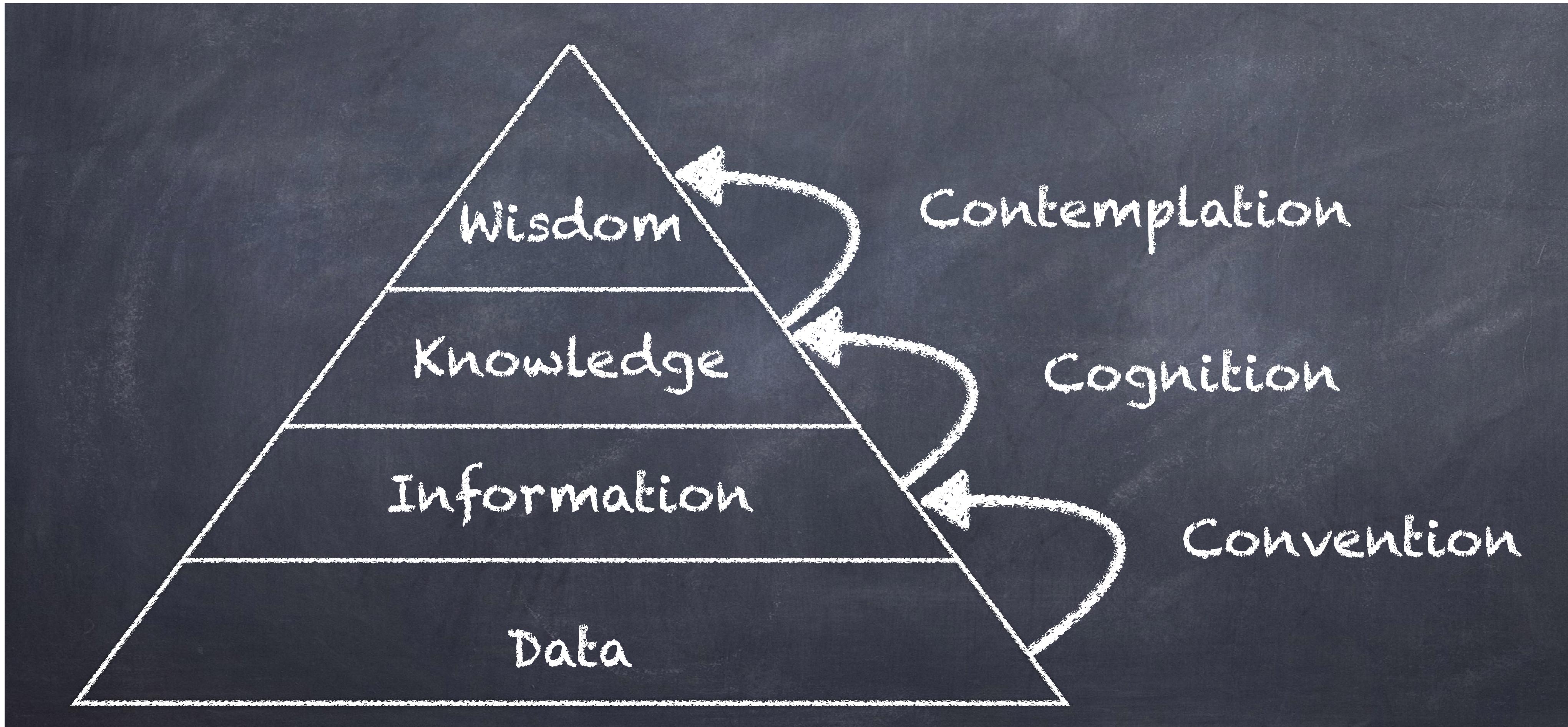
DIKW model

- **Data:** unorganised and unprocessed discrete, objective facts or observations
 - have no meaning or value because of lack of context and interpretation (e.g. raw data)
- **Information:** organised or structured data, processed in such a way that the information now has relevance for a specific purpose or context
 - meaningful, valuable, useful and relevant
 - that which reduces uncertainty (Shannon)
- **Knowledge:** Different perspectives:
 - a mix of contextual information, values, experience and rules;
 - know-how;
 - information combined with understanding and capability;
 - belief structuring" and "internalization with reference to cognitive frameworks
- **Wisdom:** the knowledge and insights into a learning experience that guides our actions
 - evaluated understanding

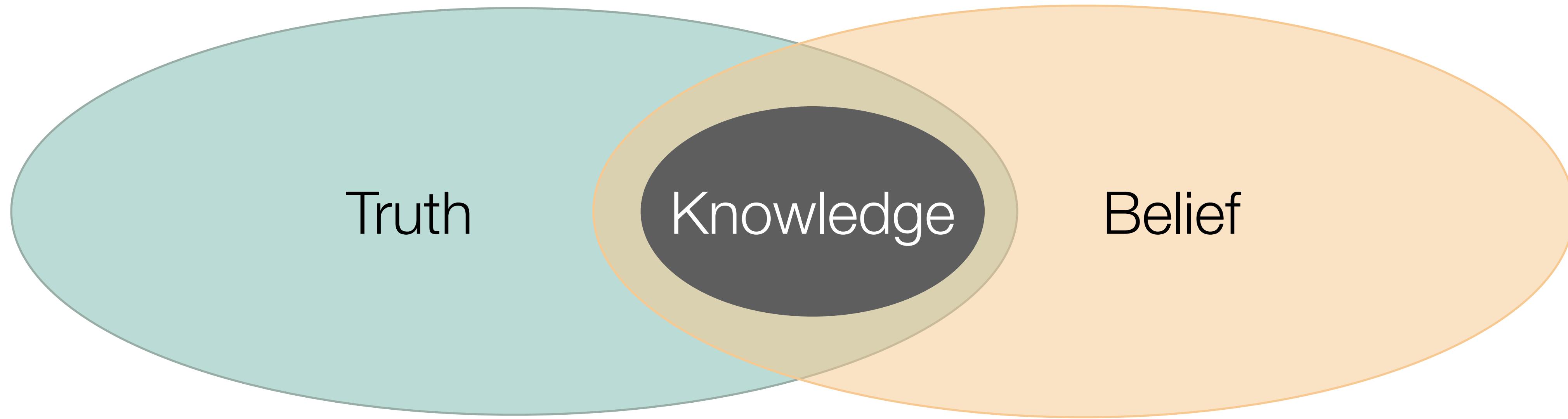


"Big data is not knowledge"
Y. Frégnac, Science 358, 6362 (2017)

DIKW model



What is knowledge



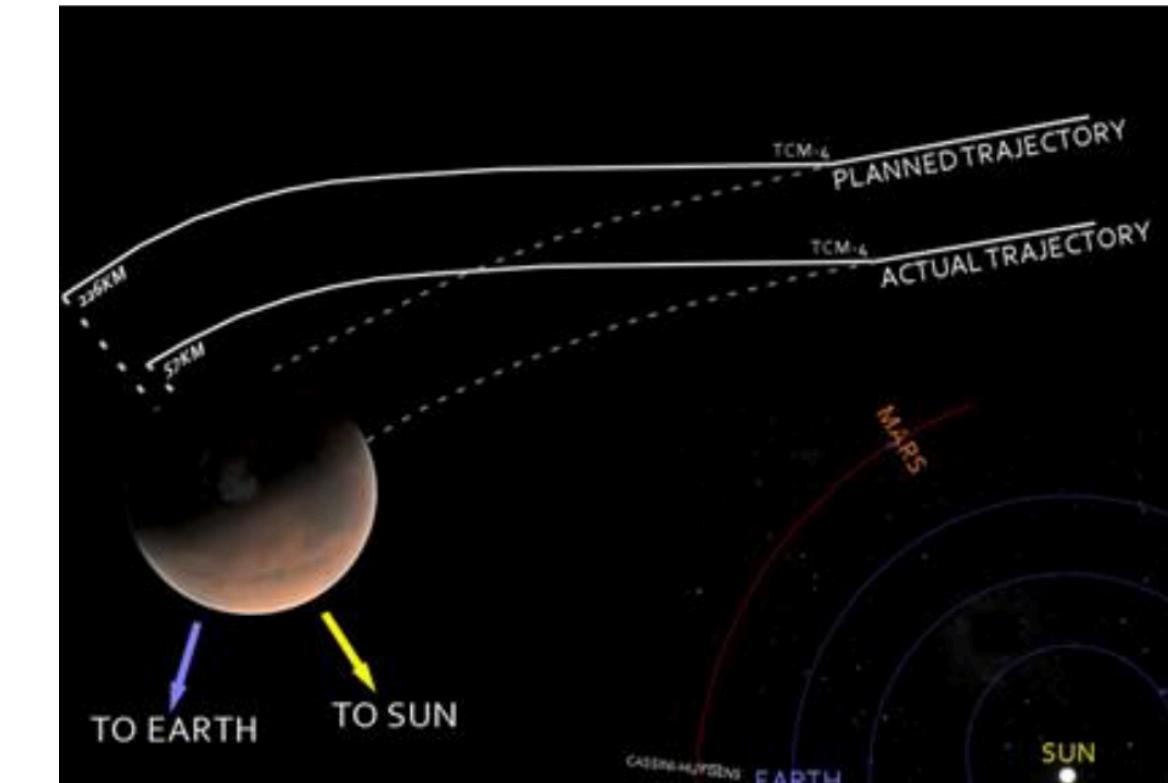
Knowledge sharing

- Sharing knowledge depends on having:
 - common symbols and concepts (**Syntax**)
 - agreement about their meaning (**Semantics**)
 - classification of concepts (**Taxonomy**)
 - associations and relations of concepts (**Thesauri**)
 - rules and knowledge about which relations are allowed and make sense (**Ontologies**)



NASA and ontology standardisation

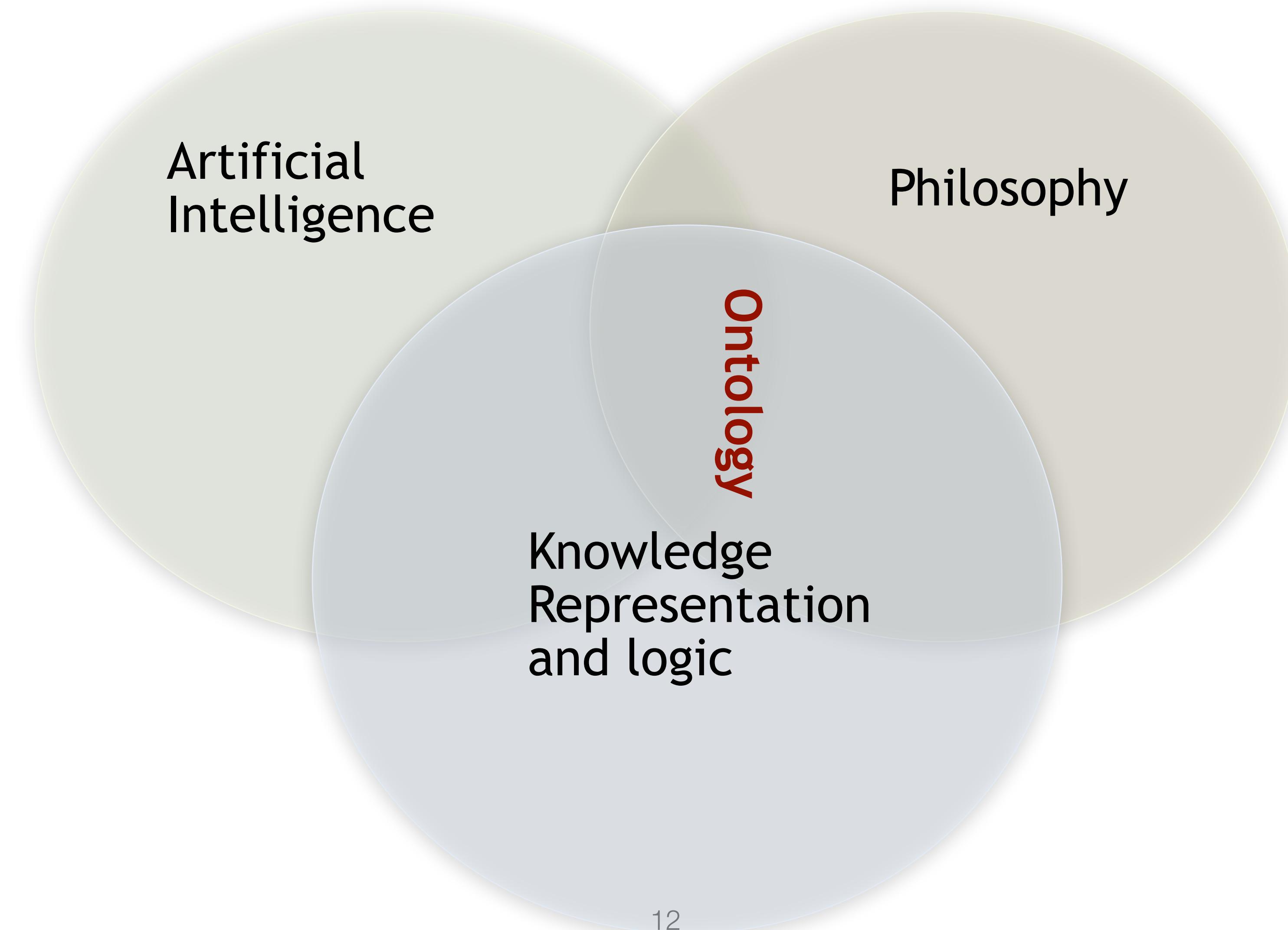
- NASA metric confusion caused the Mars Orbiter loss in 1999.



The failure to use metric units in the coding of a ground software title, **SMALL FORCES**, used in trajectory models. Specifically, thruster performance data in English units instead of metric units was used in the software application code titled **SM_FORCES** (small forces). The output from the **SM_FORCES** application code as required by a **MSOP** Project Software Interface Specification (SIS) was to be in metric units of Newton-seconds (N-s). Instead, the data was reported in English units of pound-seconds (lbf-s).

NASA Mishap Investigation Board Report

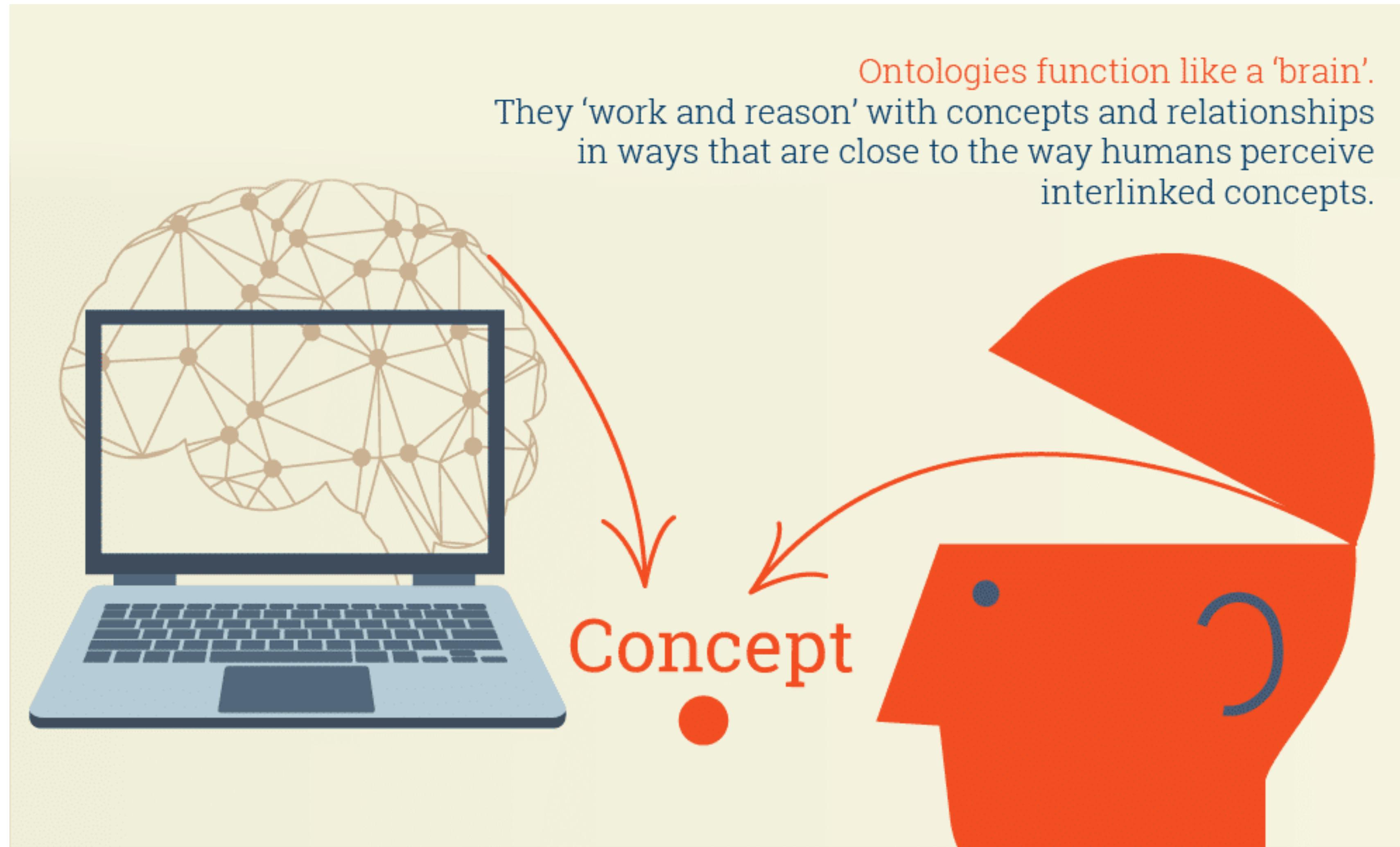
Ontology



From philosophy to computer science

- Socrates questions of being, Plato's studies of epistemology:
 - the nature of knowledge
- Aristotle's classifications of things in the world and contribution to syllogism and inductive inference:
 - logic as a precise method for reasoning about knowledge
- In computer science an unambiguous description of the concepts and relationships that can exist for one or more agents, so they can ***understand, share, and use*** this description to accomplish ***(cooperatively)*** some task on behalf of users

Ontologies are the “brain” of your app



So what is an ontology then?

“...An ontology is a (formal), explicit specification of a shared conceptualisation...”

formal: an ontology
should be machine-
readable

shared: an ontology captures consensual
knowledge, that is not private to some
individual, but accepted by a group

explicit: the types of concepts
used, and the constraints on
their use are explicitly defined

conceptualisation: an abstract model of some
phenomenon in the world which identifies the
relevant concepts of that phenomenon

What is a conceptualisation

- **Conceptualisation:** the formal structure of reality as perceived and organised by an agent, independently of:
 - the vocabulary used (i.e., the language used)
 - the actual occurrence of a specific situation
- Different situations involving the same objects, described by different vocabularies, may share the same conceptualisation.

soccer



football

Who is using ontologies

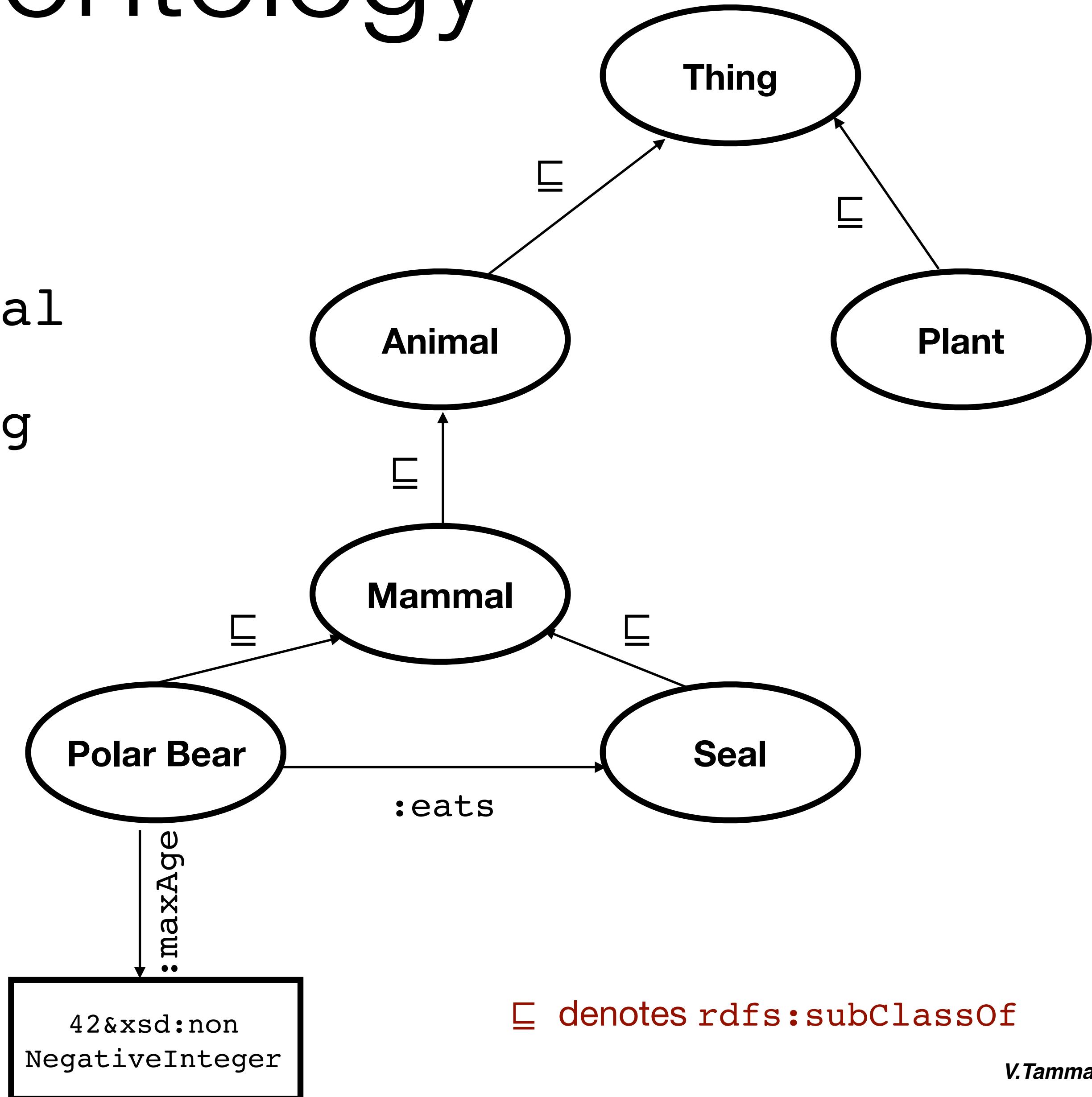


*“Who's doing this? 75% of the Fortune 500 companies have some kind of **smart data or semantics program underway**, most under the banner of 360° initiatives, comprehensive enterprise data systems, or machine learning/data science projects. **Amazon has recently added linked data capabilities** to their AWS infrastructure with the Neptune project, and **social media giants have built their entire data infrastructure around smart ontological data**. Moreover, China, Japan, England, the OECD, and the United States have all **moved critical data resources into semantic form**, and **semantics has become one of the hottest areas for investment banks such as Wells Fargo, Morgan Stanley, Citigroup, Goldman Sachs and others**. It even ties into such cutting-edge technologies as Blockchain and the Internet of Things.”*

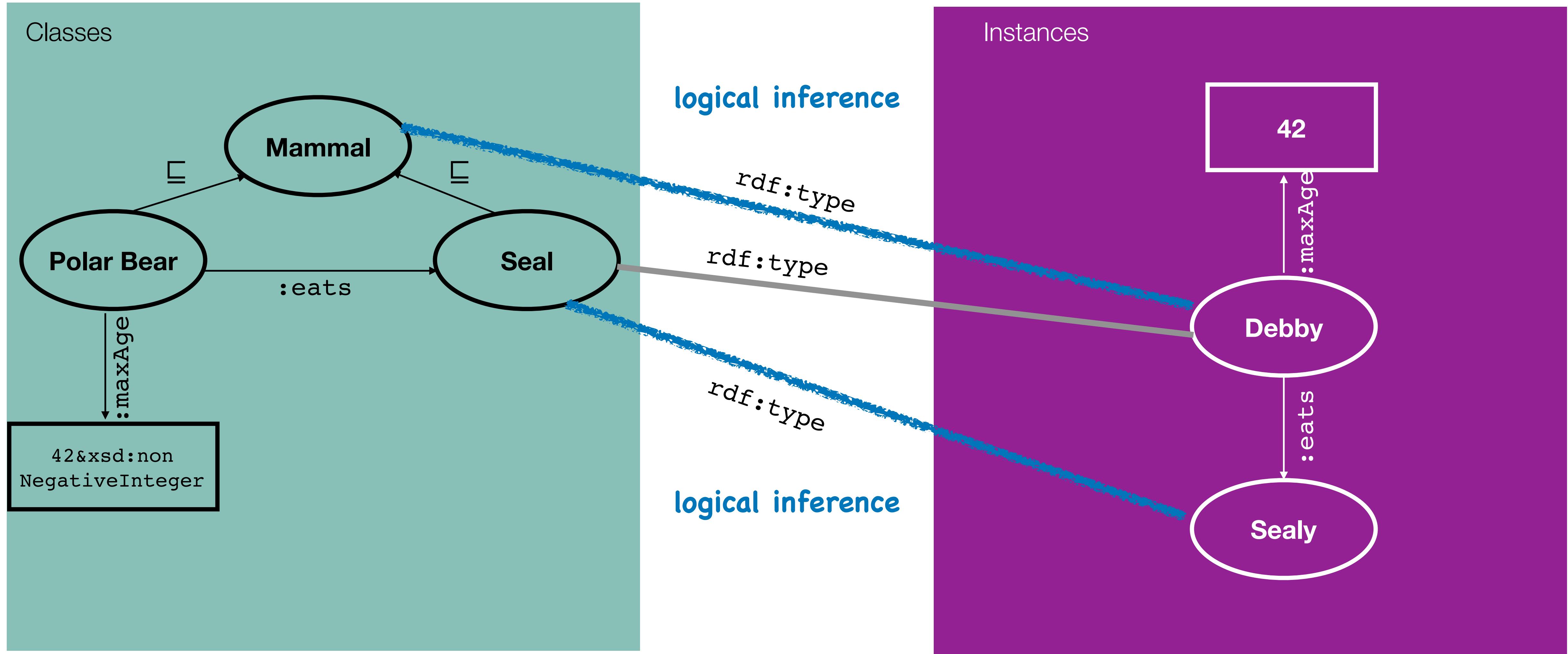
Forbes, July 2018

A simple ontology

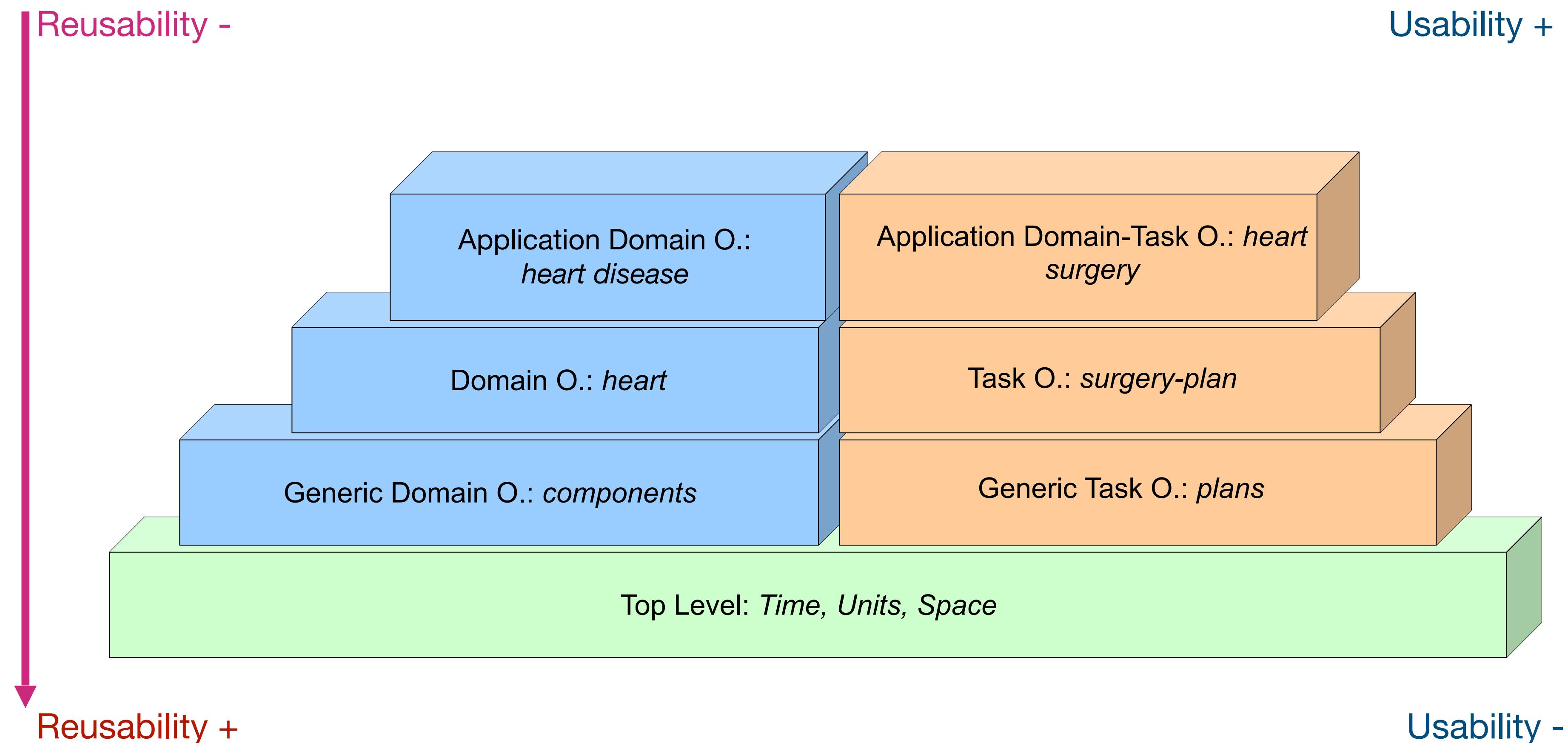
- Class: PolarBear SubClassOf:
Mammal and maxAge only 42
- Class: Mammal SubClassOf: Animal
- Class: Animal SubClassOf: Thing
- Class: Plant SubClassOf: Thing
- Class: Seal SubClassOf: Mammal
- ObjectProperty: eats
 - Domain: PolarBear
 - Range: Seal



A simple ontology: inferences



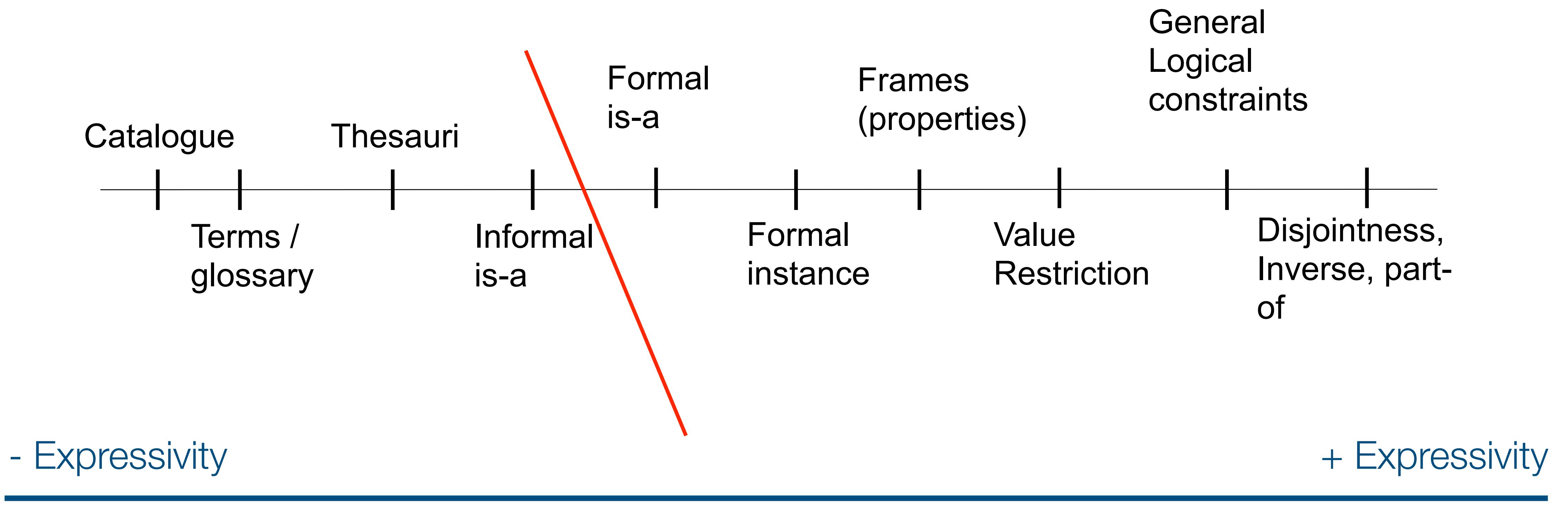
Types of Ontologies



Types of ontologies

- **Top level ontology:** general, cross domain ontologies; represent very general concepts as e.g., Time, Space, Event; independent of a specific domain or problem
 - also called Upper Ontology or Foundational Ontology
- **Domain ontology:** fundamental concepts according to a generic domain;
 - specialises terms introduced in top-level ontology
- **Task ontology:** fundamental concepts according to a general activity or task; specializes terms introduced in top-level ontology
- **Application ontology:** specialized ontology focussed on a specific task and domain;
 - often a specialization of both task and domain ontology; often specify roles played by domain entities for specific activity

Level of Granularity

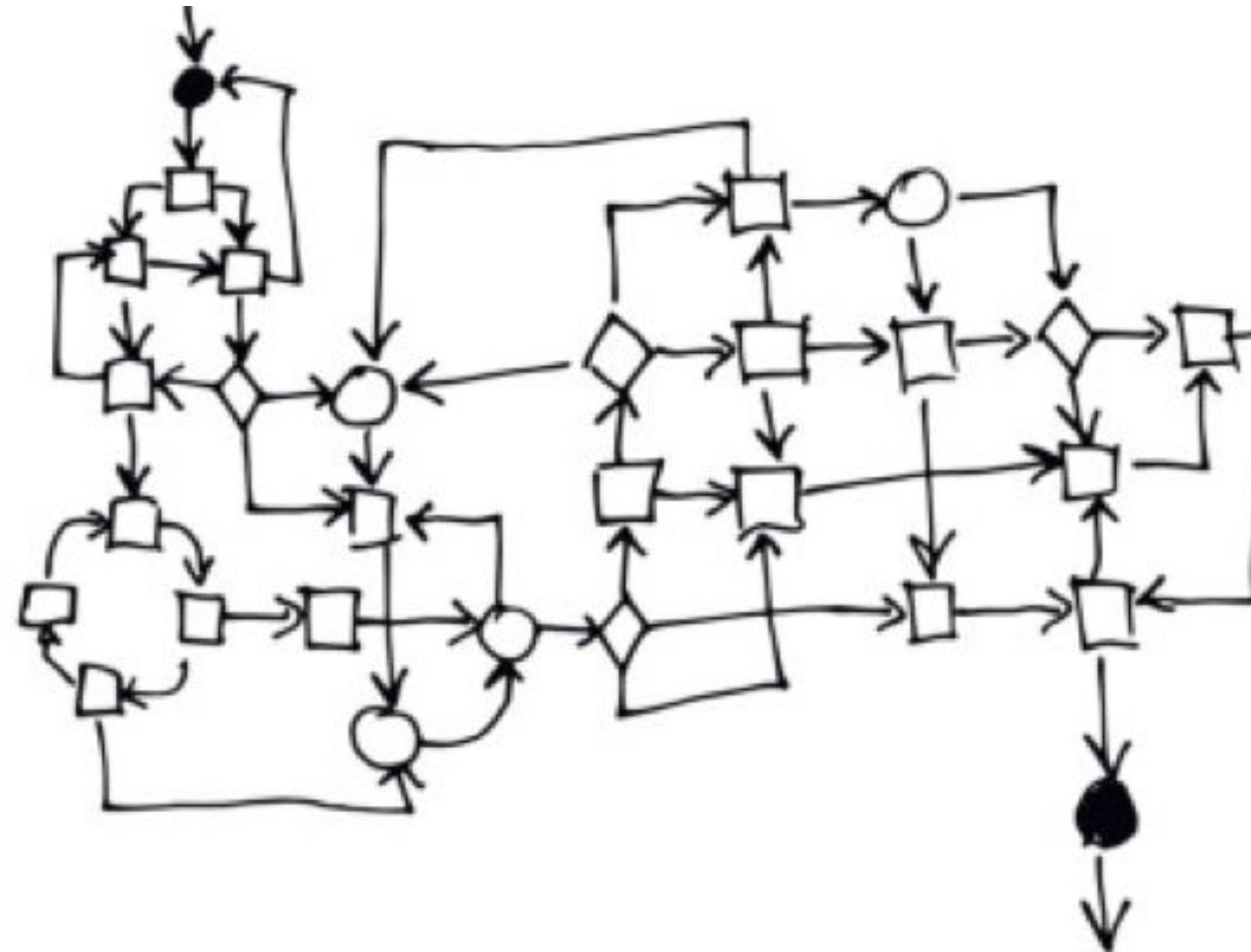


Level of Granularity

- An ontology specifies a rich description of the:
 - Terminology, concepts, vocabulary
 - Properties explicitly describing concepts
 - Relations among concepts
 - Rules distinguishing concepts,
 - refining definitions and relations (constraints, restrictions, regular expressions) relevant to a particular domain or area of interest.

Ontology development process

SOMETHING



Ontology Engineering

“The set of activities that concern the ontology development process, the ontology life cycle, and the methodologies, tools and languages for building ontologies”

Gomez-Perez et al, 2004



Ontology Engineering

- Defining terms in the domain and relations among them
 - Defining concepts in the domain (**classes**)
 - Arranging the concepts in a hierarchy (**subclass-superclass hierarchy**)
 - Defining which attributes and properties classes can have
 - and constraints on their values
 - Defining individuals and filling in property values



Methodological questions

- What part of the domain do we need to model?
- What are the constraints on the use of this knowledge?
- How can tools and techniques best be applied?
- Which languages and tools should be used in which circumstances, and in which order?
- What about issues of quality control and resource management?
 - Many of these questions for ontology engineering have been studied in other contexts
 - E.g. software engineering, object-oriented design, and knowledge engineering

Summary

- What are ontologies
 - and what do we use them for
- Types of ontology
- Ontology engineering

COMP318: Ontology based Information Systems

www.csc.liv.ac.uk/~valli/Comp318



Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

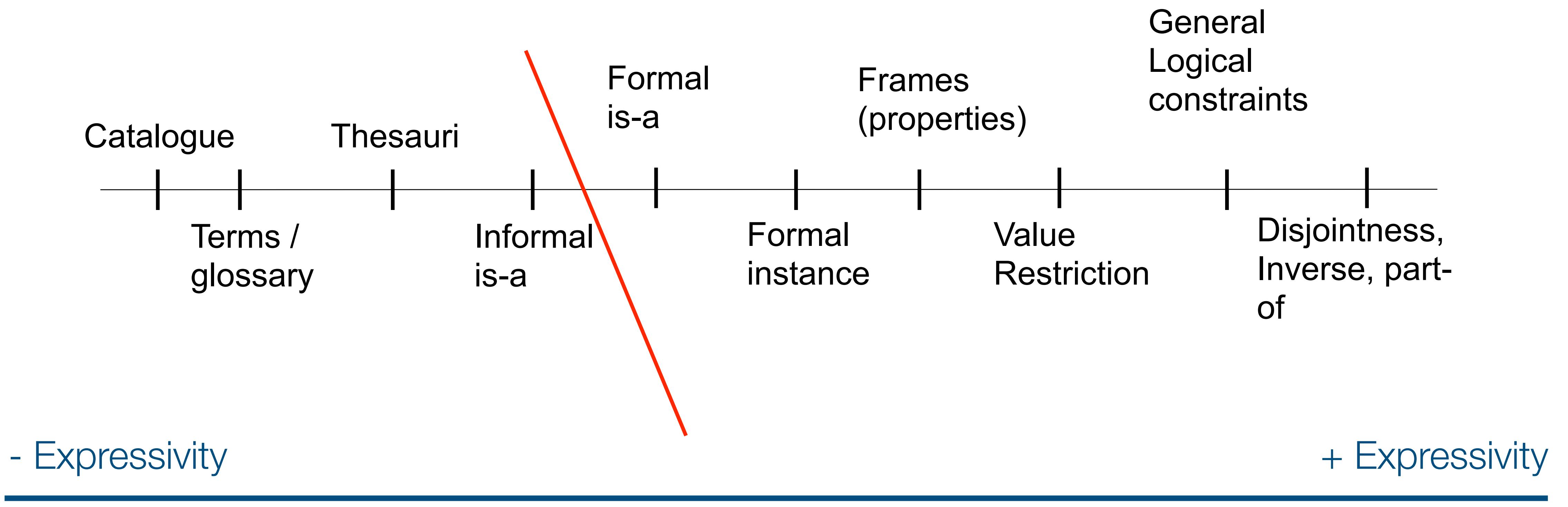
University of Liverpool

v.Tamma@liverpool.ac.uk

Where were we

- Motivation behind the need for ontologies
 - Data - Information - Knowledge - Wisdom model
 - Making knowledge explicit:
 - Concepts, relationships and constraints
- Definition of ontology in Computer Science
- Types of ontologies

Level of Granularity

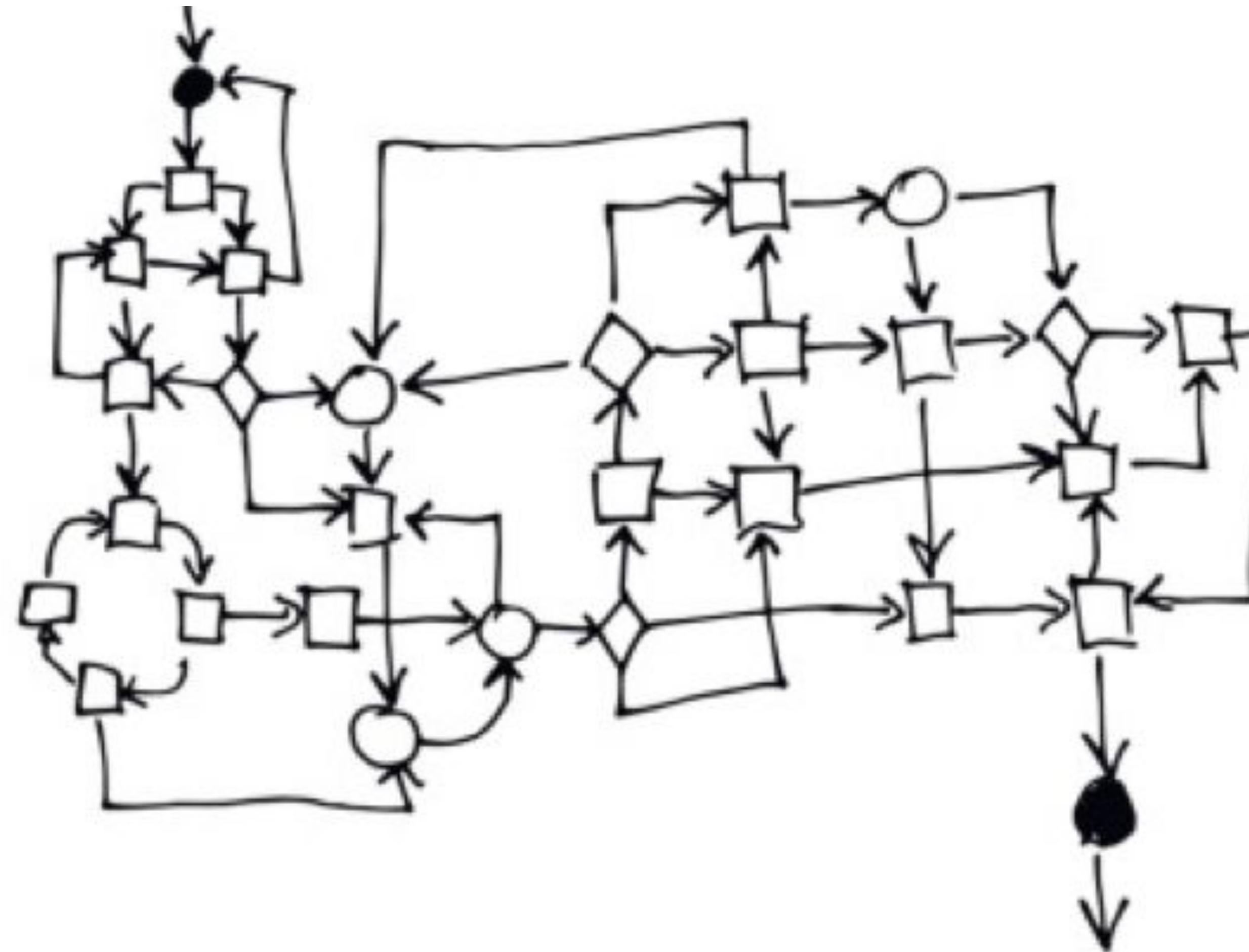


Level of Granularity

- An ontology specifies a rich description of the:
 - Terminology, concepts, vocabulary
 - Properties explicitly describing concepts
 - Relations among concepts
 - Rules distinguishing concepts,
 - refining definitions and relations (constraints, restrictions, regular expressions) relevant to a particular domain or area of interest.

Ontology development process

SOMETHING



Great Ontology

Ontology Engineering

“The set of activities that concern the ontology development process, the ontology life cycle, and the methodologies, tools and languages for building ontologies”

Gomez-Perez et al, 2004



Ontology Engineering

- Defining terms in the domain and relations among them
 - Defining concepts in the domain (**classes**)
 - Arranging the concepts in a hierarchy (**subclass-superclass hierarchy**)
 - Defining which attributes and properties classes can have
 - and constraints on their values
 - Defining individuals and filling in property values



Methodological questions

- What part of the domain do we need to model?
- What are the constraints on the use of this knowledge?
- How can tools and techniques best be applied?
- Which languages and tools should be used in which circumstances, and in which order?
- What about issues of quality control and resource management?
 - Many of these questions for ontology engineering have been studied in other contexts
 - E.g. software engineering, object-oriented design, and knowledge engineering

Principles for the design of an ontology

- **Clarity:**

- To communicate the intended meaning of defined terms
 - An ontology should communicate **effectively** the intended meaning of defined terms.
 - Definitions should be objective and should be stated with formal axioms.
 - A complete definition (defined by necessary and sufficient conditions) is preferred over a partial definition (defined by only necessary or sufficient conditions)

- **Coherence:**

- To sanction inferences that are consistent with definitions
 - If a sentence that can be inferred from the axioms contradicts a definition or example given informally, then the ontology is incoherent.

- **Extendibility:**

- To anticipate the use of the shared vocabulary

- define new terms for special uses based on the existing vocabulary, in a way that does not require the revision of the existing definitions.

- **Minimal Encoding Bias:**

- To be independent of the symbolic level
 - The conceptualization is specified at the knowledge level without depending on a particular symbol-level encoding.

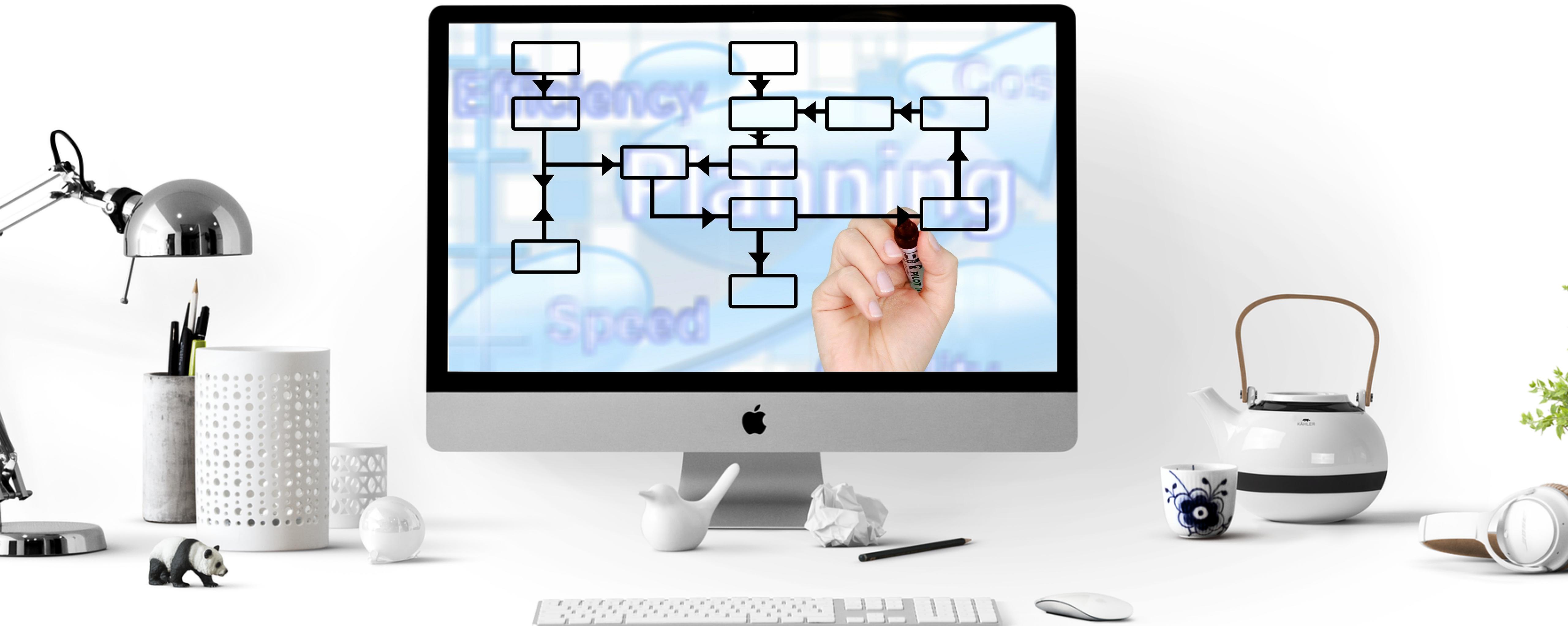
- **Minimal Ontological Commitments:**

- To make as few claims as possible about the world
 - ontological commitment is based on the consistent use of the vocabulary,
 - ontological commitment can be minimised by specifying the weakest theory and defining only those terms that are essential to the communication of knowledge consistent with the theory.

Ontological Commitment

- Agreements to use the vocabulary in a coherent and consistent manner
 - An agent commits (conforms) to an ontology if it “acts” consistently with the definitions
- The assignment of the meaning to the terms in the ontology vocabulary

Ontology engineering methodologies



Methodologies

- Various methodologies have been proposed to formalise the development process, or even some of its phases
 - Uschold & King
 - Gruninger & Fox
 - Methontology (Gomez Perez et al)
 - Ontology Design Patterns (Gangemi et al)
 - Ontology 101 (Noy & McGuinness)

Uschold & King

- Identify purpose
- Build ontology
 - Capture
 - Coding
 - Integrating
- Evaluation
- Documentation

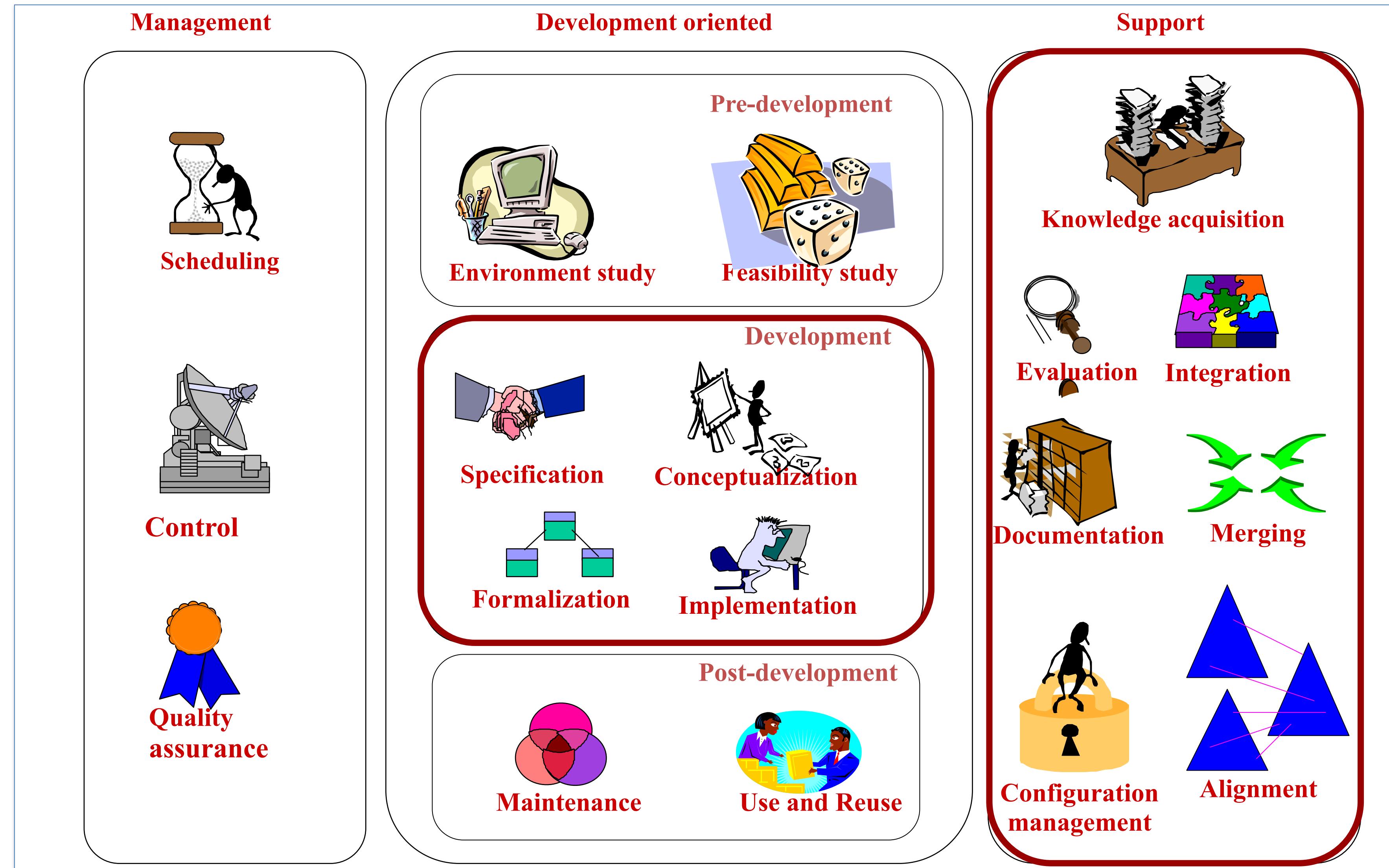
Gruninger & Fox

- Identify motivating scenarios
- Elaborate informal competency questions
- Specify terminology in FOL
 - Identify objects
 - Identify predicates
- Formal competency questions
- Specify axioms in FOL
- Specify completeness theorems

Methontology

- Development + Management Activities + Support in parallel
- Development
 - Specification
 - Conceptualization
 - Formalization
 - Implementation
 - Maintenance
- Focus on the conceptualization activity
 - Advantage: Integration of existing ontologies considered from early on.
- Conceptualization is evaluated early on, which prevents propagation of errors.

Ontology development process

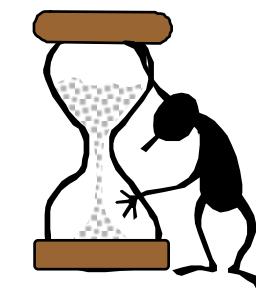


Management activities

Management

- **Scheduling**

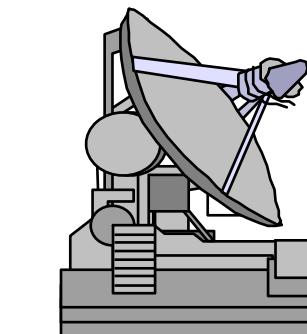
- Identification of tasks/problems to solve
- Arrangement/planning of tasks/problems to solve
- Identification of required resources (time, memory, resources)



Scheduling

- **Control**

- Ensuring the correct execution of tasks / problems to solve



Control

- **Quality Assurance**

- Ensuring the quality of all the artefacts produced during development
 - ontologies, software, and documentation



Quality
assurance

Development activities of the design process

- Pre-Development
- Development
- Post-Development
- Support



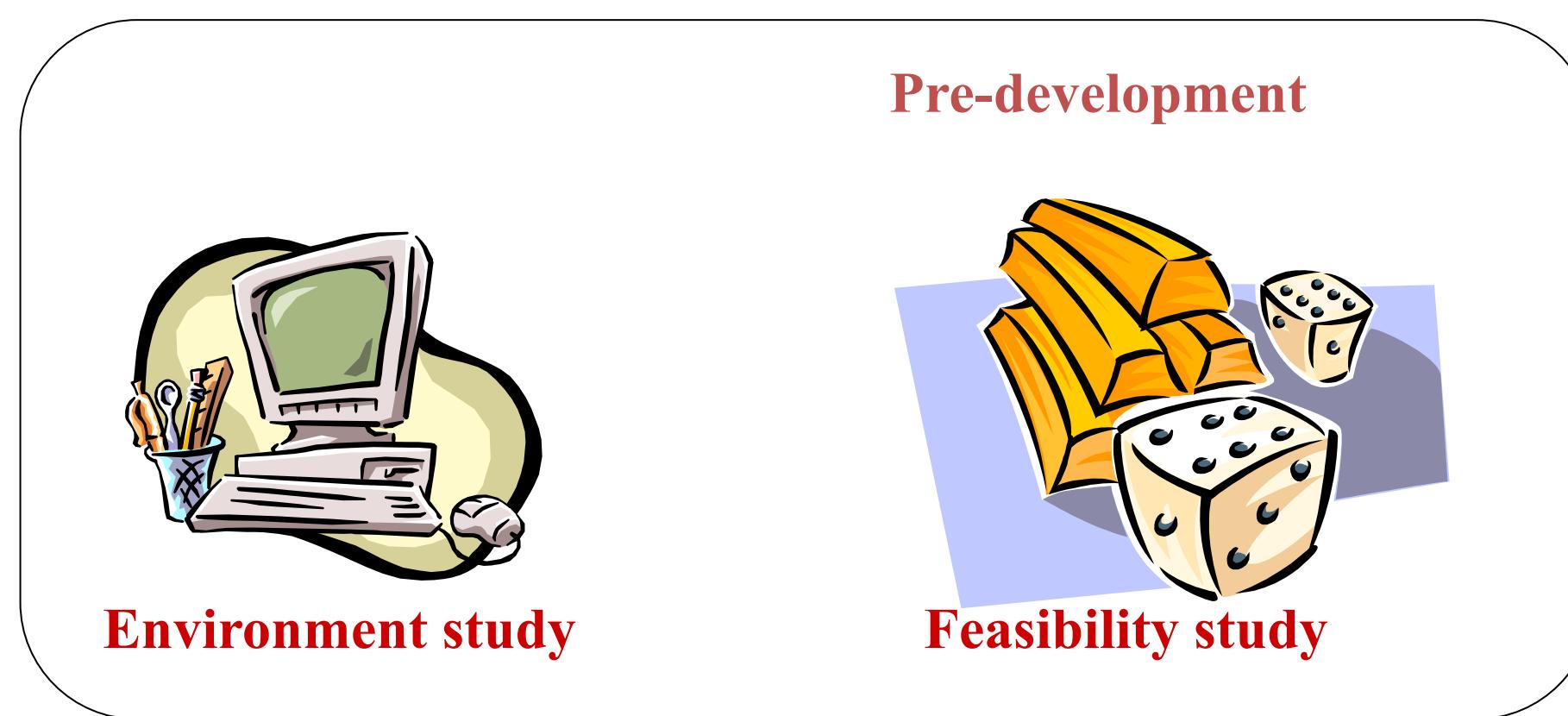
Pre-development activities

- **Environment study**

- Determine the environment in which the ontology should operate:
 - What is the designated software platform for the ontology?
 - Which applications should use the ontology?

- **Feasibility study**

- Assesses the feasibility and value of the ontology
 - Can the ontology really be developed?
 - Does it make sense to develop the ontology?



Development activities

- **Specification**

- Use-cases and requirements
 - Why is the ontology developed, what is the benefit and who are the end-users?
 - What are the questions the ontology should answer?

- **Conceptualisation**

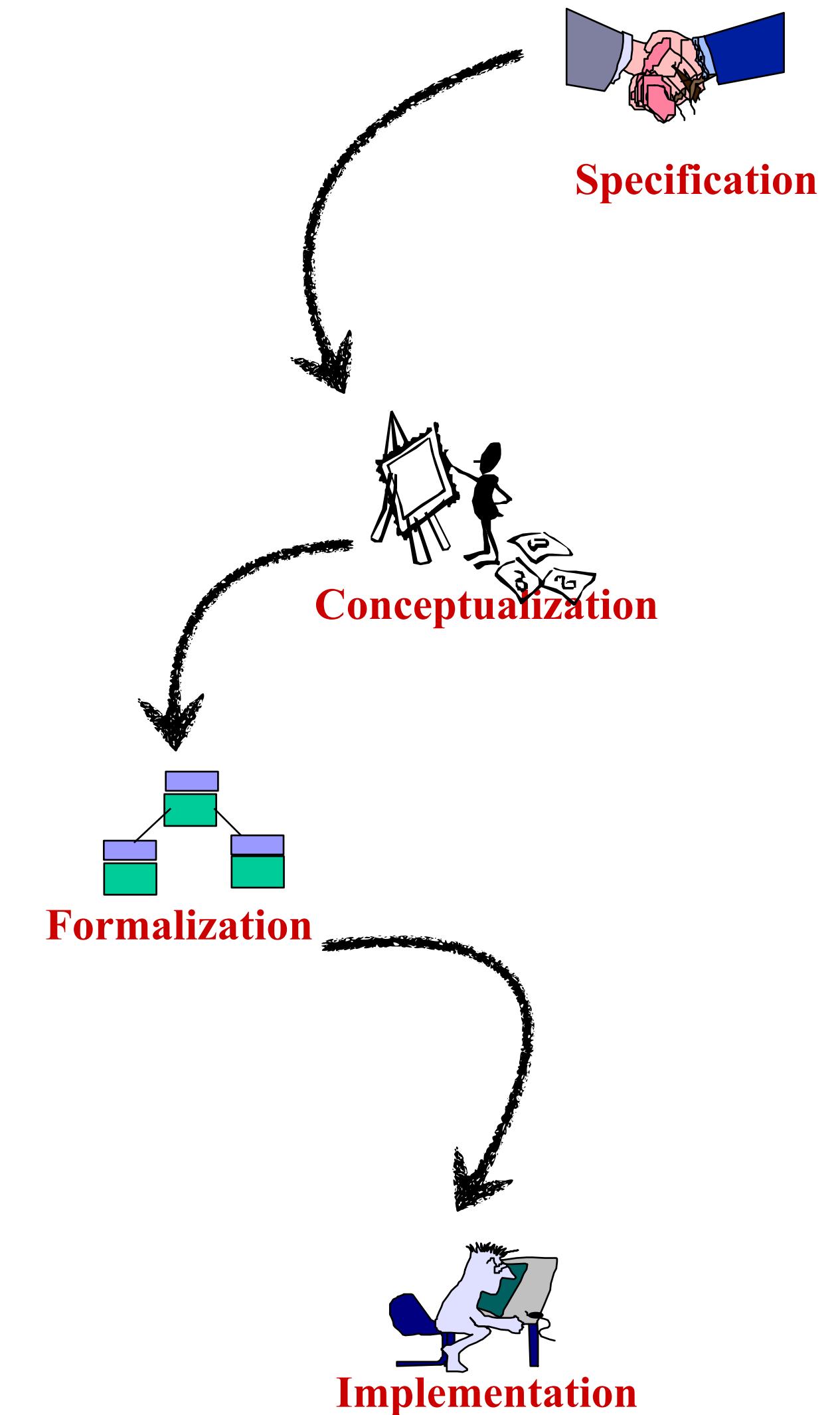
- Generation of a conceptual model that provides an abstraction of the domain model

- **Formalisation**

- Translation the conceptual model into a (semi) computable model

- **Implementation**

- Construction of a of a computable model in an ontology representation language (e.g. RDFS, OWL)



Post-Development Activities

- **Maintenance**

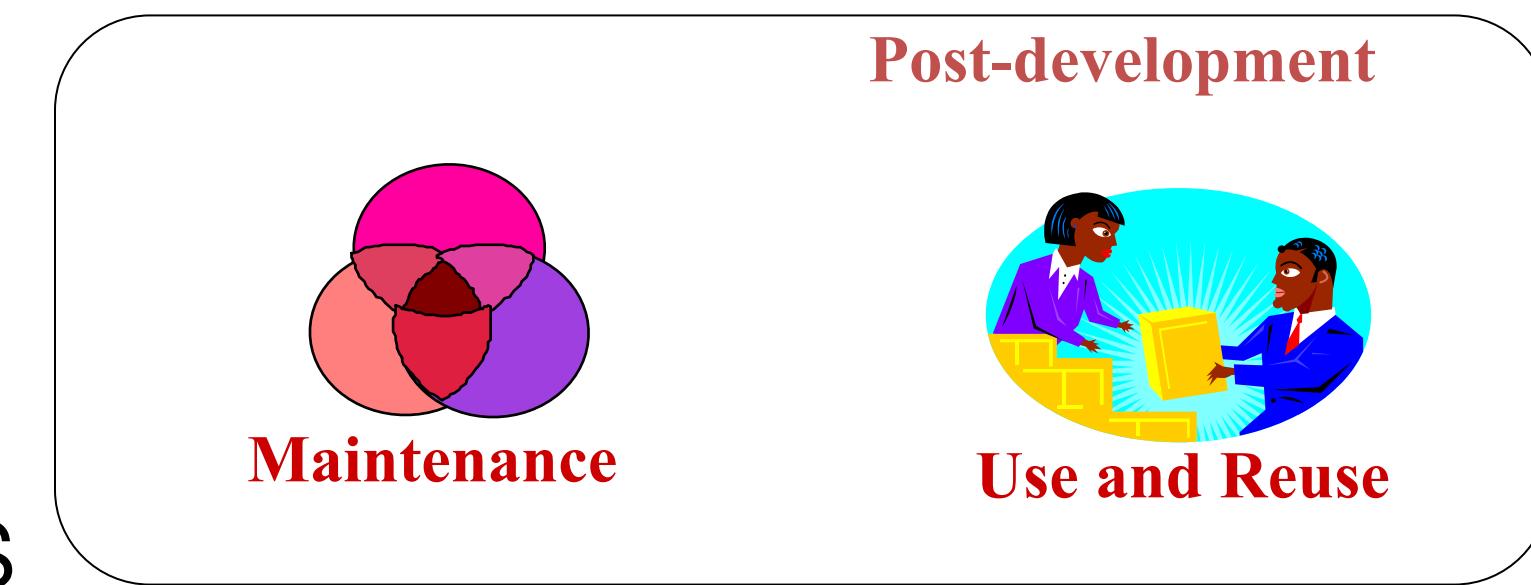
- Update and amendment of the ontology (if necessary)

- **Use**

- Usage of the ontology within the designated applications

- **Reuse**

- Use of the ontology in novel, unplanned applications
 - very common for top level ontologies



Support Activities

• Knowledge Acquisition

- The process of eliciting (tacit) knowledge from domain expert
- Learning semi-automatically the ontology from text

• Evaluation

- Technical evaluation of the outcome of each step of the ontology development process

• Documentation

- Accurate documentation of each step of the ontology development process

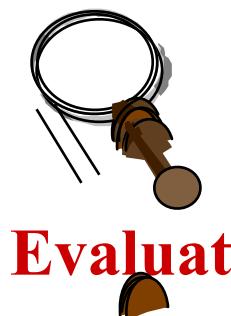
• Configuration management

- Management of the different versions of the ontology produced and its documentation
- Versioning

Support



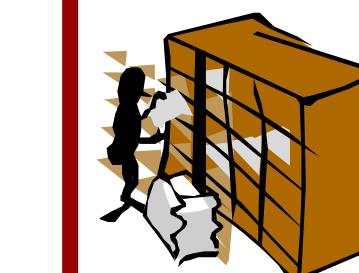
Knowledge acquisition



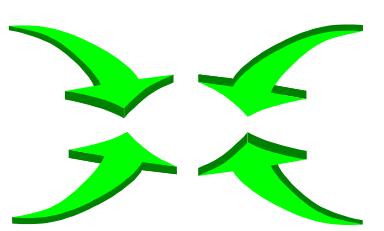
Evaluation



Integration



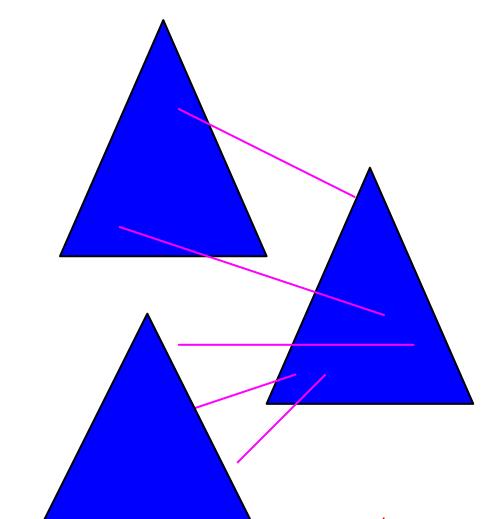
Documentation



Merging



Configuration management



Alignment

Support Activities

• Integration

- Integrating two ontologies through the definition of mappings between them
 - Definition of a global schema

• Merging

- Ontology merging describes the process of integrating two (or more) ontologies into a single one.

• Alignment

- Determine or apply mapping rules for reconciling the involved ontologies

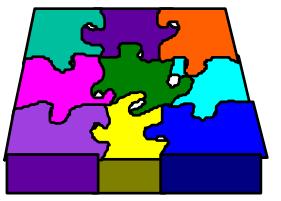
Support



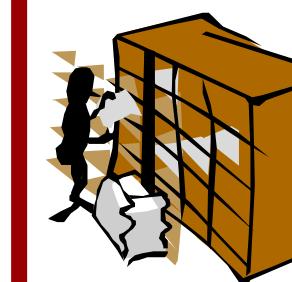
Knowledge acquisition



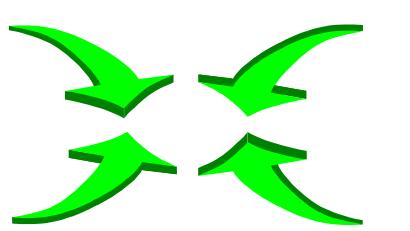
Evaluation



Integration



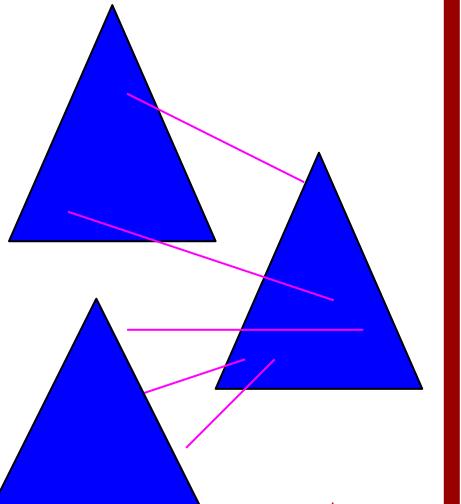
Documentation



Merging



Configuration management



Alignment

Ontology design patterns

- Adapting a design idea originally from architecture
 - recurring modeling problems
 - providing a set of adaptable standard solutions
 - a “**pattern**” is a solution to a problem in a given context

Summary

- Types of ontology
- Ontology engineering principles
- Ontology engineering methodologies

COMP318: Ontology based Information Systems

www.csc.liv.ac.uk/~valli/Comp318



Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

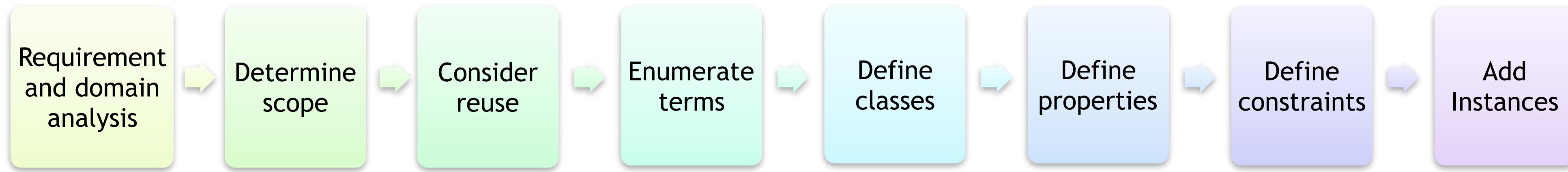
University of Liverpool

v.Tamma@liverpool.ac.uk

Where were we

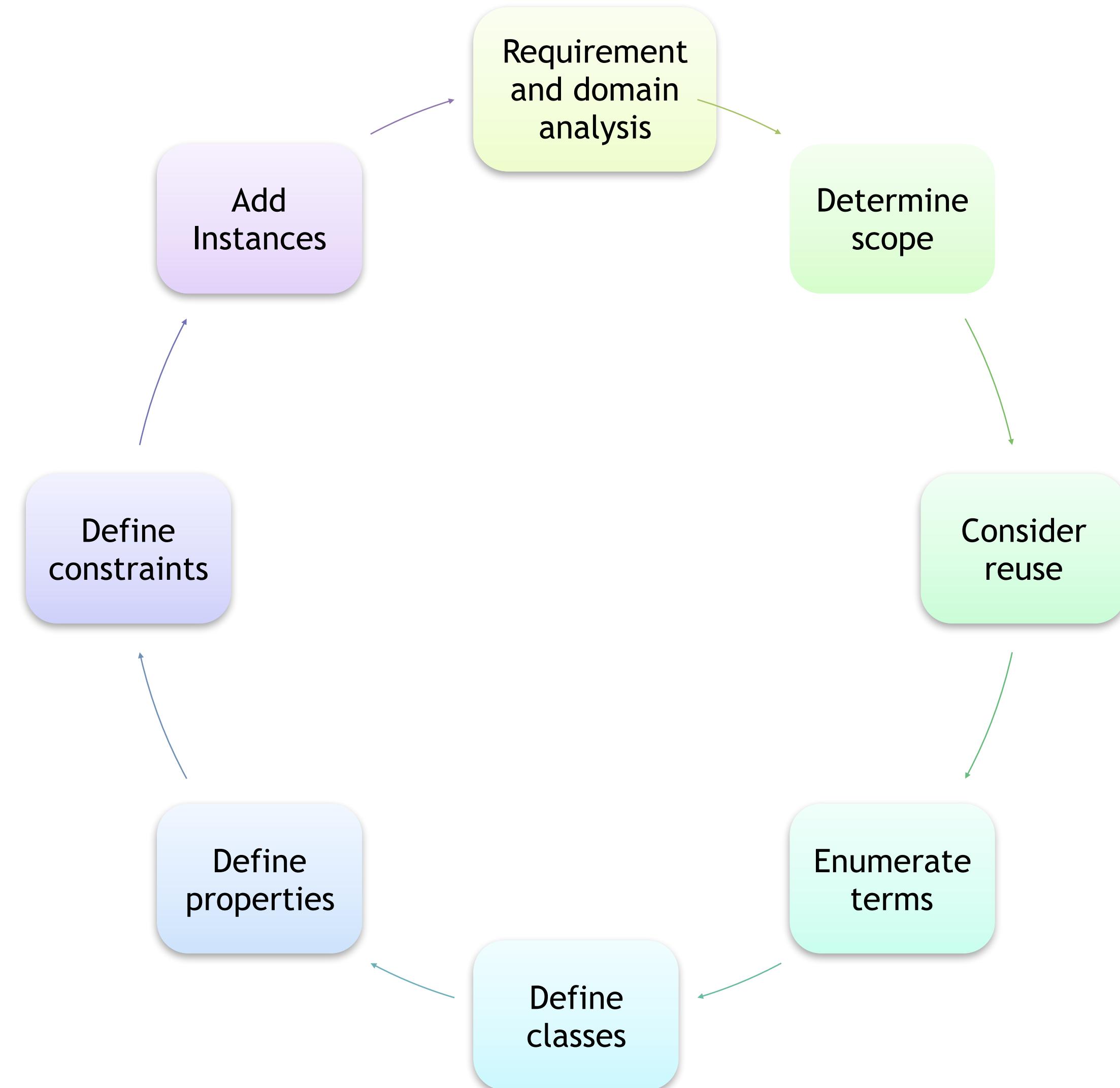
- Motivation behind the need for ontologies
- Ontology engineering
 - Principles
 - Methodologies

Ontology 101

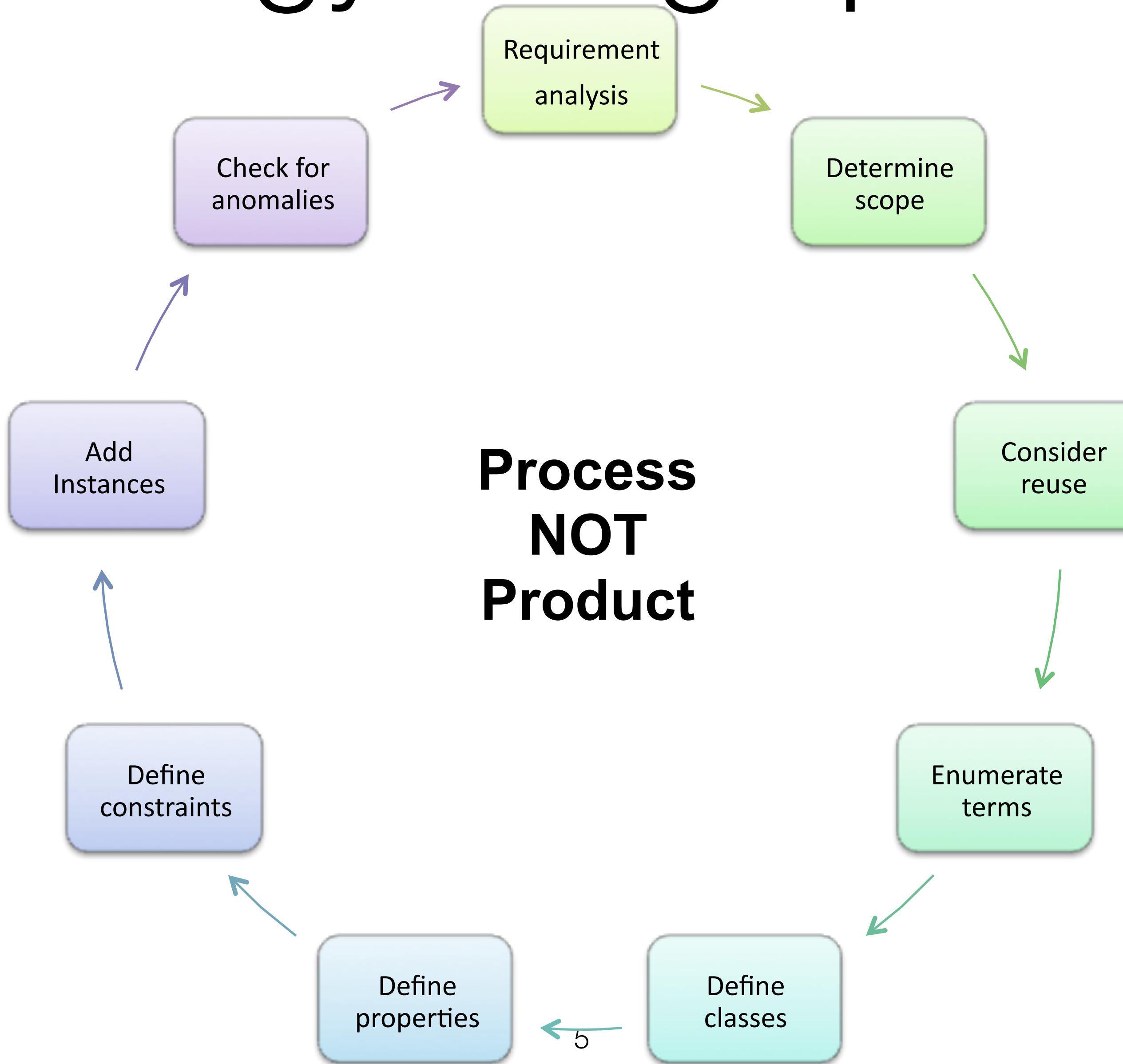


But really more like...

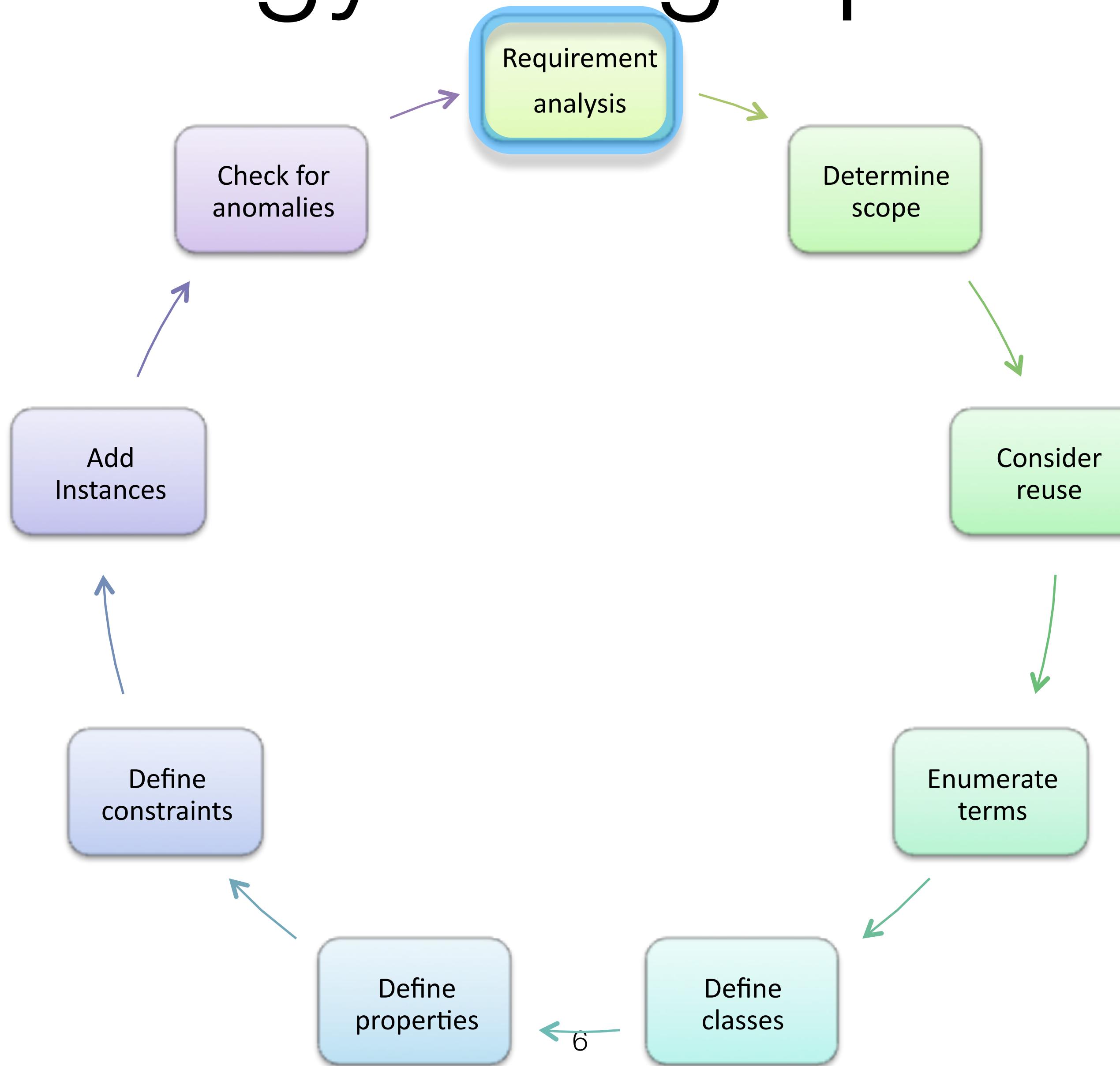
- An iterative Process that repeats continuously and improves the ontology
 - there are always different approaches for modelling an ontology
 - in practice the designated application decides about the modelling approach



Ontology design process



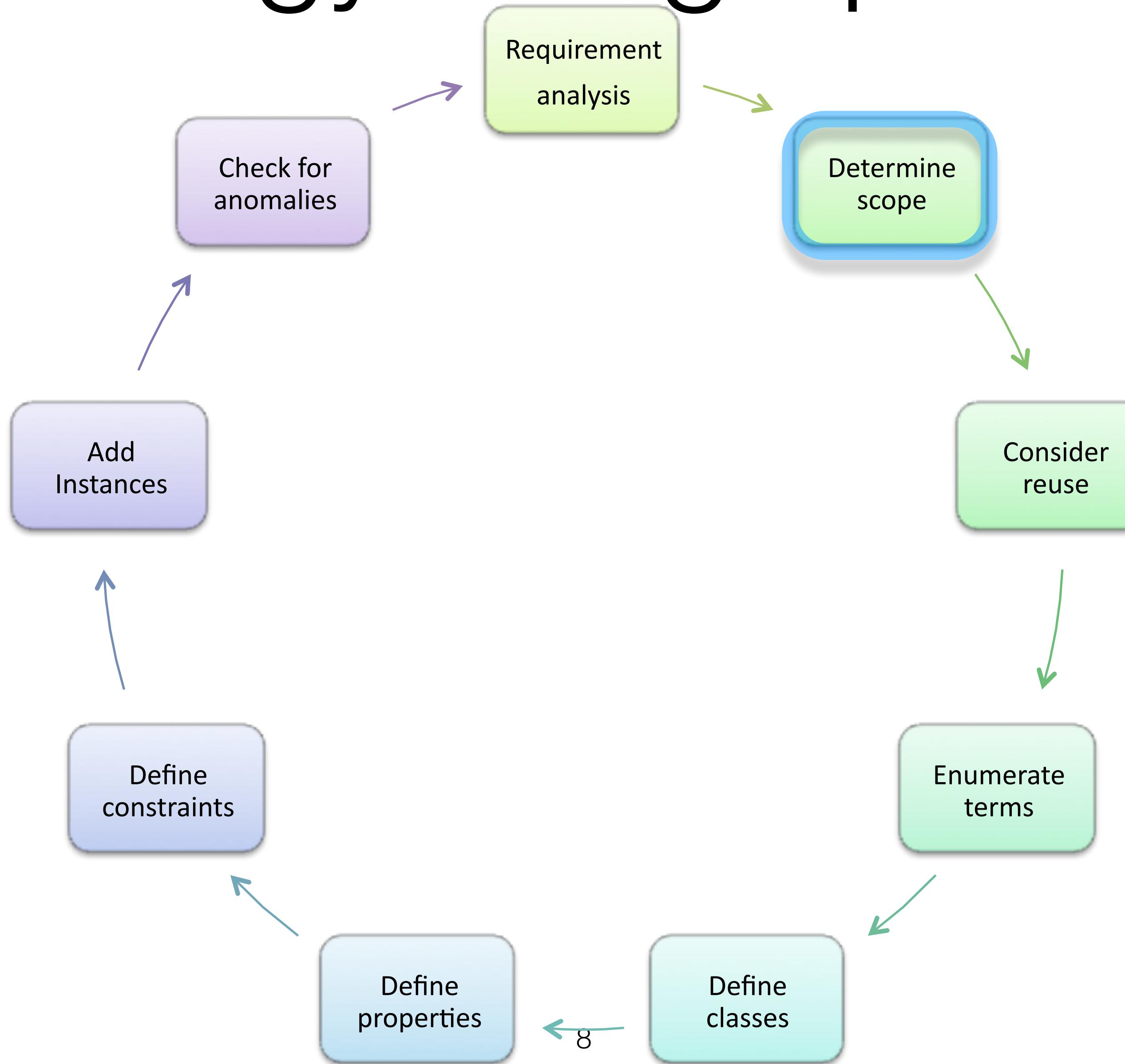
Ontology design process



Requirement analysis

- Requirements, Domain & Use Case Analysis are critical phases as in any software engineering design.
 - they allows ontology engineers to ground the work and prioritise.
- The analysis has to elicit and make explicit:
 - The nature of the knowledge and the questions (competency questions) that the ontology (through a reasoner) needs to answer;
 - *This process is crucial for scoping and designing the ontology, and for driving the architecture;*
 - Architectural issues;
 - The effectiveness of using traditional approaches with knowledge intensive approaches;

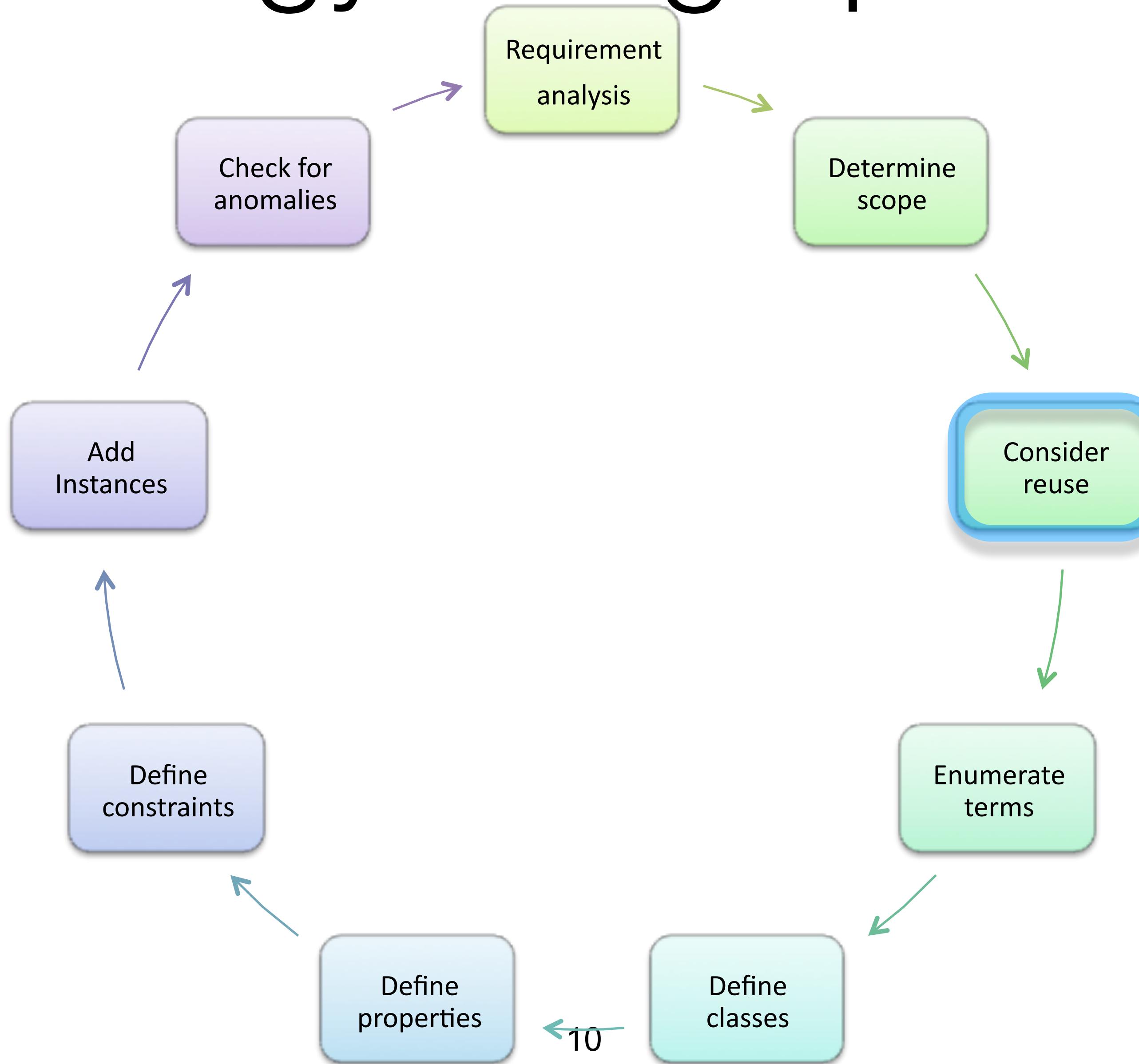
Ontology design process



Determine ontology scope

- There is no correct ontology of a specific domain
 - An ontology is an abstraction of a particular domain, and there are always viable alternatives
- What is included in this abstraction should be determined by
 - the use to which the ontology will be put
 - by future extensions that are already anticipated
- Addresses straight forward questions such as:
 - What is the ontology going to be used for
 - How is the ontology ultimately going to be used by the software implementation?
 - What do we want the ontology to be aware of?
 - What is the scope of the knowledge we want to have in the ontology?

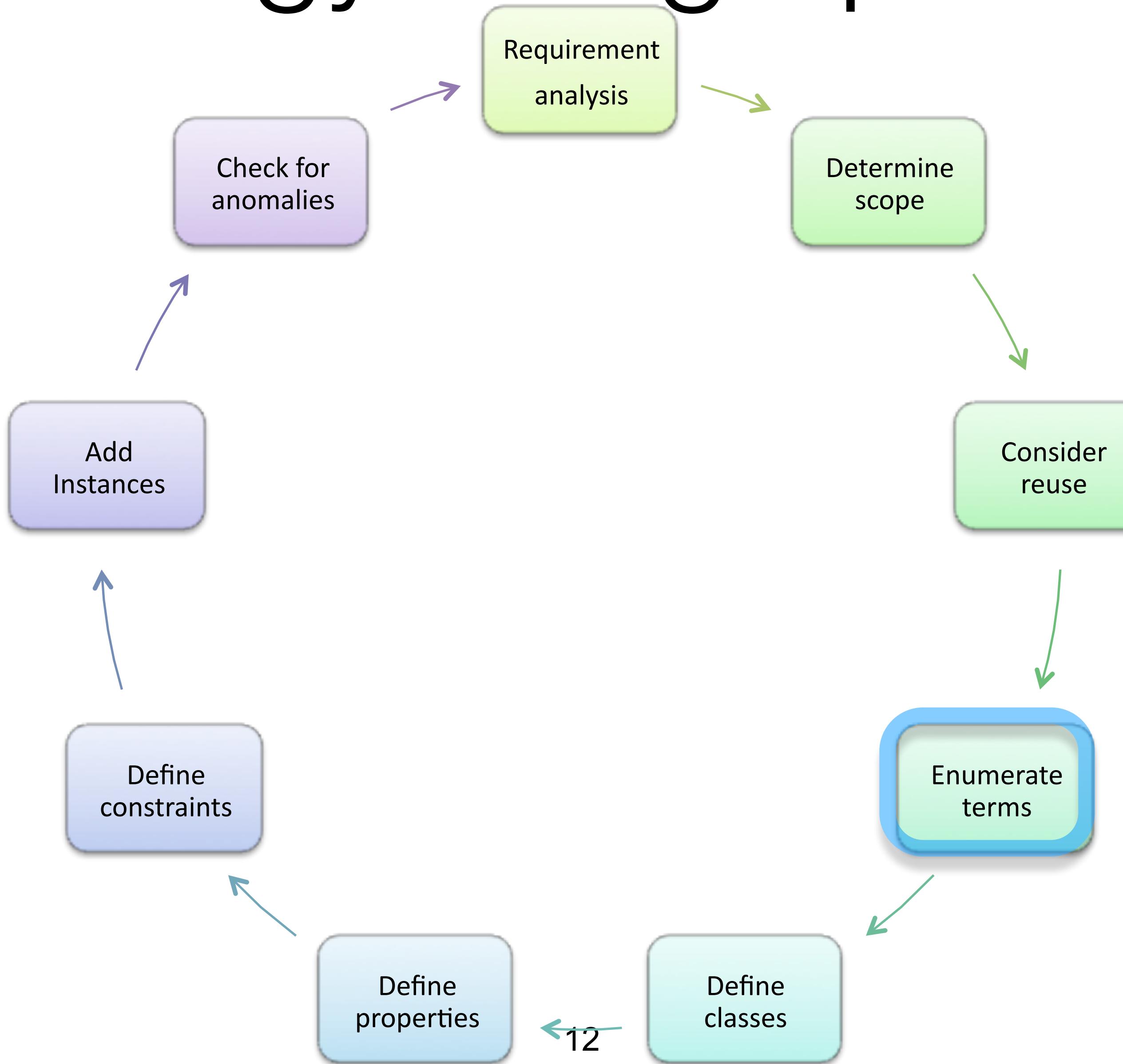
Ontology design process



Consider Reuse

- We rarely have to start from scratch when defining an ontology:
 - There is almost always an ontology available from a third party that provides at least a useful starting point for our own ontology
- Reuse allows to:
 - to save the effort
 - to interact with the tools that use other ontologies
 - to use ontologies that have been validated through use in applications:
- standard vocabularies are available for most domains, many of which are overlapping
- Identify the set that is most relevant to the problem and application issues
- A component-based approach based on modules facilitates dealing with overlapping domains:
 - Reuse an ontology module as one would reuse a software module
 - Standards, complex relationships are defined such that term usage and overlap is unambiguous and machine interpretable

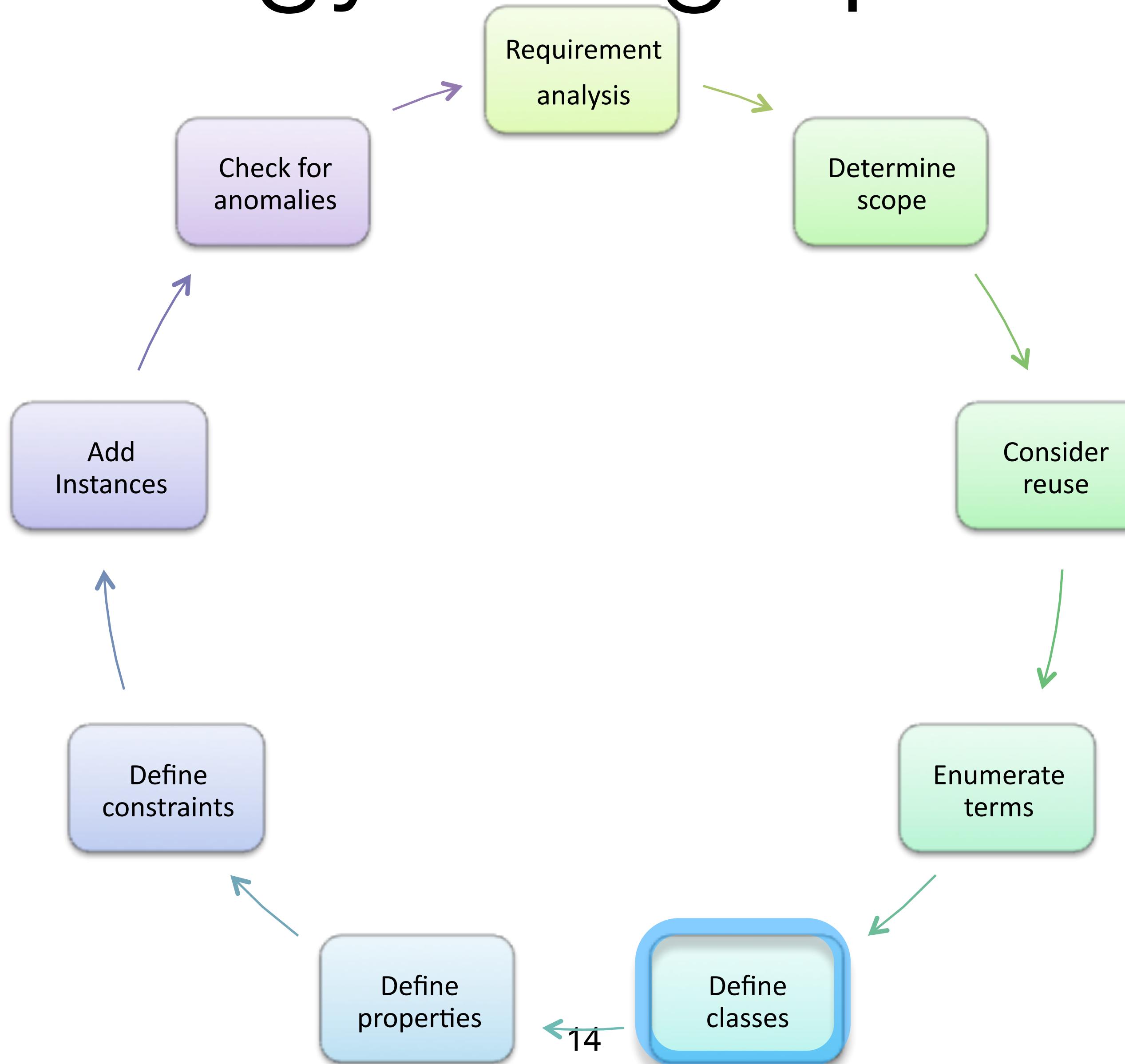
Ontology design process



Enumerate terms

- Write down in an unstructured list all the relevant terms that are expected to appear in the ontology
 - Nouns form the basis for class names
 - Verbs (or verb phrases) form the basis for property names
- Card sorting is often the best way:
 - Write down each concept/idea on a card
 - Organise them into piles
 - Link the piles together
 - Do it again, and again
 - Works best in a small group

Ontology design process



Define classes and their taxonomy

- A class is a concept in the domain:
 - Animal
 - cow, cat, fish
 - A class of properties
 - father, mother
- A class contains necessary conditions for membership
 - type of food, dwelling
- A class is a collection of elements with similar properties
- Instances of classes
 - A particular farm animal, a particular person
 - Tweety the penguin

How do we establish the taxonomy

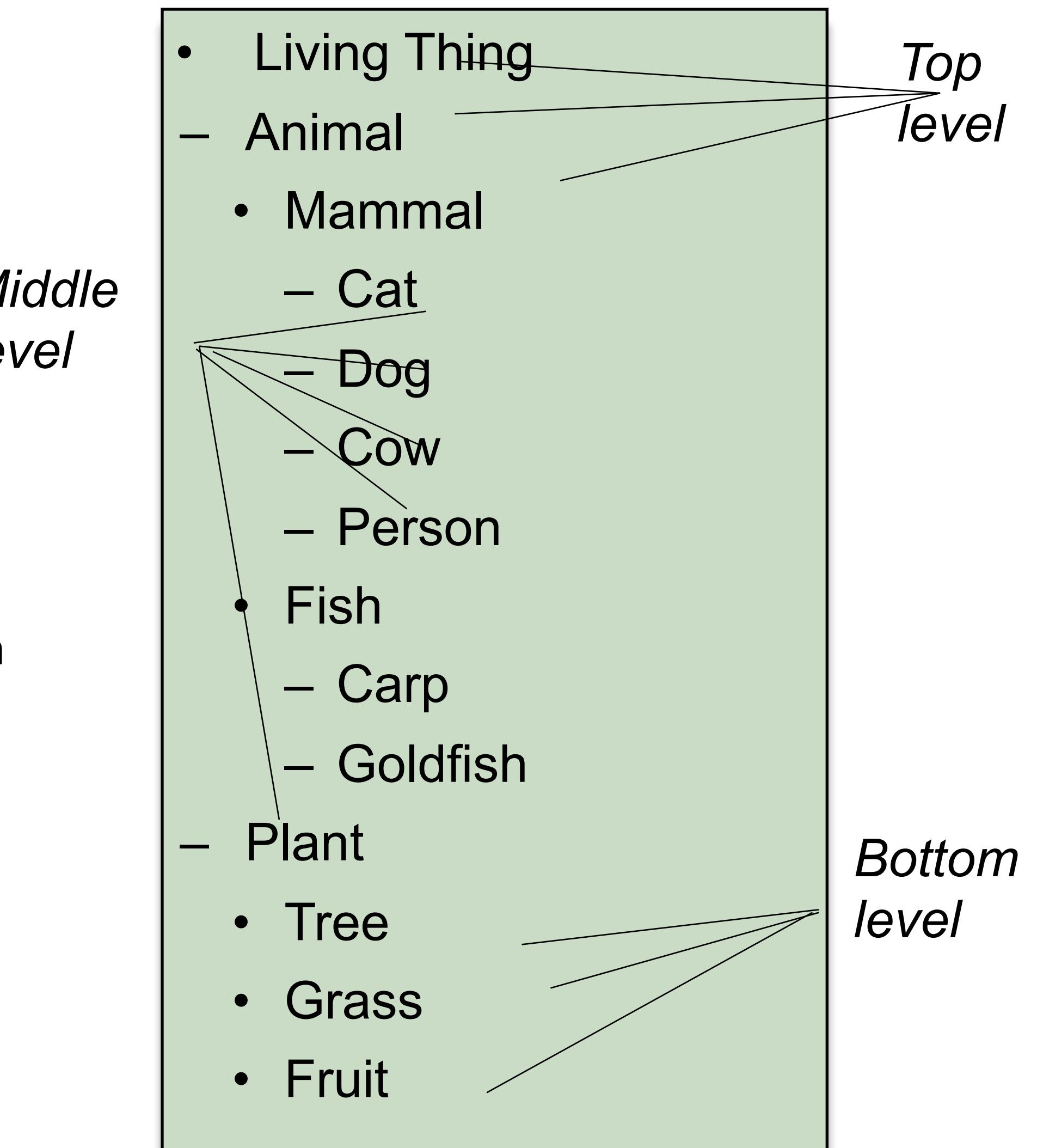
- Relevant terms must be organised in a taxonomic hierarchy
 - Choose some main axes:
 - add **abstractions** where needed
 - Identify **relations**
 - Identify **definable things**
 - e.g. *Father is an animal who has children, Herbivore, is an animal who only eats grass...*
 - Not everything is definable, “natural kinds” cannot be defined as precisely in terms of properties or constraints
 - a cat is....?

How do we establish the taxonomy

- Relevant terms must be organised in a taxonomic hierarchy
 - Distinguish between ***self standing*** things vs ***modifiers***:
 - ***Self standing*** things exist in their own right
 - typically indicated by nouns, e.g. cat, people, animal, action, process...
 - ***Modifiers*** modify other entities
 - typically denoted by adjectives and adverbs, e.g. wild vs domestic, male vs female, healthy vs sick, etc
- Make sure self-standing terms, modifiers and relations are represented in pure trees
 - no multiple inheritance!
 - these will become the “primitive” concepts from which all other concepts can be defined
 - no definable things

Levels in the class hierarchy

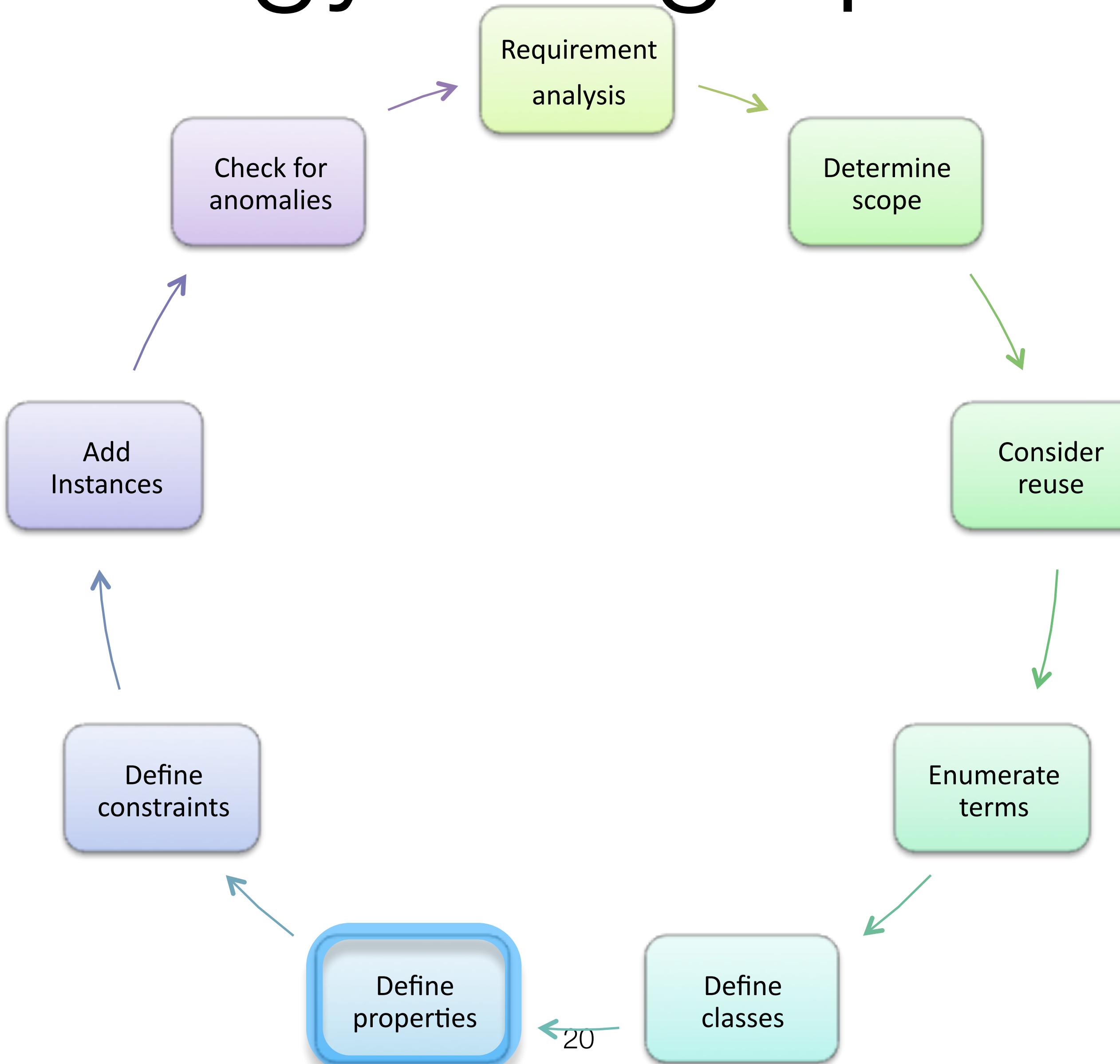
- Different modes of development
 - Top-down
 - define the most general concepts first and then specialize them
 - Bottom-up
 - define the most specific concepts and then organize them in more general classes
 - Combination (typical)
 - breadth at the top level and depth along a few branches to test design
- But... there is no single correct hierarchy
 - guidelines helps us to identify the correct ones
 - Criteria: ***is each instance of the subclass also an instance of the superclass?***



More criteria

- All the siblings must denote concepts at the **same level** of generality
 - *similar to sections and subsections in a book*
- If a class has more than a dozen direct superclasses, then an additional level of generality is needed
 - *compare to bullets in a bullet list*
 - But in some cases, if no natural classification exist, a long list might reflect the reality better.
- Class names should be either singular or plural, don't mix!
 - *Animal is not a kind-of Animals*
- Classes represent concepts in the domain, but names do not
 - a class name can change but the concept represented will still be the same
 - Synonym names for the same concepts refer to different labels, not to different classes

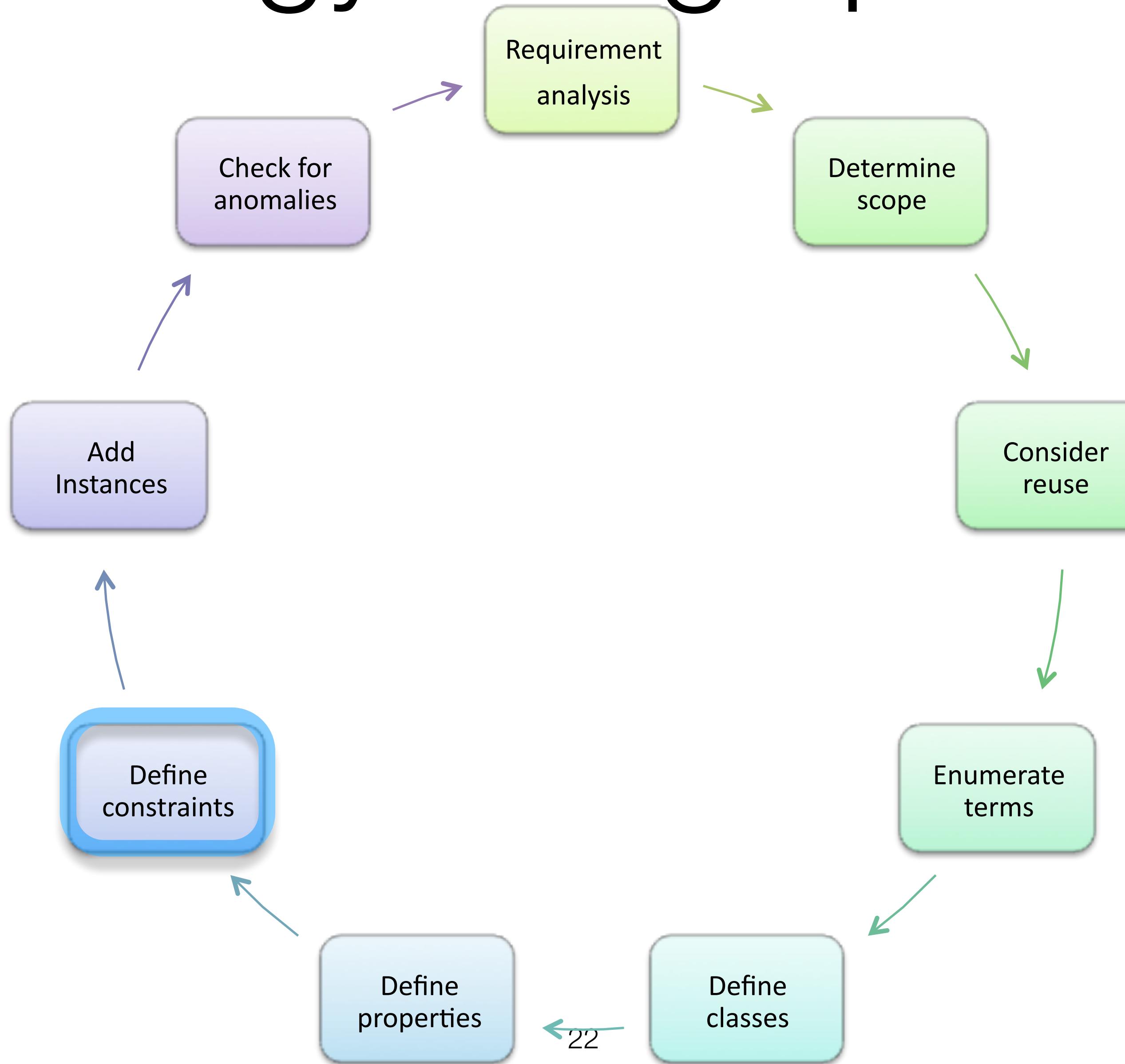
Ontology design process



Define properties

- Often interleaved with the previous step
- Properties (or roles in DL) describe the attributes of the members of a class
 - Defined in terms of domain and range constraints
 - if anything is used in a special way, then add comments
 - *Animal eat LivingThing*, domain: Animal - range: LivingThing
 - *Person owns LivingThing except Person*, domain: Person - range: LivingThing and not Person
 - *Animal parentOf Animal*, domain: Animal - range: Animal
- Defined in terms of property restrictions
 - What can we say about all instances of a class?
 - *all Cows eat some Plants*
 - *all Cats eat some Animals*
 - *all Pigs eat some Animals and eat some Plants*
- For the semantics of subClassOf whenever A is a subclass of B, every property statement that holds for instances of B must also apply to instances of A
 - It makes sense to attach properties to the highest class in the hierarchy to which they apply

Ontology design process



State constraints: definable things

- Definitions need to be **paraphrased** and **formalised** in terms of primitive classes, relations and other definable entities
- Add comments when providing definitions
 - Note any assumptions that need to be represented somewhere else.
 - Paraphrasing needs to achieve consensus on what we meant to represent and how we represent it.

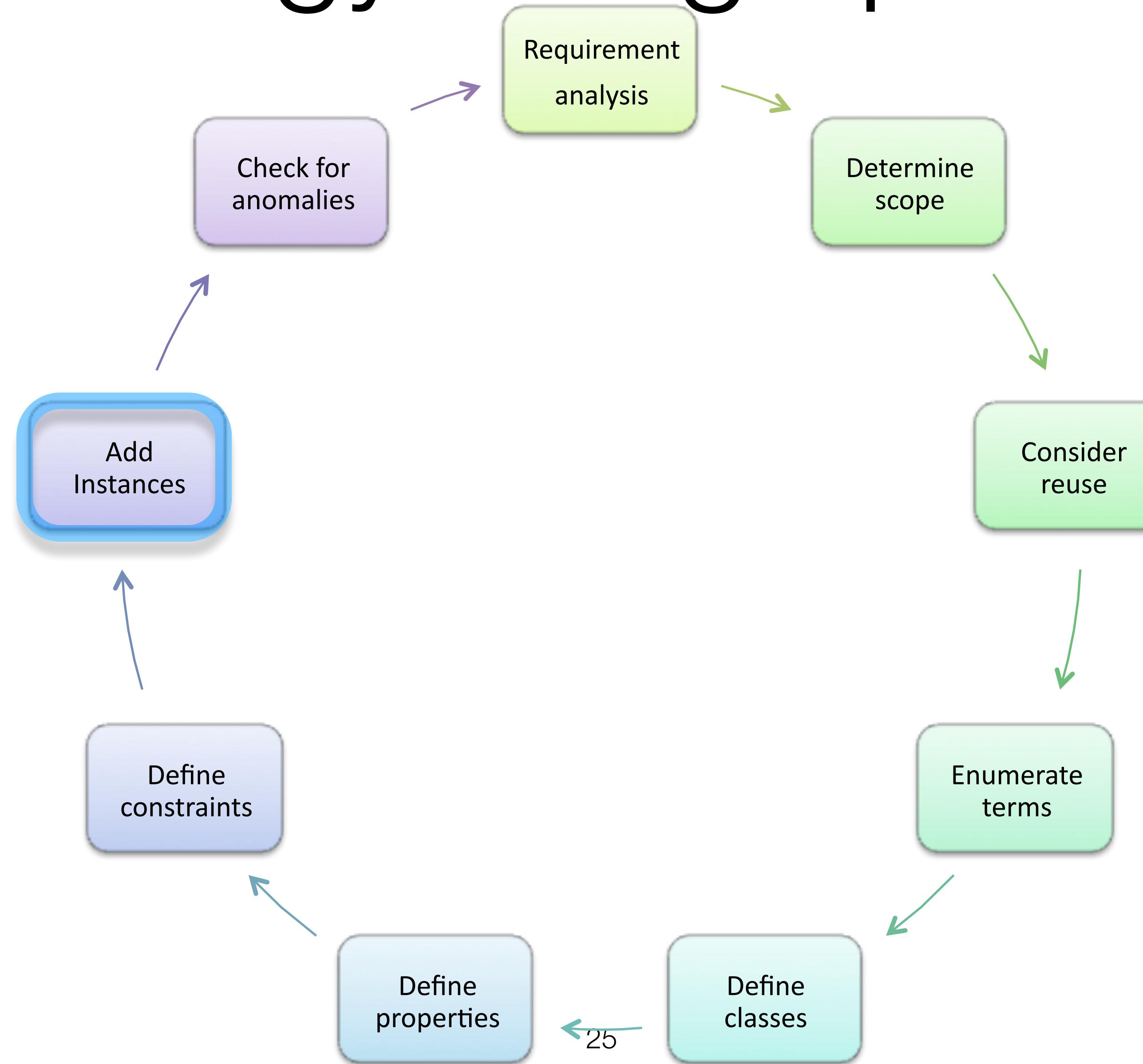
```
:Parent owl:equivalentClass [  
    rdf:type owl:Class;  
    owl:intersectionOf (:Animal [  
        owl:Restriction ;  
        :hasChild ;  
        :Animal . ])  
] .
```

```
:Herbivore owl:equivalentClass [  
    rdf:type owl:Class;  
    owl:intersectionOf (:Animal [  
        owl:Restriction ;  
        :eats ;  
        /* eats range LivingThing */  
        owl:allValuesFrom :Plant . ])  
] .
```

State constraints: definable things

- A **Parent** is an **Animal** that is a parent of some **Animal**
 - Parent = Animal and parentOf some Animal
- A **Herbivore** is an **Animal** that eats only **Plants**
 - assume that Animals eat some LivingThing
 - Herbivore ≡ Animal and eats only Plant
- An **Omnivore** is an **Animal** that eats both **Plants** and **Animals**
 - Omnivore ≡ Animal and eats some Plant and eats some Animal

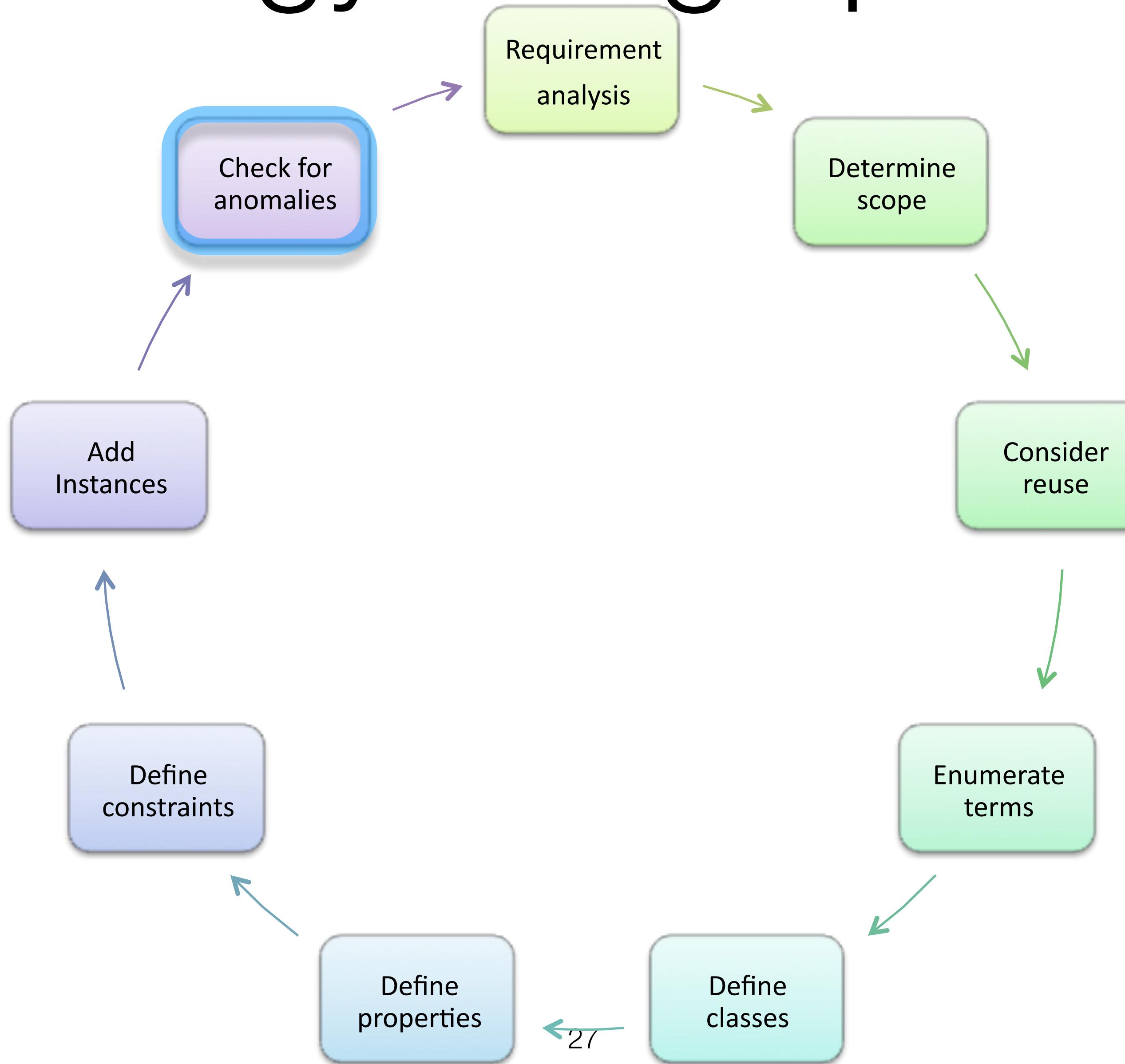
Ontology design process



Creating instances

- Create an instance of a class
 - The class becomes a direct type of the instance
 - Any superclass of the direct type is a type of the instance
- Assign property values for the instance description
 - property values should conform to the constraints asserted for the property
 - Knowledge-acquisition tools often check that constraints are satisfied

Ontology design process



Check for anomalies

- An important advantage of the use of OWL over RDF Schema is the possibility to detect inconsistencies
 - In ontology
 - incoherent ontology: at least an unsatisfiable class, class that cannot have any instance
 - In ontology+instances
 - inconsistent ontology: every class is interpreted as the empty set
- Examples of common inconsistencies
 - incompatible domain and range definitions for transitive, symmetric, or inverse properties
 - cardinality properties
 - requirements on property values can conflict with domain and range restriction
- Examples from the Pizza tutorial for Protege
 - <http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/>

Ontology representation in OWL

- We use turtle syntax for the examples
- Class definition: SubClassOf vs EquivalentClasses
 - All PizzaMargherita have, amongst other things, some mozzarella topping and some tomato topping

```
:Margherita rdf:type owl:Class ;
    rdfs:subClassOf :NamedPizza ,
        [ rdf:type owl:Restriction ;
            owl:onProperty :hasTopping ;
            owl:someValuesFrom :TomatoTopping
        ] ,
        [ rdf:type owl:Restriction ;
            owl:onProperty :hasTopping ;
            owl:someValuesFrom :MozzarellaTopping
        ] ;
```

Ontology representation in OWL

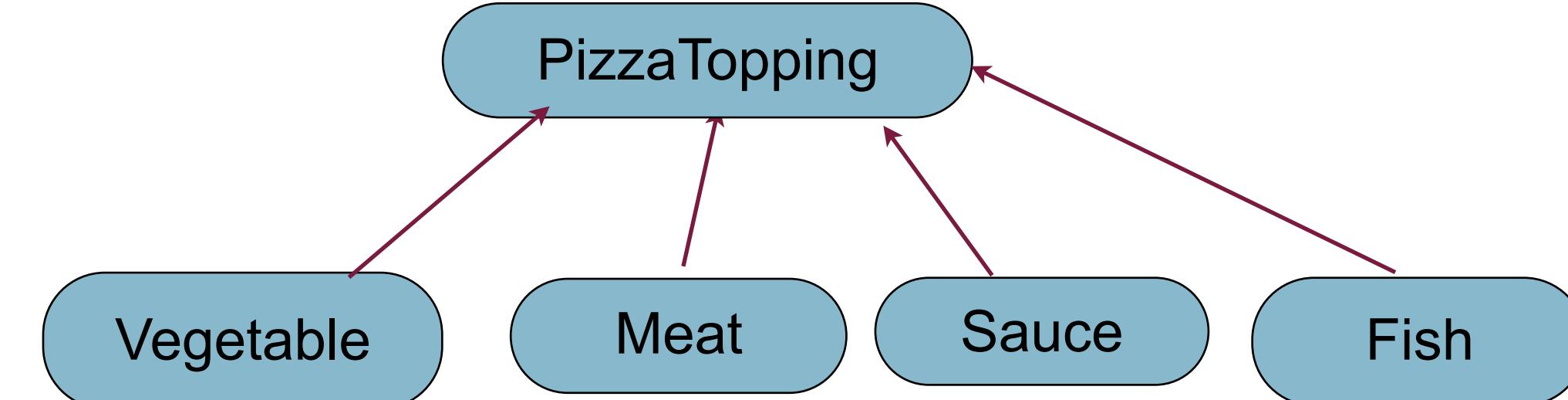
- Class definition: SubClassOf vs EquivalentClasses
 - A MeatyPizza is any pizza that has, amongst other things, at least one meat topping

```
:MeatyPizza rdf:type owl:Class ;
    owl:equivalentClass [ rdf:type owl:Class ;
        owl:intersectionOf (
            :Pizza
            [ rdf:type owl:Restriction ;
                owl:onProperty :hasTopping ;
                owl:someValuesFrom :MeatTopping
            ]
        )
    ] ;
```

Ontology representation in OWL

- Disjointness
 - States that all disjoint classes belong to different branches in the ontology tree!
 - Classes can overlap unless a disjointness axiom is added.

```
:MeatTopping rdf:type owl:Class ;  
    rdfs:subClassOf :PizzaTopping ;  
    owl:disjointWith :FishTopping ,  
        :SauceTopping ,  
        :VegetableTopping .
```



Ontology representation in OWL

- ***Existential property restriction: someValuesFrom***

- Can be used to declare primitive or defined classes
- *Indicates some or at least one*
 - *A cheeseyPizza is any pizza that has some (at least one) cheese topping*

```
:CheeseyPizza rdf:type owl:Class ;
    owl:equivalentClass
        [ rdf:type owl:Class ;
            owl:intersectionOf ( :Pizza
                [ rdf:type owl:Restriction ;
                    owl:onProperty :hasTopping ;
                    owl:someValuesFrom :CheeseTopping
                ]
            )
        ] ;
```

Ontology representation in OWL

- ***Existential property restriction: someValuesFrom***

- Can be used to declare primitive or defined classes
- *Indicates some or at least one*
 - *CheeseyPizza are pizza and have some (at least one) cheese topping*

```
:CheeseyPizza rdf:type owl:Class ;
    owl:subClassOf
        [ rdf:type owl:Class ;
            owl:intersectionOf ( :Pizza
                [ rdf:type owl:Restriction ;
                    owl:onProperty :hasTopping ;
                    owl:someValuesFrom :CheeseTopping
                ]
            )
        ] ;
```

Understanding restrictions

- ***Understanding OWL requires some understanding of how DL models the world.***
 - In DL we often use subsumption to model classes
 - OWL uses the rdfs:subClassOf for representing subsumption.
 - Suppose we want to state the cheesy toppings have some ingredient "cheese" as their "main ingredient".
 - "*Cheesy topping is a subclass of all things that have as main ingredient some cheese.*"

```
:CheeseTopping
  a owl:Class ;
  rdfs:subClassOf
    [ a owl:Restriction ;
      owl:onProperty :mainIngredient ;
      owl:someValueFrom Cheese ].
```

Understanding restrictions

- All cheesy toppings form a subset of all things that have as main ingredient some cheese.
 - If you omit the subClassOf construct you can read it as a UML class with an attribute mainIngredient and a value type constraint for the attribute.
- The subclass relation is essential for understanding the semantics of the restriction:
 - it is a necessary but not sufficient condition for the class.
 - There might be things that have cheese as main ingredient but are not a cheesy topping, hence the subset/subclass definition.

Understanding restrictions

- The definition states that the set of CheeseTopping things is exactly the same as the class of things that have as main ingredient at least one thing that is cheese.
- These are also sometimes called "defined classes";
- Classes declared with just necessary conditions are sometimes called "primitive classes".

```
:cheeseTopping
  a owl:Class ;
  owl:EquivalentTo
    [ a owl:Restriction ;
      owl:onProperty :mainIngredient ;
      owl:someValueFrom Cheese ].
```

Ontology representation in OWL

- Existential property restriction: someValuesFrom
 - Can be used to declare primitive or defined classes
 - Indicates some or at least one
 - A cheeseyPizza is any pizza that has some (at least one) cheese topping

```
:CheeseyPizza rdf:type owl:Class ;
    owl:equivalentClass
        [ rdf:type owl:Class ;
        owl:intersectionOf ( :Pizza
            [ rdf:type owl:Restriction ;
            owl:onProperty :hasTopping ;
            owl:someValuesFrom :CheeseTopping
            ]
        )
    ] ;
```

Ontology representation in OWL

- Existential property restriction: someValuesFrom
 - Can be used to declare primitive or defined classes
 - Indicates some or at least one
 - CheeseyPizza are pizza and have some (at least one) cheese topping

```
:CheeseyPizza rdf:type owl:Class ;
    owl:subClassOf
        [ rdf:type owl:Class ;
            owl:intersectionOf ( :Pizza
                [ rdf:type owl:Restriction ;
                    owl:onProperty :hasTopping ;
                    owl:someValuesFrom :CheeseTopping
                ]
            )
        ] ;
```

Ontology representation in OWL

- Universal property restriction: allValuesFrom
 - Can be used to declare primitive or defined classes
 - Indicates only or no values except
 - A thin and crispy pizza is any pizza where the base is only a thin and crispy base

```
:ThinAndCrispyBase rdf:type owl:Class ;
    rdfs:subClassOf :PizzaBase .

:ThinAndCrispyPizza rdf:type owl:Class ;
    owl:equivalentClass
        [ rdf:type owl:Class ;
        owl:intersectionOf (
            :Pizza [ rdf:type owl:Restriction ;
            owl:onProperty :hasBase ;
            owl:allValuesFrom :ThinAndCrispyBase ]
        )
    ] .
```

Ontology representation in OWL

- Universal property restriction: allValuesFrom
 - Can be used to declare primitive or defined classes
 - Indicates only or no values except
 - All thin and crispy pizza have a pizza whose base is a thin and crispy base

```
:ThinAndCrispyBase rdf:type owl:Class ;
                     rdfs:subClassOf :PizzaBase .

:ThinAndCrispyPizza rdf:type owl:Class ;
                     owl: subclassOf
                     [ rdf:type owl:Class ;
                     owl:intersectionOf (
                     :Pizza [ rdf:type owl:Restriction ;
                     owl:onProperty :hasBase ;
                     owl:allValuesFrom :ThinAndCrispyBase ]
                     )
                     ] .
```

Ontology representation in OWL

- Boolean Combinations
 - Union (disjunction)

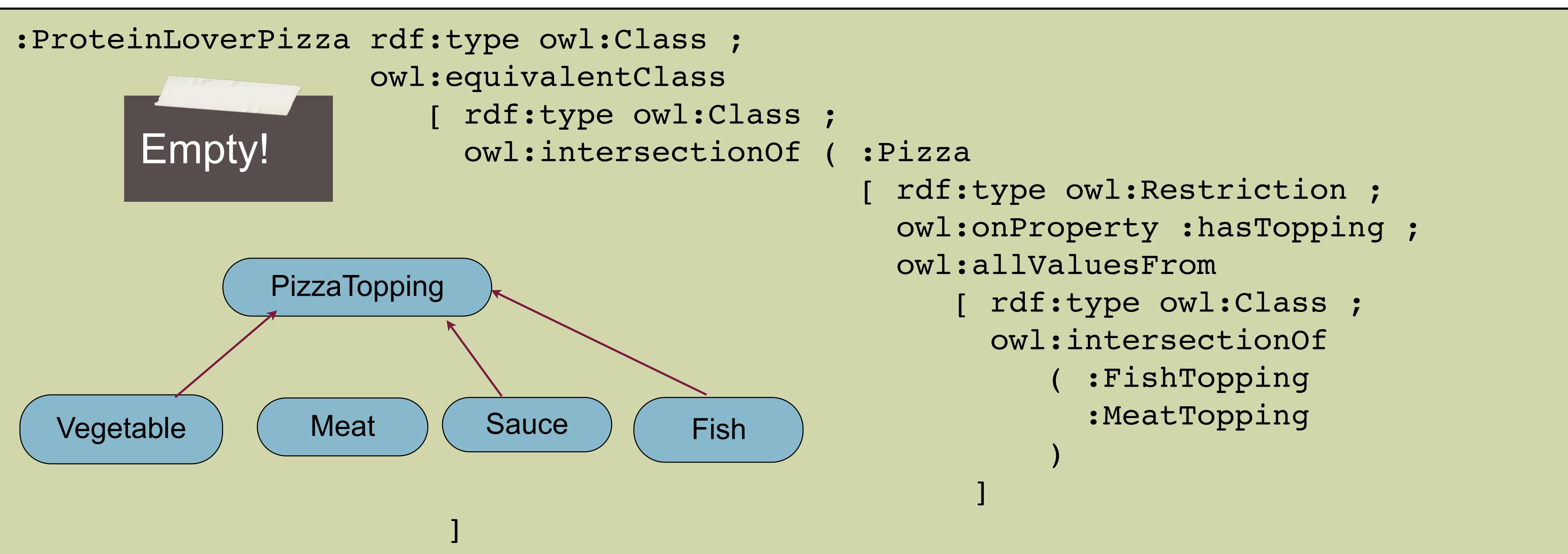
- A vegetarian pizza is any pizza which, amongst other things, has only vegetable and/or cheese toppings

```
:VegetarianPizza rdf:type owl:Class ;
    owl:equivalentClass
        [ rdf:type owl:Class ;
        owl:intersectionOf ( :Pizza
            [ rdf:type owl:Restriction ;
            owl:onProperty :hasTopping ;
            owl:allValuesFrom
                [ rdf:type owl:Class ;
                owl:unionOf ( :CheeseTopping
                    :VegetableTopping
                )
            ]
        )
    ]
```

Ontology representation in OWL

- Boolean Combinations
 - Intersection (conjunction)

- A ProteinLover's is any pizza that has, amongst other things, has only toppings that are both meat and seafood



Ontology representation in OWL

- Other Boolean Combinations
 - complement
 - A non vegetarian pizza is any pizza that is not a vegetarian one

```
:NonVegetarianPizza rdf:type owl:Class ;
    owl:equivalentClass
        [ rdf:type owl:Class ;
            owl:intersectionOf ( :Pizza
                [ rdf:type owl:Class ;
                    owl:complementOf :VegetarianPizza
                ]
            )
        ] ;
    owl:disjointWith :VegetarianPizza ;
```

Reasoning with OWL

- Reasoning with OWL is based on Description Logic reasoning
- The Open world assumption holds:
 - anything might be true unless it can be proven false
 - it states that everything we don't know is undefined
 - no single agent or observer has complete knowledge
- Reasoning is key in:
 - Designing and maintaining good quality ontologies
 - Meaningful: all named classes can have instances
 - Correct: captures intuitions of domain experts
 - Minimally redundant: no unintended synonyms
 - Answer queries, e.g.:
 - Find more general/specific classes
 - Retrieve individuals/tuples matching a given query

Common errors: “Some” does not mean “only”

- All MargheritaPizza have, amongst other things, some mozzarella topping and some tomato topping
 - It is an open world, hence these MargheritaPizza can have other types of toppings, e.g. spicyTopping

```
:Margherita rdf:type owl:Class ;
    rdfs:subClassOf :NamedPizza ,
        [ rdf:type owl:Restriction ;
            owl:onProperty :hasTopping ;
            owl:someValuesFrom :TomatoTopping
        ] ,
        [ rdf:type owl:Restriction ;
            owl:onProperty :hasTopping ;
            owl:someValuesFrom :MozzarellaTopping
        ] ;
```

“Some” does not mean “only”

```
:Margherita rdf:type owl:Class ;
    rdfs:subClassOf :NamedPizza ,
        [ rdf:type owl:Restriction ;
            owl:onProperty :hasTopping ;
            owl:someValuesFrom :TomatoTopping
        ] ,
        [ rdf:type owl:Restriction ;
            owl:onProperty :hasTopping ;
            owl:someValuesFrom :MozzarellaTopping
        ] ,
        [ rdf:type owl:Restriction ;
            owl:onProperty :hasTopping ;
            owl:allValuesFrom [ rdf:type owl:Class ;
                owl:unionOf( :MozzarellaTopping
                    :TomatoTopping
                )
            ]
        ] ,
        owl:disjointWith :Mushroom ,
            :Napoletana ,
```

All Margherita Pizza have, amongst other things, some mozzarella topping and some tomato topping and also have **only** mozzarella and/or tomato topping

“Some” does not mean “only”

```
:EmptyPizza rdf:type owl:Class ;  
    rdfs:subClassOf :Pizza ,  
        [ rdf:type owl:Restriction ;  
            owl:onProperty :hasTopping ;  
            owl:someValuesFrom owl:Thing  
        ] ;
```

*All empty pizzas,
amongst other things,
do not have any topping*
*Empty pizza satisfies
the definition of
Vegetarian Pizza,*

```
:VegetarianPizza rdf:type owl:Class ;  
    owl:equivalentClass  
        [ rdf:type owl:Class ;  
            owl:intersectionOf ( :Pizza  
                [ rdf:type owl:Restriction ;  
                    owl:onProperty :hasTopping ;  
                    owl:allValuesFrom  
                        [ rdf:type owl:Class ;  
                            owl:unionOf ( :CheeseTopping  
                                :VegetableTopping  
                            )  
                        ]  
                ]  
            )  
        ]
```

Domain and range constraints

- Domain and range constraints are axioms:
 - All things have no name except *classRange* vs having a name implies being *classDomain*
 - Violating domain and range constraints can give some unwanted results due to reasoning

```
:hasTopping rdf:type owl:InverseFunctionalProperty ,  
             owl:ObjectProperty ;  
  
    rdfs:domain :Pizza ;  
  
    rdfs:range :PizzaTopping ;  
  
    rdfs:subPropertyOf :hasIngredient .
```

Brief set of guidelines

- Always **paraphrase** a description or definition before encoding it in OWL
- Make all primitives **disjoint** (requires that primitives form trees)
- Use **ObjectSomeValuesFrom** as the default quantifier in restrictions
 - Be careful to make defined classes **defined**. The classifier will place nothing (\perp) under a primitive class
 - except in the presence of axioms/ domain/range constraints
- Don't forget the **open world assumption**.
 - Insert closure restrictions if that is what you mean
 - Be careful to model **domain** and **range** constraints.
 - Check them carefully if classification does not work as expected
 - Be careful about the use of “**and**” and “**or**” (intersection and union)
 - Run the classifier frequently; spot errors early.

Summary

- Ontology engineering
 - Ontology 101

COMP318: Ontology based Information Systems

www.csc.liv.ac.uk/~valli/Comp318



Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

v.Tamma@liverpool.ac.uk

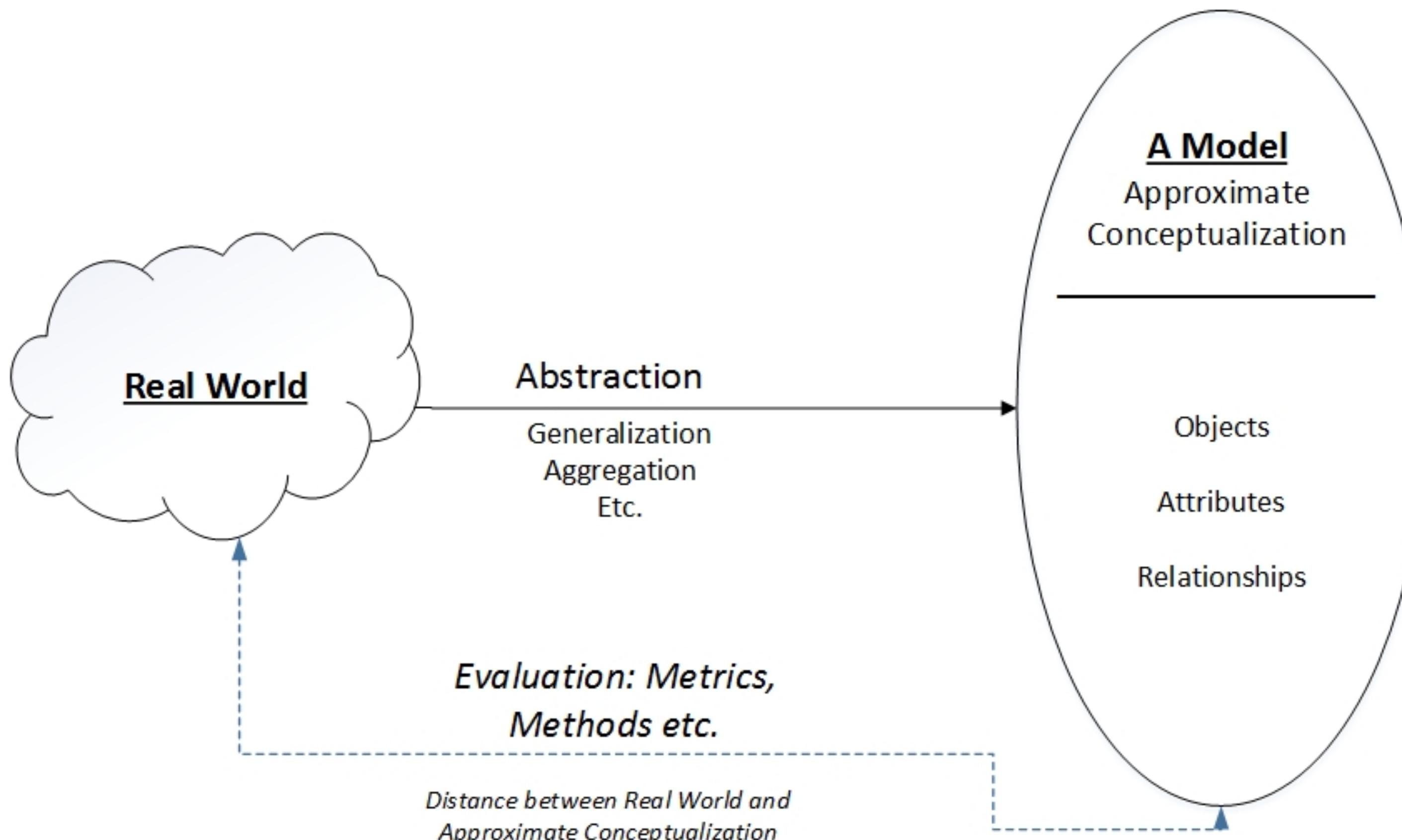
Where were we

- Ontology engineering
 - Ontology 101

Ontology evaluation

- The process of **deciding the quality of an ontology**
 - Two activities: **ontology verification** and **validation**
- **Ontology verification:** checks the encoding of the specification
 - detects errors, as e.g. circular, class hierarchies, redundant axioms, inconsistent naming schemes etc.
 - confirms that the ontology has been built according to certain specified ontology quality criteria
- **Ontology validation:** checks whether the meaning of the definitions matches with the conceptualisation the ontology is meant to specify
 - the goal is to show that the world model is compliant with the formal model

Ontology evaluation



from Hlomani & Stacey: Approaches, methods, metrics, measures and subjectivity in ontology evaluation, ACM 2014

Ontology evaluation criteria

- **Accuracy:**

- Do axioms comply to knowledge elicited from the users?
- Does the ontology capture and represents aspects of the real world correctly?

- **Adaptability:**

- Does the ontology offer the conceptual foundation for a range of anticipated tasks?
- Can the ontology be extended and specialised without the need to remove axioms (*monotonically*)?

- Does the ontology comply to procedures for extension, integration, and adaptation?

- **Clarity:**

- Does the ontology communicate effectively the intended meaning of the defined terms?
- Are the definitions objective and independent of context?
- Does the ontology use definitions or (partial) descriptions?
- Are the definitions documented?
- Is the ontology understandable?

Ontology evaluation criteria

- **Completeness:**
 - Is the domain of interest appropriately covered?
 - Are competency questions defined and can the ontology answer them?
 - Does the ontology include all relevant concepts and their lexical representations?
- **Computational efficiency:**
 - How easy and successfully can reasoners process the ontology?
- How fast can the standard reasoning processes (satisfiability, instance classification, etc.) be applied to the ontology?
- **Conciseness:**
 - Does the ontology include irrelevant axioms?
 - Does the ontology specify the weakest theory possible and define only essential terms?
 - How weak are the assumptions regarding the ontology's underlying philosophical theory about reality?

Ontology evaluation criteria

- **Consistency:**
 - Do the axioms lead to contradictions (logical consistency)?
 - Are formal and informal description of the ontology consistent?
 - Are any representation choices made purely for the convenience of notation or representation?
 - Does the translation from the knowledge level to the encoding show a minimal encoding bias?
- **Organisational Fitness:**
 - Is the ontology easily deployed within the organization?
 - Do tools within the organization put constraints on the ontology?
 - Does the ontology meet legal requirements

Measures for ontology evaluation

- Direct measurement of the criteria in the previous slides is difficult, however ...
- **Ontology Correctness**
 - **Accuracy**, e.g. using **precision** (total number correctly defined concepts over whole knowledge defined in ontology) and **recall** (total number correctly defined concepts over all knowledge that should be defined)
 - **Completeness**, e.g. using coverage of encoded axioms and axioms in specification
- **Consistency**, e.g. count terms with inconsistent meaning
- **Ontology Quality**
 - **Computational Efficiency**, e.g. through size
 - **Adaptability**, e.g. via coupling (number of external classes referenced) and cohesion (number of root, leaf, avg. inheritance depth, etc.)
 - **Clarity**, e.g. number of word senses

COMP318: Ontology alignment

www.csc.liv.ac.uk/~valli/Comp318

Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

V.Tamma@liverpool.ac.uk



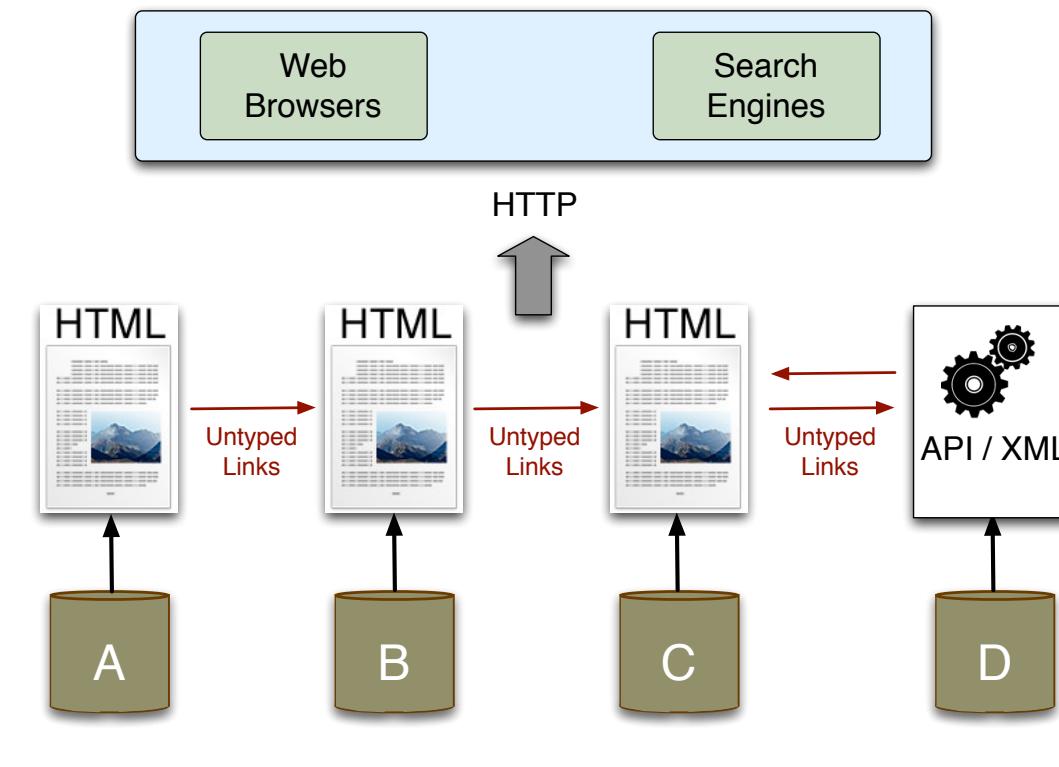
based on slides by V. Tamma, T.R. Payne, J. Euzenat, K. Janowicz and G. Schreiber

Where were we

- Ontology engineering
 - methodology
 - representation in OWL
 - violations and use of reasoning services

The global dataspace

- The Web allowed the creation of a global information space
 - where structured, semi structured and unstructured information is shared
- By 2012 over 3,000 exabytes of data were available (IDC and UC Berkeley)
 - Mainly generated through transactions, but later from interactions
- By the end of 2016, global Internet traffic reached 1.1 zettabytes per year, according to Cisco
 - 1 zettabyte = 1 sextillion bytes, or 1,000 exabytes
- Things are now connected online
 - Sensors, devices, appliances... all publishing data
 - “A cross country flight from New York to Los Angeles on a Boeing 737 plane generates a massive 240 terabytes of data”. GigaOmni Media



IN2 TORRE/CHN

Diversity in models: friend or foe?

- Different systems (sensors, services, applications, agents...) that generate or use knowledge usually make use of different ontologies
 - Similar or overlapping information is modelled in diverse ways even inside organisations with strong governance and internal communication
- These differences in modelling are an obstacle to systems' interoperability

Interoperability example

- You own a company selling digital cameras
 - You organise your information according to your own schema

```
graph TD; Stock[Stock] --> CE[Consumer_Electronics]; Stock --> CP[Cell_Phones]; CE --> PAC[Photo_and_Cameras]; CE --> Nikon[Nikon]; CE --> DSLRs[Digital_SLRs]; Nikon --> DSLRs; DSLRs[Digital_SLRs]; PAC --> Accessories[Accessories]; Accessories --> HFK[Hands_Free_Kits]; Accessories --> Batteries[Batteries]; Accessories --> Cases[Cases]
```
- Task: You want your company to sell in a marketplace, e.g.:
 - Ebay:
 - *Home > Buy > Cameras & Photo > Digital Cameras > Digital SLR > Nikon > D34XX*
 - Amazon marketplace:
 - *Home > Department > Electronic & Computers > Camera & Photo > Digital Cameras > Digital SLR > Nikon > D34XX*
- Mappings between:
 - the entries in your schema
 - to the entries of the common catalogues of the marketplaces

The need for interoperability

- Interoperability measures the extent to which different systems are able to meaningfully exchange information
 - with the aim of reaching some common goal.
- Some (subtle) considerations
 - Data does not interoperate, but is used to support interoperation.
 - (Inter)operability assumes intention and purpose, e.g., a goal.
 - Interoperability is about a degree of meaningful exchange
- Syntactic vs semantic interoperability

The need for interoperability

- **Syntactic interoperability:**

- If two or more systems are capable of communicating with each other, they exhibit syntactic interoperability when using specified data formats and communication protocols.
 - XML or SQL standards are among the tools of syntactic interoperability

- **Semantic interoperability:**

- the ability to automatically interpret the information exchanged meaningfully and accurately in order to produce useful results
 - as defined by the end users of both systems.
- The meaning of the information exchanged is unambiguously defined: what is sent is the same as what is understood.

Standards!

- Standards for representing, publishing, sharing, querying facts, knowledge, data, software and processes



- Provide a scalable approach for the discovery of knowledge that is
 - formulated in different ways, from independent actors
 - distributed physically and logically



Reusing vocabularies

FOAF Core

- [Agent](#)
- [Person](#)
- [name](#)
- [title](#)
- [img](#)
- [depiction](#) ([depicts](#))
- [familyName](#)
- [givenName](#)
- [knows](#)
- [based_near](#)
- [age](#)
- [made](#) ([maker](#))
- [primaryTopic](#) ([primaryTopicOf](#))
- [Project](#)
- [Organization](#)
- [Group](#)
- [member](#)
- [Document](#)

Social Web

- [nick](#)
- [mbox](#)
- [homepage](#)
- [weblog](#)
- [openid](#)
- [jabberID](#)
- [mbox_sha1sum](#)
- [interest](#)
- [topic_interest](#)
- [topic](#) ([page](#))
- [workplaceHomepage](#)
- [workInfoHomepage](#)
- [schoolHomepage](#)
- [publications](#)
- [currentProject](#)
- [pastProject](#)
- [account](#)
- [OnlineAccount](#)
- [accountName](#)
- [accountServiceHomepage](#)
- [PersonalProfileDocument](#)
- [tipjar](#)
- [sha1](#)
- [thumbnail](#)
- [logo](#)

WordNet Search - 3.1

- [WordNet home page](#) - [Glossary](#) - [Help](#)

Word to search for:

Display Options: (Select option to change)

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations
Display options for sense: (gloss) "an example sentence"

Noun

- S: (n) [procedure](#), [process](#) (a particular course of action intended to achieve a result) "the procedure of obtaining a driver's license"; "it was a process of trial and error"
- S: (n) [process](#), [cognitive process](#), [mental process](#), [operation](#), [cognitive operation](#) ((psychology) the performance of some composite cognitive activity; an operation that affects mental contents) "the process of thinking"; "the cognitive operation of remembering"
- S: (n) [summons](#), [process](#) (a writ issued by authority of law; usually compels the defendant's attendance in a civil suit; failure to appear results in a default judgment against the defendant)
- S: (n) [process](#), [unconscious process](#) (a mental process that you are not directly aware of) "the process of denial"
- S: (n) [process](#), [outgrowth](#), [appendage](#) (a natural prolongation or projection from a part of an organism either animal or plant) "a bony process"
- S: (n) [process](#), [physical process](#) (a sustained phenomenon or one marked by gradual changes through a series of states) "events now in process"; "the process of calcification begins later for boys than for girls"

Verb

- S: (v) [process](#), [treat](#) (subject to a process or treatment, with the aim of readying for some purpose, improving, or remedying a condition) "process cheese"; "process hair"; "treat the water so it can be drunk"; "treat the lawn with chemicals"; "treat an oil spill"
- S: (v) [process](#) (deal with in a routine way) "I'll handle that one"; "process a loan"; "process the applicants"
- S: (v) [process](#) (perform mathematical and logical operations on (data) according to programmed instructions in order to obtain the required information) "The results of the elections were still being processed when he gave his acceptance speech"
- S: (v) [action](#), [sue](#), [litigate](#), [process](#) (institute legal proceedings against; file a suit against) "He was warned that the district attorney would process him"; "She actioned the company for discrimination"
- S: (v) [march](#), [process](#) (march in a procession) "They processed into the dining room"

DBpedia version 2016-10

Dataset category: DBpedia release
Publication Year: 2017

This release is based on updated Wikipedia dumps dating from October 2016.

You can download the new DBpedia datasets in N3 / TURTLE serialisation from <http://wiki.dbpedia.org/downloads-2016-10> or directly here <http://downloads.dbpedia.org/2016-10/>.

This release took us longer than expected. We had to deal with multiple issues and included new data. Most notable is the addition of the [NIF](#) annotation datasets for each language, recording the whole wiki text, its basic structure (sections, titles, paragraphs, etc.) and the included text links. We hope that researchers and developers, working on NLP-related tasks, will find this addition most rewarding. The DBpedia [Open Text Extraction Challenge](#) (next deadline Mon 17 July for [SEMANTICS 2017](#)) was introduced to instigate new fact extraction based on these datasets.

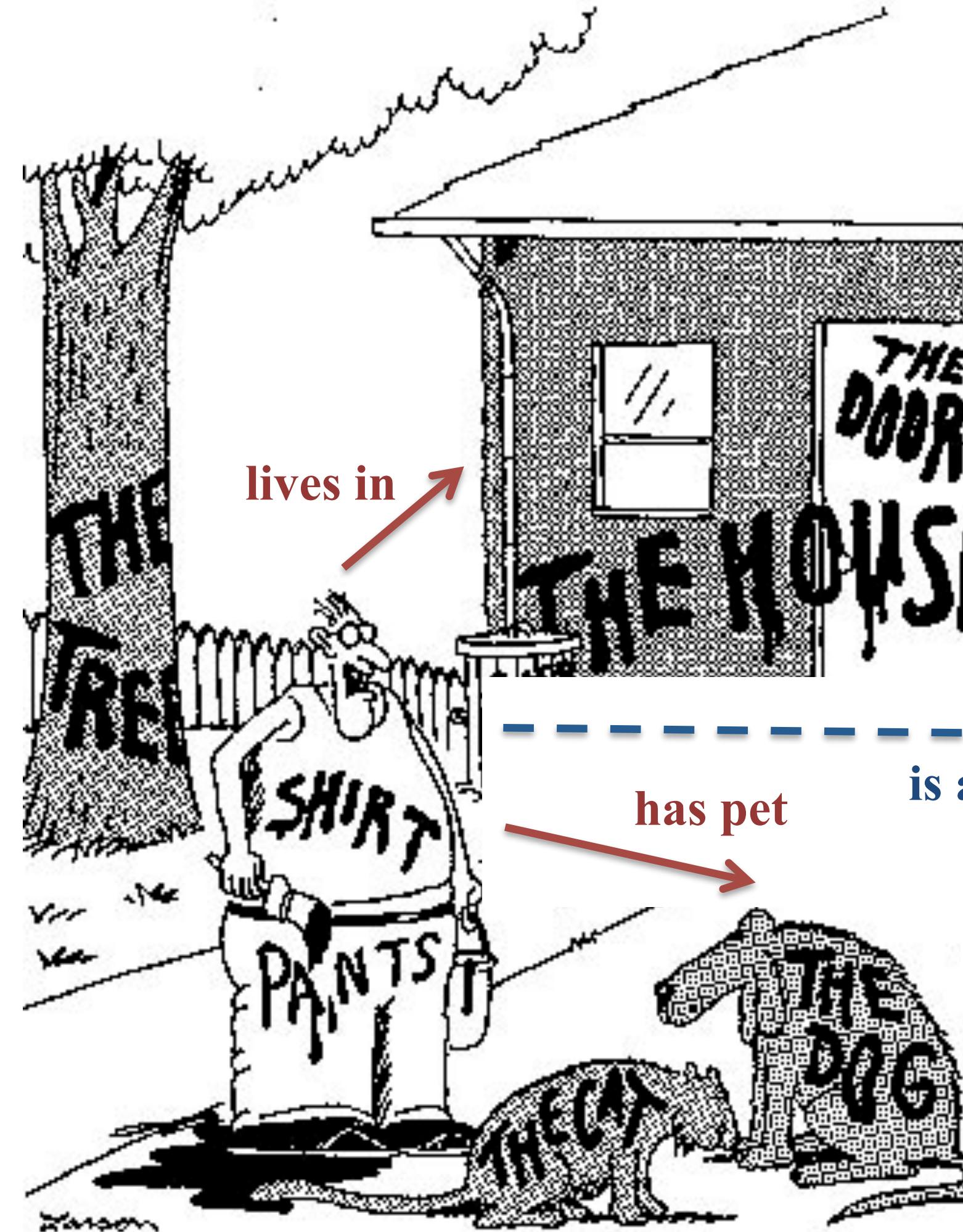
We want to thank anyone who has contributed to this release, by adding mappings, new datasets, extractors or issue reports, helping us to increase coverage and correctness of the released data. The European Commission and the [ALIGNED H2020 project](#) for funding and general support.

Join and support DBpedia

The active community of developers and engineers comes together in the DBpedia Community Committee. We will extend this Committee with the help of Pablo Mendes and Magnus Knuth. Students wishing to join should be or become a member of the [DBpedia Association](#). Please check all benefits and details on our [website](#).

Every first Wednesday of the month we organise regular development online meetings. You can join the next DBpedia dev telco on Wednesday, 5th of July (@ 2 pm CET). All info regarding the telco can be found here: <http://tinyurl.com/DBpediaDevMinutes>.

Semantic Web?

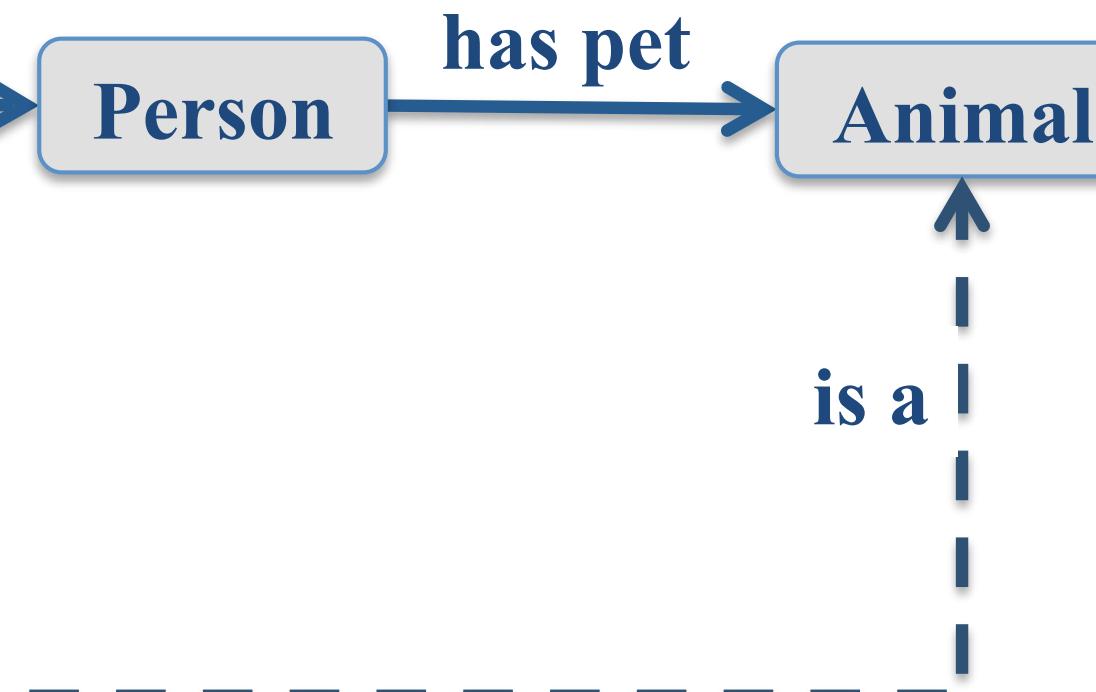


Concrete Facts

Resource Description Framework

General Knowledge

Web Ontology Language



“Now! – That should clear up a few things around here!”

8

slide by F. van Harmelen

What does meaning mean

- Semantics is the study of meaning:
 - the study of how and what symbols denote.
 - Meaning can be seen as an emergent property of interaction.
- Humans use complex processes and negotiate the approximate, common meaning of terms during interaction.
 - contextual cues, gestures...
 - Unfortunately, we cannot do so with data.

What does meaning mean here

- The implied assumption behind the call for collecting raw data is that data can be reused outside its original creation context and is free of interpretation:
 - However, there is no such thing as raw data.
 - Data is always created following particular workflows, procedures, sampling strategies,
 - is derived using specific instrumentation, is pre-processes in specific ways.
- Ontologies support interoperability and integration:
 - they cannot fix meaning,
 - but, they formally restrict the possible interpretations of the terms in a domain to intended meaning.

Myth busting slide

- **Myth 1:** Once we have an ontology data becomes interoperable
 - Actually there are already several ontologies for a domain:
 - Ontology alignment;
 - Reference ontologies;
- **Myth 2:** Semantics makes my data machine-understandable, ergo my system will be intelligent
- **Myth 3:** It's a hype, ontologies and semantics are too much overhead. What about tiny devices?
 - Ontologies are a way to share and agree on a common vocabulary and knowledge in a machine interpretable and reusable format;
 - Semantic metadata does not need to be added in the source, it can be added to the data at a later stage (e.g. in a gateway)
 - Legacy applications can be extended to work with it.
 - And ... “a little semantics goes a long way” (Jim Hendler & Tim Berners Lee)

The Architect

*When modelling a bridge,
important characteristics
include:*

*tensile strength
weight
load
etc*



**Pat Hayes in conversation
with T.R. Payne, 2001**

The Military

*When modelling a bridge,
important characteristics
include:*

*what munitions are
required to destroy it!*

No unified vocabulary

- There is never “the” correct ontology:
 - a number of different ontologies can represent the same domain
 - they all capture different contexts, perspectives, requirements
 - and depend on the task that the ontology should support
 - performed by some (autonomous) system — agent / service / API / ...

- These differences in modelling become apparent when

- These systems must be combined (**integration**)
- Or be made to work together (**interoperation**)

Semantic integration

- **Semantic Integration** is often treated as a high level cognitive task
 - even when the associated computational artefacts are low level
- Ontologies are a computational representation of the cognitive level, and are the underlying basis for automating semantic data integration.
- **Ontology alignment** is the process of determining correspondences between semantically related entities
 - classes, relationships and instances

Aligning ontologies

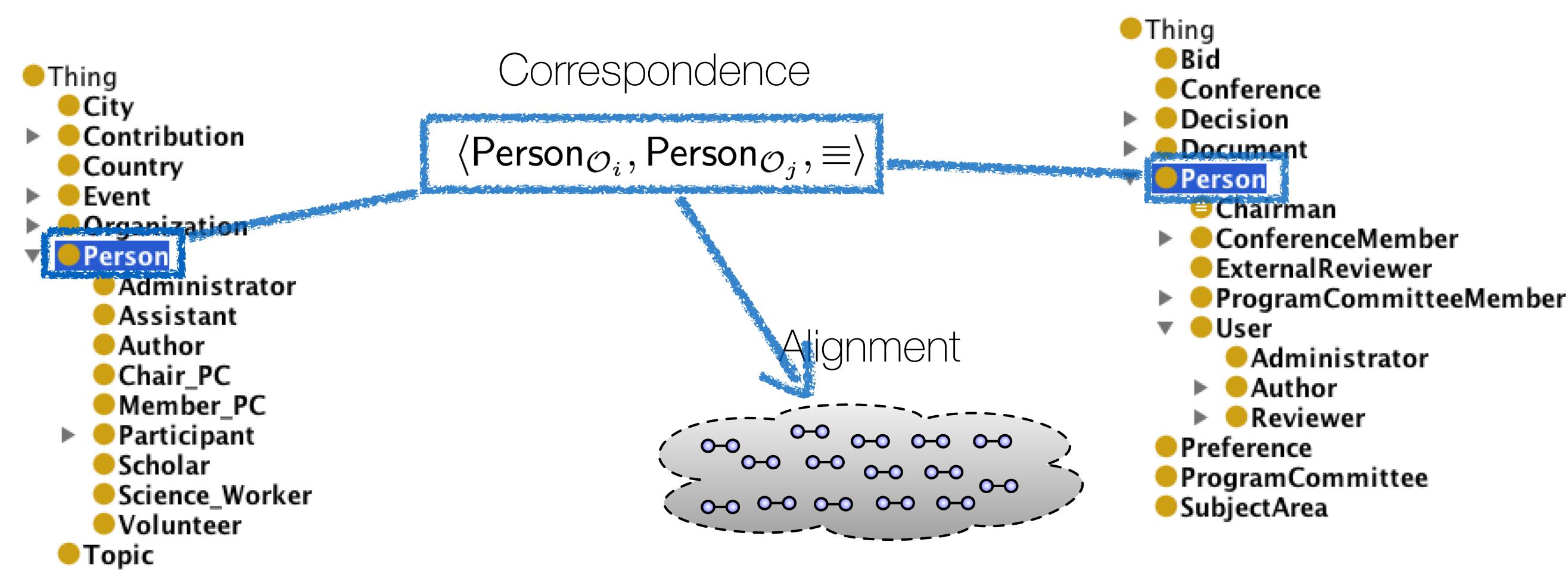
- Ontology alignment or mapping:
 - Use ontologies together by defining a set of “links” (mappings or correspondences)
 - Mappings can be of limited types, i.e. only certain logical relations
- Advantages:
 - Benefit from knowledge encoded in the other ontologies models
 - Enable access from different agents/services and across different collections
 - Partial by nature, does not need to cover the entire ontology

Alignment approaches

- Different alignment approach are available depending on
 - the expressivity of the two ontologies O and O'
 - The availability of additional inputs to the matching process:
 - Oracles, input alignment and external resources, i.e. Wordnet or BabelNet
 - The entities to match:
 - Only the T-box or schema, i.e. classes and possibly properties
 - Instances
- The majority of current ontology alignment systems align classes, and restrict the relationships to equivalence

How to represent a correspondence

- Given two ontologies O and O' , an alignment \mathcal{A} is the set of correspondences c between the entities $e \in O$ and $e' \in O'$
 - A correspondence c is the tuple $c = \langle e, e', r, w \rangle$
 - $e \in O$ and $e' \in O'$, where e and e' can be classes, properties, individuals
 - $r = \{\equiv, \sqsubseteq, \perp\}$ and $w \in [0, \dots, 1]$ is the weight



Types of correspondence relation between classes/ properties

	OWL	Example
\equiv Equivalence	<code>owl:EquivalentClass</code>	<code>O:Person ≡ O':Person</code>
\sqsubseteq Subclass	<code>rdfs:subClassOf</code> <code>rdfs:subPropertyOf</code>	<code>O:Assistant ⊑ O':User</code>
\perp Disjointness	<code>owl:disjointWith</code> , <code>owl:allDisjointClasses</code>	<code>O:Topic ⊥ O':Person</code>

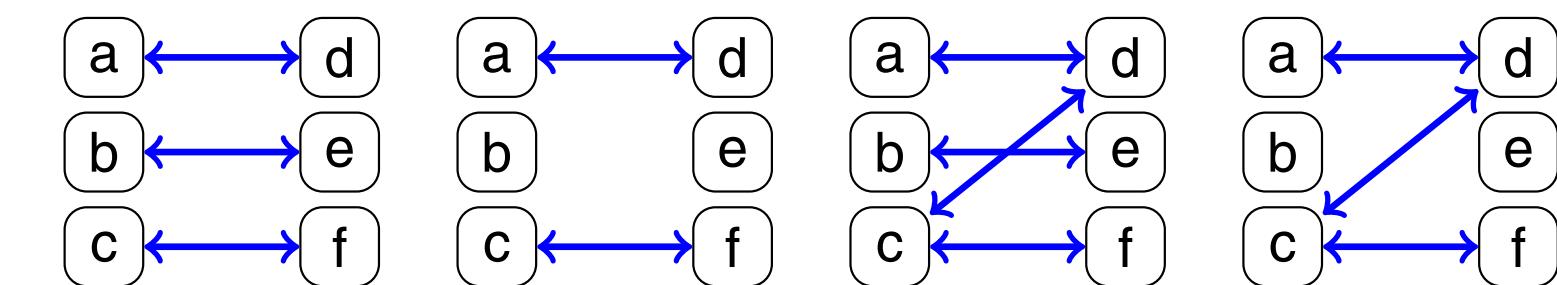
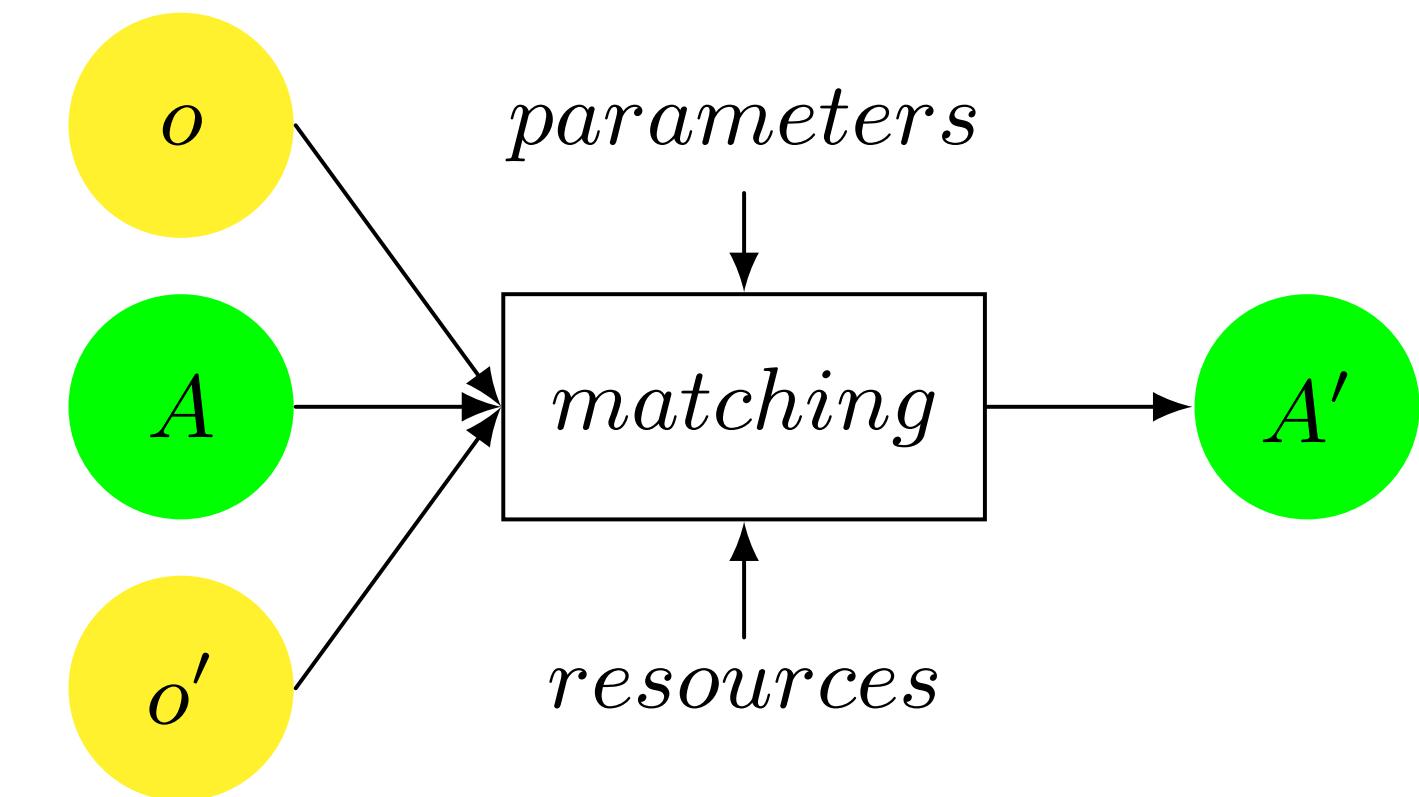
Types of correspondence relation between classes/ properties

	OWL	Example
= Equivalence	owl:sameAs	O:Florence = O':Firenze
≠ Difference	owl:differentFrom	O:John ≠ O':Ringo
∈ Instance	rdf:type	O:Beatles ∈ O':MusicGroup

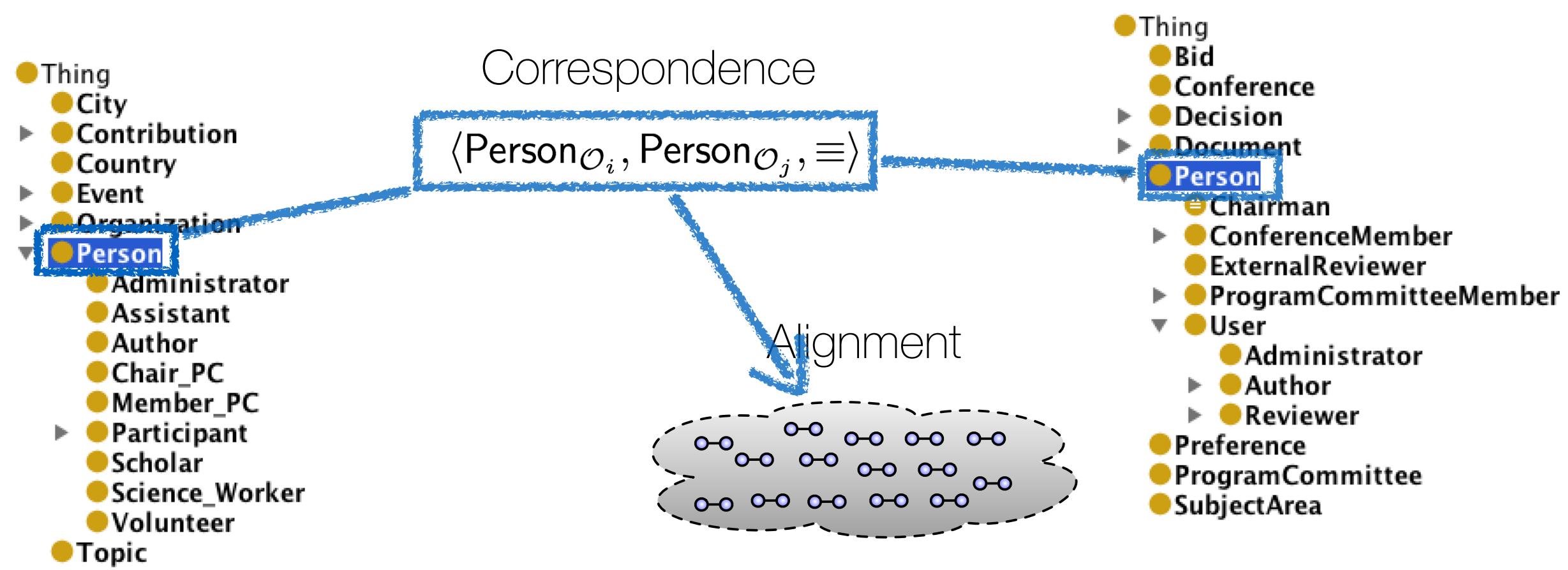
Ontology alignment and the matching process

- Alignments are generated through a *matching process*, a function f

- Input: two ontologies O and O' , and an optional input alignment A_{input} , set of parameters par , oracles and resources res
- Output: an alignment A' between O and O'
 - $A' = f(O, O', A_{input}, par, res)$
 - Set of possible correspondences, with the relationships between entities of O and O'
 - Different multiplicities possible



Alignment approaches



String based techniques

- **Syntactic approaches**

- based on the comparison of the labels used to denote entities
- Exploit measures of syntactic distance
- Assume language pre-processing
 - Stop word removal, Tokenization, Stemming

- **Taxonomy comparison**

- Identify common parents/children in the taxonomy

- Semantic similarity in taxonomies: Rada (1989), Resnick (1999)...

- **Language based approaches**

- Use external thesauri or multilingual resources
 - Relate terms in the WordNet hierarchy

- **Instance based mapping**

- Extensional, based on instance sets

String based techniques

- String based techniques are used to assess the similarity between the labels used to denote classes and properties in the ontologies.
- Exact string match
 - Prefix
 - takes as input two strings and checks whether the first string starts with the second one
 - *The prefix is a substring that is at the beginning of the original string*
 - **net** = **network**; but also **hot** = **hotel**
 - Suffix
 - takes as input two strings and checks whether the first string ends with the second one
 - *The suffix is a substring that is at the end of the original string*
 - **word** = **sword**; but also **nana** = **banana**

String based techniques

- Edit distance
 - takes as input two strings and calculates the number of edit operations
 - counting the minimum number of operations required to transform one string into the other.
 - e.g., *insertions, deletions, substitutions*
 - required to transform one string into another,
 - possibly normalised (divided) by length of the maximum string
 - EditDistance (**Nkn**, **Nikon**) = 0.4 (2/5)
 - **Nkn** → **Nikn** (add *i* at 1)
 - **Nikn** → **Nikon** (add *o* at 3)

Language pre-processing

- Often performed before applying string matching techniques
 - Stop word removal
 - common words, such as articles, prepositions, non-informative adverbs
 - Tokenization
 - extraction of terms from a document
 - text conflation and vocabulary reduction:
 - *from a document (one string) to a sequence of strings*
 - “**One quick brown fox**” vs “**One**” “**quick**” “**brown**” “**fox**”
 - Stemming
 - reducing words to their root forms
 - “**houses**” vs “**house**”

Tokenization

- Terms in ontologies are often made up of more than one word
 - e.g. class names
 - Wine, Wine grape, Wine Grape, Wine-Grape, WineGrape
 - e.g. property names
 - madeFromGrape, MadeFromGrape, Made-From-Grape, Made From Grape
- Tokenization
 - Extraction of the individual terms
 - removing punctuation and special characters
 - folding character case (e.g. all to lower case)

Stemming

- Reduces all morphological variants of a word to a single index term
 - stemming allows to recognise variations of the same word and treat them as if they were the same
 - detects mappings between concepts whose names include different forms of the same word
 - “**Mouse:Intestine_Epithelium**” vs “**NCI_Anat:Intestinal_Epithelium**”
 - Porter stemming algorithm (1980)
 - relies on a preconstructed suffix list with associated rules
 - e.g. if suffix=IZATION and prefix contains at least one vowel followed by a consonant, replace with suffix=IZE
 - “**intestine**” → “**intestinal**”

Wordnet and its senses

- WordNet is a lexical database for the English language. It is often used as a lexical ontology, although it was not developed for this purpose
 - it groups English words into sets of synonyms called **synsets** and provides short definitions and usage examples
 - records a number of relations among these synonym sets or their members, e.g. synonym...
- Both nouns and verbs are organised into hierarchies, defined through **hypernym** or **IS-A** relationships
 - All synsets are connected to other synsets by means of semantic relations, e.g.:
 - hypernyms: **Feline** is a hypernym of **Lion**
 - hyponyms: **Tiger** is a hyponym of **Feline**
 - coordinate terms: **Tiger** is a coordinate term of **Lion**, and **Lion** is a coordinate term of **Tiger**
 - meronym: **Finger** is a meronym of **Hand**
 - holonym: **Hand** is a holonym of **Arm**

Linguistic techniques using Wordnet

- A subClassOf B if A is a hyponym of B
 - *Tiger* is a hyponym of *Feline*
- A hasPart B if A is a holonym of B
 - *Country* is a holonym of *Continent*
- A ≡ B if A is a synonym of B
 - *Quantity* is a synonym of *Amount*
- A disjoint with B if A is an antonym (opposite) of B
 - or A and B are siblings
 - *Tiger* is disjoint with *Cat*

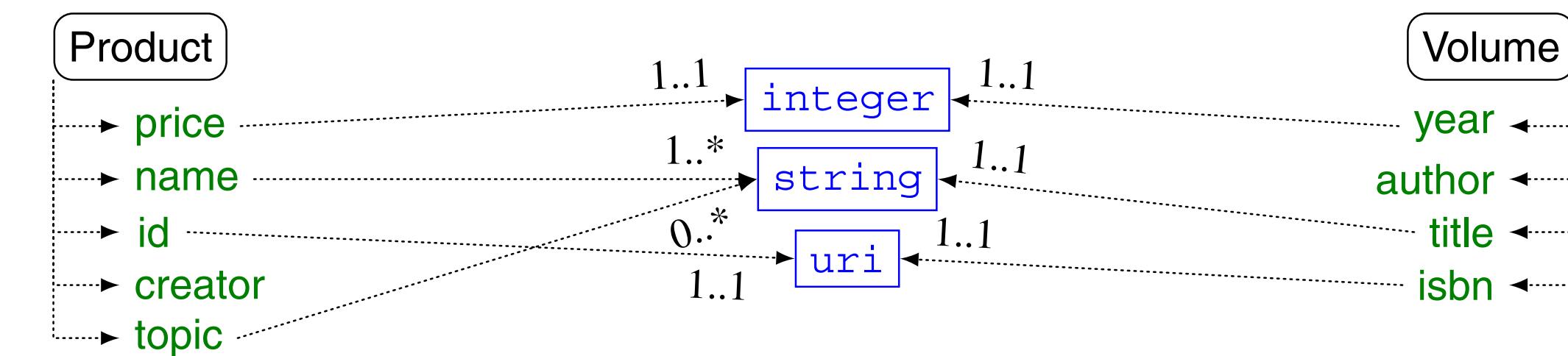
Linguistic techniques gloss-based

- WordNet gloss comparison
 - Glosses model relations obtained from references between synsets by computing semantic relatedness.
 - “*brother*”, “*sibling*”, “*family*”
 - The similarity value increases given the increase in the number of the same words occurring in both input glosses
 - The equivalence relation is returned if the resulting similarity value is above a given threshold
 - “*Maltese dog is a breed of toy dogs having a long straight silky white coat*”
 - “*Afghan hound is a tall graceful breed of hound with a long silky coat*”
 -

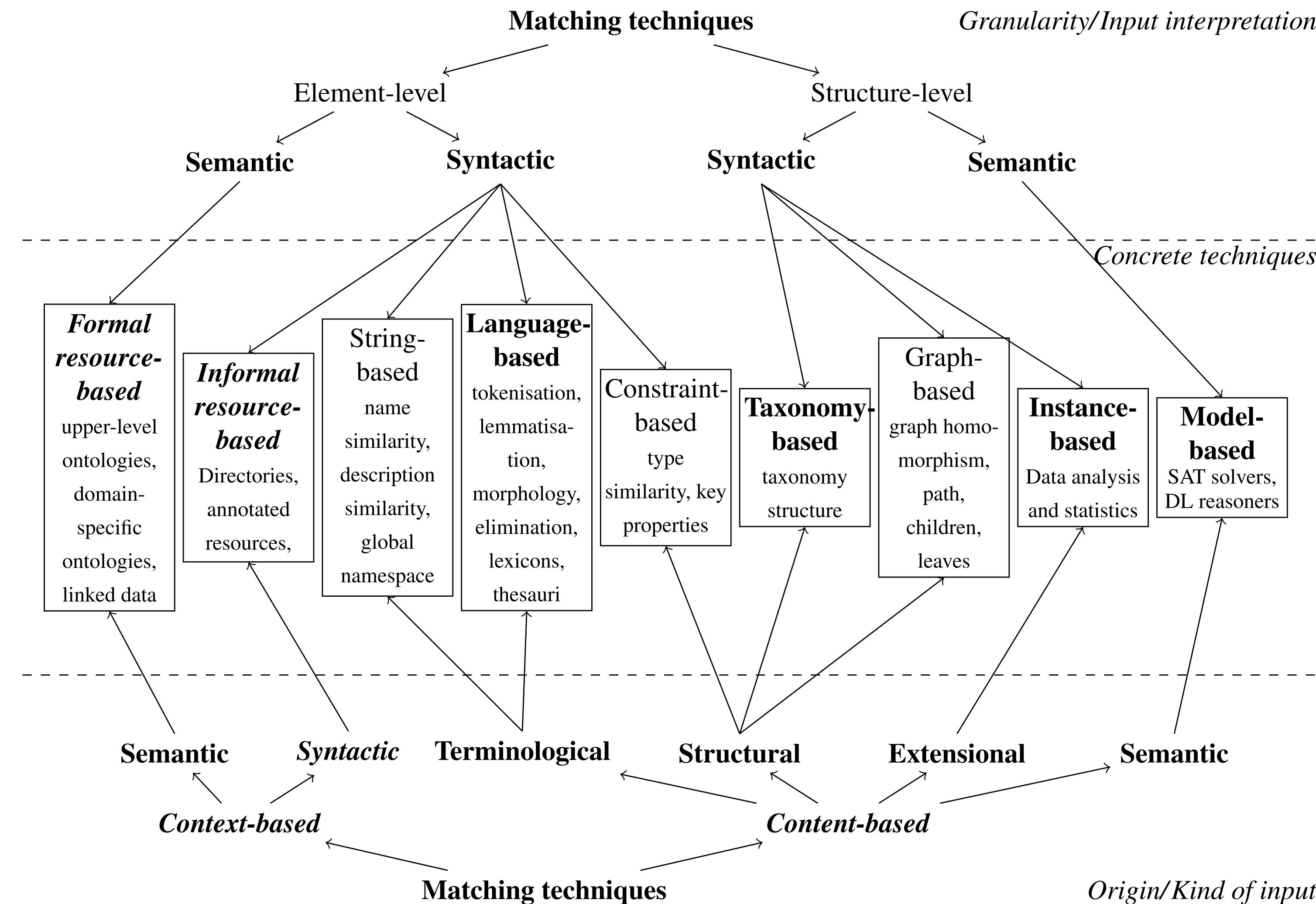
Structural techniques: hierarchy comparison

- Compare the structure of entities in ontologies instead or in addition to comparing the labels of the entities
(Euzenat and Shvaiko, 2012)

- This looks at the internal structure of an entity
- its name and annotations,
- its properties
- the properties whose value is a data type (OWL ontologies)
- the comparison of an entity with other related entities (OWL ontologies)



Classification of matching techniques



Challenges in using alignments

- Does the problem affect the way we align?
 - If we are merging ontologies, we could *align the whole ontology*
 - What about **FRANKENSTEIN** ontologies?
 - Quality vs Quantity
 - If we are using services or sensors, we should only *align the necessary concepts*
 - Is the alignment fit for purpose?
 - Fragments of the ontological space may be *confidential*, or *commercially sensitive*.
 - Disclosure or exposure is problematic



Challenges in using alignments

- Embarrassment of riches
 - What if we know many alignments?
 - Some better suited for certain tasks than others
 - Superset of different mappings, resulting in greater coverage
 - However, **aggregating alignments** may cause problems
 - Ambiguous mappings
 - Erroneous mappings



Challenges in using alignments

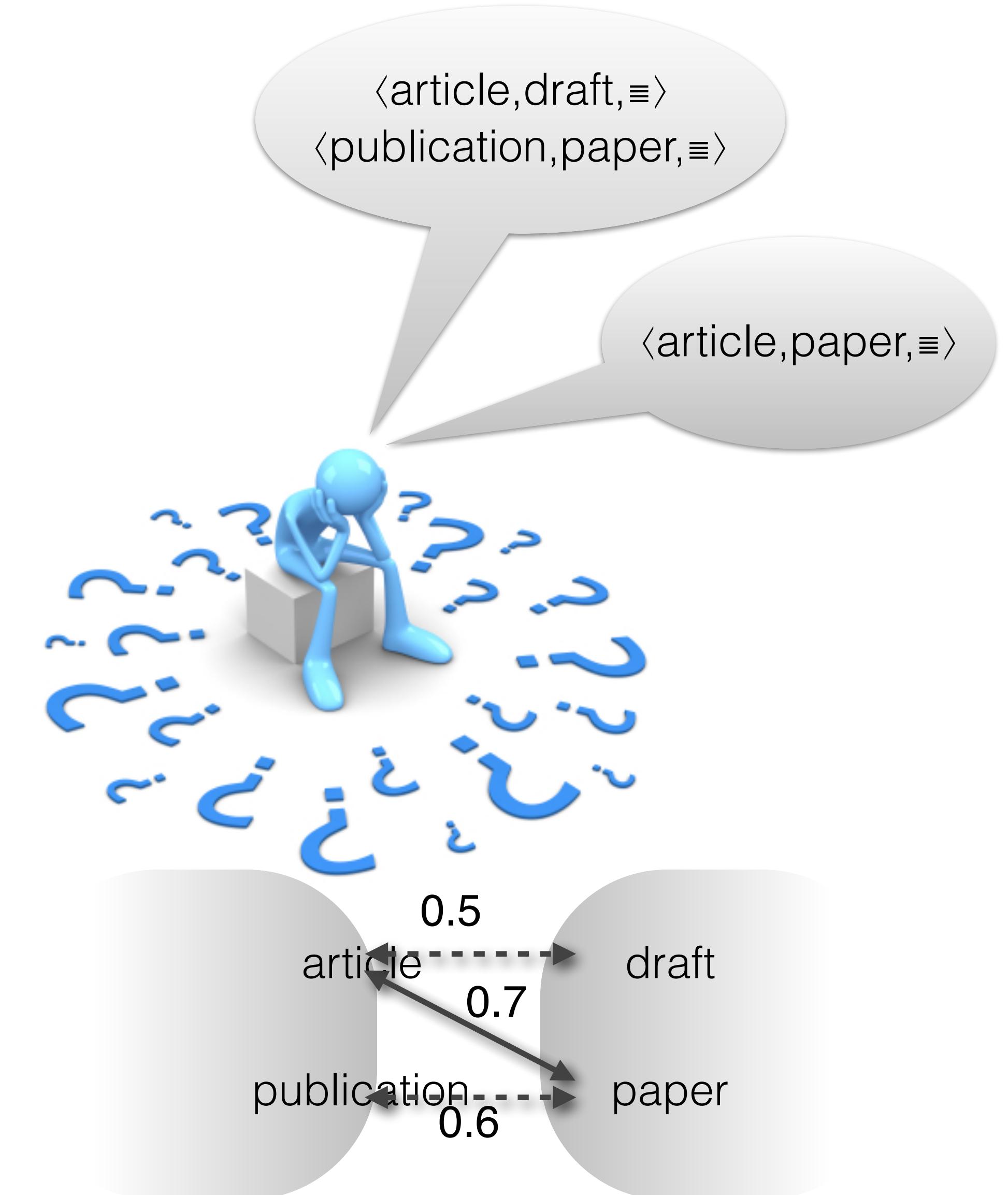
- Dearth of Knowledge

- What if have no prior alignments?
 - Need to understand how to align the ontologies
 - Easy if there are *existing services*, and the ontologies are *public*
- What if some of the ontological knowledge is *private*?
 - Disclosure of the full ontology no longer possible
 - Need to collaborate / negotiate to establish the mappings



Picking the right mappings

- Quality vs Quantity
 - Do we maximise coverage
 - Preferable when merging the whole ontology
 - Do we find the “best” mappings
 - Preferable when aligning specific signatures
- Interpretation of the output
 - Approximate vs exact
 - Graded vs absolute confidence
- Performance varies
 - Semi automatic alignment



Alignment evaluation

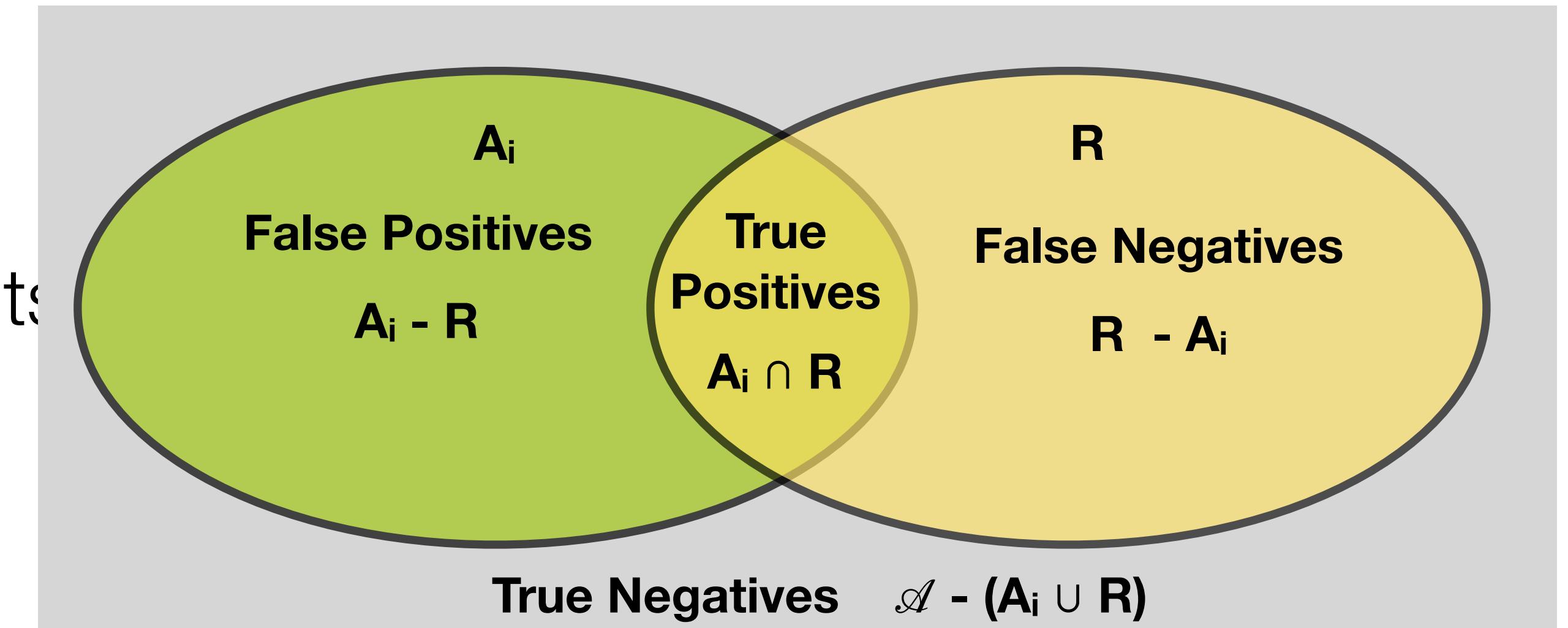
- Three types of alignment evaluation approaches:
 - Competence benchmarks: allow users to measure the extent of competence (and performance) of a particular system with regard to a set of well defined tasks
 - Usually, tasks are designed to isolate particular characteristics
 - E.g. set of tests designed
 - Comparative evaluation: compare the results of various systems or several versions of the same system on a common task.
 - The rules and the evaluation criteria MUST be clearly specified
 - Blind or nearly blind tests
 - Application-specific evaluation: compare the results of various systems wrt the output of a particular application instead of considering the alignments in isolation
 - Usually, tasks are designed to isolate particular characteristics

OAEI

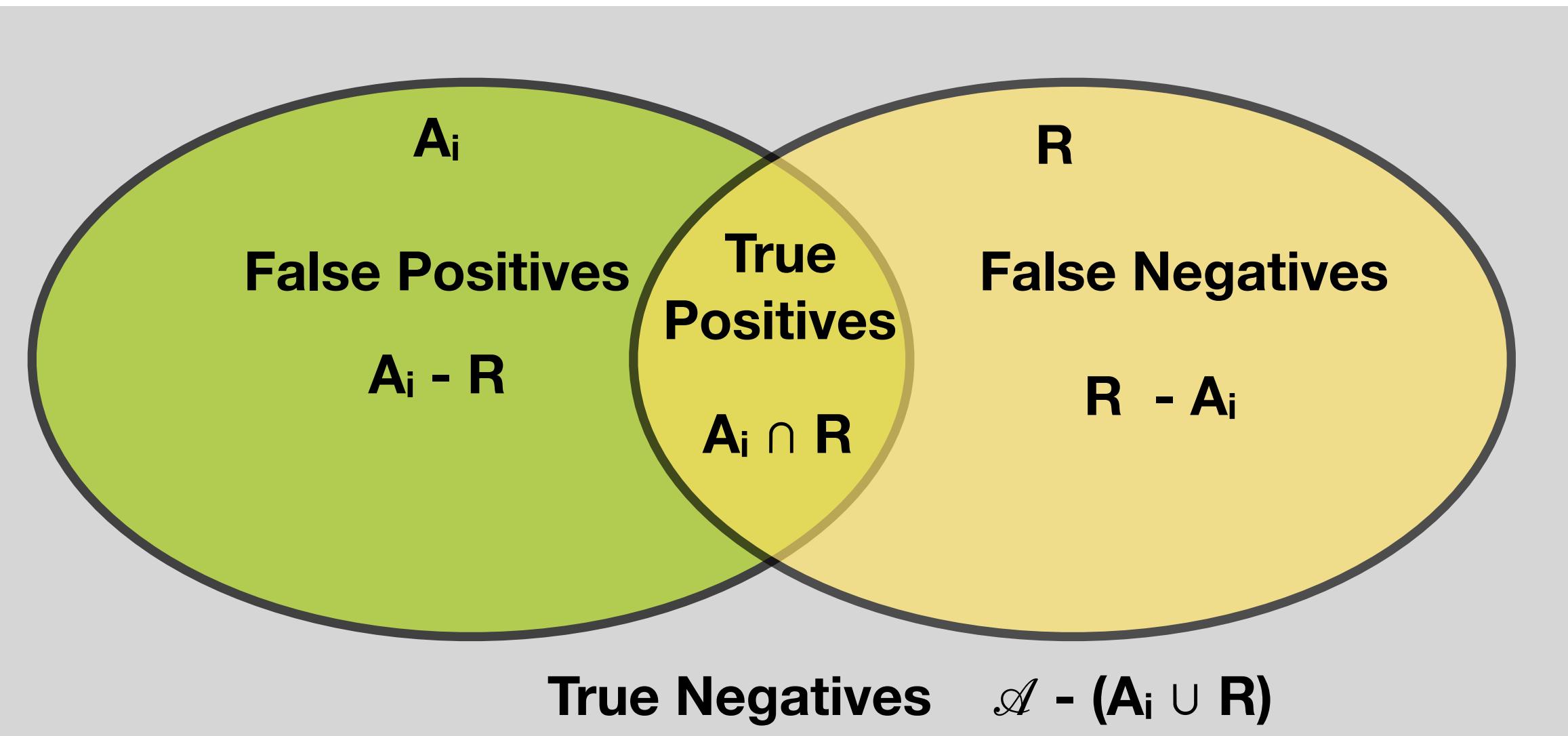
- Ontology Alignment Evaluation Initiative defines a set of benchmark dataset to be used to evaluate ontology alignment approaches.
- Annual challenge, started in 2004
- Its aims are
 - *assessing strengths and weaknesses of alignment/matching systems;*
 - *comparing performance of techniques*
 - *increase communication among algorithm developers;*
 - *improve evaluation techniques*
 - *helping improving the work on ontology alignment/matching*

Compliance Measures: Precision and Recall

- Compliance measures evaluate the degree of compliance of a system with regard to some standard
 - typically a reference alignment $\textcolor{orange}{R}$ defined as part of the OAEI benchmark
- Given:
 - the set of all possible alignments \mathcal{A}
 - an alignment to assess $A_i \in \mathcal{A}$
 - and a reference alignment $\textcolor{orange}{R}$



Compliance Measures: Precision and Recall



Precision

$$\text{Prec}(A_i, R) = \frac{|A_i \cap R|}{|A_i|}$$

Recall

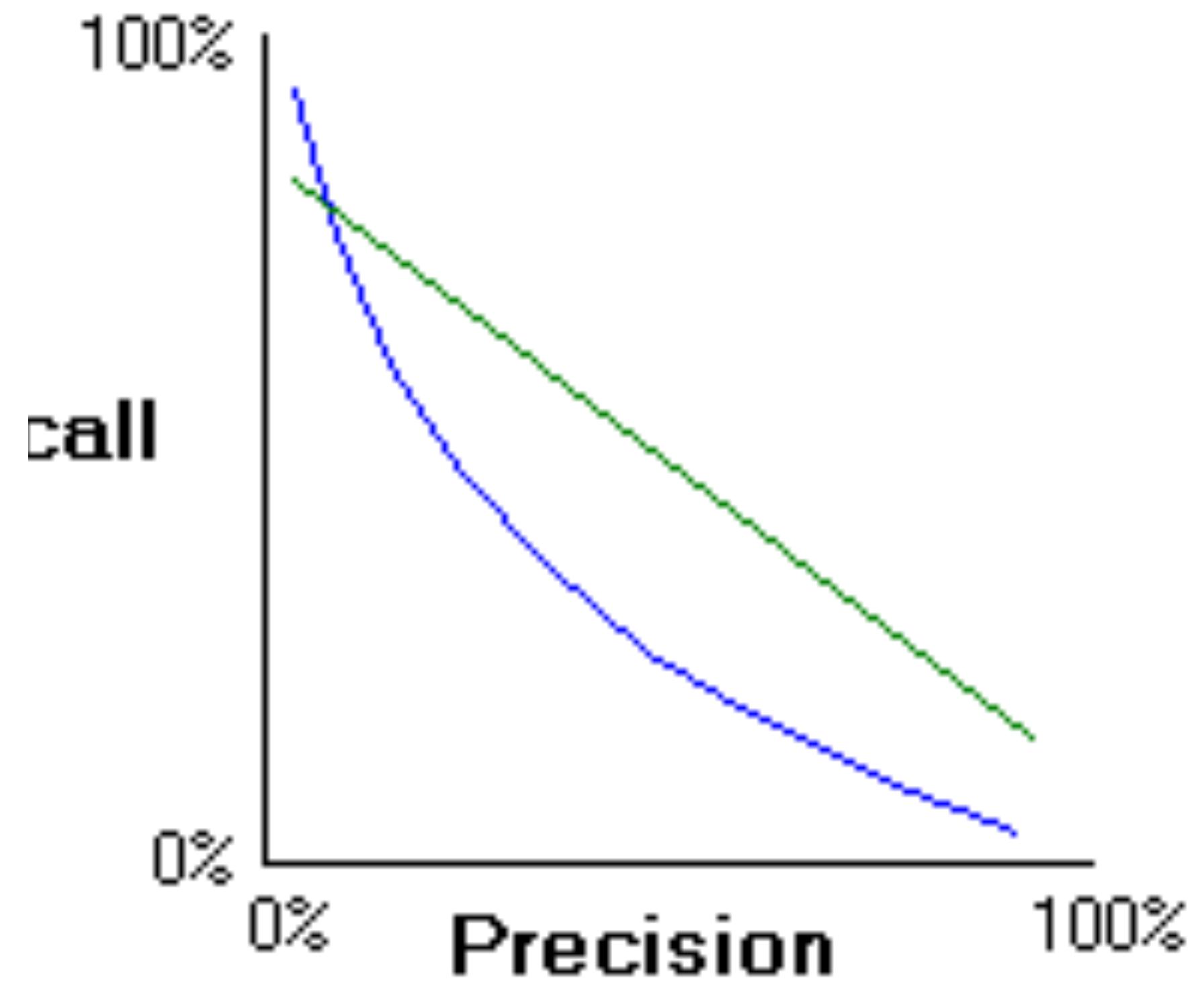
$$\text{Rec}(A_i, R) = \frac{|A_i \cap R|}{|R|}$$

F-measure

$$F_{\text{measure}}(A_i, R) = 2 \times \frac{\text{Prec}(A_i, R) \times \text{Rec}(A_i, R)}{\text{Prec}(A_i, R) + \text{Rec}(A_i, R)}$$

Precision and recall behaviour

- While the exact slope of the curve may vary between systems, the general inverse relationship between recall and precision remains.
- Why is there an inverse relationship? Much of this relationship has to do with language:
 - If the goal of a search is comprehensive retrieval, then the searcher must include synonyms, related terms, broad or general terms, etc. for each concept. They use Boolean operators to combine terms. Secondary concepts may be omitted. **As a consequence of these decisions, precision will suffer.**
 - Synonyms may not be exact, so the probability of retrieving irrelevant material increases. Broader terms may result in the retrieval of material which does not discuss the narrower search topic. Using Boolean operators may increase the probability that the terms won't be in context. **This might cause low recall!**



Precision, recall, F-measure

- **Precision:** fraction of relevant correspondences ($|A_i \cap R|$) amongst all the correspondences generated ($|A_i|$)
 - score reaches its best value at 1 (perfect precision and recall) and worst at 0.
- **Recall:** fraction of relevant correspondences produced by an alignment system ($|A_i \cap R|$) over the total number of relevant correspondences
 - those in the reference alignment
- **F-measure:** is the harmonic mean of precision and recall
 - Precision and recall should not be considered in isolation,
 - as considering just one out of precision and recall can lead to extreme but unhelpful solutions.
 - A system that returns every correspondence indiscriminately has 100% recall;
 - **Recall without precision will provide too many results.**
 - A system that returns only a single correct correspondence is 100% precise.
 - **However, only precision will limit the number of results (e.g. finding only one of the possible correct results will yield a precision of 100%).**

Summary

- The need of knowledge integration and sharing
 - support for interoperability
- Ontology alignment
 - Definition
 - Techniques
 - Evaluation metrics

COMP318: Ontologies and Semantic Web

Describing Web Resources in RDF



Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

v.Tamma@liverpool.ac.uk

Where were we

- Problems with today's web
 - User defined queries
 - Ambiguity
 - Poor support for integration tasks and complex queries
- Need for more semantics

As we read it

What is the page about

When does it take place

Where does it take place

Companies sponsoring

Keynote speakers

Vienna - Austria
ISWC 2017
THE 16TH INTERNATIONAL SEMANTIC WEB CONFERENCE
October 21-25

HOME ATTENDING CALLS IMPORTANT DATES PROGRAM ORGANIZATION SPONSORSHIP FAQ GALLERIES

HOME


Vienna
One of the most liveable cities in the world

ISWC 2017 is the premier international forum, for the Semantic Web / Linked Data Community. ISWC 2017 will bring together researchers, practitioners and industry specialists to discuss, advance, and shape the future of semantic technologies. Every year ISWC offers live exciting and fruitful days that you definitely don't want to miss!

Looking forward to seeing you in Vienna!

Go to top

Keynote Speakers

 Deborah L. McGuinness Senior Chair of Tetherless World Constellation Professor of Computer, Cognitive, and Web Sciences at Rensselaer Polytechnic Institute	 Nada Lavrač Head of Department of Knowledge Technologies at Jožef Stefan Institute Vice Dean at Jožef Stefan International Postgraduate School Professor of Computer Science at University of Nova Gorica	 Jamie Taylor Manager of the Knowledge Graph Schema Team at Google.
--	---	--

Go to top

ISWC2017 Proceedings
The online versions of ISWC2017 proceedings are available now. You can get access them by clicking links below until November 30th, 2017.
(All preprints are also linked in the program.)

IMPORTANT NEWS

- Video Lectures
- ISWC2017 is over...
- List of awards
- ISWC2017 - full program
- Semantic Web Challenge - deadline extension

SPONSORS

Platinum Sponsors




Gold Sponsors





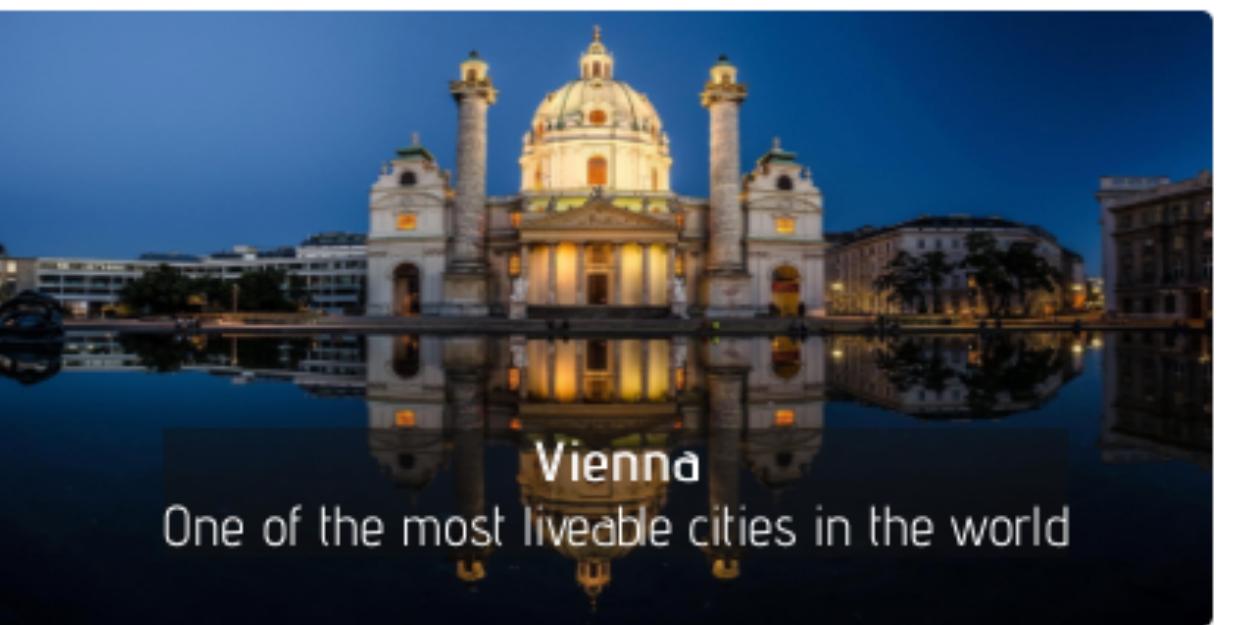








HOME



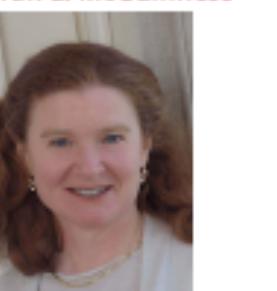
One of the most liveable cities in the world

ISWC 2017 is the premier international forum, for the Semantic Web / Linked Data Community. ISWC 2017 will bring together researchers, practitioners and industry specialists to discuss, advance, and shape the future of semantic technologies. Every year ISWC offers five exciting and fruitful days that you definitely don't want to miss!

Looking forward to seeing you in Vienna!

Keynote Speakers

Deborah L. McGuinness



Senior Chair of Tetherless
World Constellation

Professor of Computer,
Cognitive, and Web Sciences at
Rensselaer Polytechnic
Institute

Professor of Computer Science
at University of Nova Gorica

[to top](#)

Jada Lavrač



Department of
Technologies at
Institute

Jamie Taylor



Chapter 10 | Page 10

 THOMSON REUTERS

to top

SWC2017 Proceedings

The online versions of ISWC2017 proceedings are available now. You can get access them by clicking links below until November 30th, 2017.

All preprints are also linked in the program.)

As a computer reads it



Vienna - Austria

ISWC 2017

THE 16TH INTERNATIONAL SEMANTIC WEB CONFERENCE
October 21-25

HOME ATTENDING CALLS IMPORTANT DATES PROGRAM ORGANIZATION SPONSORSHIP FAQ GALLERIES

HOME


Vienna
One of the most liveable cities in the world

ISWC 2017 is the premier international forum, for the Semantic Web / Linked Data Community. ISWC 2017 will bring together researchers, practitioners and industry specialists to discuss, advance, and shape the future of semantic technologies. Every year ISWC offers five exciting and fruitful days that you definitely don't want to miss!

Looking forward to seeing you in Vienna!

Go to top

Keynote Speakers

Deborah L. McGuinness 

Senior Chair of Tetherless World Constellation
Professor of Computer, Cognitive, and Web Sciences at Rensselaer Polytechnic Institute

Nada Lavrač 

Head of Department of Knowledge Technologies at Jožef Stefan Institute
Vice Dean at Jožef Stefan International Postgraduate School

Jamie Taylor 

Manager of the Knowledge Graph Schema Team at Google.
Professor of Computer Science at University of Nova Gorica

Go to top

ISWC2017 Proceedings

The online versions of ISWC2017 proceedings are available now. You can get access them by clicking links below until November 30th, 2017.

(All preprints are also linked in the program.)

IMPORTANT NEWS

- Video Lectures
- ISWC2017 is over...
- List of awards
- ISWC2017 - full program
- Semantic Web Challenge - deadline extension

SPONSORS

Platinum Sponsors




Gold Sponsors












A richer data model

- Semantic Web: beyond machine readable to machine understandable.
 - data model:
 - data model that can be used by multiple applications
 - not only for describing documents
 - for people to describe application-specific information
 - data model that is domain independent
 - any application can use it to describe information
 - semantics:
 - mechanism to interpret the data model
 - describes the interpretations of the data items wrt the domain
 - syntax:
 - standardised exchange mechanism

Let's encode meaning in XML

- XML = eXtensible Markup Language
 - set of rules for encoding documents in machine processable form.
 - It was designed to transport and store data
 - the focus is what data is
 - XML complements HTML:
 - HTML was designed to display data
 - the focus is how data looks
 - XML is not meant to do anything!
 - it was created to structure, store and transport information.
 - XML is now the most common tool for data transmissions between all sorts of applications

XML is not the answer

- Meaning of XML documents is intuitively clear
 - “semantic” markup tags are domain terms
 - no unique way to express the same information
- But computers do not have intuition
 - Tag names per se do not provide semantics
 - The semantics are encoded outside the XML specification
- XML makes no commitment on:
 - Domain specific ontological vocabulary
 - Ontological modelling primitives
 - requires pre-arranged agreement on 1. & 2.
- Feasible for closed collaboration
 - agents in a small & stable community
 - pages on a small & stable intranet

Nesting of Tags in XML

- John Smith is a lecturer of SemWeb Technologies

```
<course name="SemWeb Technologies">
    <lecturer>John Smith</lecturer>
</course>
```

- Opposite nesting, same information!

```
<lecturer name="John Smith">
    <course>SemWeb Technologies</course>
</lecturer>
```

RDF: Resource Description Framework

- Semantic Web: beyond machine readable to machine understandable.
 - Resource Description Framework (RDF) is the W3C language for describing metadata on the Web.
- Models Meta-Data about resources on the Web using subject-predicate-object triples
 - Triples define the relationship or predicate between two entities (the subject and object)

RDF Building Blocks

Statements

Statements are subject-predicate-object triples.

They assert the properties of a resource the resource, a property, and a value

Objects can be resources or literals (atomic values - strings)

Resources are similar to entities in ER models

–“something” we want to describe

- E.g. *authors, books, publishers, places, people, hotels*

Every resource has a URI

–a URL (Web address) or

–some other kind of unique identifier: URNs

Advantages of using URIs:

–a global, worldwide, unique naming scheme

–reduces the homonym problem in distributed data

Properties

Properties are special types of resources

–they describe semantic relations between resources

- E.g. *written by, age, smaller than, etc*

–they are also identified by a URI

Three Views of a Statement

- A statement can be seen as:
 - A triple
 - A piece of a graph
 - A XML code fragment
- Thus an RDF document can be viewed as:
 - A set of triples
 - A graph (semantic net)
 - An XML document

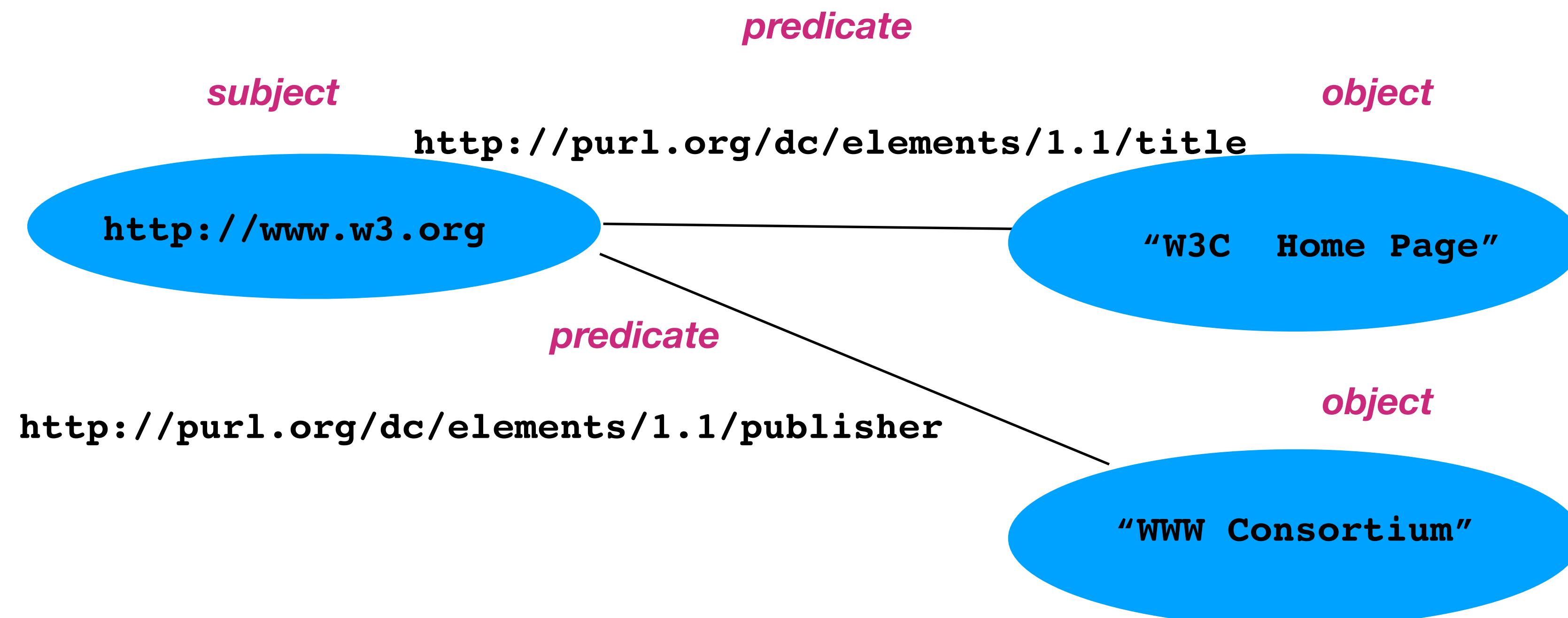
Statements as Triples

```
(John Smith,  
http://www.jsdomain.org/site-owner,  
http://www.liv.ac.uk/SemWeb)
```

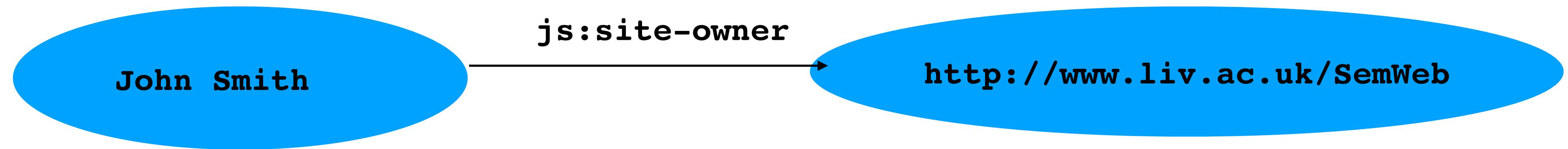
- The triple (x, P, y) can be considered as a logical formula $P(x, y)$
 - Binary predicate P relates object x to object y
 - RDF offers only binary predicates (properties)
 - mydomain:site-owner(John Smith,liv:SemWeb)

RDF Graphs

- The subject/predicate/object triples found in an RDF document form a graph:

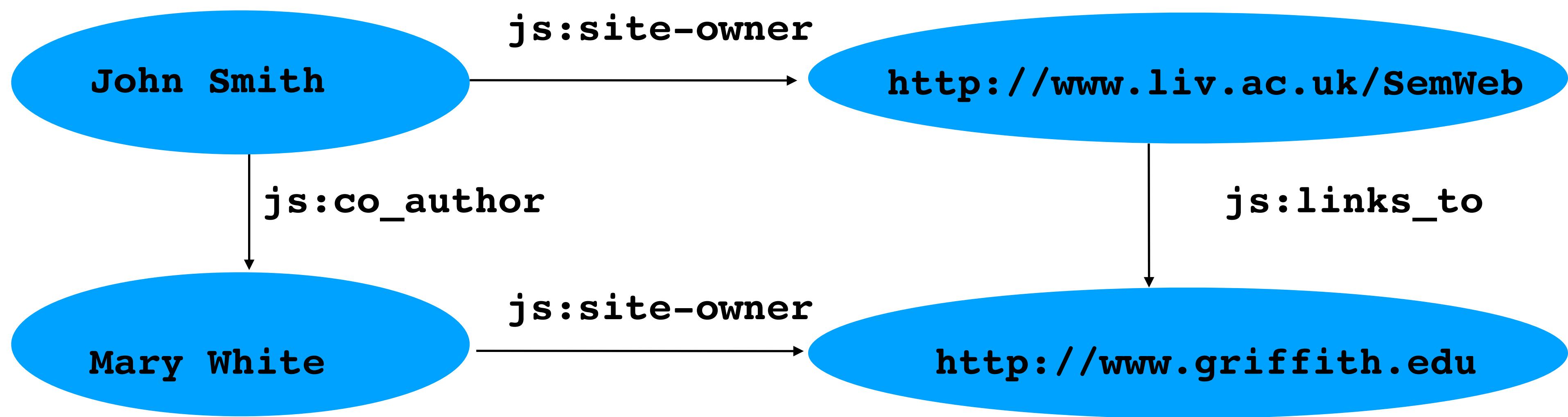


RDF Graph



- A directed graph with labeled nodes and arcs
 - **from** the resource (the **subject** of the statement)
 - **to** the value (the **object** of the statement)
- The value of a statement may be a resource
 - It may be linked to other resources

A Set of Triples as a graph (Semantic Net)



Same Statements in XML syntax

```
<rdf:RDF  
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
    xmlns:js="http://www.jsdomain.org/my-rdf-ns">  
  
<rdf:Description  
    rdf:about="http://www.liv.ac.uk/SemWeb">  
    <js:site-owner  
        rdf:resource=John Smith/>  
    </rdf:Description>  
</rdf:RDF>
```

Statements in XML

- An RDF document is represented by an XML element with the tag **rdf:RDF**
- The content of this element is a number of **descriptions**, which use **rdf:Description** tags.
- Every description makes a statement about a resource, identified in 3 ways:
 - an **about** attribute, referencing an existing resource
 - an **ID** attribute, creating a new resource
 - without a name, creating an anonymous resource

Node and Edge labels in RDF graphs

- Node and edge labels can be:
 - URI
 - Literal (string)
 - Bnode (anonymous label)
- However:
 - Only URIs and Bnodes can be the **subject** of a triple
 - Only URIs can be the predicate of a triple
 - Only URIs, Bnodes and literals can be the **object** of a triple

Complex values

- Values of properties do not need to be simple strings
- The value of a property can also be a graph node (corresponding to a resource)
 - arbitrarily complex tree and graph structures are possible
 - Values can be syntactically embedded (i.e., lexically in-line) or referenced (linked)

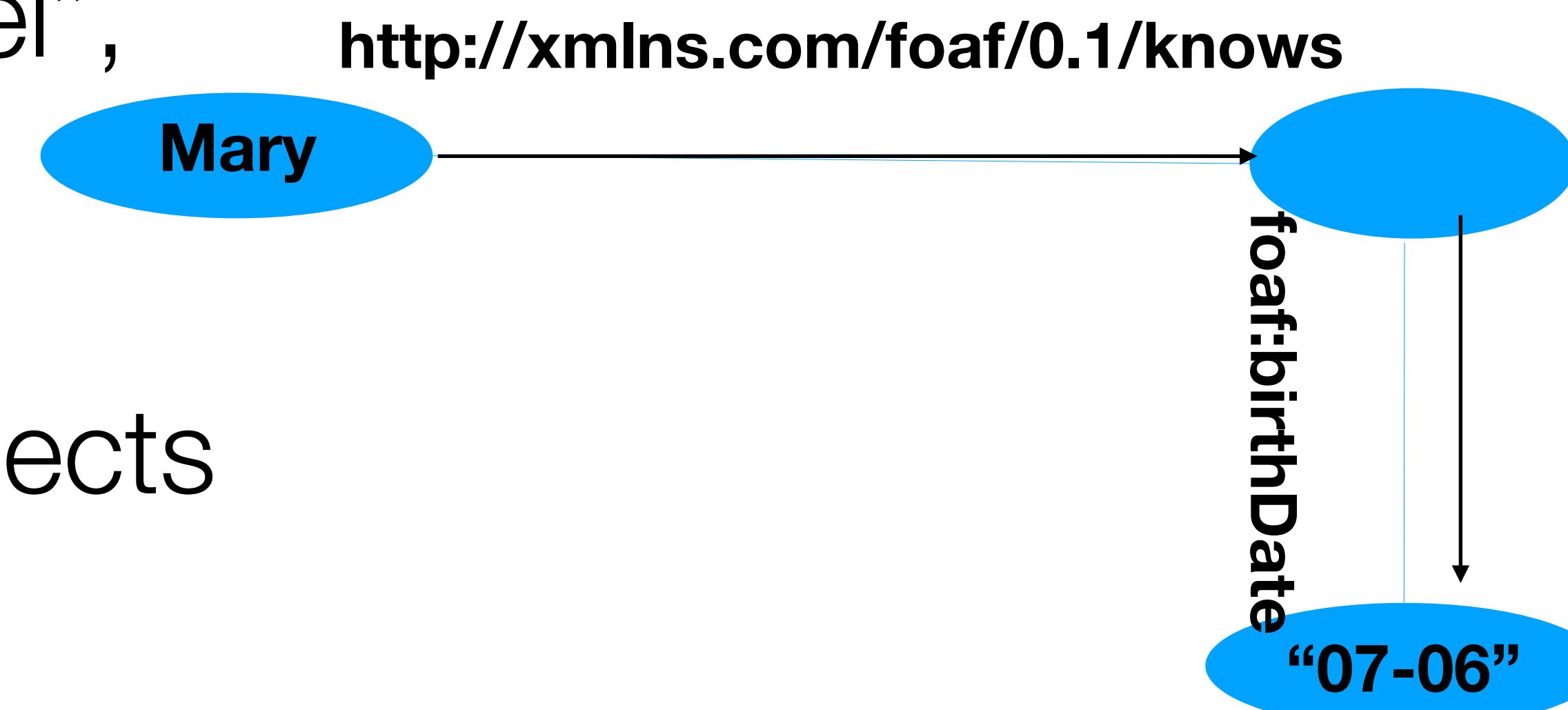
Blank Nodes

- Blank nodes (bnode) denote an RDF graph node with “anonymous label”,
 - the node is not associated with a URI

- Bnodes can be used both as subjects and objects

- For example, the statement “Mary has a friend who is born on June 7th”

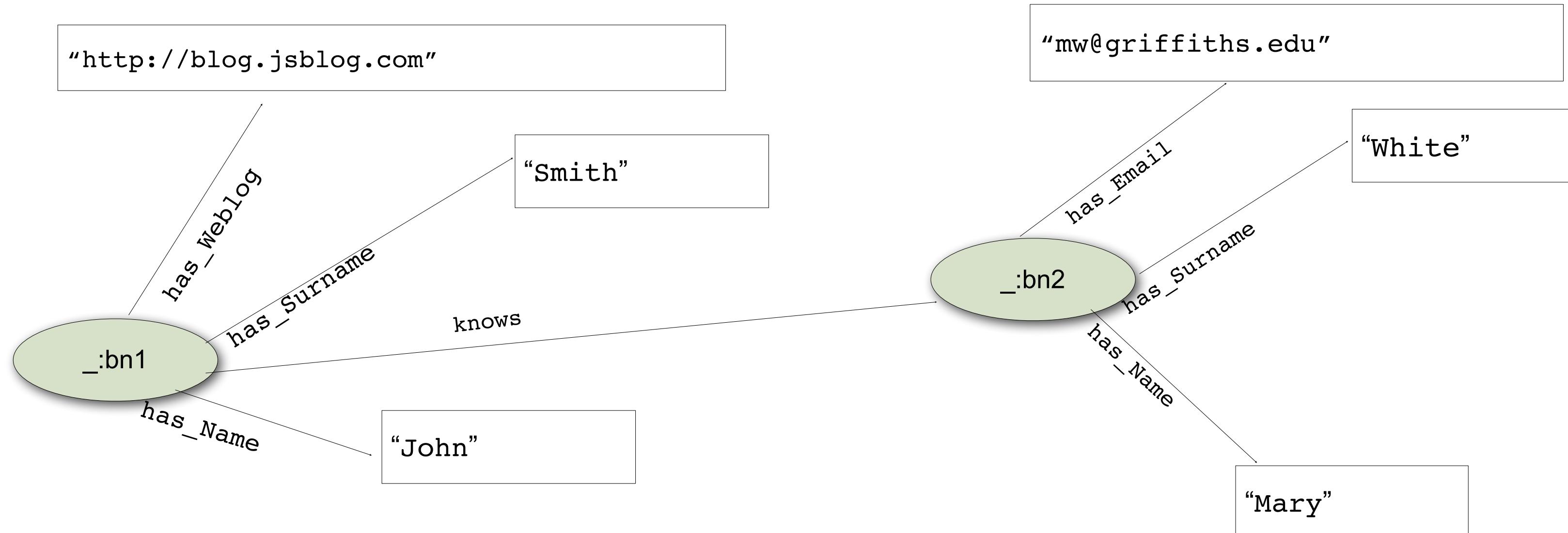
- `_:p1` is the blank node (bnode)



<code>ex: Mary</code>	<code>foaf:knows</code>	<code>_:p1</code>
<code>_:p1</code>	<code>foaf:birthDate</code>	<code>07-06</code>

Digression: blank nodes

- Social networks APIs do not issue URLs for the members of their community, even if they have lots to say about them.
- a blank node is used to represent a member and the facts about the member are linked to the blank node



Example

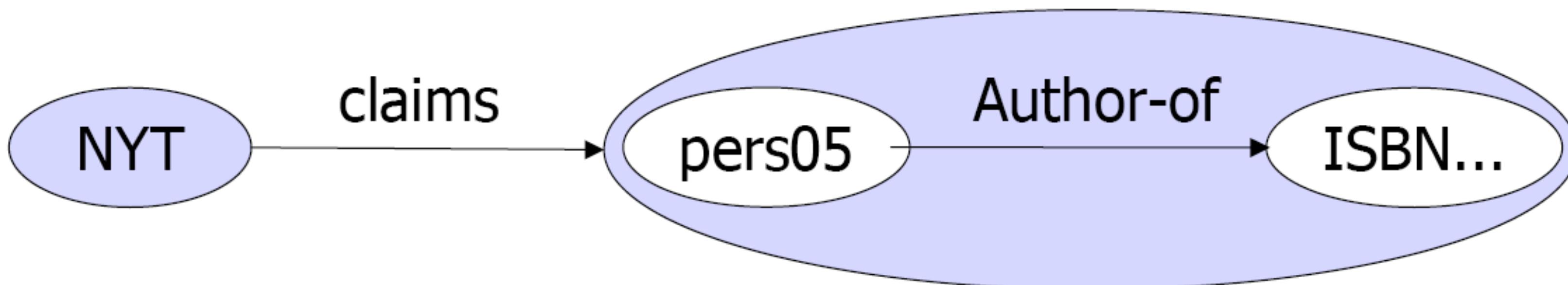
```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix dc: <http://purl.org/dc/elements/1.1/> .  
@prefix : <http://example.org/#> .  
  
<http://www.w3.org/TR/rdf-syntax-grammar>  
dc:title "RDF/XML Syntax Specification (Revised)" ;  
:editor [  
  :fullName "Dave Beckett";  
  :homePage <http://purl.org/net/dajobe/>  
] .
```

Higher order statements

- RDF allows you to make statements about other RDF statements
 - “Ralph believes that the web contains one billion documents”
- Higher-order statements
 - allow us to express beliefs (and other modalities)
 - are important for trust models, digital signatures,etc.
 - also: metadata about metadata
 - are represented by modelling RDF in RDF itself
- Reification

Reification

- Any RDF statement can be an object
- We must be able to refer to a statement using an identifier
 - allows users to point to a particular statement (and part of a graph)
- RDF allows such reference through a reification mechanism which turns a statement into a resource
 - newer versions of RDF introduce named graphs where an identifier is assigned to a set of statements



Reification Example

```
<rdf:Description rdf:about="#949352">
  <uni:name>John Smith</uni:name>
</rdf:Description>
```

```
<rdf:Statement rdf:ID="StatementAbout949352">
  <rdf:subject rdf:resource="#949352" />
  <rdf:predicate rdf:resource=
    "http://www.jsmydomain.org/uni-ns#name"/>
  <rdf:object>John Smith</rdf:object>
</rdf:Statement>
```

Recap

- Limitation of XML
- RDF
 - Basic ideas behind RDF
 - Statements about resources
 - triples
 - graphs
 - XML vocabularies
 - Modelling primitives in RDF

COMP318: Intro on Linked Open Data

www.csc.liv.ac.uk/~valli/Comp318



Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

v.Tamma@liverpool.ac.uk

The big data shredder



**Structured
Data**

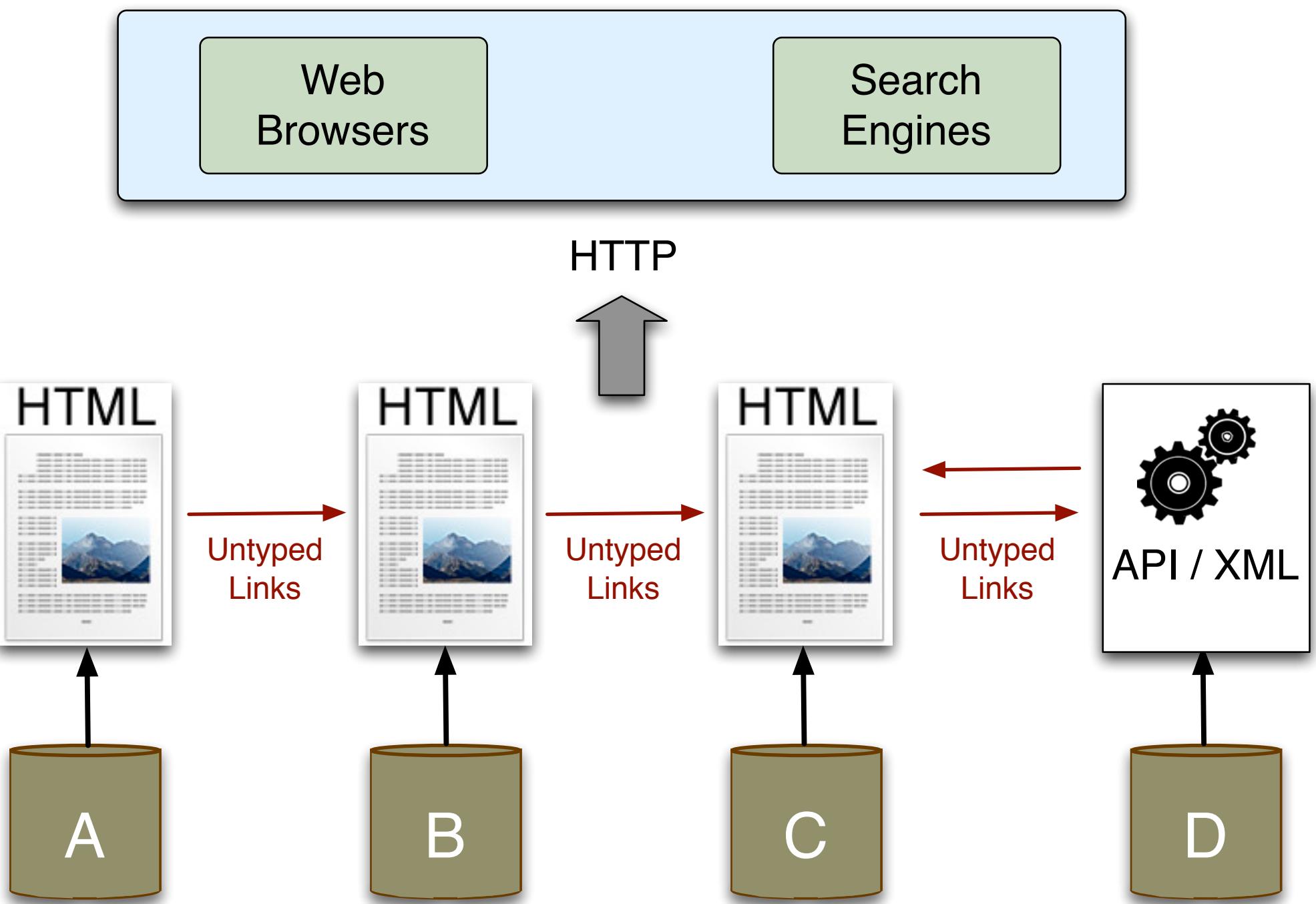


**Unstructure
Data**

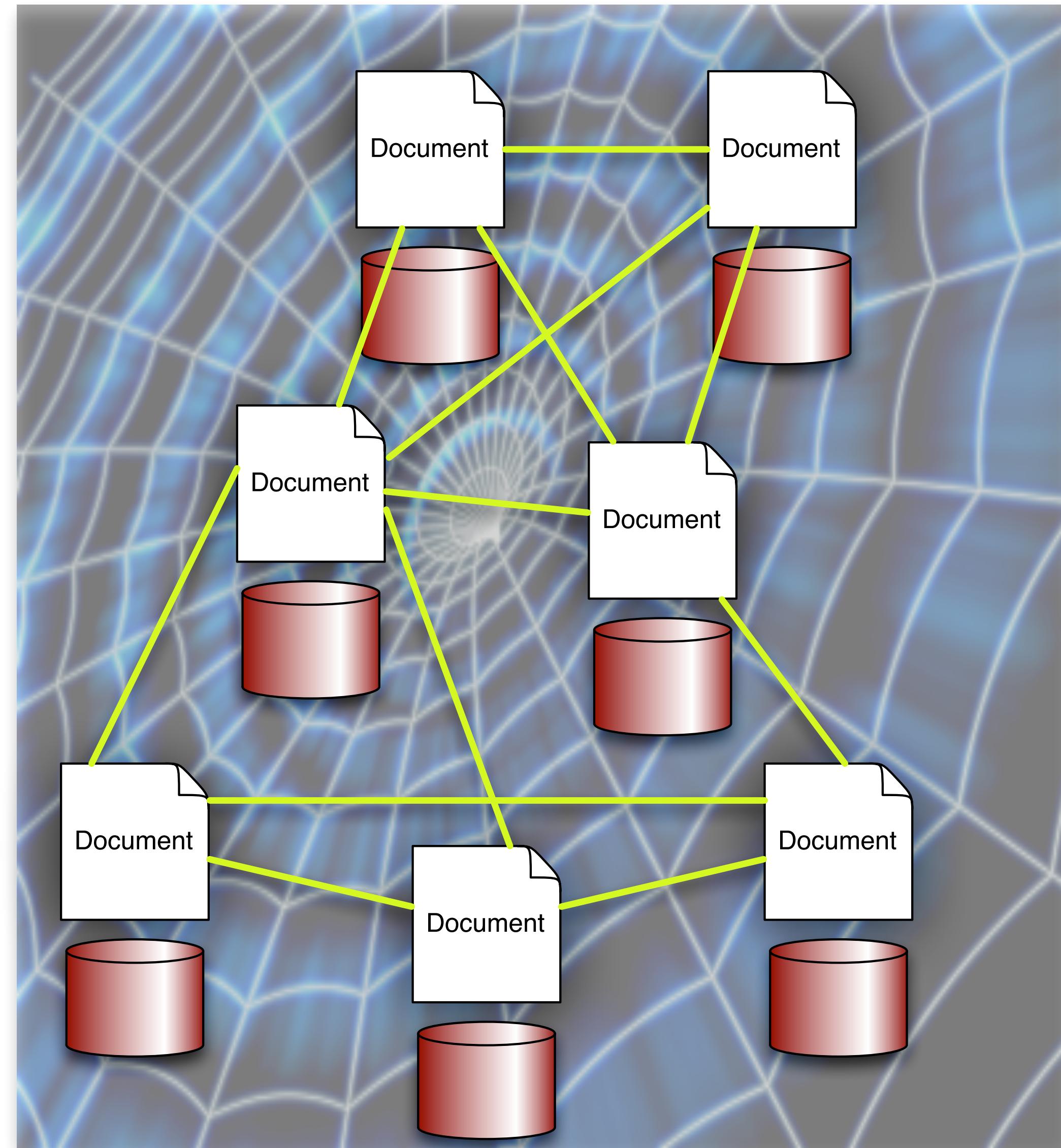
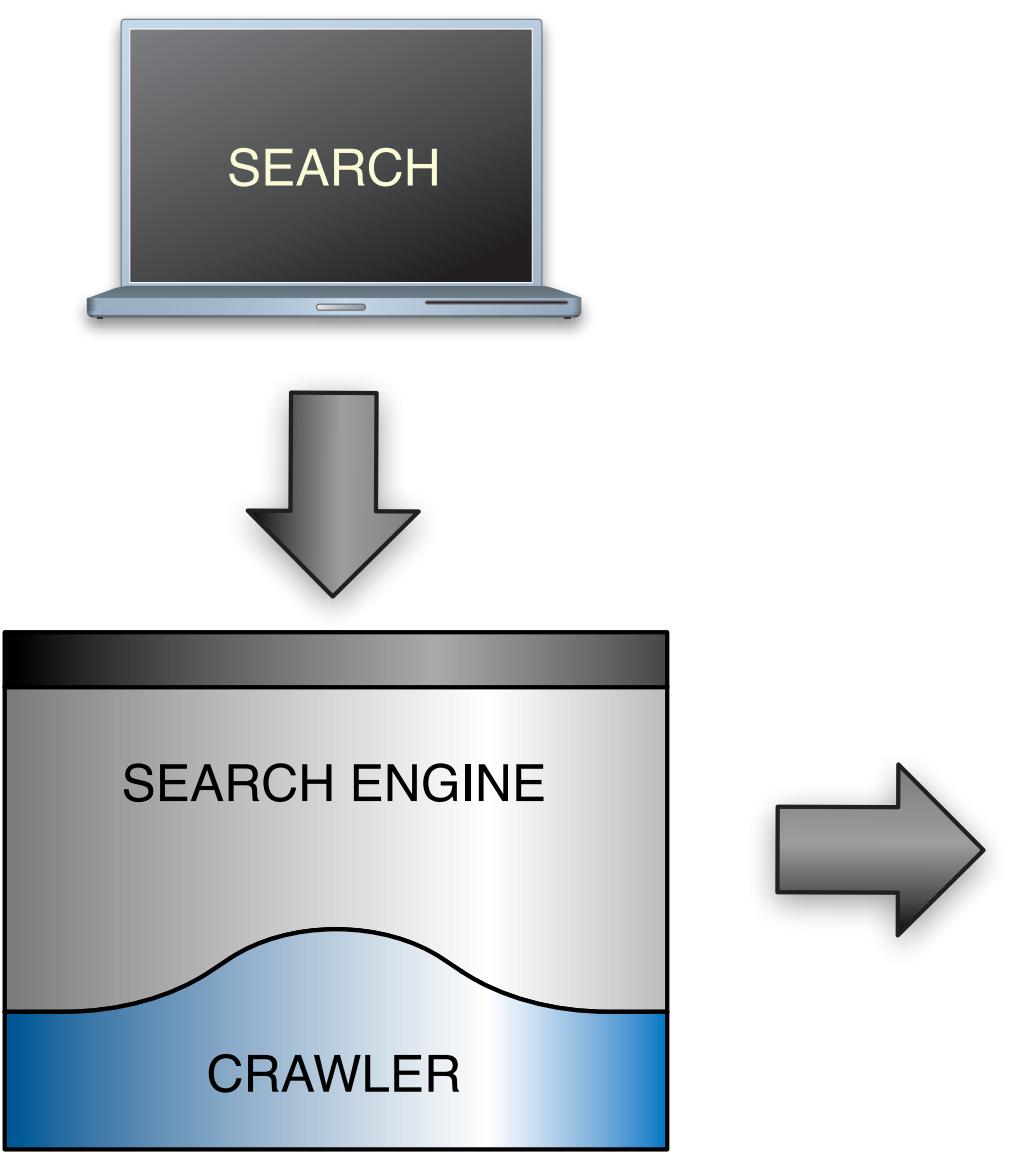
Picture by Martin Hepp

Back to the beginning: the Web

- The web as a single global information space, similar to a filesystem:
 - Mainly designed for human consumptions;
 - Primary objects: documents;
 - (Hyper) Links between: documents or parts of them;
 - Objects mainly **unstructured** or **semi-structured**;
 - **Implicit** semantics of content and links.

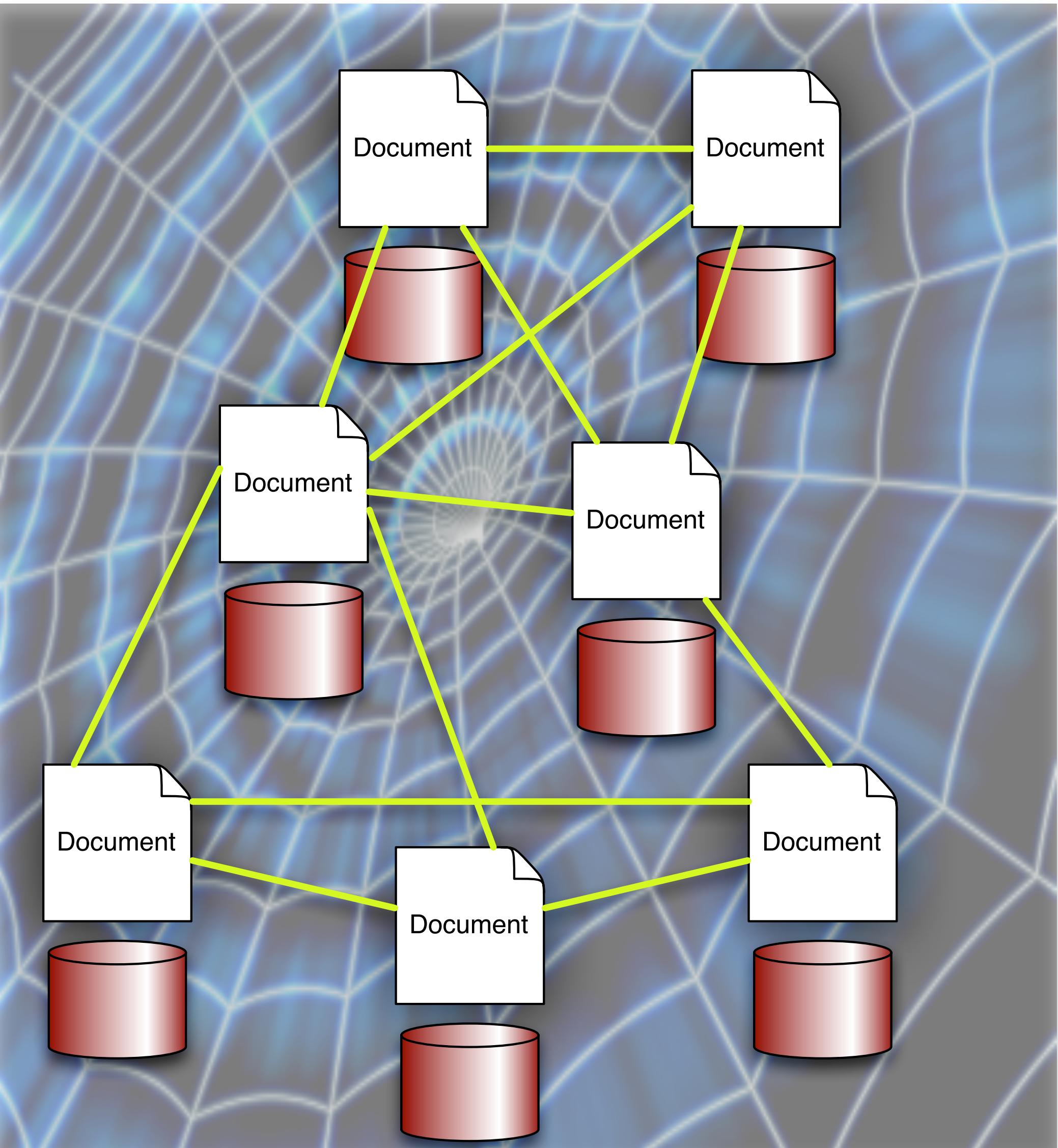


A web of documents



A web of documents

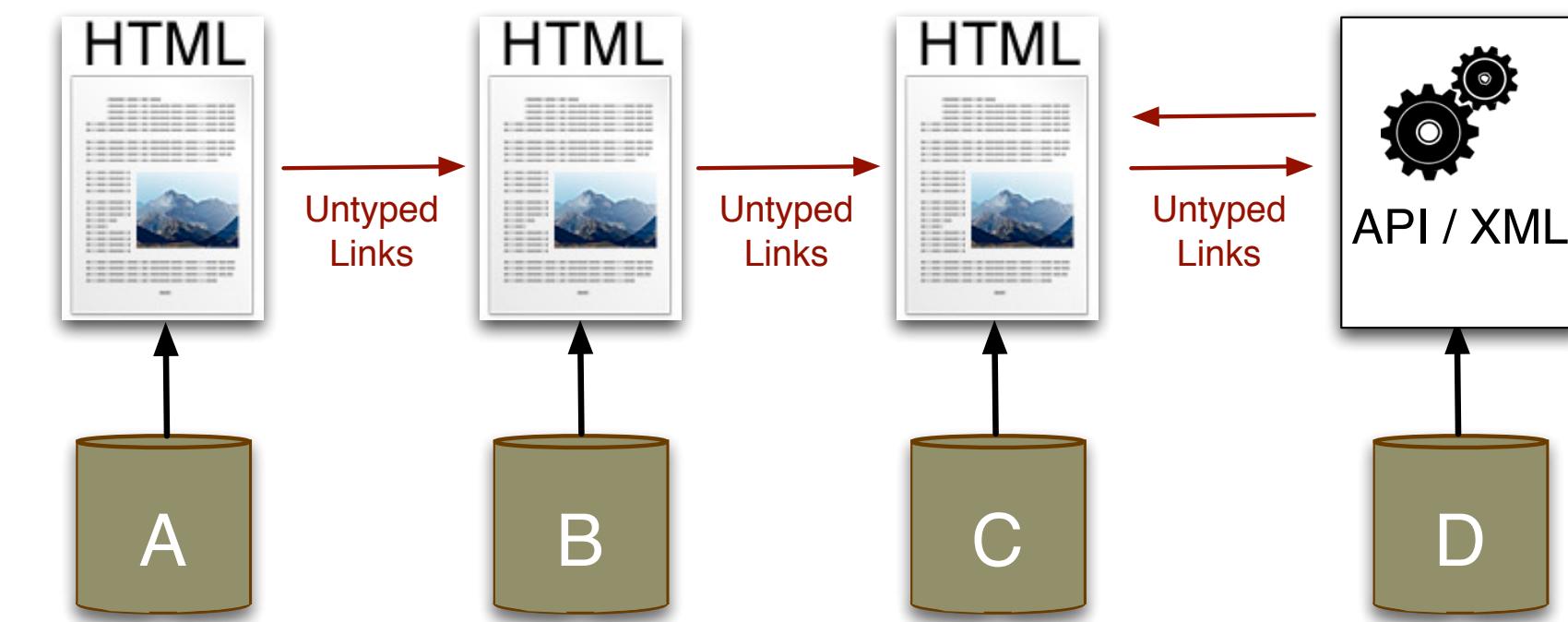
- Is this enough?
 - From a web of documents to
 - ***A web of data***



Web of Linked Documents

- The problem is that this model does not facilitate “sophisticated” data integration
 - Data integration
 - *Find all the papers published in Semantic Web related conferences in 2011*
 - Querying across different data sources
 - *Find all the papers published in WWW2011 that were authored by people from companies with less than 100 employees.*

?



Do we search or do we find?

- Ultimate goal is to have software that seamlessly can retrieve the information we need

Query: Football players who have played for Manchester City and scored a goal in the 2012 European Championship

Football players who have played for Manchester City and scored a



4,330,000 RESULTS

Narrow by language ▾

Narrow by region ▾

[Manchester City FC History, Players, Team, Information](#)www.123football.com/clubs/england/manchester-city/index.htm ▾

... of English football in 1999, to being in European ... to spare, City scored a goal to make it 5-1. City ... huge supporters of Manchester City and have played ...

[Ask the Gaffer | Football trivia, questions & answers | Orange UK](#)web.orange.co.uk/p/web_football/ask_the_gaffer ▾

... card and only three yellows - and he's scored a goal. ... Who are 9 players who have played for two of the Big Four (ManU ... foreign players Man City ...

[Archive \(part two\) | Football | guardian.co.uk](#)www.guardian.co.uk/football/2005/sep/21/theknowledge.sport ▾

21/09/2005 · ... European leagues? Players who've played for two clubs twice; Which football ... Manchester City; current Premiership players to have played ... Players who have scored ...

[Football players profiles - English footballers soccer careers](#)www.football-england.com/football_players_profiles.html ▾

Football players profiles - Accounts ... forever, wasn't he Scored a ponfull of goals. John Gidman Solid full back played for Aston Villa, Man Utd ... Francis Lee Roly-poly Man City ...

[Club Profile | Manchester City](#)www.premierleague.com/page/manchester-city-fc ▾

We have updated our cookie policy to reflect ... Last six results; BPL: 9 Feb 2013: v Southampton (A) 3 - 1 ... Official Manchester City Football Club website

[BBC Sport - Football - Championship](#)www.bbc.co.uk/sport/0/football/championship ▾

11/04/2013 · The latest BBC Championship news plus live scores, fixtures ... Watch video Leicester man hoping for goal boost ... position, team name, games played, total goal ...

[Manchester derby - Wikipedia, the free encyclopedia](#)en.wikipedia.org/wiki/Manchester_derby ▾

History · Statistics · Honours · Non-competitive derbies · All-time results

... football supporters in Manchester watched City ... City in the 24th and 45th minutes respectively, and Michael Carrick scored a consolation goal for ... Players who have played ...

[Arsenal FC vs Manchester City LIVE Commentary - Goal.com](#)www.goal.com/en-gb/match/60550/arsenal-fc-vs-manchester-city/play... ▾

... of football ... has scored a goal every 94 minutes this season, the best rate in the Premier League for all players who have played ... his Manchester City players. Having played ...

[Ask the Gaffer | Football trivia, questions & answers | Orange UK](#)web.orange.co.uk/p/football/ask_the_gaffer ▾

Web Images Maps Shopping More Search tools

About 27,700,000 results (0.41 seconds)

[Football news, results, live scores, rumours and fixtures - Goal.com](#)www.goal.com/en-gb/

Get the latest Football news, scores, results, rumours, fixtures and live action from the premier ... had earlier scored the vital second goal to send Manchester City into the FA Cup final ... Player Ratings: Chelsea 1-2 Manchester City » ... Europe's premier club competition returns as the knockout stages begin and there are ...

[Man City | Manchester City | Football Club News, Results ... - Goal...](#)www.goal.com/en/teams/england/109/manchester-city-fc

Europe Home · Europe News · England Home · England Table/Results · Italy Home ...

COMMENT: Brilliant Bayern show why the Champions League is theirs to lose » ...

PLAYER RATINGS: Manchester United 1-2 Manchester City » ... and admits he has

never watched a repeat of his side's famous defeat from last season ...

[Football News, Football Scores, Football Transfers ... - Goal.com](#)www.goal.com/en/

Get the latest worldwide football news, football scores, transfer rumours, football ...

The Portuguese star scored the second-fastest goal of his career before firing in ...

PLAYER RATINGS: Chelsea 1-2 Manchester City » The Bayern Munich defender has urged his team-mates to approach the Champions League tie against ...

[News for Football players who have played for ...](#)[Chelsea left playing catch-up as Manchester City superiority tells](#)[The Guardian \(blog\)](#) - 8 hours ago

Chelsea left playing catch-up as Manchester City superiority tells ... as he tumbled to the turf to score the first goal conceded by Manchester City in this ... The European champions have experienced it a few times during a season ... Premier League last year and are seven points apart in this season's table.

[Manchester City F.C. - Wikipedia, the free encyclopedia](#)en.wikipedia.org/wiki/Manchester_City_F.C.

Manchester City Football Club is an English Premier League football club based ...

The club has played at the City of Manchester Stadium since 2003, having ... League Championship, FA Cup, League Cup and European Cup Winners' ... the following season, despite scoring more goals than any other team in the division.

[Premier League - Wikipedia, the free encyclopedia](#)en.wikipedia.org/wiki/Premier_League

At the top of the English football league system, it is the country's primary football ...

The current champions are Manchester City, who won the title in the 2011–12 season.

... clubs playing in European competitions in 1990 (resulting in Manchester ... The first ever Premier League goal was scored by Brian Deane of Sheffield ...

What we were actually looking for

Create account Log in

Article Talk Read View source View history Search

Mario Balotelli

From Wikipedia, the free encyclopedia

Mario Barwuah Balotelli (Italian pronunciation: [maˈrjo baloˈtelli]; born Mario Barwuah; 12 August 1990) is an Italian footballer who plays as a striker for Milan and the Italian national team.^{[4][5]} He started his professional football career at Lumezzane and played for the first team twice before having an unsuccessful trial at FC Barcelona,^[6] and subsequently joining Internazionale in 2007. Inter manager Roberto Mancini brought Balotelli into the first team, but when Mancini left, Balotelli's disciplinary record fell away. He had a strained relationship with new head coach José Mourinho and was suspended from Inter's first team in January 2009 after a number of disciplinary problems.

In March 2010, he came under criticism by Inter fans after he appeared on the Italian TV show *Striscia la notizia*, wearing an Milan jersey. This damaged the prospect of him having a long career at Inter, but he did make several appearances after that. With doubts over his career at Inter, former coach Roberto Mancini had since moved to Manchester City and decided to give Balotelli a fresh chance at a new club. He joined Manchester City in August 2010, where his performances and off-field activities have continued to be enigmatic and unpredictable. Nicknamed Super Mario,^[7] he earned his first cap for the Italian national team on 10 August 2010 in a friendly match against the Côte d'Ivoire.

Contents [hide]

- 1 Early life
- 2 Club career
 - 2.1 Lumezzane
 - 2.2 Internazionale
 - 2.3 Manchester City
 - 2.3.1 2010–11
 - 2.3.2 2011–12
 - 2.3.3 2012–13
 - 2.4 Milan
- 3 International career
 - 3.1 UEFA Euro 2012
 - 3.2 2014 FIFA World Cup qualifying
 - 3.3 International goals
- 4 Style of play
- 5 Personality and reputation
- 6 Career statistics
 - 6.1 Club
 - 6.2 International
- 7 Honours
 - 7.1 Club
 - 7.2 International
 - 7.3 Individual
- 8 Outside football
 - 8.1 Personal life
- 9 References
- 10 External links

Early life

Mario Balotelli



Balotelli playing for Italy at the UEFA Euro 2012

Personal information

Full name	Mario Barwuah Balotelli ^[1]
Date of birth	12 August 1990 (age 22) ^[2]
Place of birth	Palermo, Italy
Height	1.89 m (6 ft 2 in) ^[3]
Playing position	Striker ^[2] / Forward

Club information

Current club	Milan	
Number	45	
Youth career		
2001–2006	Lumezzane	
Senior career*		
Years	Team	Appst (Gls)t
2005–2006	Lumezzane	2 (0)
2006–2010	Internazionale	59 (20)
2010–2013	Manchester City	54 (20)
2013–	Milan	7 (7)

National team

Was there any other player???

Why can't just we find it



BIBTEX



Mashups?

- **Data:** the value of observable, measurable or calculable attributes;

Data

- **Information:** is data plus conceptual commitments and interpretations (*context*) that provides some meaning.

- Information is data extracted, filtered or formatted in some way
- the context includes the class to which the attributes belong, the objects that are members of that class, object operations or behaviour, and the relationships to other objects and classes

Mashups

- **Knowledge:** an elusive concept. information having been processed, organised or structured in some way, or else as being applied or put into action.

- Knowledge as synthesis of information
- Knowledge as know how
- Knowledge as propositional (beliefs)

Information

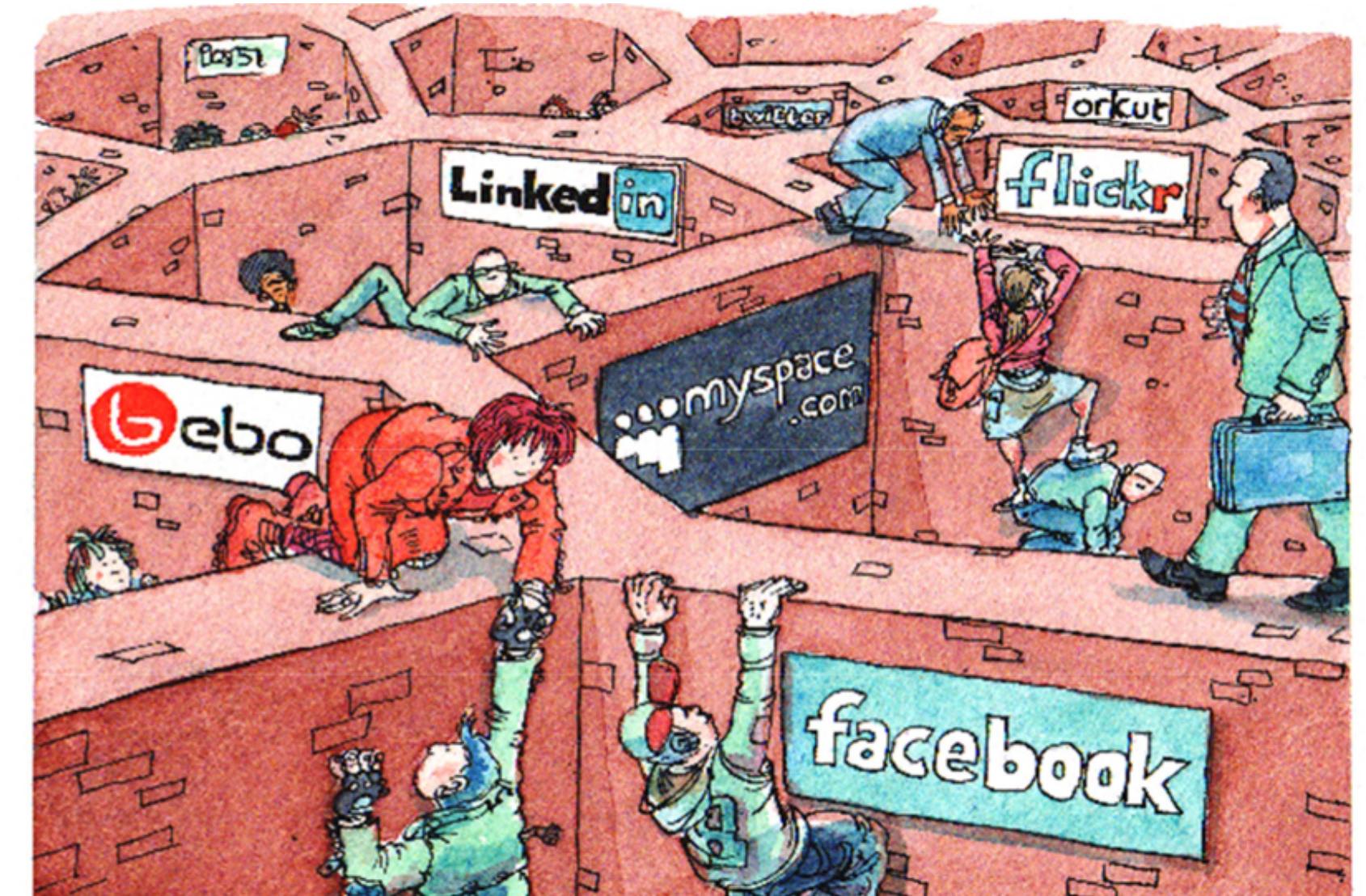
Knowledge

Mashups?

- Mashups are web pages or applications that utilise and combine data, presentation or functionalities from several data sources in order to create new services
 - *Use APIs provided by different content sites to aggregate and reuse the content in different ways*
 - *Combination, aggregation, visualisation...*
- Different types of mashups
 - **Business (or enterprise) mashups** define applications that combine their own resources, application and data, with other external Web services.
 - They are secure, visually rich Web applications that expose actionable information from diverse internal and external information sources.
 - **Consumer mashups** combines different data types. It combines data from multiple public sources in the browser and organises it through a simple browser user interface
 - Wikipediavision combines Google Map and a Wikipedia API
 - **Data mashups** combine similar types of media and information from multiple sources into a single representation. The combination of all these resources create a new and distinct Web service that was not originally provided by either source.

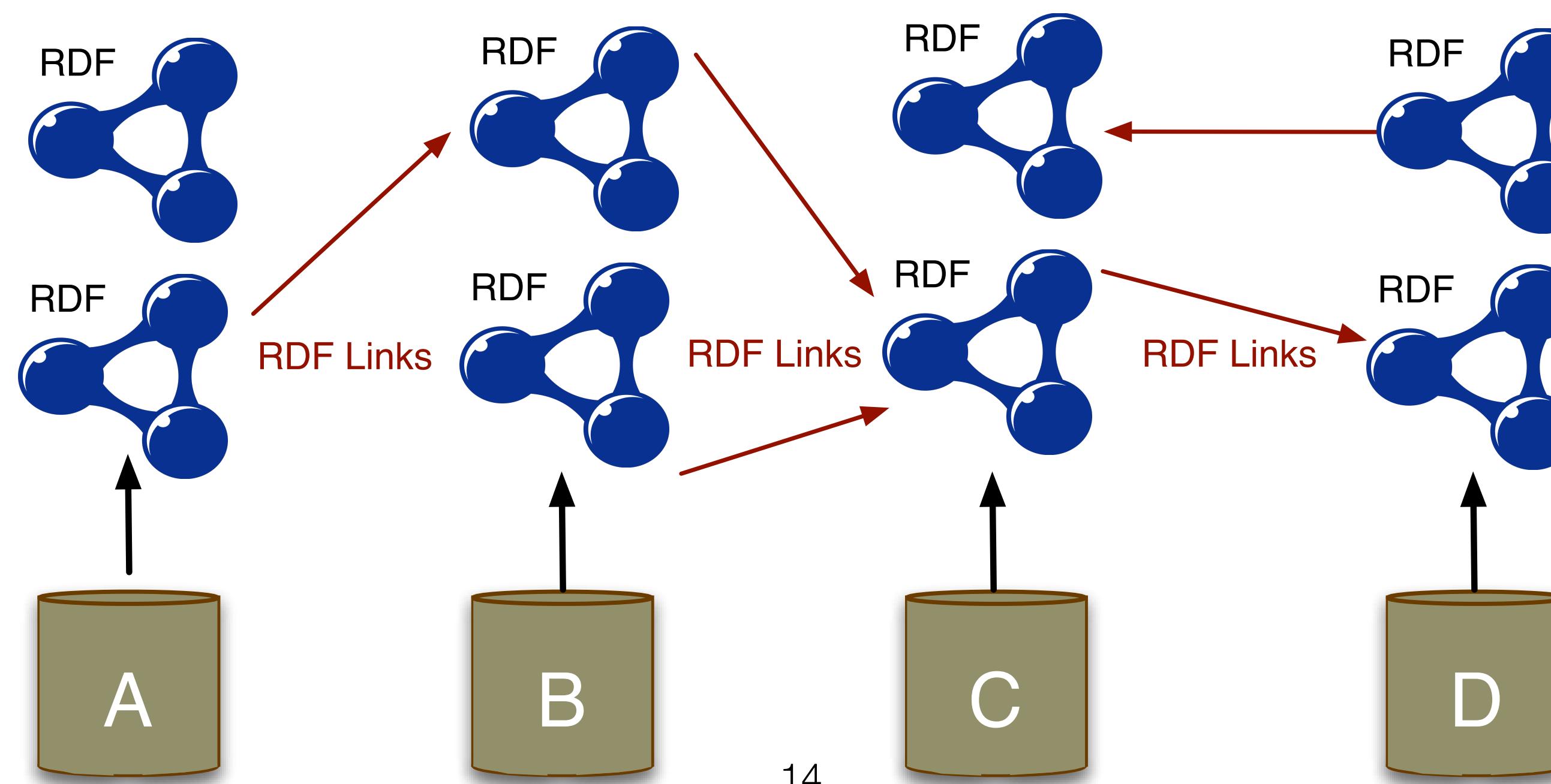
Problem with mashups and API

- API allow the mashing up of information but...
 - APIs have proprietary interfaces
 - Mashups are based on a fixed set of data sources
 - There are no **(hyper)links** that relate data items within different APIs
 - the data is disconnected
 - these are all information and data siloes



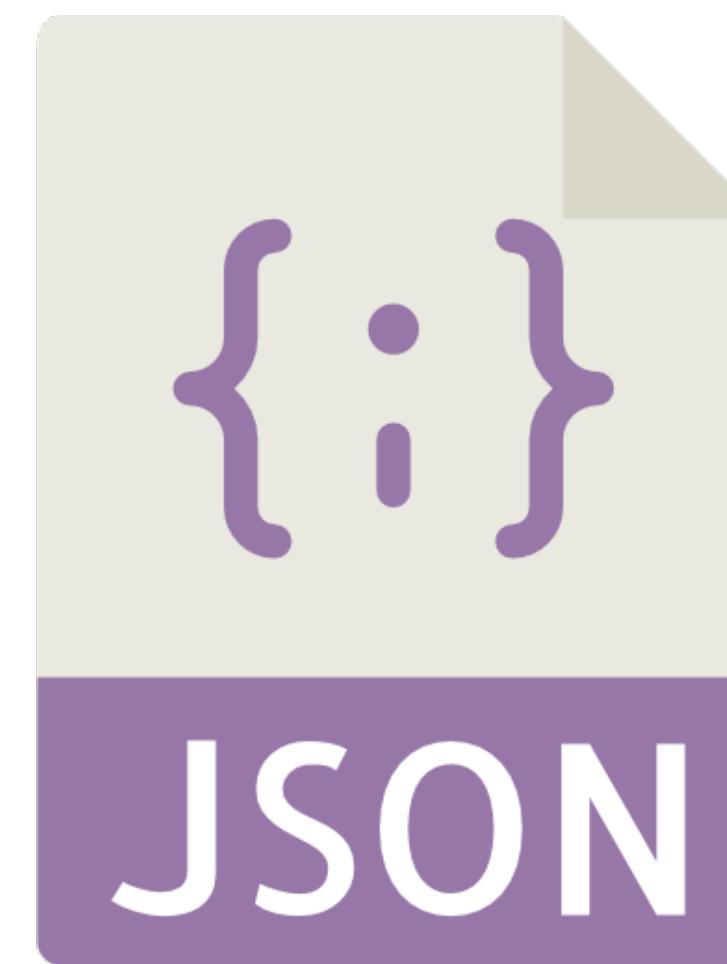
Linked data: the idea

- Use the Web to create a global dataspace
 - RDF as a means to publish structured data on the Web
 - state explicitly links between data items from different data sources
 - Link data rather than documents!



Web of data

- Isn't this what we are doing now?
 - CSV, XML, key-value data, relational data
- But it's about the *semantics* rather than the surface or exchange syntax



In Other Words

- “All kinds of conceptual things, they have names now that start with HTTP.”
- “I get important information back. I will get back some data in a standard format which is kind of useful data that somebody might like to know about that thing, about that event.”
- “I get back that information it's not just got somebody's height and weight and when they were born, it's got relationships. And when it has relationships, whenever it expresses a relationship then the other thing that it's related to is given one of those names that starts with HTTP.”

It is about:
a data (RDF)
and naming (URI)
model on Web

Tim Berners-Lee: Linked Open Data Presentation, TED 2009

What is linked data

- Linked Data refers to a set of best practices for publishing and interlinking structured data on the Web:
 - Based on 4 principles, following the web architecture:
 1. Use URIs as to identify things;
 2. Use HTTP URIs so that these things can be referred to and looked up ("dereferenced") by people and user agents.
 3. Provide useful information about the thing when its URI is dereferenced, using standard formats such as RDF/XML.
 4. Include links to related URIs in the exposed data to improve discovery of other related information on the Web.
 1. (a bit obvious, maybe) Link the data!



Tim Berners-Lee: <http://www.w3.org/DesignIssues/LinkedData>

1. Use URIs as to identify things

- Use URIs for naming everything you could think of:
 - URI references should be used to identify not just Web documents and digital content, but also:
 - real world objects and abstract concepts:
 - tangible things
 - e.g. people, places and cars,
 - more abstract things
 - e.g. the relationship type of knowing somebody, the set of all green cars in the world, or the colour green itself
- This principle extends the scope of the Web:
 - from online resources to encompass any object or concept in the world.

2. Use HTTP URIs allowing to refer and look up things on the Web

- HTTP protocol as the Web's universal access mechanism for people and user agents to dereference resources.
 - In the classic Web, HTTP URIs are used to combine globally unique identification with a simple, well-understood retrieval mechanism.
 - Thus, Linked Data advocates the use of HTTP URIs to identify objects and abstract concepts:
 - Dereferenceable URIs are recommended but not mandatory in linked data. When you do an HTTP GET (or type in a URL in your browser's address bar) something is returned, an RDF description in the best case!
 - Different servers are responsible for answering requests attempting to dereference HTTP URIs in many different namespaces.

3. Assign useful information to URIs using standards

- Provide useful information about the thing when its URI is dereferenced,
 - using standard formats such as RDF/XML or OWL
 - agreement on standardised content formats!
- Advocates the use of a single data model for publishing structured data on the Web
 - Resource Description Framework (RDF)
 - simple graph-based data model.

4. Include links to other URLs to improve discovery

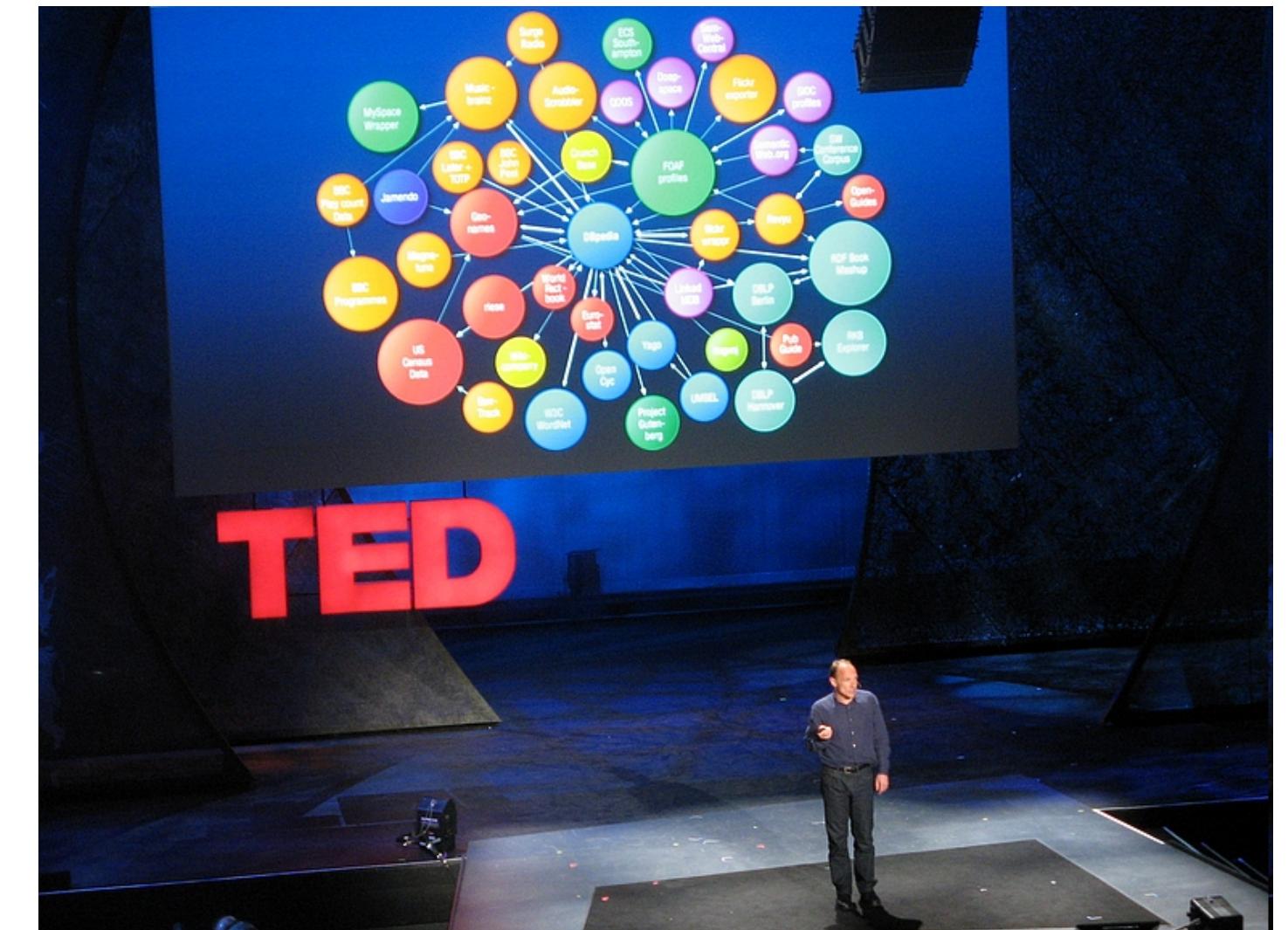
- Following from principle 1, hyperlinks connect not only Web documents, but any type of thing.
 - e.g. a hyperlink may be set between a person and a place, or between a place and a company.
- Hyperlinks that connect things in Linked Data have types which describe the relationship between the things.
 - Unlike hyperlinks on the Web.
 - e.g. a hyperlink of the type friend of may be set between two people, or a hyperlink of the type based near may be set between a person and a place.
 - When an RDF link connects URLs in different namespaces, it ultimately connects resources in different data sets
 - **These typed hyperlinks are RDF properties interpreted as hyperlinks**

4a. Link the data!

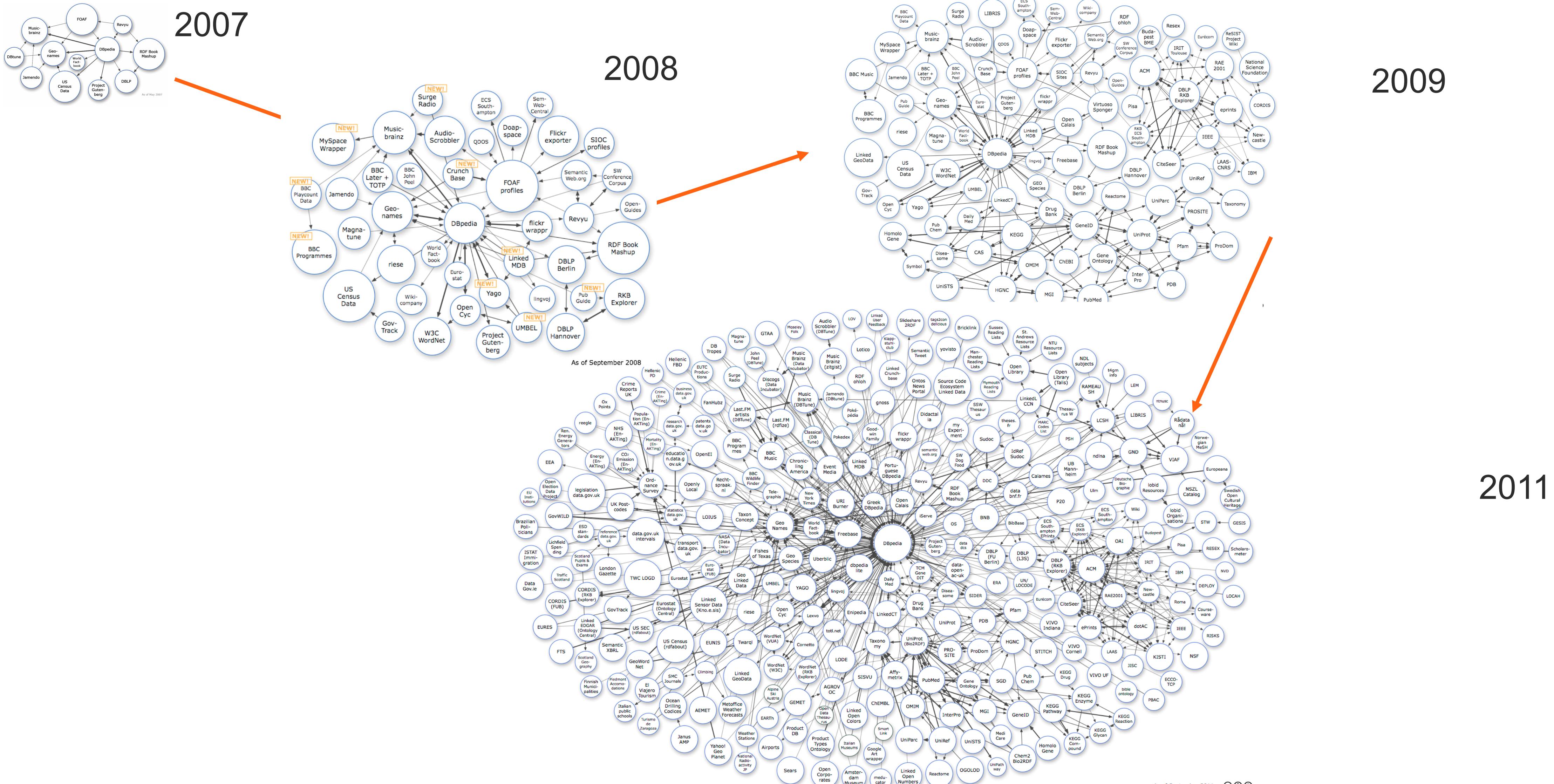
- Across the Web, many different servers are responsible for answering requests attempting to dereference HTTP URIs in many different namespaces:
 - In Linked Data, this means returning RDF descriptions of the resources identified by these URIs.
 - If an RDF link connects URIs in different namespaces, it ultimately connects resources in different data sets.
- Linked Data uses hyperlinks to connect disparate data into a single global data space.
 - just as hyperlinks in the classic Web connect documents into a single global information space,
- These links, in turn, enable applications to navigate the data space.
 - e.g. a Linked Data application that has looked up a URI and retrieved RDF data describing a person may follow links from that data to data on different Web servers, describing the place where the person lives or the company for which the person works.

But TBL was talking about open data i.e. when the sw meets open data

- Over the last few years, the Linked Data paradigm has found a huge application when combined with the publication of data with liberal licenses.
- The Open Data Movement, aims to release huge data sets often from local government authorities.
- It uses Linked Data technologies and best practices to publish a plethora of different interlinked data sets.
 - a bunch of data published with an open license is intended to be freely available to everyone to use and republished without restriction from copyright, patents or other restrictions.
- Where Open Data meets Linked Data, we have Linked Open Data.



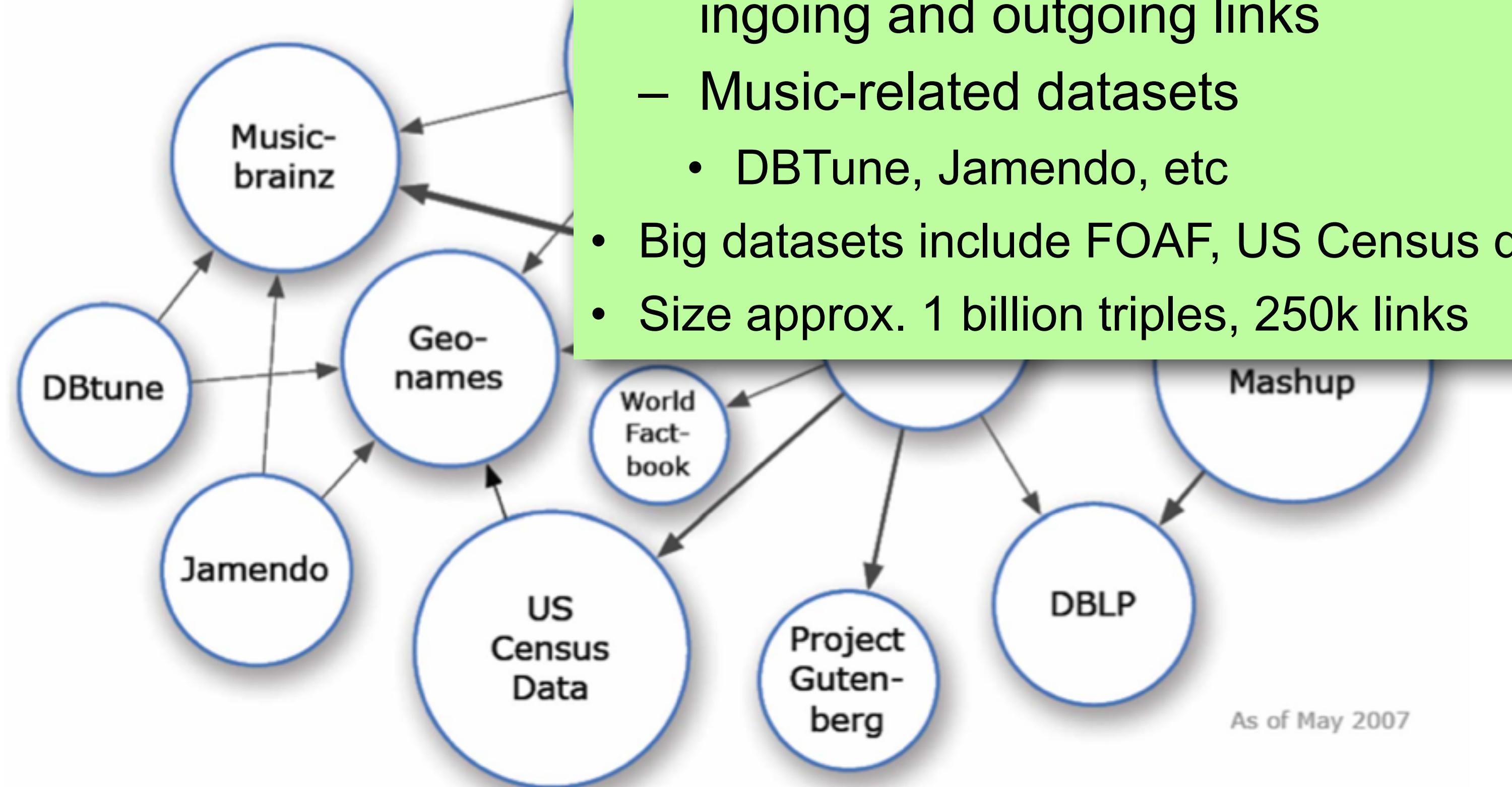
Linked Data Evolution: is there data out there?



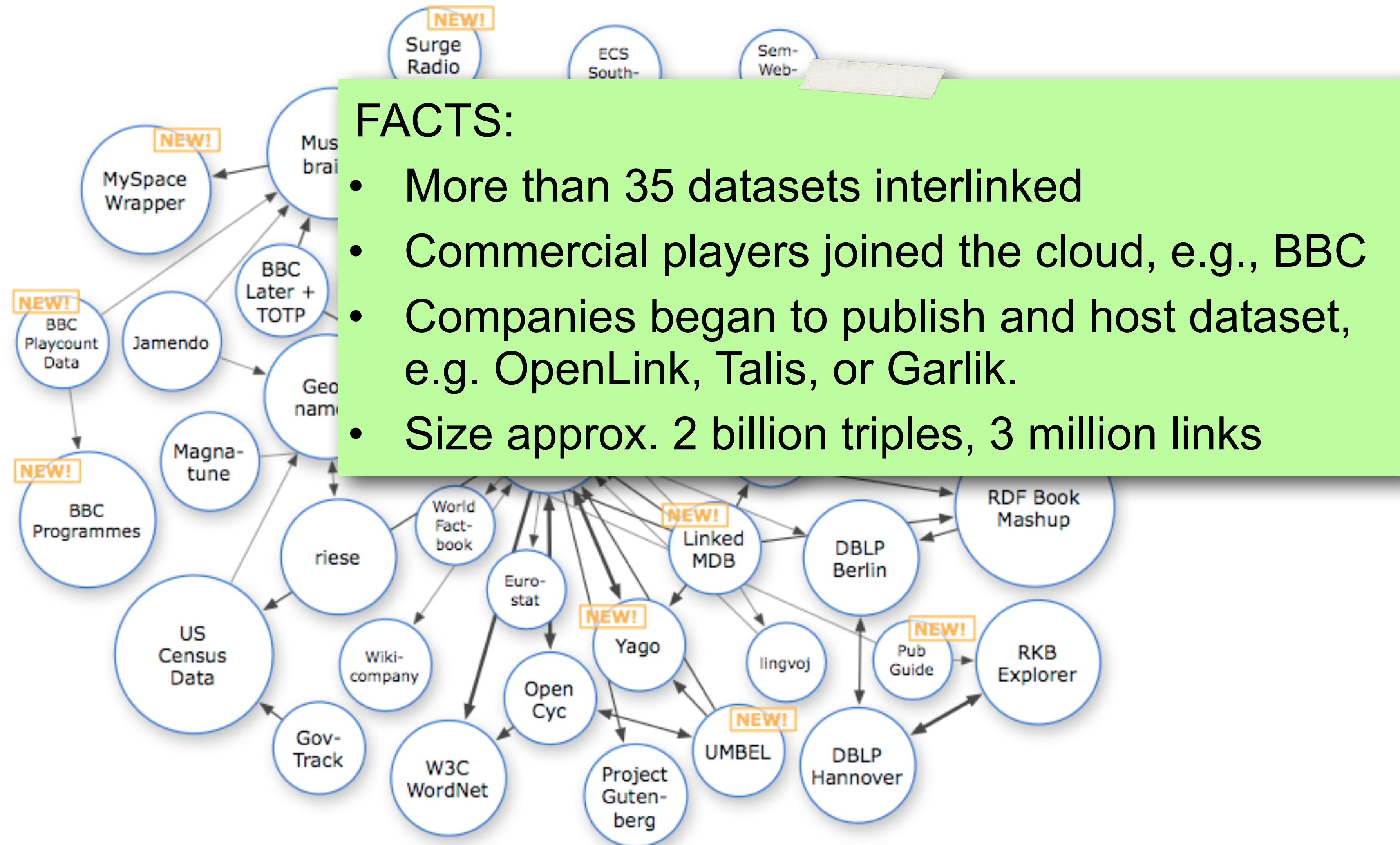
May 2007

FACTS:

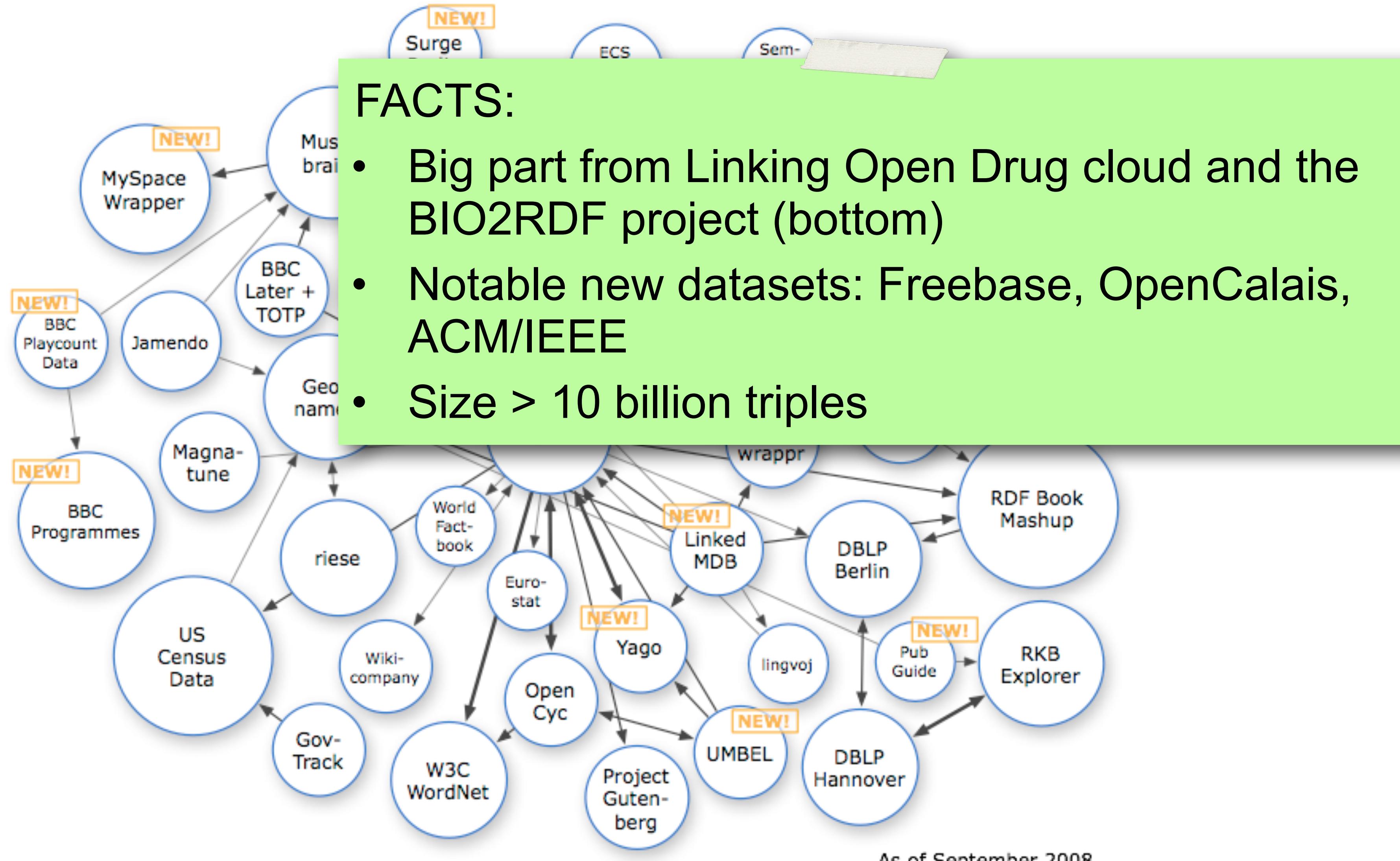
- Focal point based
 - DBpedia: RDFized version of Wikipedia; many ingoing and outgoing links
 - Music-related datasets
 - DBTune, Jamendo, etc
- Big datasets include FOAF, US Census data
- Size approx. 1 billion triples, 250k links



September 2008



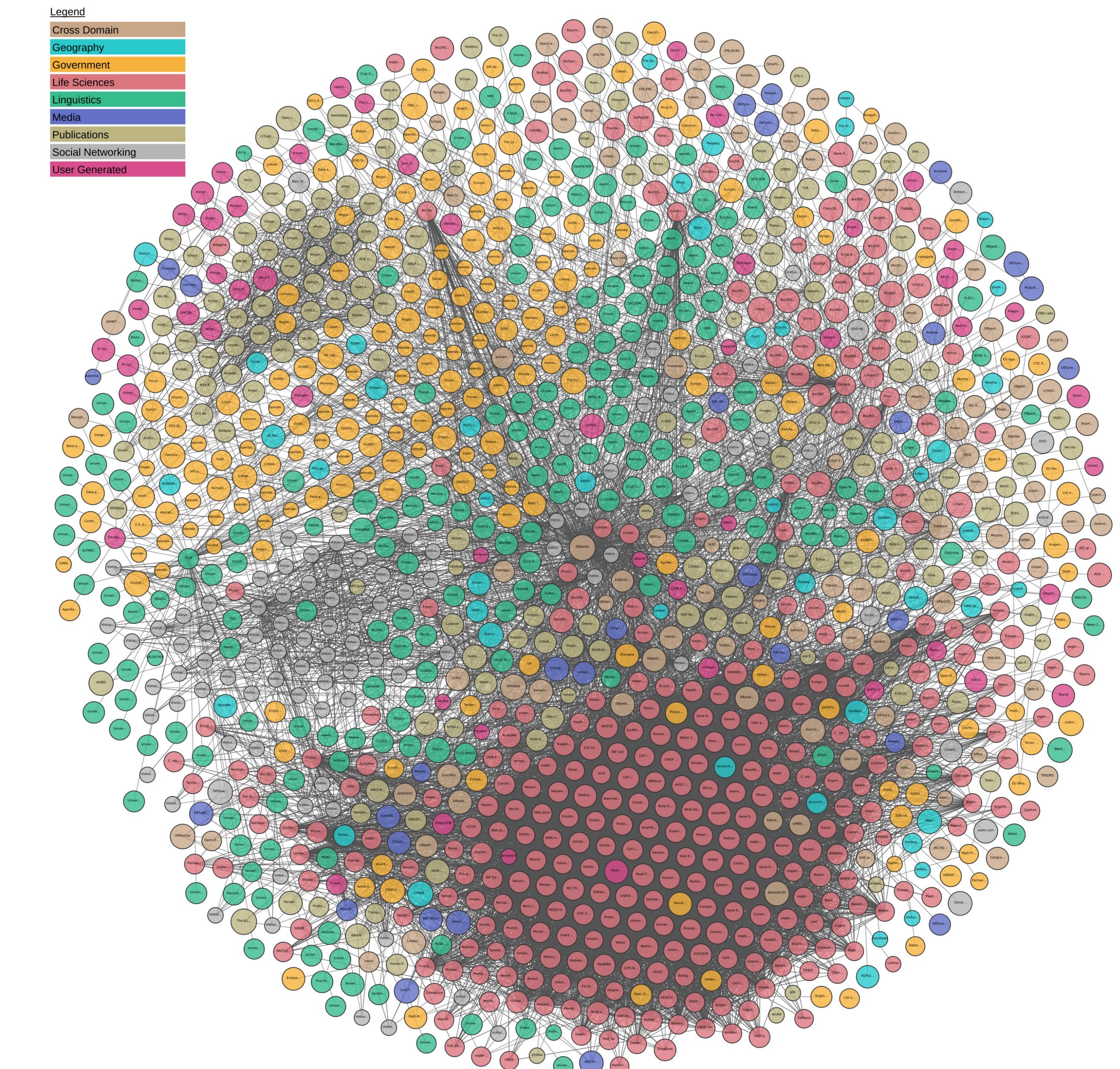
March 2009



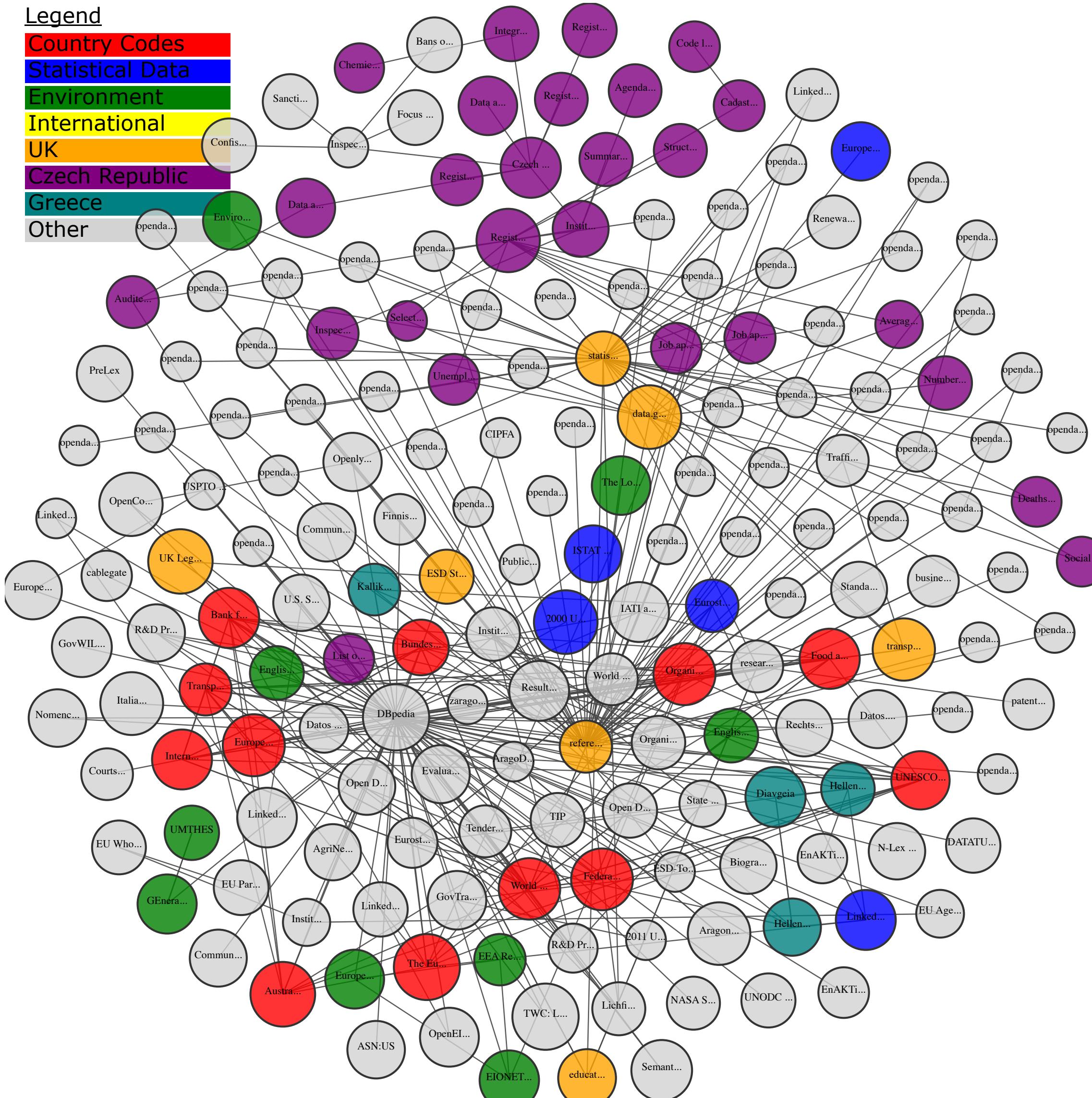
March 2019

FACTS:

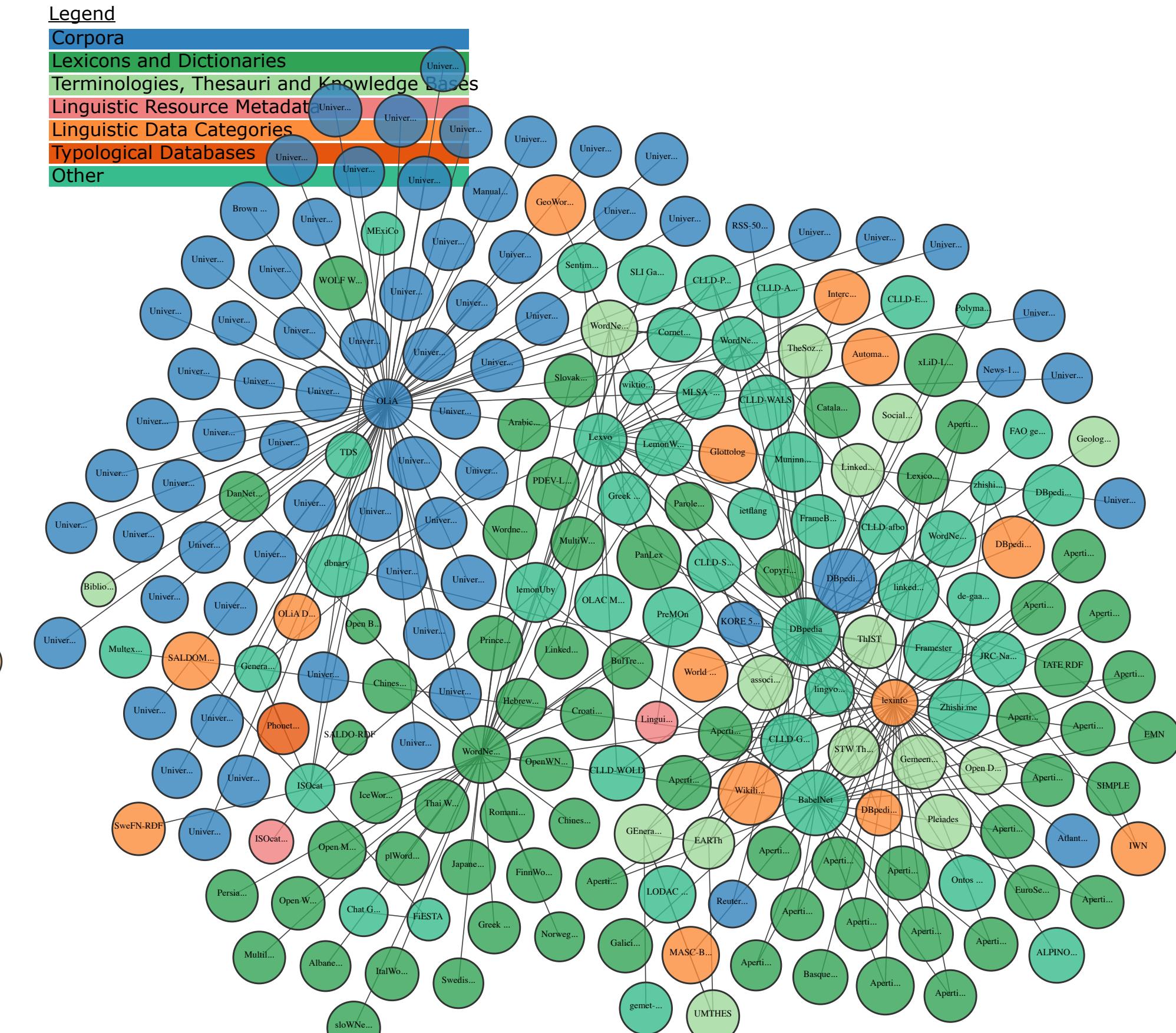
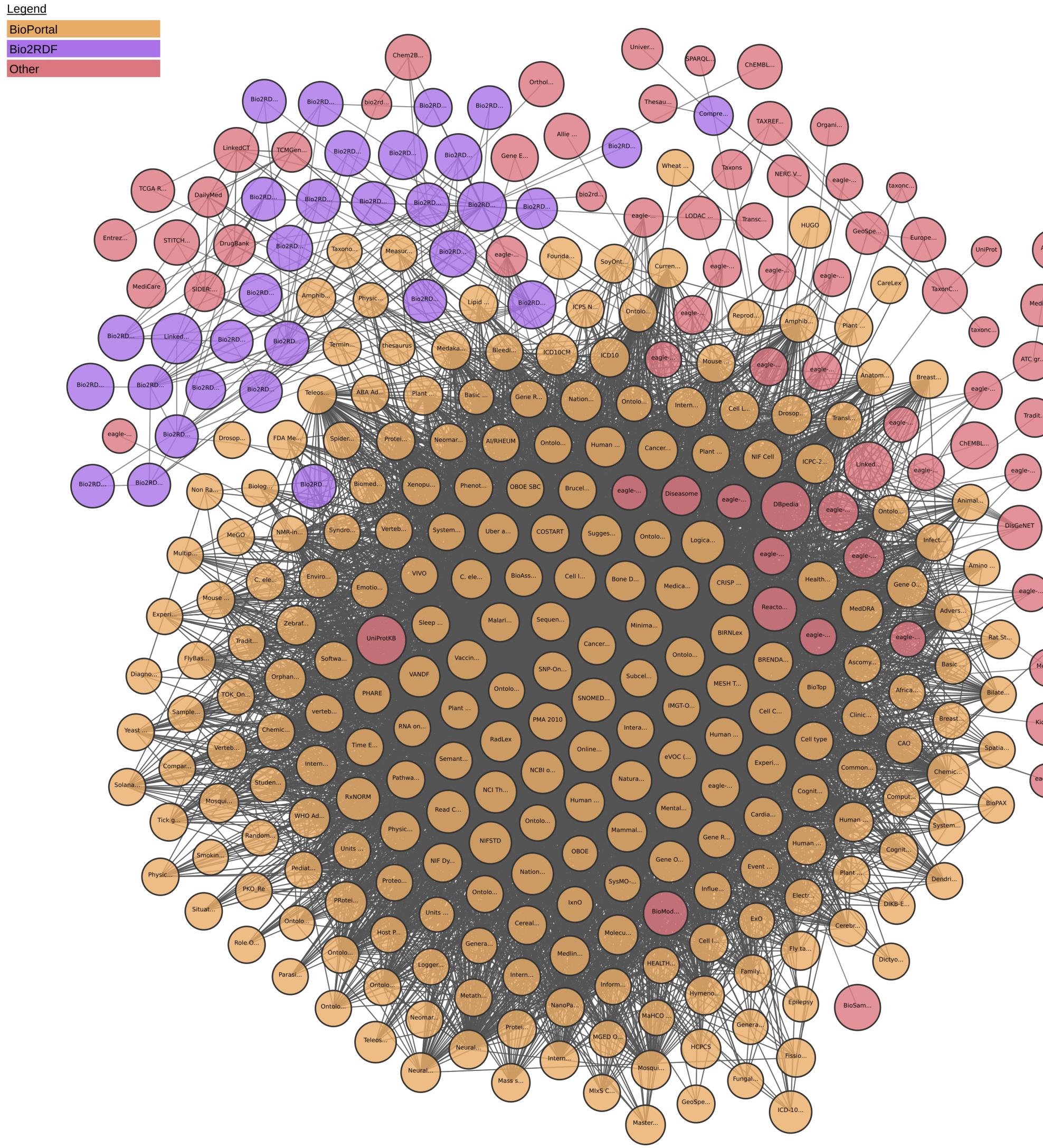
- 1,239 datasets with 16,147 links!



Government data



Largest domain LODs



The Linguistic Linked Open Data Cloud from lod-cloud.net



Where is the data

- Datasets
 - **UMBEL** - a lightweight reference structure of 20,000 subject concept classes and their relationships derived from OpenCyc, which can act as binding classes to external data; also has links to 1.5 million named entities from DBpedia and YAGO.
 - **FOAF** - a dataset describing persons, their properties and relationships
 - **VIAF (Virtual International Authority File)** - an aggregation of authority files (author names) from national libraries from around the world.
 - **Wikidata** - Wikidata is a free and open knowledge base, of more than 82,000,000 data items. It can be read and edited by both humans and machines.
 - **DBpedia** - a dataset containing extracted data from Wikipedia; it contains about 3.4 million concepts described by 1 billion triples, including abstracts in 11 different languages
 - **GeoNames** - provides RDF descriptions of more than 7,500,000 geographical features worldwide.

LOD applications building blocks

Using: Mashups

Mashups combine multiple datasets to create a new service, visualisation or information

Using: Search

Linked data search engines allow search across the web of data. Conventional search may present information derived from linked data.

Using: Productivity

Linked data facilitates data **integration** for business intelligence or research.

Storing and publishing

Linked data can be published in simple flat files on a web server, in databases with a translation layer, or in specialised 'triple stores' built to store and share linked data.

Querying: SPARQL

SPARQL Protocol and RDF Query Language provides a way to run structured queries over linked data datasets.

Representing: Vocabularies

Vocabularies provide lists (and definitions) of common terms that can be used to describe the things and relationships in a dataset.

Representing: Ontologies

Ontologies are vocabularies that record the logical relationships between their terms.

Identifying: URLs

Using HTTP Uniform Resource Locators (URLs) means that (a) data can be looked up across the Internet; (b) decisions about 'namespaces' for data are managed through the Domain Name System (DNS).

Interchanging: RDF

Resource Descriptor Framework (RDF) is a *model* for representing data as 'triples'. RDF can be serialised into a range of different file formats, including RDF-XML and text-based Turtle or N3 syntax.

Transporting: HTTP (The World Wide Web)

Data is hosted on servers that can talk Hypertext Transfer Protocol (HTTP) to each other and to browsers in order to exchange data across the Internet.

Licensing: open data

Open data is made available under licenses (or is placed in the public domain) so that others can use and build upon it, free of legal restrictions. Open standards for data files and interchange are used also.

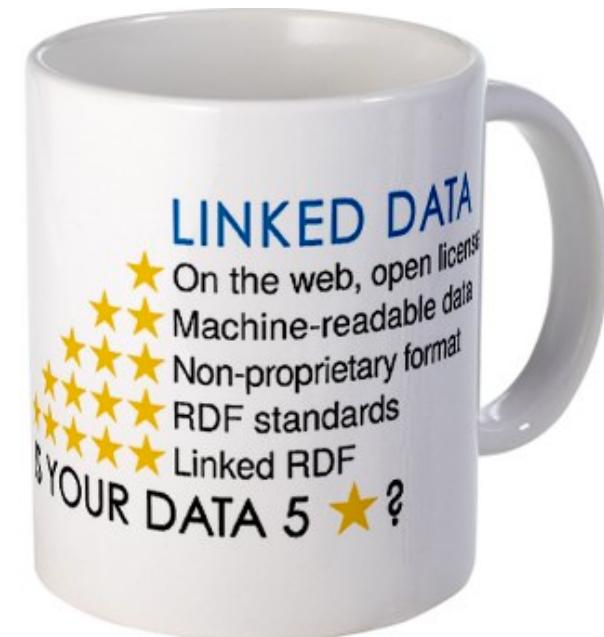
F. Bauer, M. Kaltenböck
Linked Open Data: The Essentials
A quick Start Guide for Decision Makers

Elements of the Linked Open Data Puzzle (revision 2) - 2nd May 2011. CC BY-SA-NC

Draft sketch by Tim Davies (@timdavies / tim@practicalparticipation.co.uk) for IKM Working Paper on Linked Open Data for Development. Comments welcome. Search 'linked open data puzzle' on <http://www.opendataimpacts.net> for latest version.

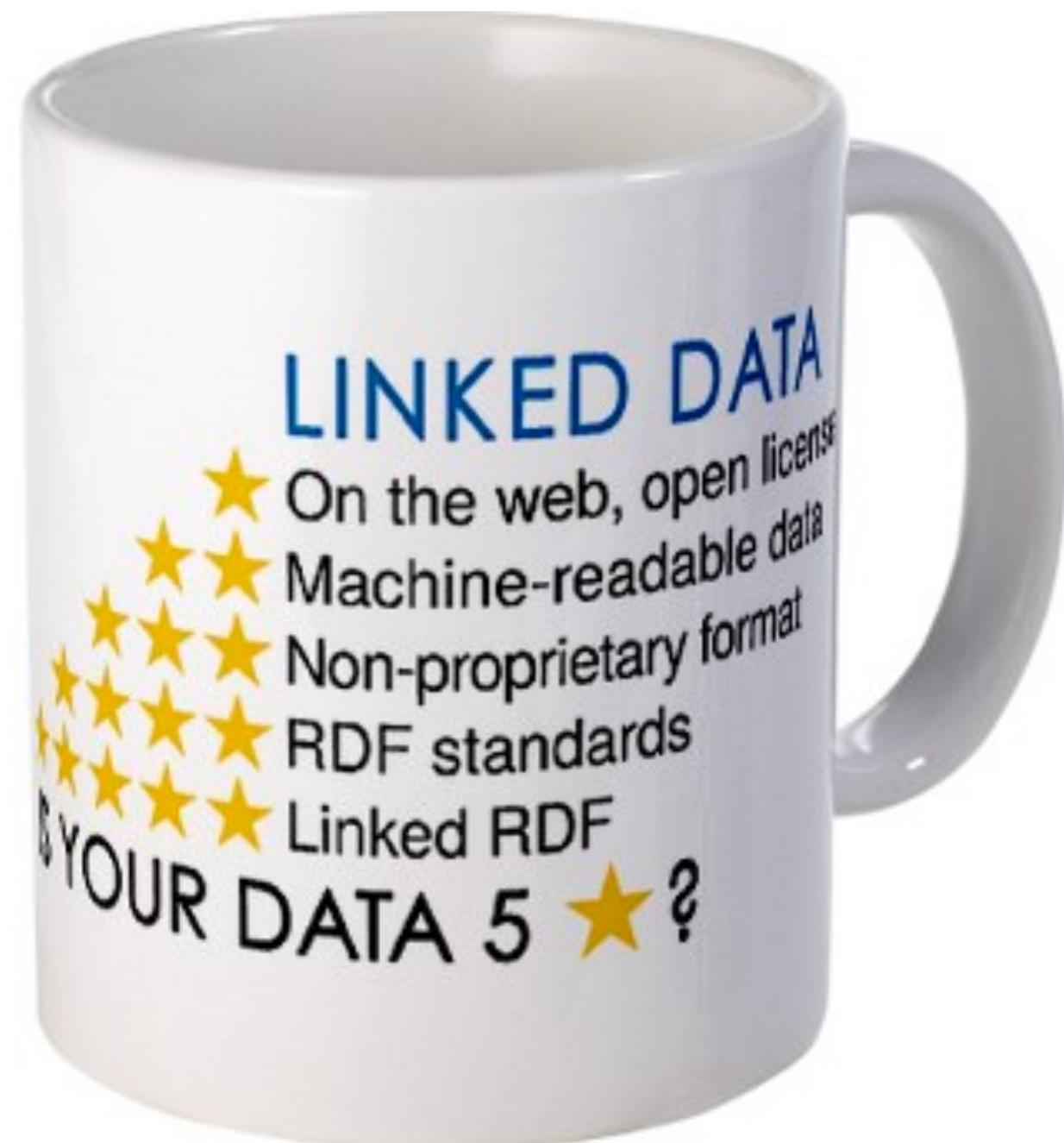
How do we publish Linked Data

- Exposing Relational Databases or other similar formats into Linked Data
 - D2R
 - Triplify
 - R2O
 - NOR2O
 - Virtuoso
 - Ultrawrap
 - ...
- Using native RDF triplestores
 - Sesame
 - Jena
 - OwlIM
 - ...
- Incorporating it in the form of RDFa in CMSs like Drupal



How do we consume Linked Data

- Linked Data browsers: To explore things and datasets and to navigate between them.
 - Tabulator Browser (MIT, USA), Marbles (FU Berlin, DE), OpenLink RDF Browser (OpenLink, UK), Zitgist RDF Browser (Zitgist, USA), Disco Hyperdata Browser (FU Berlin, DE), Fenfire (DERI, Ireland)
- Linked Data mashups: Sites that mash up (thus combine Linked data)
 - Revyu.com (KMI, UK), DBtune Slashfacet (Queen Mary, UK), DBPedia Mobile (FU Berlin, DE), Semantic Web Pipes (DERI, Ireland)
- Search engines: To search for Linked Data.
 - Falcons (IWS, China), Sindice (DERI, Ireland), MicroSearch (Yahoo, Spain), Watson (Open University, UK), SWSE (DERI, Ireland), Swoogle (UMBC, USA)



BETA This is a new service – your [feedback](#) will help us to improve it

Search results

Filter by

Publisher

Topic

Format

Open Government Licence
(OGL) only

254 results found

Best match ▾

[Organogram of Staff Roles & Salaries](#)

Published by: Serious Fraud Office

Last updated: 18 October 2016

Organogram (organisation chart) showing all staff roles. Names and salaries are also listed for the Senior Civil Servants. Organogram data is released by all central government departments and...

[Organogram of Staff Roles & Salaries](#)

Published by: UK Supreme Court

Last updated: 18 October 2016

Organogram (organisation chart) showing all staff roles. Names and

legislation.gov.uk

- Official government archive of the UK, managed by National archives.
- Best practice example of Open Government Data
 - Portal with access to all published UK legislation
 - Data from 1267 onwards, including recent changes in UK legislation
- Developed following W3C standards
- Persistent URIs for legislative data
 - identifier URIs, document URIs and representation URIs.

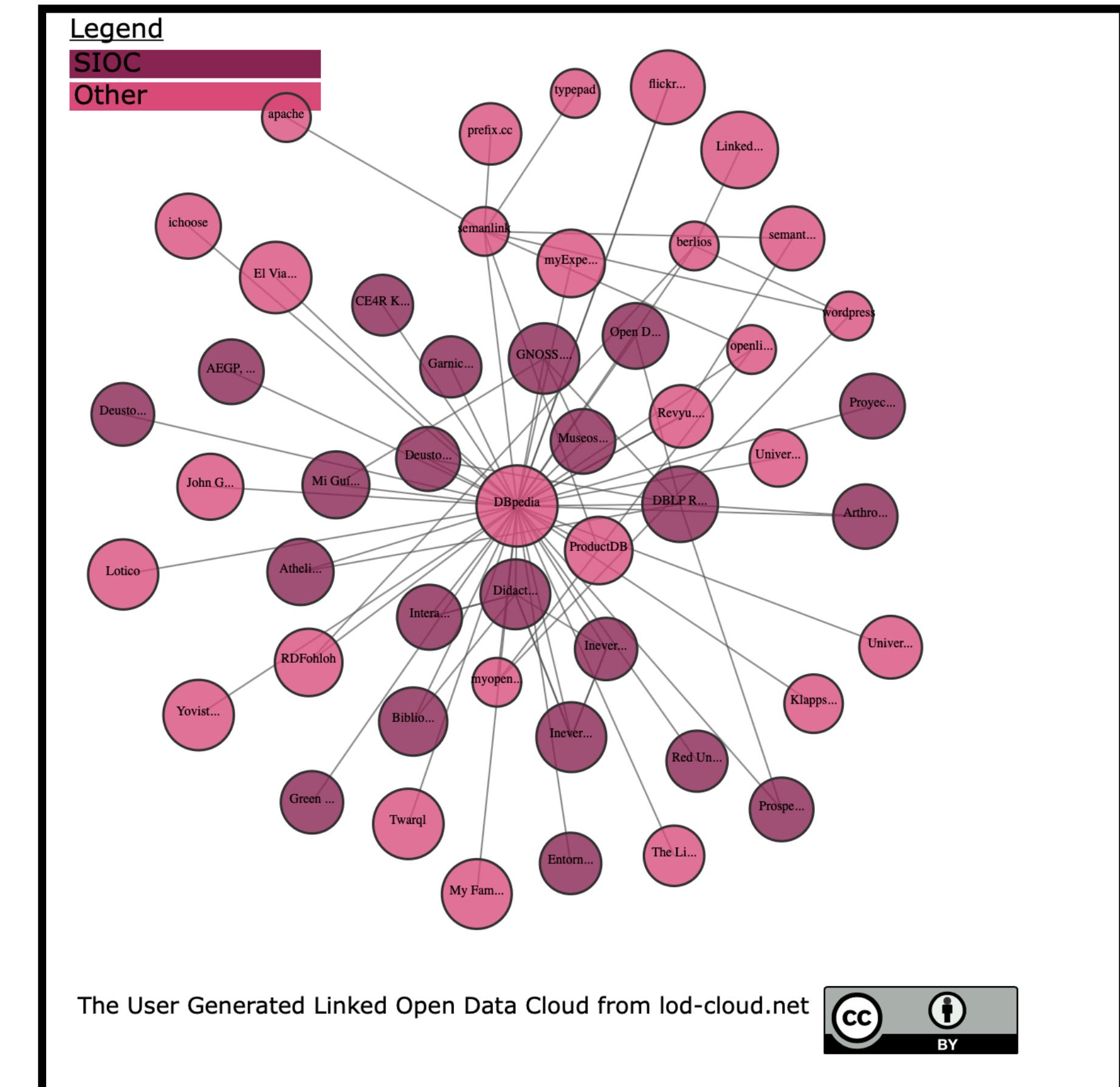
The screenshot shows the legislation.gov.uk homepage with a navigation bar at the top. Below the navigation, there's a search bar and a section titled 'URIs' which describes the Uniform Resource Identifier scheme used for legislation. It defines three levels of URLs: identifier URLs, document URLs, and representation URLs, with examples provided for each.

Is it all good

- Well, there are downsides...
 - The quantity of published Linked Data increases day by day, but...
 - Some of the data available might be either irregularly updated, or already available in other formats and APIs often becomes an issue
 - This is not happening with all the datasets, but it needs to be taken under consideration.
 - Additionally, more data needs to be available to share, extent and re-use.
- Data should be urgently published as Linked Data on the Web with appropriate licenses and provenance information.
 - Risk of creating RDF silos.
- More applications and tools needed to exploit Linked Data.
- Existing open issues make the development of Linked Data based applications a challenge:
 - difficulties to integrate data in different formats and from multiple sources,
 - the discovery of data
 - or the usability of user interfaces.

How to create Linked Data

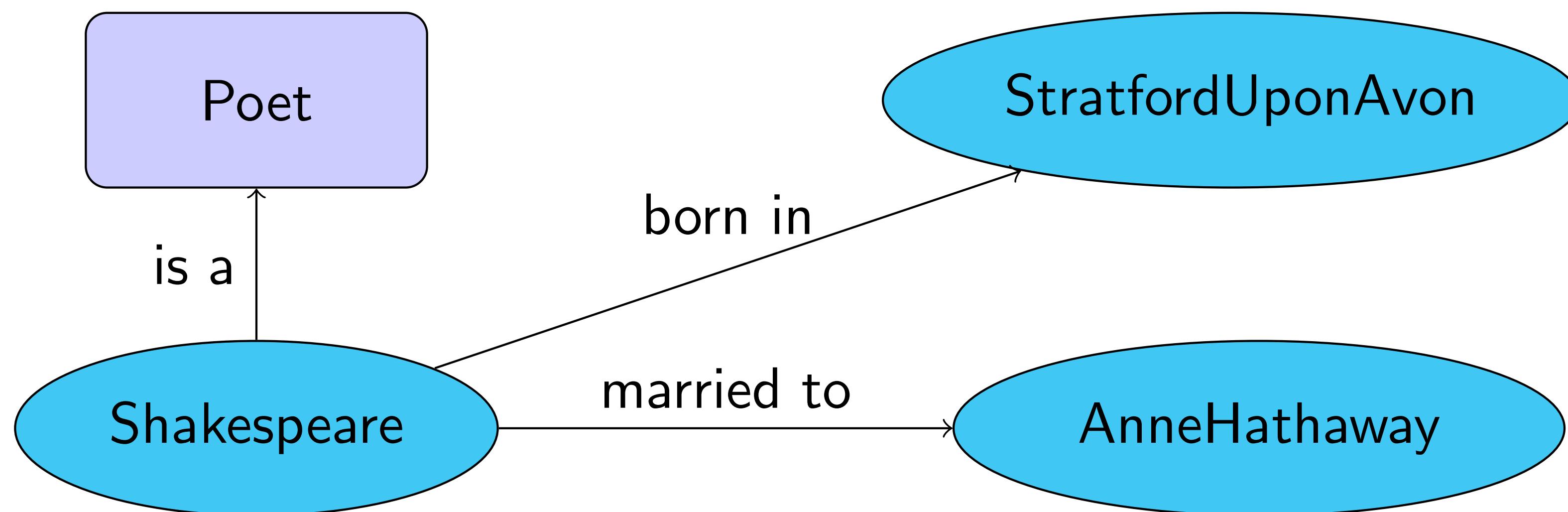
- 4-step creation process
 - Add semantics to the data
 - Model general world knowledge
 - Acquire new knowledge from inference
 - Query consistent information from different sources



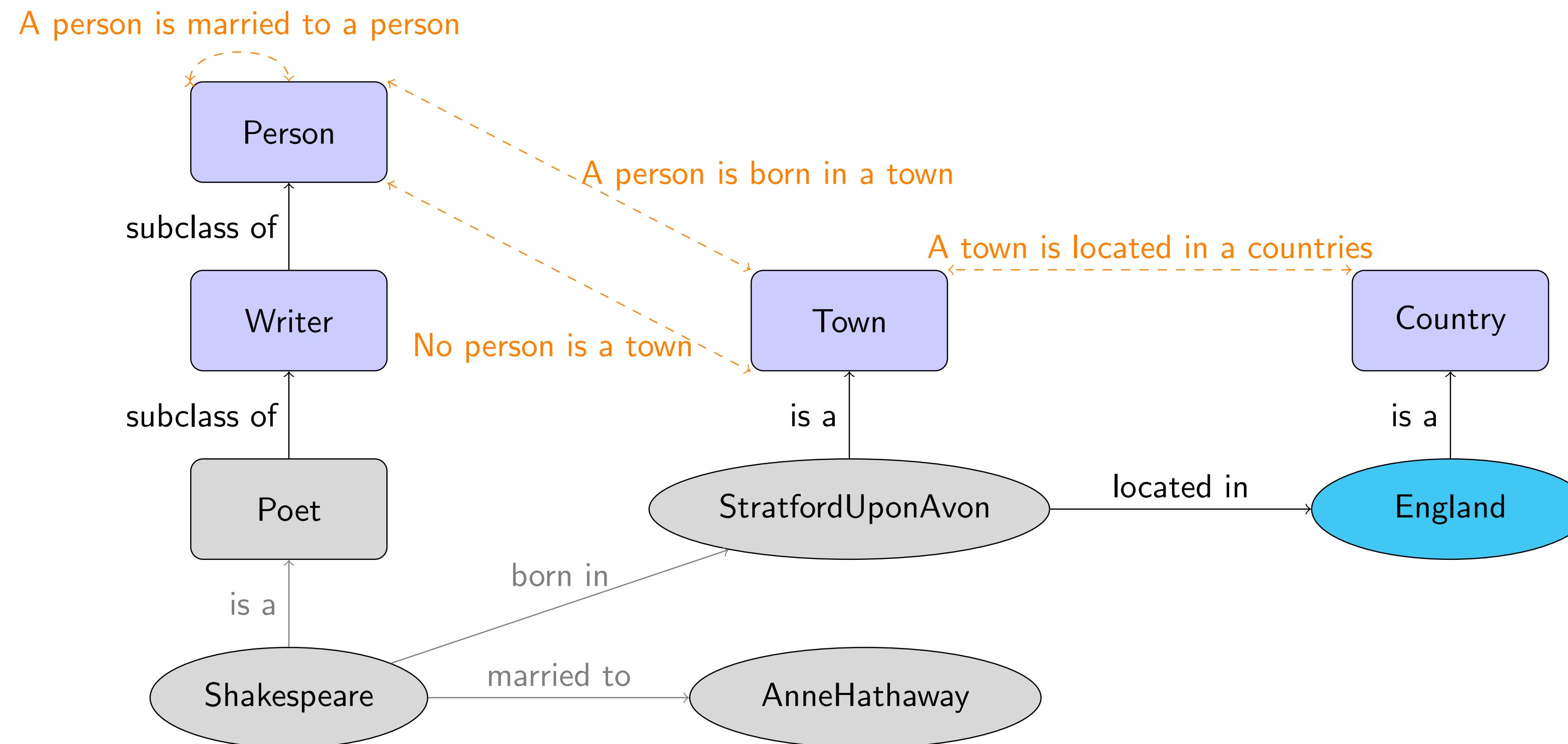
Step 1: Add semantics to the data

William Shakespeare was an English poet and playwright. Shakespeare was born and brought up in Stratford-upon-Avon. At the age of 18, he married Anne Hathaway, with whom he had three children: Susanna, and twins Hamnet and Judith. ...

[Wikipedia]



Step 2: Model general world knowledge



Step 3: Acquire new knowledge from inference

- Is Shakespeare a writer?

- facts we know:
 - Shakespeare is a poet
 - Every poet is a writer

- facts we can infer or deduce →

- - Shakespeare is a writer

- Is Anne Hathaway a person?

- - Shakespeare is married to Anne Hathaway
 - A person is married to a person

-

- - Anne Hathaway is a person

- Is Anne Hathaway a town?

- - Anne Hathaway is a person
 - No person is a town

- →

- - Anne Hathaway is NOT a town

Step 4. Query information from different sources

- List all persons . . .
 - from Wikipedia.
 - from IMDB.
 - from Wikidata.
 - from British Library
- . . .
- List all people who are authors on a paper together with a co-author of Paul Erdoss.
 - from FOAF-profiles on the web
- List all stores who sell fair trade items from Ecuador.
- List all doctors who treat dermatitis.

Recap

- Motivation for linked data
- Link data principles
- Link data adoption
- Designing of linked data

COMP318: Ontologies and Semantic Web

Describing Web Resources in RDF



Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

v.Tamma@liverpool.ac.uk

Recap

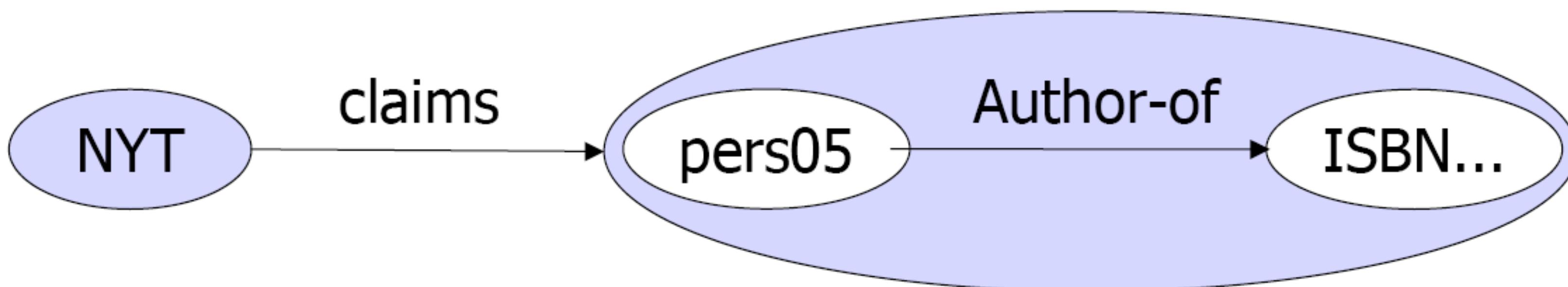
- Limitation of XML
- RDF
 - Basic ideas behind RDF
 - Statements about resources
 - triples
 - graphs
 - XML vocabularies
 - Modelling primitives in RDF
 - Reification

Higher order statements

- RDF allows you to make statements about other RDF statements
 - “Ralph believes that the web contains one billion documents”
- Higher-order statements
 - allow us to express beliefs (and other modalities)
 - are important for trust models, digital signatures,etc.
 - also: metadata about metadata
 - are represented by modelling RDF in RDF itself
- Reification

Reification

- Any RDF statement can be an object
- We must be able to refer to a statement using an identifier
 - allows users to point to a particular statement (and part of a graph)
- RDF allows such reference through a reification mechanism which turns a statement into a resource
 - newer versions of RDF introduce named graphs where an identifier is assigned to a set of statements



Reification Example

```
<rdf:Description rdf:about="#949352">  
    <uni:name>John Smith</uni:name>  
</rdf:Description>
```

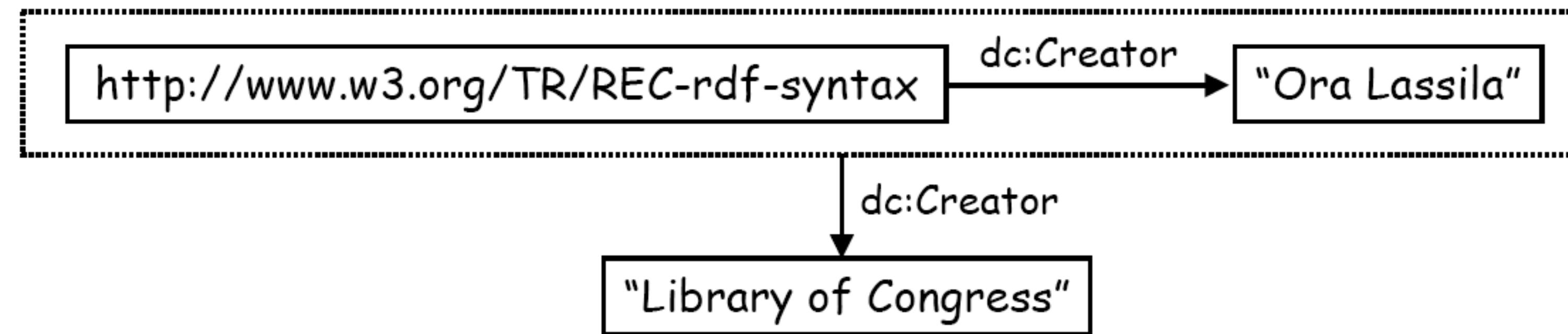
```
<rdf:Statement rdf:ID="StatementAbout949352">  
    <rdf:subject rdf:resource="#949352"/>  
    <rdf:predicate rdf:resource=  
        "http://www.jsmydomain.org/uni-ns#name"/>  
    <rdf:object>John Smith</rdf:object>  
</rdf:Statement>
```

Reification (2)

- `rdf:subject`, `rdf:predicate` and `rdf:object` allow us to access the parts of a statement
- The ID of the statement can be used to refer to it, as can be done for any description
- We write an `rdf:Description` if we don't want to talk about a statement further
- We write `rdf:Statement` if we wish to further refer to a statement

Reification (3)

- Thus, RDF provides a built-in vocabulary for reification



```
{ x, rdf:predicate, dc:creator }
{ x, rdf:subject, "http://www.w3.org/TR/RED-rdf-syntax" }
{ x, rdf:object, "Ora Lassila" }
{ x, rdf:type, "rdf:statement" }
{ x, dc:Creator, "Library of Congress" }
```

Representing the RDF language

- RDF document is a collection of triples

```
subject, predicate, object  
subject, predicate, object  
subject, predicate, object  
...  
...
```

- subject: URI or bnode
- predicate: URI
- object: URI or bnode or literal

*How do we represent these - and share them
between applications/users/etc...*

RDF Serialisation formats

- RDF has been given a syntax in XML
 - This syntax inherits the benefits of XML
 - Other serialisations of RDF possible:
 - Notation 3 (N3)
 - Syntax for RDF
 - Logical language for RDF
 - N-Quads
 - Superset of N-triples for serialising multiple RDF graphs
 - Turtle
 - Refinement of N3
 - Just RDF representation
 - JSON-LD
 - JSON based serialisation

Terse RDF Triple Language

- Turtle
 - Refinement of N3
 - Just RDF representation
- Plain text syntax for RDF
 - Based on Unicode
 - RDF 1.1 turtle recommendation in 2014
- Mechanisms for namespace abbreviation
 - Allows grouping of triples according to subject
- Shortcuts for collections
- In short:
 - Takes good things of RDF/XML
 - and leaves out angle brackets

Prefixes

- Mechanism for namespace abbreviation

- Syntax:

```
@prefix abbr: <URI>
```

- Example:

```
@prefix rdf:  
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

- Default:

```
@prefix : <URI>
```

- Example:

```
@prefix : <http://example.org/myOntology#>
```

Identifiers in Turtle

- URIs: <URI>

<<http://www.w3.org/1999/02/22-rdf-syntax-ns#>>

- Qnames (Qualified names): namespace-abbr?:localname

`rdf:type dc:title :hasName`

- Literals: "string" (@lang)? (^?type)?

"John" "Hello"@en-GB "1.4"^^xs:decimal

- Typed literal shortcuts

- integer: 2 45
- decimal: 2.4 5.67
- boolean: true false

Triples in Turtle

- Simple triple: subject predicate object .
- :john rdf:label "John".
- Grouping triples: subject predicate object ; predicate object ...

```
:john
  rdf:label "John" ;
  rdf:type ex:Person ;
  ex:homePage <http://example.org/johnspage/>.
```

Blank Nodes in Turtle

- Simple blank node: [] or _:x

```
:john ex:hasFather [ ] .
```

```
:john ex:hasFather _:x .
```

- Blank node as subject: [predicate object; predicate object ...]

```
[ ex:hasName "John" ] .
```

```
[ ex:authorOf :lotr ;
```

```
ex:hasName "Tolkien" ] .
```

- Collections: (object1 ... objectn)

```
:doc1 ex:hasAuthor  
( :john :mary ) .
```

Short for

```
:doc1 ex:hasAuthor  
[ rdf:first :john;  
rdf:rest  
[ rdf:first :mary;  
rdf:rest rdf:nil ]  
] .
```

More on URLs

- URI = Uniform Resource Identifier
 - allow for denoting resources in a general, unambiguous way
- Resource: any object that possesses a clear identity (within the context of a given application)
 - books, cities, humans, publishers, but also relations between those, abstract concepts, etc.
- already realised in some domains:
 - ISBN for books
- URIs do not need to correspond to an actual location
 - but it is good practice if they do
 - a picture, a FOAF description, a map...

Syntax of URIs

- Builds on concept of URLs but...
 - not every URI refers to a Web document
 - however, often the URL of a document is used as its URI
- URI starts with so-called URI schema separated from the following part by ":"
 - e.g, http, ftp, mailto
 - but starting with http does not necessarily mean http-accessible...
- mostly hierarchically organised

Self-defined URIs?

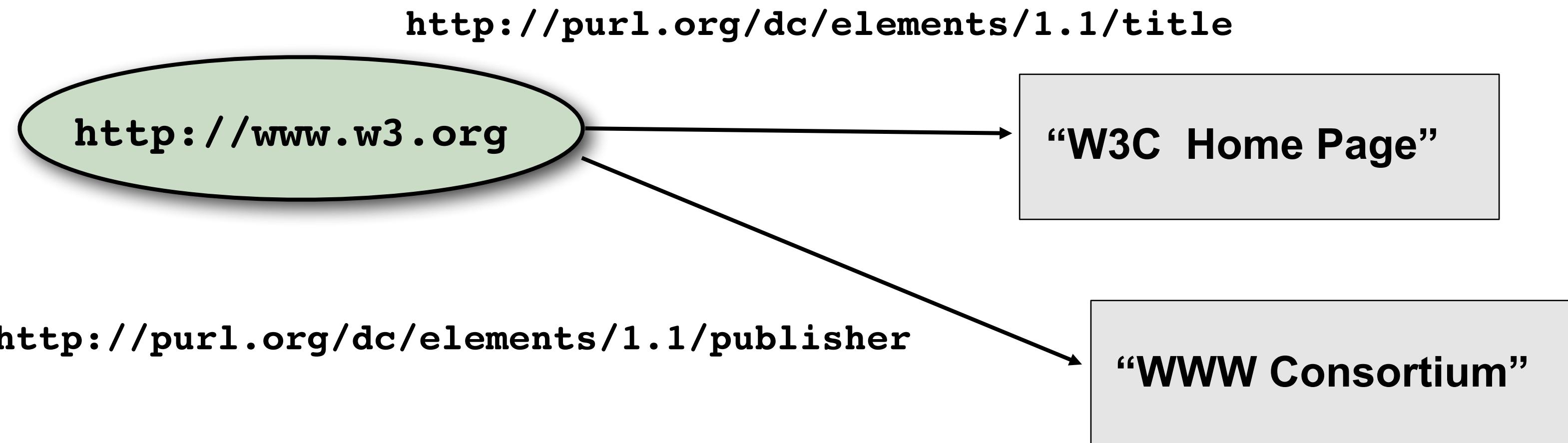
- Necessary if no URI exists (yet) for a resource
 - or it is not known
- strategy for avoiding unwanted clashes:
 - use http URIs of webspace you control
- AND provide some documentation about the URI
- Need to distinguish the URI of a resource from URI of the associated documents describing it:
 - Example: URI for “Lord of the Ring”

`http://www.wikipedia.org/wiki/LordOfTheRing#URI`

`http://www.wikipedia.org/wiki/LordOfTheRing`

Literals

- Literals represent data values
 - denoted as string
 - interpreted via assigned datatype
 - literals without explicitly associated datatype are treated like strings

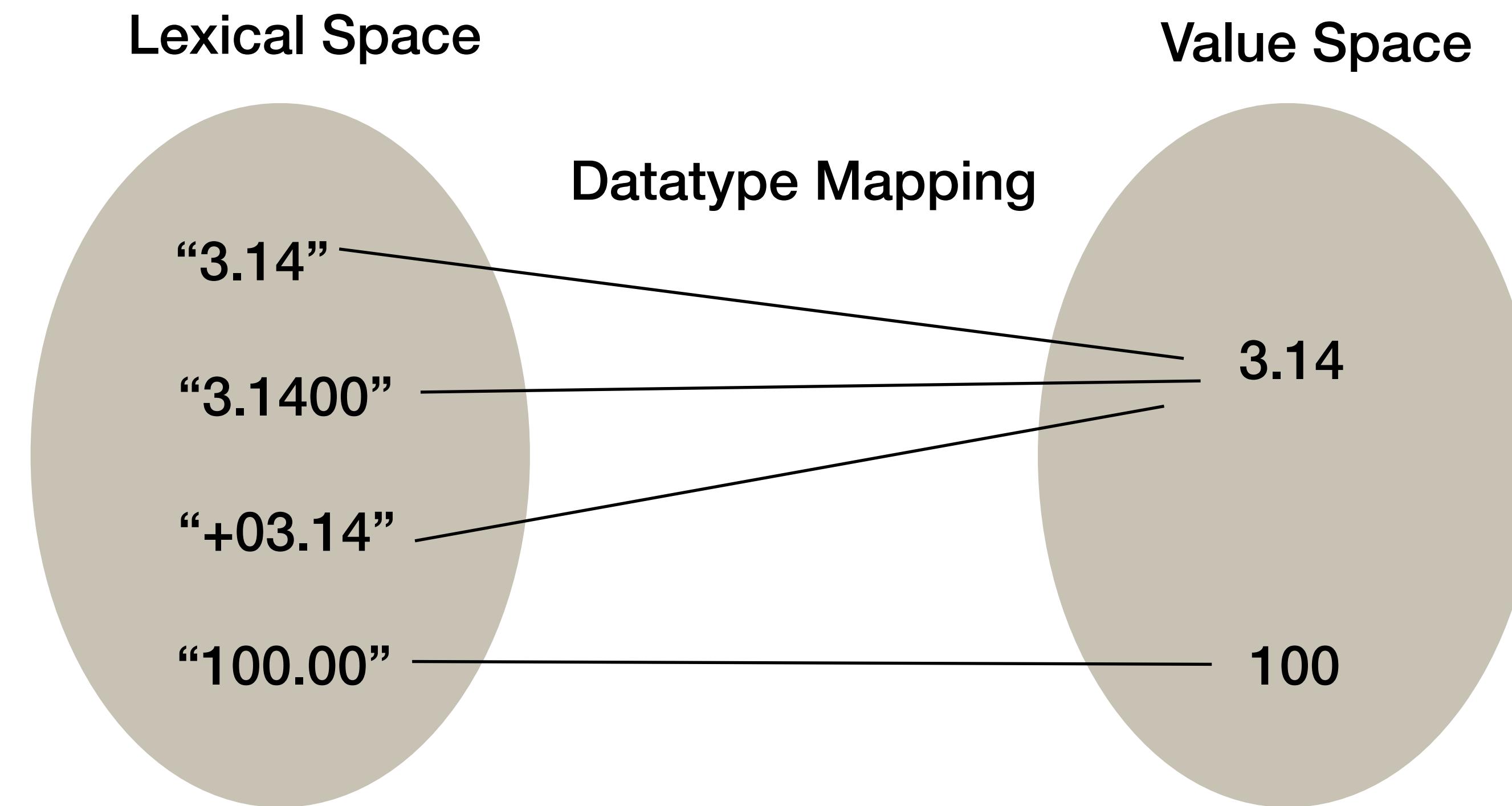


Datatypes in RDF

- Without datatypes literals are untyped, interpreted as strings
 - e.g. "02", "2", "2 . 0" all different
- typing literals with datatypes allows for more adequate treatment of values
 - semantic is clearer
- datatypes denoted by URIs and can be freely chosen
 - frequently: xsd datatypes from XML
 - syntax of typed literal: "`datavalue`"^{^^}`datatype-URI`
- `rdf:XMLLiteral` is the only datatype that is part of the RDF standard
 - denotes arbitrary balanced XML “snippets”

Datatypes in RDF

- Example: `xsd:decimal`



`"3.14"="+03.14"` holds for `xsd:decimal` but not for `xsd:string`

RDF Vocabulary

- RDF defines a number of resources and properties
 - We have already seen: rdf:XMLLiteral, rdf:type, ...
 - RDF vocabulary is defined in the namespace:
 - <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
- Classes:
 - `rdf:Property` `rdf:Statement` `rdf:XMLLiteral` `rdf:Seq` `rdf:Bag`
 - `rdf:Alt` `rdf>List`
- Properties:
 - `rdf:type` `rdf:subject` `rdf:predicate` `rdf:object` `rdf:first` `rdf:rest`
 - `rdf:n` `rdf:value`
- Resources:
 - `rdf:nil`

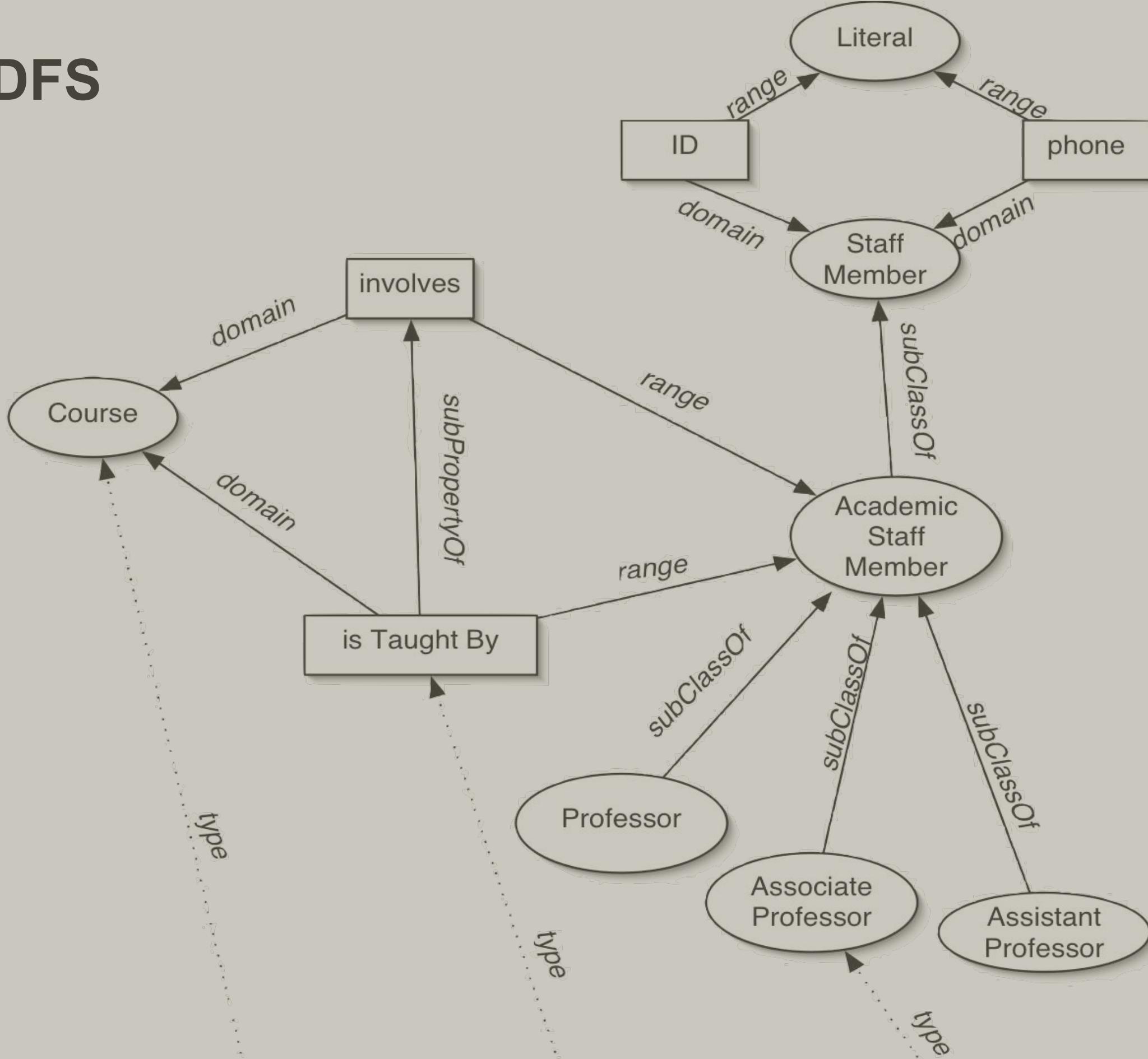
RDF Vocabulary Description Language

- Types in RDF:
 - <#john,rdf:type,#Student>
- Definition of what is “#Student” ⇒ A language for defining types in RDF:
 - Define classes:
 - “#Student is a class”
 - Relationships between classes:
 - “#Student is a sub-class of #Person”
 - Properties of classes:
 - “#Person has a property hasName”
- RDF Schema is such a language

RDF vs RDFS

- RDF language for describing structured information
 - individuals:
 - the book entitled “Lord of the rings”, the author “J.R.R Tolkien”...
 - relations between individuals:
 - The book “Lord of the rings” is authored by “J.R.R Tolkien”
 - types of literals and resources:
 - They belong to class of elements sharing the same characteristics
 - natural numbers, dates, ...
- How do we model classes of individuals?

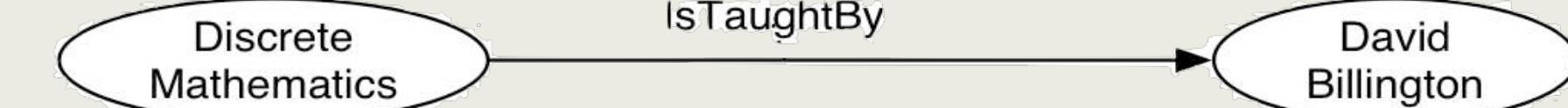
RDFS



RDFS

RDF

RDF



Recap

- RDF language
 - Turtle syntax
 - URIs
 - Datatypes
- Schema modelling in RDF: RDF Schema

COMP318

RDFS Semantics

www.csc.liv.ac.uk/~valli/Comp318



Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

v.Tamma@liverpool.ac.uk

Where were we

- RDF and RDFS
- Vocabulary and principles

RDF Vocabulary Description Language

- Types in RDF:
 - <#john,rdf:type,#Student>
- Definition of what is “#Student” ⇒ A language for defining types in RDF:
 - Define classes:
 - “#Student is a class”
 - Relationships between classes:
 - “#Student is a sub-class of #Person”
 - Properties of classes:
 - “#Person has a property hasName”
- RDF Schema is such a language

RDF vs RDFS

- RDF language for describing structured information
 - **individuals:**
 - the book entitled “Lord of the rings”, the author “J.R.R Tolkien”...
 - **relations between individuals:**
 - The book “Lord of the rings” is authored by “J.R.R Tolkien”
 - **types of literals and resources:**
 - They belong to class of elements sharing the same characteristics
 - natural numbers, dates, ...
- How do we model classes of individuals?

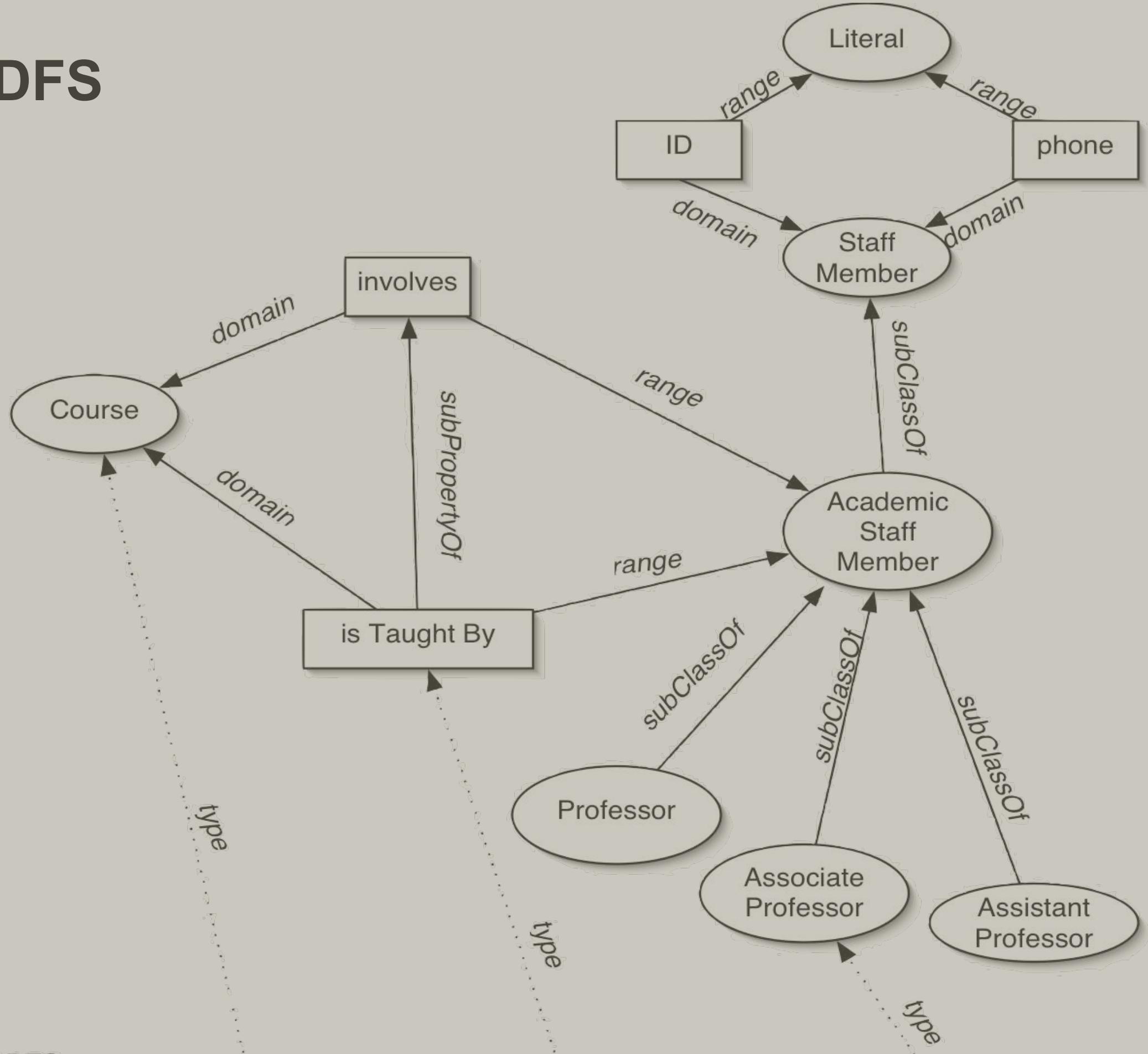
Basic Ideas of RDF Schema

- RDF is a universal language that lets users describe resources in their own vocabularies
 - RDF does not assume, nor does it define semantics of any particular application domain
- The user can do so in RDF Schema using:
 - Classes and Properties
 - Class Hierarchies and Inheritance
 - Property Hierarchies

Classes and their Instances

- We must distinguish between
 - Concrete “things” (**individual** objects) in the domain: *Semantic Web, John Smith* etc.
 - Sets of individuals sharing properties called **classes**: *lecturers, students, courses* etc.
- Individual objects that belong to a class are referred to as instances of that class
- The relationship between instances and classes in RDF is through `rdf:type`

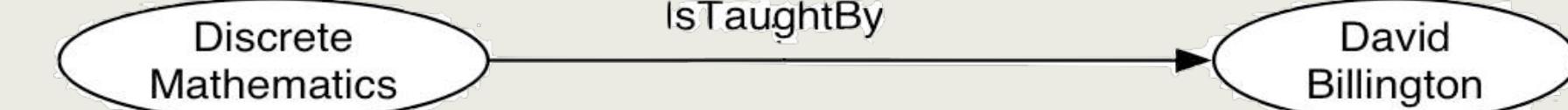
RDFS



RDFS

RDF

RDF



RDFS Primitives

- **Resource:**

- All resources are implicitly instances of `rdfs:Resource`.

- **Class:** describe sets of resources

- classes are resources themselves
 - e.g. Webpages, people, document types
- Class hierarchy can be defined through `rdfs:subClassOf`
- Every class is a member of `rdfs:Class`

- **Property:** subset of RDFS Resources that are properties

- **domain:** class associated with property, `rdfs:domain`
- **range:** type of the property values, `rdfs:range`
- Property hierarchy defined through `rdfs:subPropertyOf`

- **Statements**

- Resources that reify subject/predicate/object triples

RDFS Vocabulary

- RDFS is the RDF vocabulary description language
 - it can be used to build simple RDF vocabularies
 - it provides a data model for describing groups of related resources, and their relationships.
 - RDFS inherits RDF syntax, and thus RDFS specifications are RDF data.
 - RDFS has a simple model theoretic semantics that allows inference in the form of entailment rules.
 - RDFS vocabulary is defined in the namespace:
 - <http://www.w3.org/2000/01/rdf-schema#>

RDFS Vocabulary Description Language

- **Classes:**

- `<#Student, rdf:type, #rdfs:Class>`

- **Class hierarchies:**

- `<#Student, rdfs:subClassOf, #Person>`

- **Properties:**

- `<#hasName, rdf:type, rdf:Property>`

- **Property hierarchies:**

- `<#hasMother, rdfs:subPropertyOf, #hasParent>`

- **Associating properties with classes (a):**

- “The property #hasName only applies to # Person:”
- `<#hasName, rdfs:domain, #Person>`

- **Associating properties with classes (b):**

- “The type of the property #hasName is # xsd:string:”
- `<#hasName, rdfs:range, xsd:string>`

RDFS

- RDFS Classes
 - rdfs:Resource
 - rdfs:Class
 - rdfs:Literal
 - rdfs:Datatype
 - rdfs:Container
 - rdfs:Container Membership Property

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">

<owl:Ontology
  rdf:about="http://www.w3.org/2000/01/rdf-schema#"
  dc:title="The RDF Schema vocabulary (RDFS)"/>

<rdfs:Class rdf:about="http://www.w3.org/2000/01/rdf-schema#Resource">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#" />
  <rdfs:label>Resource</rdfs:label>
  <rdfs:comment>The class resource, everything.</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="http://www.w3.org/2000/01/rdf-schema#Class">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#" />
  <rdfs:label>Class</rdfs:label>
  <rdfs:comment>The class of classes.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdfs:Class>
...
```

RDFS

● RDFS Properties

- rdfs:domain
- rdfs:range
- rdfs:subPropertyOf
- rdfs:subClassOf
- rdfs:member
- rdfs:seeAlso
- rdfs:isDefinedBy
- rdfs:comment
- rdfs:label

```
<rdf:Property rdf:about="http://www.w3.org/2000/01/rdf-schema#subClassOf">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#" />
  <rdfs:label>subClassOf</rdfs:label>
  <rdfs:comment>The subject is a subclass of a class.</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:domain rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.w3.org/2000/01/rdf-schema#subPropertyOf">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#" />
  <rdfs:label>subPropertyOf</rdfs:label>
  <rdfs:comment>The subject is a subproperty of a property.</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:domain rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.w3.org/2000/01/rdf-schema#comment">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#" />
  <rdfs:label>comment</rdfs:label>
  <rdfs:comment>A description of the subject resource.</rdfs:comment>
  <rdfs:domain rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
...
```

Why Classes are Useful

- Impose restrictions on what can be stated in an RDF document using the schema
 - As in programming languages
 - E.g. A+1, where A is an array
 - All Lecturers must hold a PhD, so `has_title = "PhD"` for the members of the class `Lecturer`
 - Disallow nonsense from being stated

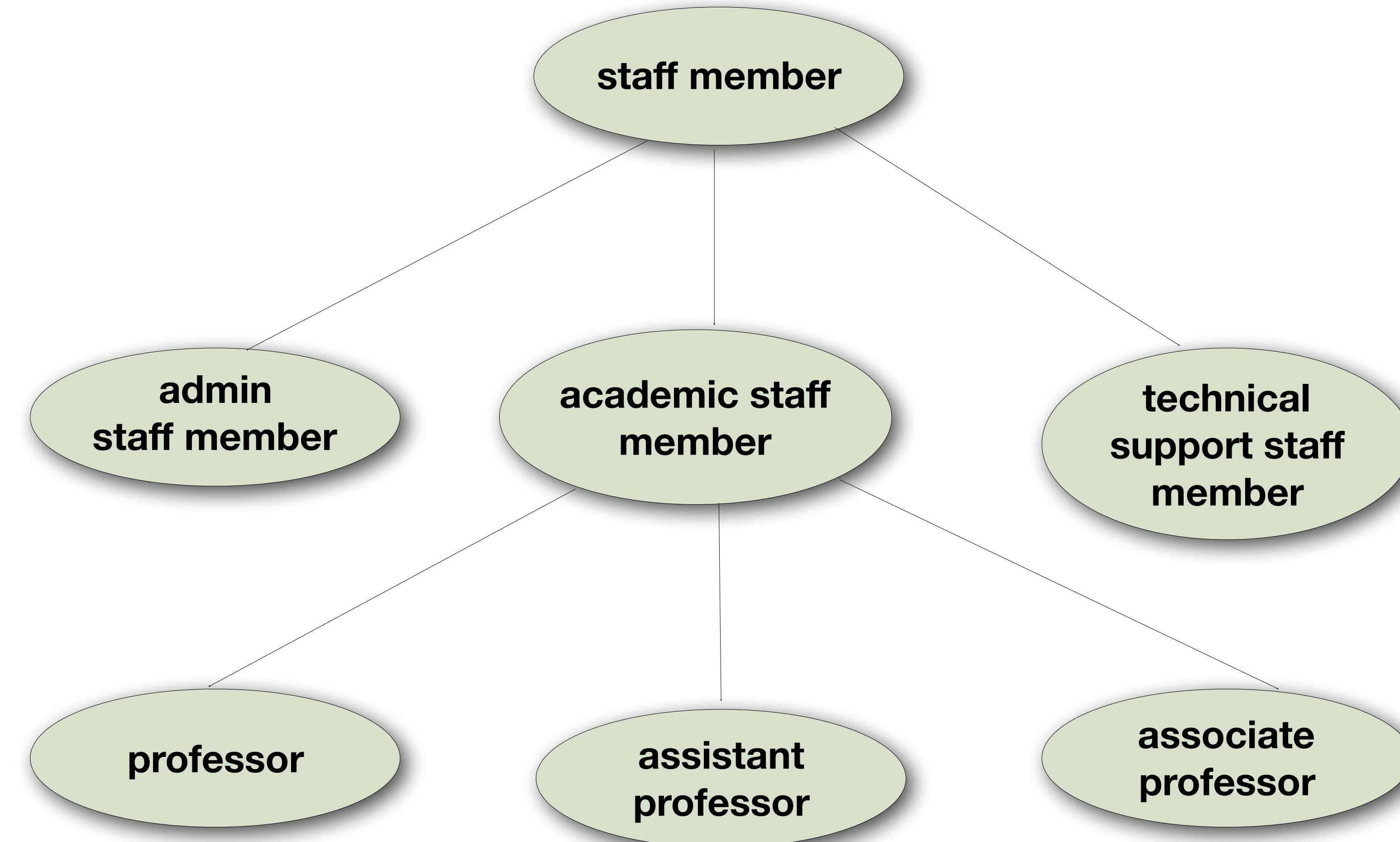
Disallow nonsensical statements

- Semantic Web is taught by Advanced Web Technologies
 - We want courses to be taught by lecturers only
 - Restriction on values of the property “is taught by” (range restriction)
- Room MZH5760 is taught by John Smith
 - Only courses can be taught
 - This imposes a restriction on the objects to which the property can be applied (domain restriction)

Class Hierarchies

- Classes can be organised in hierarchies
 - A is a subclass of B if every instance of A is also an instance of B
 - Then B is a superclass of A
- A subclass graph need not be a tree
- A class may have multiple superclasses

Class Hierarchy Example



Inheritance in Class Hierarchies

- Range restriction: Courses must be taught by academic staff members only
 - John Smith is a professor
 - He inherits the ability to teach from the class of academic staff members
- This is done in RDF Schema by fixing the semantics of “**rdfs:subClassOf**”
 - It is not up to an application (RDF processing software) to interpret “**rdfs:subClassOf**”

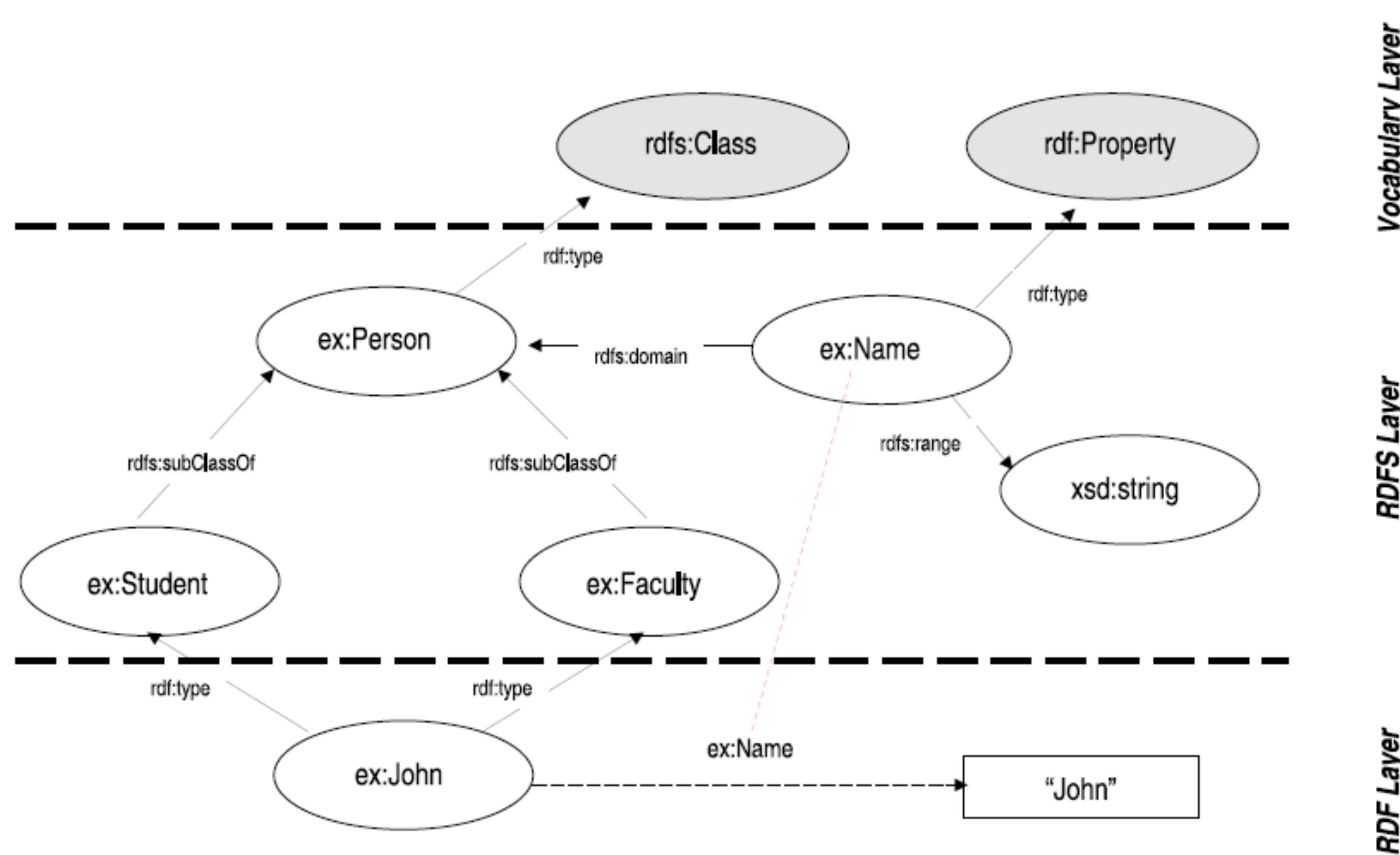
Property Hierarchies

- Hierarchical relationships for properties
 - E.g., “**is taught by**” is a sub-property of “**involves**”
 - If a course C is taught by an academic staff member A, then C also involves A
- The converse is not necessarily true
 - E.g., A may be the teacher of the course C, or
 - a tutor who marks student homework but does not teach C
 - P is a subproperty of Q, if $Q(x,y)$ is true whenever $P(x,y)$ is true

RDF Layer vs RDF Schema Layer

- “*Semantic Web is taught by John Smith*”
- The schema is itself written in RDF Schema, that can express its components:
 - subClassOf, Class, Property, subPropertyOf, Resource, etc.

RDFS Example



Vocabulary Layer

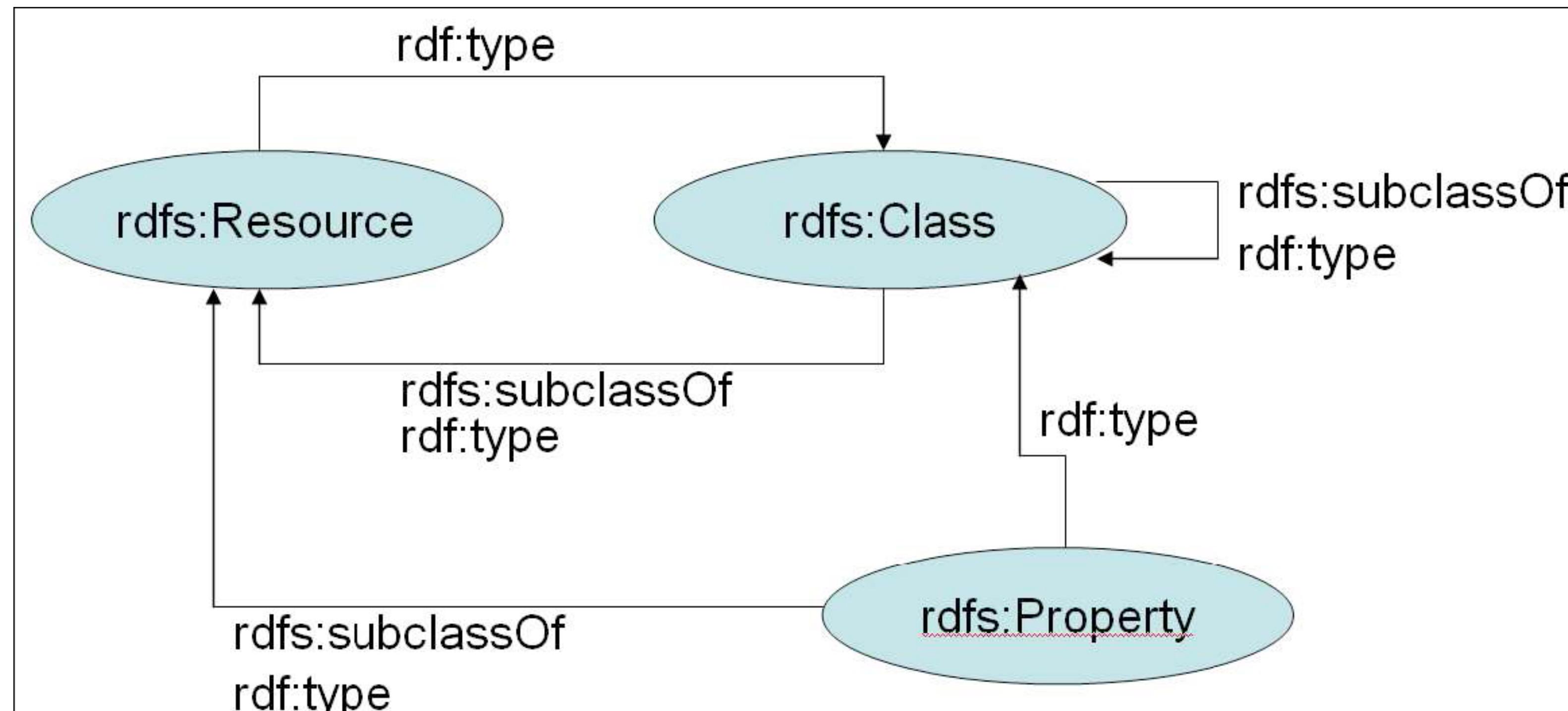
RDFS Layer

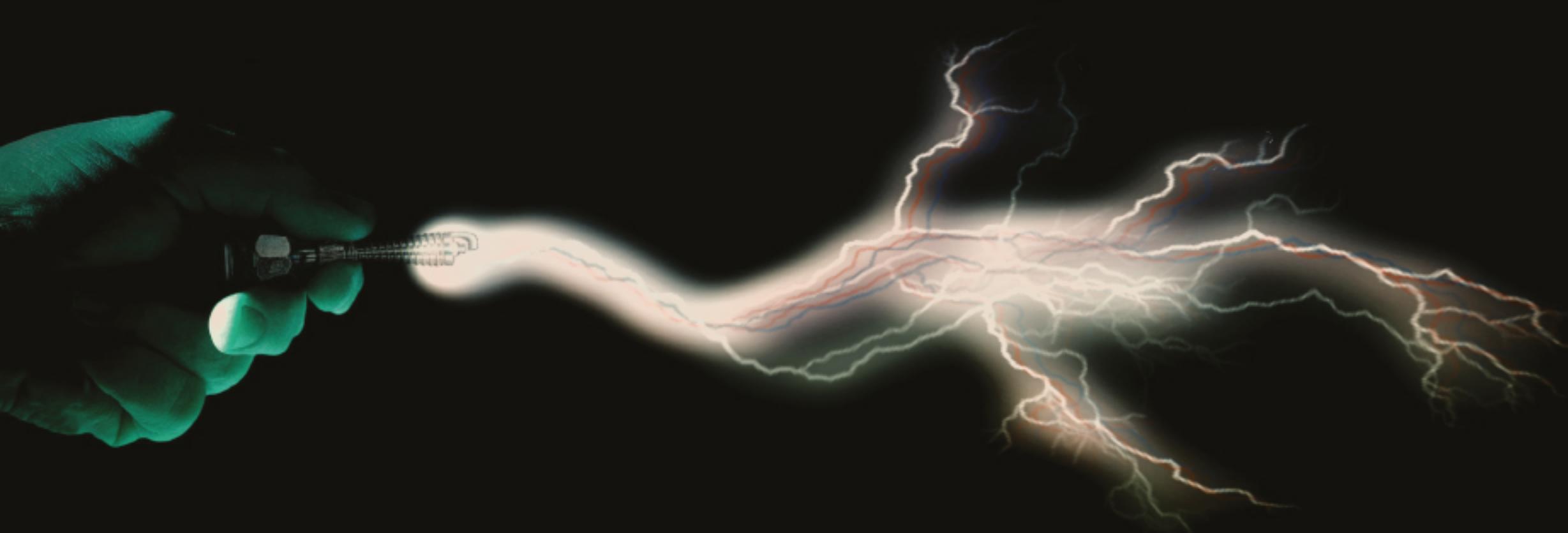
RDF Layer

Literals in RDFS

- Each literal is an rdfs:Literal
 - if we have:
 - $\langle \#john, \#hasName, "John" \rangle$
 - Which is interpreted as:
 - $\langle \#john, \#hasName, _:X \rangle$
 - $\langle _:X, \text{rdf:type}, \text{rdfs:Literal} \rangle$

RDFS Vocabulary





IGNITE YOUR FUTURE

Preparing for your future, today

8-13 February 2020
Department of Computer Science
University of Liverpool

Our Ignite Your Future series of events this year is even bigger and better!

Throughout the week events run by students for students brings you employers you need, and want, to know.

There will be tutorials where you can increase your knowledge and networking which will enable you to meet new people, build your confidence and get to know about careers in Computer Science.

Come to one or all of the events throughout the week, find out about opportunities and get up to speed with what is happening in the tech sector around the North West and beyond.

Build your network of contacts as you go, bringing you closer to your next internship, placement or job.

Find out more and book via CareerHub.



Careers and
Employability

@livunicareers #IYF2020



Computer Science Hackathon
Saturday 8 February, 10am - 7pm

How to network
Monday 10 February, 1pm - 3pm

Exploring alternative careers
Monday 10 February, 5pm - 7pm

Now you SME me!
Tuesday 11 February, 2pm - 3pm

Careers in cyber security
Tuesday 11 February, 5pm - 8pm

Computer Science skills session
Wednesday 12 February, from 1.30pm

ISE Careers Fair - Amsterdam
Thursday 13 February, all day event

Making a successful application
Thursday 13 February, 1pm - 2pm

Alumni & friends networking event
Thursday 13 February, 5pm - 8pm

Recap

- RDF as a data modelling language
 - RDF syntax
 - RDFS schema language

COMP318

RDFS Semantics

www.csc.liv.ac.uk/~valli/Comp318



Dr Valentina Tamma
Room: Ashton 2.12
Dept of computer science
University of Liverpool
Tamma@liverpool.ac.uk

Where were we

- RDF and RDFS
- Vocabulary and model

Syntax vs semantics

- **Syntax:** set of symbols without meaning
- **Semantics:** the meaning associated to the symbols



Semantics

- RDF(S) vocabulary has built-in “meaning”
- RDF(S) Semantics
 - Makes meaning explicit
 - Defines what follows from an RDF graph
- Semantic notions
 - Subgraph
 - Entailment

Example of logical consequence

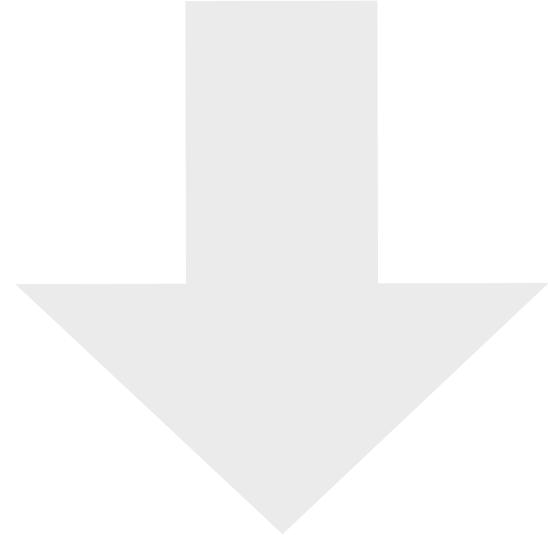
```
mus: Shoegazing rdfs:subClassOf mus: Alternative_rock  
mus: Alternative_rock rdfs: subClassOf mus: Music_genre
```

- What can we derive???

Example of logical consequence

```
mus: Shoegazing rdfs:subClassOf mus: Alternative_rock  
mus: Alternative_rock rdfs: subClassOf mus: Music_genre
```

- What can we derive???



```
mus: Shoegazing rdfs: subClassOf mus: Music_genre
```

Example of logical consequence

- In deriving the triple

```
mus: Shoegazing rdfs: subClassOf mus: Music_genre
```

we used the transitive property holding for `subClassOf`

- we used properties of the RDFS vocabulary

RDF(S) entailment

- An RDF(S) graph entails implicit triples
 - triples not explicitly contained in the graph, but that can be derived from an RDF(S) graph
 - using the special semantics of the vocabulary of the graph
 - vocabulary of the graph: set of names which occurs as the subject, predicate, object
 - Interpretations assign special meaning to the symbols in a particular vocabulary

Semantics

- RDF(S) vocabulary has built-in “meaning”
- RDF(S) Semantics
 - Makes meaning explicit
 - Defines what follows from an RDF graph
- Semantic notions
 - Subgraph
 - Entailment

Subgraph

- G₂ is a subgraph of G₁ if and only if the triples in G₂ are a subset of the triples in G₁

ex1:johnURI	ex1:hasName	ex1:johnfullname
	ex1:johnfullname	ex1:firstName "John"
ex1:johnfullname	ex1:surname	"John"

- Each of the following set of triples is a subgraph:

ex1:johnURI	ex1:hasName	ex1:johnfullname
--------------------	--------------------	-------------------------

ex1:johnURI	ex1:hasName	ex1:johnfullname
	ex1:johnfullname	ex1:firstName "John"

ex1:johnfullname	ex1:firstName "John"
ex1:johnfullname	ex1:surname "John"

RDF(S) entailment

- An RDF(S) graph entails implicit triples
 - triples not explicitly contained in the graph, but that can be derived from an RDF(S) graph
 - using the special semantics of the vocabulary of the graph
 - vocabulary of the graph: set of names which occurs as the subject, predicate, object
 - Interpretations assign special meaning to the symbols in a particular vocabulary

Entailment regimes

- Three entailment regimes
 - **simple entailment**: no particular extra conditions are posed on a vocabulary, including the RDF vocabulary itself;
 - it involves only graph transformations.
 - RDF entailment;
 - RDFS entailment: some extra conditions are posed by in the form of axiomatic triples and semantic conditions

Let's state the formalism

- a, b,
 - refer to any arbitrary URI
 - (i.e. anything that can appear in the predicate of a triple)
- u, v,
 - refer to any arbitrary URI or blank node ID
 - (i.e. anything that can appear in the subject of a triple)
- x, y,
 - refer to an arbitrary URI, blank node ID or literal
 - (i.e. anything that can appear in the object of a triple)
- _:n,
 - refer to the ID of a blank node
 - (i.e. appearing as a subject or object)
- |,
 - refers to a literal
 - (i.e. a string that is sometimes found in the object)

Deduction rules

- If the triple $\langle u, a, x \rangle$ is valid, then we can entail that the triple $\langle u, a, _:n \rangle$ is valid

$$\frac{u \ a \ x \ .}{u \ a \ _:n \ .}$$

se1

- If the triple $\langle u, a, x \rangle$ is valid, then we can entail that the triple $\langle _:n, a, x \rangle$ is valid

$$\frac{u \ a \ x \ .}{_:n \ a \ x \ .}$$

se2

Formalism

a, b, refer to any arbitrary URI

u, v, refer to any arbitrary URI or blank node ID

x, y, refer to any arbitrary URI, blank node ID or literal

$_:n$, refer to the ID of a blank node

l, refers to a literal

Simple entailment deduction rules

- URIs are all treated equally
 - we can decide whether a graph entails by applying the following rules se1 and se2 and adding the resulting triples to the original graph

$$\begin{array}{c} u \ a \ x \ . \\ \hline u \ a \ \underline{_ : n} \ . \end{array}$$

se1

*I.e. if you have an **object**, you can derive a **blank node for that object**, as long as the **subject and predicate** stay the same.*

$$\begin{array}{c} u \ a \ x \ . \\ \hline \underline{_ : n} \ a \ x \ . \end{array}$$

se2

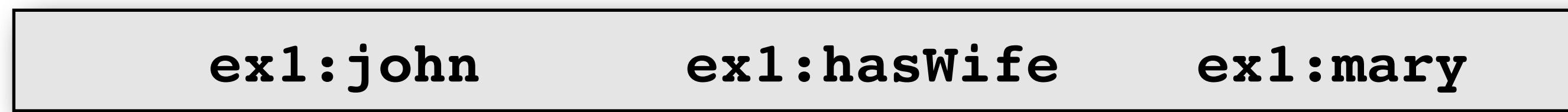
*I.e. if you have a **subject**, you can derive a **blank node for that subject**, as long as the **object and predicate** stay the same.*

Simple entailment deduction rules

- A graph G_1 simply entails a graph G_2 , if G_1 can be extended to a graph G'_1 by virtue of the rules se_1 and se_2 such that G_2 is contained in G'_1
 - $G_2 \subseteq G'_1$

Note

- se1 and se2 effectively “weaken” the subject and the object of the triples they are applied
 - by applying se2 to the triple



we derive that



- john hasWife mary is weakened into the statement someone hasWife mary (i.e. mary is a wife)
- se1 and se2 can be safely applied only if the blank node `_ :n` that is being introduced was not already present in the graph. If `_ :n` was already introduced, then the rules can be used only to assign `_ :n` with the same resource that which it was originally assigned (see example next)

Example

- Let's consider the graph G_1

```
book:uri ex:publishedBy crc:uri .  
book:uri ex:title "SW Technologies" .  
crc:uri ex:name "CRC Press" .
```

- Let's see if G_1 entails the graph G_2 below

```
book:uri ex:publishedBy _:blank1 .  
_:blank1 ex:name _:blank2 .  
_:blank1 ex:name "CRC Press" .
```

Example

se1

$$\frac{u \ a \ x \ .}{u \ a \ _{:n} \ .}$$

se2

$$\frac{u \ a \ x \ .}{_{:n} \ a \ x \ .}$$

- Let's consider the graph G1

```
book:uri ex:publishedBy crc:uri .
book:uri ex:title      "SW Technologies" .
crc:uri   ex:name       "CRC Press" .
```

- We need to find out whether (and if so, how) the deduction rules se1 and se2 can be applied to G1 to produce a graph G1' that contains G2

```
book:uri ex:publishedBy _:blank1 .
_:blank1      ex:name      _:blank2 .
_:blank1      ex:name      "CRC Press" .
```

Example

- By applying se1 to the first triple in G1 we add a triple with a blank node to the graph

```
book:uri ex:publishedBy _:blank1 .
```

- By applying se2 to crc:uri ex:name “CRC Press” we can add the triple

```
_:blank1 ex:name "CRC Press" .
```

- note that the empty node referenced by _:blank1 has been introduced by rule se1 exactly for crc:uri (and no other URI)

Example

- Finally, by applying se1 to the triple just generated

```
_blank1 ex:name "CRC Press" .
```

- we obtain the triple

```
_blank1 ex:name _blank2 .
```

- so now we have

Original Triples	book:uri ex:publishedBy crc:uri .	book:uri ex:title "SW Technologies" .	crc:uri ex:name "CRC Press" .
Derived Triples	book:uri ex:publishedBy _blank1 .	_blank1 ex:name _blank2 .	"CRC Press" .
	book:uri ex:publishedBy _blank1 .	_blank1 ex:name _blank2 .	"CRC Press" .

Example

- G_2

```
book:uri ex:publishedBy _:blank1 .  
_:blank1 ex:name _:blank2 .  
_:blank1 ex:name "CRC Press" .
```

- is contained in G'_1 and therefore G_1 entails G_2

```
book:uri ex:publishedBy crc:uri .  
book:uri ex:title "SW Technologies" .  
crc:uri ex:name "CRC Press" .  
  
book:uri ex:publishedBy _:blank1 .  
_:blank1 ex:name _:blank2 .  
_:blank1 ex:name "CRC Press" .
```

Exercise

Given the RDF graph G:

rdfs:range	rdfs:range	rdfs:Class .
:s	rdfs:domain	:b .
_ :m	rdfs:subPropertyOf	:s .
:v	:s	:x .
:v	rdf:type	_ :m .
_ :m	rdfs:subClassOf	_ :n .
:b	rdfs:subClassOf	:s .
:b	rdfs:comment	"blah" .

Is the following graph **simple**-entailed by G? Explain the answer

:b rdf:type rdfs:Literal .

Exercise

Given the RDF graph G:

rdfs:range	rdfs:range	rdfs:Class .
:s	rdfs:domain	:b .
_:m	rdfs:subPropertyOf	:s .
:v	:s	:x .
:v	rdf:type	_:m .
_:m	rdfs:subClassOf	_:n .
:b	rdfs:subClassOf	:s .
:b	rdfs:comment	"blah" .

Is the following graph **simple**-entailed by G? Explain the answer

:b rdf:type rdfs:Literal .

No, since there is no triple in the graph that has :b as subject and rdf:type as predicate

Exercise

Given the RDF graph G:

rdfs:range	rdfs:range	rdfs:Class .
:s	rdfs:domain	:t .
:u	rdfs:subPropertyOf	:s .
:a	:c	:b .
:a	rdf:type	:u .
:u	rdfs:subClassOf	:y .
:t	rdfs:subClassOf	:s .
:t	rdfs:comment	"blah" .

Is the following graph **simple**-entailed by G? Explain the answer

rdfs:range rdfs:range rdfs:Class .

Exercise

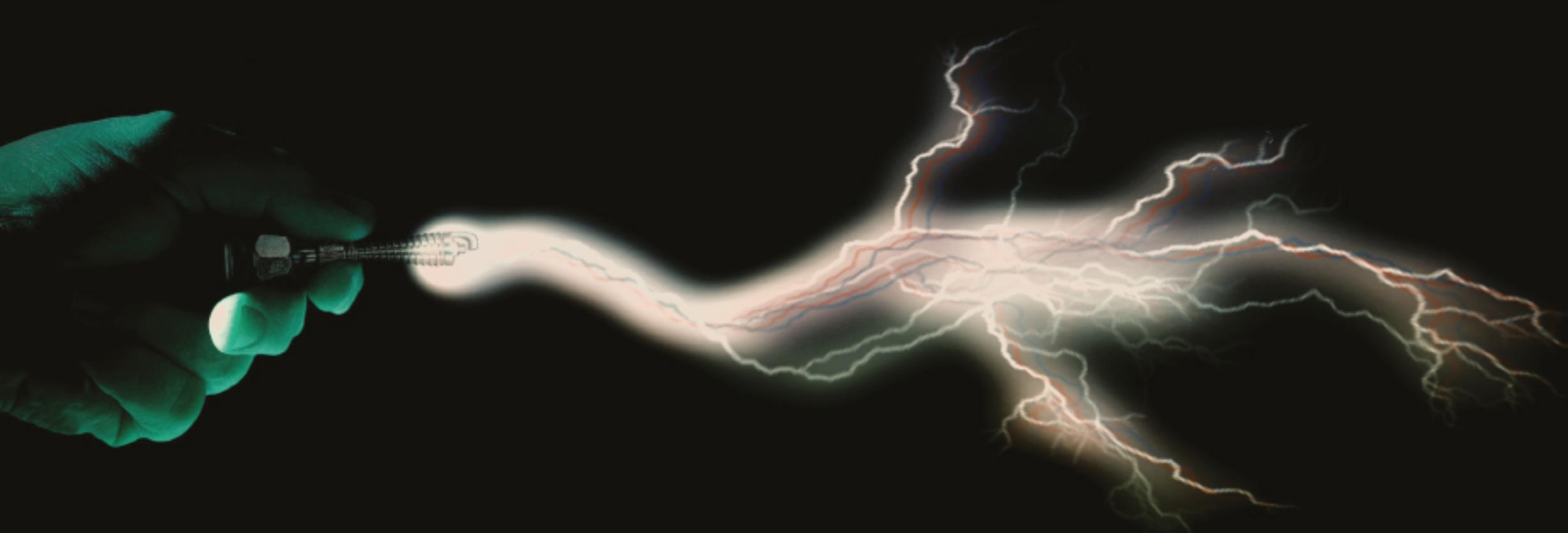
Given the RDF graph G:



	rdfs:range	rdfs:range	rdfs:Class .
:s		rdfs:domain	:t .
:u		rdfs:subPropertyOf	:s .
:a		:c	:b .
:a		rdf:type	:u .
:u		rdfs:subClassOf	:y .
:t		rdfs:subClassOf	:s .
:t		rdfs:comment	"blah" .

Is the following graph **simple**-entailed by G? Explain the answer
rdfs:range **rdfs:range** **rdfs:Class** .

Yes, it is entailed as the triple is already in the graph S. Therefore, whatever other triple we might add through the rules se1 and se2 the triple would still be contained in S



IGNITE YOUR FUTURE

Preparing for your future, today

8-13 February 2020
Department of Computer Science
University of Liverpool

Our Ignite Your Future series of events this year is even bigger and better!

Throughout the week events run by students for students brings you employers you need, and want, to know.

There will be tutorials where you can increase your knowledge and networking which will enable you to meet new people, build your confidence and get to know about careers in Computer Science.

Come to one or all of the events throughout the week, find out about opportunities and get up to speed with what is happening in the tech sector around the North West and beyond.

Build your network of contacts as you go, bringing you closer to your next internship, placement or job.

Find out more and book via CareerHub.



Careers and
Employability

@livunicareers #IYF2020



Computer Science Hackathon
Saturday 8 February, 10am - 7pm

How to network
Monday 10 February, 1pm - 3pm

Exploring alternative careers
Monday 10 February, 5pm - 7pm

Now you SME me!
Tuesday 11 February, 2pm - 3pm

Careers in cyber security
Tuesday 11 February, 5pm - 8pm

Computer Science skills session
Wednesday 12 February, from 1.30pm

ISE Careers Fair - Amsterdam
Thursday 13 February, all day event

Making a successful application
Thursday 13 February, 1pm - 2pm

Alumni & friends networking event
Thursday 13 February, 5pm - 8pm

Recap

- Simple entailment

COMP318

SPARQL

www.csc.liv.ac.uk/~valli/Comp318



Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

v.Tamma@liverpool.ac.uk

Where were we

- RDF and RDFS
- Vocabulary and model
- Entailment in RDF(S)
 - Simple entailment

SPARQL in general

- SPARQL Protocol and RDF Query Language
- SPARQL **Query Language** for RDF
 - Declarative
 - Based on the RDF data model (triples/graph)
- SPARQL **Query Results XML Format**
 - Representation of the results of SPARQL queries
- SPARQL **Protocol** for RDF
 - Transmission of SPARQL queries and the results
 - SPARQL endpoint: Web service that implements the protocol

SPARQL

- SPARQL is the query language for querying RDF. It allows users to:
 - Pull values from ***structured*** and ***semi-structured*** data
 - Explore data by querying ***unknown relationships***
 - Perform ***complex joins*** of ***disparate databases*** in a single, simple query
 - ***Transform RDF data*** from one vocabulary to another

Assumptions

- Data is represented in RDF

subject - predicate - object

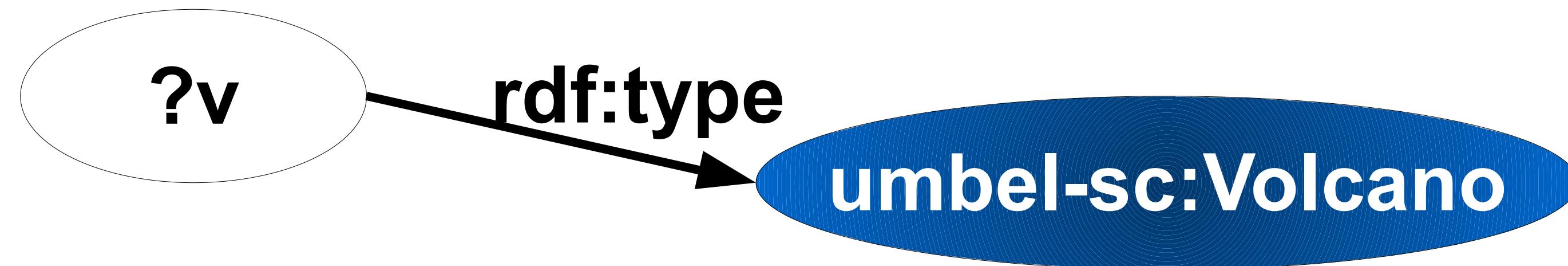
- Resources are represented by URIs
 - possibly abbreviated as ***prefixed names***
- Objects can be ***literals:*** strings, integers, booleans...
- We use Turtle syntax:
 - URIs:
 - `<http://www.sw-example.com/resource>` or `prefix:name`
 - literals:
 - `"plain string"` `"14.1"^^xsd:float` or `"string with language"@en`
 - triples:
 - `pref:subject anotherPref:predicate "object"`

SPARQL versions

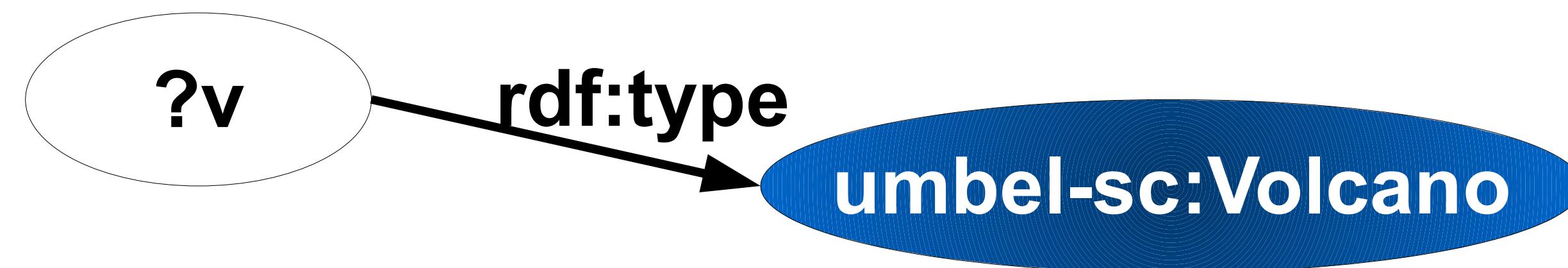
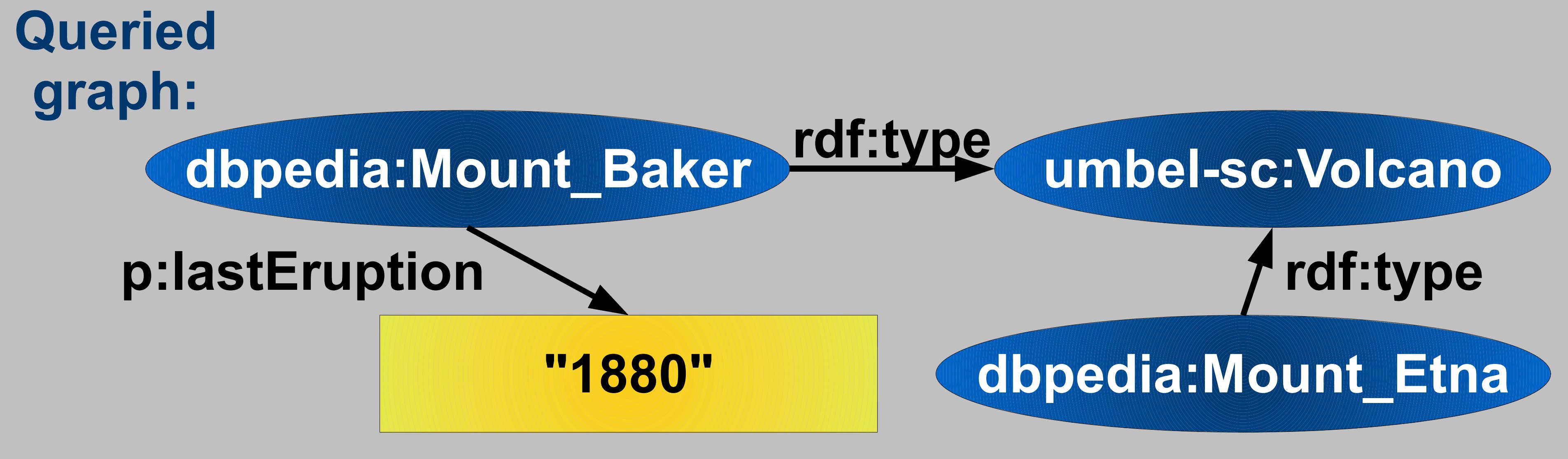
- SPARQL 1.0 (2008) included:
 - SPARQL 1.0 Query Language
 - SPARQL 1.0 Protocol
 - SPARQL Results XML Format
- SPARQL 1.1 (2013) includes:
 - SPARQL 1.1 versions of SPARQL Query and SPARQL Protocol
 - SPARQL 1.1 Update
 - SPARQL 1.1 Graph Store HTTP Protocol
 - SPARQL 1.1 Service Description
 - SPARQL 1.1 Entailments
 - SPARQL 1.1 Basic Federated Query

Main principle of SPARQL queries

- SPARQL based on the principle of **pattern matching**
 - Describe subgraphs of the queried RDF graph
 - Subgraphs that match your description yield a result
 - i.e. **graph patterns** (i.e. RDF graphs with variables)



Main principle of SPARQL queries



Results:

?v
dbpedia:Mount_Baker
dbpedia:Mount_Etna

SPARQL queries

- **PREFIX**
 - Prefix mechanism for abbreviating URIs
 - **PREFIX foo: <http://example.com/resources/>**
- **Query Pattern**
 - Result clause
 - Identifies the variables to be returned in the query answer
 - **SELECT ...**
- **DATASET DEFINITION**
 - Name(s) of the graph(s) to be queried
 - **FROM ...**
- **QUERY PATTERN**
 - Specifying what to query for in the dataset
 - Query pattern as a list of triple patterns
 - **WHERE {**
 - ...**}**
- **QUERY MODIFIERS**
 - **ORDER BY ...**

Query execution

- SPARQL queries are executed against RDF datasets (made by RDF graphs).
- A SPARQL ***endpoint*** accepts queries and returns results via HTTP.
 - endpoint is the entry point to a service, a process, or a queue or topic destination
 - **Generic** endpoints will query any Web-accessible RDF data
 - **Specific** endpoints are hardwired to query against particular datasets

SPARQL Result set

- Result sets are illustrated in:

x	y	z
=====		
"Alice"	http://example/a	

- A 'binding' is a pair (variable, RDF term). In this result set, there are three variables: x, y and z (shown as column headers).
- Each solution is shown as one row in the body of the table.
- Here, there is a single solution
 - x is bound to "Alice",
 - y is bound to <<http://example/a>>,
 - z is not bound to an RDF term.

URI abbreviation: PREFIX

- Mechanism for namespace abbreviation
- Syntax:

PREFIX abbr: <URI>

- Example:

PREFIX rdf: <<http://www.w3.org/1999/02/22-rdf-syntax-ns#>>

- Default:

PREFIX : <URI>

- Example:

PREFIX : <<http://example.org/myOntology#>>

URI abbreviation: PREFIX

```
PREFIX rdf:  
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>  
PREFIX : <http://example.org/myOntology#>
```

- Prologue:
- Prefix definitions enable CURIEs in the query
 - CURIE: defines a generic, abbreviated syntax for expressing Uniform Resource Identifiers (URIs).
 - Abbreviated URI expressed in a compact syntax, and may be found in both XML and non-XML grammars.
 - A CURIE may be considered a datatype.
- Attention: No period (“.”) character to separate (as in N3)

Selecting variables: SELECT

- Filtering variables to return
- Variables: **?string**

?x ?title ?name

- variables can match any node (resource or literal) in the RDF document

- Syntax:

SELECT var1, ... ,varn

SELECT ?x,?title

SELECT *

- Variables in **SELECT** are **distinguished** variables
- **DISTINCT** for disjoint results

Query Patterns: FROM

- Specifies the dataset to be queried
- Can be the default dataset
 - Then **FROM** can be omitted
 - **FROM** and **NAMED FROM** each with a URI to specify one or more dataset

Query Patterns: WHERE

- Graph pattern to match
 - Triple patterns are just like triples, but any part of a triple pattern can be replaced with a variable

- Set of triples:

- `{ (subject predicate object .)* }`
 - Subject: URI, QName, Blank node, Variable
 - Predicate: URI, QName, Blank node, Variable
 - Object: URI, QName, Blank node Literal, Variable

- Example:

```
{  
  _:author ex:hasName ?name .  
  _:author ex:authorOf :lotr .  
}
```

- Optional triples: OPTIONAL triple .

```
OPTIONAL :john ont:hasAge ?age
```

GRAPH PATTERNS

- Different types of graph patterns for the query pattern (WHERE clause):
 - Basic graph pattern (BGP)
 - Group graph pattern
 - Optional graph pattern
 - Union graph pattern
 - Graph graph pattern (Constraints)

Example RDF Dataset (Turtle)

```
@prefix : <http://example.org/data#> .  
@prefix ont: <http://example.org/myOntology#> .  
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .  
  
:john  
    vcard:FN "John Smith" ;  
    vcard:N [  
        vcard:Given "John" ;  
        vcard:Family "Smith" ] ;  
    ont:hasAge 32 ;  
    ont:marriedTo :mary .  
  
:mary  
    vcard:FN "Mary Smith" ;  
    vcard:N [  
        vcard:Given "Mary" ;  
        vcard:Family "Smith" ] ;  
    ont:hasAge 29 .
```

BASIC GRAPH PATTERNS

- Set of triple patterns (i.e. RDF triples with variables)
- Variable names prefixed with “?” or “\$” (e.g. ?v, \$v)
- Turtle syntax (similar to N3)
 - Syntactic sugar as in N3 (e.g. property and object lists)
- Blank nodes in SPARQL queries
 - Permitted as subject and object of a triple pattern
 - Like non-selectable variables

SPARQL Queries: all full names

“Return the full names of all people in the graph”

PREFIX vCard:

```
<http://www.w3.org/2001/vcard-rdf/3.0#>
```

```
SELECT ?fullName  
WHERE { ?x vCard:FN ?fullName }
```

fullName

=====

"John Smith"
"Mary Smith"

result:

```
:john  
  vcard:FN "John Smith" ;  
  vcard:N [  
    vcard:Given "John" ;  
    vcard:Family "Smith" ] ;  
  ont:hasAge 32 ;  
  ont:marriedTo :mary .  
  
:mary  
  vcard:FN "Mary Smith" ;  
  vcard:N [  
    vcard:Given "Mary" ;  
    vcard:Family "Smith" ] ;  
  ont:hasAge 29 .
```

SPARQL Queries: properties

“Return the relation between John and Mary”

```
PREFIX : <http://example.org/myOntology#>  
SELECT ?p  
WHERE { :john ?p :mary }
```

result:

```
p  
=====
```

<http://example.org/myOntology#marriedTo>

SPARQL Queries: complex patterns

“Return the spouse of a person by the name of John Smith”

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX ont: <http://example.org/myOntology#>
SELECT ?y
WHERE { ?x vCard:FN "John Smith".
         ?x ont:marriedTo ?y}
```

result:

```
y
=====
<http://example.org/data#mary>
```

```
:john
  vcard:FN "John Smith" ;
  vcard:N [
    vcard:Given "John" ;
    vcard:Family "Smith" ] ;
  ont:hasAge 32 ;
  ont:marriedTo :mary .

:mary
  vcard:FN "Mary Smith" ;
  vcard:N [
    vcard:Given "Mary" ;
    vcard:Family "Smith" ] ;
  ont:hasAge 29 .
```

SPARQL Queries: blank nodes

“Return the name and the first name of all people in the KB”

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>  
SELECT ?name, ?firstName  
WHERE {?x vCard:N ?name .  
       ?name vCard:Given ?firstName}
```

result:

name	firstName
=====	
_:a	“John Smith” “John”
_:b	“Mary Smith” “Mary”

OPTIONAL GRAPH PATTERNS

- Used to treat missing data
- Keyword OPTIONAL allows for optional patterns
- May yield unbound variables

SPARQL Queries: optional pattern

“Return the full name of all the people in the graph and their spouse”

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX ont: <http://example.org/myOntology#>
SELECT ?y ?name
WHERE {?x vCard:FN ?name .
      OPTIONAL {?x ont:marriedTo ?y}}
```

```
:john
  vcard:FN "John Smith" ;
  vcard:N [
    vcard:Given "John" ;
    vcard:Family "Smith" ] ;
  ont:hasAge 32 ;
  ont:marriedTo :mary .

:mary
  vcard:FN "Mary Smith" ;
  vcard:N [
    vcard:Given "Mary" ;
    vcard:Family "Smith" ] ;
  ont:hasAge 29 .
```

SPARQL Queries: optional pattern

“Return the full name of all the people in the graph and their spouse”

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX ont: <http://example.org/myOntology#>
SELECT ?y ?name
WHERE { ?x vCard:FN ?name .
        OPTIONAL { ?x ont:marriedTo ?y } }
```

results:

?name	?y
=====	

“John Smith” <http://example.org/data#mary>
“Mary Smith”

COMP318: SPARQL

www.csc.liv.ac.uk/~valli/Comp318



Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

v.Tamma@liverpool.ac.uk

Based on material by O. Hartig and Cambridge Semantics

Where were we

- SPARQL
 - RDF query language
 - SELECT Queries
 - SELECT ... FROM ... WHERE
 - Basic and OPTIONAL pattern

GRAPH PATTERNS

- Different types of graph patterns for the query pattern (WHERE clause):
 - Basic graph pattern (BGP)
 - Group graph pattern
 - Optional graph pattern
 - Union graph pattern
 - Graph graph pattern (Constraints)

Example dataset (in Turtle)

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix umbel-sc: <http://umbel.org/umbel/sc/> .  
@prefix dbpedia: <http://www.dbpedia.org/> .  
  
dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;  
    rdfs:label "Etna" ;  
    p:location dbpedia:Italy .  
dbpedia:Mount_Baker rdf:type umbel-sc:Volcano ;  
    p:location dbpedia:United_States .  
dbpedia:Beerenberg rdf:type umbel-sc:Volcano ;  
    rdfs:label "Beerenberg"@en ;  
    rdfs:label "Бееренберг"@ru .  
    p:location dbpedia:Norway .
```

UNION Graph patterns

- Union graph patterns allow us to query for possible alternatives

“Which volcanoes are located in Italy or in Norway?”

```
SELECT ?v WHERE
{?v rdf:type umbel-sc:Volcano .
{?v p:location dbpedia:Italy}
UNION
{?v p:location dbpedia:Norway}
}
?v
```

=====

dbpedia:Mount_Etna

dbpedia:Beerenberg

```
@prefix rdf: <http://www.w3.org/
1999/02/22-rdf-syntax-ns#> .
@prefix umbel-sc: <http://umbel.org/umbel/sc/
> .
@prefix dbpedia: <http://www.dbpedia.org/> .

dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;
    rdfs:label "Etna" ;
    p:location dbpedia:Italy .

dbpedia:Mount_Baker rdf:type umbel-sc:Volcano ;
    p:location
    dbpedia:United_States .

dbpedia:Beerenberg rdf:type umbel-sc:Volcano ;
    p:location dbpedia:Norway .
```

GROUP Graph patterns

```
SELECT ?v WHERE { ?v rdf:type umbel-
sc:Volcano .
```

```
{?v p:location dbpedia:Italy}
```

```
UNION
```

```
{?v p:location dbpedia:Norway}
```

```
}
```

Semantically
equivalent to

```
SELECT ?v WHERE { { ?v rdf:type umbel-
sc:Volcano }
```

```
{ { ?v p:location dbpedia:Italy } }
```

```
UNION
```

```
{?v p:location dbpedia:Norway } }
```

```
}
```

Constraints: Filters in Query Patterns

- Conditions on literal values with operators and functions
- Different forms
 - Value comparison, e.g., `>`, `!=`, `>=`
 - Numeric functions, e.g., `+`, `*`
 - SPARQL test, e.g., `BOUND(?x)`, `isLITERAL(?y)`
 - Negation, e.g., `!BOUND(?x)`
- Syntax: `FILTER` expression

```
SELECT ?v WHERE {  
  ?v rdf:type umbel-sc:Volcano ;  
    p:lastEruption ?le .  
FILTER ( ?le > 1900 ) }
```

SPARQL built-in filter functions

SPARQL		SPARQL 1.1	
Logical	!, &&,	Conditionals	IF, COALESCE
Math	+, -, *, /	Constructors	URI, BNODE, STRDT, STRLANG
Comparison	>, <, !=, =, ...	Strings	STRLEN, SUBSTR, UCASE, LCASE, STRSTARTS, STRENDSD, CONTAIS, CONCAT, ...

SPARQL built-in filter functions

SPARQL		SPARQL 1.1	
SPARQL Tests	<code>isURI</code> , <code>isBlank</code> , <code>isLiteral</code> , <code>bound</code>	More math	<code>abs</code> , <code>round</code> , <code>ceil</code> , <code>floor</code> , <code>RAND</code>
SPARQL accessors	<code>str</code> , <code>lang</code> , <code>datatype</code>	Date/Time	<code>now</code> , <code>year</code> , <code>month</code> , <code>day</code> , <code>hours</code> , <code>minutes</code> , <code>seconds</code> , <code>timezone</code>
Others	<code>sameTerm</code> , <code>langMatches</code> , <code>regex</code>	Hashing	<code>MD5</code> , <code>SHA1</code> , <code>SHA224</code> , <code>SHA256</code> , <code>SHA384</code> , <code>SHA512</code>

Solution Modifiers

- Modify the result set, but not single results
- Syntax: ORDER BY, LIMIT, OFFSET

NEGATION

“Which volcanoes do not have a name (rdfs:label)?”

```
SELECT ?v WHERE {
  ?v rdf:type umbel-sc:Volcano .
  OPTIONAL { ?v rdfs:label ?name }
  FILTER( ! BOUND(?name) )
}
?v
```

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
ns#> .
@prefix umbel-sc: <http://umbel.org/umbel/sc/> .
@prefix dbpedia: <http://www.dbpedia.org/> .

dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;
  rdfs:label "Etna" ;
  p:location dbpedia:Italy .

dbpedia:Mount_Baker rdf:type umbel-sc:Volcano ;
  p:location dbpedia:United_States .

dbpedia:Beerenberg rdf:type umbel-sc:Volcano ;
  rdfs:label "Beerenberg"@en ;
  rdfs:label "Бееренберг"@ru .
  p:location dbpedia:Norway .
```

NEGATION

“What volcanoes are not called Beerenberg?”

```
SELECT ?v WHERE {  
?v rdfs:type umbel-sc:Volcano .  
rdfs:label ?name .  
FILTER (?name != "Beerenberg")  
}  
?v  
=====
```

dbpedia:Mount_Etna
dbpedia:Mount_Baker
dbpedia:Beerenberg

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix umbel-sc: <http://umbel.org/umbel/sc/> .  
@prefix dbpedia: <http://www.dbpedia.org/> .  
  
dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;  
rdfs:label "Etna" ;  
p:location dbpedia:Italy .  
dbpedia:Mount_Baker rdf:type umbel-sc:Volcano ;  
p:location dbpedia:United_States .  
dbpedia:Beerenberg rdf:type umbel-sc:Volcano ;  
rdfs:label "Beerenberg"@en ;  
rdfs:label "Бееренберг"@ru .  
p:location dbpedia:Norway .
```



NEGATION AS FAILURE

“What volcanoes are not called Beerenberg?”

```
SELECT ?v WHERE {
  ?v rdf:type umbel-sc:Volcano .
  OPTIONAL { ?v rdfs:label ?name .
    FILTER (STR(?name) = "Beerenberg") }
  FILTER ( ! BOUND(?name) )
}
?v
=====
```

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
ns#> .
@prefix umbel-sc: <http://umbel.org/umbel/sc/> .
@prefix dbpedia: <http://www.dbpedia.org/> .

dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;
  rdfs:label "Etna" ;
  p:location dbpedia:Italy .

dbpedia:Mount_Baker rdf:type umbel-sc:Volcano ;
  p:location dbpedia:United_States .

dbpedia:Beerenberg rdf:type umbel-sc:Volcano ;
  rdfs:label "Beerenberg"@en ;
  rdfs:label "Бееренберг"@ru ;
  p:location dbpedia:Norway .
```

dbpedia:Mount_Etna

dbpedia:Mount_Baker

NEGATION AS FAILURE

- The OPTIONAL pattern in the previous query does not generate bindings in the following two cases:
 - There is no `rdfs:label` property for `?v`
 - There is an `rdfs:label` property for `?v` but its string value is not Bareenberg
- These two cases are then selected for output by the FILTER condition that uses `!bound`.

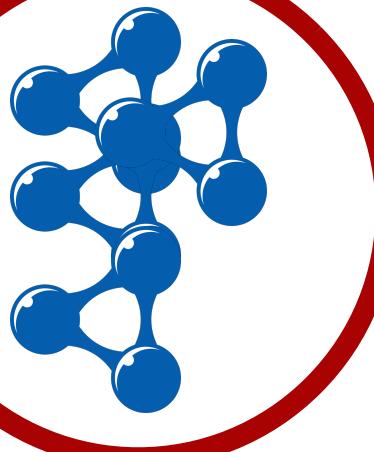
```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix umbel-sc: <http://umbel.org/umbel/sc/> .  
@prefix dbpedia: <http://www.dbpedia.org/> .  
  
dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;  
    rdfs:label "Etna" ;  
    p:location dbpedia:Italy .  
  
dbpedia:Mount_Baker rdf:type umbel-sc:Volcano ;  
    p:location dbpedia:United_States .  
  
dbpedia:Beerenberg rdf:type umbel-sc:Volcano ;  
    rdfs:label "Beerenberg"@en ;  
    rdfs:label "Бееренберг"@ru .  
    p:location dbpedia:Norway .
```

GRAPH Graph patterns

- SPARQL queries are executed against RDF datasets
- RDF datasets are composed of the default graph and zero or more **named graphs**
 - identified by a URI
- **Named graphs**
 - specified through the **FROM NAMED** clause
 - which allows us to scope the query being asked (e.g. to the graphs that comprise an application's user-data storage).
 - or hardwired in a particular endpoint
 - the **GRAPH** keyword allows portions of a query to match against the named graphs in the dataset
 - Anything outside the scope of **GRAPH** clause matches only against the default graph
 - Keyword **GRAPH** makes one of the named graphs the active graph used for pattern matching, if there is no named graph specified in the query it consider a merge of all the named graphs

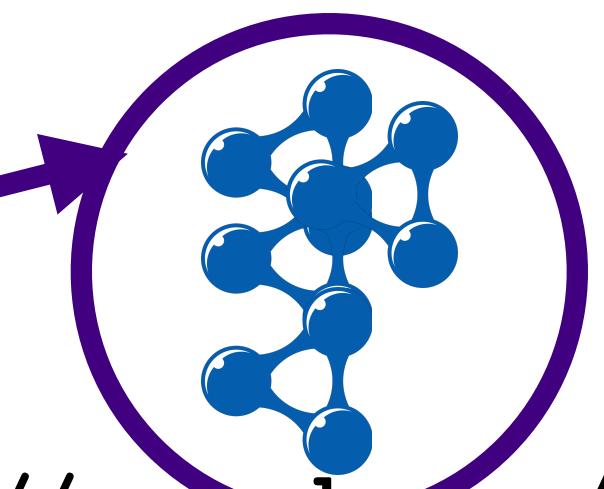
GRAPH Graph Pattern

```
dbpedia:Mount_Etna rdfs:seeAlso <http://example.org/d1>.  
dbpedia:Mount_Baker rdfs:seeAlso <http://example.org/d2>.
```



```
dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;  
rdfs:label "Etna" ;  
p:location dbpedia:Italy .
```

<http://example.org/d1>



```
dbpedia:Mount_Baker rdf:type umbel-sc:Volcano ;  
p:location dbpedia:United_States .
```

<http://example.org/d2>

```
dbpedia:Beerenberg rdf:type umbel-sc:Volcano ;  
rdfs:label "Beerenberg"@en ;  
rdfs:label "Бееренберг"@ru .  
p:location dbpedia:Norway .
```

<http://example.org/d3>

V.Tamma

GRAPH graph pattern

“Find all the volcanoes and the dataset they are described in”

```
SELECT ?g ?v
WHERE
GRAPH ?g {
    ?v rdf:type umbel-sc:Volcano . }
```

result:

?v	name
=====	
dbpedia:Mount_Etna	http://example.org/d1
dbpedia:Mount_Baker	http://example.org/d2
dbpedia:Beerenberg	http://example.org/d3

GRAPH graph pattern

“Find all the volcanoes and the dataset they are described in”

```
SELECT ?g ?v  
FROM NAMED http://example.org/d1  
FROM NAMED http://example.org/d2  
WHERE  
GRAPH ?g {  
?v rdf:type umbel-sc:Volcano . }
```

?v	name
=====	
dbpedia:Mount_Etna	http://example.org/d1
dbpedia:Mount_Baker	http://example.org/d2

SPARQL

- SPARQL is the query language for querying RDF. It allows users to:
 - Pull values from ***structured*** and ***semi-structured*** data
 - Explore data by querying ***unknown relationships***
 - Perform ***complex joins*** of ***disparate databases*** in a single, simple query
- ***Transform RDF data*** from one vocabulary to another

Result formats

- The results of SPARQL queries can be returned and/or rendered in a variety of formats:
 - **XML**. SPARQL specifies an XML vocabulary for returning tables of results.
 - **JSON**. A JSON "port" of the XML vocabulary, particularly useful for Web applications.
 - **RDF**. Certain SPARQL result clauses trigger RDF responses, which in turn can be serialized in a number of ways (RDF/XML, N-Triples, Turtle, etc.)
 - **HTML**. When using an interactive form to work with SPARQL queries.
 - Often implemented by applying an XSL transform to XML results.

Query Result Forms

- **SELECT**: Projection of query result
- **CONSTRUCT**: Returning RDF Graph
- **DESCRIBE**: Returning descriptions of RDF resource
 - not treated here
- **ASK**: “yes/no” query

Reconstructing an RDF Graph: CONSTRUCT

- CONSTRUCT { basic triple pattern* }
- Query result is an RDF graph
- Form of RDF Graph described using graph template
 - Construct graph for each pattern solution
 - Triples with unbound variables discarded
 - Illegal RDF triples discarded

CONSTRUCT Query Answers: example

- Graph

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .  
_:a foaf:name "Alice" .  
_:a foaf:mbox <mailto:alice@example.org> .
```

- Query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>  
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>  
CONSTRUCT { <http://example.org/person#Alice> vcard:FN ?name }  
WHERE { ?x foaf:name ?name }
```

- Result

```
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .  
<http://example.org/person#Alice> vcard:FN "Alice" .
```

Boolean Queries: Ask

- ASK { graph pattern }
- “Does the query have an answer?”
 - ASK replaces WHERE
 - Queries without variables are meaningful

ASK Query Answers: example

Graph

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .  
  
_:a foaf:name "Alice" .  
  
_:a foaf:homepage <http://work.example.org/alice/> .  
  
_:b foaf:name "Bob" .  
  
_:b foaf:mbox <mailto:bob@work.example> .
```

Query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>  
  
ASK { ?x foaf:name "Alice" }
```

Result

```
yes
```

Recap

- SPARQL syntax

COMP318: SPARQL

www.csc.liv.ac.uk/~valli/Comp318



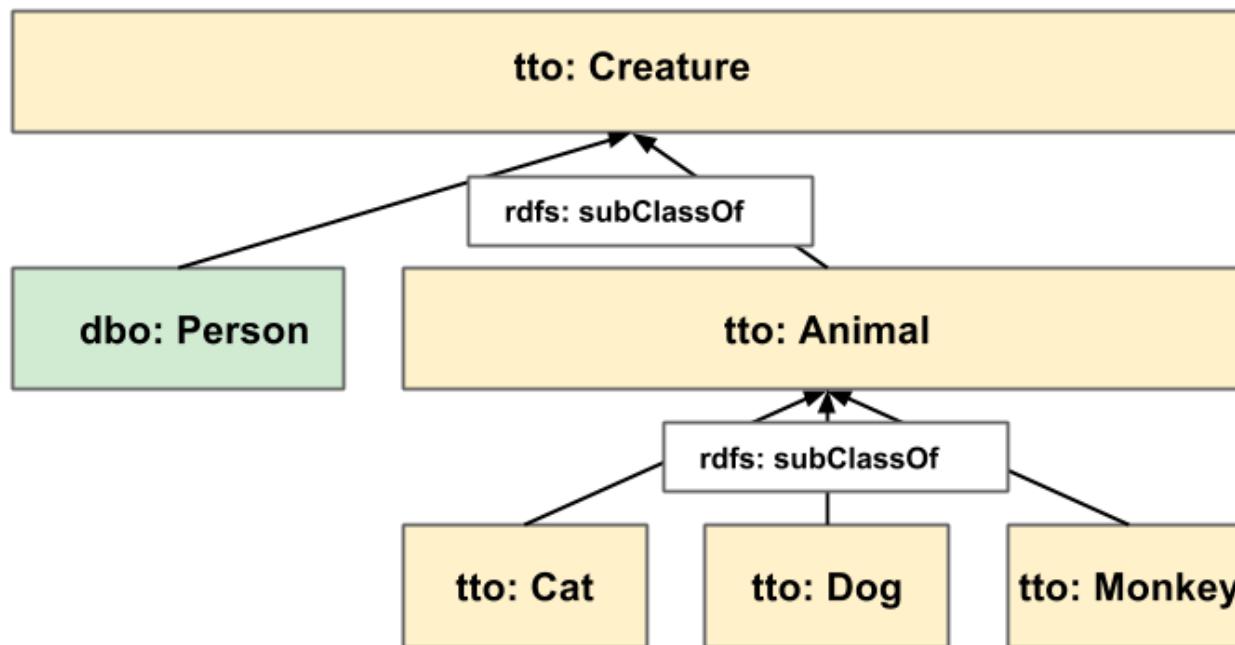
Dr Valentina Tamma
Room: Ashton 2.12
Dept of computer science
University of Liverpool
V.Tamma@liverpool.ac.uk

Where were we

- SPARQL
- Simple entailment in RDF

Data model for the exercises

```
@prefix rdf <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix tto: <http://example.org/tuto/ontology#> .
@prefix dbo: <http://dbpedia.org/ontology/> .
```



ONTOLOGY

```

▼ tto:Creature
  rdf:type rdfs:Class;
  rdfs:label "creature"^^xsd:string;
  rdfs:isDefinedBy tto: .

▼ dbo:Person
  rdfs:subClassOf tto:Creature .

▼ tto:Animal
  rdf:type rdfs:Class;
  rdfs:label "animal"^^xsd:string;
  rdfs:subClassOf tto:Creature ;
  rdfs:isDefinedBy tto: .

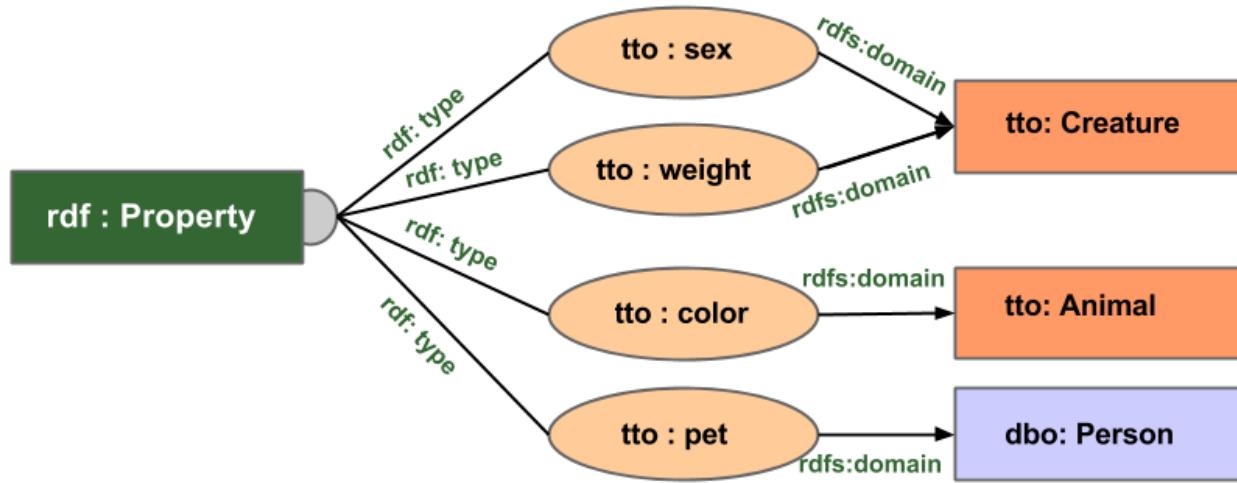
▼ tto:Cat
  rdf:type rdfs:Class;
  rdfs:label "cat"^^xsd:string;
  rdfs:subClassOf tto:Animal ;
  rdfs:isDefinedBy tto: .

▼ tto:Dog
  rdf:type rdfs:Class;
  rdfs:label "dog"^^xsd:string;
  rdfs:subClassOf tto:Animal ;
  rdfs:isDefinedBy tto: .

▼ tto:Monkey
  rdf:type rdfs:Class;
  rdfs:label "monkey"^^xsd:string;
  rdfs:subClassOf tto:Animal ;
  rdfs:isDefinedBy tto: .
  
```

The properties in the data model

```
@prefix rdf <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .  
@prefix tto: <http://example.org/tuto/ontology#> .  
@prefix dbo: <http://dbpedia.org/ontology/> .
```



```
▼ tto:sex
  rdf:type rdf:Property;
  rdfs:label "sex"^^xsd:string;
  rdfs:domain tto:Creature ;
  rdfs:range xsd:string ;
  rdfs:isDefinedBy tto: .

▼ tto:pet
  rdf:type rdf:Property;
  rdfs:label "domestic animal"^^xsd:string;
  rdfs:domain dbo:Person ;
  rdfs:range tto:Animal ;
  rdfs:isDefinedBy tto: .

▼ tto:color
  rdf:type rdf:Property;
  rdfs:label "hair of furr color"^^xsd:string;
  rdfs:domain dbo:Animal ;
  rdfs:range xsd:string ;
  rdfs:isDefinedBy tto: .

▼ tto:weight
  rdf:type rdf:Property;
  rdfs:label "weight"^^xsd:string;
  rdfs:comment "weight in kilograms"^^xsd:string;
  rdfs:domain tto:Creature ;
  rdfs:range xsd:decimal ;
  rdfs:isDefinedBy tto: .
```

dbo:<<http://dbpedia.org/ontology/>>

tto:<<http://example.org/tuto/ontology/>>

ttr:<<http://example.org/tuto/resource#>>

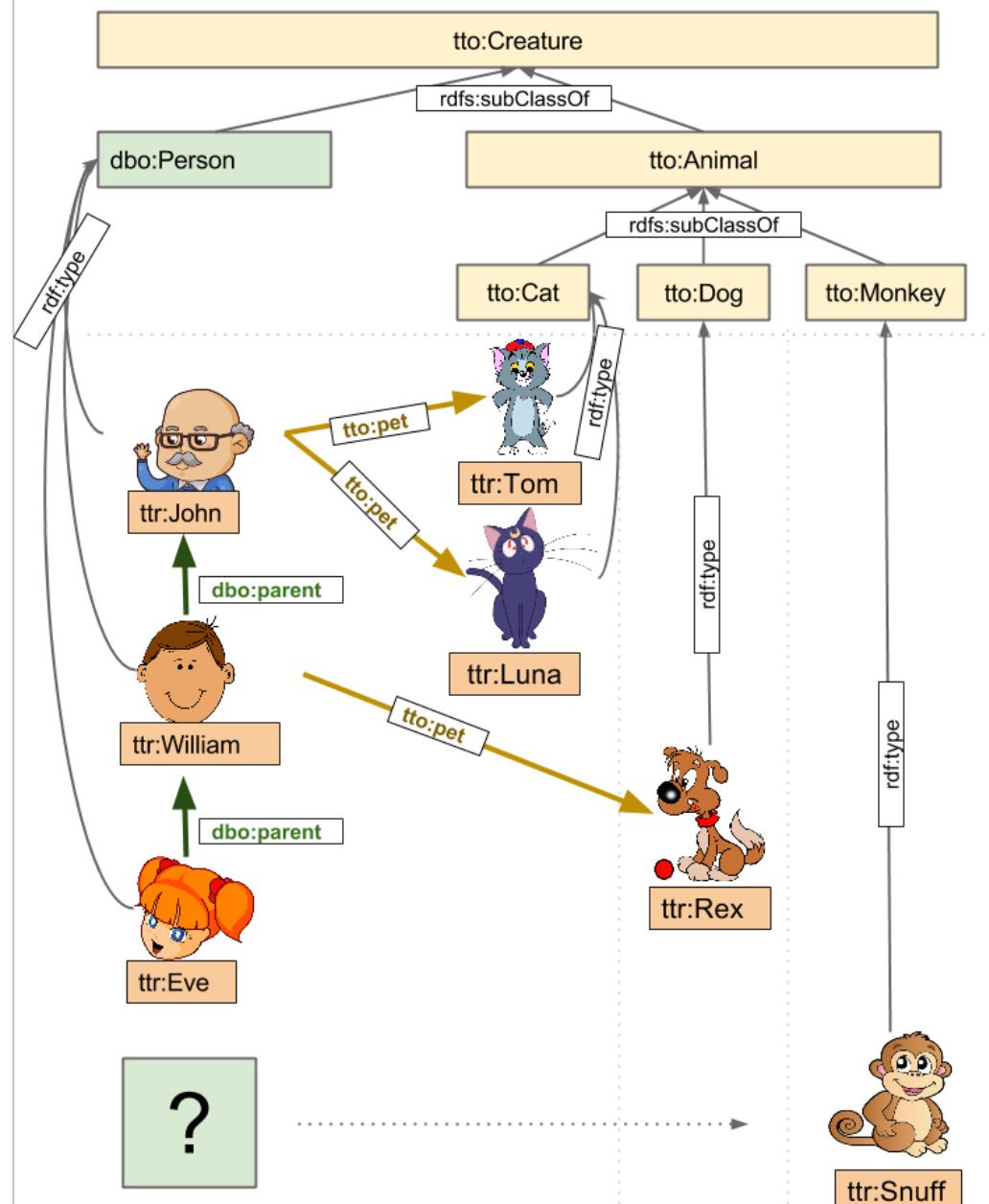
rdf:<<http://www.w3.org/1999/02/22-rdf-syntax-ns#>>

rdfs:<<http://www.w3.org/2000/01/rdf-schema#>>

NAMESPACES

ONTOLOGY

DATA



The Data

Exploration queries

- What does the data look like?
- RDF triple <**s**, **p**, **o**> in Turtle syntax

```
SELECT *
WHERE {
    ?subject ?predicate ?object .
}
LIMIT 10
```

- Asks for a sample of the data available.
 - Always include the LIMIT keyword to avoid problems of resources and memory

Exploration queries

```
SELECT *
WHERE {
    ?subject ?predicate ?object .
}
LIMIT 10
```

s	p	o
dbo:Person	rdfs:subClassOf	tto:Creature
tto:Animal	rdf:type	rdfs:Class
tto:Animal	rdfs:isDefinedBy	tto:
tto:Animal	rdfs:label	"animal"
tto:Animal	rdfs:subClassOf	tto:Creature
tto:Cat	rdf:type	rdfs:Class
tto:Cat	rdfs:isDefinedBy	tto:
tto:Cat	rdfs:label	"cat"
tto:Cat	rdfs:subClassOf	tto:Animal
tto:Creature	rdf:type	rdfs:Class

Exploration queries

- What properties are used?

```
SELECT DISTINCT ?property
WHERE
{ ?s ?property ?o . }
LIMIT 30
```

Exploration queries

Query time is 0.032[s] for 14 rows

property

rdfs:subClassOf

rdf:type

rdfs:isDefinedBy

rdfs:label

rdfs:domain

rdfs:range

rdfs:comment

dbo:parent

dbp:birthDate

dbp:name

tto:sex

tto:pet

tto:color

tto:weight

```
SELECT DISTINCT ?property  
WHERE  
{ ?s ?property ?o . }  
LIMIT 30
```

SPARQL queries

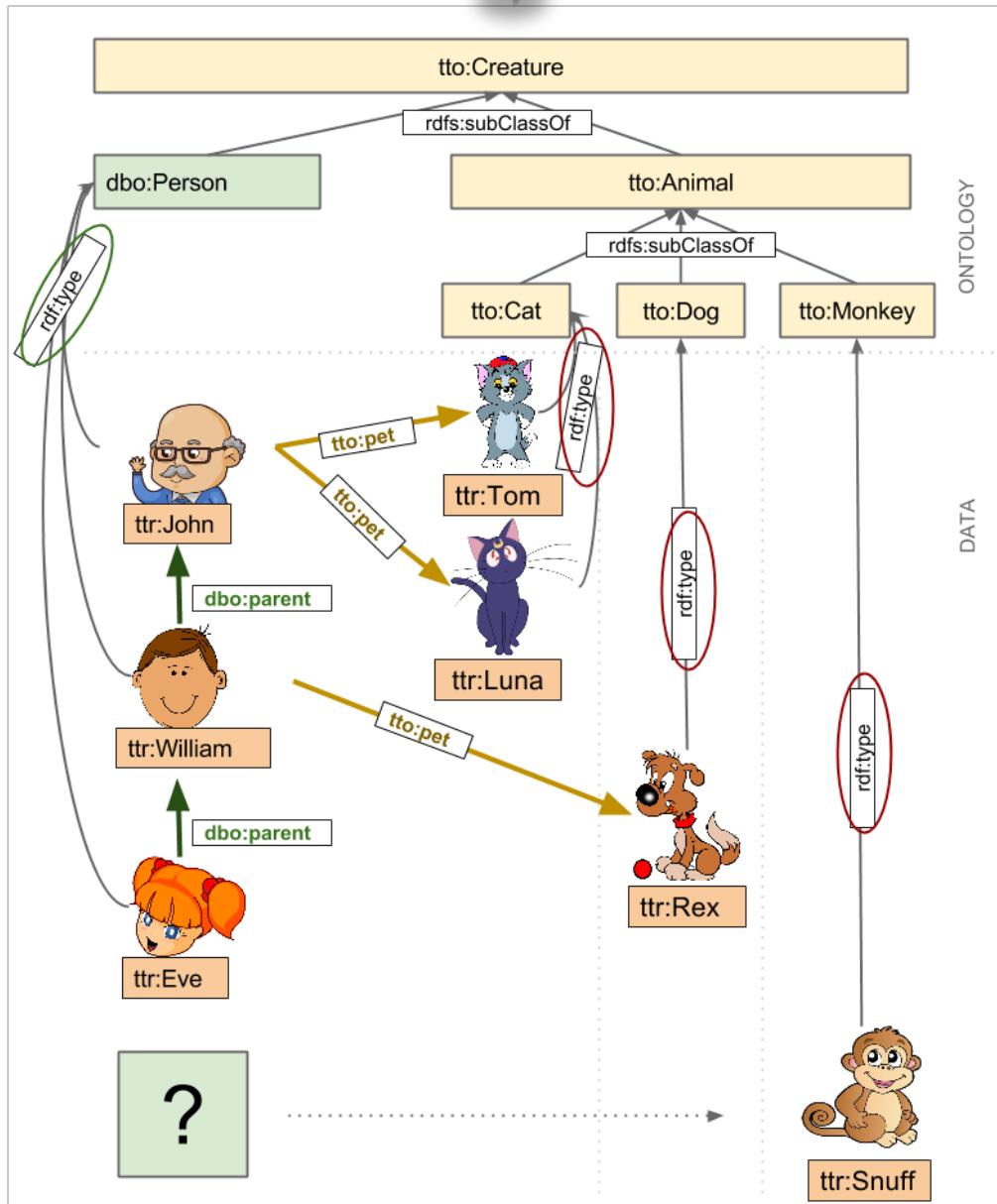
- We could be matching labels of resources
`?subject rdfs:label ?label .`
- All parts of a triple can be variables in the query

Example

- Select resources that are persons

```
SELECT ?thing WHERE {  
    ?thing rdf:type dbo:Person .  
}
```

Query answering wrt the data



Exploration queries

```
SELECT ?thing WHERE {  
    ?thing rdf:type dbo:Person .  
}
```

Query time is 0.06[s] for 3 rows

thing
ttr:Eve
ttr:John
ttr:William

Example

- Find the labels of the various resources

```
SELECT ?subject ?label WHERE {  
    ?subject rdfs:label ?label .  
}
```

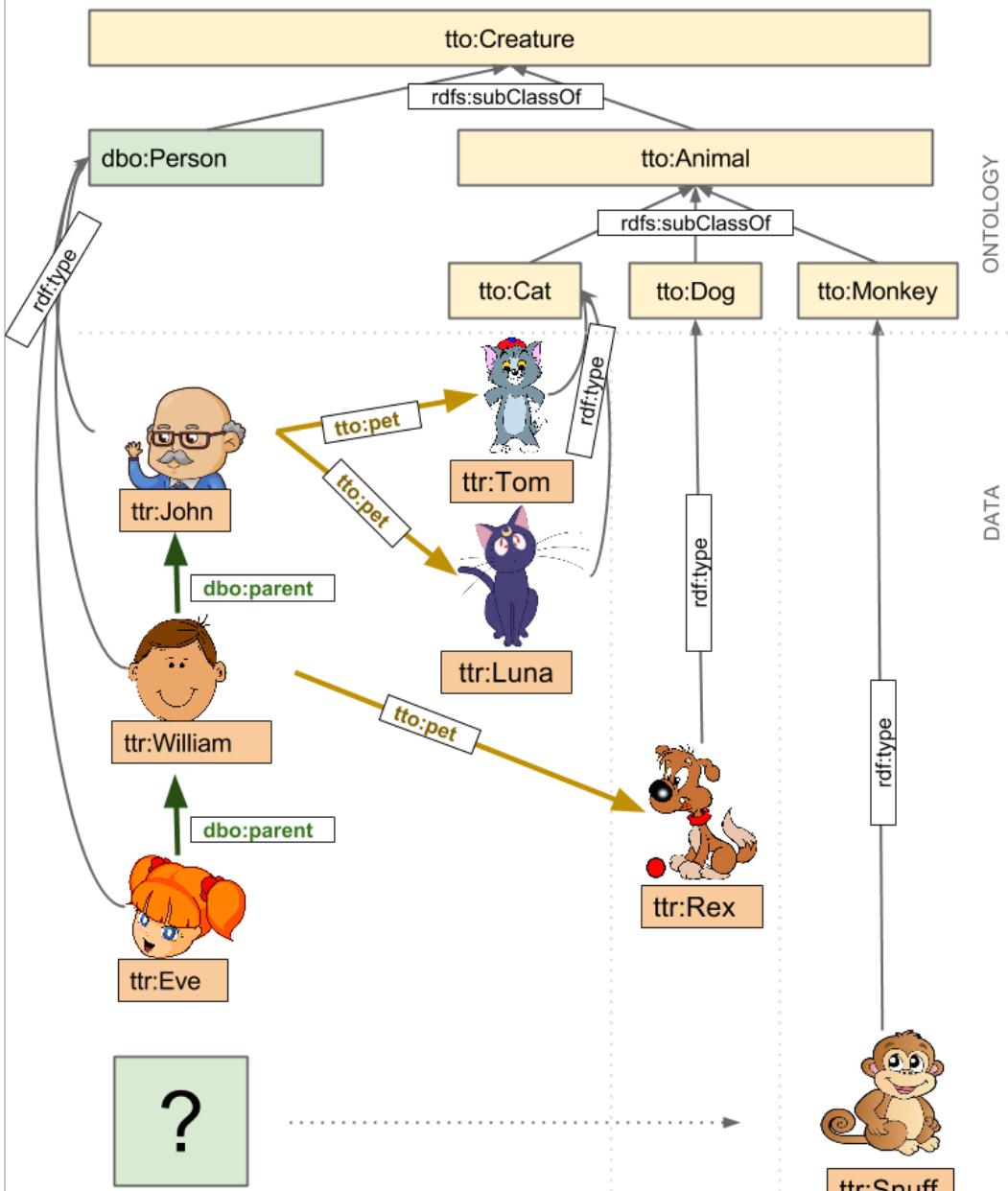
Exploration queries

```
SELECT ?subject ?label WHERE {  
    ?subject rdfs:label ?label .  
}
```

Query time is 0.041[s] for 8 rows

subject	label
tto:Animal	"animal"
tto:Cat	"cat"
tto:Creature	"creature"
tto:Dog	"dog"
tto:Monkey	"monkey"
tto:pet	"domestic animal"
tto:sex	"sex"
tto:weight	"weight"

dbo:<<http://dbpedia.org/ontology/>>
 rdf:<<http://www.w3.org/1999/02/22-rdf-syntax-ns#>>
 rdfs:<<http://www.w3.org/2000/01/rdf-schema#>>
 tto:<<http://example.org/tuto/resource#>>



Now it's your turn

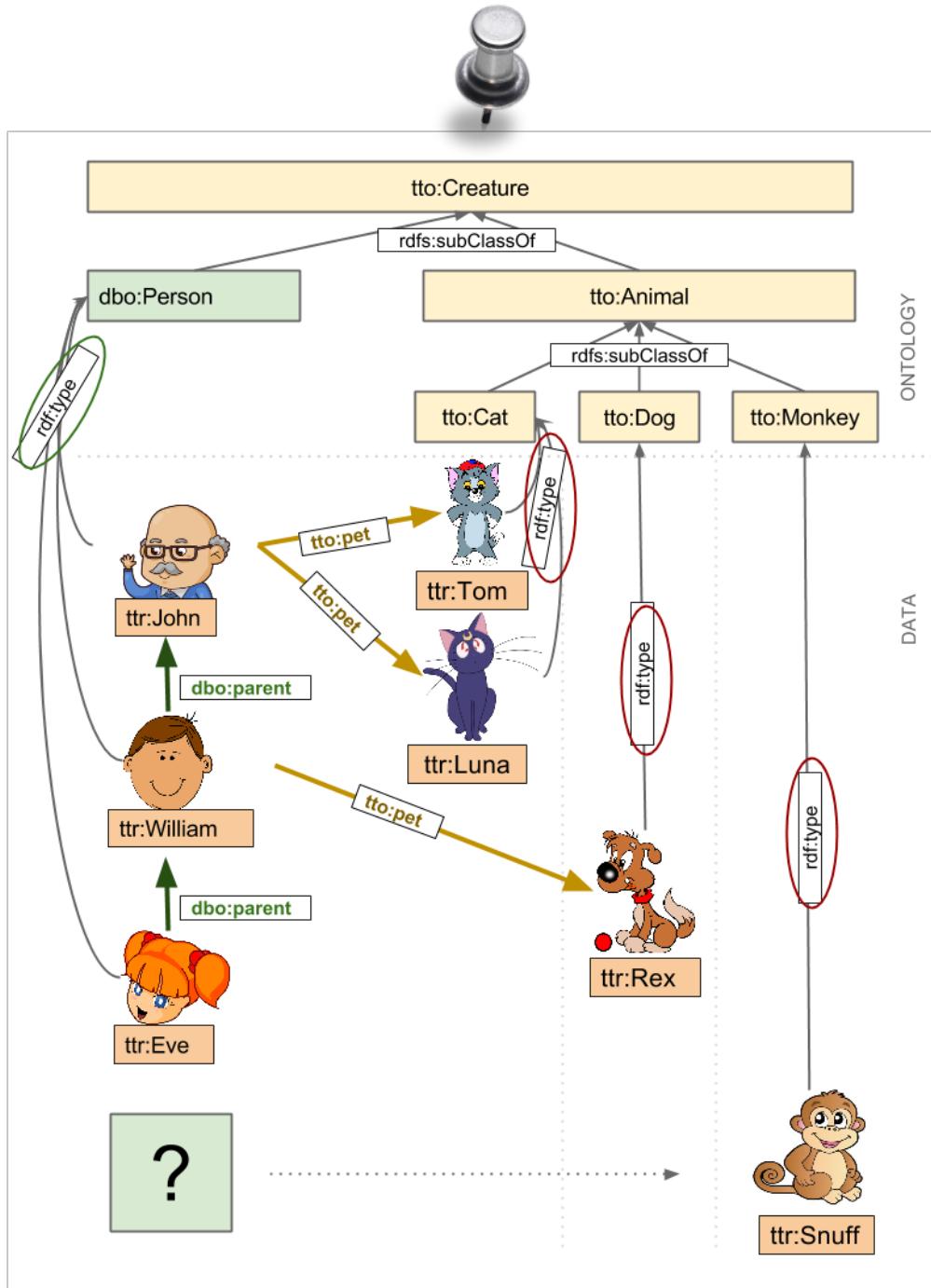
Query 1

- Select things that are cats

Query 1

- Select things that are cats

```
SELECT ?thing WHERE {  
    ?thing rdf:type tto:Cat .  
}
```



Query answering wrt the data

Query 1

```
SELECT ?thing WHERE {  
    ?thing rdf:type tto:Cat .  
}
```

Query time is 0.023[s] for 2 rows

thing

ttr:LunaCat

ttr:TomCat

Query 2

- Select all people who are female

Query 2

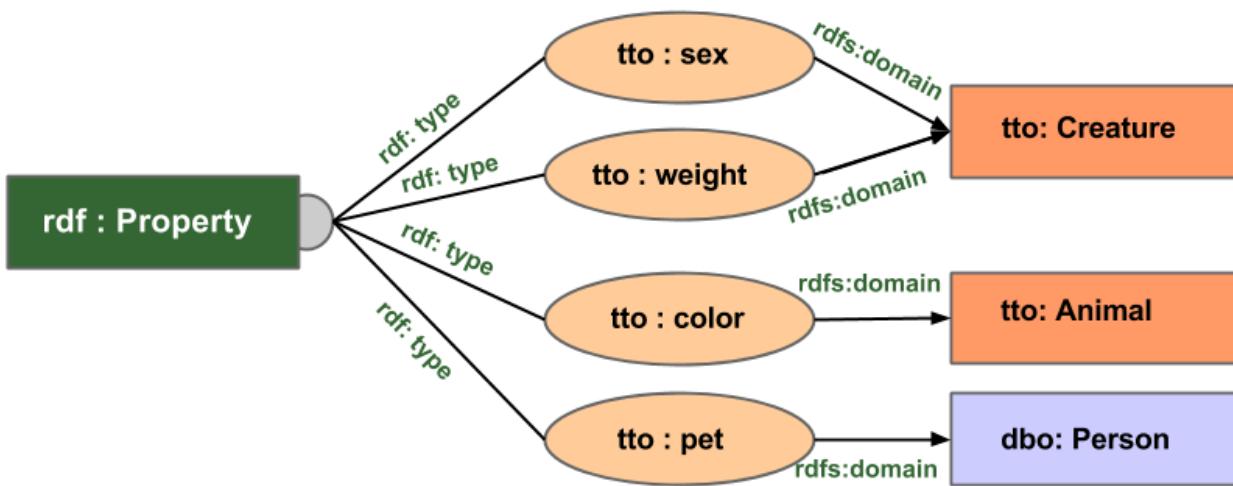
- Select all people who are female

```
SELECT ?thing WHERE {  
    ?thing a dbo:Person .  
    ?thing tto:sex "female" .  
}
```

```
SELECT ?thing WHERE {  
    ?thing a dbo:Person ;  
        tto:sex "female" .  
}
```

Properties

```
@prefix rdf <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .  
@prefix tto: <http://example.org/tuto/ontology#> .  
@prefix dbo: <http://dbpedia.org/ontology/> .
```



```
▼ tto:sex  
  rdf:type rdf:Property;  
  rdfs:label "sex"^^xsd:string;  
  rdfs:domain tto:Creature ;  
  rdfs:range xsd:string ;  
  rdfs:isDefinedBy tto: .  
  
▼ tto:pet  
  rdf:type rdf:Property;  
  rdfs:label "domestic animal"^^xsd:string;  
  rdfs:domain dbo:Person ;  
  rdfs:range tto:Animal ;  
  rdfs:isDefinedBy tto: .  
  
▼ tto:color  
  rdf:type rdf:Property;  
  rdfs:label "hair or fur color"^^xsd:string;  
  rdfs:domain dbo:Animal ;  
  rdfs:range xsd:string ;  
  rdfs:isDefinedBy tto: .  
  
▼ tto:weight  
  rdf:type rdf:Property;  
  rdfs:label "weight"^^xsd:string;  
  rdfs:comment "weight in kilograms"^^xsd:string;  
  rdfs:domain tto:Creature ;  
  rdfs:range xsd:decimal ;  
  rdfs:isDefinedBy tto: .
```

Query 2

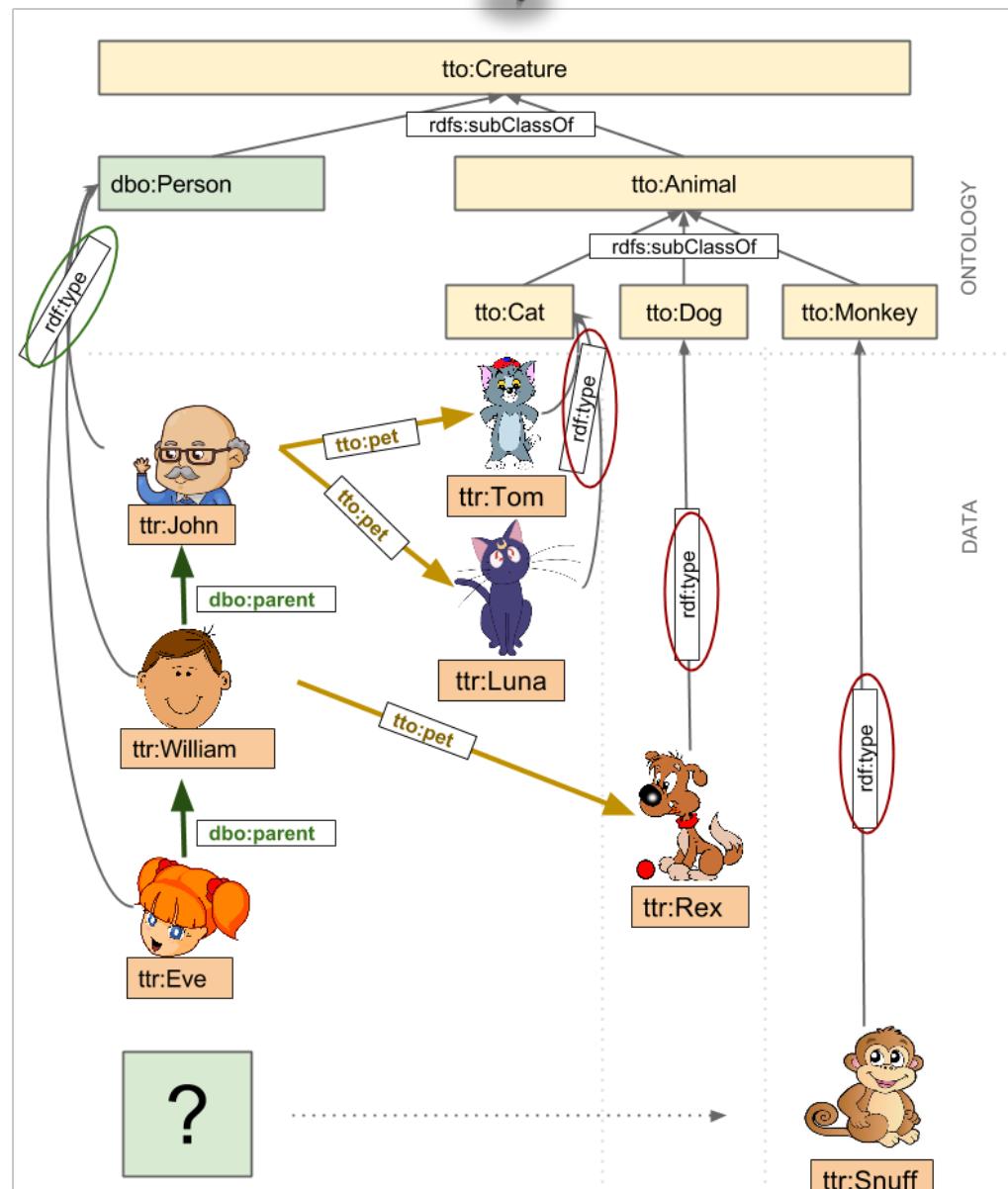
```
SELECT ?thing WHERE {  
  ?thing a dbo:Person ;  
          tto:sex "female" .  
}
```

thing
ttr:Eve

Query 3

- Select all people and their pets

Query 3



Query 3

- Select all people and their pets

```
SELECT ?person ?pet WHERE {  
    ?person rdf:type dbo:Person .  
    ?person tto:pet ?pet .  
}
```

Query 3

```
SELECT ?person ?pet WHERE {  
    ?person rdf:type dbo:Person .  
    ?person tto:pet ?pet .  
}
```

Query time is 0.046[s] for 3 rows

person	pet
ttr:John	ttr:LunaCat
ttr:John	ttr:TomCat
ttr:William	ttr:RexDog

Query 4

- Select all pets and their owners

Query 4

- Select all animals that are pets and their owners

```
SELECT ?person ?pet WHERE {  
    ?pet rdf:type ?x .  
    ?x rdfs:subClassOf tto:Animal .  
    ?person tto:pet ?pet .  
}
```

Query 4

```
SELECT ?person ?pet WHERE {  
    ?pet rdf:type ?x .  
    ?x rdfs:subClassOf  
    tto:Animal.  
    ?person tto:pet ?pet .  
}
```

Query time is 0.046[s] for 3 rows

person	pet
ttr:John	ttr:LunaCat
ttr:John	ttr:TomCat
ttr:William	ttr:RexDog

Query 5

- Count all pets by owner

Query 5

- Count all pets by owner

```
SELECT ?owner (count(?pet) as ?count)
WHERE {
    ?owner tto:pet ?pet .
} GROUP BY ?owner
```

Query 5

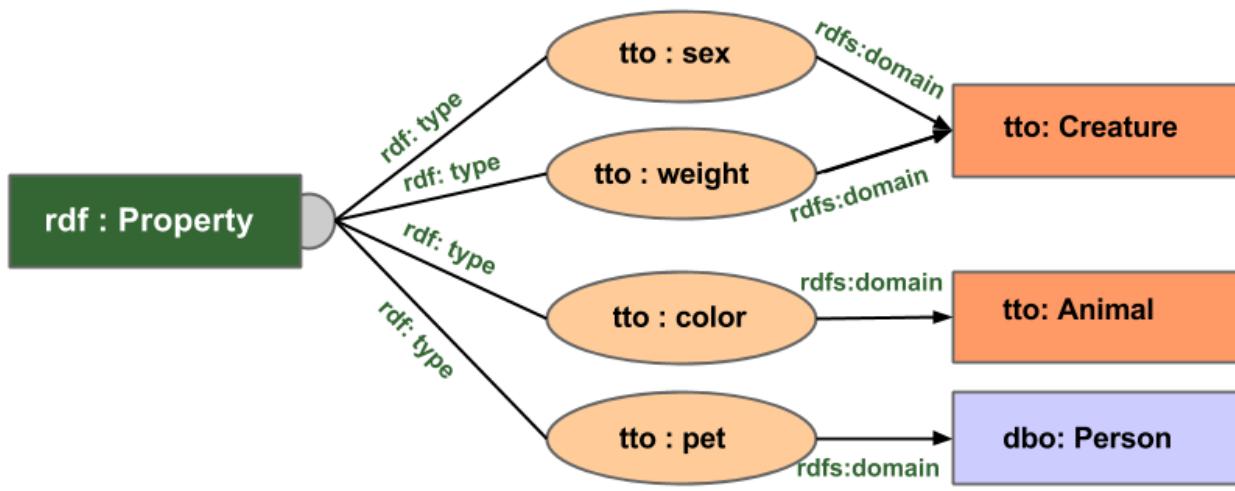
```
SELECT ?owner (count(?pet) as ?  
count) {  
?owner tto:pet ?pet .  
} GROUP BY ?owner
```

Query time is 0.06[s] for 2 rows

owner	count
ttr:John	"2"
ttr:William	"1"

The properties in the data model

```
@prefix rdf <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .  
@prefix tto: <http://example.org/tuto/ontology#> .  
@prefix dbo: <http://dbpedia.org/ontology/> .
```



```
▼ tto:sex  
  rdf:type rdf:Property;  
  rdfs:label "sex"^^xsd:string;  
  rdfs:domain tto:Creature ;  
  rdfs:range xsd:string ;  
  rdfs:isDefinedBy tto: .  
  
▼ tto:pet  
  rdf:type rdf:Property;  
  rdfs:label "domestic animal"^^xsd:string;  
  rdfs:domain dbo:Person ;  
  rdfs:range tto:Animal ;  
  rdfs:isDefinedBy tto: .  
  
▼ tto:color  
  rdf:type rdf:Property;  
  rdfs:label "hair of furr color"^^xsd:string;  
  rdfs:domain dbo:Animal ;  
  rdfs:range xsd:string ;  
  rdfs:isDefinedBy tto: .  
  
▼ tto:weight  
  rdf:type rdf:Property;  
  rdfs:label "weight"^^xsd:string;  
  rdfs:comment "weight in kilograms"^^xsd:string;  
  rdfs:domain tto:Creature ;  
  rdfs:range xsd:decimal ;  
  rdfs:isDefinedBy tto: .
```

Query 6

- Select things that have a weight between 5 and 9 kg, order by weight and specify the type

Query6

- Select things that have a weight between 5 and 9 kg, order by weight and specify the type

```
SELECT ?thing ?weight ?type WHERE {  
  ?thing tto:weight ?weight .  
  ?thing a ?type .  
  FILTER (?weight > 5 && ?weight < 9.0)  
 } ORDER BY ?weight ?type
```

Query 6

```
SELECT ?thing ?weight ?type WHERE {  
    ?thing tto:weight ?weight .  
    ?thing a ?type .  
    FILTER (?weight > 5 && ?weight < 9.0)  
} ORDER BY ?weight ?type
```

Query time is 0.029[s] for 2 rows

thing	weight	type
ttr:TomCat	"5.8"	tto:Cat
ttr:RexDog	"8.8"	tto:Dog

Query 7

- Select the name and year of birth for all the male people

dbo:<<http://dbpedia.org/ontology/>>

tto:<<http://example.org/tuto/ontology/>#>

ttr:<<http://example.org/tuto/resource#>>

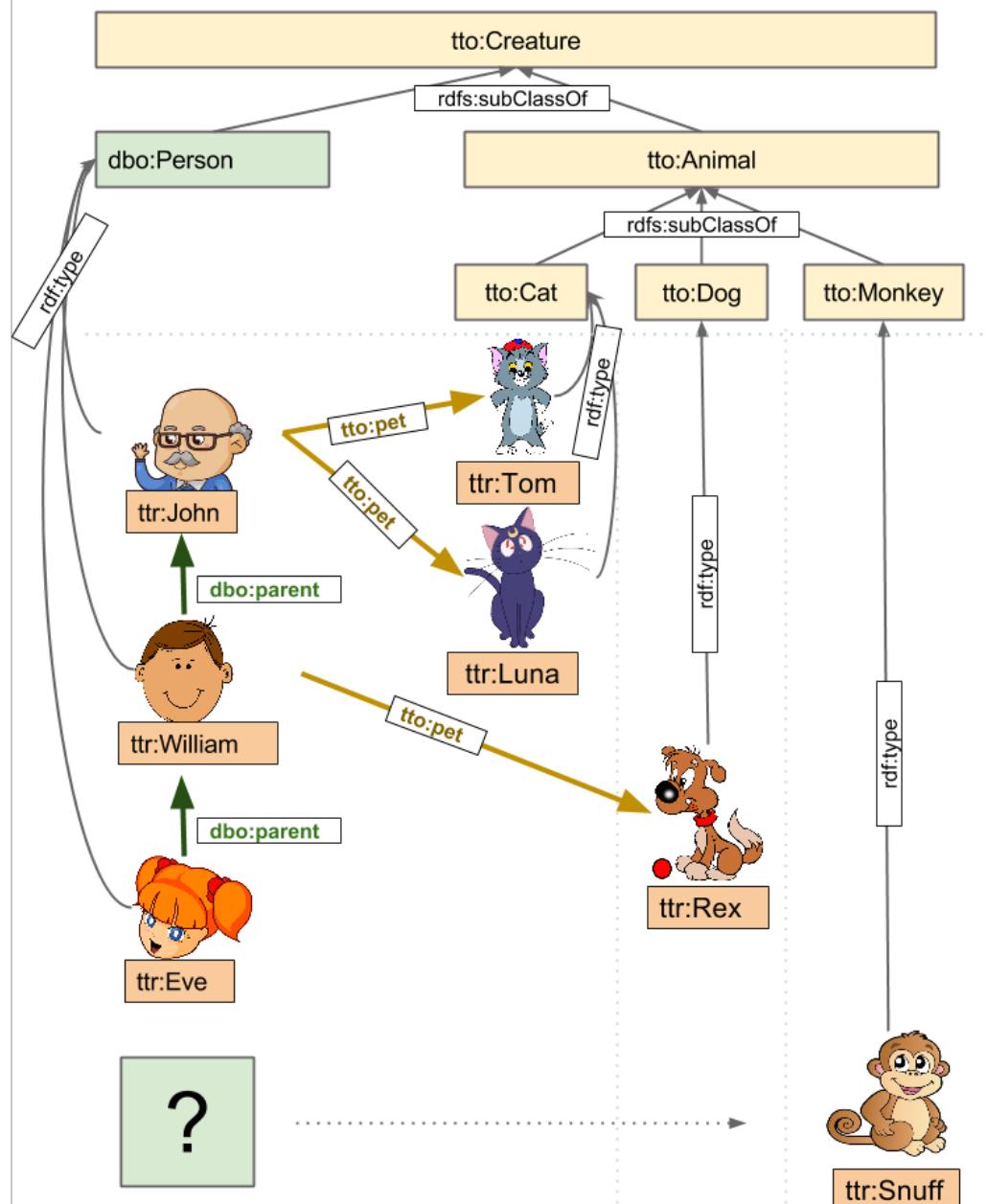
rdf:<<http://www.w3.org/1999/02/22-rdf-syntax-ns#>>

rdfs:<<http://www.w3.org/2000/01/rdf-schema#>>

NAMESPACES

ONTOLOGY

DATA



The Data

Query 7

- Select the name and year of birth for all the male people

```
select ?name ?yearBorn where {
  ?person rdf:type dbo:Person .
  ?person dbp:birthDate ?birth .
  ?person dbp:name ?name .
  ?person tto:sex "male" .
  bind (year(?birth) as ?yearBorn)
}
```

Query 7

```
select ?name ?yearBorn where {  
    ?person rdf:type dbo:Person .  
    ?person dbp:birthDate ?birth .  
    ?person dbp:name ?name .  
    ?person tto:sex "male" .  
    bind (year(?birth) as ?yearBorn)  
}
```

Query time is 0.025[s] for 2 rows

name	yearBorn
"John"	"1942"
"William"	"1978"

Query 8

- Select the direct and indirect subclass of the class Creature

Query 8

- Select the direct and indirect subclass of the class Creature

```
select ?subSpecies where {
  ?subSpecies rdfs:subClassOf+ tto:Creature .
}
```

Query 8

```
select ?subSpecies where {
    ?subSpecies rdfs:subClassOf+ tto:Creature .
}
```

Query time is 0.037[s] for 5 rows

subSpecies
dbo:Person
tto:Animal
tto:Cat
tto:Dog
tto:Monkey

Further reading (recommended)

- Jena SPARQL tutorial:

<http://jena.sourceforge.net/ARQ/Tutorial/>

- SPARQL Query Language for RDF:

<http://www.w3.org/TR/rdf-sparql-query/>

- Turtle - Terse RDF Triple Language:

<http://www.dajobe.org/2004/01/turtle/>

- SPARQL FAQ:

<http://www.thefigtrees.net/lee/sw/sparql-faq>

- Learn about SPARQL 1.1:

<http://www.dajobe.org/talks/201105-sparql-11/>

- YASGUI, YASQE: SPARQL query editors:

<http://yasgui.laurensrietveld.nl/>

<http://yasgui.org/YASQE/>

COMP318: RDFS entailment

www.csc.liv.ac.uk/~valli/Comp318



Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

v.Tamma@liverpool.ac.uk

Where were we

- SPARQL
 - RDF query language
 - Based on simple entailment
 - Syntax and exercises

Semantics

- RDF(S) vocabulary has built-in “meaning”
- RDF(S) Semantics
 - Makes meaning explicit
 - Defines what follows from an RDF graph
- Goals
 - Evaluate the truth of a triple/graph
 - Characterise the state of the world that makes a triple/graph true.

RDF(S) entailment

- An RDF(S) graph entails implicit triples
 - triples not explicitly contained in the graph, but that can be derived from an RDF(S) graph
 - **using the special semantics of the vocabulary of the graph**
 - **vocabulary of the graph:** set of names which occurs as the subject, predicate, object
 - **Interpretations** assign special meaning to the symbols in a particular vocabulary
 - **Interpretations** (Normative)
 - Mapping of RDF assertions into an abstract model, based on set-theory
 - With an “interpretation operator” $I()$, maps a RDF graph into a highly abstract set of high-cardinality sets
 - Highly theoretical model, useful to prove mathematical properties
 - **Entailments** (Informative)
 - Transformation rules to derive new assertions from existing ones
 - May be proven complete and consistent with the formal interpretation

Entailment regimes

- Three entailment regimes
 - **simple entailment**: no particular extra conditions are posed on a vocabulary, including the RDF vocabulary itself;
 - it involved only graph transformations.
 - **RDF entailment**: based on the interpretation of the RDF vocabulary;
 - **RDFS entailment**:
 - based on the interpretation of the RDFS vocabulary
 - some extra conditions are posed by in the form of axiomatic triples and semantic conditions

Inference rules

- An inference rule is a rule of the form:
$$\frac{\phi_1, \phi_2, \dots, \phi_n}{\psi}$$
- where $\phi_1, \phi_2, \dots, \phi_n$ are sentences in the language (**assumptions**), whilst ψ is a new sentence derived from the assumptions (**conclusion**)
- Inference rules are a formal description of the process for constructing new expressions from existing ones.
 - In RDF, inferences corresponding to **entailments** are described as **correct** or **valid**.

Proof theory

- Every formal logic has a set of inference rules that can be used to “prove” some formula μ from a given set of formulas Γ
- A **formal proof** is the sequential application of the inference rules that starts with Γ and ends with μ
 - $\Gamma \vdash \mu, \mu$ can be proved from Γ

Soundness and completeness

- An inference mechanism is **sound** if it derives only sentences that are entailed.
 - If $\Gamma \vdash \mu$ then $\Gamma \models \mu$
- An inference mechanism is **complete** if derives all the sentences that are entailed.
 - If $\Gamma \models \mu$ then $\Gamma \vdash \mu$

RDF inference rules

- The W3C recommendation “RDF Semantics’ provides the inference rules that corresponds to the various form of entailments mentioned;
 - simple entailment
 - RDF entailment
 - RDFS entailment

Notation

- a, b,
 - refer to any arbitrary URI
 - (i.e. anything that can appear in the predicate of a triple)
- u, v,
 - refer to any arbitrary URI or blank node ID
 - (i.e. anything that can appear in the subject of a triple)
- x, y,
 - refer to an arbitrary URI, blank node ID or literal
 - (i.e. anything that can appear in the object of a triple)
- _:n,
 - refer to the ID of a blank node
 - (i.e. appearing as a subject or object)
- |,
 - refers to a literal
 - (i.e. a string that is sometimes found in the object)

RDF Entailment

- The RDF entailment has 4 inference rules:
- **(rdfax)** Infer the triple $u \text{ a } x.$ for every RDF axiomatic triple $u \text{ a } x.$
- **(lg, literal generalisation)** If G contains $u \text{ a } l.$ then infer the triple $u \text{ a } _:n.$
 - Specialised version of SE1 that allows generalisation of a literal by a blank node
 - Other properties of this literal can be inferred via this blank node: e.g. the literal is an instance of a class
 - Literals can only appear as objects in a triple

RDF Entailment

- The RDF entailment has 4 inference rules:
- **(rdf1)** If G contains a triple $u \text{ } a \text{ } y.$ then we can infer
 - $a \text{ } \texttt{rdf:type} \text{ } \texttt{rdf:Property}.$
- **(rdf2)** If G contains a triple $u \text{ } a \text{ } l.$ where l is a well-formed XML literal then we can infer
 - $\text{_:n } \texttt{rdf:type} \text{ } \texttt{rdf:XMLLiteral}.$

RDF entailment

- **Theorem.** A graph G1 RDF-entails a graph G2 if and only if there is a graph G1' that can be derived from G1 by using the rules rdfax, lg, rdf1 and rdf2 such that G1' simply entails G2.

Inference Rules for RDFS-entailment

- Assign “meaning” to the RDFS vocabulary
- **(rdfsx)** Infer the triple $u \text{ } a \text{ } x.$ for every RDFS axiomatic triple $u \text{ } a \text{ } x.$
- **(RDFS1, literal)** If G contains $u \text{ } a \text{ } l.$ where l is a plain literal (with or without language information), then infer the triple:
 - $\text{_:n rdf:type rdfs:literal.}$

Domain and range restrictions

- **(rdfs2)** If G contains a triples a `rdfs:domain` x.
u a y. then we can infer
 - u `rdf:type` x.
- **(rdf3)** If G contains a triples a `rdfs:range` x.
u a v. then we can infer
 - v `rdf:type` x.

Everything is a resource

- **(rdfs4a)** If G contains a triple $u \text{ } a \text{ } x$.then we can infer
 - $u \text{ rdf:type rdfs:Resource}$.
- **(rdfs4b)** If G contains a triple $u \text{ } a \text{ } v$.then we can infer
 - $v \text{ rdf:type rdfs:Resource}$.
- We do not need an inference rule for predicates:
 - the relevant triple can be derived using **rdf1** and **rdfs4**

Important property of binary relations

- binary relation R on a set A is said to be:
 - Reflexive: if $x R x$, for all $x \in A$
 - A number **is equal to** itself
 - Irreflexive: if not $x R x$, for all $x \in A$
 - A number x **is not equal to** $(x+1)$
 - Symmetric: if $x R y$ implies $y R x$, for all $x, y \in A$
 - **marriedTo**: if Ross **marriedTo** Rachel then Rachel **marriedTo** Ross
 - Asymmetric: if $x R y$ not A implies $y R x$, for all $x, y \in A$
 - **parentOf**: if Rachel **parentOf** Emma then it does not imply Emma **parentOf** Rachel
 - Transitive: if $x R y$ and $y R z$ implies $x R z$, for all $x, y, z \in A$
 - **friendOf**: if Monica **friendOf** Joey and Joey **friendOf** Phoebe then Monica **friendOf** Phoebe

Reflexivity and Transitivity of `rdfs:subPropertyOf`

- **(rdfs5)** If G contains the triples
 - u `rdfs:subPropertyOf` v. and v `rdfs:subPropertyOf` x. we can infer
 - u `rdfs:subPropertyOf` x.
- **(rdfs6)** If G contains the triple
 - u `rdf:type rdf:Property`. we can infer
 - u `rdfs:subPropertyOf` u.

More on Subproperties

- **(rdfs7)** If G contains the triples
a `rdfs:subPropertyOf` b. and u a y. we can
infer
 - u b y.

Classes and instances

- **(rdfs8)** If G contains the triple
u `rdf:type rdfs:Class.` we can infer
 - u `rdfs:subClassOf rdfs:Resource.`
- **(rdfs9)** If G contains the triples
u `rdfs:subClassOf x.` and v `rdf:type u.` we can infer
 - v `rdf:type x.`

Reflexivity and Transitivity of rdfs:subClassof

- **(rdfs10)** If G contains the triple
u rdf:type rdfs:Class. we can infer
 - u rdfs:subClassOf u.
- **(rdfs11)** If G contains the triples
u rdfs:subClassOf v. and v rdfs:subClassOf x. we can infer
 - u rdfs:subClassOf x.

Containers

- **(rdfs12)** If G contains the triple
 - u `rdf:type rdfs:ContainerMembershipProperty.` we can infer
 - u `rdfs:subPropertyOf rdfs:member.`

Datatypes

- **(rdfs13)** If G contains the triple
 - u `rdf:type rdfs:Datatype.` we can infer
 - u `rdfs:subClassOf rdfs:Literal.`

Bijection between literals and surrogate bonds

- (**gl inverse of lg**) If G contains the triple
 $u \ a \ _n.$ we can infer
 - $u \ a \ l.$ However:
 - this rule can be applied only when $_n$ identifies a bnode that was introduced earlier by weakening the literal l via the rule **lg**
 - This inference rule is necessary to bring back a literal that has been substituted by a blank node using rule **lg**, then some other inference rule produced a triple with this blank node ($_n$) in the object position, and rule **gl** can now be used to bring this literal back!
 - E.g. `:Murray atp:name "Andy Murray".
 :atp:name rdfs:range atp:PlayerName.`
 - Would entail "`Andy Murray`" a `atp:PlayerName`. which is problematic. Why?

Bijection between literals and surrogate bonds

- The latter triple is not a valid RDF triple
 - a literal should not appear in the subject position!
- thus it will not be inferred (the domain of the ?v variable in the rdfs3 rule would prevent the inference). And so, to achieve the valid inference:
 - `_ :AndyMurray a atp:PlayerName .`Requires the surrogate blank node (`_ :AndyMurray`) to be used through the rule lg.
 - The inverse rule gl then allows surrogates to “travel” back as literals into the object position, though examples of such behaviour are not necessarily intuitive.

Conclusion

- RDF entailment rules
- RDFS entailment rules

Take Test: Mock Exam

Test Information

Description

Instructions This exam paper consists of 4 questions. Please complete all questions in the paper by typing your answers in the appropriate text boxes in VITAL. You might choose to work offline, but the answers will need to be entered in VITAL. Please note you only have one attempt to complete the exam paper, no multiple attempts are allowed. Please carefully check your work before submitting.

As this is not a timed exam, you are able to return to the exam paper at any time during the period May 11th -- May 13th. Please make sure you save your answers as often as possible, and that you submit the exam when you have finished. The exam is implemented as a VITAL test, but the auto-submit option is not enabled and therefore you will have to submit your test when you have finished, and before the end of May 13th.

The total time that you are expected to use to complete this paper is 1 hour.

Question Completion Status:

Please make sure that you read the wording of each question carefully, and double check your answers before submitting.

Multiple Attempts Not allowed. This Test can only be taken once.

Force Completion This Test can be saved and resumed later.

QUESTION 1

10 points

Discuss the definition of ontology in computer science and explain the role they play in ontology based information systems. (4 marks are awarded for the discussion of the definition and 6 marks are awarded for explaining how ontologies are used in ontology based information systems)

Click Save and Submit to save and submit. Click Save All Answers to save all answers.

QUESTION 2**20 points**

Explain what inference rules are and how they are used in the context of reasoning with RDF(S). **(5 marks)**

Given the RDF graph G below:

1. :Person a rdfs:Class.
2. :Man a rdfs:Class;
3. rdfs:subClassOf :Person.
4. :Parent a rdfs:Class;
5. rdfs:subClassOf :Person.
6. :Father a rdfs:Class;
7. rdfs:subClassOf :Parent;
8. rdfs:subClassOf :Man.
9. :Child a rdfs:Class;
10. rdfs:subClassOf :Person.
11. :hasParent a rdf:Property;
12. rdfs:domain :Person;
13. rdfs:range :Parent.
14. :hasFather a rdf:Property;
15. rdfs:subpropertyOf :hasParent;
16. rdfs:domain :Person;
17. rdfs:range :Parent.

▼ Question Completion Status:

18. rdfs:domain :Person;
19. rdfs:range :Parent.
20. stella a :Person;
21. :hasFather :paul.
22. paul a :Man.

For each of the statement below, decide if the graph G entails the given statement(s) and explain why this is the case, or why this is not the case. If the answer the statement(s) is entailed by G, then use the appropriate entailment rules to prove that your answer is correct. If the answer is "no", then explain, informally or formally, why this is so. Each answers to the following statements is worth 5 marks:

1. :Father rdfs:subClassOf :Person .
2. :paul a :Parent .
3. :stella :hasParent _:x .

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Path: p

Words:0

QUESTION 3**16 points**

Click Save and Submit to save and submit. Click Save All Answers to save all answers.

Arial

▼ 3 (12pt) ▼

Path: p

Words:0

QUESTION 4**24 points****Save Answer**

Represent the following statements in the most appropriate ontology language. Justify your choice. Please note some statements can be represented in more than one language whilst others can only be modelled using one ontology language. The modelling of the statement is awarded 1 mark, and 2 marks are awarded for the discussion of the appropriate ontology language to use.

Model representing the statements listed below by using the following URIs -

▼ Question Completion Status:

- a. Person, woman, and man are classes;
- b. Man is a subclass of Person;
- c. isWifeOf and isHusbandOf are functional properties;
- d. isWifeOf has domain Woman and range Man;
- e. paul is a Man;
- f. Wife is **not** a Person who is not a Husband;
- g. a Husband is someone who has at least been married once;
- h. LifetimeWife is a Wife who has one husband and only one husband;

Arial

▼ 3 (12pt) ▼

Path: p

Words:0

Click Save and Submit to save and submit. Click Save All Answers to save all answers.

COMP318: Ontology alignment

www.csc.liv.ac.uk/~valli/Comp318

Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

V.Tamma@liverpool.ac.uk



based on slides by V. Tamma, T.R. Payne, J. Euzenat, K. Janowicz and G. Schreiber

Exercise

- Let us consider an ontology alignment system, A.

Let us assume we also have a gold standard alignment handcrafted by an expert R composed by 20 correspondences.

- A returns 8 relevant correspondences and 10 non relevant ones.
- Calculate precision, recall and F-measure

Precision

$$\text{Prec}(A_i, R) = \frac{|A_i \cap R|}{|A_i|}$$

Recall

$$\text{Rec}(A_i, R) = \frac{|A_i \cap R|}{|R|}$$

F-measure

$$F_{\text{measure}}(A_i, R) = 2 \times \frac{\text{Prec}(A_i, R) \times \text{Rec}(A_i, R)}{\text{Prec}(A_i, R) + \text{Rec}(A_i, R)}$$

Exercise

- A manually constructed gold standard alignment R between ontology O and O' is composed of 38 correspondences. Let us assume that we need to compare two alignment systems:
 - System A_1 generates 22 correspondences, 18 of which are present in R ;
 - System A_2 generates 14 correspondences, 12 of which are present in R .
- Calculate precision and recall for the two systems. Can we use either measure in isolation to evaluate the alignment systems?

Precision

$$\text{Prec}(A_i, R) = \frac{|A_i \cap R|}{|A_i|}$$

Recall

$$\text{Rec}(A_i, R) = \frac{|A_i \cap R|}{|R|}$$

F-measure

$$F_{\text{measure}}(A_i, R) = 2 \times \frac{\text{Prec}(A_i, R) \times \text{Rec}(A_i, R)}{\text{Prec}(A_i, R) + \text{Rec}(A_i, R)}$$