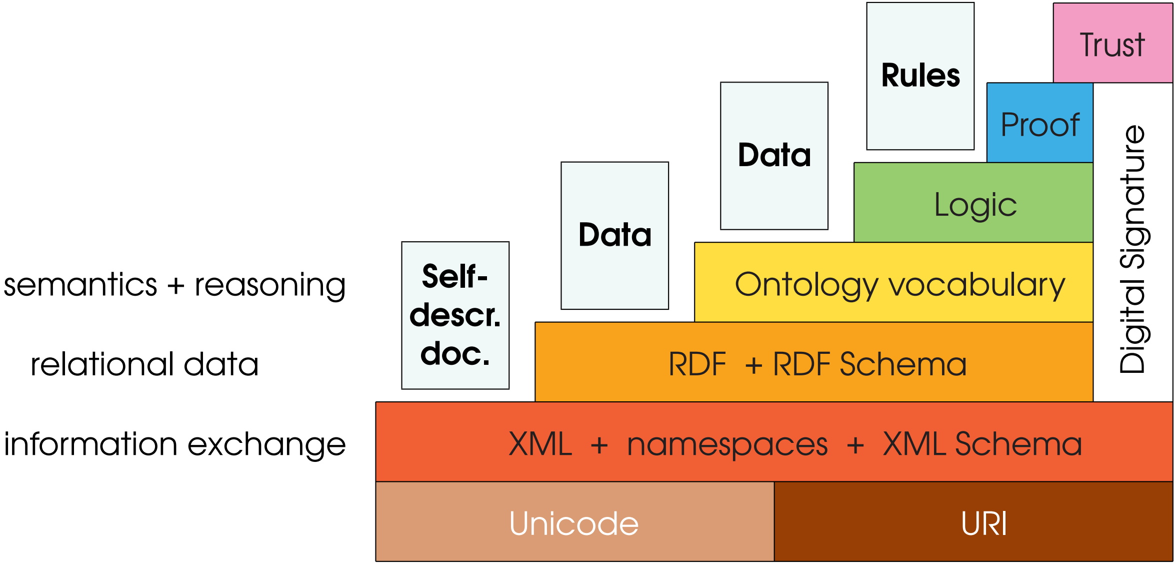
Semantic technologies: the layered approach

(by T. Berners-Lee)



Databases all over the world contain billions of items of data, but some of them are stored in proprietary database formats, while some are available only in human-readable form. To make all this data accessible to computers, we must do it with a language they can 'understand'

Why not XML?

XML doesn′t provide any means for talking about the **semantics** of data

Michael is a lecturer of the Semantic Technologies module

```
<module name=″Semantic Technologies″>
   <lecturer>Michael</lecturer>
</module>
```

or

```
<lecturer name=″Michael″>
   <module>Semantic Technologies</module>
</lecturer>
```

or

```
<teachingOffering>
   <module>Semantic Technologies</module>
   <lecturer>Michael</lecturer>
</teachingOffering>
```

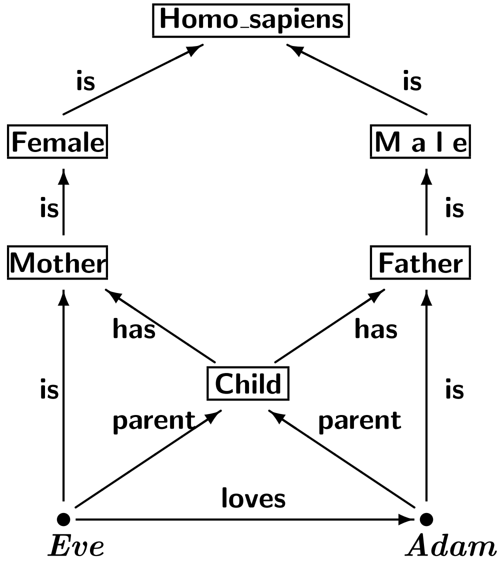
How to assign ‘meaning’ to tag nesting?

Let us see how other disciplines represent such data

Semantic networks

A **semantic network** is a graphic notation for representing **knowledge** in patterns of interconnected nodes and edges:

- the nodes represent **objects** or **concepts**
- the links represent semantic **relations** between nodes



Eve **is** a **mother**

Eve **loves** *Adam*

Every **child** **has** a **father**

Every **mother** **is** a **female**

Every **female** **is** a **human being**

. . .

Semantic networks were first developed for AI and machine translation in the 1960s, but earlier versions have long been used in philosophy, psychology, and linguistics.

Grammar: subject, predicate, object

According to a tradition going back to Aristotle, every sentence can be divided into two main constituents, one being the **subject** and the other its **predicate**.

The **subject** of a sentence is sometimes defined as the argument that generally refers to the origin of the action or the undergoer of the state shown by the predicate.

The **predicate** is the rest of the sentence apart from the subject.
A **predicate** is an expression that **can be true** of something.

An **object** in grammar is a sentence element and part of the sentence **predicate**.
It denotes somebody or something involved in the subject's 'performance' of the verb.

- **Adam loves Eve**
- **MZ teaches ST**
- **MZ is a lecturer**

Mathematical logic: relations or predicates

In mathematical logic, these sentences are represented by means of **relations** or **predicates**:

loves(adam, eve), teaches(mz, st), lecturer(mz)

- A **(binary) relation** R between sets A and B is some set of ordered pairs (x, y) such that $x \in A$ and $y \in B$. If $A = B$ then R is a relation on A.

  e.g., all pairs $(\mathit{person1}, \mathit{person2})$ such that $\mathit{person1}$ loves $\mathit{person2}$

- The **domain** of R is the set of all objects x such that $(x, y) \in R$ for some y

- The **range** of R is the set of all objects y such that $(x, y) \in R$ for some x

More generally, an **n-ary relation**, for $n \geq 1$, on some set A (such as: people or numbers or even all things in the universe) is a set of n-tuples $(x_1, x_2, \dots, x_n)$ of elements of A.

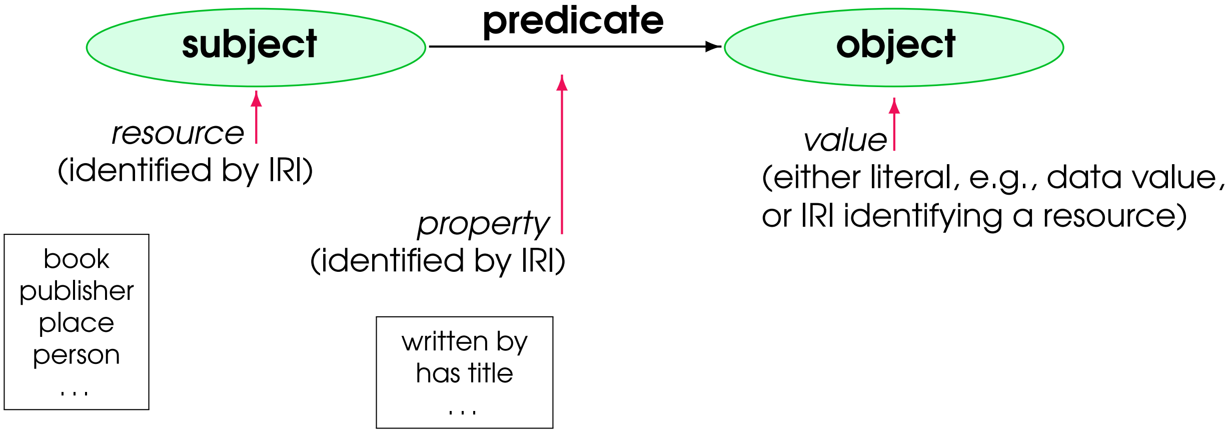
- A **1-ary (unary) relation** on A is simply a subset of A.

  Unary relations are also called **classes**

RDF (graphs instead of trees)

RDF stands for **Resource Description Framework** `http://www.w3.org/RDF/`

- **Resources** are identified by IRIs
- **Statements** describe properties of resources by means of triples of the form



subject —**predicate**→ **object**

resource
(identified by IRI)

property
(identified by IRI)

value
(either literal, e.g., data value,
or IRI identifying a resource)

book
publisher
place
person
. . .

written by
has title
. . .

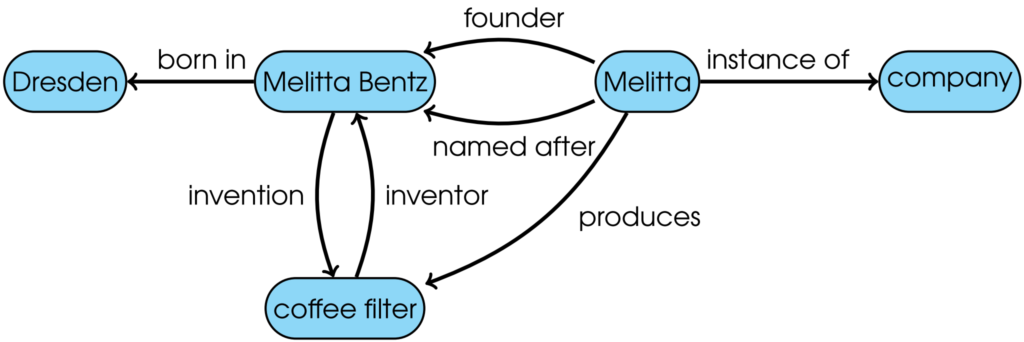
- **Properties** are identified by IRIs and, therefore, are **resources**
- there are also **blank nodes** that do not identify specific resources

RDF Graph

A collection of RDF statements can be represented as a **graph**, which is:

- **directed** (edges have a source and a target)
- **edge-labelled** (each edge has one label)
- a restricted form of **multi-graphs** (there may be multiple edges between the same vertices, but only if they have different labels)
- (partially) **vertex-labelled**: blank nodes are not labelled by IRIs or literals

Example of such a graph:



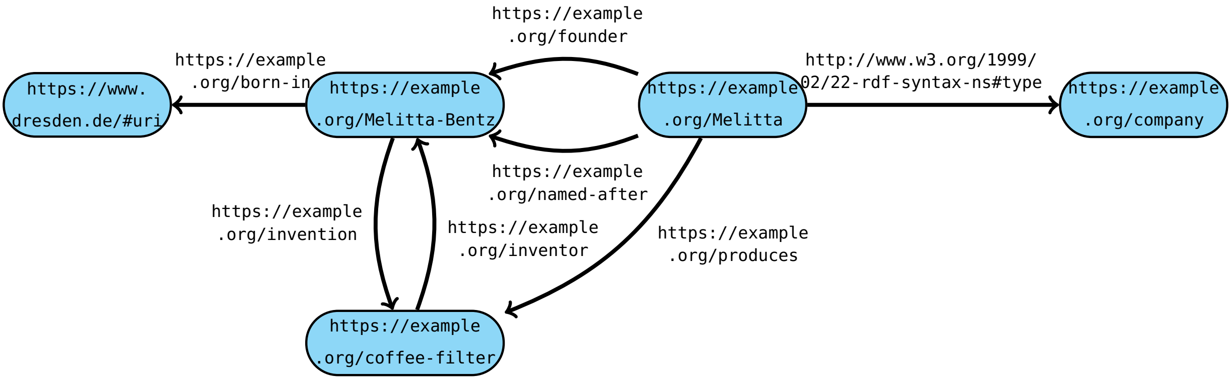
What does it say? Identify triples, their subject, predicate and object

IRIs as labels

But recall that actually subjects, predicates and (some) objects in RDF are IRIs:

- IRIs define resources that appear as vertices in the graph
- IRIs are used as arrow (property) labels

So our example RDF graph should look like


https://www.dresden.de/#uri
https://example.org/born-in
https://example.org/Melitta-Bentz
https://example.org/founder
https://example.org/named-after
https://example.org/Melitta
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
https://example.org/company
https://example.org/invention
https://example.org/inventor
https://example.org/produces
https://example.org/coffee-filter

NB. It is not always obvious what an IRI is supposed to refer to, and many IRIs may refer to the same thing — we cannot assume that all RDF data in the world is integrated.

Which IRIs to use?

Where do the IRIs that we use in RDF graphs come from?

- They can be newly created for an application
  ⇝ avoid confusion with resources in other graphs
- They can be IRIs that are already in common use
  ⇝ support information integration and re-use across graphs

Guidelines for creating new IRIs:

1. Check if you could re-use an existing IRI ⇝ avoid duplication if feasible
2. Use http(s) IRIs ⇝ useful protocols, registries, resolution mechanisms
3. Create new IRIs based on domains that you own ⇝ clear ownership; no danger of clashing with other people's IRIs
4. Don't use URLs of existing web pages, unless you want to store data about pages ⇝ avoid confusion between pages and more abstract resources
5. Make your IRIs return some useful content via http(s) ⇝ helps others to get information about your resources

Why IRIs?

IRIs may seem a bit complicated

- They look a bit technical and complex
- They are hard to display or draw in a graph
- The guidelines just given may seem quite demanding to newcomers

However, it's not that hard:

- RDF can work with any form of IRI (most tools would probably accept any Latin letter string with a colon inside!)
- The guidelines help sharing graphs across applications — a strength of RDF
- Internet domain name registration is a very simple way to define ownership in a global data space
- IRIs should not be shown to users (we'll introduce human-readable labels)

In RDF, IRIs typically look like 'normal' URLs, often with **fragment identifiers** to point at specific parts of a document (such as a section in HTML) #

`http://dublincore.org/usage/documents/principles/#element`

Data values

IRIs can represent anything, but **data values** (numbers, strings, times, . . .) should **not** be represented by IRIs!

Why not use IRIs here too?

1. Data values are the same everywhere
   ⇝ no use in application-specific IRIs

2. Many RDF-based applications need a built-in understanding of data values (e.g., for sorting content)

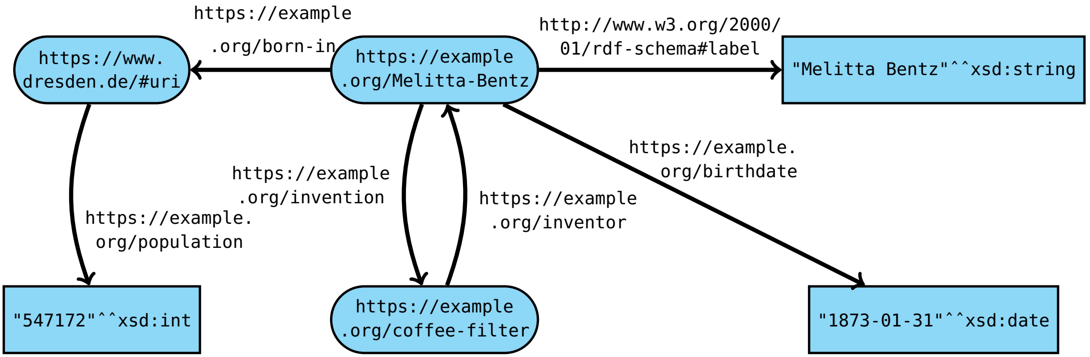
3. Data values are usually more 'interpreted' than IRIs.

   Using a hypothetical scheme '`integer`', the IRIs `integer:42` and `integer:+42` would be different, but intuitively they should represent the same number.

Encoding data values

- Data values in RDF are written as `"lexical value"^^datatype-IRI`
- They are drawn as rectangular nodes in RDF graphs

Example


https://www.dresden.de/#uri
https://example.org/born-in
https://example.org/Melitta-Bentz
http://www.w3.org/2000/01/rdf-schema#label
"Melitta Bentz"^^xsd:string
https://example.org/birthdate
https://example.org/invention
https://example.org/inventor
https://example.org/population
"547172"^^xsd:int
https://example.org/coffee-filter
"1873-01-31"^^xsd:date

RDF supports many datatypes, most of which based on XML Schema (“xsd”): `string`, `boolean`, `integer`, `float`, `dateTime`, `date`, `time`, `gYear`, etc.; see https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#section-Datatypes

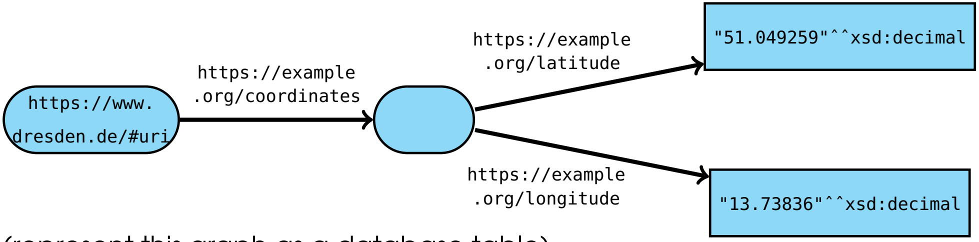
“string”@language: `"Pommes Frites"@de`, `"chips"@en-UK`, `"French fries"@en-US`

Blank nodes

RDF also supports vertices that are not identified by an IRI:
they are called **blank nodes** or **bnodes**

- Intuitively, bnodes are placeholders for some specific (but unspecified) nodes
- Their use makes the claim: ‘there is something at this position’
- Similar to existentially quantified variables ($\exists x$, i.e., ‘there exists an x’) in logic and maths

Example: Blank nodes have historically been used for **auxiliary vertices**


https://www.dresden.de/#uri
https://example.org/coordinates
https://example.org/latitude
"51.049259"^^xsd:decimal
https://example.org/longitude
"13.73836"^^xsd:decimal

(represent this graph as a database table)

We’ll discuss other uses of blank nodes later

RDF Graphs: definition

We have defined all the necessary types of **RDF terms**: IRIs, bnodes, literals.

An **RDF graph** is a set of **triples** consisting of the following parts:

- a **subject**, which is an IRI or blank node
- a **predicate**, which is an IRI
- an **object**, which is an IRI, blank node, or literal

Notes:

- This view resembles a (labelled) adjacency list encoding (for representing graphs)
- The restrictions on the use of bnodes and literals in triples are a bit arbitrary
- RDF graphs are mostly syntactic (rather what we write than what we mean)
- In particular, literals are not interpreted when defining graphs
  - $\leadsto$ multiple ways of writing the same value lead to multiple graphs
  - $\leadsto$ ill-formed literals are allowed in graphs

RDF Serialisations

What we outlined so far is the **abstract syntax** of RDF. To **exchange** RDF graphs, we need concrete representation languages.

There are numerous syntactic formats available:

- **N-Triples** as a simple line-based format
- **Turtle** adds convenient abbreviations to N-Triples
- **JSON-LD** for encoding RDF graphs in JSON
- **RDF/XML** for encoding RDF graphs in XML
- **RDFa** for embedding RDF graphs into HTML (RDF in Attributes)
- ...

Further historic/unofficial formats exist but are hardly relevant today.

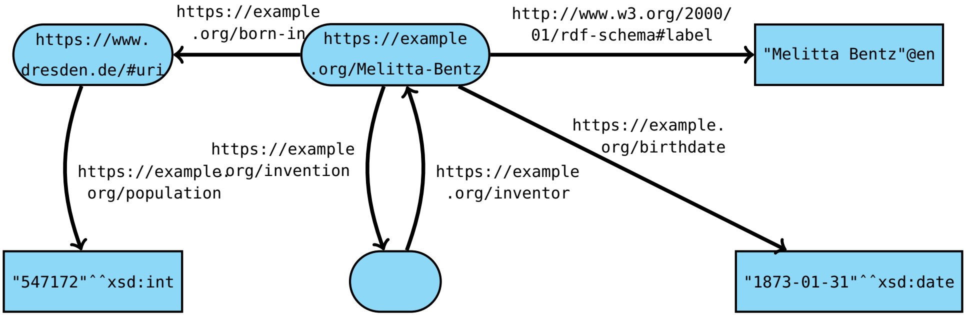
N-Triples

N-Triples is almost the simplest format conceivable:

- Each line encodes one triple:
  - IRIs are written in pointy brackets, e.g., `<https://www.dresdenrespekt.de/>`
  - Literals are written as usual with a given type IRI, e.g., `"2018-10-21"^^<http://www.w3.org/2001/XMLSchema#date>` or `"Wir sind mehr"@de`
  - Blank nodes are written as `_:stringId`, where `stringId` is a string that identifies the blank node within the document (no global meaning!)
  - Parts are separated by whitespace, and lines end with `.`
- Unicode is supported, but various escape sequences also work
- Comments are allowed after triples (nowhere else); they start with #

Full specification at https://www.w3.org/TR/n-triples/

Example



could be encoded as the following N-triples:

```
<https://example.org/Melitta-Bentz> <http://www.w3.org/2000/01/rdf-schema#label> "Melitta Bentz"@en .
<https://example.org/Melitta-Bentz> <https://example.org/birthdate>
                                        "1873-01-31"^^<http://www.w3.org/2001/XMLSchema#date> .
<https://example.org/Melitta-Bentz> <https://example.org/invention> _:1 .
<https://example.org/Melitta-Bentz> <https://example.org/born-in> <https://www.dresden.de/#uri> .
<https://www.dresden.de/#uri> <https://example.org/population>
                                             "547172"^^<http://www.w3.org/2001/XMLSchema#int> .
_:1 <https://example.org/inventor> <https://example.org/Melitta-Bentz> .
```

N-Triples: Summary

Advantages

- Very simple
- Fast and easy to parse
- Processable even with basic text-processing tools, e.g., grep

Disadvantages:

- Somewhat inefficient in terms of storage space
- Not particularly human-friendly (reading and writing)

Turtle: Terse RDF Triple Language

The Turtle format extends N-Triples with several convenient abbreviations:

- Prefix declarations and base namespaces allow us to shorten IRIs
- If we terminate triples with `;` (or with `,`) then the next triple is assumed to start with the same subject (respectively, the same subject and predicate)
- Blank nodes can be encoded using square brackets; they might contain predicate-object pairs that refer to the blank node as subject
- More liberal support for comments (possibly on own line)
- Simpler forms for some types of data values

There are several other shortcuts and simplifications. Full specification is at `https://www.w3.org/TR/turtle/`

PREFIX and BASE by example

- BASE is used to declare a base IRI, so that we can use **relative IRIs**
- PREFIX is used to declare **abbreviations** for IRI prefixes

A Turtle document for the previous example (on page 18):

```
BASE <https://example.org/>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
<Melitta-Bentz>  rdfs:label  “Melitta Bentz”@en  .
<Melitta-Bentz>  <birthdate>  "1873-01-31"^^xsd:date  .
<Melitta-Bentz>  <invention>  _:1  .
<Melitta-Bentz>  <born-in>  <https://www.dresden.de/#uri>  .
<https://www.dresden.de/#uri>  <population>  "547172"^^xsd:int  .
_:1  <inventor>  <Melitta-Bentz>  .
```

NB. Relative IRIs are still written in < and > (e.g., `<birthdate>`);
prefixed names are written without brackets (e.g., `rdfs:label`).

Use of semicolon by example

If a triple ends with ; the next triple is assumed to start with the same subject

If a triple ends with , the next triple is assumed to start with the same subject and predicate

We can write the previous example as follows:

```
BASE <https://example.org/>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>

<Melitta-Bentz>  rdfs:label  "Melitta Bentz"@en  ;
                 <birthdate>  "1873-01-31"^^xsd:date  ;
                 <invention>  _:1  ;
                 <born-in>  <https://www.dresden.de/#uri>  ;
<https://www.dresden.de/#uri>  <population>  "547172"^^xsd:int  .
_:1  <inventor>  <Melitta-Bentz>  .
```

Brackets for bnodes by example

The expression [] represents a bnode (without id)

predicate-object pairs within [...] are allowed to give further triples with the bnode as subject

We can write the previous example as follows:

```
BASE <https://example.org/>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>

<Melitta-Bentz>  rdfs:label  "Melitta Bentz"@en  ;
                 <birthdate>  "1873-01-31"^^xsd:date  ;
                 <invention>  [ <inventor>  <Melitta-Bentz> ]  ;
                 <born-in>  <https://www.dresden.de/#uri>  ;
<https://www.dresden.de/#uri>  <population>  "547172"^^xsd:int  .
```

(more examples will be given later)

Abbreviating numbers and Booleans

Numbers can be written without quotes and type for literals in default types (integer, decimal, or double)

Booleans can also be written as `true` or `false` directly

We can write the previous example as follows:

```
BASE <https://example.org/>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>

<Melitta-Bentz>  rdfs:label  "Melitta Bentz"@en  ;
                 <birthdate>  "1873-01-31"^^xsd:date  ;
                 <invention>  [ <inventor>  <Melitta-Bentz> ]  ;
                 <born-in>  <https://www.dresden.de/#uri>  ;
<https://www.dresden.de/#uri>  <population>  547172  .
```

Turtle: Summary

Advantages:

- Still quite simple
- Not hard to parse
- Human-readable (if formatted carefully)

Disadvantages:

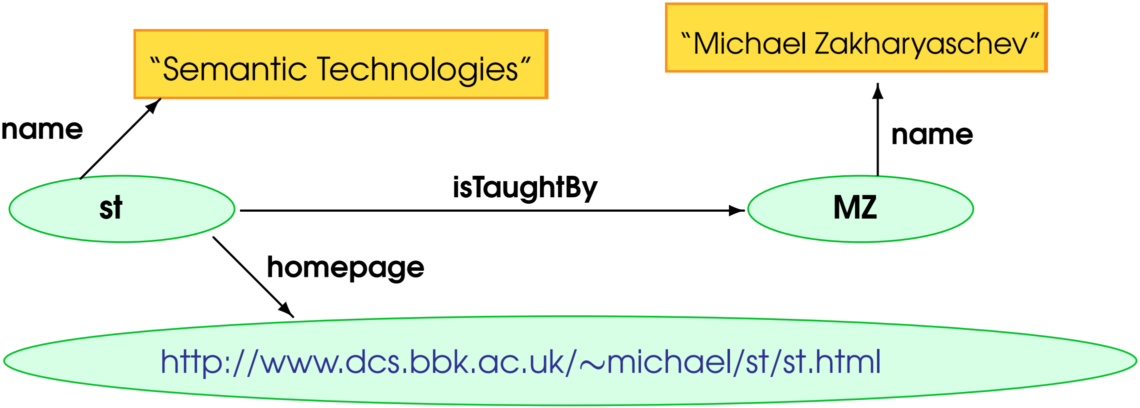
- Not safely processable with grep and similar tools

RDF: four different views of RDF

(1) N-Triples

⟨http://dcs.bbk.ac.uk/#st⟩ ⟨http://dcs.bbk.ac.uk/#name⟩ "Semantic Technologies" .
⟨http://dcs.bbk.ac.uk/#st⟩ ⟨http://dcs.bbk.ac.uk/#isTaughtBy⟩ ⟨http://dcs.bbk.ac.uk/#MZ⟩ .
⟨http://dcs.bbk.ac.uk/#st⟩ ⟨http://dcs.bbk.ac.uk/#homepage⟩
⟨http://www.dcs.bbk.ac.uk/∼michael/st/st.html⟩ .
⟨http://dcs.bbk.ac.uk/#MZ⟩ ⟨http://dcs.bbk.ac.uk/#name⟩ "Michael Zakharyaschev" .

(2) RDF Graph



RDF: four different views of RDF (cont.)

(3) Turtle (see `http://www.w3.org/TR/turtle/`)

```
# this is a complete turtle document
@prefix  dcs:  <http://dcs.bbk.ac.uk/#>  .
dcs:st  dcs:name  “Semantic Technologies”  .
dcs:st  dcs:isTaughtBy  dcs:MZ  .
dcs:st  dcs:homepage  <http://www.dcs.bbk.ac.uk/∼michael/st/st.html>  .
dcs:MZ  dcs:name  “Michael Zakharyaschev”  .
```

Abbreviating groups of triples with the same subject:

```
dcs:st  dcs:name  “Semantic Technologies”  ;
        dcs:isTaughtBy  dcs:MZ  .
```

Abbreviating groups of triples with the same subject & predicate:

```
dcs:MZ  dcs:teaches  “Semantic Technologies”  ,
                     “Fundamentals of Computing”  .
```

RDF: four different views of RDF (cont.)

(4) XML syntax:

An RDF document is represented by an XML element with tag **rdf:RDF**

The content of this element is a number of **descriptions**,
which use **rdf:Description** tags

```
<?xml version=″1.0″ encoding=″UTF-8″?>
<rdf:RDF  xmlns:rdf=″http://www.w3.org/1999/02/22-rdf-syntax-ns#″
           xmlns:dcs=″http://dcs.bbk.ac.uk/#″
           xml:base=″http://dcs.bbk.ac.uk/″>

    <rdf:Description  rdf:about=″#st″>
        <dcs:isTaughtBy  rdf:resource=″#MZ″/>
        <dcs:name>Semantic Technologies</dcs:name>
        <dcs:homepage  rdf:resource=″http://www.dcs.bbk.ac.uk/∼michael/st/″/>
    </rdf:Description>

    <rdf:Description  rdf:about=″#MZ″>
        <dcs:name>Michael Zakharyaschev</dcs:name>
    </rdf:Description>
</rdf:RDF>
```

NB. The order of descriptions or resources is not important!

RDF: XML-based syntax

- An RDF document consists of an **rdf:RDF** element
  the content of which is a number of descriptions
- The content of **rdf:Description** elements are called **property elements**

```
<rdf:Description rdf:about="#st">
    <dcs:isTaughtBy>Michael Zakharyaschev</dcs:isTaughtBy>
    <dcs:name>Semantic Technologies</dcs:name>
</rdf:Description>
```

} both define property-value pairs

- Resource references: **rdf:resource** and **rdf:ID**

```
<rdf:Description rdf:ID="st">
    <dcs:isTaughtBy rdf:resource="#MZ"/>
    <dcs:name>Semantic Technologies</dcs:name>
</rdf:Description>

<rdf:Description rdf:ID="MZ">
    <dcs:name>Michael Zakharyaschev</dcs:name>
</rdf:Description>
```

NB: **rdf:ID** **defines** the resource (#MZ is appended to the base URI)

For details see `http://www.ibm.com/developerworks/library/x-tiprdfai/`

RDF: XML-based syntax (cont.)

- Nested descriptions

```
<rdf:Description rdf:ID="st">
    <dcs:name>Semantic Technologies</dcs:name>
    <dcs:isTaughtBy>
        <rdf:Description rdf:ID="MZ">
            <dcs:name>Michael Zakharyaschev</dcs:name>
        </rdf:Description>
    </dcs:isTaughtBy>
</rdf:Description>
```

- The **rdf:type** element allows us to classify resources according to their **types**

```
<rdf:Description rdf:ID="st">
    <rdf:type rdf:resource="#module"/>           ←———   st is of type module
    <dcs:name>Semantic Technologies</dcs:name>
    <dcs:isTaughtBy rdf:resource="#MZ"/>
</rdf:Description>

<rdf:Description rdf:ID="MZ">
    <rdf:type rdf:resource="#lecturer"/>         ←———   MZ is of type lecturer
    <dcs:name>Michael Zakharyaschev</dcs:name>
</rdf:Description>
```

The same resource may be of different types; e.g., MZ can also be a **taxpayer**

RDF: abbreviated syntax

- childless property elements may be replaced by XML attributes;
- for description elements with a typing element we can use the name specified in **rdf:type** instead of **rdf:Description**

Example: (all three are equivalent)

(i)

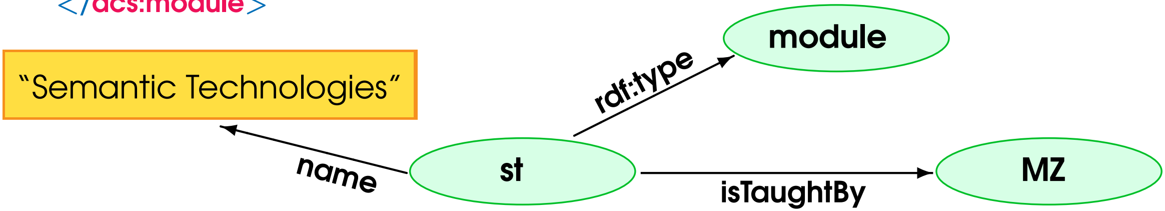
```
<rdf:Description  rdf:ID=″st″>
    <rdf:type  rdf:resource=″#module″/>
    <dcs:name >Semantic Technologies</dcs:name>
    <dcs:isTaughtBy  rdf:resource=″#MZ″/>
</rdf:Description>
```

(ii)

```
<rdf:Description  rdf:ID=″st″   dcs:name=″Semantic Technologies″>
    <rdf:type  rdf:resource=″#module″/>
    <dcs:isTaughtBy  rdf:resource=″#MZ″/>
</rdf:Description>
```

(iii)

```
<dcs:module  rdf:ID=″st″   dcs:name=″Semantic Technologies″>
    <dcs:isTaughtBy  rdf:resource=″#MZ″/>
</dcs:module>
```



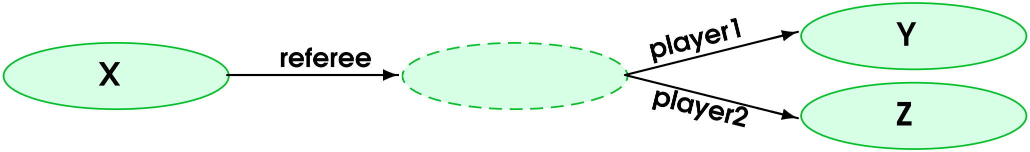
Blank nodes (for n-ary relations)

- RDF uses only **binary** relations:

  what can be done about the ternary relation

  referee(**X**, **Y**, **Z**): '**X** is a **referee** in a chess game between **Y** and **Z**'?

  Solution: introduce an **anonymous auxiliary resource**



```
<rdf:Description  rdf:ID="X">
    <referee>
        <rdf:Description>
            <player1  rdf:resource="#Y"/>
            <player2  rdf:resource="#Z"/>
        </rdf:Description>
    </referee>
</rdf:Description>
```

A **blank node** (or **bnode**) is a node in an RDF graph representing a resource for which an IRI or literal is not given. The resource represented by a blank node is called anonymous. A blank node can only be used as **subject** or **object** in an RDF triple

Exercise: give a Turtle document for the RDF graph above

Using blank nodes

Blank nodes indicate the **existence** of a thing,
without using an IRI to identify any particular thing
(not the same as assuming that the blank node indicates an `unknown' IRI)

Anonymous resources in RDF are used to:

- describe multi-component structures, like the RDF containers
- describe reification (i.e., provenance information)
- represent complex attributes without having to name explicitly auxiliary nodes
  (e.g., the address of a person consisting of the street, number, postal code, and city)
- offer protection of the inner information
  (e.g., protecting the sensitive information of the customers from the browsers)

Blank nodes: examples (1)

```
@prefix   foaf:   <http://xmlns.com/foaf/0.1/>   .

# There exist two people who know each other
_:alice   foaf:knows   _:bob   .
_:bob   foaf:knows   _:alice   .
```

blank nodes are expressed as _: followed by a label

also using [ ]

```
@prefix   foaf:   <http://xmlns.com/foaf/0.1/>   .

# Someone knows someone else, who has the name “Bob”
[ ]   foaf:knows   [ foaf:name “Bob” ]   .
```

```
@prefix   foaf:   <http://xmlns.com/foaf/0.1/>   .

[  foaf:name  “Alice”  ]  foaf:knows  [
   foaf:name  “Bob”  ;
   foaf:knows  [
      foaf:name  “Eve”  ]  ;
   foaf:mbox  <bob@example.com>  ]  .
```

```
_:a   foaf:name    “Alice”   .
_:a   foaf:knows   _:b   .
_:b   foaf:name    “Bob”   .
_:b   foaf:knows   _:c   .
_:c   foaf:name    “Eve”   .
_:b   foaf:mbox    <bob@example.com>   .
```

Blank nodes: examples (2)

```
<http://www.csd.uoc.gr/∼hy561/>  dc:title  “Web Data Management“ ;
                                  ex:professor  _:b ;
                                  ex:students  _:students ;
                                  prov:generatedBy  _:a1 .
_:b  ex:fullName  “Adam Smith” ;                                (complex attribute)
     ex:homePage  <http://www.csd.uoc.gr/∼smith/> ;
     ex:hasAddress  _:ad .
_:ad  rdf:type  ex:Address ;                                    (complex attribute)
      ex:street  “Knossou” ;
      ex:number  “122” ;
      ex:postalcode  “71409” ;
      ex:city  “Heraklion” .
_:students  rdf:type  rdf:Bag ;                                (Bag RDF container)
            dc:hasMember  _:s1 ;
            dc:hasMember  _:s2 .
_:a1  rdf:type  prov:Event ;                                (event in the lifecycle)
      prov:creator  _:b ;
      prov:atTime  “Tuesday 11 February, 06:51:00 CST”.
_:a2  rdf:type  prov:Event;
      rdf:type  prov:Update ;
      prov:ActionOver  _:a1 ;
      prov:creator  _:b ;
      prov:atTime  “Monday 17 February, 08:12:00 CST”.
```

RDF: container elements

- are used to collect a number of resources or attributes
  about which we want to make statement *as a whole*

rdf:Bag an unordered container, which may contain multiple occurrences

rdf:Seq an ordered container, which may contain multiple occurrences

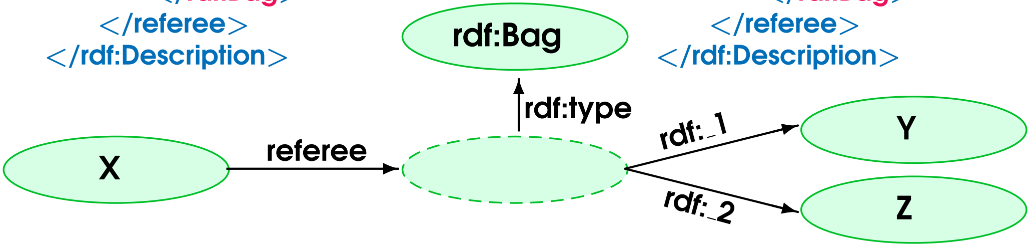
rdf:Alt a set of alternatives

- **Ex.:**

```
<rdf:Description rdf:about="#X">
   <referee>
      <rdf:Bag>
         <rdf:_1 rdf:resource="#Y"/>
         <rdf:_2 rdf:resource="#Z"/>
      </rdf:Bag>
   </referee>
</rdf:Description>
```

or

```
<rdf:Description rdf:about="#X">
   <referee>
      <rdf:Bag>
         <rdf:li rdf:resource="#Y"/>
         <rdf:li rdf:resource="#Z"/>
      </rdf:Bag>
   </referee>
</rdf:Description>
```



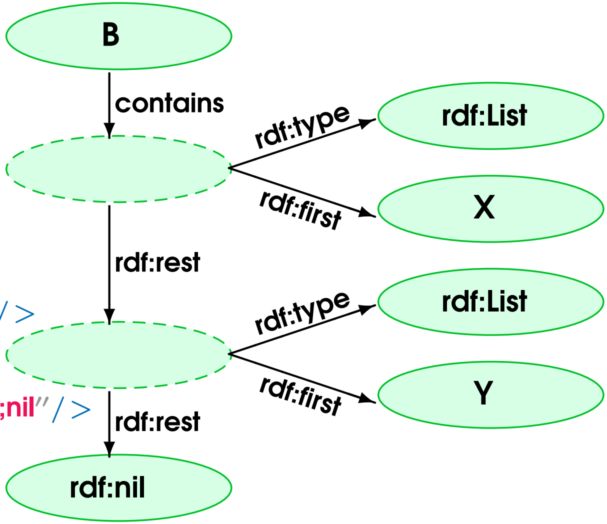
NB: Containers define anonymous auxiliary resources (see also prev. slide)

NB: There is no way to close the containers, i.e.,
to say `these are all the members of the container'

RDF: container elements (cont.)

- To describe groups containing only the specified members, use list structures

```
<rdf:Description  rdf:about="#B">
  <contains>
    <rdf:List>
      <rdf:first>
        <rdf:Description  rdf:about="#X"/>
      </rdf:first>
      <rdf:rest>
        <rdf:List>
          <rdf:first>
            <rdf:Description  rdf:about="#Y"/>
          </rdf:first>
          <rdf:rest>
            <rdf:Description  rdf:about="&rdf;nil"/>
          </rdf:rest>
        </rdf:List>
      </rdf:rest>
    </rdf:List>
  </contains>
</rdf:Description>
```

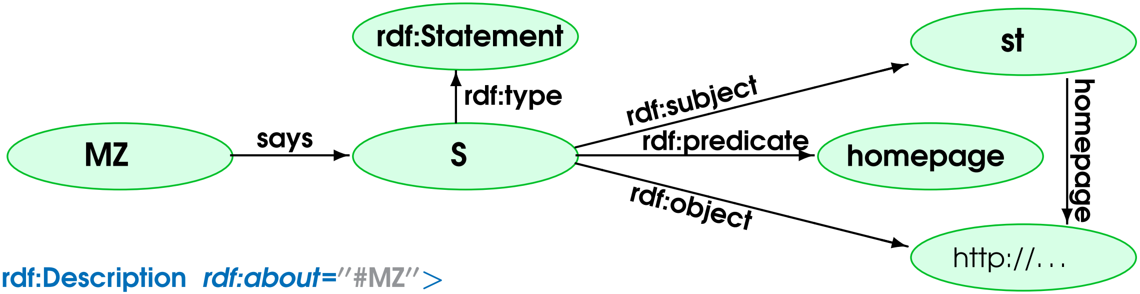


OR

```
<rdf:Description  rdf:about="#B">
    <contains  rdf:parseType="Collection">
        <rdf:Description  rdf:about="#X"/>
        <rdf:Description  rdf:about="#Y"/>
    </contains>
</rdf:Description>
```

Reification: statements about statements

‘**MZ says** that $\underbrace{\texttt{http://www.dcs.bbk.ac.uk/\~{}michael/st}\text{ is the }\textbf{homepage}\text{ for }\textbf{st}}_{\text{statement}}$’



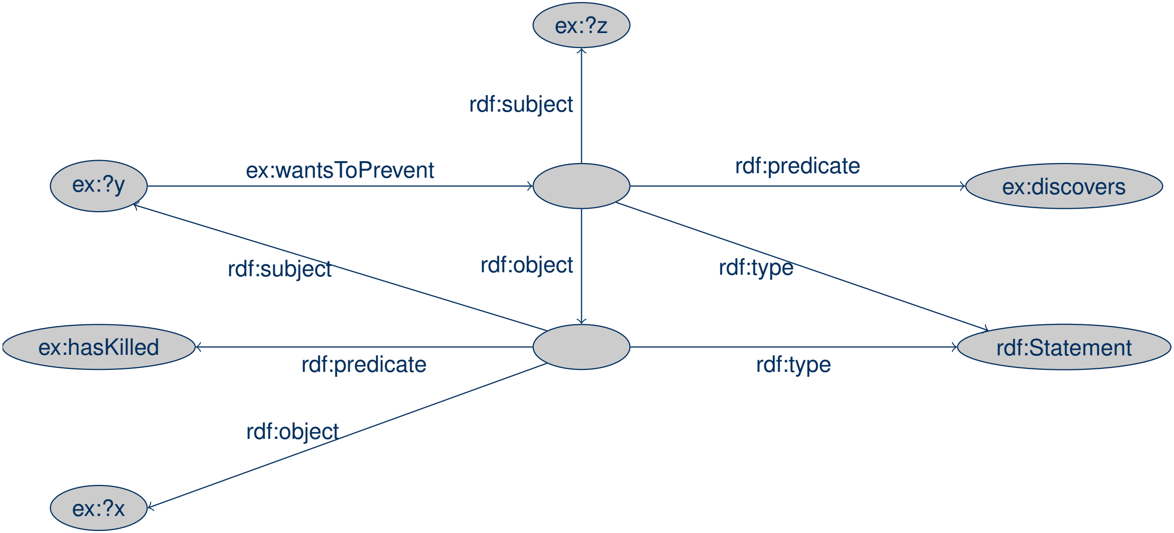
```
<rdf:Description  rdf:about="#MZ">
    <dcs:says  rdf:resource="#S"/>
</rdf:Description>

    <rdf:Description  rdf:about="#st">
        <dcs:homepage  rdf:resource="http://www.dcs.bbk.ac.uk/~michael/st"/>
    </rdf:Description>
    <rdf:Statement  rdf:ID="S">
        <rdf:subject  rdf:resource="#st"/>
        <rdf:predicate  rdf:resource="#homepage"/>
        <rdf:object  rdf:resource="http://www.dcs.bbk.ac.uk/~michael/st"/>
    </rdf:Statement>
```

Exercise: give a Turtle document for the RDF graph above

Criminal story

Translate into English



RDF: a quick summary

RDF 1.1 Turtle: official W3C document

- Example 1
- Note in 2.4
- Example 9 and Note after it
- Examples 11, 12, 14, 15

RDF and RDF Schema

Consider the RDF document (rewrite it in the Turtle syntax)

```
<dcs:academicStaff rdf:ID="PTW" dcs:name="Peter Wood"/>
<dcs:professor rdf:ID="MZ" dcs:name="Michael Zakharyaschev"/>
<dcs:module rdf:ID="st" dcs:name="Semantic Technologies">
    <dcs:isTaughtBy rdf:resource="#MZ"/>
</dcs:module>
```

How can we collect all **academicStaff** members?

What about

— professors are academic staff members

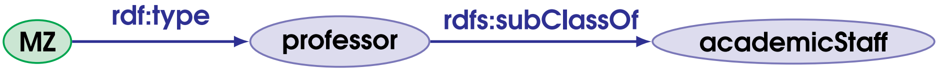
— modules are taught by academic staff members only **?**

We need statements not only about **individual objects** (such as MZ, PTW, st)
but also about **classes** of objects (such as professors, academic staff, etc.)

→ RDF Schema

What do we need?

How about the following triples?



(1) MZ is in relation **rdf:type** with **professor**

(2) professor is in relation **rdfs:subClassOf** with **academicStaff**

However, **rdf:type** and **subClassOf** are not ordinary properties.

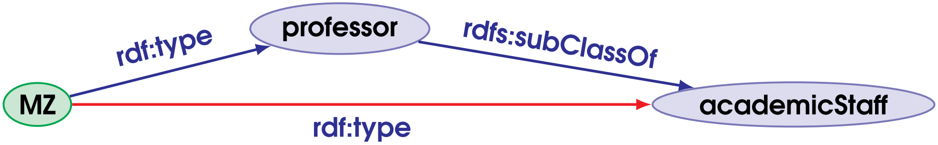
What we actually want to say is that

(1′) MZ is **an instance** of the class **professor**

(2′) every instance of **professor** is also **an instance** of **academicStaff**

In particular, **(1)** and **(2)** should **imply** that

(3) MZ is in relation **rdf:type** with **academicStaff**

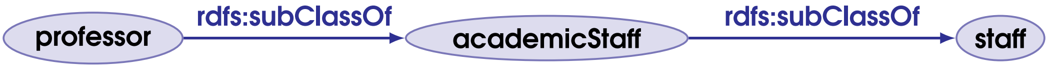


RDF Schema

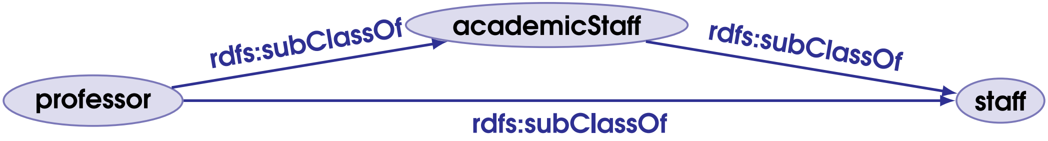
- RDF Schema defines a **‘schema vocabulary’** that supports definition of **ontologies**
  - gives ‘extra meaning’ to particular RDF predicates and resources (such as **type**, **subClassOf**)
  - this ‘extra meaning’, or **semantics**, specifies how a term should be interpreted

RDF Schema fixes the semantics

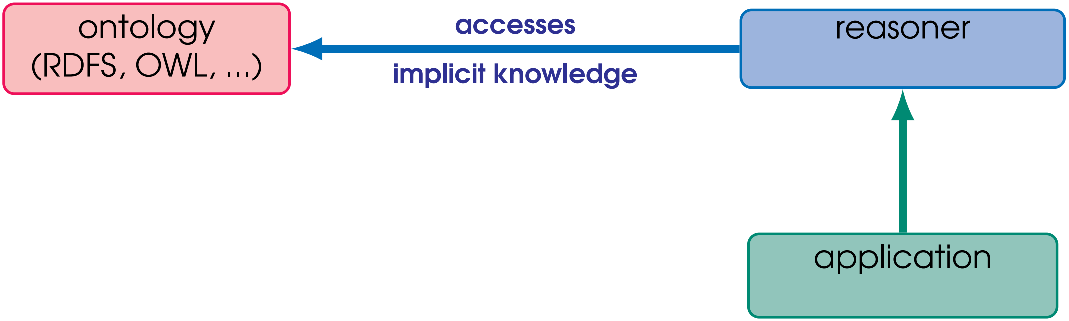
Example. Is there any **implicit knowledge** in the RDF graph?



It should **follow** that every instance of **professor** is also an instance of **staff**



Reasoner to extract implicit knowledge



The RDFS reasoner uses **entailment rules** such as

- if such and such triples are in the RDFS graph, then
  add this and that triples to the graph
- do that recursively until the graph does not change

(≈ 30 rules are specified in the RDFS Semantics document)

These rules are supposed to capture the intended semantics

RDF Schema: the language

- Core classes (predefined in RDF/S and carrying a fixed meaning)
  - **rdfs:Resource**, the class of all resources (everything is a resource)
  - **rdfs:Class**, the class of all classes (and so it contains itself as an element)

    rdfs:Class rdf:type rdfs:Class . is always **valid**
  - **rdfs:Literal**, the class of all literals (strings)
  - **rdf:Property**, the class of all properties
  - **rdf:Statement**, the class of all statements

Example:

```
<rdfs:Class rdf:ID="lecturer">
   . . .
</rdfs:Class>

<rdf:Property rdf:ID="isTaughtBy">
   . . .
</rdf:Property>
```

Thus, `lecturer' is a class and `isTaughtBy' is a property

RDF Schema: the language

- Core properties for defining relationships
  - **rdf:type**, which relates resource to its class (see slide 30)
    (the resource is declared to be an **instance** of that class)
  - **rdfs:subClassOf**, which relates a class to one of its superclasses
    all instances of a class are instances of its superclass
    (note that a class may be a subclass of more than one class)
  - **rdfs:subPropertyOf**, which relates a property to one of its superproperties
    P is a subproperty of Q if $Q(x, y)$ whenever $P(x, y)$

Example:

```
<rdfs:Class rdf:about="#lecturer">
    <rdfs:subClassOf rdf:resource="#academicStaff"/>
</rdfs:Class>

<rdf:Property rdf:about="#isTaughtBy">
    <rdfs:subPropertyOf rdf:resource="#involves"/>
</rdf:Property>
```

Thus, 'lecturer' is a subclass of 'academicStaff' and 'isTaughtBy' a subproperty of 'involves'

RDF Schema: the language

- Core properties for restricting properties
  - **rdfs:domain**, which specifies the **domain** of a property,
    (the class of those resources that may appear as **subjects** in a triple with that predicate)
  - **rdfs:range**, which specifies the **range** of a property,
    (the class of those resources that may appear as **objects** in a triple with that predicate)

Example:

```
<rdf:Property rdf:ID="isTaughtBy">
    <rdfs:domain rdf:resource="#module"/>
    <rdfs:range rdf:resource="#academicStaff"/>
</rdf:Property>
```

Thus, the domain of ‘isTaughtBy’ is a subset of ‘module’ and the range of ‘isTaughtBy’ is a subset of ‘academicStaff’

Impact of domain and range restrictions

Example:

```
<rdf:Property rdf:ID=″authourOf″>
    <rdfs:range rdf:resource=″#textBook″/ >
    <rdfs:range rdf:resource=″#crimeFiction″/ >
</rdf:Property>
```

According to the RDFS semantics, for every triple $\langle x, \textbf{authorOf}, y\rangle$,
object y is **both** a textbook **and** a crime fiction book

because the range of **authourOf** belongs to **textBook** and also to **crimeFiction**

Example: suppose

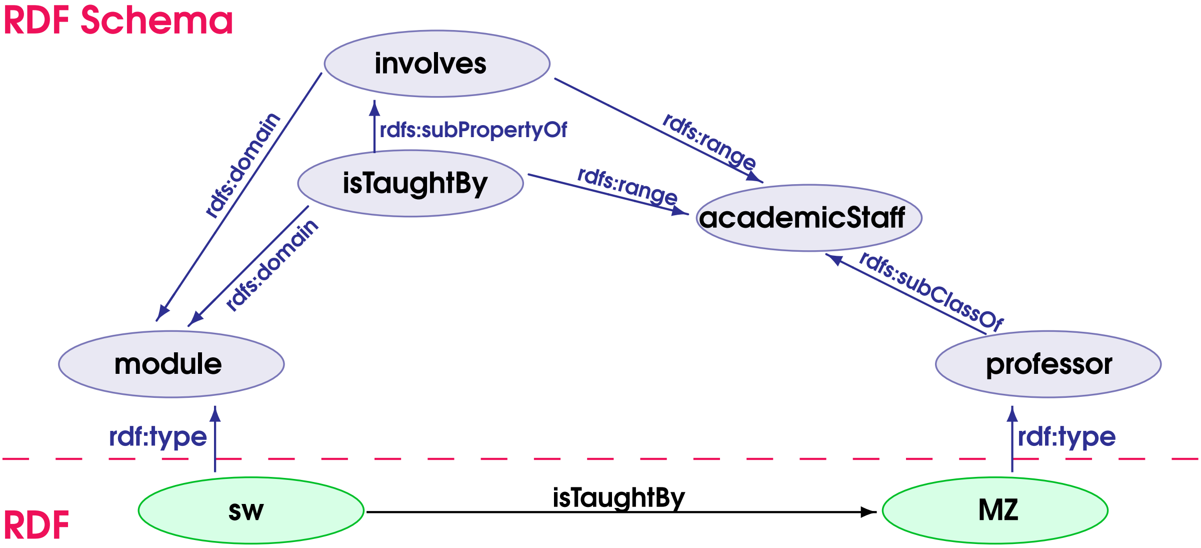
```
<rdf:Property rdf:ID=″isMarriedTo″>
    <rdfs:domain rdf:resource=″#person″/ >
    <rdfs:range rdf:resource=″#person″/ >
</rdf:Property>
```

is merged with

```
<rdf:Description rdf:about=″#ML″>
        <dcs:isMarriedTo rdf:resource=″#BBK″/ >
</rdf:Description>
<rdf:Description rdf:about=″#BBK″>
        <rdf:type rdf:resource=″#college″/ >
</rdf:Description>
```

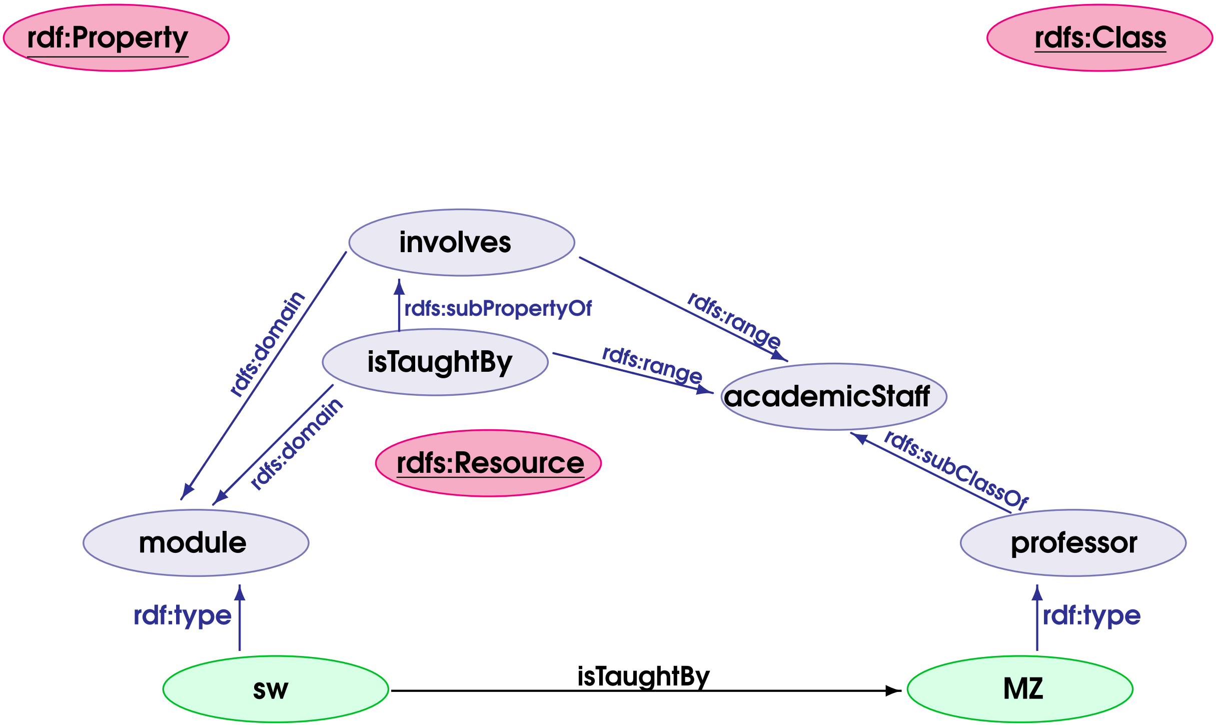
Then, according to the RDFS semantics, **BBK** must be of type **person**

RDF vs. RDFS layers



What `implicit knowledge´ is missing in the picture?

RDFS Semantics: example



RDFS Semantics: rdfs:subClassOf and rdfs:Class

- If $\langle x$, rdf:type, $C\rangle$ then $\langle C$, rdf:type, rdfs:Class$\rangle$

  'If C is a type of something then C is a class'

- If $\langle C$, rdf:type, rdfs:Class$\rangle$ then $\langle C$, rdfs:subClassOf, rdfs:Resource$\rangle$

  'Every class is a subclass of rdfs:Resource'

- If $\langle C_1$, rdfs:subClassOf, $C_2\rangle$ then

  $\langle C_1$, rdf:type, rdfs:Class$\rangle$ and $\langle C_2$, rdf:type, rdfs:Class$\rangle$

  'C_1 is a subclass of C_2 then both C_1 and C_2 are classes'

- If $\langle C$, rdf:type, rdfs:Class$\rangle$ then $\langle C$, rdfs:subClassOf, $C\rangle$

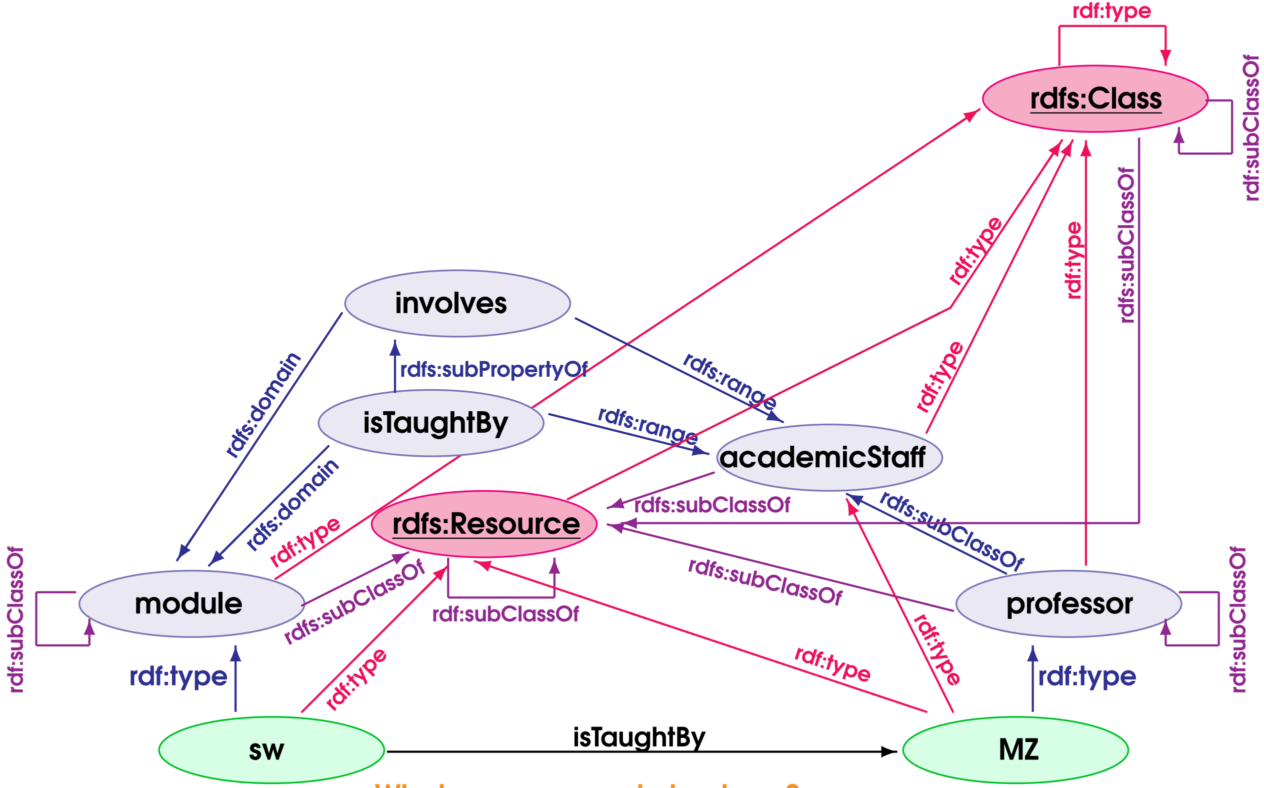
  'rdfs:subClassOf is a reflexive relation on classes'

- If $\langle C_1$, rdfs:subClassOf, $C_2\rangle$ and $\langle C_2$, rdfs:subClassOf, $C_3\rangle$ then

  $\langle C_1$, rdfs:subClassOf, $C_3\rangle$

  'rdfs:subClassOf is a transitive relation'

RDFS Semantics: rdfs:subClassOf and rdfs:Class (cont.)



What arrows are missing here?

RDFS Semantics: rdfs:subClassOf and rdfs:Class (cont.)

Summary of subclass and membership relations:

- **professor** is a subclass of **professor**, **academicStaff**, **rdfs:Resource**
- **academicStaff** is a subclass of **academicStaff**, **rdfs:Resource**
- **module** is a subclass of **module**, **rdfs:Resource**
- **rdfs:Class** is a subclass of **rdfs:Class**, **rdfs:Resource**
- **rdfs:Resource** is a subclass of **rdfs:Resource**

rdfs:subClassOf

means

'$\subseteq$'

- **MZ** is an instance of **professor**, **academicStaff**, **rdfs:Resource**
- **sw** is an instance of **module**, **rdfs:Resource**
- **professor** is an instance of **rdfs:Class**, **rdfs:Resource**
- **academicStaff** is an instance of **rdfs:Class**, **rdfs:Resource**
- **module** is an instance of **rdfs:Class**, **rdfs:Resource**
- rdfs:Class is an instance of rdfs:Class, **rdfs:Resource**
- rdfs:Resource is an instance of **rdfs:Class**, rdfs:Resource

rdf:type

means

'$\in$'

(almost)

(Class $\in$ Class?)

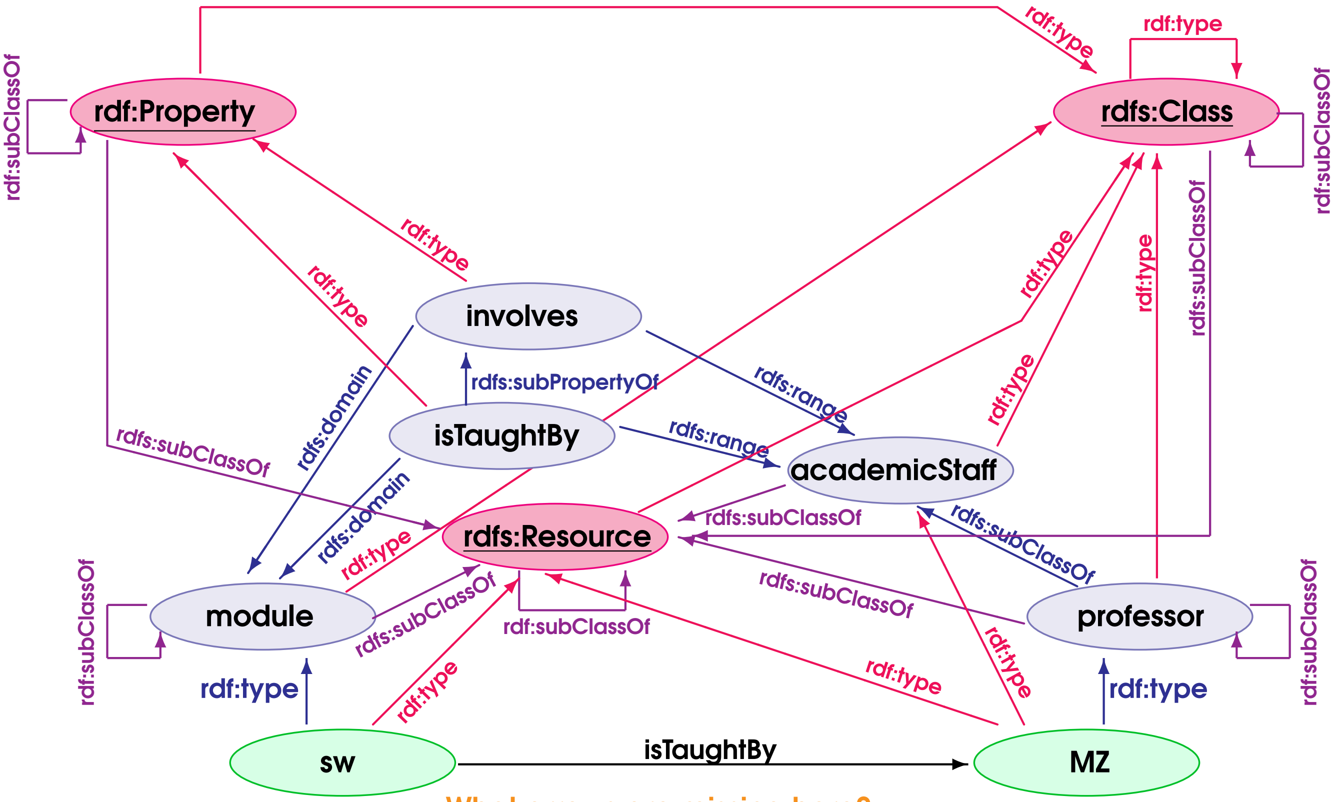
RDFS Semantics: rdfs:subPropertyOf and rdf:Property

- P is a property if and only if $\langle P$, rdf:type, rdf:Property$\rangle$
- If $\langle P_1$, rdfs:subPropertyOf, $P_2\rangle$ then
  $\langle P_1$, rdf:type, rdf:Property$\rangle$ and $\langle P_2$, rdf:type, rdf:Property$\rangle$
- If $\langle P$, rdf:type, rdf:Property$\rangle$ then $\langle P$, rdfs:subPropertyOf, $P\rangle$

  'rdfs:subPropertyOf is a reflexive relation on properties'
- If $\langle P_1$, rdfs:subPropertyOf, $P_2\rangle$ and $\langle P_2$, rdfs:subPropertyOf, $P_3\rangle$ then
  $\langle P_1$, rdfs:subPropertyOf, $P_3\rangle$

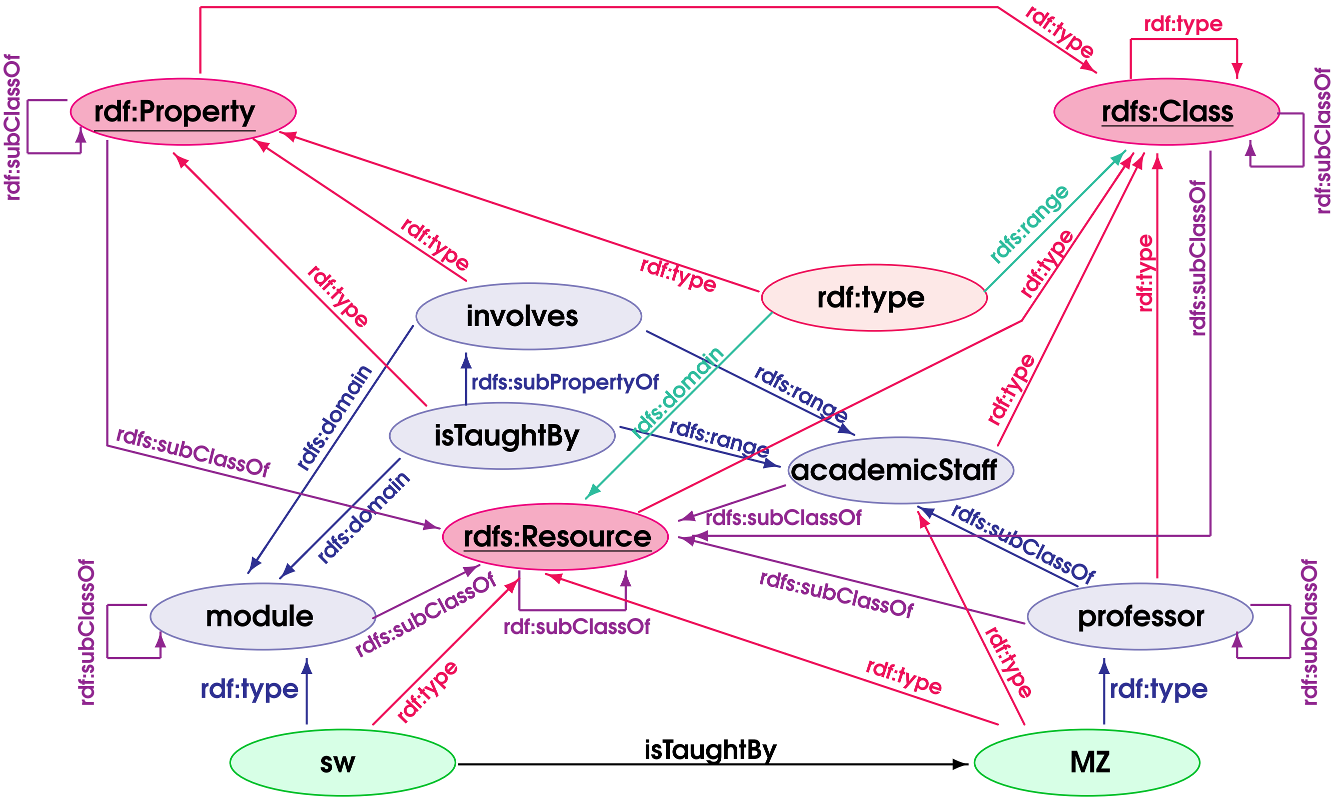
  'rdfs:subPropertyOf is a transitive relation'

RDFS Semantics: rdfs:subPropertyOf and rdf:Property (cont.)

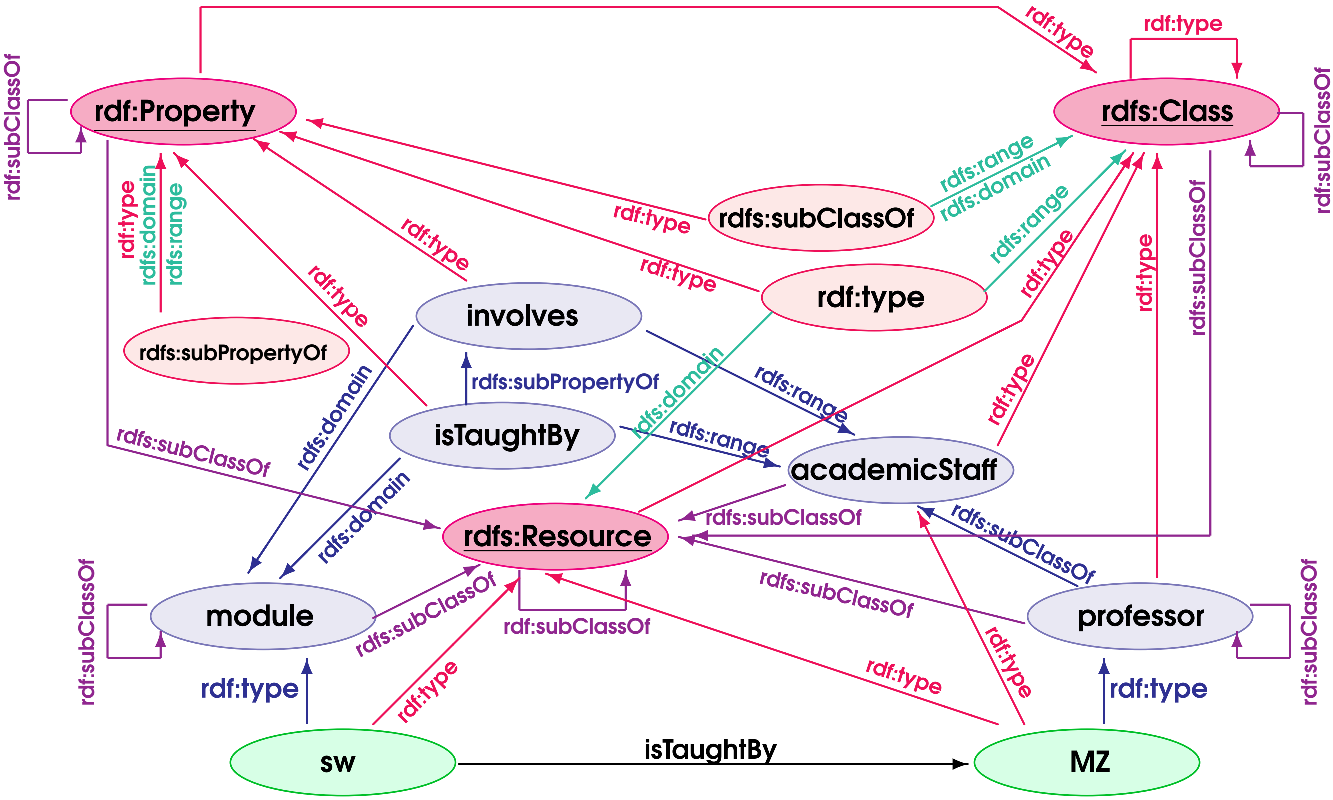


What arrows are missing here?

RDFS Semantics: rdf:type is a property



RDFS Semantics: rdfs:subClassOf and rdfs:subPropertyOf



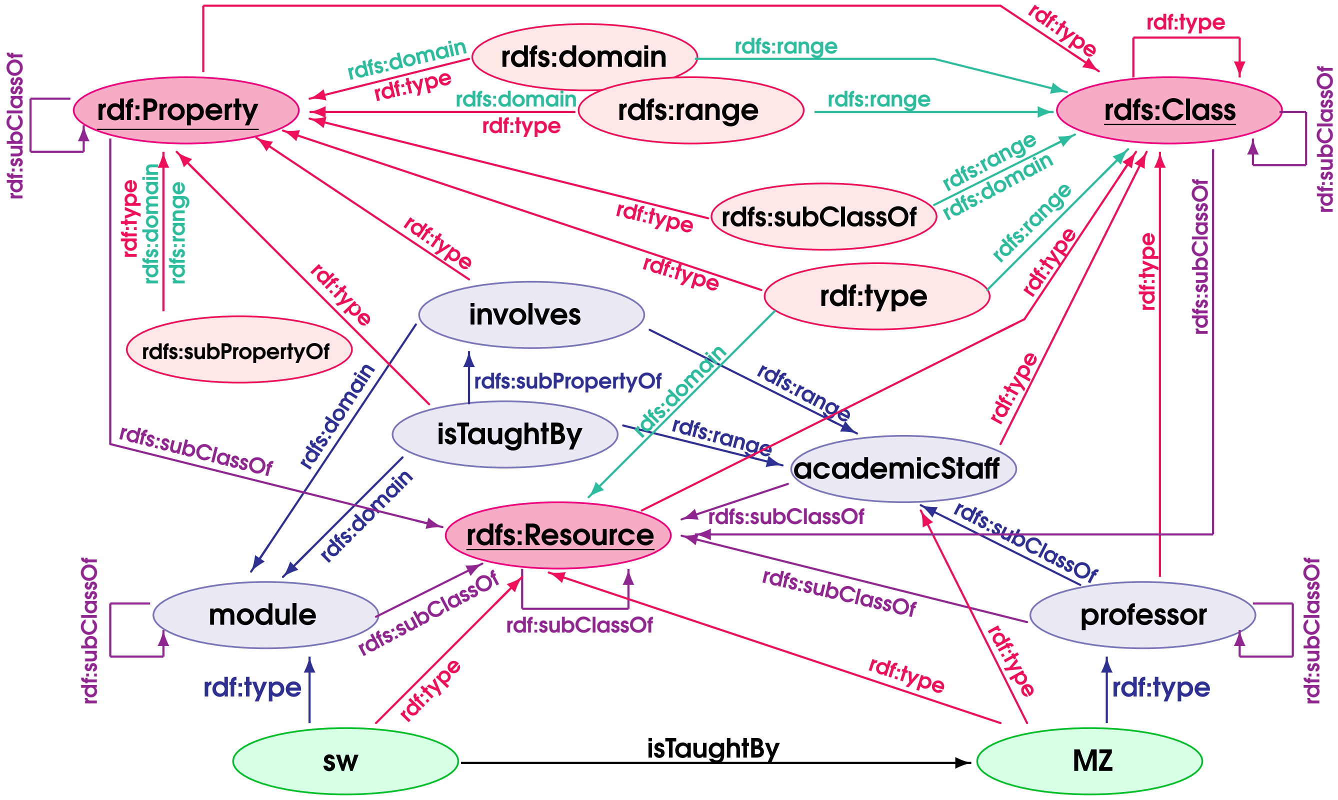
RDFS Semantics: rdfs:domain and rdfs:range

- If $\langle P,$ rdfs:domain, $C\rangle$ then $\langle C,$ rdf:type, rdfs:Class$\rangle$
- If $\langle P,$ rdfs:range, $C\rangle$ then $\langle C,$ rdf:type, rdfs:Class$\rangle$

The following are **axiomatic triples** (they must always hold)

- ⟨rdf:type, rdfs:domain, rdfs:Resource⟩
- ⟨rdfs:subClassOf, rdfs:domain, rdfs:Class⟩
- ⟨rdfs:subPropertyOf, rdfs:domain, rdf:Property⟩
- ⟨rdfs:domain, rdfs:domain, rdf:Property⟩
- ⟨rdfs:range, rdfs:domain, rdf:Property⟩
- . . .
- ⟨rdf:type, rdfs:range, rdfs:Class⟩
- ⟨rdfs:subClassOf, rdfs:domain, rdfs:Class⟩
- ⟨rdfs:subPropertyOf, rdfs:domain, rdf:Property⟩
- ⟨rdfs:domain, rdfs:domain, rdfs:Class⟩
- ⟨rdfs:range, rdfs:domain, rdfs:Class⟩
- . . .

RDFS Semantics: rdfs:domain and rdfs:range (cont.)



RDFS Semantics: not covered

- datatypes and literals:

  rdf:XMLLiteral, **rdfs:Literal** and **rdfs:Datatype**

- containers:

  rdfs:member, **rdfs:Container**, **rdfs:ContainerMembershipProperty**, **rdf:List**, **rdf:first**, **rdf:rest** and **rdf:nil**

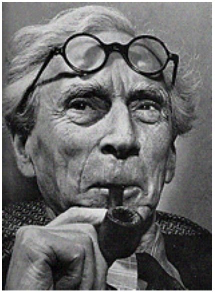
- various:

  rdfs:comment, **rdfs:seeAlso**, **rdfs:isDefinedBy** and **rdfs:label**

more information at `http://www.w3.org/TR/rdf-mt/`

Beware of Russell's paradox (1901)

Russell's paradox shows that the 'object' $\{x \mid P(x)\}$ is not always meaningful.



Consider the set $A = \{B \mid B \notin B\}$ (cf. page 53)

Give an example of an element of A

Problem: do we have $A \in A$?

For every set C, denote by $P(C)$ the statement $C \notin C$

Then $A = \{B \mid P(B)\}$.

- Suppose $A \in A$. Then not $P(A)$. Therefore, we must have $A \notin A$.
- But if $A \notin A$, then $P(A)$. Therefore, $A \in A$, which is a contradiction

Visit also `http://plato.stanford.edu/entries/russell-paradox/`

Popular version: the barber paradox

Suppose there is a town with just one male barber. According to law in this town, the barber shaves all and only those men in town who do not shave themselves.

Who shaves the barber?

- if the barber does shave himself, then the barber (himself) must not shave himself
- if the barber does not shave himself, then the barber (himself) must shave himself

Where is RDF on the Web?

- In files:

  In some serialisation: XML/RDF, Turtle, ...

  Typically small RDF graphs, i.e., max. a few 100 triples, e.g.,

  vocabularies: `http://xmlns.com/foaf/spec/index.rdf`

  tiny datasets: `http://folk.uio.no/martingi/foaf.rdf`

- From SPARQL endpoints:

  Data kept in a triple store, i.e., a database (Sesame, Jena, Redland, ...)

  `http://en.wikipedia.org/wiki/Triplestore`

  RDF is served from endpoint as results of SPARQL queries

  Exposes data (in different formats)

  with endpoint frontends, e.g., `http://dbpedia.org/resource/Norway`

  by direct SPARQL query: `http://dbpedia.org/sparql`

- There are many RDFisers which convert data to RDF

  Tabular files (CSV, Excel): XLWrap

  Relational DB: D2RQ. (`http://sws.ifi.uio.no/d2rq/`)

  W3C keeps a list: `http://www.w3.org/wiki/ConverterToRdf`

Creating RDF data and vocabularies

- Designing an easy-to-use and robust namespace is non-trivial
- Naming is difficult
- Reuse existing vocabularies if possible. Don't reinvent
  consult `http://lov.okfn.org/dataset/lov`
- IRIs are also addresses, consider publishing issues when naming
- Adhere to the policies described in best practice documents:
  Best Practice Recipes for Publishing RDF Vocabularies
  `http://www.w3.org/TR/2008/NOTE-swbp-vocab-pub-20080828/`
  Cool IRIs for the Semantic Web
  `http://www.w3.org/TR/cooluris/`
- Use `http://www.example.[com|net|org]` for prototyping and documentation

For more details, consult

`http://wifo5-03.informatik.uni-mannheim.de/bizer/pub/LinkedDataTutorial/`

The RDF vocabulary

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

  needs to be declared like all others!

- Important elements in rdf:

  type links a resource to a type (can be abbreviated)

  Resource type of all resources

  Property type of all properties

Examples:

```
geo:berlin   rdf:type   rdf:Resource   .
geo:containedIn   a   rdf:Property   .
rdf:type   a   rdf:Property   .
```

Friend Of A Friend

- People, personal information, friends, see `http://www.foaf-project.org/`
- @prefix foaf: <http://xmlns.com/foaf/0.1/> .
- Important elements in foaf:

  Person a person, alive, dead, real, imaginary

  name name of a person (also **firstName**, **familyName**)

  mbox mailbox URL of a person

  knows a person knows another person

Examples:

```
<http://dcs.bbk.ac.uk/mz/foaf#me>  a  foaf:Person  ;
                                   foaf:name   “Michael Zakharyaschev”  ;
                                   foaf:mbox   <mailto:michael@dcs.bbk.ac.uk>  ;
                                   foaf:knows  <http://.../mz/foaf#me>  .
```

Dublin Core

- Metadata for documents, see `http://dublincore.org/`
- @prefix dct: <http://purl.org/dc/terms/> .

  use this instead of legacy dc:
- Important elements in dct:

  creator a document’s main author

  created the creation date

  description a natural language description

  replaces another document superseded by this one

Examples:

```
<http://dcs.bbk.ac.uk/mz/>  dct:creator   <http://.../foaf#me>  ;
                            dct:created   ”2007-08-01”  ;
                            dct:description  ”MZ’s homepage”@en  ;
                            dct:replaces  <http://my.old.homepage/>  .
```

DBLP: Computer Science bibliography

- DBLP contains computer science publications
  `http://dblp.uni-trier.de`

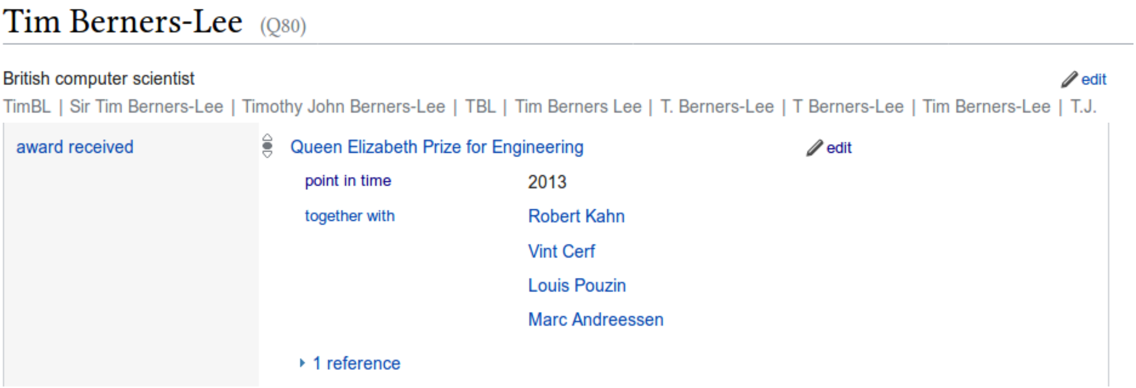
- vocabulary of the RDF version `http://dblp.l3s.de/d2r/snorql/`:

  author of a document: **dc:creator**

  title of a document: **dc:title**

  name of a person: **foaf:name**

Encoding Wikidata statements in RDF (1)



Tim Berners-Lee (Q80)

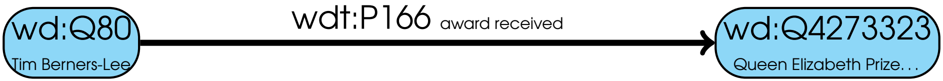
British computer scientist

TimBL | Sir Tim Berners-Lee | Timothy John Berners-Lee | TBL | Tim Berners Lee | T. Berners-Lee | T Berners-Lee | Tim Berners-Lee | T.J.

award received — Queen Elizabeth Prize for Engineering

point in time	2013
together with	Robert Kahn Vint Cerf Louis Pouzin Marc Andreessen

1 reference



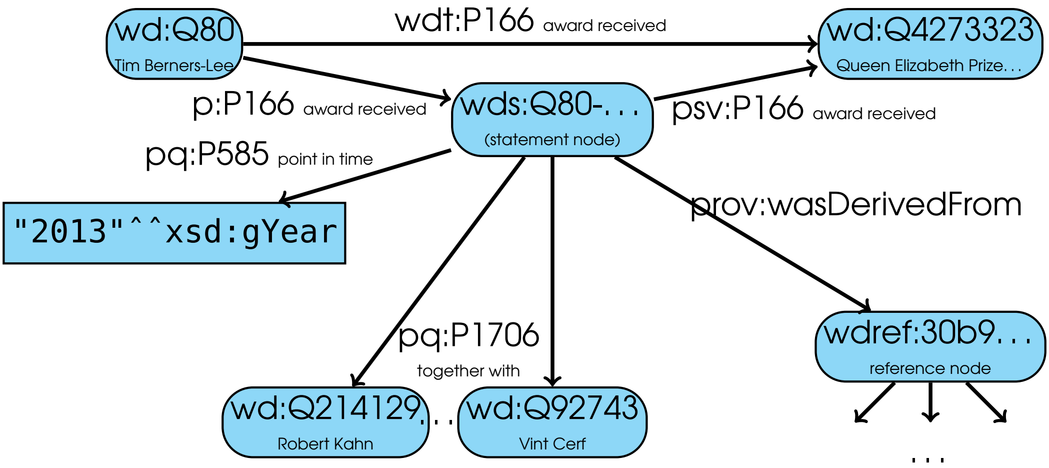
wd:Q80 Tim Berners-Lee — wdt:P166 award received → wd:Q4273323 Queen Elizabeth Prize. . .

Where to store the annotations?

Note: For prefix declarations, see `https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format`

Encoding Wikidata statements in RDF (2)

We can encode statements in the style of **reification**:



The complete Wikidata-to-RDF documentation is available online https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format

Any item can be viewed in RDF in the browser using URLs such as http://www.wikidata.org/wiki/Special:EntityData/Q80.ttl