Tutorial on the Semantic Web

Ivan Herman, W3C

(Last update: 4 May 2009)

WARNING TO THE READER!

- This is an evolving slide set. This means:
  - it changes frequently
  - there may be bugs, inconsistencies
  - it may try to reflect the latest view of technology evolution but that is often a moving target (eg, in the areas of OWL, RIF, ...)
- “Frozen” versions are instantiated for a specific presentation, and those become stable

Introduction

Towards a Semantic Web

- The current Web represents information using
  - natural language (English, Hungarian, Chinese,…)
  - graphics, multimedia, page layout
- Humans can process this easily
  - can deduce facts from partial information
  - can create mental associations
  - are used to various sensory information
    - (well, sort of… people with disabilities may have serious problems on the Web with rich media!)

Towards a Semantic Web

- Tasks often require to combine data on the Web:
  - hotel and travel information may come from different sites
  - searches in different digital libraries
  - etc.
- Again, humans combine these information easily
  - even if different terminologies are used!

However…

- However: machines are ignorant!
  - partial information is unusable
  - difficult to make sense from, e.g., an image
  - drawing analogies automatically is difficult
  - difficult to combine information automatically
    - is `<foo:creator>` same as `<bar:author>`?
  - …

Example: automatic airline reservation

- Your automatic airline reservation
  - knows about your preferences
  - builds up knowledge base using your past
  - can combine the local knowledge with remote services:
    - airline preferences
    - dietary requirements
    - calendaring
    - etc
- It communicates with remote information
  - (M. Dertouzos: The Unfinished Revolution)

Example: data(base) integration

- Databases are very different in structure, in content
- Lots of applications require managing several databases
  - after company mergers
  - combination of administrative data for e-Government
  - biochemical, genetic, pharmaceutical research
  - etc.
- Most of these data are accessible from the Web (though not necessarily public yet)

And the problem is real…

CoCoDat - Collation of Cortical Data - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://www.cocom

CoCoMac DATABASES ORT EXAMPLES

CoCoDat: Collation of Cortical [s

microcircuitry] Data

CoCoDat is a microcircuitry database that published experimental reports. The data and cellular compartment), as well as the

- Morphology
- Firing properties
- Ionic currents
- Ionic conductances
- Synaptic currents
- Connectivity

The database is available for download u data tables but also a Search Board with manual or automatic relaxation of the sea

- Brain region
- Layer
- Neuron type

http://www.cocomac.org/cocodat/catalyzer/inc

Cell Centered Database - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://ccdb.ucsd.edu/sand/main?event=gallery&action=show&dpl=y

red database

Cell Centered Database™

National Center for Microscopy and Imaging Research

Gallery

Data | Search | Gallery | Dictionary | Publications | MyCCDB | Data Download | Contact us | Help

2D image Reconstruction Segmentation Animation

y 23
0:
e 071202tjd
e:
0: P1188
ct Brain Maps
e:
ct Tom Deerinck
r:
nt BioRad RTS 2000MP
d: Multiphoton
n: montage of cerebellar cortex triple labeled for Hoescht stain (blue), IP3 receptor (green) and GFAP (red)

logged out

NeuronDB = Thalamic relay neuron - Overview (A) () - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://senselab.med.yale.edu

senselab

NeuronDB

Back

Thalamic relay neuron

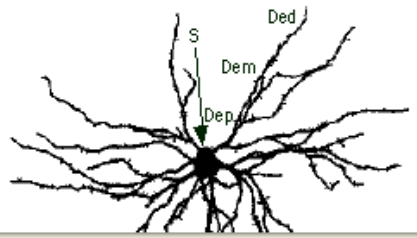
Mode: Overview | Data/Search | plus Connectivity | plus Classical References/Notes | Models

Region: Distal equivalent dendrite | Middle equivalent dendrite | Proximal equivalent dendrite | Soma | Axon hillock | Axon fiber | Axon terminal | All Compartments

Properties: Receptors | Channels | Transmitters | All Properties

Interoperation: Gene and Chromosome | Experimental Data (neurodatabase.org) | Microscopy Data (CCDB)

Neuron type: principal
Organism: Vertebrates



1. Equivalent dendrite — Show other
2. Distal equivalent dendrite — Show other
3. Middle equivalent dendrite — Show other
4. Proximal equivalent dendrite — Show other
5. Soma — Show other

Done

logged out

W3C Semantic Web

Example: social networks

- Social sites are everywhere these days (LinkedIn, Facebook, Dopplr, Digg, Plexo, Zyb, …)
- How many times did you have to add your contacts?
- Applications should be able to get to those data via standard means
  - there are, of course, privacy issues…

Example: digital libraries

- It means catalogues on the Web
  - librarians have known how to do that for centuries
  - goal is to have this on the Web, World-wide
  - extend it to multimedia data, too
- But it is more: software agents should also be librarians!
  - help you in finding the right publications

Example: semantics of Web Services

- Web services technology is great
- But if services are ubiquitous, searching issue comes up, for example:
  - “find me the best differential equation solver”
  - “check if it can be combined with the XYZ plotter service”
- It is necessary to characterize the service
  - not only in terms of input and output parameters…
  - …but also in terms of its semantics

What is needed?

- (Some) data should be available for machines for further processing
- Data should be possibly combined, merged on a Web scale
- Sometimes, data may describe other data…
- … but sometimes the data is to be exchanged by itself, like my calendar or my travel preferences
- Machines may also need to *reason* about that data

In short: we need a Web of Data!

In what follows…

- We will use a simplistic example to introduce the main Semantic Web concepts
- We take, as an example area, data integration

The rough structure of data integration

1. Map the various data onto an abstract data representation
   - make the data independent of its internal representation…
2. Merge the resulting representations
3. Start making queries on the whole!
   - queries not possible on the individual data sets

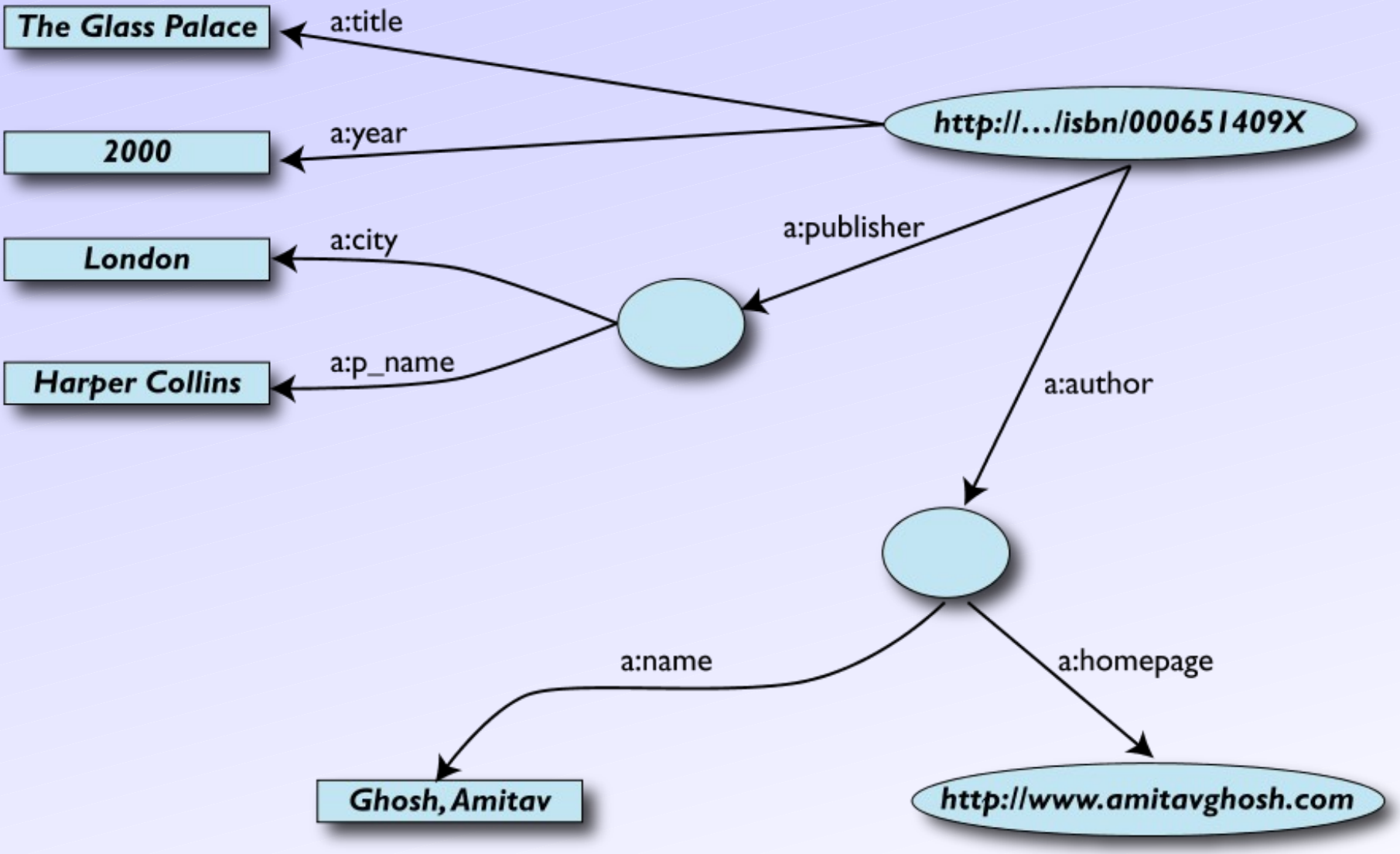
A *simplified* bookstore data (dataset "A")

ID	Author	Title	Publisher	Year
ISBN0-00-651409-X	id_xyz	The Glass Palace	id_qpr	2000

ID	Name	Home Page
id_xyz	Ghosh, Amitav	http://www.amitavghosh.com

ID	Publ. Name	City
id_qpr	Harper Collins	London

1st: export your data as a set of relations



Some notes on the exporting the data

- Relations form a graph
  - the nodes refer to the “real” data or contain some literal
  - how the graph is represented in machine is immaterial for now
- Data export does *not* necessarily mean physical conversion of the data
  - relations can be generated on-the-fly at query time
    - via SQL “bridges”
    - scraping HTML pages
    - extracting data from Excel sheets
    - etc.
- One can export *part* of the data

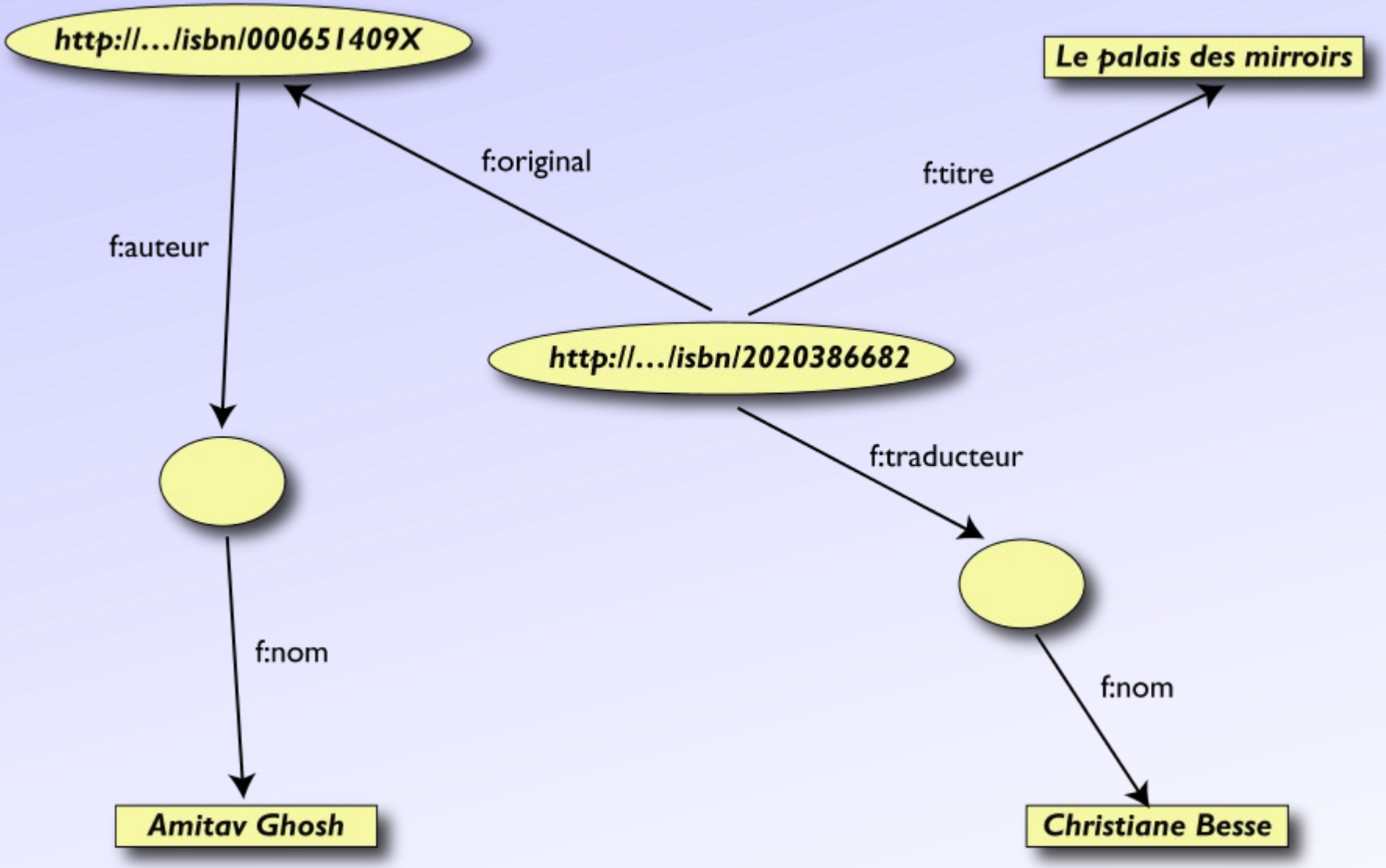
Another *bookstore data (dataset "F")*

	A	B	D	E
1	**ID**	**Titre**	**Traducteur**	**Original**
2	ISBN0 2020386682	Le Palais des miroirs	A13	ISBN-0-00-651409-X
3				

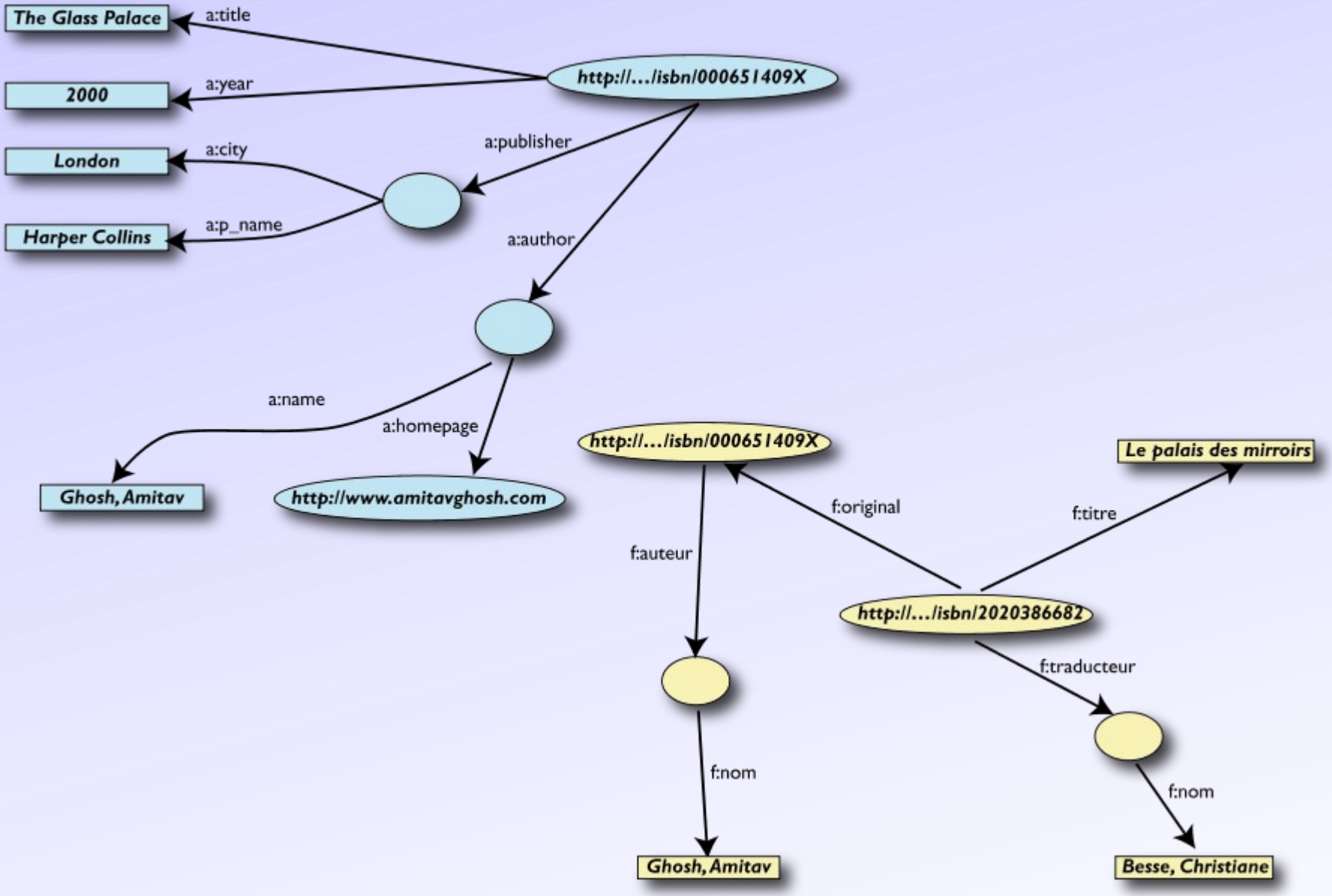
6	**ID**	**Auteur**
7	ISBN-0-00-651409-X	A12

11	**Nom**
12	Ghosh, Amitav
13	Besse, Christianne

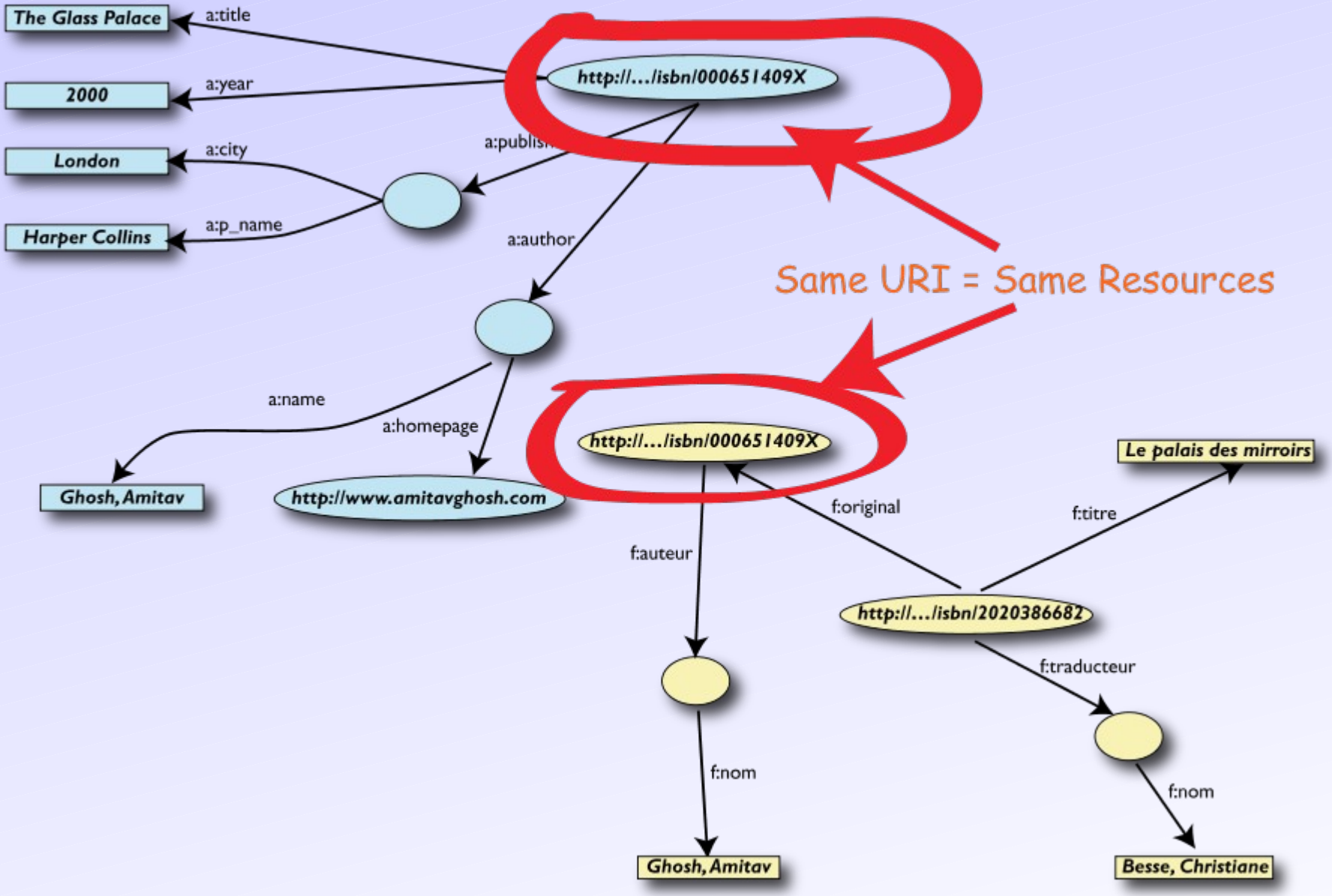
2nd: export your second set of data



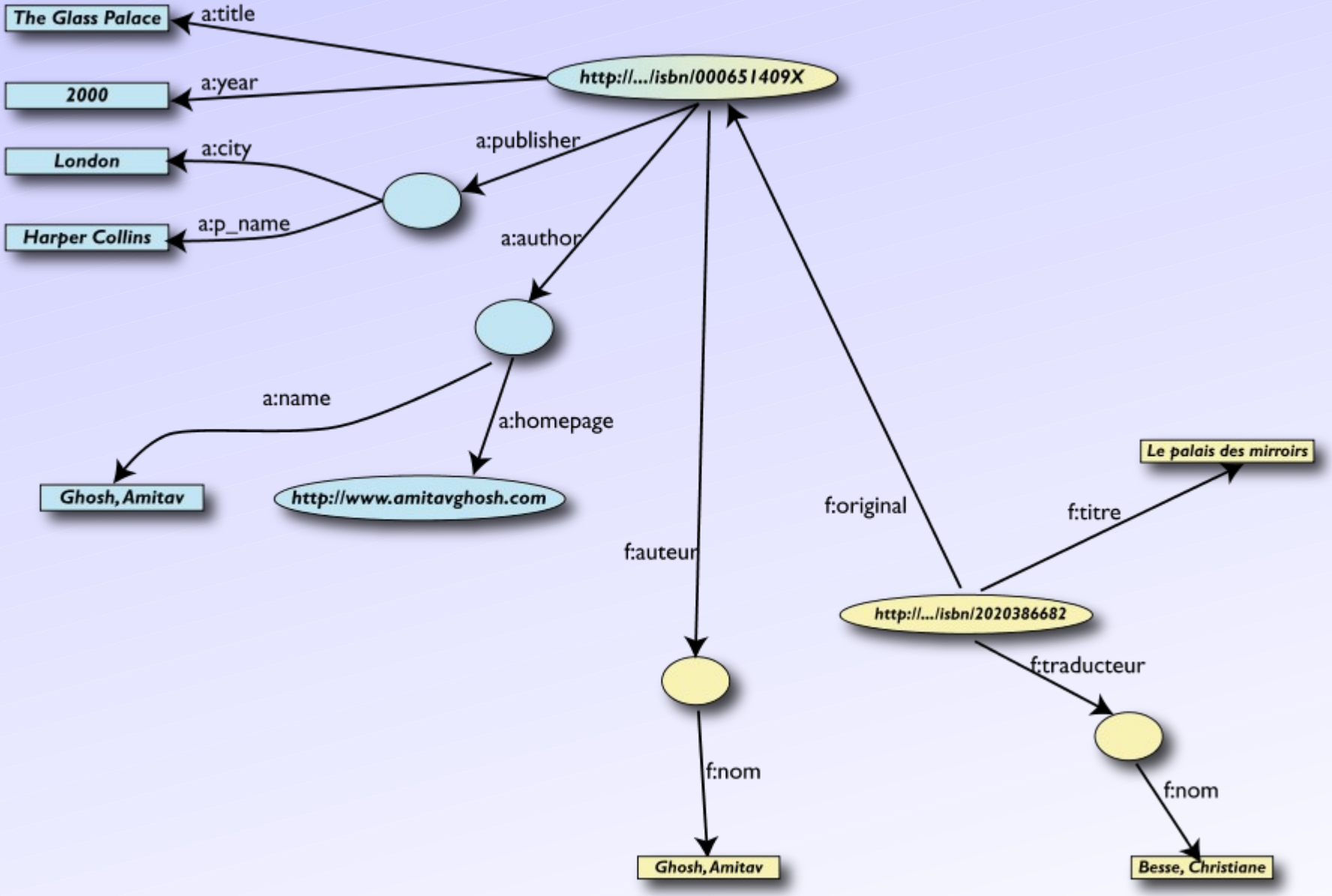
3rd: start merging your data



3rd: start merging your data (cont.)

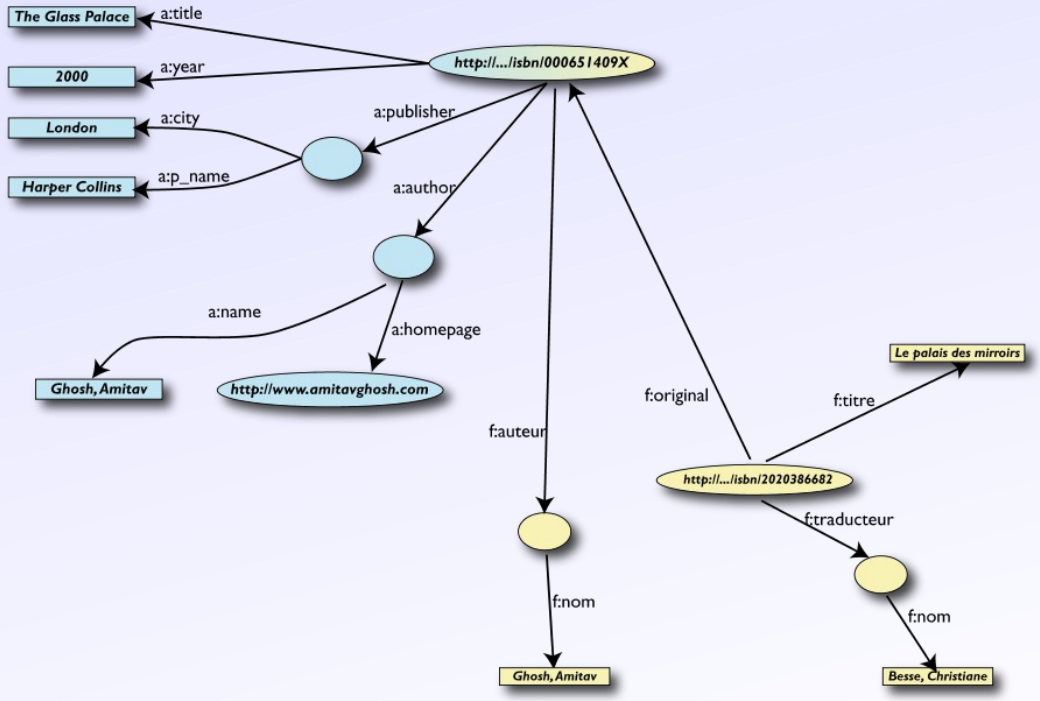


3rd: merge identical resources



Start making queries…

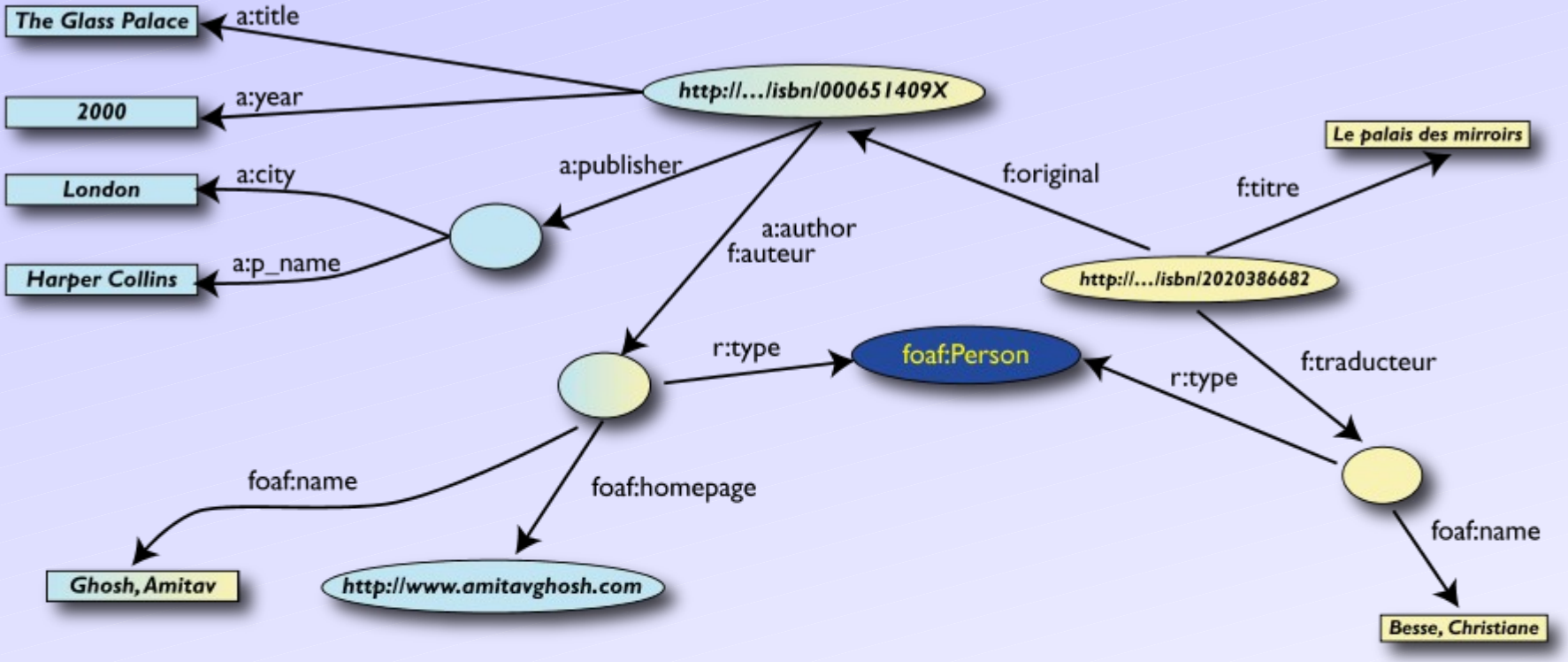
- User of data “F” can now ask queries like:
  - “give me the title of the original”
    - well, … « donnes-moi le titre de l’original »
- This information is not in the dataset “F”…
- …but can be retrieved by merging with dataset “A”!

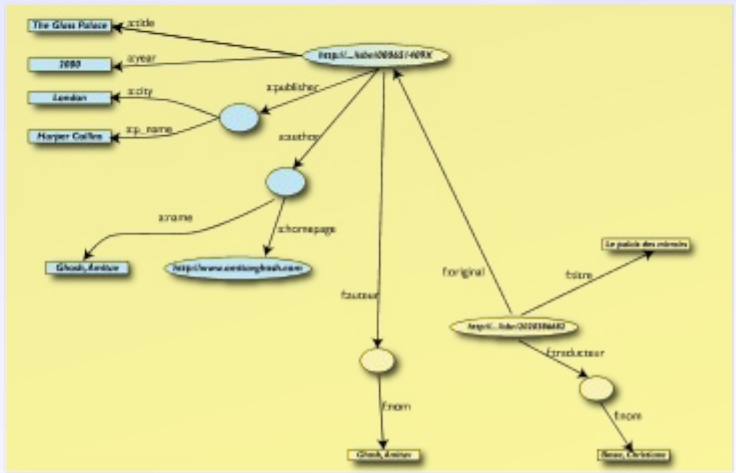


However, more can be achieved…

- We "feel" that **`a:author`** and **`f:auteur`** should be the same
- But an automatic merge doest not know that!
- Let us add some extra information to the merged data:
  - **`a:author`** same as **`f:auteur`**
  - both identify a "Person"
  - a term that a community may have already defined:
    - a "Person" is uniquely identified by his/her name and, say, homepage
    - it can be used as a "category" for certain type of resources

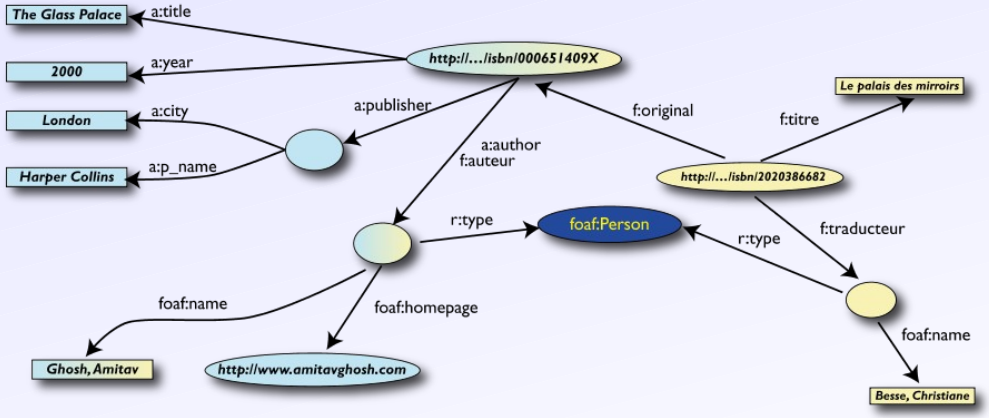
3rd revisited: use the extra knowledge





Start making richer queries!

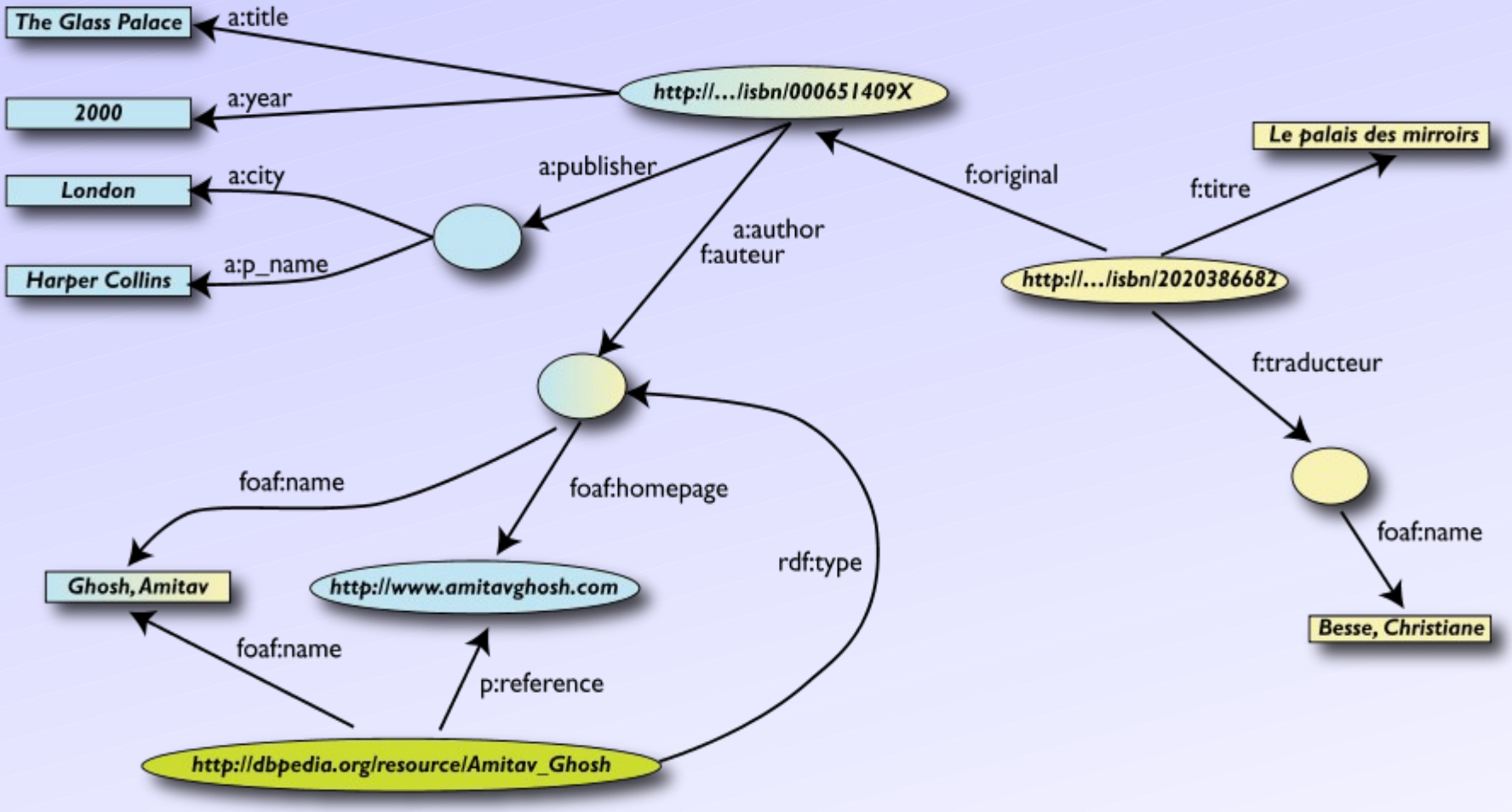
- User of dataset “F” can now query:
  - “donnes-moi la page d’accueil de l’auteur de l’originale”
    - well… “give me the home page of the original’s ‘auteur’”
- The information is not in datasets “F” or “A”…
- …but was made available by:
  - merging datasets “A” and datasets “F”
  - adding three simple extra statements as an extra “glue”



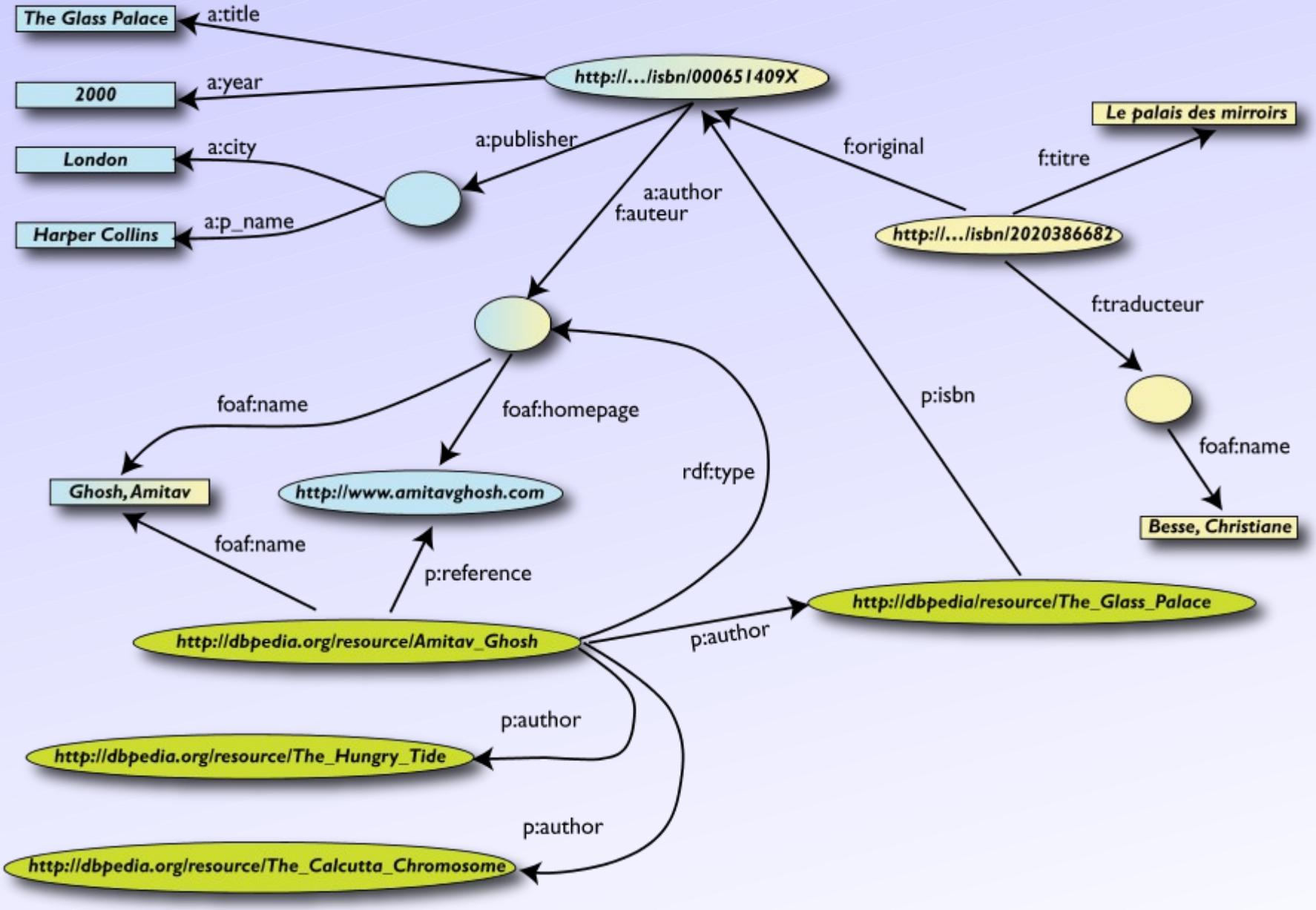
Combine with different datasets

- Using, e.g., the “Person”, the dataset can be combined with other sources
- For example, data in Wikipedia can be extracted using dedicated tools
  - e.g., the “dbpedia” project can extract the “infobox” information from Wikipedia already…

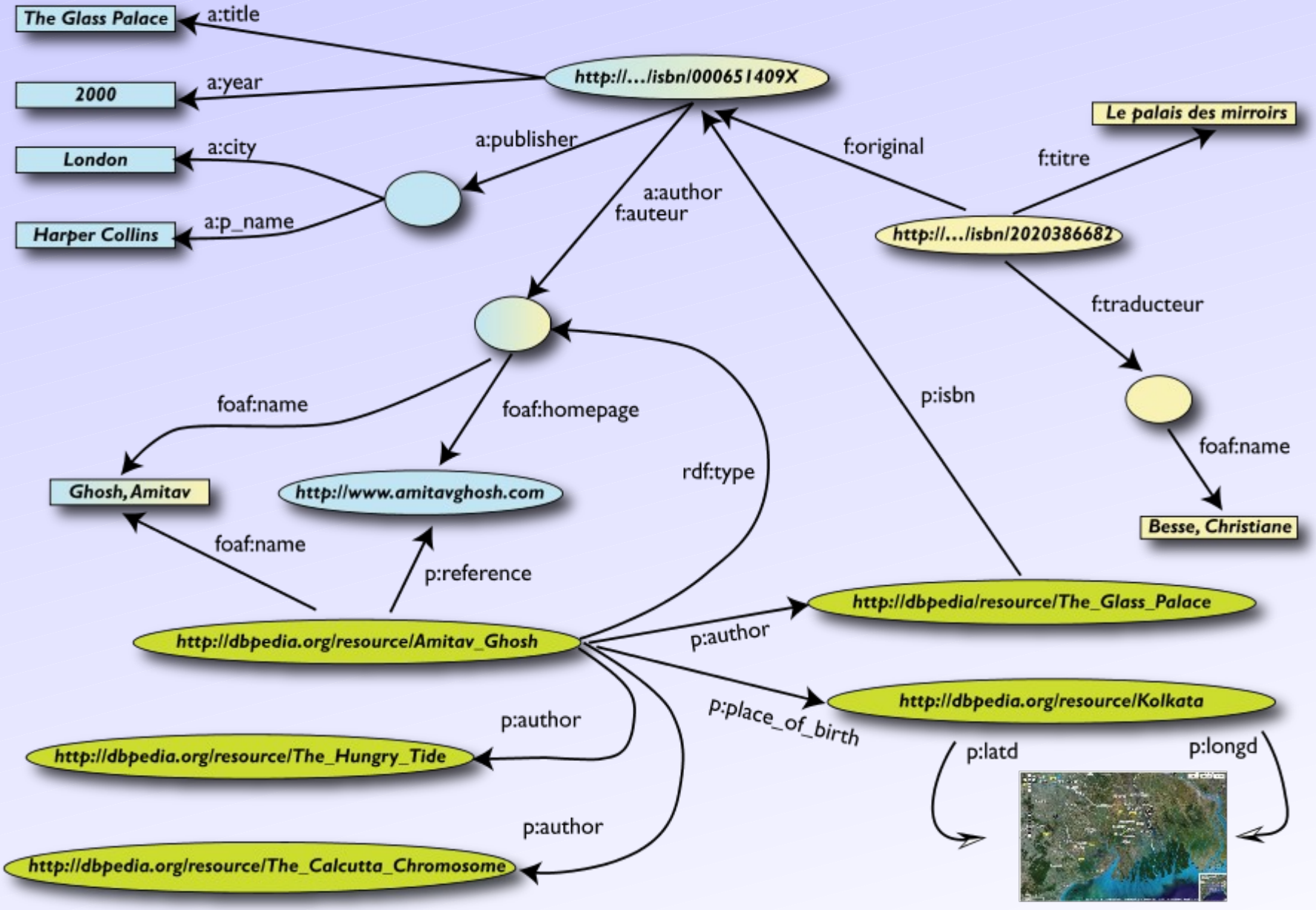
Merge with Wikipedia data



Merge with Wikipedia data



Merge with Wikipedia data



Is that surprising?

- It may look like it but, in fact, it should not be…
- What happened via automatic means is done every day by Web users!
- The difference: a bit of extra rigour so that machines could do this, too

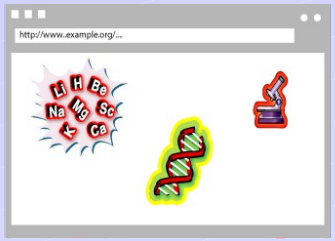
What did we do?

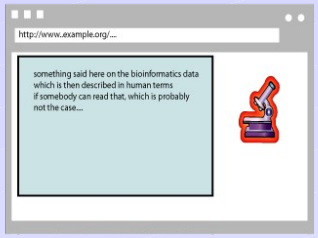
- We combined different datasets that
  - are somewhere on the web
  - are of different formats (mysql, excel sheet, XHTML, etc)
  - have different names for relations
- We could combine the data because some URI-s were identical (the ISBN-s in this case)
- We could add some simple additional information (the “glue”), also using common terminologies that a community has produced
- As a result, new relations could be found and retrieved

It could become even more powerful

- We could add extra knowledge to the merged datasets
  - e.g., a full classification of various types of library data
  - geographical information
  - etc.
- This is where *ontologies*, extra *rules*, etc, come in
  - ontologies/rule sets can be relatively simple and small, or huge, or anything in between…
- Even more powerful queries can be asked as a result

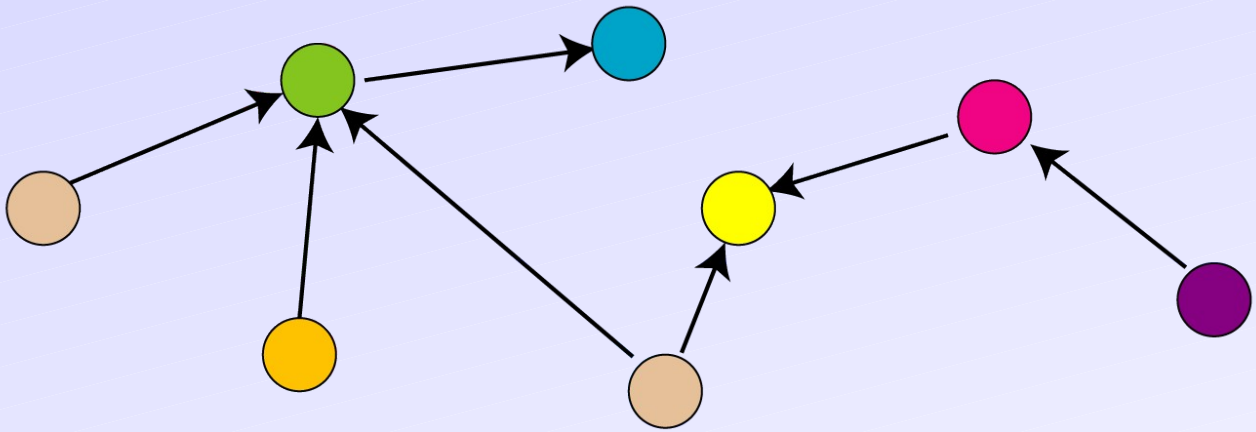
What did we do? (cont)





Applications

Query,
Manipulate,
etc.



Data represented in abstract format

Map,
Expose,
etc.

Data in various formats

The abstraction pays off because…

- … the graph representation is independent of the exact structures
- … a change in local database schema's, XHTML structures, etc, do not affect the whole
  - "schema independence"
- … new data, new connections can be added seamlessly

The network effect

- Through URI-s we can link any data to any data
- The “network effect” is extended to the (Web) data
- “Mashup on steroids” become possible

So where is the Semantic Web?

- The Semantic Web provides technologies to make such integration possible!
- Hopefully you get a full picture at the end of the tutorial…

The Basis: RDF

RDF triples

- Let us begin to formalize what we did!
  - we “connected” the data…
  - but a simple connection is not enough… data should be named somehow
  - hence the RDF Triples: *a labelled connection between two resources*

RDF triples (cont.)

- An RDF Triple **(s,p,o)** is such that:
  - "**s**", "**p**" are URI-s, ie, resources on the Web; "**o**" is a URI or a literal
    - "**s**", "**p**", and "**o**" stand for "subject", "property", and "object"
  - here is the complete triple:

```
(<http://…isbn…6682>, <http://…/original>, <http://…isbn…409X>)
```

- *RDF* is a general model for such triples (with machine readable formats like RDF/XML, Turtle, N3, RXR, …)

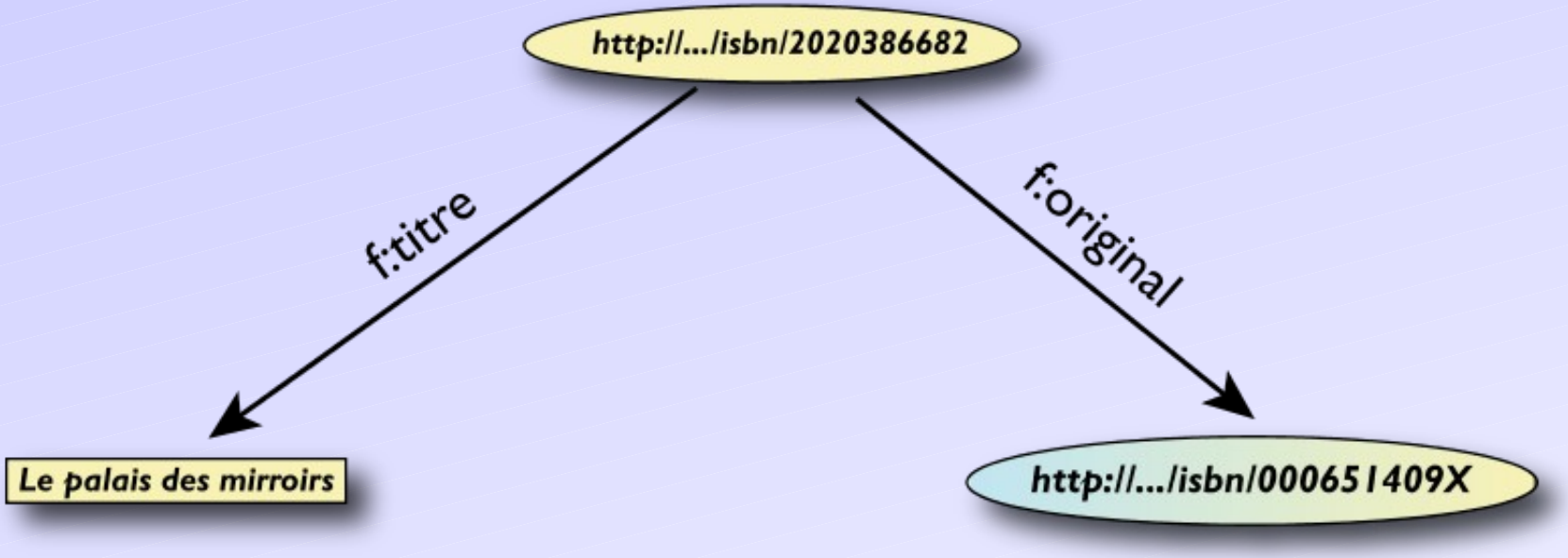
RDF triples (cont.)

- RDF triples are also referred to as “triplets”, or “statements”
- The “`p`” is also referred to as “predicate” sometimes

RDF triples (cont.)

- Resources can use *any* URI; it can denote an element within an XML file on the Web, not only a "full" resource, e.g.:
  - `http://www.example.org/file.xml#element(home)`
  - `http://www.example.org/file.html#home`
  - `http://www.example.org/file2.xml#xpath1(//q[@a=b])`
- RDF triples form a directed, labelled graph (the best way to think about them!)

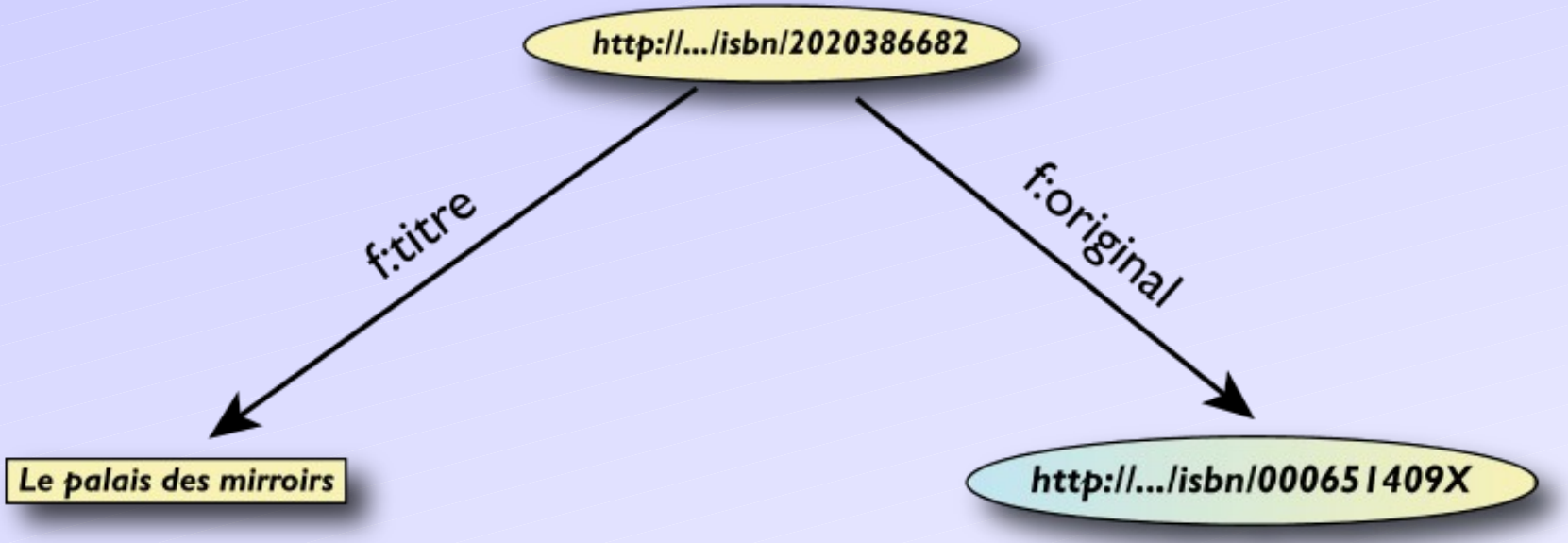
A simple RDF example (in RDF/XML)




```
<rdf:Description rdf:about="http://…/isbn/2020386682">
    <f:titre xml:lang="fr">Le palais des mirroirs</f:titre>
    <f:original rdf:resource="http://…/isbn/000651409X"/>
</rdf:Description>
```

(Note: namespaces are used to simplify the URI-s)

A simple RDF example (in Turtle)

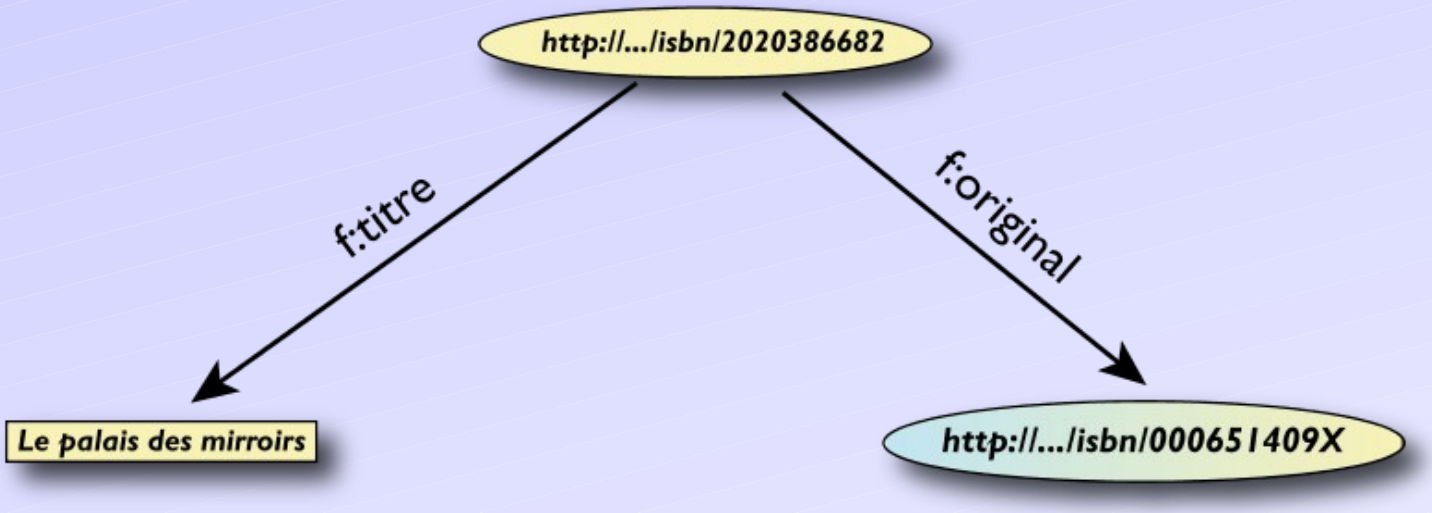



```
<http://…/isbn/2020386682>
    f:titre "Le palais des mirroirs"@fr ;
    f:original <http://…/isbn/000651409X> .
```

URI-s play a fundamental role

- URI-s made the merge possible
- *URI-s ground RDF into the Web*
  - information can be retrieved using existing tools
  - this makes the “Semantic Web”, well… “Semantic *Web*”

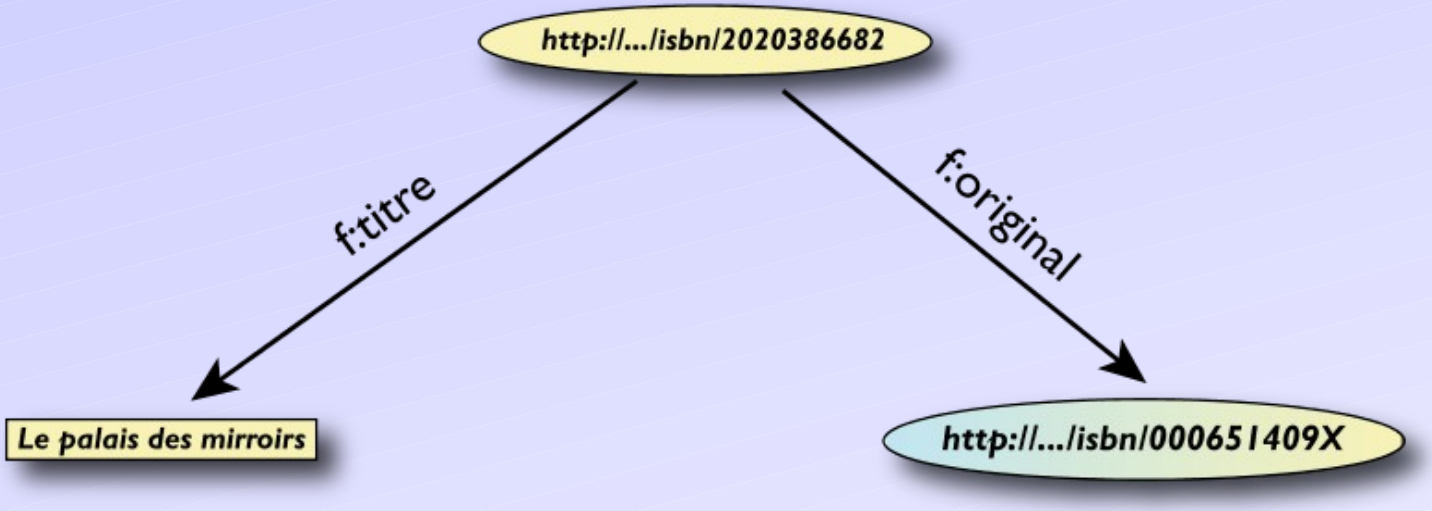
RDF/XML principles



- Encode nodes and edges as XML elements or with literals:

```
«Element for http://…/isbn/2020386682»
  «Element for original»
      «Element for http://…/isbn/000651409X»
  «/Element for original»
«/Element for http://…/isbn/2020386682»
«Element for http://…/isbn/2020386682»
  «Element for titre»
      Le palais des mirroirs
  «/Element for titre»
«/Element for http://…/isbn/2020386682»
```

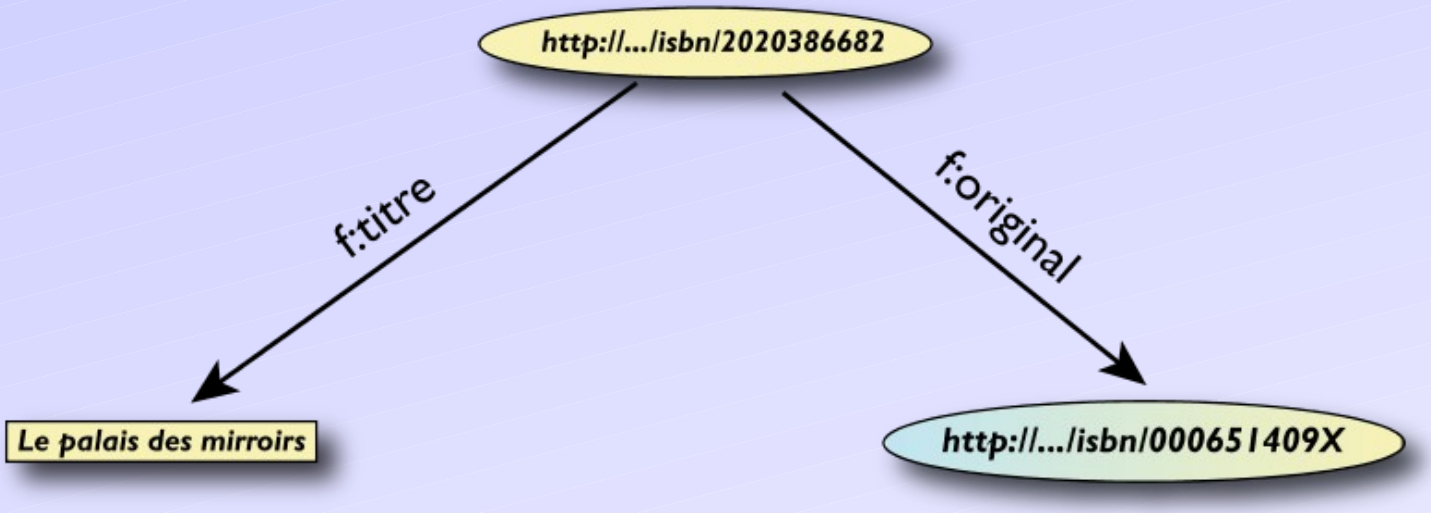
RDF/XML principles (cont.)



- Encode the resources (i.e., the nodes):

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <rdf:Description rdf:about="http://…/isbn/2020386682">
        «Element for original»
            <rdf:Description rdf:about="http://…/isbn/000651409X"/>
        «/Element for f:original»
    </rdf:Description>
<rdf:RDF>
```

RDF/XML principles (cont.)



- Encode the properties (i.e., edges) in their own namespaces:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:f="http://www.editeur.fr"">
    <rdf:Description rdf:about="http://…/isbn/2020386682">
       <f:original>
            <rdf:Description rdf:about="http://…/isbn/000651409X"/>
       </f:original>
    </rdf:Description>
<rdf:RDF>
```

Examples of RDF/XML "simplifications"

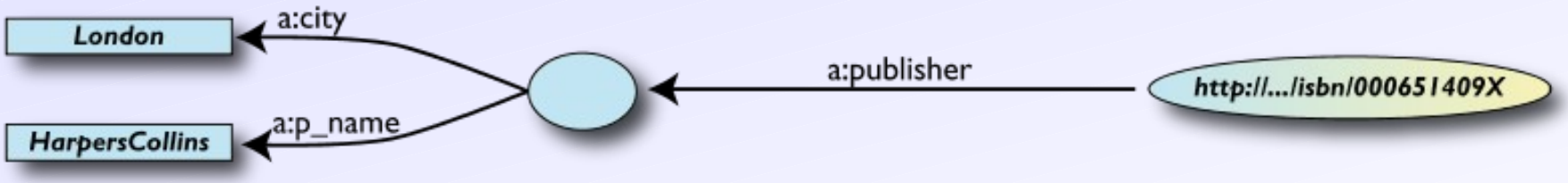
- Object references can be put into attributes
- Several properties on the same resource

```
<rdf:Description rdf:about="http://…/isbn/2020386682">
   <f:original rdf:resource="http://…/isbn/000651409X"/>
   <f:titre>
       Le palais des mirroirs
   </f:titre>
</rdf:Description>
```

- There are other "simplification rules", see the "RDF/XML Serialization" document for details

"Internal" nodes

- Consider the following statement:
  - "the publisher is a «thing» that has a name and an address"
- Until now, nodes were identified with a URI. But…
- …what is the URI of «thing»?



One solution: create an extra URI

```
<rdf:Description rdf:about="http://…/isbn/000651409X">
   <a:publisher rdf:resource="urn:uuid:f60ffb40-307d-…"/>
</rdf:Description>
<rdf:Description rdf:about="urn:uuid:f60ffb40-307d-…">
   <a:p_name>HarpersCollins</a:p_name>
   <a:city>HarpersCollins</a:city>
</rdf:Description>
```

- The resource will be “visible” on the Web
  - care should be taken to define *unique* URI-s
- Serializations may give syntactic help to define local URI-s

Internal identifier ("blank nodes")

```
<rdf:Description rdf:about="http://…/isbn/000651409X">
   <a:publisher rdf:nodeID="A234"/>
</rdf:Description>
<rdf:Description rdf:nodeID="A234">
   <a:p_name>HarpersCollins</a:p_name>
   <a:city>HarpersCollins</a:city>
</rdf:Description>
```

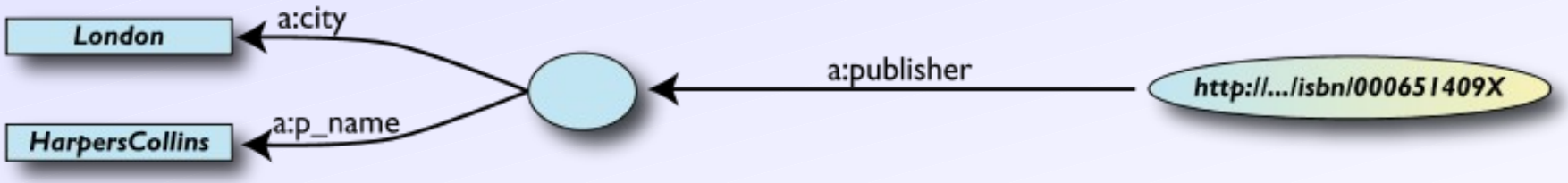
```
<http://…/isbn/2020386682> a:publisher _:A234.
_:A234 a:p_name "HarpersCollins".
```

- Syntax is serialization dependent
- A234 is invisible from outside (it is not a "real" URI!); it is an internal identifier for a resource

Blank nodes: the system can also do it

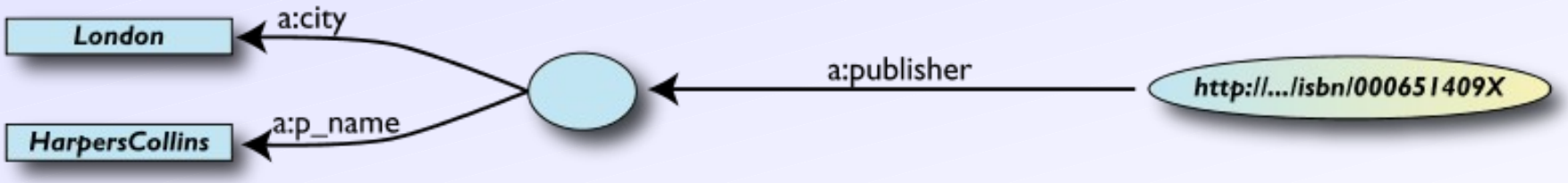
- Let the system create a “nodeID” internally (you do not really care about the name…)

```
<rdf:Description rdf:about="http://…/isbn/000651409X">
  <a:publisher>
      <rdf:Description>
          <a:p_name>HarpersCollins</a:p_name>
          …
      </rdf:Description>
  </a:publisher>
</rdf:Description>
```



Same in Turtle

```
<http://…/isbn/000651409X> a:publisher [
    a:p_name "HarpersCollins";
    …
].
```



Blank nodes: some more remarks

- Blank nodes require attention when merging
  - blanks nodes with identical nodeID-s in *different* graphs are *different*
  - implementations must be careful…
- Many applications prefer not to use blank nodes and define new URI-s “on-the-fly”
  - eg, when triples are in a database
- From a logic point of view, blank nodes represent an “existential” statement
  - “there is a resource such that…”

RDF in programming practice

- For example, using Java+Jena (HP's Bristol Lab):
  - a "Model" object is created
  - the RDF file is parsed and results stored in the Model
  - the Model offers methods to retrieve:
    - triples
    - (property,object) pairs for a specific subject
    - (subject,property) pairs for specific object
    - etc.
  - the rest is conventional programming…
- Similar tools exist in Python, PHP, etc.

Jena example

```
// create a model
Model model=new ModelMem();
Resource subject=model.createResource("URI_of_Subject")
// 'in' refers to the input file
model.read(new InputStreamReader(in));
StmtIterator iter=model.listStatements(subject,null,null);
while(iter.hasNext()) {
   st = iter.next();
   p = st.getProperty();
   o = st.getObject();
   do_something(p,o);
}
```

Merge in practice

- Environments merge graphs automatically
  - e.g., in Jena, the Model can load several files
  - the load merges the new statements automatically

One level higher up
(RDFS, Datatypes)

Need for RDF schemas

- First step towards the “extra knowledge”:
  - define the terms we can use
  - what restrictions apply
  - what extra relationships are there?
- Officially: “RDF Vocabulary Description Language”
  - the term “Schema” is retained for historical reasons…

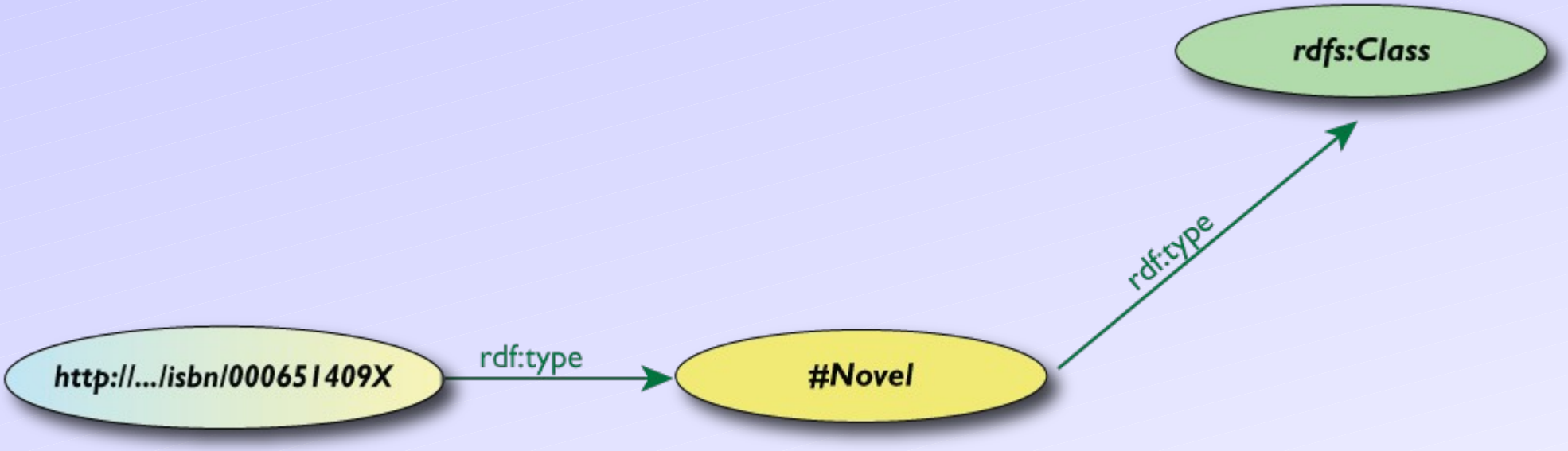
Classes, resources, …

- Think of well known traditional ontologies or taxonomies:
  - use the term “novel”
  - “every novel is a fiction”
  - “«The Glass Palace» is a novel”
  - etc.
- RDFS defines resources and classes:
  - everything in RDF is a “resource”
  - “classes” are also resources, but…
  - …they are also a collection of possible resources (i.e., “individuals”)
    - “fiction”, “novel”, …

Classes, resources, … (cont.)

- Relationships are defined among classes/resources:
  - "typing": an individual belongs to a specific class
    - "«The Glass Palace» is a novel"
    - to be more precise: "«http://.../**000651409X**» is a novel"
  - "subclassing": *all* instances of one are also the instances of the other ("every novel is a fiction")
- *RDFS formalizes these notions in RDF*

Classes, resources in RDF(S)



- RDFS defines the meaning of these terms
  - (these are all special URI-s, we just use the namespace abbreviation)

Schema example in RDF/XML

- The schema part:

```
<rdf:Description rdf:ID="Novel">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>
```

- The RDF data on a specific novel:

```
<rdf:Description rdf:about="http://…/isbn/000651409X">
   <rdf:type rdf:resource="http://…/bookSchema.rdf#Novel"/>
</rdf:Description>
```

An aside: typed nodes in RDF/XML

- A frequent simplification rule: instead of

```
<rdf:Description rdf:about="http://...">
    <rdf:type rdf:resource="http://..../something#ClassName>
    ...
</rdf:Description>
```

use:

```
<yourNameSpace:ClassName rdf:about="http://...">
    ...
</yourNameSpace:ClassName>
```

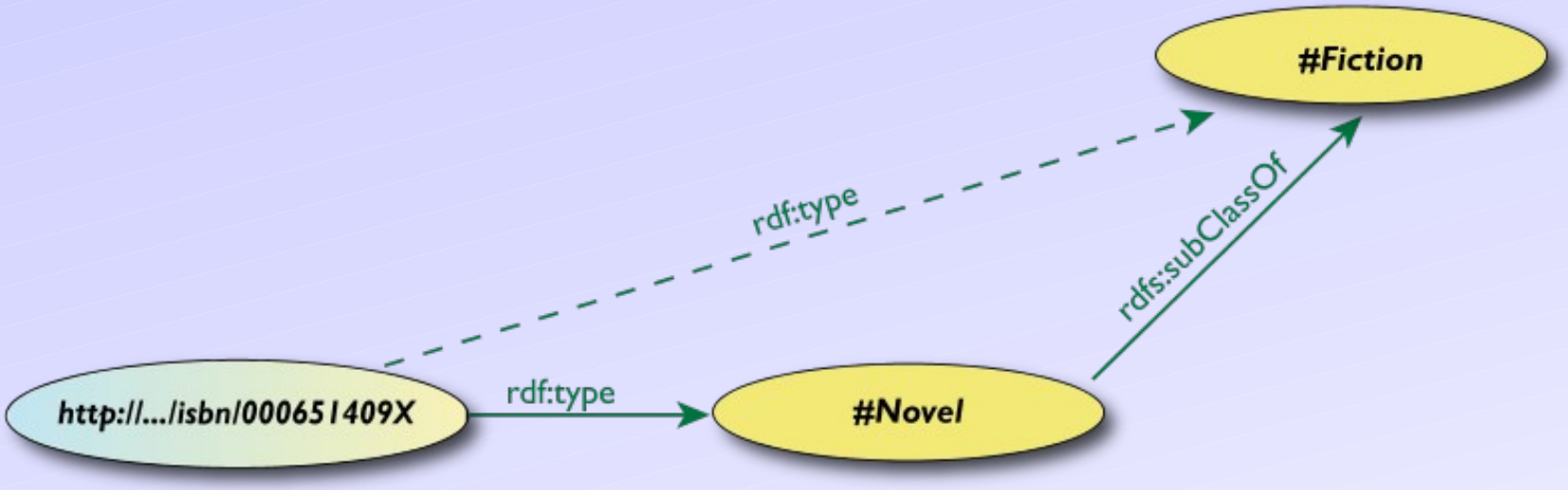
ie:

```
<a:Novel rdf:about="http://…/isbn/000651409X">
    ...
</a:Novel>
```

Further remarks on types

- A resource may belong to several classes
  - rdf:type is just a property…
  - "«The Glass Palace» is a novel, but «The Glass Palace» is also an «inventory item»…"
  - i.e., it is *not* like a datatype!
- The type information may be very important for applications
  - e.g., it may be used for a categorization of possible nodes
  - probably the most frequently used RDF property…
- (remember the "Person" in our example?)

Inferred properties




```
(<http://…/isbn/000651409X> rdf:type #Fiction)
```

- is not in the original RDF data…
- …but can be inferred from the RDFS rules
- RDFS environments return that triple, too

Inference: let us be formal…

- The RDF Semantics document has a list of (33) *entailment rules*:
  - "if such and such triples are in the graph, add this and this"
  - do that recursively until the graph does not change
- The relevant rule for our example:

```
If:
  uuu rdfs:subClassOf xxx .
  vvv rdf:type uuu .
Then add:
  vvv rdf:type xxx .
```

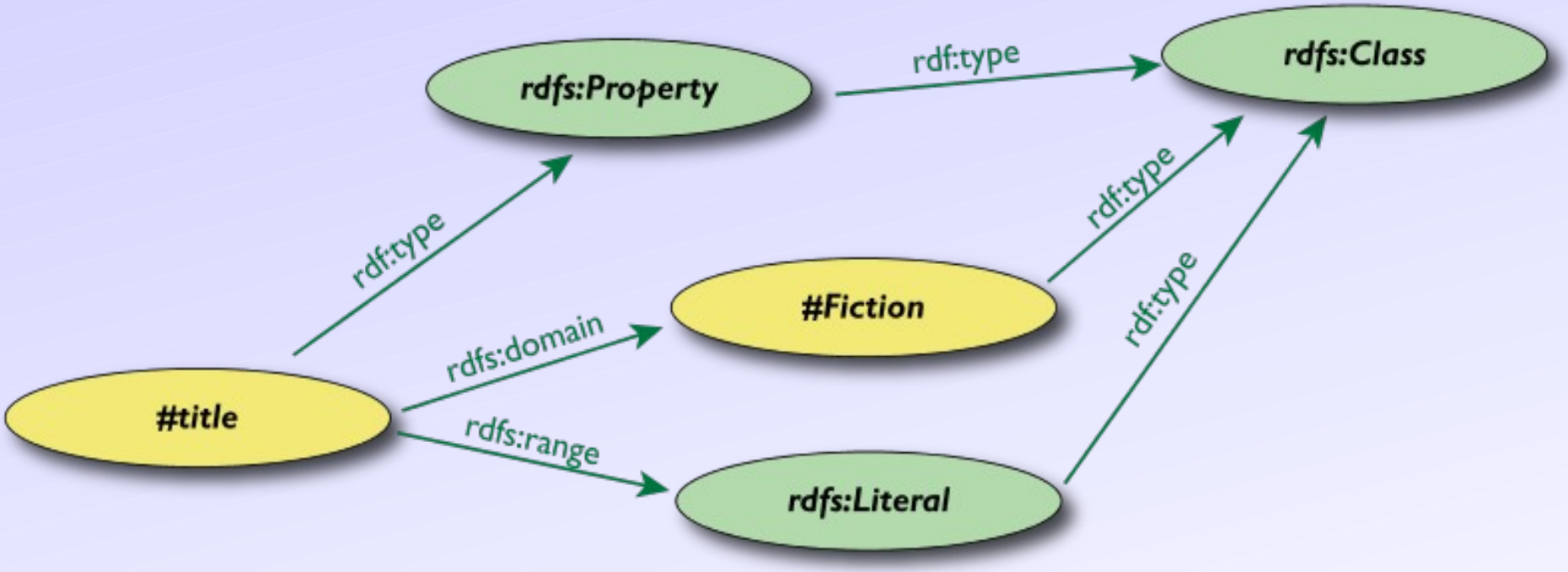
Properties

- Property is a special class (`rdf:Property`)
  - properties are also resources identified by URI-s
- There is also a possibility for a “sub-property”
  - all resources bound by the “sub” are also bound by the other
- Range and domain of properties can be specified
  - i.e., what type of resources serve as object and subject

Properties (cont.)

- Properties are also resources (named via URI–s)…
- So properties of properties can be expressed as… RDF properties
  - this twists your mind a bit, but you can get used to it
- For example, `(P rdfs:domain C)` means:
  - `P` is a property
  - `C` is a class
  - when using `P`, I can *infer* that the "subject" is of type `C`

Property specification example



Property specification serialized

- In RDF/XML:

```
<rdf:Property rdf:ID="title">
  <rdfs:domain rdf:resource="#Fiction"/>
  <rdfs:range rdf:resource="http://...#Literal"/>
</rdf:Property>
```

- In Turtle:

```
:title
  rdf:type     rdf:Property;
  rdfs:domain  :Fiction;
  rdfs:range   rdfs:Literal.
```

What does this mean?

- Again, new relations can be deduced. Indeed, if

```
:title
  rdf:type     rdf:Property;
  rdfs:domain  :Fiction;
  rdfs:range   rdfs:Literal.

<http://…/isbn/000651409X> :title "The Glass Palace" .
```

- then the system can *infer* that:

```
<http://…/isbn/000651409X> rdf:type :Fiction .
```

Literals

- Literals may have a data type
  - floats, integers, booleans, etc, defined in XML Schemas
  - full XML fragments
- (Natural) language can also be specified

Examples for datatypes

```
<rdf:Description rdf:about="http://…/isbn/000651409X">
   <page_number rdf:datatype="http://...#integer>543</page_number>
   <publ_date rdf:datatype="http://...#gYear>2000</publ_date>
   <price rdf:datatype="http://...#float>6.99</price>
</rdf:Description>
```

```
<http://…/isbn/000651409X>
      :page_number "543"^^xsd:integer ;
      :publ_date   "2000"^^xsd:gYear ;
      :price       "6.99"^^xsd:float .
```

Examples for language tags

```
<rdf:Description rdf:about="http://…/isbn/000651409X">
   <title xml:lang="en">The Glass Palace</title>
   <fr:titre xml:lang="fr">Le palais des mirroirs</fr:titre>
</rdf:Description>
```

```
<http://…/isbn/000651409X>
      :title     "The Glass Palace"@en ;
      fr:titre   "Le palais des mirroirs"@fr .
```

XML literals in RDF/XML

- XML Literals
  - makes it possible to “include” XML vocabularies into RDF:

```
<rdf:Description rdf:about="#Path">
   <axsvg:algorithmUsed rdf:parseType="Literal">
      <math xmlns="...">
        <apply>
          <laplacian/>
          <ci>f</ci>
        </apply>
      </math>
   </axsvg:algorithmUsed>
</rdf:Description/>
```

A bit of RDFS can take you far…

- Remember the power of merge?
- We could have used, in our example:
  - **`f:auteur`** is a subproperty of **`a:author`** and vice versa (although we will see other ways to do that…)
- Of course, in some cases, more complex knowledge is necessary (see later…)

Some predefined structures…

(collections, containers)

Predefined classes and properties

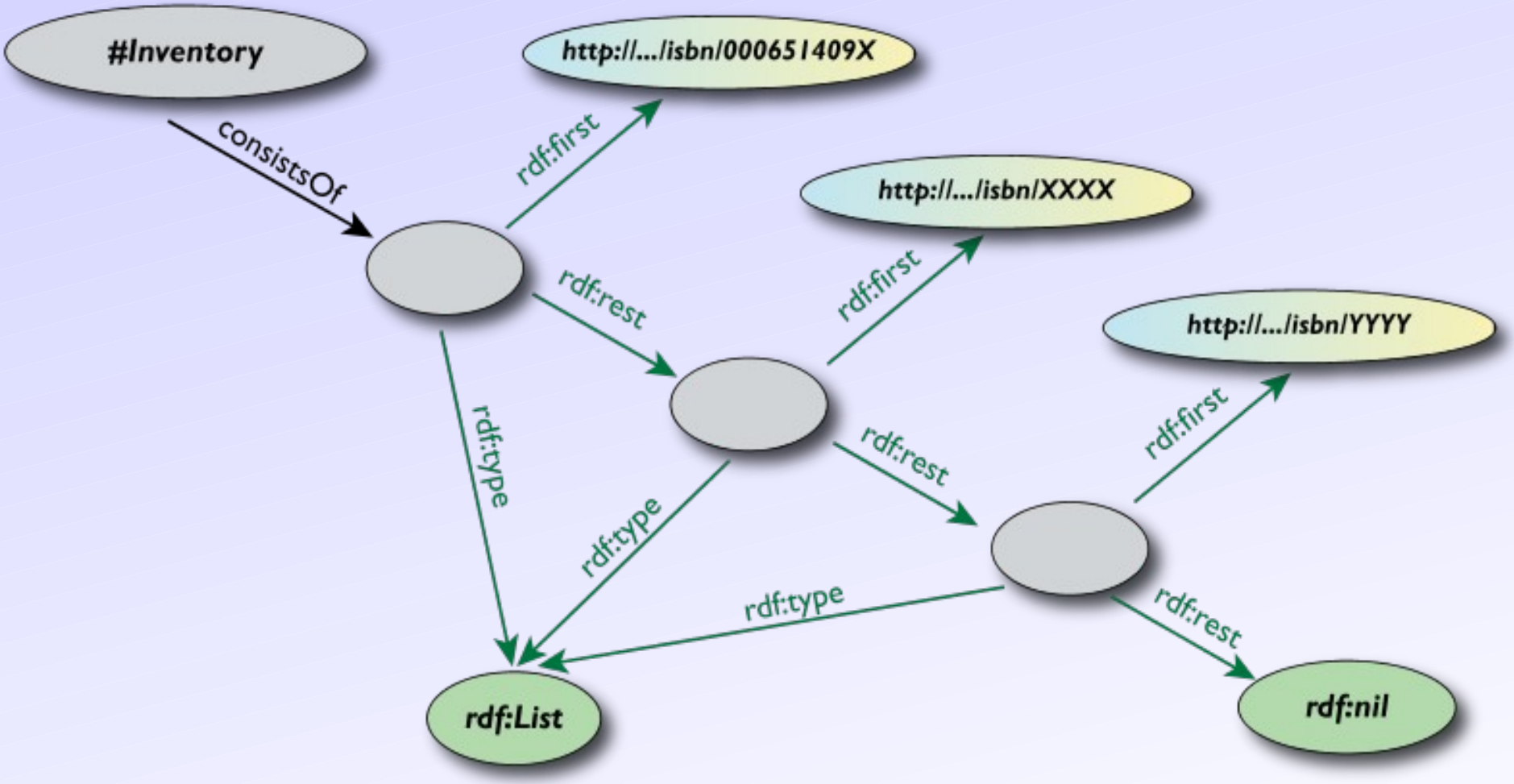
- RDF(S) has some predefined classes and properties
- These are not new “concepts” in the RDF Model, just resources with an agreed semantics
- Examples:
  - collections (a.k.a. lists)
  - containers: sequence, bag, alternatives
  - reification
  - `rdfs:comment`, `rdfs:seeAlso`, `rdf:value`

Collections (lists)

- We could have the following statement:
  - "The book inventory is a «thing» that consists of **<http://…/isbn/000651409X>**, **<http://…/isbn/000XXXX>**, …"
- But we also want to express the constituents *in this order*
- Using blank nodes is not enough

Collections (lists) (cont.)

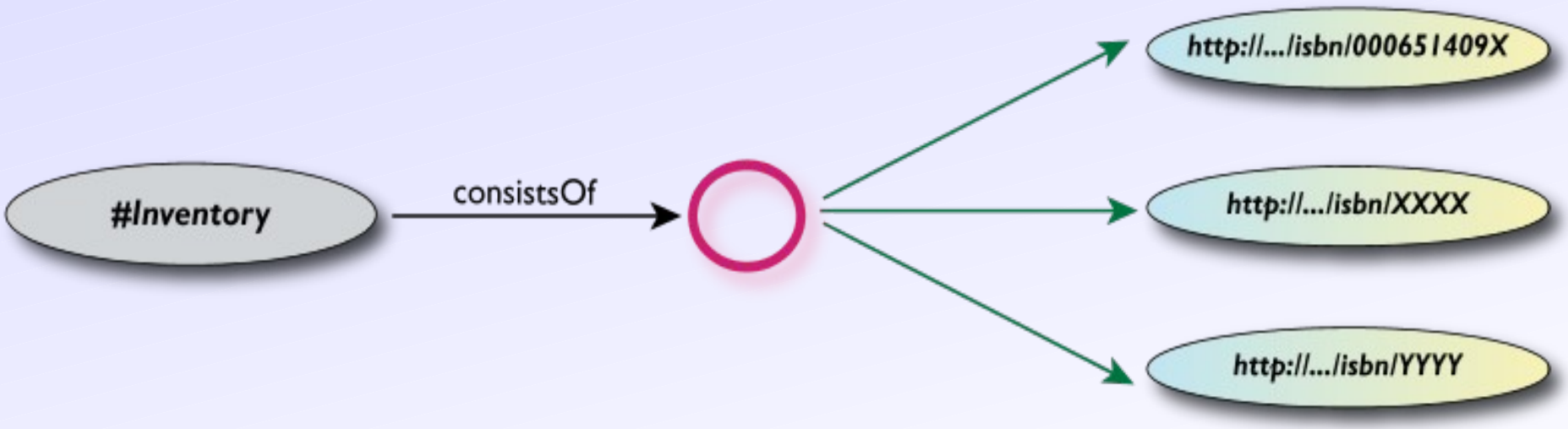
- Familiar structure for Lisp programmers…



The same in RDF/XML and Turtle

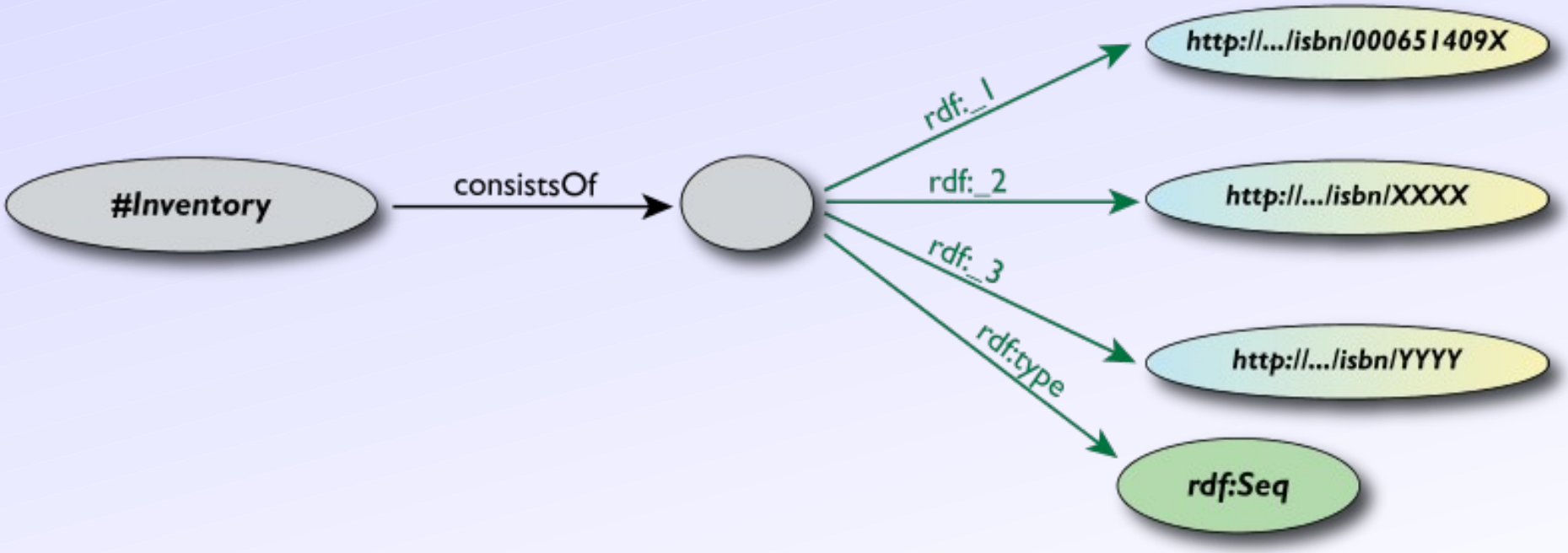
```
<rdf:Description rdf:about="#Inventory">
    <a:consistsOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://.../isbn/000651409X"/>
        <rdf:Description rdf:about="http://.../isbn/XXXX"/>
        <rdf:Description rdf:about="http://.../isbn/YYYY"/>
    </a:consistsOf>
</rdf:Description>
```

```
:Inventory a:consistsOf
     (<http://.../isbn/000651409X> <http://.../isbn/XXXX> …) .
```



Sequences

- Use the predefined:
  - RDF Schema class **`Seq`**
  - RDF properties **`rdf:_1`**, **`rdf:_2`**, …
- The *agreed* semantics is of a sequential containment



Sequences serialized

- In RDF/XML:

```
<rdf:Description rdf:about="#Inventory">
  <a:consistsOf>
      <rdf:Description>
          <rdf:type rdf:resource="http:...rdf-syntax-ns#Seq">
          <rdf:_1 rdf:resource="http://.../isbn/000651409X>
          ...
      </rdf:Description>
  </a:consistsOf>
</rdf:Description/>
```

- In Turtle:

```
:Inventory
  a:consistsOf [
    rdf:type <http:...rdf-syntax-ns#Seq>;
    rdf:_1   <http://.../isbn/000651409X>;
          ...
  ].
```

Sequences (simplified RDF/XML)

```
<rdf:Description rdf:about="#Inventory">
    <a:consistsOf>
        <rdf:Seq>
            <rdf:li rdf:resource="http://.../isbn/000651409X">
            ...
        </rdf:Seq>
    </a:consistsOf>
</rdf:Description/>
```

Other containers

- Also defined in RDFS
  - rdf:Bag
    - a general bag, no particular semantics attached
  - rdf:Alt
    - *agreed* semantics: only one of the constituents is “valid”
- Note: these containers are incompletely defined semantically; it is better not to use them…
  - use repeated predicates for bags
  - use lists for sequences

How to get RDF Data?
(Microformats, GRDDL, RDFa)

Simple approach

- Write RDF/XML or Turtle “manually”
- In some cases that is necessary, but it really does not scale…

RDF with XHTML

- Obviously, a huge source of information
- By adding some “meta” information, the same source can be reused for, eg, data integration, better mashups, etc
  - typical example: your personal information, like address, should be readable for humans *and* processable by machines
- Two solutions have emerged:
  - use microformats and convert the content into RDF
  - add RDF statements directly into XHTML via RDFa

Microformats

- *Not* a Semantic Web specification, originally
  - there is a separate microformat community
- Approach: re-use (X)HTML attributes and elements to add “meta” information
  - typically **`@abbr`**, **`@class`**, **`@title`**, …
  - different agreements for different applications

Microformat example: hCalendar

- Goal: “markup” calendaring information on your (X)HTML page
  - use a community agreement using, eg, :
    - `@class` for event name
    - `abbr` element for dates
    - `@title` for date values
    - etc.
- Automatic procedures (ie, calendaring applications) may then get to the right data

Microformat example: hCalendar

Dan Connolly, W3C

http://www.w3.org/People/Connolly/

Wiki Notes | Tiddly Notes | VeriSign PIP | Knowee | My Home | World Clock | New blog | Mobical | Favikis | Twines | SW | Other bookmarks

Dan Connolly

Research Scientist, MIT/CSAIL

connolly@w3.org *
32 Vassar Street
Room 32-G506 *
Cambridge, MA, 02139 USA
+1-617-395-0241, DanC *

>>> hCard

standards: HTML WG, TAG, GRDDL WG, RDF Calendar, QA, DAWG/SPARQL, Semantic Web IG, OWL, HTML 2, ESW

research: *breadcrumbs* journal/weblog, cwm, N3, tabulator, PAW, TAMI, microformats open source

life: family, volleyball, guitar

Dan Connolly is a research scientist at the MIT Computer Science and Artificial Intelligence Laboratory (CSAIL) in the Decentralized Information Group (DIG) and a member of the technical staff of the World Wide Web Consortium (W3C). His research interest is

>>> hCalendar Events

Mar 7-11, 2008: to **Austin, TX**
for SXSW Interactive

Apr 20 - 22 : to **Beijing, China**
for W3C AC meeting, linked data workshop
trip stuff

May 19-May 22: to **Bristol**
for TAG ftf
trip stuff

Sep 23 - 25 : in **Kansas City**
TAG meeting

Oct 20 - 25 : to **NCE**
for W3C TPAC

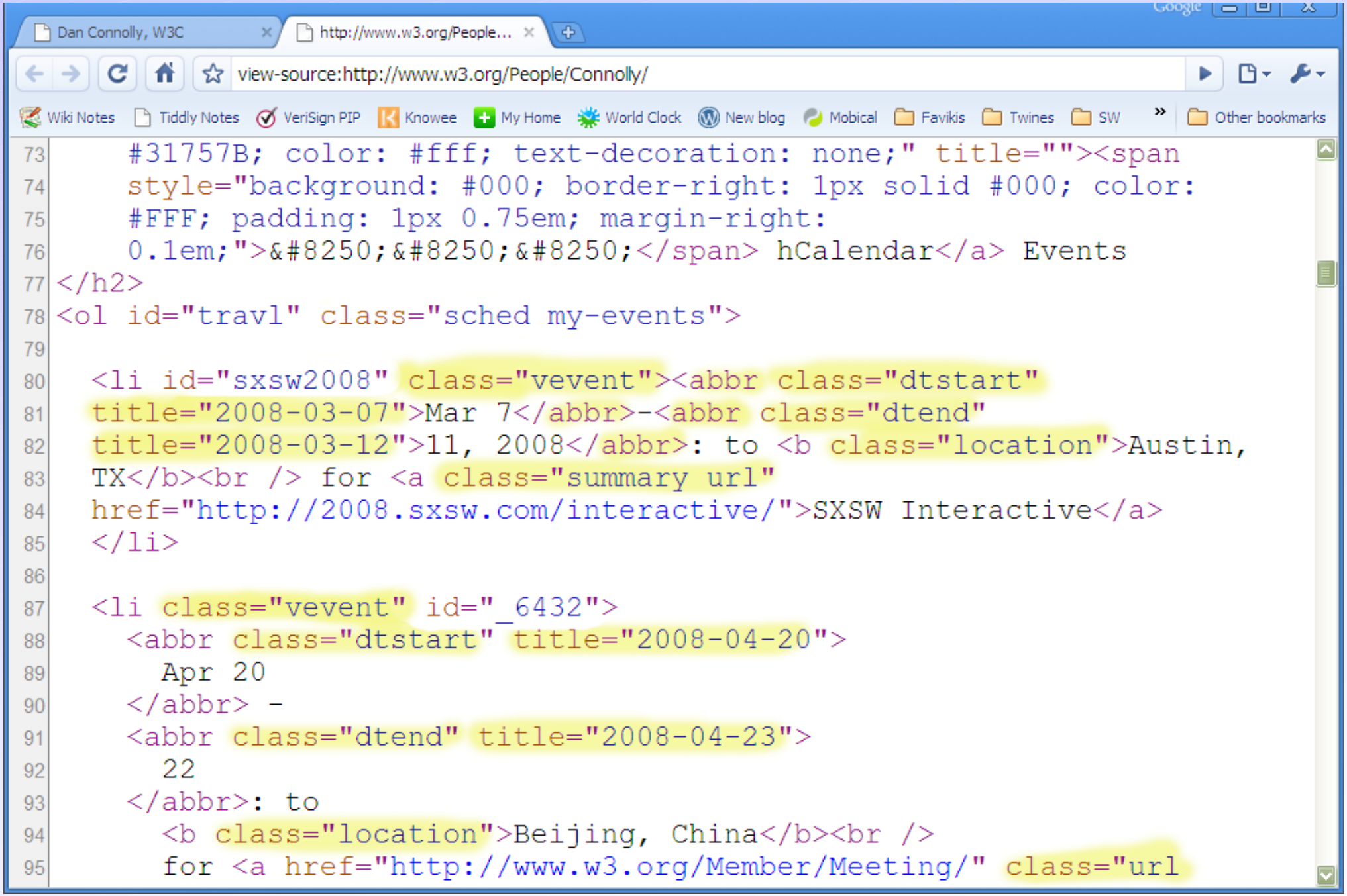
Nov 1 - 3 : to **BOS**
TAMI meeting

Dec 08 - 11 : to **Cambridge, MA**
TAG meeting
itin

Feb 2 to 7: to **Denver**
hoping to go for Web Directions North

Earlier travel/talks/events include Tools of

Behind the scenes…



Microformat extraction

- To use it on the Semantic Web, microformat data should be converted to RDF
- A simple transformation (eg, in XSLT) can be defined, yielding:

```
<http://www.w3.org/People/Connolly/#sxsw2008>
    a hcal:Vevent;
    hcal:organizer <http://www.w3.org/People/Connolly/#me>;
    hcal:summary "SXSW Interactive";
    hcal:dtstart "2008-03-07"^^xsd:date;
    hcal:dtend "2008-03-12"^^xsd:date;
    hcal:url <http://2008.sxsw.com/interactive/>;
    hcal:location "Austin, TX" .
```

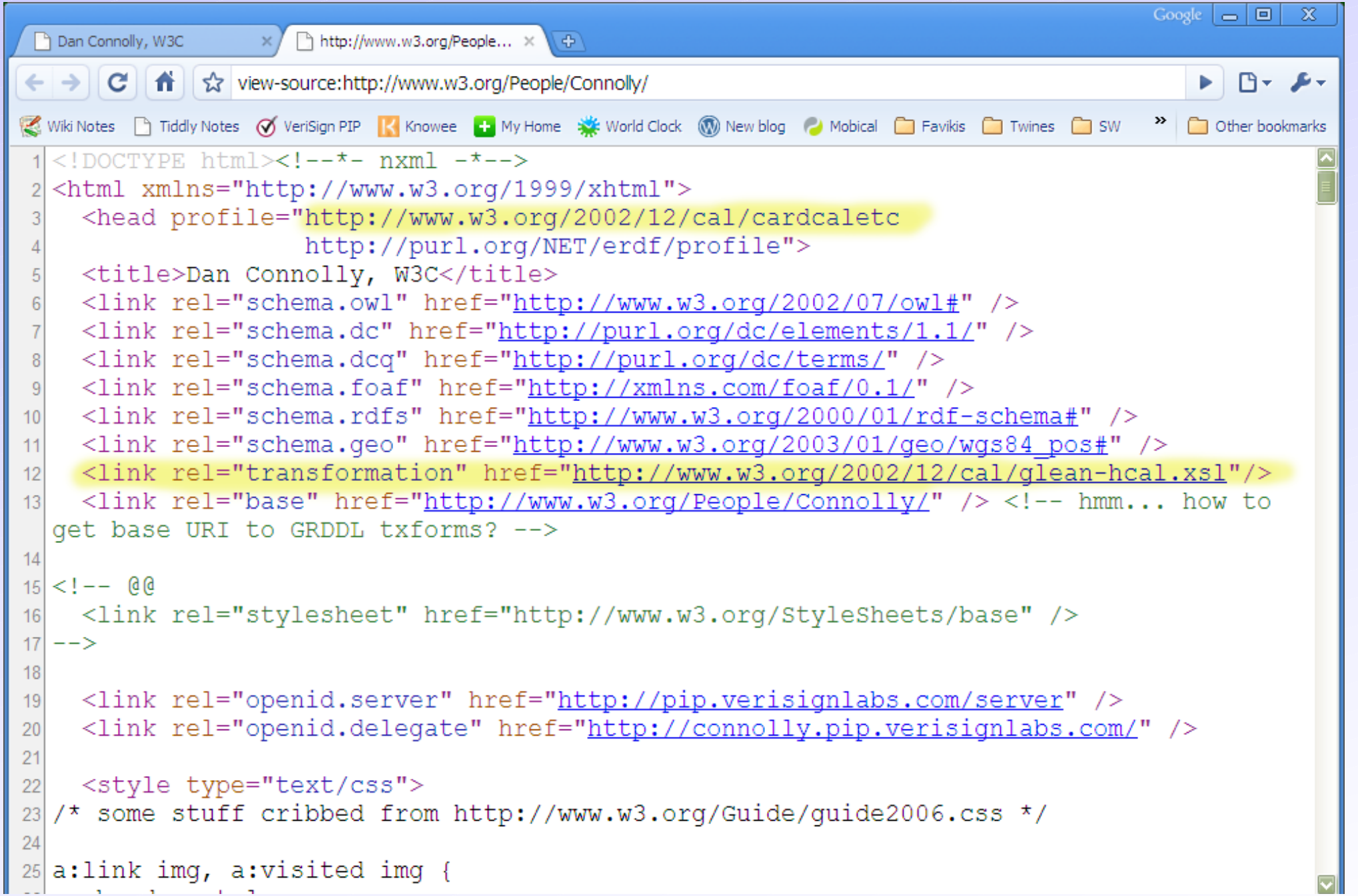
So far so good, but…

- The XSLT transformation is hCalendar specific
  - each microformat dialect needs its own
- How does a general processor find the right transformation?
- *Enter GRDDL*

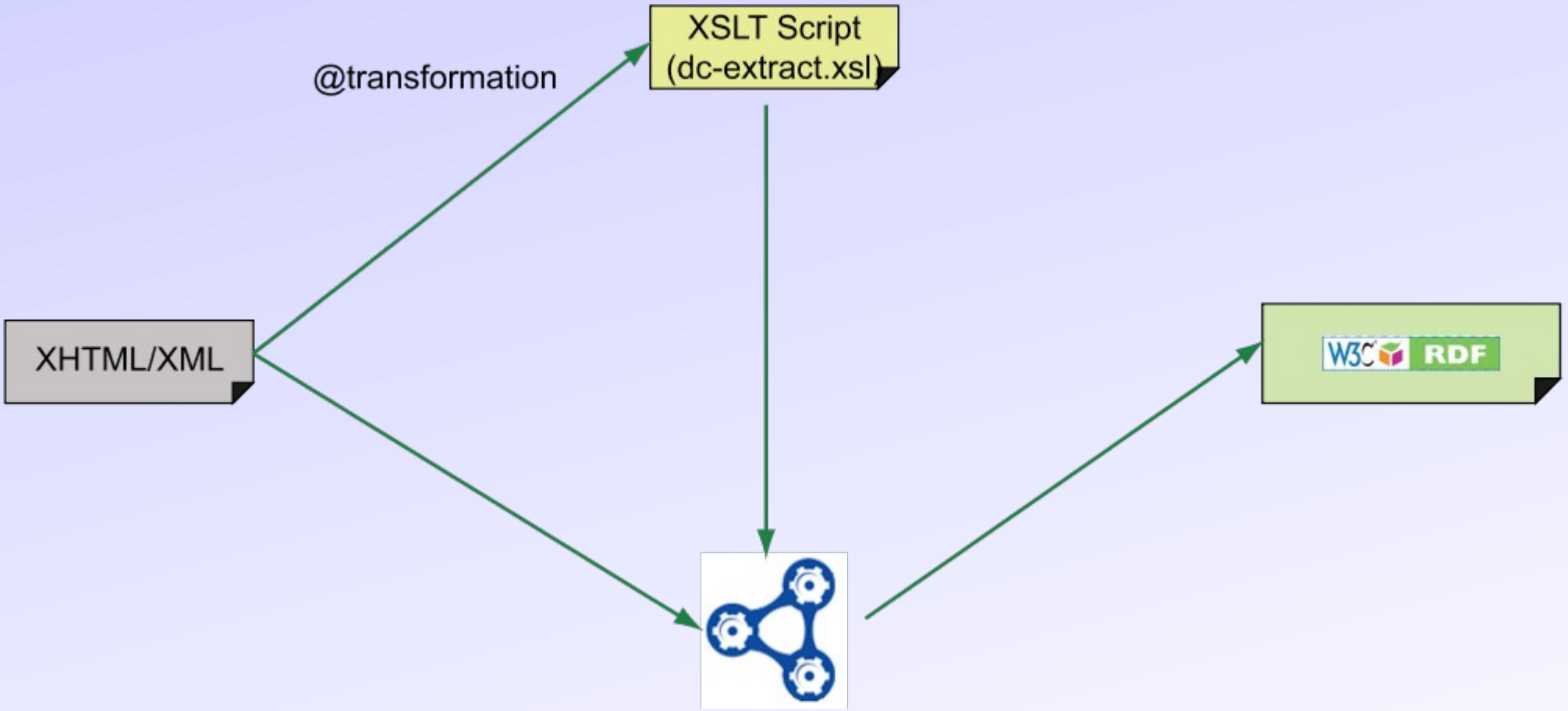
GRDDL: find the right transformation

- GRDDL defines
  - a few extra attribute values to *locate* the right transformation
  - a precise processing model on how the transformation(s) should be applied to generate RDF
- Note: we describe GRDDL in terms of XHTML (and microformats) but *GRDDL can be used for any XML data*

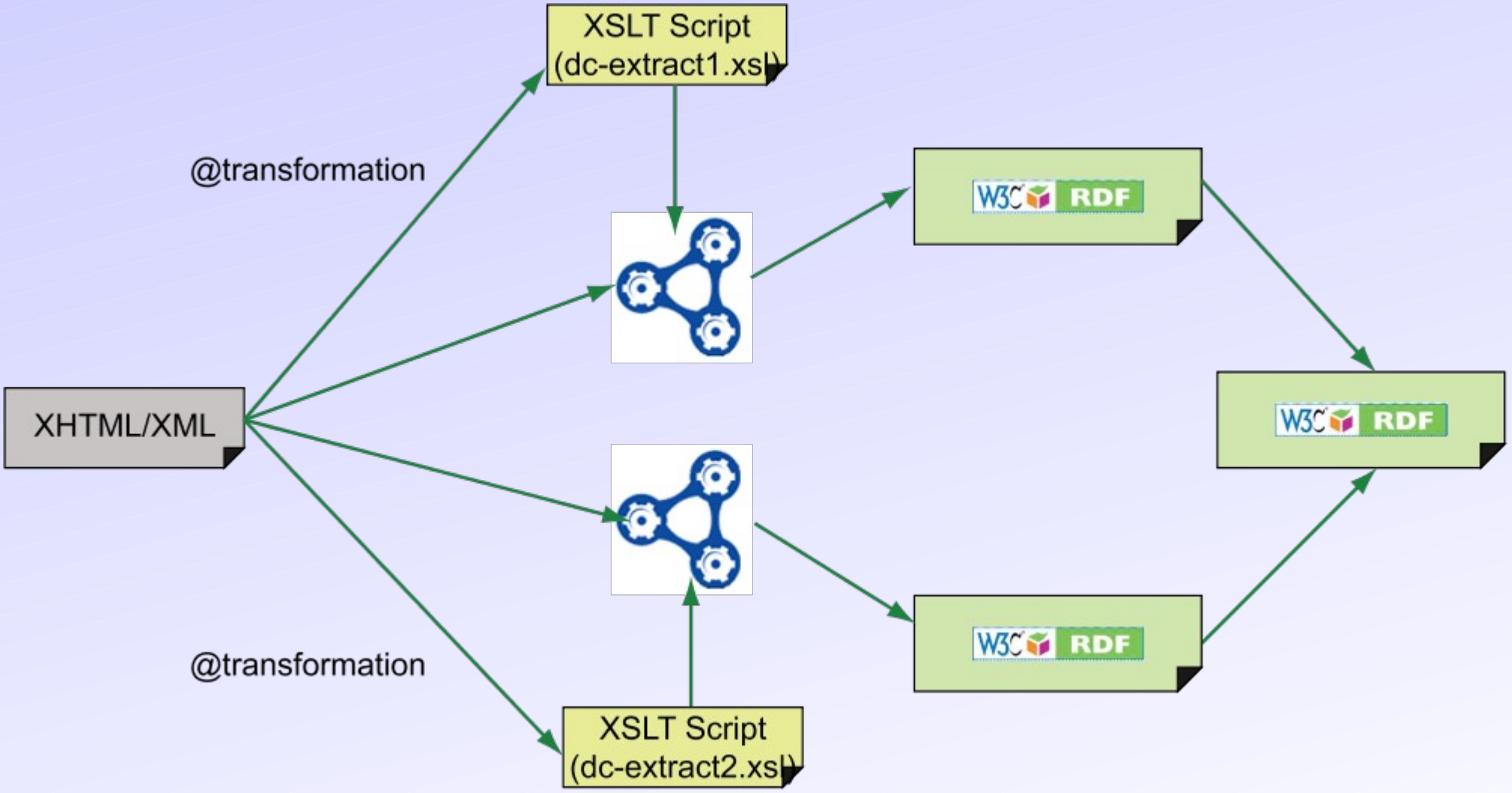
GRDDL: find the right transformation



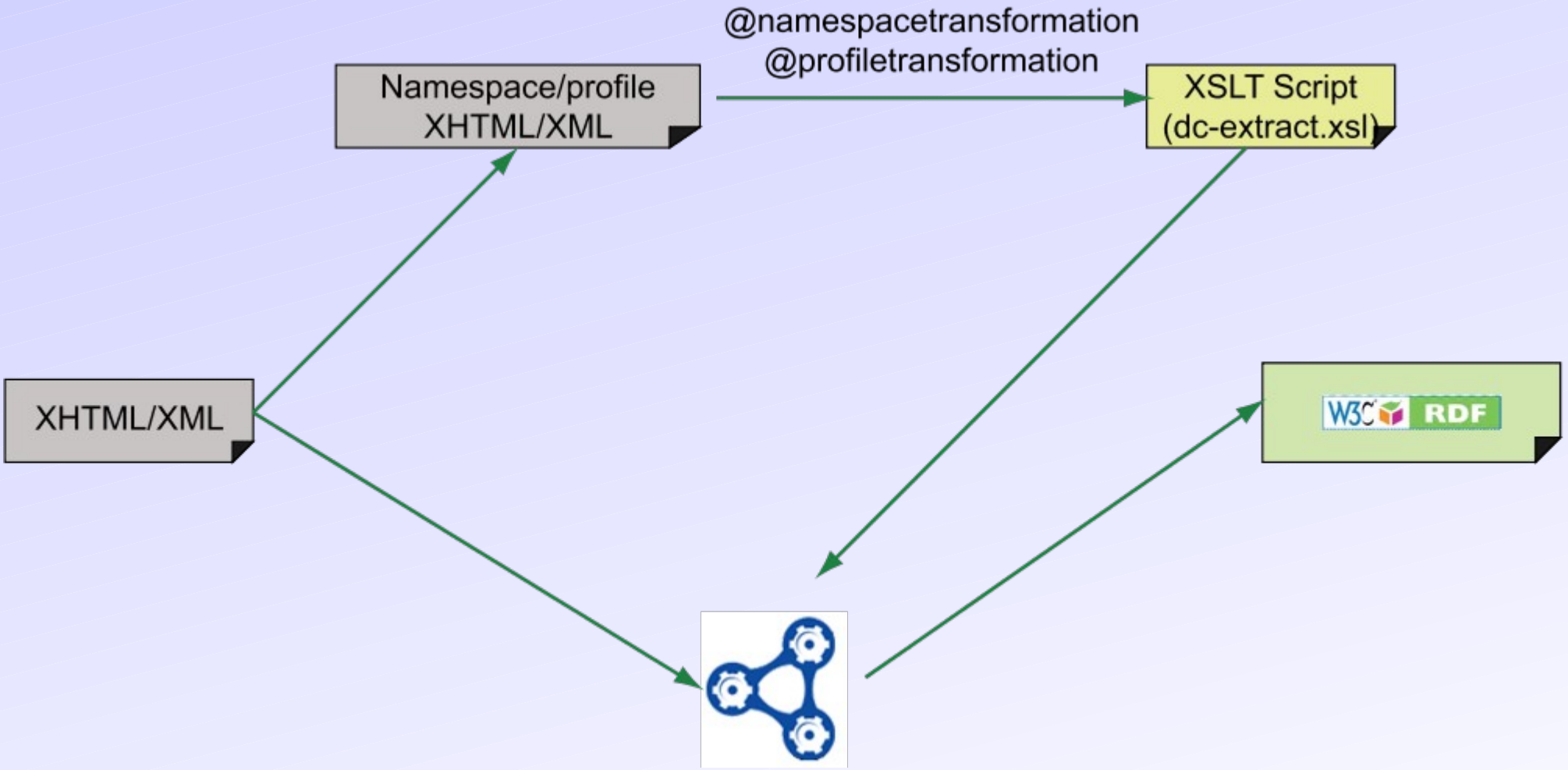
The GRDDL process: simple case



The GRDDL process: merging case



The GRDDL process: indirect case



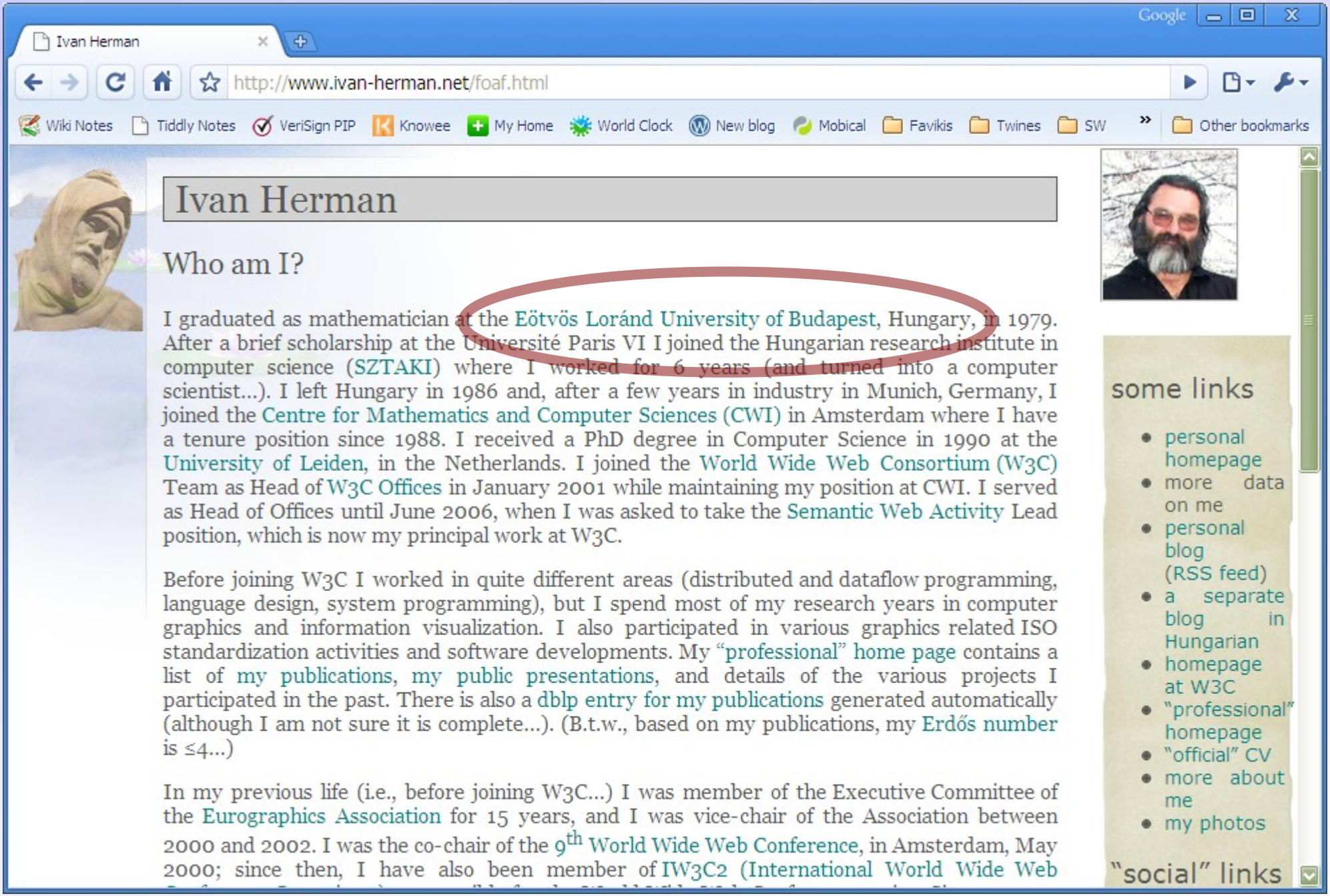
Microformats & GRDDL: pros and cons

- Pros:
  - simple to define/add new vocabularies
    - there is a strong microformat community for this
  - works with all current browsers, markup validators, etc
  - fairly user friendly, easy to understand and use
- Cons:
  - does not scale well for complex vocabularies
    - remember: needs a transformation per vocabulary
  - difficult to *mix* vocabularies within one page; what if the usage of an attribute clashes among different vocabularies?
  - some of the attributes are meant for other usage
    - eg, the **`abbr`** element, the **`@title`** attribute, …

An alternative solution: XHTML+RDFa

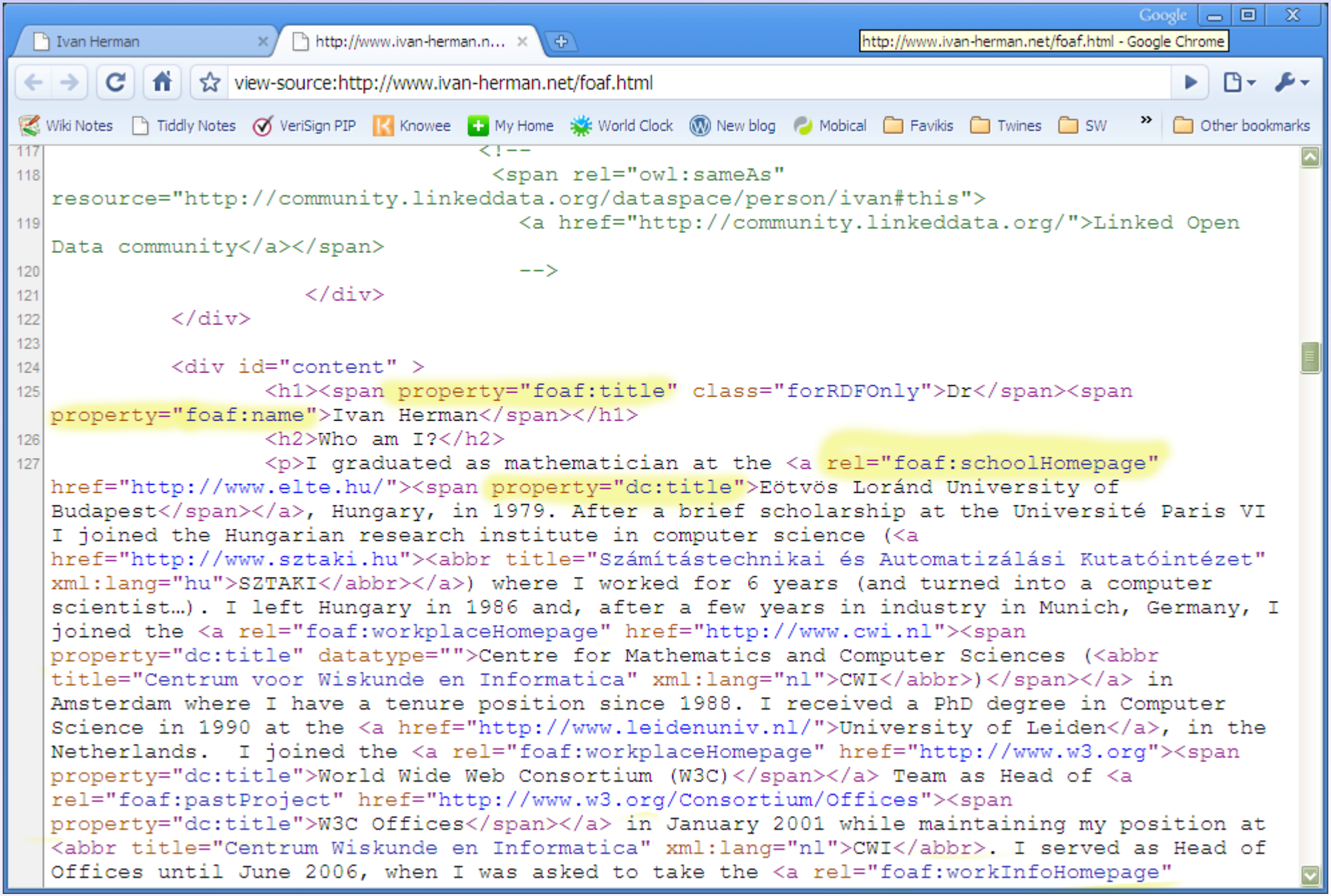
- RDFa also uses (X)HTML attributes to add “meta” information
- However
  - it also uses proprietary attributes to avoid clashes with the intended usage of the (X)HTML ones
  - it includes a namespace+URI mechanism for disambiguation
  - it is *one* set of attributes for *any* vocabularies

XHTML+RDFa example

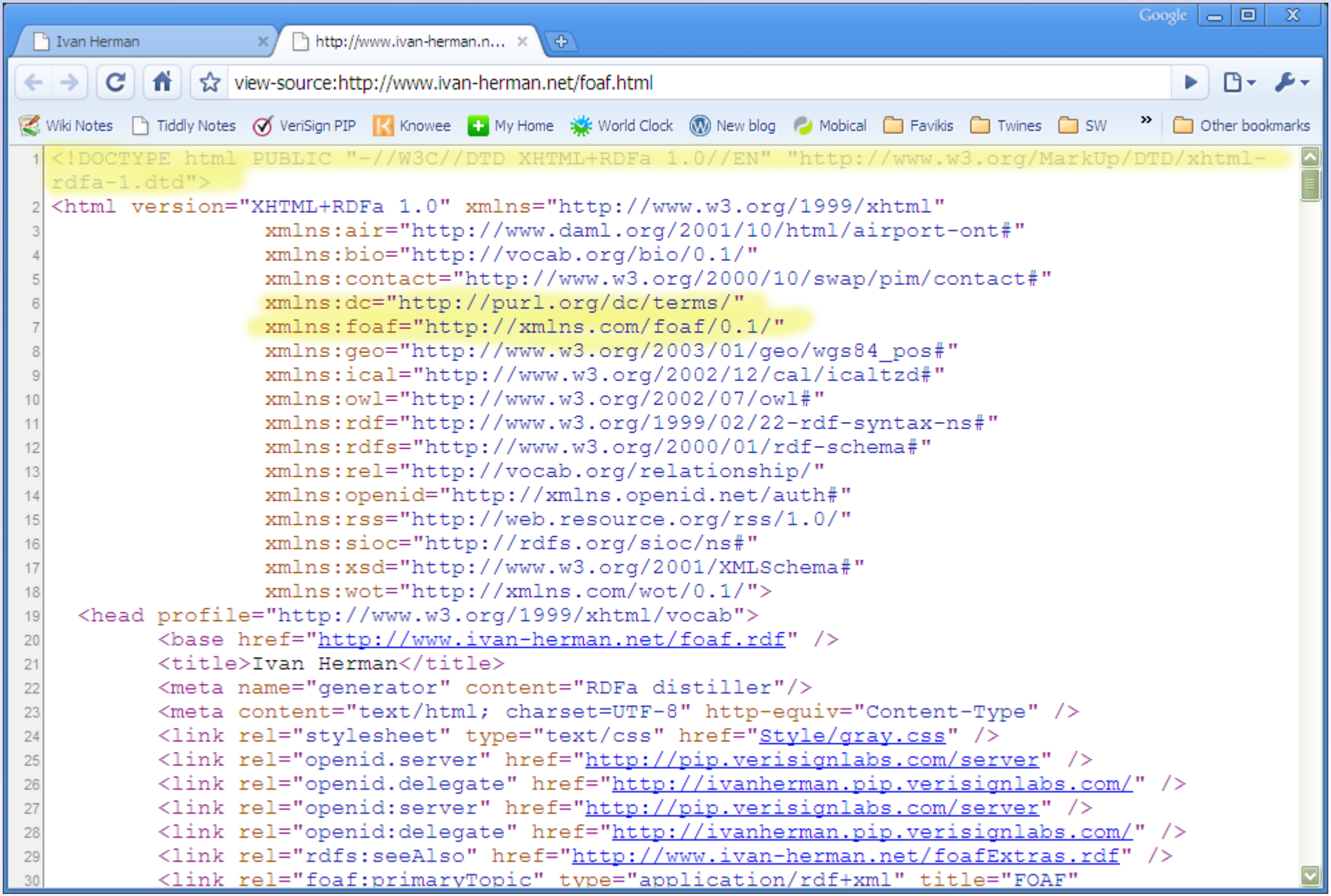


W3C Semantic Web

Same example behind the scenes…



Same example behind the scenes…



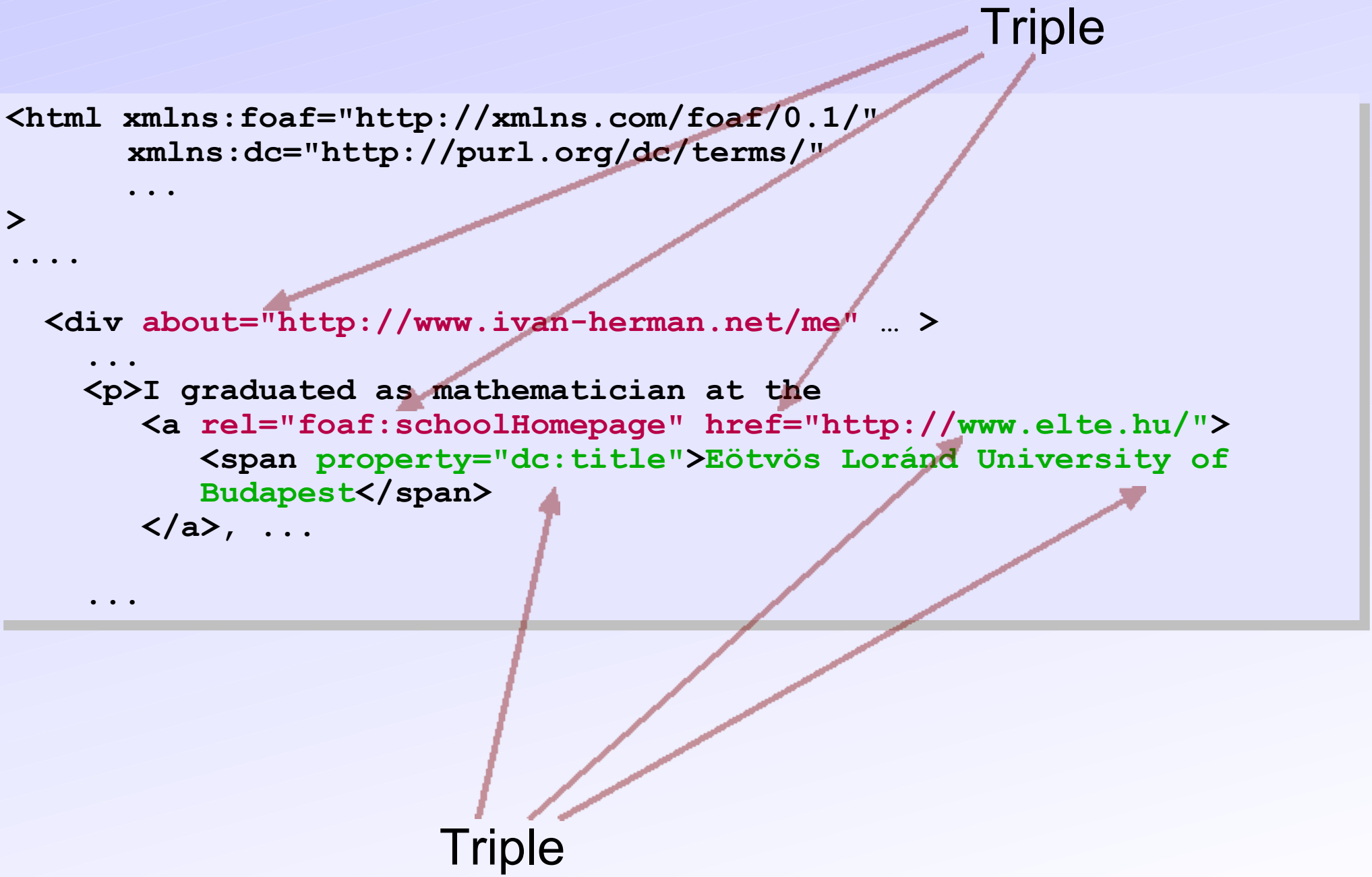
In a slightly more readable format…

```
<html xmlns:foaf="http://xmlns.com/foaf/0.1/"
      xmlns:dc="http://purl.org/dc/terms/"
      ...
>
....

  <div about="http://www.ivan-herman.net/me" … >
    ...
    <p>I graduated as mathematician at the
       <a rel="foaf:schoolHomepage" href="http://www.elte.hu/">
          <span property="dc:title">Eötvös Loránd University of
          Budapest</span>
       </a>, ...

    ...
```

In a slightly more readable format…



... yielding

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix dc:   <http://purl.org/dc/terms/>.

<http://www.ivan-herman.net/me>
    foaf:schoolHomepage <http://www.elte.hu/>.

<http://www.elte.hu/>
    dc:title "Eötvös Loránd University of Budapest".
```

Microformats or RDFa?

- There has been many unnecessary controversies
- For simple, single usage applications microformats are enough
  - GRDDL bridges them to the rest of the Semantic Web
- For more complex documents RDFa is great
- It often boils down to matters of taste…

Bridge to relational databases

- Data on the Web are mostly stored in databases
- "Bridges" are being defined:
  - a layer between RDF and the relational data
    - RDB tables are "mapped" to RDF graphs, possibly on the fly
    - different mapping approaches are being used
  - a number RDB systems offer this facility already (eg, Oracle, OpenLink, …)
- A survey on mapping techniques has been published at W3C
- W3C may engage in a standardization work in this area

Linking Open Data

Linking Open Data Project

- Goal: “expose” open datasets in RDF
- *Set RDF links among the data items* from different datasets
- Set up query endpoints
- Altogether billions of triples, millions of links…

Example data source: DBpedia

- DBpedia is a community effort to
  - extract structured (“infobox”) information from Wikipedia
  - provide a query endpoint to the dataset
  - interlink the DBpedia dataset with other datasets on the Web

Extracting structured data from Wikipedia

Amsterdam

The Keizersgracht at dusk

Location of Amsterdam

Coordinates: 52°22'23"N 4°53'32"E

Country	Netherlands
Province	North Holland
Government	
- Type	Municipality
- Mayor	Job Cohen[1] (PvdA)
- Aldermen	Lodewijk Asscher Carolien Gehrels Tjeerd Herrema Maarten van Poelgeest Marijke Vos
- Secretary	Erik Gerritsen
Area [2][3]	
- City	219 km² (84.6 sq mi)
- Land	166 km² (64.1 sq mi)
- Water	53 km² (20.5 sq mi)
- Urban	1,003 km² (387.3 sq mi)
- Metro	1,815 km² (700.8 sq mi)
Elevation [4]	2 m (7 ft)
Population (1 October 2008)[5][6]	
- City	755,269
- Density	4,459/km² (11,548.8/sq mi)
- Urban	1,364,422
- Metro	2,158,372
- Demonym	Amsterdammer
Time zone	CET (UTC+1)
- Summer (DST)	CEST (UTC+2)
Postcodes	1011 – 1109
Area code(s)	020

Website: www.amsterdam.nl

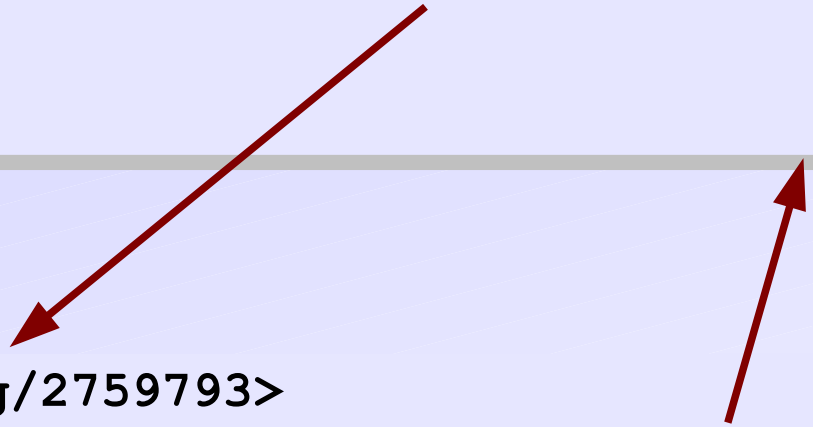
```
@prefix dbpedia <http://dbpedia.org/resource/>.
@prefix dbterm  <http://dbpedia.org/property/>.

dbpedia:Amsterdam
   dbterm:officialName "Amsterdam" ;
   dbterm:longd "4" ;
   dbterm:longm "53" ;
   dbterm:longs "32" ;
   ...
   dbterm:leaderTitle "Mayor" ;
   dbterm:leaderName dbpedia:Job_Cohen ;
   ...
   dbterm:areaTotalKm "219" ;
   ...
dbpedia:ABN_AMRO
   dbterm:location dbpedia:Amsterdam ;
   ...
```

W3C Semantic Web

Automatic links among open datasets

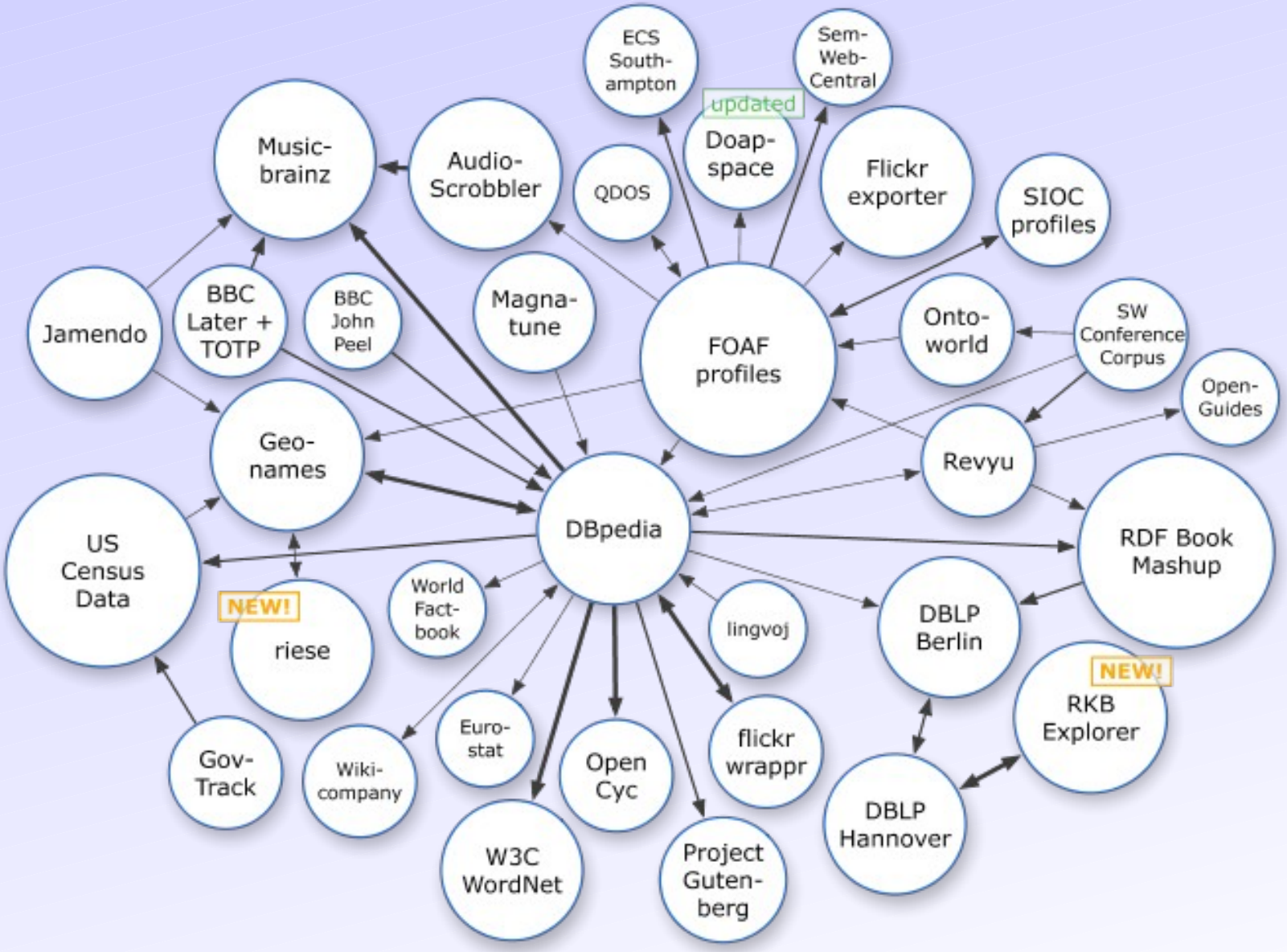
```
<http://dbpedia.org/resource/Amsterdam>
  owl:sameAs <http://rdf.freebase.com/ns/...> ;
  owl:sameAs <http://sws.geonames.org/2759793> ;
  ...
```



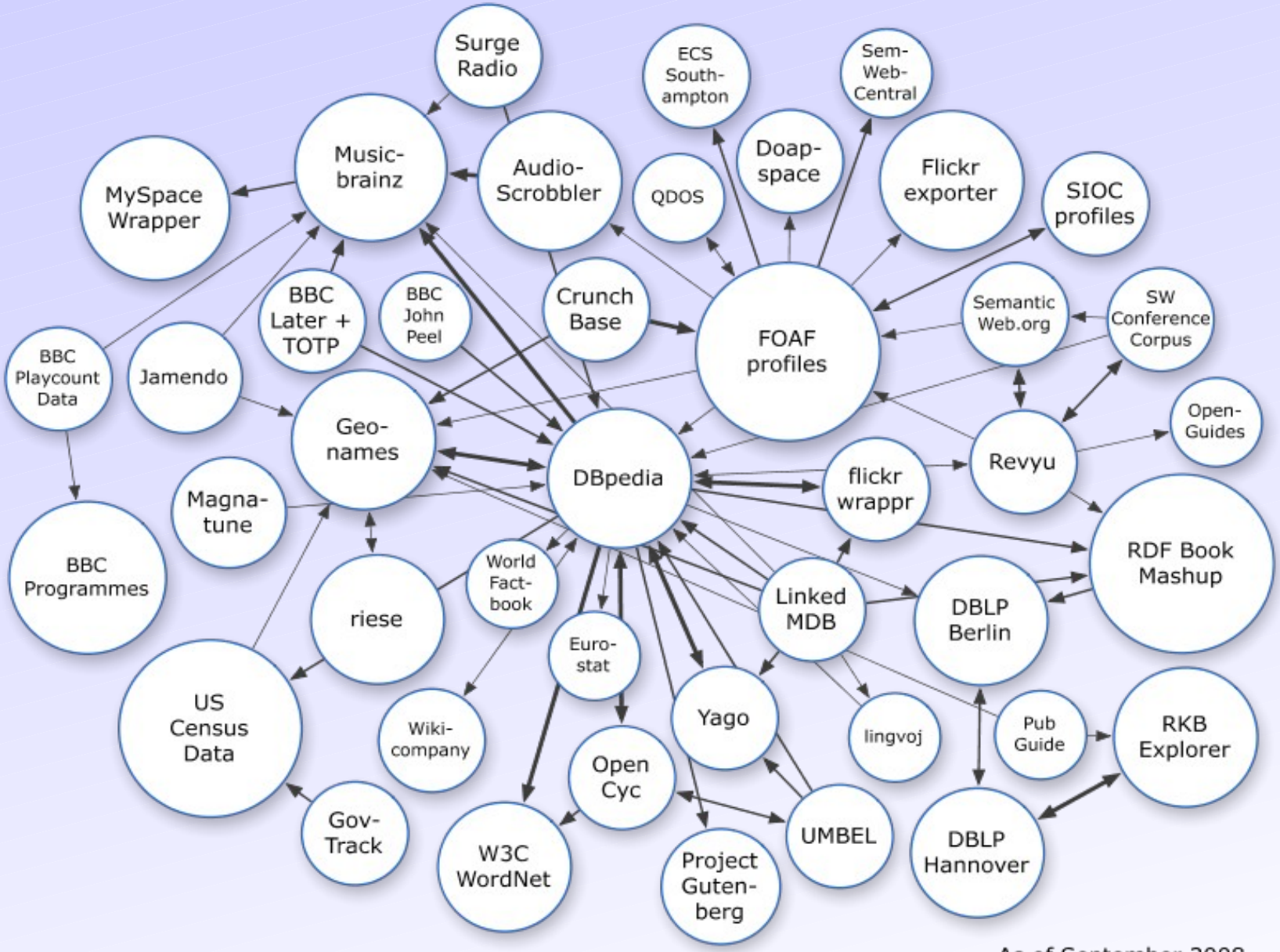
```
<http://sws.geonames.org/2759793>
  owl:sameAs <http://dbpedia.org/resource/Amsterdam>
  wgs84_pos:lat "52.3666667" ;
  wgs84_pos:long "4.8833333" ;
  geo:inCountry <http://www.geonames.org/countries/#NL> ;
 ...
```

Processors can switch automatically from one to the other…

The LOD "cloud", March 2008

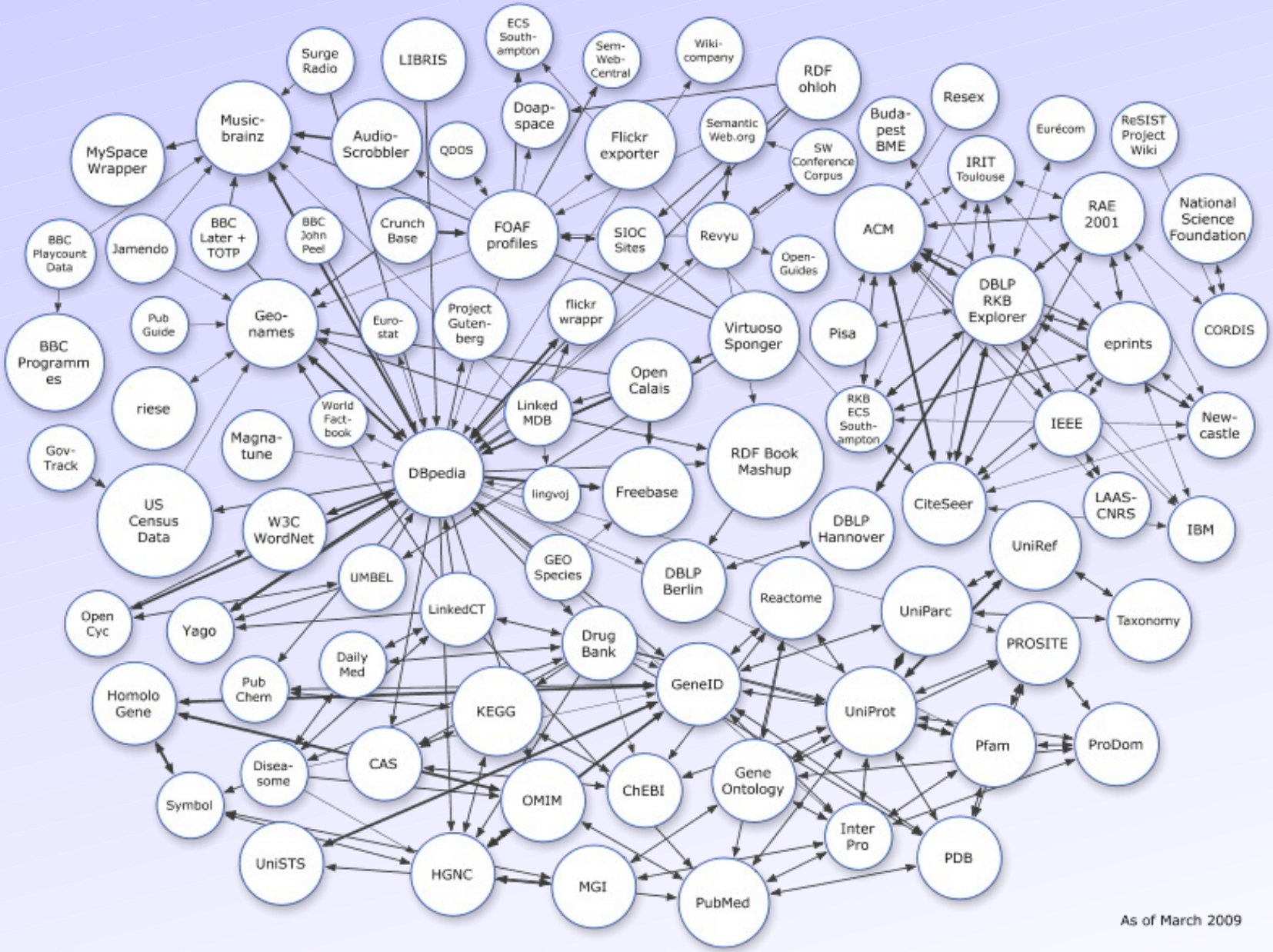


The LOD "cloud", September 2008



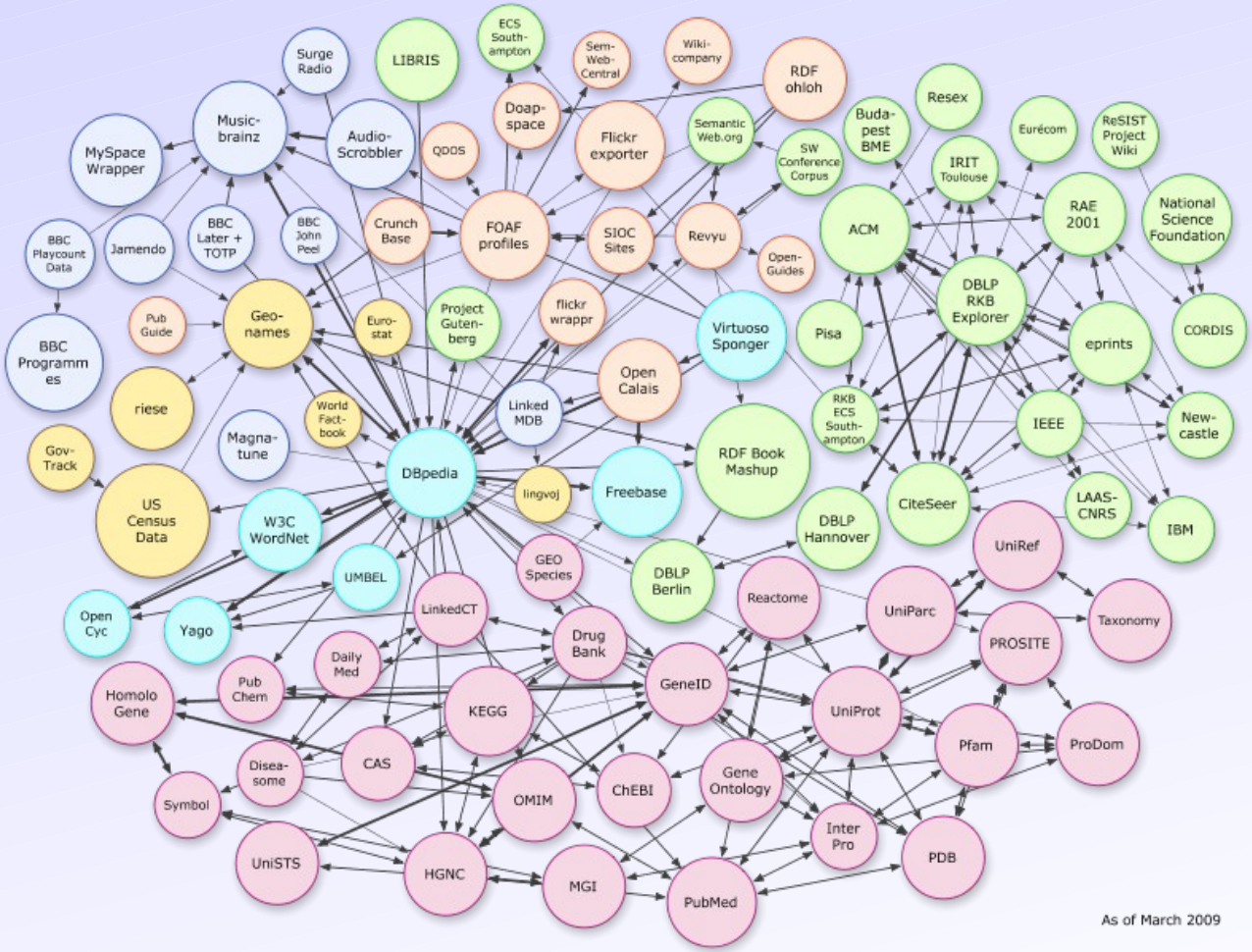
W3C Semantic Web

The LOD "cloud", March 2009



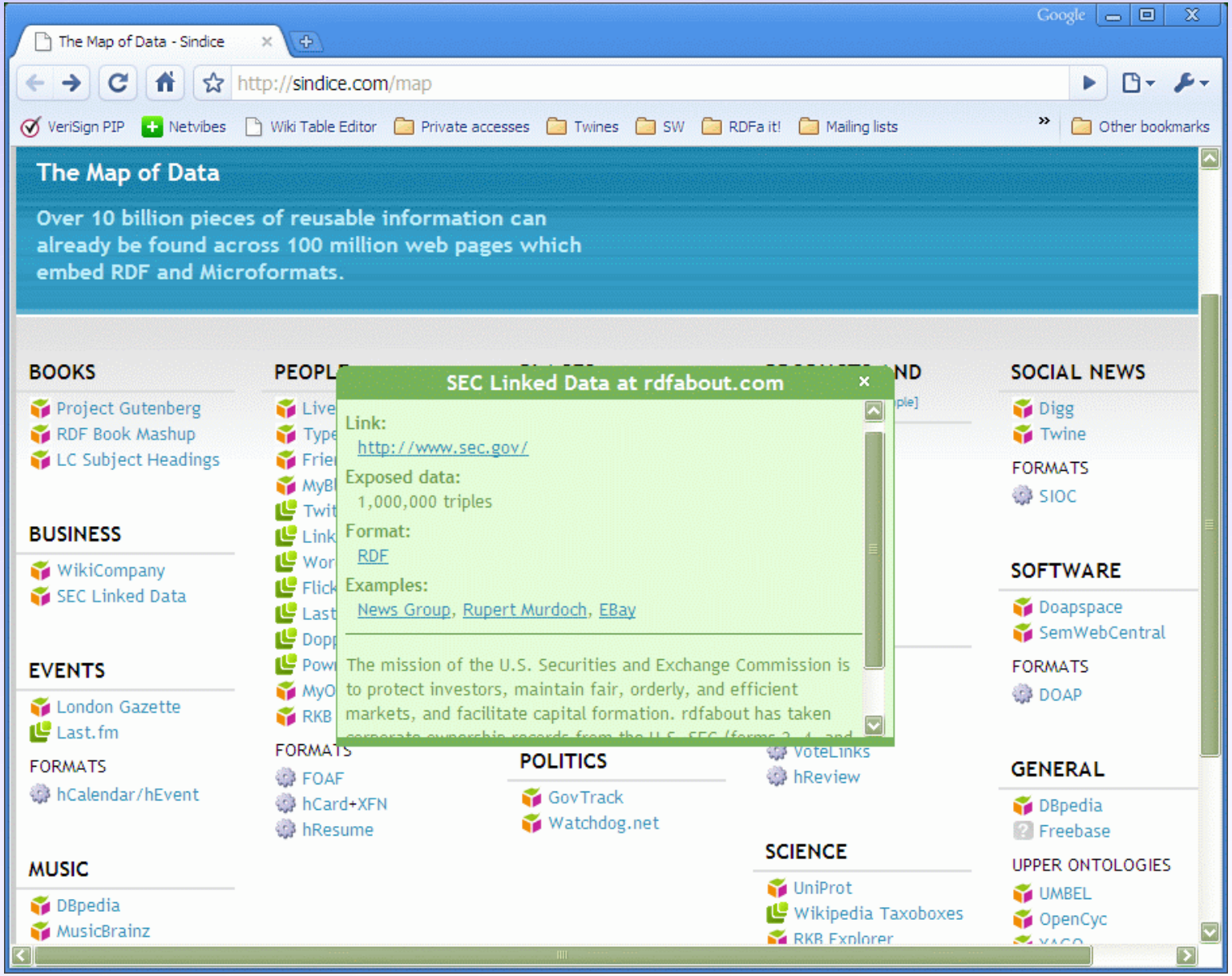
Application specific portions of the cloud

- Eg, “bio” related datasets
  - done, partially, by the “Linking Open Drug Data” task force of the HCLS IG at W3C



W3C Semantic Web

Another view of (RDF) data on the Web (Sindice)



Query RDF Data
(SPARQL)

RDF data access

- How do I *query* the RDF data?
  - e.g., how do I get to the DBpedia data?

Querying RDF graphs

- Remember the Jena idiom:

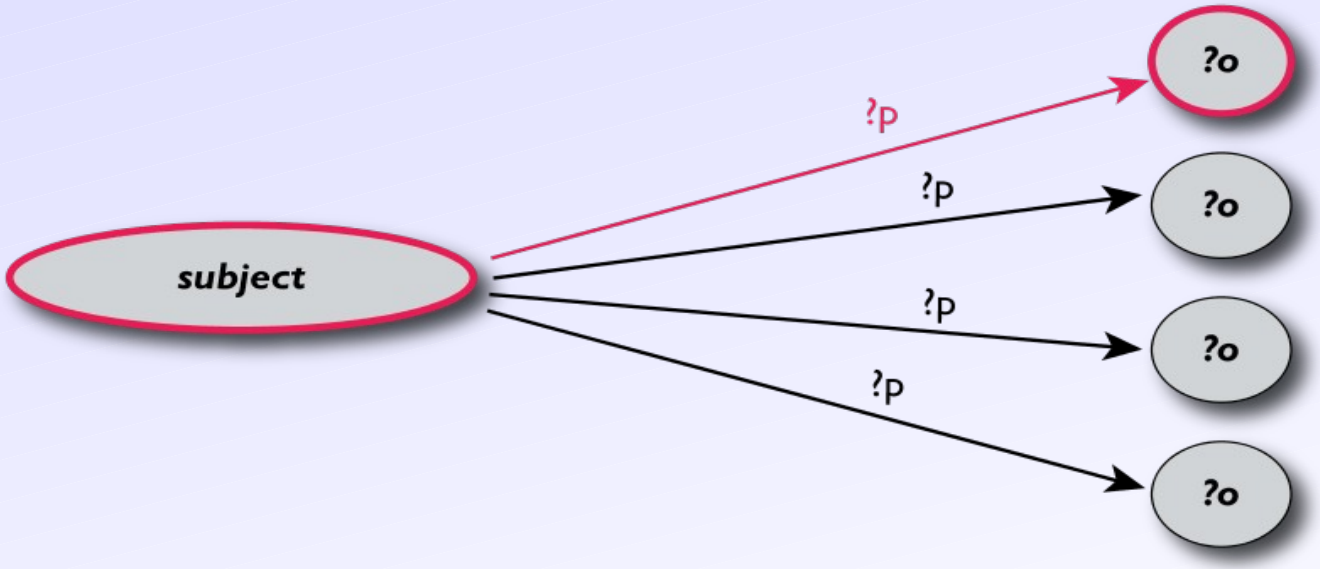
```
StmtIterator iter=model.listStatements(subject,null,null);
while(iter.hasNext()) {
    st = iter.next();
    p = st.getProperty(); o = st.getObject();
    do_something(p,o);
```

- In practice, more complex queries into the RDF data are necessary
  - something like: “give me the (a,b) pair of resources, for which there is an x such that (x parent a) and (b brother x) holds” (ie, return the uncles)
  - these rules may become quite complex
- The goal of SPARQL (Query Language for RDF)

Analyze the Jena example

```
StmtIterator iter=model.listStatements(subject,null,null);
while(iter.hasNext()) {
    st = iter.next();
    p = st.getProperty(); o = st.getObject();
    do_something(p,o);
```

- The `(subject,?p,?o)` is a *pattern* for what we are looking for (with `?p` and `?o` as "unknowns")



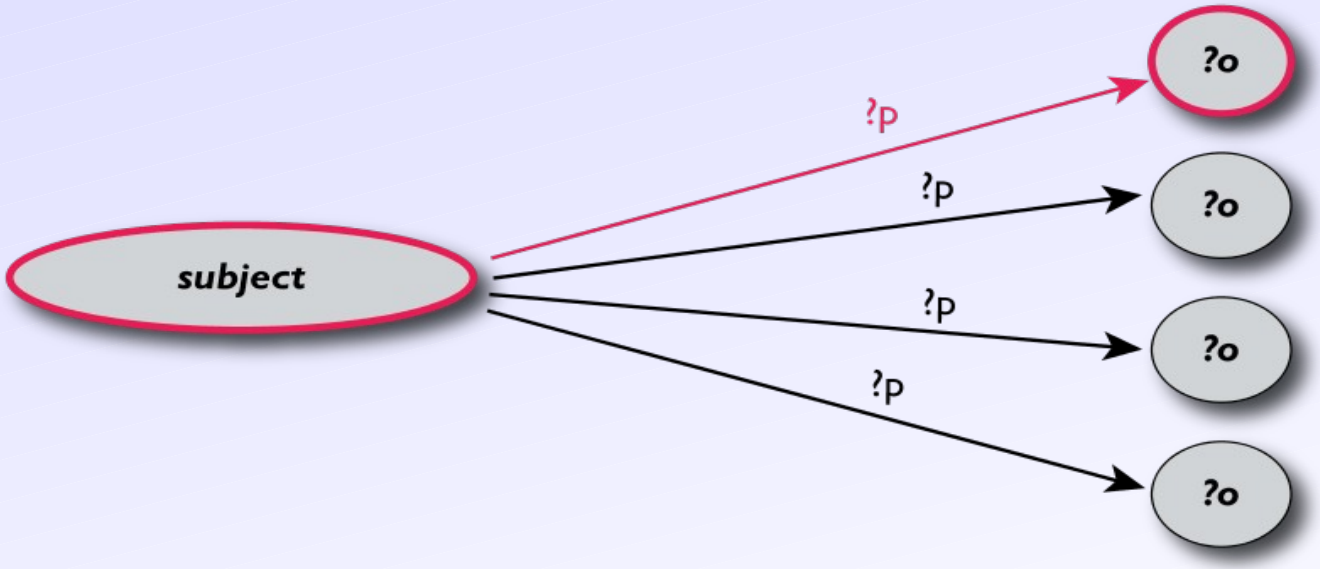
General: graph patterns

- The fundamental idea: use graph patterns
  - the pattern contains unbound symbols
  - by binding the symbols, subgraphs of the RDF graph are selected
  - if there is such a selection, the query returns bound resources

Our Jena example in SPARQL

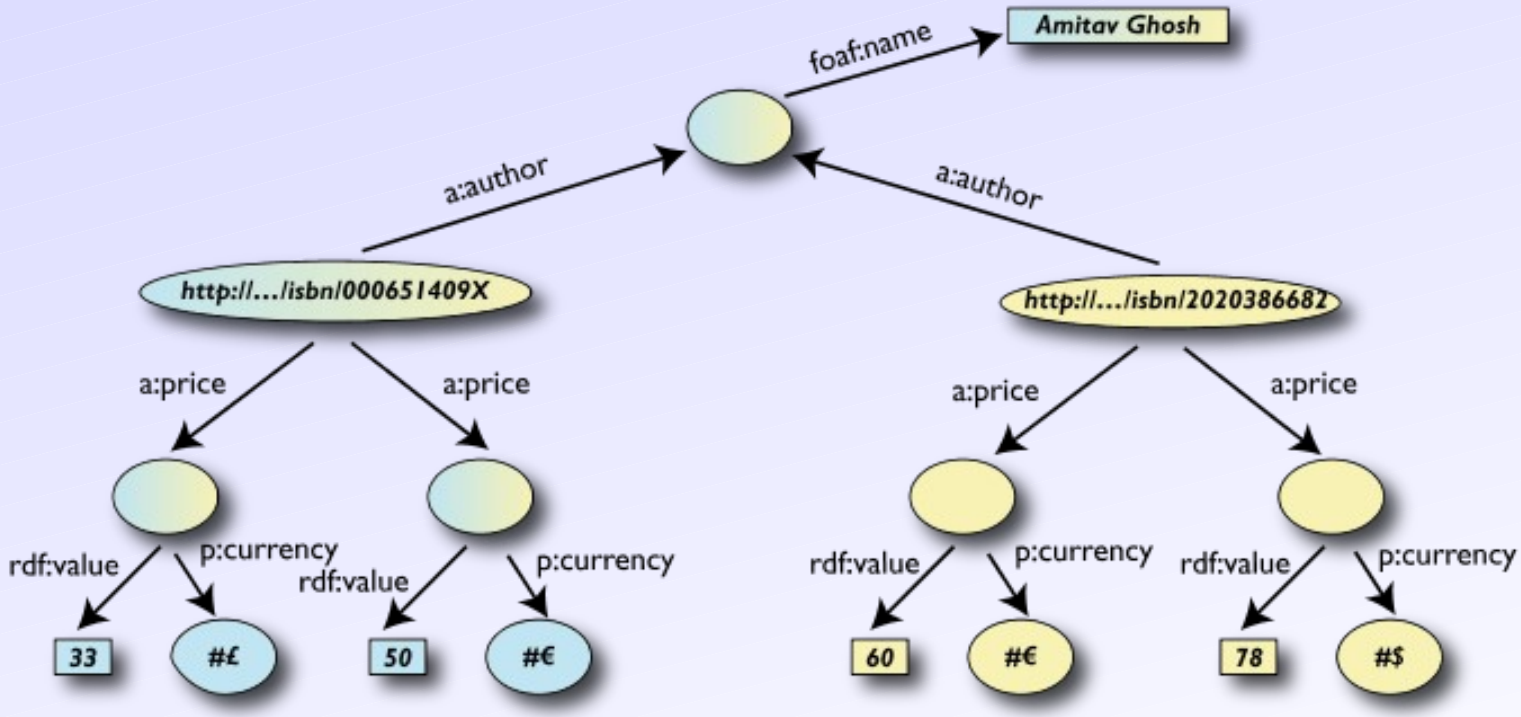
```
SELECT ?p ?o
WHERE {subject ?p ?o}
```

- The triples in **`WHERE`** define the graph pattern, with **`?p`** and **`?o`** "unbound" symbols
- The query returns *all* **`p`**,**`o`** pairs



Simple SPARQL example

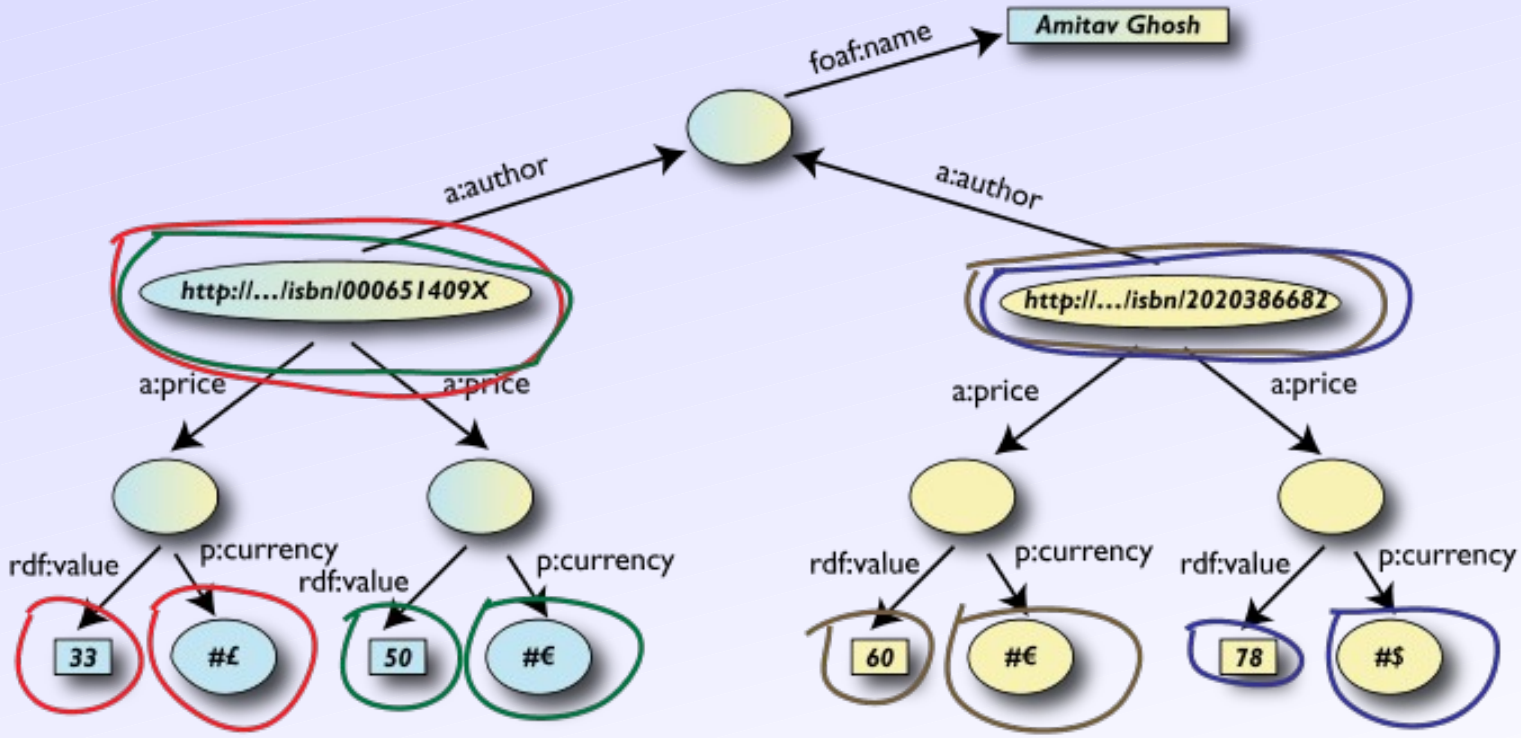
```
SELECT ?isbn ?price ?currency # note: not ?x!
WHERE {?isbn a:price ?x. ?x rdf:value ?price. ?x p:currency ?currency.}
```



Simple SPARQL example

```
SELECT ?isbn ?price ?currency # note: not ?x!
WHERE {?isbn a:price ?x. ?x rdf:value ?price. ?x p:currency ?currency.}
```

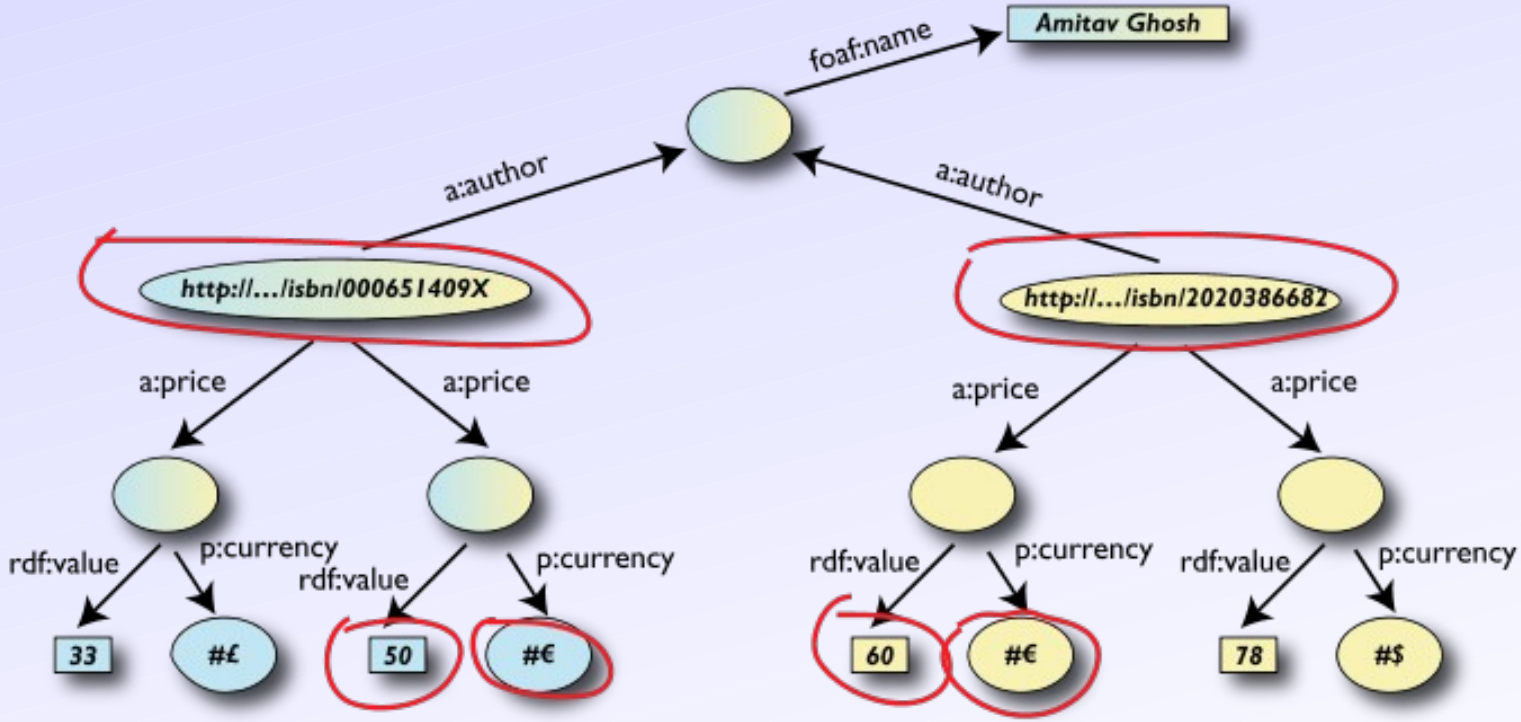
- Returns:
  [[<..49X>,33,£], [<..49X>,50,€], [<..6682>,60,€], [<..6682>,78,$]]



Pattern constraints

```
SELECT ?isbn ?price ?currency # note: not ?x!
WHERE { ?isbn a:price ?x. ?x rdf:value ?price. ?x p:currency ?currency.
        FILTER(?currency == € }
```

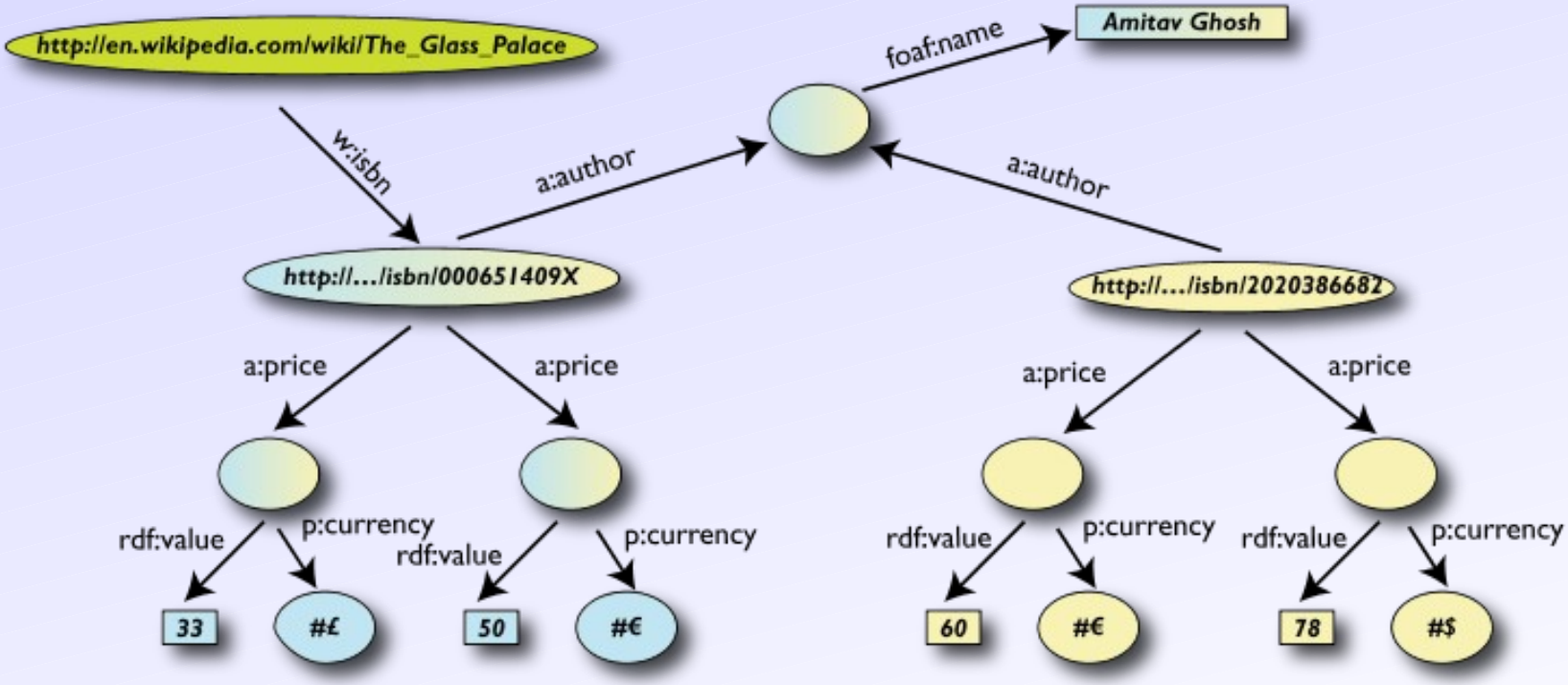
- Returns: [[<..409X>,50,€], [<..6682>,60,€]]



Optional pattern

```
SELECT ?isbn ?price ?currency ?wiki
WHERE { ?isbn a:price ?x. ?x rdf:value ?price. ?x p:currency ?currency.
        OPTIONAL ?wiki w:isbn ?isbn. }
```

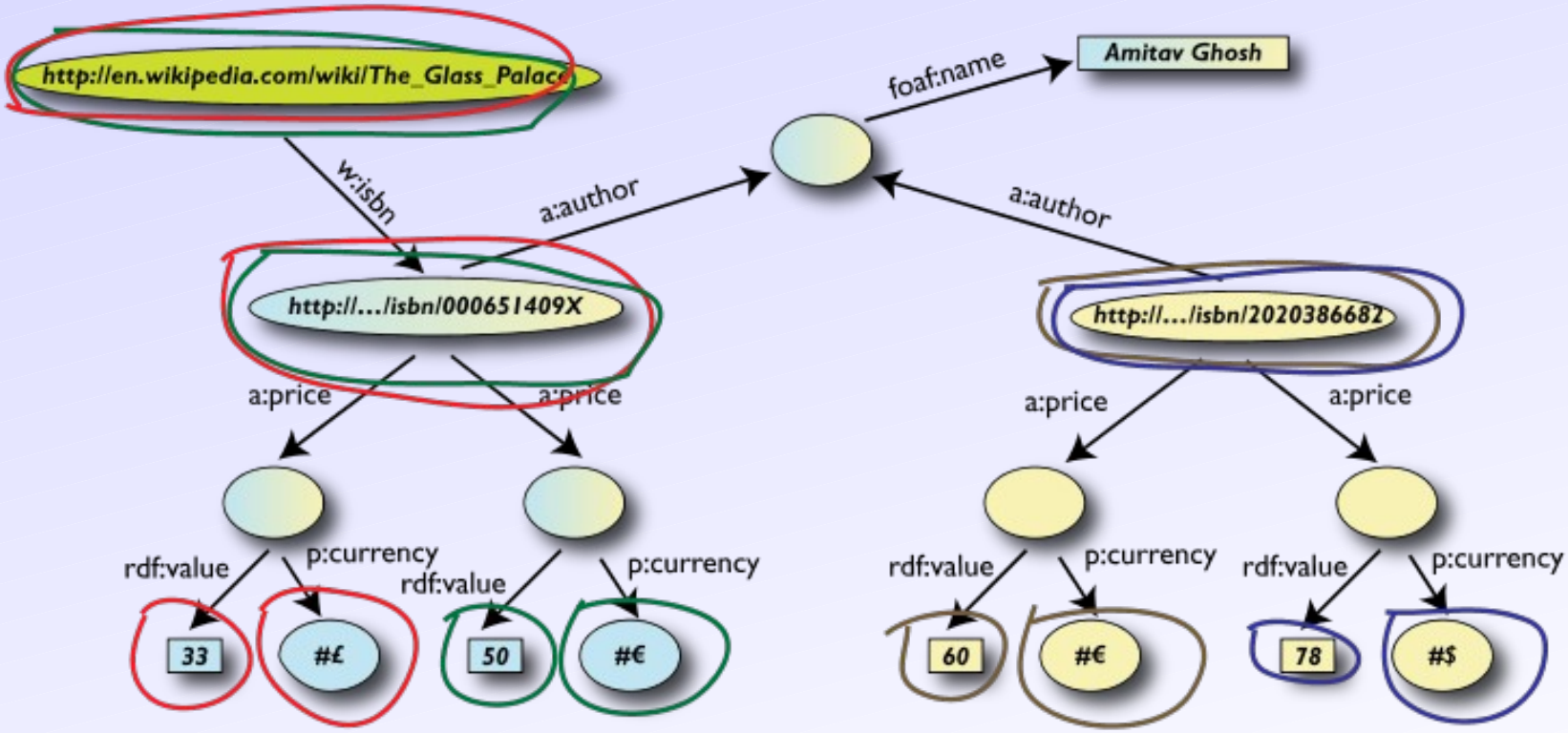
- Returns: [[<..49X>,33,£,<…Palace>], … , [<..6682>,78,$,]]



Optional pattern

```
SELECT ?isbn ?price ?currency ?wiki
WHERE { ?isbn a:price ?x. ?x rdf:value ?price. ?x p:currency ?currency.
        OPTIONAL ?wiki w:isbn ?isbn. }
```

- Returns: [[<..49X>,33,£,<…Palace>], … , [<..6682>,78,$,]]



Other SPARQL features

- Limit the number of returned results; remove duplicates, sort them, …
- Specify several data sources (via URI-s) within the query (essentially, a merge!)
- *Construct* a graph combining a separate pattern and the query results
- Use datatypes and/or language tags when matching a pattern

SPARQL usage in practice

- SPARQL is usually used over the network
  - separate documents define the protocol and the result format
    - SPARQL Protocol for RDF with HTTP and SOAP bindings
    - SPARQL results in XML or JSON formats
- Big datasets usually offer “SPARQL endpoints” using this protocol
  - typical example: SPARQL endpoint to DBpedia

Remote query/reply example

```
GET /qps?&query=SELECT+:…+WHERE:+… HTTP/1.1
User-Agent: my-sparql-client/0.0
Host: my.example

HTTP/1.1 200 OK
Server: my-sparql-server/0.0
Content-Type: application/sparql-results+xml

<?xml version="1.0" encoding="UTF-8"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#>
  <head>
    <variable name="a"/>
    ...
  </head>
  <results>
    <result ordered="false" distinct="false">
      <binding name="a"><uri>http:…</uri></binding>
      ...
    </result>
    <result> ... </result>
  </results>
</sparql>
```

The power of CONSTRUCT: "chaining" queries

```
CONSTRUCT {
    <http://dbpedia.org/resource/Amitav_Ghosh> ?p1 ?o1.
    ?s2 ?p2 <http://dbpedia.org/resource/Amitav_Ghosh>.
}
WHERE {
    <http://dbpedia.org/resource/Amitav_Ghosh> ?p1 ?o1.
    ?s2 ?p2 <http://dbpedia.org/resource/Amitav_Ghosh>.
}
```

- SPARQL endpoint
- returns RDF/XML

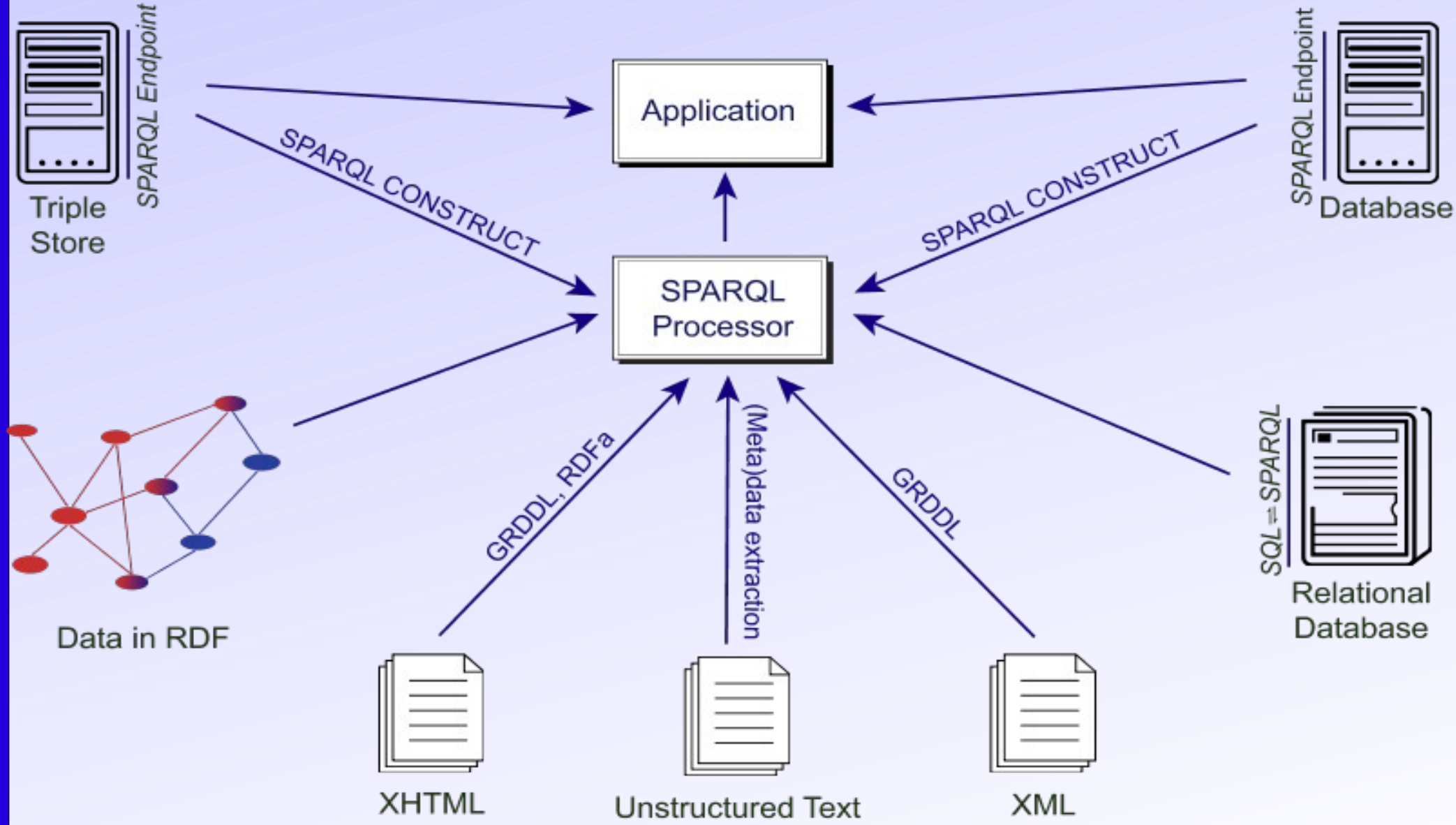
```
SELECT *
FROM <http://dbpedia.org/sparql/?query=CONSTRUCT+%7B++...>
WHERE {
  ?author_of dbpedia:author res:Amitav_Ghosh.
  res:Amitav_Ghosh  dbpedia:reference ?homepage;
                    rdf:type          ?type;
                    foaf:name         ?foaf_name.
  FILTER regex(str(?type),"foaf")
}
```

- Data reused in another query…

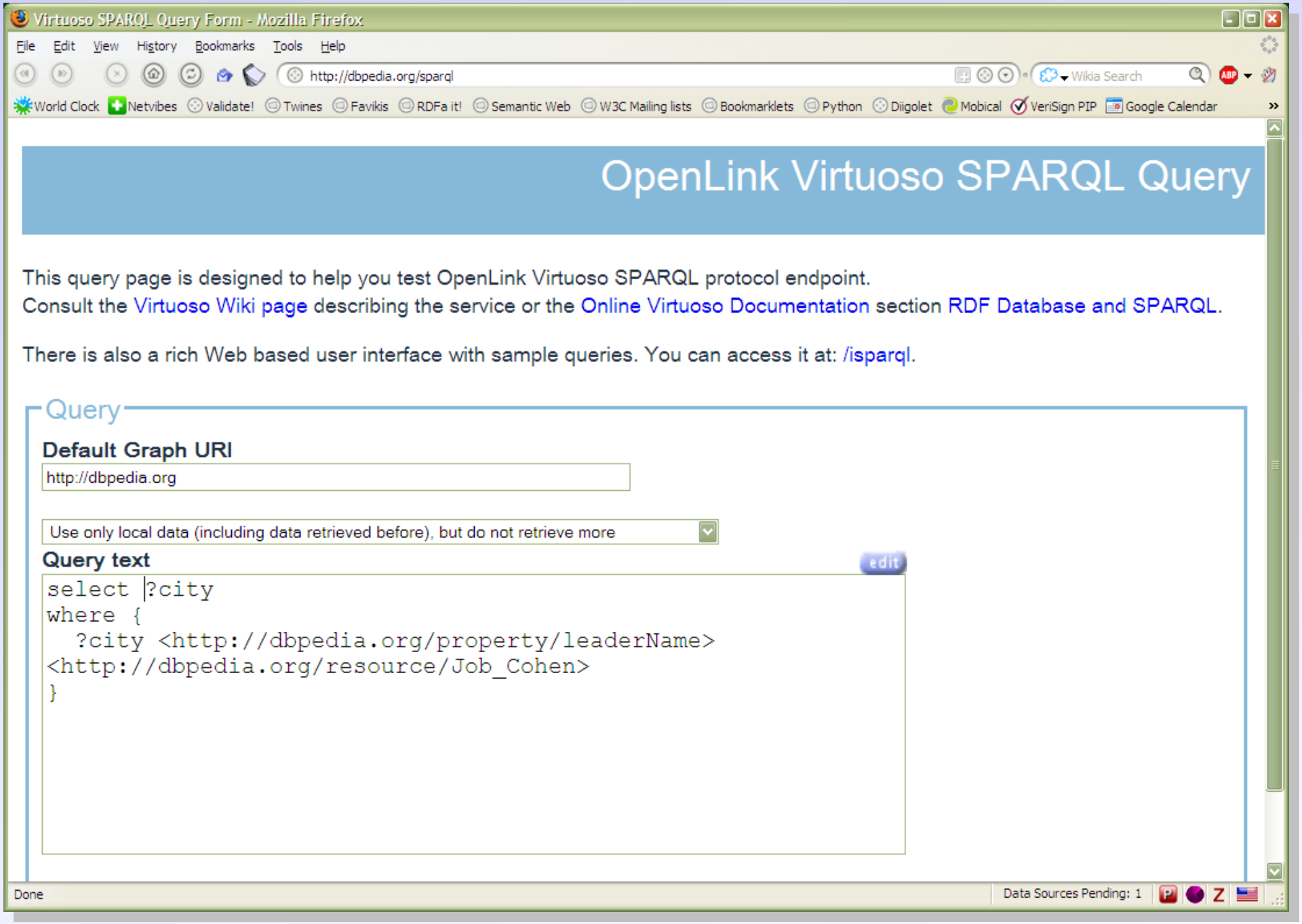
A word of warning on SPARQL…

- Some features are missing
  - control and/or description on the entailment regimes of the triple store (eg, RDFS …)
  - modify the triple store
  - querying collections or containers may be complicated
  - no functions for sum, average, min, max, …
  - ways of aggregating queries
  - …
- Delayed for a next version…
  - work on this update has started in February 2009

SPARQL as a unifying point



SPARQL-ing DBpedia



W3C Semantic Web

Ontologies
(OWL)

Ontologies

- RDFS is useful, but does not solve all possible requirements
- Complex applications may want more possibilities:
  - characterization of properties
  - identification of objects with different URI-s
  - disjointness or equivalence of classes
  - construct classes, not only name them
  - more complex classification schemes
  - can a program reason about some terms? E.g.:
    - “if «Person» resources «A» and «B» have the same «`foaf:email`» property, then «A» and «B» are identical”
  - etc.

Ontologies (cont.)

- The term *ontologies* is used in this respect:

"defines the concepts and relationships used to describe and represent an area of knowledge"

- Ie, there is a need for Web Ontology Language(s)
  - RDFS can be considered as a simple ontology language
- Languages should be a compromise between
  - rich semantics for meaningful applications
  - feasibility, implementability

Web Ontology Language = OWL

- OWL is an extra layer, a bit like RDF Schemas
  - own namespace, own terms
  - it relies on RDF Schemas
- It is a separate recommendation
  - actually… there is a 2004 version of OWL (“OWL 1”)
  - and there is an update (“OWL 2”) that should be finalized in 2009
  - this presentation is based on OWL 2
    - in what follows, “OWL 2” will mean this is an OWL 2 feature
    - everything else is valid *both* for OWL and OWL 2

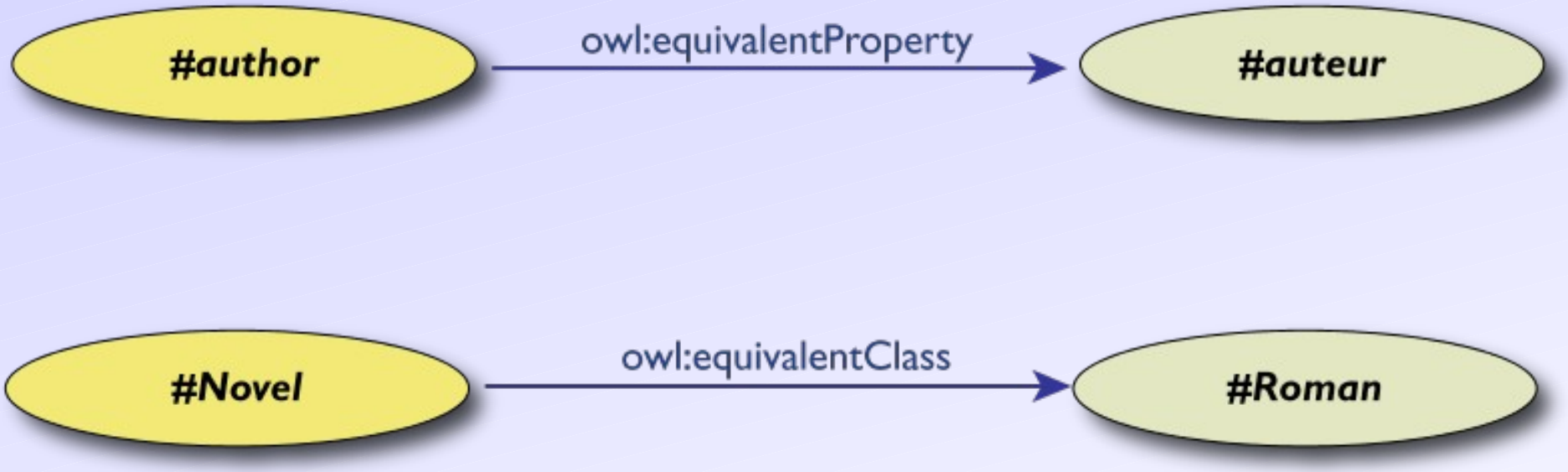
OWL is complex…

- OWL is a large set of additional terms
- We will not cover the whole thing here…

Term equivalences

- For classes:
  - **`owl:equivalentClass`**: two classes have the *same* individuals
  - **`owl:disjointWith`**: no individuals in common
- For properties:
  - **`owl:equivalentProperty`**
    - remember the **`a:author`** vs. **`f:auteur`**?
  - **`owl:propertyDisjointWith`**
- For individuals:
  - **`owl:sameAs`**: two URIs refer to the same concept ("individual")
  - **`owl:differentFrom`**: negation of **`owl:sameAs`**

Other example: connecting to French



Typical usage of `owl:sameAs`

- Linking our example of Amsterdam from one data set (DBpedia) to the other (Geonames):

```
<http://dbpedia.org/resource/Amsterdam>
  owl.sameAs <http://sws.geonames.org/2759793>;
```

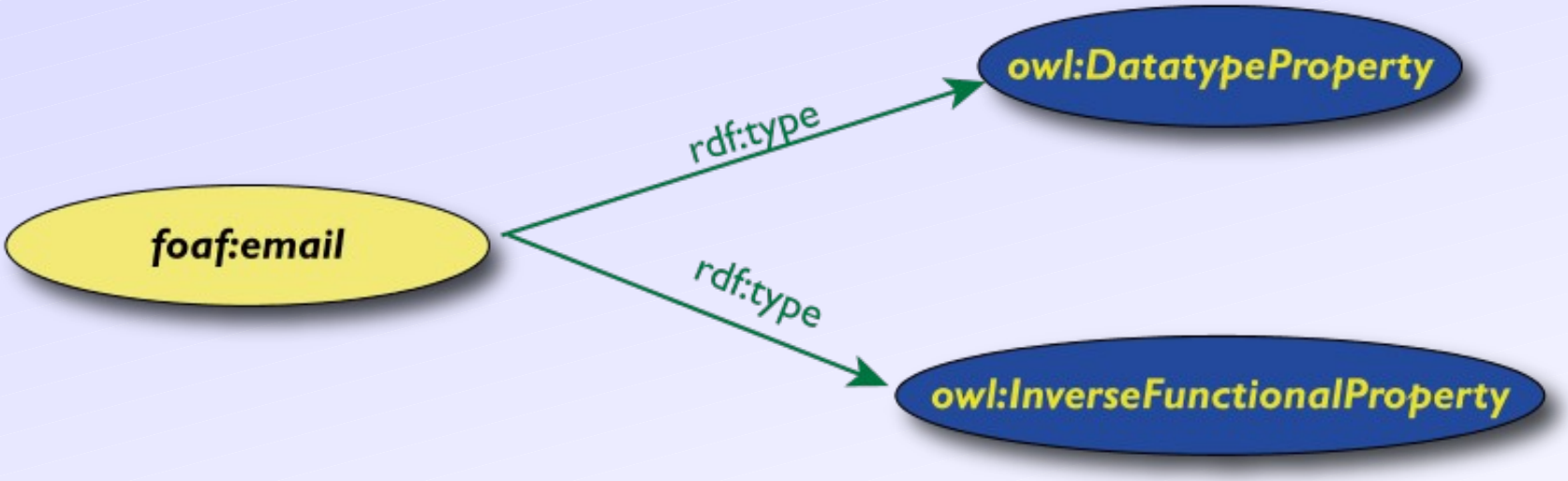
- This is the main mechanism of “Linking” in the Linking Open Data project

Property characterization

- In OWL, one can characterize the behaviour of properties (symmetric, transitive, functional, inverse functional…)
- OWL also separates *data* and *object* properties
  - “datatype property” means that its range are typed literals

Characterization example

- "`foaf:email`" may be defined as "inverse functional"
  - i.e., two different subjects cannot have identical objects



What this means is…

- If the following holds in our triples:

```
:email rdf:type owl:InverseFunctionalProperty.
<A> :email "mailto:a@b.c".
<B> :email "mailto:a@b.c".
```

then, processed through OWL, the following holds, too:

```
<A> owl:sameAs <B>.
```

- I.e., *new relationships* were discovered again (beyond what RDFS could do)

Other property characterizations

- In OWL 2 properties may also be characterized as reflexive or irreflexive
- There may be an inverse relationship among properties, eg:

```
<somebook> ex:author <somebody>.
ex:author owl:inverseOf ex:authorOf.
```

yields, in OWL:

```
<somebody> ex:authorOf <somebook>.
```

Property chains (OWL 2)

- Properties, when applied one after the other, may be subsumed by yet another one:
  - "if a person «P» was born in city «A» and «A» is in country «B» then «P» was born in country «B»"
  - more formally:

```
ex:born_in_country owl:propertyChainAxiom
            (ex:born_in_city ex:city_in_country).
```

- More than two constituents can be used
- There are some restrictions to avoid "circular" specifications

Keys (OWL 2)

- Inverse functional properties are important for identification of individuals
  - think of the email examples
- But… identification based on *one* property may not be enough

Keys (OWL 2)

"if two persons have the same emails *and* the same homepages then they are identical"

- Identification is based on the identical values of *two* properties
- The rule applies to persons only

Previous rule in OWL 2

```
:Person rdf:type owl:Class;
   owl:hasKey (:email :homepage) .
```

What it means is…

If:

```
<A> rdf:type :Person ;
    :email    "mailto:a@b.c";
    :homepage "http://www.ex.org".

<B> rdf:type :Person ;
    :email    "mailto:a@b.c";
    :homepage "http://www.ex.org".
```

then, processed through OWL 2, the following holds, too:

```
<A> owl:sameAs <B>.
```

Classes in OWL

- In RDFS, you can subclass existing classes… that's all
- In OWL, you can *construct* classes from existing ones:
  - enumerate its content
  - through intersection, union, complement
  - etc

Classes in OWL (cont)

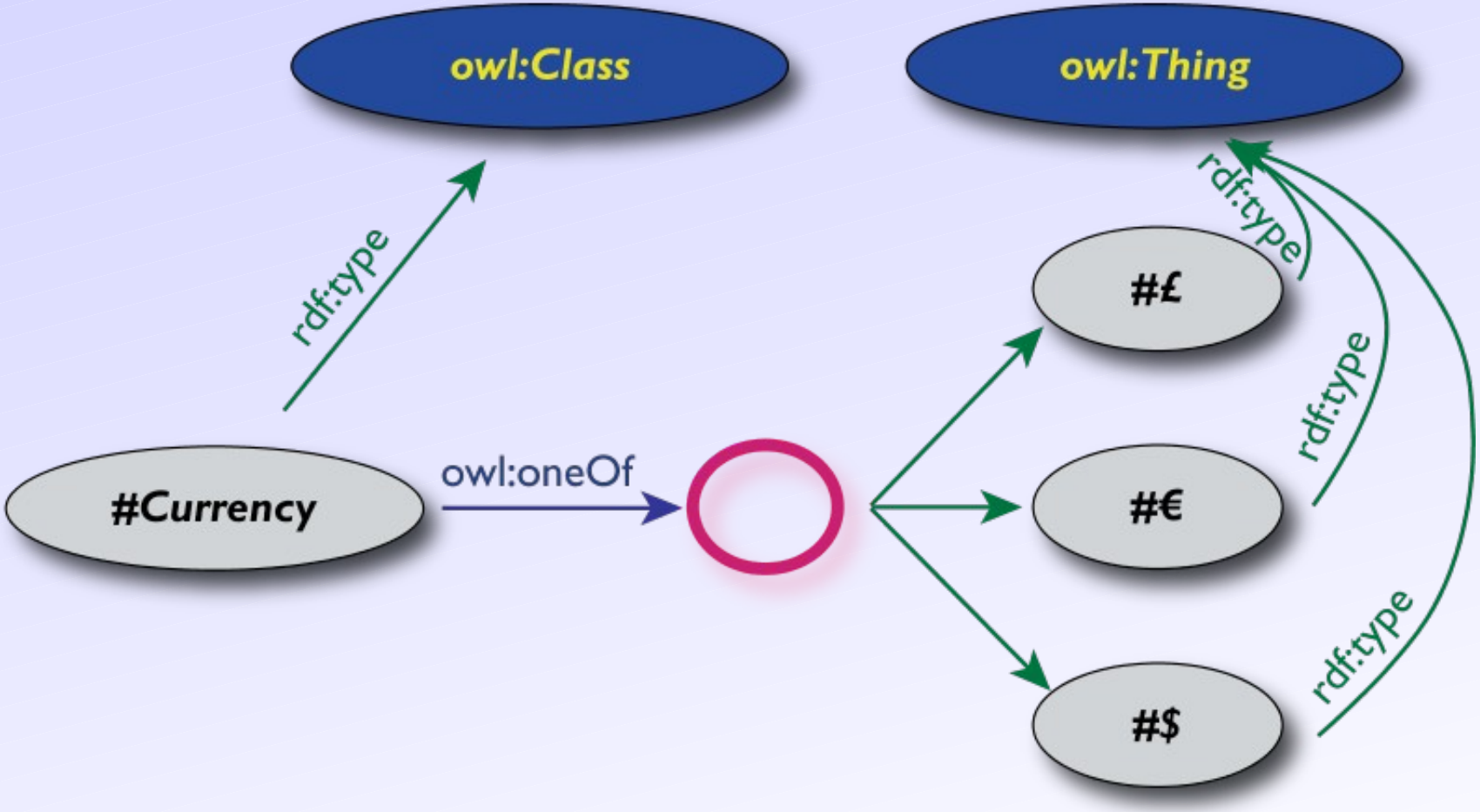
- OWL makes a stronger conceptual distinction between *classes* and *individuals*
  - there is a separate term for **owl:Class**, to make the difference
  - individuals are separated into a special class called **owl:Thing**
- Eg, a precise classification would be:

```
ex:Person rdf:type owl:Class.

<uri-for-Amitav-Ghosh>
    rdf:type owl:Thing;
    rdf:type owl:Person .
```

OWL classes can be "enumerated"

- The OWL solution, where possible content is explicitly listed:



Same serialized

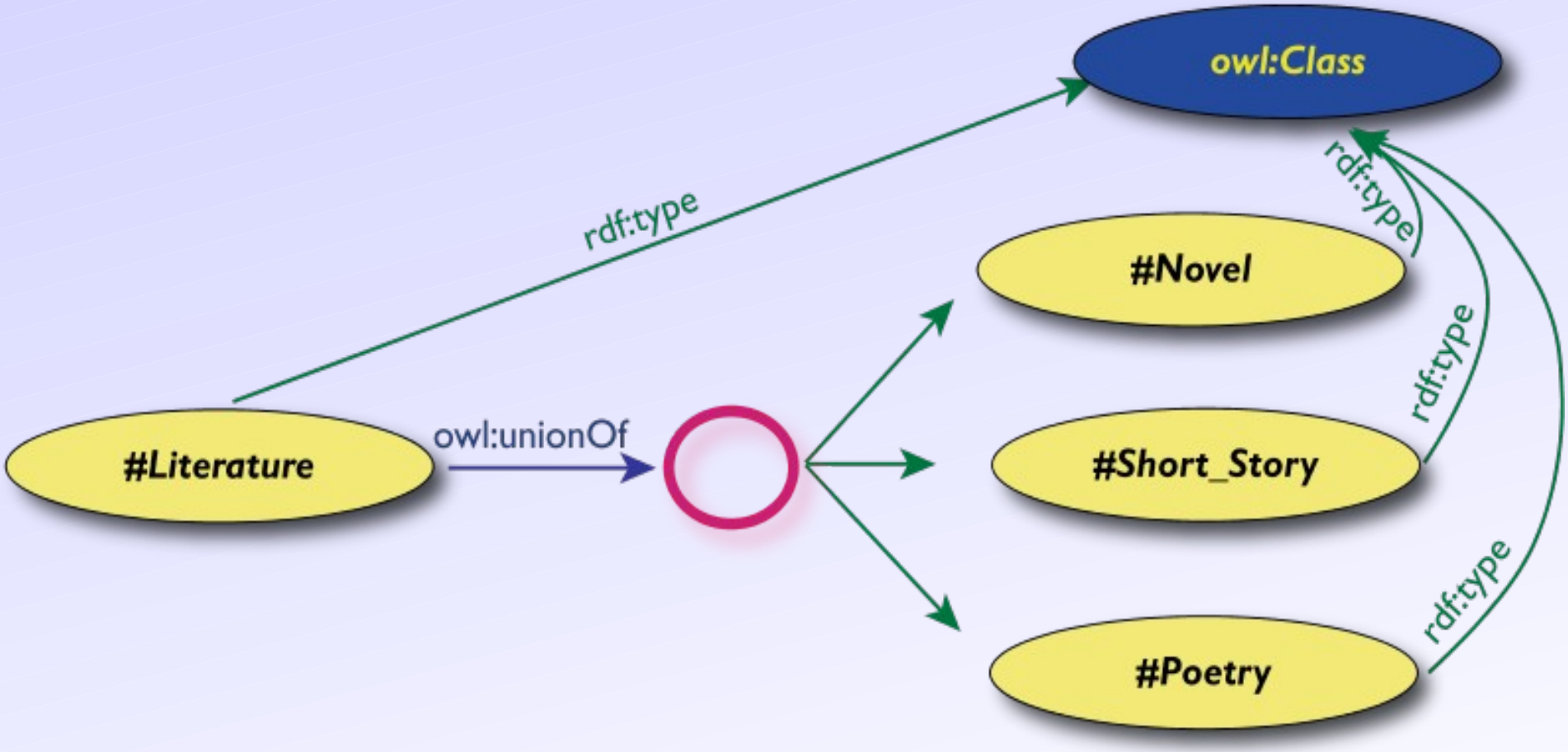
```
<owl:Class rdf:ID="Currency">
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:ID="£"/>
    <owl:Thing rdf:ID="€"/>
    <owl:Thing rdf:ID="$"/>
    …
  </owl:oneOf>
</owl:Class>
```

```
:£ rdf:type owl:Thing.
:€ rdf:type owl:Thing.
:$ rdf:type owl:Thing.
:Currency
    rdf:type owl:Class;
    owl:oneOf (:€ :£ :$).
```

- I.e., the class consists of *exactly* of those individuals

Union of classes

- Essentially, like a set-theoretical union:



Same serialized

```
<owl:Class rdf:ID="Literature">
   <owl:unionOf rdf:parseType="Collection">
       <owl:Class rdf:about="#Novel"/>
       <owl:Class rdf:about="#Short_Story"/>
       <owl:Class rdf:about="#Poetry"/>
       …
   </owl:unionOf>
</owl:Class>
```

```
:Novel            rdf:type owl:Class.
:Short_Story      rdf:type owl:Class.
:Poetry           rdf:type owl:Class.
:Literature rdf:type owlClass;
   owl:unionOf (:Novel :Short_Story :Poetry).
```

- Other possibilities: **`complementOf`**, **`intersectionOf`**, …

For example…

If:

```
:Novel            rdf:type owl:Class.
:Short_Story      rdf:type owl:Class.
:Poetry           rdf:type owl:Class.
:Literature rdf:type owlClass;
   owl:unionOf (:Novel :Short_Story :Poetry).

<myWork> rdf:type :Novel .
```

then the following holds, too:

```
<myWork> rdf:type :Literature .
```

It can be a bit more complicated…

If:

```
:Novel              rdf:type owl:Class.
:Short_Story        rdf:type owl:Class.
:Poetry             rdf:type owl:Class.
:Literature rdf:type owlClass;
   owl:unionOf (:Novel :Short_Story :Poetry).

fr:Roman owl:equivalentClass :Novel .

<myWork> rdf:type fr:Roman .
```

then, through the combination of different terms, the following still holds:

```
<myWork> rdf:type :Literature .
```

What we have so far…

- The OWL features listed so far are already fairly powerful
- E.g., various databases can be linked via **`owl:sameAs`**, functional or inverse functional properties, etc.
- Many inferred relationship can be found using a traditional rule engine

However… that may not be enough

- Very large vocabularies might require even more complex features
  - typical usage example: definition of all concepts in a health care environment
  - some major issues
    - the way classes (i.e., “concepts”) are defined
    - handling of datatypes
- OWL includes those extra features but… the inference engines become (much) more complex 

Property value restrictions

- Classes are created by *restricting* the property values on a (super)class
- For example: how would I characterize a “listed price”?
  - it is a price (which may be a general term), but one that is given in one of the “allowed” currencies (say, €, £, or $)
  - more formally:
    - the value of “`p:currency`”, when applied to a resource on listed price, *must* take one of those values…
    - …thereby defining the class of “listed price”

Restrictions formally

- Defines a class of type **`owl:Restriction`** with a
  - reference to the property that is constrained
  - definition of the constraint itself
- One can, e.g., subclass from this node when defining a particular class

```
:Listed_Price rdfs:subClassOf [
      rdf:type              owl:Restriction;
      owl:onProperty        p:currency;
      owl:allValuesFrom     :Currency.
    ].
```

Possible usage…

If:

```
:Listed_Price rdfs:subClassOf [
      rdf:type              owl:Restriction;
      owl:onProperty        p:currency;
      owl:allValuesFrom     :Currency.
    ].

:Price rdf:type :Listed_Price .

:Price p:currency <someCurrency> .
```

then the following holds:

```
<someCurrency> rdf:type :Currency .
```

Other restrictions

- **`allValuesFrom`** could be replaced by:
  - **`someValuesFrom`**
    - e.g., I could have said: there should be a price given in *at least one* of those currencies
  - **`hasValue`**, when restricted to *one specific value*
  - **`hasSelf`** (in OWL 2), for local reflexivity

Similar concept: cardinality restriction

- In a property restriction, the goal was to restrict the possible *values* of a property
- In a cardinality restriction, the *number* of relations with that property is restricted
  - "a book being on offer" could be characterized as having at *least one price property* (i.e., the price of the book has been established)

Cardinality restriction

```
:Book_on_sale rdfs:subClassOf [
      rdf:type                owl:Restriction;
      owl:onProperty          p:price;
      owl:minCardinality      "1"^^xsd:integer.
   ].
```

- could also be "**owl:cardinality**" or "**owl:maxCardinality**"

Qualified Cardinality Restriction (OWL 2)

- Combining cardinality and the “all value” restriction
  - “there should be *exactly* two listed price tags with currency value”

```
:Listed_Price rdf:type owl:Class;
    rdfs:subClassOf [
      rdf:type                    owl:Restriction;
      owl:onProperty              p:currency;
      owl:onClass                 :Currency;
      owl:qualifiedCardinality    "2"^^xsd:integer.
   ].
```

Datatypes in OWL

- RDF Literals can have a datatypes, OWL adopts those
- But more complex vocabularies require datatypes “restrictions”; eg, numeric intervals
  - “I am interested in a price range between €5 and €15”
- RDF allows any URI to be used as datatypes
  - ie, one could use XML Schemas to define, eg, numeric intervals
  - but it is very complex, and reasoners would have to understand a whole different syntax

Datatype facets (OWL 2)

- For each datatype, XML Schema defines possible restriction “facets”: min and max for numeric types, length for strings, etc
- OWL uses these facets to define *datatype ranges* for its own use

Definition of a numeric interval in OWL 2

```
:AllowedPrice rdf:type rdfs:Datatype;
    owl:onDatatype xsd:float;
    owl:withRestriction (
        [ xsd:minInclusive 5.0 ]
        [ xsd:maxExclusive 15.0 ]
    ) .
```

- The possible facets depend on the datatype: **`xsd:pattern`**, **`xsd:length`**, **`xsd:maxLength`**, ...

Typical usage of OWL 2 datatype restrictions

```
:Affordable_book rdf:type owl:Class;
    rdfs:subClassOf [
      rdf:type                    owl:Restriction;
      owl:onProperty              p:price_value;
      owl:allValuesFrom [
           rdf:type rdfs:Datatype;
           owl:onDatatype xsd:float;
           owl:withRestriction (
                [ xsd:minInclusive 5.0 ]
                [ xsd:maxExclusive 15.0 ]
           )
      ]
   ].
```

ie: an affordable book has a price between 5.0 and 15.0

But: OWL is hard!

- The combination of class constructions with various restrictions is extremely powerful
- What we have so far follows the same logic as before
  - extend the basic RDF and RDFS possibilities with new features
  - define their semantics, ie, what they "mean" in terms of relationships
  - expect to infer new relationships based on those
- However… a full inference procedure is hard 
  - not implementable with simple rule engines, for example

OWL "species"

- OWL species comes to the fore:
  - restricting *which* terms can be used and *under what circumstances (restrictions)*
  - if one abides to those restrictions, then simpler inference engines can be used
- They reflect compromises: expressibility vs. implementability
  - mathematically: what is the formal semantics is used to define the terms?

OWL Full ("RDF based semantics")

- No constraints on any of the constructs
  - `owl:Class` is equivalent to `rdfs:Class`
  - `owl:Thing` is equivalent to `rdfs:Resource`
  - this means that:
    - Class can also be an individual, a URI can denote a property as well as a Class
      - e.g., it is possible to talk about class of classes, etc.
    - one can make statements on RDFS constructs (e.g., declare `rdf:type` to be functional…)
    - etc.
- Extension of RDFS in all respects
- But: *an OWL Full ontology may be, eg, inconsistent!*

Example for a possible OWL Full problem

- Here is a syntactically valid but inconsistent ontology:

```
:A rdf:type owl:Class;
   owl:equivalentClass [
      rdf:type           owl:Restriction;
      owl:onProperty     rdf:type;
      owl:allValuesFrom :B.
   ].
:B rdf:type owl:Class;
   owl:complementOf :A.

:C rdf:type :A .
```

if **C** is of type **A** then it must be in **B**, but then it is in the complement of **A**, ie, it is not of type **A**…

OWL Full usage

- Nevertheless OWL Full can be very useful
  - it gives a generic framework to *express* many things
- Some application just need to express and interchange terms (with possible scruffiness)
- Applications may control what terms are used and how
  - in fact, they may define their own sub-language via, eg, a vocabulary
  - thereby ensuring a manageable inference procedure

OWL DL ("direct semantics")

- A number of restrictions are defined
  - classes, individuals, object and datatype properties, etc, are fairly strictly separated
  - RDFS and OWL terms are reserved
    - no statements on RDFS and OWL resources
  - the values of user's object properties must be individuals
    - i.e., properties are really used to create relationships *between individuals*
  - no characterization of *datatype* properties
  - …
- But: *well known inference algorithms exist!*

Examples for restrictions

- The following is not “legal” OWL DL:

```
<q> rdf:type <A>.          # A is a class, q is an individual

<r> rdf:type <q>.          # q cannot be used for a class, too

<A> ex:something <B>.      # properties are for individuals only

<q> ex:something <s>.      # same property cannot be used as
<p> ex:something “54”.     # same property cannot be used as
```

Example for OWL 2 conceptual restrictions

- In OWL 2 DL is a bit more relaxed
  - same symbol may be used both for a class and an instance
  - but not all “natural” inferences can be drawn in OWL 2 DL; ie, although the following is *valid*:

```
q rdf:type A.          # A is a class, q is an individual
A owl:sameAs B.        # A and B are equals as individuals
```

when using OWL 2 DL, this does *not* yield

```
q rdf:type B.
```

"DL" stands for "Description Logic"

- An area in knowledge representation
  - a special type of "structured" First Order Logic (logic with safety guards…)
  - formalism based on "concepts" (i.e., classes), "roles" (i.e., properties), and "individuals"
  - based on model theoretic semantics (like RDF, RDFS, and OWL)
- There are several variants of Description Logic
  - OWL DL are embodiments of specific Description Logics
    - for connoisseurs: OWL 2 DL ≈ $\mathcal{SROIQ}$(D)
  - some major differences: *usage of URI-s*, reference to XML Schema datatypes, …

"Description Logic" (cont.)

- Traditional DL has its own terminology:
  - named objects or concepts ⇔ definition of classes, relationships among classes
  - roles ⇔ properties
  - (terminological) axioms ⇔ subclass and subproperty relationships
  - facts or assertions ⇔ statements on individuals (**`owl:Thing`**-s)
- There is also a compact mathematical notation for axioms, assertions, etc:
  - $\mathtt{Literature} \equiv \mathtt{Novel} \sqcup \mathtt{Short\_Story} \sqcup \mathtt{Poetry}$
  - $\mathtt{Listed\_Price} \sqsubseteq \forall \mathtt{currency.Currencies}$
- You may see these in papers, books…

OWL DL usage

- Abiding to the restrictions means that very large ontologies can be developed that require precise procedures
  - eg, in the medical domain, biological research, energy industry, financial services (eg, XBRL), etc
  - the number of classes and properties described this way can go up to the many thousands
- OWL DL has become a language of choice to define and manage formal ontologies in general
  - even if their usage is not necessarily on the Web

OWL 2 also defines “profiles”

- Further restrictions on how terms can be used and what inferences can be expected
- The semantic approaches (“species”) are identical, but restrictions may ensure even more manageable implementations

OWL 2 profiles

- Classification and instance queries in polynomial time: *OWL-EL*
- Implementable on top of conventional relational database engines: *OWL-QL*
- Implementable on top of traditional rule engines: *OWL-RL*

An example: OWL-RL

- Goal: to be implementable through rule engines
- Usage follows a similar approach to RDFS:
  1) merge the ontology and the instance data into an RDF graph
  2) use the rule engine to add new triples (as long as it is possible)
  3) then, for example, use SPARQL to query the resulting (expanded) graph
- This application model is very important for RDF based applications

What can be done in OWL RL?

- Many features are available:
  - identity of classes, instances, properties
  - subproperties, subclasses, domains, ranges
  - union and intersection of classes (though with some restrictions)
  - property characterizations (functional, symmetric, etc)
  - property chains
  - keys
  - some property restrictions (but not all inferences are possible)
- All *examples* so far could be inferred with OWL RL!

What cannot be done in OWL RL?

- There are restrictions on what can be a sub and superclass. Eg, the following is not manageable:

```
B rdf:type owl:Class;
  owl:unionOf (P Q R).

A rdfs:subClassOf B .
```

- Some features are not available or are restricted:
  - not all datatypes are available
  - no (OWL 2) datatype restrictions
  - no minimum or exact cardinality restrictions
  - maximum cardinality only with 0 and 1

What cannot be done in OWL RL?

- Some “natural” conclusions cannot be drawn, eg:

```
A rdf:type owl:Class;
  owl:intersectionOf (U V S).
```

does *not* yield:

```
A rdfs:subClassOf U .
```

Another profile example: OWL QL

- Close to a subset of RL
- Cannot handle keys, functional, transitive, etc, properties
  - essentially, handles classification of terms, class and property equivalence, etc
- Cannot handle **`owl:sameAs`**
- But… queries can be translated to SQL directly
  - ie, there is no need to modify/extend the graph like in RL
  - just have a conceptual mapping to a RDB, and off you go…

Alternative syntaxes for OWL 2

- OWL constructs in RDF can be fairly verbose
- There are alternative syntaxes to express ontologies
  - direct XML encoding of ontologies (OWL/XML)
  - “functional” syntax
  - “Manchester” syntax
- *The official exchange syntax is RDF (RDF/XML)*
  - all other syntaxes are optional for tools

OWL 2 Functional syntax example

```
my:£ rdf:type owl:Thing.
my:€ rdf:type owl:Thing.
my:$ rdf:type owl:Thing.

my:Currency rdf:type owl:Class;
    owl:oneOf (my:€ my:£ my:$).

my:Listed_Price rdf:type owl:Class;
    rdfs:subClassOf [
      rdf:type              owl:Restriction;
      owl:onProperty        p:currency;
      owl:allValuesFrom     my:Currency
    ].
```

is equal to:

```
Declaration(NamedIndividual(my:£))
Declaration(NamedIndividual(my:€))
Declaration(NamedIndividual(my:$))

Declaration(Class(:Currency))
EquivalentClasses(:Currency OneOf(my:€ my:£ my:$))

SubClassOf(my:Listed_Price AllValuesFrom(p:currency my:Currency))
```

Manchester syntax example

```
my:£ rdf:type owl:Thing.
my:€ rdf:type owl:Thing.
my:$ rdf:type owl:Thing.

my:Currency rdf:type owl:Class;
    owl:oneOf (my:€ my:£ my:$).

my:Listed_Price rdf:type owl:Class;
    rdfs:subClassOf [
      rdf:type              owl:Restriction;
      owl:onProperty        p:currency;
      owl:allValuesFrom     my:Currency
    ].
```

is equal to:

```
Individual: my:€
Individual: my:£
Individual: my:$

Class: my:Currency EquivalentTo {my:€ my:£ my:$}

Class: my:Listed_Price that p:currency only my:Currency
```

Ontology development

- The hard work is to *create* the ontologies
  - requires a good knowledge of the area to be described
  - some communities have good expertise already (e.g., librarians)
  - *OWL is just a tool to formalize ontologies*
  - large scale ontologies are often developed in a community process
- Ontologies should be *shared* and *reused*
  - can be via the simple namespace mechanisms…
  - …or via explicit imports
- Applications can also be developed with very small ontologies, though

Ontologies examples

- eClassOwl: eBusiness ontology for products and services, 75,000 classes and 5,500 properties
- National Cancer Institute's ontology: about 58,000 classes
- Open Biomedical Ontologies Foundry: a collection of ontologies, including the Gene Ontology to describe gene and gene product attributes in any organism or protein sequence and annotation terminology and data (UniProt)
- BioPAX: for biological pathway data

Using thesauri, glossaries
(SKOS)

SKOS (Simple Knowledge Organization System)

- Represent and share classifications, glossaries, thesauri, etc
  - for example:
    - Dewey Decimal Classification, Art and Architecture Thesaurus, ACM classification of keywords and terms…
    - classification/formalization of Web 2.0 type tags
- Define classes and properties to add those structures to an RDF universe
  - allow for a quick port of this traditional data, combine it with other data

Example: entries in a glossary

Assertion

(i) Any expression which is claimed to be true.
(ii) The act of claiming something to be true.

Class

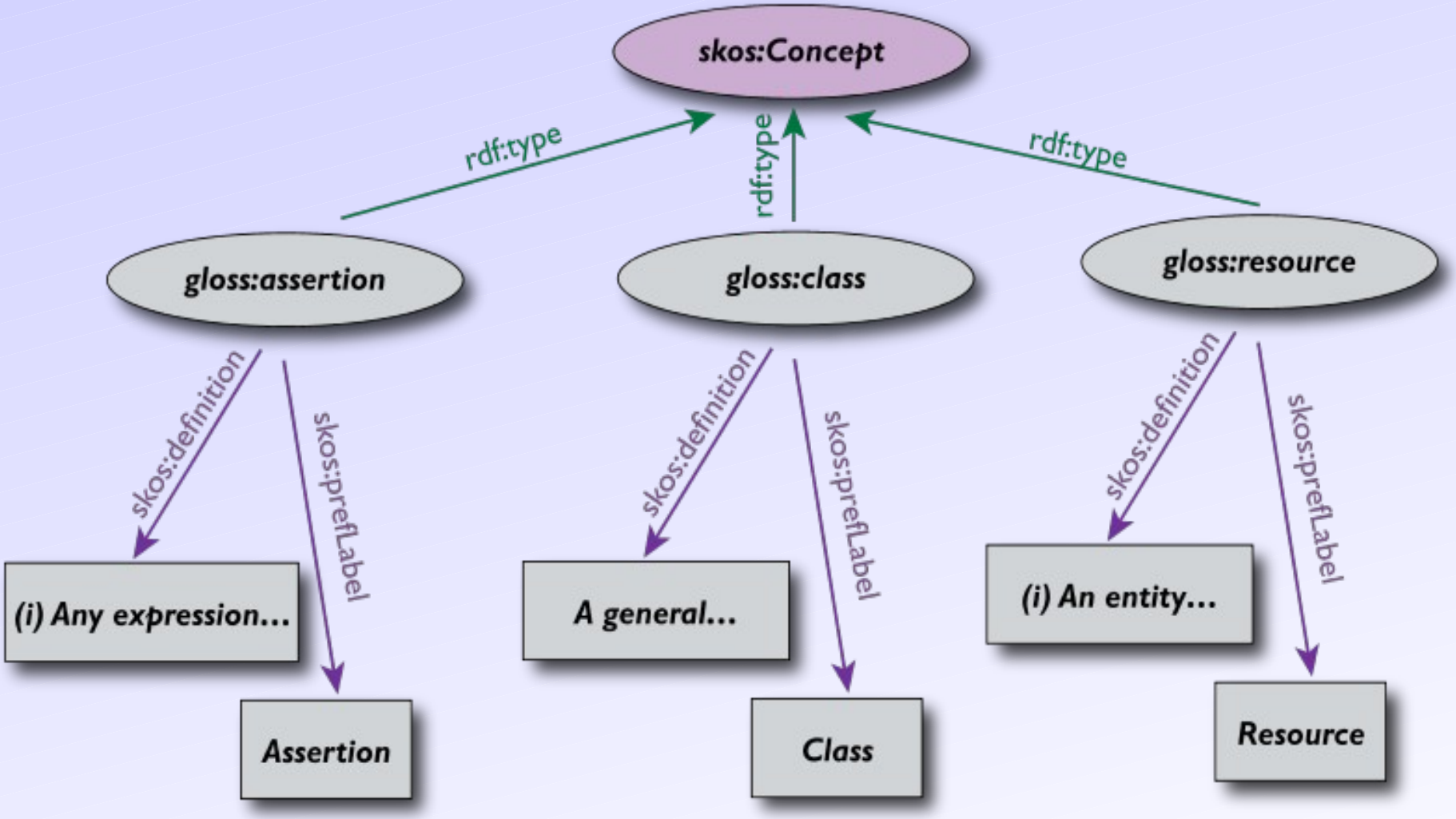
A general concept, category or classification. Something used primarily to classify or categorize other things.

Resource

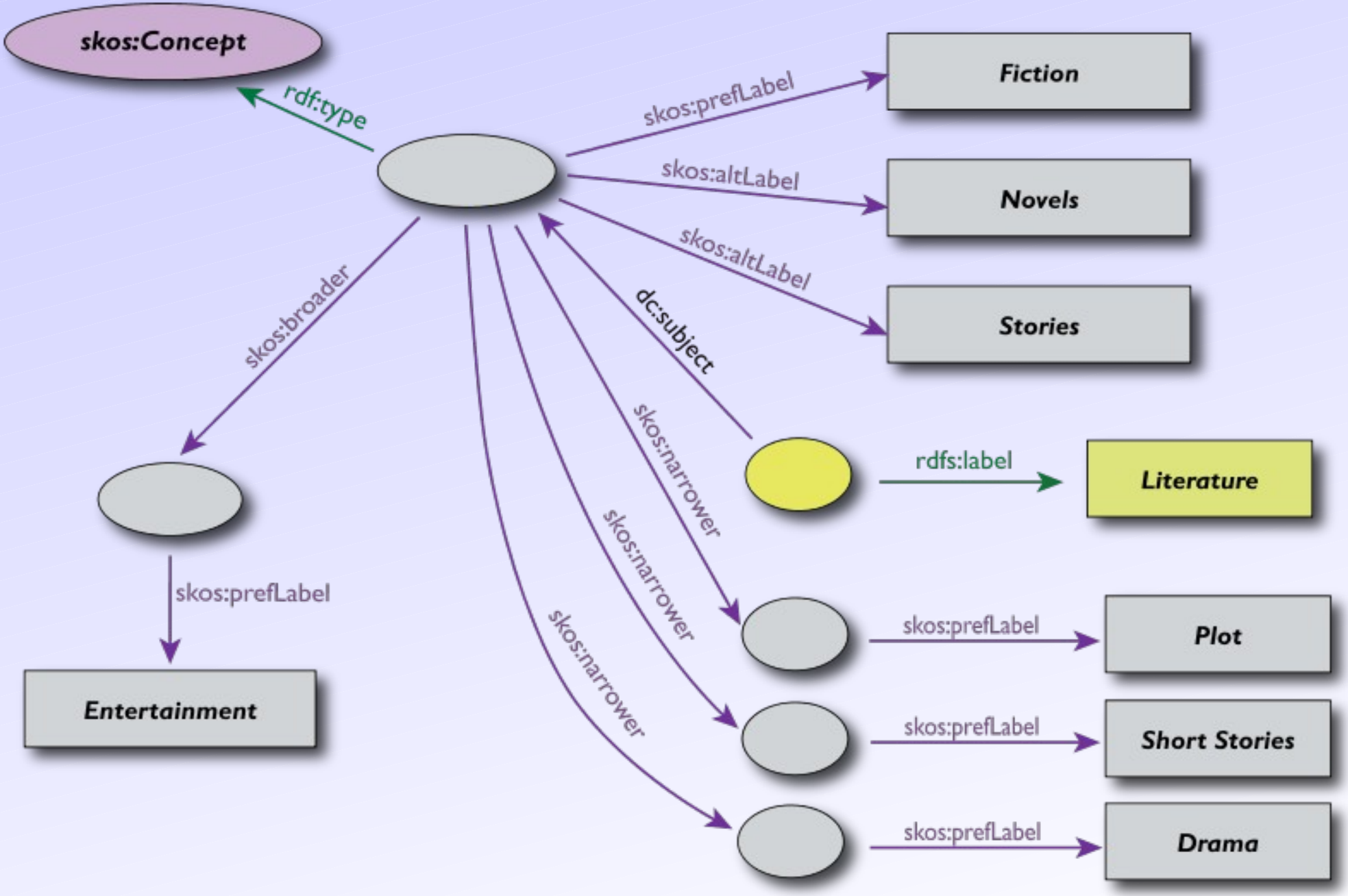
(i) An entity; anything in the universe.
(ii) As a class name: the class of everything; the most inclusive category possible.

(from the RDF Semantics Glossary)

Example: entries in a glossary in SKOS



A more complex structure



Same serialized

```
<http://dbpedia.org/resource/Category:Fiction>
    a    skos:Concept;
    skos:altLabel "Novels", "Stories", ...;
    skos:narrower
       <http://dbpedia.org/resource/Category:Plot>,
       <http://dbpedia.org/resource/Category:Short_stories>,
       <http://dbpedia.org/resource/Category:Drama>,
        ...;
    skos:prefLabel "Fiction";
    skos:broader
       <http://dbpedia.org/resource/Category:Entertainment>;
    ...
.

<http://dbpedia.org/resource/Literature>
    rdfs:label "Literature";
    dc:subject <http://dbpedia.org/resource/Category:Fiction>;
    ...
.
```

SKOS Reference overview

- Classes and Properties:
  - Basic description (**`Concept`**, **`ConceptScheme`**,…)
  - Labeling (**`prefLabel`**, **`altLabel`**,…)
  - Documentation (**`definition`**, **`historyNote`**,…)
  - Semantic relations (**`broader`**, **`narrower`**, **`related`**,…)
  - Collections (**`Collection`**, **`OrderedCollection`**,…)
  - Concept mappings (**`broadMatch`**, **`narrowMatch`**,…)

SKOS and OWL

- SKOS is geared towards some specific (though large) use cases, like
  - taxonomies, glossaries, …
  - annotations of complex structures (including ontologies)
- SKOS is a based on a very simple usage of OWL
  - roughly on the rule based level
  - the emphasis is on *organization* and not on logical inferences

Find RDF Data for resources
(POWDER)

How to “assign” RDF data to resources?

- This is important when the RDF data is used as “metadata”
- Some examples:
  - copyright information for your photographs
  - is a Web page usable on a mobile phone and how?
  - bibliographical data for a publication
  - annotation of the data resulting from a scientific experiment
  - etc

If I know the URI of the resource (photograph, publication, etc), how do I find the relevant RDF data?

The data might be embedded

- Some data formats allow the direct inclusion of (RDF) metadata:
  - SVG (Scalable Vector Graphics)
    - direct inclusion of RDF/XML
    - via RDFa attributes
  - XHTML with RDFa or microformats+GRDDL
  - JPG files using the comment area and, eg, Adobe's XMP technology

Simple linkage

- Some formats have special link statements. Eg, in (X)HTML:

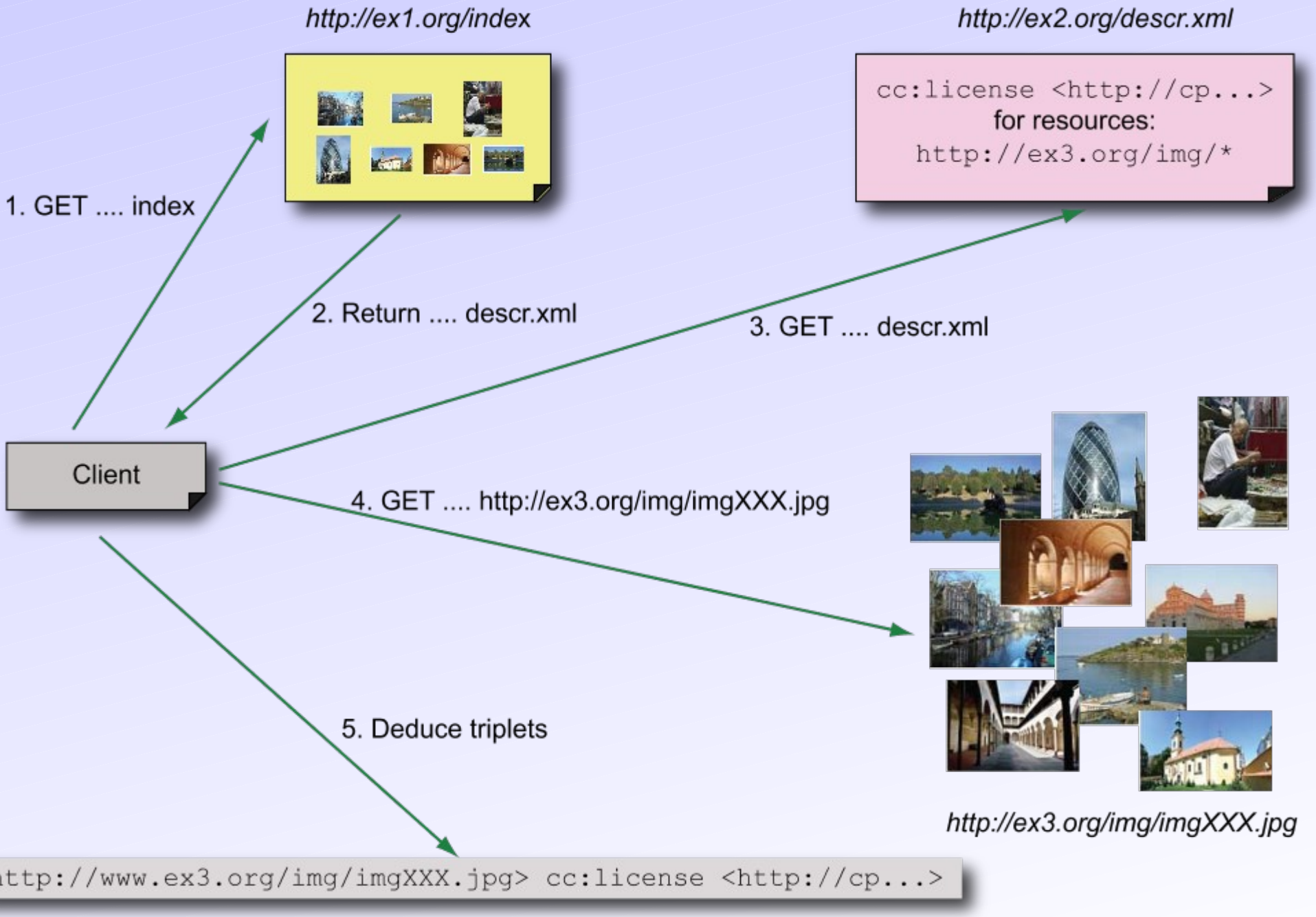
```
<html>
  <head>
    <link rel="meta" href="meta.rdf"/>
 ...
```

- Similar facilities might exist in other formats (eg, SMIL)

POWDER

- POWDER provides for a more elaborate scenarios:
  1. define a *set* of resources by constraints on the URIs; eg
     - URIs must begin with **`http://www.example.com/bla/`**
     - the port number in the URI-s should be **`XYZW`**
  2. define “description resources” that bind those resources to additional information
  3. get such description resources, eg, via a link statements, via HTTP, via SPARQL, …
- Use cases: licensing information, mobileOK (and other) trustmarks, finding accessible Web sites, content labeling (eg, child protection), …

A POWDER scenario: copyright for photos



The gory details…

- The “description resource” is an XML file:

```
<powder xmlns="http://www.w3.org/2007/05/powder#"
        xmlns:cc="http://creativecommons.org/ns#">
  <attribution>
    <issuedby src="http://www.ivan-herman.net/me"/>
  </attribution>
  <dr>
    <iriset>
      <includehosts>www.ex2.org</includehost>
      <includepathstartswith>/img/</includepathstartswith>
    </iriset>
    <descriptorset>
      <cc:license rdf:resource="http://cp:..."/>
    </descriptorset>
  </dr>
```

The gory details…

- Powder processors may then return
  - direct RDF triples, eg:

```
<http://www.ex2.org/img/imgXXX.jpg> cc:license <http://cp:...>.
```

  - or can transform this XML file into an RDF (OWL) for more generic processors
    - a canonical processing of the XML file is defined by the POWDER specification

POWDER Service

- Online POWDER service can be set up:
  - a Web service with
    - submit a URI and a resource description file
    - return the RDF statements for that URI
  - such service should be set up, eg, at W3C
- A GRDDL transform is also defined

But there is a hidden "hiccup"

- RDF makes a strong separation between
  - URI as an ID for a resource
  - URI as a datatype (**`xsd:anyURI`**)
  - there is no "bridge" between the two
- POWDER includes a small *extension* to the formal semantics of RDF for two properties:
  - **`wdrs:matchregex`** and **`wdrs:notmatchregex`** such that
    - **`(R wdrs:matchregex Regex)`** holds iff the *URI* of R matches Regex

If you want the OWL version…

```
<> wdrs:issuedBy <http://www.ivan-herman.net/me> .

_:iriset_1 a owl:Class; owl:intersectionOf (
  [ a owl:Restriction;
    owl:onProperty wsdr:matchregex ;
    owl:hasValue "..ugly regex for ex2.org"^^xsd:string ]
  [ a owl:Restriction;
    owl:onProperty wsdr:matchregex ;
    owl:hasValue "..ugly regex for /img"^^xsd:string ]
  ).

_:desc_1 a owl:Restriction;
  owl:onProperty cc:license;
  owl:hasValue <http://cp:...">.

_:iriset_1 rdfs:subClassOf _:desc_1 .
```

Consequences of the “hiccup”

- In practice this means that only “POWDER aware” agents can fully handle the description files
  - note that the extension is fairly easy to add, so it is not a big implementation issue…
- Existence of the services to provide the triplets automatically relieve the pain…

Other POWDER features

- There are a number of additional features:
  - built in authentication: description resources *must* be attributed and this is open for authentication
  - assignments may carry validity dates
  - packaging several resource descriptions in one, possibly control their processing order
  - using tags to identify resources instead of URI patterns

Rules
(RIF)

Rules

- There is a long history of rule languages and rule-based systems
  - eg: logic programming (Prolog), production rules
- Lots of small and large rule systems (from mail filters to expert systems)
- Hundreds of niche markets

Why rules on the Semantic Web?

- There are conditions that ontologies (ie, OWL) cannot express
  - a well known example is Horn rules: (P1 ∧ P2 ∧ …) → C
    - (though OWL property chains cover *some* cases)
- A different way of thinking — people may feel more familiar in one or the other

Things you may want to express

- An example from our bookshop integration:
  - “a novel with over 500 pages and costing less than €5 is a cheap book”
  - something like (in an ad-hoc syntax):

```
If { ?x rdf:type p:Novel;
        p:page_number ?p;
        p:price [
            p:currency p:€;
            rdf:value  ?z
        ].
      ?p > "500"^^xsd:integer.
      ?z < "5.0"^^xsd:double. }
then { ?x rdf:type p:CheapBook }
```

A new requirement: exchange of rules

- Applications may want to exchange their rules:
  - negotiate eBusiness contracts across platforms: supply vendor-neutral representation of your business rules so that others may find you
  - describe privacy requirements and policies, and let clients "merge" those (e.g., when paying with a credit card)
- Hence the name of the working group: *Rule Interchange Format*
  - goal is a language that
    - expresses the rules a bit like a rule language
    - can be used to exchange rules among engines

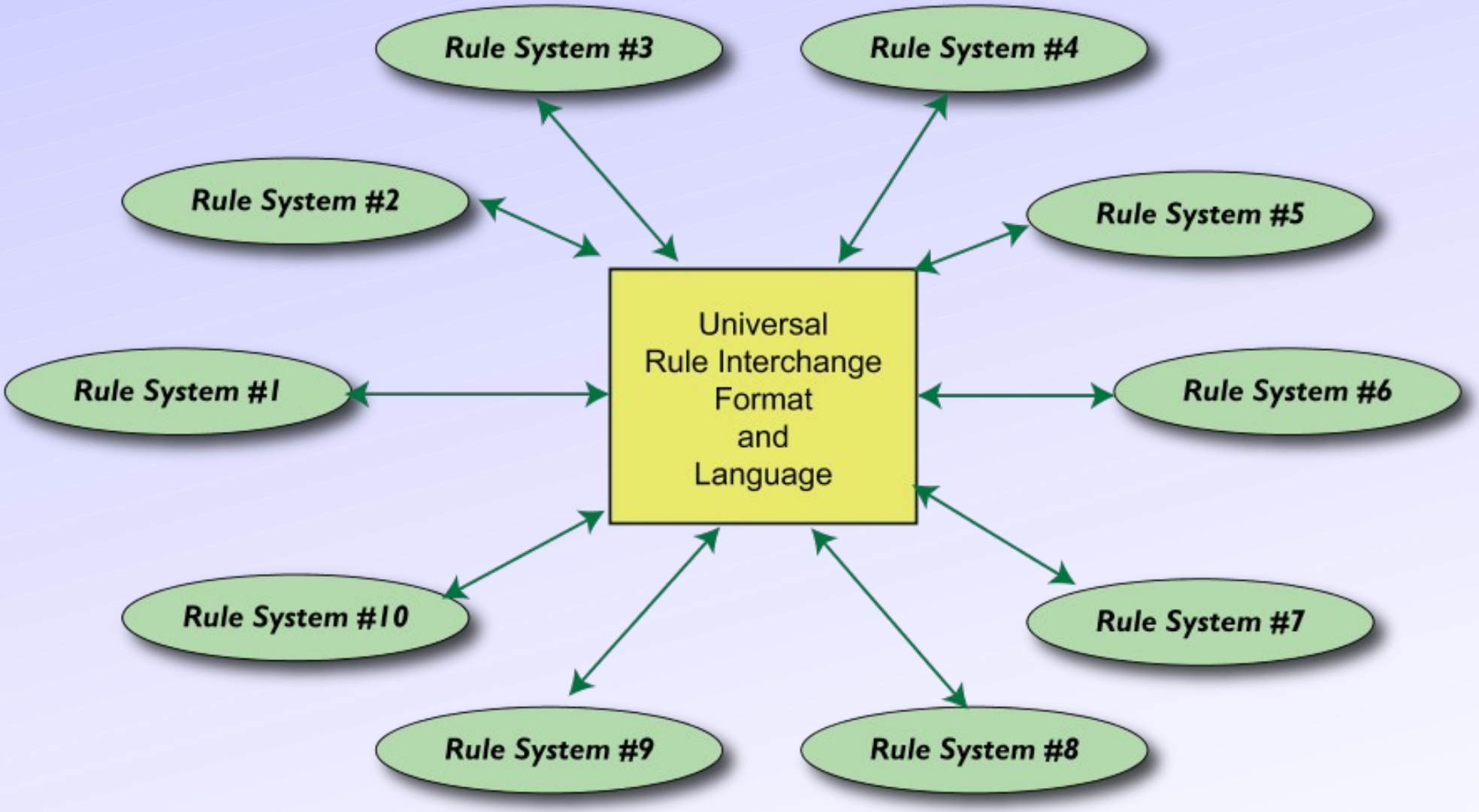
Notes on RIF

- In some ways, the goals of RIF go beyond the "core" Semantic Web
  - eg, the exchange format does not concentrate on RDF only
- But if we look at the interchange of data, then it is in line of a general vision
- And… this is what the community wanted to do…

Notes on RIF (cont)

- RIF does not concentrate on RDF only
  - ie, certain constructions go beyond what RDF can express
- But there is a “subset” that is RDF and also OWL related
- For the coming few slides, forget about RDF
  - We will come back to it. Promise!

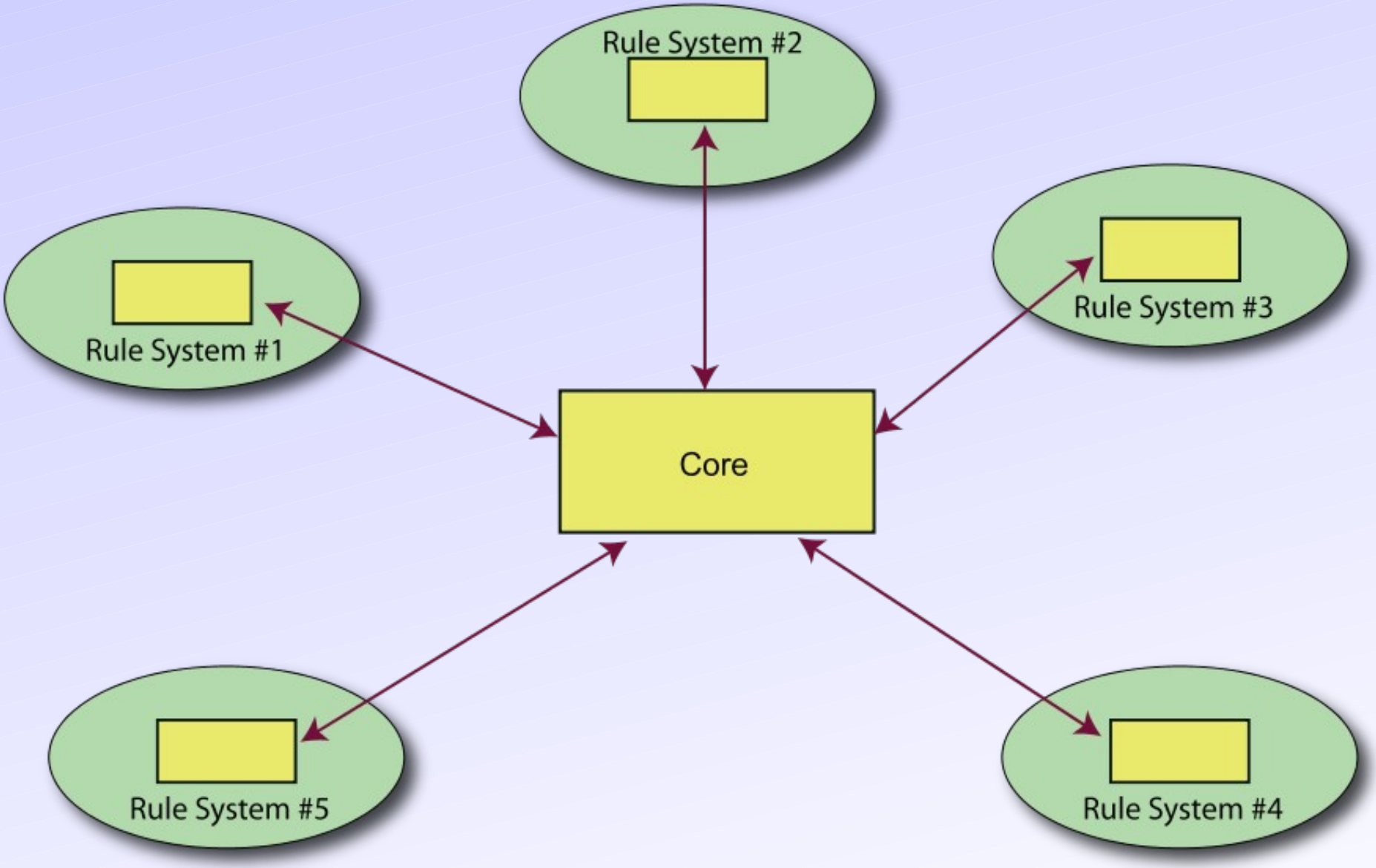
In an ideal World



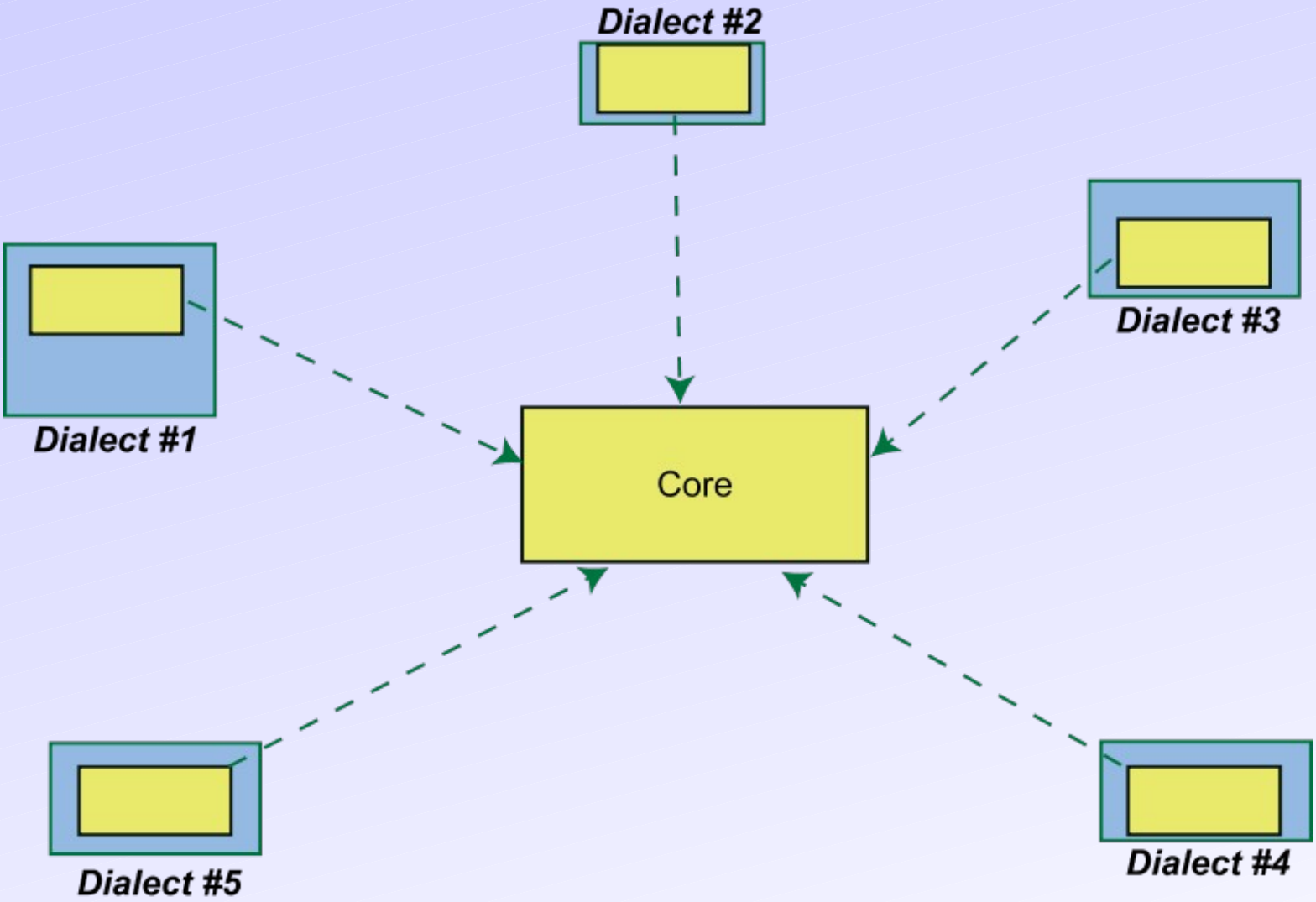
In the real World…

- Rule based systems can be very different
  - different rule semantics (based on various type of model theories, on proof systems, etc)
  - production rule systems, with procedural references, state transitions, etc
- Such universal exchange format is not feasible
- The idea is to define “cores” for a family of languages with “variants”

RIF “core”: only partial interchange

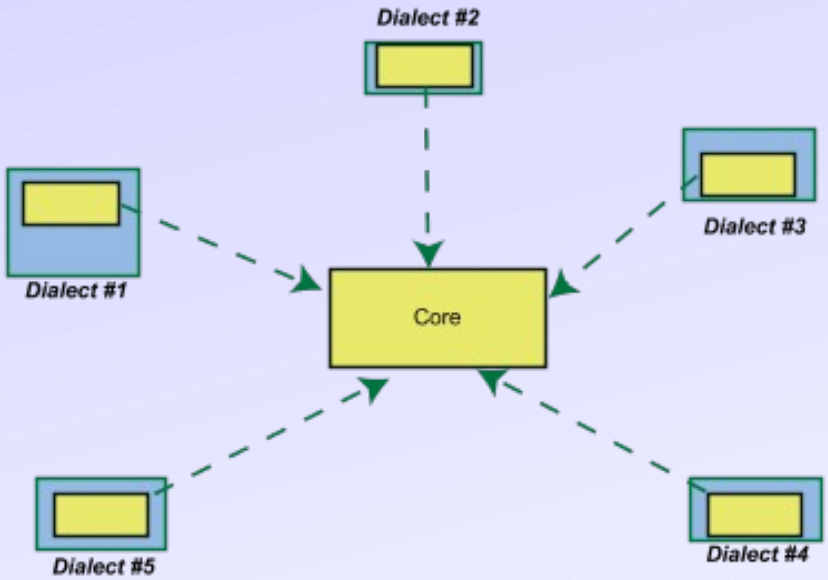


RIF "dialects"

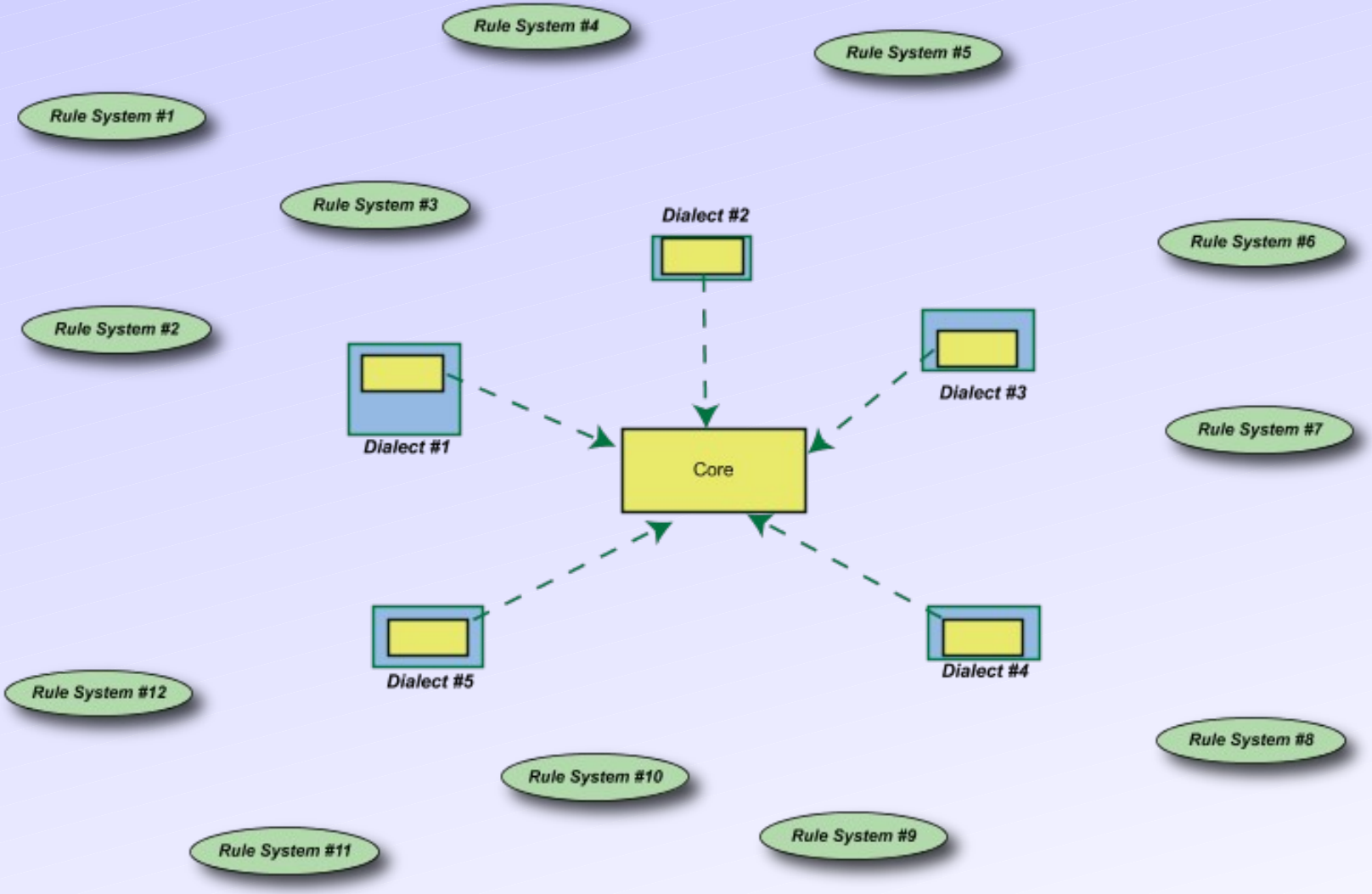


- Possible dialects: F-logic, production rules, fuzzy or probabilistic logic, …

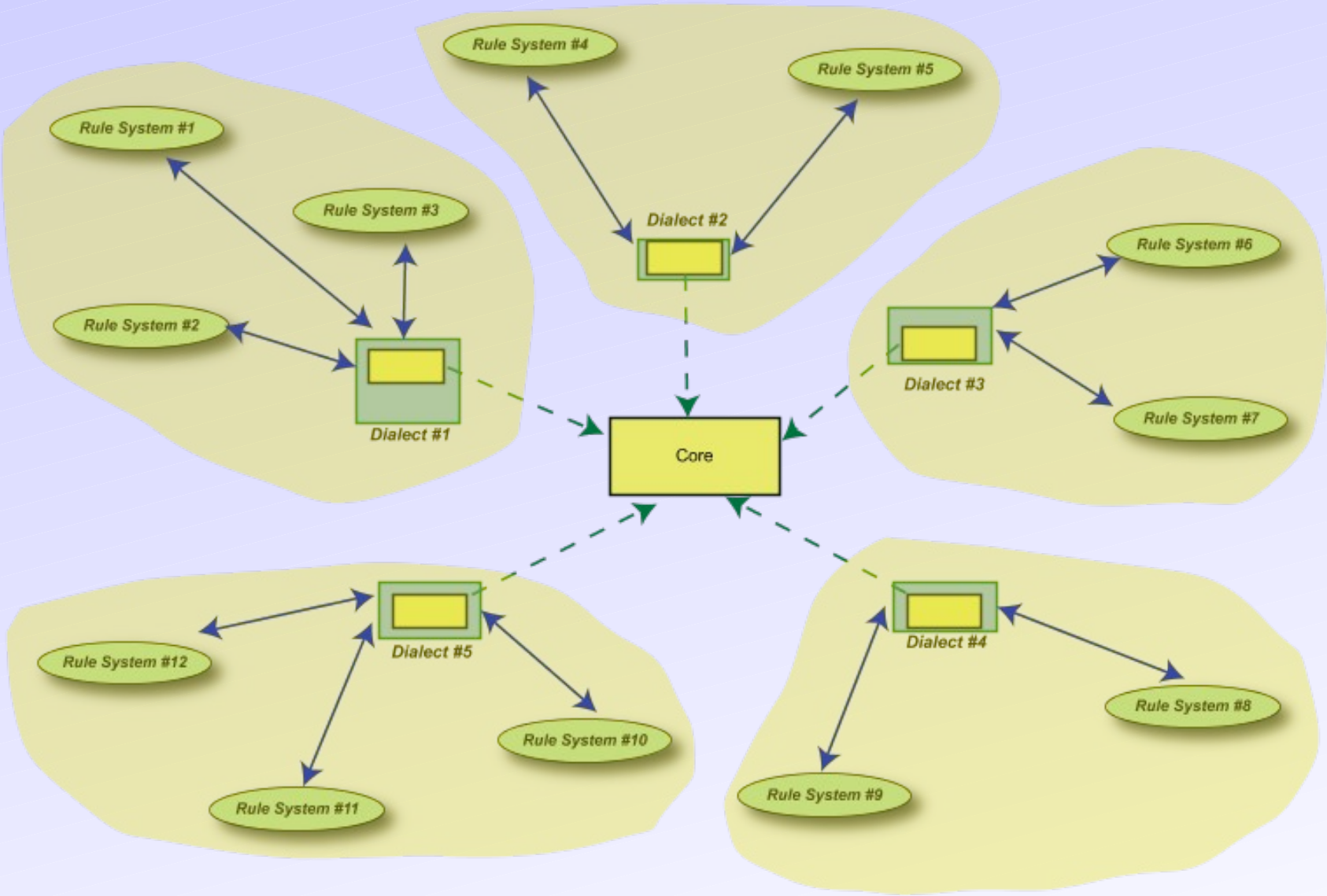
Role of dialects



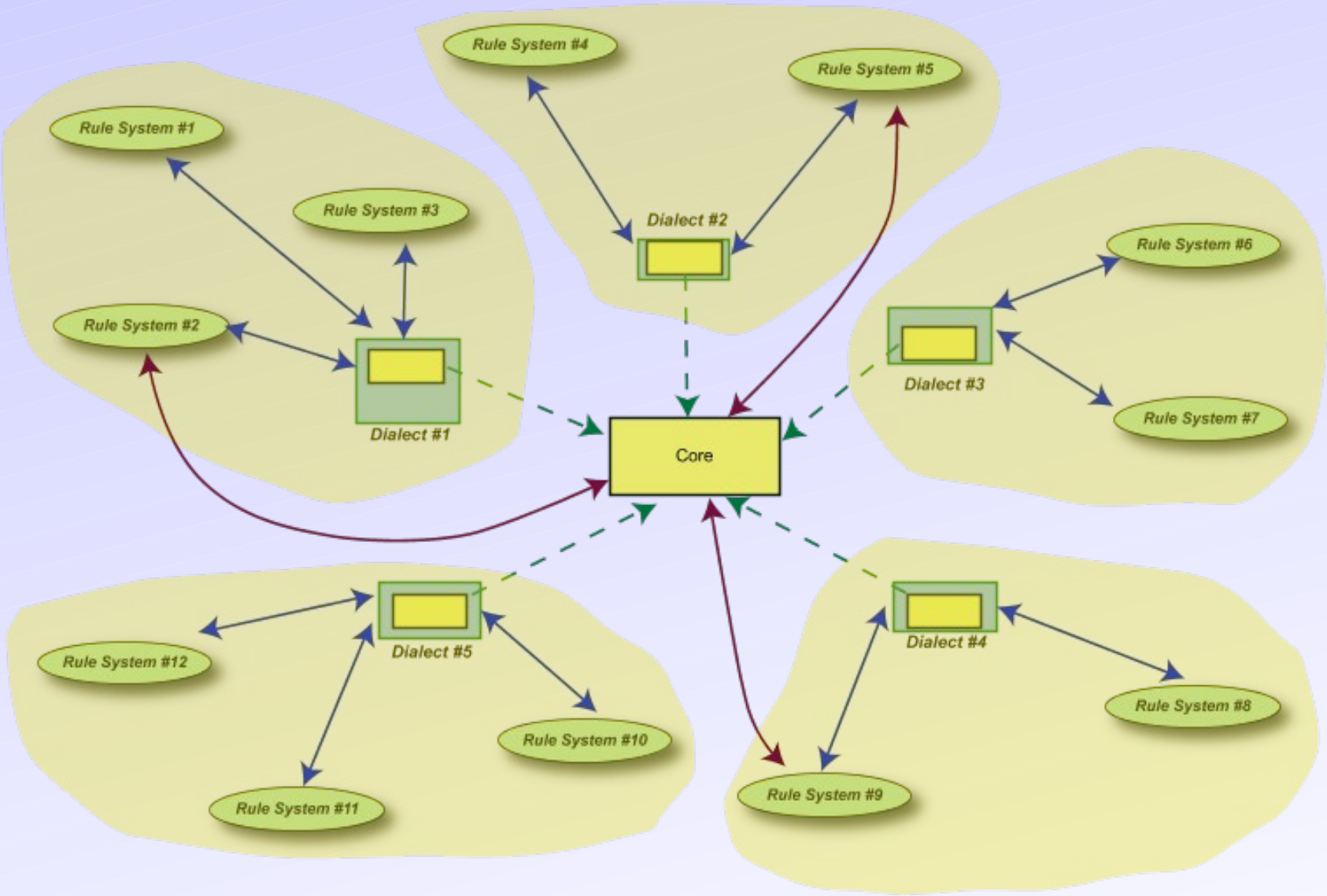
Role of dialects



Role of dialects



Role of dialects



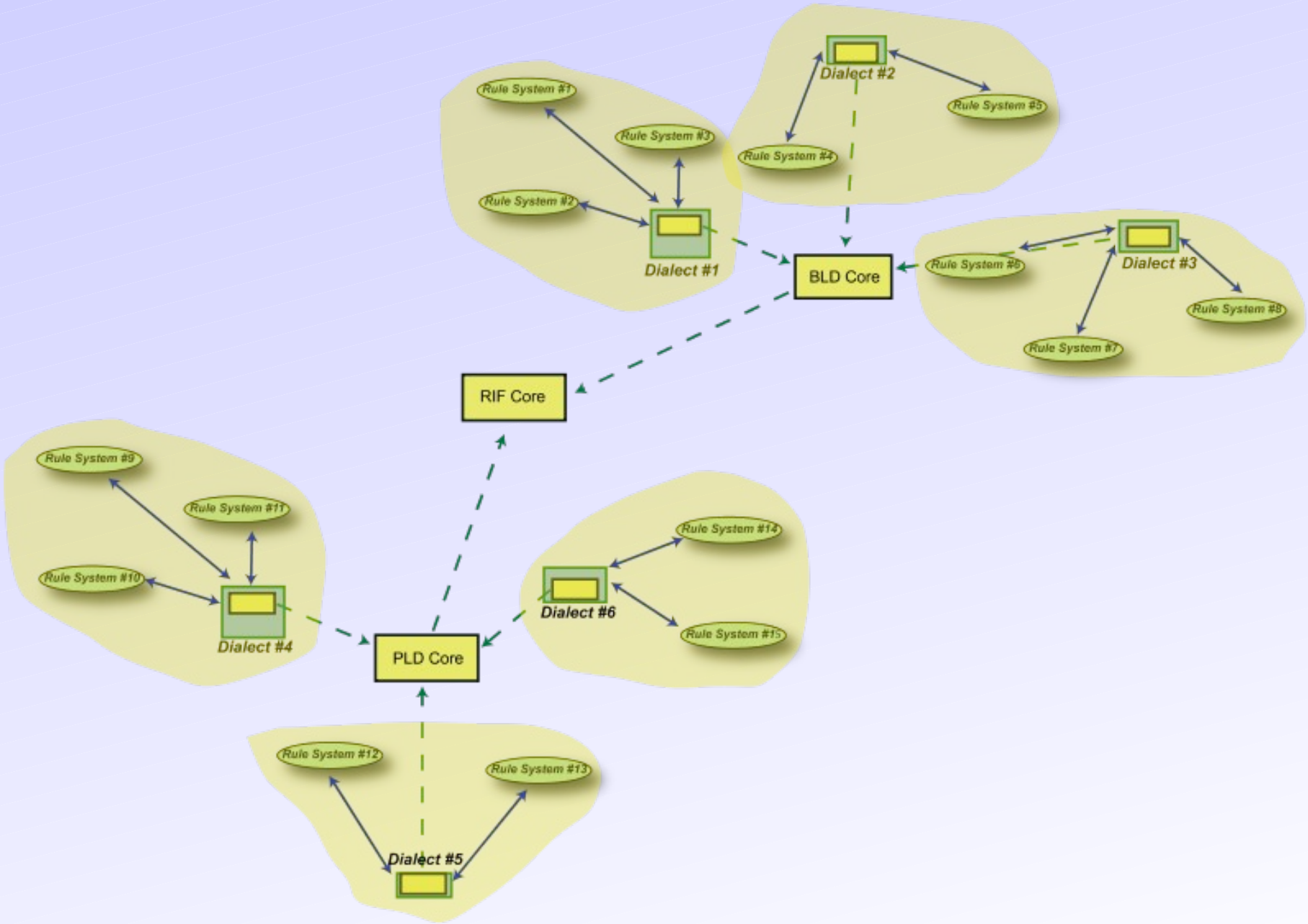
However…

- Even this model does not completely work
- The gap between production rules and “traditional” logic systems is too large
- A hierarchy of cores is necessary:
  - a *Basic Logic Dialect* and *Production Rule Dialect* as “cores” for families of languages
  - a common *RIF Core* binding these two

Schematically…

- The “BLD (Basic Logic Dialect)” is of the form:
  - “`if condition true then this is true`”
  - conditions may include functions, hierarchies
- The “PLD (Production Logic Dialect)” is of the form:
  - “`if condition is true then do something`”
- The “Core”: shared subset of major languages
  - technically: positive Horn without function terms, with some simple datatypes

Hierarchy of cores



Current status

- There is a fairly final draft for the BLD
- The work on the PLD Core is also on its way
- The Core is defined as an abstraction from BLD and PLD

RIF BLD

- RIF BLD is the closest to the needs of the RDF world
- BLD defines
  - a “presentation syntax”, which is really to… present the constructions (is not necessarily implemented in tools)
  - a formal XML syntax to encode and exchange the rules
- A BLD document is
  - some directives like import, prefix settings for URI-s, etc
  - a sequence of implications, possibly involving built-in predicates on datatypes

RIF BLD example

```
Document(
  Prefix(cpt http://example.com/concepts#)
  Prefix(ppl http://example.com/people#)
  Prefix(bks http://example.com/books#)

  Group
  (
    Forall ?Buyer ?Item ?Seller (
        cpt:buy(?Buyer ?Item ?Seller):- cpt:sell(?Seller ?Item ?Buyer)
    )
    cpt:sell(ppl:John bks:LeRif ppl:Mary)
  )
)
```

infers the following relationship:

```
cpt:buy(ppl:Mary bks:LeRif ppl:John)
```

Additional RIF BLD features

- RIF BLD includes some extra features
  - built-in datatypes and predicates
  - notion of “local names”, a bit like RDF’s blank nodes
  - a “frame-based” syntax (beyond predicates and functions):
    - `p[prop1->v1 prop2->v2]`
  - built-in abstractions for classes, subclassing, and typing:
    - `m # C`, `C1 ## C2`
- RIF BLD’s semantics follows the “usual” approach in logic

What about RDF(S), OWL, and BLD?

- Typical scenario: applications exchange rules that refer to RDF data
- To make that work:
  - RDF facts/triples have to be representable in BLD
  - harmonization on the concepts is necessary (eg, classes)
  - the formal semantics of the two worlds should also be aligned
- There is a separate document that brings these together

What about RDF(S), OWL, and BLD?

- Triples can be expressed in BLD using the frame syntax:
  - `(s p o)` is written as `s[p->o]`
    - (a bit reminiscent of the turtle syntax but '[' does *not* introduce any blank node!)
  - subclassing and typing of BLD are equivalent to their RDFS counterpart
  - the datatypes are (almost…) identical to OWL 2

Rewrite of our earlier example

We describe/exchange the rules:

```
Group
(
  Forall ?Buyer ?Item ?Seller (
      ?Buyer[cpt:buy->?Item cpt:from->?Seller] :-
          ?Seller[cpt:sell->?Item cpt:to->?Buyer]
  )
)
```

Rewrite of our earlier example

We describe/exchange the rules:

```
Group
(
  Forall ?Buyer ?Item ?Seller (
      ?Buyer[cpt:buy->?Item cpt:from->?Seller] :-
          ?Seller[cpt:sell->?Item cpt:to->?Buyer]
  )
)
```

We then *import* the RDF data:

```
ppl:Mary
   cpt:sell bks:LeRif;
   cpt:to   ppl:John .
```

Rewrite of our earlier example

We describe/exchange the rules:

```
Group
(
  Forall ?Buyer ?Item ?Seller (
      ?Buyer[cpt:buy->?Item cpt:from->?Seller] :-
          ?Seller[cpt:sell->?Item cpt:to->?Buyer]
  )
)
```

We then *import* the RDF data, and infer:

```
ppl:Mary
   cpt:sell bks:LeRif;
   cpt:to   ppl:John .

ppl:John
   cpt:buy bks:LeRif;
   cpt:from ppl:Mary .
```

Remember the what we wanted from Rules?

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

If { ?x rdf:type p:Novel;
        p:page_number ?p;
        p:price [
            p:currency :€;
            rdf:value  ?z
        ].
      ?p > "500"^^xsd:integer.
      ?z < "5.0"^^xsd:double. }
then { ?x rdf:type p:CheapBook }
```

The same with RIF BLD Presentation syntax

```
Prefix(p http://www.example.org/bookterms#)
Prefix(rdf http://www.w3.org/1999/02/22-rdf-syntax-ns#)
Prefix(pred http://www.w3.org/2007/rif-builtin-predicate#)

Forall ?x ?p ?z (
  ?x # p:CheapBook :-
    And(
      ?x # p:Novel
      ?x[p:page_numper->?p p:price->_abc]
      _abc[p:currency->€ rdf:value->?z]
      External( pred:numeric-greater-than(?p 500) )
      External( pred:numeric-less-than(?z 5.0) )
    )
)
```

A word on the syntax

- The RIF BLD Presentation syntax is… only syntax
- It can express more than what RDF needs
- Hopefully, a syntax will emerge with
  - close to one of the RDF syntaxes with a better integration of rules
  - only the relevant subset of BLD
    - note: there is a syntax called n3 that is very close…
  - can be mapped on BLD implementations as they come

Rules vs OWL?

- In a SW application, should I use RIF, OWL, or both?
- The two approaches are complimentary
  - there are things that rules cannot really express or infer
    - eg, inferencing complex relationships among classes
  - there are things that ontologies cannot really express or in only a very complicated manner
    - eg, complex Horn rules
- Often, applications require both

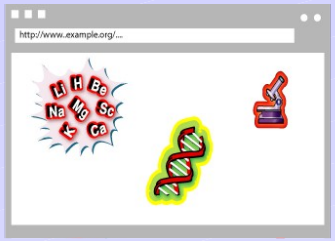
What about OWL RL?

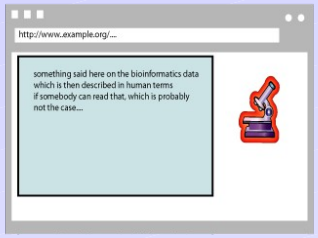
- OWL RL stands for “Rule Language”…
- OWL RL is in the intersection of RIF BLD and OWL
  - inferences in OWL RL can be expressed with rules
    - the rules are precisely described in the OWL spec, b.t.w.
  - a BLD implementation should be able to implement OWL RL by just feeding in the rules
    - (status in Febr 2009: some details are still to be fleshed out…)

What have we achieved?

(putting all this together)

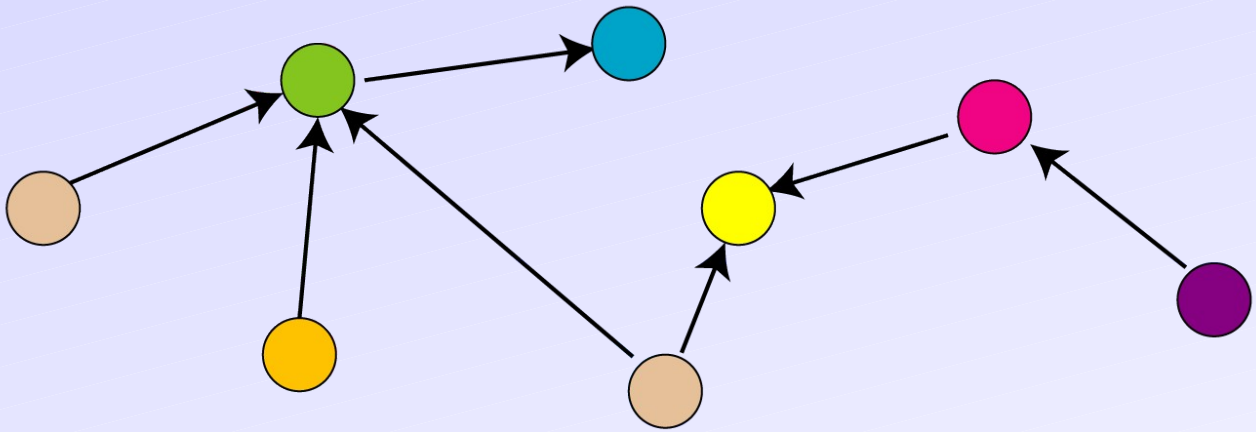
Remember the integration example?





Applications

Query, Manipulate, etc.



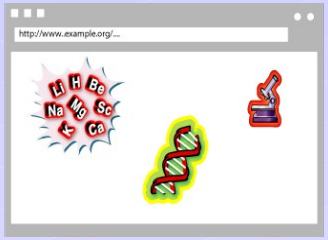
Data represented in abstract format

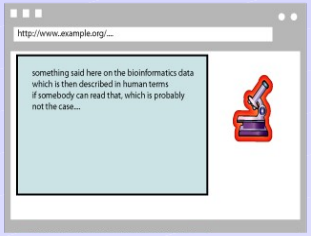
Map, Expose, etc.

Data in various formats

W3C Semantic Web

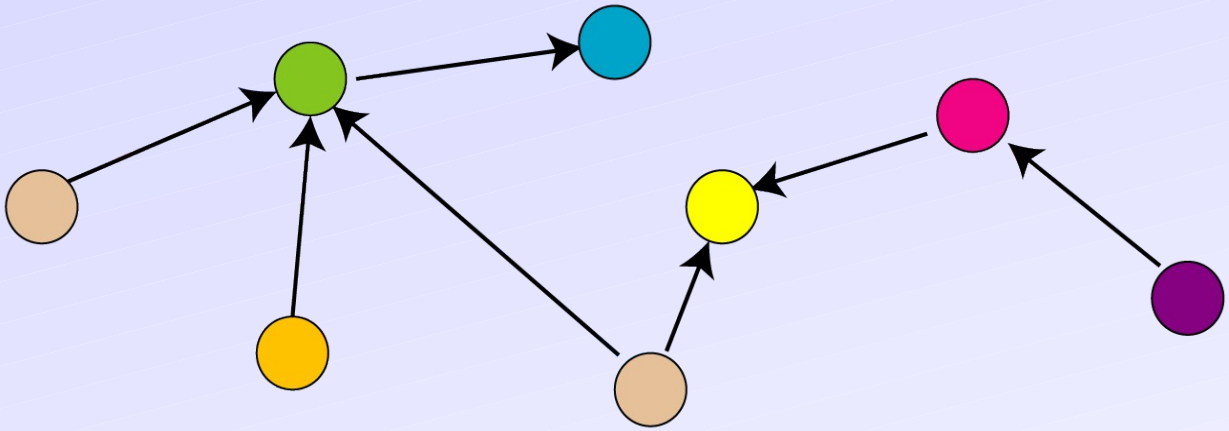
Same with what we learned





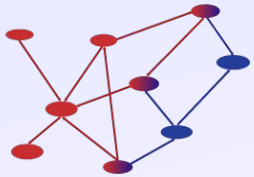
Applications

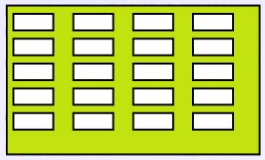
SPARQL,
OWL inferences,
etc.

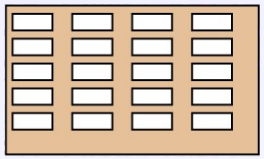


Data represented in RDF, possibly with extra knowledge (RDFS, OWL, SKOS, Rules, …)

SQL <=> RDF,
GRDDL, RDFa
etc.







Data in various formats

What is coming?

Revision of the RDF model?

- Some restrictions in RDF may be unnecessary
- Issue of “named graph”: possibility to give a URI to a set of triples and make statements on those
- Syntax issues in RDF/XML (eg, QNames in properties)
- Alternative XML serializations?
- Add a time tag to statements?
- Internationalization issues with literals (how do I set “bidi” writing?)
- …

Other items…

- Security, trust, provenance
  - combining cryptographic techniques with the RDF model, sign a portion of the graph, etc
  - trust models
- Access control on statements or groups of statements
- Quality constraints on graphs
  - “may I be sure that certain patterns are present in a graph?”
- Ontology merging, alignment, term equivalences, versioning, development, …
- etc

Other items: uncertainty

- Fuzzy logic
  - look at alternatives of Description Logic based on fuzzy logic
  - alternatively, extend RDF(S) with fuzzy notions
- Probabilistic statements
  - have an OWL class membership with a specific probability
  - combine reasoners with Bayesian networks
- A W3C Incubator Group ("Uncertainty Reasoning on the World Wide Web") has issued a report on the current status, possibilities, directions, etc
- Possible RIF dialect for fuzzy logic, for example?

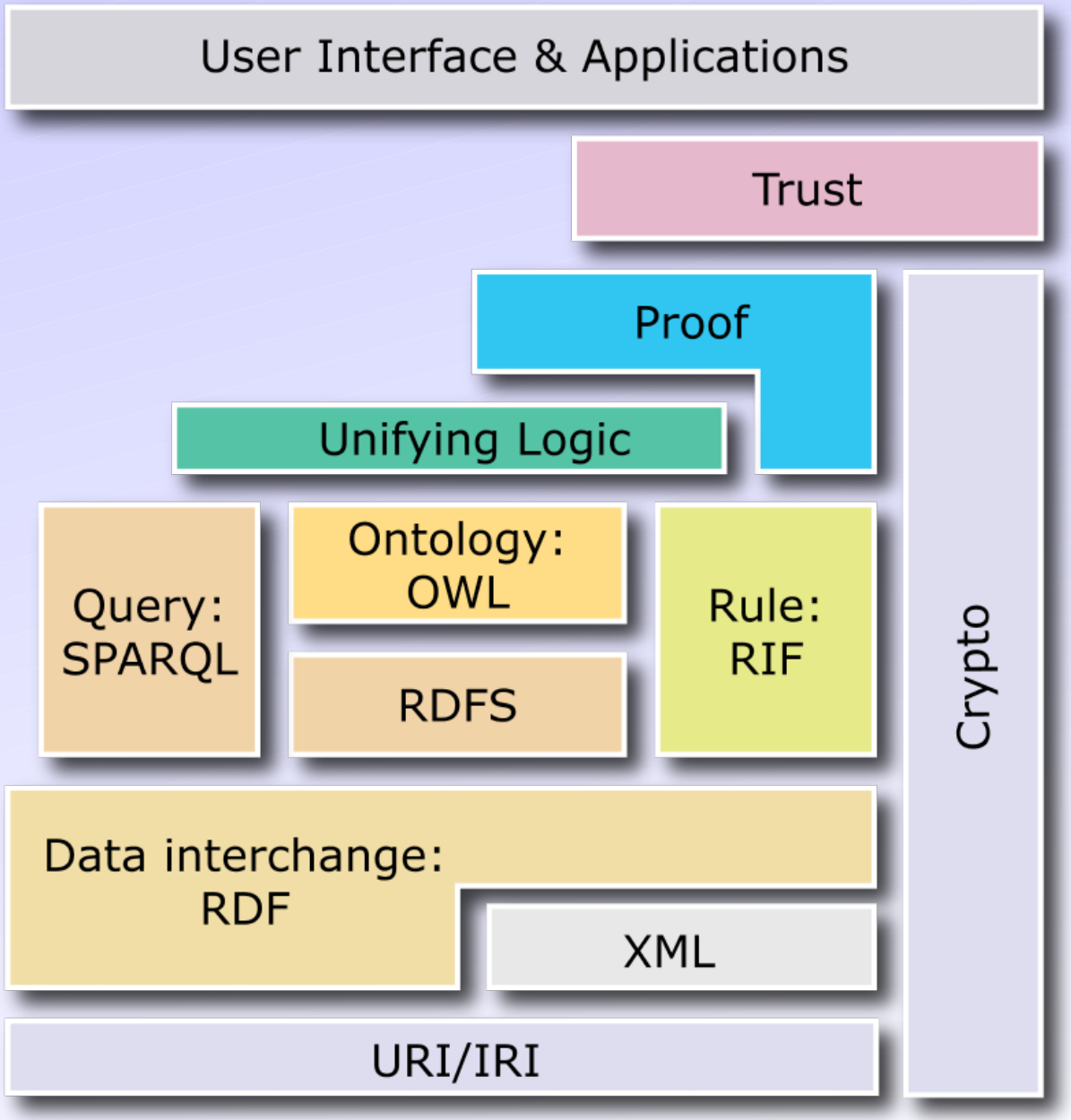
Other items: naming

- The SW infrastructure relies on unique naming of “things” via URI-s
- Lots of discussions are happening that touch upon general Web architecture, too
  - http URI-s or other URN-s?
  - URI-s for “informational resources” and “non informational resources”
  - how to ensure that URI-s used on the SW are dereferencable
  - etc

Other items: naming (cont)

- A different aspect of naming: *what* is the URI for a specific entity (regardless of the technical details)
  - what is the unique URI for, eg, Bach's Well-Tempered Clavier?
    - obviously important for, eg, music ontologies an data
  - who has the authority or the means to define and maintain such URI-s?
  - should we define characterizing properties for these and use `owl:sameAs` instead of a URI?
- The traditional library community may be of a big help in this area

The “layercake” diagram



Available documents, resources

Available specifications: Primers, Guides

- The “RDF Primer” and the “OWL Guide” give a formal introduction to RDF(S) and OWL
- SKOS has its separate “SKOS Primer”
- GRDDL Primer and RDFa Primer have been published
- The W3C Semantic Web Activity Homepage has links to all the specifications

"Core" vocabularies

- There are also a number "core vocabularies" (not necessarily OWL based)
  - Dublin Core: about information resources, digital libraries, with extensions for rights, permissions, digital right management
  - FOAF: about people and their organizations
  - DOAP: on the descriptions of software projects
  - SIOC: Semantically-Interlinked Online Communities
  - vCard in RDF
  - …
- One should never forget: ontologies/vocabularies must be shared and reused!

Some books

- G. Antoniu and F. van Harmelen: Semantic Web Primer, 2nd edition in 2008
- D. Allemang and J. Hendler: Semantic Web for the Working Ontologist, 2008
- Jeffrey Pollock: Semantic Web for Dummies, 2009
- …

See the separate Wiki page collecting book references: http://esw.w3.org/topic/SwBooks

Further information

- Planet RDF aggregates a number of SW blogs:
  - http://planetrdf.com/
- Semantic Web Interest Group
  - a forum developers with archived (and public) mailing list, and a constant IRC presence on freenode.net#swig
  - anybody can sign up on the list
    - http://www.w3.org/2001/sw/interest/

Great community…

Some good lessons

computas

- New standards (e.g. SPARQL), proposals for standardization (e.g. SPARUL), new tools (e.g. Jena), open source (e.g. Tomcat, Apache), lack of good documentation all say **high risk!!!!**
- **However, the support and maintenance from the W3C community and open source developers (e.g. Jena team) has been impressive, the support through IRC channels, mailing lists etc has been invaluable for the project.**

Slide 19

From a presentation given by David Norheim, Computas AS, ESTC2008 Conference, Vienna, Austria

SWBP Working Group documents

- Documents for ontology engineering
  - “Best Practice Recipes for Publishing RDF Vocabularies”
  - “Defining N-ary relations”
  - “Representing Classes as Property Values”;
  - “XML Schema Datatypes in RDF and OWL”
  - etc
- See the Group’s homepage for further links

Further information (cont)

- Description Logic links:
  - online course by Enrico Franconi,
  - teaching material and links by Ian Horrocks
- “Ontology Development 101”
- OWL Reasoning Examples
- Lots of papers at WWW2003-WWW2008; see also the ISWC200X conference proceedings (online since ISWC2006) as well as their European and Asian local variants

Lots of Tools (not an exhaustive list!)

- Categories:
  - Triple Stores
  - Inference engines
  - Converters
  - Search engines
  - Middleware
  - CMS
  - Semantic Web browsers
  - Development environments
  - Semantic Wikis
  - …
- Some names:
  - Jena, AllegroGraph, Mulgara, Sesame, flickurl, …
  - TopBraid Suite, Virtuoso environment, Falcon, Drupal 7, Redland, Pellet, …
  - Disco, Oracle 11g, RacerPro, IODT, Ontobroker, OWLIM, Tallis Platform, …
  - RDF Gateway, RDFLib, Open Anzo, DartGrid, Zitgist, Ontotext, Protégé, …
  - Thetus publisher, SemanticWorks, SWI-Prolog, RDFStore…
  - …

Tools

- Worth noting: major companies offer (or will offer) Semantic Web tools or systems using Semantic Web: Adobe, Oracle, IBM, HP, Software AG, webMethods, Northrop Gruman, Altova, Dow Jones, BBN, …
- See also the W3C Wiki page on tools

Deployment, applications

- See the separate slide set…

Conclusions

- The Semantic Web is there to integrate data on the Web
- The goal is the creation of a *Web of Data*

Thank you for your attention!

These slides are also available on the Web:

http://www.w3.org/People/Ivan/CorePresentations/SWTutorial/

Appendix

- There is a separate slide set on some of the formal semantics of RDF(S) and OWL…