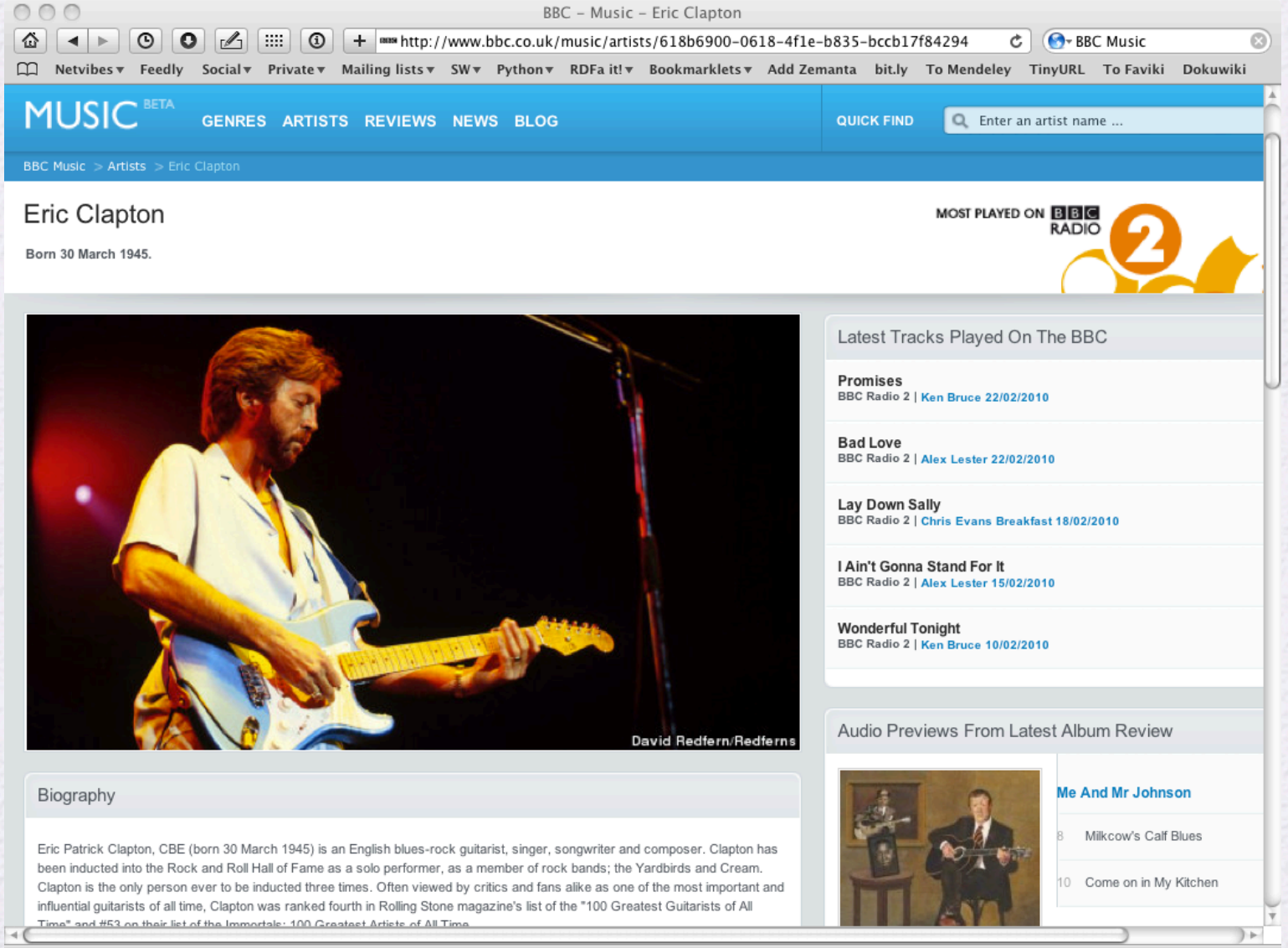
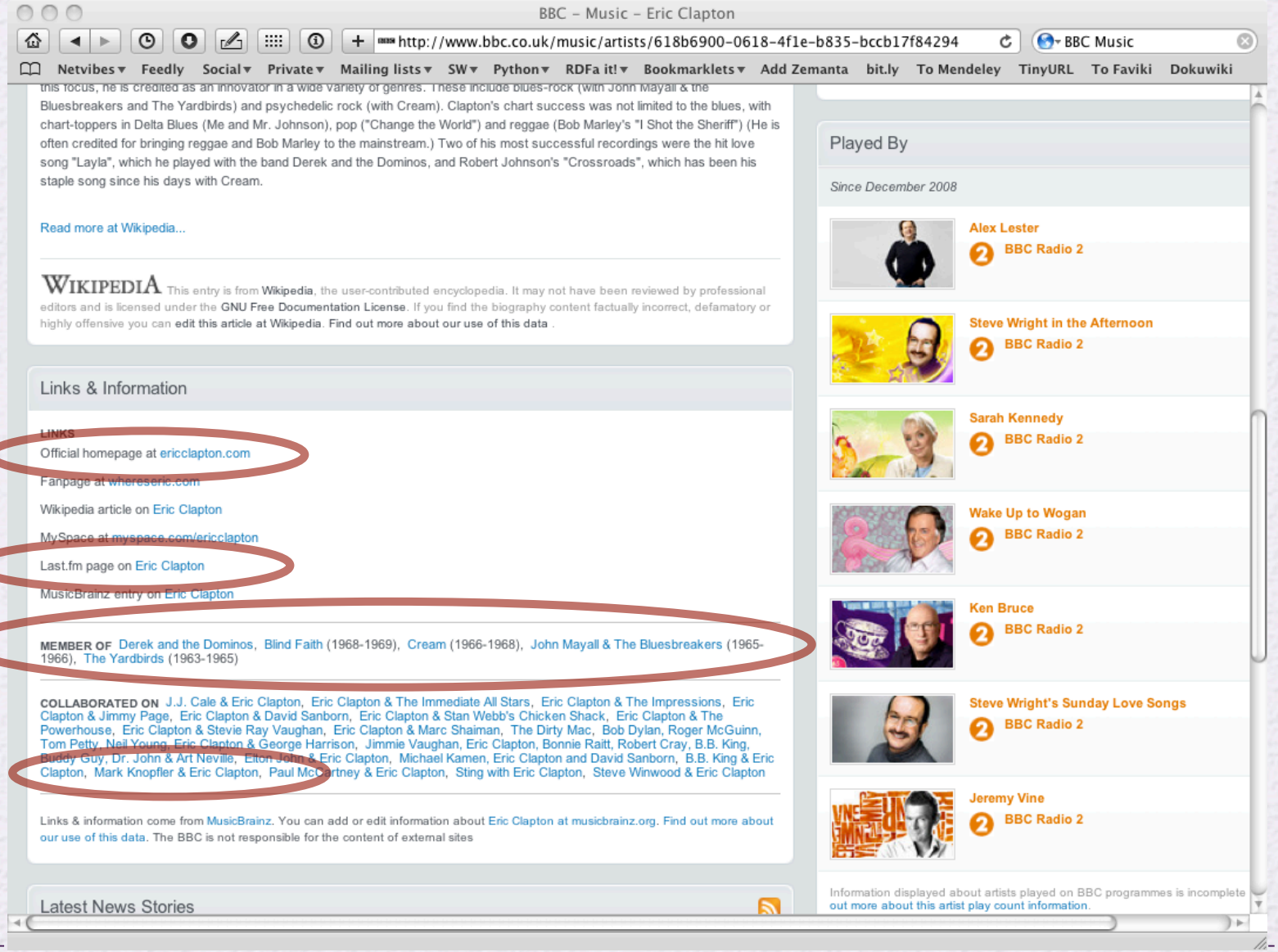
Introduction to Semantic Web Technologies

Ivan Herman, W3C
June 22nd, 2010

The Music site of the BBC



The Music site of the BBC



How to build such a site 1.

- Site editors roam the Web for new facts
  - may discover further links while roaming
- They update the site manually
- And the site gets soon out-of-date 

How to build such a site 2.

- Editors roam the Web for new data published on Web sites
- "Scrape" the sites with a program to extract the information
  - ie, *write some code* to incorporate the new data
- Easily get out of date again… 

How to build such a site 3.

- Editors roam the Web for new data via API-s
- Understand those…
  - input, output arguments, datatypes used, etc
- *Write some code* to incorporate the new data
- Easily get out of date again… 

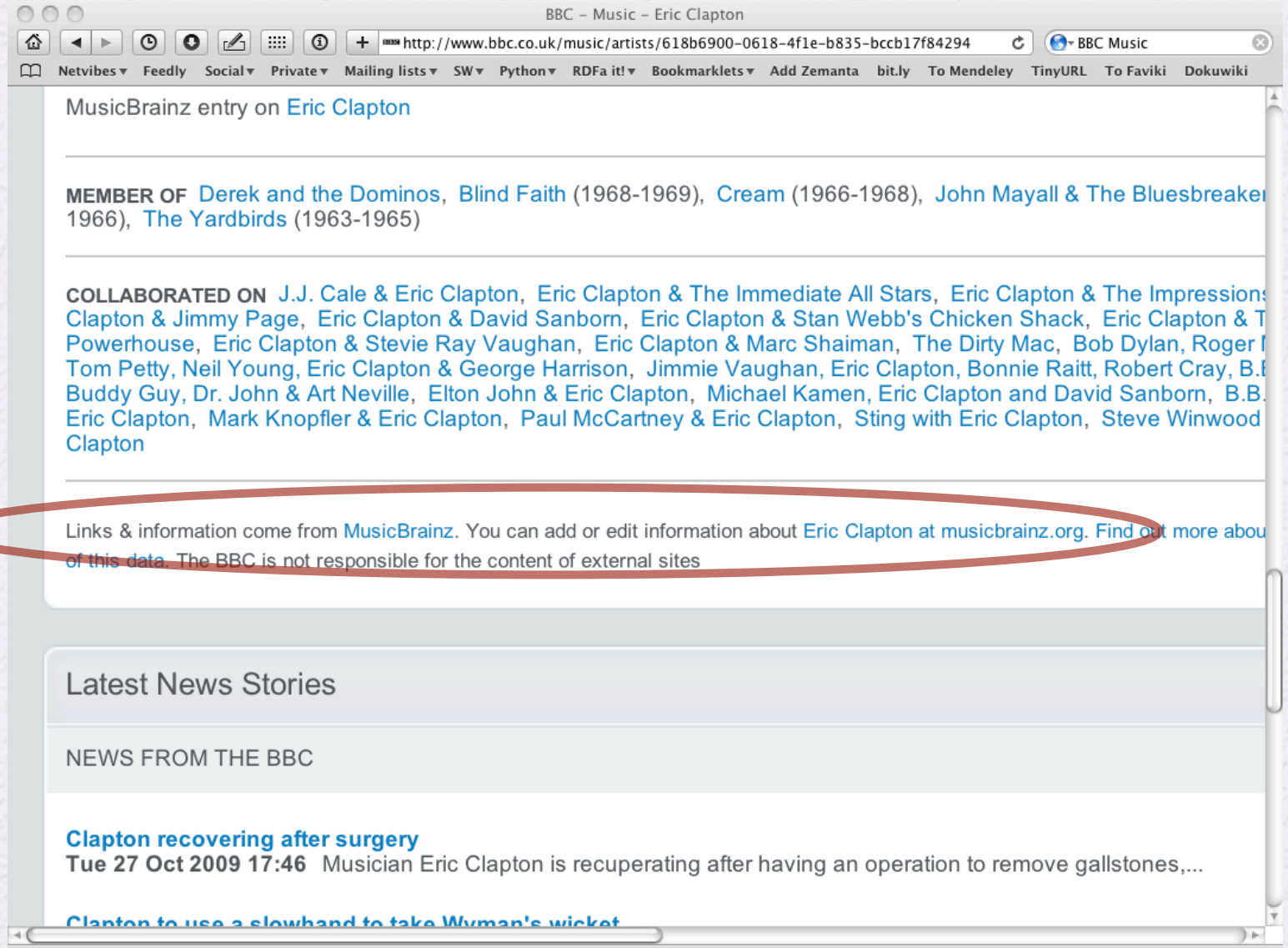
The choice of the BBC

- Use external, public datasets
  - Wikipedia, MusicBrainz, …
- They are available *as data*
  - not API-s or hidden on a Web site
  - data can be extracted using, eg, HTTP requests or standard queries

In short…

- Use the Web of Data as a Content Management System
- Use the community at large as content editors

And this is no secret...



Data on the Web

- There are more an more data on the Web
  - government data, health related data, general knowledge, company information, flight information, restaurants,…
- More and more applications rely on the availability of that data

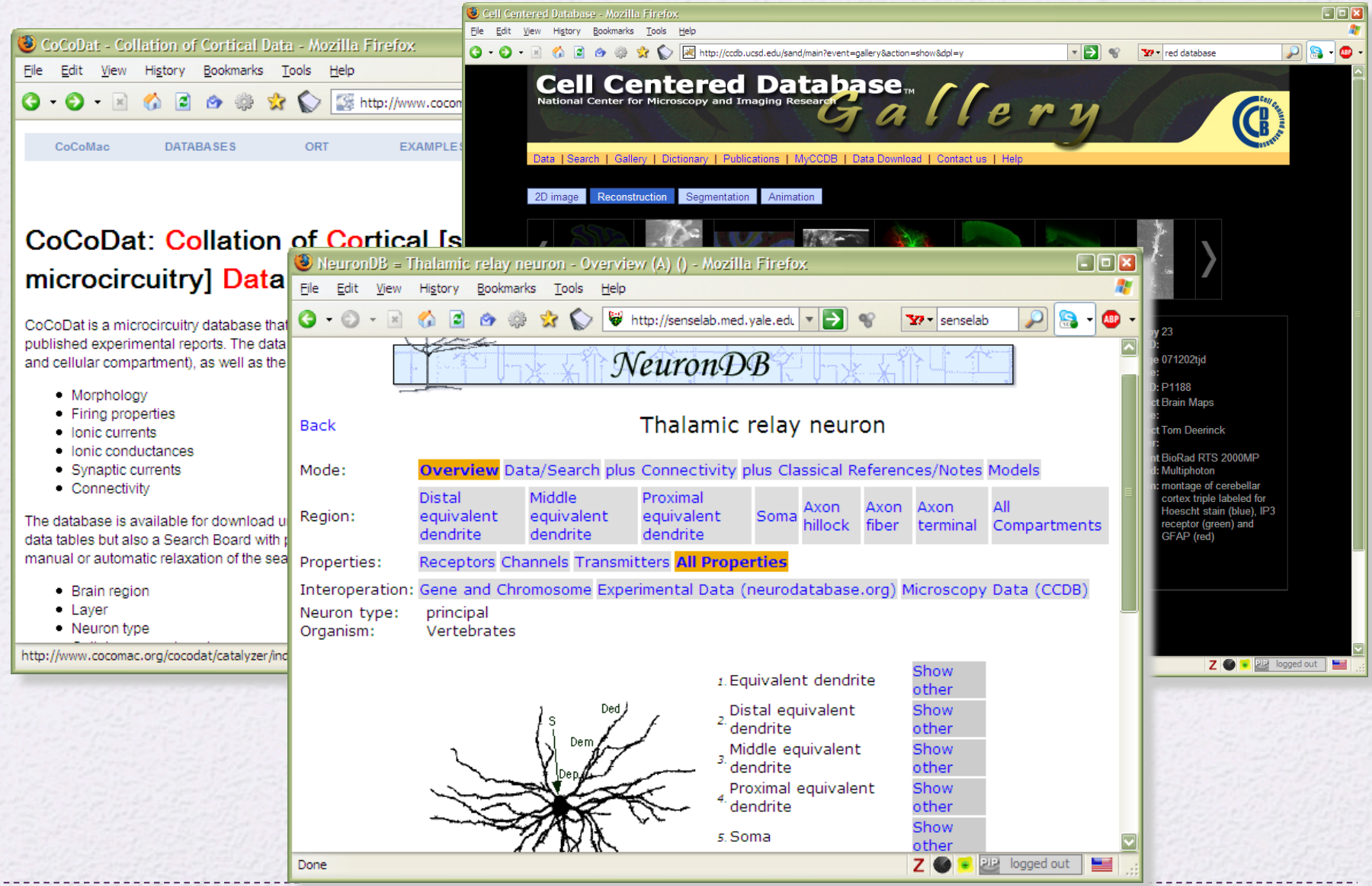
But… data are often in isolation, “silos”

Photo credit Alex (ajagendorf25), Flickr

Imagine…

- A "Web" where
  - documents are available for download on the Internet
  - but there would be no hyperlinks among them

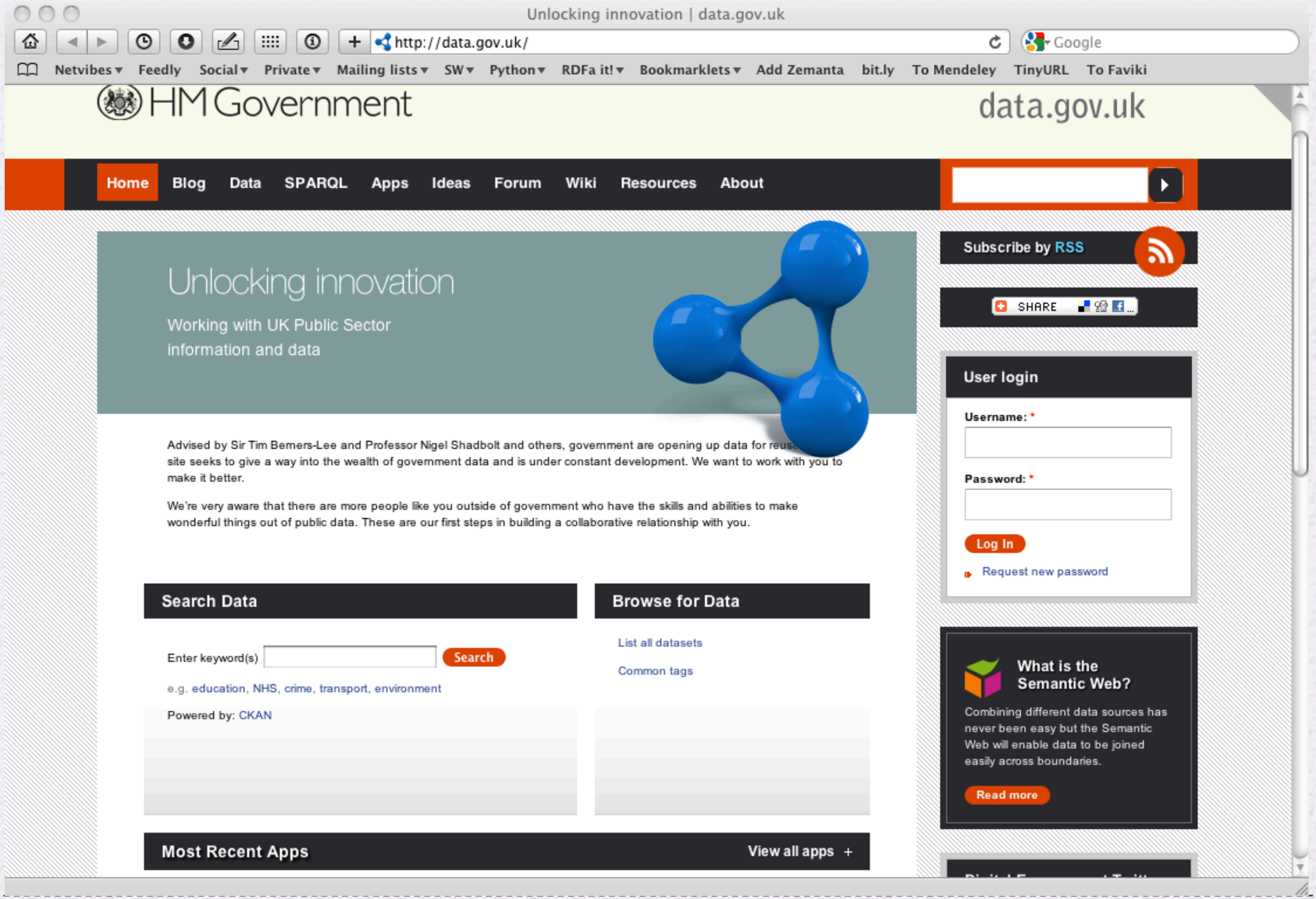
And the problem *is* real...



Data on the Web is not enough…

- We need a proper infrastructure for a real *Web of Data*
  - data is available on the Web
    - accessible via standard Web technologies
  - data are interlinked over the Web
  - ie, data can be *integrated* over the Web
- This is where Semantic Web technologies come in

A Web of Data unleashes now applications



A nice usage of UK government data

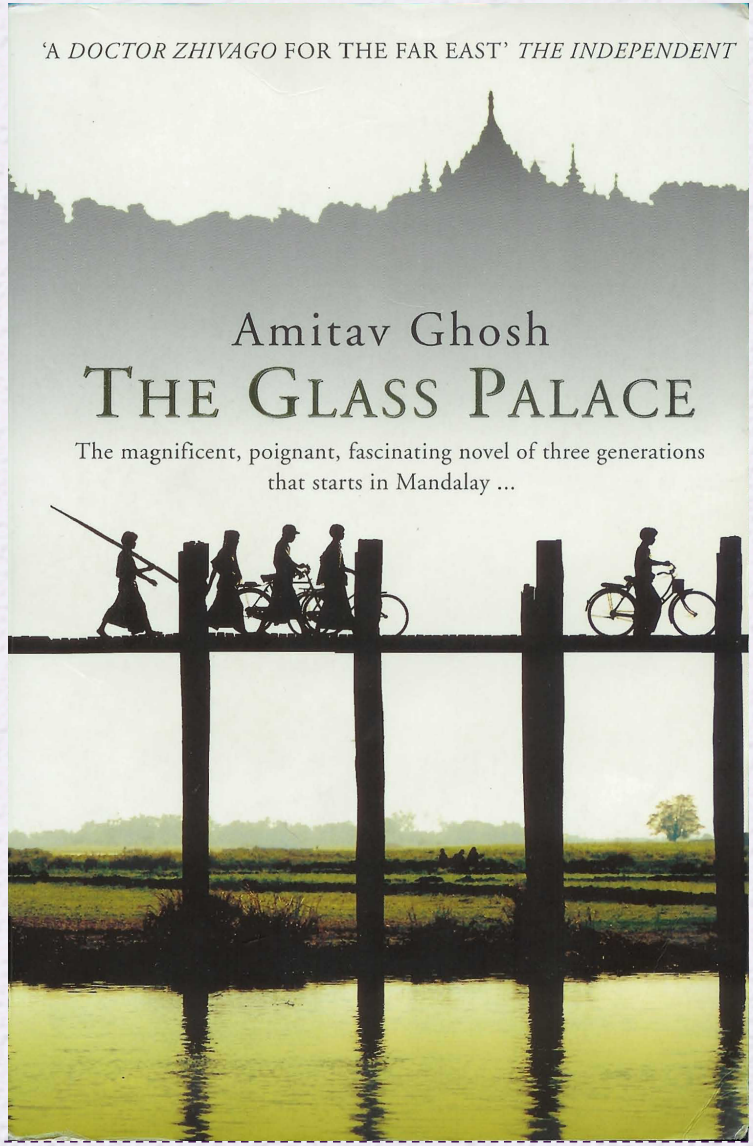
In what follows…

- We will use a simplistic example to introduce the main Semantic Web concepts

The rough structure of data integration

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

We start with a book...



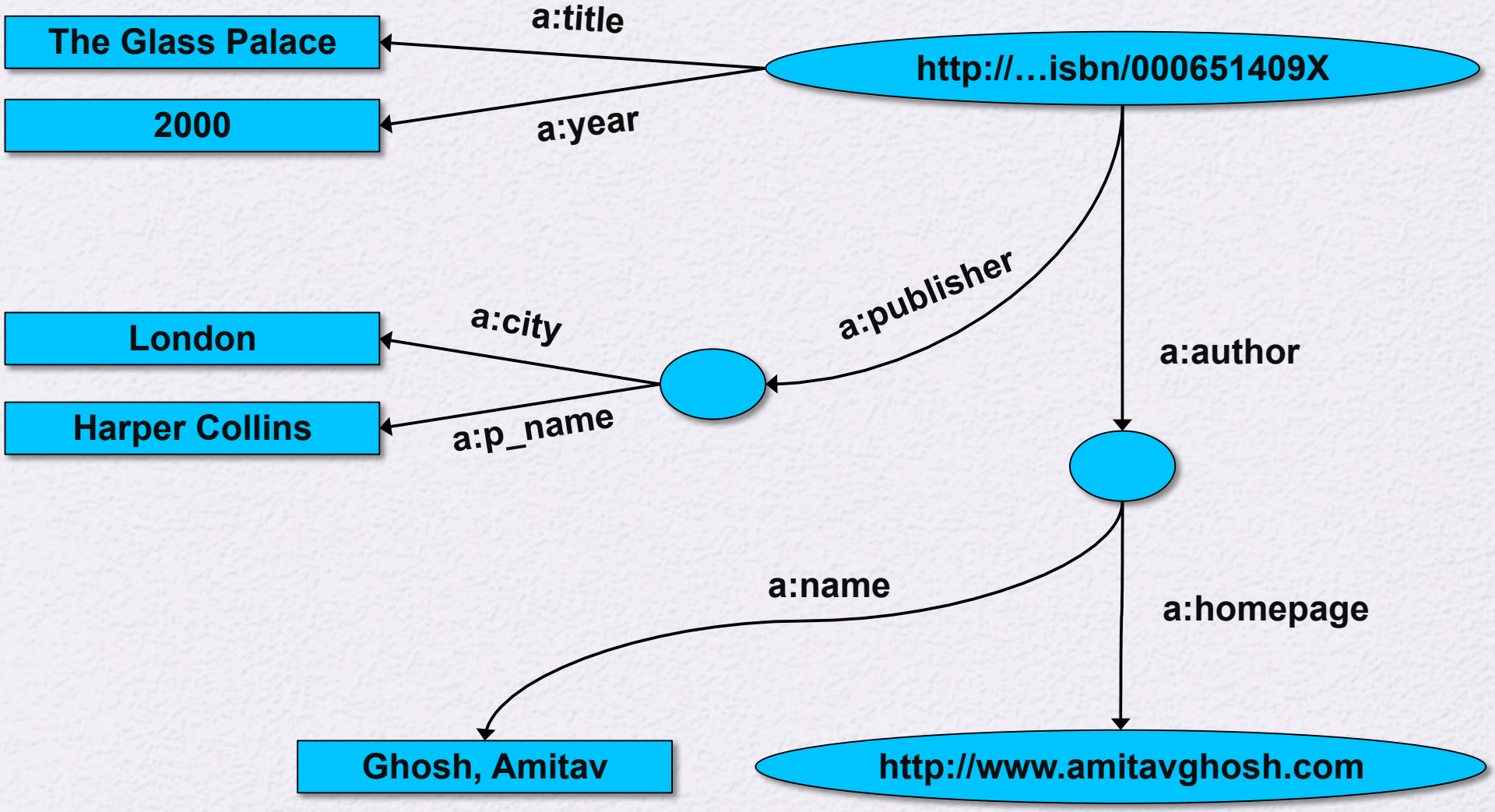
A simplified bookstore data (dataset “A”)

ID	Author	Title	Publisher	Year
ISBN 0-00-6511409-X	id_xyz	The Glass Palace	id_qpr	2000

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

ID	Publisher’s name	City
id_qpr	Harper Collins	London

1st: export your data as a set of *relations*



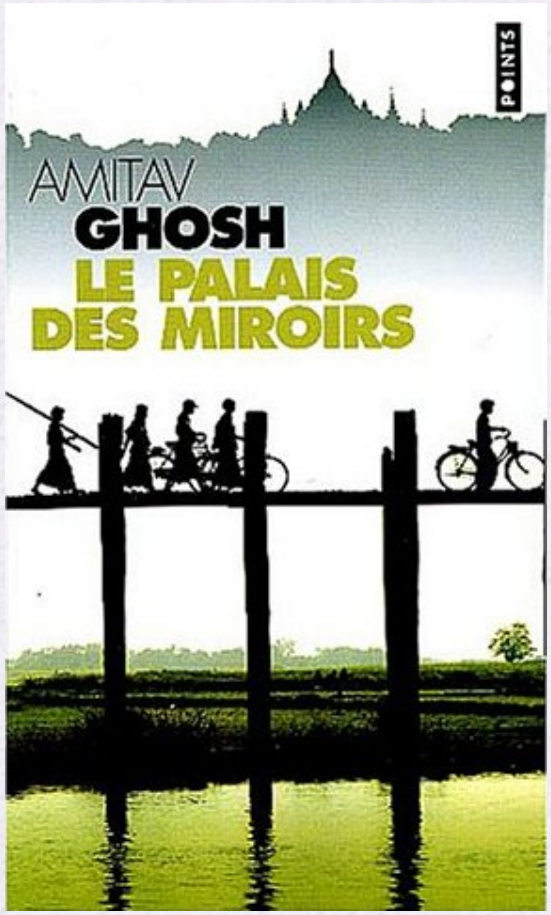
Some notes on the exporting the data

- 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

Some notes on the exporting the data

- 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

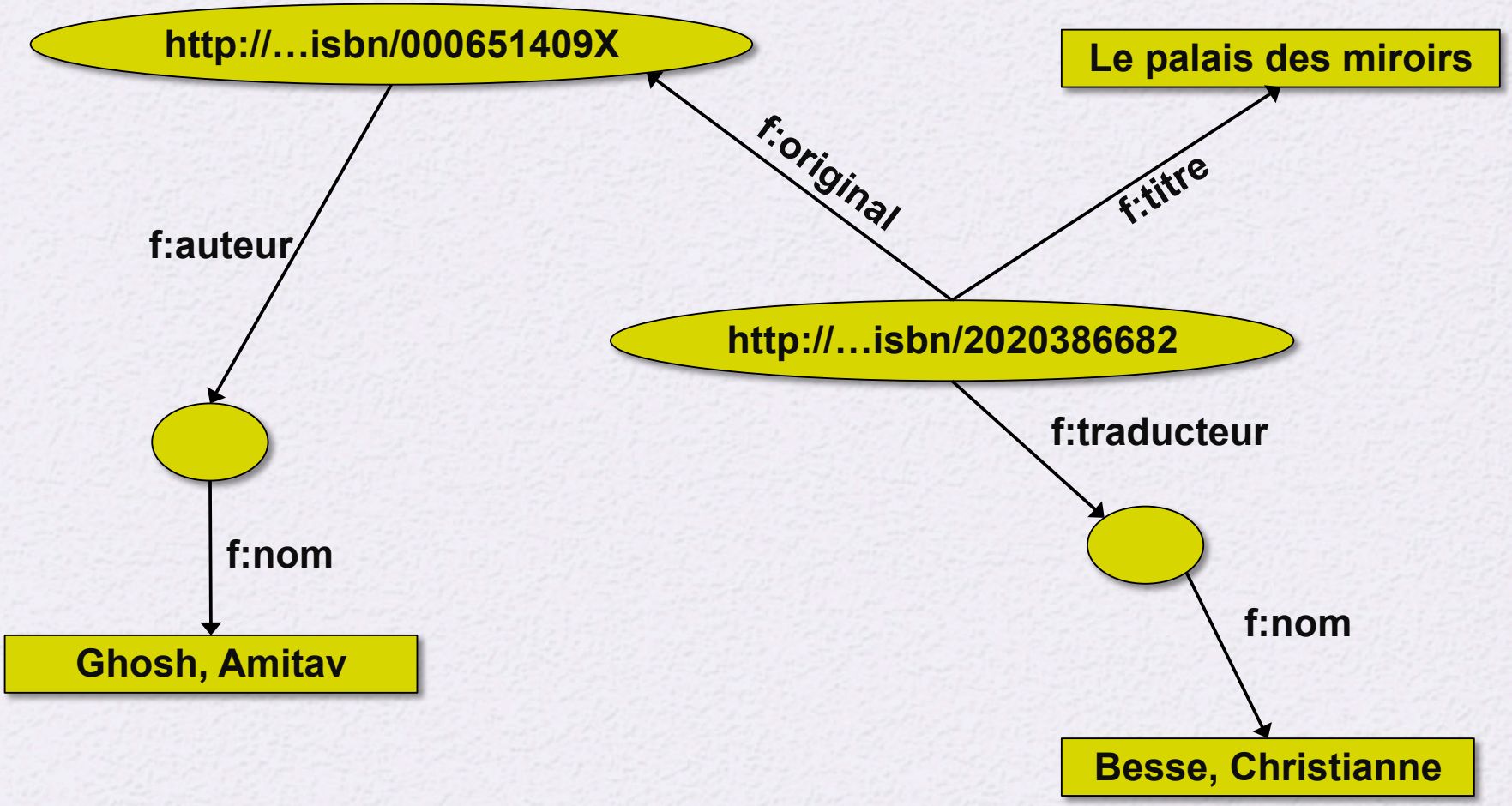
Same book in French…



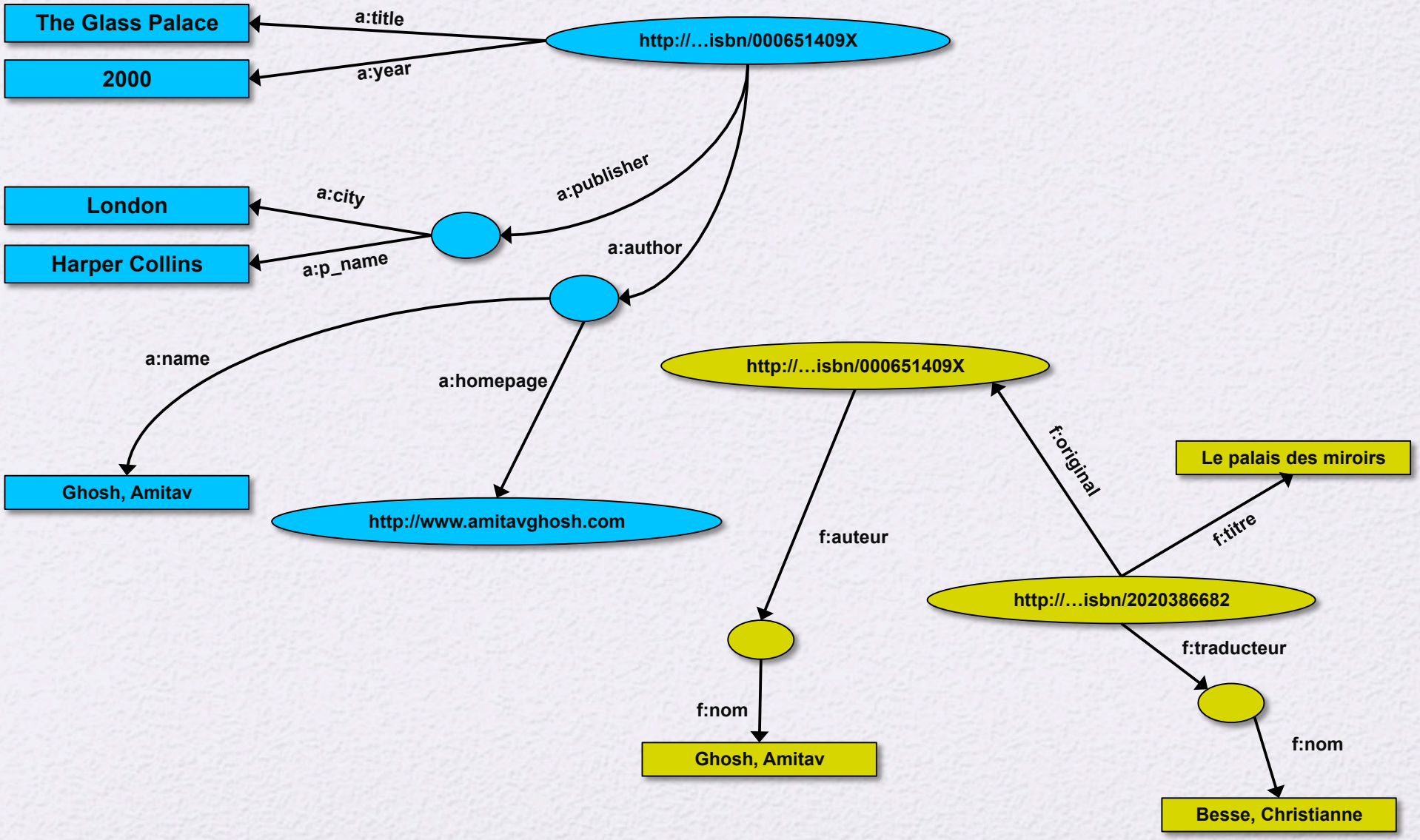
Another bookstore data (dataset “F”)

	A	B	C	D
1	**ID**	**Titre**	**Traducteur**	**Original**
2	ISBN 2020286682	Le Palais des Miroirs	$A12$	ISBN 0-00-6511409-X
3				
4				
5				
6	**ID**	**Auteur**		
7	ISBN 0-00-6511409-X	$A11$		
8				
9				
10	**Nom**			
11	Ghosh, Amitav			
12	Besse, Christianne			

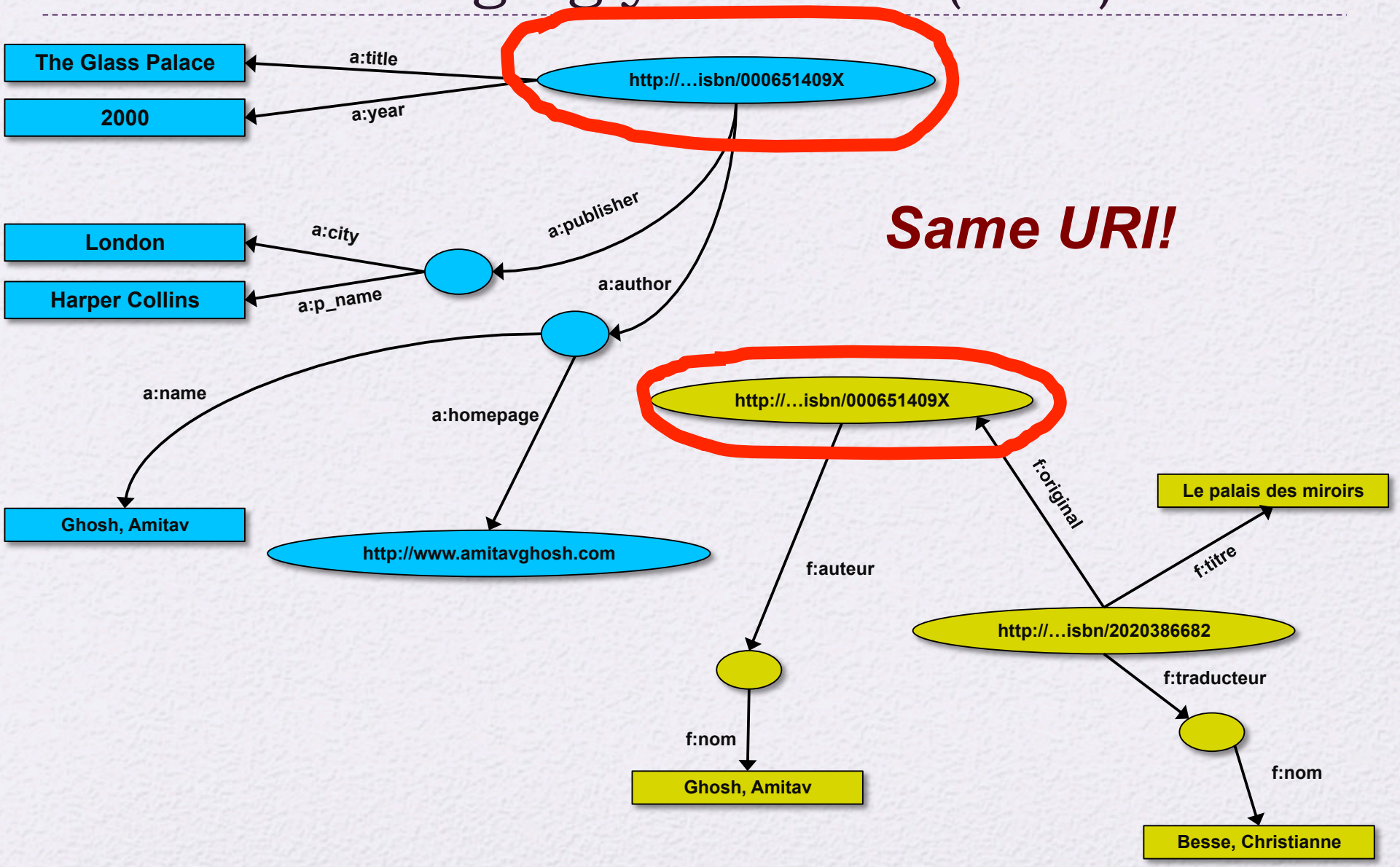
2nd: export your second set of data



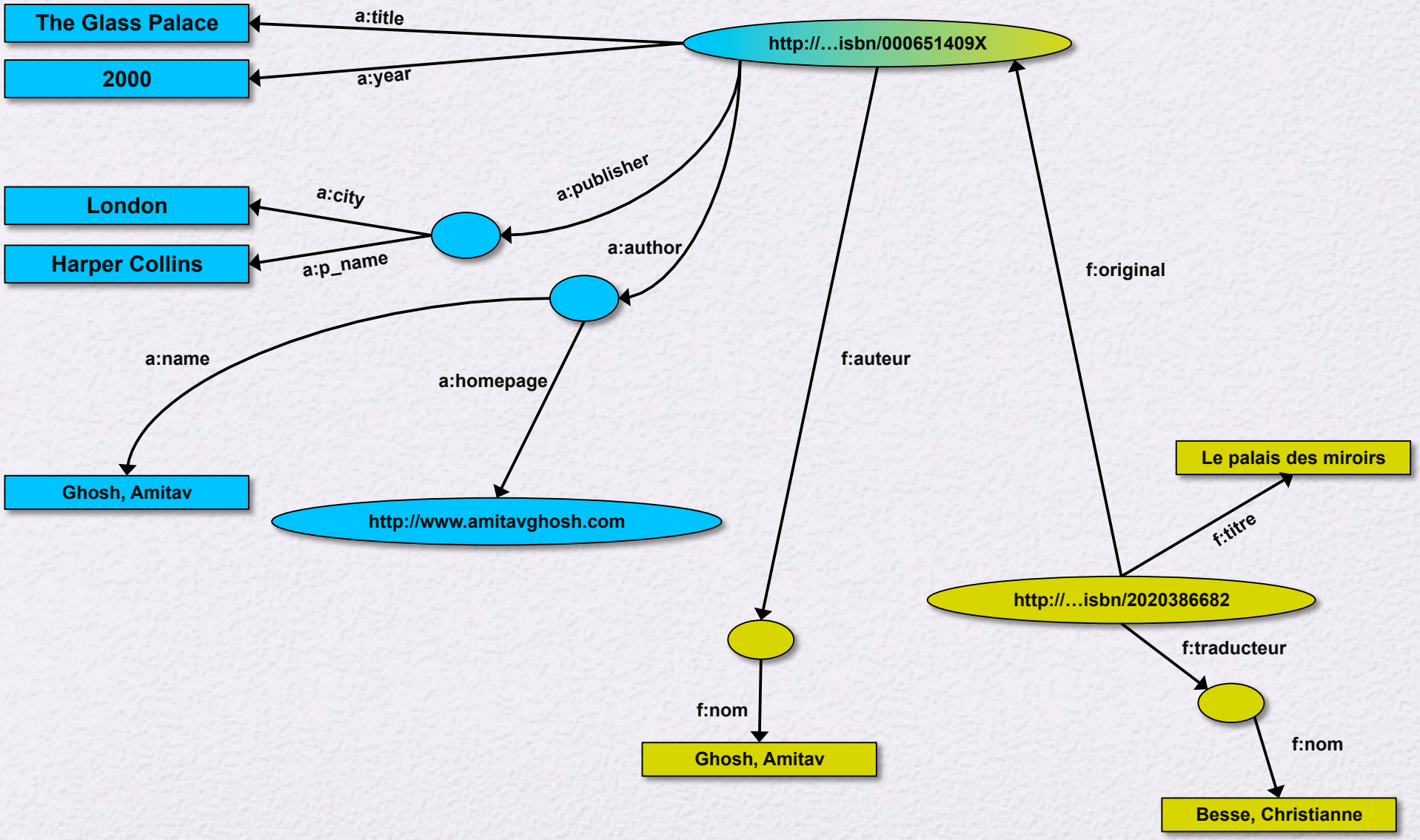
3rd: start merging your data



3rd: start merging your data (cont)

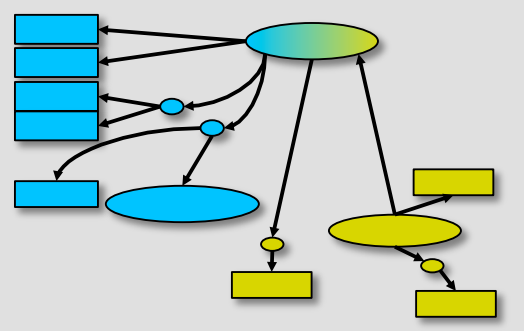


3rd: start merging your data



Start making queries…

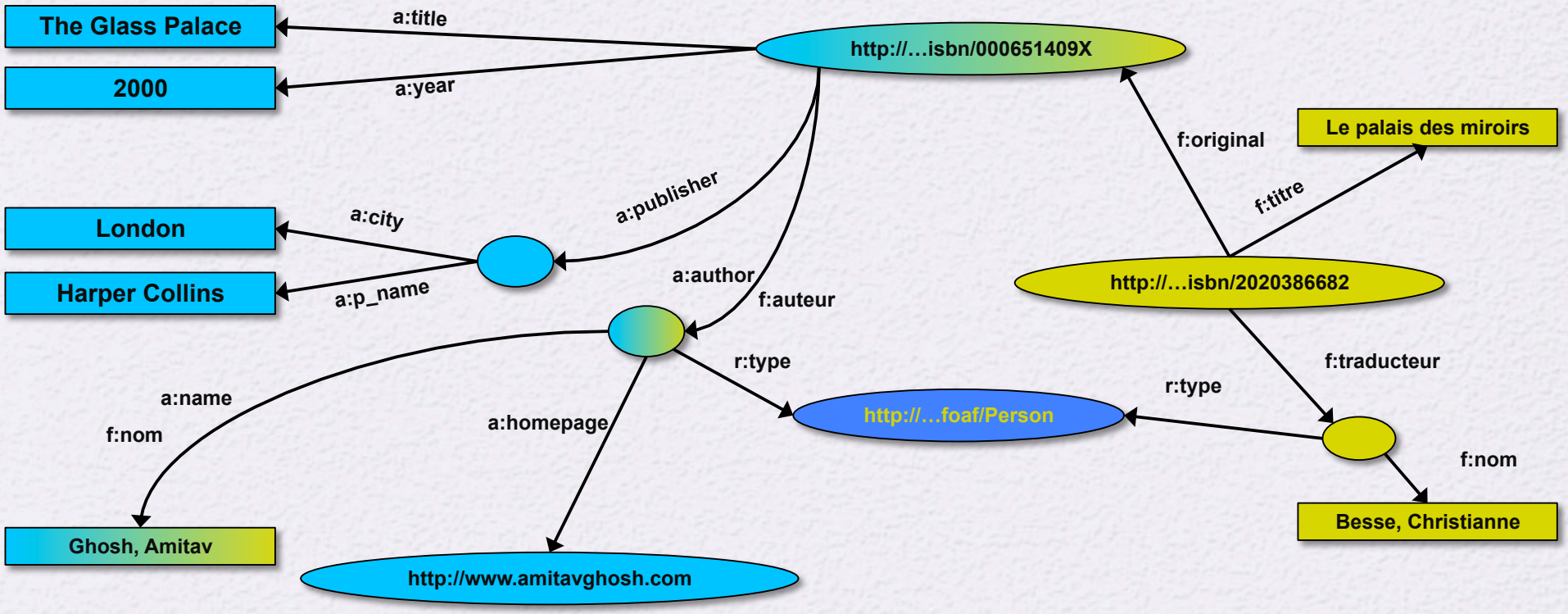
- User of data "F" can now ask queries like:
  - "give me the title of the original"
    - well, … « donnes-moi le titre de l'original »
- This information is not in the dataset "F"…
- …but can be retrieved by merging with dataset "A"!

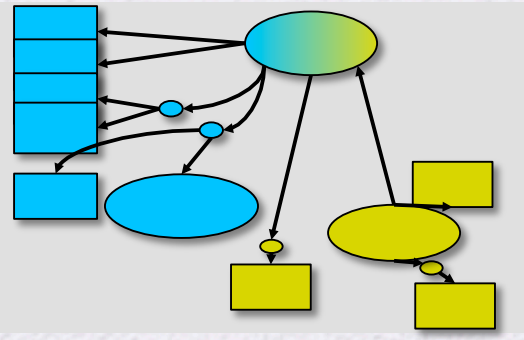


However, more can be achieved…

- 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

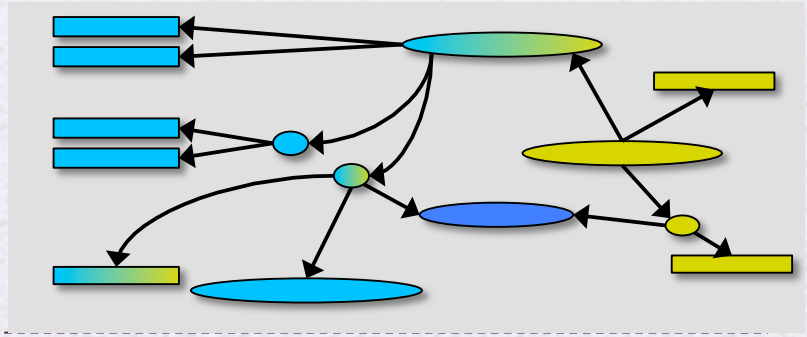
3rd revisited: use the extra knowledge





Start making richer queries!

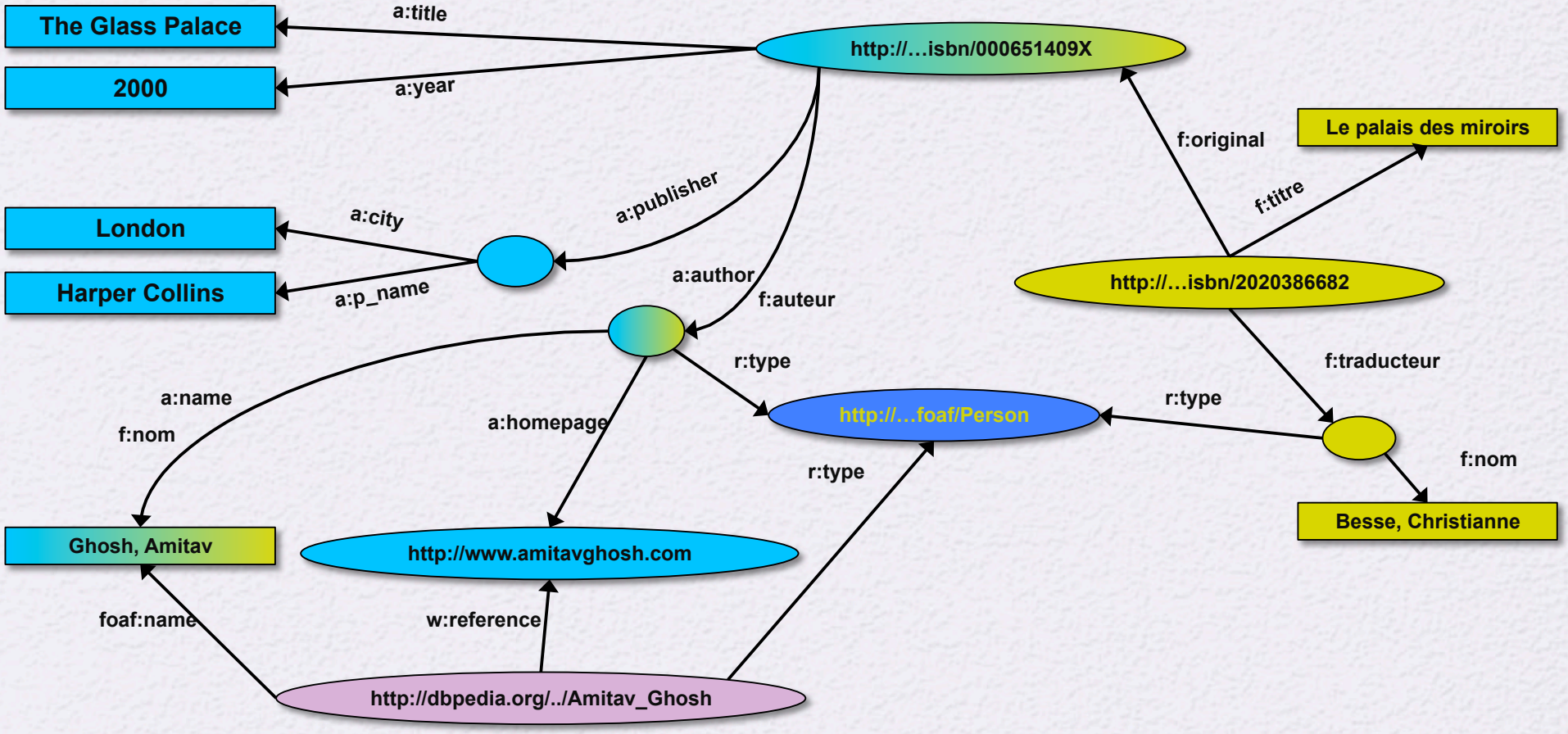
- User of dataset "F" can now query:
  - "donnes-moi la page d'accueil de l'auteur de l'original"
    - well… "give me the home page of the original's 'auteur'"
- The information is not in datasets "F" or "A"…
- …but was made available by:
  - merging datasets "A" and datasets "F"
  - adding three simple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



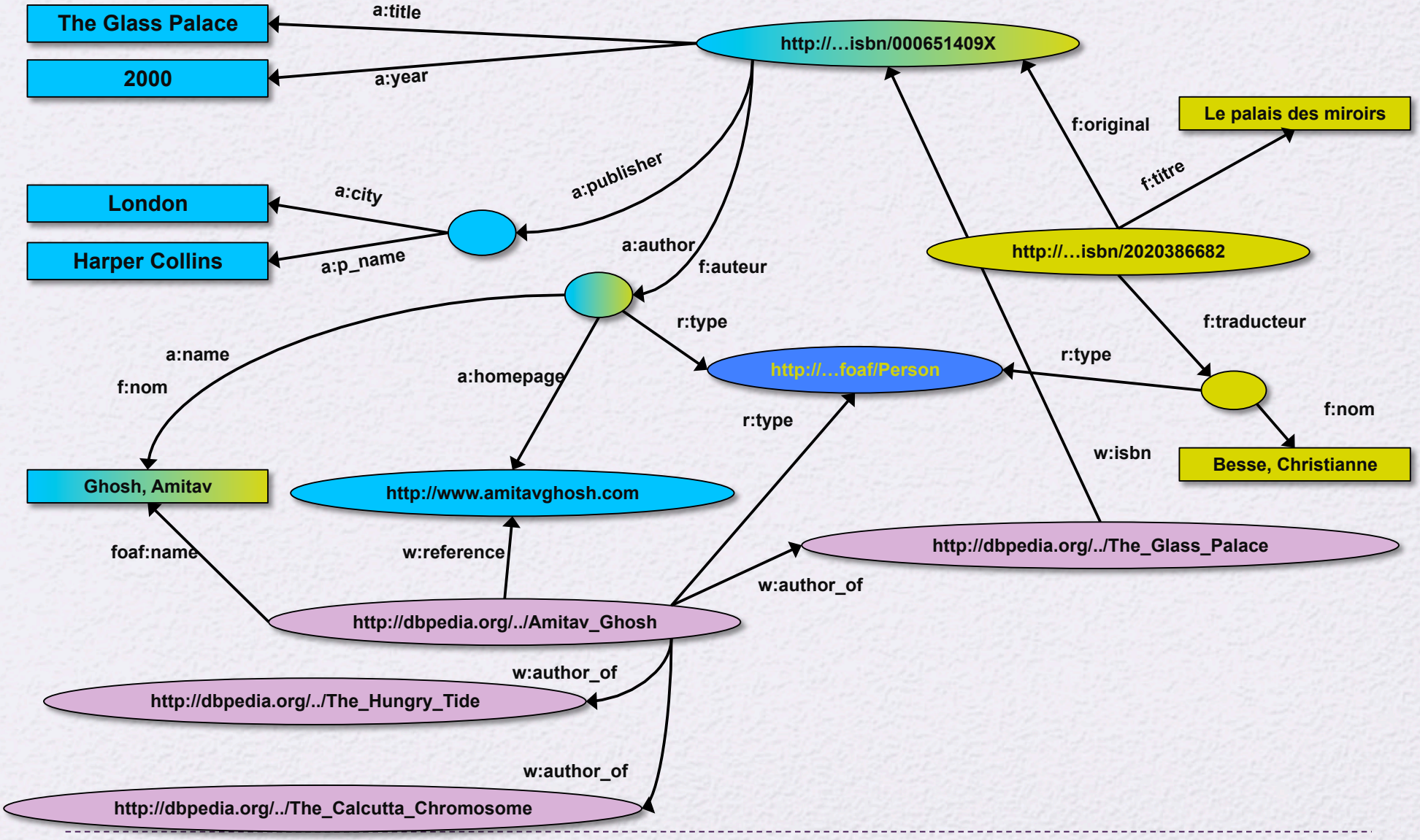
Combine with different datasets

- 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…

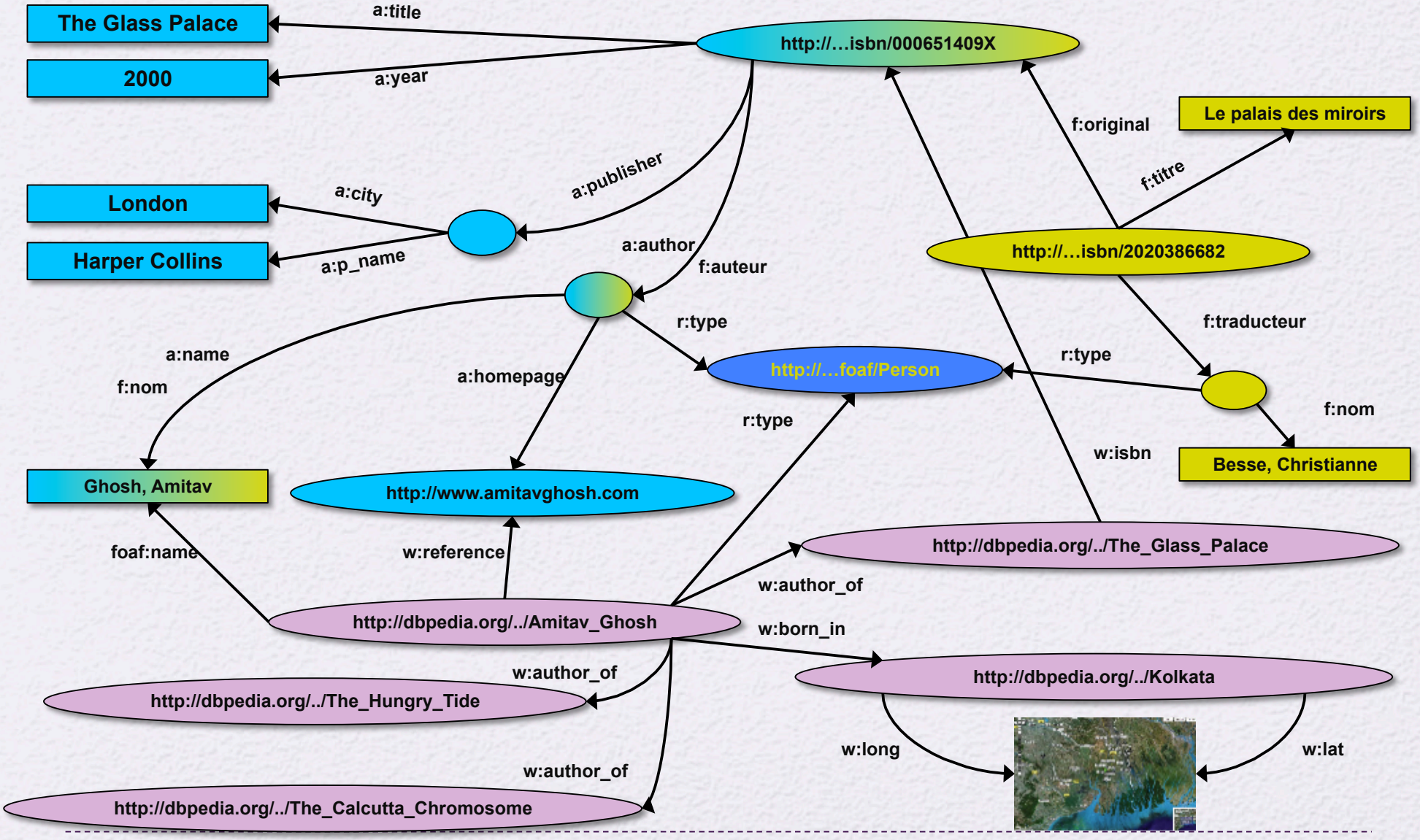
Merge with Wikipedia data



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

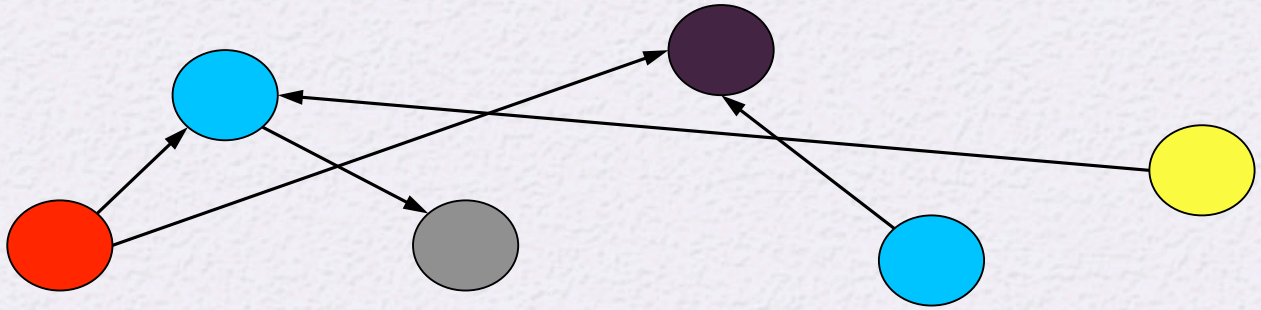
It could become even more powerful

- 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?

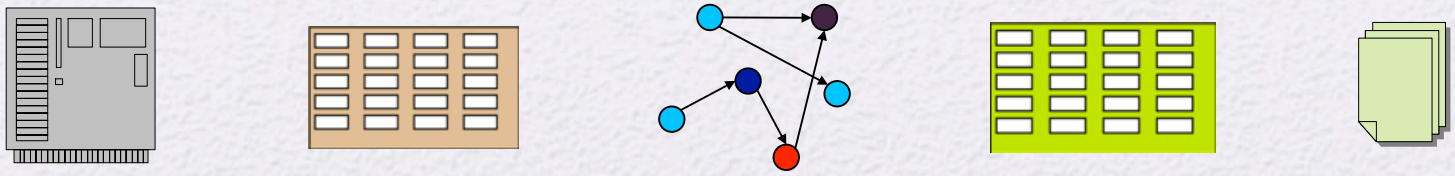
Applications

Manipulate
Query
…



Data represented in abstract format

Map,
Expose,
…



Data in various formats

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:

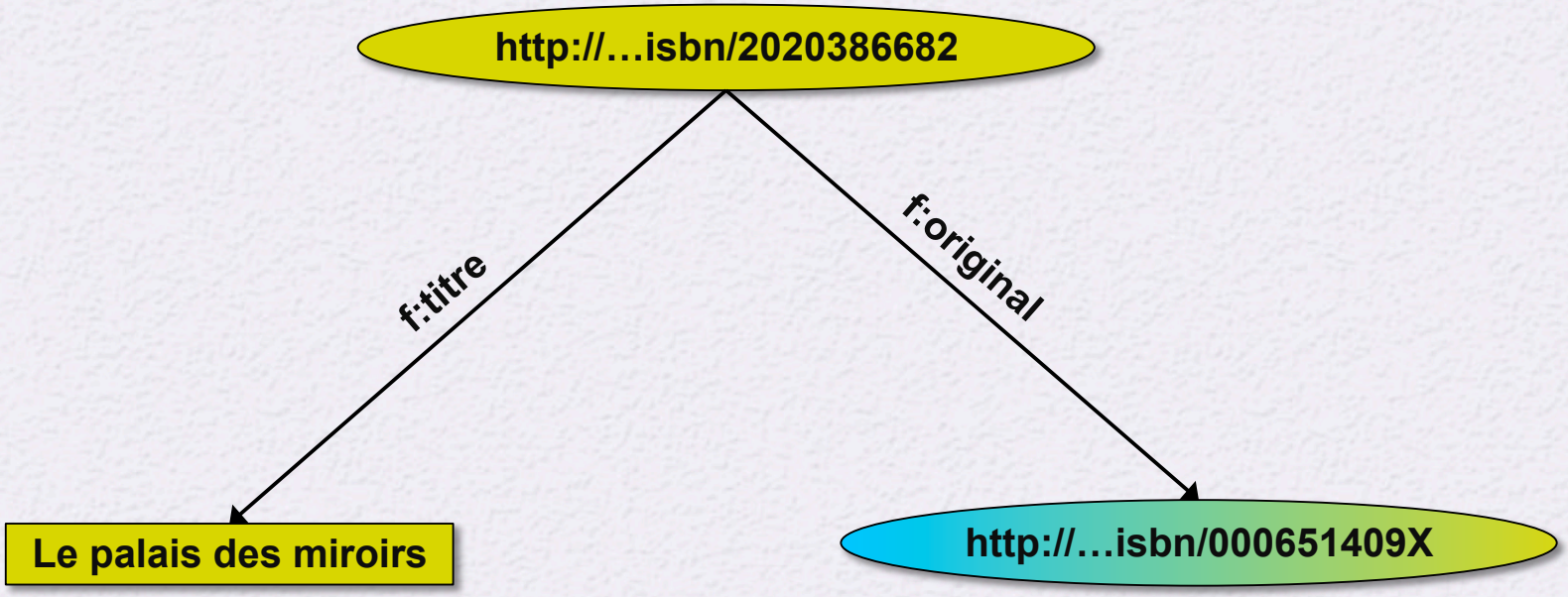
```
(<http://…isbn…6682>, <http://…/original>, <http://…isbn…409X>)
```

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

RDF triples (cont.)

- Resources can use *any* URI
  - http://www.example.org/file.html#home
  - http://www.example.org/file2.xml#xpath(//q[@a=b])
  - http://www.example.org/form?a=b&c=d
- RDF triples form a directed, labeled graph (the best way to think about them!)

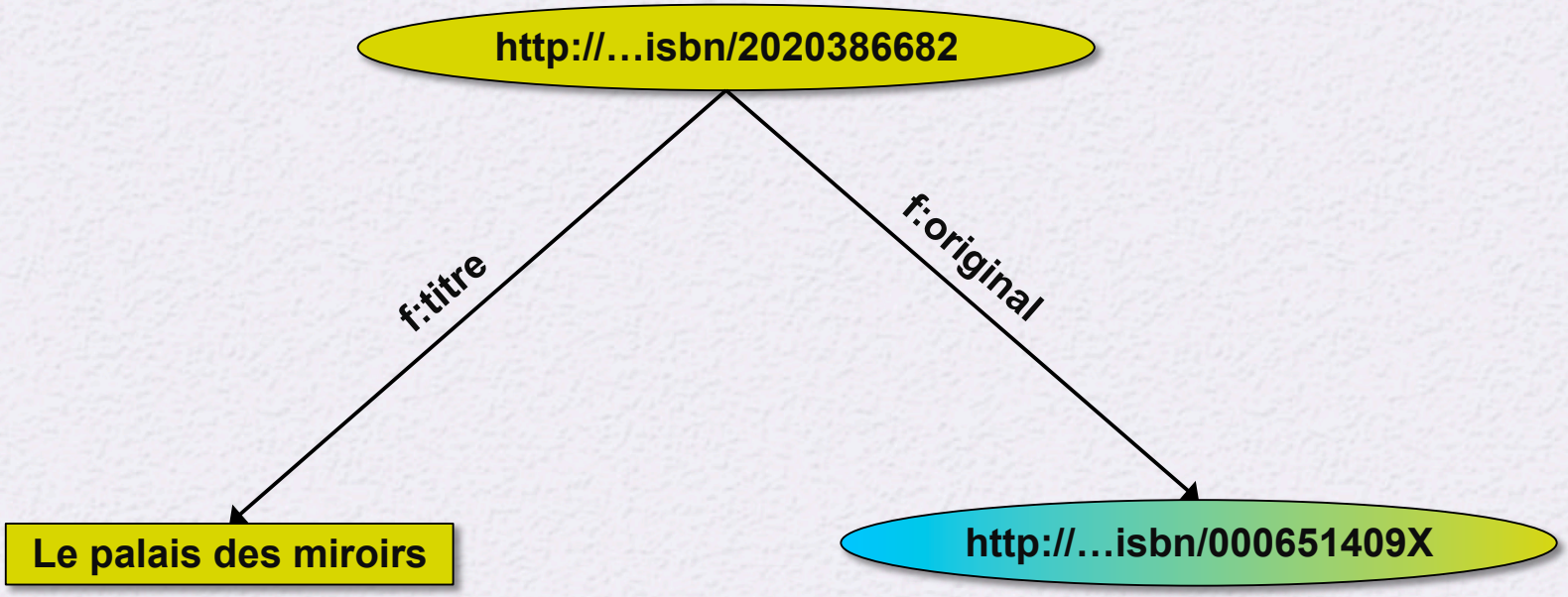
A simple RDF example (in RDF/XML)




```
<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>
```

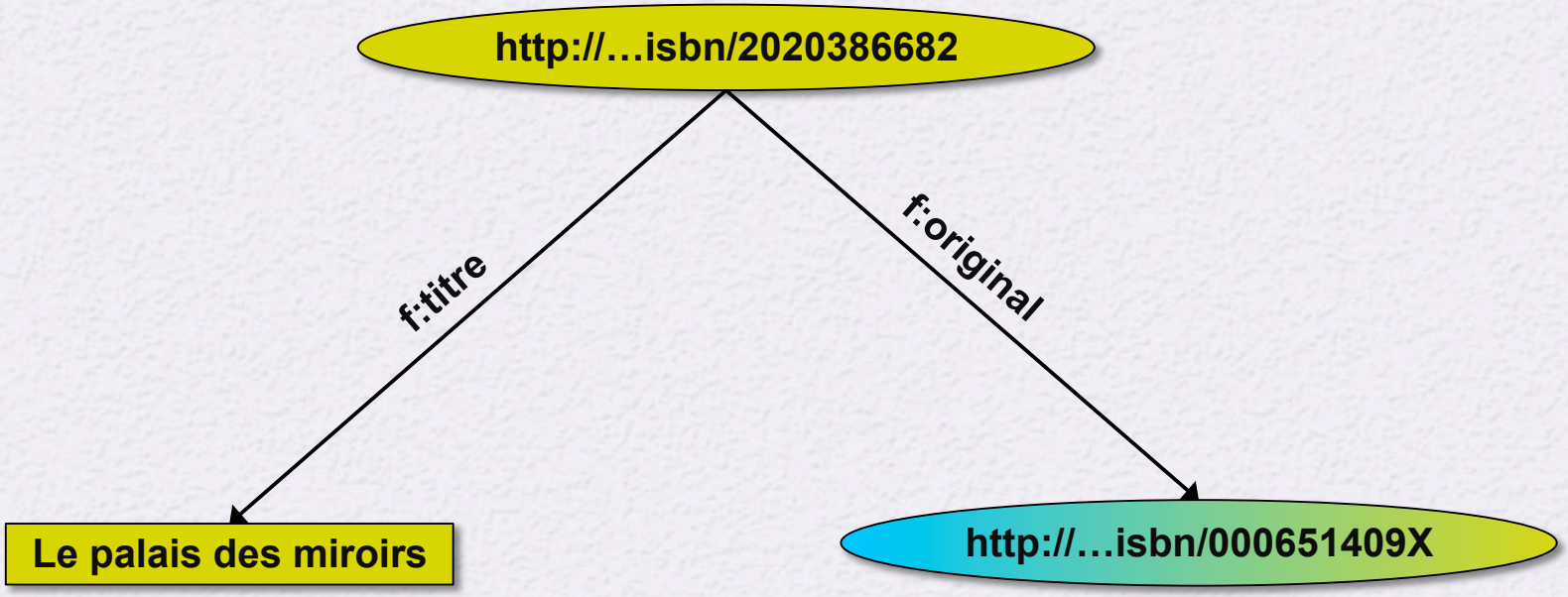
(Note: namespaces are used to simplify the URI-s)

A simple RDF example (in Turtle)




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

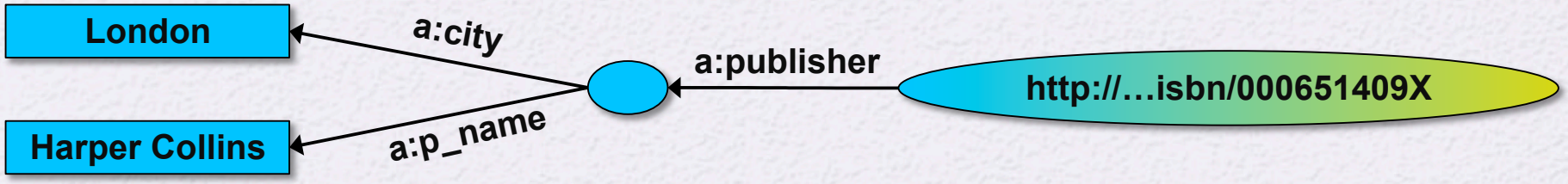
A simple RDF example (in RDFa)




```
<p about="http://…/isbn/2020386682">The book entitled
“<span property="f:title" lang="fr">Le palais des mirroirs</span>”
is the French translation of the
“<span rel="f:original" resource="http://…/isbn/000651409X">Glass
Palace</span>”</p> .
```

"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

- The resource will be "visible" on the Web
  - care should be taken to define unique URI-s

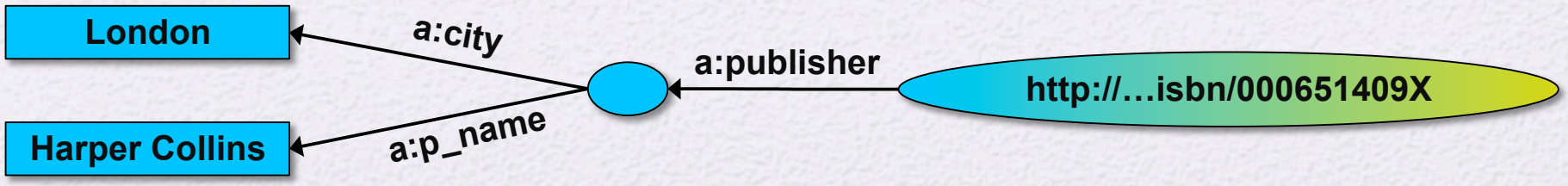
```
<rdf:Description rdf:about="http://…/isbn/000651409X">
   <a:publisher rdf:resource="urn:uuid:f60ffb40-307d-…"/>
</rdf:Description>
<rdf:Description rdf:about="urn:uuid:f60ffb40-307d-…">
   <a:p_name>HarpersCollins</a:p_name>
   <a:city>HarpersCollins</a:city>
</rdf:Description>
```

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".
```

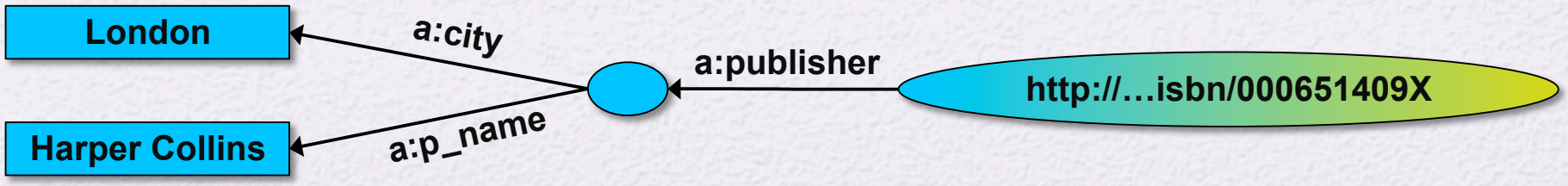
Internal = these resources are *not* visible outside



Blank nodes: the system can do it

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

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



Blank nodes when merging

- Blank nodes require attention when merging
  - blanks nodes with identical nodeID-s in different graphs are different
  - implementations must be careful…

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

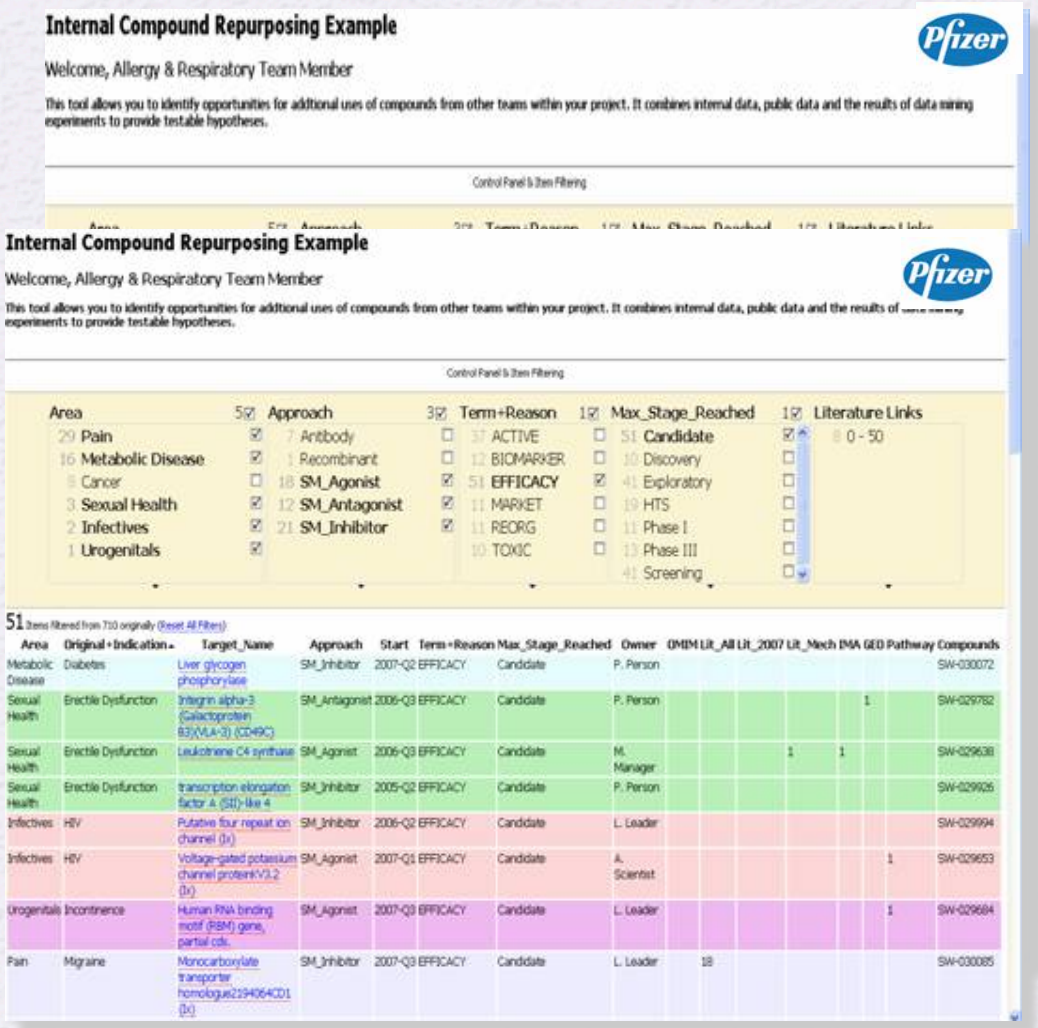
```
// create a model
Model model=new ModelMem();
Resource subject=model.createResource("URI_of_Subject")
// 'in' refers to the input file
model.read(new InputStreamReader(in));
StmtIterator iter=model.listStatements(subject,null,null);
while(iter.hasNext()) {
   st = iter.next();
   p = st.getProperty();
   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
  - merge takes care of blank node issues, too

Another relatively simple application

- Goal: reuse of older experimental data
- Keep data in databases or XML, just export key "fact" as RDF
- Use a faceted browser to visualize and interact with the result



Courtesy of Nigel Wilkinson, Lee Harland, Pfizer Ltd, Melliyal Annamalai, Oracle (SWEO Case Study)

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…

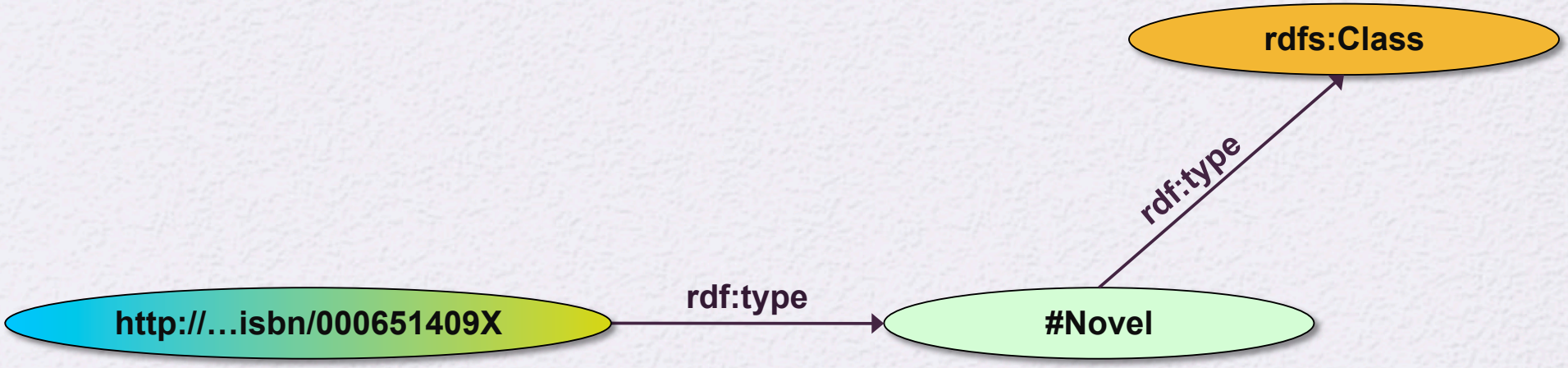
Classes, resources, …

- Think of well known traditional vocabularies:
  - 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.)

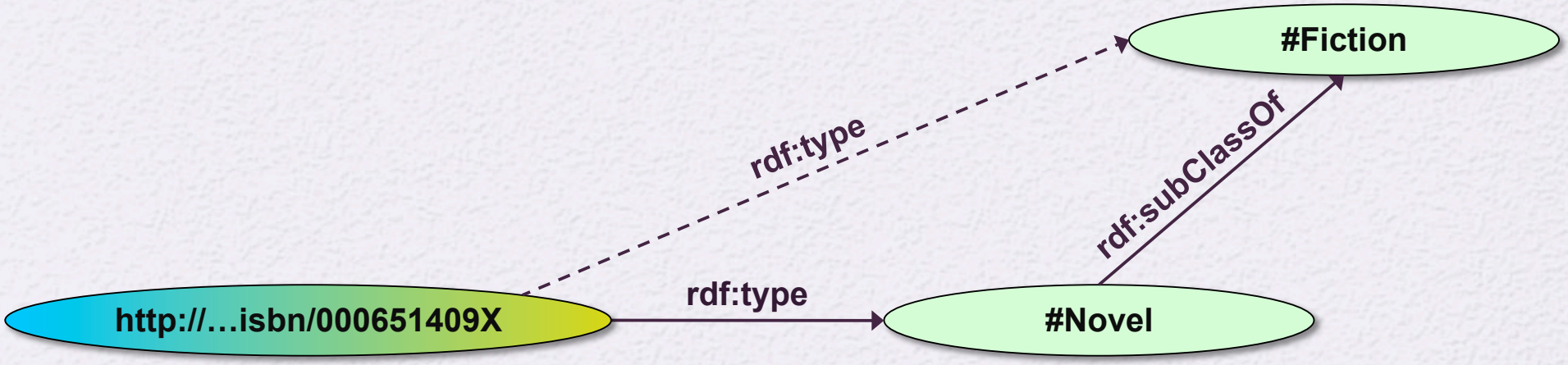
- Relationships are defined among resources:
  - "typing": an individual belongs to a specific class
    - "«The Glass Palace» is a novel"
    - to be more precise: "«http://.../000651409X» is a novel"
  - "subclassing": all instances of one are also the instances of the other ("every novel is a fiction")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

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

Example for property characterization

```
: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

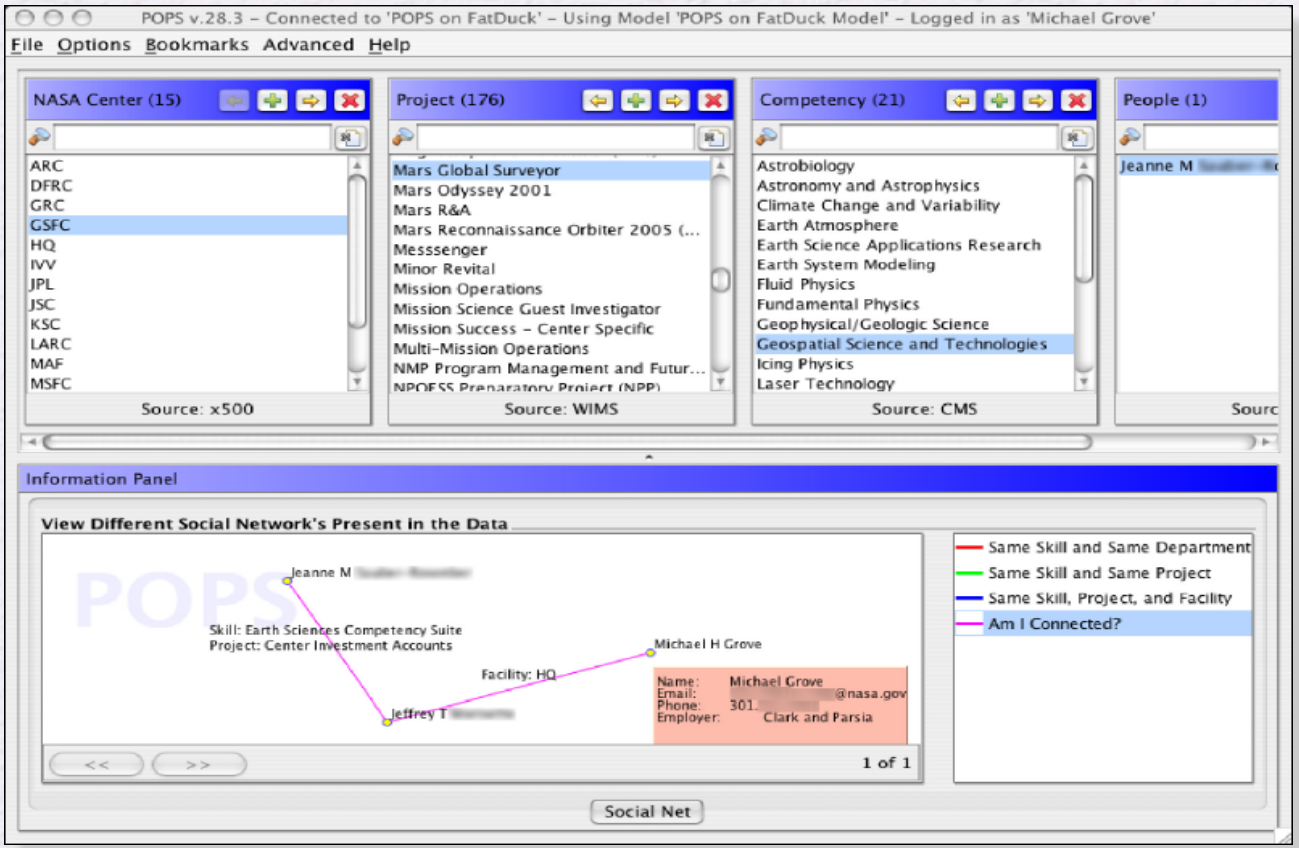
```
<http://…/isbn/000651409X>
      :page_number "543"^^xsd:integer ;
      :publ_date   "2000"^^xsd:gYear ;
      :price       "6.99"^^xsd:float .
```

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…)

Find the right experts at NASA

- Expertise locater for nearly 70,000 NASA civil servants,
  - integrate 6 or 7 geographically distributed databases, …



Michael Grove, Clark & Parsia, LLC, and Andrew Schain, NASA, (SWEO Case Study)

How to get and create RDF Data?

Simple approach

- Write RDF/XML, RDFa, 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

RDF with XML/(X)HTML (cont)

- Two solutions have emerged:
  - use microformats and convert the content into RDF
    - XSLT is the favorite approach
  - add RDF-like statements directly into XHTML via RDFa

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, …)
- W3C is working on a standard in this area

Linked Open Data

Linked Open Data Project

- Goal: "expose" open datasets in RDF
- Set RDF links among the data items from different datasets
- Set up, if possible, query endpoints

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

```
@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:leaderName dbpedia:Lodewijk_Asscher ;
   ...
   dbterm:areaTotalKm "219" ;
   ...
dbpedia:ABN_AMRO
   dbterm:location dbpedia:Amsterdam ;
   ...
```

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

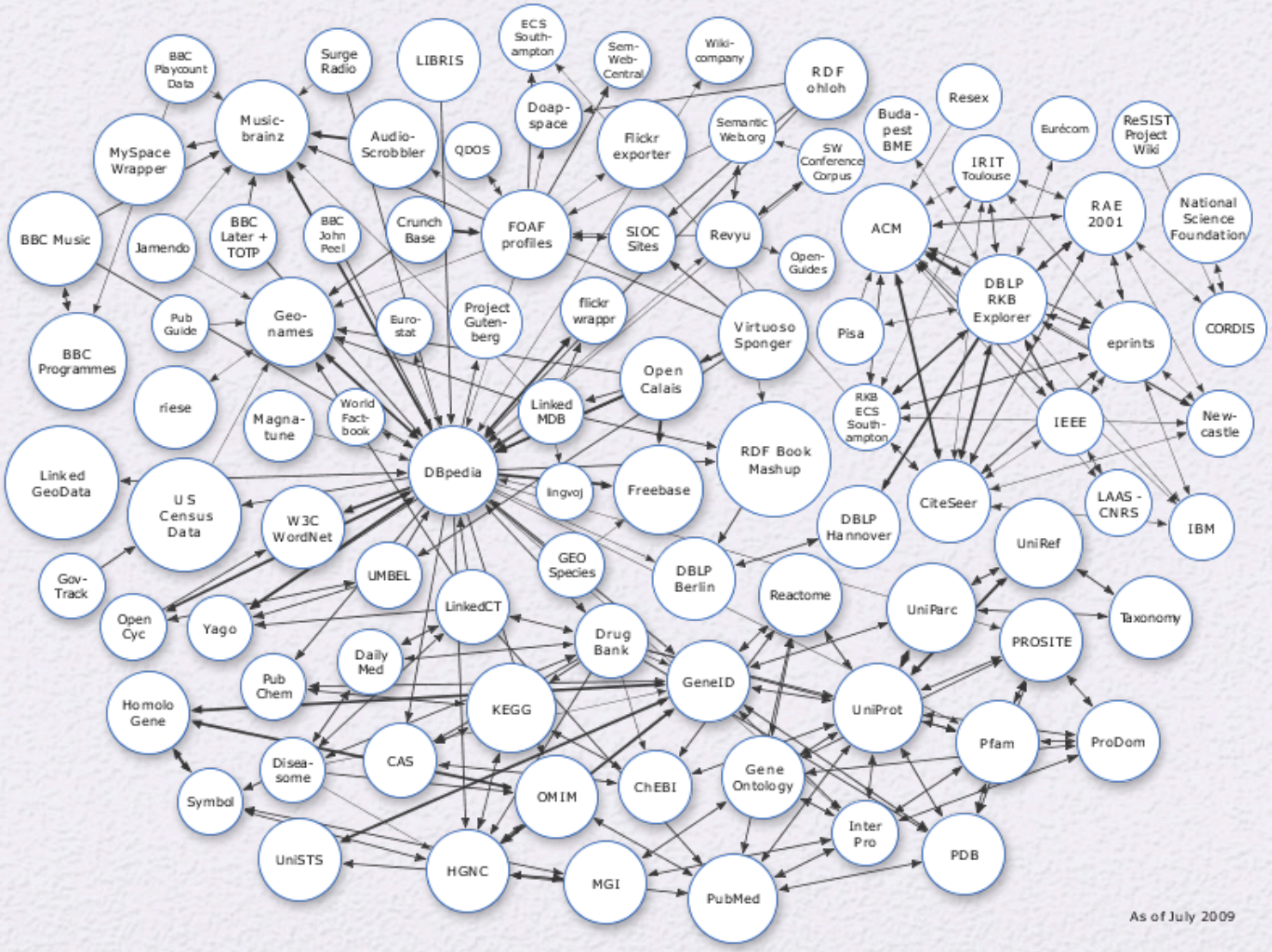
Automatic links among open datasets

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

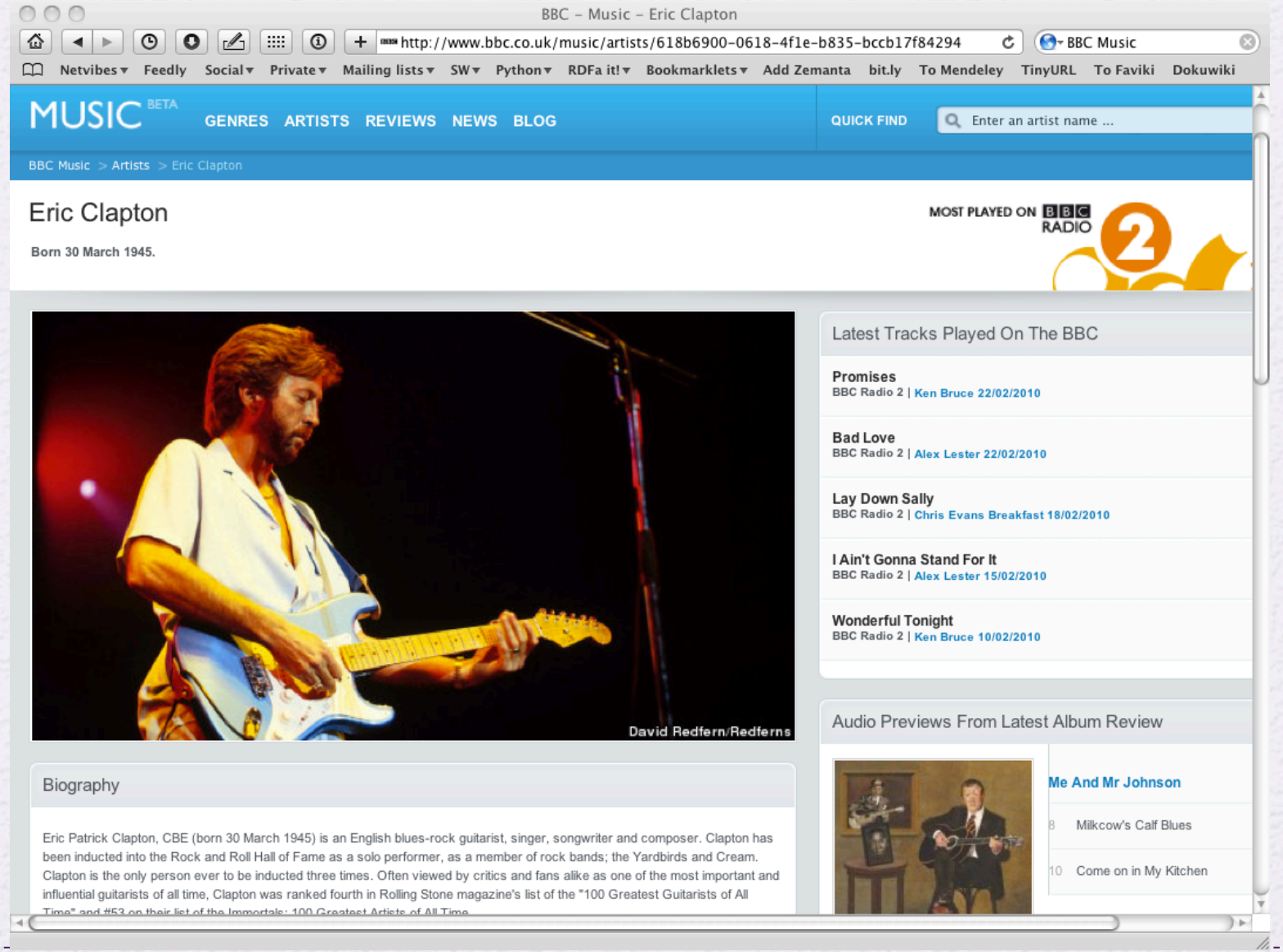
```
<http://sws.geonames.org/2759793>
  owl:sameAs <http://dbpedia.org/resource/Amsterdam>
  wgs84_pos:lat "52.3666667" ;
  wgs84_pos:long "4.8833333";
  geo:inCountry <http://www.geonames.org/countries/#NL> ;
 ...
```

Processors can switch automatically from one to the other…

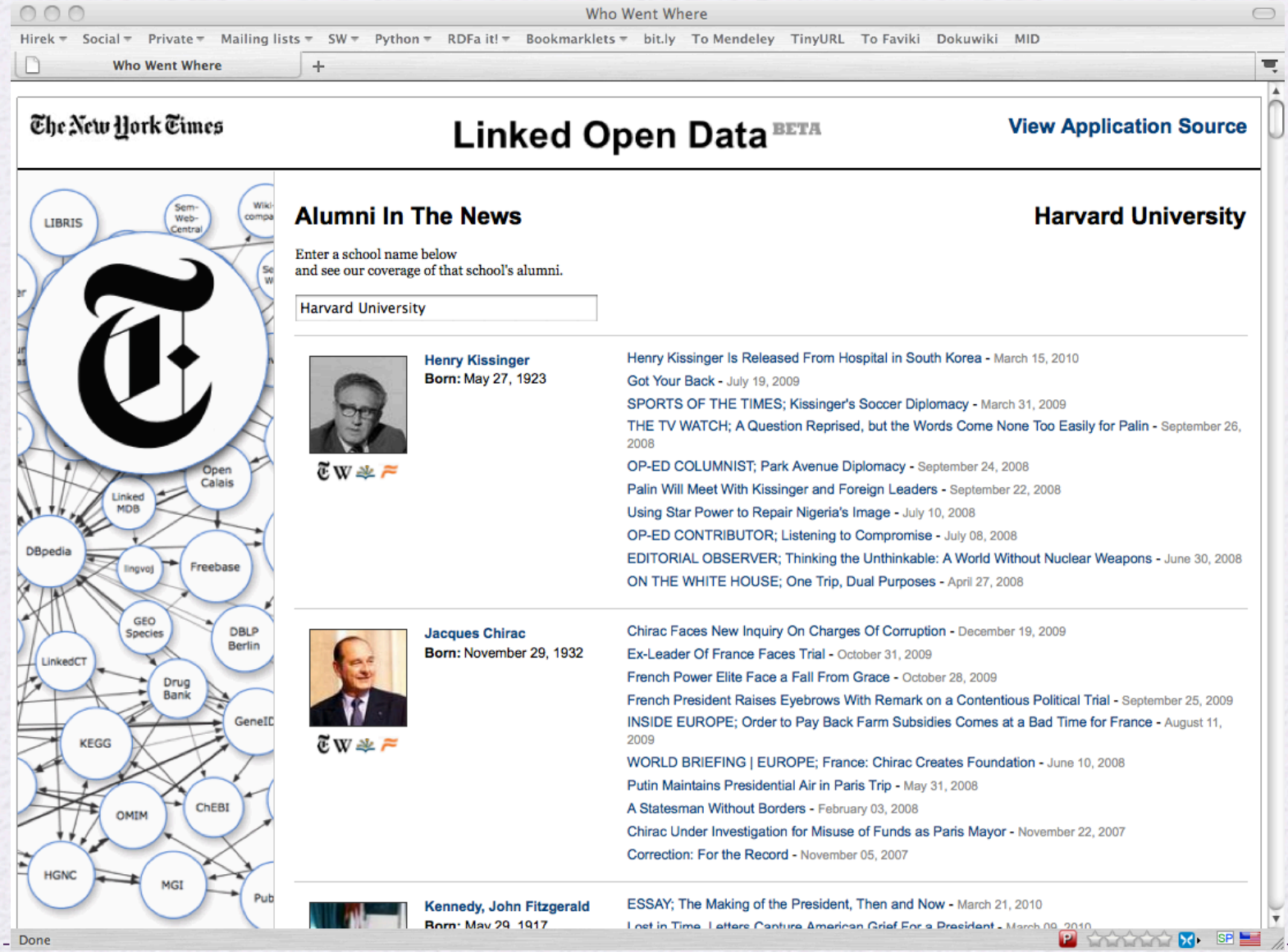
The LOD “cloud”, June 2009



Remember the BBC example?



NYT articles on university alumni



Query RDF Data
(SPARQL)

Querying RDF graphs

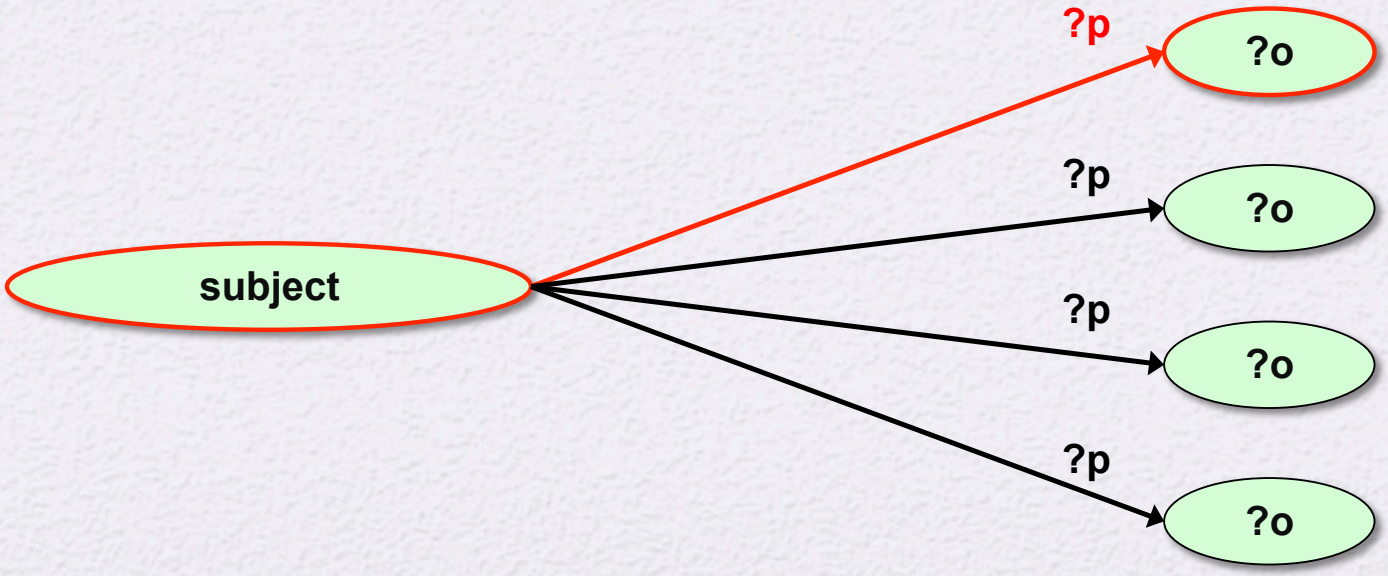
- Remember the Jena idiom:

```
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 (a,b) pairs for which there is an x such that (x parent a) and (b brother x) holds" (ie, return the uncles)
- 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);
```



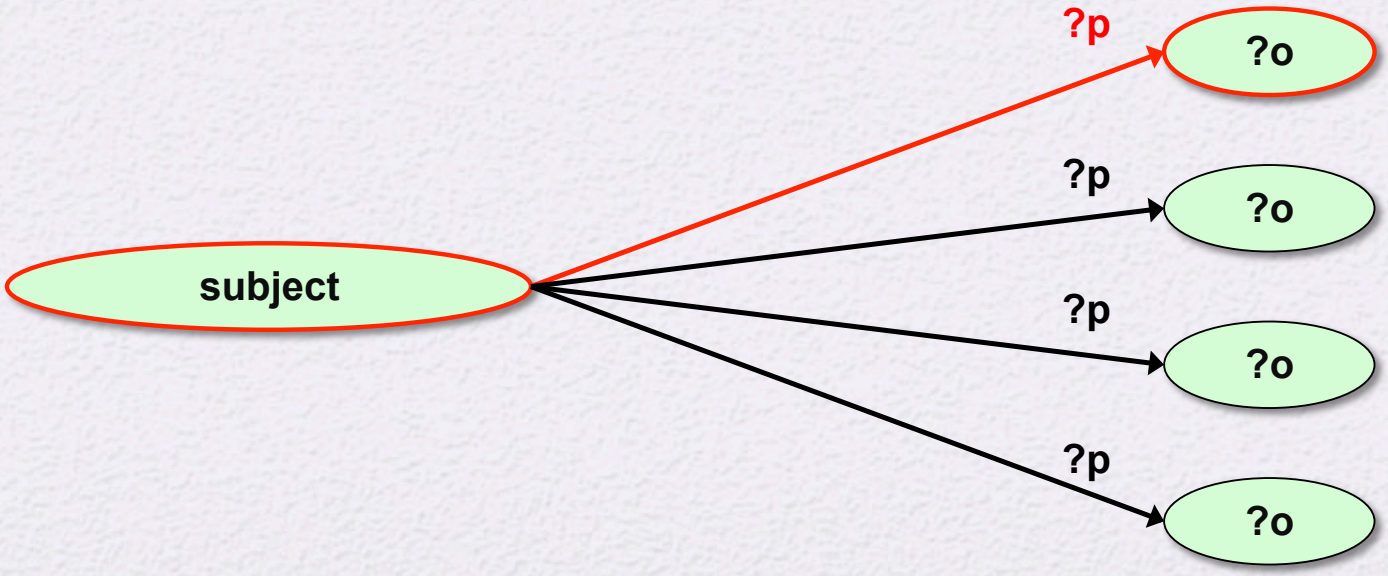
General: graph patterns

- 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

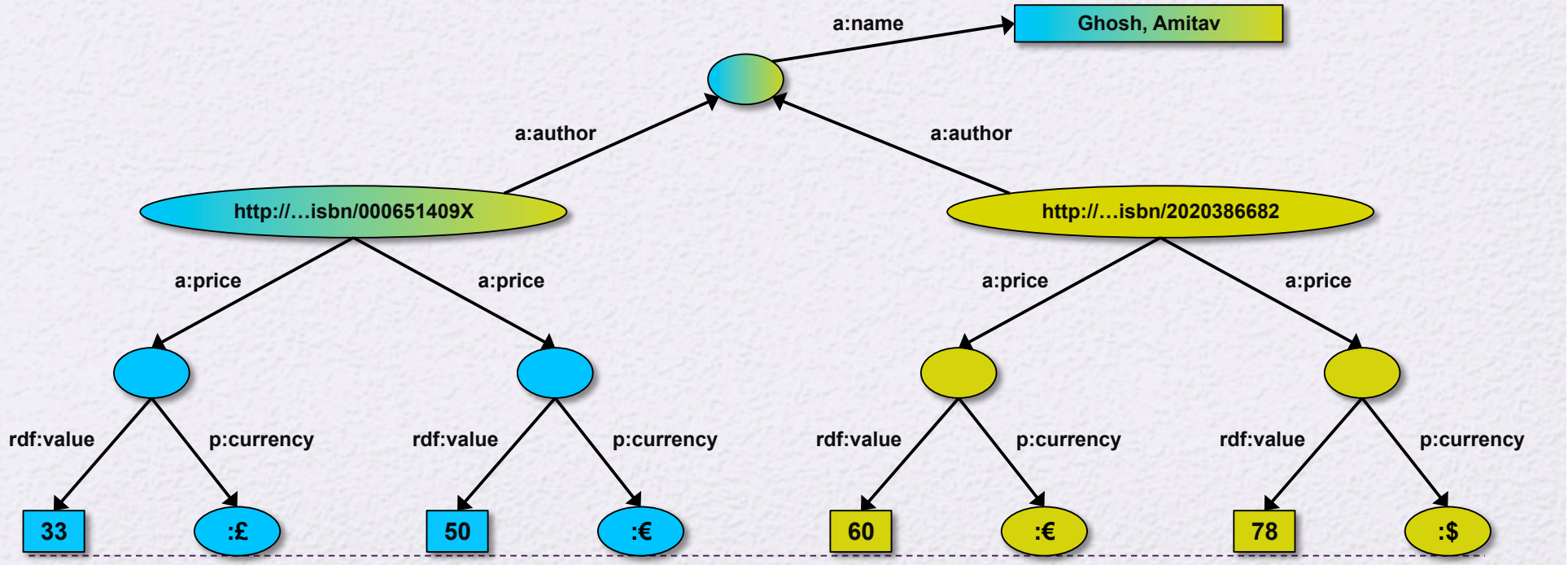
```
SELECT ?p ?o
WHERE {subject ?p ?o}
```

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



Simple SPARQL example

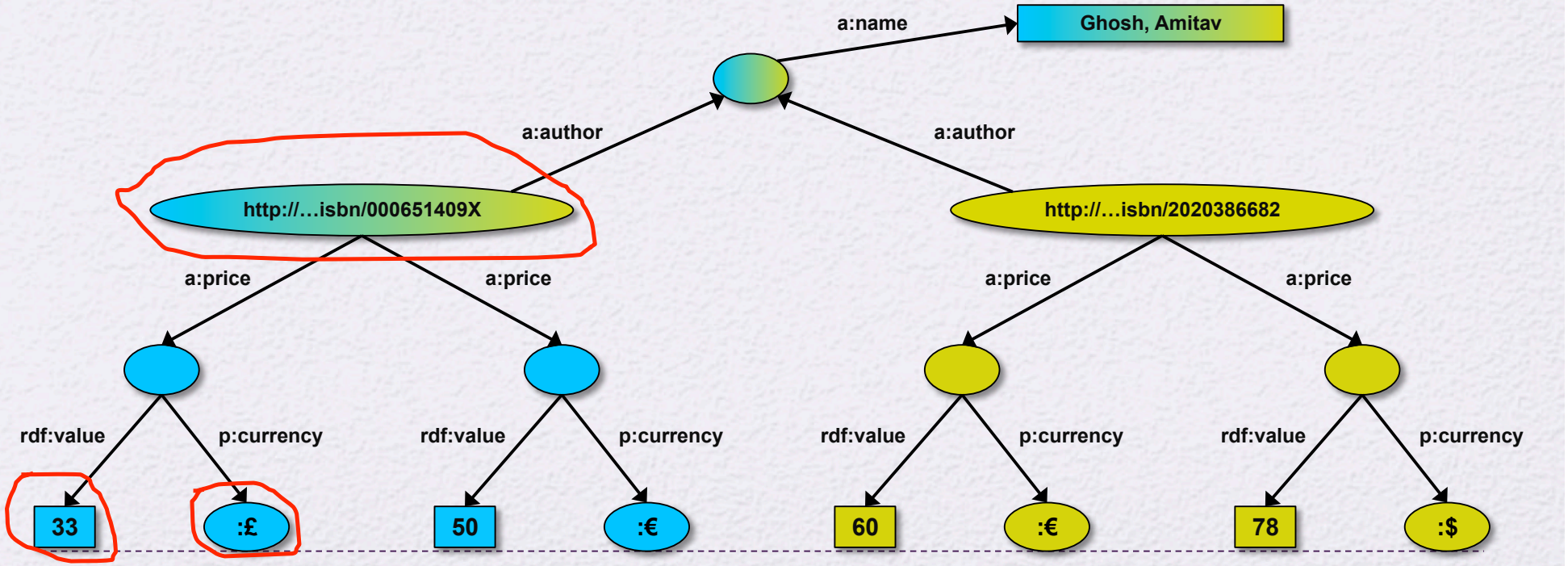
```
SELECT ?isbn ?price ?currency # note: not ?x!
WHERE {?isbn a:price ?x. ?x rdf:value ?price. ?x p:currency ?currency.}
```



Simple SPARQL example

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

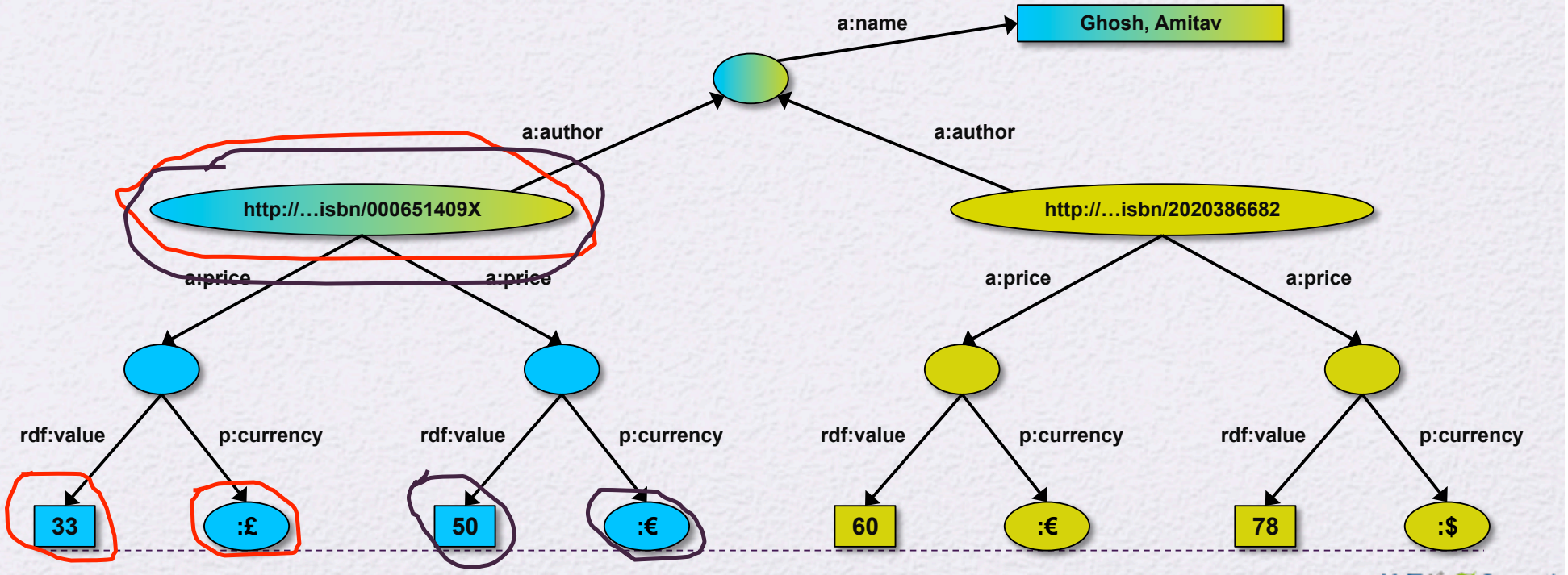
Returns: [<…409X>,33,:£]



Simple SPARQL example

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

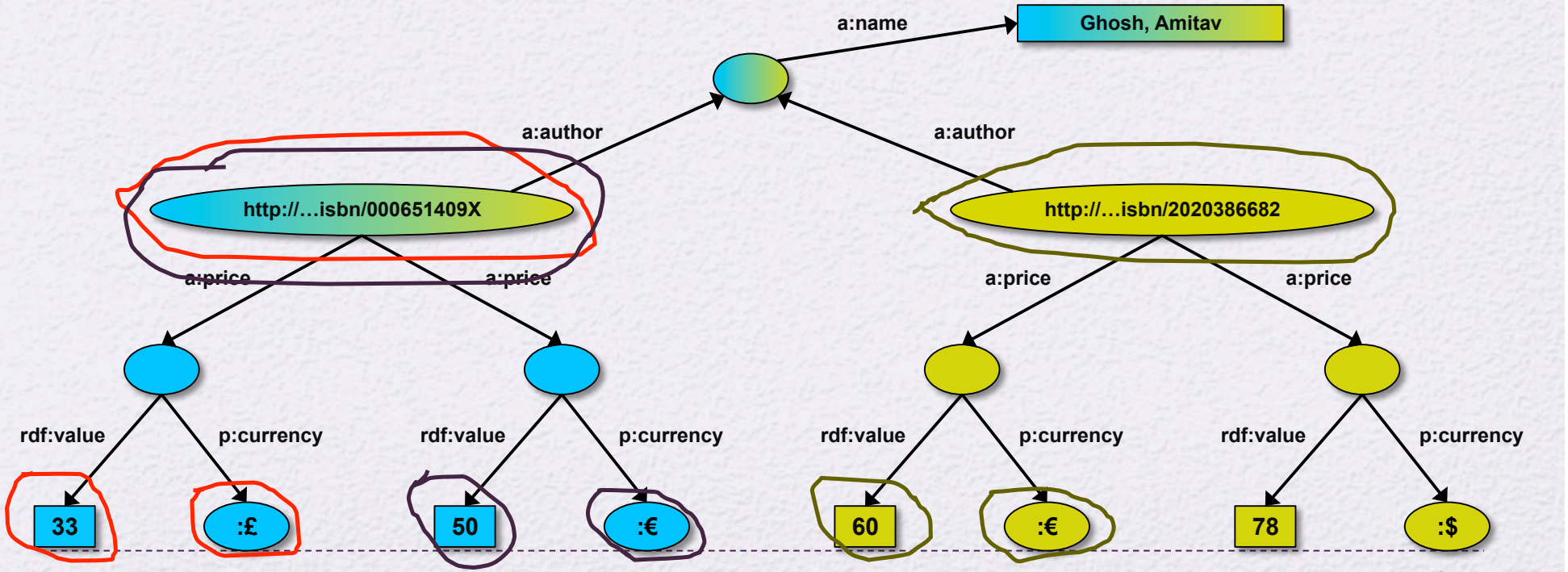
Returns: [<…409X>,33,:£], [<…409X>,50,:€]



Simple SPARQL example

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

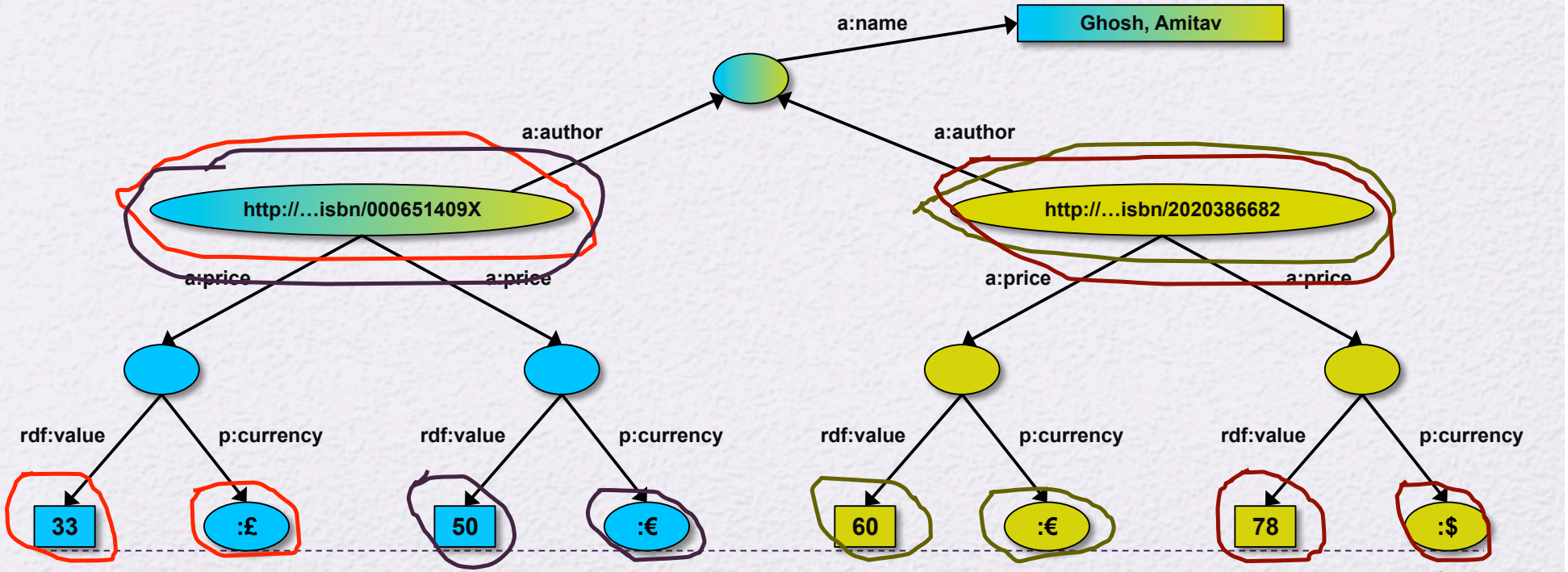
Returns: [<…409X>,33,:£], [<…409X>,50,:€],
[<…6682>,60,:€]



Simple SPARQL example

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

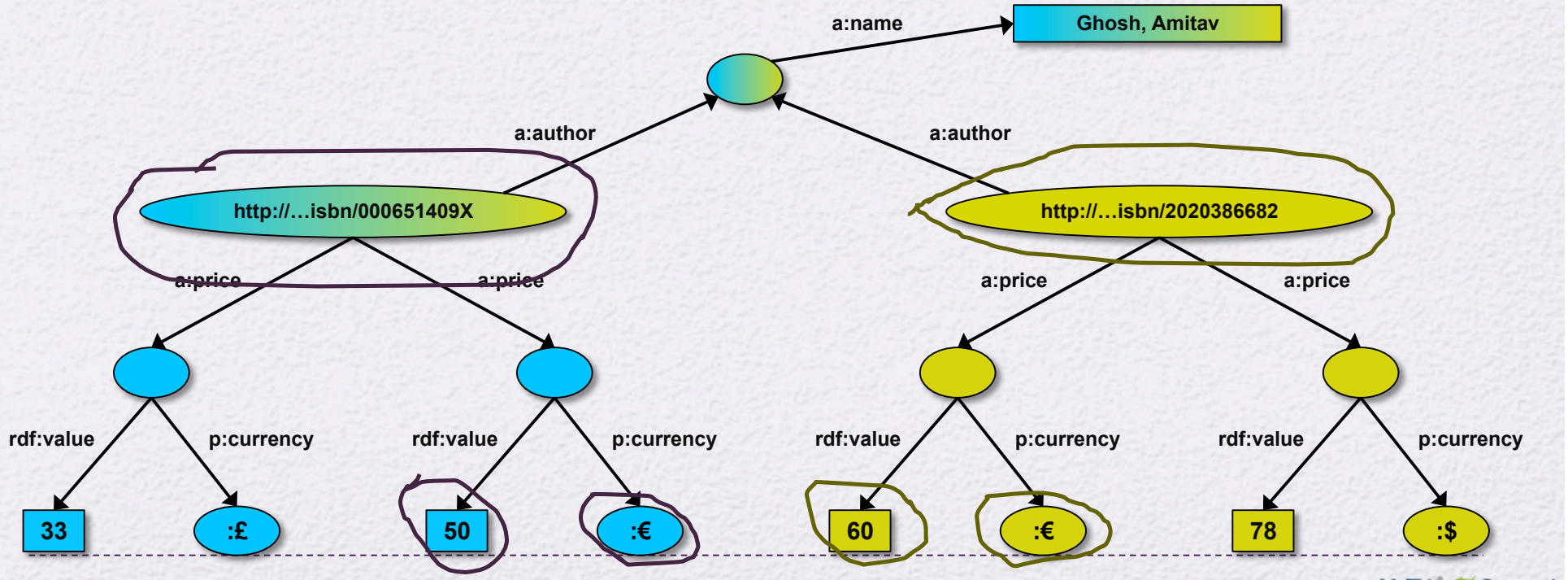
Returns: [<…409X>,33,:£], [<…409X>,50,:€],
[<…6682>,60,:€], [<…6682>,78,:$]



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,:€]



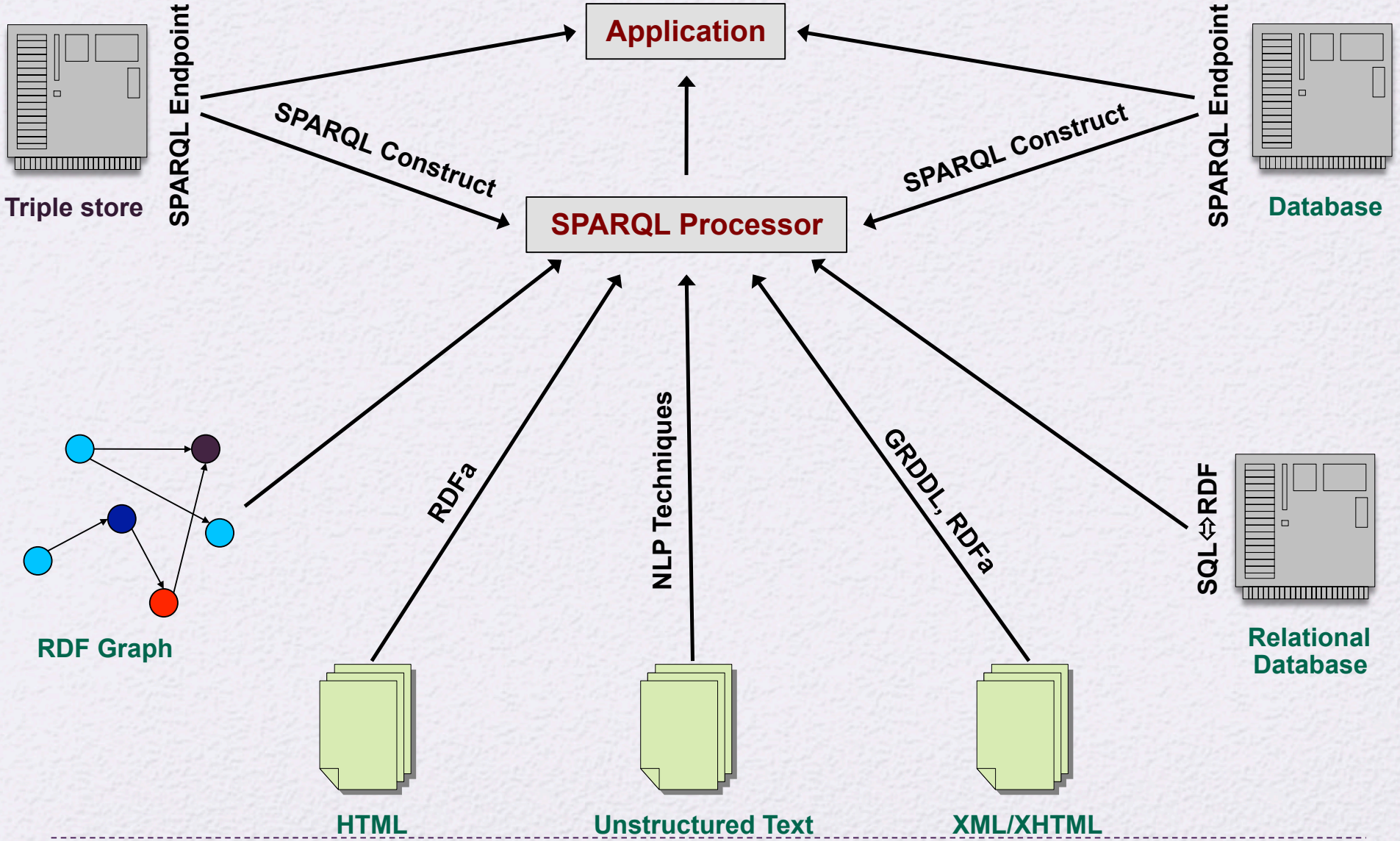
Many extra SPARQL features

- Limit the number of returned results; remove duplicates, sort them, …
- Optional branches: if some part of the pattern does not match, ignore it
- Specify several data sources (via URI-s) within the query (essentially, a merge on-the-fly!)
- Construct a graph using a separate pattern on the query results
- In SPARQL 1.1: updating data, not only query

SPARQL usage in practice

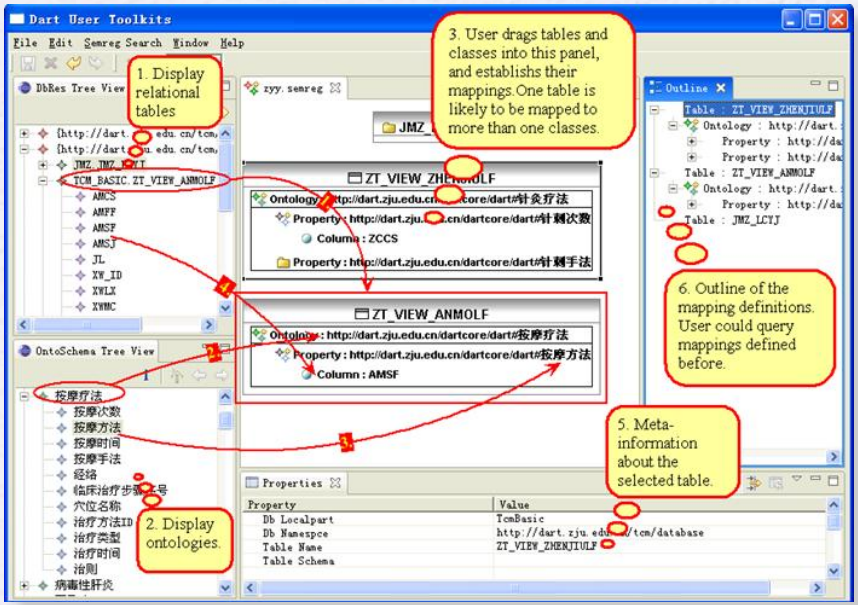
- SPARQL is usually used over the network
  - separate documents define the protocol and the result format
    - SPARQL Protocol for RDF with HTTP and SOAP bindings
    - SPARQL results in XML or JSON formats
- Big datasets often offer "SPARQL endpoints" using this protocol
  - typical example: SPARQL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

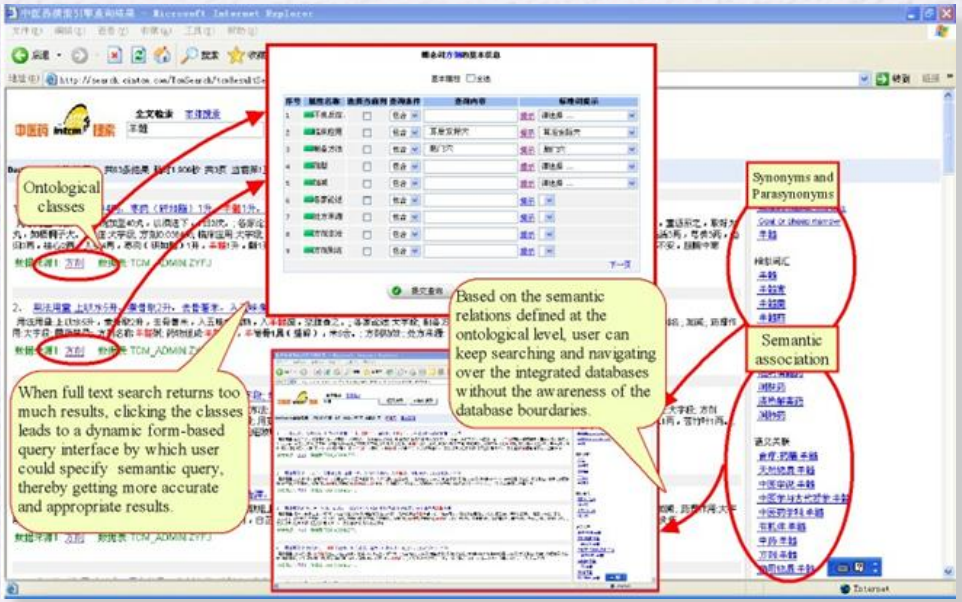
SPARQL as a unifying point



Integrate knowledge for Chinese Medicine

- Integration of a large number of TCM databases
  - around 80 databases, around 200,000 records each





Courtesy of Huajun Chen, Zhejiang University, (SWEO Case Study)

Vocabularies

Vocabularies

- Data integration needs agreements on
  - terms
    - "translator", "author"
  - categories used
    - "Person", "literature"
  - relationships among those
    - "an author is also a Person…", "historical fiction is a narrower term than fiction"
    - ie, new relationships can be deduced

Vocabularies

- There is a need for "languages" to define such vocabularies
  - to define those vocabularies
  - to assign clear "semantics" on how new relationships can be deduced

But what about RDFS?

- Indeed RDFS *is* such framework:
  - there is typing, subtyping
  - properties can be put in a hierarchy
  - datatypes can be defined
- RDFS is enough for many vocabularies
- But not for all!

Three technologies have emerged

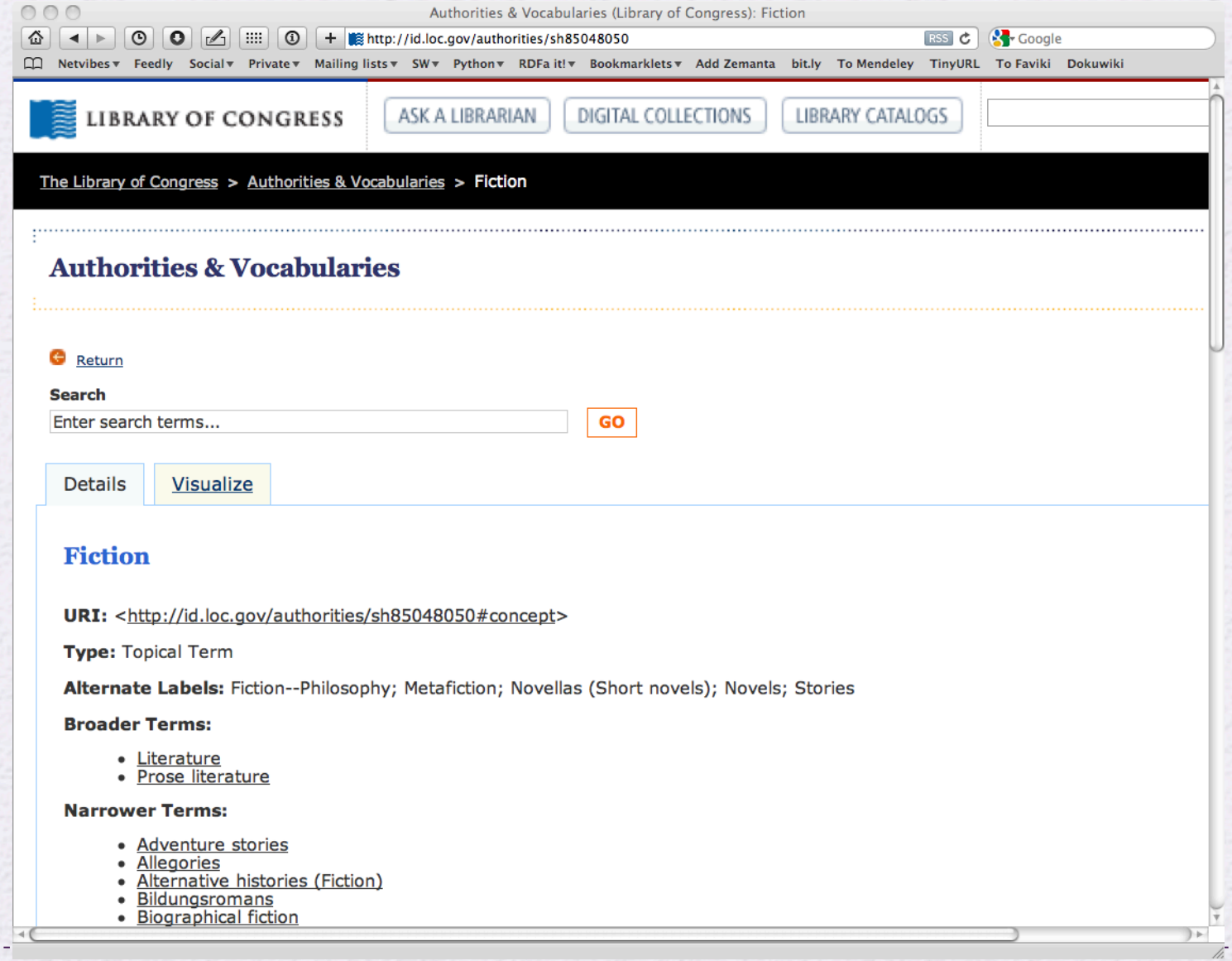
- To re-use thesauri, glossaries, etc: SKOS
- To define more complex vocabularies with a strong logical underpinning: OWL
- Generic framework to define rules on terms and data: RIF

Using thesauri, glossaries
(SKOS)

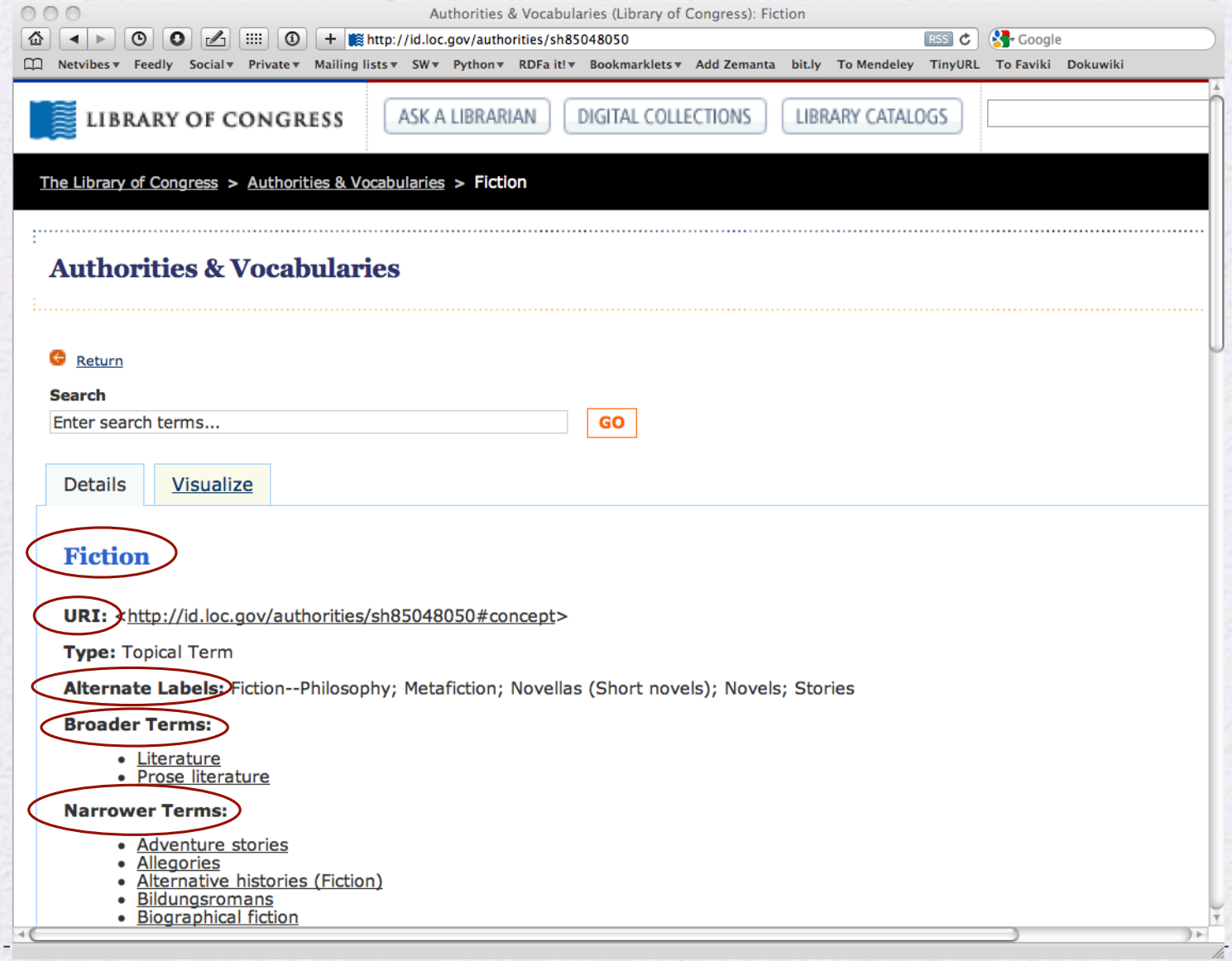
SKOS

- 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: the term “Fiction”, as defined by the Library of Congress



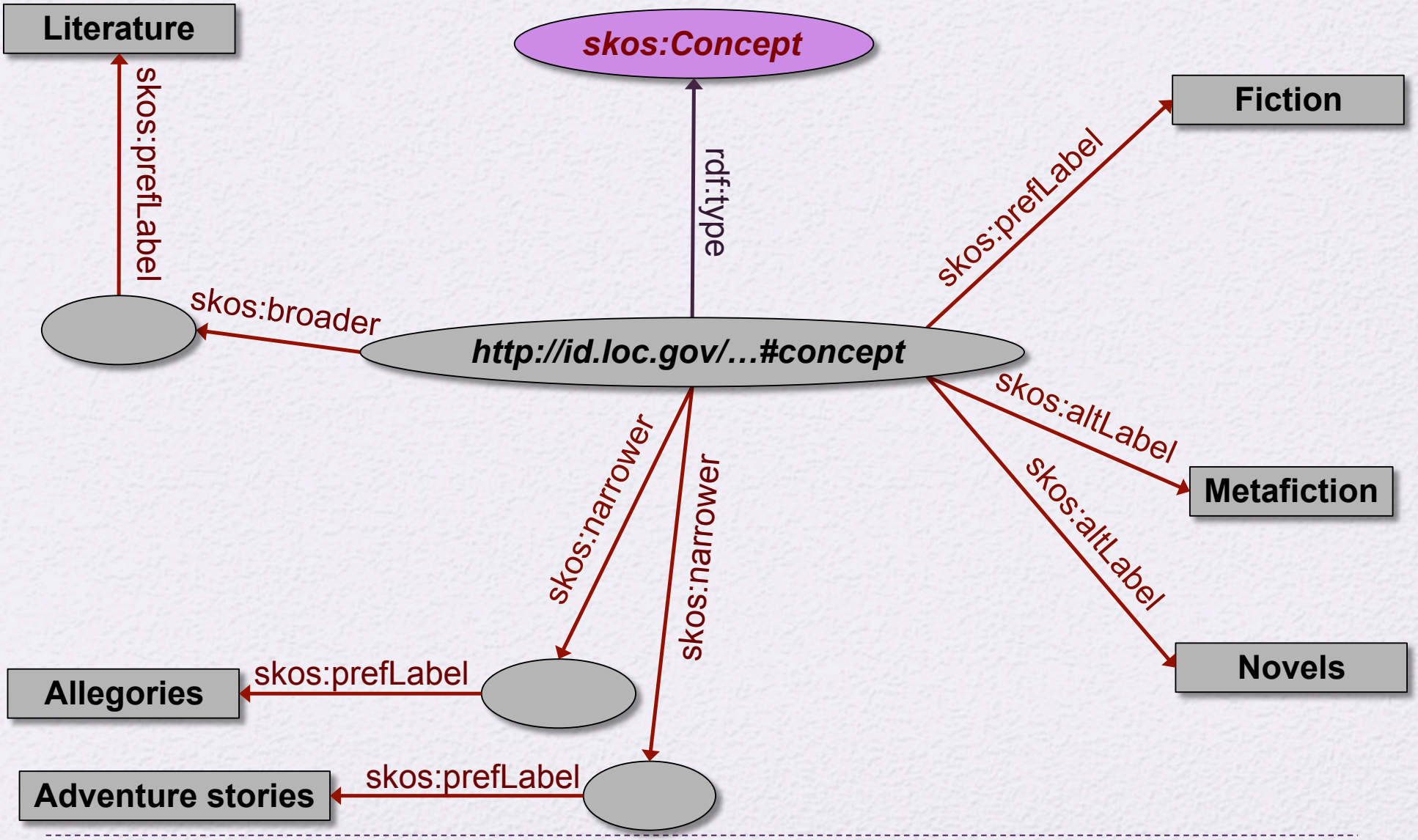
Example: the term “Fiction”, as defined by the Library of Congress



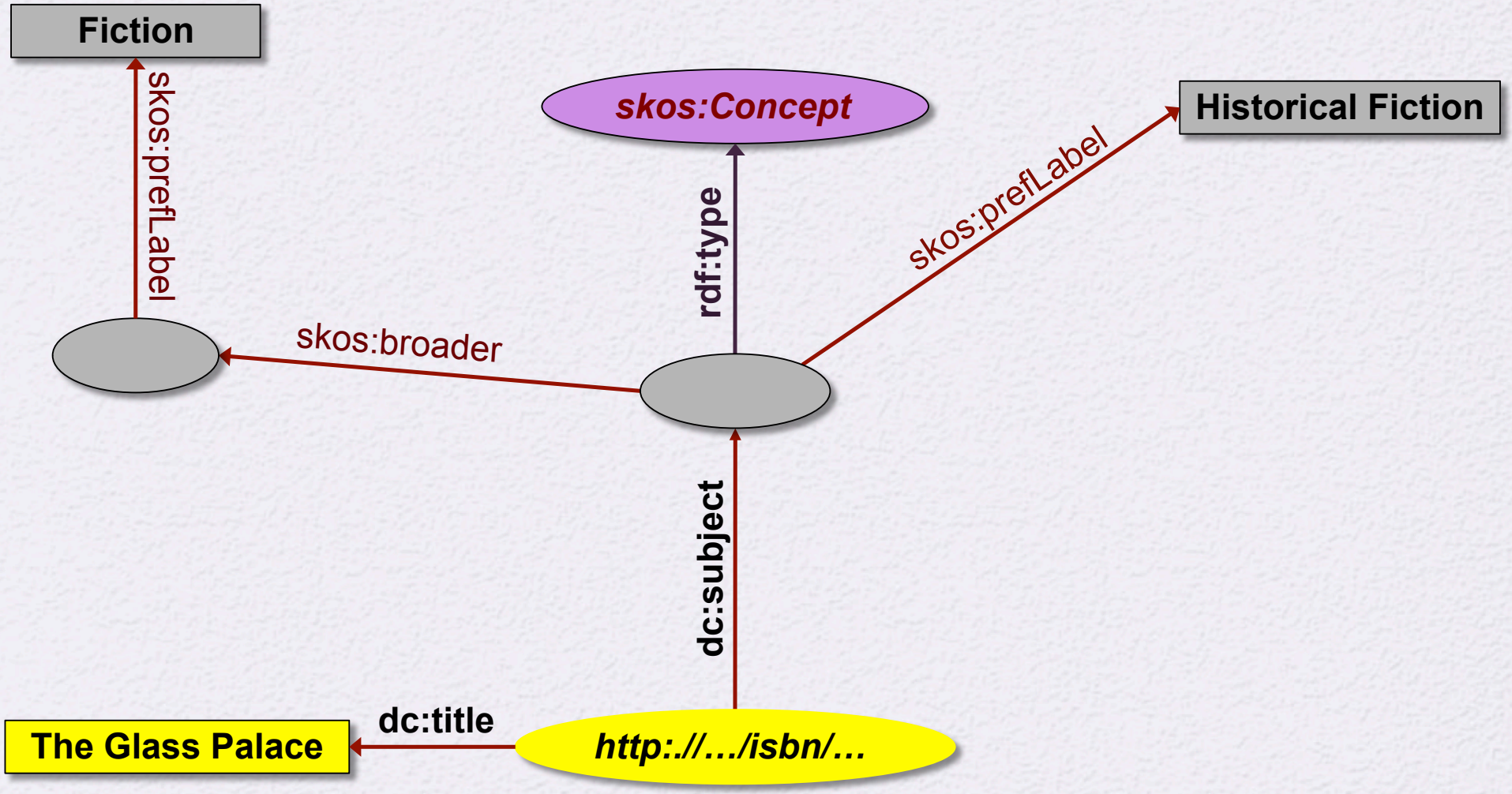
Thesauri have identical structures…

- The structure of the LOC page is fairly typical
  - label, alternate label, narrower, broader, …
  - there is even an ISO standard for such structures
- SKOS provides a basic structure to create an RDF representation of these

LOC's "Fiction" in SKOS/RDF



Usage of the LOC graph



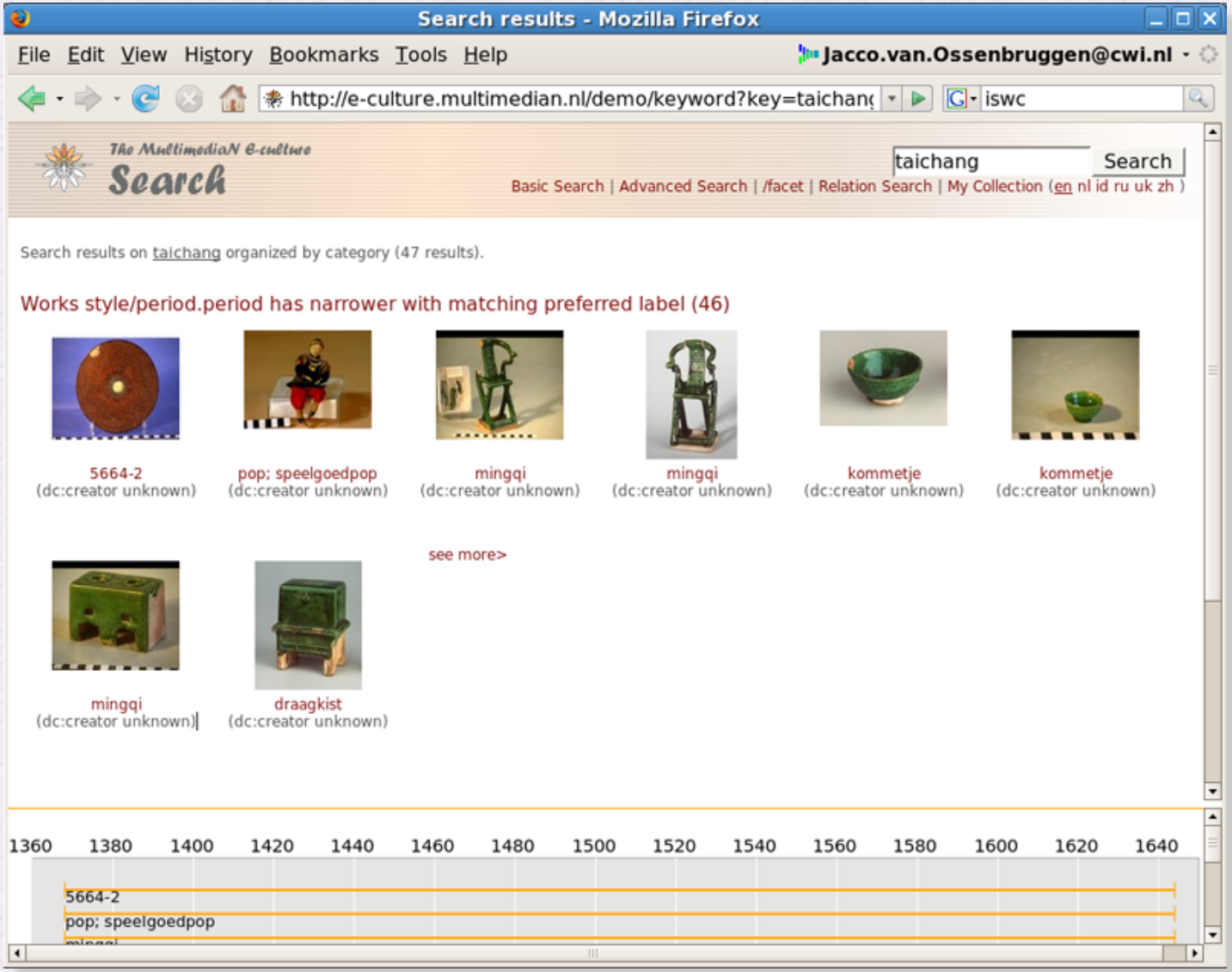
Importance of SKOS

- SKOS provides a simple bridge between the "print world" and the (Semantic) Web
- Thesauri, glossaries, etc, from the library community can be made available
  - LOC is a good example
- SKOS can also be used to organize tags, annotate other vocabularies, …

Importance of SKOS

- Anybody in the World can refer to common concepts
  - they mean the same for everybody
- Applications may exploit the relationships among concepts
  - eg, SPARQL queries may be issued on the merge of the library data and the LOC terms

Semantic portal for art collections



Courtesy of Jacco van Ossenbruggen, CWI, and Guus Schreiber, VU Amsterdam

Ontologies
(OWL)

SKOS is not enough…

- SKOS may be used to provide simple vocabularies
- But it is not a complete solution
  - it concentrates on the concepts only
  - no characterization of properties in general
  - simple from a logical perspective
    - ie, few inferences are possible

Application may want more…

- 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.

Web Ontology Language = OWL

- OWL is an extra layer, a bit like RDFS or SKOS
  - 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") published in 2009

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

Term equivalences

- 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 Linked Open Data project

Property characterization

- In OWL, one can characterize the behavior of properties (symmetric, transitive, functional, reflexive, inverse functional…)
- One property can be defined as the "inverse" of another

What this means is…

- If the following holds in our triples:

```
:email rdf:type owl:InverseFunctionalProperty.
```

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".
```

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>.
```

Keys

- 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

"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

```
: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, 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

Enumerate class content

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

- I.e., the class consists of *exactly* of those individuals and nothing else

Union of classes

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

- Other possibilities: owl:complementOf, owl:intersectionOf, …

For example…

If:

```
:Novel            rdf:type owl:Class.
:Short_Story      rdf:type owl:Class.
:Poetry           rdf:type owl:Class.
:Literature rdf:type owl:Class;
   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
  - some major issues
    - the way classes (i.e., "concepts") are defined
    - handling of datatypes like intervals
- OWL includes those extra features but… the inference engines become (much) more complex

Example: property value restrictions

- New classes are created by *restricting* the property values on a class
- For example: how would I characterize a "listed price"?
  - it is a price that is given in one of the "allowed" currencies (€, £, or $)
  - this defines a new class

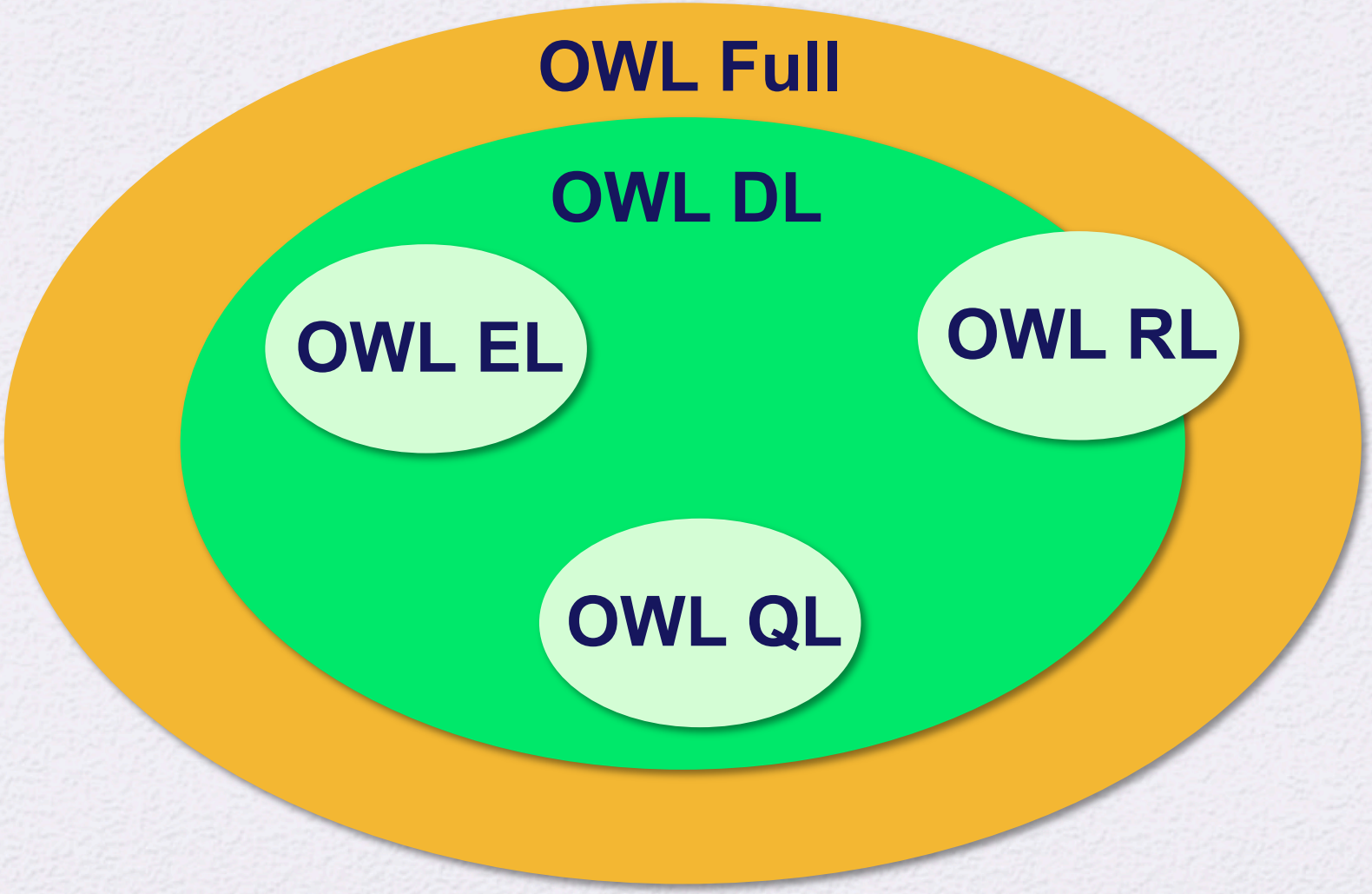
But: OWL is hard!

- 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" or profiles

- 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: expressiveness vs. implementability

OWL Species



OWL RL

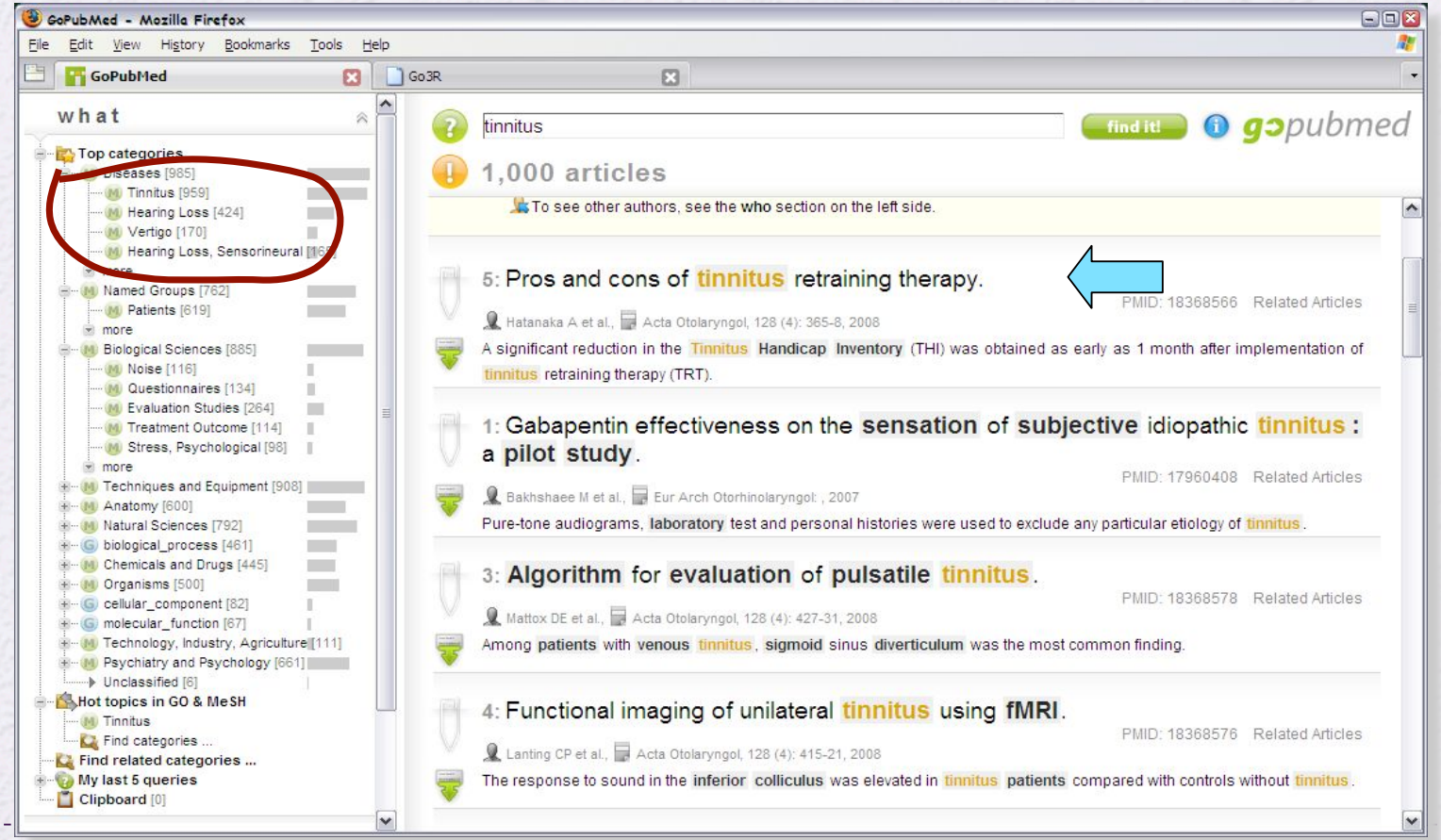
- Goal: to be implementable with rule engines
- Usage follows a similar approach to RDFS:
  - merge the ontology and the instance data into an RDF graph
  - use the rule engine to add new triples (as long as it is possible)

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 (but with some restrictions)
  - property characterizations (functional, symmetric, etc)
  - property chains
  - keys
  - *some* property restrictions
- All examples so far could be inferred with OWL RL!

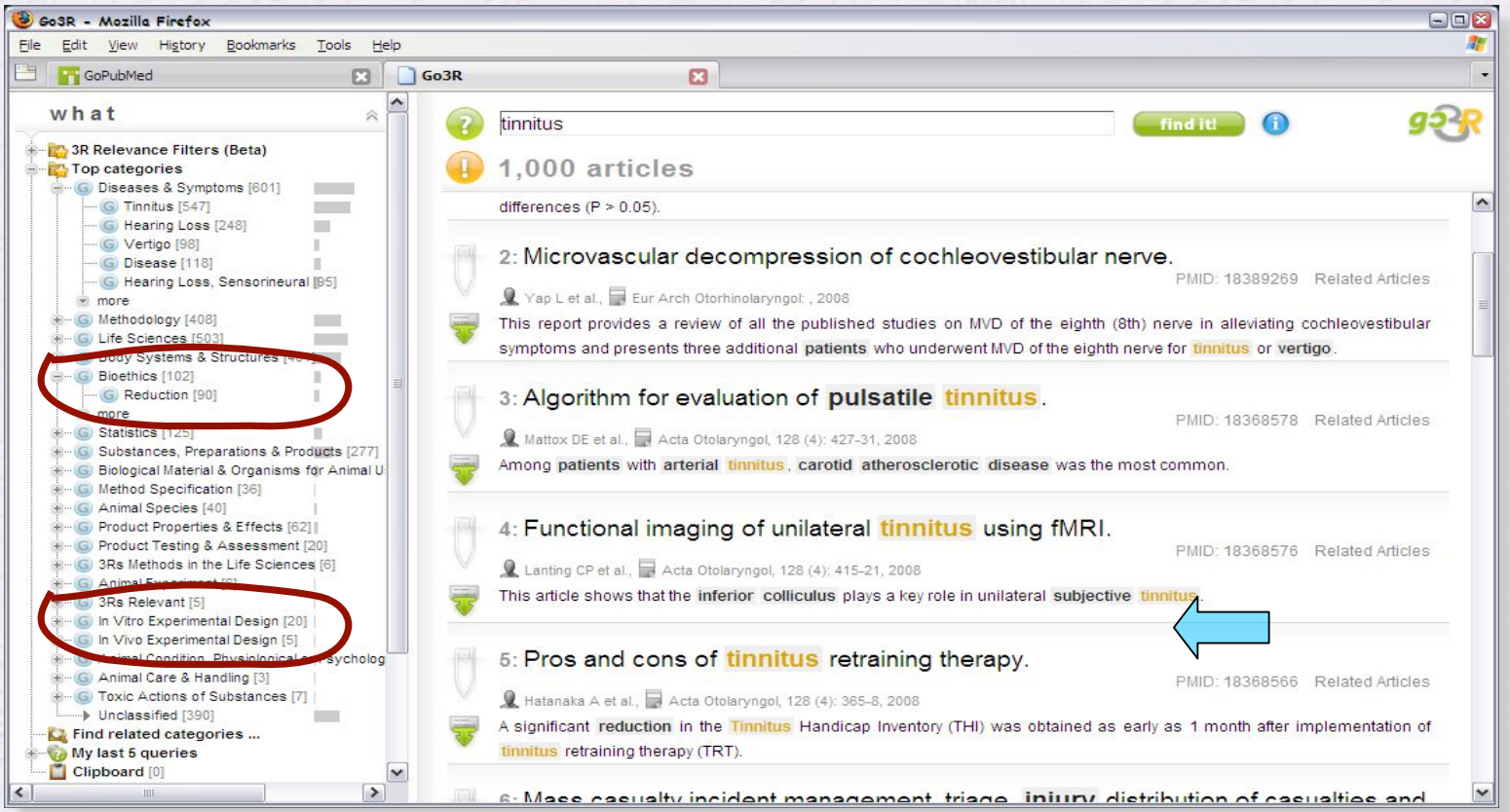
Improved Search via Ontology (GoPubMed)

- Search results are re-ranked using ontologies
  - related terms are highlighted



Improved Search via Ontology (Go3R)

- Same dataset, different ontology
  - (ontology is on non-animal experimentation)



Rules
(RIF)

Why rules on the Semantic Web?

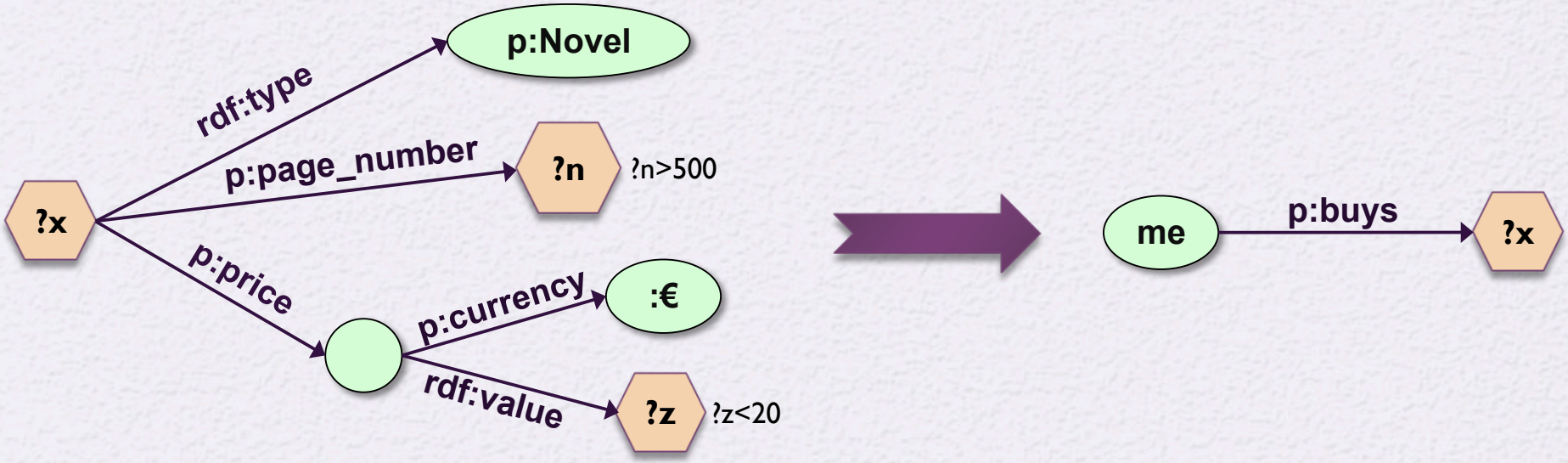
- Some conditions may be complicated in ontologies (ie, OWL)
  - eg, Horn rules: (P1 & P2 & …) → C
- In many cases applications just want 2-3 rules to complete integration
- Ie, rules may be an alternative to (OWL based) ontologies

Things you may want to express

- An example from our bookshop integration:
  - "I buy a novel with over 500 pages if it costs less than €20"
  - something like (in an ad-hoc syntax):

```
{
  ?x rdf:type p:Novel;
     p:page_number ?n;
     p:price [
         p:currency :€;
         rdf:value  ?z
     ].
  ?n > "500"^^xsd:integer.
  ?z < "20.0"^^xsd:double.
}
=>
{ <me> p:buys ?x }
```

Things you may want to express



RIF (Rule Interchange Format)

- The goals of the RIF work:
  - define simple rule language(s) for the (Semantic) Web
  - define interchange formats for rule based systems
- RIF defines several "dialects" of languages
- RIF is not bound to RDF only
  - eg, relationships may involve more than 2 entities
  - there are dialects for production rule systems

RIF Core

- The simplest RIF "dialect"
- A Core document is
  - directives like import, prefix settings for URI-s, etc
  - a sequence of logical implications

RIF Core example

```
Document(
  Prefix(cpt http://example.com/concepts#)
  Prefix(person http://example.com/people#)
  Prefix(isbn http://…/isbn/)

  Group
  (
    Forall ?Buyer ?Book ?Seller (
        cpt:buy(?Buyer ?Book ?Seller):- cpt:sell(?Seller ?Book ?Buyer)
    )
    cpt:sell(person:John isbn:000651409X person:Mary)
  )
)
```

This infers the following relationship:

```
cpt:buy(person:Mary isbn:000651409X person:John)
```

Expressivity of RIF Core

- Formally: definite Horn without function symbols, a.k.a. "Datalog"
  - eg, p(a,b,c) is fine, but p(f(a),b,c) is not
- Includes some extra features
  - built-in datatypes and predicates
  - "local" symbols, a bit like blank nodes

Expressivity of RIF Core

- There are also "safeness measures"
  - eg, variable in a consequent should be in the antecedent
  - this secures a straightforward implementation strategy ("forward chaining")

RIF Syntaxes

- RIF defines
  - a "presentation syntax"
  - a standard XML syntax to encode and exchange the rules
  - there is a draft for expressing Core in RDF
    - just like OWL is represented in RDF

What about RDF and RIF?

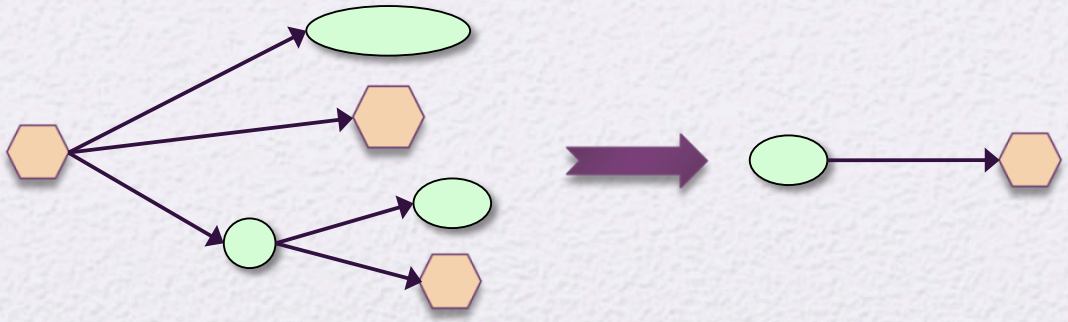
- Typical scenario:
  - the "data" of the application is available in RDF
  - rules on that data is described using RIF
  - the two sets are "bound" (eg, RIF "imports" the data)
  - a RIF processor produces new relationships

To make RIF/RDF work

- Some technical issues should be settled:
  - RDF triples have to be representable in RIF
  - various constructions (typing, datatypes, lists) should be aligned
  - the semantics of the two worlds should be compatible
- There is a separate document that brings these together

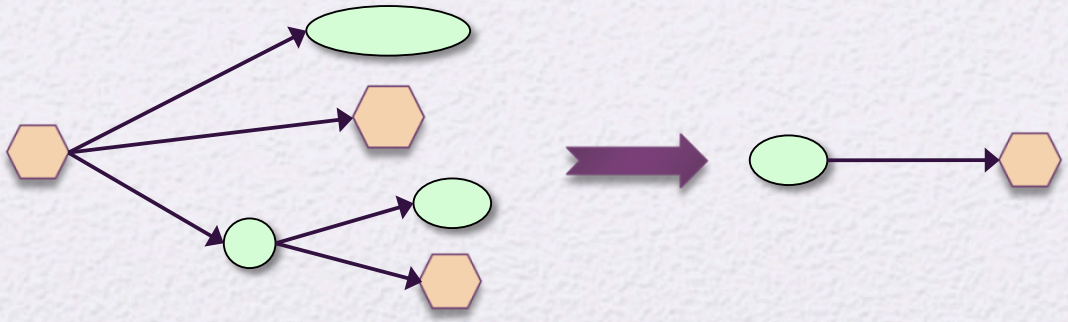
Remember the what we wanted from Rules?

```
{
  ?x rdf:type p:Novel;
     p:page_number ?n;
     p:price [
         p:currency :€;
         rdf:value  ?z
     ].
  ?n > "500"^^xsd:integer.
  ?z < "20.0"^^xsd:double.
}
=>
{ <me> p:buys ?x }
```



The same with RIF Presentation syntax

```
Document (
  Prefix …
  Group (
    Forall ?x ?n ?z (
      <me>[p:buys->?x] :-
        And(
          ?x rdf:type p:Novel
          ?x[p:page_number->?n p:price->_abc]
          _abc[p:currency->:€ rdf:value->?z]
          External( pred:numeric-greater-than(?n "500"^^xsd:integer) )
          External( pred:numeric-less-than(?z "20.0"^^xsd:double) )
        )
    )
  )
)
```



Discovering new relationships...

```
Forall ?x ?n ?z (
  <me>[p:buys->?x] :-
    And(
      ?x # p:Novel
      ?x[p:page_number->?n p:price->_abc]
      _abc[p:currency->:€ rdf:value->?z]
      External( pred:numeric-greater-than(?n "500"^^xsd:integer) )
      External( pred:numeric-less-than(?z "20.0"^^xsd:double) )
    )
)
```

Discovering new relationships…

```
Forall ?x ?n ?z (
  <me>[p:buys->?x] :-
    And(
      ?x # p:Novel
      ?x[p:page_number->?n p:price->_abc]
      _abc[p:currency->:€ rdf:value->?z]
      External( pred:numeric-greater-than(?n "500"^^xsd:integer) )
      External( pred:numeric-less-than(?z "20.0"^^xsd:double) )
    )
)
```

combined with:

```
<http://…/isbn/…> a p:Novel;
    p:page_number "600"^^xsd:integer ;
    p:price [ rdf:value "15.0"^^xsd:double ; p:currency :€ ] .
```

Discovering new relationships…

```
Forall ?x ?n ?z (
  <me>[p:buys->?x] :-
    And(
      ?x # p:Novel
      ?x[p:page_number->?n p:price->_abc]
      _abc[p:currency->:€ rdf:value->?z]
      External( pred:numeric-greater-than(?n "500"^^xsd:integer) )
      External( pred:numeric-less-than(?z "20.0"^^xsd:double) )
    )
)
```

combined with:

```
<http://…/isbn/…> a p:Novel;
    p:page_number "600"^^xsd:integer ;
    p:price [ rdf:value "15.0"^^xsd:double ; p:currency :€ ] .
```

yields:

```
<me> p:buys <http://…/isbn/…> .
```

RIF vs. OWL?

- The expressivity of the two is fairly identical
  - the emphasis are a bit different
- Using rules vs. ontologies may largely depend on
  - available tools
  - personal technical experience and expertise
  - taste…

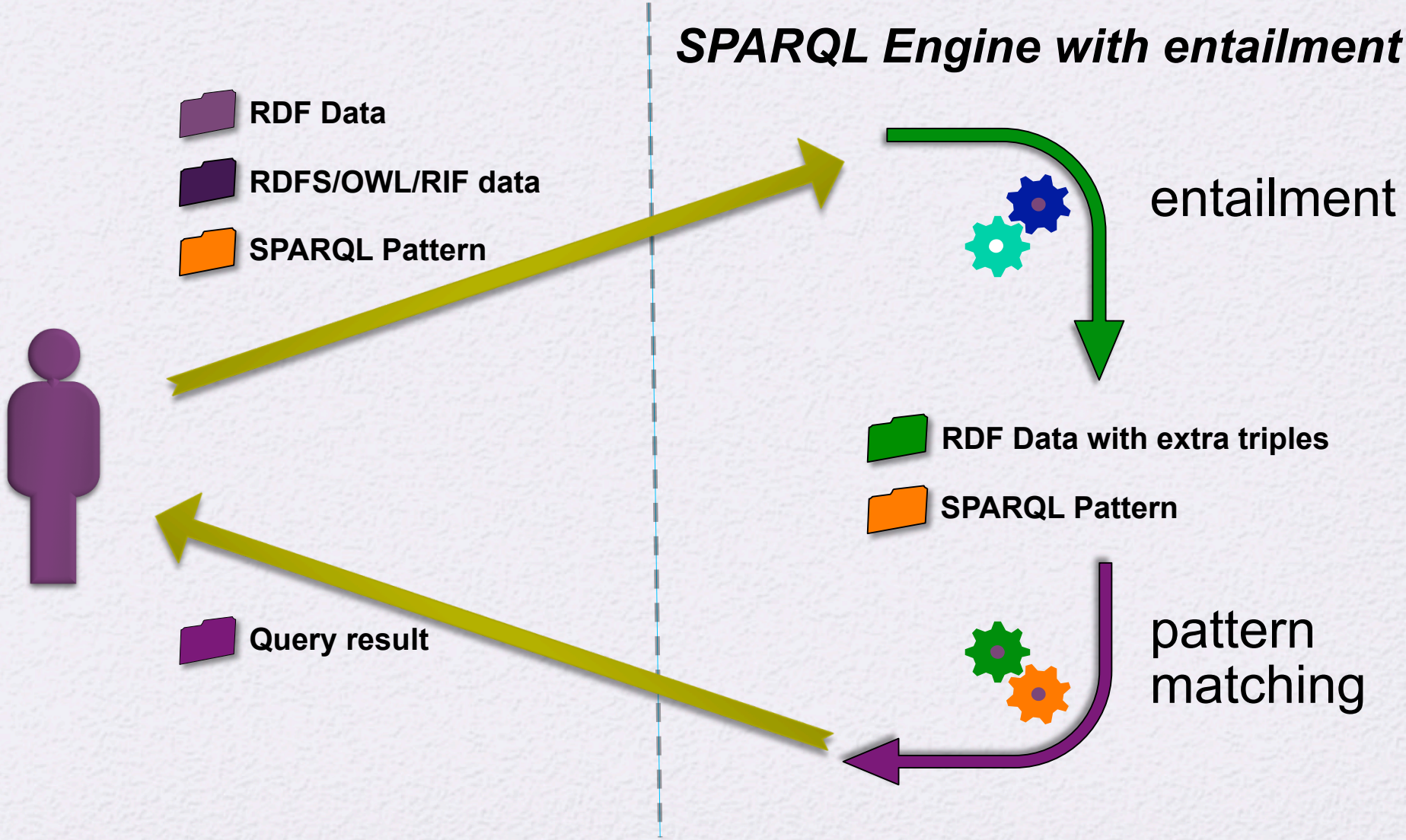
What about OWL RL?

- OWL RL stands for "Rule Language"…
- OWL RL is in the intersection of RIF Core and OWL
  - inferences in OWL RL can be expressed with RIF rules
  - RIF Core engines can act as OWL RL engines

Inferencing and SPARQL

- Question: how do SPARQL queries and inferences work together?
  - RDFS, OWL, and RIF produce new relationships
  - on what data do we query?
- Answer: in current SPARQL, that is not defined
- But, in SPARQL 1.1 it is…

SPARQL 1.1 and RDFS/OWL/RIF



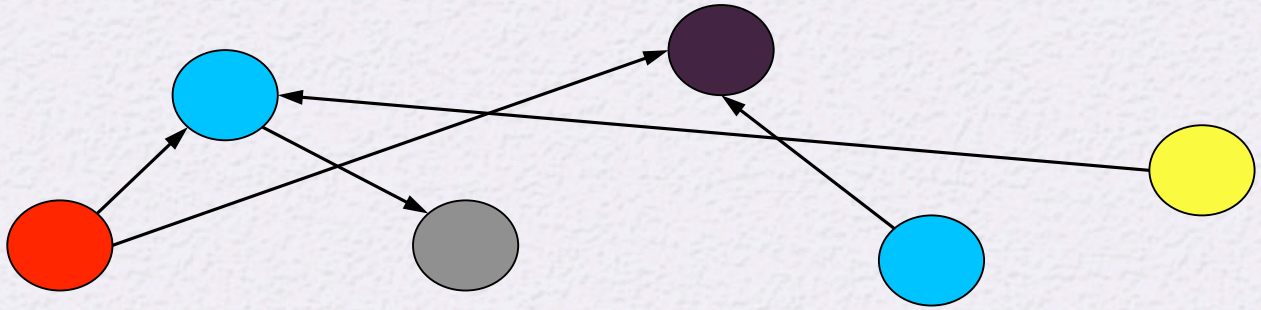
What have we achieved?

(putting all this together)

Remember the integration example?

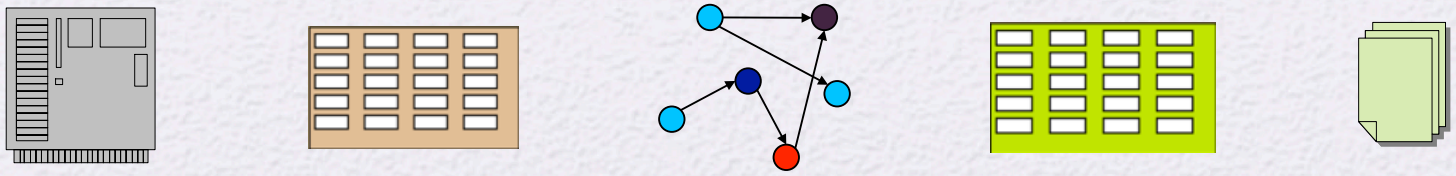
Applications

Manipulate
Query
…



Data represented in abstract format

Map,
Expose,
…

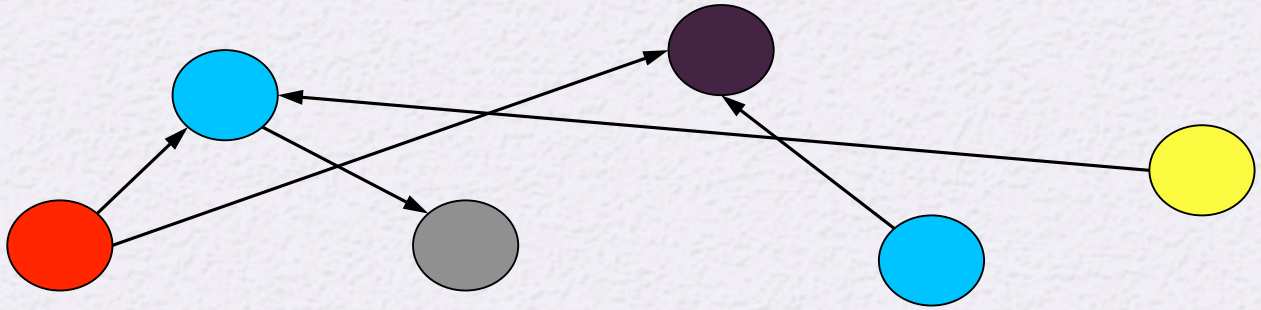


Data in various formats

Same with what we learned

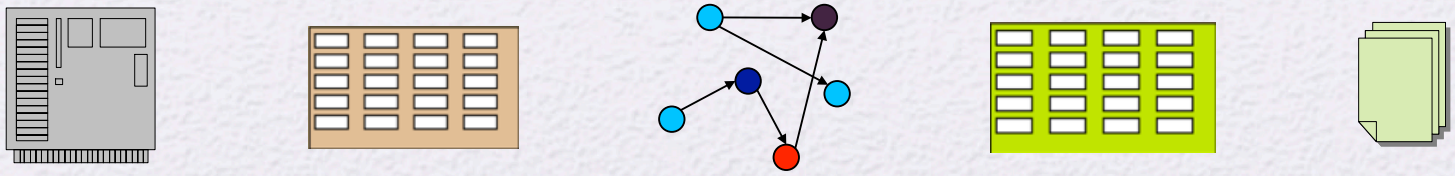
Applications

SPARQL, Inferences …



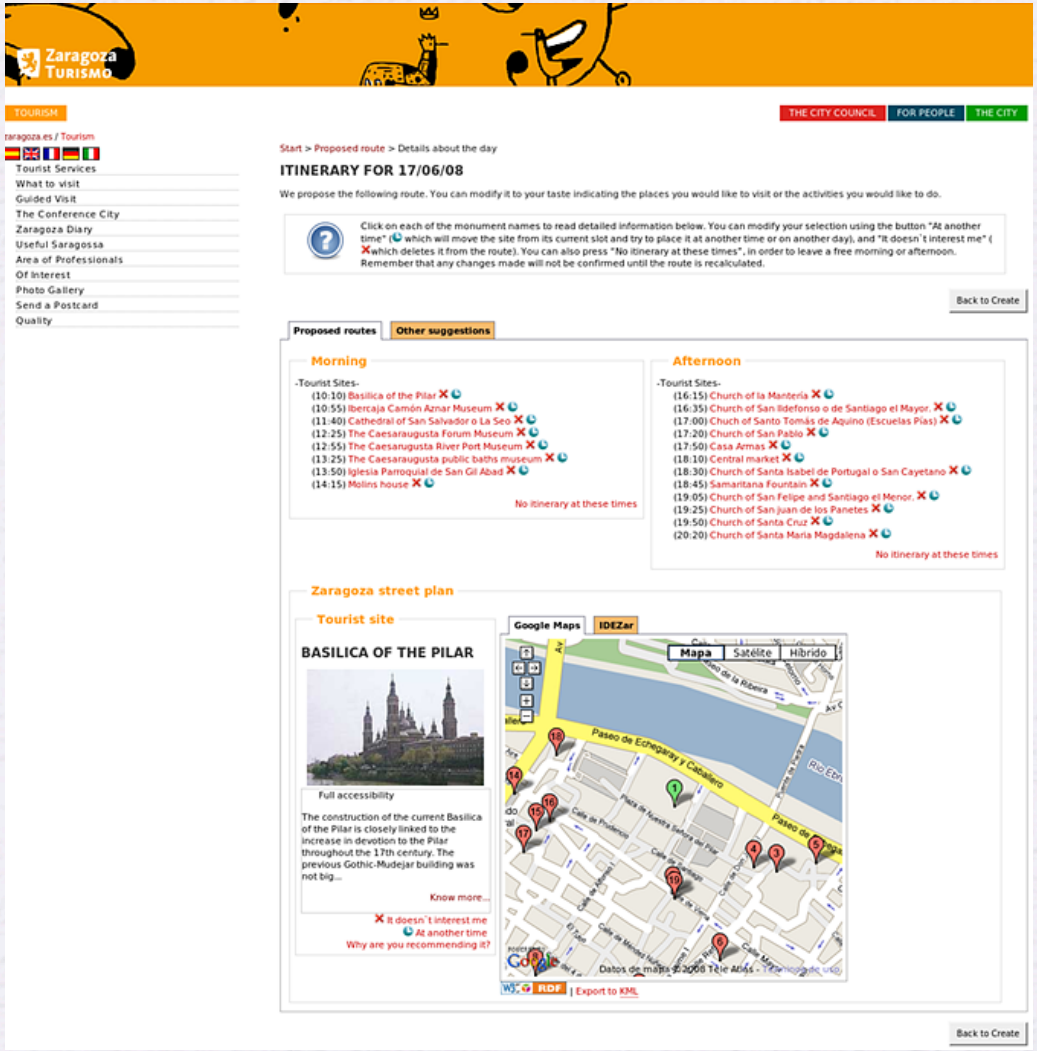
Data represented in RDF with extra knowledge (RDFS, SKOS, RIF, OWL,…)

RDB ⇔ RDF, GRDL, RDFa, …



Data in various formats

eTourism: provide personalized itinerary



- Integration of relevant data in Zaragoza (using RDF and ontologies)
- Use rules on the RDF data to provide a proper itinerary

Courtesy of Jesús Fernández, Mun. of Zaragoza, and Antonio Campos, CTIC (SWEO Use Case)

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 Wiki has links to all the specifications

“Core” vocabularies

- There are also a number “core vocabularies”
  - 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

- J. Pollock: Semantic Web for Dummies, 2009
- 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
- P. Hitzler, R. Sebastian, M. Krötzsch: Foundation of Semantic Web Technologies, 2009
- …

See the separate Wiki page collecting book references

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, Talis Platform, …
  - RDF Gateway, RDFLib, Open Anzo, DartGrid, Zitgist, Ontotext, Protégé, …
  - Thetus publisher, SemanticWorks, SWI-Prolog, RDFStore…
  - …

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/

Thank you for your attention!

These slides are also available on the Web:

http://www.w3.org/2010/Talks/0622-SemTech-IH/