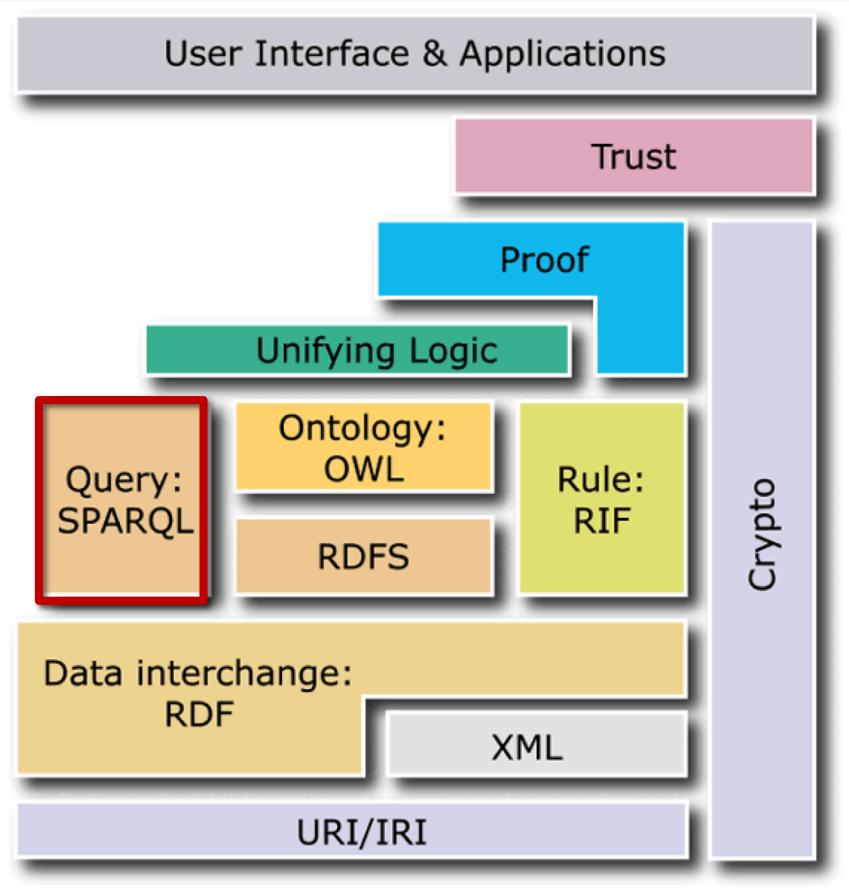
SPARQL QUERY LANGUAGE

Hala Skaf-Molli
Associate Professor
Nantes University
Hala.Skaf@univ-nantes.fr
http://pagesperso.ls2n.fr/~skaf-h

SPARQL

- RDF **query language + access protocol**
- SPARQL Protocol for RDF
  - Transmission of SPARQL queries and receiving the results
  - **SPARQL endpoint** : implements SPARQL protocol



SPARQL Example

Example at http://dbpedia.org/sparql:

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX : <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/property/>
SELECT ?name ?birth ?death ?person
WHERE {  ?person dbo:birthPlace :Nantes .
          ?person dbo:birthDate ?birth .
          ?person foaf:name ?name .
         ?person dbo:deathDate ?death .
         FILTER (?birth < "1900-01-01"^^xsd:date) . }
ORDER BY ?name
```

SPARQL results:

name	birth	death	person
"50pxAlfred Marie-Joseph Heurtaux"@en	"1893-05-19+02:00"^^xsd:date	"1985-12-29+02:00"^^xsd:date	:Alfred_Heurtaux
"Adelaide Dufrenoy"@en	"1765-12-02+02:00"^^xsd:date	"1825-03-07+02:00"^^xsd:date	:Ad%C3%A9la%C3%AFde_Dufr%C3%A9noy
"Adélaïde-Gillette Dufrénoy"@en	"1765-12-02+02:00"^^xsd:date	"1825-03-07+02:00"^^xsd:date	:Ad%C3%A9la%C3%AFde_Dufr%C3%A9noy
"Alfred Heurtaux"@en	"1893-05-19+02:00"^^xsd:date	"1985-12-29+02:00"^^xsd:date	:Alfred_Heurtaux
"Anacharsis Baizeau"@en	"1821-06-02+02:00"^^xsd:date	"1910-02-05+02:00"^^xsd:date	:Anacharsis_Baizeau
"Anne"@en	"1477-01-24+02:00"^^xsd:date	"1514-01-08+02:00"^^xsd:date	:Anne_of_Brittany
"Anne Of Brittany"@en	"1477-01-24+02:00"^^xsd:date	"1514-01-08+02:00"^^xsd:date	:Anne_of_Brittany
"Aristide Briand"@en	"1862-03-27+02:00"^^xsd:date	"1932-03-06+02:00"^^xsd:date	:Aristide_Briand
"Aristide Hignard"@en	"1822-05-19+02:00"^^xsd:date	"1898-03-19+02:00"^^xsd:date	:Aristide_Hignard
"Arthur I"@en	"1187-03-28+02:00"^^xsd:date	"1203-04-02+02:00"^^xsd:date	:Arthur_I,_Duke_of_Brittany
"Auguste Toulmouche"@en	"1829-09-20+02:00"^^xsd:date	"1890-10-15+02:00"^^xsd:date	:Auguste_Toulmouche
"Bl. Mary of the Passion, F.M.M."@en	"1839-05-20+02:00"^^xsd:date	"1904-11-14+02:00"^^xsd:date	:Mary_of_the_Passion
"Charles Lory"@en	"1823-07-29+02:00"^^xsd:date	"1889-05-02+02:00"^^xsd:date	:Charles_Lory
"Charles Monselet"@en	"1825-04-29+02:00"^^xsd:date	"1888-05-18+02:00"^^xsd:date	:Charles_Monselet
"Claude Cahun"@en	"1894-10-24+02:00"^^xsd:date	"1954-12-07+02:00"^^xsd:date	:Claude_Cahun
"Clemence Royer"@en	"1830-04-20+02:00"^^xsd:date	"1902-02-05+02:00"^^xsd:date	:Cl%C3%A9mence_Royer
"Clémence Royer"@en	"1830-04-20+02:00"^^xsd:date	"1902-02-05+02:00"^^xsd:date	:Cl%C3%A9mence_Royer
"Didier Lecour Grandmaison"@en	"1889-05-17+02:00"^^xsd:date	"1917-05-09+02:00"^^xsd:date	:Didier_Lecour_Grandmaison
"Didier Louis Marie Charles Lecour Grandmaison"@en	"1889-05-17+02:00"^^xsd:date	"1917-05-09+02:00"^^xsd:date	:Didier_Lecour_Grandmaison
"Duke of Brittany Peter II"@en	"1418-07-06+02:00"^^xsd:date	"1457-09-21+02:00"^^xsd:date	:Peter_II,_Duke_of_Brittany
"Gaston Serpelle"@en	"1846-11-03+02:00"^^xsd:date	"1904-11-02+02:00"^^xsd:date	:Gaston_Serpette
"Germain Boffrand"@en	"1667-05-15+02:00"^^xsd:date	"1754-03-18+02:00"^^xsd:date	:Germain_Boffrand
"James Tissot"@en	"1836-10-14+02:00"^^xsd:date	"1902-08-07+02:00"^^xsd:date	:James_Tissot
"Jean Alexandre Barre"@en	"1880-05-24+02:00"^^xsd:date	"1967-04-25+02:00"^^xsd:date	:Jean_Alexandre_Barr%C3%A9
"Jean Brochard"@en	"1893-03-11+02:00"^^xsd:date	"1972-06-16+02:00"^^xsd:date	:Jean_Brochard
"Jean Ferdinand Rozier"@en	"1777-11-08+02:00"^^xsd:date	"1863-12-31+02:00"^^xsd:date	:Jean_Ferdinand_Rozier
"Joseph Perotaux"@en	"1883-01-07+02:00"^^xsd:date	"1967-04-22+02:00"^^xsd:date	:Joseph_Perotaux
"Joseph Peroteaux"@en	"1883-01-07+02:00"^^xsd:date	"1967-04-22+02:00"^^xsd:date	:Joseph_Perotaux
"Jules Verne"@en	"1828-02-07+02:00"^^xsd:date	"1905-03-23+02:00"^^xsd:date	:Jules_Verne
"Louis-Albert Bourgault-Ducoudray"@en	"1840-02-01+02:00"^^xsd:date	"1910-07-03+02:00"^^xsd:date	:Louis-Albert_Bourgault-Ducoudray
"Mary Of The Passion"@en	"1839-05-20+02:00"^^xsd:date	"1904-11-14+02:00"^^xsd:date	:Mary_of_the_Passion
"Maxime Maufra"@en	"1861-05-16+02:00"^^xsd:date	"1918-05-22+02:00"^^xsd:date	:Maxime_Maufra
"Michel Coiffard"@en	"1892-07-15+02:00"^^xsd:date	"1918-10-28+02:00"^^xsd:date	:Michel_Coiffard
"Michel Joseph Callixte Marie Coiffard"@en	"1892-07-15+02:00"^^xsd:date	"1918-10-28+02:00"^^xsd:date	:Michel_Coiffard
"Peter II"@en	"1418-07-06+02:00"^^xsd:date	"1457-09-21+02:00"^^xsd:date	:Peter_II,_Duke_of_Brittany
"Pierre Jacques Etienne Cambronne"@en	"1770-12-25+02:00"^^xsd:date	"1842-01-28+02:00"^^xsd:date	:Pierre_Cambronne
"Pierre Jacques Étienne Cambronne"@en	"1770-12-25+02:00"^^xsd:date	"1842-01-28+02:00"^^xsd:date	:Pierre_Cambronne
"Pierre Roy"@en	"1880-08-09+02:00"^^xsd:date	"1950-09-25+02:00"^^xsd:date	:Pierre_Roy_(painter)
"Pierre Waldeck-Rousseau"@en	"1846-12-01+02:00"^^xsd:date	"1904-08-09+02:00"^^xsd:date	:Pierre_Waldeck-Rousseau
"Rene Waldeck-Rousseau"@en	"1846-12-01+02:00"^^xsd:date	"1904-08-09+02:00"^^xsd:date	:Pierre_Waldeck-Rousseau
"Suzanne Malherbe"@en	"1892-07-18+02:00"^^xsd:date	"1972-02-18+02:00"^^xsd:date	:Suzanne_Malherbe

A SPARQL query comprises, in order:

- *Prefix declarations*, for abbreviating URIs
- *Dataset definition*, stating what RDF graph(s) are being queried
- A *result clause*, identifying what information to return from the query
- The *query pattern*, specifying what to query for in the underlying dataset
- *Query modifiers*, slicing, ordering, and otherwise rearranging query results

```
# prefix declarations
PREFIX foo: <http://example.com/resources/> ...
 # dataset definition
FROM ...
# result clause
SELECT ...
# query pattern WHERE { ... }
# query modifiers ORDER BY ...
```

- SPARQL *variables* start with a ? and can match any node (resource or literal) in the RDF dataset.
- *Triple patterns* are just like triples, except that any of the parts of a triple can be replaced with a variable.
- The *SELECT* result clause returns a table of variables and values that satisfy the query.

SPARQL Query forms

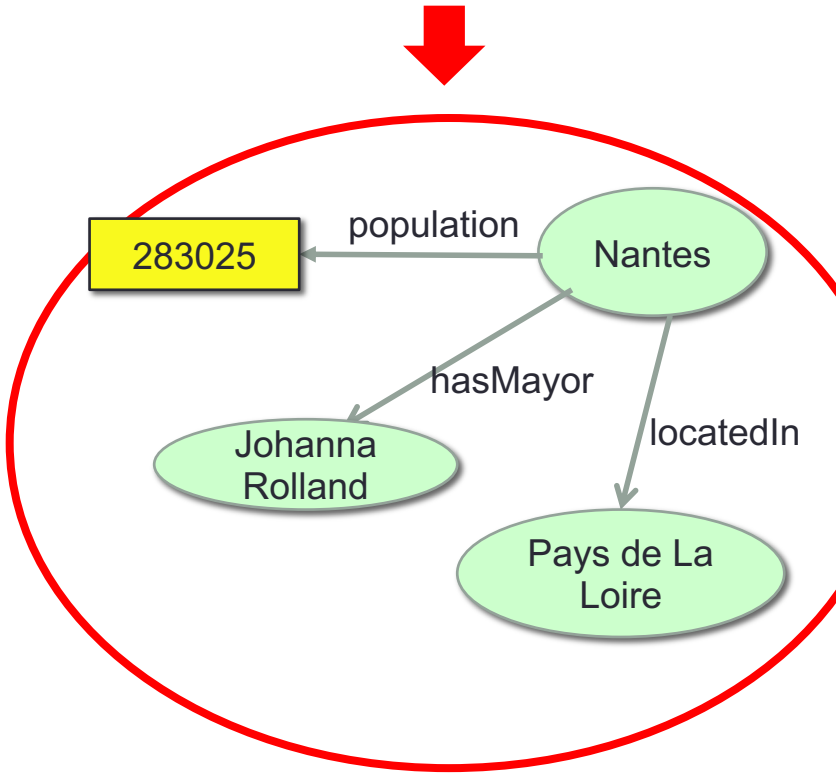
- **Select**
  - Sequence of results (i.e. sets of variable bindings)
  - Selected variables separated by space (not by comma!)
- **Construct**
  - Returns an RDF graph created from a template
  - Template: graph pattern with variables from the query pattern
- **Describe**
  - Returns an RDF graph with data about resources
  - Nondeterministic (i.e. query processor determines the actual structure of the returned RDF graph).
- **Ask**
  - Check whether there is at least one answer

SPARQL

```
PREFIX p: <http://lodpaddle.or/>

SELECT ?mayor
WHERE {
   p:Nantes   p:hasMayor   ?mayor
}
```

Find me all the values for ?mayor such that the triple is true.

Who is the mayor of Nantes?

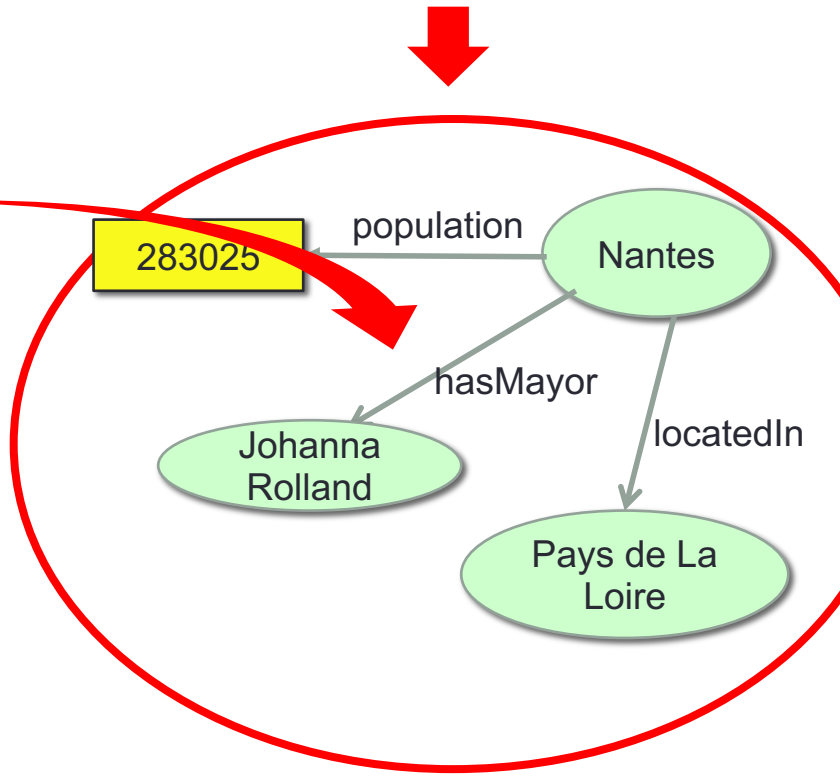


SPARQL: Triple Pattern

SPARQL is based on matching graph patterns

```
PREFIX p: <http://lodpaddle.or/>

SELECT ?mayor
WHERE {
   p:Nantes   p:hasMayor   ?mayor
}
```

Who is the mayor of Nantes?



p:Nantes p:hasMayor ?mayor

mayor

---

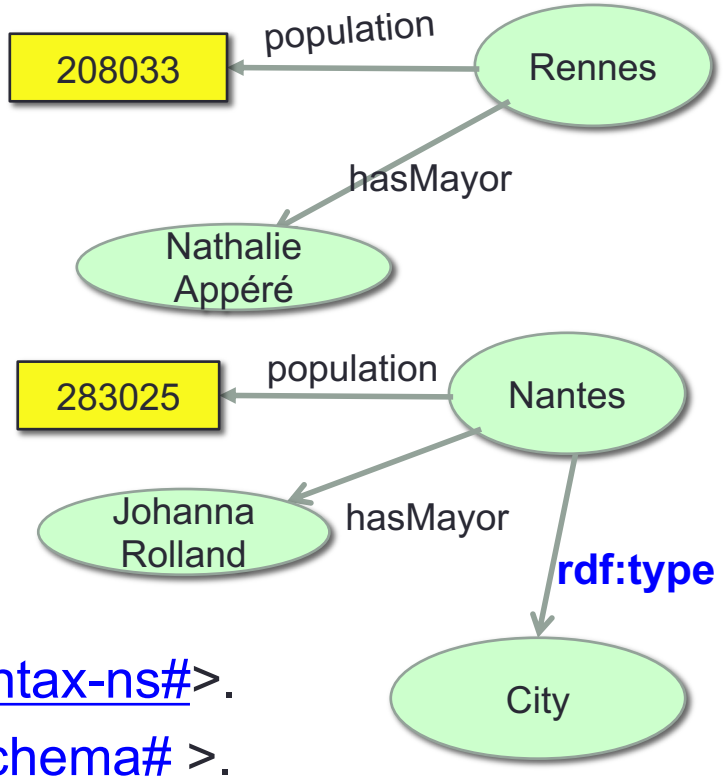
p:Joanna Rolland

Basic Graph Patterns: set of triple patterns

```
PREFIX p: <http://lodpaddle.or/>
PREFIX rdf: <http://w3c.og/..  />

SELECT ?x ?mayor
WHERE {
   ?x   p:hasMayor   ?mayor .
   ?x    rdf:type     p:City
}
```

x	mayor
p:Nantes	p:Johanna Roland

RDF Data



@prefix p: <http://lodpaddle.or/> .
@prefix rdf: <http://w3c.org/1999/02/22-rdf-syntax-ns#>.
@prefix xsd: <http://www.w3c.org/2001/XMLSchema# >.

p:Nantes p:poplulation “283025”^^xsd:integer ;
p:hasMoyer p:JohannaRolland ;
rdf:type p:City .
P:Rennes p:population “208022”^^xsd:integer;
p:hasMayor p:NatalieAppéré .

Data Set:

@prefix p: <http://lodpaddle.org/> .
@prefix rdf: http://w3c.org/1999/02/22-rdf-syntax-ns#>.
@prefix xsd: http://www.w3c.org/2001/XMLSchema# .


```
p:Nantes        p:poplulation     “283025”^^xsd:integer ;
                p:hasMoyer        p:JohannaRolland ;
                rdf:type          p:City .
P:Rennes        p:population      “208022”^^xsd:integer;
                p:hasMayor        p:NatalieAppéré .
```

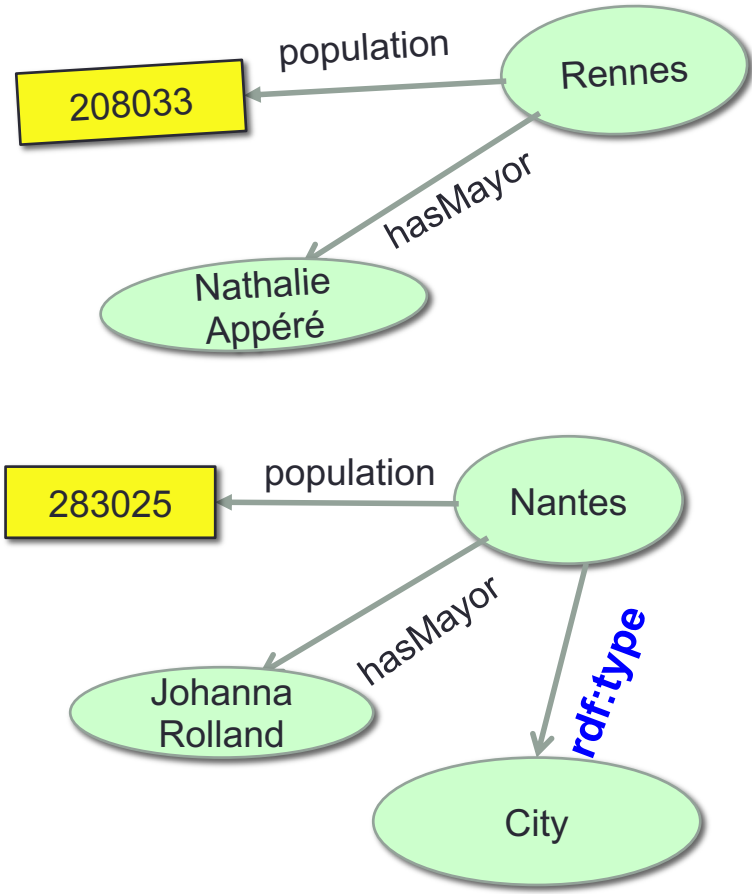
Query:

```
PREFIX p: <http://lodpaddle.org/ >
PREFIX rdf: <http://w3c.org/1999/02/22-rdf-syntax-ns#>

SELECT ?x ?mayor
WHERE {
   ?x   p:hasMayor   ?mayor ;
        rdf:type     p:City
        }
```

x	mayor
p:Nantes	p:Johanna Roland

SPARQL Filter

```
PREFIX p: <http://lodpaddle.org/>
PREFIX rdf: <http://w3c.og/..  />

SELECT ?x
WHERE {
   ?x   p:population   ?nbpop  .
   Filter (?nbpop > 250000)
}
```



x

P:Nantes

SPARQL Filter (2)

Retrieve titles starts with SPARQL .

Data:

```
@prefix dc:   <http://purl.org/dc/elements/1.1/> .
@prefix :     <http://example.org/book/> .
@prefix ns:   <http://example.org/ns#> .

:book1  dc:title  "SPARQL Tutorial" .
:book1  ns:price  42 .
:book2  dc:title  "The Semantic Web" .
:book2  ns:price  23 .
```

Query:

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
SELECT  ?title
WHERE   { ?x dc:title ?title
          FILTER regex(?title, "^SPARQL")
        }
```

Query Result:

title
"SPARQL Tutorial"

Scope of Filters (3)

Data:

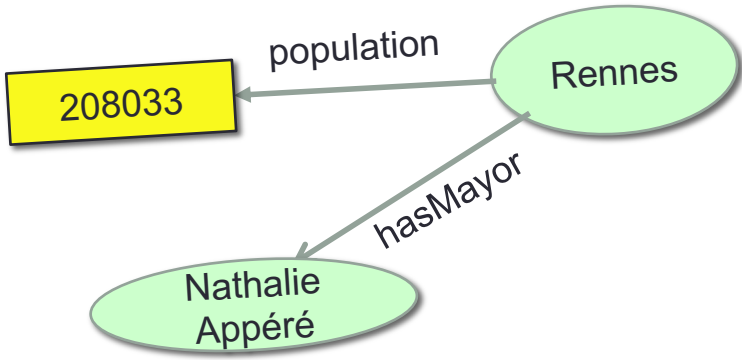
```
@prefix dc:   <http://purl.org/dc/elements/1.1/> .
@prefix :     <http://example.org/book/> .
@prefix ns:   <http://example.org/ns#> .

:book1  dc:title  "SPARQL Tutorial" .
:book1  ns:price  42 .
:book2  dc:title  "The Semantic Web" .
:book2  ns:price  23 .
```

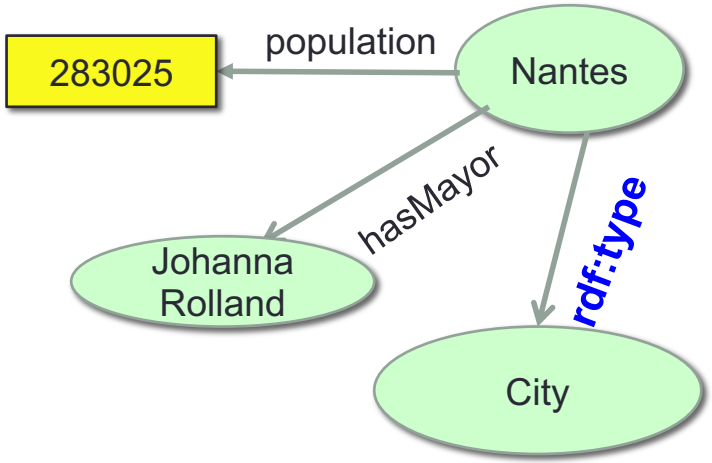
```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
PREFIX  ns:  <http://example.org/ns#>
SELECT  ?title ?price
WHERE   { ?x ns:price ?price .
          FILTER (?price < 30.5)
          ?x dc:title ?title . }
```

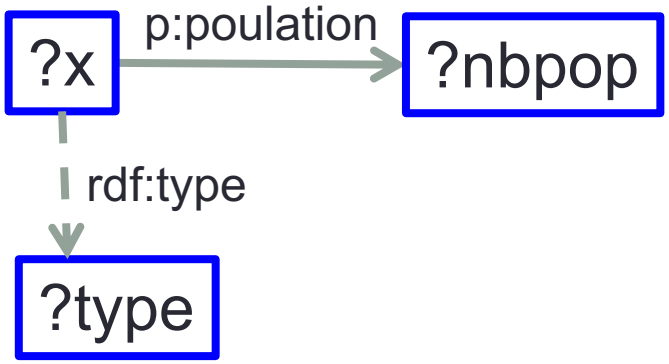
title	price
"The Semantic Web"	23

SPARQL OPTIONAL

```
PREFIX p: <http://lodpaddle.org/>
PREFIX rdf: <http://w3c.og/..  />

SELECT ?x ?type
WHERE {
   ?x   p:population   ?nbpop  .
   OPTIONAL {?x  rdf:type ?type }
}
```





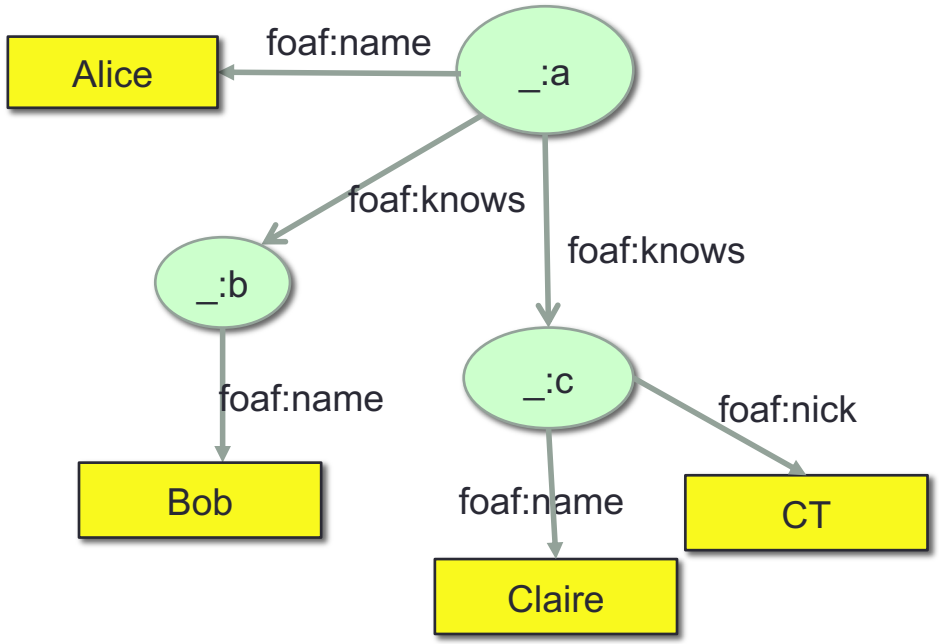


x	type
p:Nantes	p:City
p:Rennes	

SPARQL OPTIONAL

Retrieve the name of a person and the name of her friends and their nickname, if it exist..

```
PREFIX foaf :http://xmlns.com/foaf/0.1/

SELECT ?nameX ?nameY ?nickY
WHERE {
   ?x     foaf:kowns       ?y;
          foaf:name        ?nameX .
   ?y     foaf:name        ?nameY .
   OPTIONAL {?y foaf:nick ?nickY }
   }
```



nameX	nameY	nickY
"Alice"	"Bob"	
"Alice"	"Clare"	"CT"

SPARQL Filter and Optional

```
@prefix dc:   <http://purl.org/dc/elements/1.1/> .
@prefix :     <http://example.org/book/> .
@prefix ns:   <http://example.org/ns#> .

:book1  dc:title  "SPARQL Tutorial" .
:book1  ns:price  42 .
:book2  dc:title  "The Semantic Web" .
:book2  ns:price  23 .
```

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
PREFIX  ns:  <http://example.org/ns#>
SELECT  ?title ?price
WHERE   { ?x dc:title ?title .
          OPTIONAL { ?x ns:price ?price . FILTER (?price < 30) }
        }
```

title	price
"SPARQL Tutorial"	
"The Semantic Web"	23

Alternative Graph Pattern

```
@prefix dc10:  <http://purl.org/dc/elements/1.0/> .
@prefix dc11:  <http://purl.org/dc/elements/1.1/> .

_:a  dc10:title     "SPARQL Query Language Tutorial" .
_:a  dc10:creator   "Alice" .

_:b  dc11:title     "SPARQL Protocol Tutorial" .
_:b  dc11:creator   "Bob" .

_:c  dc10:title     "SPARQL" .
_:c  dc11:title     "SPARQL (updated)" .
```

```
PREFIX dc10:  <http://purl.org/dc/elements/1.0/>
PREFIX dc11:  <http://purl.org/dc/elements/1.1/>

SELECT ?title
WHERE  { { ?book dc10:title  ?title } UNION { ?book dc11:title  ?title } }
```

title
"SPARQL Protocol Tutorial"
"SPARQL"
"SPARQL (updated)"
"SPARQL Query Language Tutorial"

Solution Modifiers

- **Order by**: put the solutions in order
- **Distinct** : removes duplicates from the result set
- **Offset** : control where the solutions start from in the overall sequence of solutions
- **Limit :** puts an upper bound on the number of solutions returned

Examples

Q1: PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE {
?x foaf:name ?name }
ORDER BY ASC(?name) /* trier les résultats */

Q2: PREFIX : <http://example.org/ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX xsd: http://www.w3.org/2001/XMLSchema#
SELECT ?name
WHERE { ?x foaf:name ?name ;
}
ORDER BY **DESC**(?name)
Limit 5

Examples

```
_:x foaf:name "Alice" .
_:x foaf:mbox <mailto:alice@example.com> .
_:y foaf:name "Alice" .
_:y foaf:mbox  <mailto:asmith@example.com> .
_:z foaf:name "Alice" .
_:z foaf:mbox <mailto:alice.smith@example.com> .
```

Q1: PREFIX foaf: <http://xmlns.com/foaf/0.1/>

```
SELECT ?name
WHERE
{
 ?x foaf:name ?name }
```

Q2: PREFIX foaf: <http://xmlns.com/foaf/0.1/>

```
SELECT DISTINCT ?name
WHERE
 {
 ?x foaf:name ?name }
```

Construct

Question : construct the *foaf* graph containing the name of employees

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

_:a   org:employeeName    "Alice" .
_:a   org:employeeId      12345 .

_:b   org:employeeName    "Bob" .
_:b   org:employeeId      67890 .
```

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
PREFIX org:    <http://example.com/ns#>

CONSTRUCT { ?x foaf:name ?name }
WHERE  { ?x org:employeeName ?name }
```

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

_:x foaf:name "Alice" .
_:y foaf:name "Bob" .
```

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    >
  <rdf:Description>
    <foaf:name>Alice</foaf:name>
  </rdf:Description>
  <rdf:Description>
    <foaf:name>Bob</foaf:name>
  </rdf:Description>
</rdf:RDF>
```

SPARQL 1.1 Federated Query

- Remote source

http://people.example.org

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

 :people15 foaf:name "Alice" .
 :people16 foaf:name "Bob" .
 :people17 foaf:name "Charles" .
 :people18 foaf:name "Daisy" .
```

- Local data:

http://example.org/myfoaf.rdf

```
<http://example.org/myfoaf/I>
   <http://xmlns.com/foaf/0.1/knows>
      <http://example.org/people15> .
```

SPARQL 1.1 Federated Query

Remote source
<http://people.example.org/sparql>

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

 :people15 foaf:name "Alice" .
 :people16 foaf:name "Bob" .
 :people17 foaf:name "Charles" .
 :people18 foaf:name "Daisy" .
```

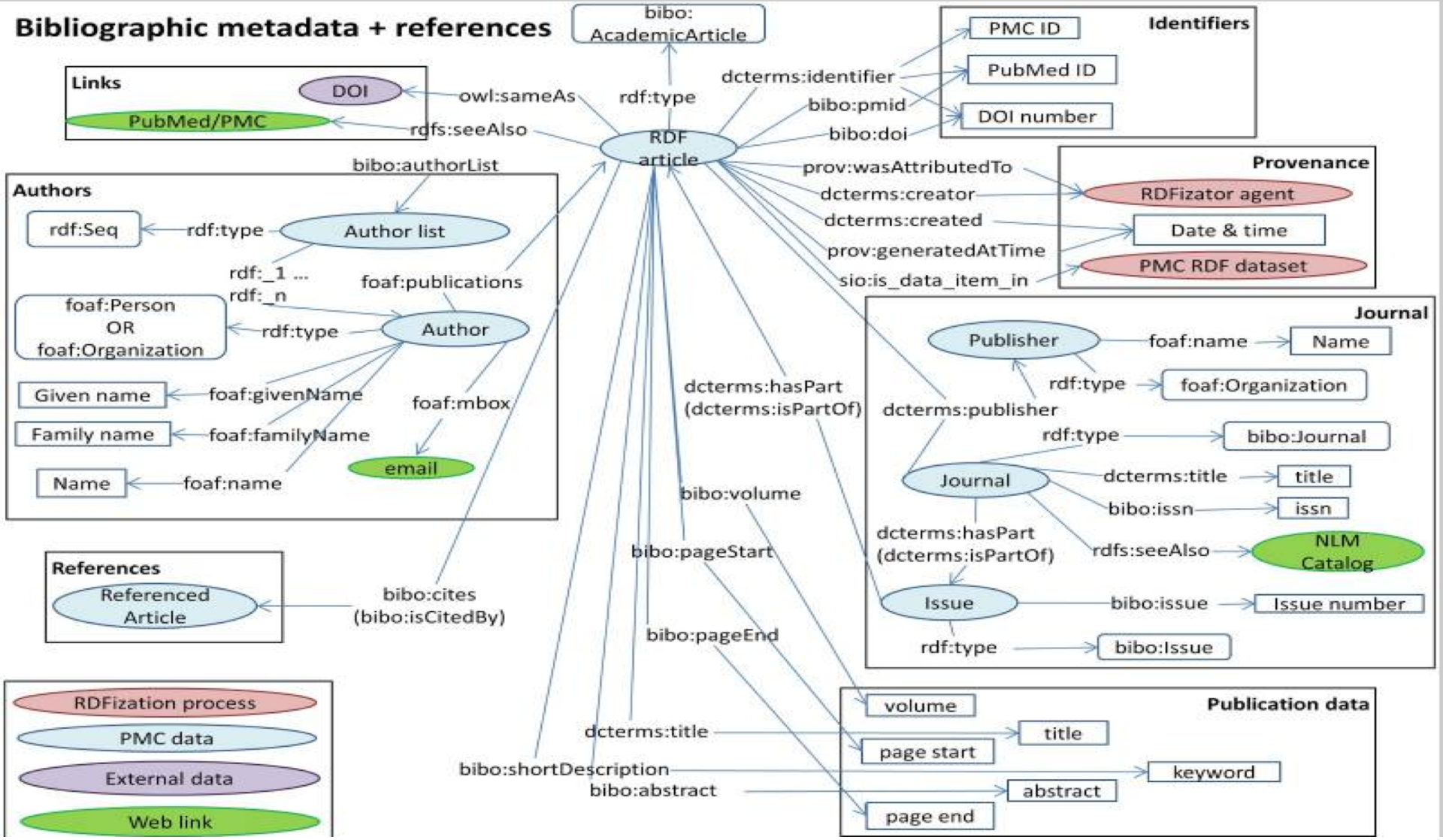
Local data: http://example.org/myfoaf.rdf

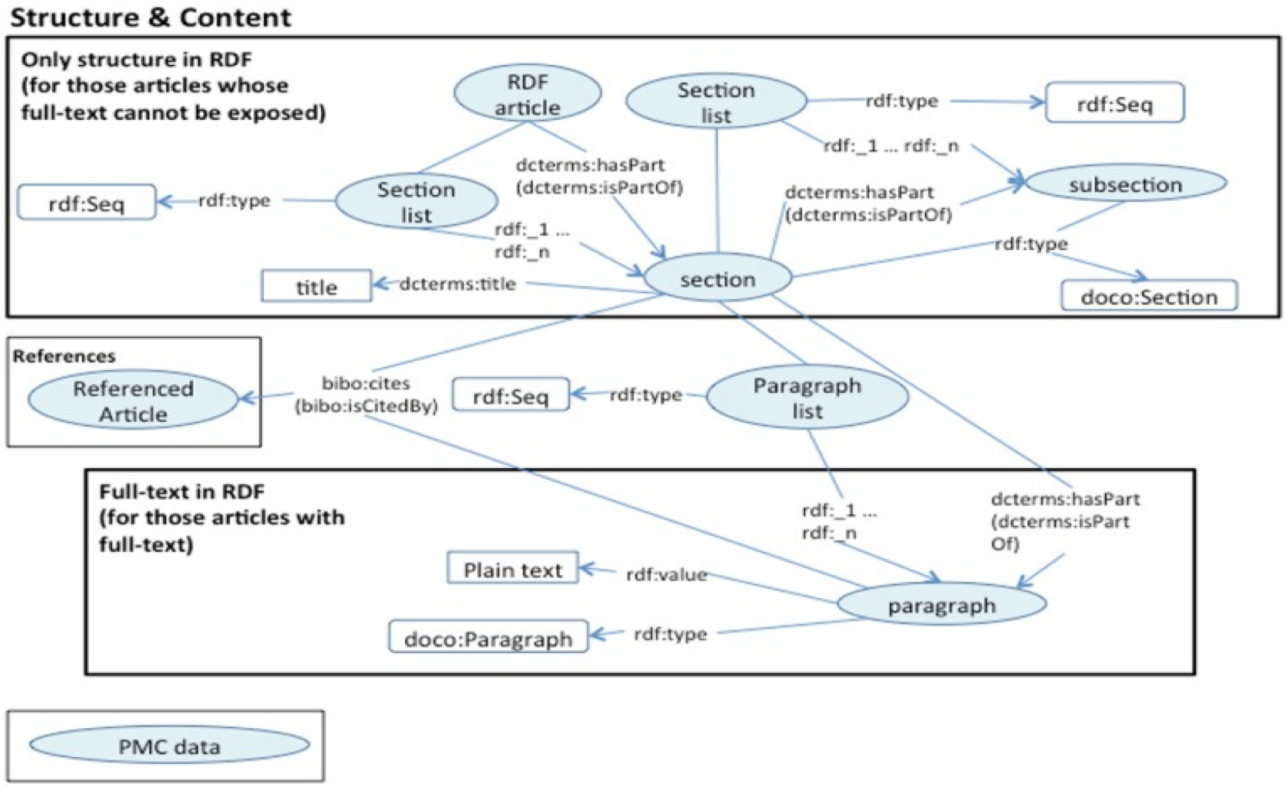
```
<http://example.org/myfoaf/I>
    <http://xmlns.com/foaf/0.1/knows>
        <http://example.org/people15> .
```

```
PREFIX foaf:
<http://xmlns.com/foaf/0.1/>
SELECT ?name
FROM <http://example.org/myfoaf.rdf>
WHERE
{
  <http://example.org/myfoaf/I>
foaf:knows ?person .
  SERVICE
<http://people.example.org/sparql> {
    ?person foaf:name ?name . }
}
```

```
name
--------
Alice
```

Example: Biotea: semantics for Pubmed Central (https://www.ncbi.nlm.nih.gov/pmc/article)





Text structure RDF model.

Retrieve “materials and methods using chloroplast DNA isolation methods”

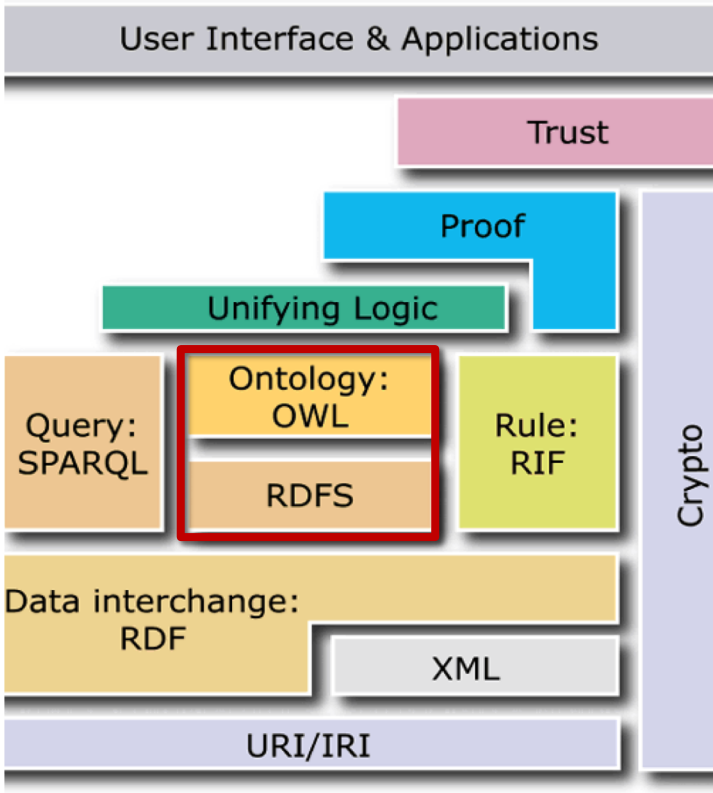
http://biotea.github.io/queries/

Summary

- SPARQL is a protocol and query language for RDF data model..
- It designed for open, decentralized Web
- **Select** is suitable for querying known endpoint with known vocabularies
- **Other Forms:**
  - **Describe** is suitable for known IRI and unknown vocabularies, the results is a RDF graph describing the requested resource
  - **Ask** discover which SPARQL endpoint could answer the query
  - **Construct** to build a graph as a result

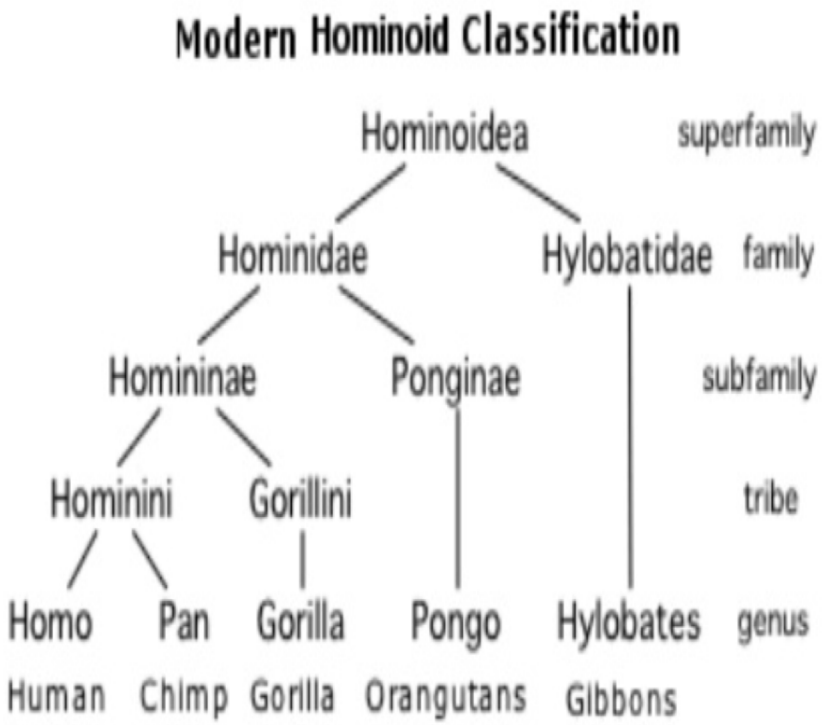
Ontology languages

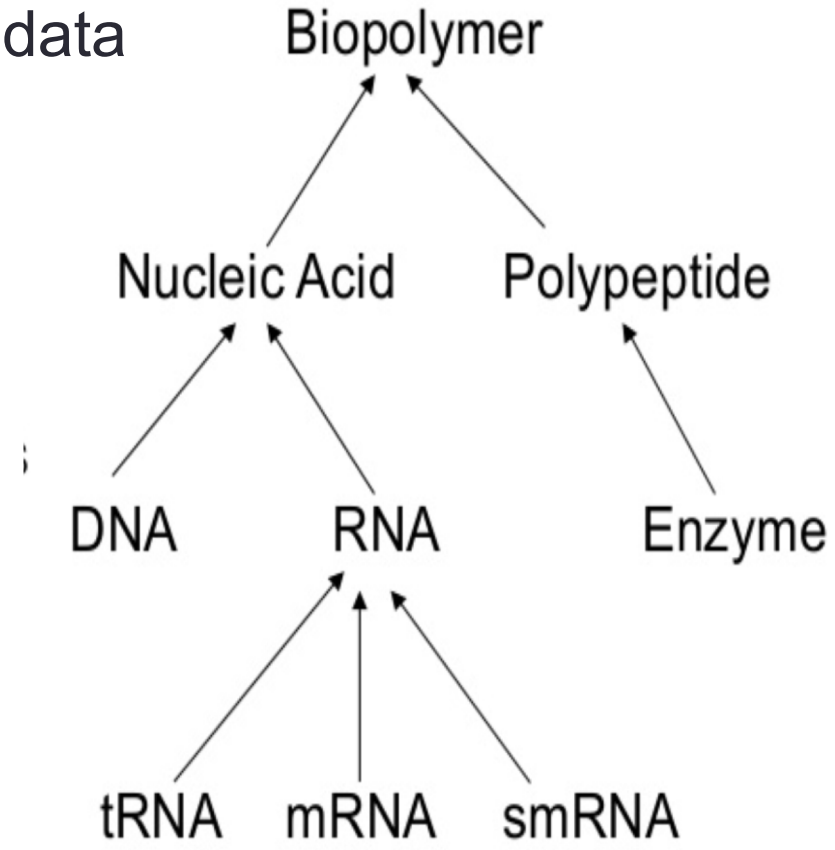
- Representation of important things in a specific domain
  - Describe types of entities (eg cells) and relations between them
- The main requirements are:
  - a well-defined syntax
  - a formal semantics
  - sufficient expressive power
  - efficient reasoning support



Ontologies for Life Science

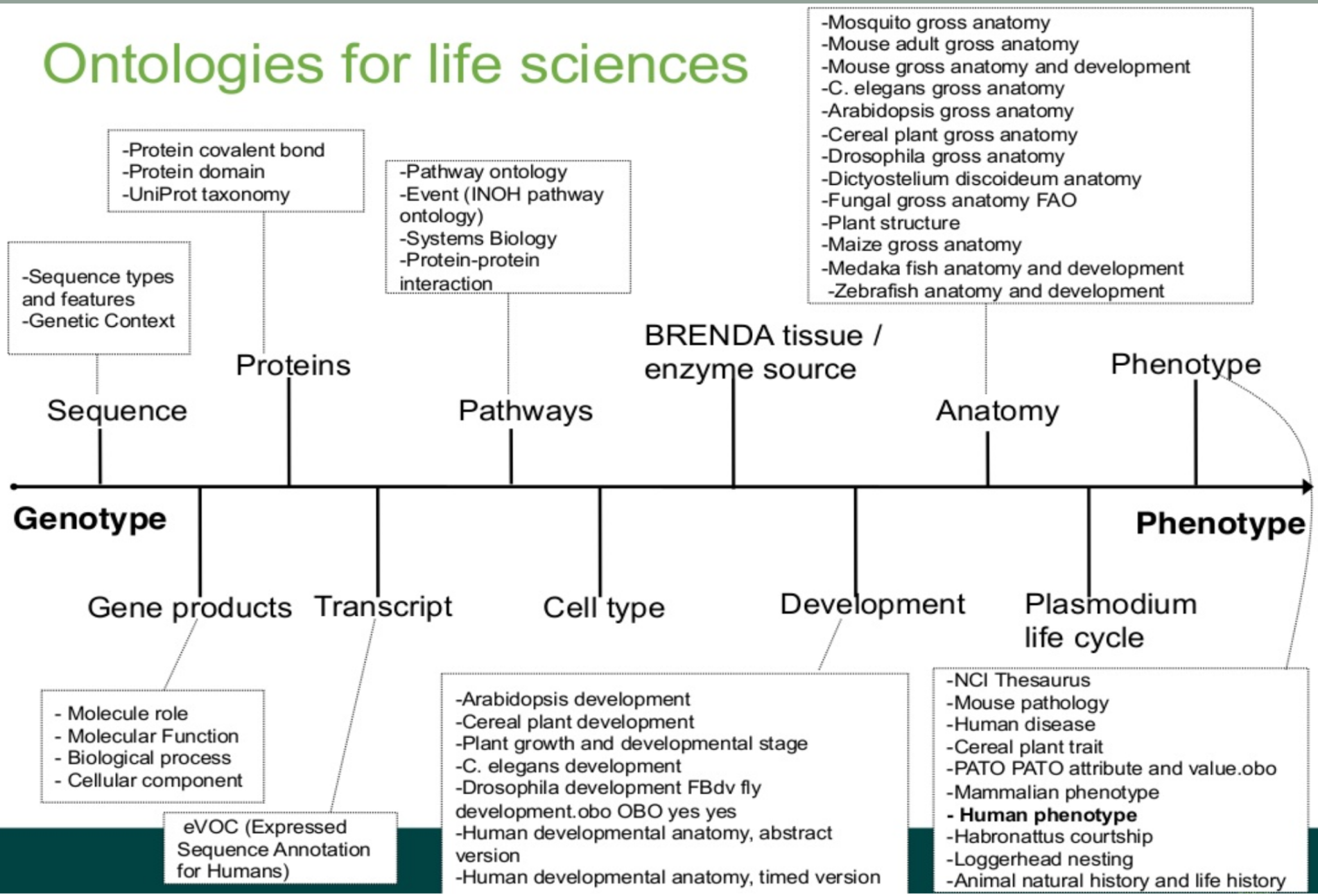
- Put things into categories
  - Helps organize the data
  - Allows to generalize over data





Source: https://fr.slideshare.net/mcourtot/ontologies-for-life-sciences-examples-from-the-gene-ontology

Ontologies for life sciences



Source: https://fr.slideshare.net/mcourtot/ontologies-for-life-sciences-examples-from-the-gene-ontology

Ontology= Schema +instances

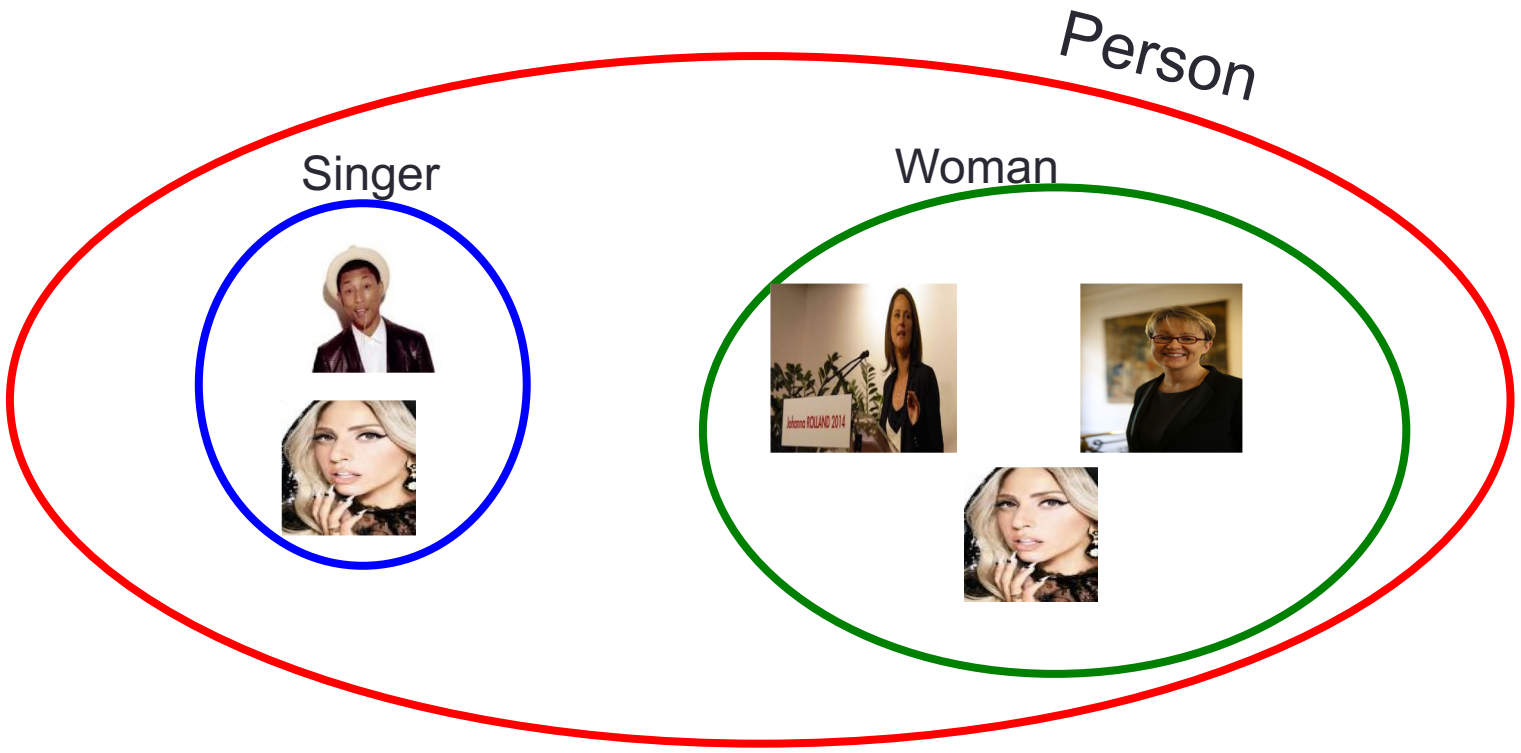
- **Schema (TBox)**
  - The set of **class** and **relation** names
  - The **constraints** that are used for two purposes
    - checking data consistency (like dependencies in databases)
    - inferring new facts
- **Instance (ABox)**
  - The set of facts
  - The set of base facts together with the inferred facts should satisfy the constraints
- Ontology (i.e., Knowledge Base, knowledge Graph) = Schema + Instance

RDFS

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

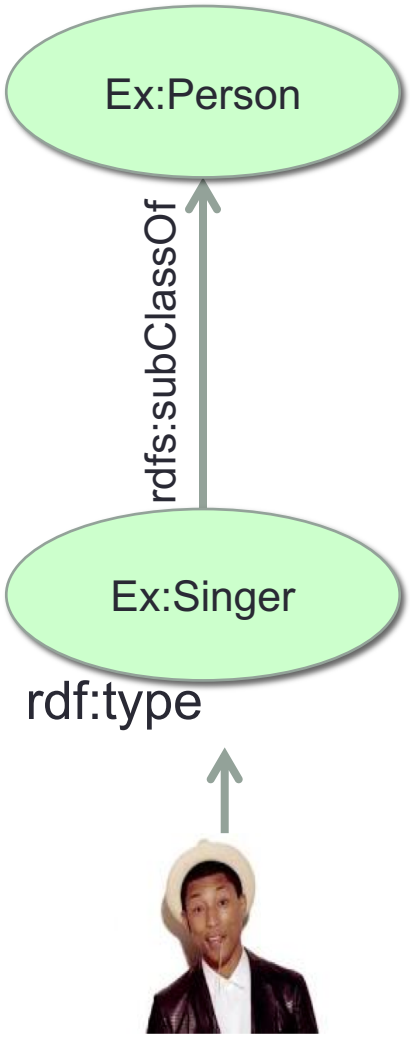
Classes

- A **class** (also called concept) can be understood as a set of similar entities



RDFS Classes

The fact that an entity belongs to a class is expressed by the **type** predicate from the standard namespace rdf (http://www.w3.org/2000/01/rdf-schema#”). (rdf:type)



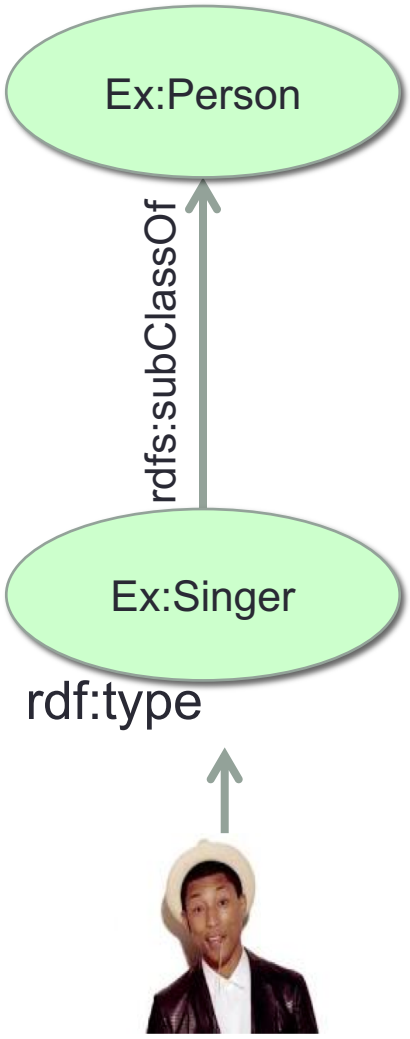
@PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema/>
@PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax/>
@PREXIF ex: http://example.org/

Ex:Williams rdf:type ex:Singer .

RDFS Classes

The fact that a class is a sub-class of another class is expressed by the
subclassOf predicate from the standard namespace **rdfs** (http://w3c.org/...).

@PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema/>
@PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax/>
@PREXIF ex: http://example.org/

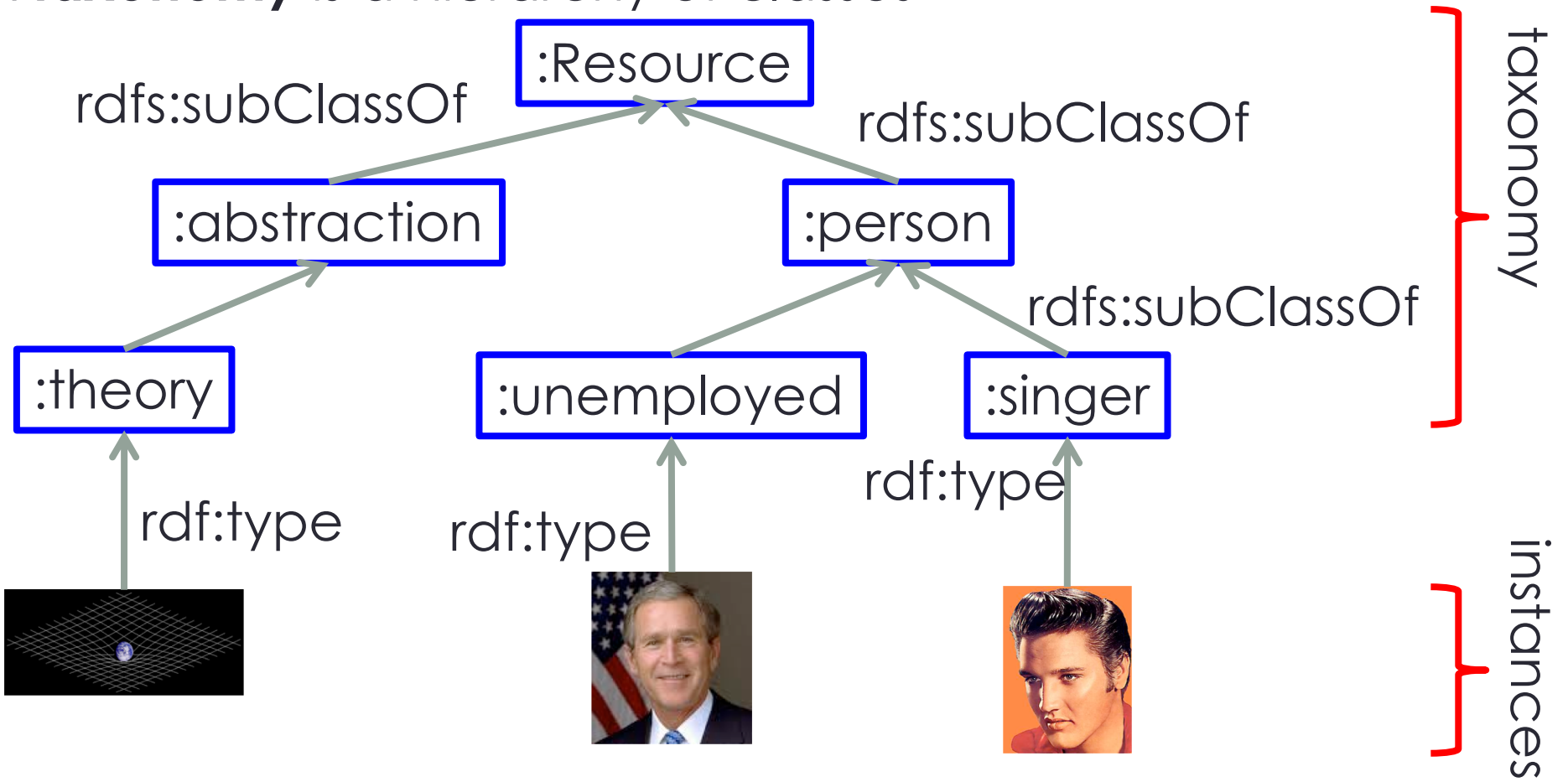


Ex:Williams rdf:type ex:Singer .

Ex:Singer rdfs:subClassOf ex:Person.

Taxonomy

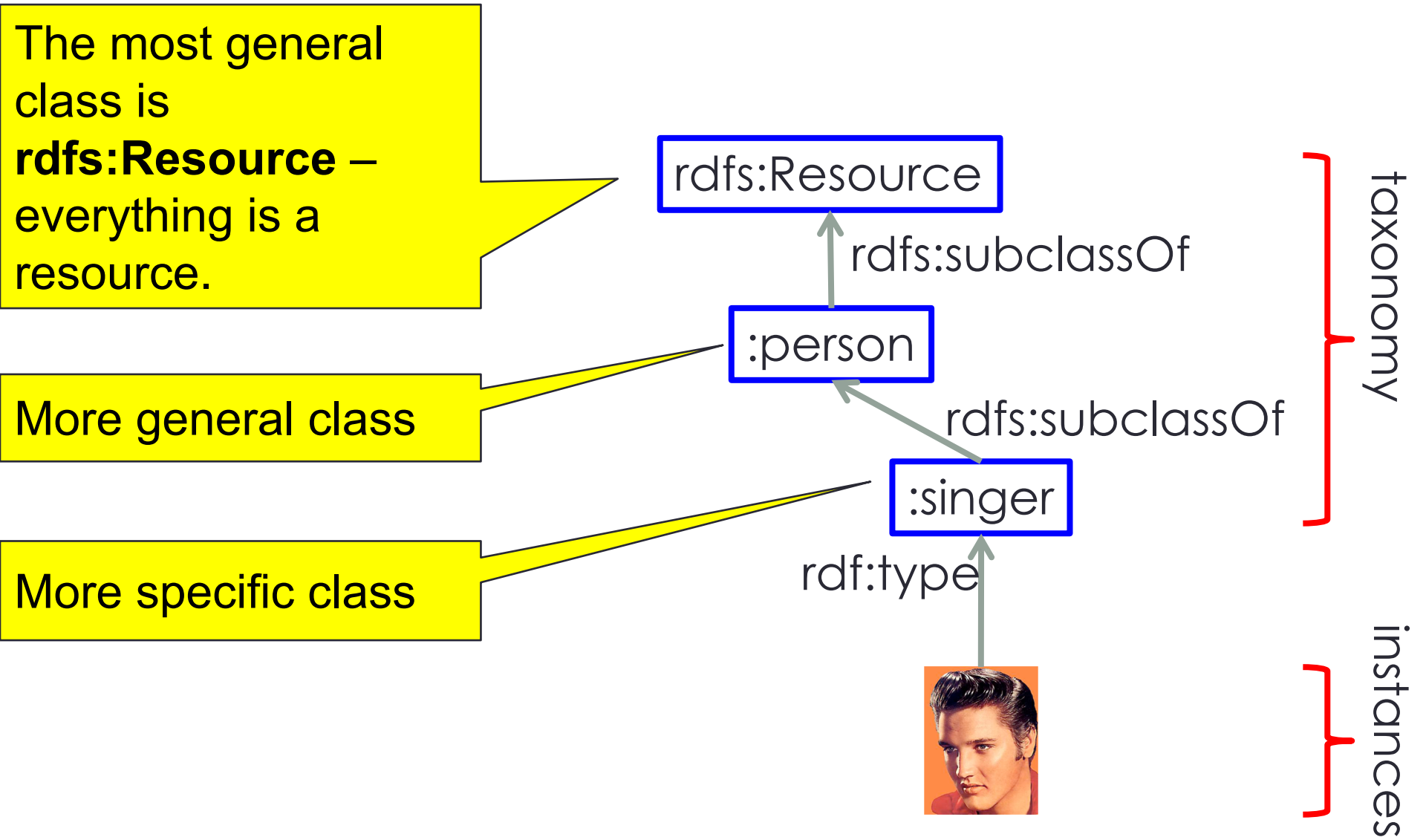
A **taxonomy** is a hierarchy of classes



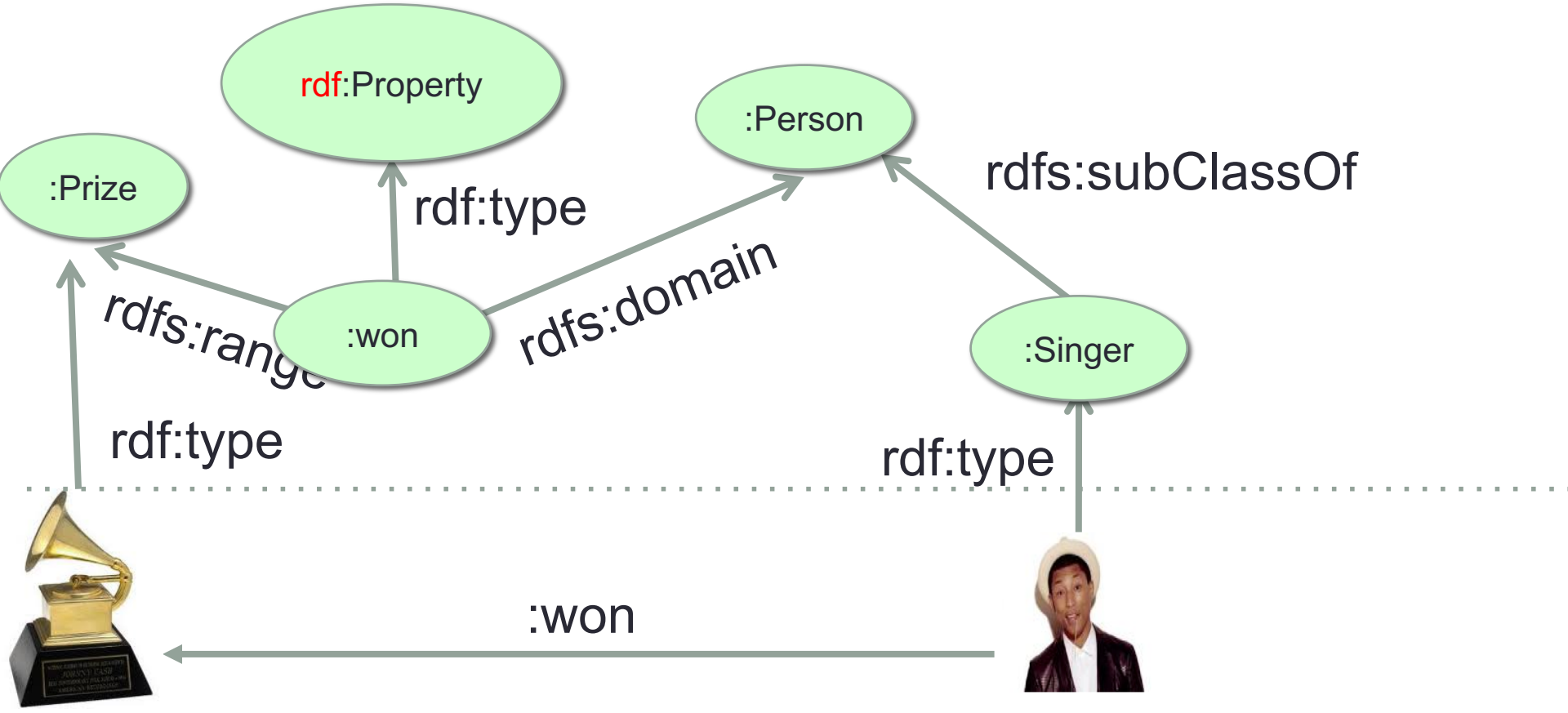
The most general class is **rdfs:Resource** – everything is a resource.

More general class

More specific class



Domain & Range



RDFS logical semantics

RDF and RDFS statements	FOL translation	DL notation
⟨ i rdf:type C ⟩	$C(i)$	$i : C$ or $C(i)$
⟨ i P j ⟩	$P(i,j)$	iPj or $P(i,j)$
⟨ C rdfs:subClassOf D ⟩	$\forall X\,(C(X) \Rightarrow D(X))$	$C \sqsubseteq D$
⟨ P rdfs:subPropertyOf R ⟩	$\forall X \forall Y\,(P(X,Y) \Rightarrow R(X,Y))$	$P \sqsubseteq R$
⟨ P rdfs:domain C ⟩	$\forall X \forall Y\,(P(X,Y) \Rightarrow C(X))$	$\exists P \sqsubseteq C$
⟨ P rdfs:range D ⟩	$\forall X \forall Y\,(P(X,Y) \Rightarrow D(Y))$	$\exists P^{-} \sqsubseteq D$

Reasoning

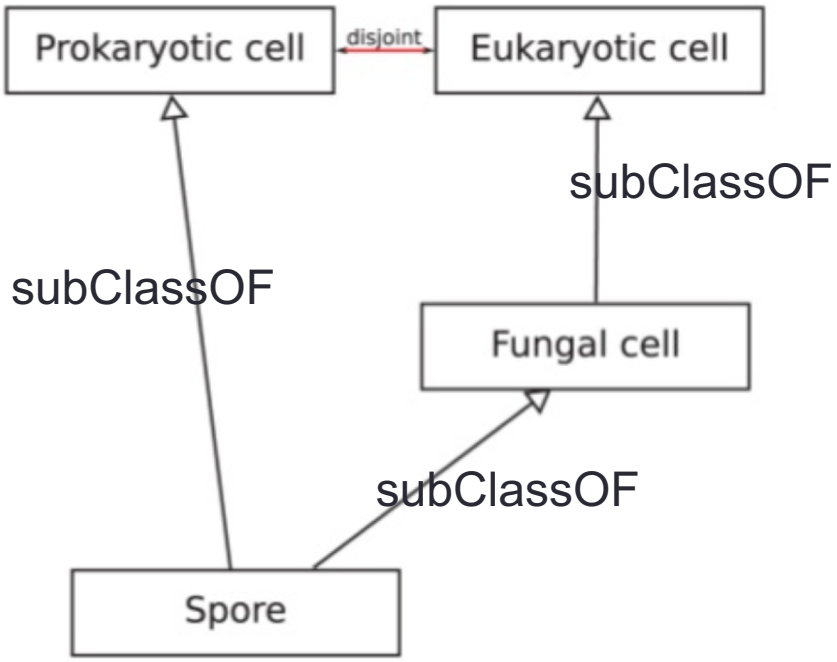
- Deduce new facts based on existing ones
- Examples of tasks required from reasoner :
  - **Satisfiability of a concept**
    - Determine whether a description of the concept is not contradictory, i.e., whether an individual can exist that would be instance of the concept.
  - **Subsumption of concepts**
    - Determine whether concept *C* subsumes concept *D*, i.e., whether description of *C* is more general than the description of *D*.
  - **Consistency of ABox with respect to TBox**
    - Determine whether individuals in ABox do not violate descriptions and axioms described by TBox.

Reasoning is critical

- Prokaryotic and Eukaryotic cell are declared disjoints
- Fungal cell is a Eukaryotic cell
- Spore is a Fungal cell and a Prokaryotic cell

⇒Unsatisfiability

⇒Solution: clarify spore (sensu Mycetozoa) AND actinomycete-type spore



http://www.plosone.org/article/info:doi/10.1371/journal.pone.0022006

Source: https://fr.slideshare.net/mcourtot/ontologies-for-life-sciences-examples-from-the-gene-ontology

Type Inferences

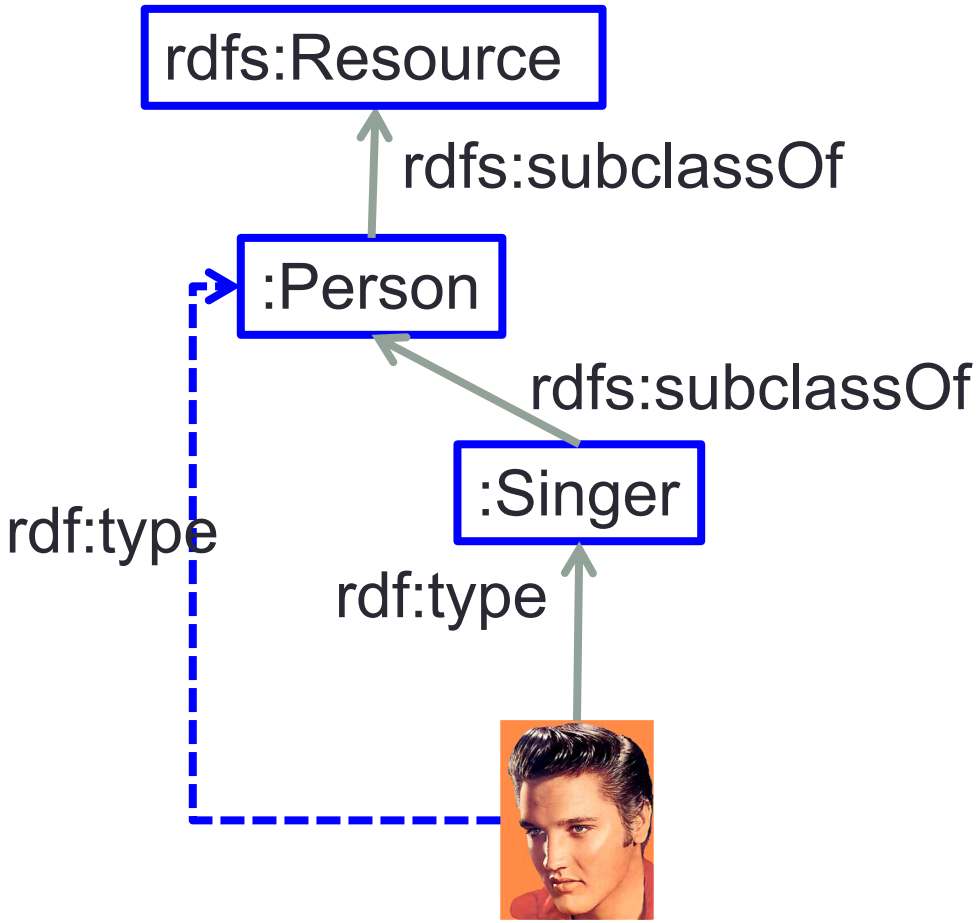
(:s rdf:type :t)

(:t rdf:type rdfs:Class)

Type Inefrences

(:s rdf:type :t)

(:t rdf:type rdfs:Class)

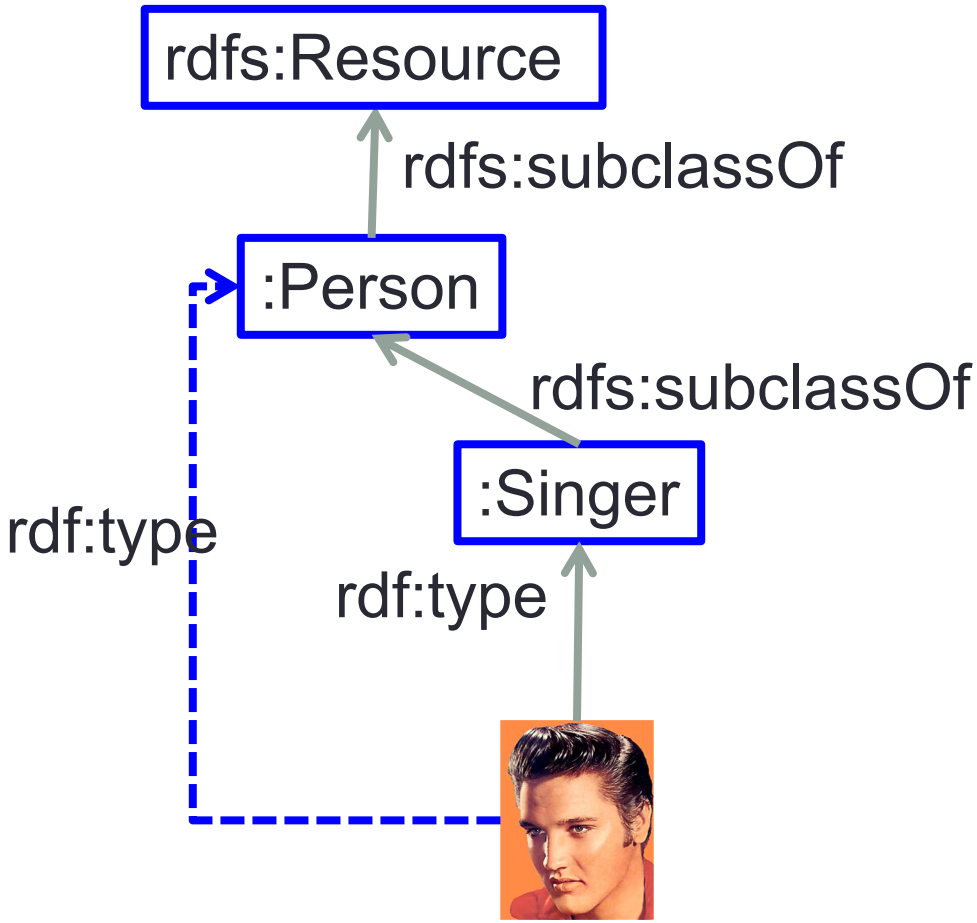


Type Inferences

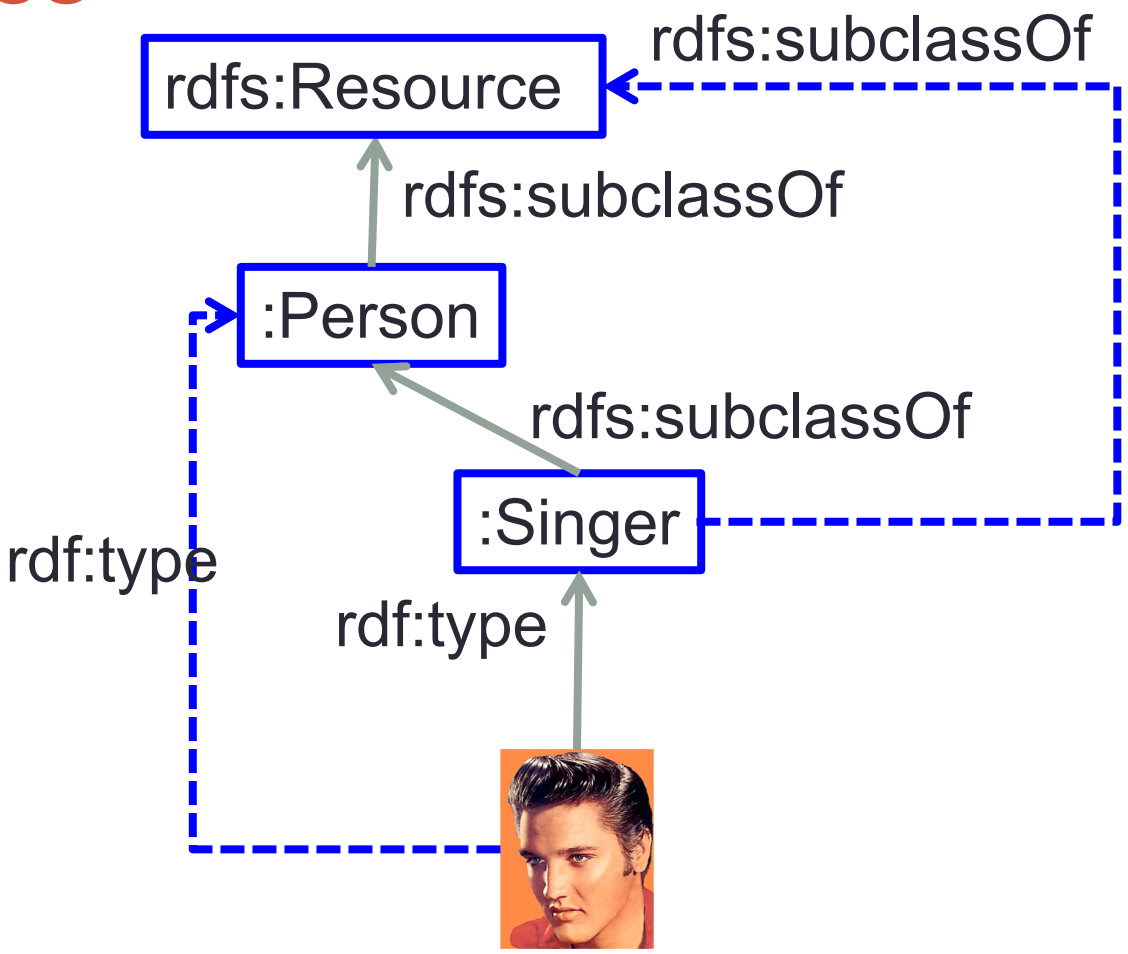
(:x rdf:type :t)
(:t rdfs:subClassOf: :z)

(:x rdf:type :z)

Every instance is an instance of all more general classes



Type Inferences

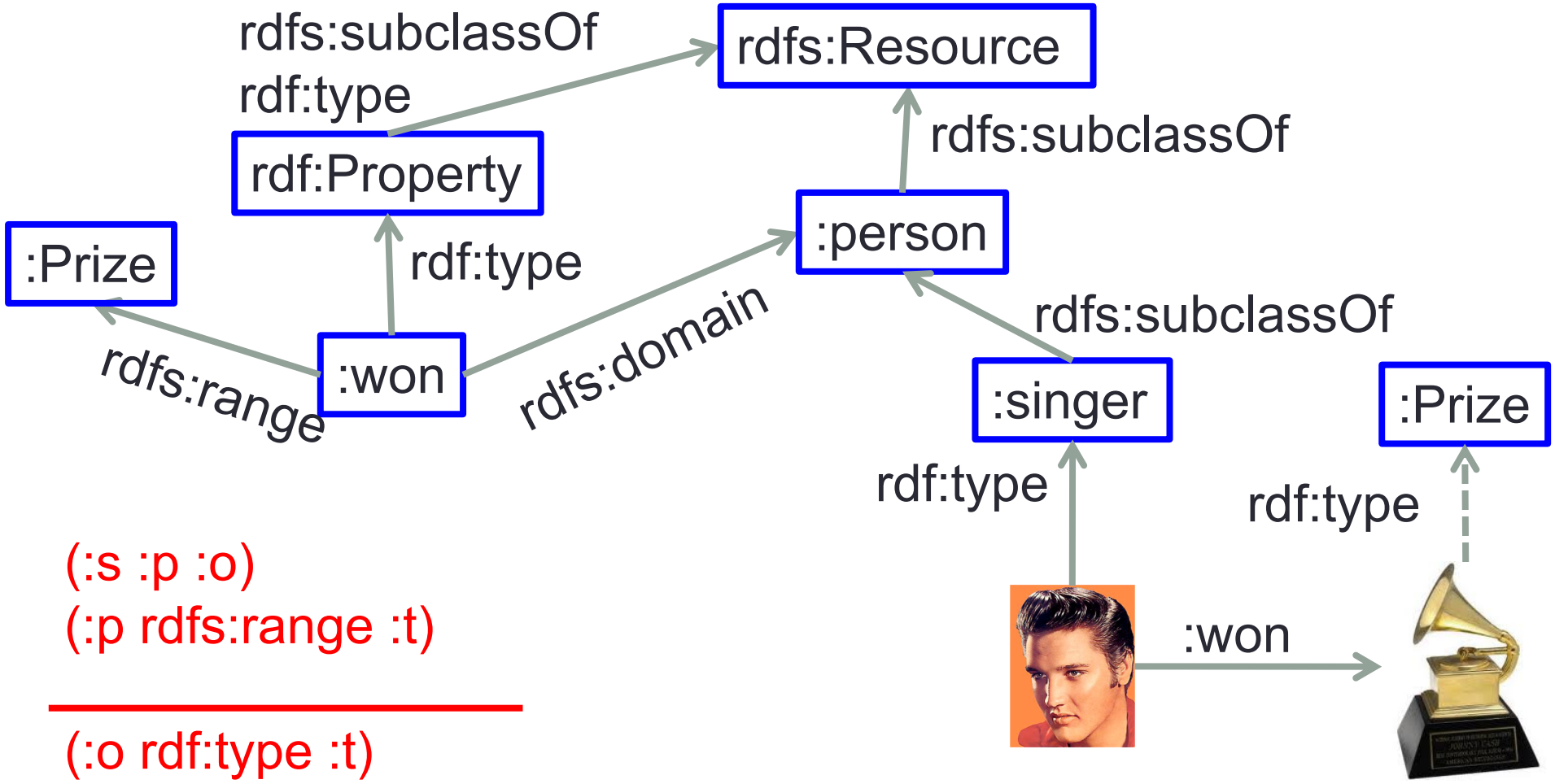


(:x rdf:type :t)
(:t rdfs:subClassOf: :z)

(:x rdf:type :z)

Every instance is an instance of all more general classes

Domain & Range Semantics



(:s :p :o)
(:p rdfs:range :t)

(:o rdf:type :t)

Same for domain

RDFS

- **Classes** and **Properties**
- **Class Hierarchies** and **Inheritance**
- **Property Hierarchies** and **Inheritance**
- **Domain** and **range**

- **RDFS does not allow to define:**
  - **Class complex:**
    - **intersection, union, negation**
  - **Cardinality**
- **Goto OWL**

Ontology Web Language (OWL)

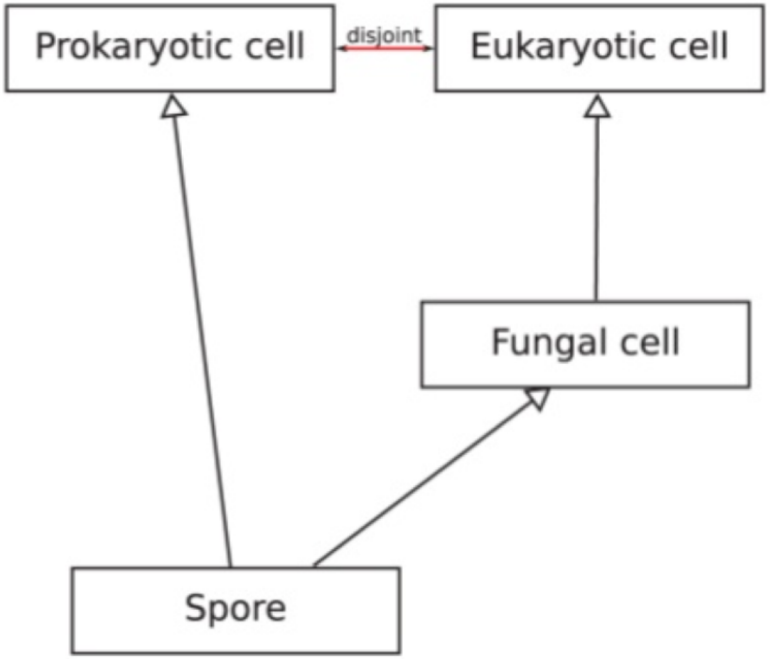
- Extension to RDFS (RDF Schema) based on description logic (DL)
  - Disjoint classes: man, woman
  - Boolean combinations of classes: person is the disjoint union of the classes man and woman
  - Cardinality restrictions: a person has exactly two parents
  - Special characteristics of properties:Transitive property, Inverse property..
- In reality, it is based on restriction (type separation) of RDF:
  - Class ≠ property ≠ Individuals

Reasoning is critical

- Prokaryotic and Eukaryotic cell are declared disjoints
- Fungal cell is a Eukaryotic cell
- Spore is a Fungal cell and a Prokaryotic cell

⇒Unsatisfiability

⇒Solution: clarify spore (sensu Mycetozoa) AND actinomycete-type spore



http://www.plosone.org/article/info:doi/10.1371/journal.pone.0022006

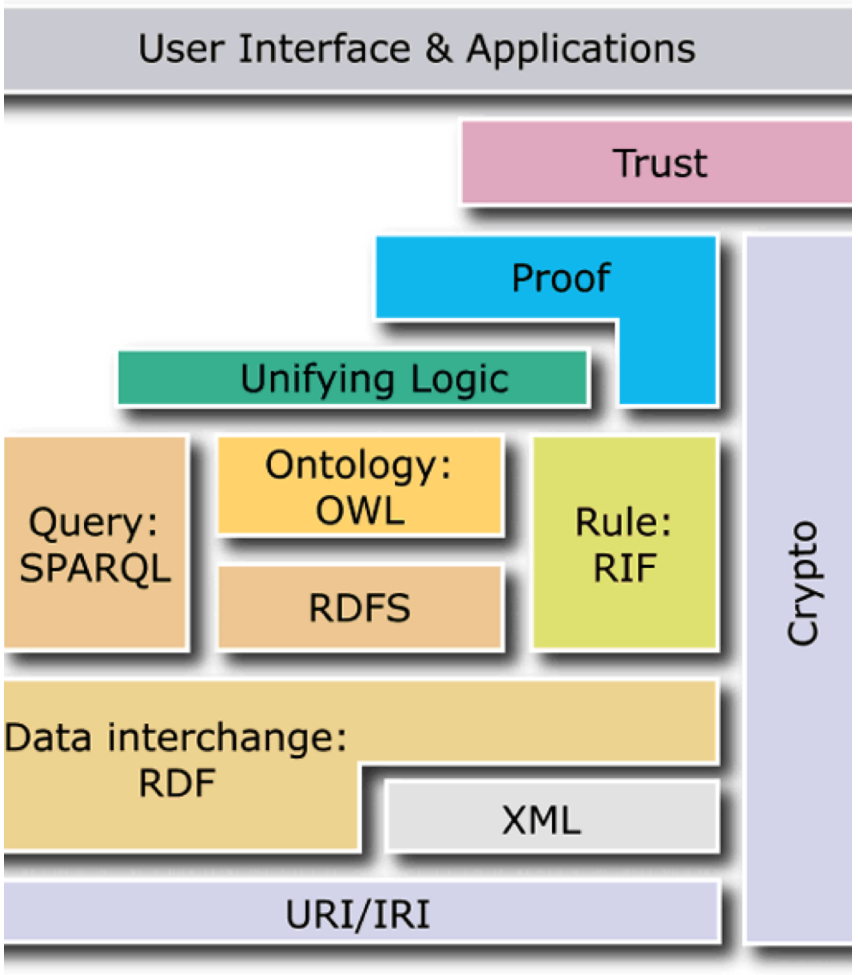
Source: https://fr.slideshare.net/mcourtot/ontologies-for-life-sciences-examples-from-

Example OWL Ontology

```
@prefix owl: <http://www.w3.org/2002/07/owl#>

:NoMeatPizza rdf:type owl:Class ;
             rdfs:subClassOf :Pizza .

:hasTopping rdf:type owl:ObjectProperty ;
            rdfs:domain :Pizza ;
            rdfs:range :PizzaTopping .


:MozzarellaTopping rdf:type owl:Class ;
                   rdfs:subClassOf :PizzaTopping ;
                   owl:disjointWith :TomatoTopping ,
                                    :VegetableTopping .
```

Vocabulaires and Ontologies

- People
- Social media
- Commerce
- Events
- Music
- Radio and tv programmes
- Music
- Life Sciences



Standard Vocabulary

A number of standard vocabularies have evolved

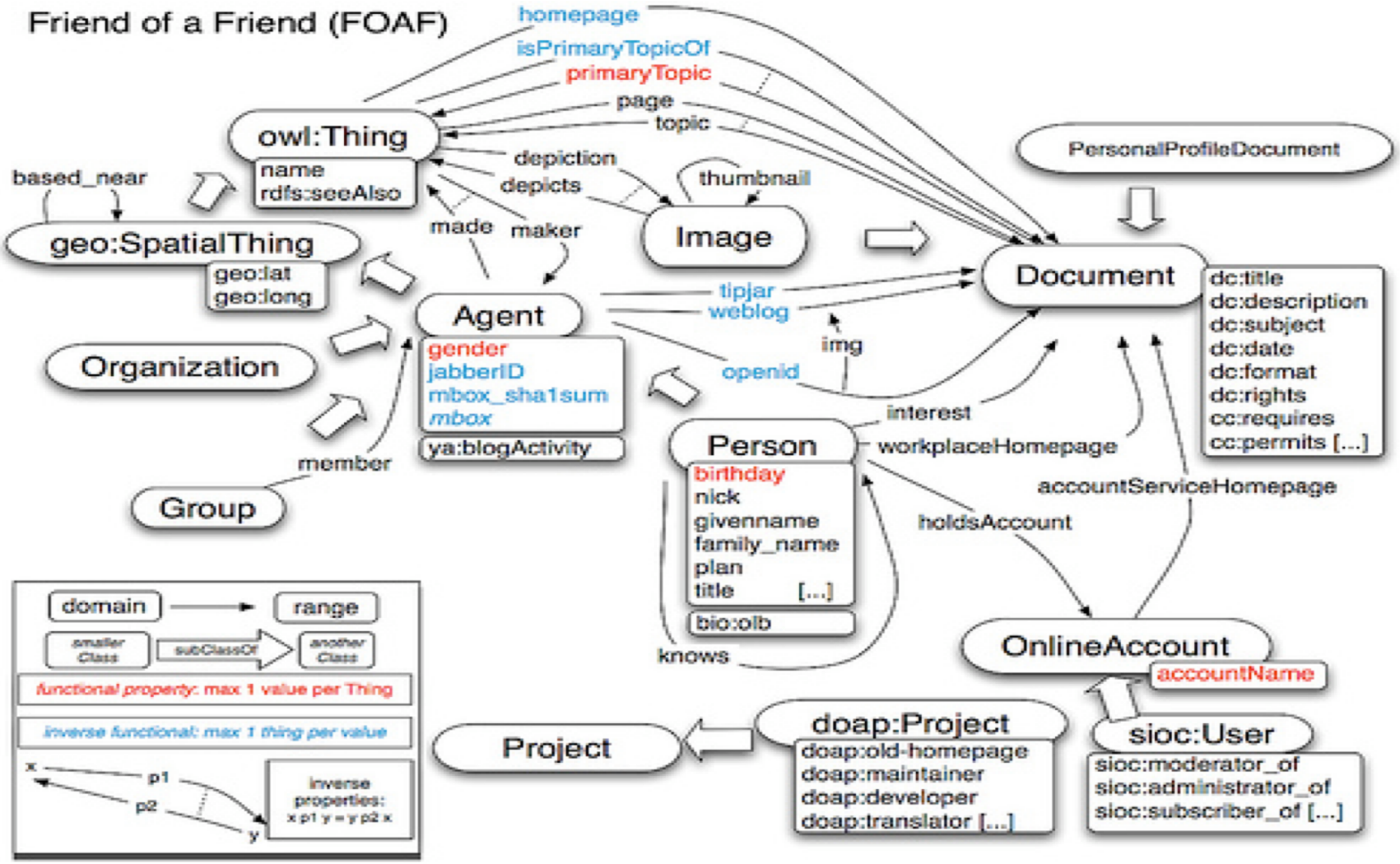
dc: Dublin Core (predicates for describing documents)
http://purl.org/dc/elements/1.1/

foaf: Friend Of A Friend (relationships between people)
http://xmlns.com/foaf/0.1/

cc: Creative Commons (types of licences)
http://creativecommons.org/ns#

Schema.org

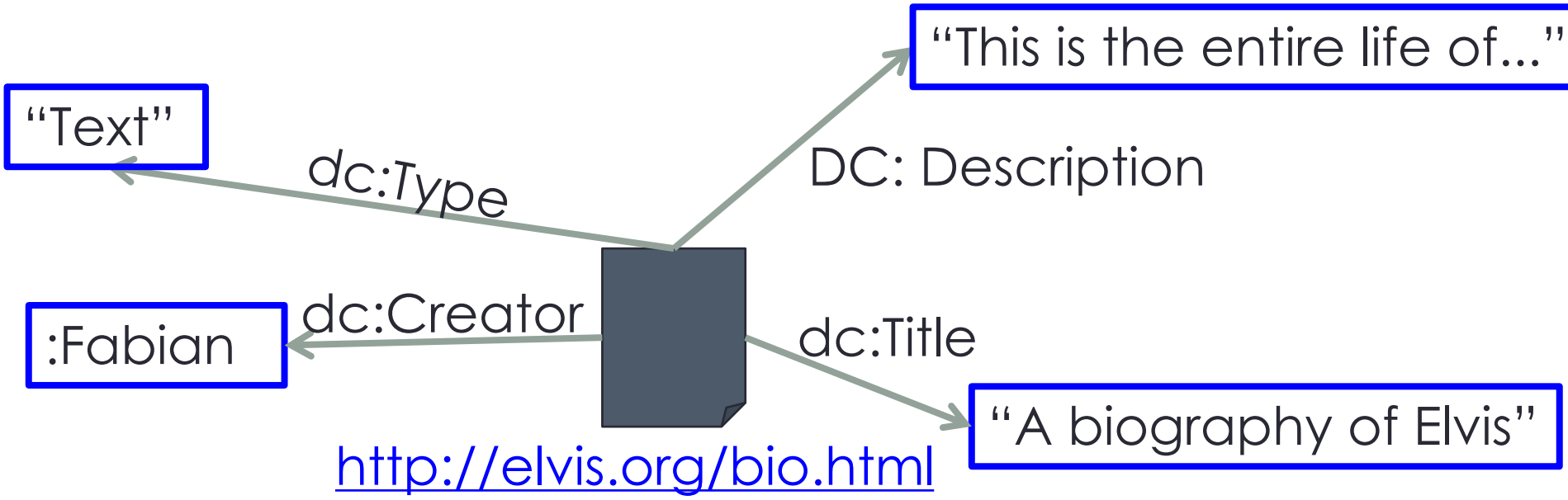
Main classes and properties of FOAF



http://danbri.org/words/2007/11/04/223

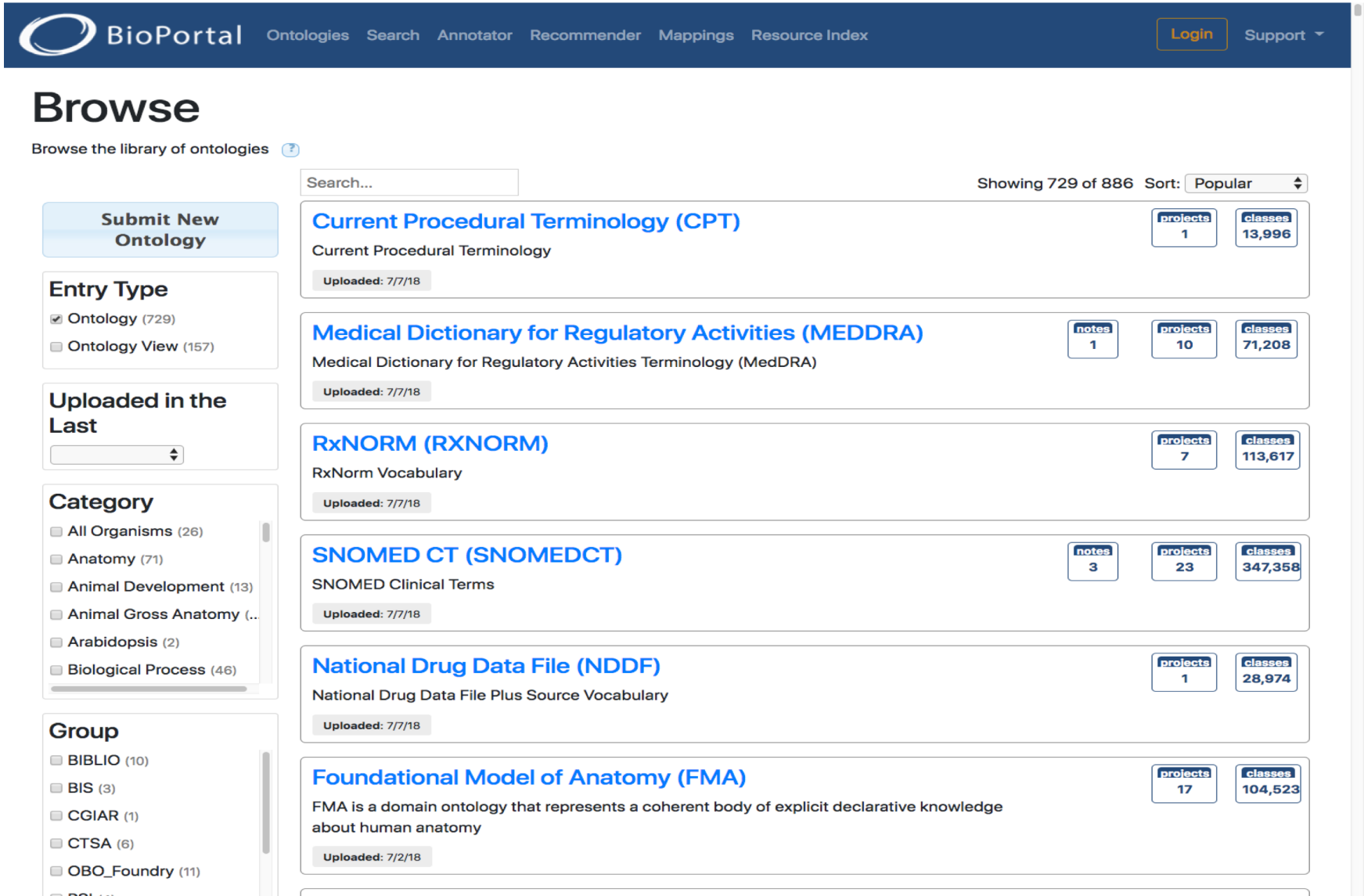
Dublin Core

- dc: Dublin Core (predicates for describing documents)
- http://purl.org/dc/elements/1.1/



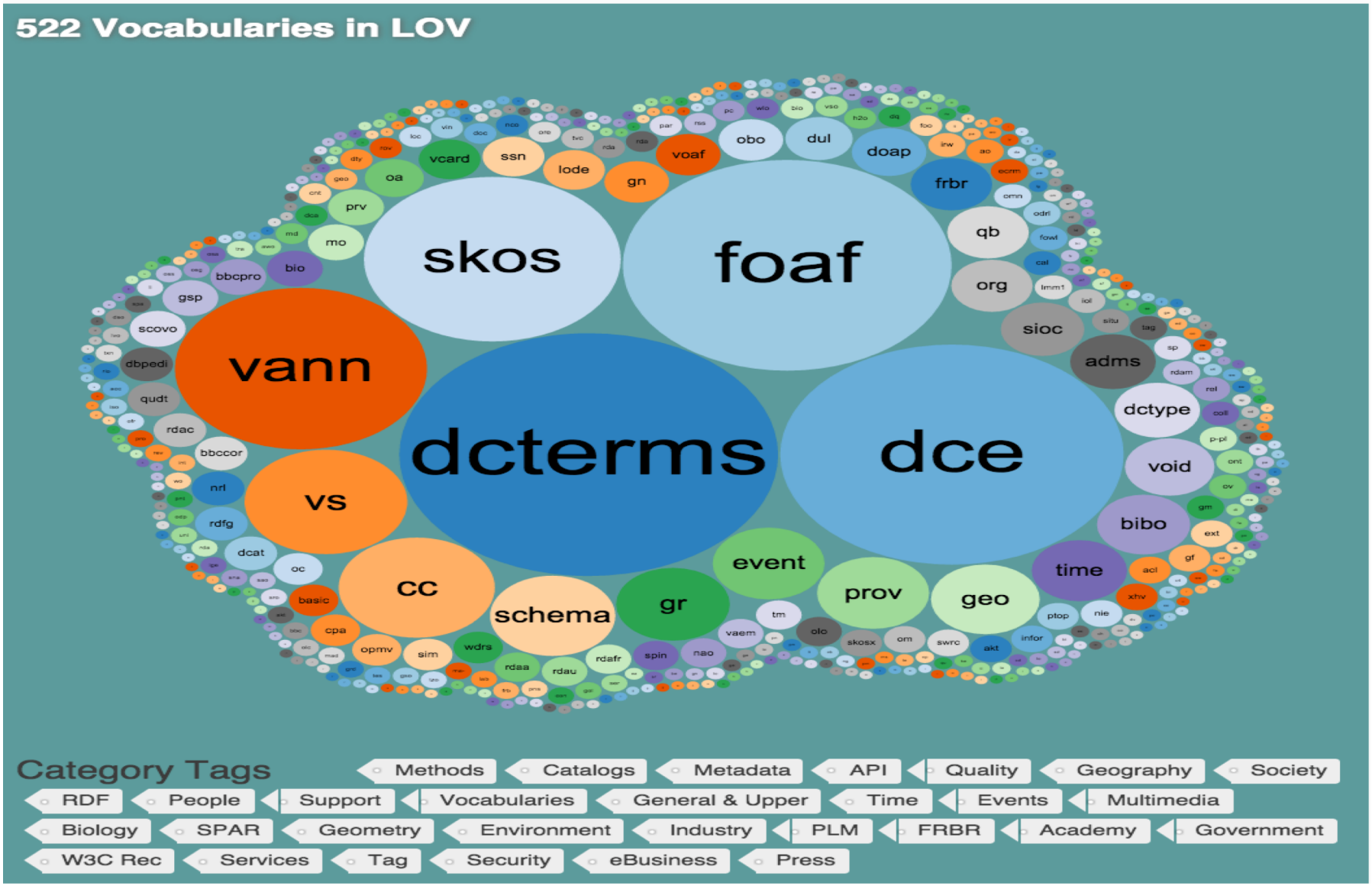
https://bioportal.bioontology.org/

Ontology in Life Sciences



Linked Open Vocabularies (LOV): ontologies used in the LOD

- http://lov.okfn.org/dataset/lov/



person Search

Filter by Domain

- City (346)
- Data & Systems (19)
- General (197)
- Library (324)
- Market (24)
- Media (86)
- Metadata (44)
- Science (51)

Filter by Type

- rdfs:Class (311)
- rdf:Property (928)
- voaf:Vocabulary (23)
- Other (72)

Filter by Vocabulary (128)

- rdarole (120)
- schema (77)
- agrelon (69)

1143 results in 128 vocabularies

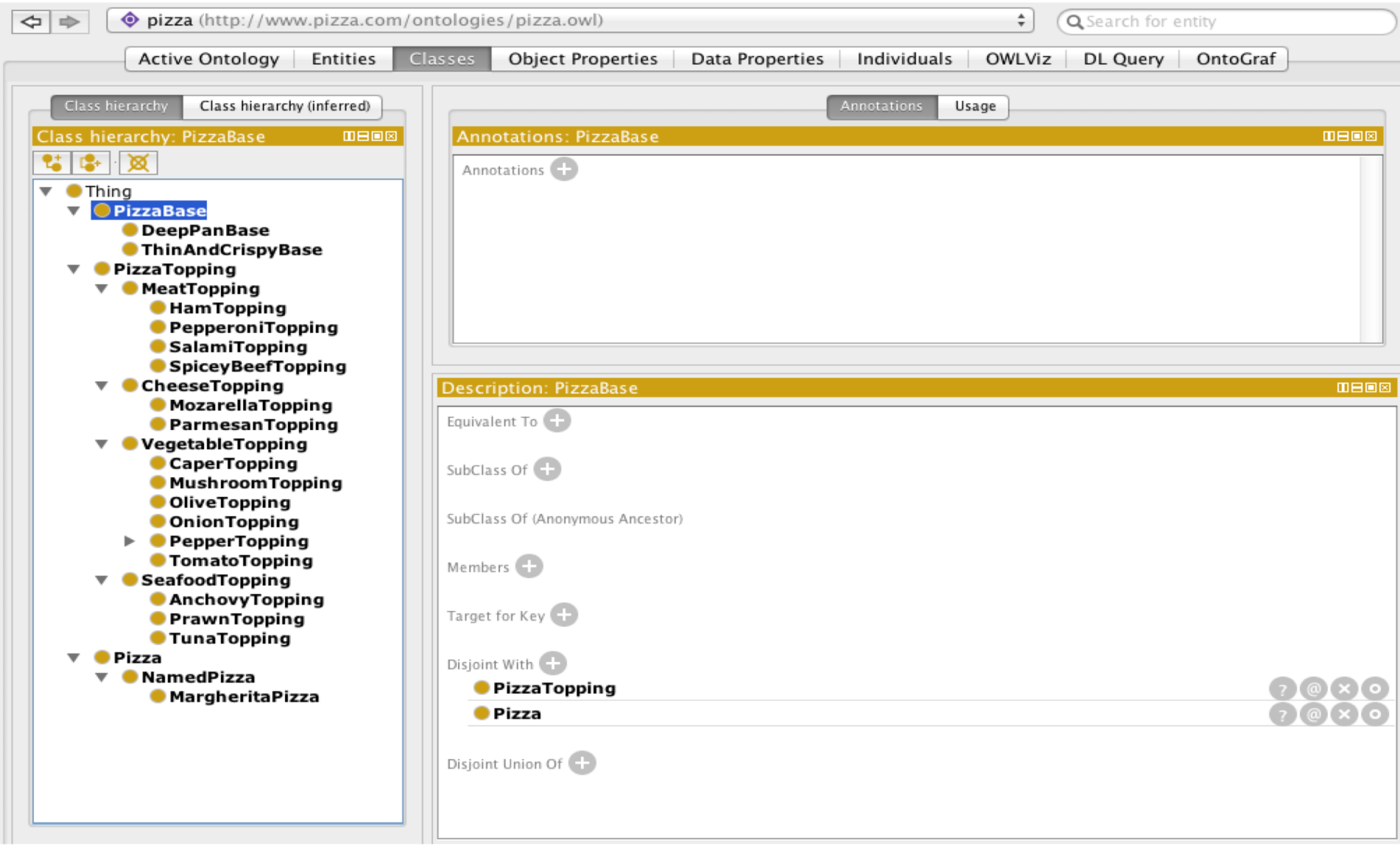
foaf:Person (owl:Class) score:0.689
rdfs:label Personne @fr
rdfs:label Persona @es
rdfs:label Person
rdfs:label Person @en
dce:title Person @en
rdfs:comment Una persona @es
rdfs:comment A person @en
rdfs:comment "A person. The foaf:Person cl.....Something is a foaf:Person if it is a person. ...
rdfs:comment A person.
dce:description "A person. The foaf:Person cl.....Something is a foaf:Person if it is a person. ... @en

crm:E21_Person (rdfs:Class) score:0.591
rdfs:label Personne @fr
rdfs:label Person @de
rdfs:label Person @en
rdfs:comment ...lass comprises real persons who live or are as..... to whether several persons are in fact identi... @en

schema:Person (owl:Class) score:0.548
rdfs:label Person
rdfs:label Person @en
rdfs:comment A person (alive, dead, undea...

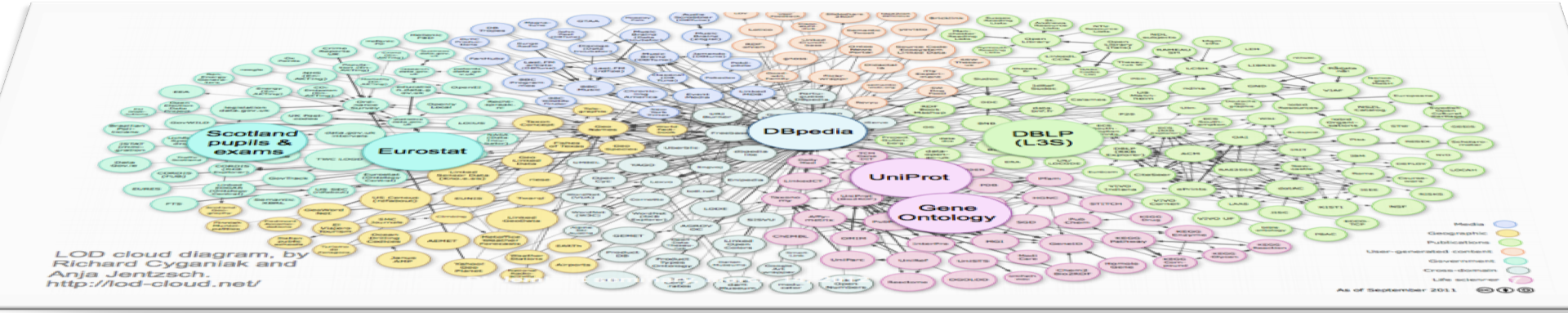
dul:Person (owl:Class) score:0.545
rdfs:label Persona {it}

Ontology construction (https://protege.stanford.edu/)



Linked Open Data

- The web is a giant knowledge graph
  - RDF data model, SPRQL Query language
- Do you want to contribute
  - Transform to RDF
  - Linkd to others
  - Consume other contributions ..
- We did it:
  - http://odpaddle.uni-nantes.fr/lodpaddle



Conclusion

- RDF
- SPARQL
- RDFS
- OWL
- Linked Data
- Web of Data : a giant distributed knowledge graph