RDF, RDFS, and OWL

John Beverley

Assistant Professor, *University at Buffalo*
Co-Director, National Center for Ontological Research
Affiliate Faculty, *Institute of Artificial Intelligence and Data Science*

Outline

- Resource Description Framework
- Resource Description Framework Schema
- Web Ontology Language

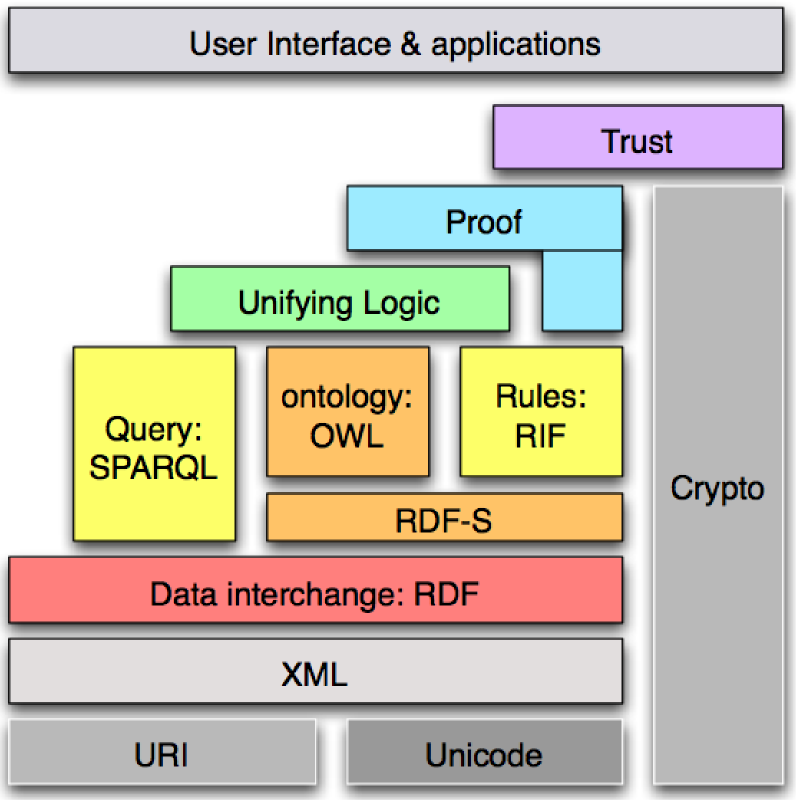
Outline

- Resource Description Framework

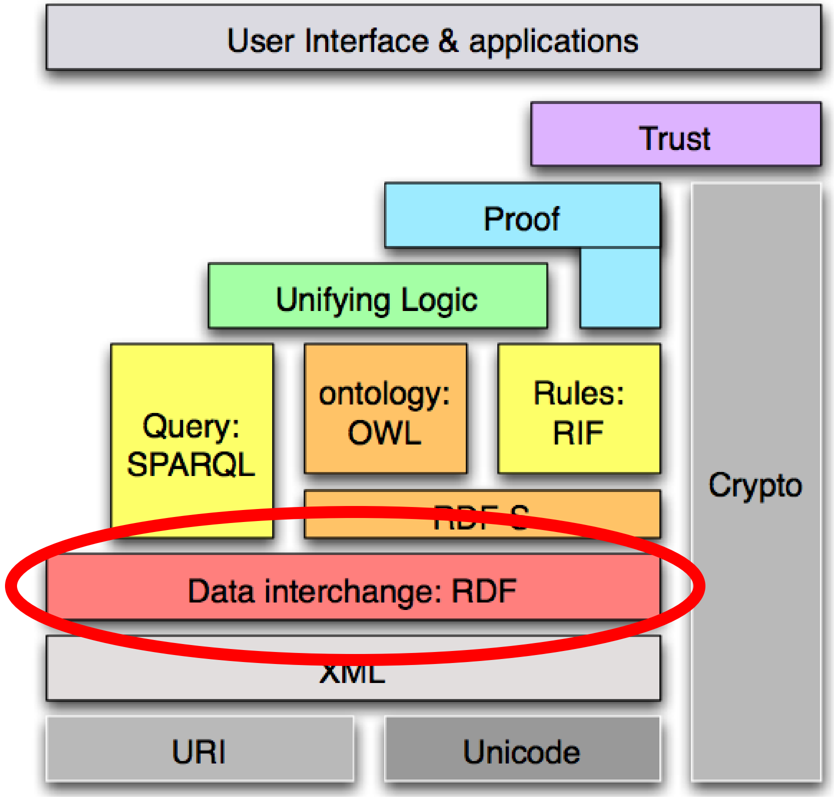
- Resource Description Framework Schema

- Web Ontology Language

Semantic Web Stack

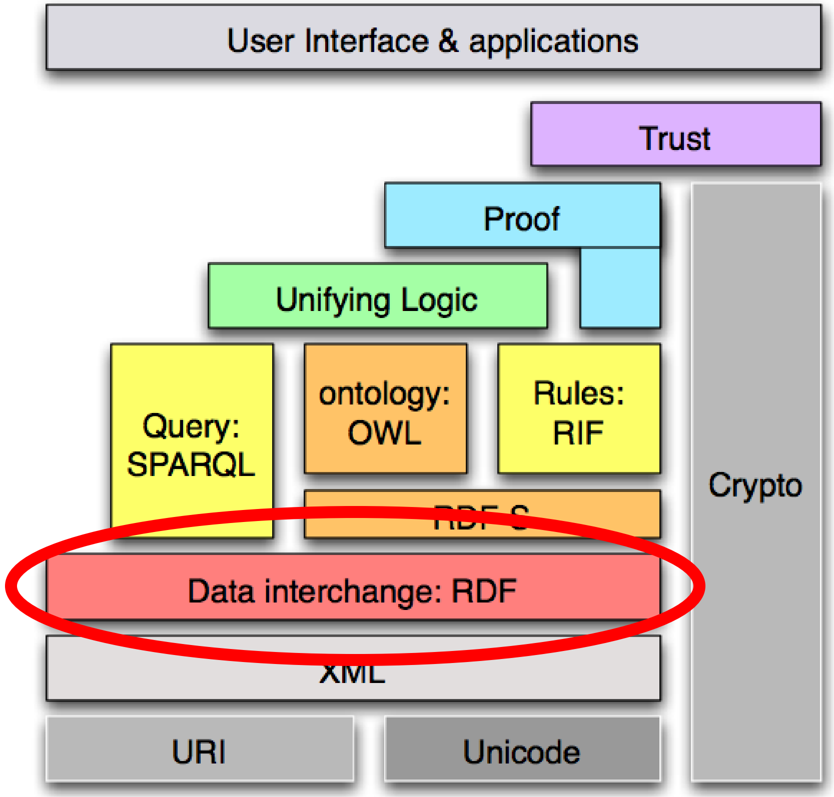


Semantic Web Stack



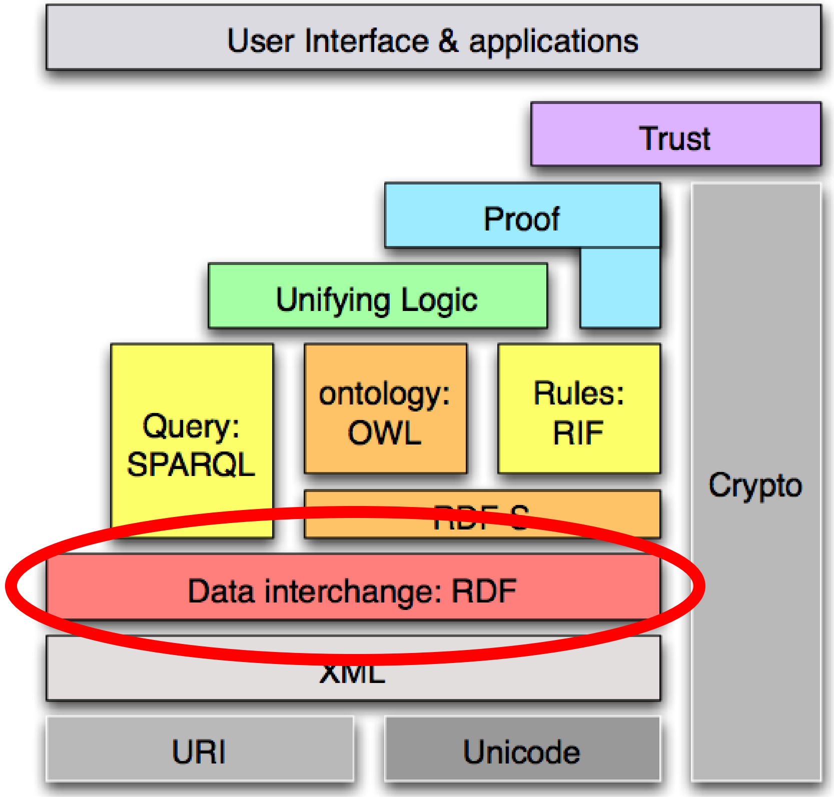
Semantic Web Stack

- "RDF" stands for:
  - **Resource**: Everything that can have a unique identifier, e.g. pages, places, people, dogs, products...
  - **Description**: attributes, features, and relations of among resources
  - **Framework**: model, languages and syntaxes for these descriptions



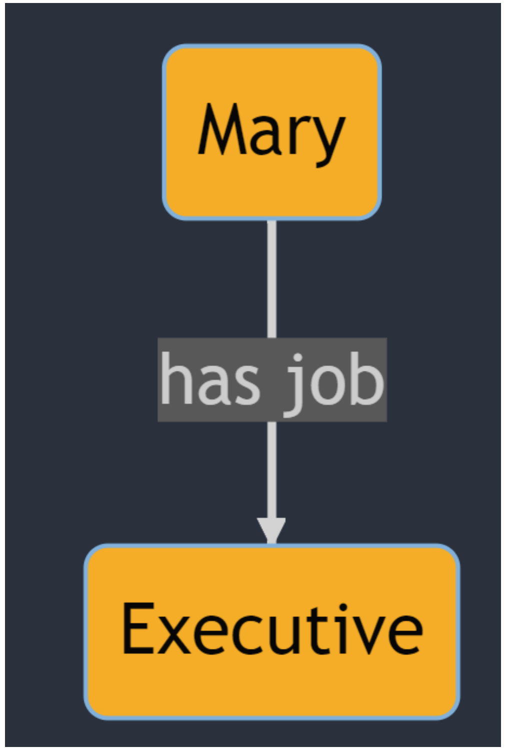
Semantic Web Stack

- "RDF" stands for:
  - **Resource**: Everything that can have a unique identifier, e.g. pages, places, people, dogs, products...
  - **Description**: attributes, features, and relations of among resources
  - **Framework**: model, languages and syntaxes for these descriptions

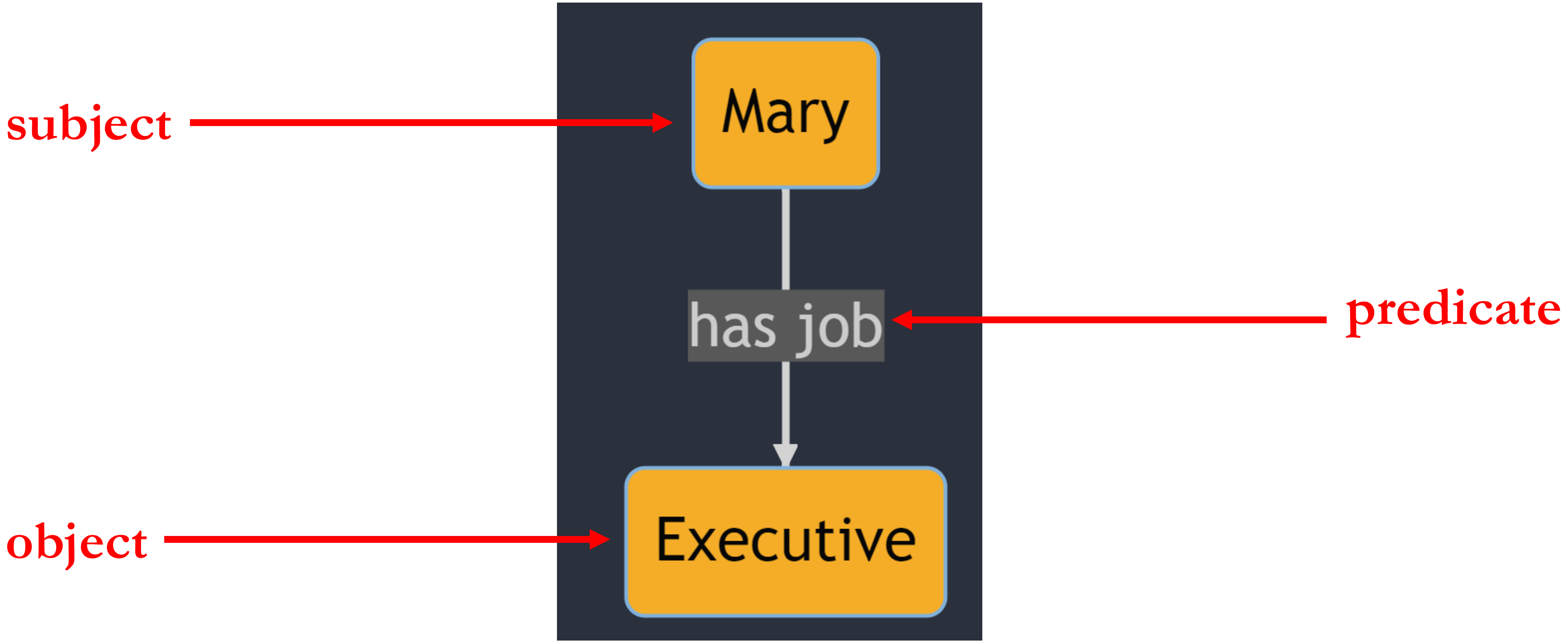


- RDF is:
  - A *data model*
  - That is based on *triples*
  - Which provide semantics for *distributed web data*

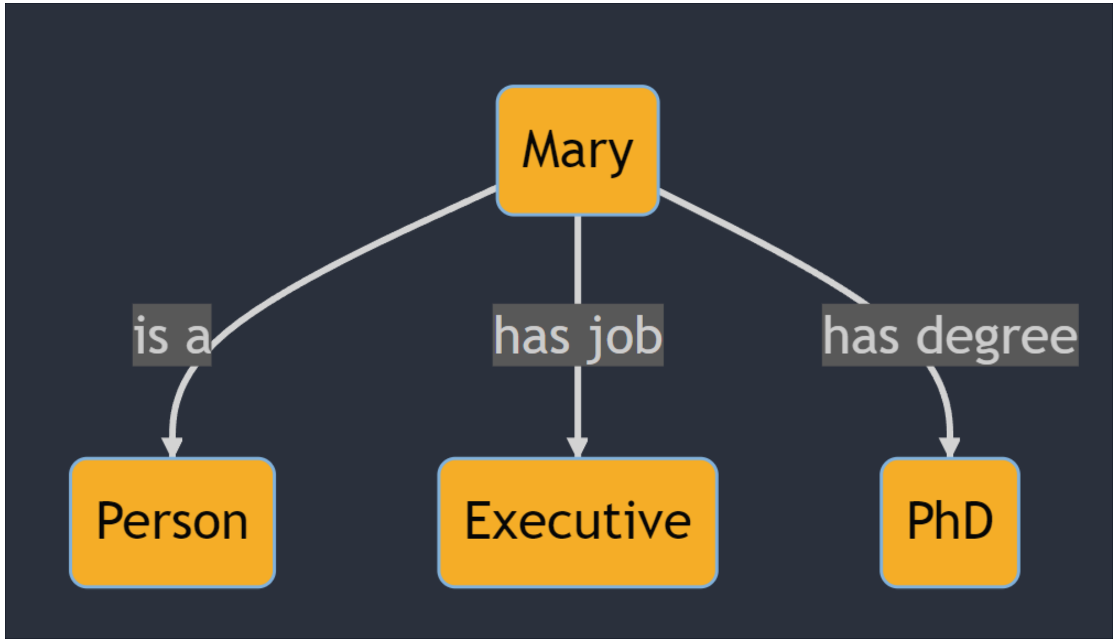
Triple Example



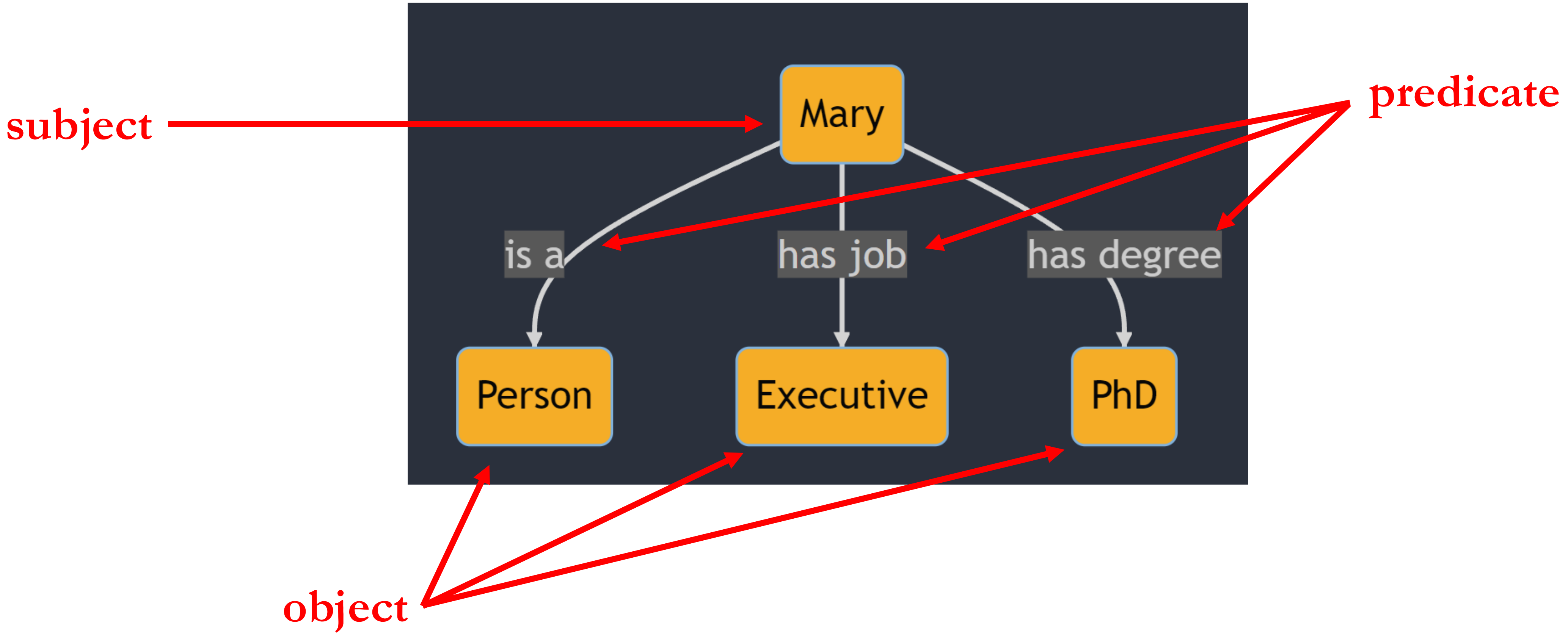
Triple Example



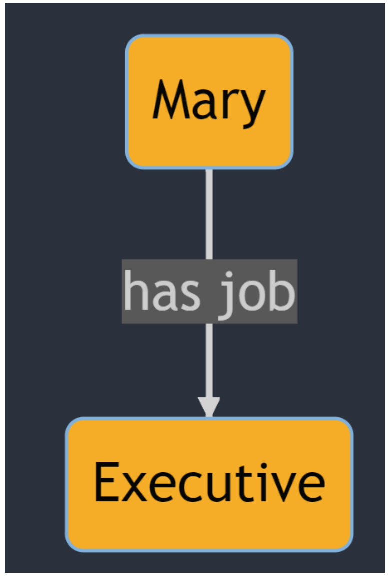
Triple Example



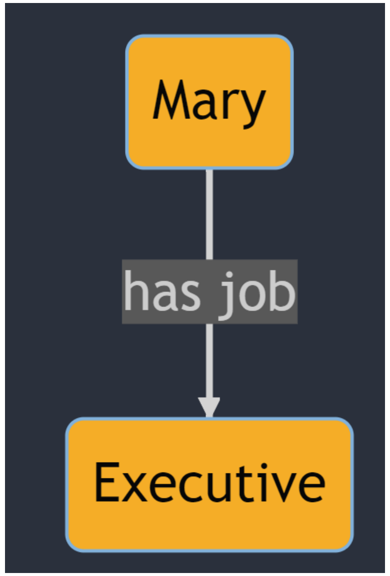
Triple Example



RDF Triple Structure



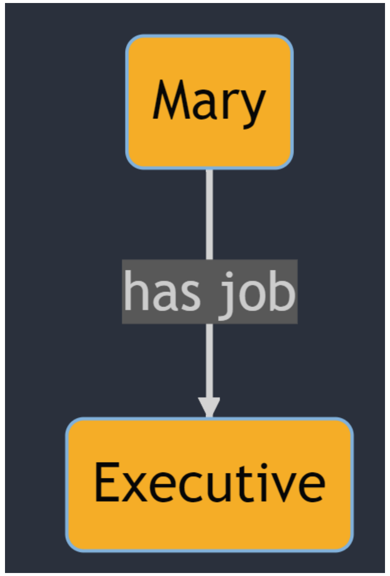
RDF Triple Structure



<https://www.example.com/occupation/Mary> <https://www.example.com/occupation/has_job> <https://www.example.com/occupation/Executive>

Subject Predicate Object

RDF Triple Structure



Uniform Resource Identifier (URI)

<https://www.example.com/occupation/Mary> <https://www.example.com/occupation/has_job> <https://www.example.com/occupation/Executive>

Subject

Predicate

Object

RDF Serializations

- Some RDF **serializations** are easier for humans to read than others, but many have been developed owing to many translational needs of users

- Common serializations you will encounter include:
  - RDF/XML
  - JSON-LD
  - Turtle
  - N-Triples

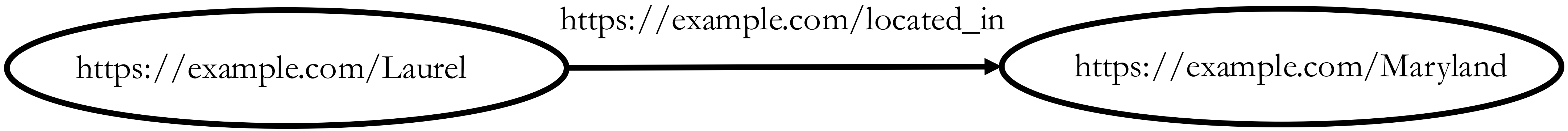
RDF Serializations

- Some RDF **serializations** are easier for humans to read than others, but many have been developed owing to many translational needs of users

- Common serializations you will encounter include:
  - RDF/XML
  - JSON-LD
  - Turtle
  - N-Triples

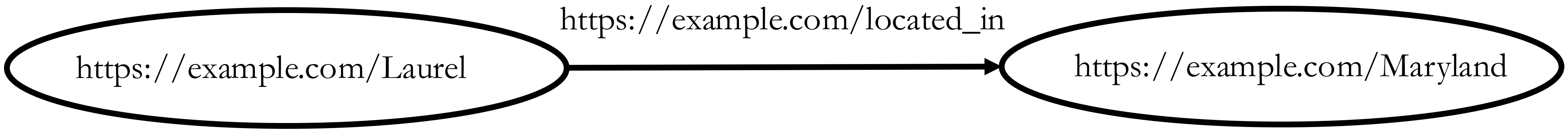
RDF/XML

- RDF/XML **serializations** express RDF as Extensible Markup Language (XML) documents, which is used widely across the web



RDF/XML

- RDF/XML **serializations** express RDF as Extensible Markup Language (XML) documents, which is used widely across the web




```
<?xml version="1.0"?>
<rdf:RDF   xmlsns:ex="https://example.com/"
           xmlsns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
<rdf:Description rdf:about="https://example.com/Laurel">
     <rdfs:label>Laurel</rdfs:label>
     <ex:located_in>Maryland</ex:located_in>
</rdf:Description>
</rdf:RDF>
```

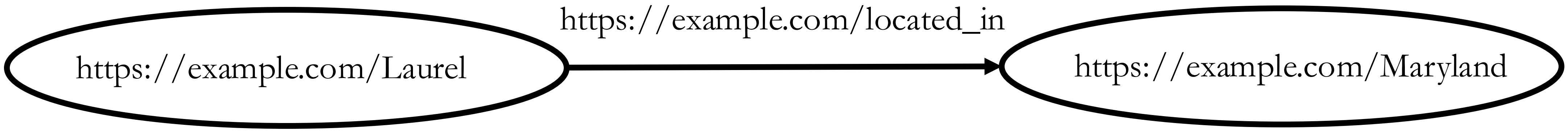
RDF Serializations

- Some RDF **serializations** are easier for humans to read than others, but many have been developed owing to many translational needs of users

- Common serializations you will encounter include:
  - RDF/XML
  - JSON-LD
  - Turtle
  - N-Triples

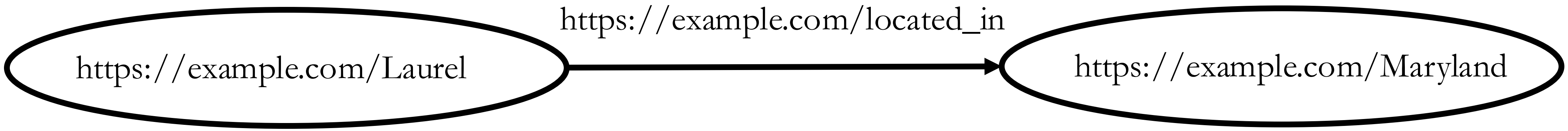
JSON-LD

- JSON-LD is a recently developed serialization, but is now the preferred way to structure data in the Google knowledge graph, so...



JSON-LD

- JSON-LD is a recently developed serialization, but is now the preferred way to structure data in the Google knowledge graph, so...



[{"@id":"https://example.com/Laurel",

https://example.com/located_in

:[{"@id":"https://example.com/Maryland"}],

http://www.w3.org/2000/01/rdf-schema#label
:[{"@value":"Laurel"}]},{"@id":"https://example.com/Maryland"}]

RDF Serializations

- Some RDF **serializations** are easier for humans to read than others, but many have been developed owing to many translational needs of users

- Common serializations you will encounter include:
  - RDF/XML
  - JSON-LD
  - Turtle
  - N-Triples

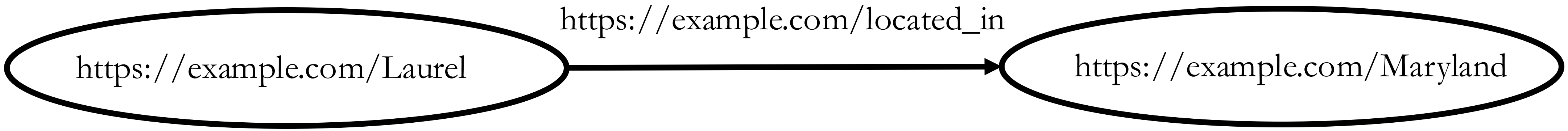
RDF Serializations

- Some RDF **serializations** are easier for humans to read than others, but many have been developed owing to many translational needs of users

- Common serializations you will encounter include:
  - RDF/XML
  - JSON-LD
  - Turtle
  - N-Triples

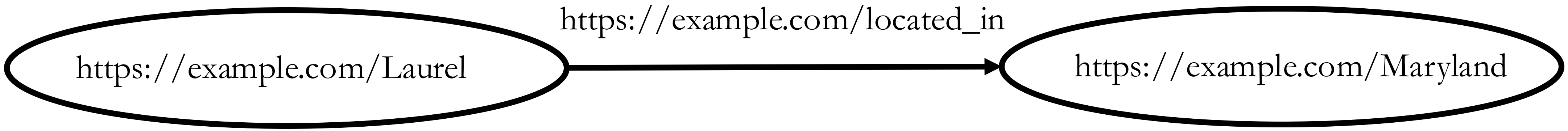
Turtle

- The Terse RDF Triple Language (Turtle) is a compact serialization of RDF that is, in my opinion, much easier to read



Turtle

- The Terse RDF Triple Language (Turtle) is a compact serialization of RDF that is, in my opinion, much easier to read




```
@prefix ex: <https://example.com/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

ex:Laurel rdfs:label “Laurel” ;
            ex:located_in ex:Maryland .
```

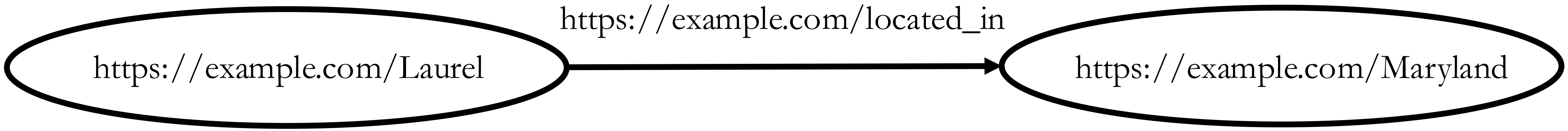
RDF Serializations

- Some RDF **serializations** are easier for humans to read than others, but many have been developed owing to many translational needs of users

- Common serializations you will encounter include:
  - RDF/XML
  - JSON-LD
  - Turtle
  - N-Triples

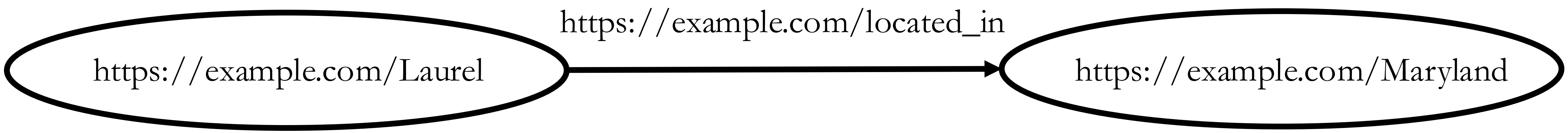
N-Triples

- The N-Triple serialization standard represents triples as unabbreviated URIs enclosed in angle brackets and strings in quotes



N-Triples

- The N-Triple serialization standard represents triples as unabbreviated URIs enclosed in angle brackets and strings in quotes



- What you see in a text file:

<https://example.com/Laurel> <https://example.com/located_in> <https://example.com/Maryland> .

<https://example.com/Laurel> <http://www.w3.org/2000/01/rdf-schema#label> “Laurel” .

Rules of RDF

- Every fact must be **expressed as a triple**
- Subjects, predicates, and objects **are names** for entities represented as URIs
- **Objects** may be literal values, but **subjects and predicates cannot**

RDF Vocabulary

- RDF is a very simple data model, with a very simple vocabulary:

rdf:Property
rdf:type
rdf:Statement
rdf:subject
rdf:predicate
rdf:object

rdf:Bag
rdf:List
rdf:first
rdf:rest
rdf:nil

RDF Vocabulary

- RDF is a very simple data model, with a very simple vocabulary:

rdf:Property	rdf:Bag
rdf:type	rdf:List
rdf:Statement	rdf:first
rdf:subject	rdf:rest
rdf:predicate	rdf:nil
rdf:object	

The class of rdf properties, i.e. relationships among rdf resources

RDF Vocabulary

- RDF is a very simple data model, with a very simple vocabulary:

rdf:Property
rdf:type
rdf:Statement
rdf:subject
rdf:predicate
rdf:object

rdf:Bag
rdf:List
rdf:first
rdf:rest
rdf:nil

Instance of rdf:Property used to state that a resource is an instance of another resource

RDF Vocabulary

- RDF is a very simple data model, with a very simple vocabulary:

rdf:Property
rdf:type
rdf:Statement
rdf:subject
rdf:predicate
rdf:object

rdf:Bag
rdf:List
rdf:first
rdf:rest
rdf:nil

The class of statements made about rdf triples

RDF Vocabulary

- RDF is a very simple data model, with a very simple vocabulary:

rdf:Property	rdf:Bag
rdf:type	rdf:List
rdf:Statement	rdf:first
rdf:subject	rdf:rest
rdf:predicate	rdf:nil
rdf:object	

Used often for *reification*

Reification

- To reify in RDF is to make a statement about a statement

- For example, Sam might introduce a triple to an ontology:

  ex:butter_potato_1 rdf:type ex:Potato

Reification

- To reify in RDF is to make a statement about a statement

- For example, Sam might introduce a triple to an ontology:

  ex:butter_potato_1 rdf:type ex:Potato

- But wants to make sure this is because Gordan Ramsey says so...

- Sam wants to represent:

  ex:Ramsey ex:believes {ex:butter_potato_1 rdf:type ex:Potato}

Reification

- To reify in RDF is to make a statement about a statement

  ex:Ramsey ex:believes {ex:butter_potato_1 rdf:type ex:Potato}

- Represented in RDF as:

```
ex:statement_1 [ rdf:type rdf:Statement;
                 rdf:subject ex:butter_potato_1 ;
                 rdf:predicate rdf:type ;
                 rdf:object ex:Potato ] .
```

Reification

- To reify in RDF is to make a statement about a statement

    ex:Ramsey ex:believes {ex:butter_potato_1 rdf:type ex:Potato}

- Represented in RDF as:

```
ex:statement_1 [ rdf:type rdf:Statement;
                 rdf:subject ex:butter_potato_1 ;
                 rdf:predicate rdf:type ;
                 rdf:object ex:Potato ] .
```

Reification

- To reify in RDF is to make a statement about a statement

```
ex:Ramsey ex:believes {ex:butter_potato_1 rdf:type ex:Potato}
```

- Represented in RDF as:

```
ex:statement_1 [ rdf:type rdf:Statement;
                 rdf:subject ex:butter_potato_1 ;
                 rdf:predicate rdf:type ;
                 rdf:object ex:Potato ] .


ex:Ramsey ex:believes ex:statement_1
```

Reification

- To reify in RDF is to make a statement about a statement

  ex:Ramsey ex:believes {ex:butter_potato_1 rdf:type ex:Potato}

- Represented in RDF as:

```
ex:statement_1 [ rdf:type rdf:Statement;
                 rdf:subject ex:butter_potato_1 ;
                 rdf:predicate rdf:type ;
                 rdf:object ex:Potato ] .
```

ex:Ramsey ex:believes ex:statement_1

RDF Vocabulary

- RDF is a very simple data model, with a very simple vocabulary:

rdf:Property	rdf:Bag
rdf:type	rdf:List
rdf:Statement	rdf:first
rdf:subject	rdf:rest
rdf:predicate	rdf:nil
rdf:object	

rdf resource used to conventionally indicate to readers that collected members are unordered, e.g. John teaches group of students

RDF Vocabulary

- RDF is a very simple data model, with a very simple vocabulary:

rdf:Property	rdf:Bag
rdf:type	rdf:List
rdf:Statement	rdf:first
rdf:subject	rdf:rest
rdf:predicate	rdf:nil
rdf:object	

class of rdf lists

RDF Vocabulary

- RDF is a very simple data model, with a very simple vocabulary:

rdf:Property	rdf:Bag
rdf:type	rdf:List
rdf:Statement	rdf:first
rdf:subject	rdf:rest
rdf:predicate	rdf:nil
rdf:object	

Distinguishes first element of list from rest, and from empty list, e.g.

RDF Vocabulary

- RDF is a very simple data model, with a very simple vocabulary:

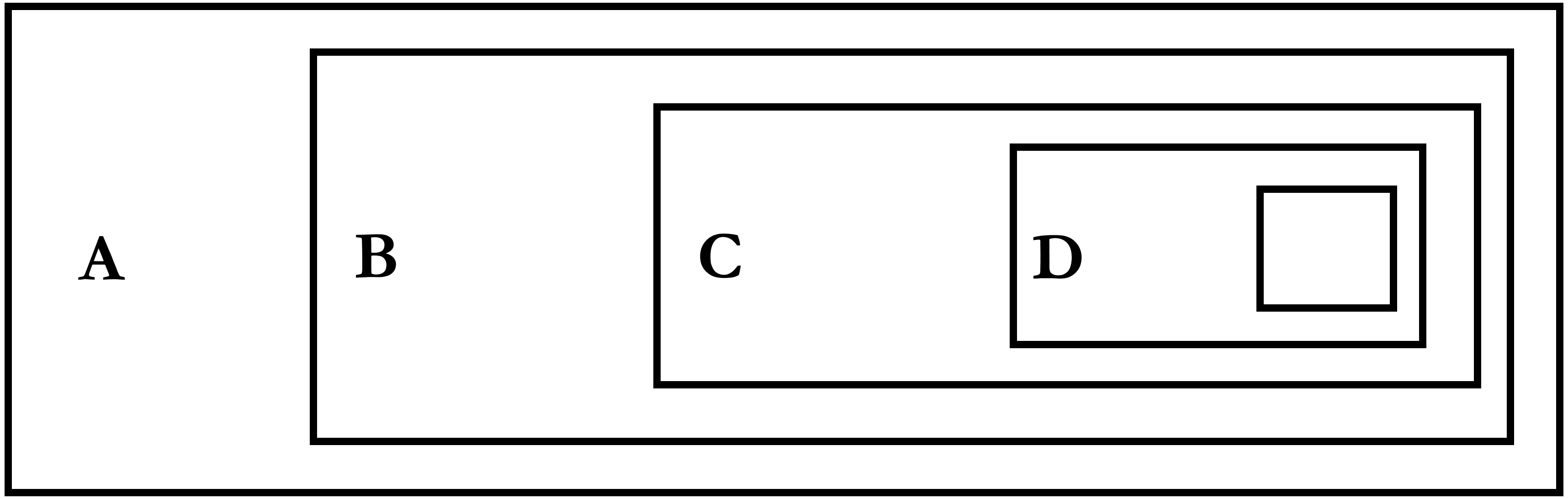
rdf:Property	rdf:Bag
rdf:type	rdf:List
rdf:Statement	rdf:first
rdf:subject	rdf:rest
rdf:predicate	rdf:nil
rdf:object	

Distinguishes first element of list from rest, and from empty list

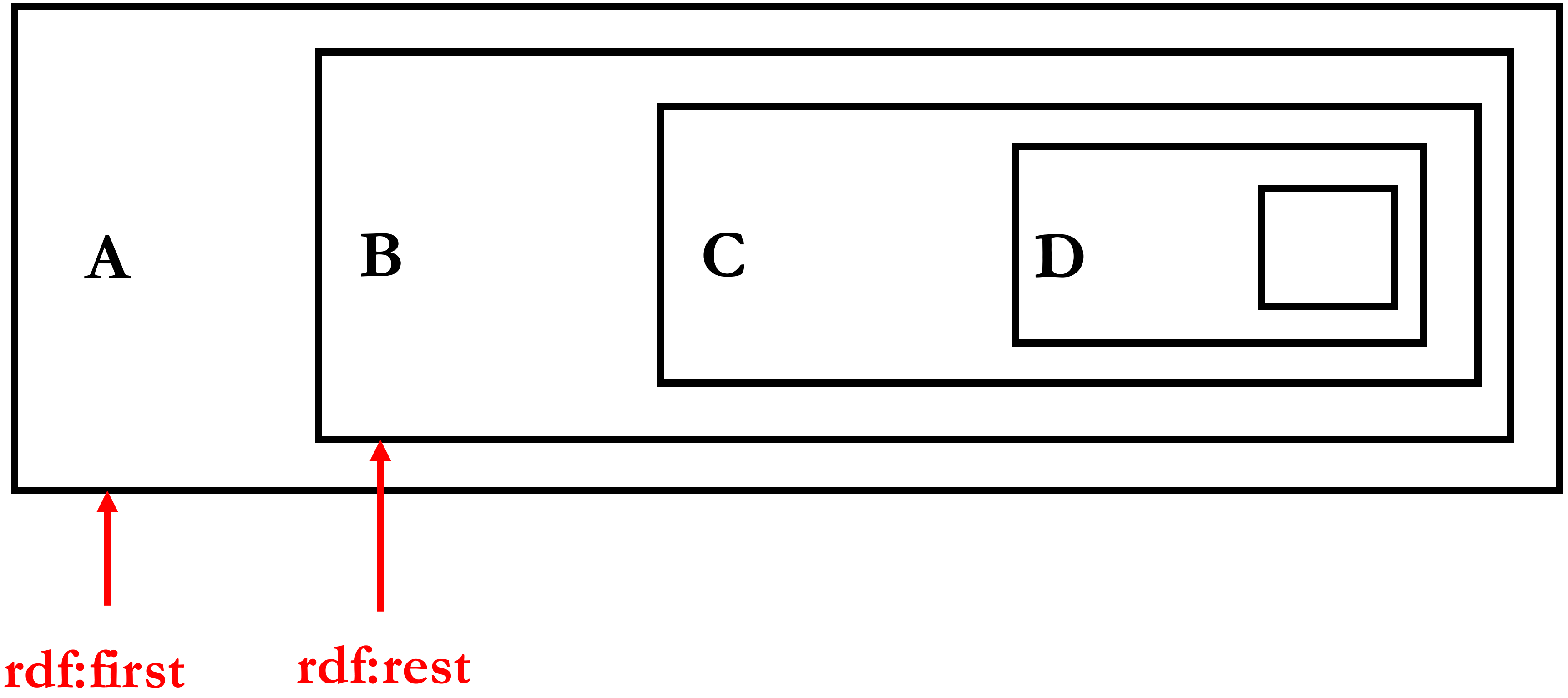
List = A,B,C,D

A B C D

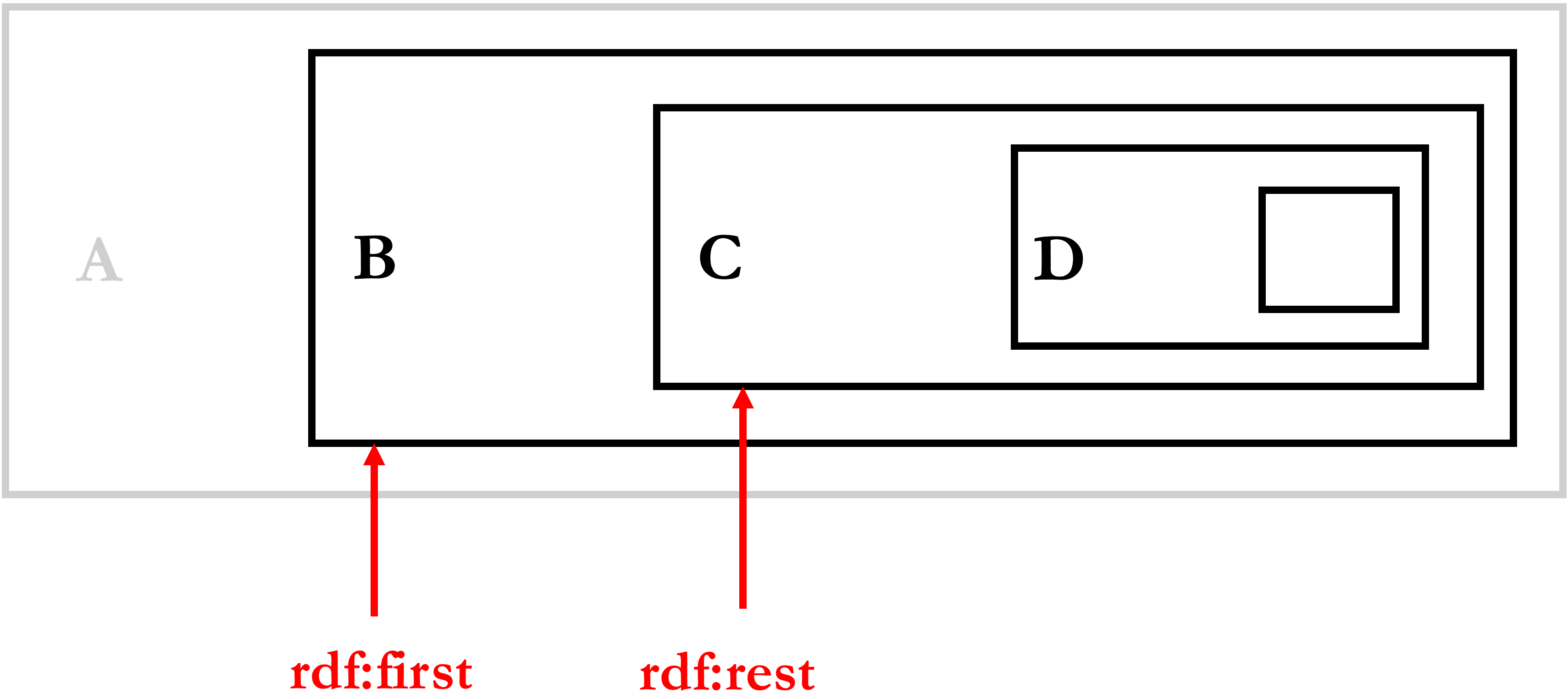
List = A,B,C,D



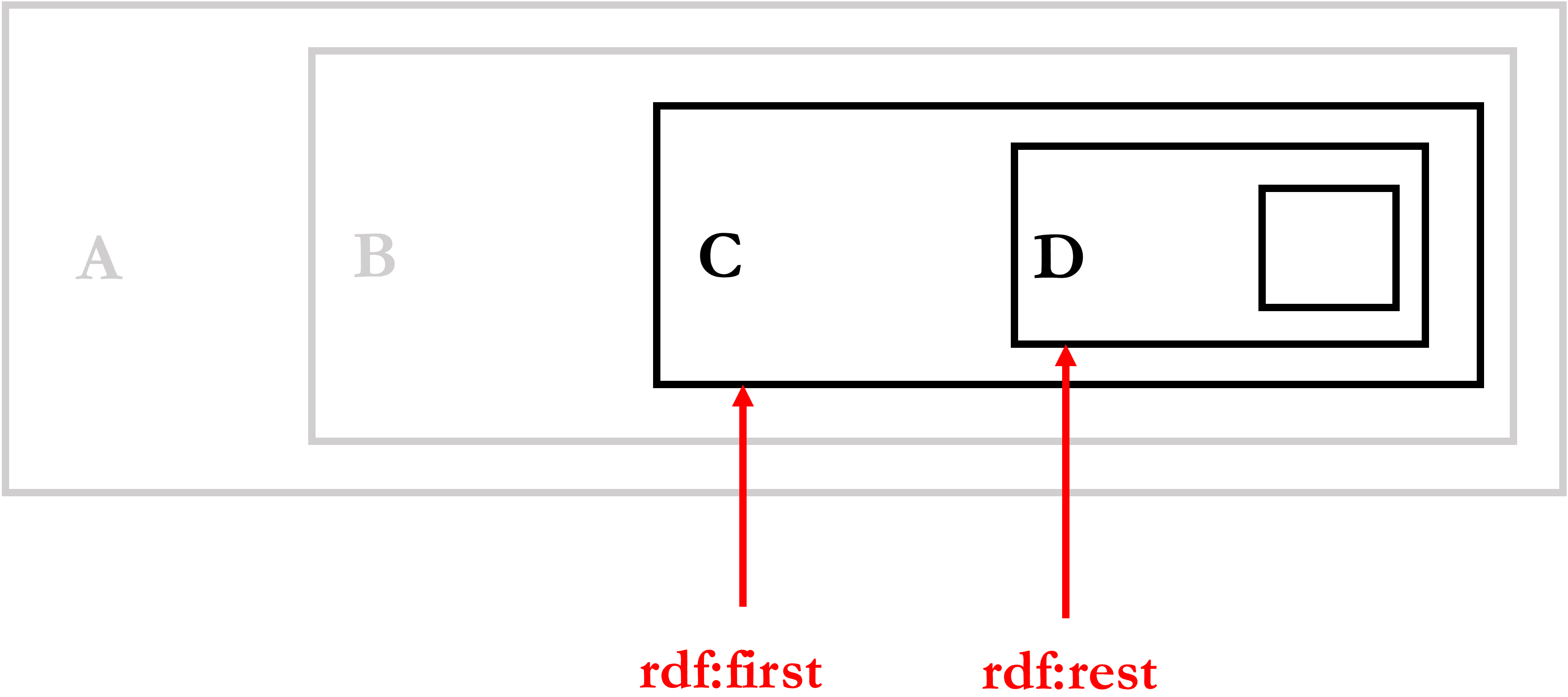
List = A,B,C,D



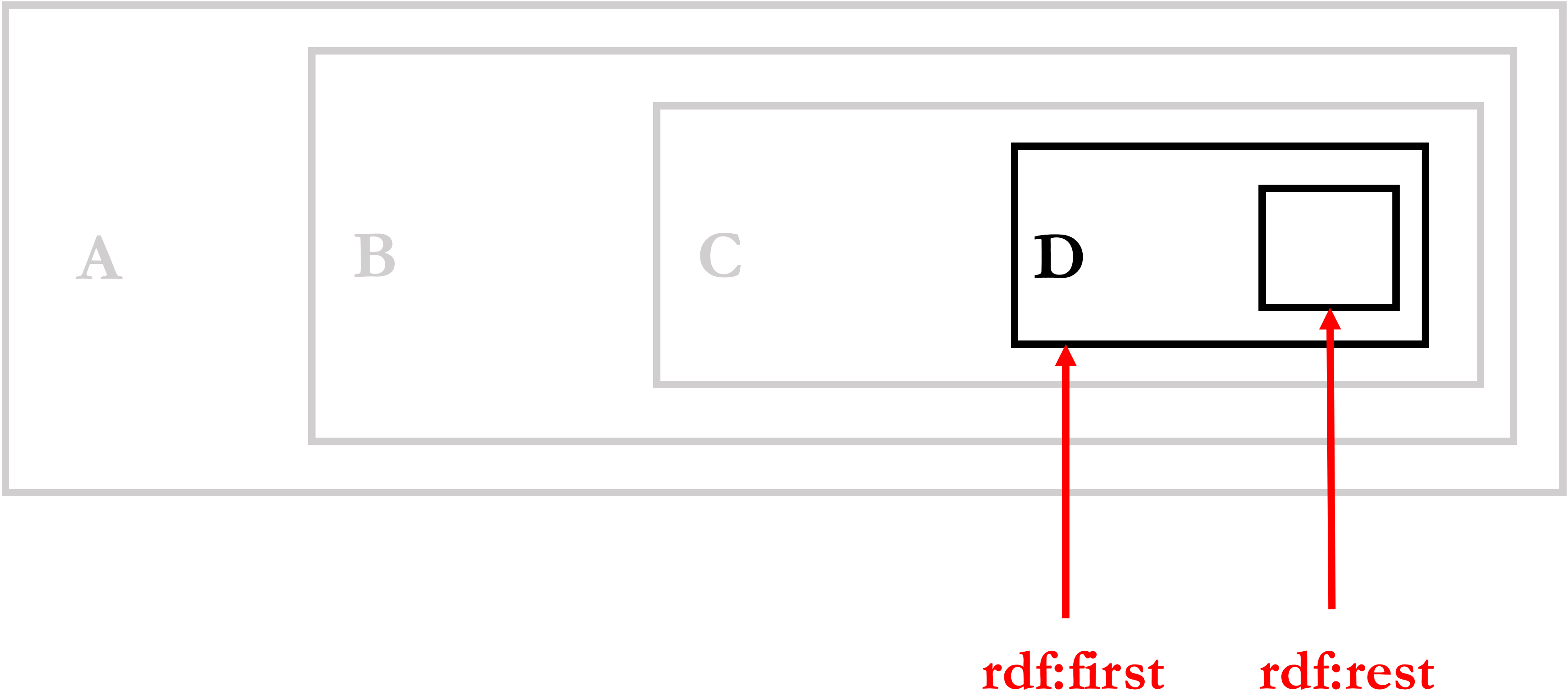
List = A,B,C,D



List = A,B,C,D



List = A,B,C,D



List = A,B,C,D

List Example

- New England Patriots are a better football team than the Premier League football team Arsenal, which is better than the Atlanta Braves…

- The ordering involved can be represented in rdf as:

List Example

- New England Patriots are a better football team than the Premier League football team Arsenal, which is better than the Atlanta Braves...

- The ordering involved can be represented in rdf as:

```
ex:list ex:football_rank [ rdf:first ex:Patriots ;
                           rdf:rest [ rdf:first ex:Arsenal ;
                                      rdf:rest [ rdf:first ex:Braves ;
                                                 rdf:rest rdf:nil ]]].
```

Subject is a list

List Example

- New England Patriots are a better football team than the Premier League football team Arsenal, which is better than the Atlanta Braves…

- The ordering involved can be represented in rdf as:

```
ex:list ex:football_rank [ rdf:first ex:Patriots ;
                           rdf:rest [ rdf:first ex:Arsenal ;
                                      rdf:rest [ rdf:first ex:Braves ;
                                                 rdf:rest rdf:nil ]]].
```

Predicate implies ordering

List Example

- New England Patriots are a better football team than the Premier League football team Arsenal, which is better than the Atlanta Braves...

- The ordering involved can be represented in rdf as:

```
ex:list ex:football_rank [ rdf:first ex:Patriots ;
                           rdf:rest [ rdf:first ex:Arsenal ;
                                      rdf:rest [ rdf:first ex:Braves ;
                                                 rdf:rest rdf:nil ]]].
```

First element of object list is Patriots

List Example

- New England Patriots are a better football team than the Premier League football team Arsenal, which is better than the Atlanta Braves…

- The ordering involved can be represented in rdf as:

```
ex:list ex:football_rank [ rdf:first ex:Patriots ;
                           rdf:rest [ rdf:first ex:Arsenal ;
                                      rdf:rest [ rdf:first ex:Braves ;
                                                 rdf:rest rdf:nil ]]].
```

Remainder of list is second

List Example

- New England Patriots are a better football team than the Premier League football team Arsenal, which is better than the Atlanta Braves...

- The ordering involved can be represented in rdf as:

```
ex:list ex:football_rank [ rdf:first ex:Patriots ;
                           rdf:rest [ rdf:first ex:Arsenal ;
                                      rdf:rest [ rdf:first ex:Braves ;
                                                 rdf:rest rdf:nil ]]].
```

First element of the second list is Arsenal

List Example

- New England Patriots are a better football team than the Premier League football team Arsenal, which is better than the Atlanta Braves…

- The ordering involved can be represented in rdf as:

```
ex:list ex:football_rank [ rdf:first ex:Patriots ;
                           rdf:rest [ rdf:first ex:Arsenal ;
                                      rdf:rest [ rdf:first ex:Braves ;
                                                 rdf:rest rdf:nil ]]].
```

Remainder of list is third

List Example

- New England Patriots are a better football team than the Premier League football team Arsenal, which is better than the Atlanta Braves...

- The ordering involved can be represented in rdf as:

```
ex:list ex:football_rank [ rdf:first ex:Patriots ;
                           rdf:rest [ rdf:first ex:Arsenal ;
                                      rdf:rest [ rdf:first ex:Braves ;
                                                 rdf:rest rdf:nil ]]].
```

First element of this list is Braves

List Example

- New England Patriots are a better football team than the Premier League football team Arsenal, which is better than the Atlanta Braves...

- The ordering involved can be represented in rdf as:

```
ex:list ex:football_rank [ rdf:first ex:Patriots ;
                           rdf:rest [ rdf:first ex:Arsenal ;
                                      rdf:rest [ rdf:first ex:Braves ;
                                                 rdf:rest rdf:nil ]]].
```

Remainder of list is fourth →

List Example

- New England Patriots are a better football team than the Premier League football team Arsenal, which is better than the Atlanta Braves…

- The ordering involved can be represented in rdf as:

```
ex:list ex:football_rank [ rdf:first ex:Patriots ;
                           rdf:rest [ rdf:first ex:Arsenal ;
                                      rdf:rest [ rdf:first ex:Braves ;
                                                 rdf:rest rdf:nil ]]].
```

Only element of fourth list is empty

RDF Vocabulary

- RDF is a very simple data model, with a very simple vocabulary:

rdf:Property
rdf:type
rdf:Statement
rdf:subject
rdf:predicate
rdf:object

rdf:Bag
rdf:List
rdf:first
rdf:rest
rdf:nil

There are more terms in the RDF vocabulary, some of which can be leveraged for rather expressive representations; those mentioned here are what you'll most often encounted though

RDF Vocabulary

- RDF is a very simple data model, with a very simple vocabulary:

rdf:Property	rdf:Bag
rdf:type	rdf:List
rdf:Statement	rdf:first
rdf:subject	rdf:rest
rdf:predicate	rdf:nil
rdf:object	

But before turning to extensions of RDF, one further topic deserves our attention…

Blank Nodes

- Thus far, we've seen how RDF allows one to identify resources across the Web with unique URIs...

- But RDF also allows for the use of resources that *don't have any specific unique identifier*

- This is useful because there are many cases in which one wants to represent that, say, some x is related to a specific entity, without being able to identify that x

Blank Nodes

- Thus far, we've seen how RDF allows one to identify resources across the Web with unique URIs...

- But RDF also allows for the use of resources that *don't have any specific unique identifier*

That is, sometimes we want to represent *existentially quantified* relata in our ontologies

Blank Nodes Example

- Suppose you know that Barry plays the violin and you'd like to represent this with an object property...but don't know which violin he plays...

ex:Barry ex:plays **???** .

Blank Nodes Example

- Suppose you know that Barry plays the violin and you'd like to represent this with an object property...but don't know which violin he plays...

ex:Barry ex:plays **[rdf:type ex:Violin]** .

Blank Nodes Example

- Suppose you know that Barry plays the violin and you'd like to represent this with an object property...but don't know which violin he plays...

ex:Barry ex:plays **[rdf:type ex:Violin]** .

- Which simply says Barry plays something that is of the type Violin

Blank Nodes Example

- Suppose you know that Barry plays the violin and you'd like to represent this with an object property...but don't know which violin he plays...

ex:Barry ex:plays **[rdf:type ex:Violin]** .

Note, there is an implicit numerical id being used here; this implicit id allows other triples in the graph to refer to the same violin Barry plays

Blank Nodes Example

- Suppose you know that Barry plays the violin and you'd like to represent this with an object property...but don't know which violin he plays...

ex:Barry ex:plays **_:0001** .

_0001 rdf:type ex:Violin .

Blank Nodes Example

- Suppose you know that Barry plays the violin and you'd like to represent this with an object property...but don't know which violin he plays...

ex:Barry ex:plays **_:0001** .

_0001 rdf:type ex:Violin .

- Importantly, this numerical id is only *locally* unique, not globally unique like a URI

Blank Nodes Example

- Suppose you know that Barry plays the violin and you'd like to represent this with an object property...but don't know which violin he plays...

ex:Barry ex:plays **_:0001** .

_:0001 rdf:type ex:Violin .

- And any time you change the serialization of an RDF file – for example, from XML to Turtle – the blank node identifiers *change*

Blank Nodes across Serializations

```
<rdf:RDF xmlns:ex="https://example.com/"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="https://example.com/Barry">
    <ex:plays rdf:nodeID="b1" />
  </rdf:Description>
  <rdf:Description rdf:nodeID="b1">
    <rdf:type rdf:resource="https://example.com/Violin" />
  </rdf:Description>
</rdf:RDF>
```

RDF/XML

```
{"@context": {
   "ex": "https://example.com/",
   "rdf": "http://www.w3.org/1999/02/22-rdf-
syntax-ns#" },
  "@id": "https://example.com/Barry",
  "ex:plays": {
   "rdf:type": "https://example.com/Violin" }}
```

JSON-LD

```
<https://example.com/Barry> <https://example.com/plays> _:b1 .
_:b1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<https://example.com/Violin> .
```

N-TRIPLES

```
ex:Barry ex:plays [ rdf:type ex:Violin ] .
```

TURTLE

Blank Nodes across Serializations

```
<rdf:RDF xmlns:ex="https://example.com/"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="https://example.com/Barry">
    <ex:plays rdf:nodeID="b1" />
  </rdf:Description>
  <rdf:Description rdf:nodeID="b1">
    <rdf:type rdf:resource="https://example.com/Violin" />
  </rdf:Description>
</rdf:RDF>
```

RDF/XML

```
{"@context": {
   "ex": "https://example.com/",
   "rdf": "http://www.w3.org/1999/02/22-rdf-
syntax-ns#" },
  "@id": "https://example.com/Barry",
  "ex:plays": {
   "rdf:type": "https://example.com/Violin" }}
```

JSON-LD

```
<https://example.com/Barry> <https://example.com/plays> _:b1 .
_:b1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<https://example.com/Violin> .
```

N-TRIPLES

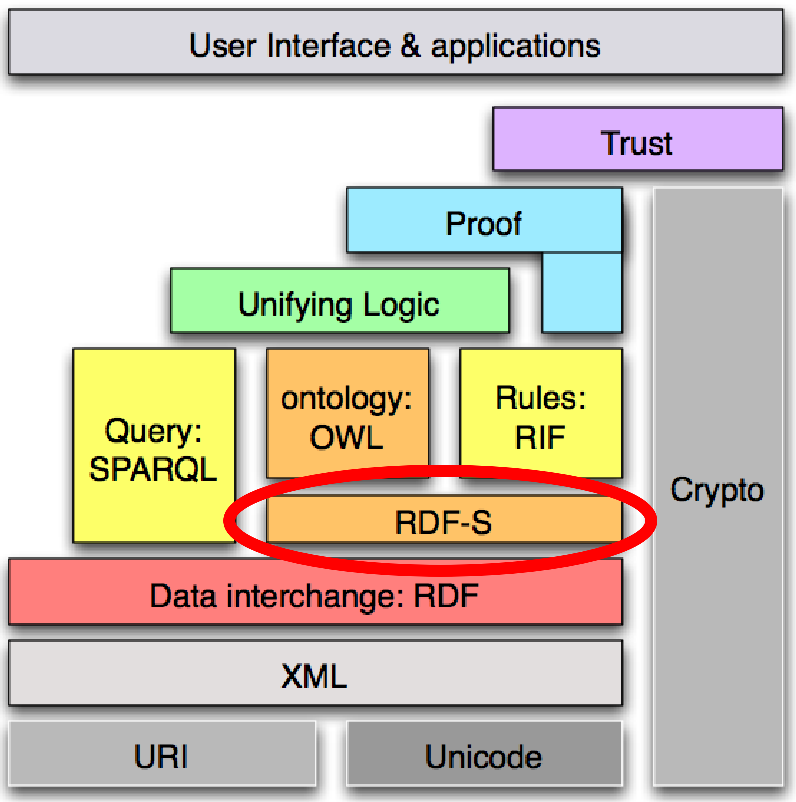
```
ex:Barry ex:plays [ rdf:type ex:Violin ] .
```

TURTLE

Outline

- Resource Description Framework
- Resource Description Framework Schema
- Web Ontology Language

Semantic Web Stack



Semantic Web Stack

- The "S" in RDFS stands for:

  Schema – A syntax and semantics for an intended extension of the RDF syntax

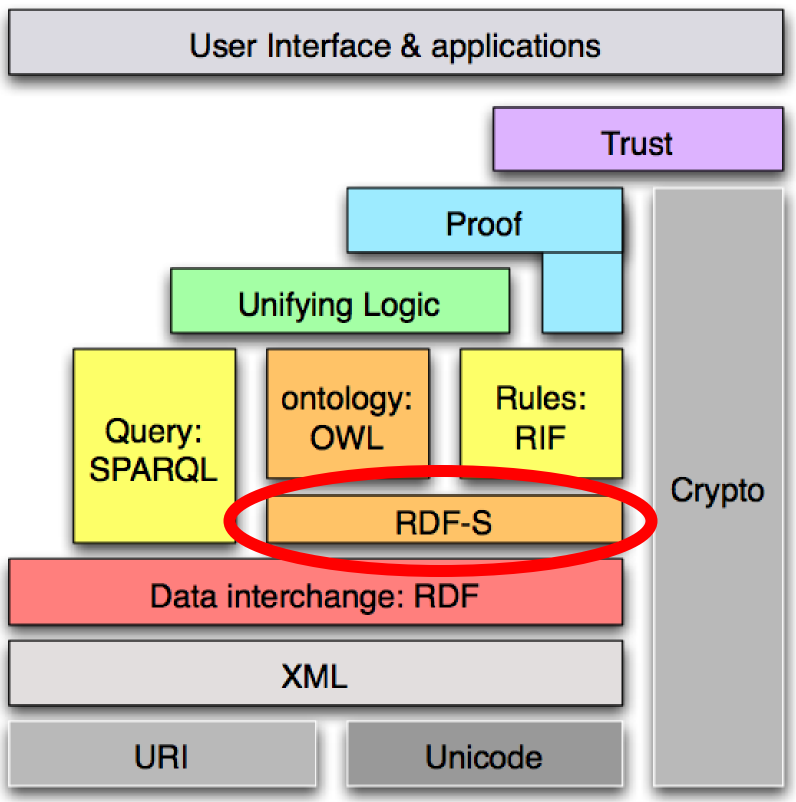
- RDFS is:

  A *vocabulary*

  That is based on the *RDF data model*

  That extends the *RDF vocabulary*

  And provide semantics for *connecting RDF resources*



Resource Description Framework Schema

- RDFs is an extension of RDF facilitating representation of:

rdfs:Class
rdfs:subClassOf
rdfs:domain
rdfs:range
rdfs:subPropertyOf
rdfs:Resource

rdfs:Datatype
rdfs:Literal
rdfs:label
rdfs:comment
rdfs:isDefinedBy

Resource Description Framework Schema

- RDFs is an extension of RDF facilitating representation of:

rdfs:Class
rdfs:subClassOf
rdfs:domain
rdfs:range
rdfs:subPropertyOf
rdfs:Resource
rdfs:Datatype
rdfs:Literal
rdfs:label
rdfs:comment
rdfs:isDefinedBy

We often say X rdf:type Y and intend Y to be a class, but strictly speaking there is no notion of "class" in rdf
rdfs:Class introduces the syntax needed for such assertions

Resource Description Framework Schema

- RDFs is an extension of RDF facilitating representation of:

rdfs:Class
rdfs:subClassOf
rdfs:domain
rdfs:range
rdfs:subPropertyOf
rdfs:Resource

rdfs:Datatype
rdfs:Literal
rdfs:label
rdfs:comment
rdfs:isDefinedBy

rdfs:subClassOf facilitates representing the "is a" hierarchy
e.g. bfo:object rdfs:subClassOf bfo:MaterialEntity

Resource Description Framework Schema

- RDFs is an extension of RDF facilitating representation of:

rdfs:Class
rdfs:subClassOf
rdfs:domain
rdfs:range
rdfs:subPropertyOf
rdfs:Resource

rdfs:Datatype
rdfs:Literal
rdfs:label
rdfs:comment
rdfs:isDefinedBy

For a binary relation R(x,y) the domain of R is whatever can occupy x and the range is whatever can occupy y

Resource Description Framework Schema

- RDFs is an extension of RDF facilitating representation of:

rdfs:Class
rdfs:subClassOf
rdfs:domain
rdfs:range
rdfs:subPropertyOf
rdfs:Resource

rdfs:Datatype
rdfs:Literal
rdfs:label
rdfs:comment
rdfs:isDefinedBy

Suppose we want to define a binary relation "has leaf".
Plausibly, the domain will be "plant" and the range will be "leaf"

Resource Description Framework Schema

- RDFs is an extension of RDF facilitating representation of:

rdfs:Class
rdfs:subClassOf
rdfs:domain
rdfs:range
rdfs:subPropertyOf
rdfs:Resource

rdfs:Datatype
rdfs:Literal
rdfs:label
rdfs:comment
rdfs:isDefinedBy

Much like we often find need to describe "is a" relationships between classes, so too we need to describe "is a" relationships between relations, e.g. x devours y is a sub-property of x eats y.

Resource Description Framework Schema

- RDFs is an extension of RDF facilitating representation of:

rdfs:Class
rdfs:subClassOf
rdfs:domain
rdfs:range
rdfs:subPropertyOf
rdfs:Resource

rdfs:Datatype
rdfs:Literal
rdfs:label
rdfs:comment
rdfs:isDefinedBy

The class of everything
Any instance represented in RDF is an instance of rdfs:Resource
Any class is a subclass of rdfs:Resource

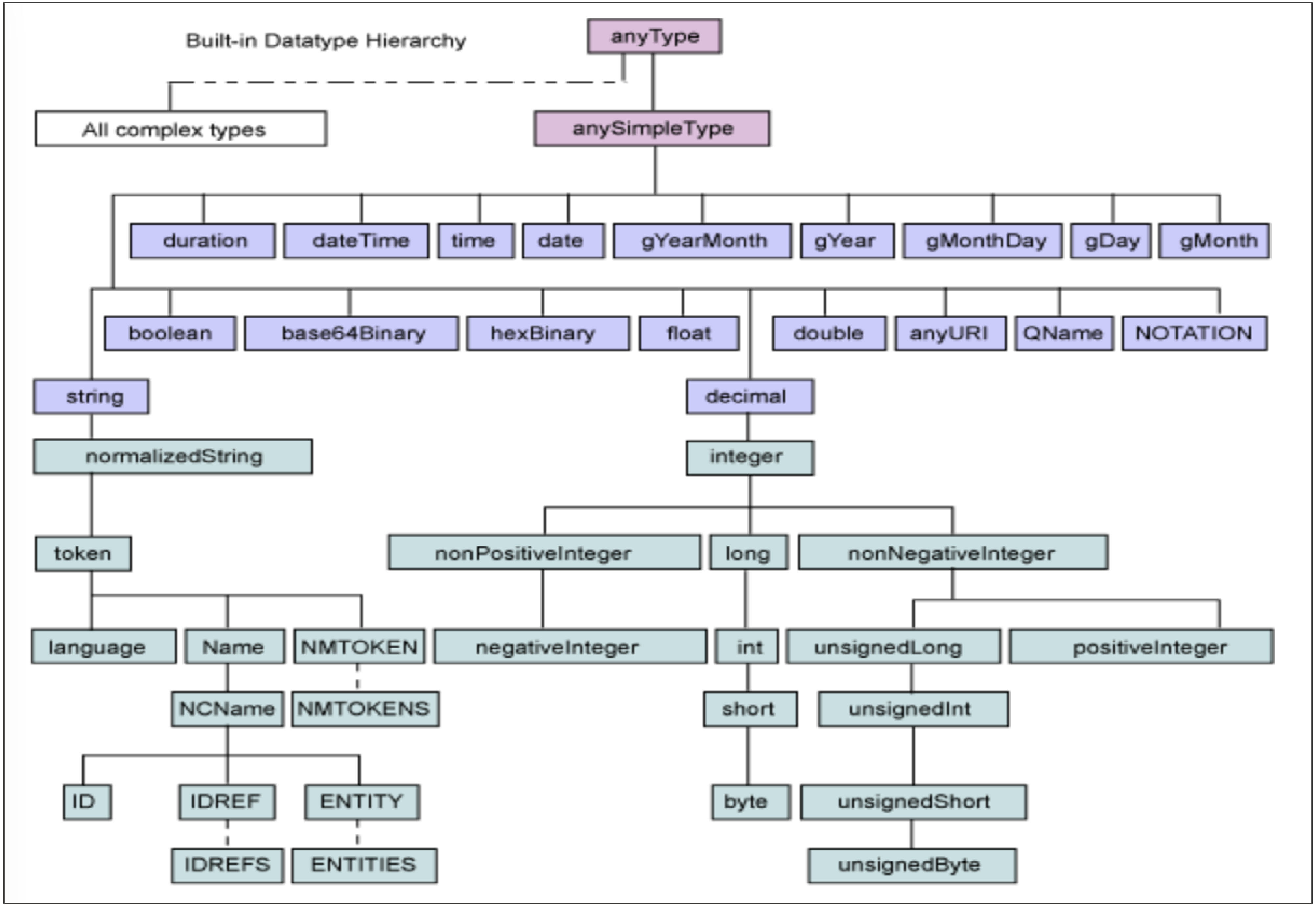
Resource Description Framework Schema

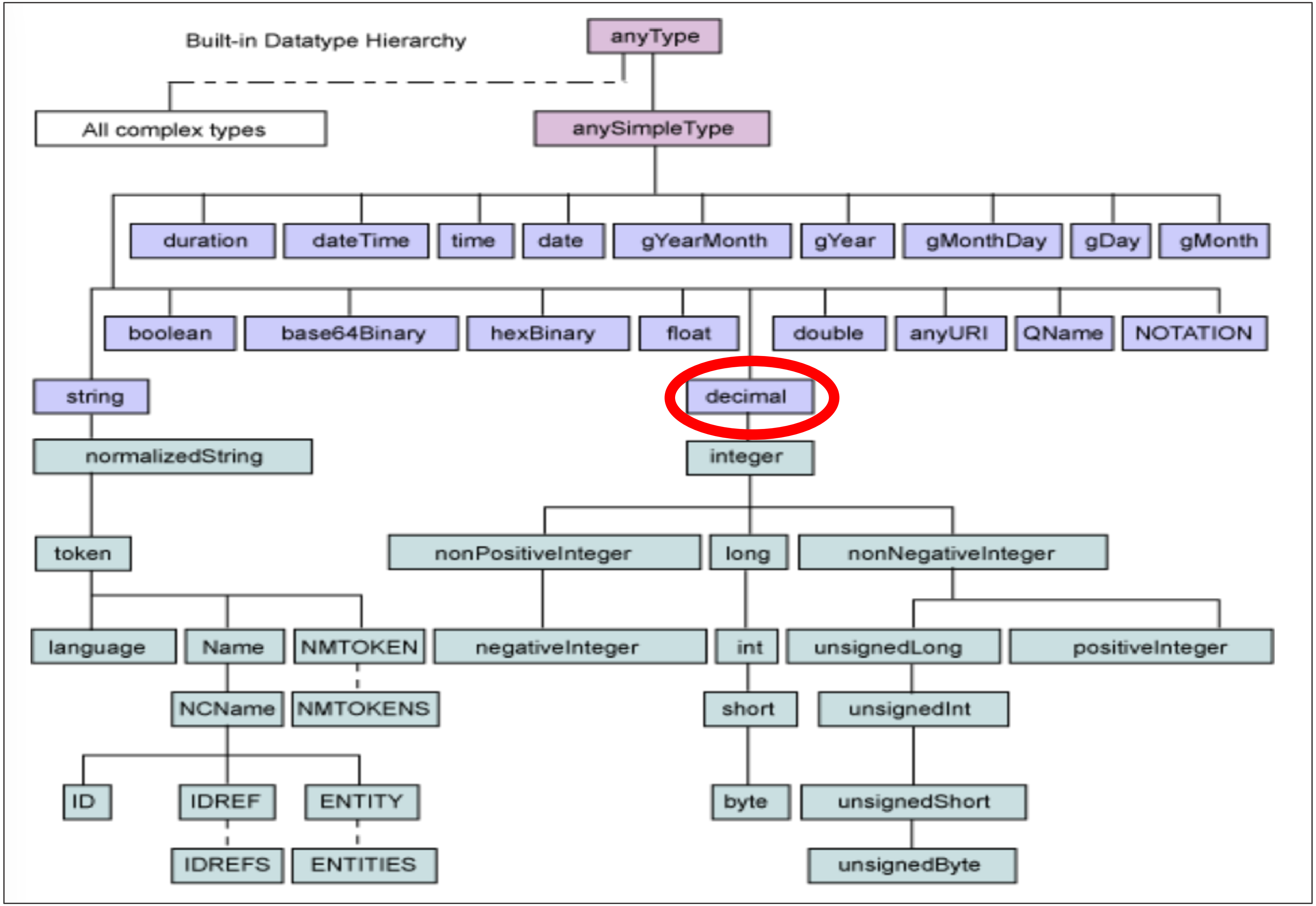
- RDFs is an extension of RDF facilitating representation of:

rdfs:Class
rdfs:subClassOf
rdfs:domain
rdfs:range
rdfs:subPropertyOf
rdfs:Resource

rdfs:Datatype
rdfs:Literal
rdfs:label
rdfs:comment
rdfs:isDefinedBy

rdfs:Datatype is a subclass of rdfs:Literal and is the class of all datatypes, such as "integer" or "decimal"


Built-in Datatype Hierarchy
anyType
All complex types
anySimpleType
duration
dateTime
time
date
gYearMonth
gYear
gMonthDay
gDay
gMonth
boolean
base64Binary
hexBinary
float
double
anyURI
QName
NOTATION
string
decimal
normalizedString
integer
token
nonPositiveInteger
long
nonNegativeInteger
language
Name
NMTOKEN
negativeInteger
int
unsignedLong
positiveInteger
NCName
NMTOKENS
short
unsignedInt
ID
IDREF
ENTITY
byte
unsignedShort
IDREFS
ENTITIES
unsignedByte


Built-in Datatype Hierarchy
anyType
All complex types
anySimpleType
duration
dateTime
time
date
gYearMonth
gYear
gMonthDay
gDay
gMonth
boolean
base64Binary
hexBinary
float
double
anyURI
QName
NOTATION
string
decimal
normalizedString
integer
token
nonPositiveInteger
long
nonNegativeInteger
language
Name
NMTOKEN
negativeInteger
int
unsignedLong
positiveInteger
NCName
NMTOKENS
short
unsignedInt
ID
IDREF
ENTITY
byte
unsignedShort
IDREFS
ENTITIES
unsignedByte

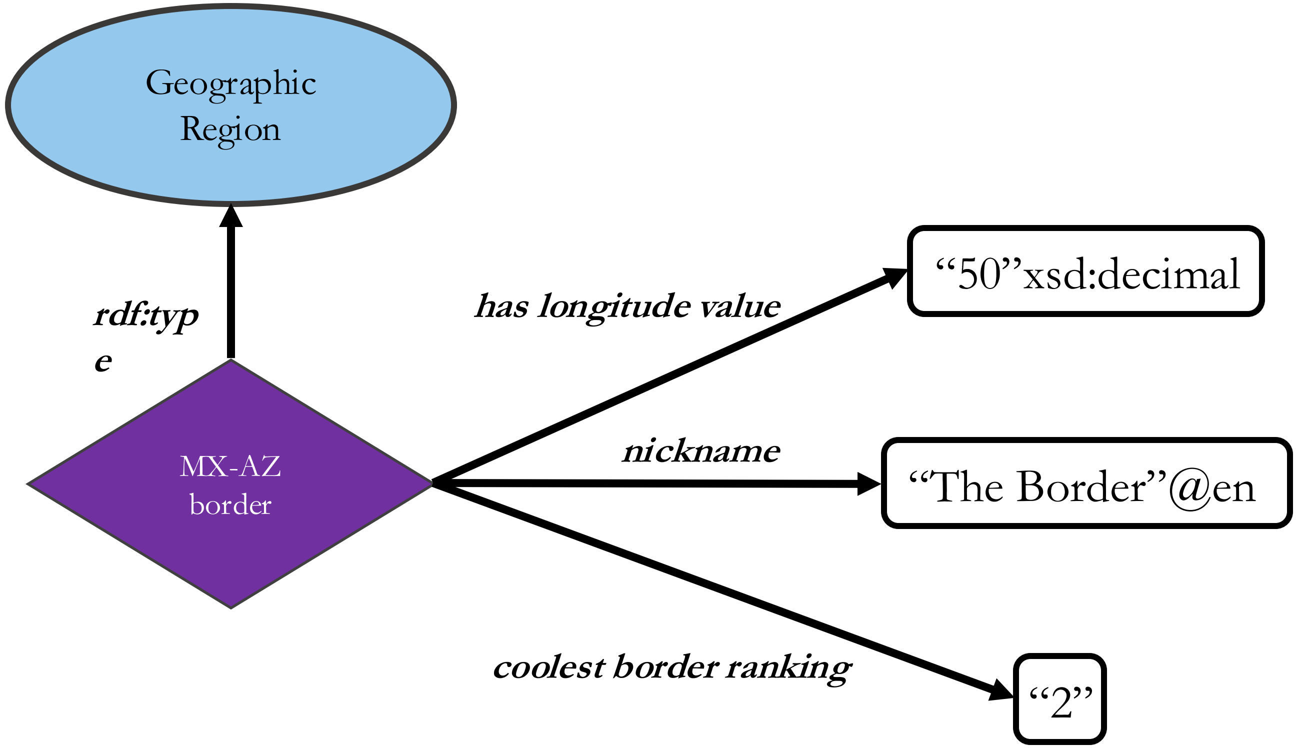
Resource Description Framework Schema

- RDFs is an extension of RDF facilitating representation of:

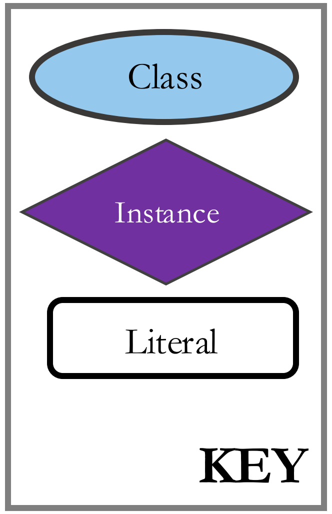
rdfs:Class	rdfs:Datatype
rdfs:subClassOf	rdfs:Literal
rdfs:domain	rdfs:label
rdfs:range	rdfs:comment
rdfs:subPropertyOf	rdfs:isDefinedBy
rdfs:Resource	

rdfs:Literal relates resources to strings, which can be “plain” or “typed”
Typed literals are strings with an associated datatype tag
Plain literals are strings without such datatype tags

Literals

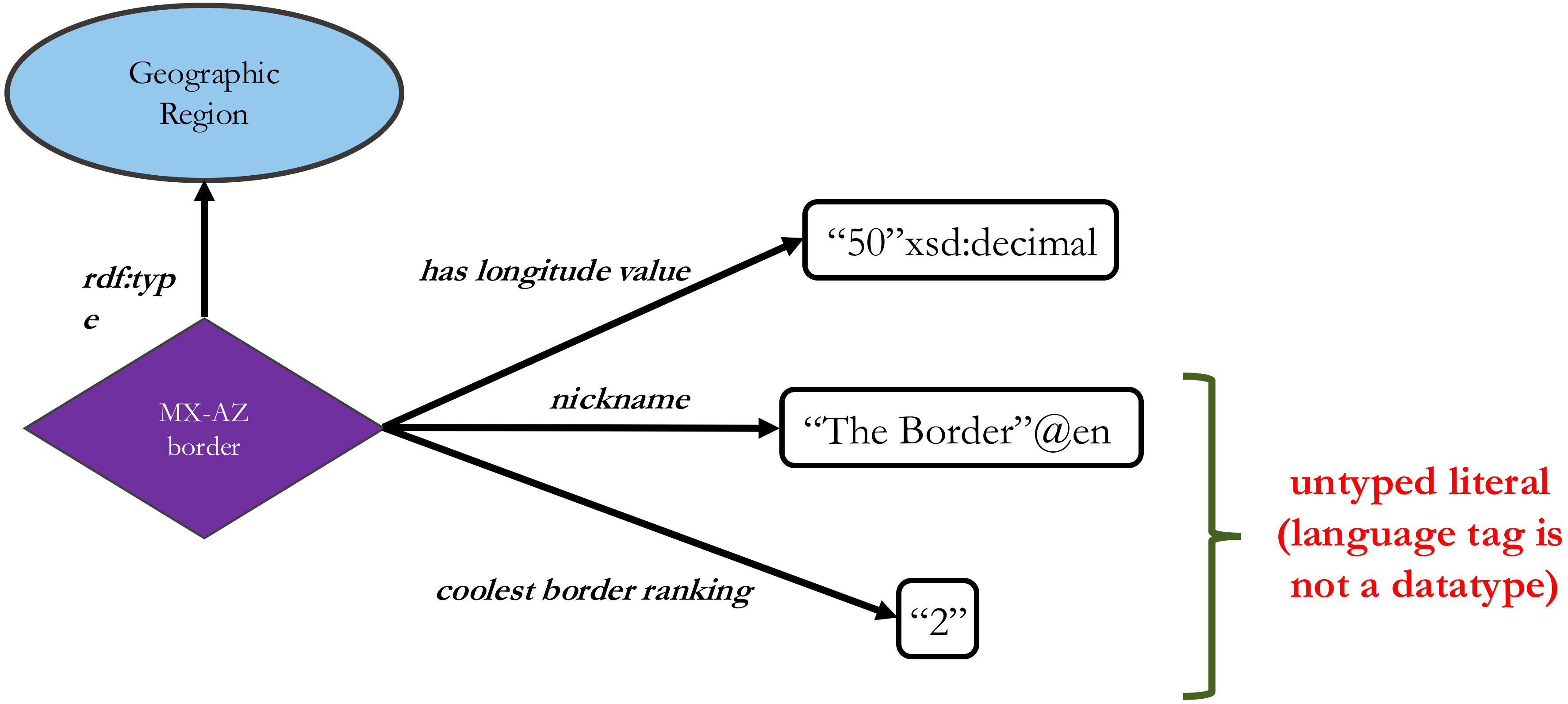


Geographic Region
rdf:type
MX-AZ border
has longitude value
“50”xsd:decimal
nickname
“The Border”@en
coolest border ranking
“2”

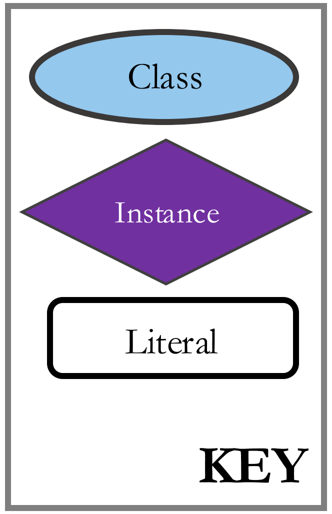


Class
Instance
Literal
KEY

Literals

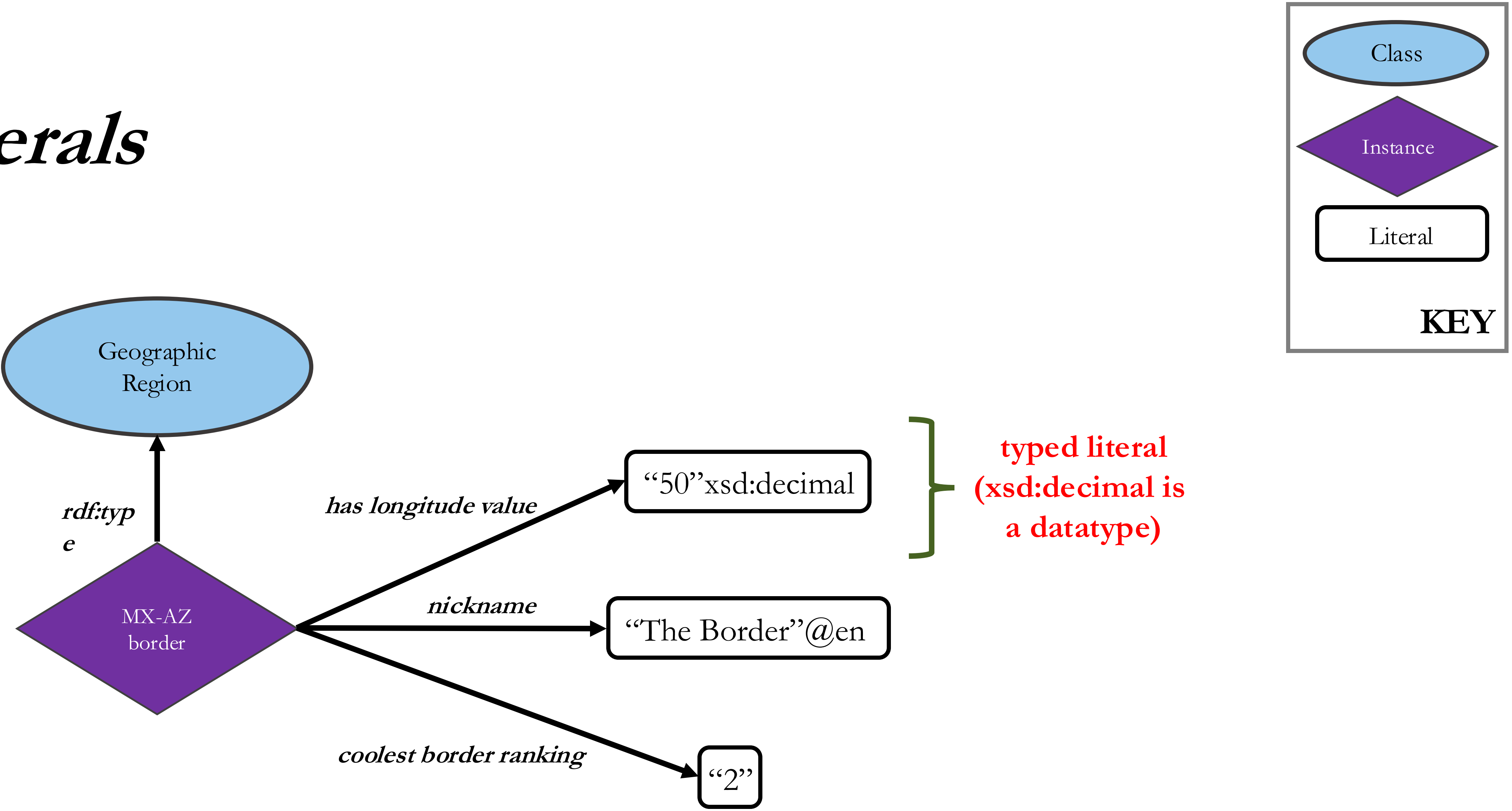


Geographic Region
rdf:typ e
MX-AZ border
has longitude value
“50”xsd:decimal
nickname
“The Border”@en
coolest border ranking
“2”
untyped literal (language tag is not a datatype)



Class
Instance
Literal
KEY

Literals


Class
Instance
Literal
KEY
Geographic Region
rdf:type
MX-AZ border
has longitude value
“50”xsd:decimal
typed literal (xsd:decimal is a datatype)
nickname
“The Border”@en
coolest border ranking
“2”

Logic of Datatypes

- xsd datatype semantics has been integrated in RDF, allowing for logical comparisons, equality, and calculations

Logic of Datatypes

- xsd datatype semantics has been integrated in RDF, allowing for logical comparisons, equality, and calculations

VALUE SPACE

Values datatype can represent.

E.g. xsd:integer has value space all integers

Logic of Datatypes

- xsd datatype semantics has been integrated in RDF, allowing for logical comparisons, equality, and calculations

VALUE SPACE

Values datatype can represent.

E.g. xsd:integer has value space all integers

LEXICAL SPACE

Strings representing valid values for datatype

E.g. xsd:integer has a lexical space including strings such as "1" and "-4"

Logic of Datatypes

- xsd datatype semantics has been integrated in RDF, allowing for logical comparisons, equality, and calculations

VALUE SPACE

Values datatype can represent.

E.g. xsd:integer has value space all integers

LEXICAL SPACE

Strings representing valid values for datatype

E.g. xsd:integer has a lexical space including strings such as “1” and “-4”

Both value **and** lexical space are needed because **some values have multiple lexical representations**, e.g. “01” and “1”

Logic of Datatypes

- When comparisons are conducted, for example by using an OWL reasoner, xsd datatype definitions are used to ensure:

  - Integer comparions are conducted **numerically**, e.g. “3”^^xsd:integer is less than “30”^^xsd:integer
  - Strings are compared **lexically**, e.g. “friend”^^xsd:string is identical to “friend”^^xsd:string, but not “Friend”^^xsd:string
  - Dates are ordered **chronologically**, e.g. “12-2-2023”^^xsd:datatime is earlier than “12-2-2024”^^xsd:datatime

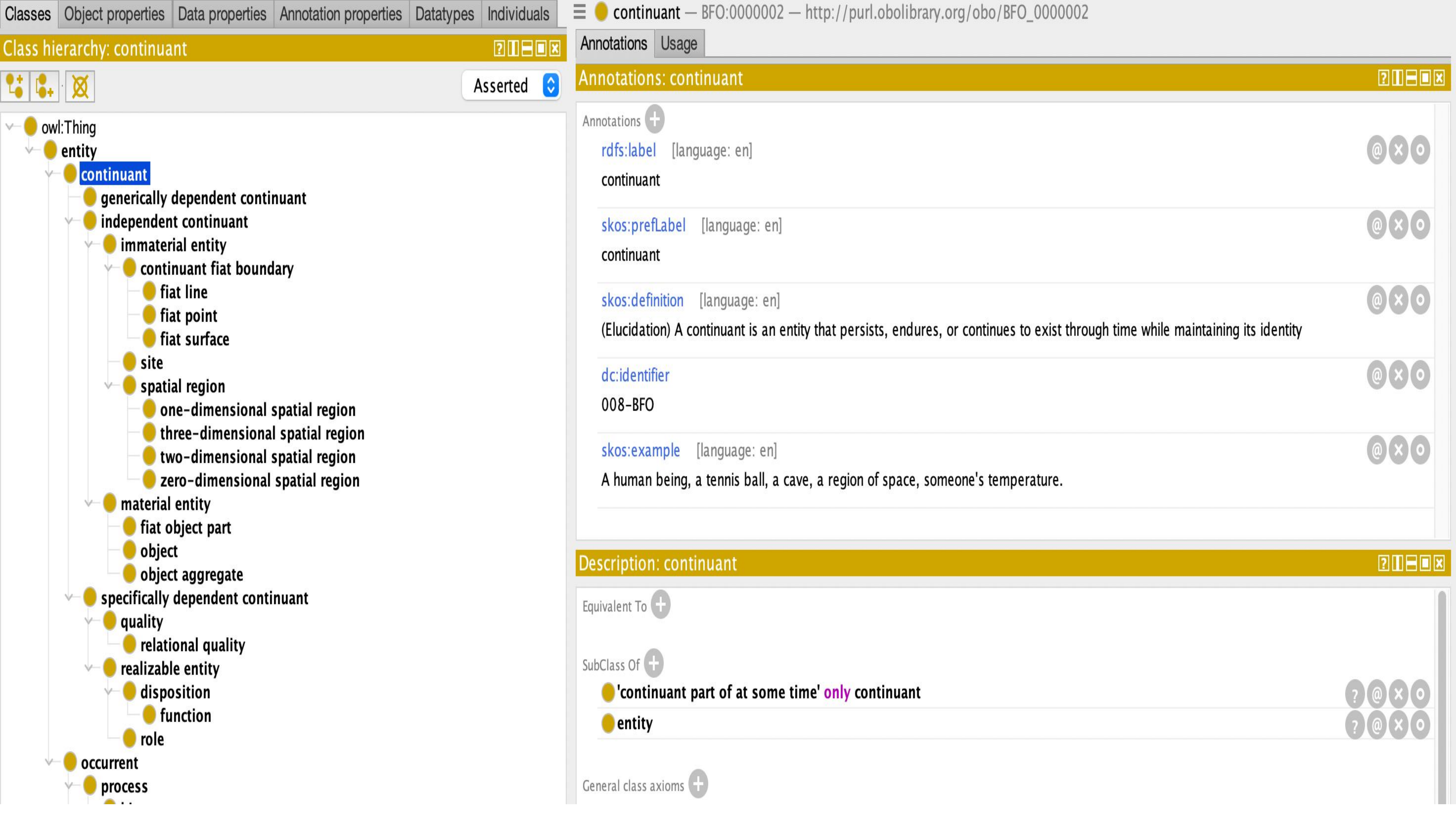
Resource Description Framework Schema

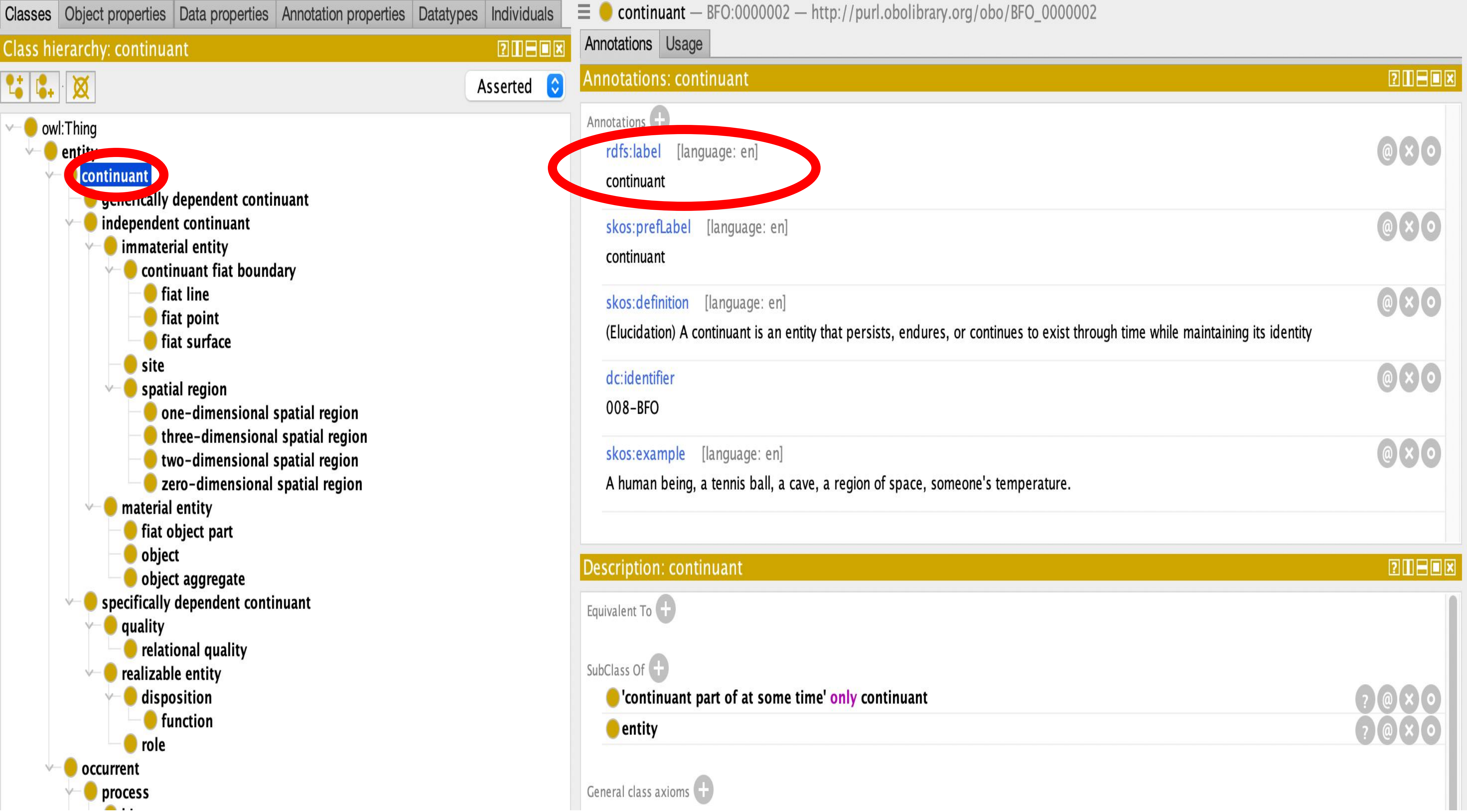
- RDFs is an extension of RDF facilitating representation of:

rdfs:Class
rdfs:subClassOf
rdfs:domain
rdfs:range
rdfs:subPropertyOf
rdfs:Resource

rdfs:Literal
rdfs:Datatype
rdfs:label
rdfs:comment
rdfs:isDefinedBy

rdfs:label is an annotation property that allows you to provide a human-readable name for resources, so you don't need to look at the ugly URI all the time…


Classes
Object properties
Data properties
Annotation properties
Datatypes
Individuals
Class hierarchy: continuant
Asserted
owl:Thing
entity
continuant
generically dependent continuant
independent continuant
immaterial entity
continuant fiat boundary
fiat line
fiat point
fiat surface
site
spatial region
one-dimensional spatial region
three-dimensional spatial region
two-dimensional spatial region
zero-dimensional spatial region
material entity
fiat object part
object
object aggregate
specifically dependent continuant
quality
relational quality
realizable entity
disposition
function
role
occurrent
process
continuant — BFO:0000002 — http://purl.obolibrary.org/obo/BFO_0000002
Annotations
Usage
Annotations: continuant
Annotations
rdfs:label [language: en]
continuant
skos:prefLabel [language: en]
continuant
skos:definition [language: en]
(Elucidation) A continuant is an entity that persists, endures, or continues to exist through time while maintaining its identity
dc:identifier
008-BFO
skos:example [language: en]
A human being, a tennis ball, a cave, a region of space, someone's temperature.
Description: continuant
Equivalent To
SubClass Of
'continuant part of at some time' only continuant
entity
General class axioms

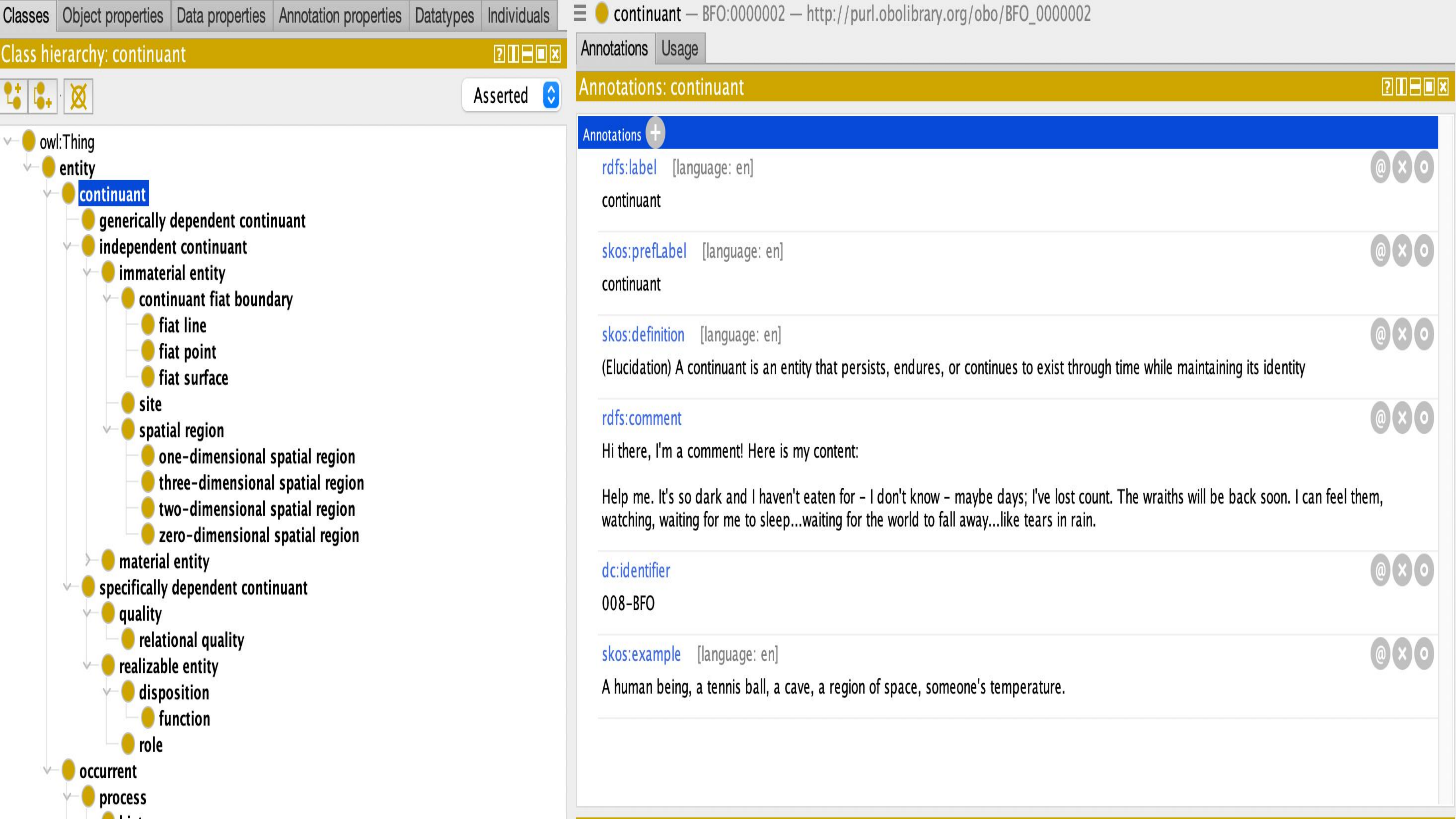

Classes
Object properties
Data properties
Annotation properties
Datatypes
Individuals
Class hierarchy: continuant
Asserted
owl:Thing
entity
continuant
generically dependent continuant
independent continuant
immaterial entity
continuant fiat boundary
fiat line
fiat point
fiat surface
site
spatial region
one-dimensional spatial region
three-dimensional spatial region
two-dimensional spatial region
zero-dimensional spatial region
material entity
fiat object part
object
object aggregate
specifically dependent continuant
quality
relational quality
realizable entity
disposition
function
role
occurrent
process
continuant — BFO:0000002 — http://purl.obolibrary.org/obo/BFO_0000002
Annotations
Usage
Annotations: continuant
Annotations
rdfs:label [language: en]
continuant
skos:prefLabel [language: en]
continuant
skos:definition [language: en]
(Elucidation) A continuant is an entity that persists, endures, or continues to exist through time while maintaining its identity
dc:identifier
008-BFO
skos:example [language: en]
A human being, a tennis ball, a cave, a region of space, someone's temperature.
Description: continuant
Equivalent To
SubClass Of
'continuant part of at some time' only continuant
entity
General class axioms

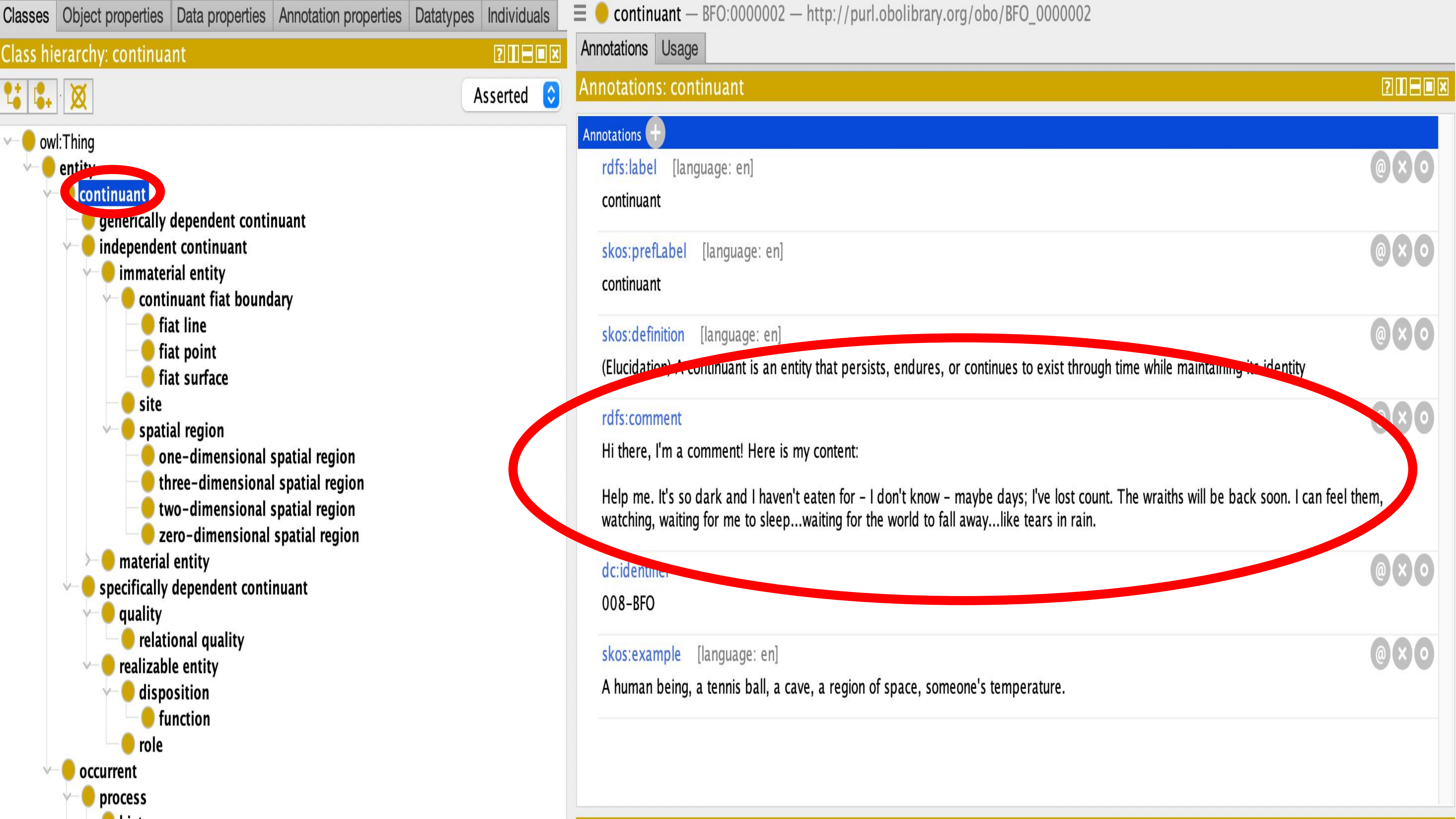
Resource Description Framework Schema

- RDFs is an extension of RDF facilitating representation of:

rdfs:Class
rdfs:subClassOf
rdfs:domain
rdfs:range
rdfs:subPropertyOf
rdfs:Resource
rdfs:Literal
rdfs:Datatype
rdfs:label
rdfs:comment
rdfs:isDefinedBy

rdfs:comment is an annotation property that allows to you create comments on resources for, say, suggested changes, updates, disagreements, etc.


Classes
Object properties
Data properties
Annotation properties
Datatypes
Individuals
Class hierarchy: continuant
Asserted
owl:Thing
entity
continuant
generically dependent continuant
independent continuant
immaterial entity
continuant fiat boundary
fiat line
fiat point
fiat surface
site
spatial region
one-dimensional spatial region
three-dimensional spatial region
two-dimensional spatial region
zero-dimensional spatial region
material entity
specifically dependent continuant
quality
relational quality
realizable entity
disposition
function
role
occurrent
process
continuant — BFO:0000002 — http://purl.obolibrary.org/obo/BFO_0000002
Annotations
Usage
Annotations: continuant
Annotations
rdfs:label [language: en]
continuant
skos:prefLabel [language: en]
continuant
skos:definition [language: en]
(Elucidation) A continuant is an entity that persists, endures, or continues to exist through time while maintaining its identity
rdfs:comment
Hi there, I'm a comment! Here is my content:
Help me. It's so dark and I haven't eaten for - I don't know - maybe days; I've lost count. The wraiths will be back soon. I can feel them, watching, waiting for me to sleep...waiting for the world to fall away...like tears in rain.
dc:identifier
008-BFO
skos:example [language: en]
A human being, a tennis ball, a cave, a region of space, someone's temperature.


Classes
Object properties
Data properties
Annotation properties
Datatypes
Individuals
Class hierarchy: continuant
Asserted
owl:Thing
entity
continuant
generically dependent continuant
independent continuant
immaterial entity
continuant fiat boundary
fiat line
fiat point
fiat surface
site
spatial region
one-dimensional spatial region
three-dimensional spatial region
two-dimensional spatial region
zero-dimensional spatial region
material entity
specifically dependent continuant
quality
relational quality
realizable entity
disposition
function
role
occurrent
process
continuant — BFO:0000002 — http://purl.obolibrary.org/obo/BFO_0000002
Annotations
Usage
Annotations: continuant
Annotations
rdfs:label [language: en]
continuant
skos:prefLabel [language: en]
continuant
skos:definition [language: en]
continuant is an entity that persists, endures, or continues to exist through time while maintaining its identity
rdfs:comment
Hi there, I'm a comment! Here is my content:
Help me. It's so dark and I haven't eaten for - I don't know - maybe days; I've lost count. The wraiths will be back soon. I can feel them, watching, waiting for me to sleep...waiting for the world to fall away...like tears in rain.
008-BFO
skos:example [language: en]
A human being, a tennis ball, a cave, a region of space, someone's temperature.

Resource Description Framework Schema

- RDFs is an extension of RDF facilitating representation of:

rdfs:Class
rdfs:subClassOf
rdfs:domain
rdfs:range
rdfs:subPropertyOf
rdfs:Resource

rdfs:Literal
rdfs:Datatype
rdfs:label
rdfs:comment
rdfs:isDefinedBy

rdfs:isDefinedBy is an annotation property indicating the primary location of a resource's definition

- entity
  - continuant
    - generically dependent continuant
    - independent continuant
    - specifically dependent continuant
  - occurrent
- obsolete class
- Version

Preferred Name	entity
Synonyms	
ID	http://purl.obolibrary.org/obo/BFO_0000001
BFO CLIF specification label	Entity
BFO OWL specification label	entity
editor note	BFO 2 Reference: In all areas of empirical inquiry we encounter general terms of two sorts. First are general terms which refer to types:animaltuberculosissurgical procedurediseaseSecond, are general terms used to refer to groups of entities which instantiate but do not correspond to the extension of any subuniversal of that universal because there is nothing intrinsic to the entities in which they – and only they – are counted as belonging to the given group. Examples are: animal purchased by the Emperortuberc a Wednesdaysurgical procedure performed on a patient from Stockholmperson identified as candidate for clinical trial #2056-55 signatory of Form 656-PPVpainting by Leonardo da VinciSuch terms, which represent what are called 'specializations' in [81 Entity doesn't have a closure axiom because the subclasses don't necessarily exhaust all possibilites. For example Werner Ceuste reality' include 4 sorts, entities (as BFO construes them), universals, configurations, and relations. It is an open question as to wh construed in BFO will at some point also include these other portions of reality. See, for example, 'How to track absolutely everyt http://www.referent-tracking.com/_RTU/papers/CeustersICbookRevised.pdf
editor preferred label	entity
elucidation	An entity is anything that exists or has existed or will exist. (axiom label in BFO2 Reference: [001-001])
example of usage	the Second World War Julius Caesar Verdi's Requiem your body mass index
imported from	http://purl.obolibrary.org/obo/uberon.owl http://purl.obolibrary.org/obo/caro.owl http://purl.obolibrary.org/obo/obi.owl http://purl.obolibrary.org/obo/BFO
label	entity
prefixIRI	BFO:0000001
prefLabel	entity
rdfs:isDefinedBy	http://purl.obolibrary.org/obo/bfo.owl
subClassOf	Thing

From the Cell Line Ontology, which imports BFO as its top-level architecture.

BFO is the primary location of the definition for "entity"

- entity
  - continuant
    - generically dependent continuant
    - independent continuant
    - specifically dependent continuant
  - occurrent
- obsolete class
- Version

Preferred Name	entity
Synonyms	
ID	http://purl.obolibrary.org/obo/BFO_0000001
BFO CLIF specification label	Entity
BFO OWL specification label	entity
editor note	BFO 2 Reference: In all areas of empirical inquiry we encounter general terms of two sorts. First are general terms which refer to types:animaltuberculosissurgical procedurediseaseSecond, are general terms used to refer to groups of entities which instantiate but do not correspond to the extension of any subuniversal of that universal because there is nothing intrinsic to the entities in which they – and only they – are counted as belonging to the given group. Examples are: animal purchased by the Emperortuberc a Wednesdaysurgical procedure performed on a patient from Stockholmperson identified as candidate for clinical trial #2056-55 signatory of Form 656-PPVpainting by Leonardo da VinciSuch terms, which represent what are called ‘specializations’ in [81 Entity doesn't have a closure axiom because the subclasses don't necessarily exhaust all possibilites. For example Werner Ceuste reality' include 4 sorts, entities (as BFO construes them), universals, configurations, and relations. It is an open question as to wh construed in BFO will at some point also include these other portions of reality. See, for example, 'How to track absolutely everyt http://www.referent-tracking.com/_RTU/papers/CeustersICbookRevised.pdf
editor preferred label	entity
elucidation	An entity is anything that exists or has existed or will exist. (axiom label in BFO2 Reference: [001-001])
example of usage	the Second World War Julius Caesar Verdi’s Requiem your body mass index
imported from	http://purl.obolibrary.org/obo/uberon.owl http://purl.obolibrary.org/obo/caro.owl http://purl.obolibrary.org/obo/obi.owl http://purl.obolibrary.org/obo/BFO
label	entity
prefixIRI	BFO:0000001
prefLabel	entity
rdfs:isDefinedBy	http://purl.obolibrary.org/obo/bfo.owl
subClassOf	Thing

From the Cell Line Ontology, which imports BFO as its top-level architecture.

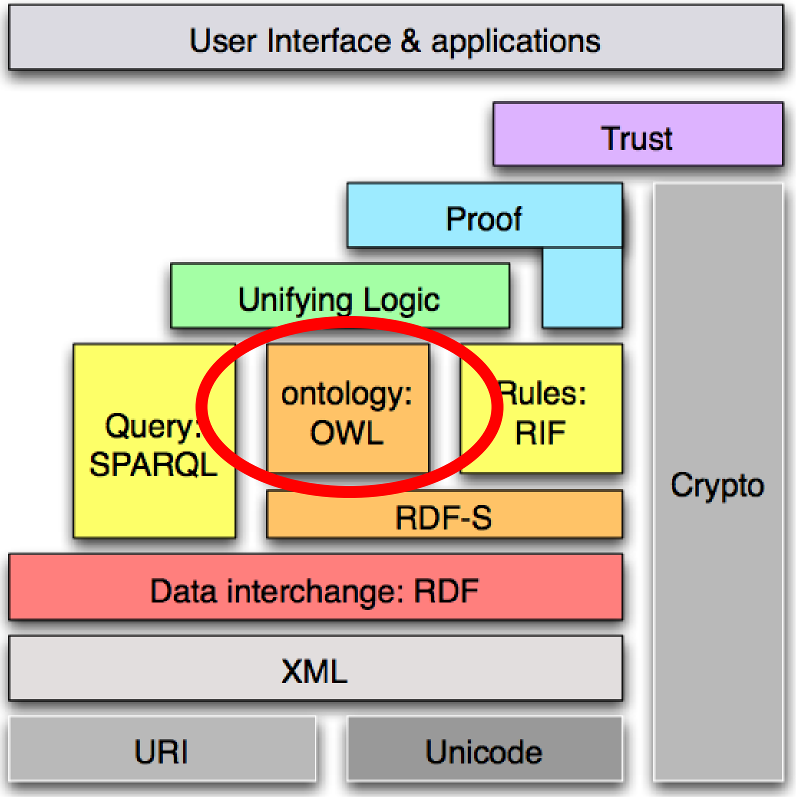
BFO is the primary location of the definition for “entity”

Outline

- Resource Description Framework
- Resource Description Framework Schema
- Web Ontology Language

Semantic Web Stack

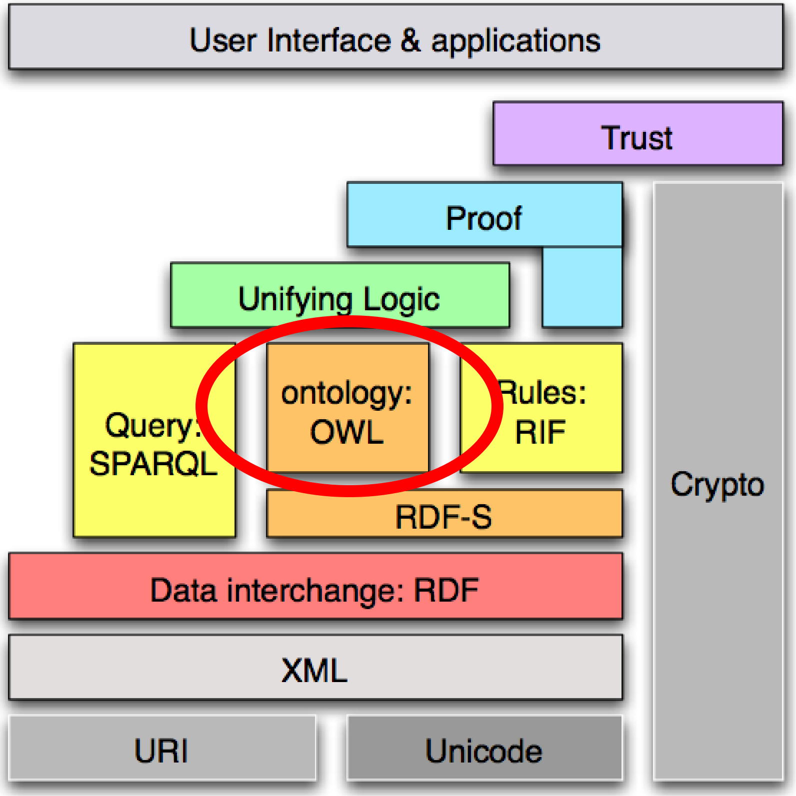
- "OWL" stands for:
  Web
  Ontology
  Language



Semantic Web Stack

- "OWL" stands for:
  Web
  Ontology
  Language

- OWL is:
  A *family of vocabularies*
  That extend *RDF and RDFS*
  Which provide semantics for constructing *logical relationships among resources*



OWL2

- We will focus on the most widely used OWL language, called "OWL2"
- OWL2 has a clear connection to description logics
- The vocabulary includes terms for $\top$, $\bot$, $\sqcup$, etc. but also complex combinations of description logic syntax such as disjointness

OWL2

- We will focus on the most widely used OWL language, called “OWL2”
- OWL2 has a clear connection to description logics
- The vocabulary includes terms for $\top$, $\bot$, $\sqcup$, etc. but also complex combinations of description logic syntax such as disjointness

```
owl:unionOf
    a rdf:Property ;
    rdfs:comment "The property that determines the collection of classes or data ranges that build a union." ;
    rdfs:domain rdfs:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "unionOf" ;
    rdfs:range rdf:List .
```

```
owl:disjointUnionOf
    a rdf:Property ;
    rdfs:comment "The property that determines that a given class is equivalent to the disjoint union of a collection of other classes." ;
    rdfs:domain owl:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "disjointUnionOf" ;
    rdfs:range rdf:List .

owl:disjointWith
    a rdf:Property ;
    rdfs:comment "The property that determines that two given classes are disjoint." ;
    rdfs:domain owl:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "disjointWith" ;
    rdfs:range owl:Class .
```

```
owl:Nothing
    a owl:Class ;
    rdfs:comment "This is the empty class." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "Nothing" ;
    rdfs:subClassOf owl:Thing .
```

```
owl:topObjectProperty
    a owl:ObjectProperty ;
    rdfs:comment "The object property that relates every two individuals." ;
    rdfs:domain owl:Thing ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "topObjectProperty" ;
    rdfs:range owl:Thing .
```

```
owl:unionOf
    a rdf:Property ;
    rdfs:comment "The property that determines the collection of classes or data ranges that build a union." ;
    rdfs:domain rdfs:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "unionOf" ;
    rdfs:range rdf:List .
```

```
owl:disjointUnionOf
    a rdf:Property ;
    rdfs:comment "The property that determines that a given class is equivalent to the disjoint union of a collection of other classes." ;
    rdfs:domain owl:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "disjointUnionOf" ;
    rdfs:range rdf:List .

owl:disjointWith
    a rdf:Property ;
    rdfs:comment "The property that determines that two given classes are disjoint." ;
    rdfs:domain owl:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "disjointWith" ;
    rdfs:range owl:Class .
```

$\perp$

```
owl:Nothing
    a owl:Class ;
    rdfs:comment "This is the empty class." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "Nothing" ;
    rdfs:subClassOf owl:Thing .
```

```
owl:topObjectProperty
    a owl:ObjectProperty ;
    rdfs:comment "The object property that relates every two individuals." ;
    rdfs:domain owl:Thing ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "topObjectProperty" ;
    rdfs:range owl:Thing .
```

OWL2

- We will focus on the most widely used OWL language, called "OWL2"
- OWL2 has a clear connection to description logics
- The vocabulary includes terms for $\top$, $\bot$, $\sqcup$, etc. but also complex combinations of description logic syntax such as disjointness

$\sqcup$

```
owl:unionOf
    a rdf:Property ;
    rdfs:comment "The property that determines the collection of classes or data ranges that build a union." ;
    rdfs:domain rdfs:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "unionOf" ;
    rdfs:range rdf:List .
```

```
owl:disjointUnionOf
    a rdf:Property ;
    rdfs:comment "The property that determines that a given class is equivalent to the disjoint union of a collection of other classes." ;
    rdfs:domain owl:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "disjointUnionOf" ;
    rdfs:range rdf:List .

owl:disjointWith
    a rdf:Property ;
    rdfs:comment "The property that determines that two given classes are disjoint." ;
    rdfs:domain owl:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "disjointWith" ;
    rdfs:range owl:Class .
```

```
owl:Nothing
    a owl:Class ;
    rdfs:comment "This is the empty class." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "Nothing" ;
    rdfs:subClassOf owl:Thing .
```

```
owl:topObjectProperty
    a owl:ObjectProperty ;
    rdfs:comment "The object property that relates every two individuals." ;
    rdfs:domain owl:Thing ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "topObjectProperty" ;
    rdfs:range owl:Thing .
```

OWL2

- We will focus on the most widely used OWL language, called "OWL2"
- OWL2 has a clear connection to description logics
- The vocabulary includes terms for $\top$, $\bot$, $\sqcup$, etc. but also complex combinations of description logic syntax such as disjointness

```
owl:unionOf
    a rdf:Property ;
    rdfs:comment "The property that determines the collection of classes or data ranges that build a union." ;
    rdfs:domain rdfs:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "unionOf" ;
    rdfs:range rdf:List .
```

```
owl:disjointUnionOf
    a rdf:Property ;
    rdfs:comment "The property that determines that a given class is equivalent to the disjoint union of a collection of other classes." ;
    rdfs:domain owl:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "disjointUnionOf" ;
    rdfs:range rdf:List .

owl:disjointWith
    a rdf:Property ;
    rdfs:comment "The property that determines that two given classes are disjoint." ;
    rdfs:domain owl:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "disjointWith" ;
    rdfs:range owl:Class .
```

```
owl:Nothing
    a owl:Class ;
    rdfs:comment "This is the empty class." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "Nothing" ;
    rdfs:subClassOf owl:Thing .
```

```
owl:topObjectProperty
    a owl:ObjectProperty ;
    rdfs:comment "The object property that relates every two individuals." ;
    rdfs:domain owl:Thing ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "topObjectProperty" ;
    rdfs:range owl:Thing .
```

OWL2

- We will focus on the most widely used OWL language, called "OWL2"
- OWL2 has a clear connection to description logics
- The vocabulary includes terms for $\top$, $\bot$, $\sqcup$, etc. but also complex combinations of description logic syntax such as disjointness
- Additionally, identity, quantified roles such as $\exists r.C$, role properties such as functional and inverse functional, as well as instance declarations and negative property assertions

OWL2

- We will focus on the most widely used OWL language, called “OWL2”
- OWL2 has a clear connection to description logics
- The vocabulary includes terms for $\top$, $\bot$, $\sqcup$, etc. but also complex combinations of description logic syntax such as disjointness
- Additionally, identity, quantified roles such as $\exists r.C$, role properties such as functional and inverse functional, as well as instance declarations and negative property assertions

```
owl:sameAs
    a rdf:Property ;
    rdfs:comment "The property that determines that two given individuals are equal." ;
    rdfs:domain owl:Thing ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "sameAs" ;
    rdfs:range owl:Thing .
```

=

```
owl:someValuesFrom
    a rdf:Property ;
    rdfs:comment "The property that determines the class that an existential property restriction refers to." ;
    rdfs:domain owl:Restriction ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "someValuesFrom" ;
    rdfs:range rdfs:Class .
```

$\exists r.C$

```
owl:FunctionalProperty
    a rdfs:Class ;
    rdfs:comment "The class of functional properties." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "FunctionalProperty" ;
    rdfs:subClassOf rdf:Property .

owl:InverseFunctionalProperty
    a rdfs:Class ;
    rdfs:comment "The class of inverse-functional properties." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "InverseFunctionalProperty" ;
    rdfs:subClassOf owl:ObjectProperty .
```

```
owl:NamedIndividual
    a rdfs:Class ;
    rdfs:comment "The class of named individuals." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "NamedIndividual" ;
    rdfs:subClassOf owl:Thing .

owl:NegativePropertyAssertion
    a rdfs:Class ;
    rdfs:comment "The class of negative property assertions." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "NegativePropertyAssertion" ;
    rdfs:subClassOf rdfs:Resource .
```

OWL2

- We will focus on the most widely used OWL language, called “OWL2”
- OWL2 has a clear connection to description logics
- The vocabulary includes terms for $\top$, $\bot$, $\sqcup$, etc. but also complex combinations of description logic syntax such as disjointness
- Additionally, identity, quantified roles such as $\exists$r.C, role properties such as functional and inverse functional, as well as instance declarations and negative property assertions

```
owl:sameAs
    a rdf:Property ;
    rdfs:comment "The property that determines that two given individuals are equal." ;
    rdfs:domain owl:Thing ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "sameAs" ;
    rdfs:range owl:Thing .

owl:someValuesFrom
    a rdf:Property ;
    rdfs:comment "The property that determines the class that an existential property restriction refers to." ;
    rdfs:domain owl:Restriction ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "someValuesFrom" ;
    rdfs:range rdfs:Class .
```

```
owl:FunctionalProperty
    a rdfs:Class ;
    rdfs:comment "The class of functional properties." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "FunctionalProperty" ;
    rdfs:subClassOf rdf:Property .

owl:InverseFunctionalProperty
    a rdfs:Class ;
    rdfs:comment "The class of inverse-functional properties." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "InverseFunctionalProperty" ;
    rdfs:subClassOf owl:ObjectProperty .
```

```
owl:NamedIndividual
    a rdfs:Class ;
    rdfs:comment "The class of named individuals." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "NamedIndividual" ;
    rdfs:subClassOf owl:Thing .

owl:NegativePropertyAssertion
    a rdfs:Class ;
    rdfs:comment "The class of negative property assertions." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "NegativePropertyAssertion" ;
    rdfs:subClassOf rdfs:Resource .
```

OWL2

- We will focus on the most widely used OWL language, called “OWL2”
- OWL2 has a clear connection to description logics
- The vocabulary includes terms for $\top$, $\bot$, $\sqcup$, etc. but also complex combinations of description logic syntax such as disjointness
- Additionally, identity, quantified roles such as $\exists r.C$, role properties such as functional and inverse functional, as well as instance declarations and negative property assertions

```
owl:sameAs
    a rdf:Property ;
    rdfs:comment "The property that determines that two given individuals are equal." ;
    rdfs:domain owl:Thing ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "sameAs" ;
    rdfs:range owl:Thing .

owl:someValuesFrom
    a rdf:Property ;
    rdfs:comment "The property that determines the class that an existential property restriction refers to." ;
    rdfs:domain owl:Restriction ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "someValuesFrom" ;
    rdfs:range rdfs:Class .
```

```
owl:FunctionalProperty
    a rdfs:Class ;
    rdfs:comment "The class of functional properties." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "FunctionalProperty" ;
    rdfs:subClassOf rdf:Property .

owl:InverseFunctionalProperty
    a rdfs:Class ;
    rdfs:comment "The class of inverse-functional properties." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "InverseFunctionalProperty" ;
    rdfs:subClassOf owl:ObjectProperty .
```

```
owl:NamedIndividual
    a rdfs:Class ;
    rdfs:comment "The class of named individuals." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "NamedIndividual" ;
    rdfs:subClassOf owl:Thing .

owl:NegativePropertyAssertion
    a rdfs:Class ;
    rdfs:comment "The class of negative property assertions." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "NegativePropertyAssertion" ;
    rdfs:subClassOf rdfs:Resource .
```

OWL2

- We will focus on the most widely used OWL language, called “OWL2”
- OWL2 has a clear connection to description logics
- OWL2 comes in two ‘flavors’ of differing logical strengths:
  - OWL2 Full
  - OWL2 DL: Direct Semantics

OWL2

- We will focus on the most widely used OWL language, called "OWL2"
- OWL2 has a clear connection to description logics
- OWL2 comes in two 'flavors' of differing logical strengths:
  - OWL2 Full
  - OWL2 DL: Direct Semantics

OWL2 Full

- OWL2 was designed for reasoning support, hence the focus on representing logical connections among resources
- However, combining the RDFS and OWL2 vocabularies without restriction, results in a language too expressive to accommodate efficient reasoning support
- *OWL2 Full* uses all the OWL2 vocabulary and allows arbitrary combinations of this vocabulary with RDF and RDFS

OWL2 Full

- This undermines efficient reasoning in part because arbitrary combinations of OWL2 and RDF **allow one to change the meaning of predefined RDF or OWL2** vocabulary items:

- For example, in **OWL2** Full you can:
  - Impose a cardinality constraint on the class of all classes, thereby limiting the number of classes that can be represented in an ontology
  - Relate any instance of a class directly to another class

OWL2

- We will focus on the most widely used OWL language, called “OWL2”
- OWL2 has a clear connection to description logics
- OWL2 comes in two ‘flavors’ of differing logical strengths:
  - OWL2 Full
  - OWL2 DL: Direct Semantics

OWL2 DL Direct Semantics

- *OWL2 DL* is a restriction of the OWL2 vocabulary by mapping it directly to a decidable description logic

OWL2 DL Direct Semantics

- *OWL2 DL* is a restriction of the OWL2 vocabulary by mapping it directly to a **decidable description logic**

- Notable consequences of this restriction include:

  - OWL2 vocabulary terms **cannot** be applied to each other
  - All OWL2 classes are **instances of owl:Class** rather than rdfs:Class
  - OWL2 properties are either owl:ObjectProperty or owl:DatatypeProperty but not both
  - OWL2 resources **cannot simultaneously** be class, property, and instance

Summary

- OWL2 DL **can be used** by standard reasoners one finds in Protege, but OWL2 Full **cannot**

- Important takeaways thus far -

  - Simple RDF will often need to be extended to be useful
  - Extended RDF will need to be restricted to be usable
  - Any legal OWL2 DL file is a legal RDF file
  - It is not the case any legal OWL2 DL file is a legal RDF file

OWL2 DL and SROIQ

- OWL2 DL corresponds to the description logic *SROIQ*

- SROIQ:
  - S = ALC
  - R = Role Axiom Extension
  - O = Nominals
  - I = Inverses
  - Q = Qualified Cardinalities

OWL2 DL and SROIQ

- OWL2 DL corresponds to the description logic SROIQ

- SROIQ:
  - S = ALC
  - R = Role Axiom Extension
  - O = Nominals
  - I = Inverses
  - Q = Qualified Cardinalities

ALC Syntax

Definition 2.1. Let $\mathbf{C}$ be a set of *concept names* and $\mathbf{R}$ be a set of *role names* disjoint from $\mathbf{C}$. The set of $\mathcal{ALC}$ *concept descriptions* over $\mathbf{C}$ and $\mathbf{R}$ is inductively defined as follows:

- Every concept name is an $\mathcal{ALC}$ concept description.
- $\top$ and $\bot$ are $\mathcal{ALC}$ concept descriptions.
- If C and D are $\mathcal{ALC}$ concept descriptions and r is a role name, then the following are also $\mathcal{ALC}$ concept descriptions:

  $C \sqcap D$ (conjunction),

  $C \sqcup D$ (disjunction),

  $\neg C$ (negation),

  $\exists r.C$, (existential restriction), and

  $\forall r.C$ (value restriction).

ALC Semantics

Definition 2.2. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$, called the *interpretation domain*, and a mapping $\cdot^{\mathcal{I}}$ that maps

- every concept name $A \in \mathbf{C}$ to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and
- every role name $r \in \mathbf{R}$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}}, \\
\bot^{\mathcal{I}} &= \emptyset, \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}}, \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}}, \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}, \\
(\exists r.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \text{there is an } e \in \Delta^{\mathcal{I}} \text{ with } (d,e) \in r^{\mathcal{I}} \text{ and } e \in C^{\mathcal{I}}\}, \\
(\forall r.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \text{for all } e \in \Delta^{\mathcal{I}},\ \text{if } (d,e) \in r^{\mathcal{I}},\ \text{then } e \in C^{\mathcal{I}}\}.
\end{aligned}
$$

OWL2 DL and SROIQ

- OWL2 DL corresponds to the description logic SROIQ

- SROIQ:
  - S = ALC
  - R = Role Axiom Extension
  - O = Nominals
  - I = Inverses
  - Q = Qualified Cardinalities

ALCO (nominal) Syntax/Semantics

ALCO Signature = ALC Signature + {{a}, {b}, …}

- {a} – Corresponds to the instance mapped to by the name "a"

- **Note**
  - Nominals allow for defining classes by enumerations of instances

OWL2 DL and SROIQ

- OWL2 DL corresponds to the description logic SROIQ

- SROIQ:
  - S = ALC
  - R = Role Axiom Extension
  - O = Nominals
  - I = Inverses
  - Q = Qualified Cardinalities

ALCI (inverses) Syntax/Semantics

ALCI Signature = ALC Signature + $\{r^{-}_{1\ldots n}\}$

- $r^{-}_{1\ldots n}$ – Corresponds to inversions of relations such as r between instances, such as the inverse of 'loves' being 'loves^{-}', i.e. 'loved by'

OWL2 DL and SROIQ

- OWL2 DL corresponds to the description logic SROIQ

- SROIQ:
  - S = ALC
  - R = Role Axiom Extension
  - O = Nominals
  - I = Inverses
  - Q = Qualified Cardinalities

ALCQ (qual. cardinality) Syntax/Semantics

ALCQ Signature = ALC Signature + {$\leqslant$n r.C, $\geqslant$n r.C}

- $\leqslant$n r.C – Corresponds to restriction that r is related to no more than n Cs
- $\geqslant$n r.C – Corresponds to restriction that r is related to no fewer than n Cs

OWL2 DL and SROIQ

- OWL2 DL corresponds to the description logic SROIQ

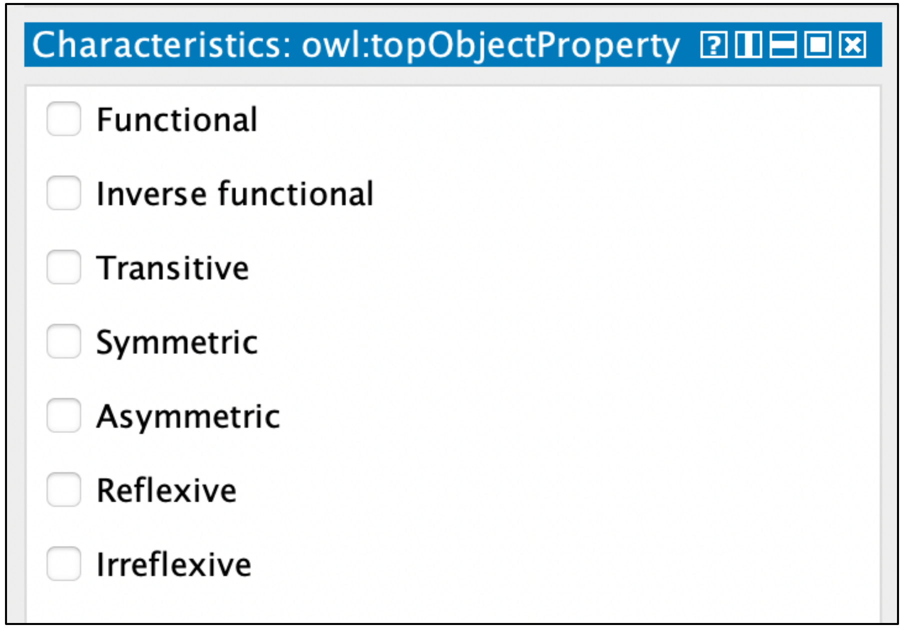
- SROIQ:
  - S = ALC
  - R = Role Axiom Extension
  - O = Nominals
  - I = Inverses
  - Q = Qualified Cardinalities

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity



Role Axiom Extension

- SROIQ’s role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity

Role Inclusion

- Allowing role inclusion axioms facilitates chaining of roles, such as:

If x owns y and y part of z then x owns z

- Role inclusion axioms have the form:

$$\mathbf{r_1 \circ \ldots \circ r_n \sqsubseteq r}$$

- But **not just any role chains** are allowed…to see why, we first define *simple* and *non-simple roles*

Role Inclusion

- Role inclusion axioms have the form:

$$\mathbf{r_1 \circ \ldots \circ r_n \sqsubseteq r}$$

- Any role inclusion axiom in which n=1, is *simple*

- For any role inclusion axiom:
  1. Every role in $\mathbf{r_1 \circ \ldots \circ r_n \sqsubseteq r}$ with n>1, is *non-simple*
  2. Every role in a simple role inclusion $\mathbf{s \sqsubseteq r}$ with a non-simple s, is *non-simple*
  3. Every r- where r is *non-simple*, is *non-simple*
  4. No other role is *non-simple*

Example

- Which of the following are *simple* and which are *non-simple*?

  - motherOf $\sqsubseteq$ parentOf
  - parentOf $\sqsubseteq$ ancestorOf
  - ancesterOf $\circ$ ancestorOf $\sqsubseteq$ ancestorOf
  - ancestorOf$^-$ $\sqsubseteq$ descendantOf$^-$

Example

1. **Every role in $r_1 \circ \ldots \circ r_n \sqsubseteq r$ with $n>1$, is *non-simple***
2. **Every role in a simple role inclusion $s \sqsubseteq r$ with a non-simple s, is *non-simple***
3. **Every r- where r is *non-simple*, is *non-simple***
4. **No other role is *non-simple***

- Which of the following are *simple* and which are *non-simple*?
  - motherOf ⊑ parentOf
  - parentOf ⊑ ancestorOf
  - ancesterOf ∘ ancestorOf ⊑ ancestorOf
  - ancestorOf− ⊑ descendantOf−

1. Every role in $r_1 \circ \ldots \circ r_n \sqsubseteq r$ with $n>1$, is *non-simple*
2. Every role in a simple role inclusion $s \sqsubseteq r$ with a non-simple s, is *non-simple*
3. Every r- where r is *non-simple*, is *non-simple*
4. No other role is *non-simple*

Example

- Which of the following are *simple* and which are *non-simple*?
  - motherOf ⊑ parentOf
  - parentOf ⊑ ancestorOf
  - ancesterOf ∘ ancestorOf ⊑ ancestorOf
  - ancestorOf– ⊑ descendantOf–

According to 1, if there is a role chain consisting of more than one occurrence of a given role, that role is non-simple

Example

1. Every role in $r_1 \circ \ldots \circ r_n \sqsubseteq r$ with $n>1$, is *non-simple*
2. Every role in a simple role inclusion $s \sqsubseteq r$ with a non-simple s, is *non-simple*
3. Every r- where r is *non-simple*, is *non-simple*
4. No other role is *non-simple*

- Which of the following are *simple* and which are *non-simple*?

  - motherOf ⊑ parentOf
  - parentOf ⊑ ancestorOf
  - ancesterOf ∘ ancestorOf ⊑ ancestorOf
  - ancestorOf− ⊑ descendantOf−

We can then conclude that ancestorOf is non-simple

Example

1. Every role in $r_1 \circ \ldots \circ r_n \sqsubseteq r$ with $n>1$, is ***non-simple***
2. Every role in a simple role inclusion $s \sqsubseteq r$ with a non-simple s, is ***non-simple***
3. Every r- where r is ***non-simple***, is ***non-simple***
4. No other role is ***non-simple***

- Which of the following are *simple* and which are *non-simple*?
  - motherOf ⊑ parentOf
  - parentOf ⊑ ancestorOf
  - ancesterOf ∘ ancestorOf ⊑ ancestorOf
  - ancestorOf− ⊑ descendantOf−

Moreover, according to 3 and the fact that ancestorOf is non-simple, it follows that ancestorOf− is also non-simple

1. Every role in $r_1 \circ \ldots \circ r_n \sqsubseteq r$ with n>1, is ***non-simple***
2. Every role in a simple role inclusion $s \sqsubseteq r$ with a non-simple s, is ***non-simple***
3. Every r- where r is ***non-simple***, is ***non-simple***
4. No other role is ***non-simple***

Example

- Which of the following are *simple* and which are *non-simple*?
  - motherOf ⊑ parentOf
  - parentOf ⊑ ancestorOf
  - ancesterOf ∘ ancestorOf ⊑ ancestorOf
  - ancestorOf– ⊑ descendantOf–

And by 2, because ancestorOf– is non-simple, so too is descendentOf–

1. Every role in $r_1 \circ \ldots \circ r_n \sqsubseteq r$ with $n>1$, is ***non-simple***
2. Every role in a simple role inclusion $s \sqsubseteq r$ with a non-simple s, is ***non-simple***
3. Every r- where r is ***non-simple***, is ***non-simple***
4. No other role is ***non-simple***

Example

- Which of the following are *simple* and which are *non-simple*?

  - motherOf ⊑ parentOf
  - parentOf ⊑ ancestorOf
  - ancesterOf ∘ ancestorOf ⊑ ancestorOf
  - ancestorOf– ⊑ descendantOf–

Lastly, by 4 all other roles are simple

Regularity

- To maintain decidability, we have to restrict any non-simple role inclusion axioms to those that have the property of being “regular”

- Without going into much detail, this just means if you’re going to have role inclusion chains, then then any combination of those roles and their inverses must exhibit a *strict partial order*

- In other words, a simple tree structure

Regularity

- To maintain decidability, we have to restrict any non-simple role inclusion axioms to those that have the property of being "regular"
- Without going into much detail, this just means if you're going to have role inclusion chains, then then any combination of those roles and their inverses must exhibit a *strict partial order*
- In other words, a simple tree structure

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity

Role Axiom Extension

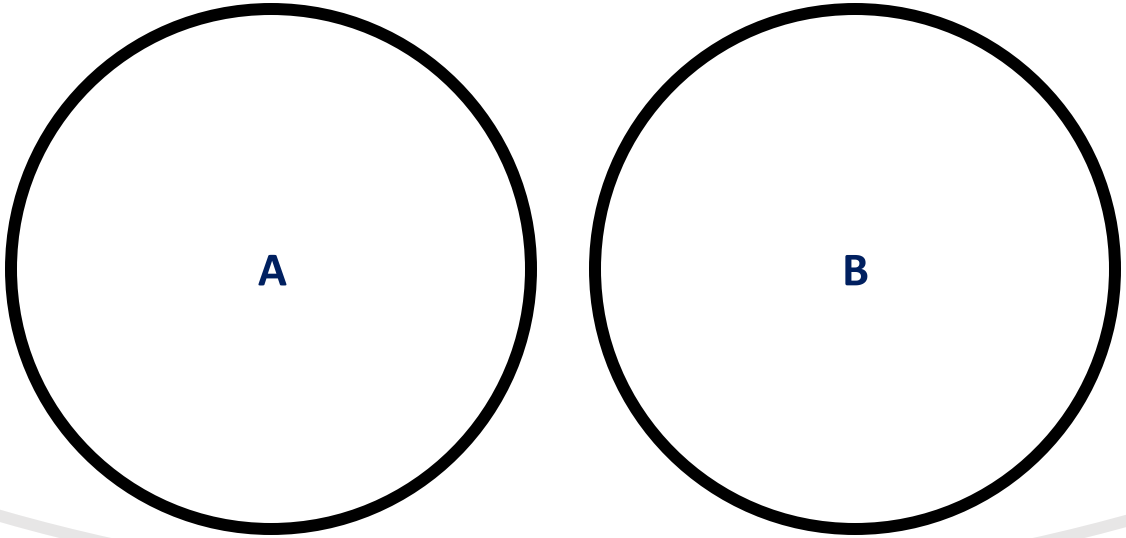
- SROIQ's role axioms include:
  - Inclusion
  - Disjointness – A and B are disjoint just in case they share no individuals

DisjointWith(A, B) = $A^I \cap B^I = \emptyset$

Role Axiom Extension

- SROIQ’s role axioms include:
  - Inclusion
  - Disjointness – A and B are disjoint just in case they share no individuals

DisjointWith(A, B) = $A^{I} \cap B^{I} = \emptyset$



Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity – If x related to y and y related to z, then x related to z

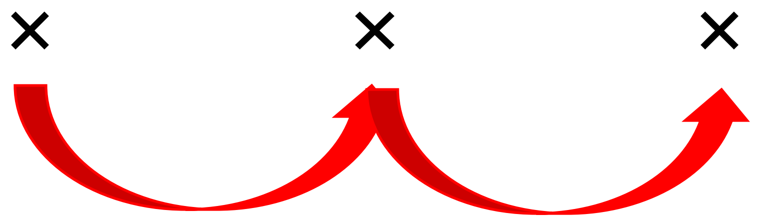
$$\mathbf{Trans(R) = R^I \circ R^I \subseteq R^I}$$

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity – If x related to y and y related to z, then x related to z

THING

$$\mathbf{Trans(R)} = R^I \circ R^I \subseteq R^I$$

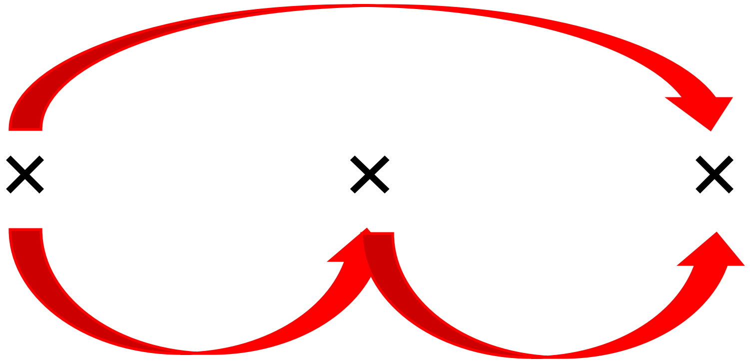


Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity – If x related to y and y related to z, then x related to z

THING

$$\mathbf{Trans(R) = R^{I} \circ R^{I} \subseteq R^{I}}$$



Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry – If x related to y then y related to x

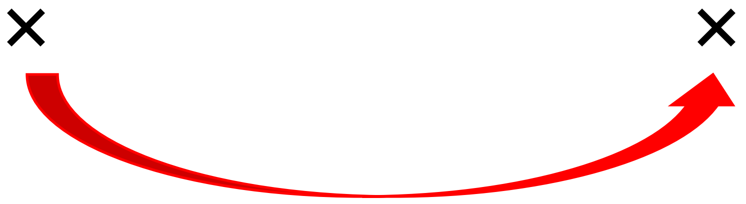
$$(x,y) \in R^I \Rightarrow (y,x) \in R^I$$

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry – If x related to y then y related to x

THING

$$(x,y) \in R^I \Rightarrow (y,x) \in R^I$$

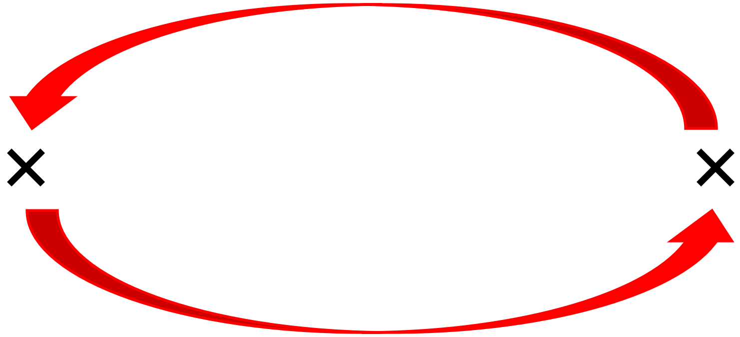


Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry – If x related to y then y related to x

THING

$$(x,y) \in R^I \Rightarrow (y,x) \in R^I$$



Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry – If x is related to y then y is not related to x

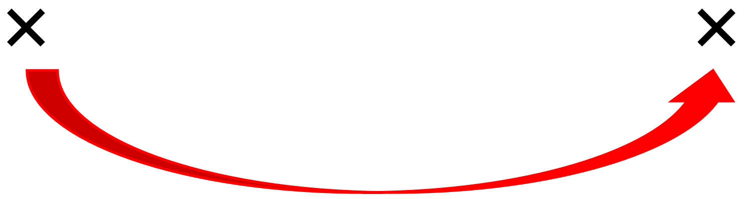
$$(x,y) \in R^I \Rightarrow (y,x) \notin R^I$$

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry – If x is related to y then y is not related to x

THING

$$(x,y) \in R^I \Rightarrow (y,x) \notin R^I$$

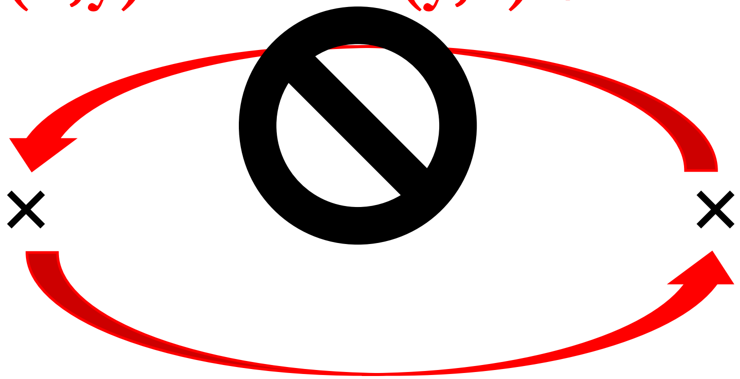


Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry – If x is related to y then y is not related to x

THING

$$(x,y) \in R^I \Rightarrow (y,x) \notin R^I$$



Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity

Role Axiom Extension

- SROIQ’s role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity – For all x, x is related to x

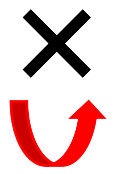
$$\{(x,x) \mid x \in \text{Domain}\} \subseteq R^I$$

Role Axiom Extension

- SROIQ’s role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity – For all x, x is related to x

THING

$$\{(x,x) \mid x \in \text{Domain}\} \subseteq R^I$$



Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity – There is no x such that x is related to x

$$\{(x,x) \mid x \in \mathrm{Domain}\} \cap R^{I} = \emptyset$$

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity – There is no x such that x is related to x

THING

$$\{(x,x) \mid x \in \text{Domain}\} \cap R^I = \emptyset$$