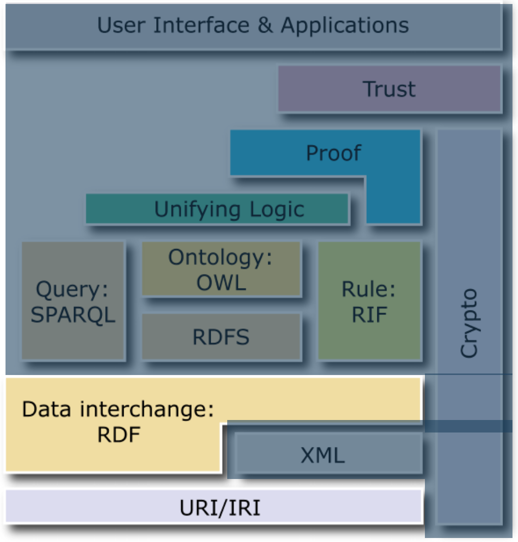
RDF

Resource Description Framework



A bit of history

- RDF was created by the W3C in 1999 as a format for structuring metadata about web pages (title, author, modification date, …)
- It has been designed to allow information to be exchanged without loss of meaning
- It became a general data format/model for data in general (rather than only metadata)
  - Proposed in 2001 as the model for data exchange in the Semantic Web
  - Nowadays there are quite big data stores in RDF
  - **The** today's data model of Semantic Technologies!

Why not XML?

- XML provides a uniform framework for interchange of data (and metadata)
- But does not provide any means of talking about the semantics (meaning) of data. E.g.:
  - there is no intended meaning associated with the nesting of tags
  - There is no (processable) intended meaning of each tag
- It is up to each application to interpret the nesting.

Nesting of tags in XML

- Consider

```
<course name="Data Modelling">
   <lecturer>Carlos Damásio</lecturer>
</course>
```

- versus

```
<lecturer name="Carlos Damásio">
   <teaches>Data Modelling</teaches>
</lecturer>
```

- The nesting are opposite, but the meaning is the same.
  - There should be a way of attributing meaning without compromising to a particular nesting
- *How would it look in a relational database?*

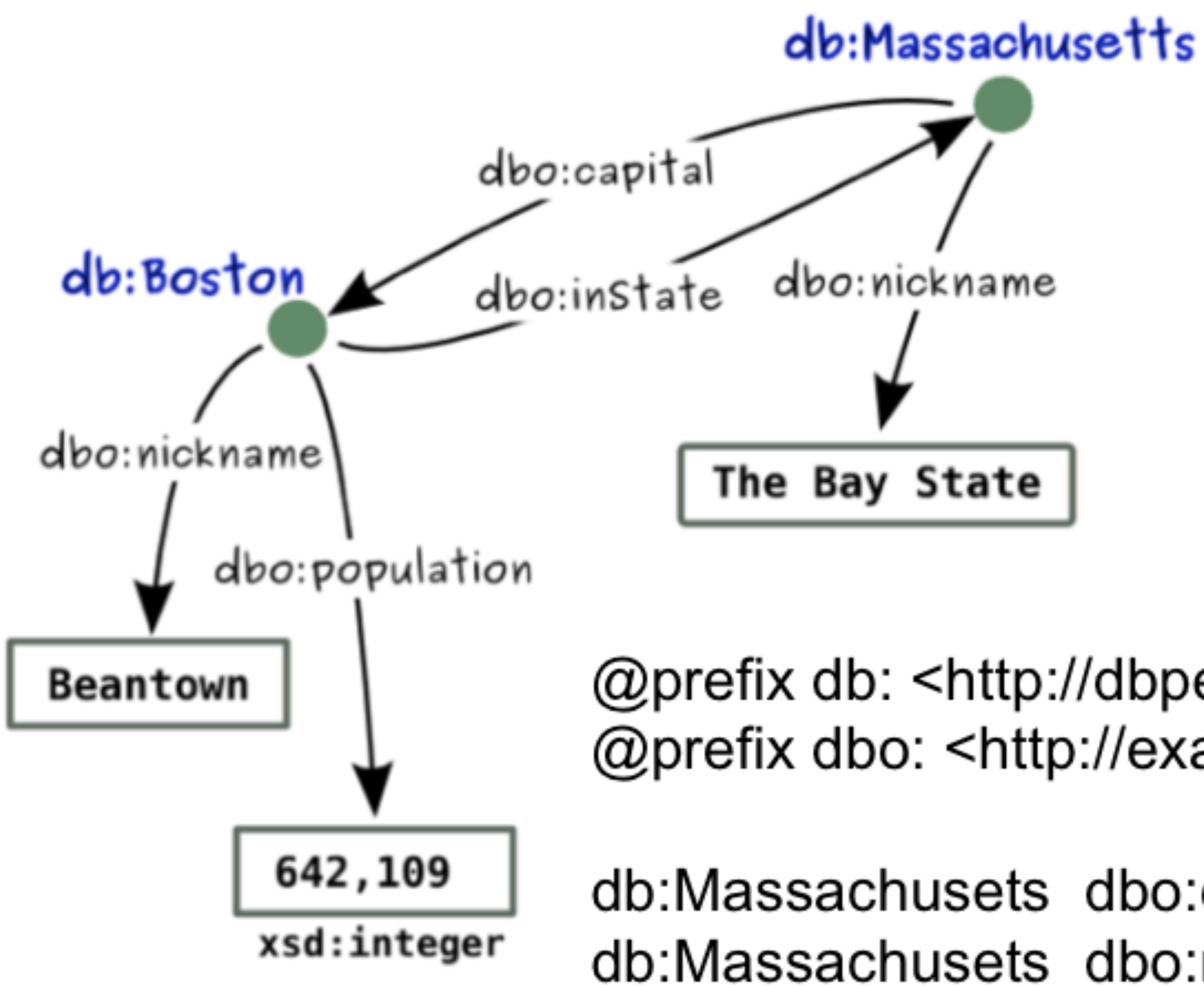
Basic Ideas of RDF

- Represent the meaning independently of the syntax
- Basic building block: object-attribute-value triple
  - It is called a statement
  - In the example, object is Carlos Damásio, attribute is lectures, value is Data Modelling
- Fundamental concepts in RDF are:
  - Resources (like the object and value above)
  - Properties (like the attribute above)
  - Statements (the triple above)

RDF syntax

- With these basic ideas, an RDF dataset is a labeled directed graph (i.e. a labeled property graph)
  - Modelling data as a graph, rather than modelling as a set of relations (in databases), or as a tree (in XML)
- There are several syntaxes for writing such a graph:
  - N3 (Notation 3) comprehensive formalism
  - N-Triples: fraction of N3
  - Turtle (Terse RDF Triple Language)
- A standard encoding of RDF into XML and JSON have also been defined
  - Facilitates interoperability of tools
  - RDF should **not** be confused with their syntactical representation!

First RDF example (in Turtle)




```
@prefix db: <http://dbpedia.org/resource/>
@prefix dbo: <http://example.org/terms/>

db:Massachusets  dbo:capital     db:Boston .
db:Massachusets  dbo:nickname    "The Bay State" .
db:Boston        dbo:inState     db:Massachusets .
db:Boston        dbo:nickname    "Beantown" .
db:Boston        dbo:population  "642,109"^^xsd:integer .
```

Resources

- A resource is just any "thing", an object we want to refer to
  - E.g. an author, a book, a place, a person, a hotel, etc
- In RDF, every resource has a unique identifier
  - Unique identifiers are crucial to disambiguate resources!
  - On the Web we already have unique identifiers, and RDF just uses them
- URIs/IRIs will be used as identifiers

URLs, URIs, and IRIs

- URL (Uniform Resource Locator)
  - Just like a web address (e.g. http://www.fct.unl.pt)
  - Use to locate a resource in the Web
- URI (Uniform Resource Identifier)
  - Looks like a URL, but does not need to identify a Web resource (it can be a person, a book, etc)
- IRI (Internationalized Resource Identifier)
  - Like URI, but using Unicode (UTF) instead of ASCII
    - E.g. http://ヒキワリ.ナットウ.ニホン
  - IRIs can be translated to URIs
    - I'll spare you the details of such a conversion.

URI syntax

`scheme:[//authority]path[?query][#fragment]`

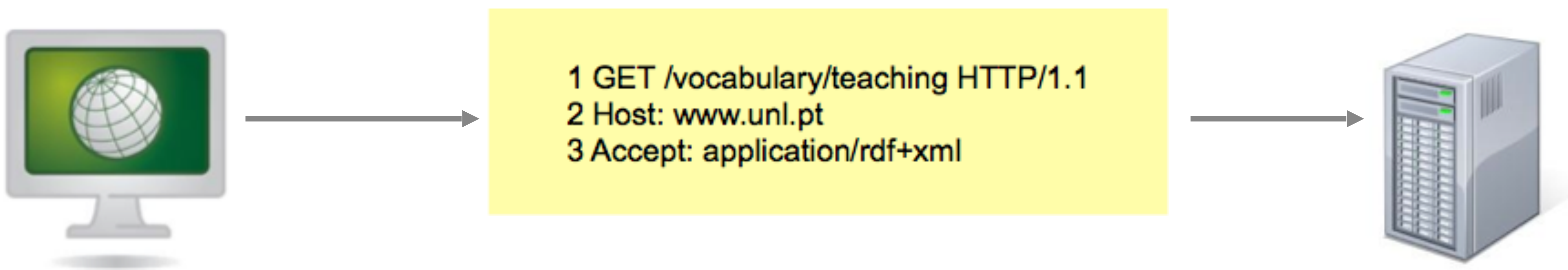
- **`scheme`**: type of URI, e.g. http, ftp, mailto, …
- **`authority`**: typically a domain name, e.g. fct.unl.pt
- **`path`**: just like a directory's path
- **`query`**: optional, usually for parameters e.g. in web services
- **`fragment`**: optional, used to refer to a part of a document

Which URIs to use?

- According to Linked Data principles always use http addresses
- How to make real world objects dereference-able?
  - Associate to URLs
  - Hash URIs or 303 URIs (See other)
- The techniques depend on the behaviour of HTML servers, and are fully explained in *Linked Data: Evolving the Web into a Global Data Space* (http://linkeddatabook.com)

Hash URI request

- Suppose we want to obtain the university definition of Teacher and Student
  - http://www.unl.pt/vocabulary/teaching#Teacher
  - http://www.unl.pt/vocabulary/teaching#Student



Hash URI request

- Suppose we want to obtain the university definition of Teacher and Student
  - http://www.unl.pt/vocabulary/teaching#Teacher
  - http://www.unl.pt/vocabulary/teaching#Student

```
HTTP/1.1 200 OK
Content-Type: application/rdf+xml;charset=utf-8


<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<rdf:Description rdf:about="http://www.unl.pt/vocabulary/teaching#Teacher">
<rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class" />
</rdf:Description>
<rdf:Description rdf:about="http://www.unl.pt/vocabulary/teaching#Student">
<rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class" />
</rdf:Description>
...
```

FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA

Properties

- Properties describe relations between resources
  - E.g. "written by", "has age", "has title", etc.
- Each property is itself also a resource
  - So, properties are also identified by URIs
- Advantages of using identifying URIs:
  - A global, worldwide, unique naming scheme
  - Reduces the homonym problem of distributed data representation
  - (Basically, with URIs everything is guaranteed to be uniquely identified, by a key)

Statements

- Statements assert properties of resources
  - Relate resources via properties
- In RDF, a statement is an object-attribute-value triple
  - It consists of a resource, a property, and a value
- They can be seen as binary predicates: attribute(object,value)
- Values can be resources or literals
  - Literals are just atomic values (e.g. strings), that don’t need to have a URI.

Representation of statements

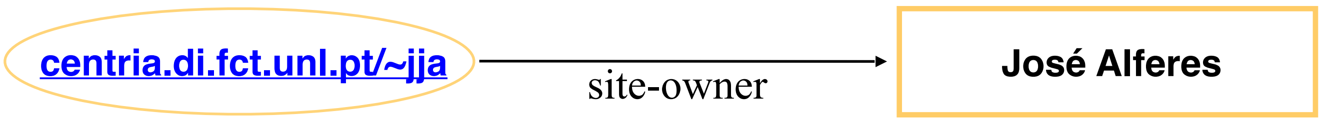
- A statement can be viewed as:
  - A triple (Object, Property, Value)
    or (Subject,Predicate,Object)
  - An arc connecting two nodes in a graph
  - A piece of XML code, representing the triple
- Accordingly, an RDF document can be viewed as:
  - A set of triples
  - A graph (semantic net)
- Do not confuse with its serialisation, namely:
  - An XML document with the triples represented according to a given predefined syntax
  - A JSON-LD document
  - N-Triples, N3 or Turtle formats

Not all arcs are valid

- In an RDF graph, RDF triples:
  - **Subject** must be: URI/IRI or blank node
  - **Predicate** must be: URI/IRI
  - **Object** can be anything (URI/IRI, blank node, literal)

- Some extensions only limit predicates:
  - **Subject** can be anything
  - **Predicate** must be: URI/IRI or blank node
  - **Object** can be anything

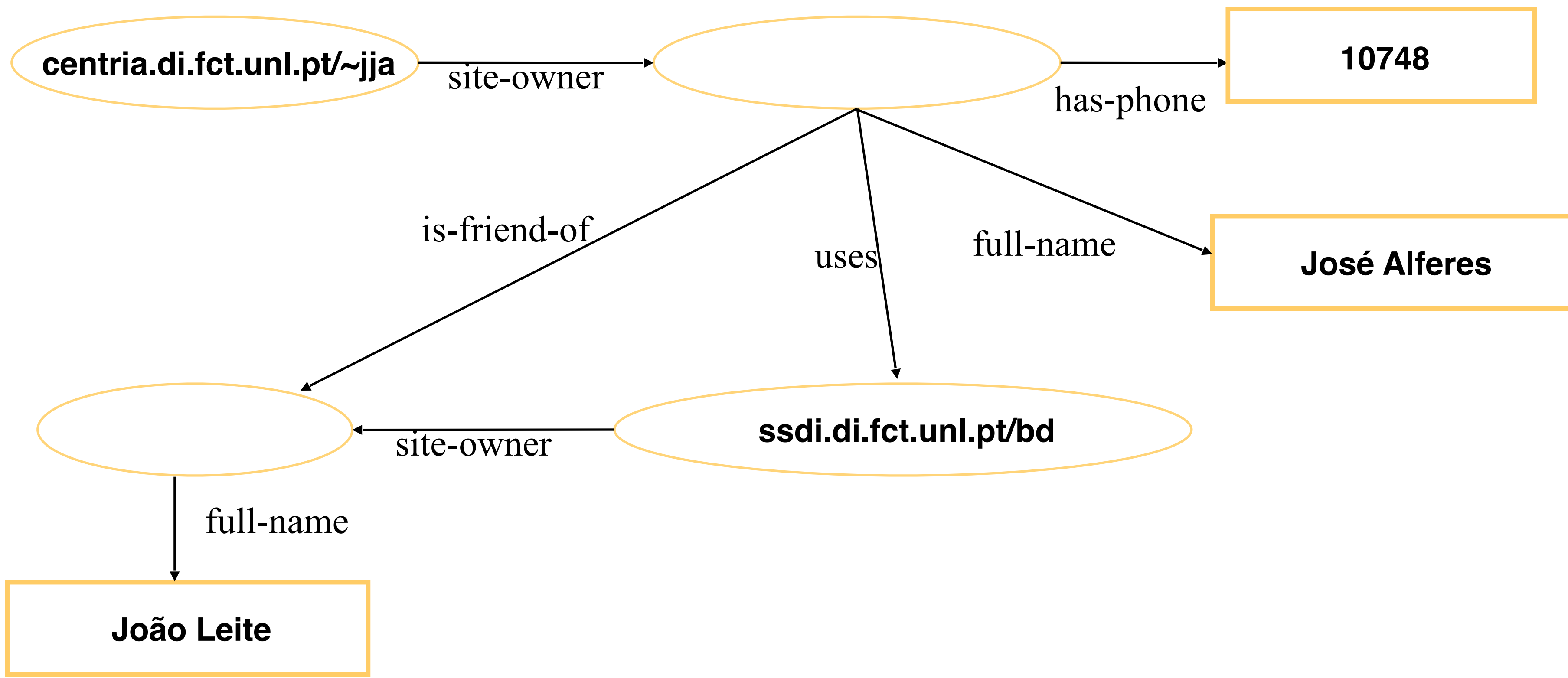
Representing triples in a graph



- This piece of a graph is representing the triple:
  - (centria.di.fct.unl.pt/~jja, site-owner, "José Alferes")
  - Or the predicate
    site-owner(centria.di.fct.unl.pt/~jja, "José Alferes")

  (these things are not proper URIs, but for the sake of the example…)
- An RDF document can be seen as a directed graph with labeled nodes and arcs
  - from the resource (the subject of the statement)
  - to the value (the object of the statement)
- Known in AI as a *semantic net*

An example of RDF graph



- The owner of the site …~jja is someone named José Alferes who has phone 10748 and is a friend of someone called João Leite, who owns …/bd web page; José Alferes uses this web page

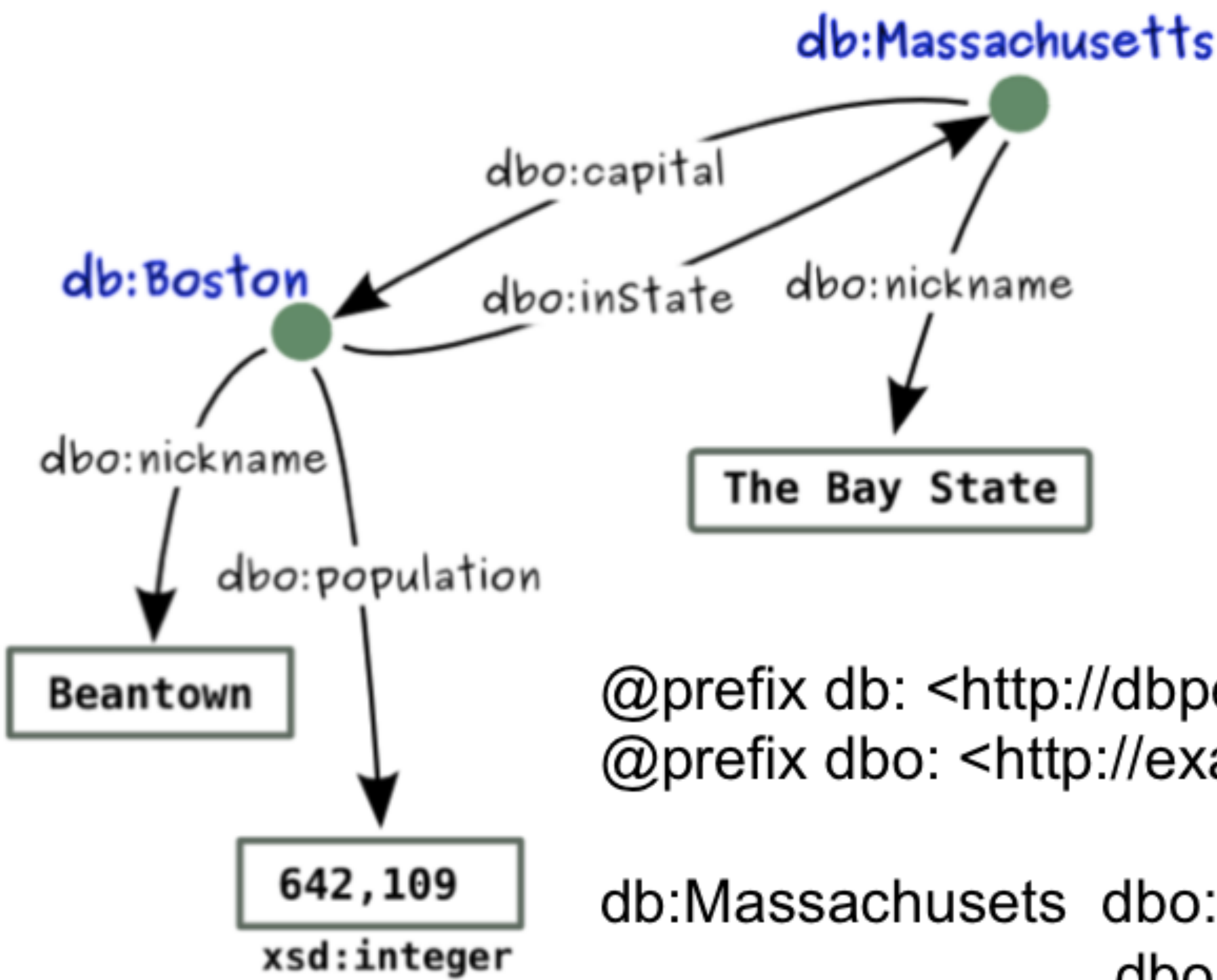
RDF as a data model

- A schema-less data model
  - Based on a graph
  - With unambiguous identifiers
  - With named relations between pair of resources
- The graph structure trivialises merging data with shared identifiers
- Triples act as least common identifier for expressing data
- Eases "navigation" through data in different locations

A bit of (Turtle) syntax

- Turtle was defined to be a simple syntax for RDF, and is standardised by W3C since February 2014 (see http://www.w3.org/TR/turtle/)
  - Triples are directed lists: Subject Property Object
  - URI are in <angle brackets>
  - End with "."
  - White spaces are ignored
  - Prefixes (simple string concatenation)
  - Grouping of triples with the same subject with ;
  - Grouping of triples with the same subject and property with ,

Turtle example



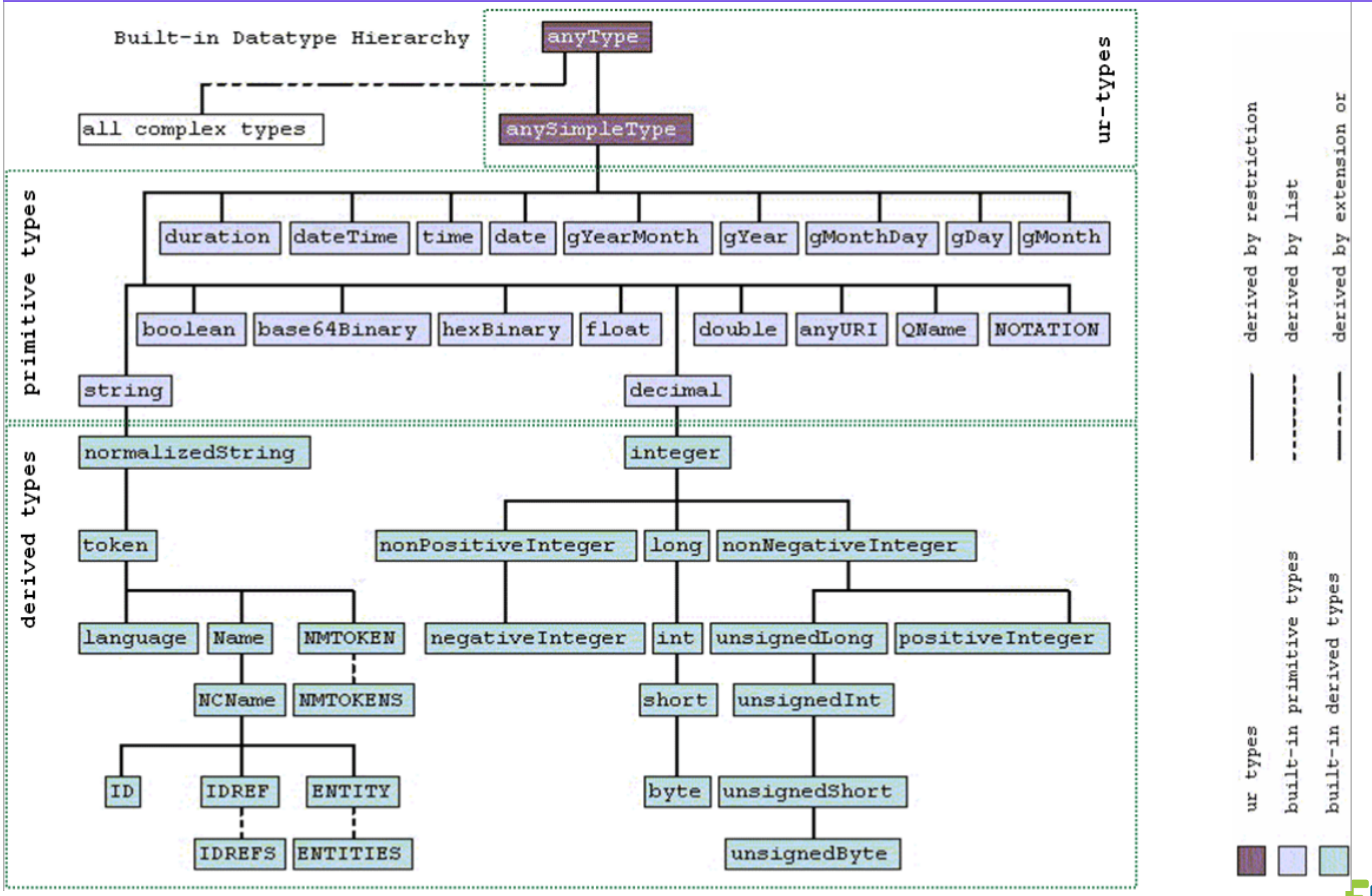

```
@prefix db: <http://dbpedia.org/resource/>
@prefix dbo: <http://example.org/terms/>

db:Massachusets  dbo:captial      db:Boston  ;
                  dbo:nickname     “The Bay State”  .
db:Boston         dbo:inState      db:Massachusets ;
                  dbo:nickname     “Beantown”  ;
                  dbo:population   “642,109”^^xsd:integer  .
```

Literals

- Represent data values
  - Encoded as strings but can be interpreted by means of datatypes
    - Literals without any type are treated as strings
  - A literal without a type is called a plain literal, and may have a language tag
- Datatypes are borrowed from XML Schema (XSD)
  - RDF does not require an implementation of datatypes, though systems usually implement most of XSD datatypes
- Some examples
  - Typed literals
    - `"Bay State"^^xsd:string` or `"604109"^^xsd:integer`
  - Plain literals
    - `"Germany"` or `"Deutschland"@de`

XSD datatypes



FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA

Type definition

- Datatypes can be defined by the user, as in XML
  - New derived simple types are derived by restriction
  - Complex types based on enumerations, unions, and lists are also possible

```
<xsd:schema …>
  <xsd:simpleType name ="humanAge">
    <xsd:restriction base="integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="150"/>
    </xsd:restriction>
  </xsd:simpleType>
  …
</xsd:schema>
```

- More, in other courses (e.g. BD, CTXML)

N-ary predicates

- RDF statements can be viewed as facts for binary predicates
- And what about predicates with more than 2 arguments?
  - Example: John has done course1 with grade 12 and course2 with grade 15

```
@prefix ex:<http://example.org/>
ex:John ex:hasCourse "course1 with 12",
                     "course2 with 15".
```

N-ary predicates

- RDF statements can be viewed as facts for binary predicates
- And what about predicates with more than 2 arguments?
  - Example: John has done course1 with grade 12 and course2 with grade 15

```
@prefix ex:<http://example.org/>
ex:John ex:hasCourse "course1";
        ex:withGrade "12";
        ex:hasCourse "course2";
        ex:withGrade "15".
```

N-ary predicates

- Solution: use auxiliary nodes!

```
@prefix ex:<http://example.org/>
ex:John ex:hasCourse ex:onecourse;
        ex:hasCourse ex:anothercourse.
ex:onecourse ex:course "course1";
             ex:withGrade "12".
ex:anothercourse ex:course "course2";
                 ex:withGrade "15".
```

- These auxiliary nodes are unique, and only used locally
- So, why do they have to have a unique global identifier?

Blank nodes

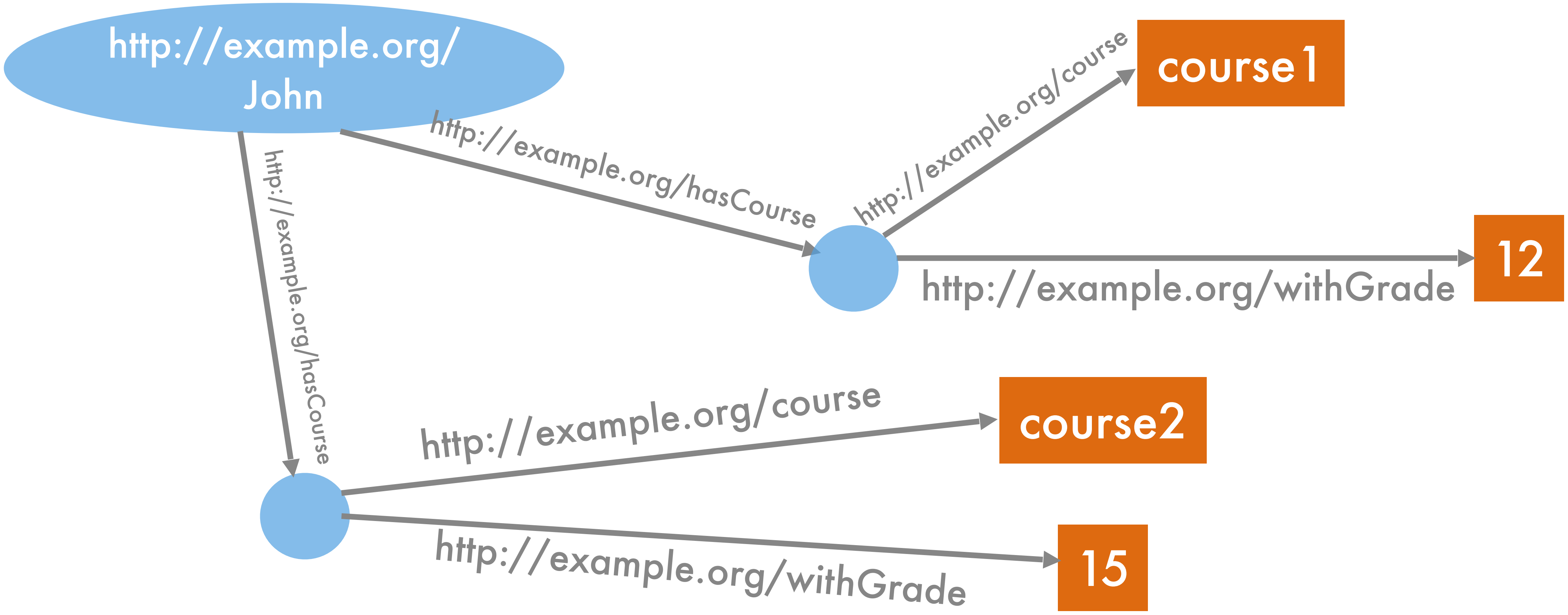
- bnodes are used for resources that do not need to be universally identified
- Just like existentially quantified variable

```
@prefix ex:<http://example.org/>
ex:John ex:hasCourse _:id1;
        ex:hasCourse _:id2.
_:id1 ex:course "course1";
      ex:withGrade "12".
_:id2 ex:course "course1";
      ex:withGrade "12".
```

```
@prefix ex:<http://example.org/>
ex:John ex:hasCourse
     [ex:course "course1";ex:withGrade "12"],
     [ex:course "course2";ex:withGrade "15"].
```

Blank nodes

```
@prefix ex:<http://example.org/>
ex:John ex:hasCourse
      [ex:course "course1";ex:withGrade "12"],
      [ex:course "course2";ex:withGrade "15"].
```



FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA

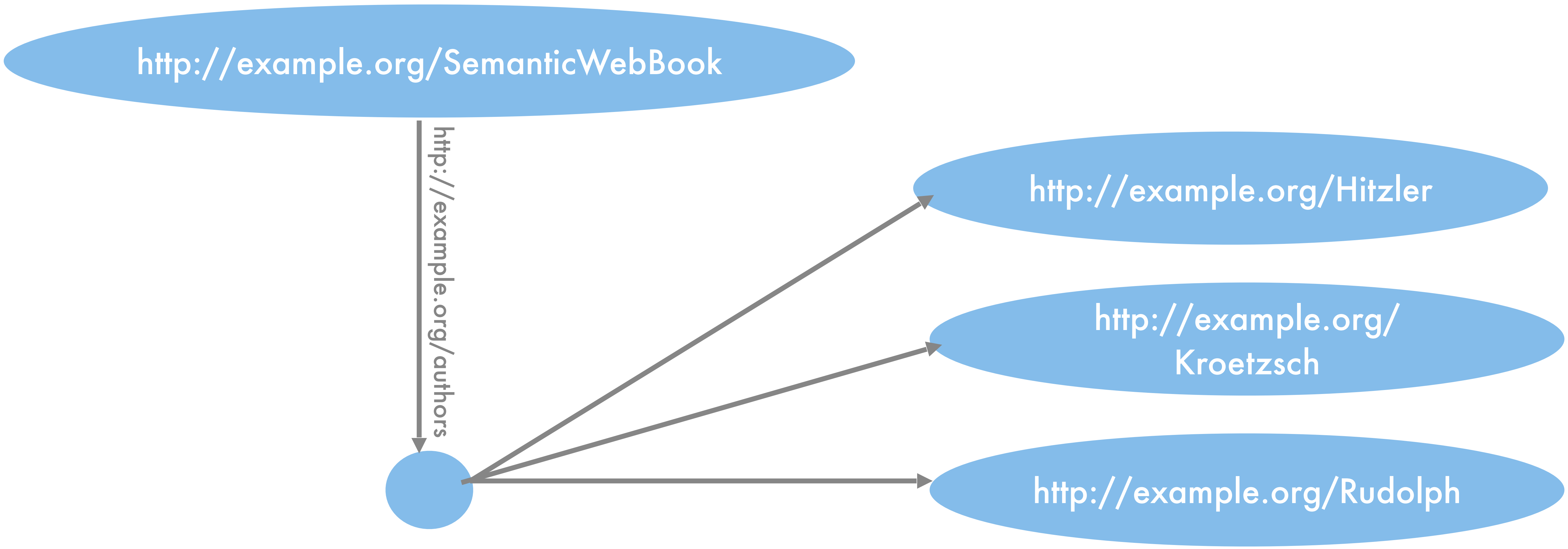
Containers

- Use when an argument has a set (or sequence) of values
  - E.g. authors of a book, lectures of a course, …
- Closed containers or `Collection`
  - no further elements can be added
- Open containers
  - Lists (order matters) - `rdf:Seq`
  - Bags (order doesn't matter) - `rdf:Bag`
  - Alternatives - `rdf:Alt`
  - *Why not sets?*

Collections

- Use blank nodes

```
@prefix ex:<http://example.org/>
ex:SemanticWebBook ex:authors
   (ex:Hitzler ex:Kroetzsch ex:Rudolph).
```



Open containers

- Use blank nodes and `rdf:type`
- For lists use `rdf:first`, `rdf:rest`, and `rdf:nil`
- For bags use `rdf:li`
- For alternatives use `rdf:_1`, `rdf:_2`, etc

```
@prefix cb:<http://clubes.pt/>
@prefix rdf:<http://www.w3.org/1999/02/22-
rdf-syntax-ns#>
cb:podium _:id1.
_:id1 rdf:type rdf:Seq;
      rdf:first "Benfica";rdf:rest _:id2.
_:id2 rdf:type rdf:Seq;
      rdf:first "Porto";  rdf:rest _:id3.
_:id3 rdf:type rdf:Seq;
      rdf:first "Sporting";rdf:rest rdf:nil.
```

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Reification

- In RDF it is possible to make statements about statements. E.g.
  - Carlos believes that José Alferes is the creator of http://ssdi.di.fct.unl.pt/rsw
  - Such statements can be used to describe belief or trust in other statements
- They amount to considering statements themselves as resources that can then be referenced
- For that, one needs to assign a unique identifier to each statement

Reifying statements

- Introduce an auxiliary node (e.g. a bnode)
- Relate to it each of the 3 parts of the original statement through `rdf:subject`, `rdf:predicate` and `rdf:object`
- **Reified statements** `rdf:type` **is** `rdf:Statement`

```
ex:Carlos ex:believes
    [rdf:type rdf:Statement;
     rdf:subject ex:JoseAlferes;
     rdf:predicate ex:creator;
     rdf:object <http://ssdi.di.fct.unl.pt/rsw>].
```

Critical view of reification

- The mechanism is quite powerful
  - But it appears misplaced in a simple language like RDF
- Making statements about statements introduces a level of complexity that is usually not necessary for a basic layer of the Semantic Web
- It may make sense in higher layers, providing richer representation capabilities
  - It is confusing, and complex, to have it in the basic layer of RDF

RDF in practice

- Today there are lots of RDF tools
  - We will try some in the labs
- There are libraries for plenty of programming languages
- Commercial systems like Oracle support it
- It is the basis for other data formats. E.g.:
  - RSS feeds (original name was RDF Site Summary)
  - Adobe XMP (eXtensible Metadata Platform)
  - SVG (Scalable Vector Graphics)

RDF in practice

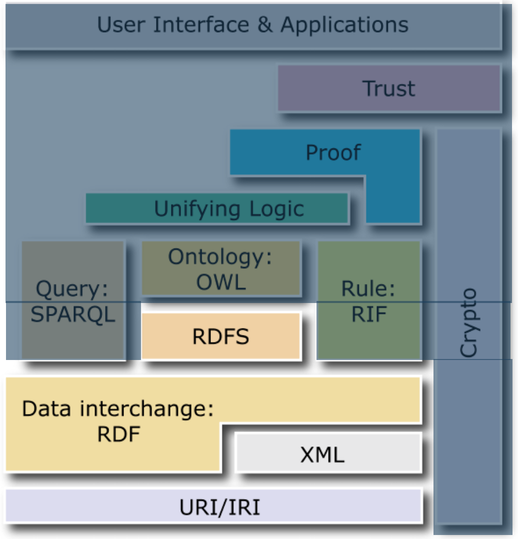
- There are freely available systems to work with large datasets (RDF- or Triple-Stores)
- And there are quite large datasets!
  - E.g. bio2rdf has 5 billion triples, dbpedia has 3 billion, geonames has 100 million, dblp has 88 millions
- linkeddata.org is quite a large "database" to play with! In 2011 it already looked like:

Domain	Number of datasets	Triples	%	(Out-)Links	%
Media	25	1,841,852,061	5.82 %	50,440,705	10.01 %
Geographic	31	6,145,532,484	19.43 %	35,812,328	7.11 %
Government	49	13,315,009,400	42.09 %	19,343,519	3.84 %
Publications	87	2,950,720,693	9.33 %	139,925,218	27.76 %
Cross-domain	41	4,184,635,715	13.23 %	63,183,065	12.54 %
Life sciences	41	3,036,336,004	9.60 %	191,844,090	38.06 %
User-generated content	20	134,127,413	0.42 %	3,449,143	0.68 %
	295	31,634,213,770		503,998,829	

FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA

RDF Schema

Defining schemas for RDF data



Why a schema language

- Like in databases, to understand the data one needs some formalisation of what it is about
  - what classes, with each types of attributes, with what relationship between classes, etc
  - this is especially important when the data is highly distributed, and provided in a collaborative way
- RDF provides a data model to state propositions about individual resources
- In a schema language we need to state propositions about generic sets of individuals
  - and also logical interdependencies between them

RDF Schema (RDFS)

- RDF is a universal language that lets users describe resources
  - It does not assume any meaning for the vocabulary used
  - It does not assume, nor does it define semantics of any particular application domain

- RDFS allows for specifying terminological knowledge, that RDF data can refer to
  - It is a language for describing (simple) semantics of arbitrary RDF
  - Uses RDF itself (with dedicated vocabulary)

- RDFS is part of RDF's W3C recommendation
  - `xmlns:rdfs = "http://www.w3.org/200/01/rdf-schema#"`
  - Notice: The relation between RDF Schema and RDF is not the same as that between XML Schema and XML

RDFS Basic Ideas

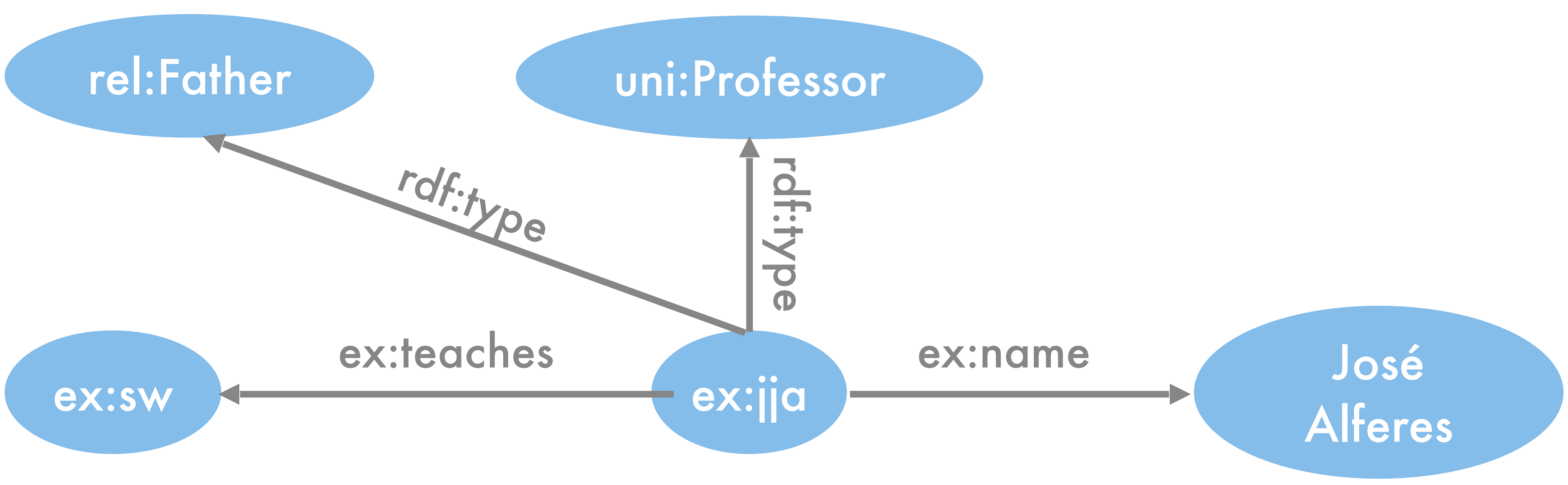
- Schemas are specified with:
  - Classes and Properties
  - Class (and Property) hierarchies and inheritance
  - Property Restriction (e.g. stating that authors of a book must be persons)
- Classes and Instances
  - Instances (defined in RDF) refer to concrete individual objects (e.g. me, this book)
  - Classes denote sets of individuals sharing some properties (e.g. persons, books)
  - In RDFS classes are also seen as objects (with URIs)
  - The relationship between instances and classes is made via special attribute `rdf:type` of the instance
    - amalgamating data and meta-data

First Schema example

```
ex:jja ex:name "José Alferes";
       ex:teaches ex:sw;
       rdf:type uni:Professor.
```

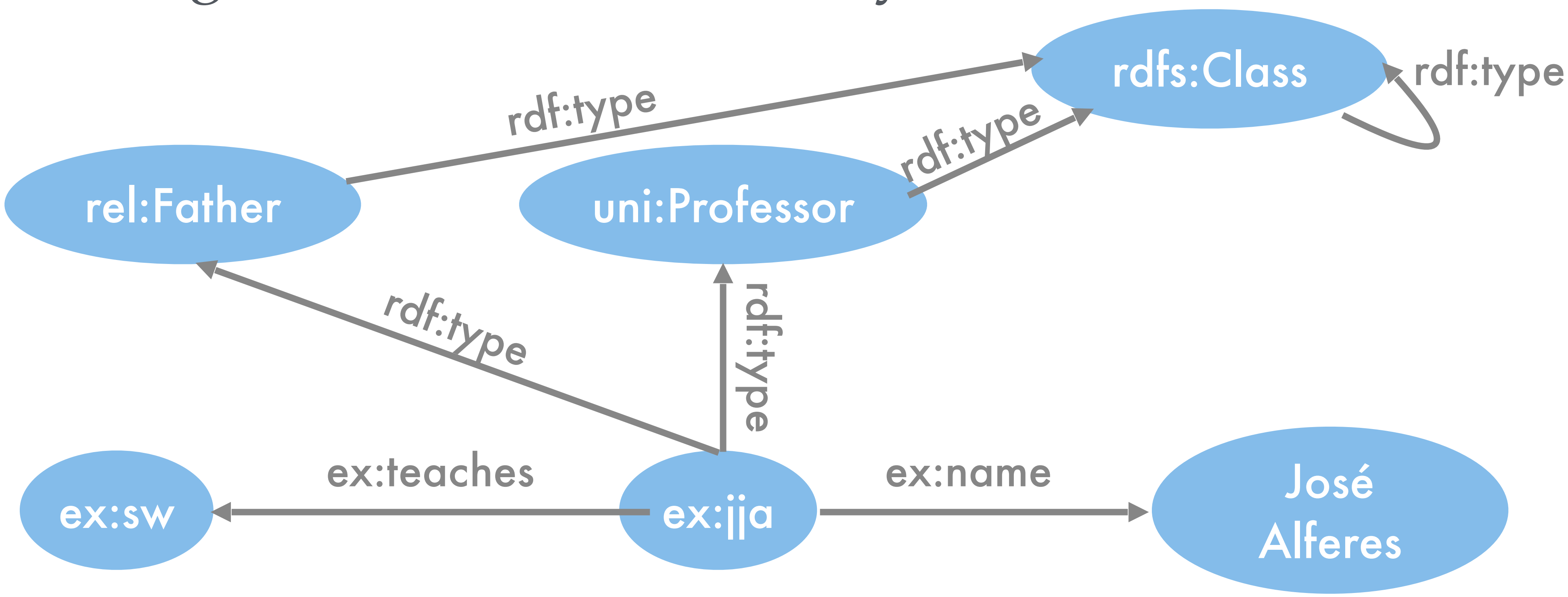
- The last statement characterises José Alferes as an instance of class professor.
  - An individual can belong to more than one class. E.g.

```
ex:jja rdf:type rel:Father.
```



The class of all classes

- In the example `ex:Professor` and `ex:Father` are objects whose type is **`rdfs:Class`**
- **`rdfs:Class`** is also of type **`rdfs:Class`** (the class of all classes)
  - The triple `rdfs:Class rdf:type rdfs:Class` virtually belongs to all datasets (already some kind of semantics)



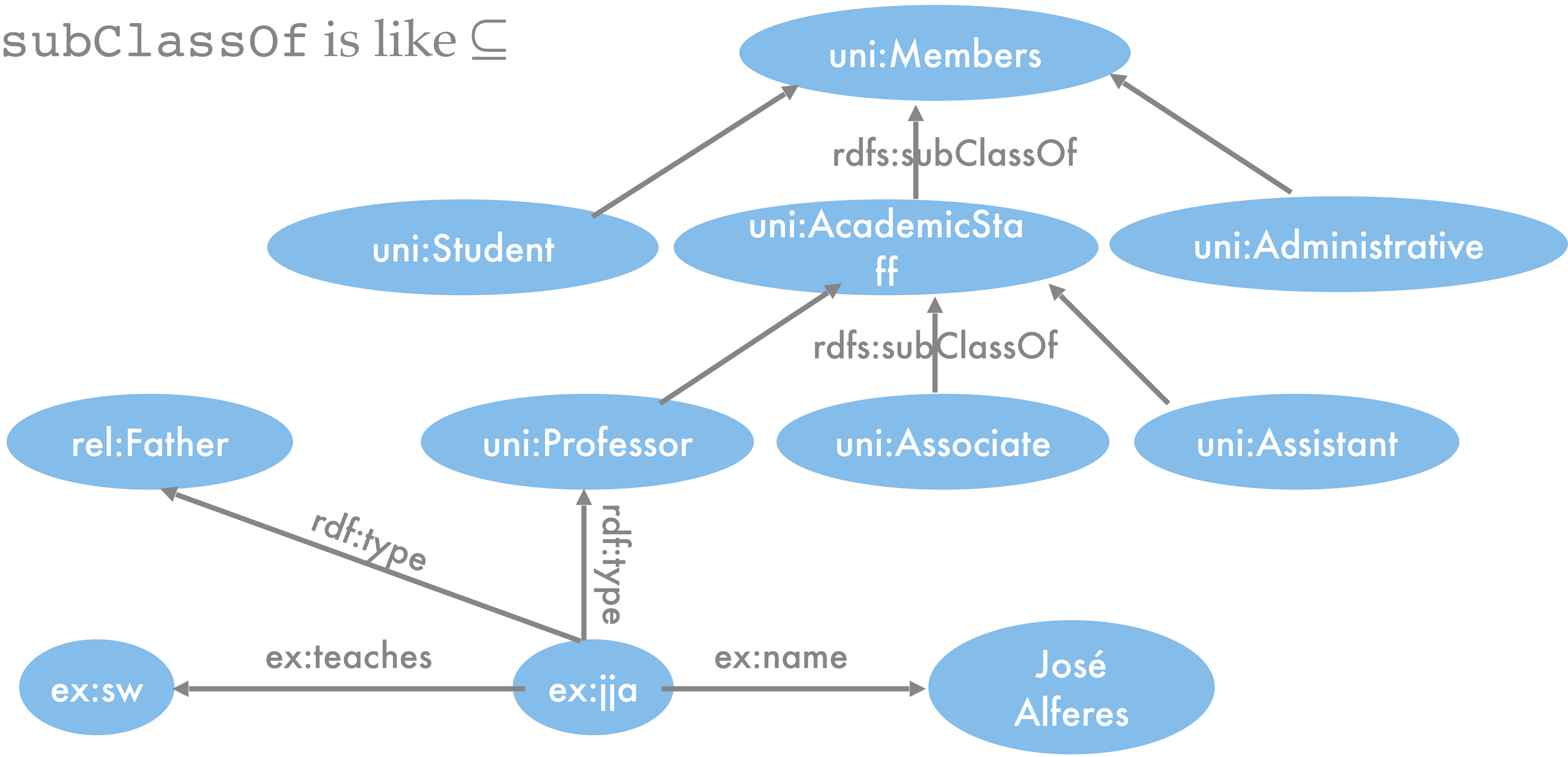
Class Hierarchies

- Suppose we are searching for all Academic Staff
  - jja should be considered
- This can be done by having a general statement, saying that all professors are academic staff (with `rdfs:subClassOf`)
- `rdfs:subClassOf` is
  - a property
  - it is reflexive and transitive
  - can be used to enforce that two classes have the same extension (i.e. the same set of individuals)
    - with `A rdfs:subClassOf B` and `B rdfs:subClassOf A`

Simple taxonomy

`rdf:type` is like $\in$

`rdfs:subClassOf` is like $\subseteq$



Properties

- Properties specify in which ways two resources are related
  - usually appear in the predicate position of triples
  - mathematically represented as binary relations (sets of pairs)
  - their `rdf:type` is `rdf:Property`
- Hierarchical relationships may also be defined for properties
  - E.g., "teaches" is a sub-property of "isInvolvedIn"
    - If a professor P teaches course C, then P is involved in C
- P `rdfs:subPropertyOf` Q, if Q(x,y) is true whenever P(x,y) is true

Property restriction

- Defined by `rdfs:domain` and `rdfs:range`
- Restrict the kind of resources that can be related via the property
  - E.g. a property "teaches" only makes sense if it is relating an academic staff to a course
  - It can also be used to declare datatypes for literals

```
ex:teaches rdfs:domain uni:AcademicStaff;
           rdfs:range uni:Course.
ex:name    rdf:range xsd:string.
```

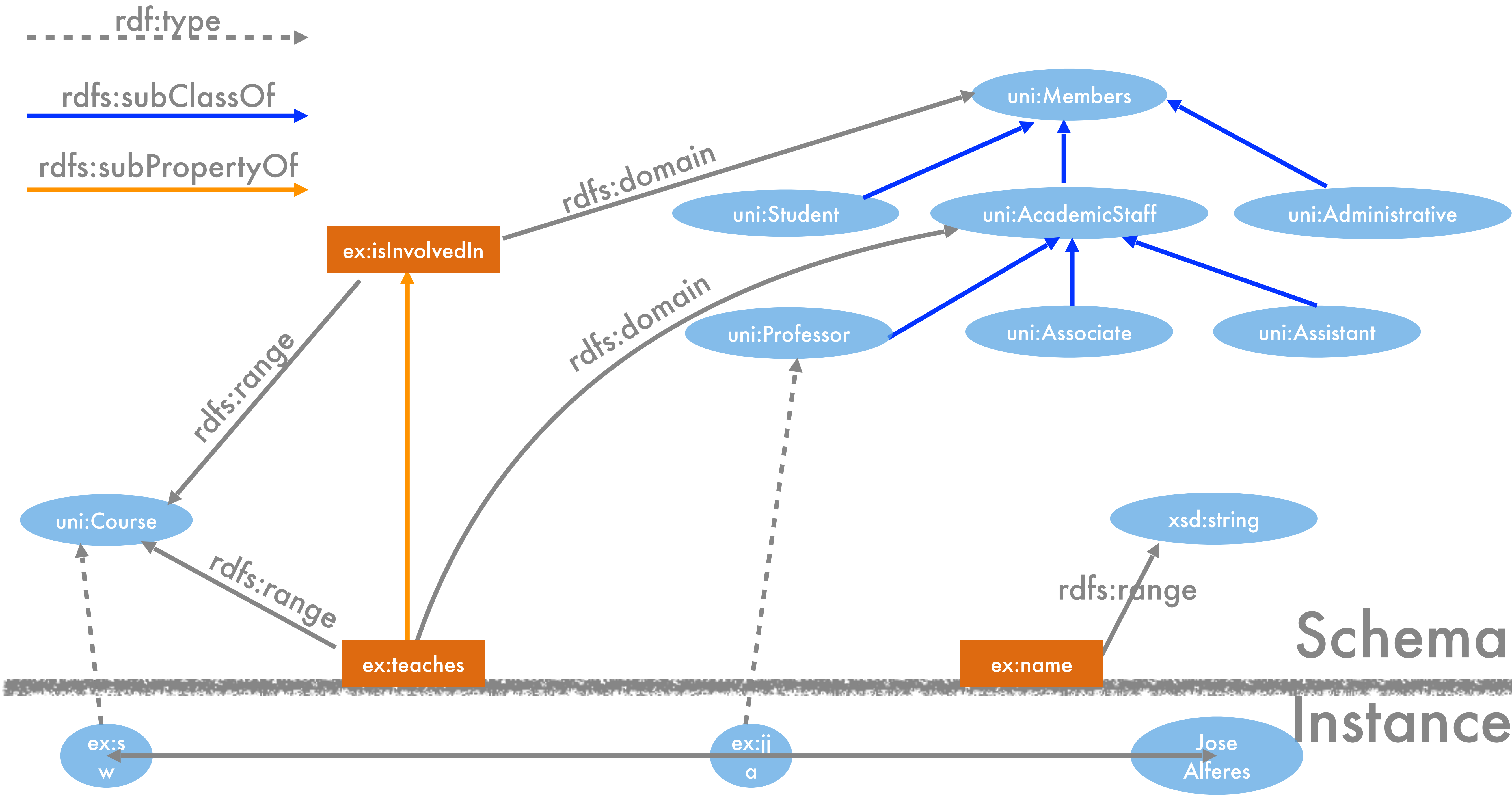
Property restriction

- Property restriction are interpreted globally and conjunctively
  - E.g.

```
ex:teaches rdfs:domain uni:Professor;
           rdfs:domain uni:Associate.
ex:cd      rdf:teaches ex:sbd.
```

  - entails that `ex:cd` is both a `uni:Professor` and a `uni:Associate`. In this case (and in general) this is not what is wanted!
  - When designing the schema, choose the most general class for domain and range!

An example schema



An example schema

```
ex:jja ex:teaches ex:sw.

ex:jja rdf:type uni:Professor.
ex:sw  rdf:type uni:Course.

uni:Professor rdfs:subClassOf uni:AcademicStaff.
uni:Associate rdfs:subClassOf uni:AcademicStaff.
uni:Assistant rdfs:subClassOf uni:AcademicStaff
uni:AcademicStaff rdfs:subClassOf uni:Member.
uni:Student rdfs:subClassOf uni:Member.
uni:administrative rdfs:subClassOf uni:Member.
ex:teaches rdf:domain uni:AcademicStaff;
           rdf:range  uni:Course;
           rdf:subPropertyOf ex:isInvolvedIn.
ex:isInvolvedIn rdf:domain uni:Member;
           rdf:range  uni:Course;
ex:name rdf:range xsd:string.
```

And also...

```
ex:jja ex:teaches ex:sw.

ex:jja rdf:type uni:Professor.
ex:sw  rdf:type uni:Course.

uni:Professor rdfs:subClassOf uni:AcademicStaff.
uni:Associate rdfs:subClassOf uni:AcademicStaff.
uni:Assistant rdfs:subClassOf uni:AcademicStaff
uni:AcademicStaff rdfs:subClassOf uni:Member.
uni:Student rdfs:subClassOf uni:Member.
uni:administrative rdfs:subClassOf uni:Member.
ex:teaches rdf:domain uni:AcademicStaff;
           rdf:range  uni:Course;
           rdf:subPropertyOf ex:isInvolvedIn.
ex:isInvolvedIn rdf:domain uni:Member;
           rdf:range  uni:Course;
ex:name rdf:range xsd:string.
```

```
uni:Professor rdf:type rdfs:Class.
uni:Associate rdf:type rdfs:Class.
uni:Assistant rdf:type rdfs:Class.
uni:AcademicStaff rdf:type rdfs:Class.
uni:administrative rdf:type rdfs:Class.
uni:administrative rdf:type rdfs:Class.
uni:Member rdf:type rdfs:Class.

ex:teaches rdf:type rdfs:Property.
ex:isInvolvedIn rdf:type rdfs:Property.
```

```
rdfs:Class          rdf:type rdfs:Class.
rdfs:Property       rdf:type rdfs:Class.
rdf:type            rdf:type rdfs:Property.
rdf:domain          rdf:type rdfs:Property.
rdfs:subClassOf     rdf:type rdfs:Property.
rdfs:subPropertyOf  rdf:type rdfs:Property.
rdfs:Class    rdfs:subClassOf rdfs:Resource.
rdfs:Property rdfs:subClassOf rdfs:Resource.
…
```

Not up to the user to define.

Meaning assigned by RDFS semantics.

Core Classes of RDF(S)

- `rdfs:Resource`, the class of all resources
- `rdfs:Class`, the class of all classes
- `rdfs:Literal`, the class of all literals
- `rdf:Property`, the class of all properties.
- `rdf:Statement`, the class of all reified statements

Core Properties

- `rdf:type`, which relates a resource to its class
  - The resource is declared to be an instance of that class
- `rdfs:subClassOf`, which relates a class to one of its superclasses
  - All instances of a class are instances of its superclass
- `rdfs:subPropertyOf`, relates a property to one of its super-properties

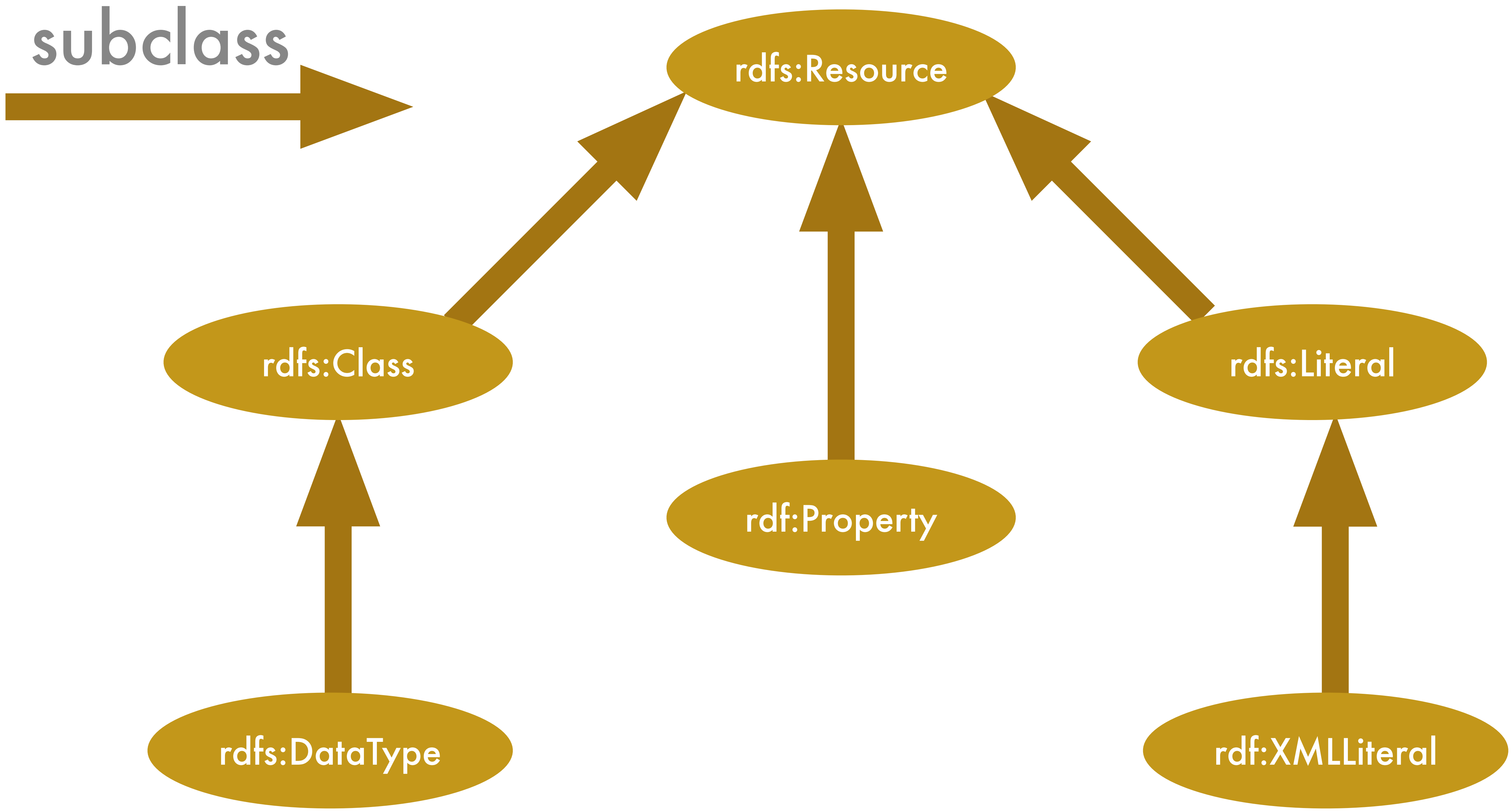
Core Properties (cont)

- `rdfs:domain`, which specifies the domain of a property P
  - The class of those resources that may appear as subjects in a triple with predicate P
  - If the domain is not specified, then any resource can be the subject
- `rdfs:range`, which specifies the range of a property P
  - The class of those resources that may appear as values in a triple with predicate P

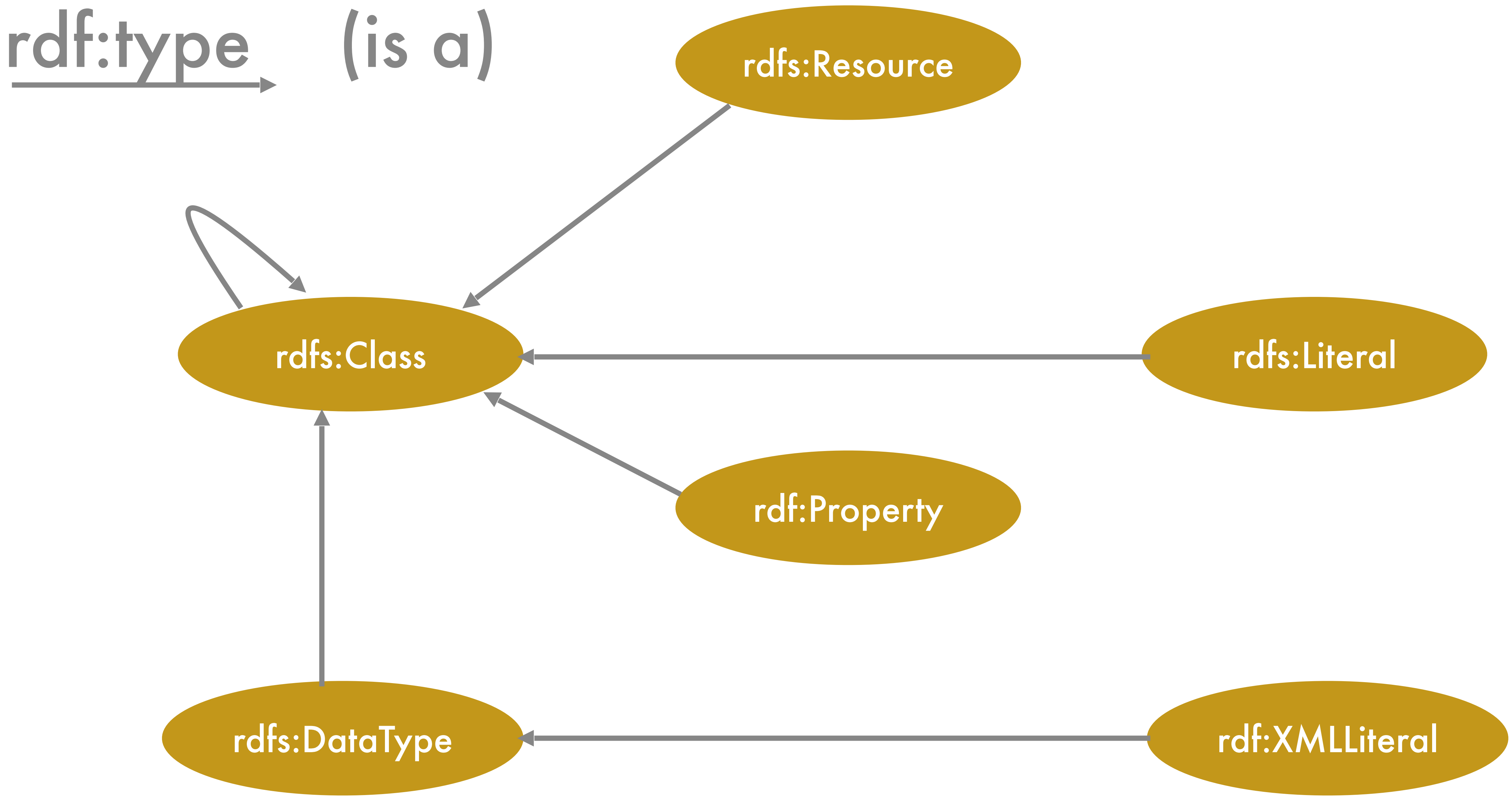
Core Classes and Properties

- `rdfs:subClassOf` and `rdfs:subPropertyOf` are transitive, by definition
- `rdfs:Class` is a subclass of rdfs:Resource
  - Because every class is a resource
- `rdfs:Resource` is an instance of `rdfs:Class`
  - `rdfs:Resource` is the class of all resources, so it is a class
- Every class is an instance of `rdfs:Class`
  - For the same reason

Subclass hierarchy

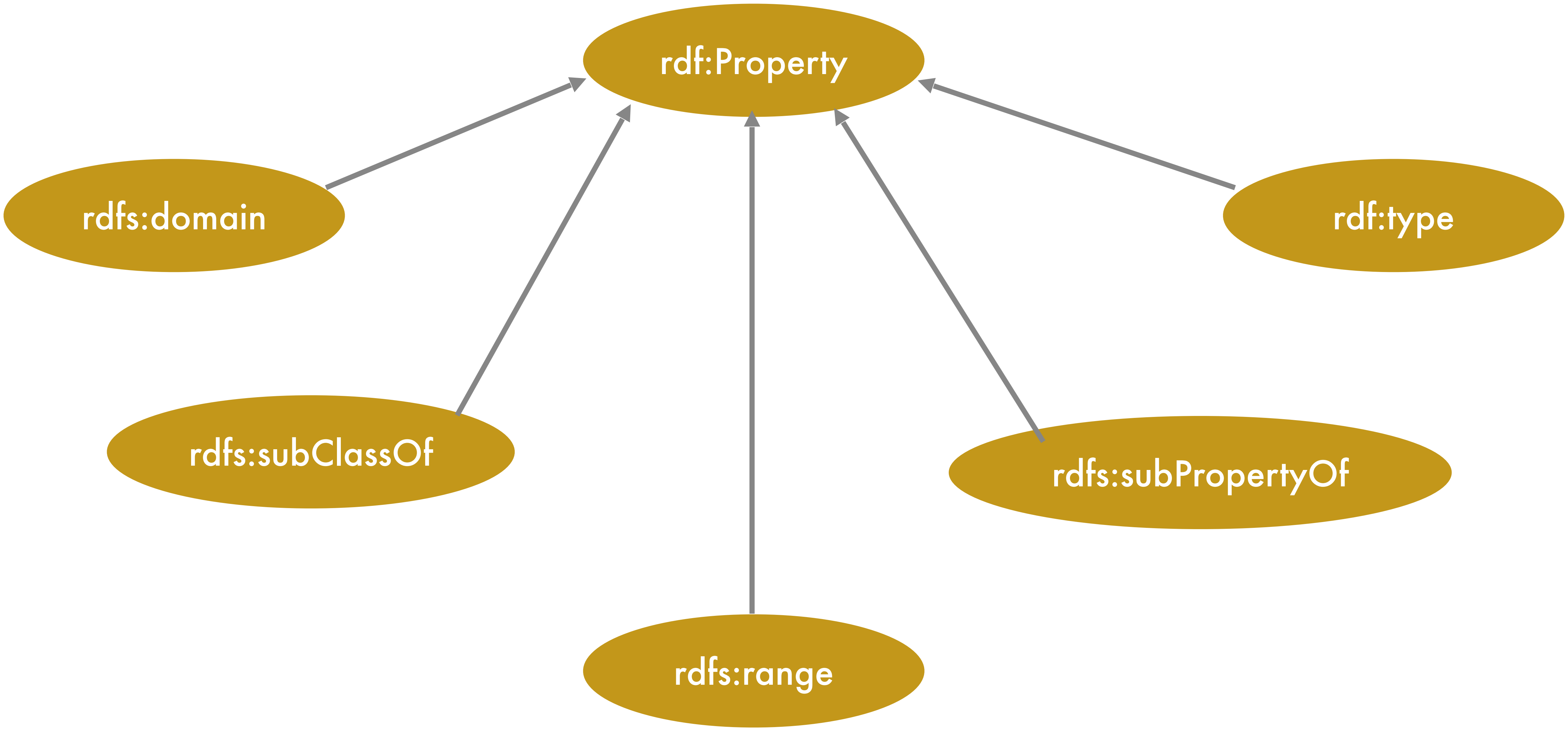


Instance relationships



FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

More instance relationships



Reification and Containers

- `rdf:subject`, relates a reified statement to its subject
- `rdf:predicate`, relates a reified statement to its predicate
- `rdf:object`, relates a reified statement to its object
- `rdf:Bag`, the class of bags
- `rdf:Seq`, the class of sequences
- `rdf:Alt`, the class of alternatives
- `rdfs:Container`, which is a superclass of all container classes, including the three above

More on containers

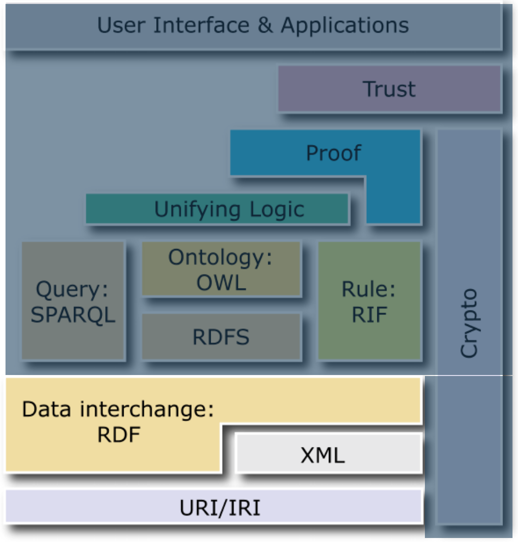
- Every property specifying that the subject contains the object is of type `rdfs:ContainerMembershipProperty`
  - E.g. `rdf:_1 rdf:type rdfs:ContainerMembershipProperty.`
- `rdfs:member` is a super property of all instances of `rdfs:ContainerMembershipProperty`
- RDFS semantics includes that
  - if `p rdf:type rdfs:ContainerMembershipProperty` and `a p b`
  - then `a rdfs:member b`

Utility properties

- `rdfs:seeAlso` relates a resource to another resource that explains it
- `rdfs:isDefinedBy` is a subproperty of `rdfs:seeAlso` and relates a resource to the place where its definition, typically an RDF schema, is found
- `rfds:comment`, associates comments, typically longer text, with a resource
- `rdfs:label`, associates a human-friendly label (name) with a resource

RDF/XML

XML syntax for RDF (supplementary material)



FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA

Why an XML syntax?

- Turtle is intuitive, understandable and machine processable
- But, there is better tool support, and off-the-shelf libraries for XML
- So, an XML syntax for RDF is more wide-spread
  - today, it is also for legacy reasons
- But remember that:
  - XML is not part of RDF (it is just another syntax)
  - E.g. serialisation of XML is irrelevant for RDF

Basic rules

- An RDF document is represented by an XML element with the tag `rdf:RDF`
  - The content of this element is a number of descriptions
- Each description, with tag `rdf:Description`, denotes a set of statements, all about a same resource
- An XML element inside a description denotes a sentence
  - The tag of the XML element denotes the attribute, and the value inside the element denotes the value of the statement
- The object resource in a `rdf:Description` may be one of the following:
  - an about attribute, referencing an existing resource
  - an ID attribute, creating a new resource
  - without a name, creating an anonymous resource (bnode)

First RDF/XML example

- As in XML, namespaces are used to disambiguate tag names
- RDF specific tags have a predefined namespace, abbreviated to rdf

```
<?xml version="1.1" encoding="utf-8" ?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:ex="http://fct.unl.pt/my-rdf">

  <rdf:Description rdf:about="http://fct.unl.pt/jja">
    <ex:name> José Alferes </ex:name>
  </rdf:Description>
</rdf:RDF>
```

FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA

Using datatypes

- With attribute `rdf:datatype`
  - and note that an `rdf:Description` may define several statements

```
<?xml version="1.1" encoding="utf-8" ?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:xsd="http://www.w3.org/2001/XLMSchema#"
 xmlns:ex="http://fct.unl.pt/my-rdf">

  <rdf:Description rdf:about="http://fct.unl.pt/jja">
    <ex:name> José Alferes </ex:name>
    <ex:age rdf:datatype="&xsd:integer"> 48 </age>
  </rdf:Description>
</rdf:RDF>
```

Referring to other resources

- Just use an `rdf:resource` attribute

```
<?xml version="1.1" encoding="utf-8" ?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:ex="http://fct.unl.pt/my-rdf">

  <rdf:Description rdf:about="http://fct.unl.pt/sw">
    <ex:courseName> Semantic Web </ex:courseName>
    <ex:taughtBy rdf:resource="http://fct.unl.pt/jja"/>
  </rdf:Description>

  <rdf:Description rdf:about="http://fct.unl.pt/jja">
    <ex:name> José Alferes </ex:name>
  </rdf:Description>
</rdf:RDF>
```

Base URIs

- XML base can be used to simplify writing of resource identifiers
  - relative URIs are recognised by the absence of the schema part; those are preceded by the XML base

```
<?xml version="1.1" encoding="utf-8" ?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:ex="http://fct.unl.pt/my-rdf"
 xml:base="http://fct.unl.pt/resources">

  <rdf:Description rdf:about="sw">
    <ex:courseName> Semantic Web </ex:courseName>
    <ex:taughtBy rdf:resource="jja"/>
  </rdf:Description>
  <rdf:Description rdf:about="jja">
    <ex:name> José Alferes </ex:name>
  </rdf:Description>
</rdf:RDF>
```

FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA

`rdf:about` versus `rdf:ID`

- An `rdf:about` indicates that the resource may be defined elsewhere
- An `rdf:ID` indicates that the resource is being defined
- There is no real difference between defining something in one place and further define it elsewhere
  - But helps for human readability
- `rdf:ID` is relative to namespace, and the resource URI has an extra # in the middle

rdf:id example

```
<?xml version="1.1" encoding="utf-8" ?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:ex="http://fct.unl.pt/my-rdf"
 xml:base="http://fct.unl.pt/resources">

  <rdf:Description rdf:id="sw">
    <ex:courseName> Semantic Web </ex:courseName>
  </rdf:Description>

  <rdf:Description rdf:id="jja">
    <ex:name> José Alferes </ex:name>
  </rdf:Description>

  <rdf:Description rdf:about="#sw">
    <ex:taughtBy rdf:resource="#jja"/>
    <ex:mainRef rdf:resource="http://press.mit/rs#semanticweb"/>
  </rdf:Description>
</rdf:RDF>
```

Nested descriptions

- Nested descriptions are possible, although the scope is always global

```
<?xml version="1.1" encoding="utf-8" ?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:ex="http://fct.unl.pt/my-rdf"
 xml:base="http://fct.unl.pt/resources">

  <rdf:Description rdf:id="sw">
    <ex:courseName> Semantic Web </ex:courseName>
    <ex:mainRef rdf:resource="http://press.mit/rs#swb"/>
    <ex:taughtBy>
      <rdf:Description rdf:id="jja">
        <ex:name> José Alferes </ex:name>
      </rdf:Description>
    </ex:taughtBy>
  </rdf:Description>
</rdf:RDF>
```

Blank nodes

- Use rdf:nodeID, to have anonymous node IDs

```
<rdf:Description rdf:id="John">
  <ex:hasCourse rdf:nodeID="id1"/>
  <ex:hasCourse rdf:nodeID="id2"/>
</rdf:Description>

<rdf:Description rdf:nodeID="id1">
  <ex:course rdf:resource="course1">
  <ex:withGrade rdf:datatype="&xsd:integer>12</ex:withGrade>
</rdf:Description>

<rdf:Description rdf:nodeID="id2">
  <ex:course rdf:resource="course2">
  <ex:withGrade rdf:datatype="&xsd:integer>15</ex:withGrade>
</rdf:Description>
```

Blank nodes

- Or without name, and nested

```
<rdf:Description rdf:id="John">
 <ex:hasCourse>
  <rdf:Description>
    <ex:course rdf:resource="course1">
    <ex:withGrade rdf:datatype="&xsd:integer>12</ex:withGrade>
  </rdf:Description>
 </ex:hasCourse>
 <ex:hasCourse>
  <rdf:Description>
    <ex:course rdf:resource="course2">
    <ex:withGrade rdf:datatype="&xsd:integer>15</ex:withGrade>
  </rdf:Description>
</rdf:Description>
```

Defining types of resources

- To be better understood with RDFS

```
<?xml version="1.1" encoding="utf-8" ?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:ex="http://fct.unl.pt/my-rdf"
 xml:base="http://fct.unl.pt/resources">

  <rdf:Description rdf:id="sw">
    <rdf:type rdf:resource="course"/>
    <ex:courseName> Semantic Web </ex:name>
    <ex:taughtBy rdf:resource="#jja"/>
  </rdf:Description>
  <rdf:Description rdf:id="jja">
    <rdf:type rdf:resource="person"/>
    <ex:name> José Alferes </ex:name>
  </rdf:Description>
</rdf:RDF>
```

Abbreviated syntax

- Simplification rules:
  1. Childless property elements within description elements may be directly replaced by XML attributes
  2. For description elements with a typing element we can use the name specified in the `rdf:type` element instead of `rdf:Description`
- These rules create syntactic variations of the same RDF statement
  - They are equivalent according to the RDF data model, although they have different XML syntax

Applying abbreviations

```
<rdf:Description rdf:id="sw">
  <rdf:type rdf:resource="course"/>
  <ex:courseName> Semantic Web </ex:name>
  <ex:taughtBy rdf:resource="jja"/>
</rdf:Description>
```

- by rule 1 becomes:

```
<rdf:Description rdf:id="sw" ex:courseName="Semantic Web">
  <rdf:type rdf:resource="course"/>
  <ex:taughtBy rdf:resource="jja"/>
</rdf:Description>
```

- and by rule 2 becomes:

```
<ex:course rdf:id="sw" ex:courseName="Semantic Web">
  <ex:taughtBy rdf:resource="#jja"/>
</ex:course>
```

Containers

- Just apply the same principles. E.g.:

```
<ex:person rdf:id="jja" name="José Alferes">
  <ex:coursesTaught>
    <rdf:Bag>
      <rdf:li rdf:resource="#sw"/>
      <rdf:li rdf:resource="#sbd"/>
    </rdf:Bag>
  </ex:coursesTaught>
```

- For collections use `rdf:parseType`

```
<rdf:Description rdf:about="http://press.mit/rs#swb">
 <ex:authors rdf:parseType="Collections">
  <rdf:Description rdf:resource="http://press.mit/rs#pascal"/>
  <rdf:description rdf:resource="http://press.mit/rs#markus"/>
  <rdf:description rdf:resource="http://press.mit/rs#sebastian"/>
 </ex:authors>
</rdf:Description>
```