# Web Ontology Languages

- Different Web ontology language
- Web Ontology Language (OWL)
- OWL characteristics

# Web Ontology Language Requirements

**Desirable features** identified for Web Ontology Language:

- Extends existing Web standards
  - Such as XML, RDF, RDFS
- Easy to understand and use
  - Should be based on familiar KR
- Formally specified
- Of “adequate” expressive power
- Possible to provide automated reasoning support

# OWL: Web Ontology Language

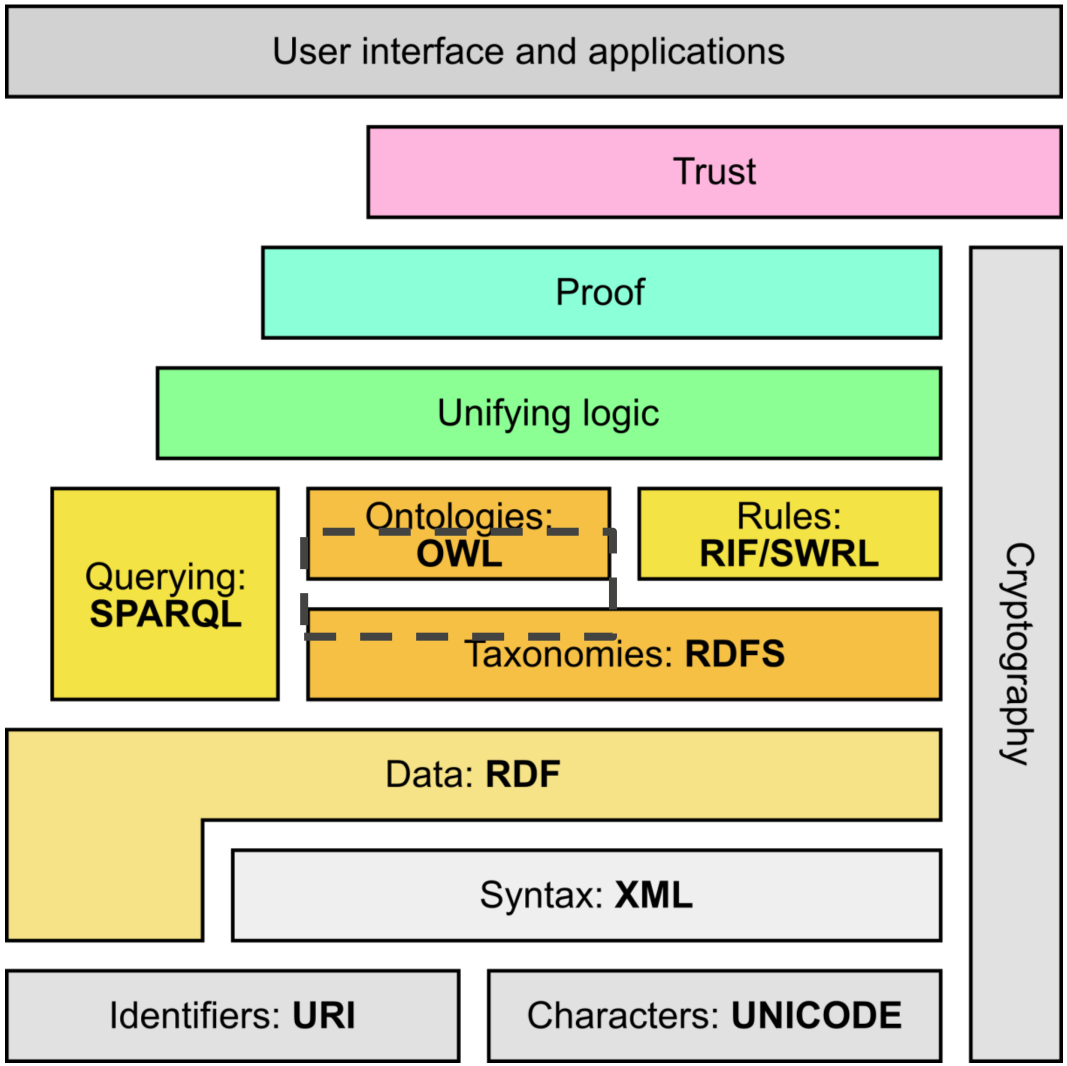
- The **Web Ontology Language** (**OWL**) is a family of knowledge representation languages or ontology languages for engineering ontologies or knowledge bases.
- **OWL DL** (Description Logic) designed to provide the maximum expressiveness possible while retaining computational completeness and availability of practical reasoning algorithms.
- OWL DL includes all OWL language constructs, but they can be used only under certain restrictions.

The built-in vocabulary for OWL all comes from the OWL namespace

http://www.w3.org/2002/07/owl#

| | | |
|---|---|---|
| http://www.w3.org/TR/owl-ref/ | OWL | 2004 |
| http://www.w3.org/TR/owl2-overview/ | OWL-2 | 2009 |

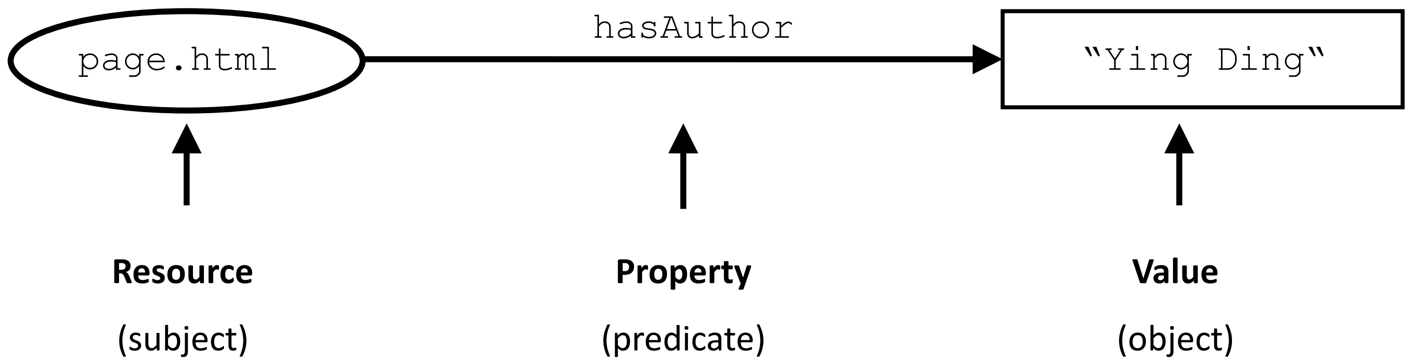
# Semantic Web Stack




Adapted from http://en.wikipedia.org/wiki/Semantic_Web_Stack

# OWL-RDFS Relationship

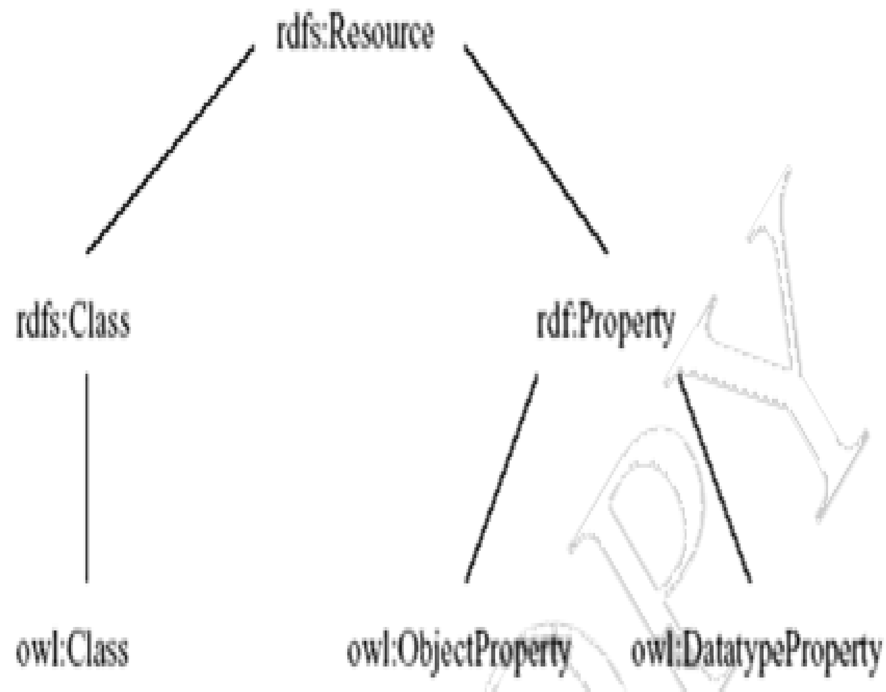
- Both use the same data model:




- OWL extends vocabulary and adds axioms to express more **complex relations of classes and properties**

# OWL Compatibility with RDF Schema

- All varieties of OWL use RDF for their syntax
- Instances are declared as in RDF, using RDF descriptions
- and typing information OWL constructors are specialisations of their RDF counterparts



# Why OWL? Limitations of RDFS

- RDFS is too weak in describing resources with sufficient details, e.g.:
  - No localised range and domain constraints
    - Cannot say that the range of *isToughtBy* is only professor when applied to professors and lecturer when applied to lecturers
  - No cardinality constraints
    - Cannot say that a course is taught by at least one professor, or persons have exactly 2 parents
  - No transitive, inverse or symmetrical properties
    - Cannot say that isPartOf is a transitive property, that hasSupervisor is the inverse of isSupervisorOf, and, that friendOf is symmetrical
  - Disjoint classes
    - Cannot say that Under-Graduate and PhD. Students are two disjoint classes
  - Boolean combinations of classes
    - Sometimes we may need to build new classes by combining other classes using union, intersection, and complement (e.g. person is the disjoint union of the classes male and female)

# Web Ontology Language (OWL)

## OWL

- A richer ontology language.
- Relations between classes
  - e.g., disjointness
- Cardinality
  - e.g. “exactly one”
- Richer typing of properties
- Characteristics of properties (e.g., symmetry)

# OWL main Building Blocks

- OWL Class Elements
- OWL Property Elements
- OWL Instances

# What’s inside an OWL ontology

- What’s inside an OWL ontology
  - Classes + class-hierarchy
  - Properties (Slots) / values
  - Relations between classes (inheritance, disjoints, equivalents)
  - Restrictions on properties (type, cardinality)
  - Characteristics of properties (transitive, …)
  - Annotations
  - Individuals
- Reasoning tasks: classification, consistency checking

# Simple Named Classes

- Every individual in the OWL world is a member of the class owl:Thing.
- Named classes
  - Example:

```
<owl:Class rdf:ID="Staff"/>
<owl:Class rdf:ID="Researcher"/>
<owl:Class rdf:ID="Academic"/>
```

- The fundamental taxonomic constructor for classes is rdfs:subClassOf.
  - Example:

```
<owl:Class rdf:ID="Researcher">
   <rdfs:subClassOf rdf:resource="#Staff" />
   ...
</owl:Class>
```

# 1. Class

- Basic concept (owl:Class) which is a subclass of rdfs:Class
- Subclasses as we know them from RDFS: rdfs:subClassOf
- – In particular, the following holds:
- owl:Class rdfs:subClassOf rdfs:Class .
- Two predefined classes: –
  - owl:Thing
  - owl:Nothing
- For each class c, the following axioms hold: –

  rdfs:subClassOf owl:Thing .

  owl:Nothing rdfs:subClassOf ?? .

- **owl:Thing** is the most general class, which contains everything
- **owl:Nothing** is the empty class

# Class relationships

- In OWL, you can construct classes from existing ones:

1. enumerate its content (individuals that together form the instances of a class)
2. intersection of two or more class descriptions
3. the union of two or more class descriptions
4. the complement of a class description

# An Example

- Woman $\equiv$ Person $\sqcap$ Female
- Man $\equiv$ Person $\sqcap$ $\neg$Woman
- Mother $\equiv$ Woman $\sqcap$ $\exists$hasChild.Person
- Father $\equiv$ Man $\sqcap$ $\exists$hasChild.Person
- Parent $\equiv$ Father $\sqcup$ Mother
- Grandmother $\equiv$ Mother $\sqcap$ $\exists$hasChild.Parent

We can further infer (though not explicitly stated):

→ Grandmother $\sqsubseteq$ Person

Grandmother $\sqsubseteq$ Man $\sqcup$ Woman

etc.

# OWL Class Elements

- owl:Class

```
<owl:Class rdf:ID="associateProfessor">
  <rdfs:subClassOf rdf:resource="#academicStaffMember"/>
</owl:Class>
```

- owl:disjoinWith

```
<owl:Class rdf:about="#associateProfessor">
  <owl:disjointWith rdf:resource="#professor"/>
  <owl:disjointWith rdf:resource="#assistantProfessor"/>
</owl:Class>
```

- owl:equivalentClass

```
<owl:Class rdf:ID="faculty">
  <owl:equivalentClass rdf:resource="#academicStaffMember"/>
</owl:Class>
```

# OWL constructs for classes

| OWL construct | Example |
| --- | --- |
| owl:Thing | |
| owl:Nothing | |
| intersectionOf($C_1 \ldots C_n$) | **Human** $\sqcap$ **Male** |
| unionOf($C_1 \ldots C_n$) | **Doctor** $\sqcup$ **Lawyer** |
| complementOf($C$) | $\neg$**Male** |
| oneOf($a_1 \ldots a_n$) | $\{$**john, mary**$\}$ |
| restriction($r$ allValuesFrom($C$)) | $\forall$**hasChild.Doctor** |
| restriction($r$ someValuesFrom($C$)) | $\exists$**hasChild.Doctor** |
| restriction($r$ minCardinality($C$)) | $\geq 2$ **hasChild.Lawyer** |
| restriction($r$ maxCardinality($C$)) | $\leq 2$ **hasChild.Lawyer** |
| restriction($r$ value($a$)) | $\exists$**citizen_of.**$\{$**France**$\}$ |

# OWL class relationships

| OWL axiom | Example |
| --- | --- |
| Class($A$ partial $C_1 \ldots C_n$)) | **Human** $\sqsubseteq$ **Physical_Object** |
| Class($A$ complete $C_1 \ldots C_n$)) | **Man** $\equiv$ **Human** $\sqcap$ **Male** |
| SubClassOf($C_1$ $C_2$) | **Human** $\sqsubseteq$ **Animal** $\sqcap$ **Biped** |
| EquivalentClasses($C_1$ $C_2$) | **Man** $\equiv$ **Human** $\sqcap$ **Male** |
| DisjointClasses($C_1$ $C_2$) | **Male** $\sqsubseteq$ $\neg$**Female** |
| SameIndividual($a_1$ $a_2$) | PresidentBush=G.W.Bush |
| DifferentIndividual($a_1$ $a_2$) | Bush$\neq$Obama |

# Class definition in owl

A class expression is the name used for either:

1- a class name (a URI), or

2- an enumeration, enclosed in <owl:Class>...</owl:Class> tags, or

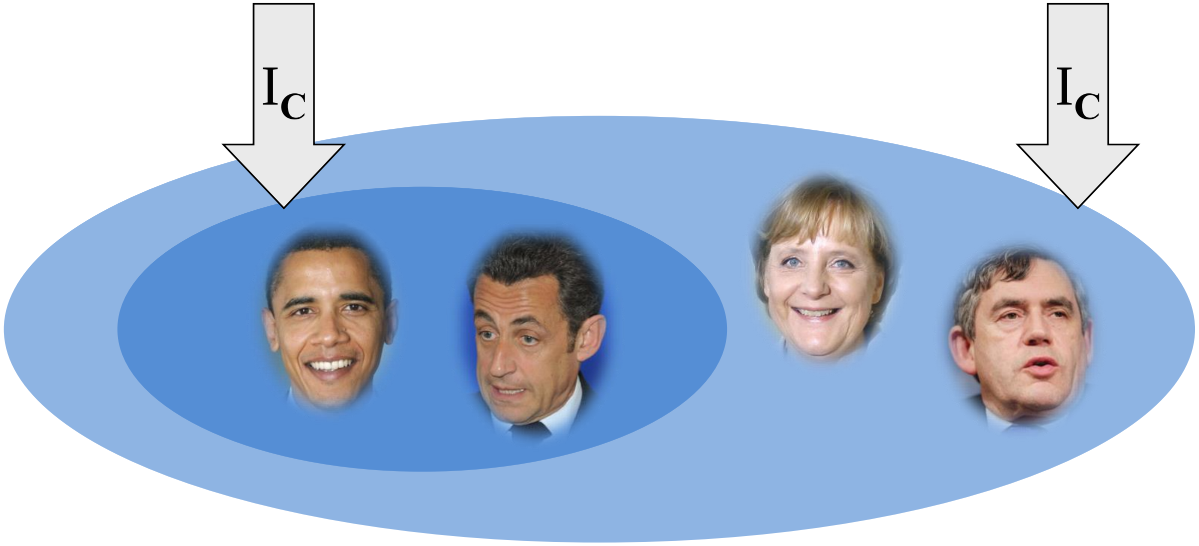
3-  a property-restriction, or

4- a boolean combination of class expressions, enclosed in <rdfs:Class>...</rdfs:Class> tags

# Class Inclusion

- classuri1 rdfs:subClassOf classuri2 .

- Example:
  ex:President rdfs:subClassOf ex:Politician



# Class Enumeration

- A class description of the "enumeration" kind is defined with the owl:oneOf property that represents exactly the enumerated individuals, no more, no less.

- The value of this property must be a list of individuals that are the instancesof the class.

- This enables a class to be described by exhaustively enumerating its instances.

- The list of individuals is typically represented by RDF construct rdf:parseType="Collection”

# Class Enumeration(cont.)

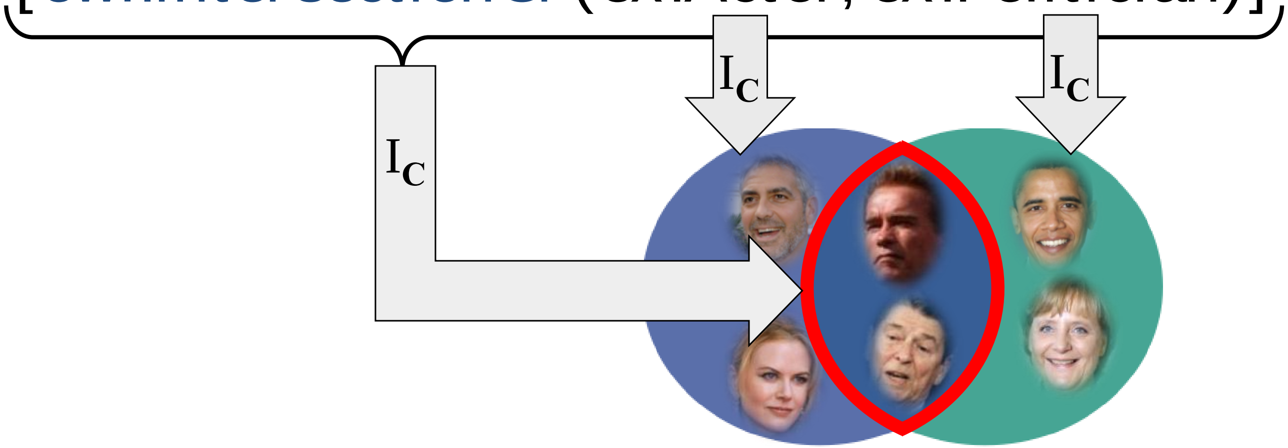
- This enables a class to be described by exhaustively enumerating its instances
- List of **individuals** that are the instances of the class

```
<owl:Class>   <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Eurasia"/>
        <owl:Thing rdf:about="#Africa"/>
        <owl:Thing rdf:about="#NorthAmerica"/>
        <owl:Thing rdf:about="#SouthAmerica"/>
        <owl:Thing rdf:about="#Australia"/>
        <owl:Thing rdf:about="#Antarctica"/>
        </owl:oneOf> </owl:Class>
```

The class extension of a class described with owl:oneOf contains exactly the enumerated individuals, no more, no less.

# Complex Classes: Intersection

- [owl:intersectionOf (class1 , ... , classn)]
- $I_C$([owl:intersectionOf (class1 , ... , classn)])
  = $I_C$(class1) Å … Å $I_C$(classn)
- Example:
  [owl:intersectionOf (ex:Actor, ex:Politician)]



# 2. Intersection

- The **owl:intersectionOf** links a class to a list of class descriptions.

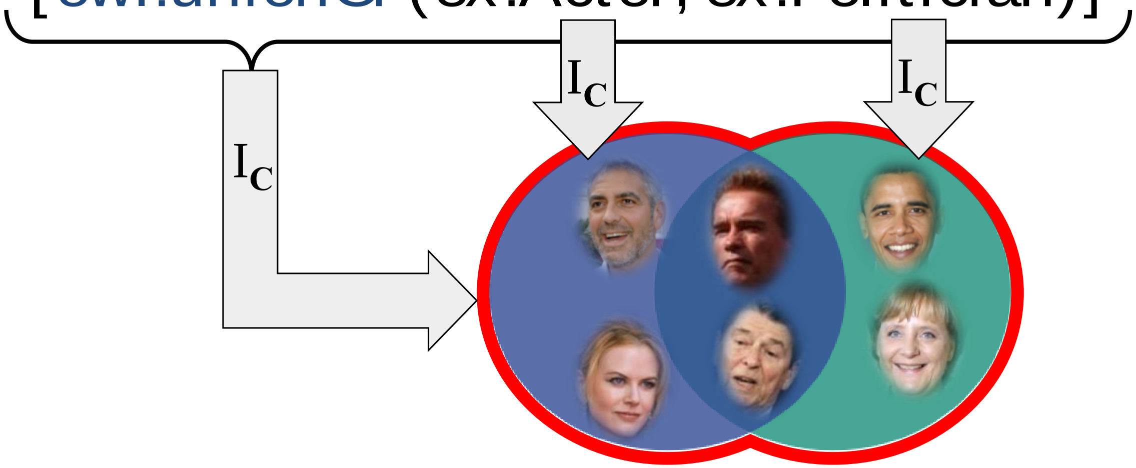
- **owl:intersectionOf:**

```
<owl:Class rdf:ID="SugaryBread">
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Bread">
        <owl:Class rdf:about="#SweetFood"/>
    </owl:intersectionOf>
</owl:Class>
```

# Complex Classes: Union

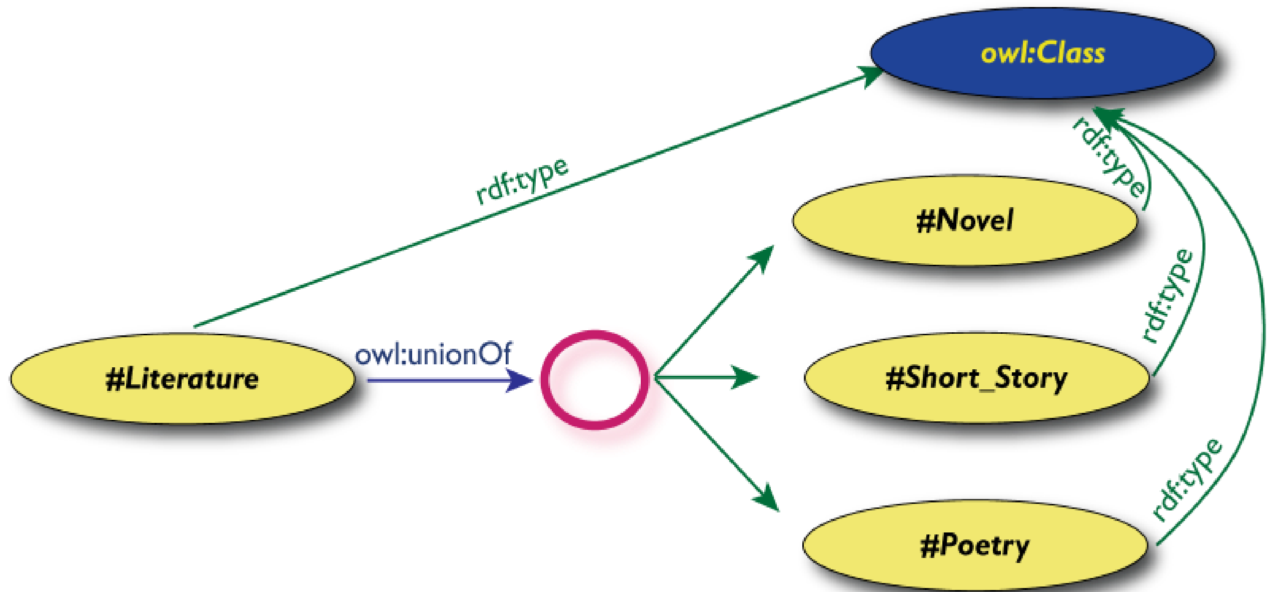
- [owl:unionOf (class1 , ... , classn)]
- $I_C$([owl:unionOf (class1 , ... , classn)])
  = $I_C$(class1) [ ... [ $I_C$(classn)
- Example:

[owl:unionOf (ex:Actor, ex:Politician)]



# 3-Union of classes

- Essentially, like a set-theoretical union:




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

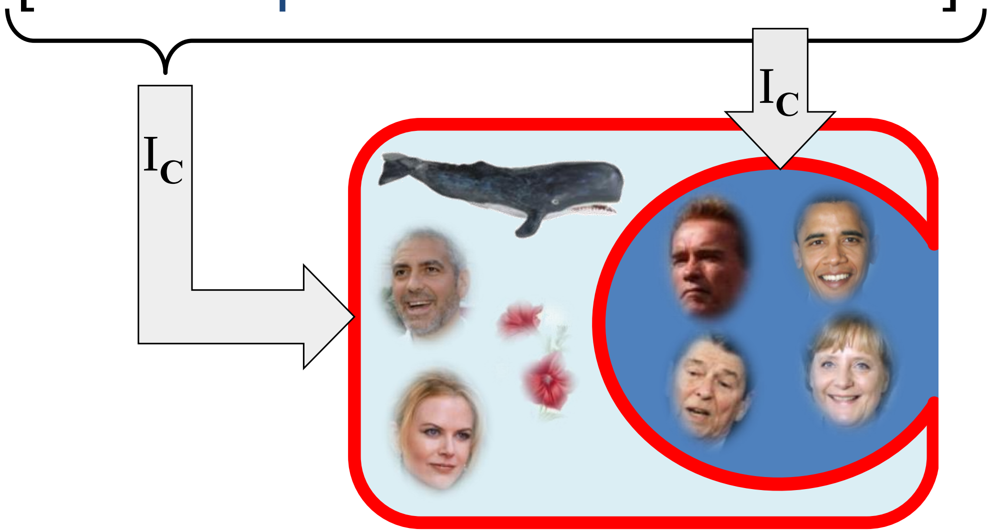
# Union of classes **(cont..)**

owl:unionOf element

```
<owl:Class rdf:ID="Fruit">
    <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#SweetFruit" />
        <owl:Class rdf:about="#NonSweetFruit" />
    </owl:unionOf>
</owl:Class>
```

# Complex Classes: Complement

- [owl:complementOf class]
- $I_C$([owl:complementOf class]) = $\Delta$ - $I_C$(class)
- Example:
  [owl:complementOf ex:Politician]



# 4-Complement of classes

```
<owl:Class rdf:ID="#Male">
    <owl:complementOf rdf:resource="#Female"/>
</owl:Class>
```

# Class Axioms

- Class axioms typically contain additional components that state necessary and/or sufficient characteristics of a class
- OWL contains three language constructs for combining class descriptions into class axioms:

1. rdfs:subClassOf : a class description is a subset of another class description.
2. owl:equivalentClass : a class description is exactly the same as another class description.
3. owl:disjointWith a class description has no members in common with the other class description.

# Class Axioms (cont.)

- owl:disjointWith

```
<owl:Class rdf:about="Mammal">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <owl:disjointWith rdf:resource="#Reptile"/>
</owl:Class>
```

```
<owl:Class rdf:about="#associateProfessor">
   <owl:disjointWith rdf:resource="#professor"/>
   <owl:disjointWith  rdf:resource="#assistantProfessor"/>
</owl:Class>
```

# Class Axioms(cont.)

**owl:equivalentClass** defines equivalence of classes

```
<owl:Class rdf:about="# faculty">
   <owl:equivalentClass rdf:resource="#academicStaffMember"/>
</owl:Class>
```

- owl:equivalentClass

```
<owl:Class rdf:about="#US_President">
<equivalentClass
     rdf:resource="#PrincipalResidentOfWhiteHouse"/>
</owl:Class>
```

# OWL Properties

- Individuals in OWL are related by *properties*. There are two types of property in OWL:

1. **Object properties** (owl:ObjectProperty relates individuals (instances) of two OWL classes.

2. **Datatype properties** (owl:DatatypeProperty) relates individuals (instances) of OWL classes to literal values.

# OWL Properties

- Datatype Property

```
<owl:DatatypeProperty rdf:ID="FName"/>
  <rdfs:domain rdf:resource="#Staff" />
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</owl:DatatypeProperty>
```

- Object Property

```
<owl:ObjectProperty rdf:ID="hasParent">
   <rdfs:domain rdf:resource="#Animal"/>
   <rdfs:range rdf:resource="#Animal"/>
</owl:ObjectProperty>
```

# OWL Property Elements

- Object property (relate objects to other objects)

```
<owl:ObjectProperty rdf:ID="teaches">
  <rdfs:range rdf:resource="#course"/>
  <rdfs:domain rdf:resource="#academicStaffMember"/>
  <owl:inverseOf rdf:resource="#isTaughtBy"/>
</owl:ObjectProperty>
```

- Data type property (relate objects to datatype values)

```
<owl:DatatypeProperty rdf:ID="age">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
  #nonNegativeInteger"/>
</owl:DatatypeProperty>
```

# Simple Properties- Example

- Using DatatypeProperty

```
<owl:Class rdf:ID=“hasName" />
<owl:DatatypeProperty rdf:ID=“nameValue" />

<rdfs:domain  rdf:resource="#PhDStudent" />
<rdfs:range rdf:resource="&xsd; string"/>
</owl:DatatypeProperty>
```

# Property Characteristics in OWL

- Datatype Properties:
DataProperty(pp:service_number range(xsd:integer))

- Property Hierarchy:
SubPropertyOf(pp:has_pet pp:likes)

- Algebraic properties:
ObjectProperty(pp:married_to Symmetric)
ObjectProperty(pp:ancestor_of Transitive)
ObjectProperty(pp:passport_nr Functional)

# Property Hierarchies

- Hierarchical relationships for properties
  - E.g., “is taught by” is a **subproperty** of “involves”.
  - If a course C is taught by an academic staff member A, then C also involves A.
- The converse is not necessarily true
  - E.g., A may be the teacher of the course C, or
  - A tutor who marks student homework but does not teach C.
- Example :
  - owl:ObjectProperty rdf:ID="hasMother">
    <rdfs:subPropertyOf rdf:resource="#hasParent"/>
    </owl:ObjectProperty>

# Property Characteristics

- OWL also allows for specifying that properties are:
- **Relations to other properties**
  - inverse
  - Equivalent
- **Logical properties**
  - TransitiveProperty
  - SymmetricProperty
- **Cardinality constraints on properties**
  - FunctionalProperty

# Property Axioms(cont.)
# Equivalent Properties

state that two properties have the same property extension : domain and range:

**owl:equivalentProperty**

```
<owl:ObjectProperty rdf:ID="lecturesIn">
<owl:equivalentProperty
    rdf:resource="#teaches"/>
</owl:ObjectProperty>
```

# Property Axioms(cont.)
# Inverse Properties

```
<owl:ObjectProperty rdf:ID="teaches">
    <rdfs:range rdf:resource="#course"/>
    <rdfs:domain rdf:resource=
        "#academicStaffMember"/>
    <owl:inverseOf rdf:resource="#isTaughtBy"/>
</owl:ObjectProperty>
```

# Flavors of a Partonomy

**isPartOf**

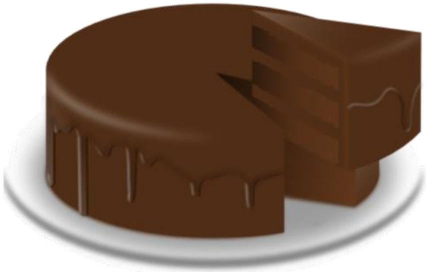
isIngradientOf

isComponentOf

isMemberOf

isContainedIn

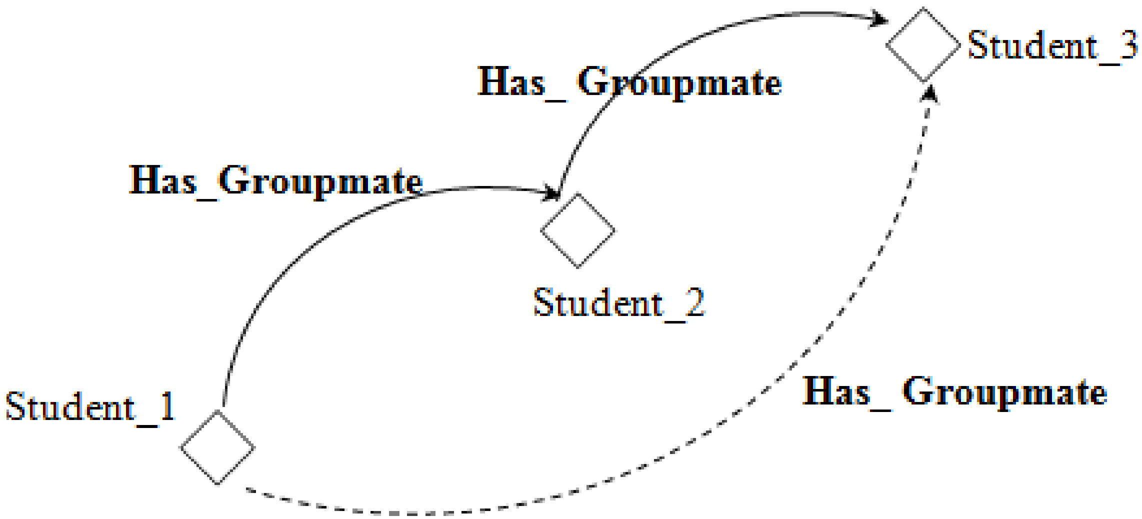
isPieceOf



# Transitive property Example

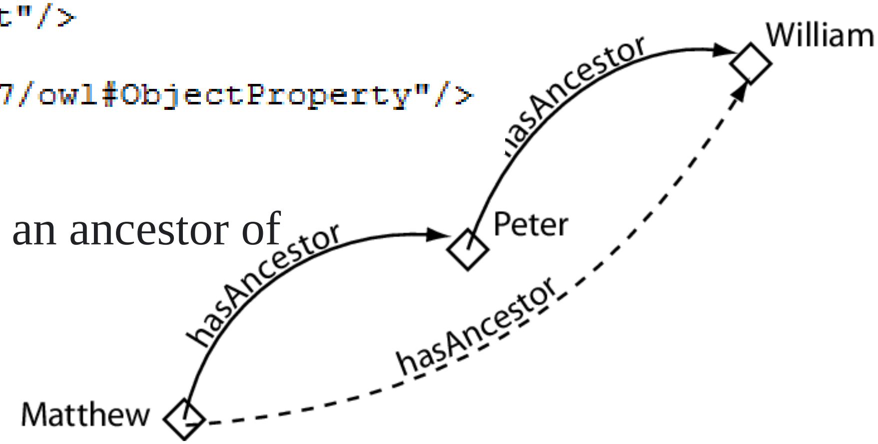
- Consider part_of property which has the characteristic of being transitive.
- If a property P is transitive, and the property relates individual a to individual b, and also individual b to individual c, then we can infer that individual a is related to individual c via property P.
- Example: is greater than, is taller than , etc

# Transitive property




```
<owl:TransitiveProperty rdf:ID="Has_Groupmate">
    <rdfs:range rdf:resource="#Student"/>
    <rdfs:domain rdf:resource="#Student"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </owl:TransitiveProperty>
```

Any ancestor of an ancestor of mine is also an ancestor of mine.



# SymmetricProperty

- A symmetric property is a property for which holds that if the pair (x,y) is an instance of P, then the pair (y,x) is also an instance of P
- Example:

```
<owl:SymmetricProperty rdf:ID="friendOf">
<rdfs:domain rdf:resource="#Human"/>
<rdfs:range rdf:resource="#Human"/> </owl:SymmetricProperty>
```

# Property Axioms(cont.) functional property

- A functional property is a property that can have only one (unique) value y for each instance x

- Example: a woman can have at most one husband

```
<owl:ObjectProperty rdf:ID=”has-husband">
<rdf:type rdf:resource="&owl;FunctionalProperty" />
<rdfs:domain rdf:resource="#Woman" />
<rdfs:range  rdf:resource="#Man" />
</owl:ObjectProperty>
```

- Both object properties and datatype properties can be declared as "functional".

# Property value restrictions

- A property restriction is a special kind of class description.
- OWL distinguishes two kinds of property restrictions: value constraints and cardinality constraints.
- Property restrictions have the following general form:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="(some property)" />
  (precisely one value or cardinality constraint)
</owl:Restriction>
```

# Property restrictions (cont..)

- **A value Constraint**

  **1- owl:allValuesFrom**

  **2- owl:someValuesFrom**

  **3- owl:hasValue**

- **Cardinality Constraint**

  **1- owl:maxCardinality**

  **2- owl:minCardinality**

  **3- owl:cardinality**

# Property Restrictions

- A (restriction) class is achieved through an **owl:Restriction** element
- This element contains an **owl:onProperty** element and one or more restriction declarations
- One type defines cardinality restrictions (at least one, at most 3,…)

# Property Restrictions (2)

- The other type defines restrictions on the kinds of values the property may take
  - **owl:allValuesFrom** specifies universal quantification
  - **owl:hasValue** specifies a specific value
  - **owl:someValuesFrom** specifies existential quantification

# owl:allValuesFrom

```
<owl:Class rdf:about="#firstYearCourse">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isTaughtBy"/>
      <owl:allValuesFrom
          rdf:resource="#Professor"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

# owl:someValuesFrom

```
<owl:Class rdf:about="#academicStaffMember">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#teaches"/>
      <owl:someValuesFrom rdf:resource=
              "#undergraduateCourse"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

# owl:hasValue

```
<owl:Class rdf:about="#mathCourse">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource=
            "#isTaughtBy"/>
            <owl:hasValue rdf:resource=
        "#949352"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

# Cardinality Restrictions

- We can specify minimum and maximum number using **owl:minCardinality** and **owl:maxCardinality**
- It is possible to specify a precise number by using the same minimum and maximum number
- For convenience, OWL offers also **owl:cardinality**

# Cardinality Restrictions (2)

**owl:minCardinality constraint describes a class of all individuals that have at least N semantically distinct values**

**For example:**

```
<owl:Class rdf:about="#course">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isTaughtBy"/>
            <owl:minCardinality rdf:datatype=
            "&xsd;nonNegativeInteger">1

            </owl:minCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

# Cardinality Restrictions (3)

```
<owl:Class rdf:ID=“PhDStudent">
   <rdfs:subClassOf  rdf:resource=“#DeptMembers"/>
   <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasSupervisor"/>
        <owl:minCardinality
             rdf:datatype="&xsd;nonNegativeInteger">1
      </owl:minCardinality>
     </owl:Restriction>
</rdfs:subClassOf>
...

</owl:Class>
```

# Property Inherence in OWL

- Some “features” of OWL are “inherited”. Others are not. For example:

- (a subPropertyOf b) and (b inverseOf c)  doesn't imply (a inverseOf c)
- (a subPropertyOf b) and (b equivalentPropertyOf c) doesn't imply (a equivalentPropertyOf c)
- (a subPropertyOf b) and (b type TransitiveProperty) doesn't imply (a type TransitiveProperty)

# Individuals (Instances)

- Individuals enable us to describe members of a class.

- On the **web**, such an assumption is **not possible**. The same person could be referred to in many different ways (i.e. with different URI references).

- OWL does not make this assumption (unless explicitly stated).

# Declaring Instances

Instances of classes are declared as in RDF:

```
<rdf:Description rdf:ID=“XYZ">
      <rdf:type rdf:resource=   "#academicStaffMember"/>
</rdf:Description>
<academicStaffMember rdf:ID=“XYZ">
          <uni:age rdf:datatype="&xsd;integer"> 39<uni:age>
</academicStaffMember>
```

# Individuals (Instances)

- OWL provides three constructs for stating facts about the **identity of individuals**:

1. owl:sameAs: is used to state that two URI references refer to the same individual.
2. owl:differentFrom: is used to state that two URI references refer to different individuals.
3. owl:AllDifferent: provides an idiom for stating that a list of individuals are all different.

# Individuals (Instances)

- Example:

```
<rdf:Description
    rdf:about="#William_Jefferson_Clinton">
<owl:sameAs rdf:resource="#BillClinton"/>
</rdf:Description>

<rdf:Description rdf:about=”#Obama”>
<owl:differentFrom  rdf:resource="#BillClinton"/>
</rdf:Description>
```

# Individuals (Instances)

- Example:

```
<owl:AllDifferent>
<owl:distinctMembers rdf:parseType="Collection">
<Opera rdf:about="#Don_Giovanni"/>
<Opera rdf:about="#Nozze_di_Figaro"/>
<Opera rdf:about="#Cosi_fan_tutte"/>
<Opera rdf:about="#Tosca"/>
<Opera rdf:about="#Turandot"/>
<Opera rdf:about="#Salome"/>
</owl:distinctMembers>
</owl:AllDifferent>
```

# URI “aliases” and Introduction to Linked Data

Berlin

Lets take city of Berlin as an example …

# URI “aliases” and Introduction to Linked Data

Berlin

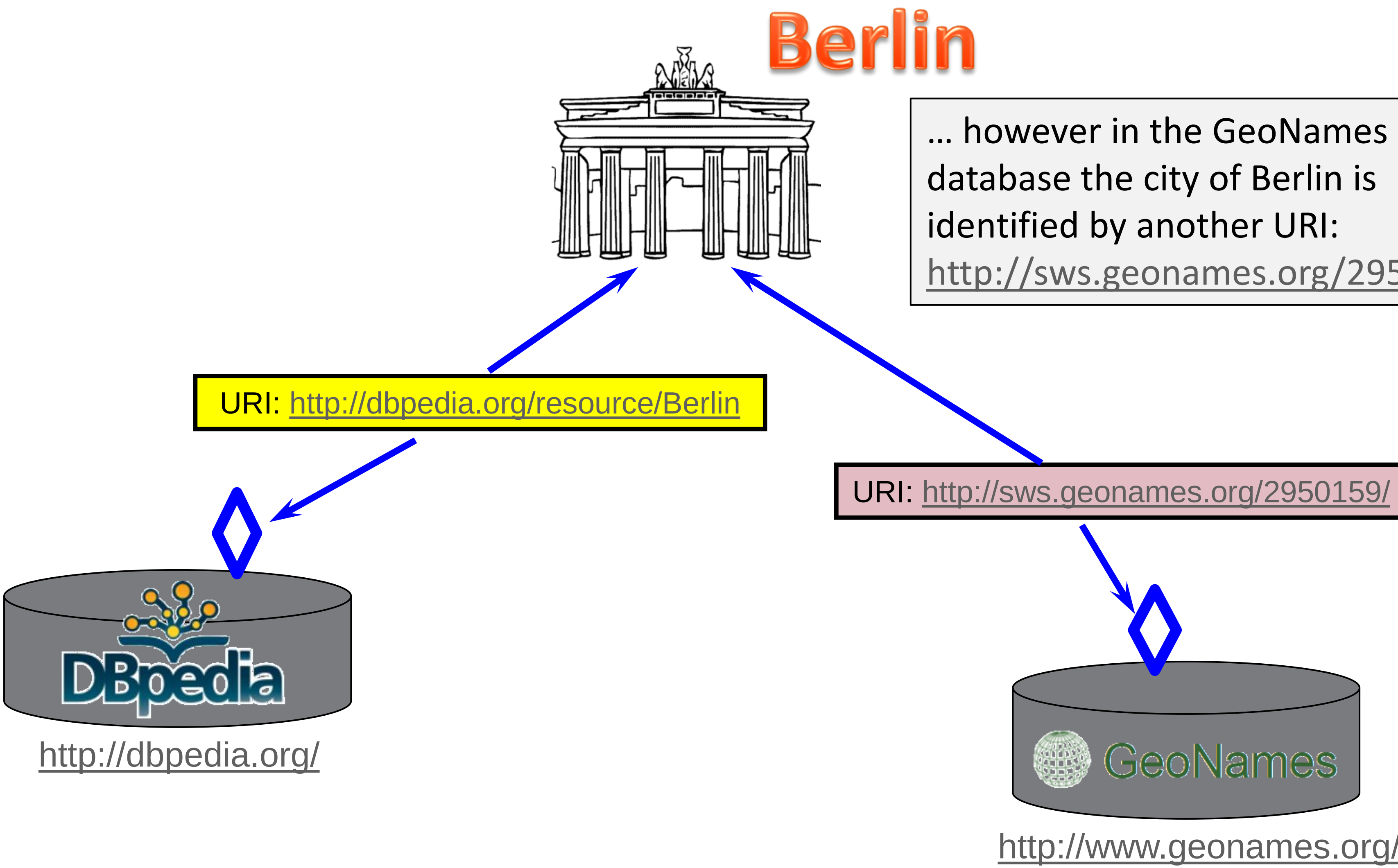
In the DBPedia the city of Berlin is identified by the following URI: http://dbpedia.org/resource/Berlin

URI: http://dbpedia.org/resource/Berlin


http://dbpedia.org/

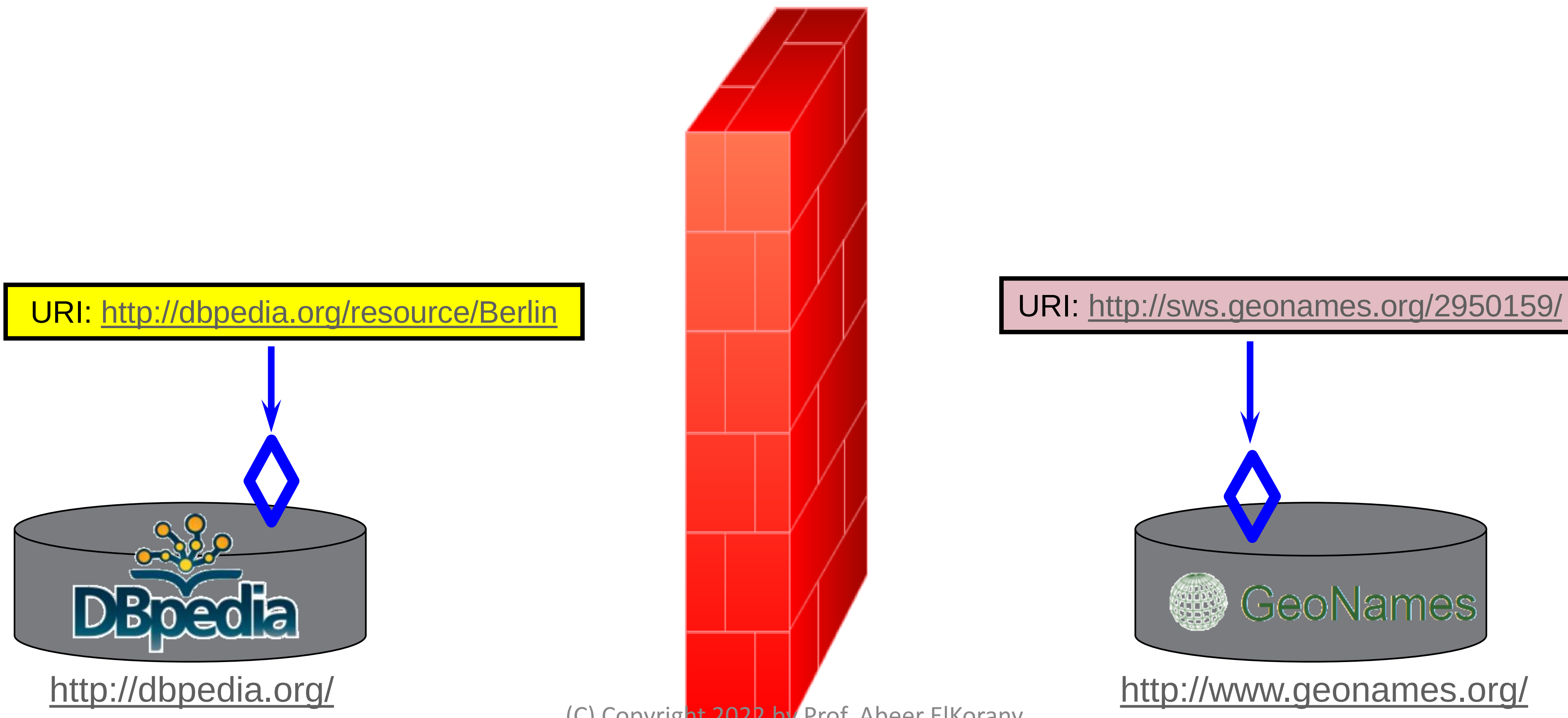
# URI “aliases” and Introduction to Linked Data

… however in the GeoNames database the city of Berlin is identified by another URI: http://sws.geonames.org/2950159/



# URI “aliases” and Introduction to Linked Data

For any software application these two URIs (URI aliases) look now like they refer to totally different objects …

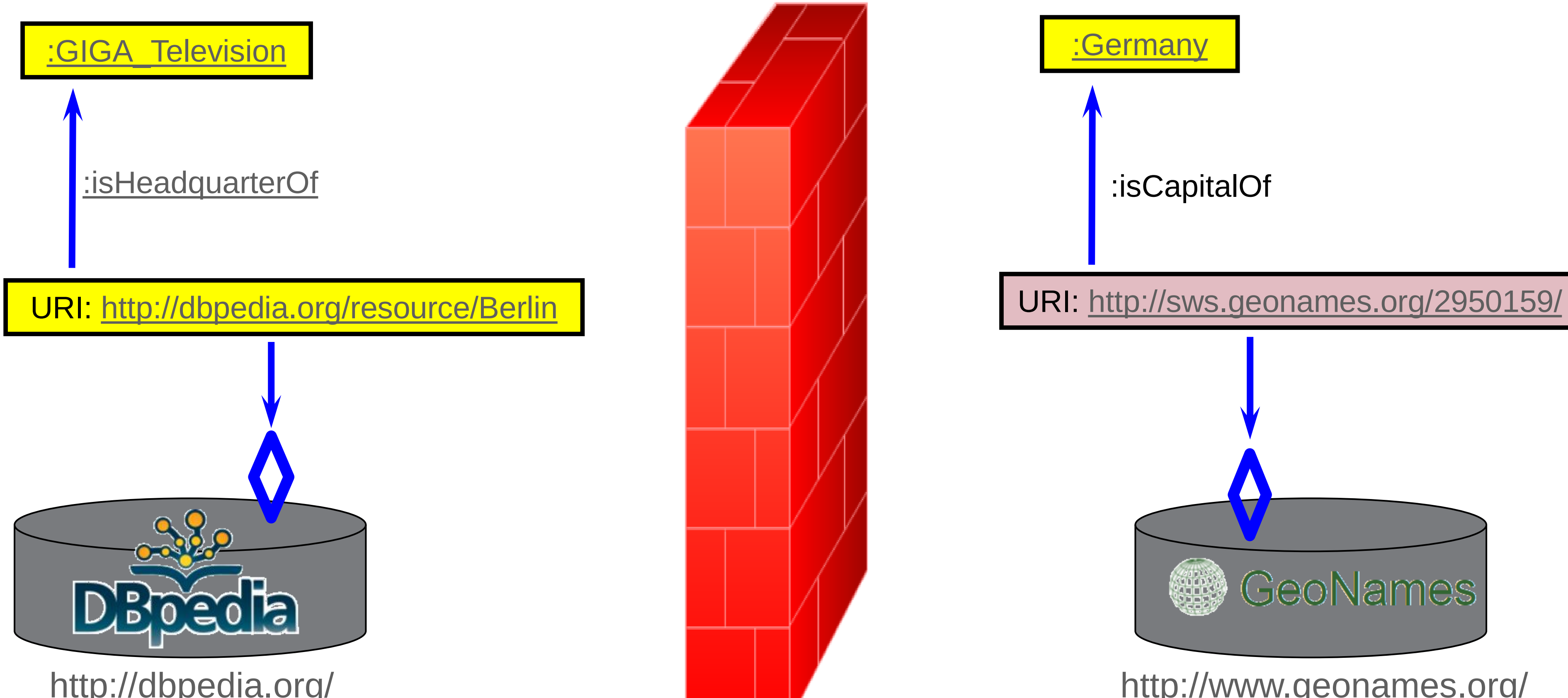


# URI "aliases" and Introduction to Linked Data

… now we have two isolated "islands" of data …

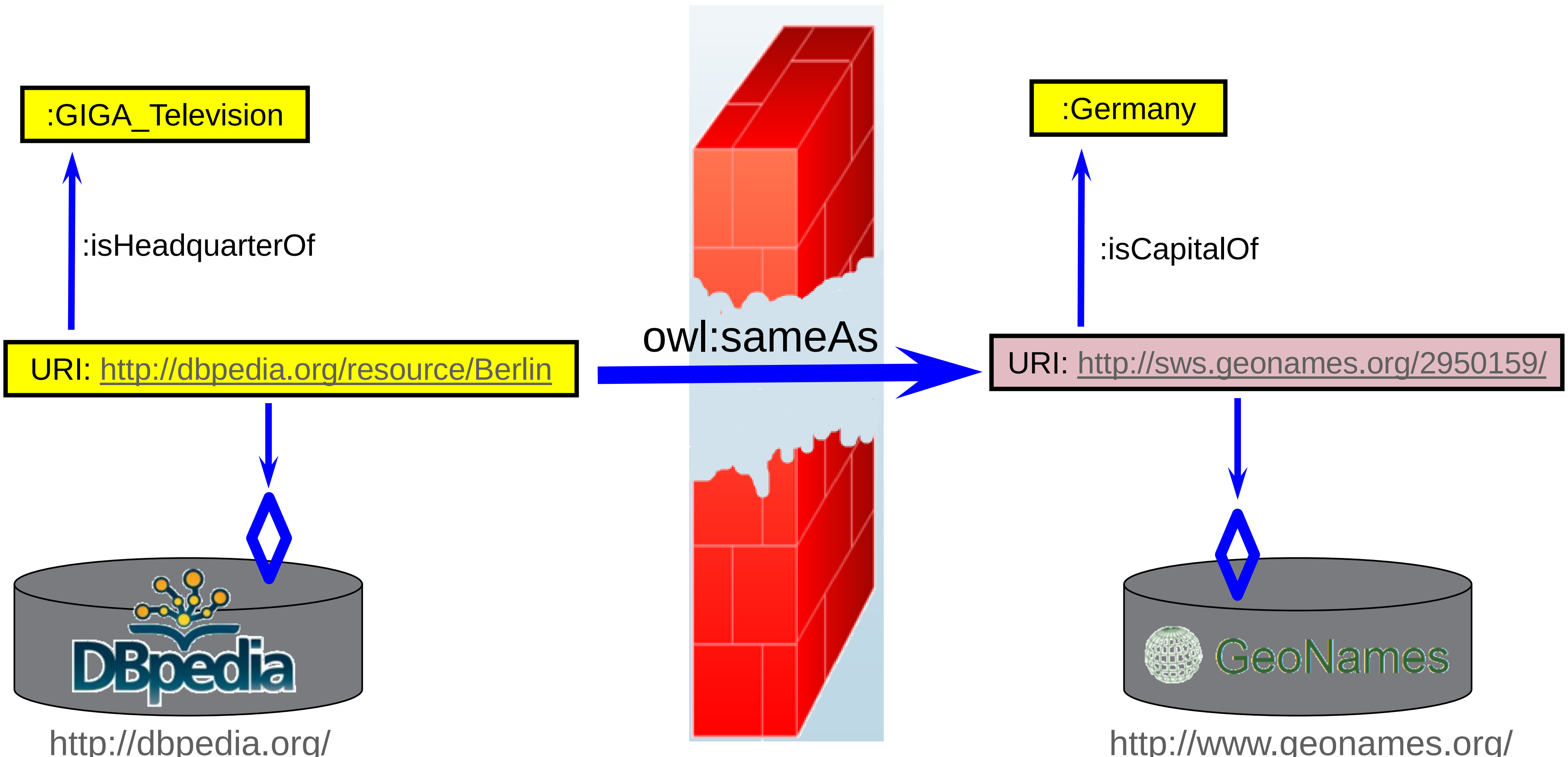
Here you can answer queries like, e.g., "*Which city is a headquarter of GIGA Television?*"

Here you can answer queries, like e.g., "*Which city is a capital of Germany?*"



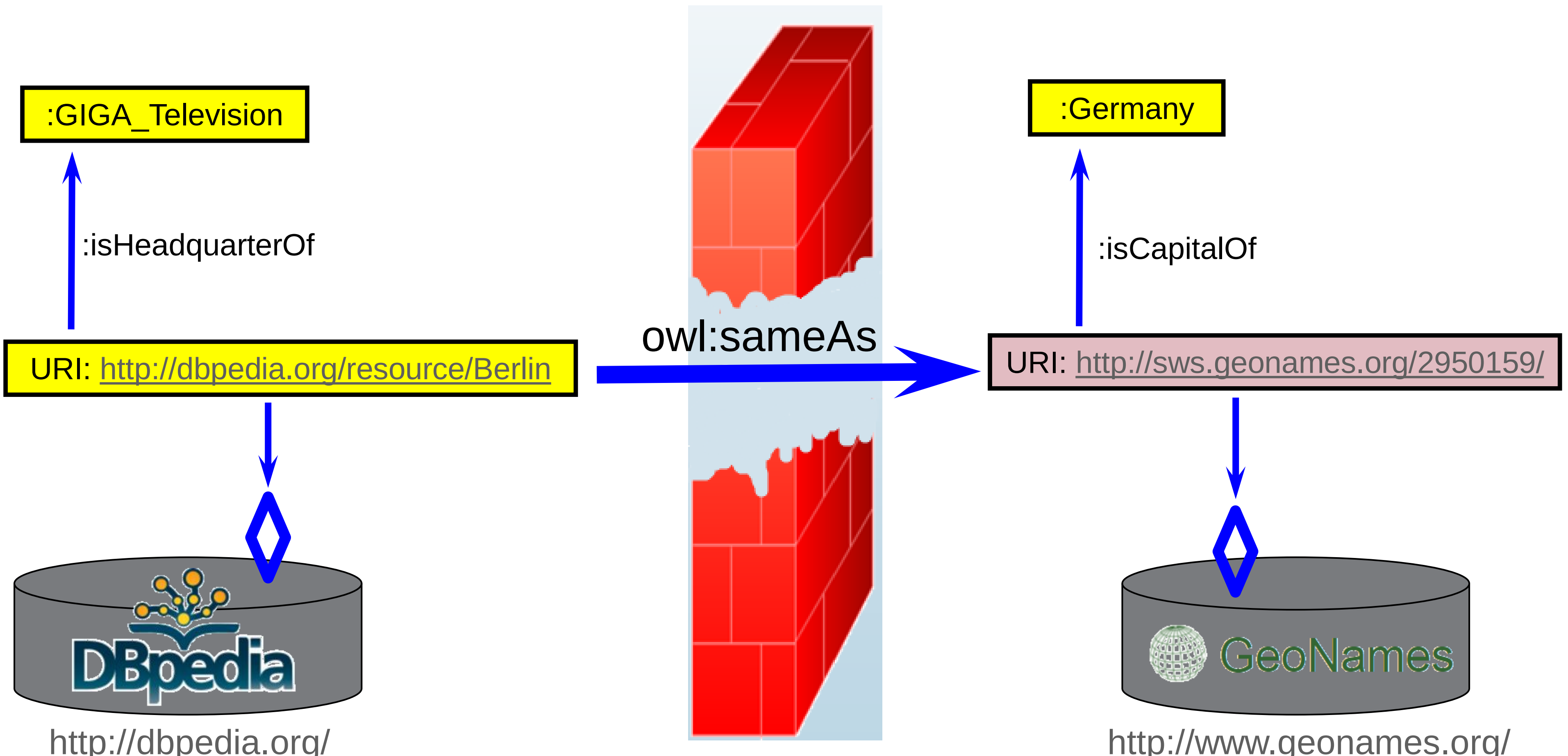
# URI “aliases” and Introduction to Linked Data

“Break the wall!” Make the connection between distinct URIs explicit ! You will obtain a Linked Data …



# URI “aliases” and Introduction to Linked Data

… now you are able to make queries over the Linked Data (distributed database) like, e.g.: “*Which companies have headquarters in the capital of Germany?*”



# Why OWL

## Reasoning About Knowledge in Ontology Languages

- Class membership
  - If x is an instance of a class C, and C is a subclass of D, then we can infer that x is an instance of D
- Equivalence of classes
  - If class A is equivalent to class B, and class B is equivalent to class C, then A is equivalent to C, too

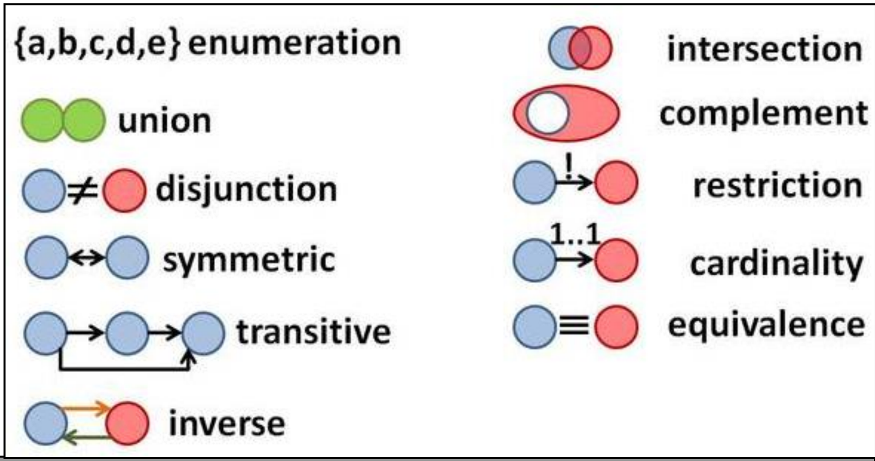
## Why OWL

# Reasoning About Knowledge in Ontology Languages (2)

- Consistency
  - X instance of classes A and B, but A and B are disjoint
  - This is an indication of an error in the ontology
- Classification
  - Certain property-value pairs are a sufficient condition for membership in a class A; if an individual x satisfies such conditions, we can conclude that x must be an instance of A

# OWL on one Slide

- **Symmetric**: if P(x, y) then P(y, x)
- **Transitive**: if P(x,y) and P(y,z) then P(x, z)
- **Functional**: if P(x,y) and P(x,z) then y=z
- **InverseOf**: if P1(x,y) then P2(y,x)
- **InverseFunctional**: if P(y,x) and P(z,x) then y=z
- **allValuesFrom**: P(x,y) and y=allValuesFrom(C)
- **someValuesFrom**: P(x,y) and y=someValuesFrom(C)
- **hasValue**: P(x,y) and y=hasValue(v)
- **cardinality**: cardinality(P) = N
- **minCardinality**: minCardinality(P) = N
- **maxCardinality**: maxCardinality(P) = N
- **equivalentProperty**: P1 = P2
- **intersectionOf**: C = intersectionOf(C1, C2, …)
- **unionOf**: C = unionOf(C1, C2, …)
- **complementOf**: C = complementOf(C1)
- **oneOf**: C = one of(v1, v2, …)
- **equivalentClass**: C1 = C2
- **disjointWith**: C1 != C2
- **sameIndividualAs**: I1 = I2
- **differentFrom**: I1 != I2
- **AllDifferent**: I1 != I2, I1 != I3, I2 != I3, …
- **Thing**: I1, I2, …




**Legend:**
Properties are indicated by: P, P1, P2, etc
Specific classes are indicated by: x, y, z
Generic classes are indicated by: C, C1, C2
Values are indicated by: v, v1, v2
Instance documents are indicated by: I1, I2, I3, etc.
A number is indicated by: N
P(x,y) is read as: “property P relates x to y”

# OWL Example Syntax: RDF/XML

```
< rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
```

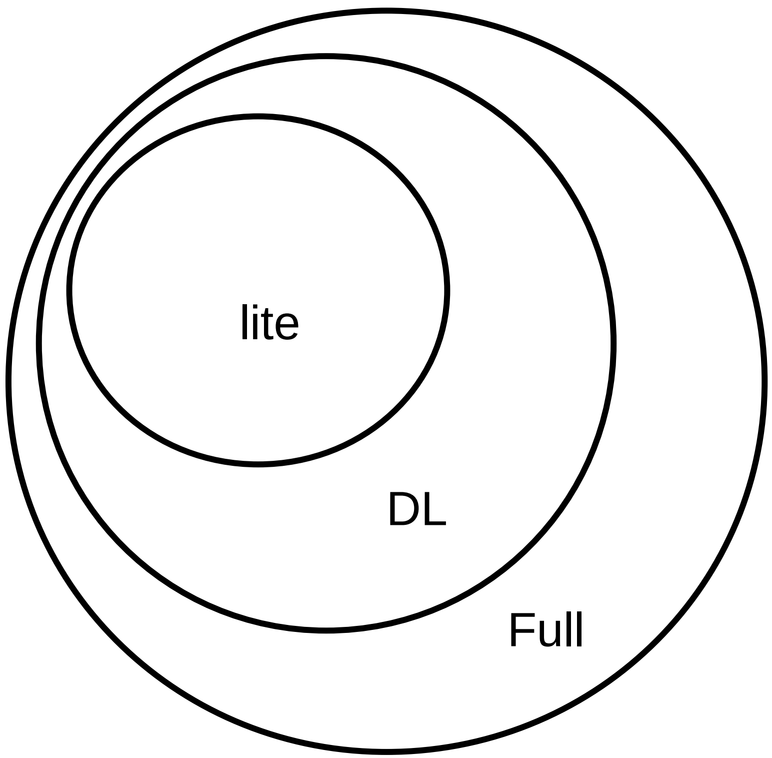
Header

```
<owl:Class rdf:ID="SpicyPizza">
  <rdfs:label xml:lang="pt">PizzaTemperada</rdfs:label>
  <rdfs:comment xml:lang="en">Any pizza that has a spicy topping is a SpicyPizza</rdfs:comment>
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Pizza"/>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#hasTopping"/>
          </owl:onProperty>
          <owl:someValuesFrom rdf:resource="#SpicyTopping"/>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

# OWL Dialects

- OWL Lite
  - Classification hierarchy
  - Simple constraints
- OWL DL
  - Maximal expressiveness while maintaining tractability
  - Standard formalization in a DL
- OWL Full
  - Very high expressiveness
  - Losing tractability
  - All syntactic freedom of RDF (self-modifying) ): e.g. statements about statements are possible. Meta-modeling as well.

# OWL comes in 3 Dialects




1. Lite

partially restricted to aid learning curve

2. DL = Description Logic

Description Logics are a fragment of First Order Logic (FOL) that can be reasoned with

3. Full

unrestricted use of OWL constructs, but cannot reason

# Three Species of OWL

- W3C'sWeb Ontology Working Group defined OWL as three different sublanguages:
  – OWL Full
  – OWL DL
  – OWL Lite
- Each sublanguage geared toward fulfilling different aspects of requirements

# Summary

- RDF (Resource Description Framework) is a simple language for expressing RDF statements.
- RDF/XML is syntax to express a RDF graph as a XML document.
- RDFS (RDF Schema) extends the vocabulary of simple RDF.
- OWL further extends the RDFS vocabulary to allow more complex inference and reasoning.
- SPARQL (Search Protocol and RDF Query Language or "sparkle") is a query language for semantic data similar to SQL.