OWL

-Web Ontology Language

Part 1

Acknowledgement

inspired by

Monika Solanki

with Thanks to

Roger L. Costello

David B. Jacobs

The MITRE Corporation

Outline

- Basic ideals of Web Ontology Language (OWL)
- The OWL Language
- OWL2 profiles
- Examples

Limitations of the Expressive Power of RDF Schema

- Local scope of properties
  - `rdfs:range` defines the range of a property (e.g. eats) for all classes
  - In RDF Schema we cannot declare range restrictions that apply to some classes only
  - e.g. we cannot say that cows eat only plants, while other animals may eat meat, too
- Disjointness of classes
  - Sometimes we wish to say that classes are disjoint (e.g. male and female)

Limitations of the Expressive Power of RDF Schema (2)

- Combinations of classes
  - Sometimes we wish 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
- Cardinality restrictions
  - e.g. a person has exactly two parents, a course is taught by at least one lecturer

Limitations of the Expressive Power of RDF Schema (3)

- Special characteristics of properties
  - Transitive property (like “greater than”)
  - Unique property (like “is mother of”)
  - Symmetric property (like “is a friend of”)
  - A property is the inverse of another property (like “eats” and “is eaten by”)

Web Ontology Language (OWL)

- OWL is developed as a vocabulary extension of RDF (the Resource Description Framework)
- Derived from the DAML+OIL Web Ontology
- Specifications
  - OWL: W3C Recommendation (2004)
    - Sublanguages: OWL Full, OWL DL, OWL lite
  - OWL2: W3C Recommendation (2009)
    - OWL2 Profiles: OWL2 RL, OWL2 EL, OWL2 QL

OWL Full

- It uses all the OWL languages primitives
- It allows the combination of these primitives in arbitrary ways with RDF and RDF Schema
- OWL Full is fully upward-compatible with RDF, both syntactically and semantically
- OWL Full is so powerful that it is undecidable
- No complete (or efficient) reasoning support

OWL DL

- OWL DL (Description Logic) is a sublanguage of OWL Full that restricts application of the constructors from OWL and RDF
- Designed to provide the maximum expressiveness possible while retaining decidability
- OWL DL permits efficient reasoning support
- But we lose full compatibility with RDF:
  - Not every RDF document is a legal OWL DL document.
  - Every legal OWL DL document is a legal RDF document.

OWL Lite

- restricts OWL DL to a subset of the language constructors
- designed to support a classification hierarchy and simple constraints
  - e.g., OWL Lite excludes enumerated classes, disjointness statements, and arbitrary cardinality (only permit cardinality values of 0 or 1).
- The advantage of this is a language that is easier to
  - grasp,by users
  - implement, for tool builders
- The disadvantage is restricted expressivity

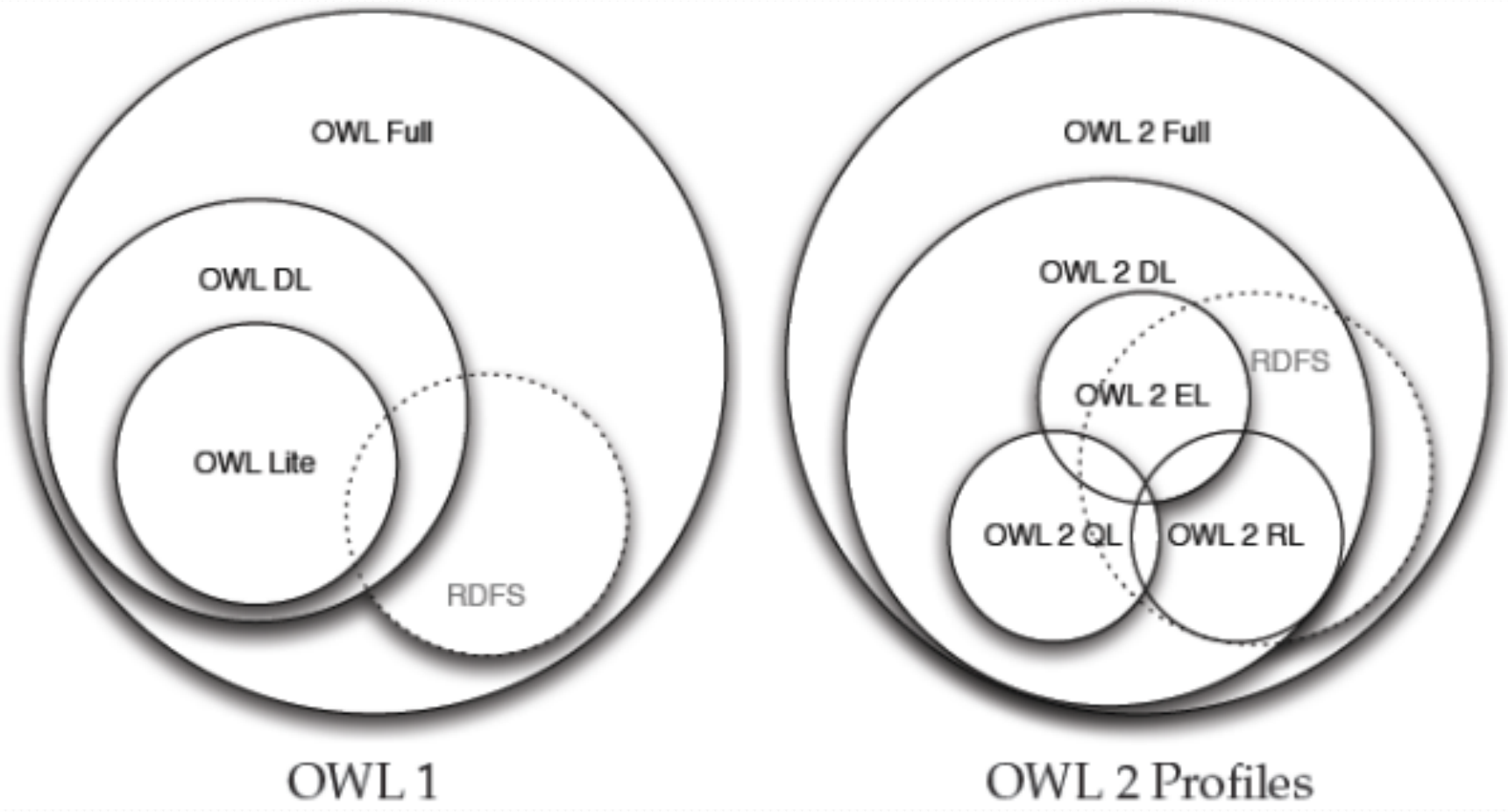
OWL2

- OWL 2 extends the W3C OWL Web Ontology Language with a small but useful set of features that have been requested by users
  - Extra "syntactic sugar"
  - Additional property and qualified cardinality constructors
  - Extended datatype support
  - Metamodelling
  - Extended annotations
- OWL2 is a subset of OWL
  - OWL 2 is backward compatible
  - Creators of OWL 1 documents need only move to OWL 2 when they want to make use of OWL 2 features

OWL2 Profiles

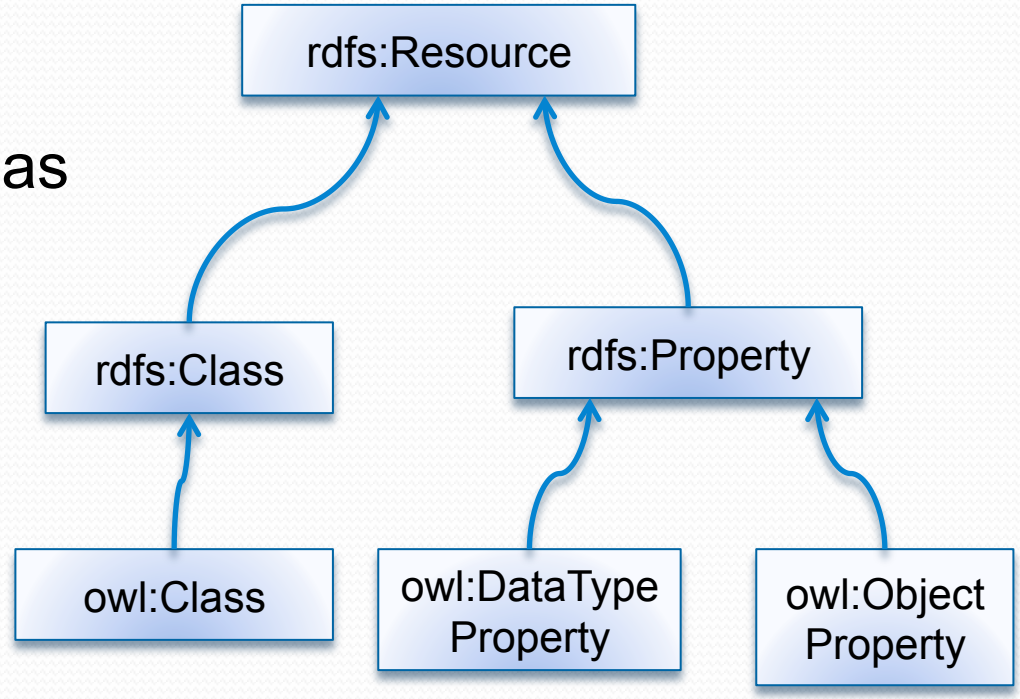
- The purpose of an OWL 2 profile is to provide a trimmed down version of OWL 2 that trades expressive power for efficiency of reasoning
- Choice of profiles
  - OWL 2 EL: used for ontologies containing a large numbers of properties and/or classes
  - OWL 2 QL: used for ontology-based applications that have large volumes of instance data, and where query answering is the most important reasoning task.
  - OWL2 RL: used for applications that require scalable reasoning without sacrificing too much expressive power

OWL sublanguages and OWL2 profiles



OWL Compatibility with RDF Schema

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



OWL Syntactic Varieties

- OWL builds on RDF and uses RDF's XML- based syntax
- An OWL ontology is an RDF graph,
  - which is in turn a set of RDF triples.
  - as with any RDF graph, an OWL ontology graph can be written in many different syntactic forms
  - a graphic syntax based on the conventions of UML has also been defined

e.g. two different syntactic forms to say Student is a subclass of Person

```
<owl:Class rdf:ID="Student">
  <owl:subClassOf>
    <owl:Class rdf:ID="Person"/>
  </owl:subClassOf>
</owl:Class>
```

```
<owl:Class rdf:ID="Student">
  <owl:subClassOf rdf:resource="#Person"/>
</owl:Class>
```

OWL XML/RDF Syntax: Header

- An OWL ontology may start with a collection of assertions for housekeeping purposes using owl:Ontology element

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
>
```

owl:Ontology

- owl:imports is a transitive property

```
<owl:Ontology rdf:about="">
<rdfs:comment>An example OWL ontology </rdfs:comment>
 <owl:priorVersion
rdf:resource="http://www.mydomain.org/uni-ns-old"/>
 <owl:imports
rdf:resource="http://www.mydomain.org/persons"/>
<rdfs:label>University Ontology</rdfs:label>
</owl:Ontology>
```

- OWL ontology can import others OWL or RDF-S ontologies
- Importing an ontology into itself is considered a null action,
  - e.g. if ontology A imports B and B imports A, then they are considered to be equivalent

Classes (1)

- Classes are defined using **owl:Class**
- **owl:Class** is a subclass of **rdfs:Class**
  - Disjointness is defined using **owl:disjointWith**

```
<owl:Class rdf:about="#PostgraduateStudent">
  <owl:disjointWith rdf:resource="#UndergraduateStudent"/>
</owl:Class>
```

- OWL allows mixing of RDF and OWL constructs

```
<owl:Class rdf:about="#PostgraduateStudent">
  <owl:disjointWith rdf:resource="#UndergraduateStudent"/>
  <rdfs:subClassOf rdf:resource="#Student"/>
</owl:Class>
```

Classes (2)

- **owl:equivalentClass** defines equivalence of classes
  - owl:equivalentClass means a class description has exactly the same class extension as another class description.

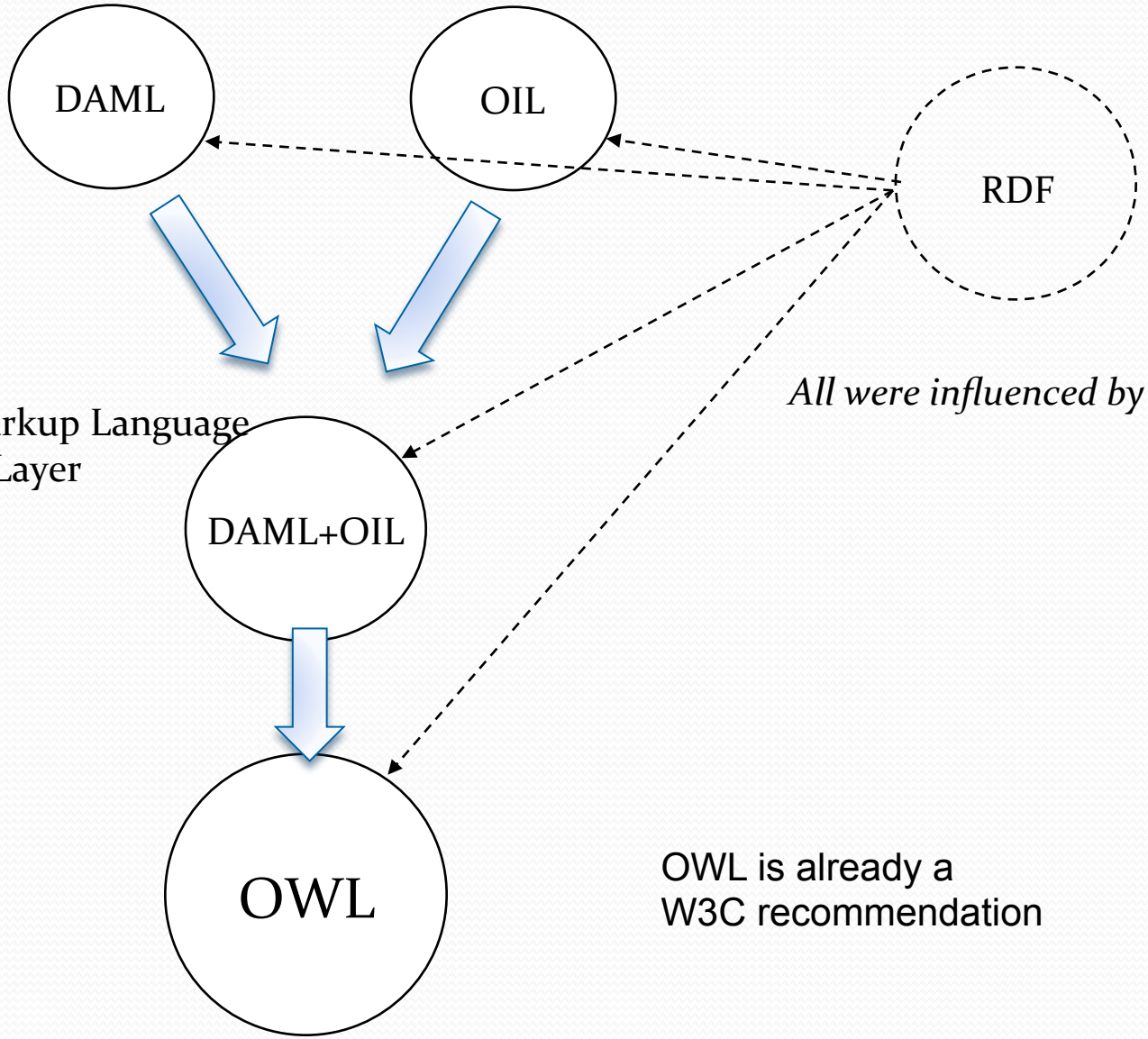
```
<owl:Class rdf:ID="Faculty">
  <owl:equivalentClass rdf:resource="#AcademicStaffMember"/>
</owl:Class>
```

- **owl:Thing** built-in most general class, it is the class of all individuals and is a superclass of all OWL classes
- **owl:Nothing** is a class that has no instances and a subclass of all OWL classes.

Prerequisites

- OWL builds on top of (i.e., extends) RDF Schema.
- We assumes that you already have a solid understanding of RDF and RDF Schema.
- Please review the chapter on RDF/RDFS before starting

Origins of OWL



DAML = DARPA Agent Markup Language
OIL = Ontology Inference Layer

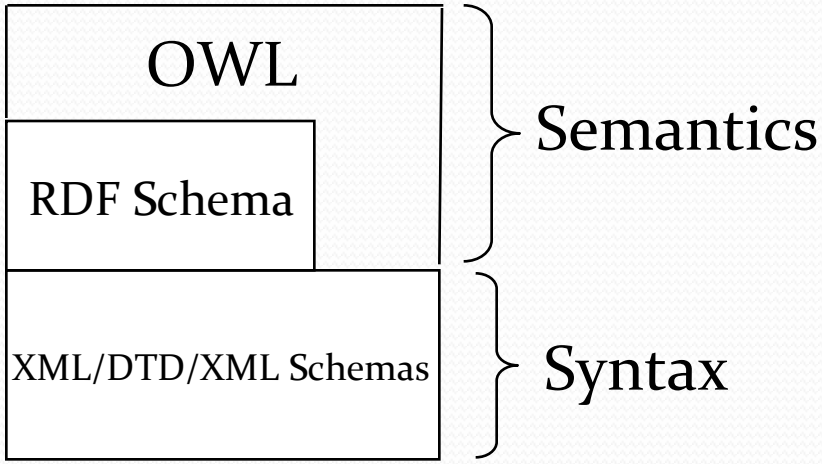
All were influenced by RDF

OWL is already a
W3C recommendation

Purpose of OWL

- The **purpose** of OWL is identical to RDF Schemas - to provide an XML vocabulary to define classes, their properties and their relationships among classes.
  - RDF Schema enables you to express very rudimentary relationships and has limited inferencing capability.
  - OWL enables you to express much richer relationships, thus yielding a much enhanced inferencing capability.
- A **benefit** of OWL is that it facilitates a much greater degree of inference making than you get with RDF Schemas.

OWL and RDF Schema enables machine-processable semantics



Overview

- OWL gives you an XML syntax to express statements about properties and classes, above and beyond what you can make with RDF Schema.
- In this chapter, we will learn about:
  - Using OWL to define classes.
  - Using OWL to define properties.
  - Using OWL to define relationships.
  - OWL statements that you can incorporate into your instance documents.

OWL = RDF Schema + more

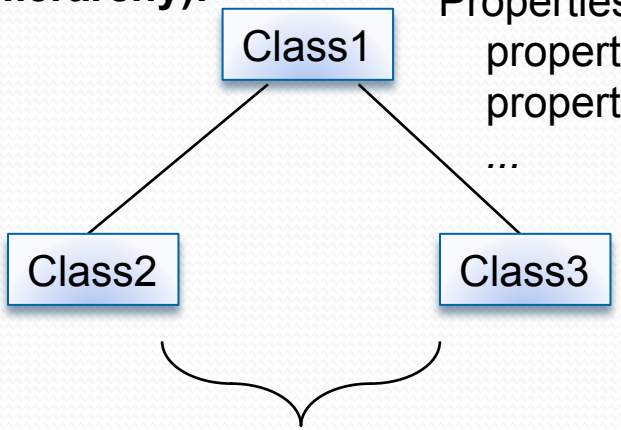
- Note: all of the elements/attributes provided by RDF and RDF Schema can be used when creating an OWL document.

  rdfs:Class
  rdf:property
  rdf:type
  rdfs:domain
  rdfs:range
  ……

- All of them can be used in OWL

Notations used in this chapter

Taxonomy (Class Hierarchy):



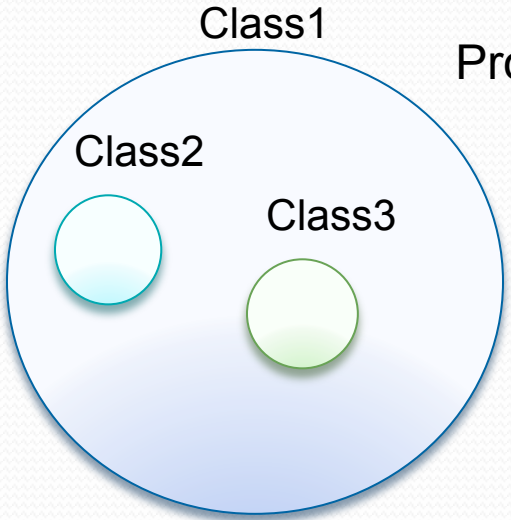
Properties:
property1: *Type1*
property2: *Type2*
...

These are the properties of Class1. The name of the property is shown (e.g., property1), and its range is shown in italics (e.g., *Type1*).

Class2 and Class3 are subclasses of Class1.

Venn Diagram:

An alternate notation to the above class hierarchy is to use a Venn diagram, as shown here.



Properties:
property1: *Type1*
property2: *Type2*
...

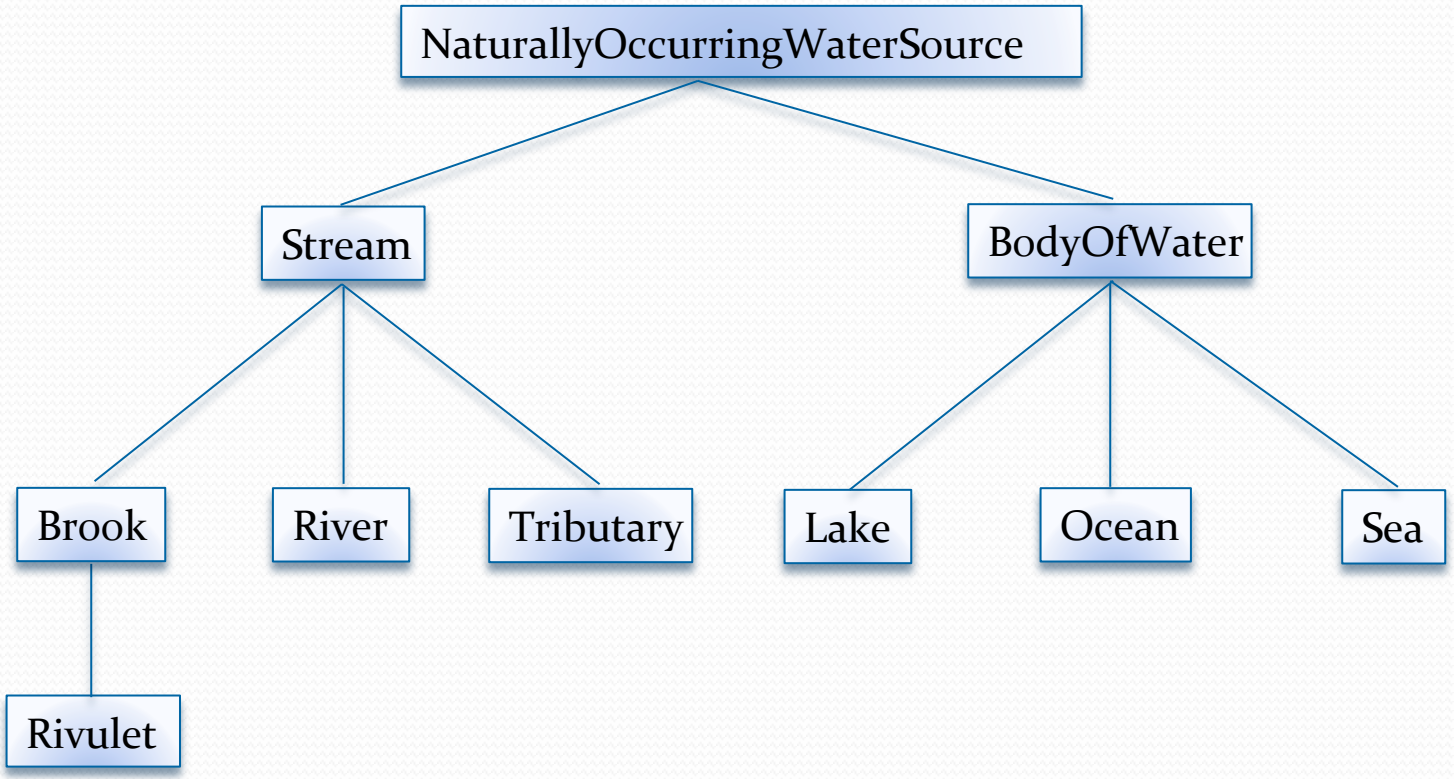
Continued

Notations used in this chapter

This notation is used to indicate that a person has only one birthplace location:

Further, a person's national insurance (NI) number is associated with only one person:

We will use a "water taxonomy" to explain OWL



Preview of OWL

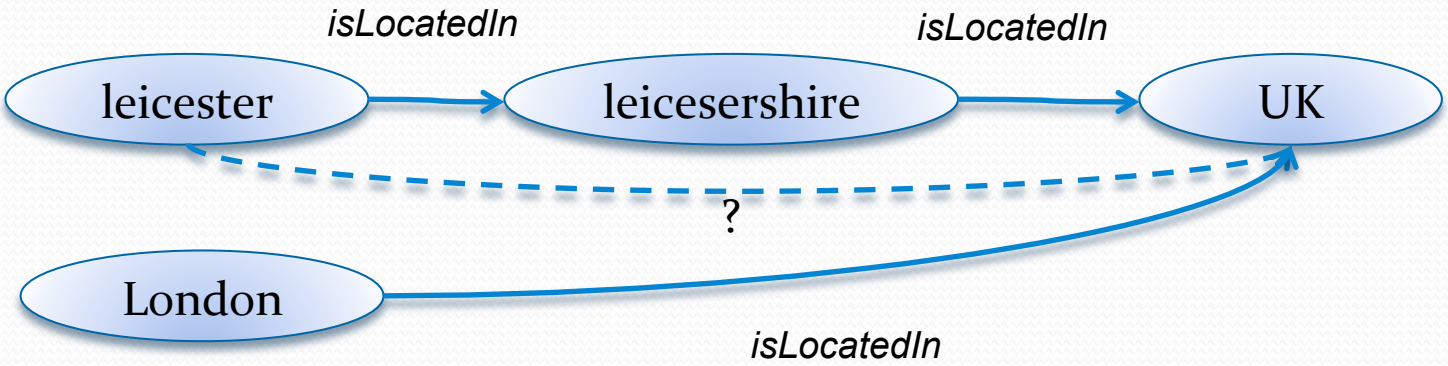
- Before getting into the details of OWL let's examine three examples that demonstrate some of the capabilities of OWL.

Why should we use OWL?

Example 1: Are they in the same country?

- Suppose we are trying to find out if Leicester and London are located in the same country.

```
<City rdf:ID="London">
  <isLocatedIn rdf:resource="#England"/>
</City>
<City rdf:ID="Leicester">
  <isLocatedIn rdf:resource="#Leicesetrshire"/>
</City>
<County rdf:ID="Leicestershire">
  <isLocatedIn rdf:resource="#England"/>
</County>
```



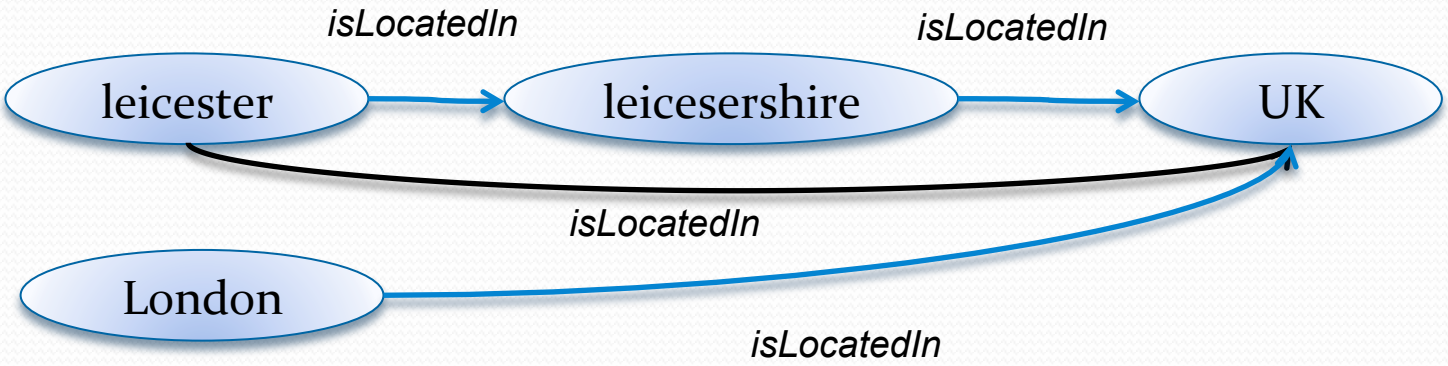
Example 1: Are they in the same country?

- In OWL, there's a special type of property called TransitiveProperty. property isLocated is declared as transitive.

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

enabling the following inference to be made:

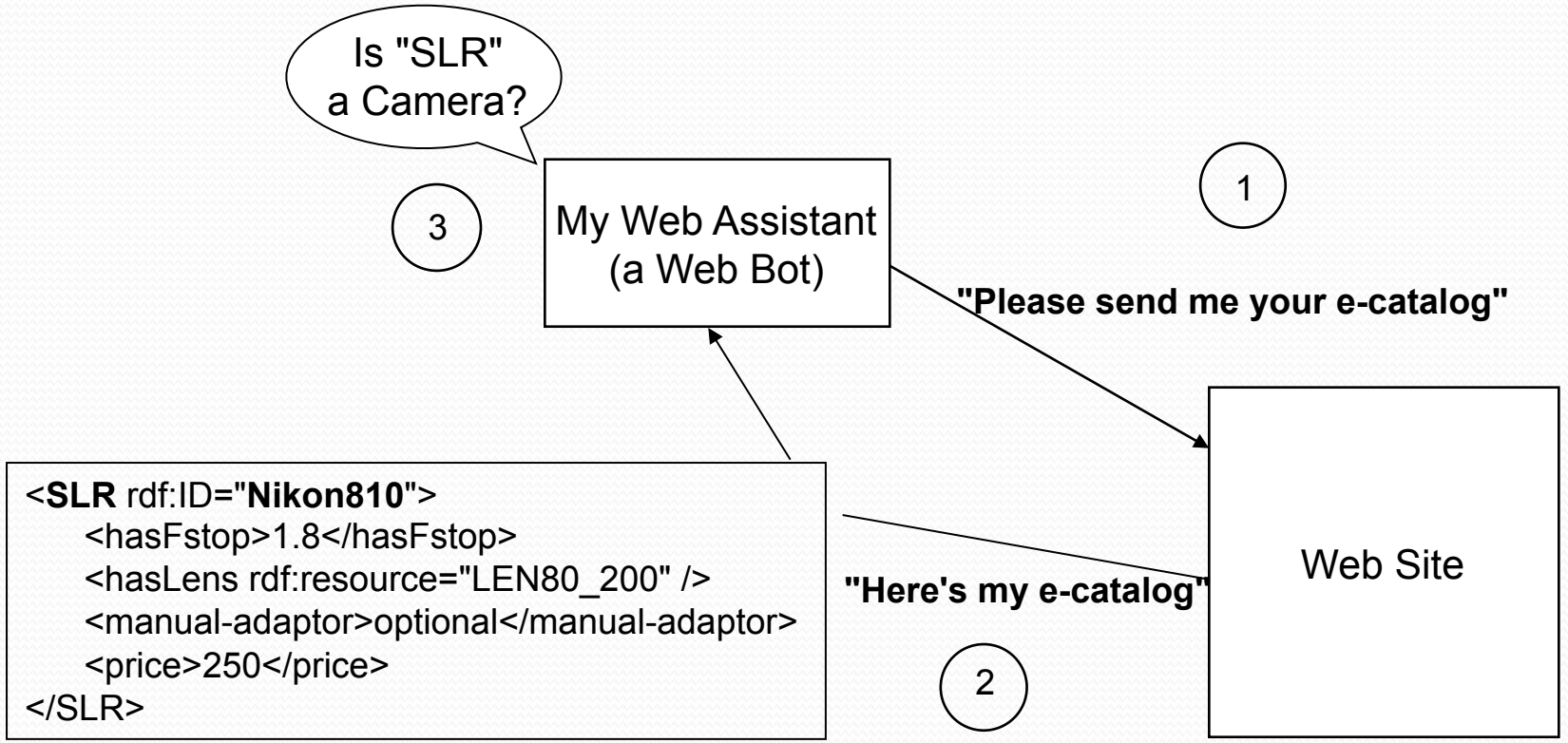
Inference: Leicester is located in England!



Lesson learned

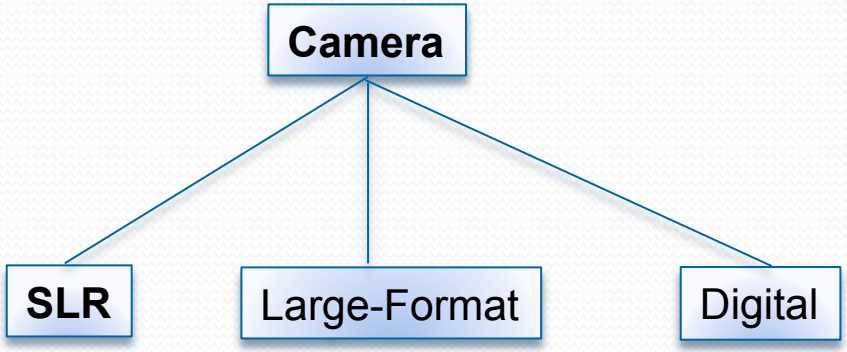
- OWL provides different types of properties with certain characteristics (e.g. TransitiveProperty), which can be used to discover implicit relationships.
  - if `p(a, b)` and `p (b, c)` than `p (a, c)`

Example 2: Using a Web Bot to Purchase a Camera



* A Web Bot is a software program which crawls the Web looking for information.

Camera OWL ontology



My Web Assistant program consults the Camera OWL Ontology. The Ontology shows how SLR is classified. The Ontology shows that SLR is a type (subclass) of Camera. Thus, my Web Assistant Bot dynamically realizes that:

Inference: The Nikon D810 is a Camera!

Lesson learned

- OWL provides elements to construct taxonomies (called class hierarchies). The taxonomies can be used to infer or deduce information.

Example 2: Who is Luke Skywalker's father?

- Upon scanning the Web, two documents were found which contain information about Luke Skywalker.

(1)

```
<Person rdf:about="http://www.person.org#Luke_Skywalker">
    <hasBiologicalFather rdf:about="http://www.emipre.mil#Darth_Vader"/>
</Person>
```

(2)

```
<Person rdf:about="http://www.person.org#Luke_Skywalker">
  <hasBiologicalFather rdf:about="http://www.jedi.org#Anakin_Skywalker"/>
</Person>
```

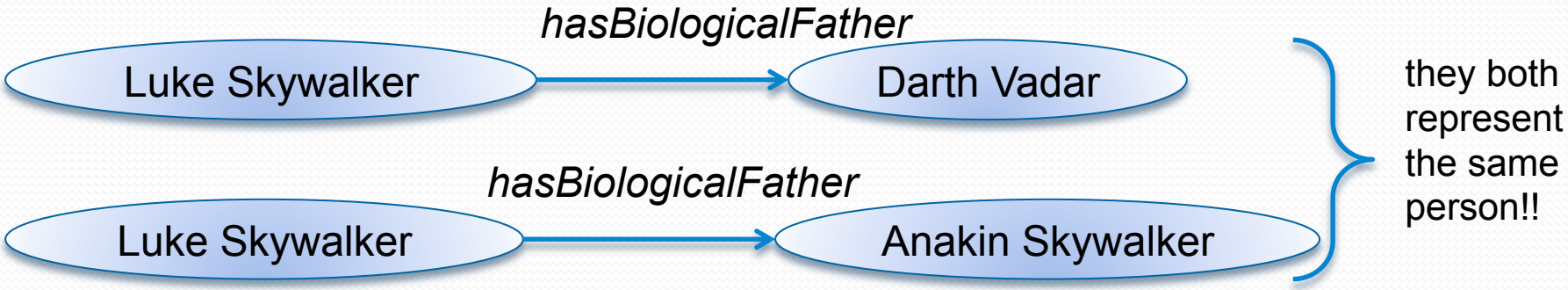
Question: who is Luke Skywalker's father?

Answer: both of them! In fact they are the same person

- The Person OWL Ontology indicates that a Person has only one biological father:

Thus, the Person OWL Ontology enables this inference to be made:

Inference : Darth Vadar and Anakin Skywalker are the same person!



Lesson learned

- In the example we saw that the Person Ontology defined this relationship:

This is read as: "A person has exactly one biological father."

This example is a specific instance of a general capability in OWL to specify that a subject Resource has exactly one value:

We saw in the example that such information can be used to make inferences.

OWL Terminology: properties that relate a resource to exactly one other resource are said to have a *cardinality=1*.

Review

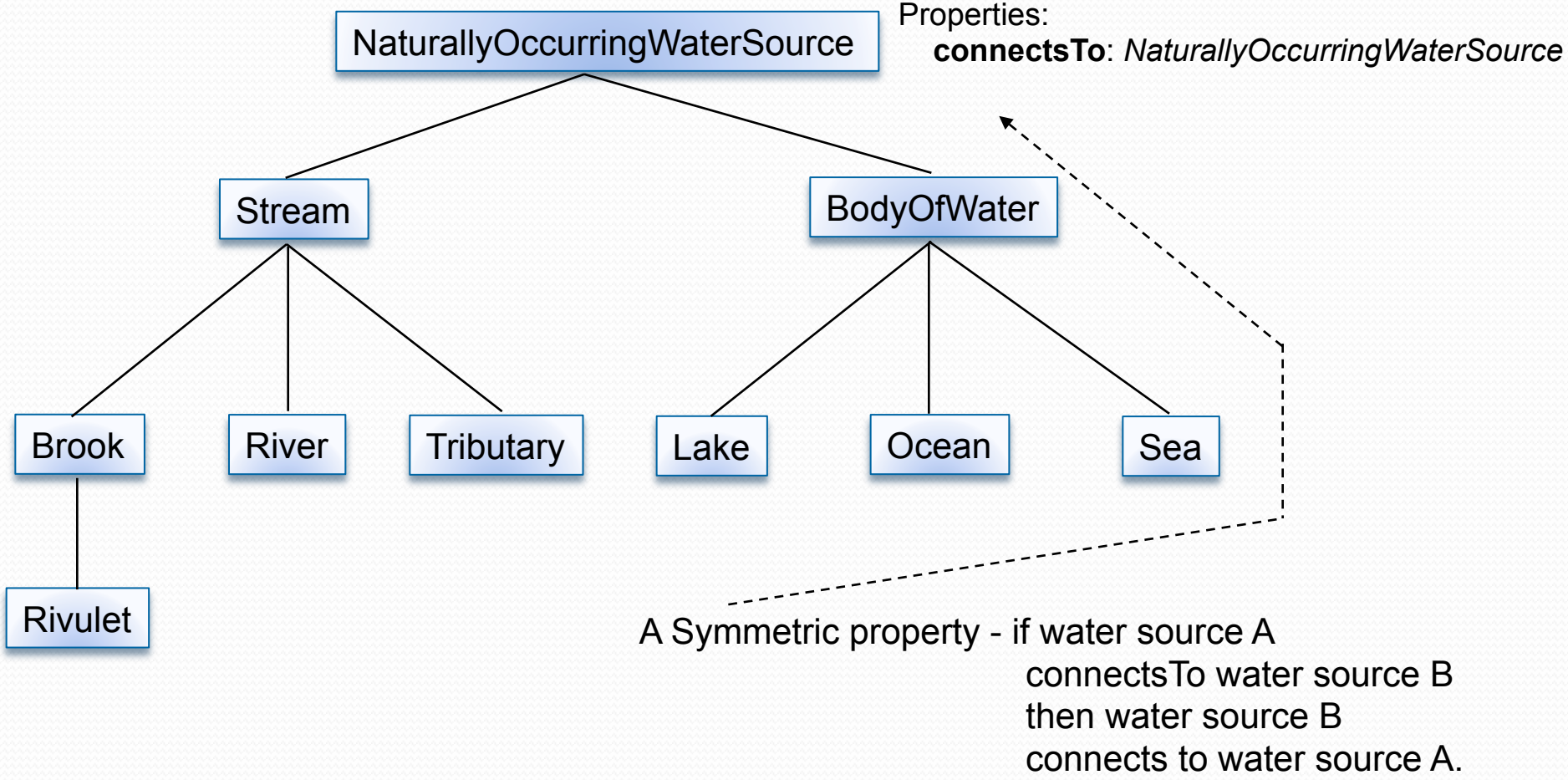
- The preceding examples demonstrated some of OWL's capabilities:
  - An OWL instance document can be enhanced with an OWL property to indicate that it is the same as another instance.
  - OWL provides the capability to construct taxonomies (class hierarchies). Such taxonomies can be used to dynamically understand how entities in an XML instance relate to other entities.
  - OWL provides the capability to specify that a subject can have only one value.
- By leveraging OWL, additional facts about your instance data can be dynamically ascertained. That is, OWL facilitates a dynamic understanding of the semantics of your data!
- Okay, that's it for the OWL preview. Now it's time to look at the entire suite of OWL capabilities ...

Using OWL to Define Properties

Symmetric Properties

- RDF Schema provides three ways to characterize a property:
  - **range:** use this to indicate the possible values for a property.
  - **domain:** use this to associate a property with a class.
  - **subPropertyOf:** use this to specialize a property.
- Note: OWL documents can also use **rdfs:range, rdfs:domain**, and **rdfs:subPropertyOf**.
- On the following slides we show the additional ways that OWL provides to characterize properties.
  - We will see that these additional property characteristics enable greater inference making.

Symmetric Properties



Symmetric Property

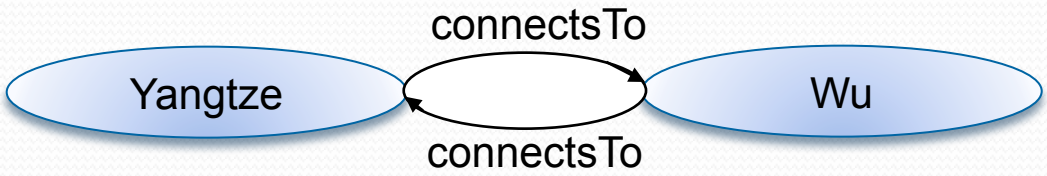
Assume that connectsTo has been *defined, in an OWL ontology, to be a Symmetric property*:

```
<?xml version="1.0"?>
<River rdf:ID="Yangtze"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns="http://www.geodesy.org/water/naturally-occurring#">
    <connectsTo>
        <River rdf:about="http://www.china.org/rivers#Wu"/>
    </connectsTo>
</River>
```

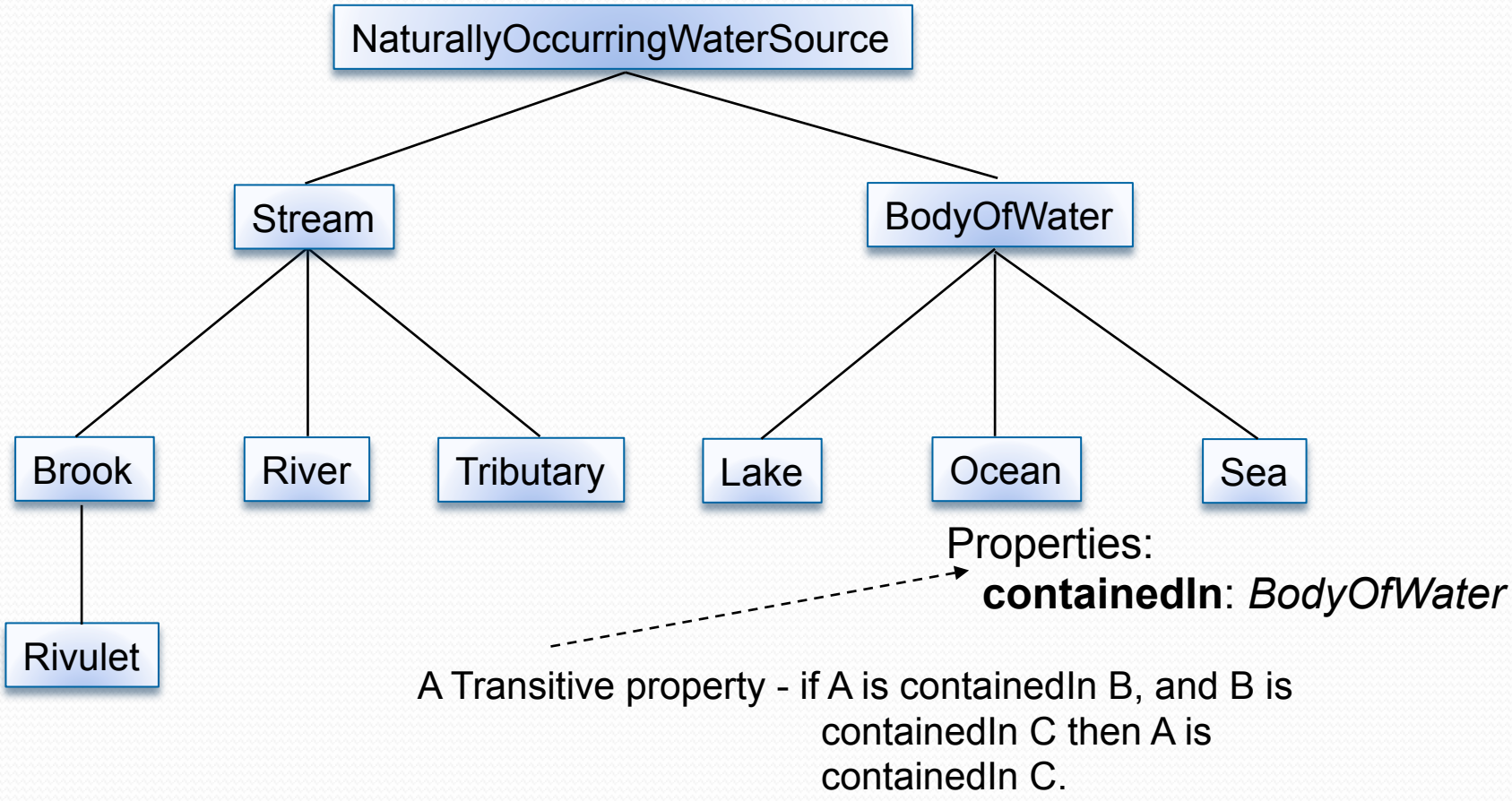
`Yangtze.rdf`

Since connectsTo has been defined to be a Symmetric property we can infer that:

The Wu River connectsTo the Yangtze River.



Transitive Property



Transitive Property

- Suppose that you retrieve these two documents from two different Web sites. One describes the `EastChinaSea` and the other describes the `ChinaSea`:

```
<?xml version="1.0"?>
<Sea rdf:ID="EastChinaSea"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns="http://www.geodesy.org/water/naturally-occurring#">
    <containedIn>
        <Sea rdf:about="http://www.china.gov#ChinaSea"/>
    </containedIn>
</Sea>
```

EastChinaSea.rdf

```
<?xml version="1.0"?>
<Sea rdf:about="http://www.china.gov#ChinaSea"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns="http://www.geodesy.org/water/naturally-occurring#">
    <containedIn>
        <Ocean rdf:about="http://www.geodesy.org#PacificOcean"/>
    </containedIn>
</Sea>
```

ChinaSea.rdf

If **containedIn** is defined to be a Transitive property then we can infer that:

The **EastChinaSea** is **containedIn** the **PacificOcean**.

Transitive Property

If containedIn is defined to be Transitive, we can infer that:

Functional Property

Suppose that there are two independent documents describing the Yangtze River:

```
<?xml version="1.0"?>
<River rdf:about="http://www.china.org/rivers#Yangtze"
          xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns="http://www.geodesy.org/water/naturally-occurring#">
    <emptiesInto rdf:resource="http://www.china.org/geography#EastChinaSea"/>
</River>
```

Yangtze-doc1.rdf

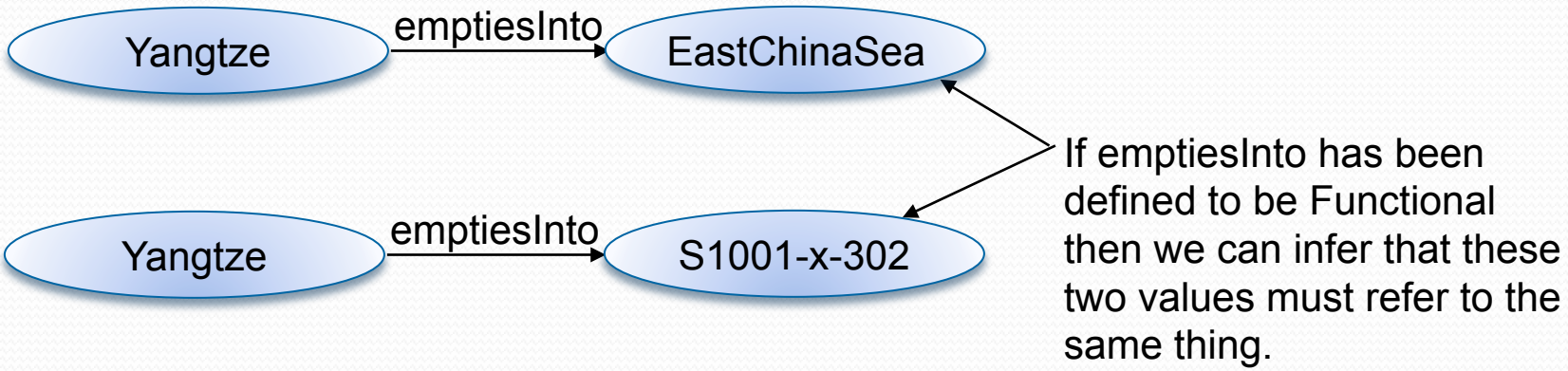
```
<?xml version="1.0"?>
<River rdf:about="http://www.china.org/rivers#Yangtze"
          xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns="http://www.geodesy.org/water/naturally-occurring#">
    <emptiesInto rdf:resource="http://www.national-geographic.org#S1001-x-302"/>
</River>
```

Yangtze-doc2.rdf

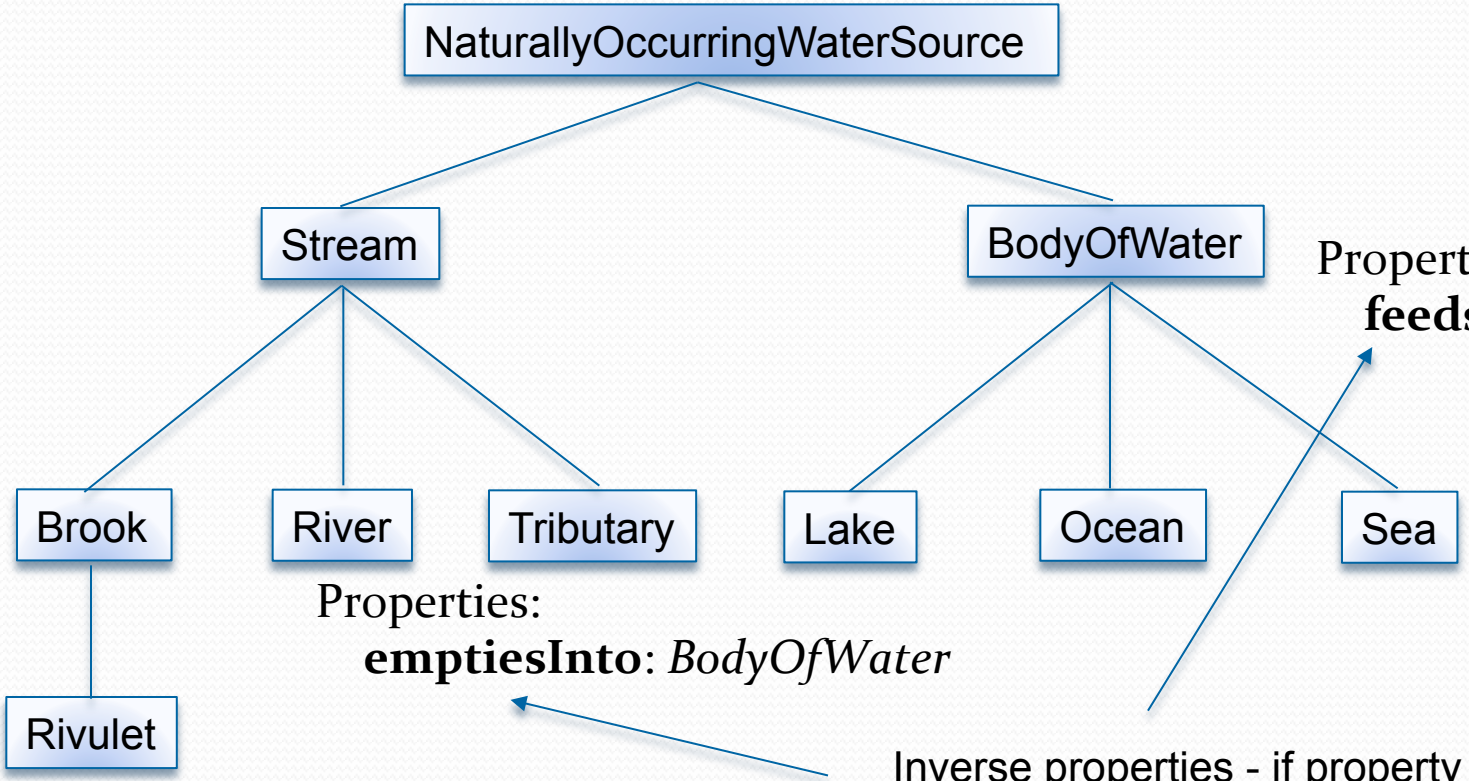
If emptiesInto is defined to be functional then we can infer that:

http://www.china.org/geography**#EastChinaSea** = http://www.national-geographic.org**#S1001-x-302**

Functional Property (cont.)



Inverse Properties



Inverse properties - if property P1 relates Resource 1 to Resource 2, then its Inverse property relates Resource 2 to Resource 1.

Inverse Properties

Consider this document:

```
<?xml version="1.0"?>
<River rdf:ID="Yangtze"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns="http://www.geodesy.org/water/naturally-occurring#">
    <emptiesInto rdf:resource="http://www.china.org/geography#EastChinaSea"/>
</River>
```

Yangtze.rdf

The above states that:

The Yangtze emptiesInto the EastChinaSea.

If emptiesInto and feedsFrom are defined to be Inverse properties then we can infer that:

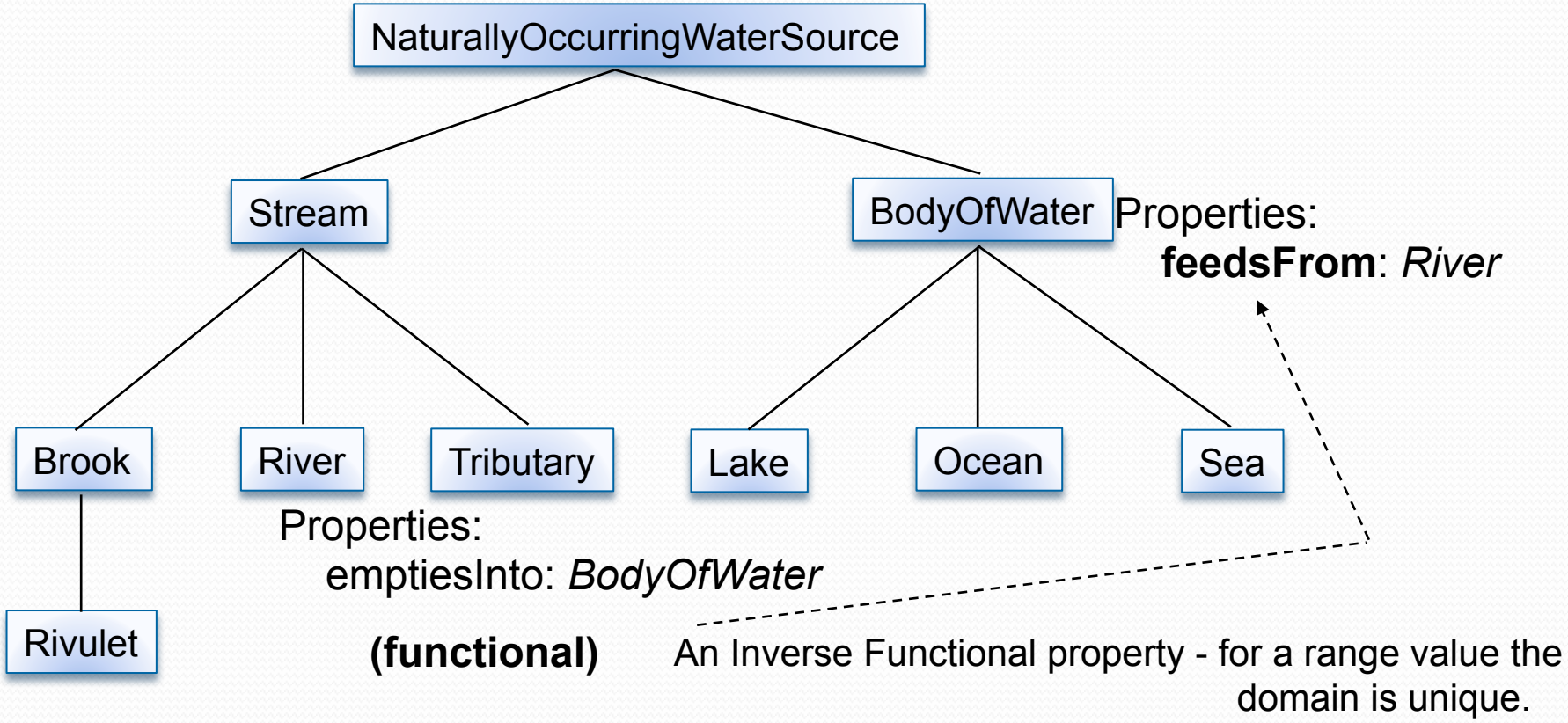
The EastChinaSea feedsFrom the Yangtze.

emptiesInto <---> feedsFrom
(Inverse Properties)

A specific instance:

The general case:

Inverse Functional Properties



Inverse Functional Property

These two independent documents discuss "feeding from" the Yangtze:

```
<?xml version="1.0"?>
<Sea  rdf:ID="EastChinaSea"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns="http://www.geodesy.org/water/naturally-occurring#">
    <feedsFrom>
       <River rdf:about="http://www.china.org/rivers#Yangtze"/>
    </feedsFrom>
</Sea>
```

EastChinaSea.rdf

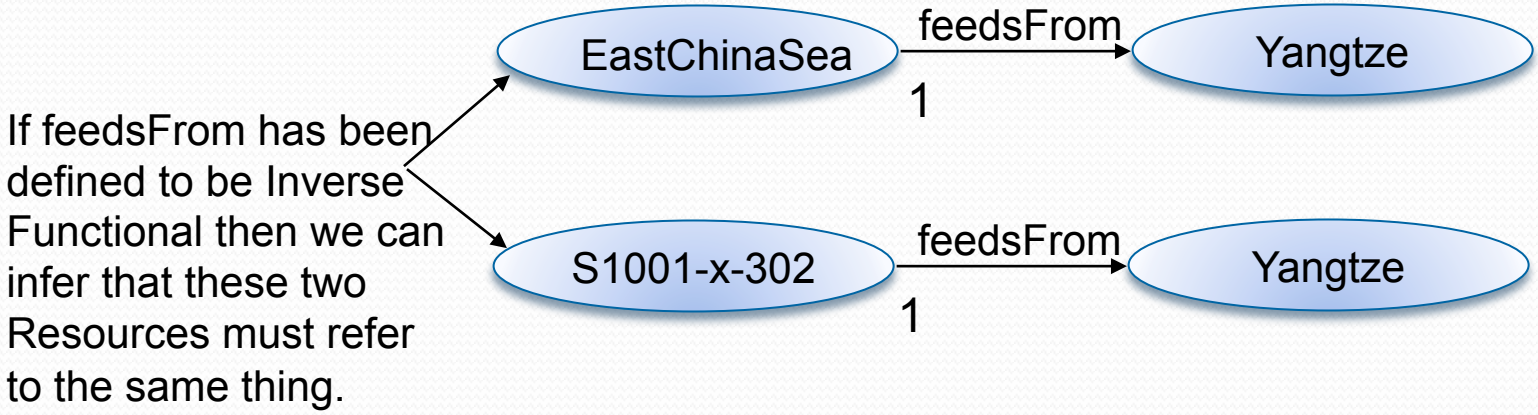
```
<?xml version="1.0"?>
<Sea rdf:ID="S1001-x-302"
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns="http://www.geodesy.org/water/naturally-occurring#">
    <feedsFrom>
       <River rdf:about="http://www.china.org/rivers#Yangtze"/>
    </feedsFrom>
</Sea>
```

S1001-x-302.rdf

Inverse Functional Property (cont.)

If **feedsFrom** has been defined to be **InverseFunctional** then we can infer that:

`EastChinaSea` **=** `S1001-x-302`.



Time for Syntax!

- On the previous slides we have seen the different ways that OWL provides to characterize properties.
- Now let's look at the OWL syntax for expressing these property characteristics.

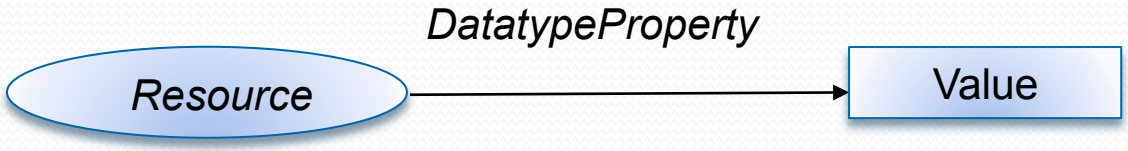
Defining Properties in OWL

- Recall that with RDF Schema the **rdf:Property** was used for both:
  - relating a **Resource** to another **Resource**
    - Example: The **emptiesInto** property relates a **River** to a **BodyOfWater**.
  - relating a **Resource** to an **rdfs:Literal** or a datatype
    - Example: The length property relates a River to a **xsd:nonNegativeInteger.**
- OWL decided that these are two classes of properties, and thus each should have its own class:
  - **owl:ObjectProperty** is used to relate a Resource to another Resource
  - **owl:DatatypeProperty** is used to relate a Resource to an **rdfs:Literal** or an XML Schema built-in datatype

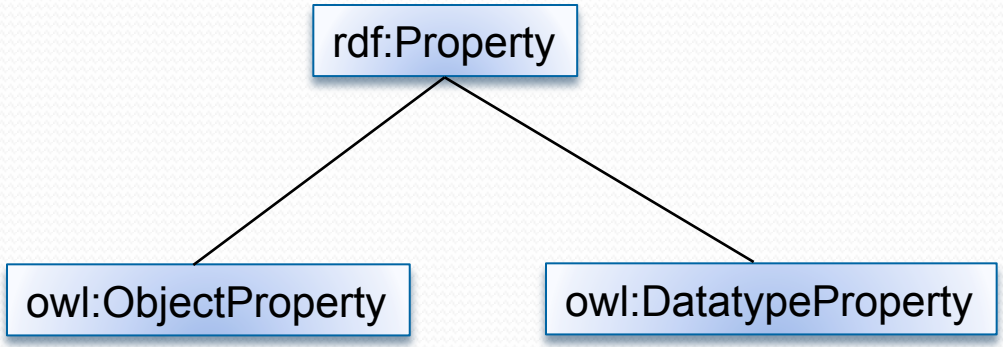
ObjectProperty vs. DatatypeProperty

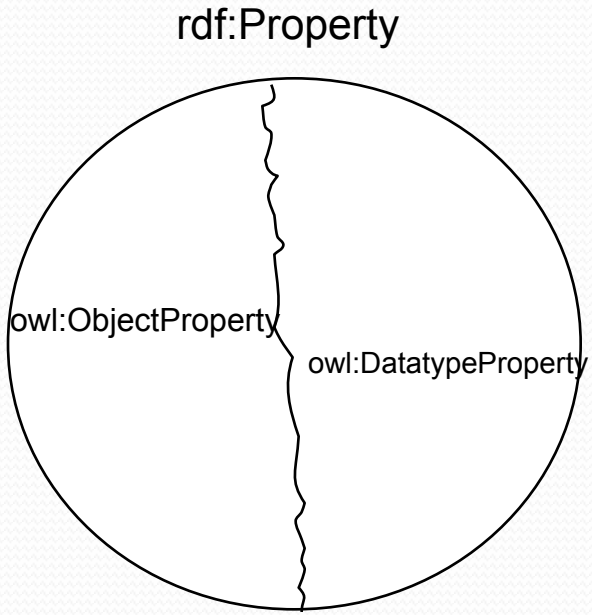
An ObjectProperty relates one Resource to another Resource:

A DatatypeProperty relates a Resource to a Literal or an XML Schema datatype:



owl:ObjectProperty and owl:DatatypeProperty are subclasses of rdf:Property





Defining Properties in OWL vs. RDF Schema

RDFS

```
<rdf:Property rdf:ID="emptiesInto">
        <rdfs:domain rdf:resource="#River"/>
        <rdfs:range rdf:resource="#BodyOfWater"/>
</rdf:Property>
```

```
<rdf:Property rdf:ID="length">
    <rdfs:domain rdf:resource="#River"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#nonNegativeInteger"/>
</rdf:Property>
```

OWL

```
<owl:ObjectProperty rdf:ID="emptiesInto">
        <rdfs:domain rdf:resource="#River"/>
        <rdfs:range rdf:resource="#BodyOfWater"/>
</owl:ObjectProperty>
```

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

The OWL Namespace

```
<?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#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xml:base="http://www.geodesy.org/water/naturally-occurring">

    <owl:ObjectProperty rdf:ID="emptiesInto">
        <rdfs:domain rdf:resource="#River"/>
        <rdfs:range rdf:resource="#BodyOfWater"/>
    </owl:ObjectProperty>

    ...

</rdf:RDF>
```

naturally-occurring.owl (snippet)

What is the URI for the properties and classes defined by an OWL document?

What is the full URI for the emptiesInto property in this OWL document:

```
<?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#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xml:base="http://www.geodesy.org/water/naturally-occurring">

   <owl:ObjectProperty rdf:ID="emptiesInto">
      <rdfs:domain rdf:resource="#River"/>
      <rdfs:range rdf:resource="#BodyOfWater"/>
   </owl:ObjectProperty>

   ...

</rdf:RDF>
```

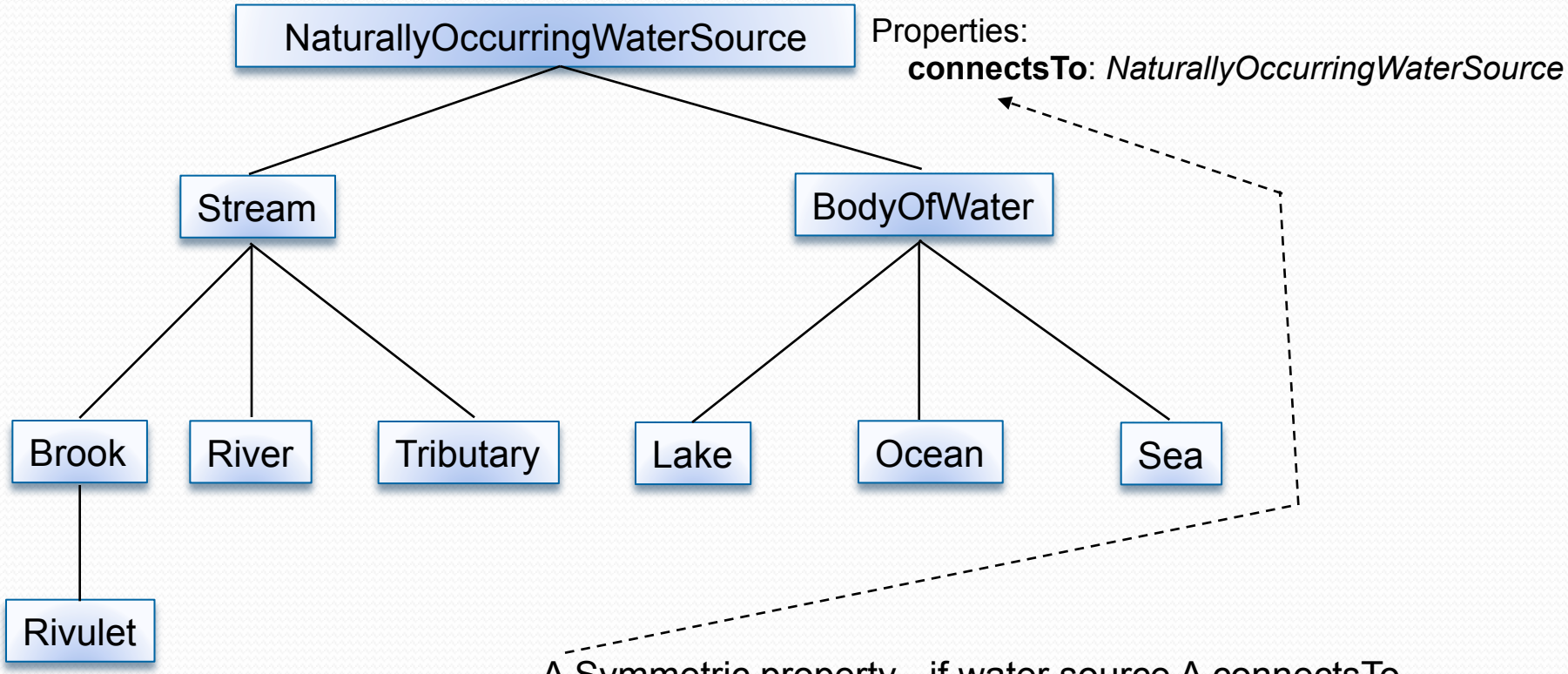
naturally-occurring.owl (snippet)

The URI for an identifier (i.e., an rdf:ID value) is the concatenation of the xml:base value (or the document URL if there is no xml:base) with "#" and the identifier. Thus, the complete URI for the above emptiesInto property is:

http://www.geodesy.org/water/naturally-occurring#emptiesInto

Note: These are the same rules that RDF Schema uses for determining the URI.

Defining Symmetric Properties



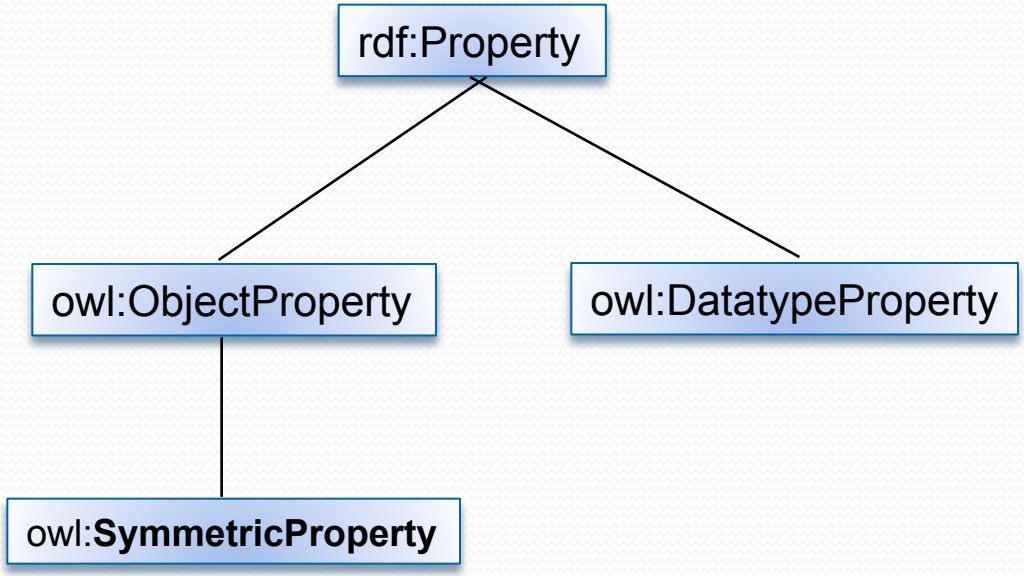
Syntax for indicating that a property is Symmetric

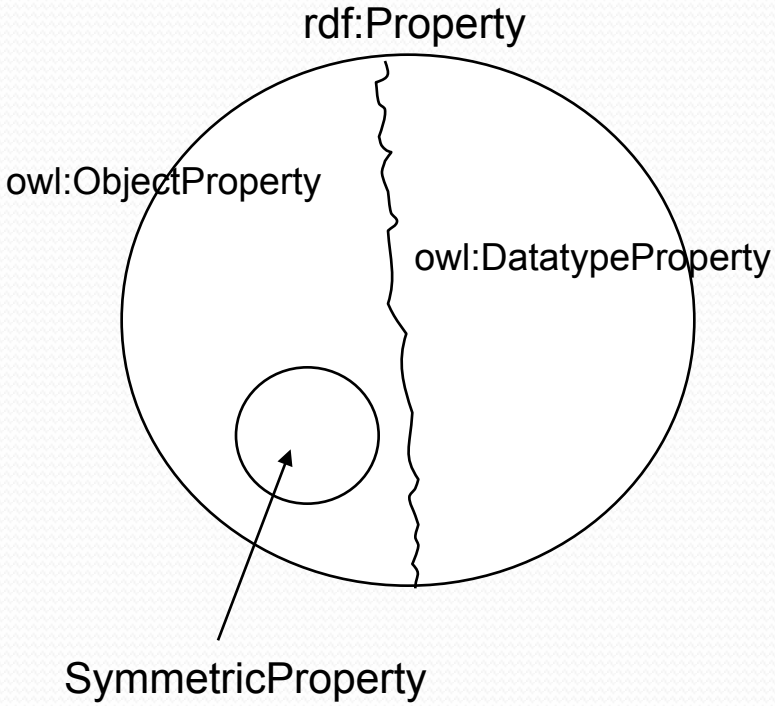
```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xml:base="http://www.geodesy.org/water/naturally-occurring">

   <owl:ObjectProperty rdf:ID="connectsTo">
      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#SymmetricProperty"/>
      <rdfs:domain rdf:resource="#NaturallyOccurringWaterSource"/>
      <rdfs:range rdf:resource="#NaturallyOccurringWaterSource"/>
   </owl:ObjectProperty>

   ...

</rdf:RDF>
```

naturally-occurring.owl (snippet)

Read this as: "connectsTo is an ObjectProperty. Specifically, it is a Symmetric Object Property."

owl:SymmetricProperty is a subclass of owl:ObjectProperty





Consequently, the range of a SymmetricProperty can only be a Resource, i.e., the range cannot be a Literal or a datatype.

Equivalent!

Read this as: "connectsTo is an **ObjectProperty**. Specifically, it is a Symmetric Object Property."

1

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

Read this as: "connectsTo is a **SymmetricProperty**."

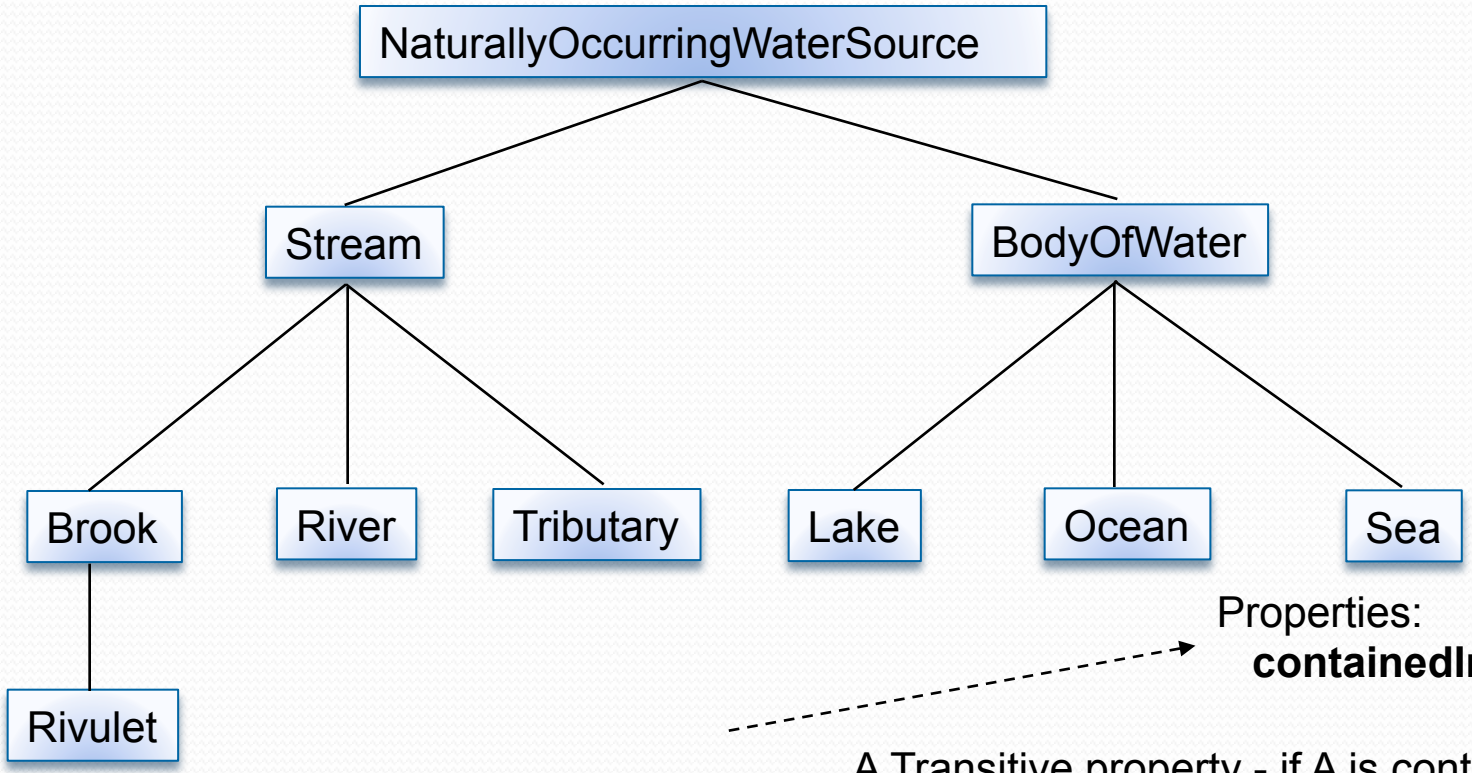
2

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

Question: Why would you ever use the first form? The second form seems a lot more straightforward. Right?

Answer: In this example, you are correct, the second form is more straightforward. However, you will see in a moment that we can define a property to be of several types, e.g., Symmetric **and** Functional. In that case it may be more straightforward to use the first form (and use multiple rdf:type elements).

Defining Transitive Properties



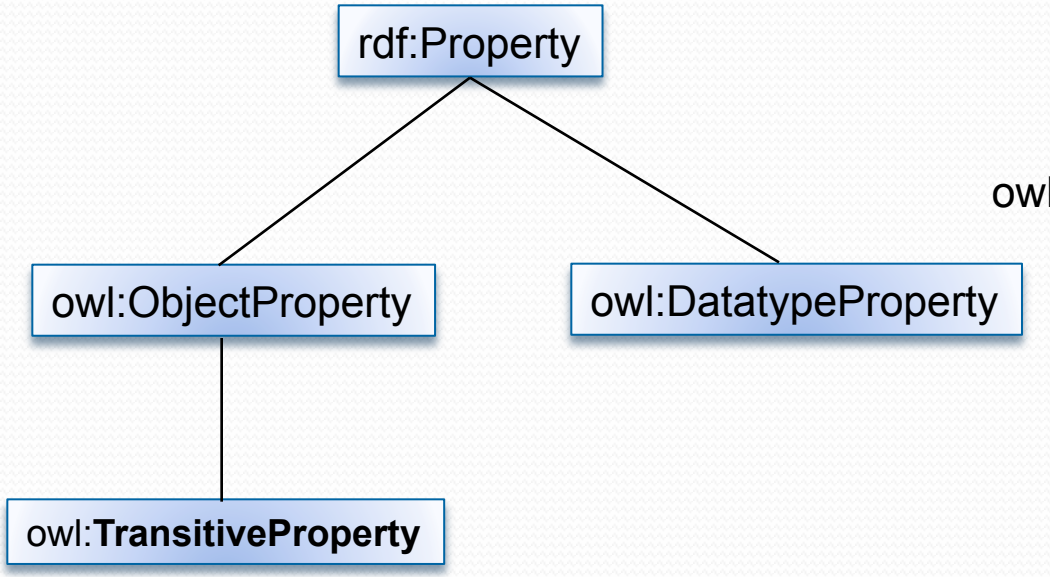
A Transitive property - if A is containedIn B, and B is containedIn C then A is containedIn C.

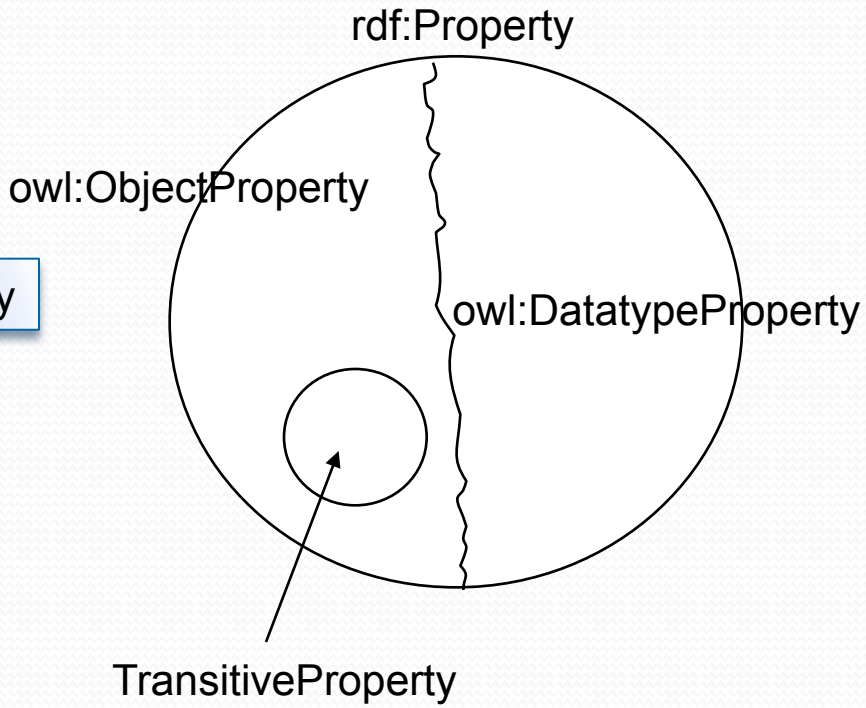
Syntax for indicating that a property is Transitive

```
<?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#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xml:base="http://www.geodesy.org/water/naturally-occurring">

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

    ...

</rdf:RDF>
```

naturally-occurring.owl (snippet)

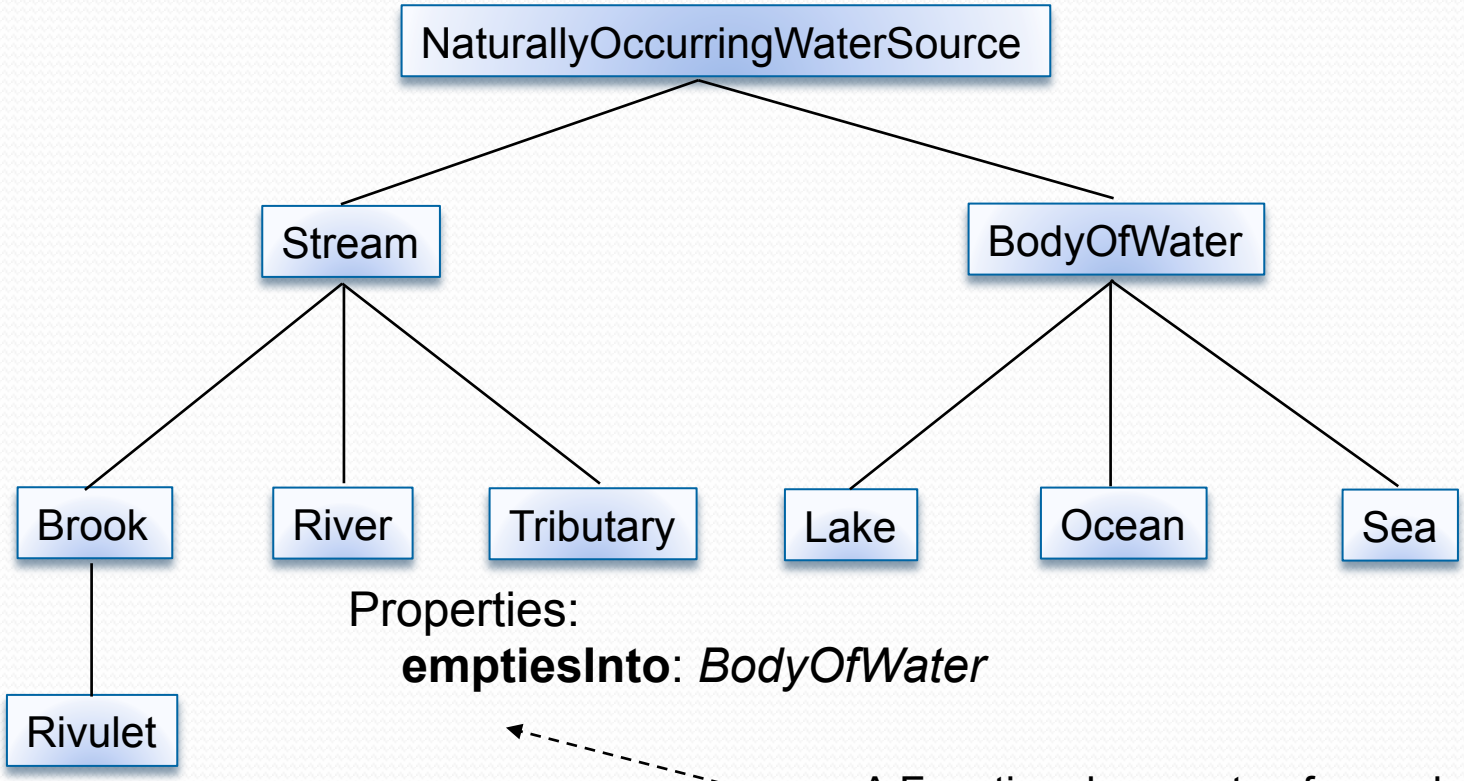
owl:TransitiveProperty is a subclass of owl:ObjectProperty





Consequently, the range of a TransitiveProperty can only be a Resource, i.e., the range cannot be a Literal or a datatype.

Defining Functional Properties

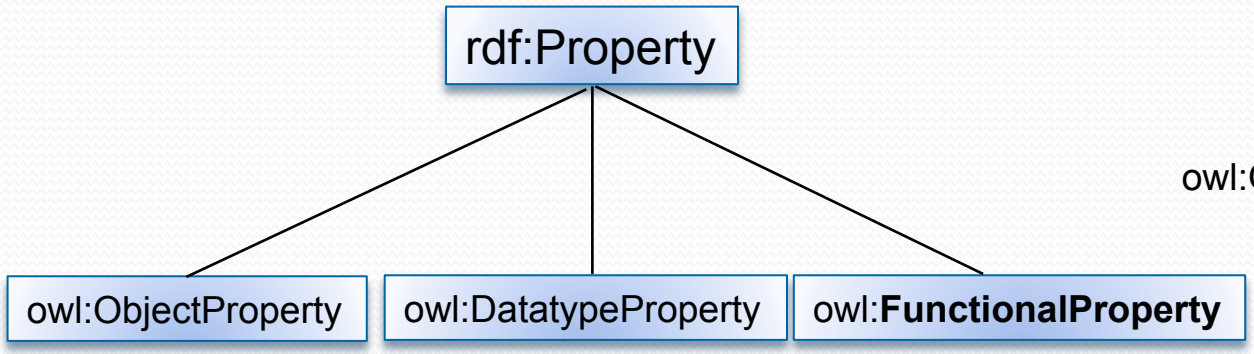


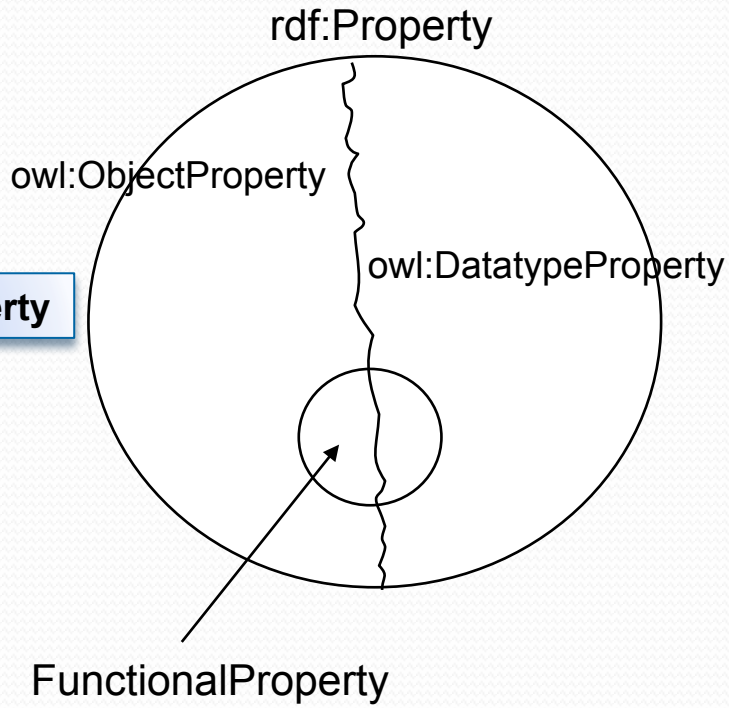
Syntax for indicating that a property is Functional

```
<?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#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xml:base="http://www.geodesy.org/water/naturally-occurring">

  <owl:ObjectProperty rdf:ID="emptiesInto">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="#River"/>
    <rdfs:range rdf:resource="#BodyOfWater"/>
  </owl:ObjectProperty>

  ...

</rdf:RDF>
```

naturally-occurring.owl (snippet)

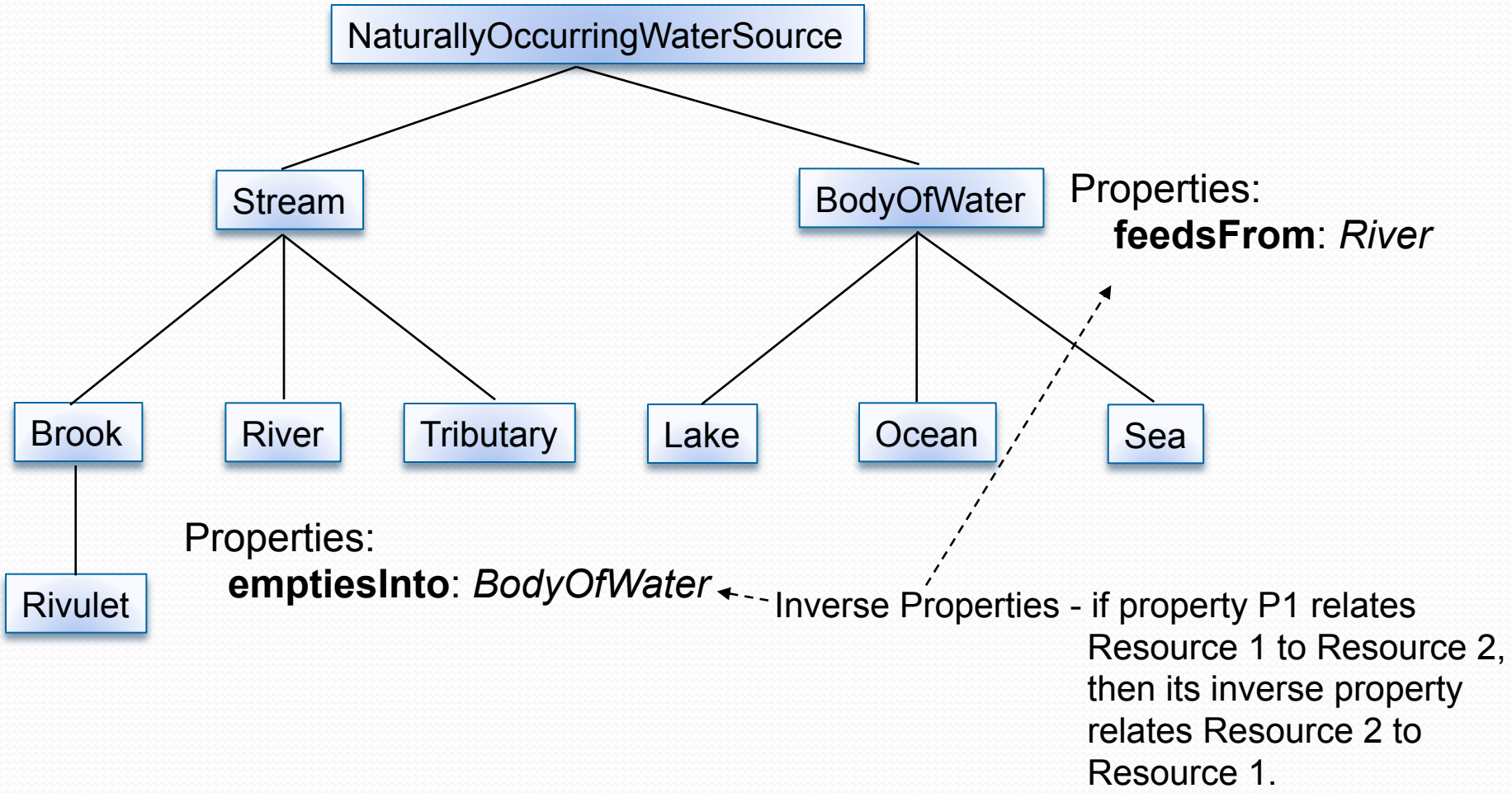
owl:FunctionalProperty is a subclass of rdf:Property





Consequently, the range of a FunctionalProperty can be either a Resource or a Literal or a datatype.

Defining Inverse Properties



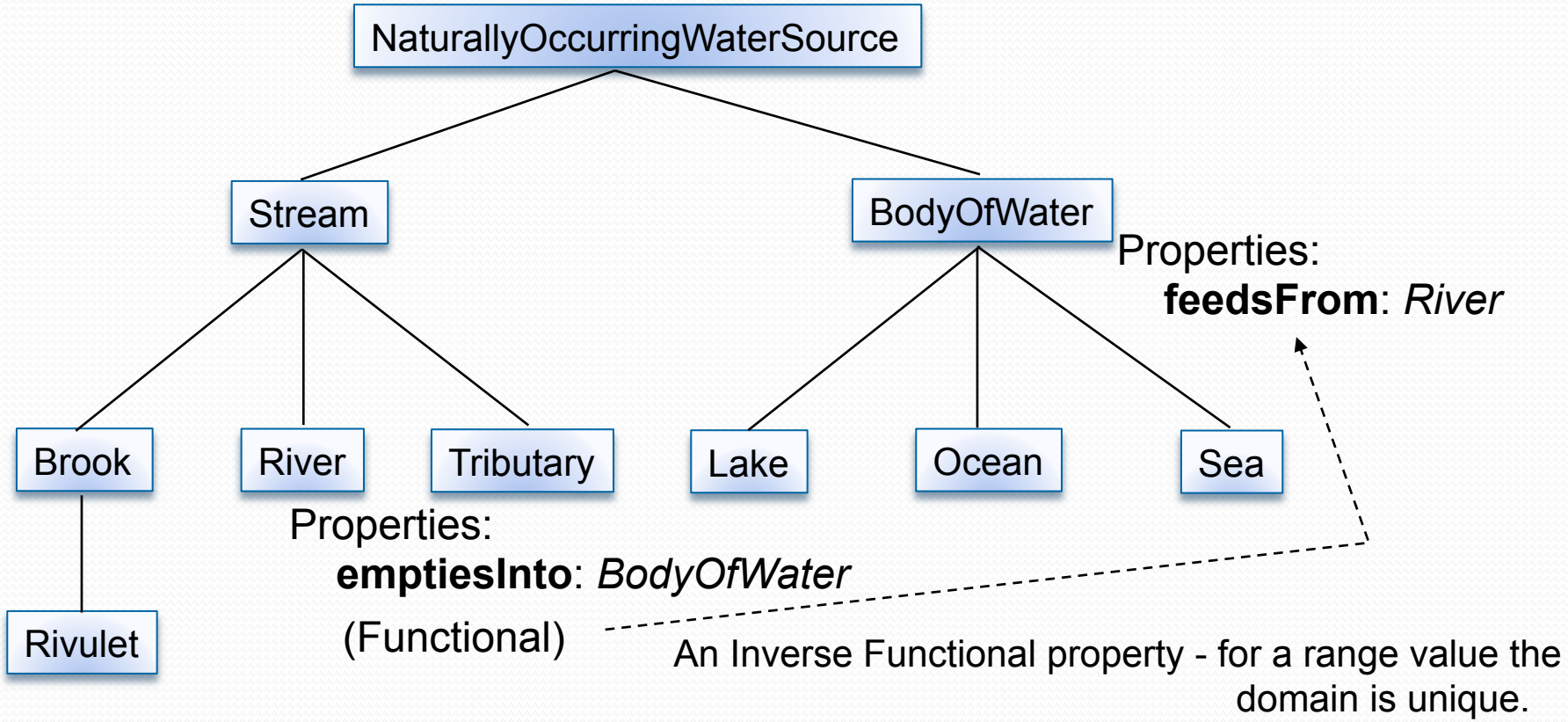
Syntax for indicating that a property is the inverse of another property

```
<?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#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xml:base="http://www.geodesy.org/water/naturally-occurring">

  <owl:ObjectProperty rdf:ID="emptiesInto">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="#River"/>
    <rdfs:range rdf:resource="#BodyOfWater"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="feedsFrom">
    <owl:inverseOf rdf:resource="#emptiesInto"/>
    <rdfs:domain rdf:resource="#BodyOfWater"/>
    <rdfs:range rdf:resource="#River"/>
  </owl:ObjectProperty>

  ...

</rdf:RDF>
```

naturally-occurring.owl (snippet)

Notice that the values for domain and range are flipped from that in emptiesInto.

Defining Inverse Functional Properties

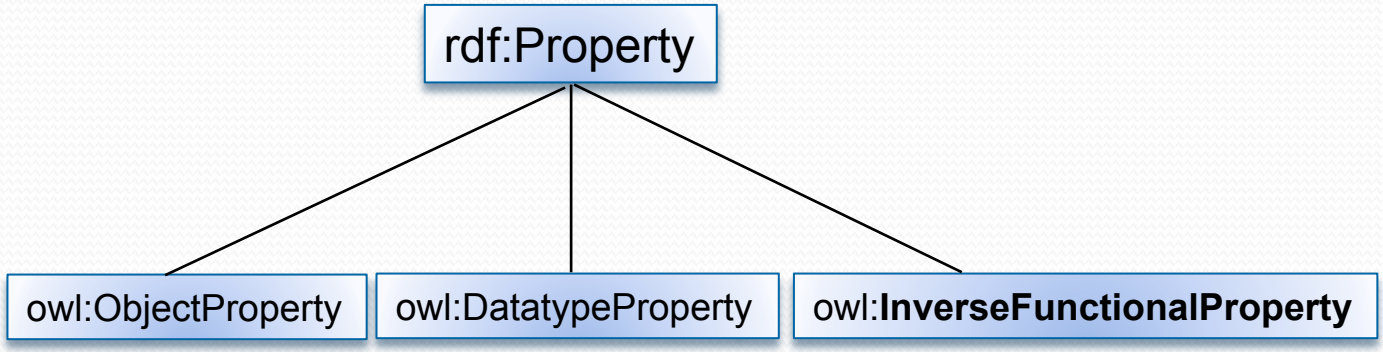


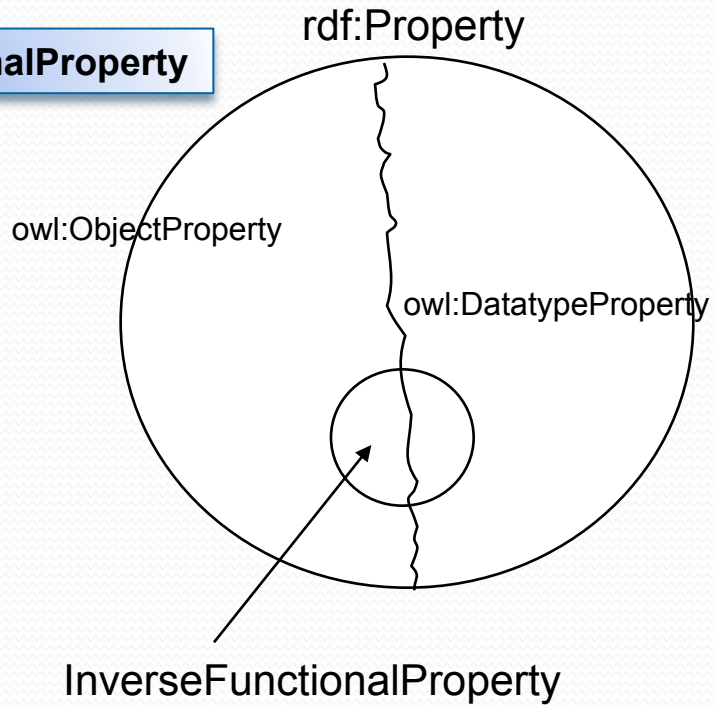
Syntax for indicating that a property is Inverse Functional

```
<?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#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xml:base="http://www.geodesy.org/water/naturally-occurring">

  <owl:ObjectProperty rdf:ID="emptiesInto">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="#River"/>
    <rdfs:range rdf:resource="#BodyOfWater"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="feedsFrom">
    <rdf:type
      rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
    <owl:inverseOf rdf:resource="#emptiesInto"/>
    <rdfs:domain rdf:resource="#BodyOfWater"/>
    <rdfs:range rdf:resource="#River"/>
  </owl:ObjectProperty>

  ...

</rdf:RDF>
```

naturally-occurring.owl (snippet)

owl:InverseFunctionalProperty is a subclass of rdf:Property





Consequently, the range of an InverseFunctionalProperty can be either a Resource or a Literal or a datatype.

An Overview of new OWL2 Properties

- OWL2 introduces several new properties
  - ReflexiveProperty
  - IrreflexiveProperty
  - AsymmetricProperty
  - EquivalentProperty
  - DisjointProperty

ReflexiveProperty

- An Reflexive property states that the property is reflexive that is, each individual is connected by this property to itself.

Property **knows** is a ReflexiveProperty because every person knows himself/herself

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

IrreflexiveProperty

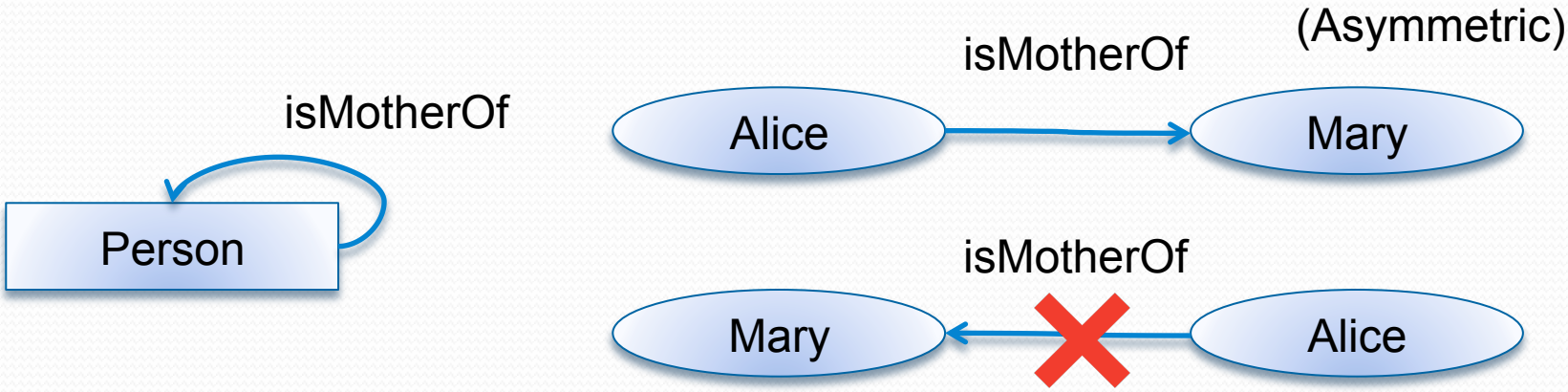
- An object irreflexive property states that the property irreflexive — that is, NO individual is connected by this property to itself

Property **isMotherOf** is an IrreflexiveProperty because no one can be the mother of herself

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

AsymmetricProperty

- If an individual x is connected by an AsymmetricProperty to an individual y, then y cannot be connected by this property to x.

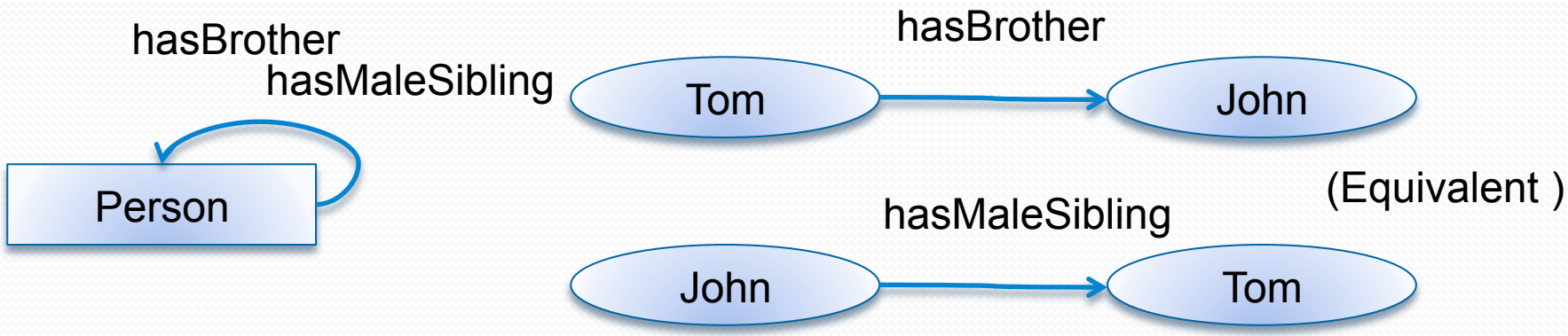


Property **knows** is also an AsymmetricProperty because if A is the mother B then B cannot be the mother of A.

```
<owl:ObjectProperty rdf:ID="isMotherOf">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#IrreflexiveProperty"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#AsymmetricProperty"/>
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="#Person"/>
</owl:ObjectProperty>
```

Equivalent Property

- If p1 is the an EquivantProperty of p2, then p1 and p2 are semantically equivalent to each other.

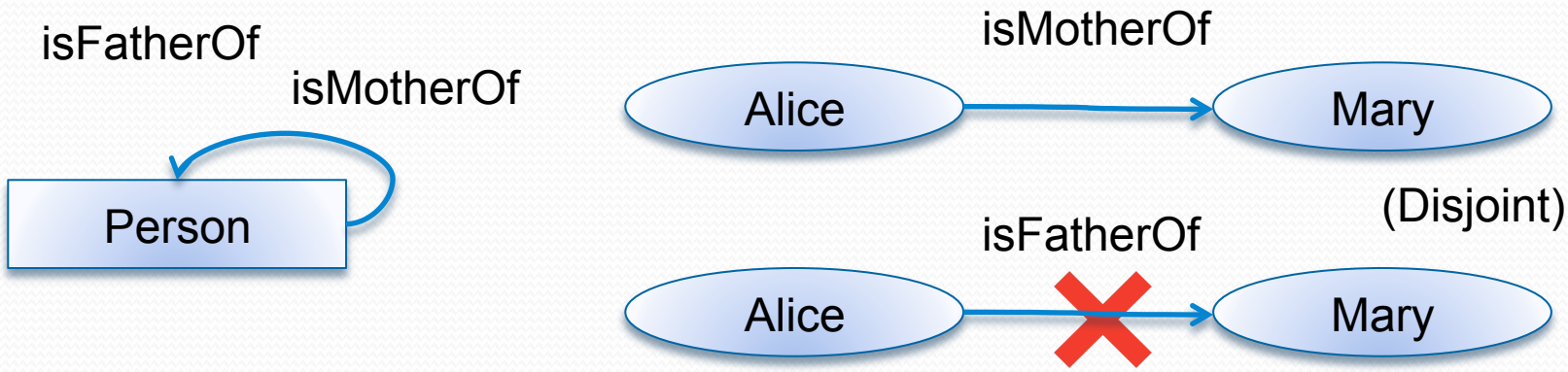


hasMaleSibling and hasBrother are semantically equivalent.

```
<owl:ObjectProperty rdf:ID="hasBrother">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#SymmetricProperty"/>
    <owl:equivalentProperty rdf:resource="#hasMaleSibling"/>
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="#Person"/>
</owl:ObjectProperty>
```

Disjoint Property

- NO individual x can be connected to an individual y by both property P1 and P2 (assuming P1 and P2 are not equivalent property of each other).



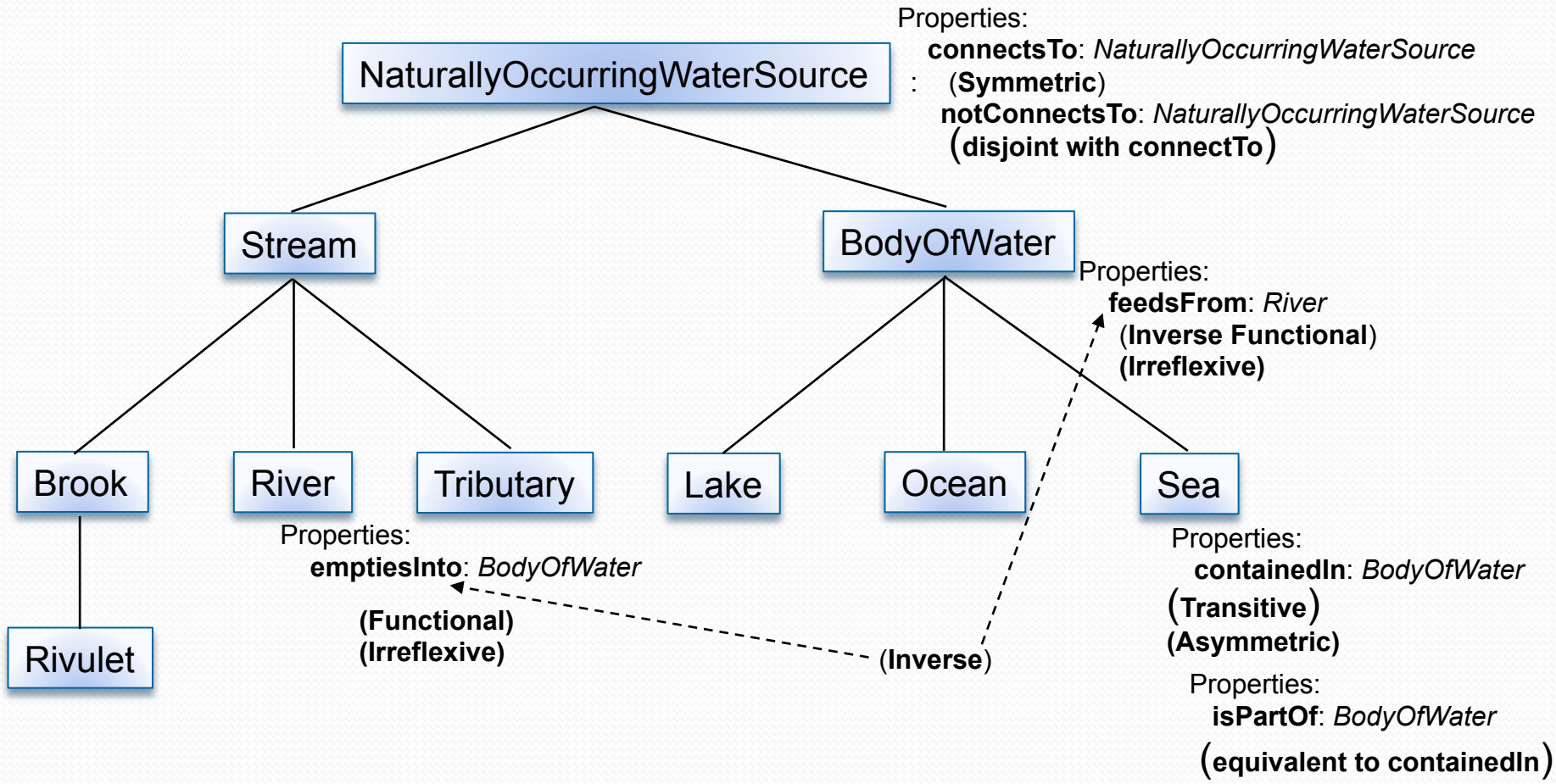
Property **knows** is also a ReflexiveProperty because is A is the mother B, B cannot be mother of A.

```
<owl:ObjectProperty rdf:ID="isMotherOf">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#IrreflexiveProperty"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#AsymmetricProperty"/>
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="#Person"/>
 </owl:ObjectProperty>
```

Summary of the different ways to characterize properties

- In the preceding slides we have seen the different ways of characterizing properties. We saw that a property may be defined to be:
  - A Symmetric property.
  - A Transitive property.
  - A Functional property.
  - A Reflexive Property
  - An Irreflexive Property
  - An Asymmetric Property
  - An EquivalentProperty
  - A DisjointProperty
  - The Inverse of another property.
  - An Inverse Functional property.

Summary of Properties for the Water Taxonomy



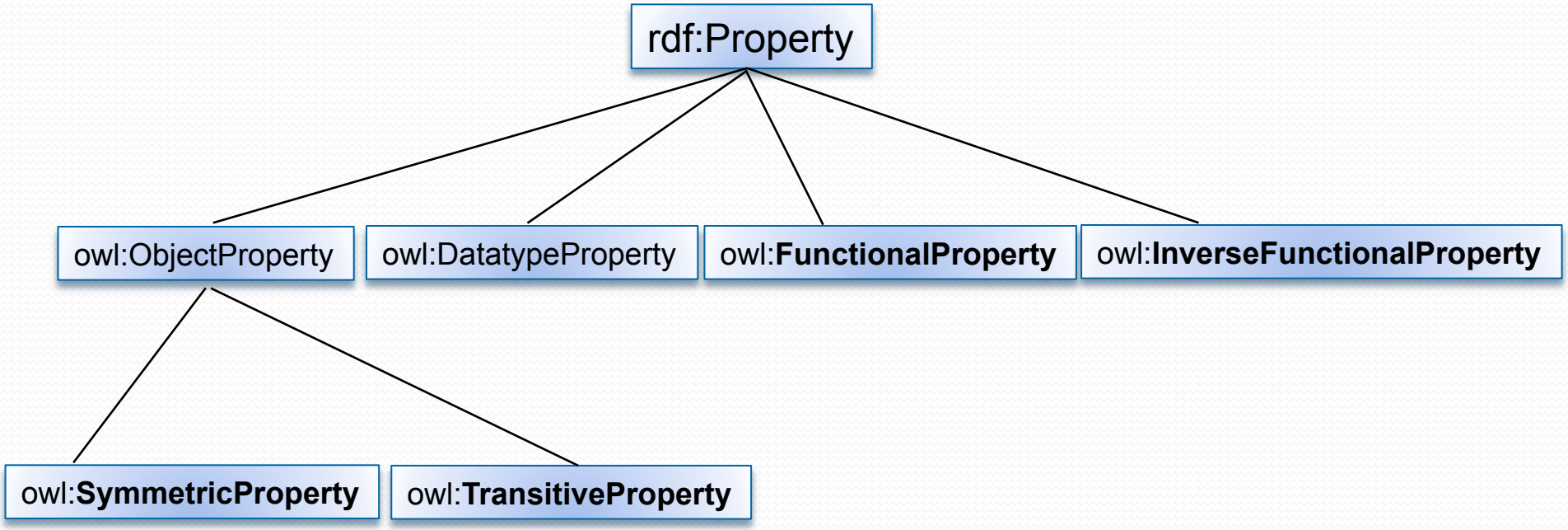
Inferences we can make now that we have characterized the properties

```
<?xml version="1.0"?>
<River rdf:ID="Yangtze"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns="http://www.geodesy.org/water/naturally-occurring#">
    <emptiesInto rdf:resource="http://www.china.org/geography#EastChinaSea"/>
    <connectsTo rdf:resource="http://www.china.org/rivers#Wu"/>
</River>
```

We can infer that:

1. The EastChinaSea feedsFrom the Yangtze. (Since emptiesInto is the inverse of feedsFrom)
2. The Wu connectsTo the Yangtze. (Since connectsTo is symmetric)
3. The EastChinaSea is a BodyOfWater. (Since the range of emptiesInto is a BodyOfWater.
4. The Wu is a NaturallyOccurringWaterSource. (Since the range of connectsTo is NaturallyOccurringWaterSource)

Hierarchy of the property classes



Consequences:

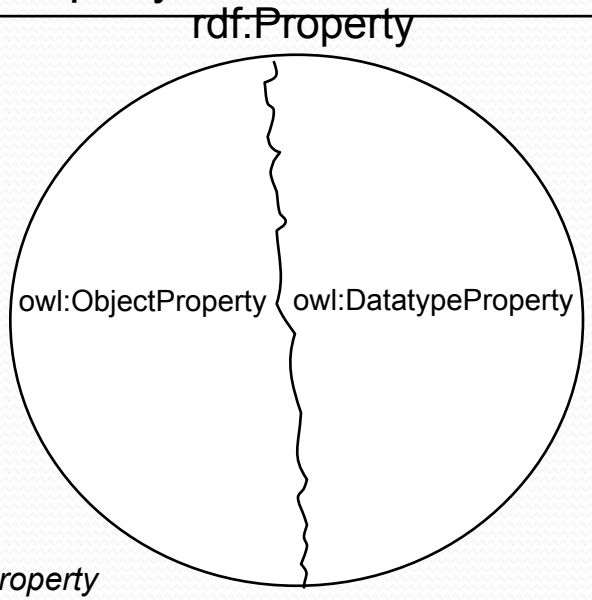
1. SymmetricProperty and TransitiveProperty can only be used to relate Resources to Resources.
2. FunctionalProperty and InverseFunctionalProperty can be used to relate Resources to Resources, or Resources to an RDF Schema Literal or an XML Schema datatype.

Why wasn't owl:inverseOf shown in the preceding class hierarchy?

Answer: owl:inverseOf is a "property", not a class, e.g.

property

```
<owl:ObjectProperty rdf:ID="feedsFrom">
        <owl:inverseOf rdf:resource="#emptiesInto"/>
        <rdfs:domain rdf:resource="#BodyOfWater"/>
        <rdfs:range rdf:resource="#River"/>
 </owl:ObjectProperty>
```

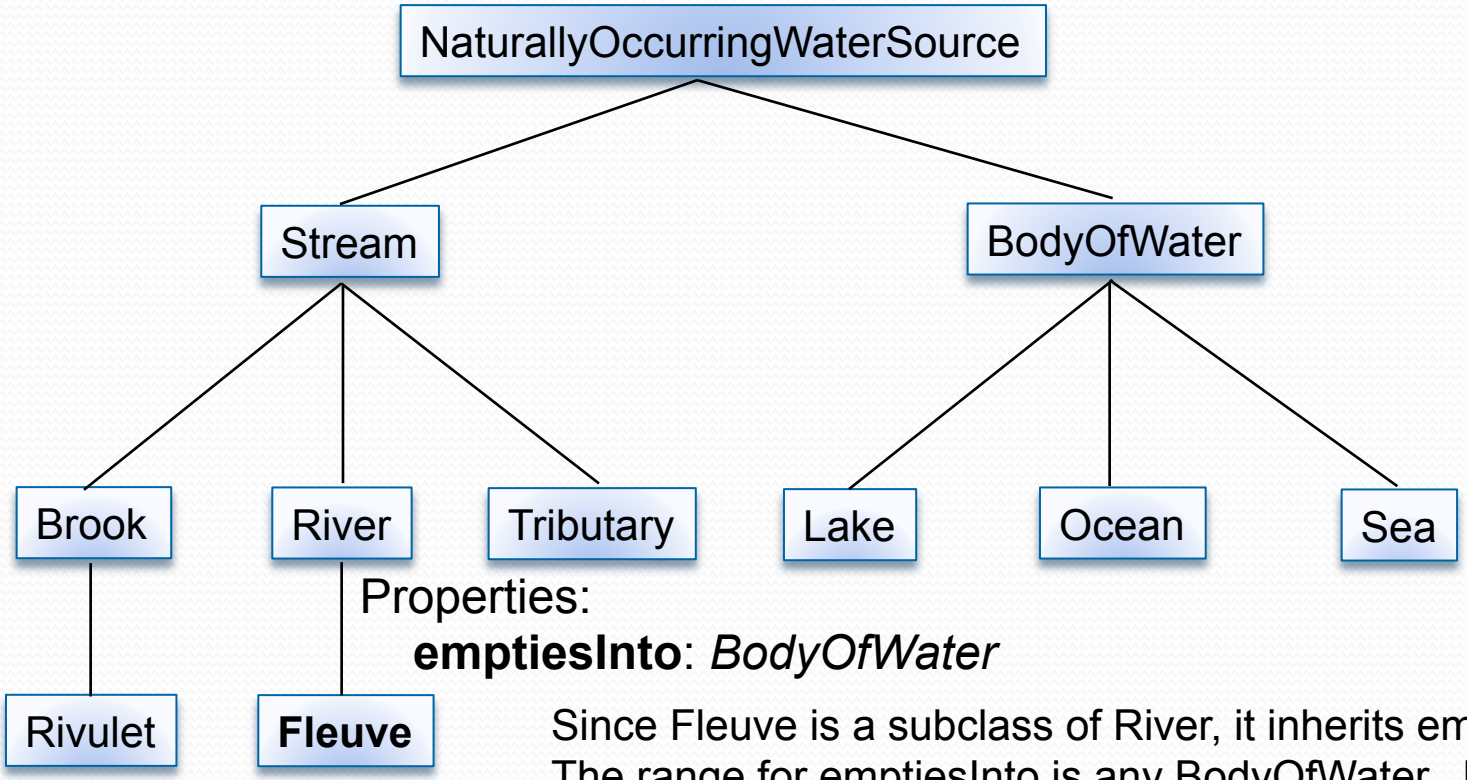


Properties:
inverseOf: *owl:ObjectProperty*

Constraining a property based upon its context

- Now we will look at ways to constrain the range of a property based upon the context (class) in which it is used ...

Sometimes a class needs to restrict the range of a property



Since Fleuve is a subclass of River, it inherits emptiesInto. The range for emptiesInto is any BodyOfWater. However, the definition of a Fleuve (French) is: "a River which emptiesInto a Sea". Thus, *in the context of the Fleuve class* we want the range of emptiesInto restricted to Sea.

Global vs Local Properties

- **rdfs:range** imposes a global restriction on the *emptiesInto* property, i.e., the **rdfs:range** value applies to *River* and all subclasses of *River*.
- As we have seen, in the context of the *Fleuve* class, we would like the *emptiesInto* property to have its range restricted to just the *Sea* class. Thus, for the *Fleuve* class we want a local definition of *emptiesInto*.
- Before we see how to do this, we need to look at how classes are defined in OWL.

Defining Classes in OWL

- OWL classes permit much greater expressiveness than RDF Schema classes.
- Consequently, OWL has created their own Class, **owl:Class**.

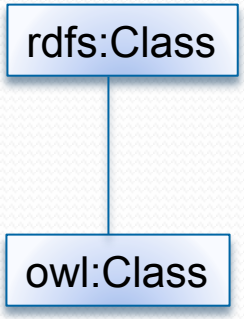
```
<rdfs:Class rdf:ID="River">
 <rdfs:subClassOf rdf:resource="#Stream"/>
</rdfs:Class>
```

RDFS

```
<owl:Class rdf:ID="River">
 <rdfs:subClassOf rdf:resource="#Stream"/>
</owl:Class>
```

OWL

owl:Class is a subclass of rdfs:Class



Defining emptiesInto (when used in *Fleuve*) to have **allValuesFrom** the Sea class

```
<?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#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xml:base="http://www.geodesy.org/water/naturally-occurring">

  <owl:Class rdf:ID="Fleuve">
    <rdfs:subClassOf rdf:resource="#River"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#emptiesInto"/>
        <owl:allValuesFrom rdf:resource="#Sea"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  …
</rdf:RDF>
```

Fleuve is a subclass of an "anonymous class"

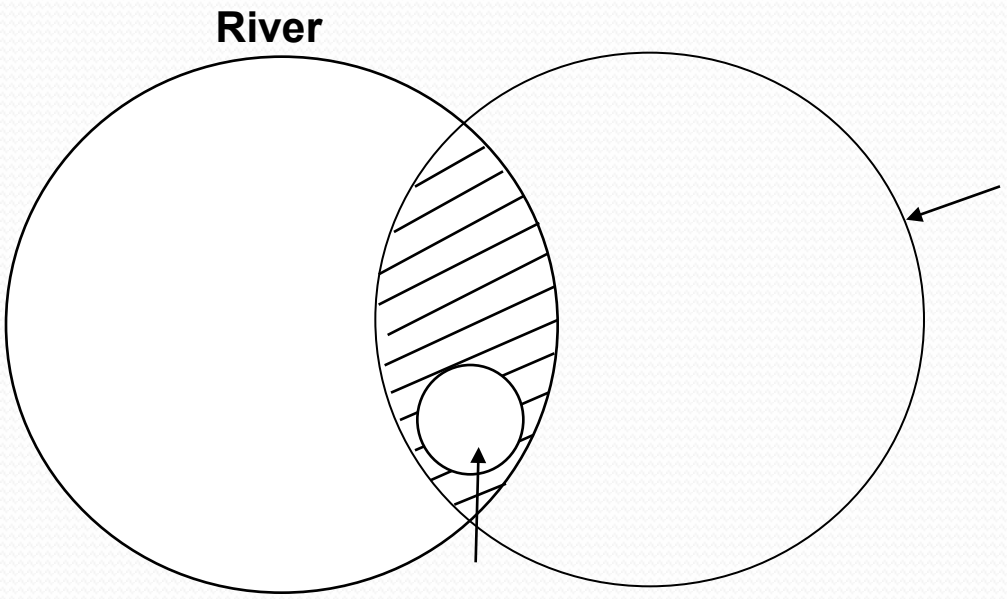
anonymous class

```
<owl:Class rdf:ID="Fleuve">
      <rdfs:subClassOf rdf:resource="#River"/>
       <rdfs:subClassOf>
            <owl:Restriction>
                  <owl:onProperty rdf:resource="#emptiesInto"/>
                  <owl:allValuesFrom rdf:resource="#Sea"/>
            </owl:Restriction>
       </rdfs:subClassOf>
</owl:Class>
```

This is read as: "The Fleuve class is a **subClassOf** River, and a **subClassOf** an *anonymous class* which has a property *emptiesInto* and all values for *emptiesInto* must be instances of *Sea*."

Here's an easier way to read this: "The *Fleuve* class is a **subClassOf** *River*. It has a property *emptiesInto*. All values for *emptiesInto* must be instances of Sea."

Definition of Fleuve



The members of this **anonymous class** are instances which have an emptiesInto property in which all values are instances of Sea.

Fleuve- a River that emptiesInto a Sea.

An instance of Fleuve

```
<?xml version="1.0"?>
<Fleuve rdf:ID="Yangtze"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns="http://www.geodesy.org/water/naturally-occurring#">
   <emptiesInto rdf:resource="http://www.china.org/geography#EastChinaSea"/>
</Fleuve>
```

Yangtze.rdf

We can infer that this value must be a Sea!

All values for *emptiesInto* must be an instance of *Sea*, in the context of the *Fleuve* class.

Two forms of rdfs:subClassOf

1

```
<rdfs:subClassOf rdf:resource="#River"/>
```

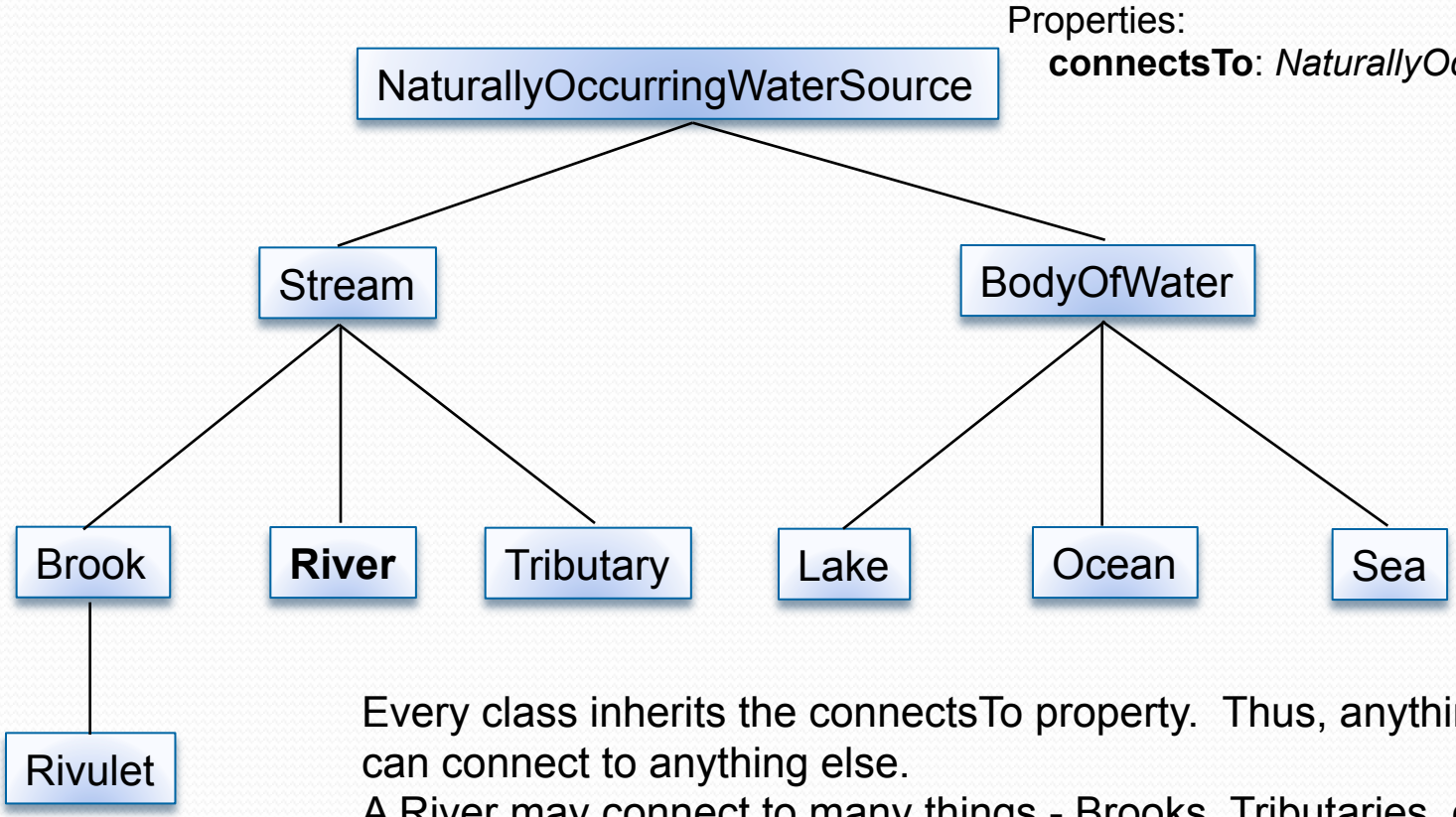
Specify the class using the rdf:resource attribute.

Specify the class using owl:Restriction.

2

```
<rdfs:subClassOf>
        <owl:Restriction>
                <owl:onProperty rdf:resource="#emptiesInto"/>
                <owl:allValuesFrom rdf:resource="#Sea"/>
        </owl:Restriction>
</rdfs:subClassOf>
```

To be a River at least one value of connectsTo must be BodyOfWater



Every class inherits the connectsTo property. Thus, anything can connect to anything else.
A River may connect to many things - Brooks, Tributaries, etc. However, one thing that it *must* connect to is a BodyOfWater (Lake, Ocean, or Sea). Thus, *in the context of* the River class the connectsTo property should have at least one value that is a BodyOfWater.

Defining connectsTo (when used in River) to have someValuesFrom the BodyOfWater class

```
<?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#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xml:base="http://www.geodesy.org/water/naturally-occurring">

  <owl:Class rdf:ID="River">
    <rdfs:subClassOf rdf:resource="#Stream"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#connectsTo"/>
        <owl:someValuesFrom rdf:resource="#BodyOfWater"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  ...

</rdf:RDF>
```

naturally-occurring.owl (snippet)

Understanding owl:someValuesFrom

```
<owl:Class rdf:ID="River">
      <rdfs:subClassOf rdf:resource="#Stream"/>
       <rdfs:subClassOf>
            <owl:Restriction>
                  <owl:onProperty rdf:resource="#connectsTo"/>
                  <owl:someValuesFrom rdf:resource="#BodyOfWater"/>
            </owl:Restriction>
       </rdfs:subClassOf>
</owl:Class>
```

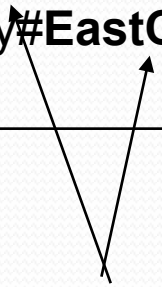
This is read as: "The *River* class is a **subClassOf** *Stream*, and a *subClassOf* an anonymous class which has a property *connectsTo* and some values (at least one) of *connectsTo* must be instances of *BodyOfWater*."

Here's an easier way to read this: "The *River* class is a *subClassOf Stream*. It has a property *connectsTo*. At least one value for *connectsTo* must be an instance of *BodyOfWater*."

An instance of River

```
<?xml version="1.0"?>
<River rdf:ID="Yangtze"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns="http://www.geodesy.org/water/naturally-occurring#">
   <connectsTo rdf:resource="http://www.china.org/rivers#Wu"/>
   <connectsTo rdf:resource="http://www.china.org/geography#EastChinaSea"/>
</River>
```

Yangtze.rdf



At least one of these values must be a *BodyOfWater (Lake, Ocean, or Sea)*! (Assume that there are no other documents which describe the Yangtze.)

At least one value for *connectsTo* must be an instance of *BodyOfWater*, *in the context of the River class*.

allValuesFrom vs. someValuesFrom

```
<owl:onProperty rdf:resource="#emptiesInto"/>
<owl:allValuesFrom rdf:resource="#Sea"/>
```

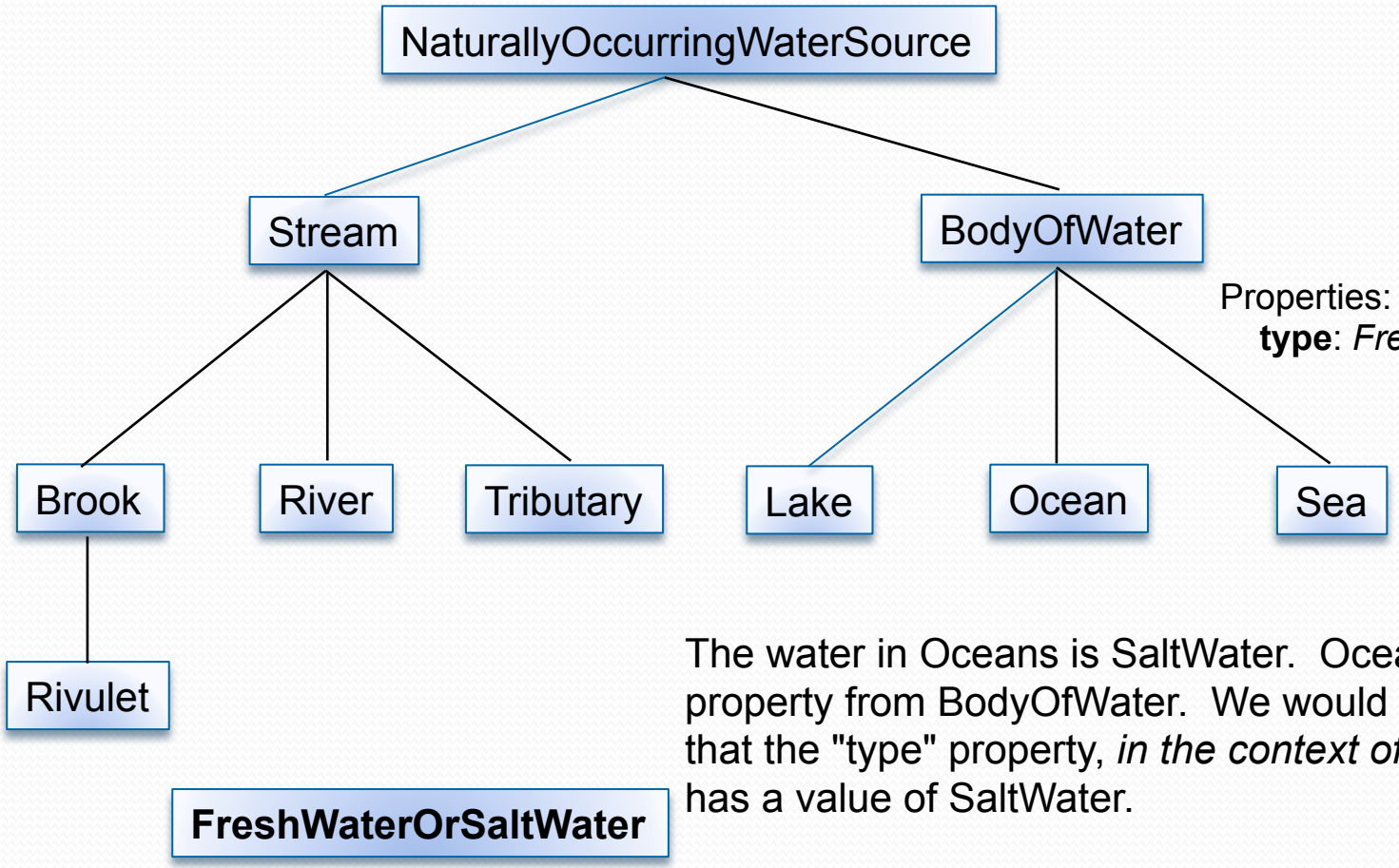
Wherever there is an *emptiesInto* property, all its values must be instances of *Sea*. [There may be zero *emptiesInto* properties.]

versus:

```
<owl:onProperty rdf:resource="#connectsTo"/>
<owl:someValuesFrom rdf:resource="#BodyOfWater"/>
```

There must be at least one *connectsTo* property whose value is *BodyOfWater*. [There must be at least one *connectsTo* property.]

All Oceans are SaltWater



The water in Oceans is SaltWater. Ocean inherits the "type" property from BodyOfWater. We would like to indicate that the "type" property, *in the context of an Ocean*, always has a value of SaltWater.

Defining the "type" property to have the value SaltWater (when used in Ocean)

```
<?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#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xml:base="http://www.geodesy.org/water/naturally-occurring">

   <FreshWaterOrSaltWater rdf:ID="SaltWater"/>

    <owl:Class rdf:ID="Ocean">
       <rdfs:subClassOf rdf:resource="#BodyOfWater"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#type"/>
                <owl:hasValue rdf:resource="#SaltWater"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>

   ...

</rdf:RDF>
```

naturally-occurring.owl (snippet)

Understanding owl:hasValue

Note that this is an **instance** of the class FreshWaterOrSaltWater.

```
<FreshWaterOrSaltWater rdf:ID="SaltWater"/>

<owl:Class rdf:ID="Ocean">
      <rdfs:subClassOf rdf:resource="#BodyOfWater"/>
       <rdfs:subClassOf>
            <owl:Restriction>
                  <owl:onProperty rdf:resource="#type"/>
                  <owl:hasValue rdf:resource="#SaltWater"/>
            </owl:Restriction>
       </rdfs:subClassOf>
</owl:Class>
```

This is read as: "The *Ocean* class is a subClassOf *BodyOfWater*, and a subClassOf an anonymous class which has a property - type - that has the value *SaltWater*."

Here's an easier way to read this: "The *Ocean* class is a subClassOf *BodyOfWater*. Every Ocean has a 'type' property whose value is *SaltWater*."

An instance of Ocean

```
<?xml version="1.0"?>
<Ocean rdf:ID="PacificOcean"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns="http://www.geodesy.org/water/naturally-occurring#">
    <type rdf:resource="http://www.geodesy.org/water/naturally-occurring#SaltWater"/>
</Ocean>
```

PacificOcean.rdf

Every instance of *Ocean* must have a property "type" whose value is *SaltWater*.
Note: it is not necessary to put the type property in an *Ocean* instance document - the "type" may be inferred from *hasValue*. That is, *the Ontology indicates that if it's an Ocean then its type is SaltWater.*

At least one "type" property must have the value *SaltWater*, *in the context of an Ocean class*.

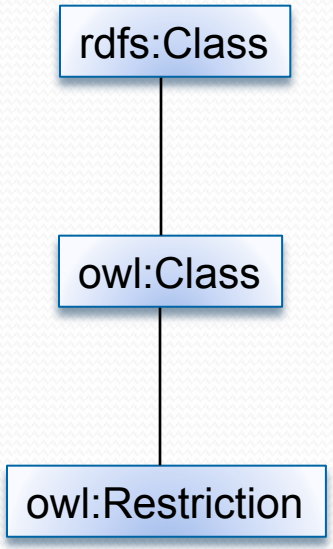
owl:hasValue means **there exists** a property with the specified value

- The owl:hasValue property restriction simply asserts that **there exists** a property with the value.
- In fact, there may be other instances of the same property that do not have the value.
- For the Ocean example, we know that every Ocean is of type of SaltWater.

Summary of the different ways a class can constrain a property

- In the preceding slides we have seen the different ways that a class can constrain a global property. We saw that a property can be constrained such that:
  - All values must belong to a certain class (use **allValuesFrom**).
  - At least one value must come from a certain class (use **someValuesFrom**).
  - It has a specific value (use **hasValue**).

Properties of the Restriction Class



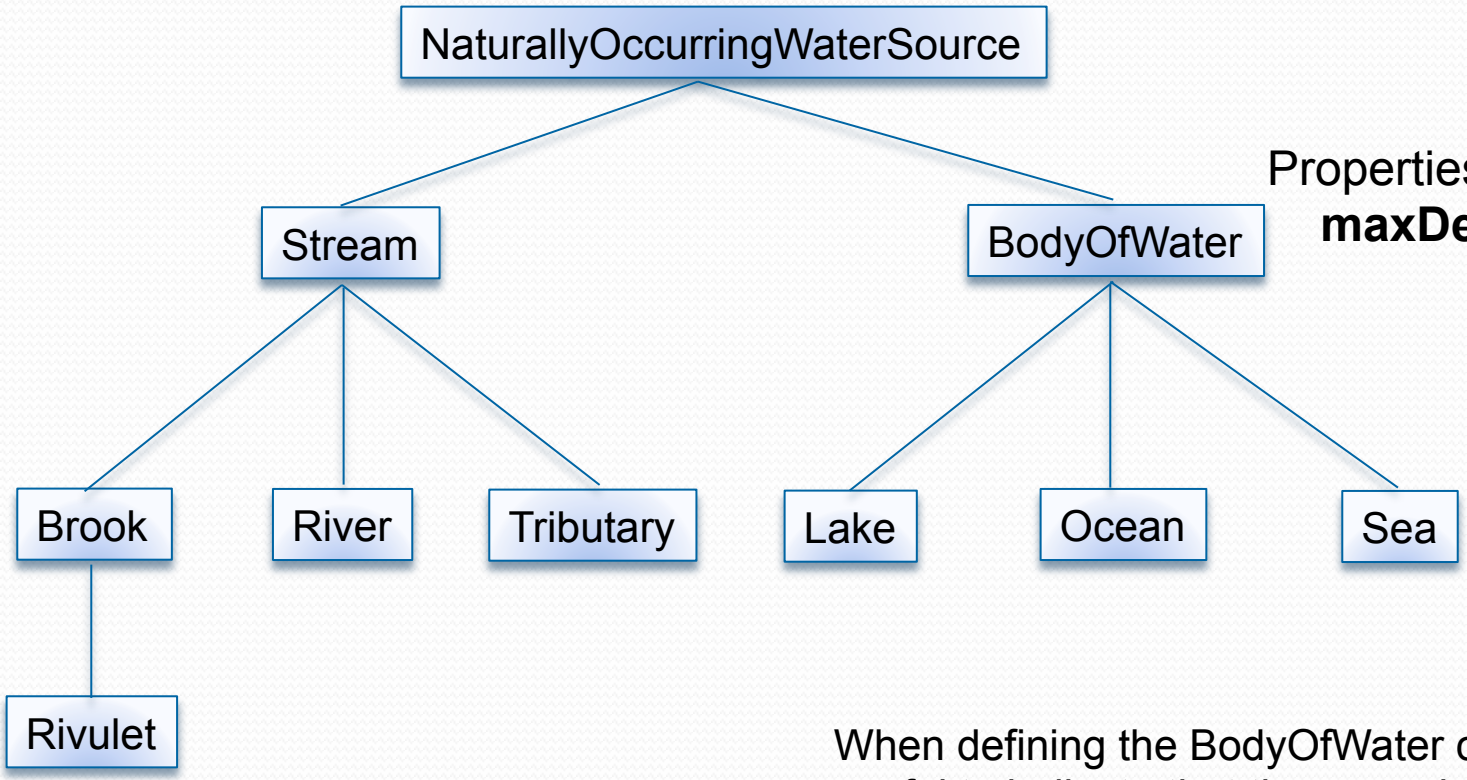
Properties:

onProperty: *rdf:Property*
allValuesFrom: *rdfs:Class*
hasValue:
someValuesFrom: *rdfs:Class*

Context-specific cardinality constraints

- Definition of cardinality: the number of occurrences.
- Now we will look at ways to constrain the cardinality of a property based upon the context (class) in which it is used ...

A BodyOfWater can have only one maxDepth (cardinality = 1)



Properties:
maxDepth: *xsd:integer*

When defining the BodyOfWater class it would be useful to indicate that there can be only one maxDepth for a BodyOfWater.

Defining the cardinality of the maxDepth property to be 1

```
<?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#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xml:base="http://www.geodesy.org/water/naturally-occurring">

   <owl:Class rdf:ID="BodyOfWater">
      <rdfs:subClassOf rdf:resource="#NaturallyOccurringWaterSource"/>
       <rdfs:subClassOf>
          <owl:Restriction>
             <owl:onProperty rdf:resource="#maxDepth"/>
             <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1
             </owl:cardinality>
          </owl:Restriction>
       </rdfs:subClassOf>
   </owl:Class>

   ...

</rdf:RDF>
```

naturally-occurring.owl (snippet)

Understanding owl:cardinality

```
<owl:Class rdf:ID="BodyOfWater">
      <rdfs:subClassOf rdf:resource="#NaturallyOccurringWaterSource"/>
       <rdfs:subClassOf>
            <owl:Restriction>
                  <owl:onProperty rdf:resource="#maxDepth"/>
                  <owl:cardinality rdf:datatype="http://www.w3.org/2001/
XMLSchema#nonNegativeInteger">1</owl:cardinality>
            </owl:Restriction>
       </rdfs:subClassOf>
</owl:Class>
```

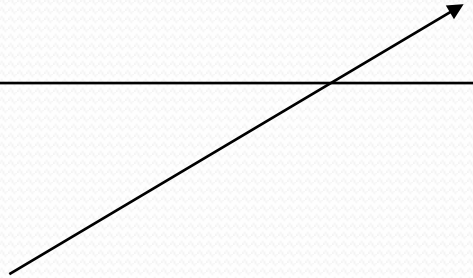
This is read as: "The *BodyOfWater* class is a subClassOf *NaturallyOccurringWaterSource*, and a subClassOf an anonymous class which has a property *maxDepth*. There can be only one *maxDepth* for a *BodyOfWater*. This is indicated by a cardinality of 1."

Here's an easier way to read this: "The *BodyOfWater* class is a subClassOf *NaturallyOccurringWaterSource*. It has a property *maxDepth*. There can be only one *maxDepth* for a *BodyOfWater*."

maxDepth of the PacificOcean

```
<?xml version="1.0"?>
<Ocean rdf:ID="PacificOcean"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns="http://www.geodesy.org/water/naturally-occurring#">
   <maxDepth rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">2300</
maxDepth>
</Ocean>
```

PacificOcean.rdf



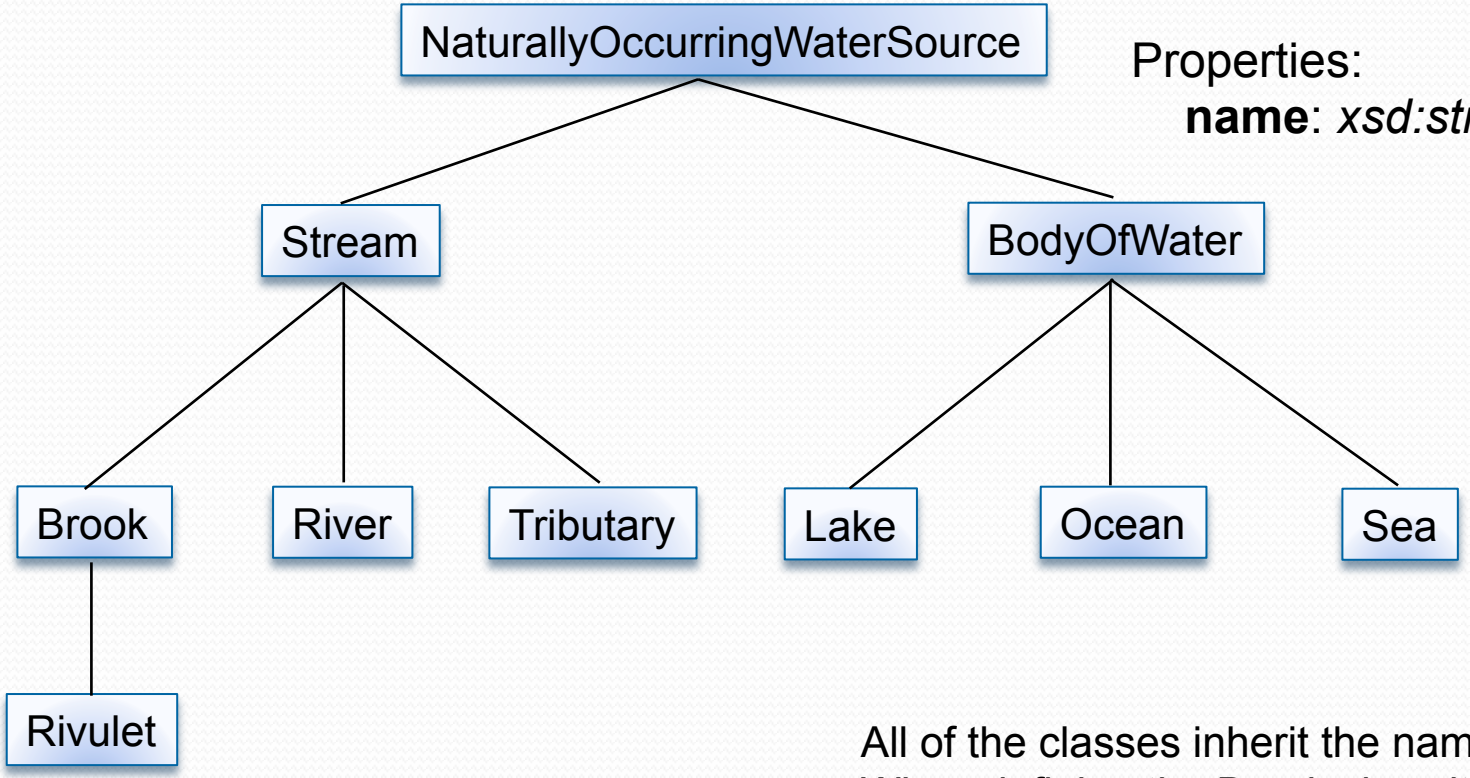
The *PacificOcean* has only one *maxDepth*.

There is only one maxDepth, *in the context of a BodyOfWater (e.g., Ocean) class*.

The cardinality is not mandating the number of occurrences of a property in an instance document!

- Differentiate between these two statements:
  - 1. In an instance document there can be only one *maxDepth* property for a *BodyOfWater*.
  - 2. A *BodyOfWater* has only one *maxDepth*.
- Do you see the difference?
  - 1. The first statement is something that you would find in an XML Schema.
  - 2. The second statement is a statement of information. It places no restrictions on the number of occurrences of the maxDepth property in an instance document. In fact, any resource may have multiple maxDepth properties. They must all be equal, however, since there can be only one maxDepth per resource.

Some Brooks have no name (minCardinality = 0)



All of the classes inherit the name property. When defining the Brook class it would be useful to indicate that a Brook might not have a name.

Defining the minCardinality of the name property to be 0

```
<?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#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xml:base="http://www.geodesy.org/water/naturally-occurring">

  <owl:Class rdf:ID="Brook">
    <rdfs:subClassOf rdf:resource="#Stream"/>
     <rdfs:subClassOf>
        <owl:Restriction>
           <owl:onProperty rdf:resource="#name"/>
           <owl:minCardinality rdf:datatype="http://www.w3.org/2001/
XMLSchema#nonNegativeInteger">0</owl:minCardinality>
        </owl:Restriction>
     </rdfs:subClassOf>
  </owl:Class>
  ...
</rdf:RDF>
```

naturally-occurring.owl (snippet)

Defining the cardinality of the name property to be a range (0-10)

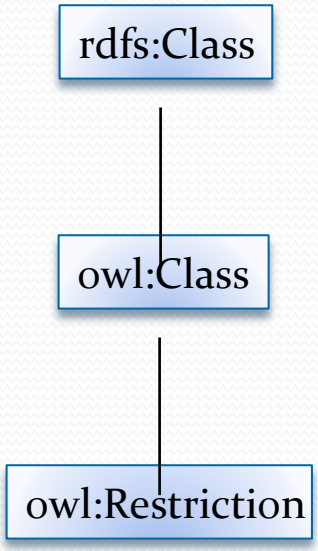
```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xml:base="http://www.geodesy.org/water/naturally-occurring">

   <owl:Class rdf:ID="Brook">
      <rdfs:subClassOf rdf:resource="#Stream"/>
       <rdfs:subClassOf>
          <owl:Restriction>
             <owl:onProperty rdf:resource="#name"/>
             <owl:minCardinality rdf:datatype="http://www.w3.org/2001/
XMLSchema#nonNegativeInteger">0</owl:minCardinality>
             <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/
XMLSchema#nonNegativeInteger">10</owl:maxCardinality>
          </owl:Restriction>
       </rdfs:subClassOf>
   </owl:Class>
   …
</rdf:RDF>
```

naturally-occurring.owl (snippet)

Summary of the different ways to express the cardinality of a property

- In the preceding slides we have seen the ways that a class can specify the cardinality of a property, using:
- cardinality
- minCardinality
- maxCardinality

Complete List of Properties of the Restriction Class



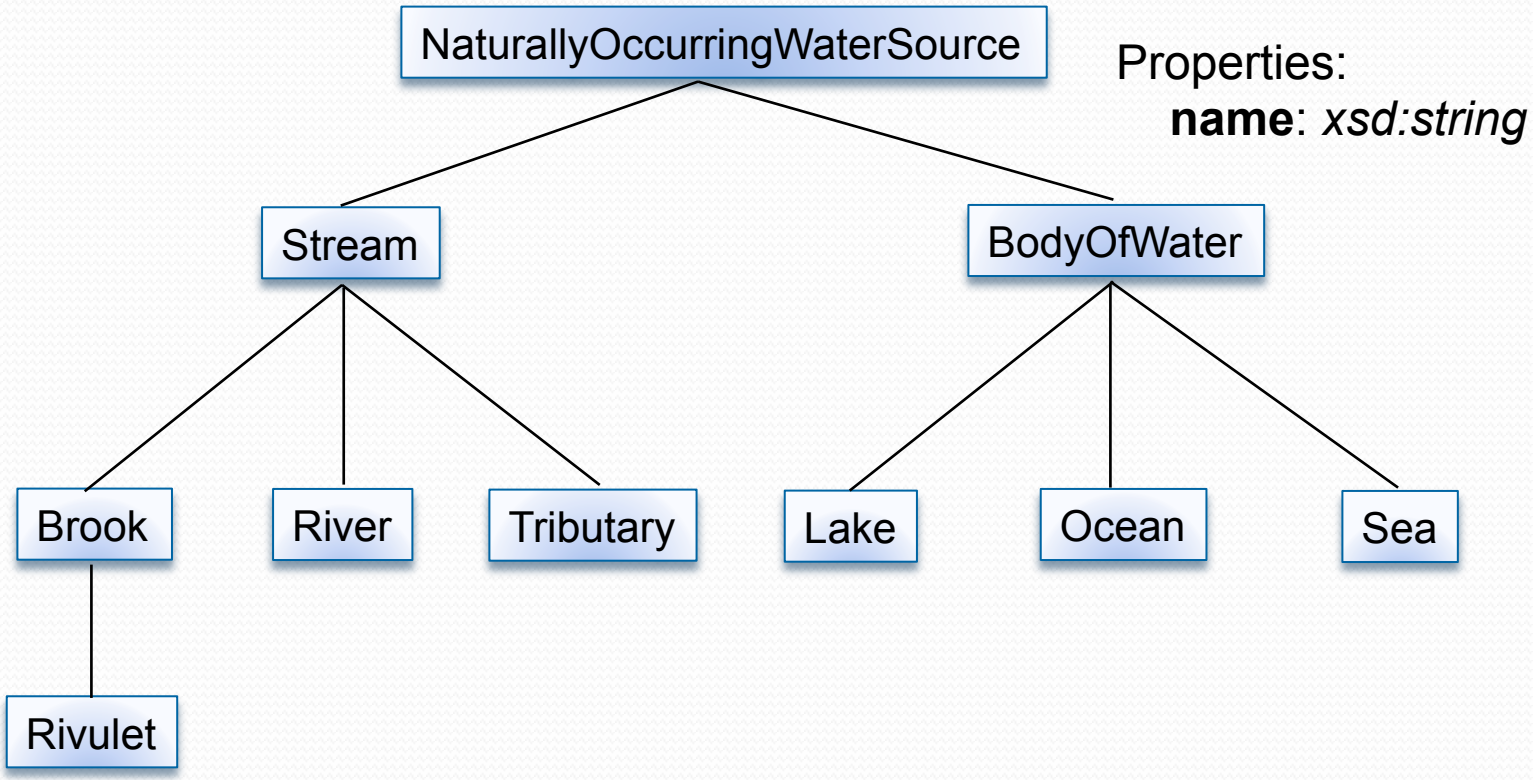
Properties:

onProperty: *rdf:Property*
allValuesFrom: *rdfs:Class*
hasValue:
someValuesFrom: *rdfs:Class*
cardinality: *xsd:nonNegativeInteger*
minCardinality: *xsd:nonNegativeInteger*
maxCardinality: *xsd:nonNegativeInteger*

Equivalent Properties

- Now we will look at the ways to express that two properties are equivalent ...

name is equivalent to the Title property in Dublin Core



Defining name to be equivalent to dc:Title

```
<?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#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xml:base="http://www.geodesy.org/water/naturally-occurring">

   <owl:DatatypeProperty rdf:ID="name">
      <owl:equivalentProperty rdf:resource="http://pur1.org/metadata/dublin-core#Title"/
>
      <rdfs:domain rdf:resource="#NaturallyOccurringWaterSource"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
   </owl:DatatypeProperty>

   ...

</rdf:RDF>
```

naturally-occurring.owl (snippet)

Note that we are using owl:DatatypeProperty to define name.