# OWL Web Ontology Language

Roger L. Costello

David B. Jacobs

The MITRE Corporation

(The creation of this tutorial was sponsored by DARPA)

# Acknowledgments

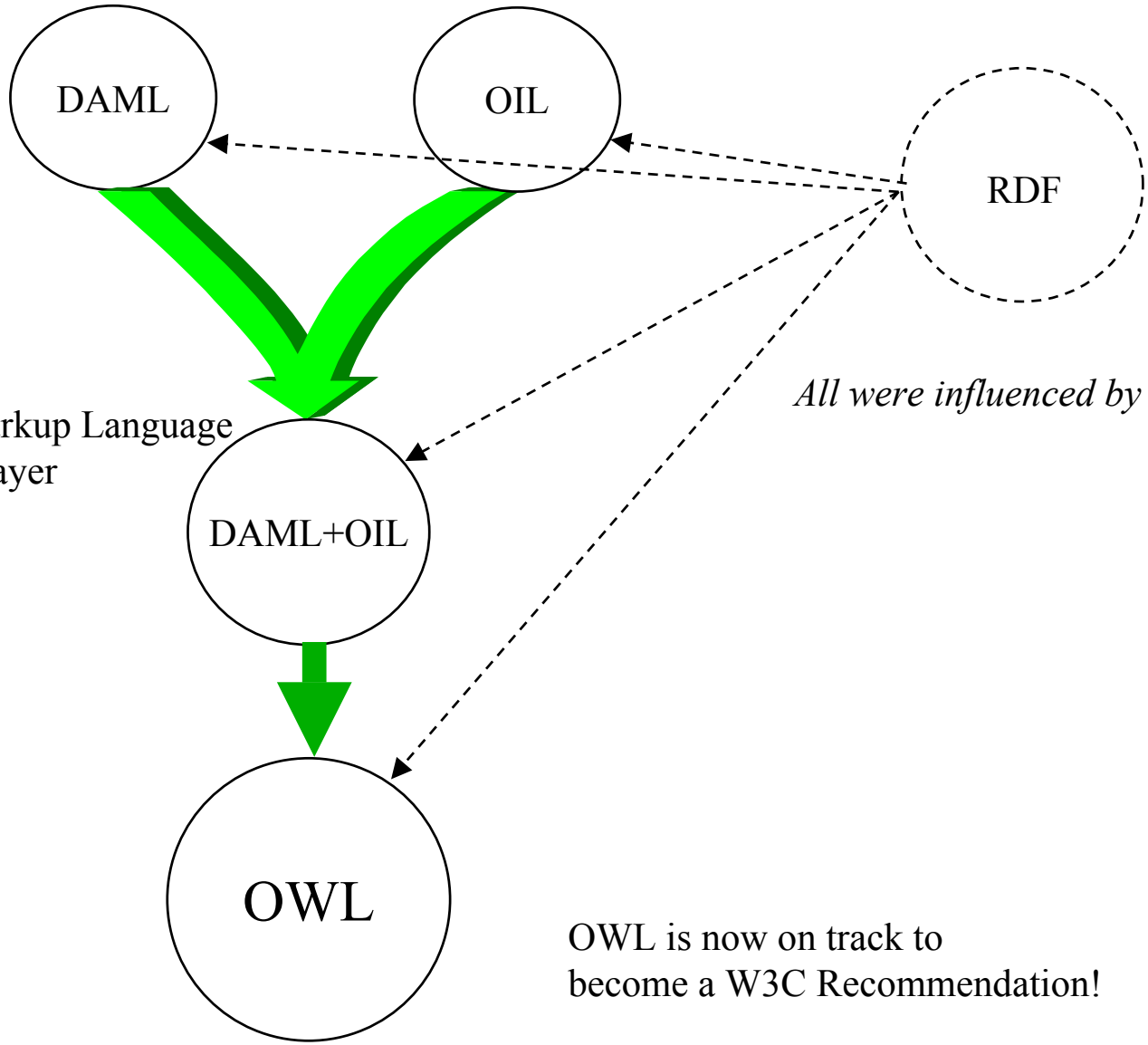

- We are very grateful to the Defense Advanced Research Projects Agency (DARPA) for funding the creation of this tutorial. We are especially grateful to Murray Burke (DARPA) and John Flynn (BBN) for making it all happen.
- We would like to thank Stephen Dyer for creating the labs.
- Special thanks to the following people for their help in answering our unending questions:

Peter F. Patel-Schneider
Mike Dean
Ian Davis
Jim Hendler
Nikita Ogievetsky
Dan Connolly
Jeff Z. Pan
Peter Crowther
Frank Manola
Deborah McGuinness
Leo Obrst
Rafael Batres
Steven Gollery
Enrico Franconi
Mike Pool
David Allsopp
Yuzhong Qu
Jonathan Borden
Mary Pulvermacher

# Prerequisites

- OWL builds on top of (i.e., extends) RDF Schema.
- This tutorial assumes that you already have a solid understanding of RDF and RDF Schema.
  - As well as this tutorial on OWL, we have also created a tutorial on RDF, and a tutorial on RDF Schema. Please see here:
    - http://www.xfront.com/rdf/
    - http://www.xfront.com/rdf-schema/

# Origins of OWL




DAML = DARPA Agent Markup Language
OIL = Ontology Inference Layer

*All were influenced by RDF*

OWL is now on track to
become a W3C Recommendation!

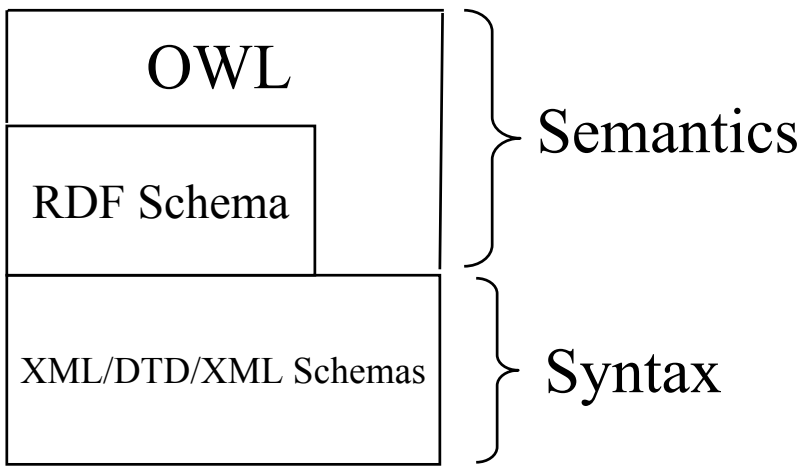
# W3C Status of OWL

- OWL is currently (April, '03) at the W3C Candidate Recommendation stage.
- OWL is targeted to go to Pre-Recommendation status in May, '03.
- OWL should be be at the W3C Recommendation status by summer '03.
- For complete OWL schedule details see:
  - http://www.w3.org/2001/sw/WebOnt/#L151

# Purpose of OWL

- The **purpose** of OWL is identical to RDF Schemas - to provide an XML vocabulary to define classes, properties and their relationships.
  - 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.
- The **benefit** of OWL is that it facilitates a much greater degree of inferencing than you get with RDF Schemas.

# OWL and RDF Schema enables *machine-processable semantics*



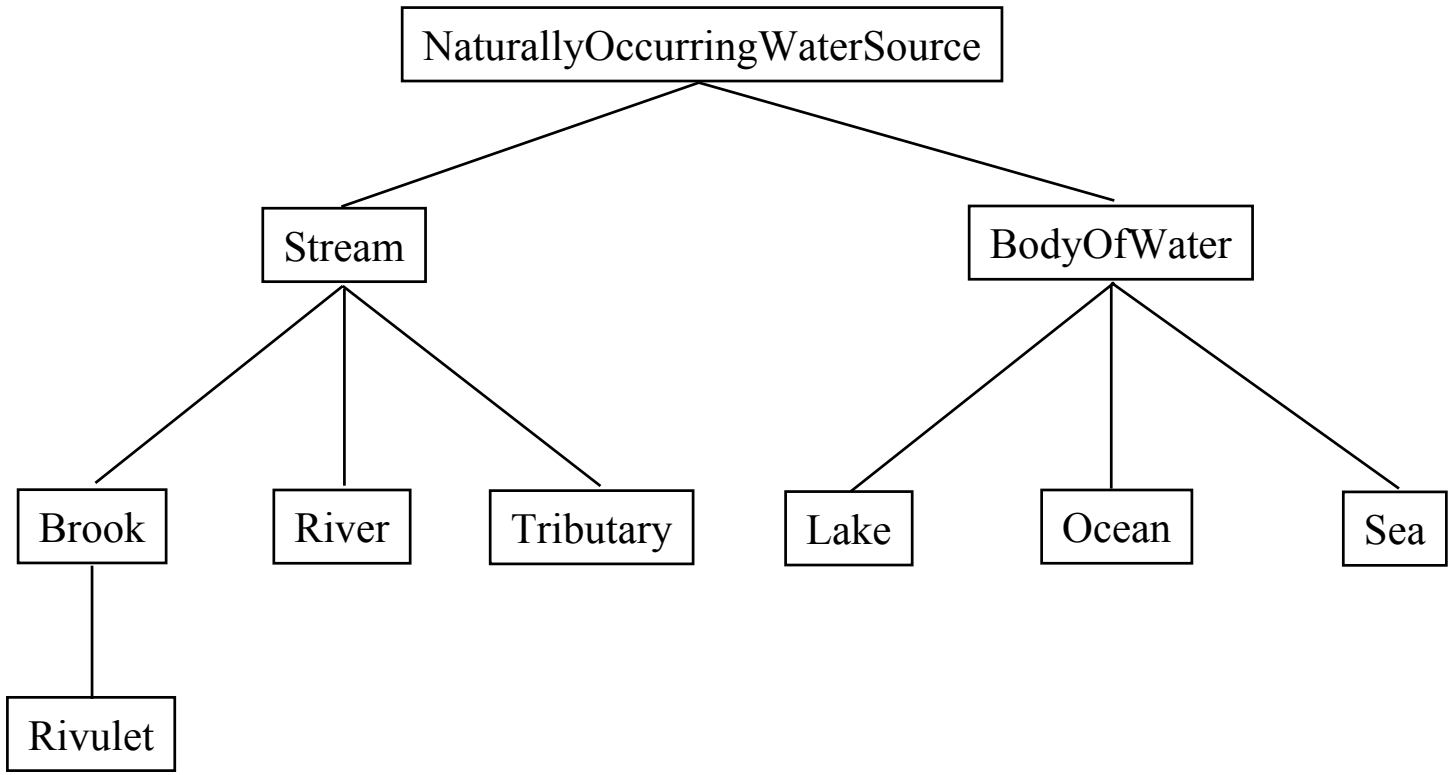
# Organization of this Tutorial

- OWL gives you a syntax to express statements about properties and classes, above and beyond what you can make with RDF Schema.
- In this tutorial we present:
  - Using OWL to define properties.
  - Using OWL to define classes.
  - OWL statements that you can incorporate into your instance documents.

# OWL = RDF Schema + more

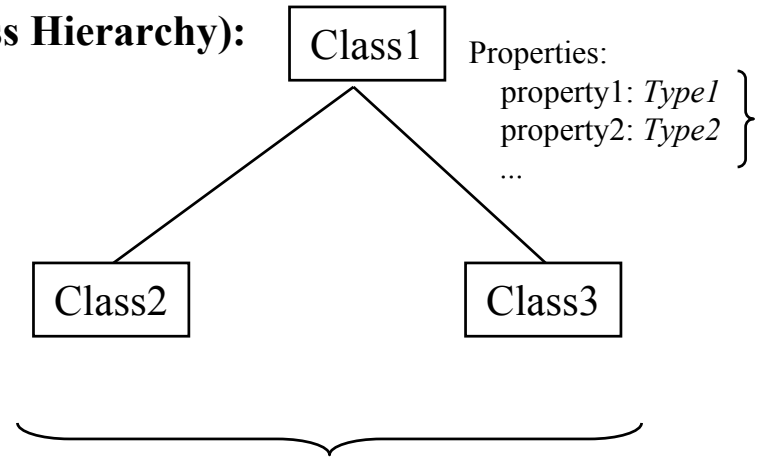
- Note: all of the elements/attributes provided by RDF and RDF Schema can be used when creating an OWL document.

# This tutorial will use the below "water taxonomy" to explain OWL



# Notations used in this Tutorial

**Taxonomy (Class Hierarchy):**



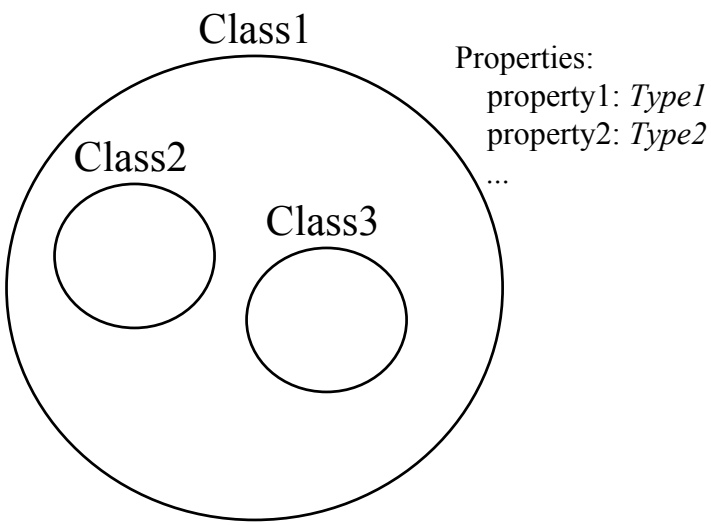

These are the properties of Class1. The name of the property is shown (e.g., property1), and its range is shown in italics (e.g., *Type1*).

Class2 and Class3 are subclasses of Class1.

---

**Venn Diagram:**

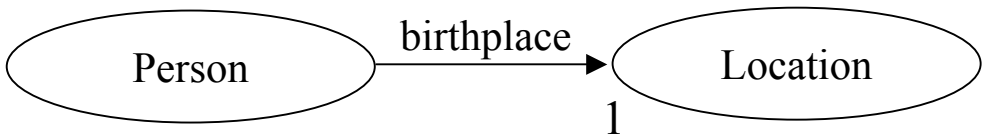
An alternate notation to the above class hierarchy is to use a Venn diagram, as shown here.



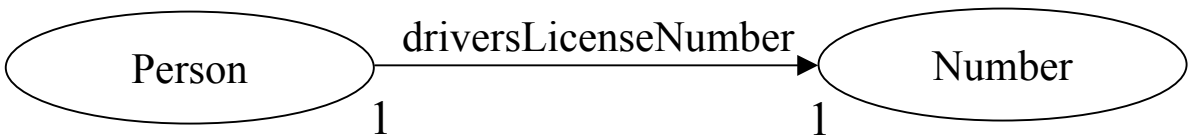

Continued →

# Notations used in this Tutorial

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




This notation is used to indicate that a person has only one driver's license number. Further, a driver's license number is associated with only one person:



# OWL Tools

- RDF Instance Creator (RIC)
  - http://www.mindswap.org/~mhgrove/RIC/RIC.shtml
  - Limited OWL capabilities
- OilEd:
  - http://oiled.man.ac.uk/
  - Editor for ontologies
  - Mostly for DAML+OIL, exports OWL but not a current representation
- OWL Validator:
  - http://owl.bbn.com/validator/
  - Web-based or command-line utility
  - Performs basic validation of OWL file
- Dumpont:
  - http://www.daml.org/2001/03/dumpont/
  - a simple class and hierarchy property viewer, which also works with OWL, e.g.,
    - http://www.daml.org/cgi-bin/dumpont?http://www.w3.org/2002/07/owl
- OWL Ontology Validator:
  - http://phoebus.cs.man.ac.uk:9999/OWL/Validator
  - a "species validator" that checks use of OWL Lite, OWL DL, and OWL Full constructs
- Euler:
  - http://www.agfa.com/w3c/euler/
  - an inference engine which has been used for a lot of the OWL Test Cases
- Chimaera:
  - http://www.ksl.stanford.edu/software/chimaera/
  - Ontology evolution environment (diagnostics, merging, light editing)
  - Mostly for DAML+OIL, being updated to export and inport current OWL
- DAML Tools Page - http://www.daml.org/tools/

# Two Minute Preview of OWL

- Before getting into the details of OWL let's spend a couple of minutes examining three examples which demonstrate some of the capabilities of OWL.

# Example 1: The Robber and the Speeder

DNA samples from a robbery identified John Walker Lindh as the suspect. Here is the police report on the robbery:

```
<Robbery rdf:ID="report-2003-03-17-XTf4">
   <description>...</description>
   <suspect>
      <Person rdf:about="http://www.person.org#John_Walker_Lindh"/>
   </suspect>
</Robbery>
```

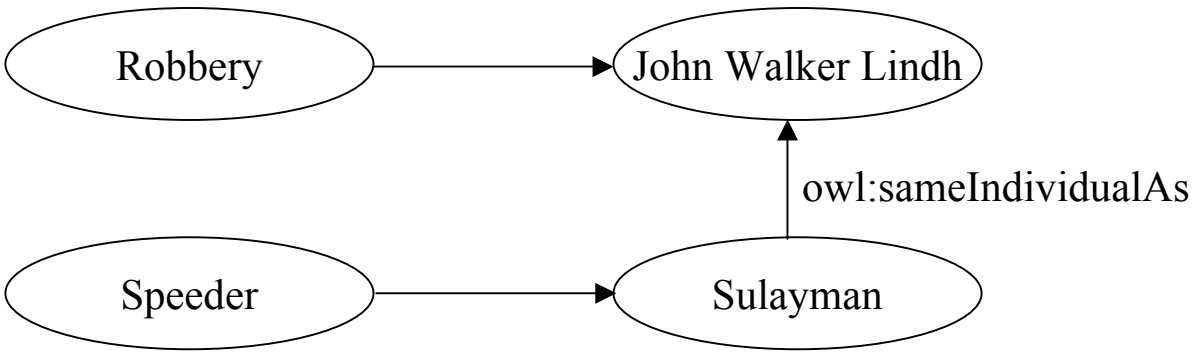
Later in the day a state trooper gives a person a ticket for speeding. The driver's license showed the name Sulayman. Here is the state trooper's report on the speeder:

```
<Speeder rdf:ID="report-2003-03-17-QWRP">
   <description>...</description>
   <driver>
      <Person rdf:about="http://www.person.org#Sulayman"/>
   </driver>
</Speeder>
```

# Any Relationship between the Robber and the Speeder?

The Central Intelligence Agency (CIA) has a file on Sulayman:

```
<Person rdf:about="http://www.person.org#Sulayman">
   <owl:sameIndividualAs rdf:resource="http://www.person.org#John_Walker_Lindh"/>
</Person>
```



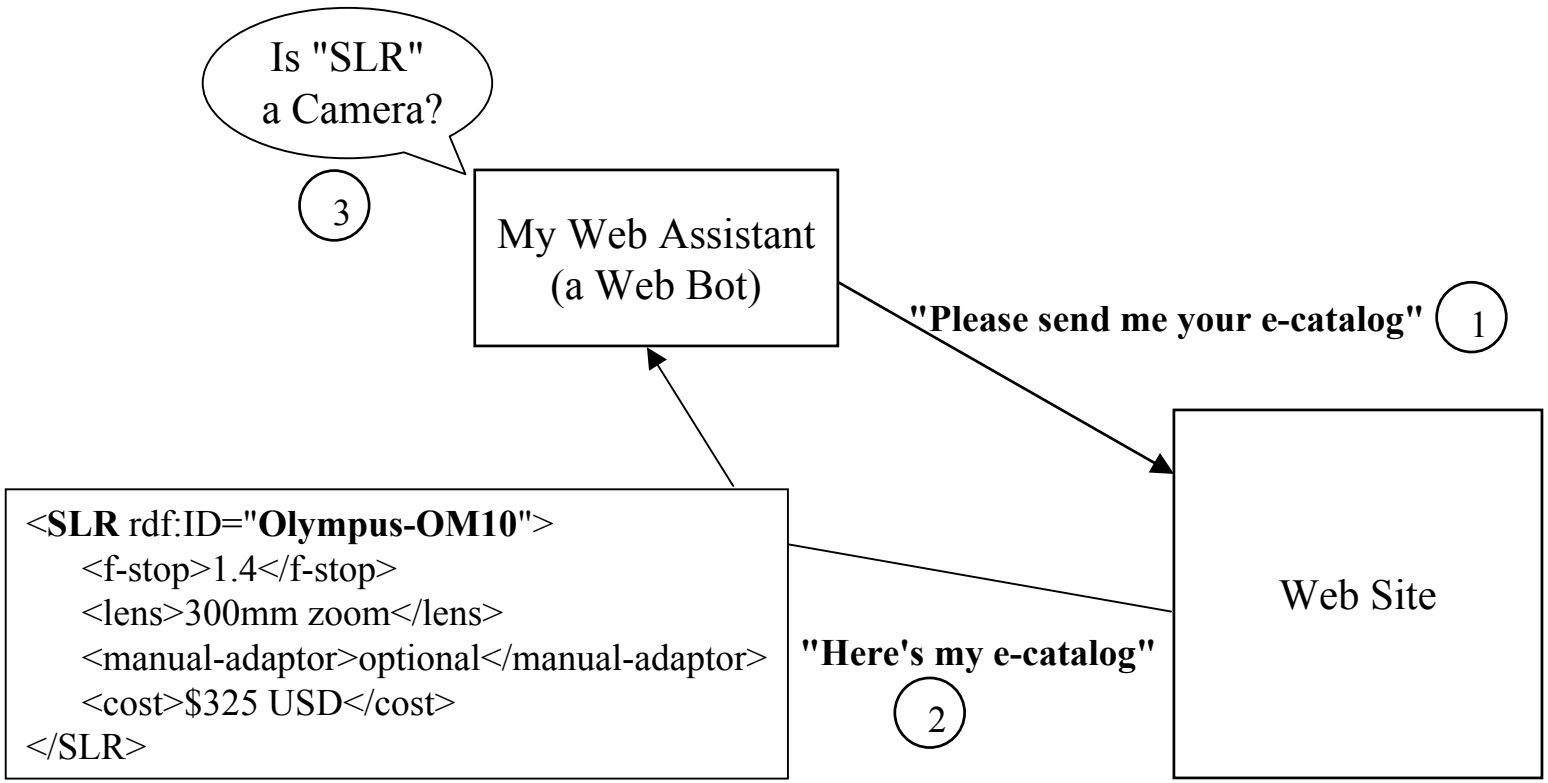

The local police, state troopers, and CIA share their information, thus enabling the following inference to be made:

**Inference**: The Robber and the Speeder are one and the same!

# Lesson Learned

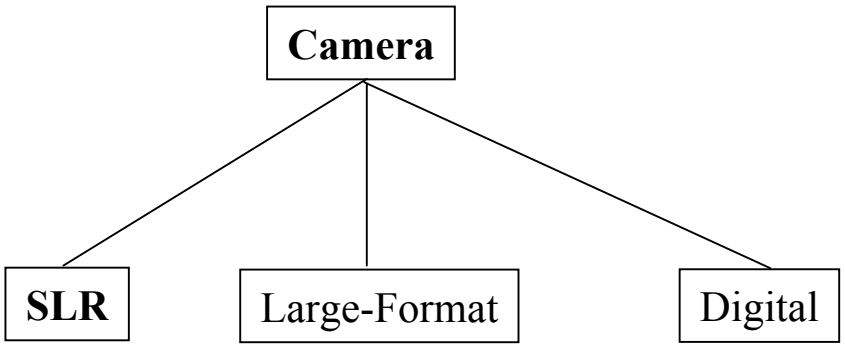
- OWL provides a property (owl:sameIndividualAs) for indicating that two resources (e.g., two people) are the same.

# 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 Olympus-OM10 SLR is a Camera!

# Lesson Learned

- OWL provides elements to construct taxonomies (called class hierarchies). The taxonomies can be used to dynamically discover relationships!

# Example 3: The Birthplace of King Kamahameha is …

Upon scanning the Web, three documents were found which contain information about King Kamahameha:

1
```
<Person rdf:about="http://www.person.org#King_Kamahameha">
   <birthplace rdf:about="http://www.states.org#Hawaii"/>
</Person>
```

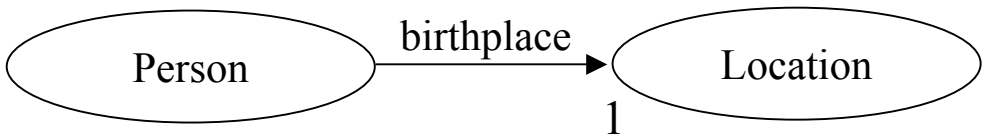
2
```
<Person rdf:about="http://www.person.org#King_Kamahameha">
   <birthplace rdf:resource="http://www.history.org#Sandwich_Islands"/>
</Person>
```

3
```
<Person rdf:about="http://www.person.org#King_Kamahameha">
   <birthplace rdf:resource="http://www.tourism.org#Aloha_State"/>
</Person>
```

**Question**: What is the birthplace of King Kamahameha?

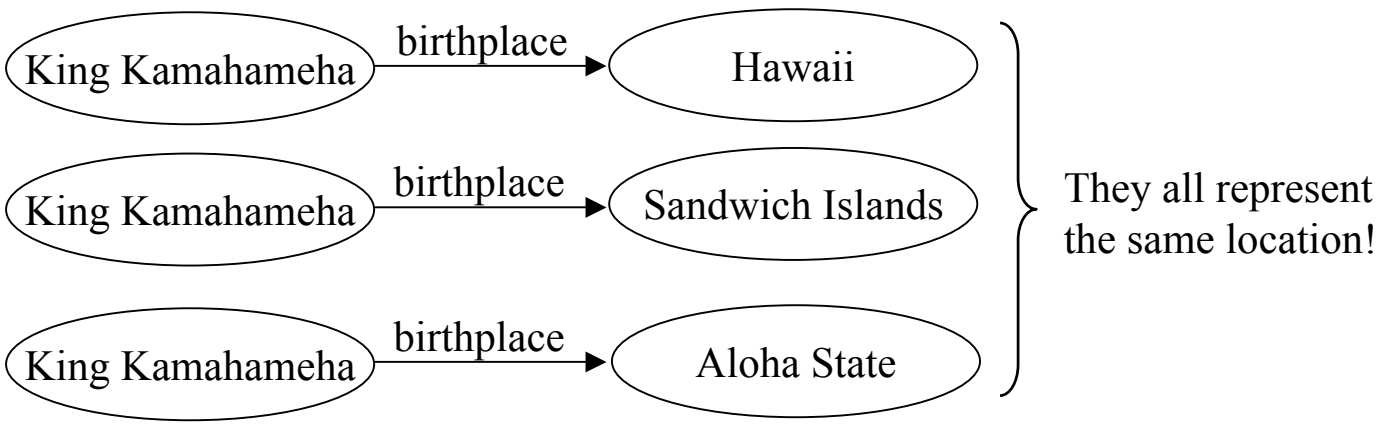
# Answer: all three!

The Person OWL Ontology indicates that a Person has only one birthplace location:



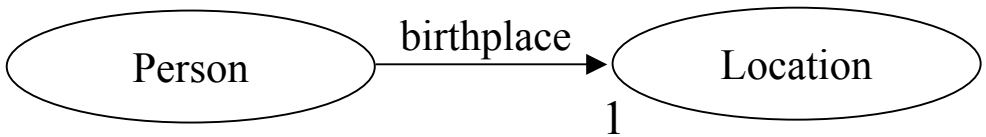

Thus, the Person OWL Ontology enables this inference to be made:

**Inference**: Hawaii, Sandwich Islands, and Aloha State all represent the same location!



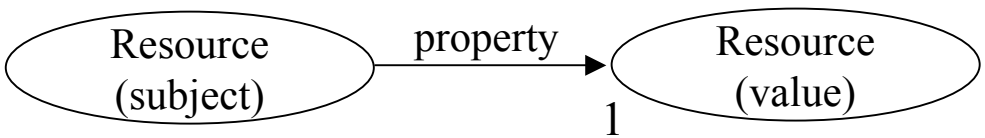
# Lesson Learned

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




This is read as: "A person has exactly one birthplace location."

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

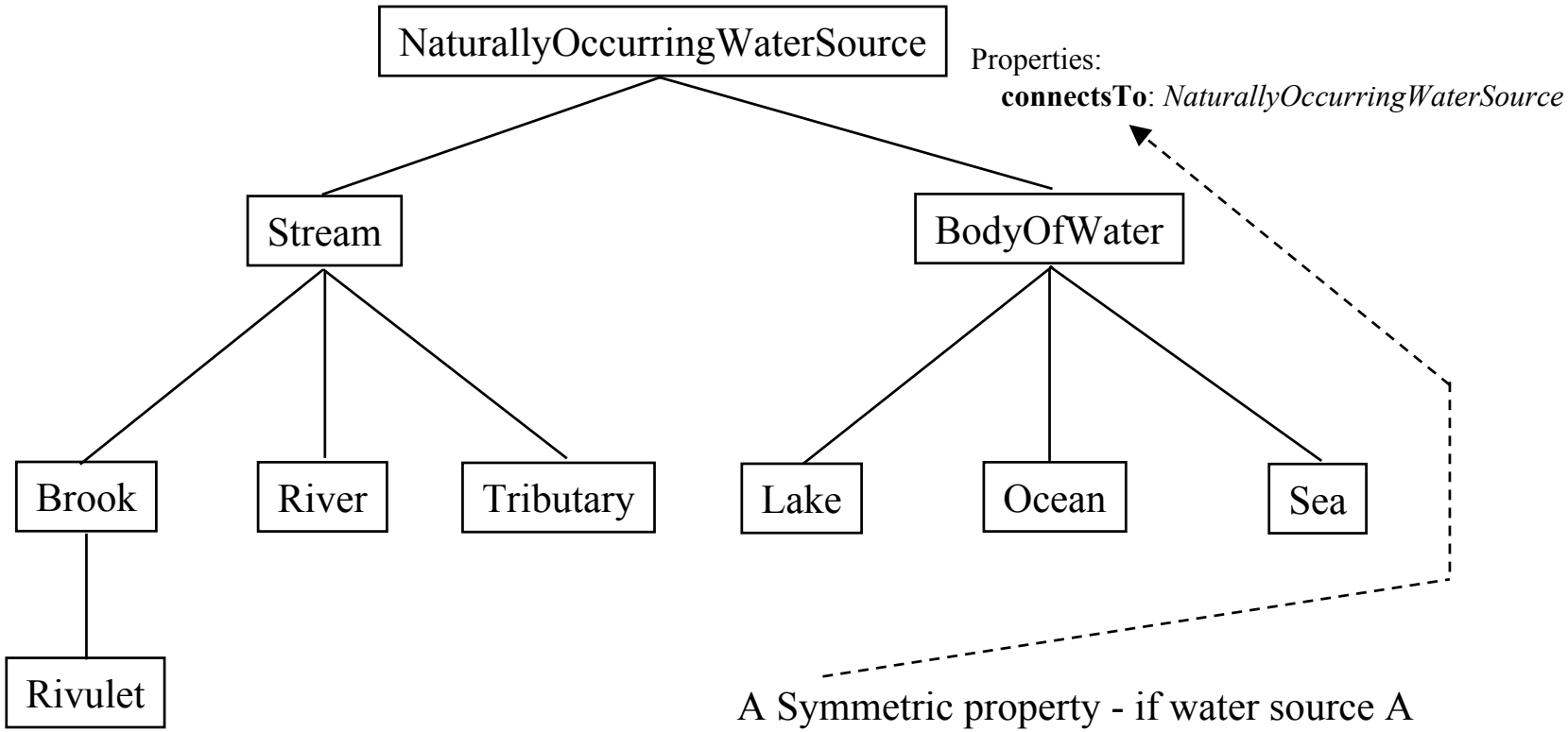
- The preceding examples demonstrated some of OWL's capabilities:
  - An OWL instance document can be enhanced with an OWL property to indicate that it is the same as another instance.
  - OWL provides the capability to construct taxonomies (class hierarchies). Such taxonomies can be used to dynamically 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

# Defining Property Characteristics

- RDF Schema provides three ways to characterize a property:
  - range: use this to indicate the range of values for a property.
  - domain: use this to associate a property with a class.
  - subPropertyOf: use this to specialize a property.
- Note: OWL documents 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 inferencing.

# Symmetric Properties




A Symmetric property - if water source A connectsTo water source B then water source B connects to water source A.

# Symmetric Property

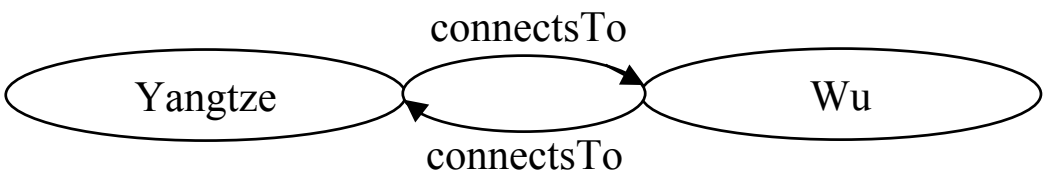
Assume that connectsTo has been *defined, in an OWL document, to be a Symmetric property*:

```
<?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>
```

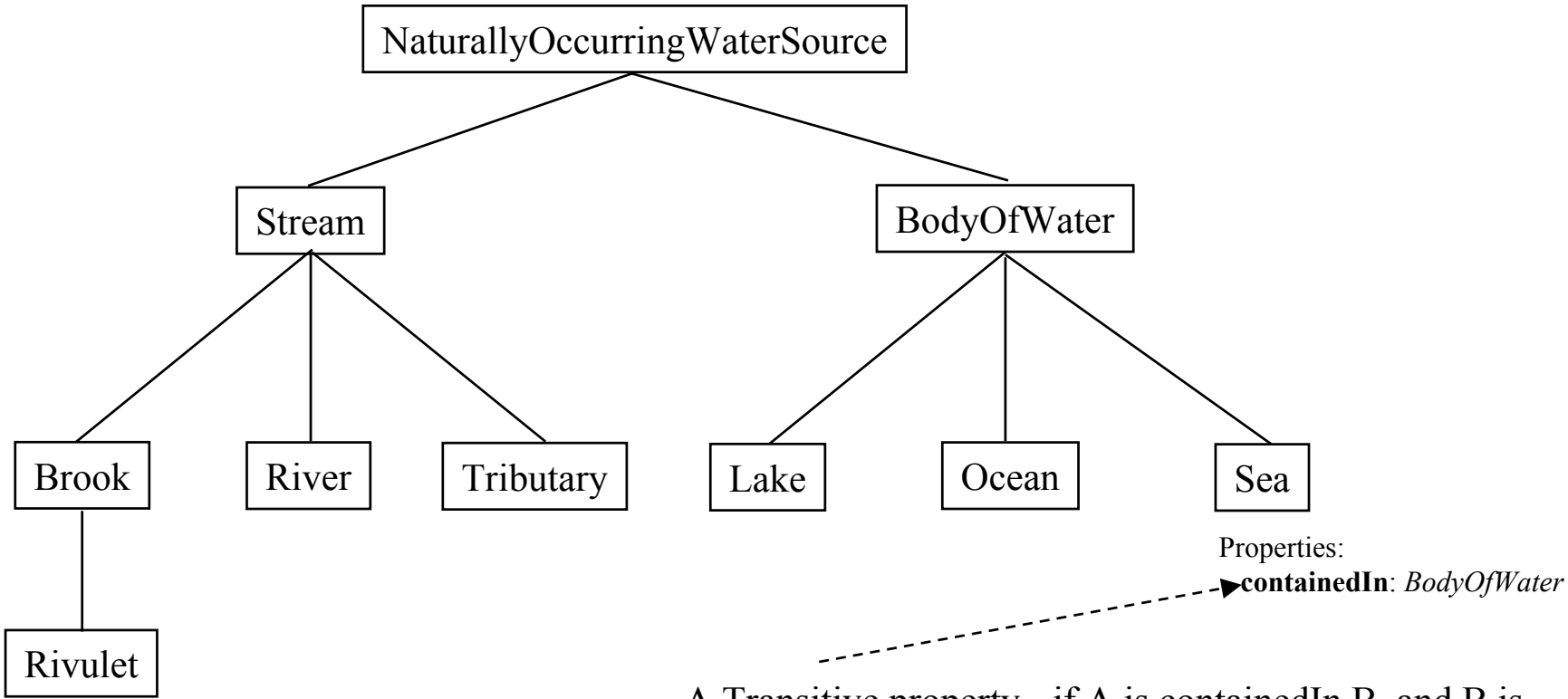
**Yangtze.rdf**

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

The Wu River connectsTo the Yangtze River.



# Transitive Properties




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

# 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>
```

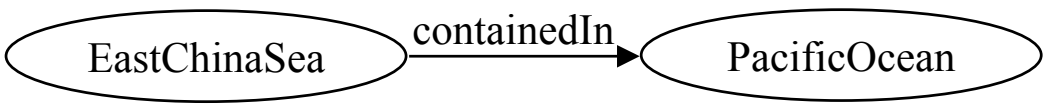
**ChinaSea.rdf**

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

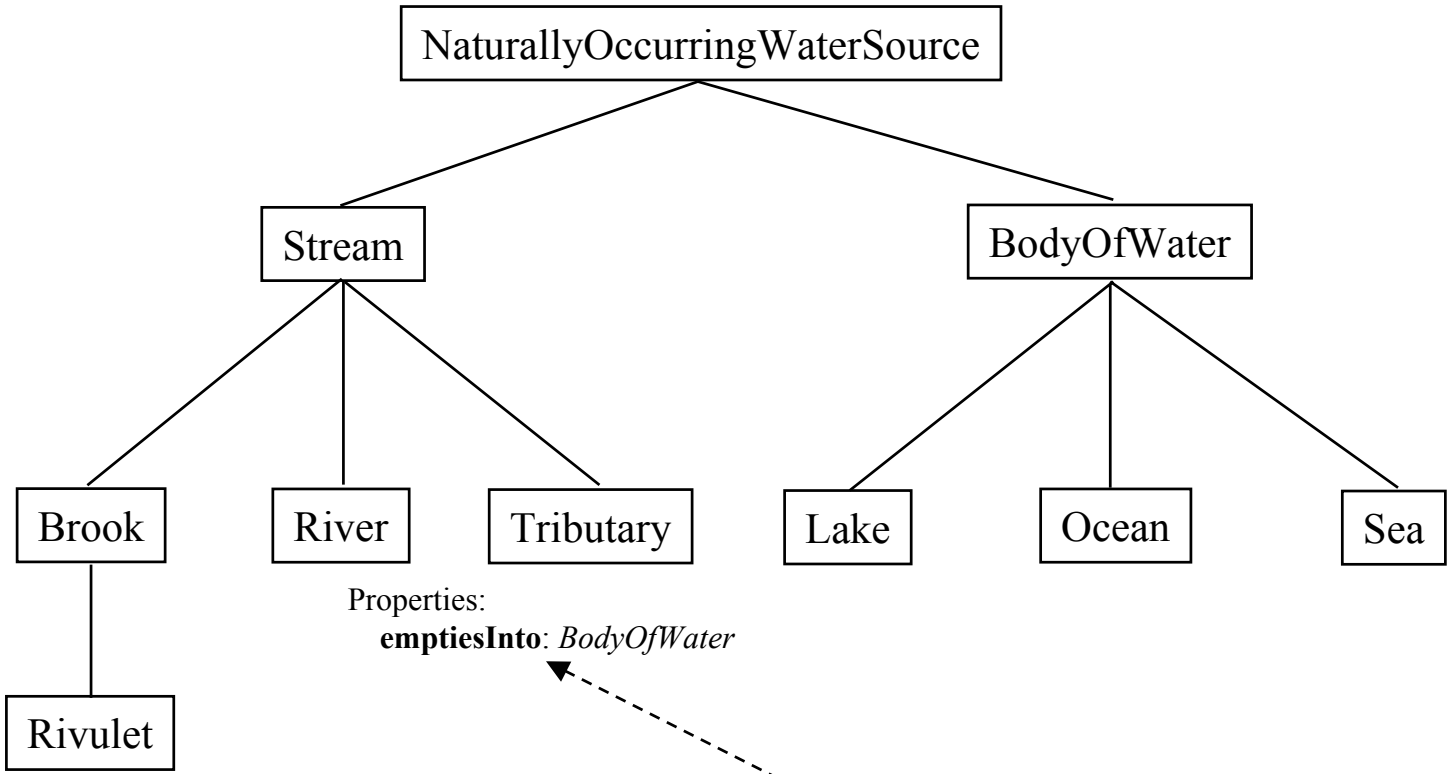
The EastChinaSea is containedIn the PacificOcean.

# Transitive Property


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



# Functional Properties




A Functional property - for each instance there is at most one value for the property.

# 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**

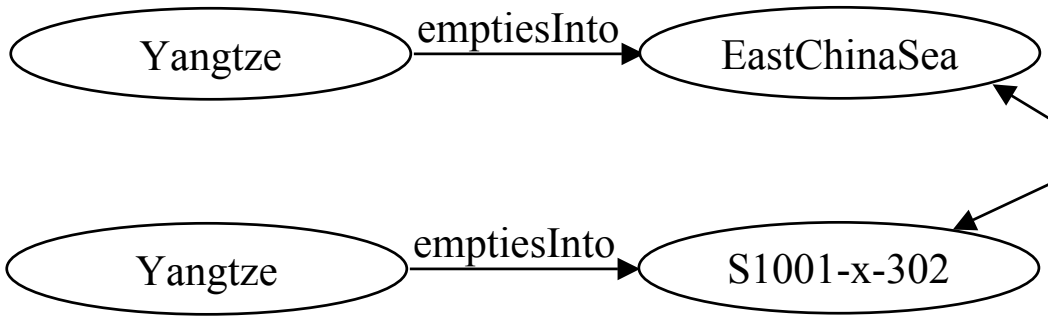
```
<?xml version="1.0"?>
<River rdf:about="http://www.china.org/rivers#Yangtze"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns="http://www.geodesy.org/water/naturally-occurring#">
   <emptiesInto rdf:resource="http://www.national-geographic.org#S1001-x-302"/>
</River>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

**Yangtze-doc2.rdf**

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

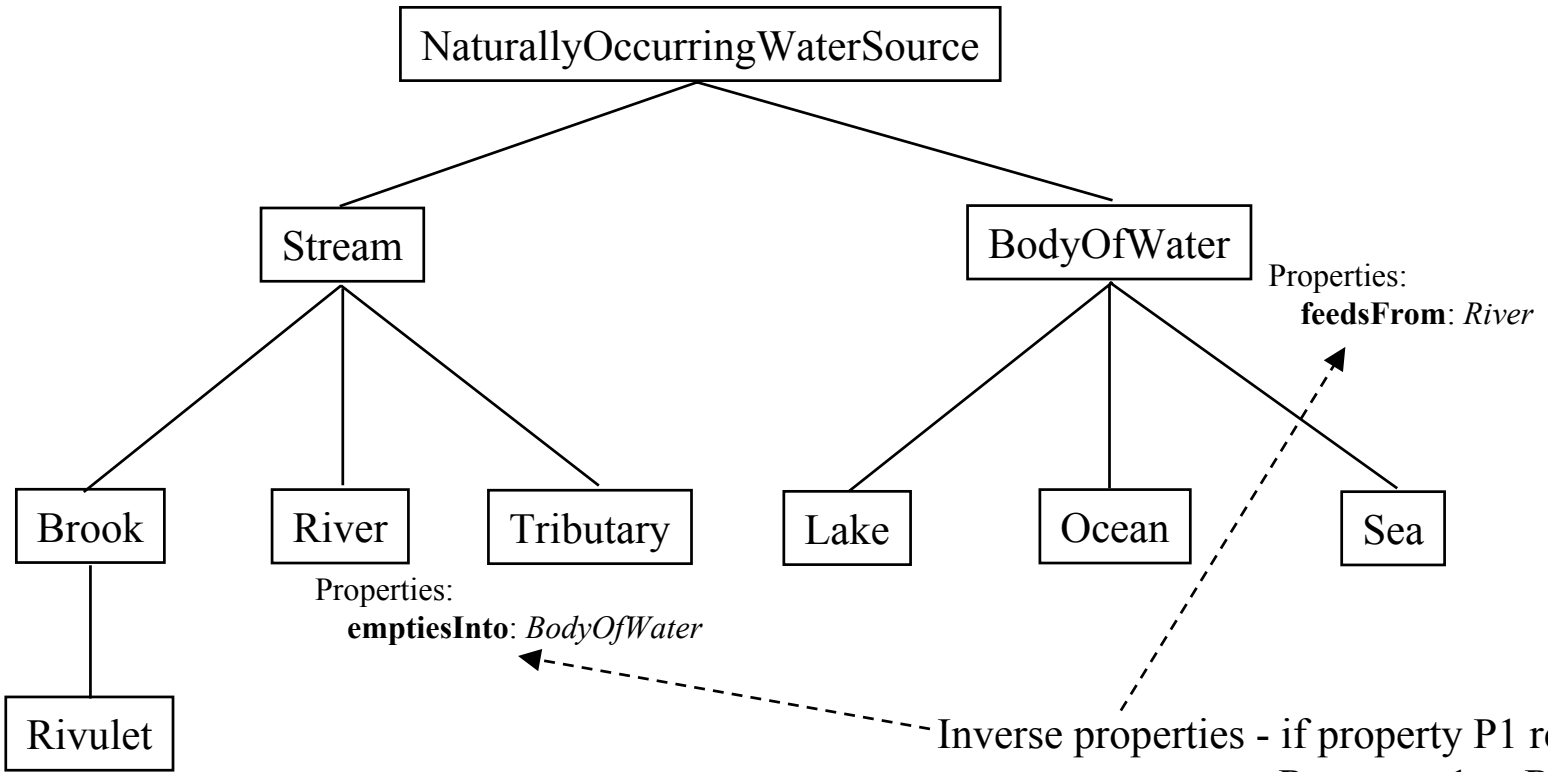
http://www.china.org/geography#**EastChinaSea** = http://www.national-geographic.org#**S1001-x-302**

# Functional Property (cont.)




If emptiesInto has been defined to be Functional then we can infer that these two values must refer to the same thing.

# 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:

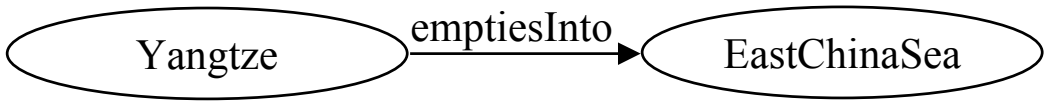
The Yangtze emptiesInto the EastChinaSea.

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

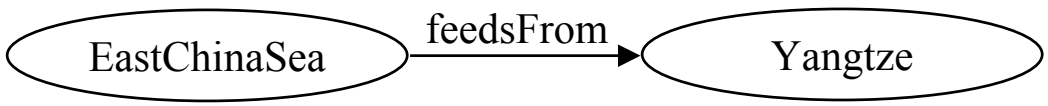
The EastChinaSea feedsFrom the Yangtze.

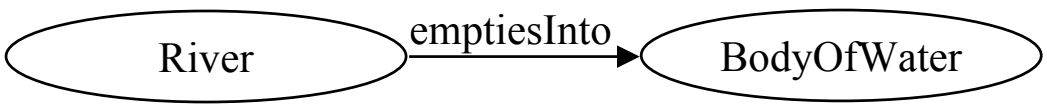
# emptiesInto <---> feedsFrom

## (Inverse Properties)

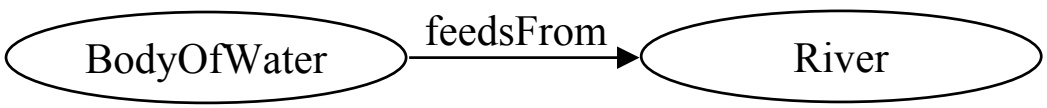



**A specific instance:**

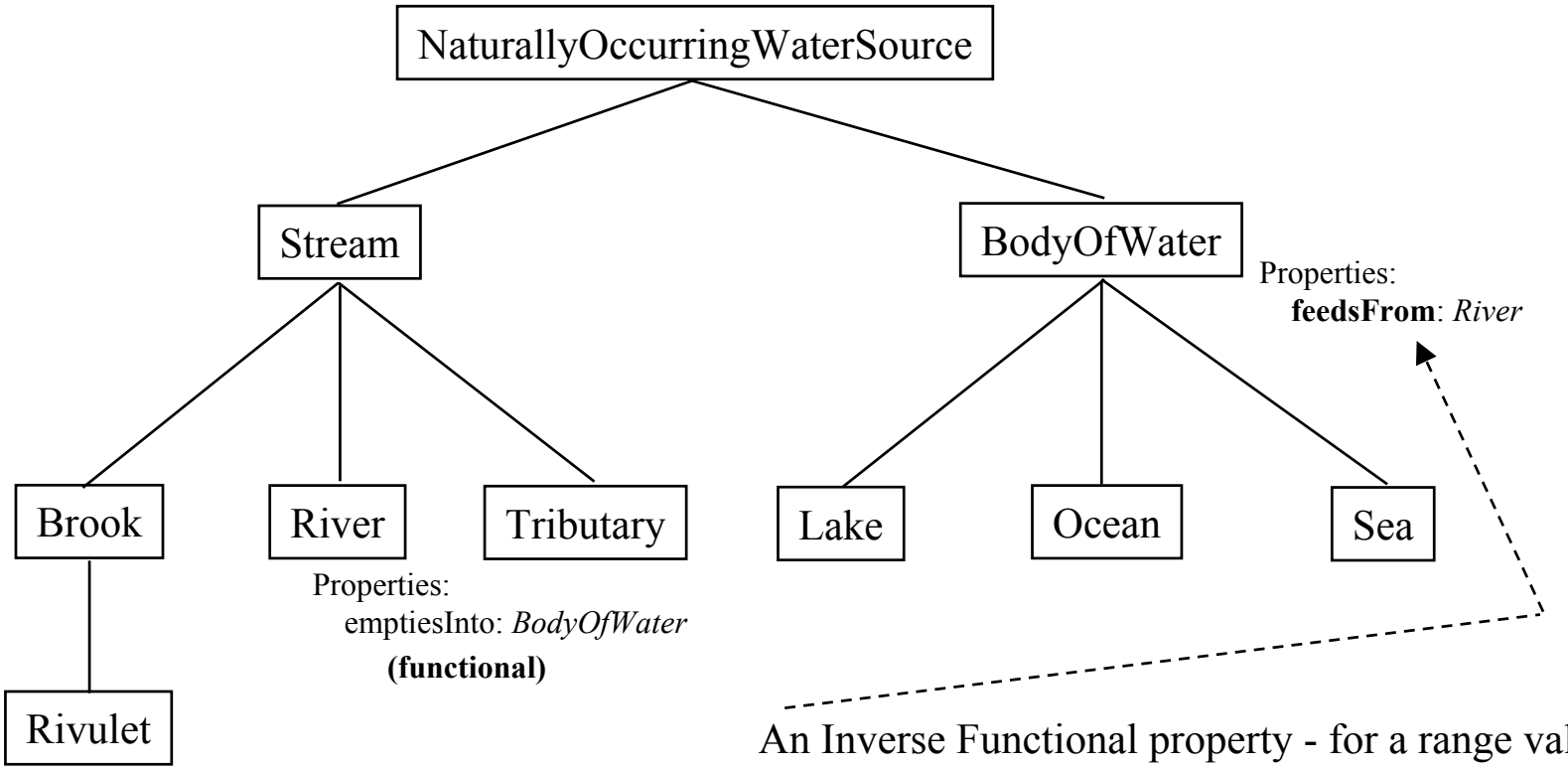



**The general case:**



# Inverse Functional Properties




An Inverse Functional property - for a range value the domain is unique.

# 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>
```

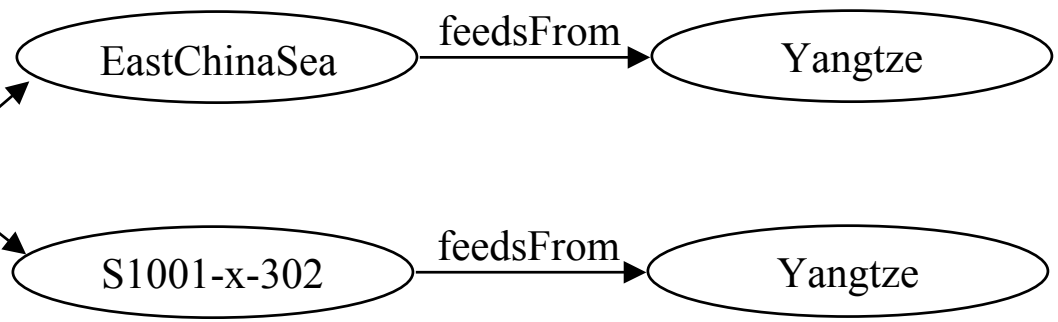
**S1001-x-302.rdf**

# Inverse Functional Property (cont.)

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

EastChinaSea = S1001-x-302.

If feedsFrom has been defined to be Inverse Functional then we can infer that these two Resources must refer to the same thing.



# Time for Syntax!

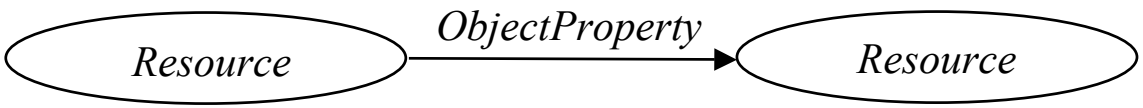
- On the previous slides we have seen the different ways that OWL provides to characterize properties.
- Now let's look at the OWL syntax for expressing these property characteristics.

# Defining Properties in OWL

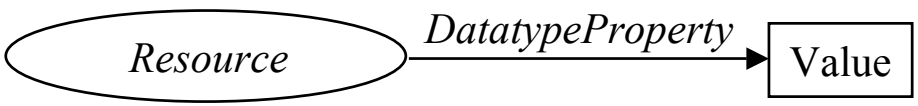
- Recall that with RDF Schema the rdf:Property was used for both:
  - relating a Resource to another Resource
    - Example: The emptiesInto property relates a River to a BodyOfWater.
  - relating a Resource to an rdfs:Literal or a datatype
    - Example: The length property relates a River to a xsd:nonNegativeInteger.
- 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

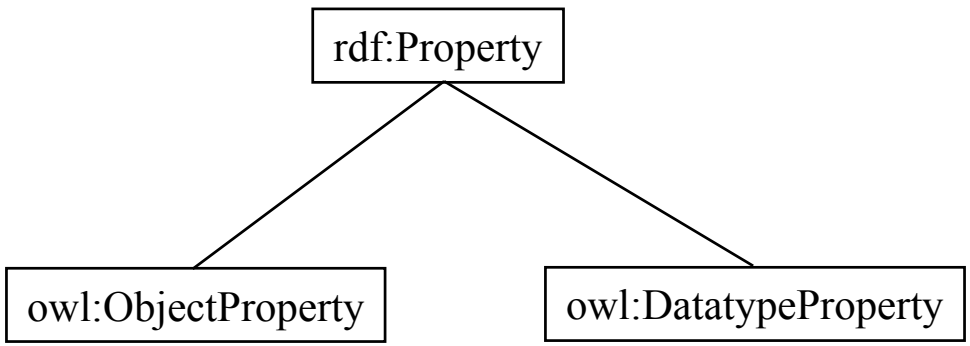
An ObjectProperty relates one Resource to another Resource:

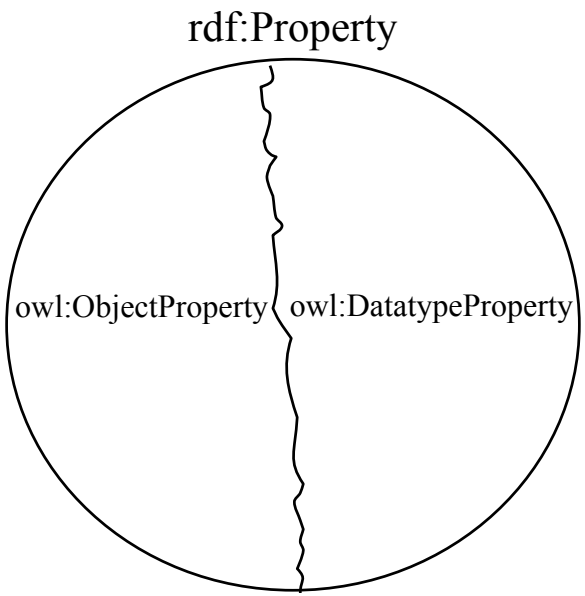



A DatatypeProperty relates a Resource to a Literal or an XML Schema datatype:



# 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>
```

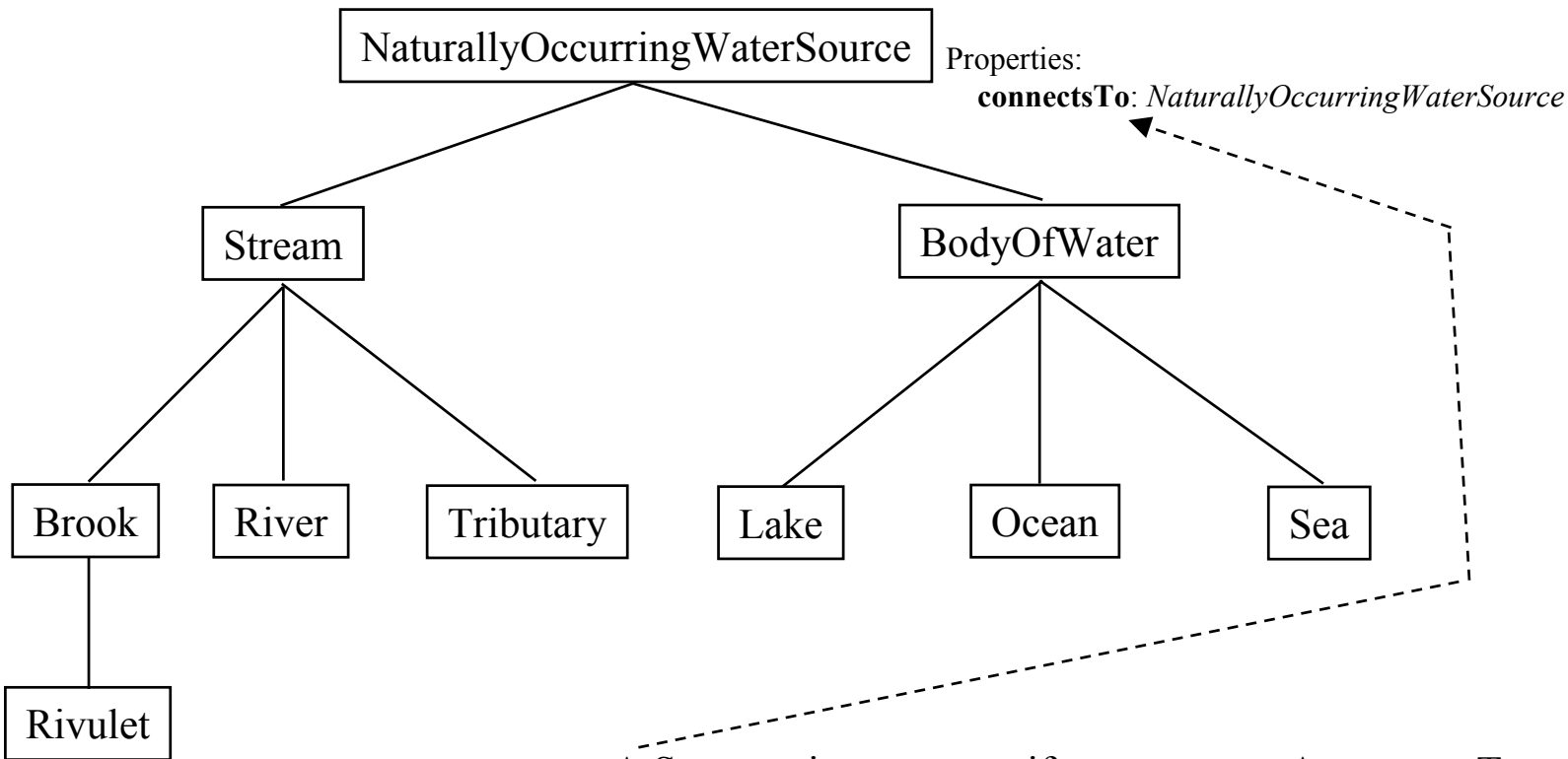
naturally-occurring.owl (snippet)

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




A Symmetric property - if water source A connectsTo water source B then water source B connects to water source A.

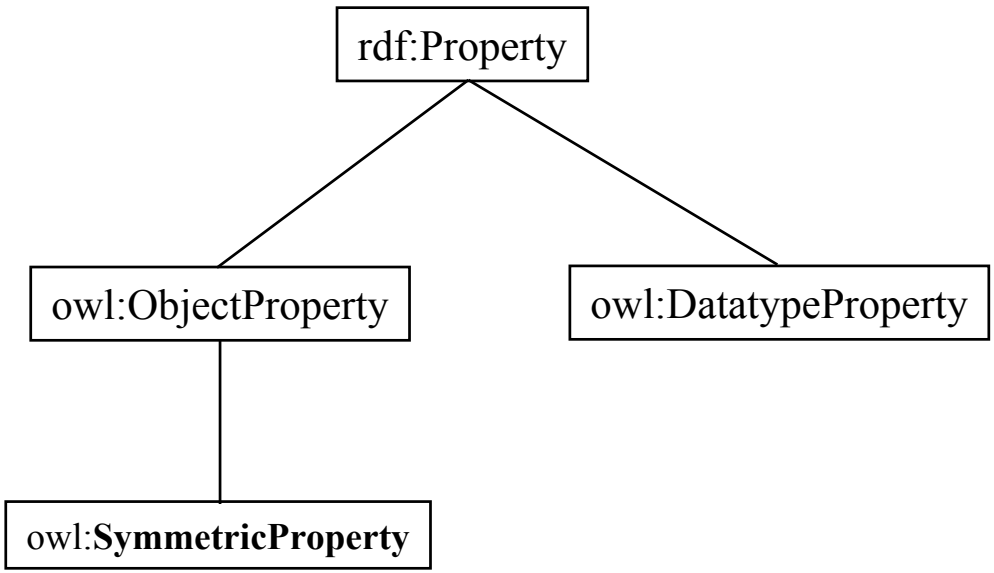
# Syntax for indicating that a property is Symmetric

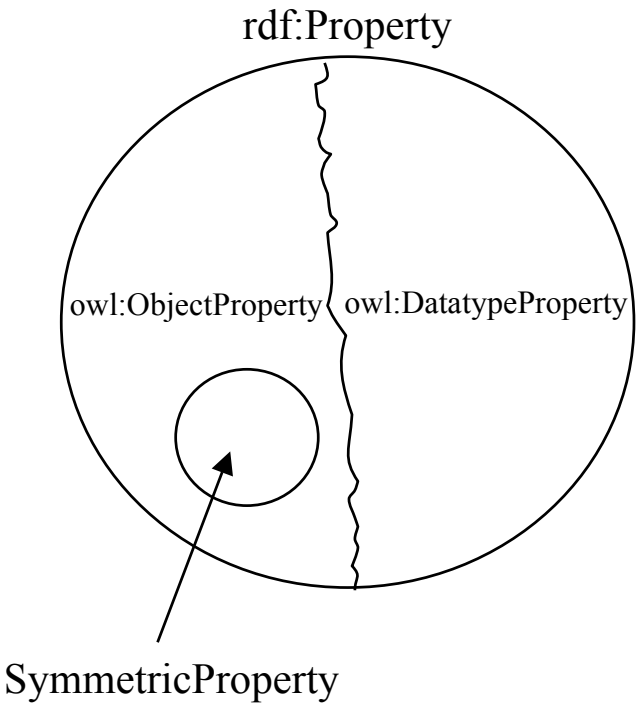
```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xml:base="http://www.geodesy.org/water/naturally-occurring">

  <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."

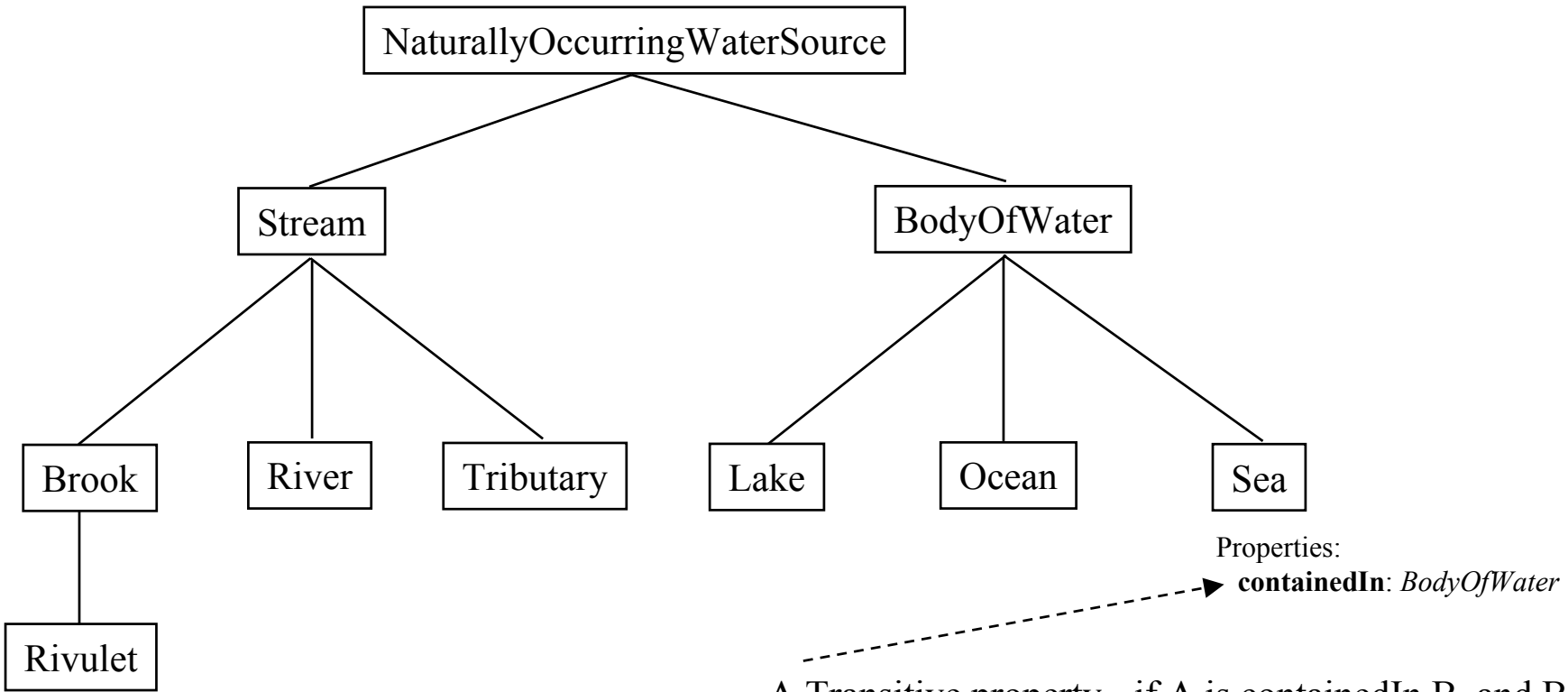
2

```
<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



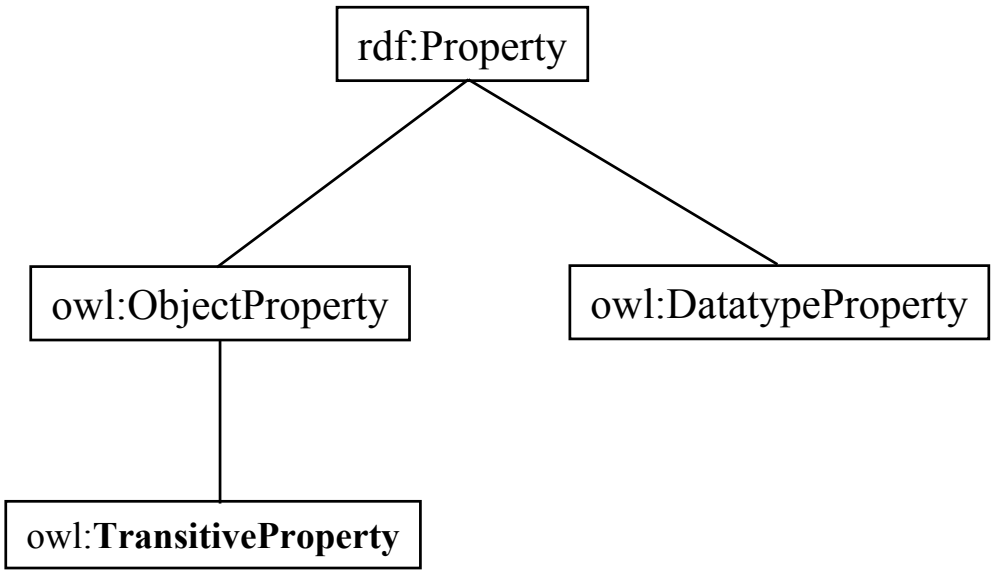

A Transitive property - if A is containedIn B, and B is containedIn C then A is containedIn C.

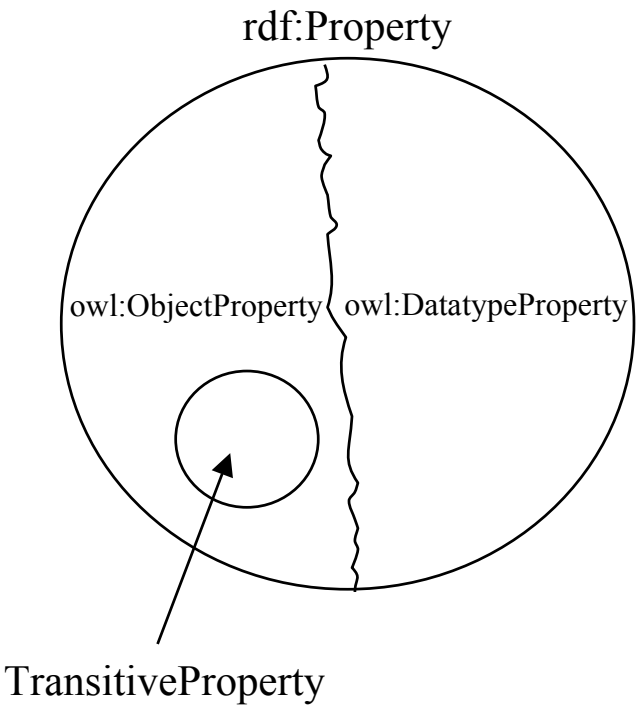
# Syntax for indicating that a property is Transitive

```
<?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)**

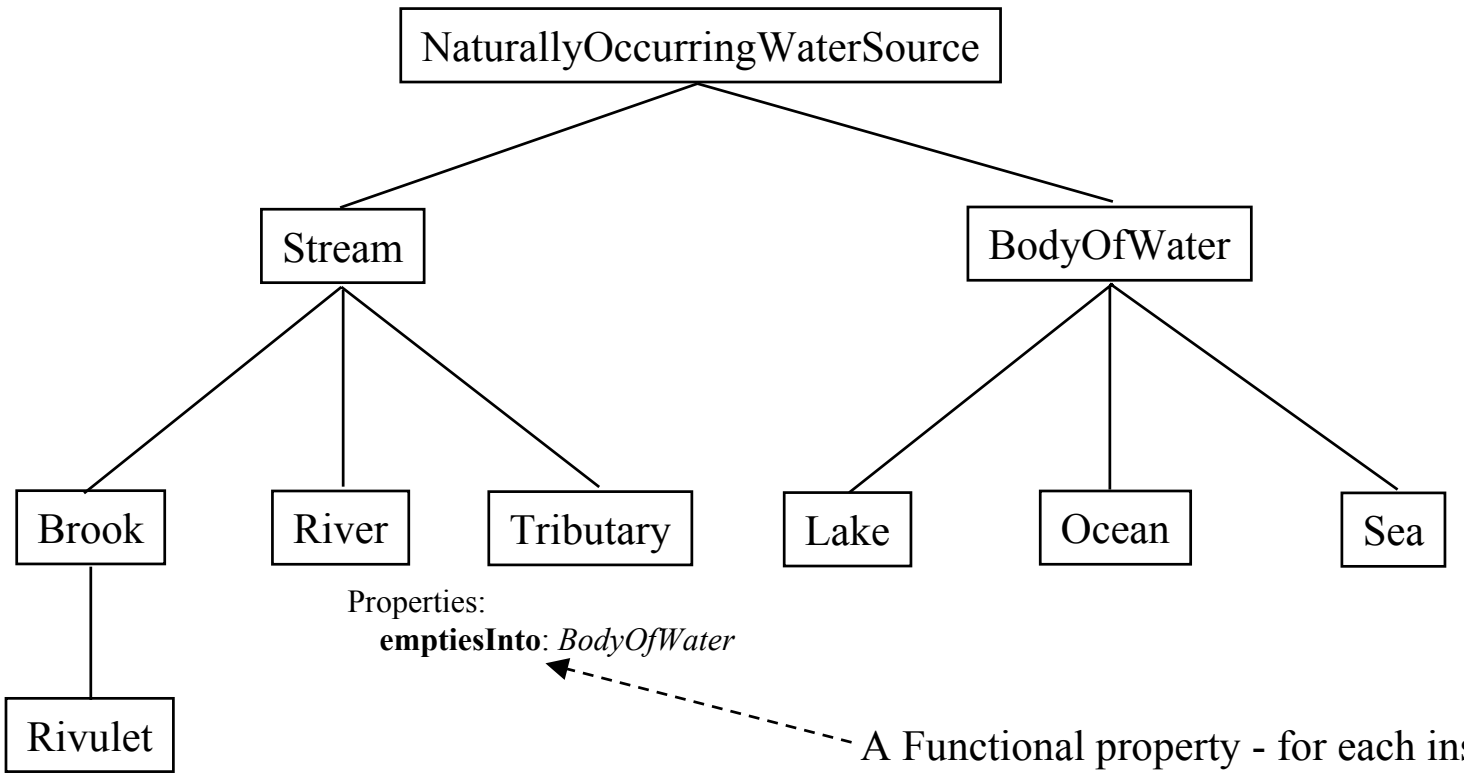
# owl:TransitiveProperty is a subclass of owl:ObjectProperty




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



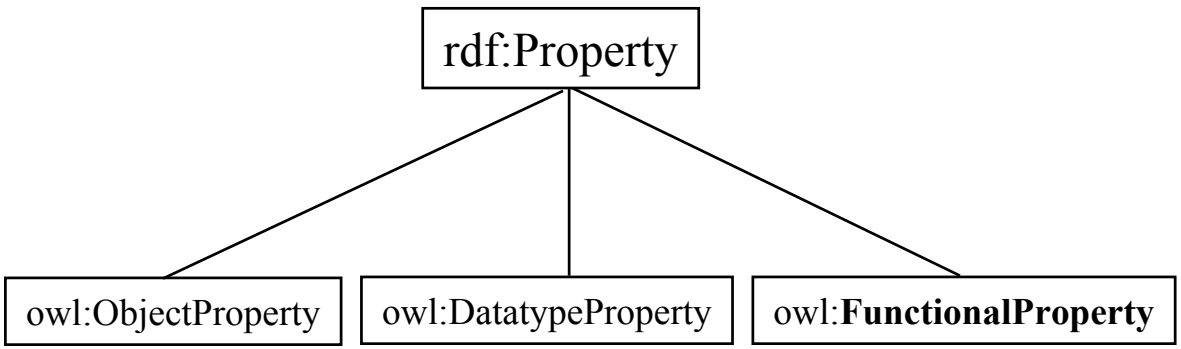

A Functional property - for each instance there is at most one value for the property.

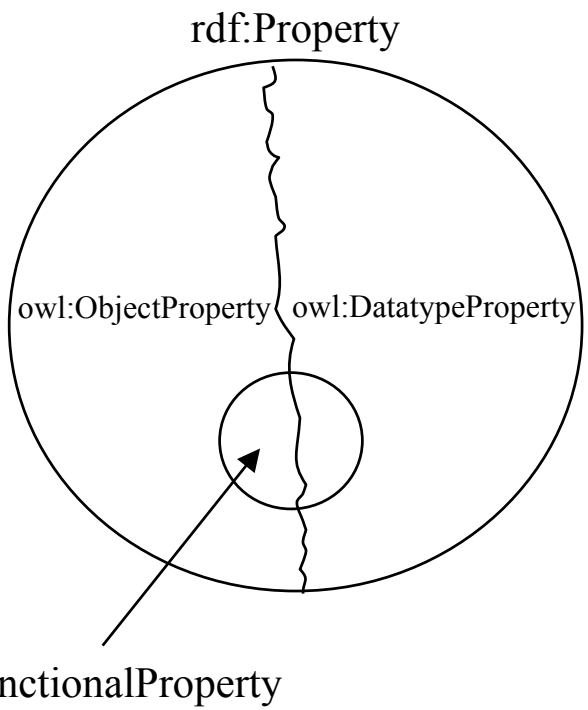
# Syntax for indicating that a property is Functional

```
<?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)**

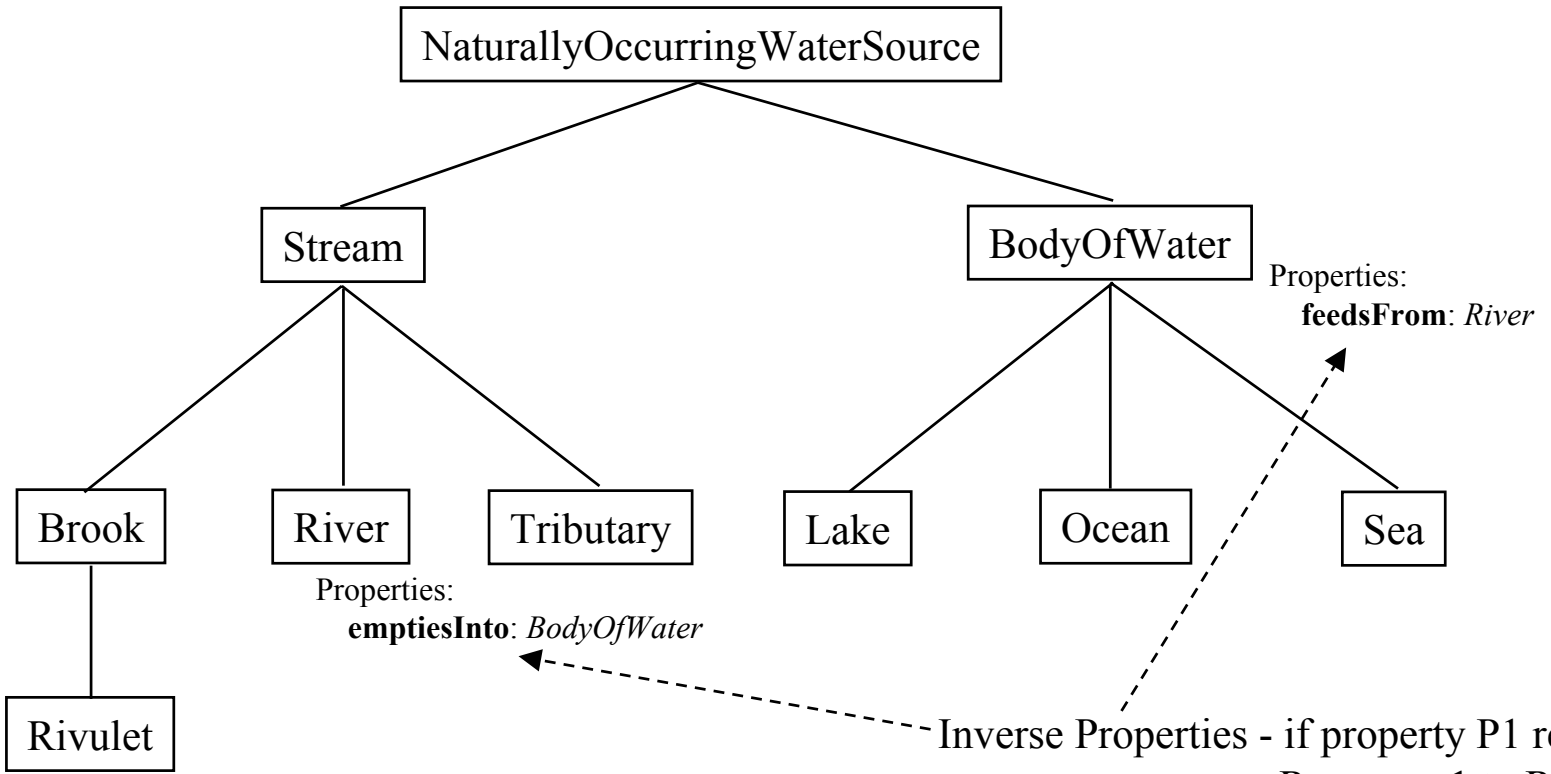
# owl:FunctionalProperty is a subclass of rdf:Property




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

# Defining Inverse Properties




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

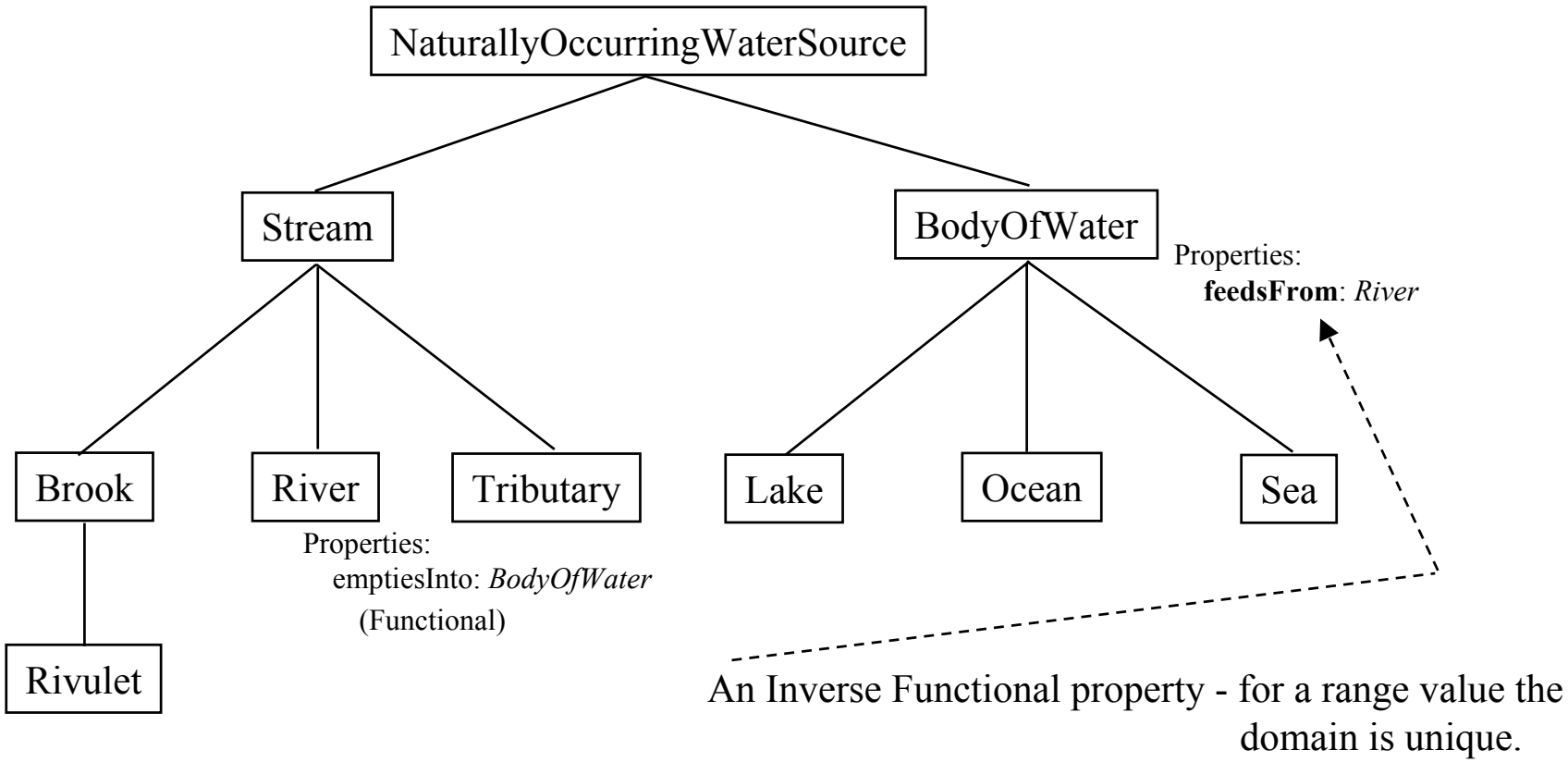
# Syntax for indicating that a property is the inverse of another property

```
<?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

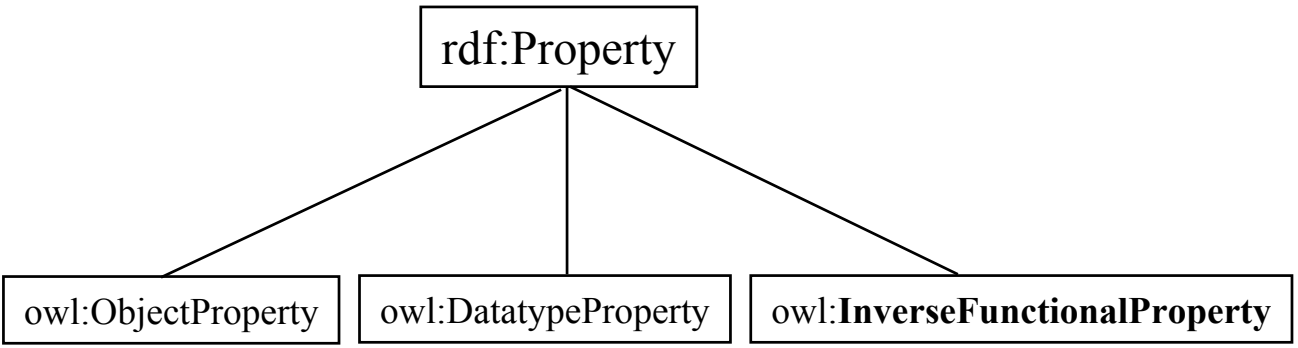


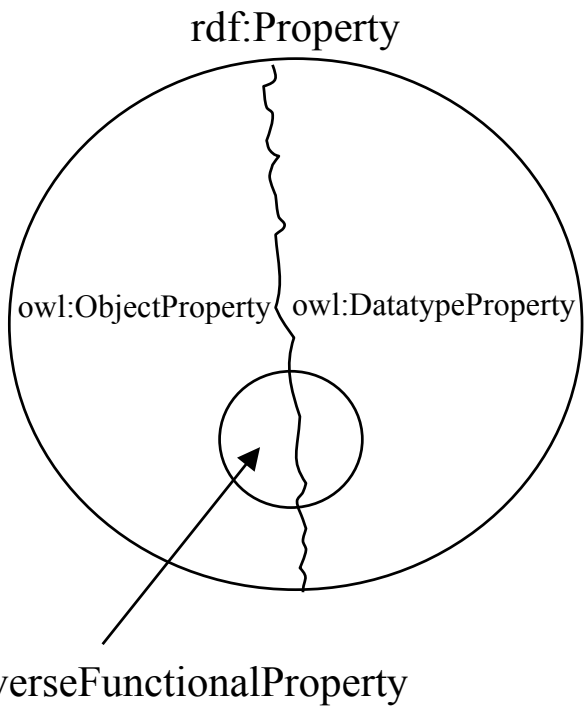
# Syntax for indicating that a property is Inverse Functional

```
<?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



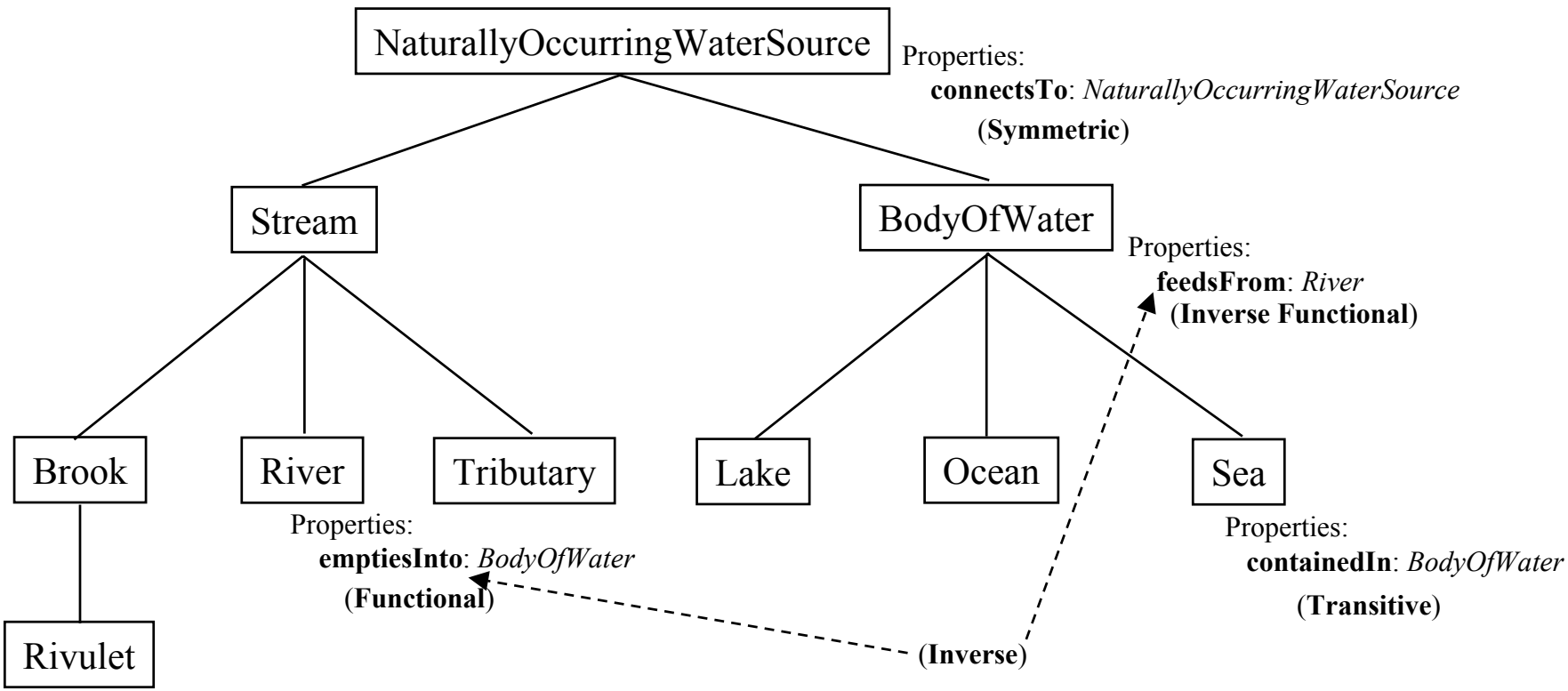




Consequently, the range of an InverseFunctionalProperty can be either a Resource or a Literal or a datatype.

# 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.
  - 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

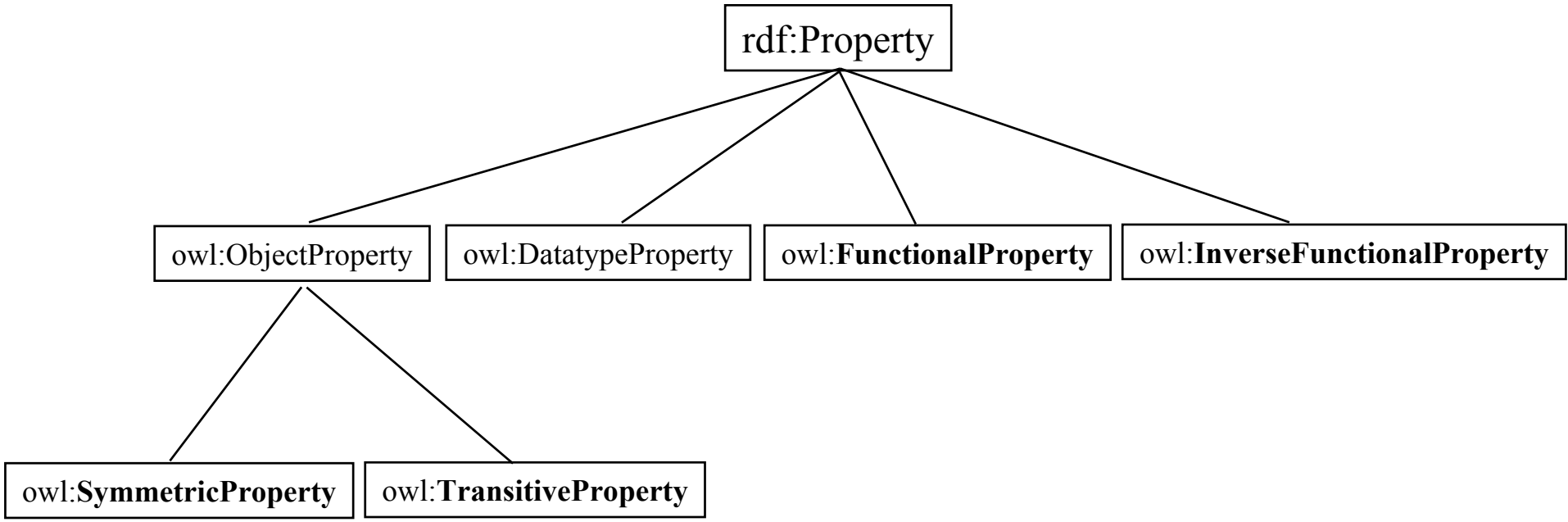
```
<?xml version="1.0"?>
<River rdf:ID="Yangtze"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns="http://www.geodesy.org/water/naturally-occurring#">
   <emptiesInto rdf:resource="http://www.china.org/geography#EastChinaSea"/>
   <connectsTo rdf:resource="http://www.china.org/rivers#Wu"/>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

**Yangtze.rdf**

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:

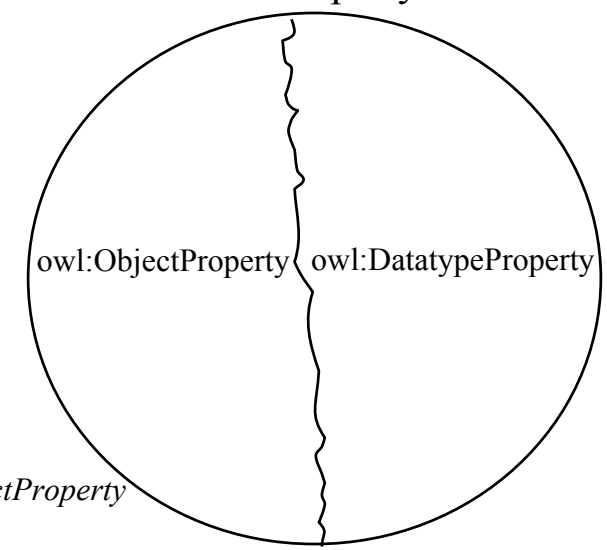
1. SymmetricProperty and TransitiveProperty can only be used to relate Resources to Resources.
2. FunctionalProperty and InverseFunctionalProperty can be used to relate Resources to Resources, or Resources to an RDF Schema Literal or an XML Schema datatype.

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

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

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

property



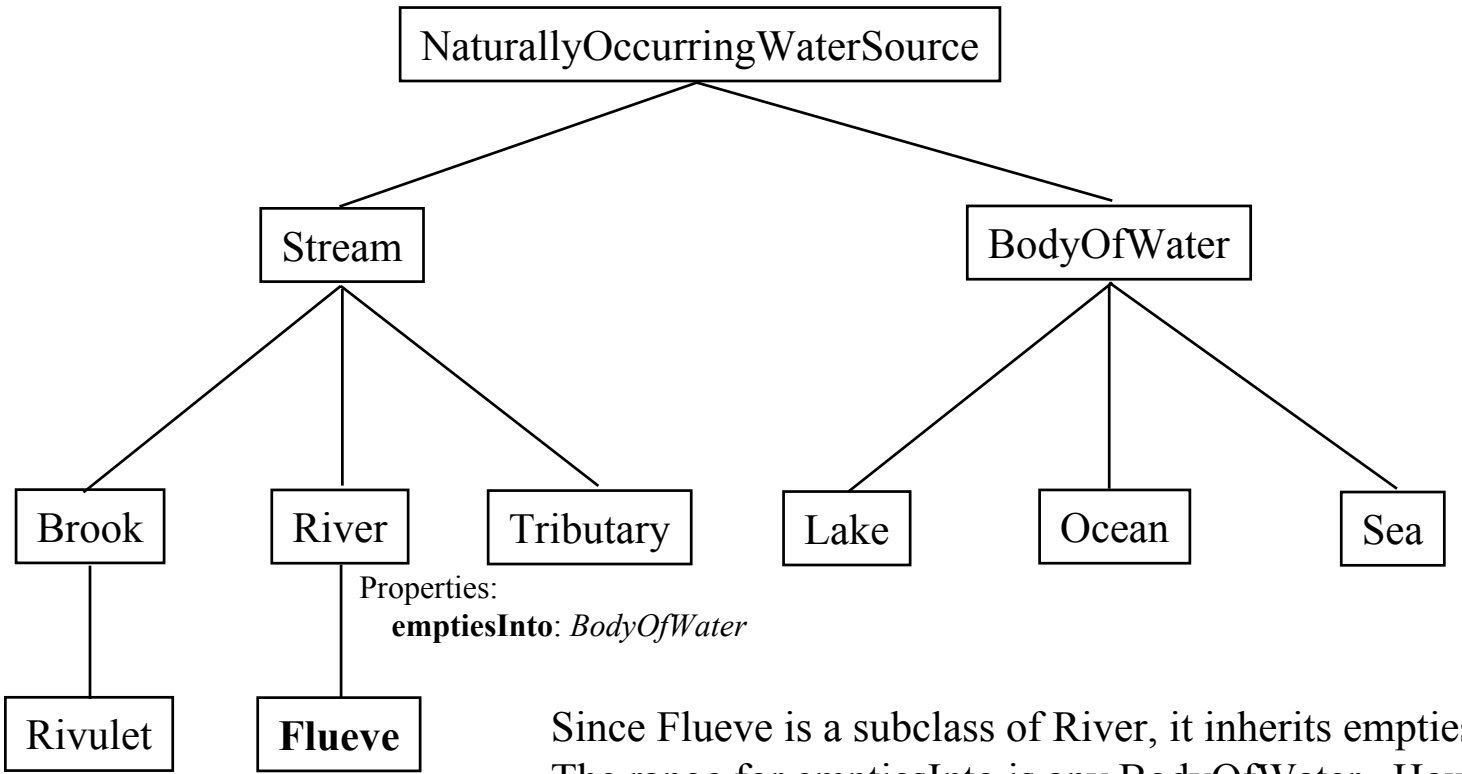

Properties:
**inverseOf**: *owl:ObjectProperty*

Do Lab1

# 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 Flueve is a subclass of River, it inherits emptiesInto. The range for emptiesInto is any BodyOfWater. However, the definition of a Flueve (French) is: "a River which emptiesInto a Sea". Thus, *in the context of the Flueve 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 Flueve class*, we would like the emptiesInto property to have its range restricted to just the Sea class. Thus, for the Flueve 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.

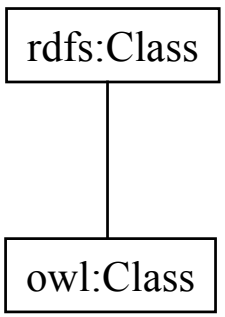
```
<rdfs:Class rdf:ID="River">
    <rdfs:subClassOf rdf:resource="#Stream"/>
</rdfs:Class>
```

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 Flueve) 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="Flueve">
    <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>
```

**naturally-occurring.owl (snippet)**

# Flueve is a subclass of an "anonymous class"

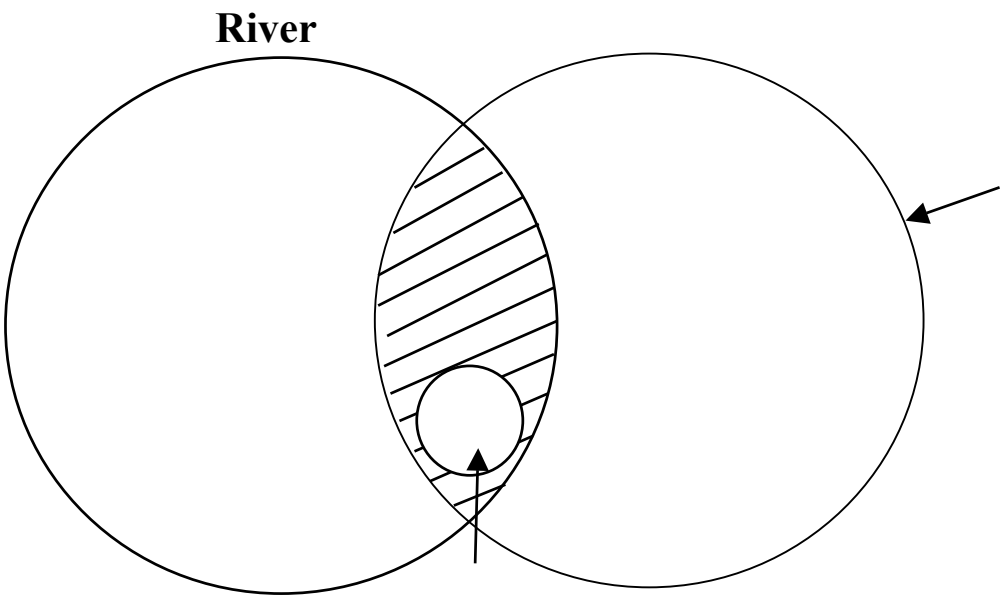
```
<owl:Class rdf:ID="Flueve">
    <rdfs:subClassOf rdf:resource="#River"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#emptiesInto"/>
            <owl:allValuesFrom rdf:resource="#Sea"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

anonymous class

This is read as: "The Flueve 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 Flueve class is a subClassOf River. It has a property emptiesInto. All values for emptiesInto must be instances of Sea."

# Definition of Flueve




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

**Flueve** - a River that emptiesInto a Sea.

# An instance of Flueve

```
<?xml version="1.0"?>
<Flueve 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"/>
</Flueve>
```

**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 Flueve class*.

# Two forms of rdfs:subClassOf

1

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

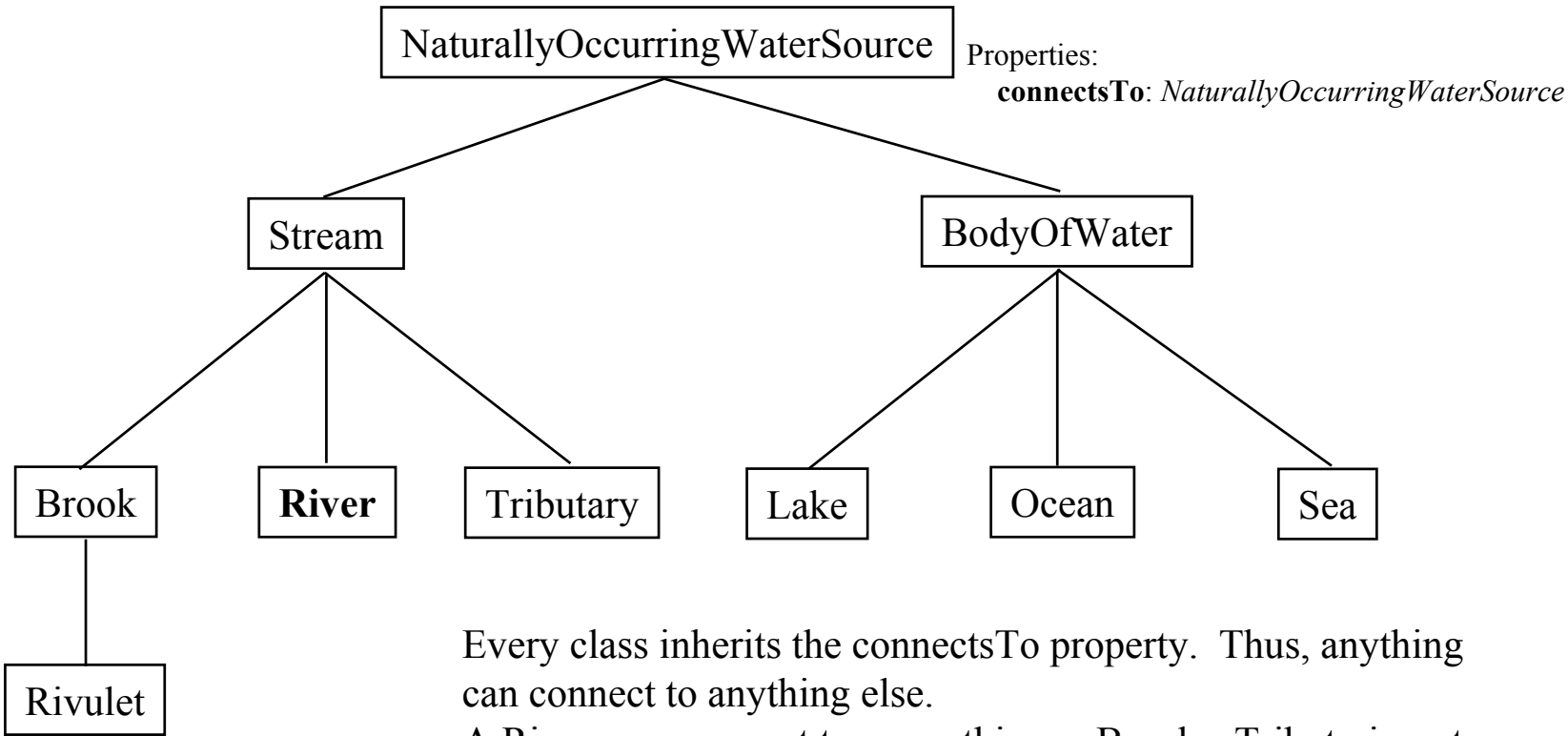
Specify the class using the rdf:resource attribute.

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"/>
```

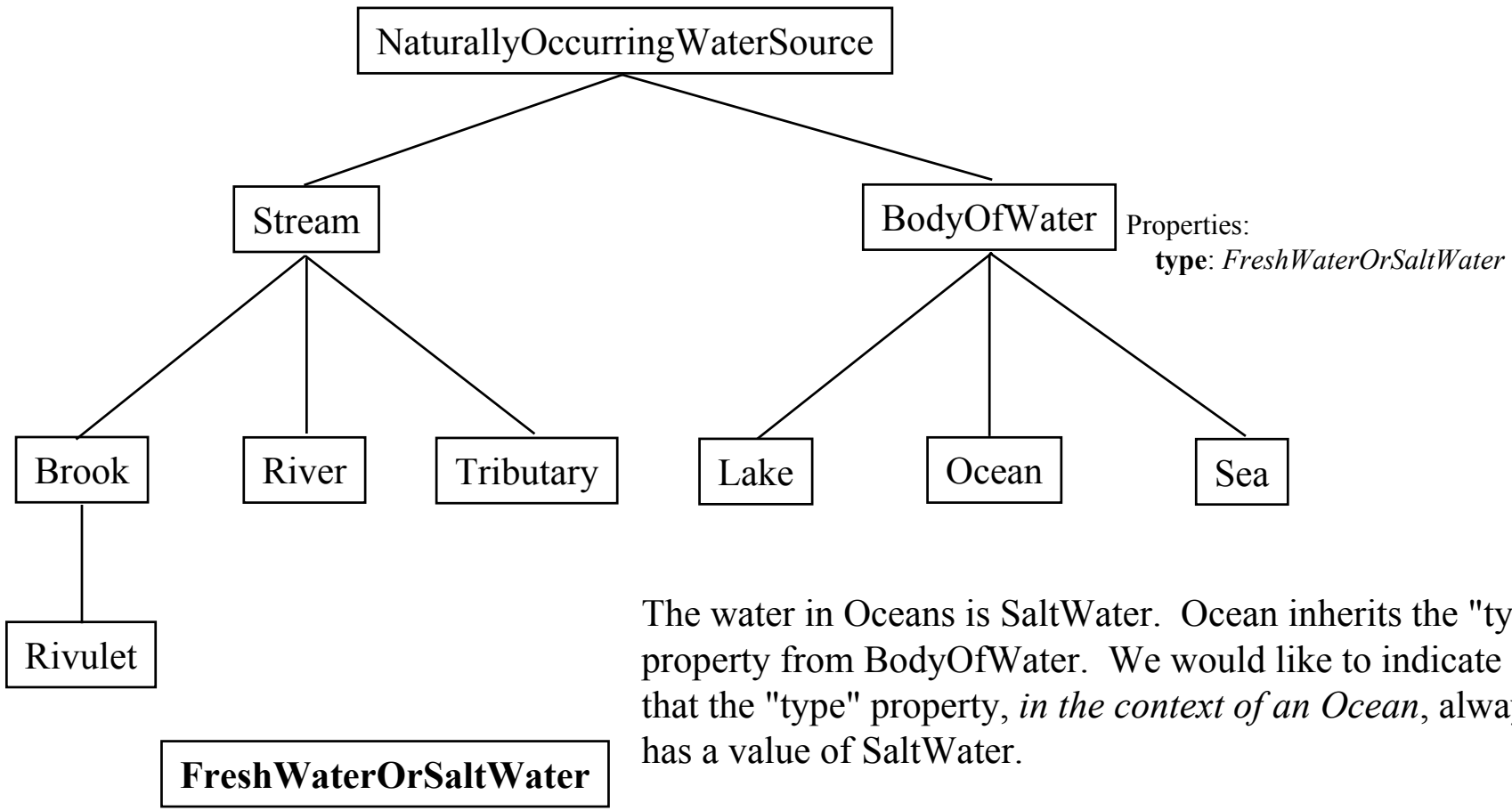
*Wherever there is an emptiesInto property*, all its values must be instances of Sea. [There may be zero emptiesInto properties.]

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

```
<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>
```

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

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*.

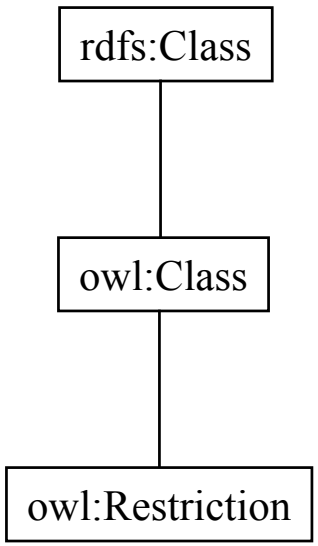
# owl:hasValue means **there exists** a property with the specified value

- 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



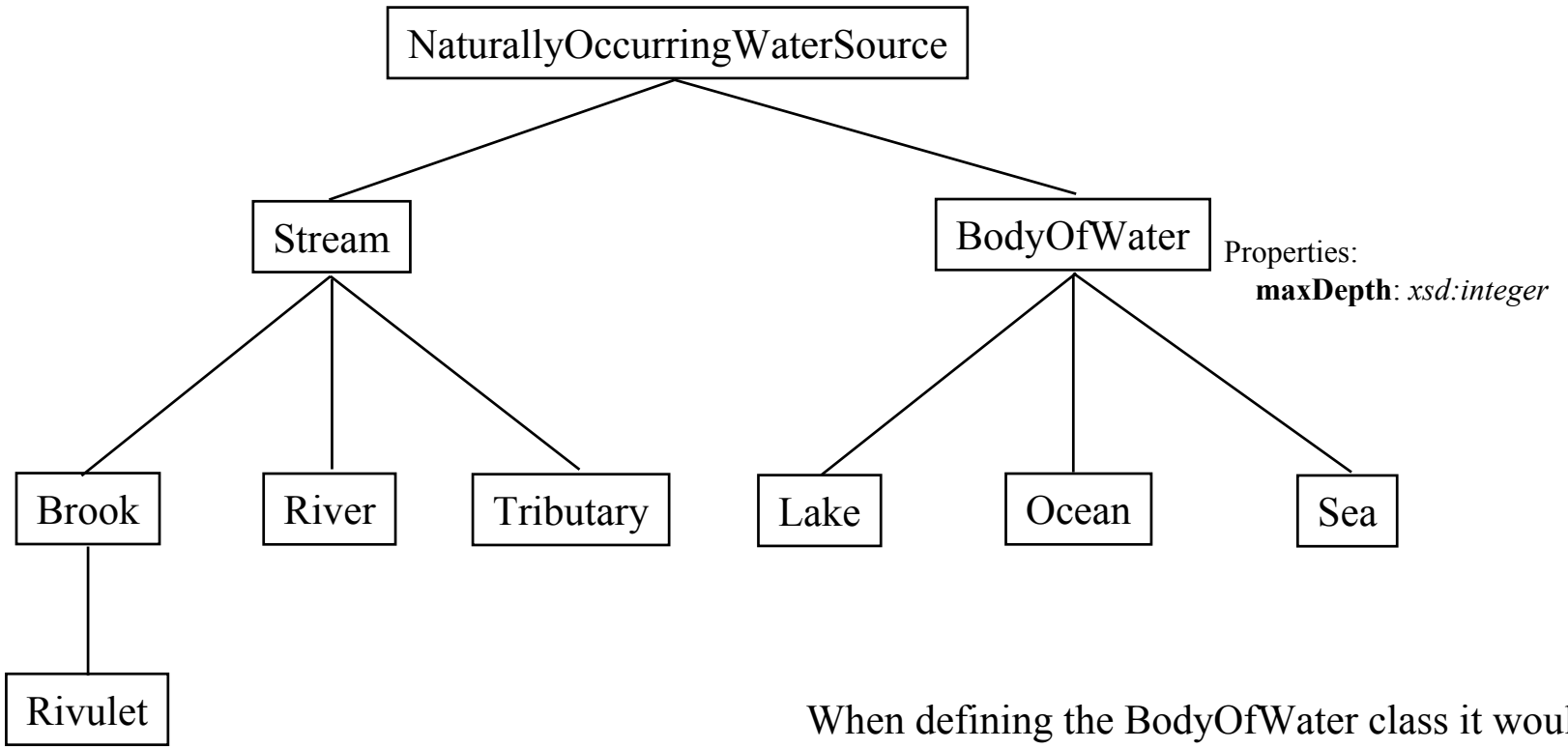

**Properties:**

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)




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**

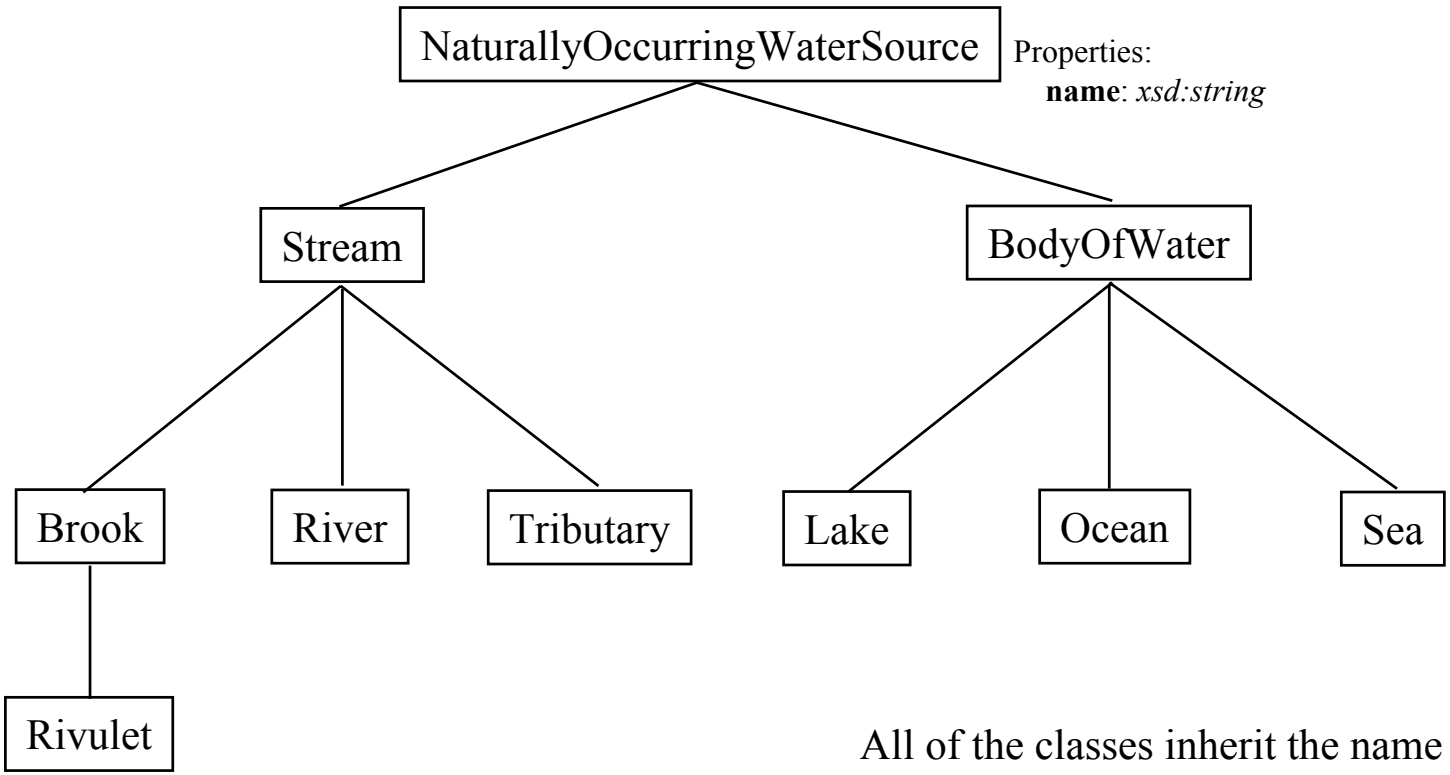
The PacificOcean has only one maxDepth.

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)

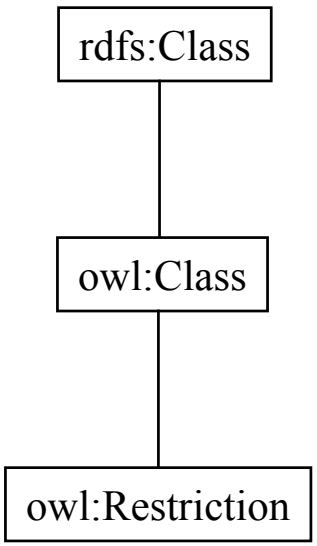
```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xml:base="http://www.geodesy.org/water/naturally-occurring">

  <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



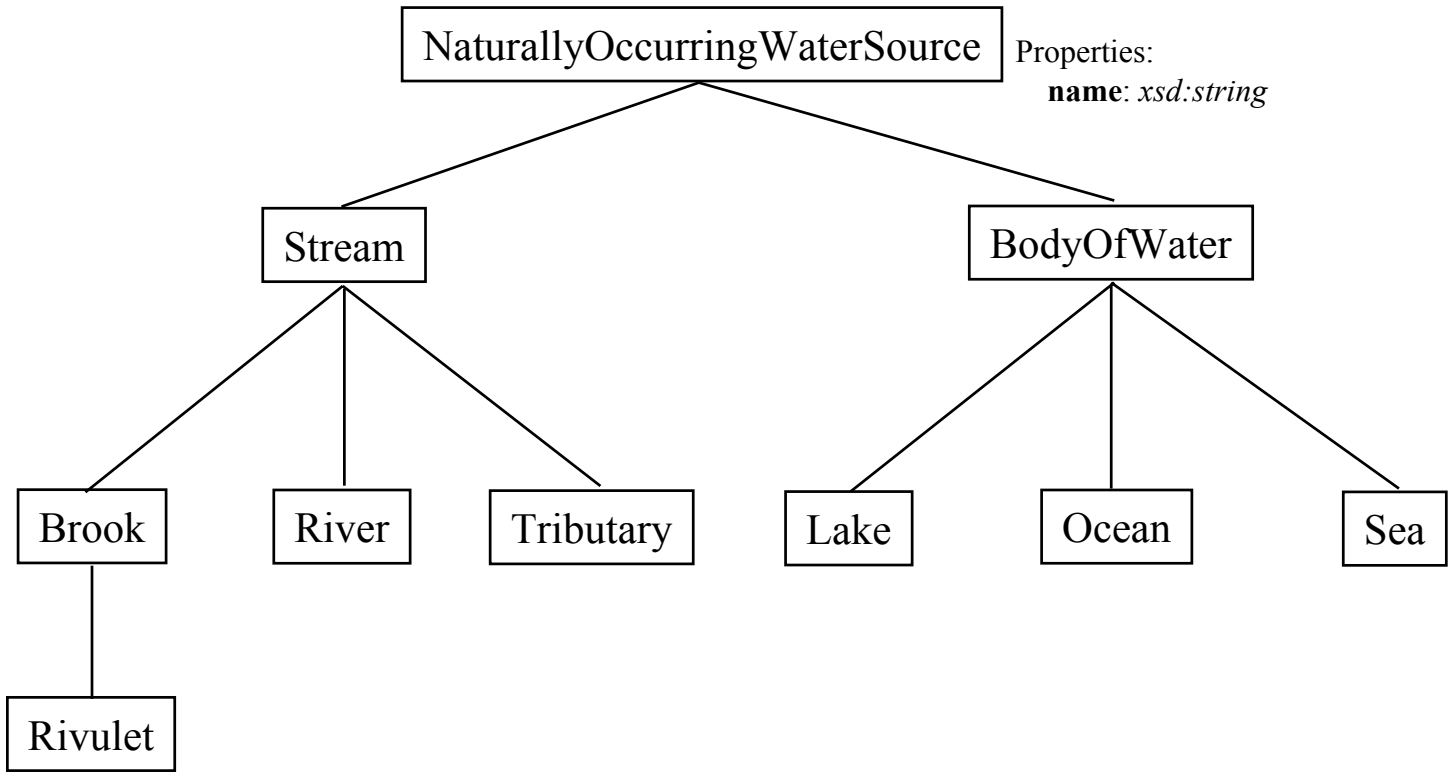

**Properties:**

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.

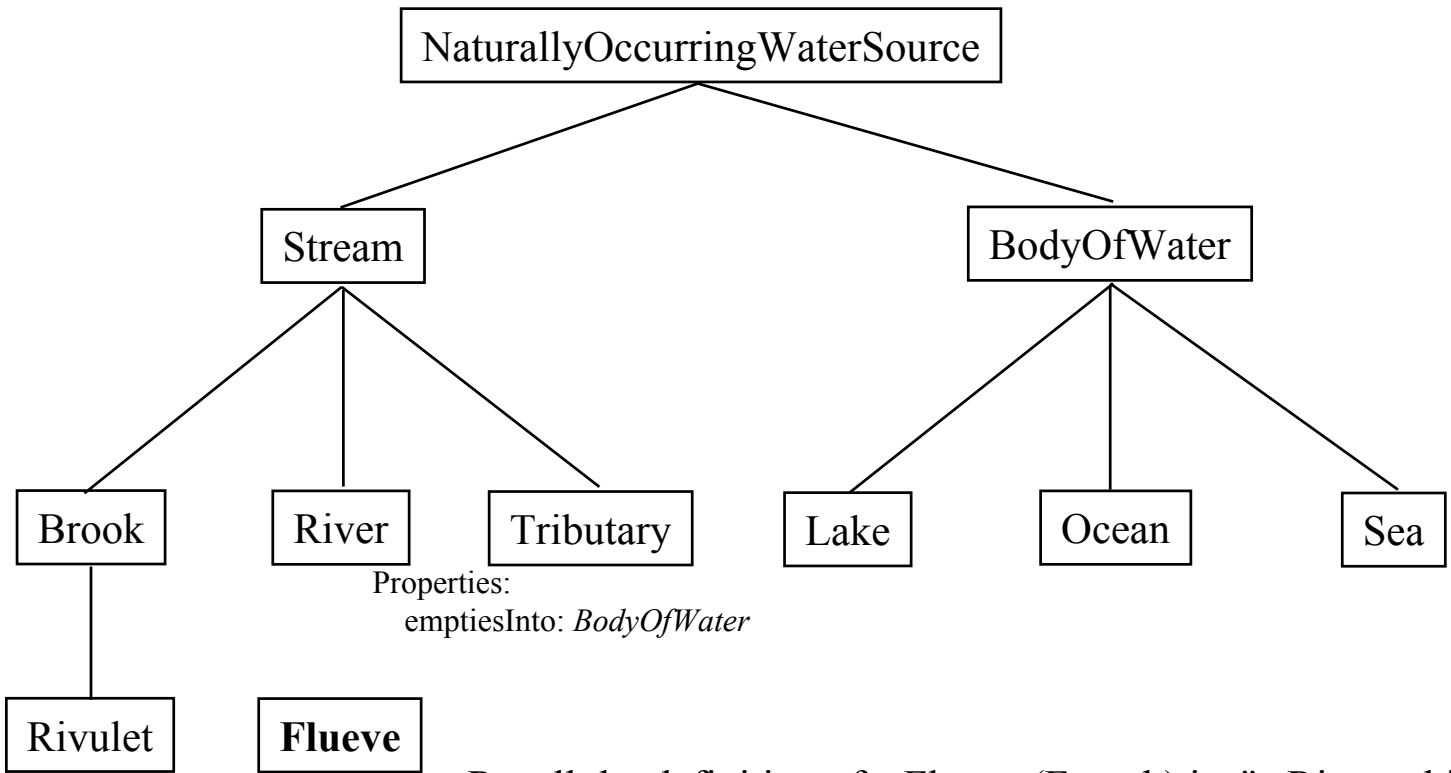
Do Lab2

# Using OWL to Define Classes

# Constructing Classes using Set Operators

- OWL gives you the ability to construct classes using these set operators:
  - intersectionOf
  - unionOf
  - complementOf

# Defining a Flueve class using the intersectionOf operator




Recall the definition of a Flueve (French) is: "a River which emptiesInto a Sea". Thus, a Flueve may be defined as the intersectionOf the River class and an anonymous class containing the emptiesInto property with allValuesFrom Sea.

# Defining Flueve

```
<?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="Flueve">
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#River"/>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#emptiesInto"/>
        <owl:allValuesFrom rdf:resource="#Sea"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>

  ...

</rdf:RDF>
```

**naturally-occurring.owl (snippet)**

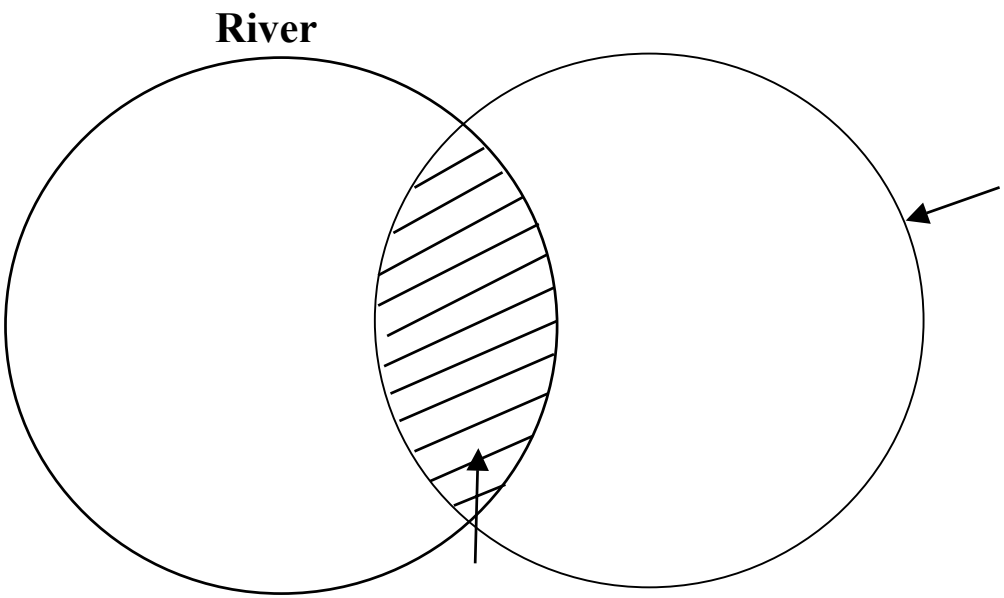
# Understanding intersectionOf

```
<?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="Flueve">
      <owl:intersectionOf rdf:parseType="Collection">
         <owl:Class rdf:about="#River"/>
         <owl:Restriction>
            <owl:onProperty rdf:resource="#emptiesInto"/>
            <owl:allValuesFrom rdf:resource="#Sea"/>
         </owl:Restriction>
      </owl:intersectionOf>
   </owl:Class>

   ...

</rdf:RDF>
```

This is read as: "The Flueve class is the intersection of the River class and an anonymous class that contains a property emptiesInto and all values are instances of Sea."

Here's an easier way to read this: "The Flueve class is a River that emptiesInto a Sea."

# Understanding intersectionOf




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

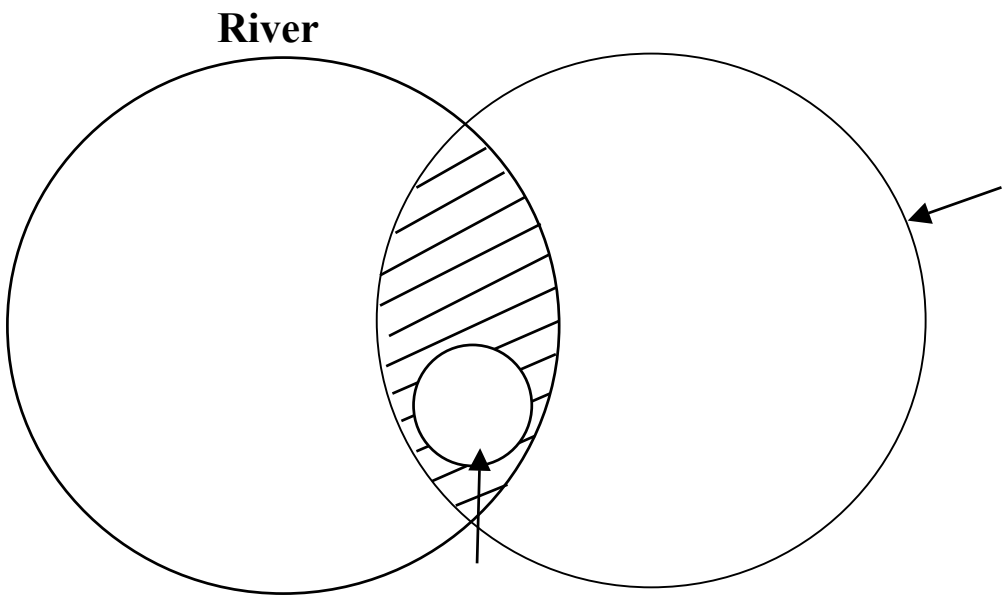
**Flueve** - a River that emptiesInto a Sea.

# Contrast with defining Flueve using 2 subClassOf statements

```
<?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="Flueve">
    <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>
```

**naturally-occurring.owl (snippet)**

# Multiple subClassOf = a subset of the intersection

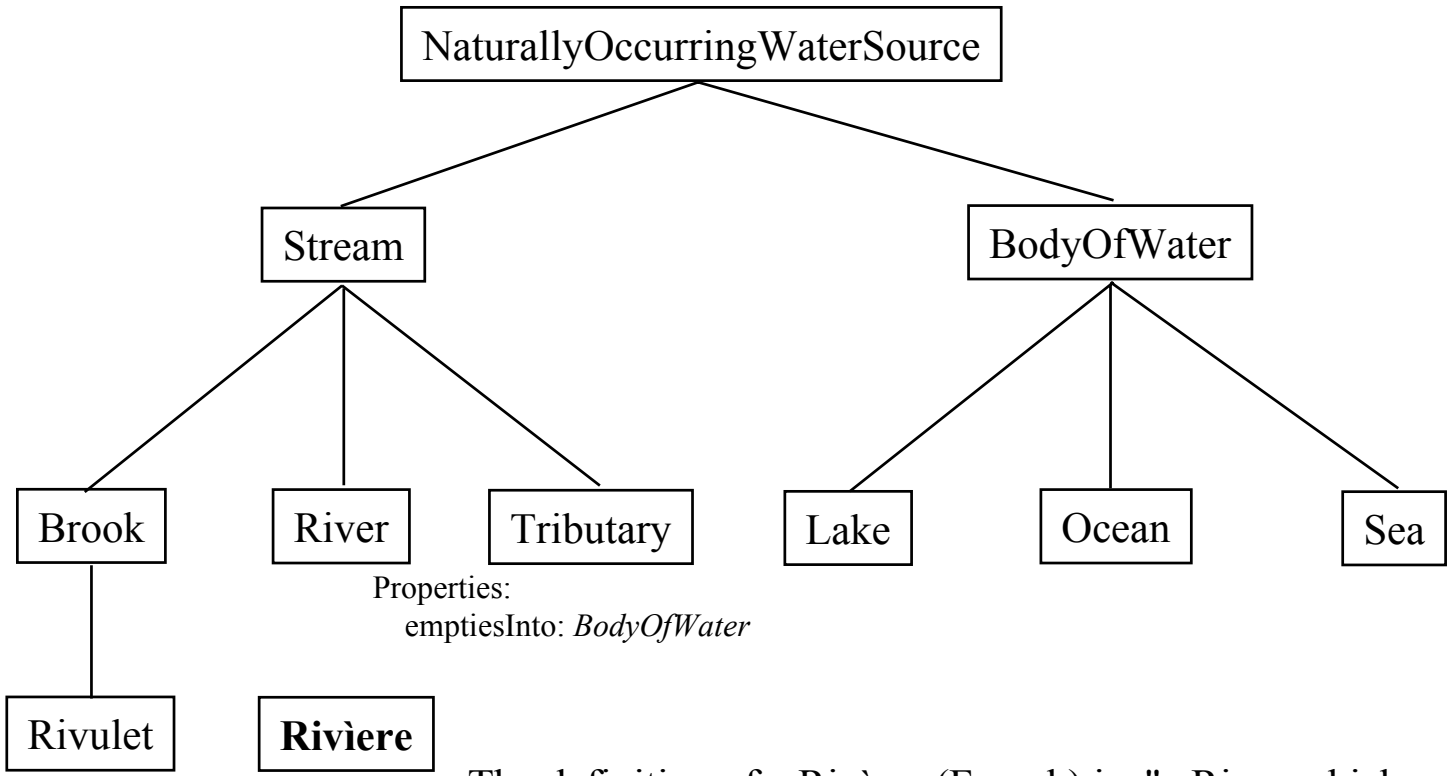



The members of this anonymous class are instances which have an emptiesInto property in which all values are instances of Sea.

**Flueve** - a Flueve is a River that emptiesInto a Sea. However, as this diagram shows, by using multiple subClassOf elements there may be Rivers which emptiesInto a Sea that are not Flueves.

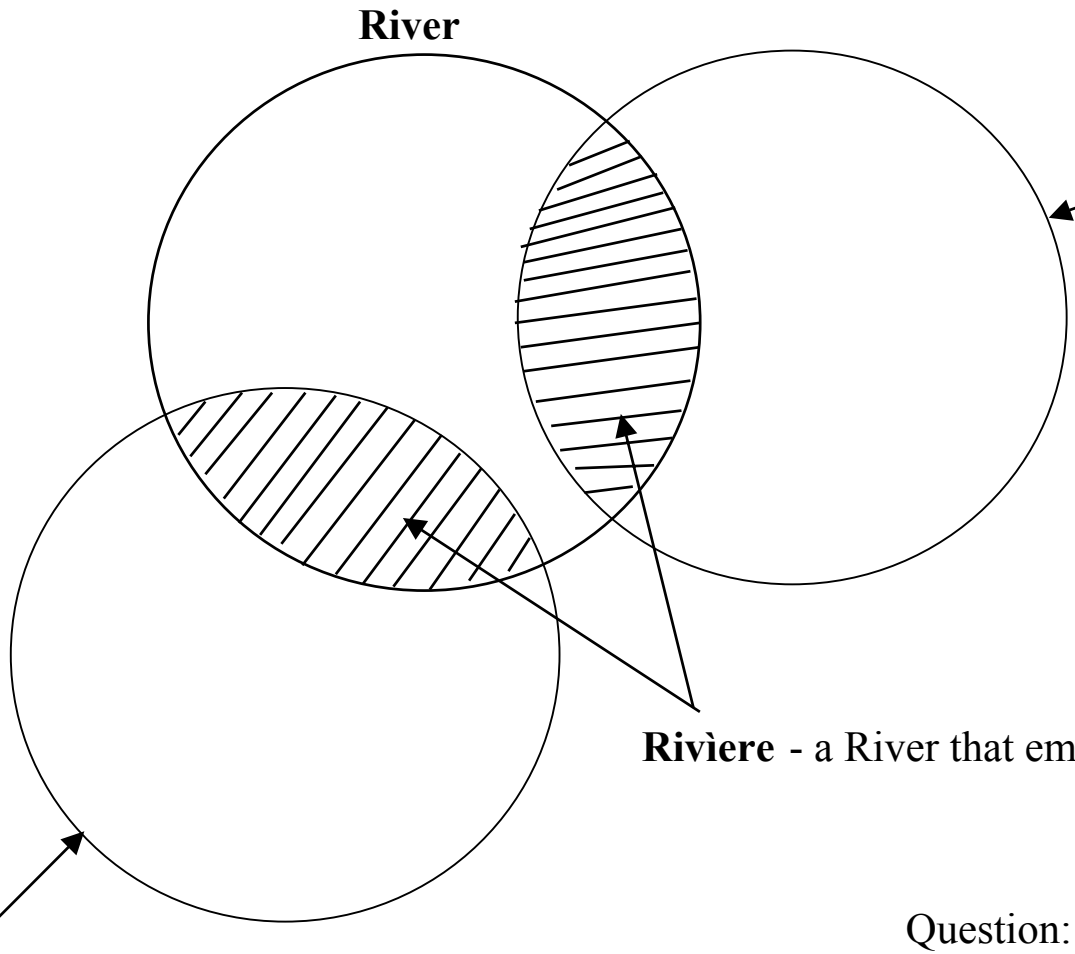
*The conjunction (AND) of two subClassOf statements is a subset of the intersection of the classes.*

# Defining a Rivìere class using the unionOf operator




The definition of a Rivìere (French) is: "a River which emptiesInto a Lake or another River". Thus, to define a Rivìere we will need to use both intersectionOf and unionOf ...

# A Rivìere is the intersection of River with the union of two classes




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

**Rivìere** - a River that emptiesInto a Sea or another River.

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

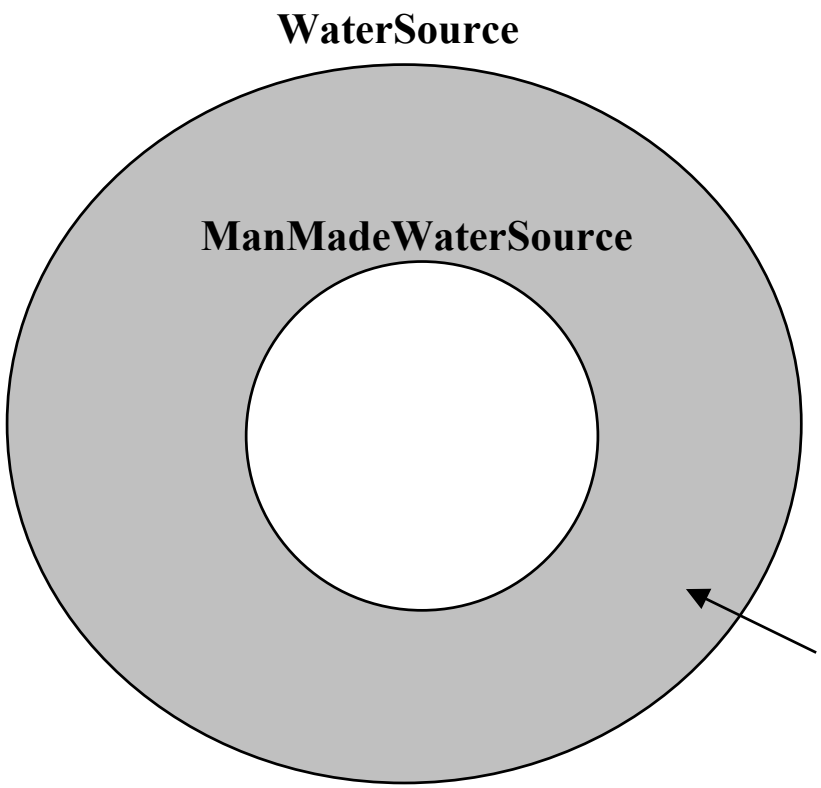
Question: do you understand why the two anonymous classes are disjoint?
Answer: because emptiesInto is a Functional Property!

# Defining Rivìere

```
<?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="Rivìere">
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#River"/>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Restriction>
            <owl:onProperty rdf:resource="#emptiesInto"/>
            <owl:allValuesFrom rdf:resource="#Sea"/>
          </owl:Restriction>
          <owl:Restriction>
            <owl:onProperty rdf:resource="#emptiesInto"/>
            <owl:allValuesFrom rdf:resource="#River"/>
          </owl:Restriction>
        </owl:unionOf>
      </owl:Class>
    </owl:intersectionOf>
  </owl:Class>

  ...

</rdf:RDF>
```

**naturally-occurring.owl (snippet)**

# Defining NaturallyOccurringWaterSource using complementOf




**NaturallyOccurringWaterSource**

- the intersection of WaterSource and the complementOf ManMadeWaterSource.

# Using complementOf

```
<?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="NaturallyOccurringWaterSource">
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#WaterSource"/>
      <owl:Class>
        <owl:complementOf rdf:resource="#ManMadeWaterSource"/>
      </owl:Class>
    </owl:intersectionOf>
  </owl:Class>

  ...

</rdf:RDF>
```

**naturally-occurring.owl (snippet)**

# Enumeration, equivalence, disjoint

- OWL gives you the ability to:
  - construct a class by enumerating its instances.
  - specify that a class is equivalent to another class.
  - specify that a class is disjoint from another class.

# Defining a class by enumerating its instances

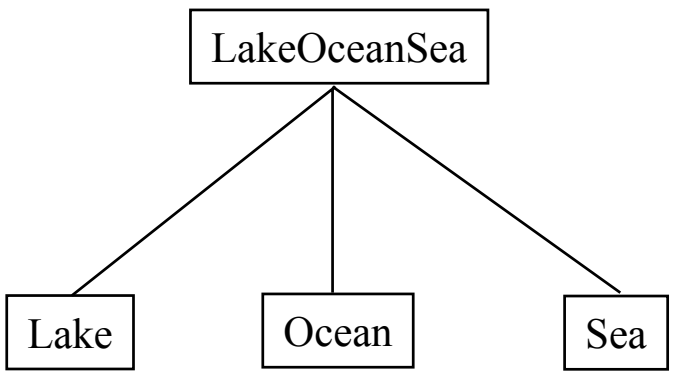
Here we are enumerating the Rivers which are protected by the Kyoto Treaty.

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

  <owl:Class rdf:ID="Kyoto-Protected-River">
    <rdfs:subClassOf rdf:resource="#River"/>
    <owl:oneOf rdf:parseType="Collection">
      <geo:River rdf:about="http://www.china.org/geography/rivers#Yangtze"/>
      <geo:River rdf:about="http://www.us.org/rivers#Mississippi"/>
      <geo:River rdf:about="http://www.africa.org/rivers#Nile"/>
      <geo:River rdf:about="http://www.s-america.org/rivers#Amazon"/>
      …
    </owl:oneOf>
  </owl:Class>

  ...

</rdf:RDF>
```

**naturally-occurring.owl (snippet)**

# Defining a class to be equivalent to another class

- owl:equivalentClass is used to state that a class is equivalent to another class.
- Example: suppose that another OWL document defines a class called LakeOceanSea as follows:




We would like to state that BodyOfWater is equivalent to LakeOceanSea.

# Defining BodyOfWater to be equivalent to LakeOceanSea

```
<?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"/>
    <owl:equivalentClass rdf:resource="http://www.other.org#LakeOceanSea"/>
  </owl:Class>

  ...

</rdf:RDF>
```

**naturally-occurring.owl (snippet)**

# Defining a class to be disjoint from another class

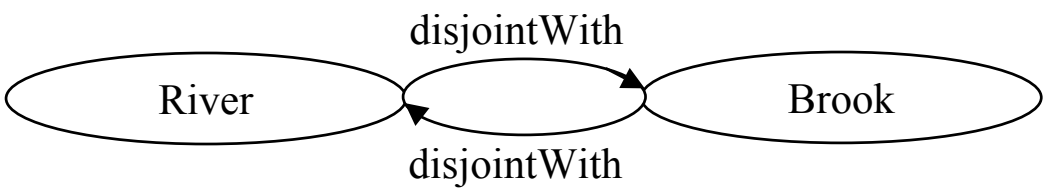
```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xml:base="http://www.geodesy.org/water/naturally-occurring">

  <owl:Class rdf:ID="River">
    <rdfs:subClassOf rdf:resource="#Stream"/>
    <owl:disjointWith rdf:resource="#Brook"/>
    <owl:disjointWith rdf:resource="#Rivulet"/>
    <owl:disjointWith rdf:resource="#Tributary"/>
  </owl:Class>

  ...

</rdf:RDF>
```

**naturally-occurring.owl (snippet)**

This definition of River indicates that a River instance cannot also be a Brook, Rivulet, or Tributary. Thus, for example, you cannot have an instance which defines the Yangtze as a Tributary.

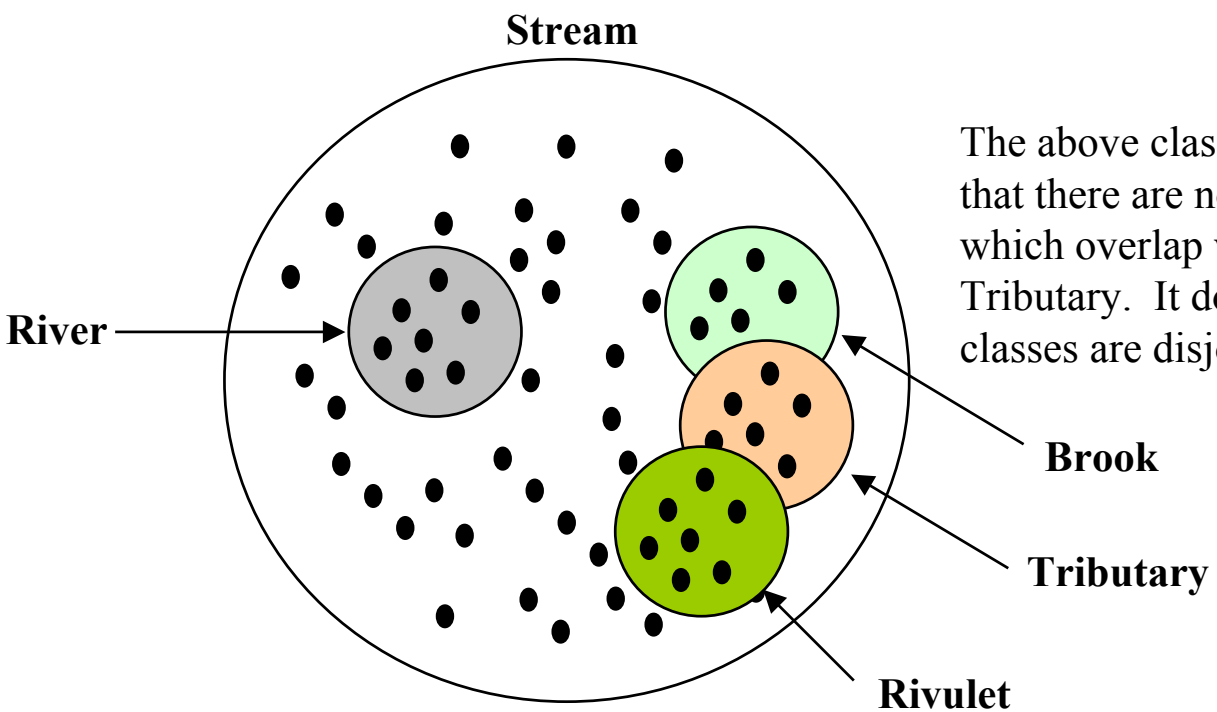
# Note: disjointWith is a SymmetricProperty!

- Example: if River is disjointWith Brook, then Brook is disjointWith River.



# River is (only) disjoint from the others

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



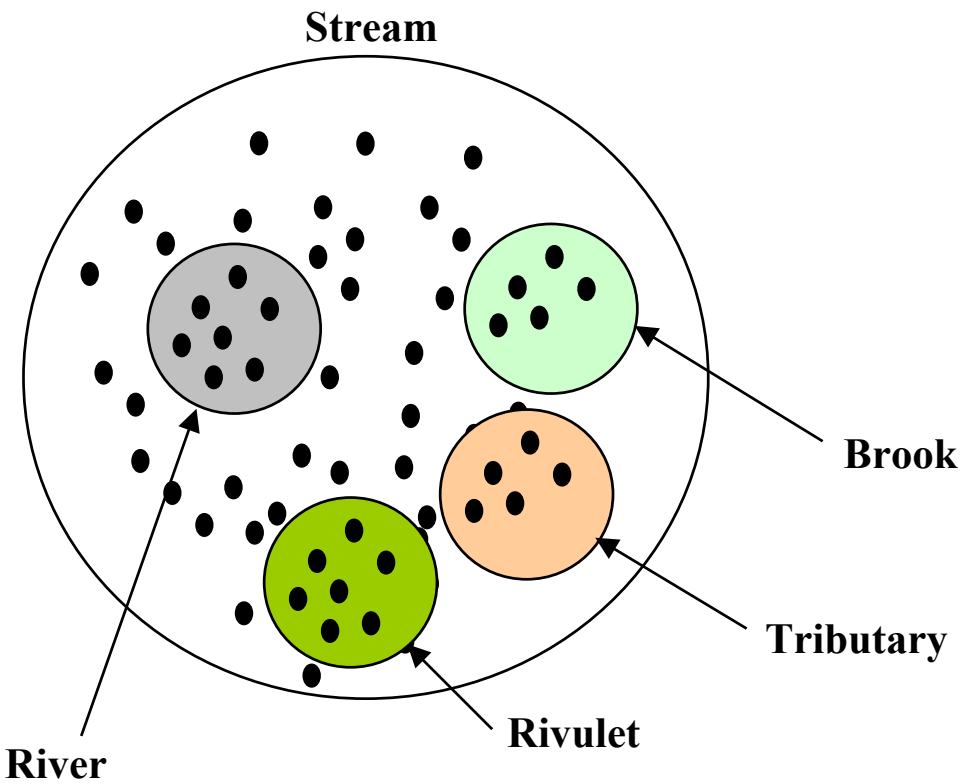

The above class definition only states that there are no instances of River which overlap with Brook, Rivulet, or Tributary. It does not state that all four classes are disjoint.

# Now we know that all are disjoint

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

```
<owl:Class rdf:ID="Brook">
        <rdfs:subClassOf rdf:resource="#Stream"/>
        <owl:disjointWith rdf:resource="#Rivulet"/>
        <owl:disjointWith rdf:resource="#Tributary"/>
</owl:Class>
```

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



# Summary of Class Properties

rdfs:Class

**Properties:**
subClassOf: *rdfs:Class*
domain: *rdfs:Class*
range: *rdfs:Class*

owl:Class

**Properties:**
intersectionOf: *rdf:List*
unionOf: *rdf:List*
complementOf: *owl:Class*
oneOf: *rdf:List*
equivalentClass: *owl:Class*
disjointWith: *owl:Class*

owl:Restriction

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

# OWL statements that you can incorporate into instances

# Indicating that two instances are the same

Consider these two instance documents:

```
<?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#">
    ...
</Sea>
```

```
<?xml version="1.0"?>
<Sea rdf:ID="S100-x-302"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns="http://www.geodesy.org/water/naturally-occurring#">
    ...
</Sea>
```

Are they referring to the same Sea? In fact, S100-x-302 is the catalog number for the East China Sea. So, these two instances do refer to the same Sea. It would be useful if we could state in an instance document that it is describing the same thing as another instance document. We use owl:sameIndividualAs to express this sameness ...

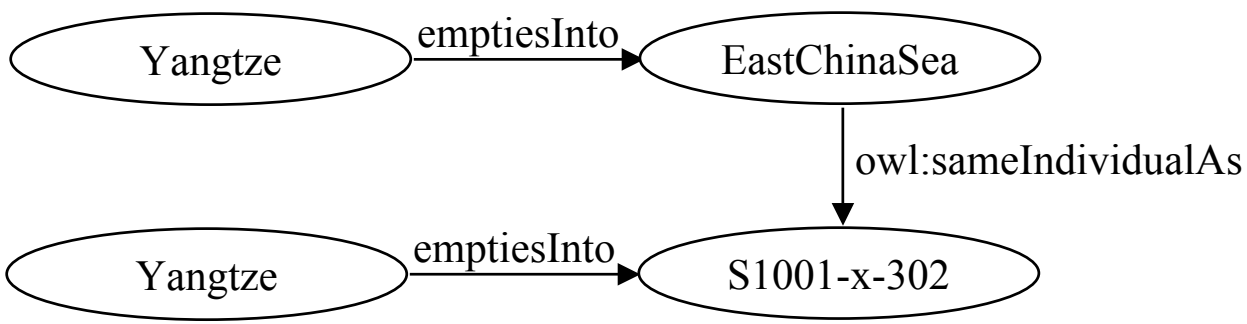
# owl:sameIndividualAs

```
<?xml version="1.0"?>
<Sea rdf:ID="EastChinaSea"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:owl="http://www.w3.org/2002/07/owl#"
        xmlns="http://www.geodesy.org/water/naturally-occurring#">
   <owl:sameIndividualAs rdf:resource="http://www.national-geographic.org#S1001-x-302"/>
    ...
</Sea>
```

We are clearly indicating that this instance is describing the same thing as the S100-x-302 instance.

# owl:FunctionalProperty and owl:sameIndividualAs can reinforce each other!

```
<?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"/>
   <emptiesInto rdf:resource="http://www.national-geographic.org#S1001-x-302"/>
</River>
```




By defining emptiesInto as a FunctionalProperty we assert that EastChinaSea and S100-x-302 must be the same. The owl:sameIndividualAs is reconfirming this!

# Indicating that two instances are different

Consider these two instance documents:

```
<?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#">
    ...
</Sea>
```

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

It may be useful to clearly state in an instance document that it is different from another instance. This is accomplished using owl:differentFrom.

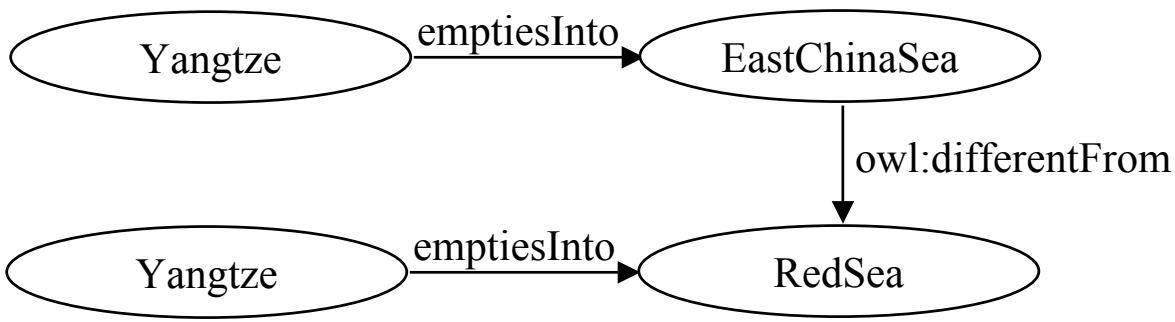
# owl:differentFrom

```
<?xml version="1.0"?>
<Sea rdf:ID="EastChinaSea"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:owl="http://www.w3.org/2002/07/owl#"
        xmlns="http://www.geodesy.org/water/naturally-occurring#">
   <owl:differentFrom rdf:resource="http://www.mediterranean.org#RedSea"/>
    ...
</Sea>
```

We are clearly indicating that this instance is different from the Red Sea instance.

# owl:FunctionalProperty combined with owl:differentFrom can expose contradictions!

```
<?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/[illegible]/naturally-occurring#">
   <emptiesInto rdf:resource="http://[illegible].org/geography#EastChinaSea"/>
   <emptiesInto rdf:resource="http://[illegible]terranean.org#RedSea"/>
</River>
```




By defining emptiesInto as a FunctionalProperty we assert that EastChinaSea and RedSea must be the same. But owl:differentFrom indicates that they are different! Thus, there is a contradiction. It indicates that the instance is in error.

# owl:AllDifferent

Using the owl:AllDifferent class we can indicate that a collection of instances are different:

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

  <owl:AllDifferent>
    <owl:distinctMembers rdf:parseType="Collection">
      <geo:Sea rdf:about="http://www.china.org/geography/rivers#EastChinaSea"/>
      <geo:Sea rdf:about="http://www.mediterranean.org#RedSea"/>
      <geo:Sea rdf:about="http://www.africa.org/rivers#ArabianSea"/>
      <geo:Sea rdf:about="http://www.philippines.org#PhilippineSea"/>
    </owl:distinctMembers>
  </owl:AllDifferent>

  ...

</rdf:RDF>
```

**naturally-occurring.owl (snippet)**

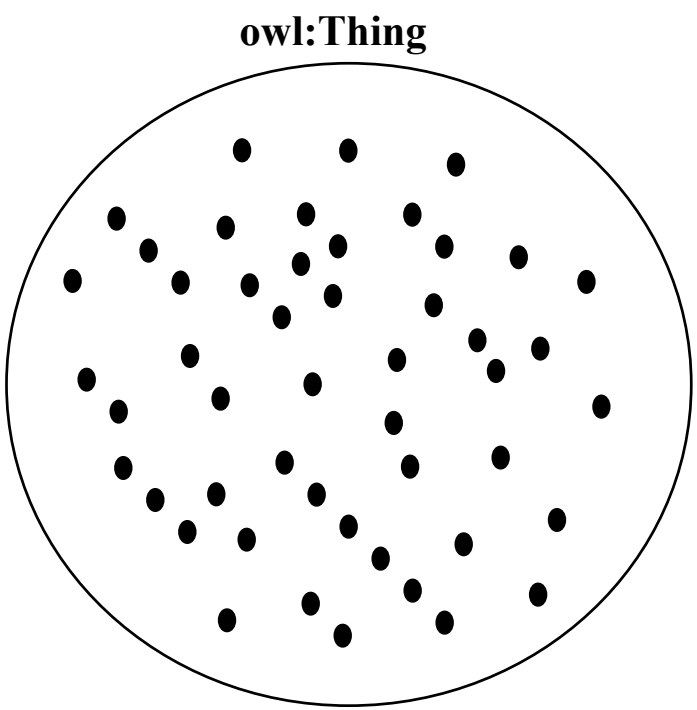
This indicates that the EastChinaSea, RedSea, ArabianSea, and PhilippineSea are all different!

# Summary of the different statements you can incorporate into instances

- In the preceding slides we saw the OWL statements that you can put into instance documents:
  - sameIndividualAs
  - differentFrom
- Question: what about AllDifferent? Answer: the owl:AllDifferent class is typically used in an ontology document, not an instance document (however, you can use it in an instance document if you like).

# owl:Thing

owl:Thing is a predefined OWL class.
**All** instances are members of owl:Thing.




Every instance in the universe is a member of owl:Thing!

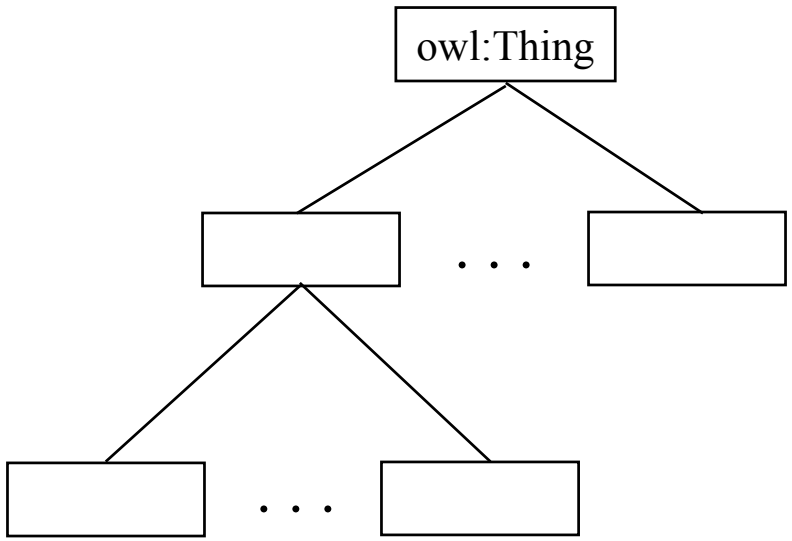
# Equivalent!

```
<?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#">
   <length>6300 kilometers</length>
   <emptiesInto rdf:resource="http://www.china.org/geography#EastChinaSea"/>
</River>
```

```
<?xml version="1.0"?>
<owl:Thing rdf:ID="Yangtze"
            xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:owl="http://www.w3.org/2002/07/owl#"
            xmlns="http://www.geodesy.org/water/naturally-occurring#">
   <rdf:type rdf:resource="http://www.geodesy.org/water/naturally-occurring#River"/>
   <length>6300 kilometers</length>
   <emptiesInto rdf:resource="http://www.china.org/geography#EastChinaSea"/>
</owl:Thing>
```

"Yangtze is a Thing.  Specifically, it is a River Thing."

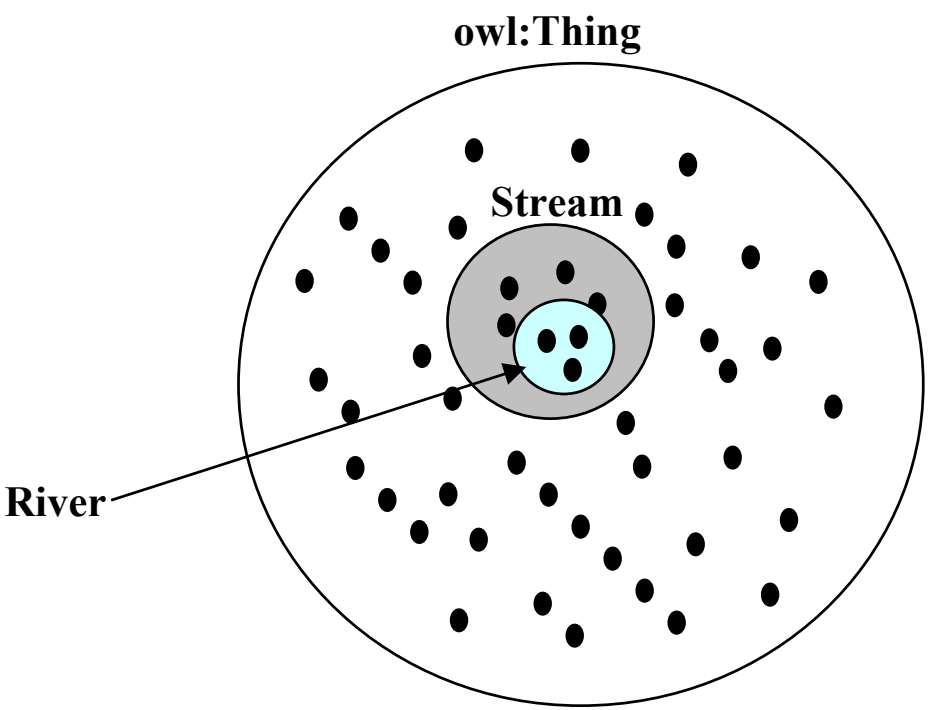
# The owl:Thing class is the root of all classes



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

Equivalent!

```
<owl:Class rdf:ID="River">
    <rdfs:subClassOf rdf:resource="#Stream"/>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl #Thing"/>
</owl:Class>
```




Do Lab3

# Importing other OWL documents

# Other OWL documents must be specified in-band

- With RDF Schema you can simply use another Schema. You don't have to "import" the Schema. Tools are expected to locate the Schema "out-of-band".
- With OWL you must explicitly import the OWL documents you will use.

# The Ontology Header

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

  <owl:Ontology rdf:about="http://www.geodesy.org/water/naturally-occurring">
    <owl:priorVersion rdf:resource="http://www.geodesy.org/water/naturally-occurring/july-02#"/>
    <owl:versionInfo>naturally-occurring.owl v 2.0</owl:versionInfo>
    <owl:imports rdf:resource="http://www.other.org/other.owl"/>
  </owl:Ontology>

  ...

</rdf:RDF>
```

All class, property, and instance definitions come **after** owl:Ontology

**naturally-occurring.owl (snippet)**

# owl:Ontology Properties

owl:Ontology

**Properties:**

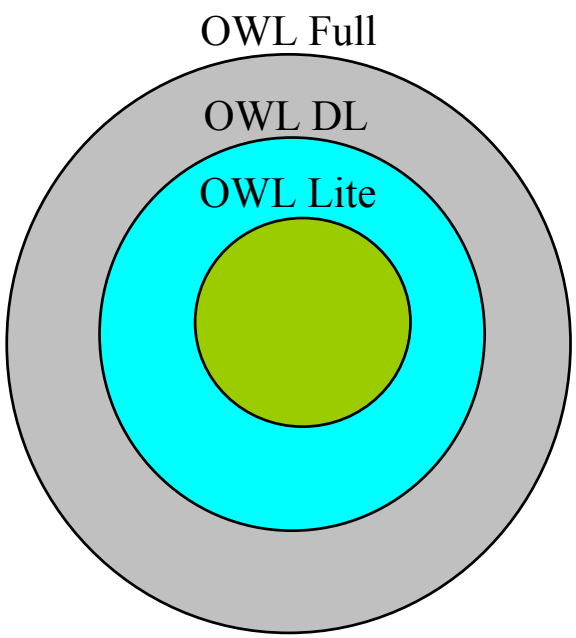
imports:
versionInfo:
priorVersion: *Ontology*
incompatibleWith: *Ontology*
backwardCompatibleWith: *Ontology*

Note: *Ontology* refers to the OWL document, e.g., naturally-occurring.owl.

# The Three Faces of OWL

# OWL Full, OWL DL, and OWL Lite

- Not everyone will need all of the capabilities that OWL provides. Thus, there are three versions of OWL:




DL = Description Logic

# Comparison

| OWL Full | OWL DL | OWL Lite |
|---|---|---|
| Everything that has been shown in this tutorial is available. Further, you can mix RDF Schema definitions with OWL definitions. | You cannot use owl:cardinality with TransitiveProperty.<br>A DL ontology cannot import an OWL Full ontology.<br>You cannot use a class as a member of another class, i.e., you cannot have metaclasses.<br>FunctionalProperty and InverseFunctionalProperty cannot be used with datatypes (they can only be used with ObjectProperty). | You cannot use owl:minCardinality or owl:maxCardinality.<br>The only allowed values for owl:cardinality is 0 and 1.<br>Cannot use owl:hasValue.<br>Cannot use owl:disjointWith.<br>Cannot use owl:oneOf.<br>Cannot use owl:complementOf.<br>Cannot use owl:unionOf. |

# Advantages/Disadvantages

- Full:
  - The advantage of the Full version of OWL is that you get the full power of the OWL language.
  - The disadvantage of the Full version of OWL is that it is difficult to build a Full tool. Also, the user of a Full-compliant tool may not get a quick and complete answer.
- DL/Lite:
  - The advantage of the DL or Lite version of OWL is that tools can be built more quickly and easily, and users can expect responses from such tools to come quicker and be more complete.
  - The disadvantage of the DL or Lite version of OWL is that you don't have access to the full power of the language.

# Related Documents

- The **OWL Guide** provides a very nice description of OWL, with many examples:
  - http://www.w3.org/TR/owl-guide/
- Here is the URL to the **OWL Reference** document:
  - http://www.w3.org/TR/owl-ref/
- For all other OWL documents, and information on the Semantic Web see:
  - http://www.w3.org/2001/sw/

# Examples

# The Robber and the Speeder (version 2)

- An expanded version of the Robber and the Speeder example is shown on the following slides. This version was created by Ian Davis. Thanks Ian!

# Robber drops gun while fleeing!

First of all a robbery takes place. The robber drops his gun while fleeing. This report is filed by the investigating officers:

```
<RobberyEvent>
   <date>...</date>
   <description>...</description>
   <evidence>
      <Gun>
         <serial>ABCD</serial>
      </Gun>
   </evidence>
   <robber>
      <Person /> <!-- an unknown person -->
   </robber>
</RobberyEvent>
```

# Speeder stopped

Subsequently a car is pulled over for speeding. The traffic officer files this report electronically while issuing a ticket:

```
<SpeedingOffence>
   <date>...</date>
   <description>...</description>
   <speeder>
      <Person>
         <name>Fred Blogs</name>
         <driversLicenseNumber>ZXYZXY</driversLicenseNumber>
      </Person>
   </speeder>
</SpeedingOffence>
```

# The speeder owns a gun with the same serial number as the robbery gun!

At police headquarters (HQ), a computer analyzes each report as it is filed. The computer uses the driver's license information to look up any other records it has about Fred Blogs (the speeder) and discovers this gun license:

```
<GunLicense>
   <registeredGun>
      <Gun>
         <serial>ABCD</serial>
      </Gun>
   </registeredGun>
   <holder>
      <Person>
         <driversLicenseNumber>ZXYZXY</driversLicenseNumber>
      </Person>
   </holder>
</GunLicense>
```

# Case Solved?

- Not yet! These questions must be answered before the speeder can be arrested as the robbery suspect:
  - Can multiple guns have the same serial number?
    - If so, then just because Fred Blogs owns a gun with the same serial number as the robbery gun does not mean it was his gun that was used in the robbery.
  - Can multiple people have the same driver's license number?
    - If so, then the gun license information may be for someone else.
  - Can a gun be registered in multiple gun licenses?
    - If so, then the other gun licenses may show the holder of the gun to be someone other than Fred Blogs.
  - Can a gun license have multiple holders of a registered gun?
    - If so, then there may be another gun license document (not available at the police HQ) which shows the same registered gun but with a different holder.
- The OWL Ontology (Police.owl) provides the information needed to answer these questions!

# Can multiple guns have the same serial number?

This OWL rule (in Police.owl) tells the computer at police HQ that *each gun is uniquely identified by its serial number*:

```
<owl:InverseFunctionalProperty rdf:ID="serial">
    <rdfs:domain rdf:resource="Gun"/>
    <rdfs:range  rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</owl:InverseFunctionalProperty>
```

# Can multiple people have the same driver's license number?

The following OWL rule tells the computer that *a driver's license number is unique to a Person*:

```
<owl:InverseFunctionalProperty rdf:ID="driversLicenseNumber">
    <rdfs:domain rdf:resource="Person"/>
    <rdfs:range  rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</owl:InverseFunctionalProperty>
```

# Can a gun be registered in multiple gun licenses?

The next OWL rule tells the computer that the registeredGun property uniquely identifies a GunLicense, i.e., *each gun is associated with only a single GunLicense*:

```
<owl:InverseFunctionalProperty rdf:ID="registeredGun">
    <rdfs:domain rdf:resource="GunLicense"/>
    <rdfs:range  rdf:resource="Gun"/>
</owl:InverseFunctionalProperty>
```

# Can a gun license have multiple holders of a registered gun?

The police computer uses the following OWL rule to determine that the gun on the license is the same gun used in the robbery. This final rule seals the speeder's fate. It tells the computer that *each GunLicense applies to only one gun and one person*. So, there is no doubt that the speeder is the person who owns the gun:

```
<owl:Class rdf:ID="GunLicense">
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
            <owl:onProperty rdf:resource="#registeredGun"/>
            <owl:cardinality>1</owl:cardinality>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#holder"/>
            <owl:cardinality>1</owl:cardinality>
        </owl:Restriction>
    </owl:intersectionOf>
</owl:Class>
```

# Summary of information provided by the Police ontology

**A gun license registers one gun to one person.**

**A gun can be registered in only one gun license.**

**Only one gun can have this serial number.**

**Only one person can have this driver's license number.**

```
<GunLicense>
   <registeredGun>
      <Gun>
         <serial>ABCD</serial>
      </Gun>
   </registeredGun>
   <holder>
      <Person>
         <driversLicenseNumber>ZXYZXY</driversLicenseNumber>
      </Person>
   </holder>
</GunLicense>
```

*We now have overwhelming evidence that the speeder is the robber!*

# Notes

The example showed that a driver's license number applies to only one person:

```
<owl:InverseFunctionalProperty rdf:ID="driversLicenseNumber">
    <rdfs:domain rdf:resource="Person"/>
    <rdfs:range  rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</owl:InverseFunctionalProperty>
```


"A driver's license number applies to only one person."

We can make an even stronger statement, because it's also true that a person has only one driver's license number:


"A driver's license number applies to only one person, and a person has only one driver's license number."

Continued →

# Notes (cont.)

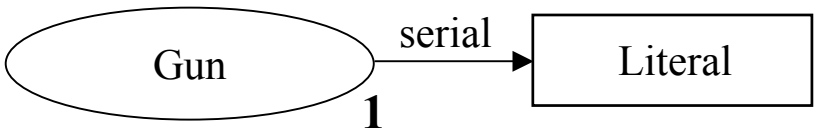
Thus, driversLicenseNumber is also a functional property:

```
<owl:DatatypeProperty rdf:ID="driversLicenseNumber">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
    <rdfs:domain rdf:resource="Person"/>
    <rdfs:range  rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</owl:DatatypeProperty>
```

# Notes (cont.)

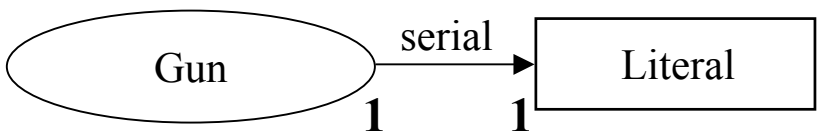
The example also showed that a serial number applies to only one gun:

```
<owl:InverseFunctionalProperty rdf:ID="serial">
    <rdfs:domain rdf:resource="Gun"/>
    <rdfs:range  rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</owl:InverseFunctionalProperty>
```




"A serial number applies to only one gun."

We can make an even stronger statement, because it's also true that a gun has only one serial number:




"A serial number applies to only one gun, and a gun has only one serial number."

Continued →

# Notes (cont.)

Thus, serial is also a functional property:

```
<owl:DatatypeProperty rdf:ID="serial">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
    <rdfs:domain rdf:resource="Gun"/>
    <rdfs:range  rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</owl:DatatypeProperty>
```

# Invitation

- Do you have a simple example which demonstrates various OWL capabilities? If so, we invite you to send it to Roger Costello at costello@mitre.org. We will incorporate your example (with credits) into this tutorial (time permitting, of course). Thanks!

# FAQ

# Confused about the difference between a Functional property and an Inverse Functional property

Consider the birthdate property in this document:

```
<Person rdf:ID="JohnDoe">
    <birthdate>March 24, 1971</birthdate>
</Person>
```

A Person has a single birthdate. Therefore, birthdate is a Functional Property.

Question: Is birthdate an Inverse Functional Property?
Answer: No. If birthdate was an Inverse Functional property then only one person could have a birthdate. There are many people that have the same birthdate. Thus, birthdate is not an Inverse Functional property.

Continued →

# Example of an Inverse Functional Property

Consider the email property in this document:

```
<Person rdf:ID="SallyJane">
    <email>sally-jane@yahoo.com</email>
</Person>
```

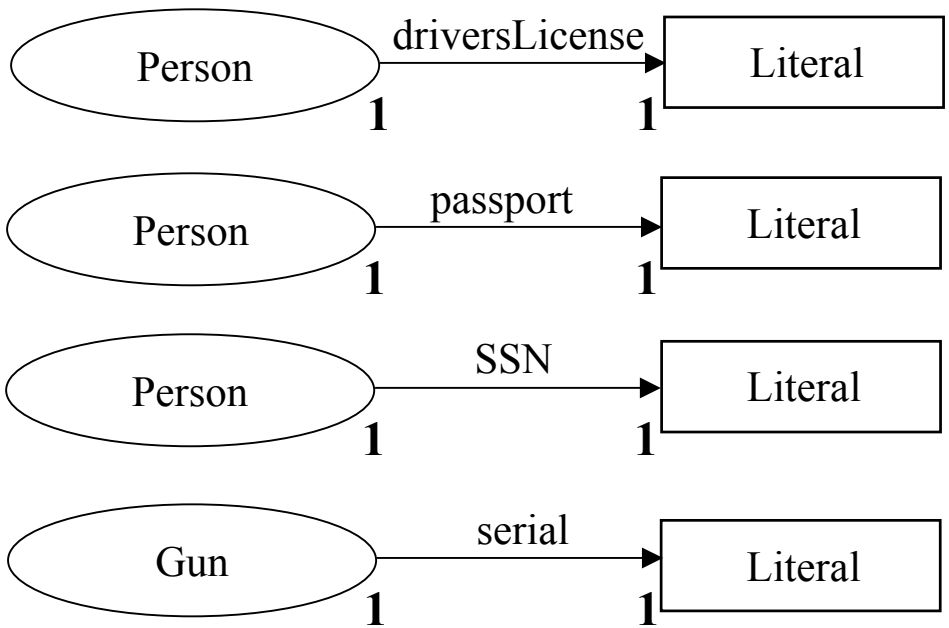
An email address applies to only one person. Therefore, email is an Inverse Functional Property.

Question: Is email a Functional Property?
Answer: No. If email was a Functional Property then a person could have only one email. Many people have multiple email addresses. Thus, email is not a Functional Property.

Continued →

# Examples of properties that are both Functional and Inverse Functional

Most ID-like properties are both Functional and Inverse Functional, e.g., driver's license, passport number, SSN, serial number, etc.



# Can an OWL Ontology also contain instance data?

In general, it is best to keep instance data separate from the ontology. Sometimes, however, mingling instance data with an ontology may be unavoidable. For example, suppose that you wish to use owl:AllDifferent to indicate that Mary, David, and Roger are all different:

```
<owl:AllDifferent>
    <owl:distinctMembers rdf:parseType="Collection">
        <Person rdf:about="#Mary"/>
        <Person rdf:about="#David"/>
        <Person rdf:about="#Roger"/>
    </owl:distinctMembers>
</owl:AllDifferent>
```

You might wish to provide, in the ontology, a "barebones" definition of the Mary instance, the David instance, and the Roger instance:

```
<Person rdf:ID="Mary"/>
<Person rdf:ID="David"/>
<Person rdf:ID="Roger"/>
```

Continued →

# Can an ontology also contain instance data? (cont.)

Now, instance documents extend the ontologies' barebones instance definitions:

```
<Person rdf:about="http://www.person.org#Roger">
    <hometown>Boston, MA</hometown>
    <workplace>The MITRE Corp.</workplace>
</Person>
```

# Difference between a Class with a property that has a maxCardinality=1 versus a Functional Property?

Both forms are equivalent! Let's take an example. Below is shown a Gun Class which is defined to have at most one serial number:

Version 1

```
<owl:Class rdf:ID="Gun">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#serial"/>
            <owl:maxCardinality>1</owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>

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

Continued →

# Defining a Class which has a Functional Property is equivalent!

The below serial property is defined to be a Functional Property, and is to be used with a Gun instance. Thus, the Gun Class has at most one serial number. The two forms are equivalent!

Version 2

```
<owl:Class rdf:ID="Gun"/>

<owl:FunctionalProperty rdf:ID="serial">
        <rdfs:domain rdf:resource="Gun" />
        <rdfs:range  rdf:resource="&rdfs;#Literal"/>
</owl:FunctionalProperty>
```

# Difference between a Class with multiple subclasses, each having a property that has a maxCardinality=1 versus multiple Functional Properties?

Both forms are equivalent! Let's take an example. Below is shown a GunLicense Class which is defined to have at most one registeredGun and at most one holder:

Version 1

```
<owl:Class rdf:ID="GunLicense">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#registeredGun"/>
            <owl:maxCardinality>1</owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#holder"/>
            <owl:maxCardinality>1</owl:maxCardinality>
        </owl:Restriction>
    <rdfs:subClassOf>
</owl:Class

<owl:ObjectProperty rdf:ID="registeredGun">
      <rdfs:domain rdf:resource="#GunLicense"/>
      <rdfs:range  rdf:resource="#Gun"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="holder">
      <rdfs:domain rdf:resource="#GunLicense"/>
      <rdfs:range  rdf:resource="#Person"/>
</owl:ObjectProperty>
```

Continued →

# Defining a Class which has multiple Functional Properties is equivalent!

The below registeredGun property and holder property are defined to be Functional Properties, and are to be used with a GunLicense instance. So, the GunLicense Class has at most one registeredGun and at most one holder. Thus, the two forms are equivalent!

Version 2

```
<owl:Class rdf:ID="GunLicense"/>

<owl:FunctionalProperty rdf:ID="registeredGun">
    <rdfs:domain rdf:resource="#GunLicense"/>
    <rdfs:range  rdf:resource="#Gun"/>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="holder">
    <rdfs:domain rdf:resource="#GunLicense"/>
    <rdfs:range  rdf:resource="#Person"/>
</owl:FunctionalProperty>
```