


# OWL Modeling with RDF

**COURSE MATERIALS FOR CU-003**
**http://www.cerebra.com/university**

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# OWL-DL Modeling with RDF

## Module One: Introduction

CU-003: OWL Modeling with RDF. © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Welcome to the Cerebra University CU-003 OWL-DL Modeling with RDF course, a multi-day hands-on workshop targeted to training software data modelers the essentials of using RDF and OWL in enterprise software systems. This course takes a proactive approach to learning, requiring the student to follow along, perform self-directed exercises and engage a professional instructor discussion about how best to leverage these powerful languages.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# Introductions

- Your name
- Company affiliation
- Title, function and job responsibility
  - Information, data modeling background
  - IT systems experience
  - Collaborative commerce experience
  - Sales, business development experience
- Reasons for enrolling in this course
- Expectations for this course

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# About Cerebra University

- Cerebra University is a joint collaboration between Cerebra and Semantic Arts focused on providing best-in-class education about emerging standards and semantic technology solutions.




- A consulting firm specializing in reducing complexity for Enterprise Information Systems.
- Key areas of expertise include SOA and Semantic architectures.




- A software vendor offering tools and technology for improving the efficiency of information management.
- Products include Cerebra Server, Cerebra Repository, Cerebra Construct and Workbench.

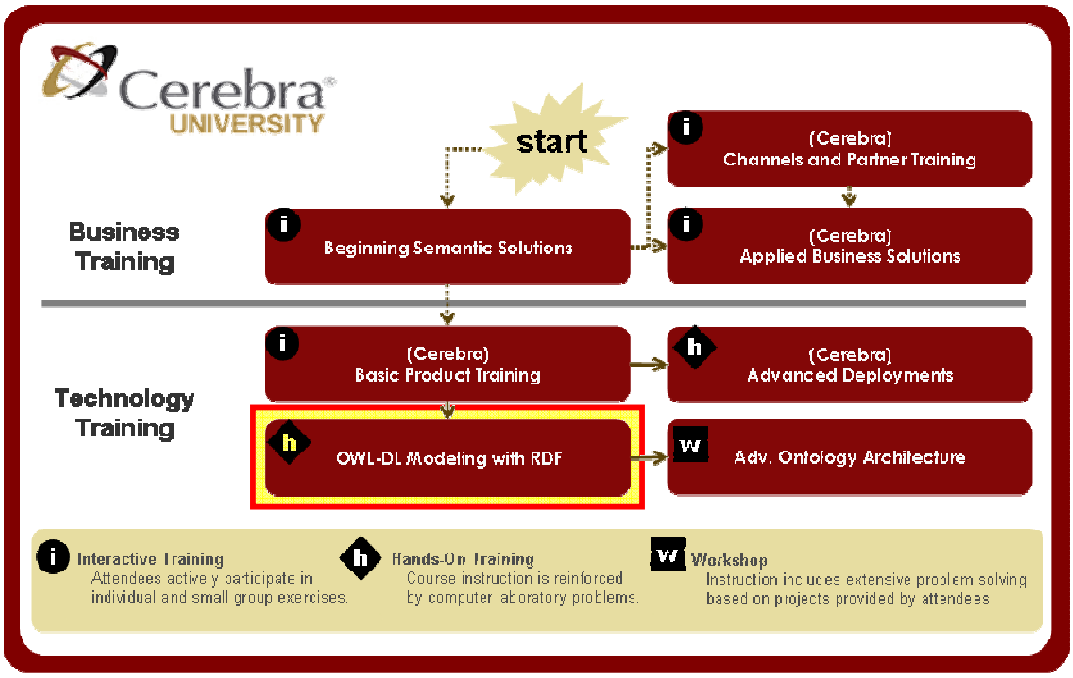
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# Training Programs




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# Agenda

| | Monday, Feb 27 | Tuesday, Feb 28 | Wednesday, Mar 01 |
|---|---|---|---|
| 8 am | | | |
| 9 00 | **Module 1:** *Intro to OWL* | **Module 3:** *Equivalence, Disjointness, and Subsumption* | **Module 5:** *Complex Classes* |
| 10 00 | | | |
| 11 00 | **Module 2:** *Individuals, Properties, and Classes* | **Module 3:** *Equivalence, Disjointness, and Subsumption* | **Module 5:** *Complex Classes* |
| 12 pm | | | |
| 1 00 | **Module 2:** *Individuals, Properties, and Classes* | **Module 4:** *More Properties* | **Module 6:** *How OWL is Different* |
| 2 00 | | | |
| 3 00 | **Module 2:** *Individuals, Properties, and Classes* | **Module 4:** *More Properties* | **Module 7:** *The Complement Module*<br>**Module 8:** *Wrap up* |
| 4 00 | | | |

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# What We Will Cover

- OWL and Description Logics
- Getting started with Cerebra Construct and Cerebra Server
- Hands-on using most of the capabilities of OWL/DL and the Cerebra XQuery Language

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# What We Won't Cover

- Advanced design elements for Ontology modeling
- In-depth, product-specific features of any particular tool
- Protégé, Swoop, etc.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# At the end of this class you should be able to:

- Capture an existing ontological design in Construct
- Build standard, complex and restriction classes in Construct
- Submit an ontology to the Cerebra Server
- Make reasonable queries against that ontology

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# At the end of this class you should understand:

- The difference between a satisfiable and provable query
- What the "open-world" assumption means
- The difference between declaring and deriving membership in a class
- How Description Logic modeling differs from Object-Oriented or E/R Modeling
- How OWL is similar to and different from business rules engines.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# Our Approach

- We will show you how to do something hands-on, while you follow along;
- Then explain what it is we've done;
- Then allow you to do it on your own.
- We will often intentionally cause you to create syntactical or logical modeling errors in order for you to see how to resolve them.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# Course Agenda

- Brief Intro: What are semantics, the Semantic Web, OWL and RDF?
- Lab 1: Building an Ontology – Multiple exercises covering individuals, classes, properties, semantic queries, and mapping to a data base.
- Lab 2: Digging Deeper – Properties, Domains and Ranges, Special Property Types, Equivalence, Disjointness and Subsumption.

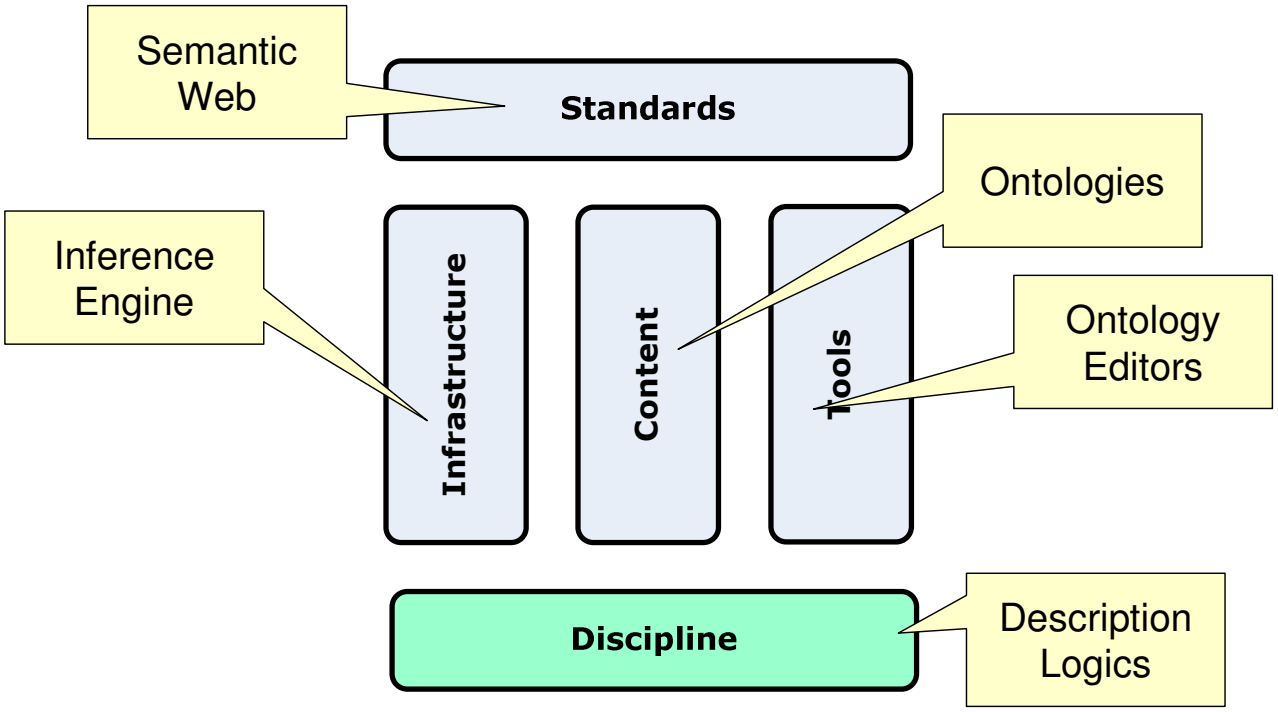
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# Semantic Technology



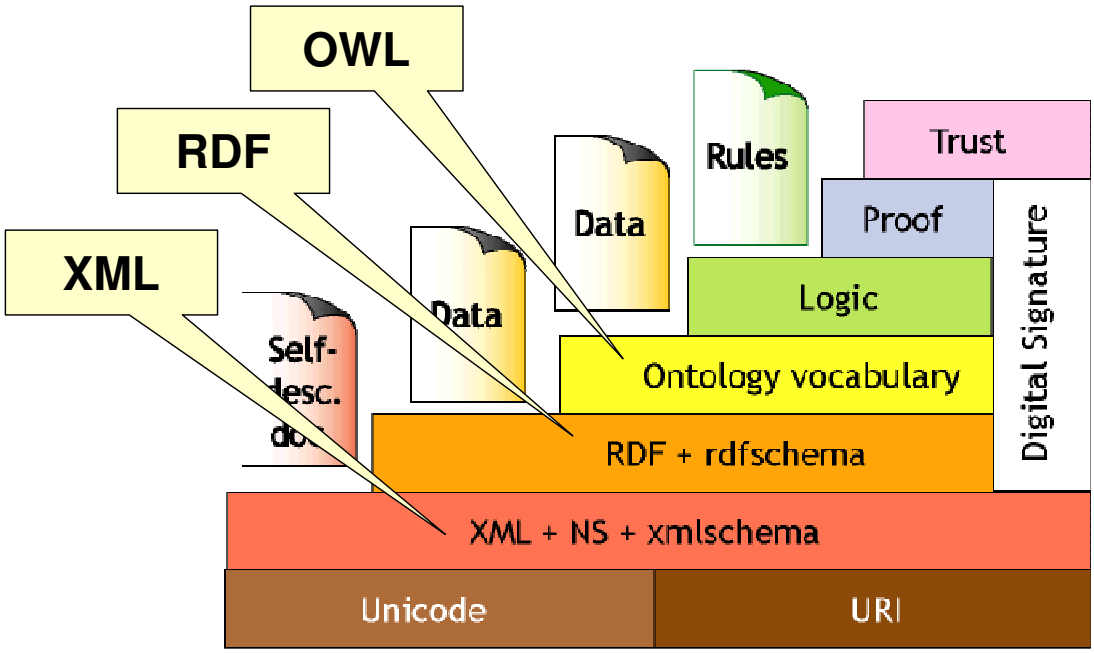

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# W3C Semantic Web Stack




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# OWL/RDF and XML

- XML is a universally accepted way to serialize structured data.
- While some journalists touted it as adding semantics to the web, the reality is different.
- XML tags are more expressive than HTML tags.
- But still rely entirely on humans to interpret.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# Namespaces

- A namespace is a domain that serves to guarantee the uniqueness of identifiers.
- You could think of your area code as a name space for local phone numbers.
- On the Internet, namespaces are identified by URIs (Universal Resource Identifier).

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# URIs

- Universal Resource Identifiers look a lot like URLs.
- The chief difference is a URL is meant to be a page that you could actually address and go to on the Web.
- A URI may be, but need not be, addressable. It merely needs to provide a means for establishing uniqueness of names.
- Note that in most cases you need not "go to" the identified URI to make use of the name space.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

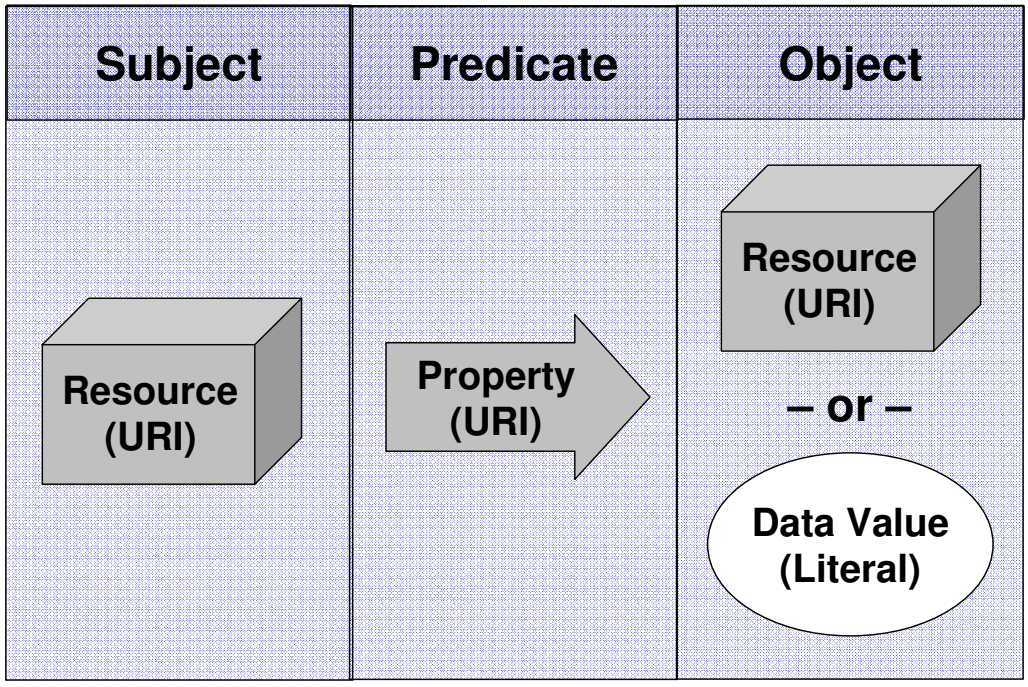
# RDF

- The Resource Description Framework is a standard way to make assertions.
- The assertions are in the form: subject → predicate → object.
- Also called a "triple."

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

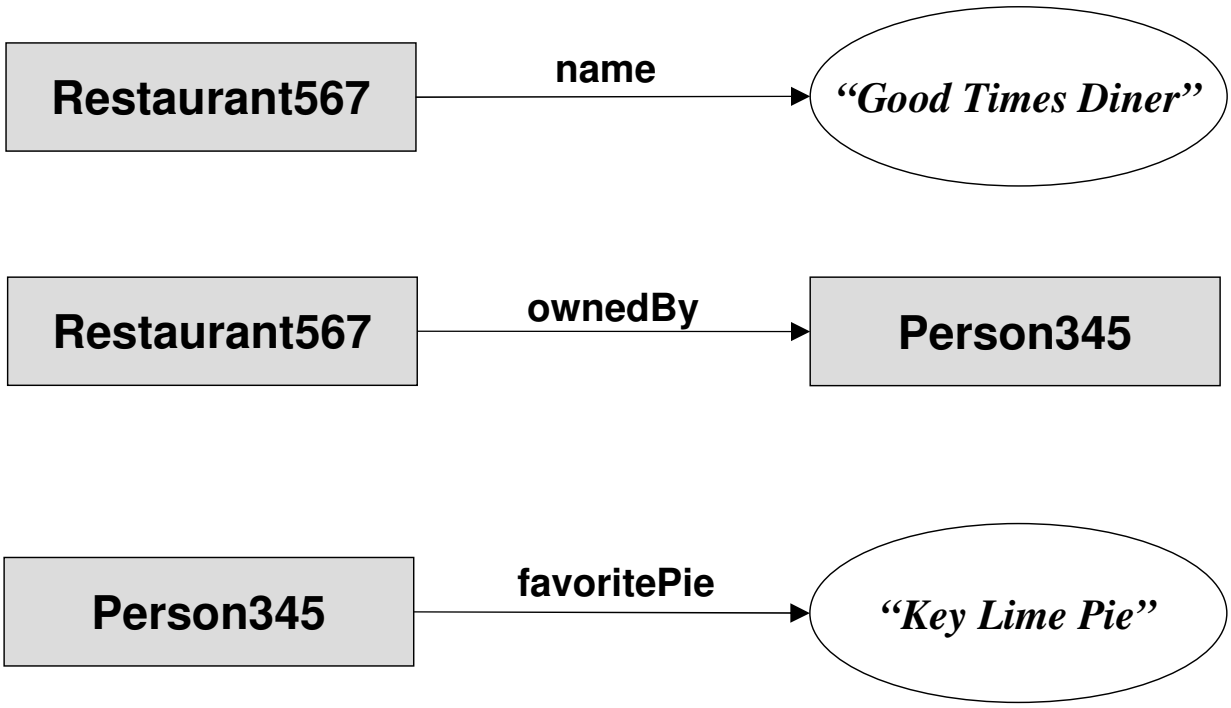
# An RDF Triple




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# RDF Examples



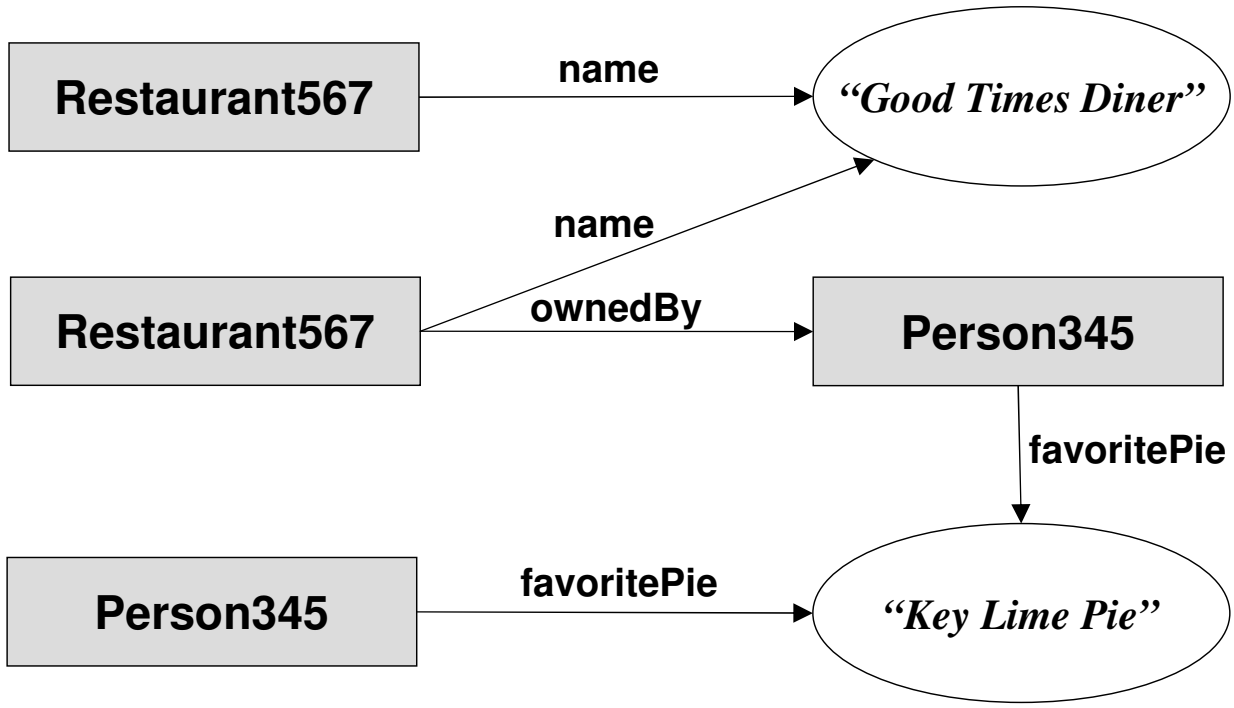

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# RDF Node Folding




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# RDF and XML

- RDF can, and usually is, serialized as XML.
- But RDF isn't dependent on XML, nor is it a dialog of XML.

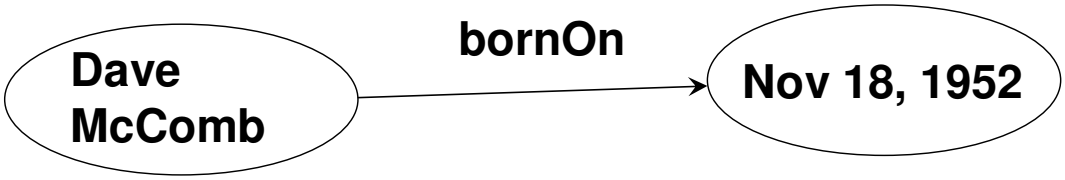
```
<rdf:Description rdf:ID="Person345">
    <favoritePie>Key Lime Pie</favoritePie>
</rdf:Description>
<rdf:Description rdf:about="#Restaurant567">
    <name>Good Times Diner</name>
    <ownedBy rdf:resource="#Person345"/>
</rdf:Description>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# RDF Assertions

- RDF is primarily about making concrete assertions.




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

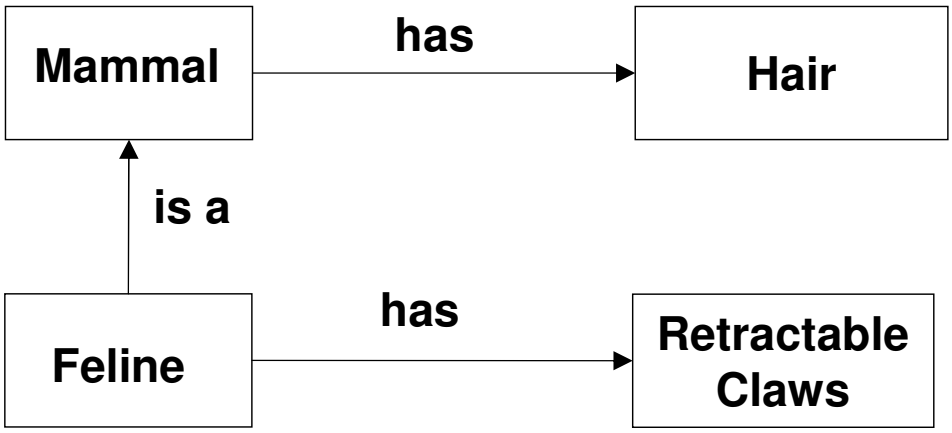
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# OWL

- OWL adds a layer on top of RDF that allows us to make statements about "classes" of items.




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# OWL Varieties

- There are actually four layered standards in play here:
  - RDF/S
  - OWL Lite
  - OWL/DL
  - OWL/Full
- For most of this course we will focus on OWL/DL.
- When it becomes important we will distinguish among and describe these other varieties.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# OWL/DL

- The "DL" stands for Description Logics.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# Description Logics

- A branch of mathematics and logic
- Has been in use for 30+ years
- There have been various implementations of Description Logic Languages over the years.
- But OWL/DL is the first widely adopted standard in this area.
- Henceforth we will use OWL/DL and Description Logics interchangeably, except in the rare cases where there may be a difference in coverage of the standard.

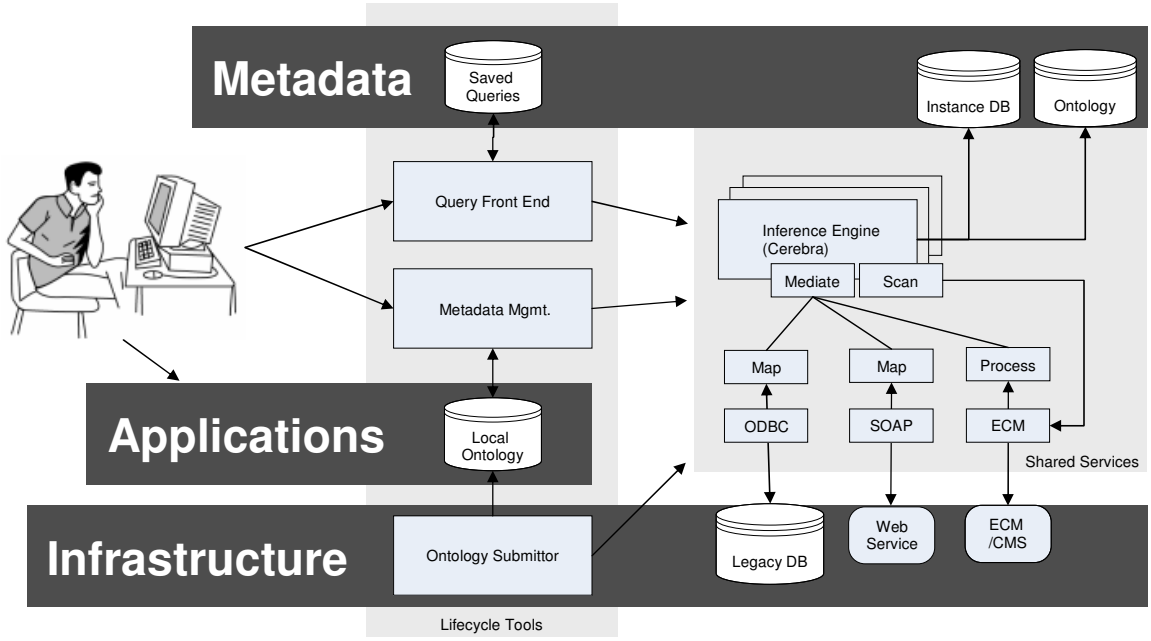
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# Architecturally




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# Module Review

- Instructor led review session
- Pop-quiz is optional!

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# OWL-DL Modeling with RDF

## Module Two:
## Individuals, Properties and Classes

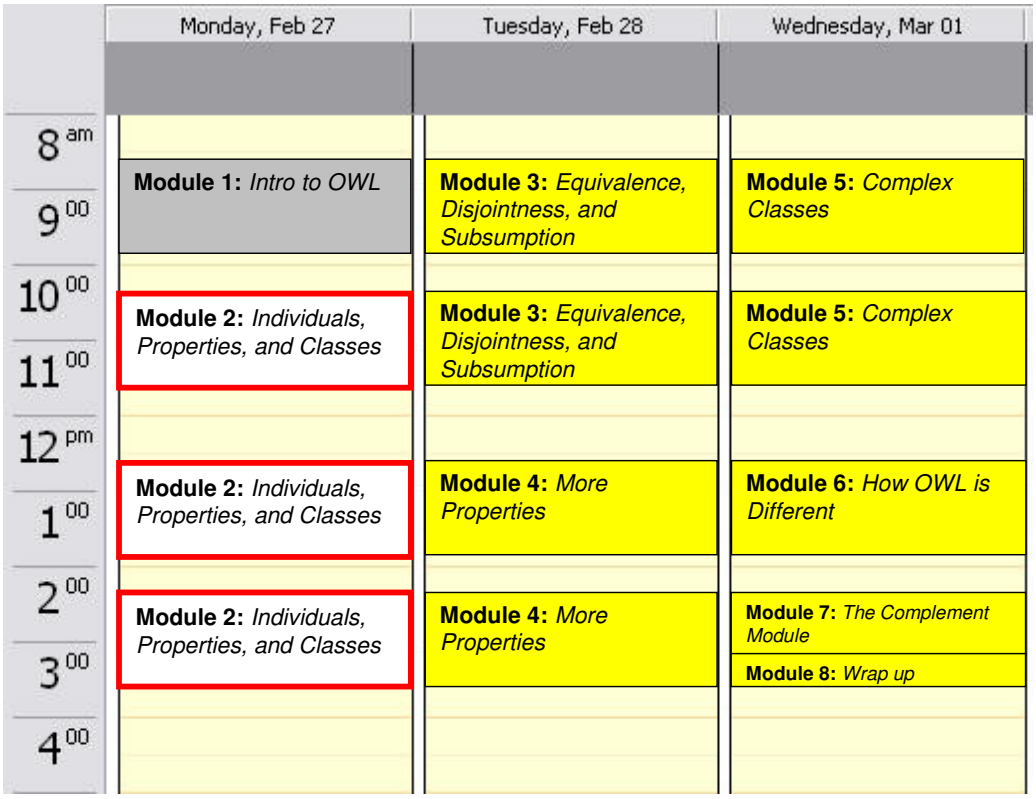
CU-003: OWL Modeling with RDF. © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Agenda




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Construct: Ontology modeling tool

- Cerebra Construct is an ontology modeling tool that integrates into Microsoft Visio.
- It can be used on its own or as a client for Cerebra Server.
- It exposes Cerebra extensions to OWL-DL such as DB mapping.
- It supports Import/Export to OWL-DL.

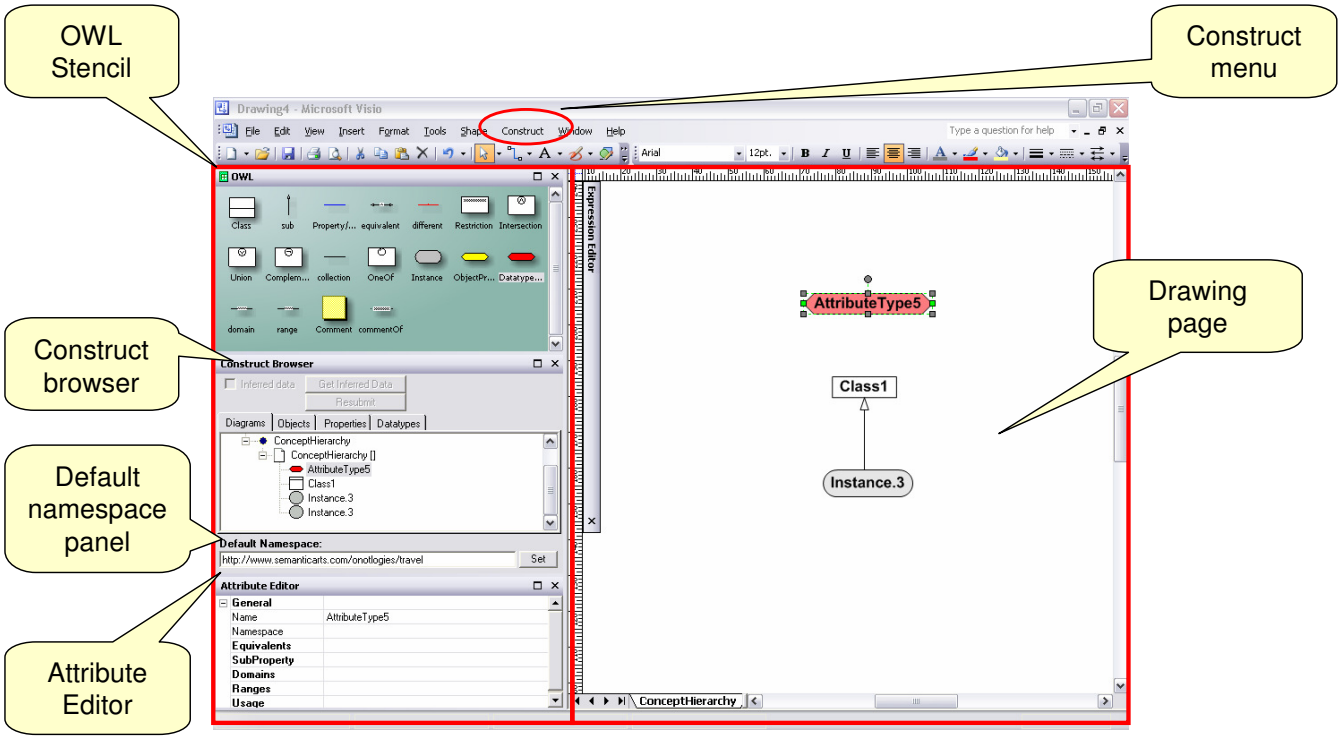
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

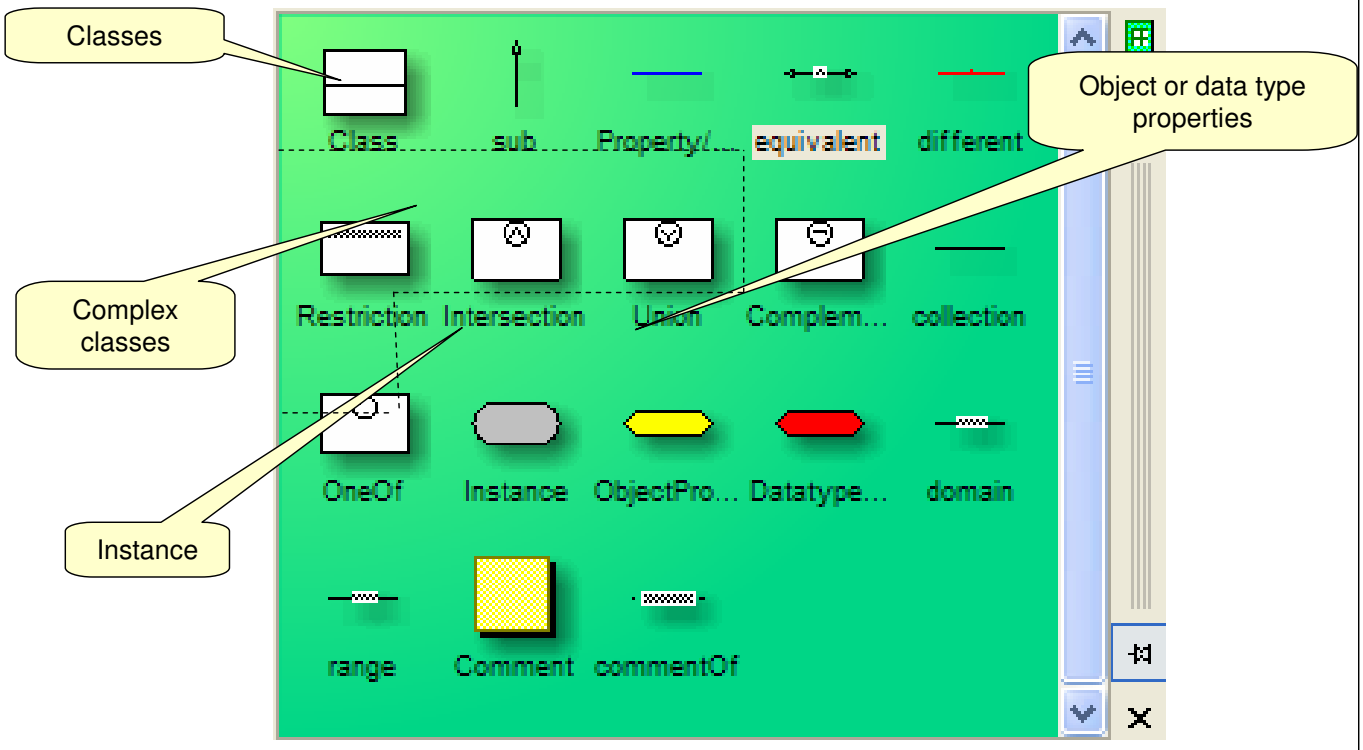
# The Construct Interface




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

## Student Notes:

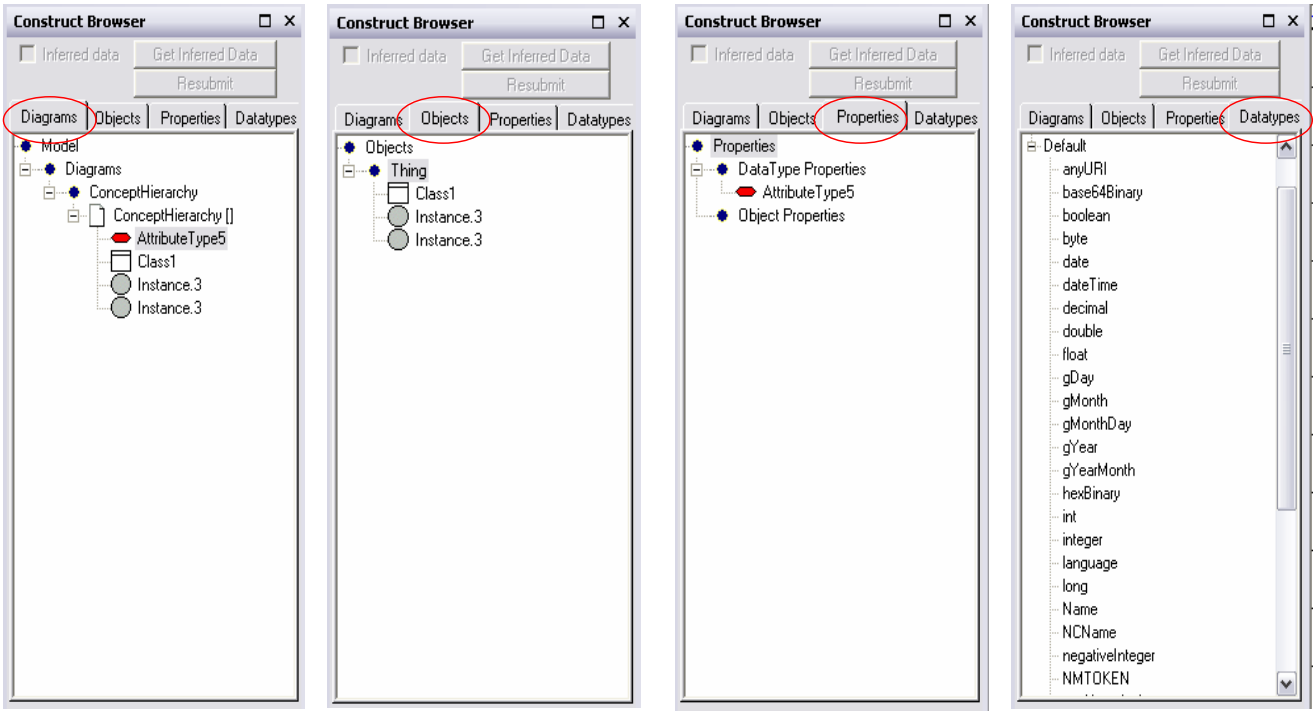
# Ontology Modeling Icons




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

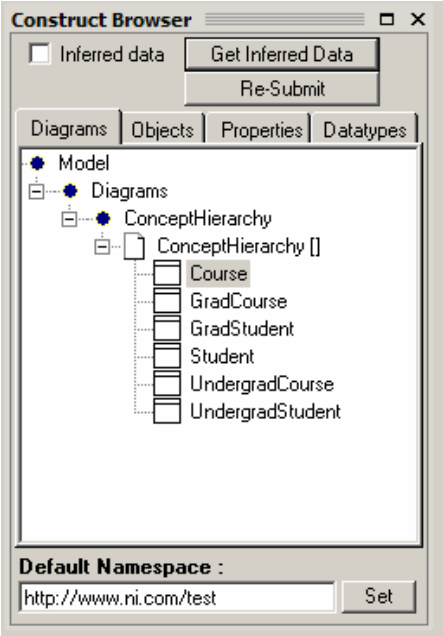
# Construct Browser Tabs




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Ontology namespaces

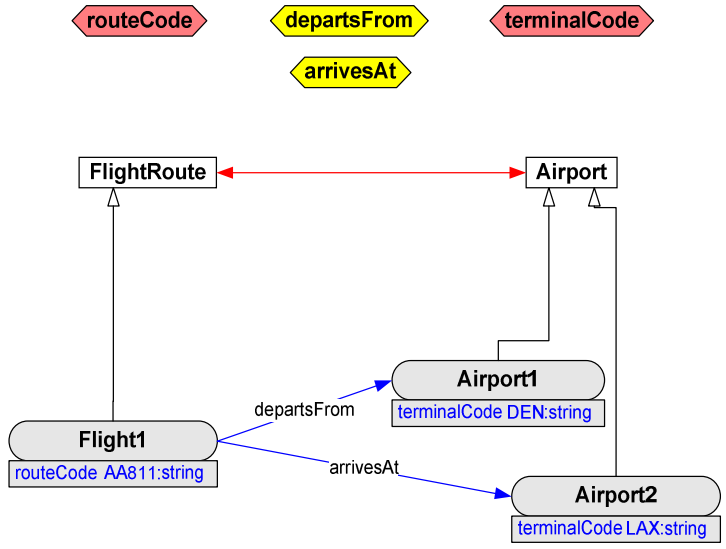



1. Every ontology object can be assigned a namespace value.
2. The ontology can be assigned a default namespace.
3. The object namespace forms a part of the unique identifier for an ontology object.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# The Travel Ontology




Throughout this course each student is going to build and test a simple travel ontology using Construct.

In this first module we will incrementally build the first part of that ontology that describes FlightRoutes and Airports.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Set the Default Namespace

- In an OWL ontology **classes**, **properties** and **individuals** are all identified with **URI**s (Universal Resource Identifiers).
- URIs uniquely identify a resource:

http://www.semanticarts.com/ontologies/travel#Airport




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Our first ontology

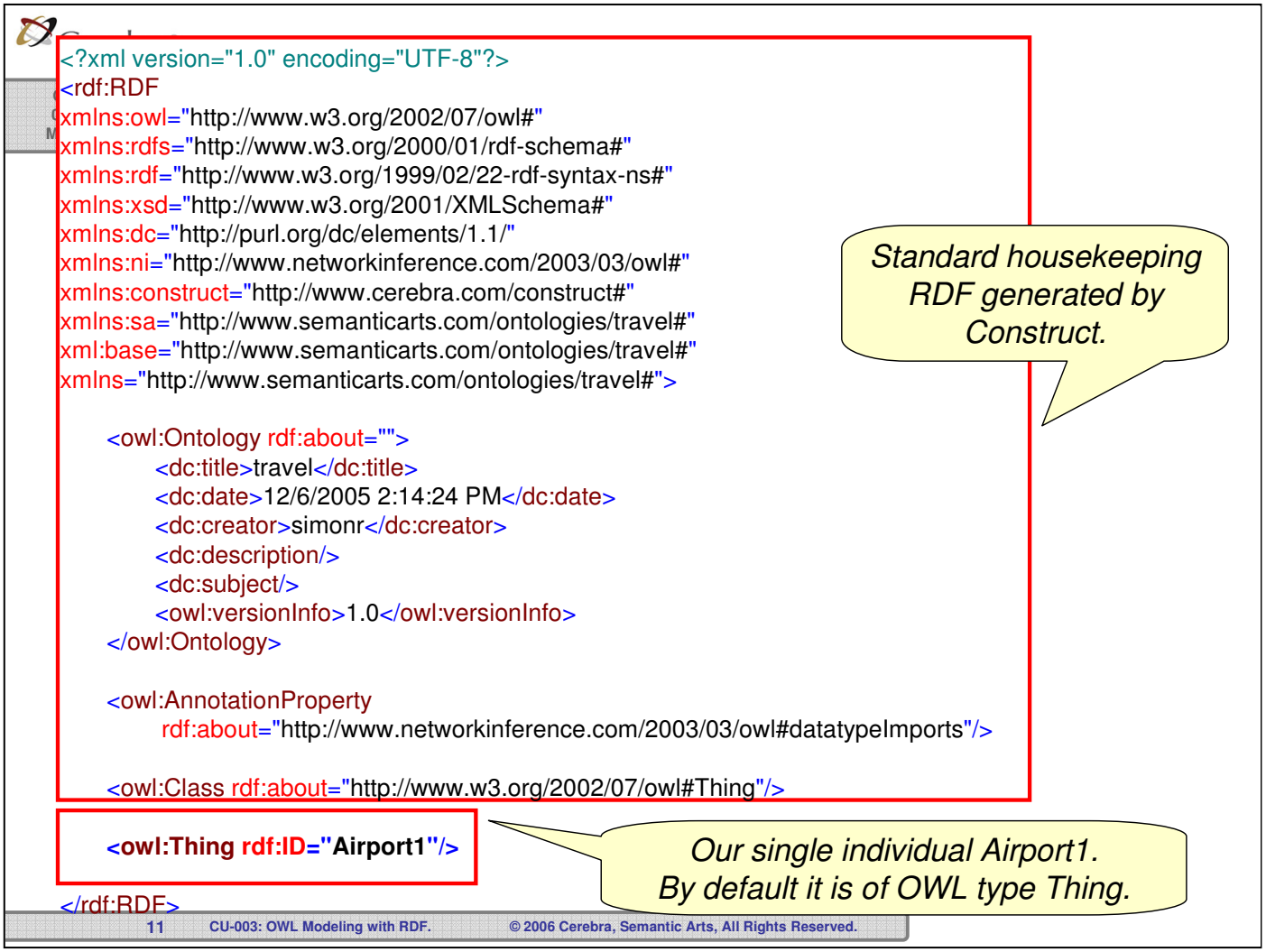



We have asserted that an individual thing exists in our domain, uniquely identified as http://www.semanticarts.com/ontologies/travel#Airport1

where http://www.semanticarts.com/ontologies/travel# is the default namespace for the ontology.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:




## Student Notes:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:ni="http://www.networkinference.com/2003/03/owl#"
  xmlns:construct="http://www.cerebra.com/construct#"
  xmlns:sa="http://www.semanticarts.com/ontologies/travel#"
  xml:base="http://www.semanticarts.com/ontologies/travel#"
  xmlns="http://www.semanticarts.com/ontologies/travel#">

    <owl:Ontology rdf:about="">
        <dc:title>travel</dc:title>
        <dc:date>12/6/2005 2:14:24 PM</dc:date>
        <dc:creator>simonr</dc:creator>
        <dc:description/>
        <dc:subject/>
        <owl:versionInfo>1.0</owl:versionInfo>
    </owl:Ontology>

    <owl:AnnotationProperty
        rdf:about="http://www.networkinference.com/2003/03/owl#datatypeImports"/>

    <owl:Class rdf:about="http://www.w3.org/2002/07/owl#Thing"/>

    <owl:Thing rdf:ID="Airport1"/>

</rdf:RDF>
```

*The Semantic Web stack. Each standard builds on the one below it.*

| |
|---|
| OWL<br>*Logical layer* |
| RDF Schema<br>*Ontological vocabulary layer* |
| RDF and RDF/XML<br>*Basic relational language layer* |
| XML and XMLSchema<br>*Transport Syntax Layer* |
| URIs and Namesapces<br>*Symbol / reference Layer* |

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

## Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:ni="http://www.networkinference.com/2003/03/owl#"
xmlns:construct="http://www.cerebra.com/construct#"
xml:base="http://www.semanticarts.com/ontologies/travel#"
xmlns:sa="http://www.semanticarts.com/ontologies/travel#"
xmlns="http://www.semanticarts.com/ontologies/travel#">

    <owl:Ontology rdf:about="">
        <dc:title>travel</dc:title>
        <dc:date>12/6/200
        <dc:creator>simon
        <dc:description/>
        <dc:subject/>
        <owl:versionInfo>1.0</owl:versionInfo>
    </owl:Ontology>

    <owl:AnnotationProperty
        rdf:about="http://www.networkinference.com/2003/03/owl#datatypeImports"/>

    <owl:Class rdf:about="http://www.w3.org/2002/07/owl#Thing"/>

    <owl:Thing rdf:ID="Airport1"/>

</rdf:RDF>
```

Default Namespace:
http://www.semanticarts.com/ontologies/travel
Set


© 2006 Cerebra, Semantic Arts, All Rights Reserved.

## Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Individuals – The Essentials

- **Individuals** represent either physical things or virtual concepts that the ontology describes and categorizes.
- An individual does not have to belong to a class other than the OWL super class *Thing*.
- An individual can be a member of many different classes.
- An individual is uniquely identified by a **URI** (Universal Resource Identifier).
- An individual is sometimes called an **instance**.

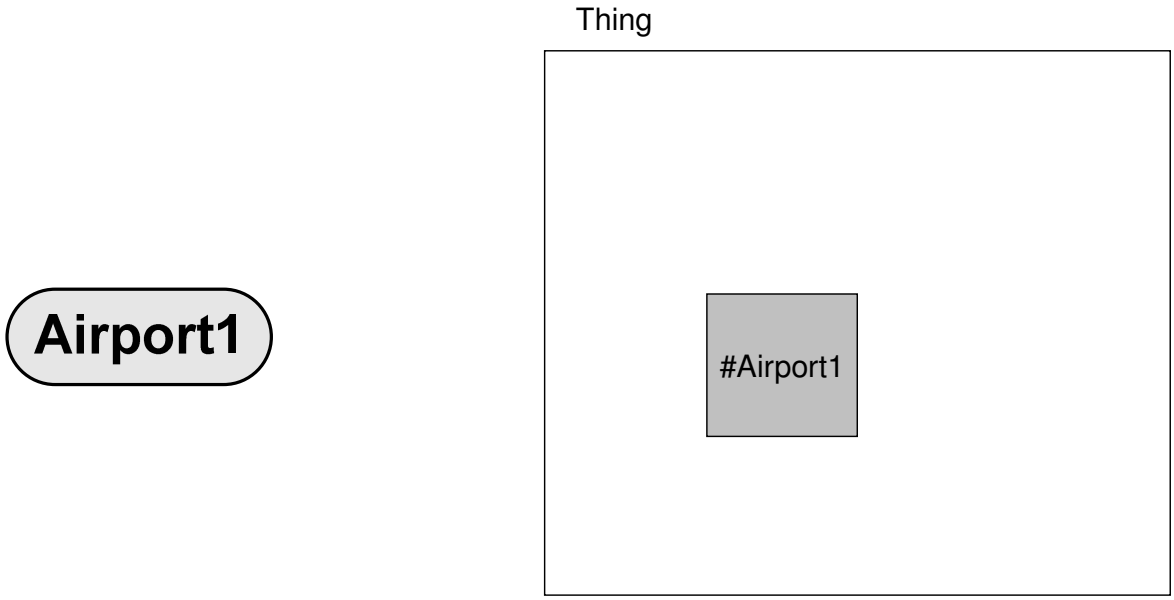
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

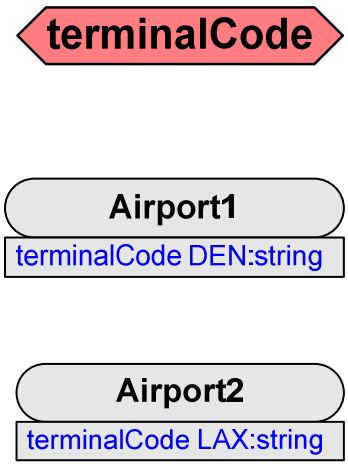
# What we have asserted




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# A Datatype Property




We have defined a property *terminalCode* that can have a single string value that can be used to describe *Thing*s in general.

We used it to describe our individuals:

*Airport1* (has a) *terminalCode* "DEN"
*Airport2* (has a) *terminalCode* "LAX"

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# The exported OWL

*We'll cover what being a functional property means later in the course.*

```
<owl:DatatypeProperty rdf:ID="terminalCode">
   <rdf:type rdf:resource=
        "http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:DatatypeProperty>

<owl:Thing rdf:ID="Airport1">
    <sa:terminalCode rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#string">DEN</sa:terminalCode>
</owl:Thing>

<owl:Thing rdf:ID="Airport2">
   <sa:terminalCode rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#string">LAX</sa:terminalCode>
</owl:Thing>
```

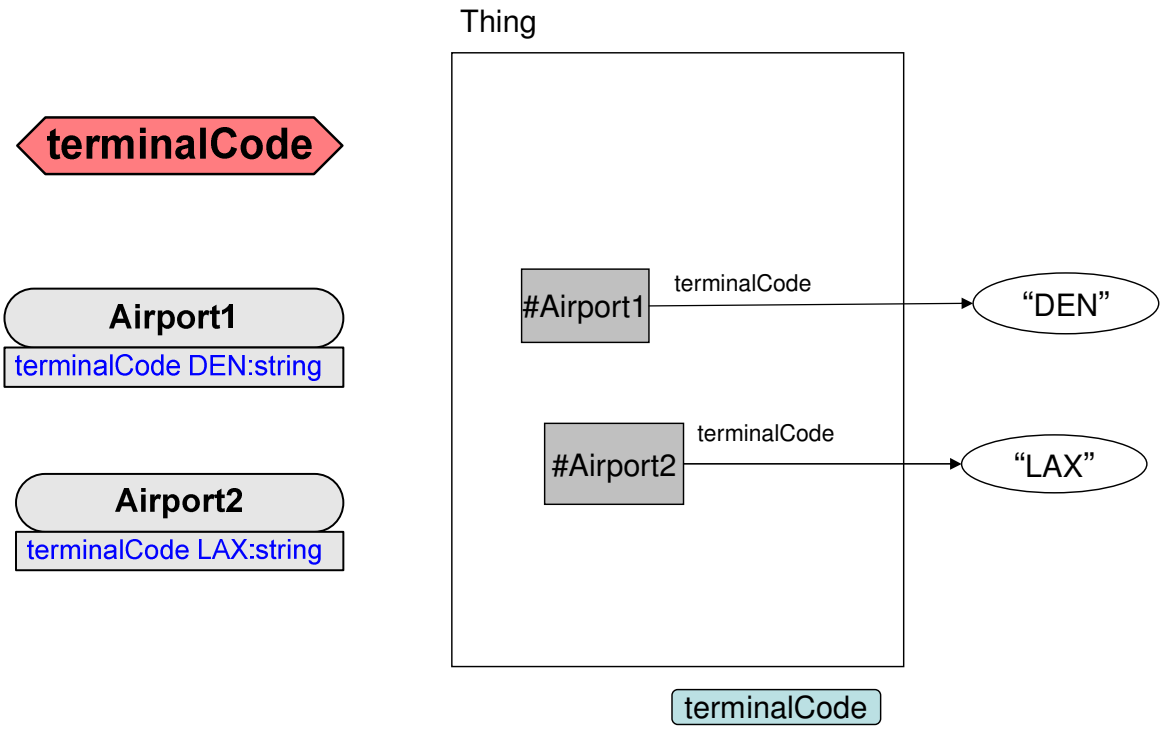
OWL uses the XML schema data types.

For this individual terminalCode has a value of the string literal "LAX".

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# What we have asserted




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Datatype Properties – The Essentials

- Properties are used to describe individuals.
- Properties are uniquely identified by a URI, the same as individuals.
- Properties are not dependent on classes. For example, you could use terminalCode to describe individual airports, bus stations or train stations.
- Datatype properties take a typed literal as a value.

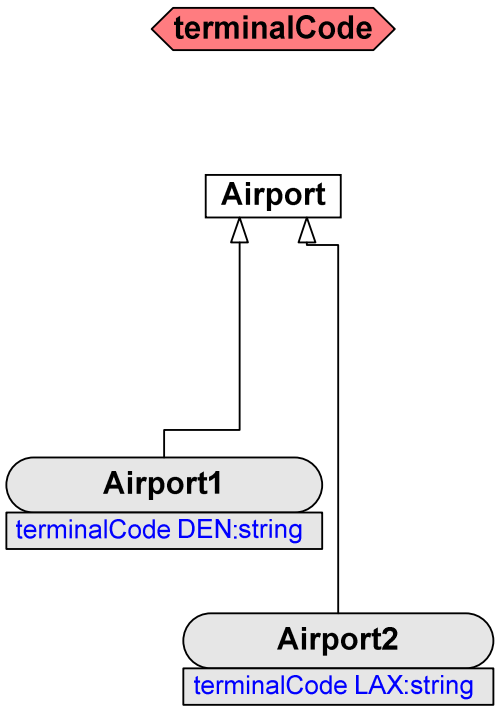
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# A Named Class




We have defined a class Airport and declared that the individuals Airport1 and Airport2 are members of that class.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# The exported OWL

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

*Declares the class and names it Airport.*

```
<sa:Airport rdf:ID="Airport1">
    <sa:terminalCode rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#string">DEN</sa:terminalCode>
</sa:Airport>
```

*Instead of Thing, the individuals are now typed as Airports.*

```
<sa:Airport rdf:ID="Airport2">
    <sa:terminalCode rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#string">LAX</sa:terminalCode>
</sa:Airport>
```

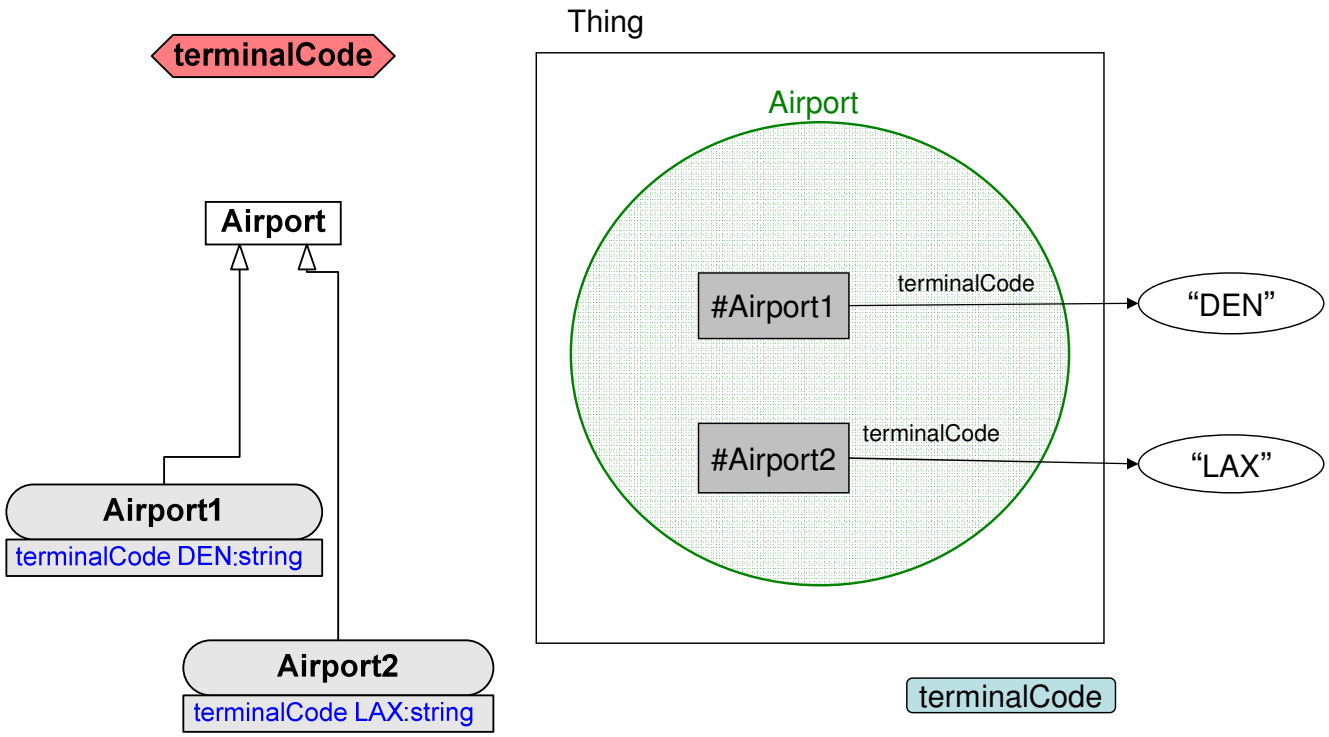
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# What we have asserted




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Classes – The Essentials

- A class is a set of individuals. A named class is a set with a URI.
- All classes are sub-classes of the OWL class Thing. E.g., all Airports are types of Thing.
- Individuals can be members of multiple classes.
- One way to assert that an individual is a member of a class is to declare it explicitly.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

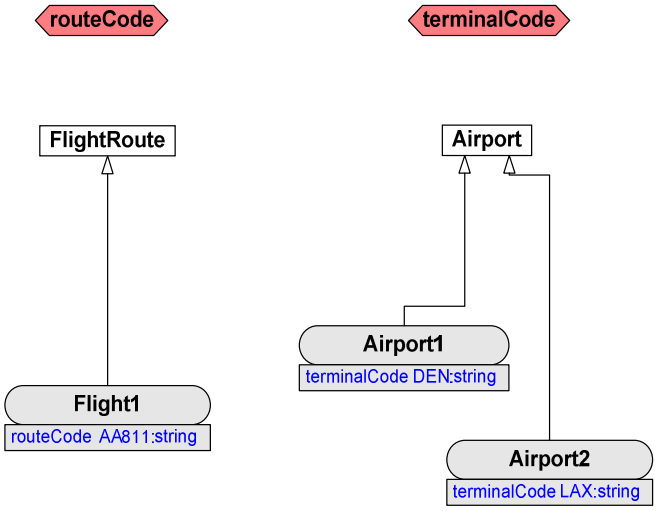
# Exercise 1

- Add another individual to the ontology with the URI *Flight1*.
- Add a data type property *routeCode*.
- Assert that *Flight1* has a *routeCode* of "AA811".
- Define a class *FlightRoute*.
- Declare individual *Flight1* to be a a member of class *FlightRoute*.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Exercise 1




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Query Syntax Basics

A brief explanation of the
Cerebra XQuery language

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Querying an Ontology

- In order for the knowledge and information modeled in an ontology to be useful, there must be a way to query the ontology. (Otherwise, it's just a pretty picture.)
- Cerebra Server provides two query types:
  - **Conjunctive** queries are based on XQuery and return **individuals** and **data values** (literals) from the ontology.
  - **Taxonomic** queries return **classes** and **properties** and will be covered later.
- This section will provide a brief overview of how the conjunctive query language looks and will explain a simple query.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

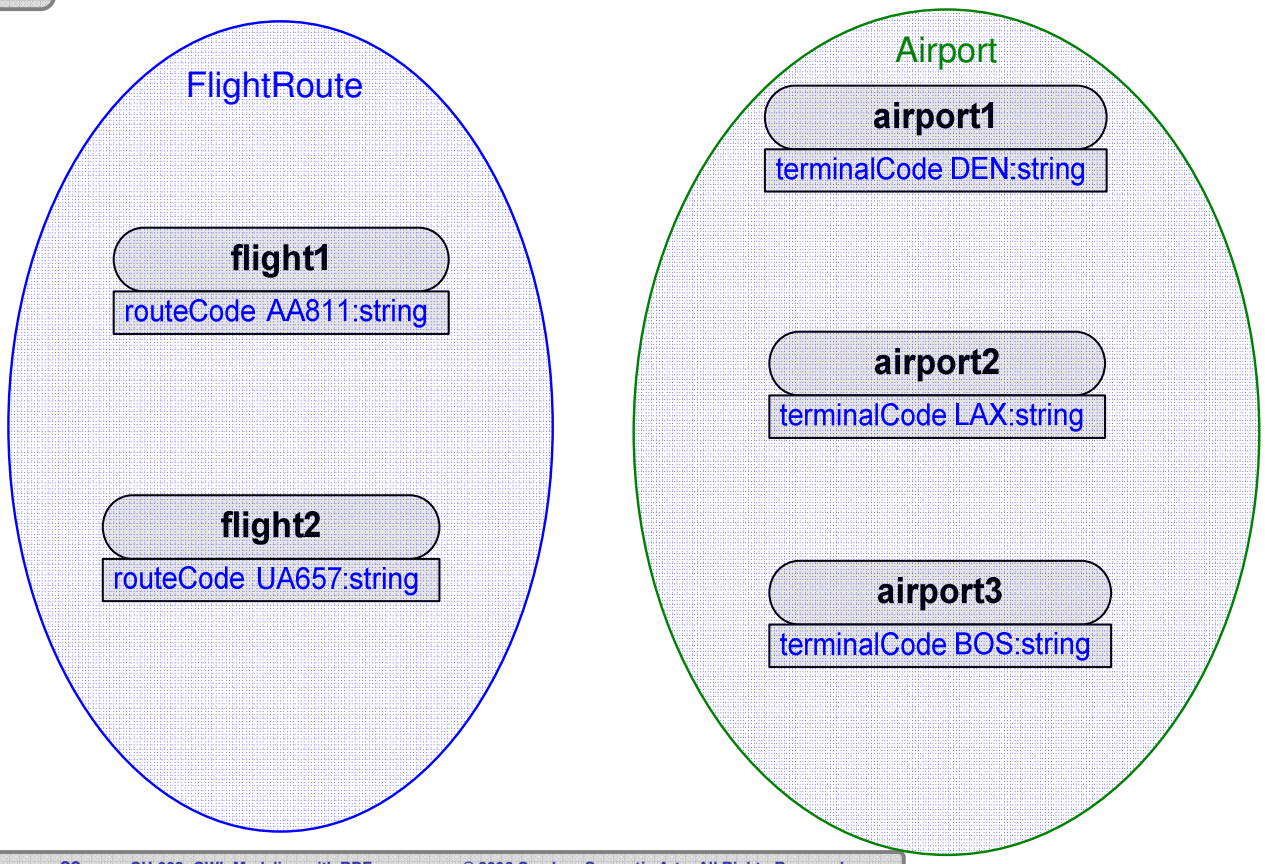
# Cerebra Query Functions

- Cerebra's query language extends the functionality of XQuery by using a set of Cerebra-specific functions.
- These functions are used to define the conditions that a query must meet, and are found in the *where* clause of the query.
- We will introduce these functions as they are used in the examples.
- The *Cerebra Server Developer's Guide* contains a complete reference.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# A Simple Travel Ontology




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Querying the Flight Ontology

*What individuals in the ontology are members of 'Airport'?*

```
declare default function namespace
        "http://www.networkinference.com/cerebra/functions/3.1"
declare default element namespace
        "http://www.semanticarts.com/ontologies/travel"

for $i in instances()

where every $m in models() satisfies
  member($i,Airport,$m)

return <Airport>{$i}</Airport>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Querying the Flight Ontology

*What individuals in the ontology are members of 'Airport'?*

```
declare default function namespace
        "http://www.networkinference.com/cerebra/functions/3.1"
declare default element namespace
        "http://www.semanticarts.com/ontologies/travel"

for $i in instances()

where every $m in models() satisfies
  member($i,Airport,$m)

return <Airport>{$i}</Airport>
```

*Cerebra's XQuery language uses a set of custom, Cerebra-specific functions. This is a required declaration that tells the Cerebra Server inference engine which functions to use.*
*(For the most part, this line will never change.)*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Querying the Flight Ontology

*What individuals in the ontology are members of 'Airport'?*

```
declare default function namespace
        "http://www.networkinference.com/cerebra/functions/3.1"
declare default element namespace
        "http://www.semanticarts.com/ontologies/travel"

for $i in instances()

where every $m in models() satisfies
  member($i,Airport,$m)

return <Airport>{$i}</Airport>
```

*This line declares the default namespace for classes, properties, and individuals referred to in the query. This declaration is a time-saver. It avoids having to type out the fully qualified name of each resource. The namespaces in your query must match those in the ontology being queried.*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Querying the Flight Ontology

*What individuals in the ontology are members of 'Airport'?*

```
declare default function namespace
        "http://www.networkinference.com/cerebra/functions/3.1"
declare default element namespace
        "http://www.semanticarts.com/ontologies/travel"

for $i in instances()

where every $m in models() satisfies
  member($i,Airport,$m)

return <Airport>{$i}</Airport>
```

*This "for" statement creates a variable $i (variables are always referred to with a dollar sign) and binds it to the set of every individuals in the ontology. Other bindings are possible; this will be covered in detail later.*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Querying the Flight Ontology

*What individuals in the ontology are members of 'Airport'?*

```
declare default function namespace
        "http://www.networkinference.com/cerebra/functions/3.1"
declare default element namespace
        "http://www.semanticarts.com/ontologies/travel"

for $i in instances()

where every $m in models() satisfies
  member($i,Airport,$m)

return <Airport>{$i}</Airport>
```

*We are looking only for individuals that satisfy the conditions in the "where" clause. In this case, "where $i is a member of Airport." (Models and satisfiability will be covered in the next section.) Note that we do not need to include a namespace for Airport since it will use the default declared above.*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# The *member* Function

member(*individual,class,model)*

- The *member* function checks to see if *individual* is a member of *class* in the given model.
- (Models are covered in detail in the next section.)

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Querying the Flight Ontology

*What individuals in the ontology are members of 'Airport'?*

```
declare default function namespace
        "http://www.networkinference.com/cerebra/functions/3.1"
declare default element namespace
        "http://www.semanticarts.com/ontologies/travel"

for $i in instances()

where every $m in models() satisfies
  member($i,Airport,$m)

return <Airport>{$i}</Airport>
```

*For every valid result found, the XML in the return block will be added to the output. In this case, each individual found that matches the criterion will be added to the output surrounded by a set of <Airport> tags.*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Querying the Flight Ontology

*What individuals in the ontology are members of 'Airport'?*

```
declare default function namespace
        "http://www.networkinference.com/cerebra/functions/3.1"
declare default element namespace
        "http://www.semanticarts.com/ontologies/travel"

for $i in instances()

where every $m in models() satisfies
  member($i,Airport,$m)

return <Airport>{$i}</Airport>
```

*Putting curly braces around a variable evaluates to the URI of the individual or property to which it is bound. The fully qualified URI will be returned (namespace and name).*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Querying the Flight Ontology

*What individuals in the ontology are members of 'Airport'?*

```
<RESULT>
  <Airport>
    http://www.semanticarts.com/ontologies/travel#airport1
  </Airport>
  <Airport>
    http://www.semanticarts.com/ontologies/travel#airport2
  </Airport>
  <Airport>
    http://www.semanticarts.com/ontologies/travel#airport3
  </Airport>
</RESULT>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Exercise 2a

- Submit your travel ontology to the server.
- Query the ontology to find every *FlightRoute*.
  - member($i,FlightRoute,$m)

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Exercise 2b

- Query the travel ontology for *FlightRoute*s, this time using the statement:

```
where some $m in models() satisfies
  member($i,FlightRoute,$m)
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Open World Reasoning

Open world reasoning and
provable and satisfiable queries

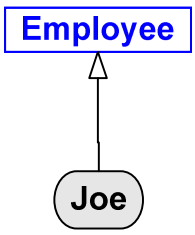
CU-003: OWL Modeling with RDF. © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Three simple assertions




Contractor

```
<owl:Class rdf:ID="Employee"/>
<owl:Class rdf:ID="Contractor"/>

<construct:Employee rdf:ID="Joe"/>
```

```
for $i in instances()
where every $m in models() satisfies
      member($i, Employee, $m)
return {$i}
```

```
<RESULT>
http://www.cerebra.com/construct#Joe
</RESULT>
```

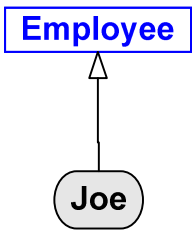
```
for $i in instances()
where every $m in models() satisfies
      member($i, Contractor, $m)
return {$i}
```

```
<RESULT>
</RESULT>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Three simple assertions




Contractor

```
<owl:Class rdf:ID="Employee"/>
<owl:Class rdf:ID="Contractor"/>

<construct:Employee rdf:ID="Joe"/>
```

```
for $i in instances()
where some $m in models() satisfies
      member($i, Employee, $m)
return {$i}
```

```
<RESULT>
http://www.cerebra.com/construct#Joe
</RESULT>
```

```
for $i in instances()
where some $m in models() satisfies
      member($i, Contractor, $m)
return {$i}
```

```
<RESULT>
http://www.cerebra.com/construct#Joe
</RESULT>
```

?

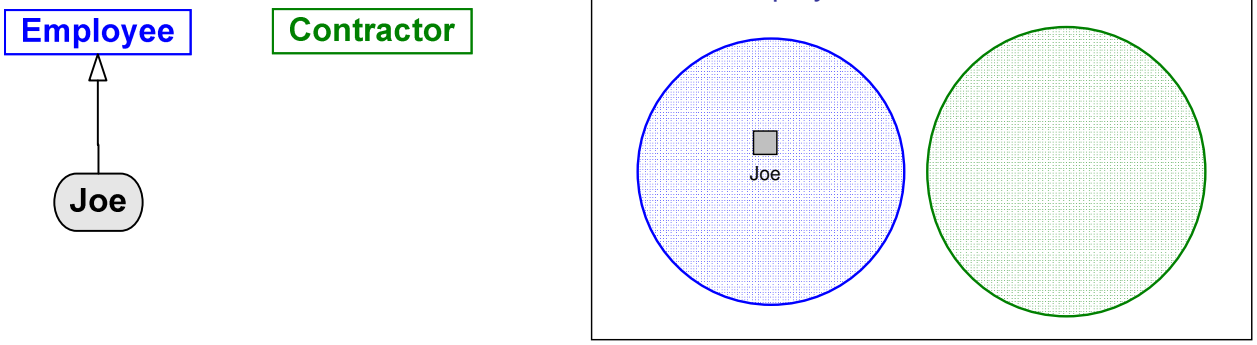
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# What it looked like we asserted



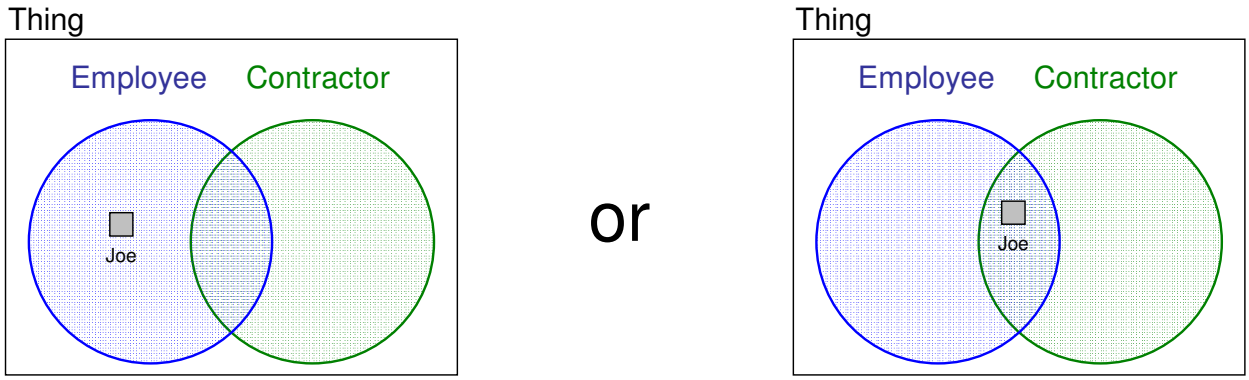

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# What we actually asserted




These are the two **models** of our domain that conform to the three assertions in the ontology. In both, Joe is an Employee.

- In one, Joe is an Employee but not a Contractor.
- In the other, Joe is both an Employee and a Contractor.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

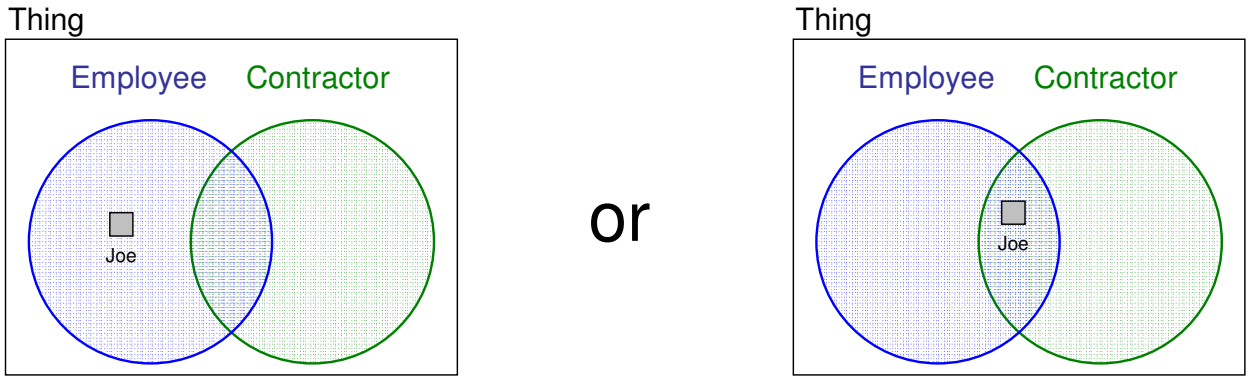
An OWL ontology defines a space of possible interpretations of the domain being modeled. An interpretation is a fixed model of all instances in the ontology, the relationship between those instances, and exactly what classes those members are an instance of. Some facts are true in in EVERY interpretations and some facts are true in in SOME interpretations but not others.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# What we actually asserted




We never explicitly asserted that:

- An Employee could not also be a Contractor
- Joe was not a Contractor

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Open World, Closed World

- OWL assumes an open world.
- In a closed world, only what is asserted is the case. "Joe is an Employee" -- no more, no less.
- In an open world, other strange and wonderful things may be the case. "Joe could be a Contractor."
- In an open world ontology, anything is possible as long as it is still consistent with the ontology's assertions.
- A closed world ontology describes one definitive model of the world.
- An open world ontology describes multiple possible models of the world.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Provable Queries

A fact is **provable** if it is true in **every model**.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Satisfiable Queries

A fact is **satisfiable** if it is true in **some models**.

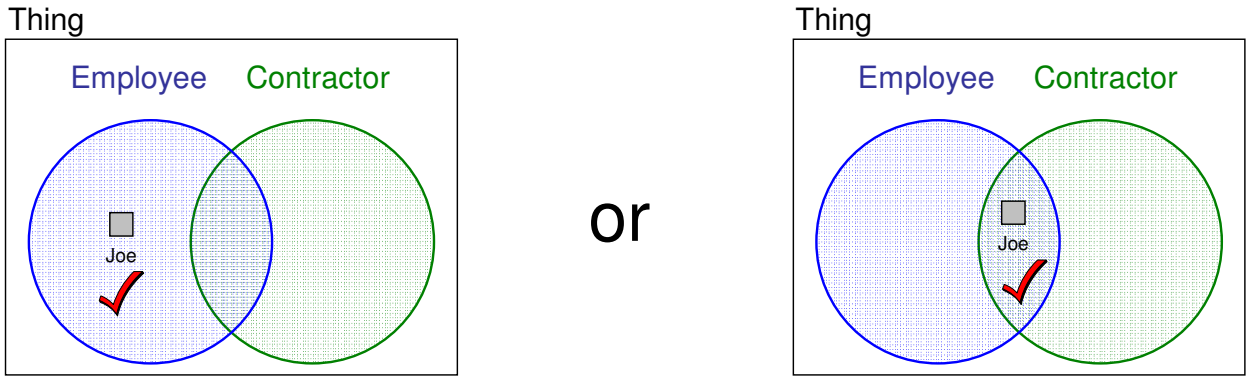
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# What instances are Employees in every model?




```
for $i in instances()
where every $m in models() satisfies
      member($i, Employee, $m)
return {$i}
```

```
<RESULT>
http://www.cerebra.com/construct#Joe
</RESULT>
```

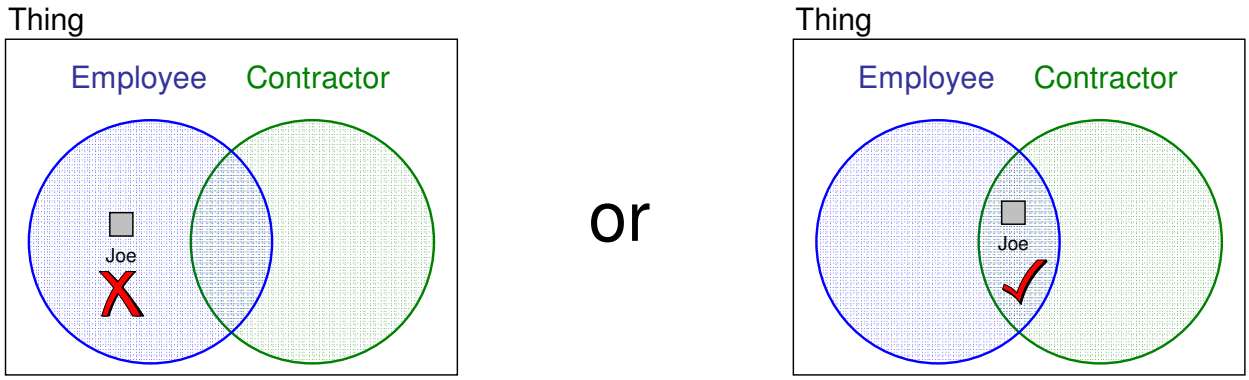
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# What individuals are a Contractor in every model?




```
for $i in instances()
where every $m in models() satisfies
      member($i, Contractor, $m)
return {$i}
```

```
<RESULT>
</RESULT>
```

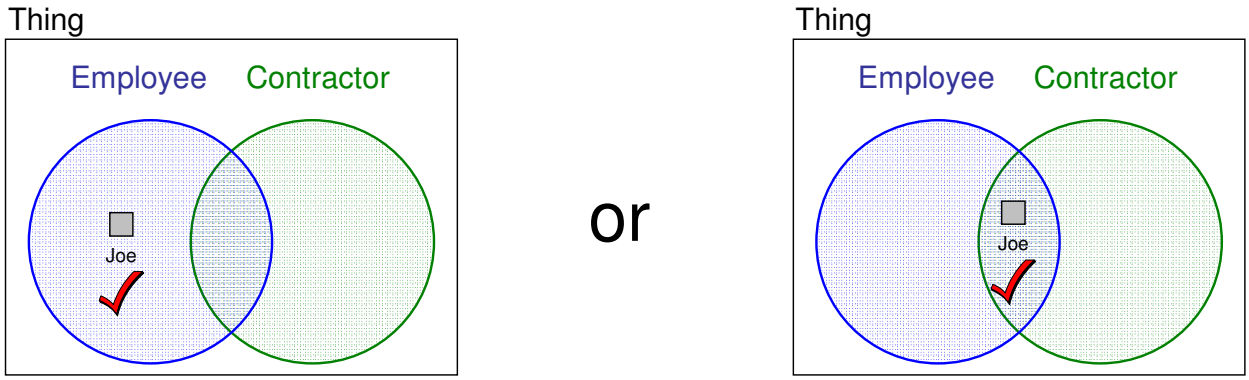
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# What individuals are Employees in some models?




```
for $i in instances()
where some $m in models() satisfies
      member($i, Employee, $m)
return {$i}
```

```
<RESULT>
http://www.cerebra.com/construct#Joe
</RESULT>
```

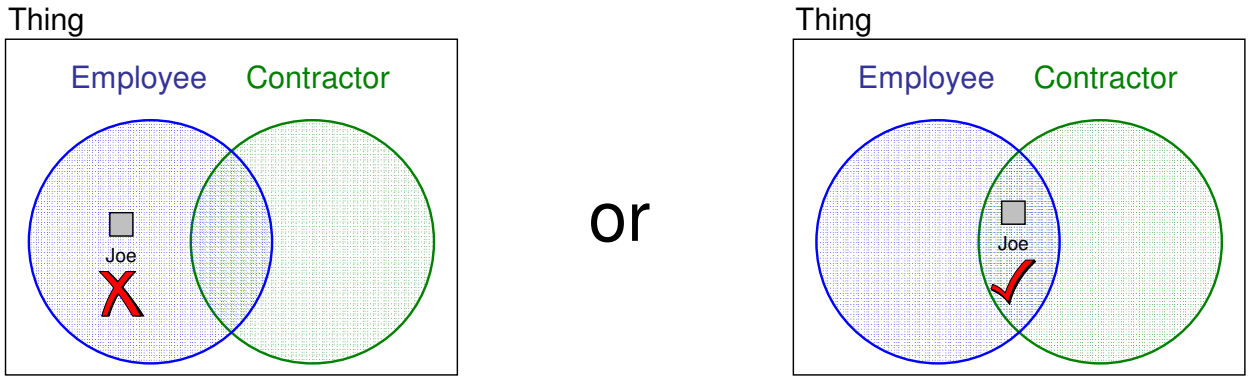
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# What individuals are a Contractor In some models?




```
for $i in instances()
where some $m in models() satisfies
      member($i, Contractor, $m)
return {$i}
```

```
<RESULT>
http://www.cerebra.com/construct#Joe
</RESULT>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Provable vs. Satisfiable

provable = every

satisfiable = some

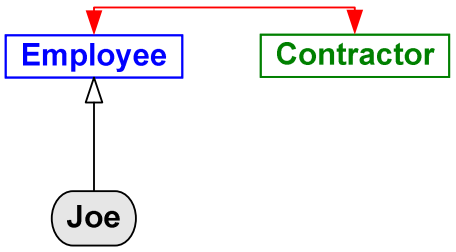
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# What we might have meant to assert




```
<owl:Class rdf:ID="Employee"/>
<owl:Class rdf:ID="Contractor"/>

<construct:Employee rdf:ID="Joe"/>

<owl:Class rdf:ID="Employee">
    <owl:disjointWith rdf:resource="#Contractor"/>
</owl:Class>
```

```
for $i in instances()
where some $m in models() satisfies
    member($i, Contractor, $m)
return {$i}
```

```
<RESULT>
</RESULT>
```

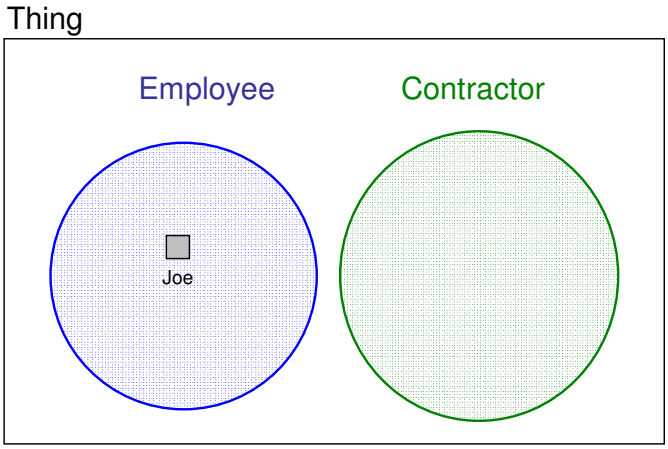
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# What we might have meant to assert




There is now only one possible model of our domain that conforms to the assertions in the ontology.

In this situation, a provable ("every") query and a satisfiable ("some") query will always return the same result.

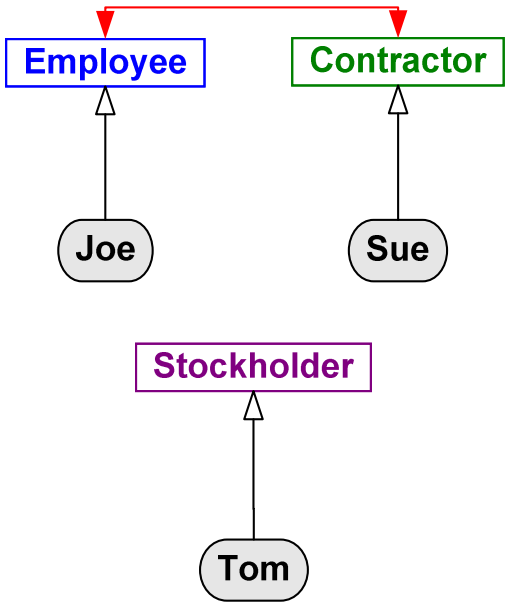
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Another example




```
<owl:Class rdf:ID="Employee"/>
<owl:Class rdf:ID="Contractor"/>
<owl:Class rdf:ID="Stockholder"/>

<owl:Class rdf:ID="Employee">
    <owl:disjointWith rdf:resource="#Contractor"/>
</owl:Class>

<construct:Employee rdf:ID="Joe"/>
<construct:Contractor rdf:ID="Sue"/>
<construct:Stockholder rdf:ID="Tom"/>
```

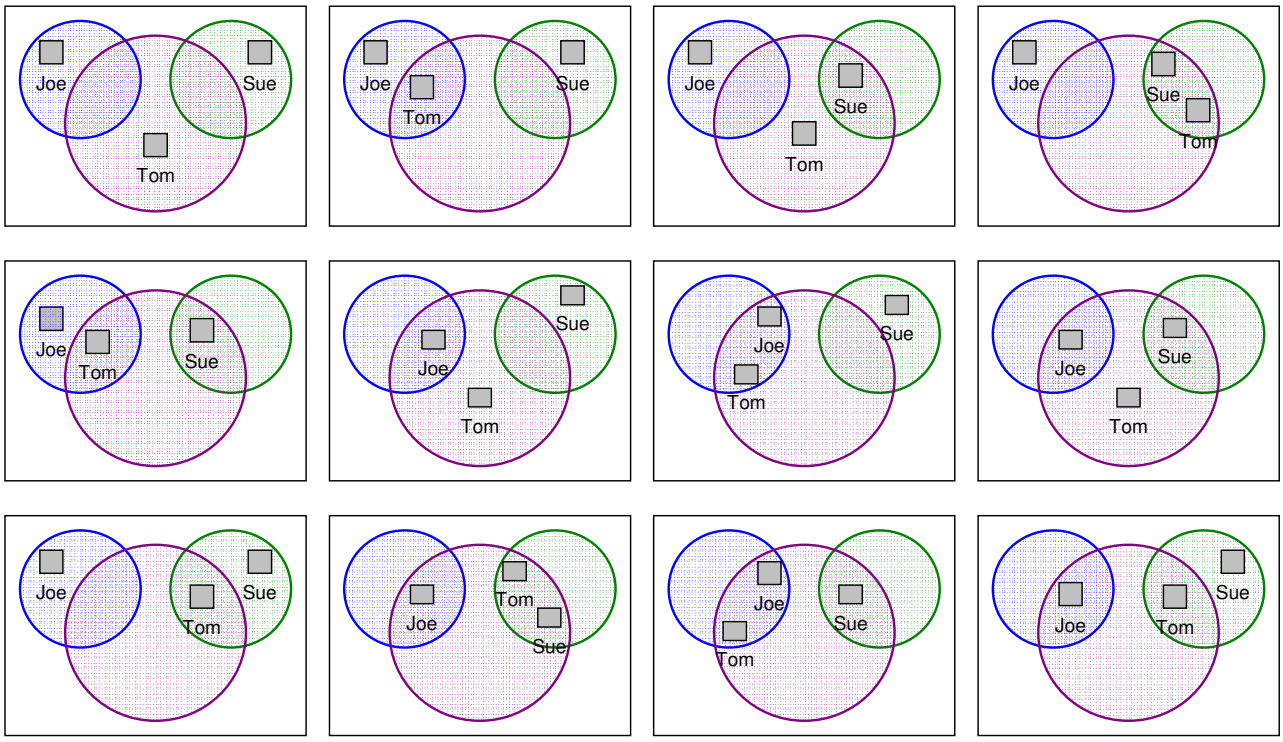
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# This ontology has 12 possible interpretations (models)




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

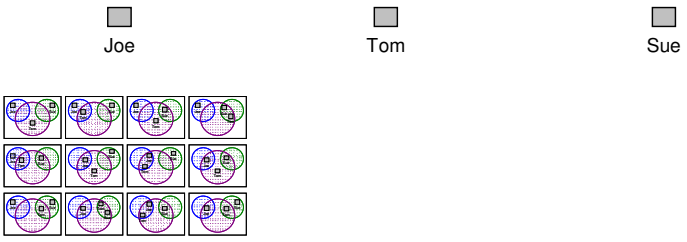
# Understanding a satisfiable (some models) query

```
for $i in instances()
where some $m in models() satisfies
      member($i, Employee, $m)
      and member($i, Stockholder, $m)
return  <PossibleEmployeeStockholder>{$i}</PossibleEmployeeStockholder>
```

For each individual in the ontology

For each possible model allowed by the ontology

Select any individual that meets the criteria (both an Employee and a Stockholder) in at least one (some) of the models.



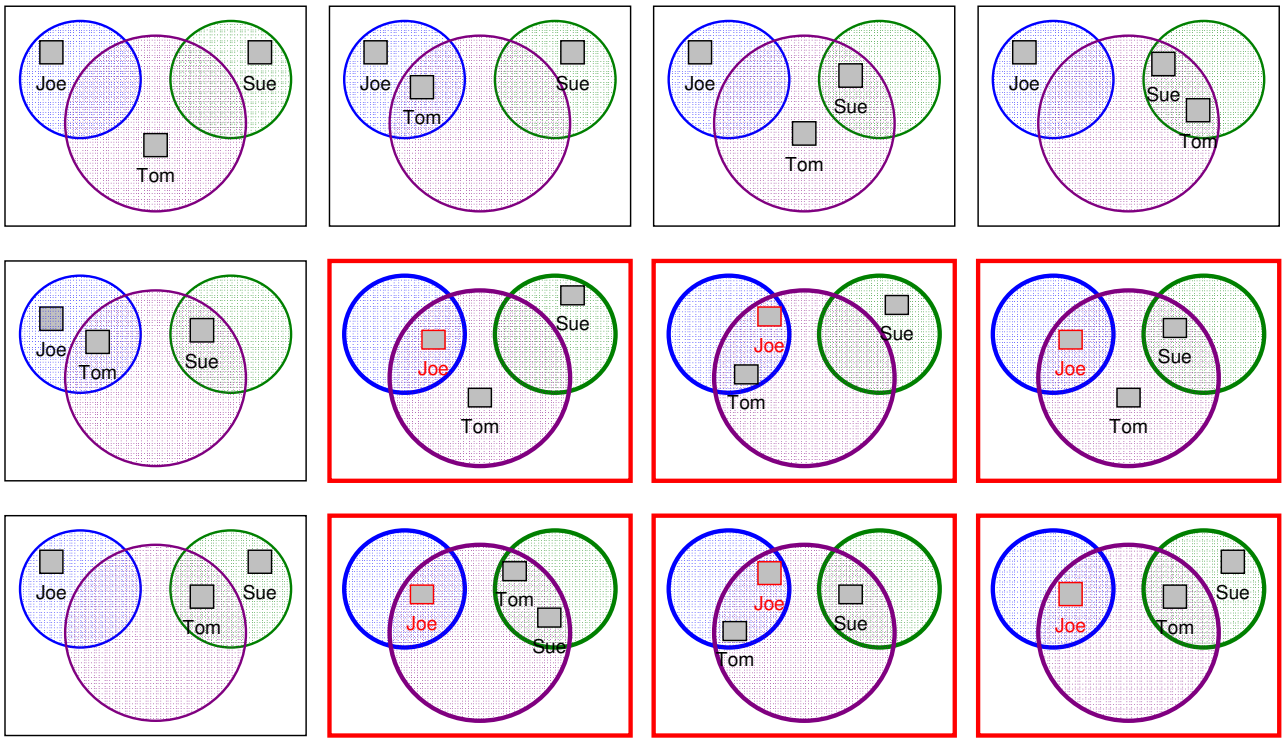

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Joe is an Employee Stockholder in at least one model



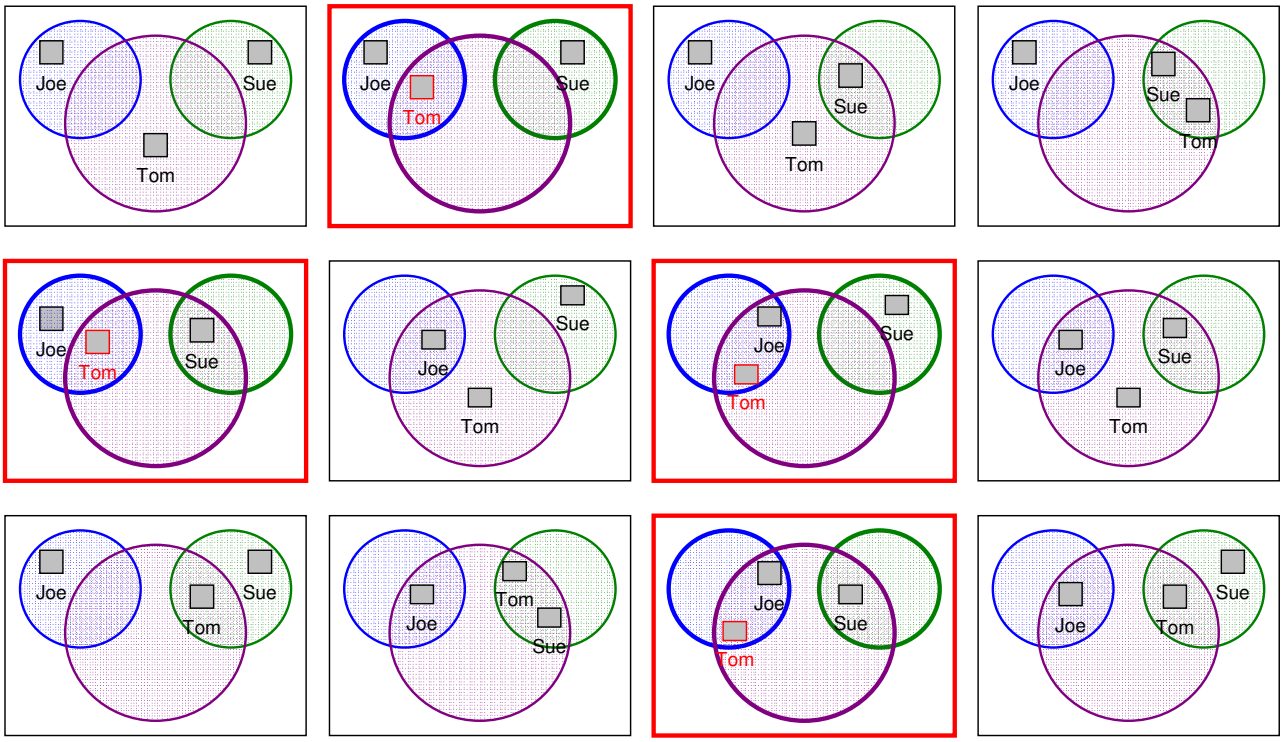

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Tom is an Employee Stockholder in at least one model



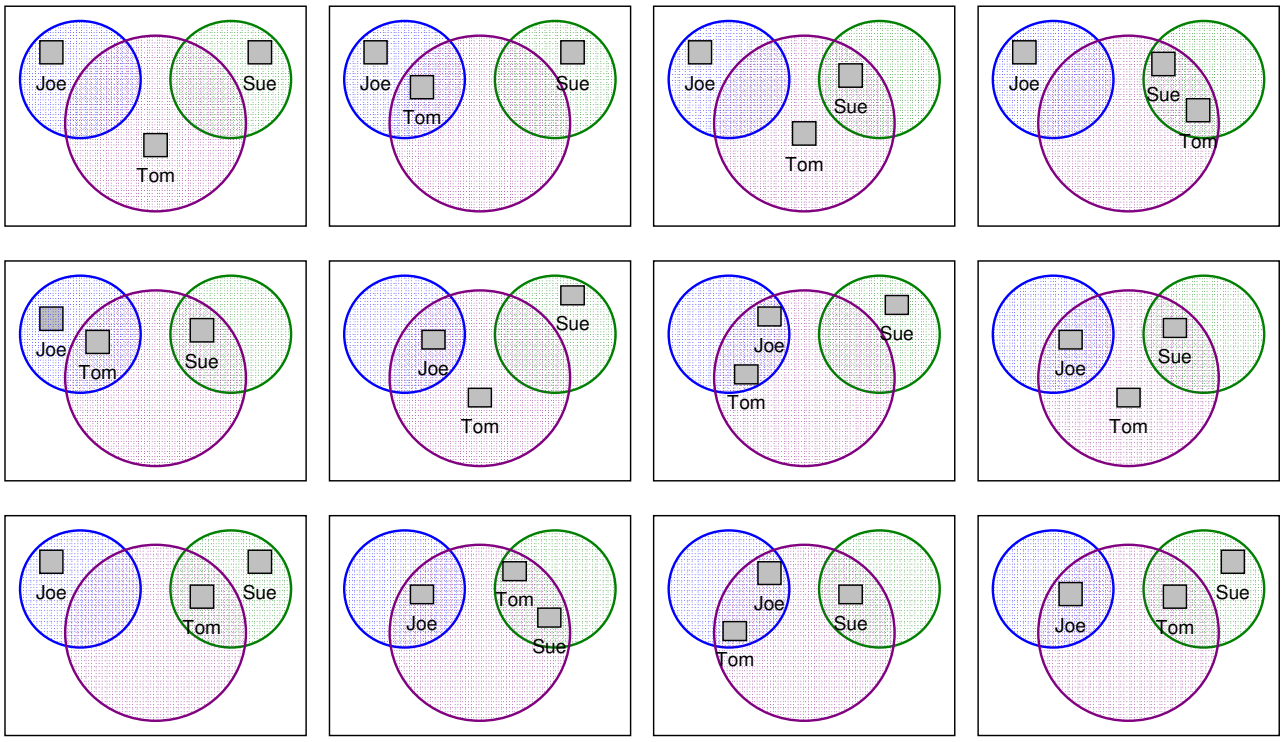

61 CU-003: OWL Modeling with RDF. © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Sue is not an Employee Stockholder in any of the models




62 CU-003: OWL Modeling with RDF. © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Understanding a satisfiable (some models) query

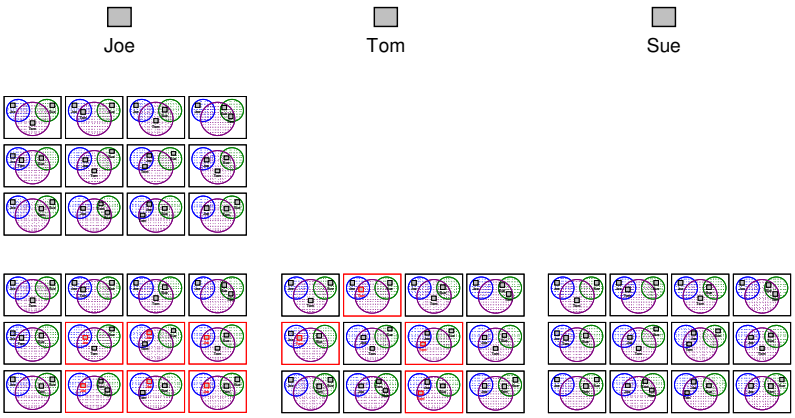
```
for $i in instances()
where some $m in models() satisfies
      member($i, Employee, $m)
      and member($i, Stockholder, $m)
return <PossibleEmployeeStockholder>{$i}</PossibleEmployeeStockholder>
```

For each individual in the ontology

For each possible model allowed by the ontology

Select any individual that meets the criteria (both an Employee and a Stockholder) in at least one (some) of the models.

Return some information about each selected individual.




http://www.cerebra.com/construct#**Joe**
http://www.cerebra.com/construct#**Tom**

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

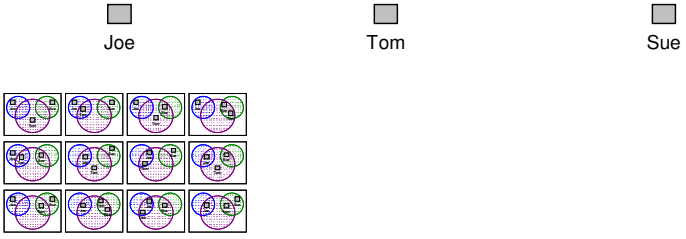
# Understanding a provable (every model) query

```
for $i in instances()
where every $m in models() satisfies
      member($i, Stockholder, $m)
return  <KnownStockholder>{$i}</KnownStockholder >
```

For each individual in the ontology

For each possible model allowed by the ontology

Select any individual that meets the criteria (is an Stockholder) in every model.



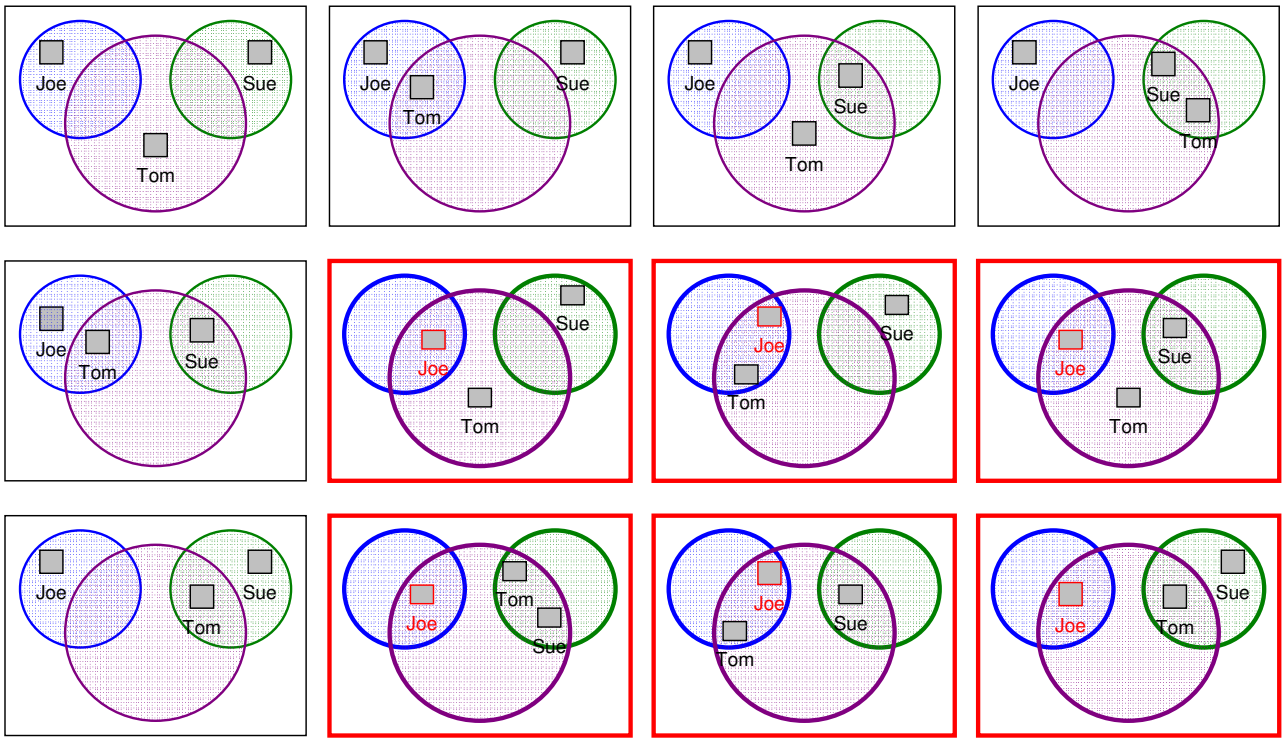

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Joe is not a Stockholder in every model



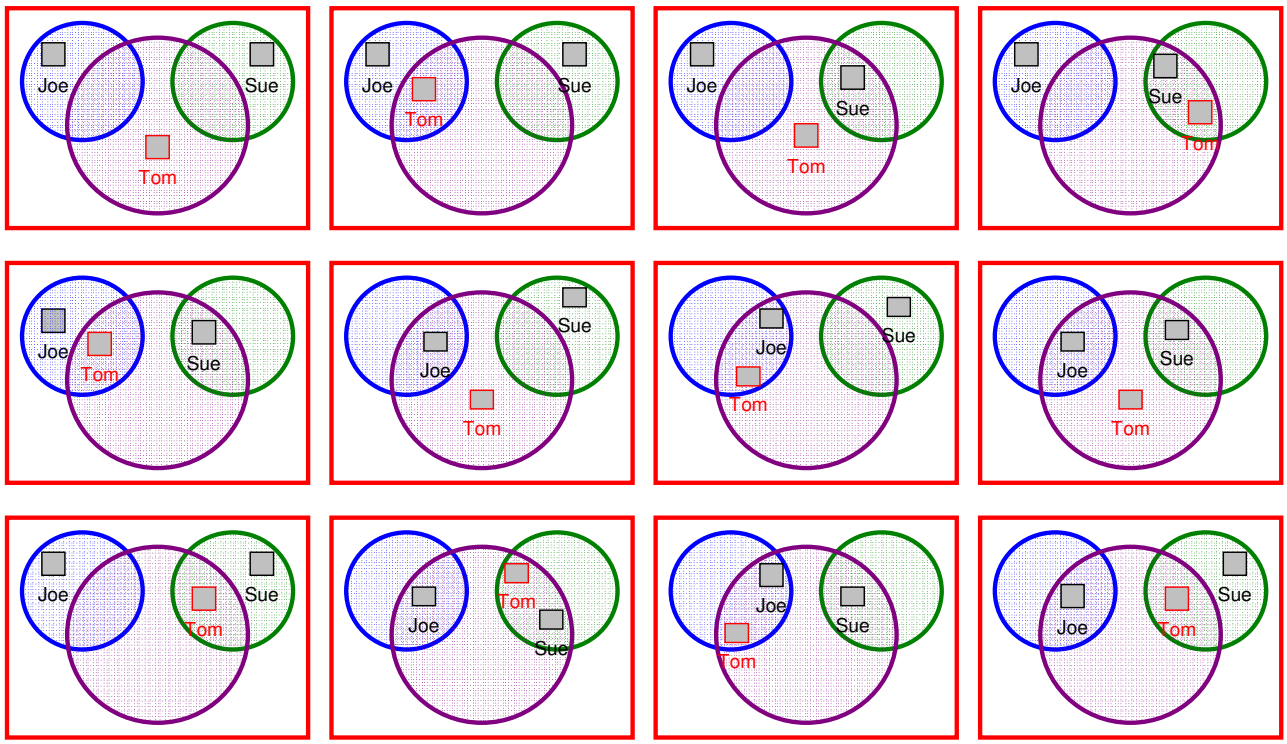

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Tom is a Stockholder in every model



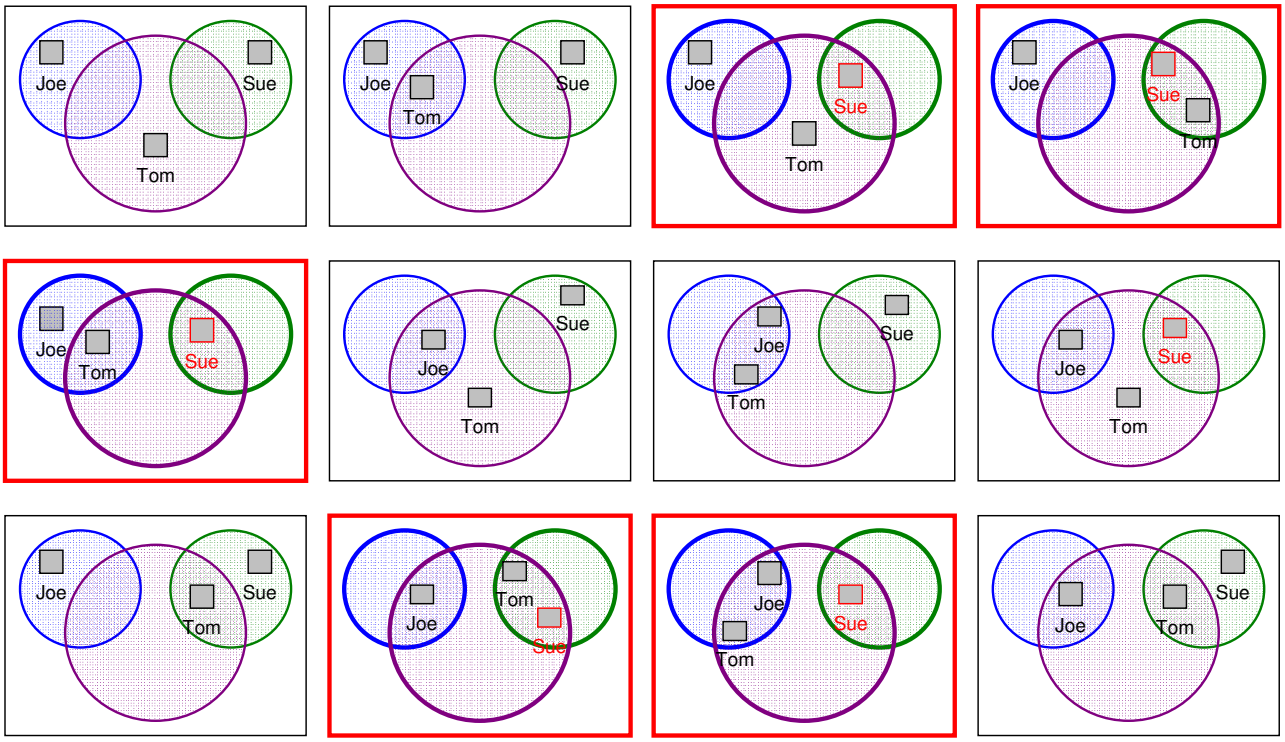

66 CU-003: OWL Modeling with RDF. © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Sue is not a Stockholder in every model




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Understanding a provable (every model) query

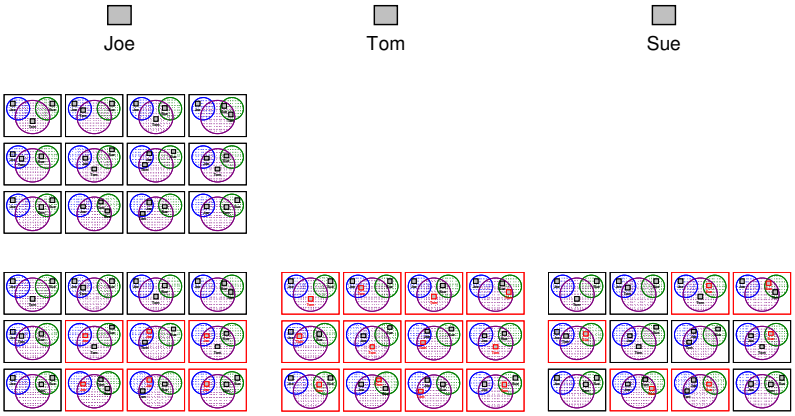
```
for $i in instances()
where every $m in models() satisfies
      member($i, Stockholder, $m)
return <KnownStockholder>{$i}</KnownStockholder >
```

For each individual in the ontology

For each possible model allowed by the ontology

Select any individual that meets the criteria (is an Stockholder) in every model.

Return some information about each selected individual.




http://www.cerebra.com/construct#**Tom**

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Note that if you declare that an instance is a member of the class it will always be provable a member of that class.

## Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Provably False

Something is provably false if it is false in every possible model of the ontology.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Understanding a provably not query

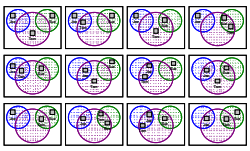
```
for $i in instances()
where every $m in models() satisfies
      not( member($i, Employee, $m) )
return  {$i}
```

For each individual in the ontology

For each possible model allowed by the ontology

Select any individual that meets the criteria (is not an Employee) in every model.

Joe Tom Sue



 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

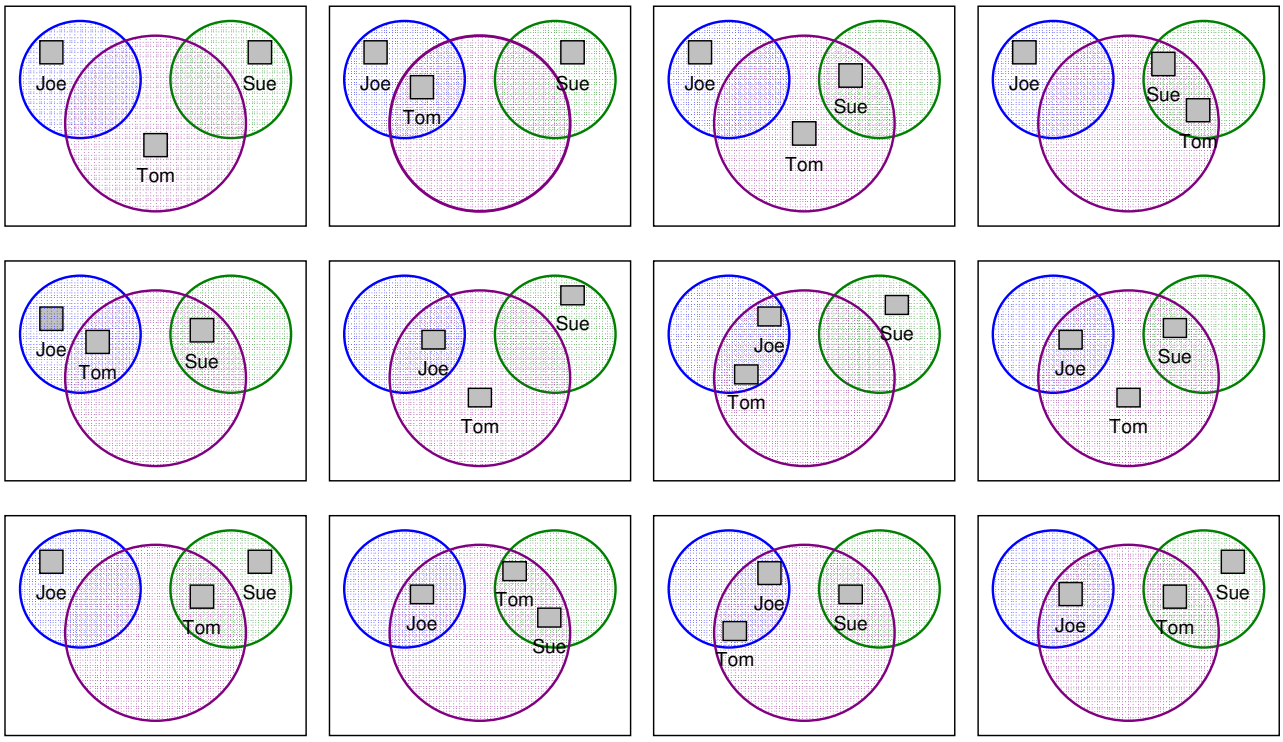
The “not” operator needs to be distinguished with a namespace suffix as there is a not operator in the XQuery language itself.

## Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# There is no model in which Joe is not an Employee




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

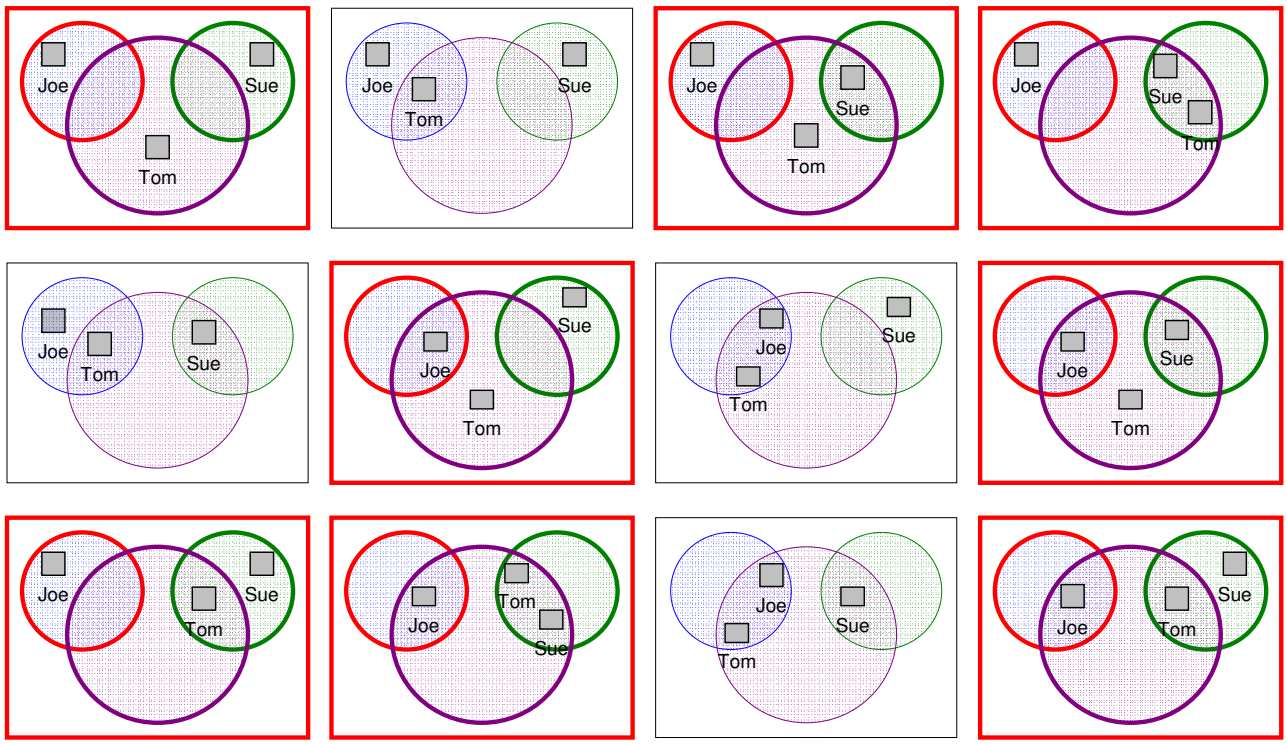
Notice that the class structure remains the same for every interpretation it is the individuals that create the different models.

## Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Tom is not an Employee in some models and is an Employee in some models



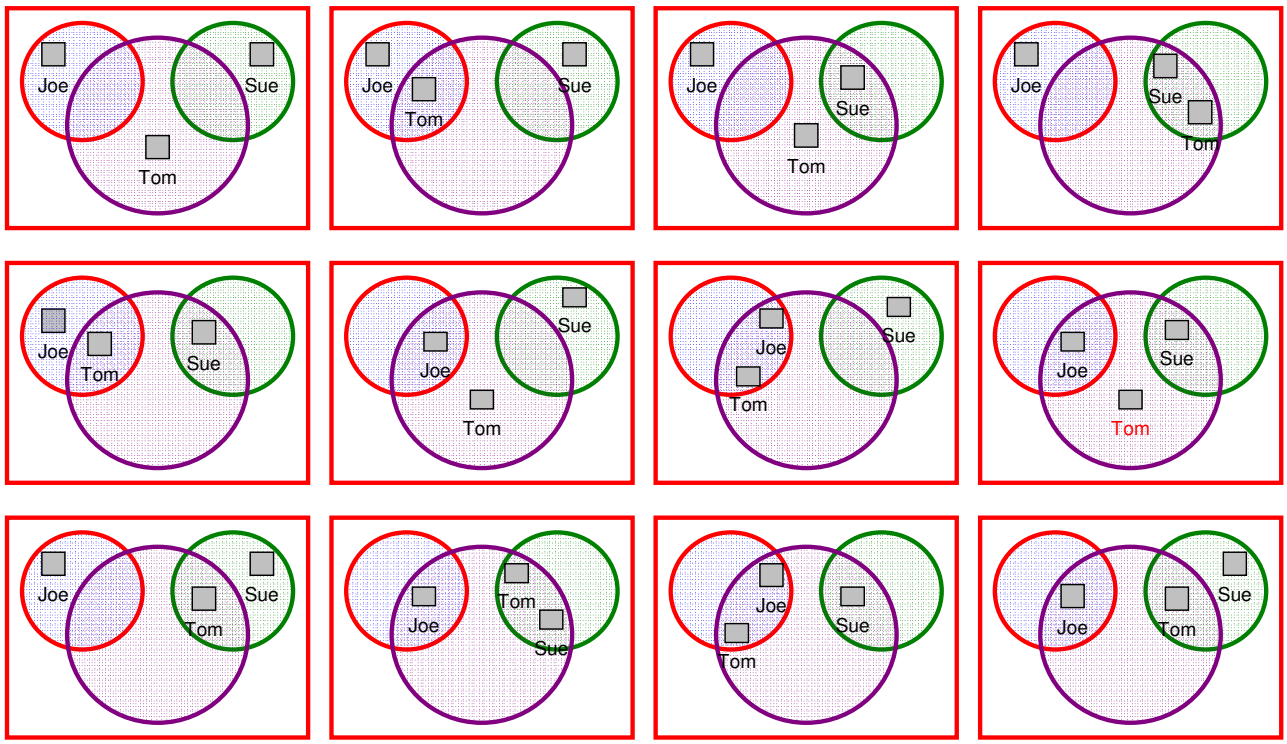

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# In every model Sue is not an Employee




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Understanding a provably not query

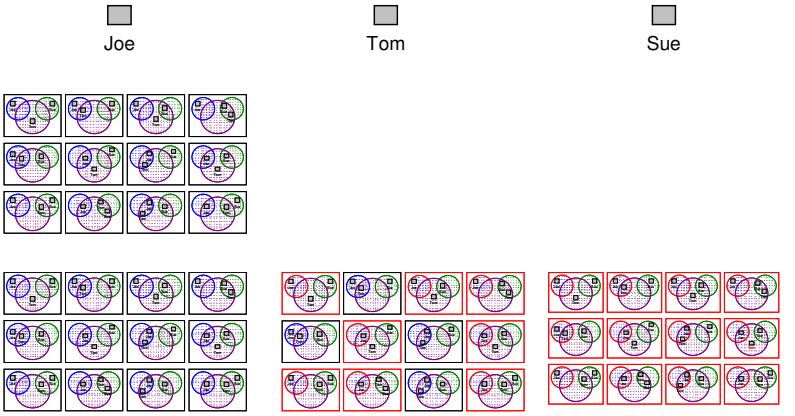
```
for $i in instances()
where every $m in models() satisfies
    not( member($i, Employee, $m) )
return  {$i}
```

For each individual in the ontology

For each possible model allowed by the ontology

Select any individual that meets the criteria (is not an Employee) in every model.

Return some information about each selected individual.




http://www.cerebra.com/construct#**Sue**

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Question : What result would we get if we asked for individuals who were satisfiably NOT employees. Warning: the not function is not a complement operator There is a difference between saying:

- Everything that is not an employee and
- Everything that is provably not an employee

More on this later.

## Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Open World Queries

- If the criteria are provably true for an individual they will also be satisfiably true.
  - e.g., Joe is provably an employee so he is also satisfiably an employee.
- If the criteria are provably false for the individual they will also be satisfiably false.
  - e.g., Sue is provably not an employee so she is also satisfiably not an employee.
- Every model has to conform to all the assertions in the ontology. Anything that is explicitly asserted in the ontology will be true in all models.
  - e.g., If I declare Joe to be an employee explicitly he will be an employee in every model.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

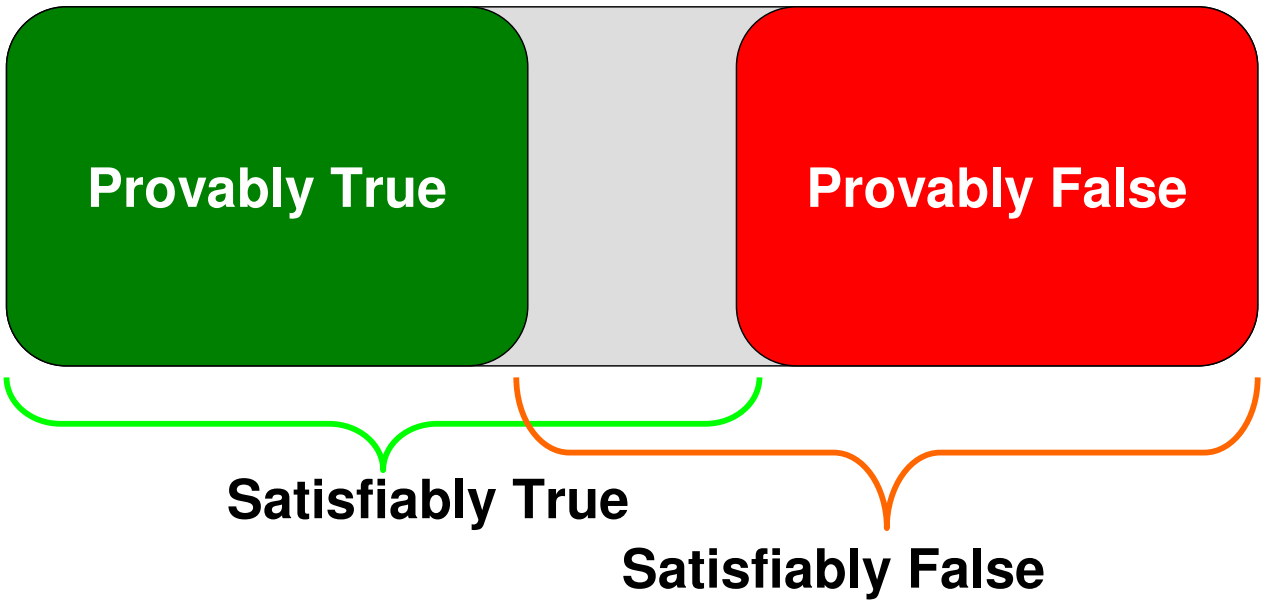
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Provable vs. Satisfiable

*(every)* *(some)*




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

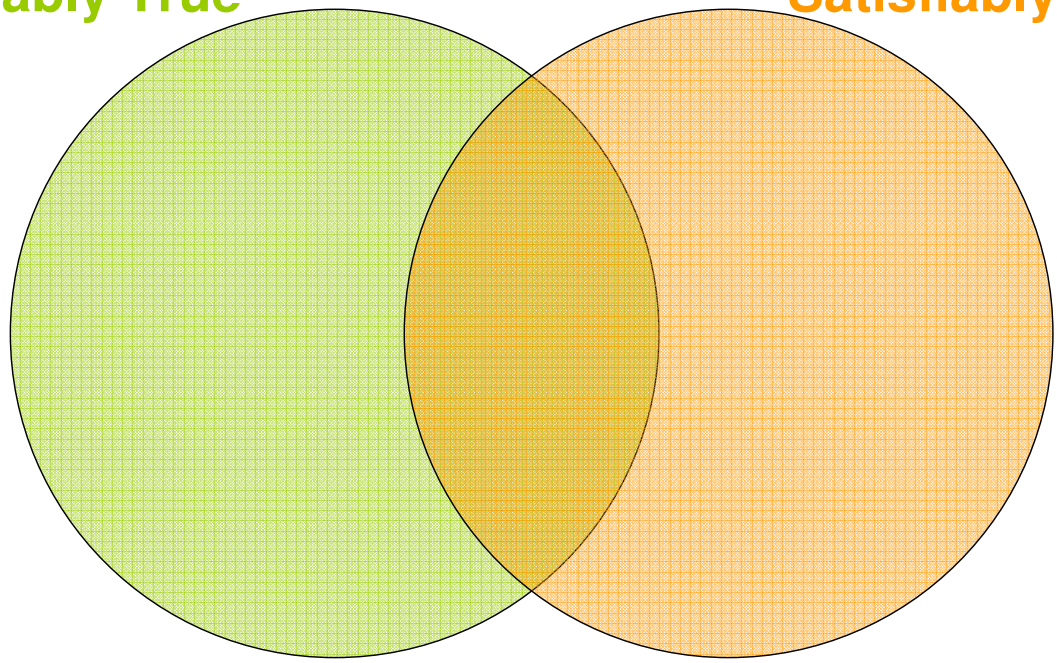
Cerebra UNIVERSITY

CU-003
02/2006
Module 2

*(every)* *(some)*

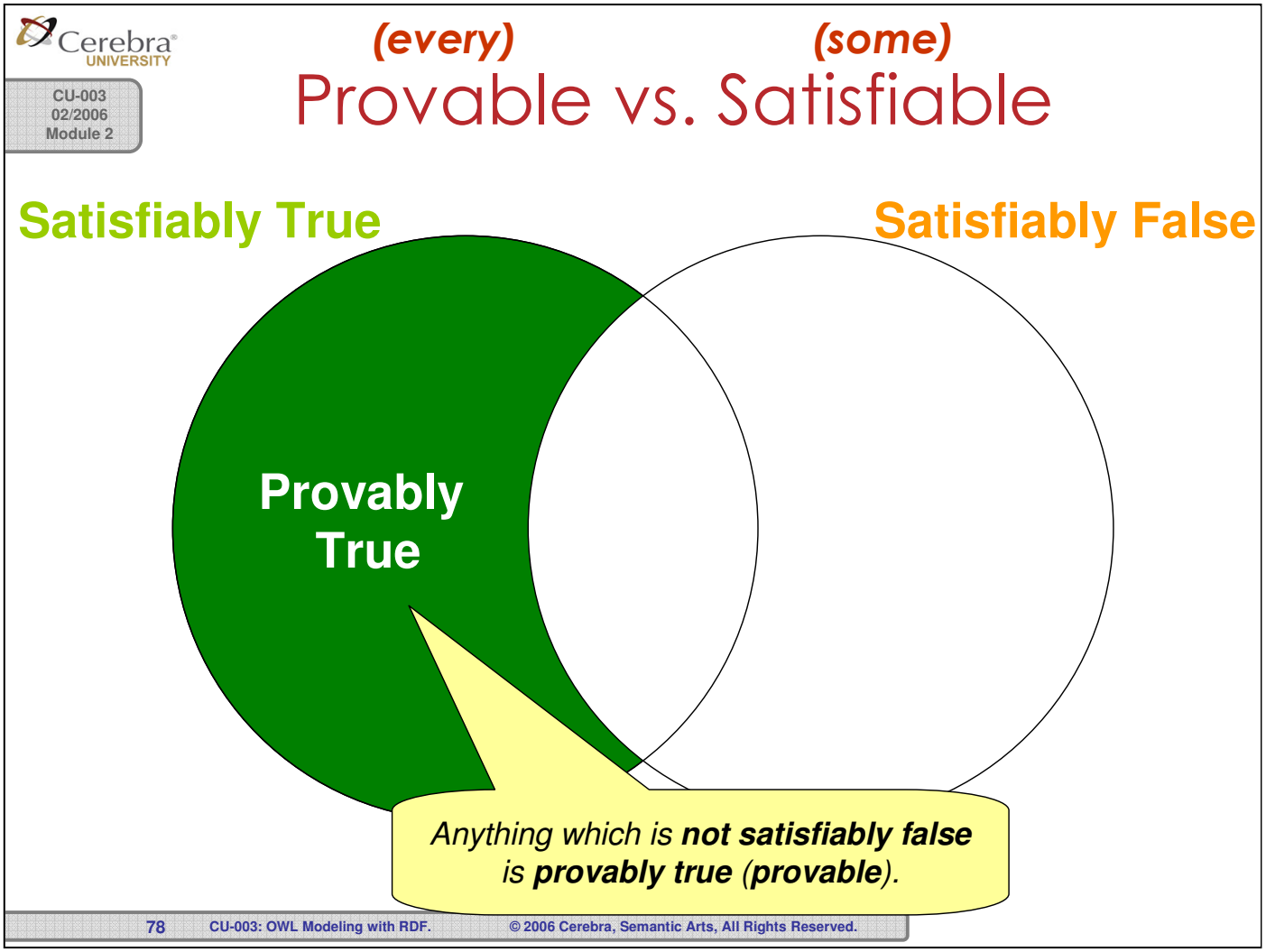
# Provable vs. Satisfiable

**Satisfiably True** **Satisfiably False**



 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

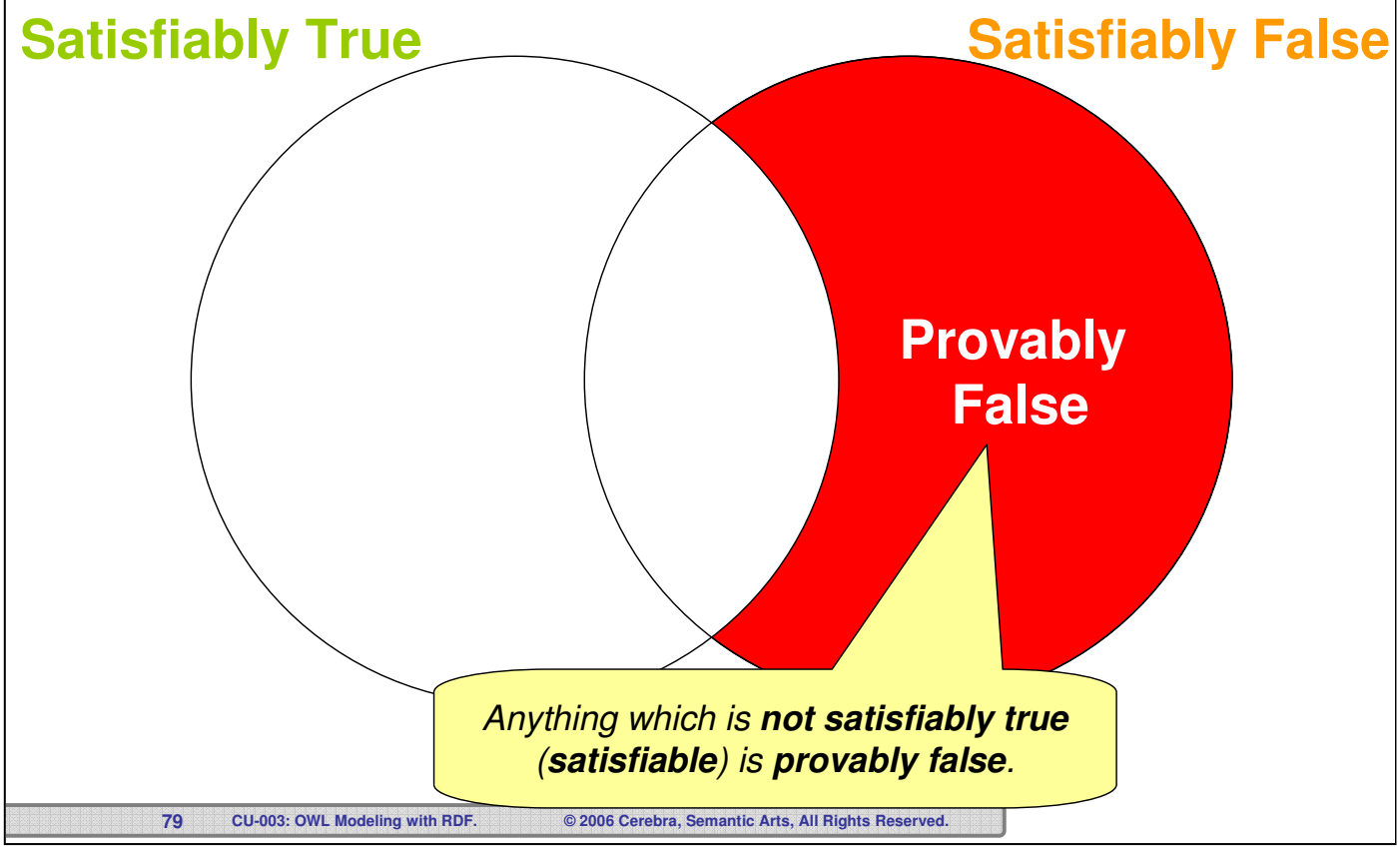



Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# *(every)* *(some)* Provable vs. Satisfiable





© 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# This is certainly confusing, but is it useful?

- The ability to write satisfiable queries allows us to query what *might or may be* the case.
  - Which workers are candidates for our stock incentive plan?
  - If a patient has these symptoms, what are the possible diagnoses?
  - Given what we know about the claim at this point, what is our potential liability?
  - Given what we know about our patients, which are potential candidates for this clinical trial?
  - What diagnostic test should we run to narrow down this electrical problem?
- Whenever we want to ask questions about a situation in which we don't have all the information, or want to ask what-if type questions, then open world reasoning is a powerful tool.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Exercise 3

- In your *Travel* ontology make the *FlightRoute* and *Airport* classes distinct.
- Query the ontology for satisfiable *FlightRoute*s. (i.e., individuals which are *FlightRoute*s in **some** models.)
  - Member($i,FlightRoute,$m)

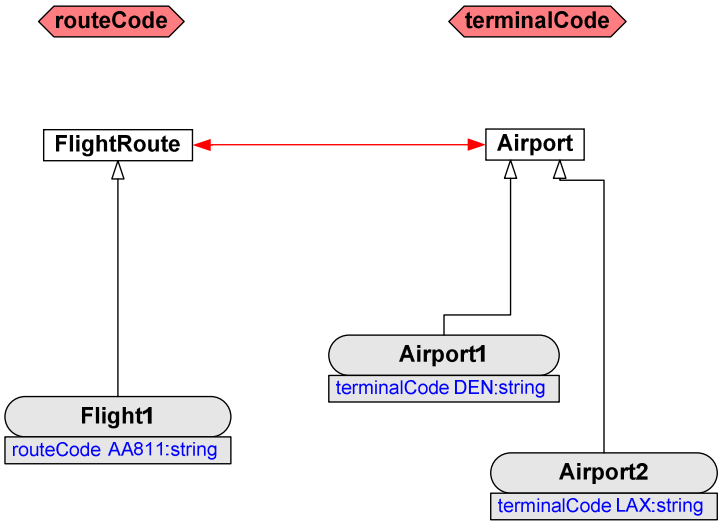
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Exercise 3



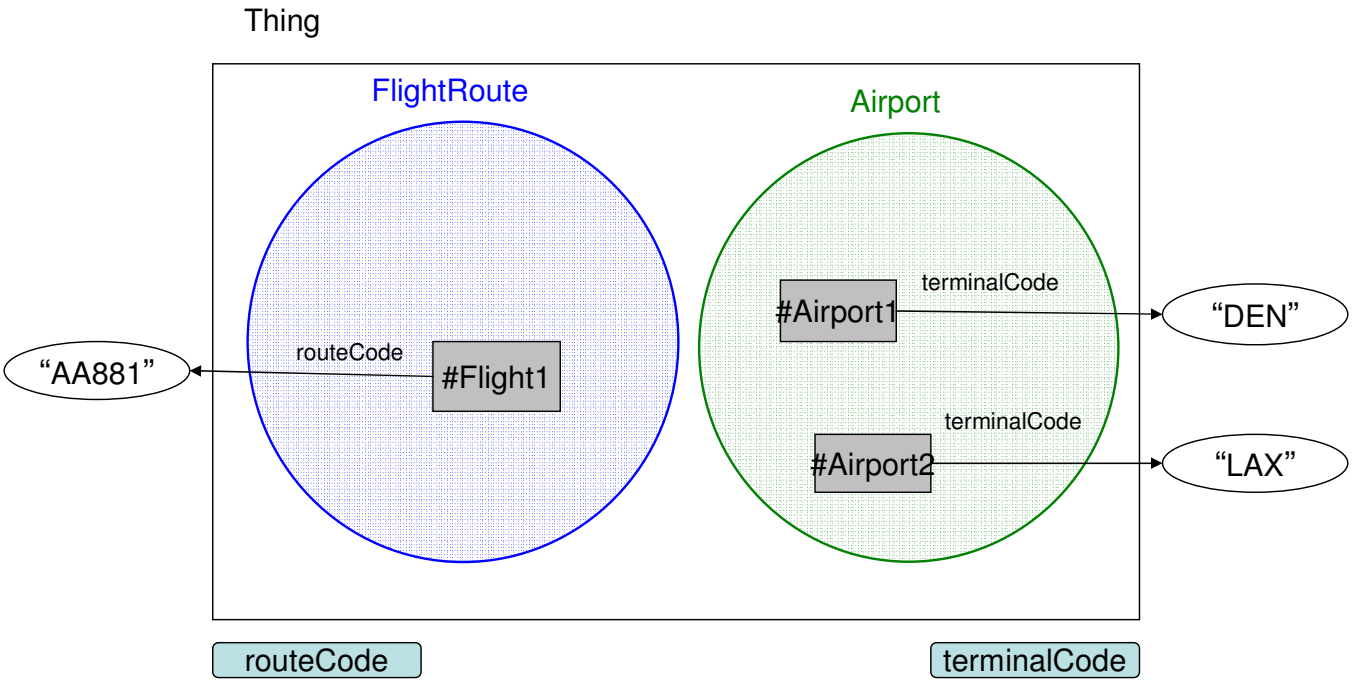

```
<owl:Class rdf:ID="FlightRoute">
   <owl:disjointWith rdf:resource="#Airport"/>
</owl:Class>

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

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# What we have asserted



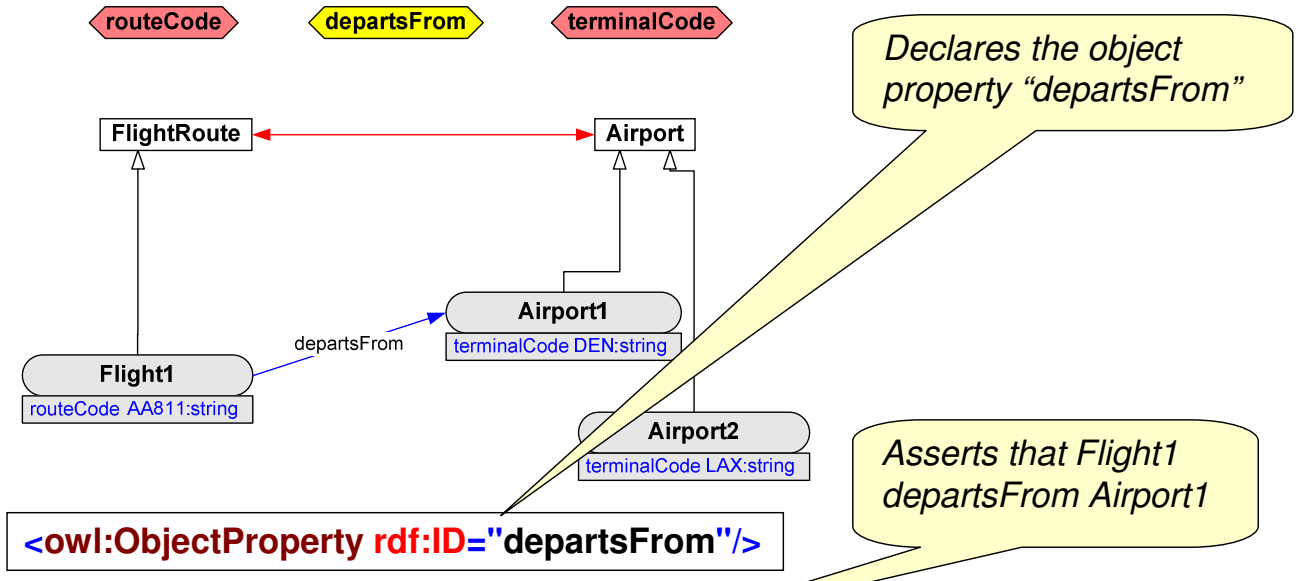

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# An Object Property



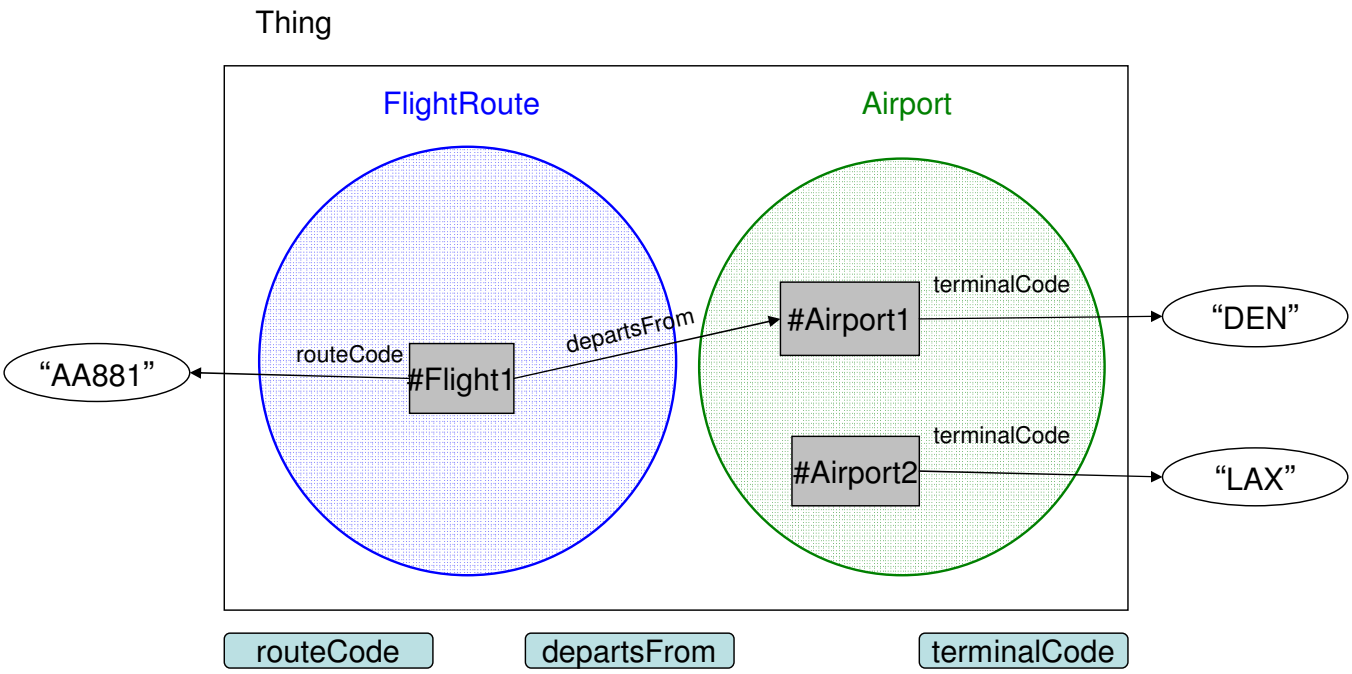

```
<owl:ObjectProperty rdf:ID="departsFrom"/>

<sa:FlightRoute rdf:ID="Flight1">
  <sa:departsFrom rdf:resource="#Airport1"/>
    <sa:routeCode rdf:datatype="http://....#string">AA811</sa:routeCode>
</sa:FlightRoute>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# What we have asserted




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Object Properties – The Essentials

- Object properties associate one individual with another (as opposed to data type properties that associate an individual and a literal).
- Properties are not dependent on classes. For instance, we could use *departsFrom* to describe the relationship between *BusRoutes* and *BusTerminals* elsewhere in the ontology.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Exercise 4

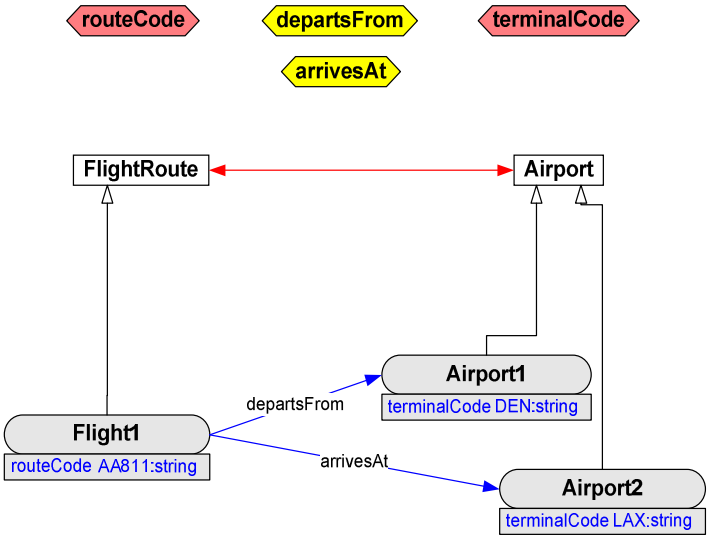
- Add an object property *arrivesAt* to the ontology.
- Assert that *Flight1 arrivesAt Airport2*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Note that while you're hooking up properties if they stuck you'll get a dialog box to pick the property. No dialog box = arrow isn't attached. Show in model tree

Student Notes:

# Exercise 4



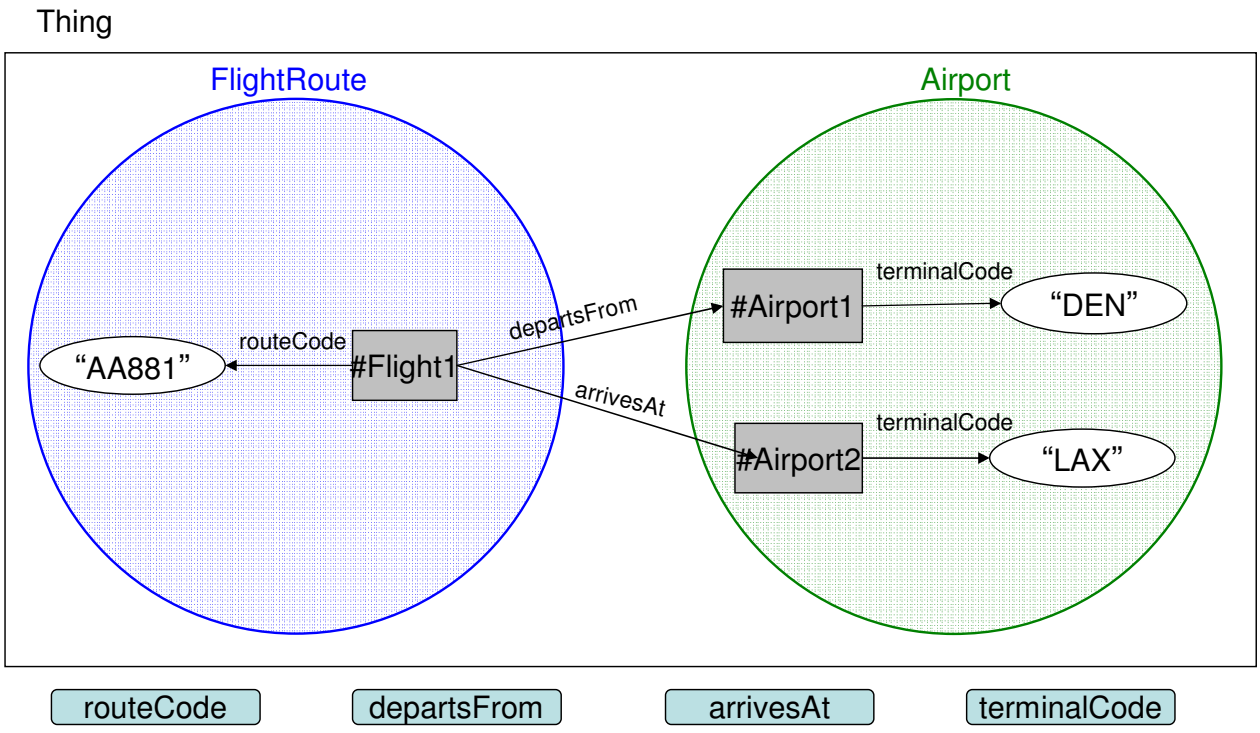

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# What we have asserted




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Exercise 5

Expand the ontology to include

- Another instance of the class Airport
  - URI Airport3
  - terminalCode "BOS"
- Another instance of the class FlightRoute
  - URI Flight2
  - routeCode "UA657"
  - departsFrom Airport2
  - arrivesAt Airport3

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# More Ontology Querying

A more in-depth look at how Cerebra Server executes a conjunctive query on an ontology

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# More Ontology Querying

- This section will introduce two new Cerebra functions, which are used to query the relationships between individuals.
  - related()
  - matches()
- We will do a detailed walkthrough of how Cerebra Server executes a query.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

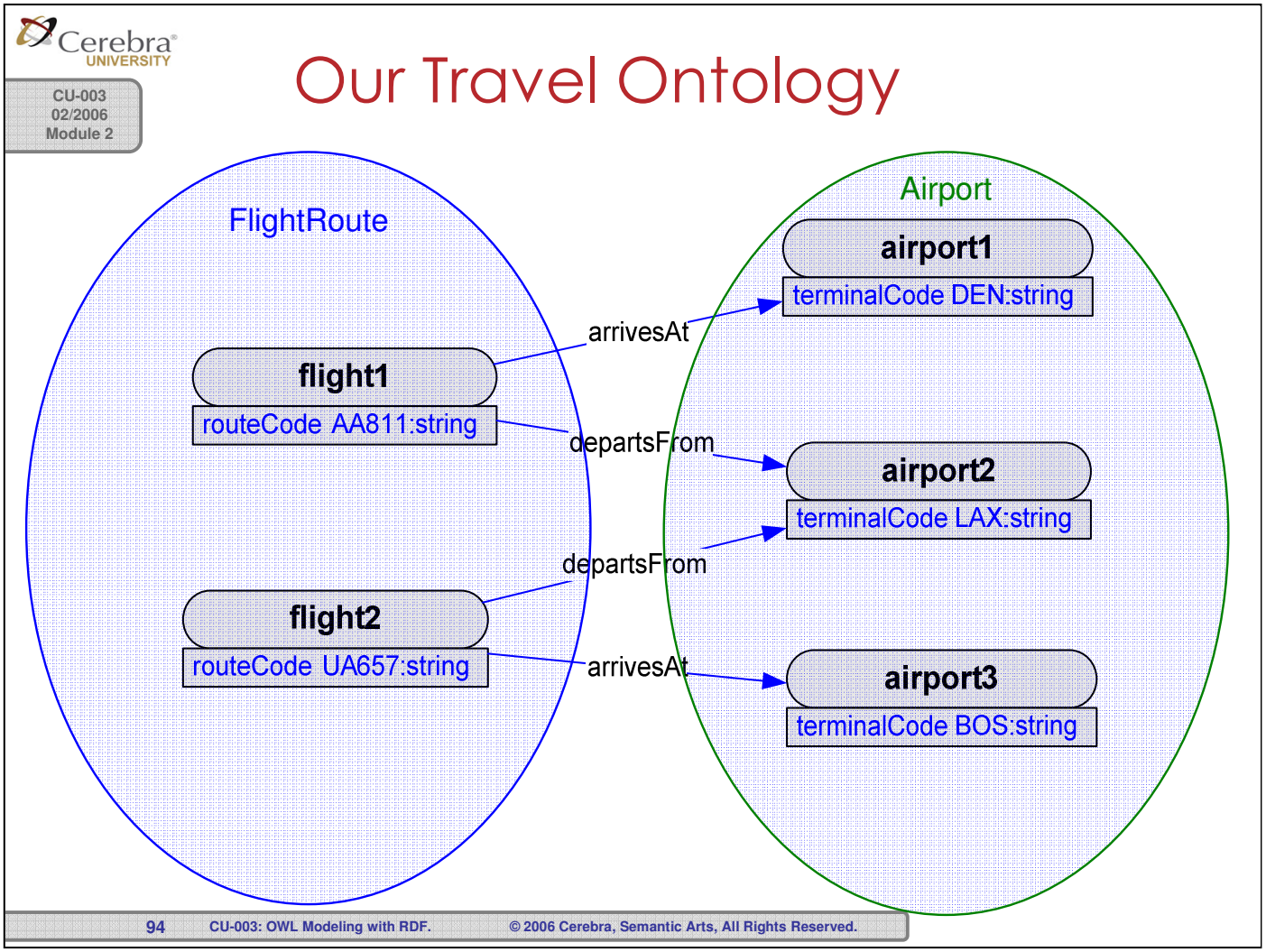
# The *related* Function

related(*individual1*,*property*,*individual2*,*model*)

- The *related* function checks to see if *individual1* and *individual2* are related by *property* in the given model.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:




Student Notes:

# A Simple Query

*Which flights arrive at which airports?*

```
declare default function namespace
        "http://www.networkinference.com/cerebra/functions/3.1"
declare default element namespace
        "http://www.semanticarts.com/ontologies/travel"

for $a in instances()
for $f in instances()

where every $m in models() satisfies

  member($a,Airport,$m)
  and member($f,Flight,$m)
  and related($f,arrivesAt,$a,$m)

return <Airport id='{$a}'>{$f}</Airport>
```

*$a is an Airport, $f is a Flight, and $f arrives at $a.*
*This will give us every airport and the flights that arrive at it.*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# The *for* Clause

*Which flights arrive at which airports?*

```
declare default function namespace
        "http://www.networkinference.com/cerebra/functions/3.1"
declare default element namespace
        "http://www.semanticarts.com/ontologies/travel"

for $a in instances()
for $f in instances()

where every $m in models() satisfies

  member($a,Airport,$m)
  and member($f,Flight,$m)
  and related($f,arrivesAt,$a,$m)

return <Airport id='{$a}'>{$f}</Airport>
```

*The for clause defines an iterative loop. The variable $a will be bound in turn to every value returned by the instances() function, which returns every individual in the ontology.*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# The *for* Clause

- The instances() function returns every individual in the ontology. $a will have the following values:
- The query will be run once for each value.

| $a |
|---|
| flight1 |
| flight2 |
| airport1 |
| airport2 |
| airport3 |

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# The *for* Clause

*Which flights arrive at which airports?*

```
declare default function namespace
        "http://www.networkinference.com/cerebra/functions/3.1"
declare default element namespace
        "http://www.semanticarts.com/ontologies/travel"

for $a in instances()
for $f in instances()

where every $m in models() satisfies

  member($a,Airport,$m)
  and member($f,Flight,$m)
  and related($f,arrivesAt,$a,$m)

return <Airport id='{$a}'>{$f}</Airport>
```

*A second variable $f will be bound in turn to each individual in the ontology. The query will be run with every potential pair of individuals.*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# The *for* Clause

- This gives us the following possible bindings for our variables:

| $a | $f |
|---|---|
| flight1 | flight1 |
| flight1 | flight2 |
| flight1 | airport1 |
| flight1 | airport2 |
| flight1 | airport3 |

| $a | $f |
|---|---|
| flight2 | flight1 |
| flight2 | flight2 |
| flight2 | airport1 |
| flight2 | airport2 |
| flight2 | airport3 |

| $a | $f |
|---|---|
| airport1 | flight1 |
| airport1 | flight2 |
| airport1 | airport1 |
| airport1 | airport2 |
| airport1 | airport3 |

| $a | $f |
|---|---|
| airport2 | flight1 |
| airport2 | flight2 |
| airport2 | airport1 |
| airport2 | airport2 |
| airport2 | airport3 |

| $a | $f |
|---|---|
| airport3 | flight1 |
| airport3 | flight2 |
| airport3 | airport1 |
| airport3 | airport2 |
| airport3 | airport3 |

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# The *where* Clause

*Which flights arrive at which airports?*

```
declare default function namespace
        "http://www.networkinference.com/cerebra/functions/3.1"
declare default element namespace
        "http://www.semanticarts.com/ontologies/travel"

for $a in instances()
for $f in instances()

where every $m in models() satisfies

  member($a,Airport,$m) and
  member($f,Flight,$m) and
  related($f,arrivesAt,$a,$m)

return <Airport id='{$a}'>{$f}</Airport>
```

*Because of how we created our ontology, there is only one possible model. When there is only one model, a satisfiable query and a provable query will produce identical results.*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# The *where* Clause

*Which flights arrive at which airports?*

```
declare default function namespace
        "http://www.networkinference.com/cerebra/functions/3.1"
declare default element namespace
        "http://www.semanticarts.com/ontologies/travel"

for $a in instances()
for $f in instances()

where every $m in models() satisfies

  member($a,Airport,$m)
  and member($f,Flight,$m)
  and related($f,arrivesAt,$a,$m)

return <Airport id='{$a}'>{$f}</Airport>
```

*We will now test each of our variable bindings, and filter out all but those that match these criteria.*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# The *where* Clause

- We will now test each of our variable bindings, and filter out all but those that satisfy our criteria:

| $a | $f |
|---|---|
| flight1 | flight1 |
| flight1 | flight2 |
| flight1 | airport1 |
| flight1 | airport2 |
| flight1 | airport3 |

| $a | $f |
|---|---|
| flight2 | flight1 |
| flight2 | flight2 |
| flight2 | airport1 |
| flight2 | airport2 |
| flight2 | airport3 |

| $a | $f |
|---|---|
| airport1 | flight1 |
| airport1 | flight2 |
| airport1 | airport1 |
| airport1 | airport2 |
| airport1 | airport3 |

| $a | $f |
|---|---|
| airport2 | flight1 |
| airport2 | flight2 |
| airport2 | airport1 |
| airport2 | airport2 |
| airport2 | airport3 |

| $a | $f |
|---|---|
| airport3 | flight1 |
| airport3 | flight2 |
| airport3 | airport1 |
| airport3 | airport2 |
| airport3 | airport3 |

**member($a,Airport,$m)**
**and member($f,Flight,$m)**
**and related($f,arrivesAt,$a,$m)**

**This leaves us two sets of bindings:**

**$a = airport1, $f = flight1**
***– and –***
**$a = airport3, $f = flight2**

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# The *return* Clause

*Which flights arrive at which airports?*

```
declare default function namespace
        "http://www.networkinference.com/cerebra/functions/3.1"
declare default element namespace
        "http://www.semanticarts.com/ontologies/travel"

for $a in instances()
for $f in instances()

where every $m in models() satisfies

  member($a,Airport,$m)
  and member($f,Flight,$m)
  and related($f,arrivesAt,$a,$m)

return <Airport id='{$a}'>{$f}</Airport>
```

*For every valid set of bindings, this snippet of XML will be added to the output. (Remember that putting curly braces around a variable name evaluates to its fully qualified URI.)*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# The *return* Clause

*Which flights arrive at which airports?*

```
<RESULT>
  <Airport id='http://www.semanticarts.com/frequentflier#airport1'>
    http://www.semanticarts.com/ontologies/travel#flight1
  </Airport>
  <Airport id='http://www.semanticarts.com/frequentflier#airport3'>
    http://www.semanticarts.com/ontologies/travel#flight2
  </Airport>
</RESULT>
```

*Unfortunately, this result is not very useful. Unless we remember which airport is referred to as* "airport1"*, for example, this doesn't tell us much. We need a way to include the* datatype properties *of these individuals in our query result.*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# The *related* Function (alternate definition)

related(*individual,property,datavalue,model)*

- While the previous definition checked for object properties, the *related* function can also be used to check for datatype properties.
- This usage checks to see if *individual* has the datatype property *property*, with *datavalue* as its value in the given model.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Querying for Datatype Properties

*Which flights arrive at which airports?*

```
for $a in instances()
for $f in instances()
for $n in datavalues()
for $c in datavalues()

where every $m in models() satisfies

  member($a,Airport,$m)
  and member($f,Flight,$m)
  and related($f,arrivesAt,$a,$m)
  and related($f,routeCode,$n,$m)
  and related($c,terminalCode,$c,$m)

return <Airport id='{$c}'>{$n}</Airport>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Querying for Datatype Properties

*Which flights arrive at which airports?*

```
for $a in instances()
for $f in instances()
for $n in datavalues()
for $c in datavalues()

where every $m in models() satisfies

  member($a,Airport,$m)
  and member($f,Flight,$m)
  and related($f,arrivesAt,$a,$m)
  and related($f,routeCode,$n,$m)
  and related($c,terminalCode,$c,$m)

return <Airport id='{$c}'>{$n}</Airport>
```

*The datavalues() function binds a variable to the set of all data values in the ontology.*
*In this case the "for" causes the query to iterate through every data value in the ontology.*


© 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Querying for Datatype Properties

*Which flights arrive at which airports?*

```
for $a in instances()
for $f in instances()
for $n in datavalues()
for $c in datavalues()

where every $m in models() satisfies

  member($a,Airport,$m)
  and member($f,Flight,$m)
  and related($f,arrivesAt,$a,$m)
  and related($f,routeCode,$n,$m)
  and related($a,terminalCode,$c,$m)

return <Airport id='{$c}'>{$n}</Airport>
```

*$f has the routeCode $n, and $a has the terminalCode $c. Using datavalues allows us to work with strings and numbers, rather than with RDF URIs.*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

For all instances that are airports and
- all instances that are flights flights that arrive at airports
- filter for the arrival airports
- find data value properties for the route code on a flight
- find data value properties for the terminal code on that arrival flight

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Querying for Datatype Properties

*Which flights arrive at which airports?*

```
for $a in instances()
for $f in instances()
for $n in datavalues()
for $c in datavalues()

where every $m in models() satisfies

  member($a,Airport,$m)
  and member($f,Flight,$m)
  and related($f,arrivesAt,$a,$m)
  and related($f,routeCode,$n,$m)
  and related($a,terminalCode,$c,$m)

return <Airport id='{$c}'>{$n}</Airport>
```

*Instead of asking for the URIs of the individuals returned, we are asking for the literal values of their datatype properties.*
*{$c} and {$n} evaluate to a literal string.*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Querying for Datatype Properties

*Which flights arrive at which airports?*

```
for $a in instances()
for $f in instances()
for $n in datavalues()
for $c in datavalues()

where every $m in models() satisfies

  member($a,Airport,$m)
  and member($f,Flight,$m)
  and related($f,arrivesAt,$a,$m)
  and related($f,routeCode,$n,$m)
  and related($a,terminalCode,$c,$m)

return <Airport id='{$c}'>{$n}</Airport>
```

*Putting curly braces around data value variables evaluates to their literal value. (In this case, their literal string value.)*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Querying for Datatype Properties

*Which flights arrive at which airports?*

```
<RESULT>
   <Airport id='DEN'>AA811</Airport>
   <Airport id='BOS'>UA657</Airport>
</RESULT>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Querying for a Specific Individual

- So far our query has been: "Which flights arrive at which airports?"
- What if we want to ask a question about a specific individual?
- We can always refer to an individual by name in a query, but this requires that we know its URI.
- How would we ask the question: "Which flights depart from LAX?"
  - i.e., an *Airport* with the *terminalCode* "LAX"

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# The *matches* Function

matches(*datavalue,datatype,matchstring)*

- The *matches* function is used to perform string and numerical comparisons on a data value.
- The *matches* function can be used to match a data value to a string or list of strings.
- It can also be used to match a data value to a number, a range of numbers, or multiple ranges.
- See the appendix for a syntax reference.
- A full reference on possible match strings is in the *Cerebra Server Developer's Guide*.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Querying for a Specific Individual

*Which flights depart from LAX?*

```
for $a in instances()
for $f in instances()
for $n in datavalues()
for $c in datavalues()

where every $m in models() satisfies

  member($a,Airport,$m)
  and member($f,Flight,$m)
  and related($f,departsFrom,$a,$m)
  and related($f,routeCode,$n,$m)
  and related($a,terminalCode,$c,$m)
  and matches($c,"xs:string","LAX")

return <Flight>{$n}</Flight>
```

*We have confirmed that $c is the terminalCode for $a, which is an Airport.*
*Here we are checking if the data value represented by $c is a string, and matches the string "LAX".*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

In the return string we don't need $C anymore as we are only interested in the airport with a terminal code of LAX.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Querying for a Specific Individual

*Which flights depart from LAX?*

```
<RESULT>
   <Flight>AA811</Flight>
   <Flight>UA657</Flight>
</RESULT>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Common Query Gotchas

- Forgetting to save and resubmit your ontology.
- Forgetting to set your default namespaces.
- Forgetting the AND/OR operator after a function call.
- Querying an inconsistent ontology. (Run the function isConsistent() to check this.)

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Exercise 4

- Query your travel ontology for a *FlightRoute* that departs from DEN and arrives at LAX.
  - *departsFrom* an *Airport* with the *terminalCode* "DEN"
  - *arrivesAt* an *Airport* with the *terminalCode* "LAX"

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 2

# Module Review

- Instructor led review session
- Pop-quiz is optional!

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# OWL-DL Modeling with RDF

## Module Three: Equivalence, Disjointness, and Subsumption

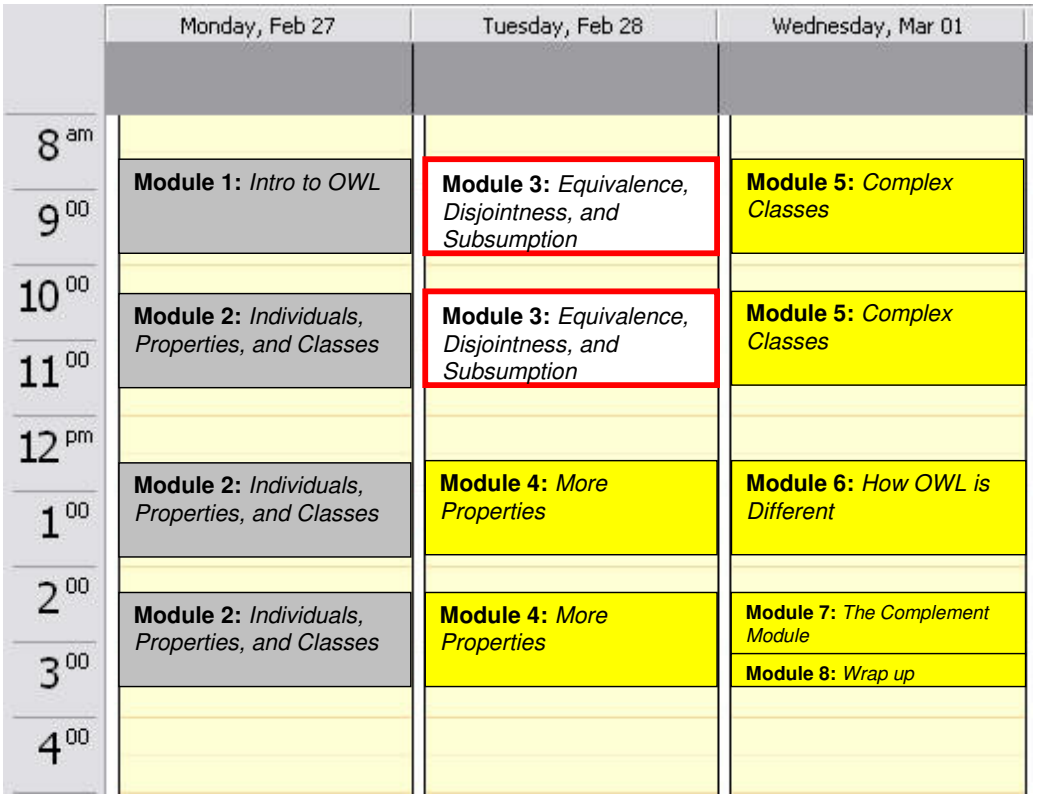
CU-003: OWL Modeling with RDF. © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Agenda




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Inferred Class Membership

- We have already seen that we can declare individuals and make them members of classes.
- However, the real power of OWL lies in the fact that we can also *infer* class membership based on what we know.
- Property relationships can also be inferred.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Construct Diagrams

- You have now had some hands-on experience working with Cerebra Construct.
- Construct provides a graphical view of an ontology, but *it only shows those facts which are **explicitly** declared to be true*.
- In order to represent ***inferred*** information, we will combine Construct diagrams with a simple graphical convention.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Provable or Satisfiable?

- Recall that in an open-world model, anything is possible as long as it is consistent with the ontology's assertions.
- Therefore, the inference engine actually has two types of inferred facts: those which are **known to be true**, and those which are **known to be false**.
- Any other fact or relationship is possible or **satisfiable**, that is, it is true in some models and false in others.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Adding Inferred Information

- Explicitly declared facts will be shown on the diagram just as they are drawn in Construct.
- Facts which are inferred to be **true** will be shown in green:




- Facts which are inferred to be **false** will be shown in red:



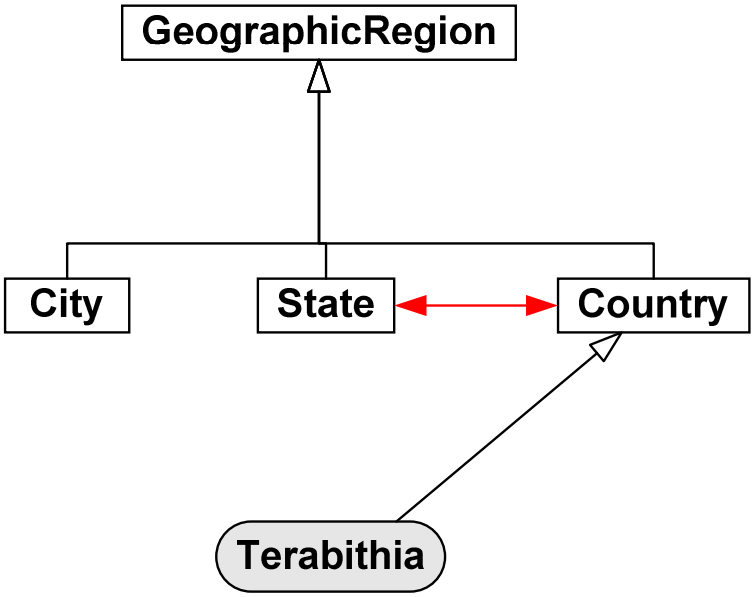

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Example




*Don't worry if you don't understand everything in this example. Just focus on what the various symbols mean. (The specifics will be explained in the upcoming section.)*

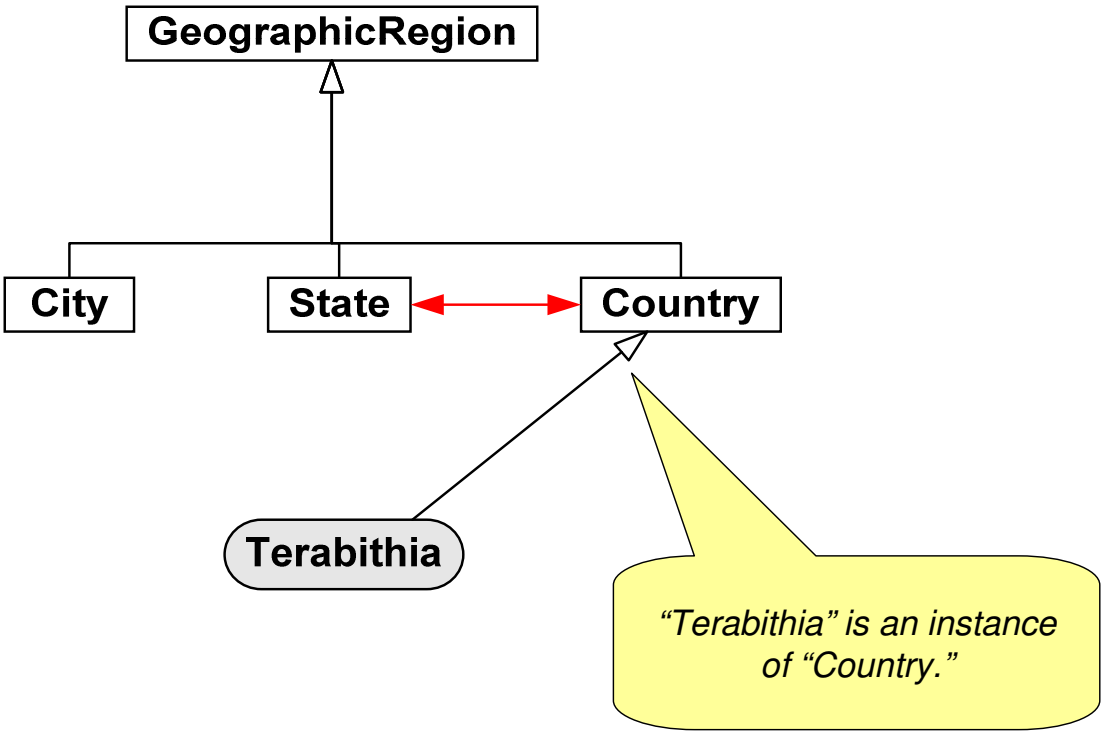
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Declared Facts



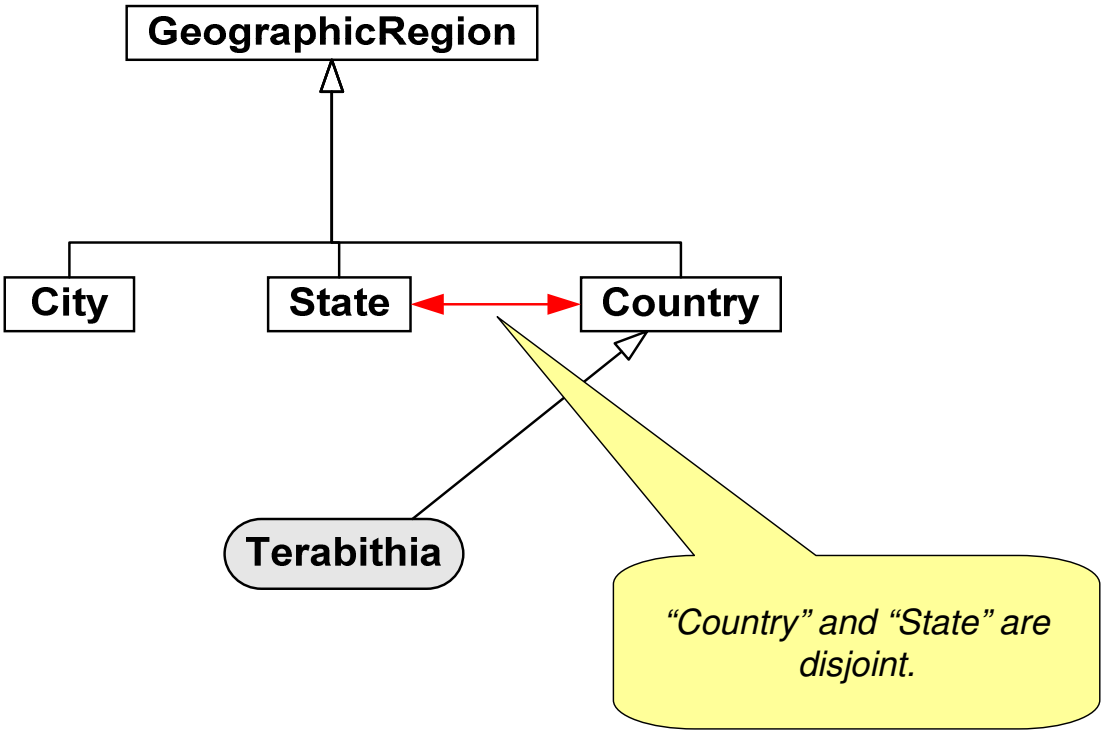

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Declared Facts



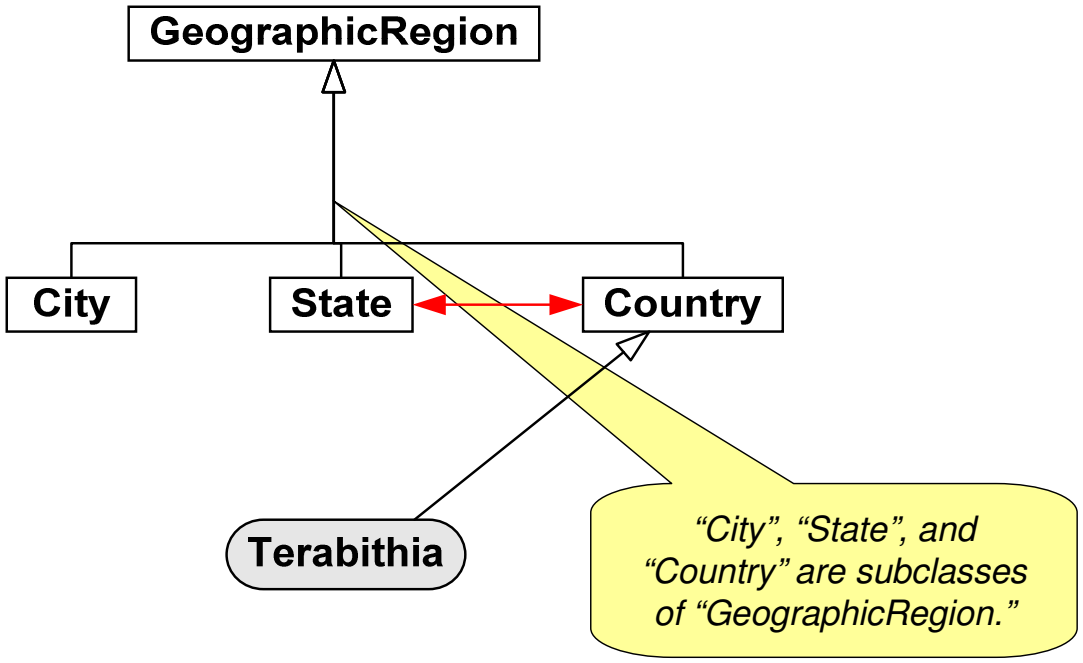

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Declared Facts



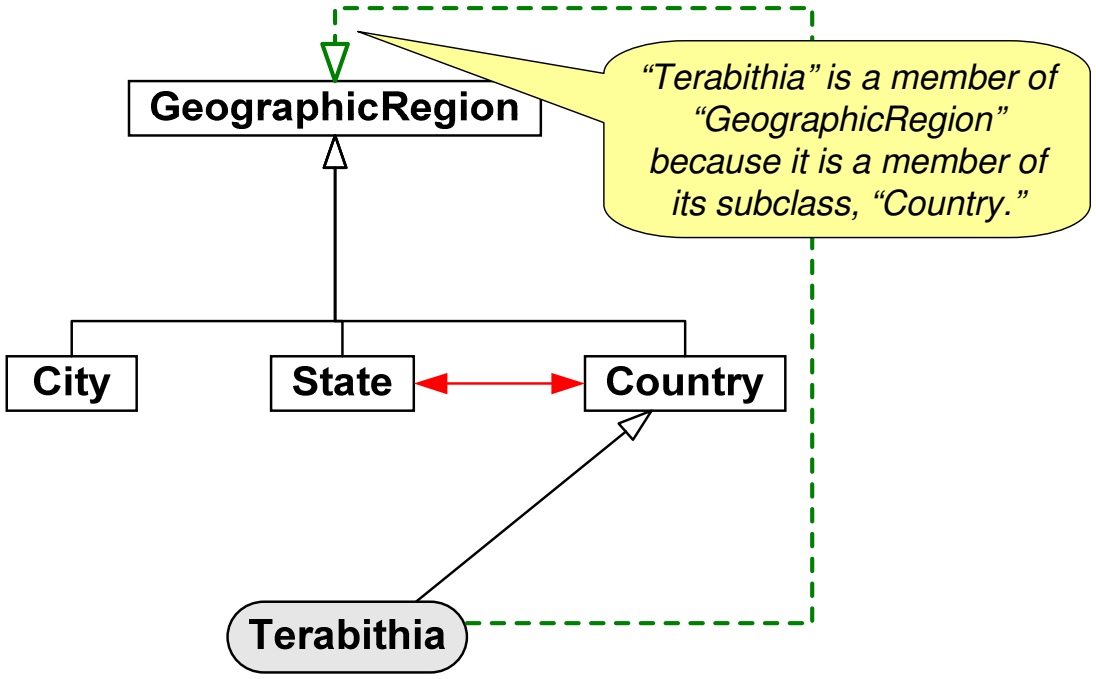

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

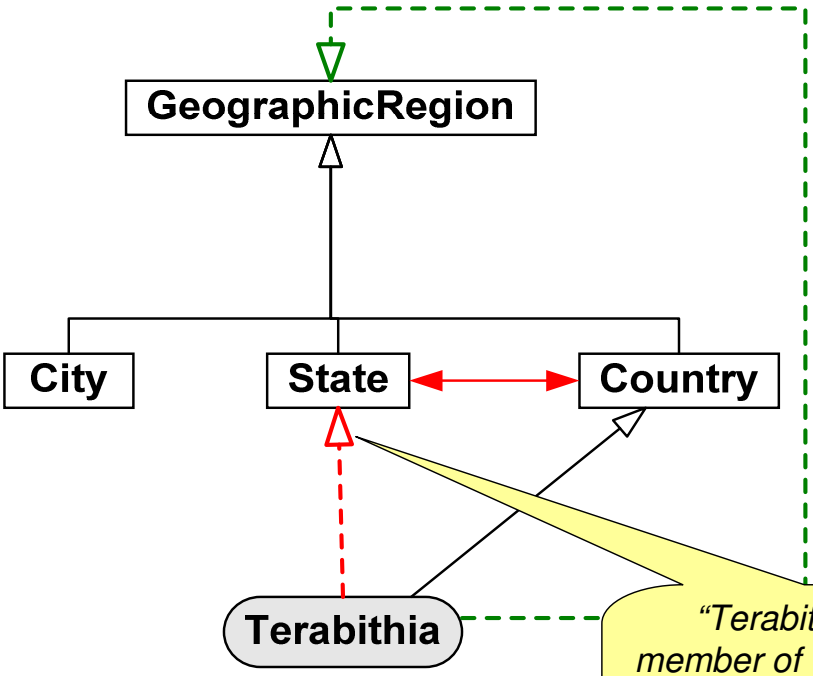
# Inferred Facts




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Inferred Facts



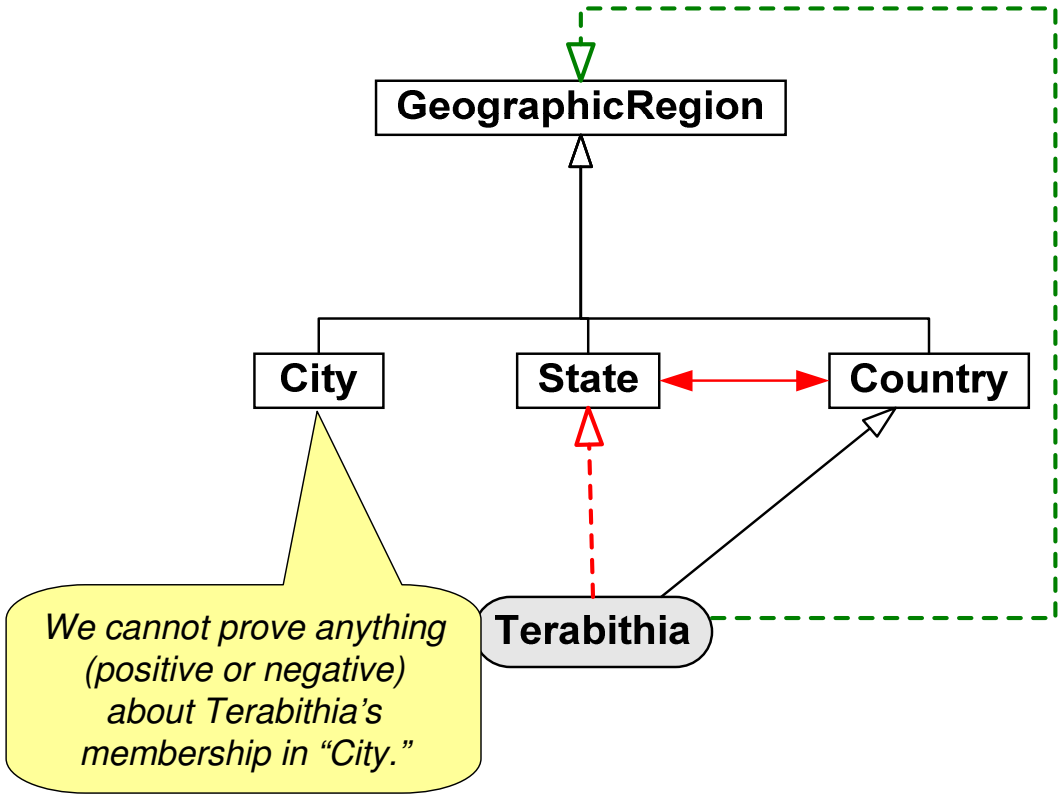

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Satisfiability



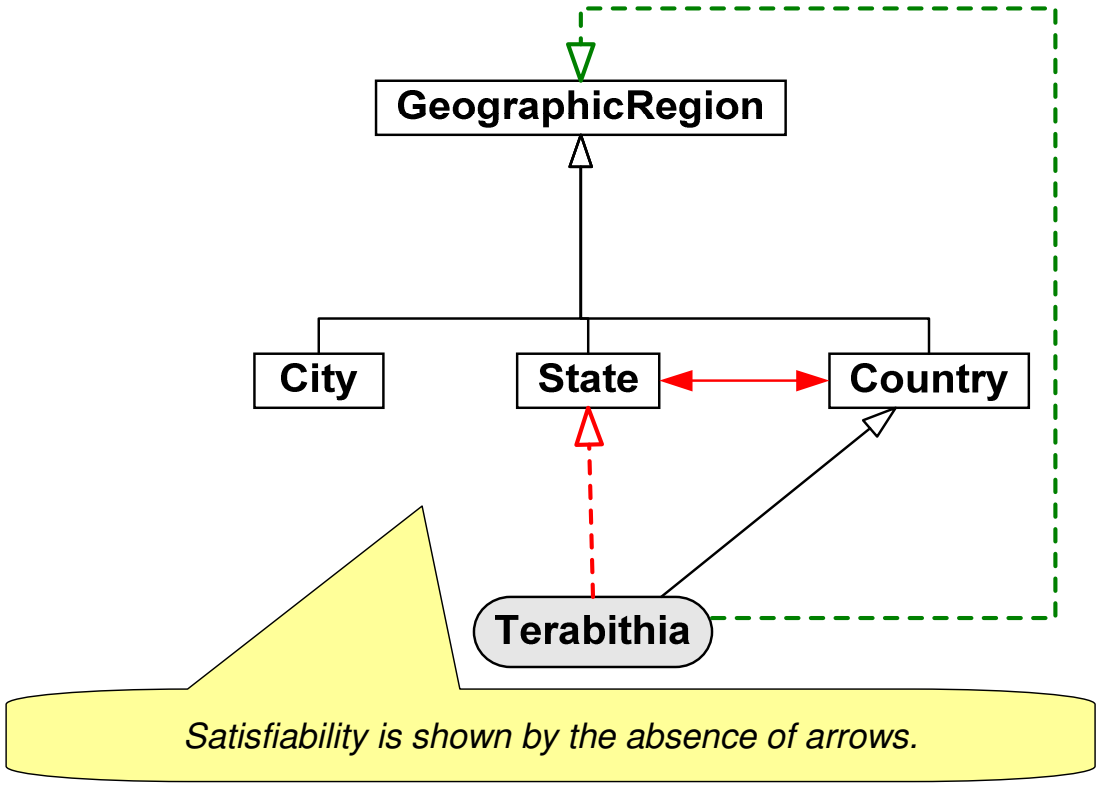

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Satisfiability



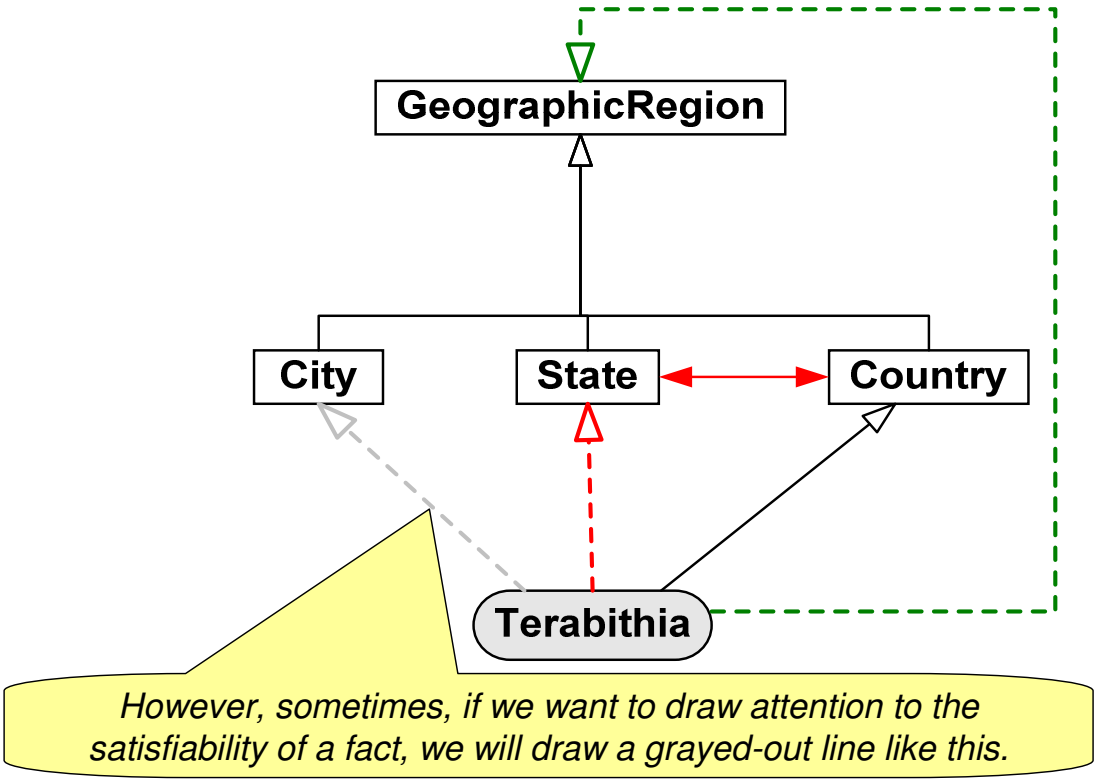

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Satisfiability




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Showing Inferred Facts

- Remember that the dotted lines will not appear on your Construct diagram.
- They represent the additional information which will be inferred when your ontology is analyzed by the reasoner.
- We will use these arrows to show how additional facts are inferred based on what is already known.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Equivalence, Disjointness, and Subsumption

- This section will introduce three important concepts in ontology modeling.
- As a demonstration, we will build a simple geographic ontology.
- In the exercise, you will continue to expand your travel ontology using these concepts.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# The Geography Ontology

- To demonstrate the concepts in this module, we will build a simple geography ontology.
- We will begin by defining some basic classes. Then we will expand the ontology to model the relationships between them.

**Country** **State** **Province** **City**

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Exercise One – Setup

- For the class exercises, you will expand your "Flights and Airports" ontology into a "Travel" ontology.
- Open the Construct file:
  **Module3_StartingOntology.vsd**

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Exercise One – Analysis

- This file contains the ontology from the previous module, with a few differences:
  - *FlightRoute* and *Airport* are no longer disjoint.
  - The *terminalCode* datatype property has been replaced with *airportCode*.
  - The *routeCode* datatype property has been replaced with *flightNumber*.
- The reasons for these changes will become clear during the exercises.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Exercise One

- Define the classes *BusRoute* and *TrainRoute* in your ontology.
- Define the classes *BusStation* and *TrainStation*.
- Define the datatype property *stationCode*.
- Define the datatype properties *busRouteNumber* and *trainRouteID*.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Exercise One

- Define an individual *busStation1*.
  - Make it an instance of BusStation.
  - Give it the *stationCode* "LAX".
- Define an individual *busStation2*.
  - Make it an instance of BusStation.
  - Give it the *stationCode* "DEB".
- Define an individual *busRoute1*.
  - Make it an instance of BusRoute.
  - Give it the *busRouteNumber* "1302".

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Exercise One

- Define an individual *trainStation1*.
  - Make it an instance of TrainStation.
  - Give it the *stationCode* "LAX".
- Define an individual *trainStation2*.
  - Make it an instance of TrainStation.
  - Give it the *stationCode* "CHI".
- Define an individual *trainRoute1*.
  - Make it an instance of TrainRoute.
  - Give it the *trainRouteID* "4 Southwest Chief".

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Exercise One

- Assert that *busRoute1 departsFrom busStation1* and *arrivesAt busStation2*.
- Assert that *trainRoute1 departsFrom trainStation1* and *arrivesAt trainStation2*.

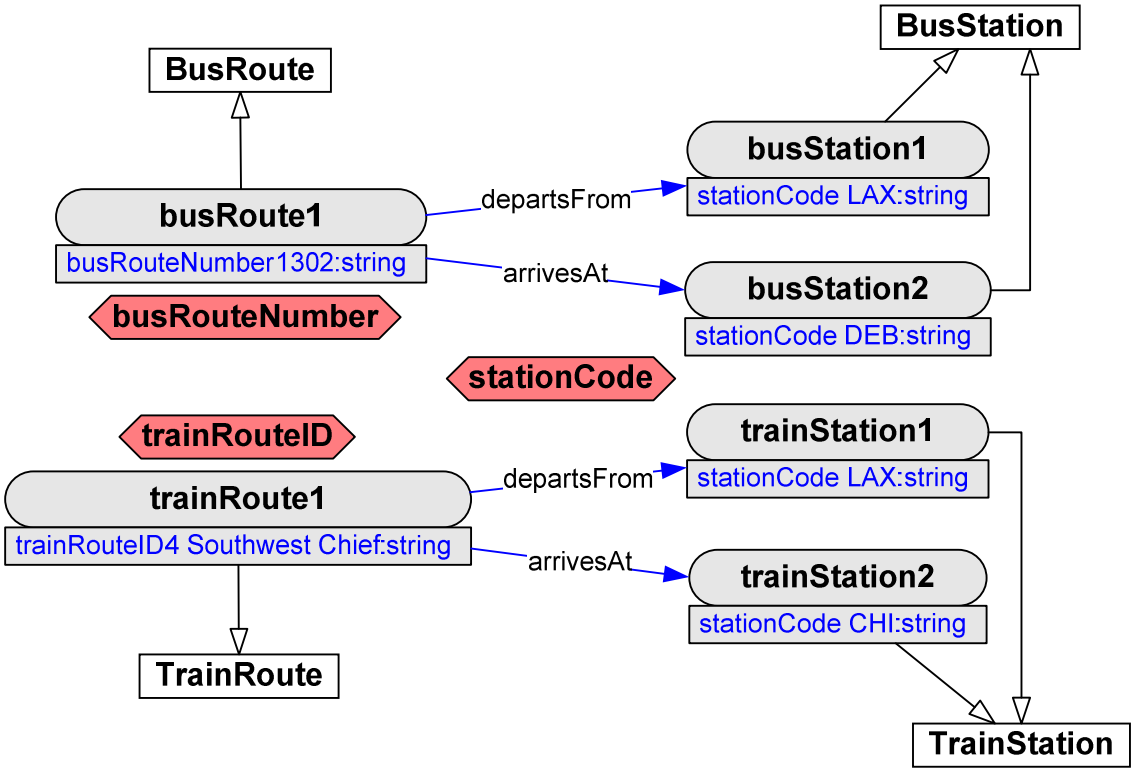
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Exercise One




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Equivalence

- Equivalence asserts that two things are ***the same***.
- Equivalence can be asserted between two ***classes***, two ***properties***, or two ***individuals***.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Equivalent Classes

- Asserting that two classes are equivalent states that ***every member of that class is also a member of the equivalent class.***
- Equivalent classes can be a useful way to solve the ***synonym problem***.
- For example, suppose that in our geography ontology that we refer, in different places, to **cities** and **towns**.
- *If we don't care about this distinction*, and we want anything previously labeled a *Town* to also be considered a *City*, we can simply declare the classes equivalent.

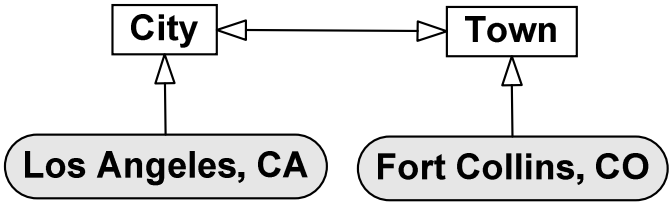
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

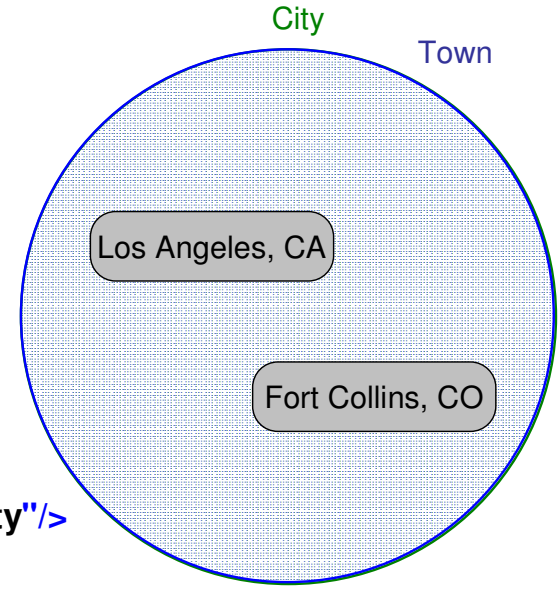
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Equivalent Classes




```
<owl:Class rdf:ID="City“/>
<owl:Class rdf:ID="Town">
   <owl:equivalentClass rdf:resource="#City"/>
</owl:Class>
<geo:City rdf:ID="Los Angeles, CA"/>
<geo:Town rdf:ID="Fort Collins, CO"/>
```

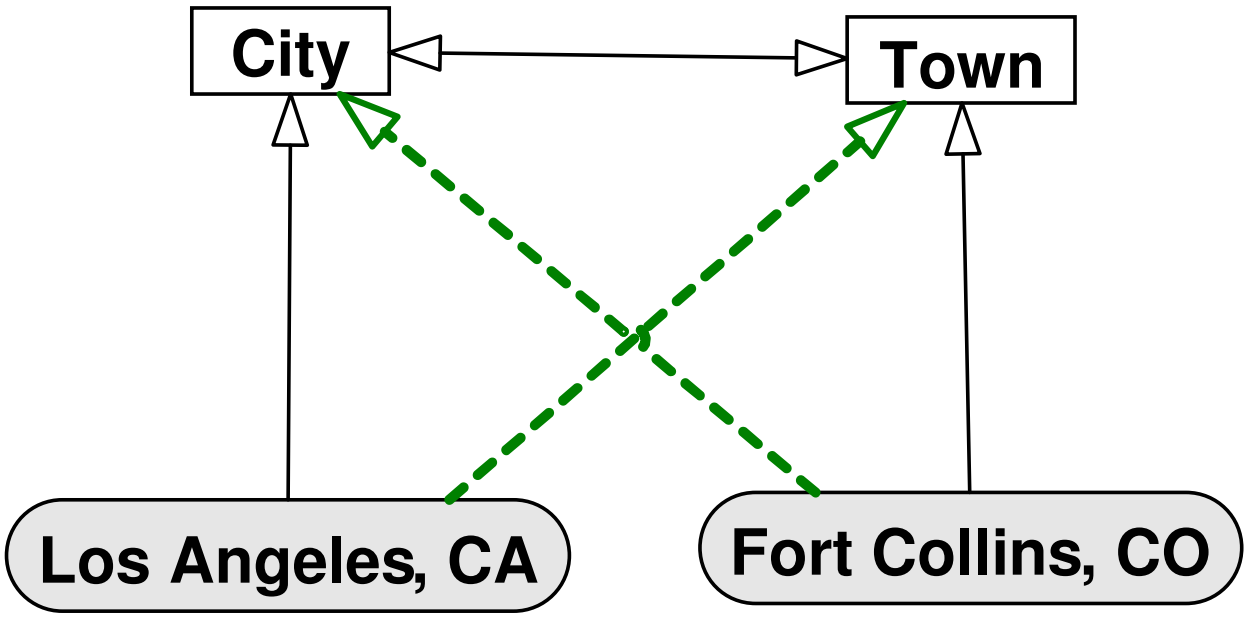
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Equivalent Classes




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Exercise Two

- Add a class called *RailRoute* to your ontology.
- Make *RailRoute* equivalent to *TrainRoute*.
- Resubmit and query the ontology for **provable** *RailRoute*s and *TrainRoute*s.
  - member($i,RailRoute,$m)

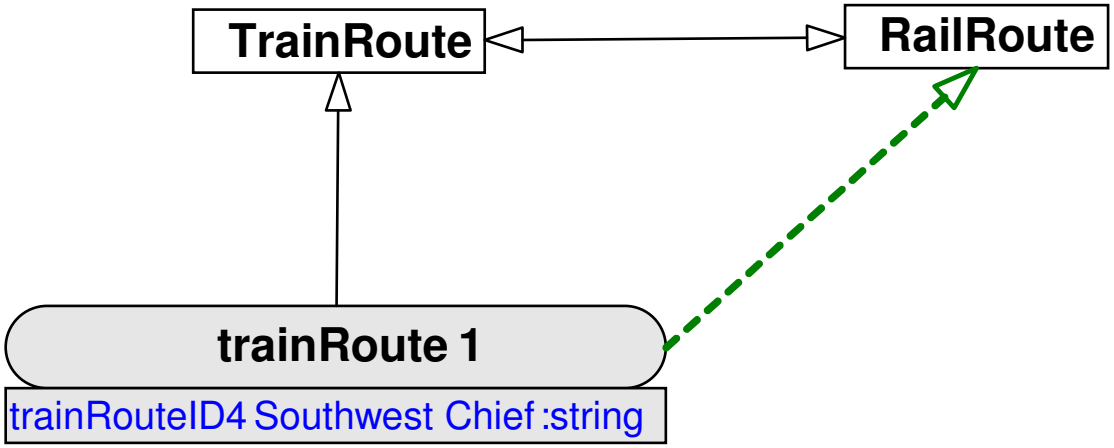
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Exercise Two




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

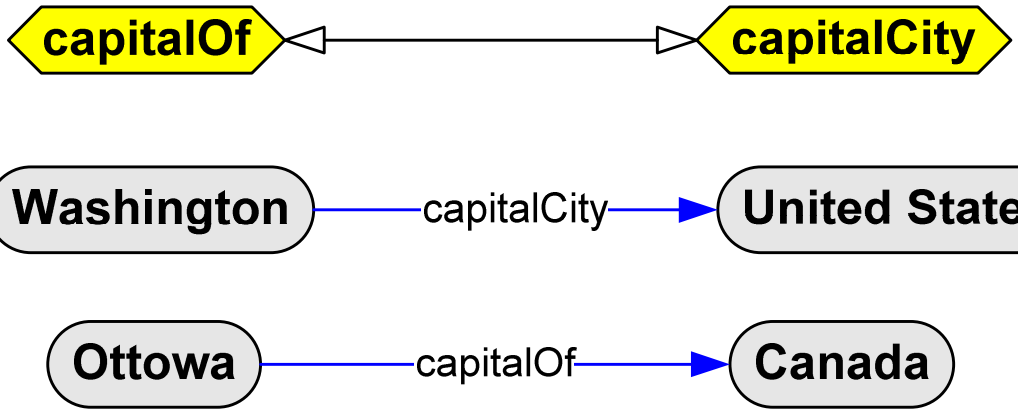
# Equivalent Properties

- Similar to equivalent classes, *declaring two properties equivalent makes them* ***interchangeable***.
- This is again a useful way to solve the ***synonym problem***.
- Suppose that we have defined a property *capitalCity* that relates a city to a country.
- If we had elsewhere used the property *capitalOf*, we could use property equivalence to make them interchangeable.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Equivalent Properties




```
<owl:ObjectProperty rdf:ID="capitalOf“/>
<owl:ObjectProperty rdf:ID="capitalCity">
   <owl:equivalentProperty rdf:resource="#capitalOf"/>
</owl:ObjectProperty>
```

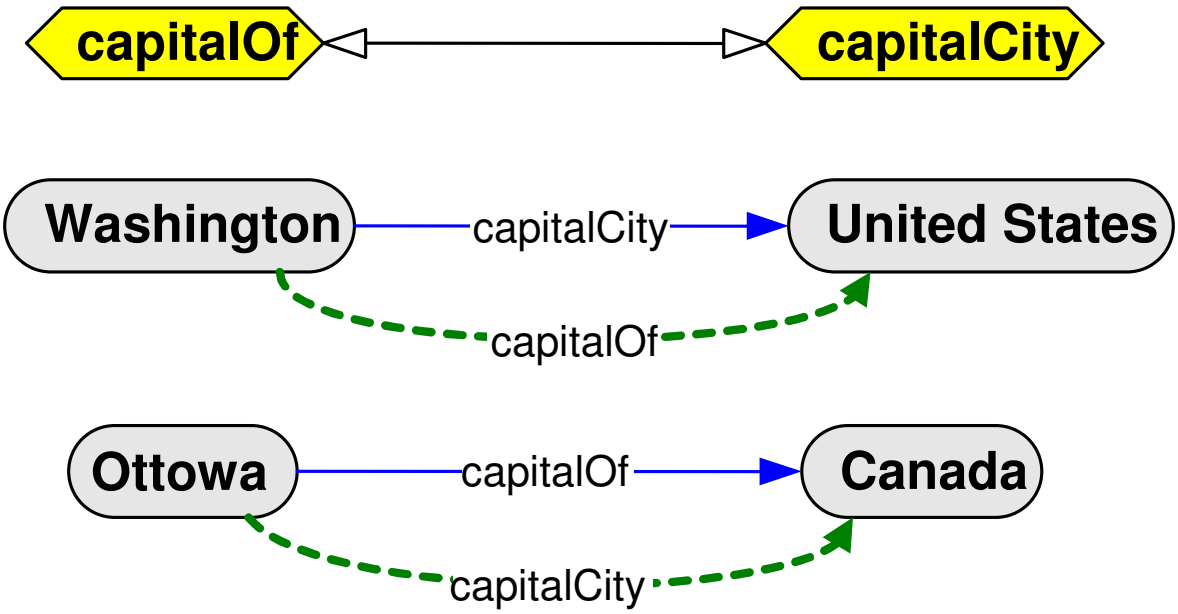
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Equivalent Properties




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Exercise Three

- Create an object property and call it *leaves*.
- Assert that *leaves* and *departsFrom* are equivalent properties.
- Resubmit and query the ontology for all occurrences of the *leaves* property, of the *departsFrom* property.
  - *related($a,leaves,$b,$m)*
  - *related($a,departsFrom,$b,$m)*

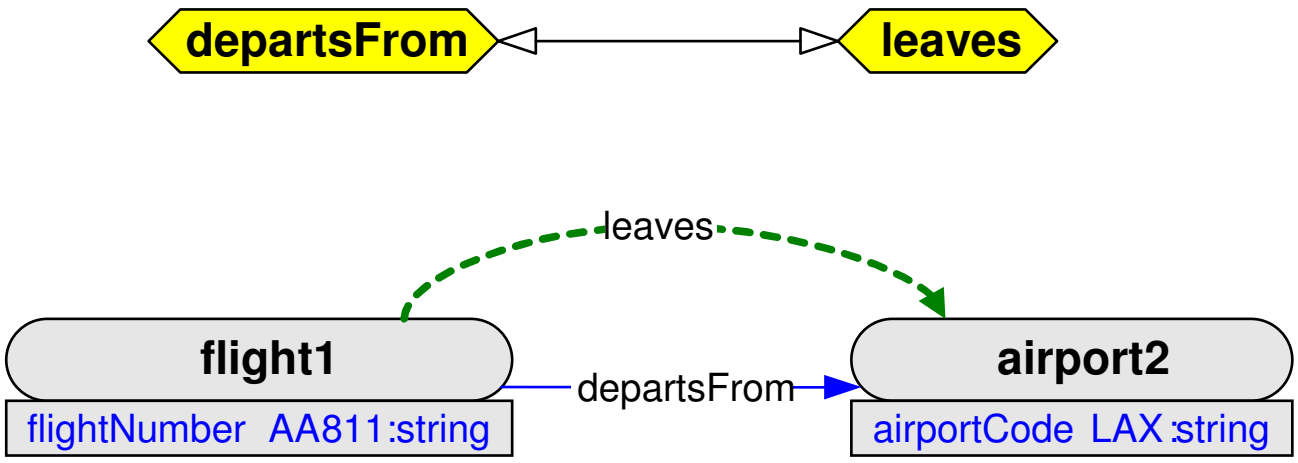
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Exercise Three




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Equivalent Individuals

- This gets a bit tricky.
- There is no assumption in OWL that two individuals are different from each other.
- If two individuals are equivalent, ***everything asserted about one individual is also true of the other***.
- This is true despite the fact that they will have ***distinct URIs***.

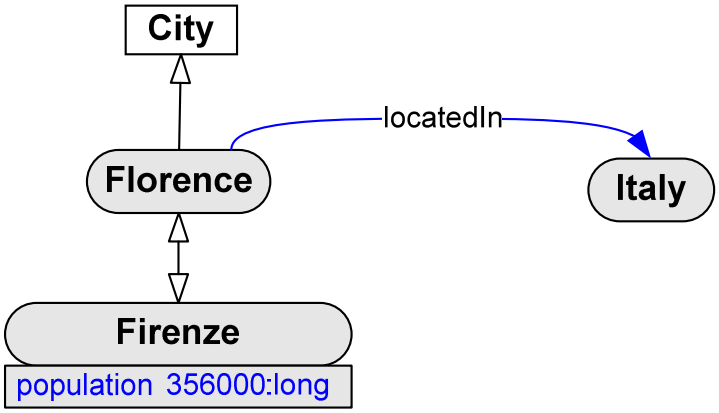
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Equivalent Individuals




```
<geo:City rdf:ID="Florence">
   <geo:locatedIn rdf:resource="#Italy"/>
</geo:City>
<owl:Thing rdf:ID="Firenze">
   <geo:population rdf:datatype="http://...#long">356000</geo:population>
   <owl:sameAs rdf:resource="#Florence"/>
</owl:Thing>
```

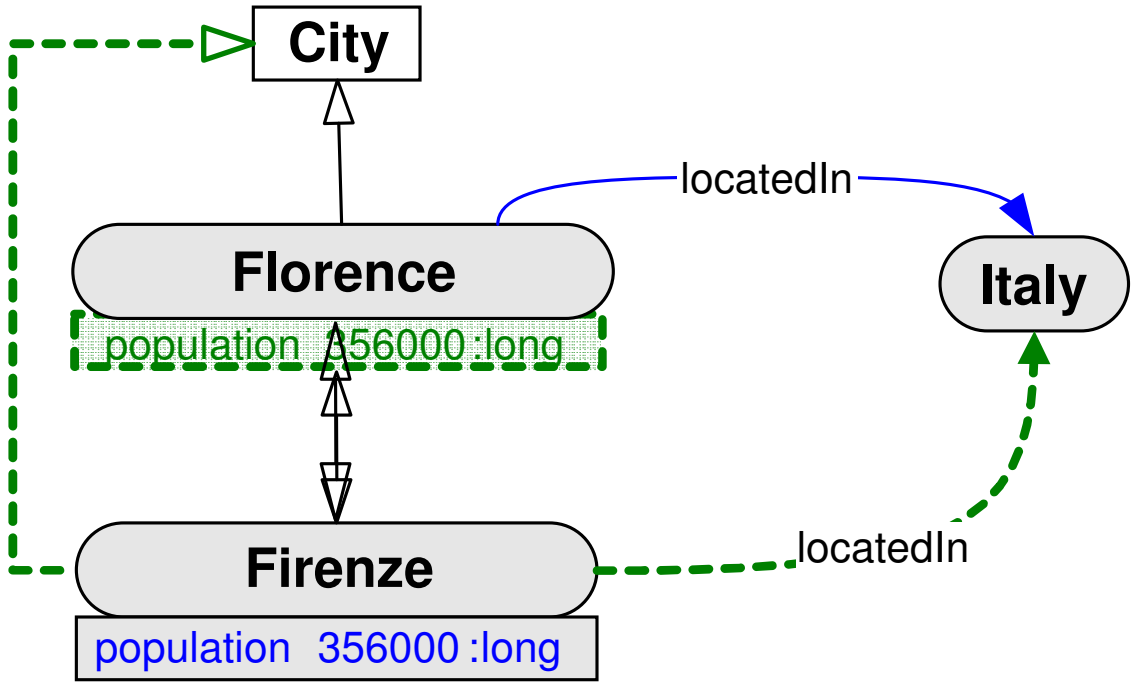
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Equivalent Individuals




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Exercise Four

- You may have noticed that three individuals in our ontology have the *terminalCode* "LAX."
- Los Angeles International Airport is a departure point not just for planes, but for buses and trains as well.
- Our three OWL individuals all refer to the same real-world thing.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Exercise Four

- Assert that *busStation1*, *trainStation1*, and *airport2* are all equivalent individuals.
  - How many equivalence axioms does this require?
- Resubmit and query your ontology for things that depart from *airport2* (*busStation1*, etc.).
  - related($i,departsFrom,airport2,$m)
- Query your ontology for provable *Airport*s, *TrainStation*s, *BusStation*s.
  - member($i,Airport,$m)

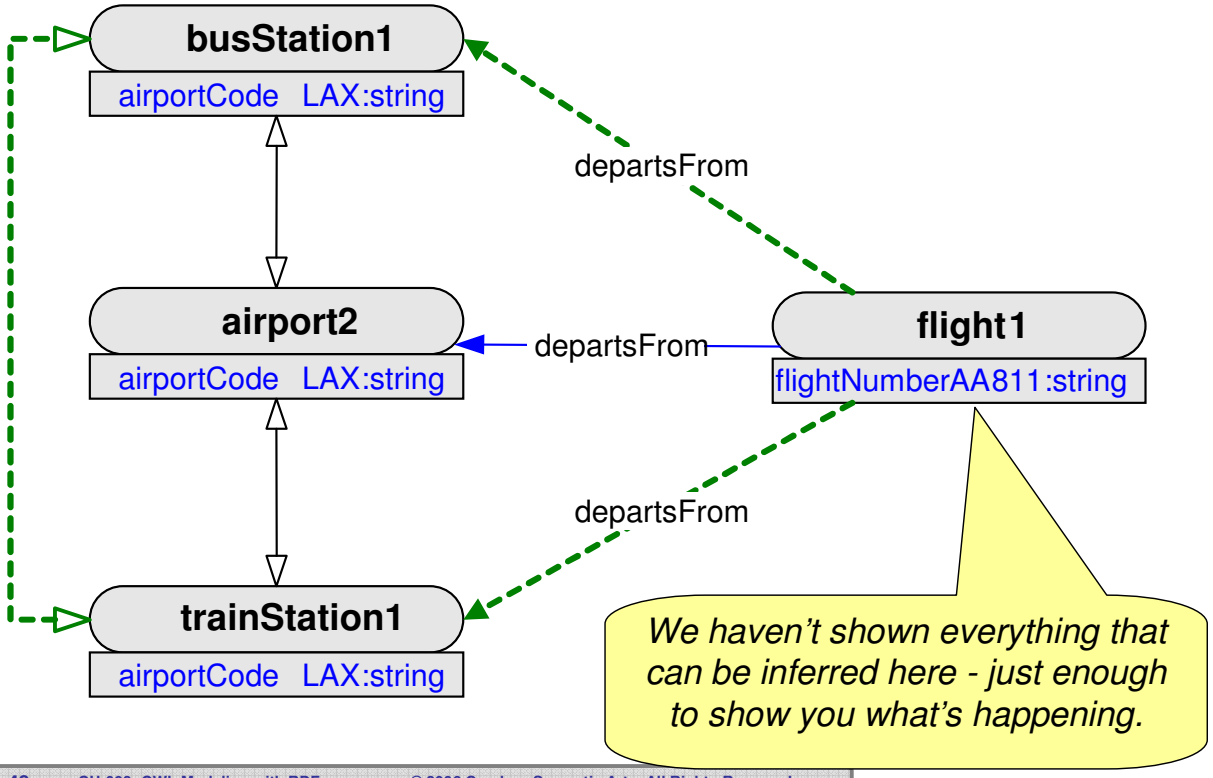
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Exercise Four



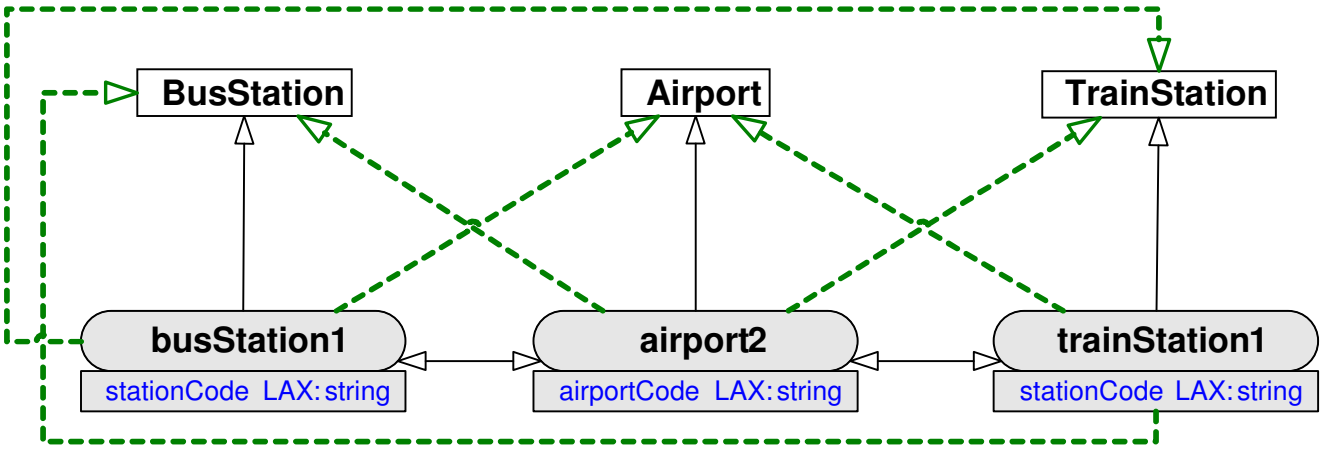

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Exercise Four




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Disjointness

- We have already seen this used in the previous section.
- Disjointness asserts that two things are ***different***.
- Disjointness can be asserted between two ***classes*** or two ***individuals***.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Disjoint Classes

- Recall from the previous section that an individual can be a member of more than one class, and can possibly be a member of any class unless it is *provably demonstrated otherwise*.
- Asserting that two classes are disjoint asserts that *any member of that class is provably **not a member** of the disjoint class.*
  - *i.e., the classes can have no members in common.*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Disjoint Classes

- To begin our geographic ontology we defined four basic classes:

**Country** | **State** | **Province** | **City**

- We can think of examples of cities that are also countries, and even cities that are also states.
- For the purposes of our ontology, let's declare that ***a state cannot also be a country***.

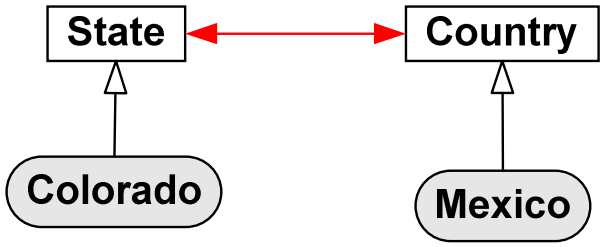
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

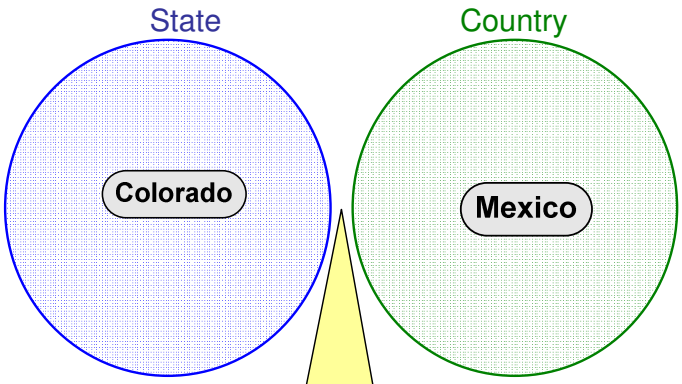
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Disjoint Classes




```
<owl:Class rdf:ID="State">
   <owl:disjointWith rdf:resource="#Country"/>
</owl:Class>
<geo:State rdf:ID="Colorado"/>
<geo:Country rdf:ID="Mexico"/>
```

*There is no overlap between these two classes. Nothing can be a member of both.*

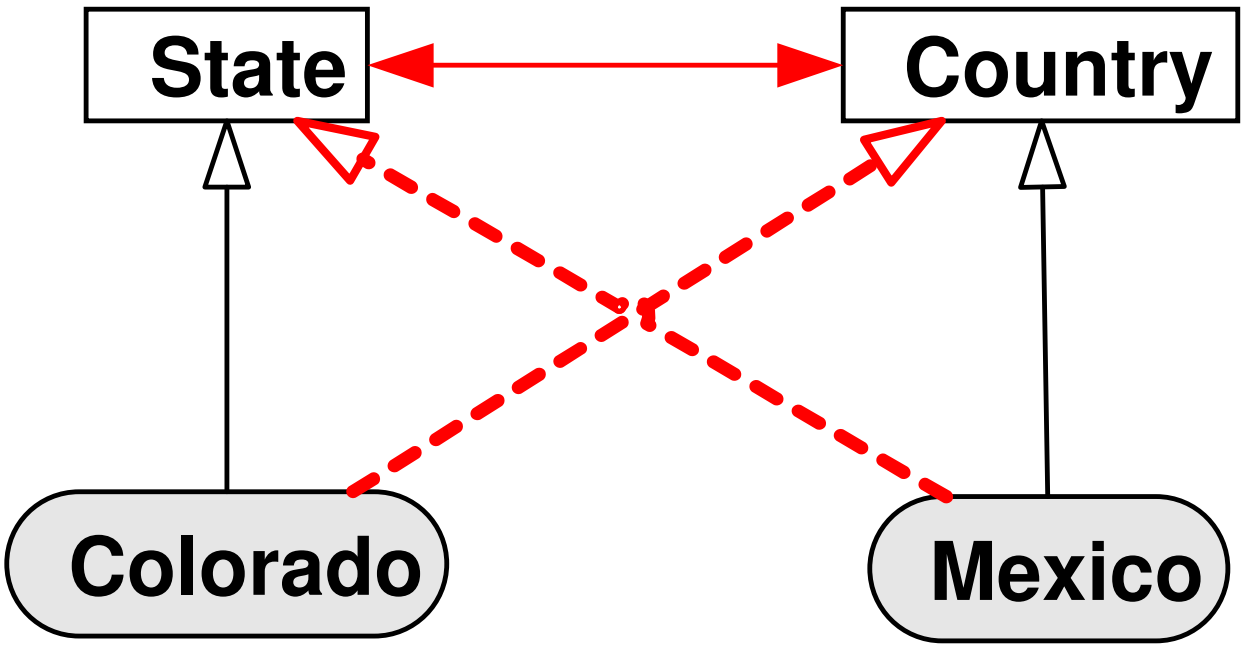
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Disjoint Classes




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Exercise Five – Class Discussion

- In the last module, we made *FlightRoute* and *Airport* disjoint.
- Should we make *FlightRoute*, *BusRoute*, and *TrainRoute* disjoint?
- Should we make *Airport*, *BusStation*, and *TrainStation* disjoint?

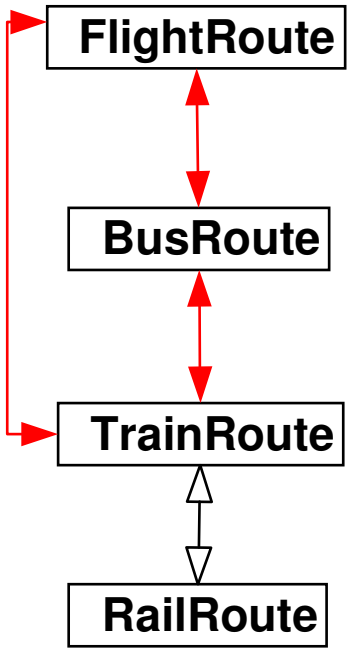
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

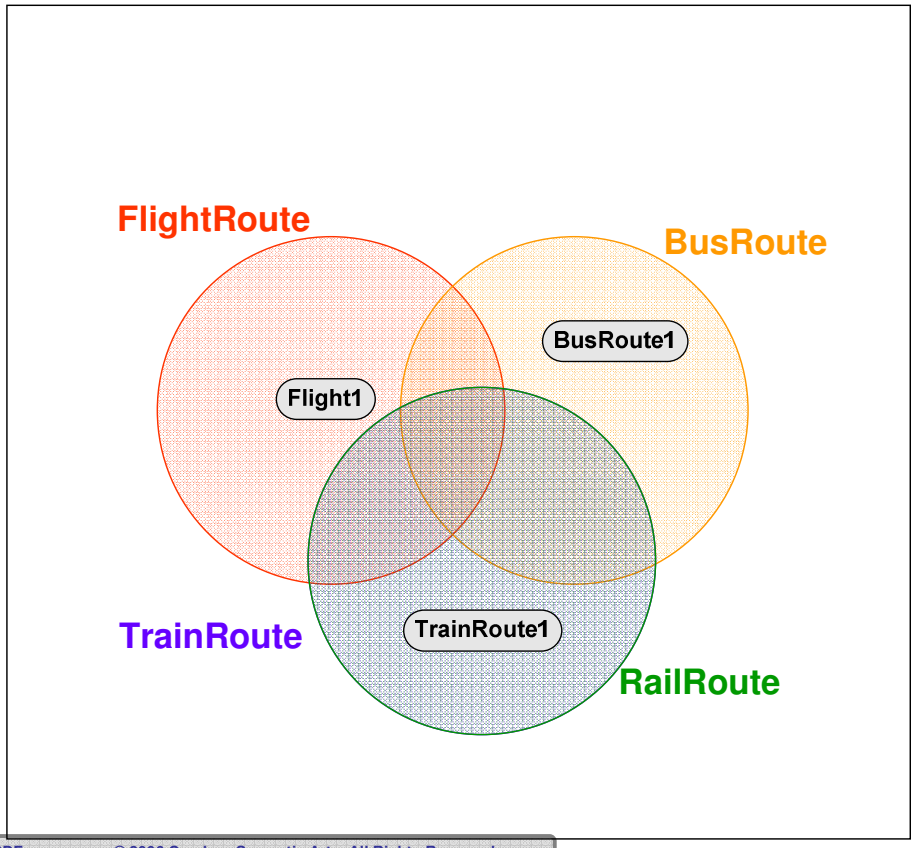
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Exercise Five




50 CU-003: OWL Modeling with RDF. © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

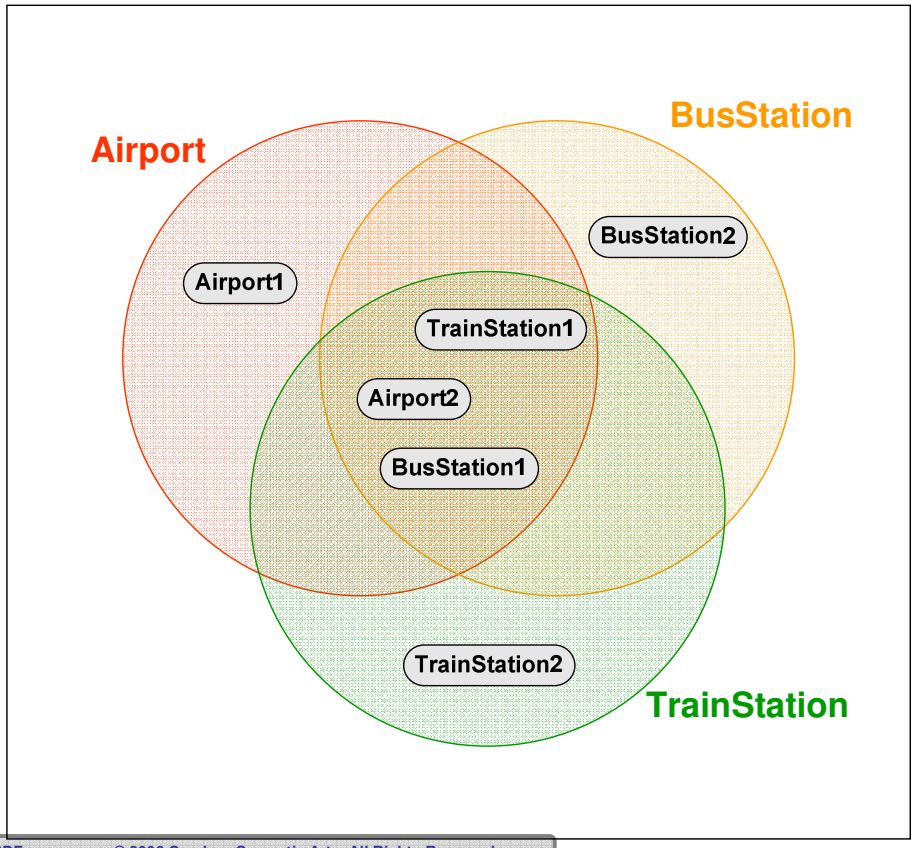
CU-003
02/2006
Module 3

# Exercise Five

**BusStation**

**Airport**

**TrainStation**




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Disjoint Individuals

- Since it is possible in OWL for two individuals to be equivalent, there needs to be a way to declare them distinct.
- Declaring two individuals disjoint *prevents the inference engine from considering the possibility that they are equivalent.*
  - i.e., they are **provably *not* equivalent**

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Disjoint Individuals

- Suppose in our geography ontologies we had two references to "Kansas City."
- One of these instances refers to Kansas City, Missouri, and the other refers to Kansas City, Kansas.
- Declaring them to be disjoint means that the inference engine will never consider the possibility that these two are the same.

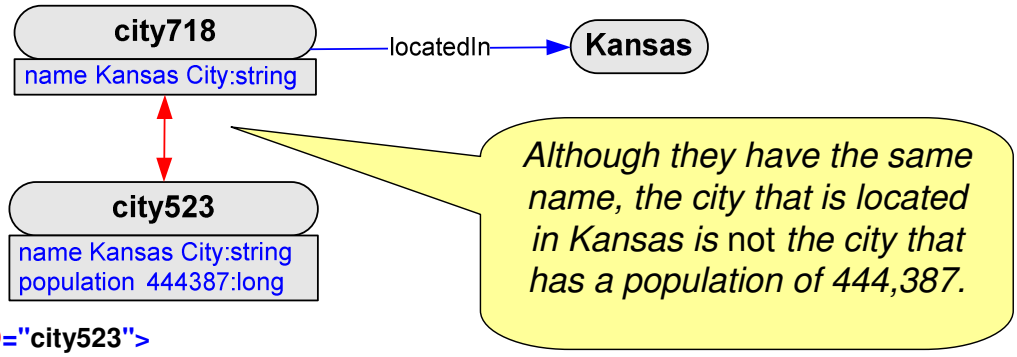
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Disjoint Individuals




```
<geo:City rdf:ID="city523">
   <geo:name rdf:datatype="http://...#string">Kansas City</geo:name>
   <geo:population rdf:datatype="http://...#long">444387</geo:population>
   <owl:differentFrom rdf:resource="#city718"/>
</geo:City>
<geo:City rdf:ID="city718">
   <geo:locatedIn rdf:resource="#Kansas"/>
   <geo:name rdf:datatype="http://...#string">Kansas City</geo:name>
</geo:City>
```

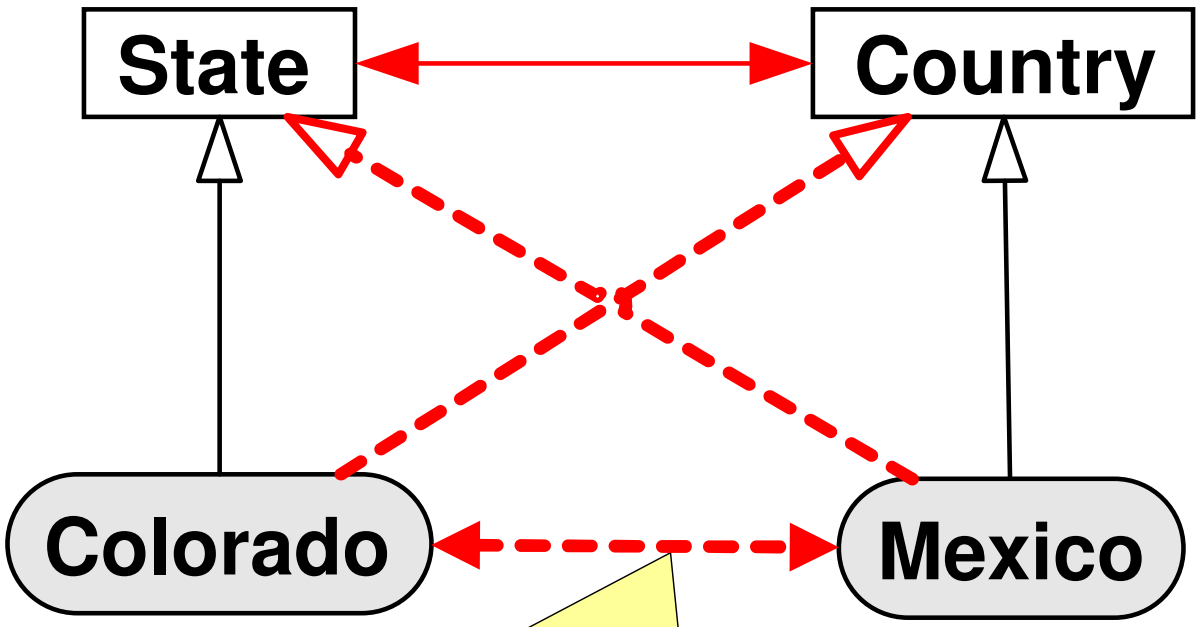
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Disjoint Individuals




*Because Colorado and Mexico are members of disjoint classes, we can also infer that they are individually disjoint.*

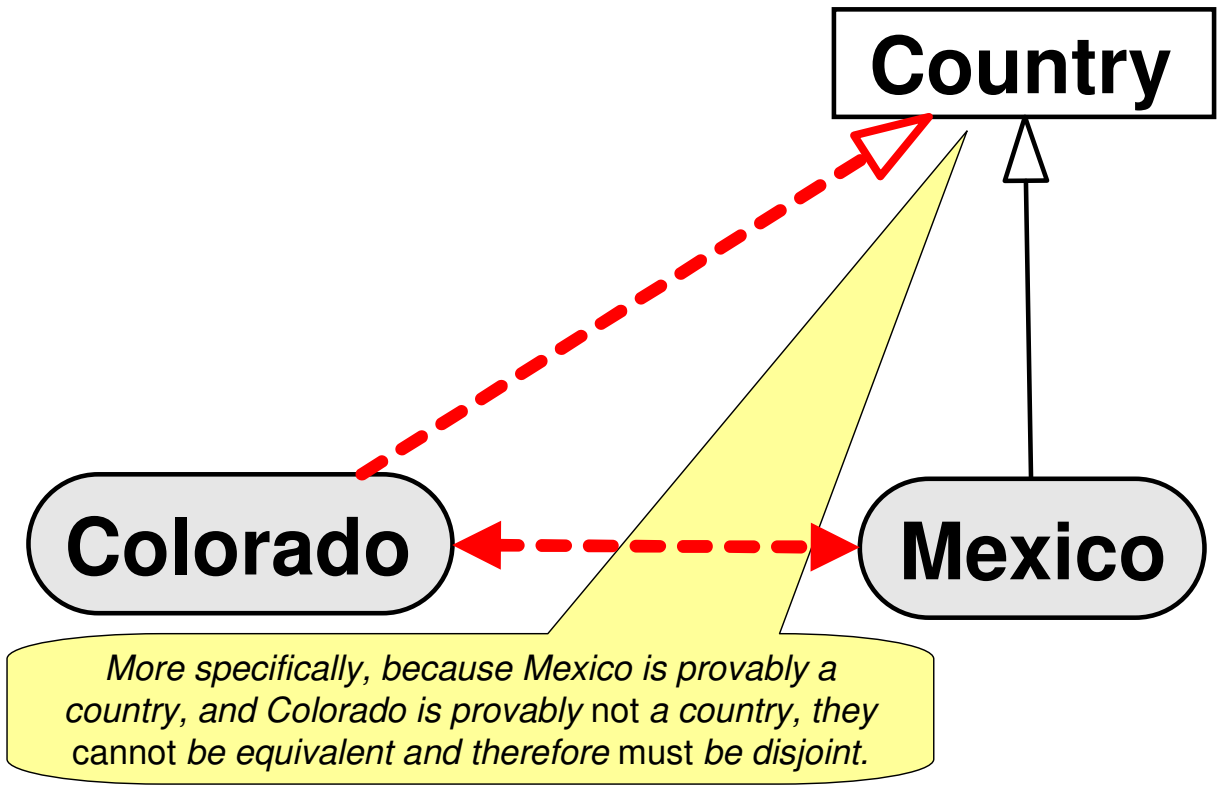
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Disjoint Individuals



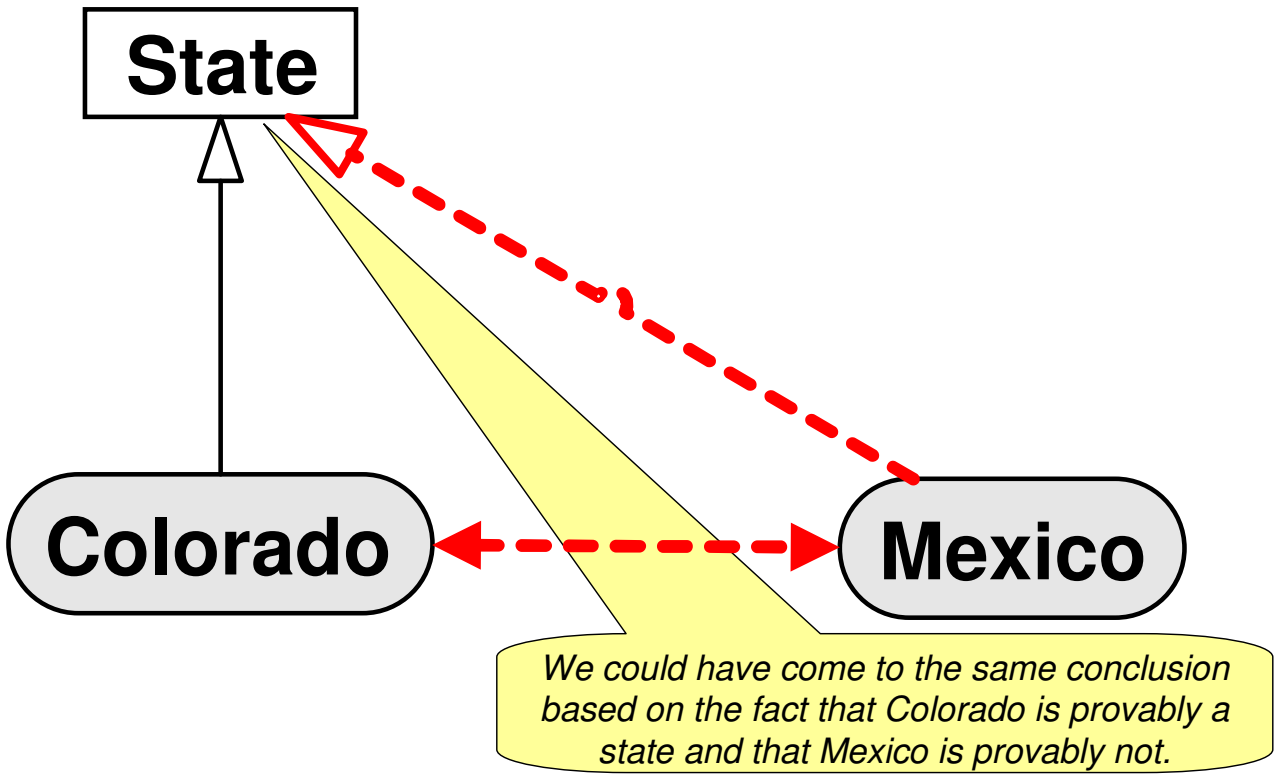

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Disjoint Individuals




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

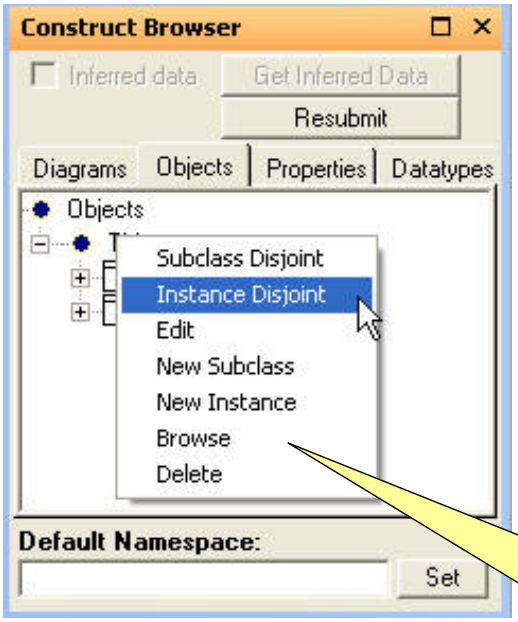
# Disjoint Individuals

- In practice, you will often find that *all* of your individuals are disjoint (or at least a large number of them).
- Rather than having to declare every binary disjointness relationship, OWL offers a simple shorthand method for giving a list of individuals, and declaring that each is disjoint from the others.
- For example, in a U.S. Geography ontology, all of the states are distinct from one another.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Disjoint Individuals




```
<owl:AllDifferent>
  <owl:distinctMembers
        rdf:parseType="Collection">
    <owl:Thing rdf:about="#Alabama"/>
    <owl:Thing rdf:about="#Alaska"/>
    <owl:Thing rdf:about="#Arizona"/>
    ...
    <owl:Thing rdf:about="#Wisconsin"/>
    <owl:Thing rdf:about="#Wyoming"/>
  </owl:distinctMembers>
</owl:AllDifferent>
```

*Right-click on the class name in the Construct Browser to get this menu.*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# The Unique Names Assumption

- The need for individual equivalence and disjointness axioms stems from the fact that OWL/RDF does not have a **unique names assumption**.

- Under a unique names assumption, every identifier refers to a unique individual, concept, or "thing" in the real world.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

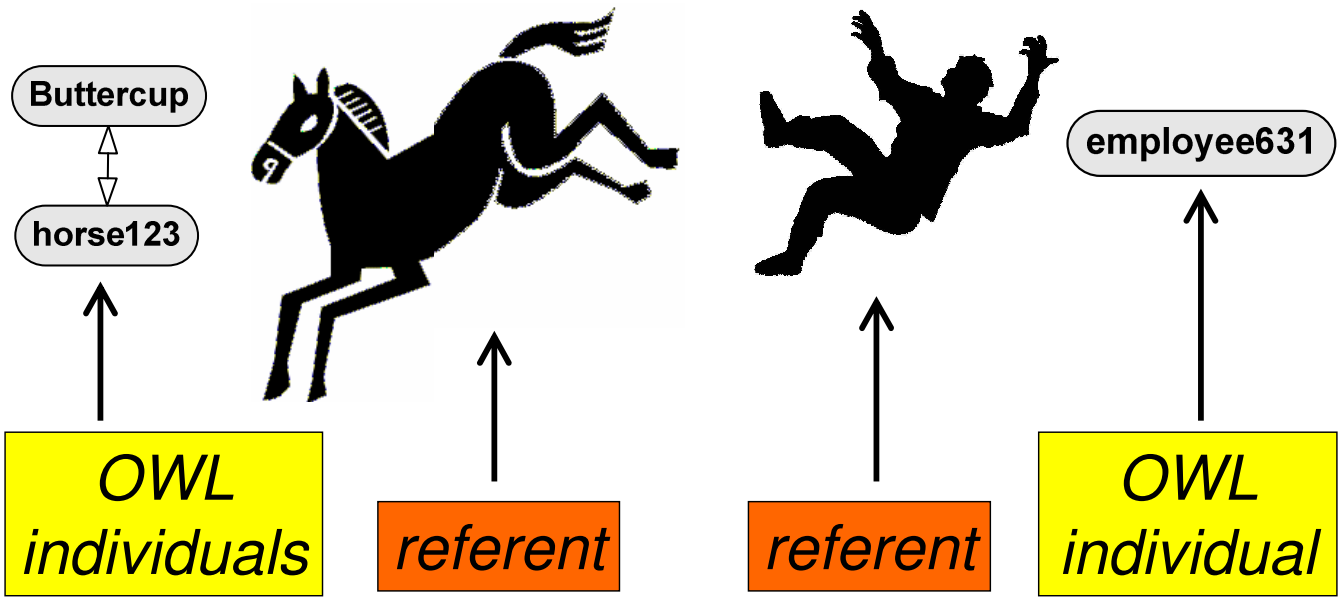
# Referents

- As we have seen, in OWL two distinct individuals can "point" to the same real-world entity.
- This real-world entity is the individual's **referent**.
- Two individuals can share a common referent; this makes them equivalent.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# "The referent is what kicks you."*




* Ferdinand de Saussure (1857-1913)

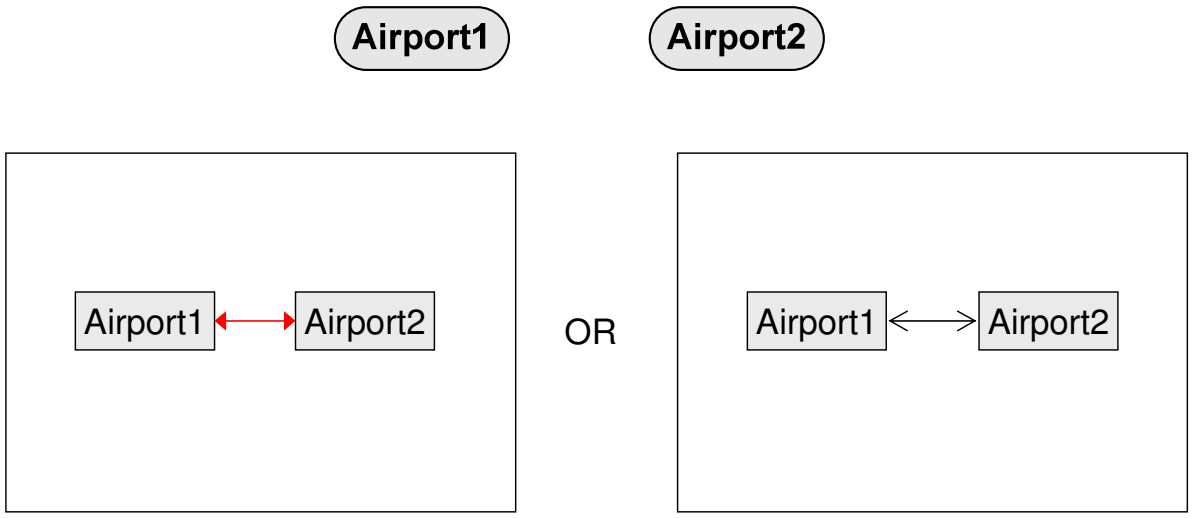
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Individuals in the Open World




In this model **Airport1** and **Airport2** represent different real world airports. (The URIs identify different referents.)

In this model **Airport1** and **Airport2** represent the same real world airport. (The URIs identify the same referent.)

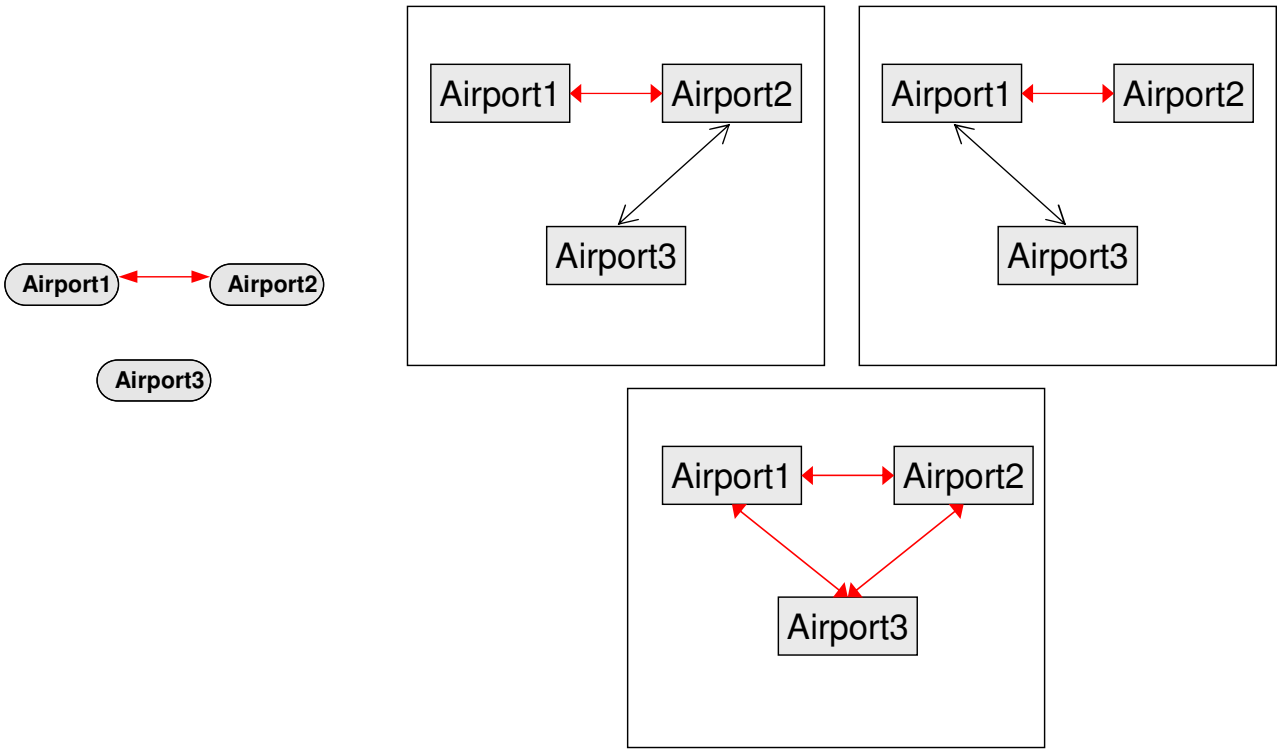
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Individuals in the Open World



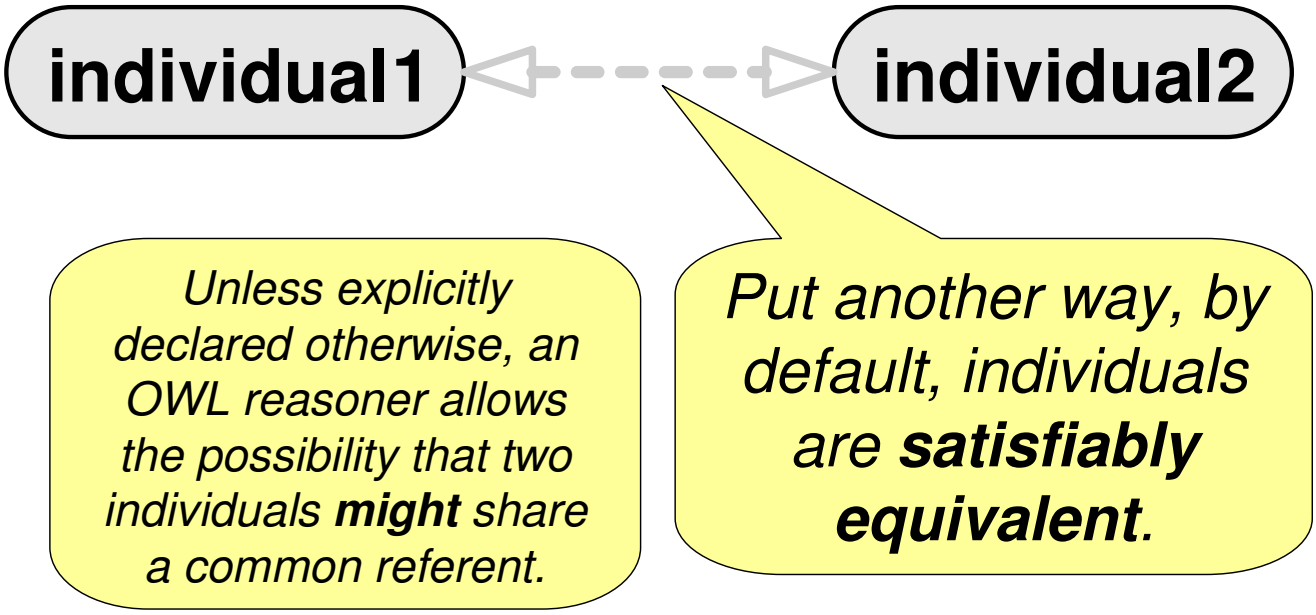

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Referents, OWL, and the Open World




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Subsumption

- Subsumption expresses a ***subset of*** relationship.
- Subsumption can be applied to **classes** and **properties**.
- **Supersumption** is also possible.
  - i.e., taking a group of existing concepts and making them subsets of a new one.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Subclasses

- Subclasses are best thought of in terms of set theory and **subset** relationships.
- If an individual is a member of a class, ***it is provably a member of its superclass(es).***
  - The converse is not necessarily true.
- *A class can have multiple sub- or superclasses.*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Subclasses

- So far, our ontology has concepts like *City*, *State*, and *Country*.
- It might be useful to generalize these concepts by adding a *GeographicRegion* superclass, which encompasses all geographic concepts.

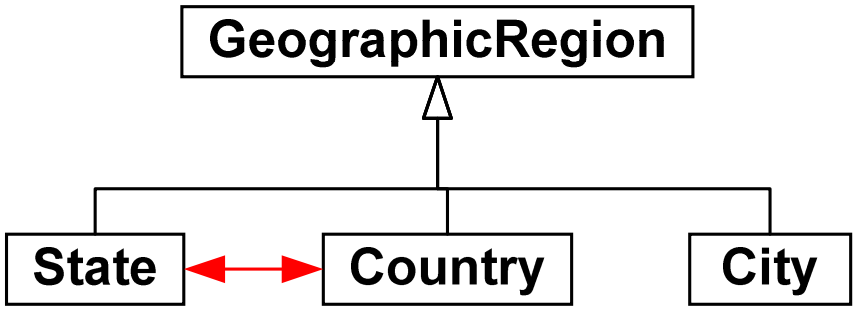
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Subclasses




```
<owl:Class rdf:ID="City">
   <rdfs:subClassOf rdf:resource="#GeographicRegion"/>
</owl:Class>
<owl:Class rdf:ID="State">
   <rdfs:subClassOf rdf:resource="#GeographicRegion"/>
   <owl:disjointWith rdf:resource="#Country"/>
</owl:Class>
<owl:Class rdf:ID="Country">
   <rdfs:subClassOf rdf:resource="#GeographicRegion"/>
</owl:Class>
```

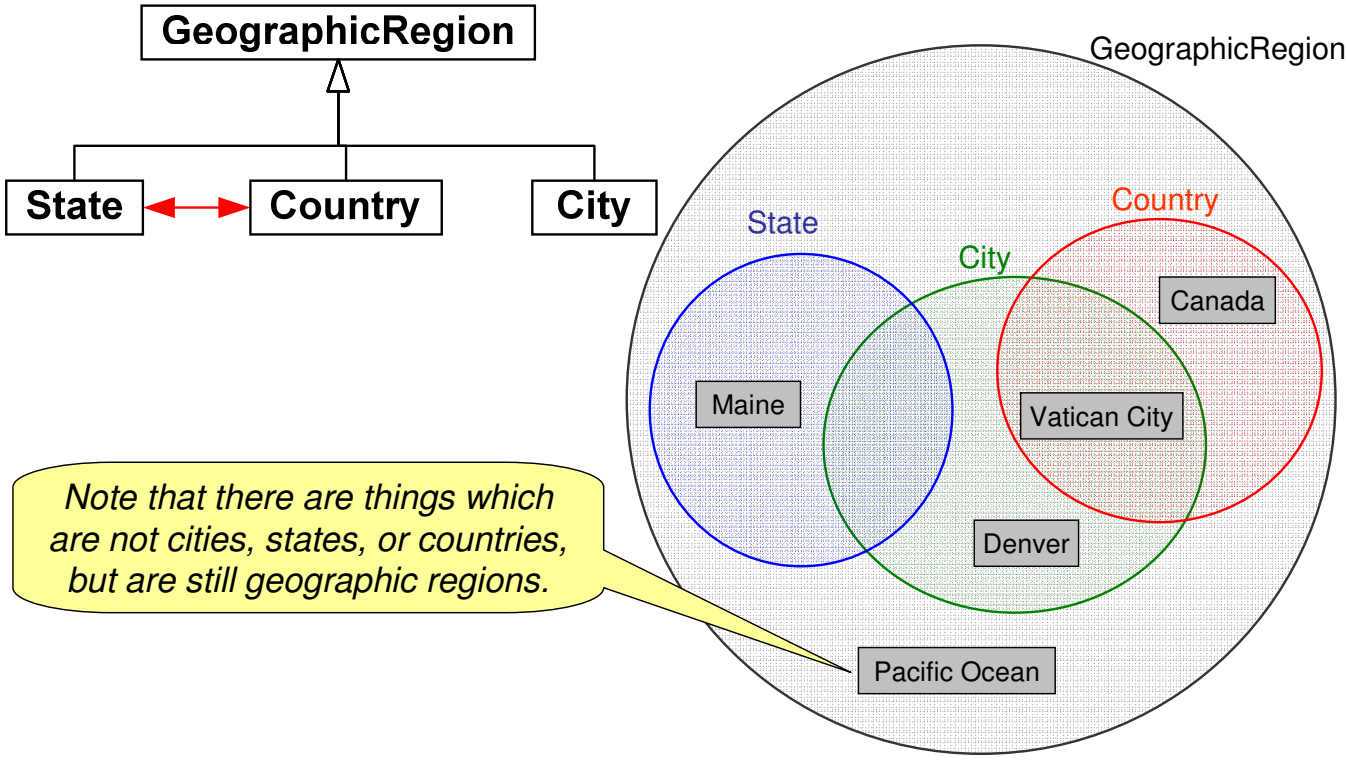
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Subclasses



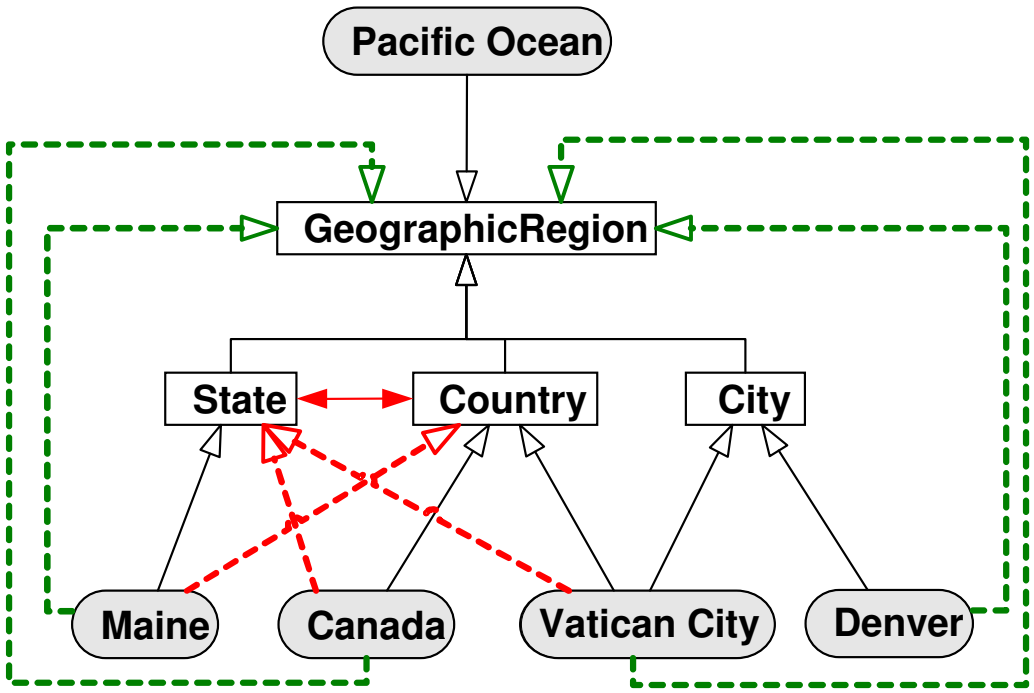

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Subclasses




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Exercise Six

- Generalize your *BusStation*, *TrainStation*, and *Airport* classes into a superclass called *Station*.
- Generalize your *FlightRoute*, *BusRoute*, and *TrainRoute* classes into a superclass called *TravelRoute*.

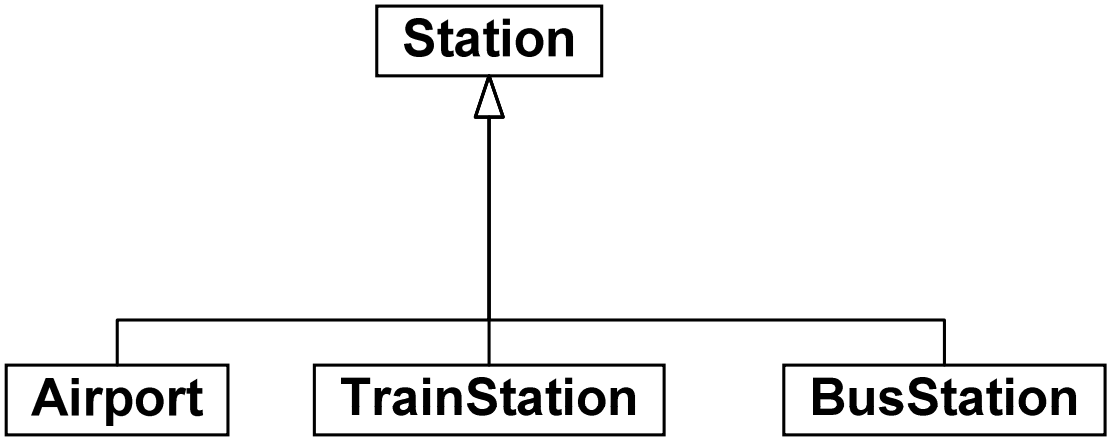
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

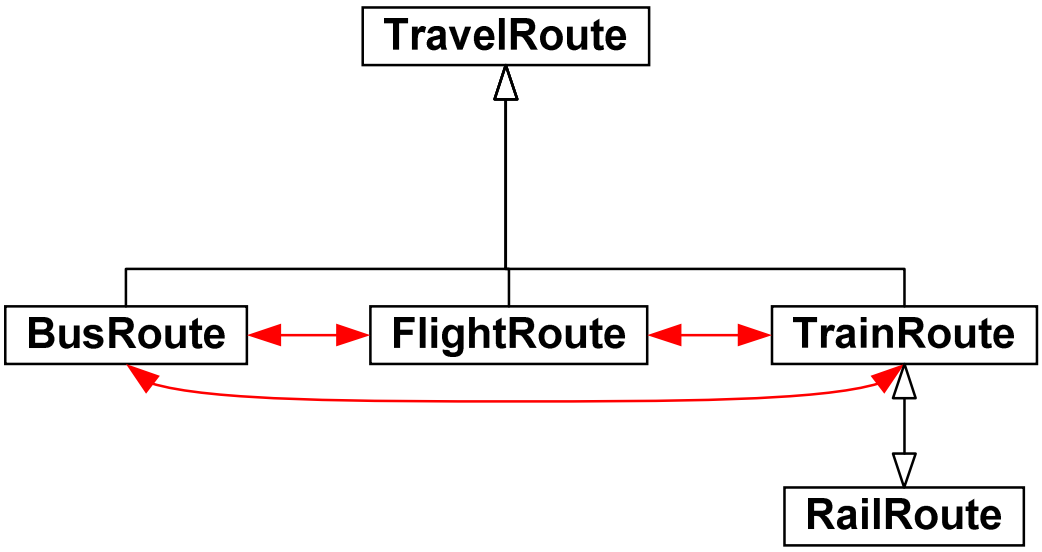
# Exercise Six




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Exercise Six




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Exercise Six – Class Discussion

- Do you need to make *RailRoute* a subclass of *TravelRoute*?

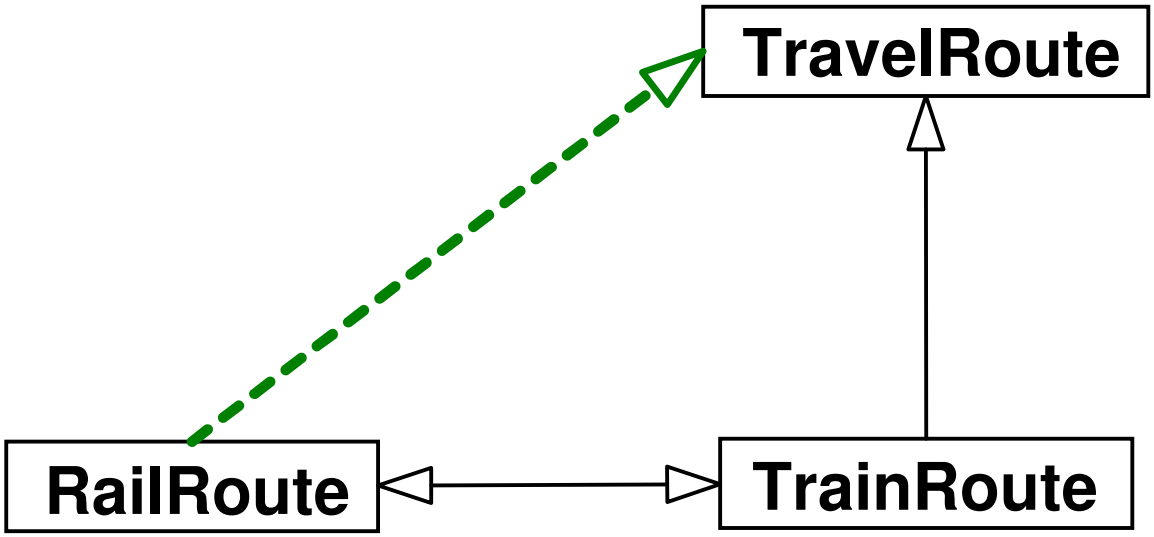
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Exercise Six




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Exercise Six – Class Discussion

- Should you make *TravelRoute* and *Station* disjoint?
- What effect would this have on the their subclasses?

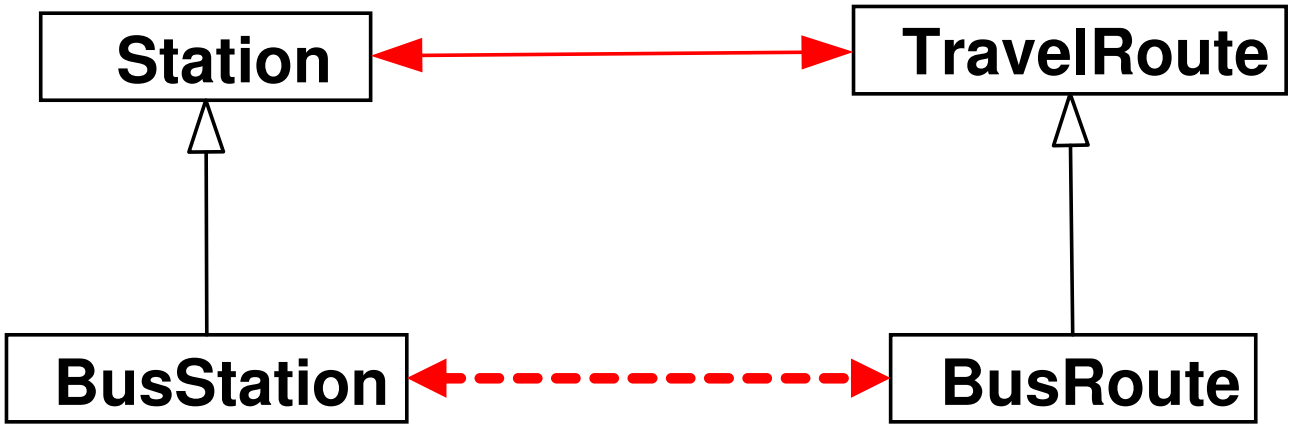
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Exercise Six



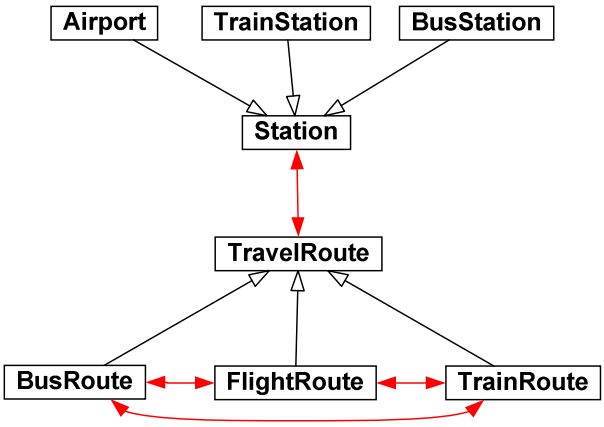

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

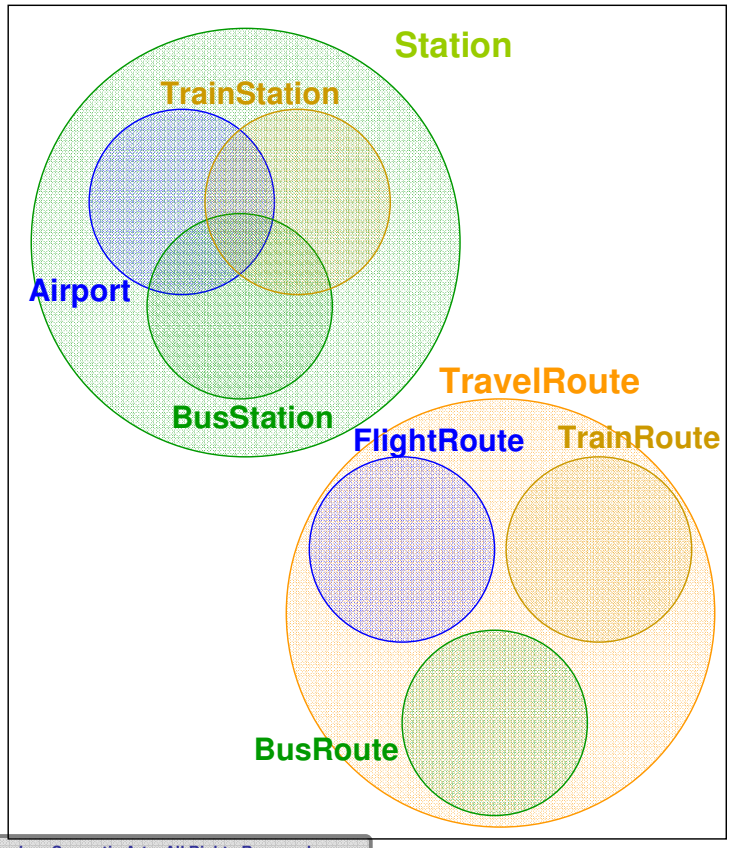
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Exercise Six




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Subproperties

- Subproperties are used when participation in one property implies participation in another.
  - *e.g., "hasParent" and "hasAncestor"*
- Unlike property equivalence, property subsumption does not make the properties interchangeable.
  - *"hasParent" implies "hasAncestor," but not vice versa*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

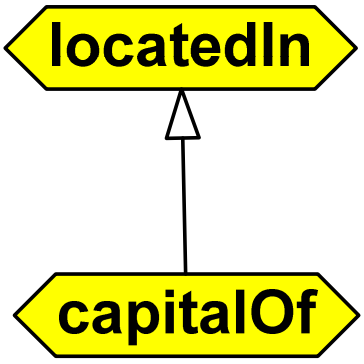
# Subproperties

- In our geography ontology, declaring that a city is the capital of a region necessarily implies that it is located in that region.
- We can express this relationship by making capitalOf a subproperty of locatedIn.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Subproperties




```
<owl:ObjectProperty rdf:ID="locatedIn"/>
<owl:ObjectProperty rdf:ID="capitalOf">
   <rdfs:subPropertyOf rdf:resource="#locatedIn"/>
</owl:ObjectProperty>
```

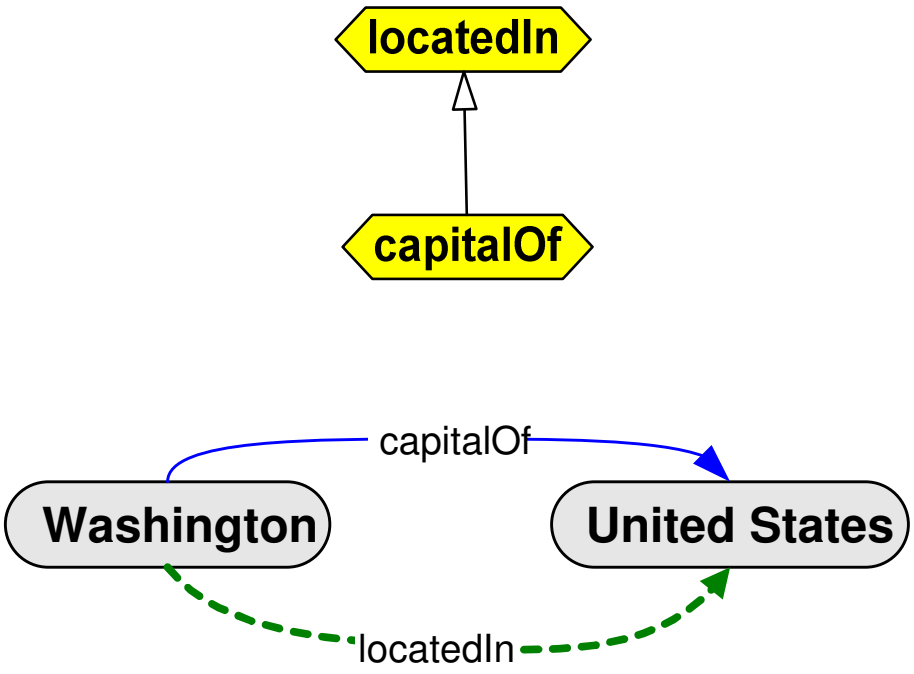
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Subproperties




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Exercise Seven

- It's clear that *flightNumber*, *busRouteNumber*, and *trainRouteID* all describe the same type of attribute.
- The same is true of *airportCode* and *stationCode*.
- Using subproperties, we can express this generalization.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

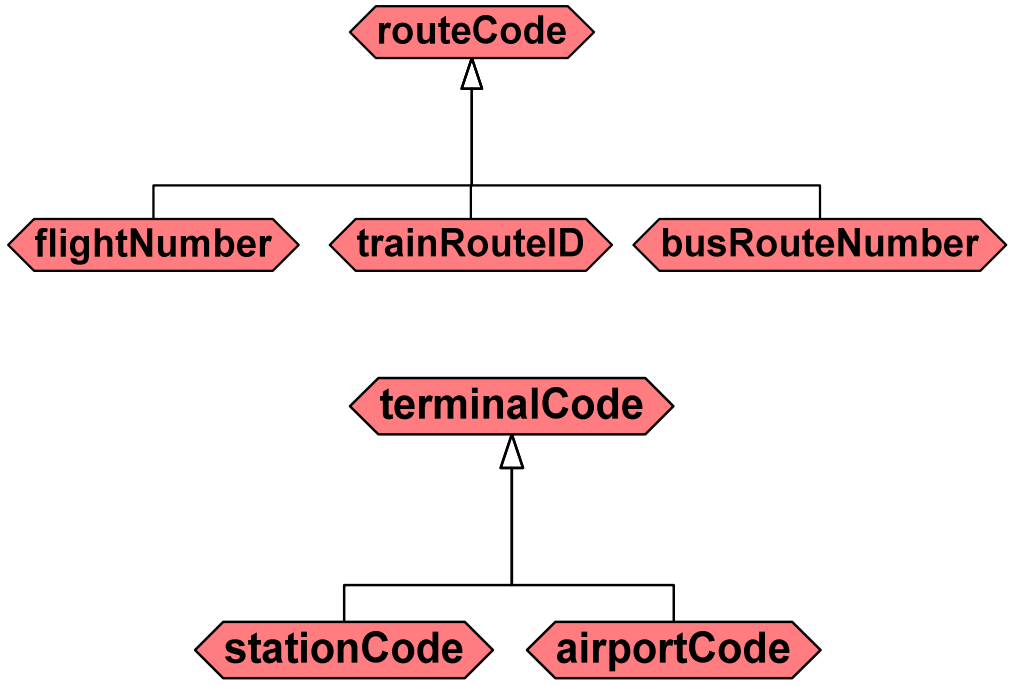
# Exercise Seven

- Define the datatype properties *routeCode* and *terminalCode*.
- Make *flightNumber*, *busRouteNumber*, and *trainRouteID* subproperties of *routeCode*.
- Make *stationCode* and *airportCode* subproperties of *terminalCode*.
- Query your ontology for individuals and their *routeCode*s, *terminalCode*s.
  - related($i,routeCode,$d,$m)

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

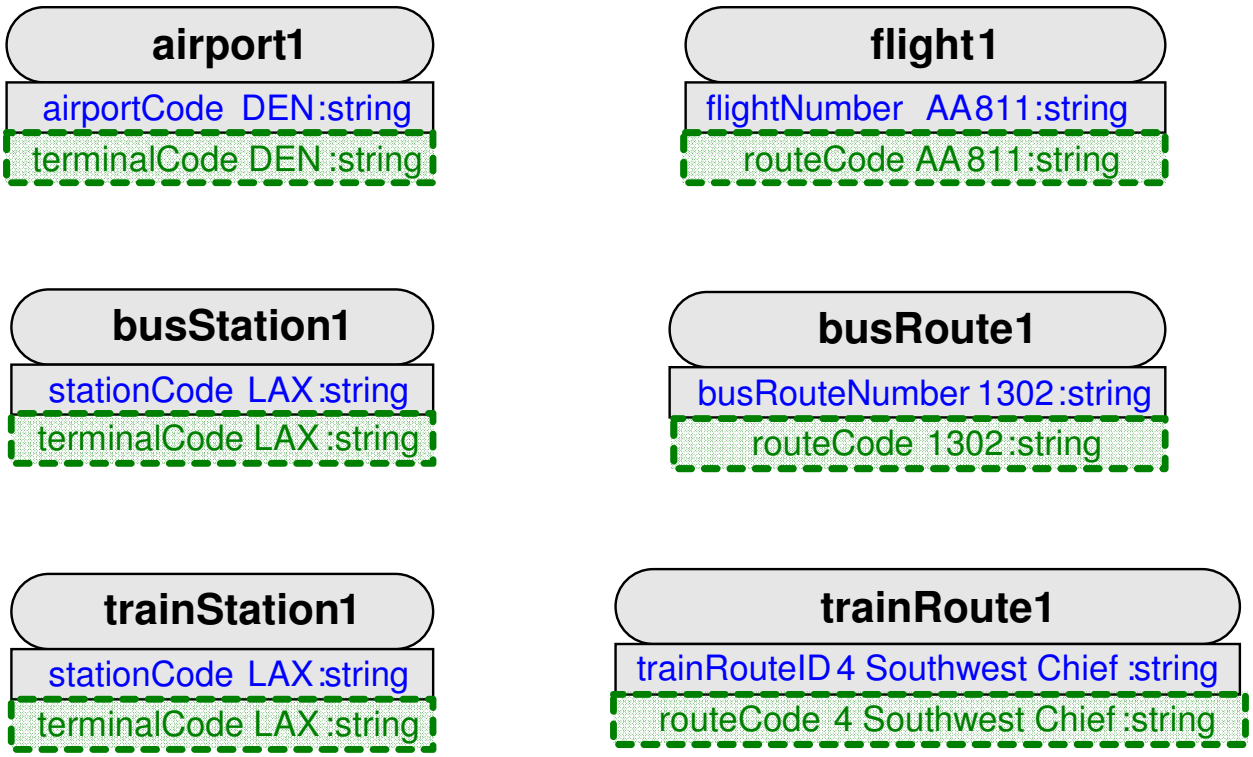
# Exercise Seven




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Exercise Seven




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# TBox and ABox

- In Description Logics, classes and properties, and the relationships between them, are referred to as the *Terminological Box* or **TBox**.
- Individuals and the relationships between them form the *Assertional Box* or **ABox**.

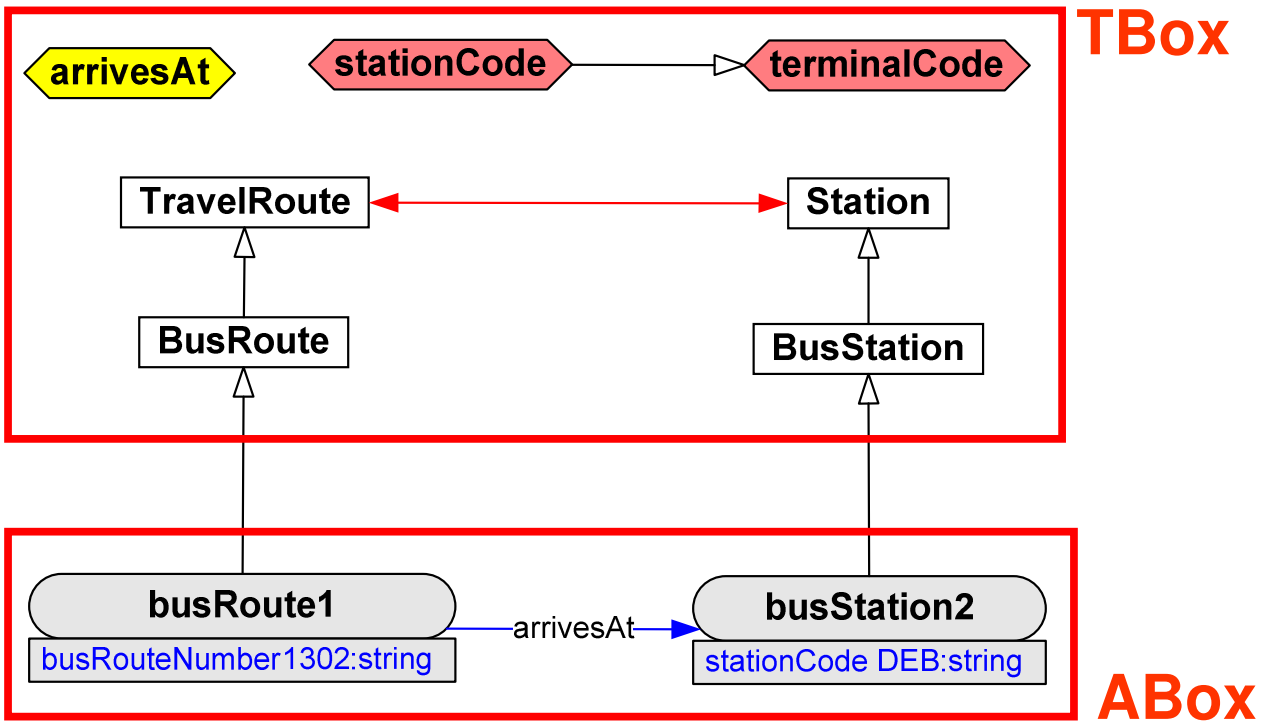
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# TBox and ABox



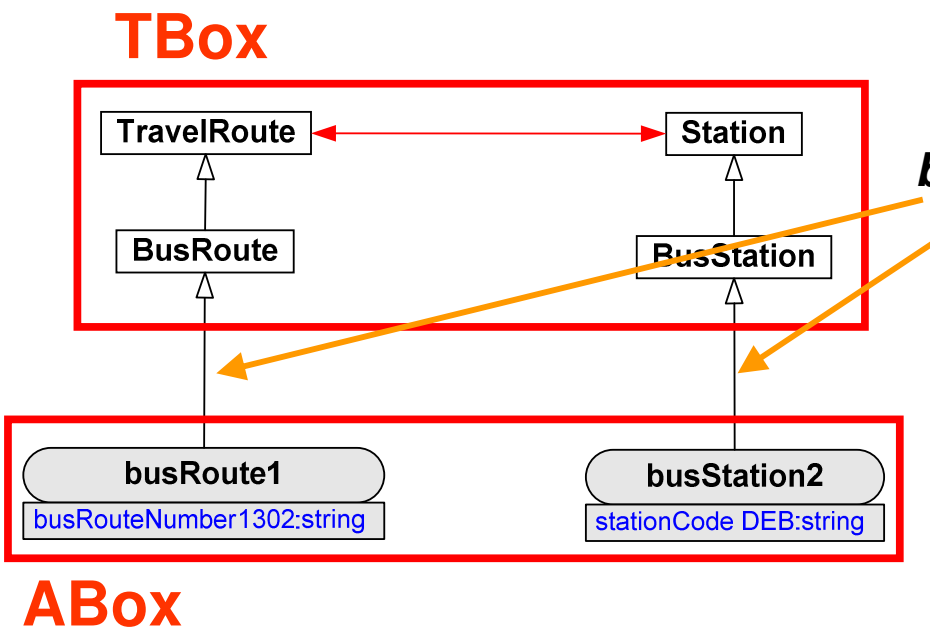

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Classes and Individuals

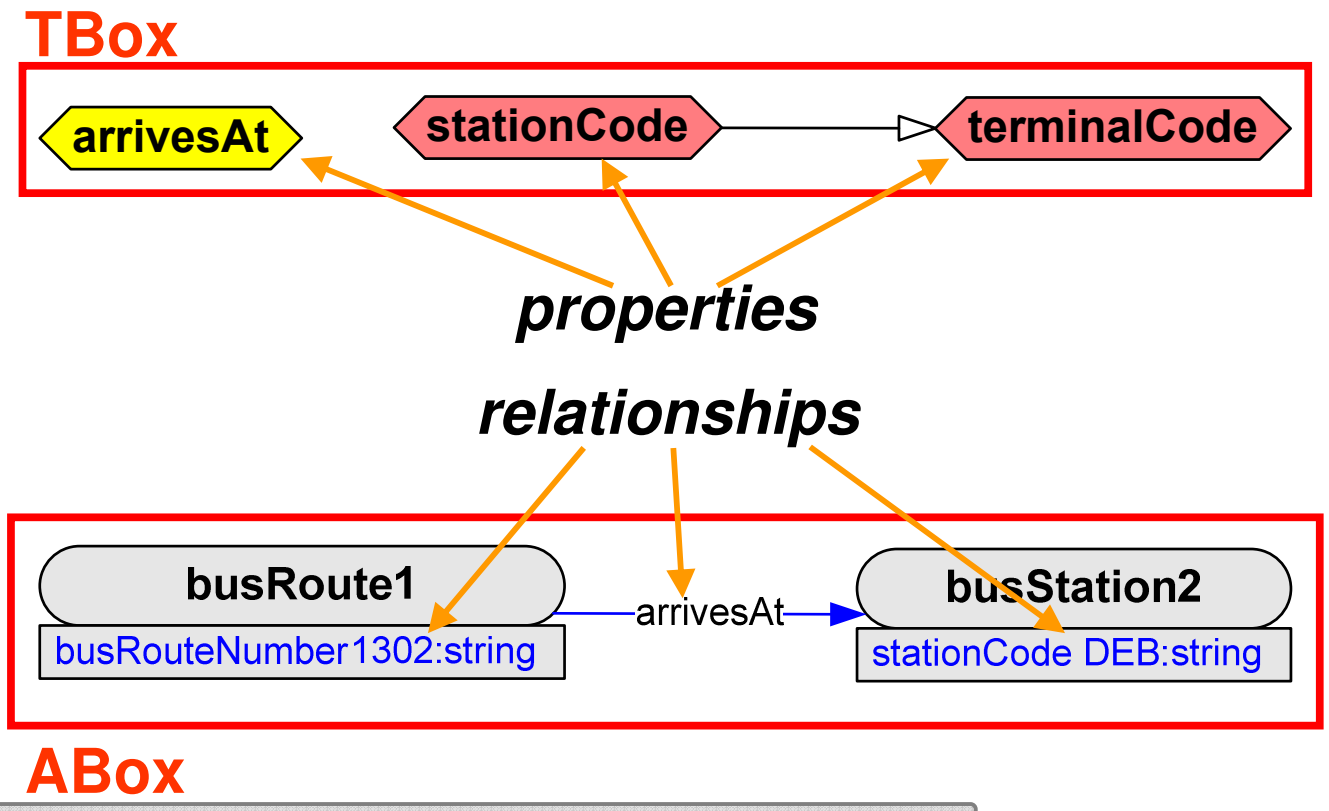



***"Instance of" relationships create a bridge between the TBox and the ABox.***

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Properties and Relationships



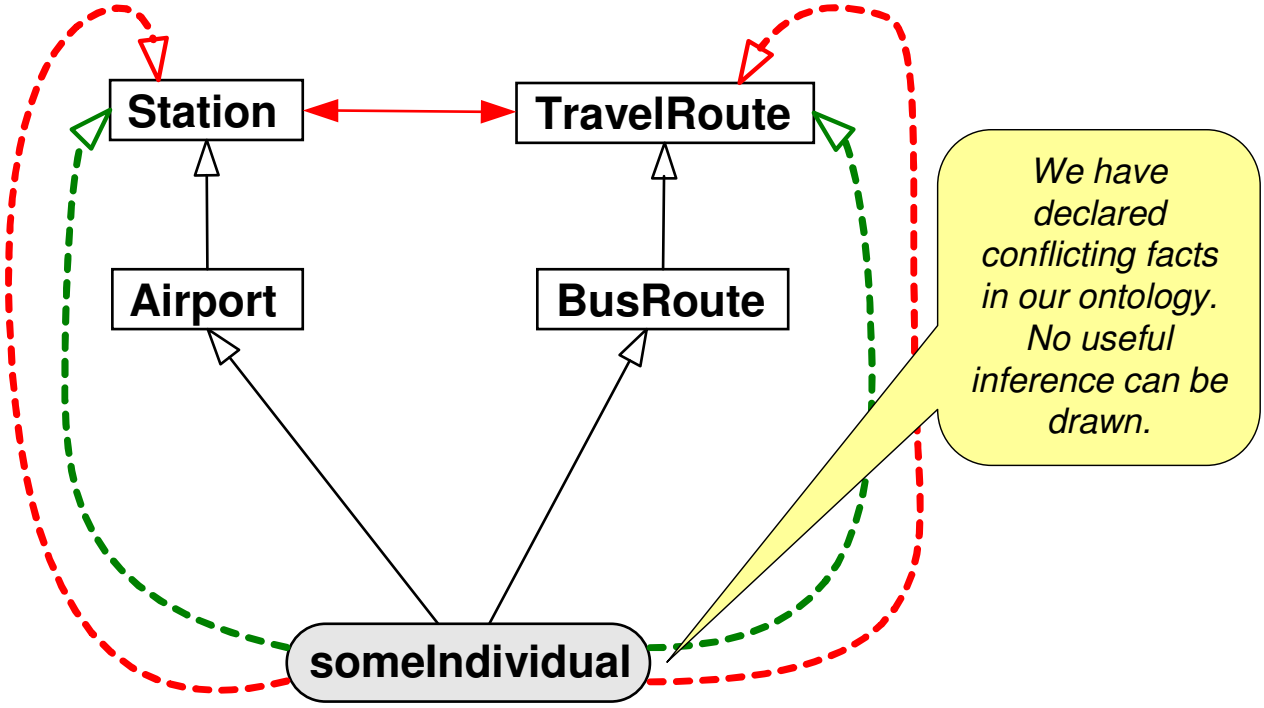

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Inconsistent Ontologies




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Inconsistent Ontologies

- The reasoner will still run queries on an inconsistent ontology, but may return nonsensical results.
- The **`isConsistent()`** taxonomic query can be used to query for consistency.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 3

# Module Review

- Instructor led review session
- Pop-quiz is optional!

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# OWL-DL Modeling with RDF

## Module Four: More Properties

### Special Property Types
### Property Domain and Range
### Properties and the Open world

CU-003: OWL Modeling with RDF. © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Agenda

| | Monday, Feb 27 | Tuesday, Feb 28 | Wednesday, Mar 01 |
|---|---|---|---|
| 8 am | | | |
| 9 00 | **Module 1:** *Intro to OWL* | **Module 3:** *Equivalence, Disjointness, and Subsumption* | **Module 5:** *Complex Classes* |
| 10 00 | | | |
| 11 00 | **Module 2:** *Individuals, Properties, and Classes* | **Module 3:** *Equivalence, Disjointness, and Subsumption* | **Module 5:** *Complex Classes* |
| 12 pm | | | |
| 1 00 | **Module 2:** *Individuals, Properties, and Classes* | **Module 4:** *More Properties* | **Module 6:** *How OWL is Different* |
| 2 00 | | | |
| 3 00 | **Module 2:** *Individuals, Properties, and Classes* | **Module 4:** *More Properties* | **Module 7:** *The Complement Module*<br>**Module 8:** *Wrap up* |
| 4 00 | | | |

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Special Property Types

## Inverse, Transitive, Functional, and Symmetric Properties

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Inverse Properties

- Declaring two properties to be inverses means that a declared relationship in one direction implies the inverse relationship in the other direction.
- Inverse properties suggest the same relationship, but with the subject and object reversed.

```
<owl:ObjectProperty rdf:ID="capitalOf"/>
<owl:ObjectProperty rdf:ID="hasCapital">
   …
   <owl:inverseOf rdf:resource="#capitalOf"/>
   …
</owl:ObjectProperty>
```

*Declare an object property* **hasCapital…**

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Inverse Properties

- Declaring two properties to be inverses means that a declared relationship in one direction implies the inverse relationship in the other direction.
- Inverse properties suggest the same relationship, but with the subject and object reversed.

```
<owl:ObjectProperty rdf:ID="capitalOf"/>
<owl:ObjectProperty rdf:ID="hasCapital">
   …
   <owl:inverseOf rdf:resource="#capitalOf"/>
   …
</owl:ObjectProperty>
```

*…and make it the inverse of* **capitalOf**

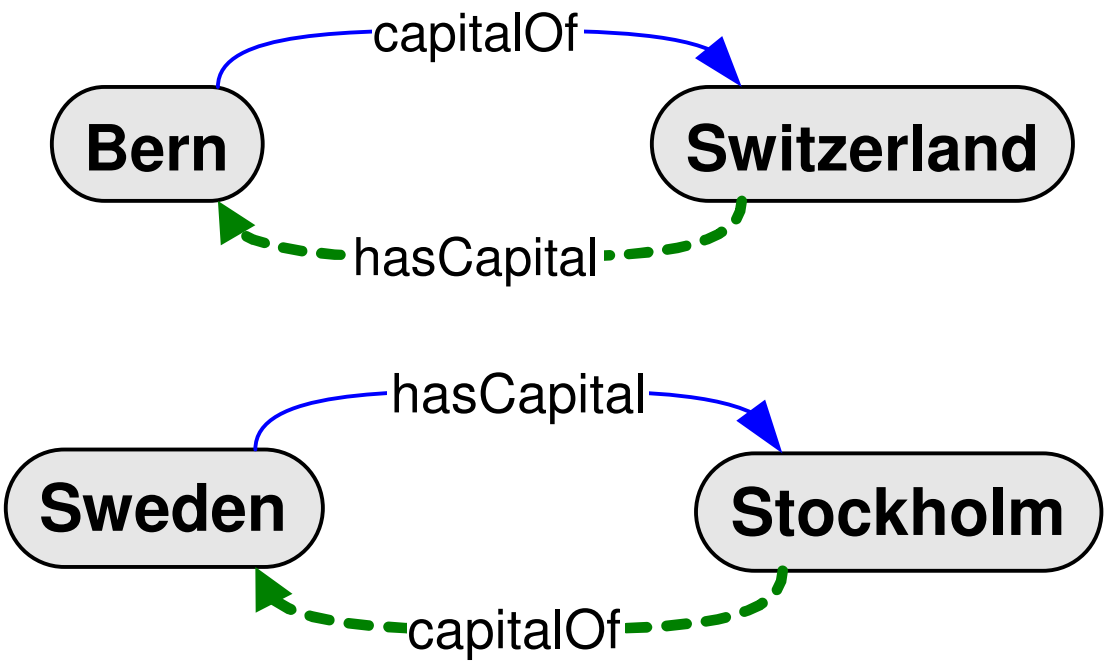
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Inverse Properties




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Special Property Types

- In addition to being either object or datatype properties, properties can optionally have three other attributes:
  - transitive
  - functional
  - symmetric
- These special property types are used to infer property relationships in the ontology other than the one being declared.

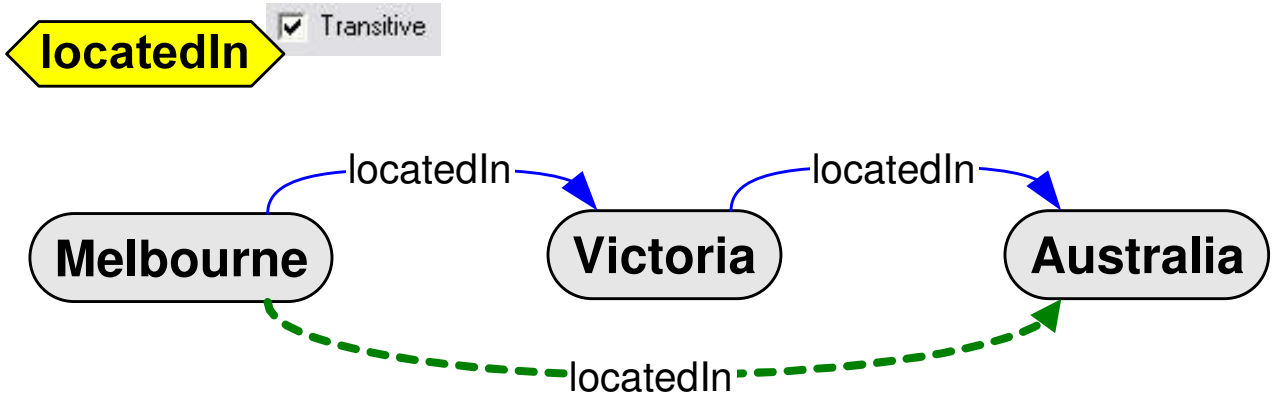
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Transitive Properties




```
<owl:ObjectProperty rdf:ID="locatedIn">
   <rdf:type rdf:resource=
         "http://www.w3.org/2002/07/owl#TransitiveProperty"/>
</owl:ObjectProperty>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

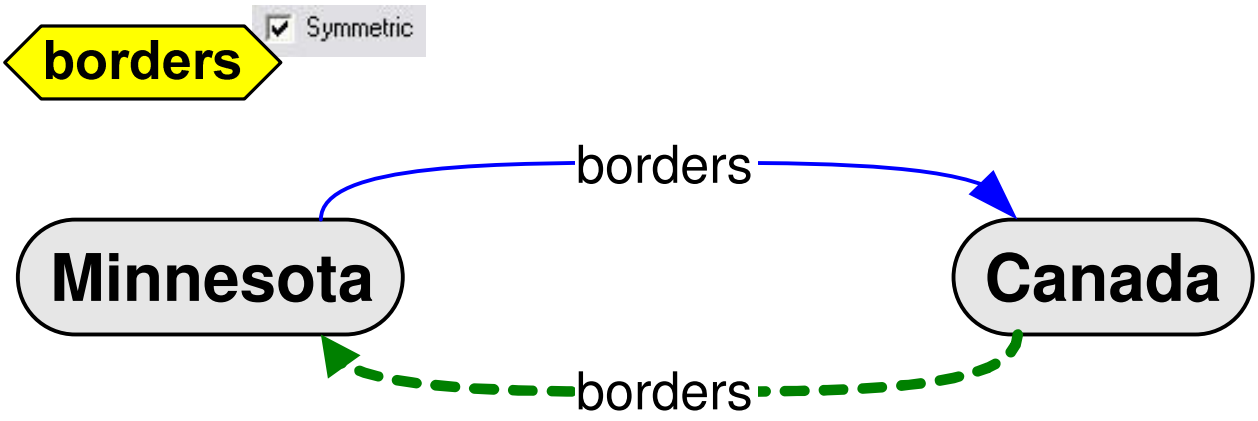
If you are used to ER or OO transitive relationships are often used in a similar way to recursive relationships. Draw an example of how you would do this in an ER diagram.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Symmetric Properties




```
<owl:ObjectProperty rdf:ID="borders">
   <rdf:type rdf:resource=
         "http://www.w3.org/2002/07/owl#SymmetricProperty"/>
</owl:ObjectProperty>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

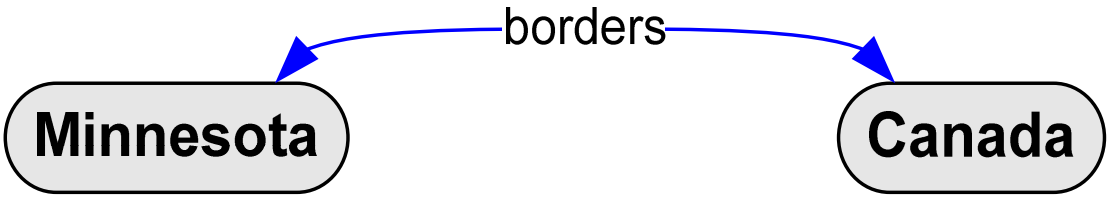
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Symmetric Properties

*When a property is declared symmetric, Construct will change the property link to a double arrow:*




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Functional Properties

- This gets a little trickier.
- If a property is functional, then for any given subject, there can be only one object.
  - *e.g., a person can only have one biological mother.*
- This seems simple enough, except for the fact that ***this does not act as a constraint***.
- So how does it work?

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Functional Properties

- In our ontology, we would like to declare that a region can have at most one capital.

```
<owl:ObjectProperty rdf:ID="hasCapital">
  <rdf:type rdf:resource=
       "http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:ObjectProperty>
```

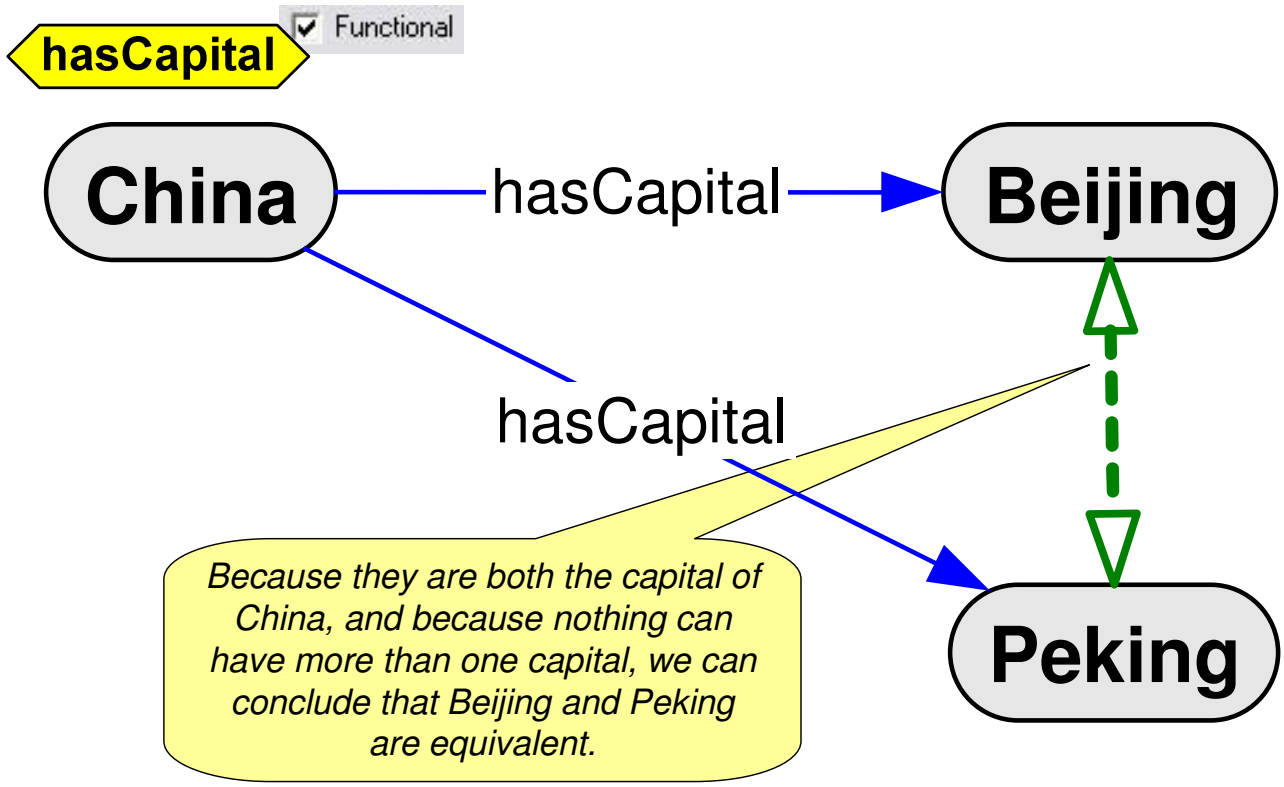
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Functional Properties



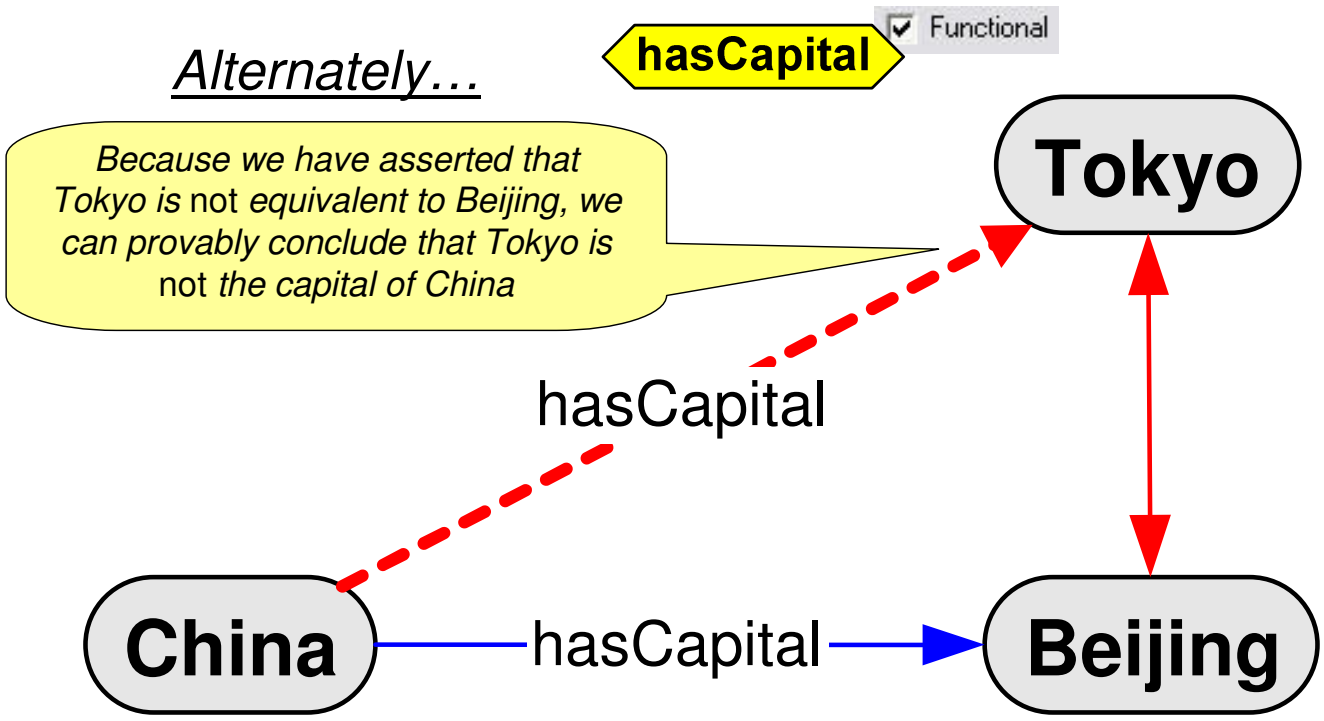

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

i.e This not like a cardinality constraint. You can have more than one hasCapital property relationship but what that means is that the two objects are considered equivalent.

This seams weird to us ER modelers but is consistent with how we humans make sense of the world.

Student Notes:

# Functional Properties

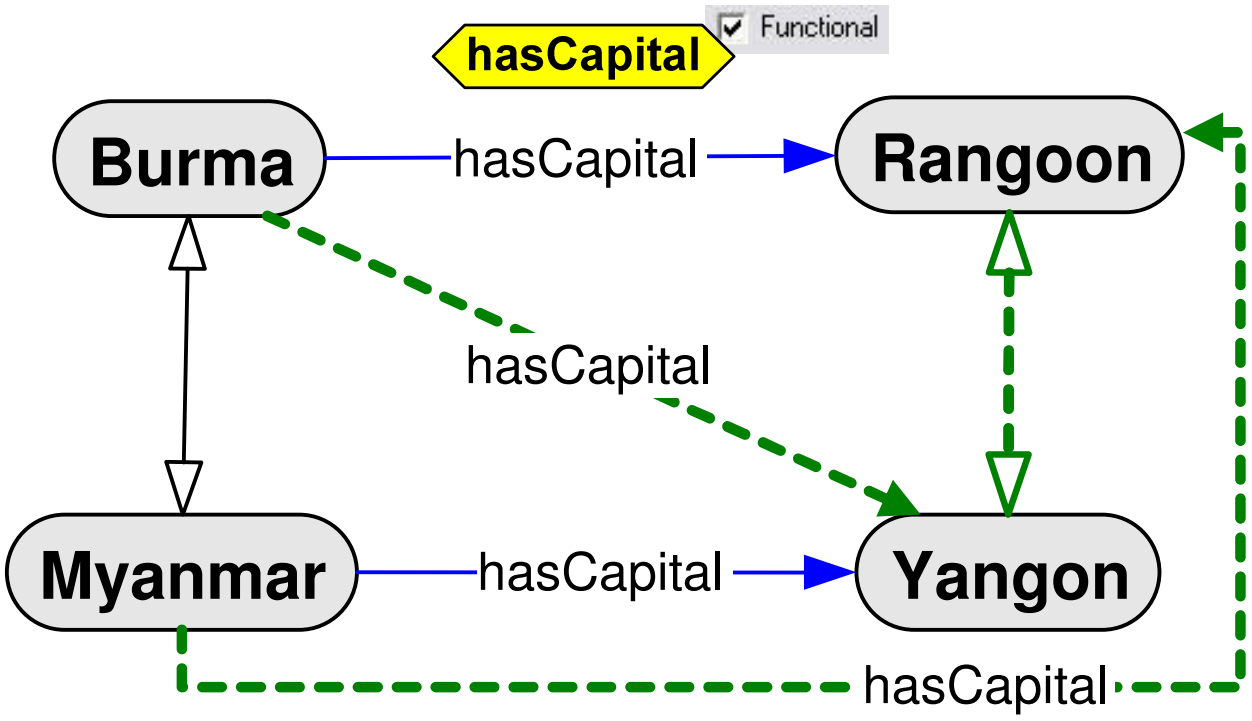



 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

This is something to look for in each construct. You can usually infer both a positive and negative fact.

Student Notes:

# Functional Properties




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Remember that has Capital is functional. It can only have one subject for a given object. Burma can only have one capital.

Student Notes:

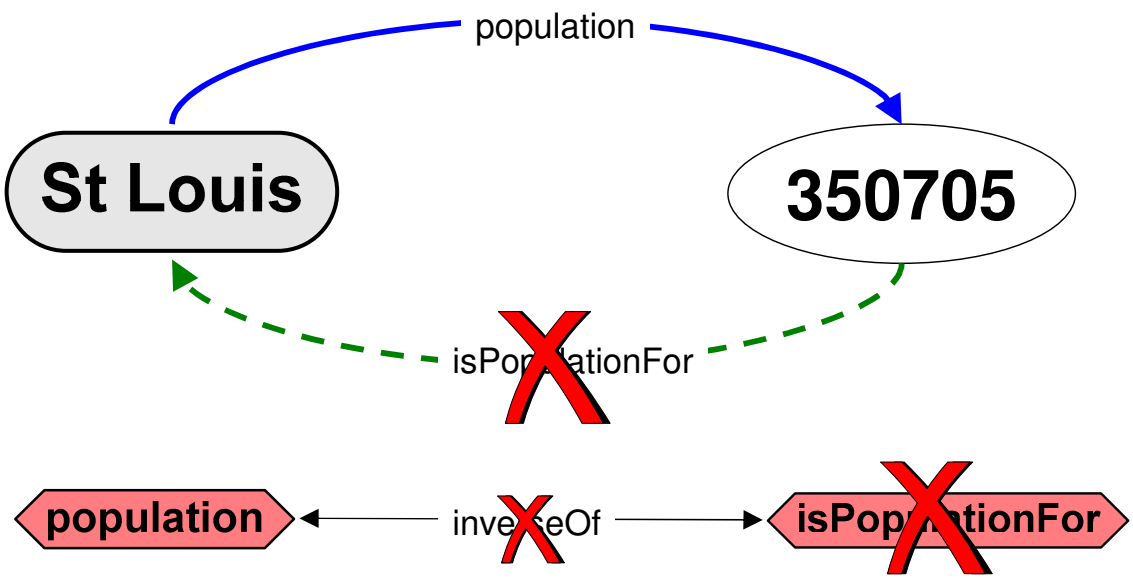
# Datatype Properties

- Remember that instead of relating an individual to an individual, a datatype property relates an individual to a literal data value.
- Literals, however, ***cannot appear as the subject of a property relationship.***
- This has some significant implications…

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

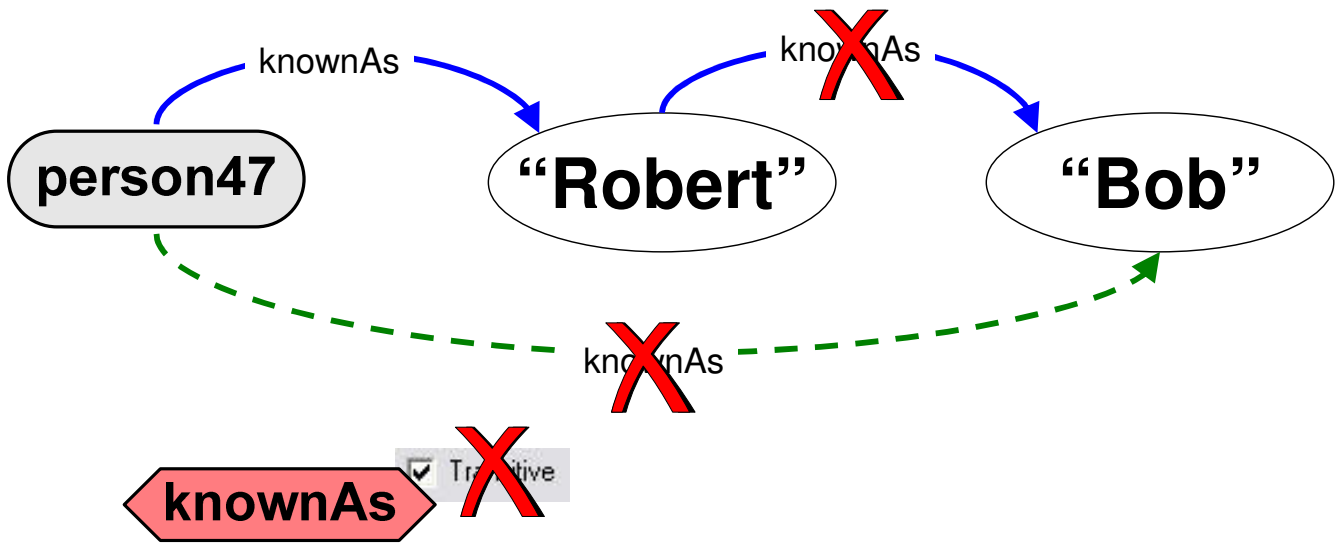
# Datatype Properties ***Cannot*** Have Inverses




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Datatype Properties ***Cannot*** Be Transitive



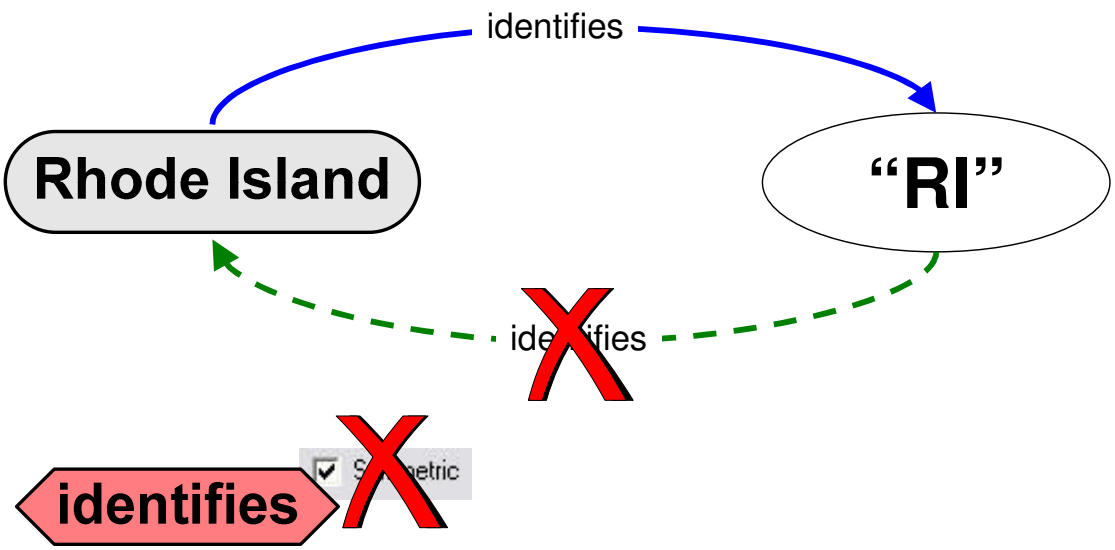

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Datatype Properties ***Cannot*** Be Symmetric




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Datatype Properties ***Can*** Be Functional

- Datatype properties can be functional.
- But what does this really mean?

```
<owl:DatatypeProperty rdf:ID="name">
   <rdf:type rdf:resource=
         "http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:DatatypeProperty>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

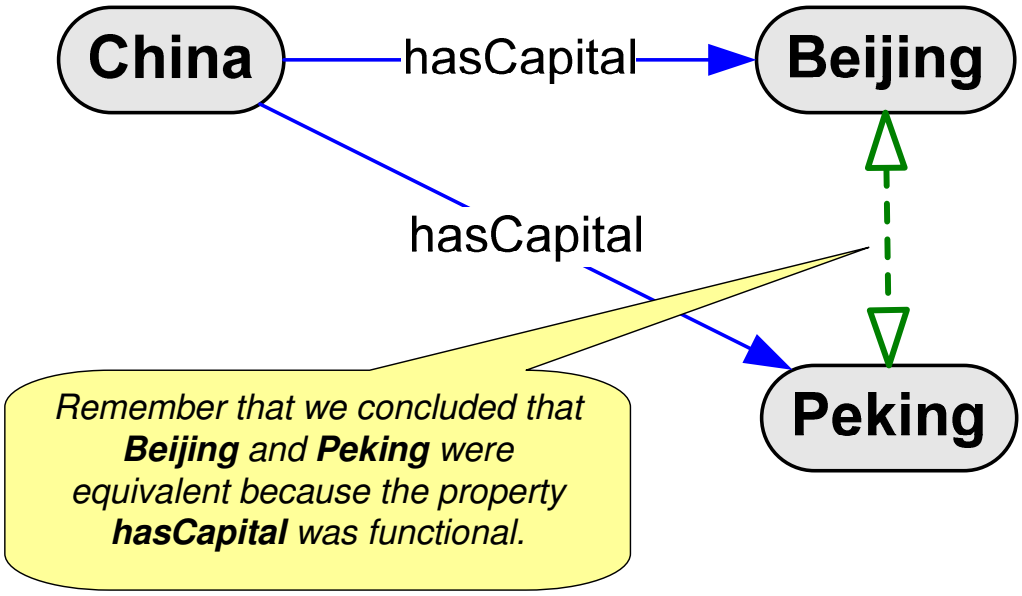
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Functional Datatype Properties

- Let's start with something like our earlier example…




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

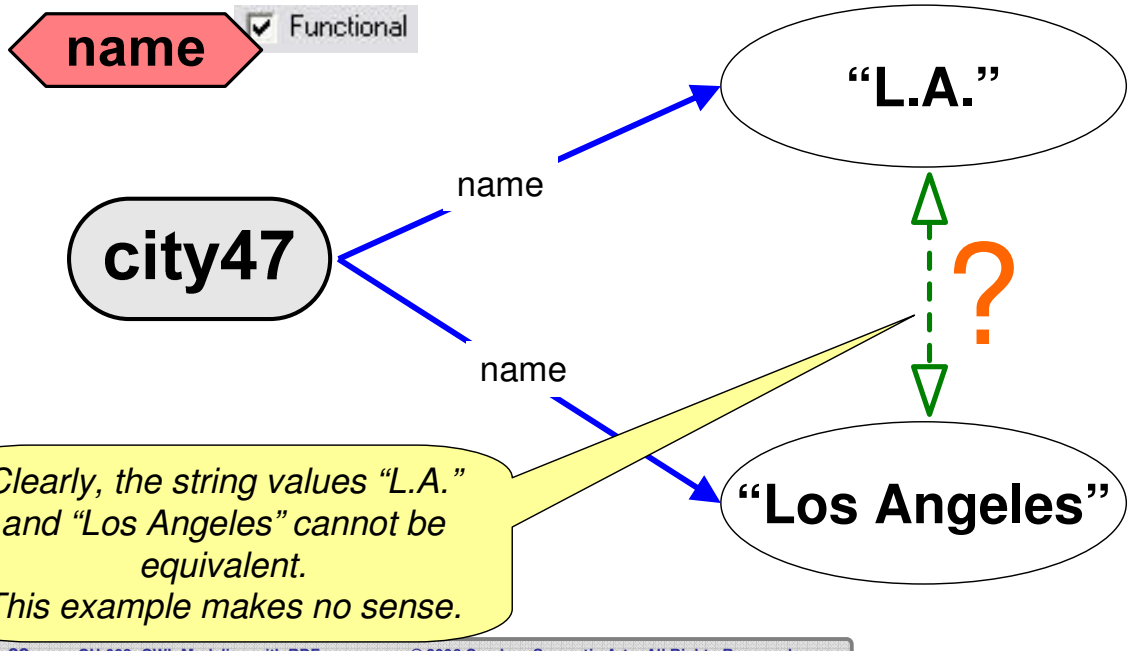
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Functional Datatype Properties

- So what does this mean for data values?




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Again name is functional

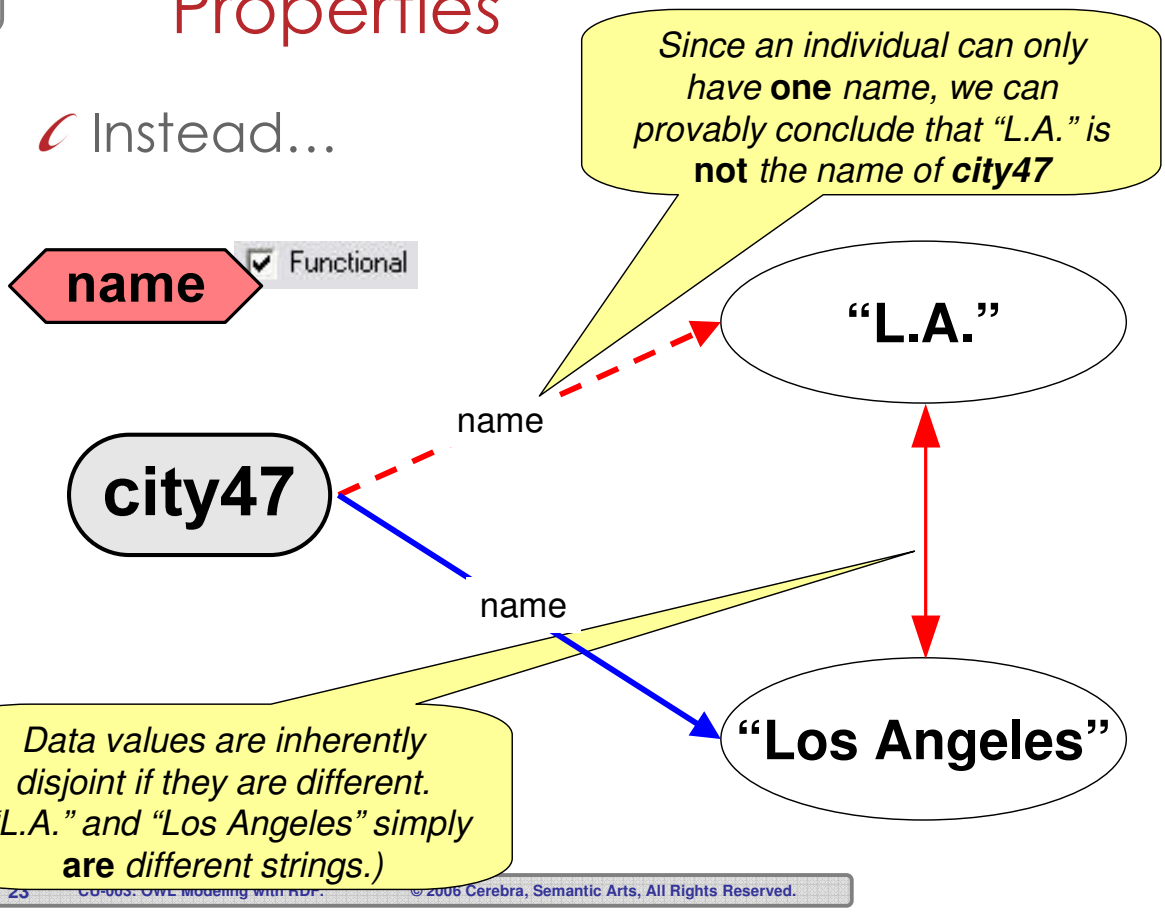
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Functional Datatype Properties

- Instead…





© 2006 Cerebra, Semantic Arts, All Rights Reserved.

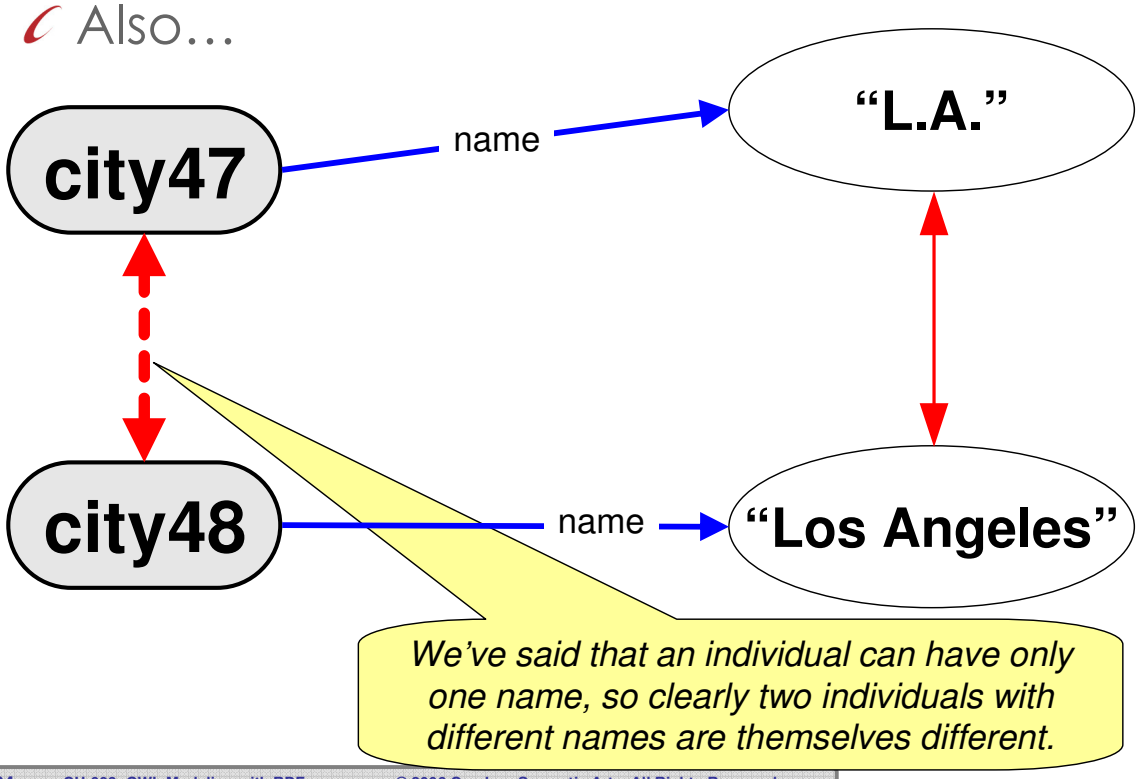
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Functional Datatype Properties

- Also…




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Functional Datatype Properties

- This is probably more like what you expected to see.
- A functional datatype property means that there is ***at most one*** value of that property for a given individual.
- As the last example showed, however, **name** is clearly not a good candidate for a functional datatype property.
- This introduces a…

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Construct Issue

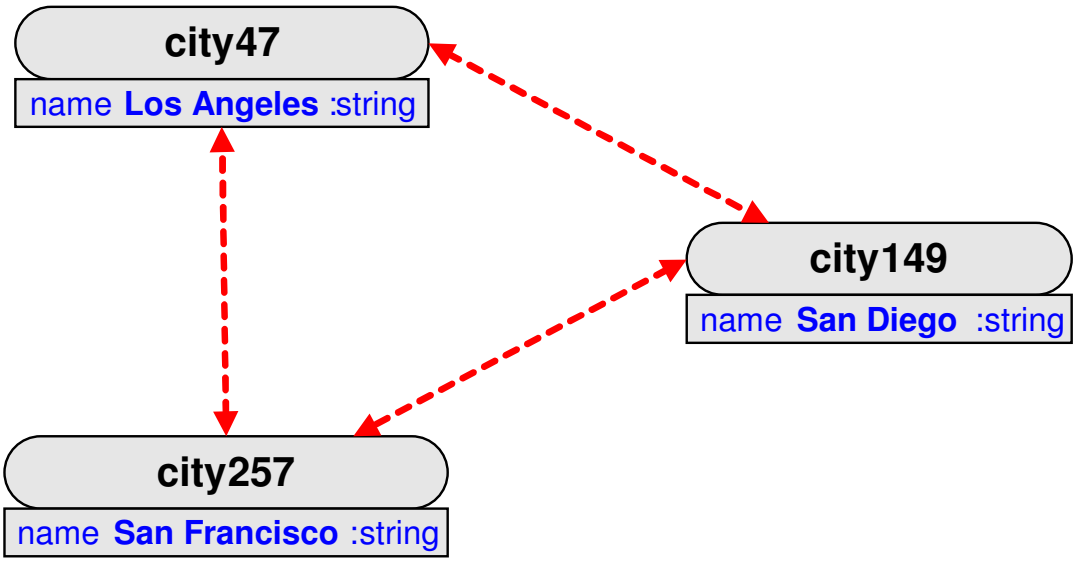
- In Construct, datatype properties are functional.
  - There is no way to select or deselect this option.
- This is not a limitation of either OWL or the Cerebra Server inference engine.
- *Neither the OWL standard nor Cerebra Server has any problem with non-functional datatype properties.*
- For now we will simply live with this limitation.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# What This Means…

- In your ontology, if individuals have different values for a datatype property, they are ***provably disjoint***:

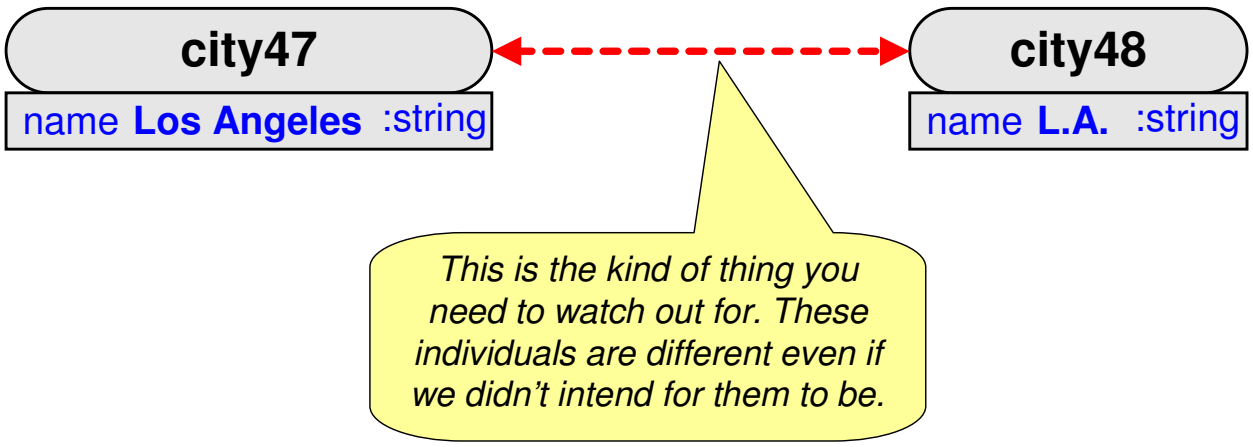



 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# What This Means…

- In your ontology, if individuals have different values for a datatype property, they are ***provably disjoint***:




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Another way to put this is that if we are importing or mapping our instance data from a data base. If you map a key field in a relational data base then all your instances are going to be considered to be disjoint. We'll mention this again in the mapping module.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# More Properties

## Properties as Sets
## Properties and the Open World

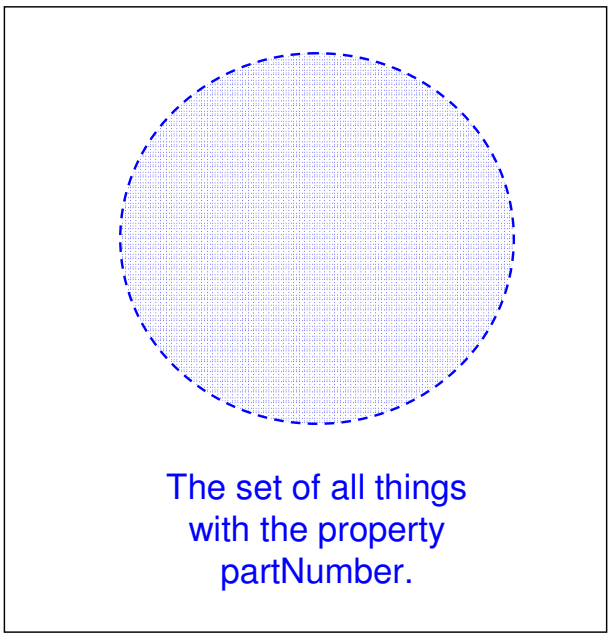
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# What is a property ?



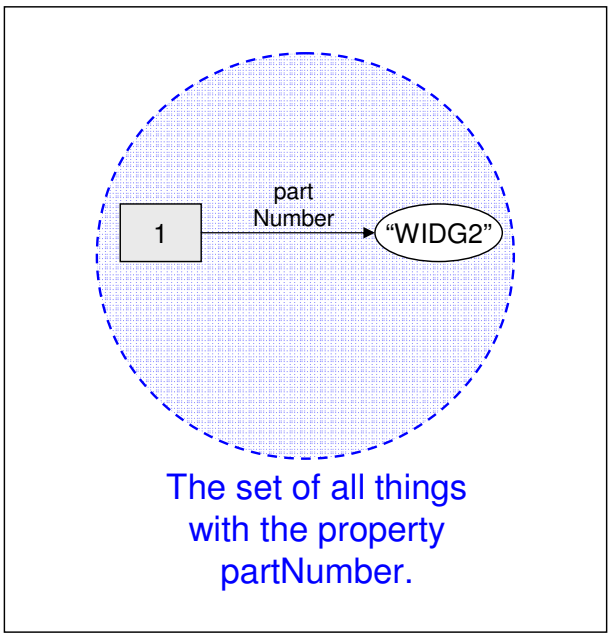

partNumber

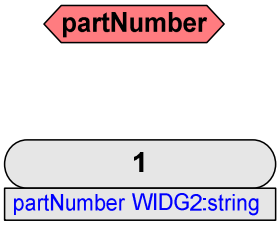
Adding a property to an ontology is equivalent to defining an anonymous set of all things that have that property.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# What is a property ?




Describing an individual with that property is equivalent to assigning it to this anonymous set.

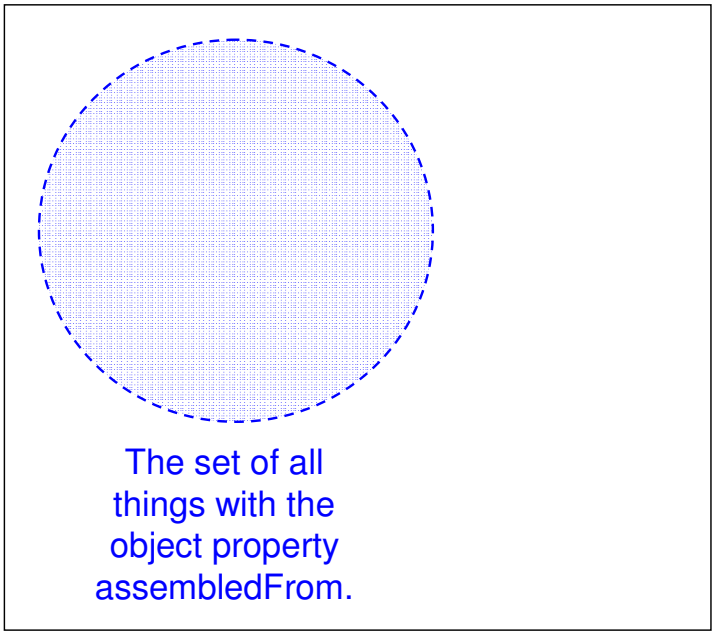
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# The same is true of Object Properties



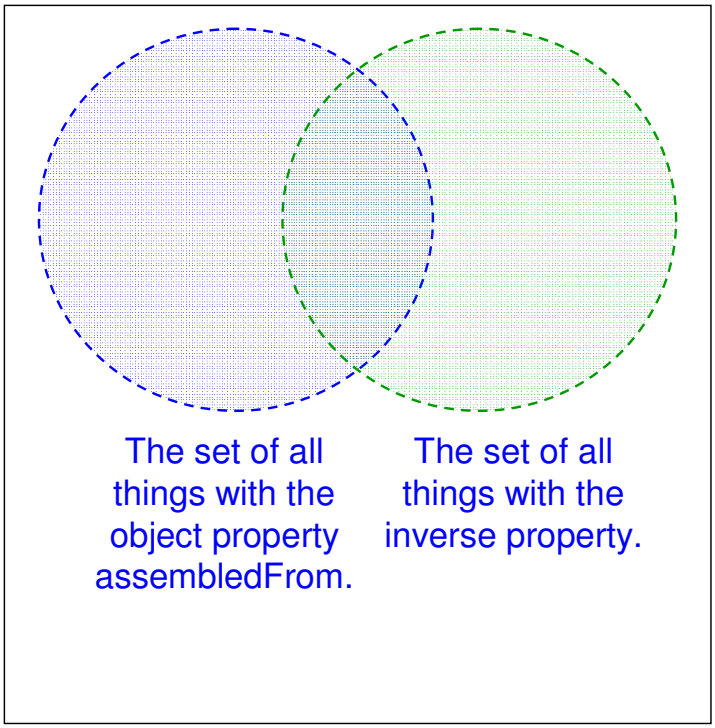

**asembledFrom**

Defines an anonymous class of all individuals who have that object property.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Inverse Properties




assembledFrom

inverseOfAssembledFrom

Every object property has an implied inverse property. (In this example, an implied partOf property.)

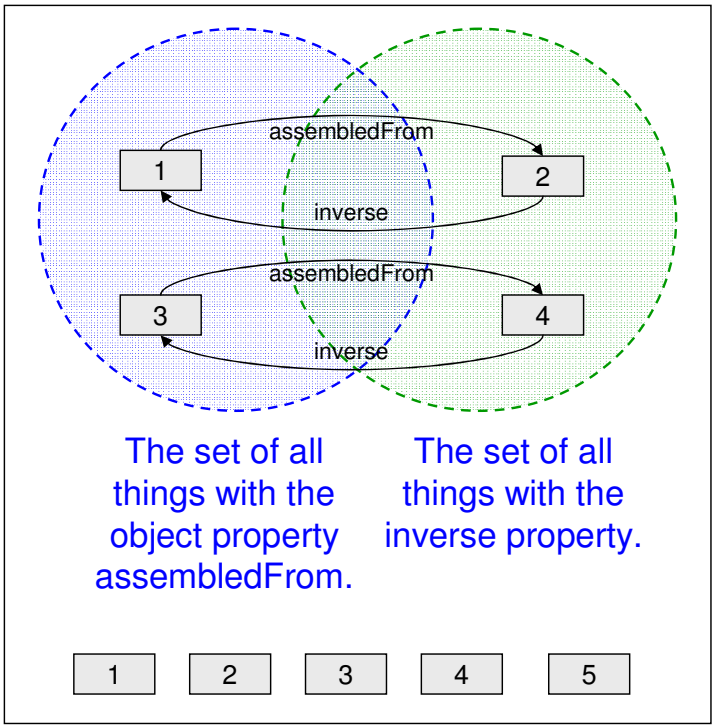
The inverse property can be considered to describe the set of individuals that have that inverse property.

In other words, the set of all things that are the object (target) of assembledFrom.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Inverse Properties




assembledFrom

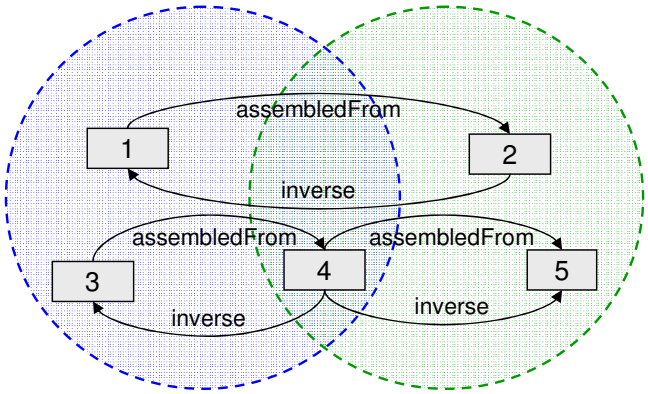
inverseOfAssembledFrom

Associating instances using the object property categorizes those instances into one or the other of the classes.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Inverse Properties




The set of all things with the object property assembledFrom.

The set of all things with the inverse property.

Thing

**assembledFrom**

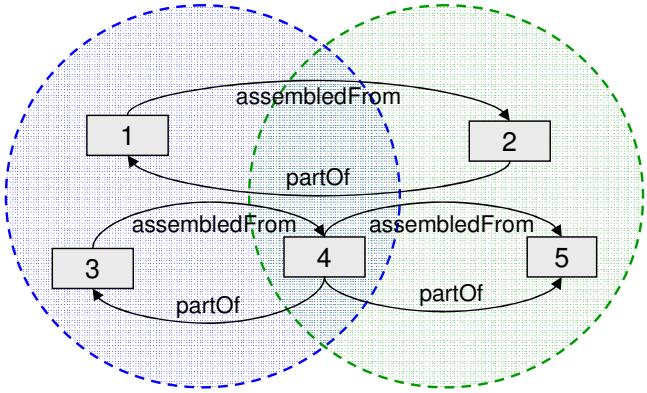
**inverseOfAssembledFrom**

An individual that is both the object and the subject of the property is categorized in the intersection of the two classses.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Named Inverse Properties




The set of all things with the object property assembledFrom.

The set of all things with the property partOf.

Thing

**assembledFrom**

**partOf**

We have the option of naming the inverse property so we can refer to it in the ontology.

```
<owl:ObjectProperty rdf:ID="assembledFrom">
   <owl:inverseOf rdf:resource="#partOf"/>
</owl:ObjectProperty>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

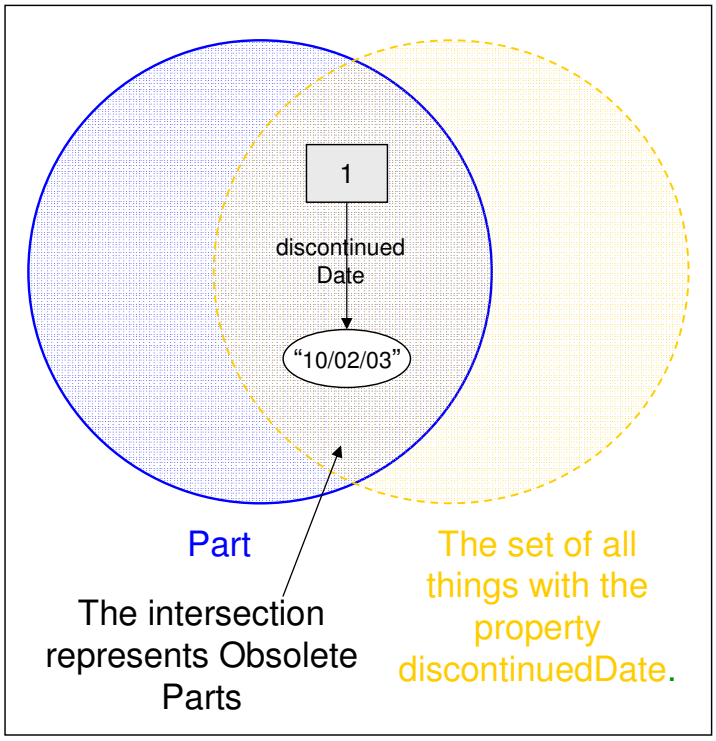
# This is weird but is it useful?

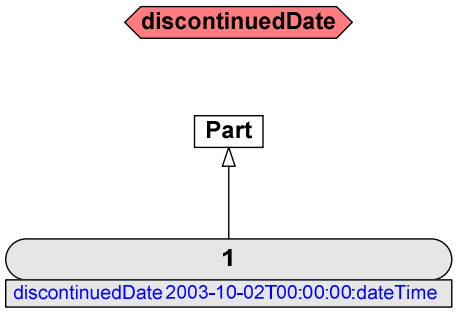
- We are used to thinking of properties as being part of a class in the same way that columns are part of a table in a relational data base.
- To an open world reasoner, properties appear as characteristics of an individual that include or exclude that individual from some category.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Example – Obsolete Parts




The reasoner can categorize individuals based on both declared class membership and properties.

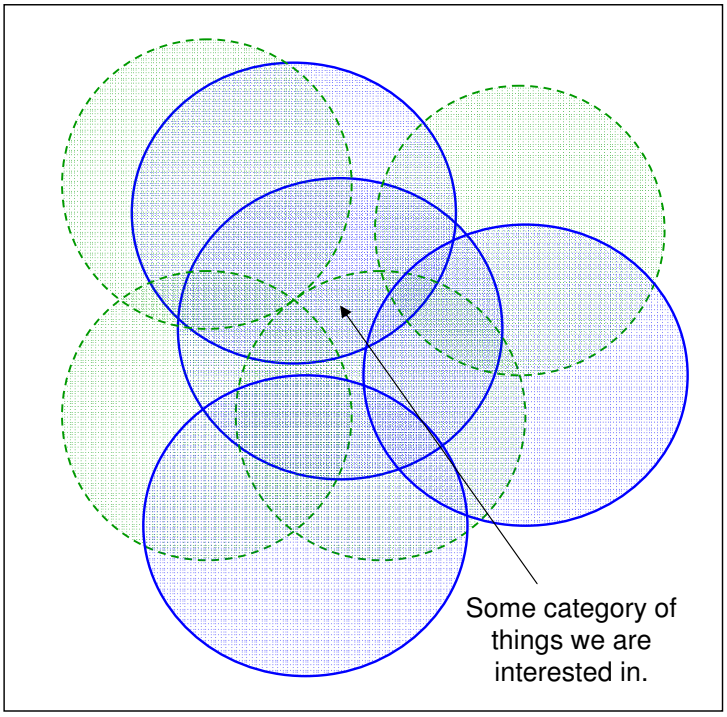
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# An ontology as intersecting sets




You can think of an ontology as a series of a priori categories (or sets) created from base classes and properties.

Each category overlaps with every other category creating multiple intersections. Each intersection represents another possible category in the domain.

One of our jobs as ontologists is to define and name the particular intersections that are of interest to us, so that in turn we can identify individuals that are members of that category.

We do this in OWL using restriction classes. More on this later.

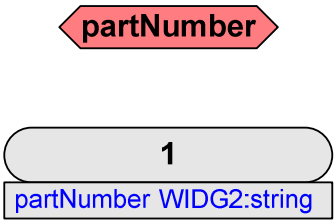
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Properties in an Open World




```
for $i in instances()
for $d in datavalues()

where every $m in models() satisfies
  related($i,partNumber,$d,$m)

return {$i}
```

```
<RESULT>
http://www.cerebra.com/construct#1
</RESULT>
```

```
for $i in instances()
for $d in datavalues()

where some $m in models() satisfies
  related($i,partNumber,$d,$m)

return {$i}
```

```
<RESULT>
http://www.cerebra.com/construct#1
</RESULT>
```

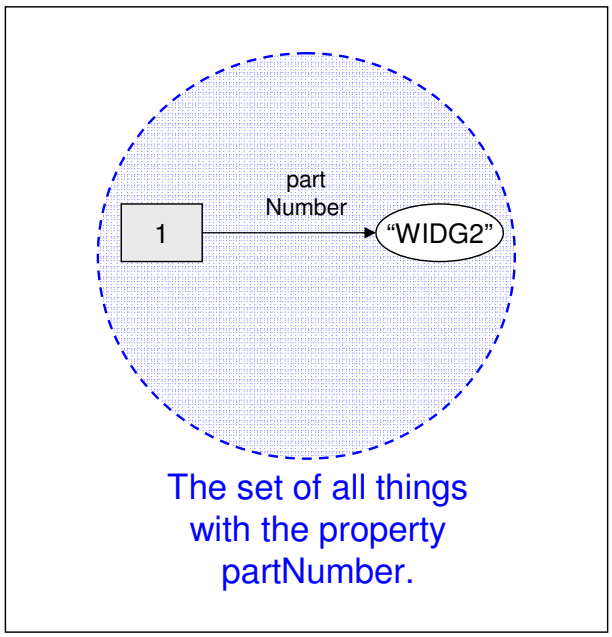
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

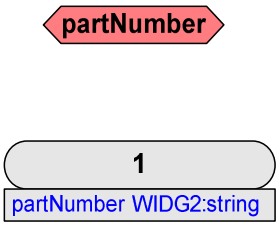
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Properties in an open world




In this ontology there is only one possible model.

Individual 1 is provably a member of the set of things that have a partNumber.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Properties in an open world

**partNumber**

**1**

```
for $i in instances()
for $d in datavalues()

where every $m in models() satisfies
  related($i,partNumber,$d,$m)

return {$i}
```

```
<RESULT>
</RESULT>
```

```
for $i in instances()
for $d in datavalues()

where some $m in models() satisfies
  related($i,partNumber,$d,$m)

return {$i}
```

```
<RESULT>
http://www.cerebra.com/construct#1
</RESULT>
```

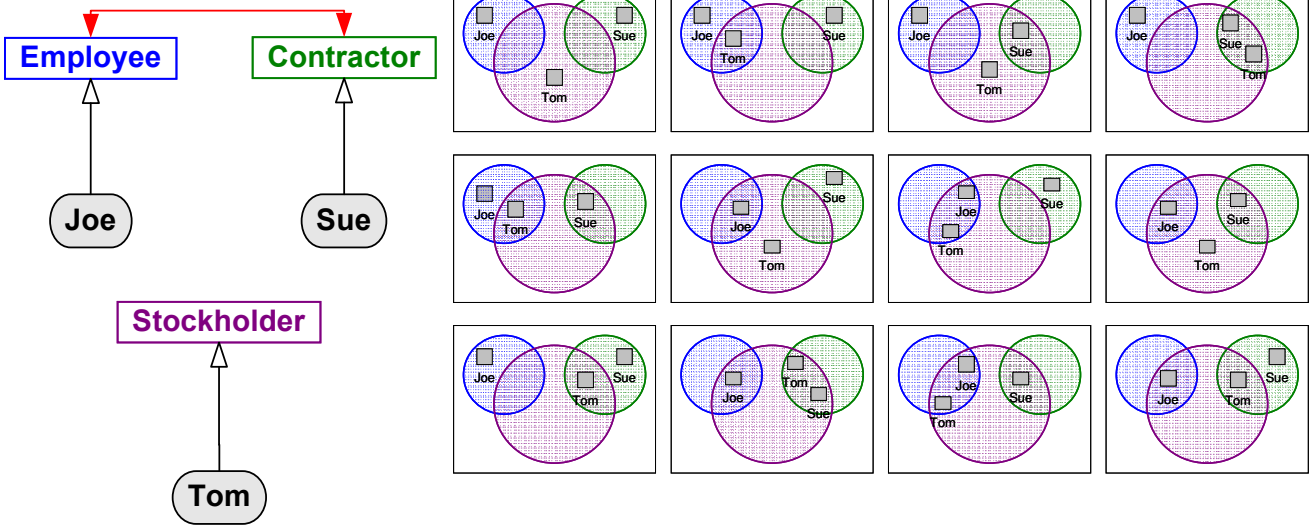
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# The Open World Revisited




In the first lab we saw how unasserted class membership could generate multiple models in an open world system.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

It's the things that you don't assert that generate possible alternative models

We never said whether

Tom was an employee or not

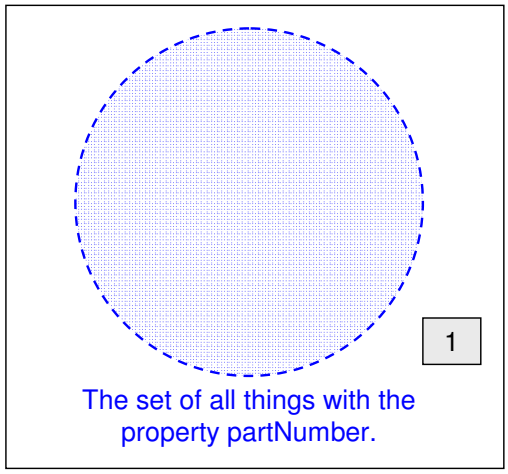
Tom was a Contractor or not

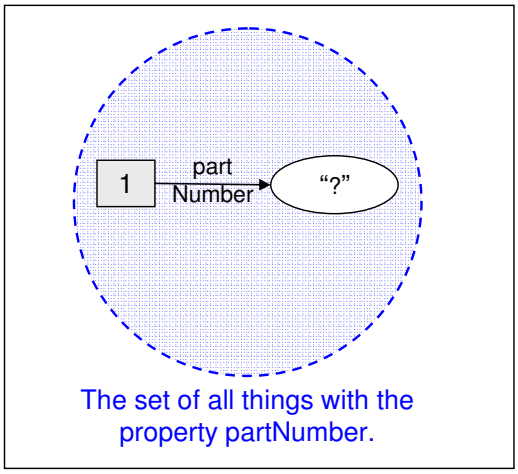
Sue was a Stockholder or not

Joe was a stockholder or not.

## Student Notes:

# Properties in an open world




There are two possible models for this ontology.

- In one, the individual does not have the property partNumber.
- In the other, the individual has the property but that property is as yet unknown.

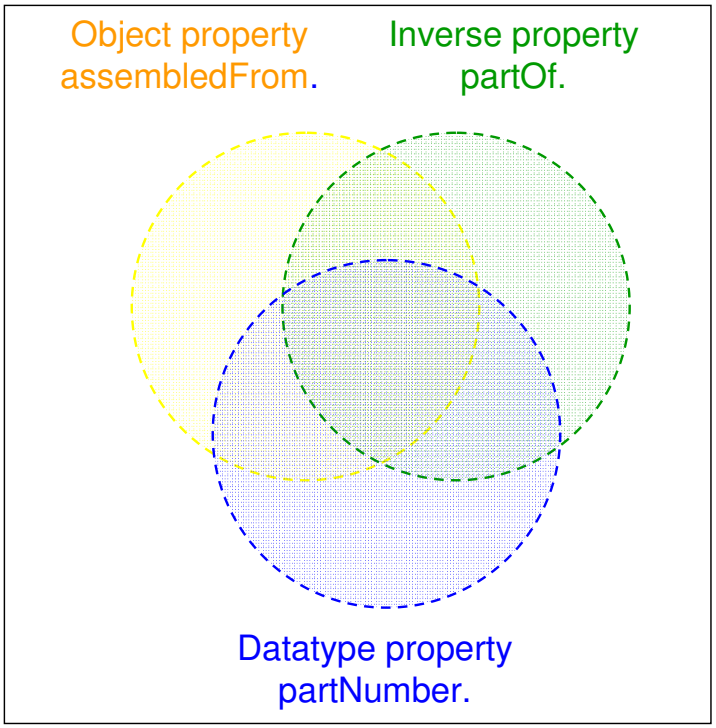
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

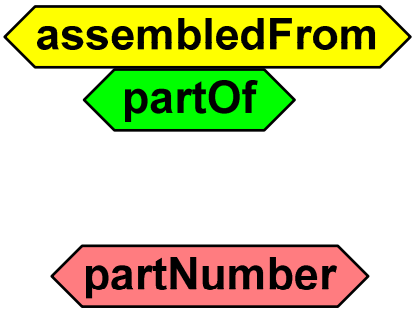
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Adding properties adds more sets



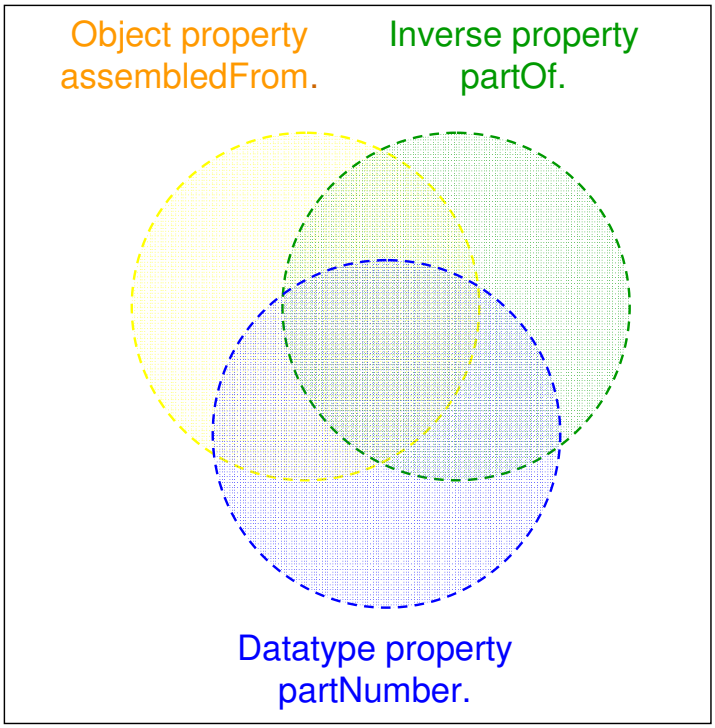

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Adding an instance creates multiple possible models




assembledFrom

partOf

partNumber

1

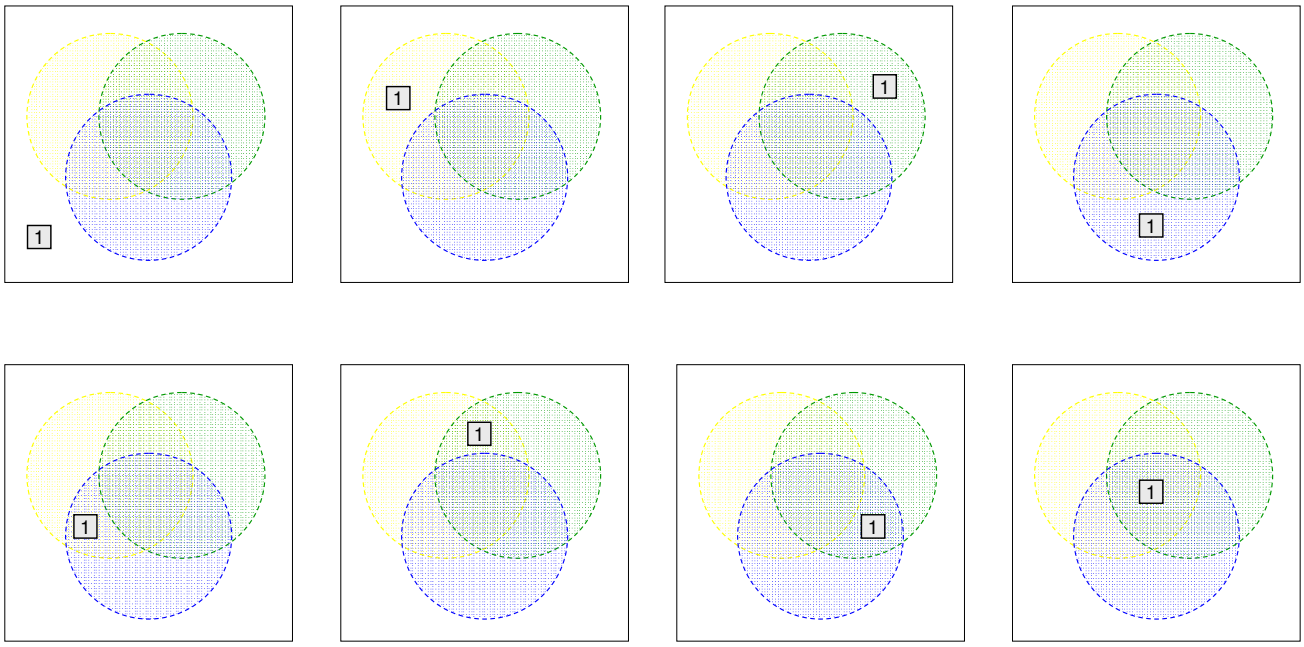
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# In fact, it introduces 8 models




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# In Summary

- With an open world reasoner we can ask if an individual "provably" has a property (i.e., it has that property in all possible models of the ontology), or "satisfiably" has that property (i.e., it has that property in at least one model of the ontology).
- Effectively, we are asking the reasoner if the instance is provably or satisfiably a member of the set of things with that property.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# In Summary

- The difficulty with this open world assumption as regards properties is that you never know if:
  - 1) An individual has that property but it has not yet been asserted (missing data).
  - 2) The ontology is in fact complete and the individual does not have that property.
- This makes certain types of queries difficult to perform.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

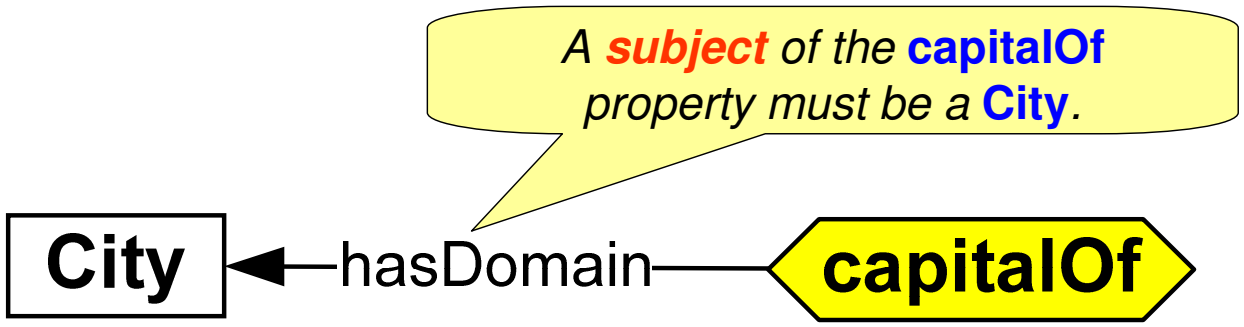
CU-003
02/2006
Module 4

# Property Domain and Range

CU-003: OWL Modeling with RDF. © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Property Domain

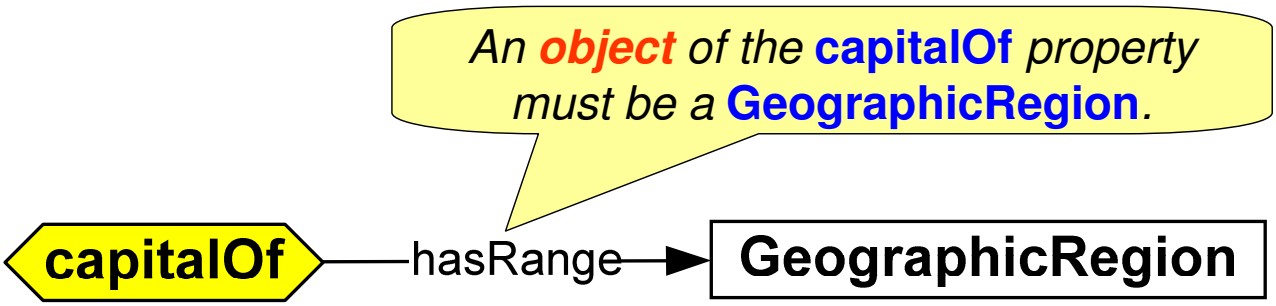



```
<owl:ObjectProperty rdf:ID="capitalOf">
   <rdfs:domain rdf:resource="#City"/>
</owl:ObjectProperty>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Property Range




```
<owl:ObjectProperty rdf:ID="capitalOf">
   <rdfs:range rdf:resource="#GeographicRegion"/>
</owl:ObjectProperty>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Other definitions of Domain and Range

- **Relational:** A domain is a **type**. A set of values which are valid for a particular attribute. (Relational does not have a concept of range.)
- **Mathematical:** The domain of a function is the set of all values which can be **inputs**. The range is the set of all values which can be **outputs**.
- **Object-Oriented/UML:** The domain is the **class** or entity that has an attribute. The range is the **type** of that attribute.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# What does it mean in OWL?

- Traditionally, a domain is thought of as a constraint on a property.
  - e.g., "only cities can be capitals."
- In OWL, however, asserting a domain for a property does ***not*** place a constraint on the individuals that can have that property.
- So what does it do?

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Chris Date defines domain as similar to a type.

Student Notes:

# Property Domain

- Let's begin with an object property from our geography ontology:



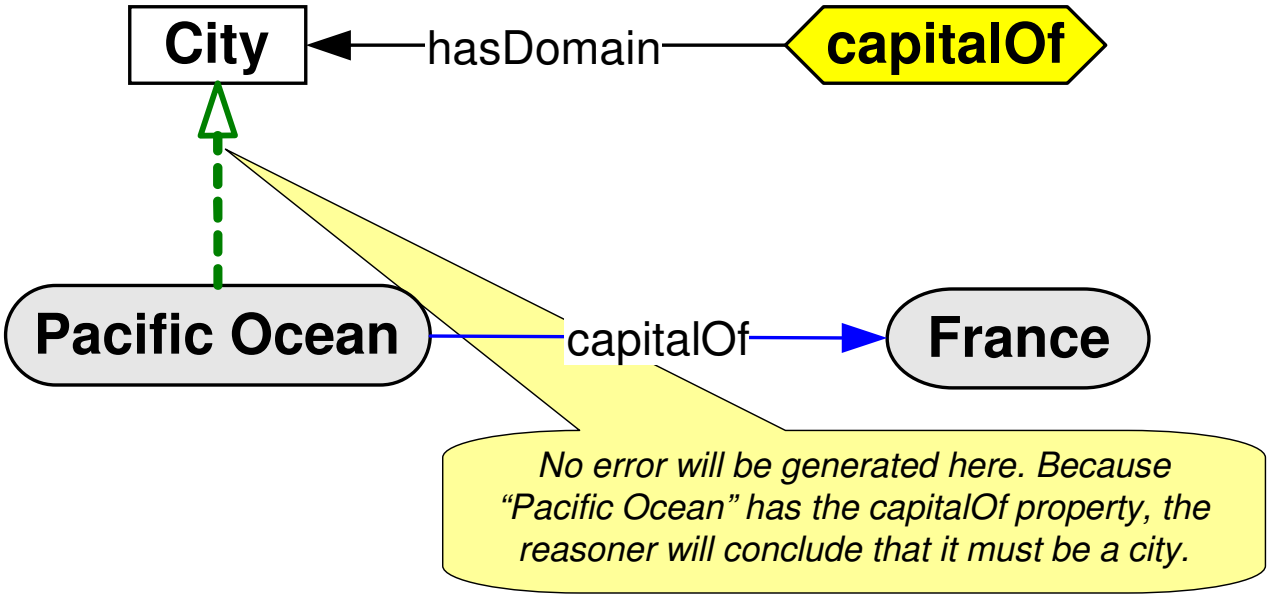

```
<owl:ObjectProperty rdf:ID="capitalOf">
   <rdfs:domain rdf:resource="#City"/>
</owl:ObjectProperty>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# What have we said?

- To illustrate, let's make a nonsense assertion.




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

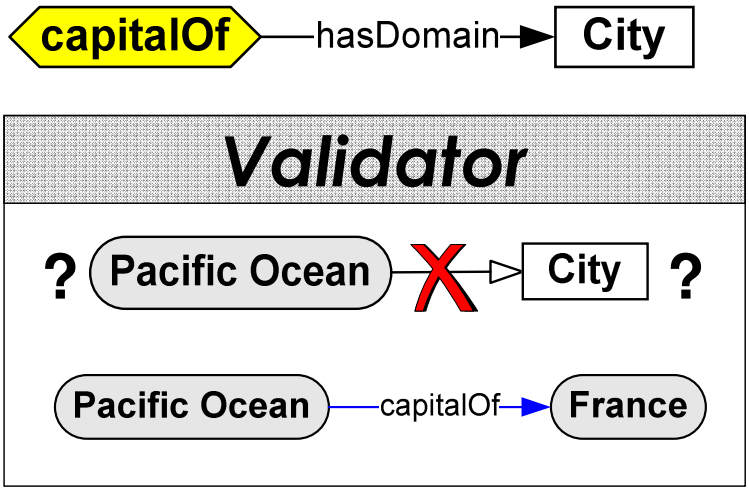
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# What's the difference?

- Traditionally, setting a property's domain constrains the set of things that can have that property:

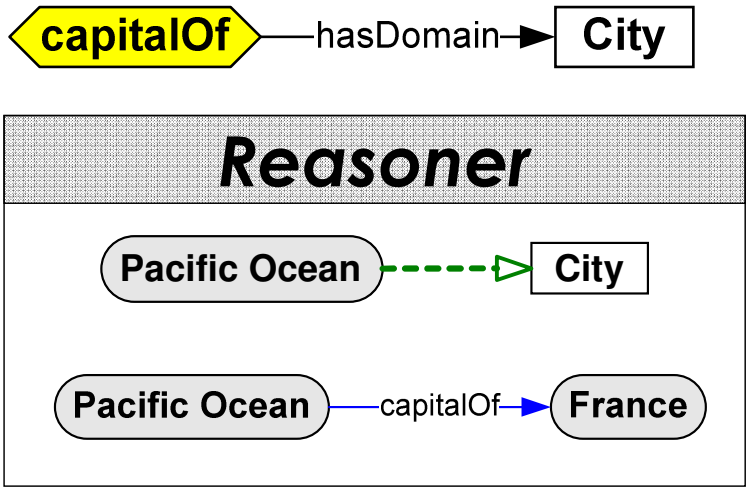



 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# What's the difference?

- Instead, this is how an OWL reasoner works.




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# What are the implications?

- Setting a property's domain does not act as a constraint against which future assertions are checked.
- This fact is both powerful and problematic, and requires us to rethink the approaches used with other technologies.
- We will explore a number of these, starting with…

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

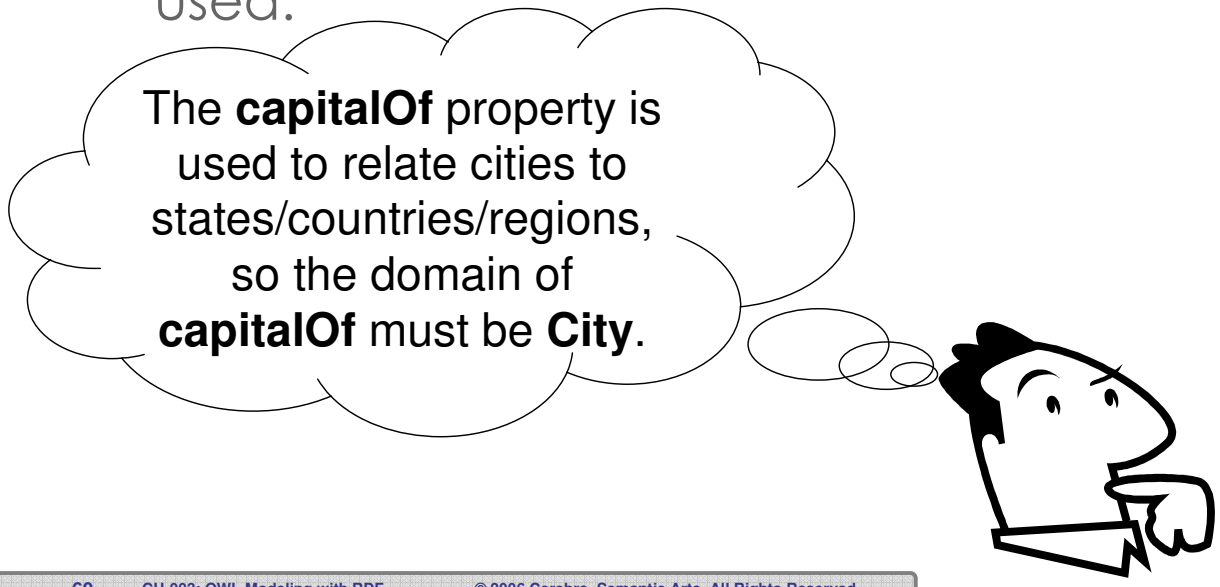
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# The Naïve Approach

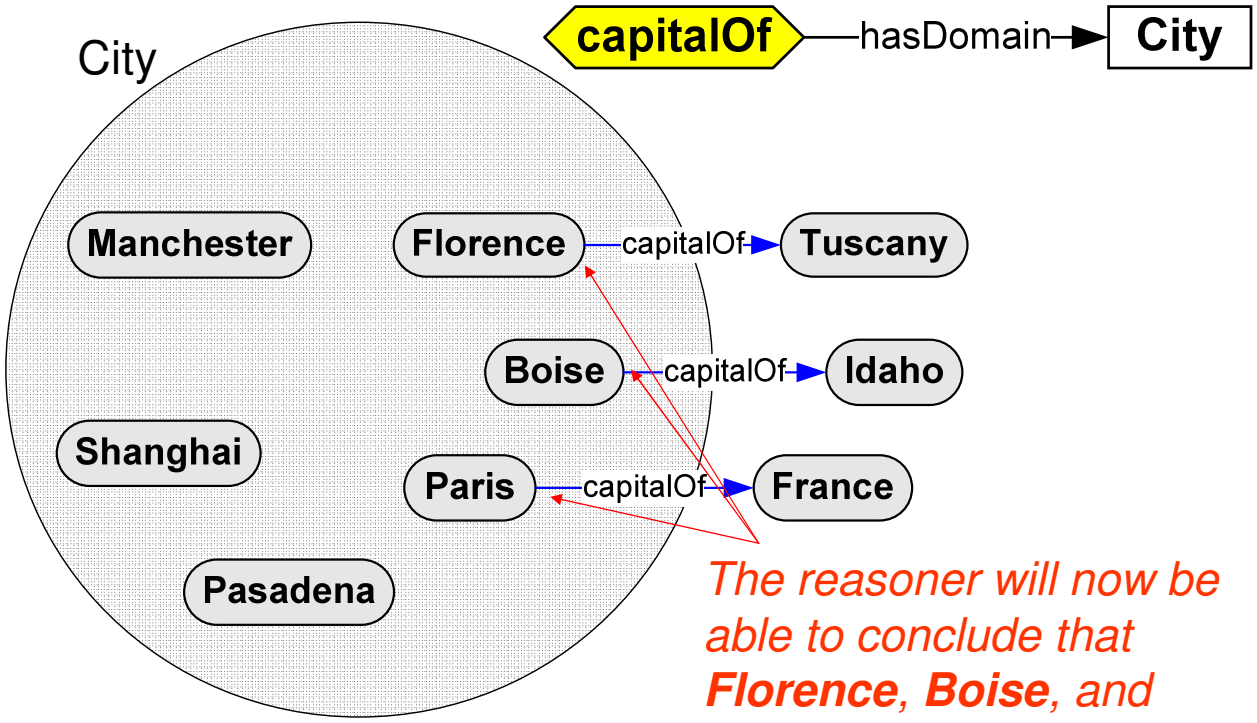
- Traditionally, when modeling relationships, a modeler will base his/her assertions on how the relationship is used:




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# The Naïve Approach




*The reasoner will now be able to conclude that **Florence**, **Boise**, and **Paris** are cities.*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# The Naïve Approach

- Since domain and range do ***not constrain*** possible property relationships, it doesn't make sense to use them as though they do.
- Domain and range are used to ***infer*** class membership based on property relationships.
- *In the last example, our purpose in using the property was **not** to figure out what was a city.*

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# What's really going on…

- Traditionally, a property's domain defined the set of things that could ***potentially*** have that property.
- In OWL, however, a property's domain classifies the set of things that ***do*** have that property.
- Having a property ***makes*** an individual a member of that property's domain.

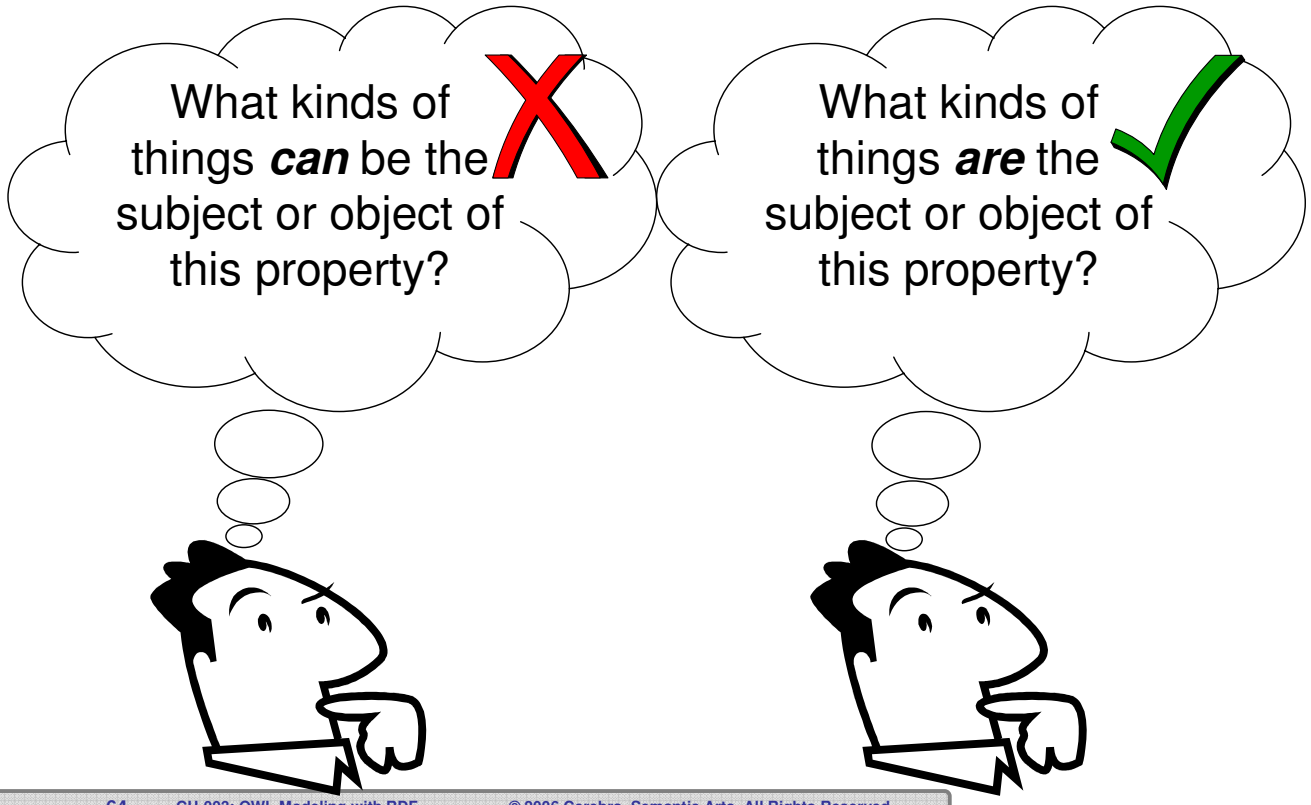
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# How to Assign Domain and Range…



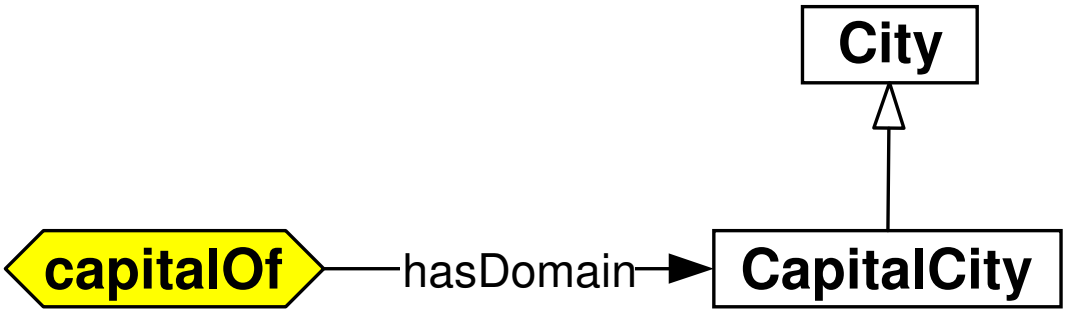

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

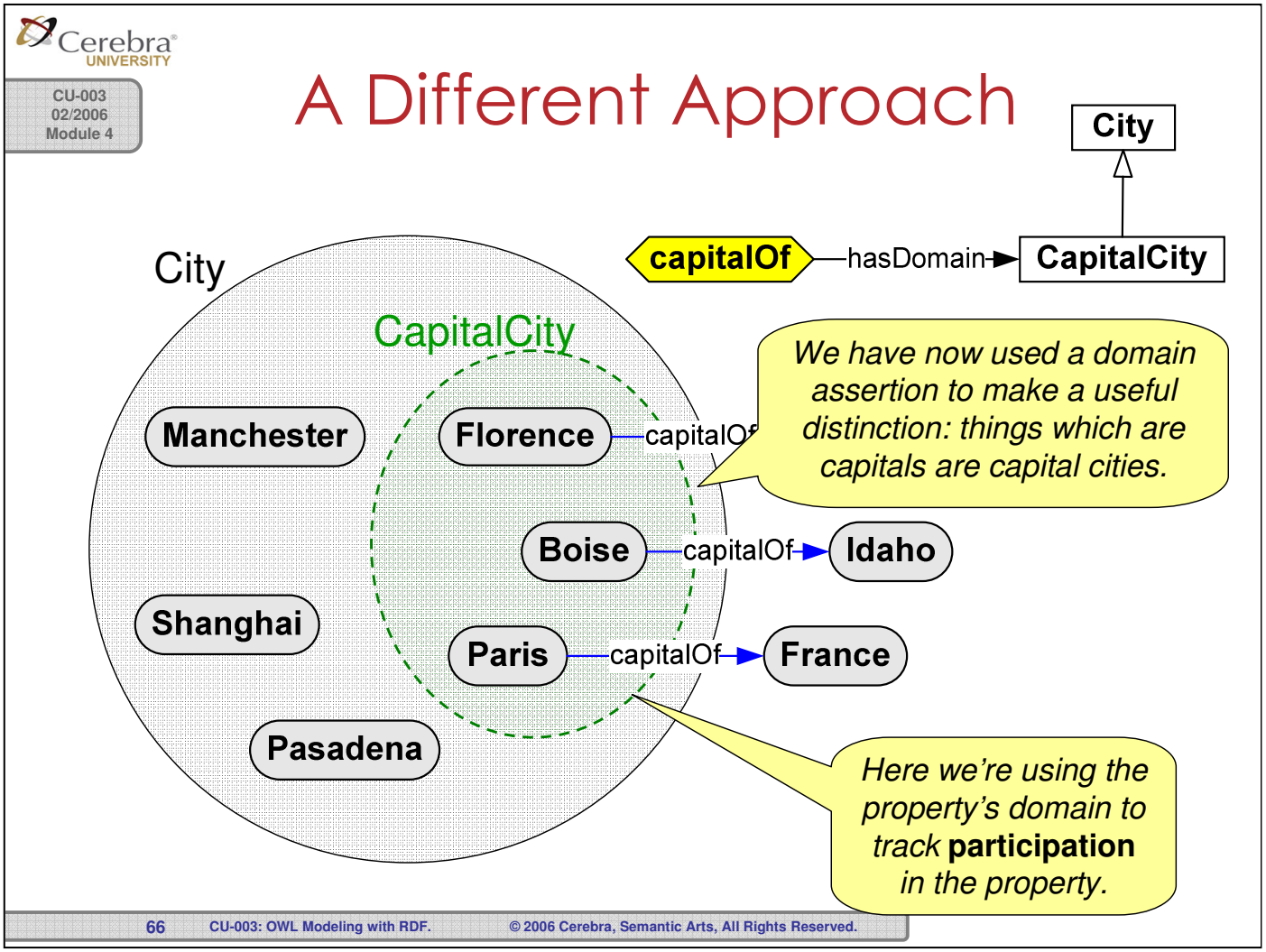
# Let's try something else…




```
<owl:Class rdf:ID="CapitalCity">
   <rdfs:subClassOf rdf:resource="#City"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="capitalOf">
   <rdfs:domain rdf:resource="#CapitalCity"/>
</owl:ObjectProperty>
```

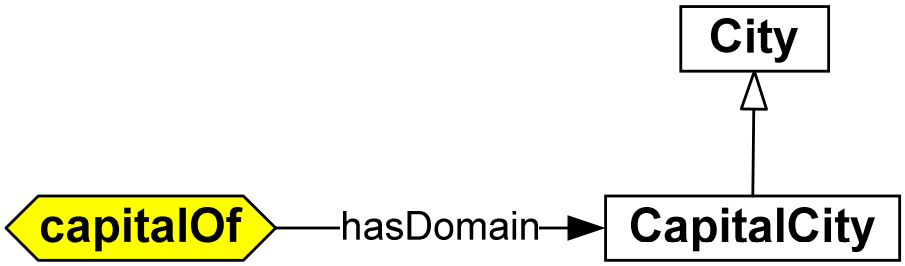
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:




Student Notes:

# Using Property Domains



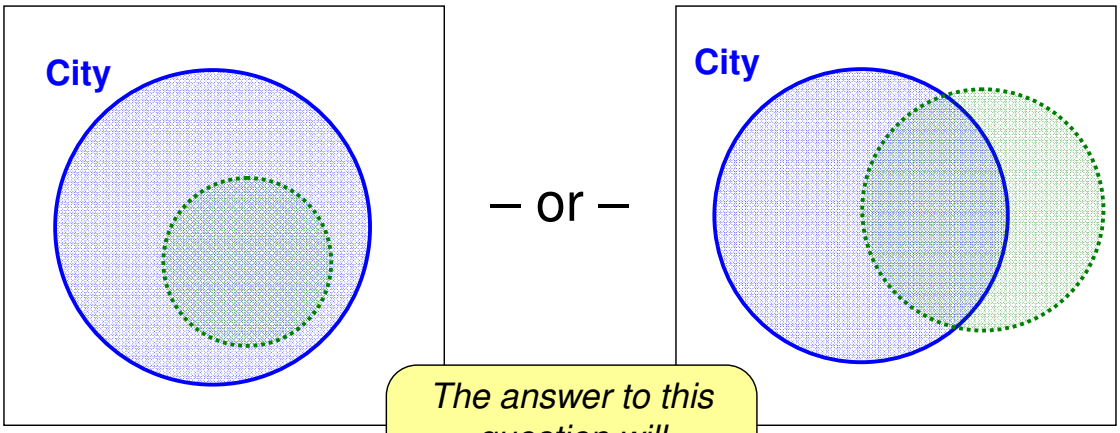

- In this example, **CapitalCity** is a subclass of **City.**
- This allows us to infer that that anything which has the **capitalOf** relationship is a **City** (as well as a **CapitalCity**).

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Using Property Domains

- This forces us to ask the question: "*are cities the only things that can be capitals?*"




*The answer to this question will determine how we model our ontology.*

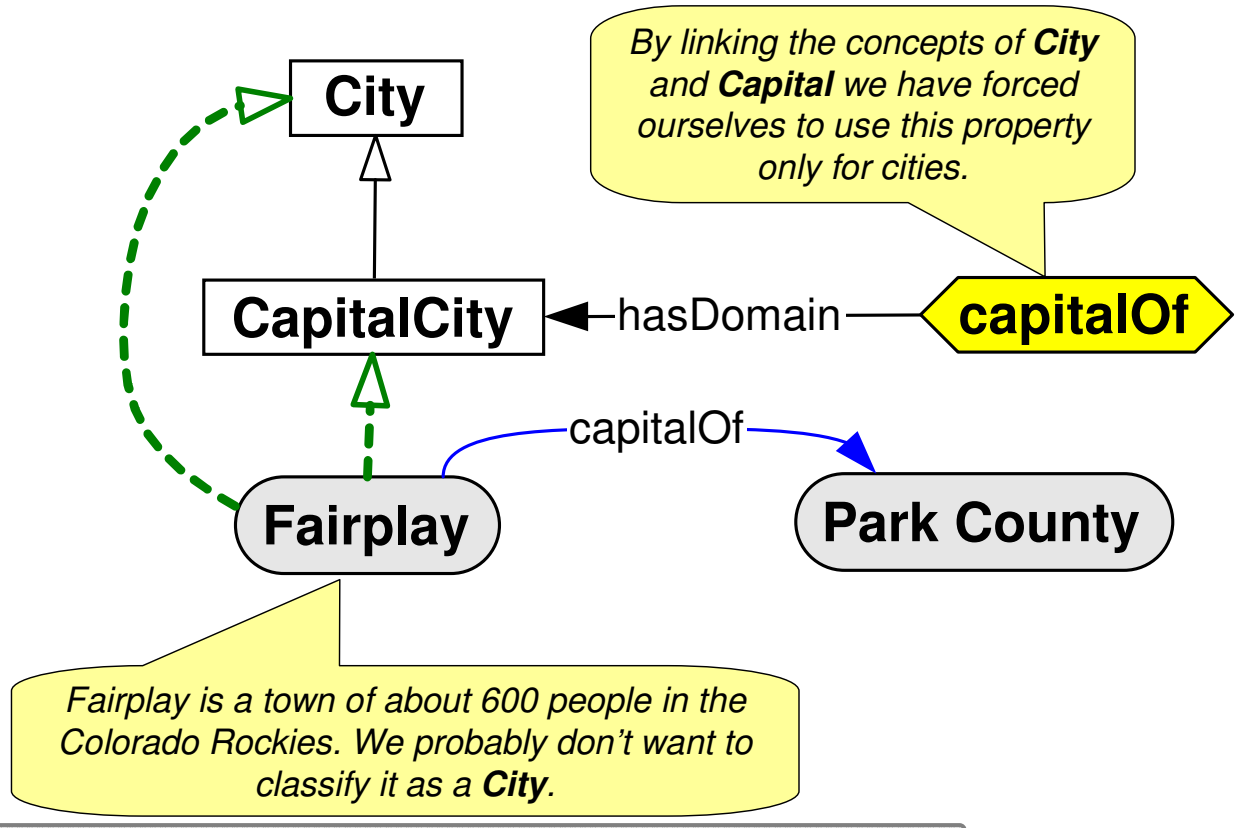
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# What's the Problem?




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Let's Try Another Approach...

*We have declared no explicit relationship between **Capital** and **City**.*

**City**

**capitalOf** —hasDomain→ **Capital**

```
<owl:Class rdf:ID="Capital">
<owl:ObjectProperty rdf:ID="capitalOf">
   <rdfs:domain rdf:resource="#Capital"/>
</owl:ObjectProperty>
```

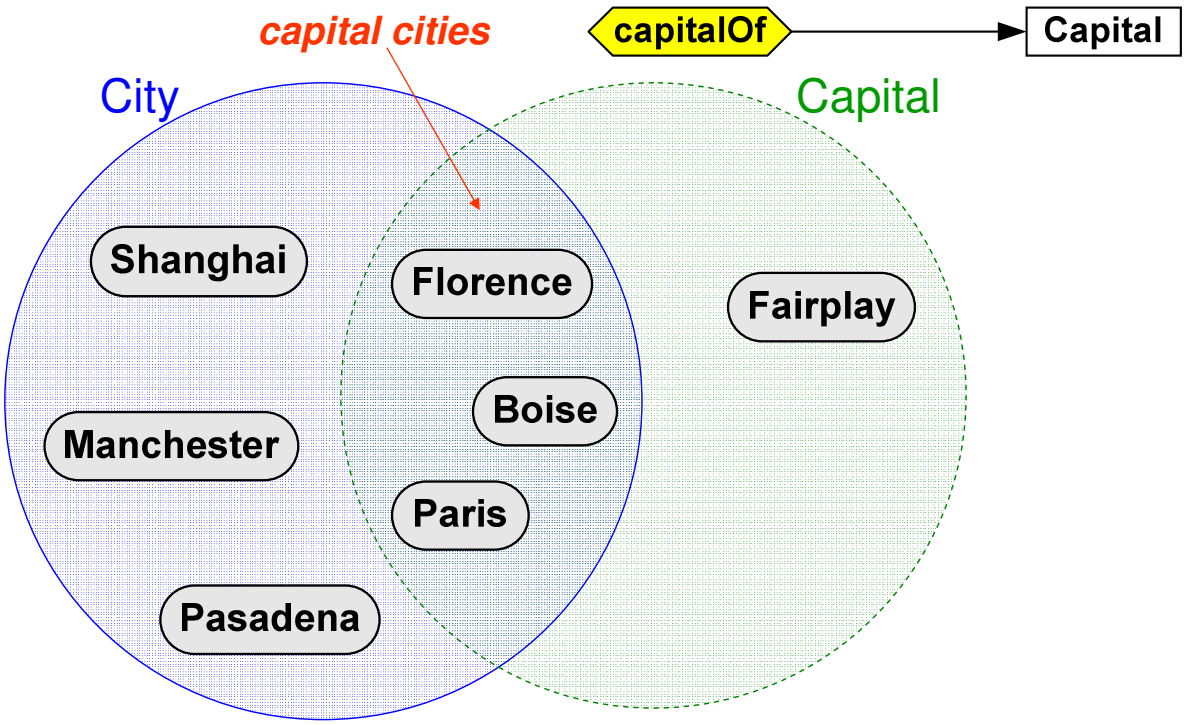
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Another approach...




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# What about range?

- So far, although we've referred to "domain and range," our examples have focused only on domain.
- Range is the mirror image of domain; it applies to the ***objects*** of properties.
- If an individual is the object of a property relationship, it is a member of that property's range.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

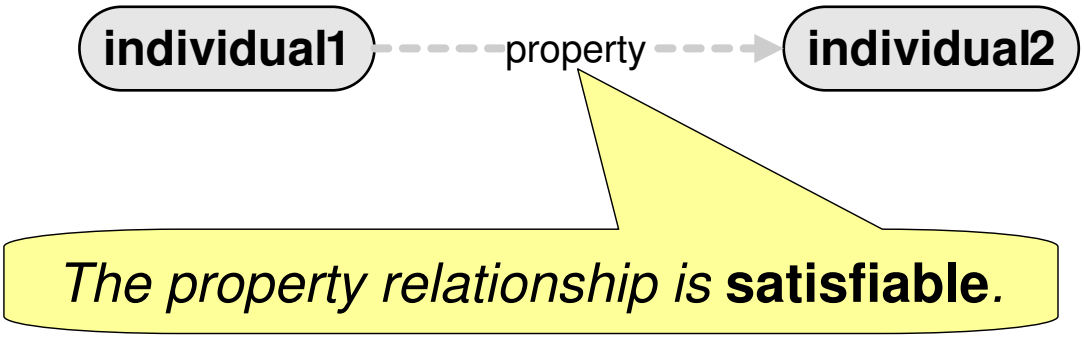
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Domain, Range, and the Open World

- Remember from the previous section that just because a relationship is not **declared** to exist between two individuals, this doesn't mean that it doesn't exist:




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

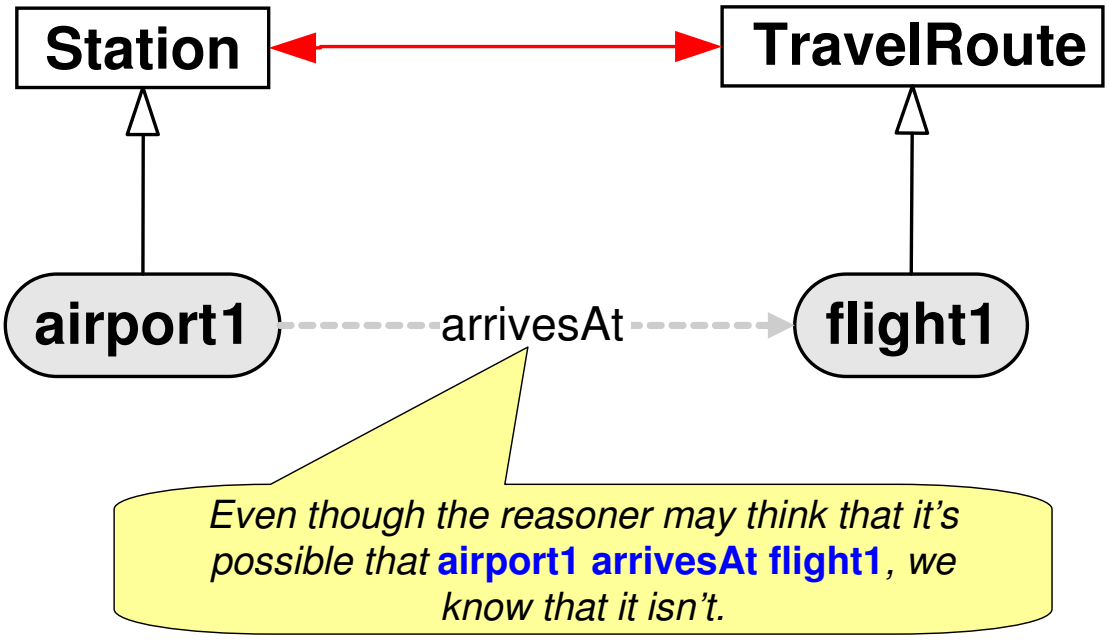
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Domain, Range, and the Open World

- This can lead to all sorts of nonsense:



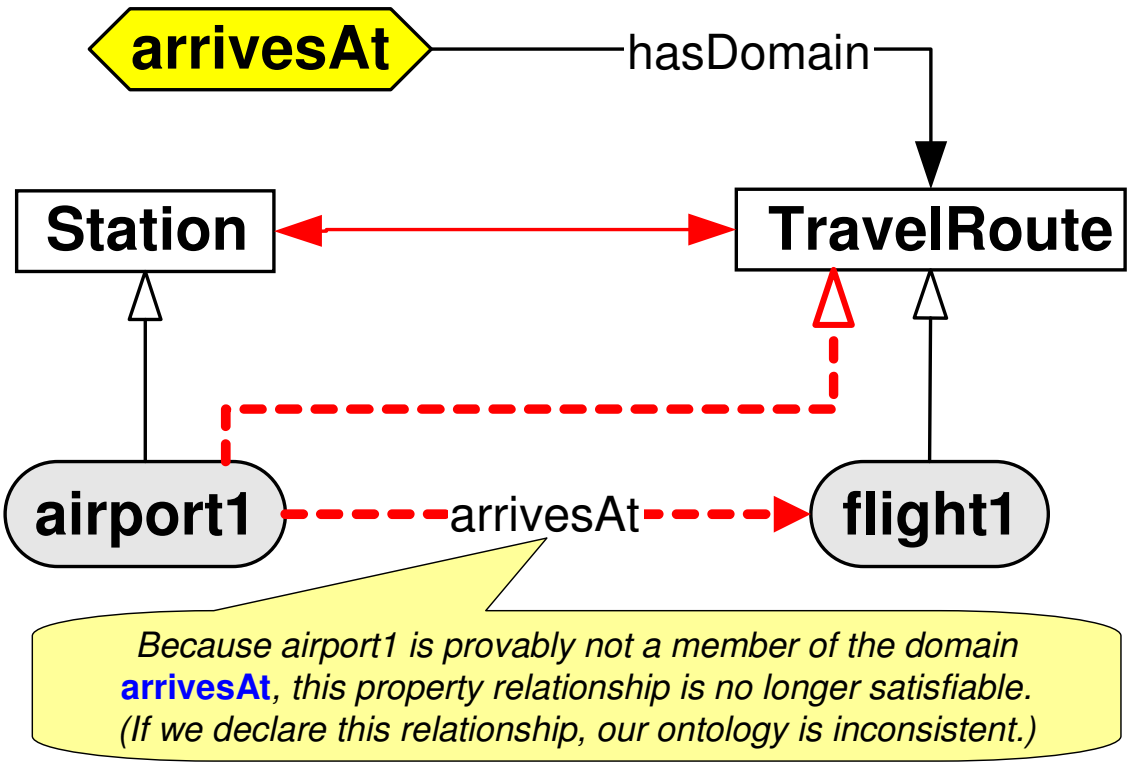

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Domain, Range, and the Open World




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Using Domain and Range as Constraints

- What we want to do is to separate our classes into three categories:
  1. Things that ***do*** participate in the property
  2. Things that ***can*** particpate in the property
  3. Things that ***cannot*** participate in the property

***This is the domain***

***This is disjoint with the domain***

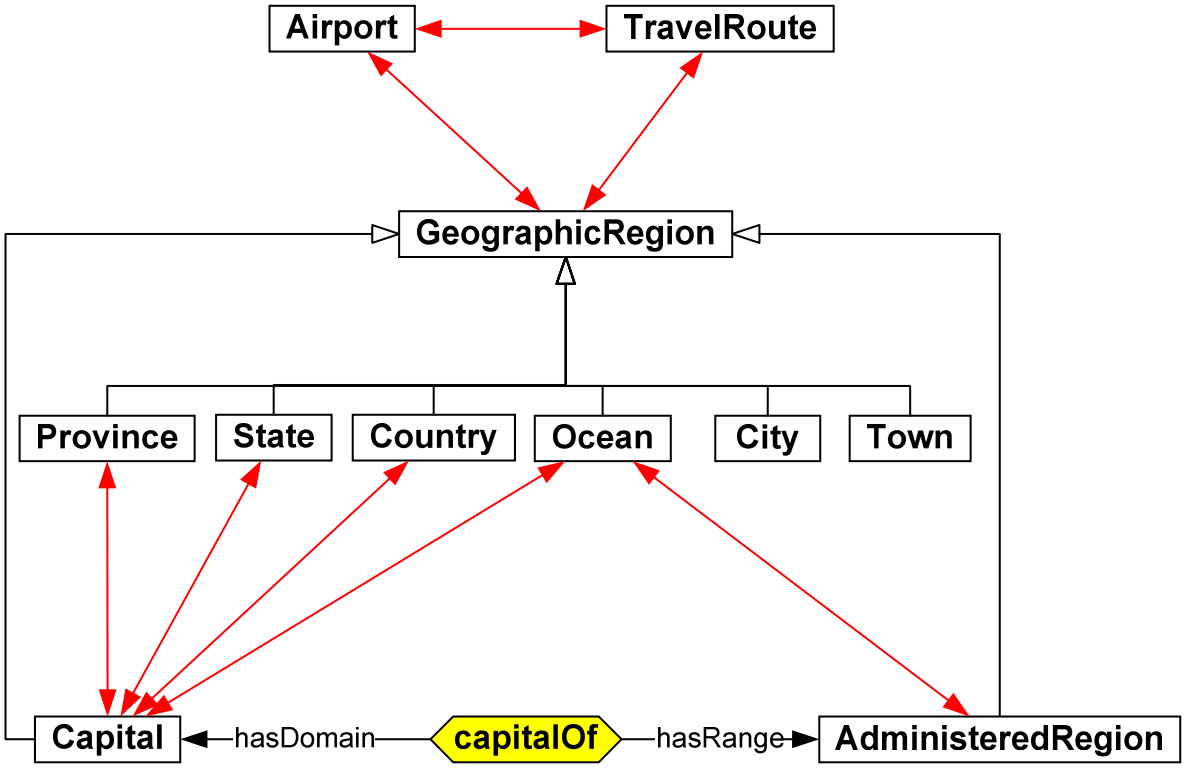
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Using Domain and Range




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Class Discussion

- How do the following affect domain and range?
  - Inverse properties
  - Transitive properties
  - Symmetric properties
- What would be a good domain and/or range for:
  - hasCapital
  - borders
  - locatedIn
  - departsFrom
  - arrivesAt

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Inverse - The domain of a property if the range of it's inverse. The range of a property is the domain of it's inverse.

Transitive - The domain and range must be the same. Think back to transitivity's association with recursion and this will make sence.

Symmetric - With symmetric properties the domain and range are the same set of individuals.

has Capital – Domain - Geo-Political area or Political Unit & Geographical Area

borders – Domain and Range - Geographical Area / Special plane

located in – Domain – Physical object - Range - Physical location

departs From – Domain – Route – Range - Geographic Location

arrives At – Domain - Route – Range – Geographic Location

This points out that an appropriate domain and range for a particular property is highly domain specific which is another reason why you should be cautious in using these constructs.

Domain is global meaning that anywhere the property is used in an ontology then it applies. If your ontology is going to be imported into someone else's then those domain and ranges would have to be generally applicable.

Cerebra UNIVERSITY

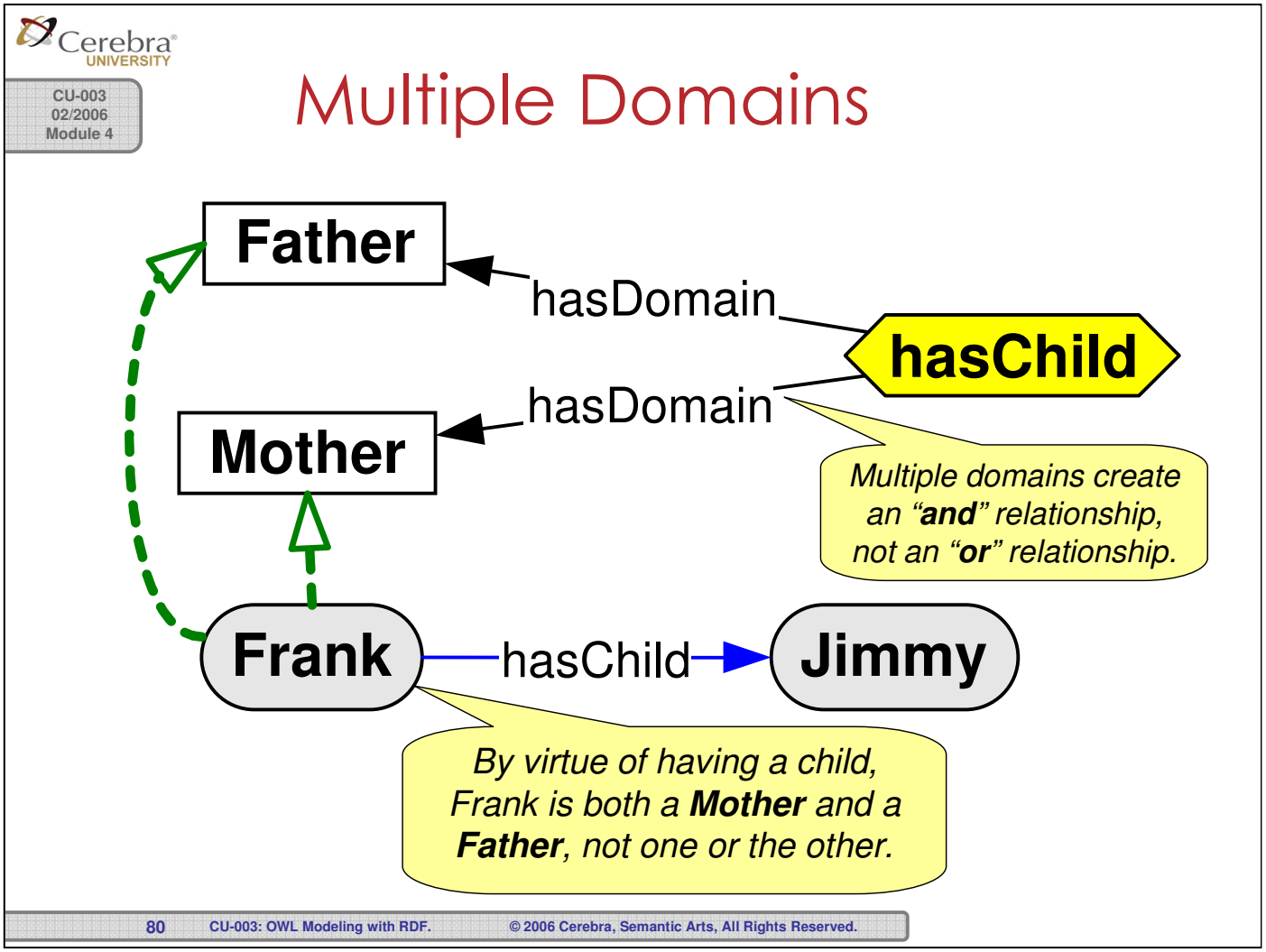
CU-003
02/2006
Module 4

# Multiple Domains, Multiple Ranges

- It is possible to declare more than one domain or range for a property.
- In this case, the domain or range is the ***intersection*** of these multiple classes.
- Having a property makes an individual a member of ***all*** that property's domains.
- In practice, we recommend that you don't do this.
- Why?

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:




The main problem here with domains and range is that it is hard to keep track of what has been inferred here. By adding a property to an instance you potentially have added membership to one or more domain classes and one or more range classes. Now assume that every property has one or more domain and range classes on it and you describe your individual with four or five properties. There is a lot of possibility for creating an inconsistent ontology here and it is often tough to track the inconsistency down.

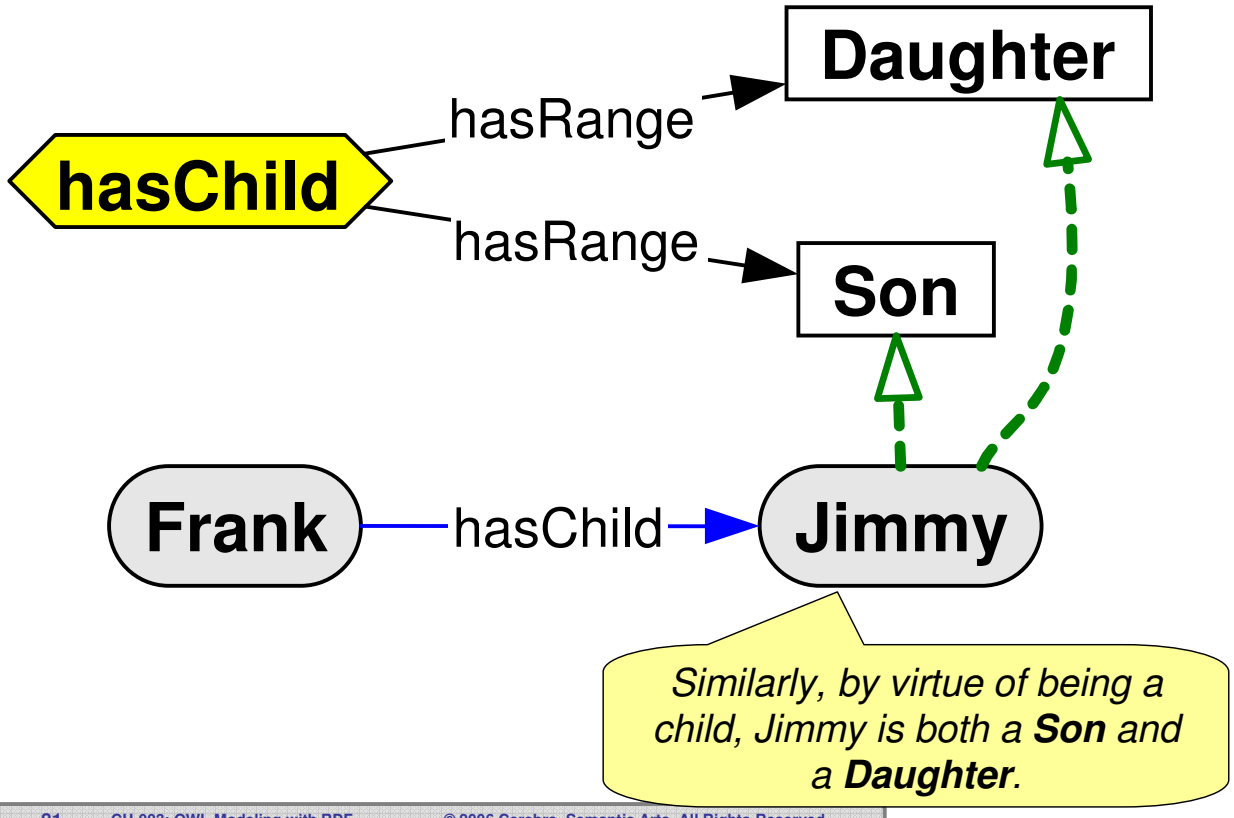
There is an alternative construct or inferring membership based on properties which we will get to when we talk about restriction classes.

## Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Multiple Ranges




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 4

# Module Review

- Instructor led review session
- Pop-quiz is optional!

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# OWL-DL Modeling with RDF

## Module Five: Complex Classes

CU-003: OWL Modeling with RDF. © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Agenda

| | Monday, Feb 27 | Tuesday, Feb 28 | Wednesday, Mar 01 |
|---|---|---|---|
| 8 am | | | |
| 9 00 | **Module 1:** *Intro to OWL* | **Module 3:** *Equivalence, Disjointness, and Subsumption* | **Module 5:** *Complex Classes* |
| 10 00 | | | |
| 11 00 | **Module 2:** *Individuals, Properties, and Classes* | **Module 3:** *Equivalence, Disjointness, and Subsumption* | **Module 5:** *Complex Classes* |
| 12 pm | | | |
| 1 00 | **Module 2:** *Individuals, Properties, and Classes* | **Module 4:** *More Properties* | **Module 6:** *How OWL is Different* |
| 2 00 | | | |
| 3 00 | **Module 2:** *Individuals, Properties, and Classes* | **Module 4:** *More Properties* | **Module 7:** *The Complement Module*<br>**Module 8:** *Wrap up* |
| 4 00 | | | |

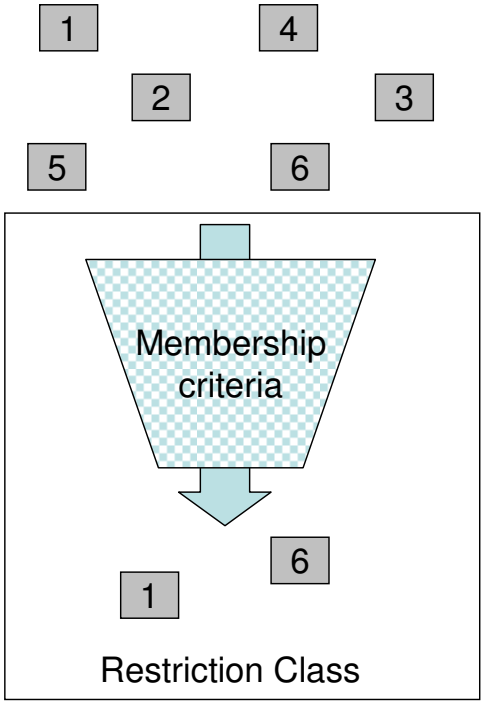
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Complex Classes




*Complex classes* act like categorization filters. They categorize individuals that are members of the class by filtering on membership criteria.

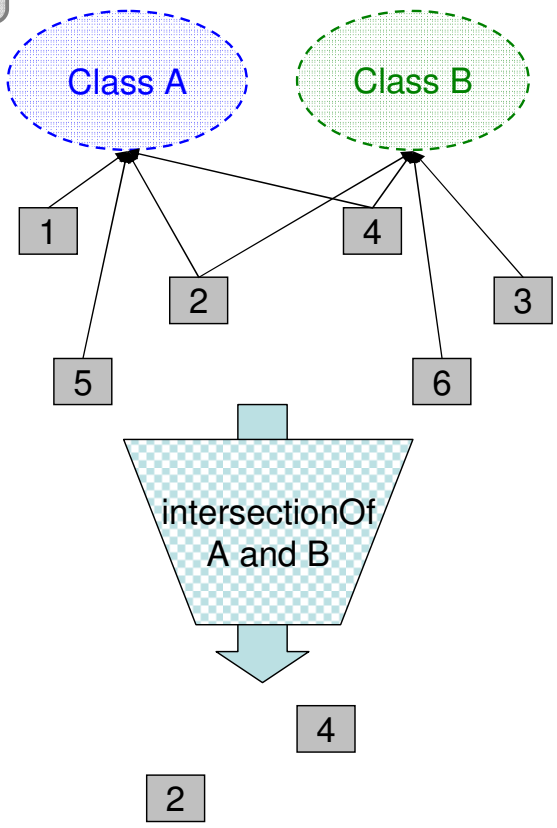
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Boolean Class Combinations




*Boolean class combinations* filter individuals based on the individual existing class membership using the unionOf, complementOf and intersectionOf constructs.

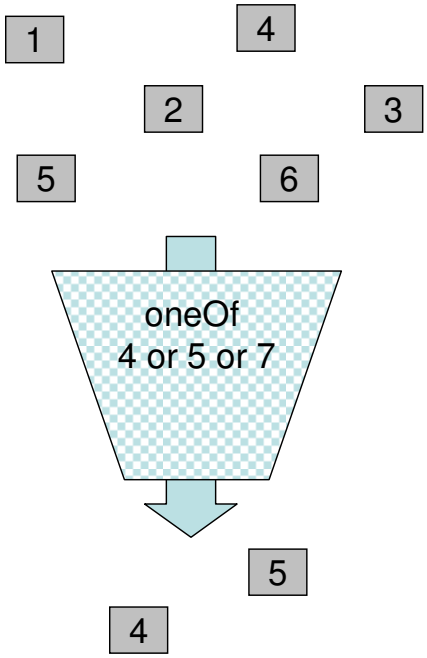
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

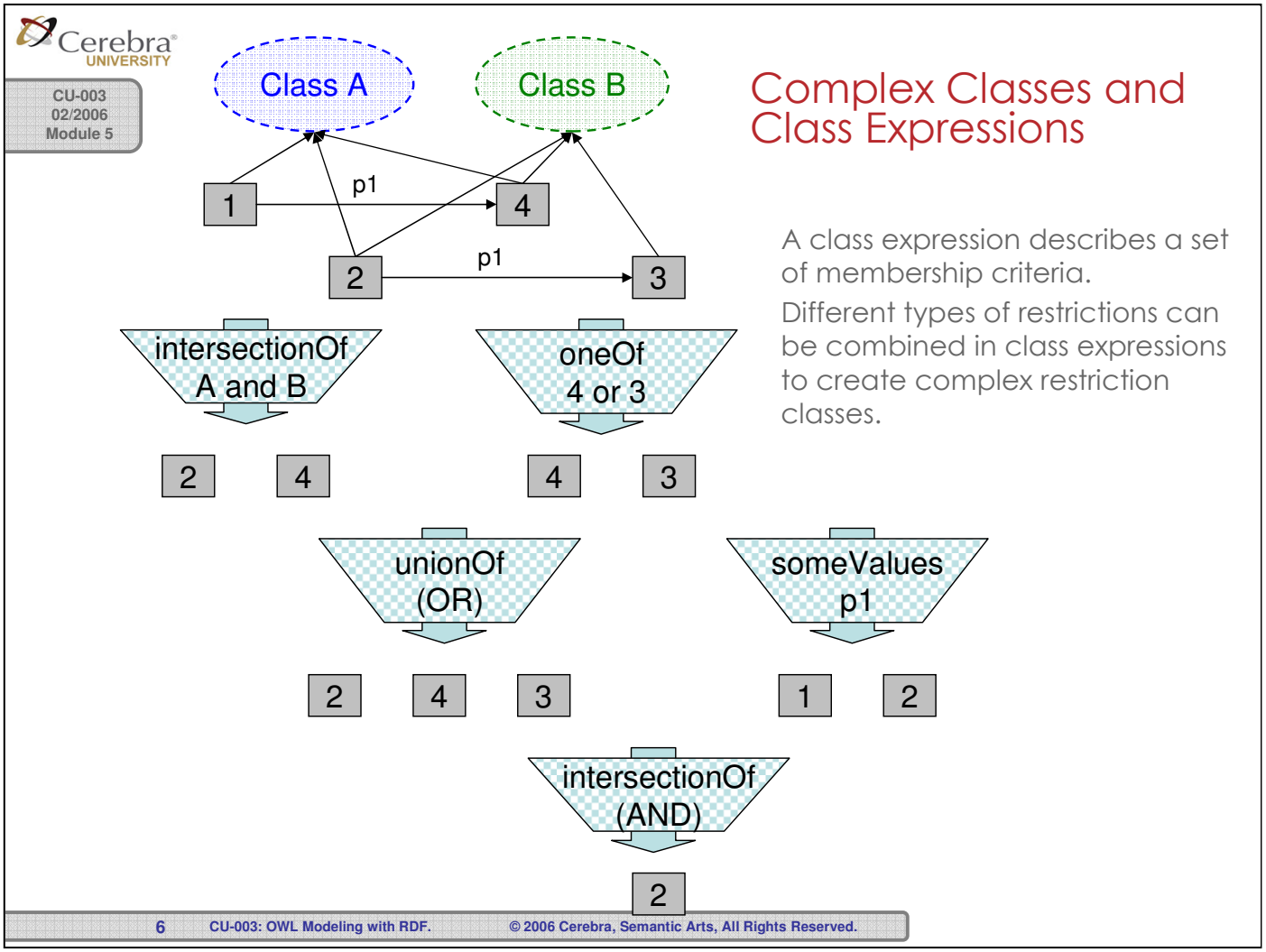
# Enumerations




*Enumerations* filter individuals based on a predefined set of members using the oneOf construct.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:




Student Notes:

Cerebra UNIVERSITY

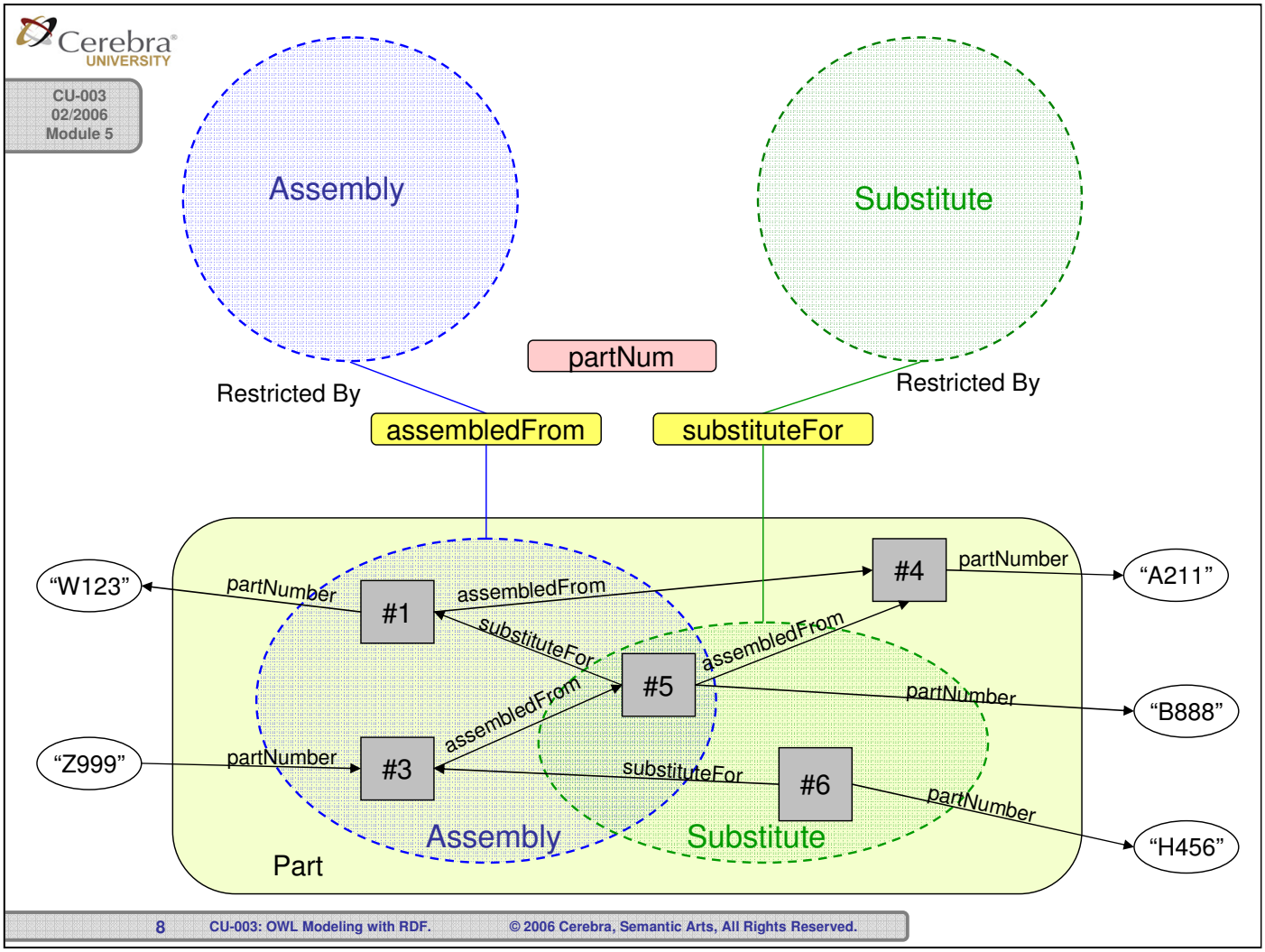
CU-003
02/2006
Module 5

# Dynamic Categorization

- The primary use of complex classes is to dynamically categorize individuals based on membership criteria.
- Instead of explicitly declaring an instance to be in a certain class, we define the membership criteria for the class and let the reasoner assign membership.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:




Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Exercise 1

- Open the ontology `Module5_Exercise1`.
- Follow along with instructor to extend the ontology to build the complex class described on the previous slide (with the numbers).
- Query the ontology to test the assertions as you go. (Name each complex class as you go, to allow querying it.)
- Export the ontology and examine the OWL.

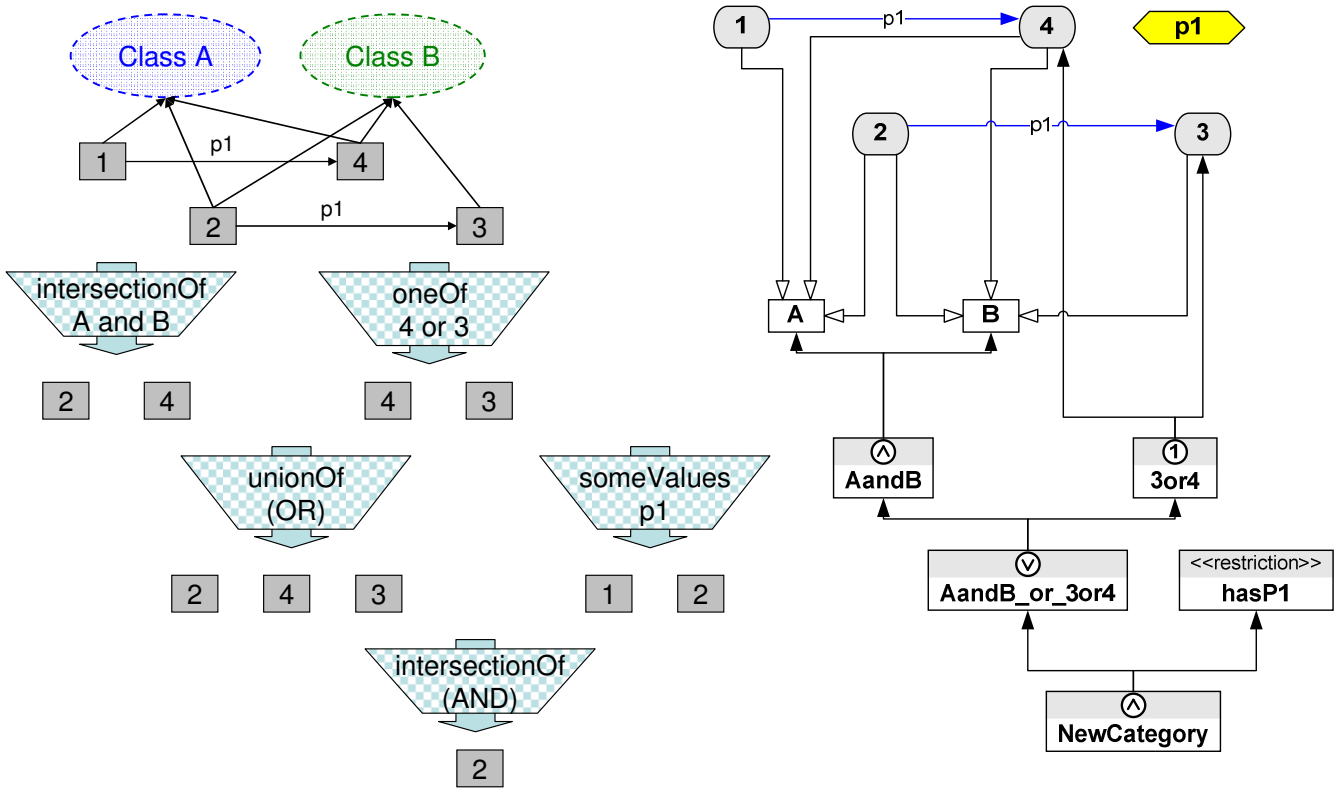
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Exercise 1




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

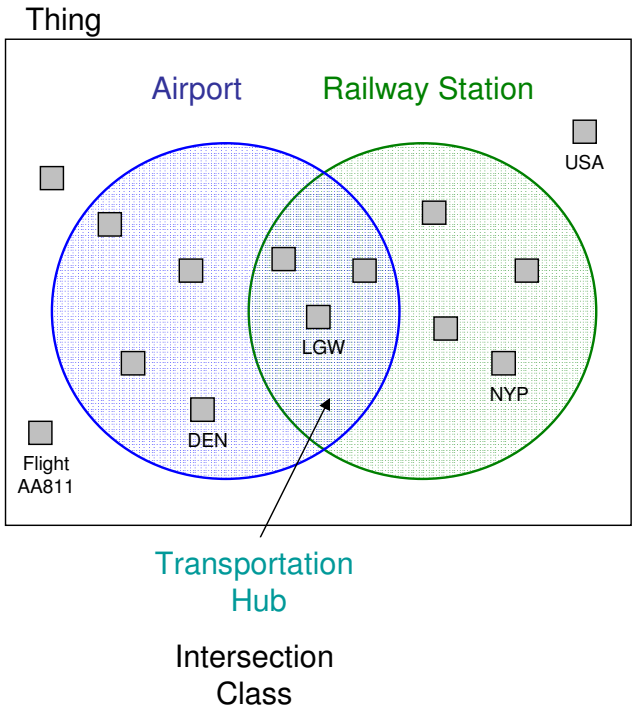
# Boolean Class Combinations

owl:intersectionOf
owl:unionOf
owl:complementOf

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Intersection




*The class described by the intersection property includes exactly all the individuals that are common to all the identified classes.*

TransportationHub is the set of individuals that are members of both Airport AND RailwayStation.

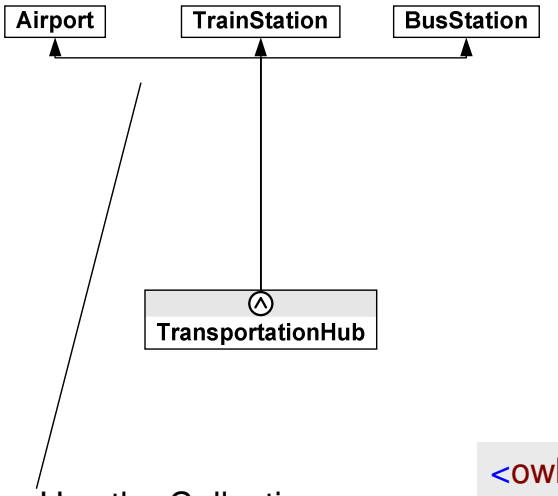
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Intersection




Use the Collection arrow to create boolean classes.

```
<owl:Class rdf:ID="Airport"/>
<owl:Class rdf:ID="TrainStation"/>
<owl:Class rdf:ID="BusStation"/>

<owl:Class rdf:ID="TransportationHub">
    <owl:intersectionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#Airport"/>
            <owl:Class rdf:about="#TrainStation"/>
            <owl:Class rdf:about="#BusStation"/>
     </owl:intersectionOf>
  </owl:Class>
```

```
<owl:Class rdf:ID=" rdf:ID"class reference ">
    <owl:intersectionOf rdf:parseType="Collection">
          List of class expressions
     </owl:intersectionOf>
 </owl:Class>
```

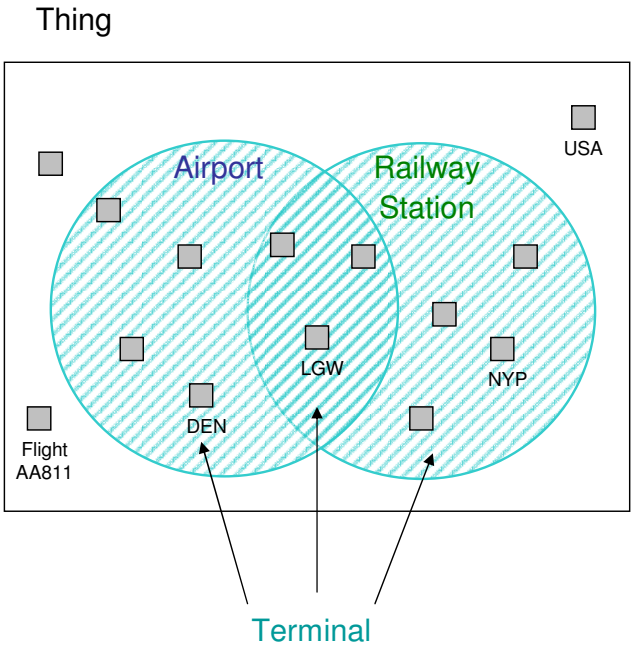
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Union




*Describes a class that includes all the members of the specified classes.*

Terminal is the set of individuals that are members of either Airport OR RailwayStation (or both).

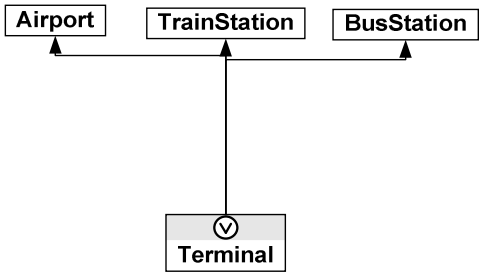
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Union




```
<owl:Class rdf:ID="Airport"/>
<owl:Class rdf:ID="TrainStation"/>
<owl:Class rdf:ID="BusStation"/>

<owl:Class rdf:ID="TransportationHub">
    <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Airport"/>
        <owl:Class rdf:about="#TrainStation"/>
        <owl:Class rdf:about="#BusStation"/>
    </owl:unionOf>
</owl:Class>
```

```
<owl:Class rdf:ID=" rdf:ID"class reference ">
    <owl:unionOf rdf:parseType="Collection">
        List of class expressions
    </owl:unionOf>
</owl:Class>
```

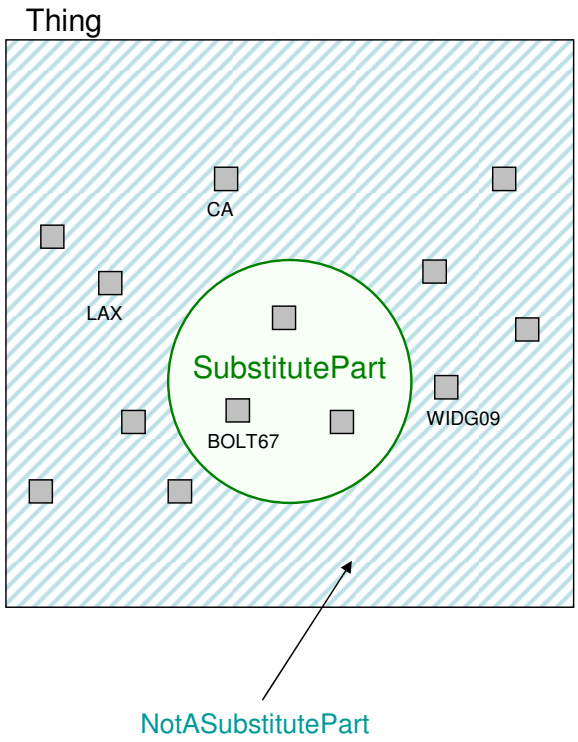
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Complement




*Describes a class that includes all the members that do **not** belong to a specified class.*

NotASubstitutePart is the set of individuals that are not SubstituteParts.

In practice, complement is used along with intersection to get useful results.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

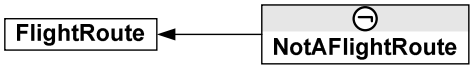
In other words, every instance in the ontology that is not a member of the class.
You can have more than one class being complemented.

## Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Complement




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

<owl:Class rdf:ID="NotAFlightRoute">
    <owl:complementOf>
      <owl:Class rdf:about="#FlightRoute"/>
    </owl:complementOf>
</owl:Class>
```

```
<owl:Class rdf:ID=" rdf:ID"class reference ">
    <owl:complementOf rdf:parseType="Collection">
          List of class expressions
    </owl:complementOf>
</owl:Class>
```

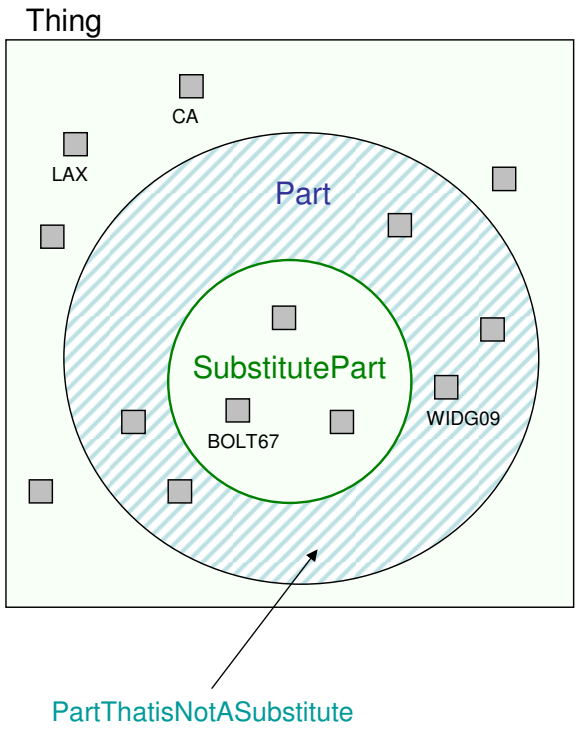
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Complement




In practice, complement is used along with intersection to get useful results.

PartThatIsNotASubstitute is the set of individuals that are Parts but not SubstituteParts.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

I.e every instance in the ontology that is not a member of the class.

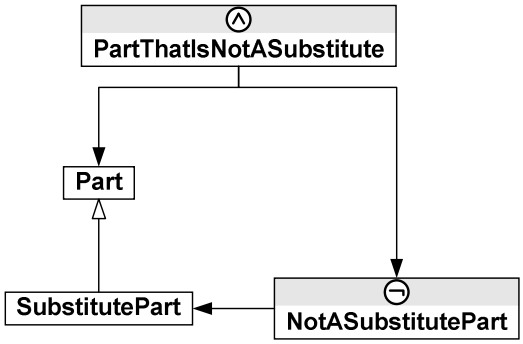
You can have more than one class being complemented.

## Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Complement




```
<owl:Class rdf:ID="Part"/>
<owl:Class rdf:ID="SubstitutePart">
      <rdfs:subClassOf rdf:resource="#Part"/>
</owl:Class>

<owl:Class rdf:ID="NotASubstitutePart">
      <owl:complementOf>
         <owl:Class rdf:about="#SubstitutePart"/>
      </owl:complementOf>
</owl:Class>

<owl:Class rdf:ID="PartThatIsNotASubstitute">
   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Part"/>
      <owl:Class rdf:about="#NotASubstitutePart"/>
   </owl:intersectionOf>
</owl:Class>
```

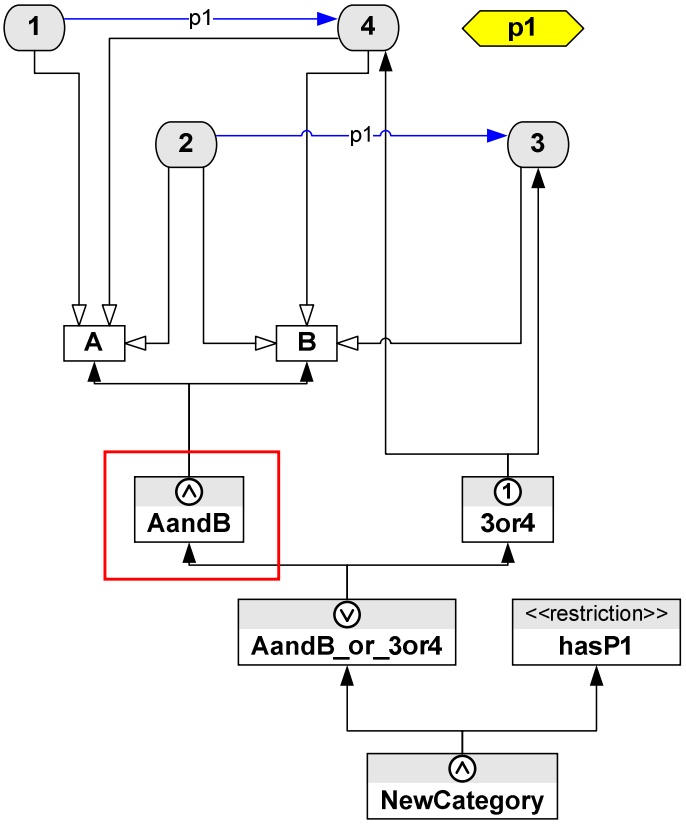
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Exercise




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

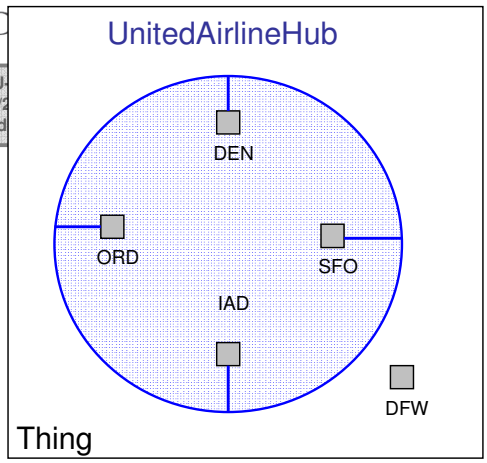
Student Notes:

# Enumerated Classes

owl:oneOf

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

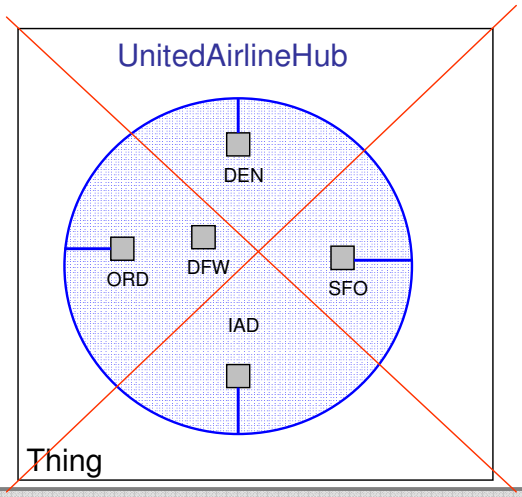
Student Notes:




# Enumeration

*An enumerated class is a class with a predefined set of member inststances.*

UnitedAirlineHub is the set of individuals DEN, SFO, IAD, ORD and no others.




The important aspect of an enumeration is that it eliminates all possible models where other instances are members of the class. That is, it closes the class. All other (disjoint) individuals can now be inferred not to be members of the class.

If we had just defined a class with four member instances, this would not have been the case.

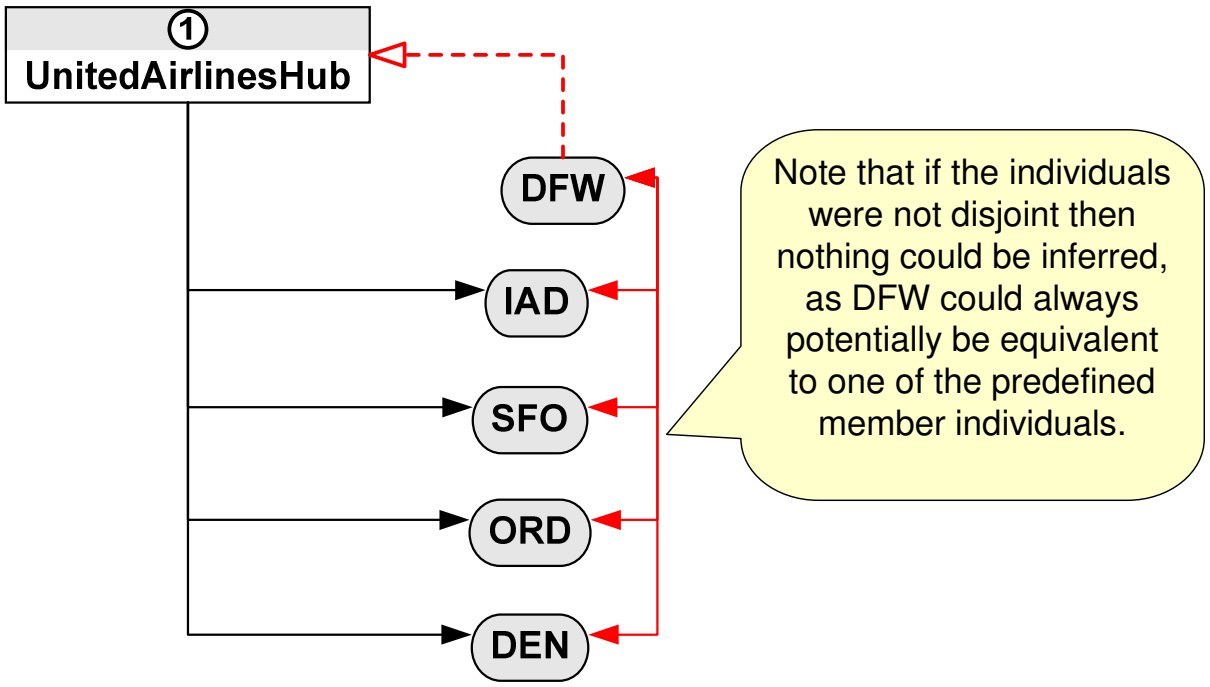
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Enumeration



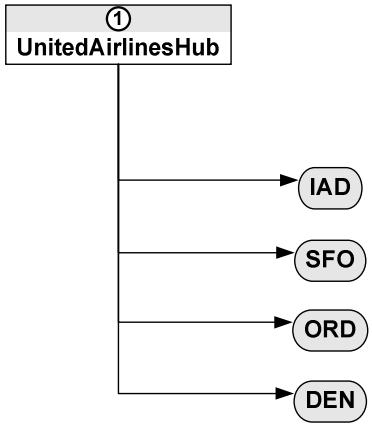

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Enumeration




```
<owl:Class rdf:ID="UnitedAirlinesHub">
        <owl:oneOf rdf:parseType="Collection">
                <owl:Thing rdf:about="#IAD"/>
                <owl:Thing rdf:about="#SFO"/>
                <owl:Thing rdf:about="#ORD"/>
                <owl:Thing rdf:about="#DEN"/>
        </owl:oneOf>
</owl:Class>
```

```
<owl:Class rdf:ID=" rdf:ID"class reference ">
    <owl:oneOf rdf:parseType="Collection">
        List of individuals
    </owl:oneOf>
</owl:Class>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Enumeration

- There are often better modeling alternatives to the use of Enumeration classes, for example, adding a hubFor object property between the Airport and Airline classes.
- If you do use them, then:
  - There should be a small number of instances in the class.
  - The members are known a priori when the ontology is established.
  - Membership is highly stable.
  - The complement of the class should be of interest, e.g., *Airport*s that are ***not*** *UnitedAirlineHub*s.

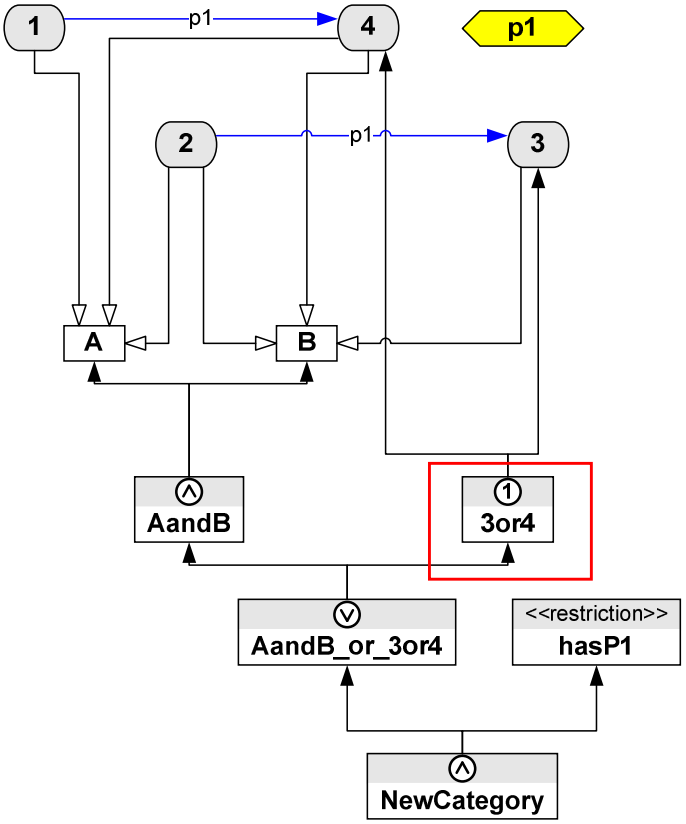
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Exercise




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Local Property Restriction Classes

owl:someValuesFrom
owl:allValuesFrom
owl:hasValue
owl:minCardinality
owl:maxCardinality
owl:cardinality

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Local Property Restrictions

- The property constraints we have looked at so far are global constraints:
  - owl:functional
  - owl:symmetric
  - owl:transitive

  They apply wherever that property is used throughout the ontology.

- Local property restrictions apply to the use of the property with a particular class. They include the restrictions:
  - owl:someValuesFrom
  - owl:allValuesFrom
  - owl:hasValue
  - owl:minCardinality
  - owl:maxCardinality
  - owl:cardinality

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# someValuesFrom




**Airport** is the name of a restriction class defining the set of all individuals that have at least one hasDeparting property referencing a FlightRoute.

*Alternatively:*

An Airport is anything that a FlightRoute departs from.

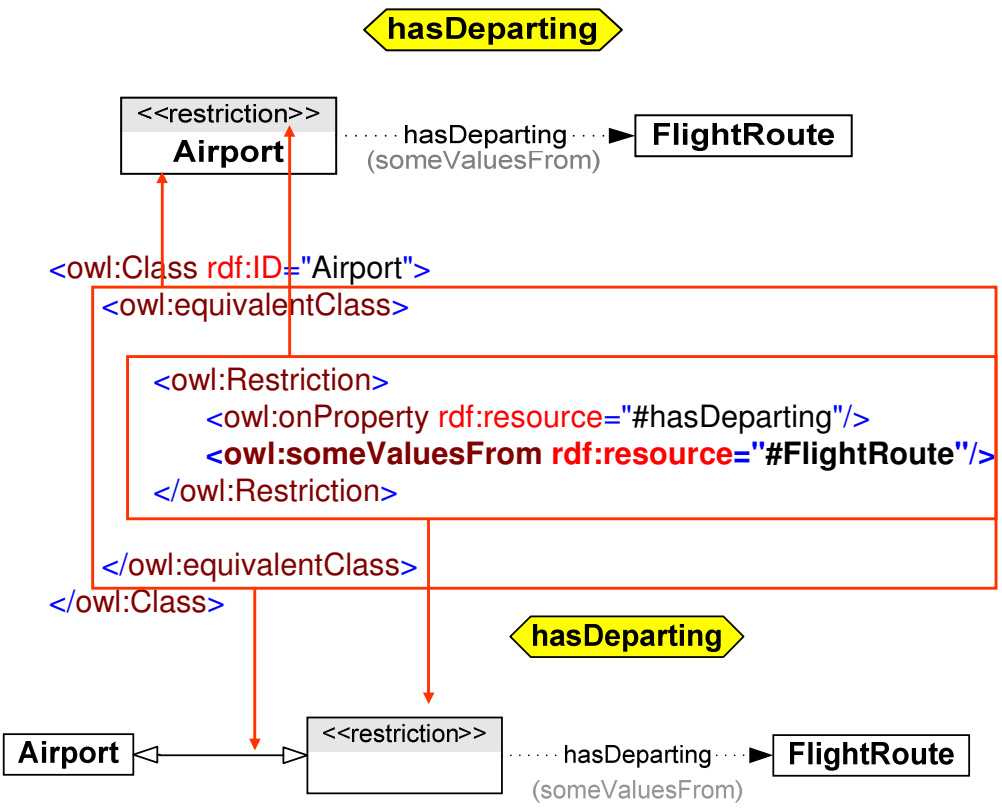
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# someValuesFrom

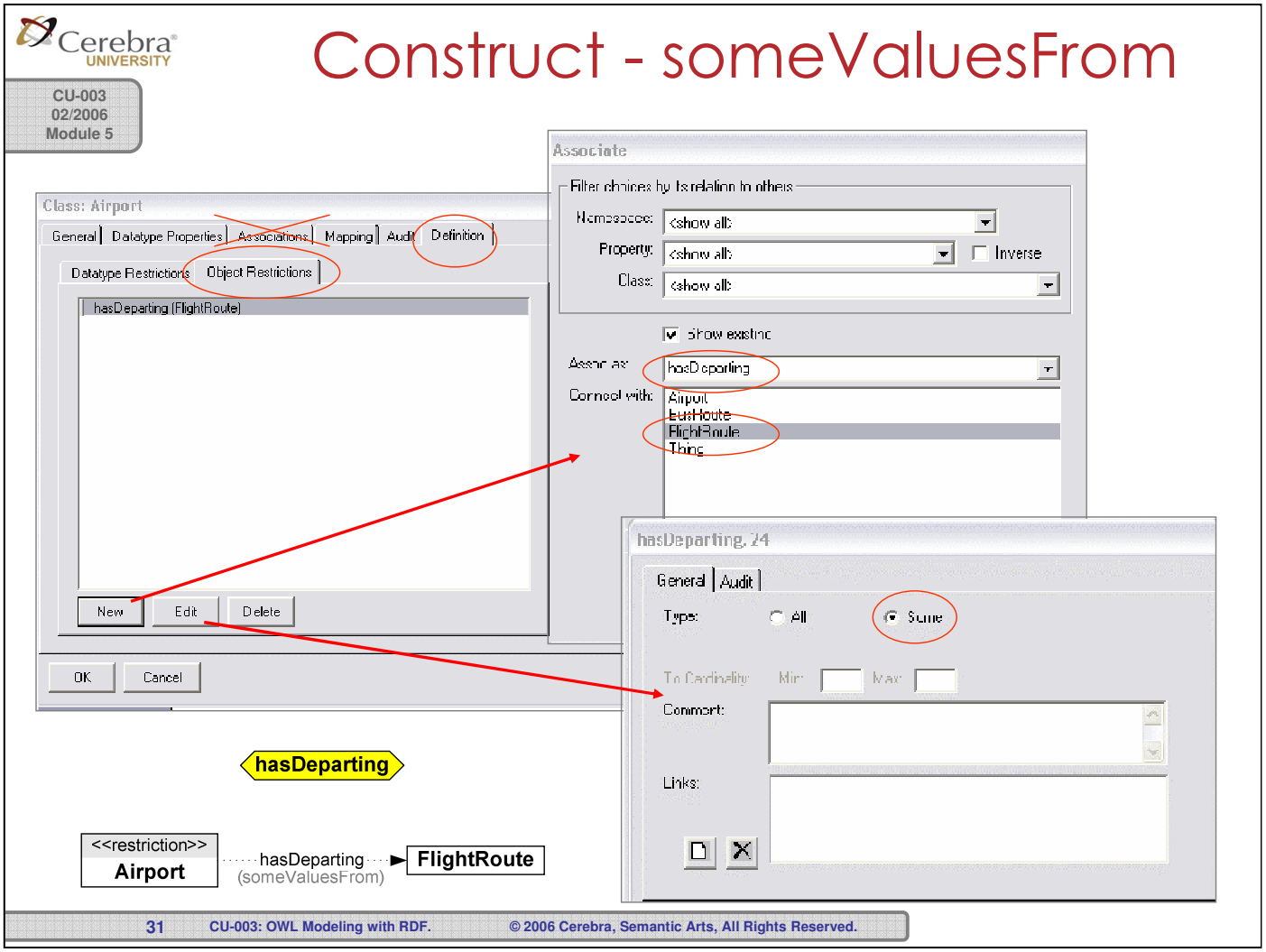



 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Sometimes called equivalence semantics

## Student Notes:

# Construct - someValuesFrom

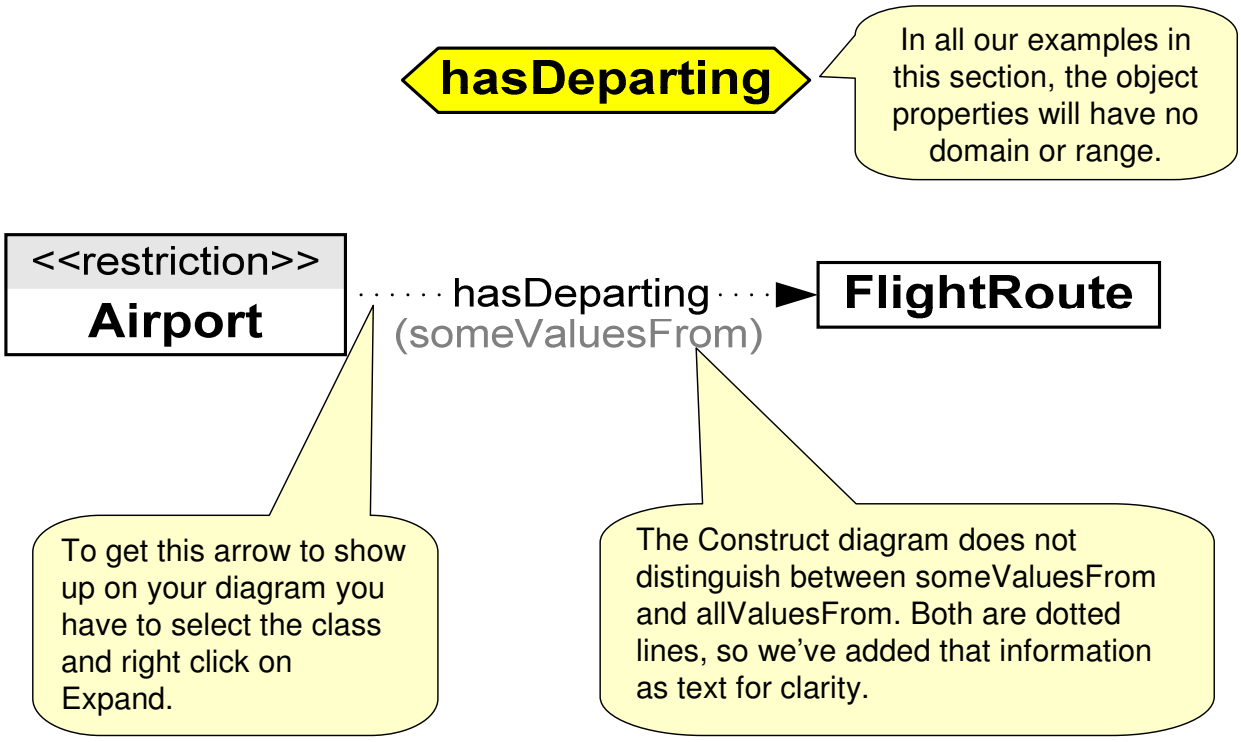



## Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

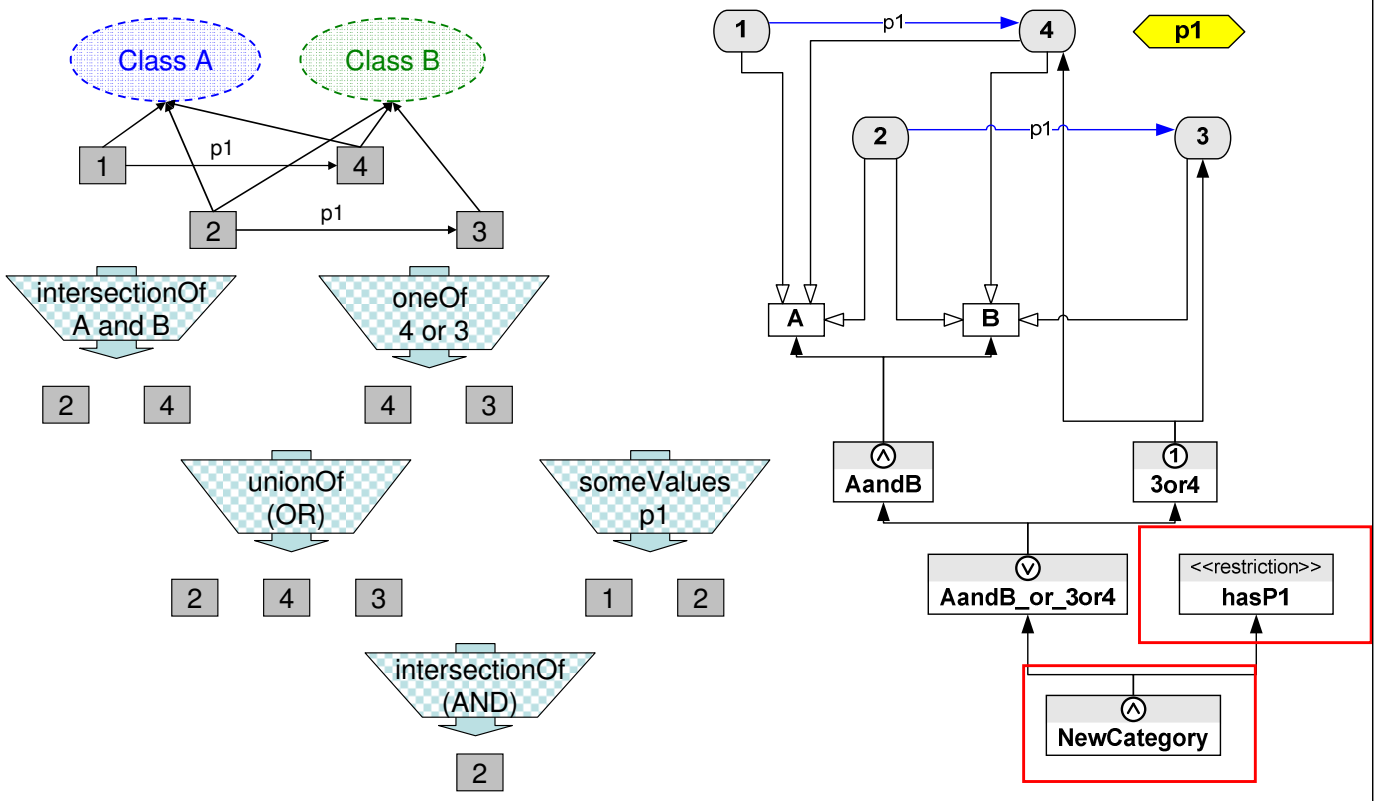
# Construct - someValuesFrom




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Exercise



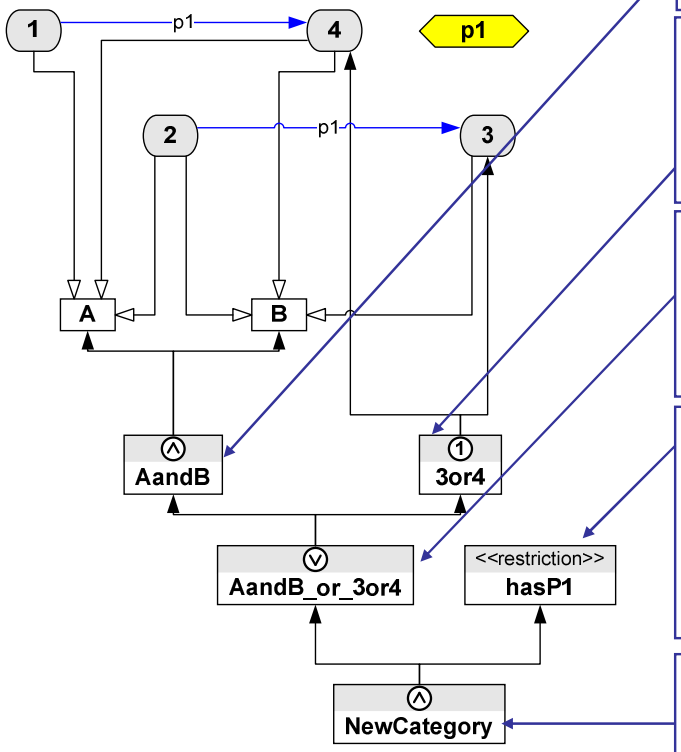

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Named Class Example




```
<owl:Class rdf:ID="AandB">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#B"/>
    <owl:Class rdf:about="#A"/>
  </owl:intersectionOf>
</owl:Class>
```

```
<owl:Class rdf:ID="3or4">
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#3"/>
    <owl:Thing rdf:about="#4"/>
  </owl:oneOf>
</owl:Class>
```

```
<owl:Class rdf:ID="AandB_or_3or4">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#AandB"/>
    <owl:Class rdf:about="#3or4"/>
  </owl:unionOf>
</owl:Class>
```

```
<owl:Class rdf:ID="hasP1">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#p1"/>
      <owl:someValuesFrom rdf:resource="http://..owl#Thing"/>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

```
<owl:Class rdf:ID="NewCategory">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#AandB_or_3or4"/>
    <owl:Class rdf:about="#hasP1"/>
  </owl:intersectionOf>
</owl:Class>
```

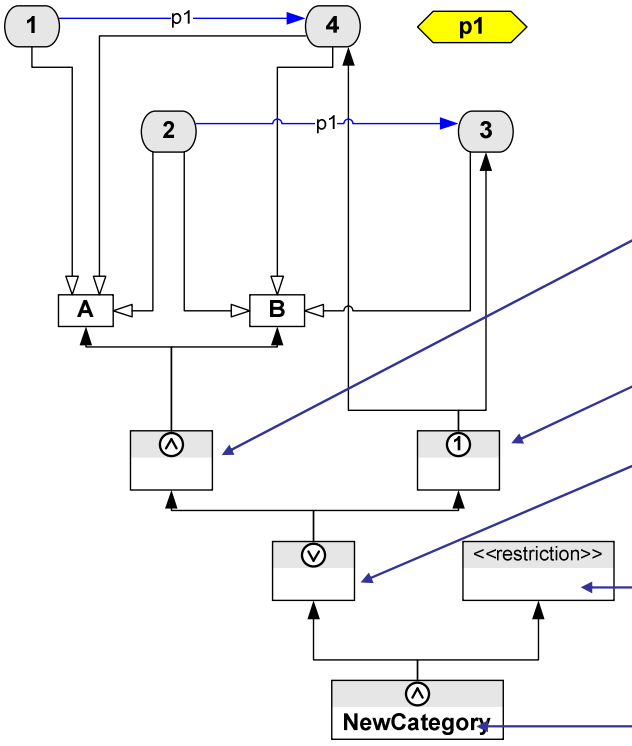
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

## Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Anonymous Class Example




```
<owl:Class rdf:ID="NewCategory">
 <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class>
   <owl:unionOf rdf:parseType="Collection">
    <owl:Class>
     <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#B"/>
      <owl:Class rdf:about="#A"/>
     </owl:intersectionOf>
    </owl:Class>
    <owl:Class>
     <owl:oneOf rdf:parseType="Collection">
      <owl:Thing rdf:about="#3"/>
      <owl:Thing rdf:about="#4"/>
     </owl:oneOf>
    </owl:Class>
   </owl:unionOf>
  </owl:Class>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#p1"/>
   <owl:someValuesFrom rdf:resource="http://…owl#Thing"/>
  </owl:Restriction>
 </owl:intersectionOf>
</owl:Class>
```

35 CU-003: OWL Modeling with RDF. © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Should you name your intermediate classes.

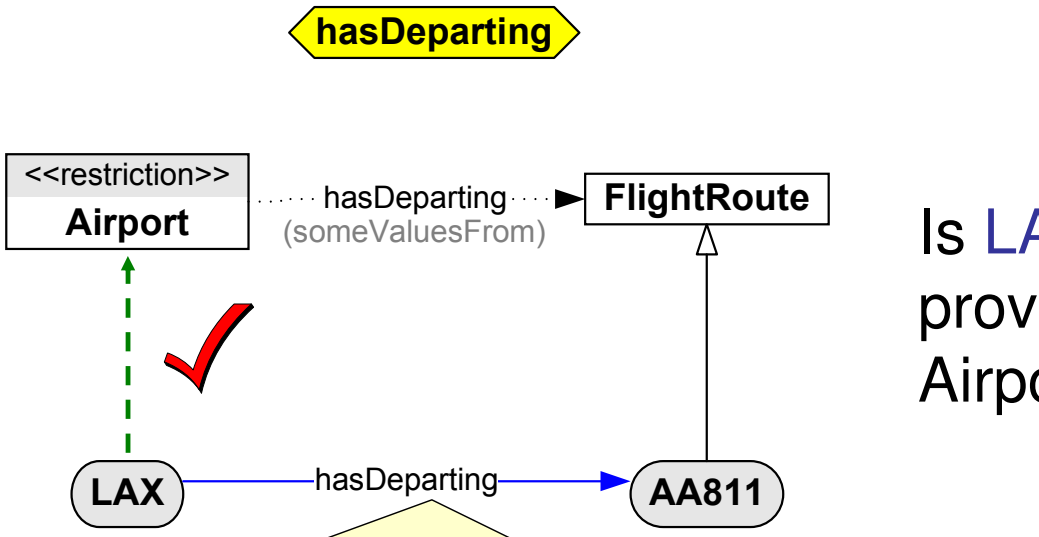
Yes for testing purposes.

Yes if you want to use that class in some other complex expression.

Possibly take them out if the classes are not individually meaningful to users and they will be exposed to them

## Student Notes:

# someValuesFrom




Is LAX provably an Airport ?

- AA811 will depart from LAX in every possible model.
- So there will always be at least one departing FlightRoute from LAX in every model.
- So LAX will be an Airport in every model.
- i.e., LAX is provably an Airport.

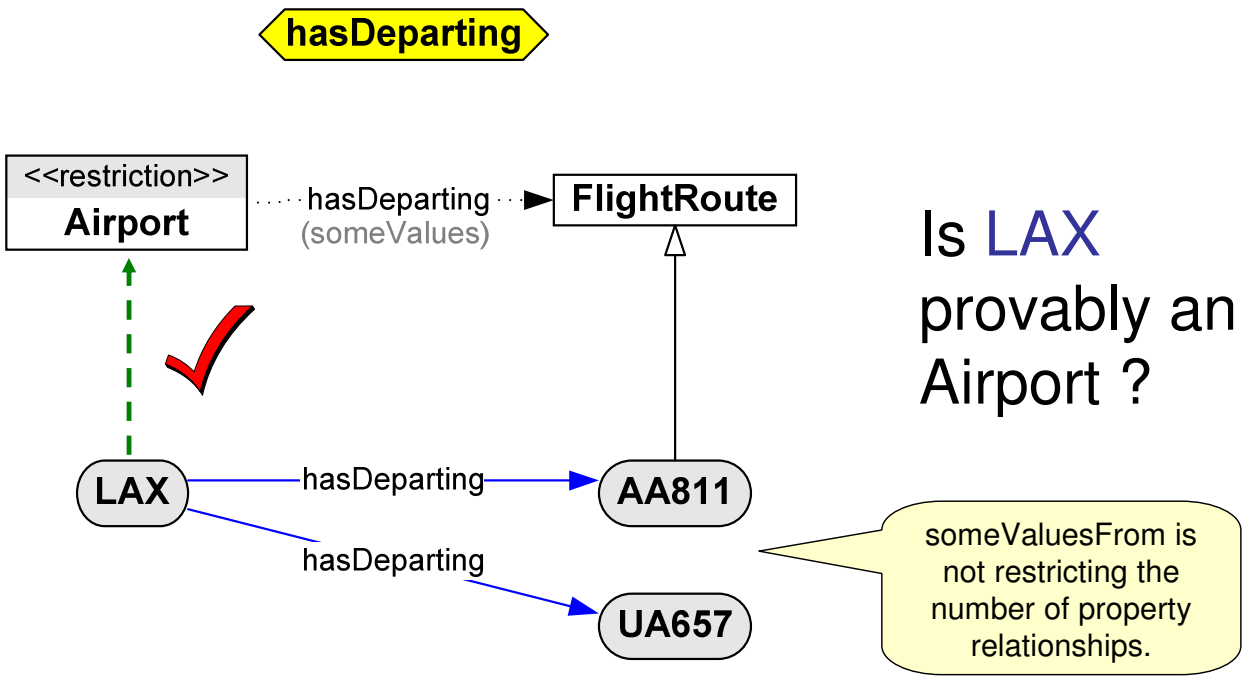
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# someValuesFrom




Is LAX provably an Airport ?

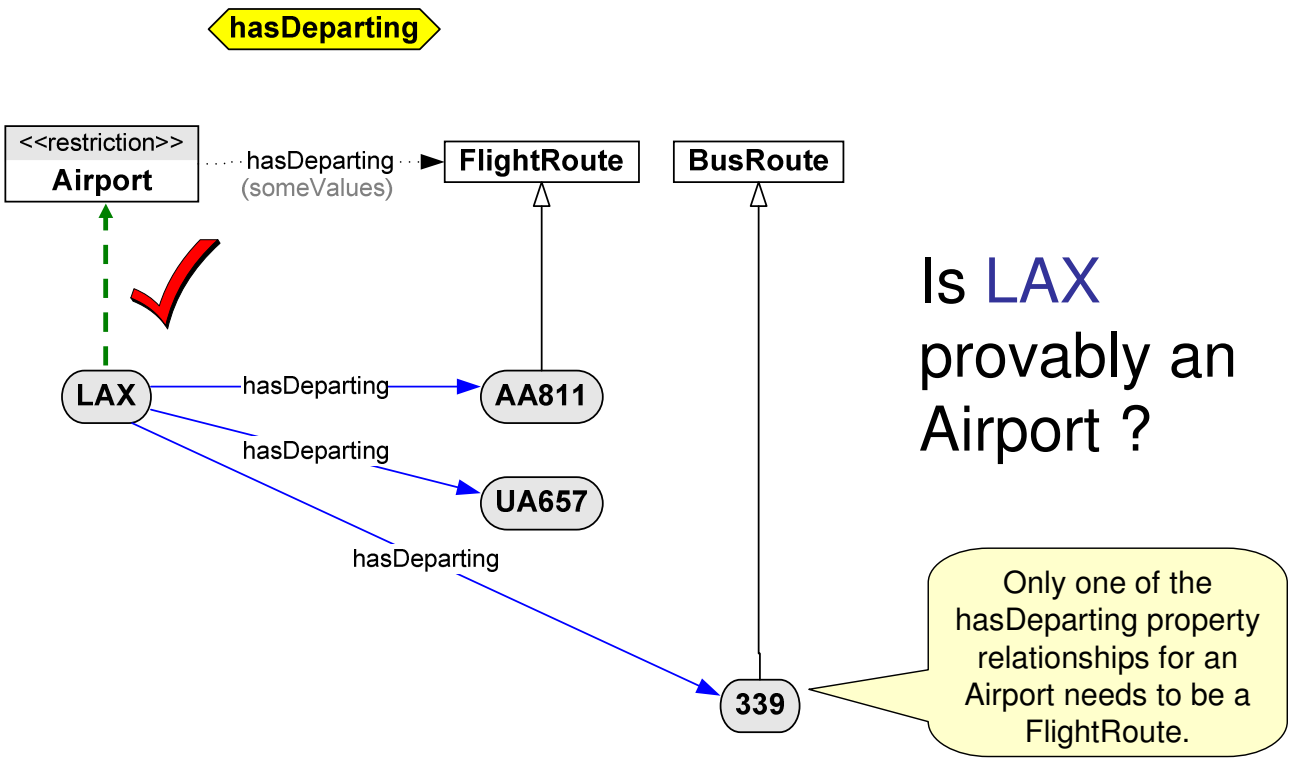
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# someValuesFrom

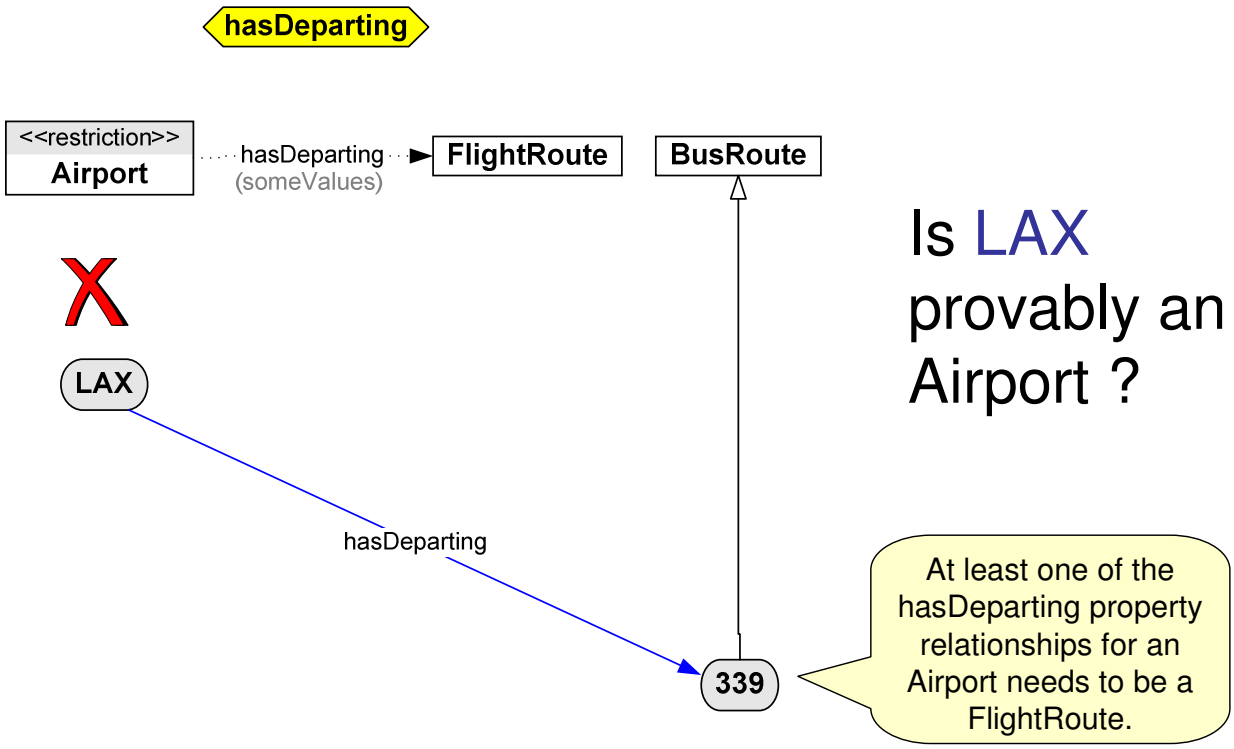



Is LAX provably an Airport ?

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# someValuesFrom



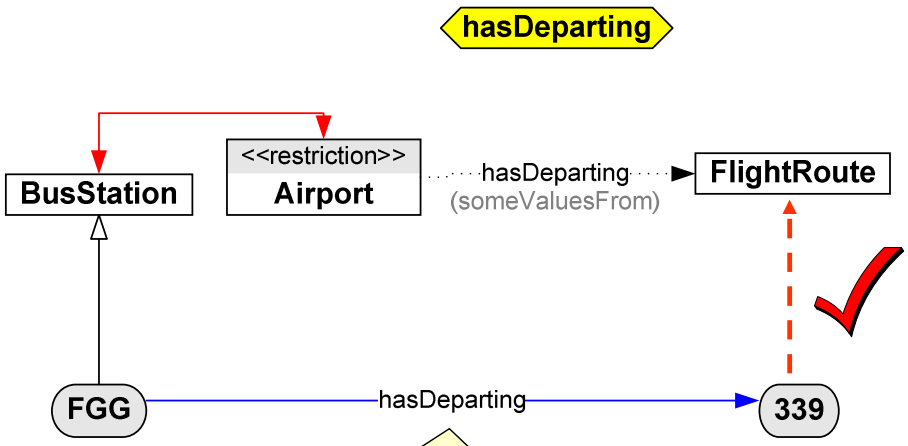

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# someValuesFrom




Is 339 provably NOT a FlightRoute ?

- If 339 was a FlightRoute in some models, then FGG would be an Airport in some models.
- But Airports and BusStations are disjoint.
- So FGG will never be an Airport in any model.
- So 339 will never be a FlightRoute in any model.

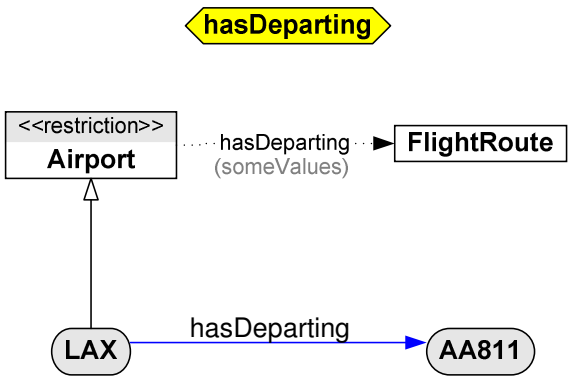
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Declaration without evidence



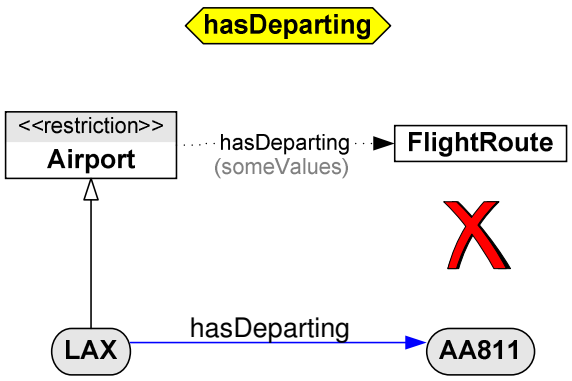

Is this a consistent ontology ?



 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Declaration without evidence




Is AA811 provably a FlightRoute?

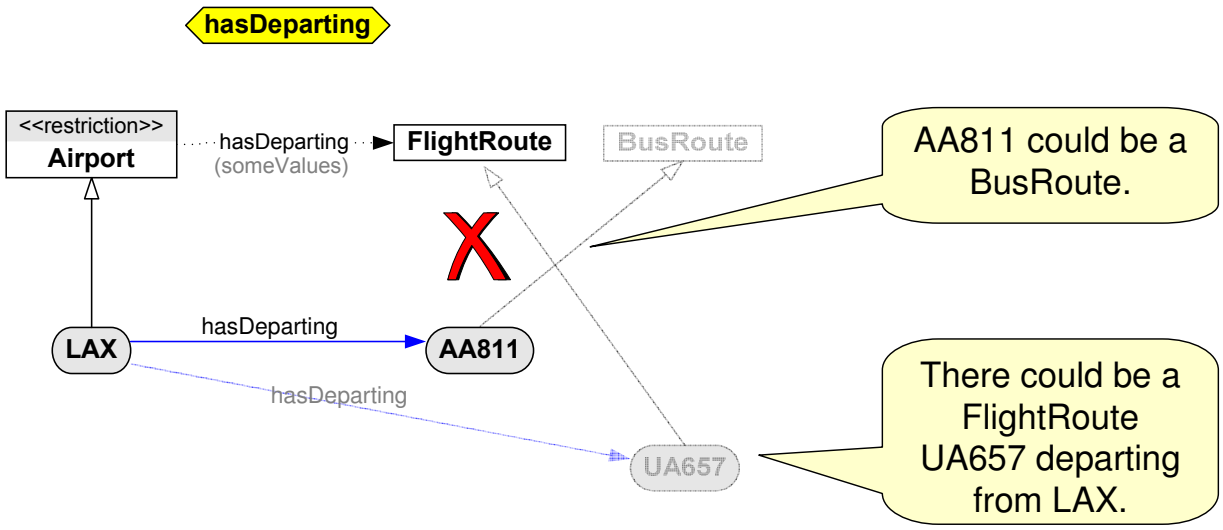
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Declaration without evidence




Just because an individual is declared to be a member of a class does not mean all the properties required of that class have been asserted. They may still be unknown.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

If you think about it this is how humans operate. We categorize things all the time without knowing everything about them.

## Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Some Implications

- An assertion is not provable if another assertion, consistent with the ontology, could be made that would disprove it.
- In other words, there is at least one model of the ontology where the assertion is untrue.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Some Implications

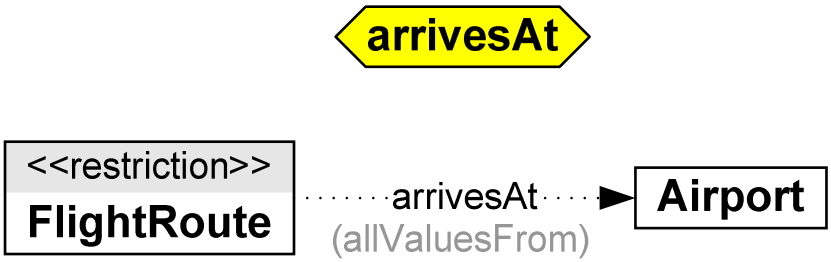
- Property restrictions are not constraints in the way that referential integrity declarations are in a standard data base. Adding a property to an individual will never result in an error message.
- We can know that a property must exist without knowing its value. E.g., since LAX is an Airport we know that it has at least one departing Flight, but we don't necessarily know what that flight is.
- It is possible to create an inconsistent ontology by:
  1. Declaring an instance is a member of a restriction class.
  2. Later giving it properties inconsistent with that class.

  (But you still won't get an error message.)

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# allValuesFrom




FlightRoute is the name of a restriction class defining the set of all individuals where all the arrivesAt property values belong to the Airport class.

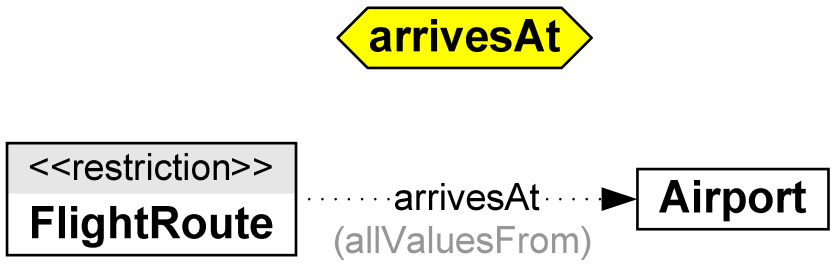
*Alternatively:*

A FlightRoute arrivesAt only Airports.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# allValuesFrom




```
<owl:Class rdf:ID="FlightRoute">
    <owl:equivalentClass>

        <owl:Restriction>
            <owl:onProperty rdf:resource="#arrivesAt"/>
            <owl:allValuesFrom rdf:resource="#Airport"/>
        </owl:Restriction>

    </owl:equivalentClass>
</owl:Class>
```

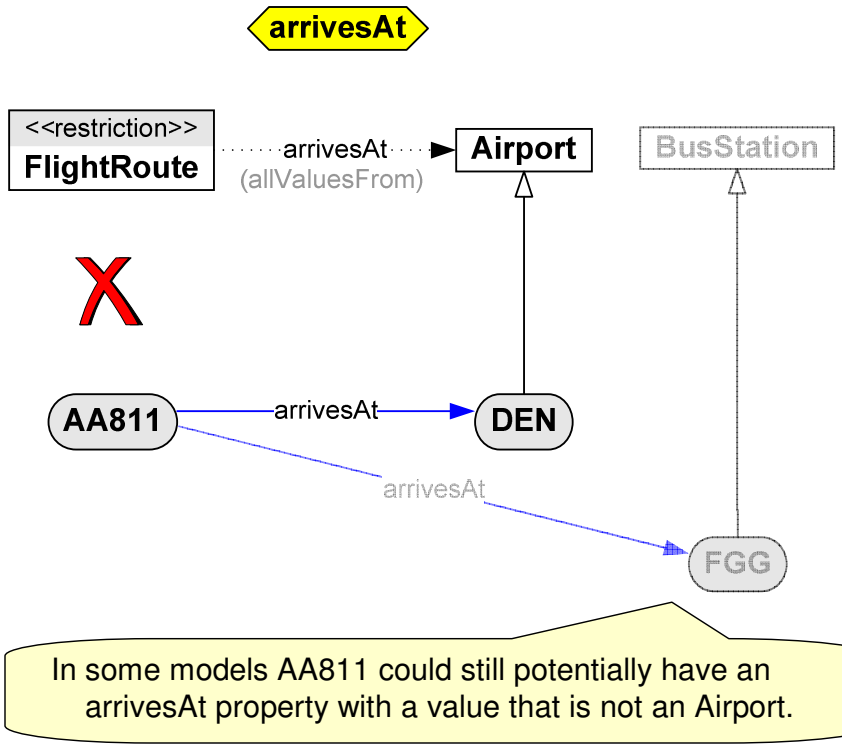
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# allValuesFrom




Is AA811 provably a Flight Route ?

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

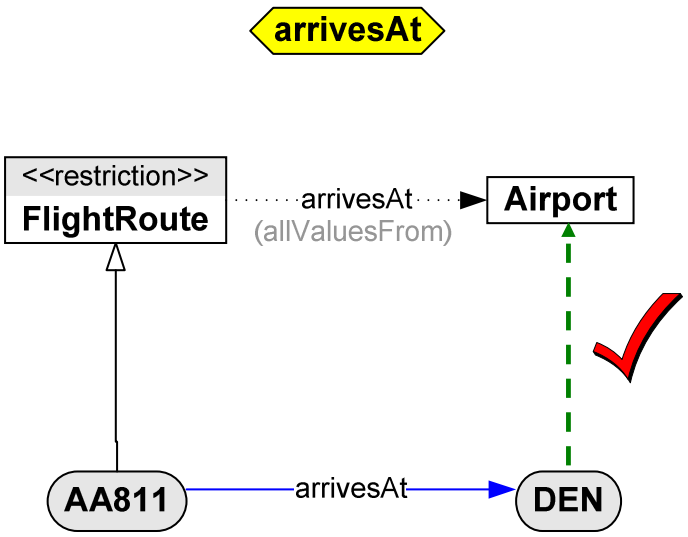
Strictly speaking FGG would potentially have to be a Thing as we had not introduced the bus station class in the previous slide.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# allValuesFrom




Is DEN provably an Airport ?

- If DEN was not an Airport in some models, then AA811 would not be a FlightRoute in some models.
- But AA811 is declared to be a FlightRoute in all models.
- So DEN must be an Airport in all models.

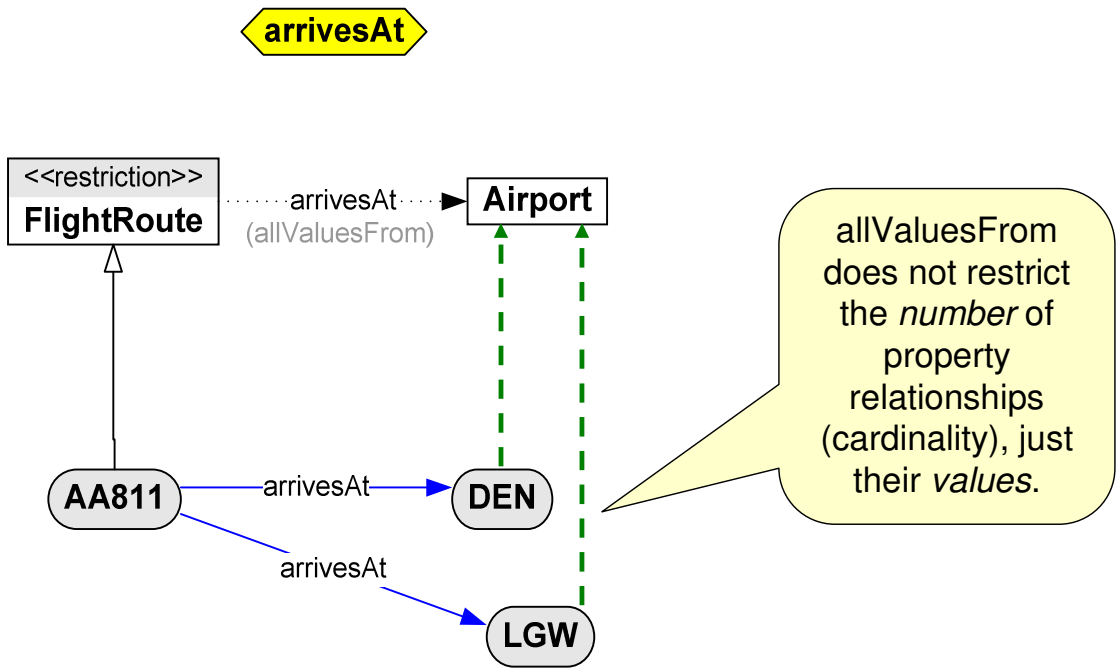
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# allValuesFrom

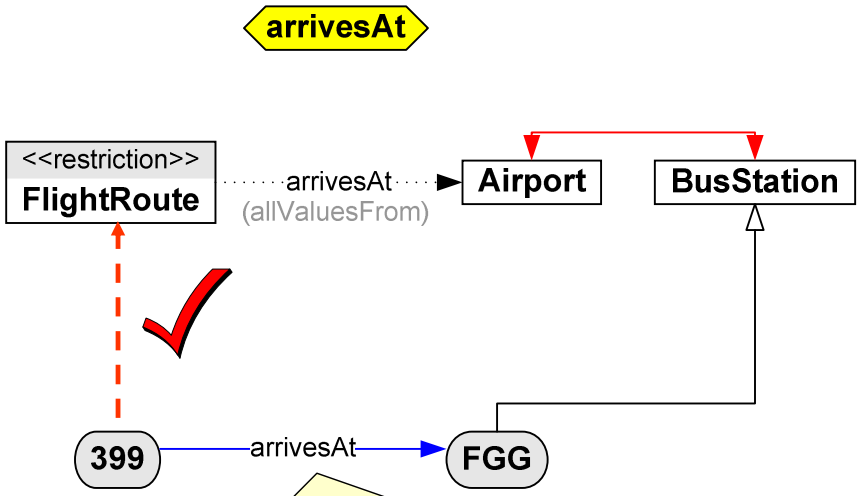



 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

If you want to say that all a flight route can depart form only one airport use a function l property or a cardinality constraint.

## Student Notes:

# allValuesFrom




Is 399 provably NOT a FlightRoute ?

- If *399* was not a *FlightRoute* in some models, then in some models *399* would have to arrive at only *Airport*s.
- But we declared that in all models 399 arrives in at least one *BusStation*.
- So *399* is not a *FlightRoute* in any model.

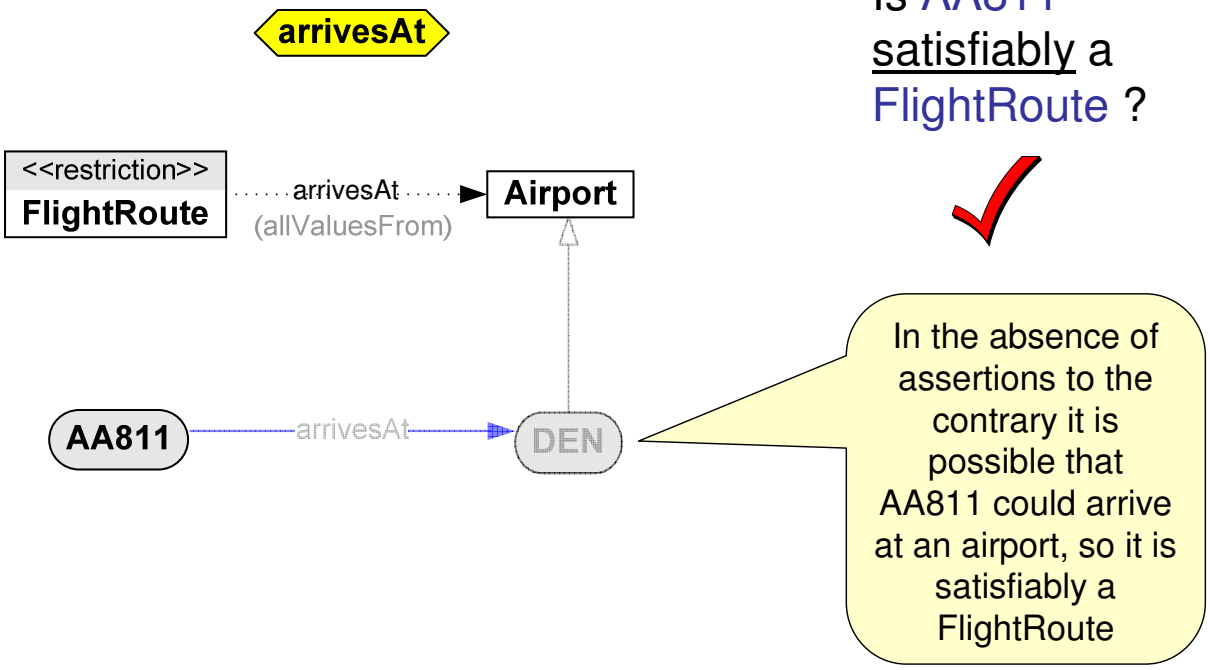
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# allValuesFrom




Is AA811 satisfiably a FlightRoute ?

In the absence of assertions to the contrary it is possible that AA811 could arrive at an airport, so it is satisfiably a FlightRoute

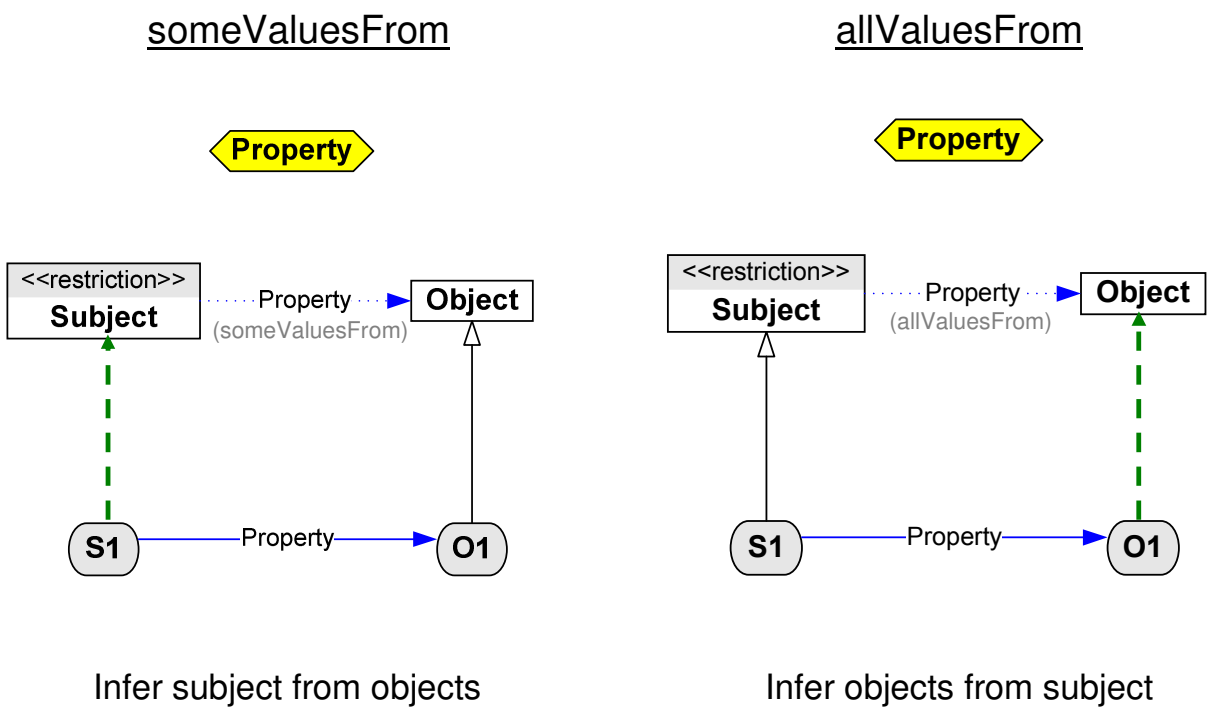
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# someValuesFrom vs. allValuesFrom




Infer subject from objects

Infer objects from subject

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

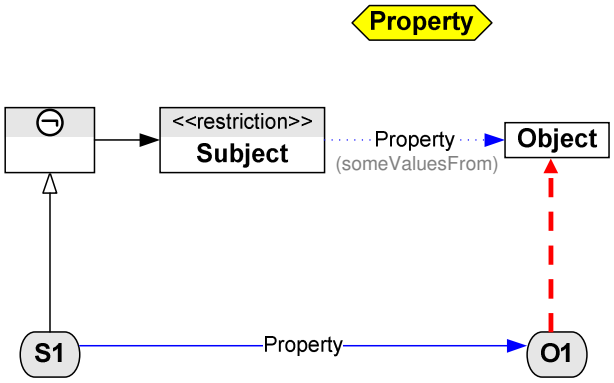
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

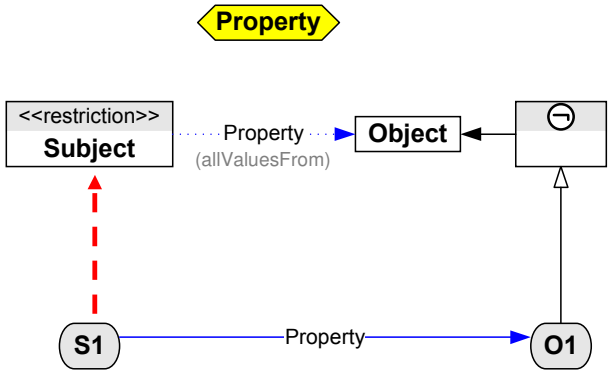
# someValuesFrom vs. allValuesFrom

## someValuesFrom




Infer “not an object”
from “not a subject”

## allValuesFrom




Infer “not a subject”
from “not an object”

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Datatype Restrictions

- For functional properties, someValuesFrom and allValuesFrom are semantically equivalent.
- Remember that in Construct, all data type properties are assumed to be Functional.
- That's why you don't get the option to select this when editing Datatype restrictions. (If you export the OWL it actually generates the allValuesFrom construct.)

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

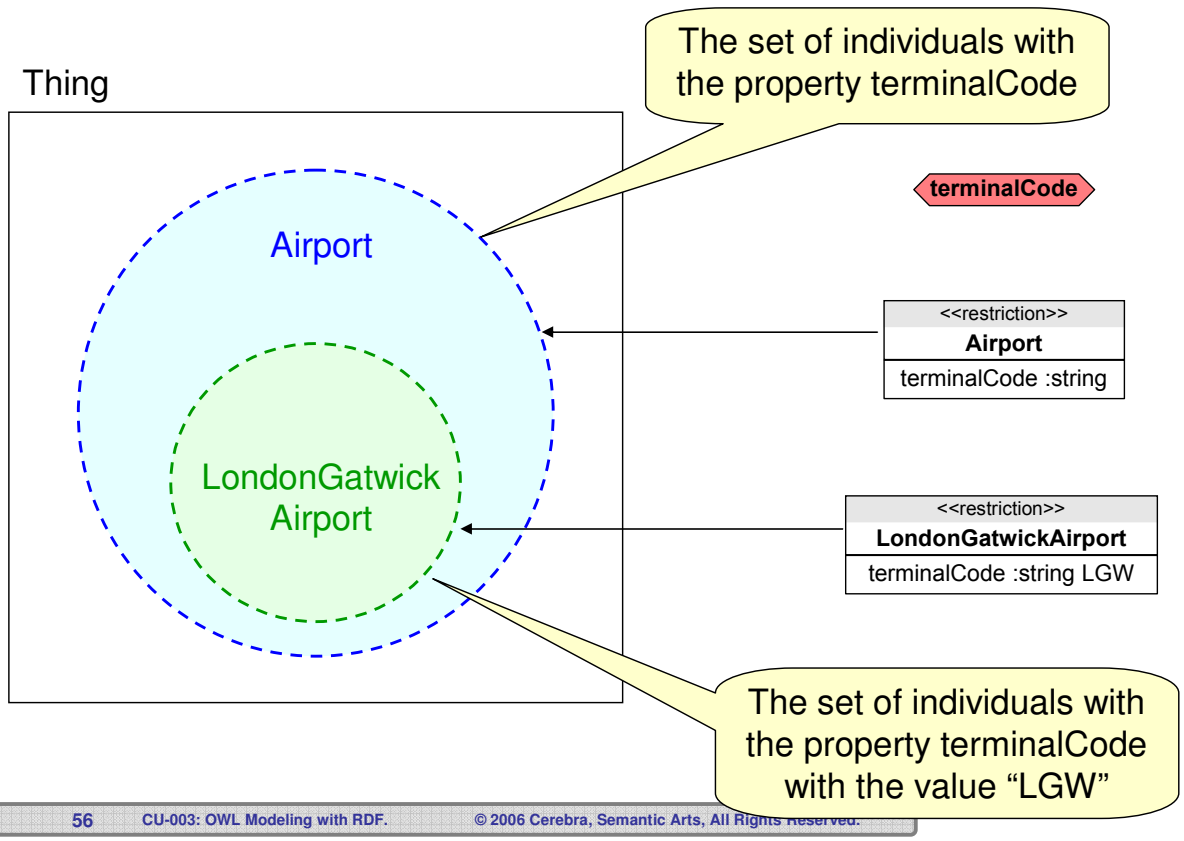
Some of 1 and all of 1 are essentially the same

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# hasValue Restrictions




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

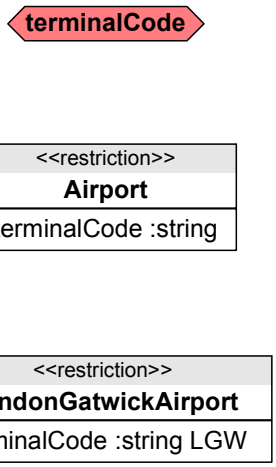
Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# hasValue Restrictions

```
<owl:Class rdf:ID="Airport">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#terminalCode"/>
      <owl:allValuesFrom rdf:resource="http://...#string"/>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>

<owl:Class rdf:ID="LondonGatwickAirport">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#terminalCode"/>
      <owl:hasValue rdf:datatype="http://...#string">LGW</owl:hasValue>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Enumerated Data Values owl:DataRange

- OWL DL allows data ranges to be specified using the owl:oneOf, rdf:list, rdf:first and rdf:rest constructs.
- You can use the owl:oneOf construct to enumerate values for datatype property restrictions in the same way as you can use it to enumerate individuals in an enumerated class.
- E.g., the restriction class LondonAirports is the class of all individuals with the property terminalCode and values from the list LGW, LHR, LTN, STN, LCY.
- Construct does not currently support this. You can work around this by creating the union of a number of hasValue restriction classes.
- Cerebra Server will correctly interpret the OWL.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Cardinality Restrictions

- By default OWL classes can have an arbitrary number of values associated with the classes' properties. Cardinality constraints restrict the number of values a property can have for each subject for a particular class.

- OWL DL supports
  - owl:minCardinality
  - owl:maxCardinality
  - owl:cardinality

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# owl:minCardinality

- At least the specified number of distinct values must be associated with members of the restricted class.

```
<owl:Class rdf:ID="IndirectFlightRoute">
    <owl:equivalentClass>

        <owl:Restriction>
            <owl:onProperty rdf:resource=#hops/>
            <owl:minCardinality
                rdf:datatype='http://..#nonNegativeInteger'>3</owl:minCardinality>
        </owl:Restriction>

    </owl:equivalentClass>
</owl:Class>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# maxCardinality

- Limits the number of unique values for a particular class property.
  - A maximum cardinality of 0 is equivalent to saying the property is not allowed for that class.
  - A maxCardinality of 1 is equivalent to making the property functional for that class only. If more than one relationship is added for that property, the subjects are assumed to be equivalent.

```
<owl:Class rdf:ID="NonStopFlightRoute">
    <owl:equivalentClass>

        <owl:Restriction>
            <owl:onProperty rdf:resource=#hops/>
            <owl:maxCardinality
                rdf:datatype='http://..#nonNegativeInteger'>2</owl:maxCardinality>
        </owl:Restriction>

    </owl:equivalentClass>
</owl:Class>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# owl:Cardinality

- Absolute Cardinality - Restricts the number of unique values to exactly the specific number.

```
<owl:Class rdf:ID="TheEarth">
    <owl:equivalentClass>

        <owl:Restriction>
            <owl:onProperty rdf:resource=#borders/>
            <owl:Cardinality
                rdf:datatype='http://..#nonNegativeInteger'>0</owl:Cardinality>
        </owl:Restriction>

    </owl:equivalentClass>
</owl:Class>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Cardinality Notes

- Because the reasoner cannot assume that all properties have been asserted, membership cannot be inferred based on maxCardinality or Cardinality.
- If the class has more than the maximum, or other than the absolute cardinality, the reasoner can infer non membership.
- Similarly, the reasoner can infer membership based on minCardinality but can never infer non membership.
- Construct does not support cardinality restrictions.
- Cerebra Server supports cardinality values of 0 or 1 but no others.

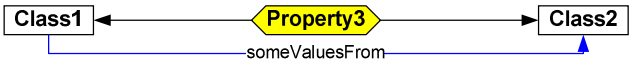
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Alternative Cardinality Patterns

| Min | Max | | Alternative To OWL restriction |
|---|---|---|---|
| 0 | 1 |   | owl:maxCardinality 1 |
| 1 | 1 |   | owl:minCardinality 1<br>owl:maxCardinality 1 |
| 1 | N |   | owl:minCardinality 1 |
| 0 | N |   | No restriction |

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Note that using functional properties has the disadvantage that Functionality is global whereas maxCardinality=1 is a local restriction.

## Student Notes:

Cerebra UNIVERSITY

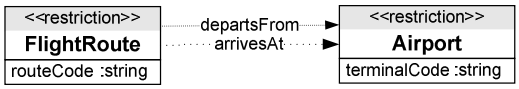
CU-003
02/2006
Module 5

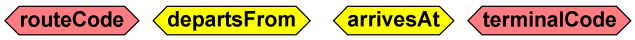
# Equivalence vs. Subclass Semantics

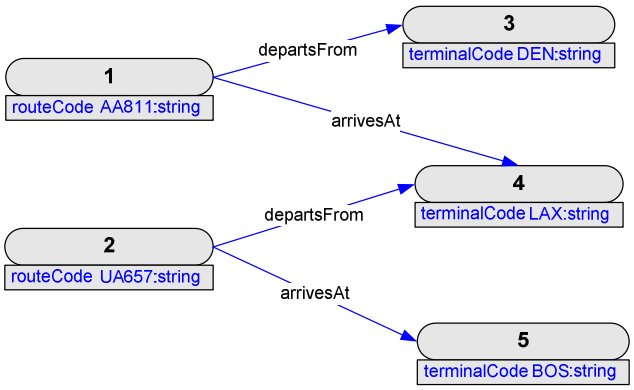
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Exercise 2




1. Load the ontology Module5_Exercise2a.
2. Query the ontology first for provable then for satisfiable FlightRoutes.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Answer 2

## Query FlightRoutes

```
for $i in instances()
where every $m in models() satisfies
    member($i, FlightRoute, $m)
return $i
```

```
<RESULT>
 http://www.cerebra.com/sa#1
 http://www.cerebra.com/sa#2
</RESULT>
```

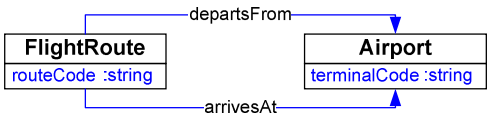
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

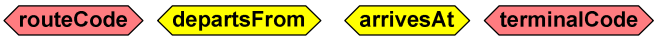
Student Notes:

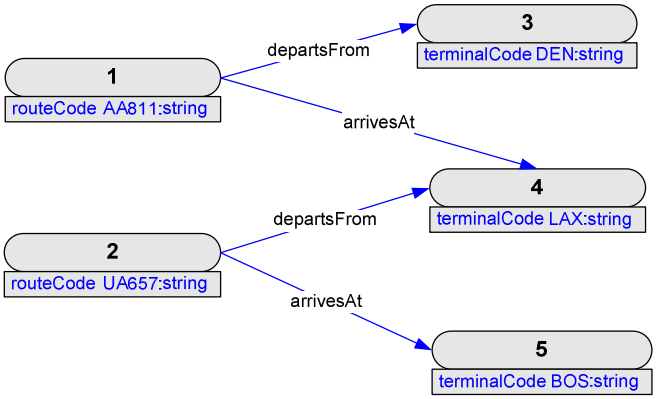
Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Exercise 3




1. Load the ontology Module5_Exercise2b.
2. Query the ontology first for provable then for satisfiable FlightRoutes.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Answer 3

## Query FlightRoutes

```
for $i in instances()
where every $m in models() satisfies
    member($i, FlightRoute, $m)
return $i
```

```
<RESULT>
</RESULT>
```

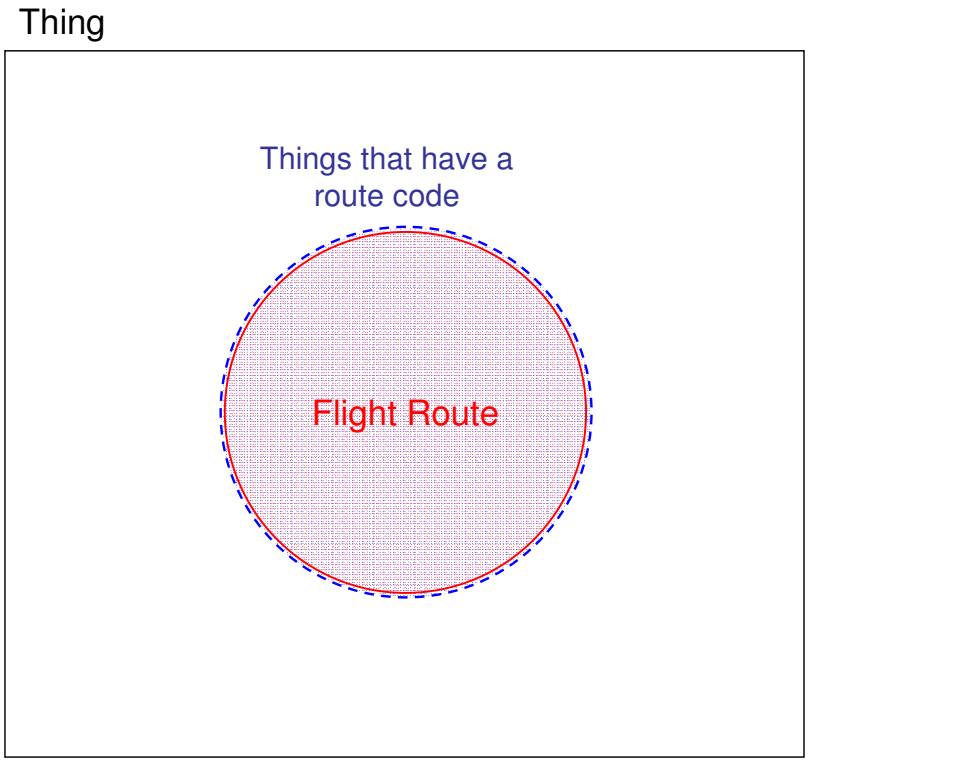
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

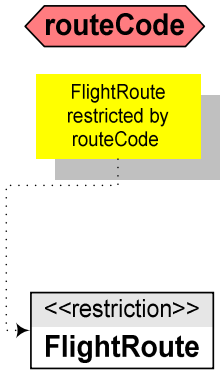
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Add a property definition…



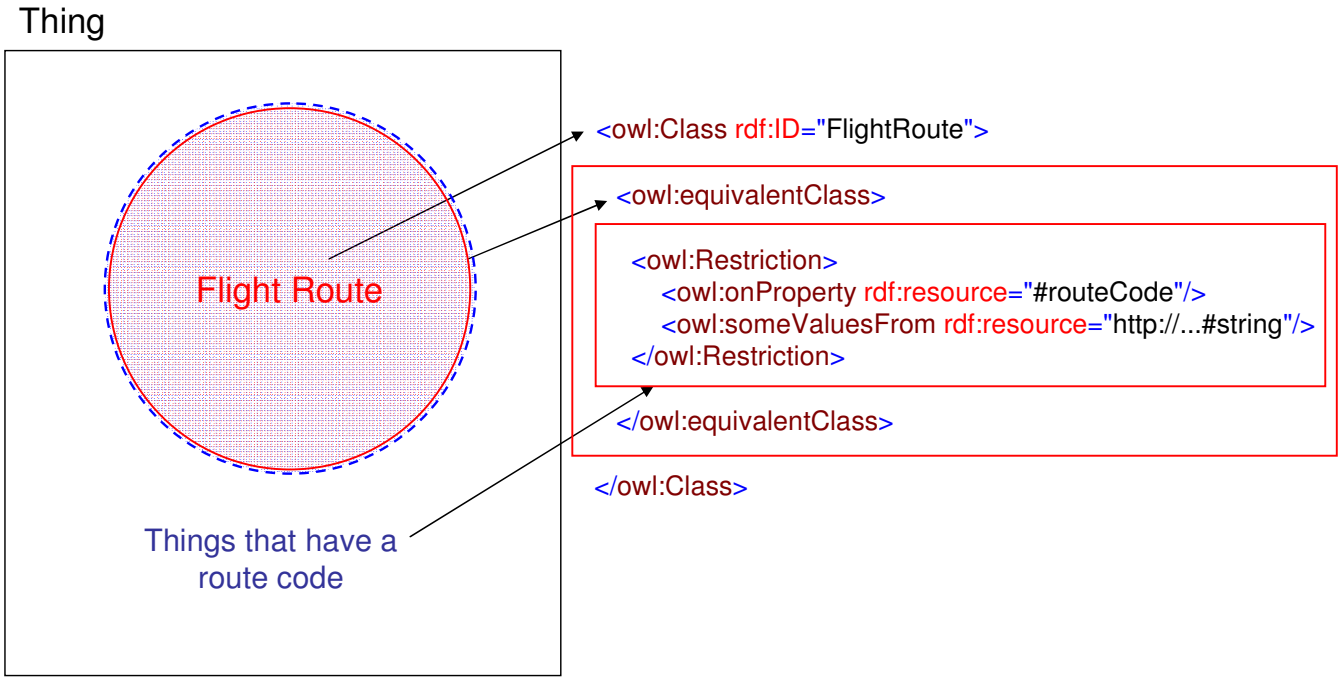

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

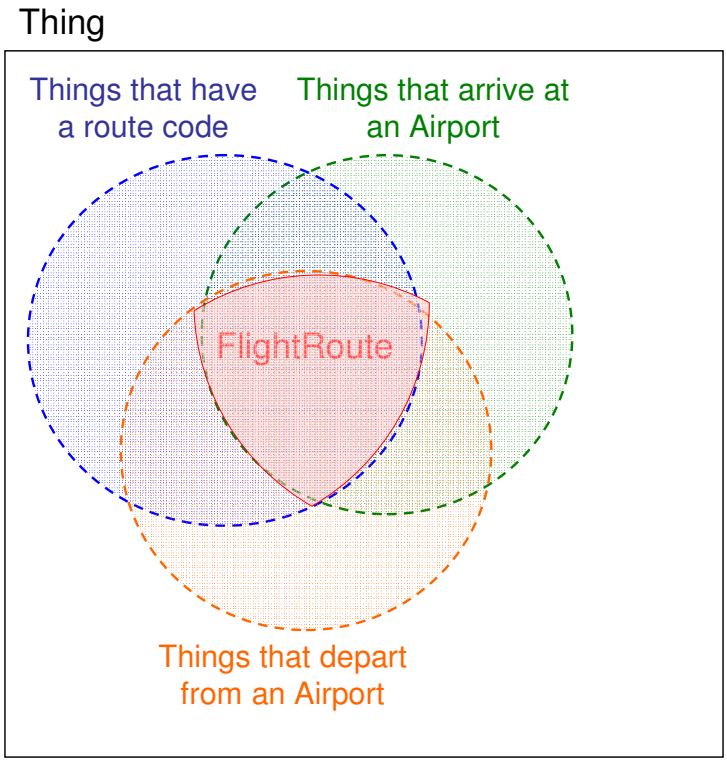
# This corresponds to the OWL syntax

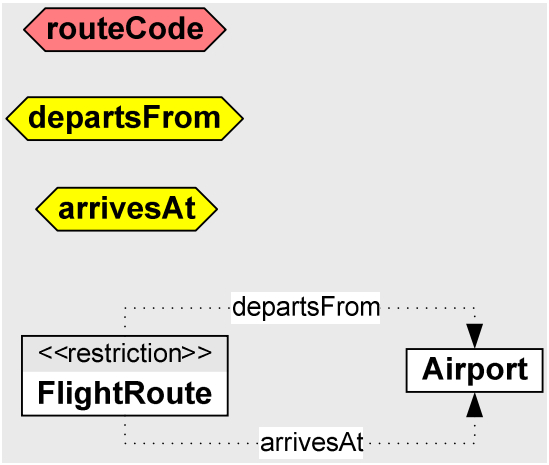



 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Add the object property definitions...




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

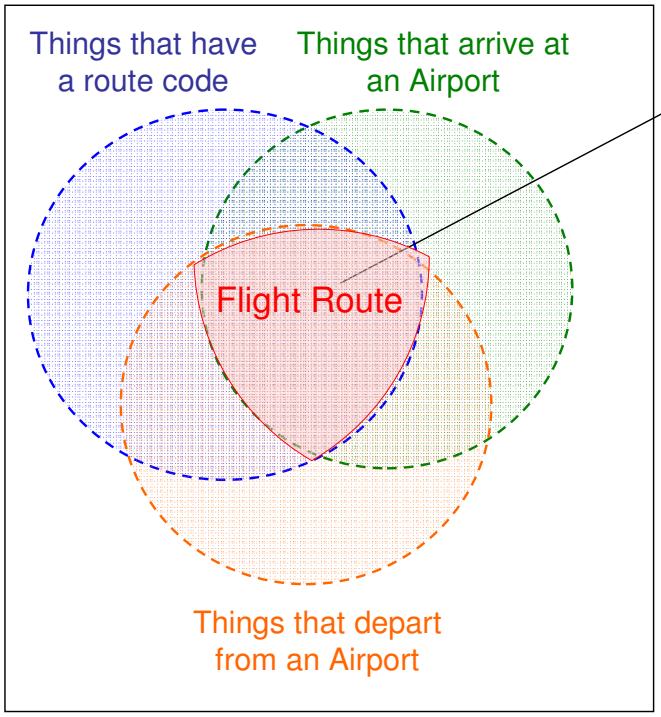
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# The OWL for FlightRoute

Thing




```
<owl:Class rdf:ID="FlightRoute">
  <owl:equivalentClass>
   <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">

     <owl:Restriction>
      <owl:onProperty rdf:resource="#routeCode"/>
      <owl:allValuesFrom rdf:resource="http://...#string"/>
     </owl:Restriction>

     <owl:Restriction>
      <owl:onProperty rdf:resource="#arrivesAt"/>
      <owl:someValuesFrom rdf:resource="#Airport"/>
     </owl:Restriction>

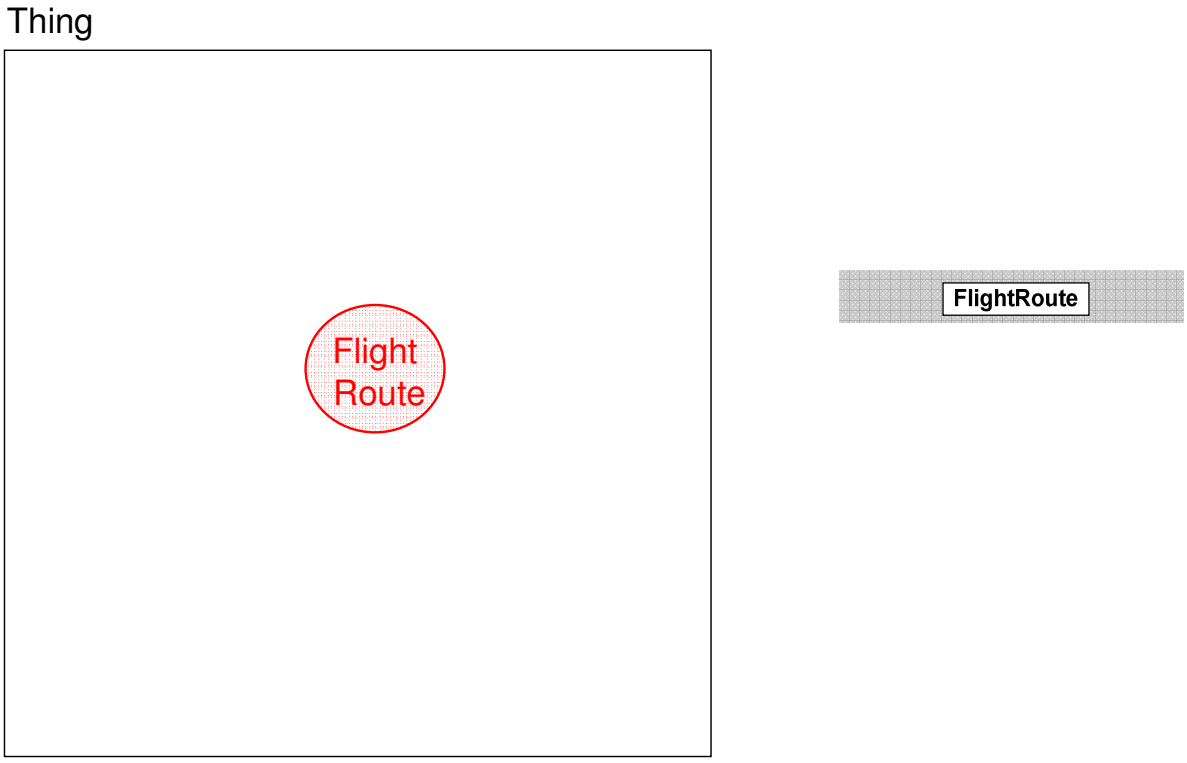
     <owl:Restriction>
      <owl:onProperty rdf:resource="#departsFrom"/>
      <owl:someValuesFrom rdf:resource="#Airport"/>
     </owl:Restriction>

    </owl:intersectionOf>
   </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Declare a standard class…



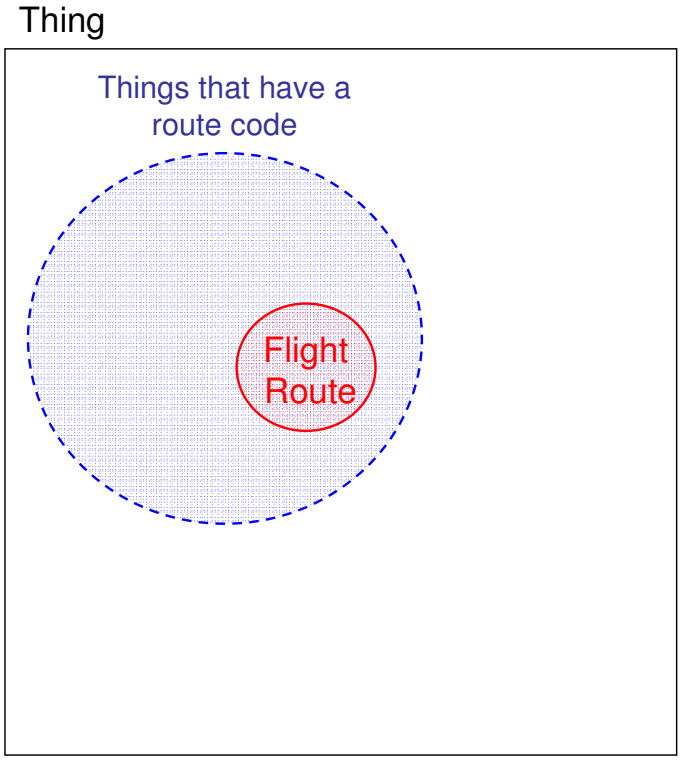

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

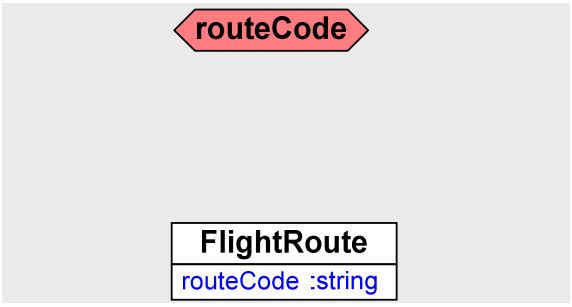
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Add a datatype property…




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

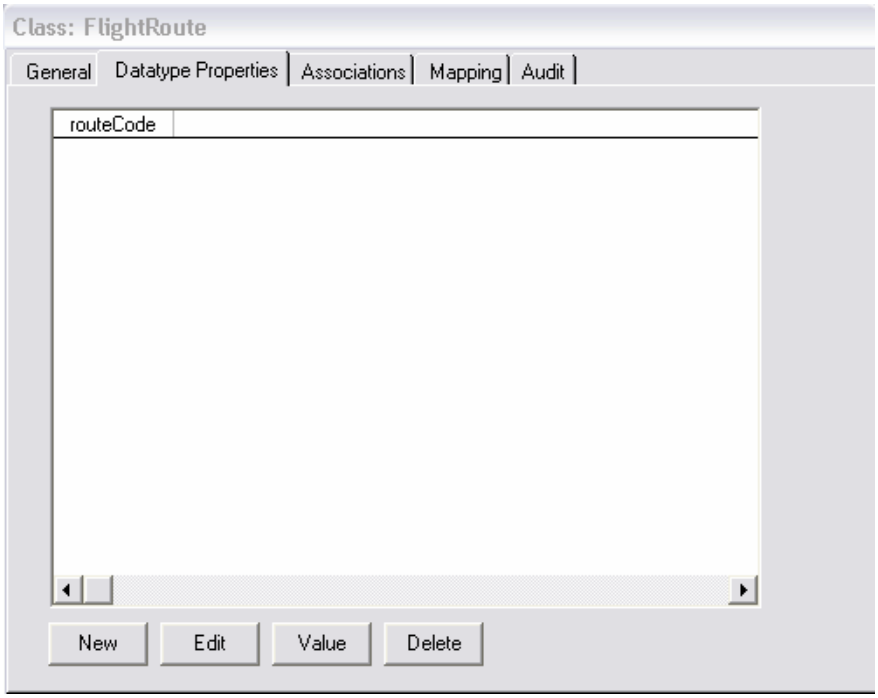
It is possible that airport should in fact overlap with the anonymous class. It seems like if might be possible for airport to have a route code as we have not stated that it cannot. Or when a class is defined does it implicitly have no properties.

## Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Adding a Datatype Property to a class in Construct



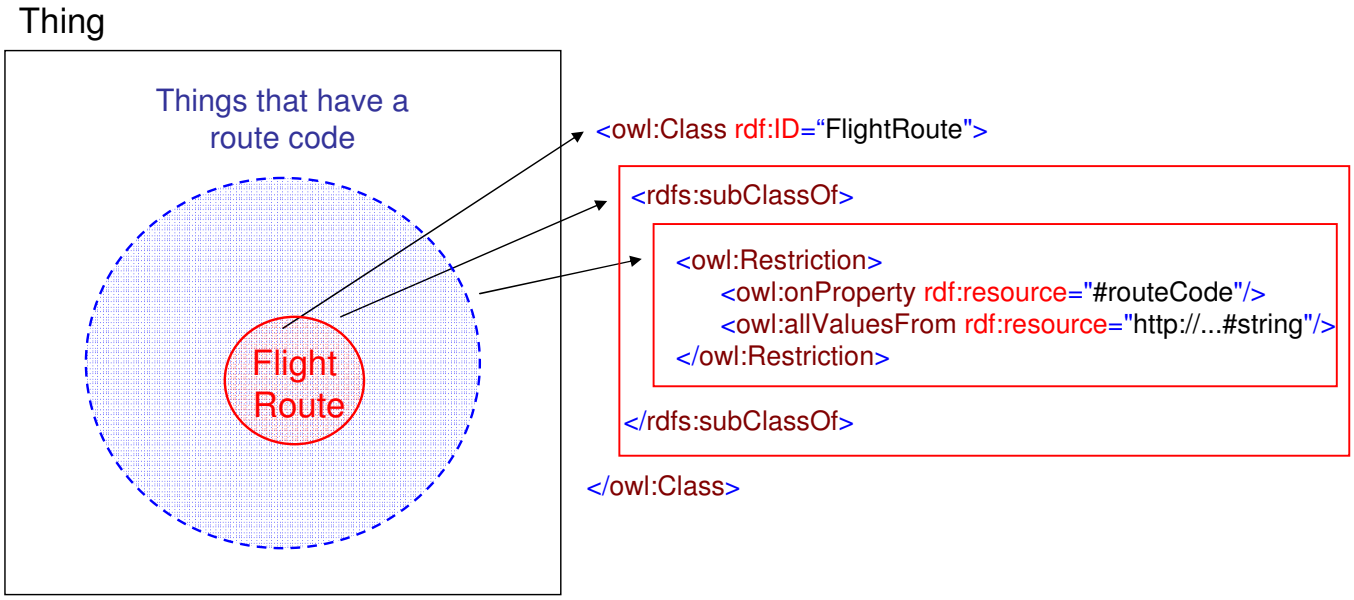

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

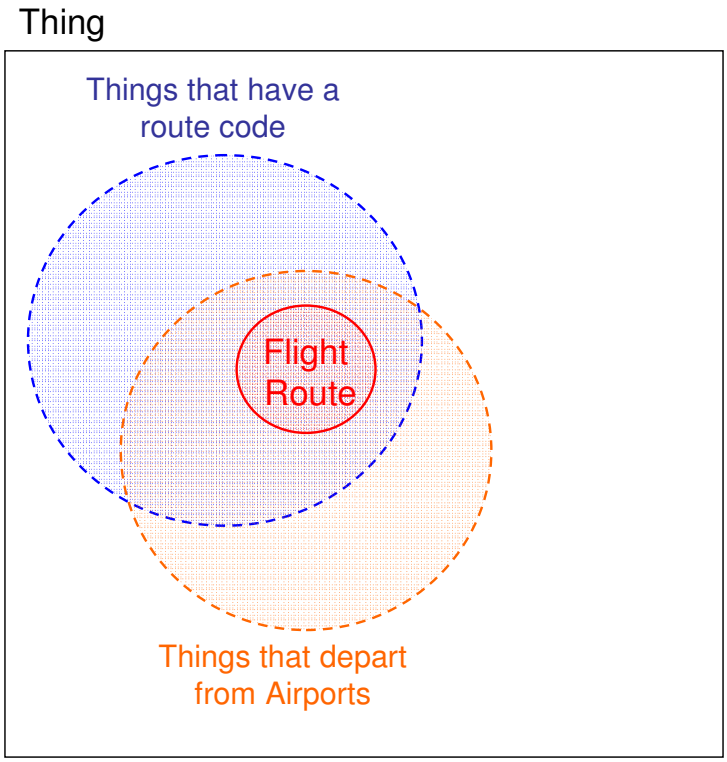
# This corresponds to the OWL syntax

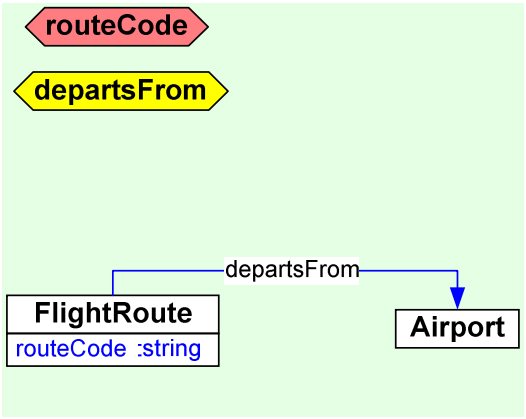



 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Add an object property…



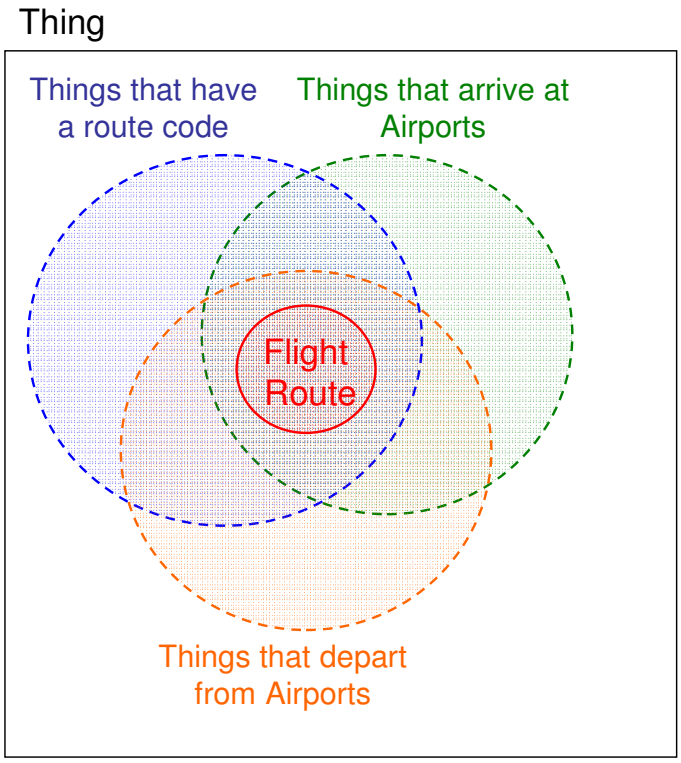

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

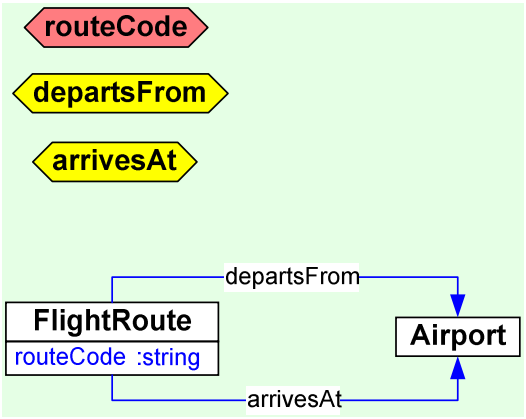
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Add another object property



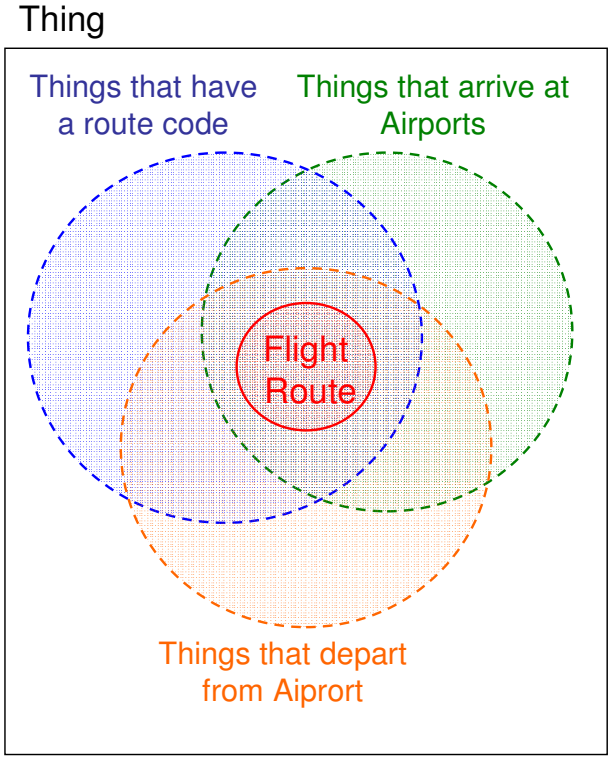

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# The OWL for FlightRoute




```
<owl:Class rdf:ID="FlightRoute">

  <rdfs:subClassOf>
     <owl:Restriction>
          <owl:onProperty rdf:resource="#routeCode"/>
          <owl:allValuesFrom rdf:resource=
              "http://...#string"/>
     </owl:Restriction>
  </rdfs:subClassOf>

  <rdfs:subClassOf>
     <owl:Restriction>
          <owl:onProperty rdf:resource="#departsFrom"/>
          <owl:someValuesFrom rdf:resource="#Airport"/>
     </owl:Restriction>
  </rdfs:subClassOf>

  <rdfs:subClassOf>
      <owl:Restriction>
          <owl:onProperty rdf:resource="#arrivesAt"/>
          <owl:someValuesFrom rdf:resource="#Airport"/>
      </owl:Restriction>
  </rdfs:subClassOf>

</owl:Class>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# One way to think about this

- Each someValuesFrom property restriction defines an anonymous class of all things that have that property.
- All of those anonymous classes intersect with each other.
- When we associate a class with one or more properties, we are essentially declaring that the class is a subtype of one of those intersections.

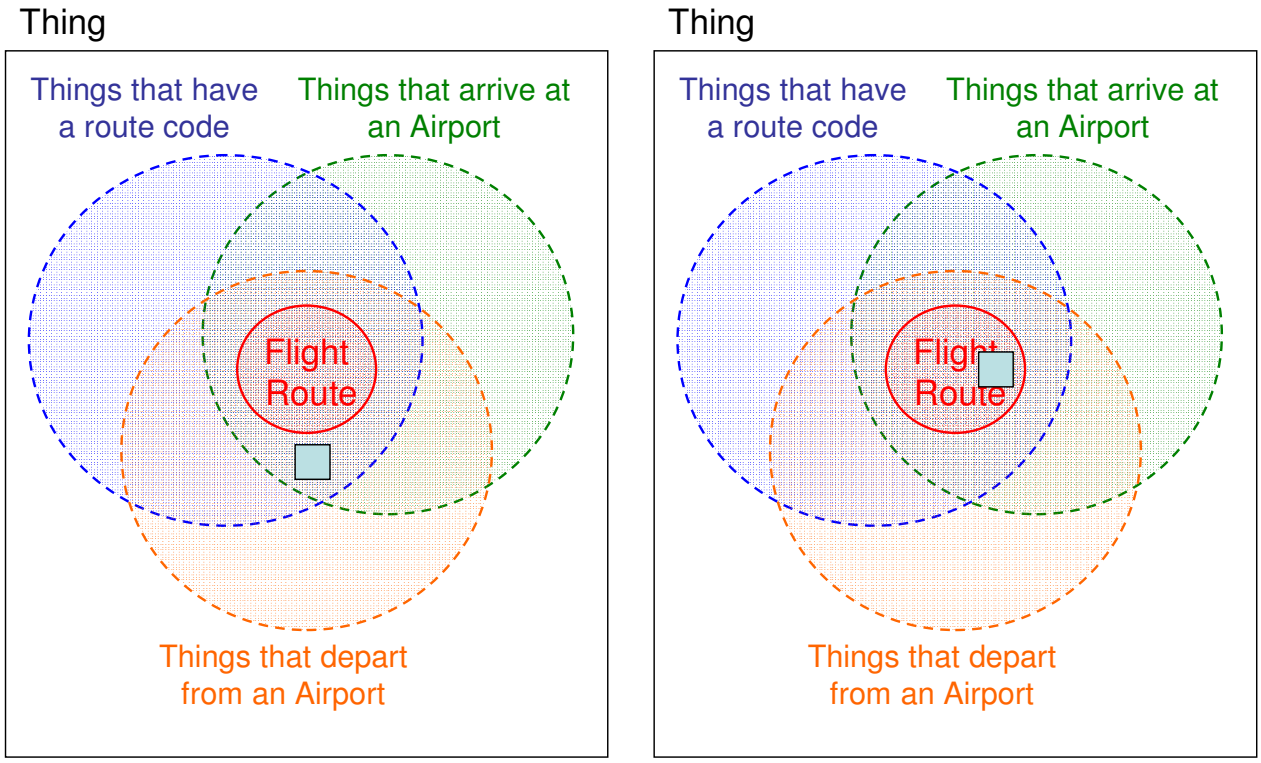
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Instances with all three properties may or may not be FlightRoutes



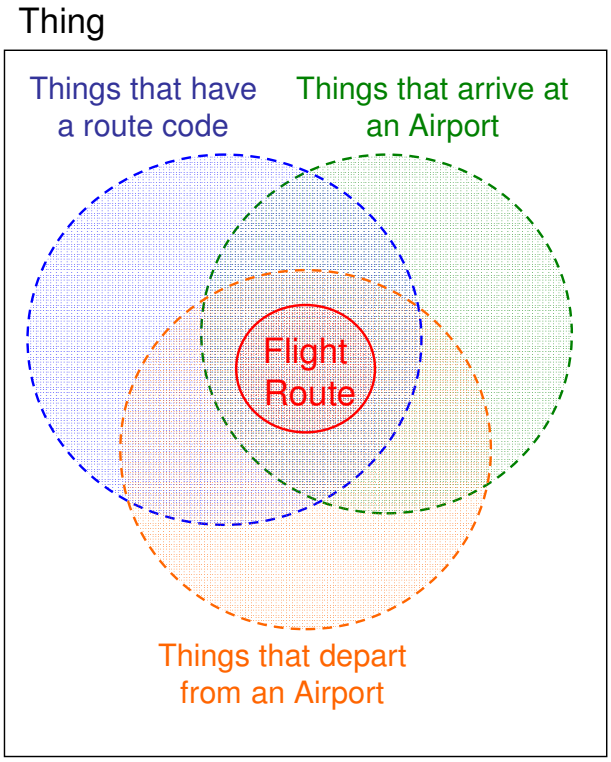

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

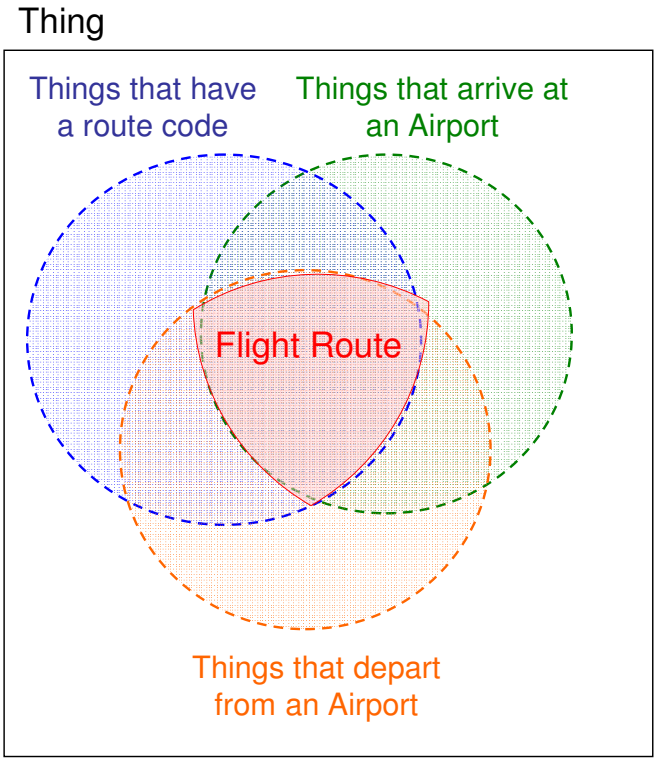
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Subclass Semantics vs. Equivalence Semantics



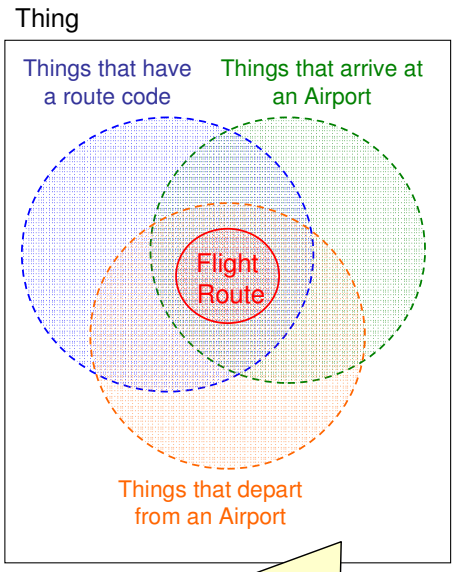

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

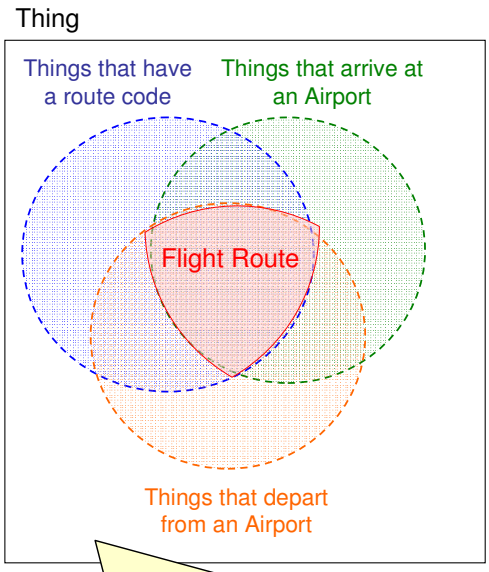
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Necessary vs. Necessary and Sufficient




For an instance to be a FlightRoute it is NECESSARY that it have a routeCode, arriveAt an Airport and departFrom an Airport, but that is NOT enough evidence to infer it is a FlightRoute.

For an instance to be a FlightRoute it is NECESSARY that it have a routeCode, arriveAt an Airport and departFrom an Airport and that is SUFFICIENT evidence to infer it is a FlightRoute

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Subclass Semantics vs. Equivalence Semantics

```
<owl:Class rdf:ID="FlightRoute">

  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#routeCode"/>
      <owl:allValuesFrom rdf:resource=
        "http://...#string"/>
    </owl:Restriction>
  </rdfs:subClassOf>

  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#departsFrom"/>
      <owl:someValuesFrom rdf:resource="#Airport"/>
    </owl:Restriction>
  </rdfs:subClassOf>

  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#arrivesAt"/>
      <owl:someValuesFrom rdf:resource="#Airport"/>
    </owl:Restriction>
  </rdfs:subClassOf>

</owl:Class>
```

```
<owl:Class rdf:ID="FlightRoute">
  <owl:equivalentClass>
    <owl:Class>

      <owl:intersectionOf rdf:parseType="Collection">

        <owl:Restriction>
          <owl:onProperty rdf:resource="#routeCode"/>
          <owl:allValuesFrom rdf:resource="http://...#string"/>
        </owl:Restriction>

        <owl:Restriction>
          <owl:onProperty rdf:resource="#arrivesAt"/>
          <owl:allValuesFrom rdf:resource="#Airport"/>
        </owl:Restriction>

        <owl:Restriction>
          <owl:onProperty rdf:resource="#departsFrom"/>
          <owl:allValuesFrom rdf:resource="#Airport"/>
        </owl:Restriction>

      </owl:intersectionOf>

    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

 © 2006 Cerebra, Semantic Arts, All rights reserved.

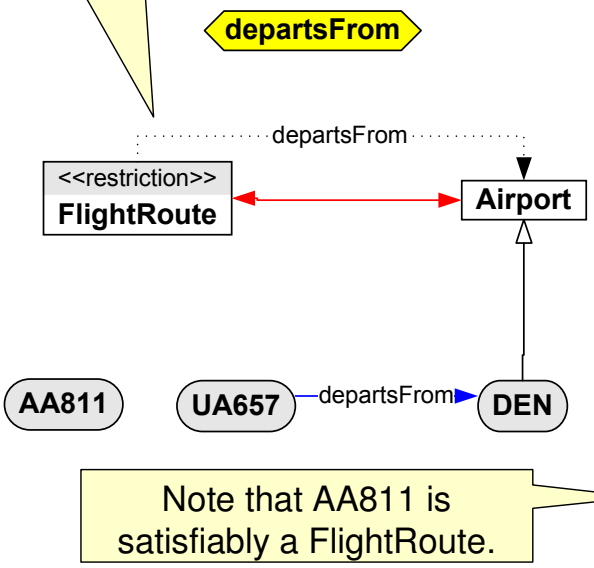
## Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# A restriction is not a constraint

FlightRoute is a restriction class of things that departFrom an Airport.




Note that AA811 is satisfiably a FlightRoute.

```
for $i in instances()
where every $m in models() satisfies
      member($i, FlightRoute,$m)
return $i
```

```
<RESULT>
http://www.cerebra.com/...#UA657
</RESULT>
```

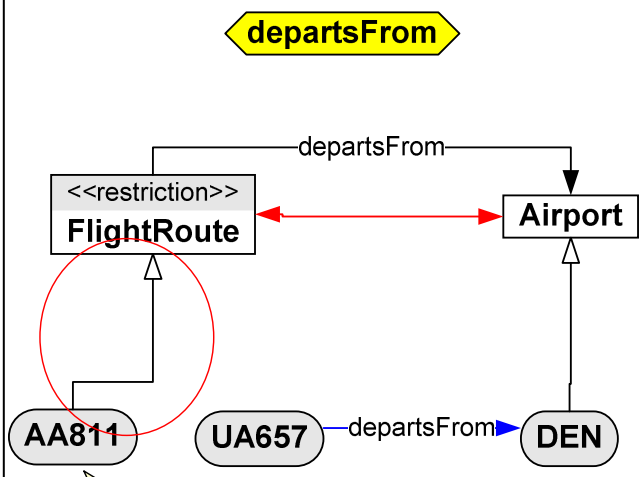
```
for $i in instances()
where some $m in models() satisfies
      member($i, FlightRoute,$m)
return $i
```

```
<RESULT>
http://www.cerebra.com/...#UA657
http://www.cerebra.com/...#AA811
</RESULT>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# A restriction is not a constraint




AA811 is now provably a FlightRoute even though it does not have a departsFrom property.

```
for $i in instances()
where every $m in models() satisfies
      member($i, FlightRoute,$m)
return $i
```

```
<RESULT>
http://www.cerebra.com/...#AA811
http://www.cerebra.com/...#UA657
</RESULT>
```

```
for $i in instances()
where some $m in models() satisfies
      member($i, FlightRoute,$m)
return $i
```

```
<RESULT>
http://www.cerebra.com/...#AA811
http://www.cerebra.com/...#UA657
</RESULT>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# What you can declare…

- You can declare an instance to be a member of a restriction class as long as it is satisfiably a member of that class.

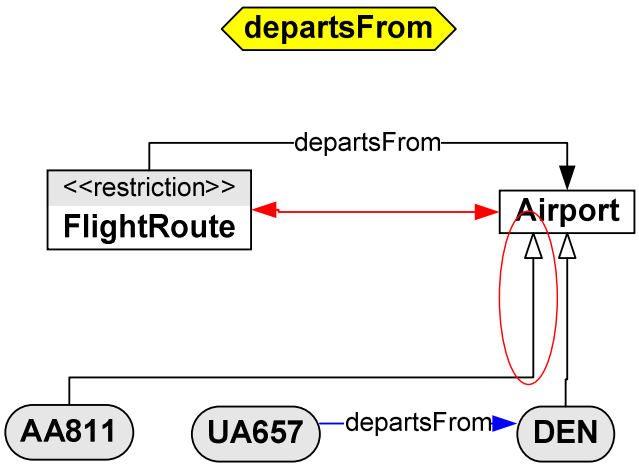
*In other words*

- As there are no assertions in the ontology that are inconsistent with AA811 being a Flight Route, we can declare it to be one, even though we don't know its departure Airport.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# A non-satisfiable example




```
for $i in instances()
where every $m in models() satisfies
      member($i, FlightRoute,$m)
return $i
```

```
<RESULT>
http://www.cerebra.com/...#UA657
</RESULT>
```

```
for $i in instances()
where some $m in models() satisfies
      member($i, FlightRoute,$m)
return $i
```

```
<RESULT>
http://www.cerebra.com/...#UA657
</RESULT>
```

AA811 is no longer satisfiably a FlightRoute as things cannot be both FlightRoutes and Airports at the same time.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# A restriction is not a constraint but…

- You can use the open world characteristics of the ontology to query instances with potential missing data, e.g., what FlightRoute instances are satisfiable but not provable?
- You can use the metadata in the ontology to drive an external validation process.
- You can add instances to the ontology and declare them to be members of a class, then test to see if that makes the ontology inconsistent.
- Some inference engines allow you to query to see where the inconsistency is in the ontology. In Cerebra you can run a query on an inconsistent ontology to return all classes that are equivalent to nothing. These have potentially erroneous class definitions.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 5

# Exercise 3

- Open the ontology `Module5_Exercise3`.
- Design, implement and test extensions to the ontology to include the restriction class:

  USInternationalFlightRoute
- Bonus Exercise - Design a restriction class

  InternationalFlightRoute

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Stress that we want a restriction class; not a query.
Not declare any new membership everything must be derived using restriction classes.
Have to design from the bottom up. Use the filter analogy.
Write out the problem in terms that are part of the ontology.
You are not allowed to enumerate all the flights in using a onOf. Restriction.
In the ontology flight route is using subclass semantics so our solution may be incomplete.

Student Notes:

# Module Review

- Instructor led review session
- Pop-quiz is optional!

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# OWL-DL Modeling with RDF

## Module Six: How OWL is Different

CU-003: OWL Modeling with RDF. © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# Agenda

| | Monday, Feb 27 | Tuesday, Feb 28 | Wednesday, Mar 01 |
|---|---|---|---|
| 8 am | | | |
| 9 00 | **Module 1:** *Intro to OWL* | **Module 3:** *Equivalence, Disjointness, and Subsumption* | **Module 5:** *Complex Classes* |
| 10 00 | | | |
| 11 00 | **Module 2:** *Individuals, Properties, and Classes* | **Module 3:** *Equivalence, Disjointness, and Subsumption* | **Module 5:** *Complex Classes* |
| 12 pm | | | |
| 1 00 | **Module 2:** *Individuals, Properties, and Classes* | **Module 4:** *More Properties* | **Module 6:** *How OWL is Different* |
| 2 00 | | | |
| 3 00 | **Module 2:** *Individuals, Properties, and Classes* | **Module 4:** *More Properties* | **Module 7:** *The Complement Module*<br>**Module 8:** *Wrap up* |
| 4 00 | | | |

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# You Assume Too Much

OWL/RDF for the relationally and OO fixated.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# Some rough terminology equivalents

| Relational | Object Oriented | OWL / RDF |
|---|---|---|
| Database | Domain | Ontology |
| Row | Object | Individual |
| Table | Class | Class |
| Column | Property | Property |

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# What gets asserted ?

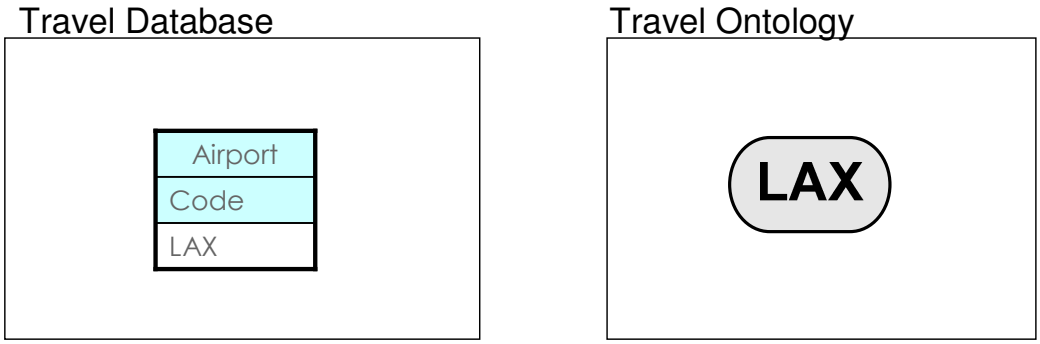
- What does the query engine assume is asserted when we add a row to a table?
- What does an object-oriented runtime engine assume is asserted when we instantiate an object?
- What does a reasoner assume is asserted when an individual is added to an ontology?

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Existence

| Assumption | Relational Query Engine | OO Runtime | Reasoner |
|---|---|---|---|
| The row/object/individual exists in the domain. | Yes | Yes | Yes |




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

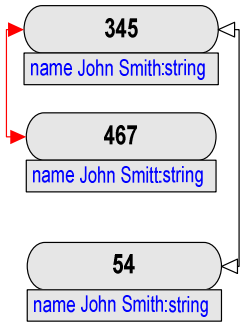
Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# Synonyms and Homonyms

| Assumption | Relational Query Engine | OO Runtime | Reasoner |
|---|---|---|---|
| The row/object/individual is distinct from any other row/object/individual in the domain. | Yes | Yes | No |

| Patient ID | Patient Name | ..... |
|---|---|---|
| 345 | John Smith | |
| 467 | John Smitt | |
| 54 | John Smith | |




The Unique Name Assumption: Individuals with different names are different individuals.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

By adding an assertion we can simply state if tow instances are the seam or different.

In a relational system we would have tow either merge two rows or split two rows in most cases a complicated operation.

This is related to the Unique Name Assumption typically made in database applications which assumes that different individuals with different names (Keys) are indeed different individuals. In this case Patient 345 and 467 are in fact the same individual.

## Student Notes:

Cerebra UNIVERSITY

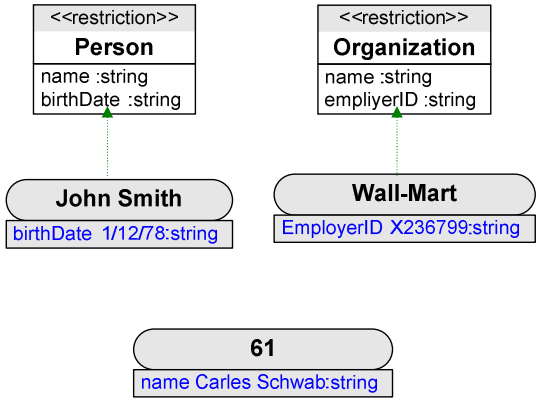
CU-003
02/2006
Module 6

# Deferred categorization

| Assumption | Relational Query Engine | OO Runtime | Reasoner |
|---|---|---|---|
| The row/object/individual is a member of the table/class. | Yes | Yes | No |

| Person | | |
|---|---|---|
| Person ID | Name | Birth Date |
| 345 | John Smith | 1/12/78 |

| Organization | | |
|---|---|---|
| Org ID | Name | Employer ID |
| 13 | Wal-Mart | X236799 |




New party transaction

| Name | Birth Date | Employer ID |
|---|---|---|
| Charles Schwab | ? | ? |

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Part of the Open World Assumption. A statement cannot be assumed to be true on the basis of a failure to prove it.

Several of the assumptions discussed here are a result of this assumption.

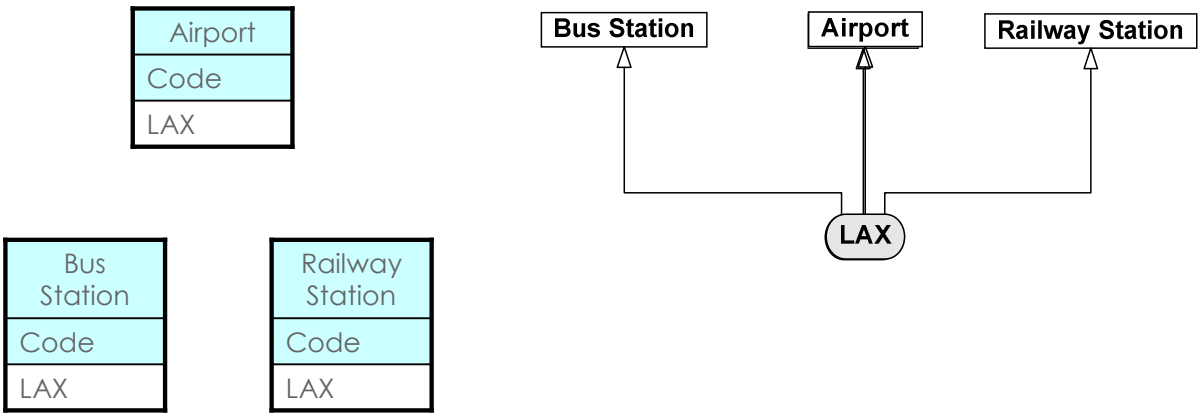
## Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# Multiple categorization

| Assumption | Relational Query Engine | OO Runtime | Reasoner |
|---|---|---|---|
| The row/object/individual is not a member of any other class besides its class/table and its super classes. | Yes | Yes | No |




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Referent centric representation. You start with objects in the real world and add properties as you discover them. Equivalence aside all the information about a referent is associated with that referent. In Relational each aspect of the object is normalized out into separate rows.

Categorize first populate second.

Populate first categorize second.

## Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

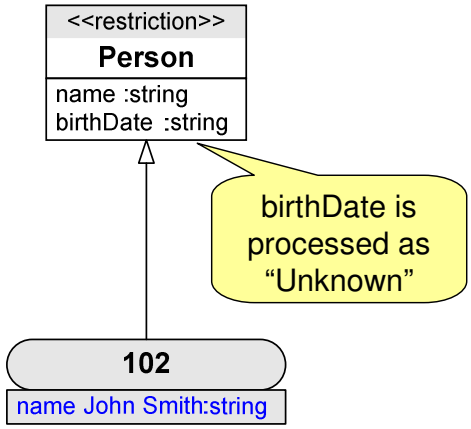
# Knowing what you don't know

| Assumption | Relational Query Engine | OO Runtime | Reasoner |
|---|---|---|---|
| The row/object/individual inherits all the properties of the class/table. | Yes | Yes | No |

| Person | | |
|---|---|---|
| Person ID | Name | Birth Date |
| 345 | John Smith | NULL |

Process as "Not Applicable" or "Unknown" ?

someValuesOf(name)
someValuesOf(birthDate)




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

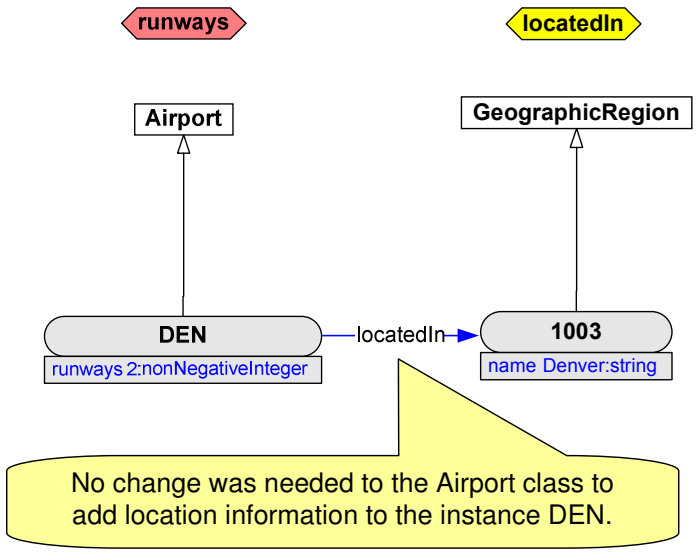
# Domain Extensibility

| Assumption | Relational Query Engine | OO Runtime | Reasoner |
|---|---|---|---|
| The row/object/individual has no other properties other than those inherited from its class/table. | Yes | Yes | No |

| Airport | |
|---|---|
| Code | Runways |
| DEN | 2 |

| Airport | | |
|---|---|---|
| Code | Runways | Geo-RegionID |
| DEN | 2 | 1003 |

| GeographicRegion | |
|---|---|
| Geo-RegionID | Name |
| 1003 | Denver |




11 CU-003: OWL Modeling with RDF. © 2006 Cerebra, Semantic Arts, All Rights Reserved.

There are really three ways to look at this one (1) I have an instance of a class with an unknown property relationship and I assert it. – We already covered this one under knowing "Knowing What You Don't Know'

(2) You have an existing property defined in the domain but not associated with any class or categorization. Never the less you use this property describe an individual. – This gives you the flexibility to collect information and deciding how you are going to use it later without making any design change.

(3) Lastly there is the possibility of adding an entirely new property to the domain and using it to describe what an individual. This is in effect what happens on import and what is shown in the example.

The fact that you don't have to change schema to add relationships also helps in marinating the stability. The more stable the schema the less impact on other other users who are dependant on it.

This is a relatively small improvement but it may be significant.

STML to HTML moved web site development away form the specialist.

This may move data base movement away from the DBA. Allows reuse of information directly by and end user without copying it.

Compare to Access

## Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# What gets assumed ?

| Assumption | Relational Query Engine | OO Runtime | Reasoner |
|---|---|---|---|
| The row/object/individual exists in the domain. | Yes | Yes | Yes |
| The row/object/individual is distinct from any other row/object/individual in the domain. | Yes | Yes | No |
| The row/object/individual is a member of the table/class. | Yes | Yes | No |
| The row/object/individual is not a member of any other class besides class/table and its super classes. | Yes | Yes | No |
| The row/object/individual inherits all the properties of the class/table. | Yes | Yes | No |
| The row/object/individual has no other properties other than the those inherited from its class/table. | Yes | Yes | No |

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# You Assume Too Much

- In general, OWL/RDF makes fewer assumptions about the knowledge being represented.
- In each technology, the assumption (or lack of it) is sometimes helpful and sometimes unhelpful.
- You could do a similar comparison for other constructs such as creating a class property or relationship in OWL/RDF versus relational/OO.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

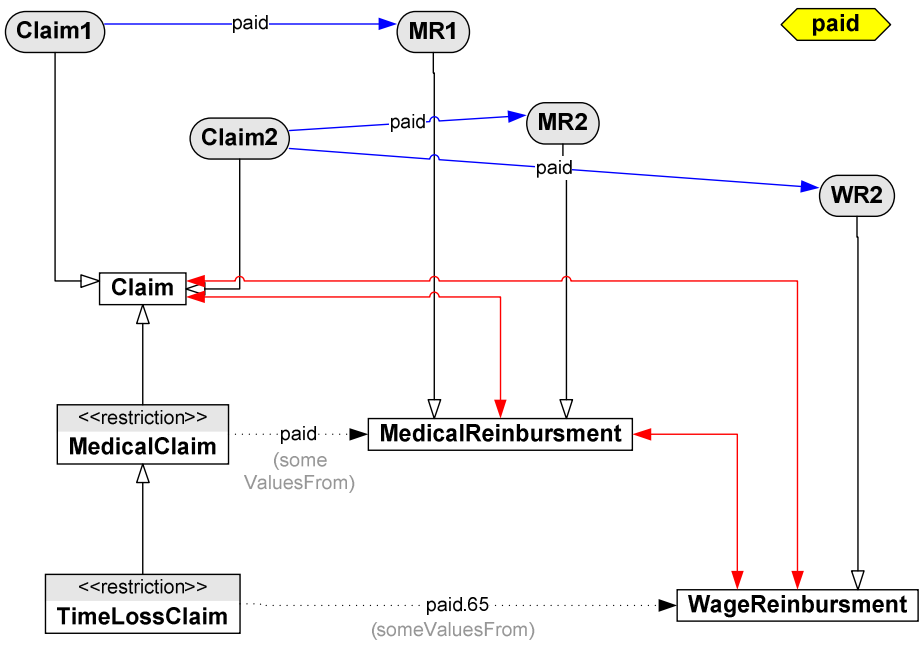
Student Notes:

# Insurance Claim Exercise

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Exercise 1 - Claims Ontology

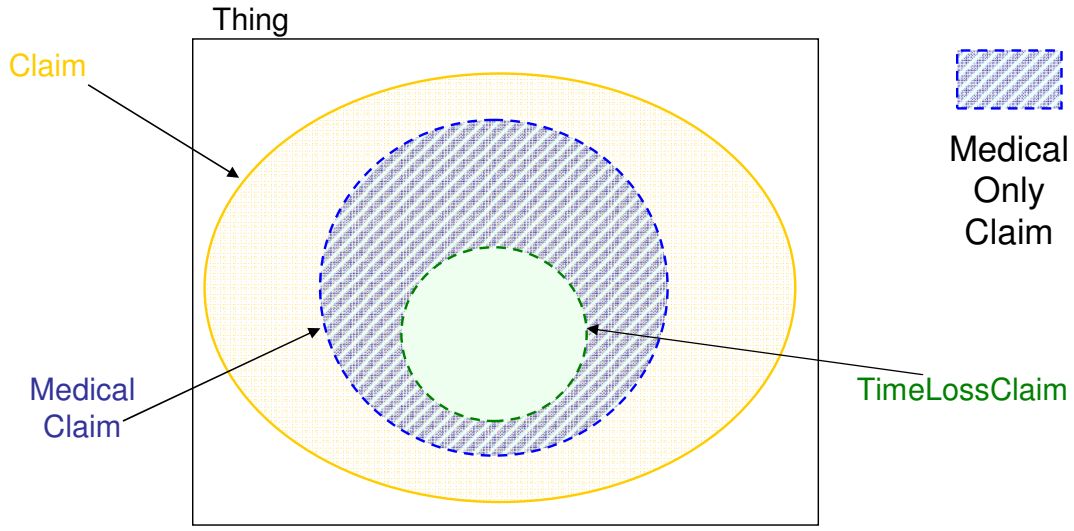



 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Exercise 1

- Load the ontology `Module6_Exercise`"
- Query the ontology for Claims that are Medical Only, e.g., a MedicalClaim that is not a TimeLossClaim.




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Rat Holes



- If you are anything like us, the idea that this simple seeming query is not doable in OWL is hard to accept.
- We called it the "rat hole" problem because, when faced with this situation, there was a constant temptation to keep trying to solve it using some cunning combination of complements and property restrictions.
- To try and save you the time, here is a review of the main rat holes we have repeatedly been down.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# Rat Holes



- Make an additional restriction class MedicalOnlyClaim with an allValuesFrom restriction on paid MedicalReinbursement.
  - Remember you cannot infer subjects from their objects using allValuesFrom. Any Claim could always potentially have an unasserted non medical payment.
- Make MedicalReimbursement an allValuesFrom restriction class on MedicalClaim.
  - This is a legitimate assertion and does effectively infer membership of Claim1 and Claim2 in MedicalClaim, but it does not prevent those claims having or potentially having a WageReinbursment payment.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# Rat Holes



- Query on the intersection of Medical Claims and the complement Time loss claims

```
for $i in instances()
where every $m in models() satisfies
        member($i, MedicalClaim,$m) and
        not( member($i, TimeLossClaim,$m))
return $i
```

  - Nice try but *not( member($i, TimeLossClaim,$m))* does not give you all the individuals you have left after eliminating all provable members of TimeLossClaim, rather, it gives you all individuals that are *provably* not a TimeLossClaim. There are no such claims as all claims could potentially have a paid Wage reimbursement.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# Rat Holes



- Create the complement of TimeLossClaim, then query on the intersection of NotATimeLossClaim and MedicalClaim.

```
for $i in instances()
where every $m in models() satisfies
        member($i, MedicalClaim,$m) and
        member($i,NotATimeLossClaim,$m)
return $i
```

  - Same basic problem as the previous rat hole. The restriction class NotATimeLossClaim has no provable members as all claims could potentially have a paid WageReinbursment. The intersection of the two conditions returns an empty set.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Rat Holes



- Run two queries: provable MedicalClaims and provable TimeLossClaims, and post-process them to get the first result set minus the second.
  - This is as close as you can get to an answer to the problem. In practice what you are doing is using closed world reasoning for the post-processing.
- Alternately, run two queries: provable MedicalClaims and satisfiable NotATimeLossClaims, and post-process them to get the intersection of the two result sets.
  - This achieves the same result as above.

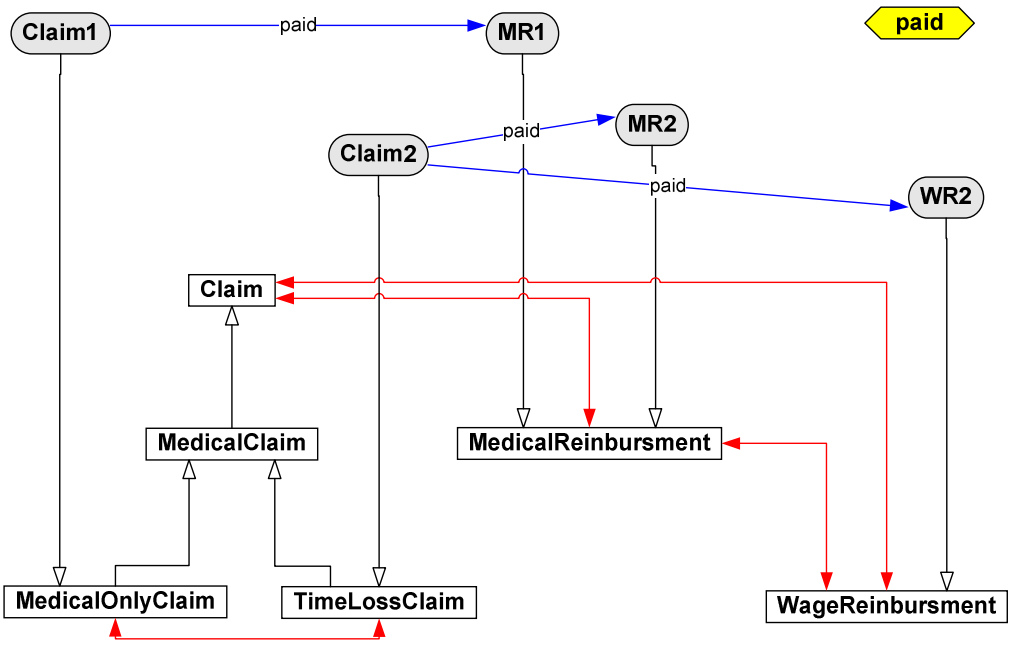
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# Declared Medical-Only Claims




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Limitations Of OWL

How to recognize what can't be represented in OWL and what to do about it.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# The Only Child

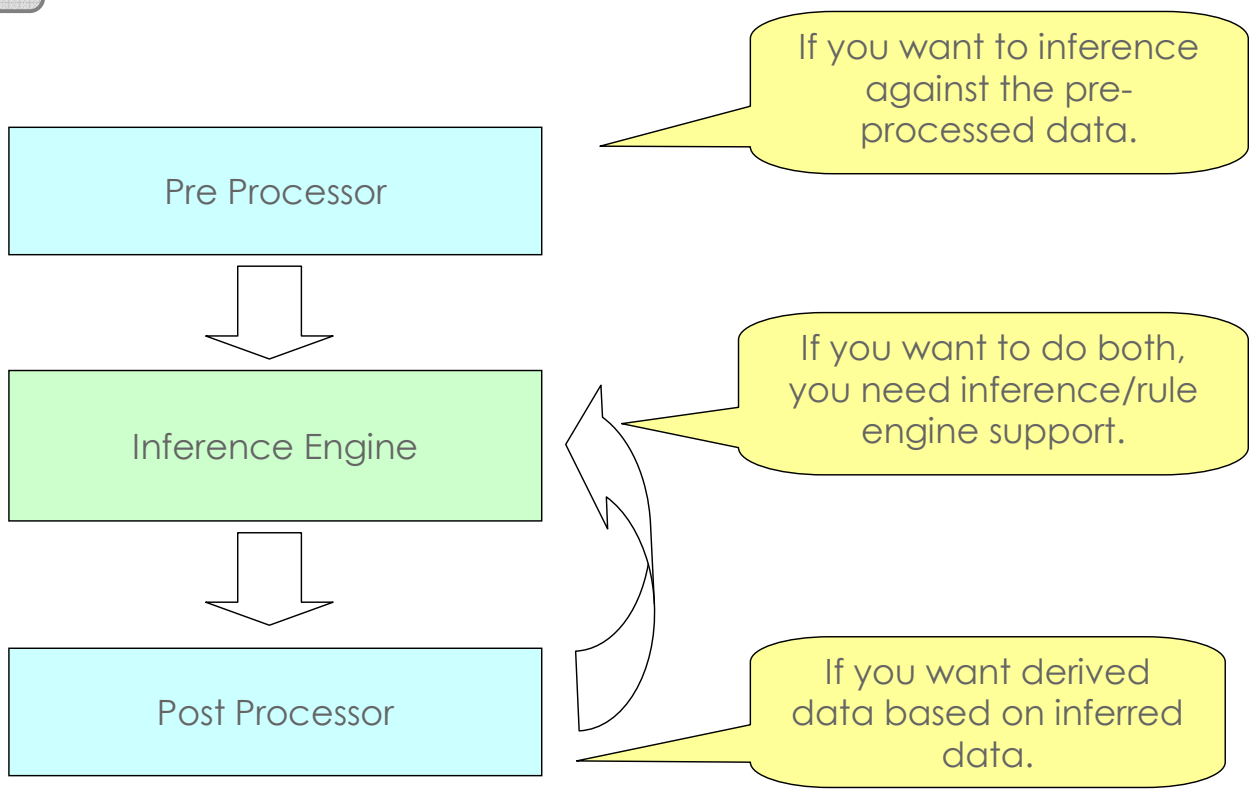
**You cannot Infer Membership based on the Absence of a Property Relationship.**

- Linguistic Trigger
  - Often an issue if you are trying to create a category described in terms of:
    - "Only" some property relationship
    - "Less than" some number of relationships
    - "Exactly" some number of relationships
- Examples
  - Only child
  - Medical-only claim
  - An inactive customer has fewer than 2 orders
  - Three cheese topping pizza
- Alias
  - The Only Child problem
  - "Absence of evidence is no evidence of absence."

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# To Pre or Post Process?




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# The Only Child

- Pre-processing workaround
  - Categorize "only children" prior to loading the ontology and declare membership of the OnlyChild class based on that.
- Post-processing workaround
  - Assuming that the user "knows" that the ontology is complete, you can run one or more satisfiable or provable queries and derive a result.
- Future OWL features
  - This limitation is pretty much inherent in an open world approach. However, some theoreticians have postulated a "known" query function which returns individuals based on what is currently asserted in the ontology rather than what is provable or satisfiable.
  - Potentially, vendors could implement some proprietary version of this.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# The Uncle Problem

**You Cannot Infer The Existence of a Relationship Based on the Existence of Two or More other Relationships.**

- Linguistic Trigger
  - You find yourself describing the property in terms of a rule between properties, e.g., "if this property relationship and/or that property relationship then this property/relationship is true."
- Examples
  - If one person is a brother of a second person and the second person is a father to a third person, then the first person is the uncle of the third person.
  - If a flight "departsFrom" an airport and the Airport is "located In" a city then the flight "departsFrom" a city.
- Alias
  - Property composition

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# The Uncle Problem

- Pre-processing workaround
  - Derive the "Uncle" relationship prior to loading the ontology. Of course, if you do this, then the "hasParent" and "hasBrother" relationship cannot have been inferred. On the other hand, the "hasUncle" relationship can be used as a basis for inference.
- Post-processing workaround
  - Derive the uncle relationship from the results of queries on the "hasParent" and "hasBrother" relationships. This would allow these relationships to have been inferred.
- Future OWL features
  - There are several competing proposals currently under consideration for a semantic web rules language to be used in conjunction with OWL.
  - e.g., SWRL, a combination of OWL and RuleML

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# The Uncle Problem: SWRL Abstract Syntax

hasParent(?x1,?x2) & hasBrother(?x2,?x3) => hasUncle(?x1,?x3)

If it is true that
Individual x1 hasParent Individual x2 And
Individual x2 hasBrother Individual x3
this implies that
Individual x1 hasUncle Individual x3

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Antecedent – Body

Consequent - Head

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# The Uncle Problem: SWRL XML Concrete Syntax

```
<ruleml:imp>
 <ruleml:_rlab ruleml:href="#TheUncleRule"/>

<ruleml:body>
   <swrlx:individualPropertyAtom  swrlx:property="hasParent">
    <ruleml:var>x1</ruleml:var>
    <ruleml:var>x2</ruleml:var>
   </swrlx:individualPropertyAtom>
   <swrlx:individualPropertyAtom  swrlx:property="hasBrother">
    <ruleml:var>x2</ruleml:var>
    <ruleml:var>x3</ruleml:var>
   </swrlx:individualPropertyAtom>
 </ruleml:_body>

 <ruleml:_head>
  <swrlx:individualPropertyAtom  swrlx:property="hasUncle">
   <ruleml:var>x1</ruleml:var>
   <ruleml:var>x3</ruleml:var>
  </swrlx:individualPropertyAtom>
 </ruleml:_head>

</ruleml:imp>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Note that, at least in SWRL you are limited to implying

An individual has some object relationship or property relationship

An individual is a member of a class.

You cannot create new individuals.

SWRL is a way of asserting rules in terms of concepts described in an OWL ontology.

## Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# The International Flight Problem

**You Cannot Create a Restriction by Comparing Two Property Relationships as if they were Variables.**

- Linguistic Trigger
  - You find yourself wanting to describe the conditions for a class in terms of a "join" between two properties (e.g., if property A equals property B), as opposed to a comparison to a specific property value (e.g., if property A equals value X).
- Example
  - International Flight – Flights where the departure country is different form the arrival country.
  - Training Flight – Flights where the departure airport is the same as the arrival airport.
- Alias
  - The join problem

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# The International Flight Problem

- Pre-processing workaround
  - You could perform the join prior to loading the ontology and explicitly declare membership of the class.
- Post-processing workaround
  - You can perform the comparison on the results of the ontology query.
- Future OWL features and extensions
  - The semantic web rules language is likely to allow these sorts of constraints to be expressed.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# The International Flight Problem: SWRL Abstract Syntax

departsCountry(?f1,?c1) &

arrivesCountry(?f1,?c2) &

differentFrom($c1, $c2)

=> InternationalFlight(?f1)

If it is true that
Individual f1 departsCountry Individual c1 And
Individual f1 arrivesCountry Individual c2 And
Individual c1 is different from Individual c2
this implies that
individual f1 is an instance of InternationalFlight

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

HORN rules
Antecedent – Body
Consequent - Head

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# Not A Sibling

You Cannot Assert the Absence of a Property Relationship.

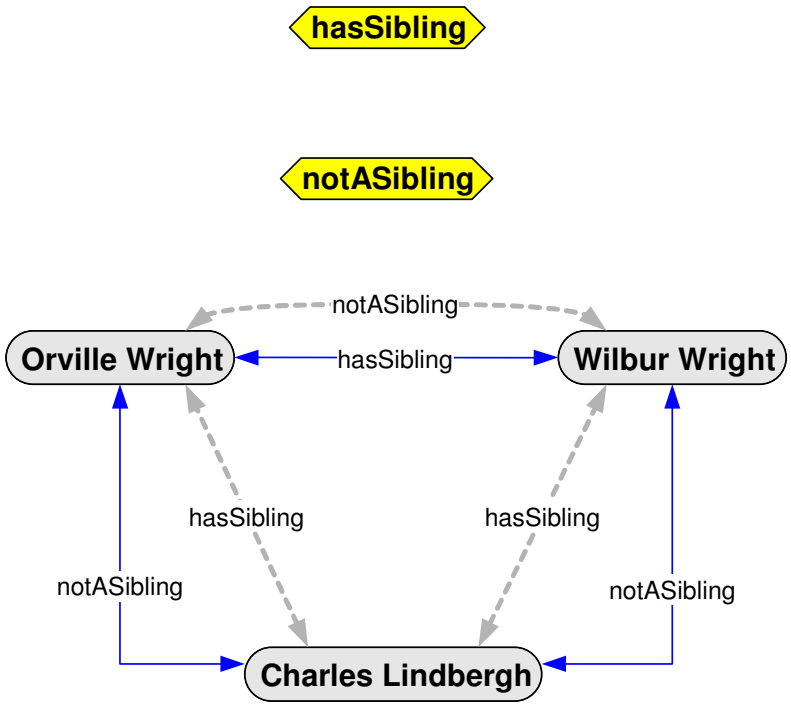
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# Not A Sibling




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

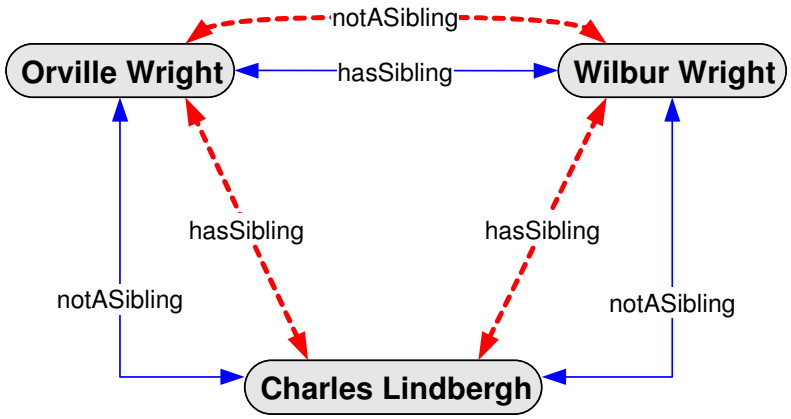
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# Not A Sibling




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# Not A Sibling

**You Cannot Assert the Absence of a Property Relationship.**

- Linguistic Trigger
  - You find yourself wanting to describe an individual as not having a specific relationship, e.g., "is not his brother."
- Example
  - "not a sibling"
  - "does not border on"
- Alias
  - The disjoint property problem

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# Not A Sibling

- Pre-processing workaround
  - You could use pre-processing to check for situations where two individuals are simultaneously a sibling and not a sibling.
  - However, this will not overcome the fundamental problem that a query of satisfiable siblings will include those individuals who have been asserted not to be siblings.
- Post-processing workaround
  - You could post process your satisfiable hasSiblings query result to exclude provable notASibling results.
- Future OWL features
  - OWL 1.1 is in the very early stages of approval by the W3C. It includes a proposal for a disjoint property construct.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

You can never actual assert the non existence of a property but you can assert something else which will cause that non existence to be inferred.

There are other OWL workarounds. E.g. A restriction class for each individual with siblings restricts the cardinality of those siblings.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# Classification by mathematical function

## You Cannot Classify Individuals Based on a Mathematical Function

- Linguistic Trigger
  - You find yourself wanting to describe a class using terms like total, sum, average or some other mathematical term. However, you can determine if an integer data value is within a static range using custom data types and the Cerebra Server extensions.
- Example
  - A LargePackage is one where the length + height + depth in inches is greater than 100
- Alias
  - The calculation problem
  - Procedural attachments

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# Classification by mathematical function

- Pre-processing workaround
  - Make your calculation prior to loading the ontology and add it explicitly as another datatype property relationship.
- Post-processing workaround
  - You could, of course, post process a calculation but you won't be able to reason on the result.
- Future OWL features and extensions
  - Cerebra supports a proprietary extension to OWL that allows some classification based on custom data type ranges, e.g., a young driver is one whose age is under 25.
  - OWL 1.1 may include support for custom data types similar to the Cerebra extensions.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Actual calculations is not likely to happen any time soon in OWL.

Neither is it a part of SWIRL

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# Classification by date

## You Cannot Classify a Class Based on Temporal Criteria

- Linguistic Trigger
  - You find your self wanting to describe a class using temporal terms like "occurred within time period."
- Example
  - Flights that arrive in the morning
  - Conflicting appointment
- Alias
  - The temporal reasoning problem

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# Classification by date

- Pre-processing workaround
  - You could translate the date/time into a number /representation (a.k.a. Excel) and use custom datatype extensions such as Cerebra's to reason against them.
- Post-processing workaround
  - Post-process date information returned by a query.
- Future OWL features and extensions
  - OWL 1.1 support for custom data types is likely to extend only to "well behaved" XML Schema Datatypes.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# Classification By Sequence

## You Cannot Classify Individuals based on their Sequence within the Ontology

- As with rows in a relational table, the individuals and concepts in an OWL/RDF ontology have no inherent sequence.
- Linguistic Trigger
  - You find yourself wanting to describe a class using sequencing terms like "next," "previous," "last."
- Example
  - First chapter
  - Next task

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Classification by sequence

- Pre-processing workaround
  - If you load a sequence number as part of the data you could restrict on absolute sequence such as "first," "second," "third,"
  - If you chain the individuals together with an object property (e.g., "precedes/succeeds") on load, you can reason against them.
- Post-processing workaround
  - As you cannot predict the sequence of results returned by a query, this is not an option.
- Future OWL features and extensions
  - Abstract high level concepts such as sequence are the subject of upper ontologies (e.g., SUMO). A reasoner that committed to parts of such an ontology could potentially support sequence-based inferencing, however, that sequence information would still have to be explicitly asserted.

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

There is rdf:Seq is an ordered container. However it is not clear that the reasoner will

The rdf:List construct implements what appears to be a linked list but again it is not clear that the reasoner does anything with this.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 6

# Module Review

- Instructor led review session
- Pop-quiz is optional!

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# OWL-DL Modeling with RDF

## Module Ten: Wrap Up

CU-003: OWL Modeling with RDF. © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Agenda

| | Monday, Feb 27 | Tuesday, Feb 28 | Wednesday, Mar 01 |
|---|---|---|---|
| 8 am | | | |
| 9 00 | **Module 1:** *Intro to OWL* | **Module 3:** *Equivalence, Disjointness, and Subsumption* | **Module 5:** *Complex Classes* |
| 10 00 | | | |
| 11 00 | **Module 2:** *Individuals, Properties, and Classes* | **Module 3:** *Equivalence, Disjointness, and Subsumption* | **Module 5:** *Complex Classes* |
| 12 pm | | | |
| 1 00 | **Module 2:** *Individuals, Properties, and Classes* | **Module 4:** *More Properties* | **Module 6:** *How OWL is Different* |
| 2 00 | | | |
| 3 00 | **Module 2:** *Individuals, Properties, and Classes* | **Module 4:** *More Properties* | **Module 8:** *Wrap up* |
| 4 00 | | | |

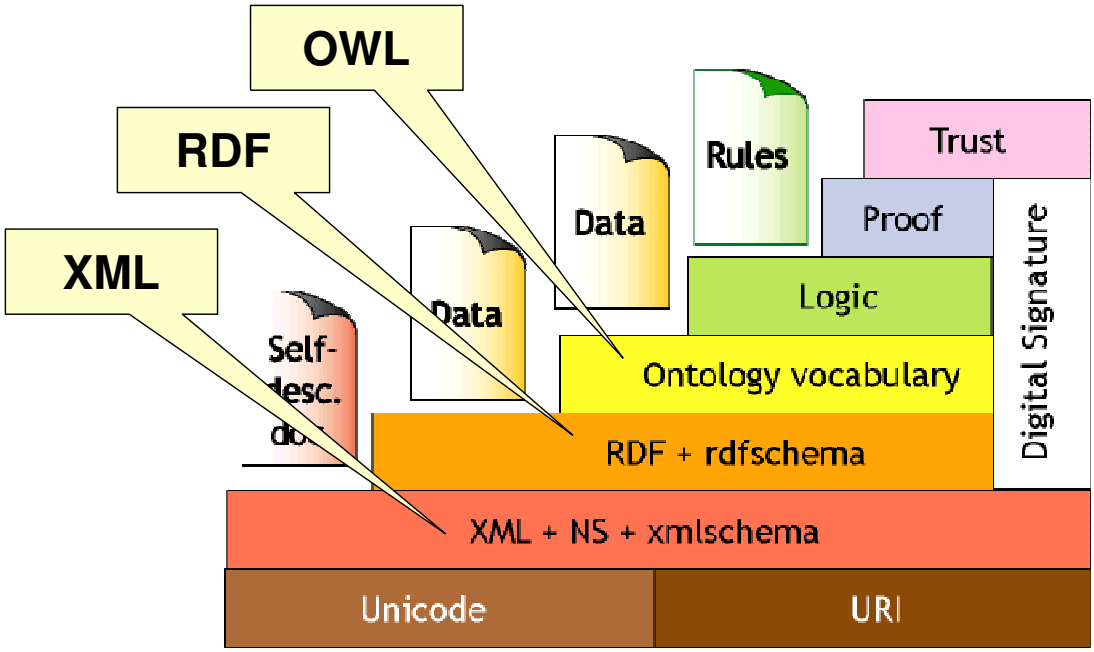
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Semantic Web "Official Stack"



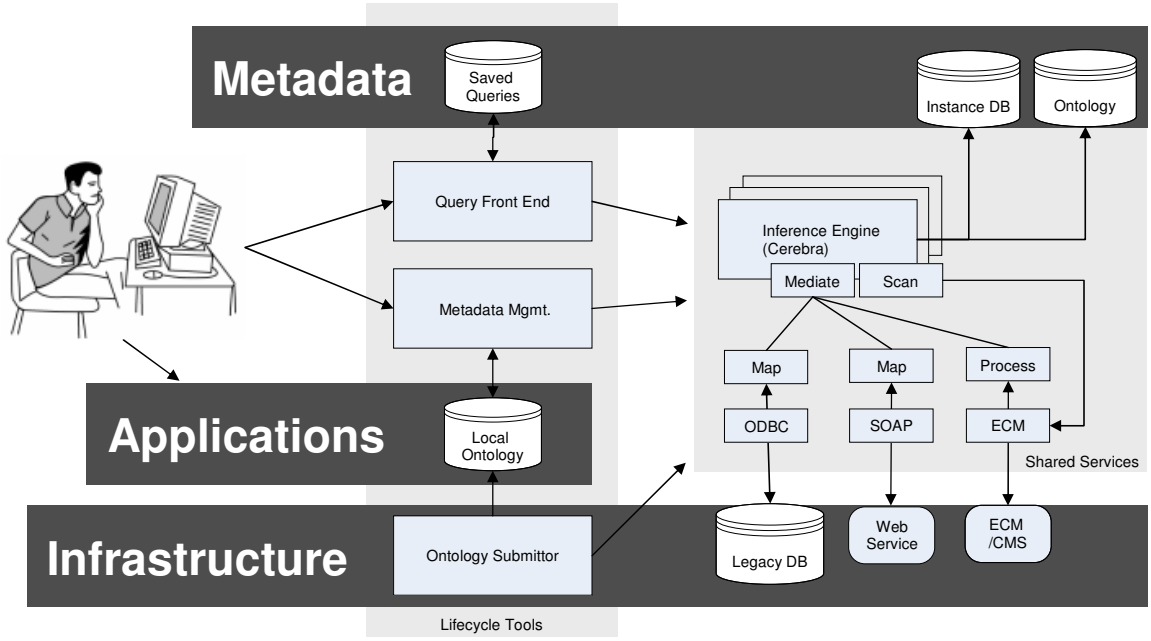

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 1

# Architecturally



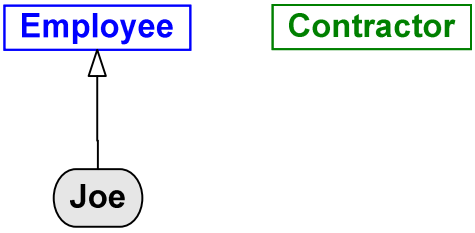

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

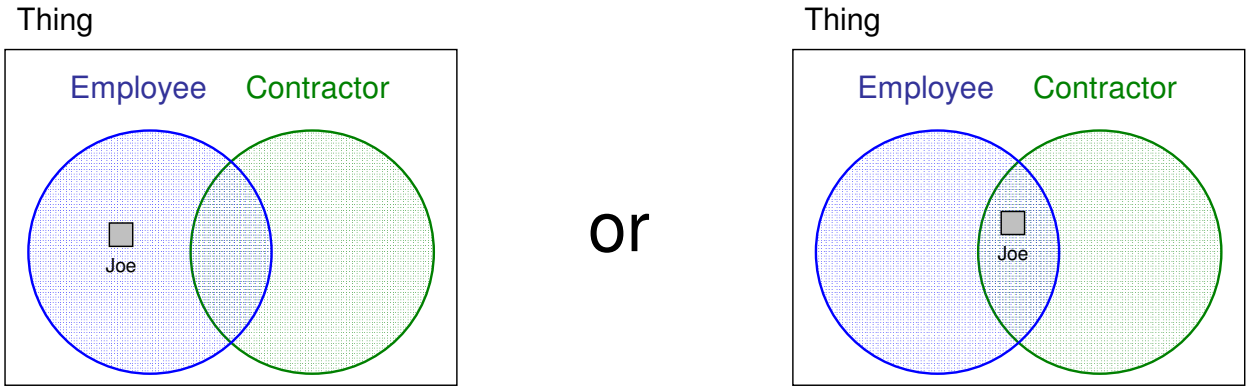
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Membership in an Open World




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

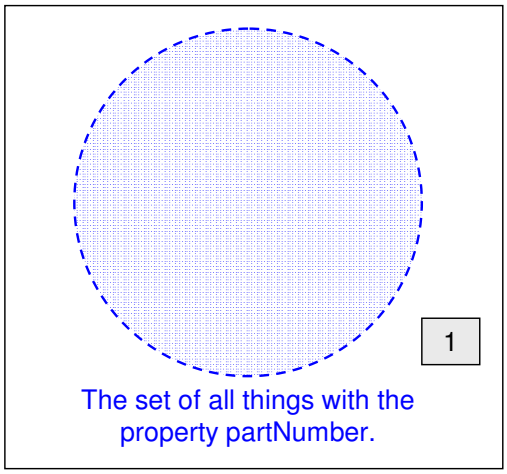
An OWL ontology defines a space of possible interpretations of the domain being modeled. An interpretation is a fixed model of all instances in the ontology, the relationship between those instances, and exactly what classes those members are an instance of. Some facts are true in in EVERY interpretations and some facts are true in in SOME interpretations but not others.

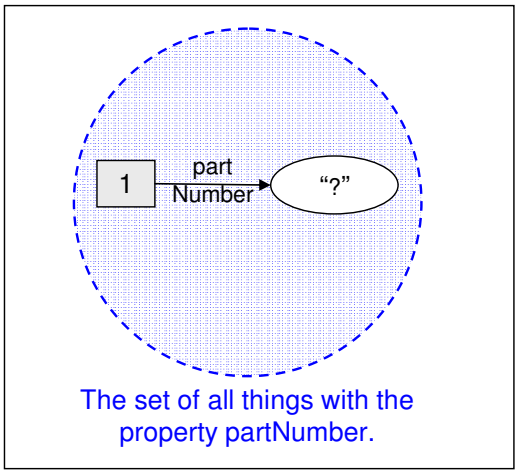
## Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Properties in an open world




**partNumber**

**1**

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

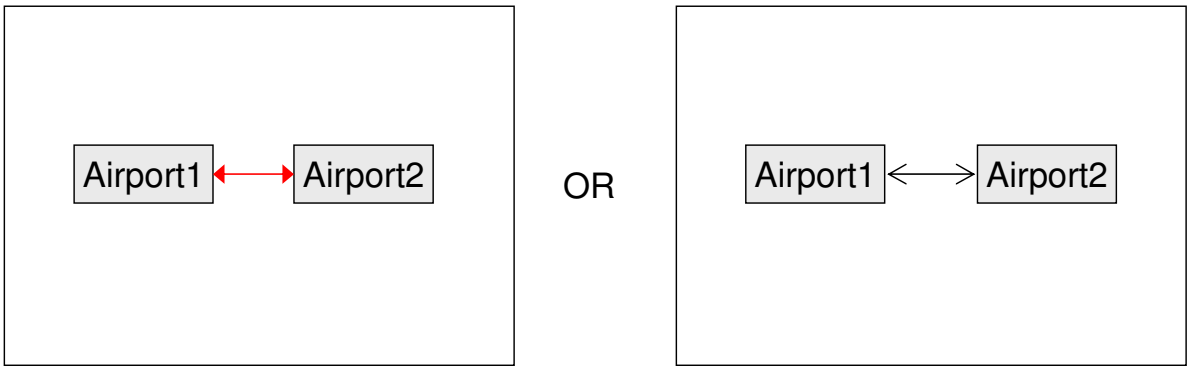
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Individuals In The Open World

Airport1 Airport2




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Provable vs. Satisfiable

- Provable assertions are true in all models.
- Satisfiable assertions are true in some models.

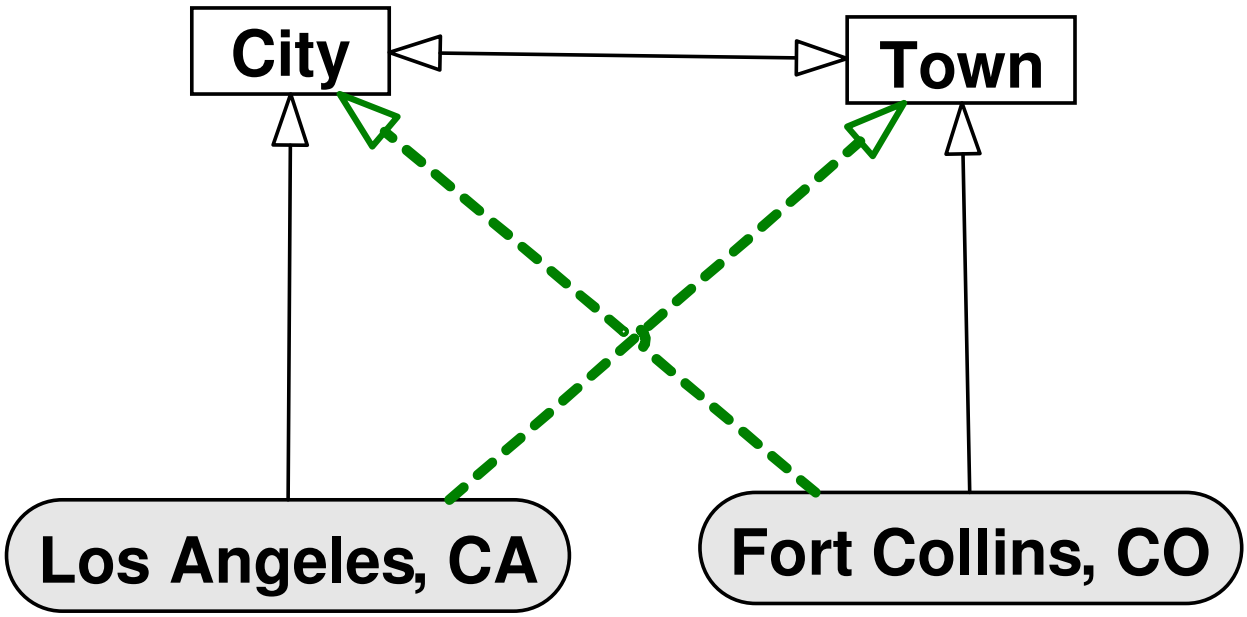
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Equivalent Classes



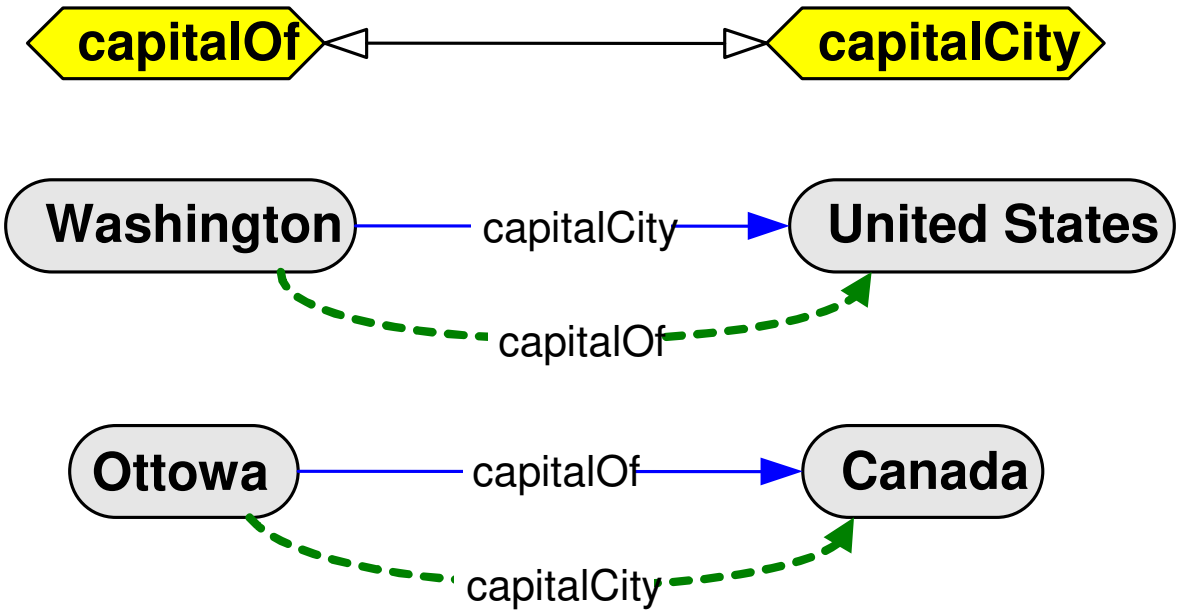

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Equivalent Properties



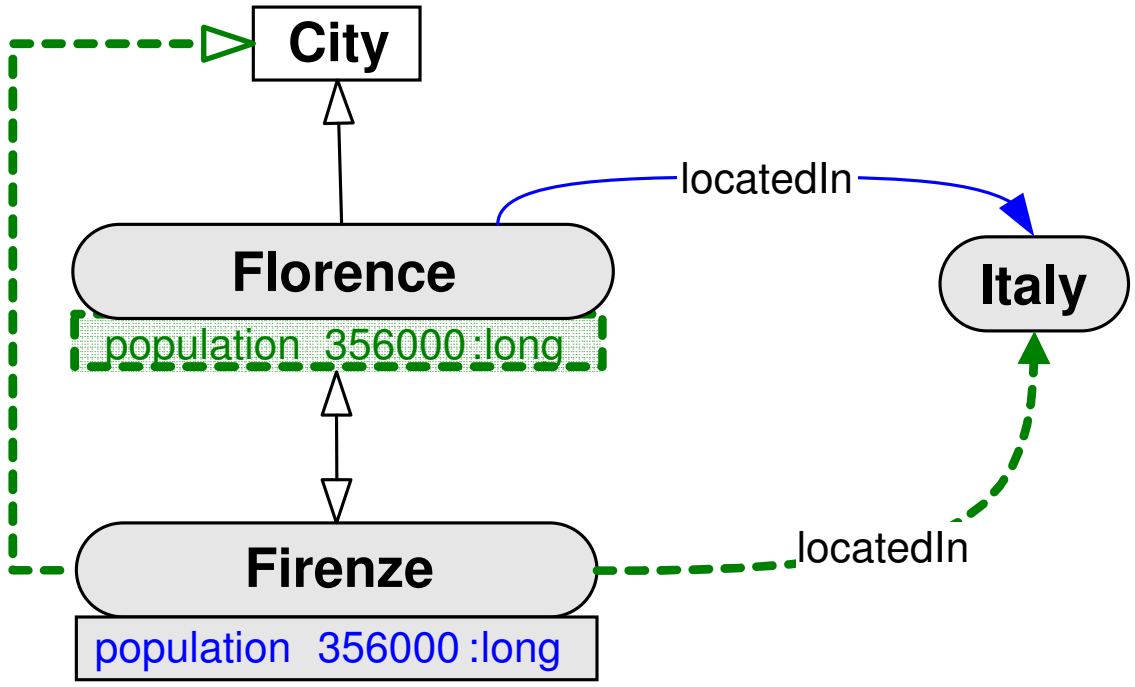

9 CU-003: OWL Modeling with RDF. © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Equivalent Individuals



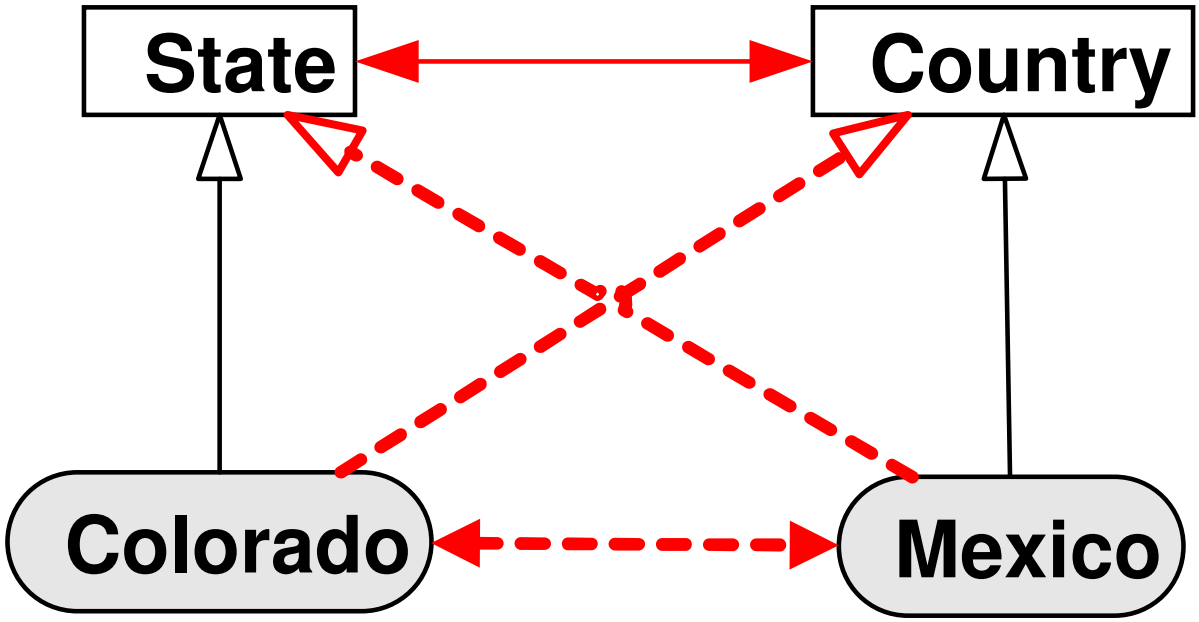

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

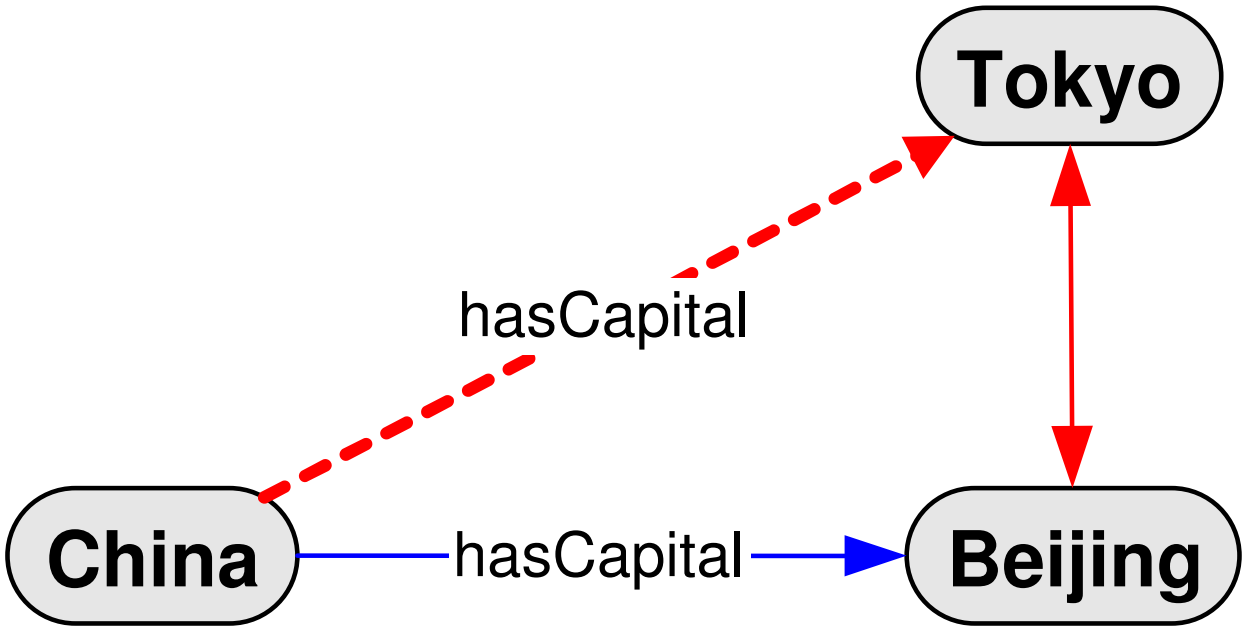
# Disjoint Classes




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Functional Properties



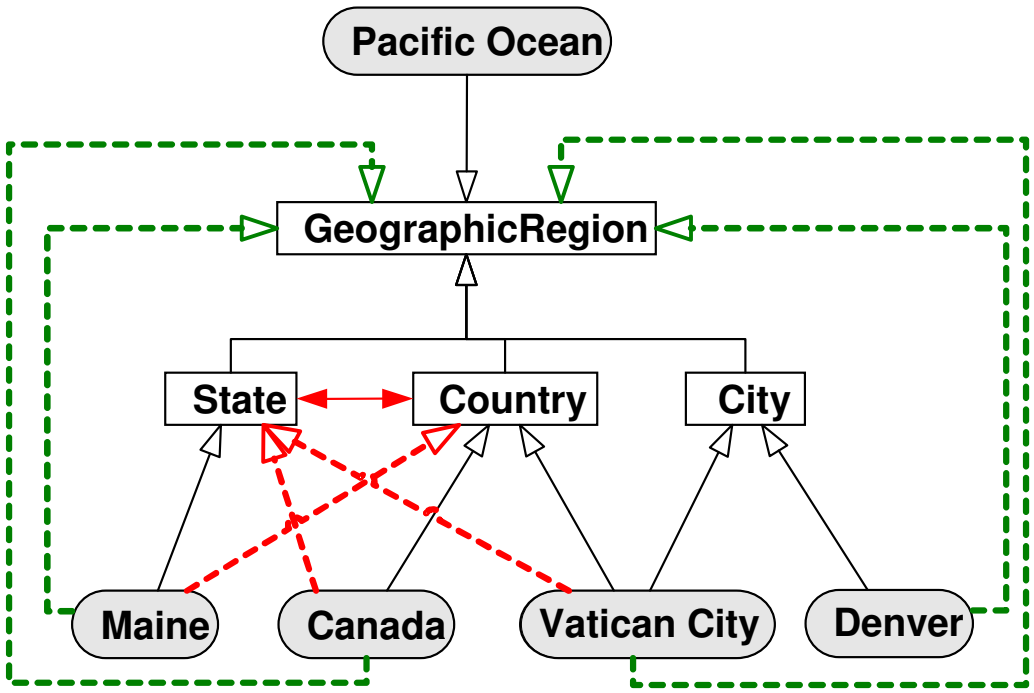

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Subclasses



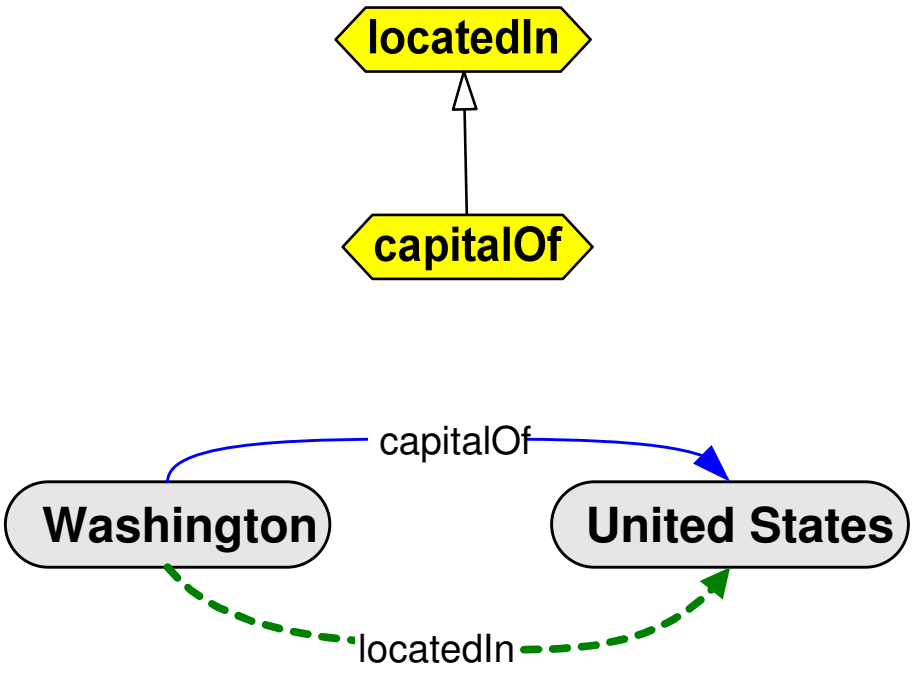

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Subproperties



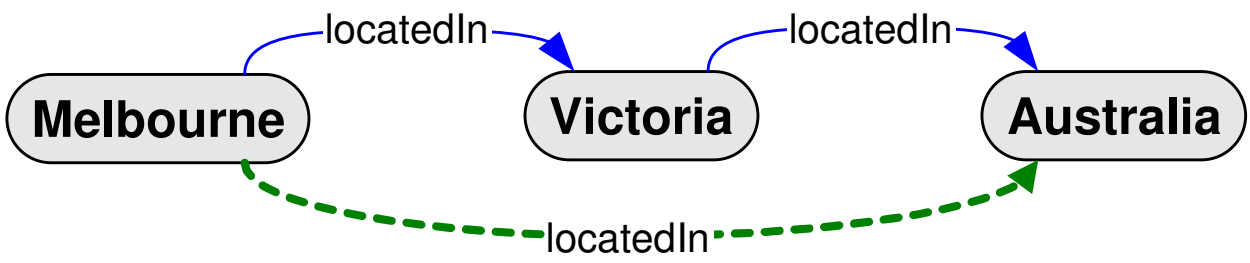

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Transitive Properties

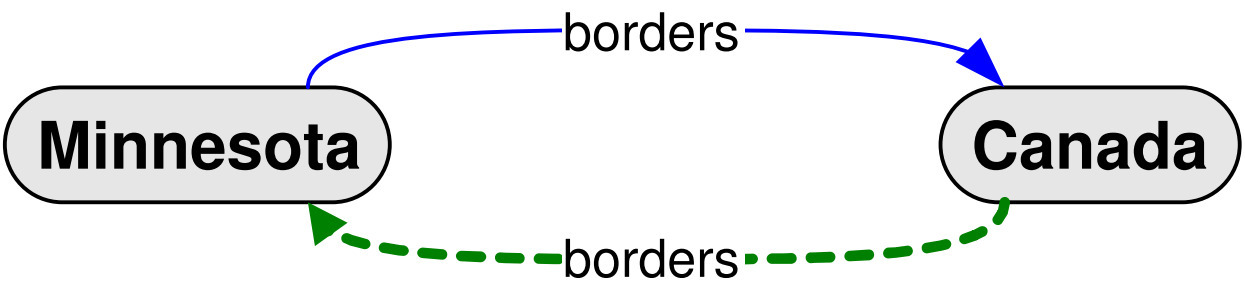



```
<owl:ObjectProperty rdf:ID="locatedIn">
   <rdf:type rdf:resource=
         "http://www.w3.org/2002/07/owl#TransitiveProperty"/>
</owl:ObjectProperty>
```

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Symmetric Properties




```
<owl:ObjectProperty rdf:ID="borders">
   <rdf:type rdf:resource=
         "http://www.w3.org/2002/07/owl#SymmetricProperty"/>
</owl:ObjectProperty>
```

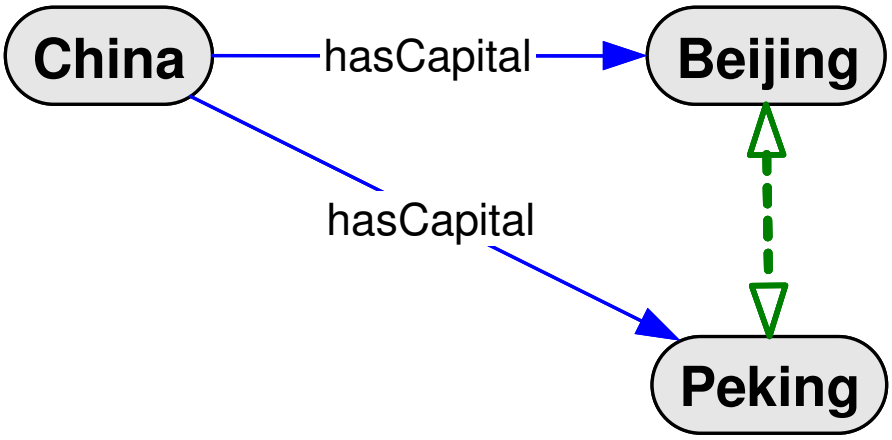
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Functional Properties




```
<owl:ObjectProperty rdf:ID="hasCapital">
   <rdf:type rdf:resource=
         "http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:ObjectProperty>
```

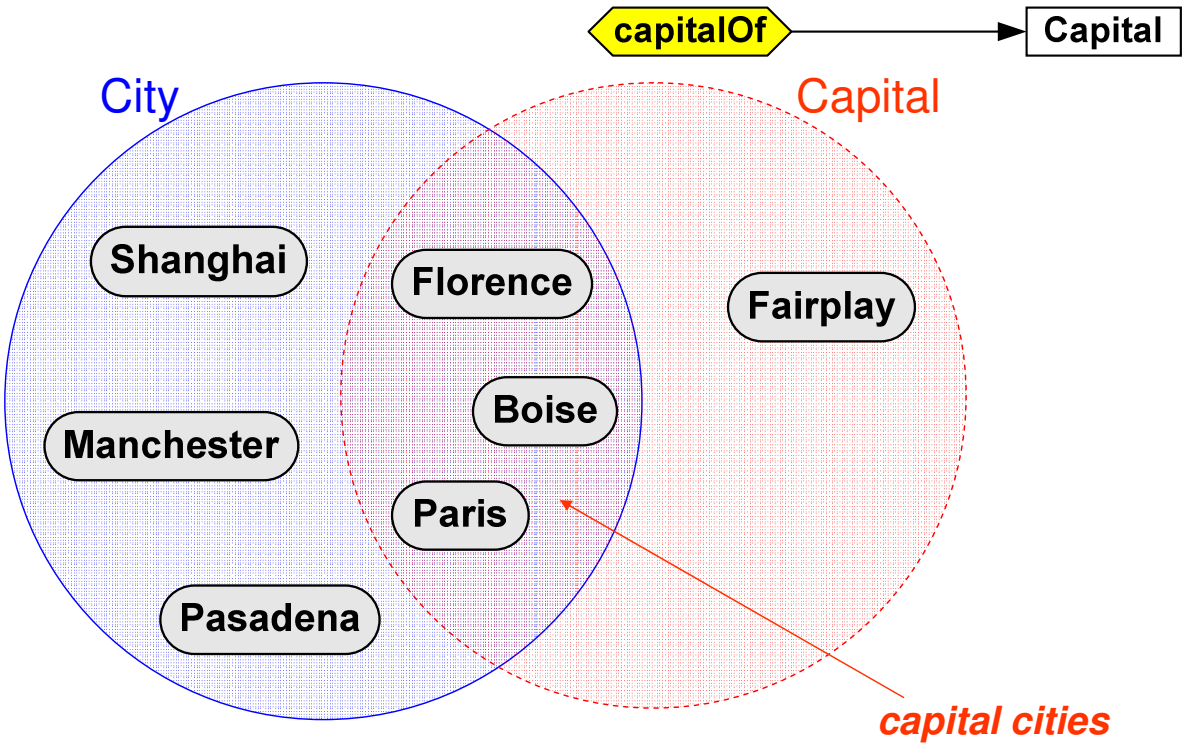
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Domain And Range



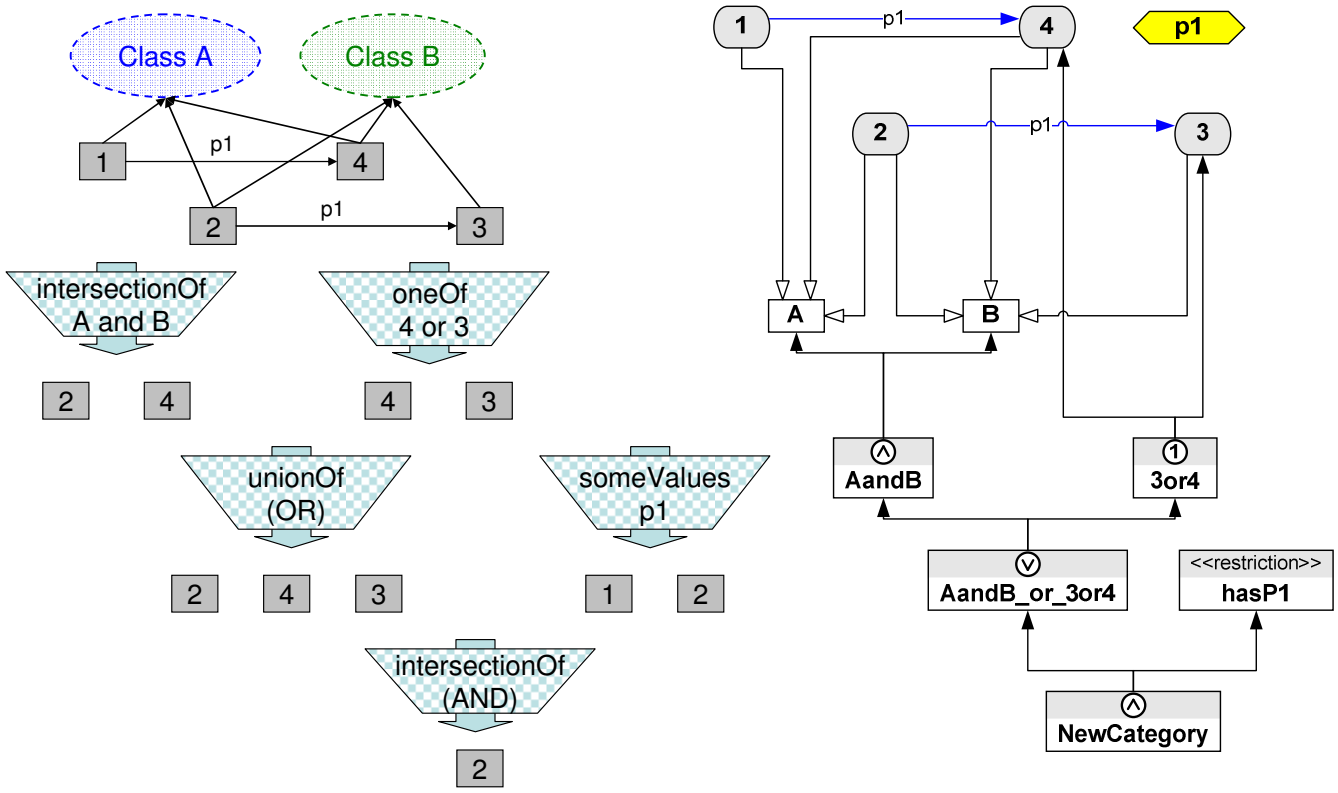

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Complex Classes



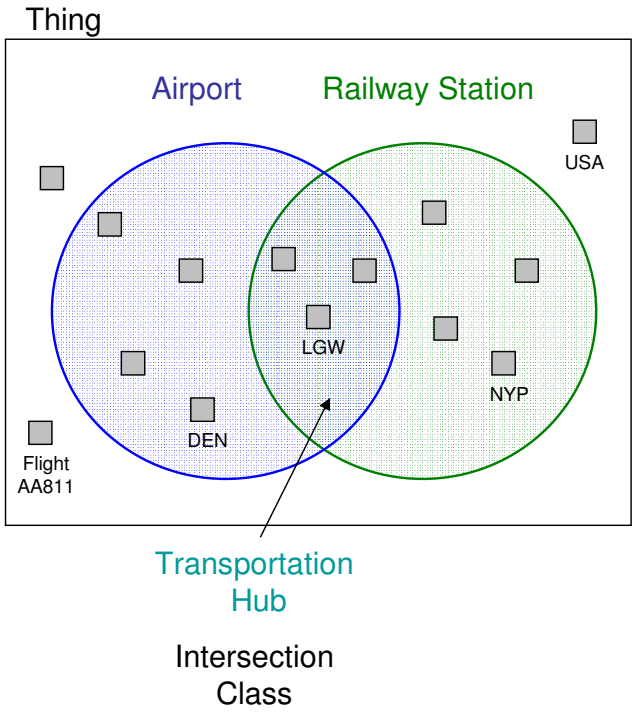

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

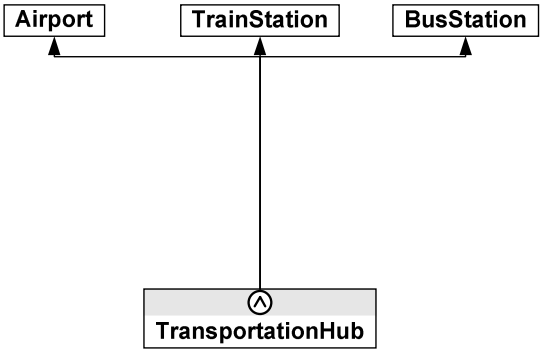
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Intersection




Intersection
Class

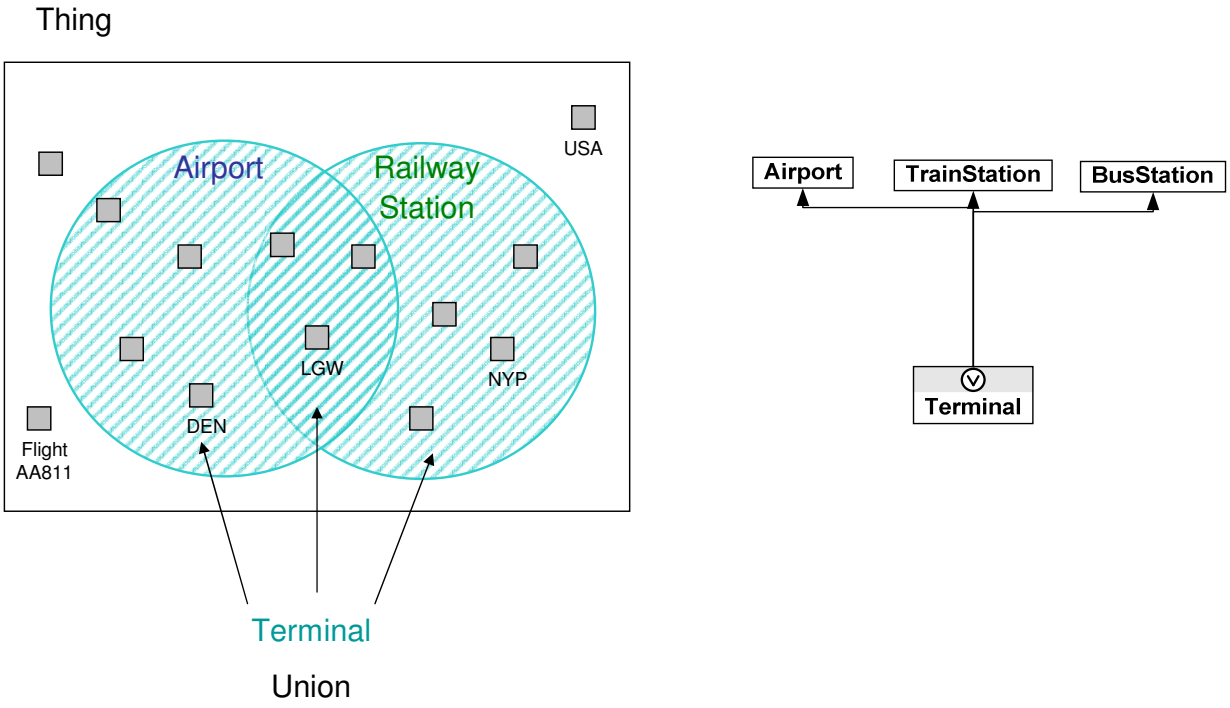
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Union



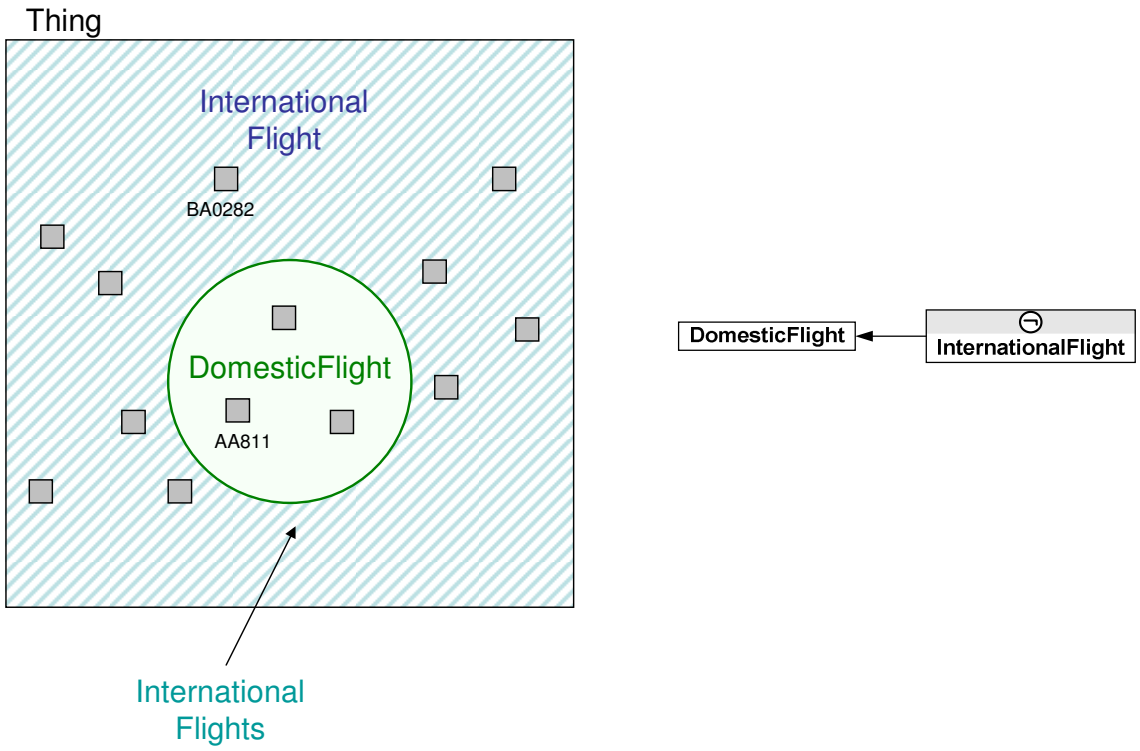

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

## Student Notes:

Cerebra UNIVERSITY

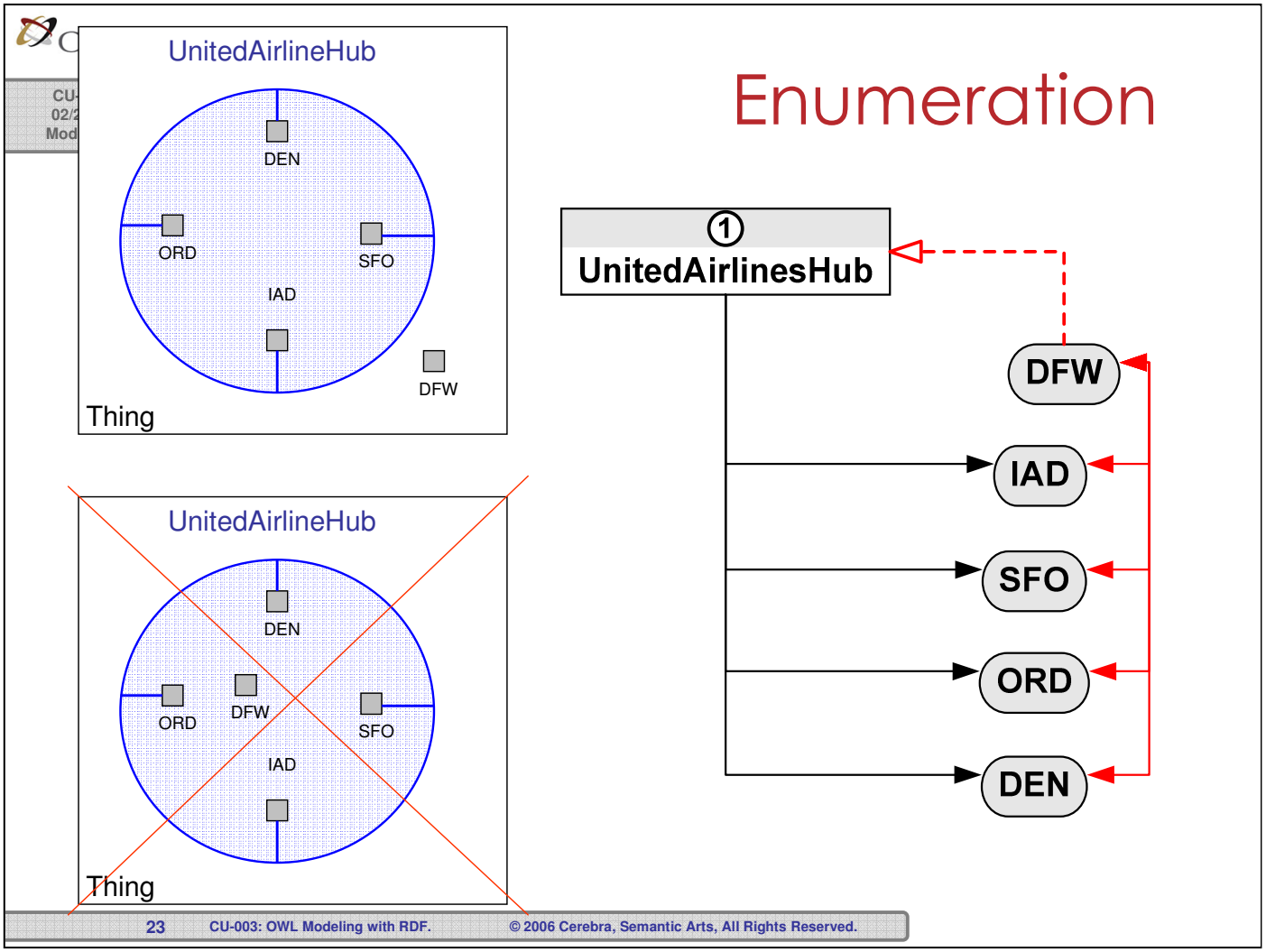
CU-003
02/2006
Module 8

# Complement




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:



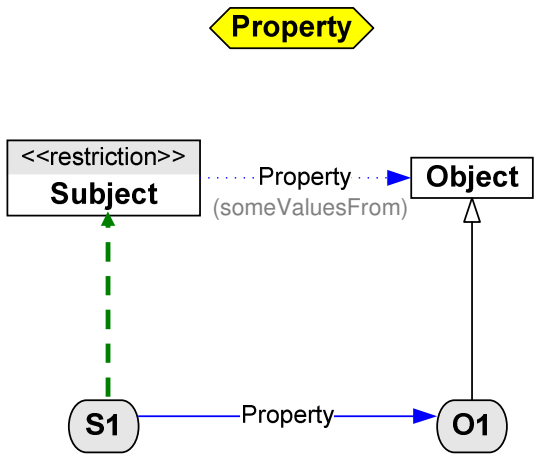

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

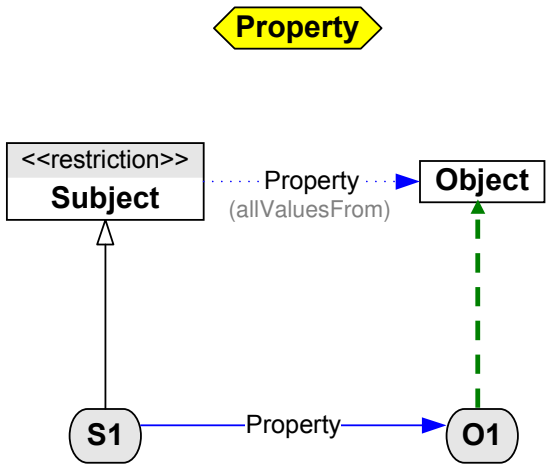
# someValuesFrom vs. allValuesFrom

## someValuesFrom




Infer subject from objects

## allValuesFrom




Infer objects from subject

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

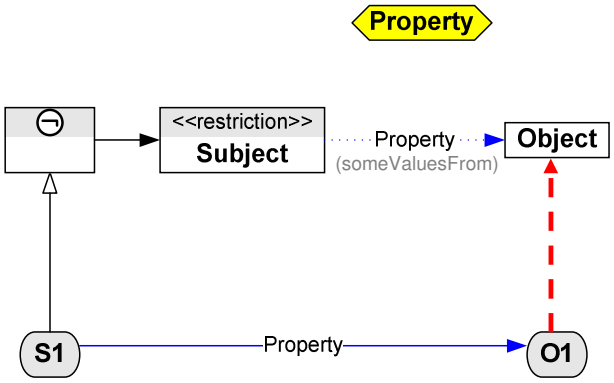
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

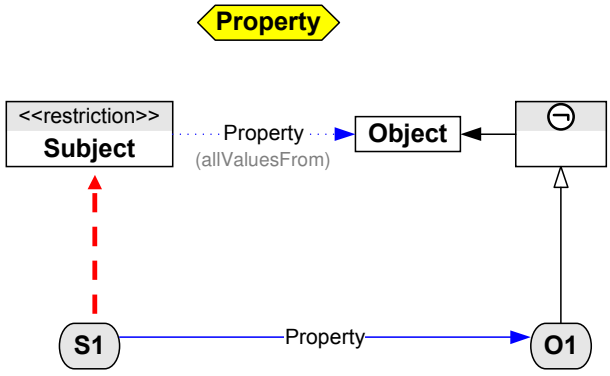
# someValuesFrom vs. allValuesFrom

## someValuesFrom




Infer “not an object” from “not a subject”

## allValuesFrom




Infer “not an subject” from “not an object”

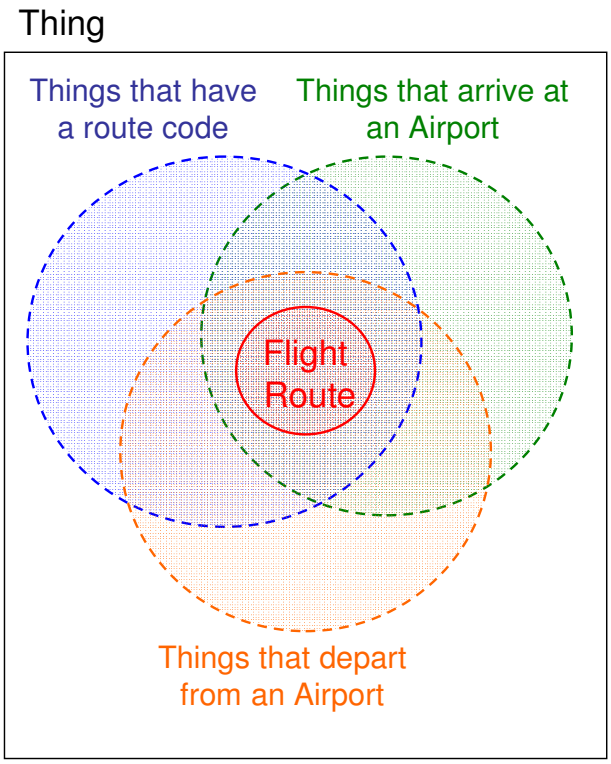
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

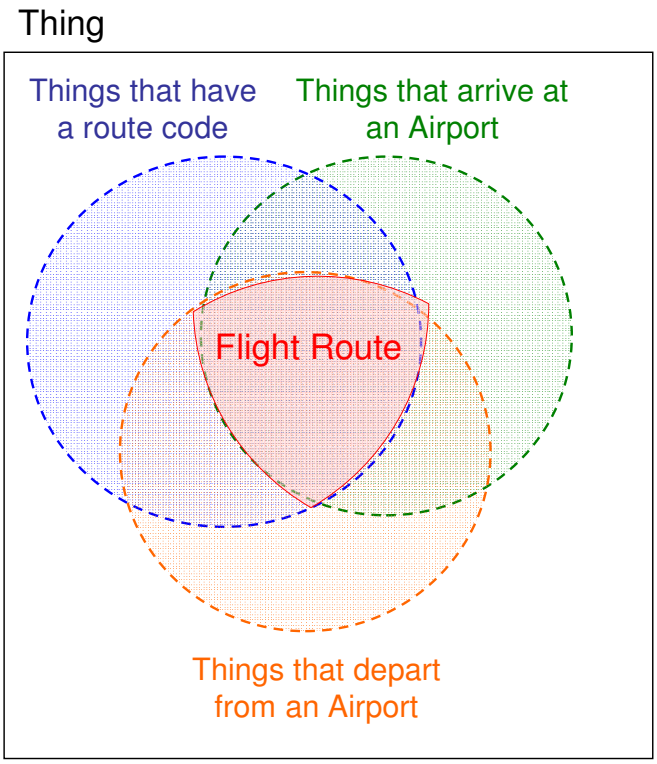
Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Subclass Semantics vs. Equivalence Semantics




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Assumptions

| Assumption | Relational Query Engine | OO Runtime | Reasoner |
|---|---|---|---|
| The row/object/individual exists in the domain. | Yes | Yes | Yes |
| The row/object/individual is distinct from any other row/object/individual in the domain. | Yes | Yes | No |
| The row/object/individual is a member of the table/class. | Yes | Yes | No |
| The row/object/individual is not a member of any other class besides class/table and its super classes. | Yes | Yes | No |
| The row/object/individual inherits all the properties of the class/table. | Yes | Yes | No |
| The row/object/individual has no other properties other than the those inherited from its class/table. | Yes | Yes | No |

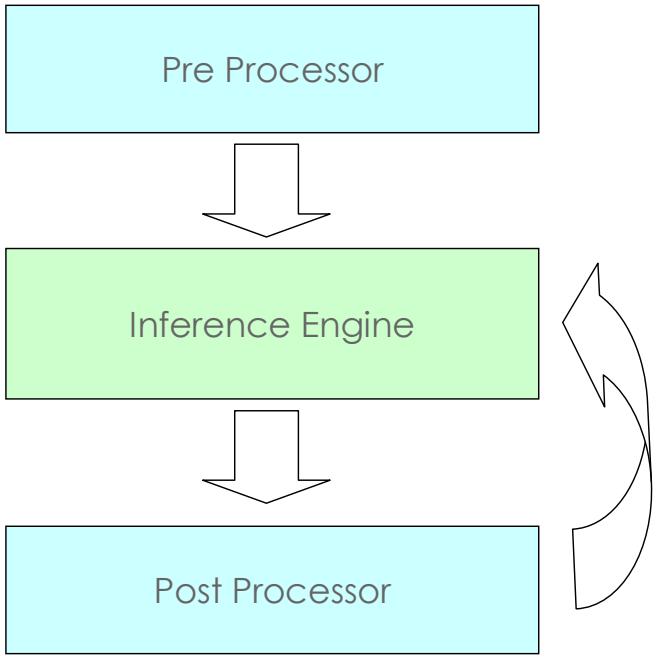
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Limits Of Owl




- The Only Child
- The Uncle Problem
- Not A Sibling
- Classification by mathematical function
- Classification by date
- Classification By Sequence

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Plus…

- Versioning
- Owl Dialects
- Custom Data Types

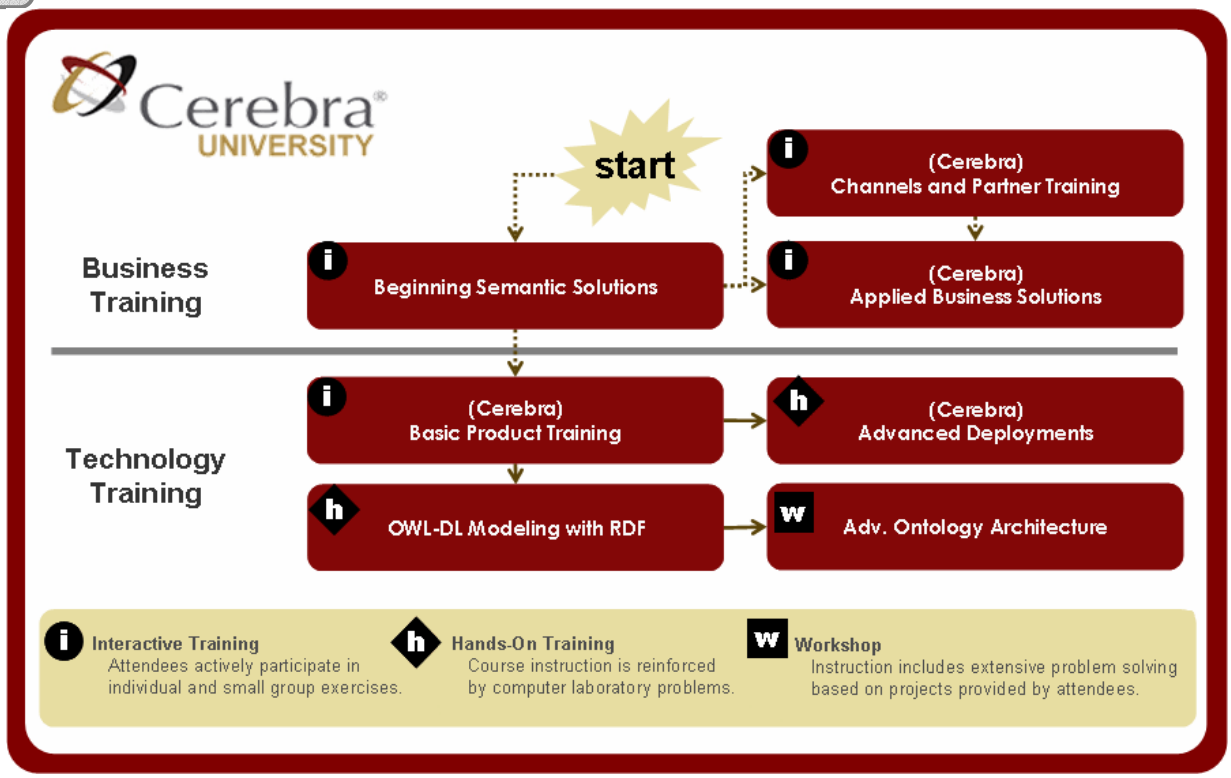
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Cerebra University



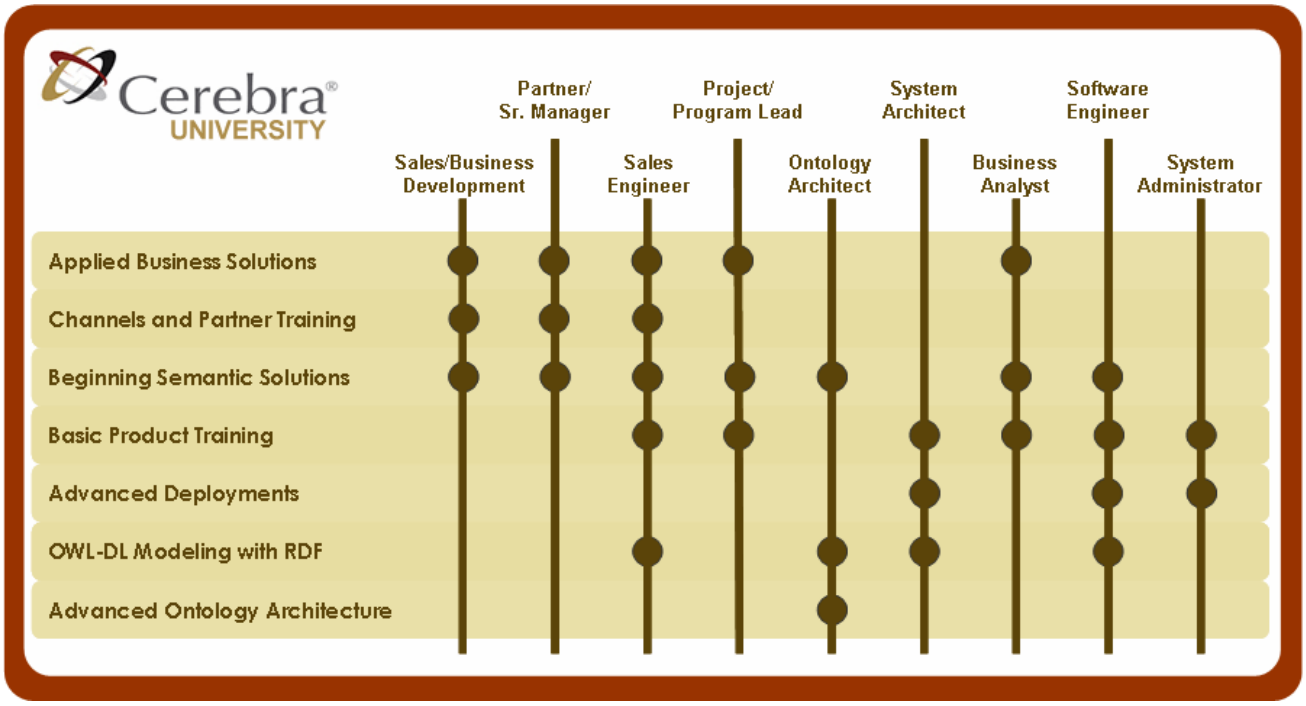

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

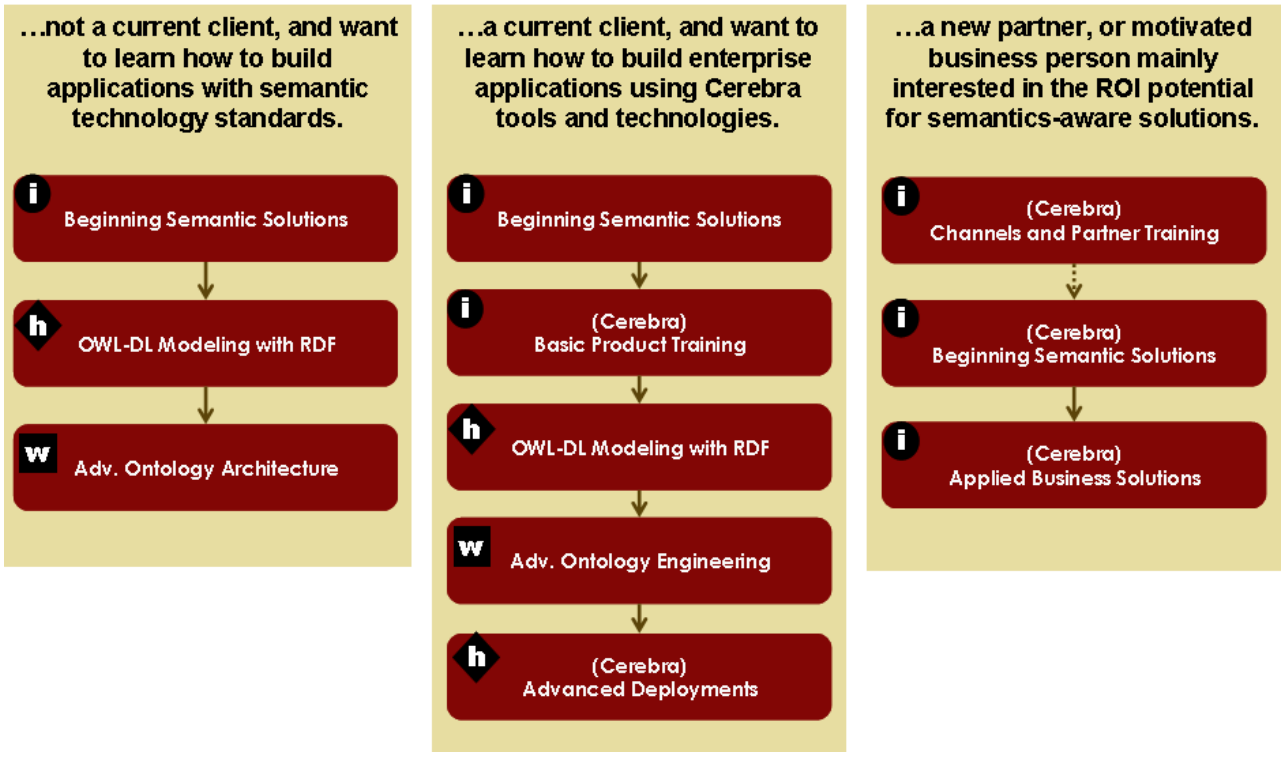
# Courses By Title




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Tracks by role




 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Advanced Ontology Architecture

- Designing Formal Taxonomies: Distinguishing formal and navigational taxonomies. We build the Easel Taxonomy.
- Designing Instance Based Taxonomies: Abox taxonomies and mereology.
- Mereology and Patonomy: After reviewing the differences with Part/Whole structures, students build a bill of materials for a particular Easel.
- Beyond Taxonomies: We examine issues in the topology of ontologies, how to import and reuse them. Students build a very simple enterprise ontology for integrating the Easel company's front and back office systems.
- Production Rules: Issues of getting a reasoner to work in a rules environment.
- Architectural Concerns: Other issues with getting Semantics to work in SOA and other complex environments.
- Testing and Validation: Issues in testing and validation, such as how to debug an inconsistent ontology.
- Performance Issues: Guidelines for knowing what to avoid, and when to be concerned about performance.

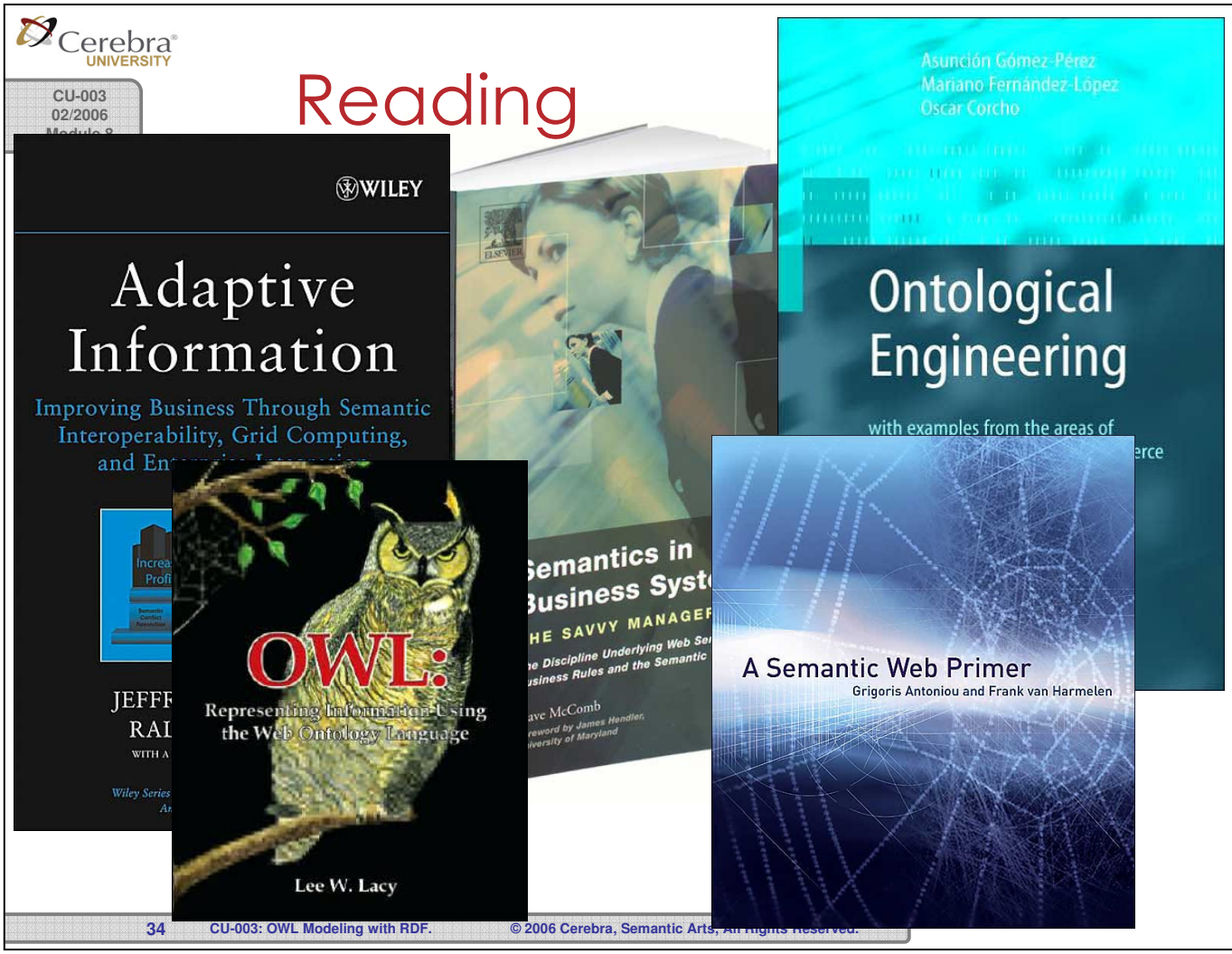
 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006

# Reading




 © 2006 Cerebra, Semantic Arts, All rights reserved.

Student Notes:

Cerebra UNIVERSITY

CU-003
02/2006
Module 8

# Web sites/blogs

- W3c's key web pages
  - http://www.w3.org/2001/sw/
  - http://www.w3.org/2004/OWL/
- Cerebra – Jeff Pollock's blogs, etc.
  - http://cerebra.com/products/inner_circle.html
- Semantic Arts
  - Articles http://semanticarts.com/Default.aspx?tabid=2161
  - Blog http://semantic-conference.blogs.com/semtech06/
- Mindswap has a number of good papers at
  - http://www.mindswap.org/papers/

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:

# Conferences

- Semantic Technology Conference
  - Annual, Bay Area, first week of March
  - http://semantic-conference.com/
- Semantic Web Conference
  - Annual, different venues, early November
  - 2006 is in Athens
  - http://iswc2005.semanticweb.org/

 © 2006 Cerebra, Semantic Arts, All Rights Reserved.

Student Notes:




## About Cerebra University

Cerebra University is a joint partnership between Cerebra and Semantic Arts committed to providing best-in-class education about the most advanced ways to use semantic technology to improve business efficiency, reduce costs, and enable new opportunities.




www.cerebra.com




www.semanticarts.com

## For More Information

For scheduling, pricing, or questions about Cerebra University Just-in-Time Training Workshops tailored to your needs, please send email to: training@cerebra.com