

Chapter 2

RDF Syntax 2



Topics

- Basic concepts of RDF
 - Resources, properties, values, statements, triples
 - URIs and URIrefs
 - RDF graphs
 - Literals, qnames
- Vocabularies and modeling
 - Vocabularies
 - Blank nodes, data modeling, types, reification
 - Lists, bags, collections
- Serialization of RDF graphs
 - XML, Turtle, Ntriples
- Critique of RDF

RDF type

- RDF has a **type** predicate that links a resource to another that denotes its type
 - **ex:john rdf:type foaf:Person .**
 - <http://example.org/john> >
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/0.1/Person> .
- RDFS adds sub-type concept & constraints between predicates & types of their arguments
- OWL adds still more concepts operating on types

Structured Values in RDF

- Given the triple like:

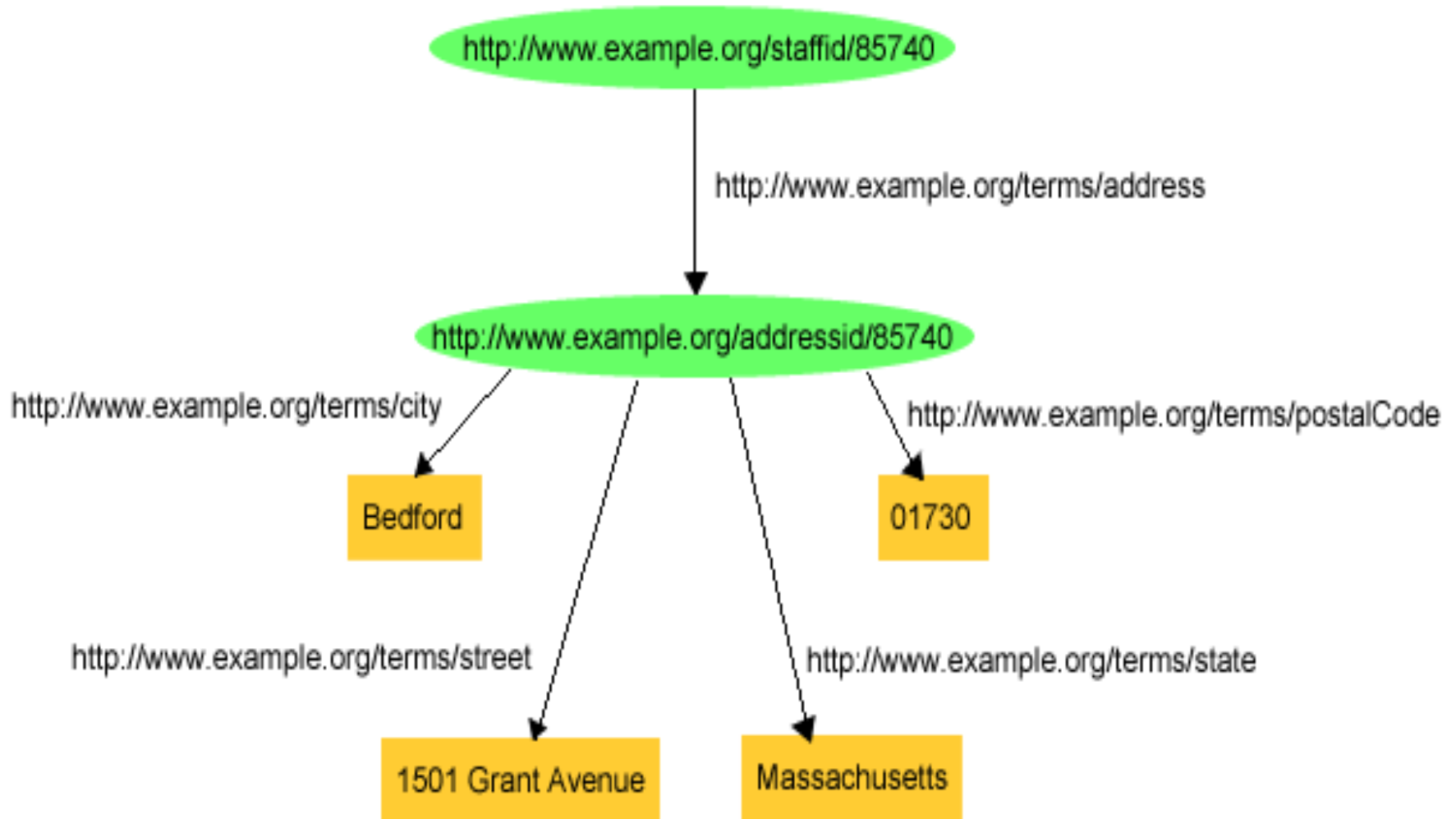
ex:857 exstaff:address "15 Grant Ave, Bedford, MA 01730".

- How can we represent separate information for the street, city, state and zip code?

- Two possibilities:

- Use four predicates (e.g., exstaff:street_address, ...) to associate values with exstaff:857
- Create an address resource to attach four predicates to & link it to exstaff:address with the ex:address predicate

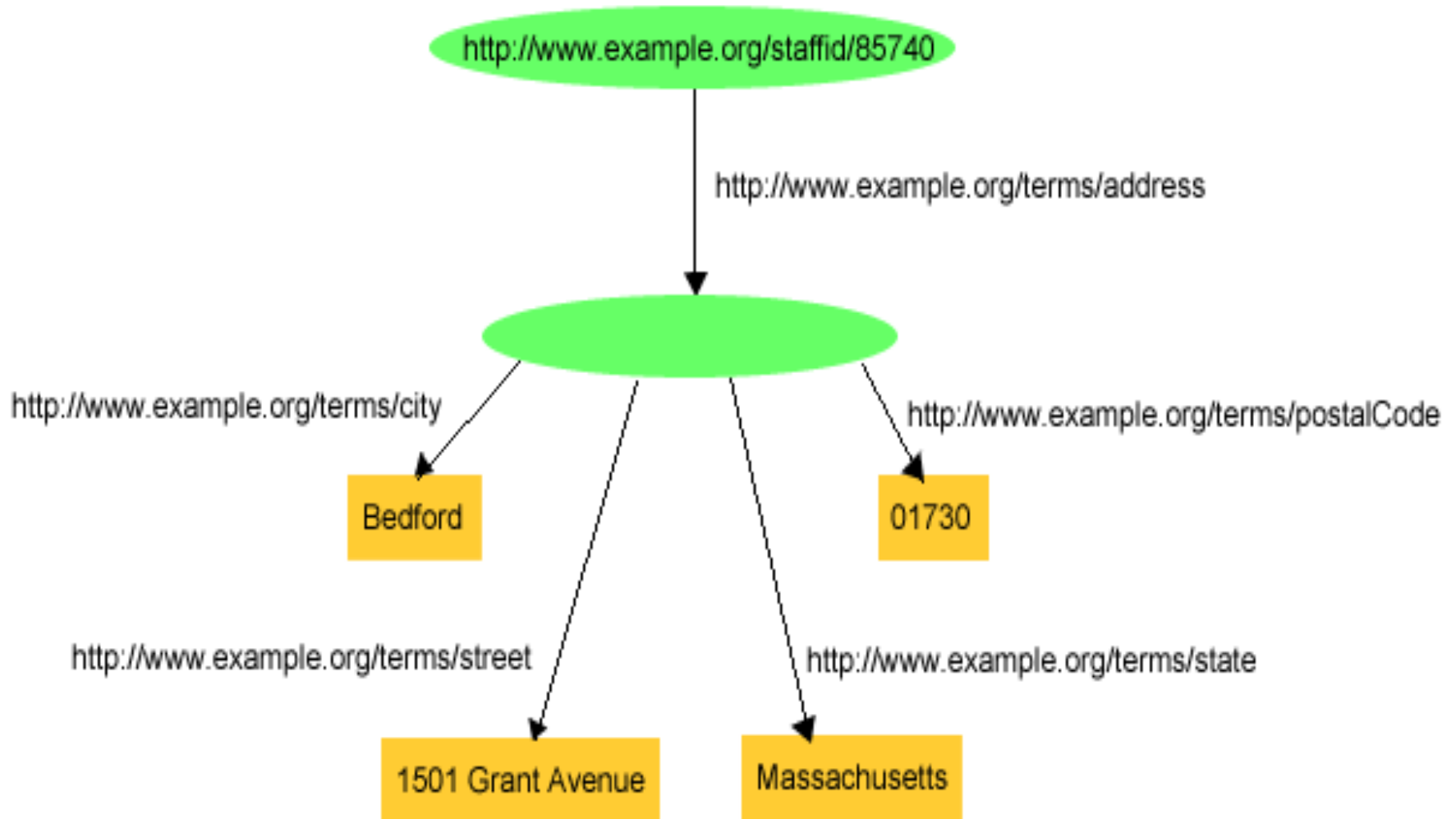
Structured Values in RDF



Structured Values in RDF

- This approach involves adding **many “inter-mediate” URIrefs** (e.g., exaddressid:85740) for aggregate concepts like John's address
- Such concepts may never need to be referred to directly from outside a particular graph, and hence **may not require “universal” identifiers**
- RDF allows us to use **blank nodes** with special **blank node identifiers** to deal with this issue
 - Node IDs in the `_` namespace are bnodes, e.g. `_:`

Blank Node, aka bnode



Blank Nodes Using Triples

exstaff:85740 exterms:address ?? .

?? exterms:postalCode "01730" .

Exstaff:72120 exterms:address ??? .

??? exterms:postalCode "01702" .

- Need to ensure that bnodes for 85740's and 72120's addresses are distinct
- Graphical notation does this by using two different objects for the bnodes
- RDF allows us to assign a special ID to a bnode while still maintaining its blank node nature

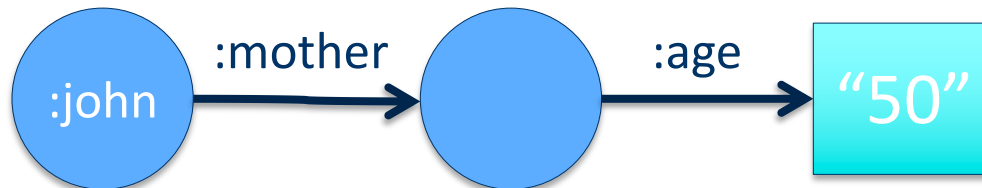
Blank Node Identifiers

```
exstaff:85740 exterm:address _:johnaddress .  
_:johnaddress exterm:street "1501 Grant Avenue" .  
_:johnaddress exterm:postalCode "01730" .
```

- Distinct bnode must have **different** bnode ids
- Bnode ids have significance only in a **single** graph
 - *dbpedia:Alan_Turing* refers to the same thing in every graph, but a bnode `_:1` in two different graphs may not
 - Merging two graphs requires us to rename their bnode ids to avoid accidental conflation (e.g., `_:1 => _:100`)
- Bnode ids may only appear as subjects or objects and **not as predicates** in triples

Semantics of Blank Nodes

- In terms of **first-order logic**, blank nodes correspond to existentially quantified variables
- Another example: “John’s mother is 50”
- **FOL**: $\exists x \text{ mother}(\text{john}, x) \wedge \text{age}(x, 50)$
- **RDF**: `:john :mother _:32 . _:32 :age "50" .`
- **FOL**: $\exists x \text{ mother}(x, \text{John}) \wedge \text{age}(x, 32)$



Blank nodes are good for

- Representing **n-ary relationships** in RDF
e.g., the relationship between John Smith and the street, city, state, and postal code components of his address
- To make statements about **resources that don't have URIs** but are described by relationships with other resources that do
e.g., John's mother; My car's engine

Example

- To make statements about a person named Jane Smith we could use her email address URI (<mailto:jane@example.org>) to denote her
- If we do so, how can we record information both about **Jane's mailbox** (e.g., the server it is on) as well as about **Jane herself** (e.g., her current physical address)? Similarly, if we use her Web page URI etc.

Bnode Example

When Jane herself does not have a URI, a blank node provides a better way of modeling this situation

```
_:jane ex:mailbox <mailto:jane@example.org> .
```

```
_:jane rdf:type ex:Person .
```

```
_:jane ex:name "Jane Smith" .
```

```
_:jane ex:empID "23748" .
```

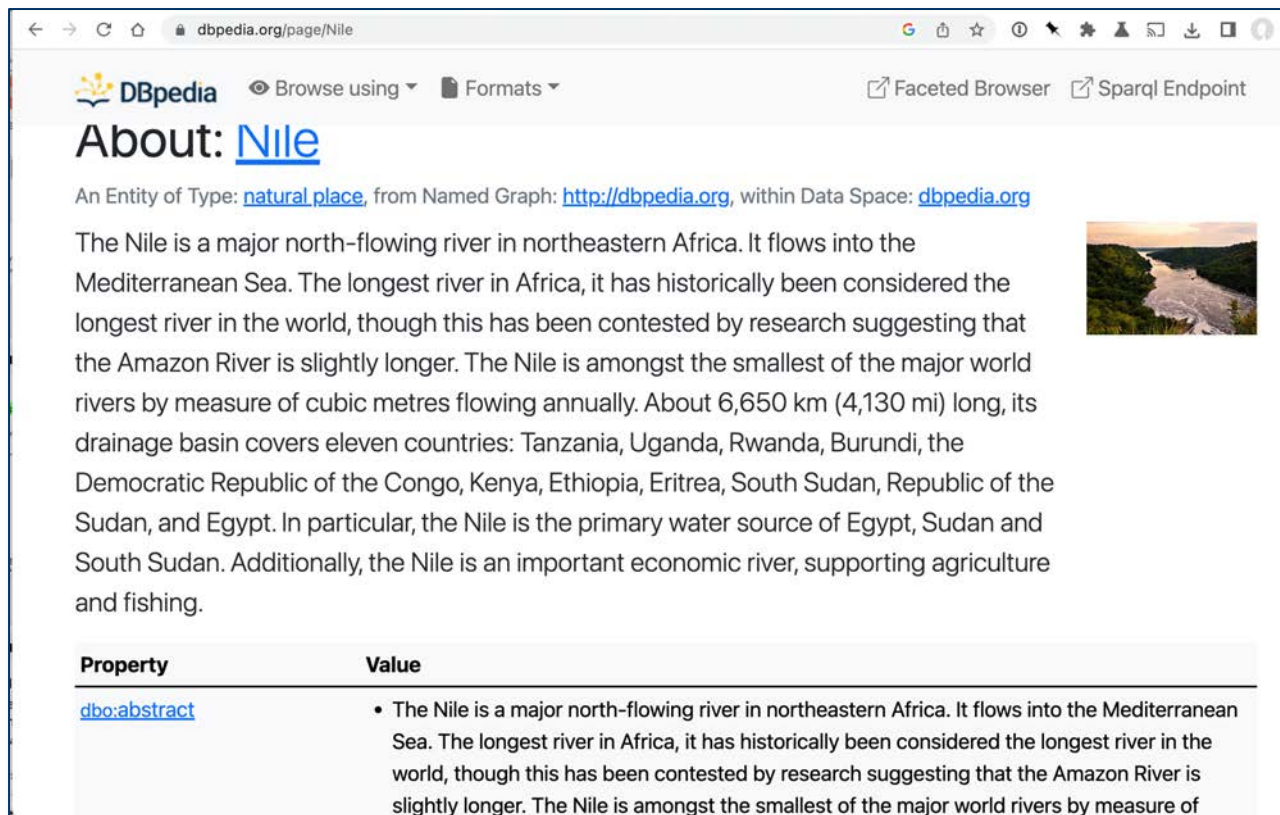
```
_:jane ext:age "26" .
```

Another use case: Measurements

- What does this mean?

[dbr:Nile](#) [dbp:length](#) "6853"^^xsd:integer

- Click on [dbp:length](#) to see its definition



The screenshot shows a web browser displaying the DBpedia page for 'Nile'. The page title is 'About: Nile'. Below the title, it states 'An Entity of Type: [natural place](#), from Named Graph: <http://dbpedia.org>, within Data Space: [dbpedia.org](#)'. The main text describes the Nile as a major north-flowing river in northeastern Africa, flowing into the Mediterranean Sea. It is the longest river in Africa and has historically been considered the longest river in the world, though this has been contested by research suggesting that the Amazon River is slightly longer. The Nile is amongst the smallest of the major world rivers by measure of cubic metres flowing annually. About 6,650 km (4,130 mi) long, its drainage basin covers eleven countries: Tanzania, Uganda, Rwanda, Burundi, the Democratic Republic of the Congo, Kenya, Ethiopia, Eritrea, South Sudan, Republic of the Sudan, and Egypt. In particular, the Nile is the primary water source of Egypt, Sudan and South Sudan. Additionally, the Nile is an important economic river, supporting agriculture and fishing. To the right of the text is a small image of a river flowing through a valley. At the bottom of the page, there is a table with two columns: 'Property' and 'Value'. The first row shows the property '[dbo:abstract](#)' and its value, which is the same text as the main body of the page.

Property	Value
dbo:abstract	<ul style="list-style-type: none">• The Nile is a major north-flowing river in northeastern Africa. It flows into the Mediterranean Sea. The longest river in Africa, it has historically been considered the longest river in the world, though this has been contested by research suggesting that the Amazon River is slightly longer. The Nile is amongst the smallest of the major world rivers by measure of

Another use case: Measurements

- What does this mean?

[dbr:Nile](#) [dbp:length](#) "6853"^^xsd:integer

- We can click on [dbp:length](#) to see its definition

[dbp:length](#) rdf:type rdf:Property .

[dbp:length](#) rdfs:label "Length"@en .

- Unfortunately, the definition doesn't specify the unit of measurement 😞

Another use case: Measurements

- What does this mean?

dbr:Nile dbp:length "6853"^^xsd:integer

- Measurements typically have a numeric *value* and a *unit*
 - **Weight:** 2.4 pounds vs. 2.4 kilograms
 - **Length:** 5 miles vs. 5 kilometers
 - **Price:** 29.00 in US Dollars vs. 21.16 Euro
 - **Time:** 30 years vs. 3 milliseconds
- We can use a bnode to represent a measurement as a pair with a value and unit

Measurements

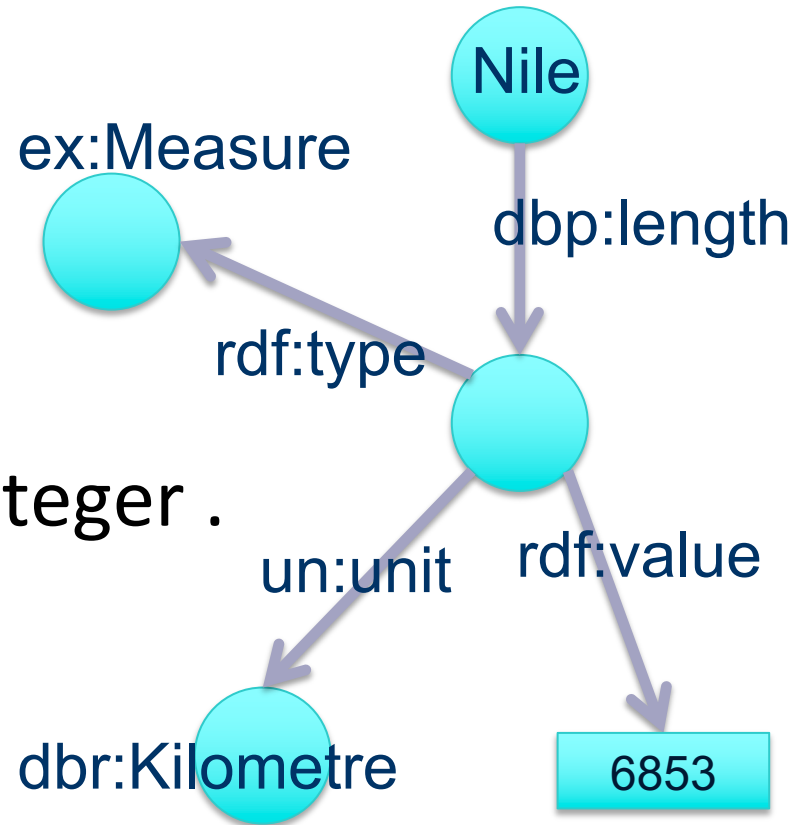
- What does this mean?

```
dbr:Nile dbp:length _:1 .
```

```
_:1 rdf:type ex:Measure .
```

```
_:1 rdf:value "6853"^^xsd:integer .
```

```
_:1 un:units dbr:Kilometre .
```



- The RDF namespace has a *value* property but assigns no specific meaning to it

Containers

Container Elements

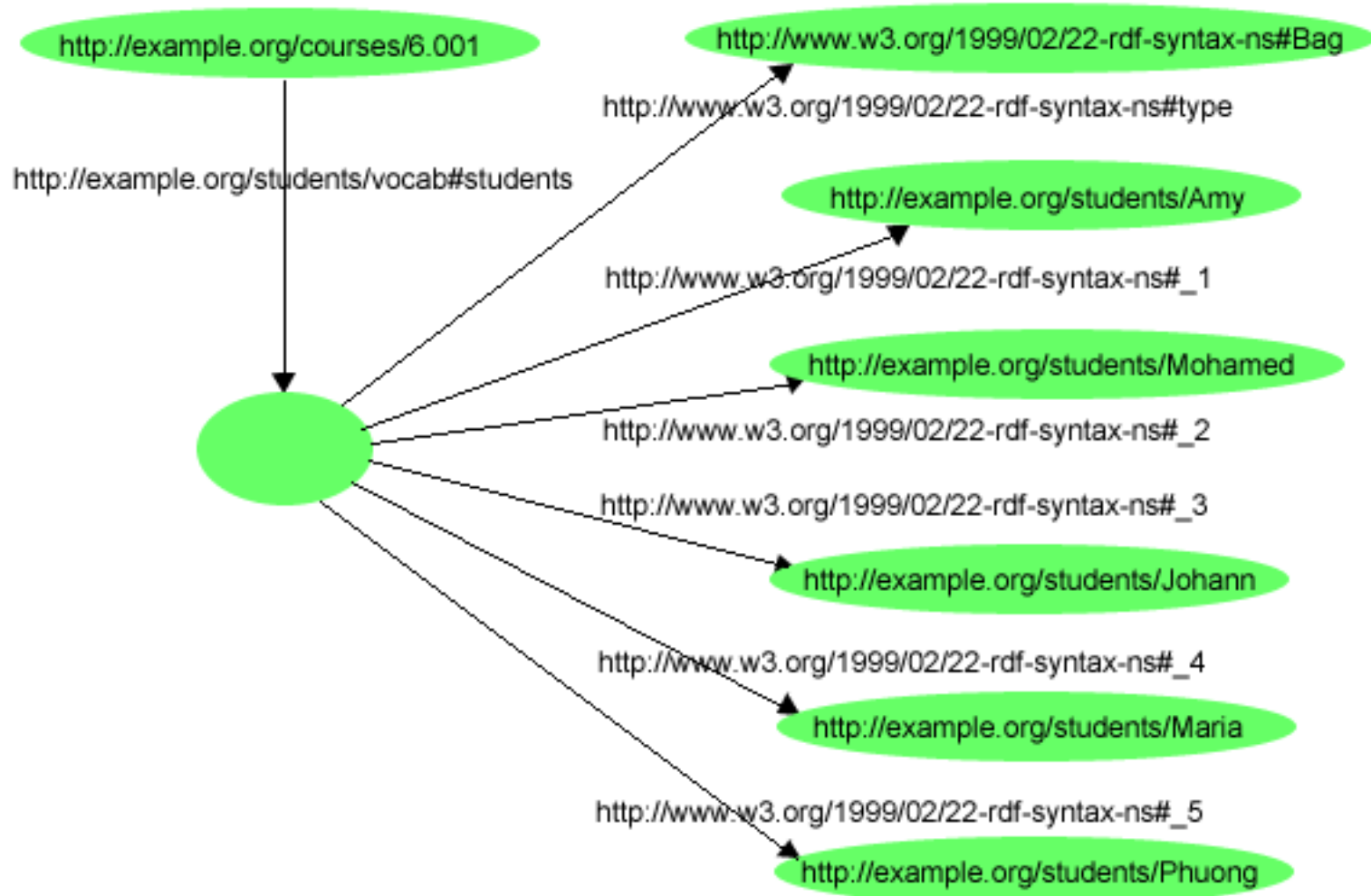
- RDF has some vocabulary to describe collections of things and make statements about them
- E.g., we may wish to talk about the courses given by a particular lecturer
- The content of container elements are named **rdf:_1**, **rdf:_2**, etc.
 - Alternatively **rdf:li**
- Containers seem a bit messy in RDF, but are needed
- `:john :teaches [a rdf:Bag; rdf:li :cmisc201, :cmisc202, cmisc345 .] .`

Three Types of Container Elements

- **rdf:Bag** an unordered container, allowing multiple occurrences
e.g., members of the faculty, documents in a folder
- **rdf:Seq** an ordered container, which may contain multiple occurrences
e.g., modules of a course, items on an agenda, alphabetized list of staff members
- **rdf:Alt** a set of alternatives
e.g., the document home site and its mirrors, translations of a document in various languages

Example for a Bag

Let's describe a course with a collection of students



Example for a Bag

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

@prefix s: <http://example.org/students/vocab#>.

<http://example.org/courses/6.001>

s:students [

 a rdf:Bag;

 rdf:_1 <http://example.org/students/Amy>;

 rdf:_2 <http://example.org/students/Mohamed>;

 rdf:_3 <http://example.org/students/Johann>;

 rdf:_4 <http://example.org/students/Maria>;

 rdf:_5 <http://example.org/students/Phuong>.

].

Bags and Seqs are never full!

- RDF's semantics is "open world", so...
 - Not possible "to close" the container, to say: "these are **all** elements, there are no more"
 - RDF is a graph, with no way to exclude the possibility that there is another graph somewhere describing additional members
- Lists are collections with only the specified members mentioned.
- Described using a linked list pattern via:
 - `rdf:List`, `rdf:first`, `rdf:rest`, `rdf:nil`

Open vs. closed world semantics

- Reasoning systems make a distinction between open and closed world semantics
 - OWS: being unable to prove that something is true or false says nothing about its veracity
 - CWS: what cannot be proven to be true is false
- Default model for Semantic Web is OWS
 - This was a design decision made early on

Open vs. closed world semantics

- Classical logic uses Open World Semantics

Being unable to prove $P=NP$ doesn't convince us that it's false

- Database systems typically assume CWS

The DB includes all trains between NYC and DC

- Prolog's unprovable operator (not or $\backslash+$) supports CWS

`flys(x) :- bird(x), $\backslash+$ flightless(x).`

`flightless(x) :- penguin(x); ostrich(x); emu(x).`

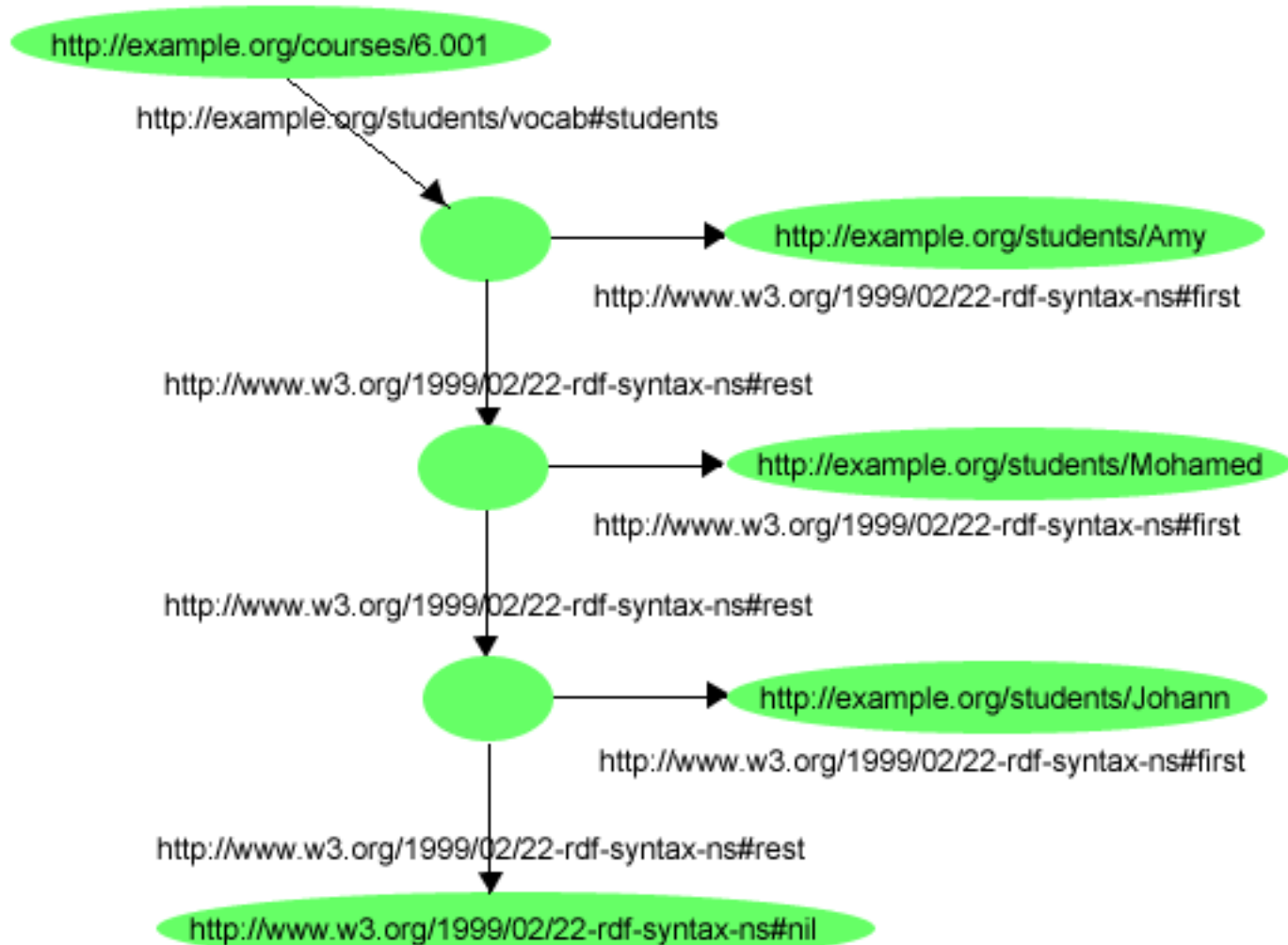
- Some systems let us specify for which predicates we have complete knowledge and for which we don't

- If UMBC's DB doesn't list you as registered for CMSC691, you are not registered

- UMBC's DB system knows some of your minors but not all

RDF Lists

An ordered list of the three students in a class



RDF Lists

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

@prefix s: <http://example.org/students/vocab#>.

<http://example.org/courses/6.001>

s:students

[a rdf:List;

 rdf:first <http://example.org/students/Amy>;

 rdf:rest [a rdf:list

 rdf:first <http://example.org/students/Mohamed>;

 rdf:rest [a rdf:List;

 rdf:first <http://example.org/students/Johann>;

 rdf:rest rdf:nil]]] .

RDF Lists

Turtle has special syntax to represent lists:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
```

```
@prefix s: <http://example.org/students/vocab#>.
```

```
<http://example.org/courses/6.001>
```

```
  s:students (
```

```
    <http://example.org/students/Amy>
```

```
    <http://example.org/students/Mohamed>
```

```
    <http://example.org/students/Johann>
```

```
  ).
```

Critique of RDF

RDF Critique: Properties

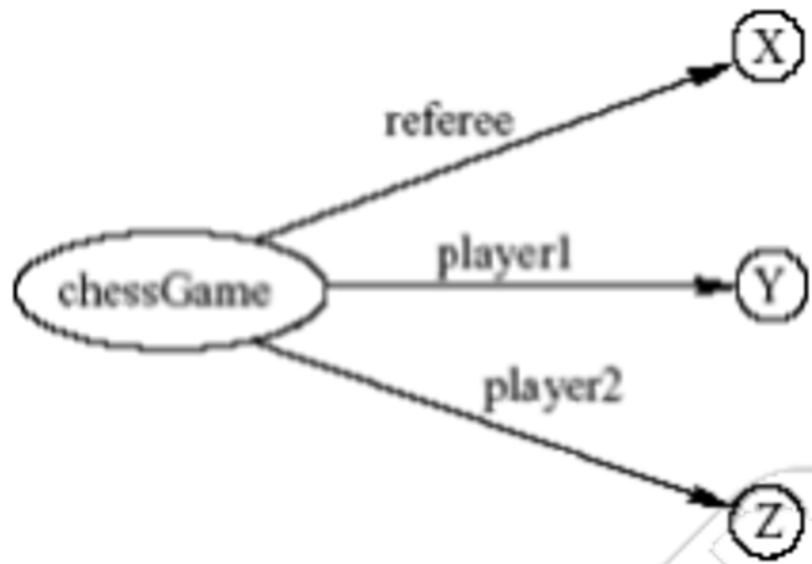
- Properties are special kinds of resources
 - Properties can be used as the object in an object-attribute-value triple (statement)
 - Defined **independent** of resources
- This possibility offers flexibility
- But it is unusual for modelling languages and OO programming languages
- It can be confusing for modellers

RDF Critique: Binary Predicates

- RDF uses only binary properties
 - This is a restriction because often we use predicates with more than two arguments
 - But binary predicates can simulate these
- Example: **referee(X, Y, Z)**
 - **X** is referee in a chess game between players **Y** and **Z**
- Example: **between(NYC, Newark, Philadelphia)**

RDF Critique: Binary Predicates

- We introduce:
 - a new auxiliary resource **chessGame**
 - the binary predicates **ref**, **player1**, and **player2**
- We can represent **referee(X,Y,Z)** as:



RDF Critique: Reification

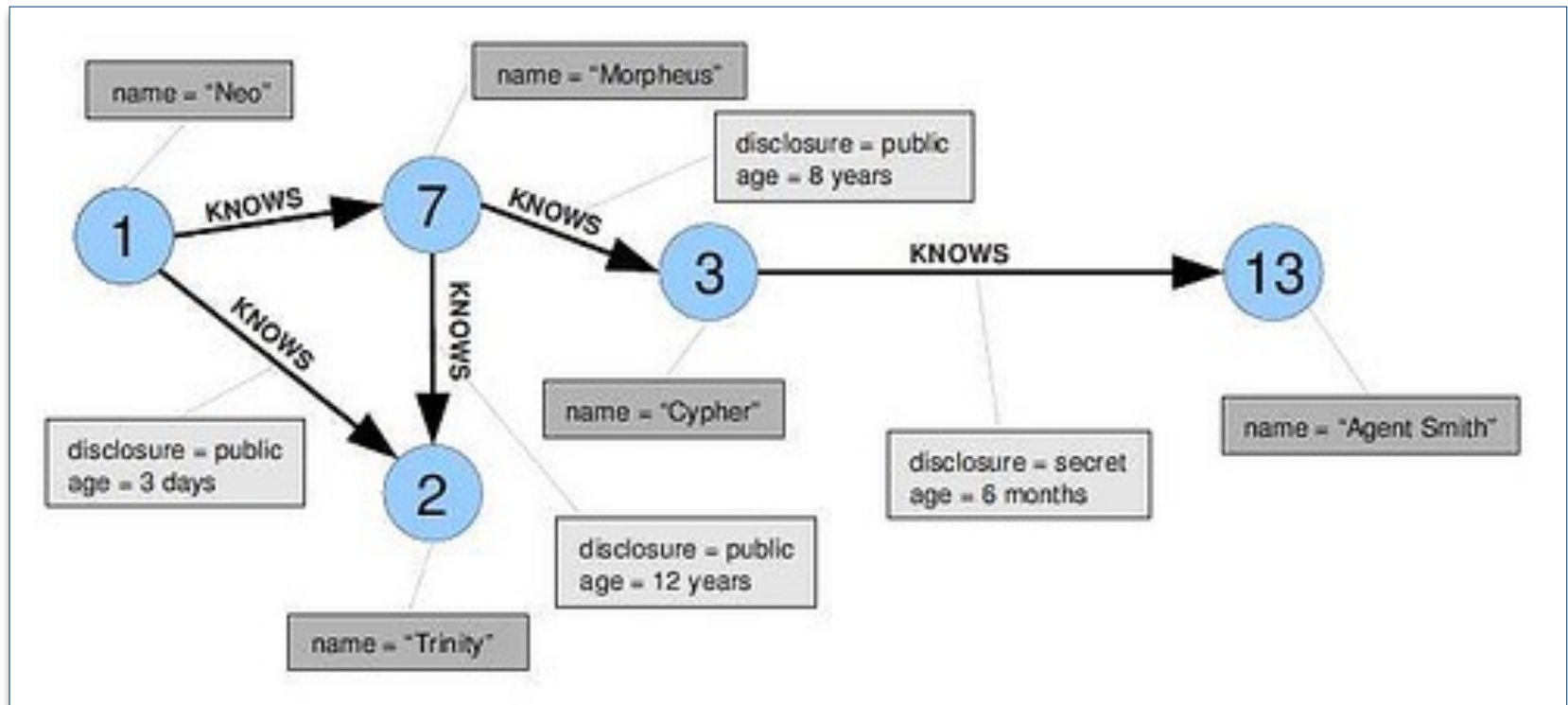
- The reification mechanism is quite powerful
- It may be misplaced in a simple language like RDF
- Making statements about statements introduces a level of complexity that is not necessary for a basic layer of the Semantic Web
- Instead, it would have appeared more natural to include it in more powerful layers, which provide richer representational capabilities

RDF Critique: Graph Representation

- The simple graph or network representation has more drawbacks
- Linear languages introduce ways to represent this with parentheses or a way to represent a block structure
- Scoping, for example, is clumsy at best in RDF
believe(john, and (love(bob, carol), love(carol, bob)))
- Some of these are addressed through the notion of a *named graph* in RDF

RDF graph model is simple

- RDF's graph model is a simple one
- [Neo4J](#) is a popular graph database where both nodes and links can have properties



RDF Critique: Summary

- RDF has its idiosyncrasies and is not an optimal modeling language **but**
- It is already a de facto standard
- It has sufficient expressive power
 - Reasonable foundation on which to build
- Using RDF offers the benefit that information maps unambiguously to a model