

# 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

Types

# 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

# Data Modeling

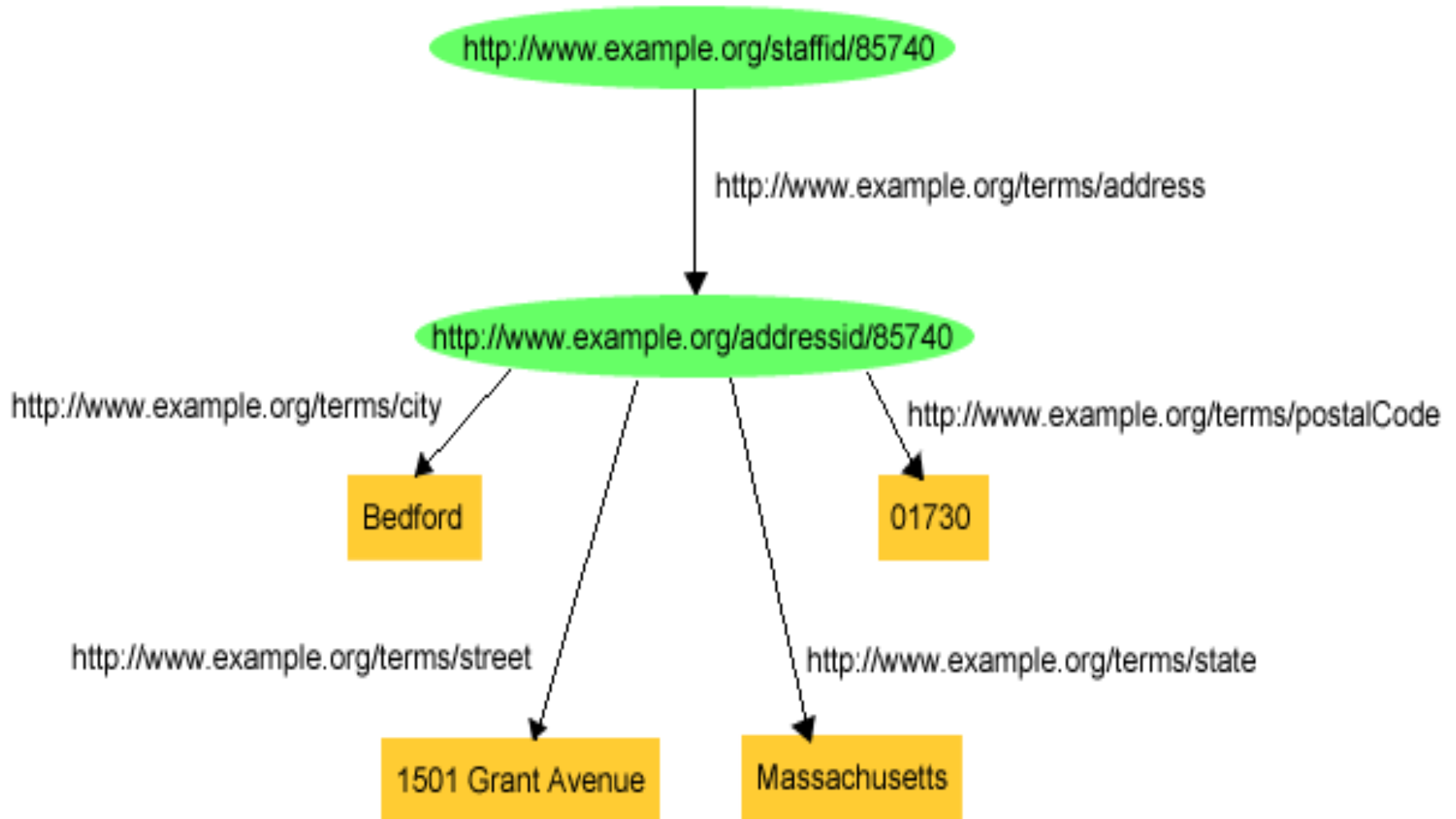
# Structured Values in RDF

- Given the triple like:

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

- How can we best 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 the four predicates to and link that to exstaff:address with the ex:address predicate

# Structured Values in RDF



# Structured Values in RDF

Pr as triples:

exstaff:85740 exterms:address exaddressid:85740 .

exaddressid:85740 exterms:street "1501 Grant Ave" .

exaddressid:85740 exterms:city "Bedford" .

exaddressid:85740 exterms:state "MD" .

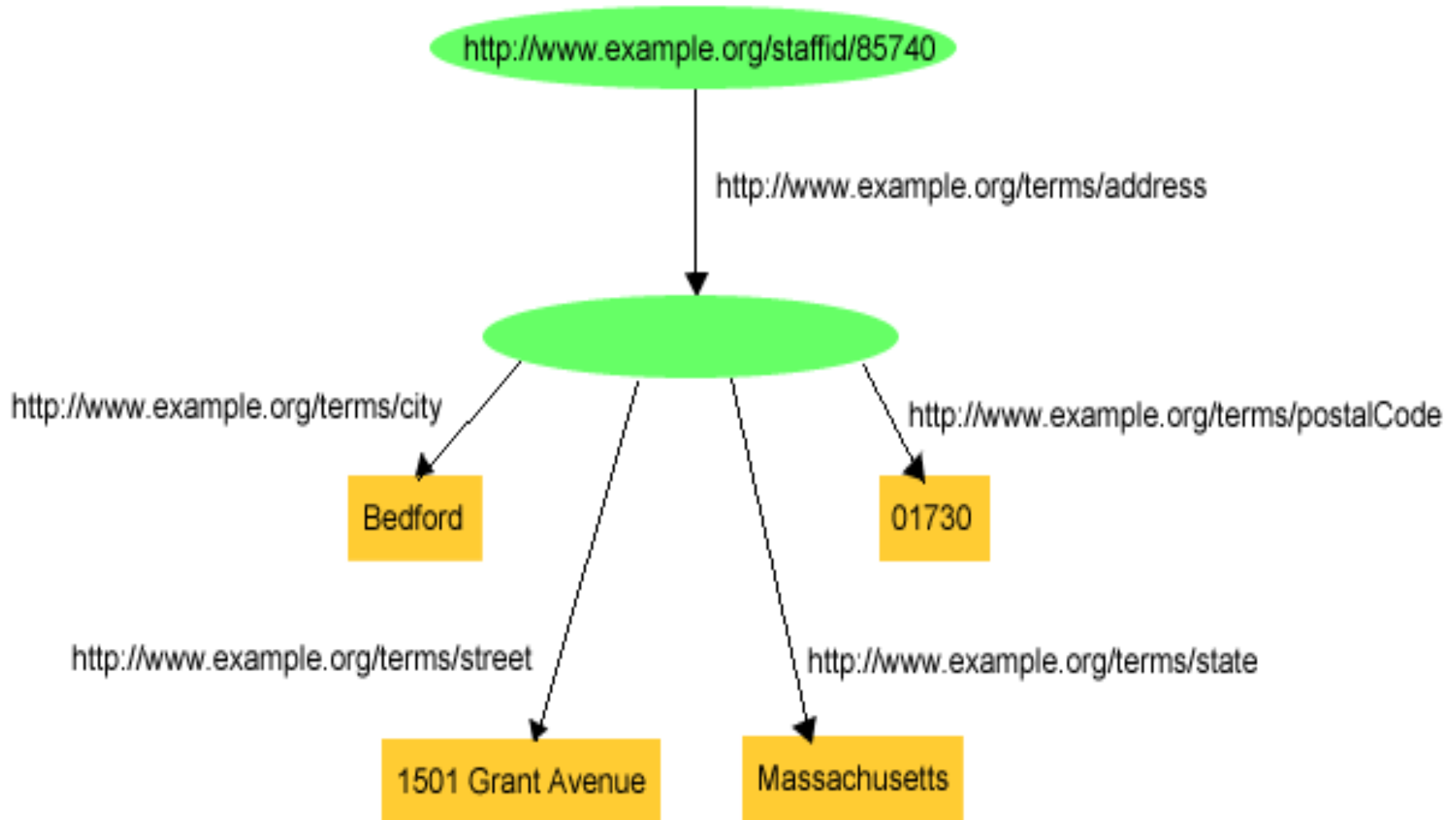
exaddressid:85740 exterms:postalCode "01730" .



# 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** and **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" .

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

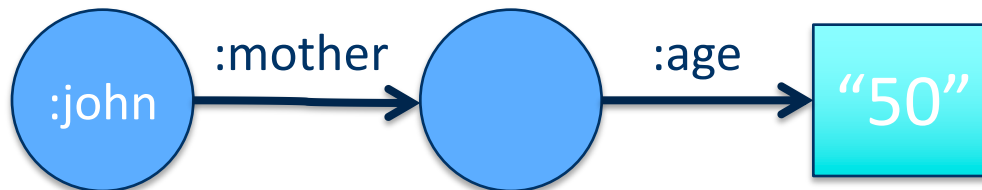
# Blank Node Identifiers

```
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" .`



# 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

# Example

- To make statements about Jane Smith we could use her email address URI (<mailto:jane@example.org>) to denote her
- Well, if we do so, how are we going to 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 exterm:mailbox <mailto:jane@example.org> .  
_:jane rdf:type exterm:Person .  
_:jane exterm:name "Jane Smith" .  
_:jane exterm:empID "23748" .  
_:jane exterm:age "26" .
```

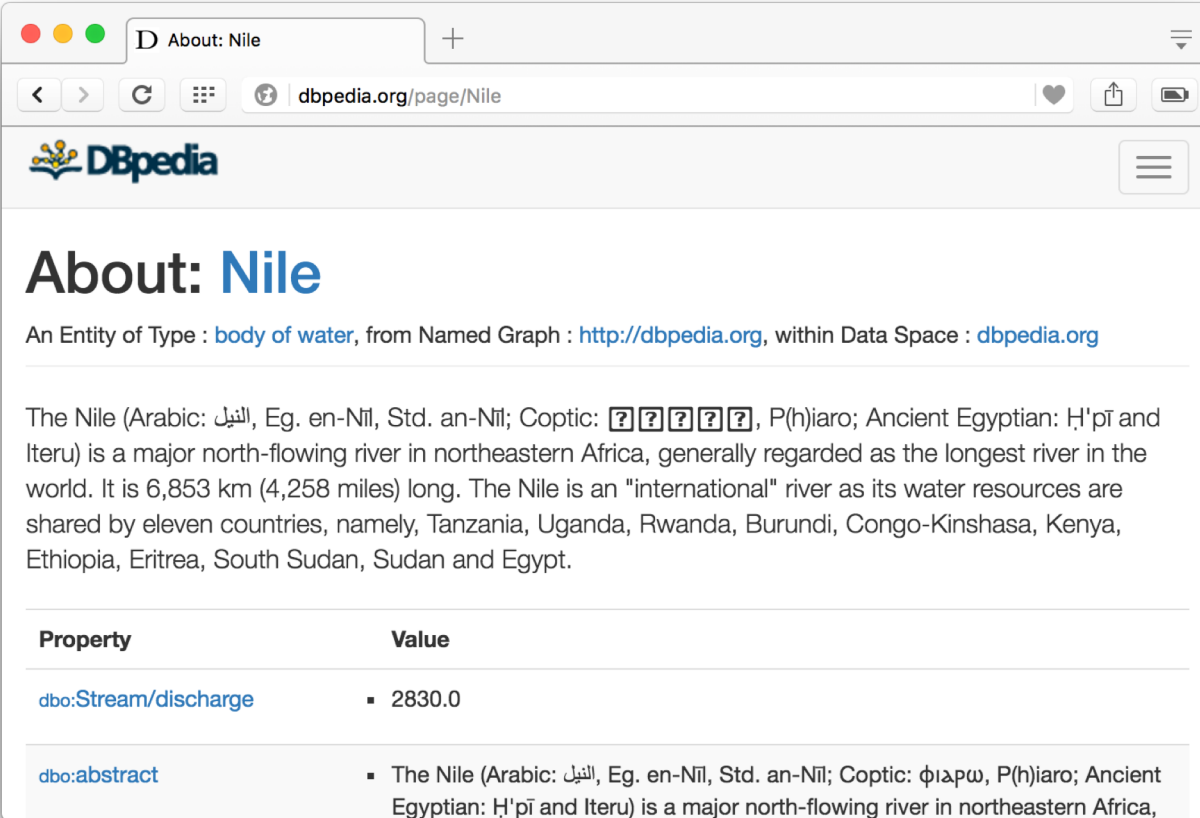


# Another use case: Measurements

- What does this mean?

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

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



The Nile (Arabic: النيل, Eg. en-Nīl, Std. an-Nīl; Coptic: Ⲛⲓⲗⲏ, P(h)iaro; Ancient Egyptian: Ḥ'pī and Iteru) is a major north-flowing river in northeastern Africa, generally regarded as the longest river in the world. It is 6,853 km (4,258 miles) long. The Nile is an "international" river as its water resources are shared by eleven countries, namely, Tanzania, Uganda, Rwanda, Burundi, Congo-Kinshasa, Kenya, Ethiopia, Eritrea, South Sudan, Sudan and Egypt.

Property	Value
<a href="#">dbo:Stream/discharge</a>	▪ 2830.0
<a href="#">dbo:abstract</a>	▪ The Nile (Arabic: النيل, Eg. en-Nīl, Std. an-Nīl; Coptic: Ⲛⲓⲗⲏ, P(h)iaro; Ancient Egyptian: Ḥ'pī and Iteru) is a major north-flowing river in northeastern Africa,

# 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

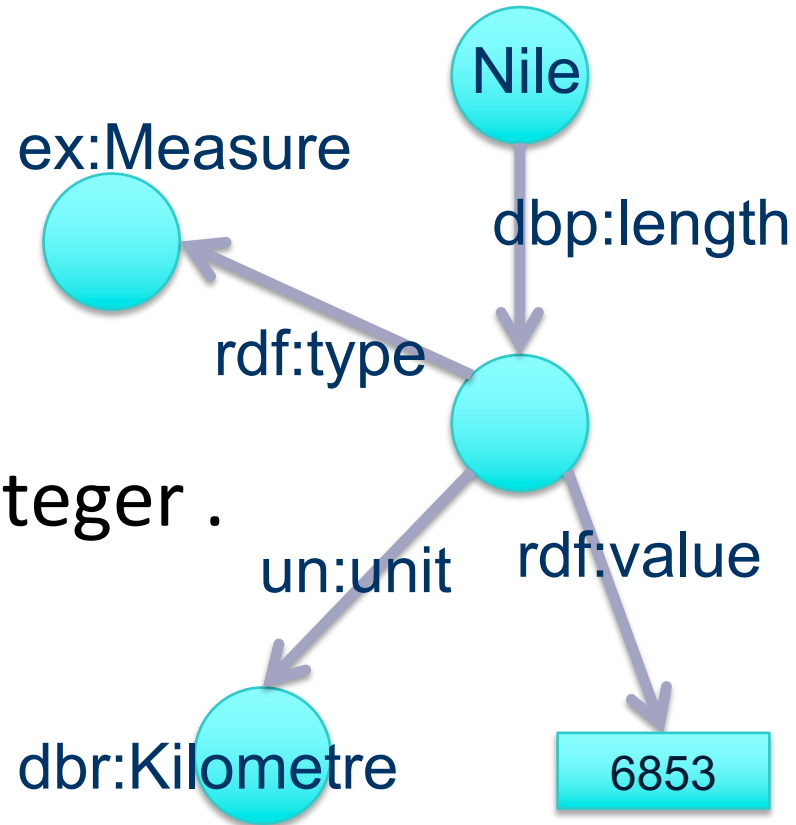
- What does this mean?

```
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

# Serialization

# RDF Serialization

- Abstract model for RDF is a graph
- Serialize as text for exchange, storage, viewing and editing in text editors
- The big three
  - XML/RDF – the original
  - Ntriples – simple, but verbose; good for processing
  - Turtle – compact, easy for people to read and write
- Special formats
  - Trig – a format for named graphs
  - RDFa – embed RDF in HTML attributes
  - JSON-LD – RDF statements as a JSON object

# XML encoding for RDF

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:bib="http://daml.umbc.edu/ontologies/bib/">
<rdf:Description about="http://umbc.edu/~finin/talks/idm02/">
  <dc:title>Intelligent Information Systems on the Web </dc:Title>
  <dc:creator>
    <rdf:Description >
      <bib:name>Tim Finin</bib:Name>
      <bib:email>finin@umbc.edu</bib:Email>
      <bib:aff resource="http://umbc.edu/" />
    </rdf:Description>
  </dc:creator>
</rdfdescription>
</rdf:RDF>
```

RDF/XML is a W3C  
Standard widely used for  
storage and exchange

Being supplanted by other  
forms

Complex and confusing so we  
won't spend time on it

# Ntriples

- Good for ingesting into a program or store
- Sequence of triples each terminated with a “.”
- URIs encased in angle brackets; no QNames; literals in double quotes
- Trivial to parse/generate; common download format for RDF datasets (e.g., [DBpedia](#))
- Uses lots of characters due to repeated URLs, but compresses well

[W3C Specification](#)

```
<http://example.org/Turing><http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .  
<http://example.org/Turing> <http://xmlns.com/foaf/0.1/name> "Alan Turing" .  
<http://www.w3.org/2001/sw/RDFCore/ntriples/> <http://xmlns.com/foaf/0.1/mbox> <mailto:alan@turing.org> .
```



# Turtle

- Ntriples  $\subset$  Turtle  $\subset$  N3
- Compact, easy to read and write and parse
- Qnames, [ ] notation for blank nodes, ; and ,

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

@prefix dc: <http://purl.org/dc/elements/1.1/> .

@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<http://www.w3.org/TR/rdf-syntax-grammar>

dc:title "RDF/XML Syntax Specification (Revised)" ;

dc:creator [ foaf:name "Dave Beckett";

foaf:mbox <mailto:dave@beckett.org> ,

<mailto:dbeck@gmail.com>

].

# Some details

- @PREFIX lines define namespace abbreviations
- Basic pattern is  
Subj pred1 value1;  
    pred2 value2;  
    pred3 value3, value4 .
- Special notation for the `rdf:type` predicate  
:john a **foaf:Person**; foaf:name "John Smith" .
- Special notation for anonymous bnodes  
:john foaf:knows [ a foaf:Person; foaf:nick "Bob" ].

# Notation3 or N3

- N3 was an early turtle-like notation developed by Sir Tim\_Berners Lee himself
- Included support for inference rules
  - See [CWM](#) for software
- Never became a recommended W3C standard
  - Some of its features were problematic for OWL
  - Supplanted by Turtle

# Try...

- Some simple RDF serialization [examples](#)
- Simple.ttl

# A simple Turtle example

@prefix foaf: <http://xmlns.com/foaf/0.1/> .

@prefix : <#> .

:john a foaf:Person;

foaf:gender "Male";

foaf:name "John Smith", "Johnny Smith";

foaf:knows :mary,

[a foaf:Person;

foaf:mbox <mailto:mary.smith@gmail.com>] .

:mary a foaf:Person;

foaf:name "Mary Smith" .

# Notation translation

- Most modern Semantic Web software can read and write rdf in all major serializations
  - E.g., Protégé, Jena, Sesame,
- There are also simple programs that can convert between them
  - [rdf2rdf](#) is an example written in Java

# Reification

# Reification

- Sometimes we wish to make **statements about other statements**

E.g., to record provenance data, probability, or to assert  
*:john :believes { :mary :loves :john }*

- We must be able to refer to a statement using an identifier
- RDF allows such reference through a reification mechanism which turns a statement into a resource

# Reify

- Etymology: Latin *res* thing
- Date: 1854
- To regard (something abstract) as a material or concrete thing



## Wikipedia: reification (computer science)

Reification is the act of making an abstract concept or low-level implementation detail of a programming language accessible to the programmer, often as a first-class object. For example,

- The C programming language reifies the low-level detail of memory addresses
- The Scheme programming language reifies continuations (approximately, the call stack)
- In C#, reification is used to make parametric polymorphism implemented as generics a first-class feature of the language
- ...

# Reification Example

:949352 uni:name "Grigoris Antoniou" .

reifies as

```
[a rdf:Statement;  
  rdf:subject: :949352  
  rdf:predicate uni:name;  
  rdf:object "Grigoris Antoniou" ] .
```

# Another reification example

“Alice suspects that Bob loves Carol”

@prefix ep: <<http://example.com/epistimology>>

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

@prefix xsd: <http://www.w3.org/2001/XMLSchema>

:bob :loves :carol .

[:alice ep:believes

[a rdf:Statement;

  rdf:subject :bob;

  rdf:predicate :loves;

  rdf:object :carol;

  ex:certainty “0.50”^^xsd:integer]

# Reification

- **rdf:subject**, **rdf:predicate** & **rdf:object** allow us to access the parts of a statement
- The **ID** of the statement can be used to refer to it, as can be done for any description
- We write an **rdf:Description** if we don't want to talk about a statement further
- We write an **rdf:Statement** if we wish to refer to a statement

# Containers

# Container Elements

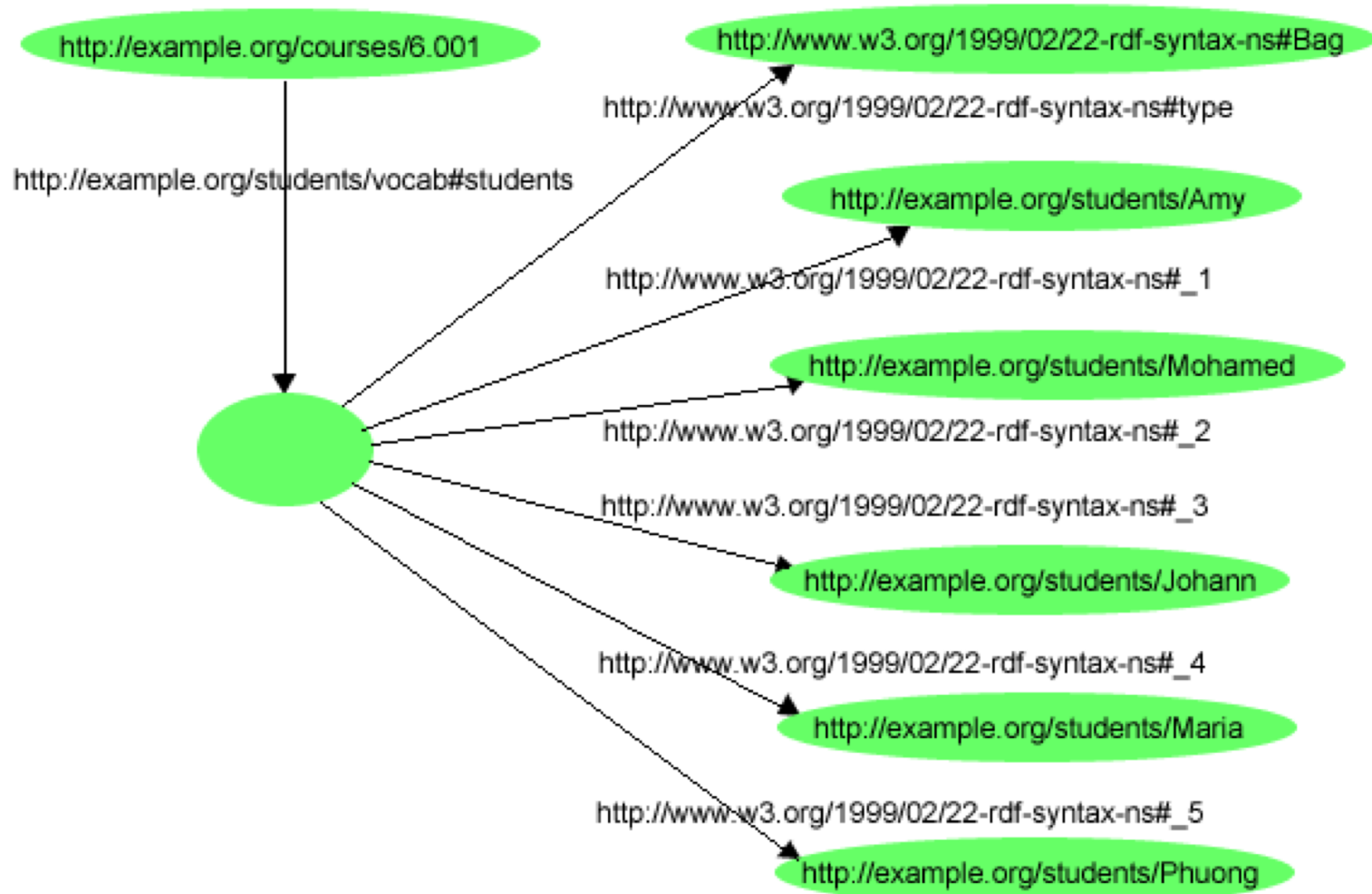
- RDF has some vocabulary to describe collections of things and make statements about them
- E.g., we may wish to talk about the courses given by a particular lecturer
- The content of container elements are named **rdf:\_1**, **rdf:\_2**, etc.
  - Alternatively **rdf:li**
- Containers seem a bit messy 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).`

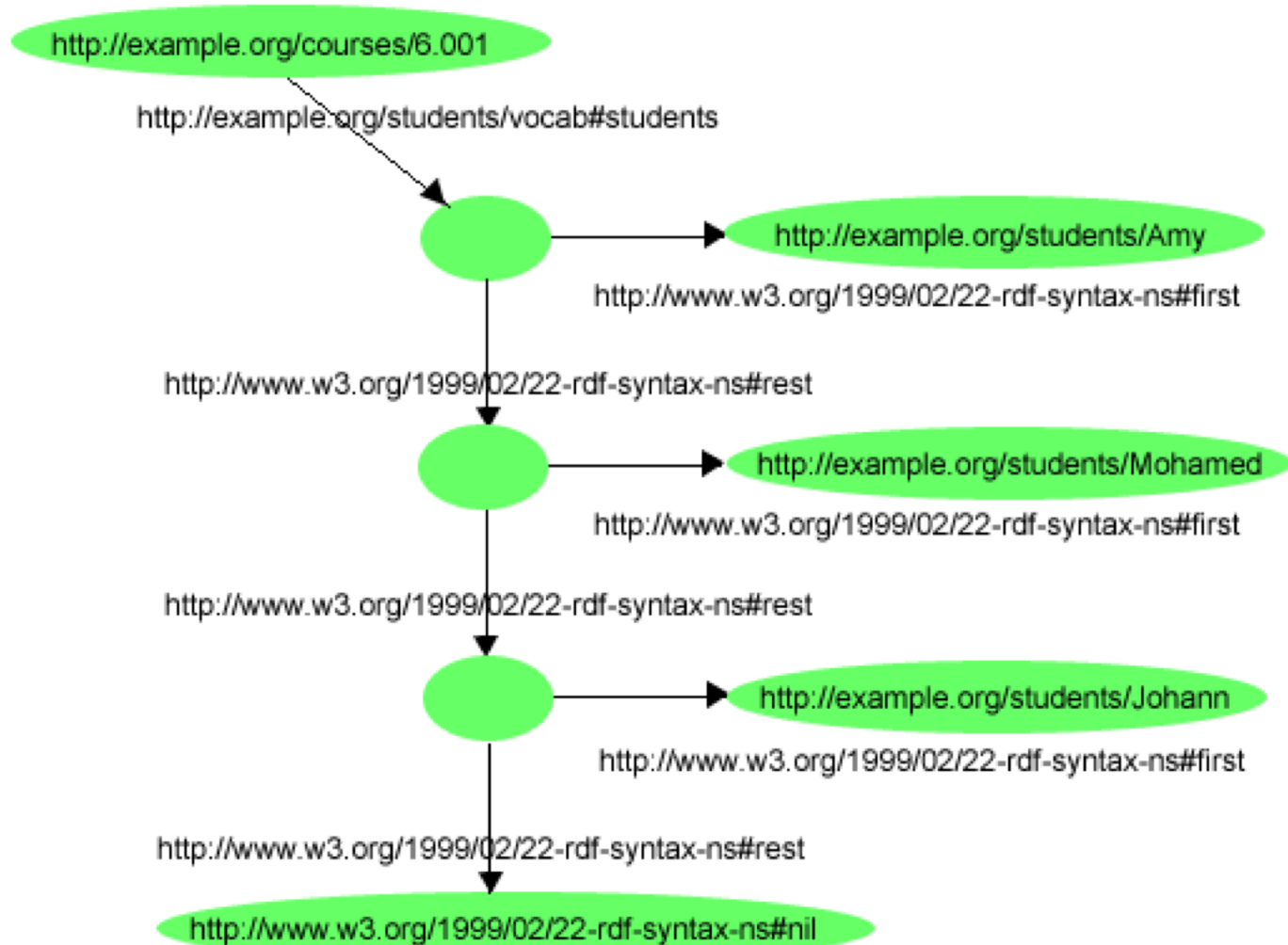
- Some systems let us specify for which predicates we have complete knowledge and for which we don't

- 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

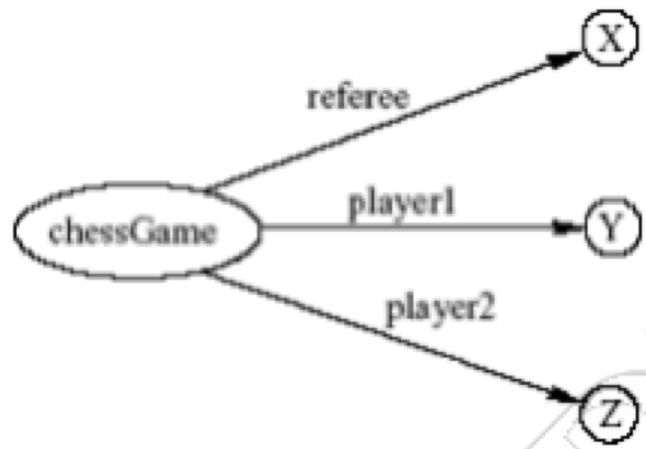
- Properties are special kinds of resources
  - Properties can be used as the object in an object-attribute-value triple (statement)
  - Defined **independent** of resources
- This possibility offers flexibility
- But it is unusual for modelling languages and OO programming languages
- It can be confusing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

# 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

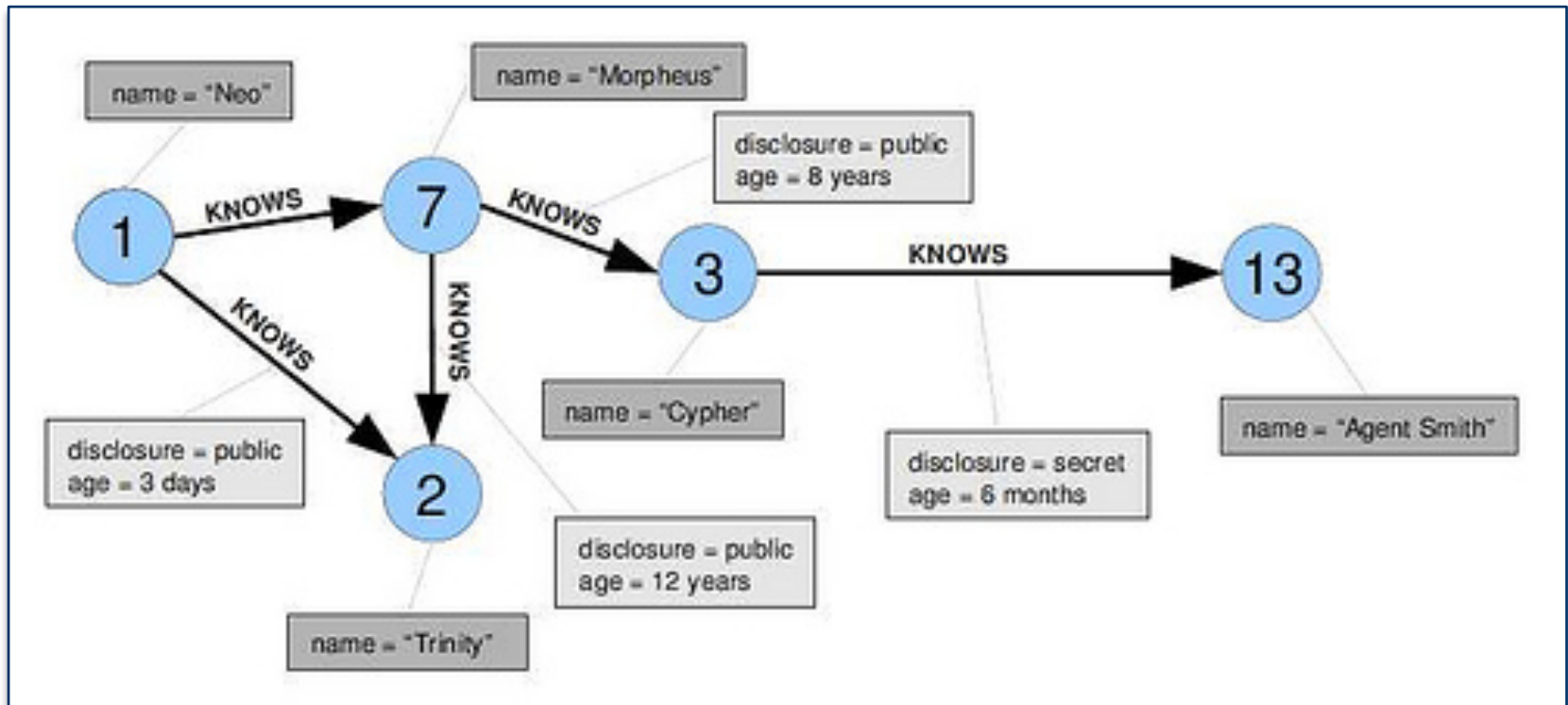
- The reification mechanism is quite powerful
- It appears misplaced in a simple language like RDF
- Making statements about statements introduces a level of complexity that is not necessary for a basic layer of the Semantic Web
- Instead, it would have appeared more natural to include it in more powerful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

# Conclusion



# 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