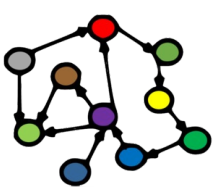
Welcome to INFO216:
Knowledge Graphs
Spring 2022

Andreas L Opdahl
<Andreas.Opdahl@uib.no>

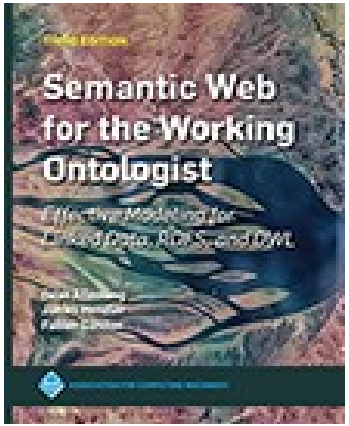
Session 2: Representing KGs (RDF)

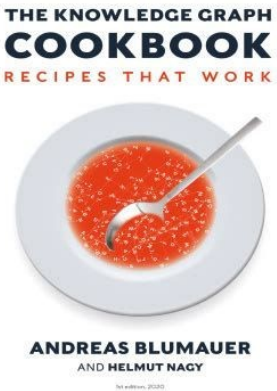
- Themes:
  - Resource Description Framework (RDF)
    - a normal form for semantic data
    - a central semantic standard
  - RDFLib's basic API
    - creating and deleting graphs, input/output, listing statements, managing literals, type mappings



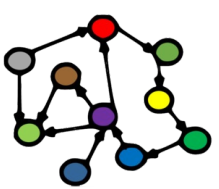
Reading

- Sources:
  - Allemang, Hendler & Gandon (2020): Semantic Web for the Working Ontologist, 3rd edition chapter 3
  - Blumauer & Nagy (2020): Knowledge Graph Cookbook – Recipes that Work (for example pages 92-100, 125-128)
  - materials in the wiki: *wiki.uib.no/info216*
    - RDF Primer
    - rdflib documentation



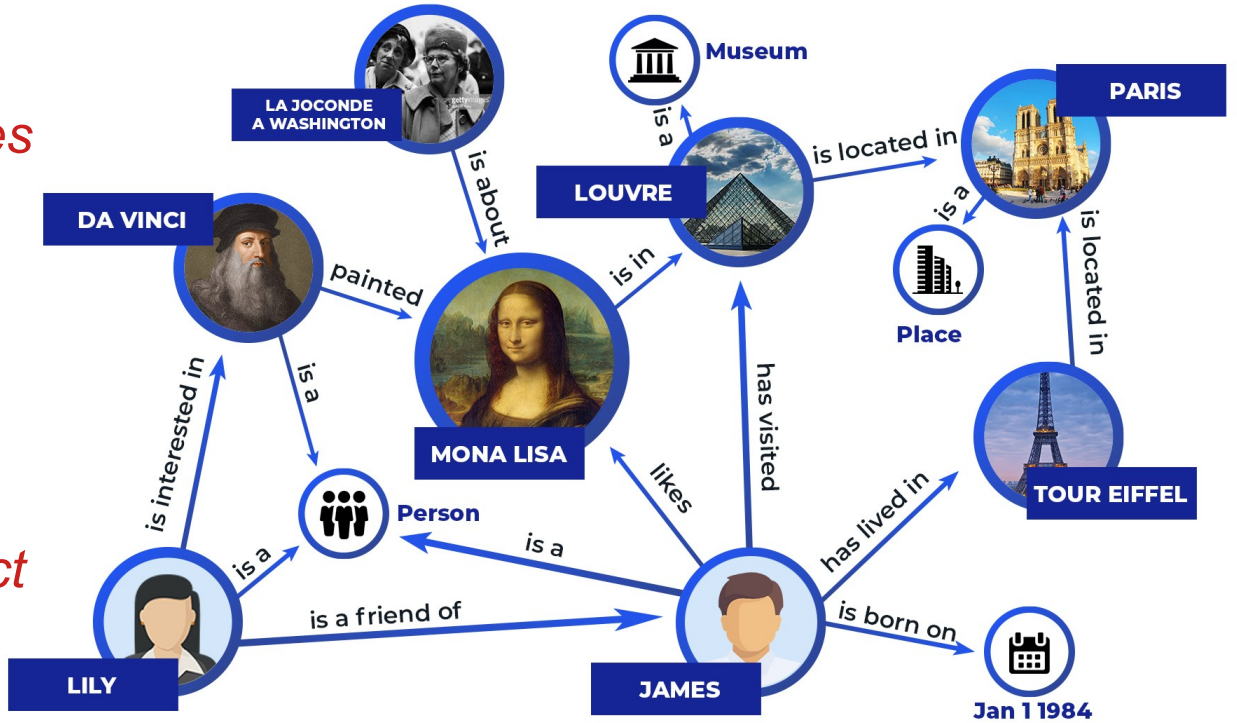


Resource Description Framework (RDF)



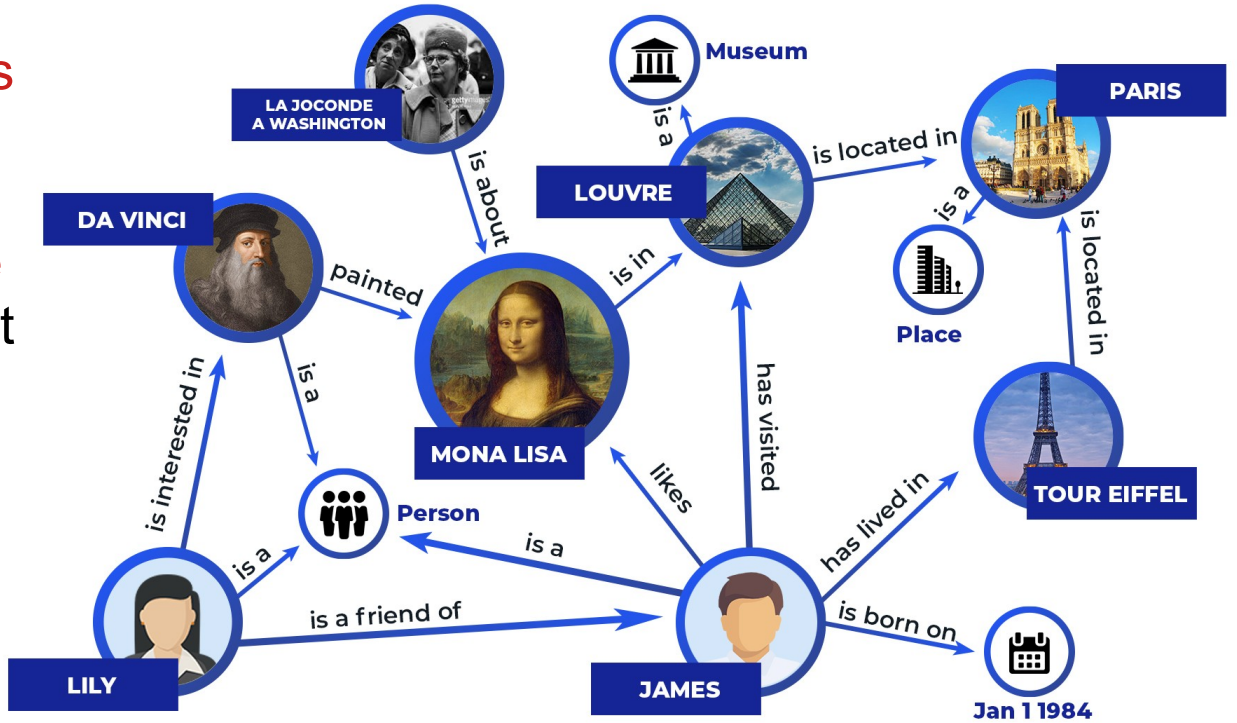
Knowledge graph

- A *graph* of *nodes* connected by directed *edges*
- Nodes can represent *resources* or *values*
- Edges represent *relations*
- Each node–edge–node *triple* represents a *fact*
  - *subject–predicate–object*
  - *head–relation–tail*
- A *knowledge graph* represents *knowledge* as connected *facts*



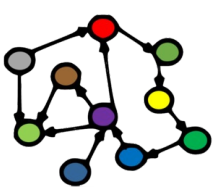
Knowledge graph → semantic knowledge graph

- Through standard identifiers for resources, relations, and types supported by formal definitions, inference and reasoning, KGs attempt to capture (some of) the meaning of data
- The result is semantic knowledge graphs
- In addition to the primary data, semantic KGs contain semantic metadata



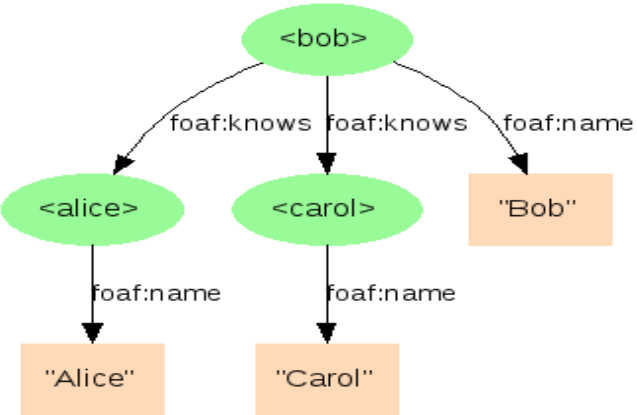
How can we represent semantic KGs?

- Semantic knowledge graphs rely heavily on the *Resource Description Framework (RDF)*
  - a normal form for semantic data (data with associated metadata about its meaning)
  - usable both for the data and their metadata
  - both are represented as KGs
  - either *native/reified*, *embedded, or virtual*
- More expressive vocabularies are available as KGs
  - more types and relations and more formal definitions
  - *RDF Schema (RDFS), "RDFS Plus"*
  - *Web Ontology Language (OWL)*
  - *they all* (can be said to) *build on RDF*



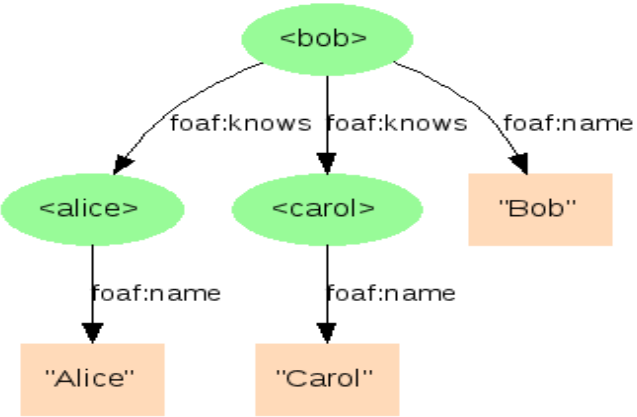
How can we represent semantic KGs?

- RDF models (KGs) consist of statements (triples)
  - of *subject predicate object* .
  - or *subject predicate literal* .
- The subject:
  - must be a *resource*
  - physical, informational, conceptual...
- The predicate:
  - must be a *property* (subtype of resources)
- The object:
  - either a *resource*
  - or a *value* (string, number... – *not* resources)



Uniform Resource Identifiers (URIs) identify resources, including types and relations

- Triples of *subject predicate object .*
  - ...or *subject predicate literal* .
  - Uniform Resource Identifiers (URIs)
  - serialisations, e.g., in *Turtle*:

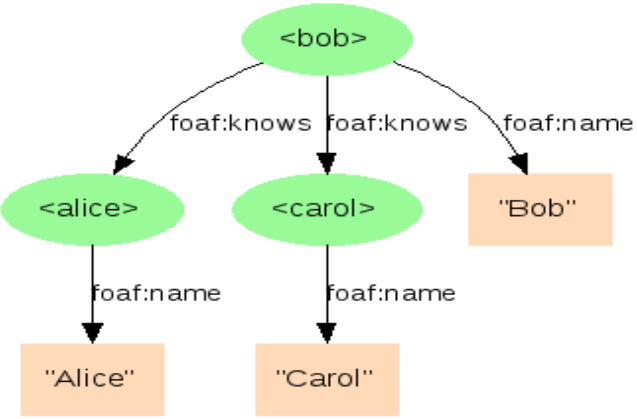


Uniform Resource Identifiers (URIs) identify resources, including types and relations

```
:bob    rdf:type     foaf:Person .
:bob    foaf:name    "Bob" .
:bob    foaf:mbox    <mailto:alice@example.org> .
:bob    foaf:knows   :alice .
:bob    foaf:knows   :carol .
```

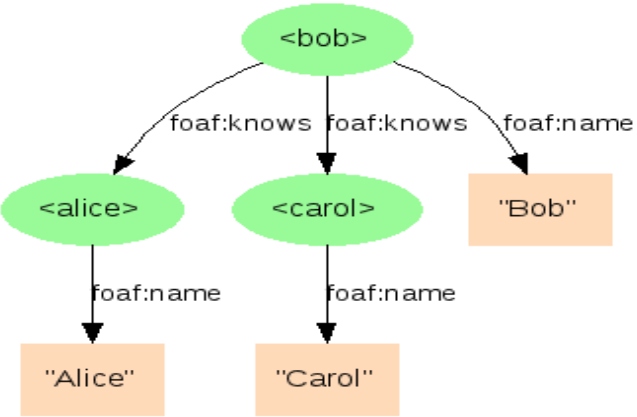
- Triples of *subject predicate object .*
  - ...or of *subject predicate literal* .
  - Uniform Resource Identifiers (URIs)
  - serialisations, e.g., in *Turtle*:

```
@prefix : <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

:bob    rdf:type     foaf:Person .
:bob    foaf:name    "Bob" .
:bob    foaf:mbox    <mailto:alice@example.org> .
:bob    foaf:knows   :alice .
:bob    foaf:knows   :carol .
```



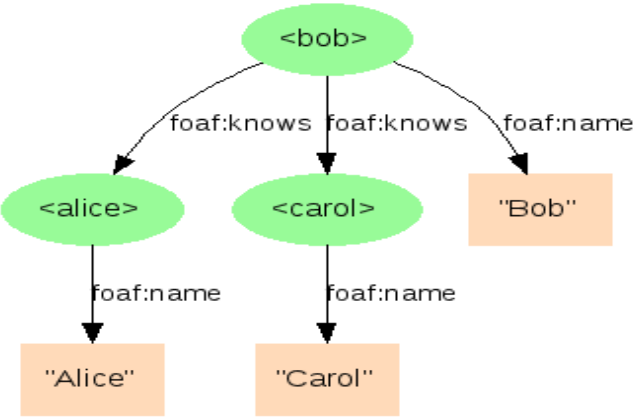
Uniform Resource Identifiers (URIs) identify resources, including types and relations

- Triples of *subject predicate object .*
  - ...or of *subject predicate literal* .
  - Uniform Resource Identifiers (URIs)
  - serialisations, e.g., in *Turtle*:



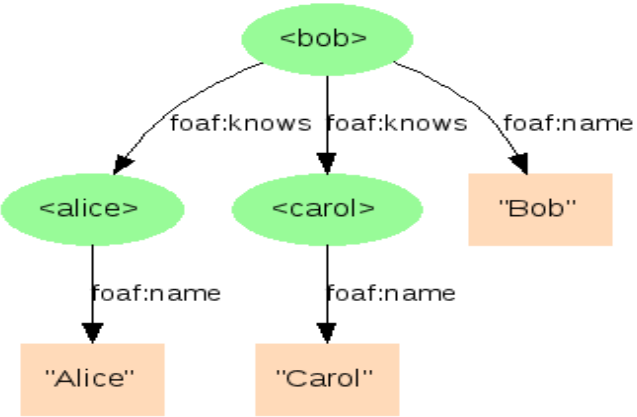

```
@prefix : <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

:bob    rdf:type      foaf:Person ;
        foaf:name     "Bob" ;
        foaf:mbox     <mailto:alice@example.org> ;
        foaf:knows    :alice ;
        foaf:knows    :carol .
```

- Triples of *subject predicate object .*
  - ...or of *subject predicate literal* .
  - Uniform Resource Identifiers (URIs)
  - serialisations, e.g., in *Turtle*::




```
@prefix : <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

:bob    rdf:type      foaf:Person ;
        foaf:name     "Bob" ;
        foaf:mbox     <mailto:alice@example.org> ;
        foaf:knows    :alice ,
                      :carol .
```

- Triples of *subject predicate object .*
  - ...or of *subject predicate literal* .
  - Uniform Resource Identifiers (URIs)
  - serialisations, e.g., in *Turtle*:



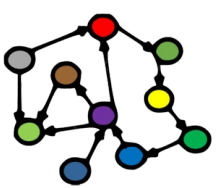

```
@prefix : <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

:bob    a           foaf:Person ;
        foaf:name   "Bob" ;
        foaf:mbox   <mailto:alice@example.org> ;
        foaf:knows  :alice ,
                    :carol .
```

Prefixing

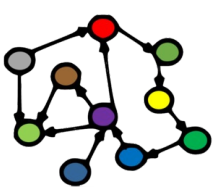
- XML Qualified Name (QName):
  - from “eXtensible Markup Language” (XML)
  - provides short forms for much used URI bases
- Much used prefixes (here in Turtle syntax):

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix ex: <http://www.example.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

  - ...or self-defined prefixes
  - see http://*prefix.cc*
- Example: *http://www.w3.org/2001/XMLSchema#string* can be written with a prefix as: *xsd:string*

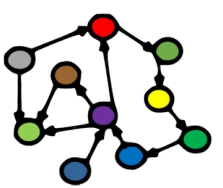


Programming RDF
(and RDFS, SPARQL...)
with Python



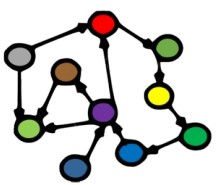
RDFLib

- A library and API (Application Programming Interface) for programming RDF and SPARQL in Python
  - simple, powerful and *pythonic*
  - parsers and serialisers for most RDF formats
  - a *Graph* interface
  - with multiple alternative *Stores*
  - SPARQL 1.1 Query and Update



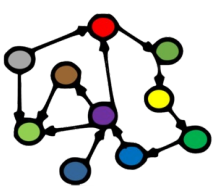
RDFLib

- A library and API (Application Programming Interface) for programming RDF and SPARQL in Python
  - simple, powerful and *pythonic*
  - parsers and serialisers for most RDF formats
  - a *Graph* interface
  - with multiple alternative *Stores*
  - SPARQL 1.1 Query and Update
- More APIs and tools later:
  - a triple store (RDF database): *Blazegraph*
  - an API for accessing triples stores: *SPARQLWrapper*
  - a tool for OWL ontologies: *Protegé-OWL*
  - an OWL library for Python: most likely *owlready2*



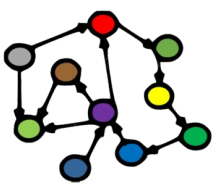
RDFLib interfaces

- Graph:
  - an RDF model
  - a Python collection (set) of triples
  - adding, removing, listing, and searching for triples
  - combine with other graphs
  - writing to and reading from RDF files
  - responding to SPARQL queries and updates
  - backed by an in-memory or persistent store



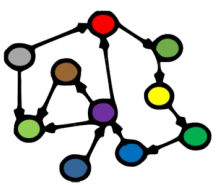
RDFLib interfaces

- Triples / statements:
  - ordinary 3-item Python tuples
    - immutable sequences
    - >>> triple = (s, p, o) # creates a triple
    - >>> s[0] # returns the subject
- URIRef:
  - a node with a URI (represents resources, types, relations)
- Literal:
  - a typed or untyped value
  - untyped values (strings) can be language-tagged
- BNode:
  - a blank node (a resource without a URI)



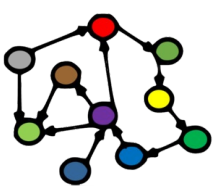
Creating models and statements

- import rdflib
- Creation:
  - Graph: g = rdflib.Graph()
  - Resource: res = rdflib.URIRef(*resURIstr*)
  - Property: prop = rdflib.URIRef(*propURIstr*)
  - Literal: lit = rdflib.Literal*(pythonValue)*
- Add/remove triple:
  - g.add((res, prop, lit))
  - g.remove((res, prop, lit))
- Close persisted model:
  - g.close()



Serialising and parsing

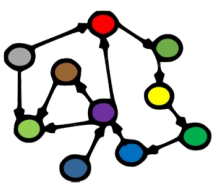
- Serialising:
  - g.serialize(destination=*fileNameStr,* format='ttl')
  - ttl_str = g.serialize(format='ttl').decode()
  - ttl_str = g.serialize(format='json-ld').decode()
    - requires: pip install rdflib_jsonld
- Parsing:
  - g.parse(location=*fileNameStr,* format='ttl')
  - g.parse(source=*webURLStr*, format='ttl')
  - g.parse(data=*pythonStr,* format='ttl')



Listing statements

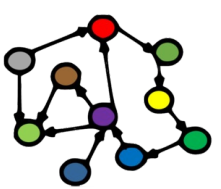
- Retrieving statements (triples):
  - for triple in g:
    do_something(triple)
    - s = triple[0], p = triple[1], o = triple[2]
  - for s, p, o in g:
    do_something(s, p, o)
  - for s, p, o in g.triples((sub, pred, obj)):
    do_something(s, p, o)
    - sub, pred, obj can be None
  - for s, p, o in g[sub : pred : obj]:
    do_something(s, p, o)
    - sub, pred, obj can be empty: s, p, o must match

Python overloading!



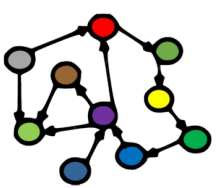
Selecting statements

- Convenience methods:
  - for s in g.subjects(p, o):
    do_something(s)
  - for p, o in g.predicates_objects(s):
    do_something(p, o)
  - if the result is known to be unique:
    - o1 = g.value(s, p)
    - g.set((s, p, o2))

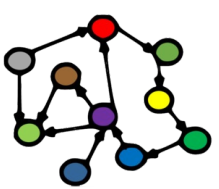


RDFLib interfaces

- Namespaces:
  - predefined:
    - RDF, RDFS, OWL, XSD, FOAF, SKOS, DC, DCTERMS
    - rdflib.namespace.RDF.type
    - or… from rdflib.namespace import RDF
  - user-defined:
    - >>> i2s = rdflib.Namespace('http://i2s.uib.no/')
      >>> i2s.MainAuthor
      rdflib.term.URIRef(u'http://i2s.uib.no/MainAuthor')
  - add prefix to graph:
    - >>> g.bind('i2s', i2s)

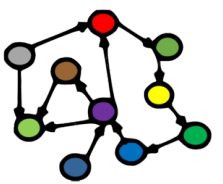


Resources, properties, and literals



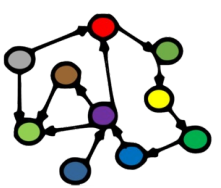
Resources

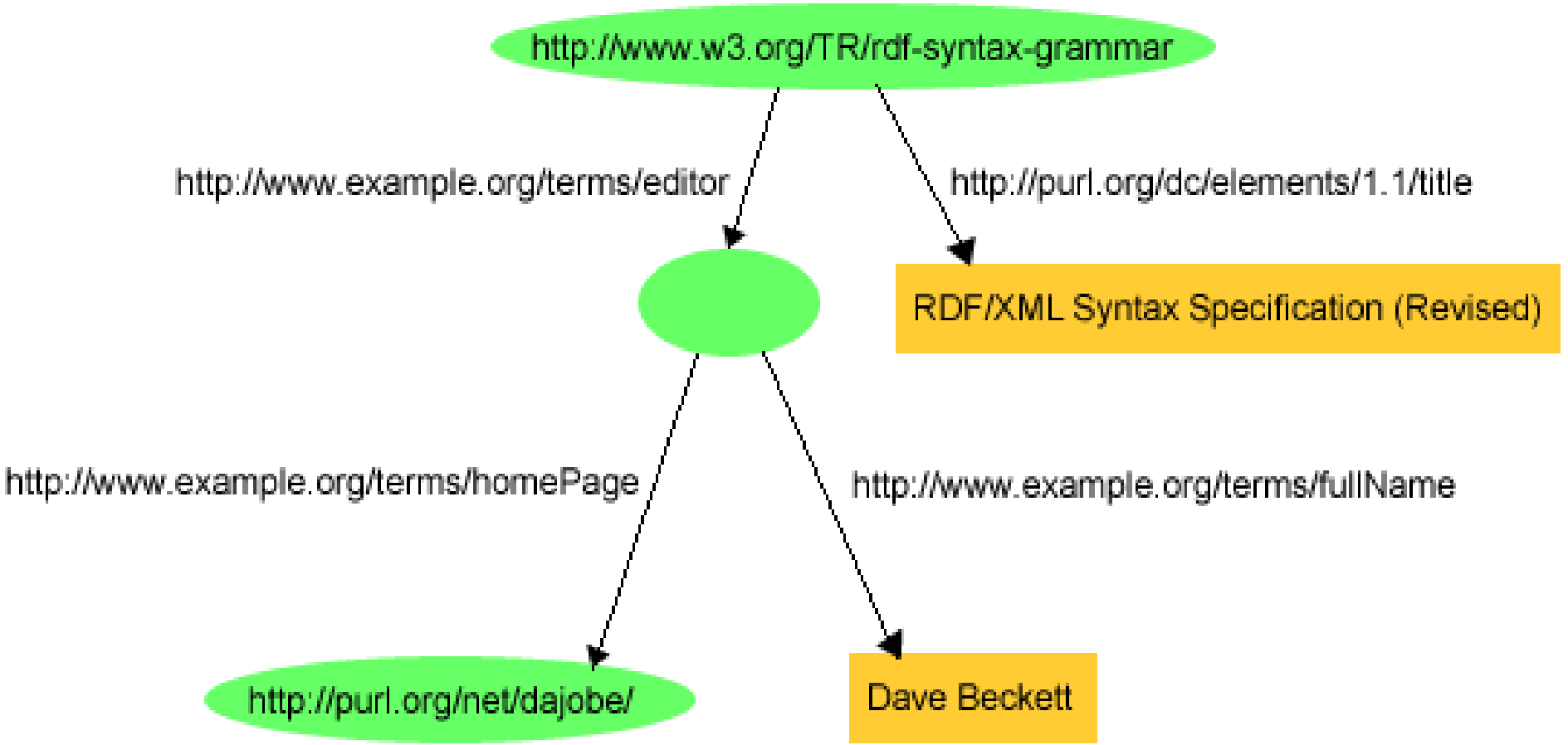
- Resources may be physical phenomena (including people and artefacts), information resources, concepts, constructs...
  - can be most things, really, as well as information about them
  - can be the *subject* or *object* in a statement
    - but only rdf:Property can be *predicate*
  - can be *named* by an URI or *anonymous* (a blank *node*)
- Resources can have one or more *rdf:type*-s
  - dbpedia:Magnus_Carlsen rdf:type dbpedia:ChessPlayer .
- Every resource has the rdf:type *rdfs:Resource*
  - dbpedia:Magnus_Carlsen rdf:type rdfs:Resource .
  - dbpedia:ChessPlayer rdf:type rdfs:Resource .
- *Convention: resource names start with a capital letter*



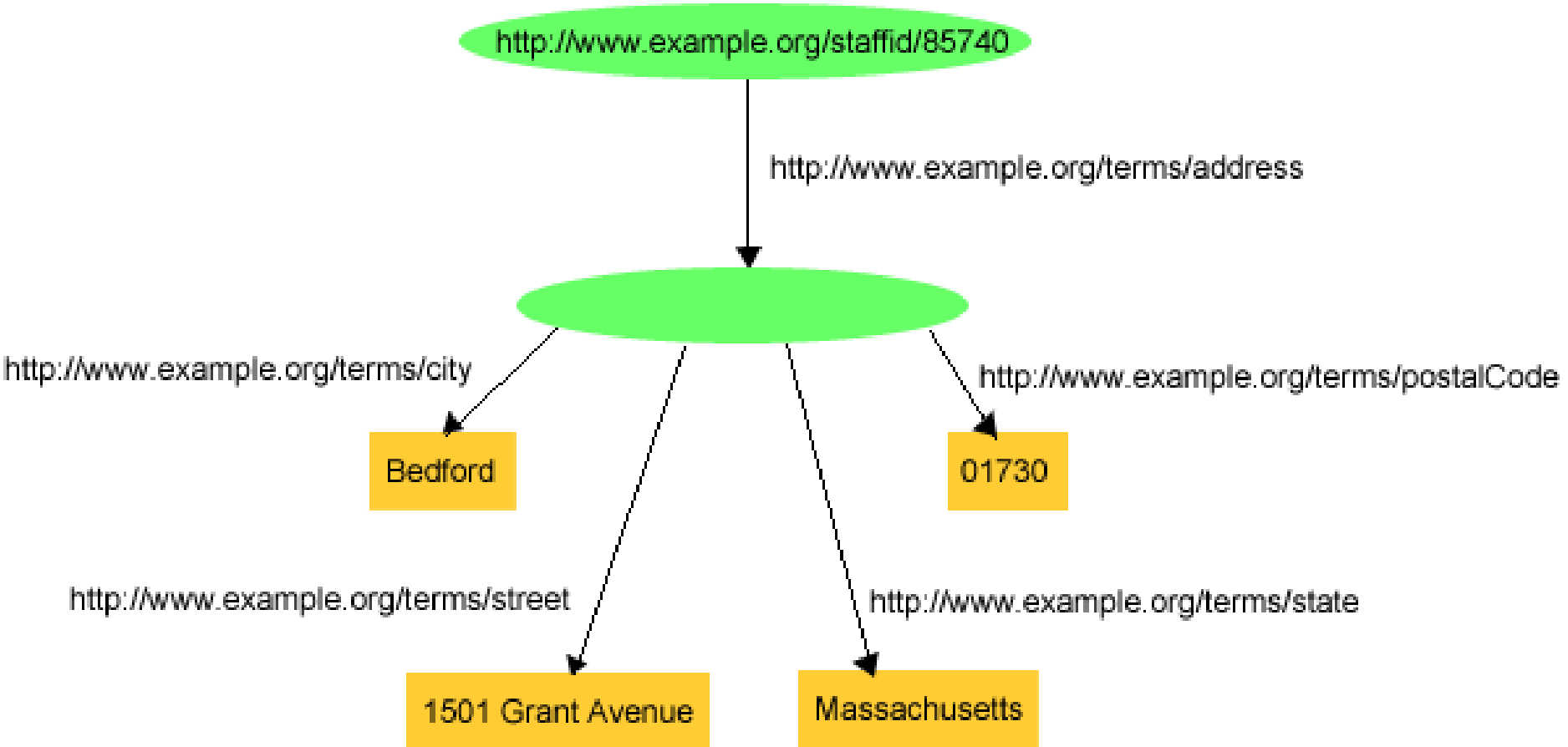
Anonymous resources (blank nodes)

- Some nodes have no URIs:
  - cannot be referenced from the outside
  - *can* have a (non-URI) identifier used inside the graph
- Uses:
  - you are not sure (yet) which URI to use
  - you do not want to reveal the URI to others (privacy, competitors...)
  - to group properties that are related
- *Anonymous resources cause some problems*
  - they are not supported by all RDF technologies
  - *minting* of new URIs is an alternative





From http://www.w3.org/TR/rdf-primer/

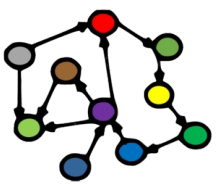


From http://www.w3.org/TR/rdf-primer/

Turtle syntax for blank nodes

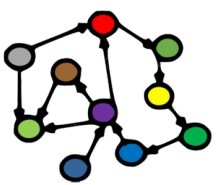
<http://www.w3.org/TR/rdf-syntax-grammar>
    <http://purl.org/dc/elements/1.1/title>
        "RDF/XML Syntax Specification (Revised)" .
<http://www.w3.org/TR/rdf-syntax-grammar>
    <http://www.example.org/terms/editor>
        [] .
[]
    <http://www.example.org/terms/homePage>
        <http://purl.org/net/dajobe> .
[]
    <http://www.example.org/terms/fullName>
        "Dave Beckett" .

Each represents a *different* anon. node...



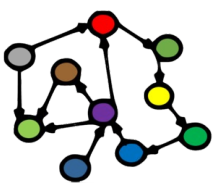
Turtle syntax for blank nodes

```
<http://www.w3.org/TR/rdf-syntax-grammar>
        <http://purl.org/dc/elements/1.1/title>
                "RDF/XML Syntax Specification (Revised)" .
<http://www.w3.org/TR/rdf-syntax-grammar>
        <http://www.example.org/terms/editor>
                _:blank1 .
_:blank1
        <http://www.example.org/terms/homePage>
                <http://purl.org/net/dajobe> .
_:blank1
        <http://www.example.org/terms/fullName>
                "Dave Beckett" .
```



Turtle syntax for blank nodes

```
<http://www.w3.org/TR/rdf-syntax-grammar>
        <http://purl.org/dc/elements/1.1/title>
                "RDF/XML Syntax Specification (Revised)" ;
        <http://www.example.org/terms/editor>
                _:blank1 .
_:blank1
        <http://www.example.org/terms/homePage>
                <http://purl.org/net/dajobe> ;
        <http://www.example.org/terms/fullName>
                "Dave Beckett" .
```



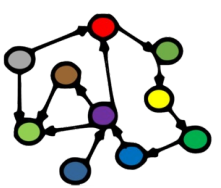
Turtle syntax for blank nodes

<http://www.w3.org/TR/rdf-syntax-grammar>
    <http://purl.org/dc/elements/1.1/title>
        "RDF/XML Syntax Specification (Revised)" ;
    <http://www.example.org/terms/editor>
        [] .

[]

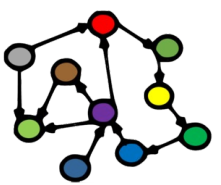
Each represents a *different* anon. node...

    <http://www.example.org/terms/homePage>
        <http://purl.org/net/dajobe> ;
    <http://www.example.org/terms/fullName>
        "Dave Beckett" .



Turtle syntax for blank nodes

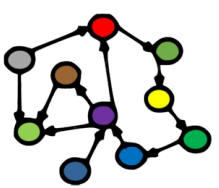
```
<http://www.w3.org/TR/rdf-syntax-grammar>
        <http://purl.org/dc/elements/1.1/title>
                "RDF/XML Syntax Specification (Revised)" ;
        <http://www.example.org/terms/editor>
                [
                        <http://www.example.org/terms/homePage>
                                <http://purl.org/net/dajobe> ;
                        <http://www.example.org/terms/fullName>
                                "Dave Beckett"
                ] .
```



Properties

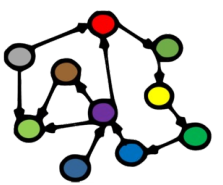
- Properties are resources that
  - express a relationship between resources
  - ...or between resources and literal values
- Mostly used as a *predicate*
  - but can *sometimes* be a subject or object
  - example:
    - *dc:name* is a property in the Dublin Core vocabulary
    - it can also be the subject in RDF statements:

      dc:name rdf:type rdf:Property .
- *Convention: property names start with lower-case letters*



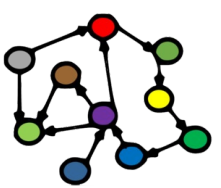
Literals

- Untyped (simple) literals: only a character string
  - f eks “29”
  - *strings can have a language code!*
    “Göteborg”@”se”, “Gothenburg”@”en”
- Typed literals: a string + an URI (ref)
  - the type is defined of the URI
  - XML Schema Definition (XSD) language is common
  - two built-in RDF types: rdf:XMLLiteral, rdf:HTML
  - ...but other types can also be used
- Syntax depends on the serialisation, e.g., TURTLE:
  - "29"^^<http://www.w3.org/2001/XMLSchema#integer>
  - or with a prefix: "29"^^<xsd:integer>



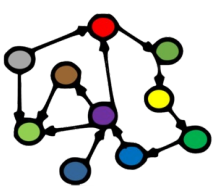
XML Schema Definition (XSD) types

- Most XSD types can be used in RDF:
  xsd:string, xsd:boolean, xsd:decimal, xsd:integer, xsd:float, xsd:double, xsd:dateTime, xsd:dateTimeStamp, xsd:time, xsd:date, xsd:gYearMonth, xsd:gYear, xsd:gMonthDay, xsd:gDay, xsd:gMonth, xsd:duration, xsd:yearMonthDuration, xsd:dayTimeDuration, xsd:hexBinary, xsd:base64Binary, xsd:anyURI, xsd:normalizedString, xsd:token, xsd:language, xsd:NMTOKEN, xsd:Name, xsd:NCName, xsd:positiveInteger, xsd:nonPositiveInteger, xsd:negativeInteger, xsd:long, xsd:int, xsd:short, xsd:byte, xsd:nonNegativeInteger, xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedShort, xsd:unsignedByte
- Not all XML Schema types can be used in RDF:
  - *must be a set of string values*
  - *...that can be mapped into*
  - *...a well-defined value space*



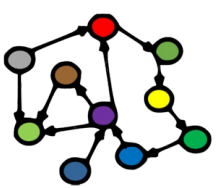
XML Schema Definition (XSD) types

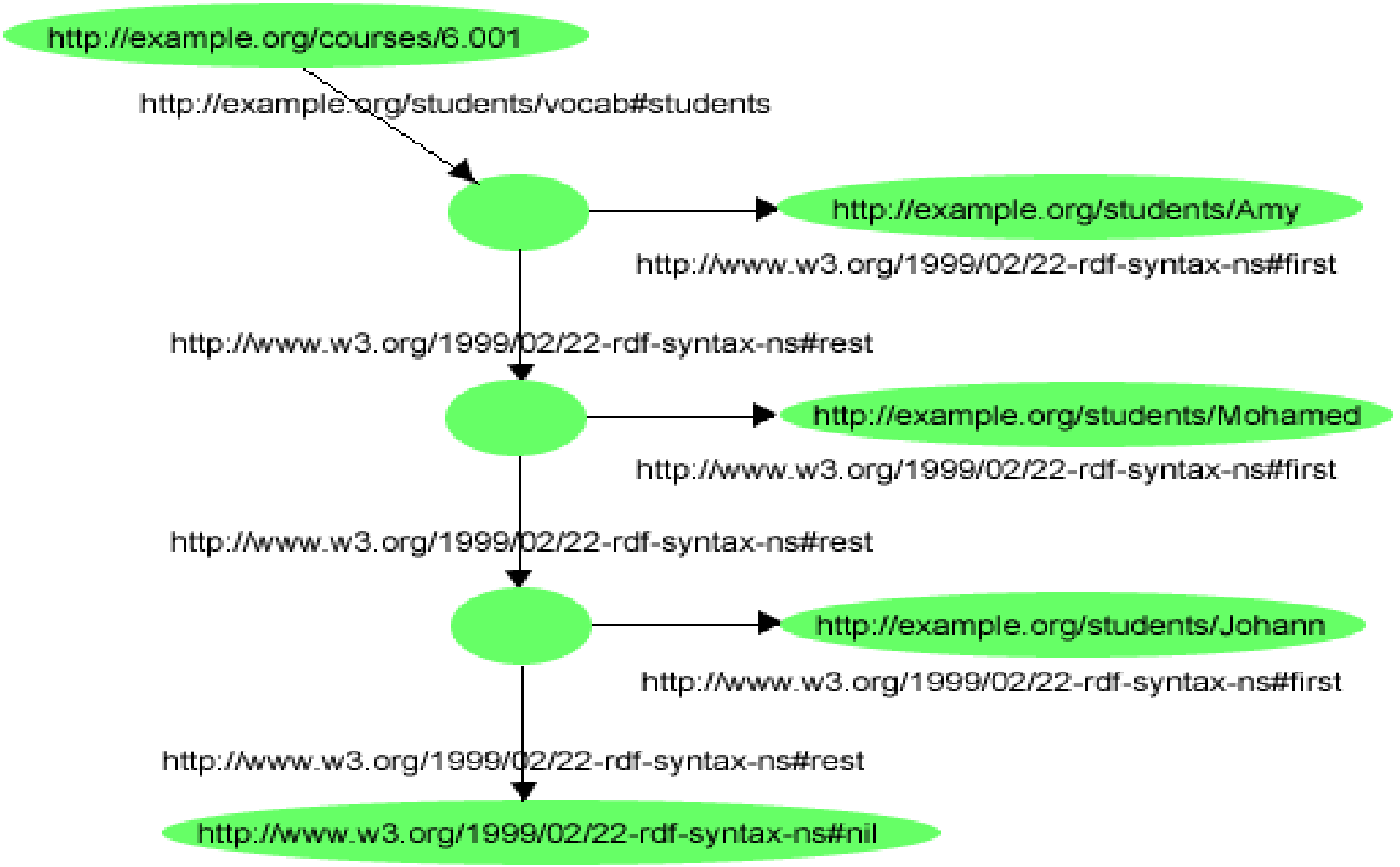
- Most XSD types can be used in RDF:
  xsd:string, xsd:boolean, xsd:decimal, xsd:integer, xsd:float, xsd:double, xsd:dateTime, xsd:dateTimeStamp, xsd:time, xsd:date, xsd:gYearMonth, xsd:gYear, xsd:gMonthDay, xsd:gDay, xsd:gMonth, xsd:duration, xsd:yearMonthDuration, xsd:dayTimeDuration, xsd:hexBinary, xsd:base64Binary, xsd:anyURI, xsd:normalizedString, xsd:token, xsd:language, xsd:NMTOKEN, xsd:Name, xsd:NCName, xsd:positiveInteger, xsd:nonPositiveInteger, xsd:negativeInteger, xsd:long, xsd:int, xsd:short, xsd:byte, xsd:nonNegativeInteger, xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedShort, xsd:unsignedByte
- Not all XML Schema types can be used in RDF:
  - *must be a set of string values*
  - *...that can be mapped into*
  - *...a well-defined value space*



Collections

- Containers are not closed
  - we *cannot assume it only has the members we know of*
  - others can add more members to the list *without deleting triples* (i.e., *monotonically*)
- Collections (with rdf:type rdf:List):
  - can only have the listed members
  - rdf:first gives the first RDF node in the list
  - rdf:rest gives the rest of the list
  - rdf:nil is an empty list

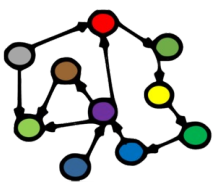


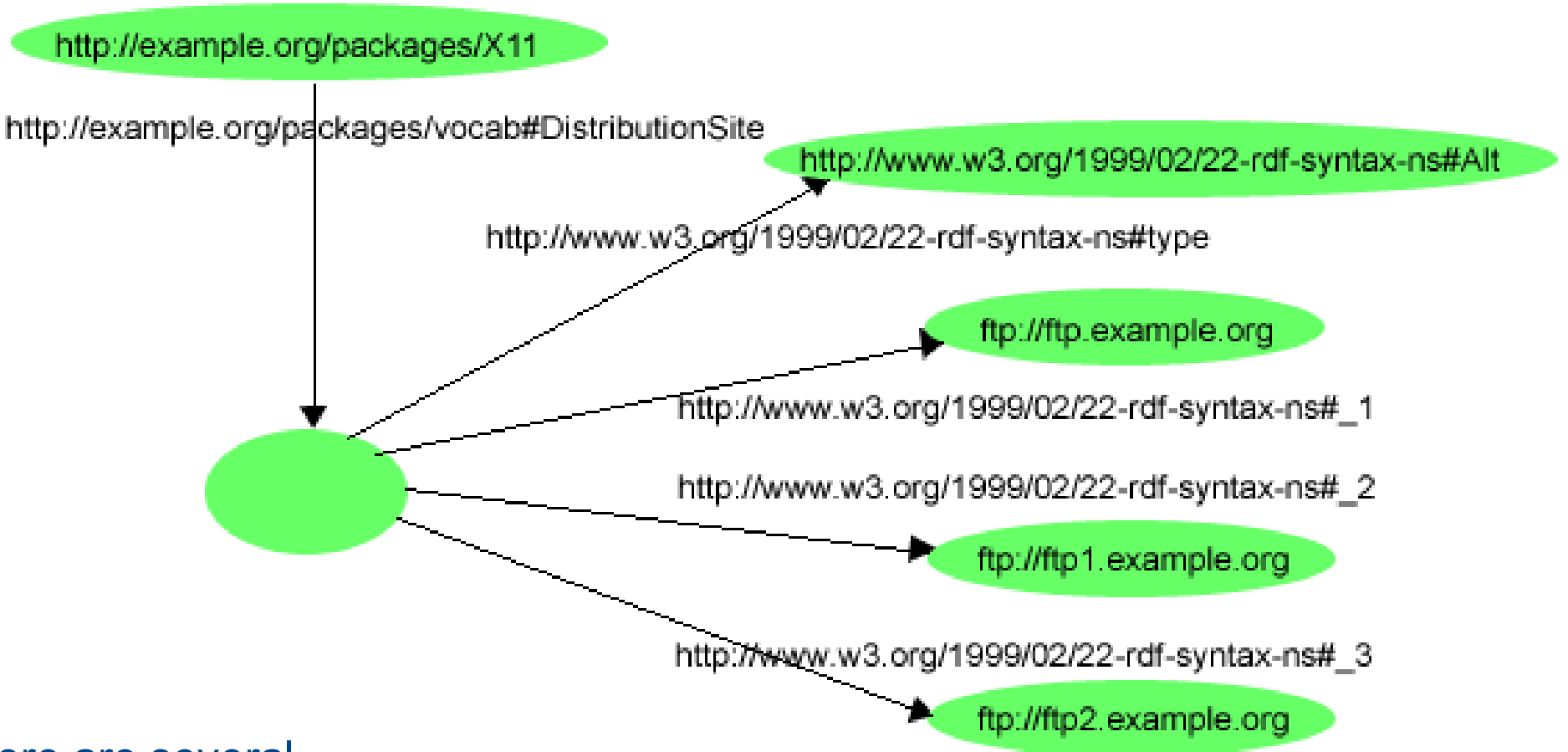


From http://www.w3.org/TR/rdf-primer/

Containers

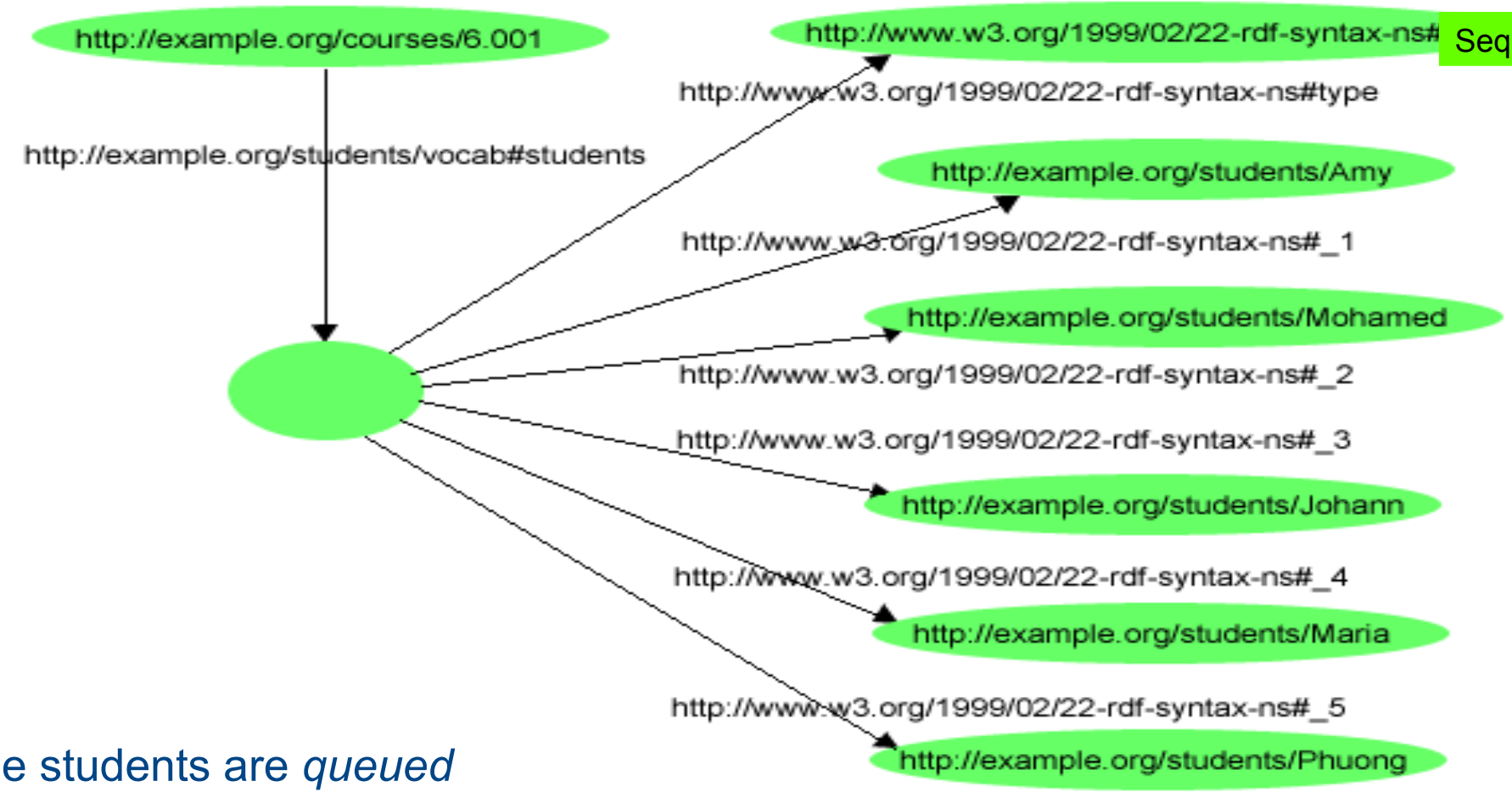
- Containers can be used when a subject is related in the same way to many RDF nodes that are
  - ordered and/or duplicated
  - have rdf:type rdfs:Container
- Container nodes are often anonymous (can have an URI)
  - have RDF nodes as members (rdfs:member)
  - have special properties rdf:_1, rdf:_2 etc. to pick out particular members
- rdf:Alt – several alternative resources
- rdf:Seq – lists of RDF nodes, can have duplicates
- rdf:Bag – orderless RDF nodes, can have duplicates





There are several
alternative distribution sites.

From http://www.w3.org/TR/rdf-primer/



The students are *queued up* in order for the course.

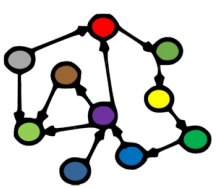
From http://www.w3.org/TR/rdf-primer/

Other knowledge graph formats

Other types of knowledge graphs

- *Labelled Property Graphs (LPG)*
  - becoming increasingly popular
  - not inherently semantic/linked
  - but can be used semantically, e.g., to store RDF
  - has so far not been standardised:
    - different tools use different query languages, exchange formats
    - standardisation is moving quickly forward

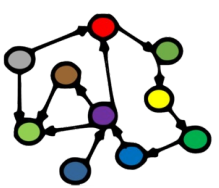
- Our focus remains on *RDF-based knowledge graphs*:
  - what we call *semantic knowledge graphs*



Other types of knowledge graphs

- *Non-semantic knowledge graphs*
  - many recent ML approaches use graph data
  - e.g., graph embeddngs, link prediction
  - but the graphs are not necessarily *dereferenced*
    - they can use human-understandable labels
    - but they do not use standard URI
  - but can be used semantically too, e.g., on RDF data

- Our focus remains on *RDF-based knowledge graphs*:
  - what we call *semantic knowledge graphs*



Next week:
Querying and updating KGs (SPARQL)