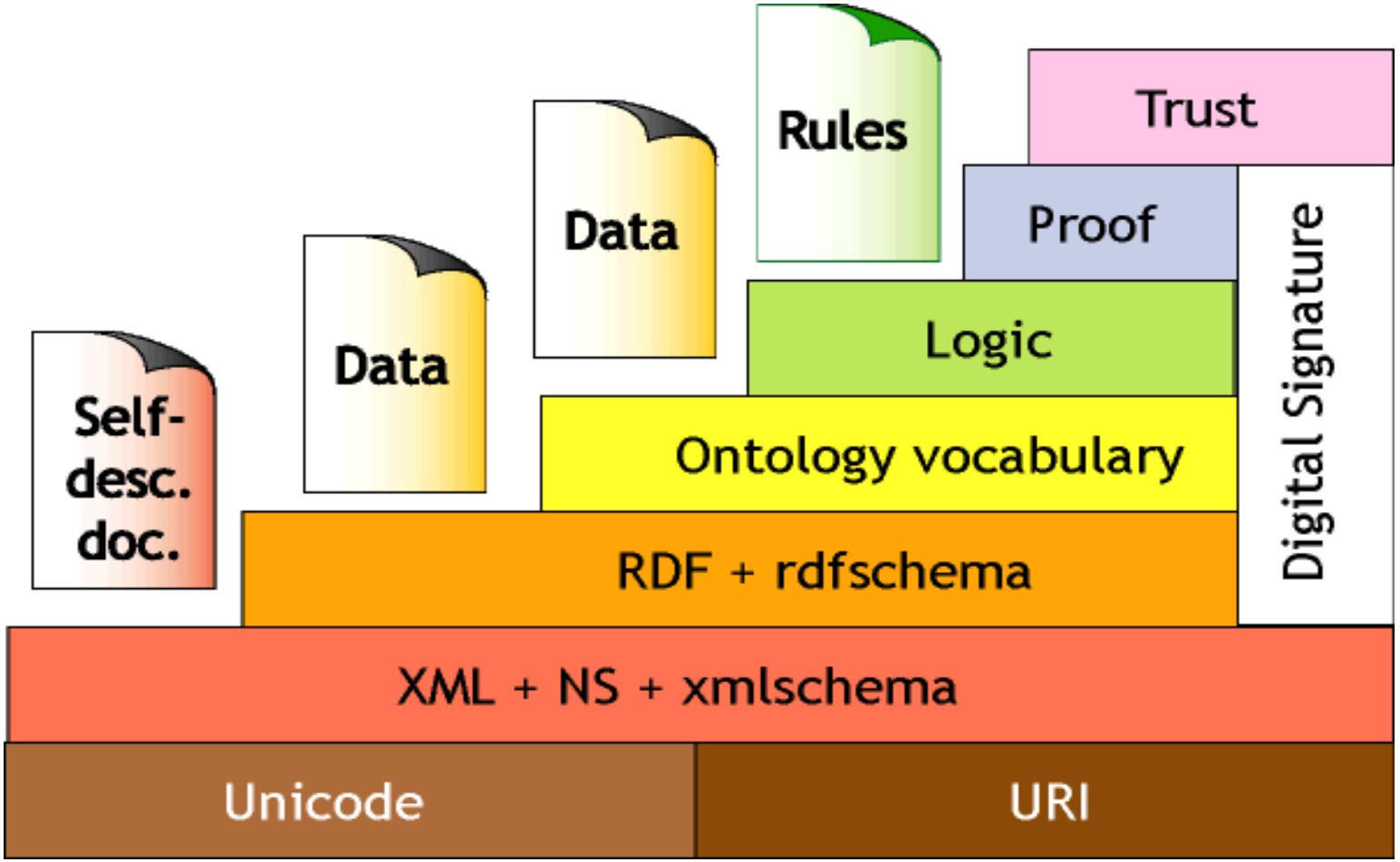
Knowledge Representation and Semantic Technologies

The RDF layer

Riccardo Rosati

Corso di Laurea Magistrale in Ingegneria Informatica
Sapienza Università di Roma
2020/2021

The Semantic Web Tower



Syntax and semantics

- Syntax: the structure of data
- Semantics: the meaning of data
- Two conditions necessary for interoperability:
  - Adopt a **common syntax**: this enables applications to *parse* the data.
  - Adopt a **means for understanding** the semantics: this enables applications to *use* the data.

XML

- XML: eXtensible Mark-up Language
- XML documents are written through a user-defined set of tags
- tags are used to express the “semantics” of the various pieces of information

XML: example

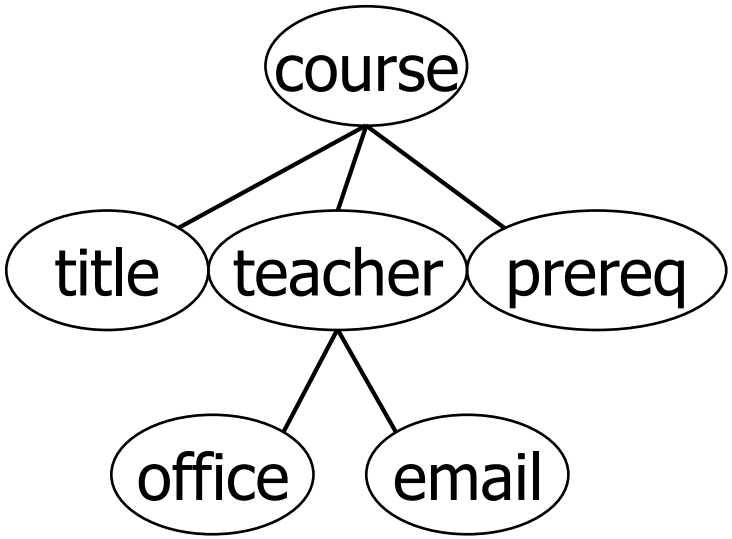
```
<course date="2007">
  <title>Seminari di Ingegneria del Software
  </title>
  <teacher>
    <name>Giuseppe De Giacomo</name>
    <email>degiacomo@dis.uniroma1.it</email>
  </teacher>
  <prereq>none</prereq>
</course>
```

XML

- XML: document = labelled tree
- node = label + attributes/values + contents

```
<course date="...">
  <title>...</title>
  <teacher>
    <office>...</office>
    <email>...</email>
  </teacher>
<prereq>...</prereq>
</course>
```

=



XML

- XML Schema = grammar for describing legal trees and datatypes
- can we use XML to represent semantics?

XML and semantics

```
<Predator>
        …
</Predator>
```

- Predator: a medium-altitude, long-endurance unmanned aerial vehicle system.
- Predator : one that victimizes, plunders, or destroys, especially for one's own gain.
- Predator : an organism that lives by preying on other organisms.
- ...

Limitations for semantic markup

- XML makes no commitment on:
  1. Domain-specific ontological vocabulary
  2. Ontological modeling primitives
- Requires pre-arranged agreement on 1 and 2
- Only feasible for closed collaboration:
  - agents in a small & stable community
  - pages on a small & stable intranet
- Not suited for sharing Web-resources

RDF

- RDF is a data model
  - the model is domain-neutral, application-neutral and ready for internationalization
  - the model can be viewed as directed, labeled graphs or as an object-oriented model (object/attribute/value)
- RDF data model is an abstract, conceptual layer independent of XML
  - consequently, XML is a transfer syntax for RDF, not a component of RDF
  - RDF data might never occur in XML form

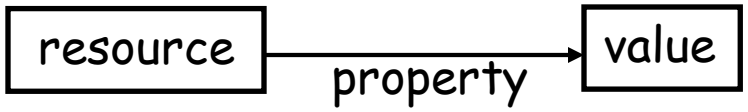
RDF model

RDF model = set of RDF triples

triple = expression (statement)

(subject, predicate, object)

- subject = resource
- predicate = property (of the resource)
- object = value (of the property)

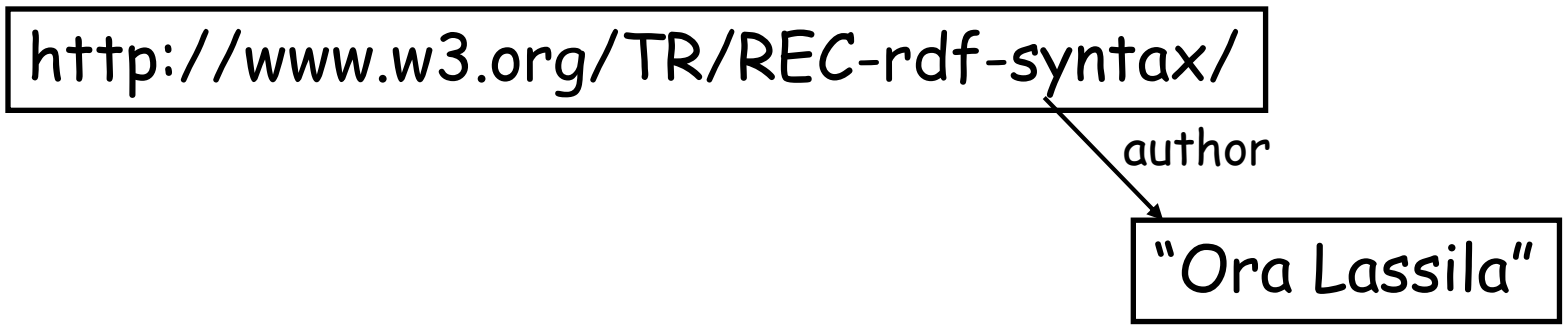


RDF triples

example: "the document at
`http://www.w3c.org/TR/REC-rdf-syntax`
has the author Ora Lassila"

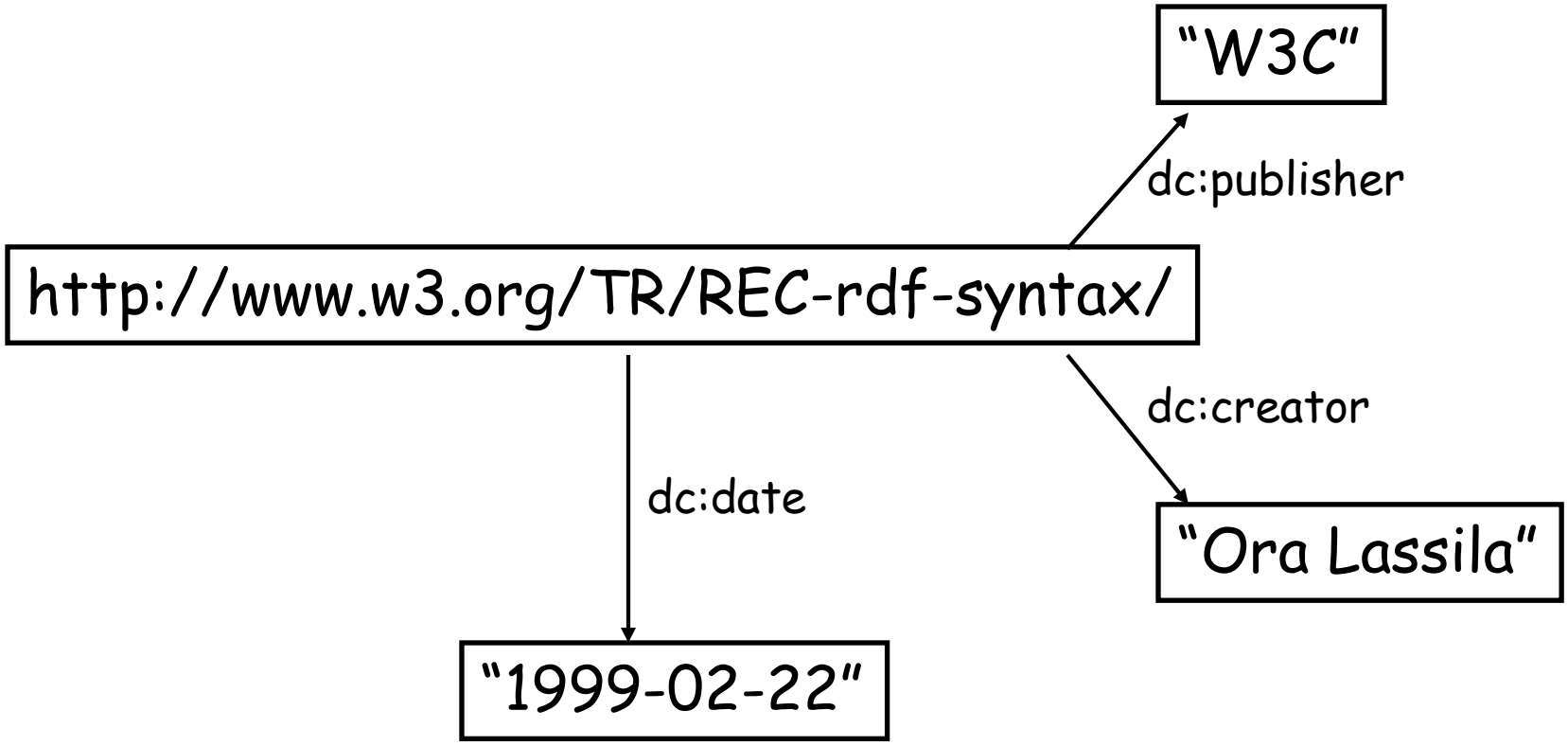
triple:

`http://www.w3c.org/TR/REC-rdf-syntax author "OraLassila"`



⇒ RDF model = **graph**

RDF graph: example



Node and edge labels in RDF graphs

node and edge labels:

- **URI**
- **literal** (string)
- **blank node** (anonymous label)

but:

- a literal can only appear in object positions
- a blank node can only appear in subject or object positions

- remark: URIs are used as predicates, i.e., graph nodes can be used as edge labels (RDF has meta-modeling abilities)

Blank nodes

- **blank node** (bnode) = RDF graph node with "anonymous label" (i.e., not associated with an URI)
- example:
- "Jean has a friend born the 21st of April"

```
ex:Jean   foaf:knows       _:p1
_:p1      foaf:birthDate   04-21
```

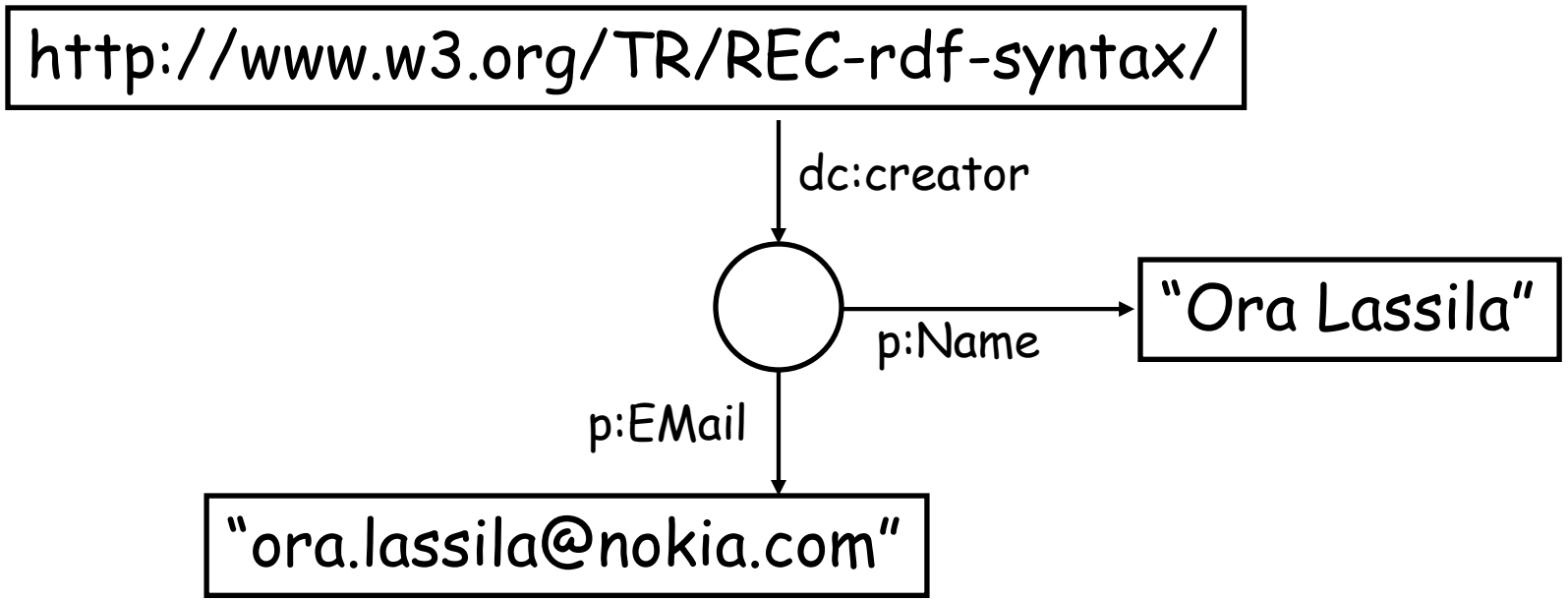
  - `_:p1` is the blank node (b-node)
- bnodes can be used both as subjects and objects

Complex values

- values of properties need not be simple strings
- the value of a property can also be a graph node (corresponding to a resource)
- using blank nodes, arbitrarily complex tree and graph structures are possible

Complex values

example:

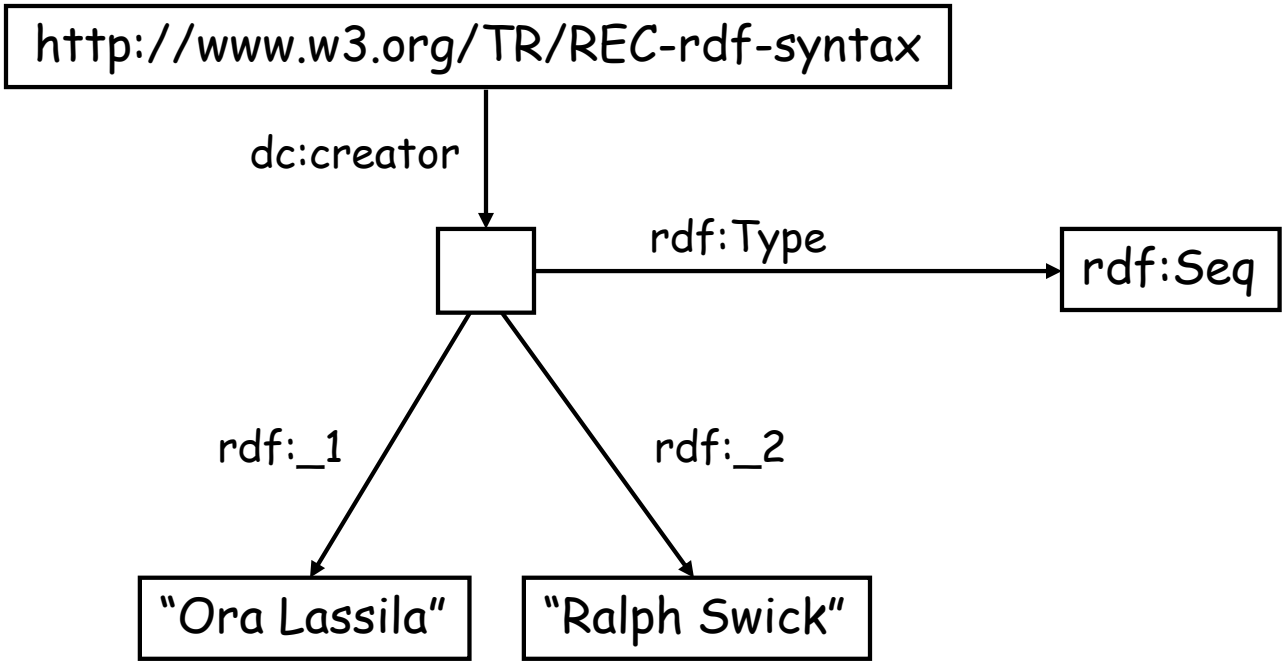


Containers

- Containers are collections
  - they allow grouping of resources (or literal values)
- It is possible to make statements about the container (as a whole) or about its members individually
- Different types of containers exist
  - **bag -** unordered collection
  - **seq -** ordered collection (= “sequence”)
  - **alt -** represents alternatives
- It is also possible to create collections based on URI patterns
- Duplicate values are permitted (no mechanism to enforce unique value constraints)

Containers

example:



Higher-order statements

- One can make RDF statements about other RDF statements
  - example: “Ralph believes that the web contains one billion documents”
- Higher-order statements
  - allow us to express beliefs (and other modalities)
  - are important for trust models, digital signatures,etc.
  - also: metadata about metadata
  - are represented by modeling RDF in RDF itself

$\Rightarrow$ reification

Reification

Reification in RDF = using an RDF statement as the subject (or object) of another RDF statement

Examples of statement that need reification to be expressed in RDF:

- “the New York Times claims that Joe is the author of book ABC”
- “the statement “The technical report on RDF was written by Ora Lassila” was written by the Library of Congress”

Reification

- RDF provides a built-in predicate vocabulary for reification:
  - **rdf:subject**
  - **rdf:predicate**
  - **rdf:object**
  - **rdf:statement**
- Using this vocabulary (i.e., these URIs from the rdf: namespace) it is possible to represents a triple through a blank node

Reification: example

- the statement “The technical report on RDF was written by Ora Lassila” can be represented by the following four triples:

```
_:x rdf:predicate dc:creator.
_:x rdf:subject http://www.w3.org/TR/REC-rdf-syntax.
_:x rdf:object “Ora Lassila”.
_:x rdf:type rdf:statement.
```

- The blank node _:x is the **reification** of the statement (it is an anonymous URI that represents the whole triple)
- Now, “The statement “The technical report on RDF was written by Ora Lassila” was written by the Library of Congress” can be represented using the bnode _:x, by adding to the above four triples the following triple:

```
_:x dc:creator “Library of Congress”.
```

RDF syntaxes

RDF model = edge-labeled graph = set of triples

- graphical notation (graph)
- (informal) triple-based notation

  e.g., **`(subject, predicate, object)`**
- formal syntaxes:
  - N3 notation
  - Turtle notation
  - concrete (serialized) syntax: RDF/XML syntax

RDF syntaxes

- N3 notation:

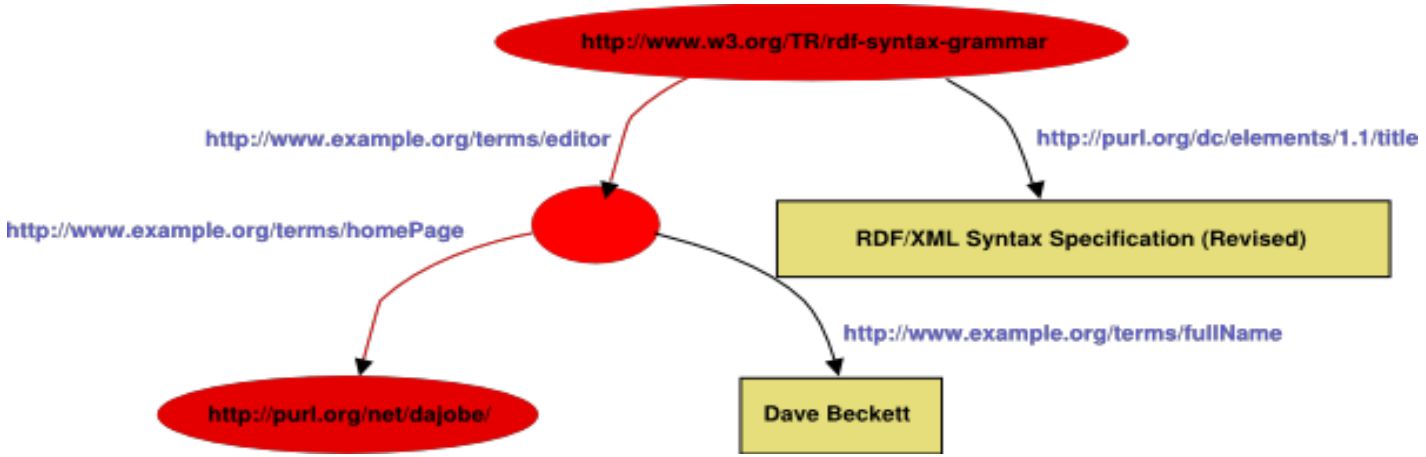
```
subject predicate object .
```

- Turtle notation: example:

```
@prefix
    rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
:mary rdf:type <http://www.ex.org/Gardener>.
:mary :worksFor :ElJardinHaus.
:mary :name "Dalileh Jones"@en.
_:john :worksFor :ElJardinHas.
_:john :idNumber "54321"^^xsd:integer.
```

- concrete (serialized) syntax: RDF/XML syntax

RDF/XML syntax: Example




```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description rdf:about="http://purl.org/net/dajobe/">
        </rdf:Description>
      </ex:homePage>
    </rdf:Description>
  </ex:editor>
</rdf:Description>
```

RDF/XML syntax: Example (2)

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:exterms="http://www.example.com/terms/">
  <rdf:Description rdf:ID="item10245">
    <exterms:model rdf:datatype="&xsd;string">Overnighter</exterms:model>
    <exterms:sleeps rdf:datatype="&xsd;integer">2</exterms:sleeps>
    <exterms:weight rdf:datatype="&xsd;decimal">2.4</exterms:weight>
    <exterms:packedSize rdf:datatype="&xsd;integer">784</exterms:packedSize>
  </rdf:Description>
</rdf:RDF>
```

RDFa: RDF in XHTML documents

- RDFa = RDF in attributes
- Provides a simple way for embedding RDF semantic annotations into XHTML documents
- More precisely: RDFa specifies a set of attributes that can be used in XHTML elements
- Such attributes allow for expressing RDF triples inside the document
- W3C standard (2008)
- No compatibility problem (XHTML browsers correctly display documents with RDFa attributes)

RDFa: Example (1)

consider the following XHTML document:

```
...
<h2>The trouble with Bob</h2>
<h3>Alice</h3>
...
```

and suppose that the above h2 element specifies the title and the h3 element specifies the author of the document.

We can add RDFa attributes to the above code (and use the Dublin Core (DC) vocabulary) to specify this:

```
<div about="http://example.com/twb"
 xmlns:dc="http://purl.org/dc/elements/1.1/">
  <h2 property="dc:title">The trouble with Bob</h2>
  <h3 property="dc:creator">Alice</h3> ...
</div>
```

RDFa: Example (1)

```
<div about="http://example.com/twb"
 xmlns:dc="http://purl.org/dc/elements/1.1/">
  <h2 property="dc:title">The trouble with Bob</h2>
  <h3 property="dc:creator">Alice</h3> ...
</div>
```

- The above use of the `about` and `property` attribute corresponds to defining the following RDF triples:

```
http://example.com/twb dc:title "The trouble with Bob".
http://example.com/twb dc:author "Alice".
```

- The `about` attribute specifies the URI that is the subject
- The `property` attribute specifies the predicate
- The objects are the literals corresponding to the content of the element

RDFa: Example (2)

Consider the following document fragment:

```
<div>
  <p>Alice Birpemswick</p>
  <p>
    Email:
    <a href="mailto:alice@example.com">alice@example.com</a>
  </p>
  <p>
    Phone: <a href="tel:+1-617-555-7332">+1 617.555.7332</a>
  </p>
</div>
```

We want to specify that the above information is contact information about Alice

RDFa: Example (2)

we use RDFa and the Friend-of-a Friend (FOAF) vocabulary:

```
<div typeof="foaf:Person"
     xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <p property="foaf:name">Alice Birpemswick</p>
  <p>
    Email:
    <a rel="foaf:mbox"
       href="mailto:alice@example.com">alice@example.com</a>
  </p>
  <p>
    Phone:
    <a rel="foaf:phone"
       href="tel:+1-617-555-7332">+1 617.555.7332</a>
  </p>
</div>
```

RDFa: Example (2)

The above use of the `about` and `property` attribute corresponds to defining the following RDF triples:

```
_:x rdf:type foaf:person
_:x foaf:name “Alice Birpemswick”
_:x foaf:mbox mailto:alice@example.com
_:x foaf:phone tel:+1-617-555-7332
```

Notice: the subject of the above triples is a blank node because we did not assign the `about` attribute to the div element

RDF Schema

RDFS = RDF Schema

- Defines small **vocabulary** for RDF:
  - Class, subClassOf, type
  - Property, subPropertyOf
  - domain, range
- correspond to a set of RDF predicates:

  $\Rightarrow$ meta-level

  $\Rightarrow$special (predefined) “meaning”

RDFS

- vocabulary for defining **classes** and **properties**
- vocabulary for classes:
  - **`rdfs:Class`** (a resource is a class)
  - **`rdf:type`** (a resource is an instance of a class)
  - **`rdfs:subClassOf`** (a resource is a subclass of another resource)

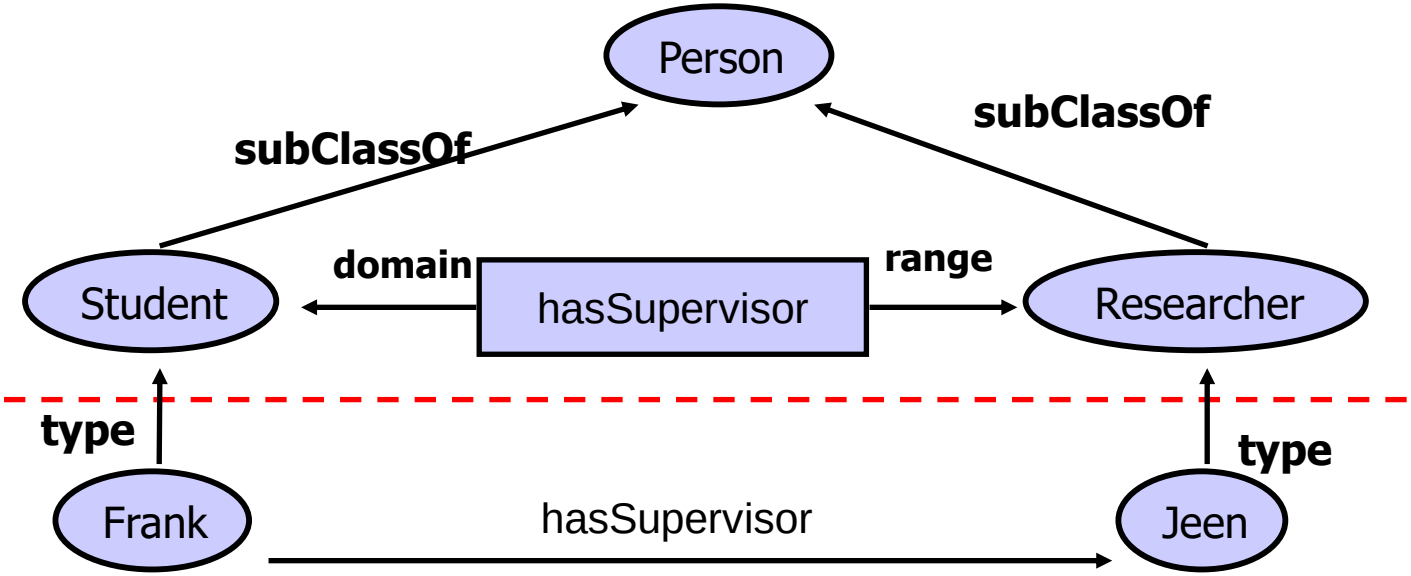
RDFS

vocabulary for properties:

- **`rdf:Property`** (a resource is a property)
- **`rdfs:domain`** (denotes the first component of a property)
- **`rdfs:range`** (denotes the second component of a property)
- **`rdfs:subPropertyOf`** (expresses ISA between properties)

RDFS

example:



RDFS: XML syntax

example:

```
<rdf:Description ID="MotorVehicle">
    <rdf:type resource="http://www.w3.org/...#Class"/>
    <rdfs:subClassOf
    rdf:resource="http://www.w3.org/...#Resource"/>
</rdf:Description>

<rdf:Description ID="Truck">
    <rdf:type resource="http://www.w3.org/...#Class"/>
    <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>
```

RDFS: XML syntax

example (cont.):

```
<rdf:Description ID="registeredTo">
    <rdf:type resource="http://www.w3.org/...#Property"/>
    <rdfs:domain rdf:resource="#MotorVehicle"/>
    <rdfs:range rdf:resource="#Person"/>
</rdf:Description>

<rdf:Description ID=”ownedBy">
    <rdf:type resource="http://www.w3.org/...#Property"/>
    <rdfs:subPropertyOf rdf:resource="#registeredTo"/>
</rdf:Description>
```

RDF + RDFS: semantics

- what is the exact meaning of an RDF(S) graph?
- initially, a formal semantics was not defined!
- main problems:
  - bnodes
  - meta-modeling
  - formal semantics for RDFS vocabulary
- In 2004, a model-theoretic semantics was provided

$\Rightarrow$formal definition of entailment and query answering over RDF(S) graphs

Incomplete information in RDF graphs

- bnodes = existential values (null values)

$\Rightarrow$ introduce incomplete information in RDF graphs

- an RDF graph can be seen as an incomplete database represented in the form of a naive table
- an RDF graph is represented by a unique table T, with values being constants or named existential variables
- an RDF graph can be seen as a boolean conjunctive query over the unique relation T

Formal semantics for RDF + RDFS

- formal semantics for RDFS vocabulary
- RDFS statements = **constraints** over the RDF graph
- entailment in RDF + RDFS = reasoning (query answering) over an **incomplete database with constraints**

RDFS: meta-modeling abilities

example (meta-classes):

```
(ex:MotorVehicle, rdf:type, rdfs:Class)
(ex:myClasses, rdf:type, rdfs:Class)
(ex:MotorVehicle, rdf:type,  ex:myClasses)
```

RDF + RDFS: semantics

problems with meta-data:

```
:a rdf:type :C .
:C rdf:type :R .
:R rdf:type :a .
```

or

```
:C rdf:type :C .
```

are correct (formally meaningful) RDF statements

$\Rightarrow$ but no intuitive semantics

Querying RDF: SPARQL

Simple Protocol And RDF Query Language

- W3C standardisation effort similar to the XQuery query language for XML data
- Data Access Working Group (DAWG)
- suitable for remote use (remote access protocol)

SPARQL

example:

```
PREFIX
   abc: <http://mynamespace.com/exampleOntology#>
SELECT ?capital ?country
WHERE { ?x abc:cityname ?capital.
        ?y abc:countryname ?country.
        ?x abc:isCapitalOf ?y.
        ?y abc:isInContinent abc:africa. }
```

SPARQL

```
PREFIX
   abc: <http://mynamespace.com/exampleOntology#>
SELECT ?capital ?country
WHERE { ?x abc:cityname ?capital.
        ?y abc:countryname ?country.
        ?x abc:isCapitalOf ?y.
        ?y abc:isInContinent abc:africa. }
```

- Variables are outlined through the "?" prefix ("$" is also possible).
- The ?capital and the ?country will be returned.
- The SPARQL query processor returns all hits matching the pattern of the four RDF-triples.
- "property orientation" (class matches can be conducted solely through class-attributes/properties)

SPARQL

- **basic graph pattern** (BGP) query = RDF graph with possibly variables as labels
- SPARQL defines on top of a BGP additional operators (AND, FILTER, UNION, OPTIONAL)

SPARQL query: example

RDF graph:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
 _:a foaf:name "Johnny Lee Outlaw" .
 _:a foaf:mbox <mailto:jlow@example.com> .
_:b foaf:name "Peter Goodguy" .
_:b foaf:mbox <mailto:peter@example.org> .
_:c foaf:mbox <mailto:carol@example.org> .
```

query:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
        ?x foaf:mbox ?mbox }
```

result:

“Johnny Lee Outlaw”	<mailto:jlow@example.com>
“Peter Goodguy”	<mailto:peter@example.com>

Use of filters: example

RDF graph:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
 _:a foaf:name "Johnny Lee Outlaw" .
 _:a foaf:mbox <mailto:jlow@example.com> .
_:b foaf:name "Peter Goodguy" .
_:b foaf:mbox <mailto:peter@example.org> .
_:c foaf:mbox <mailto:carol@example.org> .
```

query:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
        ?x foaf:mbox ?mbox .
        FILTER regex(?name, "^J") }
```

result:

“Johnny Lee Outlaw”	<mailto:jlow@example.com>

Optional patterns: example

RDF graph:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
 _:a foaf:name "Johnny Lee Outlaw" .
 _:a foaf:mbox <mailto:jlow@example.com> .
_:b foaf:name "Peter Goodguy" .
_:b foaf:mbox <mailto:peter@example.org> .
_:c foaf:mbox <mailto:carol@example.org> .
```

query:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:mbox ?mbox .
        OPTIONAL {?x foaf:name ?name } }
```

result:

“Johnny Lee Outlaw”	<mailto:jlow@example.com>
“Peter Goodguy”	<mailto:peter@example.com>
	<mailto:carol@example.org>

Optional patterns: example (2)

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
_:a rdf:type foaf:Person .
_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@example.com> .
_:a foaf:mbox <mailto:alice@work.example> .
_:b rdf:type foaf:Person .
_:b foaf:name "Bob" .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
        OPTIONAL { ?x foaf:mbox ?mbox } }
```

“Alice”	<mailto:alice@example.com>
“Alice”	<mailto:alice@work.example>
“Bob”	

SPARQL 1.1: property paths

- SPARQL 1.1 (released in 2013) extends the first varsion of SPARQL in different ways (e.g., aggregate functions, entailment regimes)
- In particular, it allows for expressing **property paths** in queries
- A property path is essentially a regular expression using properties (URIs)
- Property paths allow for expressing paths of arbitrary length along the RDF graph, thus providing a fundamental graph-oriented feature to SPARQL

Property paths in SPARQL

Syntax Form	Property Path Expr. Name	Matches
iri	PredicatePath	An IRI. A path of length one.
^elt	InversePath	Inverse path (object to subject).
elt1 / elt2	SequencePath	A sequence path of elt1 followed by elt2.
elt1 | elt2	AlternativePath	A alternative path of elt1 or elt2 (all possibilities are tried).
elt*	ZeroOrMorePath	A path that connects the subject and object of the path by zero or more matches of elt.
elt+	OneOrMorePath	A path that connects the subject and object of the path by one or more matches of elt.
elt?	ZeroOrOnePath	A path that connects the subject and object of the path by zero or one matches of elt.
!iri or !(iri_1| ...|iri_n)	NegatedPropertySet	Negated property set. An IRI which is not one of iri_i. !iri is short for !(iri).
!^iri or !(^iri_1| ...|^iri_n)	NegatedPropertySet	Negated property set where the excluded matches are based on reversed path. That is, not one of iri_1...iri_n as reverse paths. !^iri is short for !(^iri).
!(iri_1| ...|iri_j|^iri_{j+1}| ...|^iri_n)	NegatedPropertySet	A combination of forward and reverse properties in a negated property set.
(elt)		A group path elt, brackets control precedence.

Exercise 9: property paths

Express through a SPARQL query the following request:

- Return the names of all the ancestors and the descendants of John, assuming that the RDF graph uses the properties :hasFather and :hasMother to express kinship relations.

Exercise 9: solution

```
PREFIX ex: <http://example.org/example/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?z
WHERE { ex:John (ex:hasFather | ex:hasMother)+ ?x .
        ?x foaf:name ?z . }
UNION
      { ?x (ex:hasFather | ex:hasMother)+ ex:John .
        ?x foaf:name ?z . }
```

Exercise 10: property paths

Express through a SPARQL query the following request:

- Return all the classes which John belongs to.

Exercise 10: solution

```
PREFIX ex: <http://example.org/example/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
   schema#>
SELECT ?z
WHERE { ex:John rdf:type ?c .
        ?c rdfs:subClassOf* ?x . }
```

Or, equivalently:

```
PREFIX ex: <http://example.org/example/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
   schema#>
SELECT ?z
WHERE { ex:John rdf:type/rdfs:subclassOf* ?c . }
```

RDF in the real world

- RDF and SPARQL are W3C standards
- Widespread use for metadata representation, e.g.
  - Apple (MCF)
  - Adobe (XMP)
  - Mozilla/Firefox
- Oracle supports RDF, and provides an extension of SQL to query RDF data
- HP has a big lab (in Bristol) developing specialized data stores for RDF (Jena)

RDF in the real world

- current main application of RDF: **linked data**
- linked data = using the Web to create **typed links** between data from different sources
- i.e.: create a **Web of data**
- DBpedia, Geonames, US Census, EuroStat, MusicBrainz, BBC Programmes, Flickr, DBLP, PubMed, UniProt, FOAF, SIOC, OpenCyc, UMBEL, Virtual Observatories, freebase,…
- each source: up to several million triples
- overall: over 30 billions triples (2012)

Linked Data

Linked Data: set of best practices for publishing and connecting structured data on the Web using URIs and RDF

Basic idea: apply the general architecture of the World Wide Web to the task of sharing structured data on global scale

•The Web is built on the idea of setting hyperlinks between documents that may reside on different Web servers.
•It is built on a small set of simple standards:

- Global identification mechanism: URIs, IRIs
- Access mechanism: HTTP
- Content format: HTML

Linked Data builds directly on Web architecture and applies this architecture to the task of sharing data on global scale

Linked data principles

1. Use **URIs** as names for things.
2. Use HTTP URIs, so that people can look up those names (**dereferenceable URIs**).
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).
4. Include **links** to other URIs, so that they can discover more things.

Dereferenceability = URIs are not just used for identifying entities: since they can be used in the same way as URLs, they also enable locating and retrieving resources describing and representing these entities on the Web.

Linked data principles

Just as hyperlinks in the classic Web connect documents into a single global information space, Linked Data uses hyperlinks to connect disparate data into a single **global data space**.

These links, in turn, enable applications to navigate the data space.

For example, a Linked Data application that has looked up a URI and retrieved RDF data describing a person may follow links from that data to data on different Web servers, describing, for instance, the place where the person lives or the company for which the person works.

Linked Data lifecycle

1. Extraction
2. Storage & Querying
3. Authoring
4. Linking
5. Enrichment
6. Quality Analysis
7. Evolution & Repair
8. Search, Browsing & Exploration

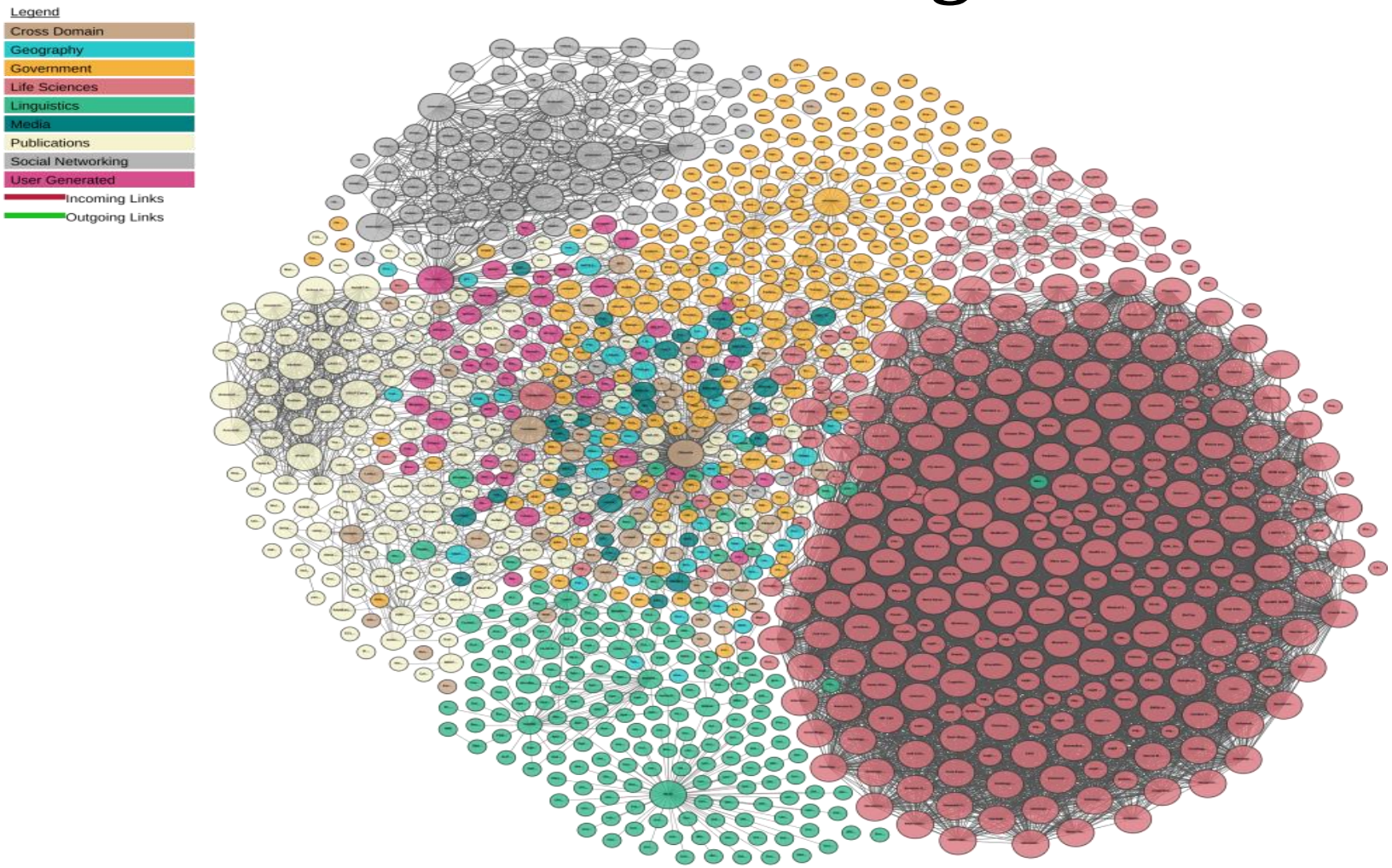
Open/Closed Linked Data

Linked data may be **open** (publicly accessible and reusable) or **closed**

Linking Open Data (LOD): project which aims at creating an open Linked Data network

http://lod-cloud.net

The LOD cloud diagram



Linking Open Data cloud diagram 2017, by Andrejs Abele, John P. McCrae, Paul Buitelaar, Anja Jentzsch and Richard Cyganiak. http://lod-cloud.net/

Use of RDF vocabularies

- Crucial aspect of Linked Data (and of RDF usage in general): which URIs represent predicates (links)?
- Recommended practice in LOD: if possible, **use existing RDF vocabularies** (and preferably the most popular ones)
- In this way, a de-facto standard is created: all LOD sites use the same URI to represent the same property, and the semantics of such properties is shared (i.e., known by every application)
- This makes it possible for all applications to really understand the semantics of links

Popular vocabularies

- **Friend-of-a-Friend (FOAF)**, vocabulary for describing people
- **Dublin Core (DC)** defines general metadata attributes
- **Semantically-Interlinked Online Communities (SIOC)**, vocabulary for representing online communities
- **Description of a Project (DOAP)**, vocabulary for describing projects
- **Simple Knowledge Organization System (SKOS)**, vocabulary for representing taxonomies and loosely structured knowledge
- **Music Ontology** provides terms for describing artists, albums and tracks
- **Review Vocabulary**, vocabulary for representing reviews
- **Creative Commons (CC)**, vocabulary for describing license terms

RDF/SPARQL tools

- **Jena** = Java framework for handling RDF models and SPARQL queries (http://jena.sourceforge.net/)
- **ARC** = PHP implementation of a RDF/SPARQL engine (http://arc.semsol.org/)
- **Virtuoso** = database system able to natively deal with RDF data and SPARQL queries (http://virtuoso.openlinksw.com/)
- …and many more
- see http://esw.w3.org/topic/SparqlImplementations

Back to the “semantic” issue

- does RDF solve the “semantic problem” that XML does not solve?
- no, unless the vocabulary used is **shared**
- e.g., the “meaning” of a predicate (edge) in an RDF graph must be known (shared) by every application
- solutions?
  - “popular” vocabularies/ontologies
    - e.g., FOAF, Dublin Core,…
  - “bottom-up” creation of shared vocabularies
    - tagging (e.g., flickr)
    - microformats