Welcome to INFO216: Knowledge Graphs

Spring 2023

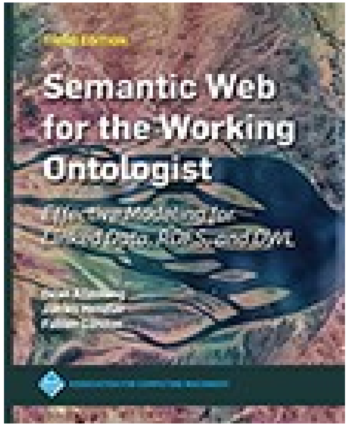
Andreas L Opdahl
<Andreas.Opdahl@uib.no>

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
  - about INFO216
  - a little more *background*
    - what are the *semantic web*, *semantic technologies*, *and linked data?*

Reading

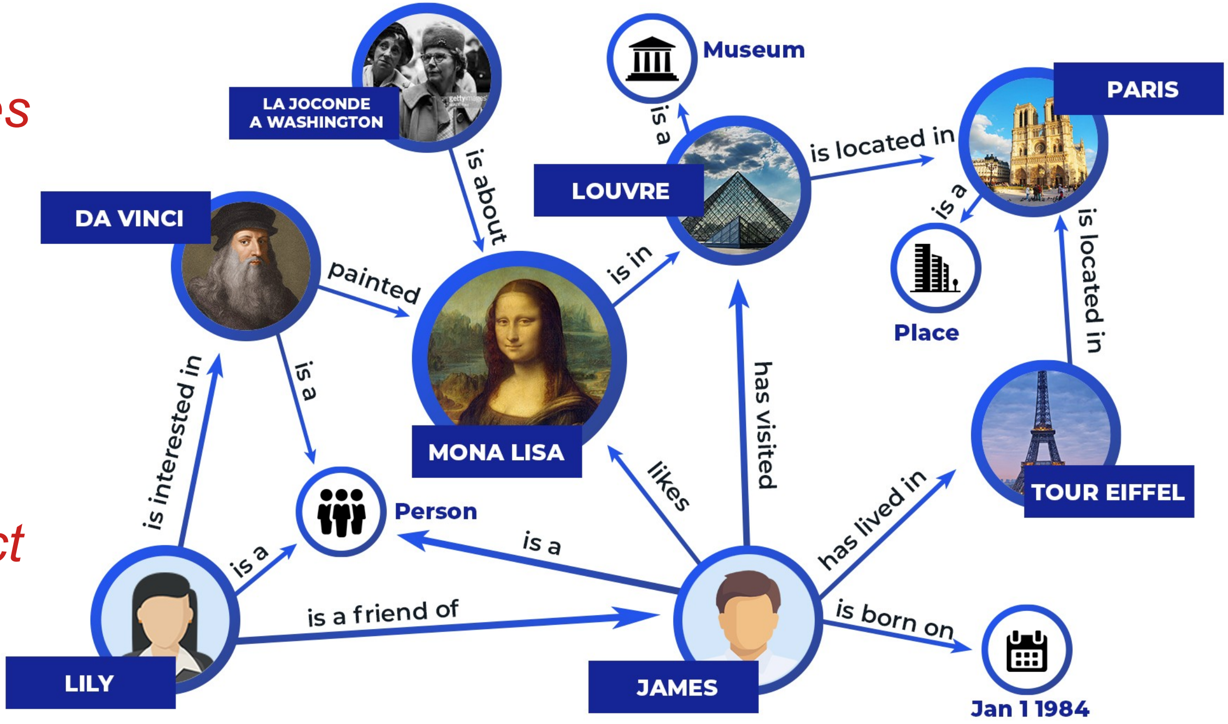
- Sources:
  - Allemang, Hendler & Gandon (2020):
    Semantic Web for the Working Ontologist, 3rd edition:
    chapter 3
  - Blumauer & Nagy (2020):
    Knowledge Graph Cookbook – Recipes that Work:
    for example pages 92-100, 125-128, 164-167 (*supplementary)*
- Materials in the wiki <http://wiki.uib.no/info216>:
  - RDF Primer
  - rdflib documentation



Resource Description Framework (RDF)

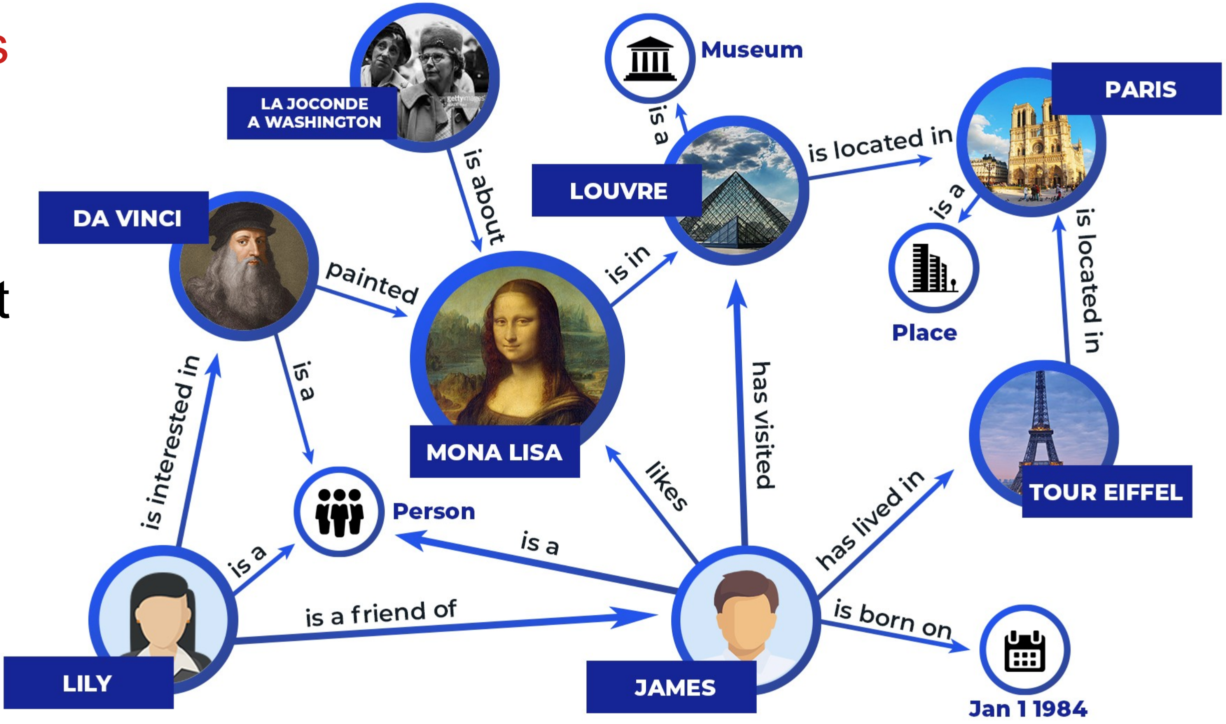
Knowledge graph

- 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 *triples* connected by *nodes*



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

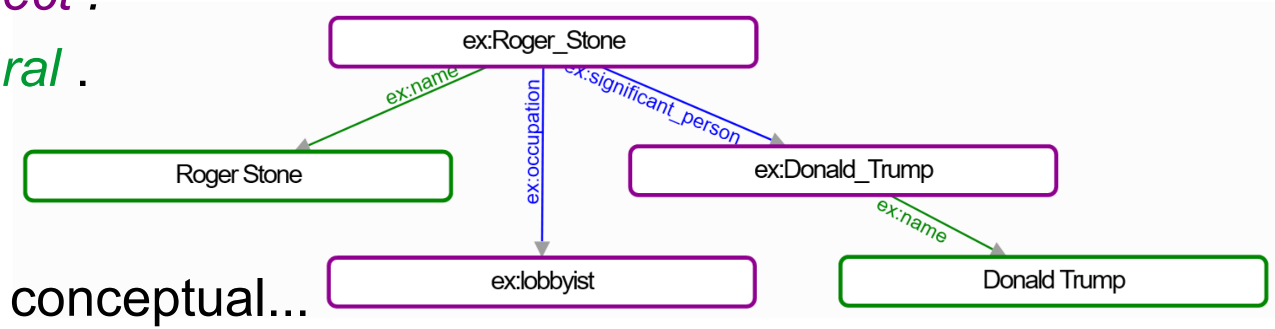


How can we represent semantic KGs?

- 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?

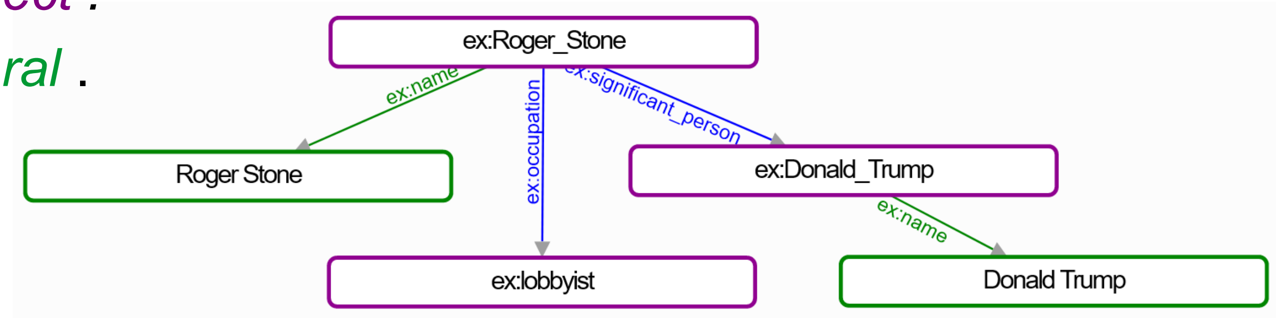
- Resource Description Framework (RDF)
- RDF models (KGs) consist of statements (triples)
  - of *subject predicate object* .
  - or *subject predicate literal* .
- The subject:
  - must be a *resource*
  - physical, informational, conceptual...



- The predicate:
  - must be a *property* (subtype of *resource*)
- The object:
  - is either a *resource*
  - or a *literal* (or a *value*: string, number... – *not* a *resource*)

How can we represent semantic KGs?

- Resource Description Framework (RDF → S02)
- RDF models (KGs) consist of statements (triples)
  - of *subject predicate object .*
  - or *subject predicate literal* .
- Serialisations, e.g., *Turtle*:



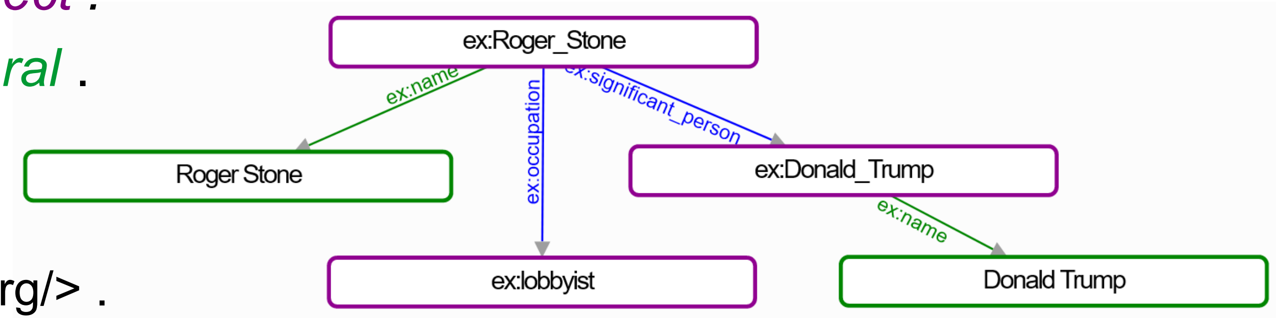
ex:Roger_Stone ex:name "Roger Stone" .
ex:Roger_Stone ex:occupation ex:lobbyist .
ex:Roger_Stone ex:significant_person ex:Donald_Trump .

ex:Donald_Trump ex:name "Donald Trump" .

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

How can we represent semantic KGs?

- Resource Description Framework (RDF → S02)
- RDF models (KGs) consist of statements (triples)
  - of *subject predicate object* .
  - or *subject predicate literal* .
- Serialisations, e.g., *Turtle*:

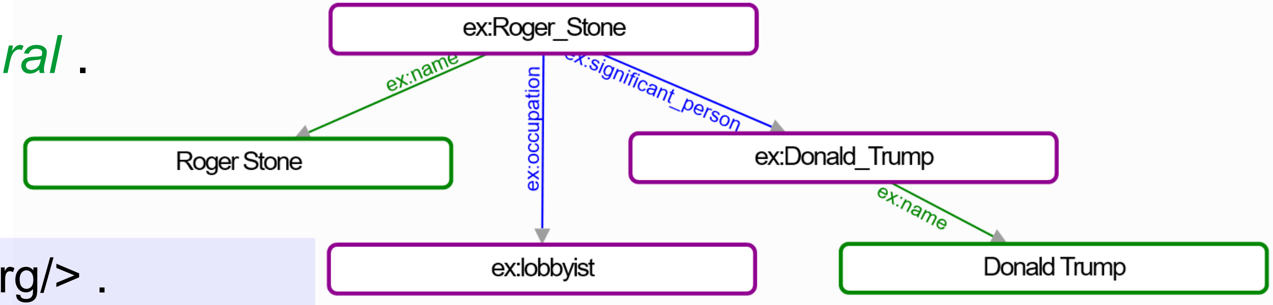


```
@prefix ex: <http://example.org/> .

ex:Roger_Stone ex:name "Roger Stone" .
ex:Roger_Stone ex:occupation ex:lobbyist .
ex:Roger_Stone ex:significant_person ex:Donald_Trump .

ex:Donald_Trump ex:name "Donald Trump" .
```

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

How can we represent semantic KGs?

- Resource Description Framework (RDF → S02)
- RDF models (KGs) consist of statements (triples)
  - of *subject predicate object* .
  - or *subject predicate literal* .
- Serialisations, e.g., *Turtle*:



```
@prefix ex: <http://example.org/> .

ex:Roger_Stone
    ex:name "Roger Stone" ;
    ex:occupation ex:lobbyist ;
    ex:significant_person ex:Donald_Trump .

ex:Donald_Trump
    ex:name "Donald Trump" .
```

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

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 (→S01)

- RDFLib:
  - an API 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*
  - supports SPARQL 1.1 Query and Update

RDFLib (→S01)

- RDFLib:
  - an API 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*
  - supports SPARQL 1.1 Query and Update
- More APIs and tools later:
  - a triple store (RDF database): *Blazegraph*
  - APIs for queries and rules: *SPARQLWrapper*, *OWL-RL* and *pySHACL*
  - a tool for OWL ontologies: *Protegé-OWL*
  - an OWL library for Python: most likely *owlready2*

RDFLib graphs (→S01)

- Graph:
  - a graph holds an RDF model
  - is a Python collection (set) of triples
  - supports adding, removing, listing, and searching for triples
  - supports writing to and reading from RDF files
  - responds to SPARQL queries and updates
  - backed by an in-memory or persistent *Store*
  - can be combined with other graphs

RDFLib resources (→S01)

- URIRef: a node with a URI (represents resources, types, relations)
- Namespace: a more compact way to create resources, types, and properties
  - predefined:
    - RDF, RDFS, OWL, XSD, FOAF, SKOS, DC, DCTERMS
    - `>>> from rdflib import RDF`
    - `>>> from rdflib.namespace import ...`
  - add prefix to graph:
    - `>>> g.bind(‘i2s’, i2s)`
- Triples / statements: ordinary 3-item Python tuples
- Literals: a typed or untyped value; strings can be language-tagged
- BNode:
  - a blank node (a resource without a URI)

Resources, properties, and literals

Resources

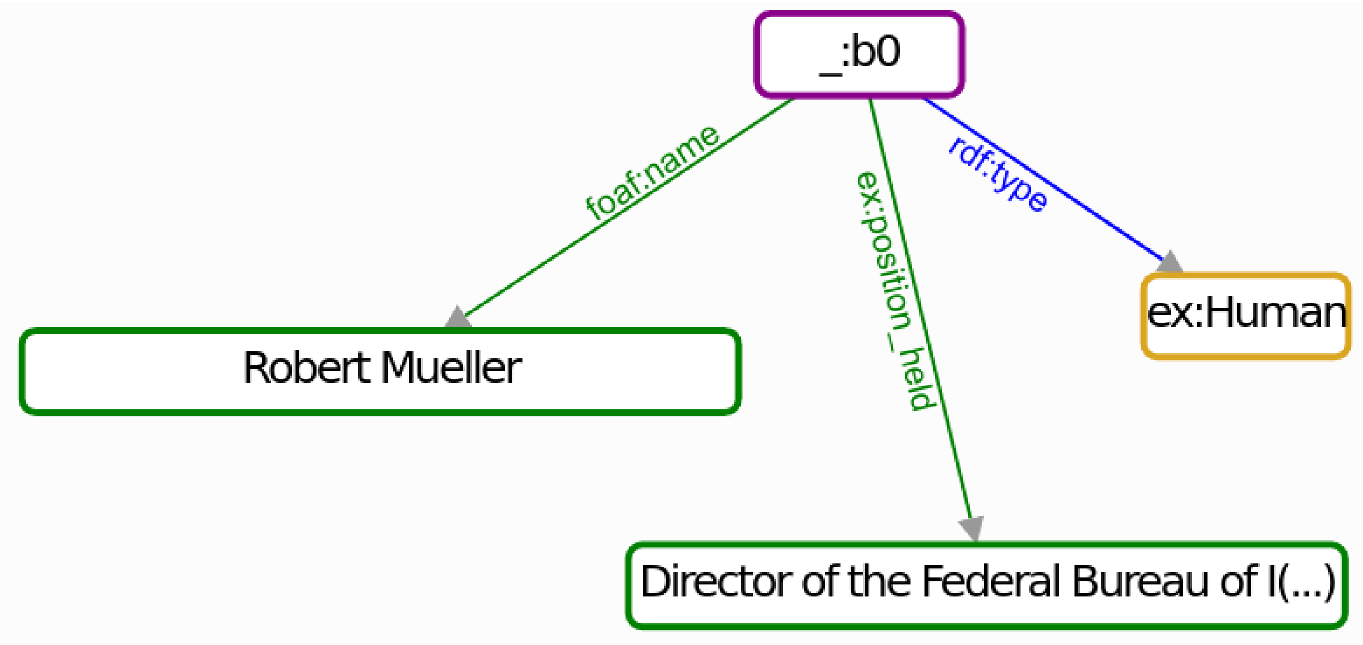
- RDF resources represent physical phenomena (including people and artefacts), information resources, concepts, constructs...
  - the nodes in knowledge graphs
  - can represent most things, really, as well as information about them
  - can be the *subject* or *object* in a statement
    - can also be *predicates*, but then we call them properties
  - can be *named* by an URI or *anonymous* (a blank *node*)
- A resources can have one or more *rdf:type*-s
  - ex:Robert_Mueller rdf:type ex:Human .
  - ex:Robert_Mueller **a** ex:Human **,** ex:Omnivore **,** rdfs:Resource .
- Every resource has the rdf:type *rdfs:Resource*
- *Convention: resource names start with a capital letter*

More Turtle shorthands!

Anonymous resources (blank nodes)

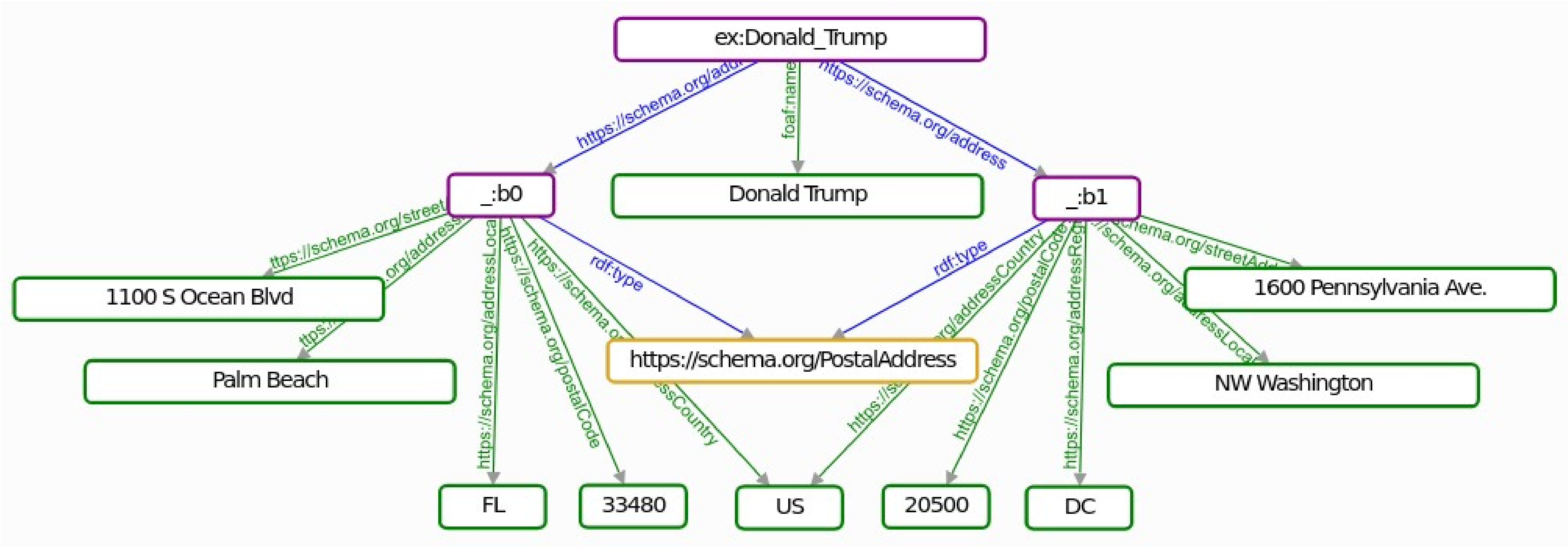
- Some resources (nodes) do not need URIs
- When to use?
  - when you do not (yet) know the right URI
  - when you do not want to reveal the URI (sensitive, business critical...)
  - when you need to group properties that are related
- Advantage:
  - no need to invent (“mint”) unnecessary URIs
- Disadvantages:
  - not supported by all RDF technologies
  - cannot be referenced from the outside
    - but can still have a local (non-URI) identifier *inside the graph*

Anonymous resources (blank nodes)



```
g = Graph()
g.bind('ex', EX)

robertMueller = BNode()
g.add((robertMueller, RDF.type, EX.Human))
g.add((robertMueller, FOAF.name, Literal('Robert Mueller', lang='en')))
g.add((robertMueller, EX.position_held, Literal('Director of the Federal Bureau of Investigation', lang='en')))
```

Anonymous resources (blank nodes)



Turtle syntax for blank nodes

```
ex:Mueller_Investigation ex:chairperson [] .
[] a ex:Human .
[] ex:position_held "Director of the Federal Bureau of Investigation"@en .
[] foaf:name "Robert Mueller"@en .
```

Each [] represents a *different* anonymous resource (blank node)

Turtle syntax for blank nodes

```
ex:Mueller_Investigation ex:chairperson _:b0 .
_:b0 a ex:Human .
_:b0 ex:position_held "Director of the Federal Bureau of Investigation"@en .
_:b0 foaf:name "Robert Mueller"@en .
```

Correct representation with *graph-internal labels*

Turtle syntax for blank nodes

```
ex:Mueller_Investigation ex:chairperson _:b0 .
_:b0 a ex:Human ;
     ex:position_held "Director of the Federal Bureau of Investigation"@en ;
     foaf:name "Robert Mueller"@en .
```

Correct representation with *graph-internal labels*

Turtle syntax for blank nodes

```
ex:Mueller_Investigation ex:chairperson [
        a ex:Human ;
        ex:position_held "Director of the Federal Bureau of Investigation"@en ;
        foaf:name "Robert Mueller"@en
] .
```

Predicate-object pairs embedded in the anonymous-node bracket.

Properties

- RDF properties are (a subtype of) resources that *either*
  - represent a relation from one resource to another *or*
  - represent a relation from a resource to a literal value
- Mostly used as a *predicate* in triples (statements)
  - examples:
    - rdf:type is a property defined by the (very small) RDF vocabulary
    - dc:title is a property in the Dublin Core (DC) vocabulary
    - foaf:name is a property in the Friend-of-a-Friend (FOAF) vocabulary
- Can *sometimes* be a subject or object in triples (statements)
    - foaf:name rdf:type rdf:Property .
- *Convention: property names start with lower-case letters*

Resource types

- RDFS classes are resources that represent the types of other resources
  - also nodes in knowledge graphs
  - usually with one or more rdf:type arrows pointing to them
  - often the *object* in a statement (but can sometimes be *subjects*)
- Examples:
  - ex:Human, ex:Omnivore, rdfs:Resource .
  - rdf:Property, rdfs:Resource, rdfs:Class .
- Every resource type itself has the rdf:type *rdfs:Class*
- *Convention: resource type names start with a capital letter (because they are resources)*

Literals

- RDF literals are used to represent values that describe resources (features)
  - always the *object* in a statement (triple)
- Untyped (simple) literals:
  - just a character string: “2001”, “““sixth director of the FBI””” *or*
  - a character string with a language code (ISO 639-1):
    “ Robert Mueller”@”en”, “رـابرتــمولر”@”fa”
- Typed literals:
  - a character string with *a URI that represents a literal type:*
    "2001"^^<http://www.w3.org/2001/XMLSchema#integer>
    "2001"^^<xsd:year>
- Every literal itself has the rdf:type *rdfs:Literal*

The examples are written in Turtle!

Literal types

- RDFS literal types are resources that represent the types of literals
  - also nodes in knowledge graphs
  - usually with one or more rdf:type arrows pointing to them
  - often the *object* in a statement (but can sometimes be *subjects*)
- XML Schema Definition (XSD) language is most used to represent literal types, for example xsd:string, xsd:integer, xsd:decimal, xsd:double, xsd:date, xsd:dateTime, xsd:anyURI
- Built-in literal types defined by RDF: rdf:XMLLiteral, rdf:HTML
- Other literal types can also be used, even self-defined ones
- Every literal type itself has the rdf:type *rdfs:Datatype*

XML Schema Definition (XSD) types

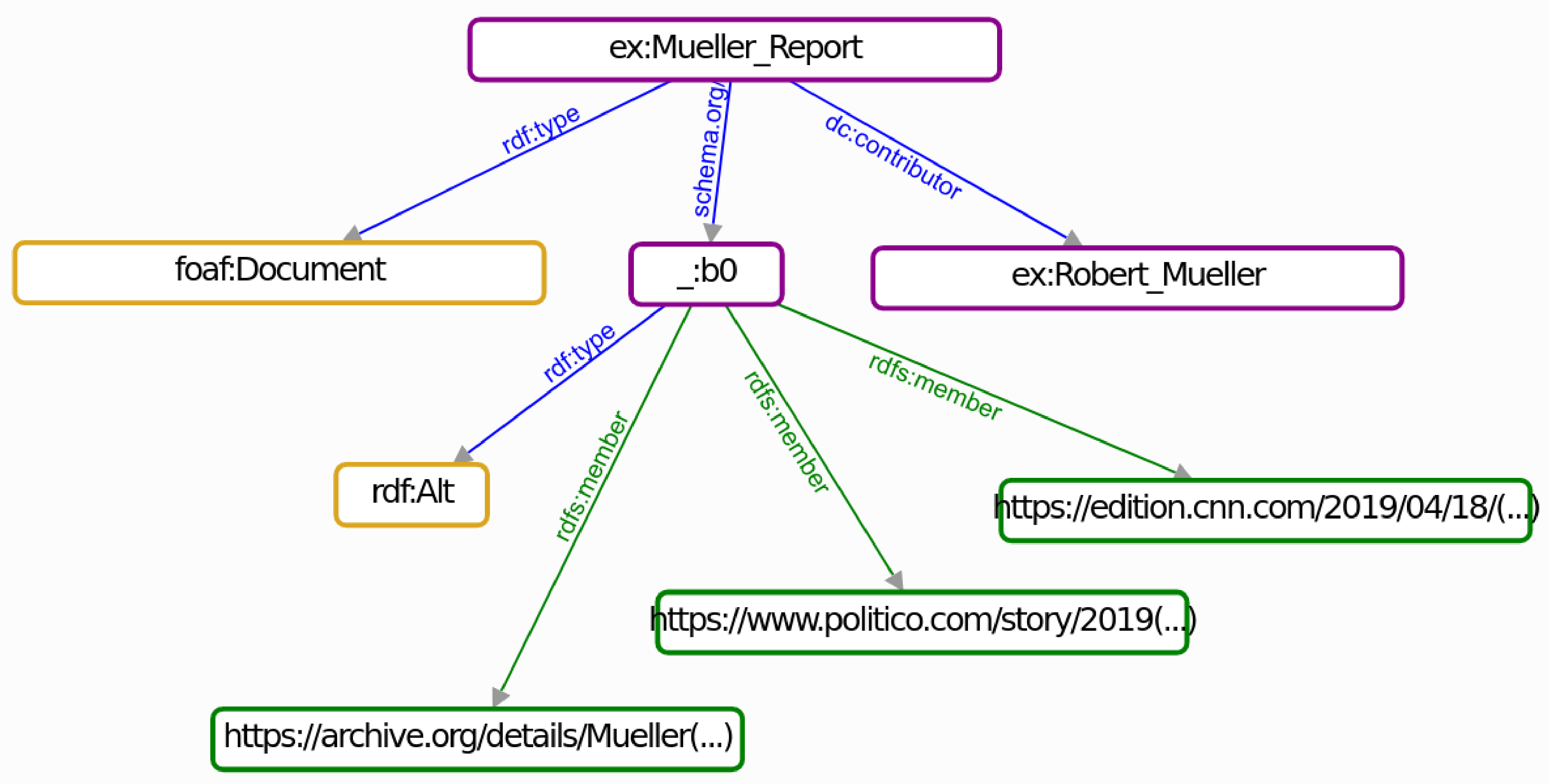
- Most XSD types can be used in RDF:
  xsd:string, xsd:boolean, xsd:decimal, xsd:integer, xsd:float, xsd:double, xsd:dateTime, xsd:dateTimeStamp, xsd:time, xsd:date, xsd:gYearMonth, xsd:gYear, xsd:gMonthDay, xsd:gDay, xsd:gMonth, xsd:duration, xsd:yearMonthDuration, xsd:dayTimeDuration, xsd:hexBinary, 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*

Containers

- An RDF container represents an *open* grouping of other resources
  - often the *subject* in a statement
  - usually with one or more rdfs:member arrows pointing from it
  - open: allows adding new members (without deleting triples)
  - often anonymous (blank), but not necessarily
- Every container has the rdf:type rdfs:Container
- Three subtypes:
  - rdf:type rdf:Alt – represents *alternative* resources
  - rdf:type rdf:Seq – represents resources that are *ordered*
    - special properties *rdf:_1, rdf:_2, ...* represent order of members
  - rdf:type rdf:Bag – represents resources that may be *duplicates*

Containers: alternatives

There are several *alternative* distribution sites.



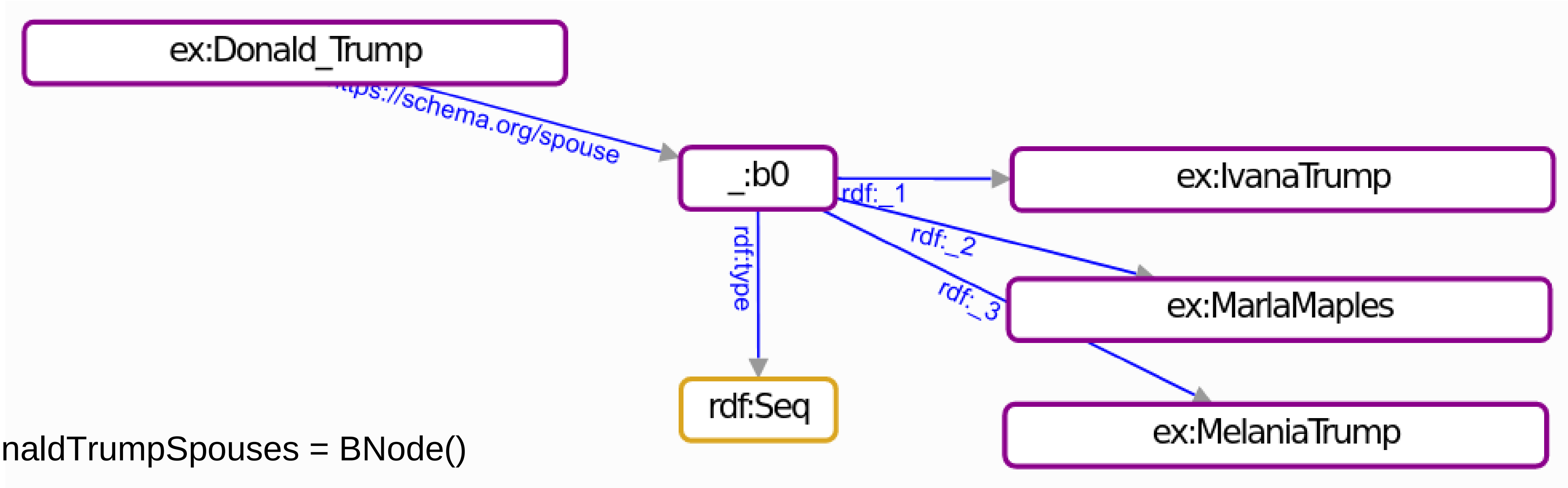
There are several *alternative* distribution sites.

Containers: alternatives

```
muellerReportArchives = BNode()
g.add((muellerReportArchives, RDF.type, RDF.Alt))
archive1 = 'https://archive.org/details/MuellerReportVolume1Searchable/’ \
                              ’Mueller%20Report%20Volume%201%20Searchable/'
archive2 = 'https://edition.cnn.com/2019/04/18/politics/full-mueller-report-pdf/index.html'
archive3 = 'https://www.politico.com/story/2019/04/18/mueller-report-pdf-download-text-file-1280891'

g.add((muellerReportArchives, RDFS.member, Literal(archive1, datatype=XSD.anyURI)))
g.add((muellerReportArchives, RDFS.member, Literal(archive2, datatype=XSD.anyURI)))
g.add((muellerReportArchives, RDFS.member, Literal(archive3, datatype=XSD.anyURI)))

g.add((EX.Mueller_Report, RDF.type, FOAF.Document))
g.add((EX.Mueller_Report, DC.contributor, EX.Robert_Mueller))
g.add((EX.Mueller_Report, SCHEMA.archivedAt, muellerReportArchives))
```

Containers: sequences

The wives are *ordered* and we can *add more* without deleting triples.



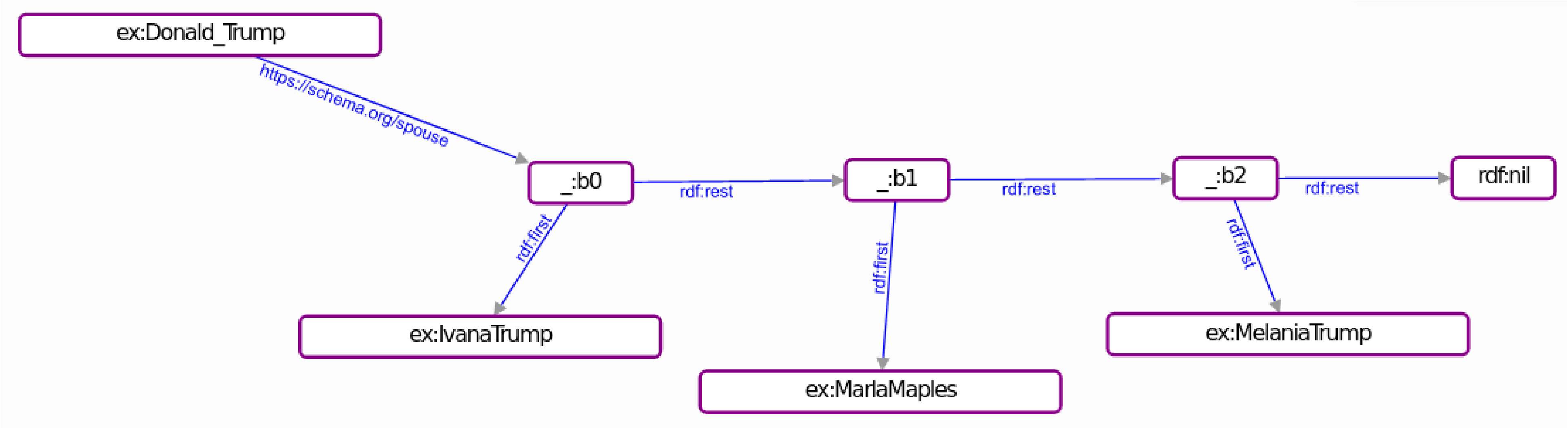
```
donaldTrumpSpouses = BNode()

g.add((donaldTrumpSpouses, RDF.type, RDF.Seq))
g.add((donaldTrumpSpouses, RDF._1, EX.IvanaTrump))
g.add((donaldTrumpSpouses, RDF._2, EX.MarlaMaples))
g.add((donaldTrumpSpouses, RDF._3, EX.MelaniaTrump))

g.add((EX.Donald_Trump, SCHEMA.spouse, donaldTrumpSpouses))
```

Collections

- An RDF collection represents a *closed* grouping of other resources
  - often the *subject* in a statement
  - with one rdf:first and one rdf:rest arrows pointing from it
  - closed: prohibits adding new members (without deleting triples)
  - often anonymous (blank), but not necessarily
- Every collection has the rdf:type rdf:List
  - rdf:first gives the first resource in the list (has rdf:type rdf:Property)
  - rdf:rest gives the rest of the list (has rdf:type rdf:Property)
  - rdf:nil represents an empty list (has rdf:type rdf:List)

Collections: lists

The wives remain *ordered* but we cannot *add more* wives without deleting triples.



```
donaldTrumpSpouses = BNode()
Collection(g, donaldTrumpSpouses, [
   EX.IvanaTrump, EX.MarlaMaples, EX.MelaniaTrump
])
g.add((EX.Donald_Trump, SCHEMA.spouse, donaldTrumpSpouses))
```

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 embeddings, 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*

A brief history of KGs

Tim Berners-Lee's call for a transition

- From around 1990: creation of a *Web of Documents*
  - the "plain old web" (PoW)
  - document-centric
  - document-to-document links
  - for humans
- From around 2000: transition to a *Web of Data*
  - document- *and data-centric*
  - doc-to-doc *and data-to-data links*
  - for humans *and machines*
  - also called the *Semantic Web*, *Web 3.0*, the *Web of Knowledge,* the *Linked Open Data (LOD) cloud*, the *Giant Global Graph (GGG)*, ...

Tim Berners-Lee
Inventor of the
World Wide Web
(WWW, 1989)

Tim Berners-Lee's call for a transition

- There's an enormous amount of data on the web
  - ...but the data are mostly not linked
    (think of a world wide web without document links!)
  - availability, accessibility does not go all the way
  - *what if we had standard ways of representing data so that linkable data could always be automatically linked?*
  - *enormous potential to solve, simplify, speed up... many critical information handling problems*
- This is the purpose of *semantic technologies*
- This is the vision that led to today's *semantic knowledge graphs*

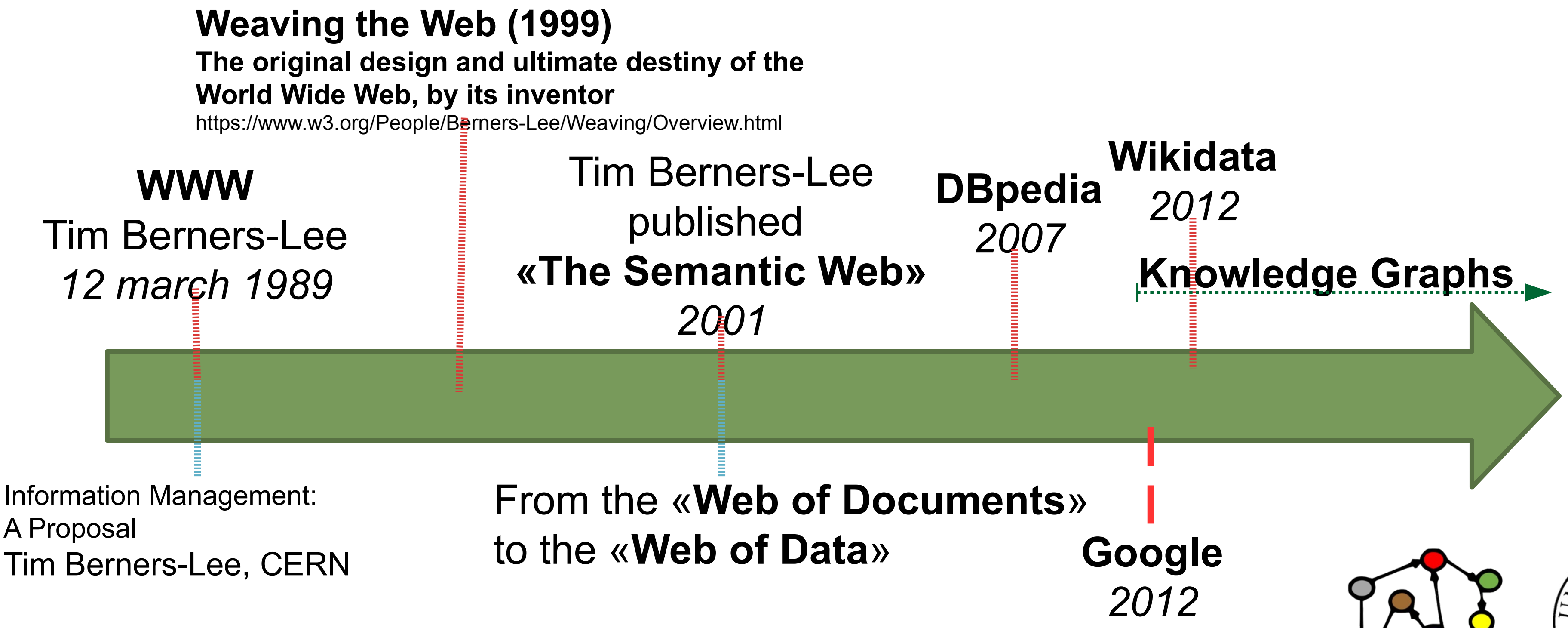
Tim Berners-Lee
Inventor of the
World Wide Web
(WWW, 1989)

Tim Berners-Lee: <http://www.youtube.com/watch?v=HeUrEh-nqtU>

Many independent, but related developments

- The *Linked Open Data (LOD)* cloud:
  - interlinking semantic datasets, making them openly available: DBpedia (2007-), Wikidata (2012-), …
- *Knowledge graphs:*
  - currently popular term for semantic graph representations of (primarily) factual information (Google, 2012)
- *Enterprise knowledge graphs:*
  - company-internal semantic data
  - linked open data and semantic-web technologies used inside an enterprise or cluster

Semantic web and WWW history



Weaving the Web (1999)

The original design and ultimate destiny of the World Wide Web, by its inventor

https://www.w3.org/People/Berners-Lee/Weaving/Overview.html

WWW
Tim Berners-Lee
12 march 1989

Tim Berners-Lee
published
«The Semantic Web»
2001

DBpedia
2007

Wikidata
2012

Knowledge Graphs

Information Management:
A Proposal
Tim Berners-Lee, CERN

From the «**Web of Documents**»
to the «**Web of Data**»

Google
2012

Tim Berners-Lee: http://www.youtube.com/watch?v=HeUrEh-nqtU

Information Management: A Proposal: https://cds.cern.ch/record/369245/files/dd-89-001.pdf

Common themes

- *Graph representations* of knowledge
  - RDF, RDFS, OWL, SPARQL
  - a recent alternative: Labelled-Property Graphs (LPGs)
- *Semantically tagged* data
  - well-defined tags (terms)
    - defined in standard vocabularies
    - formal ontologies, description logic
- *Global* and *interlinked*
  - standard formats, technologies, resource URIs, etc.
- From the start *open* and *community-based*

The LOD cloud

- *http://lod-cloud.net/*
  - which datasets mention resources in other datasets?
  - >1250 datasets with >15000 links between them
    - started in 2007
    - exponential-like growth for a few years
    - consolidating since ca 2017
- *How big is the LOD cloud?*
  - hard to measure exactly (old stats: http://*lodstats.aksw.org*)
  - approx. 150G (150 000M) triples from >3000 data sets (2020)
  - *Wikidata <http://wikidata.org>* is the largest general one:
    - >100M resources (items), >1,2G (1200M) triples

Next week:

Querying and updating KGs

(SPARQL)