Semantics, knowledge graphs and ontologies in practice

Jose Emilio Labra Gayo
WESO Research group
University of Oviedo, Spain

Schedule

Day	Title	Topics
Day 1.	Semantic Technologies and Knowledge graphs	Semantic Web Linked data Knowledge graphs RDF data model Property graphs Wikibase graphs Examples and applications
Day 2.	RDF data modelling and SPARQL	Data modelling exercises with RDF and turtle SPARQL
Day 3.	Validating RDF data	Shape Expressions (ShEx) SHACL Validating Knowledge Graphs
Day 4.	Advanced topics	ShEx and SHACL compared Reasoning RDFS OWL Nanopublications

Representing information in RDF

RDF = data model to exchange information in the Web

Some considerations & trade-offs

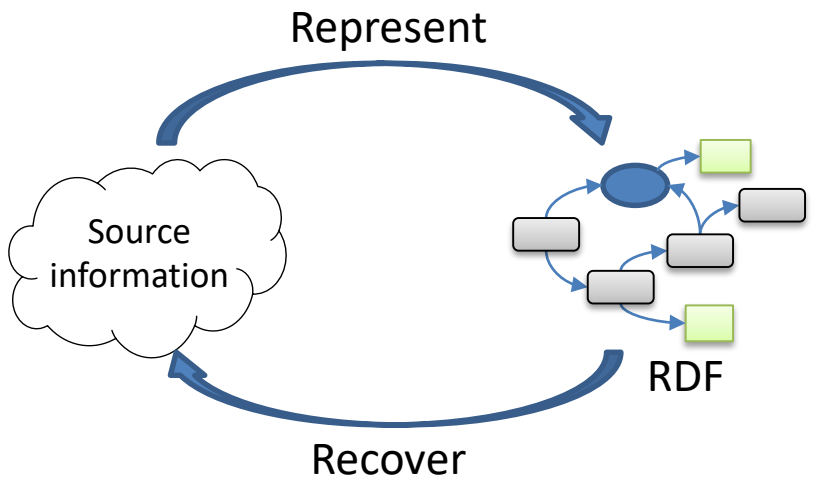
- Semantic accuracy
- Human readability
- Flexibility and schemaless
- Interoperability & performance

Semantic accuracy

Avoid semantic loss

Round-tripping

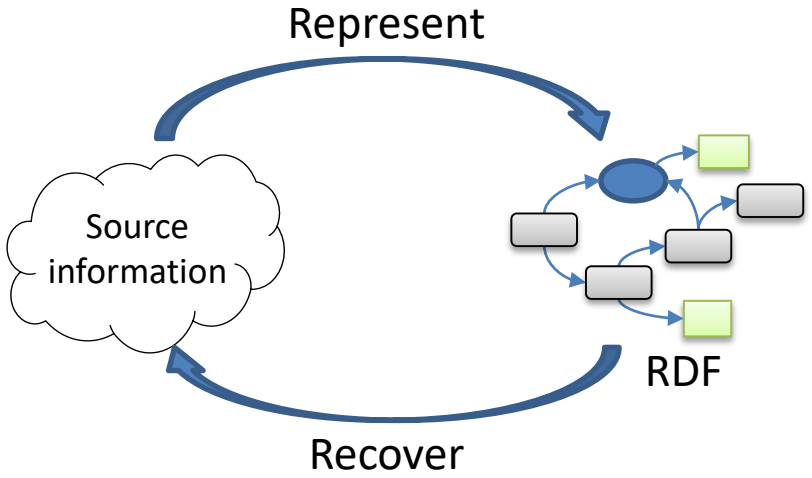
- From original representation to RDF
- From RDF recover original representation



Semantic accuracy

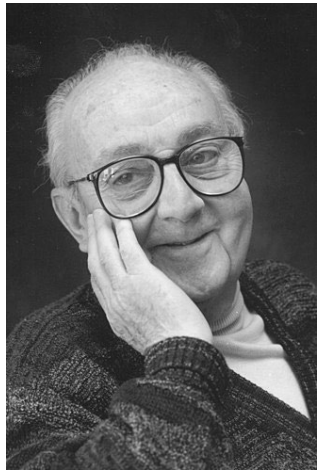
We should be careful about

Map-territory relationship



"All models are wrong, but some are useful"

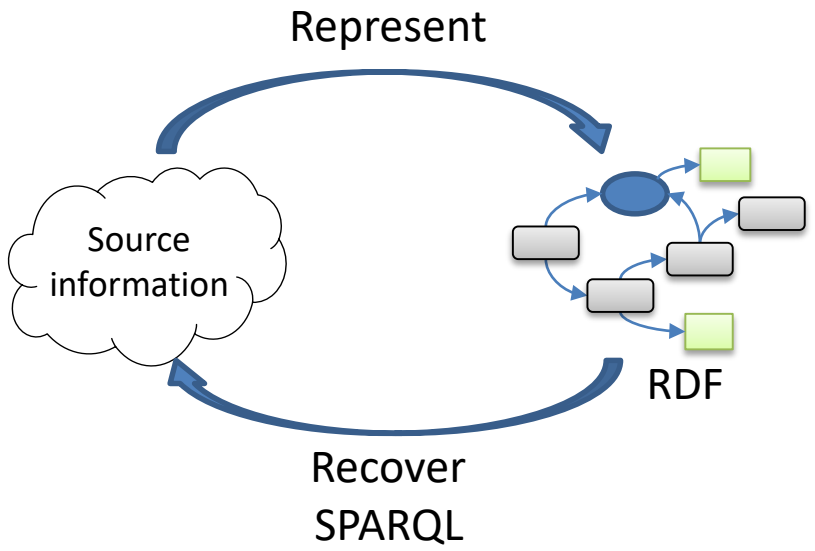
G. Box aphorism



George Box, source: Wikipedia

Semantic accuracy

- Represent in RDF
  - Convert from existing data
    - Relational databases and tabular data (CSV)
    - Hierarchical data
      - XML
      - JSON
- Recover from RDF
  - SPARQL queries



Human readability

- RDF as a communication language
  - Turtle can be human readable
  - Useful for debugging
- Big RDF datasets can be unreadable
  - Graph metaphor may not be useful for large data in practice

Flexibility and schemaless

- RDF data is very flexible
  - Several ways to model/represent the same information
  - Schemaless: No need to commit to some constraining schema
- Too much freedom?
  - We usually have some implicit schema
  - Knowing the structure of the data can be useful
    - Improves communication and documentation
    - Less need for defensive programming
    - Possible optimizations and more security

Interoperability

- RDF data should be machine processable
  - Adopt common vocabularies and URIs
  - Don't reinvent the wheel
  - Avoid ambiguity
  - Provide context and provenance for assertions
- Verbosity
  - Too much information can decrease readability/performance
    - Example: audiovisual content

Towards RDF data modelling methodology

Before

- Identify stakeholders
- Create competency questions
- Collect examples
- License and provenance

Data modelling

- Vocabulary selection
- URI design
- Define data shapes
- Setup infrastructure
- Convert existing sources

After

- Maintain pipelines
- Document endpoint
  - Examples, queries, shapes,...
- Engage users
  - Example apps & APIs
  - Hackathons
  - Data visualizations

RDF data modelling phase

Vocabulary selection

URI design

Define Data Shapes

Setup infrastructure

Conversion from existing sources

Vocabulary selection

- Find existing vocabularies
  - Examples:
    - LOV: Linked open vocabularies: https://lov.linkeddata.es/dataset/lov/
    - Bioportal: https://bioportal.bioontology.org/
- Create new vocabularies?
  - Sometimes it is necessary
    - Your concepts are not exactly the same as existing ones
    - You don't want too many external dependencies
  - Always try to map to existing vocabularies
    - owl:sameAs, skos:related, rdfs:seeAlso

URI design

- Cool URIs
  - Cool URIs don't change: https://www.w3.org/Provider/Style/URI
  - Cool URIs for the semantic web: https://www.w3.org/TR/cooluris/
- Some typical decisions: Opaque vs descriptive URIs
  - Opaque: http://www.wikidata.org/entity/Q14317
  - Descriptive URIs: http://dbpedia.org/resource/Oviedo
- Use URI patterns
  - Example: UK URI patterns
    - http://ukgovld.github.io/ukgovldwg/recommendations/uri-patterns.html

Data shapes

Understand your data

Define topology of RDF Graph

- Implicit vs explicit schemas
- Open vs closed data

Data shapes: ShEx, SHACL

Shape patterns?

Setup infrastructure

- Where do we store the data?
  - Not only native RDF Triplestores
  - Other possibilities: graph databases, relational databases
- RDF as a communication layer
  - Enable SPARQL endpoint
- Follow linked data principles
  - Content negotiation
  - Enable HTML views of data

Conversion from existing sources

- Beware of different data models/capabilities
  - Hierarchical data: JSON, XML
  - Tabular data: Excel, CSV
    - http://shexml.herminiogarcia.com/spec
  - Relational databases:
    - Direct mapping: https://www.w3.org/TR/rdb-direct-mapping/
    - More specific mappings: R2RML https://www.w3.org/TR/r2rml/
- Mapping technologies
  - Keep schema information?

Some RDF data modeling patterns

N-ary relationships

Tabular data

Representing order

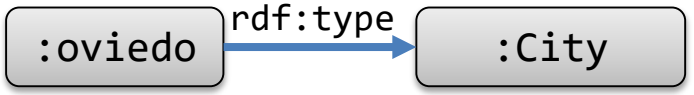
Reification and provenance

Grouping RDF triples and datasets

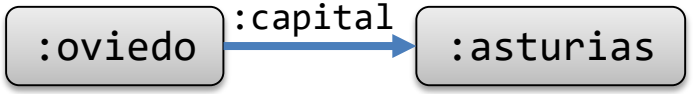
N-ary relationships

RDF can only express relationships between 1, 2 elements

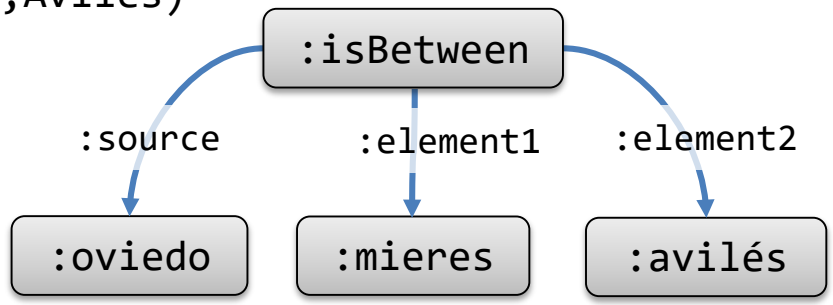
1-ary: *Oviedo is a city* `city(Oviedo)`



2-ary: *Oviedo is the capital of Asturias* `capital(Oviedo,Asturias)`



3-ary: *Oviedo is between Mieres and Avilés* `isBetween(Oviedo,Mieres,Aviles)`



Typical approach (*reify the relationship*)

Create an auxiliary node that represents the relationship

Add new relationships between nodes and the auxiliary node

Defining N-ary Relations on the Semantic Web:
https://www.w3.org/TR/swbp-n-aryRelations/

Tabular data

Example

Course

CID	Code	Title	Room	Teacher
23	CS101	Programming	A1	144
34	A102	Algebra	B2	144

Teacher

TeacherID	FirstName	LastName
144	Alice	Cooper

Each table can be seen as an n-ary relationship

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

:23 a :Course ;
    :code "cs101" ;
    :title "Programming"@en ;
    :room "A1" ;
    :teacher :144 .
:34 a :Course ;
    :code "A102" ;
    :title "Algebra"@en .
    :room "B2" ;
    :teacher :144 .
:144 a :Teacher ;
     :firstName "Alice" ;
     :lastName "Cooper" .
```

RDB2RDF: A Direct Mapping of Relational Data to RDF.
https://www.w3.org/TR/2012/REC-rdb-direct-mapping-20120927/

Representing order

RDF can easily represent sets but not lists

Several solutions

- Linked lists (RDF collections)
- Order-indicating properties (RDF containers)
- Add order annotations to values
- Give up order

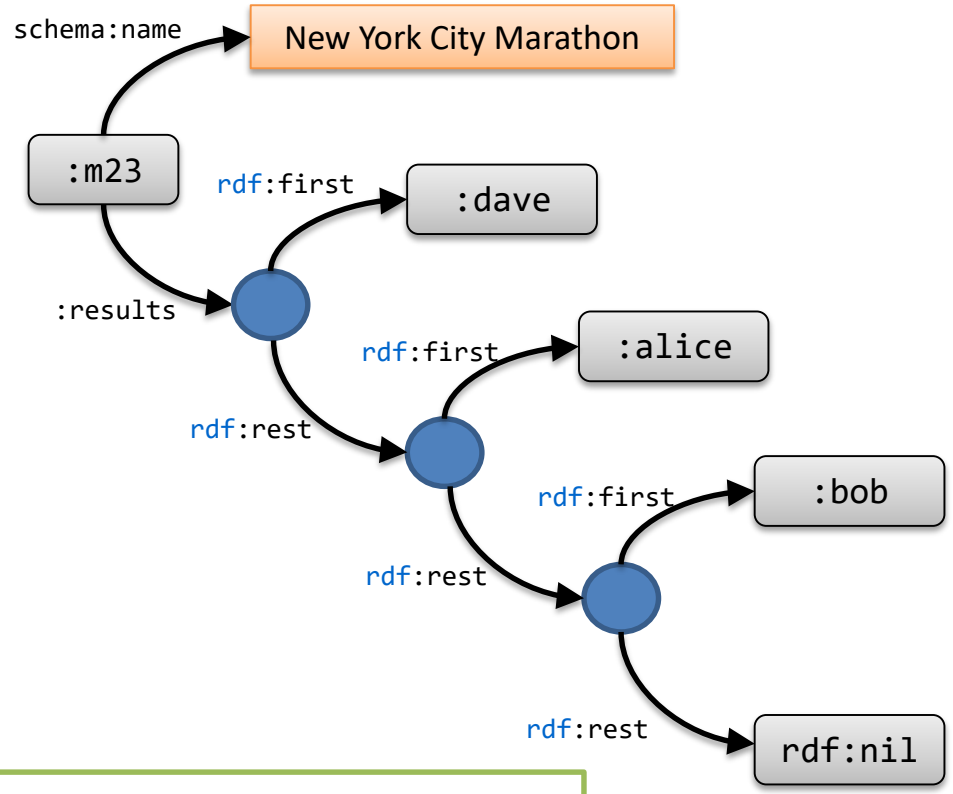
Solution 1
Representing order with linked lists

Ordered lists

```
:m23 schema:name "New York City Marathon ";
     :results    ( :dave :alice :bob ) .
```

Internally, represented as linked lists

```
:m23 schema:name "New York City Marathon ";
     :results _:1 .
_:1 rdf:first :dave ;
    rdf:rest _:2 .
_:2 rdf:first :alice ;
    rdf:rest _:3 .
_:3 rdf:first :bob ;
    rdf:rest rdf:nil .
```



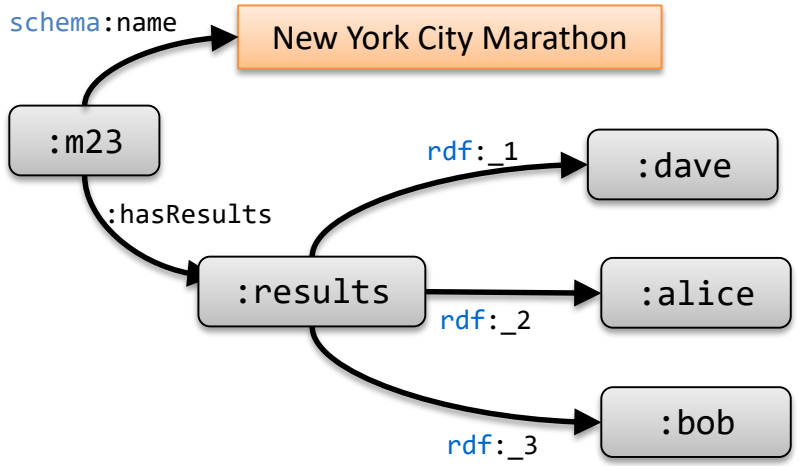
Pros: Elegant representation, easy insert/delete, mark end of list
Cons: Inefficient access to a given element

Solution 2
Representing order with properties

Use properties that indicate the order

RDF already has some specific properties: `rdf:_1, rdf:_2, ...`

```
:m23 schema:name "New York City Marathon ";
     :hasResults :results .
:results rdf:_1 :dave ;
         rdf:_2 :alice ;
         rdf:_3 :bob   .
```



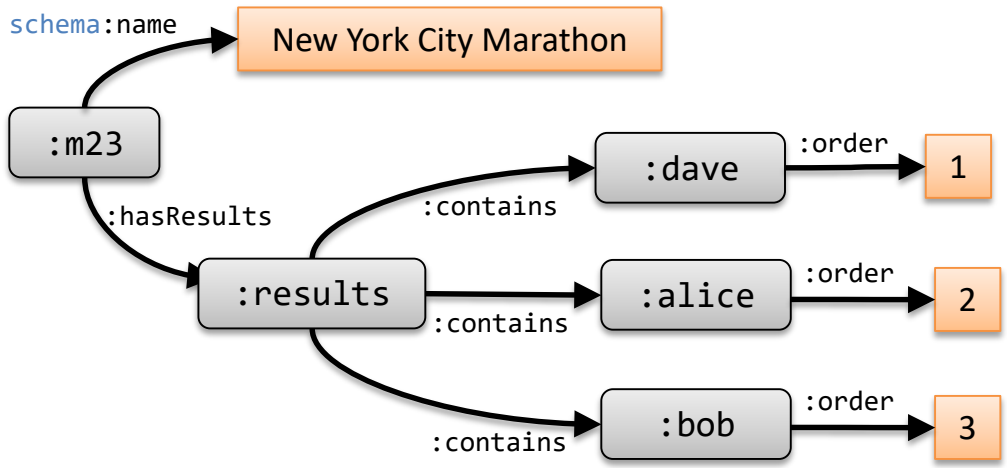
Pros: Direct access to each element

Cons: Not easy to detect the structure of list (length of the list, missing values, ...)
Harder to insert/delete elements

Solution 3
Representing order with annotated values

Annotate the elements with a value that indicates order

```
:m23 schema:name "New York City Marathon ";
     :hasResults :results .
:results :contains :dave, :alice, :bob.
:dave   :order 1 .
:alice  :order 2 .
:bob    :order 3 .
```



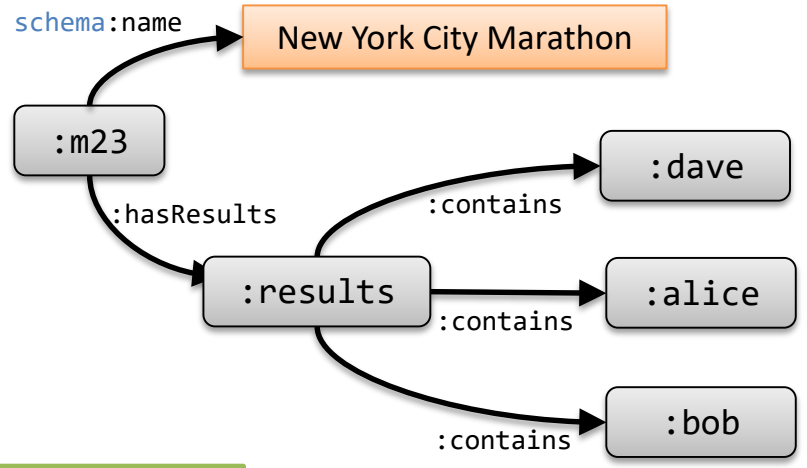
Pros: Direct access to each element is possible, length of list available
Cons: It is possible to create inconsistencies (elements with same order)
Harder to insert/delete elements

Solution 4
Ignore the order

Sometimes order is not really required

Give up the order and represent lists as sets

```
:m23 schema:name "New York City Marathon ";
     :hasResults :results .
:results :contains :dave, :alice, :bob.
```



Pros: Easy to do in RDF, may suffice for many use cases
Cons: No order

Solution 5
Combine several approaches

RDF is very versatile

It is possible to combine several approaches

```
:m23 schema:name "New York City Marathon ";
     :hasResults :results .
:results :contains :dave, :alice, :bob.
         :order    ( :dave, :alice, :bob ) .
```

Pros: May offer the pros of the different approaches
Cons: Increased data volumen, redundancy and possible inconsistencies

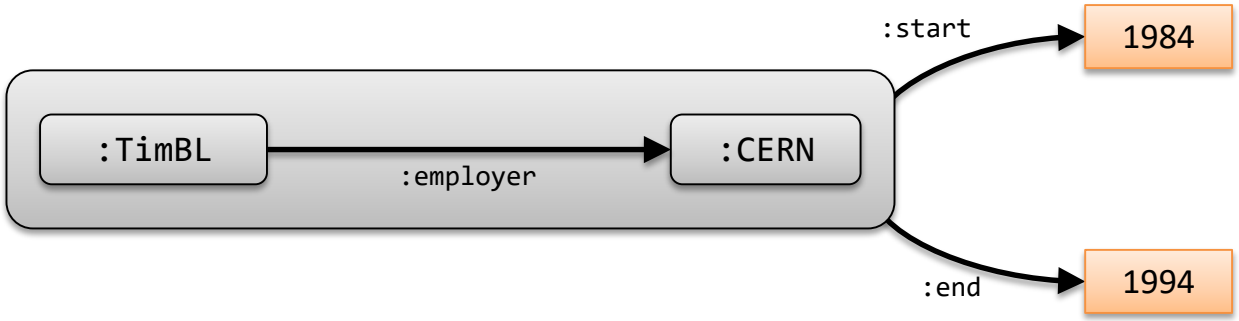
Reification

Reification: add statements about statements

Example: *Tim Berners-Lee is employed at CERN* *(between 1984 and 1994)*

Some approaches

- Standard RDF reification
- N-ary relations
- RDF-*
- Named graphs



Reification approach 1
Standard RDF reification

Introduced already in RDF 1.0

Predicates `rdf:subject, rdf:predicate, rdf:subject`

Class `rdf:Statement`

```
:s1 a rdf:Statement ;
    rdf:subject   :TimBl ;
    rdf:predicate :employer ;
    rdf:object    :CERN ;
    :start        "1984"^^xsd:gYear ;
    :end          "1994"^^xsd:gYear .
```

Pros: It is part of RDF, since RDF 1.0

Cons: Not easy to manage and not very flexible. Not compatible with OWL DL

Reification approach 2
Statements as n-ary relations

Create an auxiliary node to represent the statement

Add properties to relate the nodes with that auxiliary node

```
:timBl :employer :e .
:e :organization :CERN ;
   :start        "1984"^^xsd:gYear ;
   :end          "1994"^^xsd:gYear .
```

Pros: It can be directly expressed in RDF

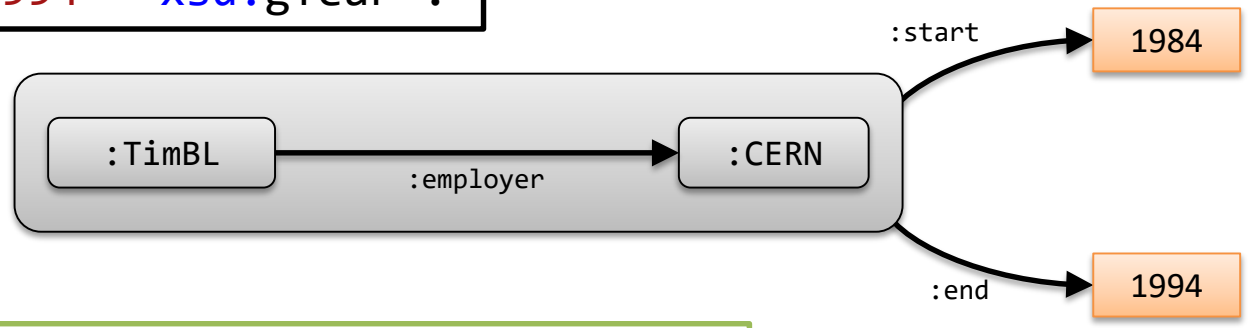
Cons: Requires the creation of auxiliary nodes and properties

Reification approach 3
RDF-*

RDF-* = RDF extension

where graphs can be either subjects or objects of a statement

```
<< :timBl :employer :CERN >> :start "1984"^^xsd:gYear ;
                              :end   "1994"^^xsd:gYear .
```



Pros: It expresses directly reification

Cons: Not yet widely adopted. It may require tools to convert to RDF

RDF-* Community group specification: https://w3c.github.io/rdf-star/cg-spec

Reification approach 4
Named graphs

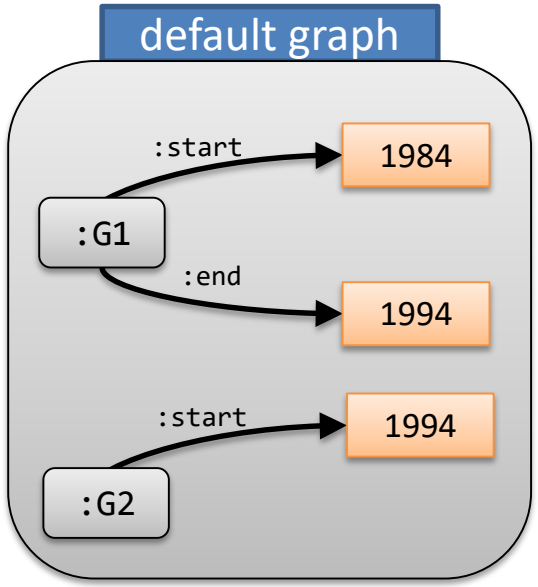
RDF datasets = collection of RDF graphs (supported also by SPARQL)

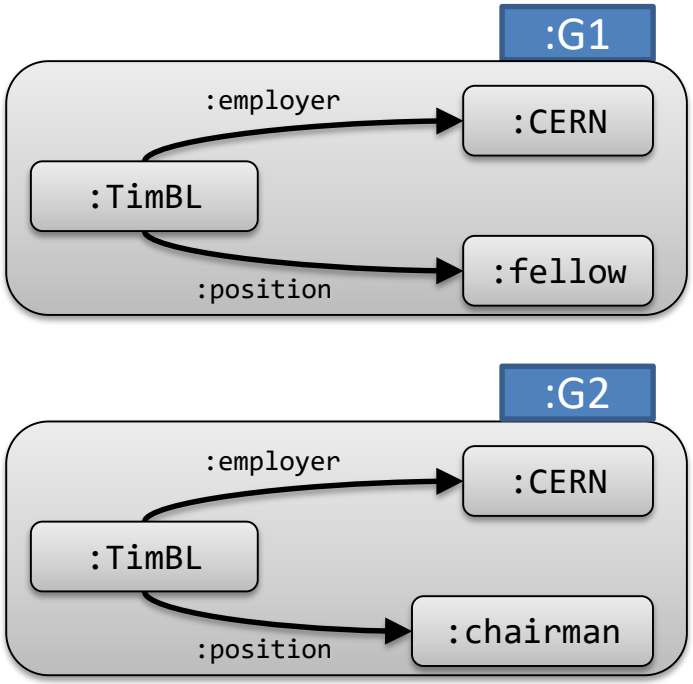
- A default graph
- Zero or more named graphs (name = IRI/Blank node)

TRIG = Turtle extension that can express RDF datasets

```
:G1 :start "1984"^^xsd:gYear .
:G1 :end   "1994"^^xsd:gYear .
:G2 :start "1994"^^xsd:gYear .

:G1 {
  :timBl :employer :CERN ;
         :position :fellow
}
:G2 {
  :timBl :employer :W3C .
         :position :chairman
}
```





Some references

Linked data patterns

https://patterns.dataincubator.org/

Ontology design patterns

http://ontologydesignpatterns.org/

Working ontologist book

http://workingontologist.org/

Best Practices for Publishing Linked Data.

https://www.w3.org/TR/ld-bp/

Data on the Web Best Practices

https://www.w3.org/TR/dwbp/