Shaping Knowledge and Interoperable Graphs

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

About me…

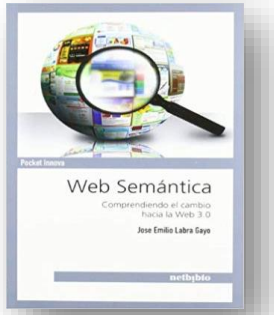
Main researcher at WESO (Web Semantics Oviedo)

Some books:

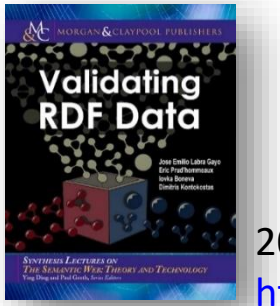
- "*Web semántica*" (in Spanish), 2012
- "*Validating RDF data*", 2017
- "Knowledge Graphs", 2021

…and some software:

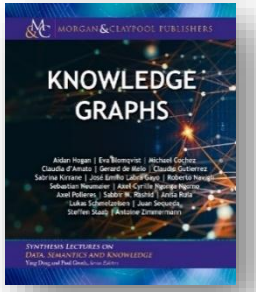
- RDFShape (RDF playground)
- rudof



2012



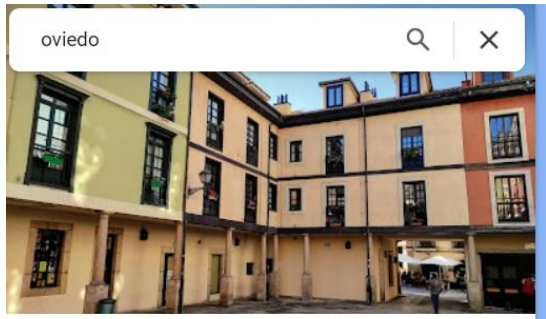
2017 HTML version:
http://book.validatingrdf.com

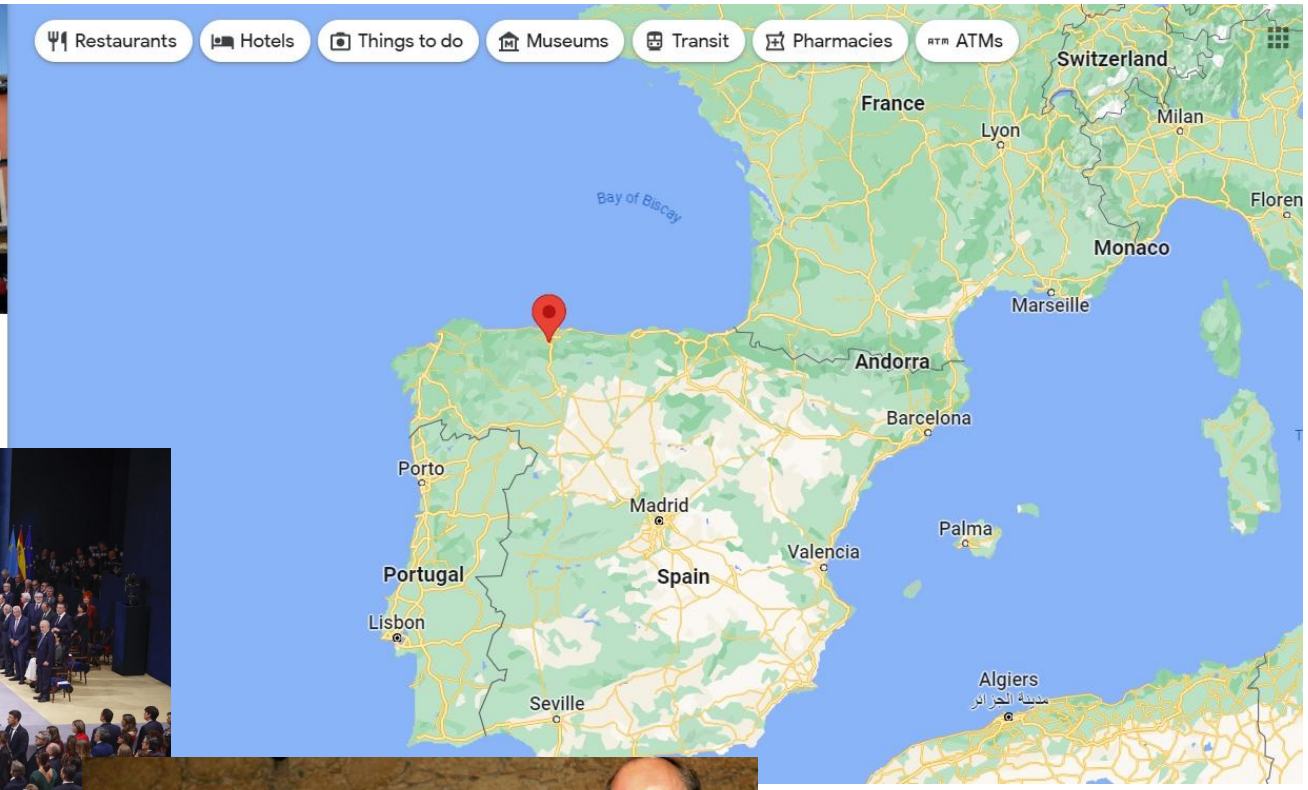


2021, HTML version
https://kgbook.org/

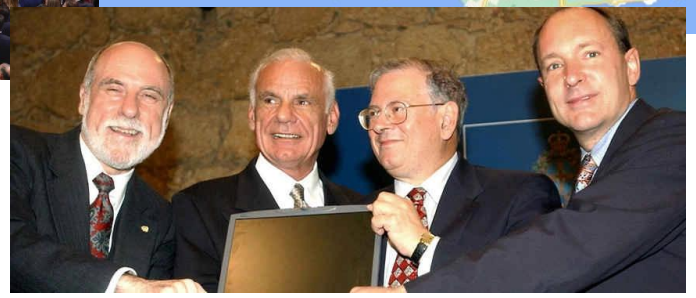
http://labra.weso.es

About Oviedo, Asturias





Princess of Asturias Awards



2002, Tim Berners-Lee in Asturias receiving the Prince of Asturias Award
Together with: Vinton Cerf, Lawrance Roberts and Robert Kahn

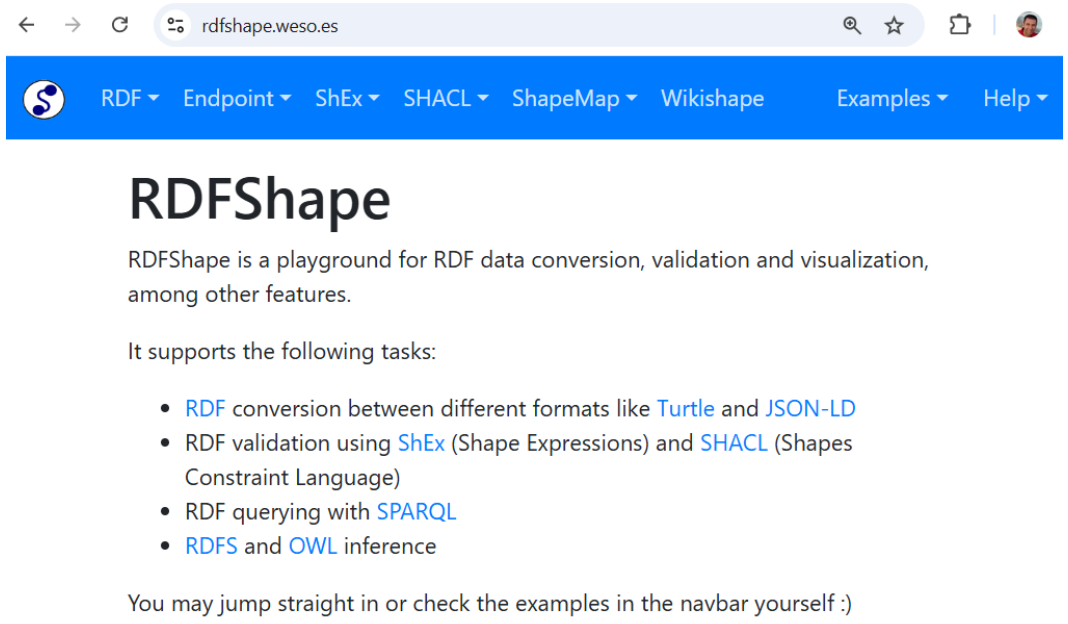
Slides: https://www.validatingrdf.com/tutorial/iswc2025/

Tutorial materials

Interactive – through the web

RDFShape: RDF playground

Wikishape: Tailored to Wikibase



RUDOF

Command line

Rudof

Binaries in Windows, Linux, Mac

Interactive (Jupyter)

rudof

Python bindings

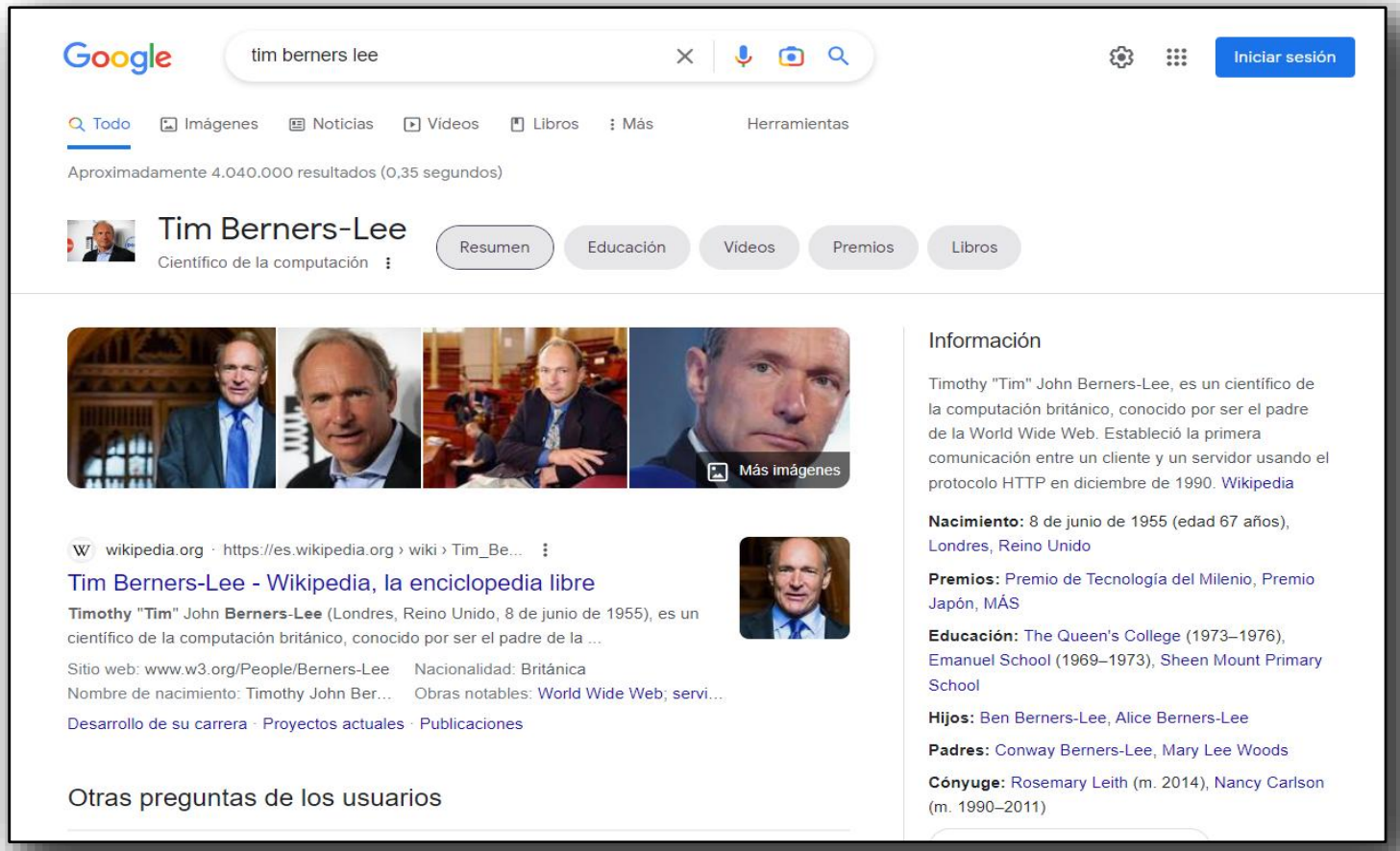
https://rudof-project.github.io/tutorials

Contents

- Introduction to Knowledge graphs
- Types of Knowledge Graphs:
  - RDF, Property graphs, Wikibase, RDF-Star
- Shaping RDF: ShEx & SHACL, DCTAP
- Shaping other types of Knowledge graphs:
  - Wikibase and Wikidata graphs
  - Property Graphs
  - RDF-1.2

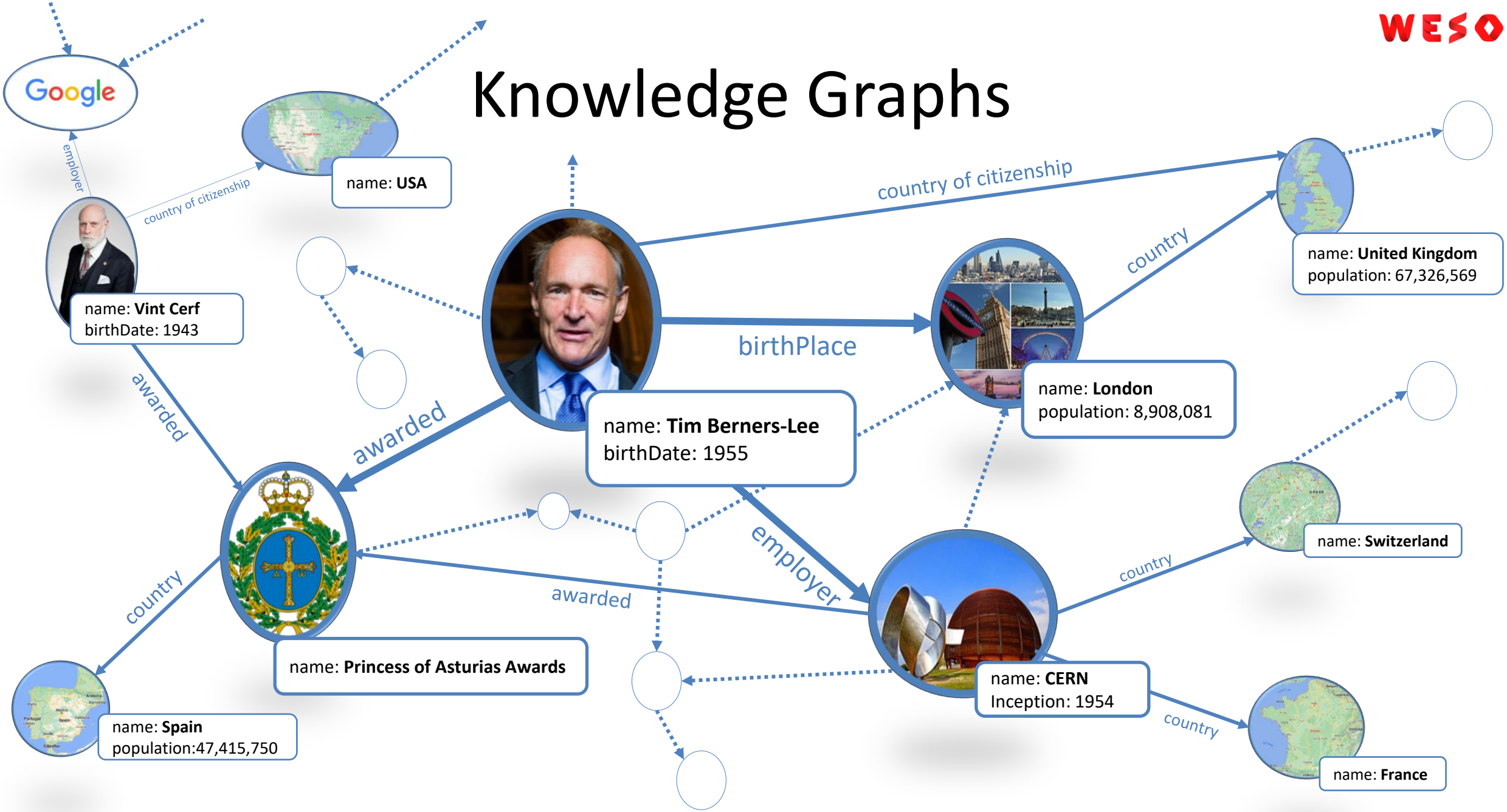
Knowledge Graphs

Current notion of Knowledge Graphs, popular after Google, 2012*



Link: https://www.blog.google/products/search/introducing-knowledge-graph-things-not/

Knowledge Graphs



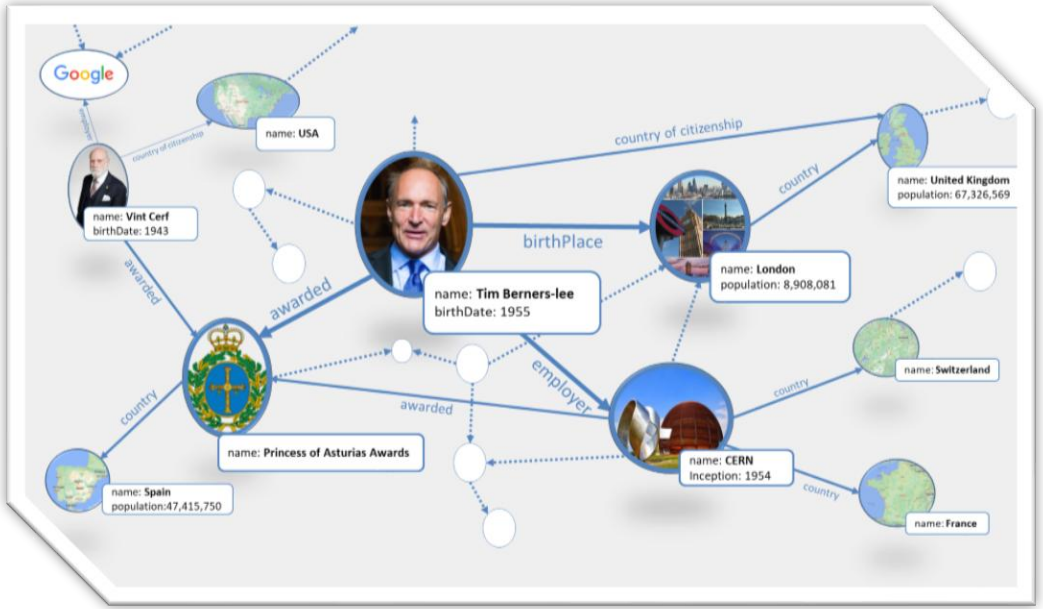
Knowledge Graphs

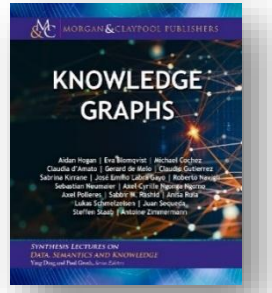
Knowledge graph = *a **graph of data***

*intended to accumulate and convey **knowledge** of the real world*

*whose nodes represent **entities** of interest and*

*whose edges represent **relations** between these entities.*





https://kgbook.org/

Applications of Knowledge Graphs

Improve search results

Question answering

Data governance

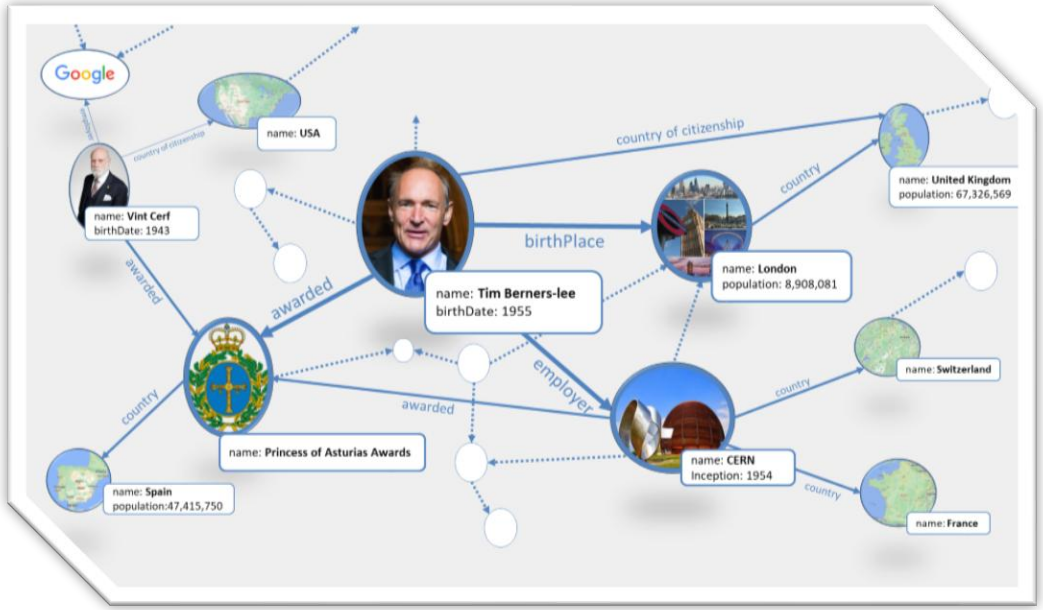
Handling heterogenous data

Recommender systems

Chatbots, NLP, LLMs

Explainability in AI

. . .



Contents

Types of Knowledge Graphs

- Traditional RDF
- Labeled Property graphs
- Wikibase graphs
- RDF 1.2

RDF

Resource Description Framework

Lingua franca of the Semantic Web

- 1997 1st public Working draft https://www.w3.org/TR/WD-rdf-syntax-971002, RDF/XML
- 1999 1st W3C Rec https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/, RDF Model and Syntax
- 2004 - RDF Revised https://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, Turtle
- 2008 - SPARQL 1.0, https://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/
- 2014 - RDF 1.1 https://www.w3.org/TR/rdf11-concepts/, SPARQL 1.1, JSON-LD
- 2017 - SHACL 1.0 https://www.w3.org/TR/2017/REC-shacl-20170720/
- 2025 - RDF 1.2 https://www.w3.org/TR/rdf12-concepts Statements about triples

RDF graphs

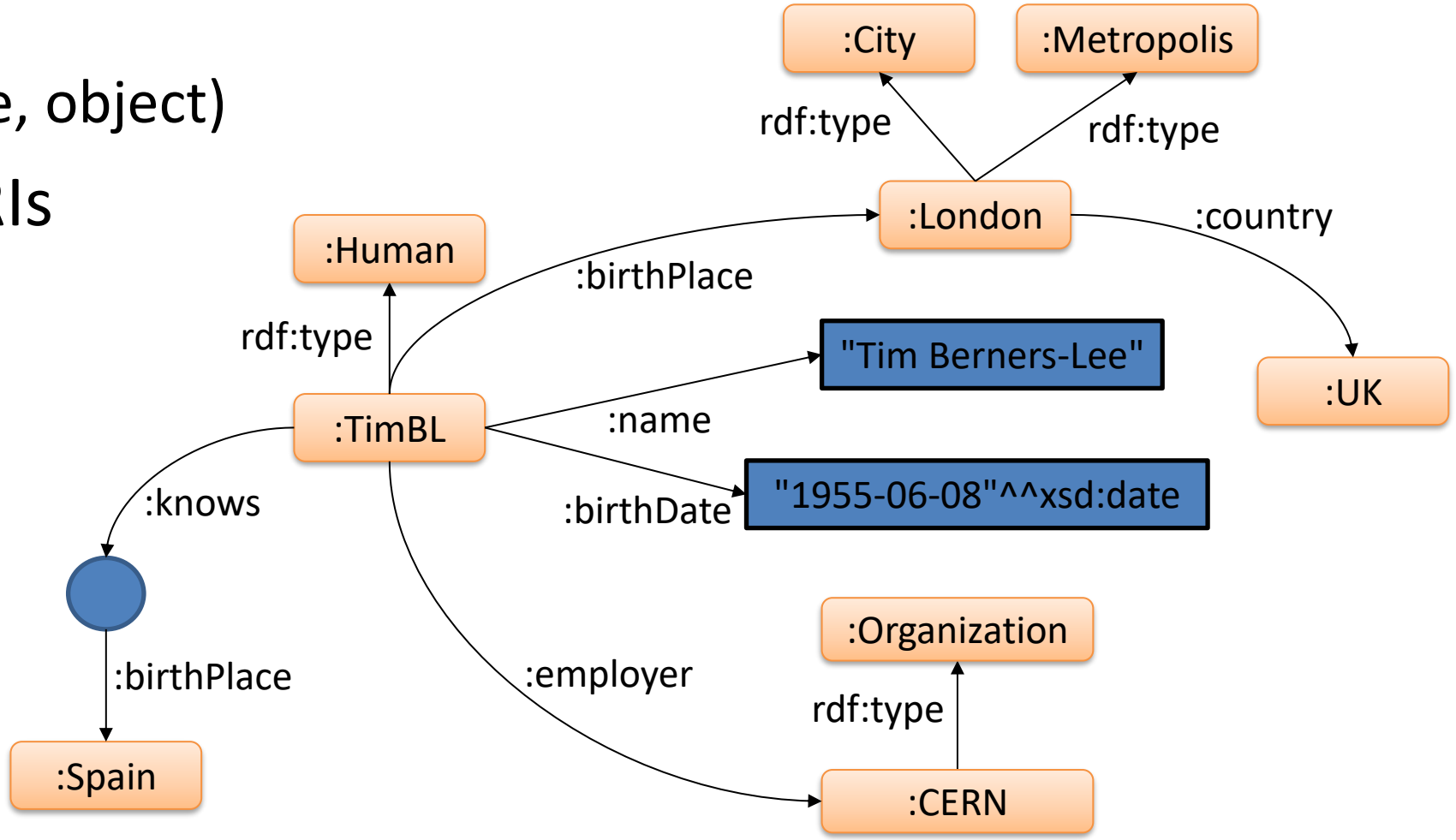
Based on triples

(subject, predicaje, object)

Most nodes are URIs

Interoperability

Simple & flexible



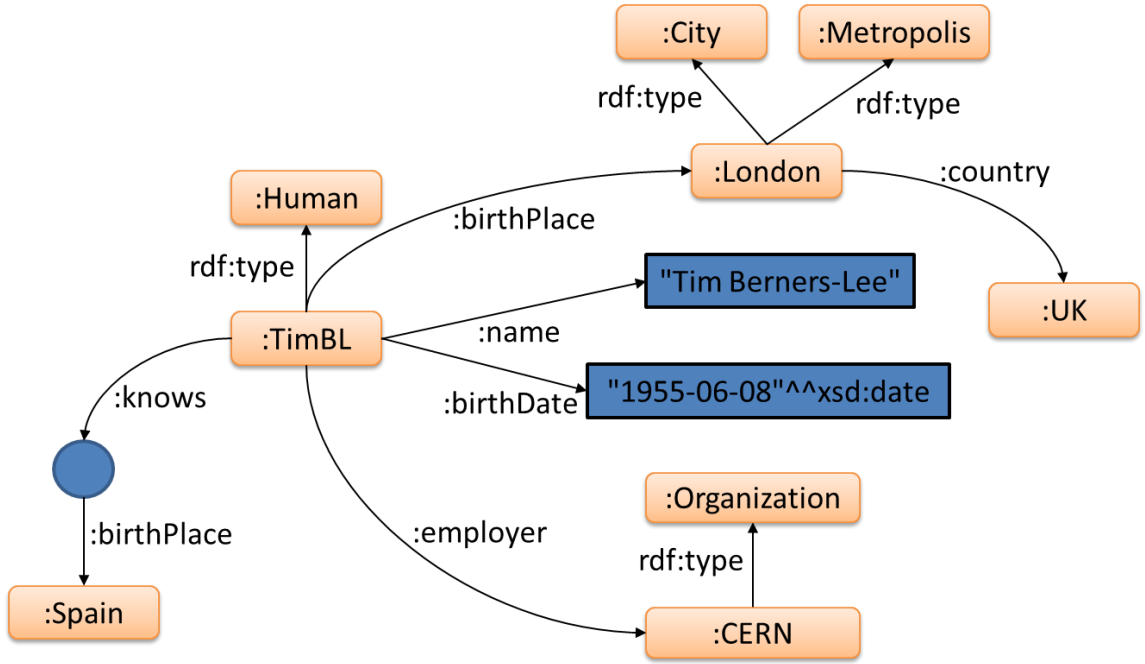
RDF ecosystem

One data model, several syntaxes: Turtle, N-Triples, JSON-LD

Vocabularies: RDF Schema, OWL, SKOS, etc.

Turtle

```
prefix :       <http://example.org/>
prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
prefix xsd:    <http://www.w3.org/2001/XMLSchema#>
prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

:timbl  rdf:type     :Human ;
        :birthPlace  :london ;
        :name        "Tim Berners-Lee" ;
        :birthDate   "1955-06-08"^^xsd:date ;
        :employer    :CERN ;
        :knows       _:1    .
:london rdf:type     :City, :Metropolis ;
        :country     :UK .
:CERN   rdf:type     :Organization .
_:1     :birthPlace  :Spain .
```



https://rdfshape.weso.es/link/17344285150

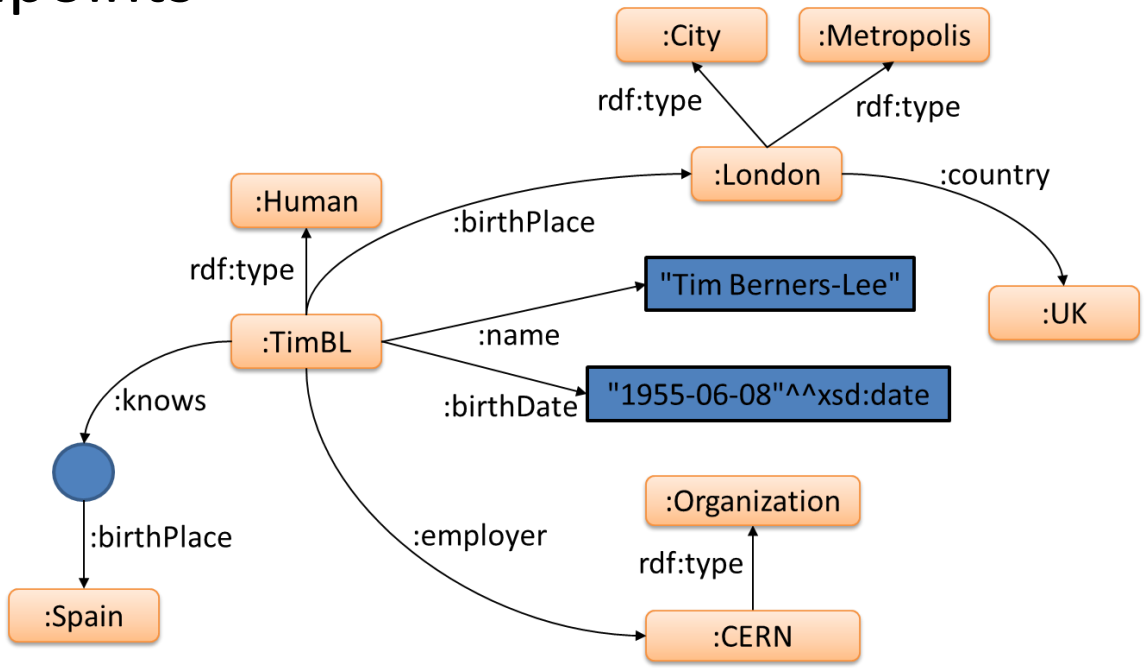
RDF ecosystem: SPARQL

SPARQL = RDF query language and protocol

Enables the creation of SPARQL endpoints

```
SELECT ?person ?date ?country WHERE {
    ?person :birthDate  ?date .
    ?person :birthPlace ?place .
    ?place  :country    ?country
}
```

?person	?date	?country
:timbl	1955-06-08	:UK



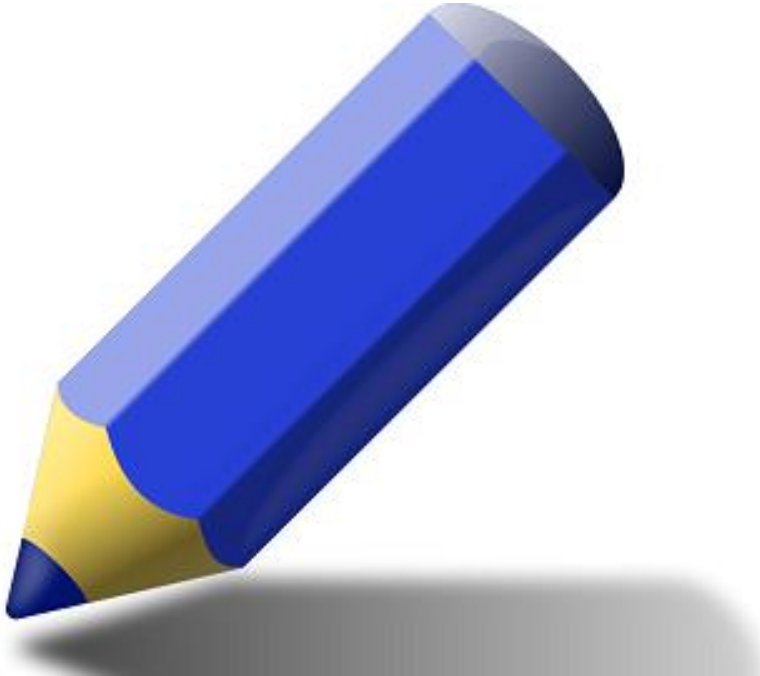
Try it: https://rdfshape.weso.es/link/17313175698

Let’s check some RDF tools

Let’s check the tools for RDF

- Declare that:
  - Tim Berner’s Lee won the Princess of Asturias Award

- ❑ Did you use RDFShape?
- ❑ Did you use the Jupyter Colab?
- ❑ Did you use rudof as a CLI
- ❑ Do you think your RDF is OK?



Labeled Property graphs

- Since ~2007, very popular model in industry
  - Neo4j, Amazon Neptune, Oracle, etc
  - Several query languages: Cypher, Gremlin, PGQL, …
- 2024 - GQL has been published
  - Recent publication of ISO/IEC FDIS 39075
  - Developed by the “SQL” committee
  - Influenced by Cypher, PGQL, etc.

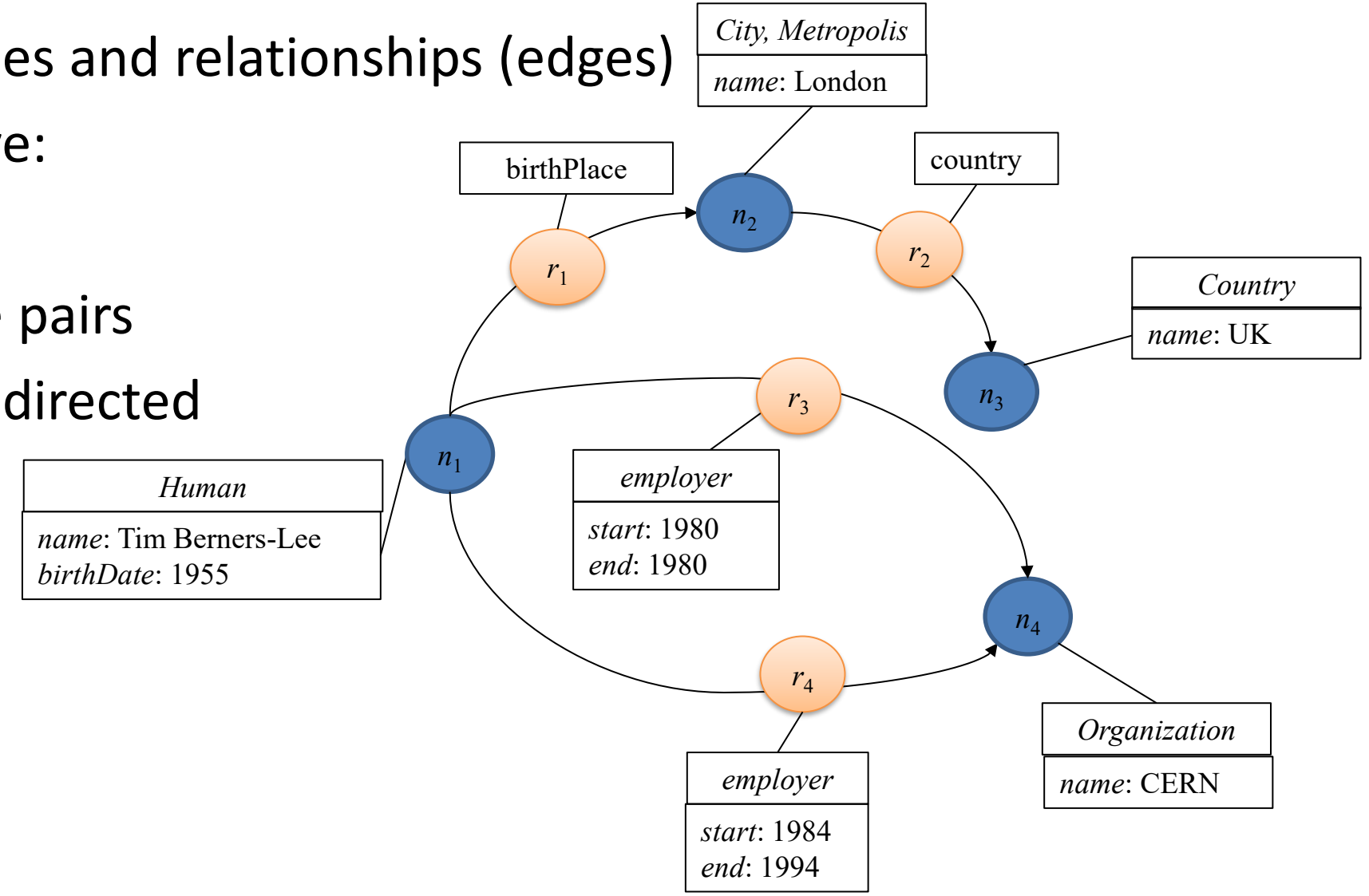
Property graphs

Graph structure with nodes and relationships (edges)

Nodes and edges can have:

- Labels
- A set of property-value pairs

Edges can be directed/undirected



Wikidata based Knowledge Graphs

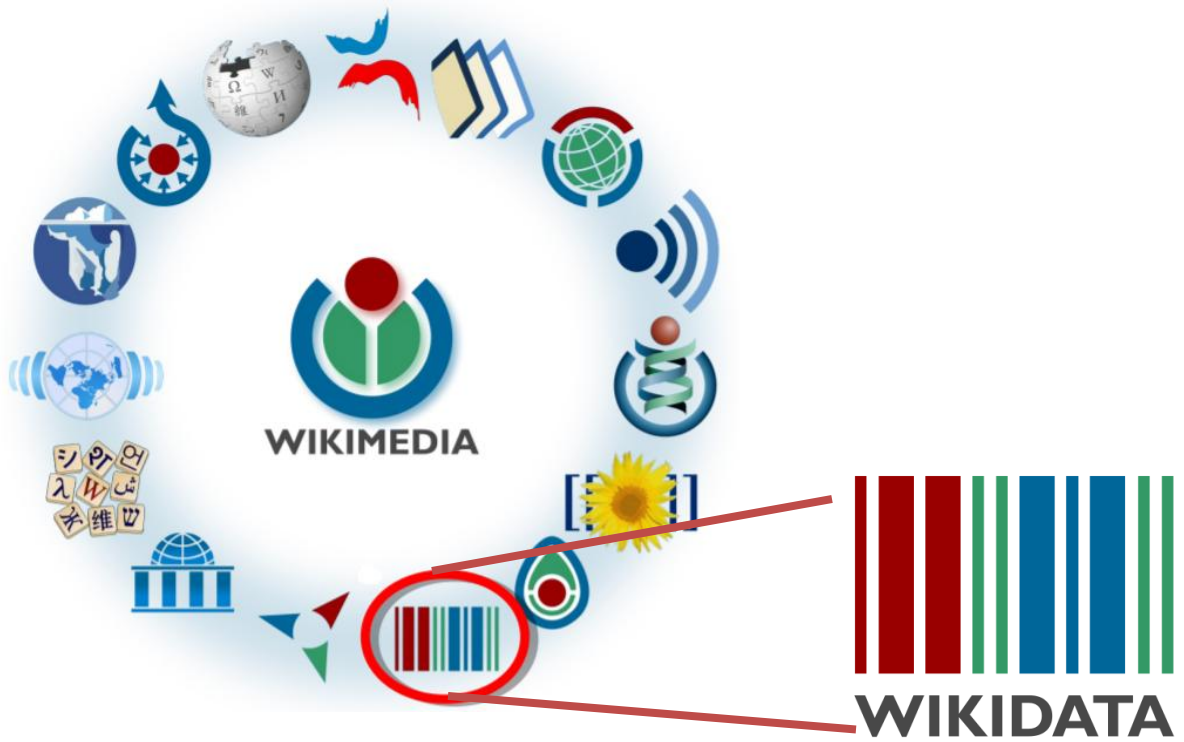
Wikidata was created in 2012 as a collaborative knowledge graph

https://www.wikidata.org/

Developed and supported by Wikimedia Deutschland

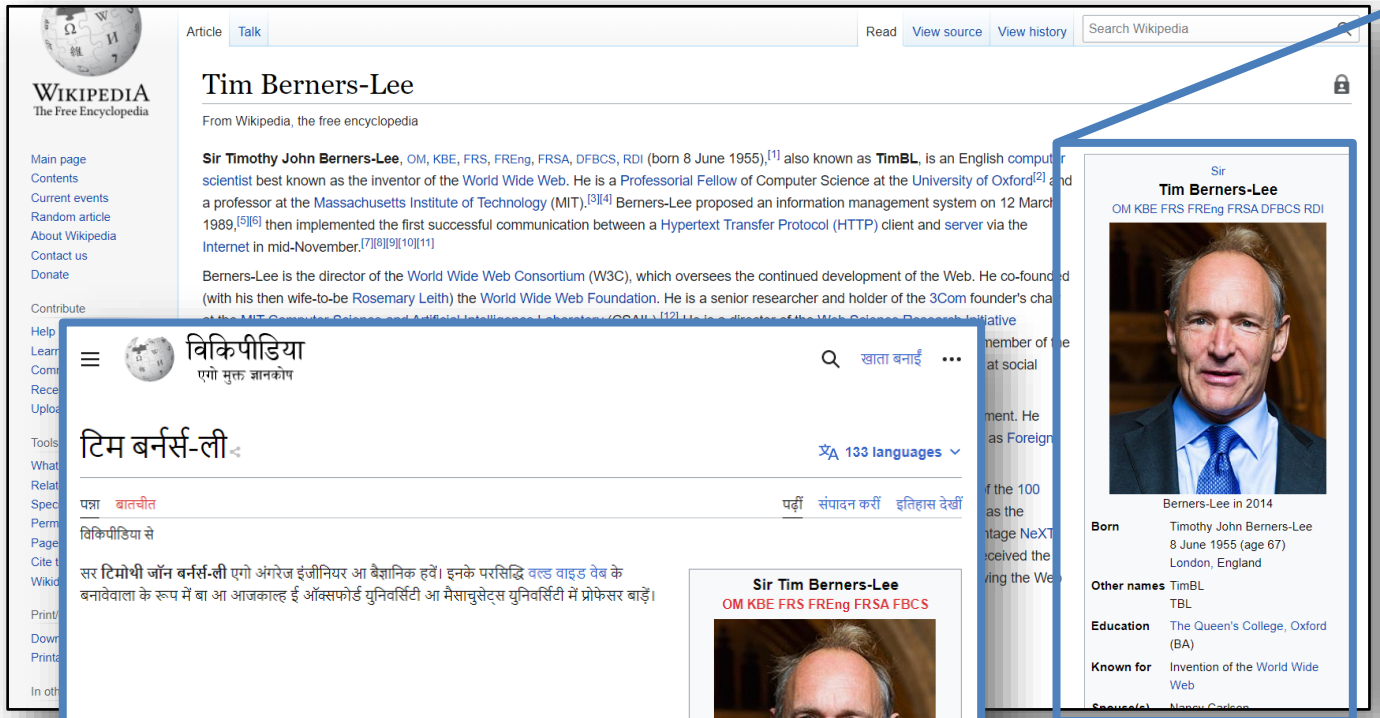
Initial goal:

Support multilingual infoboxes in Wikipedia



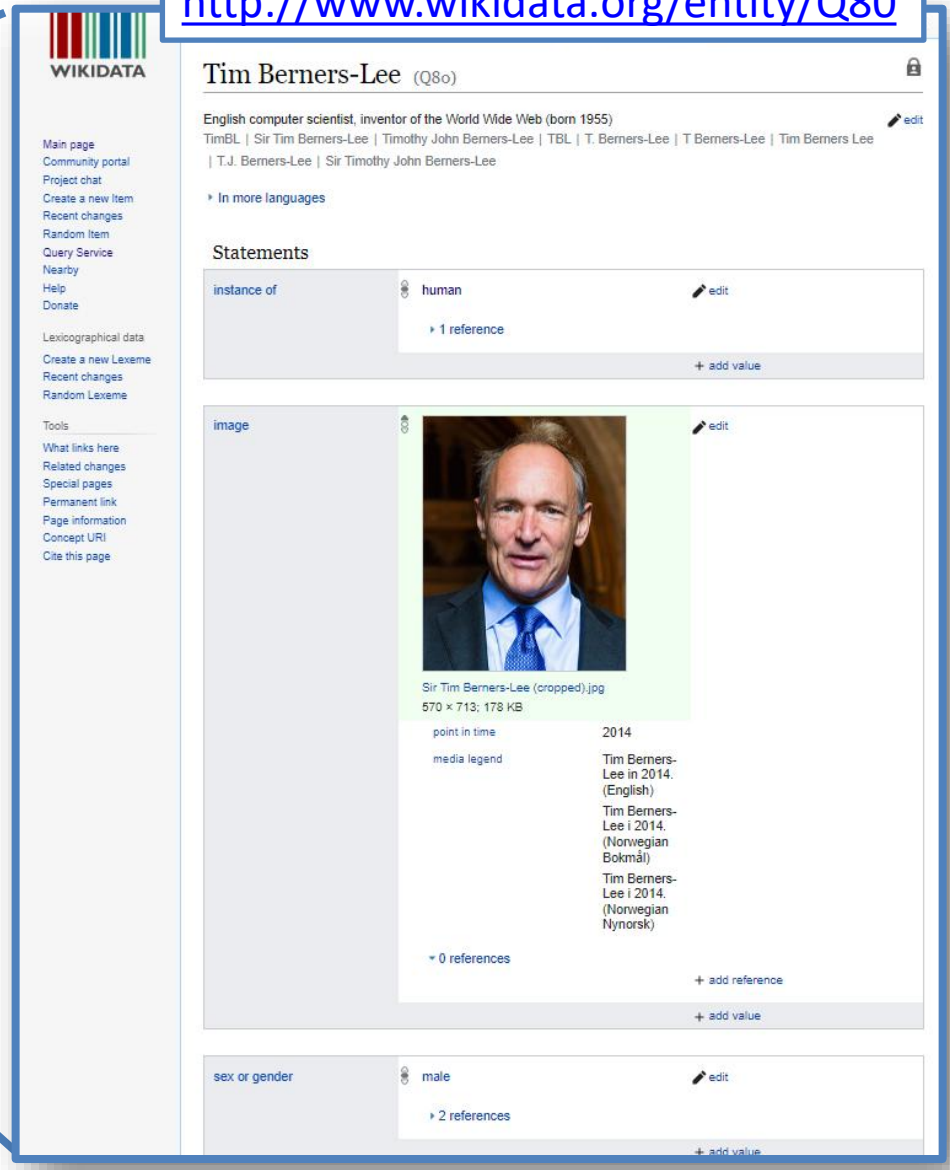
Wikidata

English Wikipedia page of Tim Berners-Lee



Bihari Wikipedia of Tim Berners-Lee

http://www.wikidata.org/entity/Q80



Wikidata

Wikidata as a commons Knowledge Graph

119,272,346 items, 2,423,893,283 edits (11/2024):
https://www.wikidata.org/wiki/Wikidata:Statistics

Free and open license: CC0

Multilingual

Collaborative (Humans and bots)

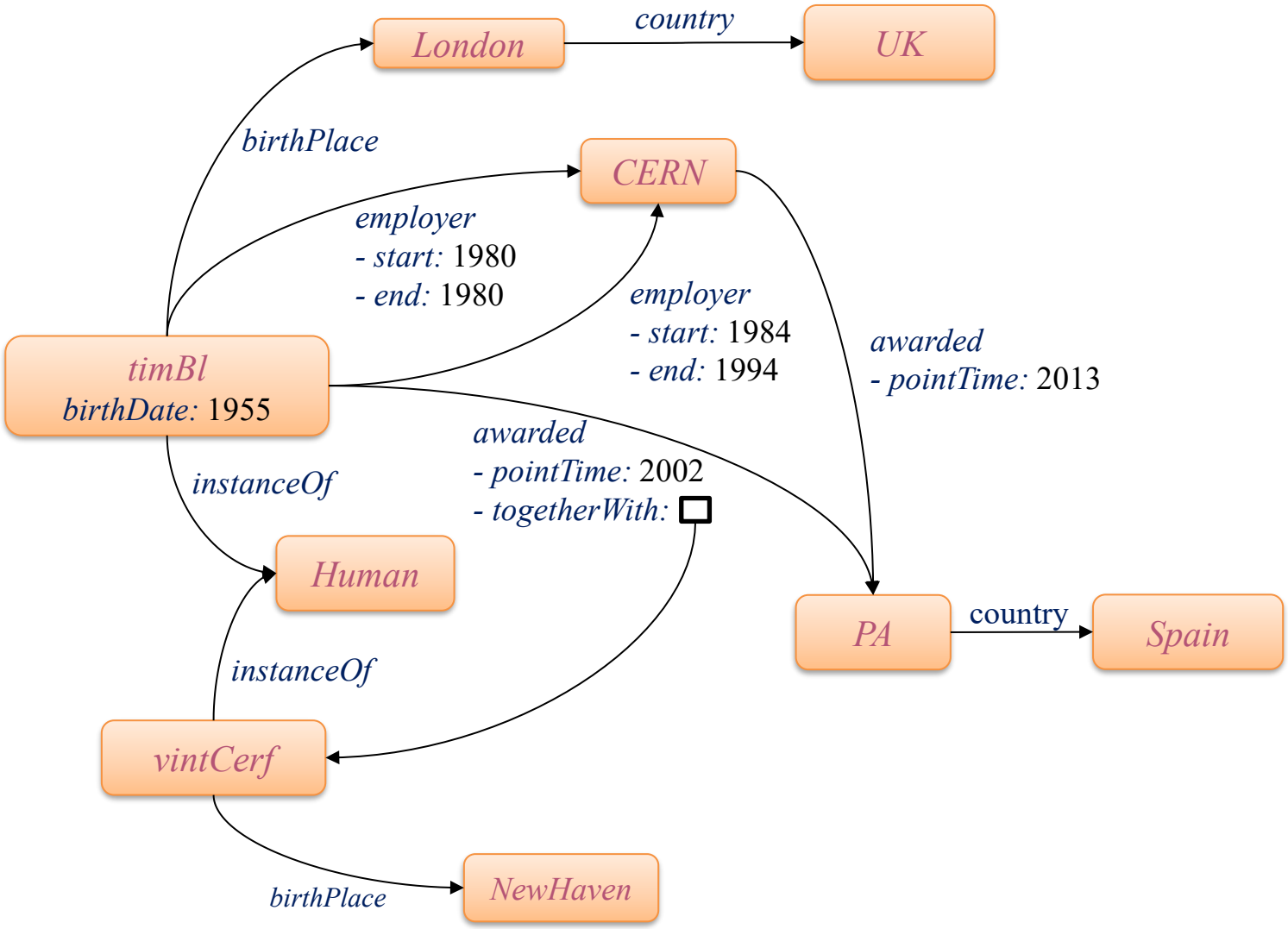
Public SPARQL endpoint

Software that supports Wikidata = Wikibase

Example from Wikidata

Information about Tim Berners Lee

http://www.wikidata.org/entity/Q80



Wikibase

- Wikibase: Software suite that implements Wikidata (https://wikiba.se/)
  - Set of extensions of Media Wiki (https://www.mediawiki.org/wiki/Wikibase)
    - Implemented in PHP (backend) + Javascript (frontend)
  - Can be hosted locally through Docker
  - Also available Cloud service: Wikibase.cloud (https://www.wikibase.cloud/)
- Wikibase instance: Application using Wikibase software
  - More info: Wikibase.world

Wikibase and RDF

Wikibase offers an RDF serialization of each entity

Several ways to get the RDF serialization*:

- SPARQL endpoint: Query service
- RDF Dumps
- Directly, for example: https://www.wikidata.org/wiki/Special:EntityData/Q80.ttl

RDF dump format defined in: RDF_Dump_Format specification

Several namespaces:

wd: for ítems

wdt: for properties

…

Custom reification model to serialize qualifiers and references

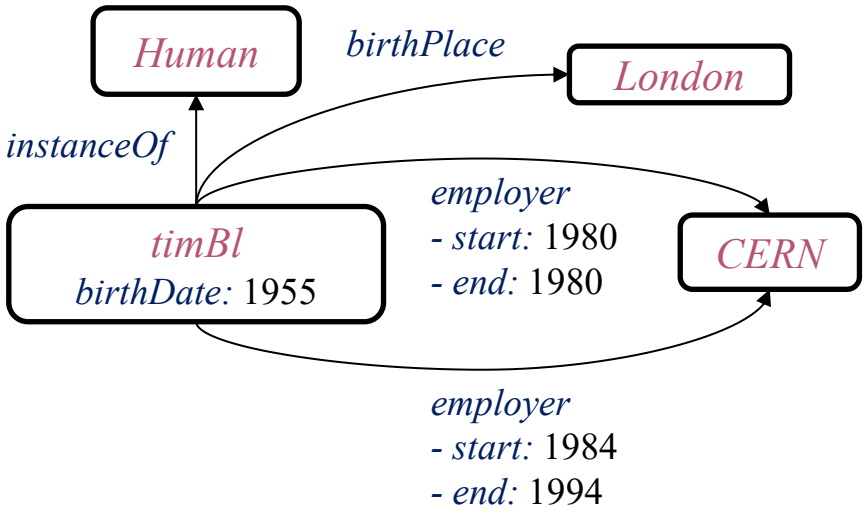
Serialization of complex datatypes requires several triples

*There can be small differences between the RDF serializations as described here

Wikibase and RDF (example)

Simplified RDF dump

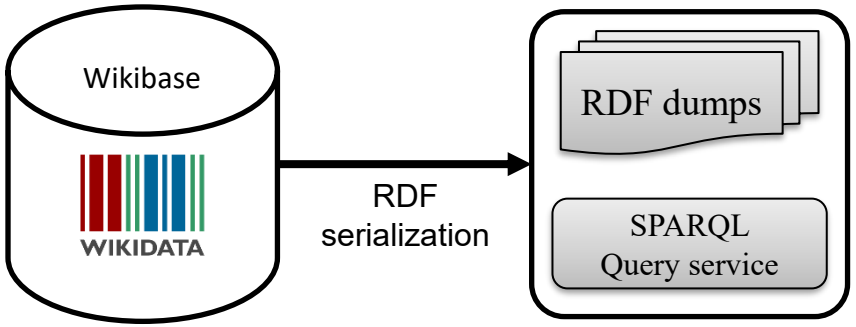
```
wd:Q80 a wikibase:Item ;
  wdt:P31  wd:Q5 ;
  wdt:P19  wd:Q84 ;
  wdt:P569 "1955-06-08T00:00:00Z"^^xsd:dateTime ;
  wdt:P108 wd:Q42944 ;
  p:P108   s:Q80-fcba864c, :Q80-4fe7940f
  #...
  .

:Q80-4fe7940f a wikibase:Statement ;
  ps:P108 wd:Q42944 ;
  pq:P580 "1984-01-01T00:00:00Z"^^xsd:dateTime ;
  pq:P582 "1994-01-01T00:00:00Z"^^xsd:dateTime .

s:Q80-fcba864c a wikibase:Statement ;
    ps:P108 wd:Q42944 ;
    pq:P580 "1980-06-01T00:00:00Z"^^xsd:dateTime ;
    pq:P582 "1980-12-01T00:00:00Z"^^xsd:dateTime .
```



timBl	wd:Q80
London	wd:Q84
Human	wd:Q5
CERN	wd:Q42944
birthDate	wdt:P569
instanceOf	wdt:P31
birthPlace	wdt:P19
employer	wdt:P108
start	pq:P580
end	pq:P582

Wikibase and RDF

Query service available: SPARQL endpoint




```
select ?name ?date?country where {
 wd:Q80 wdt:P1559 ?name .
 wd:Q80 wdt:P569  ?date .
 wd:Q80 wdt:P19   ?place .
 ?place wdt:P17   ?country
}
```

?name	?date	?country
Tim Berners-lee	1955-06-08	:UK

Try it: https://w.wiki/5yGu

RDF 1.2 (previously RDF-Star)

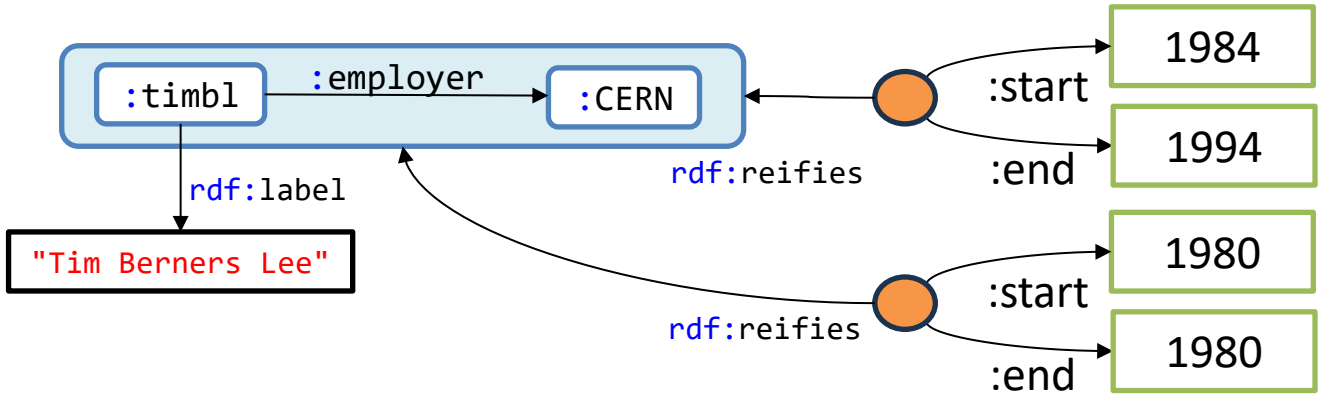
Currently under discussion (https://github.com/w3c/rdf-star-wg/wiki)

Add statements about triples (called triple terms)

Reifiers

```
:timbl rdfs:label "Tim Berners Lee“ ;
       :employer :CERN {| :start "1984" ;
                          :end   "1994" |}
                       {| :start "1980" ;
                          :end   "1980" |} .
```

```
:timbl rdfs:label "Tim Berners Lee" ;
       :employer  :CERN .
_:r rdf:reifies   <<( :timbl :employer :CERN )>> .
_:r :start        "1984" ;
    :end          "1994" .
_:s rdf:reifies   <<( :timbl :employer :CERN )>> .
_:s :start        "1980" ;
    :end          "1980" .
```



Note:
In this example, we are asserting that Tim's employer is CERN

RDF 1.2: Non-asserted triple terms

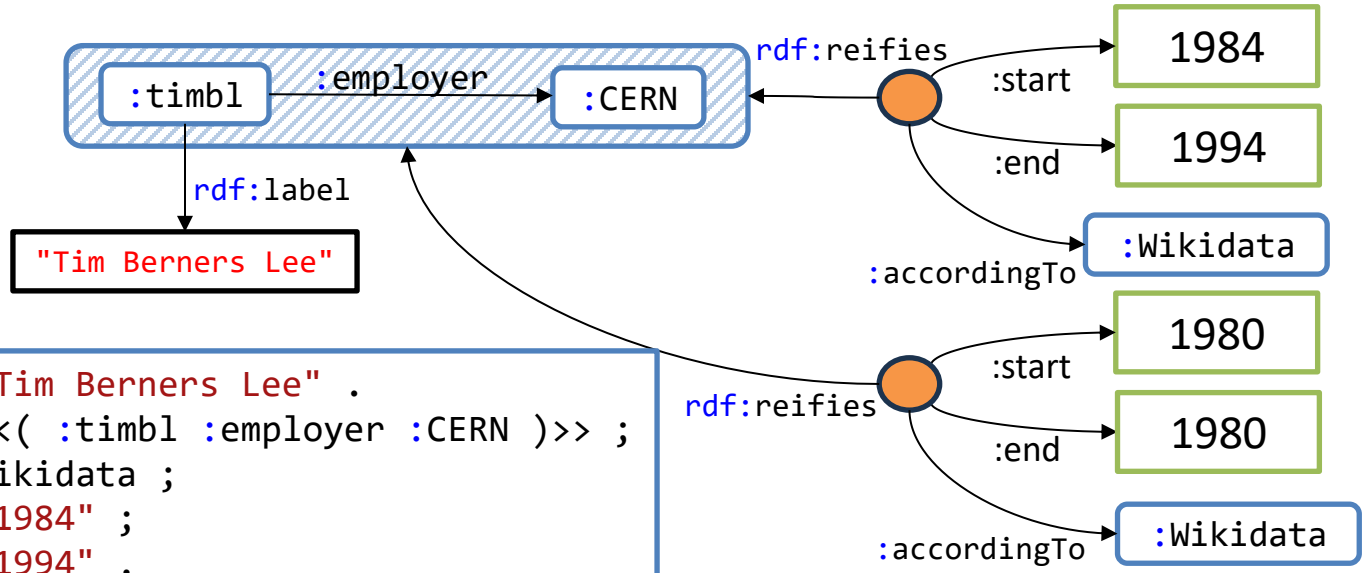
Triple terms can be reified without being asserted

We can say that Tim's employer is the CERN according to Wikidata

But we don't need to assert whether Tim's employer is the CERN or not

```
:timbl rdfs:label "Tim Berners Lee“ .
<< :timbl :employer :CERN >>
    :accordingTo :Wikidata ;
    :start       "1984" ;
    :end         "1994" .
<< :timbl :employer :CERN >>
    :accordingTo :Wikidata ;
    :start       "1980" ;
    :end         "1980" .
```

```
:timbl rdfs:label "Tim Berners Lee" .
_:1 rdf:reifies   <<( :timbl :employer :CERN )>> ;
    :accordingTo :Wikidata ;
    :start       "1984" ;
    :end         "1994" .
_:2 rdf:reifies   <<( :timbl :employer :CERN )>> ;
    :accordingTo :Wikidata ;
    :start       "1980" ;
    :end         "1980" .
```

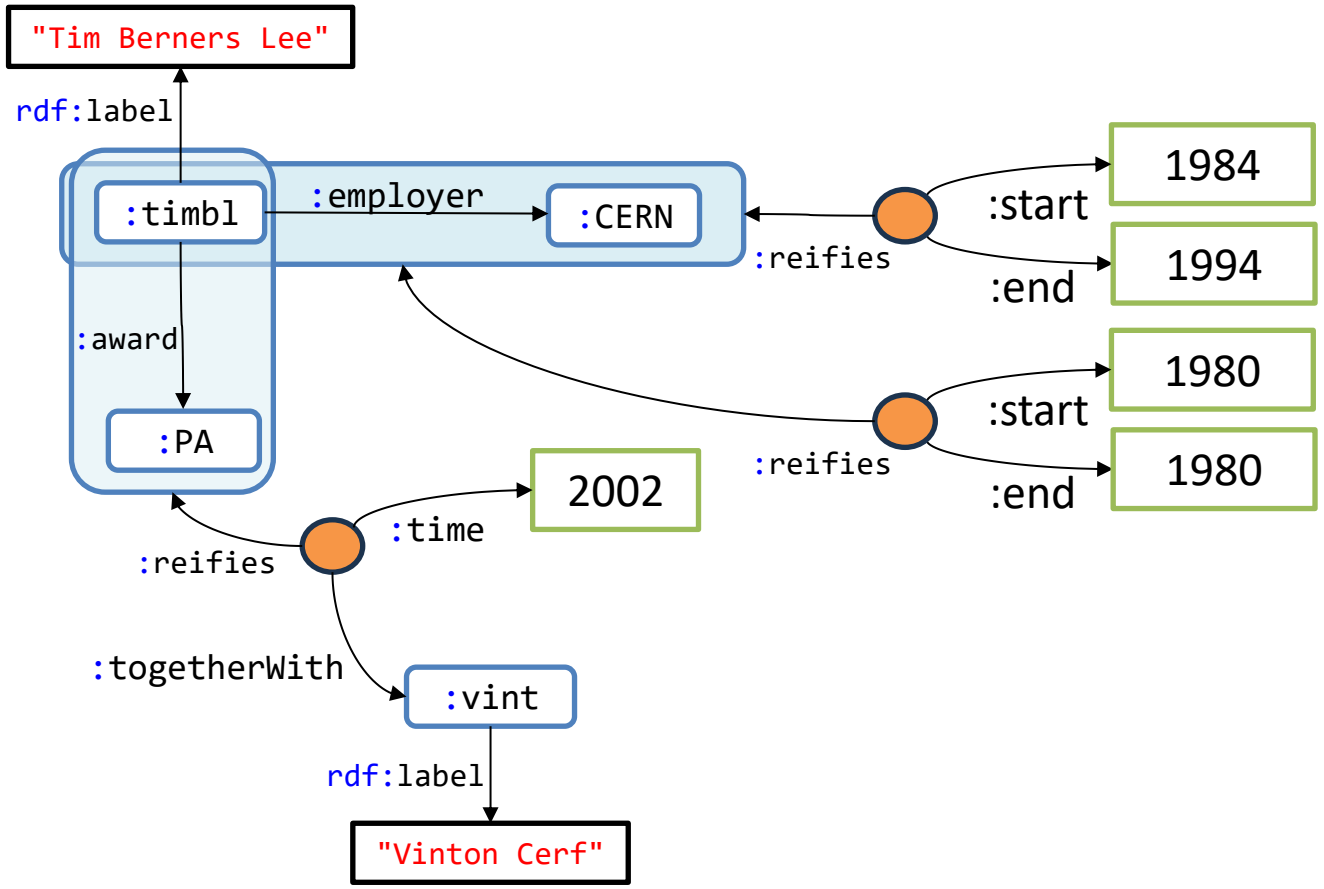


RDF 1.2

RDF 1.2 also supports Wikibase style models

```
:timbl rdfs:label "Tim Berners Lee" ;
       :employer :CERN {| :start         1984 ;
                          :end           1994 |}
                       {| :start         1980 ;
                          :end           1980 |} ;
       :award     :PA  {| :time          2002 ;
                          :togetherWith :vint |} .
:vint  rdfs:label "Vinton Cerf" .
```

```
:timbl rdfs:label "Tim Berners Lee" ;
       :employer  :CERN ; :award :PA.
_:r rdf:reifies   <<(:timbl :employer :CERN )>> ;
    :start        "1984" ;
    :end          "1994" .
_:s rdf:reifies   <<(:timbl :employer :CERN )>> ;
    :start        "1980" ;
    :end          "1980" .
_:t rdf:reifies   <<(:timbl :award :PA )>> ;
    :time         "2002" ;
    :togetherWith :vint  .
:vint rdfs:label  "Vinton Cerf" .
```



RDF 1.2

And things that Wikibase can not support, like qualifying qualifiers…

```
:timbl a :Human ;
   rdfs:label "Tim Berners Lee" ;
   :employer :CERN {| :start "1984"          {| :accordingTo :wikidata {| :statedIn "2024-05-02"^^xsd:date |} |} ;
                      :end   "1994“          {| :accordingTo :wikidata {| :statedIn "2024-05-02"^^xsd:date |} |}
                   |}
                   {| :start "1980"          {| :accordingTo :wikidata {| :statedIn "2024-05-01"^^xsd:date |} |} ;
                      :end   "1980"          {| :accordingTo :wikidata {| :statedIn "2024-05-01"^^xsd:date |} |}
                   |} ;
   :award    :PA   {| :year 2002             {| :accordingTo :wikidata {| :statedIn "2024-05-01"^^xsd:date |} |} ;
                      :together_with :vint {| :accordingTo :wikidata {| :statedIn "2024-05-01"^^xsd:date |} |}
                   |}
.
```

Do you want to draw that graph?

SPARQL 1.2

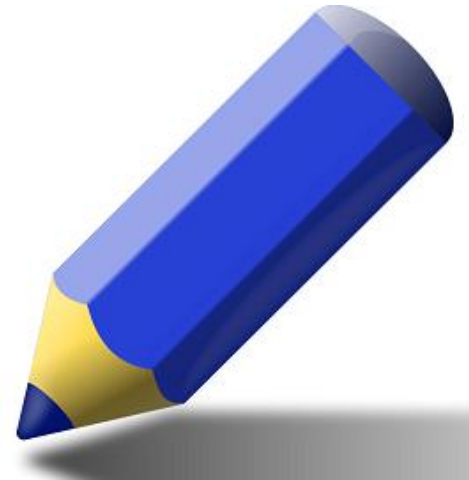
Current draft: https://www.w3.org/TR/sparql12-query/

Thanks to Oxigraph, rudof supports RDF 1.2 and SPARQL 1.2

Example query:

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

SELECT ?person ?employer ?start ?end WHERE {
   ?person a :Human ;
           :employer ?employer {| :start ?start ;
                                  :end   ?end
                               |}
}
```

Run some SPARQL 1.2 queries with rudof



Contents

WESO

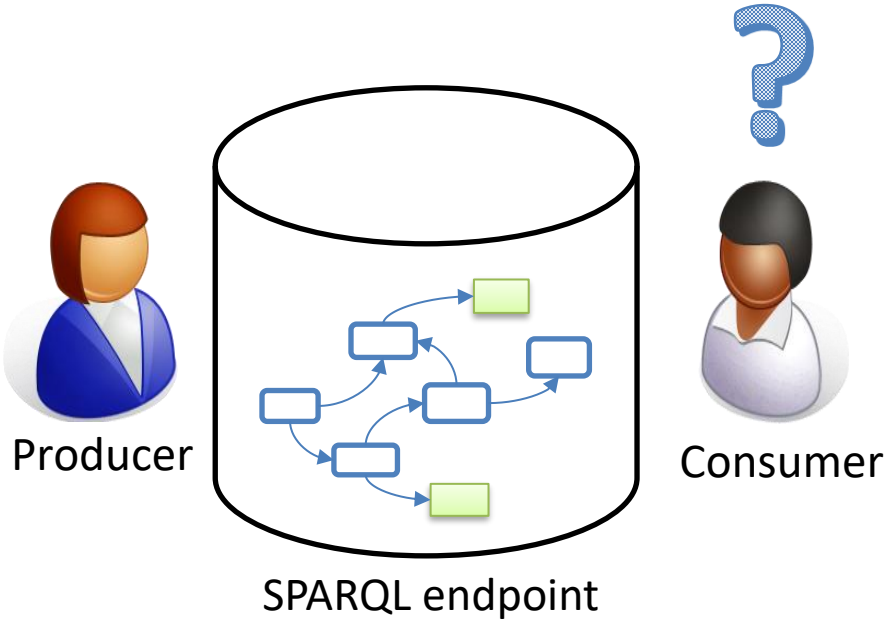
RDF, the good parts...

- RDF as an integration language
  - RDF as a *lingua franca* for semantic web and linked data
  - Basis for knowledge representation
- RDF flexibility
  - Data can be adapted to multiple environments
  - Reusable data by default
- RDF tools
  - RDF data stores & SPARQL
  - Several serializations: Turtle, JSON-LD, RDF/XML...
  - Can be embedded in HTML (Microdata/RDFa)

RDF, the other parts

Consuming & producing RDF

- Describing and validating RDF content
- SPARQL endpoints are not well documented
  - Typical documentation = set of SPARQL queries
  - Difficult to know where to start doing queries



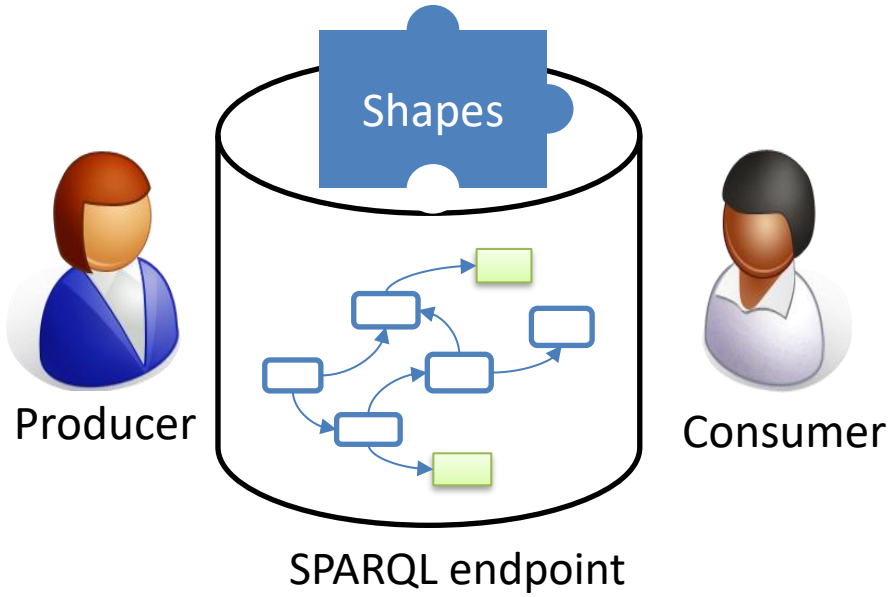
Why describe & validate RDF?

For producers

- Developers can understand the contents they are going to produce
- They can ensure they produce the expected structure
- Advertise and document the structure
- Generate interfaces

For consumers

- Understand the contents
- Verify the structure before processing it
- Query generation & optimization



Running example: some errors

Well formed RDF can contain some mistakes

Usually, RDF processors don't complain

RDF is very flexible

```
:timbl a              :Human              ;
       :name          "Tim Berners-Lee"   ;
       :knows         :bob                .

:alice a              :Human ;
       :name          234    .

:bob   a              :Human           ;
       :name          "Bob", "Robert"  ;
       :knows         :CERN            .

:carol  a             :Human  ;
        :birthPlace   :london .

:CERN   a             :Organization ;
        :name         "CERN"        .
```

Try it: https://rdfshape.weso.es/link/17497132139

In other technologies...

Technology	Schema
Relational Databases	DDL
XML	DTD, XML Schema, RelaxNG, Schematron
JSON	JSON Schema
RDF	?

Fill that gap

ShEx & SHACL

2013 RDF Validation Workshop

- Conclusions of the workshop:
  - *There is a need of a higher level, concise language for RDF Validation*
- ShEx initially proposed (v 1.0)

2014 W3c Data Shapes WG chartered

2017 SHACL accepted as W3C recommendation

2017 ShEx 2.0 released as W3C Community group draft

2019 ShEx adopted by Wikidata

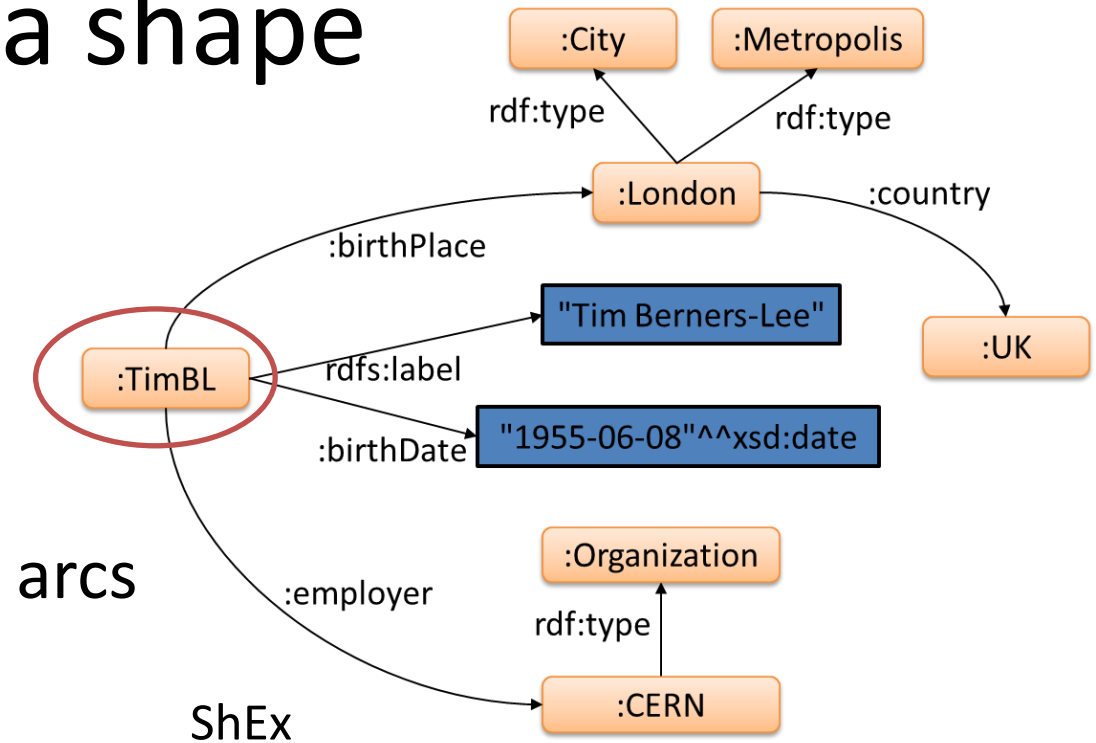
2025 IEEE ShEx (*work in progress*)

2025 Data Shapes WG chartered again for SHACL 1.2 (*work in progress*)

Example of a shape

A shape describes

- The form of a node (node constraint)
- Incoming/outgoing arcs from a node
- Possible values associated with those arcs



RDF Node

```
:timbl rdfs:label   "Tim Berners-Lee" ;
       :birthPlace  :london ;
       :birthDate   "1955-06-08"^^xsd:date ;
       :employer    :CERN .
```

ShEx

```
<Researcher> {
  :name        xsd:string          ;
  :birthPlace  @<Place>          ? ;
  :birthDate   xsd:date          ? ;
  :employer    @<Organization> * ;
}
```

Try it: https://rdfshape.weso.es/link/16685137872

Shape Expressions - ShEx

Goal: Concise and human-readable language

3 syntaxes:

- ShExC: Compact syntax, similar to Turtle or SPARQL
- ShExJ: JSON(-LD), for the spec
- ShExR: RDF, based on JSON-LD

Note: Round tripping is possible, convert from one to the other

Semantics inspired by regular expressions

ShEx libraries and demos

Implementations & libraries:

- shex.js: Javascript
- Jena-ShEx: Java
- SHaclEX: Scala (Jena/RDF4j)
- PyShEx: Python
- shex-java: Java
- Ruby-ShEx: Ruby
- RDF-Elixir: Elixir
- rudof: Rust 

Online demos & playgrounds

- ShEx-simple
- RDFShape
- Wikishape
- ShEx-Java
- ShExValidata

More info: http://shex.io

Simple example

Prefix declarations as in Turtle/SPARQL

```
prefix :       <http://example.org/>
prefix xsd:    <http://www.w3.org/2001/XMLSchema#>
prefix schema: <http://schema.org/>

:Book IRI AND {
 schema:name   xsd:string   ;
 :related      @:Book     *
}
```

Nodes conforming to `:Book` must

- Be `IRI`s and
- Have property `schema:name` with a value of type `xsd:string` (exactly one)
- Have property `:related` with values conforming to `:Book` (zero or more)

RDF Validation using ShEx

Schema

```
:Book IRI AND {
 :name    xsd:string   ;
 :related @:Book      *
}
```

RDF Data

```
:a   :name    "Title A" ;
     :related :b  .
:b   :related :a ;
     :name    "Title B".
:c   :name    "Title C1", "Title C2" .
:d   :name    234 .
:e   :namme   "Title E" .
:f   :name    "Title F" ;
     :related :a, _:1 .
_:1  :name    "Unknown title" .
```

Shape map

```
:a@:Book   ✓
:b@:Book,  ✓
:c@:Book,  ✘
:d@:Book,  ✘
:e@:Book,  ✘
:f@:Book   ✘
```

Try it (RDFShape)
Try it (Simple ShExDemo)

Validation process

Input: RDF data, ShEx schema, Shape map

Output: Result shape map

ShEx Schema

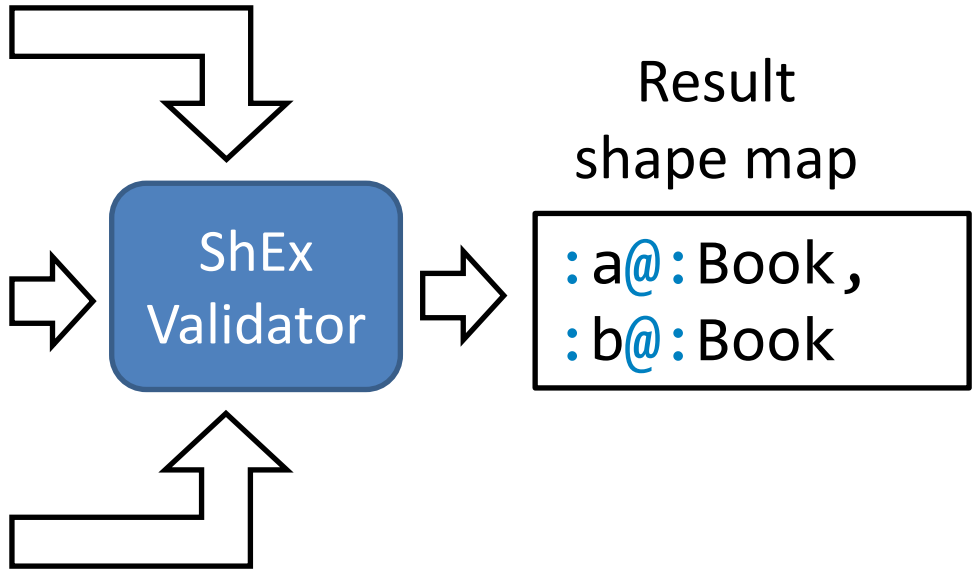
```
:Book IRI AND {
 :name     xsd:string   ;
 :related @:Book     *
}
```

Shape map

```
:a@:Book
```

RDF data

```
:a   :name     "Title A" ;
     :related  :b   .

:b   :related  :a ;
     :name     "Title B".
```



Result shape map

```
:a@:Book,
:b@:Book
```

Node constraints

Constraints over a node (without considering its neighbourhood)

```
:Book {
 :name           xsd:string
 :datePublished  xsd:date
 :numberOfPages  MinInclusive 1
 :author         @:Person
 :genre          [:Fiction :NonFiction ] ;
 :isbn           /isbn:[0-9X]{10}/
 :publisher      IRI
 :audio          .
 :maintainer     @:Person OR
                 @:Organization
}
:Person {}
:Organization {}
```

```
:item23
 :name           "Weaving the Web"                ;
 :datePublished  "2012-03-05"^^xsd:date           ;
 :numberOfPages  272                              ;
 :author         :timbl                           ;
 :genre          :NonFiction                      ;
 :isbn           "isbn:006251587X"                ;
 :publisher      <http://www.harpercollins.com/>  ;
 :audio          <http://audio.com/item23>        ;
 :maintainer     :alice                           .
```

Try it:

Cardinalities

Inspired by regular expressions: +, ?, *, {m,n}

By default {1,1}

```
:Book {
 :name           xsd:string          ;
 :numberOfPages  xsd:integer  ?      ;
 :author         @:Person     +      ;
 :publisher      IRI          ?      ;
 :maintainer     @:Person     {1,3}  ;
 :related        @:Book       *
}
:Person {}
:Organization {}
```

```
:item23
  :name           "Weaving the Web"       ;
  :numberOfPages  272                     ;
  :author         :timbl, :markFischetti  ;
  :maintainer     :alice,:bob             .
```

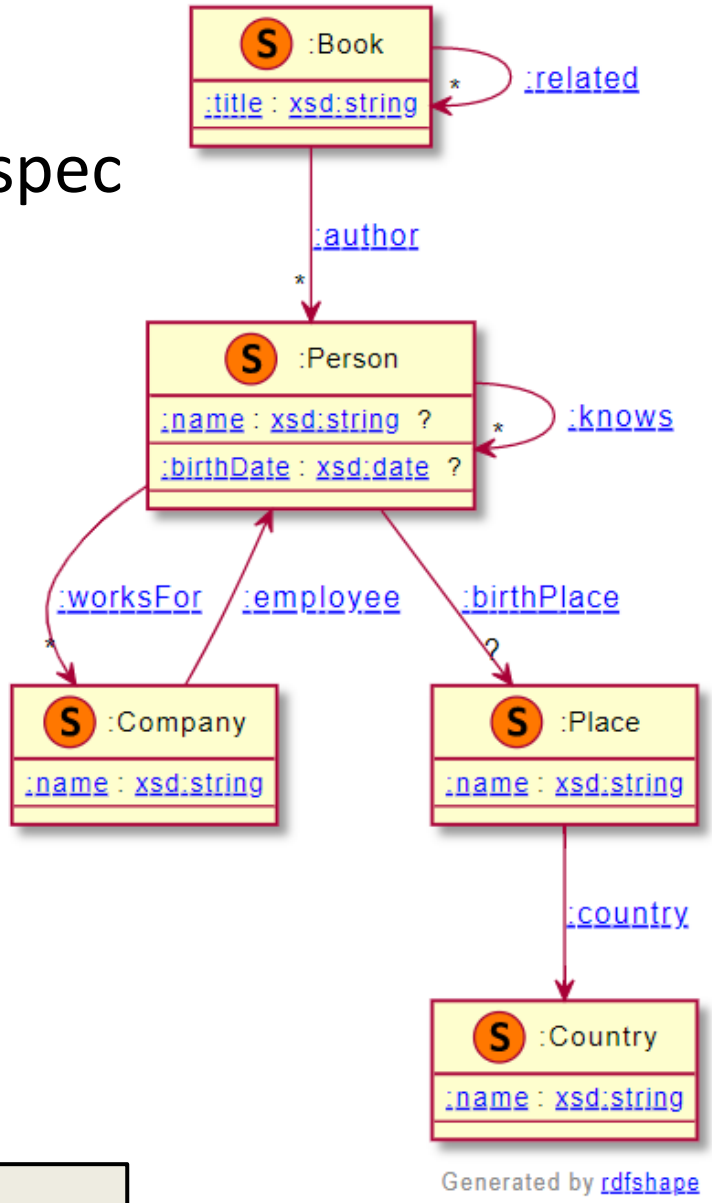
Try it: RDFShape

Recursive schemas

Support for recursive (cyclic) data models is part of the spec

Well formed semantics based on stratified negation

```
:Book    {  :title        xsd:string    ;
            :author       @:Person    * ;
            :related      @:Book      * }
:Person  {  :name         xsd:string  ? ;
            :birthDate    xsd:date    ? ;
            :birthPlace   @:Place     ? ;
            :knows        @:Person    * ;
            :worksFor     @:Company   * }
:Place    { :name         xsd:string    ;
            :country      @:Country     }
:Country  { :name         xsd:string    }
:Company  { :name         xsd:string    ;
            :employee     @:Person      }
```



Try it: RDFShape

Open/Closed content models

RDF semantics mostly presume open content models

Shape expressions are open by default

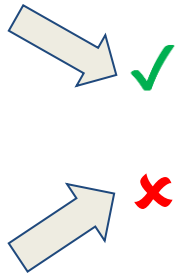
Enable extensibility

But…some use cases require closed content models

Added CLOSED keyword

```
:Book {
 :name     xsd:string   ;
}
```

```
:Book CLOSED {
 :name     xsd:string   ;
}
```



```
:a :name "Weaving the web" ;
   :isbn "006251587X"      .
```

Try it: RDFShape

Open/Closed properties

Property values are closed by default (closed properties)

```
:Book {
 :code /isbn:[0-9X]{10}/ ;
}
```

✓
```
:item23  :code "isbn:006251587X" .
```

✗
```
:item23  :code 23                .
```

Properties can be repeated

```
:Book {
  :code  /isbn:[0-9X]{10}/ ;
  :code  /isbn:[0-9]{13}/
}
```

```
:item23 :code "isbn:006251587X"    ,
        :code "isbn:9780062515872" .
```

EXTRA declares properties as open

```
:Book EXTRA :code {
 :code /isbn:[0-9X]{10}/ ;
}
```

```
:item23 :code "isbn:006251587X" ,
        :code 23                .
```

Try it: RDFShape

Triple expressions

“Unordered” regular expressions: Regular bag expressions

```
:Person {
 ( :name       xsd:string |
   :firstName xsd:string  + ;
   :lastName  xsd:string
 ) ;
 :birthDate   xsd:date    ?
}
```

```
(name |
  firstName + ;
  lastName
) ;
birthDate ?
```

```
(n | f + ; l ) ; b?
```

```
:alice :name       "Alice Cooper"           ;
       :birthDate "2010-02-23"^^xsd:date .

:bob   :firstName "Robert"                  ;
       :lastName  "Smith"                   .

:carol :firstName "Carol"                   ;
       :lastName  "King"                    ;
       :firstName "Carole"                  .
```

```
:alice → n
:bob   → f l
:carol → f l f
```

```
:dave  → n f
```

```
:dave :name       "Dave Navarro"            ;
      :firstName "Dave"                     .
```

Try it: RDFShape

Logical operators

Shape Expressions can be combined with AND, OR, NOT

```
:Book {
 :name   xsd:string ;
 :author @:Person OR @:Organization ;
}

:AudioBook @:Book AND {
 :name           MaxLength 20 ;
 :readBy         @:Person     ;
} AND NOT {
 :numberOfPages . +
}

:Person {}
:Organization {}
```

```
:item24 :name          "Weaving the Web" ;
        :author        :timbl            ;
        :readBy        :timbl            .
```

```
:item23 :name          "Weaving the Web" ;
        :author        :timbl            ;
        :numberOfPages 272               ;
        :readBy        :timbl            .
```

ShEx allows negation (NOT) combined with recursion (semantics based on stratified negation)

Try it: RDFShape

Importing schemas

`import` statement can be used to import schemas

http://validatingrdf.com/tutorial/examples/book.shex

```
:Book {
  :title    xsd:string   ;
}
:Person {
  :name     xsd:string ? ;
}
```

```
import <https://www.validatingrdf.com/examples/book.shex>

:AudioBook @:Book AND {
 :title          MaxLength 20 ;
 :readBy         @:Person     ;
}
```

```
:item24 :name          "Weaving the Web" ;
        :author        :timbl            ;
        :readBy        :timbl            .
```

Try it: RDFShape

Machine processable annotations

Annotations based on RDF

Lots of applications, e.g. generate forms from shapes

```
prefix schema: <http://schema.org/>
prefix : <http://example.org/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix ui: <http://www.w3.org/ns/ui#>

start = @:User

:User {
 schema:name      xsd:string                    // ui:label “Name"       ;
 schema:birthDate xsd:dateTime                  // ui:label "Birth date" ;
 schema:gender    [ schema:Male schema:Female ] // ui:label "Gender"
}
```

Try it:
Eric’s demo
https://rdfshape.weso.es/link/17095003321

More complex Shape Maps

Shape Maps define which nodes validate with which shapes

Examples:

`{FOCUS a :Person}@:User`	https://rdfshape.weso.es/link/17095033013
`{FOCUS schema:name _}@:Book`	https://rdfshape.weso.es/link/17095014386

`SPARQL """`
`PREFIX schema:`
`<http://schema.org/>`
`SELECT ?book WHERE {`
`?book schema:name ?name;`
`schema:author ?author`
`}"""@:Book`
	https://rdfshape.weso.es/link/17095031848

Inheritance model for ShEx

`extends` allows to reuse existing shapes adding new content

Handles closed properties and shapes

```
:Book {
 :name    xsd:string ;
 :author  @:Person   ;
 :code    /isbn:[0-9]{13}/   ;
 :code    /isbn:[0-9X]{10}/
}

:LibraryBook extends @:Book {
 :code    /internal:[0-9]*/ ;
}
```

```
:item23 :name       "Weaving the Web"    ;
        :author     :timbl               ;
        :code       "isbn:006251587X"    ;
        :code       "isbn:9780062515872" ;
        :code       "internal:234"       .
```

Other features

- Multiple inheritance
- Abstract shapes

More details: ***Shape Expressions with Inheritance***, Iovka Boneva, Jose Emilio Labra Gayo, Eric Prud'hommeaux, Katherine Thornton, Andra Waagmeester
Extended Semantic Web Conference, Portoroz, Slovenia, 2025 – 2025
https://labra.weso.es/publication/2025_shex_inheritance/

Try it:
https://rdfshape.weso.es/link/17487753484

3 syntaxes: ShExC, ShExJ, ShExR

ShExC

```
prefix :       <http://example.org/>
prefix xsd:
<http://www.w3.org/2001/XMLSchema#>
prefix schema: <http://schema.org/>

:Book {
 schema:name   xsd:string   ;
 :related      @:Book     *
}
```

ShExR (RDF, Turtle)

```
:Book a sx:ShapeDecl ;
  sx:shapeExpr [ a sx:Shape ;
   sx:expression [ a sx:EachOf ;
    sx:expressions (
      [ a sx:TripleConstraint ;
        sx:predicate schema:name ;
        sx:valueExpr  [ a sx:NodeConstraint ;
                        sx:datatype xsd:string
                      ] ]
      [ a sx:TripleConstraint ;
        sx:predicate :related ;
        sx:valueExpr  [ a sx:NodeConstraint ;
        sx:valueExpr :Book ] ] ) ] ] .
```

It's posible to roudtrip from each one

ShExJ (JSON LD)

```
{ "type" : "Schema",
  "@context" : "http://www.w3.org/ns/shex.jsonld",
  "shapes" : [ {
      "type" : "Shape",
      "id" : "http://example.org/Book",
      "expression" : {
        "type" : "EachOf",
        "expressions" : [ {
            "type" : "TripleConstraint",
            "predicate" : "http://schema.org/name",
            "valueExpr" : {
              "type" : "NodeConstraint",
              "datatype" : "http://www.w3.org/2001/XMLSchema#string"
            }
          },
          { "predicate" : "http://example.org/related",
            "valueExpr" : "http://example.org/Book",
            "min" : 0,
            "max" : -1,
            "type" : "TripleConstraint"
          }
        ] } } ] }
```

SHACL

W3C recommendation: https://www.w3.org/TR/shacl/ (July 2017)

Inspired by SPIN, OSLC & ShEx

2 parts: SHACL-Core, SHACL-SPARQL

Syntax = RDF vocabulary

ShEx and SHACL

ShEx

```
<Researcher> {
 :name        xsd:string        ;
 :birthPlace  @:Place         ? ;
 :birthDate   xsd:date        ? ;
 :employer    @:Organization  * ;
 :knows       @:Person        *
}
```

SHACL

```
<Researcher> a sh:NodeShape ;
 sh:targetClass :Human ;
 sh:property [ sh:path rdfs:label ;
          sh:minCount 1; sh:maxCount 1;
          sh:datatype xsd:string ;
         ] ;
 sh:property [ sh:path :birthPlace ;
          sh:node   <Place>
         ] ;
 sh:property  [ sh:path :birthDate ;
          sh:maxCount 1;
          sh:datatype xsd:date
         ] ;
 sh:property [ sh:path :employer ;
          sh:node   <Organization>
         ] ;
 sh:property [ sh:path :knows ;
          sh:node   <Researcher>
         ] .
```

Note:
Cyclic data models are implementation dependent in SHACL

Try it: https://tinyurl.com/yh28fsct

ShEx and SHACL

Comparing ShEx (compact syntax) with SHACL (compact syntax*)

ShEx

```
:Researcher {
 :name        xsd:string        ;
 :birthPlace @:Place          ? ;
 :birthDate   xsd:date        ? ;
 :employer    @:Organization  * ;
 :knows       @:Researcher    *
}
```

SHACL

```
shape @:Researcher -> :Human {
 rdfs:label  xsd:string     [1..1] ;
 :birthPlace @:Place                ;
 :birthDate  xsd:date       [1..]  ;
 :employer   @:Organization         ;
 :knows      @:Researcher
}
```

Note: SHACL Compact syntax is being discussed for SHACL 1.2
More info: https://w3c.github.io/shacl/shacl-compact-syntax/

Some SHACL implementations

Name	Parts	Language - Library	Comments
Topbraid SHACL API	SHACL Core, SPARQL	Java (Jena)	Used by TopBraid composer
SHACL playground	SHACL Core	Javascript (rdflib.js)	http://shacl.org/playground/
SHACL-S (SHaclEX)	SHACL Core	Scala (Jena, RDF4j)	http://rdfshape.weso.es
pySHACL	SHACL Core, SPARQL	Python (rdflib)	https://github.com/RDFLib/pySHACL
Corese SHACL	SHACL Core, SPARQL	Java (STTL)	http://wimmics.inria.fr/corese
RDFUnit	SHACL Core, SPARQL	Java (Jena)	https://github.com/AKSW/RDFUnit
Jena SHACL	SHACL Core, SPARQL	Java (Jena)	https://jena.apache.org/
RDf4j SHACL	SHACL Core	Java (RDF4J)	https://rdf4j.org
Stardog	SHACL Core, SPARQL	Java	https://www.stardog.com
Zazuko SHACL	SHACL Core	Javascript	https://github.com/zazuko/rdf-validate-shacl
maplib	SHACL Core	Rust	https://github.com/DataTreehouse/maplib
rudof NEW	SHACL core (in progress)	Rust	https://rudof-project.github.io/

RDFShape online demo supports: SHaclEX (SHACL-s), JenaSHACL, SHACL TQ (SHACL TopBraid API)

Basic example

```
prefix :       <http://example.org/>
prefix sh:     <http://www.w3.org/ns/shacl#>
prefix xsd:    <http://www.w3.org/2001/XMLSchema#>
prefix schema: <http://schema.org/>

:UserShape a sh:NodeShape ;
   sh:targetNode :alice, :bob, :carol ;
   sh:nodeKind sh:IRI ;
   sh:property :hasName,
               :hasEmail .
:hasName sh:path schema:name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string .
:hasEmail sh:path schema:email ;
   sh:minCount 1;
   sh:maxCount 1;
   sh:nodeKind sh:IRI .
```

Shapes graph

```
:alice schema:name "Alice Cooper" ;
       schema:email <mailto:alice@mail.org> .

:bob   schema:firstName "Bob" ;
       schema:email <mailto:bob@mail.org> .

:carol schema:name "Carol" ;
       schema:email "carol@mail.org" .
```

☹ ☹

Data graph

Try it. RDFShape https://tinyurl.com/y46b2f8q

Same example with blank nodes

```
prefix :       <http://example.org/>
prefix sh:     <http://www.w3.org/ns/shacl#>
prefix xsd:    <http://www.w3.org/2001/XMLSchema#>
prefix schema: <http://schema.org/>

:UserShape a sh:NodeShape ;
   sh:targetNode :alice, :bob, :carol ;
   sh:nodeKind sh:IRI ;
   sh:property [
    sh:path      schema:name ;
    sh:minCount 1; sh:maxCount 1;
    sh:datatype xsd:string ;
  ] ;
  sh:property [
   sh:path      schema:email ;
   sh:minCount 1; sh:maxCount 1;
   sh:nodeKind sh:IRI ;
  ] .
```

Shapes graph

```
:alice schema:name "Alice Cooper" ;
       schema:email <mailto:alice@mail.org> .

:bob   schema:firstName "Bob" ;
       schema:email <mailto:bob@mail.org> .

:carol schema:name "Carol" ;
       schema:email "carol@mail.org" .
```

☹ ☹

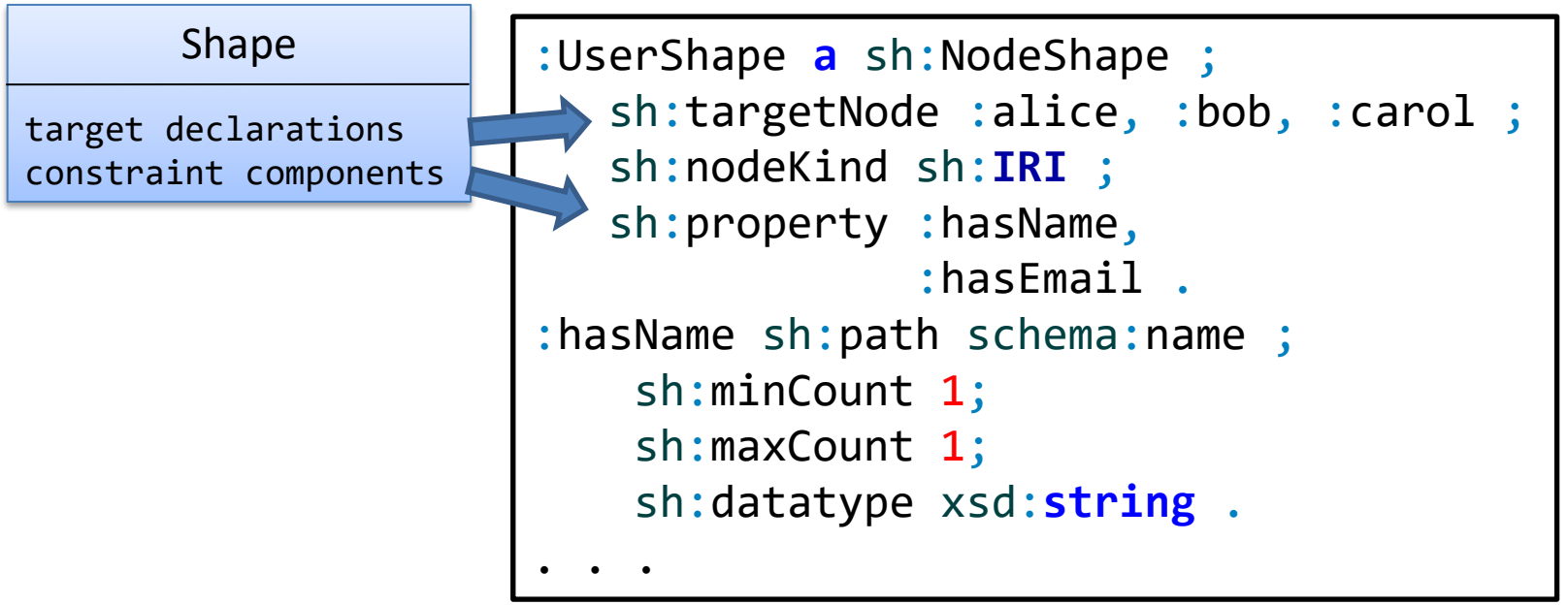
Data graph

Try it. RDFShape https://tinyurl.com/y4ycv2vn

Some definitions about SHACL

Shape: collection of targets and constraints components

- Targets: specify which nodes in the data graph must conform to a shape
- Constraint components: Determine how to validate a node



Shapes graph and data graph

Conceptually: 2 graphs

- Shapes graph: an RDF graph that contains shapes
- Data graph: an RDF graph that contains data to be validated

Note: They can be the same

```
:UserShape a sh:NodeShape ;
   sh:targetNode :alice, :bob, :carol ;
   sh:nodeKind sh:IRI ;
   sh:property :hasName,
               :hasEmail .
:hasName sh:path schema:name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string .
. . .
```

Shapes graph

```
:alice schema:name "Alice Cooper" ;
       schema:email <mailto:alice@mail.org> .

:bob   schema:firstName "Bob" ;
       schema:email <mailto:bob@mail.org> .

:carol schema:name "Carol" ;
       schema:email "carol@mail.org" .
```

Data graph

Validation Report

The output of the validation process is a list of violation errors

No errors ⇒ RDF conforms to shapes graph

```
[ a              sh:ValidationReport ;
  sh:conforms    true
].
```

```
[ a              sh:ValidationReport ;
  sh:conforms    false ;
  sh:result      [
   a               sh:ValidationResult ;
   sh:focusNode    :bob ;
   sh:message
     "MinCount violation. Expected 1, obtained: 0" ;
   sh:resultPath schema:name ;
   sh:resultSeverity sh:Violation ;
   sh:sourceConstraintComponent
     sh:MinCountConstraintComponent ;
   sh:sourceShape  :hasName
] ;
...
```

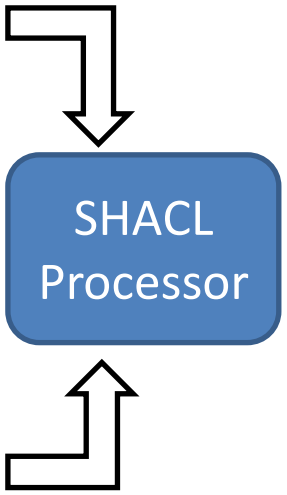
SHACL processor

Shapes graph with target declarations

```
:UserShape a sh:NodeShape ;
   sh:targetNode :alice, :bob, :carol ;
   sh:nodeKind sh:IRI ;
   sh:property :hasName,
               :hasEmail .
:hasName sh:path schema:name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string .
. . .
```

Data Graph

```
:alice schema:name "Alice Cooper" ;
       schema:email <mailto:alice@mail.org>.

:bob   schema:name "Bob" ;
       schema:email <mailto:bob@mail.org> .

:carol schema:name "Carol" ;
       schema:email <mailto:carol@mail.org>
.
```



Validation report

```
[ a              sh:ValidationReport ;
  sh:conforms    true
].
```

Importing shapes graphs

SHACL processors follow `owl:imports` declarations

It extends the current shapes graph with the imported shapes

```
:UserShape a sh:NodeShape ;
   sh:targetNode :alice, :bob, :carol ;
   sh:nodeKind sh:IRI ;
   sh:property :hasName .
:hasName sh:path schema:name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string .
```

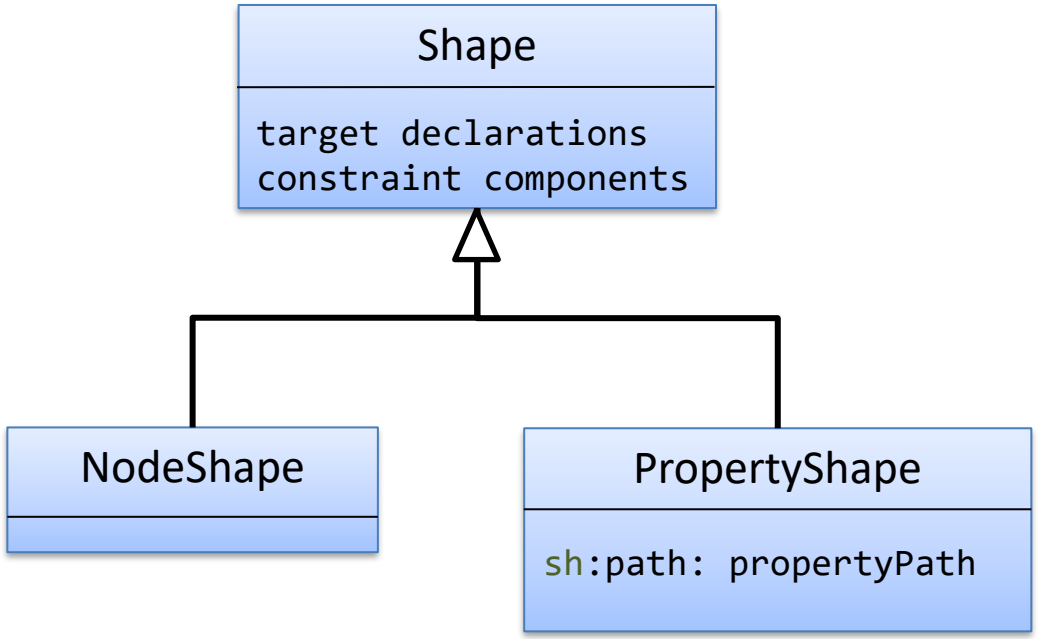
```
<> owl:imports <http://example.org/UserShapes> .

:TeacherShape a sh:NodeShape;
  sh:targetClass :Teacher ;
  sh:node :UserShape ;
  sh:property [
   sh:path :teaches ;
   sh:minCount 1;
   sh:datatype xsd:string;
 ] .
```

Node and property shapes

2 types of shapes:

NodeShape: constraints about shapes of nodes

PropertyShapes: constraints on property path values of a node



NodeShape

PropertyShape

```
:UserShape a sh:NodeShape ;
    sh:targetNode :alice, :bob, :carol ;
    sh:nodeKind sh:IRI ;
    sh:property :hasName,
                :hasEmail .
:hasName sh:path schema:name ;
     sh:minCount 1;
     sh:maxCount 1;
     sh:datatype xsd:string .
. . .
```

Node Shapes

Constraints about a focus node

```
:UserShape  a sh:NodeShape ;
      sh:nodeKind     sh:IRI ;
      sh:targetClass  :User .
```

```
:alice a :User .

<http://example.org/bob> a :User .

_:1 a :User .
```

☹

Try it: https://tinyurl.com/y6qyqo5g

Property shapes

Constraints about a given property and its values for the focus node

- `sh:property` associates a shape with a property shape
- `sh:path` identifies the path

```
:User a sh:NodeShape ;
   sh:property [
      sh:path     schema:email ;
      sh:nodeKind sh:IRI
   ] .
```

```
:alice a :User ;
       schema:email <mailto:alice@mail.org> .

:bob   a :User;
       schema:email <mailto:bob@mail.org> .

:carol a :User;
       schema:email "carol@mail.org" .
```

Paths in property shapes

Subset of SPARQL property paths using the following names:

- `inversePath`
- `alternativePath`
- `zeroOrMorePath`
- `oneOrMorePath`
- `zeroOrOnePath`

```
:User a sh:NodeShape, rdfs:Class ;
   sh:property [
    sh:path [sh:inversePath schema:follows ];
    sh:nodeKind sh:IRI ;
  ] .
```

```
:alice a :User;
       schema:follows :bob .

:bob   a :User .                    ☹

:carol a :User;
       schema:follows :alice .

_:1 schema:follows :bob .
```

Constraint components

- Constraints associated with shapes
  - They have parameters whose values specify the constraints
  - SHACL-core provides a list of predefined constraint components
    - Most of them have one parameter which identifies them

Convention:
Parameter: sh:xx
Constraint component: sh:xxConstraintComponent

```
:UserShape a             sh:NodeShape ;
           sh:nodeKind   sh:IRI .
```

Constraint component
sh:nodeKindConstraintComponent

Parameter
sh:nodeKind

Value of Parameter
sh:IRI ;

NOTE: Custom constraint components can be defined in SHACL-SPARQL

Repeated parameter

Each value of the parameter declares a different constraint

```
:UserShape a          sh:NodeShape;
           sh:class   foaf:Person ;
           sh:class   schema:Person .
```

```
:alice a schema:Person, foaf:Person .

:bob a schema:Person .
```

SHACL Core constraint components

Type	Constraints
Cardinality	`minCount, maxCount`
Types of values	`class, datatype, nodeKind`
Values	`node, in, hasValue, property`
Range of values	`minInclusive, maxInclusive` `minExclusive, maxExclusive`
String based	`minLength, maxLength, pattern`
Language based	`languageIn, uniqueLang`
Logical constraints	`not, and, or, xone`
Closed shapes	`closed, ignoredProperties`
Property pair constraints	`equals, disjoint, lessThan, lessThanOrEquals`
Non-validating constraints	`name, description, order, group`
Qualified shapes	`qualifiedValueShape, qualifiedValueShapesDisjoint` `qualifiedMinCount, qualifiedMaxCount`

See later

Human friendly messages

Message declares the message that will appear in the validation report in case of violation

```
:UserShape a sh:NodeShape ;
   sh:targetClass :User ;
   sh:property [
    sh:path       schema:name ;
    sh:minCount   1 ;
    sh:message    "Where is the name?"
  ] .
```

```
:bob    a :User ;
        schema:alias "Bob" . ☹
```

```
:report a :ValidationReport ;
 sh:conforms false ;
 sh:result [ a sh:ValidationResult ;
  sh:resultSeverity            sh:Violation ;
  sh:sourceConstraintComponent sh:MinCountConstraintComponent ;
  sh:sourceShape               ... ;
  sh:focusNode                 :bob ;
  sh:resultPath                schema:name ;
   sh:resultMessage            "Where is the name?" ;
].
```

Severities

Declare the level of the violation

- 3 predefined levels: Violation (default), Warning, Info

```
:UserShape a sh:NodeShape ;
 sh:targetClass :User ;
 sh:property [
    sh:path     schema:name ;
    sh:datatype xsd:string ;
    sh:severity sh:Warning
  ] .
```

```
:bob    a :User ;
        schema:alias "Bob" .
```

```
:report a :ValidationReport ;
 sh:conforms false ;
 sh:result [ a sh:ValidationResult ;
   sh:resultSeverity      sh:Warning ;
   sh:sourceConstraintComponent sh:MinCountConstraintComponent ;
   sh:sourceShape               ... ;
   sh:focusNode                 :bob ;
   sh:resultPath                schema:name ;
   sh:resultMessage             "MinCount Error" ;
].
```

Deactivating shapes

Deactivate a shape

Useful when importing shapes

UserShapes

```
:UserShape a sh:NodeShape;
 sh:targetClass :User ;
 sh:property :HasName ;
 sh:property :HasEmail .

:HasName sh:path schema:name ;
  sh:datatype xsd:string .

:HasEmail sh:path schema:email ;
  sh:minCount 1;
  sh:nodeKind sh:IRI .
```

```
<> owl:imports <UserShapes> .

:TeacherShape  a  sh:NodeShape;
   sh:targetClass :Teacher ;
   sh:node        :UserShape ;
   sh:property [ sh:path      :teaches ;
    sh:minCount  1;
    sh:datatype  xsd:string;
  ] .

:HasEmail sh:deactivated true .
```

Target declarations

Targets specify nodes that must be validated against the shape

Several types

Value	Description
`targetNode`	Directly point to a node
`targetClass`	All nodes that have a given type
`targetSubjectsOf`	All nodes that are subjects of some predicate
`targetObjectsOf`	All nodes that are objects of some predicate

Target node

Directly declare which nodes must validate the against the shape

```
:UserShape a sh:NodeShape ;
   sh:targetNode :alice, :bob, :carol ;
   sh:property [
    sh:path schema:name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string ;
  ] ;
  sh:property [
   sh:path schema:email ;
   sh:minCount 1;
   sh:maxCount 1;
   sh:nodeKind sh:IRI ;
  ] .
```

```
:alice schema:name "Alice Cooper" ;
       schema:email <mailto:alice@mail.org> .

:bob   schema:givenName "Bob" ;
       schema:email <mailto:bob@mail.org> .

:carol schema:name "Carol" ;
       schema:email "carol@mail.org" .
```

Target class

Selects all nodes that have a given class

Looks for rdf:type declarations*

```
:UserShape a sh:NodeShape ;
 sh:targetClass :User ;
 sh:property [
    sh:path schema:name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string ;
 ] ;
 sh:property [
   sh:path schema:email ;
   sh:minCount 1;
   sh:maxCount 1;
   sh:nodeKind sh:IRI ;
 ] .
```

```
:alice a :User;
       schema:name "Alice Cooper" ;
       schema:email <mailto:alice@mail.org> .

:bob   a :User;
       schema:givenName "Bob" ;
       schema:email <mailto:bob@mail.org> .

:carol a :User;
       schema:name "Carol" ;
       schema:email "carol@mail.org" .
```

* Also looks for rdfs:subClassOf*/rdf:type declarations

Implicit class target

A shape with type sh:Shape and rdfs:Class is a scope class of itself

The targetClass declaration is implicit

```
:User a sh:NodeShape, rdfs:Class ;
  sh:property [
     sh:path schema:name ;
     sh:minCount 1;
     sh:maxCount 1;
     sh:datatype xsd:string ;
  ] ;
  sh:property [
    sh:path schema:email ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:nodeKind sh:IRI ;
  ] .
```

```
:alice a :User;
       schema:name "Alice Cooper" ;
       schema:email <mailto:alice@mail.org> .

:bob   a :User;
       schema:givenName "Bob" ;
       schema:email <mailto:bob@mail.org> .

:carol a :User;
       schema:name "Carol" ;
       schema:email "carol@mail.org" .
```

targetSubjectsOf

```
:UserShape a sh:NodeShape;
 sh:targetSubjectsOf :teaches ;
 sh:property [
    sh:path schema:name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string ;
 ] .
```

```
:alice :teaches :Algebra ;      #Passes as :UserShape
       schema:name "Alice" .

:bob   :teaches :Logic ;        #Fails as :UserShape
       foaf:name "Robert" .

:carol foaf:name 23 .           # Ignored
```

targetObjectsOf

```
:UserShape a              sh:NodeShape;
 sh:targetObjectsOf :isTaughtBy ;
 sh:property [
    sh:path      schema:name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string ;
 ] .
```

```
:alice schema:name "Alice" . #Passes as :UserShape

:bob   foaf:name "Robert" .  #Fails as :UserShape

:carol foaf:name 23 .        # Ignored

:algebra :isTaughtBy :alice, :bob .
```

Core constraint components

Type	Constraints
Cardinality	`minCount, maxCount`
Types of values	`datatype, class, nodeKind`
Values	`node, in, hasValue`
Range of values	`minInclusive, maxInclusive` `minExclusive, maxExclusive`
String based	`minLength, maxLength, pattern, stem, uniqueLang`
Logical constraints	`not, and, or, xone`
Closed shapes	`closed, ignoredProperties`
Property pair constraints	`equals, disjoint, lessThan, lessThanOrEquals`
Non-validating constraints	`name, value, defaultValue`
Qualified shapes	`qualifiedValueShape, qualifiedMinCount, qualifiedMaxCount`

Cardinality constraints

Constraint	Description
`minCount`	Restricts minimum number of triples involving the focus node and a given predicate. Default value: 0
`maxCount`	Restricts maximum number of triples involving the focus node and a given predicate. If not defined = unbounded

```
:User a sh:NodeShape ;
   sh:property [
    sh:path     schema:follows ;
    sh:minCount 2 ;
    sh:maxCount 3 ;
  ] .
```

```
:alice schema:follows :bob,
                      :carol .

:bob   schema:follows :alice .   ☹

:carol schema:follows :alice,
                      :bob,      ☹
                      :carol,
                      :dave .
```

Try it. https://tinyurl.com/y5wjrowt

Datatypes of values

Constraint	Description
datatype	Restrict the datatype of all value nodes to a given value

```
:User a sh:NodeShape ;
   sh:property [
    sh:path     schema:birthDate ;
    sh:datatype xsd:date ;
  ] .
```

```
:alice schema:birthDate "1985-08-20"^^xsd:date .

:bob   schema:birthDate "Unknown"^^xsd:date .

:carol schema:birthDate 1990 .
```

Try it: https://tinyurl.com/y42ec72v

Class of values

Constraint	Description
`class`	Verify that each node in an instance of some class It also allows instances of subclasses*

(*) The notion of SHACL instance is different from RDFS
It is defined as `rdfs:subClassOf*/rdf:type`

```
:User a sh:NodeShape, rdfs:Class ;
 sh:property [
  sh:path  schema:follows ;
  sh:class :User
 ] .
```

```
:Manager rdfs:subClassOf :User .

:alice a :User;
       schema:follows :bob .
:bob   a :Manager ;
       schema:follows :alice .
:carol a :User;
       schema:follows :alice, :dave .
:dave  a :Employee .
```

☹

Kind of values

Constraint	Description

nodeKind	Possible values:
	`BlankNode, IRI, Literal,`
	`BlankNodeOrIRI, BlankNodeOrLiteral, IRIOrLiteral`

```
:User a sh:NodeShape, rdfs:Class ;
  sh:property [
   sh:path     schema:name ;
   sh:nodeKind sh:Literal ;
  ];
  sh:property [
   sh:path     schema:follows ;
   sh:nodeKind sh:BlankNodeOrIRI
  ];
  sh:nodeKind sh:IRI .
```

```
:alice a :User;
       schema:name    _:1 ;
       schema:follows :bob .

:bob   a :User;
       schema:name  "Robert";
       schema:follows [ schema:name "Dave" ] .

:carol a :User;
       schema:name    "Carol" ;
       schema:follows "Dave"  .

_:1 a :User .
```

☹

☹

☹

Constraints on values

Constraint	Description
hasValue	Verifies that the focus node has a given value
in	Enumerates the value nodes that a property may have

```
:User a sh:NodeShape, rdfs:Class ;
  sh:property [
   sh:path      schema:affiliation ;
   sh:hasValue :OurCompany ;
  ];
  sh:property [
   sh:path schema:gender ;
   sh:in   (schema:Male schema:Female)
  ] .
```

```
:alice a :User;
       schema:affiliation :OurCompany ;
       schema:gender schema:Female .

:bob   a :User;
       schema:affiliation :AnotherCompany ;
       schema:gender schema:Male .

:carol a :User;
       schema:affiliation :OurCompany ;
       schema:gender schema:Unknown .
```

☹

☹

Constraints on values with another shape

Constraint	Description
node	All values of a given property must have a given shape Recursion is not allowed in current SHACL

```
:User a sh:NodeShape, rdfs:Class ;
  sh:property [
   sh:path schema:worksFor ;
   sh:node :Company ;
  ] .

:Company a sh:Shape ;
  sh:property [
   sh:path     schema:name ;
   sh:datatype xsd:string ;
  ] .
```

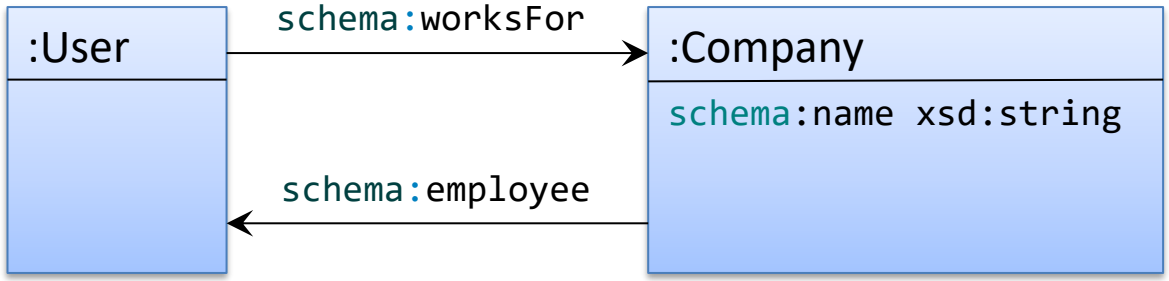
```
:alice a :User;
       schema:worksFor :OurCompany .

:bob   a :User;
       schema:worksFor :Another .

:OurCompany
    schema:name "OurCompany" .

:Another
    schema:name 23 .
```

Value shapes and recursion

Can we define cyclic data models as the following?

```
:User a sh:NodeShape ;
  sh:property [
   sh:path  schema:worksFor ;
   sh:node :Company ;
  ] .

:Company a sh:Shape ;
  sh:property [
   sh:path     schema:name ;
   sh:datatype xsd:string ;
  ] ;
 sh:property [
   sh:path schema:employee ;
   sh:node :User ;
 ]  .
```

Try it: https://tinyurl.com/y3hkka6s



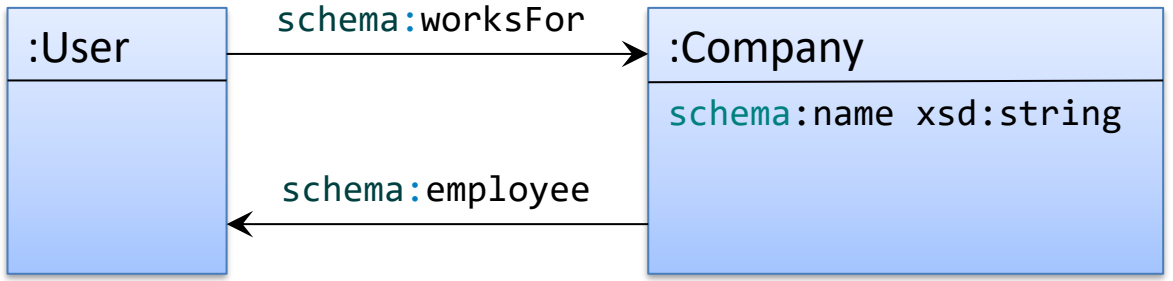

```
:alice schema:worksFor :OneCompany .
:bob   schema:worksFor :OneCompany .
:carol schema:worksFor :OneCompany .

:OneCompany schema:name "One" ;
  schema:employee :alice, :bob, :carol .
```

SHACL spec leaves this as implementation dependent

An approach to avoid recursion

Add rdf:type arcs for every resource and use `sh:class`

```
:User a sh:NodeShape ;
  sh:property [
   sh:path   schema:worksFor ;
   sh:class :Company ;
  ] .

:Company a sh:Shape ;
  sh:property [
   sh:path     schema:name ;
   sh:datatype xsd:string ;
  ] ;
 sh:property [
   sh:path   schema:employee ;
   sh:class :User ;
 ] .
```




```
:alice a :User ;
       schema:worksFor :OneCompany .
:bob   a :User ;
       schema:worksFor :OneCompany .
:carol a :User ;
       schema:worksFor :Something .

:OneCompany a :Company ;
       schema:name "One" ;
       schema:employee :alice, :bob, :carol .
```

☹

Try it: https://tinyurl.com/yynnts8o

Logical Operators

Constraint	Description
and	Conjunction of a list of shapes
or	Disjunction of a list of shapes
not	Negation of a shape
xone	Exactly one (similar XOR for 2 arguments)

Conjunction: and

Although it can be declared explicitly, it is the default behavior

```
:User a sh:NodeShape ;
  sh:and (
   [ sh:property [
      sh:path     schema:name;
      sh:minCount 1;
     ]
   ]
   [ sh:property [
      sh:path     schema:affiliation;
      sh:minCount 1;
     ]
   ]
  ) .
```

≡

```
:User a sh:Shape ;
   [ sh:property [
      sh:path     schema:name;
      sh:minCount 1;
     ]
   ]
   [ sh:property [
      sh:path     schema:affiliation;
      sh:minCount 1;
     ]
   ]
.
```

Disjunction: or

```
:User a sh:NodeShape ;
  sh:or (
   [ sh:property [
      sh:predicate foaf:name;
      sh:minCount 1;
     ]
   ]
   [ sh:property [
      sh:predicate schema:name;
      sh:minCount 1;
     ]
   ]
  ) .
```

```
:alice schema:name "Alice" .

:bob   foaf:name "Robert" .

:carol rdfs:label "Carol" .
```

Negation: not

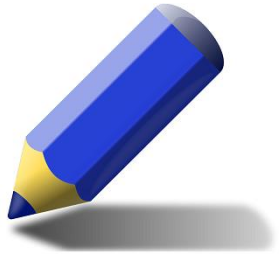
```
:NotFoaf a sh:NodeShape ;
 sh:not [ a sh:Shape ;
  sh:property [
   sh:predicate foaf:name ;
   sh:minCount 1 ;
  ] ;
 ] .
```

```
:alice schema:name "Alice" .

:bob   foaf:name "Robert" .

:carol rdfs:label "Carol" .
```

Exactly one: xone

```
:UserShape a sh:NodeShape ;
  sh:targetClass :User ;
  sh:xone (
   [ sh:property [
       sh:path      foaf:name;
       sh:minCount 1;
     ]
   ]
   [ sh:property [
       sh:path      schema:name;
       sh:minCount 1;
     ]
   ]
  ) .
```

```
:alice a :User ;                  #Passes as :User
       schema:name "Alice" .

:bob   a :User ;                  #Passes as :User
       foaf:name   "Robert" .

:carol a :User ;                  #Fails as :User
       foaf:name   "Carol";
       schema:name "Carol" .

:dave  a :User ;                  #Fails as :User
       rdfs:label  "Dave" .
```

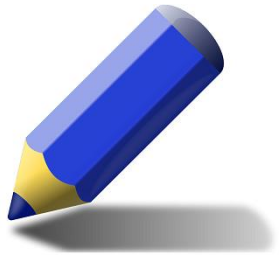


Exercise

IF-THEN pattern

All products must have `:productID` and, if a product has `rdf:type` `schema:Vehicle` then it must have the properties `schema:vehicleEngine` and `schema:fuelType`

```
:p1 a :Book;                    # Conforms
    schema:productID      "P1" .

:p2 a schema:Vehicle ;          # Conforms
    schema:productID      "P2" ;
    schema:fuelType       "Gasoline" ;
    schema:vehicleEngine "X2" .

:p3 a schema:Vehicle ;           # Fails
    schema:productID      "P3" .
```



Exercise

IF-THEN-ELSE pattern

All products must have `:productID` and, if a product has `rdf:type` `schema:Vehicle` then it must have the properties `schema:vehicleEngine` and `schema:fuelType` else it must have `schema:category` with a string value.

Value ranges

Constraint	Description
minInclusive	<=
maxInclusive	>=
minExclusive	<
maxExclusive	>

```
:Rating a sh:NodeShape ;
 sh:property [
   sh:path          schema:ratingValue ;
   sh:minInclusive 1 ;
   sh:maxInclusive 5 ;
   sh:datatype      xsd:integer
 ] .
```

```
:bad        schema:ratingValue 1 .

:average    schema:ratingValue 3 .

:veryGood   schema:ratingValue 5 .

:zero       schema:ratingValue 0 .
```

Try it: https://tinyurl.com/yy4bsavj

String based constraints

Constraint	Description
`minLength`	Restricts the minimum string length on value nodes
`maxLength`	Restricts the maximum string length on value nodes
`pattern`	Checks if the string value matches a regular expression

minLength/maxLength

Checks the string representation of the value

- This cannot be applied to blank nodes
- If minLength = 0, no restriction on string length

```
:User a sh:NodeShape ;
 sh:property [
   sh:path      schema:name ;
   sh:minLength 4 ;
   sh:maxLength 10 ;
 ] .
```

```
:alice schema:name "Alice" .

:bob schema:name "Bob" .                    ☹

:carol  schema:name :Carol .                ?

:strange schema:name _:strange .            ☹
```

Try it: https://tinyurl.com/y5laj3vf

pattern

Checks if the values matches a regular expression
It can be combined with sh:flags

```
:Product a sh:NodeShape ;
 sh:property [
   sh:path    schema:productID ;
   sh:pattern "^P\\d{3,4}" ;
   sh:flags   "i" ;
 ] .
```

```
:car   schema:productID "P2345" .

:bus   schema:productID "p567" .

:truck schema:productID "P12" .   ☹

:bike  schema:productID "B123" .  ☹
```

Language based constraints

Constraint	Description
`languageIn`	Declares the allowed languages of a literal
`uniqueLang`	Specifies that no pair of nodes can have the same language tag

languageIn

Specifies the allowed language that a literal can have

```
:ProductShape a sh:NodeShape;
  sh:targetClass :Product ;
  sh:property [
    sh:path       rdfs:label ;
    sh:languageIn ("es" "en" "fr")
] .
```

```
:p234 a :Product ;
  rdfs:label "jamón"@es, "ham"@en .

:p235 a :Product ;
  rdfs:label "milk"@en .

:p236 a :Product ;
  rdfs:label "Käse"@de .

:p237 a :Product ;
  rdfs:label "patatas"@es ,
             "kartofeln"@de .
```

☹

☹

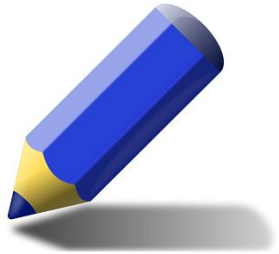
uniqueLang

Checks that no pair of nodes use the same language tag

```
:CountryShape a sh:NodeShape ;
  sh:targetClass :Country ;
  sh:property [
    sh:path       skos:prefLabel ;
    sh:uniqueLang true
] .
```

```
:spain  a :Country;
 skos:prefLabel "Spain"@en,
                "España"@es .

:france a :Country;
  skos:prefLabel "France",
                 "France"@en,
                 "Francia"@es .

:italy  a :Country .

:usa    a :Country;
        skos:prefLabel "USA"@en,
                       "United States"@en.
```

☹



Exercise

Nodes must have exactly one literal per language in English and Spanish for property `skos:prefLabel`

Property pair constraints

Constraint	Description
equals	The sets of values of both properties at a given focus node must be equal
disjoint	The sets of values of both properties at a given focus node must be different
lessThan	The values must be smaller than the values of another property
lessThanOrEquals	The values must be smaller or equal than the values of another property

```
:User a sh:NodeShape ;
 sh:property [
  sh:path   schema:givenName ;
  sh:equals foaf:firstName
 ];
 sh:property [
  sh:path      schema:givenName ;
  sh:disjoint  schema:lastName
] .
```

```
:alice schema:givenName "Alice";
       schema:lastName  "Cooper";
       foaf:firstName   "Alice" .

:bob   schema:givenName "Bob";
       schema:lastName  "Smith" ;
       foaf:firstName   "Robert" .

:carol schema:givenName "Carol";
       schema:lastName  "Carol" ;
       foaf:firstName   "Carol" .
```

☹

☹

Closed shapes

Constraint	Description
`closed`	Valid resources must only have values for properties that appear in `sh:property`
`ignoredProperties`	Optional list of properties that are also permitted

```
:User a sh:NodeShape ;
  sh:closed true ;
  sh:ignoredProperties ( rdf:type ) ;
  sh:property [
    sh:path schema:givenName ;
  ];
  sh:property [
    sh:path schema:lastName ;
  ] .
```

```
:alice schema:givenName "Alice";
       schema:lastName "Cooper" .

:bob   a :Employee ;
       schema:givenName "Bob";
       schema:lastName "Smith" .

:carol schema:givenName "Carol";
       schema:lastName "King" ;
       rdfs:label "Carol" .
```

☹

Qualified value shapes

Problem with repeated properties

Example: Books have two IDs (an isbn and an internal code)

```
:Book a sh:NodeShape ;
   sh:property [
    sh:path     schema:productID ;
    sh:minCount 1;
    sh:datatype xsd:string ;
    sh:pattern  "^isbn"
  ];
  sh:property [
    sh:path     schema:productID ;
    sh:minCount 1;
    sh:datatype xsd:string ;
    sh:pattern  "^code"
  ] .
```

```
:b1 schema:productID "isbn:123-456-789" ;
    schema:productID "code234" .
```

It fails!!

Qualified value shapes

Qualified value shapes verify that certain number of values of a given property have a given shape

```
:Book a sh:NodeShape;
 sh:property [
  sh:path schema:productID ;
  sh:minCount 2; sh:maxCount 2; ];
 sh:property [
  sh:path schema:productID ;
  sh:qualifiedMinCount 1 ;
  sh:qualifiedValueShape [
   sh:pattern "^isbn"
 ]];
 sh:property [
  sh:path schema:productID ;
  sh:qualifiedMinCount 1 ;
  sh:qualifiedValueShape [
   sh:pattern "^code" ;
  ]] .
```

```
:b1 schema:productID "isbn:123-456-789" ;
    schema:productID "code234" .
```

Non-validating constraints

Can be useful to annotate shapes or design UI forms

Constraint	Description
name	Provide human-readable labels for a property
description	Provide a description of a property
order	Relative order of the property
group	Group several constraints together

```
:User a sh:NodeShape ;
 sh:property [
  sh:path schema:url ;
  sh:name "URL";
  sh:description "User URL";
  sh:order 1
];
 sh:property [
  sh:path schema:name ;
  sh:name "Name";
  sh:description "User name";
  sh:order 2
] .
```

Non-validating constraints

```
:User a sh:NodeShape ;
 sh:property [ sh:path schema:url ;
  sh:name "URL";
  sh:group :userDetails
];
 sh:property [ sh:path schema:name ;
  sh:name "Name"; sh:group :userDetails
];
sh:property [ sh:path schema:address ;
  sh:name "Address"; sh:group :location
];
sh:property [ sh:path schema:country ;
  sh:name "Country"; sh:group :location
] .
```

```
:userDetails a sh:PropertyGroup ;
  sh:order 0 ;
  rdfs:label "User details" .

:location a sh:PropertyGroup ;
  sh:order 1 ;
  rdfs:label "Location" .
```

An agent could generate a form like:

User details

URL: _____

Name: _____

Location

Address: _____

Country: _____

SHACL-SPARQL

SPARQL constraints

Constraints based on SPARQL code.

When the SPARQL query return validation errors a violation is reported

SPARQL constraints have type sh:SPARQLConstraint

Constraint	Description
`message`	Message in case of error
`sparql`	SPARQL code that is run
`prefixes`	Points to namespace prefix declarations defined by sh:declare: Each one has: sh:prefix: Prefix alias sh:namespace: namespace IRI

SPARQL constraints

Special variables are pre-binded by the SHACL-SPARQL processor

Constraint	Description
`$this`	Focus Node
`$shapesGraph`	Can be used to query the shapes graph in named graphs Similar to: `GRAPH $shapesGraph { ... }`
`$currentShape`	Current shape

SPARQL constraints

Mappings between result rows and error validation information

Constraint	Description
`sh:focusNode`	Value of `$this` variable
`sh:subject`	Value of `?subject` variable
`sh:predicate`	Value of `?predicate` variable
`sh:object`	Value of `?object` variable
`sh:message`	Value of `?message` variable
`sh:sourceConstraint`	The constraint that was validated against
`sh:sourceShape`	The shape that was validated against
`sh:severity`	sh:ViolationError by default or the value of sh:severity

SPARQL constraints

Example: Name must be the concatenation of singleName and familyName

```
:UserShape a sh:NodeShape ;
 sh:targetClass :User ;
 sh:sparql [ a sh:SPARQLConstraint ;
  sh:message "schema:name must equal schema:givenName+schema:familyName";
  sh:prefixes [ sh:declare [
    sh:prefix "schema" ;
    sh:namespace "http://schema.org/"^^xsd:anyURI ;
   ]] ;
 sh:select
  """SELECT $this (schema:name AS ?path) (?name as ?value)
     WHERE {
      $this schema:name ?name .
      $this schema:givenName ?givenName .
      $this schema:familyName ?familyName .
      FILTER (!isLiteral(?value) ||
              !isLiteral(?givenName) || !isLiteral(?familyName) ||
              concat(str(?givenName), ' ', str(?familyName))!=?name )
     }""" ;
] .
```

```
:alice a :User ;
  schema:givenName "Alice" ;
  schema:familyName "Cooper" ;
  schema:name "Alice Cooper" .

:bob a :User ;
  schema:givenName "Bob" ;
  schema:familyName "Smith" ;
  schema:name "Robert Smith" .
```

☹

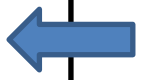
https://tinyurl.com/y6p7zyon

SPARQL constraint components

SHACL-SPARQL allows to declare custom constraint components

Once defined, they can be used like bult-in constraint components

```
:ProductShape a sh:NodeShape ;
 sh:targetClass :Product ;
 sh:property [
  sh:path     :color ;
  :size       3 ;
  sh:minCount 1 ;
 ] .
```



```
:c1 :color (255 0 255) .

:c2 :color (255 0 210 345) .

:c3 :color (255 0) .
```

☹

☹

SPARQL constraint components

```
:FixedListConstraintComponent
 a sh:ConstraintComponent ;
 sh:parameter [
  sh:path        :size ;
  sh:name        "Size of list" ;
  sh:description "The size of the list" ;
 ] ;
 sh:labelTemplate "Size of values: \"{$size}\"" ;
 sh:propertyValidator :fixedLengthValidator .
```

Two types of validators:
SPARQLSelectValidator
SPARQLASKValidator

```
:fixedLengthValidator  a sh:SPARQLSelectValidator ;
  sh:message
   "{$PATH} must have length {?size}, not {?count}" ;
  sh:prefixes [ sh:declare [
    sh:prefix "rdf" ;
    sh:namespace
     "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   ]
  ] ;
  sh:select """SELECT $this ?value $count WHERE {
     $this $PATH ?value .
     { SELECT $this ?value
              (COUNT(?member) AS ?count)
              $size WHERE {
          ?value rdf:rest*/rdf:first ?member
      } GROUP BY $this ?value $size
     }
     FILTER (!isBlank(?value) || ?count != $size)
  }"""
.
```

https://tinyurl.com/y3l5ksah

SPARQL constraint components

Property	Description
`sh:parameter`	Declares the parameters of the constraint component The values are subclasses of property shapes sh:path declares the parameter name sh:optional declares if the parameter is optional
`sh:labelTemplate`	Suggests how constraints are rendered. Can refer to parameter names using: `$varName`
`sh:nodeValidator`	Associates a node shape validator
`sh:propertyValidator`	Associates a property shape validator

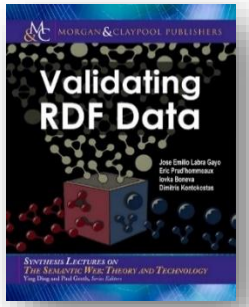
SPARQL based validators can be SELECT or ASK based validators

ShEx and SHACL: Several common features…

	ShEx	SHACL
Both employ the word "shape"	✓	✓
Can be used to validate RDF graphs	✓	✓
Node constraints	✓	✓
Constraints on incoming/outgoing arcs	✓	✓
Cardinalities on properties	✓	✓
RDF syntax	✓	✓
Extension mechanism	✓	✓

ShEx and SHACL compared

More information in chapter 7 of Validating RDF data book



2017 HTML version:
http://book.validatingrdf.com

Or in some talks like:
https://labra.weso.es/talk/2018_validatingrdfdata_stanford/

ShEx and SHACL
description - validation - constraints

Description
Tell what something is about
What data we want/expect

ShEx

Validation
Check that data conforms with expectations

SHACL

Constraint
Rule that something has to obey
What data we don't want/expect

DublinCore

WESO

DCTAP

Metadata model

Tabular: CSV, Spreadsheets

Easy to understand and edit by domain experts

It can be used as a template and a *single-source-of-truth*

- DCTAP can be converted to Shapes (ShEx/SHACL/etc.)

DCTAP example

shapeID	propertyID	propertyLabel	mandatory	repeatable	valueDataType	valueShape
Researcher	rdfs:label	Label	TRUE	FALSE	xsd:string	
	:birthPlace	Place of birth	FALSE	FALSE		Place
	:birthDate	Date of birth	FALSE	FALSE	xsd:date	
	:employer	Employer	FALSE	TRUE		Organization
Place						
Organization						

```
<Researcher> {
 :name        xsd:string          ;
 :birthPlace @<Place>           ? ;
 :birthDate   xsd:date          ? ;
 :employer   @<Organization> * ;
}
```

In ShEx

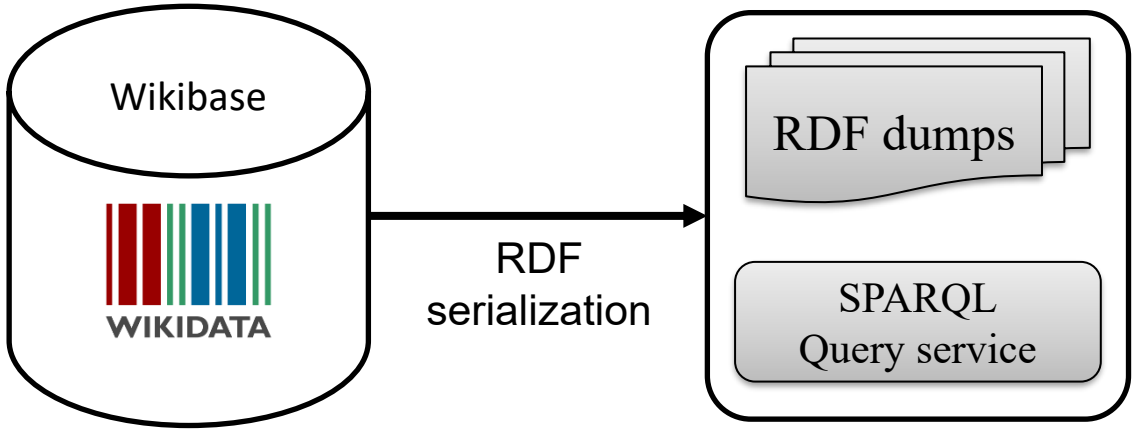
Contents

Wikibase and RDF

Wikibase graphs are also available through SPARQL endpoint

Internally, Wikibase has 2 DBs:

- Relational database (MariaDB)
- RDF Triplestore (Blazegraph)
  - Plans to update



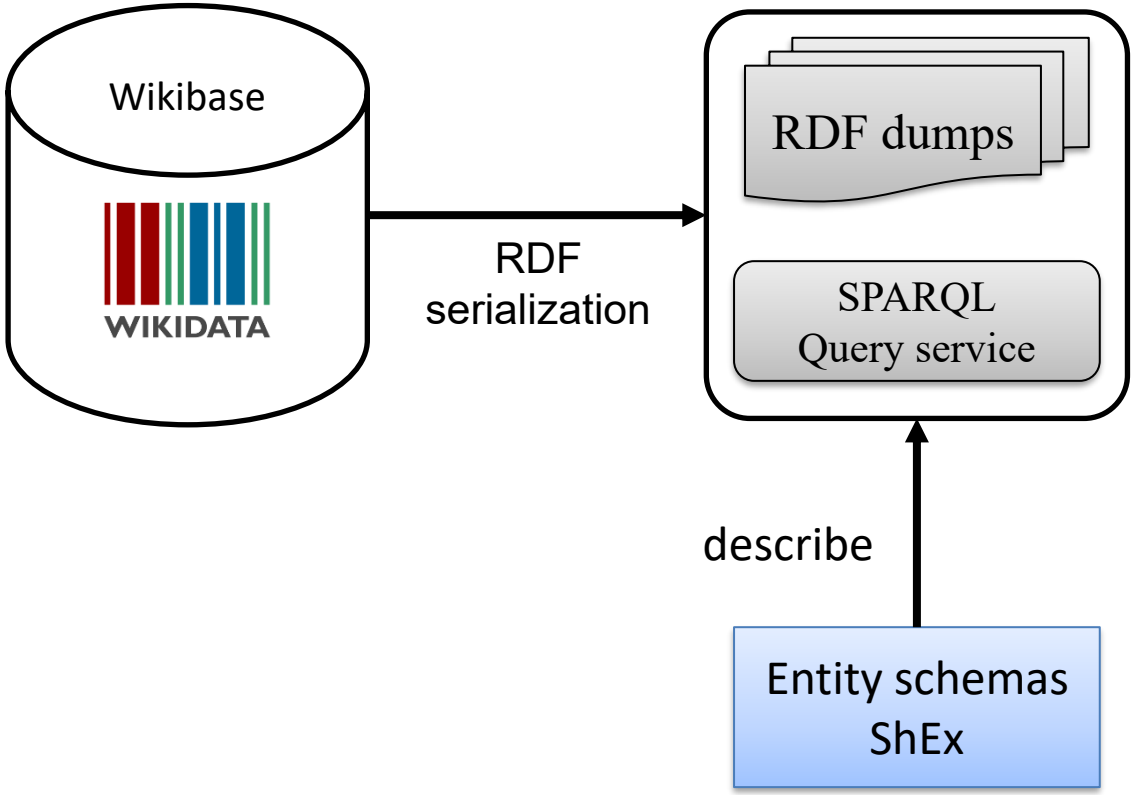

```
select ?name ?date?country where {
 wd:Q80 wdt:P1559 ?name .
 wd:Q80 wdt:P569  ?date .
 wd:Q80 wdt:P19   ?place .
 ?place wdt:P17   ?country
}
```

?name	?date	?country
Tim Berners-lee	1955-06-08	:UK

Try it: https://w.wiki/5yGu

Wikibase and RDF: Entity Schemas

If we have RDF, we can use ShEx to describe and validate entities



Entity Schemas

They can be used to describe Wikidata entities

Adopted in 2019 as a new Wikidata namespace Exxx

Example:

https://www.wikidata.org/wiki/EntitySchema:E10

Directory of entity schemas:

https://www.wikidata.org/wiki/Wikidata:Database_reports/EntitySchema_directory

Example of an Entity Schema

Q80.ttl

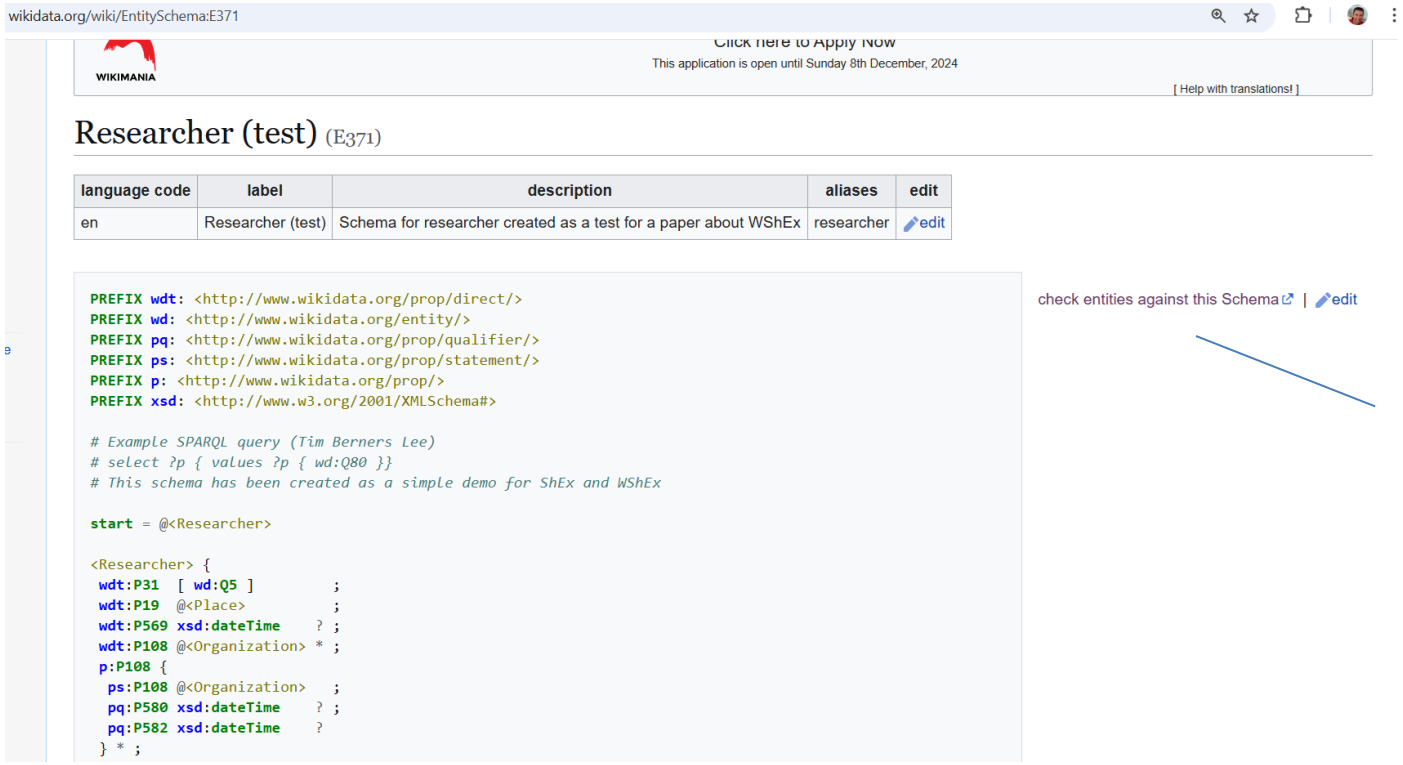
```
wd:Q80 rdf:type wikibase:Item ;
  wdt:P31 wd:Q5 ;
  wdt:P19 wd:Q84 ;
  wdt:P569 "1955-06-08T00:00:00Z"^^xsd:dateTime ;
  wdt:P108 wd:Q42944 ;
  p:P108 s:Q80-fcba864c, :Q80-4fe7940f
  #...
  .

:Q80-4fe7940f rdf:type wikibase:Statement ;
  wikibase:rank wikibase:NormalRank ;
  ps:P108 wd:Q42944 ;
  pq:P580 "1984-01-01T00:00:00Z"^^xsd:dateTime ;
  pq:P582 "1994-01-01T00:00:00Z"^^xsd:dateTime .

s:Q80-fcba864c a wikibase:Statement ;
    wikibase:rank wikibase:NormalRank ;
    ps:P108 wd:Q42944 ;
    pq:P580 "1980-06-01T00:00:00Z"^^xsd:dateTime ;
    pq:P582 "1980-12-01T00:00:00Z"^^xsd:dateTime .
```

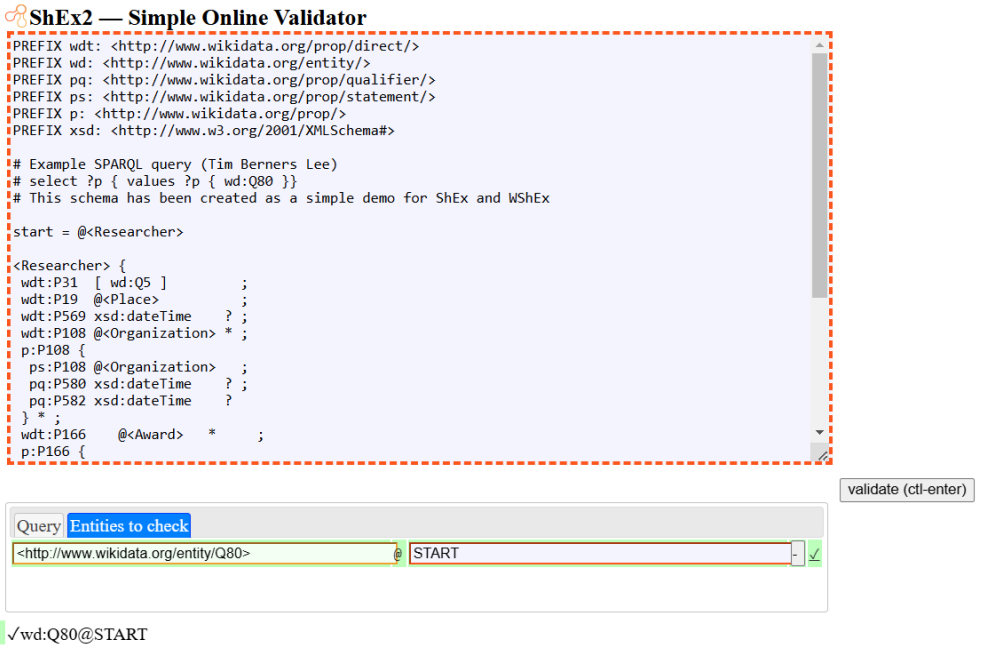
Entity-schema - ShEx

```
PREFIX pq: <.../prop/qualifier/>
PREFIX ps: <.../prop/statement/>
PREFIX p: <.../prop/>
PREFIX wdt: <.../prop/direct/>
PREFIX wd: <.../entity/>
PREFIX xsd: <...XMLSchema#>

<Researcher> {
 wdt:P31  [ wd:Q5 ]          ;
 wdt:P19  @<Place>           ;
 wdt:P569 xsd:dateTime       ;
 wdt:P108 @<Employer>      * ;
 p:P108 { ps:P108 @<Employer>     ;
          pq:P580 xsd:dateTime  ? ;
          pq:P582 xsd:dateTime  *
 } *
}
<Place> { }
<Employer> { }
```

Using Entity Schemas for validation

Example: https://www.wikidata.org/wiki/EntitySchema:E371

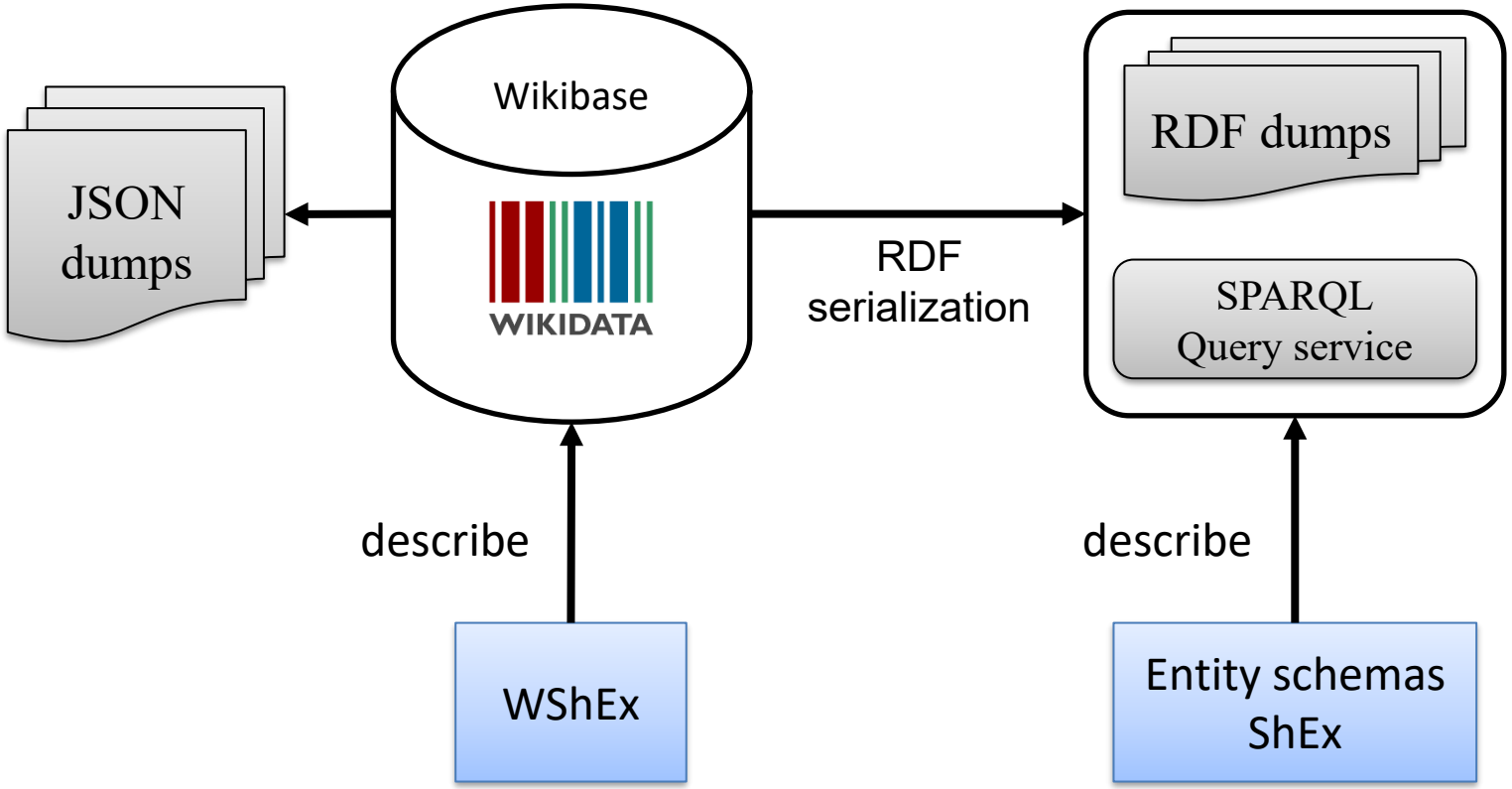




WShEx

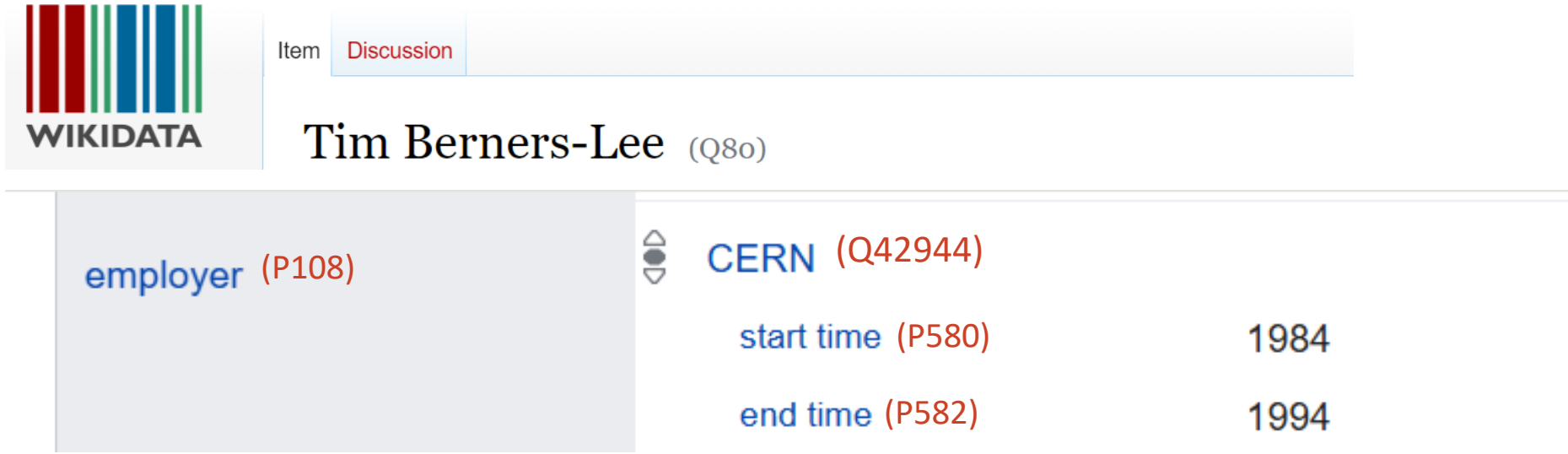
Although Entity Schemas and ShEx just work, they are indirectly describing Wikibase

WShEx has been proposed as a language to describe the Wikibase data model?



https://www.weso.es/WShEx/

Wikibase RDF representation



RDF

```
wd:Q80 wdt:P108 wd:Q42944 ;
 p:P108 s:Q80-fcba864c
 ...

s:Q80-fcba864c a wikibase:Statement ;
    ps:P108 wd:Q42944 ;
    pq:P580 1984 ;
    pq:P582 1994 .
```

Could be represented as

```
:Q80 :P108 :Q42944 {|
  :P580 1984 ;
  :P582 1994
 |}
```

Inspired by RDF-1.2 annotated triples syntax

ShEx vs WShEx

Entity-schema - ShEx

```
PREFIX pq: <.../prop/qualifier/>
PREFIX ps: <.../prop/statement/>
PREFIX p: <.../prop/>
PREFIX wdt: <.../prop/direct/>
PREFIX wd: <.../entity/>
PREFIX xsd: <...XMLSchema#>

<Researcher> {
 wdt:P31   [ wd:Q5 ]          ;
 wdt:P19   @<Place>           ;
 wdt:P569  xsd:dateTime       ;
 wdt:P108  @<Employer>      * ;
 p:P108 { ps:P108 @<Employer>     ;
          pq:P580 xsd:dateTime  ? ;
          pq:P582 xsd:dateTime  ?
 } *
}
<Place> { }
<Employer> { }
```

WShEx

```
PREFIX : <.../entity/>

<Researcher> {
 :P31   [ :Q5 ]    ;
 :P108  @<Employer> {| :P580 Time ? ,
                       :P582 Time ?
                     |} *
}
<Award> { :P17 @<Country> }
<Country> {}
```

https://arxiv.org/abs/2208.02697

Contents

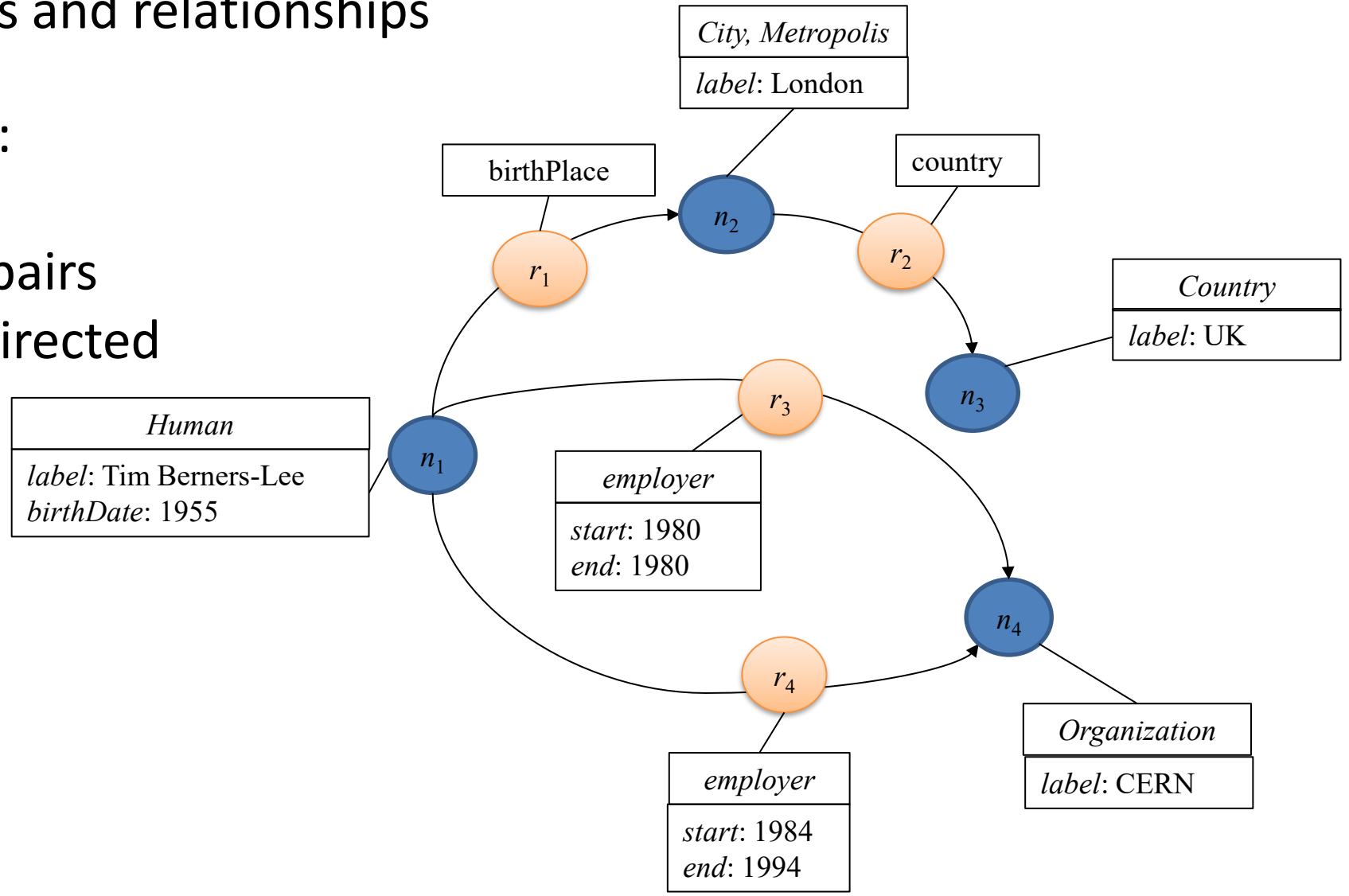
Property graphs

Graph structure with nodes and relationships (edges)

Nodes and edges can have:

- Labels
- A set of property-value pairs

Edges can be directed/undirected



Shaping Property graphs with GQL

GQL defines the concept of Graph Types

- It describes the graph in terms of restrictions on labels, properties, nodes, edges and topology
- Graph types constrain the set of nodes that can be contained in a graph

Multiple graphs can refer to the same graph type

Graph types can be created independently:

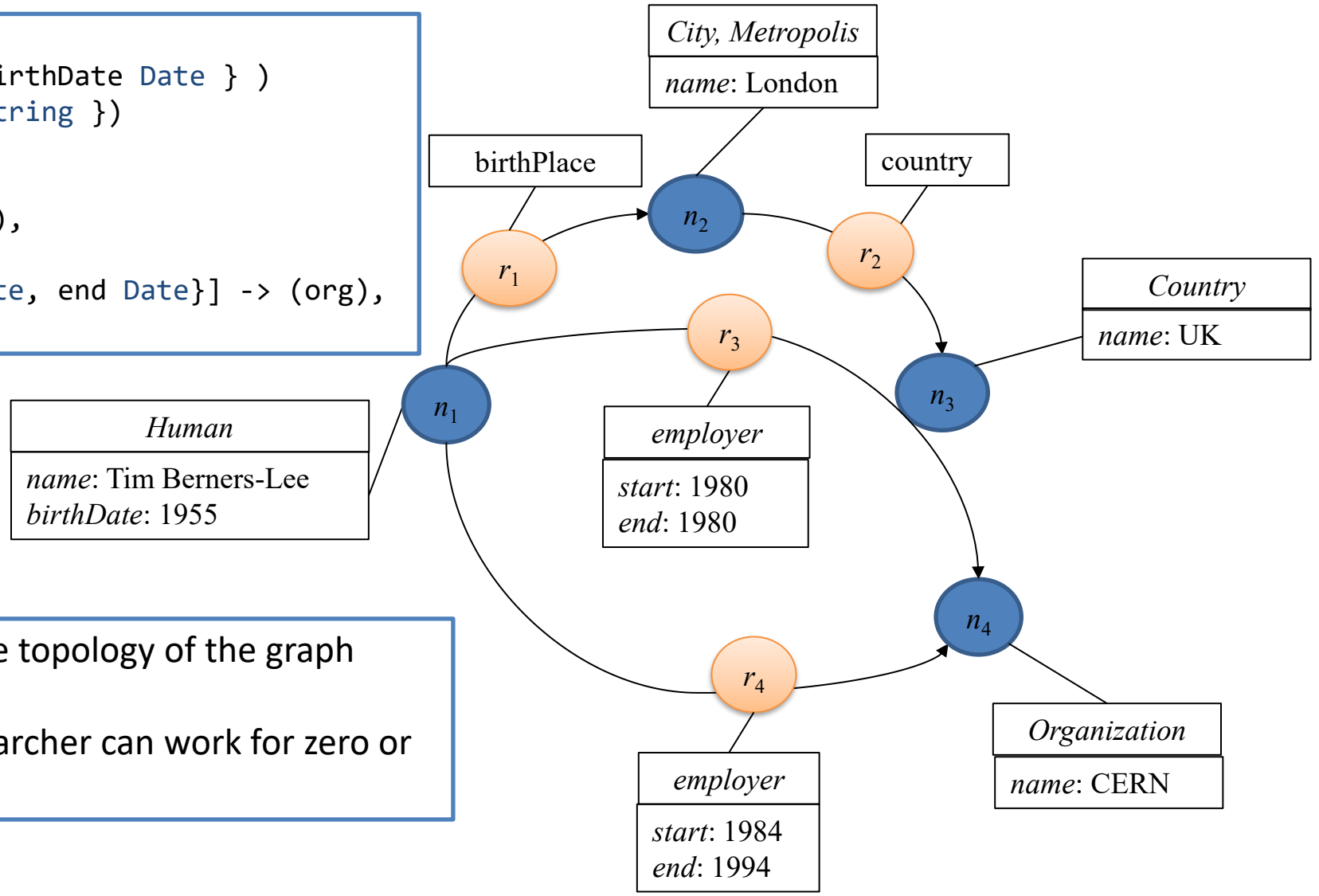
```
CREATE GRAPH TYPE name { graph type spec }
```

Or when creating the graph:

```
CREATE GRAPH …content… TYPED { graph type spec }
```

Shaping property graphs with GQL: Example

```
CREATE GRAPH TYPE example {
  (researcher :Human => { name String, birthDate Date } )
  (place :City & :Metropolis => { name String })
  (country :Country => { name String })
  (org :Organization => { name String })
  (researcher) -[ :birthPlace ]-> (place),
  (place) -[ :country ]-> (country),
  (researcher) -[:employer => { start Date, end Date}] -> (org),
}
```



No Cardinality constraints about the topology of the graph
Example:

- How can we indicate that a researcher can work for zero or more employers?

PShEx

Proposal to extend ShEx to handle property graphs

Similar to WShEx, but adapted for Property graphs

We add a new description level for property-value pairs (in nodes and edges)

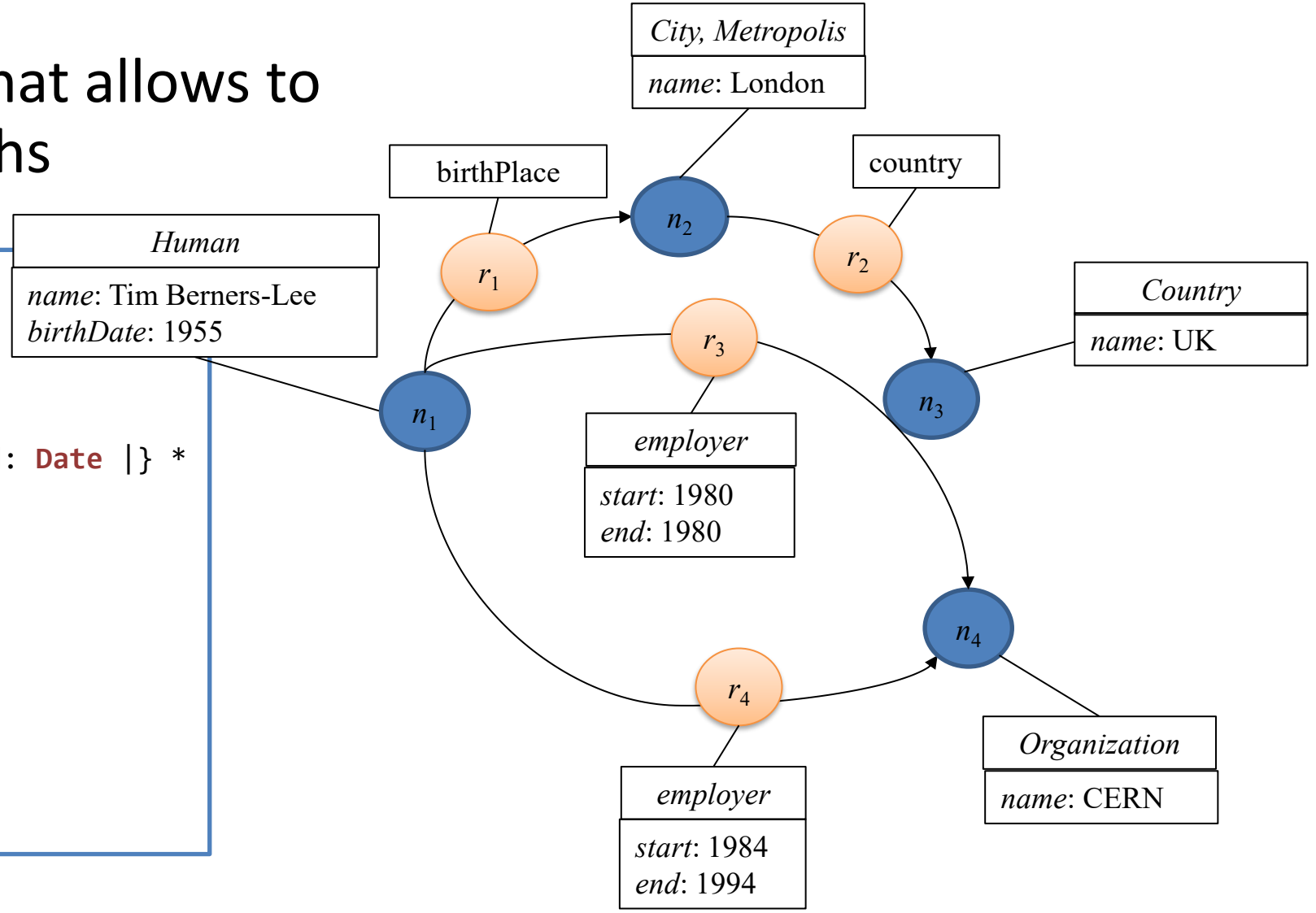
It allows to declare cardinality constraints on the topology of the graph

Extending Shape Expressions for different types of knowledge graphs , Jose Emilio Labra Gayo, In 1st Workshop on *Data Quality meets Machine Learning and Knowledge Graphs*, DQMLKG, part of *Extended Semantic Web Conference 2024*, ESWC24. - 2024, http://labra.weso.es/publication/2024_shex_extensions/

Shaping property graphs: PShEx

PShEx = ShEx extension that allows to describe property graphs

```
<Researcher> [( Human )] [|
  name:       String     ;
  birthDate:  Date     ?
|] AND {
  birthPlace: @<Place> ? ;
  employer:   @<Org> {| start: Date; end: Date |} *
}
<Place> [( City)]{
  country: @<Country>
}
<Country> {
  name: String
}
<Org> {
  name: String
}
```



PGSchema PC

Recent proposal: PGSchema with property constraints

Accepted at K-Cap conference 2025 (https://www.k-cap.org/)

More information: https://www.weso.es/pgschemapc

The functionality will be integrated in rudof

Contents

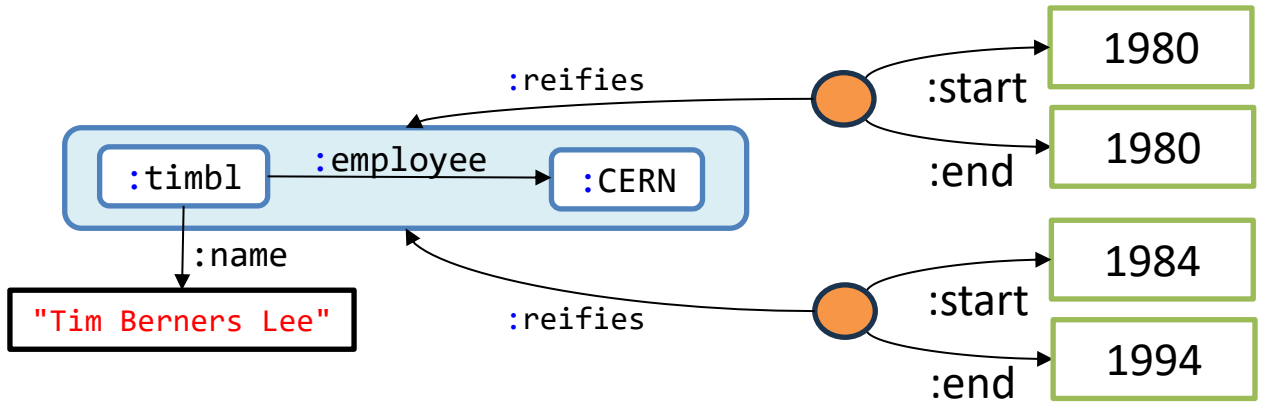
RDF 1.2

Current working draft: https://www.w3.org/TR/rdf12-concepts/

Main novelty: Add statements about triples: reifiers

```
:timbl :name  "Tim Berners Lee“ ;
       :employer :CERN .
_:r1 rdf:reifies << :timbl :employer :CERN >> .
_:r1 :start 1980 ;
     :end   1980 .
```

```
_:r2 rdf:reifies << :timbl :employer :CERN >> .
_:r2 :start 1984 ;
     :end   1994 .
```



Alternative syntax

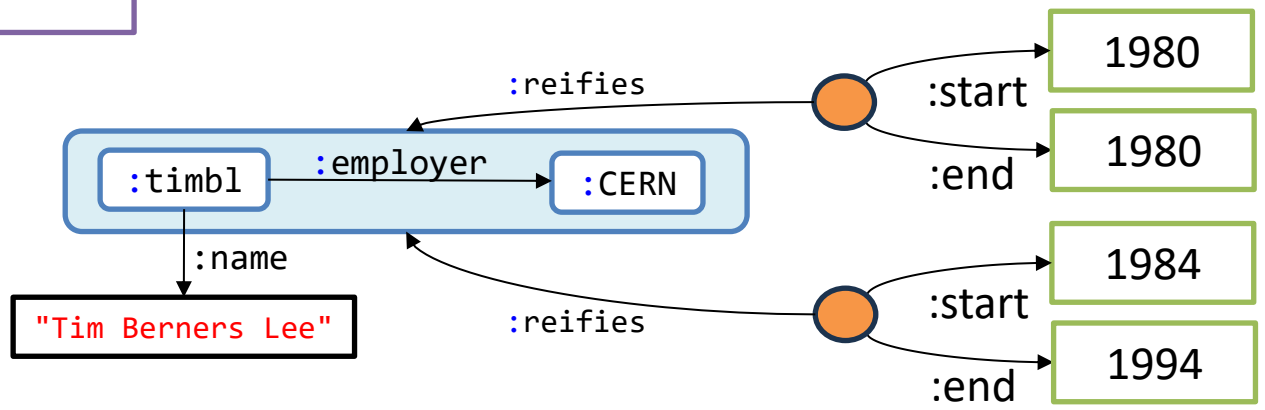
```
:timbl :name  "Tim Berners Lee";
       :employer :CERN {|
          :start 1980; :end 1980
        |} {|
          :start 1984; :end 1994
        |} .
```

ShEx for RDF 1.2 (ShEx Star)

Note: work in progress in the roadmap for rudof

```
<Researcher> {
 :name   xsd:string ;
 :employer @<Org> {|
         :start xsd:date,
         :start xsd:date
 |} *
}
<Org> {}
```

```
:timbl :name  "Tim Berners Lee";
       :employer :CERN {|
          :start 1980;
          :end   1980
       |} {|
          :start 1984;
          :end   1994
       |} .
```



SHACL 1.2

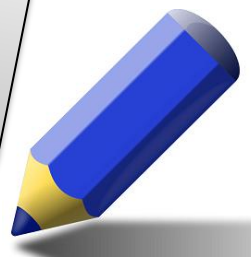
Currently under development at W3C: https://www.w3.org/TR/shacl12-core/

```
:UserShape a sh:NodeShape ;
   sh:targetClass :User ;
   sh:property [
     sh:path :employer ;
     sh:reificationRequired true ;
     sh:reifierShape [
       sh:property [
         sh:path :start ;
         sh:datatype xsd:integer ;
         sh:minCount 1 ; sh:maxCount 1
       ] ;
       sh:property [
         sh:path :end ;
         sh:datatype xsd:integer ;
         sh:minCount 1 ; sh:maxCount 1
       ]]] .
```

```
:timbl :name  "Tim Berners Lee";
       :employer :CERN {|
          :start 1980;
          :end   1980
        |} {|
          :start 1984;
          :end   1994
        |} .
```

New

The example is already supported in rudof



Convergence on graph schema languages

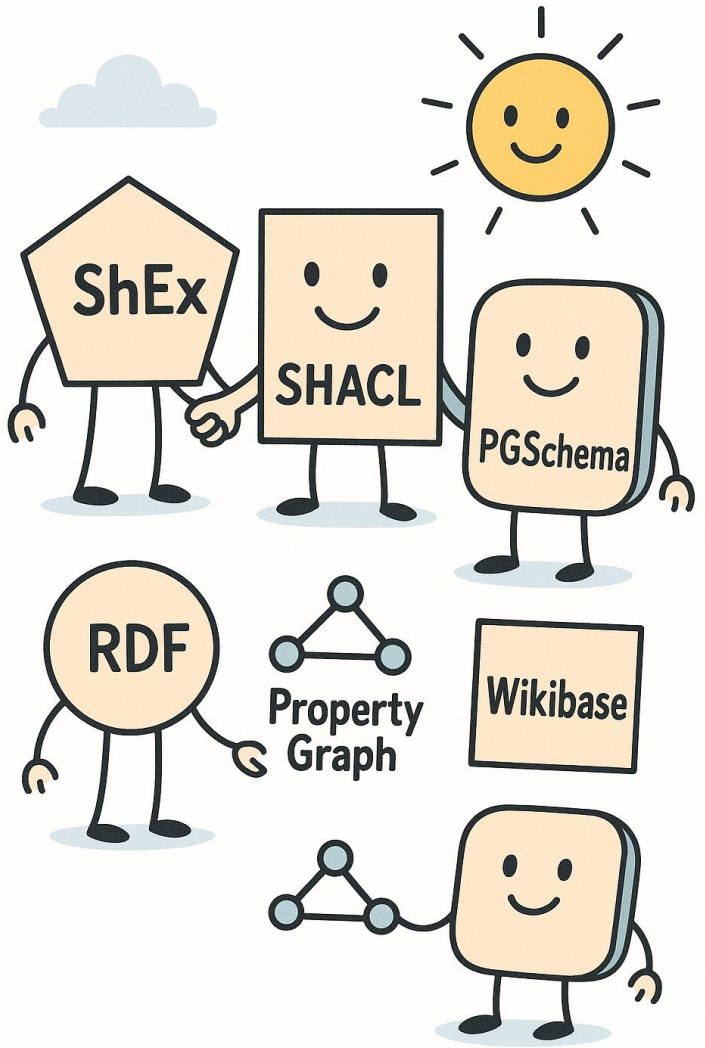
More theoretical work is currently under development:

Common Foundations for SHACL, ShEx, and PG-Schema, Shqiponja Ahmetaj, Iovka Boneva, Jan Hidders, Katja Hose, Jose Emilio Labra Gayo, Wim Martens, Fabio Mogavero, Filip Murlak, Cem Okulmus, Axel Polleres, Ognjen Savković, Mantas Šimkus, Dominik Tomaszuk, International World Wide Web Conference, Sidney, Australia, 2025 – 2025

https://labra.weso.es/publication/2025_common_foundations_shacl_shex_pgschema/

Conclusions

- Several types of Knowledge Graphs
  - Different models
  - Could they converge in the future?
- Shapes increase the quality of Knowledge graphs
  - Can be extended for other types of Knowledge Graphs
    - Wikibase graphs: ShEx, WShEx
    - Property graphs: PShEx, PGSchema, PGSchemaPC, …
    - RDF 1.2: ShEx (next), SHACL 1.2
  - Could the shapes languages converge in the future?



Generated by ChatGPT