Shaping different types of Knowledge Graphs

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

About me…

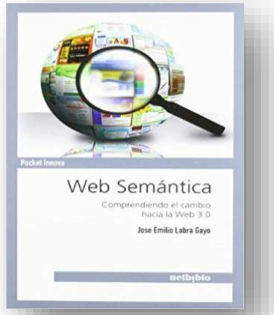
Main researcher at WESO (Web Semantics Oviedo)

Some books:

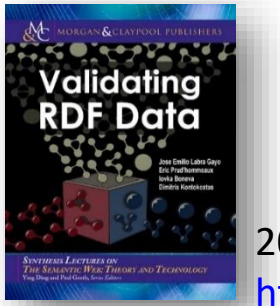
- "*Web semántica*" (in Spanish), 2012
- "*Validating RDF data*", 2017
- "Knowledge Graphs", 2021

…and some software:

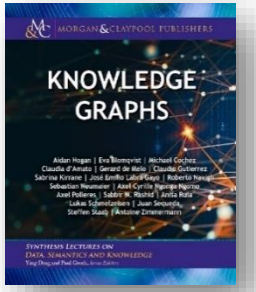
- RDFShape (RDF playground)
- rudof



2012



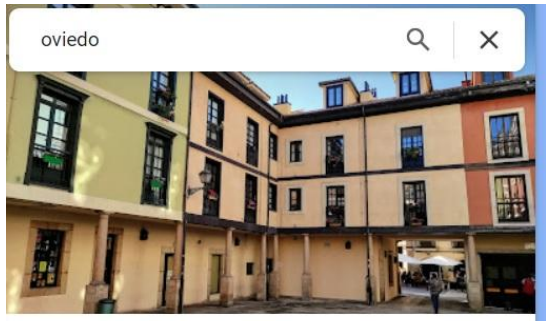
2017 HTML version:
http://book.validatingrdf.com

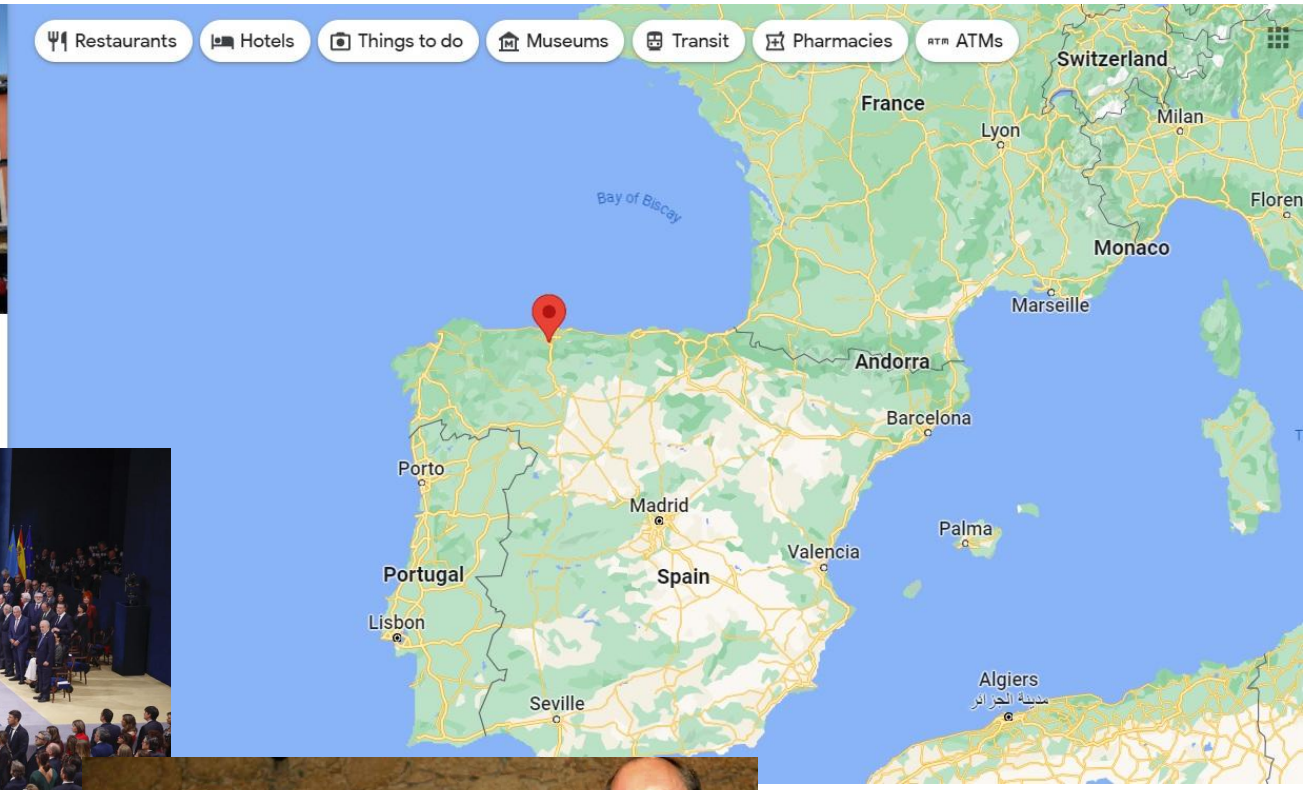


2021, HTML version
https://kgbook.org/

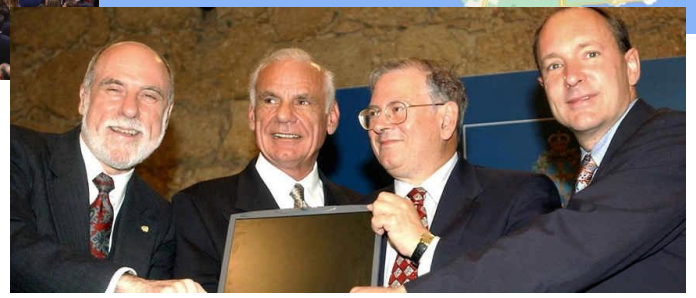
http://labra.weso.es

About Oviedo, Asturias





Princess of Asturias Awards



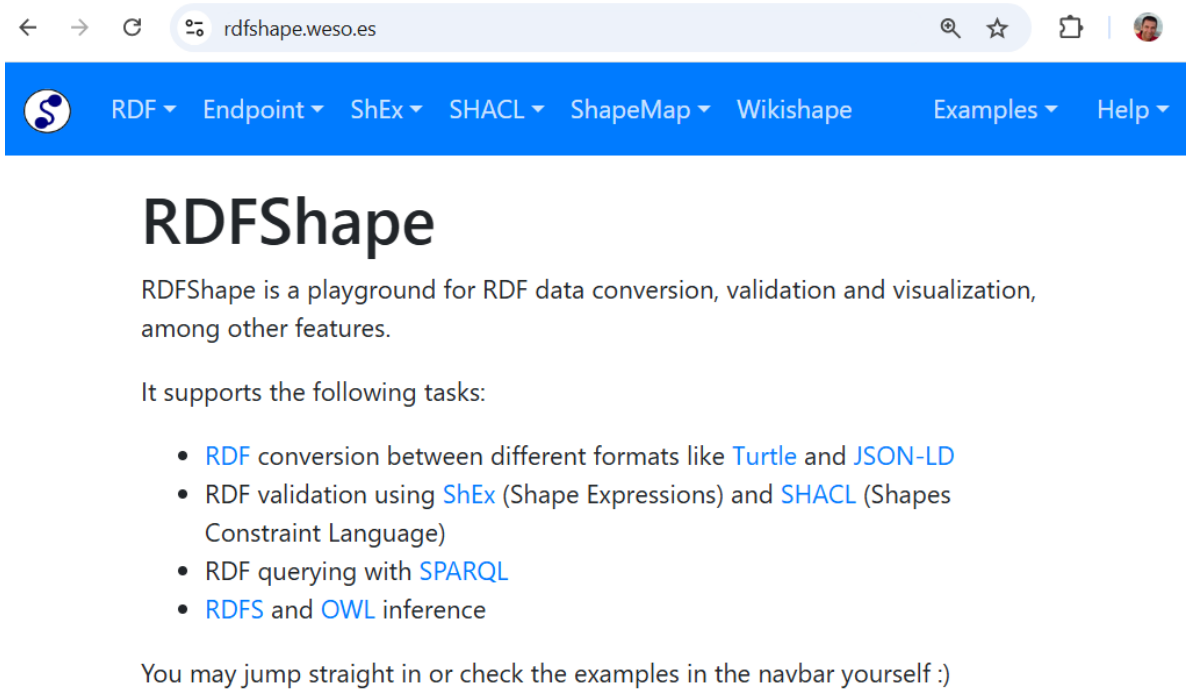
2002, Tim Berners-Lee in Asturias receiving the Prince of Asturias Award
Together with: Vinton Cerf, Lawrance Roberts and Robert Kahn

More slides: https://www.validatingrdf.com/tutorial/iswc2025/

If you want to play…

Online web playground

RDFShape: RDF playground



Or using rudof

RUDOF

What is rudof?

Library that supports Knowledge graphs

- Different data models: RDF & RDF 1.2, Wikibase, Property graphs

Shapes or schemas for Knowledge Graphs

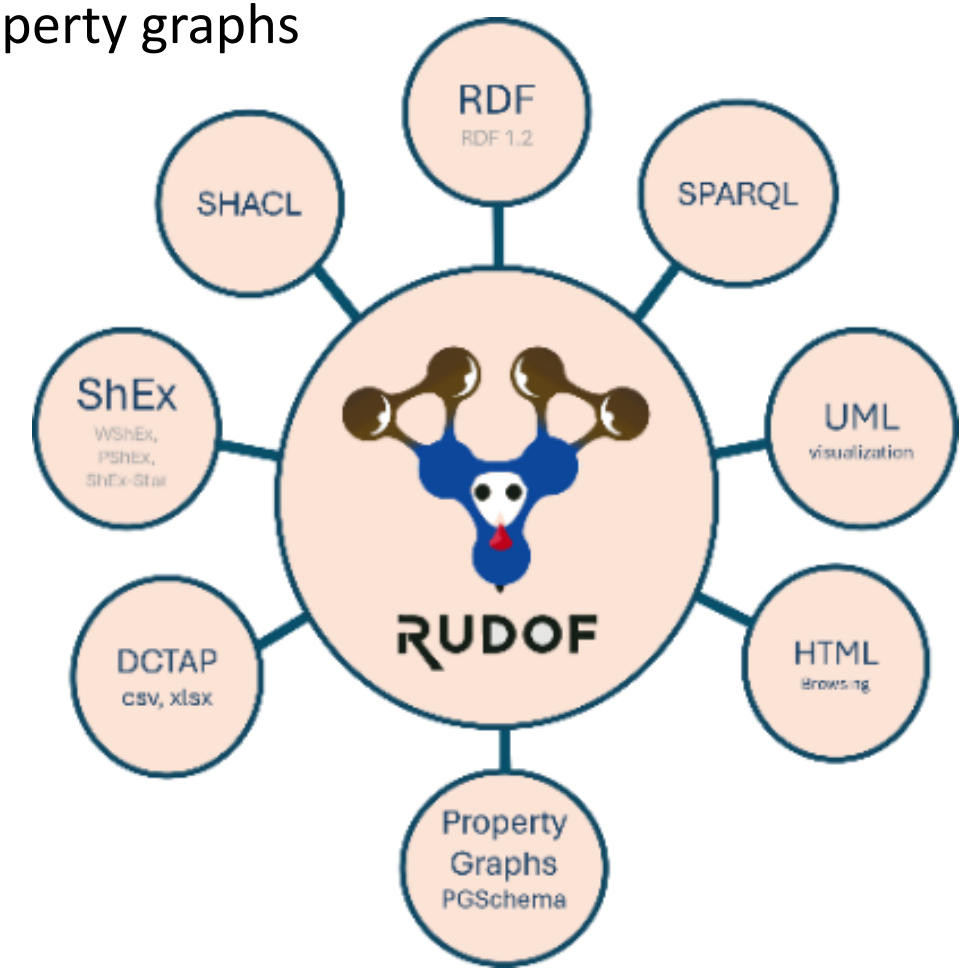
- ShEx, SHACL, DCTAP, PGSchema, …

Other tools:

- Querying (SPARQL)
- Visualizations (UML, HTML, SVG,...)
- Generation
- …

Available as:

- Command line and Interactive Shell NEW
  - Windows, Linux, MacOS
- Python Bindings: Jupyter Notebooks
- MCP server

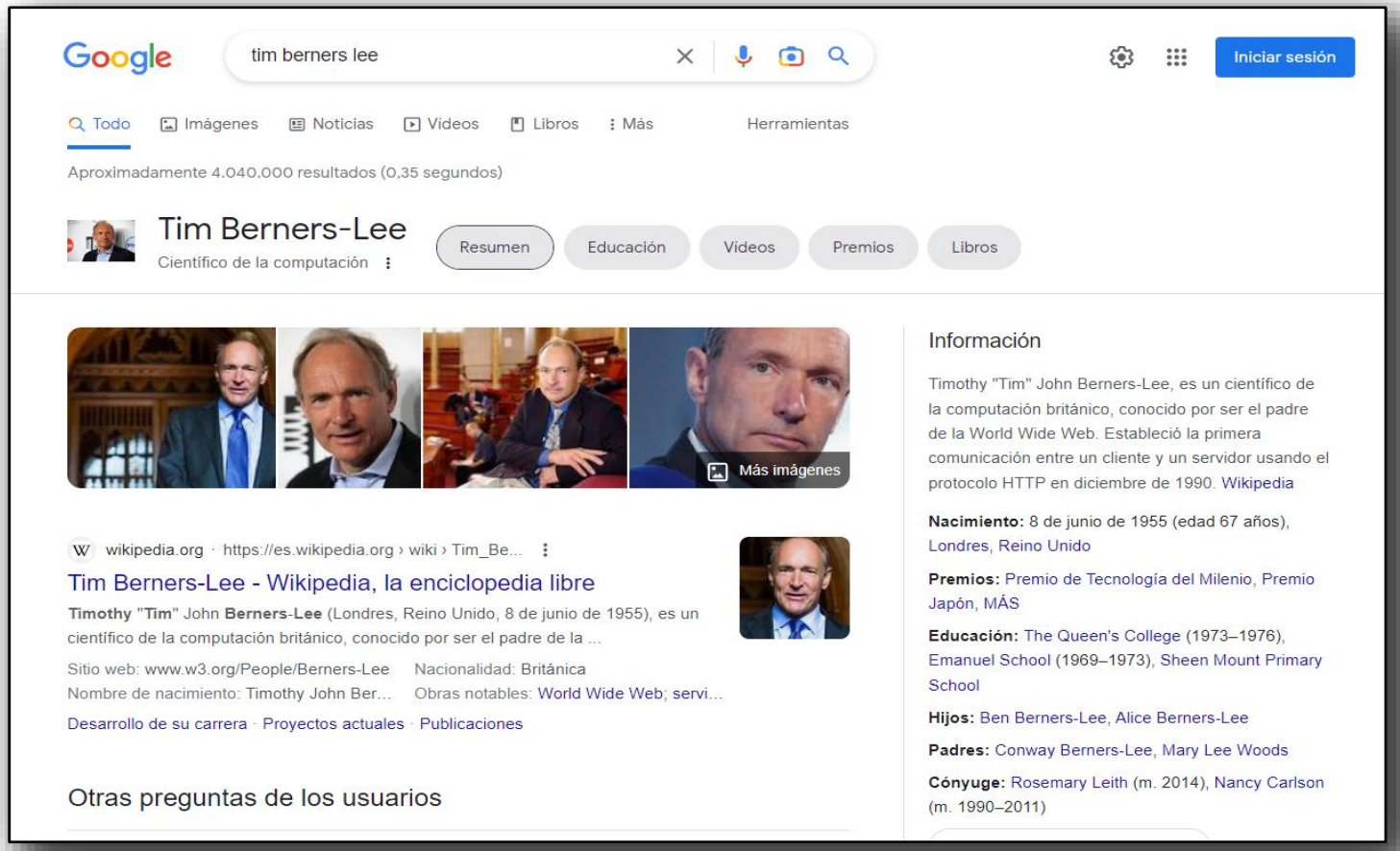


Contents

- Knowledge graphs & Types of Knowledge Graphs:
  - RDF, Property graphs, Wikibase, RDF-1.2
- Shapes for RDF: ShEx & SHACL, DCTAP
- Shapes for other types of Knowledge graphs:
  - Shaping Wikibase and Wikidata graphs
  - Shaping Property Graphs
  - Shaping RDF 1.2
- Some ideas for future work

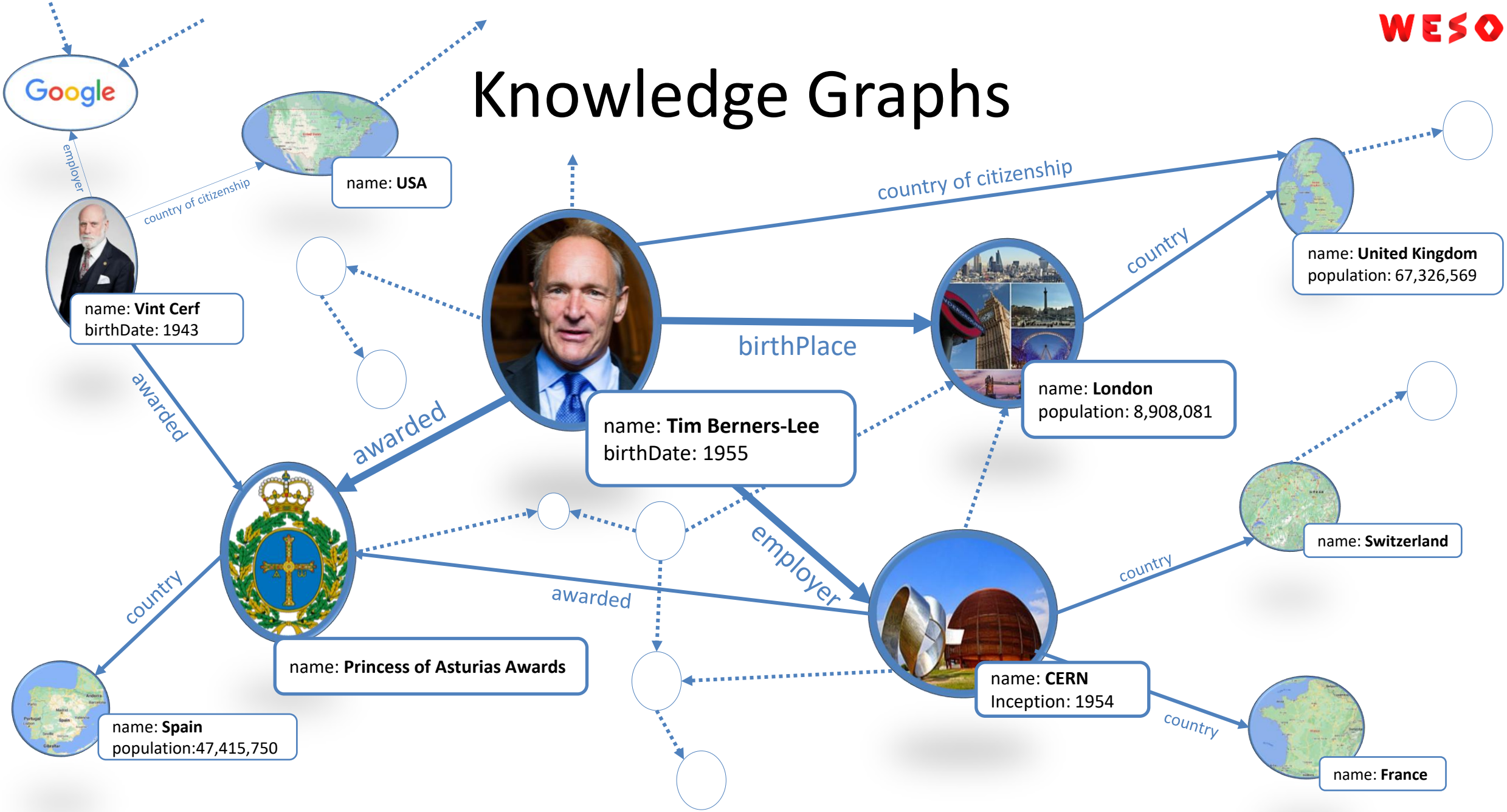
Knowledge Graphs

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



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

Knowledge Graphs



Types of Knowledge Graphs

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

RDF graphs

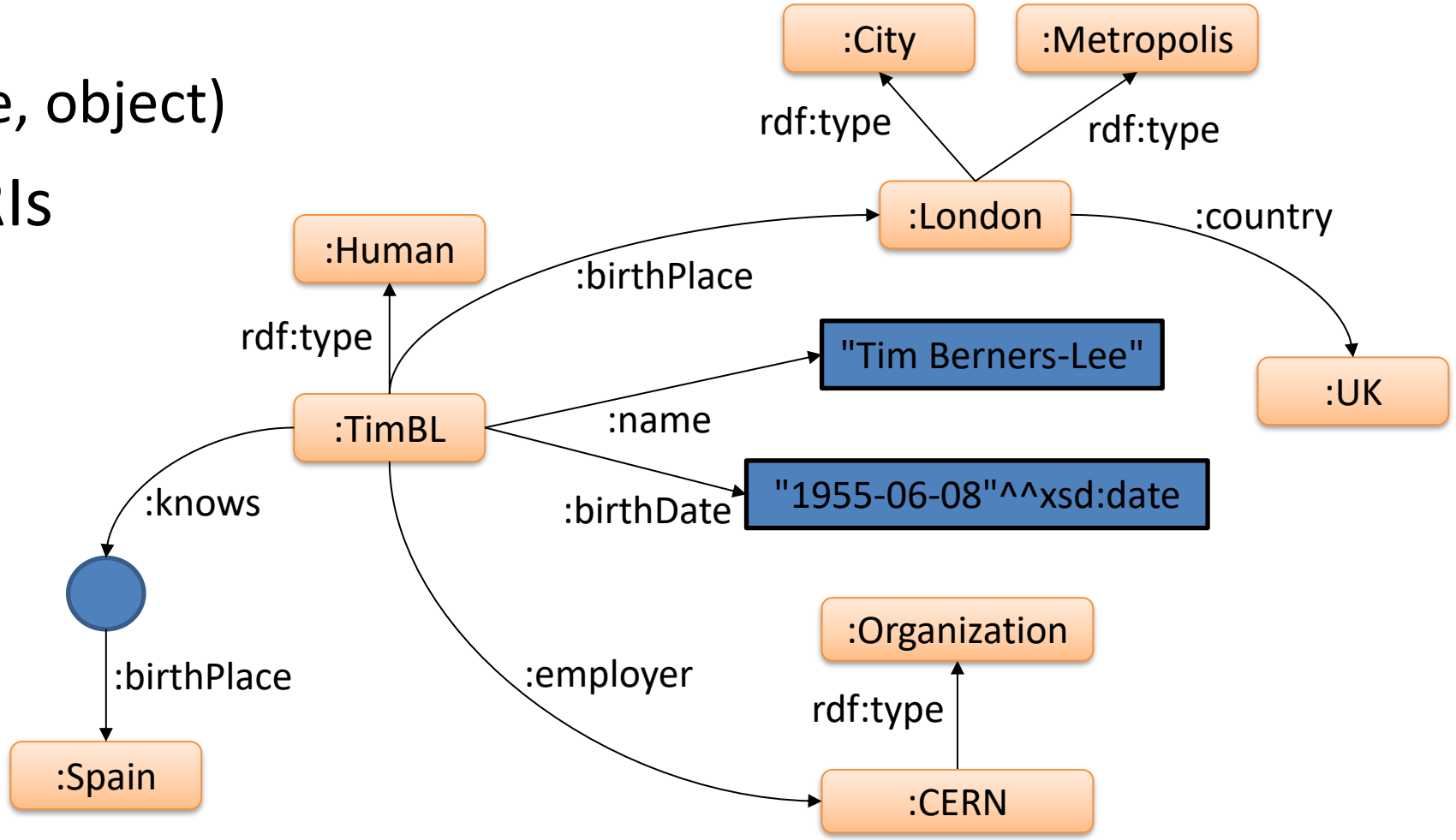
Based on triples

(subject, predicaje, object)

Most nodes are URIs

Interoperability

Simple & flexible



RDF ecosystem

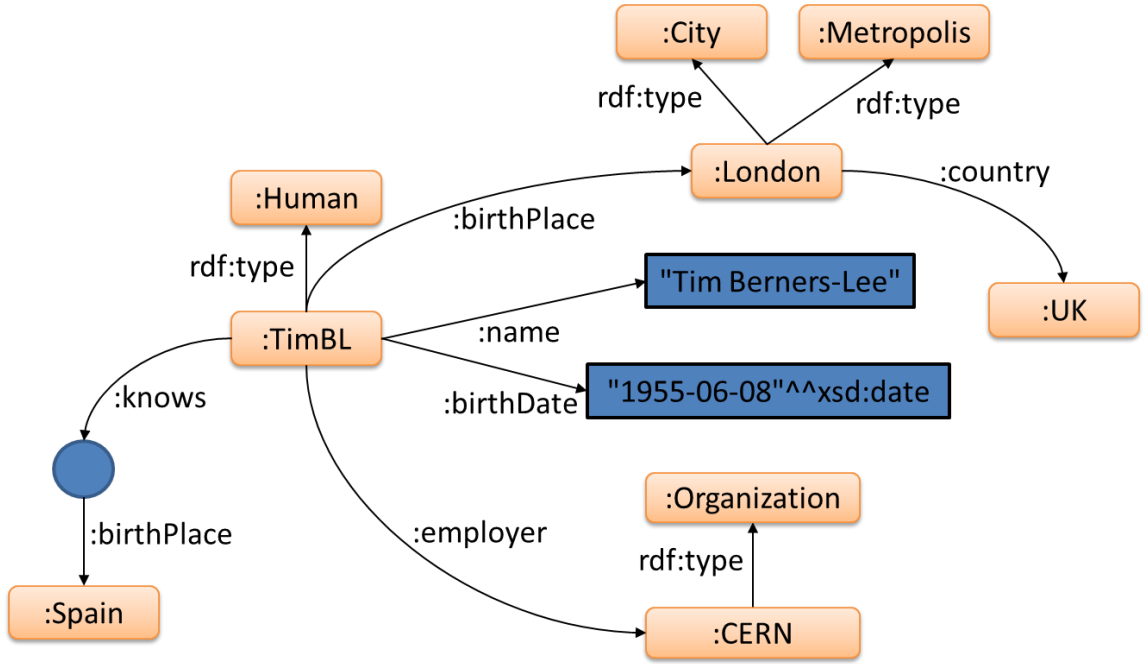
One data model, several syntaxes: Turtle, N-Triples, JSON-LD

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

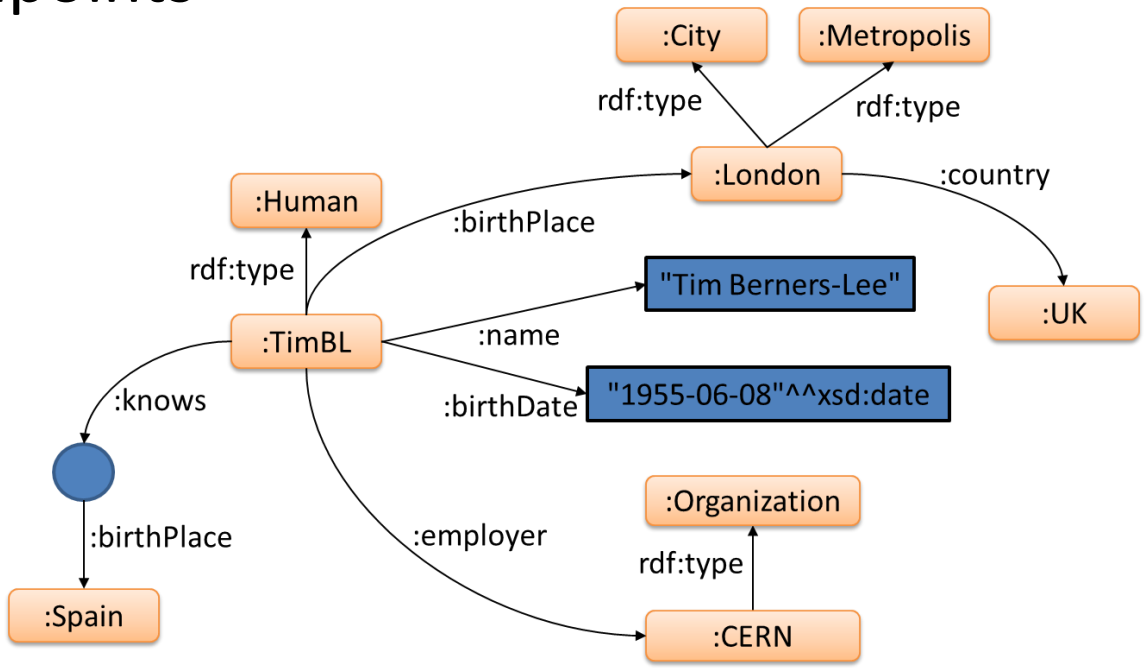
RDF ecosystem: SPARQL

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

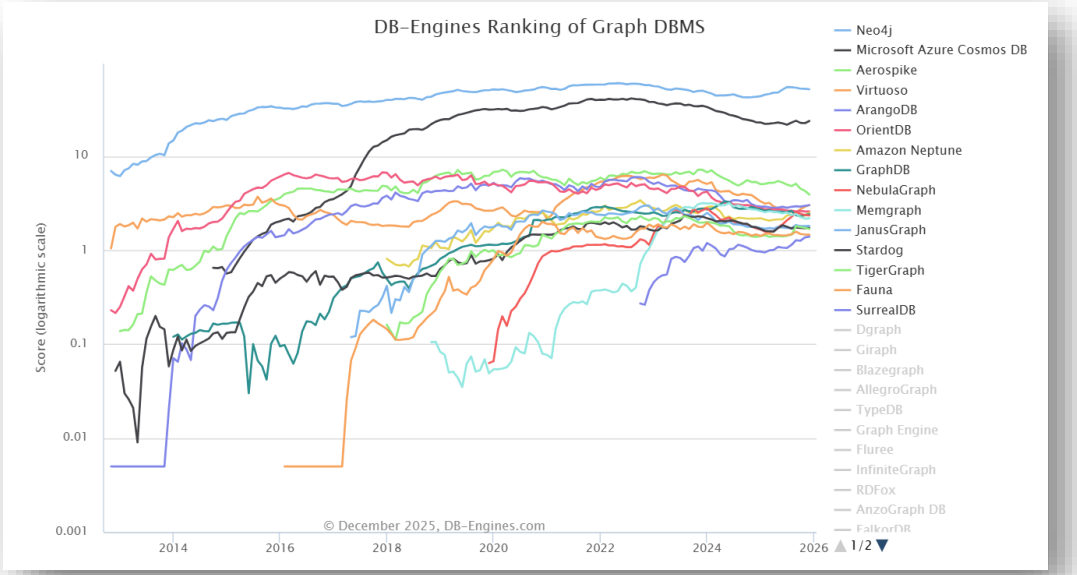
Labeled Property graphs

Popular model in industry

- neo4j, Amazon Neptune, Microsoft, etc

GQL (Graph Query Language)

- Published in 2024: ISO/IEC FDIS 39075
- Developed by the “SQL” committee
- Influenced by Cypher, PGSchema, etc.



Source: https://db-engines.com/en/ranking_trend/graph+dbms

Also adopted to implement Knowledge Graphs

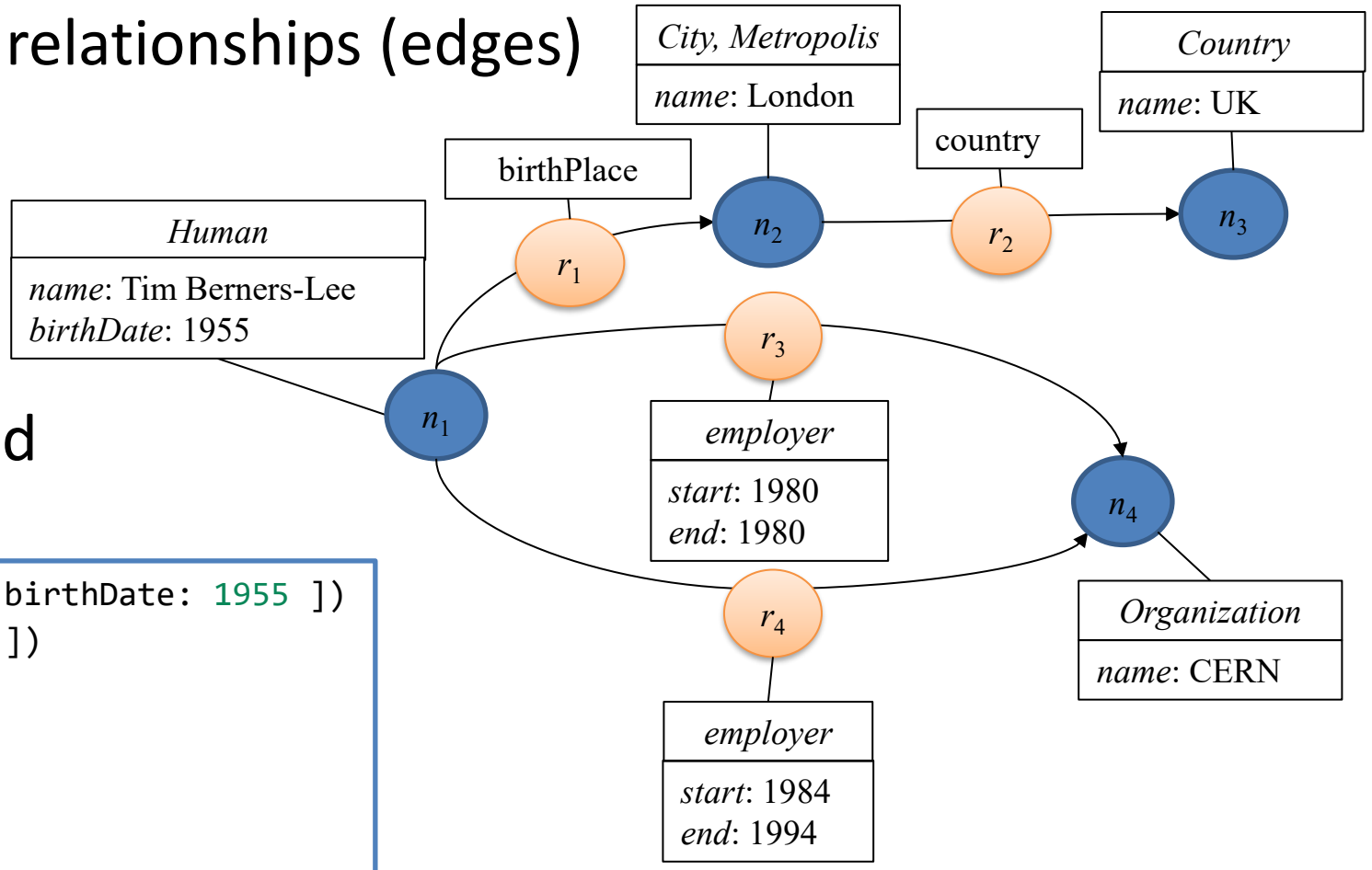
Property graphs

Graph structure with nodes and relationships (edges)

Nodes and edges can have:

- Labels
- A set of property-value pairs

Edges can be directed/undirected




```
(n1 {Person} [ name: "Tim Berners-Lee", birthDate: 1955 ])
(n2 {City, Metropolis} [ name: "London" ])
(n3 {Country} [ name: "UK" ])
(n4 {Organization} [ name: "CERN" ])

(n1) - (r1 {birthPlace}) -> (n2)
(n2) - (r2 {country}) -> (n3)
(n1) - (r3 {employer} [ start: 1980, end: 1980 ]) -> (n4)
(n1) - (r4 {employer} [ start: 1984, end: 1994 ]) -> (n4)
```

YarsPG notation

Wikidata based Knowledge Graphs

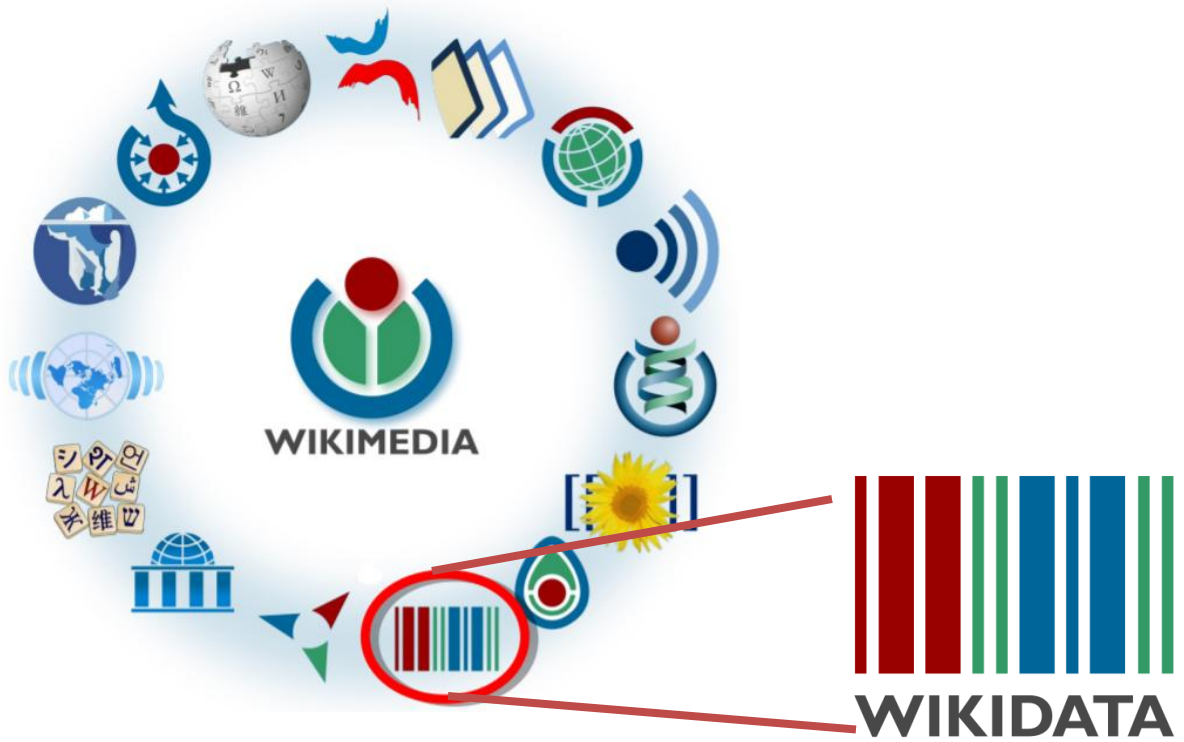
Wikidata was created in 2012 as a collaborative knowledge graph

https://www.wikidata.org/

Developed and supported by Wikimedia Deutschland

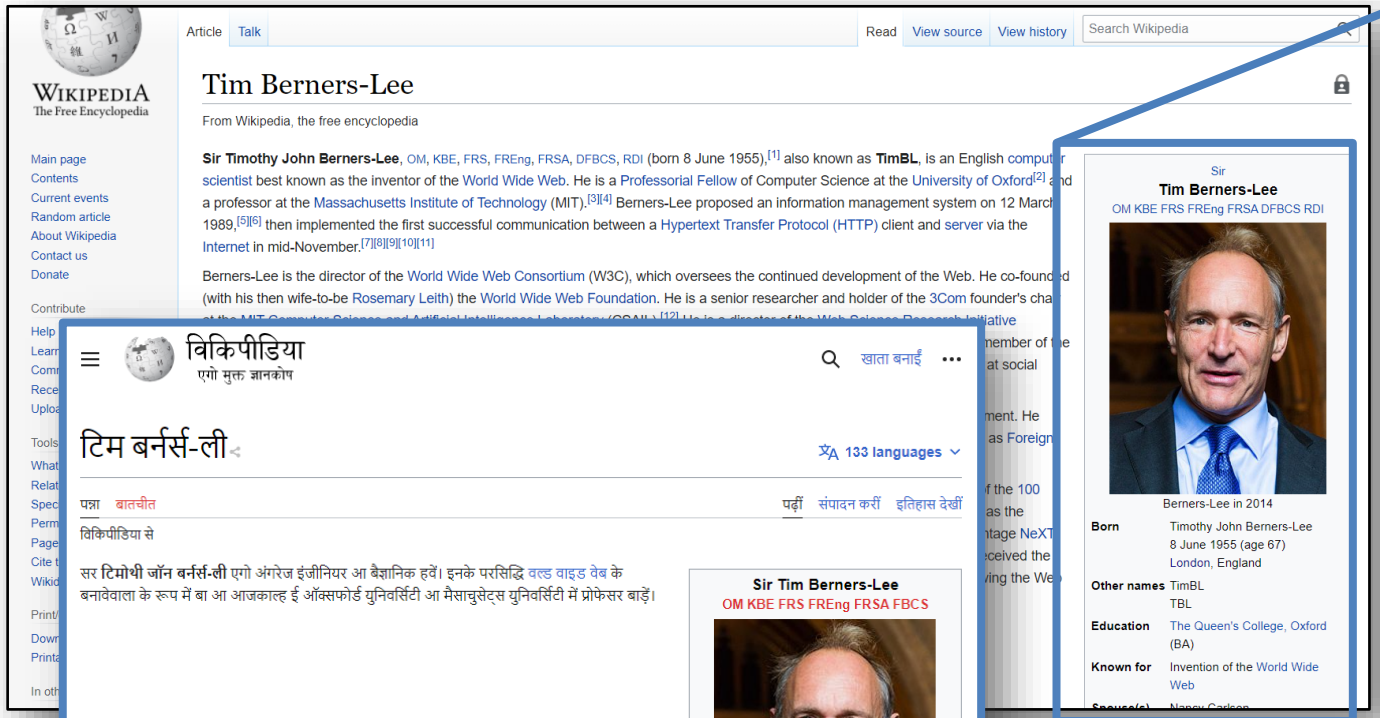
Initial goal:

Support multilingual infoboxes in Wikipedia



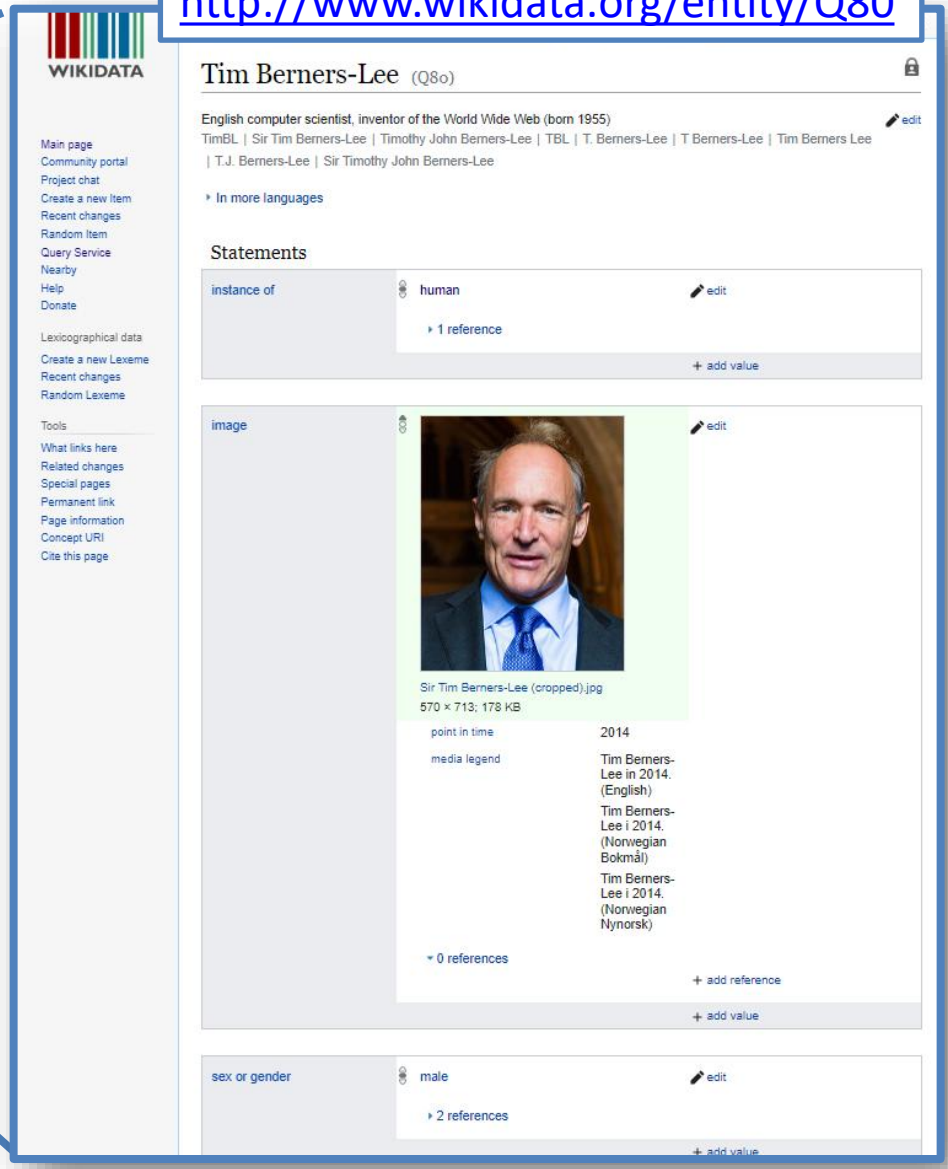
Wikidata

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

123,011,796 items, 2,535,483,822 edits (08/2026):
https://www.wikidata.org/wiki/Wikidata:Statistics

Free and open license: CC0

Multilingual

Collaborative (Humans and bots)

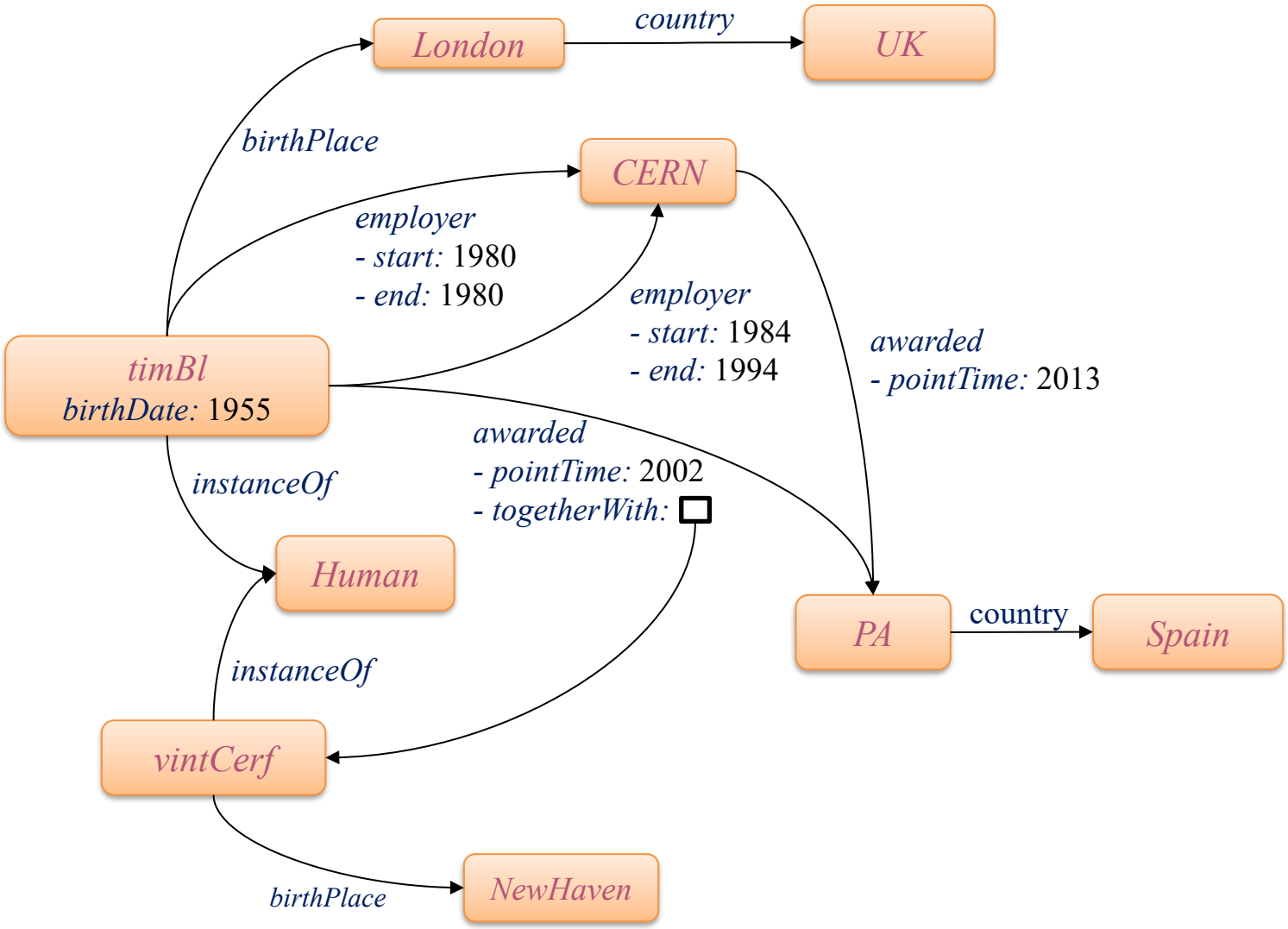
Public SPARQL endpoint

Software that supports Wikidata = Wikibase

Example from Wikidata

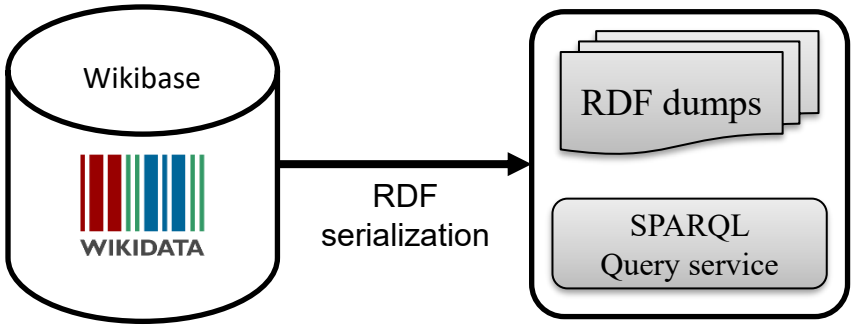
Information about Tim Berners Lee

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



Wikibase 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

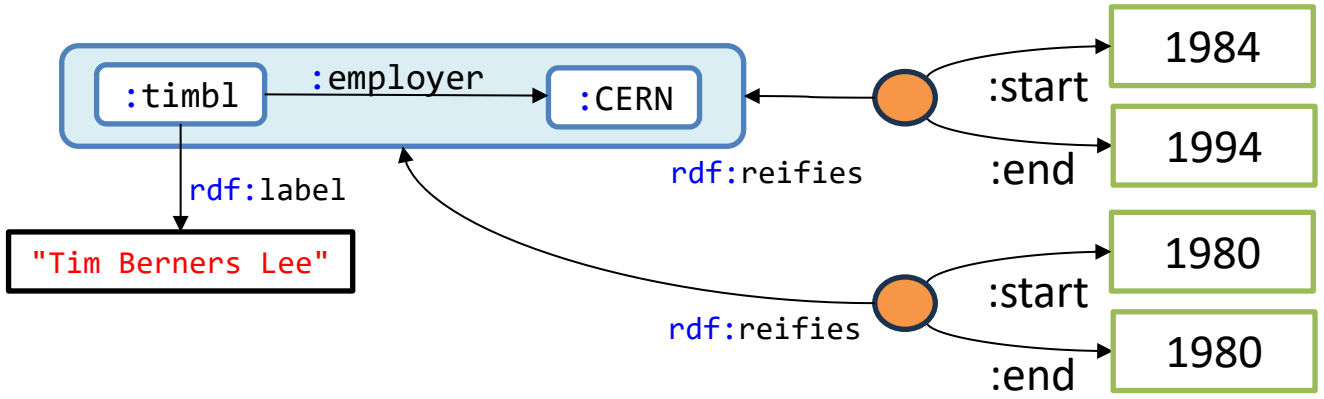
Under development (almost done): https://www.w3.org/TR/rdf12-concepts/

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

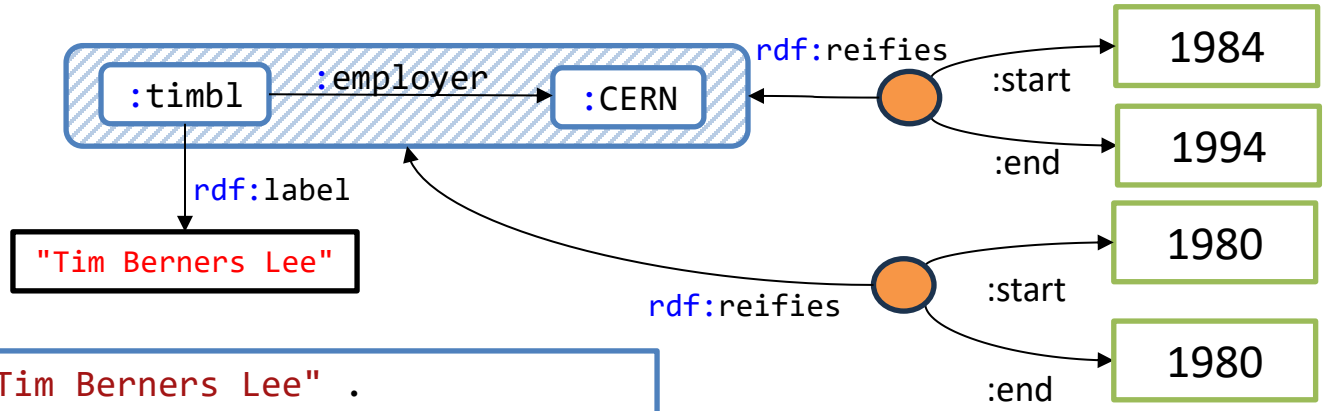
RDF 1.2: Non-asserted triple terms

Triple terms can be reified without being asserted

We can talk about a triple that says Tim's employer is the CERN

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 >>
    :start        "1984" ;
    :end          "1994" .
<< :timbl :employer :CERN >>
    :start        "1980" ;
    :end          "1980" .
```



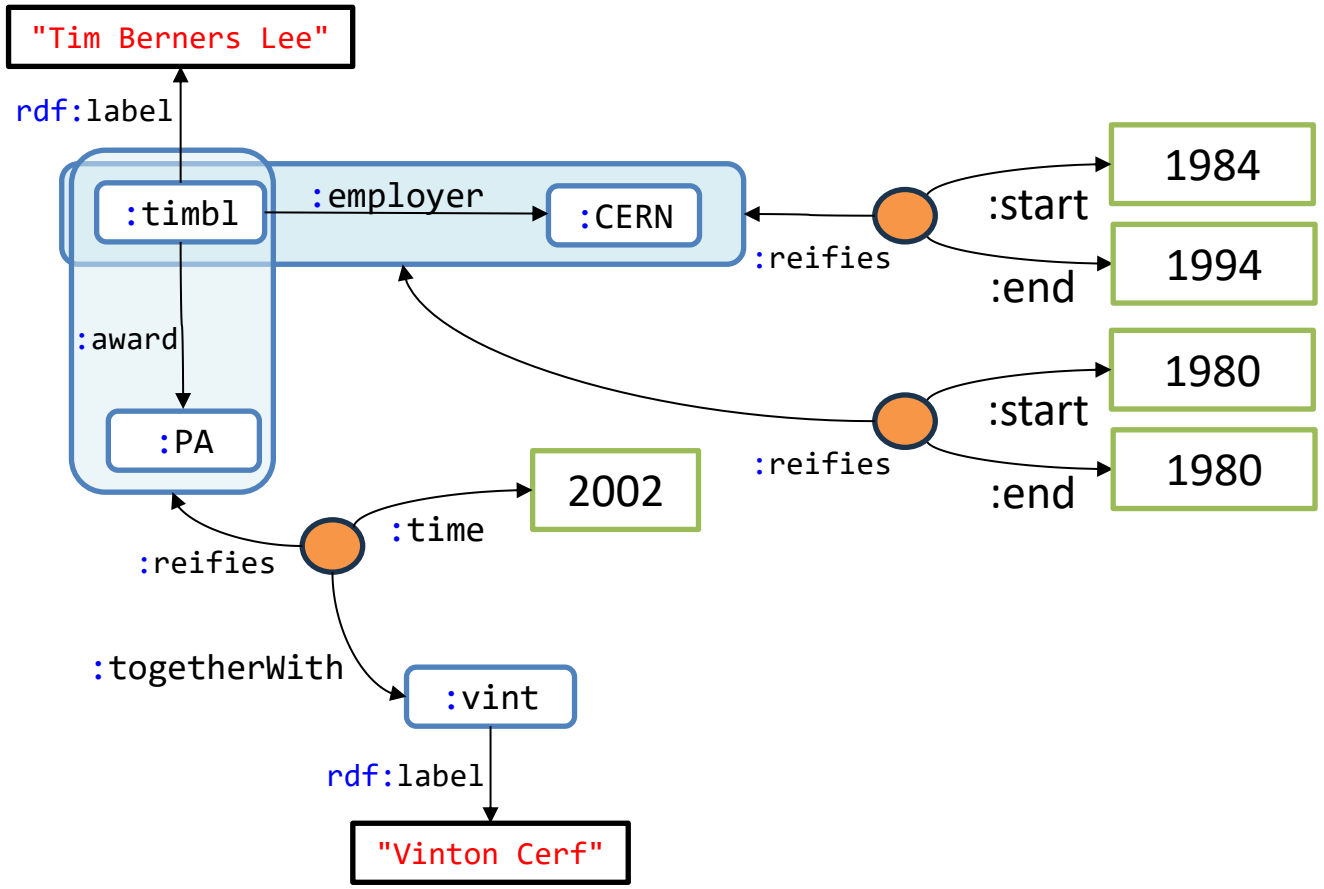

```
:timbl rdfs:label "Tim Berners Lee" .
_:1 rdf:reifies   <<( :timbl :employer :CERN )>> ;
    :start        "1984" ;
    :end          "1994" .
_:2 rdf:reifies   <<( :timbl :employer :CERN )>> ;
    :start        "1980" ;
    :end          "1980" .
```

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" .
```

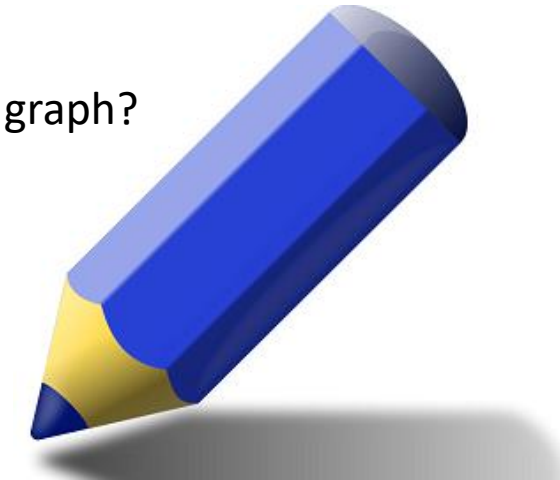


RDF 1.2

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

```
:timbl 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/

rudof supports RDF 1.2 and SPARQL 1.2 (thanks to Oxigraph)

Example query:

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

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

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:

- Shaping Wikibase and Wikidata graphs
- Shaping Property Graphs
- Shaping RDF 1.2

Some ideas for future work

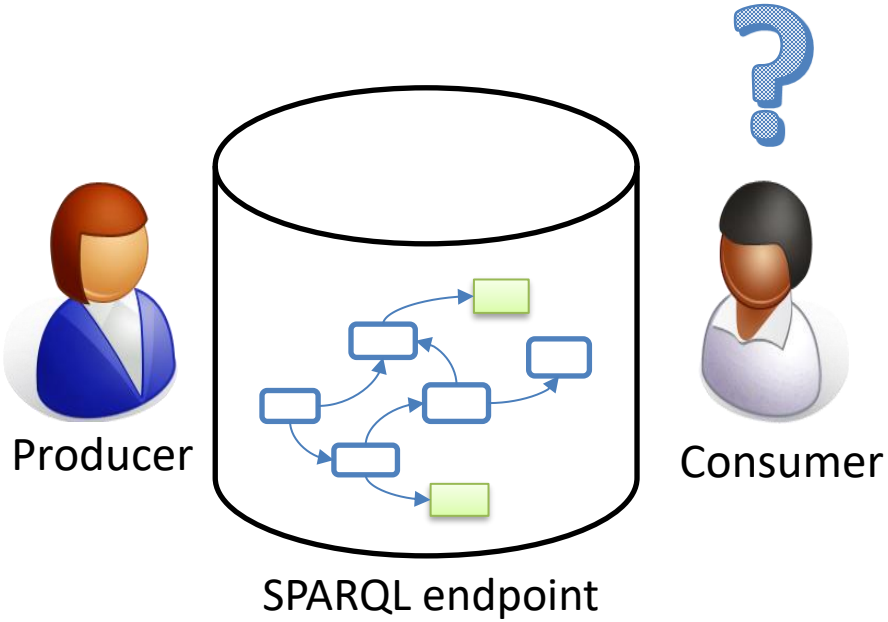
WESO

RDF, the good parts...

- As an integration language
  - RDF as a *lingua franca* for semantic web and linked data
  - Basis for knowledge representation
- Flexibility
  - Data can be adapted to multiple environments
  - Reusable data by default
- Ecosystem
  - RDF data stores & SPARQL
  - Multiple serializations: Turtle, JSON-LD, RDF/XML...
  - Inference and ontologies

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



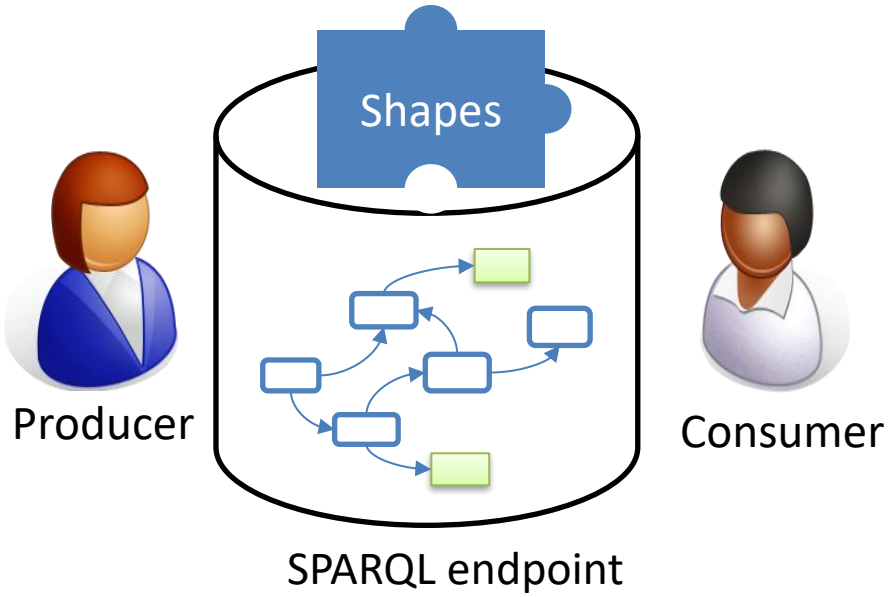
Why describe & validate RDF?

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 mistakes
  - Usually, RDF processors don't complain
  - RDF is very flexible
    - A nice feature *sometimes*

```
: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 :Dog                .

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

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

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

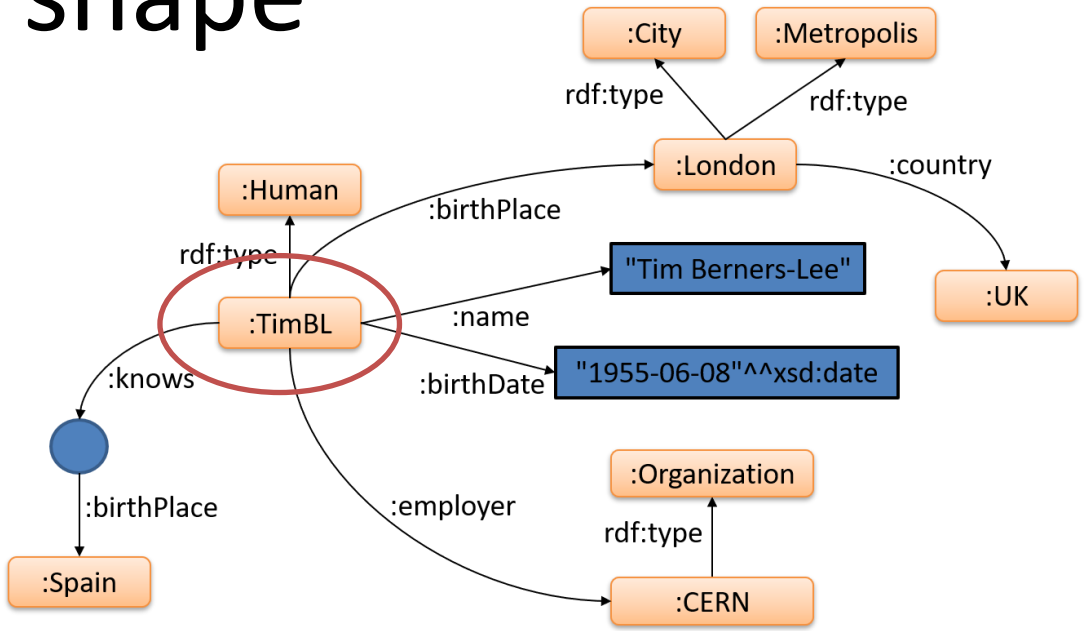
2025 IEEE ShEx (*work in progress*)

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 :name  "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

- Semantics inspired by regular expressions
- 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

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/>

:Person {
 :name        xsd:string        ;
 :knows       @:Person          *
}
```

Nodes conforming to `:Person` must

- Have property `:name` with a value of type `xsd:string` (exactly one)
- Have property `:knows` with values conforming to `:Person` (zero or more)

RDF Validation using ShEx

Schema

```
:Person {
 :name     xsd:string   ;
 :knows @:Person      *
}
```

Shape map

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

RDF Data

```
:a   :name     "Alice"                ;
     :knows    :b                     .
:b   :knows    :a                     ;
     :name     "Bob"                  .
:c   :name     "Carol", "Carole"      .
:d   :name     234                    .
:e   :namme    "Emily"                .
:f   :name     "Frank"                ;
     :email    "frank@mail.com"       ;
     :knows    :a, :c                 .
```

Try it (RDFShape)
Try it (Simple ShExDemo)

Validation process

Input: RDF data, ShEx schema, Shape map

Output: Result shape map

ShEx Schema

```
:Person {
 :name     xsd:string   ;
 :knows    @:Person     *
}
```

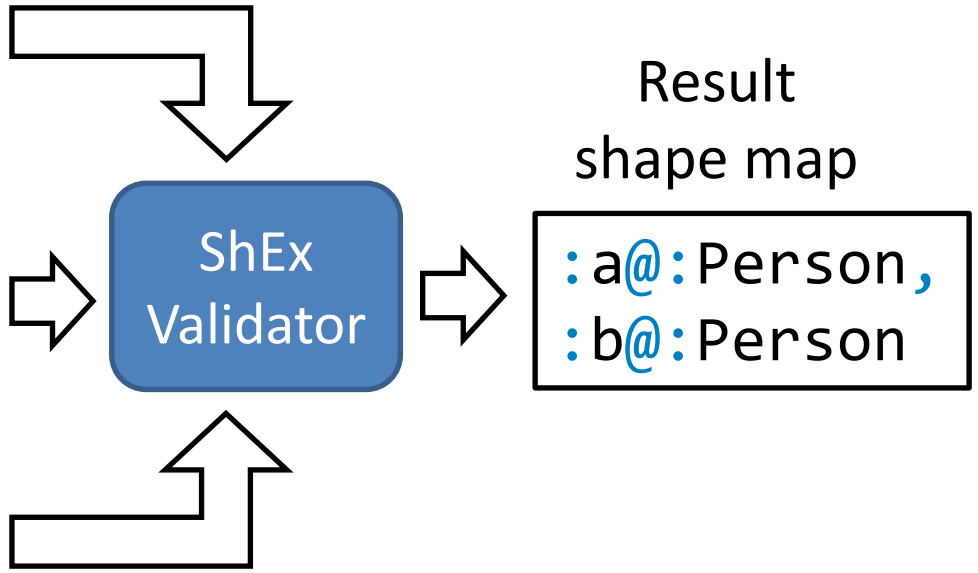
Shape map

```
:a@:Person
```

RDF data

```
:a   :name     "Alice" ;
     :knows    :b      .

:b   :knows    :a      ;
     :name     "Bob"   .
```



Result shape map

```
:a@:Person,
:b@:Person
```

More complex Shape Maps

Shape Maps define which nodes validate with which shapes

Examples:

`{FOCUS a :Person}@:Person`	https://tinyurl.com/466hp7h6
`{FOCUS :name _}@:Person`	https://tinyurl.com/yk9z3fz7

`SPARQL """`
`PREFIX : <http://example.org/>`
`SELECT ?person WHERE {`
`?person :name ?name;`
`:knows :a`
`}"""@:Person`
	https://tinyurl.com/438z5t8j

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

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)

Validation Report

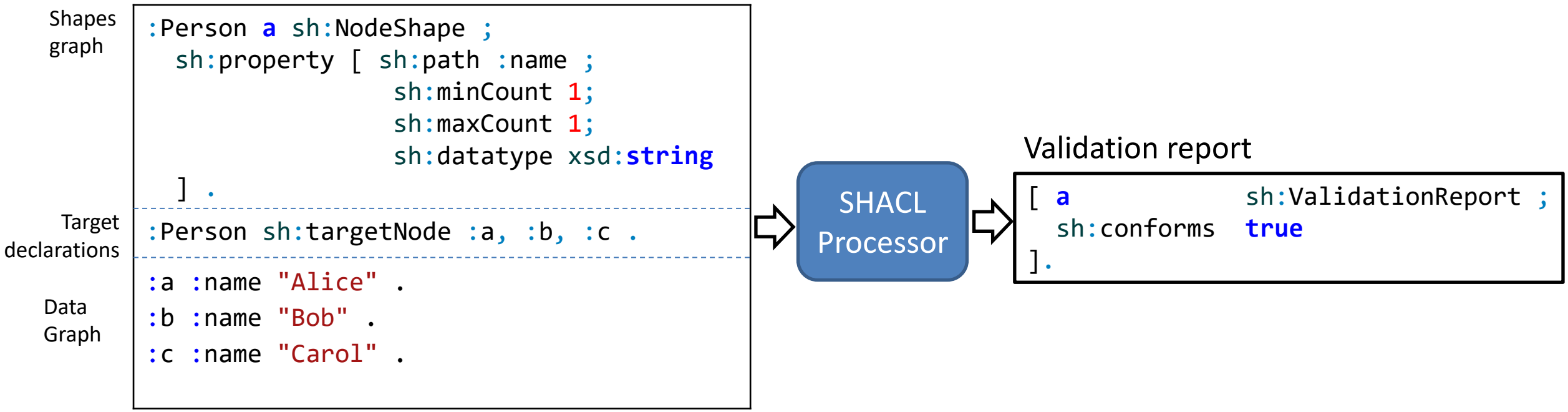
The output of the validation process is a list of violation errors

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    :b ;
   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

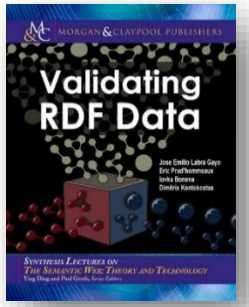


ShEx and SHACL: Several common features…

	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.)

DublinCore

WESO

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> * ;
}
:Place> {}
:Organization {}
```

In ShEx

Contents

Introduction to Knowledge graphs

Types of Knowledge Graphs:

- RDF, Property graphs, Wikibase, RDF-1.2

Shaping RDF: ShEx & SHACL, DCTAP

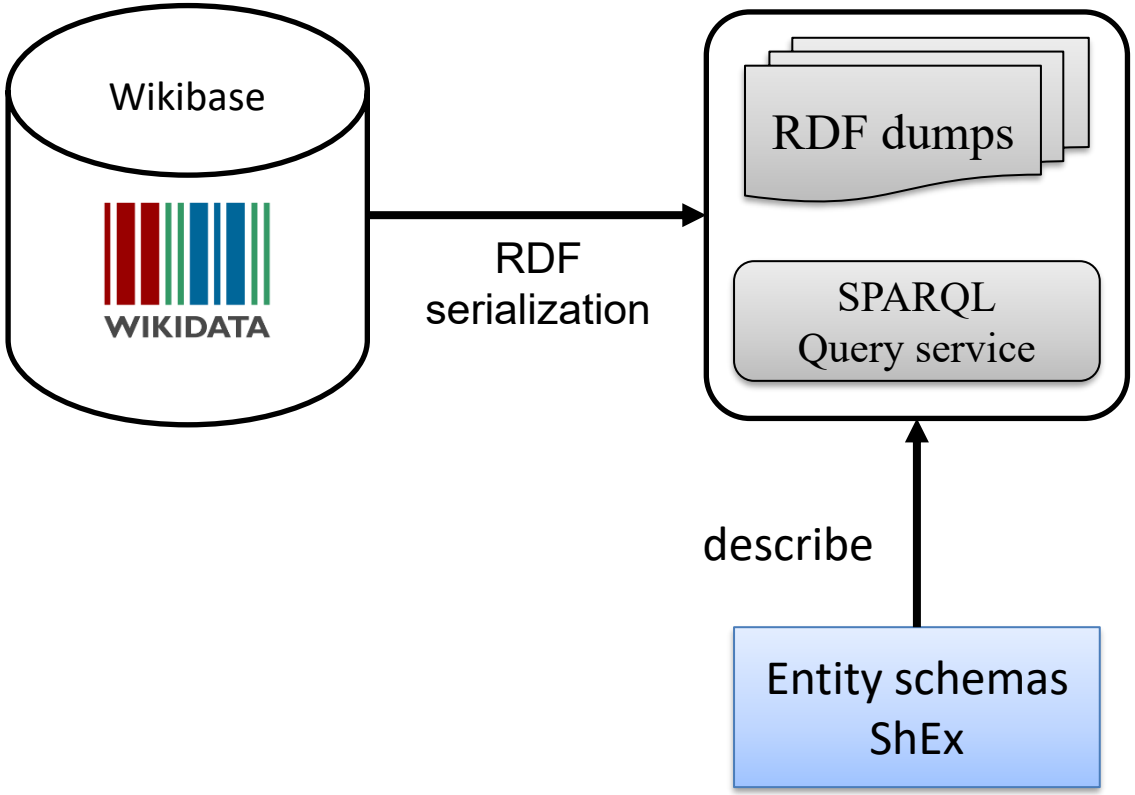
Shaping other types of Knowledge graphs:

- Shaping Wikibase and Wikidata graphs
- Shaping Property Graphs
- Shaping RDF 1.2

Some ideas for future work

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

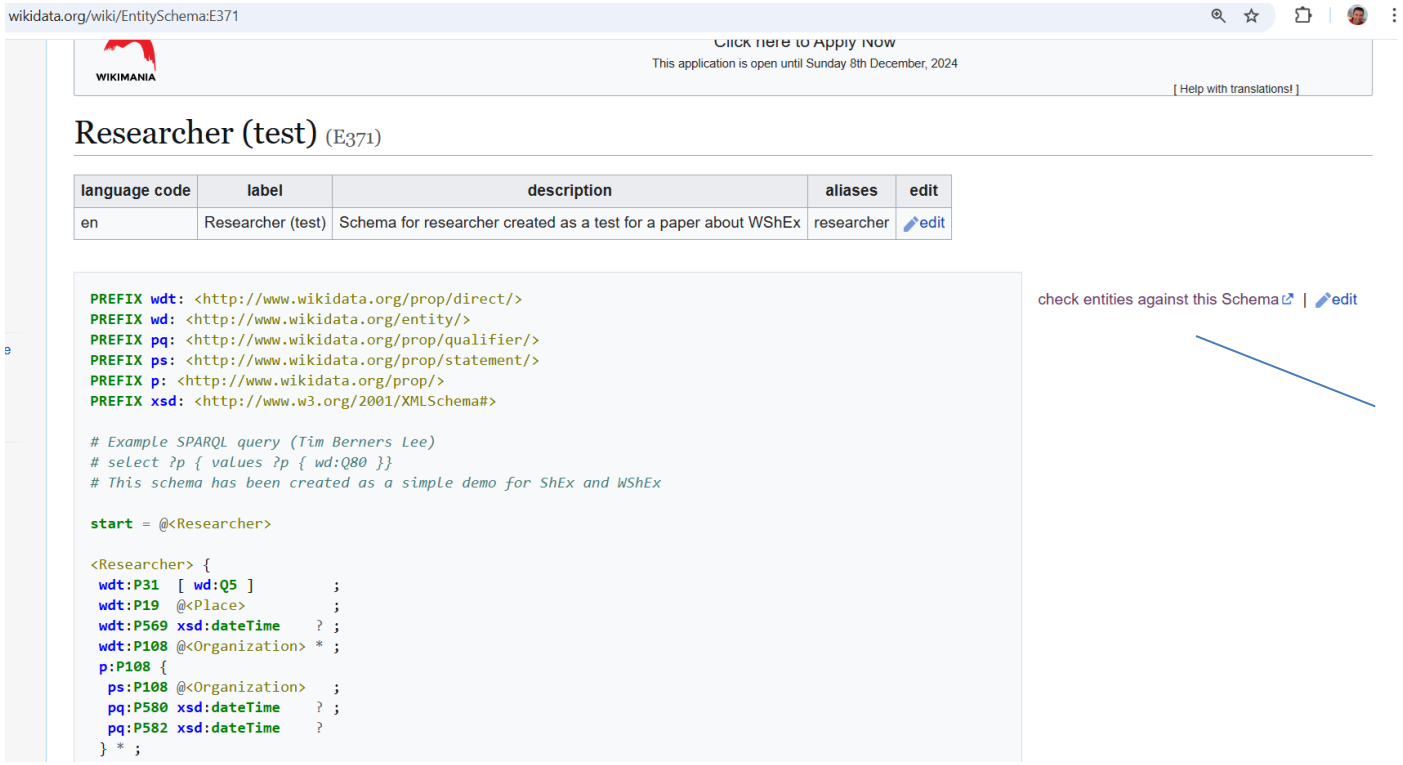
E371

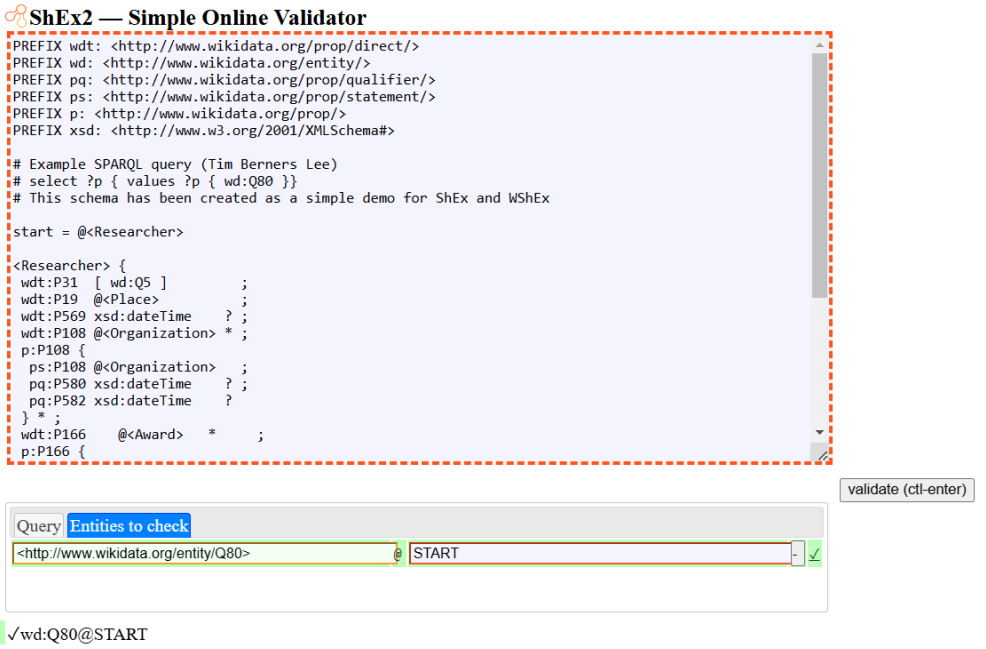
```
PREFIX pq: <.../prop/qualifier/>
PREFIX ps: <.../prop/statement/>
PREFIX p: <.../prop/>
PREFIX wdt: <.../prop/direct/>
PREFIX wd: <.../entity/>
PREFIX xsd: <...XMLSchema#>

<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

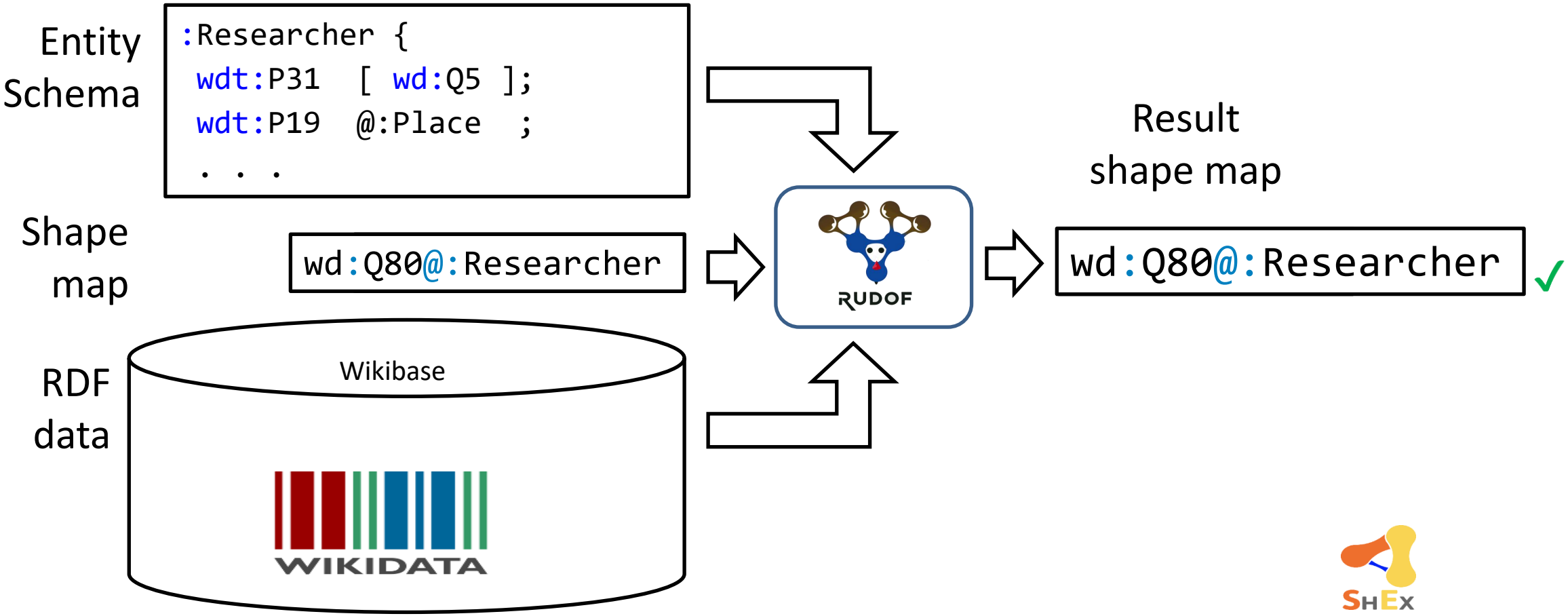
Example: https://www.wikidata.org/wiki/EntitySchema:E371





Using rudof with Wikibase

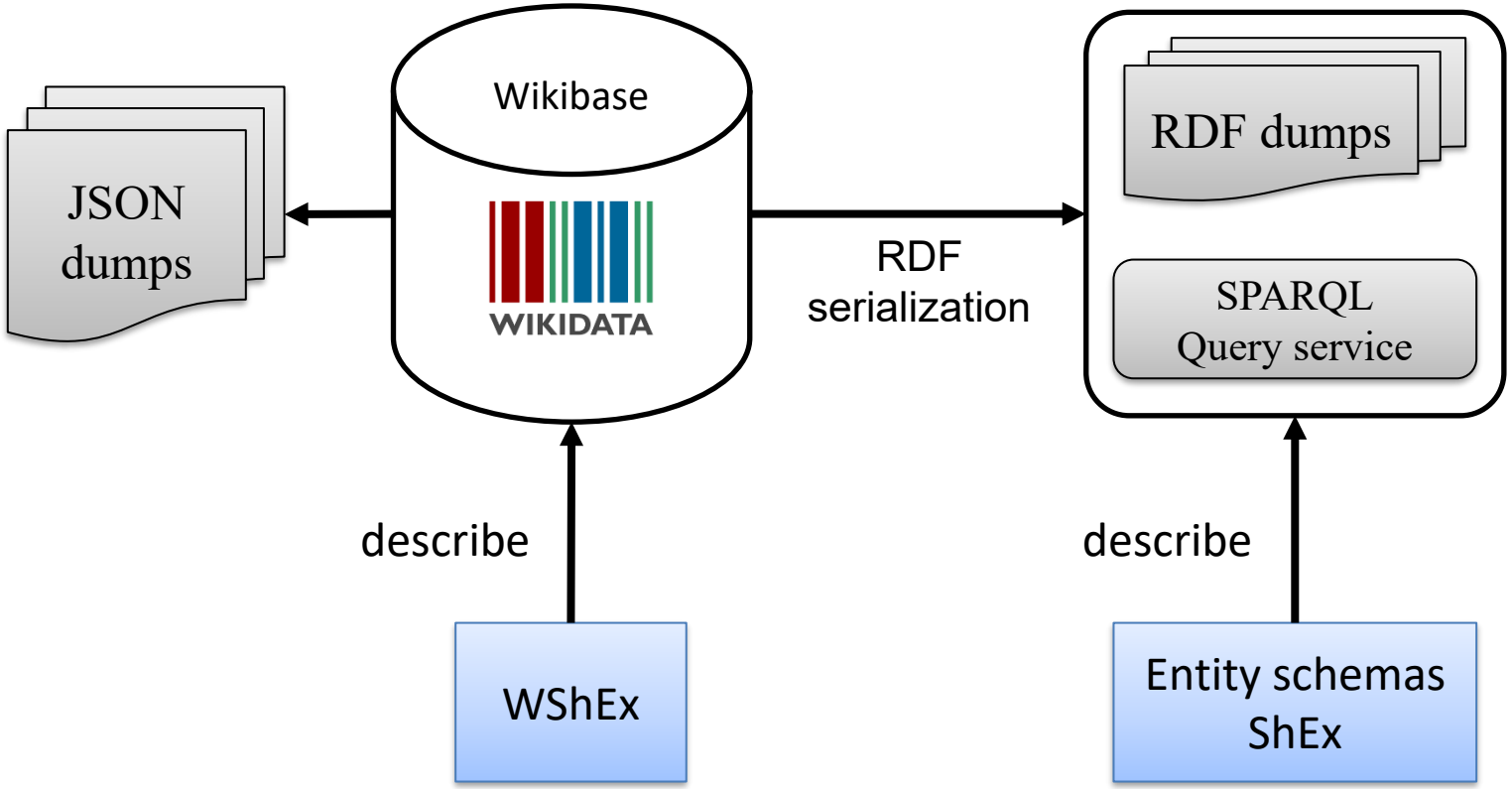
rudof supports validating against SPARQL endpoints



WShEx

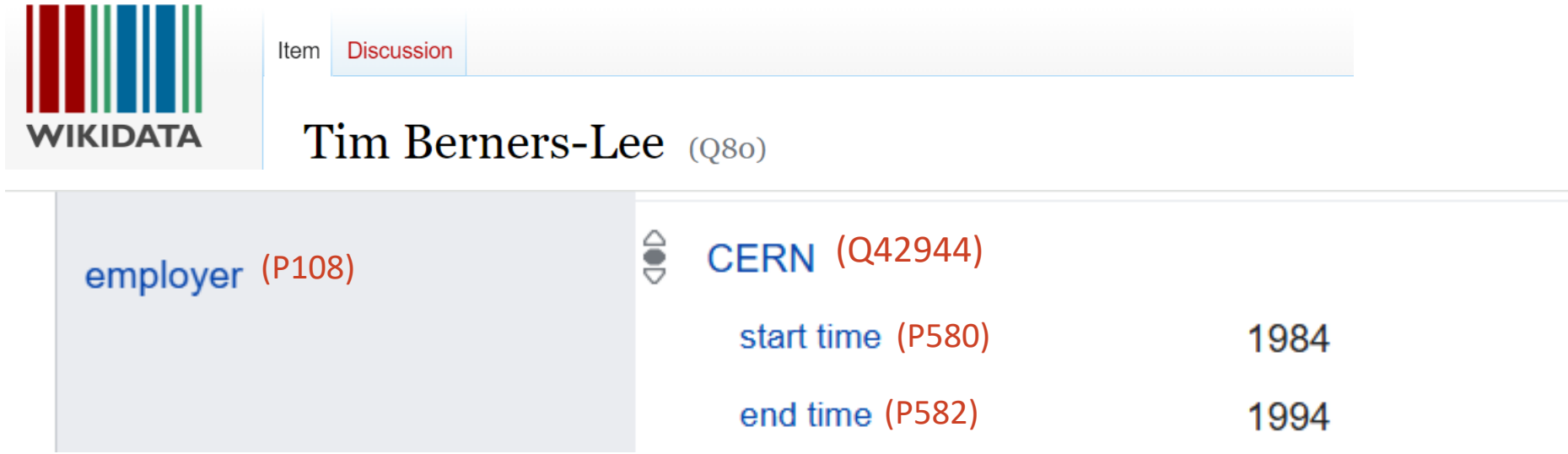
Although Entity Schemas and ShEx just work, they describe Wikibase indirectly

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

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

https://arxiv.org/abs/2208.02697

Contents

Introduction to Knowledge graphs

Types of Knowledge Graphs:

- RDF, Property graphs, Wikibase, RDF-1.2

Shaping RDF: ShEx & SHACL, DCTAP

Shaping other types of Knowledge graphs:

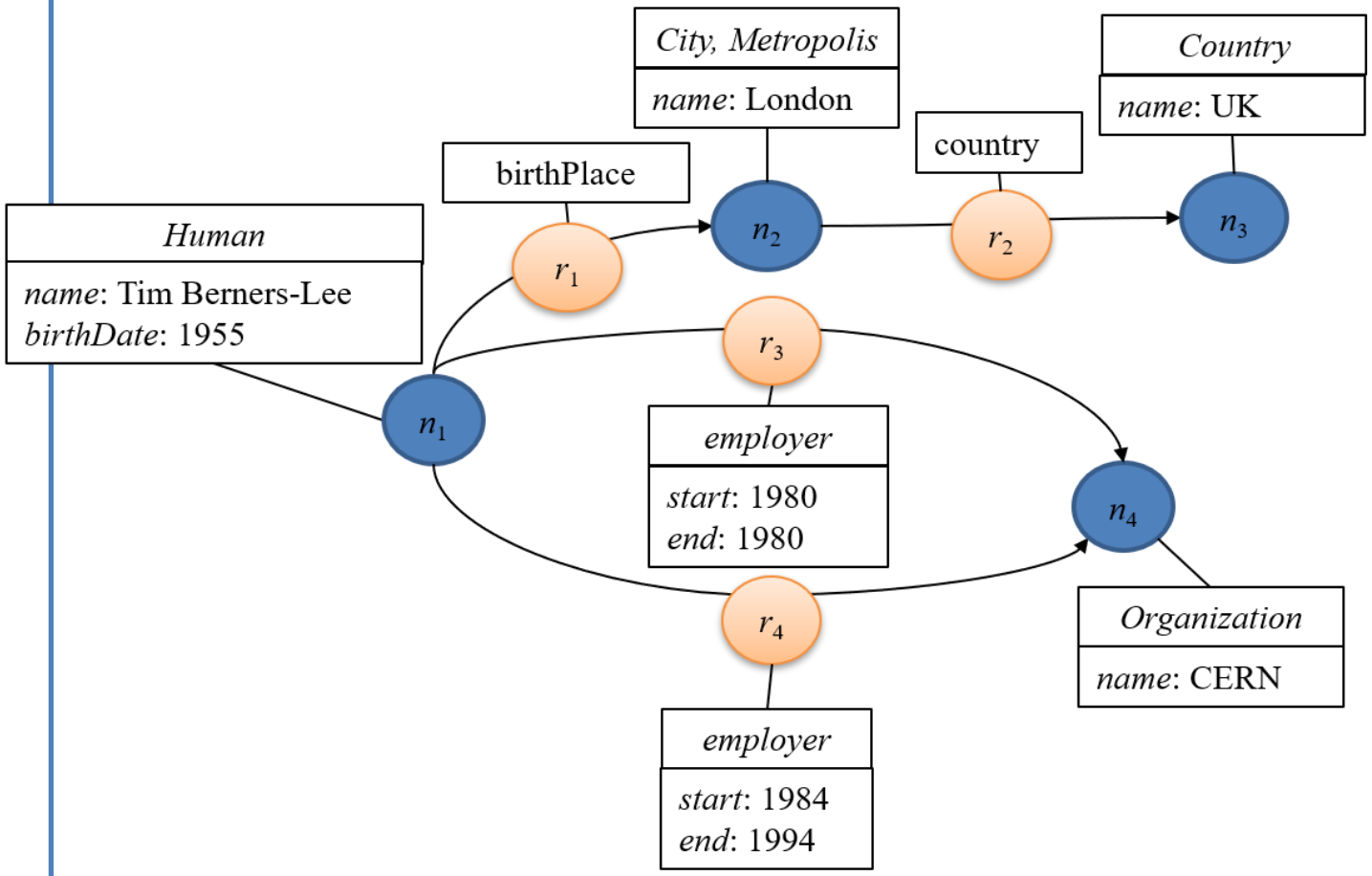
- Shaping Wikibase and Wikidata graphs
- Shaping Property Graphs
- Shaping RDF 1.2

Some ideas for future work

PGSchema

Paper: *PG-Schema: Schemas for proprety graphs,* https://doi.org/10.1145/3589778

```
CREATE NODE TYPE ( PersonType : Person {
   (name: STRING ),
   OPTIONAL birthDate: INTEGER }) ;
CREATE NODE TYPE ( OrganizationType : Organization {
   (name: STRING ) }) ;
CREATE NODE TYPE ( PlaceType : City & Metropolis {
   (name: STRING ) }) ;
CREATE NODE TYPE ( CountryType : Country {
   (name: STRING ) }) ;
CREATE EDGE TYPE
 (:@PersonType) -[EmployerType : employer {
      start: INTEGER,
      end:   INTEGER }]-> (:@OrganizationType) ;
CREATE EDGE TYPE (:@PersonType)
                 -[BirthPlaceType : birthPlace]->
                 (:@PlaceType) ;
CREATE EDGE TYPE (:@PlaceType)
                 -[CountryEdgeType : country]->
                 (:@CountryType)
```



PGSchema prototype in rudof

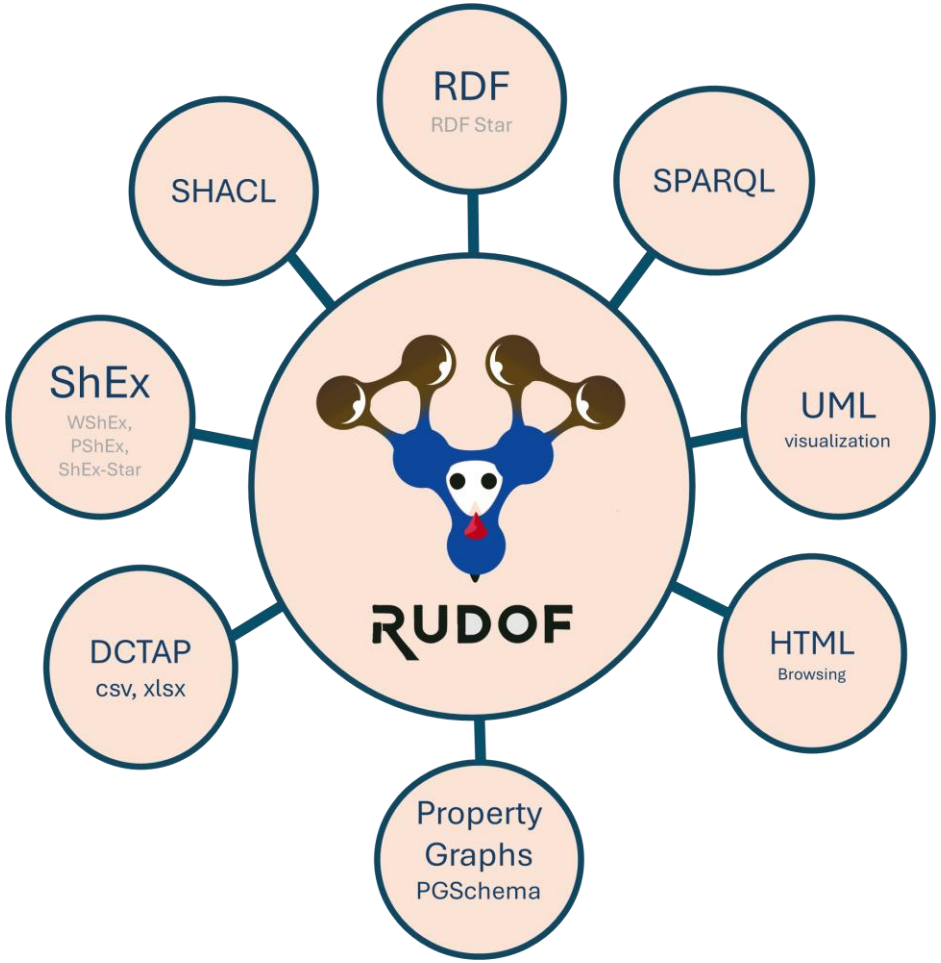
Prototype implementation in rudof

3 inputs:

- Property graph using YarsPG notation
- PGSchema declaration
- Map declaration ≈ typemap

Output:

- Validation result



Contents

Introduction to Knowledge graphs

Types of Knowledge Graphs:

- RDF, Property graphs, Wikibase, RDF-1.2

Shaping RDF: ShEx & SHACL, DCTAP

Shaping other types of Knowledge graphs:

- Shaping Wikibase and Wikidata graphs
- Shaping Property Graphs
- Shaping RDF 1.2

Some ideas for future work

ShEx for RDF 1.2

Note: work in progress…(rudof's roadmap)

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

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

SHACL 1.2

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

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

New

The example is already supported in rudof

```
: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
       ]]] .
```

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:

- Shaping Wikibase and Wikidata graphs
- Shaping Property Graphs
- Shaping RDF 1.2

Some ideas for future work

ShEx/SHACL in practice

Static analysis and performance

Benchmarking and optimizing rudof's SHACL implementation

Extending and Optimizing the rudof SHACL validator, Á. García, S, Bustamante, Jose E. Labra, O. Corcho

DMKG2026, 2nd International Workshop on Data Management for Knowledge Graphs, ISWC Conference 2026, Bari, Italy

Hypothesis: Some subsets of ShEx/SHACL can be very performant

Improve validation reporting and messages

Hypothesis: Validation results ≈ proof tree

Incremental and streaming validation

Hypothesis: Derivatives algorithm can work in streaming mode

Property Graphs

Adding cardinality constraints to Edge types

- Express more complex topology definitions ⇢ ShEx|SHACL

Integrate with existing implementations

- Neo4j, Amazon Neptune, Microsoft Fabric, etc.

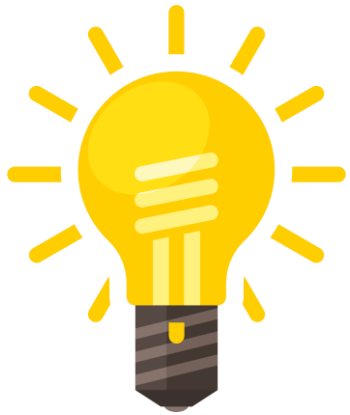
Increase the expressivity of current type maps

- Selection of nodes/edges ≈ GQL query

Performance & benchmarks

Wikibase & Wikidata

- Performance and validation in practice
  - Leverage on QLever (https://rudof-project.github.io/rudof/cli_usage/backend.html)
  - Caching validation results & parallel validation
  - Representing property constraints as Shapes
- WShEx
  - Implement WShEx in rudof
- Subsetting approaches based on shapes
  - Port wdsub to rudof



Convergence on Graph Schema languages

Common Foundations for SHACL, ShEx, and PG-Schema, S. Ahmetaj, I. Boneva, J. Hidders, K. Hose, M. Jakubowski, J. Labra, W. Martens, F. Mogavero, F. Murlak, C. Okulmus, A. Polleres, O. Savković, M. Šimkus, D. Tomaszuk, International World Wide Web Conference, Sidney, Australia, 2025

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

Common Foundations for Recursive Shape Languages, S. Ahmetaj, I. Boneva, J. Hidders, M. Jakubowski , J. Labra, W. Martens, F. Mogavero, F. Murlak, C. Okulmus, O. Savković, M. Šimkus, D. Tomaszuk, 23rd International Conference on Principles of Knowledge Representation and Reasoning, Lisbon, 2026

sheval: A Framework for Evaluating Shape Languages, S. Ahmetaj, I. Boneva, J. Hidders, M. Jakubowski , J. Labra, W. Martens, F. Murlak, C. Okulmus, O. Savković, M. Šimkus, D. Tomaszuk, DMKG2026, 2nd International Workshop on Data Management for Knowledge Graphs, ISWC Conference 2026, Bari, Italy

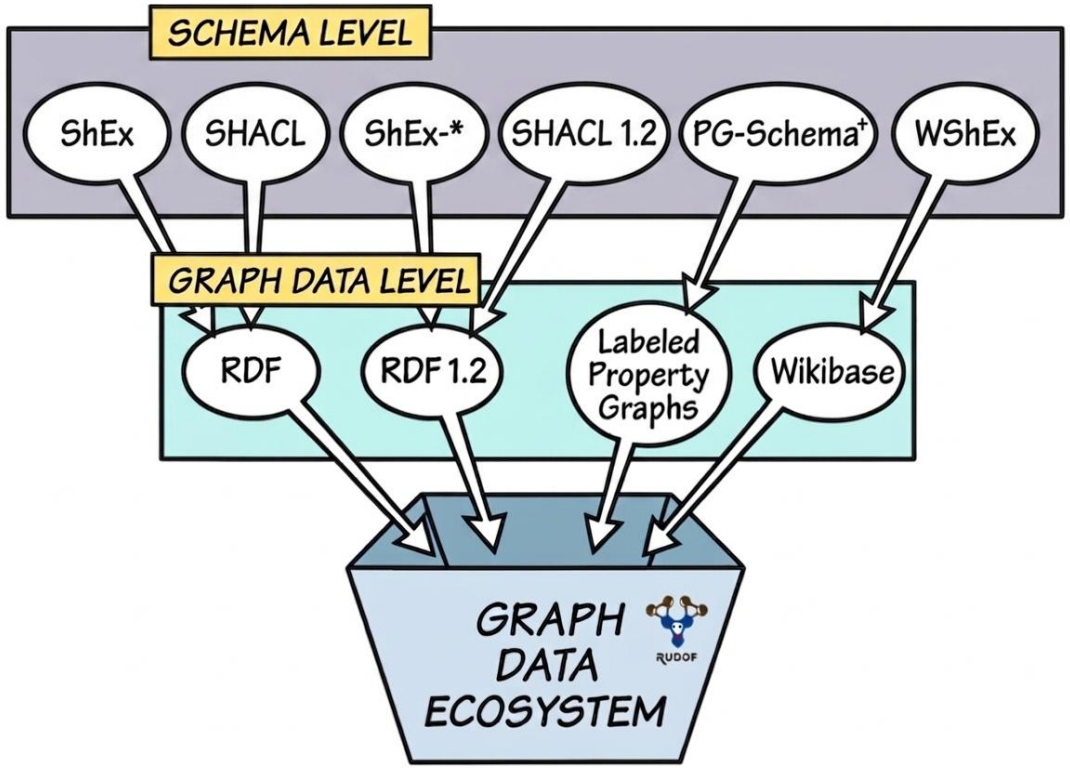
Translating RDF 1.2 Graphs to Property Graphs, M. Jakubowski, D. Tomaszuk, D. Fernández, R. Taelman, R. Dedecker, J. Labra, K. Hose, DMKG2026, 2nd International Workshop on Data Management for Knowledge Graphs, ISWC Conference 2026, Bari, Italy

Combination with other technologies

- Shapes + Rules
  - Combining shapes and rules
  - SHACL 1.2 rules vs SPARQL Rules
- Inferring/Extracting Shapes from existing graphs
  - Implement sheXer in Rust, extend sheXer to Property graphs
- Knowledge Graph Construction
  - Implement ShExML in Rust

Conclusions

- Several types of Knowledge Graphs
  - Different models
  - Could they converge in the future?
- Shapes can increase quality of KGs
  - Different types of KGs
    - Wikibase graphs: ShEx, WShEx
    - Property graphs: PGSchema, GQL, …
    - RDF 1.2: ShEx (next), SHACL 1.2
  - Could the shapes languages converge in the future?



Picture Generated by Gemini & ChatGPT

End of presentation