Validating RDF data: Challenges and perspectives

Jose Emilio Labr Gayo

WESO Research Group (WEb Semantics Oviedo)

University of Oviedo, Spain

About me...

Founded WESO (Web Semantics Oviedo) research group

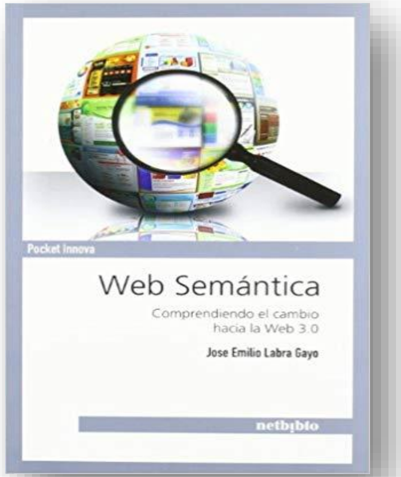
- Practical applications of semantic technologies since 2004
- Several domains: e-Government, e-Health

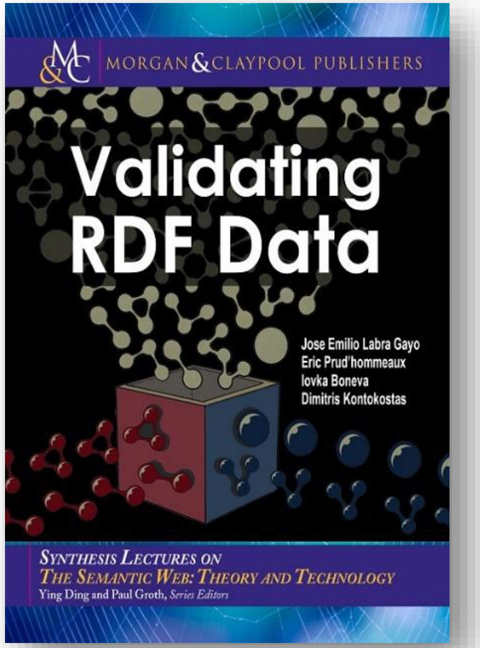
Some books:

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

...and software:

- SHaclEX (Scala library, implements ShEx & SHACL)
- RDFShape (RDF playground)





HTML version: http://book.validatingrdf.com
Examples: https://github.com/labra/validatingRDFBookExamples

Contents

Short intro to semantic Web & knowledge graphs

Short overview of RDF

Understanding the RDF validation problem

2 technologies: ShEx/SHACL

Challenges & perspectives

For longer presentations about ShEx/SHACL, see the ISWC'18 tutorial slides:
http://www.validatingrdf.com/tutorial/iswc2018/
Tutorial at Summer School

Semantic Web

Vision of the Web as **web of data**

- Not only pages, but also data: linked data
- Data *understandable* by machines

Related with:

- Big Data
- Artificial intelligence
  - Knowledge representation

Example: Wikipedia vs Wikidata

WIKIDATA

Tim Berners-Lee
Source: Wikipedia

1,677,782,739
Websites online right now

Source: http://www.internetlivestats.com/total-number-of-websites/
Date: 05/04/2019 (19:05h)

Who consumes Web information?

People

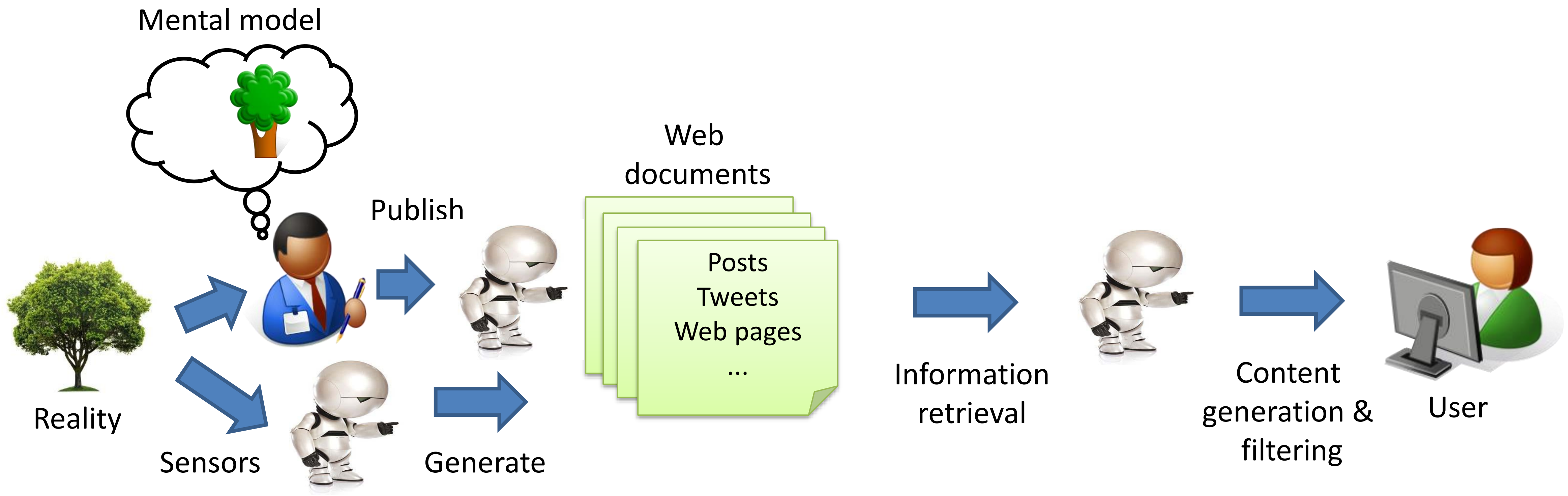
- We access through some device

...and **Machines**

- They show us web pages (browsers)
- ...but also manage information (bots)
  - Filtering content
  - Doing suggestions
  - ...

"If Google doesn't *understand* your Web page, it almost doesn't exist"

Web information



Web information must be machine-processable
Semantic web technologies focus on how to achieve this goal

People vs Machines

People	Machines
Creativity, imagination	Programmed for some tasks
Unpredictable (we do mistakes)	Predictable (no mistakes*)
We get tired with repetitive tasks	No problem with repetitive tasks
Understanding based on context	Problems to understand the context

*when well programmed

Information ***understandable*** by machines?

Problem: Ambiguity and context identification

Example: **"Oviedo has a temperature of 36 degrees"**

Identifiers (URIs) in Wikidata

Oviedo ? It can be... A city in Spain — http://wikidata.org/entity/Q14317

...or a city in Florida, USA — http://www.wikidata.org/entity/Q1813449

...or a soccer player — http://www.wikidata.org/entity/Q325997

...has a temperatrure of... — https://www.wikidata.org/wiki/Property:P2076

Machine representation

http://www.wikidata.org/entity/Q14317 → https://www.wikidata.org/prop/direct/P2076 → 36^^<http://www.w3.org/2001/XMLSchema#integer>

Knowledge representation for machines

Simple statement (triple 3 elements): subject, predicate, object

http://www.wikidata.org/entity/Q14317 —https://www.wikidata.org/prop/direct/P2076→ 36^^<http://www.w3.org/2001/XMLSchema#integer>

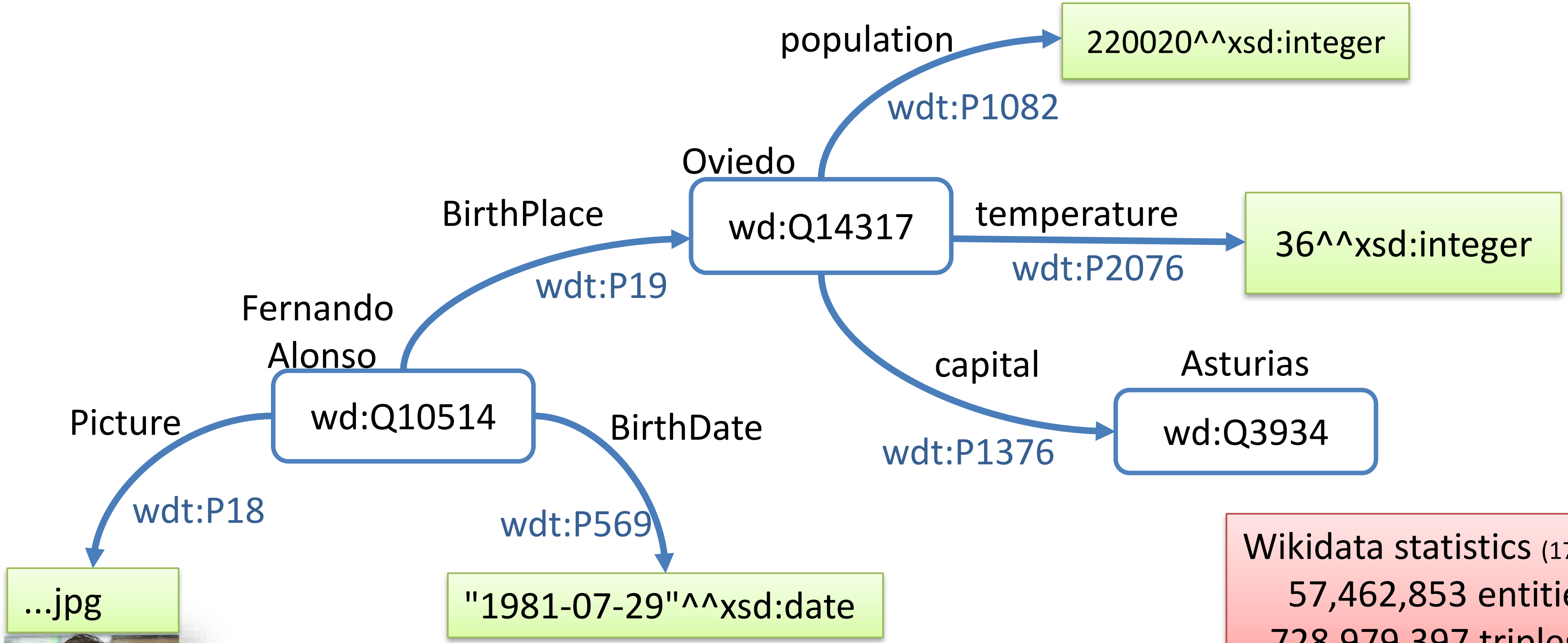
subject predicate object

Prefix declarations help simplify long URIs

```
wd:   <http://www.wikidata.org/entity/>
wdt:  <https://www.wikidata.org/prop/direct/>
xsd:   <http://www.w3.org/2001/XMLSchema#>
```

wd:Q14317 —wdt:P2076→ 36^^xsd:integer

Knowledge graphs

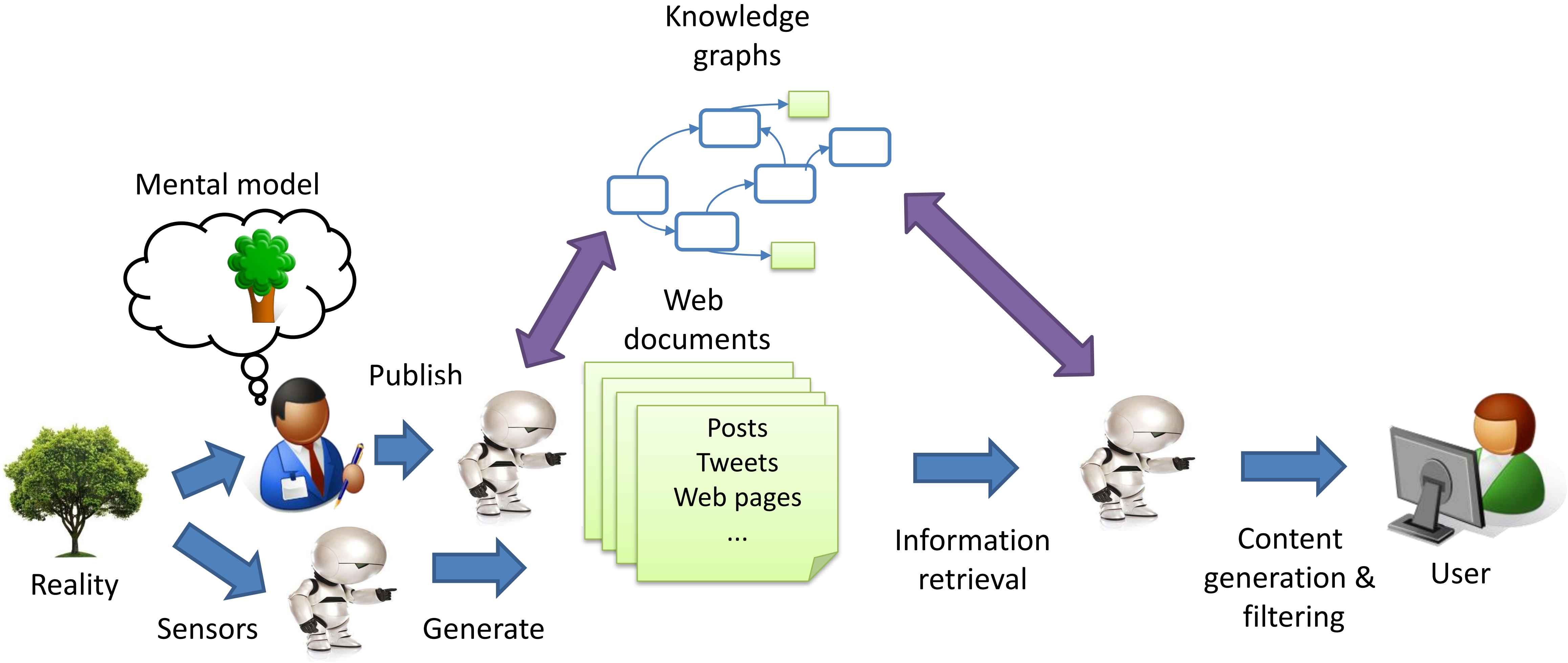


Wikidata statistics (17-07-2019)
57,462,853 entities
728,979,397 triples

Source: https://www.wikidata.org/wiki/Wikidata:Statistics

Wikidata is an example of a knowledge graph
There are a lot more:
- Open: DBpedia, BabelNet, etc
- Proprietary: Google, facebook, Microsoft, etc. also have knowledge graphs

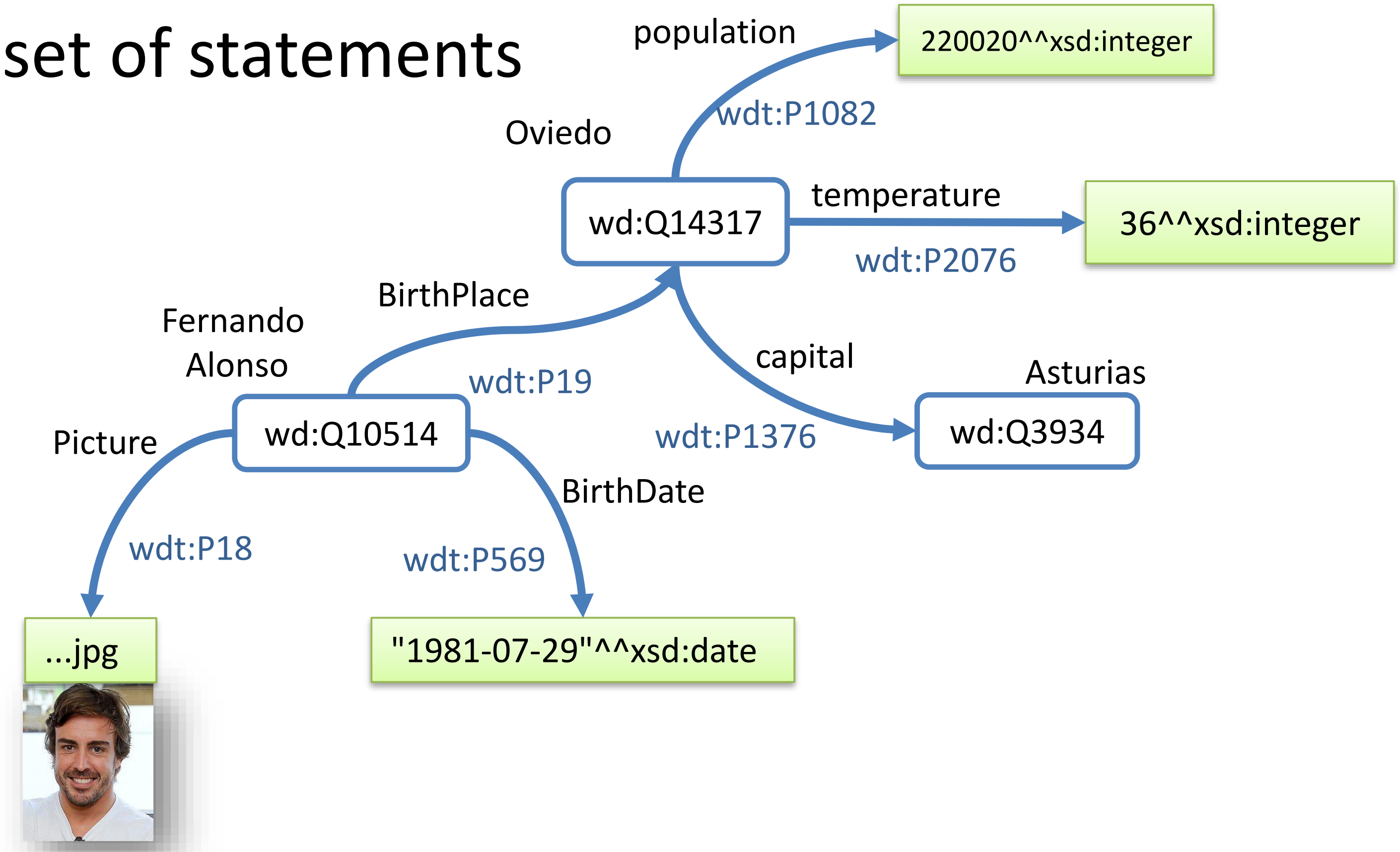
Web information & knowledge graphs



Knowledge graphs are a key part of the Web nowadays

RDF

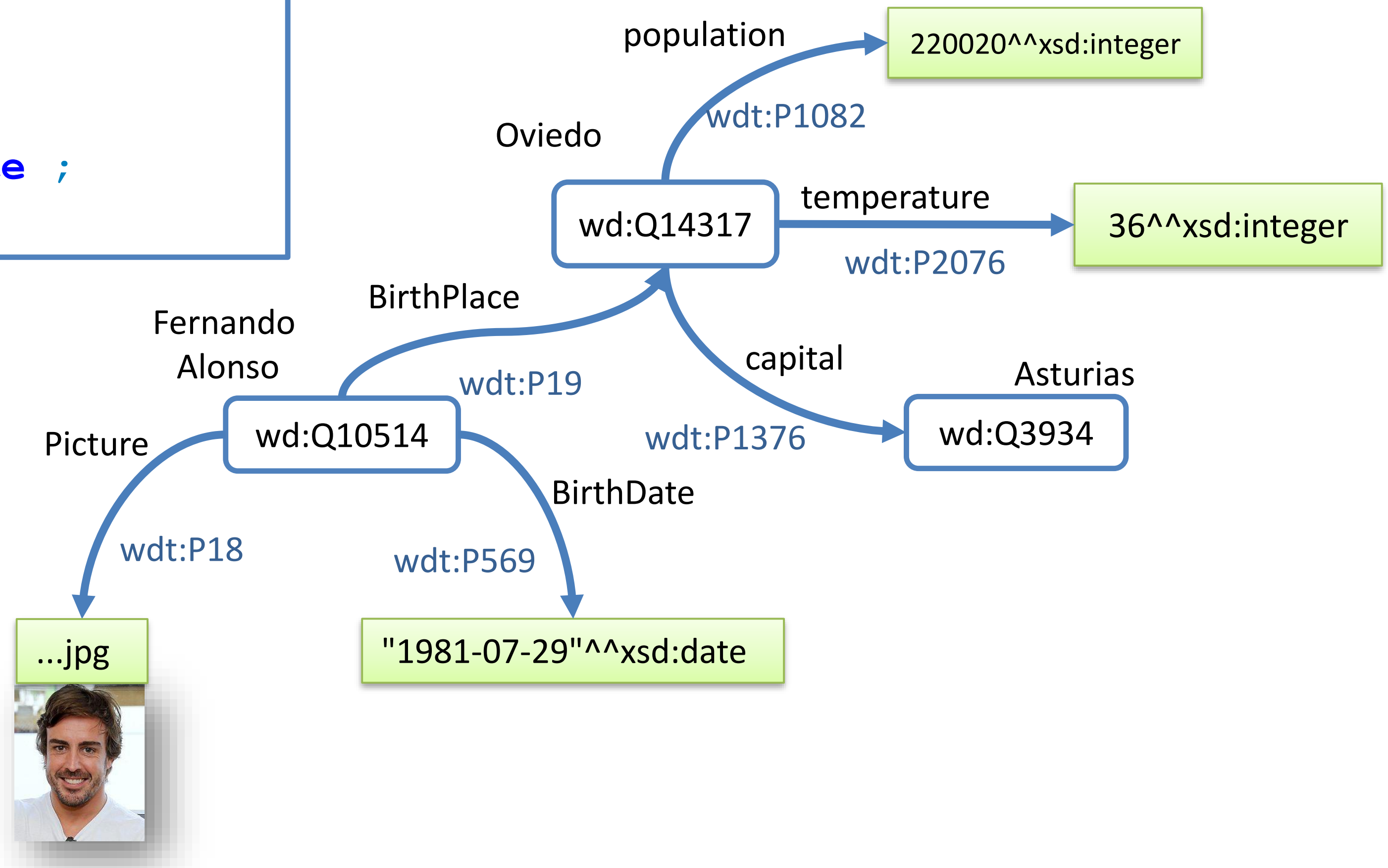
- RDF (Resource Description Framework)
  - Allows to represent knowledge graphs
  - RDF data model = graph as a set of statements
  - Predicates = URIs
- Several syntaxes
  - Turtle, N-Triples, JSON-LD,...



RDF syntaxes: Turtle

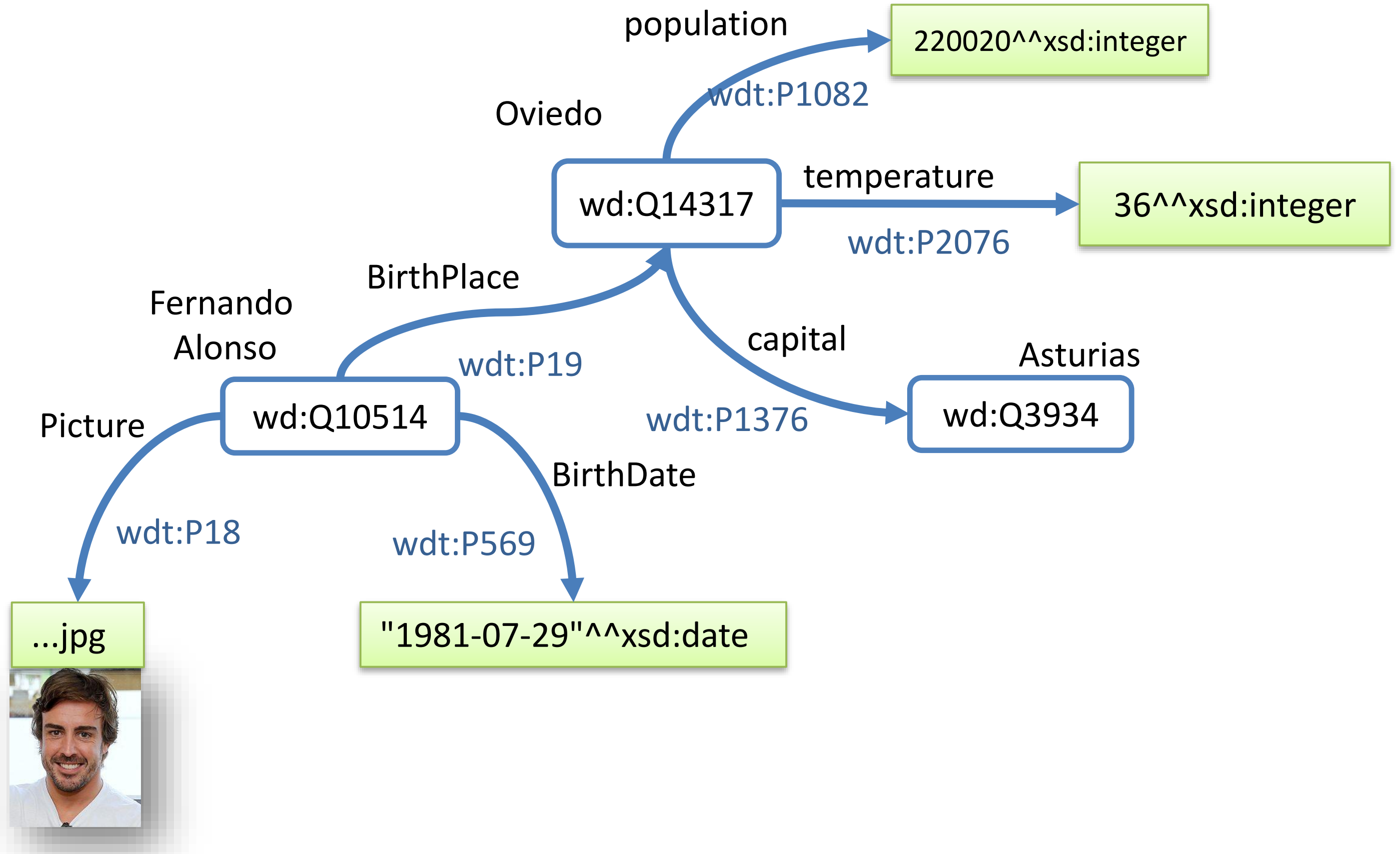
```
prefix wd:  <http://www.wikidata.org/entity/>
prefix wdt: <https://www.wikidata.org/prop/direct/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

wd:Q14317 wdt:P1082 220020 ;
          wdt:P2076 36 ;
          wdt:P1376 wd:Q3934 .
wd:Q10514 wdt:P19   wd:Q14317 ;
          wdt:P569  "1981-07-29"^^xsd:date ;
          wdt:P18   <picture.jpg> .
```



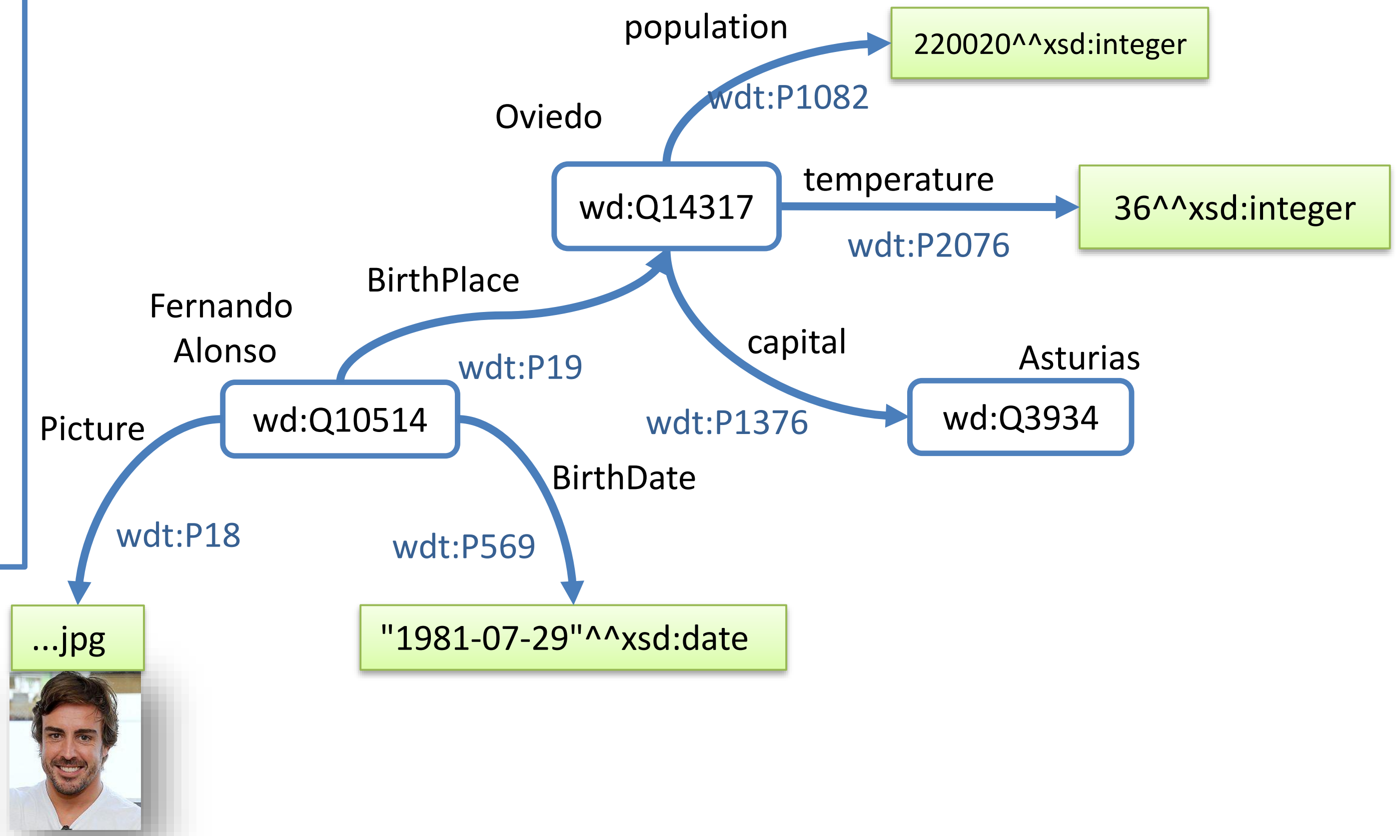
RDF syntaxes: N-Triples

```
<http://www.wikidata.org/entity/Q14317> <https://www.wikidata.org/prop/direct/P1082> "220020"^^<http://www.w3.org/2001/XMLSchema#integer> .
<http://www.wikidata.org/entity/Q14317> <https://www.wikidata.org/prop/direct/P1376> <http://www.wikidata.org/entity/Q3934> .
<http://www.wikidata.org/entity/Q14317> <https://www.wikidata.org/prop/direct/P2076> "36"^^<http://www.w3.org/2001/XMLSchema#integer> .
<http://www.wikidata.org/entity/Q10514> <https://www.wikidata.org/prop/direct/P18>    <picture.jpg> .
<http://www.wikidata.org/entity/Q10514> <https://www.wikidata.org/prop/direct/P19>    <http://www.wikidata.org/entity/Q14317> .
<http://www.wikidata.org/entity/Q10514> <https://www.wikidata.org/prop/direct/P569>  "1981-07-29"^^<http://www.w3.org/2001/XMLSchema#date> .
```



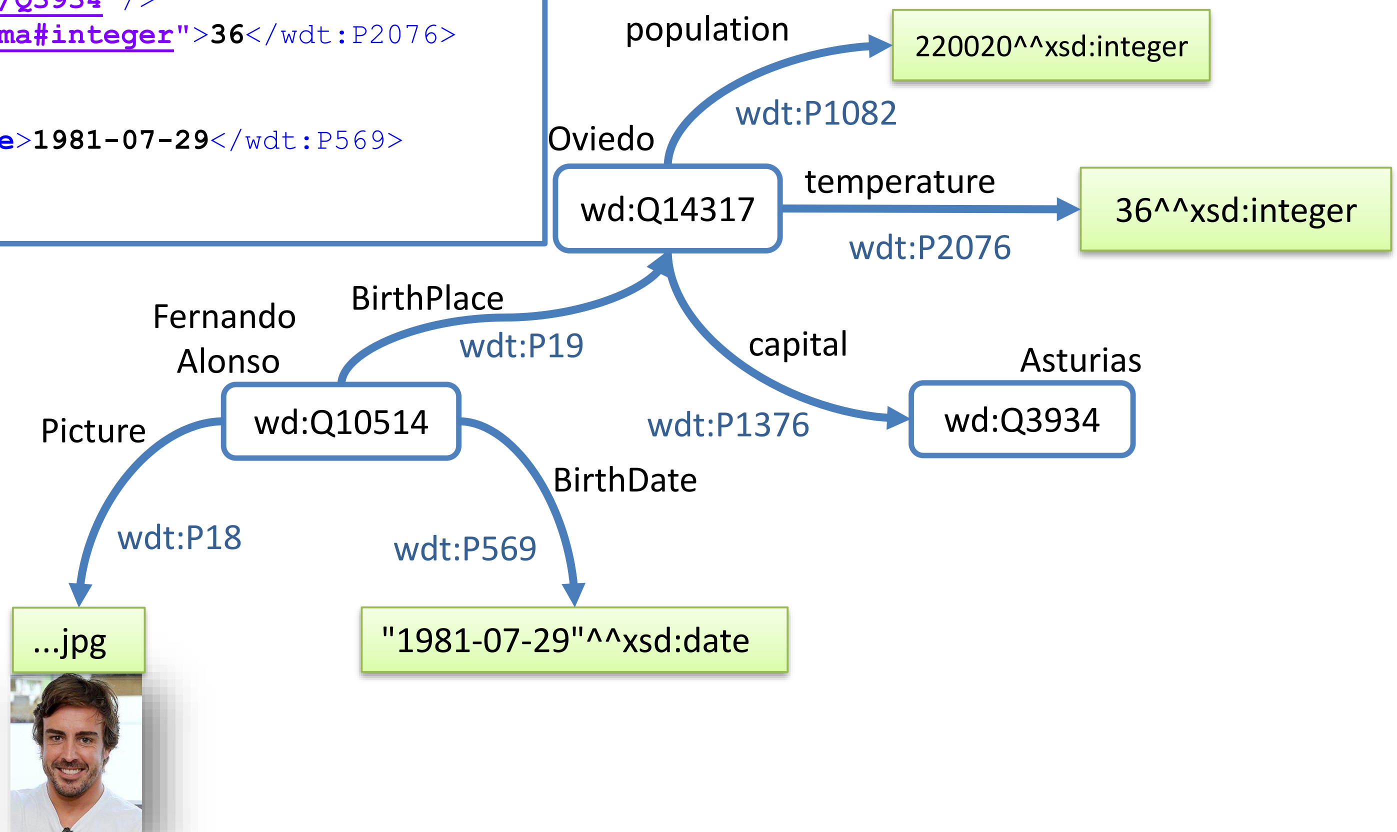
RDF syntaxes: JSON-LD

```
{
  "@context" : "http://...wikidata.json",
  "@graph" : [ {
    "@id"          : "wd:Q10514",
    "wdt:P18"      : "picture.jpg",
    "wdt:P19"      : "wd:Q14317",
    "P569"         : "1981-07-29"
  }, {
    "@id"          : "wd:Q14317",
    "wdt:P1082" : 220020,
    "wdt:P1376" : "wd:Q3934",
    "wdt:P2076" : 36
  }
 ]
}
```

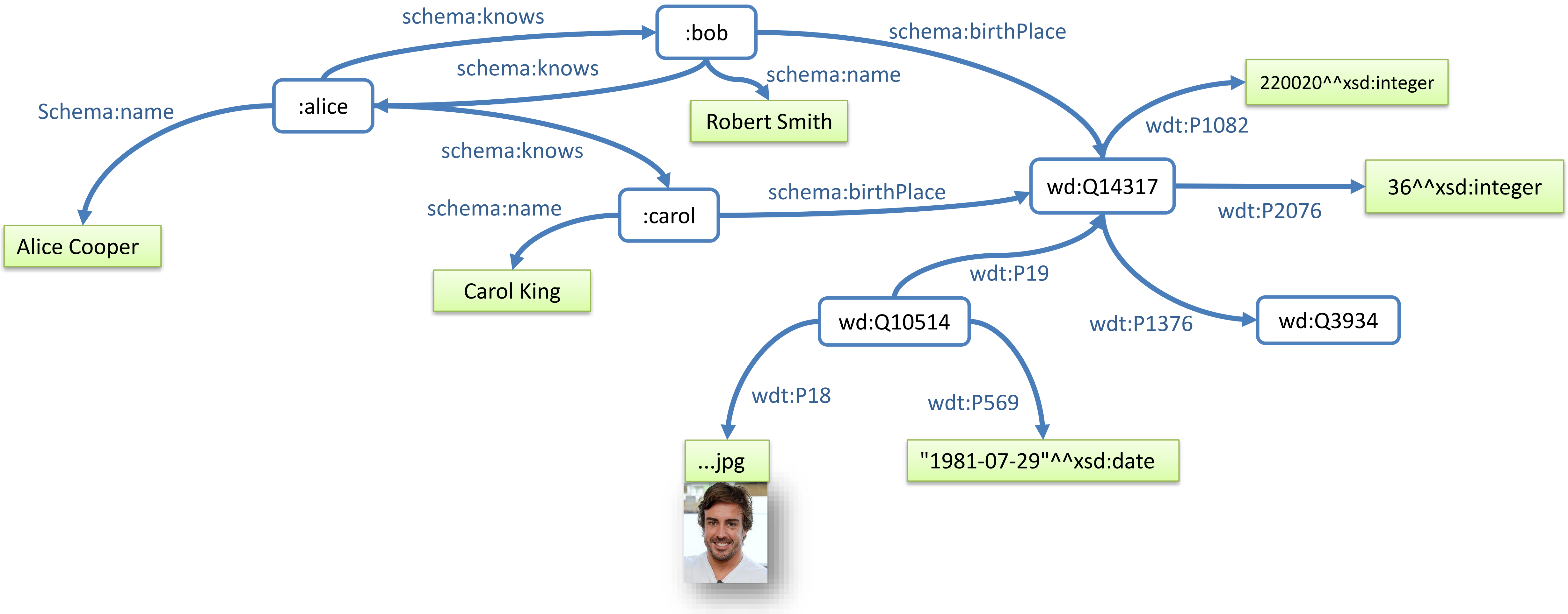


RDF syntaxes: RDF/XML

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:wdt="https://www.wikidata.org/prop/direct/"
 xmlns:wd="http://www.wikidata.org/entity/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
 <rdf:Description rdf:about="http://www.wikidata.org/entity/Q10514">
  <wdt:P18>picture.jpg</wdt:P18>
  <wdt:P19>
    <rdf:Description rdf:about="http://www.wikidata.org/entity/Q14317">
     <wdt:P1082 rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">220020</wdt:P1082>
        <wdt:P1376 rdf:resource="http://www.wikidata.org/entity/Q3934"/>
        <wdt:P2076 rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">36</wdt:P2076>
      </rdf:Description>
    </wdt:P19>
    <wdt:P569 rdf:datatype=http://www.w3.org/2001/XMLSchema#date>1981-07-29</wdt:P569>
  </rdf:Description>
</rdf:RDF>
```



RDF graphs are composable



RDF helps information integration

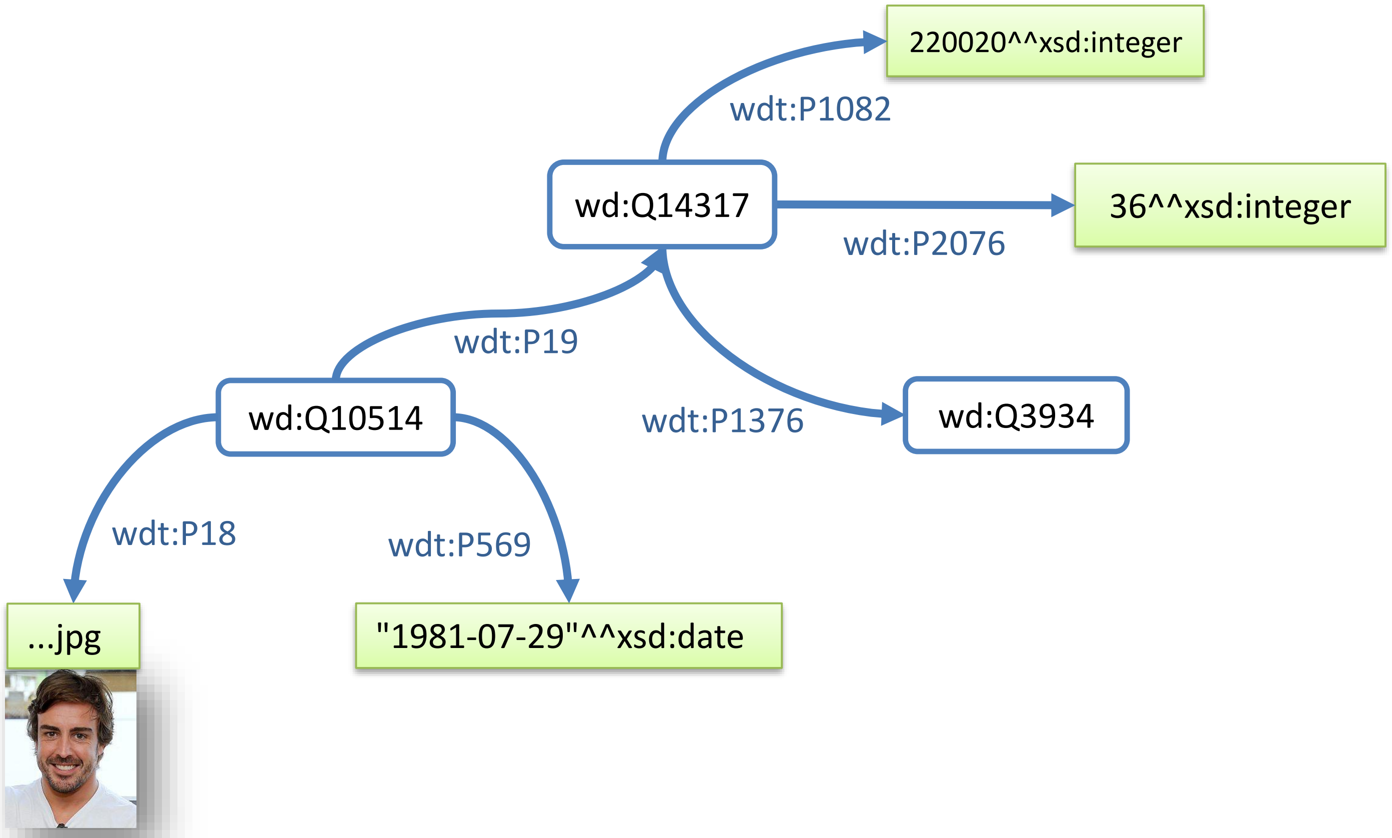
RDF data model

Graph = set of statements (triples)

Predicates (arcs) = URIs

Subjects: URIs*

Objects: URIs or literals*



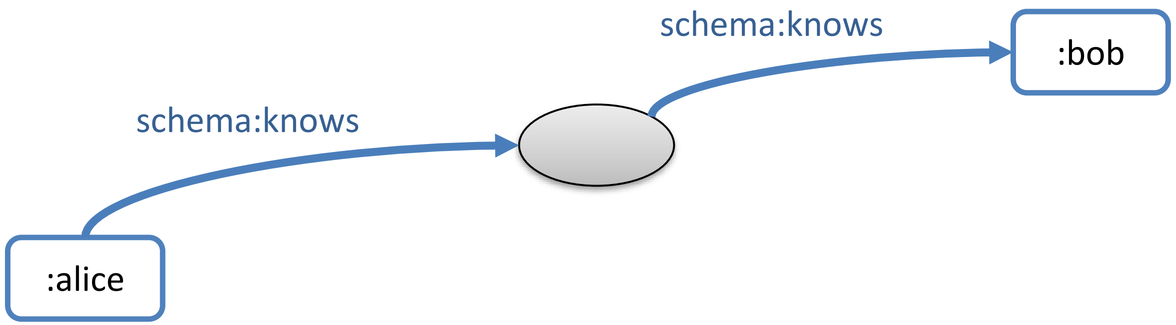
*Exception: blank nodes (next slide)

RDF data model: Blank nodes

A statement about something

Example: "Alice knows someone who knows Bob"

$$\exists x\, (\, \texttt{:alice}\ \ \texttt{:knows}\ \ x \quad \wedge \quad x\ \ \texttt{:knows}\ \ \texttt{:bob}\,)$$



Notation (in Turtle)

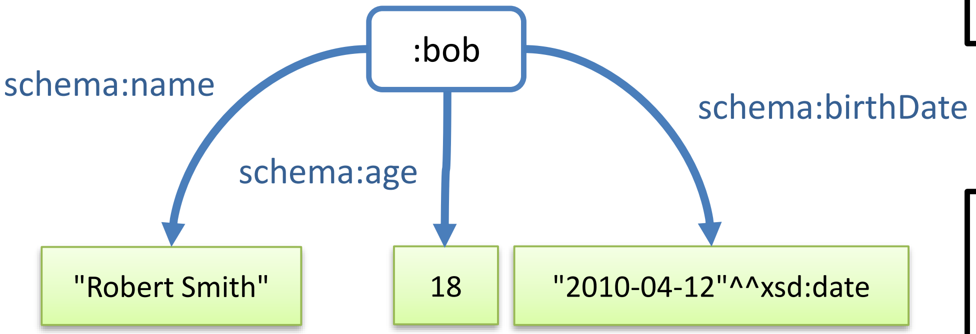
```
:alice   schema:knows _:x    ;
_:x      schema:knows :bob   .
```

```
:alice   schema:knows [
  schema:knows  :bob
] .
```

RDF data model: Literals

Objects can also be literals

- Literals contain a lexical form and a datatype
- Common datatypes: XML Schema primitive datatypes
- If not specified, a literal has type xsd:string



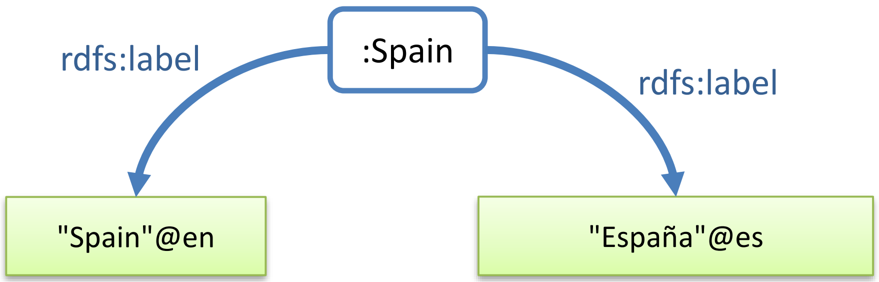

```
:bob schema:name       "Robert"  ;
     schema:age        18        ;
     schema:birthDate  "2010-04-12"^^xsd:date .
```

```
:bob schema:name       "Robert"^^xsd:string   ;
     schema:age        "18"^^xsd:integer      ;
     schema:birthDate  "2010-04-12"^^xsd:date .
```

RDF data model: Language tagged strings

String literals can be qualified by a language tag

- They have datatype `rdfs:langString`




```
:spain rdfs:label "Spain"@en ;
       rdfs:label "España"@es .
```

...and that's all

Yes, the RDF Data model is simple

Simple
is
better

RDF ecosystem

SPARQL

Vocabularies

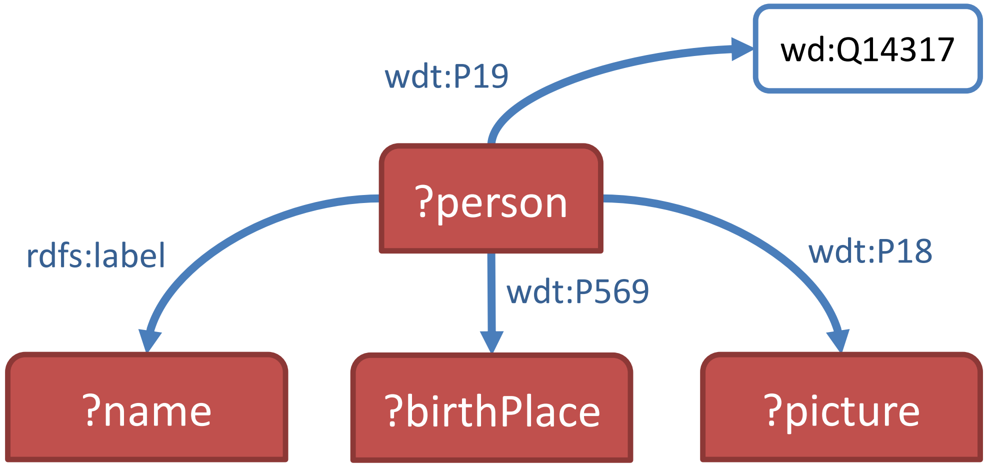
Ontologies and inference

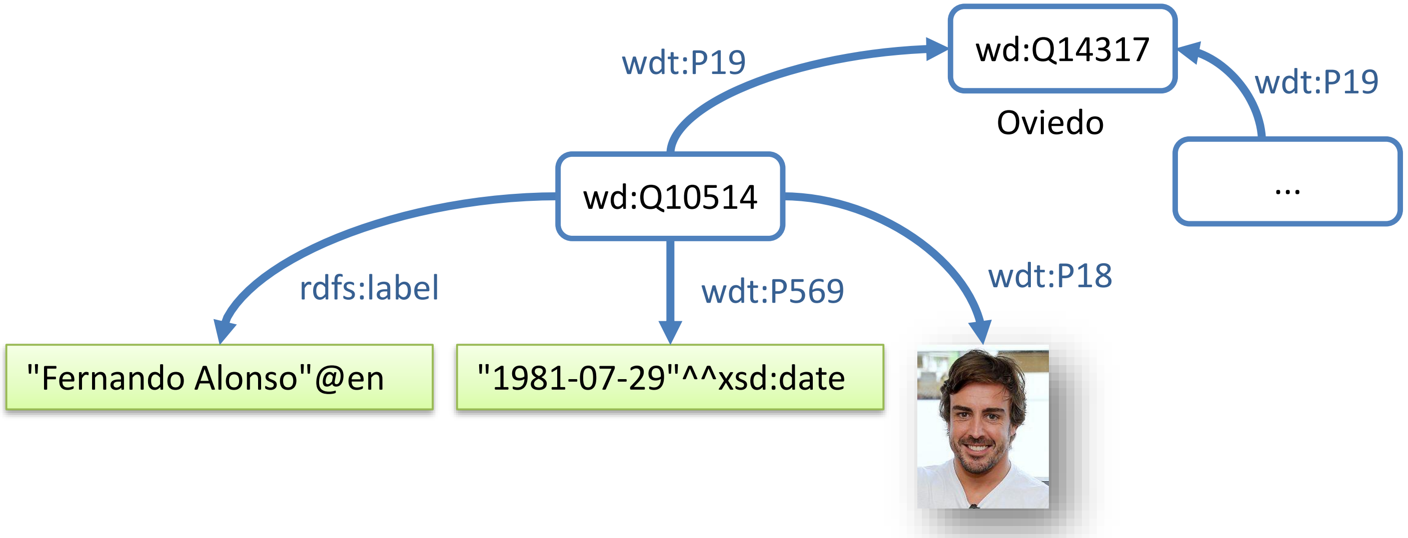
SPARQL

SPARQL = Simple Protocol and RDF Query Language

- Similar to SQL, but for RDF
- Example: People born in Oviedo

```
SELECT ?name ?birthDate ?picture WHERE {
  ?person wdt:P19     wd:Q14317 ;
          rdfs:label ?name ;
          wdt:P569   ?birthDate ;
          wdt:P18    ?picture .
  FILTER (Lang(?name) = 'es')
}
```





Shared entities and vocabularies

URIs as global identifiers allow to reuse them

Lots of vocabularies: domain specific or general purpose

Examples:

schema.org: properties that major browsers can recognize

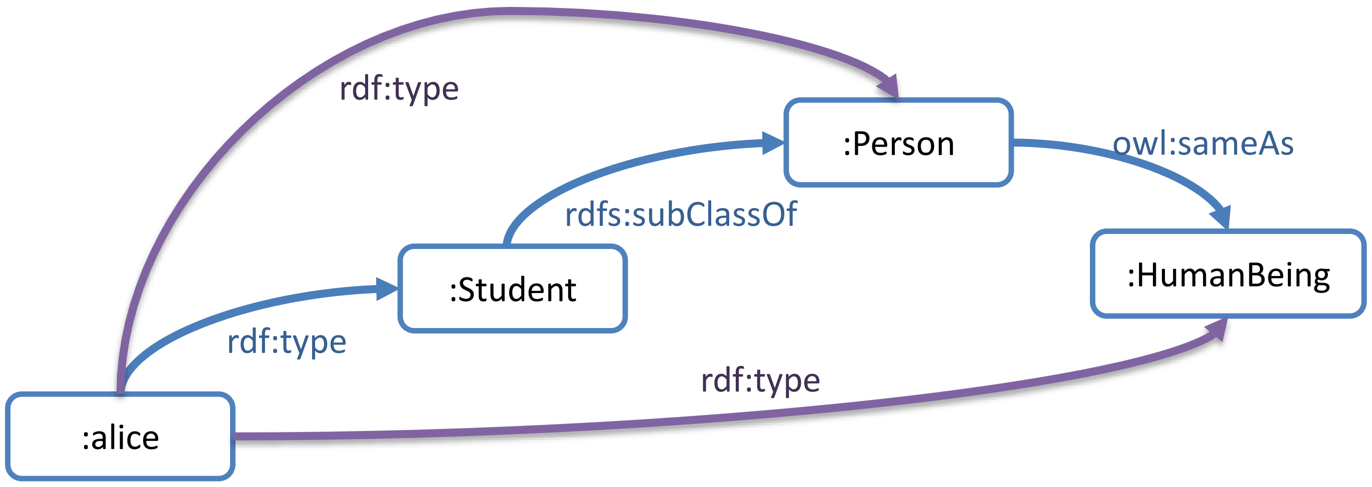
Linked Open Vocabularies

Inference and ontologies

Some vocabularies define properties that can infer new information

- RDF Schema: Classes, subclasses, etc.
- OWL: Ontologies using description logics

Trade-off between expressiveness and complexity



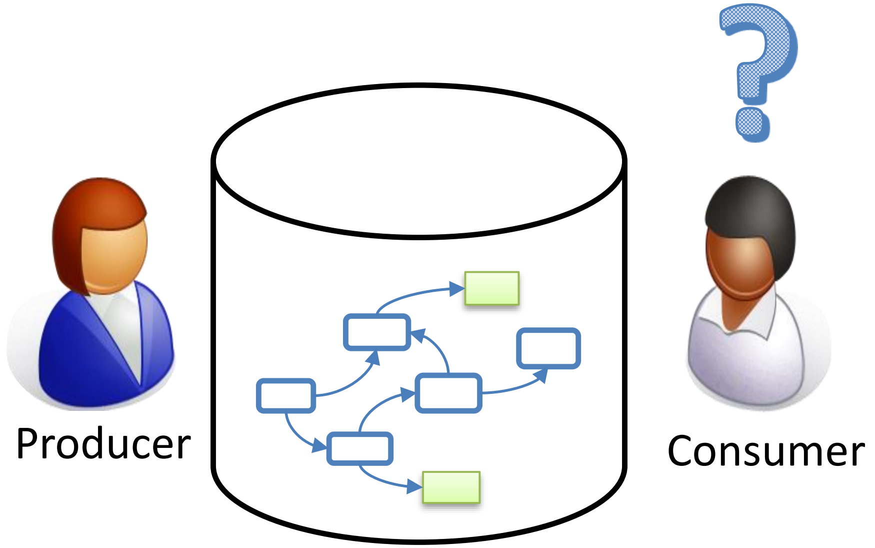
RDF, the good parts...

- RDF as an integration language
- RDF as a *lingua franca* for semantic web and linked data
- RDF flexibility & integration
  - Data can be adapted to multiple environments
  - Open and reusable data by default
- RDF for knowledge representation
- RDF data stores & SPARQL

RDF, the other parts

Consuming & producing RDF

- Multiple syntaxes: Turtle, RDF/XML, JSON-LD, ...
- Embedding RDF in HTML
- Describing and validating RDF content



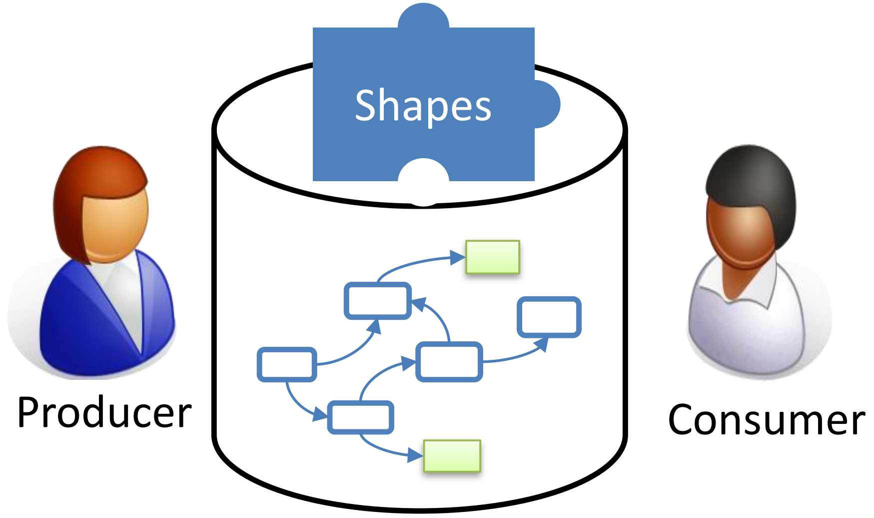
Why describe & validate RDF?

For producers

- Developers can understand the contents they are going to produce
- 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



Similar technologies

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

Fill that gap

Understanding the problem

Identifying the shape of graphs...

- Shapes can describe the form of a node (node constraint)
- ...and the number of possible arcs incoming/outgoing from a node
- ...and the possible values associated with those arcs

RDF Node

```
:alice schema:name   "Alice";
       schema:knows :bob, :carol .
```

```
IRI schema:name   string  1
    schema:knows IRI     0, 1,...
```

Shape
RDF Node that
represents a User

ShEx

```
<UserShape> IRI {
  schema:name  xsd:string   ;
  schema:knows IRI         *
}
```

Understanding the problem

Repeated properties

- The same property can be used for different purposes in the same data
- Example: A product must have 2 codes with different structure

```
:product schema:productID "isbn:123-456-789";
         schema:productID "code456" .
```

A practical example from FHIR

- See: http://hl7-fhir.github.io/observation-example-bloodpressure.ttl.html

Understanding the problem

Shapes ≠ types

Nodes in RDF graphs can have 0, 1 or many `rdf:type` declarations

- A type can be used in multiple contexts, e.g. `foaf:Person`

Nodes are not necessarily annotated with discriminating types

- Nodes with type `:Person` can represent friends, students, patients,...
- Different meanings and different structure depending on context
- Specific validation constraints for different contexts

Understanding the problem

RDF validation ≠ ontology definition ≠ instance data

- Ontologies are usually focused on domain entities
- RDF validation is focused on RDF graph features (lower level)

Different levels

Ontology

```
schema:knows a owl:ObjectProperty ;
 rdfs:domain schema:Person ;
 rdfs:range  schema:Person .
```

Constraints RDF *Validation*

A user must have only two properties:
schema:name of value xsd:string
schema:knows with an IRI value

```
<User> IRI {
  schema:name  xsd:string ;
  schema:knows IRI
}
```

Instance data

```
:alice schema:name "Alice";
       schema:knows :bob .
```

Why not using SPARQL to validate?

Pros:

- Expressive
- Ubiquitous

Cons

- Expressive
- Idiomatic
  - many ways to encode the same constraint

Example: Define SPARQL query to check:
There must be one `schema:name` which must be a `xsd:string`, and one `schema:gender` which must be `schema:Male` or `schema:Female`

```
ASK {{ SELECT ?Person {
         ?Person schema:name ?o .
       } GROUP BY ?Person HAVING (COUNT(*)=1)
     }
     { SELECT ?Person {
         ?Person schema:name ?o .
         FILTER ( isLiteral(?o) &&
                  datatype(?o) = xsd:string )
        } GROUP BY ?Person HAVING (COUNT(*)=1)
     }
     { SELECT ?Person (COUNT(*) AS ?c1) {
         ?Person schema:gender ?o .
       } GROUP BY ?Person HAVING (COUNT(*)=1)}
       { SELECT ?Person (COUNT(*) AS ?c2) {
         ?S schema:gender ?o .
         FILTER ((?o = schema:Female ||
                  ?o = schema:Male))
       } GROUP BY ?Person HAVING (COUNT(*)=1)}
     FILTER (?c1 = ?c2)
}
```

Previous/other approaches

SPIN, by TopQuadrant http://spinrdf.org/

SPARQL templates, Influenced SHACL

Stardog ICV: http://docs.stardog.com/icv/icv-specification.html

OWL with UNA and CWA

OSLC resource shapes: https://www.w3.org/Submission/shapes/

Vocabulary for RDF validation

Dublin Core Application profiles (K. Coyle, T. Baker)

http://dublincore.org/documents/dc-dsp/

RDF Data Descriptions (Fischer et al)

http://ceur-ws.org/Vol-1330/paper-33.pdf

RDFUnit (D. Kontokostas)

http://aksw.org/Projects/RDFUnit.html

...

ShEx and 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 Community group draft
- 2018 ShEx adopted by Wikidata

Short intro to ShEx

ShEx (Shape Expressions Language)

Concise and human-readable language for RDF validation & description

Syntax similar to SPARQL, Turtle

Semantics inspired by regular expressions & RelaxNG

2 syntaxes: Compact and RDF/JSON-LD

Official info: http://shex.io

Semantics: http://shex.io/shex-semantics/, primer: http://shex.io/shex-primer

ShEx implementations and playgrounds

Libraries:

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

Online demos & playgrounds

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

Simple example

Prefix declarations as in Turtle/SPARQL

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

<User> IRI {
 schema:name  xsd:string   ;
 schema:knows @<User>     *
}
```

Nodes conforming to <User> shape must:

- Be IRIs
- Have exactly one schema:name with a value of type xsd:string
- Have zero or more schema:knows whose values conform to <User>

RDF Validation using ShEx

Schema

```
<User> IRI {
 schema:name   xsd:string    ;
 schema:knows @<User>      *
}
```

Shape map

```
:alice@<User>
:bob   @<User>
:carol@<User>
:dave  @<User>
:emily@<User>
:frank@<User>
:grace@<User>
```

Try it (RDFShape): https://goo.gl/97bYdv
Try it (ShExDemo):https://goo.gl/Y8hBsW

Data

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

:bob   schema:knows  :alice ;
       schema:name   "Robert".

:carol schema:name    "Carol", "Carole" .

:dave  schema:name    234 .

:emily foaf:name      "Emily" .

:frank schema:name "Frank" ;
       schema:email <mailto:frank@example.org> ;
       schema:knows :alice, :bob .

:grace schema:name "Grace" ;
       schema:knows :alice, _:1 .

_:1 schema:name   "Unknown" .
```

Validation process

Input: RDF data, ShEx schema, Shape map

Output: Result shape map

ShEx Schema

```
:User {
 schema:name  xsd:string ;
 schema:knows @:User *
}
```

Shape map

```
:alice@:User, :bob@:User, :carol@:User
```

RDF data

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

:bob   schema:knows :alice ;
       schema:name  "Robert".

:carol schema:name   "Carol", "Carole" .
```

ShEx Validator

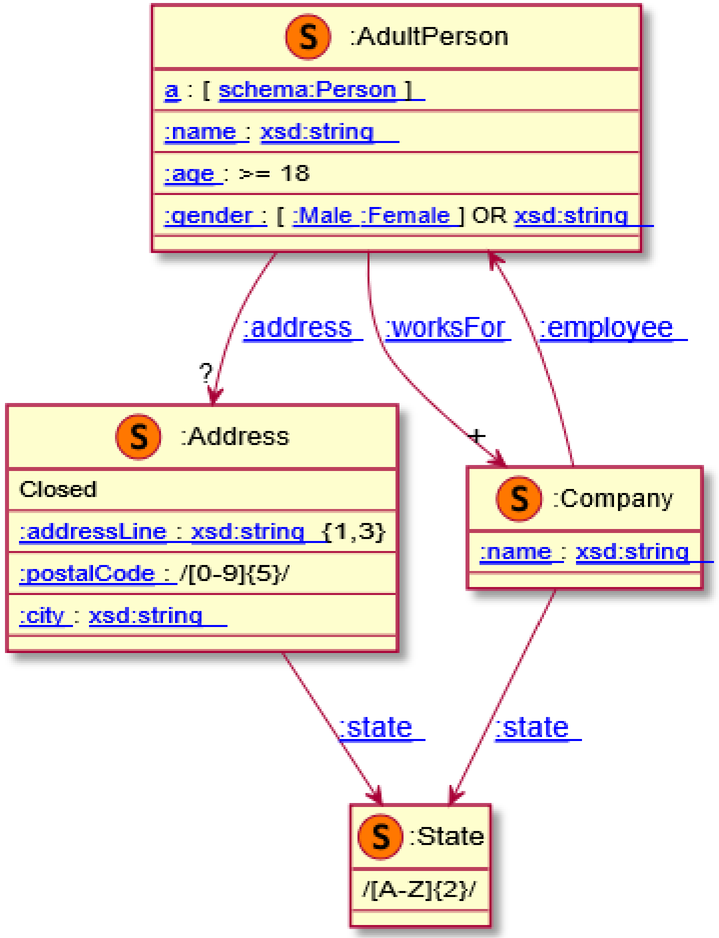
Result shape map

```
:alice@:User,
:bob@:User,
:carol@!:User
```

Example with more ShEx features

```
:AdultPerson EXTRA rdf:type {
 rdf:type   [ schema:Person ]              ;
 :name      xsd:string                     ;
 :age       MinInclusive 18                ;
 :gender    [:Male :Female] OR xsd:string ;
 :address   @:Address ?                    ;
 :worksFor  @:Company +
}
:Address CLOSED {
 :addressLine xsd:string {1,3}
 :postalCode  /[0-9]{5}/
 :state        @:State
 :city         xsd:string
}
:Company {
 :name     xsd:string
 :state    @:State
 :employee @:AdultPerson *
}
:State   /[A-Z]{2}/
```

```
:alice rdf:type :Student, schema:Person ;
 :name       "Alice" ;
 :age        20 ;
 :gender     :Male ;
 :address    [
  :addressLine  "Bancroft Way" ;
  :city         "Berkeley" ;
  :postalCode   "55123" ;
  :state        "CA"
 ] ;
 :worksFor  [
  :name       "Company" ;
  :state      "CA"      ;
  :employee   :alice
] .
```



Try it: https://tinyurl.com/yd5hp9z4

More info about ShEx

See:

ShEx by Example (slides):

https://figshare.com/articles/ShExByExample_pptx/6291464

ShEx chapter from Validating RDF data book:

http://book.validatingrdf.com/bookHtml010.html

Short intro to SHACL

W3C recommendation:

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

RDF vocabulary

2 parts: SHACL-Core, SHACL-SPARQL

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/
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

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://goo.gl/T1uuzv

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" .          ☹
```

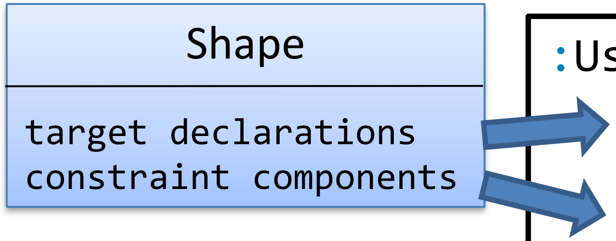
Data graph

Try it. RDFShape https://goo.gl/ukY5vq

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




```
: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 .
. . .
```

Validation Report

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

No errors $\Rightarrow$ 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

```
: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> .
```

SHACL Processor

Validation report

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

SHACL Core built-in 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`

Longer example

In ShEx

```
:AdultPerson EXTRA a {
 a           [ schema:Person ]               ;
 :name       xsd:string                      ;
 :age        MinInclusive 18                 ;
 :gender     [:Male :Female] OR xsd:string   ;
 :address    @:Address ?                     ;
 :worksFor   @:Company +                     ;
}
:Address CLOSED {
 :addressLine xsd:string {1,3}               ;
 :postalCode  /[0-9]{5}/                     ;
 :state       @:State                        ;
 :city        xsd:string
}
:Company {
 :name     xsd:string                        ;
 :state    @:State                           ;
 :employee @:AdultPerson *                   ;
}
:State   /[A-Z]{2}/
```

Try it: https://tinyurl.com/ycl3mkzr

In SHACL

```
:AdultPerson a sh:NodeShape ;
  sh:property [
   sh:path rdf:type ;
   sh:qualifiedValueShape [
      sh:hasValue schema:Person
   ];
   sh:quali
   sh:quali
 ] ;
 sh:targetN
   sh:prope
    sh:minC
    sh:data
  ] ;
  sh:proper
   sh:minCo
   sh:in (
  ] ;
  sh:proper
   sh:maxCo
   sh:minIr
  ] ;
 sh:propert
   sh:node :Address
   sh:minCount 1 ; sh
  ] ;
  sh:property [ sh:path :worksFor ;
    sh:node :Company ;
    sh:minCount 1 ; sh:maxCoun
  ].
```

```
:Address a sh:NodeShape ;
 sh:closed true ;
 sh:property [ sh:path :addressLine;
   sh:datatype xsd:string ;
   sh:min
 ] ;
 sh:prope
   sh:pat
   sh:min
 ] ;
 sh:prope
   sh:dat
   sh:min
 ] ;
 sh:prope
   sh:nod
 ] .
```

```
:Company a sh:NodeShape ;
 sh:property [ sh:path :name ;
   sh:datatype xsd:string
   ] ;
 sh:property [
   sh:path :state ;
   sh:node :State
 ] ;
 sh:property [ sh:path :employee ;
   sh:node :AdultPerson ;
 ] ;.

:State a sh:NodeShape ;
   sh:pattern "[A-Z]{2}" .
```

Its recursive!!! (not well defined SHACL)
Implementation dependent feature

More info about SHACL

SHACL by example (slides):

https://figshare.com/articles/SHACL_by_example/6449645

SHACL chapter at Validating RDF data book

http://book.validatingrdf.com/bookHtml011.html

Some challenges and perspectives

Theoretical foundations of ShEx/SHACL

Inference shapes from data

Validation Usability

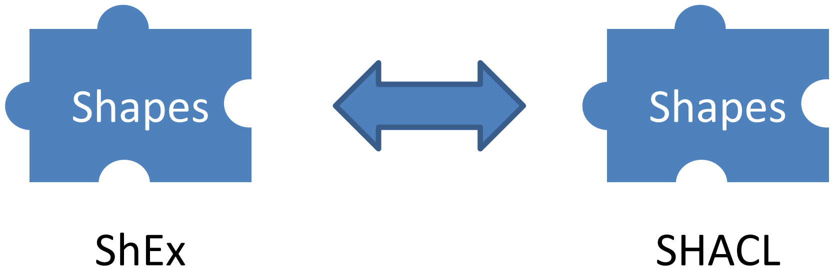
RDF Stream validation

Schema ecosystems

- Wikidata
- Solid

Theoretical foundations of ShEx/SHACL

- Conversion between ShEx and SHACL
  - SHaclEX library converts subsets of both
  - Challenges
    - Recursion and negation
    - Performance and algorithmic complexity
    - Detect useful subsets of the languages
  - Convert to SPARQL
- Schema/data mapping



Inference of Shapes from Data

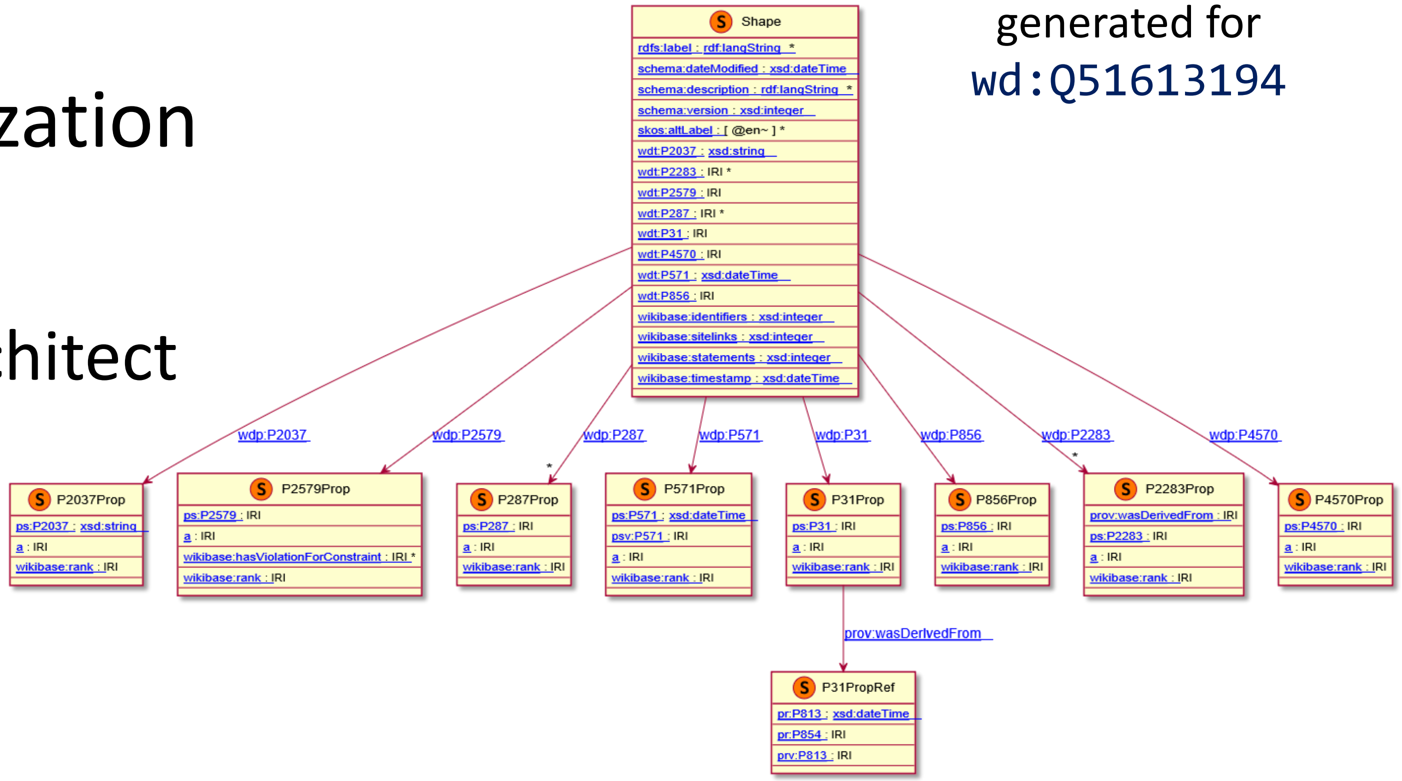
Useful use case in practice

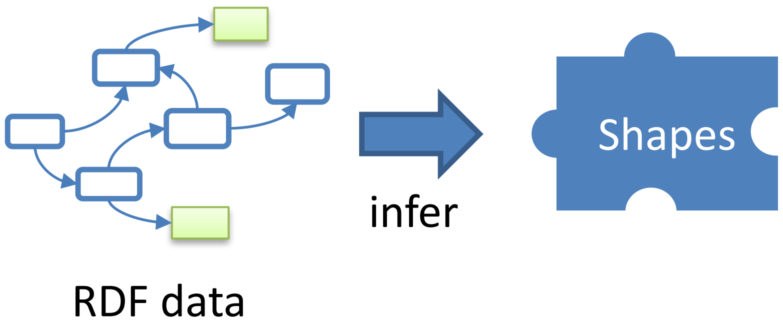
Knowledge Graph summarization

Some prototypes:

ShExer, RDFShape, ShapeArchitect

Shape Expression generated for `wd:Q51613194`

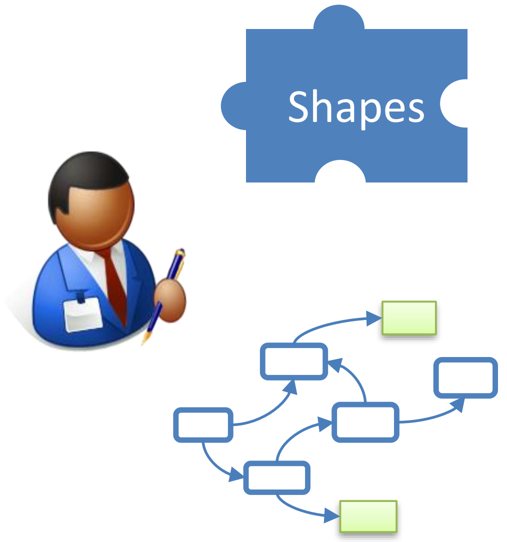




Try it with RDFShape:
https://tinyurl.com/y8pjcbyf

Validation usability

- Learning from users
  - Early adopters: WebIndex, HL7 FHIR, Eclipse Lyo, GenWiki,…
  - Improve error information/visualization/navigation/repairing
- Authoring/visualization tools
- Propose annotation sets
  - UI generation
  - Error reporting/suggestion (SHOULD/MUST/…)

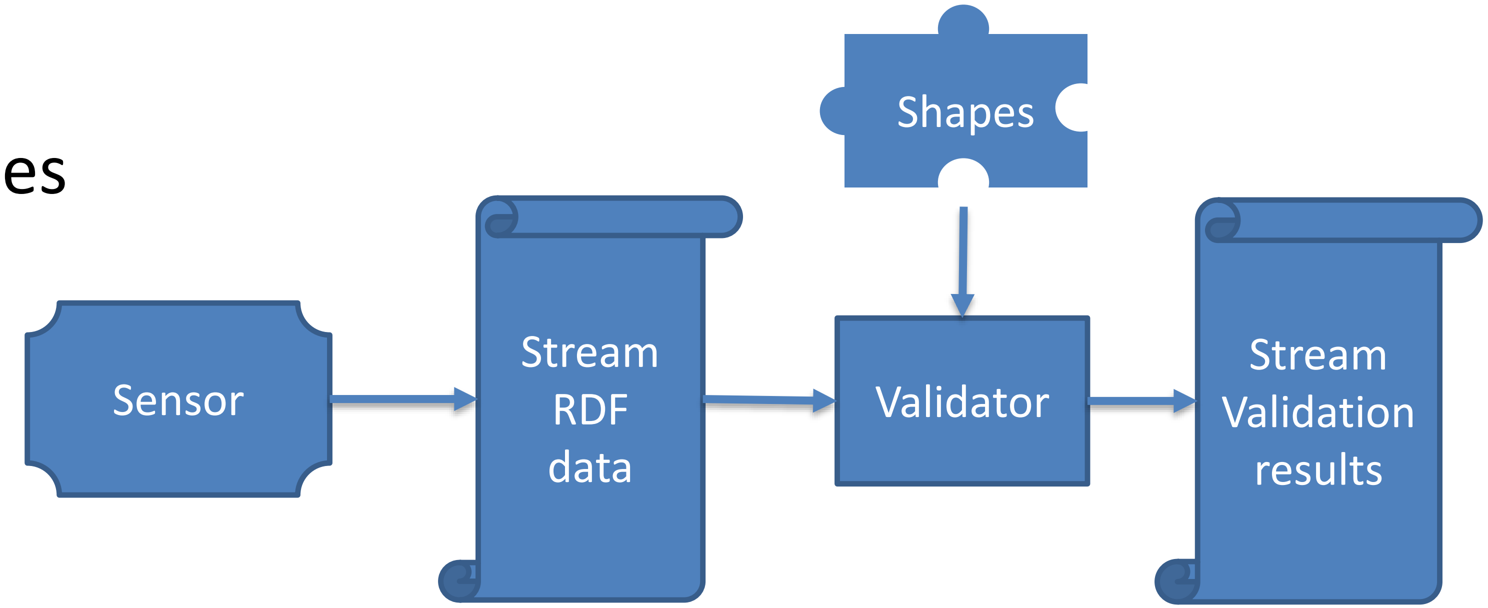


RDF Stream validation

Validation of RDF streams

Challenges:

- Incremental validation
- Named graphs
- Addition/removal of triples



Schema ecosystems: Wikidata

In May, 2019, Wikidata announced ShEx adoption

New namespace for schemas

- Example: https://www.wikidata.org/wiki/EntitySchema:E2

It opens lots of opportunities/challenges

- Schema evolution and comparison

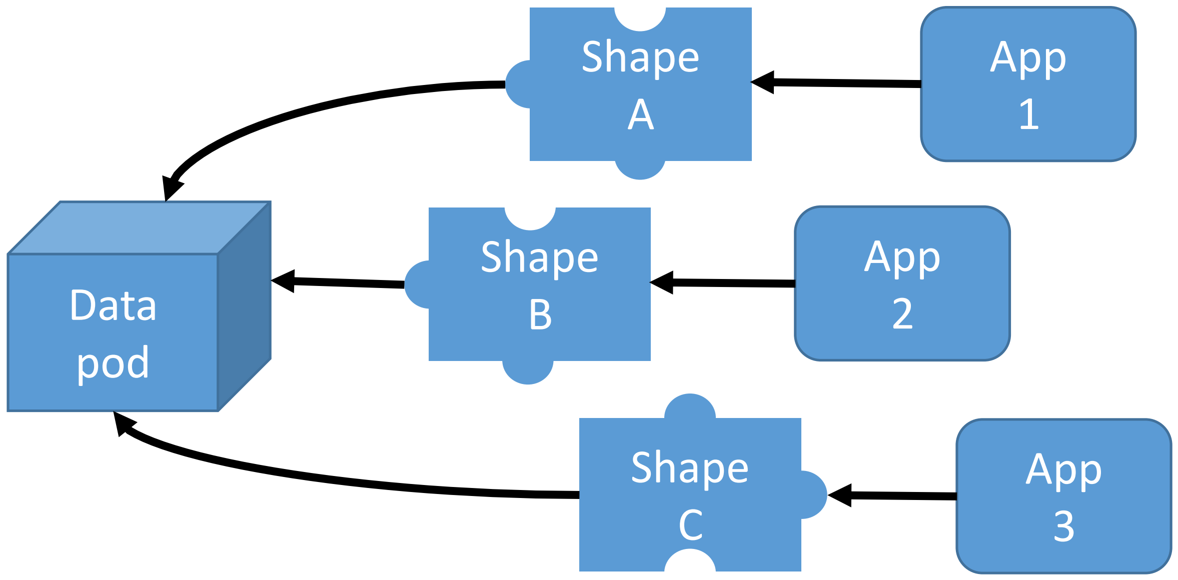
Schema ecosystems: Solid project

SOLID (SOcial Linked Data): Promoted by Tim Berners-Lee

Goal: Re-decentralize the Web

- Separate data from apps
- Give users more control about their data
- Internally using linked data & RDF

Shapes needed for interoperability



See:
https://ruben.verborgh.org/blog/2019/06/17/shaping-linked-data-apps/

Conclusions

RDF as a basis for knowledge graphs

Explicit schemas can help improve data quality

2 languages proposed: ShEx/SHACL

Lots of new challenges and opportunities

End of presentation