Metadata for RDF Statements: The RDF-star Approach

Olaf Hartig
(Linköping University)
@olafhartig

Pierre-Antoine Champin
(W3C/ERCIM)
@pchampin

LINKÖPING UNIVERSITY

W3C

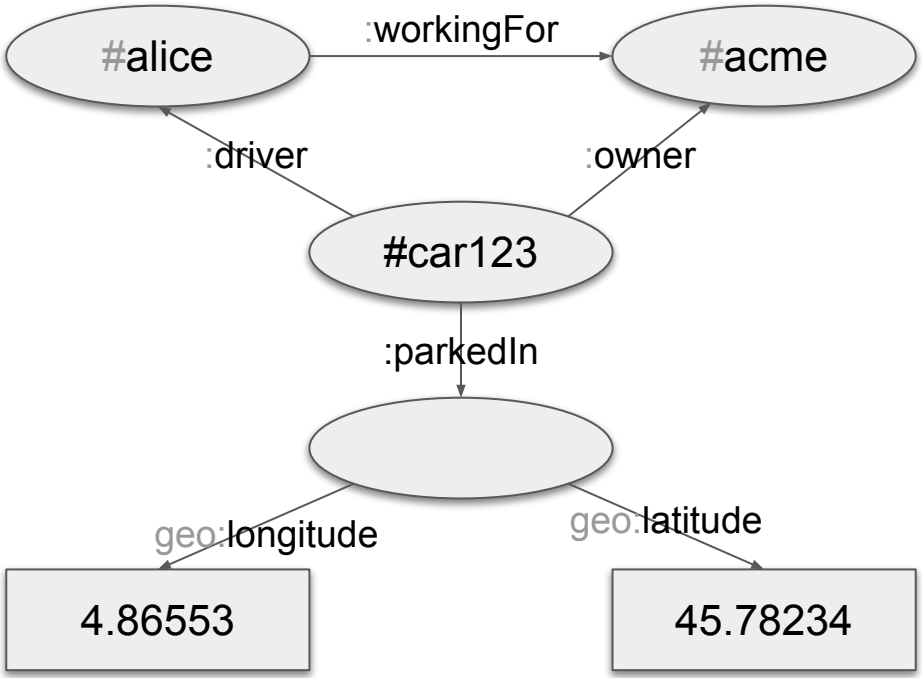
LOTICO - 2021-03-25

Part 1

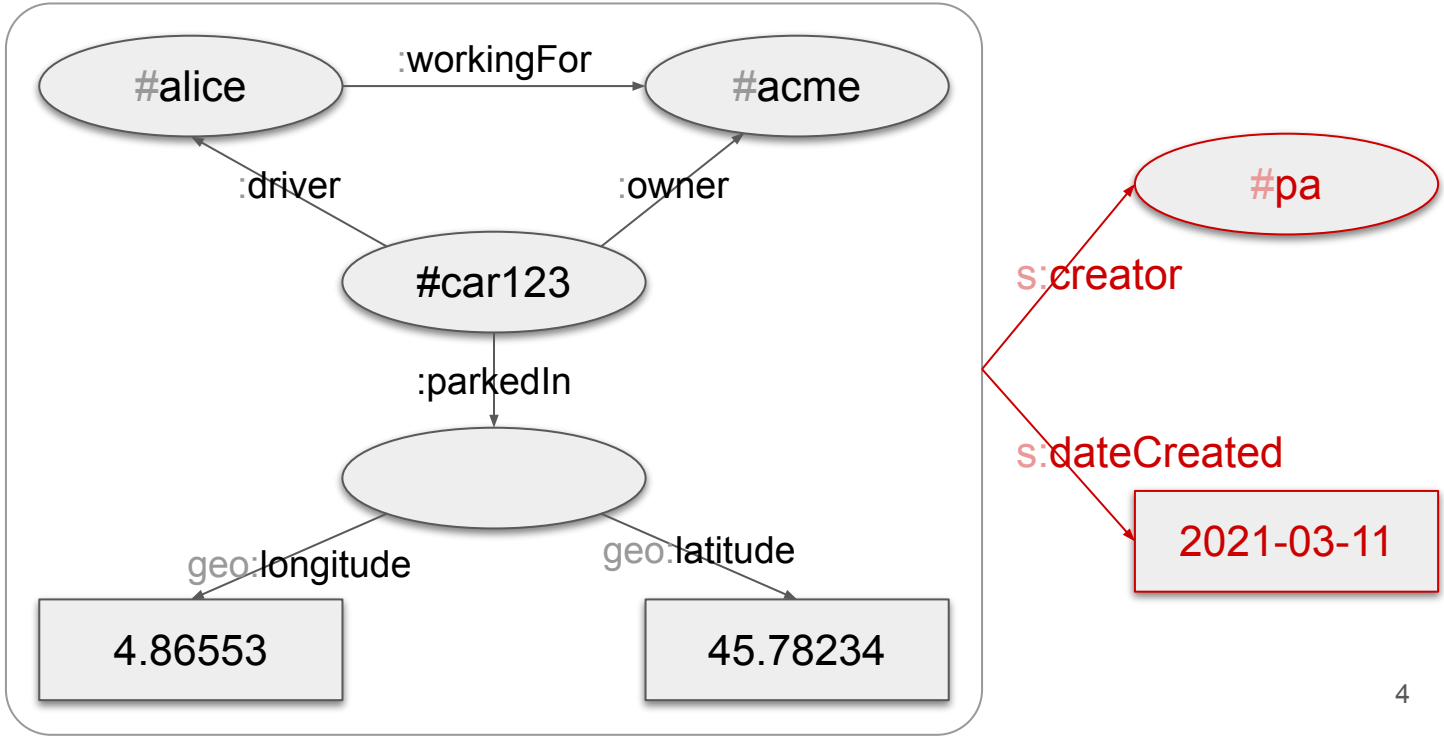
Motivation

RDF and Statement-level metadata: *it's complicated!*

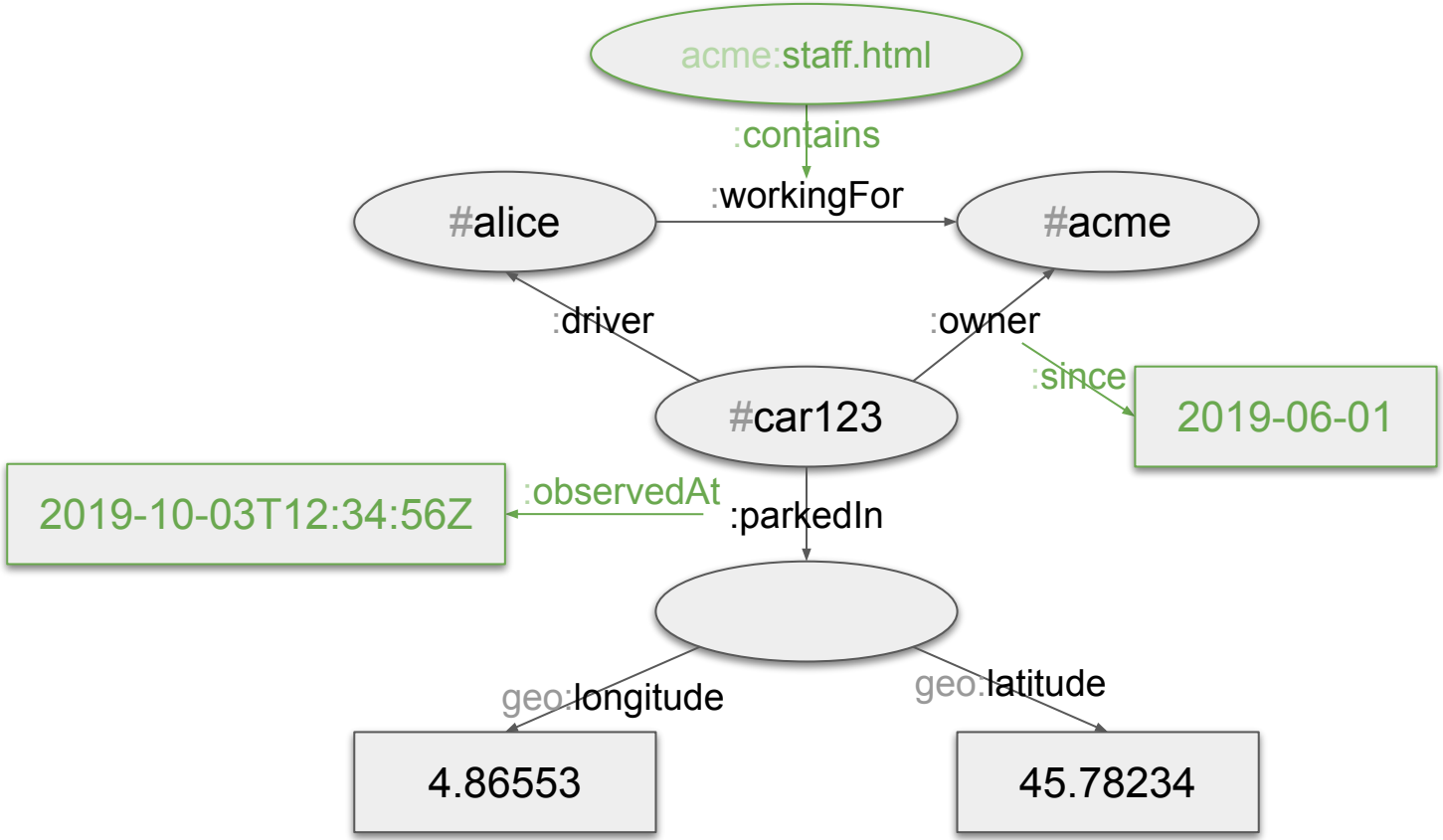
Example RDF data



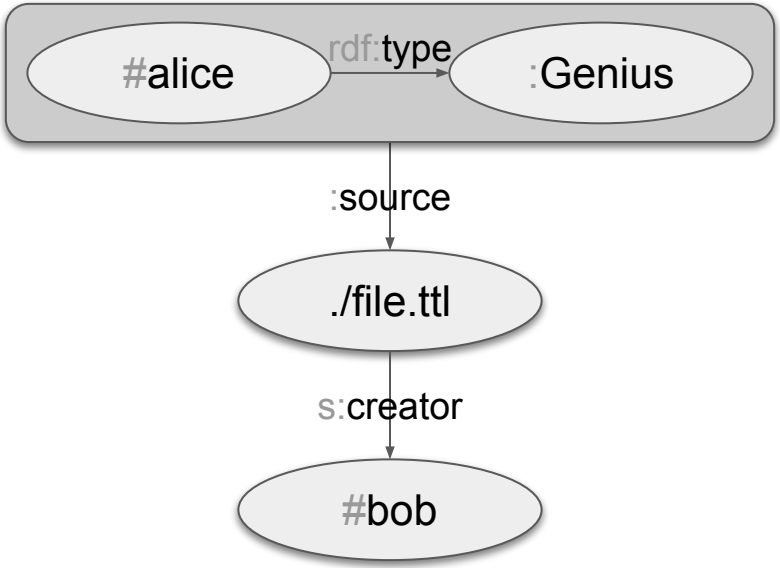
Example RDF data + graph-level metadata



Example RDF data + statement-level metadata



Example #2 RDF data + statement-level metadata



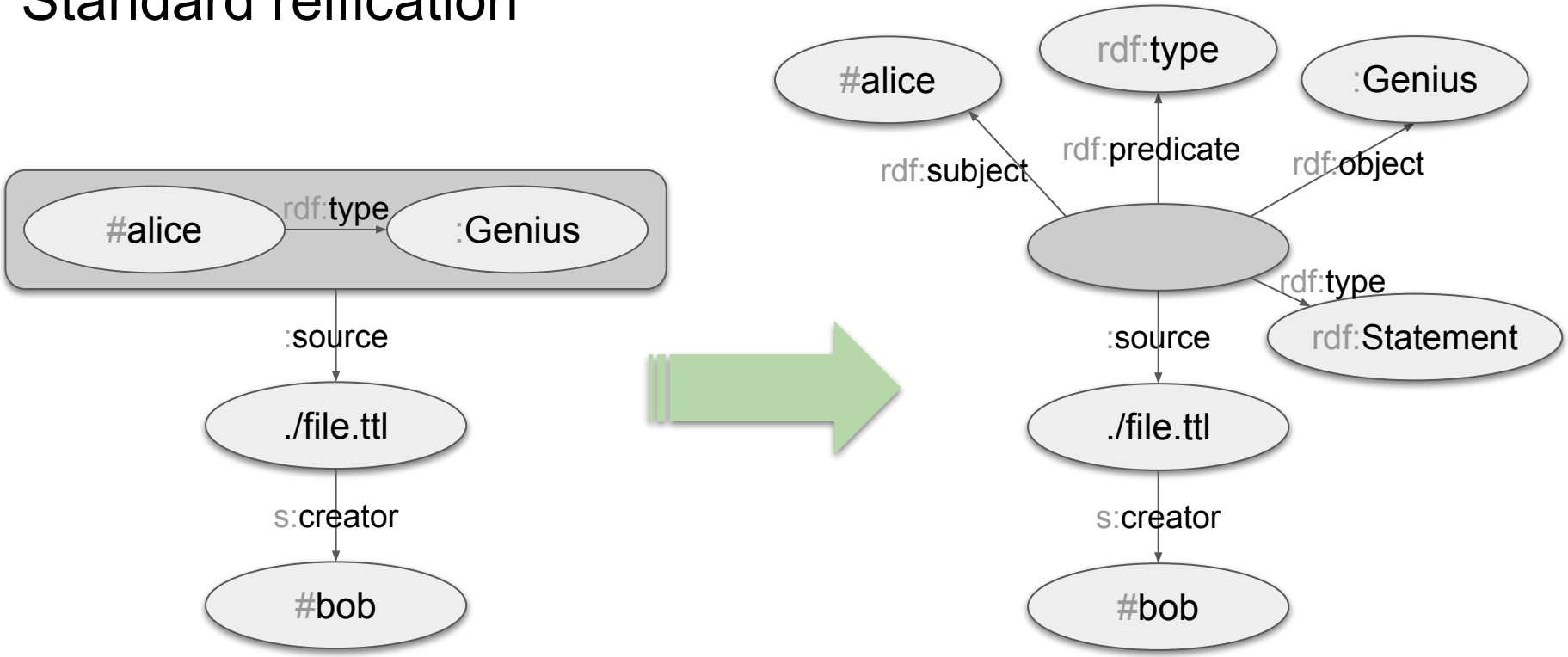
Existing approaches

Standard reification
(RDF 1.0 1999)

Single-triple named graphs
(Carroll et al. 2005,
RDF 1.1 2014)

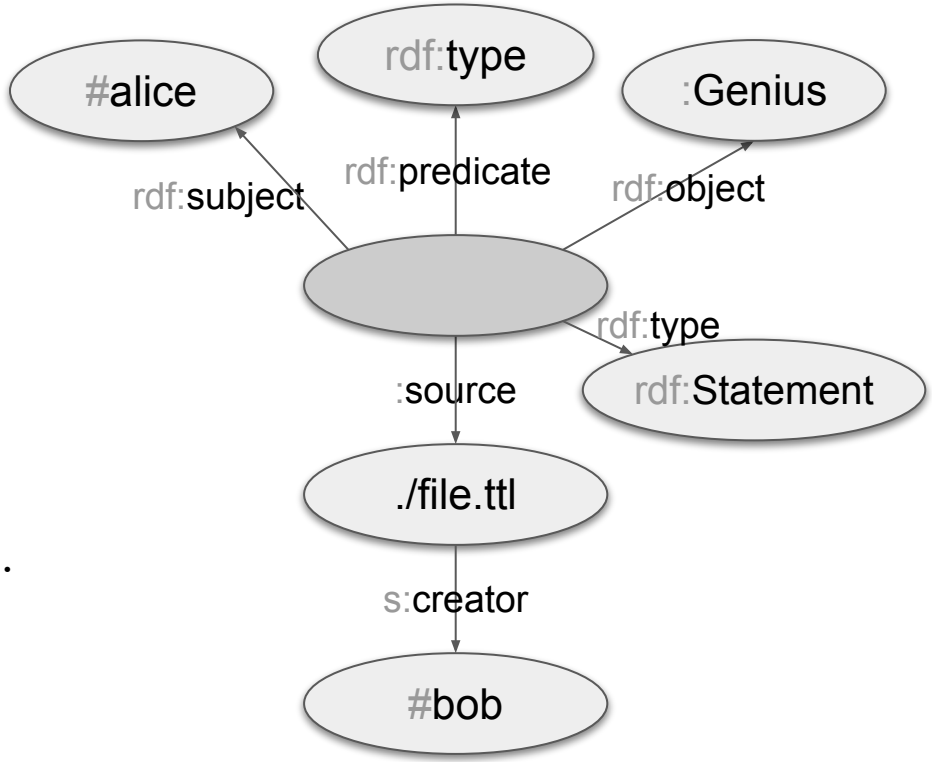
Singleton properties
(Nguyen et al. 2014)

Standard reification

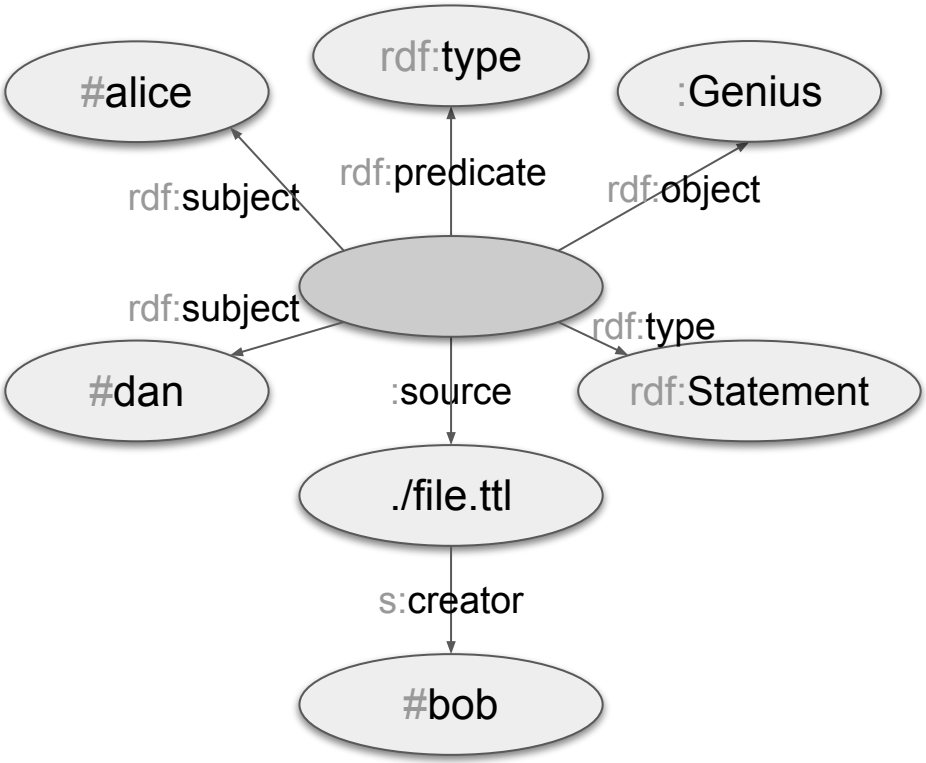


Querying standard reification

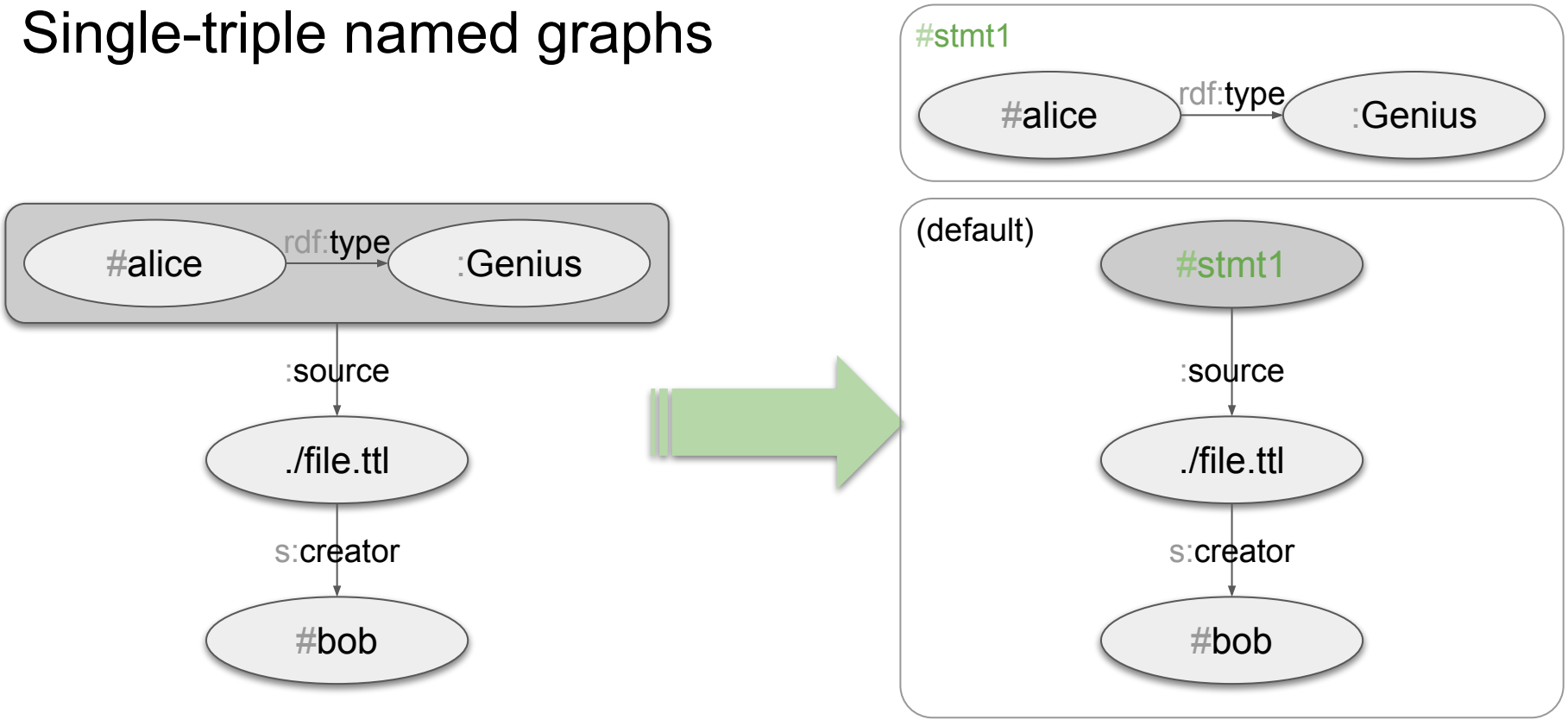
```
# Who is a genius according to whom?

PREFIX …

SELECT  ?allegedGenius  ?accordingTo
WHERE {
   ?claim   rdf:type       rdf:Statement .
   ?claim   rdf:subject    ?allegedGenius .
   ?claim   rdf:predicate  rdf:type .
   ?claim   rdf:object     :Genius .

   ?claim  :source         ?src .
   ?src    s:creator       ?accordingTo .
}
```



Incomplete / overloaded reified statement

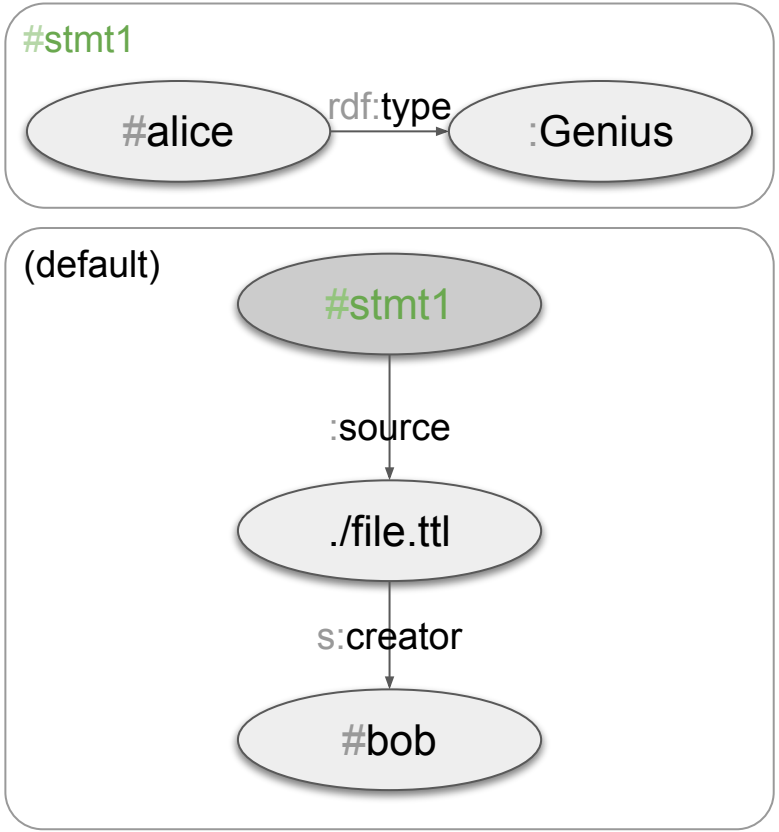


Single-triple named graphs

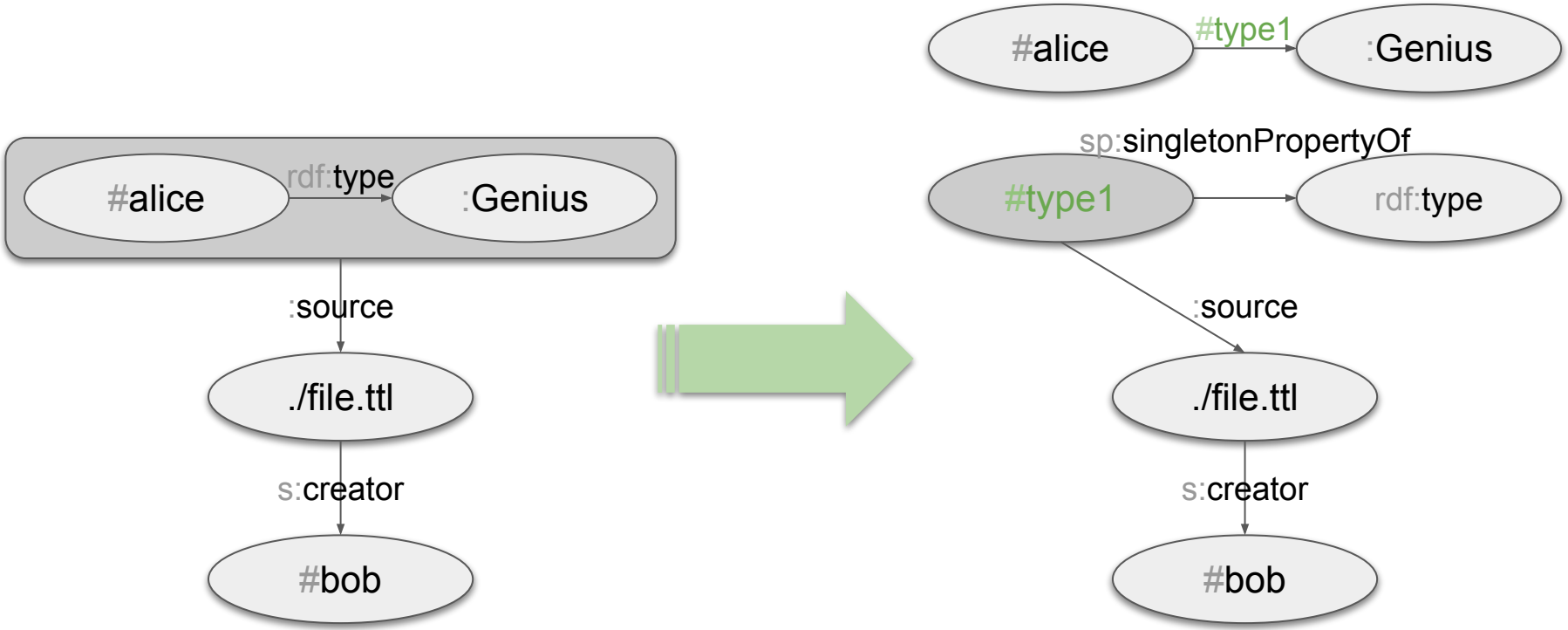


Querying single-triple named graphs

```
# Who is a genius according to whom?

PREFIX …

SELECT  ?allegedGenius  ?accordingTo
WHERE {
   GRAPH ?claim {
      ?allegedGenius  rdf:type  :Genius .
   }

   ?claim            :source    ?src .
   ?src              s:creator  ?accordingTo .
}
```



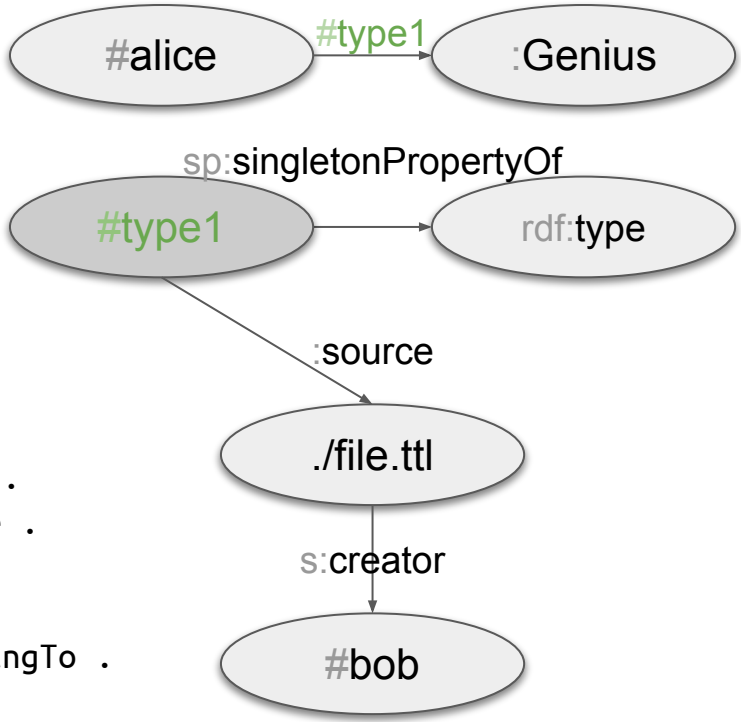
Singleton properties



Nguyen et al., "Don't like RDF reification? Making statements about statements using singleton property." WWW 2014.

Querying singleton properties

```
# Who is a genius according to whom?

PREFIX …

SELECT  ?allegedGenius  ?accordingTo
WHERE {
   ?allegedGenius  ?claim                    :Genius .
   ?claim          sp:singletonPropertyOf    rdf:type .

   ?claim          :source                   ?src .
   ?src            s:creator                 ?accordingTo .
}
```



Nguyen et al., "Don't like RDF reification? Making statements about statements using singleton property." WWW 2014.

Existing approaches: summary

Standard reification (RDF 1.0, 1999)

Pros:

- Standard

Cons:

- Verbose
- Incomplete / overloaded reified statements

Single-triple named graphs (Carroll et al. 2005, RDF 1.1, 2014)

Pros:

- Standard

Cons:

- Unspecified semantics
- Clutters datasets with “artificial” named graphs

Singleton properties (Nguyen et al., 2014)

Pros:

- Relatively concise

Cons:

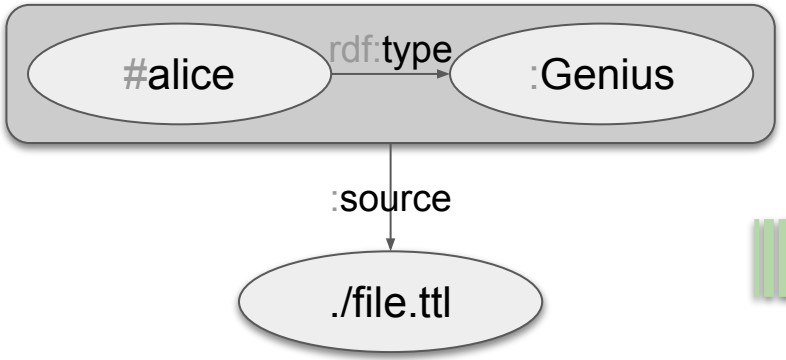
- Performance issues on many RDF systems

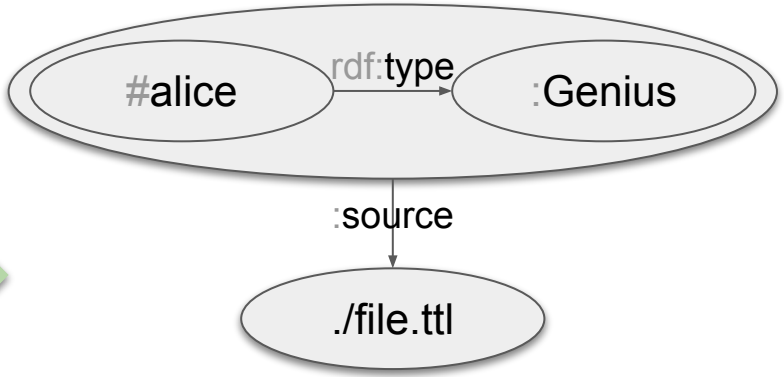
Part 2

Overview of the RDF-star approach

Why don’t we use *the triple itself* when talking about it?

Basic idea: nested triples






```
<< <#alice> a :Genius >>   :source   <./file.ttl> .
```

subject predicate object

...and nested triple patterns

```
SELECT ?allegedGenius ?accordingTo
WHERE {
  ?claim   rdf:type       rdf:Statement .
  ?claim   rdf:subject    ?allegedGenius .
  ?claim   rdf:predicate  rdf:type .
  ?claim   rdf:object     :Genius .
  ?claim   :source        ?src .
  ?src     s:creator      ?accordingTo .
}
```

```
SELECT  ?allegedGenius  ?accordingTo
WHERE {
  <<?allegedGenius rdf:type :Genius>> :source ?src .
  ?src  s:creator  ?accordingTo .
}
```

```
<< <#alice> a :Genius >>   :source   <./file.ttl> .
```

subject predicate object

...and nested triple patterns

```
SELECT ?allegedGenius ?accordingTo
WHERE {
  ?claim1  rdf:type       rdf:Statement .
  ?claim1  rdf:subject    ?allegedGenius .
  ?claim1  rdf:predicate  rdf:type .
  ?claim1  rdf:object     :Genius .
  ?claim1  :source         ?src1 .
  ?claim2  rdf:type       rdf:Statement .
  ?claim2  rdf:subject    ?allegedGenius .
  ?claim2  rdf:predicate  rdf:type .
  ?claim2  rdf:object     :Nerd .
  ?claim2  :source         ?src2 .

   FILTER ( ?src1 != ?src2 )

   ?src1   s:creator    ?accordingTo .
   ?src2   s:creator    ?accordingTo .
}
```

...and nested triple patterns

```
SELECT ?allegedGenius ?accordingTo
WHERE {
  ?claim1  rdf:type       rdf:Statement .
  ?claim1  rdf:subject    ?allegedGenius .
  ?claim1  rdf:predicate  rdf:type .
  ?claim1  rdf:object     :Genius .
  ?claim1  :source         ?src1 .
  ?claim2  rdf:type       rdf:Statement .
  ?claim2  rdf:subject    ?allegedGenius .
  ?claim2  rdf:predicate  rdf:type .
  ?claim2  rdf:object     :Nerd .
  ?claim2  :source         ?src2 .

   FILTER ( ?src1 != ?src2 )

   ?src1   s:creator    ?accordingTo .
   ?src2   s:creator    ?accordingTo .
}
```

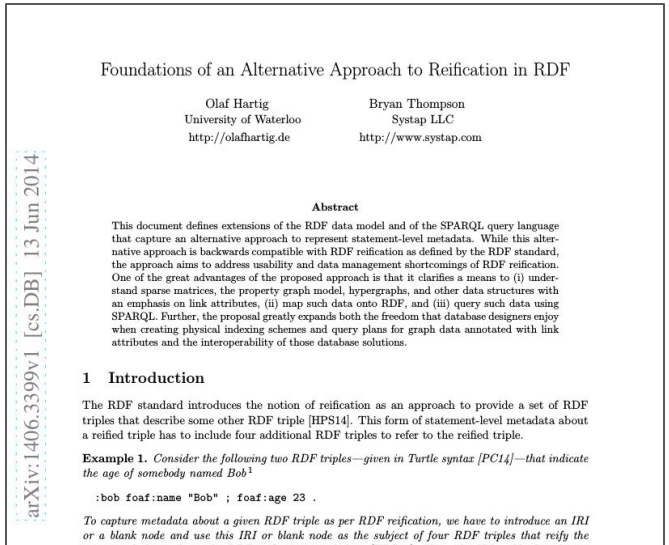
```
SELECT  ?allegedGenius  ?accordingTo
WHERE {
  <<?allegedGenius rdf:type :Genius>> :source ?src1.
  <<?allegedGenius rdf:type :Nerd>>   :source ?src2.

  FILTER ( ?src1 != ?src2 )

  ?src1  s:creator  ?accordingTo .
  ?src2  s:creator  ?accordingTo .
}
```

Historical perspective

- Ideas of such nested triples / triple patterns in the community since many years
  - Including an implementation in Blazegraph (“reification done right”)
  - Dagstuhl seminar on Semantic Data Management, April 2012

Historical perspective

- Ideas of such nested triples / triple patterns in the community since many years
  - Including an implementation in Blazegraph (“reification done right”)
  - Dagstuhl seminar on Semantic Data Management, April 2012
- Tech.report that defines the approach under the name RDF*/SPARQL*, June 2014
  - RDF* abstract syntax, Turtle* format, SPARQL* syntax and evaluation semantics



Foundations of an Alternative Approach to Reification in RDF

Olaf Hartig
University of Waterloo
http://olafhartig.de

Bryan Thompson
Systap LLC
http://www.systap.com

Abstract

This document defines extensions of the RDF data model and of the SPARQL query language that capture an alternative approach to represent statement-level metadata. While this alternative approach is backwards compatible with RDF reification as defined by the RDF standard, the approach aims to address usability and data management shortcomings of RDF reification. One of the great advantages of the proposed approach is that it clarifies a means to (i) understand sparse matrices, the property graph model, hypergraphs, and other data structures with an emphasis on link attributes, (ii) map such data onto RDF, and (iii) query such data using SPARQL. Further, the proposal greatly expands both the freedom that database designers enjoy when creating physical indexing schemes and query plans for graph data annotated with link attributes and the interoperability of those database solutions.

1 Introduction

The RDF standard introduces the notion of reification as an approach to provide a set of RDF triples that describe some other RDF triple [HPS14]. This form of statement-level metadata about a reified triple has to include four additional RDF triples to refer to the reified triple.

Example 1. *Consider the following two RDF triples—given in Turtle syntax [PC14]—that indicate the age of somebody named Bob*[1]

```
:bob foaf:name "Bob" ; foaf:age 23 .
```

To capture metadata about a given RDF triple as per RDF reification, we have to introduce an IRI or a blank node and use this IRI or blank node as the subject of four RDF triples that reify the

Historical perspective

- Ideas of such nested triples / triple patterns in the community since many years
  - Including an implementation in Blazegraph (“reification done right”)
  - Dagstuhl seminar on Semantic Data Management, April 2012
- Tech.report that defines the approach under the name RDF*/SPARQL*, June 2014
  - RDF* abstract syntax, Turtle* format, SPARQL* syntax and evaluation semantics
- Adoption
  - Blazegraph, AnzoGraph, Stardog, GraphDB, Neo4j’s neosemantics
  - Apache Jena, Eclipse RDF4J, RDF.rb, N3.js, EYE
  - YAGO 4 knowledge graph released as a Turtle* file



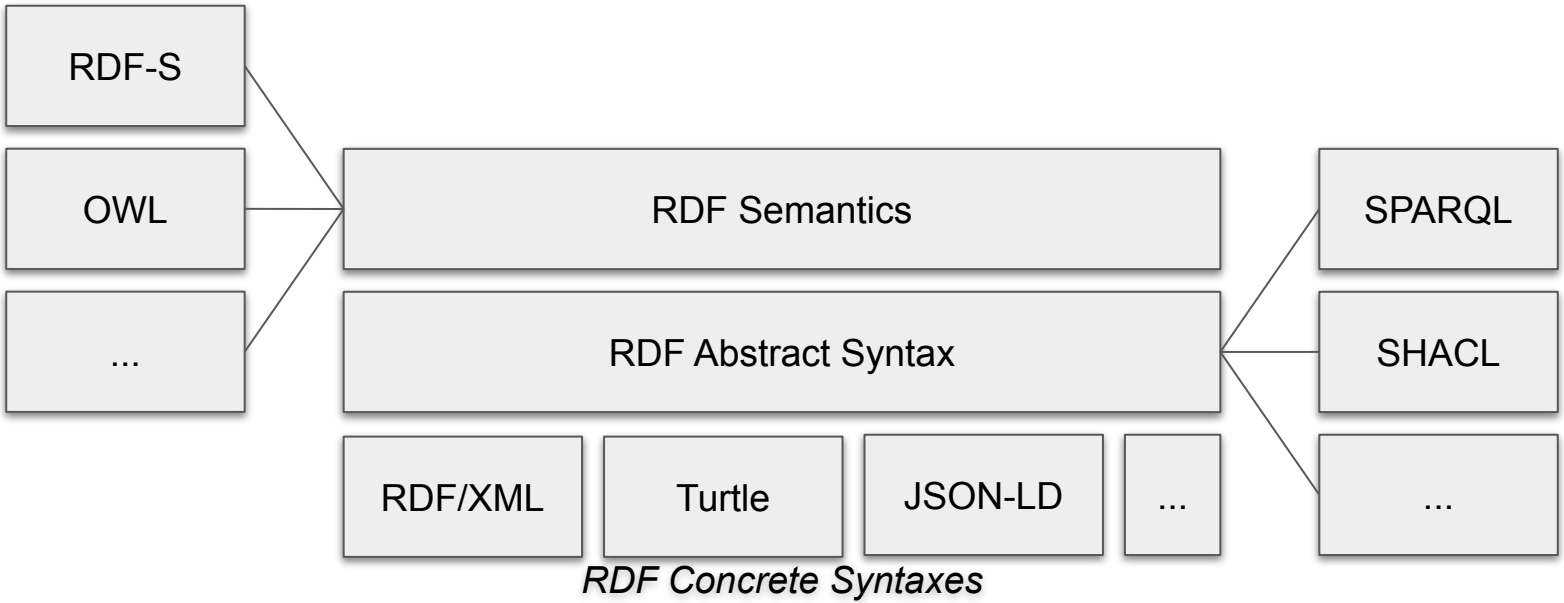
Historical perspective

- Ideas of such nested triples / triple patterns i
  - Including an implementation in Blazegraph (“reificati
  - Dagstuhl seminar on Semantic Data Management, A
- Tech.report that defines the approach under
  - RDF* abstract syntax, Turtle* format, SPARQL* synt
- Adoption
  - Blazegraph, AnzoGraph, Stardog, GraphDB, Neo4j’s
  - Apache Jena, Eclipse RDF4J, RDF.rb, N3.js, EYE
  - YAGO 4 knowledge graph released as a Turtle* file
- W3C Workshop on Web Standardization for Graph Data, Berlin, March 2019

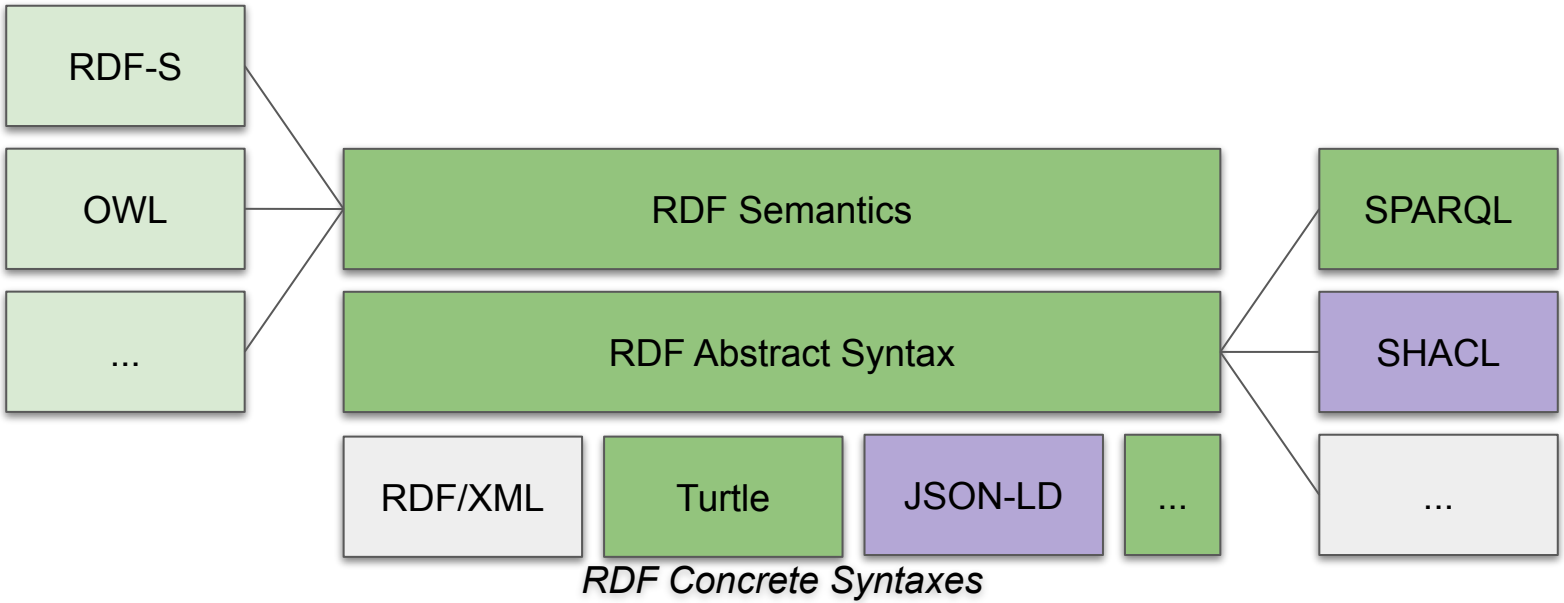
Historical perspective

- Ideas of such nested triples / triple patterns in the community since many years
  - Including an implementation in Blazegraph (“reification done right”)
  - Dagstuhl seminar on Semantic Data Management, April 2012
- Tech.report that defines the approach under the name RDF*/SPARQL*, June 2014
  - RDF* abstract syntax, Turtle* format, SPARQL* syntax and evaluation semantics
- Adoption
  - Blazegraph, AnzoGraph, Stardog, GraphDB, Neo4j’s neosemantics
  - Apache Jena, Eclipse RDF4J, RDF.rb, N3.js, EYE
  - YAGO 4 knowledge graph released as a Turtle* file
- W3C Workshop on Web Standardization for Graph Data, Berlin, March 2019
- Community task force as part of the W3C RDF-DEV CG
  - Mixture of implementers, users, and academic researchers
  - Goal: create a spec that captures all aspects of the approach in the form of a CG report, plus a collection of corresponding test suites
  - Some aspects of the approach have changed → new names: RDF-star, SPARQL-star, etc.

RDF and related specs

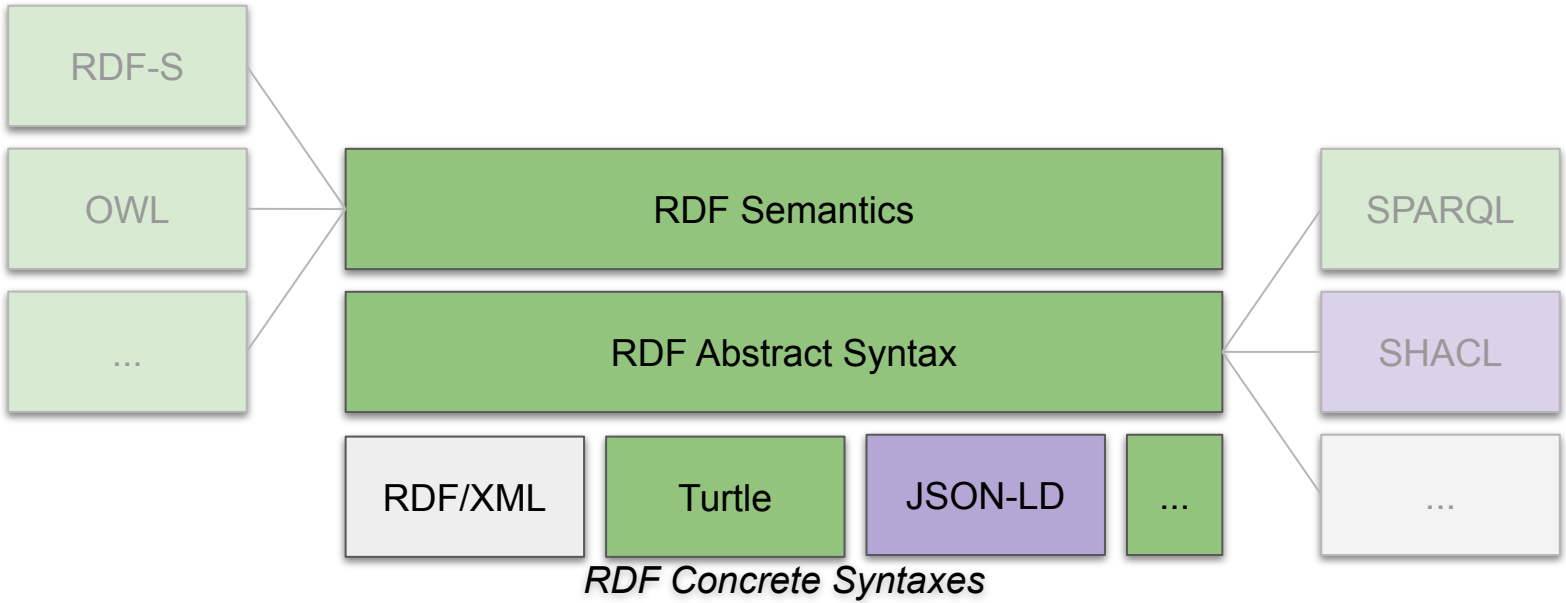


RDF and related specs → RDF-star

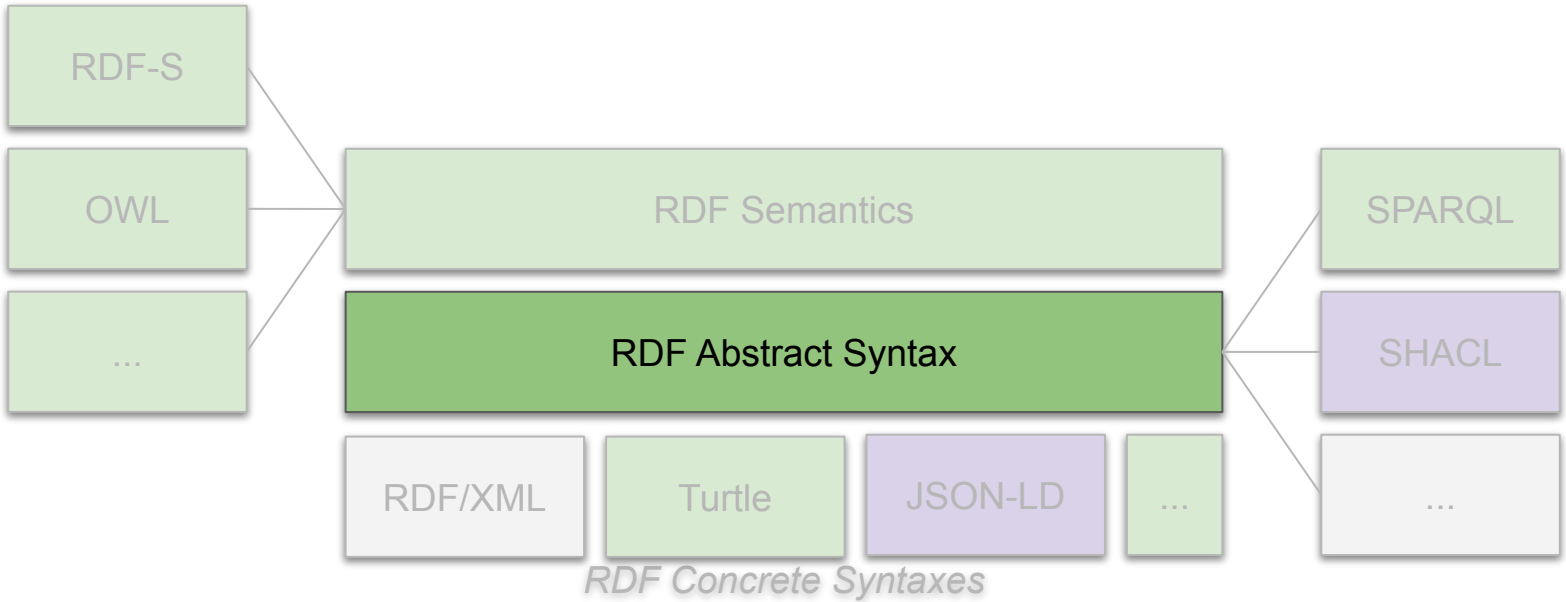


Part 3

From RDF to RDF-star



RDF-star abstract syntax



Part 3 : RDF-star data

Graph ::= Triple*

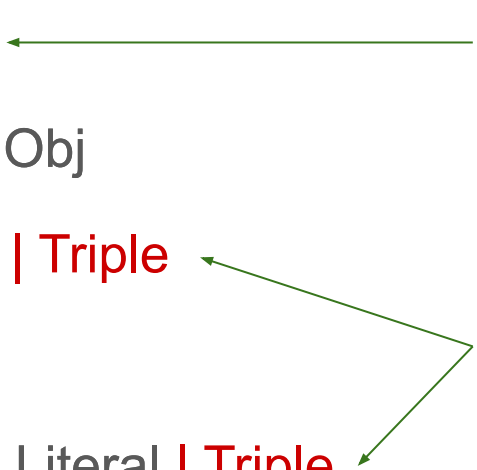
Triple ::= Subj Pred Obj

Subj ::= IRI | Bnode | Triple

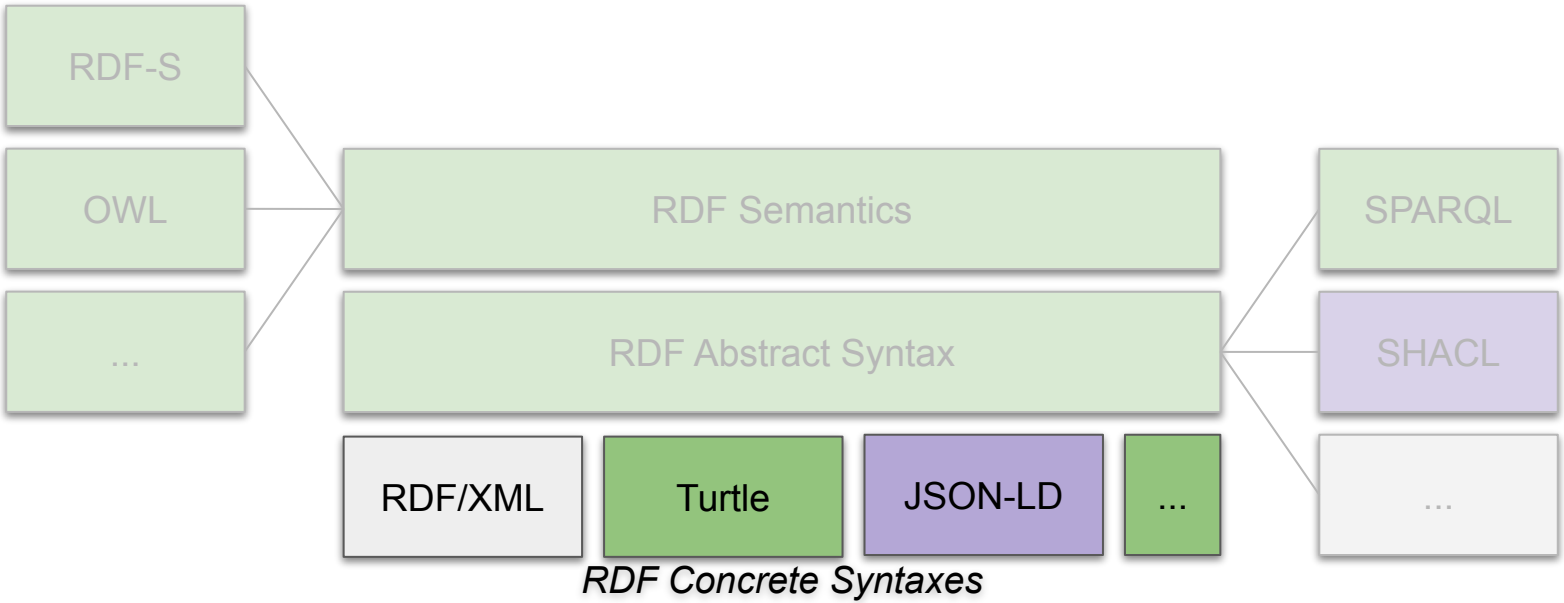
Pred ::= IRI

Obj ::= IRI | Bnode | Literal | Triple

Dataset ::= Graph (IRI Graph)*

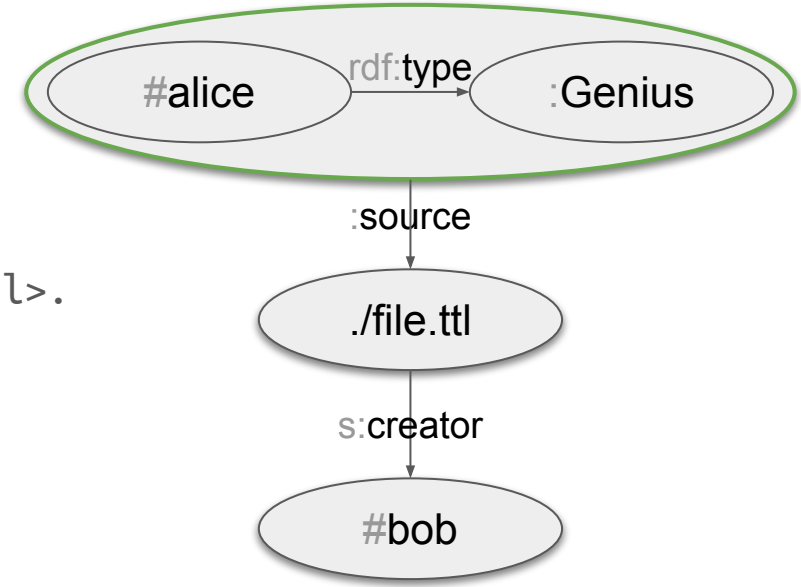


RDF-star Concrete syntaxes



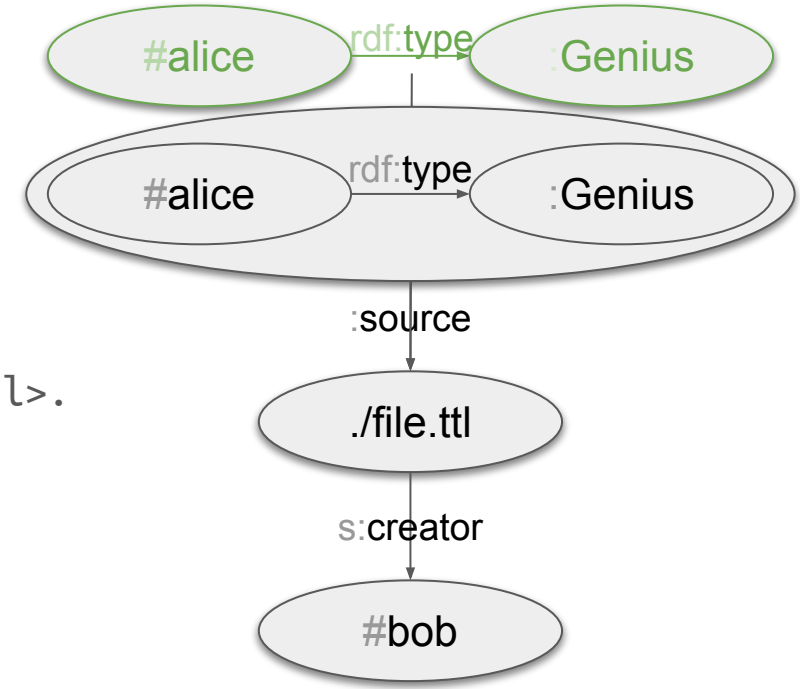
Turtle-star

```
<< <#alice> a :Genius >> :source <./file.ttl>.
<./file.ttl> s:creator <#bob>.
```



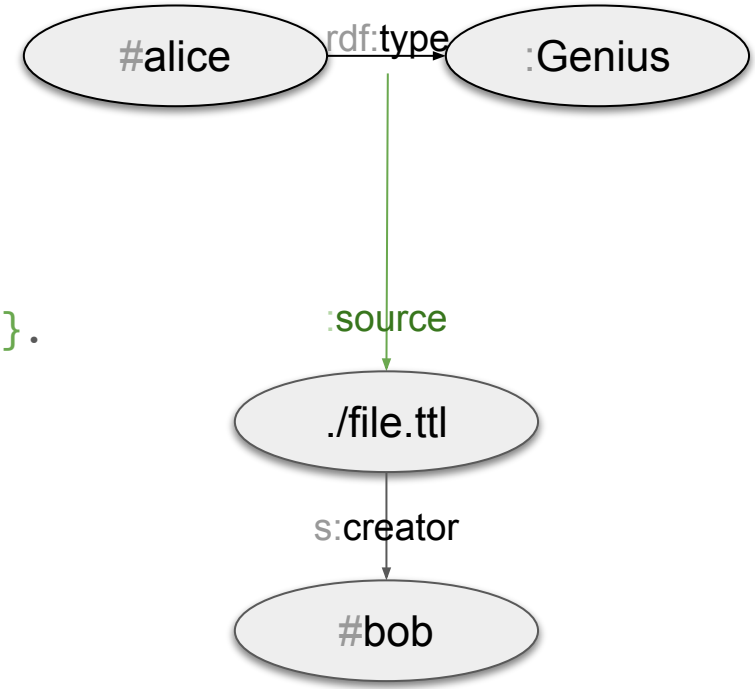
Turtle-star

```
<#alice> a :Genius.
<< <#alice> a :Genius >> :source <./file.ttl>.
<./file.ttl> s:creator <#bob>.
```

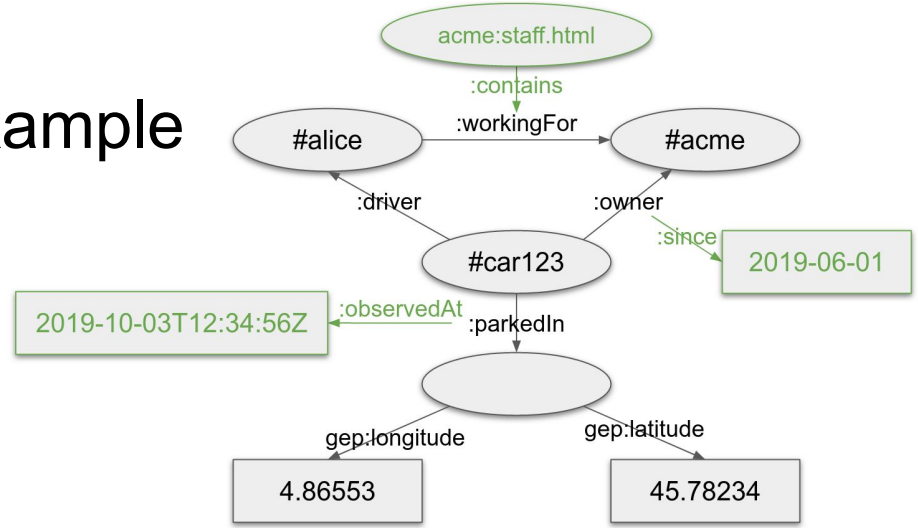


Turtle-star: annotation syntax

```
<#alice> a :Genius {| :source <./file.ttl> |}.

<./file.ttl> s:creator <#bob>.
```



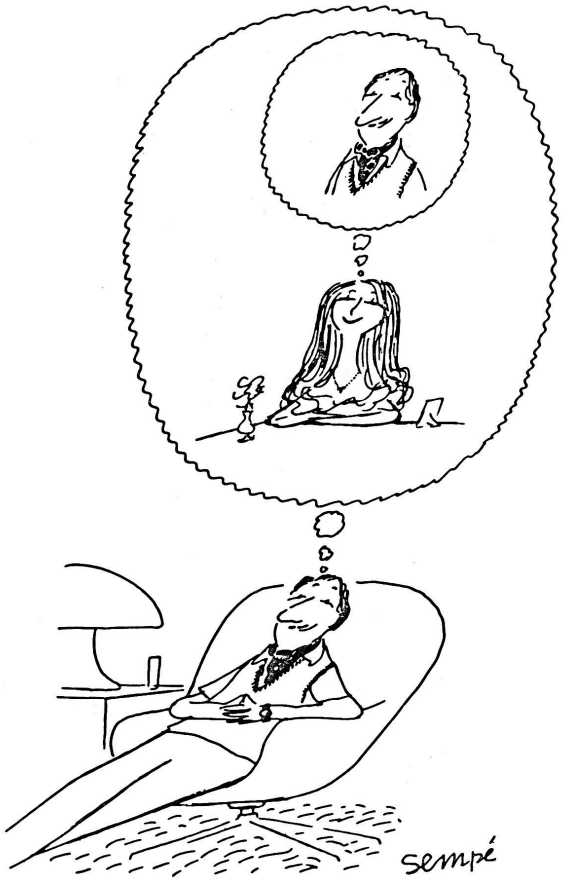
Turtle-star: a more complex example




```
<#alice> :workingFor <#acme>.
<#car123>
    :driver <#alice>;
    :owner <#acme> {| :since "2019-06-01"^^xsd:date |};
    :parkedIn _:pos {| :observedAt "2019-10-03T12:34:56Z"^^xsd:dateTime |}.
_:pos
    geo:longitude 4.86553;
    geo:latitude 45.78234.
acme:staff.html :contains << <#alice> :workingFor <#acme> >>.
```

Turtle-star: deeply nested triples

```
<#cartoon>
    :depicts <<
        <#bob> :dreamingOf <<
            <#alice> :dreamingOf <#bob>
        >>
    >>.
```

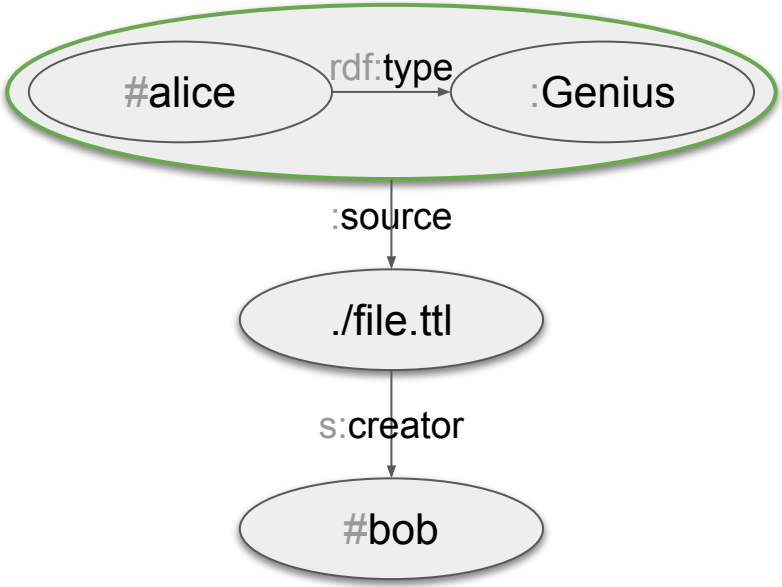


Other concrete syntaxes

- N-Triples-star
  - Subset of Turtle-star
  - One (potentially nested) RDF-star triple per line, no annotation syntax here
- N-Quads-star
  - For RDF-star datasets
  - N-Triples-star + optional graph name
- JSON-LD-star
  - https://json-ld.github.io/json-ld-star/

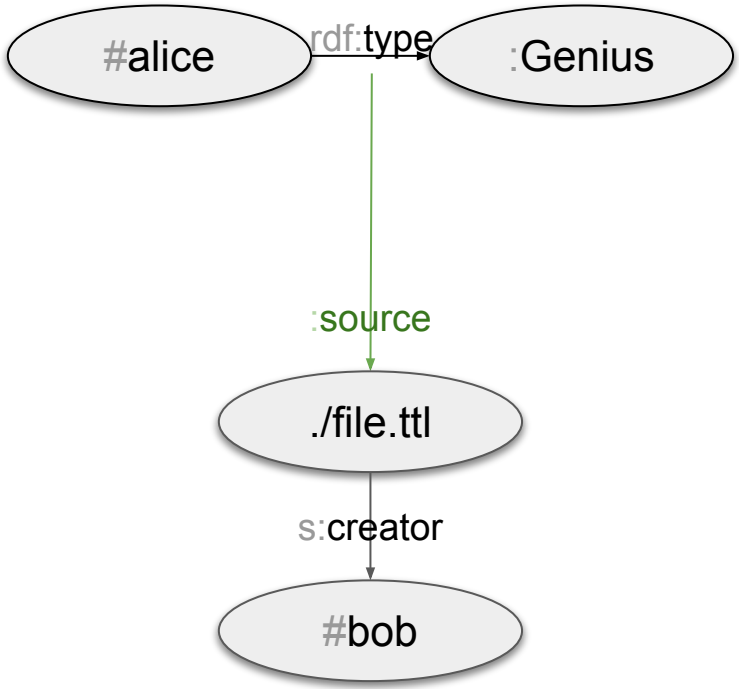
JSON-LD-star

```
{
    “@context”: “...”,
    “id”: {
        “id”: “#alice”,
        “type”: “Genius”
    },
    “source”: {
        “id”: “./file.ttl”,
        “creator”: “#bob”
    }
}
```

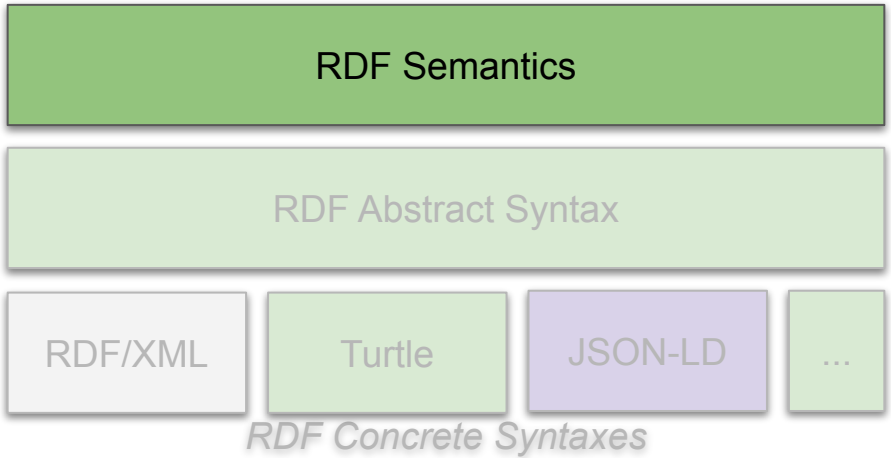


JSON-LD-star annotation

```
{
    “@context”: “...”,
    “id”: “#alice”,
    “type”: {
        “id”: “Genius”,
        “@annotation”: {
            “source”: {
                “id”: “./file.ttl”,
                “creator”: “#bob”
            }
        }
    }
}
```



RDF-star semantics



RDF-star semantics relies on standard RDF semantics

- Don’t reinvent the wheel
- RDF-star can (in theory) be implemented on top of a standard RDF system
- Semantics extensions (RDF-S, OWL…) can be directly applied to RDF-star

Embedded triples are not automatically asserted

- Goodbye PG-mode and SA-mode
- Welcome annotation syntax

```
<< <#alice> a :Genius >>
          :source <./file.ttl>.
```

```
<#alice> a :Genius
    {| :source <./file.ttl> |}.
```

Asserted triples can not be “cancelled”

- Not in RDF, not in RDF-star
- (monotonic semantics)

```
<#alice> :workingFor <#acme> {|
    :since “2012-03-04”xsd:date;
    :until “2018-09-10”xsd:date;
|}.
```

```
@prefix s: <http://schema.org/>.

<< <#acme> s:employee <#alice> >>
    :since “2012-03-04”xsd:date;
    :until “2018-09-10”xsd:date;
.
```

Triples are “unique”

- They are *entirely* identified by their subject, predicate and object
- << :s :p :o >> is the **same** triple everywhere it occurs

```
<< <#alice> :workingFor <#acme> >>
   :addedBy <#alice>;
   :addedIn <./file1.ttl>.

<< <#alice> :workingFor <#acme> >>
   :addedBy <#bob>;
   :addedIn <./file2.ttl>.
```

```
<< <#alice> :workingFor <#acme> >>
        :occurences _:occ1, _:occ2.

_:occ1 :addedBy <#alice>;
        :addedIn <./file1.ttl>.

_:occ2 :addedBy <#bob>;
        :addedIn <./file2.ttl>.
```

Referential opacity

- Aside:

```
:superman owl:sameAs :clark .
:superman :can       :fly .
```

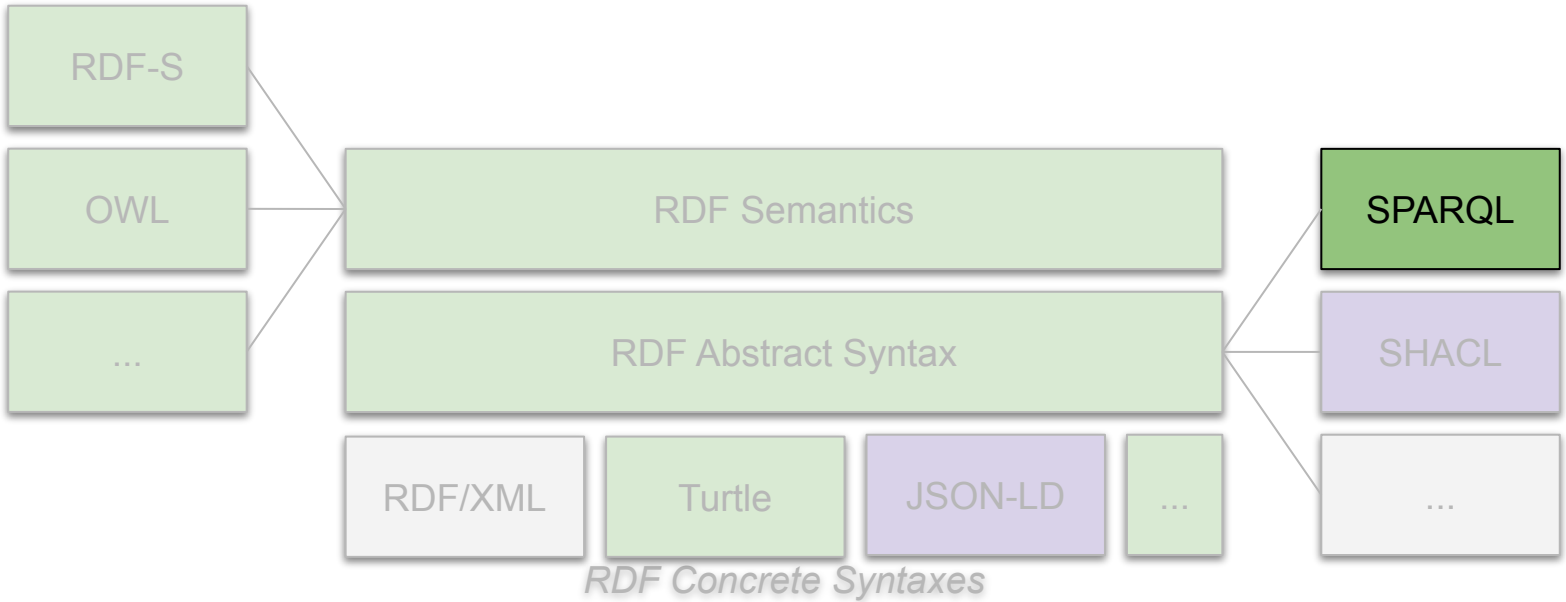
→ `:clark :can :fly .`

- An “explainable AI” use-case

```
<< :superman :can :fly >>
  prov:wasDerivedFrom
    ( << :superman owl:sameAs :clark >>
      << :superman :can       :fly   >> ) .
```

Part 4

From SPARQL to SPARQL-star



Nested Triple Patterns

```
SELECT  ?allegedGenius  ?accordingTo
WHERE {
  <<?allegedGenius rdf:type :Genius>> :source ?src .
  ?src  s:creator  ?accordingTo .
}
```

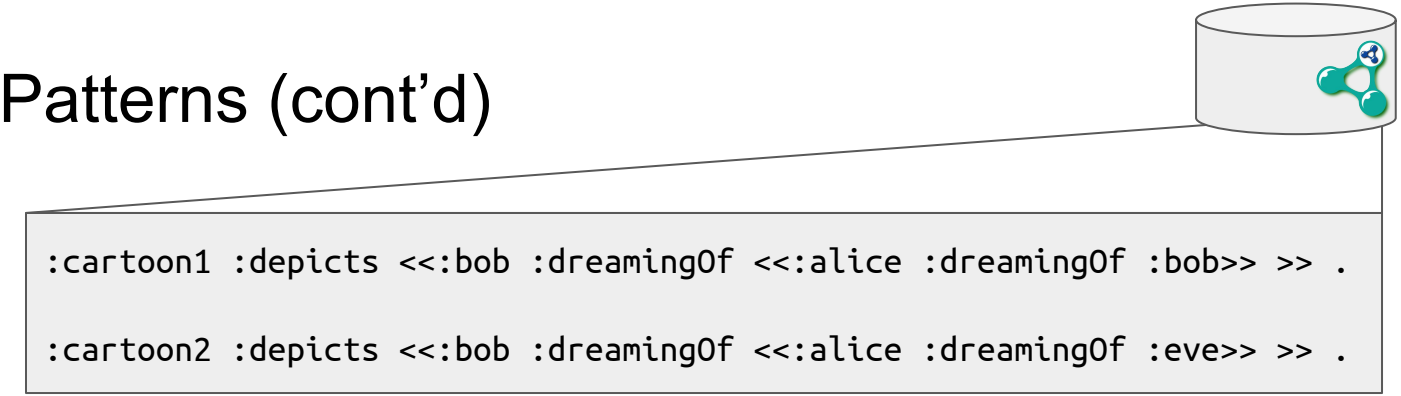
Nested Triple Patterns (cont’d)

```
SELECT  ?allegedGenius  ?accordingTo
WHERE {
  <<?allegedGenius rdf:type :Genius>> :source ?src1.
  <<?allegedGenius rdf:type :Nerd>>   :source ?src2.

  FILTER ( ?src1 != ?src2 )

  ?src1  s:creator  ?accordingTo .
  ?src2  s:creator  ?accordingTo .
}
```

Nested Triple Patterns (cont’d)

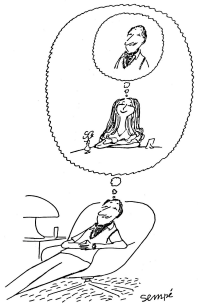
```
SELECT  ?allegedGenius
WHERE {
  <<?allegedGenius rdf:type :Genius>> :source ?src1 ;
                                      :source ?src2 .

  FILTER ( ?src1 != ?src2 )

  ?src1  s:creator  ?accordingTo1 .
  ?src2  s:creator  ?accordingTo2 .

  FILTER ( ?accordingTo1 != ?accordingTo2 )
}
```

Nested Triple Patterns (cont'd)



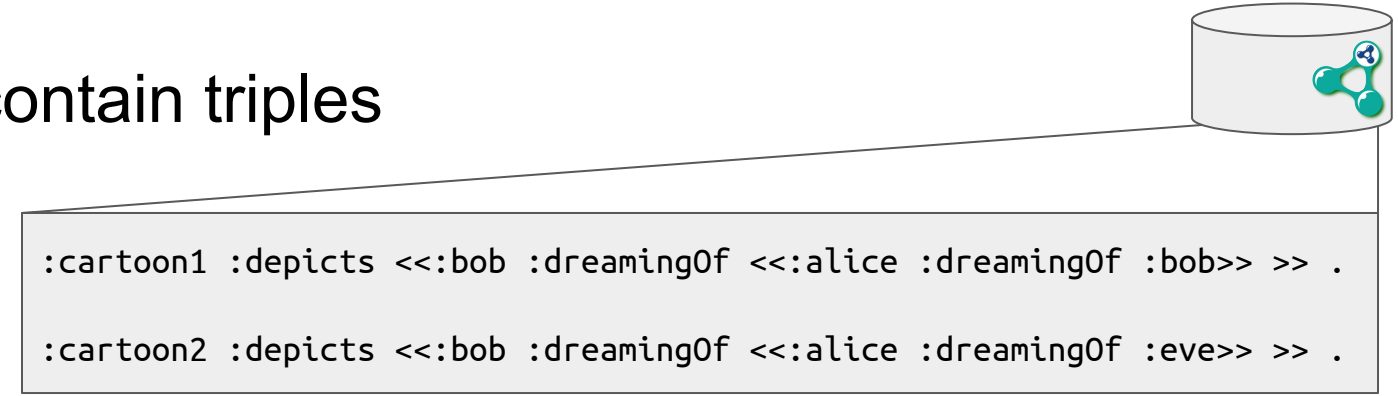
```
:cartoon1 :depicts <<:bob :dreamingOf <<:alice :dreamingOf :bob>> >> .

:cartoon2 :depicts <<:bob :dreamingOf <<:alice :dreamingOf :eve>> >> .
```

```
SELECT ?cartoon ?b
WHERE {
  ?cartoon :depicts <<?b :dreamingOf <<?a :dreamingOf ?b>> >>
}
```



?cartoon	?b
:cartoon1	:bob

Results may contain triples

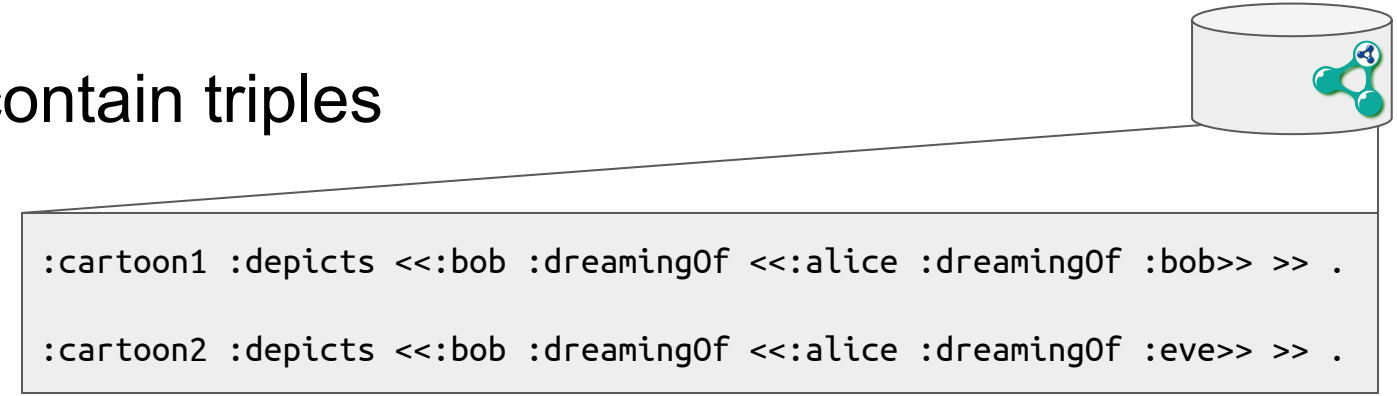


```
:cartoon1 :depicts <<:bob :dreamingOf <<:alice :dreamingOf :bob>> >> .

:cartoon2 :depicts <<:bob :dreamingOf <<:alice :dreamingOf :eve>> >> .
```

```
SELECT ?cartoon ?y
WHERE {
  ?cartoon :depicts <<?b :dreamingOf ?y >>
}
```

?cartoon	?y
:cartoon1	(:alice, :dreamingOf, :bob)
:cartoon2	(:alice, :dreamingOf, :eve)

Results may contain triples

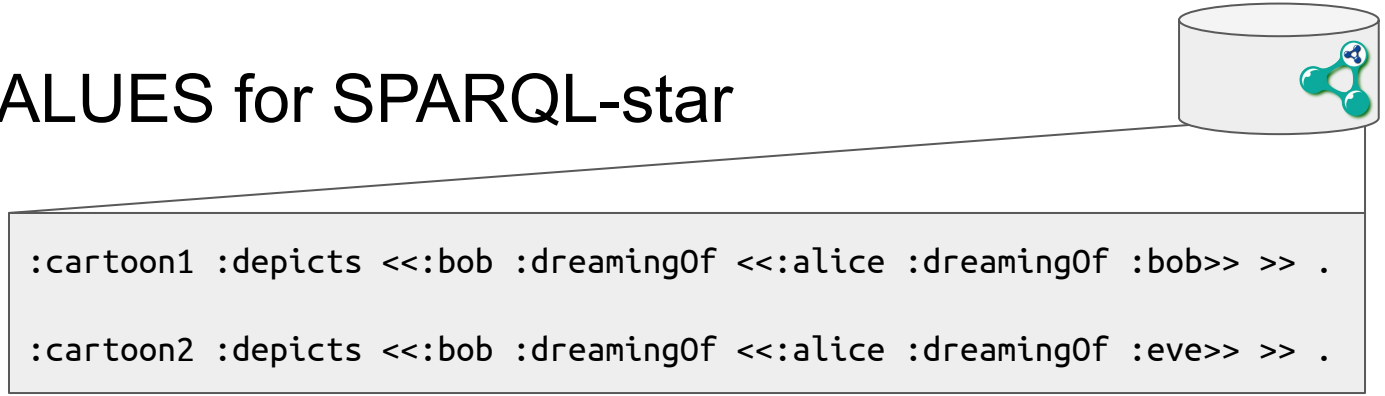


```
:cartoon1 :depicts <<:bob :dreamingOf <<:alice :dreamingOf :bob>> >> .

:cartoon2 :depicts <<:bob :dreamingOf <<:alice :dreamingOf :eve>> >> .
```

```
SELECT ?cartoon ?y
WHERE {
  ?cartoon :depicts ?y
}
```

?cartoon	?y
:cartoon1	(:bob, :dreamingOf, (:alice, :dreamingOf, :bob))
:cartoon2	(:bob, :dreamingOf, (:alice, :dreamingOf, :eve))

Extension of VALUES for SPARQL-star

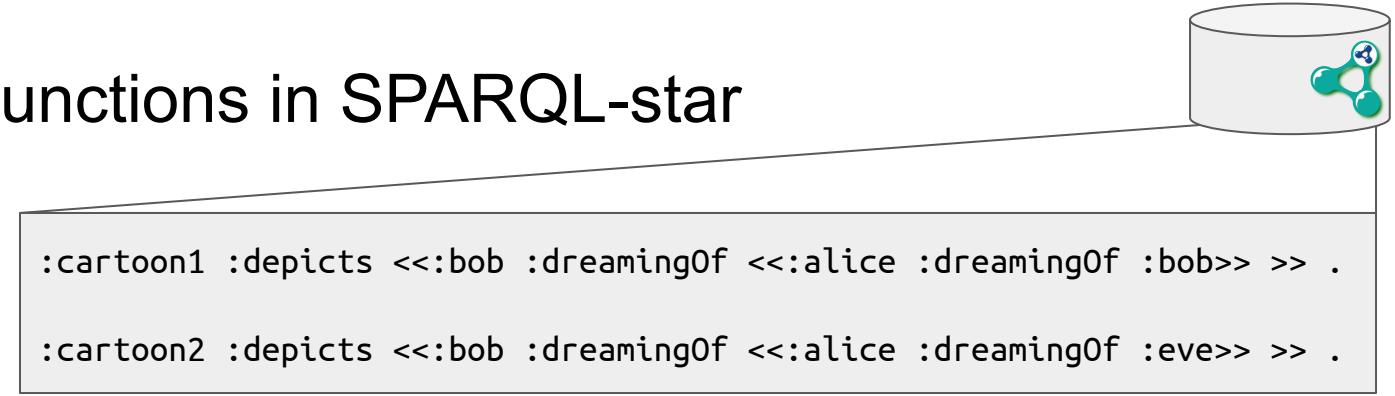


```
:cartoon1 :depicts <<:bob :dreamingOf <<:alice :dreamingOf :bob>> >> .

:cartoon2 :depicts <<:bob :dreamingOf <<:alice :dreamingOf :eve>> >> .
```

```
SELECT ?cartoon ?y
WHERE {
  VALUES ?y { <<:alice :dreamingOf :bob>>  <<:alice :dreamingOf :eve>> }

  ?cartoon :depicts << :bob :dreamingOf ?y >>
}
```

?cartoon	?y
:cartoon1	(:alice, :dreamingOf, :bob)
:cartoon2	(:alice, :dreamingOf, :eve)

New Built-In Functions in SPARQL-star



```
:cartoon1 :depicts <<:bob :dreamingOf <<:alice :dreamingOf :bob>> >> .

:cartoon2 :depicts <<:bob :dreamingOf <<:alice :dreamingOf :eve>> >> .
```

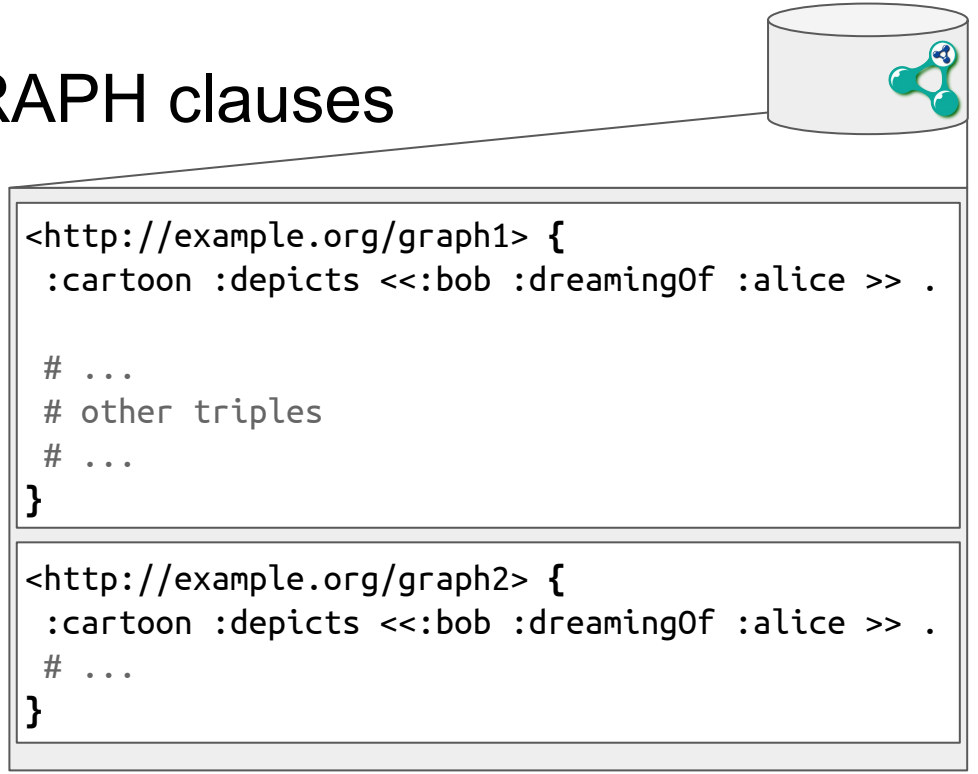
```
SELECT ?cartoon ?y ?o
WHERE {
  ?cartoon :depicts << :bob :dreamingOf ?y >>
  FILTER( isTriple(?y) )
  BIND( Object(?y) AS ?o )
}
```

?cartoon	?y	?o
:cartoon1	(:alice, :dreamingOf, :bob)	:bob
:cartoon2	(:alice, :dreamingOf, :eve)	:eve

```
SELECT ?cartoon ?t
WHERE {
  ?cartoon :depicts << ?x :dreamingOf ?y >>
  BIND( TRIPLE(?x,:state,:sleeping) AS ?t )
}
```

?cartoon	?t
:cartoon1	(:bob, :state, :sleeping)
:cartoon2	(:bob, :state, :sleeping)

Nested Triple Patterns in GRAPH clauses



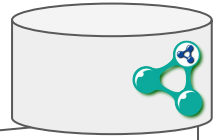
```
<http://example.org/graph1> {
 :cartoon :depicts <<:bob :dreamingOf :alice >> .

 # ...
 # other triples
 # ...
}
```

```
<http://example.org/graph2> {
 :cartoon :depicts <<:bob :dreamingOf :alice >> .
 # ...
}
```

```
SELECT ?cartoon ?g1 ?g2
WHERE {
  GRAPH ?g1 {
   ?cartoon :depicts <<?b :dreamingOf ?x>>
  }
  GRAPH ?g2 {
   ?cartoon :depicts <<?b :dreamingOf ?x>>
  }
  FILTER ( ?g1 != ?g2 )
}
```

?cartoon	?g1	?g2
:cartoon	http://example.org/graph1	http://example.org/graph2
:cartoon	http://example.org/graph2	http://example.org/graph1

Annotation Syntax in SPARQL-star



```
<< :alice :rdf:type :Genius >>  :source  <./file1.ttl> .
:alice :rdf:type :Genius .
<< :alice :rdf:type :Nerd >>  :source  <./file2.ttl> .
```

≡

```
:alice :rdf:type :Genius {| :source  <./file1.ttl> |} .
<< :alice :rdf:type :Nerd >>  :source  <./file2.ttl> .
```

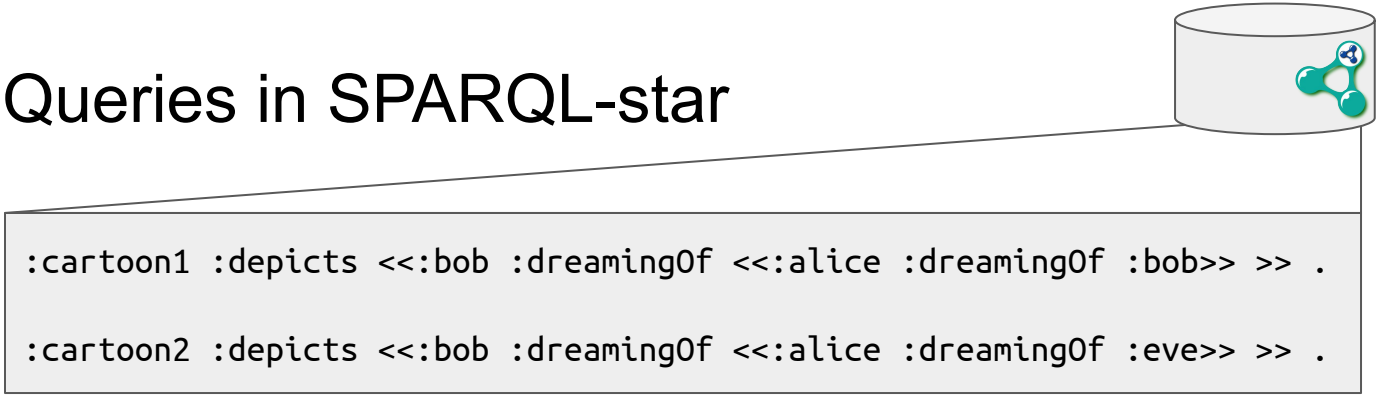
```
SELECT ?allegedGenius ?type ?src
WHERE {
    ?allegedGenius rdf:type ?type {| :source ?src |} .
}
```

≡

```
SELECT ?allegedGenius ?type ?src
WHERE {
    ?allegedGenius rdf:type ?type .
 << ?allegedGenius rdf:type ?type >> :source ?src .
}
```

?allegedGenius	?type	?src
:alice	:Genius	./file1.ttl

CONSTRUCT Queries in SPARQL-star

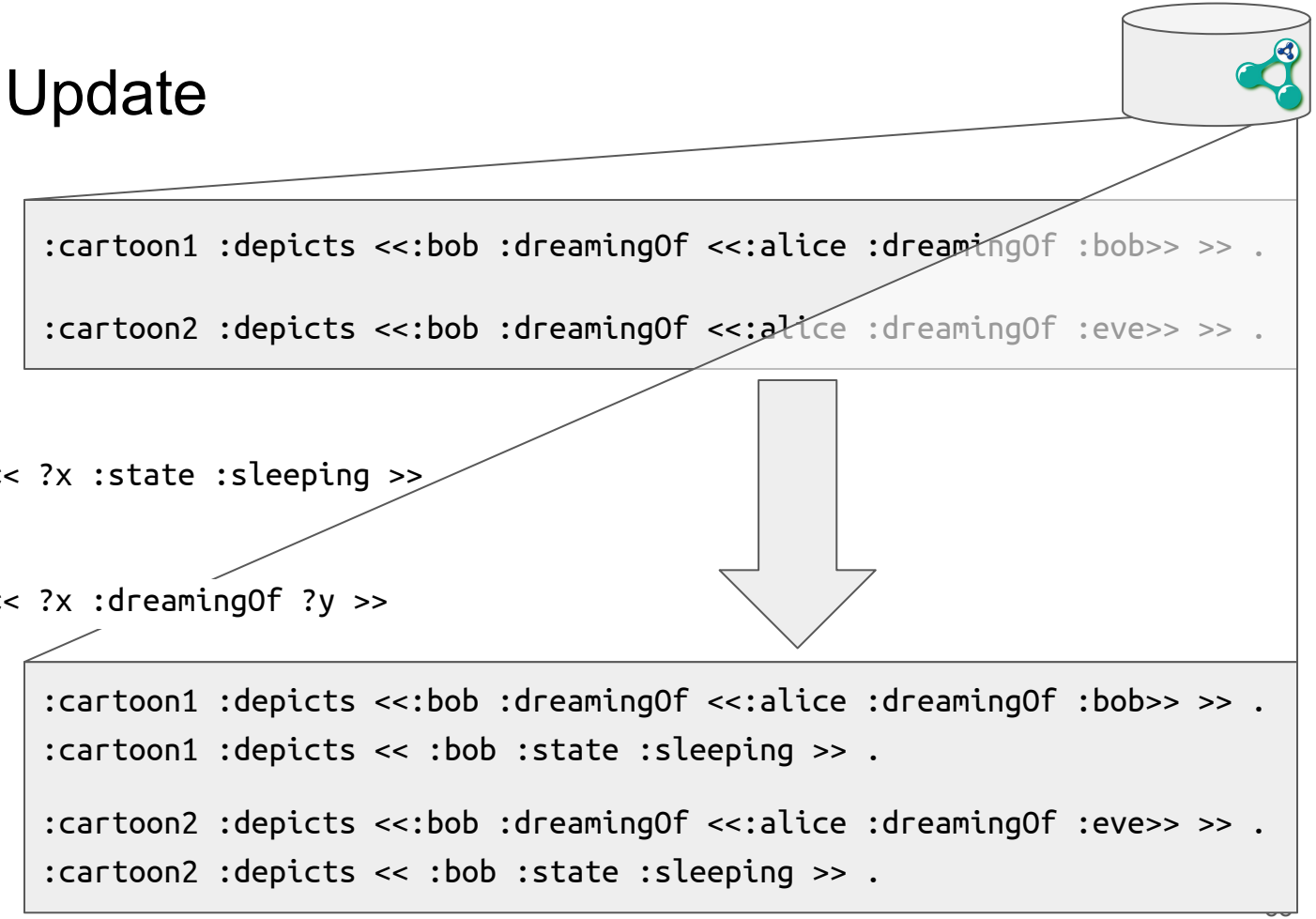


```
:cartoon1 :depicts <<:bob :dreamingOf <<:alice :dreamingOf :bob>> >> .

:cartoon2 :depicts <<:bob :dreamingOf <<:alice :dreamingOf :eve>> >> .
```

```
CONSTRUCT {
  ?cartoon :depicts << ?x :state :sleeping >>
}
WHERE {
  ?cartoon :depicts << ?x :dreamingOf ?y >>
}
```

```
:cartoon1 :depicts << :bob :state :sleeping >> .

:cartoon2 :depicts << :bob :state :sleeping >> .
```

SPARQL-star Update




```
INSERT {
  ?cartoon :depicts << ?x :state :sleeping >>
}
WHERE {
  ?cartoon :depicts << ?x :dreamingOf ?y >>
}
```

Other Contributions for SPARQL-star

- Extension of the SPARQL result formats (JSON and XML)
- Test suites
  - 61 syntax tests for SPARQL-star
  - 29 evaluation tests

Part 5

Conclusion and perspectives

- The specification is relatively stable now
  - https://w3c.github.io/rdf-star/
  - second public draft on its way
  - expecting implementation reports from various implementers
  - → moment of truth

- What's next
  - SHACL-star ?
  - moving to recommendation track ?

Thank you for your attention

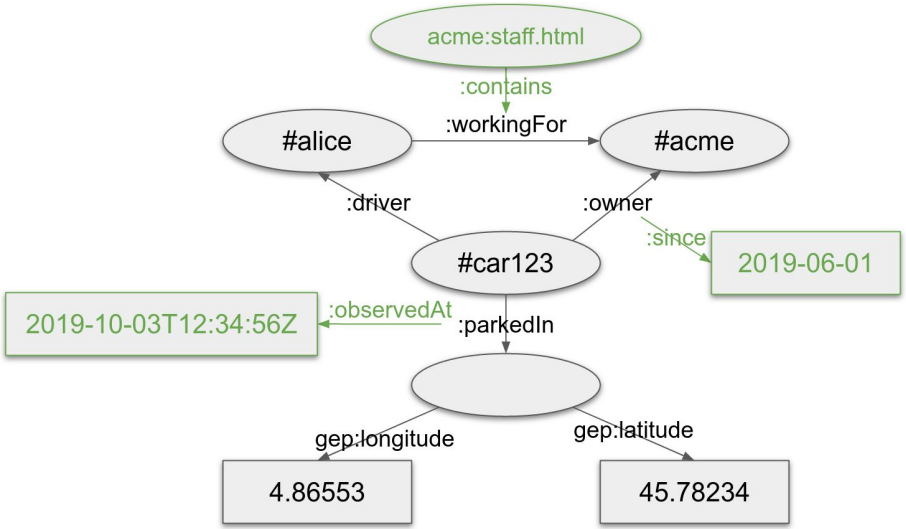
Do you have any question?

https://w3c.github.io/rdf-star/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825333, as well as from both the Swedish Research Council (Vetenskapsrådet, project reg. no. 2019-05655) and the CENIIT program at Linköping University (project no. 17.05).

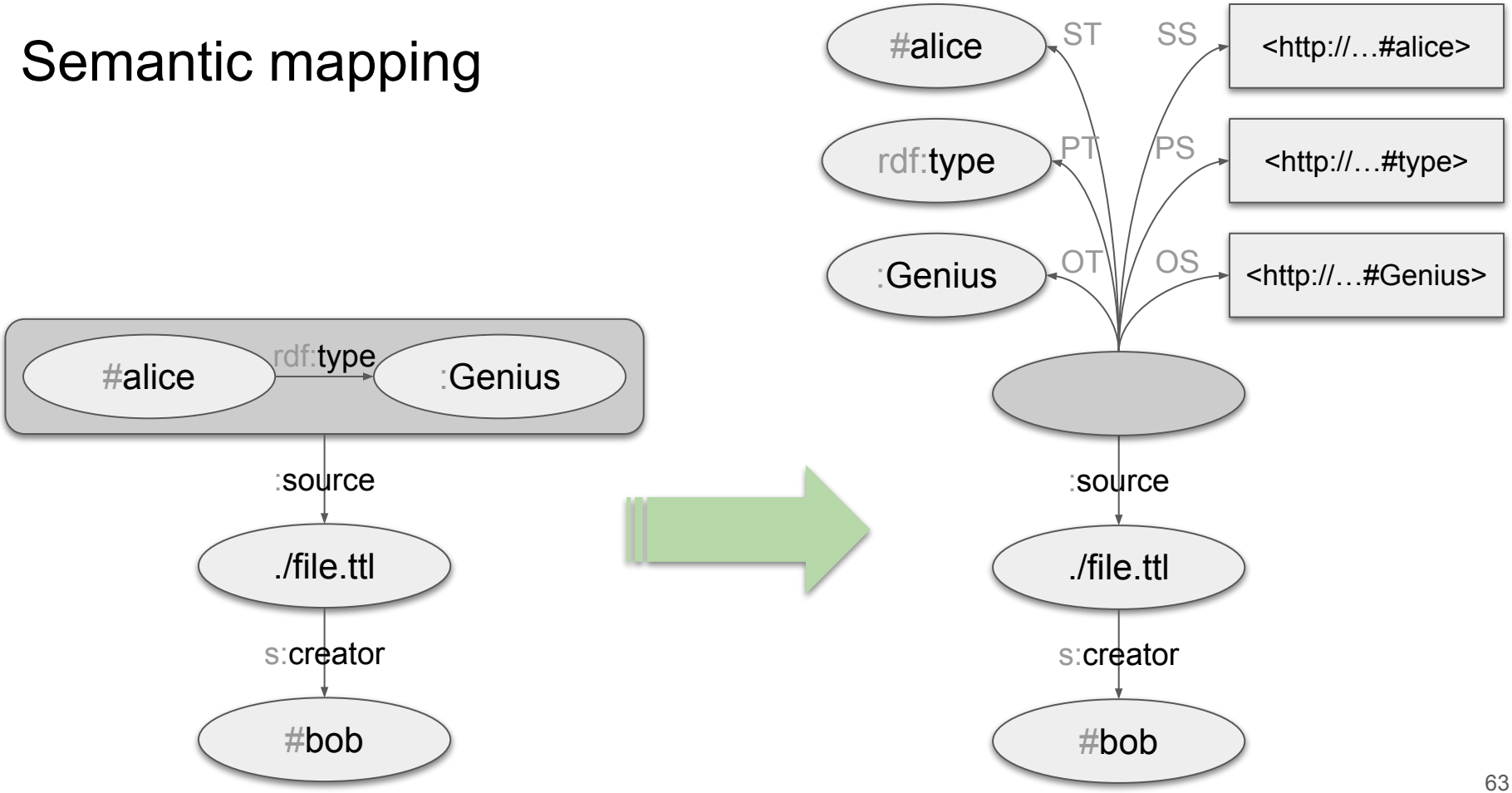
Swedish Research Council

JSON-LD-star: a more complex example




```
{ “@context”: “...”,
  “id”: “#car123”,
  “driver”: { “id”: “#alice”,
    “metadata”: {
      “containedBy”: “acme:staff.html”
  } },
  “owner”: { “id”: “#acme”,
    “metadata”: {
      “since”: “2019-06-01”
  } },
  “parkedIn”: {
    “metadata”: {
      “observedAt”: “2019-10-03T12:34:56Z”
    },
    “longitude”: 4.86553,
    “latutude”: 45.78234
} } }
```

Semantic mapping



Semantic mapping

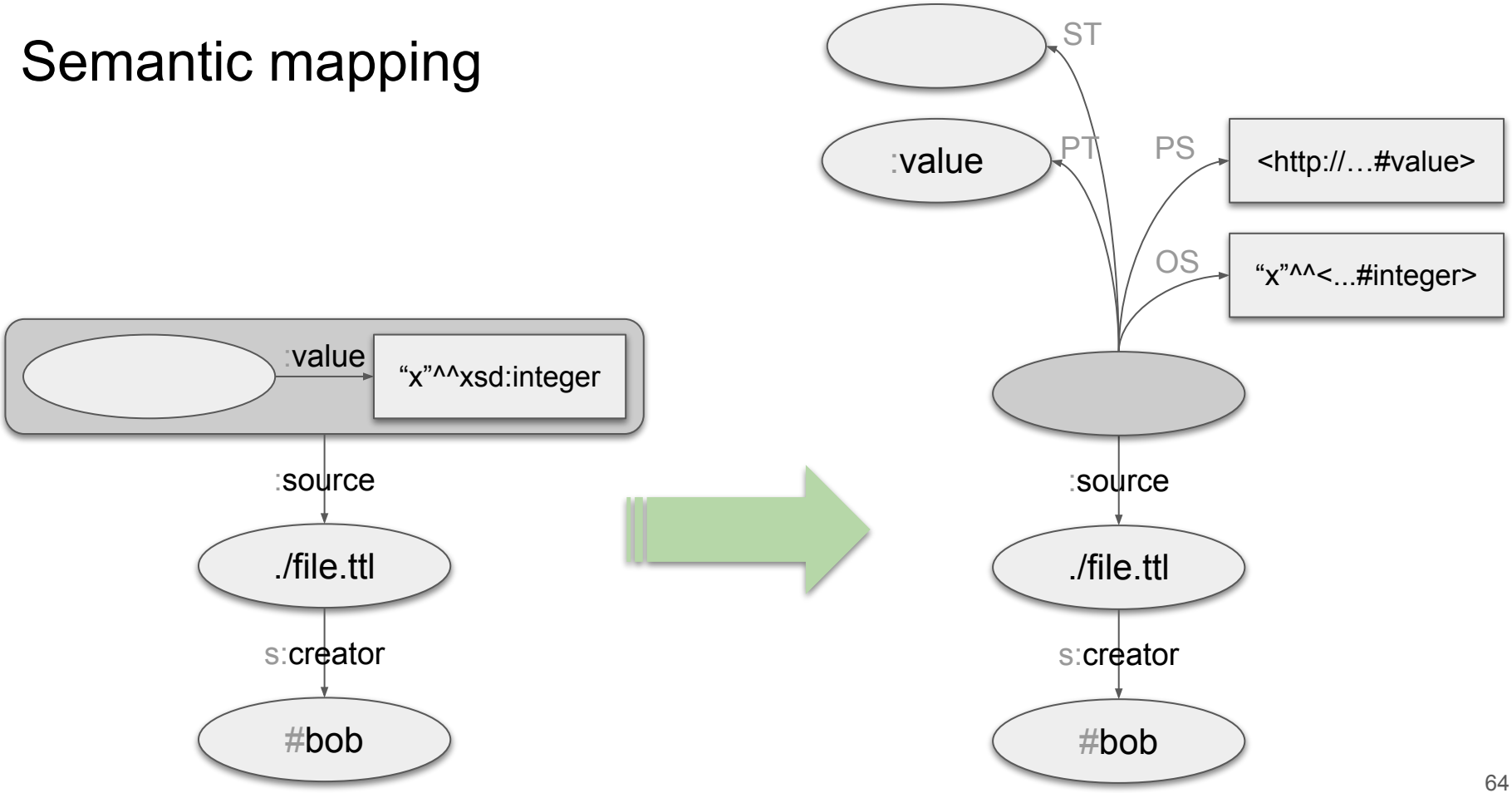


Table of Content