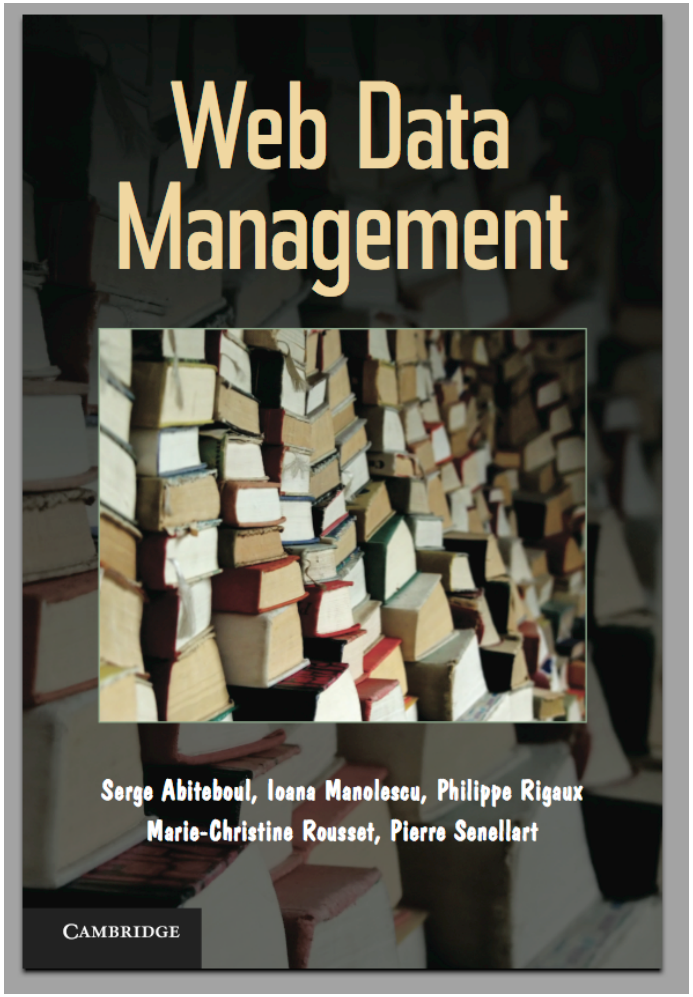
Querying the Semantic Web

Federico Ulliana

Slides partially collected from many sources on the Web.

Readings

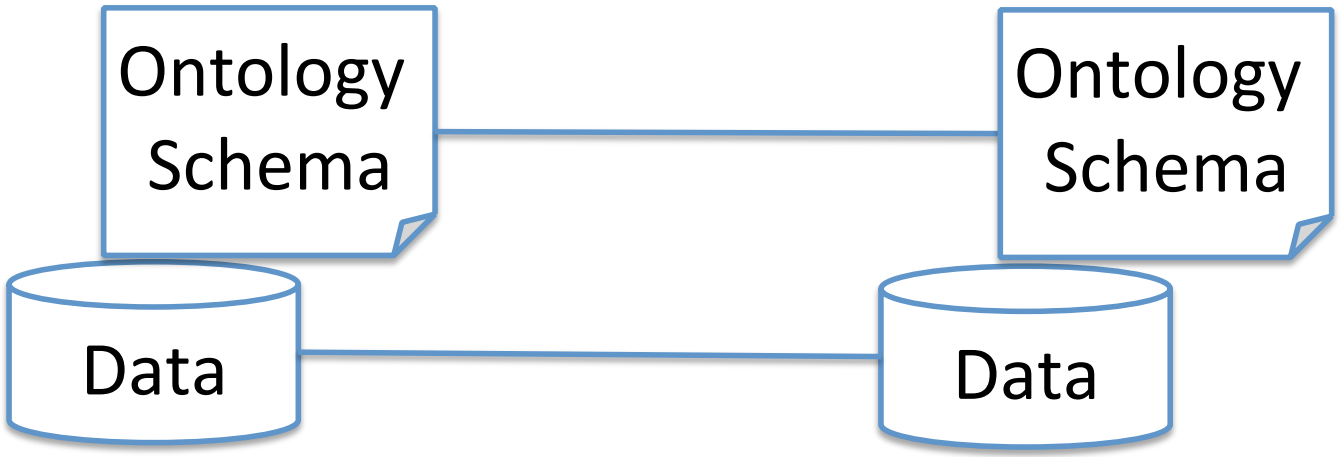


webdam.inria.fr/Jorge/

Semantic Web

http://www-sop.inria.fr/members/Fabien.Gandon/docs/semweb/#/

A web of ***linked data*** and ***linked schemas***



#1 – Open Data

publish data on the web

```
The number of graduates per year
    at Montpellier University
```

#2 – Web Data

web standards to identify things (URIs) and describe them (RDF)

```
http://myOpenData.org/Universities/UM
```

#3 – Linked Data

reuse and link web identifiers

```
http://myOpenData.org/Universities/UM

                rdf:sameAs

   http://fr.dbpedia.org/resource/
      Université_de_Montpellier
```

#4 – Open Schemas

publish knowledge as ontology schemas

```
A university is a Public Institution

Every university has a Dean
```

#5 – Web Schemas

schemas are web data

http://dbpedia.org/ontology/university

rdfs:subClassOf

http://dbpedia.org/ontology/educational_institution

#6 – Linked Schemas

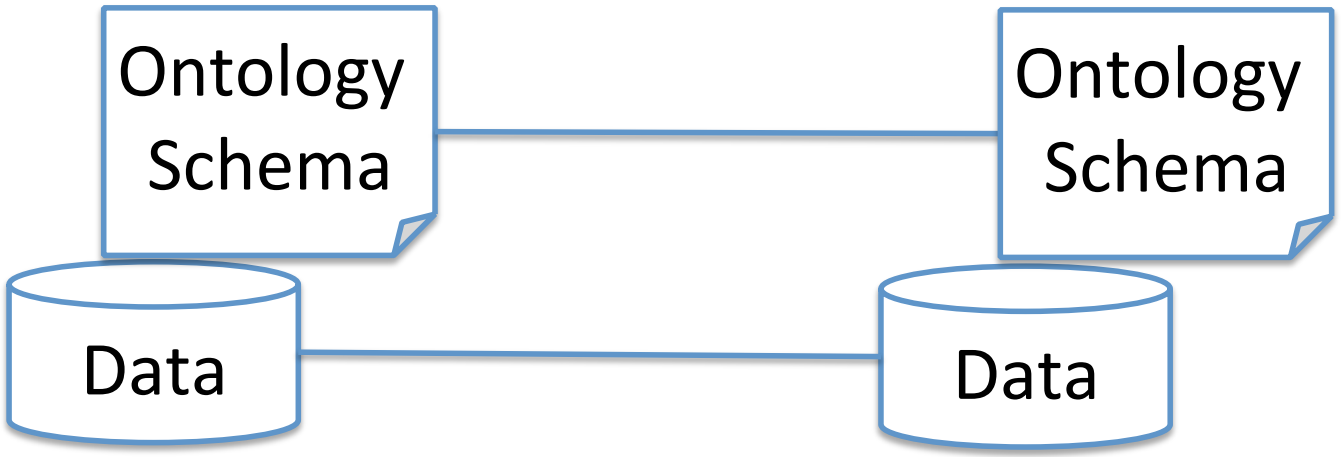
reuse and link to other schemas

```
http://www.freebase.com/education/
university

rdfs:sameAs

http://dbpedia.org/ontology/university
```

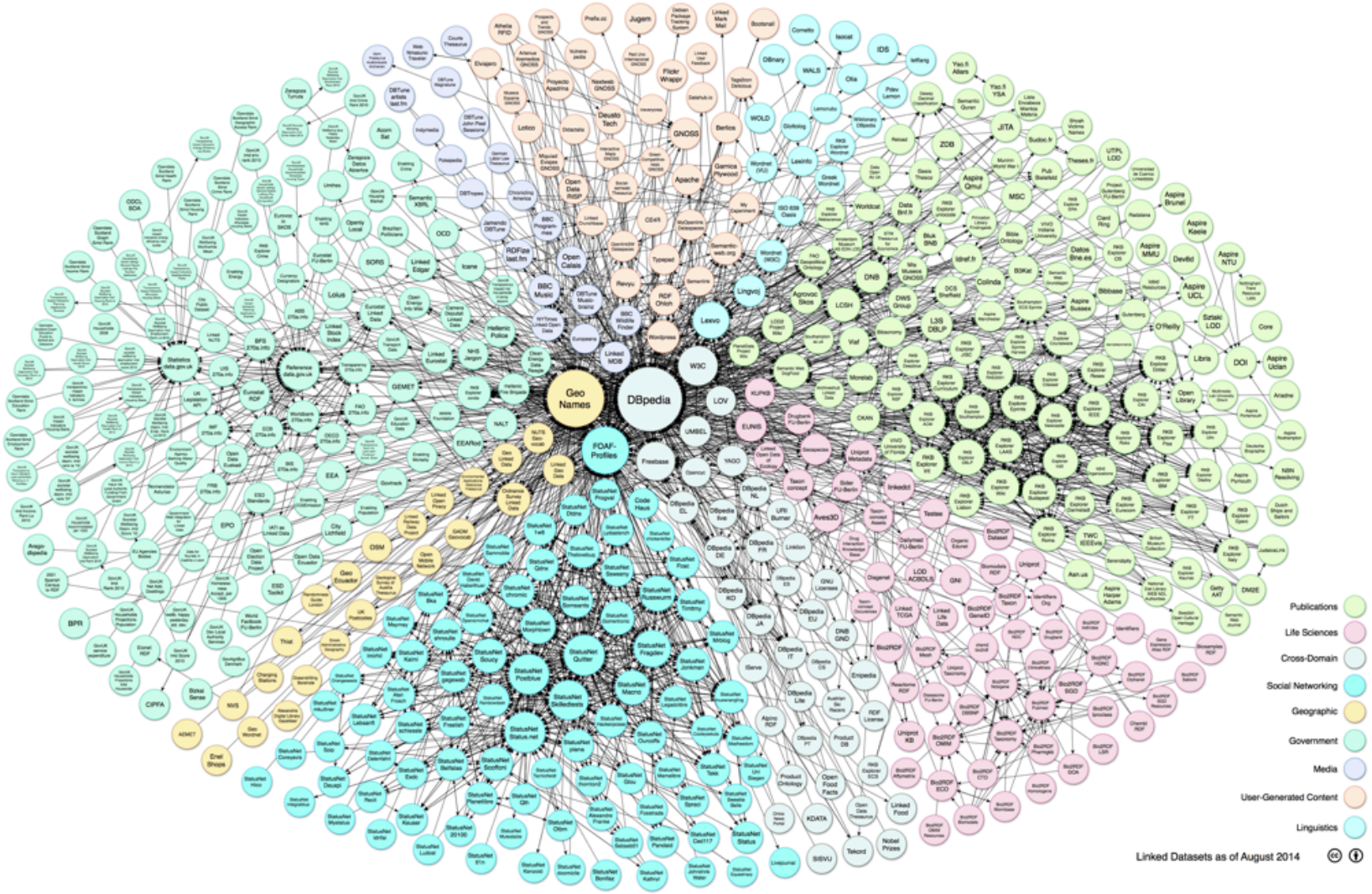
Semantic Web

http://www-sop.inria.fr/members/Fabien.Gandon/docs/semweb/#/

A web of ***linked data*** and ***linked schemas***



Linked-Data Cloud (august 2014)



Linked-Data Cloud (august 2014)

Datasets by topical domain.

Topic	Datasets	%
Government	183	18.05%
Publications	96	9.47%
Life sciences	83	8.19%
User-generated content	48	4.73%
Cross-domain	41	4.04%
Media	22	2.17%
Geographic	21	2.07%
Social web	520	51.28%
Total	1014	

The challenge

Provide (transparent) access to data (at web-scale) whilst dealing with

- linked data (#3)
- linked ontology schemas (#6)

DATA INTEGRATION
BASED ON
SEMANTIC WEB TECHNOLOGIES
(NOT ONLY WEB DATA)

2017

- The Linked Data is still too heterogeneus (or complementary) to generate profitable economic « industrial » value

- The main actors in 2017 are ***public institutions***, they produce open data and shared knowledge
  - The biggest driver is the Life Science domain

- The Semantic Web remains a long term vision of a nicely indexed and better queriable Web

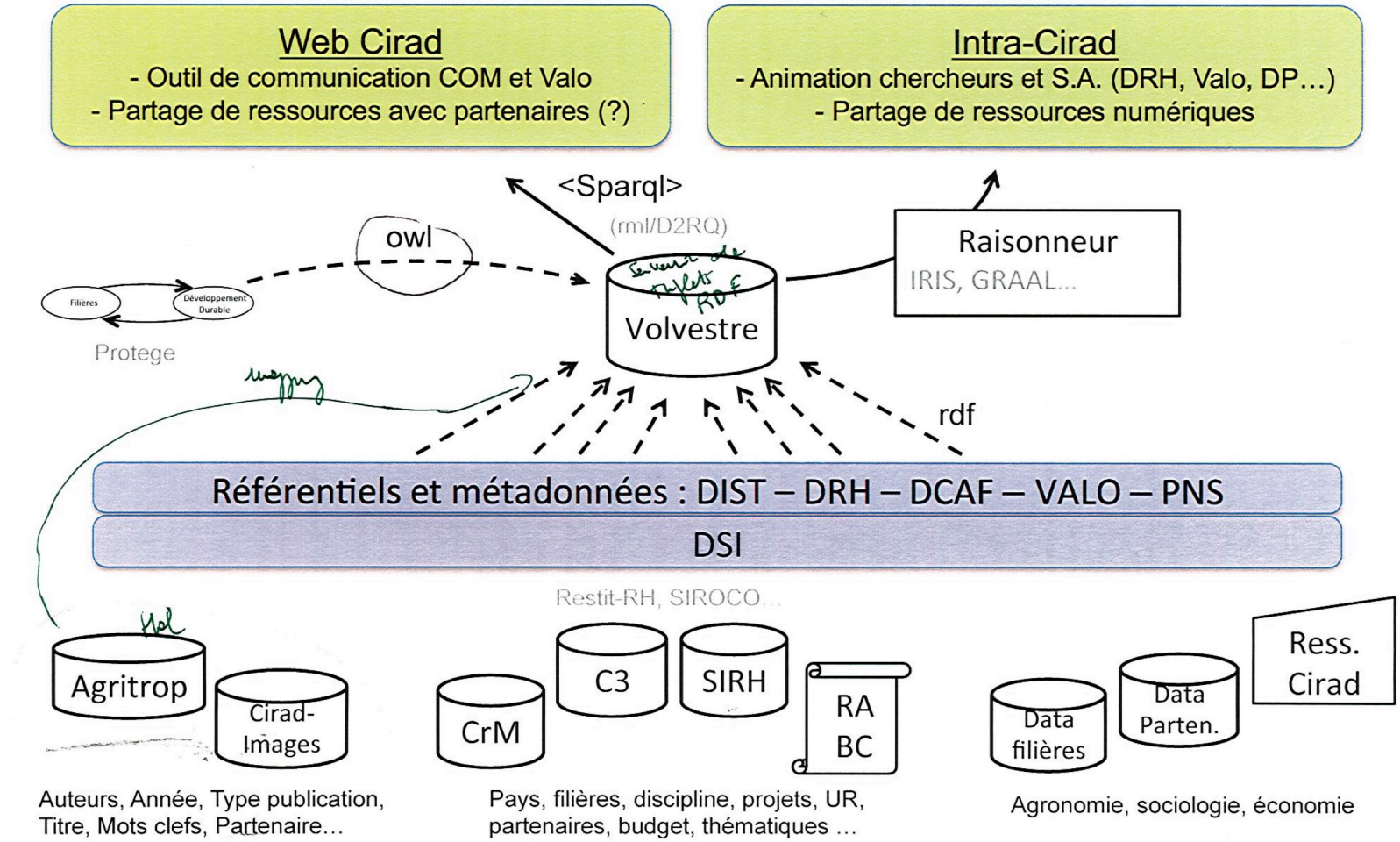
But ... SW techologies apply to any data!

- Reality : most of the SW projects today are run by a *small* group of institutions (often one)
- SW projects start because many independent databases have been accumulated over time, and there is a need for making them cooperate
- Data **integration**, data **interoperabilty**, data **quality**, is perhaps the current largest market for SW technologies.

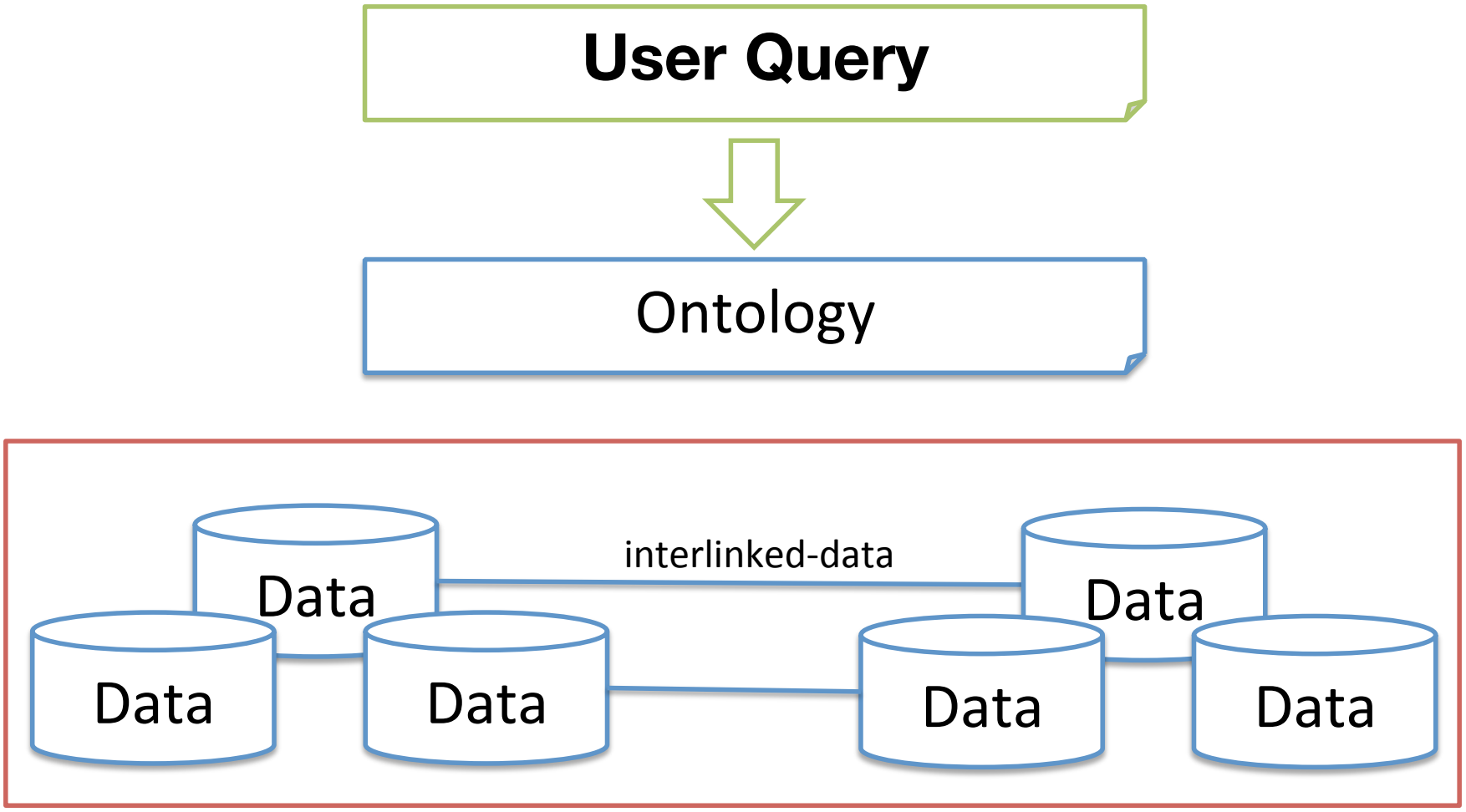
Système à Base de Connaissances du CIRAD

http://www.cirad.fr/

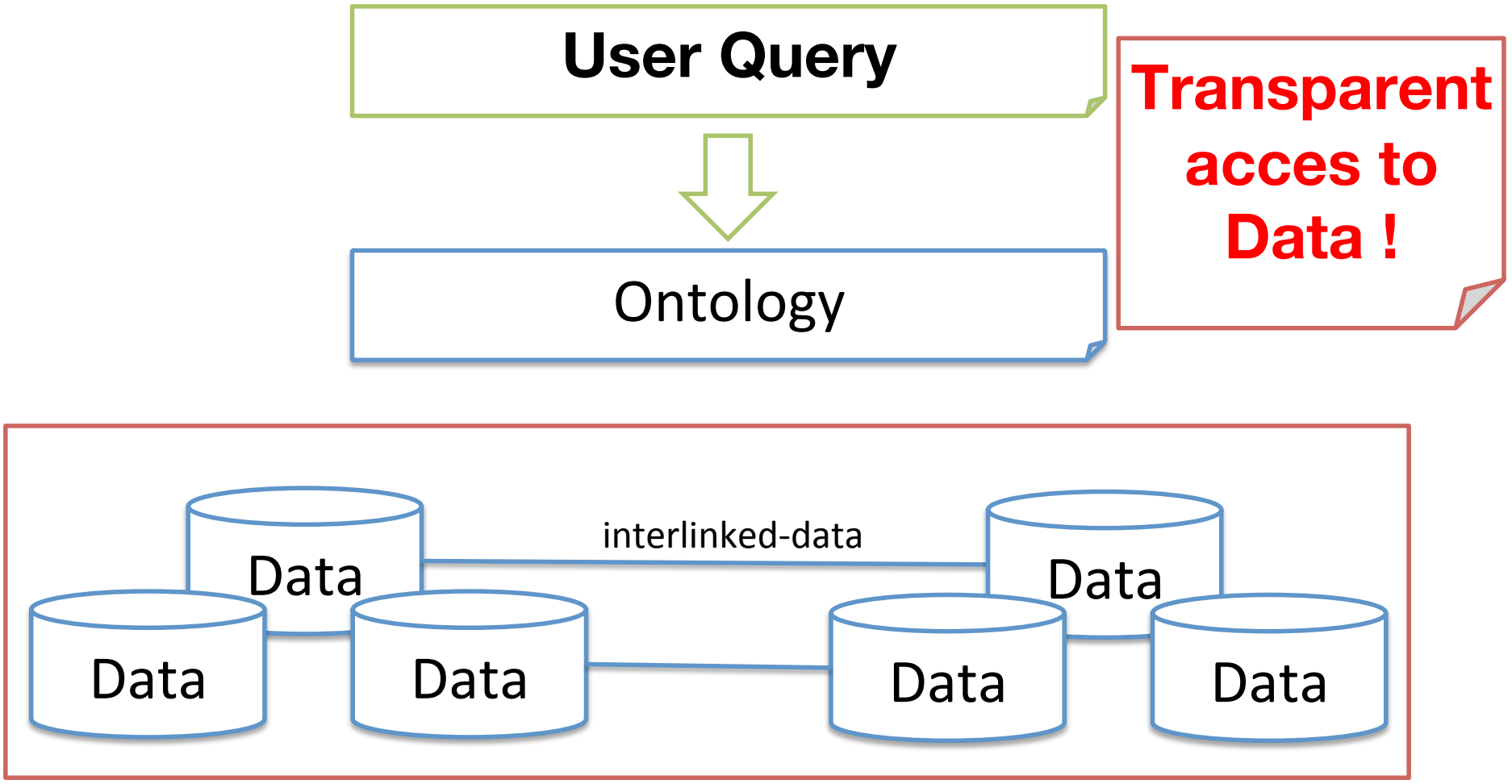
Fonctionnement du SBC



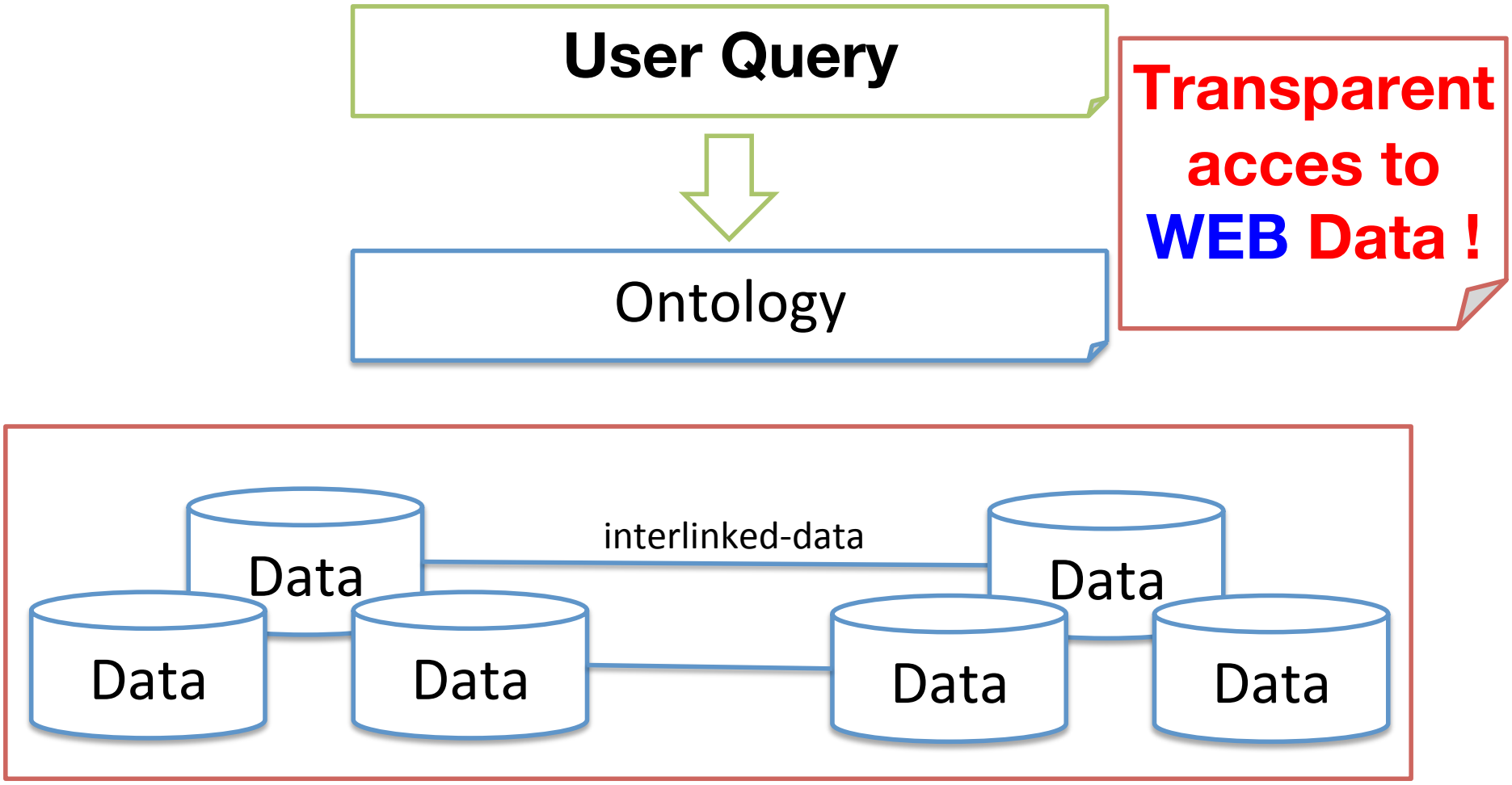
Data Integration



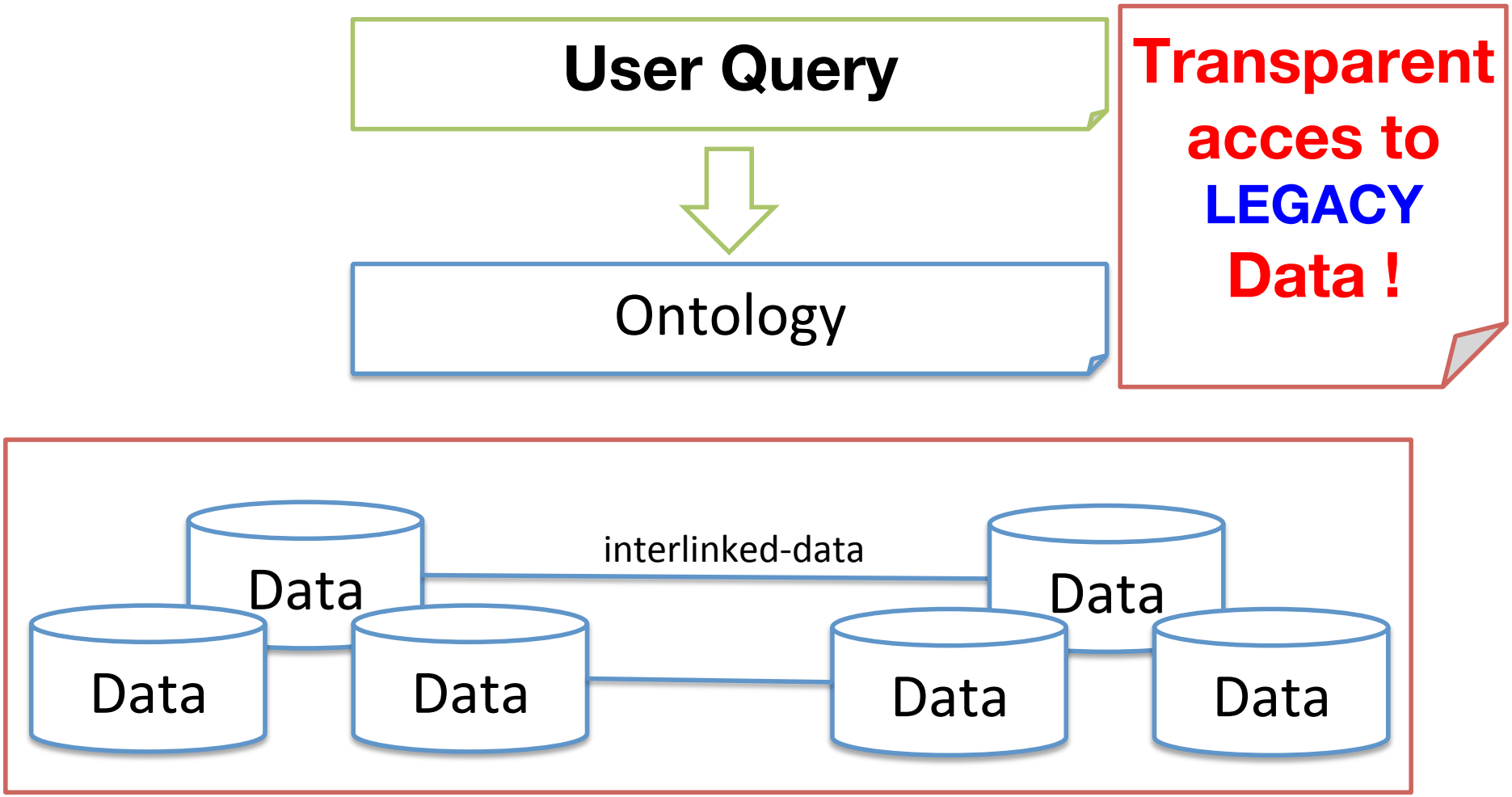
Data Integration



Data Integration



Data Integration



Data Integration with SW Tech

- A superficial understanding of the standards often results in a bad data integration system
- You need to know data-linking and ontology-alignment to better query a set of federated databases
- You need to know the RDF data-model in detail, as well as its query language SPARQL
- You need to know the impact of ontologies, and the balance between expressivity and efficiency

Semantic Web Seminars (3 Mai)

- Yann Nicolas (ABES) - Qualité et l'interopérabilité de données dans les grands catalogues documentaires
- Patrice Buche (INRA) – Un système d'aide à la décision en agronomie alimenté par des données scientifiques hétérogènes
- Konstantin Todorov (UM) - Données en réutilisation dans le domaine de la musique en fonction des usages
- Federico Ulliana (UM) – Annotation sémantique fonctionnelle de modèles CAO 3D

BACK TO RDF QUERYING : WHAT MAKES SEMANTIC WEB DATA SO SPECIAL ?

1 - Semi-structured Data

RDF Data **= set of triples**

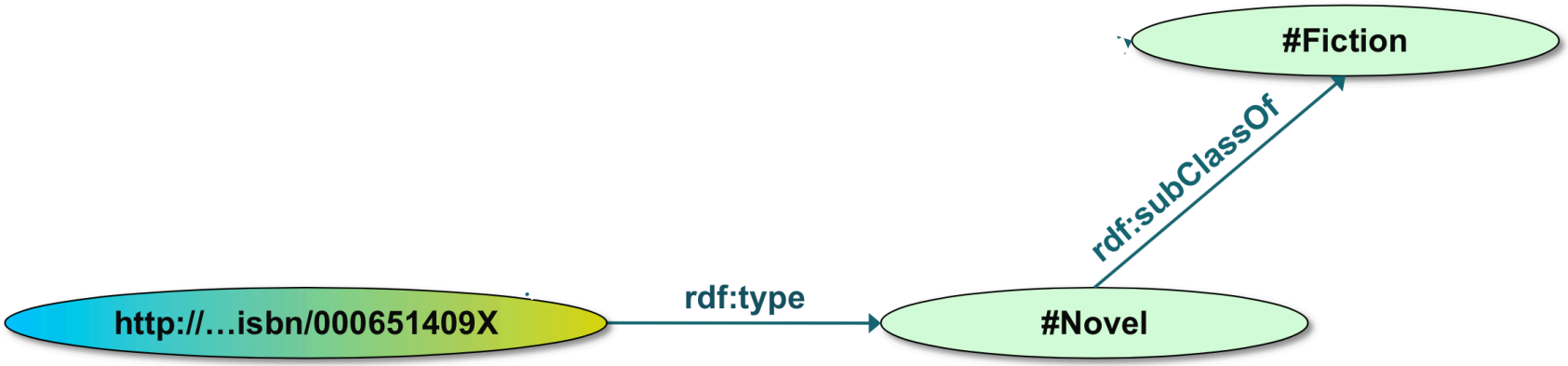
No “initialization”, no need to define tables

Data-structure unknown a priori! (high flexibility)

Data-model for “generalized graphs” (see later)

2 - Ontologies

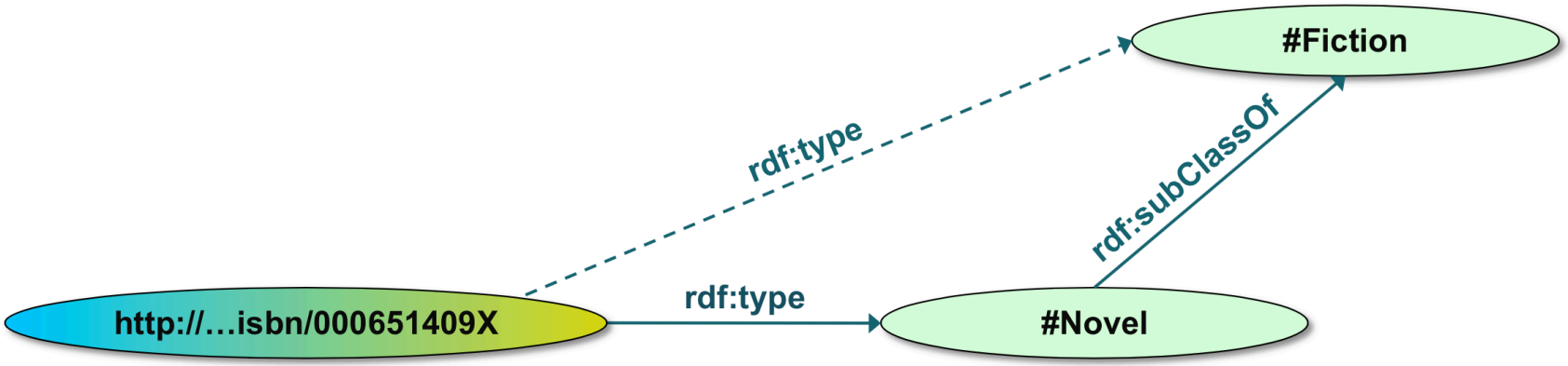
Complete query answering relies on reasoning



Here the Linked-Data becomes the Semantic Web

2 - Ontologies

Complete query answering relies on reasoning

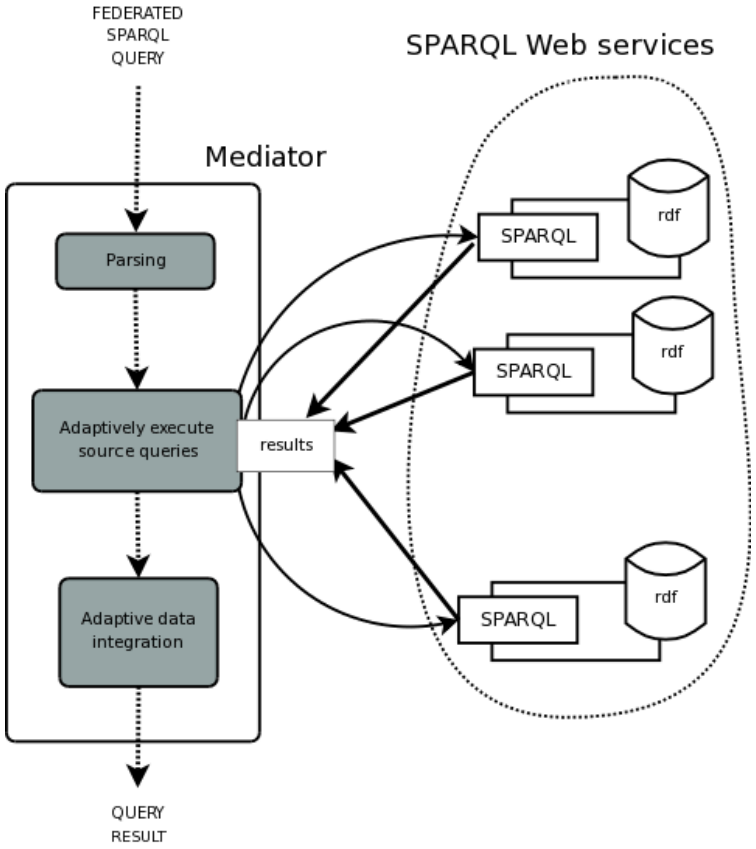


Here the Linked-Data becomes the Semantic Web

3 - HTTP-based Data Access

Endpoints

Federated-systems



The Query Answering Problem

Input

- Data
- Query

Output

- Answer set

COMPARING DATA-MODELS : RELATIONAL, GRAPH, AND RDF.

RDF and Graphs

- Very often we hear that RDF is a language for graphs.

- Is this claim accurate ? And what is a language for graphs anyways ?

Datamodels

Relational Datamodel

Consider the relational schema with a single relation

`Person(name,age,city)`

How to draw a relational database ?

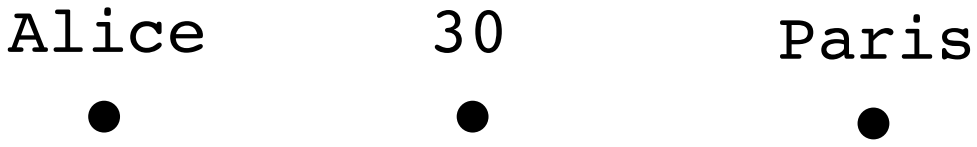
Person

Alice	30	Paris
Bob	42	Paris

How to draw a relational database ?

Person

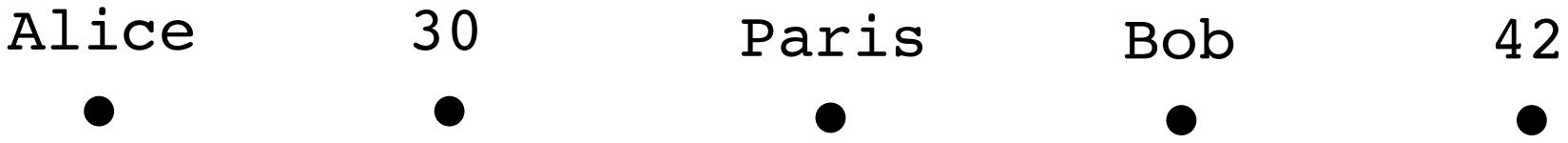
Alice	30	Paris
Bob	42	Paris



How to draw a relational database ?

Person

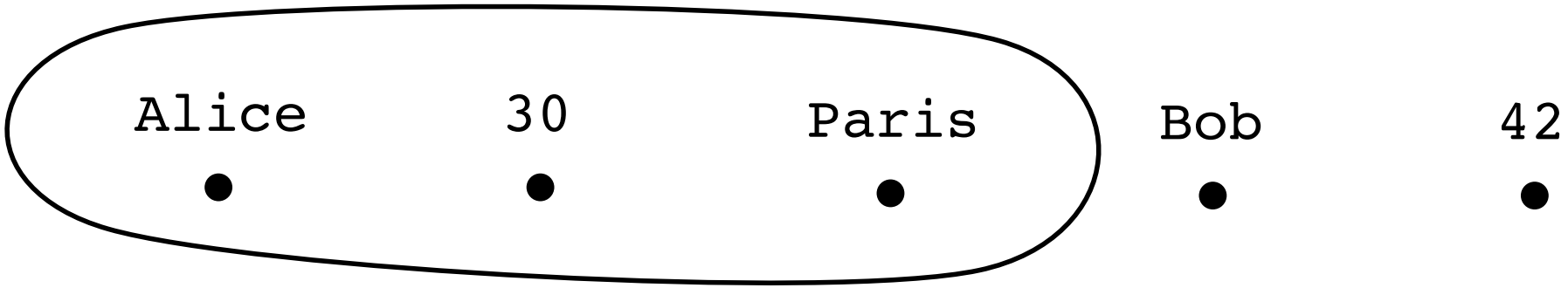
Alice	30	Paris
Bob	42	Paris



How to draw a relational database ?

Person

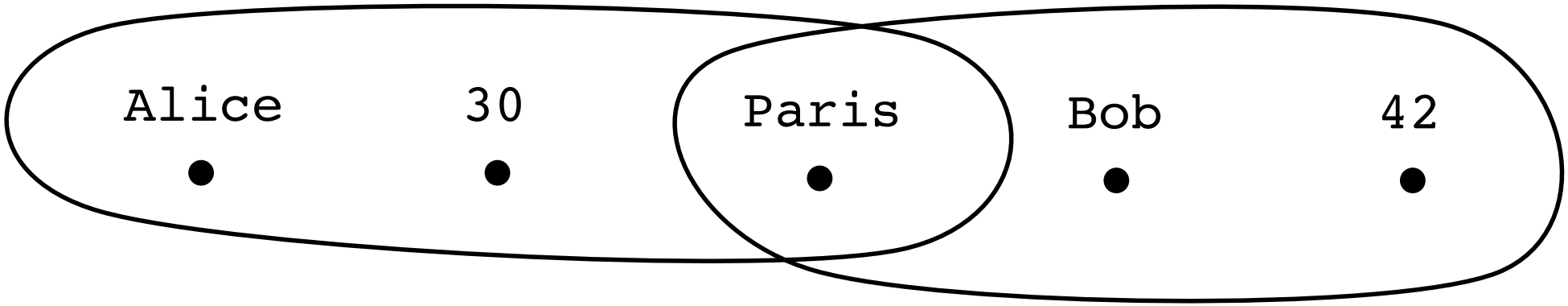
Alice	30	Paris
Bob	42	Paris



How to draw a relational database ?

Person

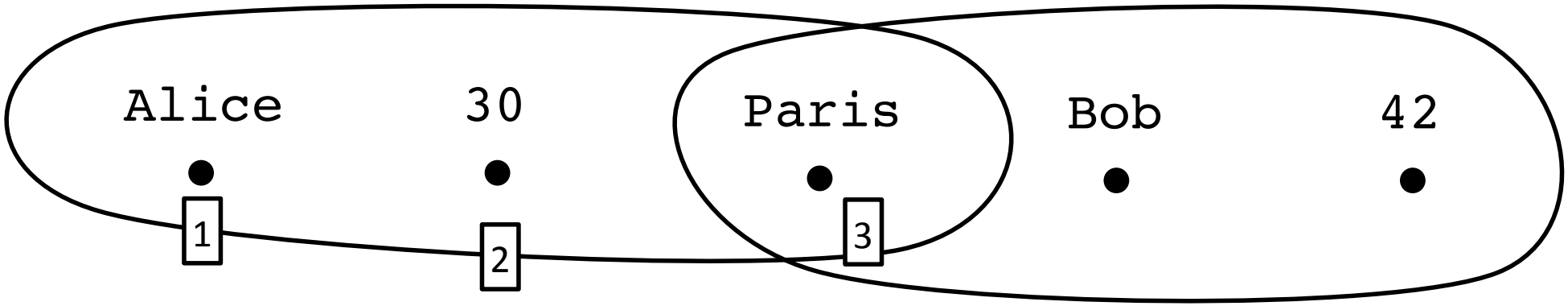
Alice	30	Paris
Bob	42	Paris



How to draw a relational database ?

Person

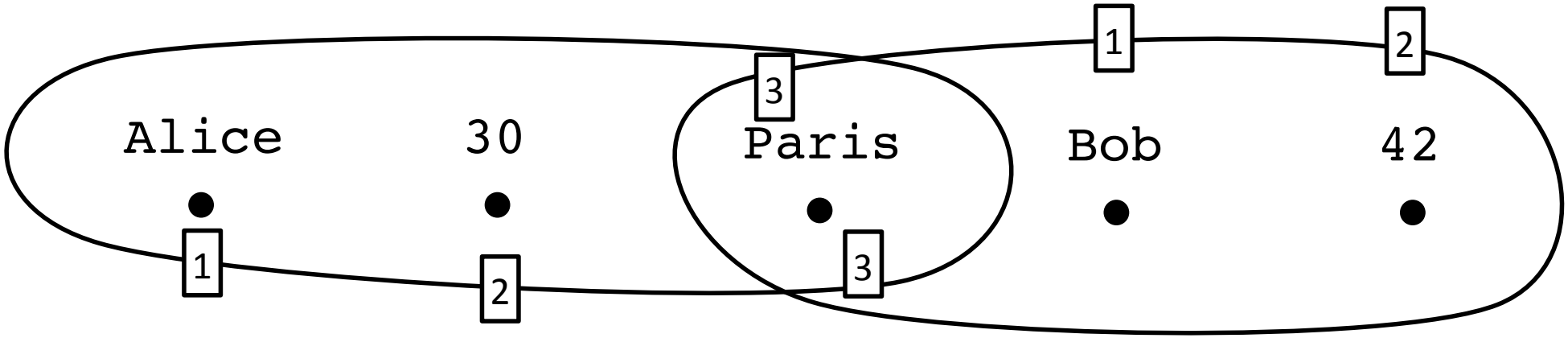
Alice	30	Paris
Bob	42	Paris



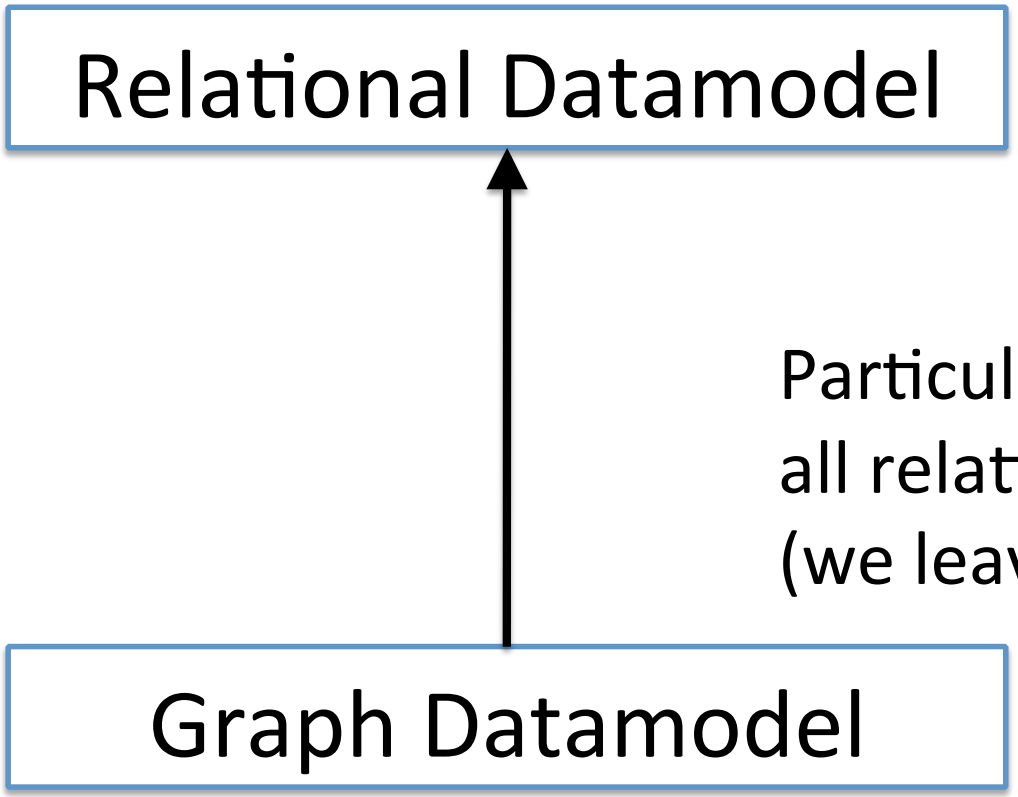
How to draw a relational database ?

Person

Alice	30	Paris
Bob	42	Paris



Datamodels



Particular case, where all relations are **binary** (we leave **unary** aside).

From n-ary to 2-ary

hasAge(name,age)

Alice	30
Bob	42

livesIn(name,city)

Alice	Paris
Bob	Paris

Person(name,age,city)

Alice	30	Paris
Bob	42	Paris

How to draw a graph database ?

`hasAge`

Alice	30
Bob	42

`livesIn`

Alice	Paris
Bob	Paris

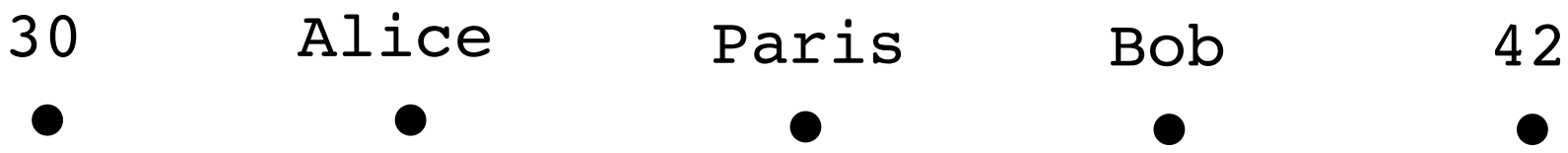
How to draw a graph database ?

hasAge

Alice	30
Bob	42

livesIn

Alice	Paris
Bob	Paris



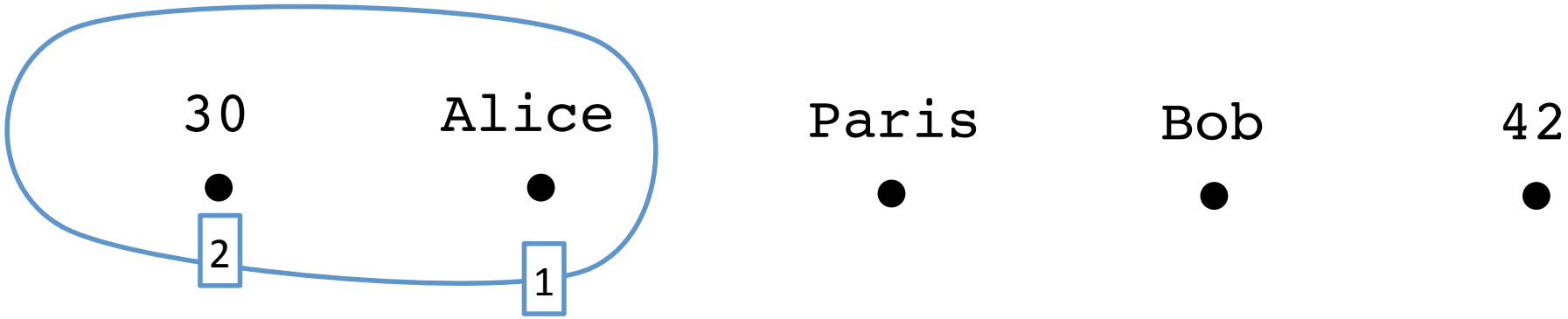
How to draw a graph database ?

hasAge

Alice	30
Bob	42

livesIn

Alice	Paris
Bob	Paris



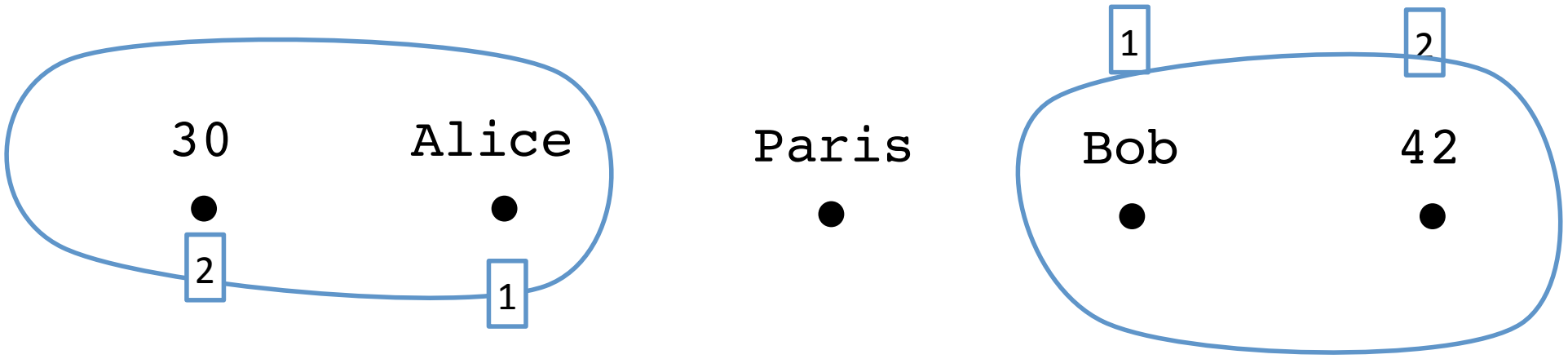
How to draw a graph database ?

hasAge

Alice	30
Bob	42

livesIn

Alice	Paris
Bob	Paris



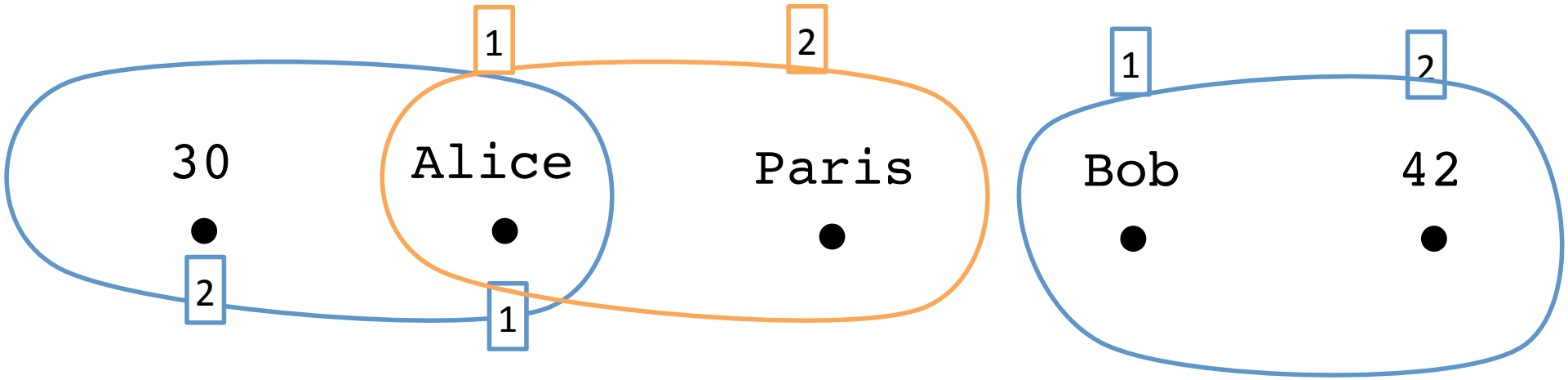
How to draw a graph database ?

hasAge

Alice	30
Bob	42

livesIn

Alice	Paris
Bob	Paris



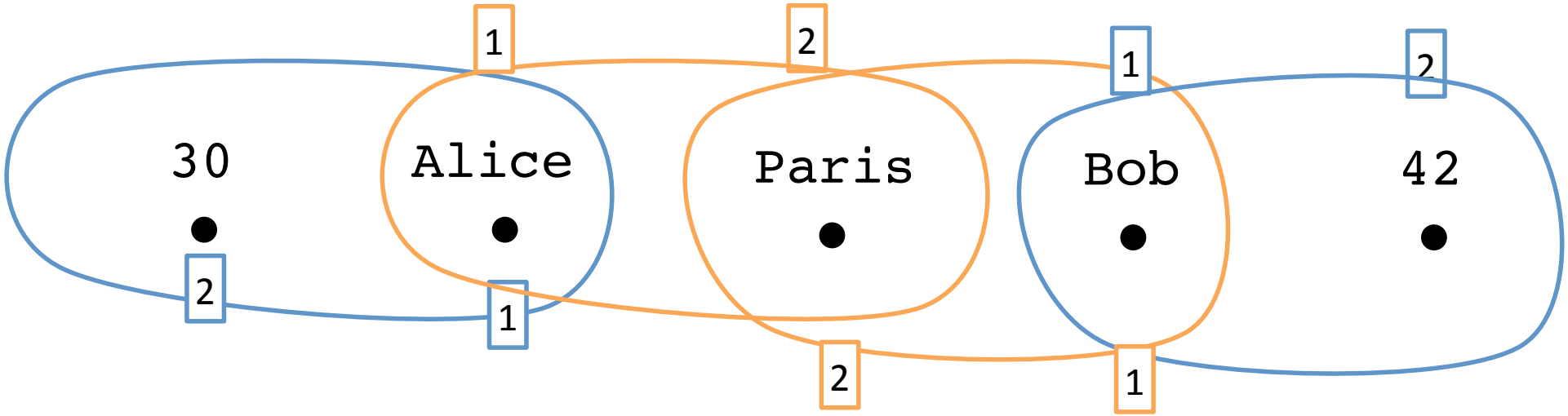
How to draw a graph database ?

hasAge

Alice	30
Bob	42

livesIn

Alice	Paris
Bob	Paris



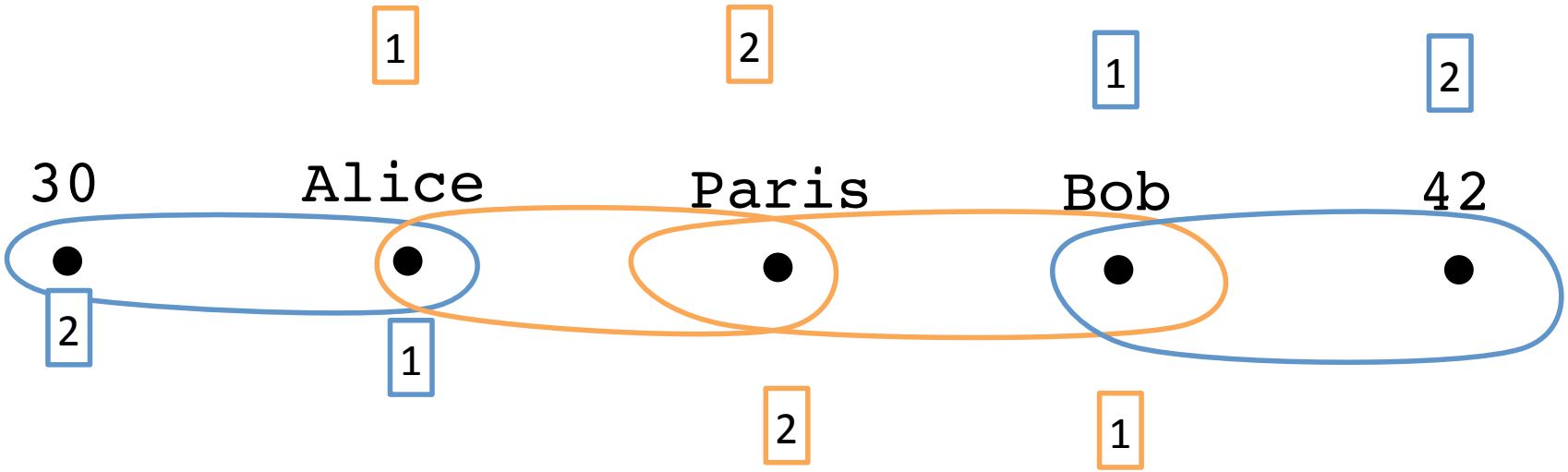
How to draw a graph database ?

hasAge

Alice	30
Bob	42

livesIn

Alice	Paris
Bob	Paris



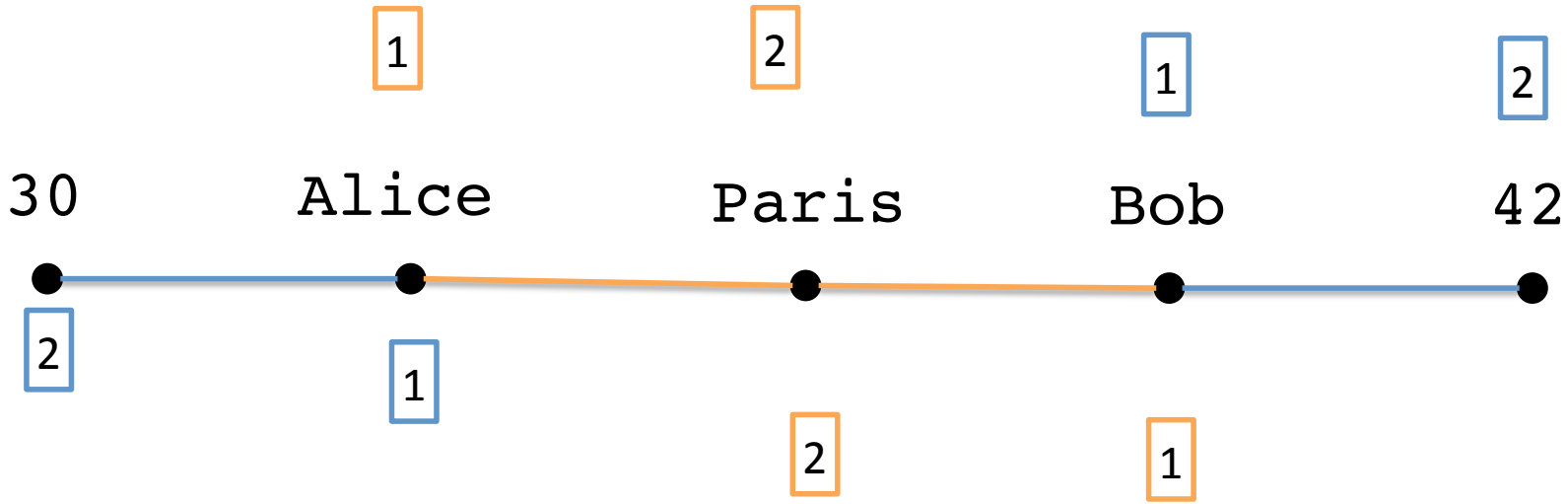
How to draw a graph database ?

hasAge

Alice	30
Bob	42

livesIn

Alice	Paris
Bob	Paris



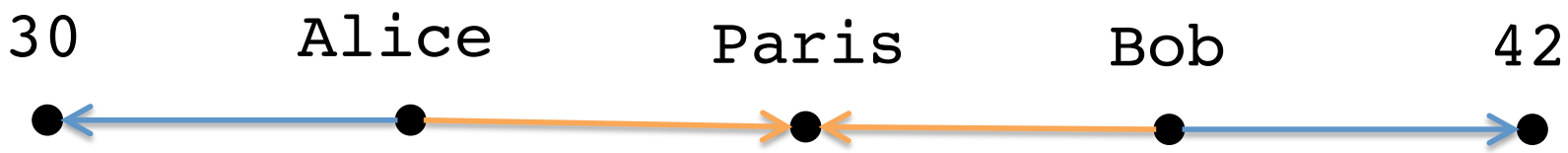
How to draw a graph database ?

hasAge

Alice	30
Bob	42

livesIn

Alice	Paris
Bob	Paris



How to query a relational database

Person

Alice	30	Paris
Bob	42	Paris

```
SELECT name, age, city
FROM   Person
```

How to query a graph database

hasAge

Alice	30
Bob	42

livesIn

Alice	Paris
Bob	Paris

```
SELECT  hasAge.*, livesIn.city
FROM    hasAge, livesIn
WHERE hasAge.name = livesIn.name
```

How to query a graph database

hasAge

Alice	30
Bob	42

livesIn

Alice	Paris
Bob	Paris

```
SELECT  hasAge.*, livesIn.city
FROM    hasAge, livesIn
WHERE hasAge.name = livesIn.name
```

N-ary rel. decomposed to 2-ary → expect lots of joins

Let's change de syntax (1)! (but not the semantics)

hasAge

Alice	30
Bob	42

livesIn

Alice	Paris
Bob	Paris

avoid to pre-list all relations

```
SELECT  hasAge.*, livesIn.city
FROM    hasAge, livesIn
WHERE   hasAge.name = livesIn.name
```

N-ary rel. decomposed to 2-ary → expect lots of joins

Let's change de syntax (2)! (but not the semantics)

hasAge

Alice	30
Bob	42

livesIn

Alice	Paris
Bob	Paris

avoid to pre-list all relations

```
SELECT  hasAge.*, livesIn.city
FROM    hasAge, livesIn
WHERE   hasAge.name = livesIn.name
```

avoid big list of joins

N-ary rel. decomposed to 2-ary → expect lots of joins

Let's change de syntax (3)! (but not the semantics)

hasAge

Alice	30
Bob	42

livesIn

Alice	Paris
Bob	Paris

selected attributes become variables

avoid to pre-list all relations

```
hasAge.*, livesIn.city
FROM      hasAge, livesIn
WHERE     hasAge.name = livesIn.name
```

avoid big list of joins

N-ary rel. decomposed to 2-ary → expect lots of joins

Let's change de syntax (3)! (but not the semantics)

hasAge

Alice	30
Bob	42

livesIn

Alice	Paris
Bob	Paris

```
SELECT  ?name, ?age, ?city

WHERE { ?name hasAge ?age  .
        ?name livesIn ?city  . }
```

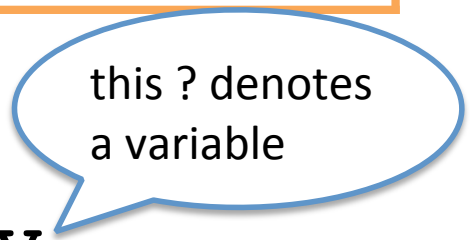
Let's change de syntax (3)! (but not the semantics)

hasAge

Alice	30
Bob	42

livesIn

Alice	Paris
Bob	Paris




```
SELECT  ?name, ?age, ?city

WHERE { ?name hasAge ?age  .
        ?name livesIn ?city }
```

Let's change de syntax (3)! (but not the semantics)

hasAge

Alice	30
Bob	42

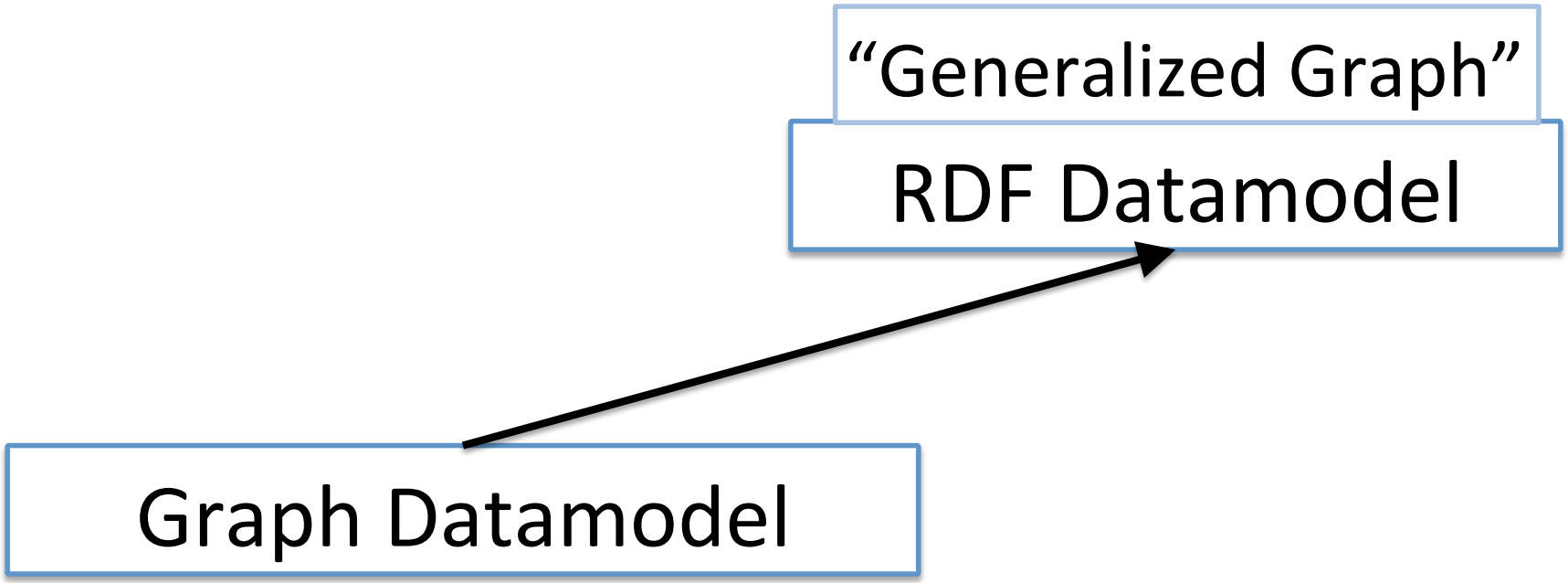
livesIn

Alice	Paris
Bob	Paris

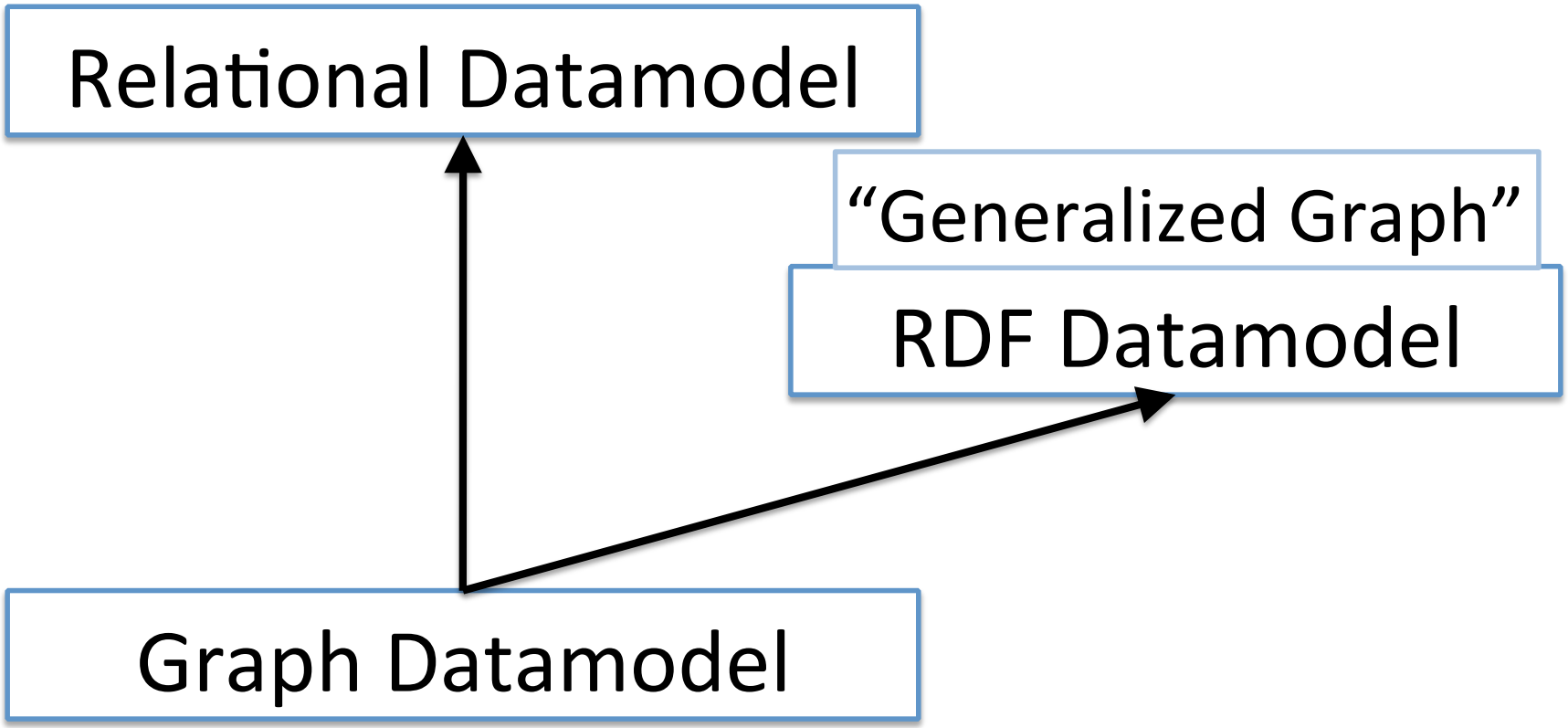
```
SELECT  ?name, ?age, ?city

WHERE { ?name hasAge ?age  .
        ?name livesIn ?city }
```

this dot stands for an AND

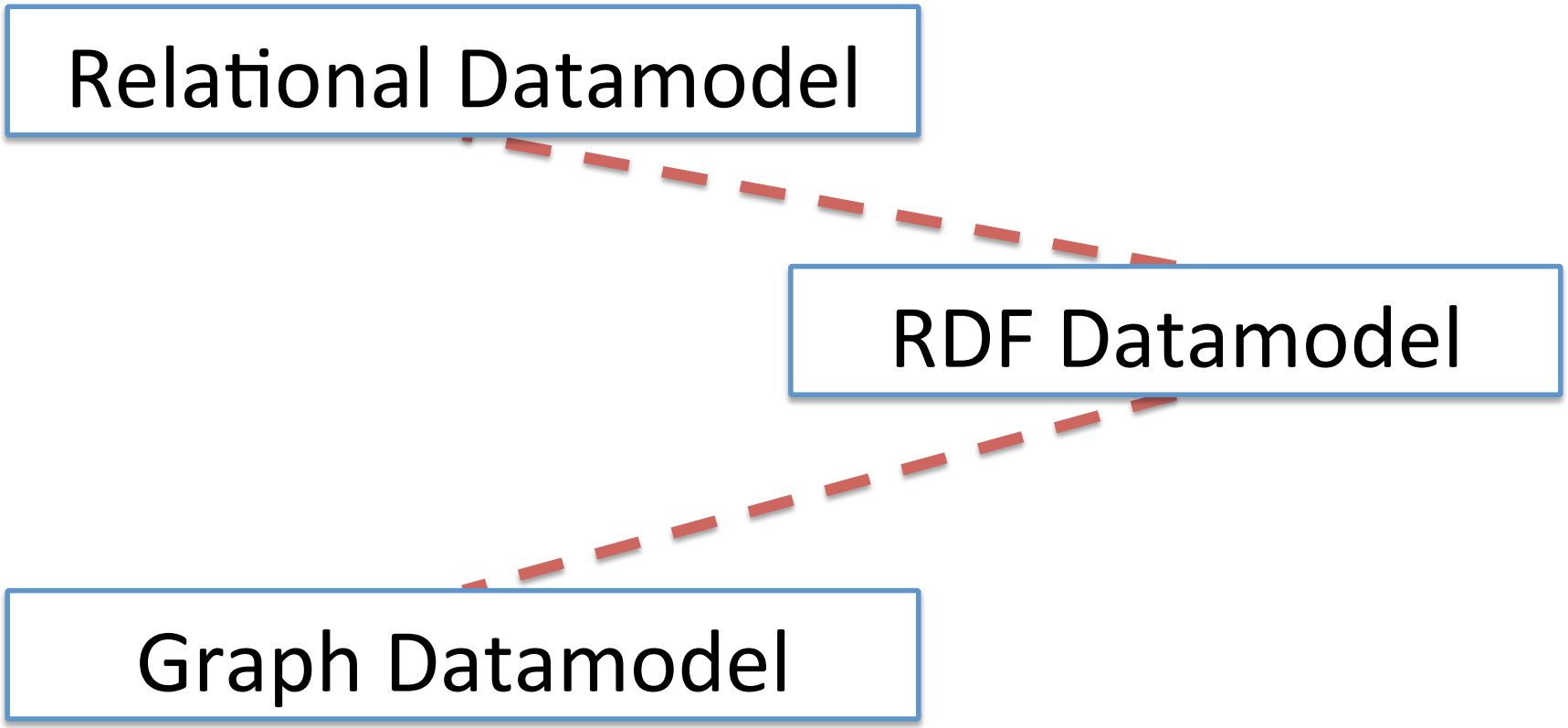
Datamodels



Graphs are retrocompatible with Relational & RDF, but…



… what about the other way around?



Answer : some (weird) RDF triples need a specific encoding to be expressed as graphs

RDF Datamodel

Set of triples `<s,p,o>` where

	URI	Literal	Blank Node
s	✓	✗	✓
p	✓	✗	✗
o	✓	✓	✓

and nothing else.

When drawing (admissible) triples may yield (weird) results

URI URI URI

< London, ACME_Transports, Paris >

< ACME_Transports, partOf,Eurostar>

URI URI URI

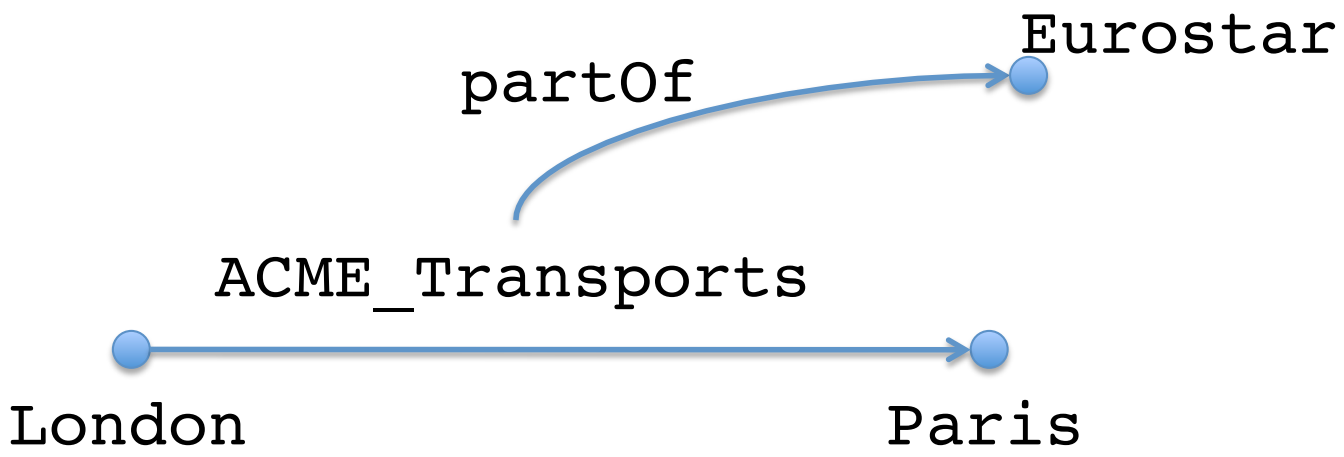
When drawing (admissible) triples may yield (weird) results

URI URI URI

< London, ACME_Transports, Paris >

< ACME_Transports, partOf,Eurostar>

URI URI URI



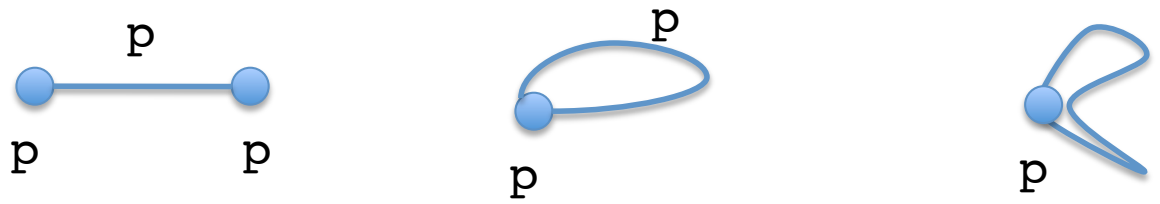
Example of RDF triples inexpressible in relational / graph datamodel :

- `ACME_Transports`
  must choose to be **either** a *relation-name* or a *value* in relational/graph models

- `ACME_Transports` is a value in RDF, which gives the “illusion” that it can be both.

Drawing triples may yield (weird) results

- Even worse : how to draw the RDF triple <p,p,p> ?

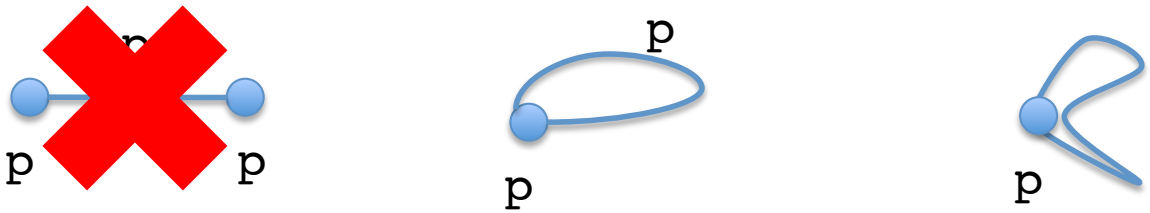


<rdf:type, rdf:type, rdf:type>

<rdfs:subClassOf, rdfs:subClassOf, rdfs:subClassOf>

Drawing triples may yield (weird) results

- Even worse : how to draw the RDF triple <p,p,p> ?

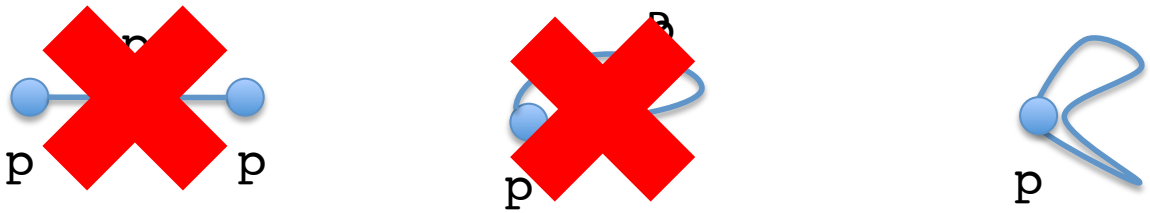


<rdf:type, rdf:type, rdf:type>

<rdfs:subClassOf, rdfs:subClassOf, rdfs:subClassOf>

Drawing triples may yield (weird) results

- Even worse : how to draw the RDF triple <p,p,p> ?



<rdf:type, rdf:type, rdf:type>

<rdfs:subClassOf, rdfs:subClassOf, rdfs:subClassOf>

Drawing triples may yield (weird) results

- Even worse : how to draw the RDF triple <p,p,p> ?

<rdf:type, rdf:type, rdf:type>

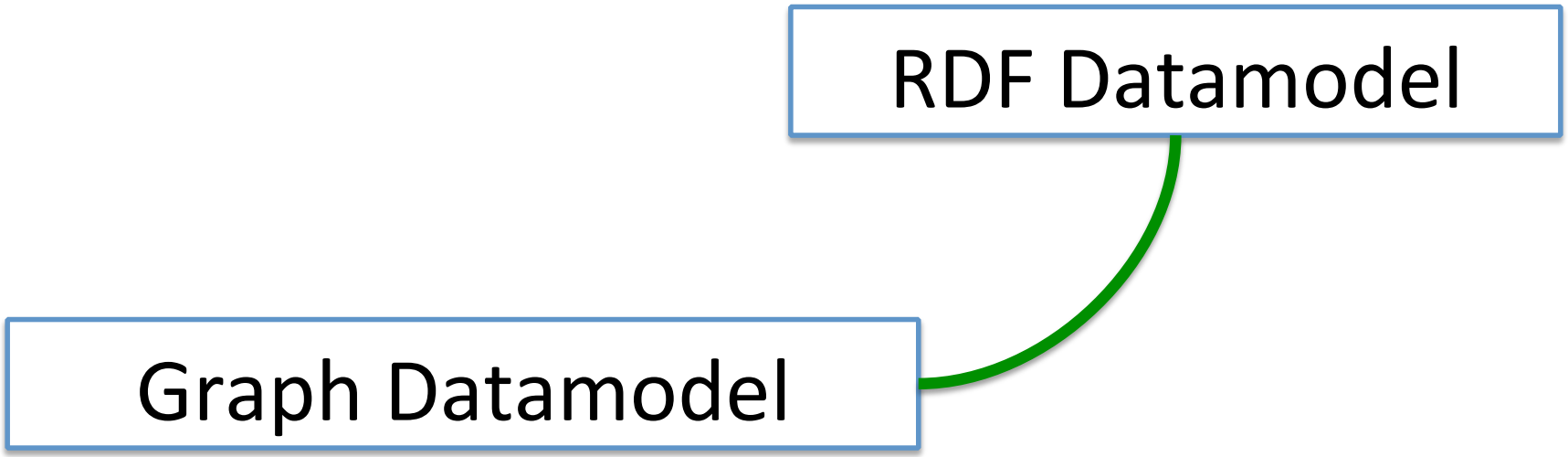
<rdfs:subClassOf, rdfs:subClassOf, rdfs:subClassOf>

Encodings to the rescue

- At least, we can always find an encoding of data that allow to travel between the data-models

- But remember : things may get tricky when defining query answering and (more importantly) reasoning

Encoding
RDF → GraphDatabase



Encoding
RDF → GraphDatabase

<s,p,o>

→

t_0 is a fresh constant standing for the triple

subject(t_0,s) **predicate**(t_0,p) **object**(t_0,o)

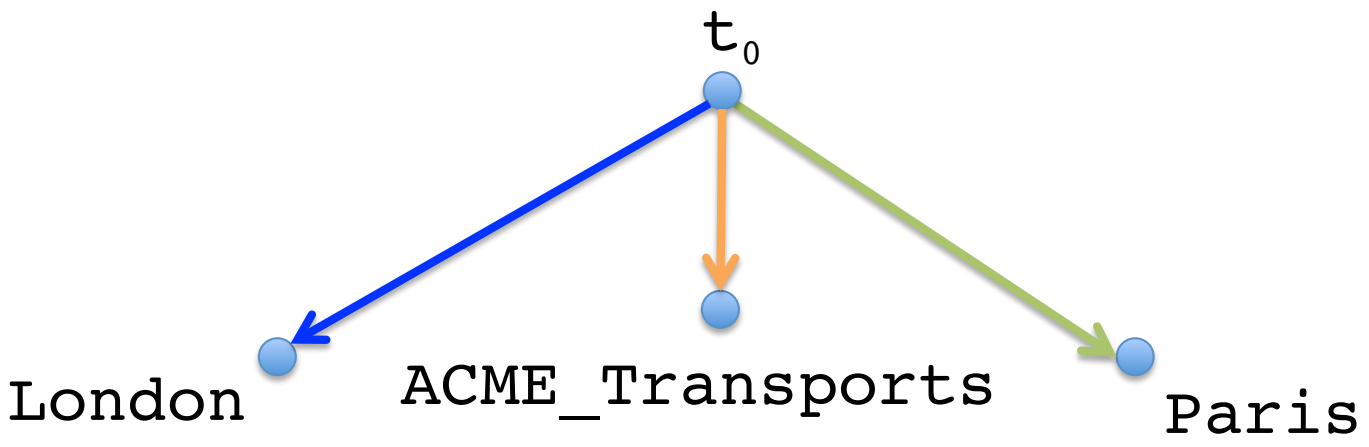
Encoding
RDF → GraphDatabase

<s,p,o>

→

t_0 is a fresh constant standing for the triple

subject(t_0,s) **predicate**(t_0,p) **object**(t_0,o)



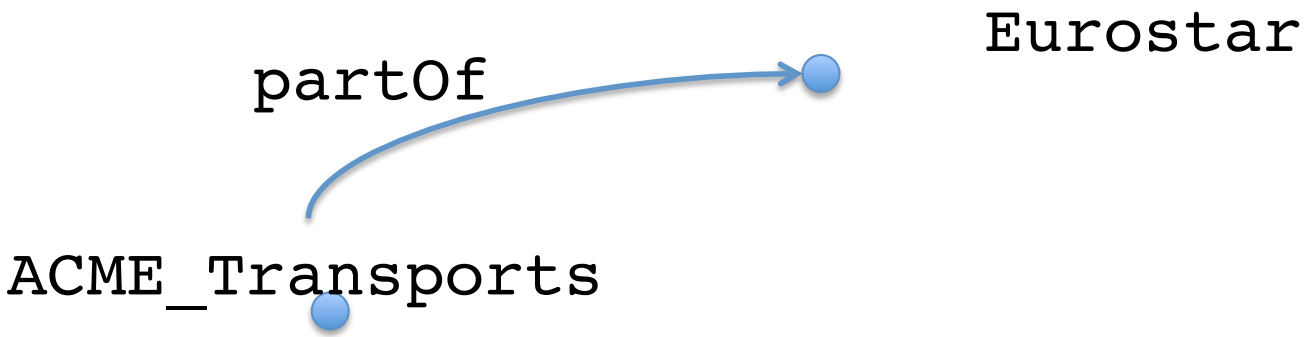
Encoding
RDF → GraphDatabase

<s,p,o>

→

t_1 is a fresh constant standing for the triple

subject(t_1,s) **predicate**(t_1,p) **object**(t_1,o)



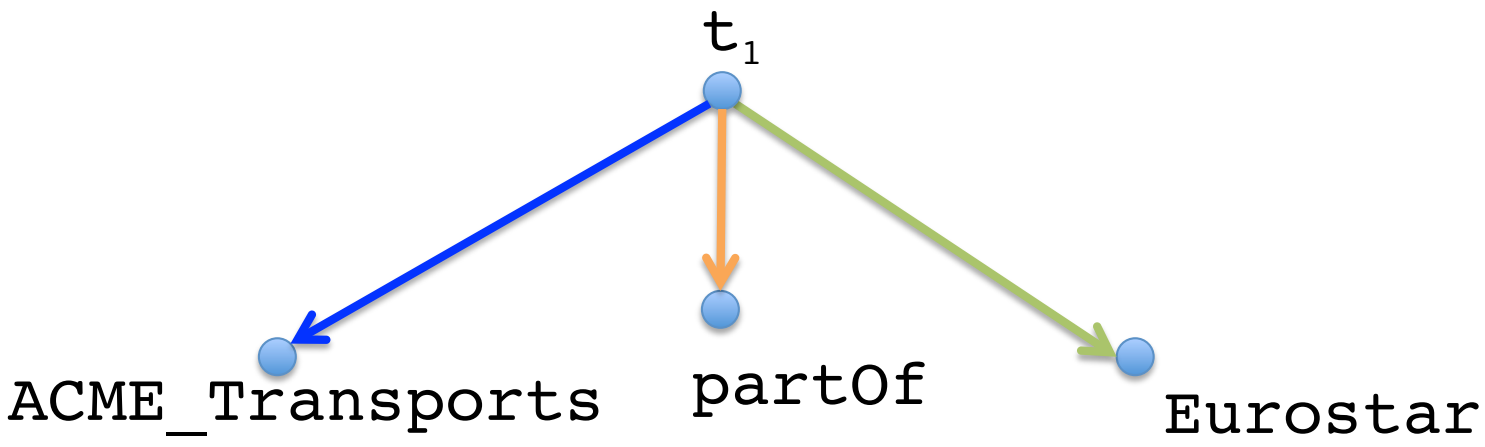
Enconding
RDF → GraphDatabase

<s,p,o>

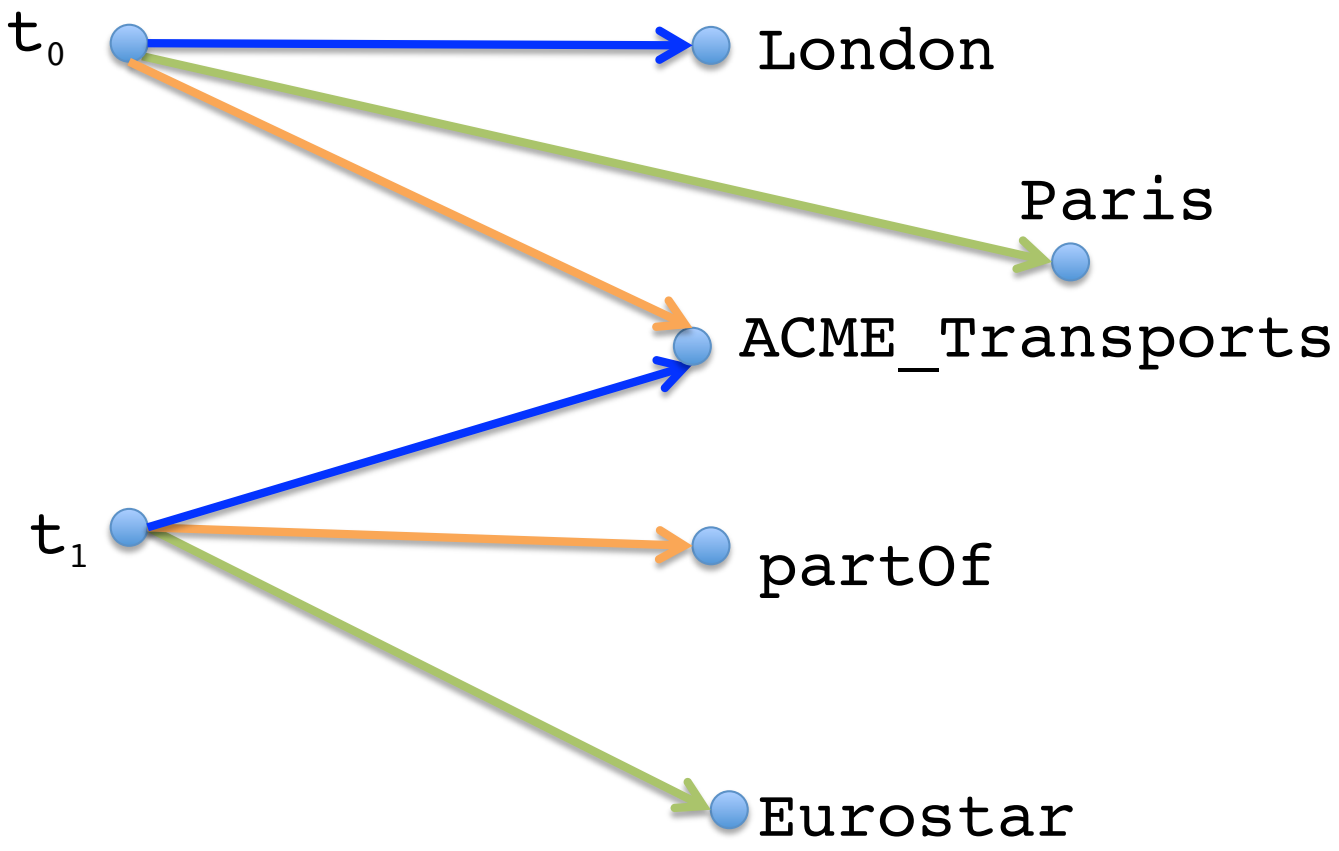
→

t_1 is a fresh constant standing for the triple

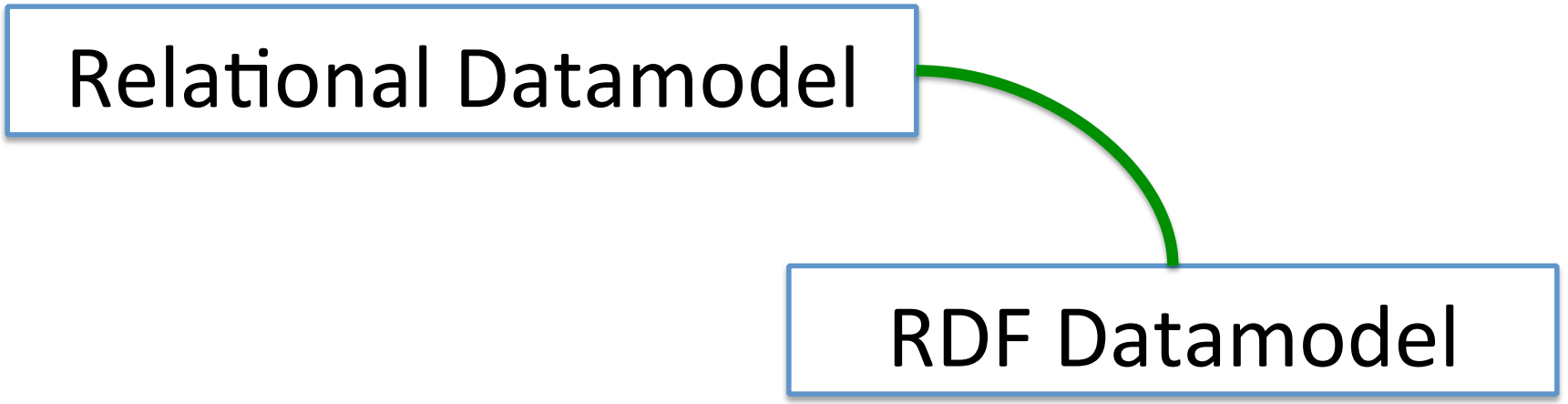
subject(t_1,s) **predicate**(t_1,p) **object**(t_1,o)



Encoding
RDF → GraphDatabase



Encoding
RDF → RelationalDatabase

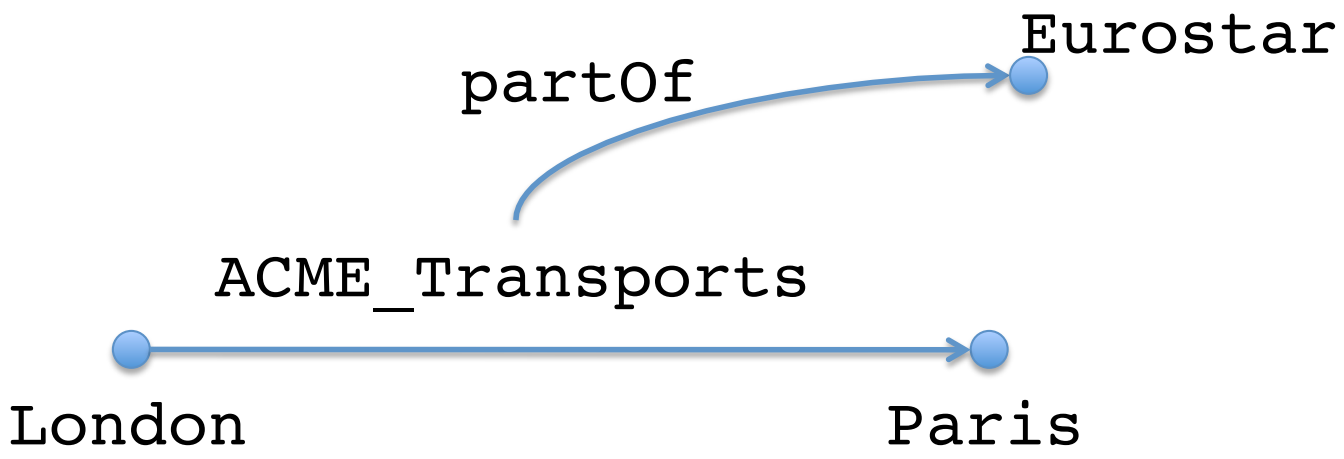


<s,p,o> → Triple(s,p,o)

Encoding
RDF → RelationalDatabase

Triple

London	ACME_Transports	Paris
ACME_Transports	partOf	Eurostar



Travelling between data-models

- Graph data-model retrocompatible with RDF

  `p(s,o)` $\rightarrow_{OK}$ `<s,p,o>`

- But general RDF too large for Graph-Datamodel

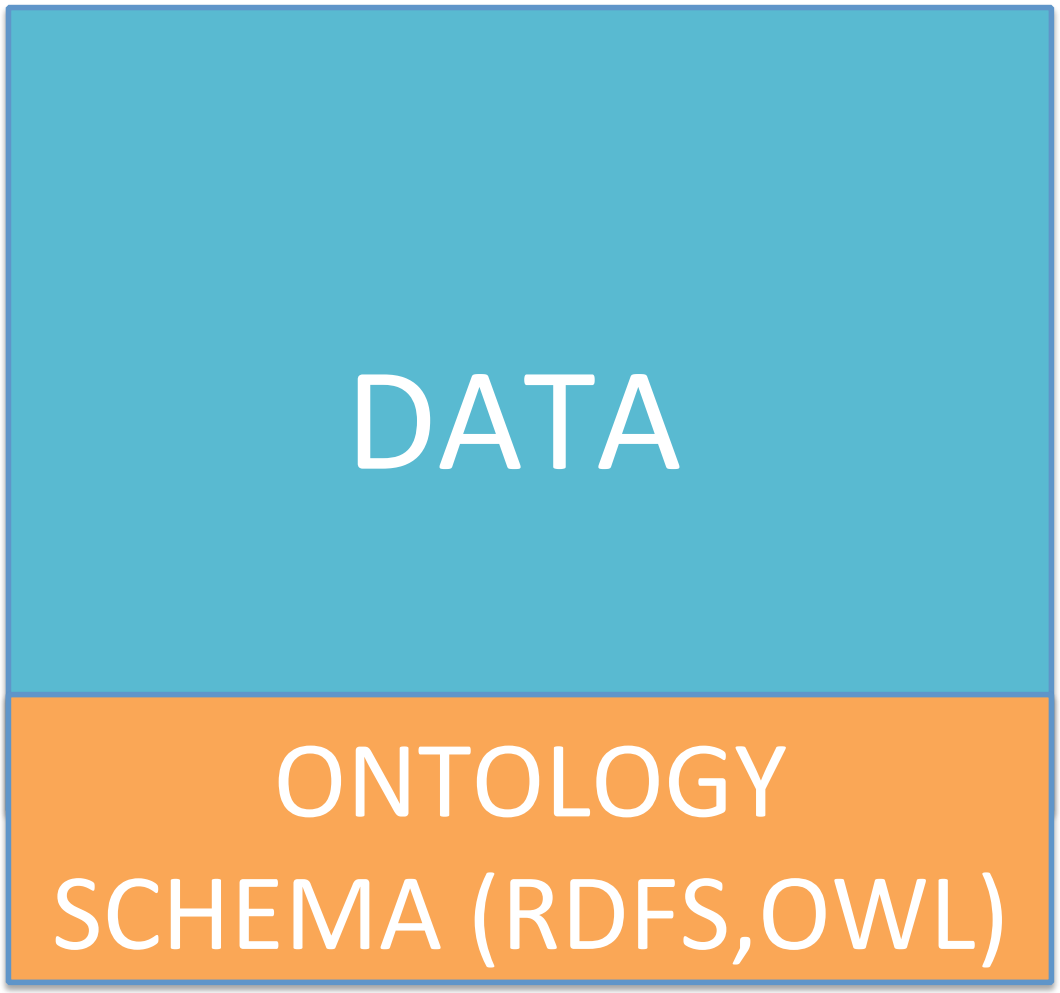
  `<p,p,p>` $\rightarrow_{NO}$ ~~`p(p,p)`~~

- Encodings RDF → Graph or Relational
  - *The trick*: RDF properties treated as values.

Summing Up

- By knowing SQL over binary relations you already know 90% of SPARQL.
- But SPARQL over RDF data is more than SQL over binary graphs.
- Because RDF is more than a binary graph
  - some triples are beyond graphs (meta-modelling)
  - in practice, it contains also the ontology schema and it is possible to query both at once (meta-querying)!
- Because SPARQL has more feautres (see next)
  - variables can be properties
  - regular expression paths

Anatomy of an RDF dataset



Under the Regime of Graphs

- It is sometimes necessary to impose that RDF data represents a graph.

- This is an hypothesis used throughout W3C standards to define « entailment regimes »
  - something “imposed” to control the behavior of data

- There is also the possibility of interpreting everything as value but, technically, do not expect a first order semantics as usual.

Under the Regime of Graphs

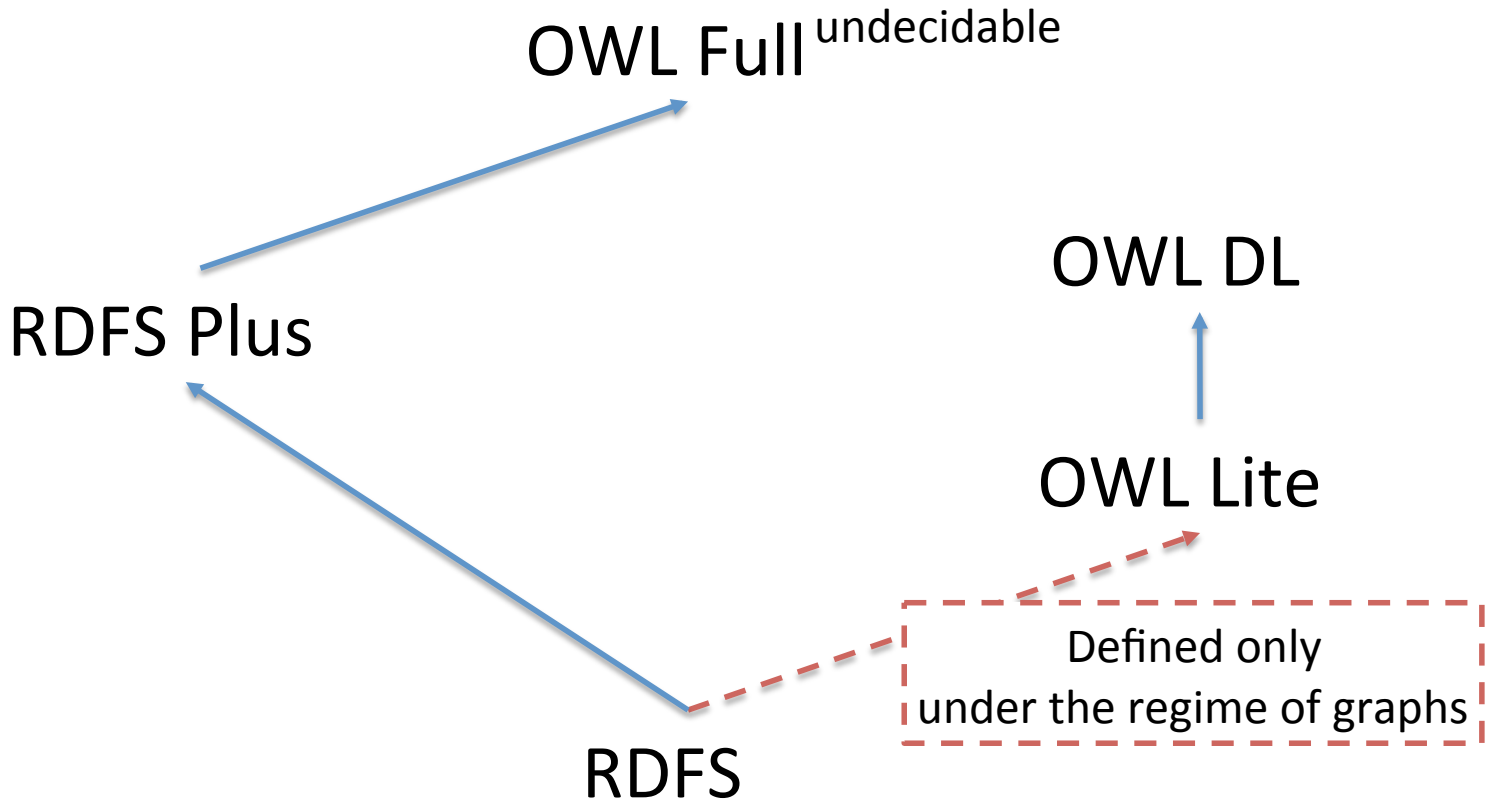
- An RDF dataset **D** is binary-graph only if

uri-properties(**D**) ∩ uri-subjects&objects(**D**) = ∅

uri-classes(**D**) are only objects of `rdf:type` triples

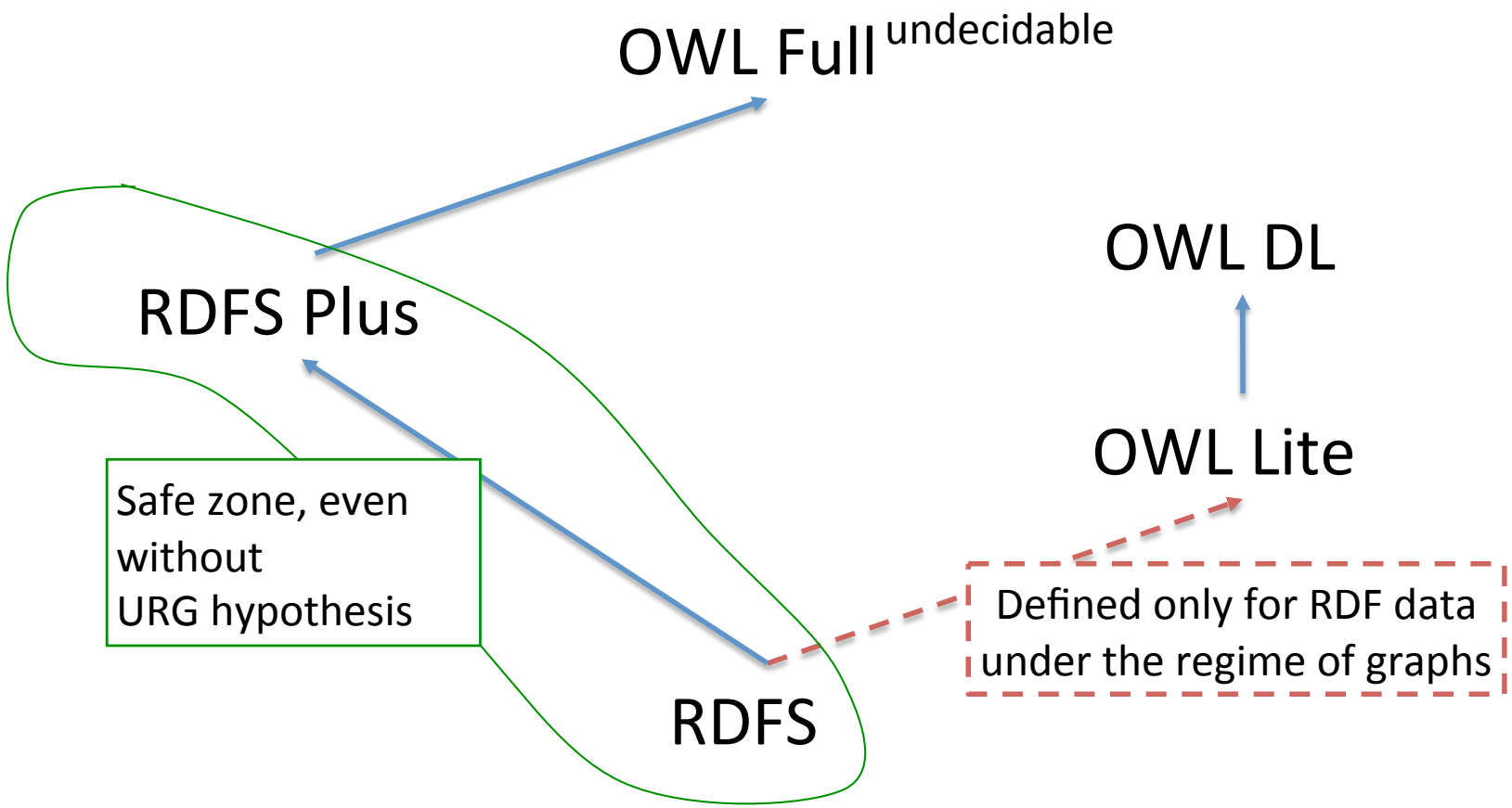
- Why is this so important ?

for Answering Queries with Ontologies !



- OWL-DL/Lite is defined **only** under the regime of graphs
- OWL Full imposes no such restriction, but it is undecidable

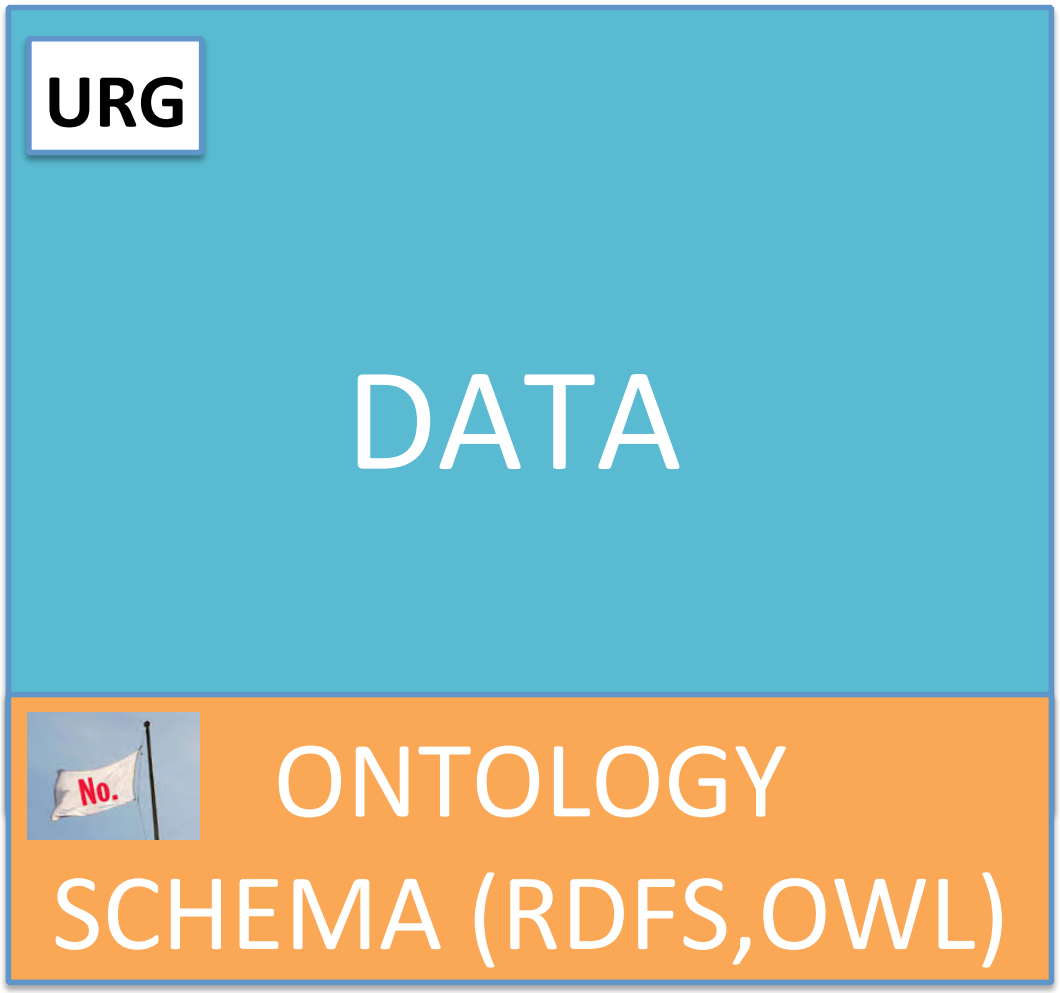
for Answering Queries with Ontologies !



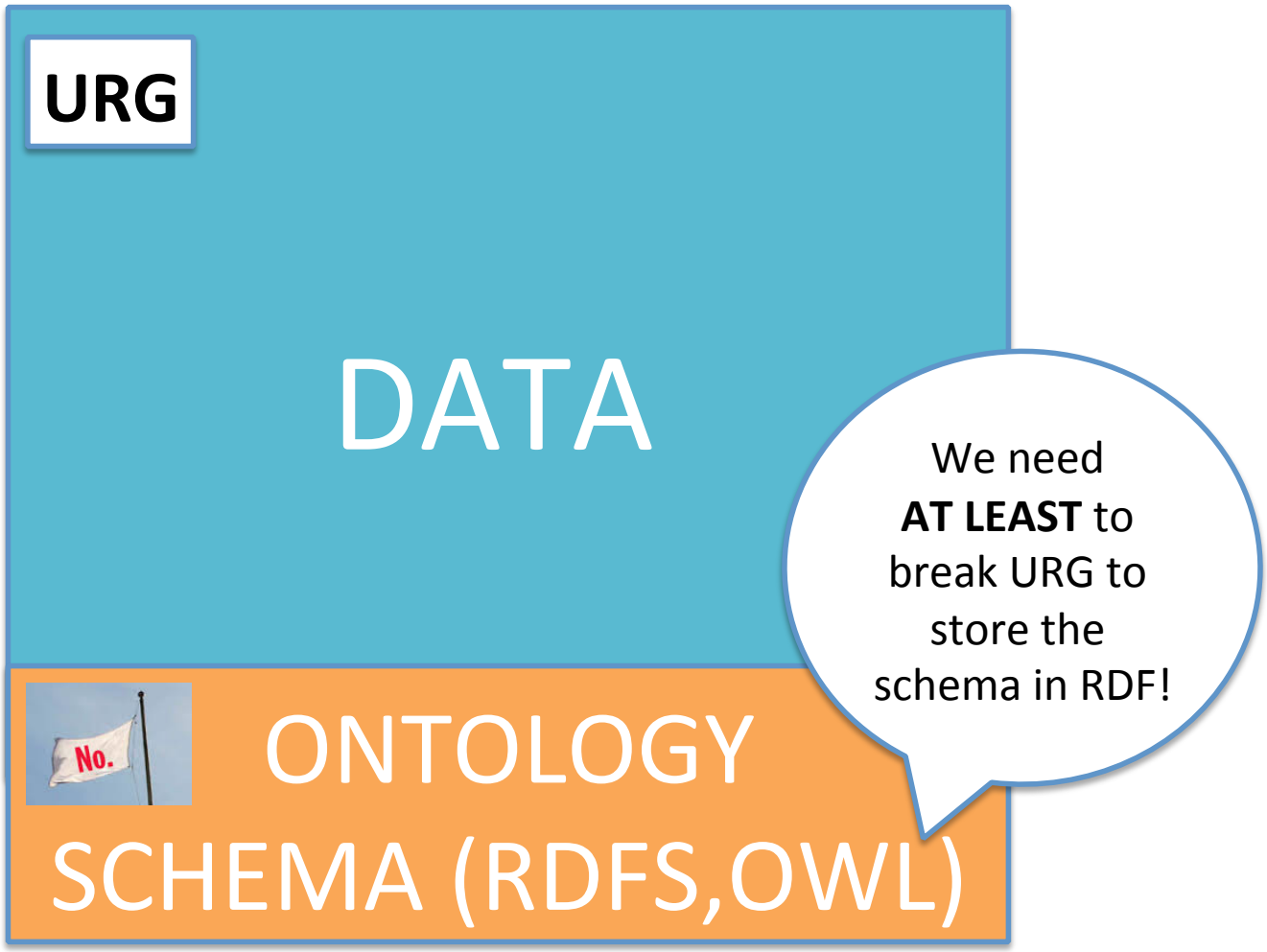
Conclusion: if RDF is not URG, better do reasoning in safe zone

From now on, we denote by **URG** the hypothesis of being Under the Regime of Graphs.

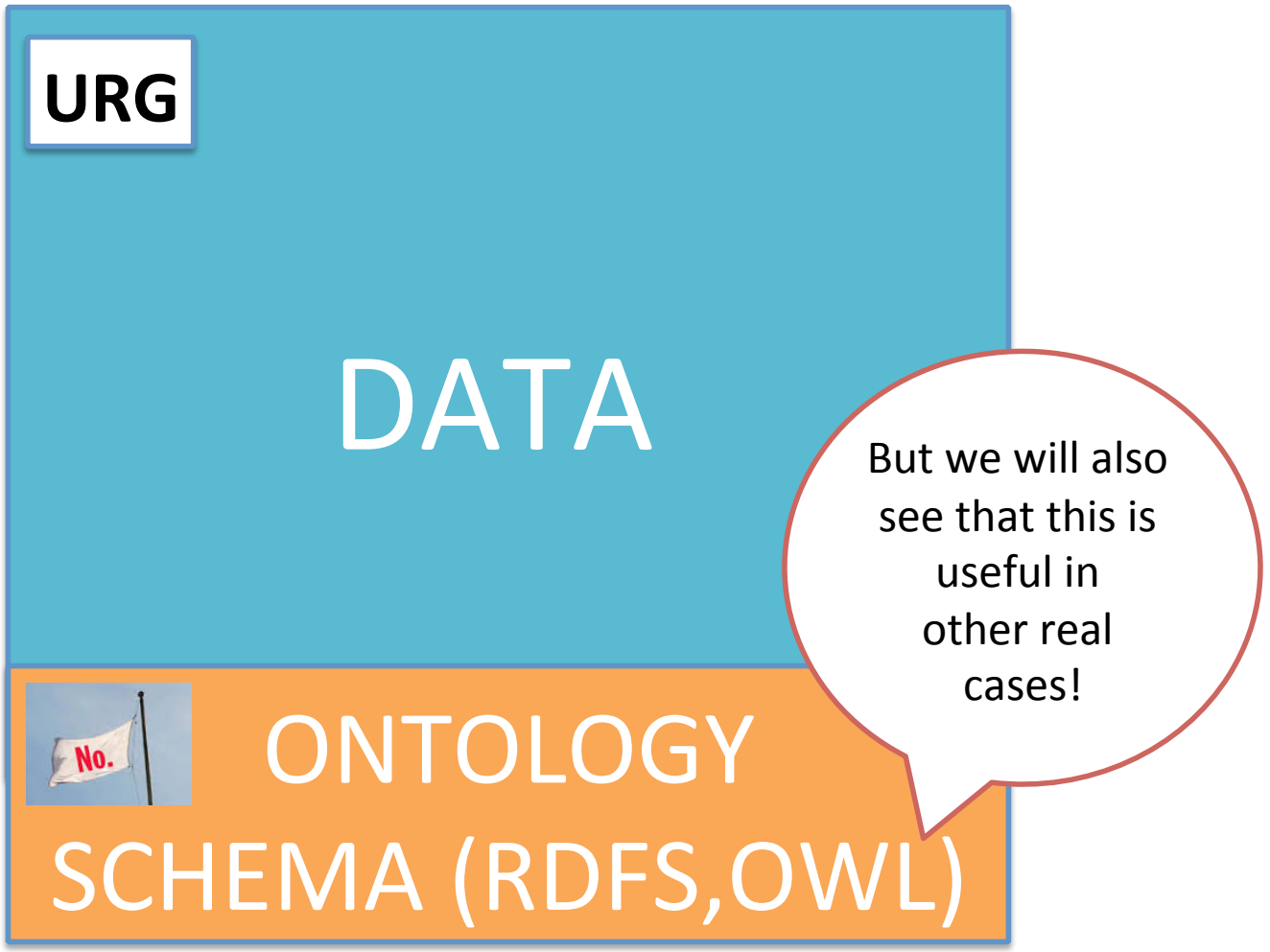
Anatomy of an RDF dataset



Anatomy of an RDF dataset



Anatomy of an RDF dataset

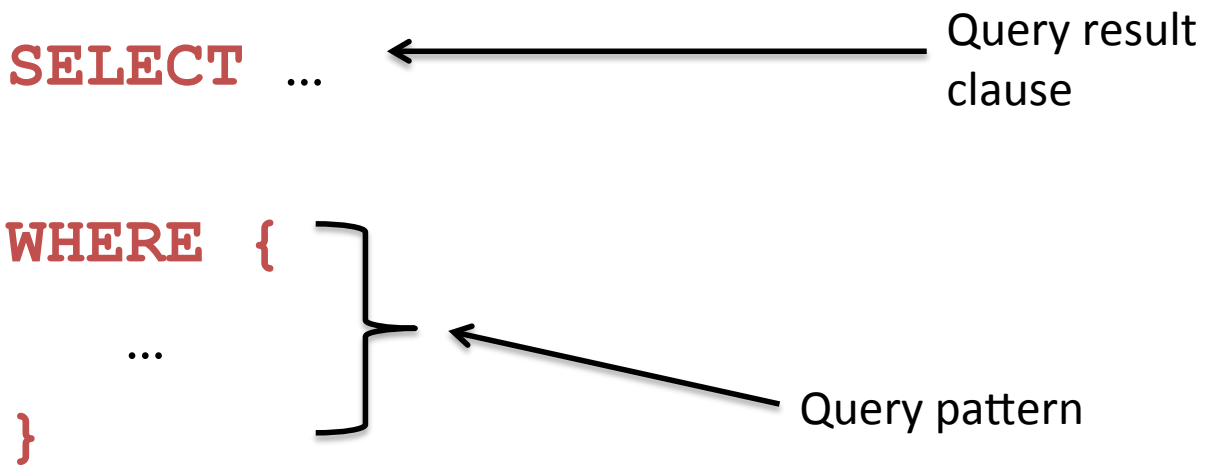


SPARQL :
AN RDF QUERY LANGUAGE

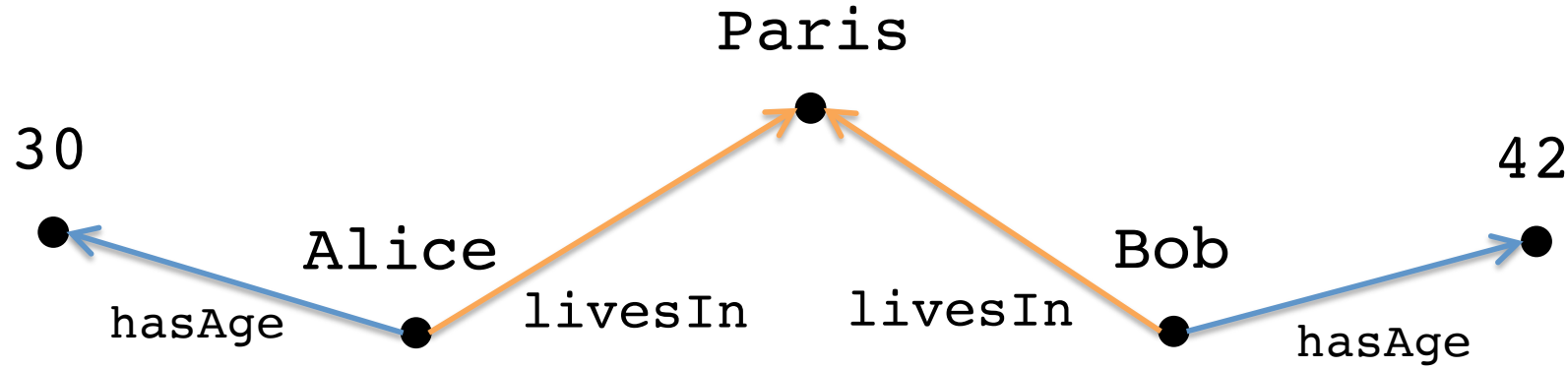
SPARQL Features

- Triple Patterns
- Prefixes
- From
- Construct
- Ask
- Limit
- Offset
- Order By
- Group By
- Limit
- Offset
- Multilingual
- Property Paths
- Federated Queries

Basic SPARQL Query Syntax



Conjunctive Queries (URG) in SPARQL : a different syntax for relational semantics




```
SELECT  ?name1, ?name2

WHERE { ?name1 livesIn ?city   .
        ?name2 livesIn ?city   . }
```

Conjunctive Queries (URG) in SPARQL : a different syntax for relational semantics

hasAge

Alice	30
Bob	42

livesIn

Alice	Paris
Bob	Paris

```
SELECT  ?name1, ?name2

WHERE { ?name1 livesIn ?city   .
        ?name2 livesIn ?city   . }
```

Conjunctive Queries (URG) in SPARQL : a different syntax for relational semantics

hasAge

Alice	30
Bob	42

livesIn

Alice	Paris
Bob	Paris

(Alice, Alice)
(Alice, Bob)
(Bob, Alice)
(Bob, Bob)

Recall requirement #2 - SPARQL must deal with Web Data

- `Alice, Bob, livesIn, hasAge` have no meaning in the Web

- We need URI(s) to treat them as resources

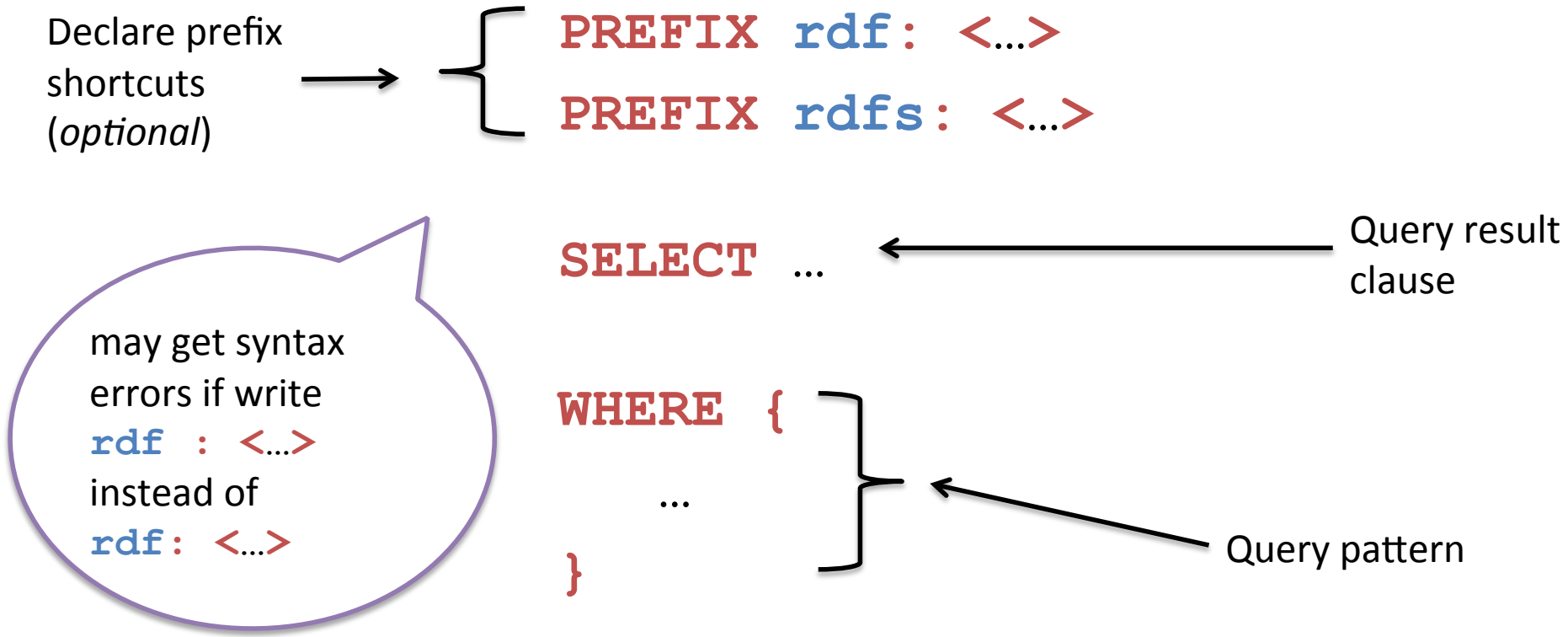
- URI = NAMESPACE + LOCAL-IDENTIFIER

`http://myOntology.org/livesIn`

URI(s) make reading/writing queries uncomfortable

```
SELECT  ?name1, ?name2

WHERE { ?name1
  http://myOntology.org/livesIn
                              ?city .
        ?name2
  http://myOntology.org/livesIn
                                   ?
                 city .}
```

Declaring Namespaces with PREFIX



More PREFIX=More Succinctness

```
PREFIX myOnto :
          <http://myOntology.org/>

SELECT   ?name1, ?name2

WHERE {  ?name1 myOnto:livesIn ?city.

         ?name2 myOnto:livesIn ?city.}
```

Common Prefixes

rdf:	<http://xmlns.com/foaf/0.1/>
rdfs:	<http://www.w3.org/2000/01/rdf-schema#>
owl:	<http://www.w3.org/2002/07/owl#>
xsd:	<http://www.w3.org/2001/XMLSchema#>
dc:	<http://purl.org/dc/elements/1.1/>
foaf:	<http://xmlns.com/foaf/0.1/>

More at http://prefix.cc

FOAF (2000-2015+)

- Friend of a Friend (FOAF): a dictionary of people-related terms

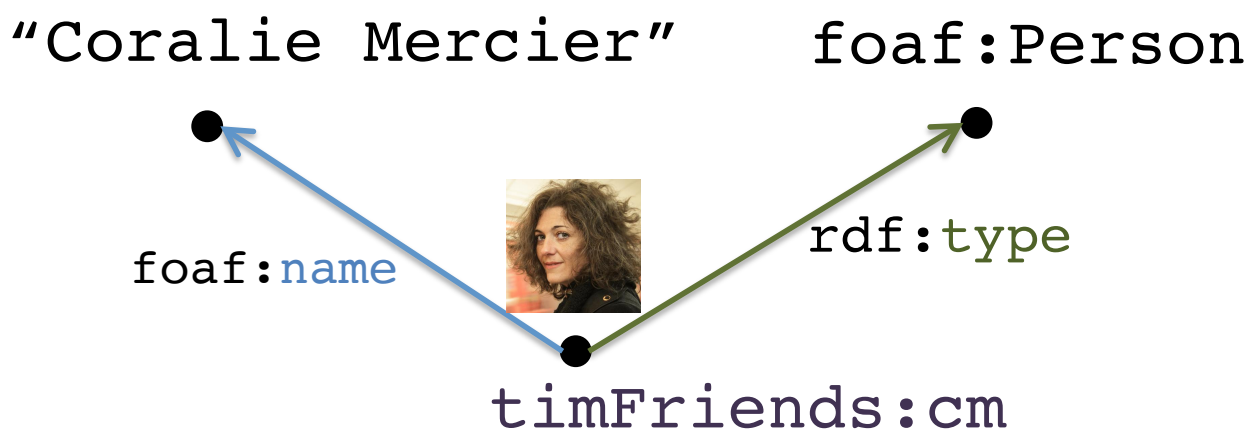
`foaf:name` `foaf:Person`

`foaf:mbox` `foaf:homepage` …

- Tim Berners-Lee's FOAF file (public available)

```
PREFIX timFriends:
<http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf#>
```

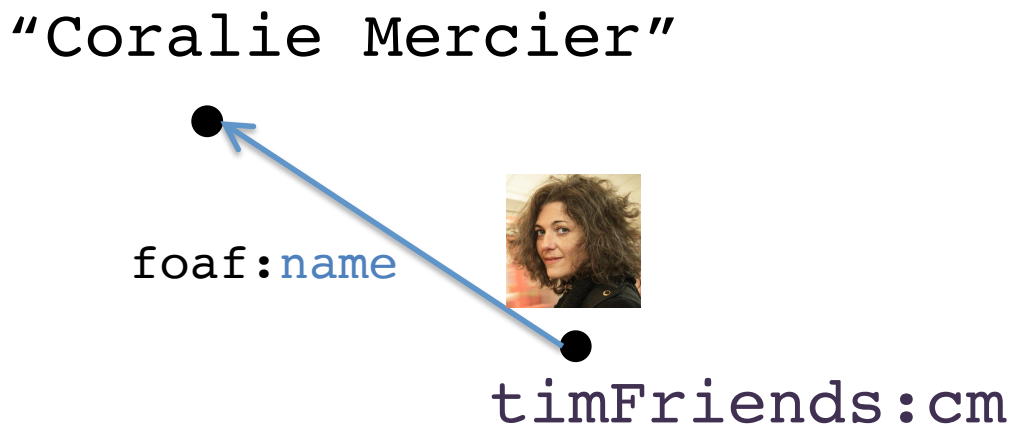
A Tim's friend



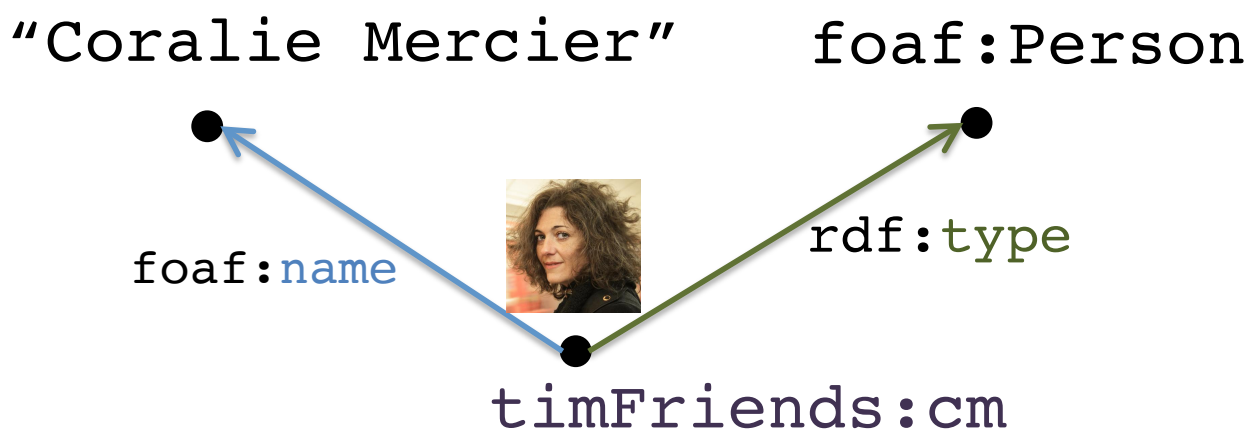
Typical design pattern for entities

timFriends:cm

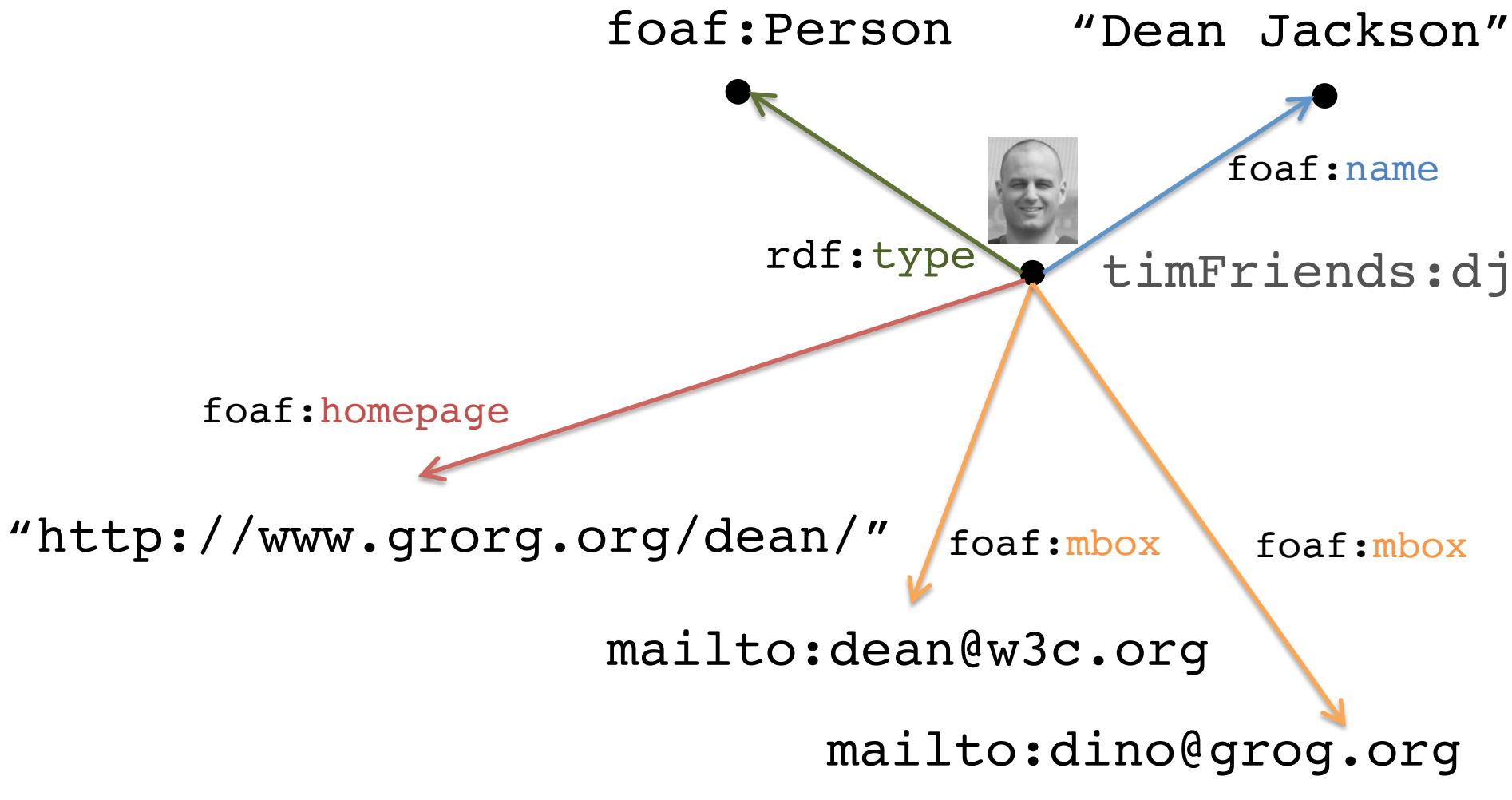
Typical design pattern for entities



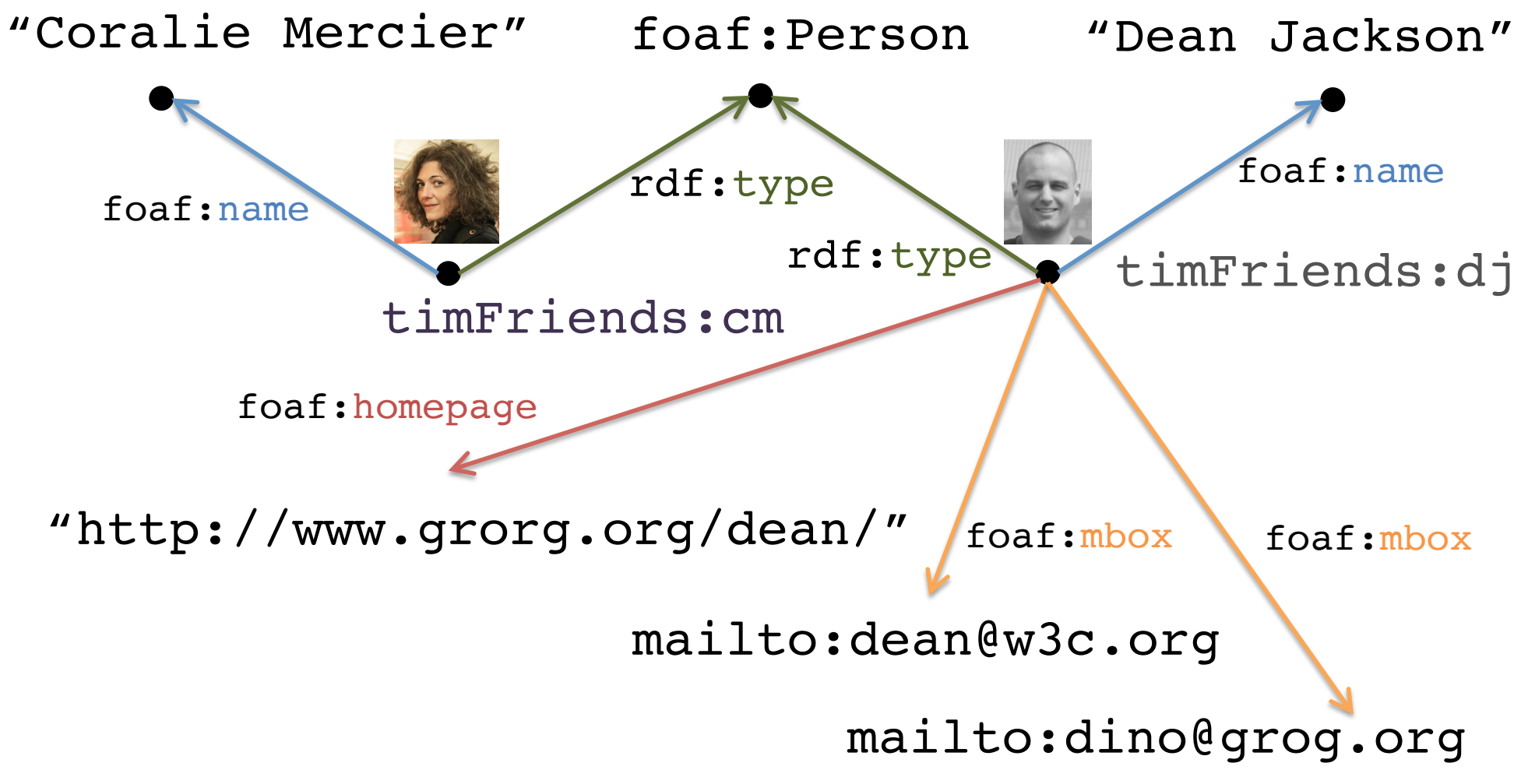
Typical design pattern for entities



Another Tim's friend



Let's query on Tim's friends



Let's query on Tim's friends (URG)

foaf:name

timFriends:cm	"Coralie Mercier"
timFriends:dj	"Dean Jackson"

foaf:mbox

timFriends:dj	"mailto:dean@w3c.org"
timFriends:dj	"mailto:dino@grog.or"g

rdf:type

timFriends:cm	foaf:Person
timFriends:dj	foaf:Person

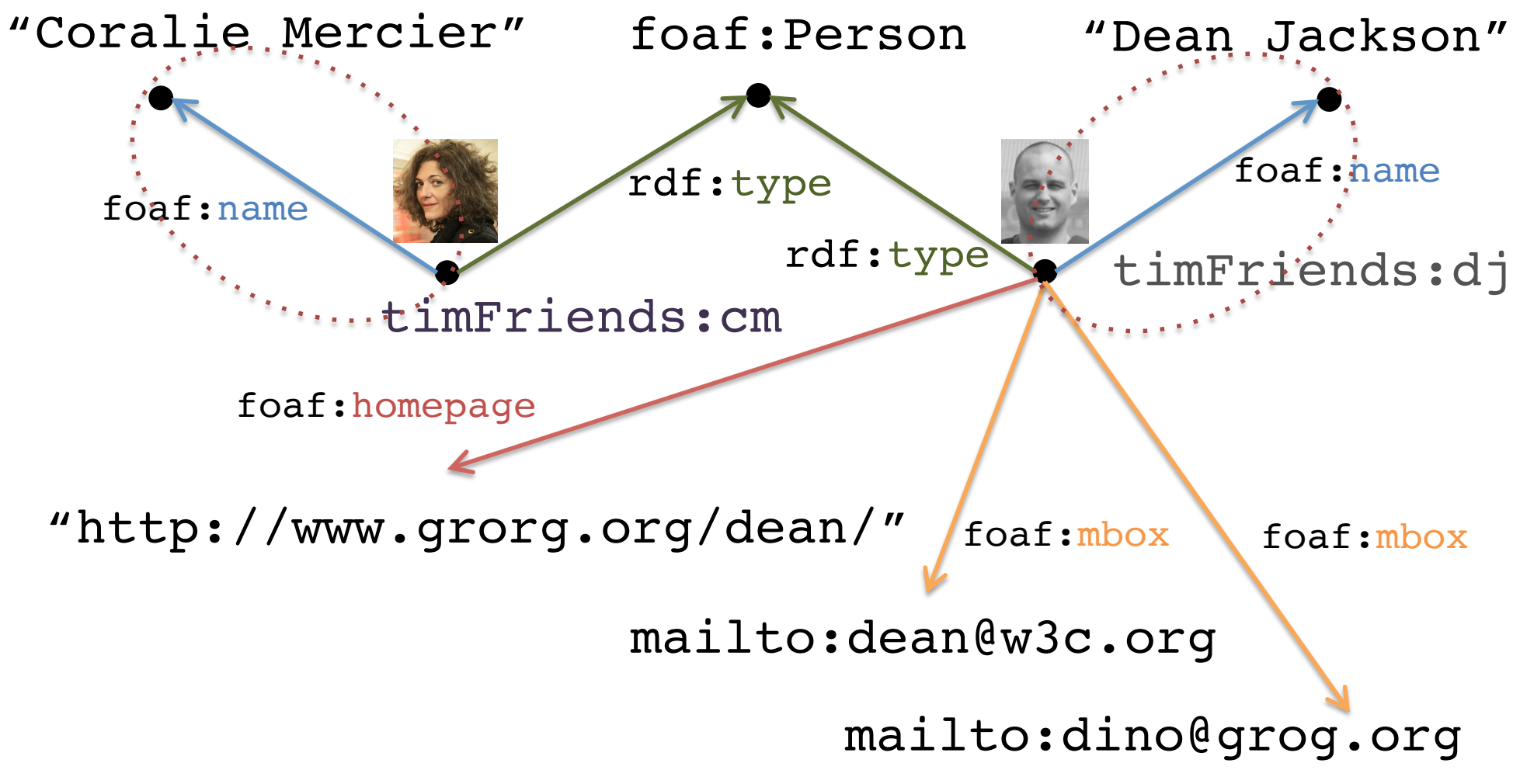
foaf:homepage

timFriends:dj	"http://www.grorg.org/dean/"

Let's find their names

```
PREFIX foaf:
        <http://xmlns.com/foaf/0.1/>

SELECT  ?name

WHERE {
      ?person    foaf:name    ?name   }
```

Let's find their names



Query answer = projection on mapping table

query-variable $\rightarrow_{\text{mapped\_into}}$ *URI* | *Literal* | *BNode*

	?person	?name
mapping 1 (gives query answer 1)		
mapping 2 (gives query answer 2)		

Query answer = projection on mapping-table

query-variable $\rightarrow_{\text{mapped\_into}}$ *URI* | *Literal* | *BNode*

	?person	?name
mapping 1 (gives query answer 1)	timFriends:cm	"Coralie Mercier"
mapping 2 (gives query answer 2)	timFriends:dj	"Dean Jackson"

Query answer = projection on mapping-table

query-variable $\rightarrow_{\text{mapped\_into}}$ *URI* | *Literal* | *BNode*

mapping 1 (gives query answer 1)

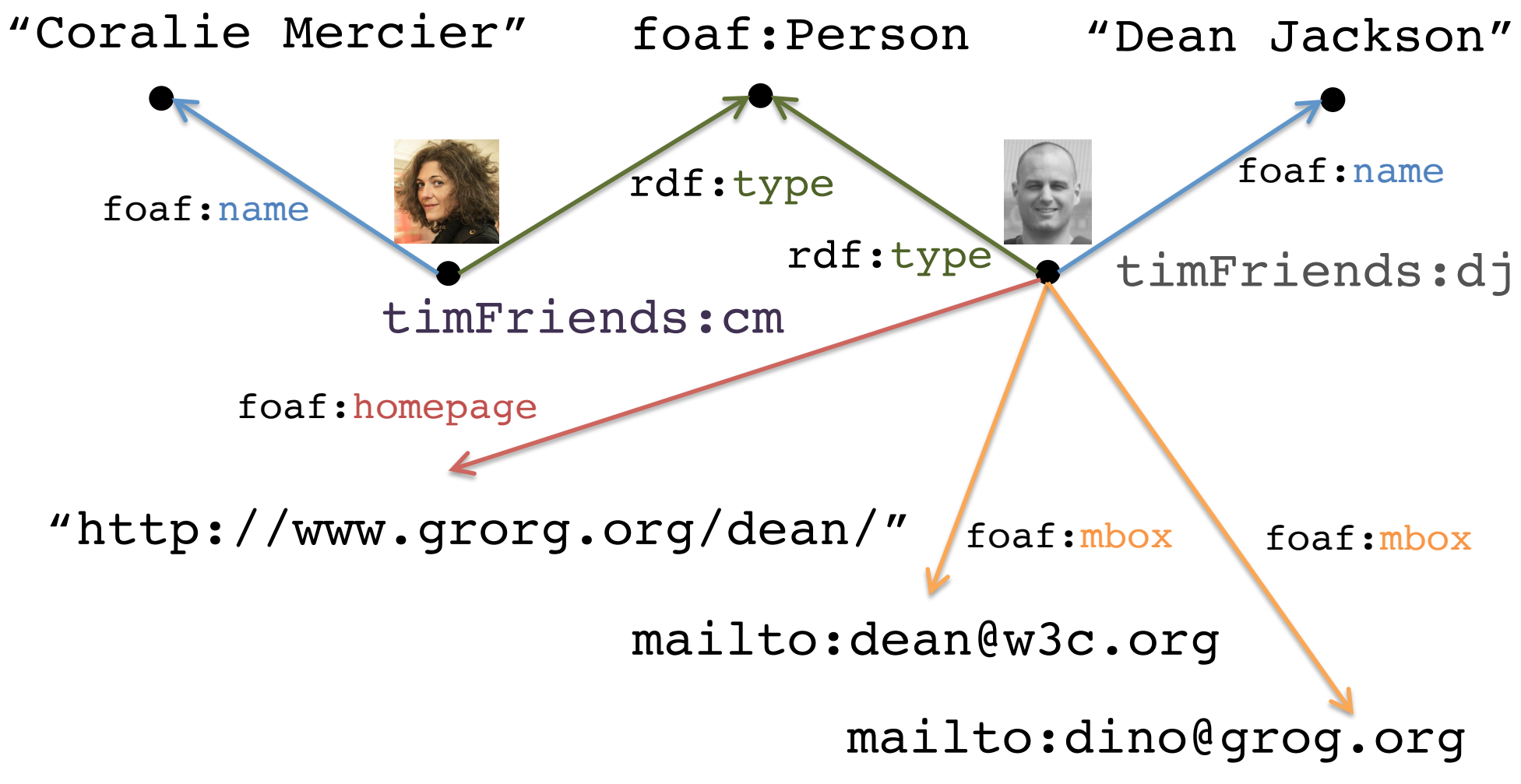
mapping 2 (gives query answer 2)

?name
"Coralie Mercier"
"Dean Jackson"

Name of Tim's friends with mail ?

```
PREFIX foaf:
          <http://xmlns.com/foaf/0.1/>


SELECT ?person , ?name , ?email

WHERE { ?person  foaf:name  ?name .
        ?person  foaf:mbox  ?email . }
```

Name of Tim's friends with mail ?



Query answer = projection on mapping-table

query-variable $\rightarrow_{\text{mapped\_into}}$ *URI* | *Literal* | *BNode*

	?person	?name	?email
mapping 1	timFriends:dj	“Dean Jackson”	"mailto:dean@w3.org"
mapping 2	timFriends:dj	“Dean Jackson”	"mailto:dino@grorg.org”

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>

SELECT ?person, ?name, ?email
WHERE { ?person   foaf:name   ?name .
        ?person   foaf:mbox   ?email .  }
```

Back to our query

```
PREFIX foaf:
          <http://xmlns.com/foaf/0.1/>


SELECT ?person , ?name

WHERE { ?person  foaf:name  ?name .
        ?person  foaf:mbox  ?email . }
```

Back to our query

```
PREFIX foaf:
        <http://xmlns.com/foaf/0.1/>


SELECT ?person , ?name
```

(before we assumed a local graph)

```
WHERE { ?person  foaf:name  ?name .
        ?person  foaf:mbox  ?email . }
```

Back to our query

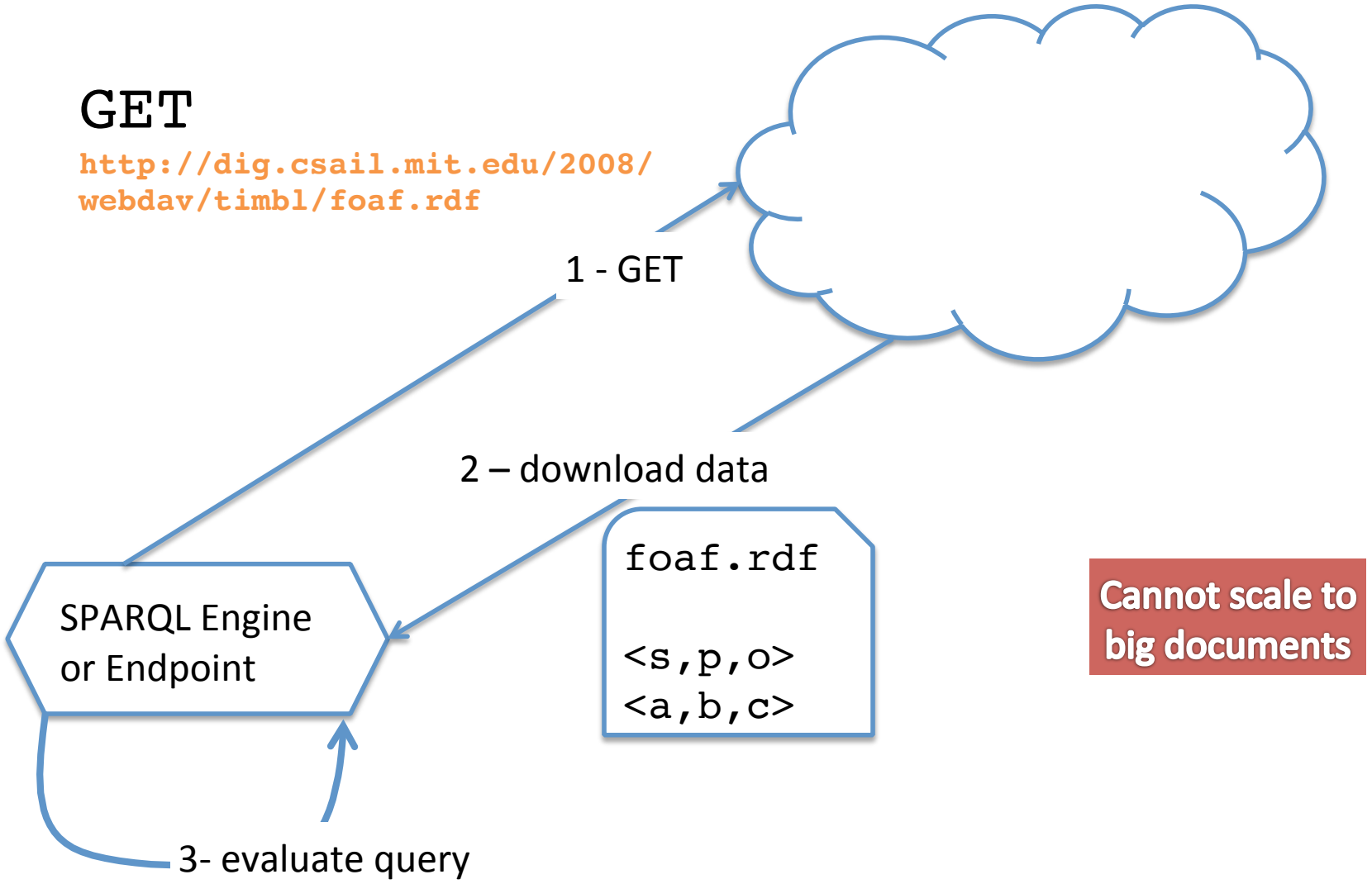
```
PREFIX foaf:
         <http://xmlns.com/foaf/0.1/>


SELECT ?person , ?name
         (but this is not necessary)
WHERE { ?person  foaf:name  ?name .
        ?person  foaf:mbox  ?email . }
```

HTTP-Based Query

FROM

```
PREFIX foaf:
          <http://xmlns.com/foaf/0.1/>

SELECT ?person , ?name
FROM <http://dig.csail.mit.edu/2008/
                    webdav/timbl/foaf.rdf>
WHERE { ?person  foaf:name  ?name .
        ?person  foaf:mbox  ?email . }
```

HTTP-Based Query



Convert Vocabulary : FOAF to Vcard

CONSTRUCT

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT { ?person vCard:FN ?name .
            ?person vCard:URL ?url    . }

FROM
   <http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf>

WHERE {
          ?person foaf:name ?name .
          ?person foaf:homepage ?url . }
```

- CONSTRUCT returns an RDF graph by taking results of the equivalent SELECT

Query answer = Triples!

(constructed from mapping-table)

mapping I

?person	?name	?homepage
timFriends:dj	"Dean Jackson"	"http://www.grorg.org/dean/"

<timFriends:dj, **vCard:FN,** "Dean Jackson">

<timFriends:dj, **vCard:URL,** "http://www.grorg.org/dean/">

Does Tim have at least two friends?

ASK

```
ASK

FROM
   <http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf>

WHERE {
      ?person1  rdf:type   foaf:Person .

      ?person2  rdf:type   foaf:Person .




}
```

`ASK` queries return a boolean value (true | false)

Does Tim have at least two friends?

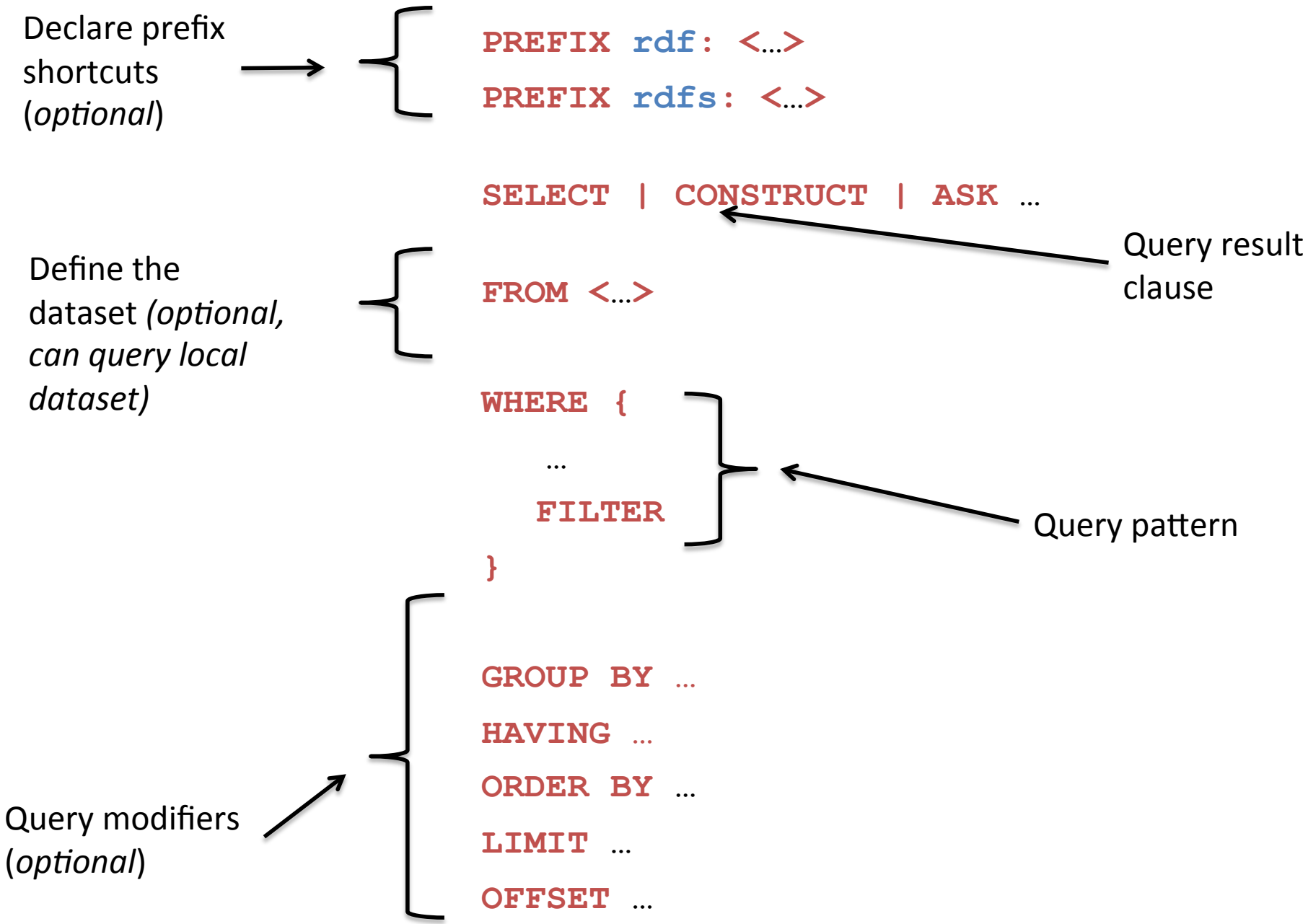
ASK

```
ASK

FROM
   <http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf>

WHERE {
      ?person1   rdf:type    foaf:Person .

      ?person2   rdf:type    foaf:Person

      FILTER( ?person1 != ?person2 )

}
```

attention aux fausses réponses

ASK queries return a boolean value (true | false)

Anatomy of a Query

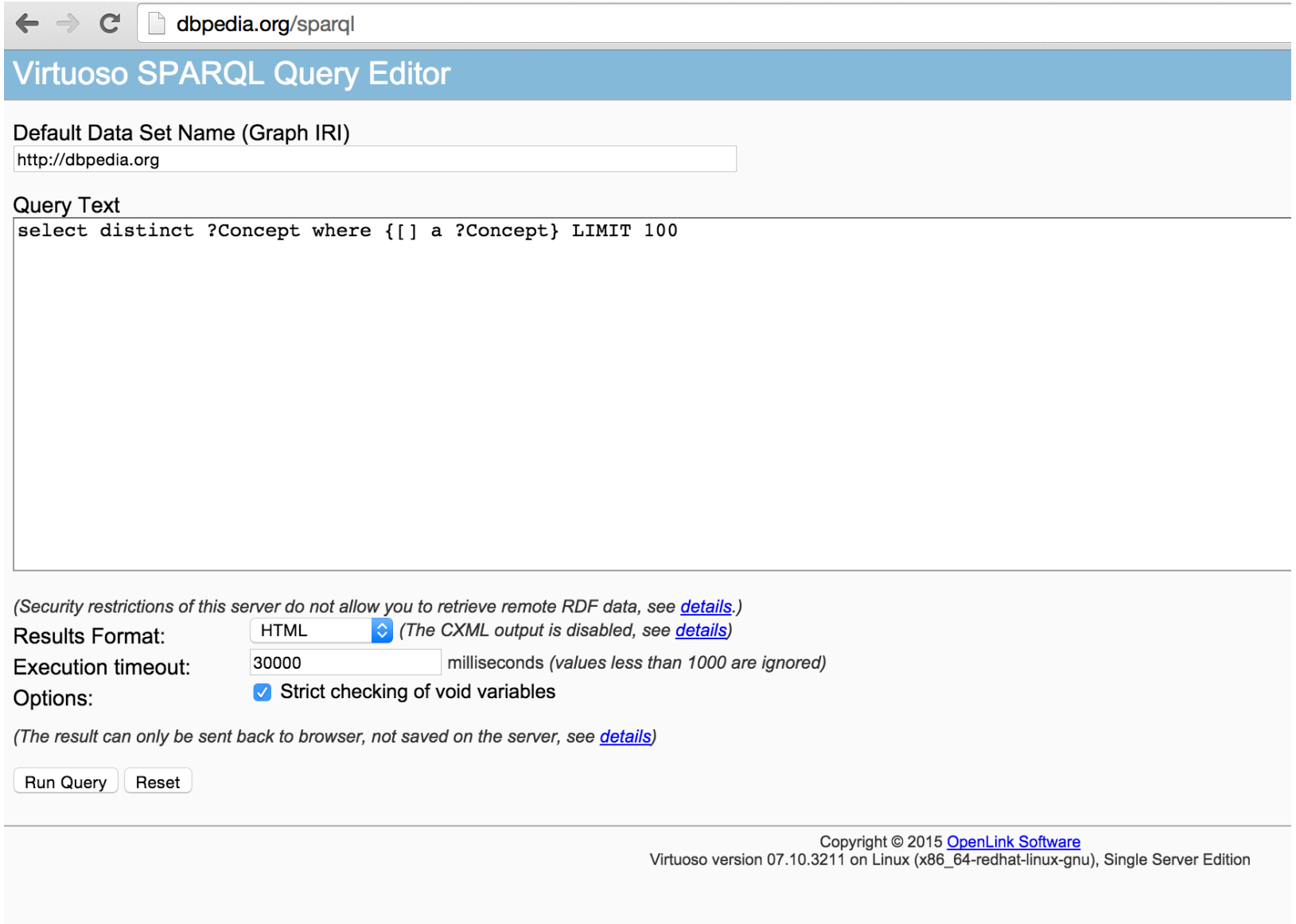


RDF version of information from Wikipedia

- data is extracted from Wikipedia's infoboxes, category hierarchy, article abstracts, and various external links

- contains over 3 billion triples

DBPedia SPARQL Endpoint



Endpoints

- A SPARQL endpoint accepts queries and returns results via HTTP.
- Generic endpoints will query any Web-accessible RDF data.
- Specific endpoints are hardwired to query against particular datasets (e.g. DBPedia).

DBPedia Ontology Schema

- Overall, >600 classes and >2500 properties

  Some main classes

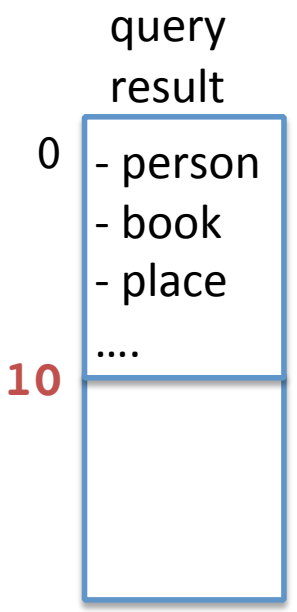
Class	Instances
Resource (overall)	4,233,000
Place	735,000
Person	1,450,000
Species	251,000
Work	411,000
Organisation	241,000

- Question : how to learn the ontology schema ?

Find **10** DBPedia concepts

LIMIT

```
SELECT DISTINCT ?class

WHERE {

    ?s rdf:type ?class
}

LIMIT 10
```



LIMIT the number of query results

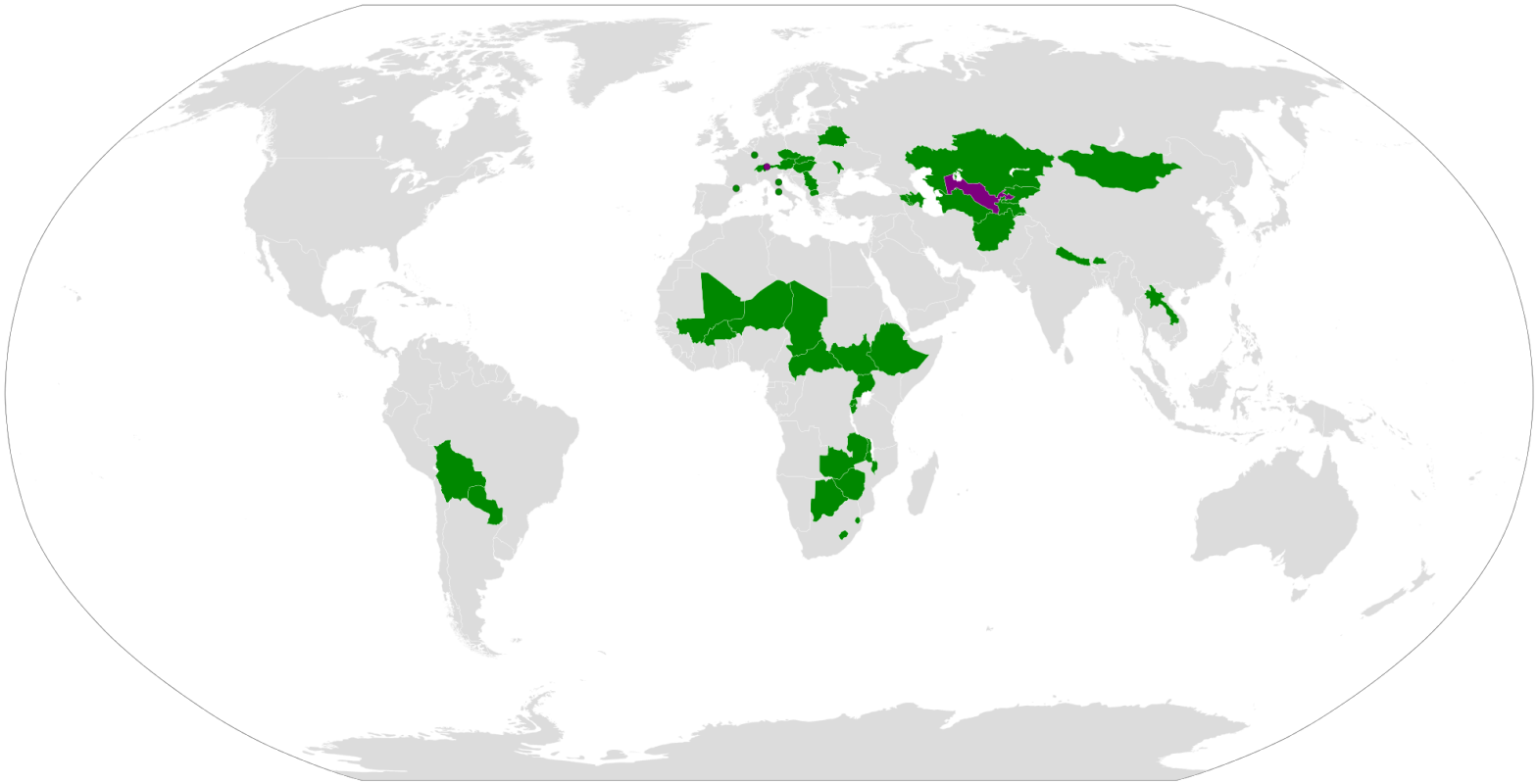
Split the previous list in two

OFFSET

```
SELECT DISTINCT ?concept

WHERE {

    ?s rdf:type ?concept
}

OFFSET 5

LIMIT 10
```

← Start from index

Find all landlocked countries



Prefixes used in the following queries

```
PREFIX rdf: <http://xmlns.com/foaf/0.1/>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX dbp: <http://dbpedia.org/class/yago/>

PREFIX prop: <http://dbpedia.org/property/>

PREFIX dbo: <http://dbpedia.org/ontology/>
```

Find all landlocked countries

```
SELECT ?country_name

WHERE {

?country rdf:type dbp:LandlockedCountries.

?country rdfs:label ?country_name .

}
```

Countries with 15 million+ inhabitants ?

```
SELECT ?country_name ?population

WHERE {

?country rdf:type dbp:LandlockedCountries.

?country rdfs:label ?country_name .

?country prop:populationEstimate
                                    ?population .

FILTER (?population > 15000000) .
}
```

Rank DBPedia-Countries by Population

```
SELECT ?country_name ?population

WHERE {

?country rdf:type dbp:LandlockedCountries.

?country rdfs:label ?country_name .

?country prop:populationEstimate ? population .

FILTER (?population > 15000000) .
}

ORDER BY DESC(?population)
```

How many DBPedia-people per country ?

GROUP BY

```
SELECT ?country_name
       (COUNT(DISTINCT ?country) AS ?N_people)

WHERE {

   ?country  rdf:type dbo:country .
   ?country rdfs:label ?country_name .
   ?person dbo:nationality ?country
}
GROUP BY ?country
```

- Group-by in the same spirit as SQL (HAVING, COUNT, MIN..)

Literals

Plain literals:

```
"a plain literal"
```

Plain literal with language tag:

```
"bonjour"@fr
```

Typed literal:

```
"13"^^xsd:integer
```

Shortcuts:

```
true ⇨ "true"^^xsd:boolean
3    ⇨   "3"^^xsd:integer
4.2  ⇨  "4.2"^^xsd:decimal
```

Extract country names in English (Multilingual)

```
SELECT ?country_name ?population

WHERE {
   ?country rdf:type dbp:LandlockedCountries.
   ?country rdfs:label ?country_name .
   ?country prop:populationEstimate ?population .

     FILTER (?population > 15000000 &&
             langMatches(  lang(?country_name) , "EN")) .

} ORDER BY DESC(?population)
```

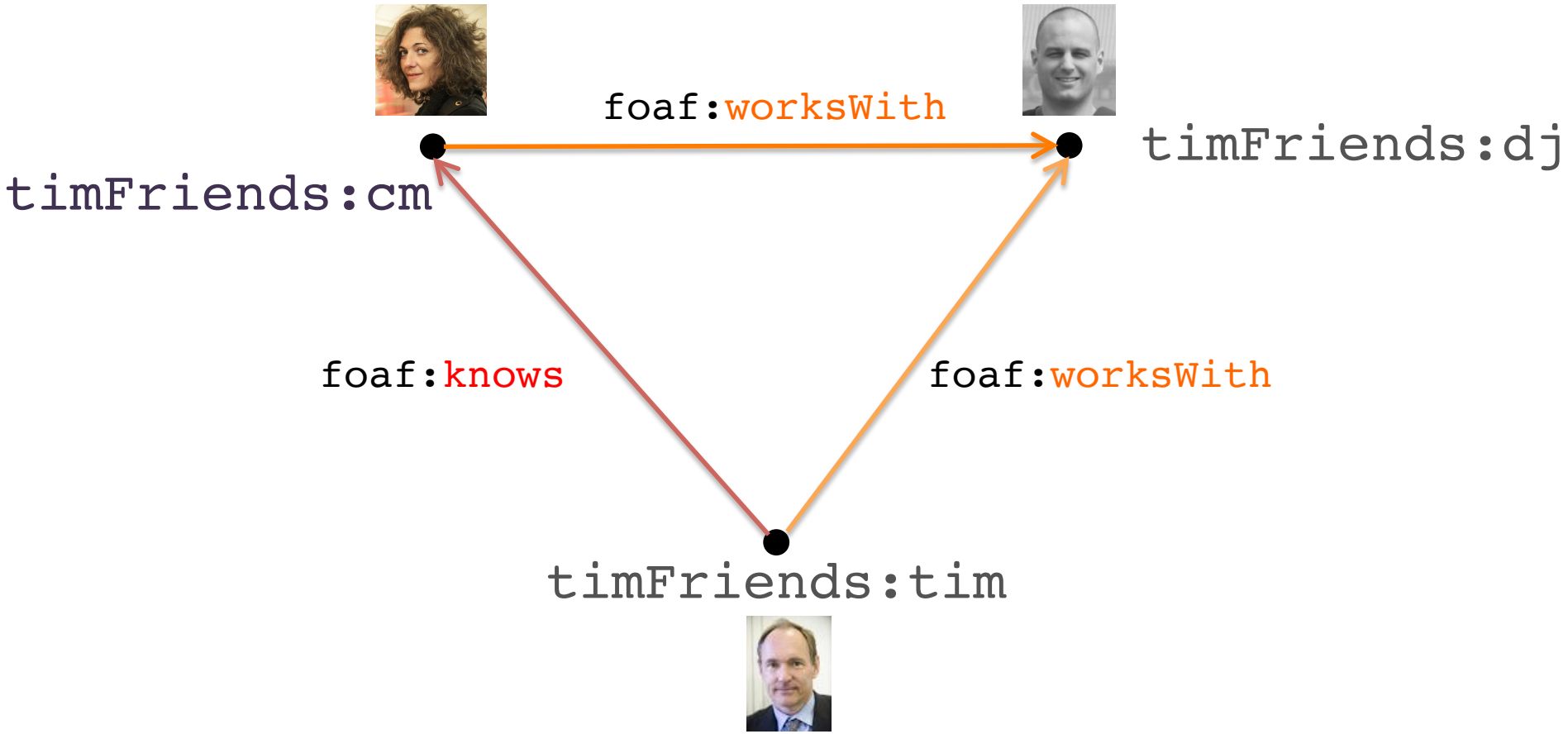
- `lang` extracts a literal's language tag, if any; `langMatches` matches a language tag

Multilingual Results (“EN”, “FR”)

?country_name	?population
“Ethiopia"@en	93877025
"Afghanistan"@en	31108077
"Uzbekistan"@en	30183400
"Malawi"@en	16407000
"Burkina Faso"@en	15730977

?country_name	?population
“Éthiopie”@fr	93877025
"Afghanistan”@fr	31108077
"Ouzbékistan”@fr	30183400
"Malawi”@fr	16407000
"Burkina Faso”@fr	15730977

A small social graph



Tim’s friends in the social graph

```
PREFIX timFriends:
  <http://dig.csail.mit.edu/2008/
                webdav/timbl/foaf.rdf>

SELECT ?anyP ?person

WHERE {
     timFriends:tim    ?anyP  ?person
}
```

Query answer = projection on mapping-table

query-variable ➔$_{\text{mapped\_into}}$ *URI* | *Literal* | *BNode*

	?anyP	**?person**
mapping 1	foaf:worksWith	timFriends:dj
mapping 2	foaf:knows	timFriends:cm

```
SELECT ?anyP ?person
WHERE {
       timFriends:tim   ?anyP  ?person
}
```

Most common use of property variables

```
SELECT ?anyP ?person

WHERE {
   timFriends:tim  ?anyP   ?person .
   ?anyP  rdfs:subPropertyOf foaf:knows
}
```

Over a dataset containing also

<foaf:worksWith,rdfs:subPropertyOf,foaf:knows>

Metaquerying

- (in the presence of ontologies) this is the case where the classes and properties in the ontology can match the variables in the query, and may thus be returned as query answers

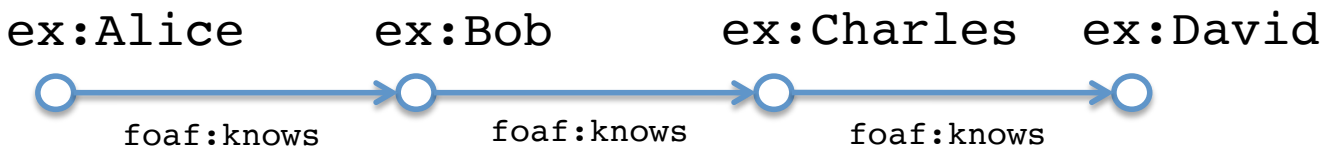
- This mechanism allows to express queries that are beyond first order logic

Property Paths

- Predicates are combined with regular-expression-like operators:

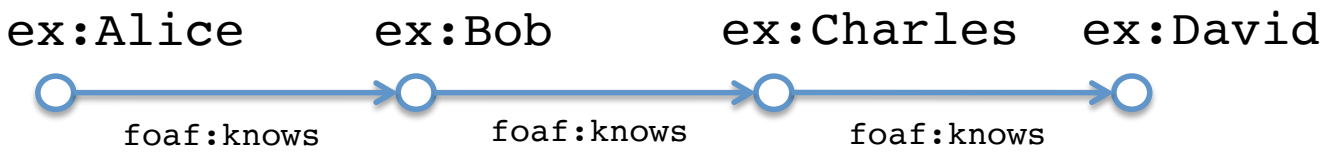
Construct	Meaning
`path1/path2`	Forwards path (`path1` followed by `path2`)
`^path1`	Backwards path (object to subject)
`path1|path2`	Either `path1` or `path2`
`path1*`	`path1`, repeated zero or more times
`path1+`	`path1`, repeated one or more times
`path1?`	`path1`, optionally
`!uri`	Any predicate except `uri`
`!^uri`	Any backwards (object to subject) predicate except `uri`

Is there a path between Alice and David ?



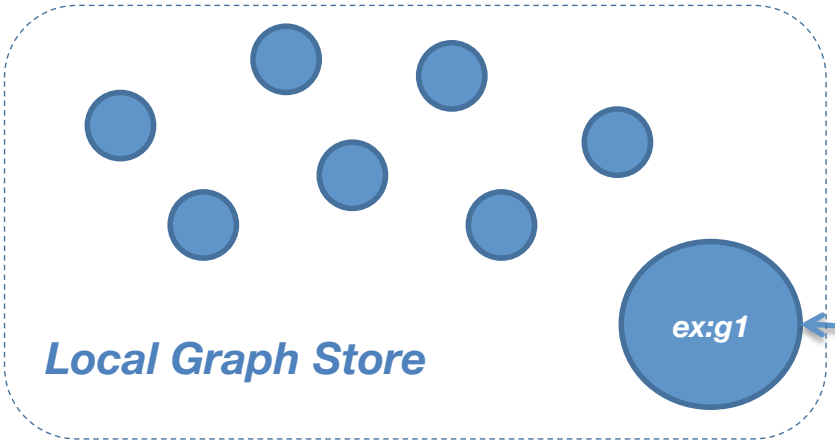

```
ASK

WHERE   {

   ex:Alice     foaf:knows    ex:David

}
```

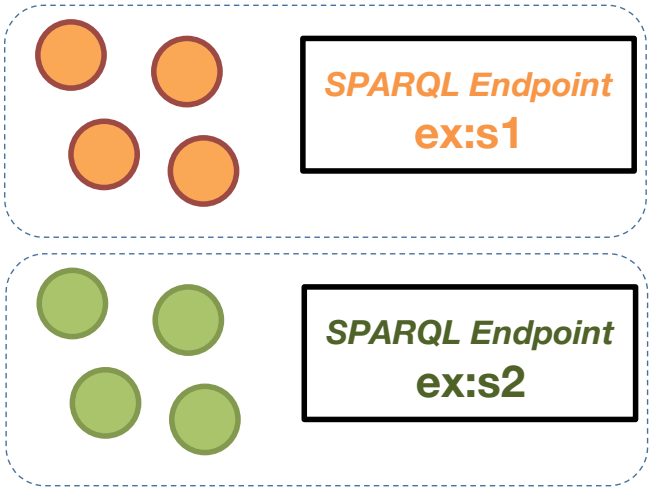
Is there a path between Alice and David ?

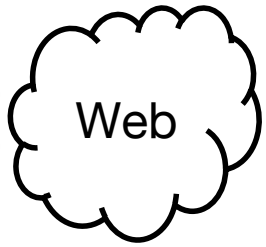



```
ASK

WHERE   {

   ex:Alice    (foaf:knows)+    ex:David

}
```

Federated Query








```
SELECT …

WHERE {
      sub-query_local

   SERVICE ex:s1
   { sub-query_ex:s1 }

   SERVICE ex:s2
   { sub-query_ex:s2 }
}
```

Find the birth dates of all of the actors in Star Trek: The Motion Picture

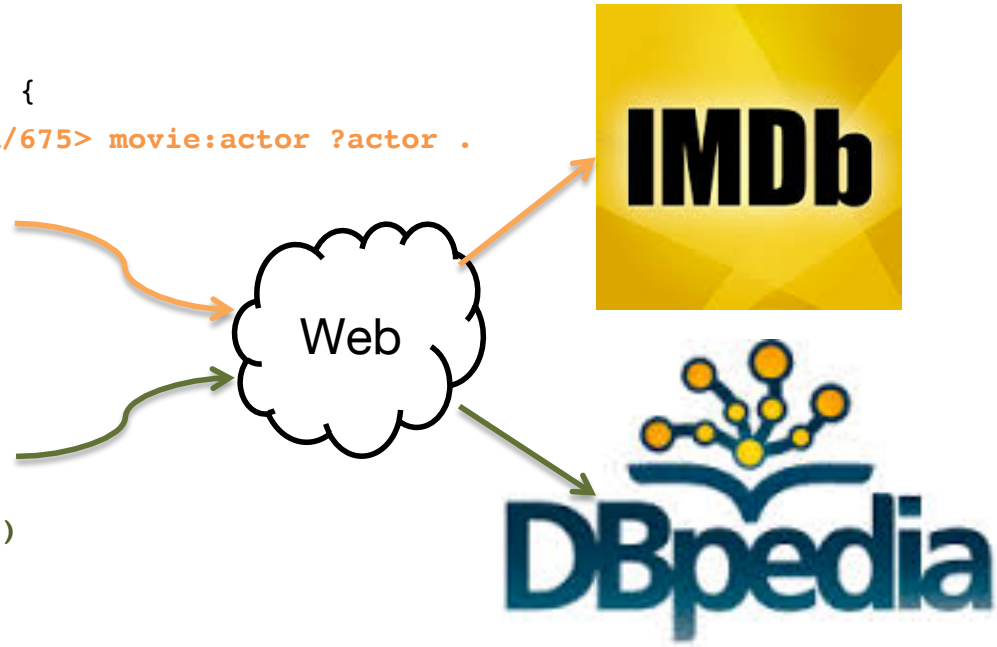
```
PREFIX movie: <http://data.linkedmdb.org/resource/movie/>
PREFIX dbpedia: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?actor_name ?birth_date
FROM <http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf> # placeholder graph

WHERE {
  SERVICE <http://data.linkedmdb.org/sparql> {
    <http://data.linkedmdb.org/resource/film/675> movie:actor ?actor .
    ?actor movie:actor_name ?actor_name
  }

SERVICE <http://dbpedia.org/sparql> {
    ?actor2 a dbpedia:Actor .
            foaf:name ?actor_name_en .
            dbpedia:birthDate ?birth_date .
    FILTER(STR(?actor_name_en) = ?actor_name)
  }
}
```

The SERVICE keyword is used to send part of a query against a remote SPARQL endpoint.

- The BIND is necessary because names in dbpedia have language tags, while names in LinkedMDB do not.

Find birthday of all actors in Star Trek

```
PREFIX movie: <http://data.linkedmdb.org/resource/movie/>
PREFIX dbpedia: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?actor_name ?birth_date
FROM <http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf> # placeholder graph

WHERE {
  SERVICE <http://data.linkedmdb.org/sparql> {
    <http://data.linkedmdb.org/resource/film/675> movie:actor ?actor .
    ?actor movie:actor_name ?actor_name
  }

SERVICE <http://dbpedia.org/sparql> {
    ?actor2 a dbpedia:Actor .
            foaf:name ?actor_name_en .
            dbpedia:birthDate ?birth_date .
    FILTER(STR(?actor_name_en) = ?actor_name)
  }
}
```

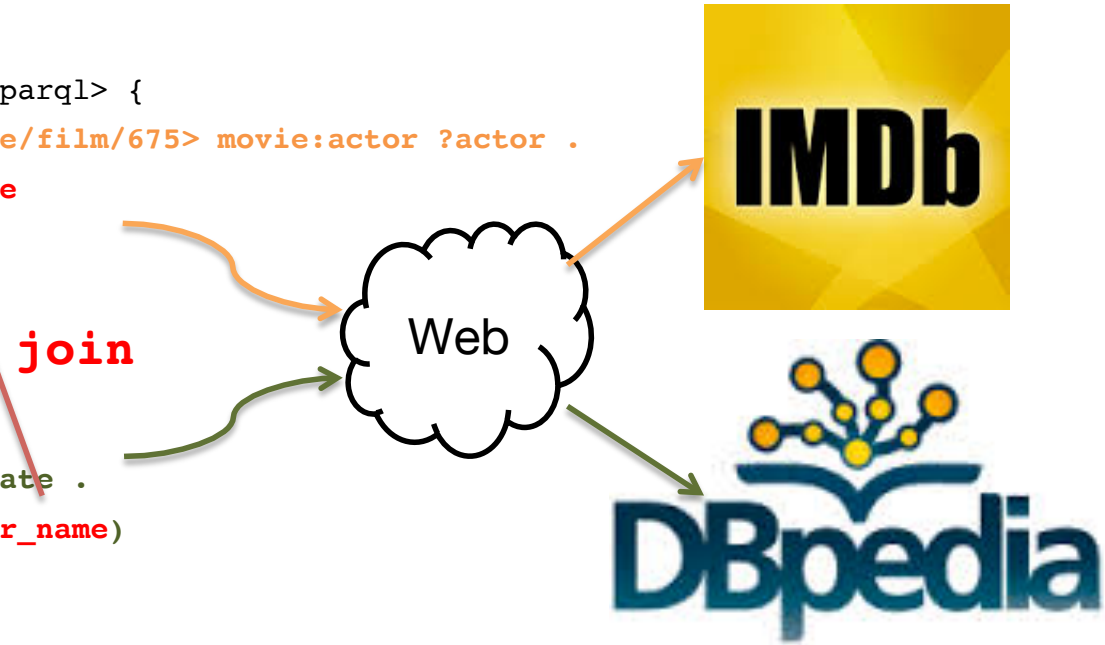


The SERVICE keyword is used to send part of a query against a remote SPARQL endpoint.

- The BIND is necessary because names in dbpedia have language tags, while names in LinkedMDB do not.

Find birthday of all actors in Star Trek

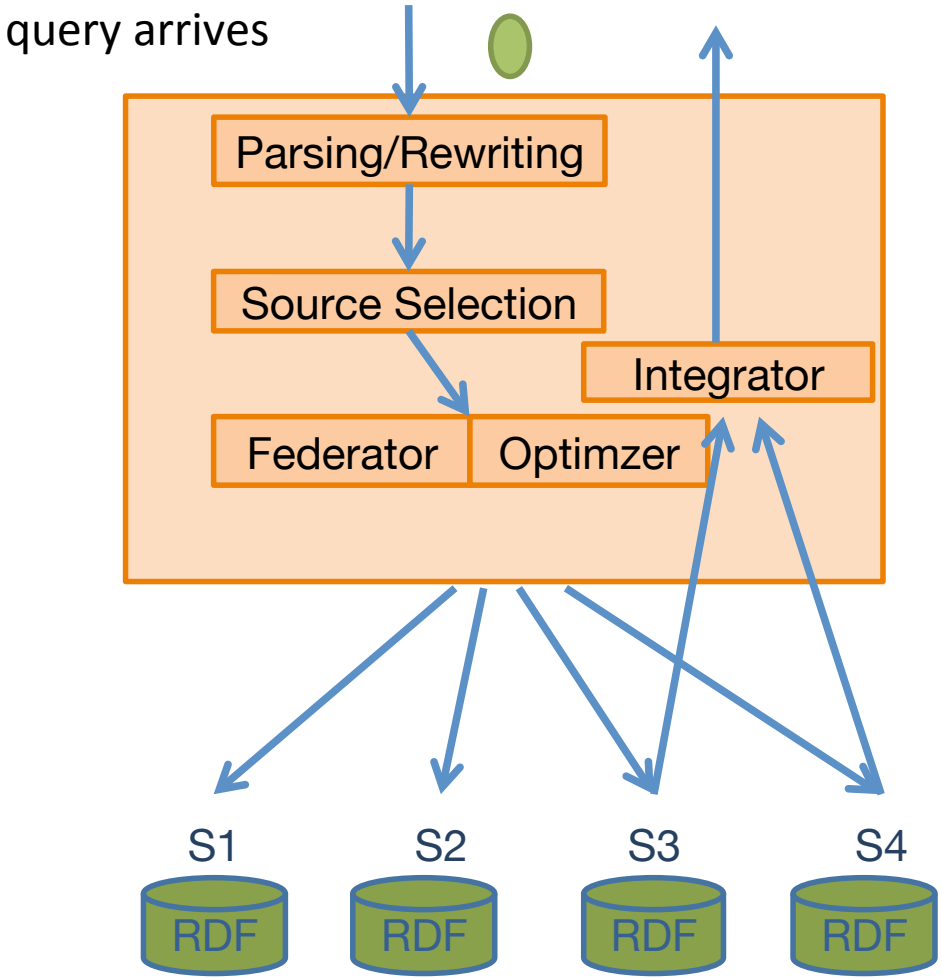
```
PREFIX movie: <http://data.linkedmdb.org/resource/movie/>
PREFIX dbpedia: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?actor_name ?birth_date
FROM <http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf> # placeholder graph

WHERE {
  SERVICE <http://data.linkedmdb.org/sparql> {
    <http://data.linkedmdb.org/resource/film/675> movie:actor ?actor .
    ?actor movie:actor_name ?actor_name
  }

SERVICE <http://dbpedia.org/sparql> {
    ?actor2 a dbpedia:Actor .
            foaf:name ?actor_name_en .
            dbpedia:birthDate ?birth_date .
    FILTER(STR(?actor_name_en) = ?actor_name)
  }
}
```



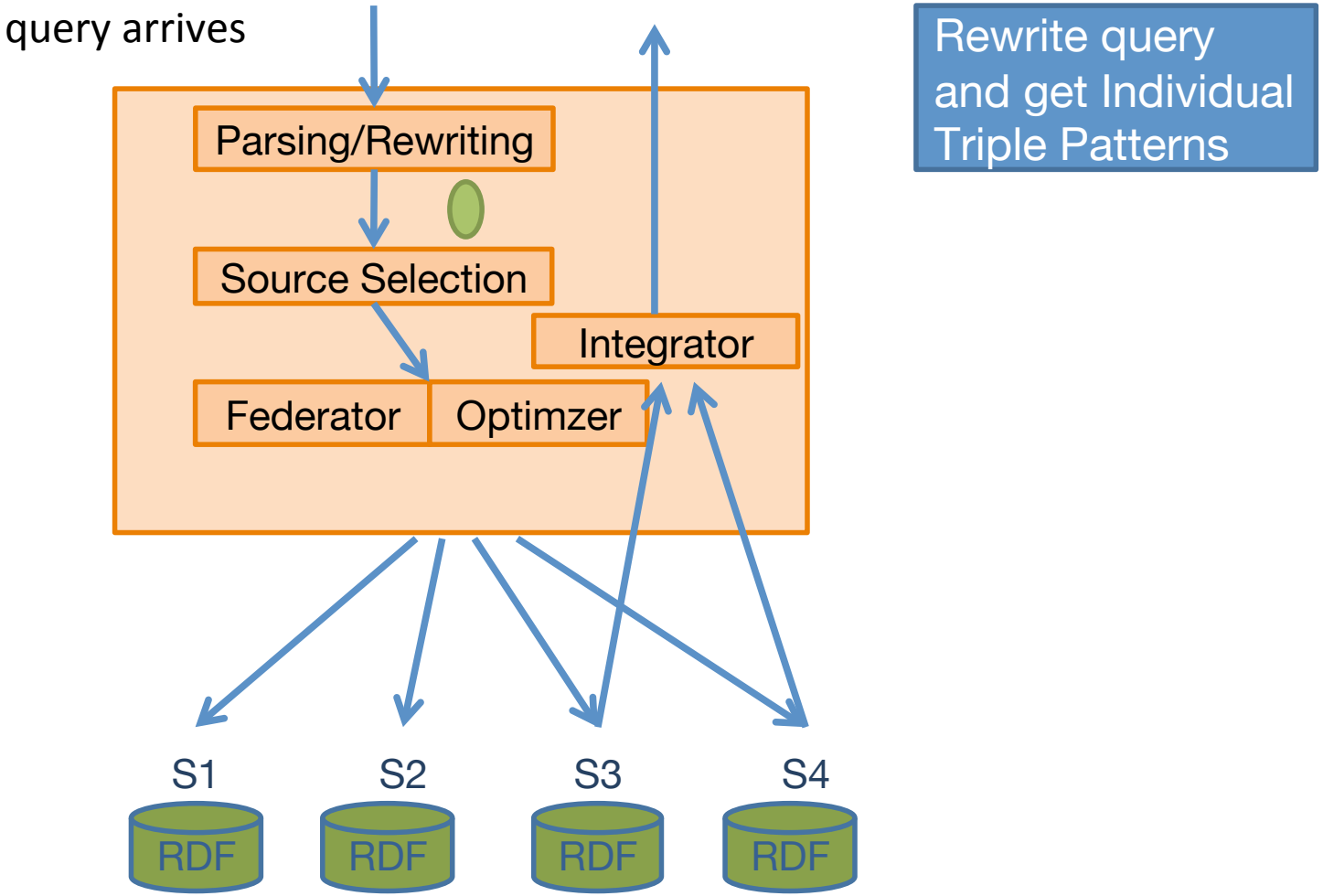
The SERVICE keyword is used to send part of a query against a remote SPARQL endpoint.

- The BIND is necessary because names in dbpedia have language tags, while names in LinkedMDB do not.

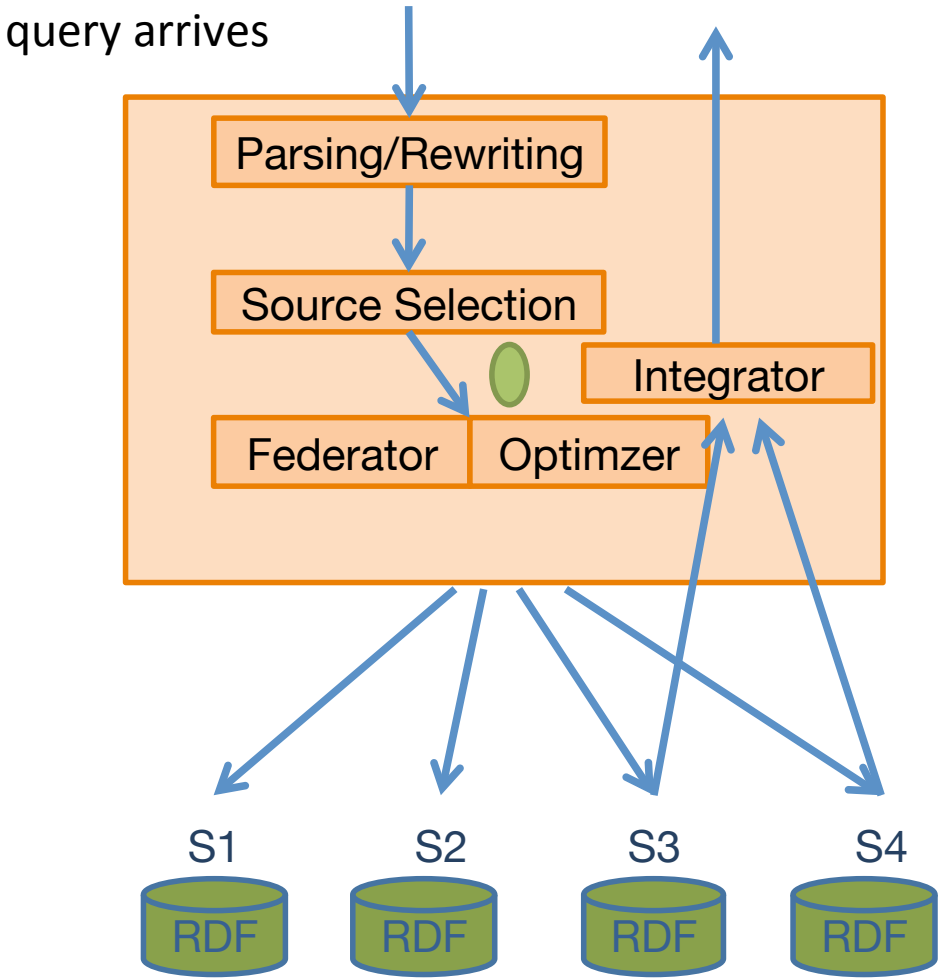
SPARQL Endpoint Federation



SPARQL Endpoint Federation



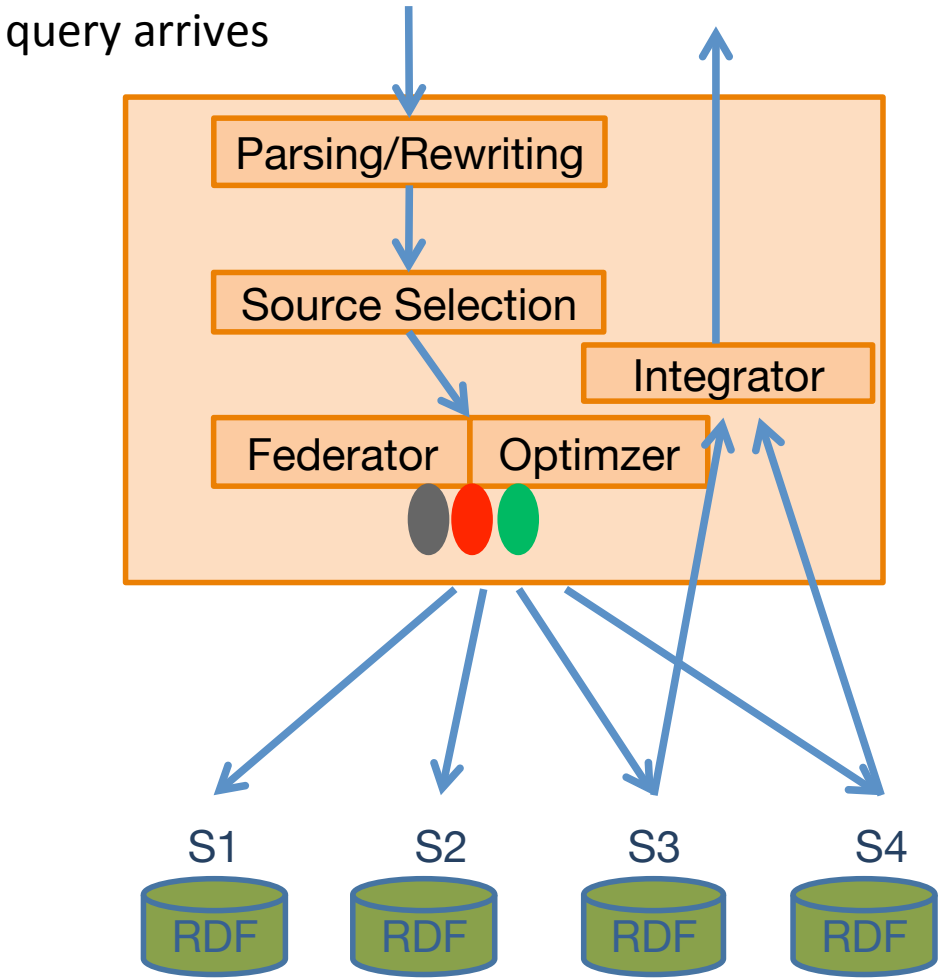
SPARQL Endpoint Federation



Rewrite query and get Individual Triple Patterns

Identify capable source against Individual Triple Patterns

SPARQL Endpoint Federation

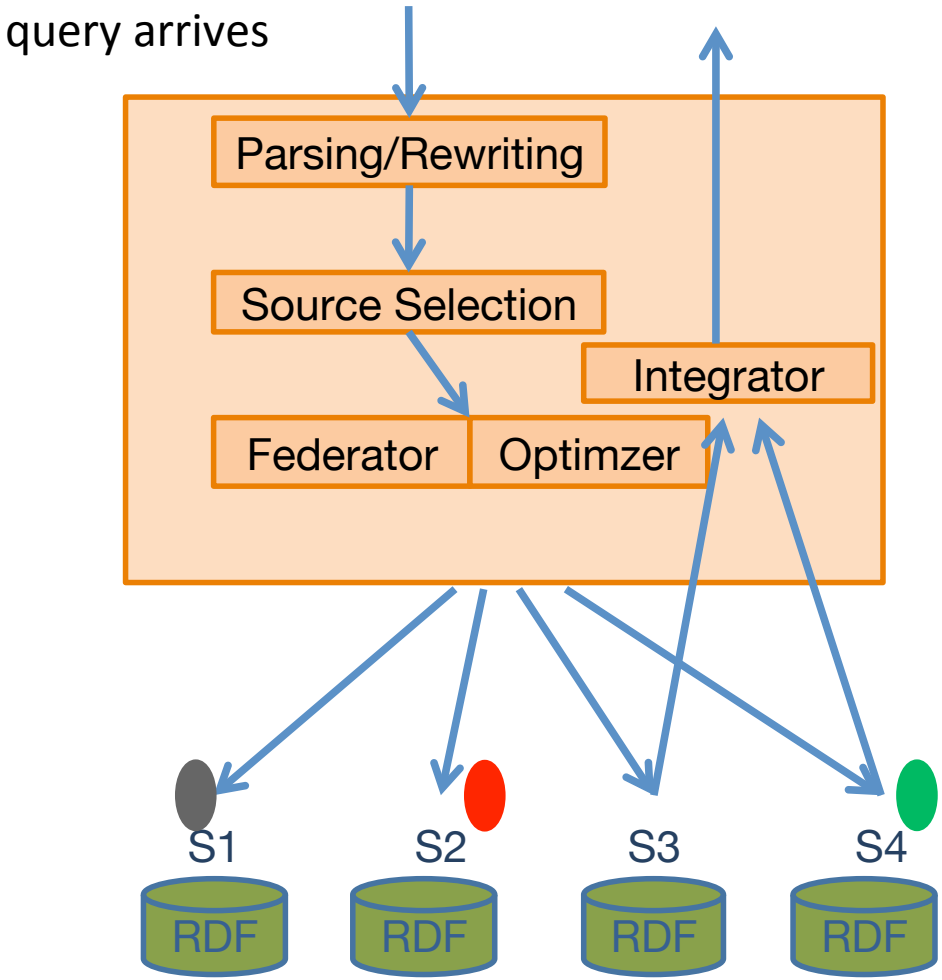


Rewrite query and get Individual Triple Patterns

Identify capable source against Individual Triple Patterns

Generate optimized sub-query Exe. Plan

SPARQL Endpoint Federation



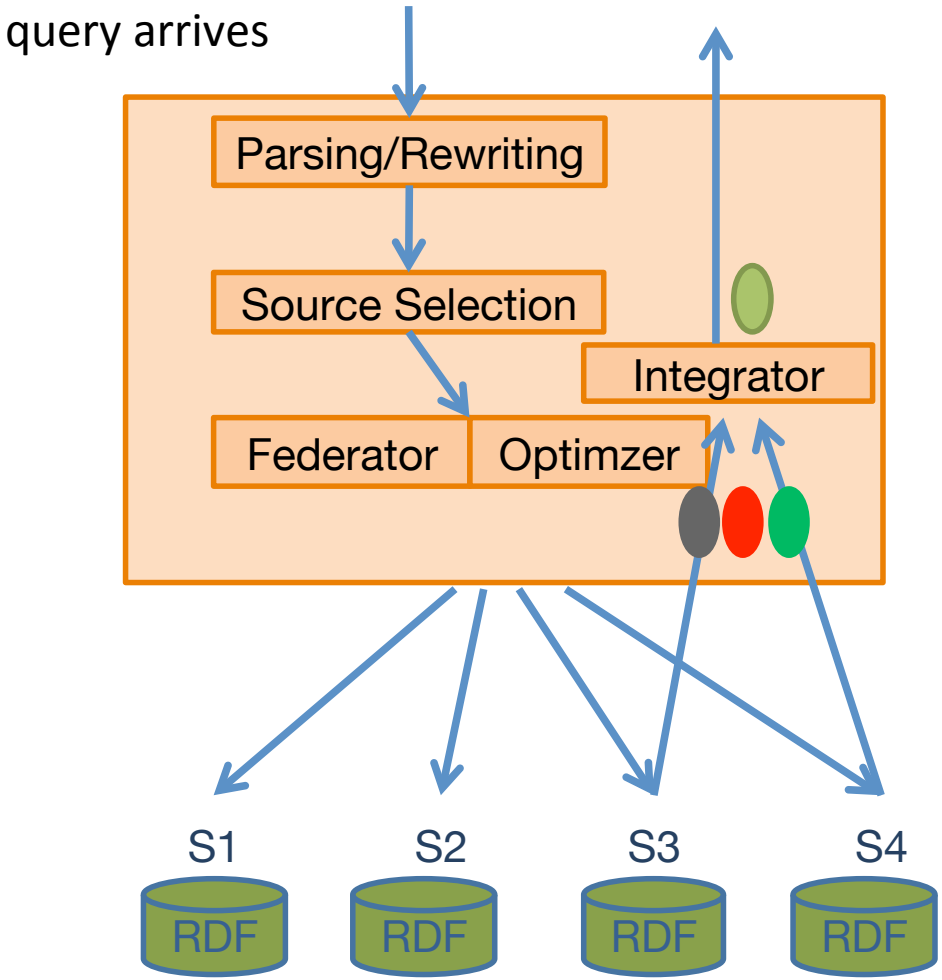
Rewrite query and get Individual Triple Patterns

Identify capable source against Individual Triple Patterns

Generate optimized sub-query Exe. Plan

Execute sub-queries

SPARQL Endpoint Federation



Rewrite query and get Individual Triple Patterns

Identify capable source against Individual Triple Patterns

Generate query execution Plan

Execute sub-queries

Ship-back & Integrate results

SUMMING UP

Basics

URIs

Write full URIs:

```
<http://this.is.a/full/URI/written#out>
```

Abbreviate URIs with prefixes:

```
PREFIX foo: <http://this.is.a/URI/prefix#>
… foo:bar …
```

⇨ http://this.is.a/URI/prefix#bar

Shortcuts:

a ⇨ rdf:type

Literals

Plain literals:

```
"a plain literal"
```

Plain literal with language tag:

```
"bonjour"@fr
```

Typed literal:

```
"13"^^xsd:integer
```

Shortcuts:

```
true ⇨ "true"^^xsd:boolean
3    ⇨   "3"^^xsd:integer
4.2  ⇨ "4.2"^^xsd:decimal
```

Variables

Variables:

```
?var1, ?anotherVar, ?and_one_more
```

Comments

Comments:

```
# Comments start with a '#' and
# continue to the end of the line
```

Triple Patterns

Match an exact RDF triple:

```
ex:myWidget ex:partNumber "XY24Z1" .
```

Match one variable:

```
?person foaf:name "Lee Feigenbaum" .
```

Match multiple variables:

```
conf:SemTech2009 ?property ?value .
```

4 Types of SPARQL Queries

SELECT queries

Project out specific variables and expressions:

```
SELECT ?c ?cap (1000 * ?people AS ?pop)
```

Project out all variables:

```
SELECT *
```

Project out distinct combinations only:

```
SELECT DISTINCT ?country
```

Results in a table of values (in XML or JSON):

?c	?cap	?pop
ex:France	ex:Paris	63,500,000
ex:Canada	ex:Ottawa	32,900,000
ex:Italy	ex:Rome	58,900,000

CONSTRUCT queries

Construct RDF triples/graphs:

```
CONSTRUCT {
   ?country a ex:HolidayDestination .
       ex:arrive_at ?capital .
       ex:population ?population .
}
```

Results in RDF triples (in any RDF serialization):

```
ex:France a ex:HolidayDestination .
    ex:arrive_at ex:Paris .
    ex:population 635000000 .
ex:Canada a ex:HolidayDestination .
    ex:arrive_at ex:Ottawa .
    ex:population 329000000 .
```

ASK queries

Ask whether or not there are any matches:

```
ASK
```

Result is either "true" or "false" (in XML or JSON):

```
true, false
```

DESCRIBE queries

Describe the resources matched by the given variables:

```
DESCRIBE ?country
```

Result is RDF triples (in any RDF serialization) :

```
ex:France a geo:Country .
  ex:continent geo:Europe .
  ex:flag <http://…/flag-france.png> .
  …
```

Category	Functions / Operators	Examples
Logical & Comparisons	`!, &&, ||, =, !=, <, <=, >, >=, IN, NOT IN`	`?hasPermit || ?age < 25`
Conditionals *(SPARQL 1.1)*	`EXISTS, NOT EXISTS, IF, COALESCE`	`NOT EXISTS { ?p foaf:mbox ?email }`
Math	`+, -, *, /, abs, round, ceil, floor, RAND`	`?decimal * 10 > ?minPercent`
Strings *(SPARQL 1.1)*	`STRLEN, SUBSTR, UCASE, LCASE, STRSTARTS, CONCAT, STRENDS, CONTAINS, STRBEFORE, STRAFTER`	`STRLEN(?description) < 255`
Date/time *(SPARQL 1.1)*	`now, year, month, day, hours, minutes, seconds, timezone, tz`	`month(now()) < 4`
SPARQL tests	`isURI, isBlank, isLiteral, isNumeric, bound`	`isURI(?person) || !bound(?person)`
Constructors *(SPARQL 1.1)*	`URI, BNODE, STRDT, STRLANG, UUID, STRUUID`	`STRLANG(?text, "en") = "hello"@en`
Accessors	`str, lang, datatype`	`lang(?title) = "en"`
Hashing *(1.1)*	`MD5, SHA1, SHA256, SHA512`	`BIND(SHA256(?email) AS ?hash)`
Miscellaneous	`sameTerm, langMatches, regex, REPLACE`	`regex(?ssn, "\\d{3}-\\d{2}-\\d{4}")`

Some Public SPARQL Endpoints

Name	URL	What's there?
SPARQLer	http://sparql.org/sparql.html	General-purpose query endpoint for Web-accessible data
DBPedia	http://dbpedia.org/sparql	Extensive RDF data from Wikipedia
DBLP	http://www4.wiwiss.fu-berlin.de/dblp/snorql/	Bibliographic data from computer science journals and conferences
LinkedMDB	http://data.linkedmdb.org/sparql	Films, actors, directors, writers, producers, etc.
World Factbook	http://www4.wiwiss.fu-berlin.de/factbook/snorql/	Country statistics from the CIA World Factbook
bio2rdf	http://bio2rdf.org/sparql	Bioinformatics data from around 40 public databases

TD/TP

- Comprendre l'intérrogation des données RDF

- Nouveautés
  - L'utilisation d'un EndPoint
  - La découverte de son contenut
  - Syntaxe SPARQL et requêtes chemins

Name of Tim's friends without mail ?

MINUS & NOT EXISTS

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?name ?url

FROM
   <http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf>

WHERE {
       ?person  rdf:type   foaf:Person .

  MINUS

       { ?person foaf:mbox ?mail  }
}
```

- MINUS removes variable bindings that match the second triple pattern

Name of Tim's friends without mail ?

MINUS & NOT EXISTS

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?name ?url

FROM
   <http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf>

WHERE {
        ?person  rdf:type   foaf:Person .

  FILTER(NOT EXISTS(?person foaf:mbox ?mail))


}
```