

# A Generic Mapping-based Query Translation from SPARQL to Various Target Database Query Languages

F. Michel, C. Faron-Zucker, J. Montagnat  
I3S laboratory, CNRS, Univ. Nice Sophia



# Towards a Web of Data

---

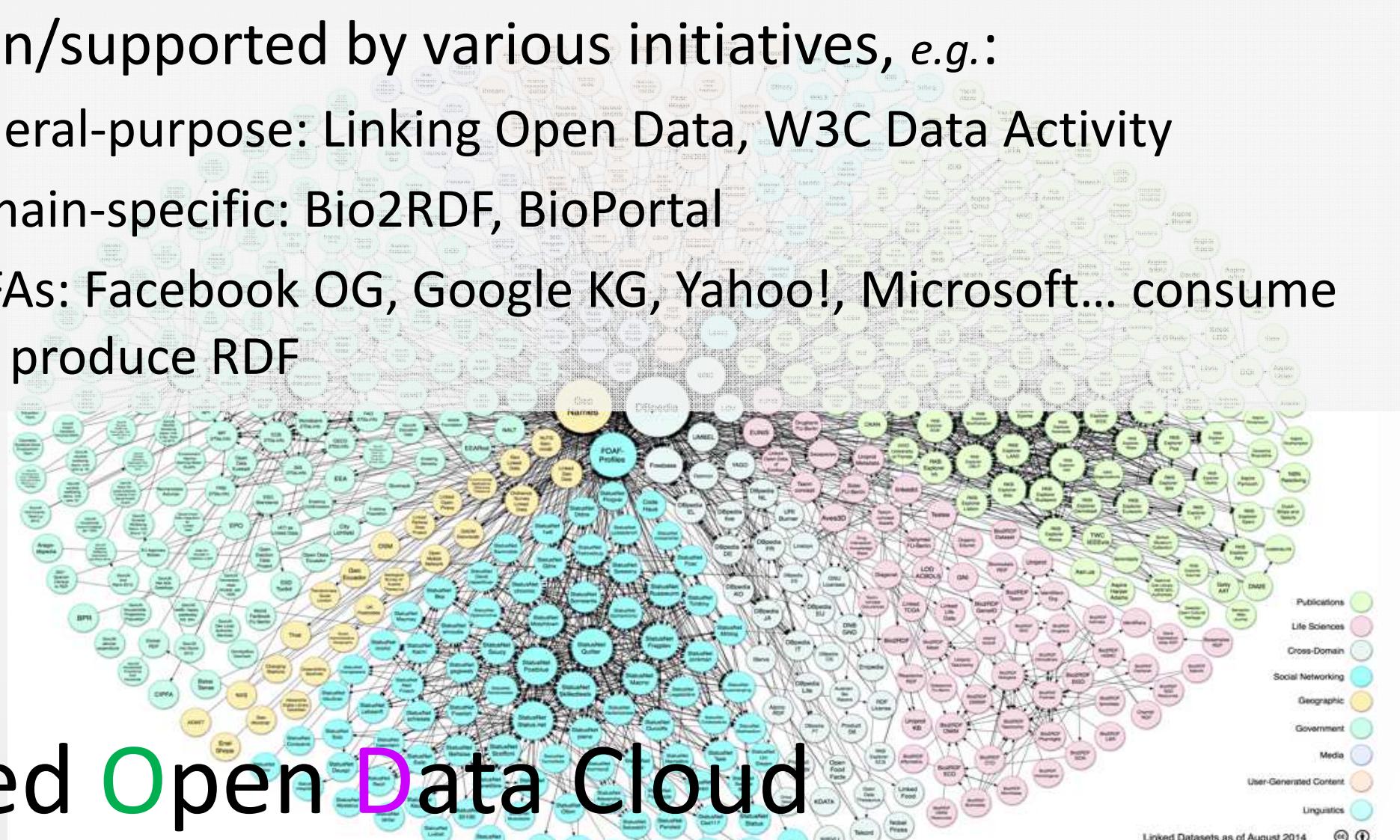
## From a Web of Documents ...to a Web of (Linked) Data

- Publication/interlinking of open datasets
  - In a **common machine-readable format**
  - Using **common vocabularies**
- Linking data increases its value
  - Produce new knowledge
  - Mash up with related data
  - Opportunity for new (unexpected) usage
- Citizenship demand for access to public data (scientific, government...)

# Towards a Web of Data

- Driven/supported by various initiatives, e.g.:
  - General-purpose: Linking Open Data, W3C Data Activity
  - Domain-specific: Bio2RDF, BioPortal
  - GAFAs: Facebook OG, Google KG, Yahoo!, Microsoft... consume and produce RDF

## Linked Open Data Cloud

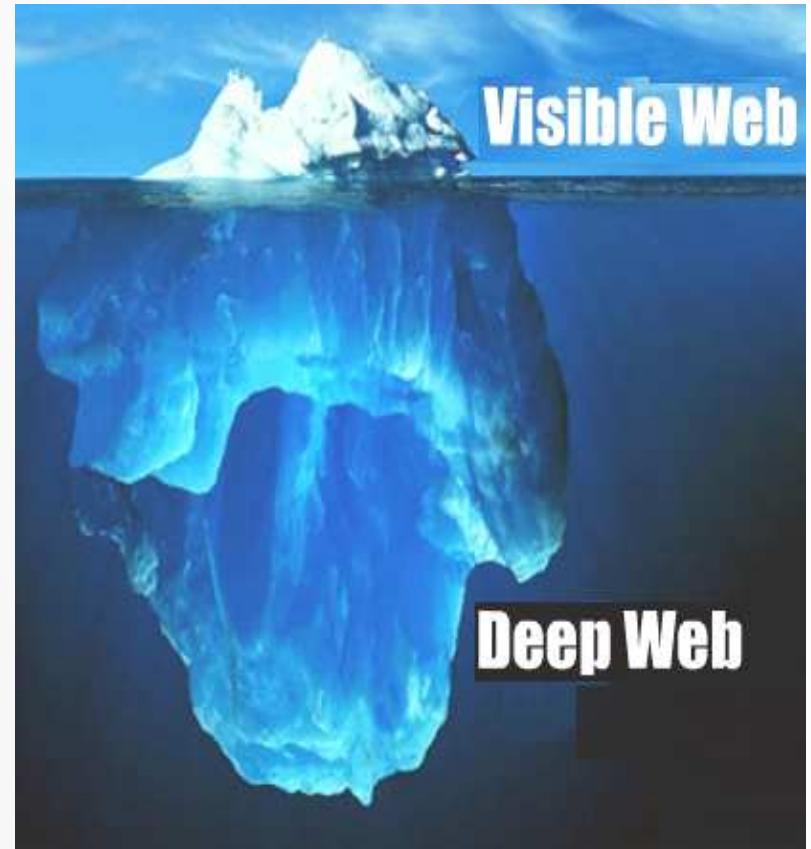


Linked Datasets as of Aug. 30<sup>th</sup> 2014. (c) R. Cyganiak & and A. Jentzsch

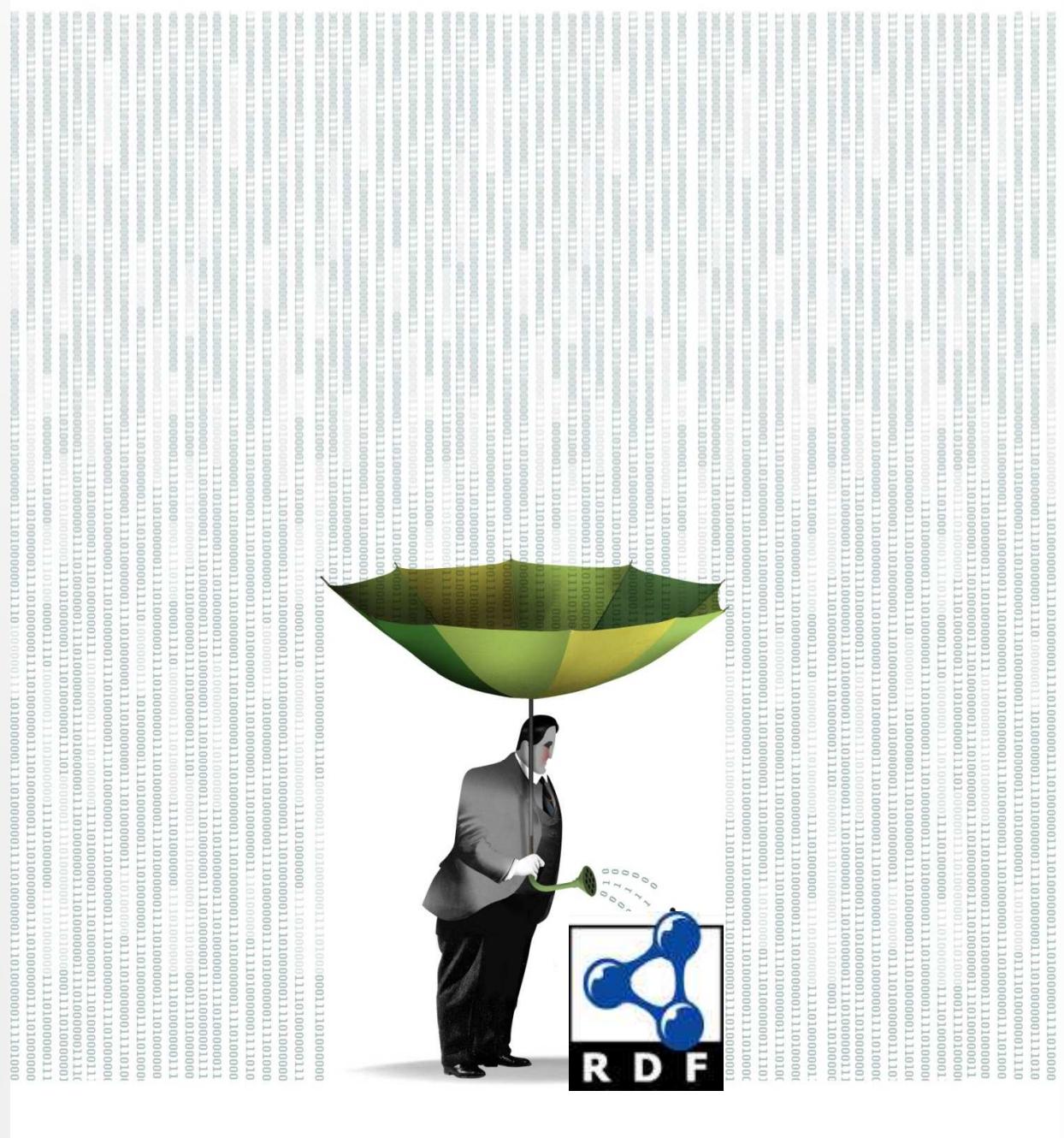
# Web-scale data integration

---

- Need to access data from the **Deep Web**
  - Strd./unstrd. data  
hardly indexed by search engines,  
hardly linked with other data sources
- Exponential data growth goes in
  - Various types of DBs:  
RDB, Native XML, LDAP directory,  
OODB, NoSQL, NewSQL, ...
  - Heterogeneous data models and  
query capabilities
- Whatever the type of DB... it can  
be of interest for the Web of Data  
... “Raw Data Now” (T. Berners Lee)



# Populate the Web of Data with Legacy Data

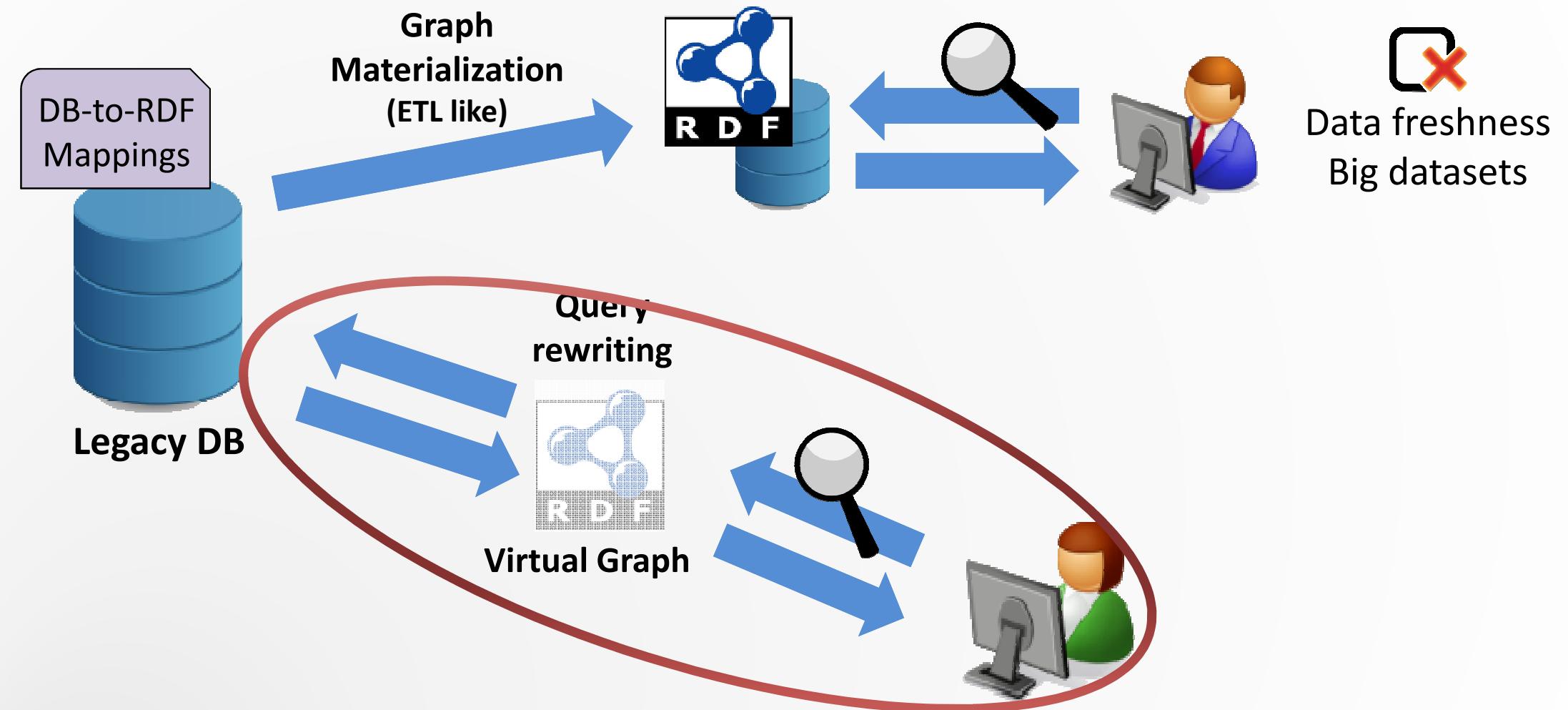


# Previous works

---

- Focused on data formats
  - [HTML](#): RDFa, Microformats
  - [XML](#): Using XPath (RML), XQuery (XSPARQL, SPARQL2XQuery), XSLT (Scissor-Lift, GRDDL), XSD-to-OWL (SPARQL2XQuery)
  - [CSV/TSV/Spreadsheets](#): CSV on the web (W3C WG)
  - [JSON](#): using JSONPath (RML), JSON-LD
- Focused on types of database
  - Extensive work on [RDBs](#): D2RQ, Virtuoso, R2RML...
  - XML native DBs: SPARQL2XQuery
  - [NoSQL](#) stores: [xR2RML](#)
- Integration frameworks: DataLift, RML, Asio Tool Suite...

# Previous works



# SPARQL rewriting in the general case

---

Goal:

Enable SPARQL access to a large range of heterogeneous databases

Previous works: SPARQL rewriting **closely coupled** with the **target QL expressiveness** (SQL, XQuery):  
support of joins, unions, nested queries, filtering, string manipulation etc.

Solution proposed: two-steps approach

1. Translate SPARQL into a pivot Abstract Query Language (AQL) under “*target DB-to-RDF*” mappings: generic mapping language needed
2. Translate from the Abstract QL to the QL of the target database

# SPARQL rewriting in the general case

---

Goal:

Enable SPARQL access to a large range of heterogeneous databases

Previous works: SPARQL rewriting **closely coupled** with the **target QL expressiveness** (SQL, XQuery):  
support of joins, unions, nested queries, filtering, string manipulation etc.

Solution proposed: two-steps approach

1. Translate SPARQL into a pivot Abstract Query Language (AQL) under “*target DB-to-RDF*” mappings: generic mapping language needed
2. Translate from the Abstract QL to the QL of the target database

# Agenda

---



The xR2RML mapping language

The SPARQL translation method

Application

Conclusions & perspectives

# Agenda

---



The xR2RML mapping language

The SPARQL translation method

Application

Conclusions & perspectives

# The xR2RML mapping language

---

- Describe mappings from various types of DB to RDF
  - Query the target database
  - Pick data elements from query results
  - Translate them to (subject, predicate, object) using arbitrary ontologies
- Independent of any target database
  - Allow **any declarative query language**
  - Allow **any syntax to reference data elements** within query results (column name, JSONPath, XPath, attribute name...)
- Extends W3C R2RML (*backward compatible*) and RML
- Turtle RDF Syntax
- Mapping graph = set of “triples maps” ~ mappings

# The xR2RML mapping language: example

```
{ "id": 106, "firstname": "John",  
  "emails": ["john@foo.com", "john@example.org"],  
  "contacts": ["chris@example.org", "alice@foo.com"] }
```



```
<#Mbox> a rr:TriplesMap;  
  xrr:logicalSource [ xrr:query "db.people.find({'emails':{$ne: null}})" ];  
  rr:subjectMap [ rr:template "http://example.org/member/{$id}" ];  
  rr:predicateObjectMap [  
    rr:predicate foaf:mbox;  
    rr:objectMap [ xrr:reference "$.emails.*"; rr:termType rr:Literal ] ].
```

xR2RML

```
<http://example.org/member/106> foaf:mbox "john@foo.com".  
<http://example.org/member/106> foaf:mbox "john@example.org".
```



# The xR2RML mapping language: example

```
{ "id": 106, "firstname": "John",  
  "emails": ["john@foo.com", "john@example.org"],  
  "contacts": ["chris@example.org", "alice@foo.com"] }
```



```
{ "id": 327, "firstname": "Alice",  
  "emails": ["alice@foo.com"],  
  "contacts": ["john@foo.com"] }
```



```
<#Knows> a rr:TriplesMap;  
  
xrr:logicalSource [ xrr:query "db.people.find({'contacts':{$size: {$gte:1}}})" ];  
rr:subjectMap [ rr:template "http://example.org/member/{$.id}" ];  
rr:predicateObjectMap [  
  rr:predicate foaf:knows;  
  rr:objectMap [  
    rr:parentTriplesMap <#Mbox>;  
    rr:joinCondition [ rr:child ".$.contacts.*"; rr:parent ".$.emails.*" ] ] ].
```

xR2RML

```
<http://example.org/member/106> foaf:knows <http://example.org/member/327>.  
<http://example.org/member/327> foaf:knows <http://example.org/member/106>.
```



# Agenda

---



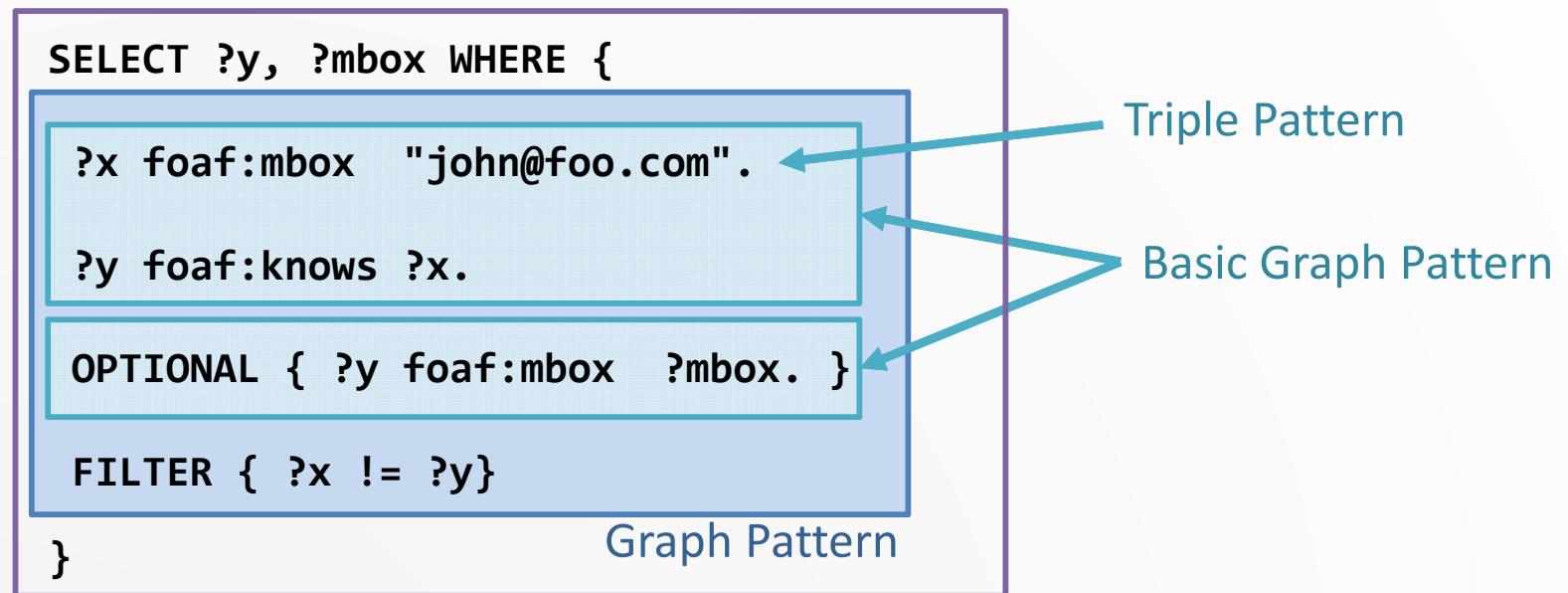
The xR2RML mapping language

The SPARQL translation method

Application

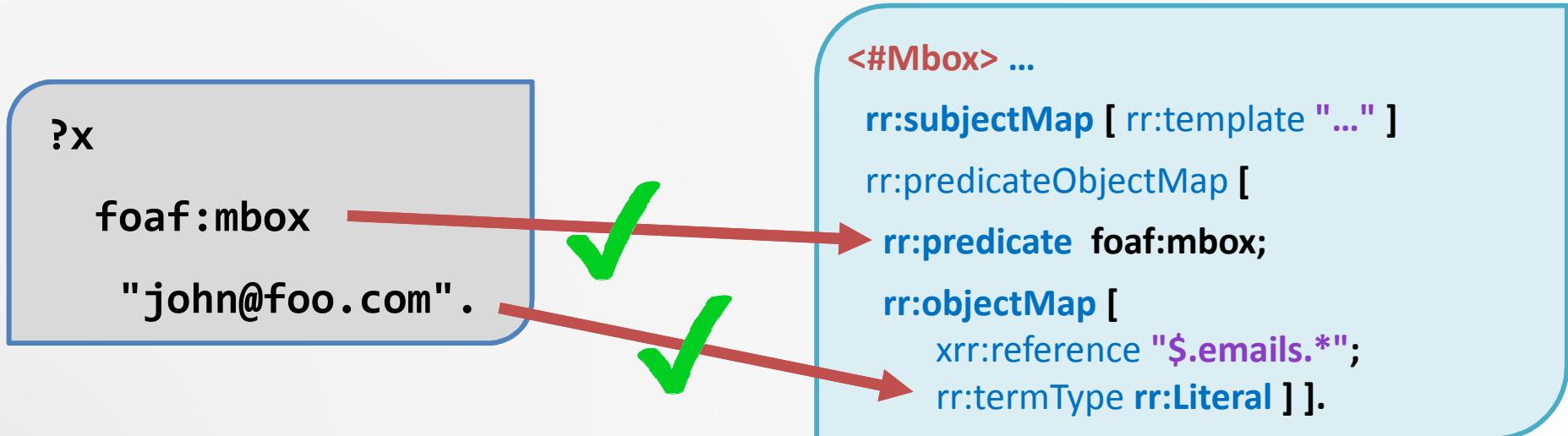
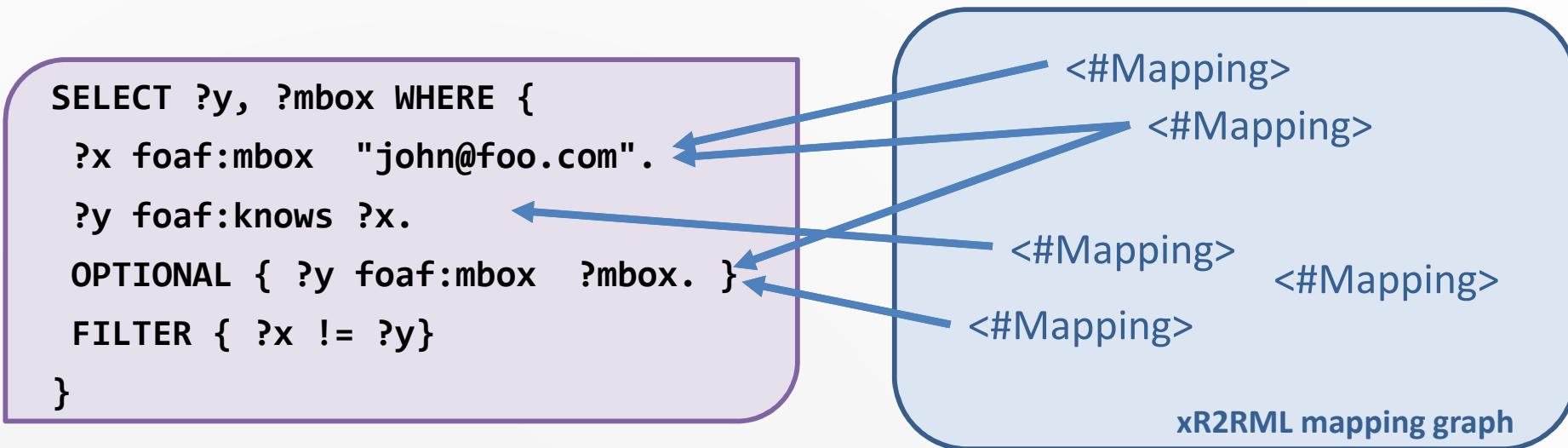
Conclusions & perspectives

# SPARQL-to-AQL rewriting steps



- 1. Triple Pattern Bindings:** figure out minimal set candidate mappings for each triple pattern
- 2. Rewrite the SPARQL Graph Pattern into the AQL, under triple pattern bindings, entail conditions**
- 3. Optimization the resulting Abstract Query**

# (1) Triples patterns bindings



# (1) Triples patterns bindings

## 1. Initial set of mappings for each triple pattern

- Check compatibility: term type, datatype, lang
- Check unsatisfiable SPARQL filter constraints about a terms type, data type, language: *isIRI*, *isLiteral*, *isBlank*, *lang()*, *datatype()*...  
e.g. “rr:termType rr:Literal” does not match “*isIRI*(?var)”

## 2. Reduce bindings

- Consider join constraints implied by shared variables

```
SELECT ?x WHERE {
  ?y foaf:mbox "john@foo.com".//tp2
  ?x foaf:knows ?y.                                //tp3
}
```

Bindings:

(tp2, <#Mbox>)  
(tp3, <#Knows>)

Shared variable **?y** → compatibility between  
<#Mbox>'s subject map  
and  
<#Knows>'s object map

M. Rodríguez-Muro, M. Rezk. *Efficient SPARQL-to-SQL with R2RML mappings*, Web Semant. Sci. Serv. Agents World Wide Web. 33 (2015) 141–169.  
J. Unbehauen, C. Stadler, S. Auer. *Accessing relational data on the web with SparqlMap*, in: Semantic Technol., Springer, 2013: pp. 65–80.

## (2) Rewrite each Triple Pattern

---

### Bindings

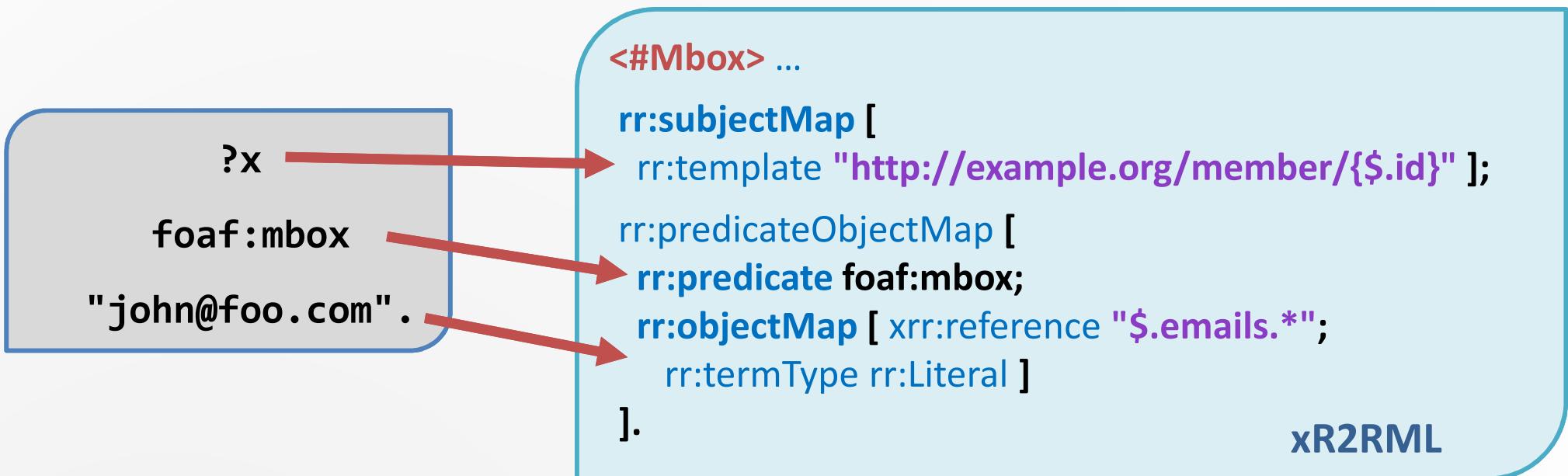
```
(tp1, <#Mbox>)
(tp2, <#Mbox>)
(tp3, <#Knows>)
```

tp2  $\xleftarrow{\text{match}}$  <#Mbox>



*Atomic Abstract Query*  
{ From, Project, Where }

## (2) Rewrite each Triple Pattern



**Condition 1: `$.id != null`**

**Condition 2: `$.emails.*` produces "john@foo.com"**

## (2) Rewrite from SPARQL to the AQL

Usual SPARQL-to-SQL example:

```
SELECT ?x WHERE {  
  ?x foaf:age ?age.  
  ...  
  FILTER (?age > 30)  
}
```



```
SELECT t2.X FROM  
( SELECT t1.ID AS X, t1.AGE AS AGE  
  FROM PERSON t1  
  ...  
 ) AS t2  
 WHERE (t2.AGE > 30)
```

FILTER → encapsulating SELECT WHERE clause

Relies on the DB engine to optimize the query

In the general case, no assumption on the target DB:

→ Need to optimize at the earliest stage:  
**push “down” filter conditions** in the translation of triple patterns

## (2) Rewrite from SPARQL to the AQL

Function  $trans_m$  (*graph pattern, filter*)

Rewrites a well-designed SPARQL graph pattern  
into the AQL, under a set of xR2RML mappings  $m$

$trans_m(P1 \text{ AND } P2, f)$  →  $trans_m(P1, f) \text{ INNER JOIN } trans_m(P2, f) \text{ ON } \text{var}(P1) \cap \text{var}(P2)$

$trans_m(P1 \text{ OPTIONAL } P2, f)$  →  $trans_m(P1, f) \text{ LEFT JOIN } trans_m(P2, f) \text{ ON } \text{var}(P1) \cap \text{var}(P2)$

$trans_m(P1 \text{ UNION } P2, f)$  →  $trans_m(P1, f) \text{ LEFT JOIN } trans_m(P2, f) \text{ ON } \text{var}(P1) \cap \text{var}(P2)$   
 $\text{UNION}$   
 $trans_m(P2, f) \text{ LEFT JOIN } trans_m(P1, f) \text{ ON } \text{var}(P1) \cap \text{var}(P2)$

$trans_m(tp, f)$  →  $transTP_m(tp, sparqlCond(tp, f))$

$trans_m(P \text{ FILTER } f', f)$  →  $trans_m(P, f \&& f') \text{ FILTER } sparqlCond(P, f \&& f')$

$trans_m(P)$  →  $trans_m(P, \text{true})$

***sparqlCond: Push filter conditions*** in the translation of **relevant** triple patterns

## (2) Rewrite from SPARQL to the AQL

Function *spaqlCond()*:

- Push down filter conditions in the translation of triples patterns
- Make inner-queries as selective as possible
- Limit the size of intermediary results

```
SELECT ?x WHERE {  
?x foaf:mbox ?mbox.          // tp1  
?y foaf:mbox "john@foo.com".  // tp2  
?x foaf:knows ?y.            // tp3  
FILTER {  
  contains(str(?mbox), "foo.com") // c1  
  && ?x != ?y                 // c2  
}}
```



```
transTPm(tp1, c1)  
INNER JOIN transTPm(tp2, true) ON {}  
INNER JOIN transTPm(tp3, c2) ON {?x, ?y}  
FILTER c2
```

# Example

```
SELECT ?x WHERE {
  ?x foaf:mbox ?mbox.          // tp1
  ?x foaf:mbox "john@foo.com". // tp2
  FILTER {
    ?mbox != "john@foo.com" }   // c1
}
```

```
{ From: { "db.people.find({'emails':{$ne:null}})" },
  Project: { $.id AS ?x, $.emails.* AS ?mbox },
  Where: { isNotNull($.id), isNotNull($.emails.*), sparqlFilter(?mbox != "john@foo.com") }

INNER JOIN

{ From: { "db.people.find({'emails':{$ne: null}})" },
  Project: { $.id AS ?x },
  Where: { isNotNull($.id), equals($.emails.* , "john@foo.com") } }

ON { ?x }
```

Abstract Query

# (3) Optimization

---

- Abstract Query effective but may be inefficient
  - Unnecessary complexity: multiple joins, unions, redundancy
- Study/reuse of common query optimization techniques
  - **Self-Join / Optional-Self-Join Elimination**
    - When the same mapping is bound to different triple patterns
  - **Self-Union Elimination**
    - When multiple mappings are bound to the same triple pattern
  - **Projection Pushing**

```
SELECT DISTINCT ?p WHERE { ?s ?p ?o }
```
  - **Filter propagation** in joined queries

B. Elliott, E. Cheng, C. Thomas-Ogbugji, Z.M. Ozsoyoglu, A complete translation from SPARQL into efficient SQL, in: Proc. Int. Database Eng. Appl. Symp. 2009, ACM, 2009: pp. 31–42.

M. Rodríguez-Muro, M. Rezk. *Efficient SPARQL-to-SQL with R2RML mappings*, Web Semant. Sci. Serv. Agents World Wide Web. 33 (2015) 141–169.

J. Unbehauen, C. Stadler, S. Auer. *Accessing relational data on the web with SparqlMap*, in: Semantic Technol., Springer, 2013: pp. 65–80.

# Agenda

---



The xR2RML mapping language

The SPARQL translation method

Application

Conclusions & perspectives

# Application: SPARQL-to-MongoDB

---

- Prototype implementation for MongoDB
  - SPARQL-to-AQL implemented as a DB-independent component  
Extendable to other target DBs
  - AQL-to-MongoDB QL
    - Not straightforward due to MongoDB limitations:
      - No join, no nested query, union hardly supported
      - Limited comparison filters, JavaScript filters discouraged
    - Much work falls back on the query processing engine
- Two concrete use cases
  - SKOS representation of a taxonomical reference
  - Biological studies on rice phenotype data



# Agenda

---



The xR2RML mapping language

The SPARQL translation method

Application

Conclusions & perspectives

# Conclusions & perspectives

---

- Goal: foster the development of SPARQL interfaces to heterogeneous databases
- Formalized approach:
  - Generalize existing works on SQL and XQuery
  - Rely on a DB-independent mapping language: xR2RML
  - Encompass all DB-independent steps of the rewriting process
  - Leave only DB-specific rewriting as a last step
- Prototype implementation for MongoDB
  - Used in two real world contexts
- Perspectives
  - Perform benchmarking
  - Use it with distributed SPARQL query engine

# Conclusions & perspectives

---

- SW vs. NoSQL: two un-reconciliable worlds?

Different paradigms:

- SW manages highly connected graphs,
- NoSQL's manage isolated documents, joins hardly supported

NoSQL DBs

- pragmatically gave up on consistency and rich query features
- trade-off to high throughput/availability, horizontal elasticity

- Filling the gap between the two worlds is not straightforward  
The experience of MongoDB shows challenges.

- Huge potential source of LOD, can't be ignored anymore

## Contacts:

Franck Michel

Catherine Faron-Zucker

Johan Montagnat



<https://github.com/frmichel/morph-xr2rml/>

[1] F. Michel, L. Djimenou, C. Faron-Zucker, and J. Montagnat. *Translation of Relational and Non-Relational Databases into RDF with xR2RML*. In proc. of WebIST 2015.

[2] C. Callou, F. Michel, C. Faron-Zucker, C. Martin, J. Montagnat. *Towards a Shared Reference Thesaurus for Studies on History of Zoology, Archaeozoology and Conservation Biology*. In SW4SH workshop, ESWC'15.

[3] F. Michel, C. Faron-Zucker, and J. Montagnat. *Mapping-based SPARQL access to a MongoDB database*. Technical report, CNRS, 2015. <https://hal.archives-ouvertes.fr/hal-01245883v4>.