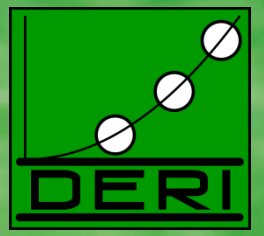
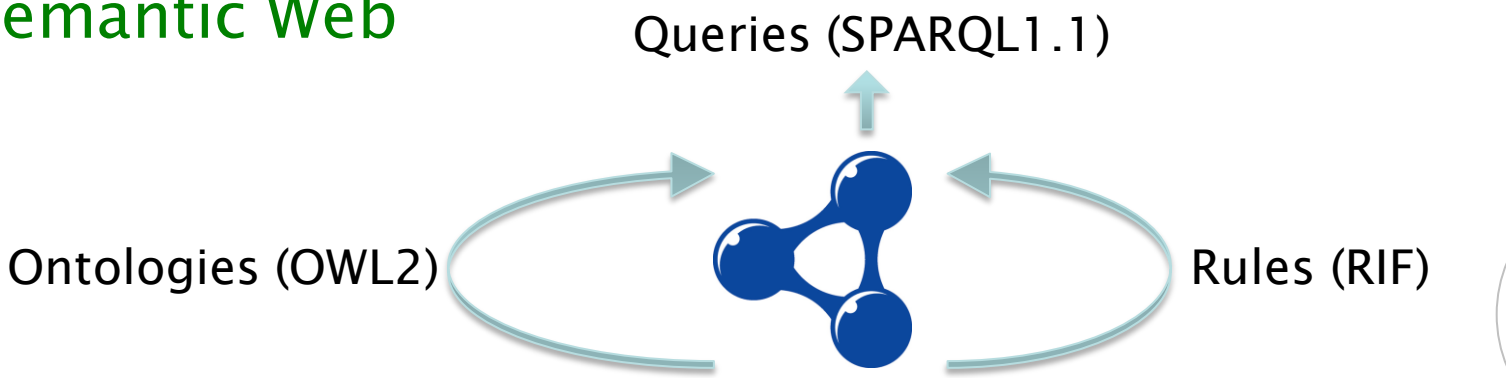


RDF and XML: Towards a Unified Query Layer

Nuno Lopes, Stefan Bischof, Orri Erling, **Axel Polleres**, Alexandre Passant, Diego Berrueta, Antonio Campos, Jérôme Euzenat, Kingsley Idehen, Stefan Decker, Stéphane Corlosquet, Jacek Kopecky, Janne Saarela, Thomas Krennwallner, Davide Palmisano, and Michal Zaremba
(supporters of the W3C member submission)

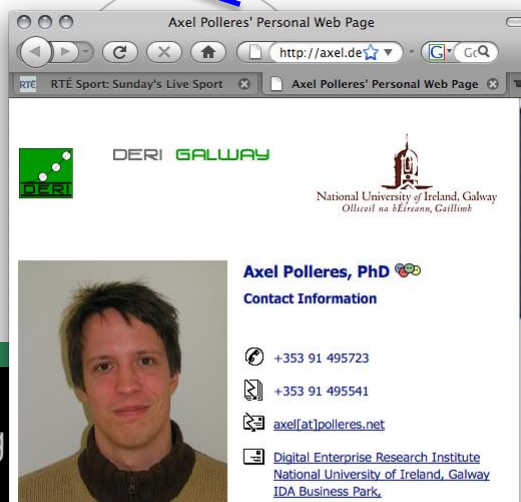
Semantic Data Access…
… that would be the vision

- RDF: common simple Data format underlying the Semantic Web



id	FNAME	LNAME	AGE
1	Bob	McBob	42
2	John	Johnson	24
3	Steve	Smith	38

```
<s1:messages>
 <s1:msg id="m1" format="bin">
  <s1:sendername>3PO</s1:sendername>
  <s1:receivername>R2D2<s1:receivername>
  <s1:payload>0111010001</s1:payload>
 </s1:msg>
<s1:msg id="m2" format="text">
  <s1:sendername>Obiwan</s1:sendername>
  <s1:receivername>Luke<s1:receivername>
  <s1:payload>Use the force</s1:payload>
 </s1:msg>
<s1:messages>
```



Semantic Data Access…
… that would be the vision

Queries

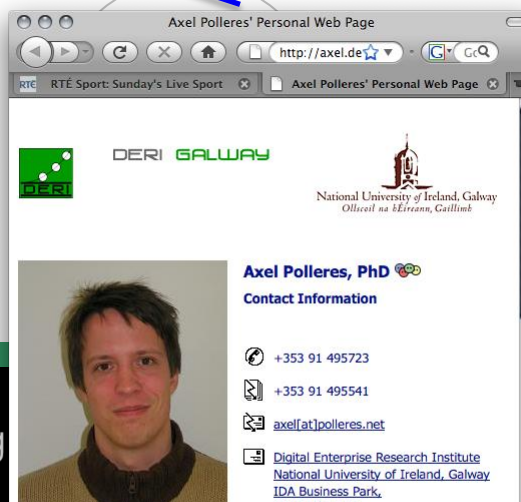
Ontologies

Rules

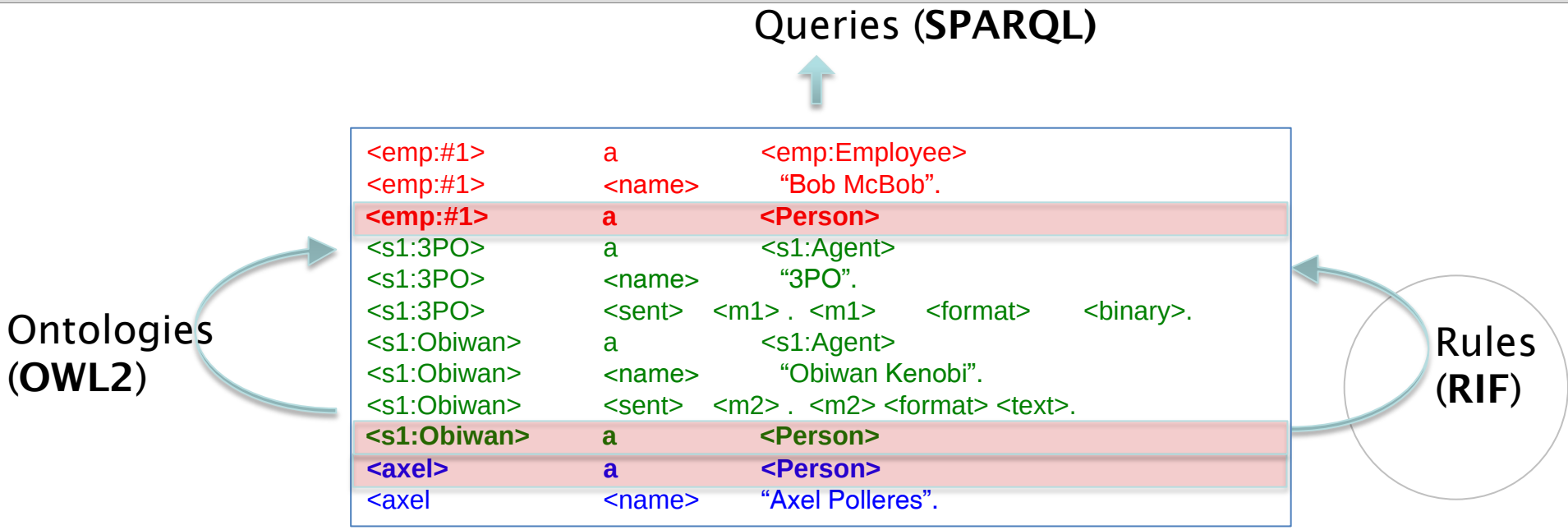
```
<emp:#1>        a          <emp:Employee>
<emp:#1>        <name>       “Bob McBob”.
…
<s1:3PO>        a          <s1:Agent>
<s1:3PO>        <name>       “3PO”.
<s1:3PO>        <sent>   <m1> .  <m1>     <format>     <binary>.
<s1:Obiwan>     a          <s1:Agent>
<s1:Obiwan>     <name>       “Obiwan Kenobi”.
<s1:Obiwan>     <sent>   <m2> .  <m2> <format> <text>.
…
<axel>          a          <Person>
<axel           <name>     “Axel Polleres”.
```

id	FNAME	LNAME	AGE
1	Bob	McBob	42
2	John	Johnson	24
3	Steve	Smith	38

```
<s1:messages>
 <s1:msg id=“m1” format=“bin”>
  <s1:sendername>3PO</s1:sendername>
  <s1:receivername>R2D2<s1:receivername>
  <s1:payload>0111010001</s1:payload>
 </s1:msg>
<s1:msg id=“m2” format=“text”>
  <s1:sendername>Obiwan</s1:sendername>
  <s1:receivername>Luke<s1:receivername>
  <s1:payload>Use the force</s1:payload>
 </s1:msg>
<s1:messages>
```



Semantic Data Access…
… that would be the vision



- **Query (SPARQL)**: "Give me all Persons"

```
SELECT ?X WHERE { ?X a <Person> }
```

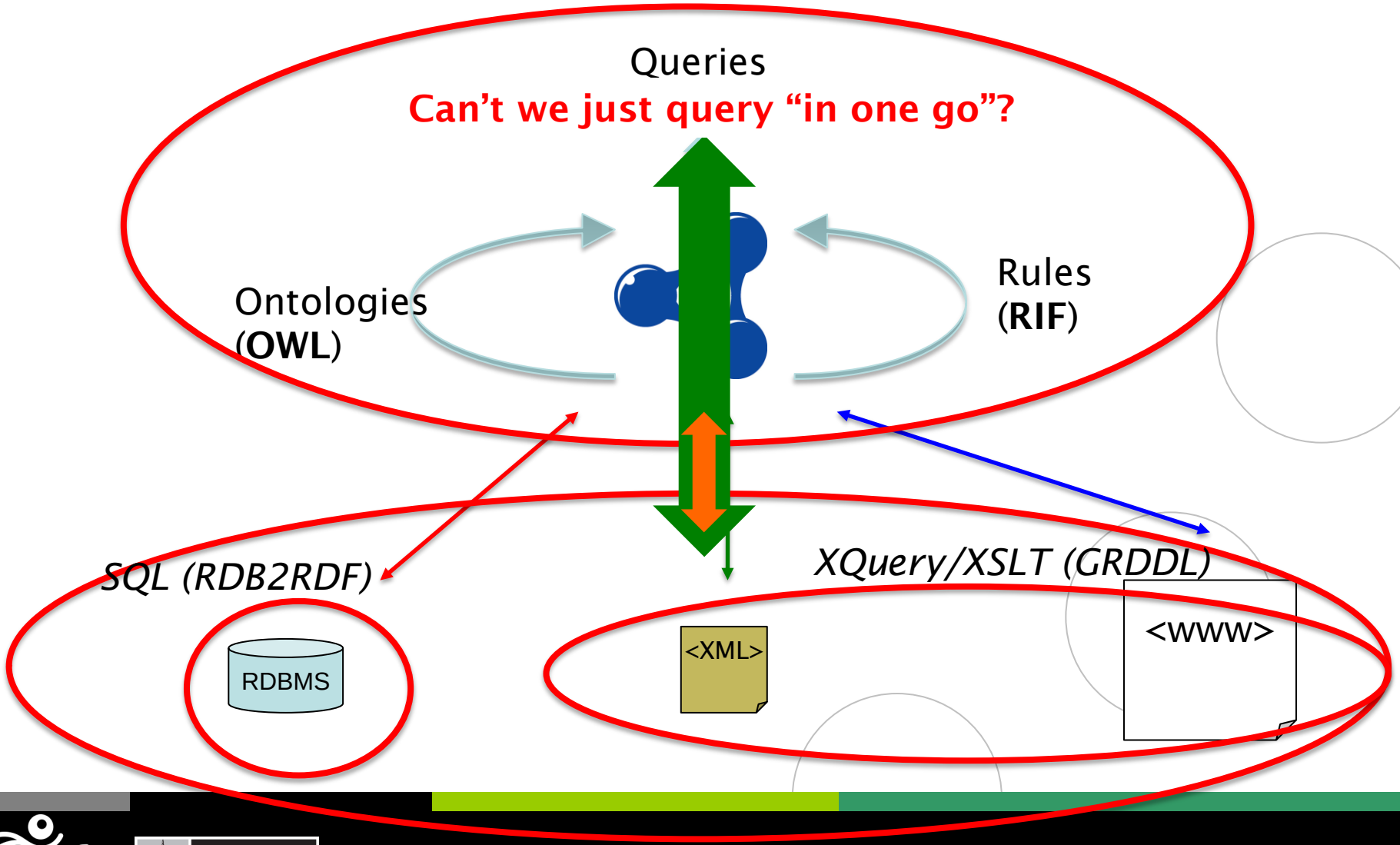
- **Ontologies (OWL2, DL)**: *"<emp:Employee> is a subclass of <Person>"*

$$Employee \sqsubseteq Person$$

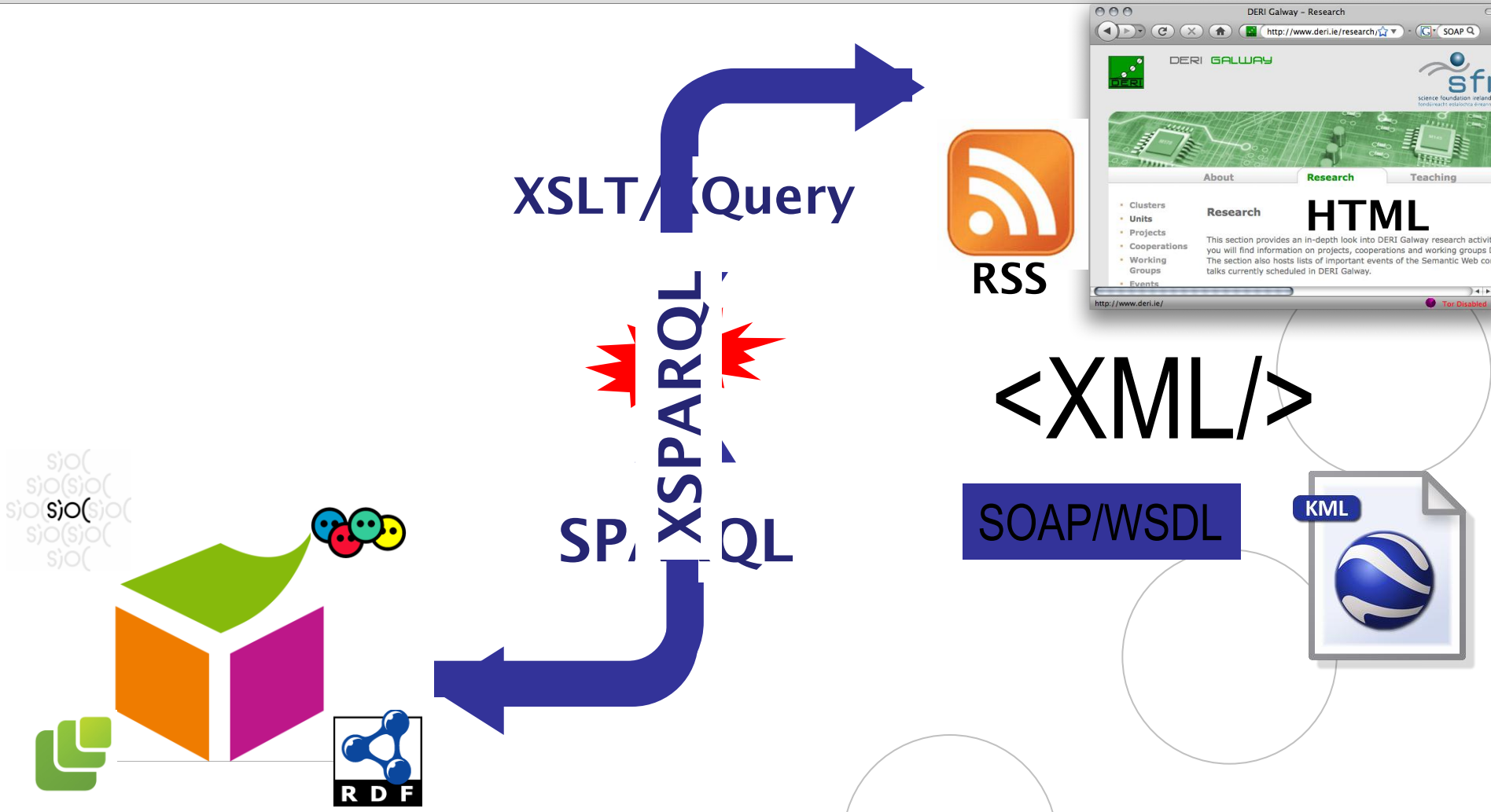
- **Rules (RIF, LP)**: "<s1:Agents> who never sent binary are Persons"

$$Person(x) :- Agent(x), \mathtt{not}\ sent(x, "binary").$$

Bridging Gaps to the data “below”:



Our starting motivation (2008):



Example:

relations.xml

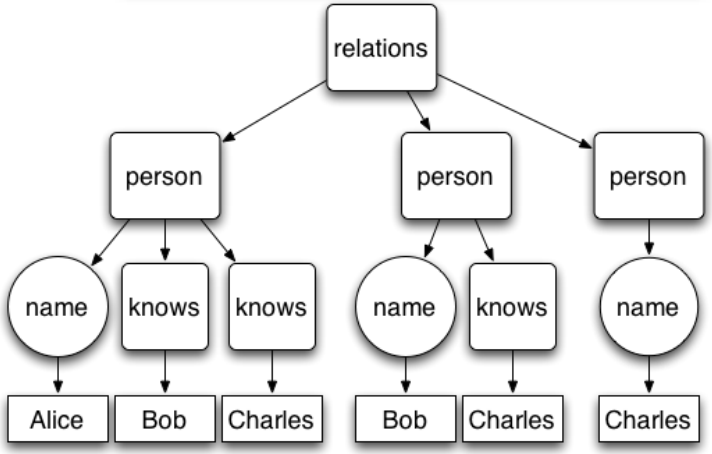
```
<relations>
  <person name="Alice">
    <knows>Bob</knows>
    <knows>Charles</knows>
  </person>
  <person name="Bob">
    <knows>Charles</knows>
  </person>
  <person name="Charles"/>
</relations>
```

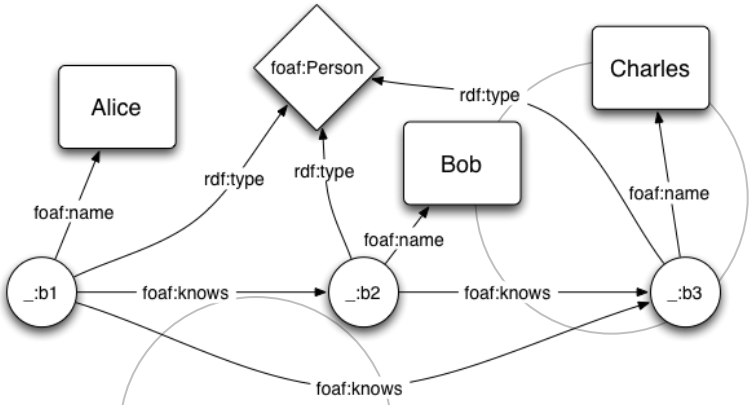
Lowering

Lifting

relations.rdf

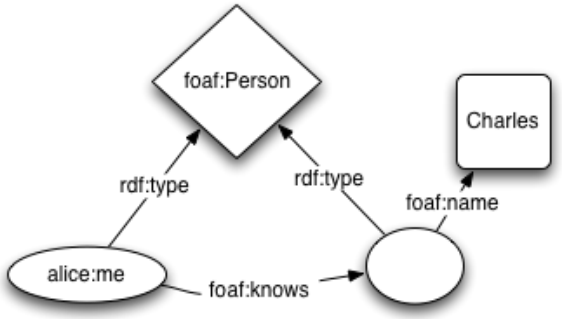
```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:b1 a foaf:Person;
     foaf:name "Alice";
     foaf:knows _:b2;
     foaf:knows _:b3.
_:b2 a foaf:Person; foaf:name "Bob";
     foaf:knows _:b3.
_:b3 a foaf:Person; foaf:name "Charles".
```





So, why are XSLT, XQuery not enough?

- Because RDF ≠ RDF/XML !!!



1) many different RDF/XML representations…

```
<rdf:RDF xmlns:foaf="http://xmlns.com/foaf/0.1/"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <foaf:Person rdf:about="alice/me">
      <foaf:knows>
         <foaf:Person foaf:name="Charles"/>
      </foaf:knows>
   </foaf:Person>
</rdf:RDF>
```

```
<rdf:RDF xmlns:foaf="http://xmlns.com/foaf/0.1/"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <rdf:Description rdf:nodeID="x">
      <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
      <foaf:name>Charles</foaf:name>
   </rdf:Description>
   <rdf:Description rdf:about="alice/me">
      <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
      <foaf:knows rdf:nodeID="x"/>
   </rdf:Description>
</rdf:RDF>
```

```
<rdf:RDF xmlns:foaf="http://xmlns.com/foaf/0.1/"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <rdf:Description rdf:about="alice/me">
      <foaf:knows rdf:nodeID="x"/>
   </rdf:Description>
   <rdf:Description rdf:about="alice/me">
      <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
   </rdf:Description>
   <rdf:Description rdf:nodeID="x">
      <foaf:name>Charles</foaf:name>
   </rdf:Description>
   <rdf:Description rdf:nodeID="x">
      <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
   </rdf:Description>
</rdf:RDF>
```

2) … and actually a lot of RDF data residing in RDF stores, accessible via SPARQL endpoints already, rather than in RDF/XML

Our approach: XSPARQL (W3C submission)

- New query language… but don't reinvent!

XQuery + SPARQL = **XSPARQL**

Prolog:	**P**	`declare namespace` *prefix*`="`*namespace-URI*`"` or `prefix` *prefix*: <*namespace-URI*>	
Body:	**F** **L** **W** **O**	`for` *var* `in` *XPath-expression* `let` *var* := *XPath-expression* `where` *XPath-expression* `order by` *expression*	**or**
	F' **D** **W** **M**	`for` *varlist* `from` / `from named` <*dataset-URI*> `where` {*pattern* } `order by` *expression* `limit` *integer* > 0 `offset` *integer* > 0	
Head:	**C**	`construct` { *template (with nested XSPARQL)* }	**or**
	R	`return` *XML + nested XSPARQL*	

Example: Mapping from RDF to XML


```
<relations>
{ for $Person $Name
  from <relations.rdf>
  where { $Person foaf:name $Name }
  order by $Name
  return
    <person name="{$Name}">
    {for $FName
     from <relations.rdf>
     where {
        $Person foaf:knows $Friend .
        $Person foaf:name $Name .
        $Friend foaf:name $Fname }
     return <knows>{$FName}</knows>
      } </person>
}</relations>
```

```
<relations>
  <person name="Alice">
    <knows>Bob</knows>
    <knows>Charles</knows>
  </person>
  <person name="Bob">
    <knows>Charles</knows>
  </person>
  <person name="Charles"/>
</relations>
```

Example: Adding value generating functions to SPARQL


```
construct { :me foaf:knows _:b .
            _:b foaf:name {fn:concat("""",?N," ",?F,""""")} }
from <MyAddrBookVCard.rdf>
where {
   ?ADDR vc:Given ?N .
   ?ADDR vc:Family ?F .
}
```

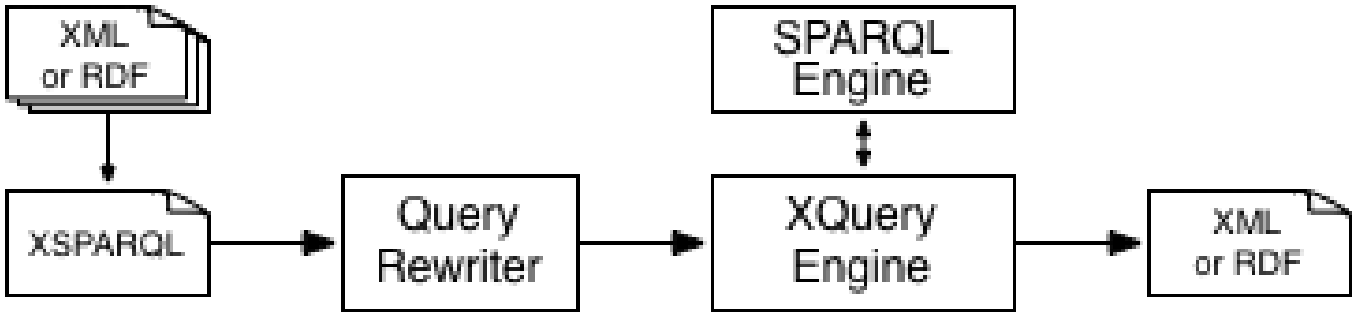
```
…
:me foaf:knows _:b1. _:b1 foaf:name "Peter Patel-Schneider" .
:me foaf:knows _:b2. _:b2 foaf:name "Stefan Decker" .
:me foaf:knows _:b3. _:b3 foaf:name "Thomas Eiter" .
…
```

Implementation and semantics:

- Formal Semantics (XSPARQL1.0):
  - Based on XQuery formal Semantics
  - Can be implemented based on rewriting to XQuery
- Challenges/Limitations:
  - Nesting, scope of RDF dataset…
  - different “type systems” of RDF/XML (sequences)
  - adding ontological inference (to resolve heterogeneities)
  - We are working on this in XSPARQL1.1!

Formal Semantics:

- Initial idea (and formalised in XSPARQL1.0):
  - extension of the XQuery semantics by plugging in SPARQL semantics in a modular way



  - Rewriting algorithm is defined for embedding XSPARQL into native XQuery plus interleaved calls to a SPARQL endpoint

Rewriting XSPARQL to XQuery…


```
construct { _:b foaf:name {fn:concat("""",$N," ",$F,"""")} }
from <vcard.rdf>
where { $P vc:Given $N . $P vc:Family $F . }
```

```
let $aux_query := fn:concat("http://localhost:2020/sparql?query=",
                  fn:encode-for-uri(
                      "select $P $N $F from <vcard.rdf>
                       where {$P vc:Given $N. $P vc:Family $F.}"))
for $aux_result at $aux_result_pos
                  in doc($aux_query)//sparql_result:result
let $P_Node :=
let $N_Node :=
let $F_Node :=
let $N := dat
let $N_RDFTerm := local:rdf_term($N_NodeType,$N)
...
return ( fn:c
         ( f                                          "." )
```

1. Encode SPARQL in HTTP call SELECT Query
2. Execute call, via fn:doc function
3. Collect results from SPARQL result format(XML)
4. `construct` becomes `return` that outputs triples.

Formalisation

- Current formalisation embeds rewriting in the functional semantics of XQuery:
  http://xsparql.deri.org/spec/xsparql-semantics.html#id:flwor-expressions

mapping rules $[\cdot]_{Expr'}$ inherit from the definitions of XQuery's $[\cdot]_{Expr}$

$$\left[\begin{array}{l}\texttt{for } \$VarName_1 \ldots \$VarName_n \ DatasetClause \texttt{ where} \\ GroupGraphPattern\ SolutionModifier\ ReturnClause\end{array}\right]_{Expr'}$$

==

$$\left\lceil \texttt{let } \$\_aux\_queryresult := \left[\begin{array}{l} VarName_1 \ldots VarName_n\ DatasetClause \\ \texttt{where } GroupGraphPattern\ SolutionModifier\end{array}\right]_{SparqlQuery} \right\rceil$$

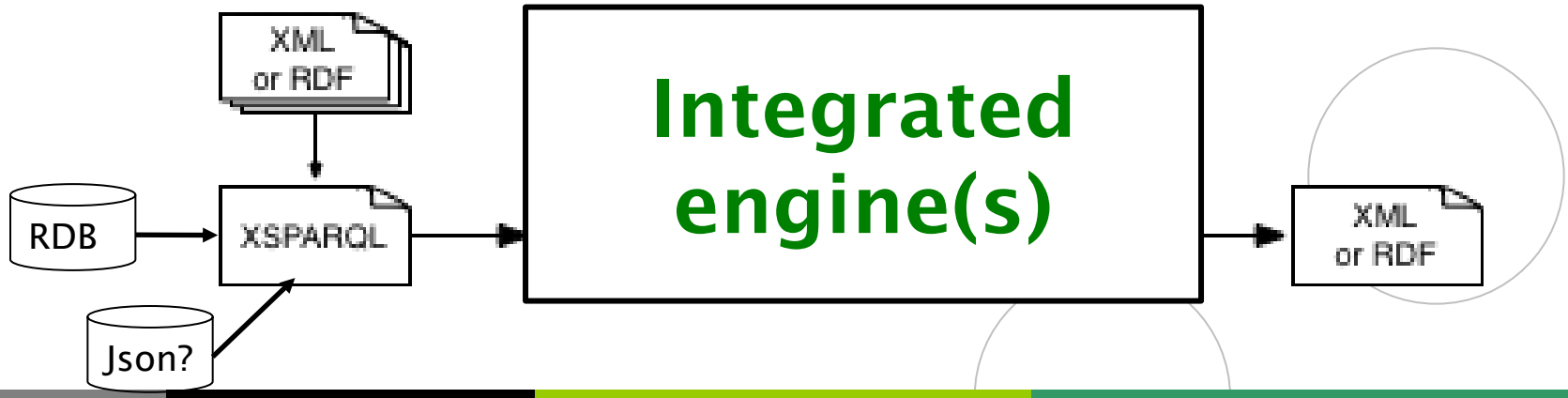
$$\left[\begin{array}{l}\$VarName_1 \ldots \$VarName_n\ DatasetClause \\ \texttt{where } GroupGraphPattern\ SolutionModifier\end{array}\right]_{SparqlQuery}$$

==

$$fs{:}sparql\left(\left[\begin{array}{l}\texttt{fs:serialize("SELECT } \$VarName_1 \ldots \$VarName_n\texttt{", } DatasetClause\texttt{, " where \{ ",} \\ \texttt{fs:serialize(} GroupGraphPattern\texttt{), " \} } SolutionModifier\texttt{")}\end{array}\right]_{Expr'}\right)$$

Ongoing work/Outlook:

- Simple rewriting semantics has some limitations, which we are currently working on:
  - **Adding RDFS/OWL entailment(and other SPARQL1.1 features)**
  - **Integrate RDB Querying (SQL), Json, etc.**
  - **Optimisations**…
  - Nesting,scope of RDF dataset…
  - Different "type systems" of RDF/XML (sequences)…



RDB Querying sketch

Extract foaf:knows relations from a RDB with two tables: containing persons and their knows relations

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

for p1.name as $x, p2.name as $y
from person as p1, person as p2, relation as r
where {  p1.name = r.person and
         p2.name = r.knows }

construct { $x foaf:knows $y }
```

RDBForClause to specify input from RDB tables

SQL Where pattern

Output RDF (or XML)

- Use as an RDB2RDF exporter…

Unified RDF/XML/RDB Query Layer = XSPARQL + RDB2RDF ?

Optimisations – Example:

- E.g. dependent Join… i.e.

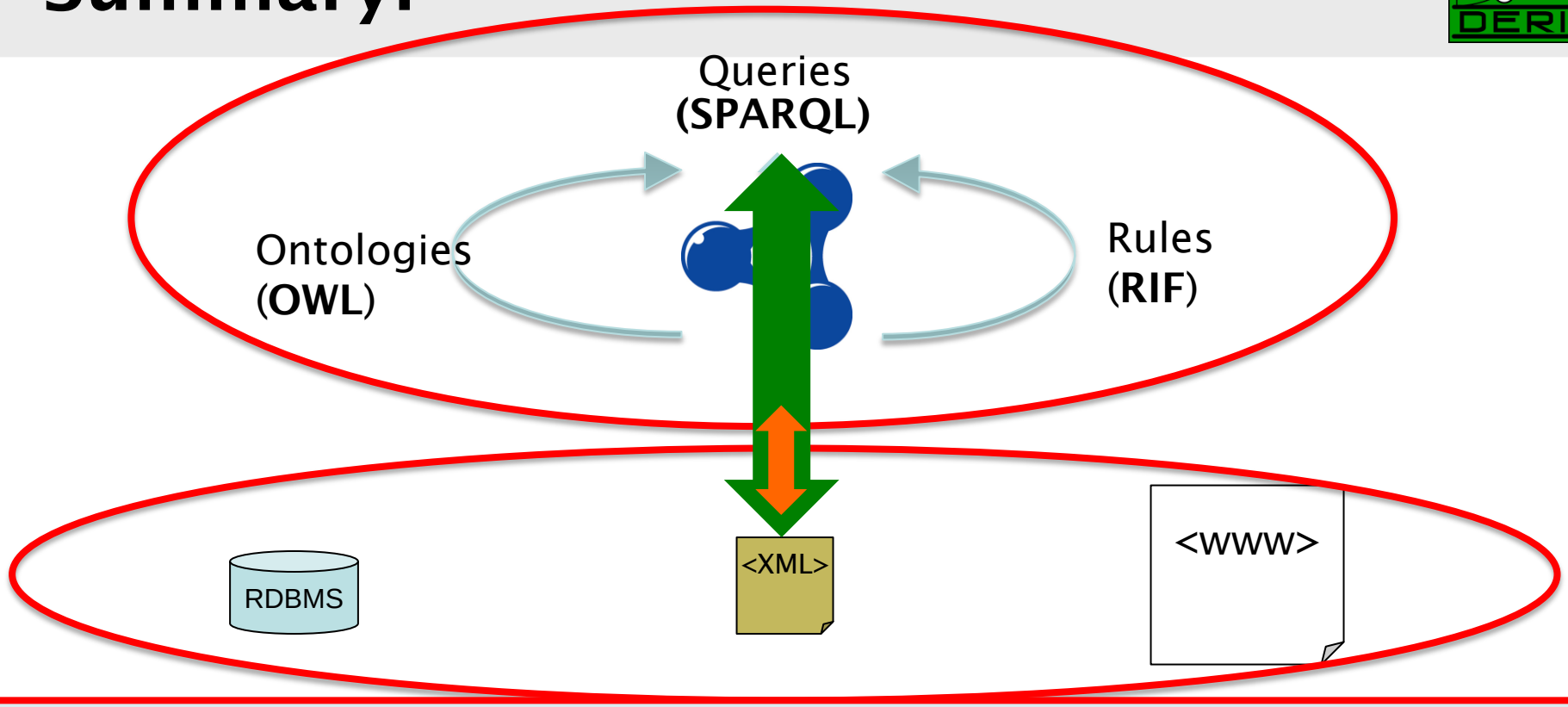
```
<relations>
{ for $Person $Name
  from <relations.rdf>
  where { $Person foaf:name $Name }
  order by $Name
  return
    <person name="{$Name}">
    {for $Fname
     where {
        $Person foaf:knows $Friend .
        $Friend foaf:name $Fname }
     return <knows>{$FName}</knows>
     } </person>
}</relations>
```

```
<relations>
{ let $aux := select $Person $Name $FName
  from <relations.rdf>
  where { $Person foaf:name $Name .
          $Person foaf:knows $Friend .
          $Friend foaf:name $Fname }

  for $Name in  $aux.Name
  return
    <person name="{$Name}">
    { for $FName in $aux.Fname
       where $aux.Name = $Name
      return <knows>{$FName}</knows>
     } </person>
}</relations>
```

Only one SPARQL query

Summary:



One Unified RDF/XML/RDB Query Layer to combine XQuery+SPARQL+GRDDL+SQL ?

- Enable optimisation across layers
- query each source format in native language instead of multistep transformation via "narrow" interfaces (e.g. SPARQL→ SPARQL-Resuly/XML→ Xquery/XSLT)
- Enable declarative view (a la Relational Algebra for core fragment of the language)
- Tighter integration of various source formats that populate the (Semantic) Web
- This needs a cross-activity effort in W3C? Semantic Web + XML + others?

Nesting,scope of RDF dataset…

Remember the query from before…. We were slightly cheating:

```
<relations>
{ for $Person $Name
  from <relations.rdf>
  where { $Person foaf:name $Name }
  order by $Name
  return
    <person name="{$Name}">
    {for $FName
     from <relations.rdf>
     where {
        $Person foaf:knows $Friend .
        $Person foaf:name $Name .
        $Friend foaf:name $Fname }
     return <knows>{$FName}</knows>
      } </person>
}</relations>
```

Nesting,scope of RDF dataset…

Remember the query from before…. This is what one would rather expect

```
<relations>
{ for $Person $Name
  from <relations.rdf>
  where { $Person foaf:name $Name }
  order by $Name
  return
   <person name="{$Name}">
   {for $FName

    where {
      $Person foaf:knows $Friend .

      $Friend foaf:name $Fname }
    return <knows>{$FName}</knows>
     } </person>
}</relations>
```

- The nested query should be over the same Dataset as the outer query, bindings to bnodes should be preserved
- Two separate, independent SPARQL calls don't work anymore

Solution:

- We need to add Dataset to the dynamic environment in the semantics.
- We need a special SPARQL implementation that allows several calls to the same active graph.

Different “type systems” of RDF/XML (e.g. sequences)…

- Social Graph queries a la [1]:

Give me all pairs of co-authors and their joint publications.

```
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix dc:      <http://purl.org/dc/elements/1.1/>

let $ds := for *
from <http://dblp.l3s.de/d2r/resource/authors/Axel_Polleres>
where { $pub dc:creator [] }
construct {
           { for * from $pub where { $p dc:creator $o . }
            construct {$p dc:creator <{$o}>} } }

let $allauthors :=
distinct-values(for $o from $ds where {$p dc:creator $o}
order by $o
return $o)

for $auth at $auth_pos in $allauthors
 for $coauth in $allauthors[position() > $auth_pos]
    let $commonPubs := count(
     { for $pub from $ds where { $pub dc:creator $auth, $coauth } return $pub }
    )
where ($commonPubs > 0)
construct { [ :author1 $auth; :author2 $coauth; :commonPubs $commonPubs ] }
```

Assignment of graphs to variables needs new datatype RDFGraph

Nested CONSTRUCTs queries

Lists of RDFTerms need new datatype RDFTerm

1. Mauro San Martín, Claudio Gutierrez: Representing, Querying and Transforming Social Networks with RDF/SPARQL. ESWC 2009: 293-307

Use: SIOC-2-RSS

- XSPARQL+SIOC enables **customised RSS export**:

```
<channel>
<title>
        {for $name
         from <http://www.johnbreslin.com/blog/index.php?sioc_type=site>
         where { [a sioc:Forum] sioc:name $name }
         return $name}
</title>
        {for $seeAlso
         from <http://www.johnbreslin.com/blog/index.php?sioc_type=site>
         where { [a sioc:Forum] sioc:container_of [rdfs:seeAlso $seeAlso] }
return <item>
                {for $title $descr $date
                 from $seeAlso
                 where { [a sioc:Post] dc:title $title ;
                                       sioc:content $descr;
                                       dcterms:created $date
                 return <title>$title</title>
                        <description>$descr</description>
                        <pubDate>$date</pubDate>}
```

RSS2.0

XML SPARQL RDF

"Great stuff,... I have not seen any SIOC to RSS xslt examples or vice versa" (John Breslin, creator of SIOC)

Use: Generate KML from FOAF+Geo:

- XSPARQL+FOAF and GEO data enables **KML map data**:


```
<kml xmlns="http://www.opengis.net/kml/2.2">
{
for $person $name $long $lat
from <http://www.deri.ie/Stefan_Decker>
where { $person; foaf:name $name;
        foaf:based_near [ geo:long $long; geo:lat $lat ] }
return <Placemark>
  <name>{fn:concat("Location of ", $name)}</name>
  <Point>
    <coordinates>{fn:concat($long, ",", $lat,
",0")}</coordinates>
  </Point>
</Placemark>
}
</kml>
```

XSPARQL