XSPARQL

Traveling between the XML and RDF Worlds – and Avoiding the XSLT Pilgrimage

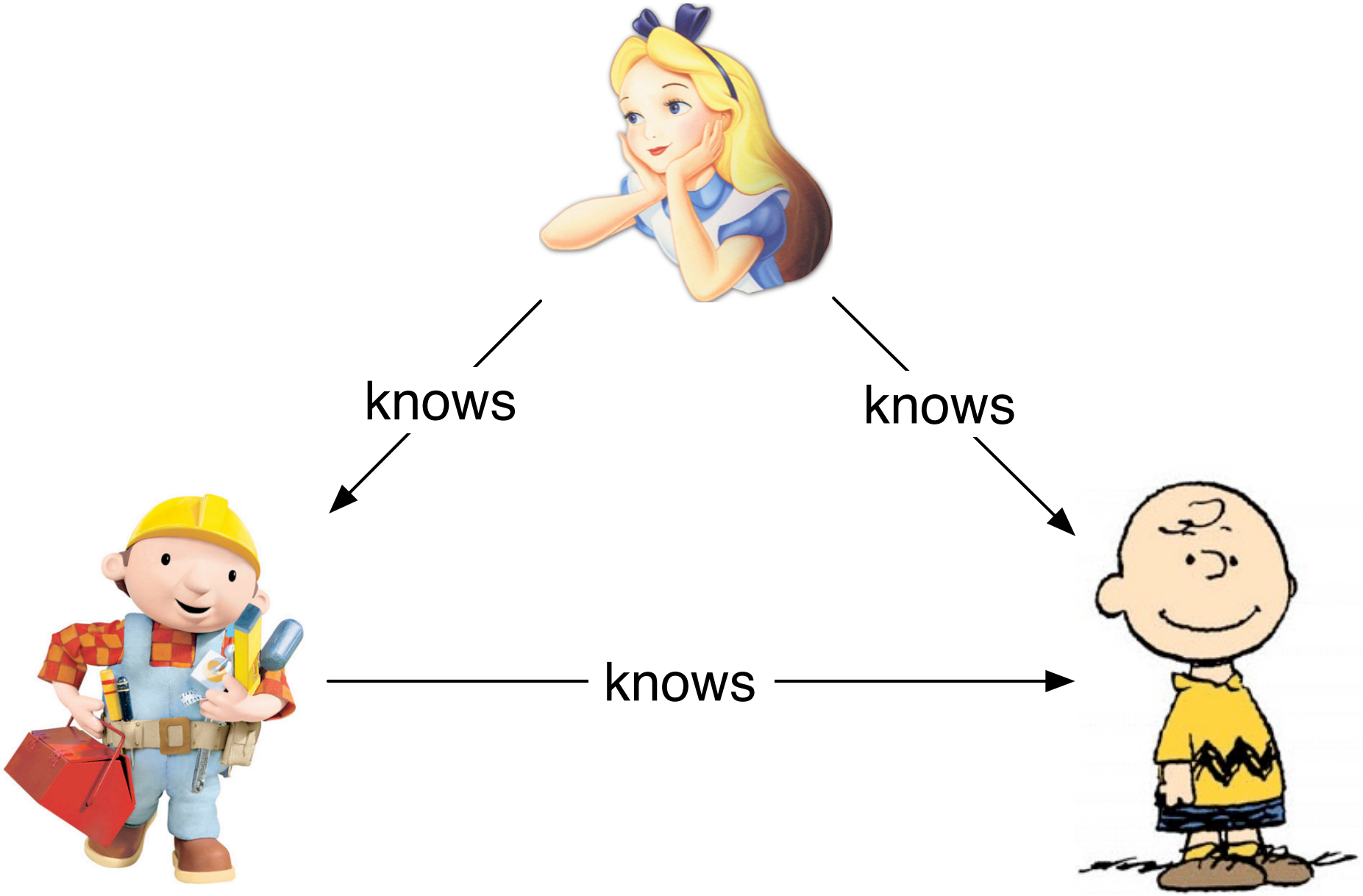
Waseem Akhtar[1] Jacek Kopecký[3]
Thomas Krennwallner[1,2] Axel Polleres[1]

[1]Digital Enterprise Research Institute, National University of Ireland, Galway

[2]Knowledge-Based Systems Group, Institute for Information Systems, TU Wien

[3]STI Innsbruck, University of Innsbruck, Austria

Alice, Bob, and Charles



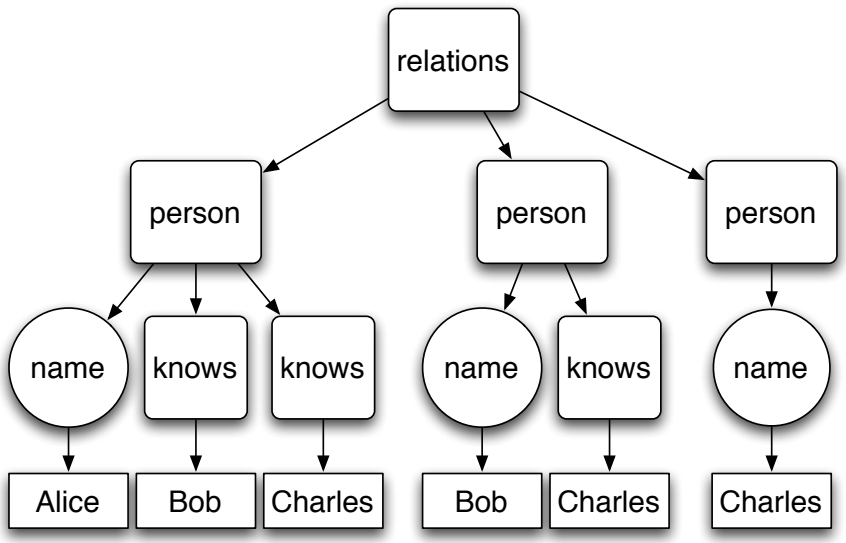
Motivation

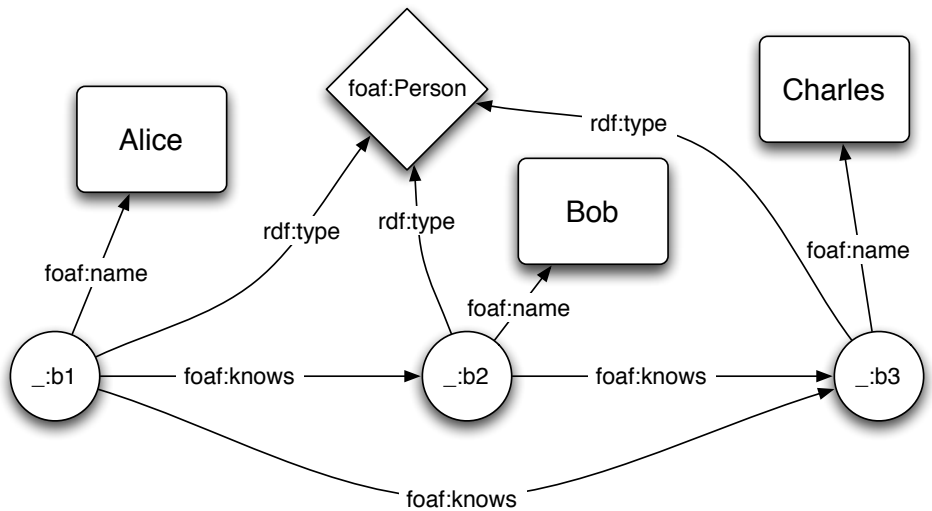
relations.xml

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

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".
```





Motivation

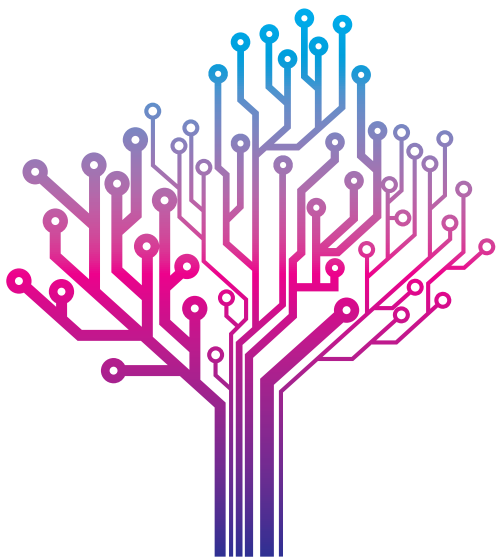
relations.xml

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

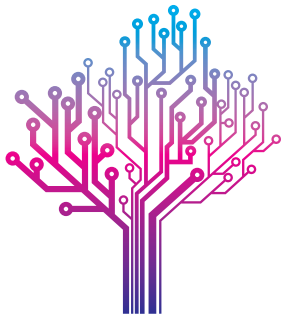
?

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".
```

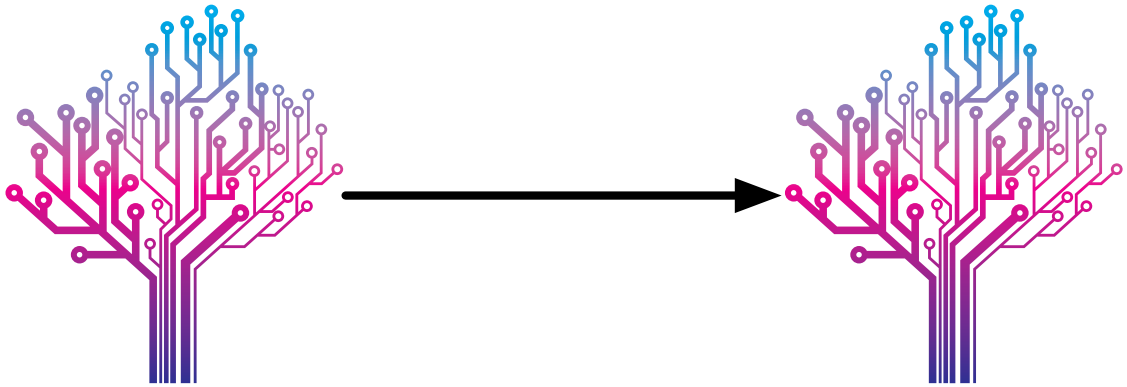


Motivation



- Two standards for different types of data: XML and RDF
- XML data model differs from RDF data model
- XML and RDF query languages have different objectives
  ... but not so different at all.

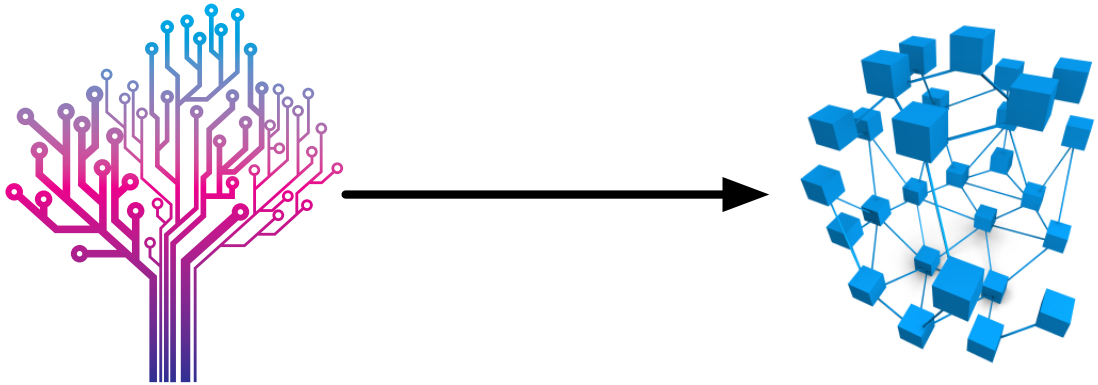
Mapping between XML and RDF



from \ to	XML	RDF
XML	XSLT, XQuery	
RDF		
RDF + XML		

Solved

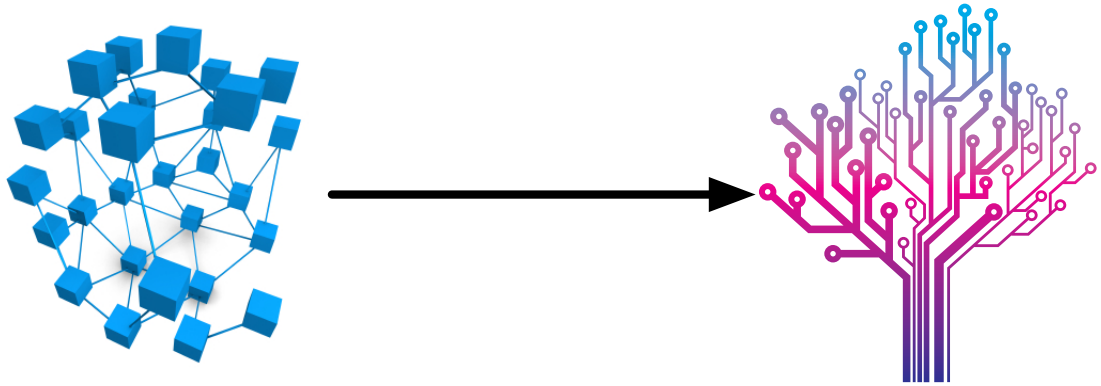
Mapping between XML and RDF



from \ to	XML	RDF
XML	XSLT, XQuery	XSLT, XQuery ?
RDF		
RDF + XML		

Tedious

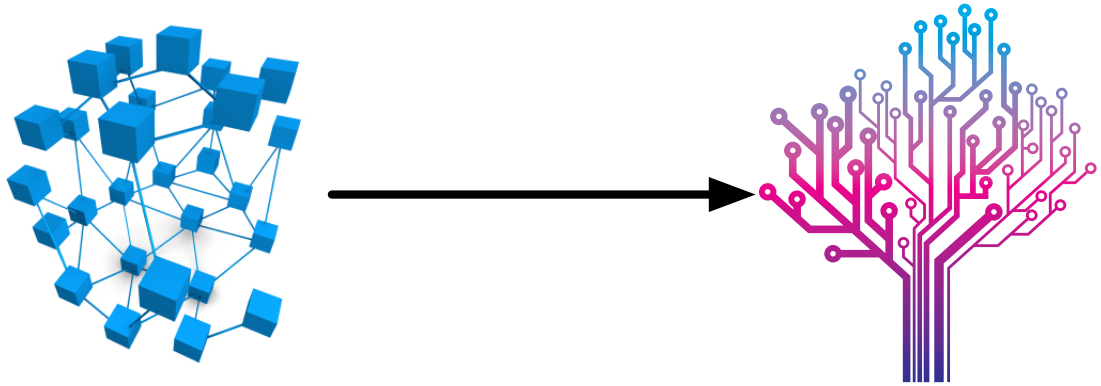
Mapping between XML and RDF



from \ to	XML	RDF
XML	XSLT, XQuery	XSLT, XQuery ?
RDF	XSLT, XQuery ?	
RDF + XML		

Issue: ambiguous XML representation of RDF

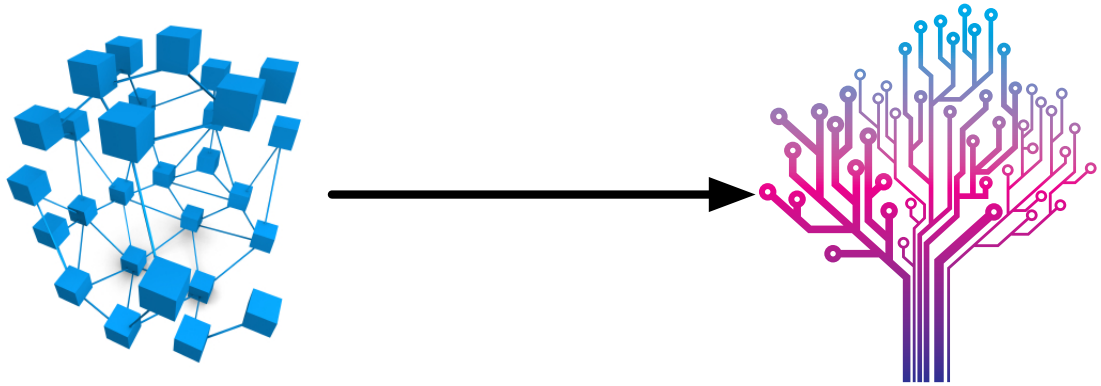
Mapping between XML and RDF



from \ to	XML	RDF
XML	XSLT, XQuery	XSLT, XQuery ?
RDF	XSLT, XQuery ?	
RDF + XML		

Issue: extract the whole RDF store as XML

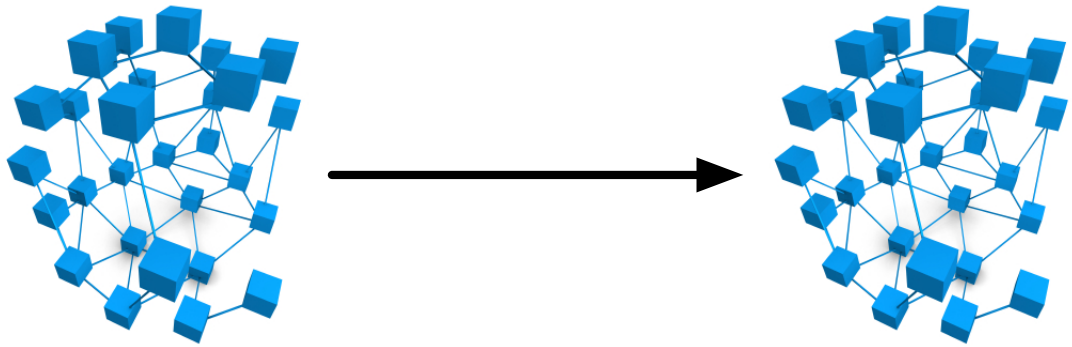
Mapping between XML and RDF



from \ to	XML	RDF
XML	XSLT, XQuery	XSLT, XQuery ?
RDF	XSLT, XQuery ?	
RDF + XML		

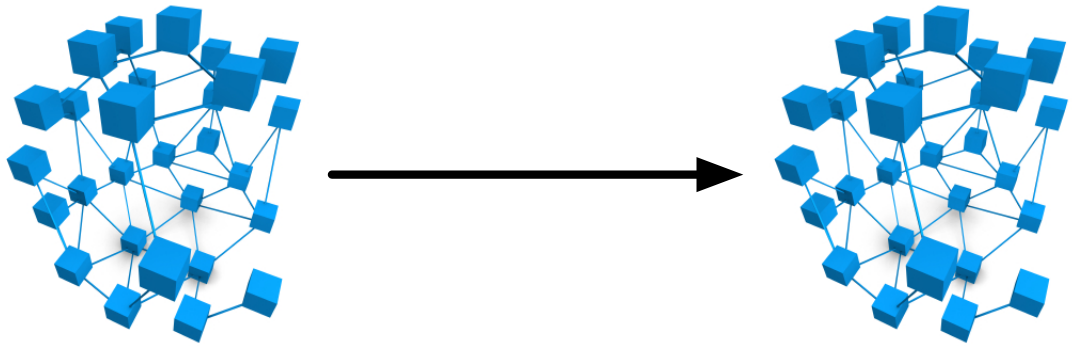
Issue: higher entailment regime

Mapping between XML and RDF



from \ to	XML	RDF
XML	XSLT, XQuery	XSLT, XQuery ?
RDF	XSLT, XQuery ?	SPARQL
RDF + XML		

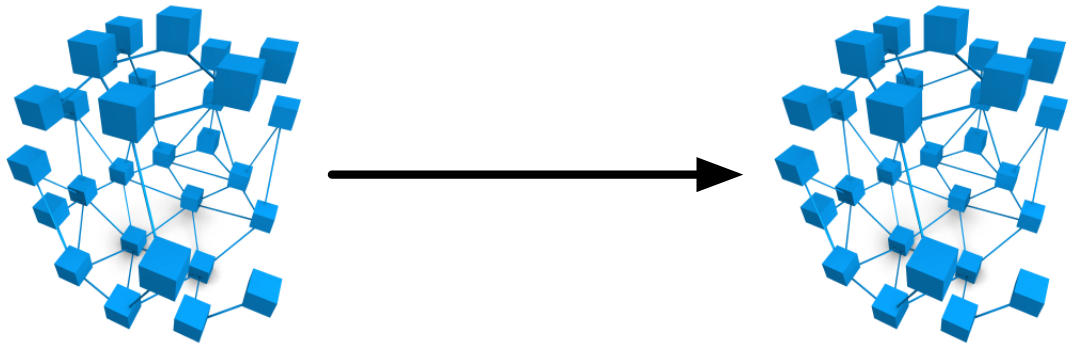
Mapping between XML and RDF



from \ to	XML	RDF
XML	XSLT, XQuery	XSLT, XQuery ?
RDF	XSLT, XQuery ?	SPARQL ?
RDF + XML		

```
construct { _:b foaf:name ?FN }
from <vcard.rdf>
where { ?P vc:FN ?FN }
```

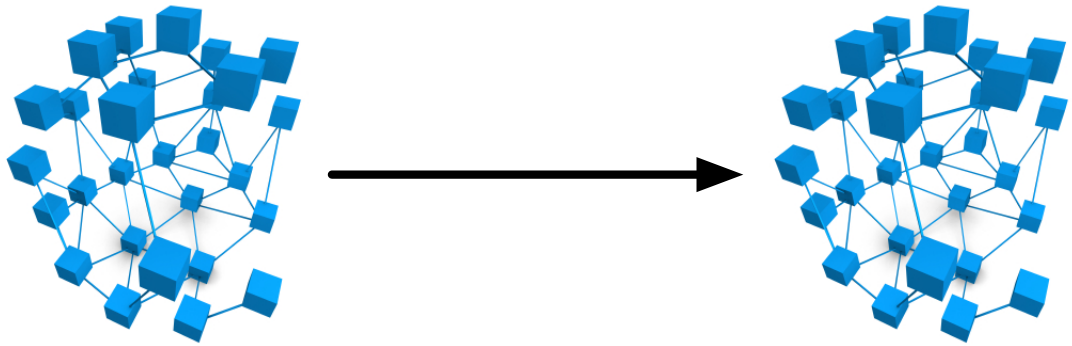
Mapping between XML and RDF



from \ to	XML	RDF
XML	XSLT, XQuery	XSLT, XQuery ?
RDF	XSLT, XQuery ?	SPARQL ?
RDF + XML		

```
construct { _:b foaf:name ??  }
from <vcard.rdf>
where { ?P vc:Given ?N . ?P vc:Family ?F }
```

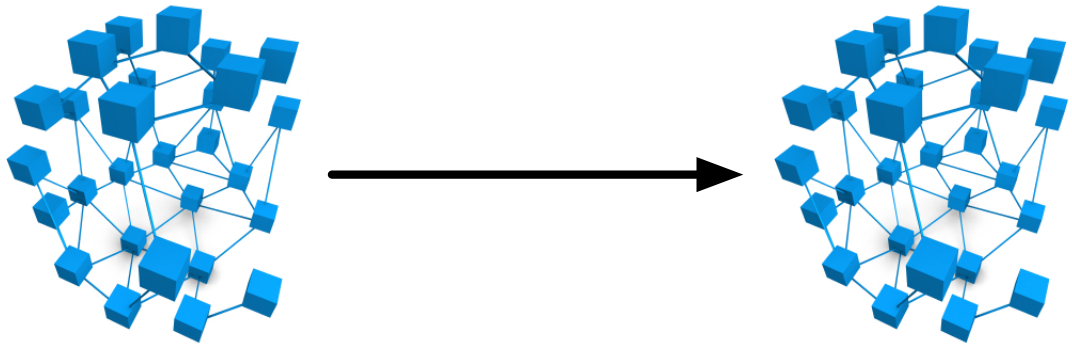
Mapping between XML and RDF



from \ to	XML	RDF
XML	XSLT, XQuery	XSLT, XQuery ?
RDF	XSLT, XQuery ?	SPARQL ?
RDF + XML		

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

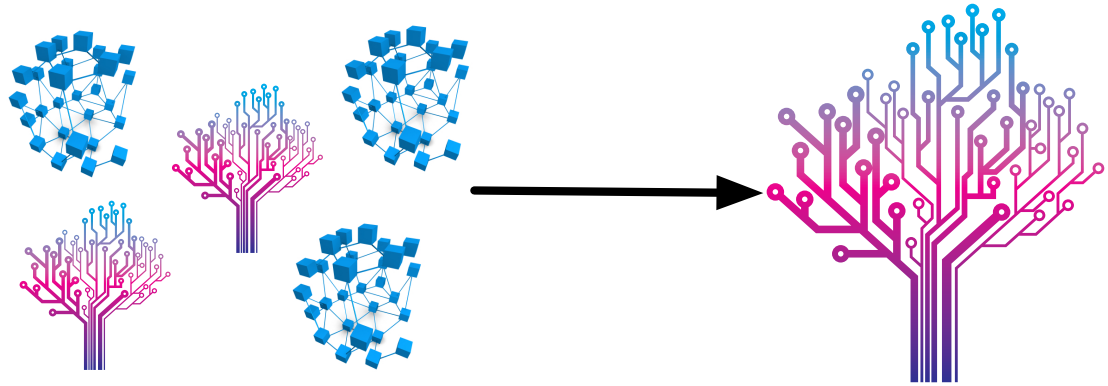
Mapping between XML and RDF



from \ to	XML	RDF
XML	XSLT, XQuery	XSLT, XQuery ?
RDF	XSLT, XQuery ?	SPARQL ?
RDF + XML		

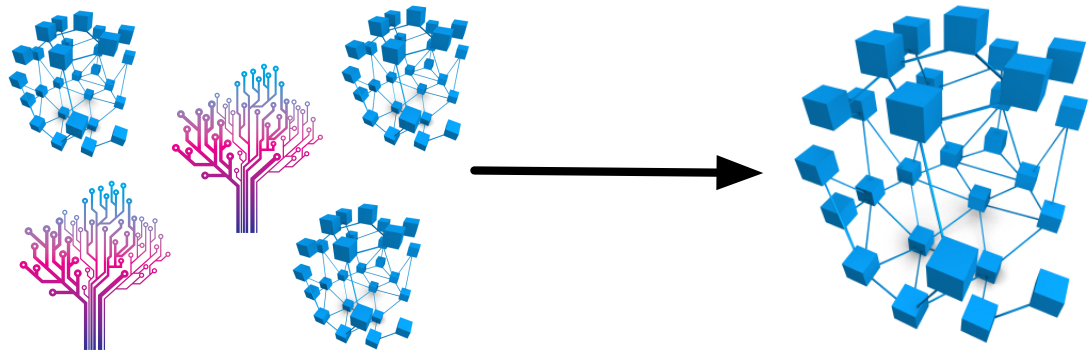
```
construct { _:b foaf:name fn:concat("""",?N," ",?F,"""") }
from <vcard.rdf>
where { ?P vc:Given ?N . ?P vc:Family ?F }
```

Mapping between XML and RDF



from \ to	XML	RDF
XML	XSLT, XQuery	XSLT, XQuery ?
RDF	XSLT, XQuery ?	SPARQL ?
RDF + XML	?	

Mapping between XML and RDF



from \ to	XML	RDF
XML	XSLT, XQuery	XSLT, XQuery ?
RDF	XSLT, XQuery ?	SPARQL ?
RDF + XML	?	?

XQuery and SPARQL – A comparison

Schematic view on XQuery

Prolog:	**P**	`declare namespace` *prefix*`="`*namespace-URI*`"`
Body:	**F**	`for` *var* `in` *XPath-expression*
	L	`let` *var* `:=` *XPath-expression*
	W	`where` *XPath-expression*
	O	`order by` *XPath-expression*
Head:	**R**	`return` *XML + nested XQuery*

Schematic view on SPARQL

Prolog:	**P**	`prefix` *prefix*: <*namespace-URI*>
Head:	**C**	`construct` { *template* }
Body:	**D**	`from` / `from named` <*dataset-URI*>
	W	`where` { *pattern* }
	M	`order by` *expression*
		`limit` *integer* > 0
		`offset` *integer* > 0

XSPARQL: Combining XQuery with SPARQL

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

Mapping RDF to RDF

Generate fullname from first and last name:

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

Mapping RDF to RDF

Generate fullname from first and last name:

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

```
_:b1 foaf:name "Waseem Akhtar"
_:b2 foaf:name "Jacek Kopecky"
_:b3 foaf:name "Axel Polleres"
.
.
.
```

Mapping 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>
```

Mapping 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>
```

Mapping 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>
```

Mapping 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>
```

Mapping 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>
```

Mapping 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>
```

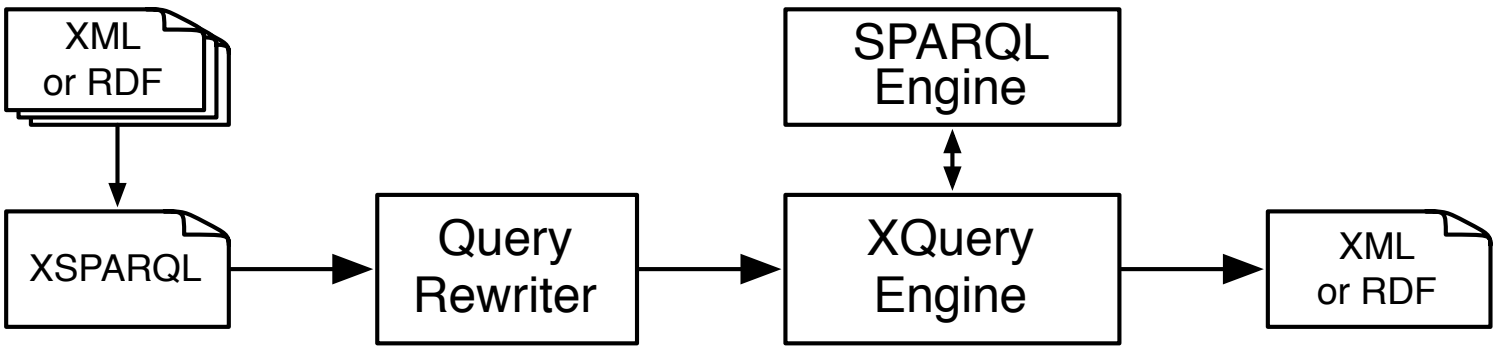
```
<relations>
 <person name="Alice">
  <knows>Bob</knows>
  <knows>Charles</knows>
 </person>
 <person name="Bob">
  <knows>Charles</knows>
 </person>
 <person name="Charles"/>
</relations>
```

XSPARQL Semantics + Implementation

- Formal semantics of XSPARQL: extension of the XQuery semantics by plugging in SPARQL semantics in a modular way

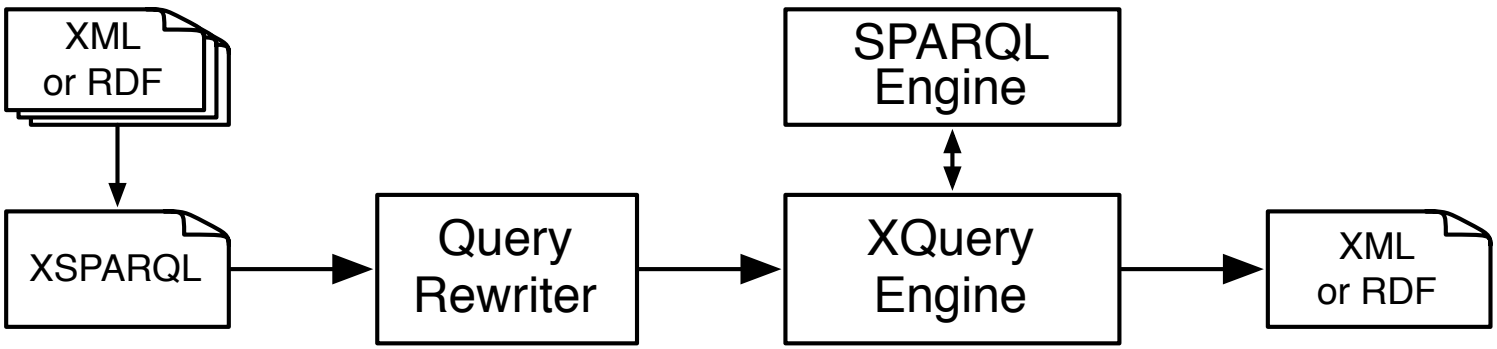
XSPARQL Semantics + Implementation

- Formal semantics of XSPARQL: extension of the XQuery semantics by plugging in SPARQL semantics in a modular way



XSPARQL Semantics + Implementation

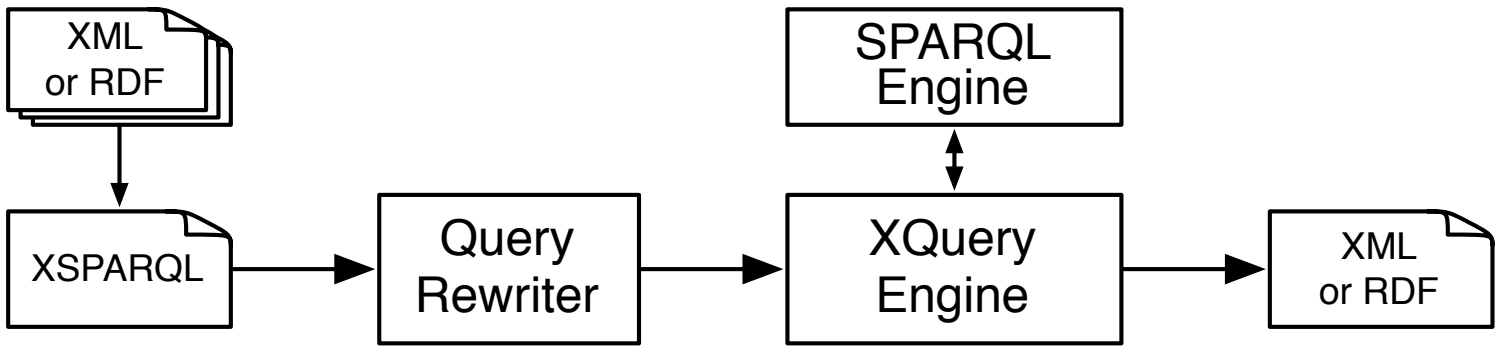
- Formal semantics of XSPARQL: extension of the XQuery semantics by plugging in SPARQL semantics in a modular way



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

XSPARQL Semantics + Implementation

- Formal semantics of XSPARQL: extension of the XQuery semantics by plugging in SPARQL semantics in a modular way

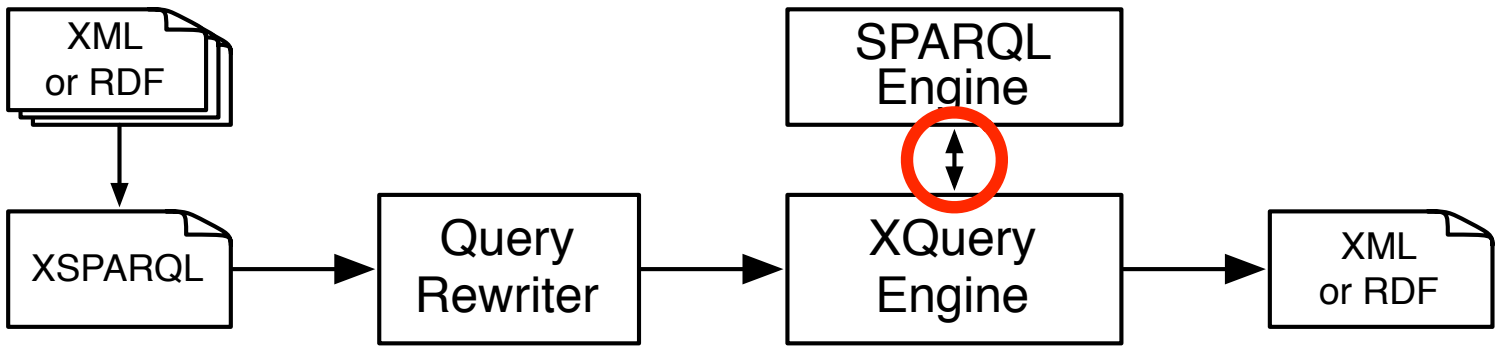


- Rewriting algorithm is defined for embedding XSPARQL into native XQuery plus interleaved calls to a SPARQL endpoint

- Benefits: rely on off-the-shelf components

XSPARQL Semantics + Implementation

- Formal semantics of XSPARQL: 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

- Benefits: rely on off-the-shelf components

- Ongoing work: Optimization

Rewriting XSPARQL to XQuery

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

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.}"))
```

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
```

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 := $aux_result/sparql_result:binding[@name="P"]
let $N_Node := $aux_result/sparql_result:binding[@name="N"]
let $F_Node := $aux_result/sparql_result:binding[@name="F"]
let $N := data($N_Node/*)
let $N_NodeType := name($N_Node/*)
let $N_RDFTerm := local:rdf_term($N_NodeType,$N)
. . .
```

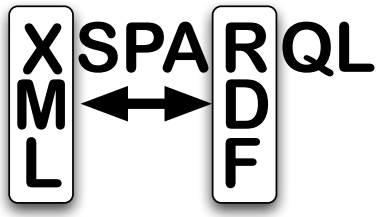
Rewriting XSPARQL to XQuery

```
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 := $aux_result/sparql_result:binding[@name="P"]
let $N_Node := $aux_result/sparql_result:binding[@name="N"]
let $F_Node := $aux_result/sparql_result:binding[@name="F"]
let $N := data($N_Node/*)
let $N_NodeType := name($N_Node/*)
let $N_RDFTerm := local:rdf_term($N_NodeType,$N)
. . .
return ( fn:concat("_:b",$aux_result_pos," foaf:name "),
         ( fn:concat("""",$N_RDFTerm," ",$F_RDFTerm,"""") ), "." )
```

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 := $aux_result/sparql_result:binding[@name="P"]
let $N_Node := $aux_result/sparql_result:binding[@name="N"]
let $F_Node := $aux_result/sparql_result:binding[@name="F"]
let $N := data($N_Node/*)
let $N_NodeType := name($N_Node/*)
let $N_RDFTerm := local:rdf_term($N_NodeType,$N)
. . .
return ( fn:concat("_:b",$aux_result_pos," foaf:name "),
         ( fn:concat("""",$N_RDFTerm," ",$F_RDFTerm,""") ), "." )
```

Conclusion and Outlook



- Querying XML and RDF at one shot
- Producing XML or RDF from XML and/or RDF
- Extending SPARQL by expressive XQuery constructs
- Extending XQuery by SPARQL graph patterns

Ongoing work:

- Optimization
- extend XSPARQL with nested FLWOR in `where` part
- XSPARUL

`http://xsparql.deri.org/`