Centre for
Data Analytics

Insight

SYRql: A Dataflow Language for Large Scale Processing of RDF

Fadi Maali, Padmashree Ravindra, Kemafor Anyanwu, and Stefan Decker

October 22nd 2014

ISWC 2014

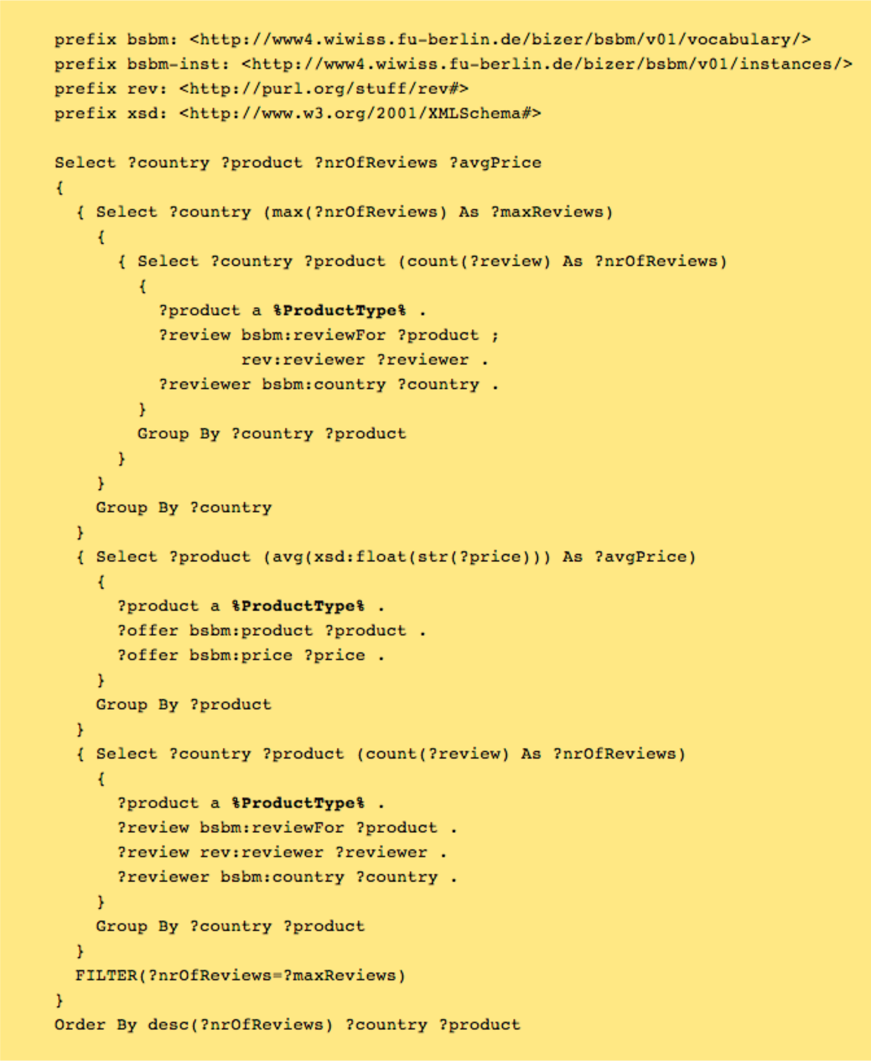
IRISH RESEARCH COUNCIL
An Chomhairle um Thaighde in Éirinn
DCU
NUI Galway
OÉ Gaillimh
UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh
UCD DUBLIN
sfi

Outline

- Motivating a dataflow language
- RDF Algebra: the underlying data model
- SYRql, the language
- Evaluation

IRISH RESEARCH COUNCIL
An Chomhairle um Thaighde in Éirinn

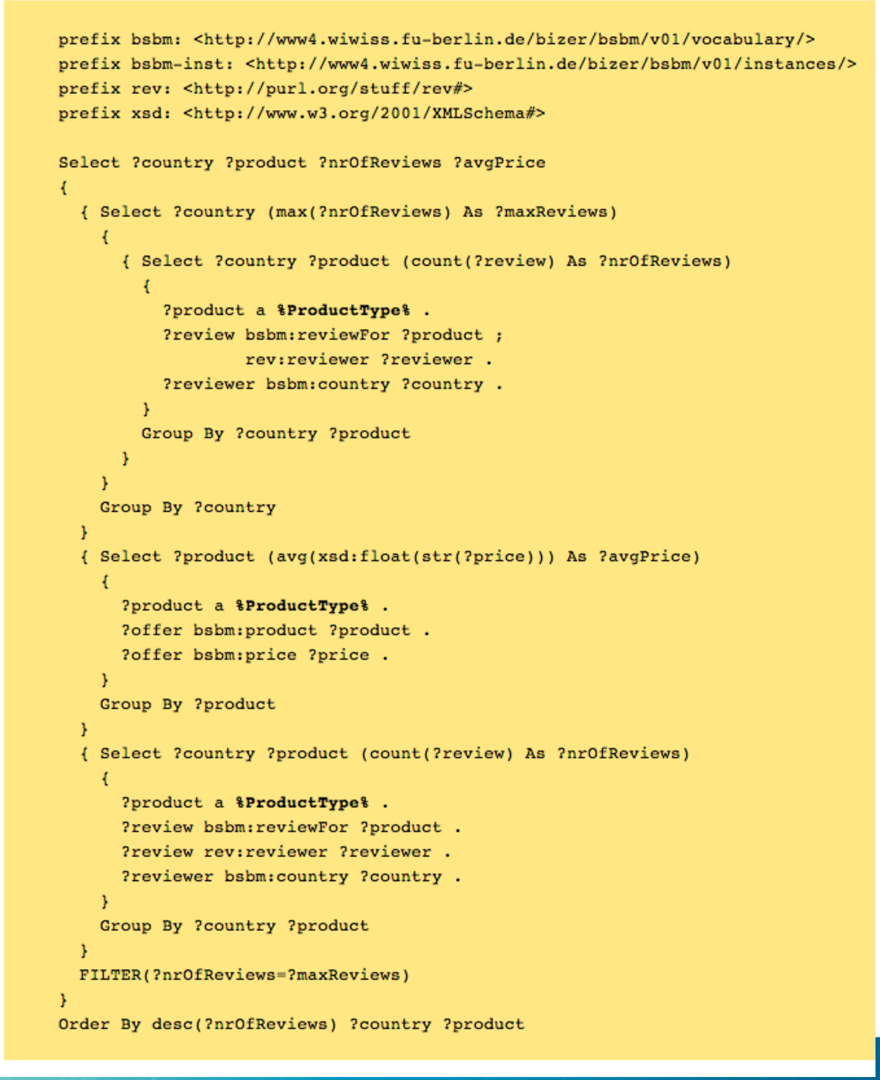
I (We) ❤ SPARQL, but…



```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix rev: <http://purl.org/stuff/rev#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

Select ?country ?product ?nrOfReviews ?avgPrice
{
  { Select ?country (max(?nrOfReviews) As ?maxReviews)
    {
      { Select ?country ?product (count(?review) As ?nrOfReviews)
        {
          ?product a %ProductType% .
          ?review bsbm:reviewFor ?product ;
                  rev:reviewer ?reviewer .
          ?reviewer bsbm:country ?country .
        }
        Group By ?country ?product
      }
    }
    Group By ?country
  }
  { Select ?product (avg(xsd:float(str(?price))) As ?avgPrice)
    {
      ?product a %ProductType% .
      ?offer bsbm:product ?product .
      ?offer bsbm:price ?price .
    }
    Group By ?product
  }
  { Select ?country ?product (count(?review) As ?nrOfReviews)
    {
      ?product a %ProductType% .
      ?review bsbm:reviewFor ?product .
      ?review rev:reviewer ?reviewer .
      ?reviewer bsbm:country ?country .
    }
    Group By ?country ?product
  }
  FILTER(?nrOfReviews=?maxReviews)
}
Order By desc(?nrOfReviews) ?country ?product
```

I (We) ❤ SPARQL, but…



```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix rev: <http://purl.org/stuff/rev#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

Select ?country ?product ?nrOfReviews ?avgPrice
{
  { Select ?country (max(?nrOfReviews) As ?maxReviews)
    {
      { Select ?country ?product (count(?review) As ?nrOfReviews)
        {
          ?product a %ProductType% .
          ?review bsbm:reviewFor ?product ;
                  rev:reviewer ?reviewer .
          ?reviewer bsbm:country ?country .
        }
        Group By ?country ?product
      }
    }
    Group By ?country
  }
  { Select ?product (avg(xsd:float(str(?price))) As ?avgPrice)
    {
      ?product a %ProductType% .
      ?offer bsbm:product ?product .
      ?offer bsbm:price ?price .
    }
    Group By ?product
  }
  { Select ?country ?product (count(?review) As ?nrOfReviews)
    {
      ?product a %ProductType% .
      ?review bsbm:reviewFor ?product .
      ?review rev:reviewer ?reviewer .
      ?reviewer bsbm:country ?country .
    }
    Group By ?country ?product
  }
  FILTER(?nrOfReviews=?maxReviews)
}
Order By desc(?nrOfReviews) ?country ?product
```

SPARQL AND + Filters

Linear Time

+ Union

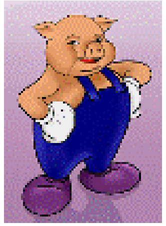
NP-Complete

+ *

PSPACE-Complete

What about other Big Data languages?

```
?prod a :PoductType .
?r :reviewFor ?prod .
?r :reviewer ?rev
```



```
rdf = LOAD 'data' USING PigStorage(' ')
   AS (S,P,O);

SPLIT rdf INTO
       reviewers IF P = ':reviewer',
       reviews IF P = ':reviewFor',
       prods IF P = 'a' and
           O = 'ProductType';

tmp1 = JOIN prods BY S, reviews BY O;

tmp2 = JOIN tmp BY reviews::S,
   reviewers BY S;.
```

SYRql... it looks like...

```
$rdf = load('/bsbm20k');

$janReviews = $rdf -> pattern(
                '?review rev:reviewFor ?product .
                 ?review dc:date ?date .')
       -> filter (?date >= "2008-01-01")
      -> group by ?product into janCnt:count(?review);
```

SYRql… it looks like Pig Latin

```
$rdf = load('/bsbm20k');

$janReviews = $rdf -> pattern(
               '?review rev:reviewFor ?product .
                ?review dc:date ?date .')
       -> filter (?date >= "2008-01-01")
      -> group by ?product into janCnt:count(?review);
```

SYRql... it looks like SPARQL

```
$rdf = load('/bsbm20k');

$janReviews = $rdf -> pattern(
               '?review rev:reviewFor ?product .
                ?review dc:date ?date .')
       -> filter (?date >= "2008-01-01")
      -> group by ?product into janCnt:count(?review);
```

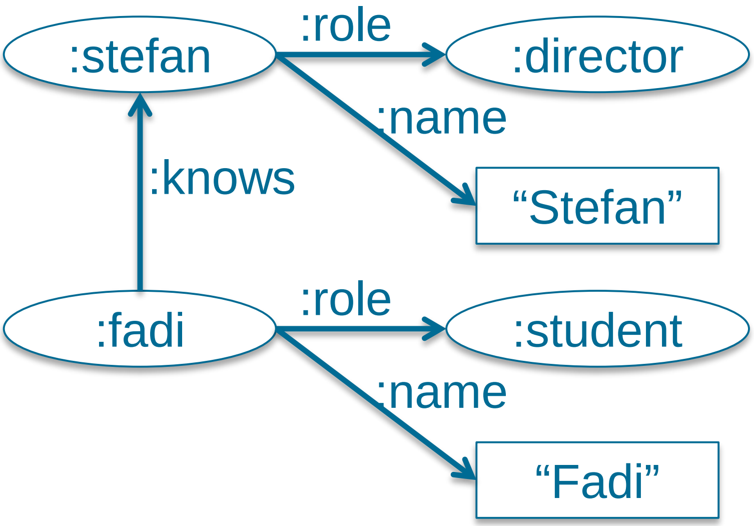
Two interesting questions follow:

1. Data model and set of operators

2. Evaluation and Optimisation

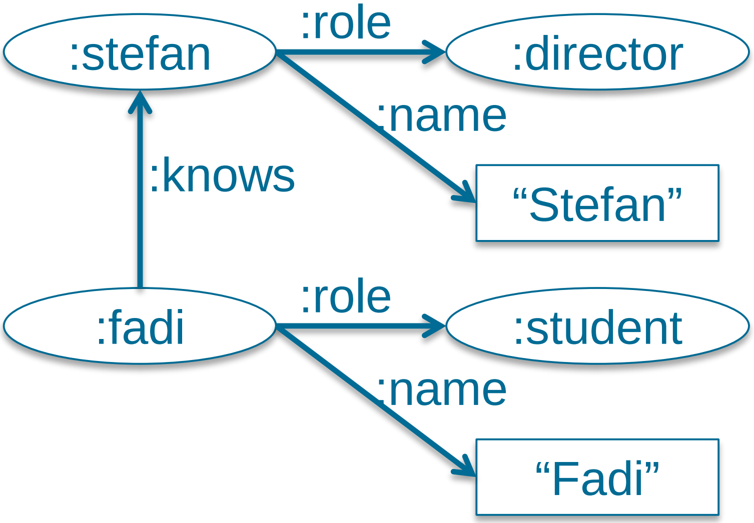
?

The Problem with SPARQL Algebra



```
?person :role ?role .
?person :name ?name
```

The Problem with SPARQL Algebra



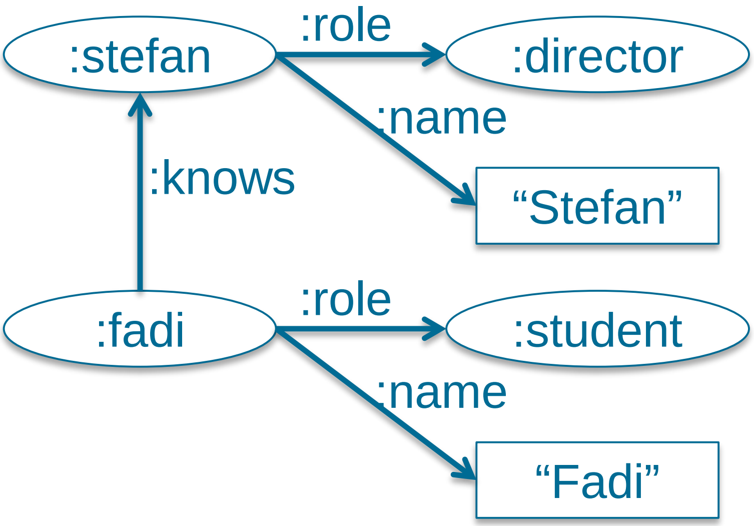
`?person :role ?role`

?person	?role
:stefan	:directo r
:fadi	:student

`?person :name ?name`

?perso n	?name
:stefan	"Stefan"
:fadi	"Fadi"

The Problem with SPARQL Algebra



```
?person :role ?role
```

?person	?role
:stefan	:director
:fadi	:student

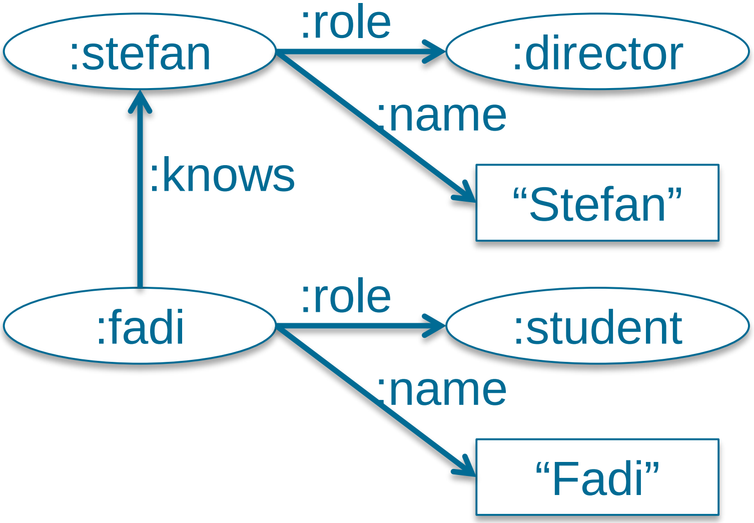
```
?person :name ?name
```

?person	?name
:stefan	"Stefan"
:fadi	"Fadi"

⋈

?person	?role	?name
:stefan	:director	"Stefan"
:fadi	:student	"Fadi"

The Problem with SPARQL Algebra



`?person :role ?role`

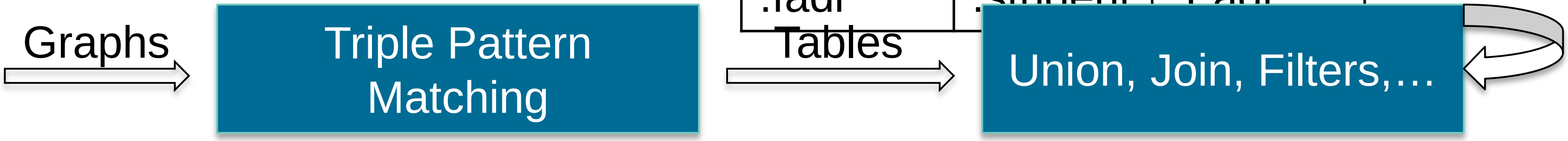
?person	?role
:stefan	:director
:fadi	:student

`?person :name ?name`

?person	?name
:stefan	"Stefan"
:fadi	"Fadi"

⋈

?person	?role	?name
:stefan	:director	"Stefan"
:fadi	:student	"Fadi"

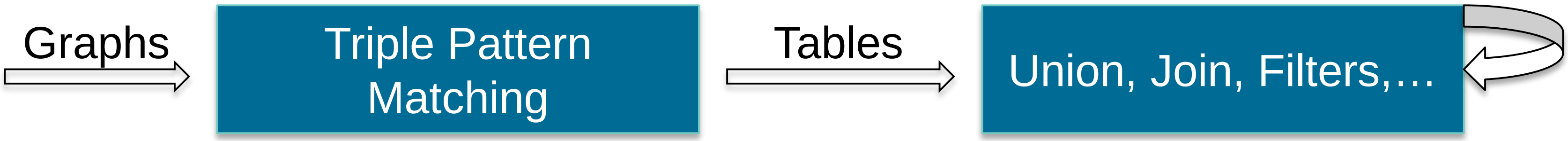

Graphs

Triple Pattern Matching

Tables

Union, Join, Filters,…

The Problem with SPARQL Algebra



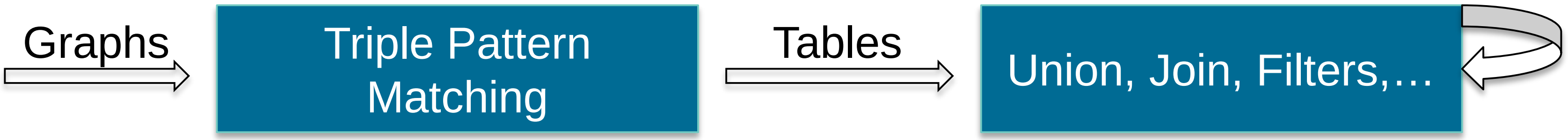
You cannot join or union two graphs

```
$g1 -> union $g2
```

You cannot apply a triple pattern to the results of another triple pattern matching

```
$v1 = $g1 -> pattern(‘?s :country ?o’)
$v2 = $v1 -> pattern(‘?s :country :Ireland’)
```

The Problem with SPARQL Algebra



Graphs

Triple Pattern Matching

Tables

Union, Join, Filters,…

You cannot join or union two graphs

```
      $g1 -> union $g2
```

You cannot apply a triple pattern to the results of another triple pattern matching

```
      $v1 = $g1 -> pattern(‘?s :country ?o’)
      $v2 = $v1 -> pattern(‘?s :country :Ireland’)
```

SPARQL Algebra is not fully compositional

RDF Algebra

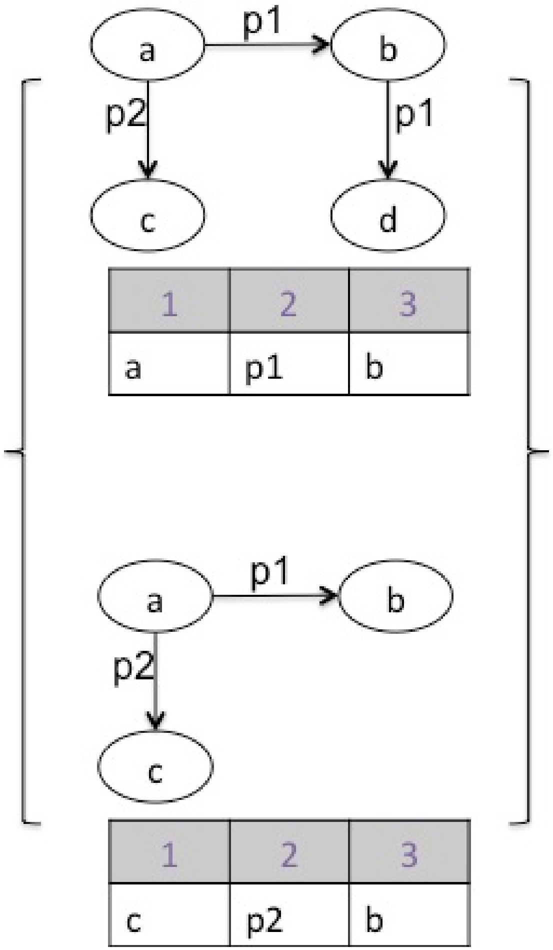
Pair graphs and bindings together

The input and output of all operators are sets of such pairs

Similar to Relational Algebra

Syntax and semantics are formally defined

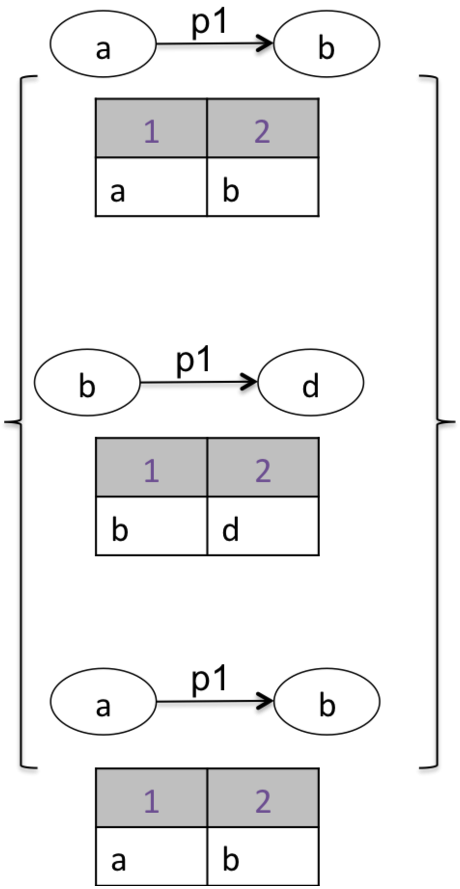
RDF Algebra Example (1/2)



1	2	3
a	p1	b

1	2	3
c	p2	b

[? p1 ?] =

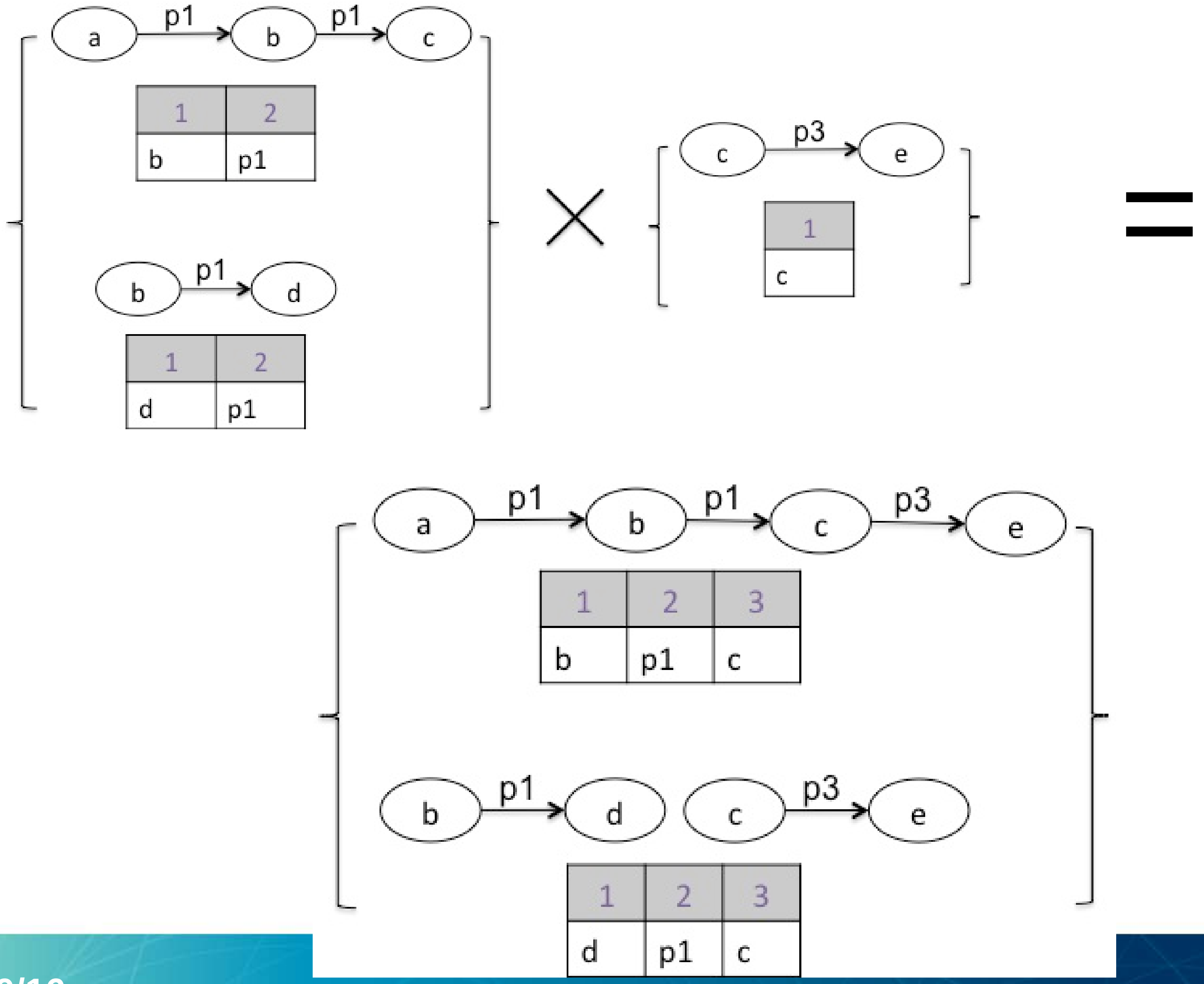


1	2
a	b

1	2
b	d

1	2
a	b

RDF Algebra Example (2/2)



1	2
b	p1

1	2
d	p1

1
c

1	2	3
b	p1	c

1	2	3
d	p1	c

In Relation to SPARQL

RDF Algebra expressions can express SPARQL 1.1 basic graph patterns with filters, aggregations and assignments

Furthermore, extending graphs with new triples are defined as a core operator

Algebraic Properties

e[? p o][? ? o] = e[? p o]

Applying a ‘less selectvie’ triple pattern does not change the results

(e1 X e2)[? p o] = (e1[? ? o] X e2[? ? o])[? p o]

Substitute ‘less selective’ expressions to speed up cross product evaluation

We captures the notion of selectivity via a formally-defined partial oreder relationship

We believe that this is applicable in more scenarios such as RDF results caching and view management.

Algebraic Properties

e[? p o][? ? o] = e[? p o]

Applying a ‘less selectvie’ triple pattern does not change the results

We captures the notion of selectivity via a formally-defined partial oreder relationship

We believe that this is applicable in more scenarios such as RDF results caching and view management.

Algebraic Properties

(e1 X e2)[? p o] = (e1[? ? o] X e2[? ? o])[? p o]

Substitute ‘less selective’ expressions to speed up cross product evaluation

We captures the notion of selectivity via a formally-defined partial oreder relationship

We believe that this is applicable in more scenarios such as RDF results caching and view management.

Algebraic Properties

e[? p o][? ? o] = e[? p o]

Applying a ‘less selectvie’ triple pattern does not change the results

(e1 X e2)[? p o] = (e1[? ? o] X e2[? ? o])[? p o]

Substitute ‘less selective’ expressions to speed up cross product evaluation

We believe that this can be applicable in wider set of scenarios such as RDF results caching and view management

SYRql

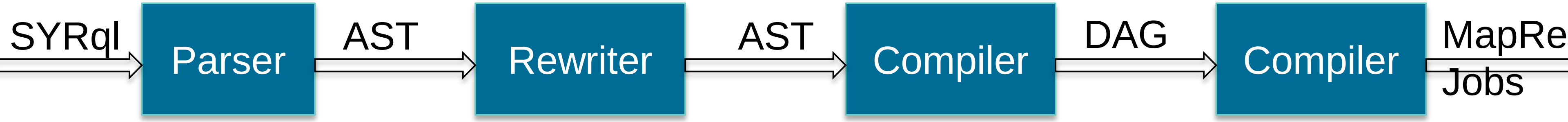
```
$rdf = load('/bsbm20k');

$janReviews = $rdf -> pattern(
               '?review rev:reviewFor ?product .
                ?review dc:date ?date .')
       -> filter (?date >= "2008-01-01")
      -> group by ?product into janCnt:count(?review);
```

SYRql Implementation

Use JSON-LD for data representation

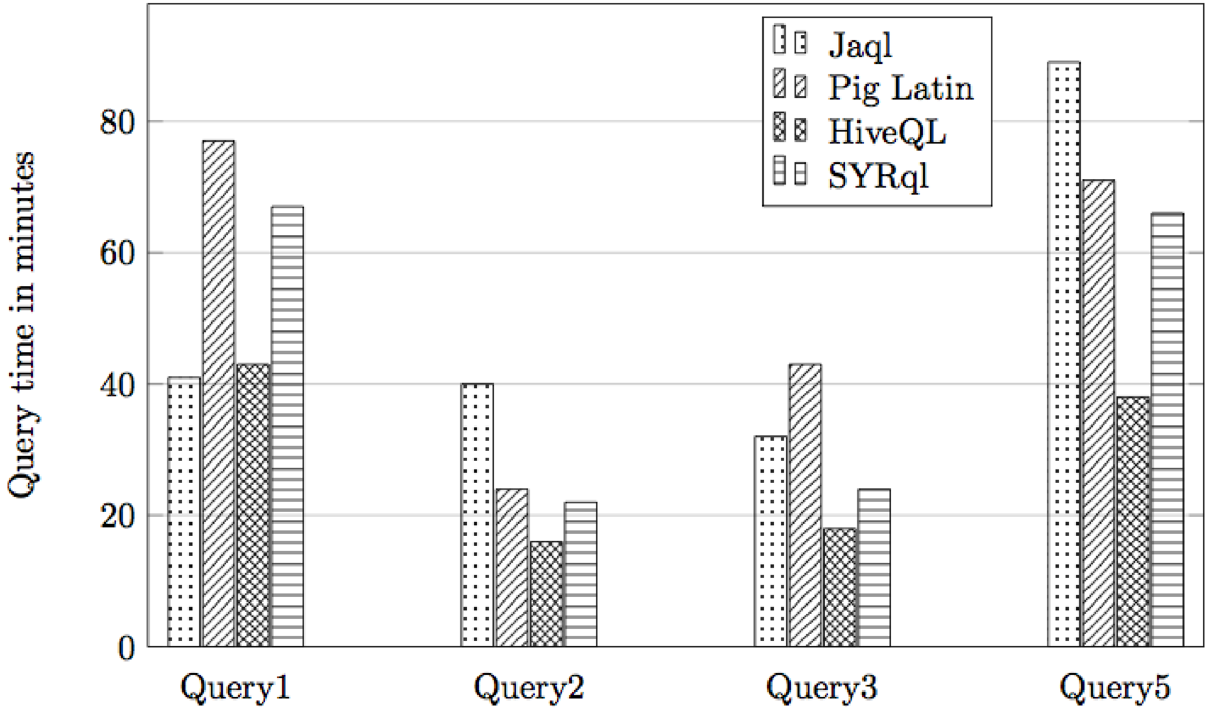
Translates SYRql scripts into a sequence of MapReduce jobs.



SYRql Evaluation

Benchmarked based on Berlin SPARQL Benchmark Business Intelligence Usecase.

140 million triple on 10-node cluster



Conclusion

RDF Algebra provides a closed underlying data model for RDF data processing

RDF Algebra offers unique optimisation opportunities

SYRql, a dataflow language for Big RDF data processing