UPPSALA UNIVERSITET

Scientific SPARQL: Semantic Web Queries over Scientific Data

Andrej Andrejev, Tore Risch
Department of Information Technology, Uppsala University

andrej.andrejev@it.uu.se
tore.risch@it.uu.se

- Introduction
- Motivating example
- System architecture
- SciSPARQL features
- Array processing in SSDM
- Related work

UPPSALA UNIVERSITET

Motivation

RDF is good data model for describing scientific experiments and data, but:

- Scientific data often involves numerical arrays
- No specific array type in RDF
- Collection represented by triples very inefficient to represent arrays

SPARQL is good for searching data and meta-data, including scientific data, but:

- SPARQL has no support for queries involving array operations
- SPARQL has no support for plugging in numerical array algorithms

UPPSALA UNIVERSITET

Our contributions

Extended SPARQL to support queries for scientific applications:
SciSPARQL, Scientific SPARQL

1. SciSPARQL extends SPARQL with numerical operations, in particular
   - numerical array operations
   - user defined functions (e.g. in Java or Python)
   - user defined aggregate functions
   - views defined as functions

2. Implemented SciSPARQL in SSDM:
Scientific SPARQL Database Manager
   - Binary memory-efficient array storage and access
   - Compact representation of views of array slices
     avoids copying arrays in memory
   - Support for (distributed) triple-store back-ends
     array proxy objects

- Introduction
- Motivating example
- System architecture
- SciSPARQL features
- Array processing in SSDM
- Related work

UPPSALA UNIVERSITET

Motivating example

Grid as a 2x3 array of integers:

1	2	3
4	5	6

UPPSALA UNIVERSITET

Motivating example

Grid as a 2x3 array of integers:

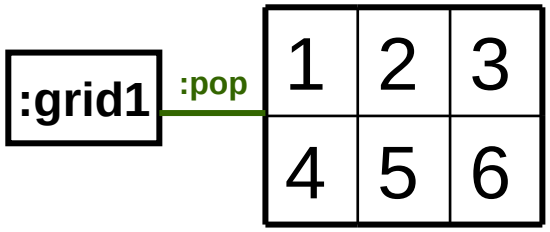
1	2	3
4	5	6

Turtle syntax for a triple, containing this array as a *collection:*

```
:grid1 :pop ((1 2 3)
             (4 5 6)) .
```

Motivating example

Grid as a 2x3 array of integers:



TOTAL: 1 triple

Turtle syntax for a triple, containing this array as a *collection:*

```
:grid1 :pop ((1 2 3)
             (4 5 6)) .
```

rectangular (2x3) array of integers

TOTAL: 17 triples

UPPSALA UNIVERSITET

Example queries

Complete Turtle example:

```
@prefix : <http://udbl.uu.se/pop#> .
:grid1 :species "A" .
:grid1 :pop ((1 2 3)
             (4 5 6)) .
:grid2 :species "B" .
:grid2 :pop ((1 0 1)
             (0 1 0)) .
```

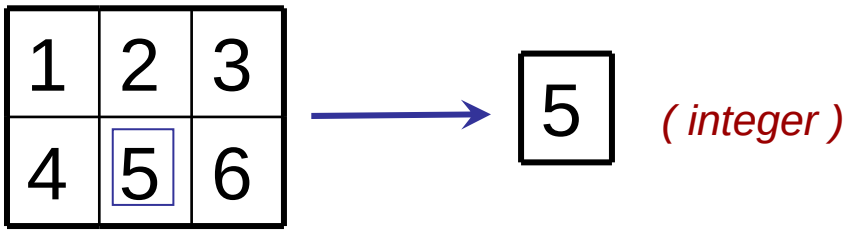
Example queries

Complete Turtle example:

```
@prefix : <http://udbl.uu.se/pop#> .
:grid1 :species "A" .
:grid1 :pop ((1 2 3)
             (4 5 6)) .
:grid2 :species "B" .
:grid2 :pop ((1 0 1)
             (0 1 0)) .
```

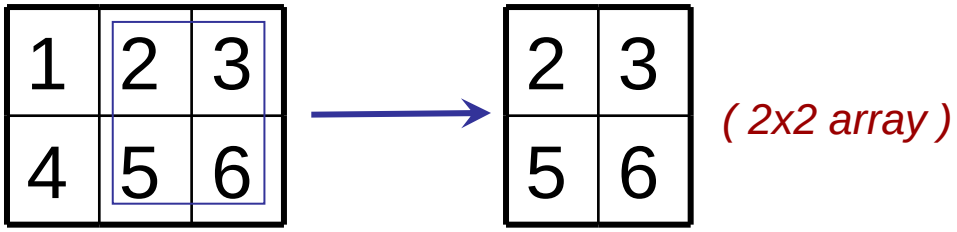
Select array element

```
SELECT (?a[1,1] AS ?res)
 WHERE { ?g :pop ?a .
         ?g :species "A" }
```



Select array slice:

```
SELECT (?a[:,1:3] AS ?res)
 WHERE { ?g :pop ?a .
         ?g :species "A" }
```



Example queries

Complete Turtle example:

```
@prefix : <http://udbl.uu.se/pop#> .
:grid1 :species "A" .
:grid1 :pop ((1 2 3)
             (4 5 6)) .
:grid2 :species "B" .
:grid2 :pop ((1 0 1)
             (0 1 0)) .
```

Select array element

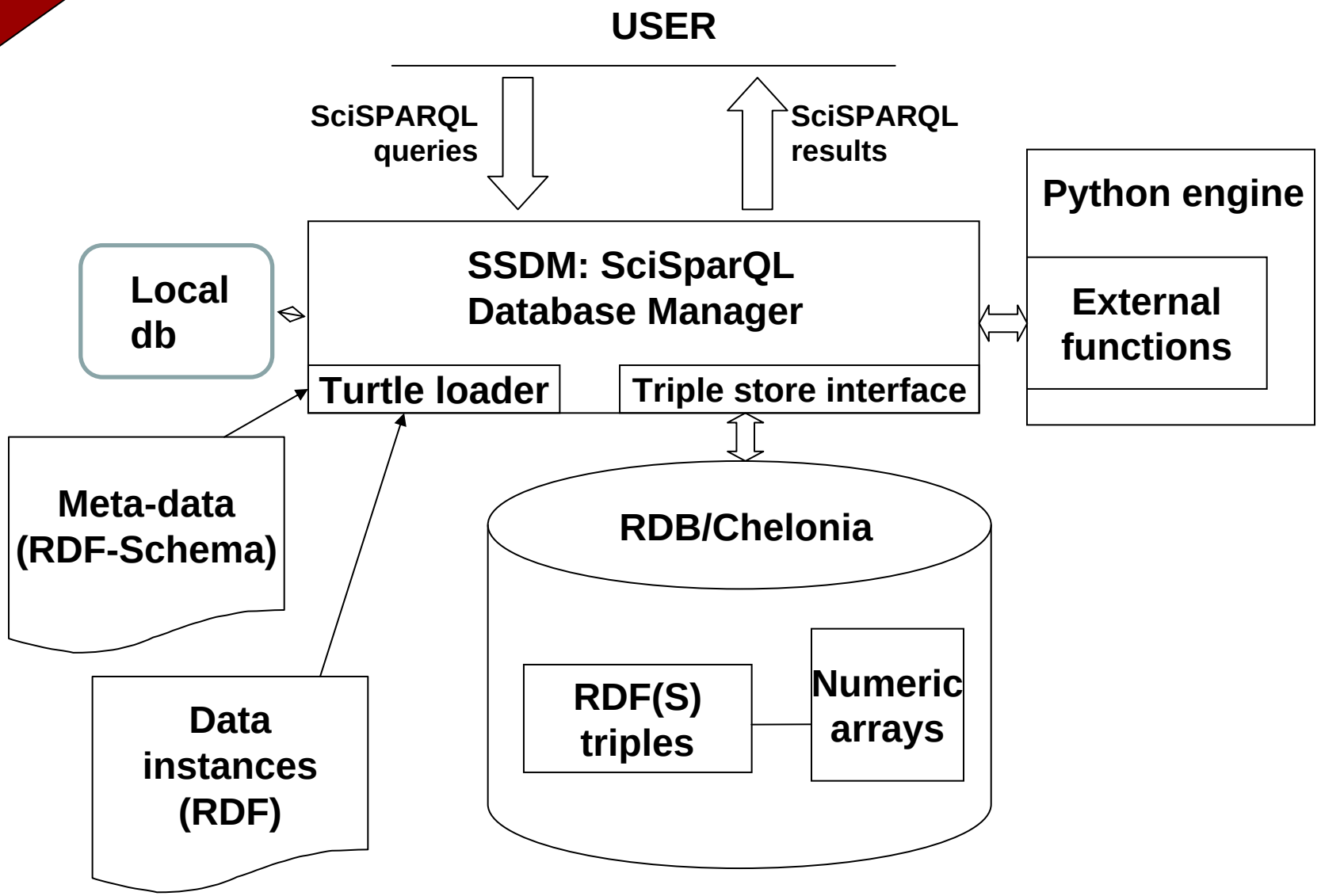
```
SELECT ?elt11
 WHERE { ?g :pop ?a .
         ?g :species "A" .
         ?a rdf:rest ?t1 .
         ?t1 rdf:first ?t2 .
         ?t2 rdf:rest ?t3 .
         ?t3 rdf:first ?elt11 }
```

Select array slice:

?

- Introduction
- Motivating example
- System architecture
- SciSPARQL features
- Array processing in SSDM
- Related work

SSDM System architecture

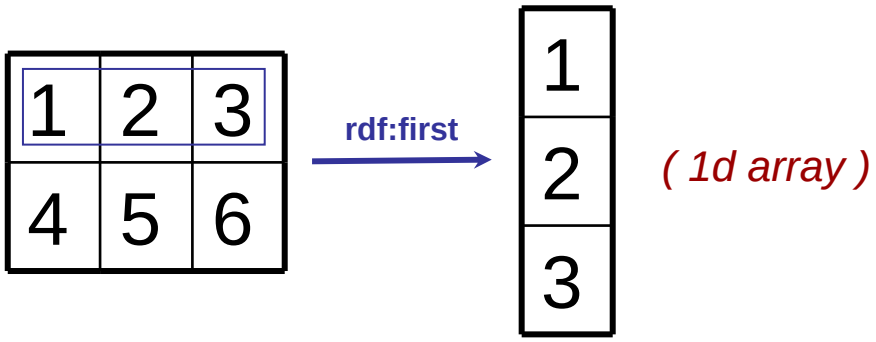


- Introduction
- Motivating example
- System architecture
- SciSPARQL features
- Array processing in SSDM
- Related work

Array operations

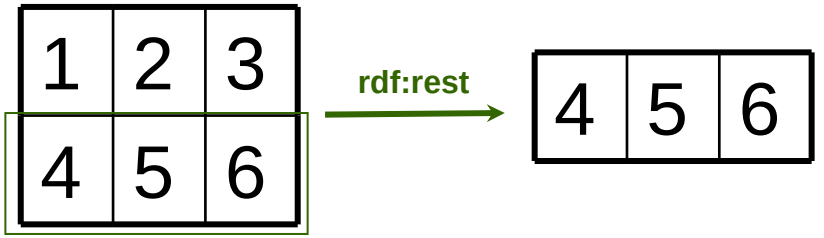
In SPARQL, only **`rdf:first`** and **`rdf:rest`** relationships are defined for collections:

```
SELECT ?res
 WHERE { ?g :pop ?a .
         ?g :species "A" .
         ?a rdf:first ?res }
```



(1d array)

```
SELECT ?res
 WHERE { ?g :pop ?a .
         ?g :species "A" .
         ?a rdf:rest ?res }
```

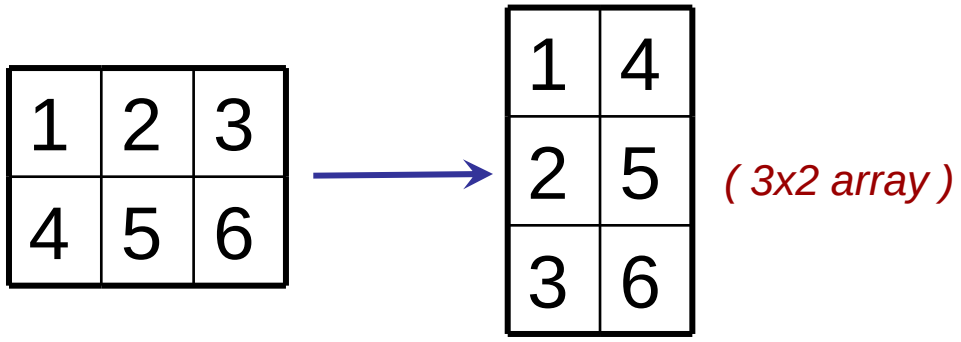


(1x3 array)

SciSPARQL array queries

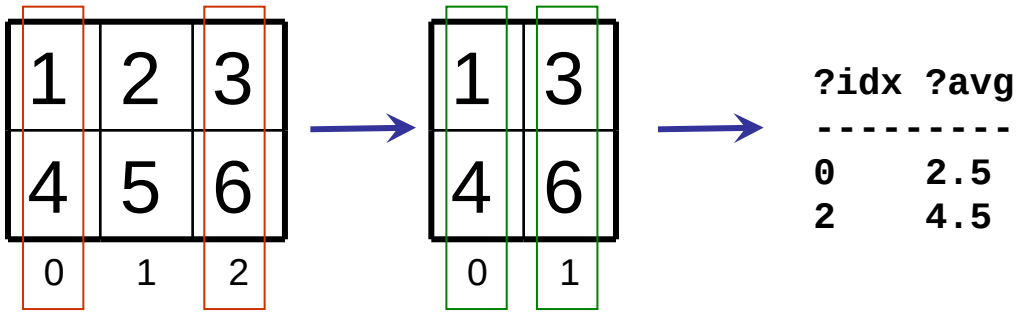
Select transposed matrix

```
SELECT (permute(?a,1,0) AS ?res)
 WHERE { ?g :pop ?a .
         ?g :species "A" }
```



Select average value of every second column, together with column index

```
SELECT (?i*2 AS ?idx)
       (mean(?a[:,::2][:,?i]) AS ?avg)
 WHERE { ?g :pop ?a .
         ?g :species "A" }
```



?idx	?avg
0	2.5
2	4.5

SciSPARQL array queries

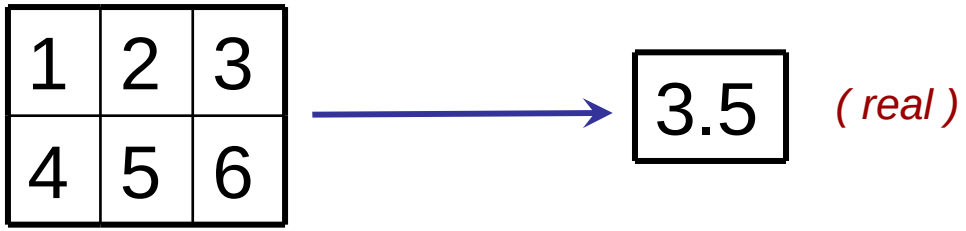
Efficiency problem:

Would like to avoid copying the arrays, and efficiently represent array slices, dimension permutations, and other array operations

SciSPARQL array queries

Select average value of the array:

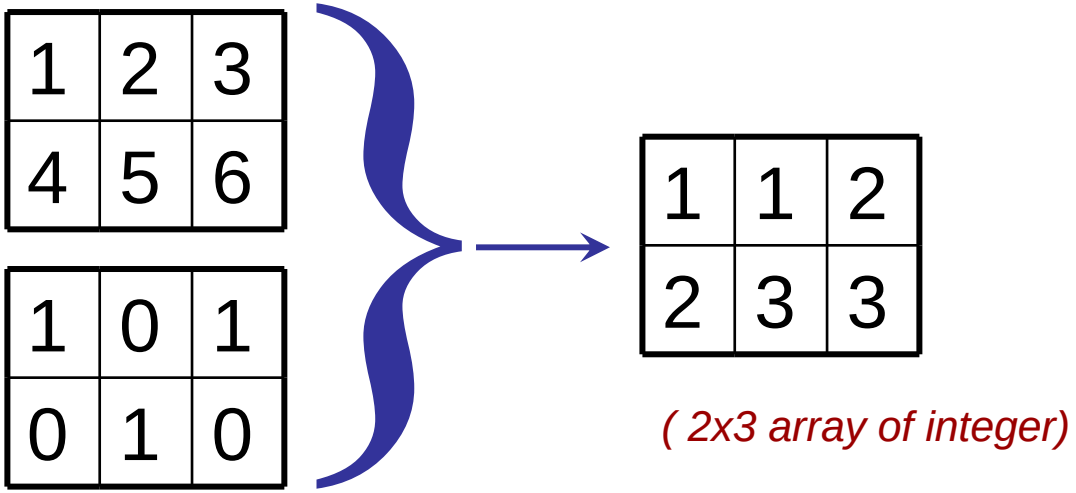
```
SELECT (mean(?a) AS ?res)
 WHERE { ?g :pop ?a .
         ?g :species "A" }
```



Select array of average values (aggregation across several arrays):

```
SELECT (AVG(?a) AS ?res)
 WHERE { ?g :pop ?a }
```

implicit grouping
since aggregate function is applied



SciSPARQL user-defined functions

Define foreign function

```
DEFINE FUNCTION pyplus(?a ?b)
 AS PYTHON 'foreign.plus';
```

Python

```
def plus(a, b): return a+b;
```

Define foreign aggregate function

```
DEFINE AGGREGATE pysum(?b)
 AS PYTHON 'foreign.mysum';
```

Python

```
def mysum(b):
    return sum([i[0] for i in b])
```

Define functional view
(total population per species type)

```
DEFINE FUNCTION totalPop(?species) AS
SELECT (array_sum(?a) AS ?res)
WHERE { ?g :pop ?a .
        ?g :species ?species }
```

- Introduction
- Motivating example
- System architecture
- SciSPARQL features
- Array processing in SSDM
- Related work

How arrays are represented...

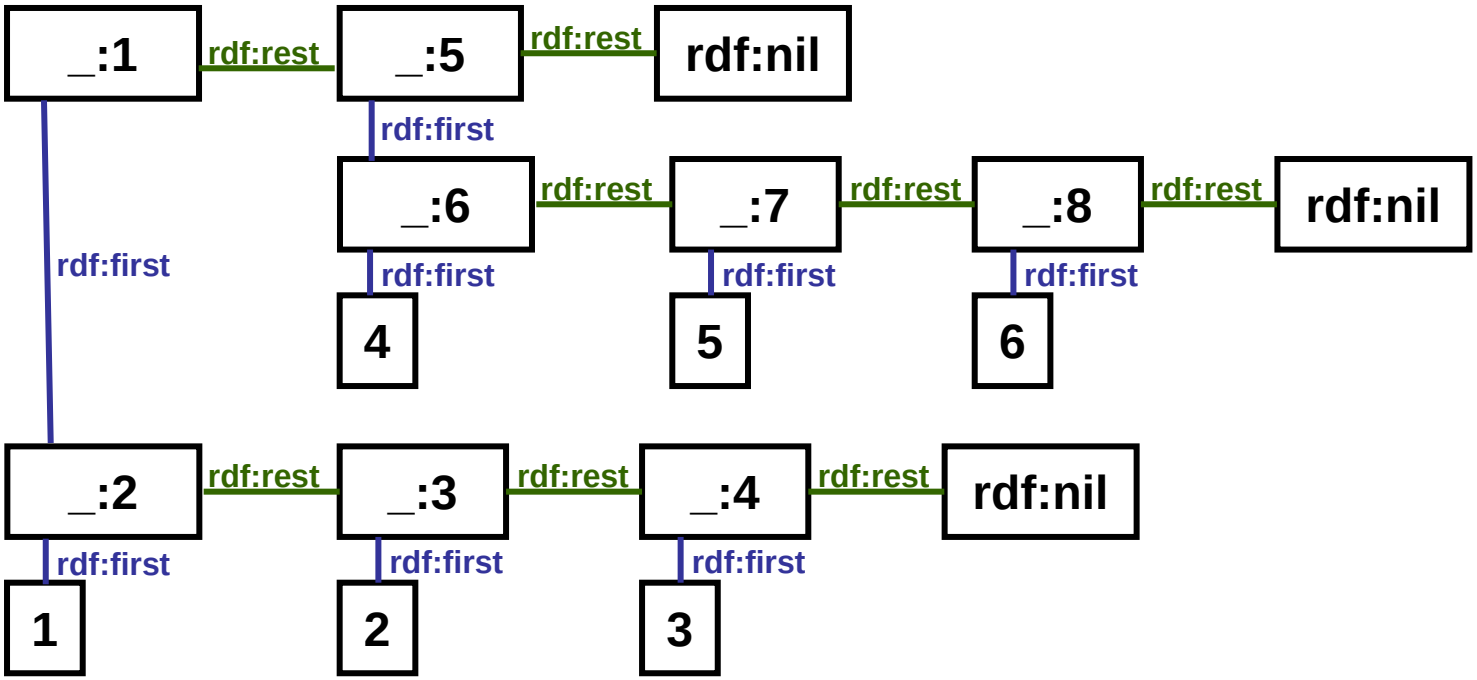
A 2x3 array of integers:

1	2	3
4	5	6

Turtle syntax for a triple, containing this array as a *collection:*

```
:grid1 :pop ((1 2 3)
             (4 5 6)) .
```

Linked-list representation of this collection:



How arrays are represented...

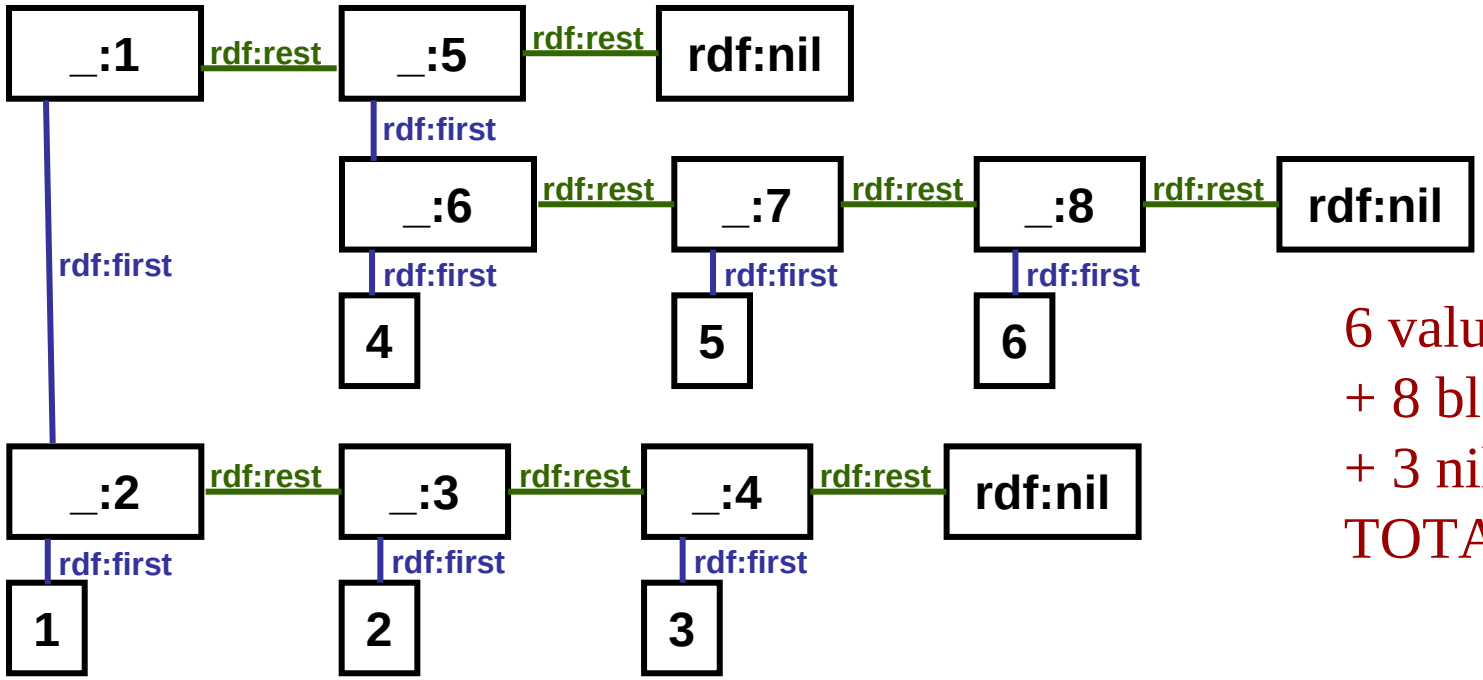
A 2x3 array of integers:

1	2	3
4	5	6

Turtle syntax for a triple, containing this array as a *collection:*

```
:grid1 :pop ((1 2 3)
             (4 5 6)) .
```

Linked-list representation of this collection:



6 value nodes
+ 8 blank nodes
+ 3 nil nodes
TOTAL: 17 nodes
16 arcs

How arrays are represented...

A 2x3 array of integers:

1	2	3
4	5	6

Turtle syntax for a triple, containing this array as a *collection:*

```
@prefix : <http://udbl.uu.se/pop#> .
:grid1 :pop ((1 2 3)
             (4 5 6)) .
```

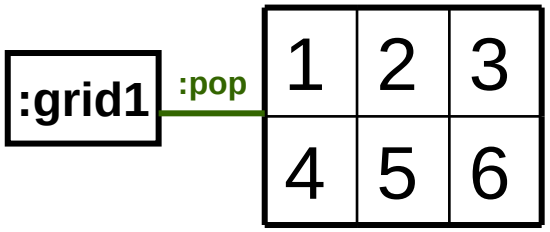
Underlying set of RDF triples (in N-Triples syntax) produced by Jena:

```
<http://udbl.uu.se/pop#grid1> <http://udbl.uu.se/pop#pop> _:1 .
_:1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#first> _:2 .
_:1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> _:5 .
_:2 <http://www.w3.org/1999/02/22-rdf-syntax-ns#first> "1"^^<http://www.w3.org/2001/XMLSchema#integer> .
_:2 <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> _:3 .
_:3 <http://www.w3.org/1999/02/22-rdf-syntax-ns#first> "2"^^<http://www.w3.org/2001/XMLSchema#integer> .
_:3 <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> _:4 .
_:4 <http://www.w3.org/1999/02/22-rdf-syntax-ns#first> "3"^^<http://www.w3.org/2001/XMLSchema#integer> .
_:4 <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> <http://www.w3.org/1999/02/22-rdf-syntax-ns#nil> .
_:5 <http://www.w3.org/1999/02/22-rdf-syntax-ns#first> _:6 .
_:5 <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> <http://www.w3.org/1999/02/22-rdf-syntax-ns#nil> .
_:6 <http://www.w3.org/1999/02/22-rdf-syntax-ns#first> "4"^^<http://www.w3.org/2001/XMLSchema#integer> .
_:6 <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> _:7    .
_:7 <http://www.w3.org/1999/02/22-rdf-syntax-ns#first> "5"^^<http://www.w3.org/2001/XMLSchema#integer> .
_:7 <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> _:8 .
_:8 <http://www.w3.org/1999/02/22-rdf-syntax-ns#first> "6"^^<http://www.w3.org/2001/XMLSchema#integer> .
_:8 <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> <http://www.w3.org/1999/02/22-rdf-syntax-ns#nil> .
```

TOTAL: 17 triples

How arrays are represented...

A 2x3 array of integers:



TOTAL: 1 triple

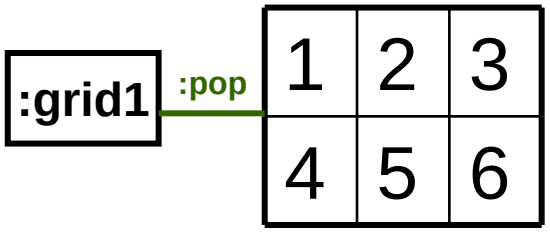
Turtle syntax for a triple, containing this array as a *collection:*

```
:grid1 :pop ((1 2 3)
             (4 5 6)) .
```

rectangular (2x3) array of integers

How arrays are represented...

A 2x3 array of integers:



TOTAL: 1 triple

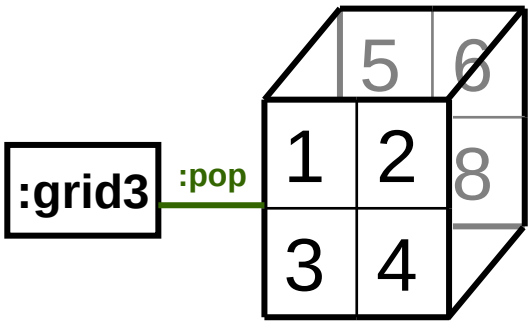
Turtle syntax for a triple, containing this array as a *collection:*

```
:grid1 :pop ((1 2 3)
             (4 5 6)) .
```

rectangular (2x3) array of integers

```
:grid3 :pop (((1 2) (3 4))
             ((5 6) (7 8))).
```

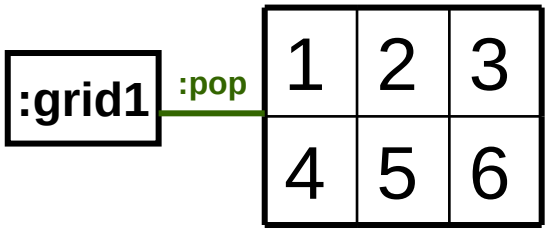
rectangular (2x2x2) array of integers



TOTAL: 1 triple
vs. 29 triples
in naive representation

How arrays are represented...

A 2x3 array of integers:



TOTAL: 1 triple

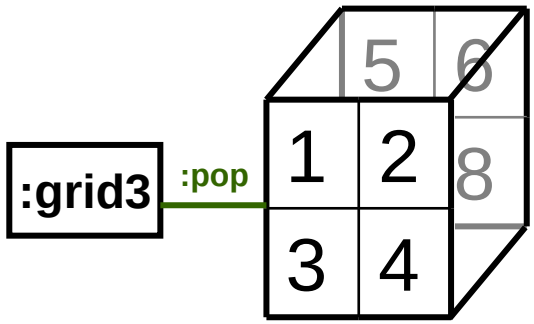
Turtle syntax for a triple, containing this array as a *collection:*

```
:grid1 :pop ((1 2 3)
             (4 5 6)) .
```

rectangular (2x3) array of integers

```
:grid3 :pop (((1 2) (3 4))
             ((5 6) (7 8))) .
```

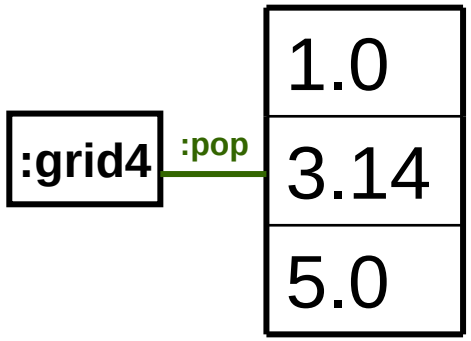
rectangular (2x2x2) array of integers



TOTAL: 1 triple
vs. 29 triples
in naive representation

```
:grid4 :pop (1 3.14 5) .
```

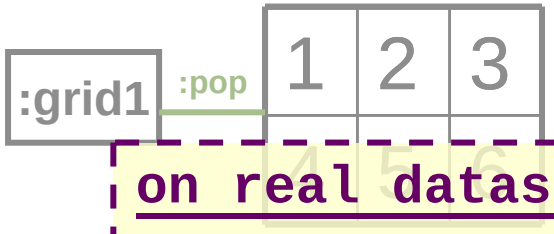
1d array of real numbers



TOTAL: 1 triple
vs. 7 triples
in naive representation

How arrays are represented...

A 2x3 array of integers:



Turtle syntax for a triple, containing this array as a *collection:*

```
:grid1 :pop ((1 2 3)
            (4 5 6)) .
```

rectangular (2x3) array of integers

on real datasets:

725 triples, 125 1d arrays 70 elements each:

~20 times less memory used

86 triples, 86 2d arrays 10k elements each:

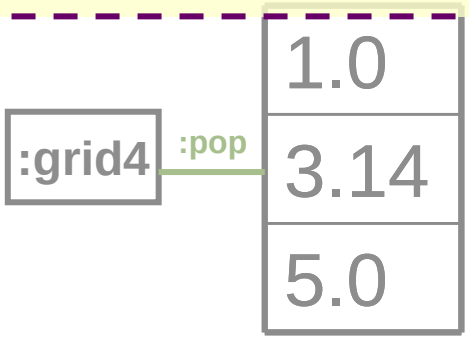
~81 times less memory used

```
:grid3 :pop (((1 2) (3 4))
             ((5 6) (7 8))) .
```

rectangular (2x2x2) array of integers

```
:grid4 :pop (1 3.14 5) .
```

1d array of real numbers



TOTAL: 1 triple
vs. 7 triples
in naive representation

UPPSALA UNIVERSITET

How arrays are represented...

```
:s :p ( 1 (2 3) 4) .
```

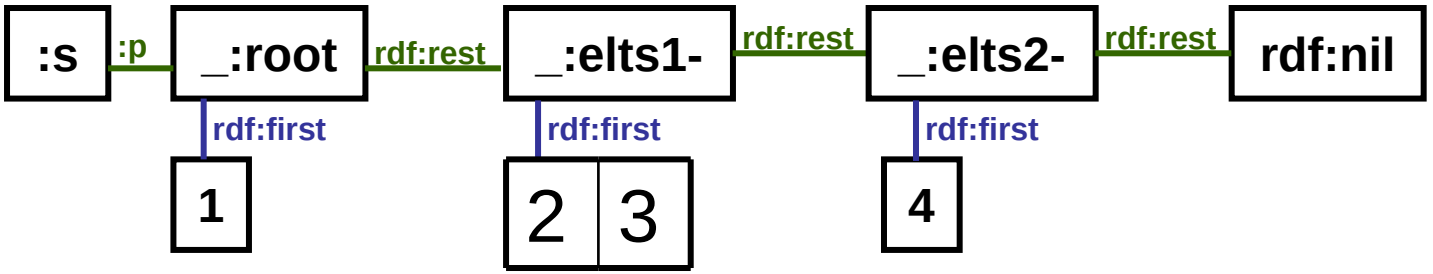
not an array

How arrays are represented...

```
:s :p ( 1 (2 3) 4) .
```

not an array

1d array of integers



TOTAL:
7 triples

Array storage and descriptors

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{pmatrix}$$

type	size	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
integer	**15**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**

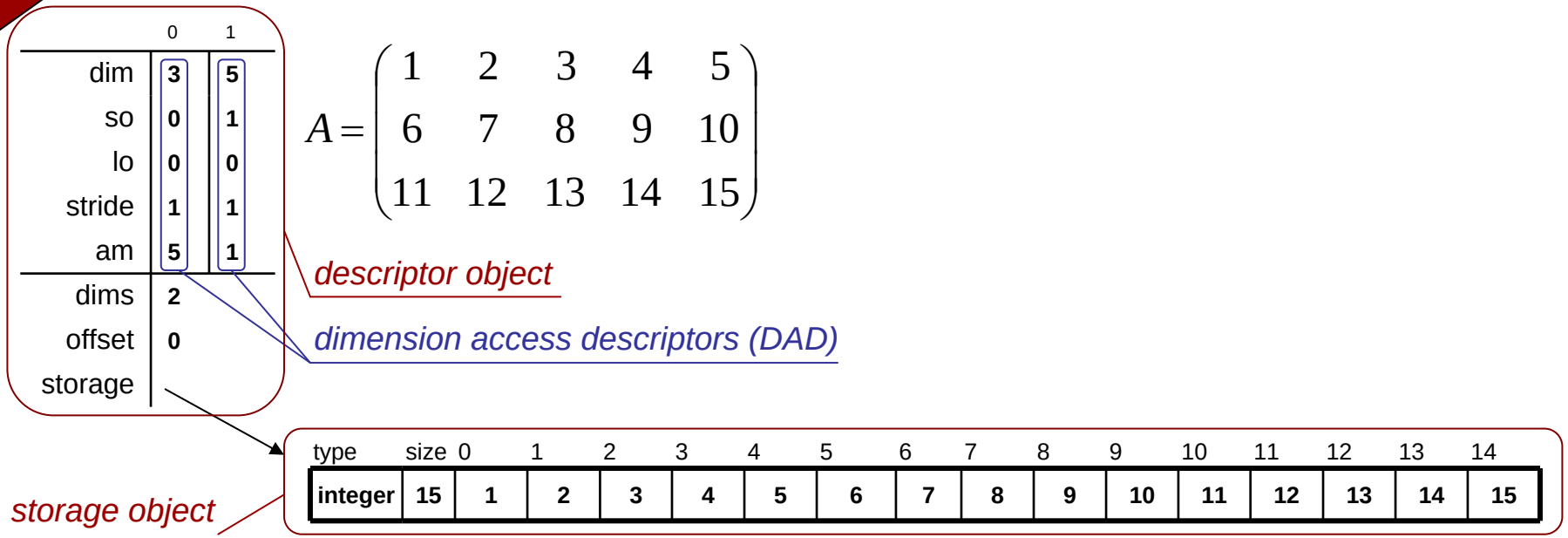
Array storage and descriptors

	0	1
dim	**3**	**5**
so	**0**	**1**
lo	**0**	**0**
stride	**1**	**1**
am	**5**	**1**
dims	**2**	
offset	**0**	
storage		

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{pmatrix}$$

type	size	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
integer	**15**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**

Array storage and descriptors



$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{pmatrix}$$

Array storage and descriptors

	0	1
dim	3	5
so	0	1
lo	0	0
stride	1	1
am	5	1
dims	2	
offset	0	
storage		

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{pmatrix}$$

descriptor object

dimension access descriptors (DAD)

$$A^T = \begin{pmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{pmatrix}$$

	0 ↔ 1	
dim	5	3
so	1	0
lo	0	0
stride	1	1
am	1	5
dims	2	
offset	0	
storage		

storage object

type	size	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
integer	15	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Array storage and descriptors

	0	1
dim	3	5
so	0	1
lo	0	0
stride	1	1
am	5	1
dims	2	
offset	0	
storage		

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{pmatrix}$$

descriptor object

dimension access descriptors (DAD)

$$A^T = \begin{pmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{pmatrix}$$

	0 ↔ 1	
dim	5	3
so	1	0
lo	0	0
stride	1	1
am	1	5
dims	2	
offset	0	
storage		

storage object

type	size	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
integer	15	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

?a[1:3,0:5:2]=

$$\begin{pmatrix} 6 & 8 & 10 \\ 11 & 13 & 15 \end{pmatrix}$$

	0	1
dim	2	3
so	0	1
lo	1	0
stride	1	2
am	5	1
dims	2	
offset	0	
storage		

Array storage and descriptors

	0	1
dim	3	5
so	0	1
lo	0	0
stride	1	1
am	5	1
dims	2	
offset	0	
storage		

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{pmatrix}$$

descriptor object

dimension access descriptors (DAD)

$$A^T = \begin{pmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{pmatrix}$$

	0 ↔ 1	
dim	5	3
so	1	0
lo	0	0
stride	1	1
am	1	5
dims	2	
offset	0	
storage		

storage object

type	size	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
integer	15	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

?a[1:3,0:5:2]=

$$\begin{pmatrix} 6 & 8 & 10 \\ 11 & 13 & 15 \end{pmatrix}$$

	0	1
dim	2	3
so	0	1
lo	1	0
stride	1	2
am	5	1
dims	2	
offset	0	
storage		

?a[1,:]=

$$\begin{pmatrix} 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{pmatrix}$$

	0
dim	5
so	1
lo	0
stride	1
am	1
dims	1
offset	5
storage	

Array storage and descriptors

	0	1
dim	3	5
so	0	1
lo	0	0
stride	1	1
am	5	1
dims	2	
offset	0	
storage		

descriptor object

dimension access descriptors (DAD)

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{pmatrix}$$

$$A^T = \begin{pmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{pmatrix}$$

	0 ↔ 1	
dim	5	3
so	1	0
lo	0	0
stride	1	1
am	1	5
dims	2	
offset	0	
storage		

storage object

type	size	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
integer	15	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

?a[1:3,0:5:2]=

$$\begin{pmatrix} 6 & 8 & 10 \\ 11 & 13 & 15 \end{pmatrix}$$

	0	1
dim	2	3
so	0	1
lo	1	0
stride	1	2
am	5	1
dims	2	
offset	0	
storage		

?a[1,:]=

$$\begin{pmatrix} 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{pmatrix}$$

	0
dim	5
so	1
lo	0
stride	1
am	1
dims	1
offset	5
storage	

?a[:,2]=

$$\begin{pmatrix} 3 \\ 8 \\ 13 \end{pmatrix}$$

	0
dim	3
so	0
lo	0
stride	1
am	5
dims	1
offset	2
storage	

- Introduction
- Motivating example
- System architecture
- SciSPARQL features
- Array processing in SSDM
- Related work

UPPSALA UNIVERSITET

Related work

- Other SPARQL language extensions:
  - Semantic Paths [Kochut & Janik, 2007]
  - Windows over streams of RDF [Barbieri et.al., 2010]
- Extending RDBMS with array semantics
  - AQuery [Lerner & Shasha, 2003]
  - SciQL [Kersten et.al, 2011]
- Storing arrays as BLOBs
  - RasDaMan [Furtado & Baumann, 1999]
  - UDFs in T-SQL [Dobos et.al., 2011]
- Specialized array databases and languages
  - APL [Iverson 1962]
  - AQL [Libkin et.al. 1996]
  - SciDB [Cudre-Mauroux et.al. 2009]
- Foreign Functions in SPARQL
  - SESAME
  - CORESE

Summary and future work

Extended SPARQL to support queries for scientific applications:
SciSPARQL, Scientific SPARQL

Implemented SciSPARQL in SSDM:
Scientific SPARQL Database Manager

Can be downloaded from
http://www.it.uu.se/research/group/udbl/ssdm/

Ongoing and future work:
Evaluation on real-life examples,
Performance evaluations and tuning,
Integration with scalable back-end storage,
Array specific query optimization

THANK YOU

```
@prefix : <http://udbl.uu.se/YeastPolarization#> .
:Experiment001 a :YeastPolarizationExperiment ;
:ModelName "ALL_Alt" ;
:ModelVersion 1 ;
:SimulationAlgorithm “ISSA” ;
:InputType
"GradientWithSwitching_Input” ;
:Diffusion 0.01 ;
:TimeStep 30 .
[] a :TrajectoryData ;
:inExperiment :Experiment001 ;
:Km 10 ;
:kon 0.01 ;
:TrajNo 1 ;
:Width (0 17.82 10.8 34.1 ) #typically longer!
[] a :TrajectoryData ;
:inExperiment :Experiment001 ;
:Km 10 ;
:kon 0.01 ;
:TrajNo 1 ;
:Width (0 3.56 12.4 22.41 )
```

UPPSALA UNIVERSITET

1) What is the mean and variance of the values of each trajectory, having *kon* parameter below 0.05?

```
PREFIX : <http://udbl.uu.se/YeastPolarization#>
SELECT ?Km ?kon ?TrajNo
       (mean(?Width) AS ?WidthMean)
       (variance(?Width) AS ?WidthVariance)
WHERE { ?trData a :TrajectoryData ;
                :inExperiment :Experiment001 ;
                :Km ?Km ;
                :kon ?kon ;
                :TrajNo ?trajNo ;
                :Width ?Width .
        FILTER (?kon < 0.05) }
```

UPPSALA UNIVERSITET

2) For each combination of the parameters *km* and *kon*, where *kon* is below 0.05, compute the *mean trajectory* where each value is the average of the stored trajectory values.

```
PREFIX : <http://udbl.uu.se/YeastPolarization#>
SELECT ?Km ?kon
       (meanAgg(?Width) AS ?MeanTrajectory)
WHERE { ?trData a :TrajectoryData ;
                :inExperiment :Experiment001 ;
                :Km ?Km ;
                :kon ?kon ;
                :Width ?Width .
        FILTER (?kon < 0.05) }
```

UPPSALA UNIVERSITET

3) What is the mean of the last five trajectory values in trajectories with time step of 2 minutes?

```
PREFIX : <http://udbl.uu.se/YeastPolarization#>
SELECT ?Km ?kon ?TrajNo ?step
      (mean(?Width[((adims(?Width)[0]-4*?step)-1)
                    ::?step]) AS ?L5Mean)
WHERE { ?trData a :TrajectoryData ;
                :inExperiment :Experiment001 ;
                :Km ?Km ;
                :kon ?kon ;
                :TrajNo ?trajNo ;
                :Width ?Width .
        :Experiment001 :TimeStep ?timestep .
        BIND (round(120/?timestep) AS ?step) .
        FILTER (?kon < 0.05) ;
```

UPPSALA UNIVERSITET

4) Define a function computing the final time of a given parameter *?trajectory*:

```
PREFIX : <http://udbl.uu.se/YeastPolarization#>
DEFINE FUNCTION final_time(?trajectory)
 AS SELECT ((adims(?width)[0]-1)*?timestep AS ?res)
     WHERE { ?trajectory :inExperiment ?experiment ;
                         :Width ?width .
             ?experiment :TimeStep ?timestep }
```