Semantic Web Queries over Scientific Data

Andrej Andrejev

Uppsala Database Lab
Department of Information Technology, Uppsala University

- Introduction
- RDF & SPARQL
- RDF with Arrays & Scientific SPARQL
- Scientific SPARQL Database Manager
- Extensible Array Storage
- Array Query Benchmark
- Summary

Data and Metadata

Massive numeric data, e.g.

- instrumental measurements
- simulations
- solutions to Partial Differential Equations

…

Typically ordered along a number of orthogonal axes.

- experiment setup
- tools and methods used
- realization parameters
- data structure and quantities
- provenance (links to input data)
- usage restrictions

…

Data and Metadata

Massive numeric data, e.g.

- instrumental measurements
- simulations
- solutions to Partial Differential Equations

...

Typically ordered along a number of orthogonal axes.

- experiment setup
- tools and methods used
- realization parameters
- data structure and quantities
- provenance (links to input data)
- usage restrictions

...

scale beyond storage capacity of a single server

>>

fit into main memory

Scientific Data Management Practice

- Different data providers, terminology, level of detail
- Different storage formats
- Typically, need to download and convert before doing any meaningful analysis
- Need to manually write scripts / software
- Not all metadata is explicitly stored in the dataset: specialists responsible for generating the data should be involved when analyzing it

Scientific Data Management Practice

- Different data providers, terminology, level of detail
- Different storage formats
- Typically, need to download and convert before doing any meaningful analysis
- Need to manually write scripts / software
- Not all metadata is explicitly stored in the dataset: specialists responsible for generating the data should be involved when analyzing it

Example:
PhD position in High Energy Physics
main requirement: proficiency in C

- Introduction
- RDF & SPARQL
- RDF with Arrays & Scientific SPARQL
- Scientific SPARQL Database Manager
- Extensible Array Storage
- Array Query Benchmark
- Summary

RDF* and Semantic Web

*Resource Description Framework

- graph-based data model
  - adding a new node or edge to a graph is easier than adding a column to a table
- no need to define schema upfront
  - low cost of entry
- nodes and edges are identified with globally-unique terms, identified by URIs
- common vocabularies (“ontologies”) of such terms, e.g. FOAF, DublinCore, SDMX, and more specialized ones

RDF* and Semantic Web

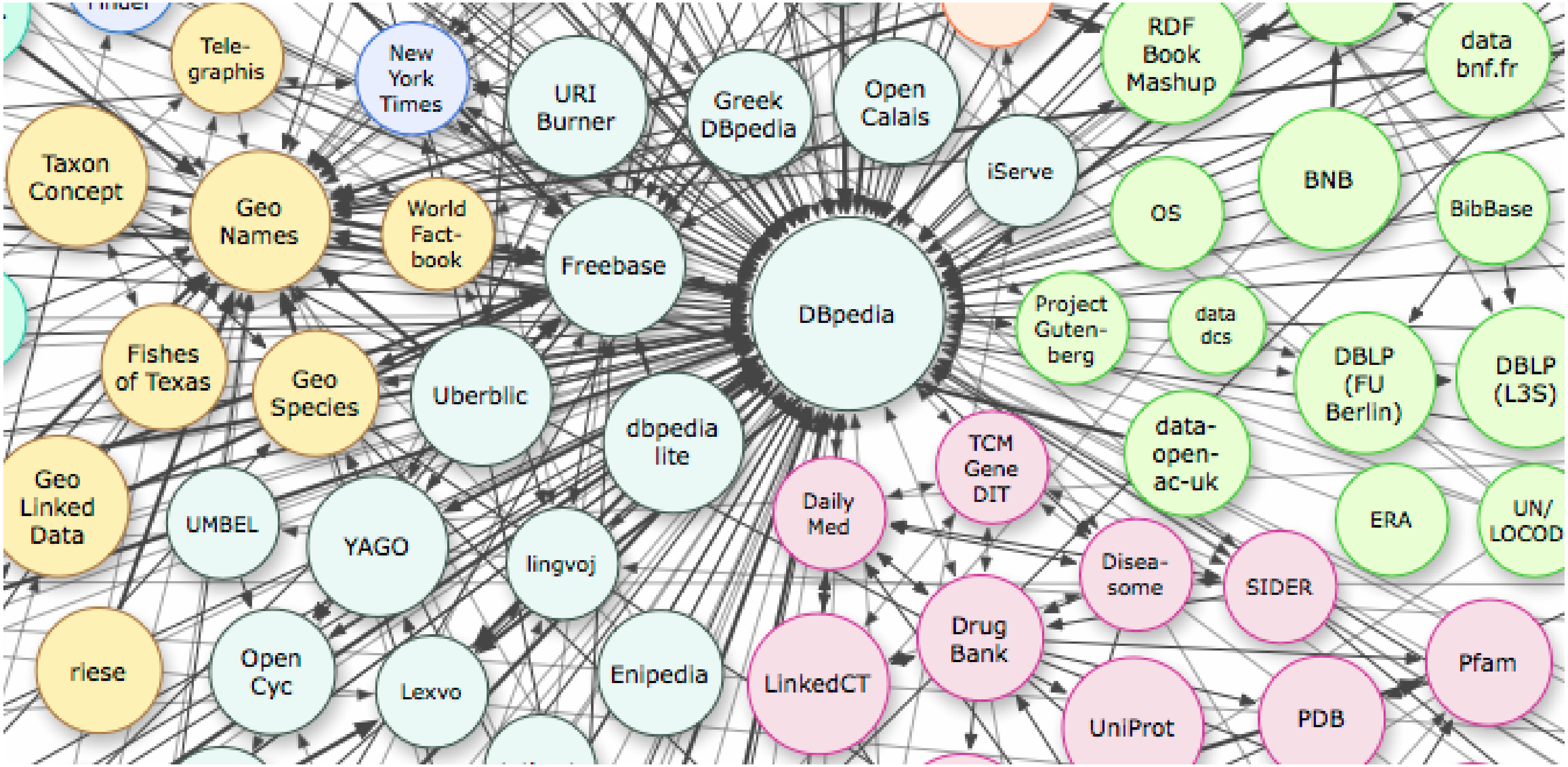
*Resource Description Framework

- vocabularies may refer to each other via logical relationships, e.g. *owl:sameAs*, *rdfs:subPropertyOf*, etc.
  - RDF Schema & OWL are the basic data modeling tools
  - RIF & SWRL are used to define knowledge inference rules

- well-known RDF datasets such as DBPedia are used as hubs to make other RDF datasets interconnected, by referring to the common entities

- lots of general-purpose RDF data are already published and available online

Linked Open Data



- Introduction
- RDF & SPARQL
- RDF with Arrays & Scientific SPARQL
- Scientific SPARQL Database Manager
- Extensible Array Storage
- Array Query Benchmark
- Summary

SPARQL*

*SPARQL Protocol and Query Language

- declarative query language
  - users specify ***what*** data to retrieve, not ***how***
- graph pattern matching
  - possible to express complex semantic relationships
  - regular path expressions: “recursive queries”
  - well-equipped to handle datasets with incomplete, redundant, or conflicting data, common in Linked Data context
- grouping and aggregation
  - specifying graph data summaries and levels of detail
  - condensing and filtering RDF data before returning the result set

→ **graph query and transformation language**

Research Questions

1. How can RDF and SPARQL be extended to be suitable for scientific and engineering numeric data representation and analysis tasks?
   In particular, those which combine data and metadata?

2. How can extended SPARQL query processing be implemented on the basis of a database management system?

Research Questions

1. How can RDF and SPARQL be extended to be suitable for scientific and engineering numeric data representation and analysis tasks?
   In particular, those which combine data and metadata?

   RDF with Arrays

   Scientific SPARQL

2. How can extended SPARQL query processing be implemented on the basis of a database management system?

Research Questions

1. How can RDF and SPARQL be extended to be suitable for scientific and engineering numeric data representation and analysis tasks?
   In particular, those which combine data and metadata?

RDF with Arrays

Scientific SPARQL

2. How can extended SPARQL query processing be implemented on the basis of a database management system?

Scientific SPARQL Database Manager (SSDM)

- Introduction
- RDF & SPARQL
- RDF with Arrays & Scientific SPARQL
- Scientific SPARQL Database Manager
- Extensible Array Storage
- Array Query Benchmark
- Summary

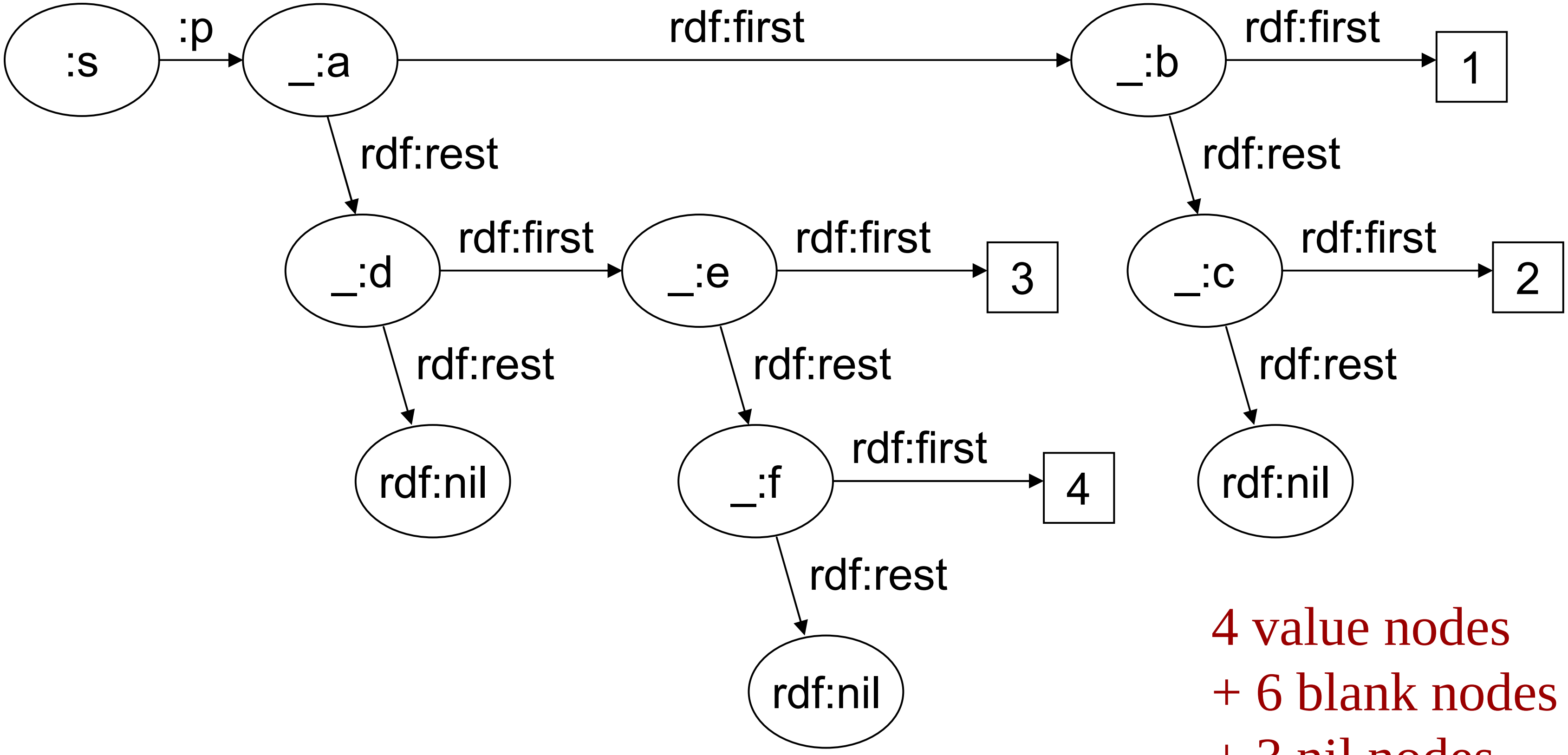
RDF: Collections

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

RDF: Collections

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

RDF graph:

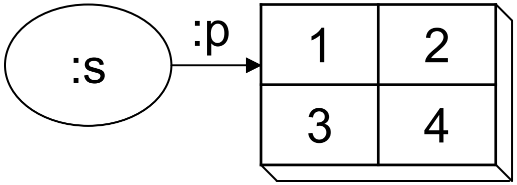


4 value nodes
+ 6 blank nodes
+ 3 nil nodes
TOTAL: 14 nodes
13 edges

RDF with Arrays

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

RDF with Arrays graph:



1	2
3	4

TOTAL: 2 nodes
1 edge

- **Graph queries: faster**
- **Array queries: possible**

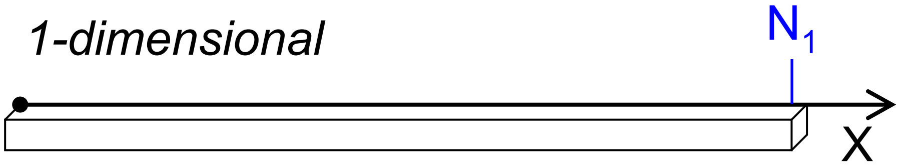
Array Model

$$A: \underbrace{\{1..N_1\} \times \ldots \times \{1..N_n\}}_{\text{domain}} \rightarrow \underbrace{R}_{\text{range}}$$

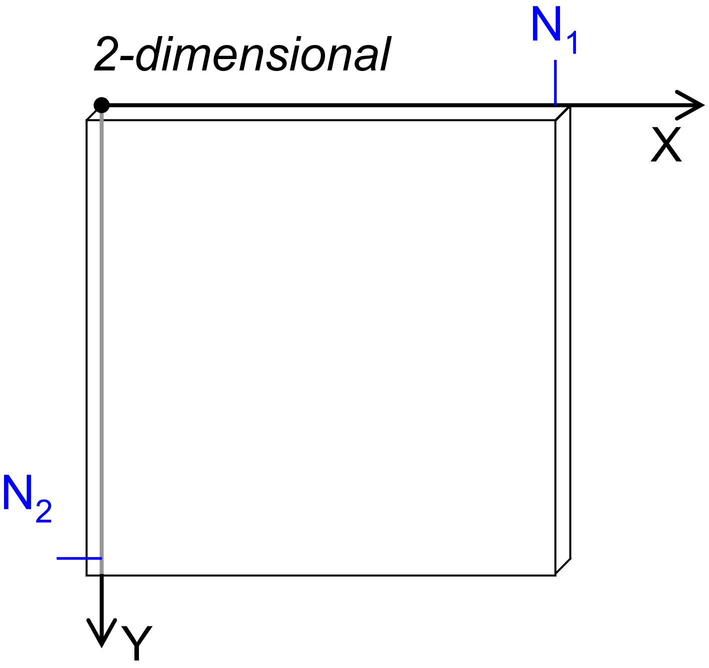
array shape:

$$<N_1 \ldots N_n>$$

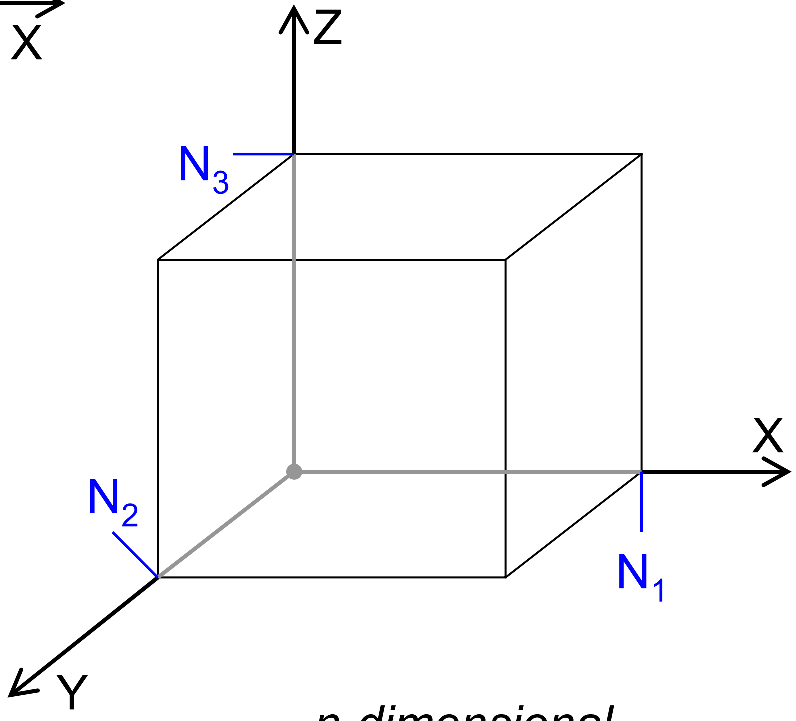
1-dimensional



2-dimensional



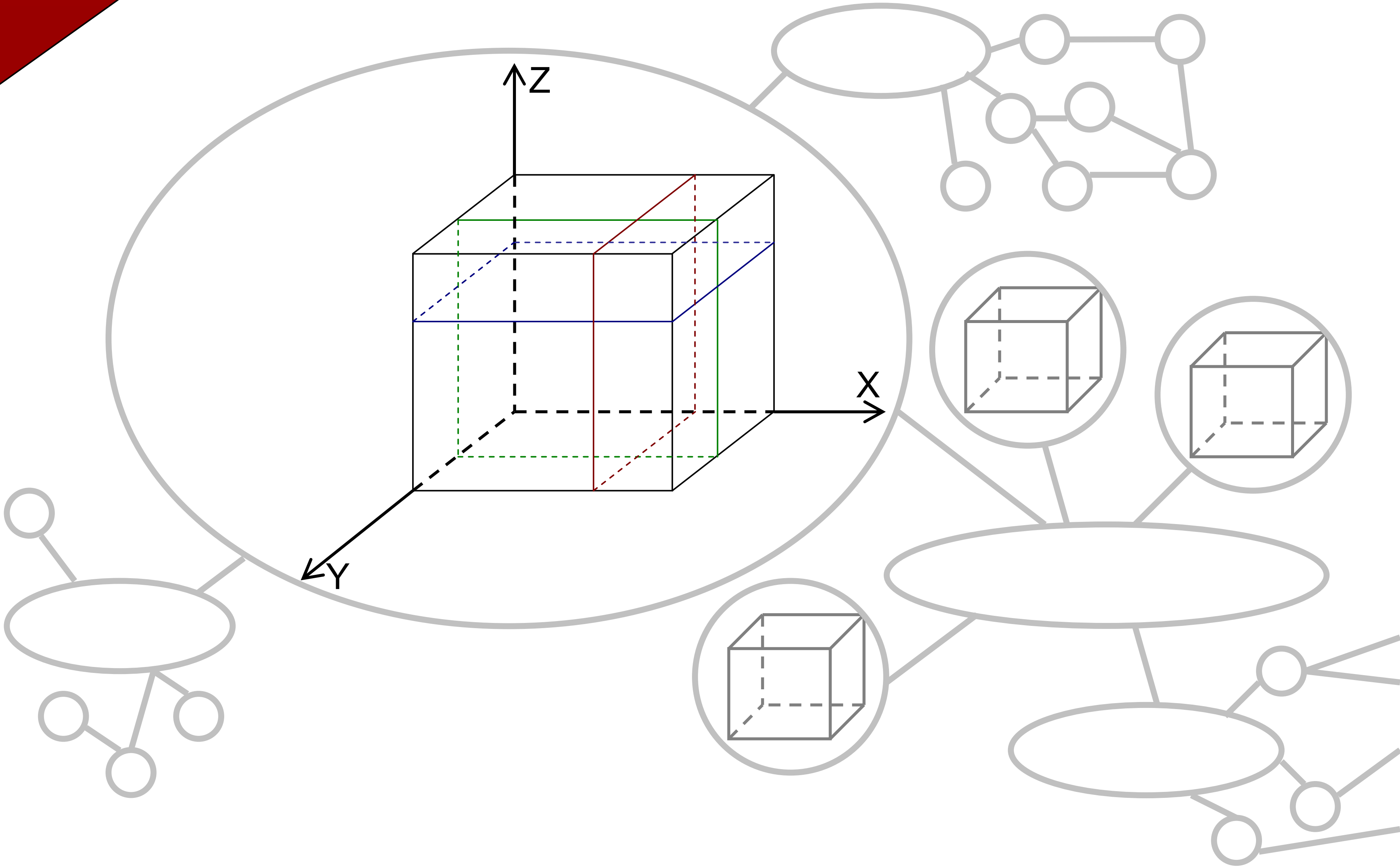
3-dimensional



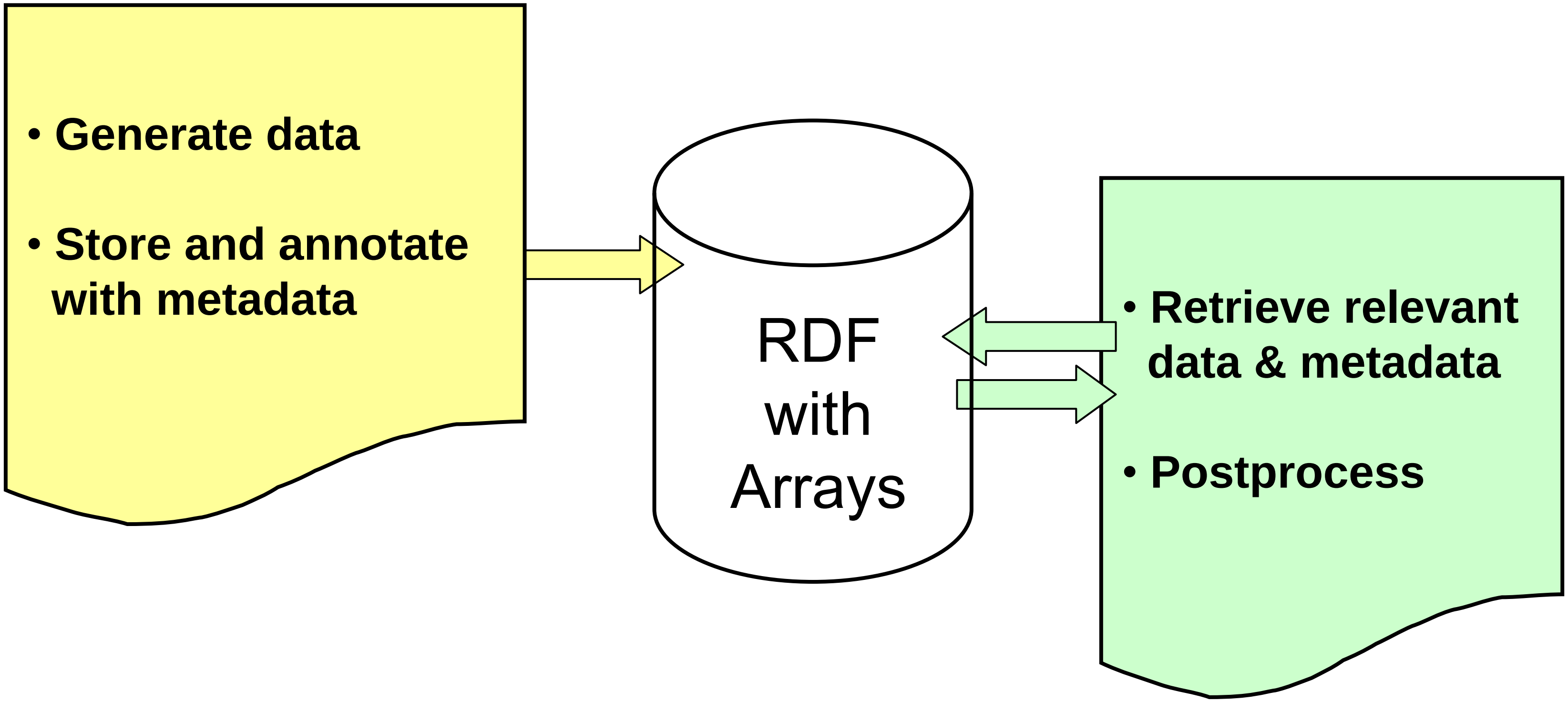
n-dimensional

?

RDF with Arrays



Workflows

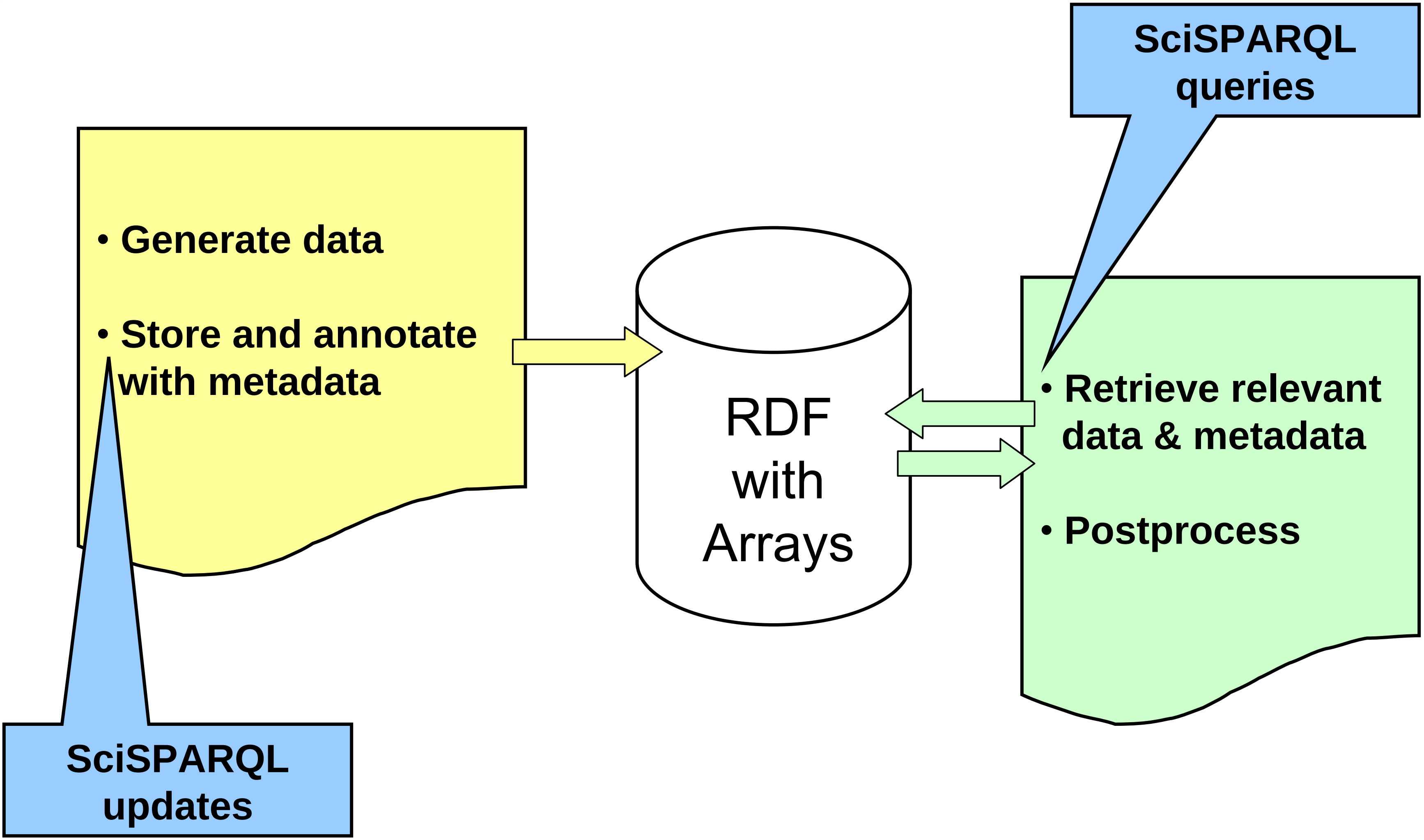


- **Generate data**
- **Store and annotate with metadata**

RDF with Arrays

- **Retrieve relevant data & metadata**
- **Postprocess**

Workflows



- Generate data
- Store and annotate with metadata

SciSPARQL updates

RDF with Arrays

SciSPARQL queries

- Retrieve relevant data & metadata
- Postprocess

- Introduction
- RDF & SPARQL
- RDF with Arrays & Scientific SPARQL
- Scientific SPARQL Database Manager
- Extensible Array Storage
- Array Query Benchmark
- Summary

Scientific SPARQL

A strict superset of W3C SPARQL 1.1 Standard

+

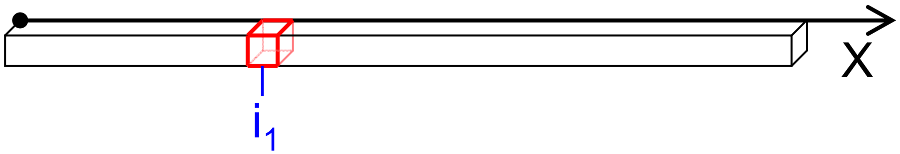
- Array operations

Array Element Access

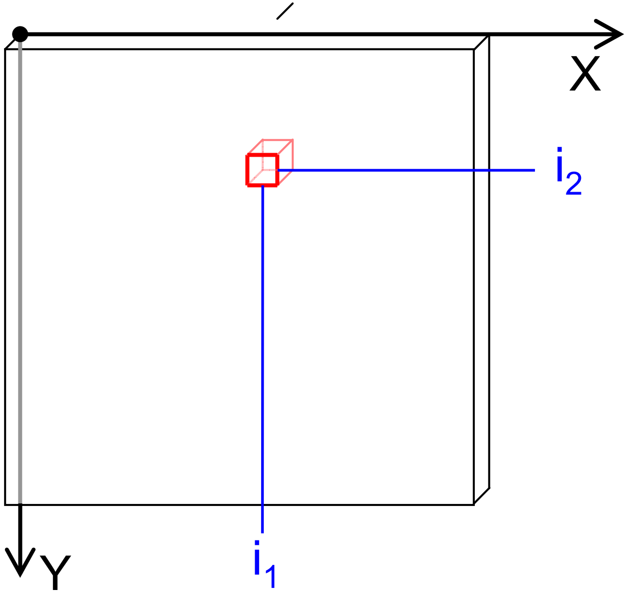
$$A[\underbrace{i_1, \dots i_n}_{\text{subscripts}}]$$

$$1 \leq i_k \leq N_k$$

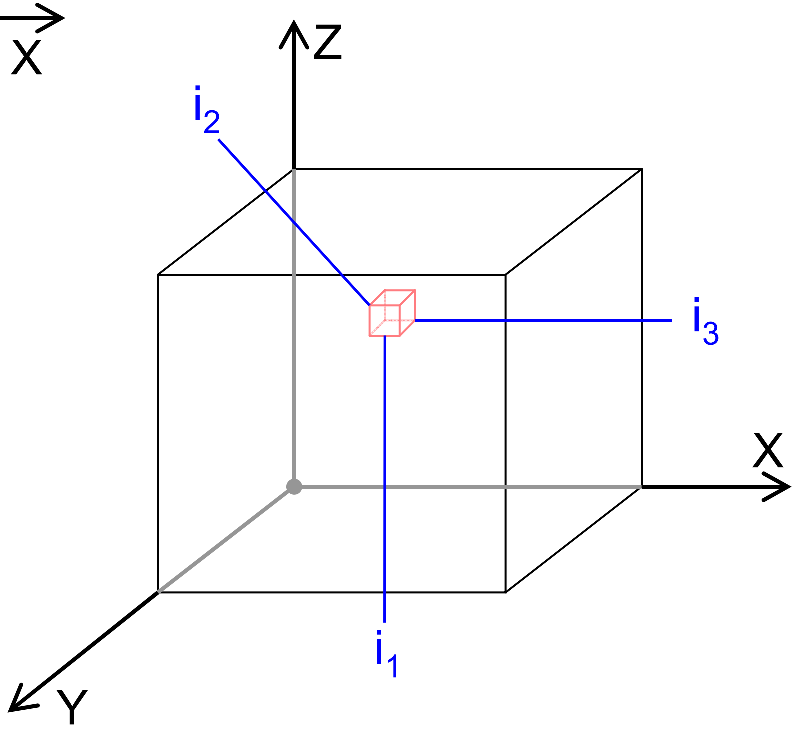
1-dimensional



2-dimensional



3-dimensional

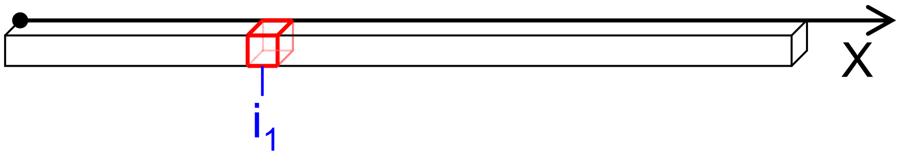


Array Projection

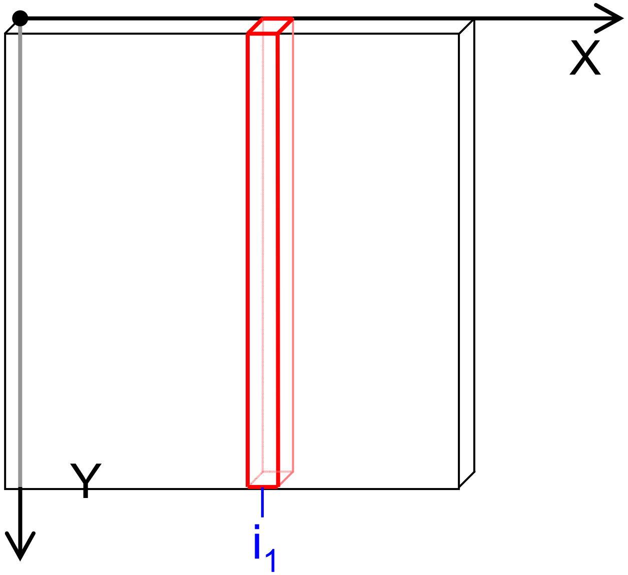
e.g. along the first dimension

$A[i_1]$ $\quad\quad 1 \leq i_1 \leq N_1$

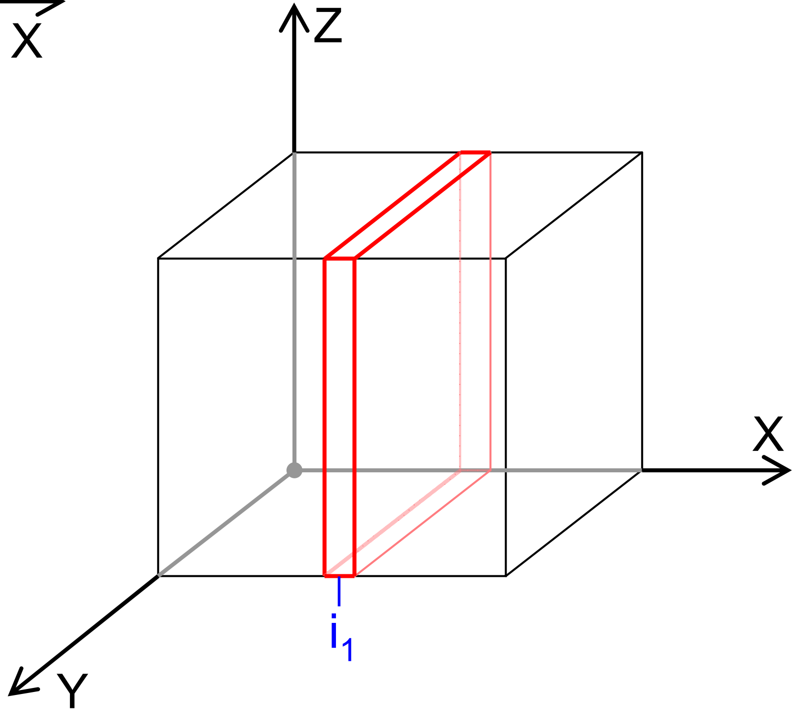
1-dimensional



2-dimensional



3-dimensional



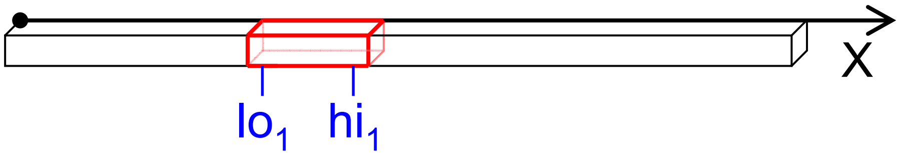
Array Range Selection

e.g. along the first dimension

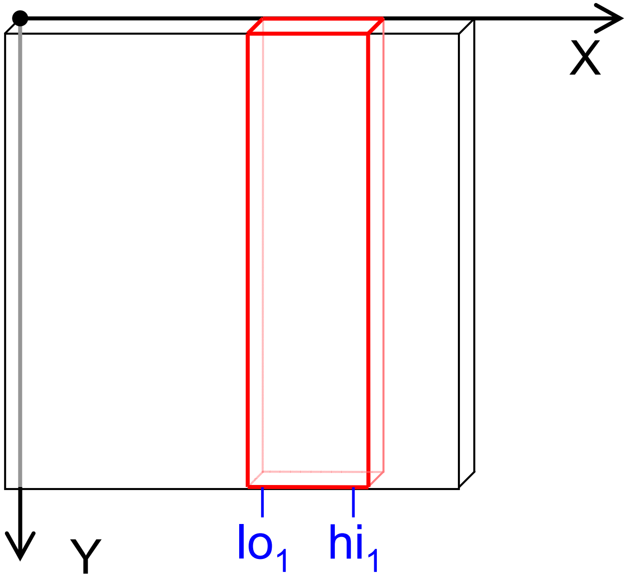
$A[lo_1:hi_1]$

$1 \le lo_1 \le hi_1 \le N_1$

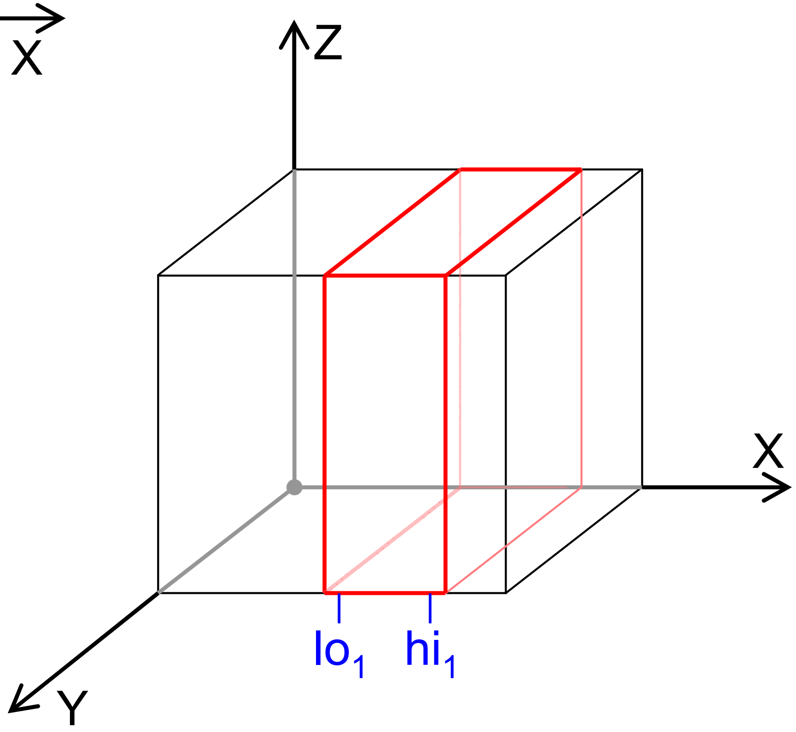
1-dimensional



2-dimensional



3-dimensional



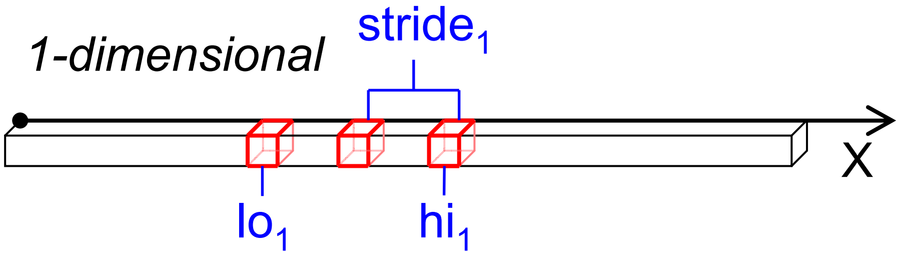
Array Range Selection (with a stride)

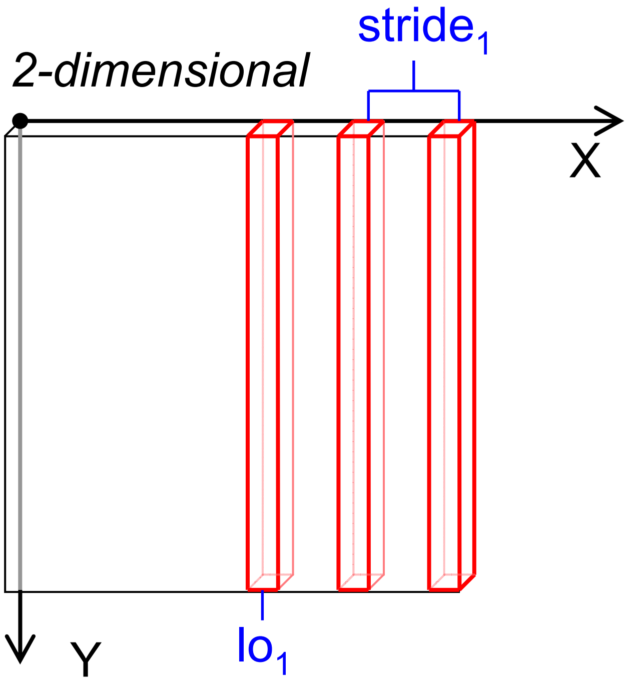
e.g. along the first dimension

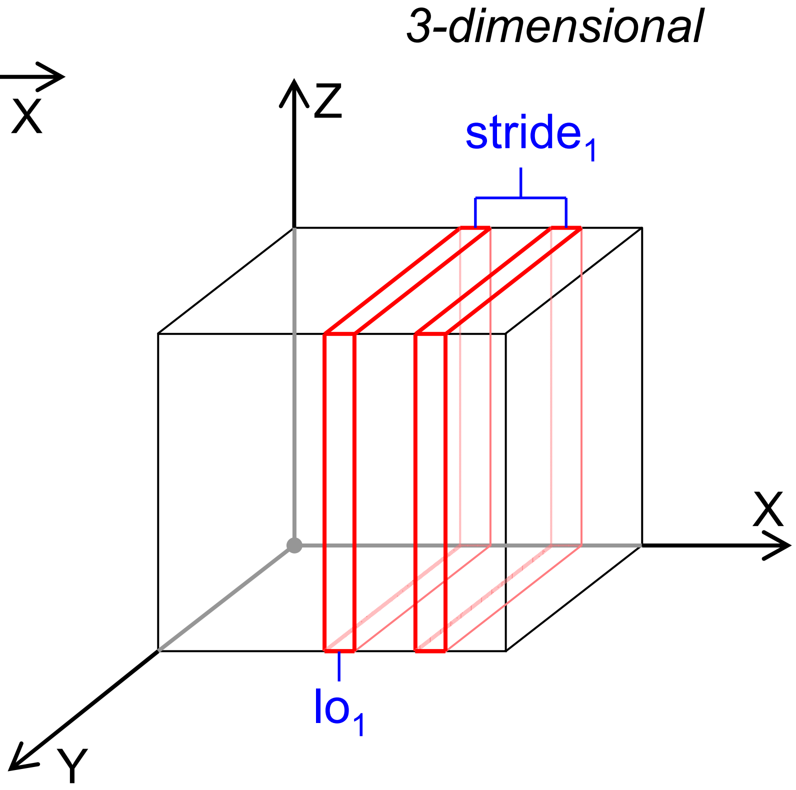
$A[lo_1{:}stride_1{:}hi_1]$

$1 \le lo_1 \le hi_1 \le N_1$

$stride_1 \ge 1$







UPPSALA UNIVERSITET

Scientific SPARQL

A strict superset of W3C SPARQL 1.1 Standard

+

- Array operations

range selection projection

```
SELECT (?A[?lo:?stride:, ?i])
        AS ?result)
 WHERE ...
```

Scientific SPARQL

A strict superset of W3C SPARQL 1.1 Standard

+

- Array operations

bound to all valid row subscripts

```
SELECT ?i, (?A[?i] AS ?result)
 WHERE { ...
         FILTER (mod(?i, 2) = 1) }
```

# return every odd row in ?A

```
SELECT ?i, (?A[?i,?i] AS ?result)
 WHERE ...
```

# return diagonal elements of ?A

Scientific SPARQL

A strict superset of W3C SPARQL 1.1 Standard

+

- Array operations

extended to operate on arrays

```
SELECT (abs(?A-?B) AS ?result)
 WHERE { [] :a ?A ;
            :b ?B }
```

# absolute difference between :a and :b properties (element-wise if arrays)

Scientific SPARQL

A strict superset of W3C SPARQL 1.1 Standard

+

- Array operations
- Functional views (Parameterized queries)

```
DEFINE FUNCTION sse(?x)
 AS SELECT (array_sum(sqr(?A-?B))
            AS ?result)
 WHERE { ?x :a ?A ;
            :b ?B }
# get sum-of-squared-error between :a and :b properties of ?x
```

Scientific SPARQL

A strict superset of W3C SPARQL 1.1 Standard

+

- Array operations
- Functional views (Parameterized queries)
- Second-order functions and closures

```
SELECT (ARGMIN(sse(*)) AS ?x)
# get the node ?x with minimal SSE

SELECT (ARGMIN(param_sse(*, 1.75))
        AS ?result)
# get the node ?x with minimal parameterized SSE
```

Scientific SPARQL

A strict superset of W3C SPARQL 1.1 Standard

+

- Array operations
- Functional views (Parameterized queries)
- Second-order functions and closures
  - array mappers

```
SELECT (MAP(xsd:integer, f(*, *), ?A, ?B)
        AS ?result)
 WHERE { ?x :a ?A ;
            :b ?B }
# apply f(x,y) to the corresponding pairs
  of ?A and ?B elements
```

Scientific SPARQL

A strict superset of W3C SPARQL 1.1 Standard

+

- Array operations
- Functional views (Parameterized queries)
- Second-order functions and closures
- Foreign functions

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

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

Scientific SPARQL

A strict superset of W3C SPARQL 1.1 Standard

+

- Array operations
- Functional views (Parameterized queries)
- Second-order functions and closures
- Foreign functions
  - multi-directionality and cost models

```
DEFINE FUNCTION sqroot(?x) AS
   FOR 'bf' JAVA 'MyLib/sqroot' COST 4 FANOUT 2
   FOR 'fb' JAVA 'MyLib/square' COST 1 FANOUT 1
```

Scientific SPARQL

A strict superset of W3C SPARQL 1.1 Standard

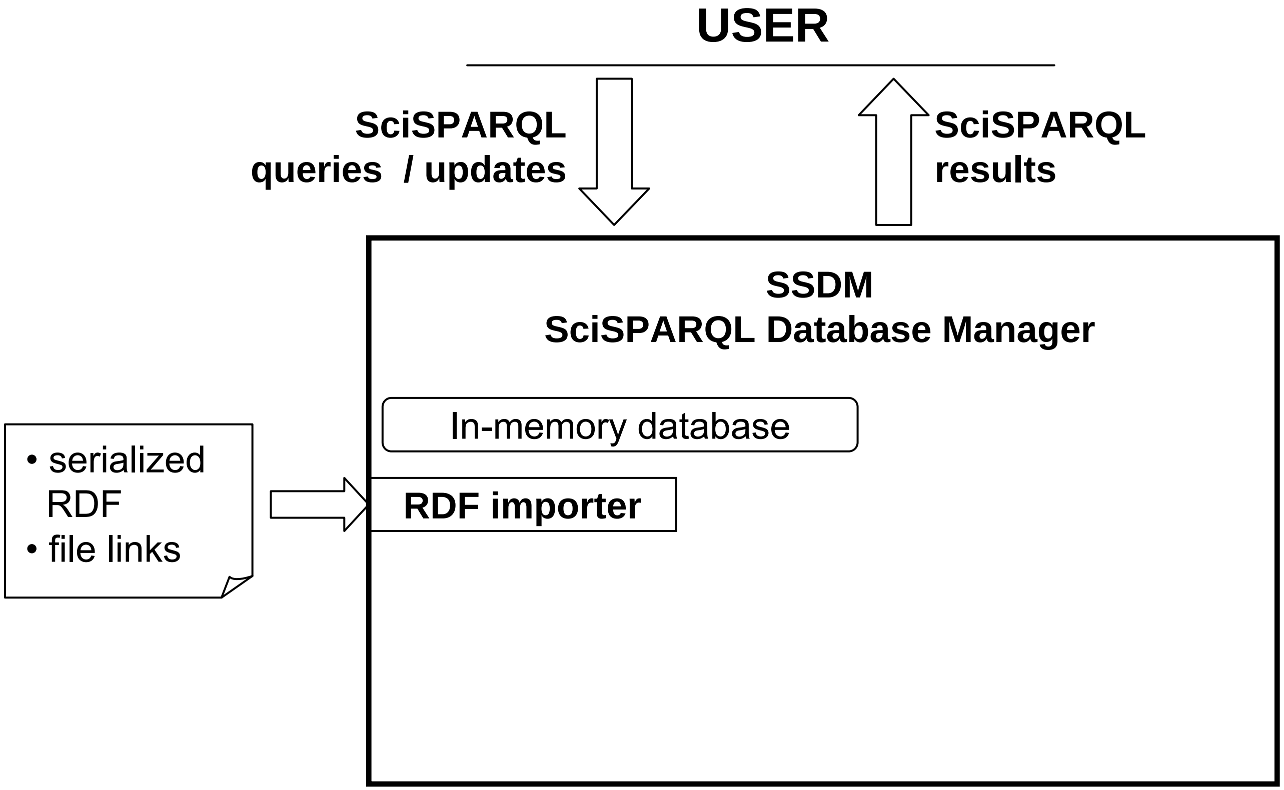
+

- Array operations
- Functional views (Parameterized queries)
- Second-order functions and closures
- Foreign functions
  - multi-directionality and cost models

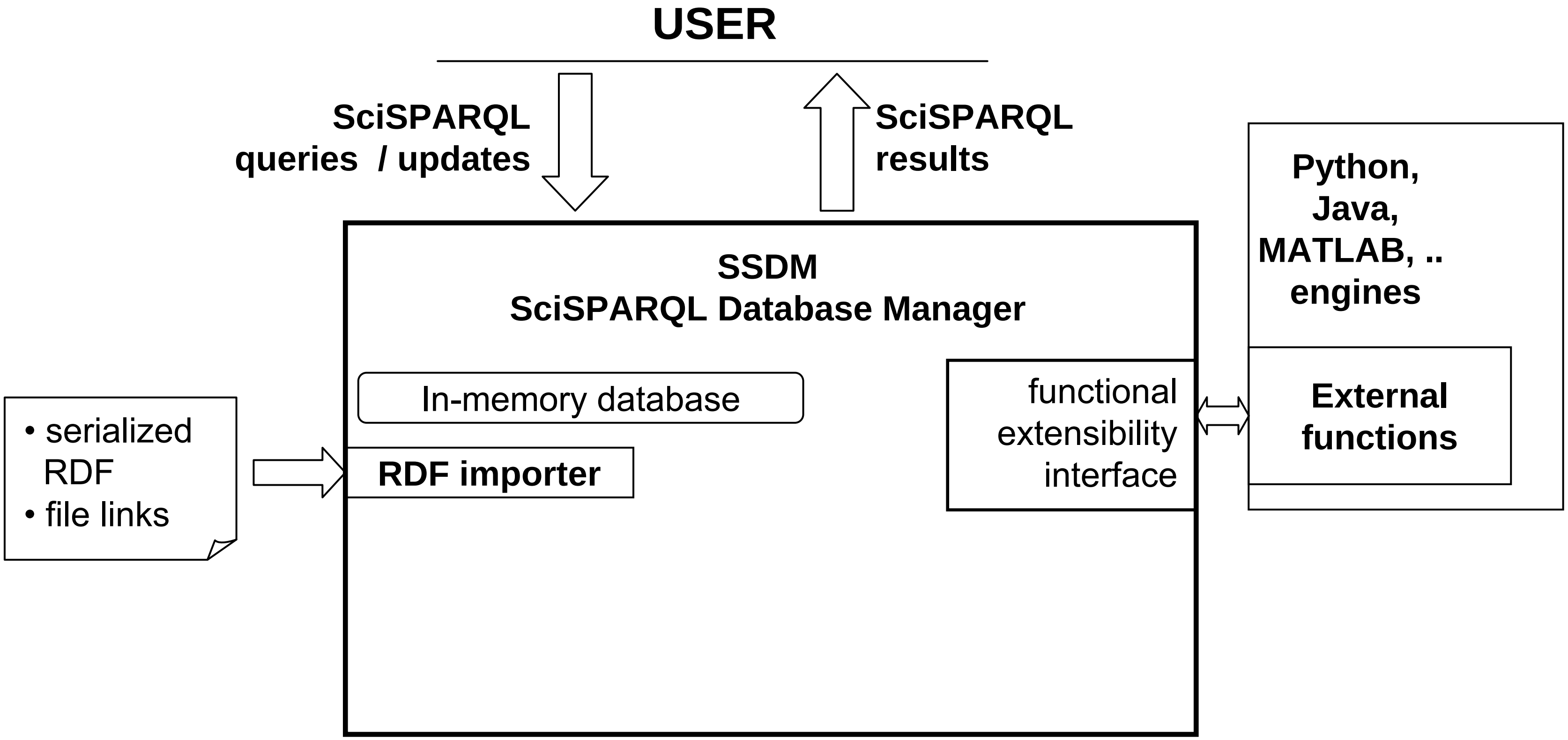
```
DEFINE FUNCTION sqroot(?x) AS
   FOR 'bf' JAVA 'MyLib/sqroot' COST 4 FANOUT 2
   FOR 'fb' JAVA 'MyLib/square' COST 1 FANOUT 1
```

- Introduction
- RDF & SPARQL
- RDF with Arrays & Scientific SPARQL
- Scientific SPARQL Database Manager
  - architecture
  - SciSPARQL query processing
  - internal array storage
- Extensible Array Storage
- Array Query Benchmark
- Summary

SSDM Architecture

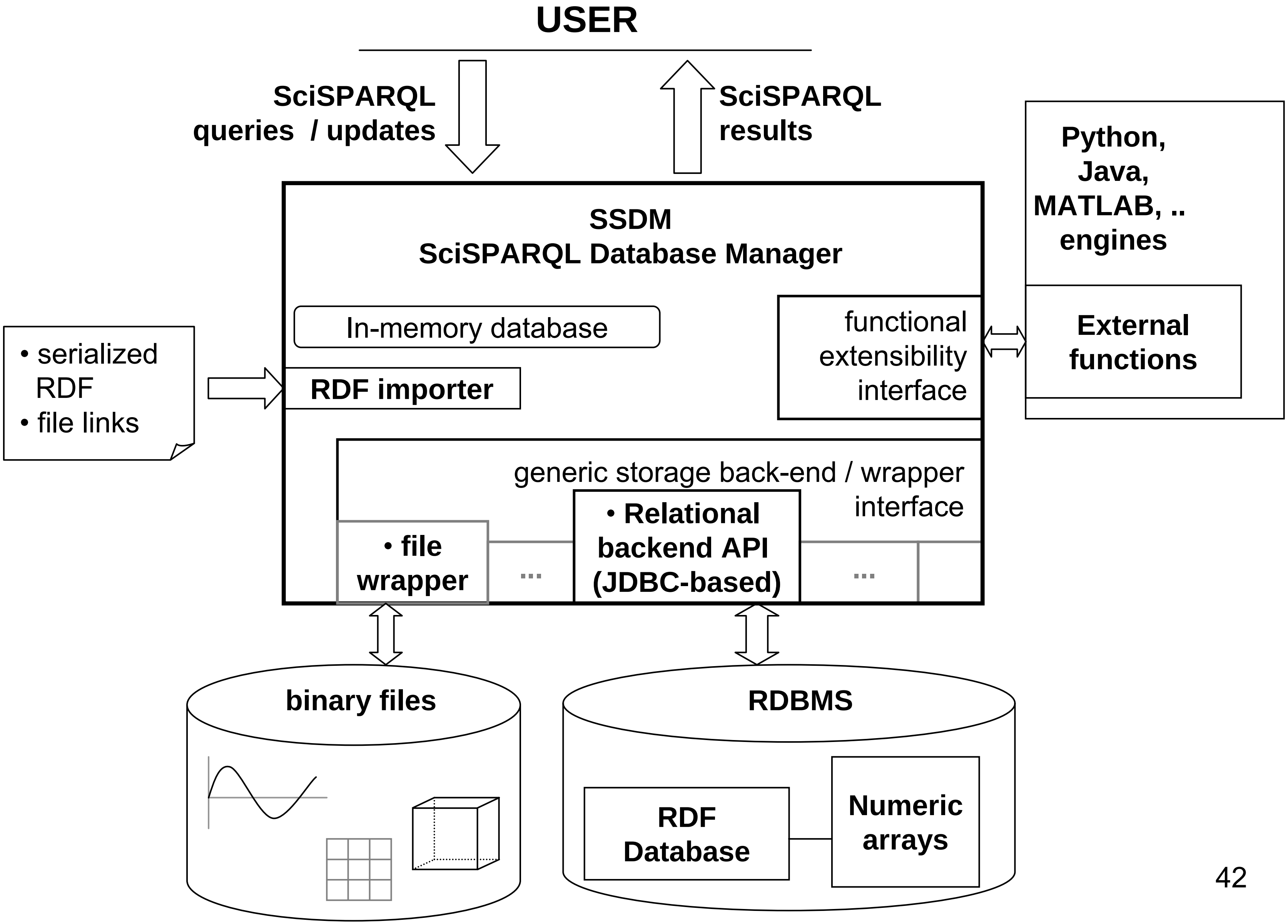


SSDM Architecture

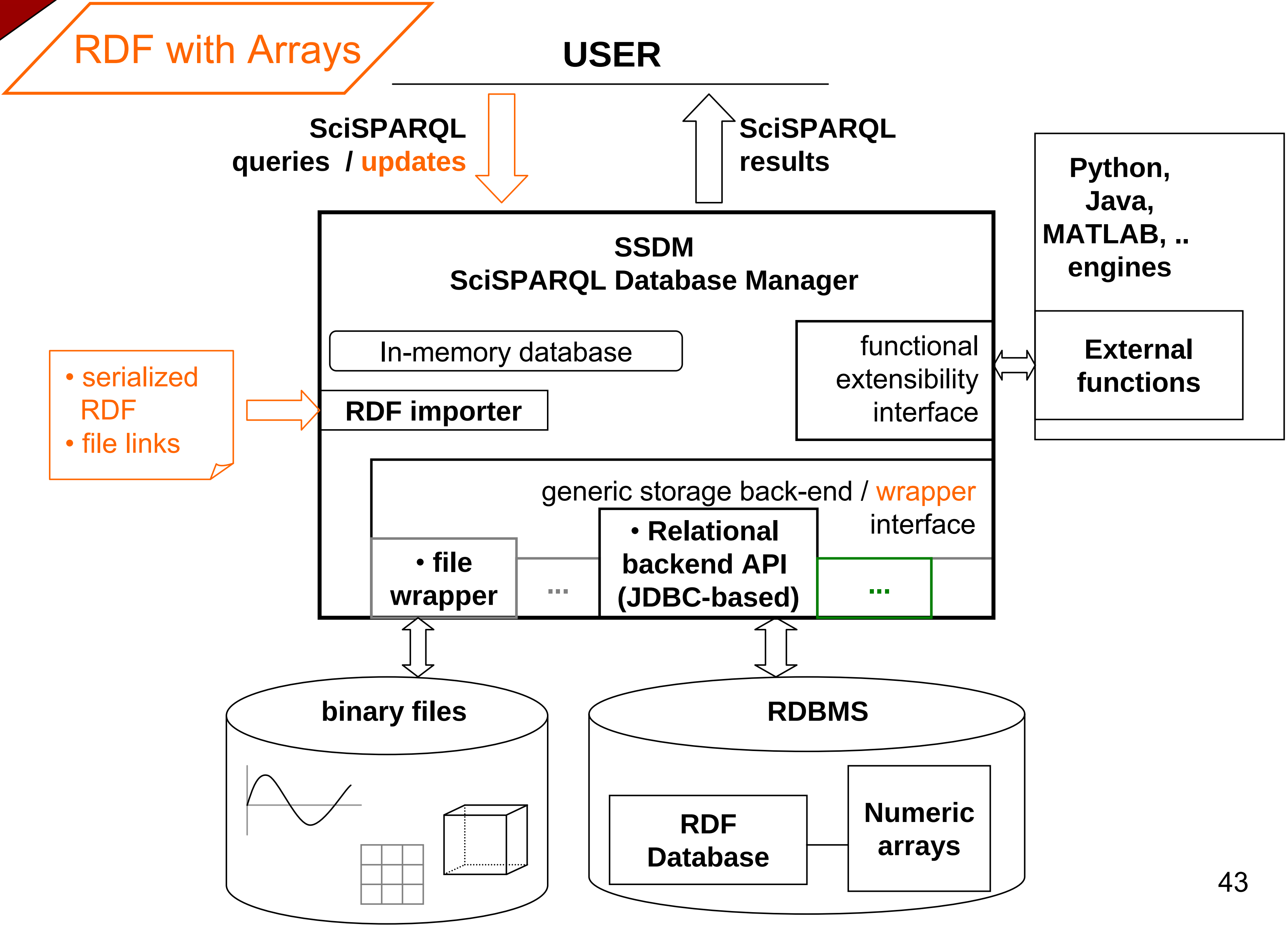


USER
SciSPARQL queries / updates
SciSPARQL results
SSDM
SciSPARQL Database Manager
In-memory database
RDF importer
functional extensibility interface
• serialized RDF
• file links
Python, Java, MATLAB, .. engines
External functions

SSDM Architecture


USER
SciSPARQL queries / updates
SciSPARQL results
SSDM
SciSPARQL Database Manager
In-memory database
RDF importer
functional extensibility interface
Python, Java, MATLAB, .. engines
External functions
• serialized RDF
• file links
generic storage back-end / wrapper interface
• file wrapper
...
• Relational backend API (JDBC-based)
...
binary files
RDBMS
RDF Database
Numeric arrays

RDF with Arrays datasets



RDF with Arrays

USER

SciSPARQL queries / updates

SciSPARQL results

- serialized RDF
- file links

SSDM
SciSPARQL Database Manager

In-memory database

RDF importer

functional extensibility interface

Python, Java, MATLAB, .. engines

External functions

generic storage back-end / wrapper interface

- file wrapper

...

- Relational backend API (JDBC-based)

...

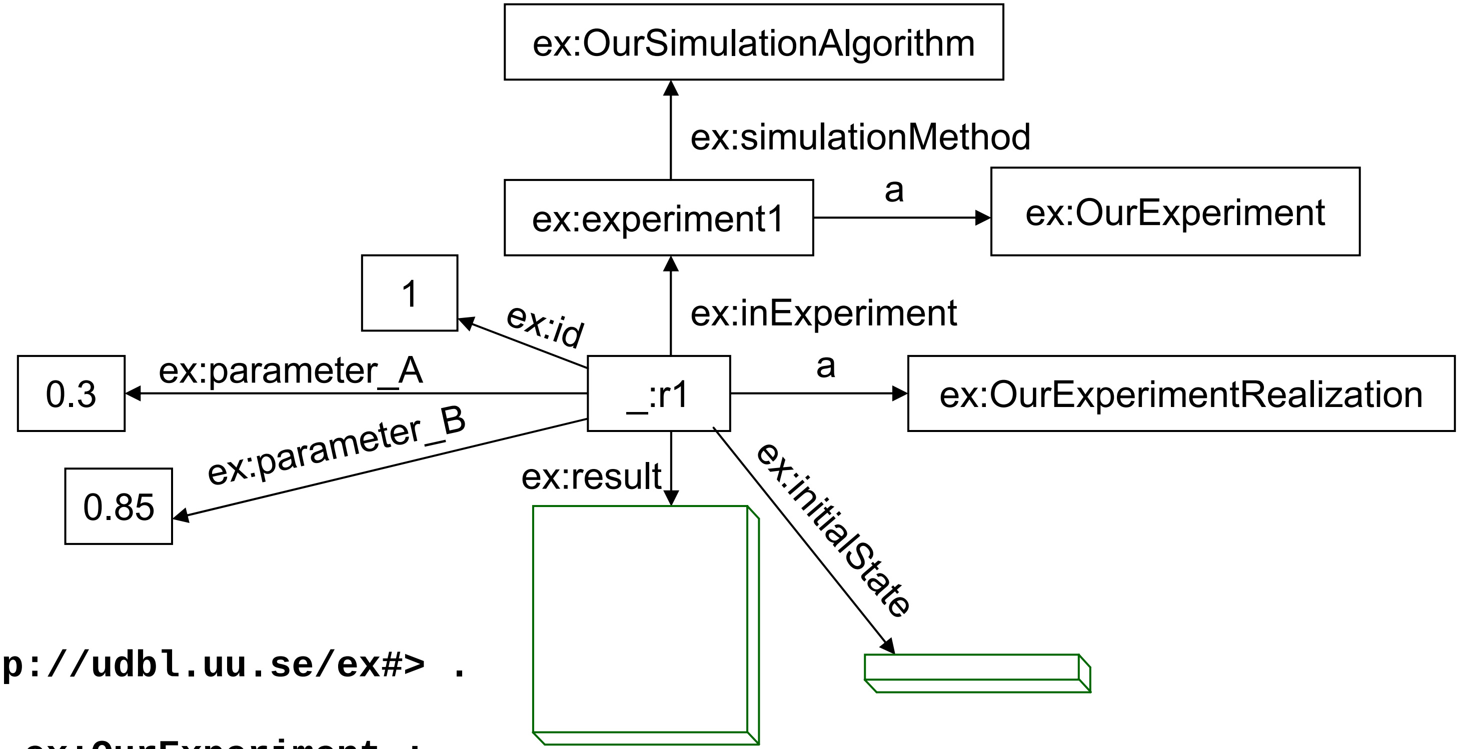
binary files

RDBMS

RDF Database

Numeric arrays

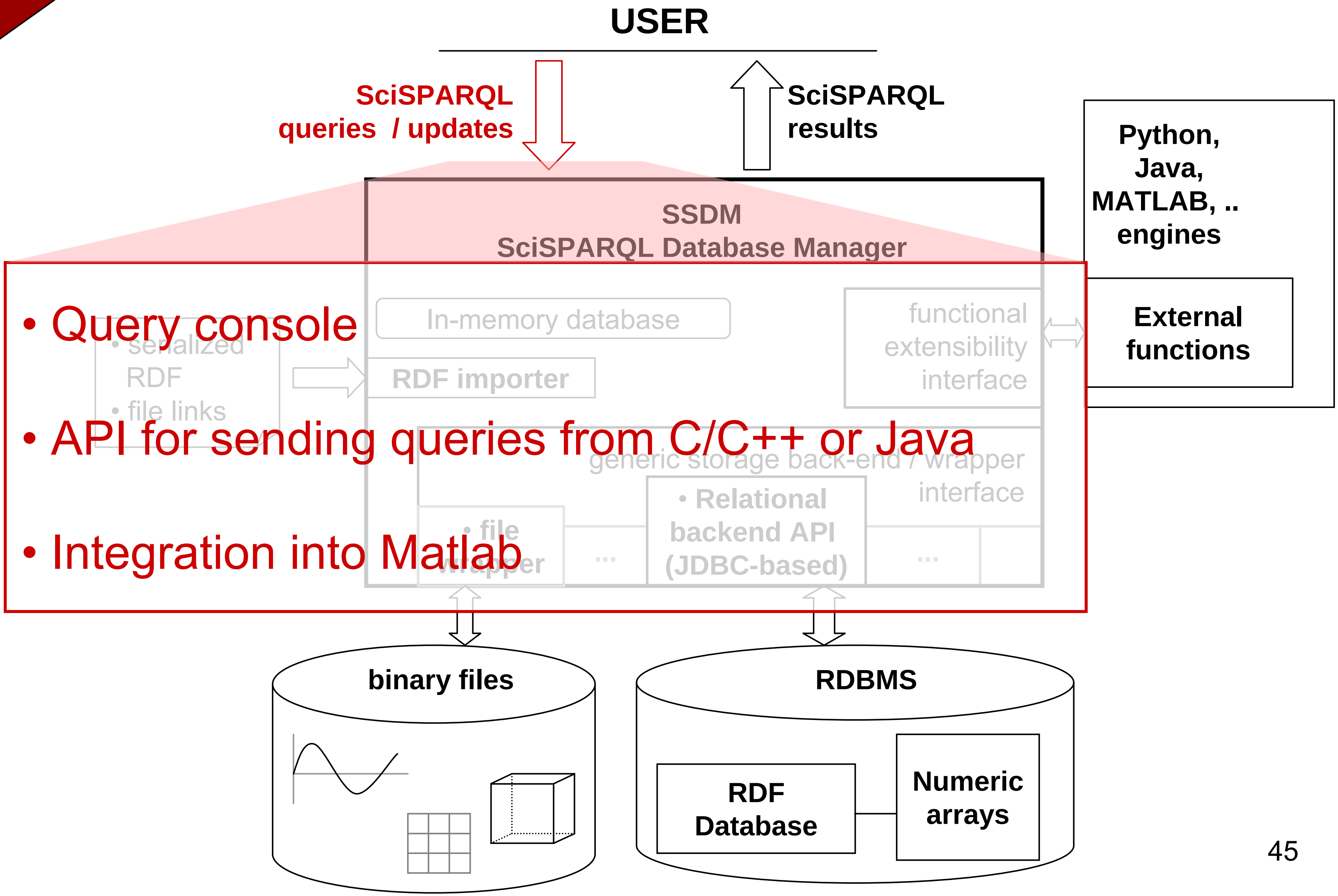
RDF with Arrays datasets



```
@prefix ex: <http://udbl.uu.se/ex#> .

ex:experiment1 a ex:OurExperiment ;
               ex:simulationMethod ex:OurSimulationAlgorithm .
_:r1 a ex:OurExperimentRealization ;
     ex:inExperiment ex:experiment1 ;
     ex:id 1 ;
     ex:initialState (0 0.5 1 1 1 1 0.5 0) ; #arrays value serialized as RDF
     ex:iterations 1000 ;
     ex:parameter_A 0.3 ;
     ex:parameter_B 0.85 ;
     ex:result <file://realization_1.mat#Res> . #array values as file links
```

SciSPARQL Query Interfaces

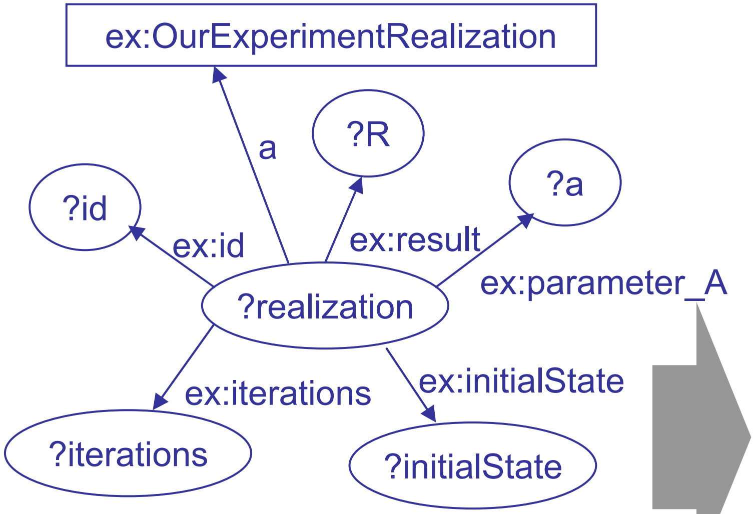

USER
SciSPARQL queries / updates
SciSPARQL results
SSDM
SciSPARQL Database Manager
In-memory database
functional extensibility interface
• serialized RDF
• file links
RDF importer
generic storage back-end / wrapper interface
• file wrapper
...
• Relational backend API (JDBC-based)
...
Python, Java, MATLAB, .. engines
External functions
binary files
RDBMS
RDF Database
Numeric arrays

- Query console
- API for sending queries from C/C++ or Java
- Integration into Matlab

- Introduction
- RDF & SPARQL
- RDF with Arrays & Scientific SPARQL
- Scientific SPARQL Database Manager
  - architecture
  - SciSPARQL query processing
  - internal array storage
- Extensible Array Storage
- Array Query Benchmark
- Summary

SciSPARQL Query Semantics

```
PREFIX ex: <http://udbl.uu.se/ex#>
SELECT ?id (array_avg(?R[:, ?iterations]) AS ?res)
 WHERE { ?realization a ex:OurExperimentRealization ;
                      ex:id ?id ;
                      ex:result ?R ;
                      ex:iterations ?iterations ;
                      ex:parameter_A ?a ;
                      ex:initialState ?initialState
         FILTER ( array_max(?initialState) < 0.75
                  && ?a >= 0.25 ) }
```



array_max(?InitialState) < 0.75
&& ?a ≥ 0.25

array_avg(?R[:, ?iterations])

Match pattern **Filter solutions** **Postprocess**

SciSPARQL Query Processing

- **AmosQL representation**

```
PREFIX ex: <http://udbl.uu.se/ex#>
SELECT ?id (array_avg(?R[:, ?iterations]) AS ?res)
 WHERE { ?realization a ex:OurExperimentRealization ;
                      ex:id ?id ;
                      ex:result ?R ;
                      ex:iterations ?iterations ;
                      ex:parameter_A ?a ;
                      ex:initialState ?initialState
         FILTER ( array_max(?initialState) < 0.75
                 && ?a >= 0.25 ) }
```

```
select id, rdf:array_avg(aref(R,1,rdf:minus(iterations,1)))
  from Literal realization, Literal a, Literal initialState,
       Literal iterations, Literal R, Literal id
 where (realization, URI('http://www.w3.org/1999/02/22-rdf-syntax-ns#type'),
        URI('http://udbl.uu.se/ex#OurExperimentRealization')) in GRAPH(0)
   and (realization, URI('http://udbl.uu.se/ex#id'), id) in GRAPH(0)
   and (realization, URI('http://udbl.uu.se/ex#result'), R) in GRAPH(0)
   and (realization, URI('http://udbl.uu.se/ex#iterations'), iterations)
       in GRAPH(0)
   and (realization, URI('http://udbl.uu.se/ex#parameter_A'), a) in GRAPH(0)
   and (realization, URI('http://udbl.uu.se/ex#initialState'), initialState)
       in GRAPH(0)
   and rdf:array_max(initialState)<0.75 and a>=0.25;
```

SciSPARQL Query Processing

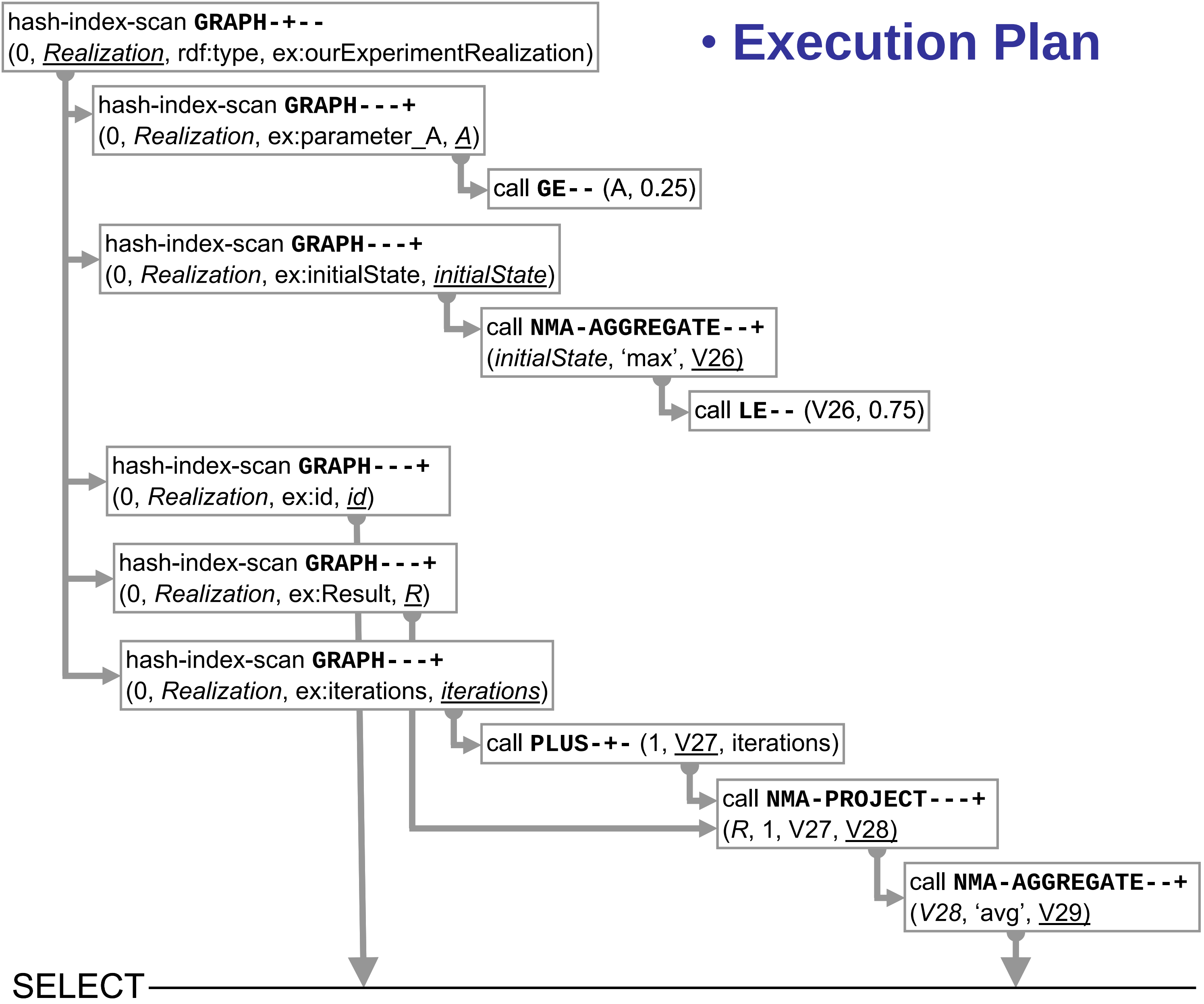
- **ObjectLog representation**

```
PREFIX ex: <http://udbl.uu.se/ex#>
SELECT ?id (array_avg(?R[:, ?iterations]) AS ?res)
 WHERE { ?realization a ex:OurExperimentRealization ;
                      ex:id ?id ;
                      ex:result ?R ;
                      ex:iterations ?iterations ;
                      ex:parameter_A ?a ;
                      ex:initialState ?initialState
         FILTER ( array_max(?initialState) < 0.75
                  && ?a >= 0.25 ) }
```

```
(*SELECT* ID+ _V29+) <-
(AND (GRAPH 0 REALIZATION
            #[URI "http://www.w3.org/1999/02/22-rdf-syntax-ns#type"]
            #[URI "http://udbl.uu.se/ex#OurExperimentRealization"])
     (GRAPH 0 REALIZATION #[URI "http://udbl.uu.se/ex#id"] ID)
     (GRAPH 0 REALIZATION #[URI "http://udbl.uu.se/ex#result"] R)
     (GRAPH 0 REALIZATION #[URI "http://udbl.uu.se/ex#iterations"] ITERATIONS)
     (GRAPH 0 REALIZATION #[URI "http://udbl.uu.se/ex#parameter_A"] A)
     (GRAPH 0 REALIZATION #[URI "http://udbl.uu.se/ex#initialState"] INITIALSTATE)
     (RDF:ARRAY_MAX INITIALSTATE _V26)
     (< _V26 0.75)
     (>= A 0.25)
     (RDF:MINUS ITERATIONS 1 _V27)
     (AREF R 1 _V27 _V28)
     (RDF:ARRAY_AVG _V28 _V29))
```

SciSPARQL Query Processing

- **Execution Plan**



hash-index-scan **GRAPH-+--**
(0, *Realization*, rdf:type, ex:ourExperimentRealization)

hash-index-scan **GRAPH---+**
(0, *Realization*, ex:parameter_A, *A*)

call **GE--** (A, 0.25)

hash-index-scan **GRAPH---+**
(0, *Realization*, ex:initialState, *initialState*)

call **NMA-AGGREGATE--+**
(*initialState*, 'max', V26)

call **LE--** (V26, 0.75)

hash-index-scan **GRAPH---+**
(0, *Realization*, ex:id, *id*)

hash-index-scan **GRAPH---+**
(0, *Realization*, ex:Result, *R*)

hash-index-scan **GRAPH---+**
(0, *Realization*, ex:iterations, *iterations*)

call **PLUS-+-** (1, V27, iterations)

call **NMA-PROJECT---+**
(*R*, 1, V27, V28)

call **NMA-AGGREGATE--+**
(*V28*, 'avg', V29)

SELECT

- Introduction
- RDF & SPARQL
- RDF with Arrays & Scientific SPARQL
- Scientific SPARQL Database Manager
  - architecture
  - SciSPARQL query processing
  - internal array storage
- Extensible Array Storage
- Array Query Benchmark
- Summary

Internal Array 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**

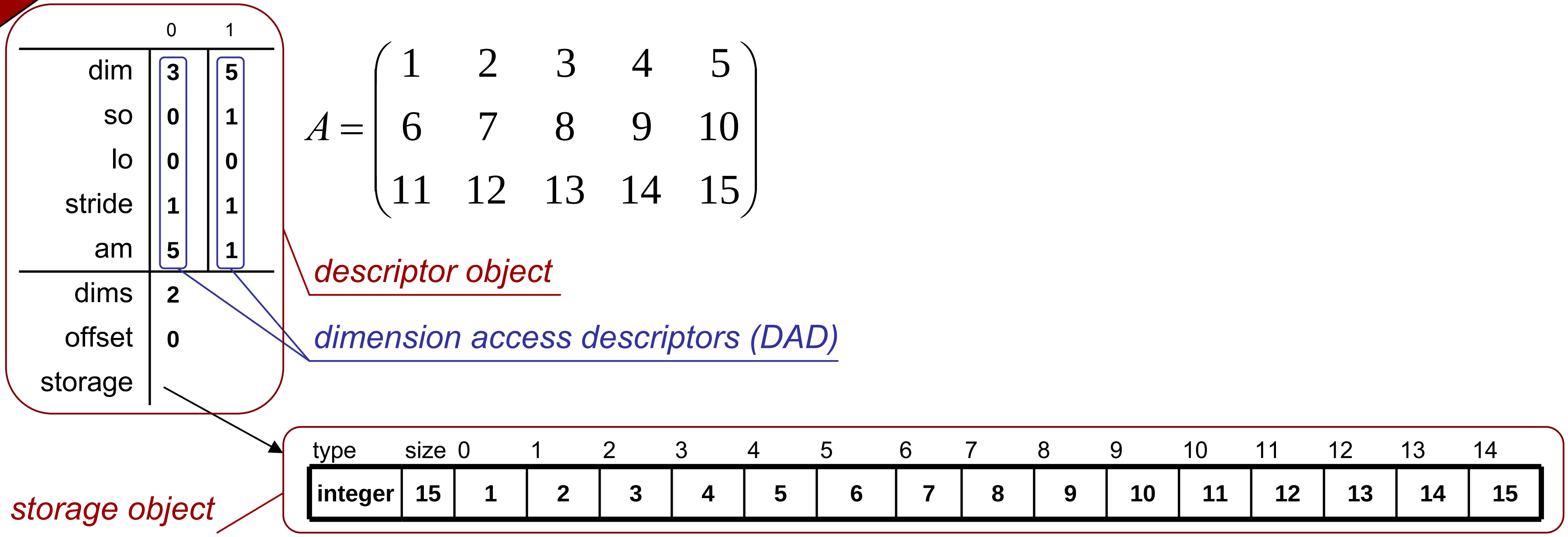
Internal Array Storage

	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**

Internal Array Storage



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

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

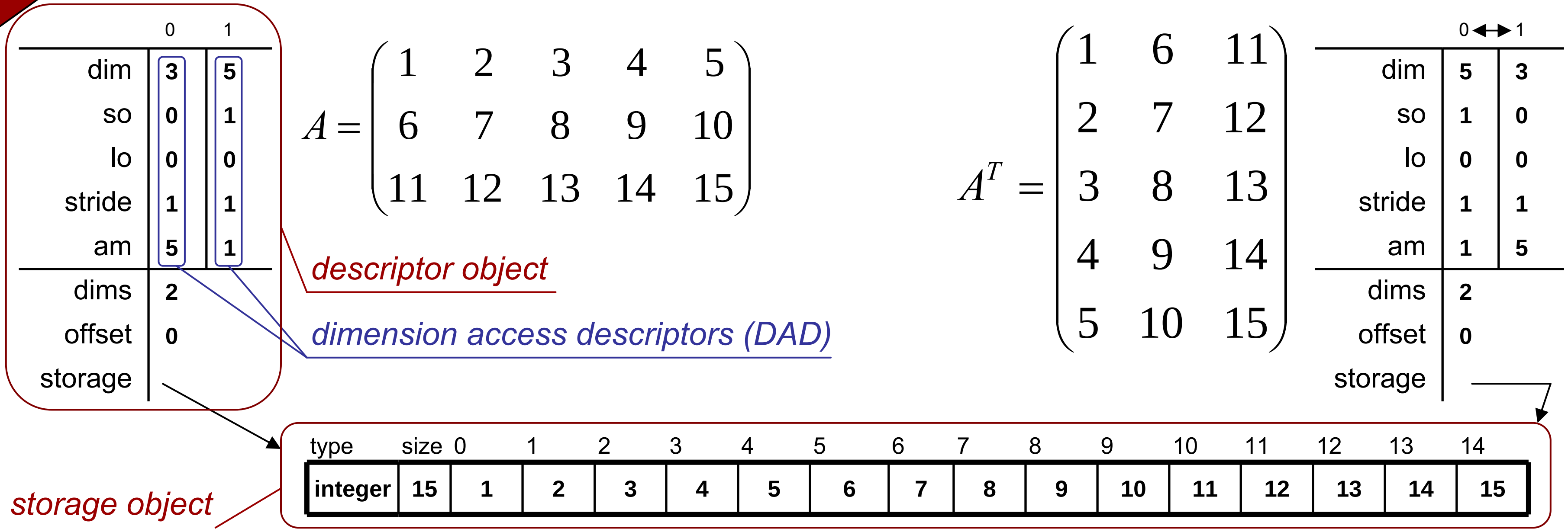
descriptor object

dimension access descriptors (DAD)

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

storage object

Internal Array Storage



	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)

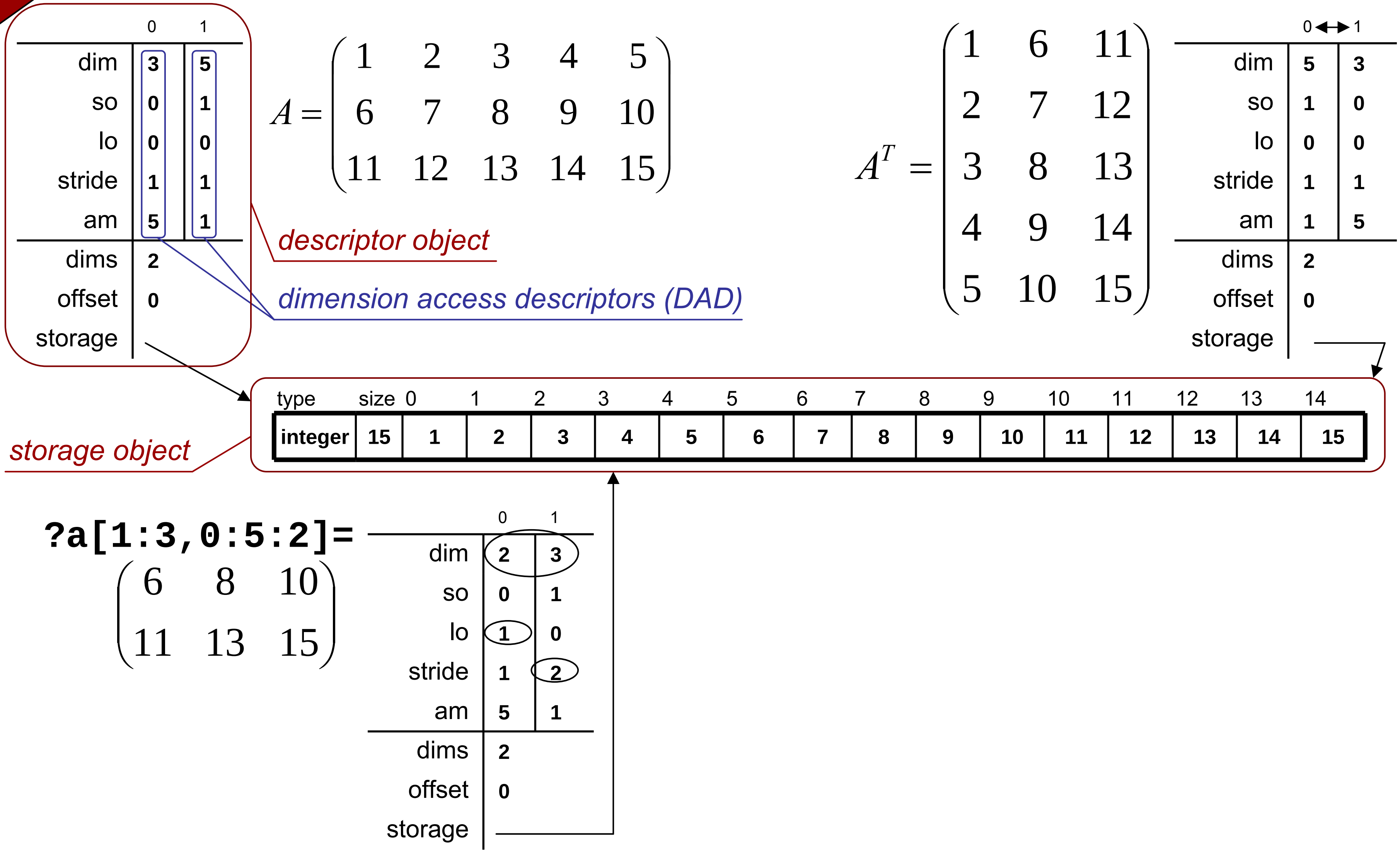
$$A^T = \begin{pmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{pmatrix}$$

	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

Internal Array Storage



	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

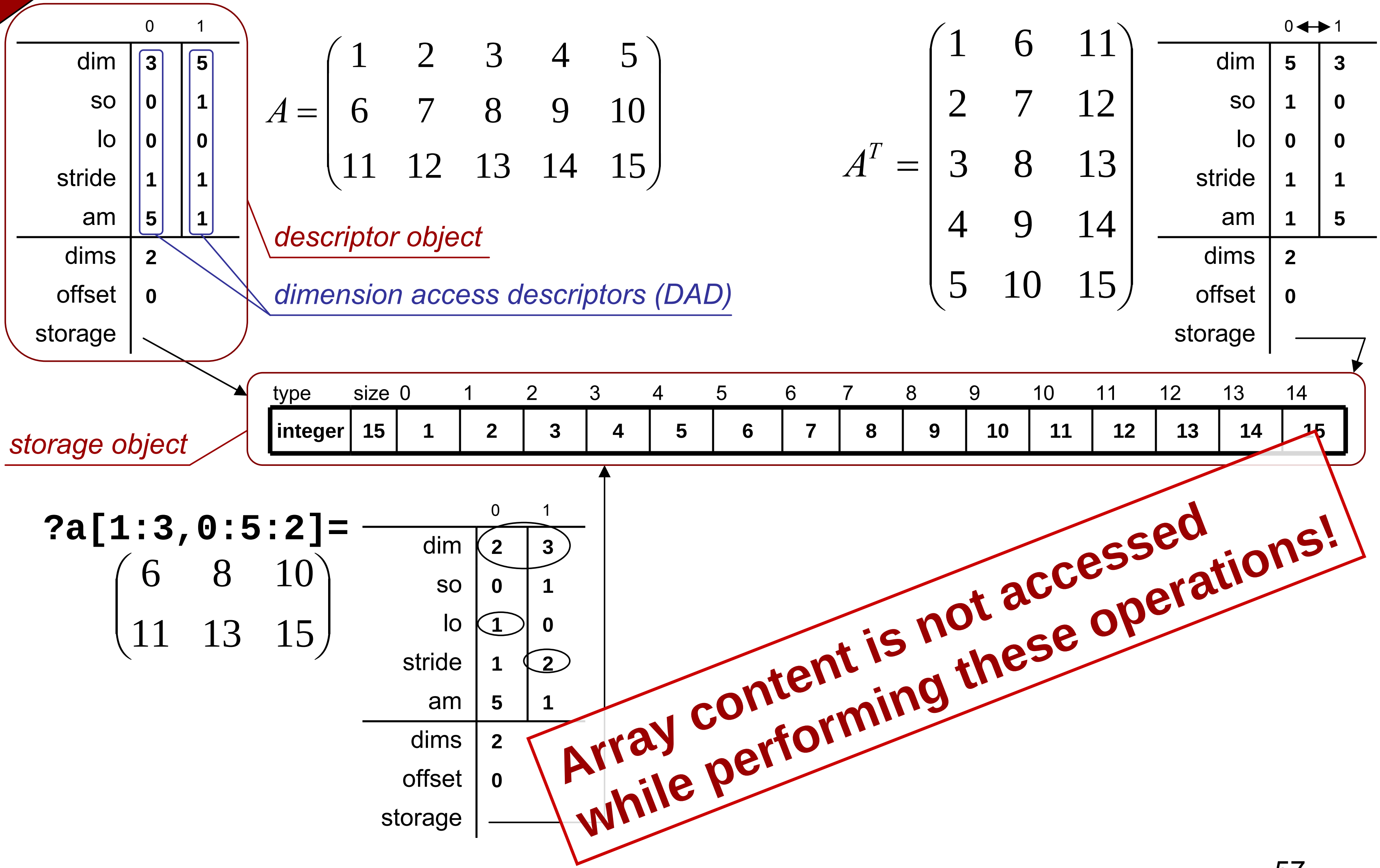
type	size	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
integer	15	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

```
?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		

Internal Array Storage



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

	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)

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

	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**

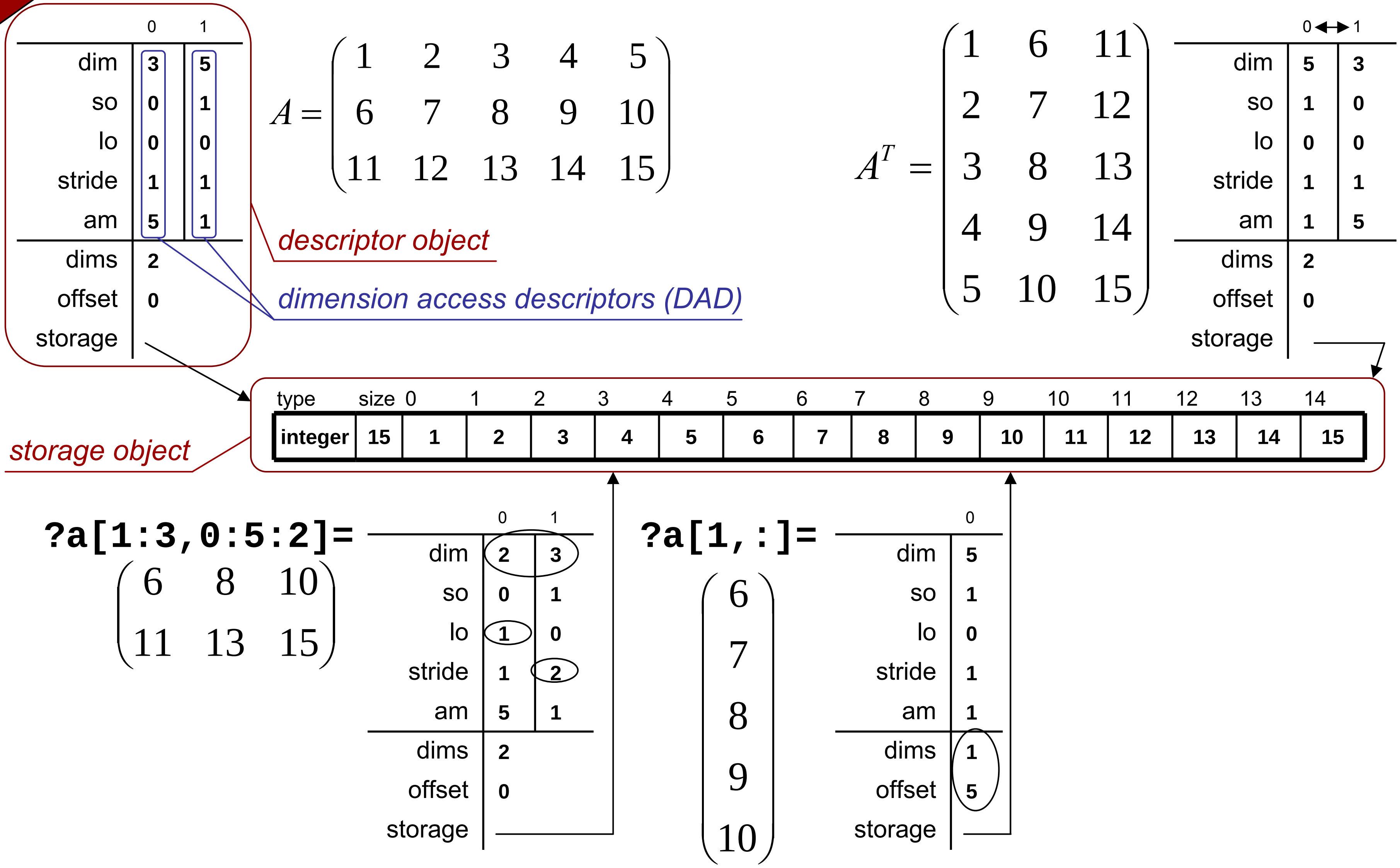
`?a[1:3,0:5:2]=`

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

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

Array content is not accessed while performing these operations!

Internal Array Storage



	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]=
```

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

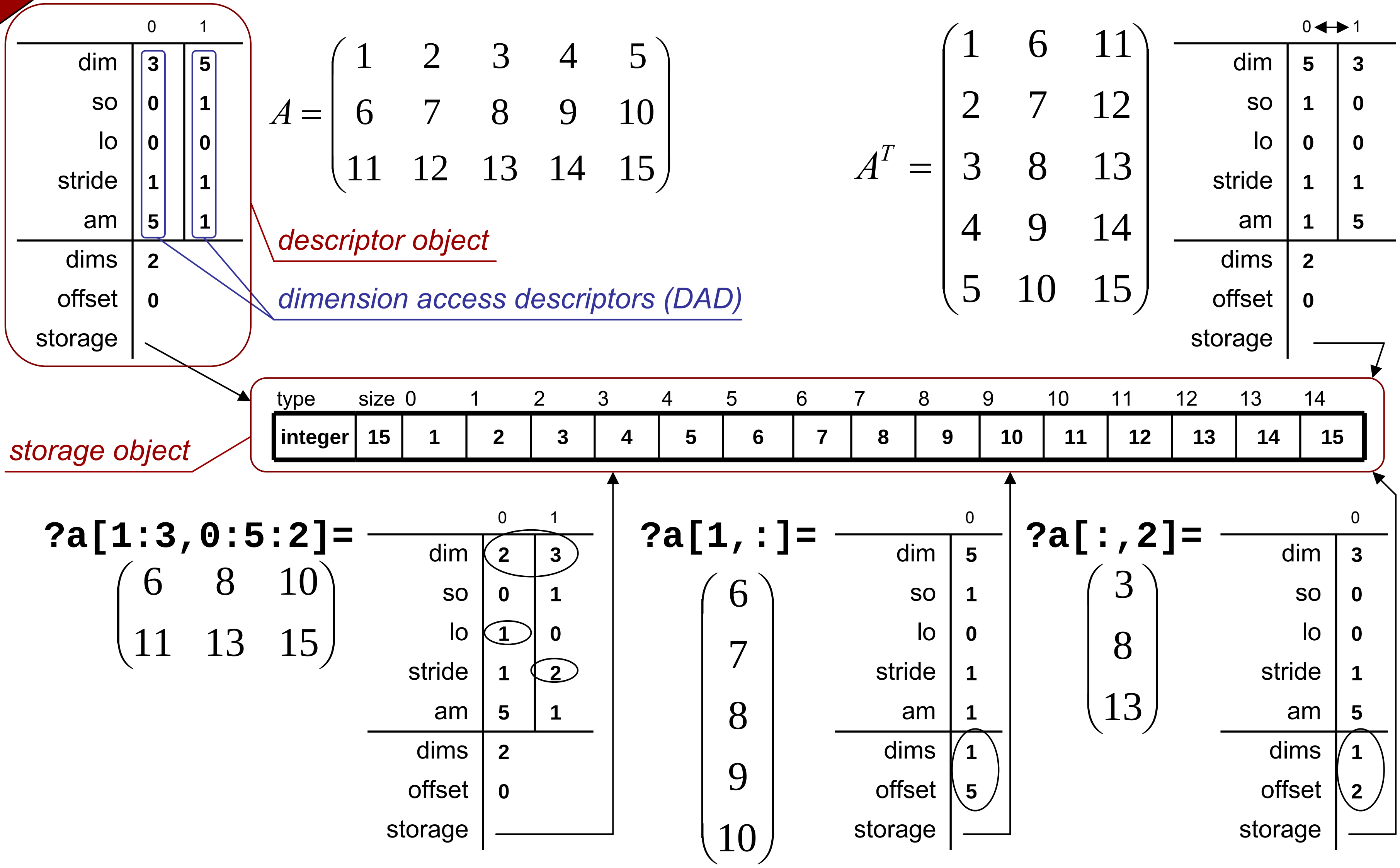
	0	1
dim	2	3
so	0	1
lo	1	0
stride	1	2
am	5	1
dims	2	
offset	0	
storage		

```
?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	

Internal Array Storage



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

	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)

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

	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]=
```

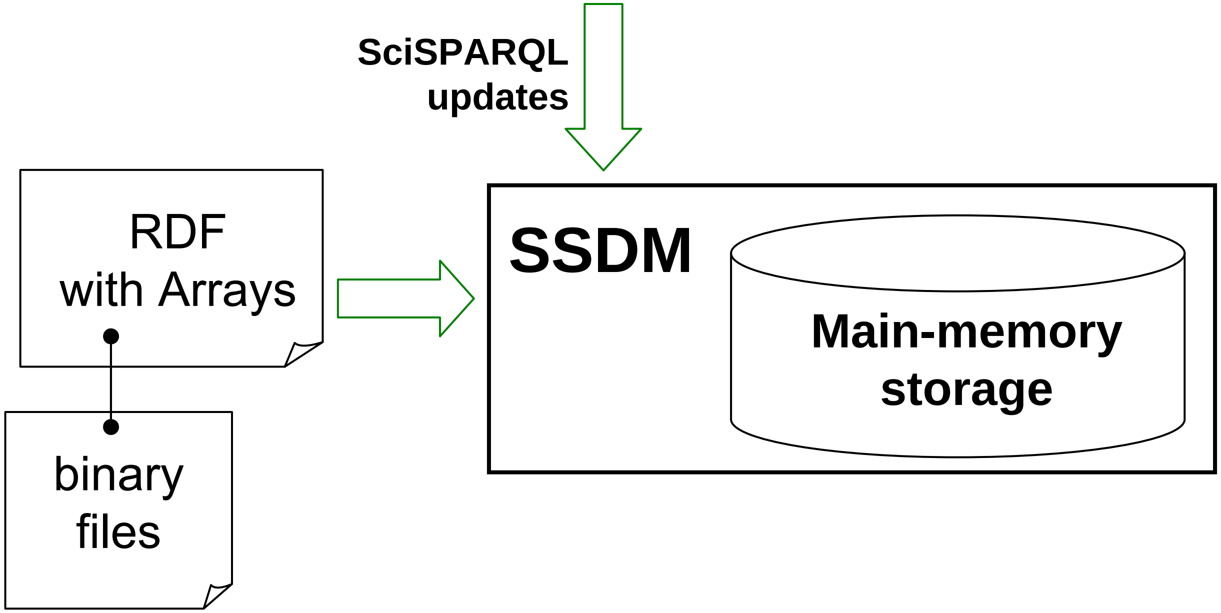
$$\begin{pmatrix} 3 \\ 8 \\ 13 \end{pmatrix}$$

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

- Introduction
- RDF & SPARQL
- RDF with Arrays & Scientific SPARQL
- Scientific SPARQL Database Manager
- Extensible Array Storage
  - usage scenarios / configurations
  - Array Storage Extensibility Interface
  - relational back-end
- Array Query Benchmark
- Summary

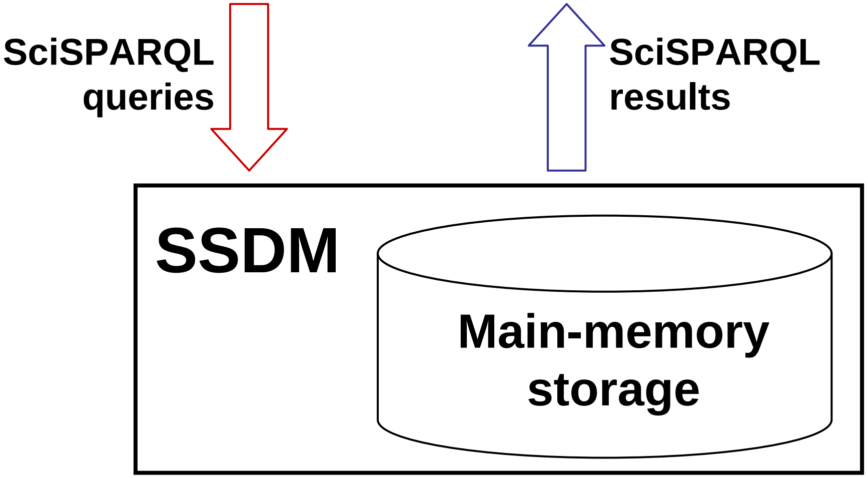
Usage Scenarios / Configurations

- **Main-memory**



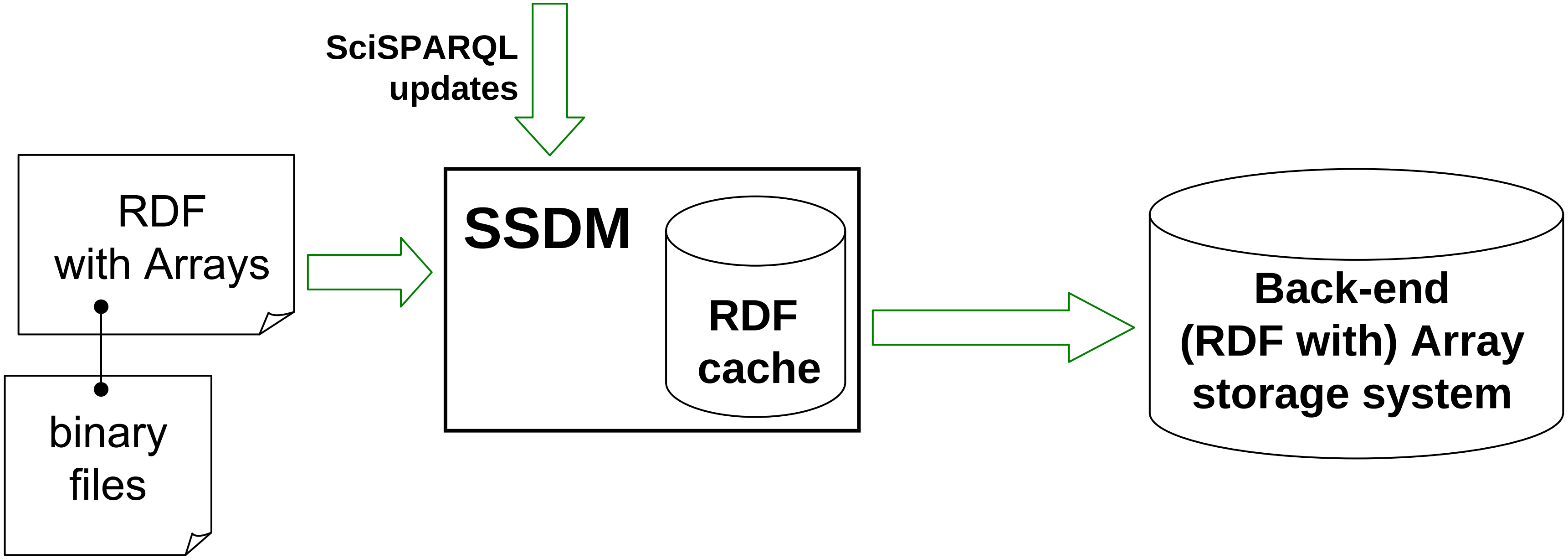
Usage Scenarios / Configurations

- **Main-memory**



Usage Scenarios / Configurations

- **Back-end**

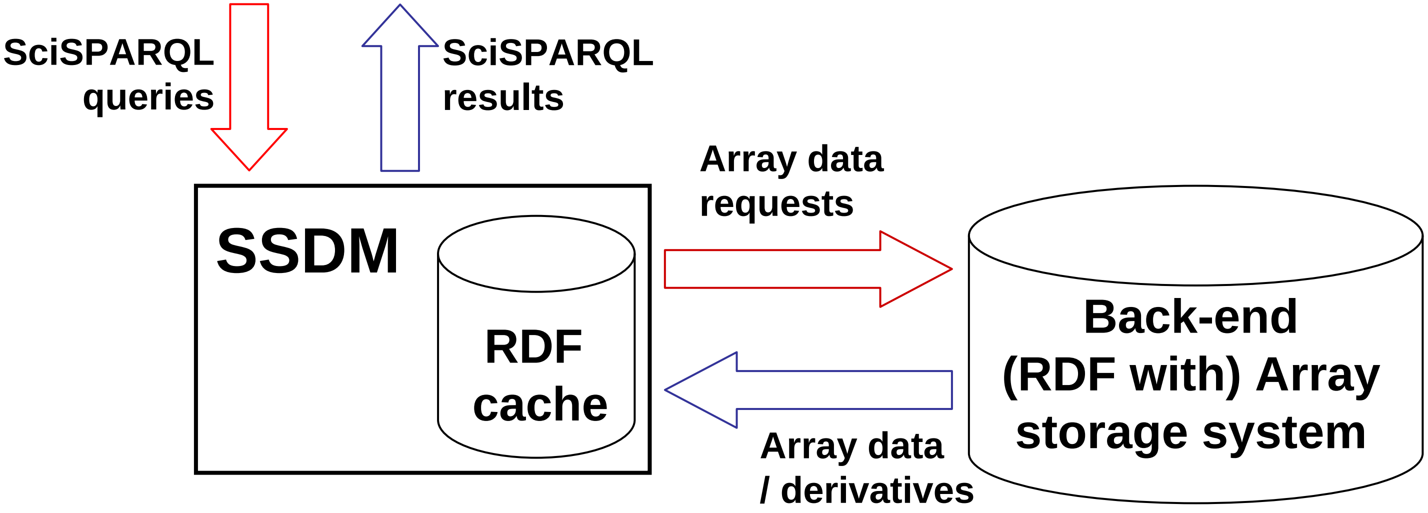


Implemented interfaces:

- **binary files (.mat / HDF5)**
- **RDBMS supporting SQL**
- **RasDaMan**

Usage Scenarios / Configurations

- Back-end

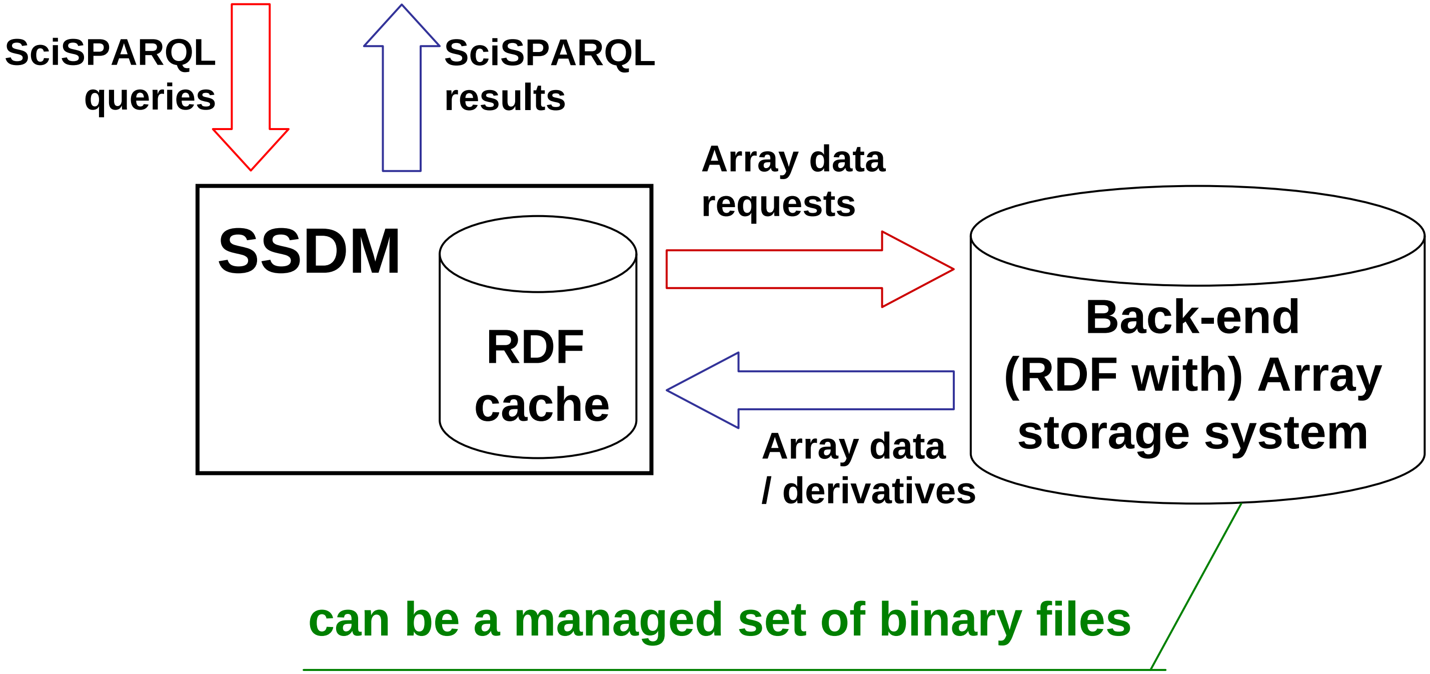


Implemented interfaces:

- binary files (.mat / HDF5)
- RDBMS supporting SQL
- RasDaMan

Usage Scenarios / Configurations

- **Back-end**

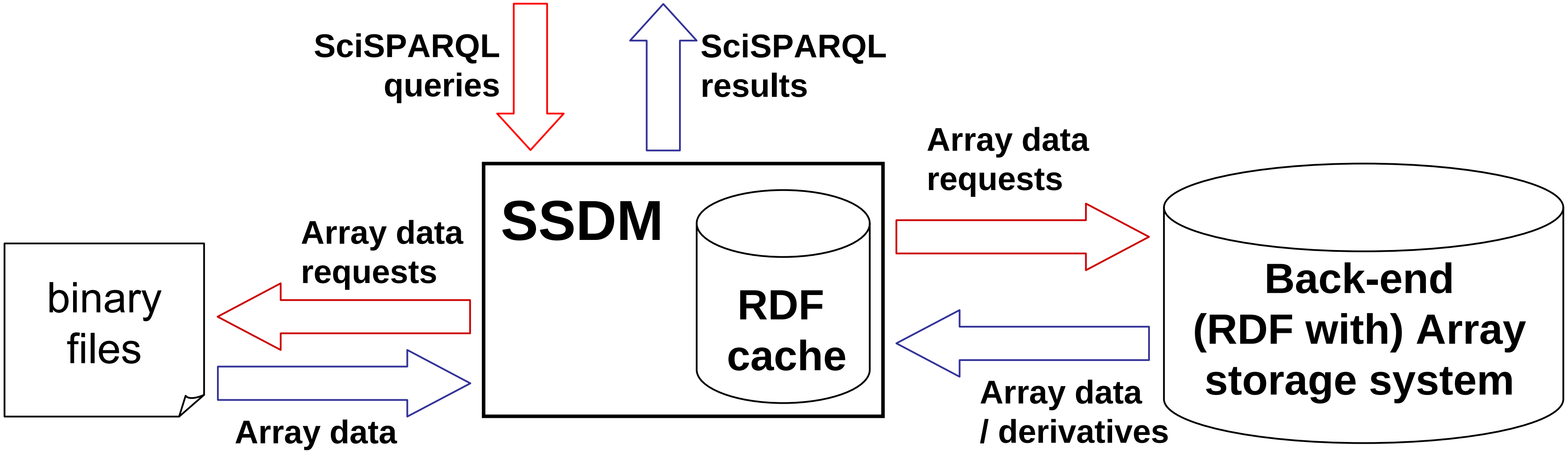


Implemented interfaces:

- **binary files (.mat / HDF5)**
- **RDBMS supporting SQL**
- **RasDaMan**

Usage Scenarios / Configurations

- **Back-end (lazy)**

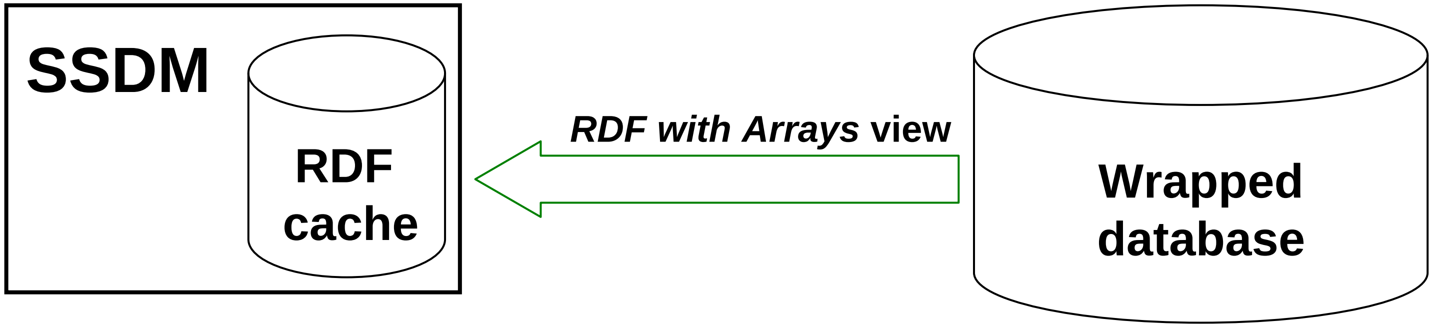


Implemented interfaces:

- **binary files (.mat / HDF5)**

Usage Scenarios / Configurations

- Wrapper

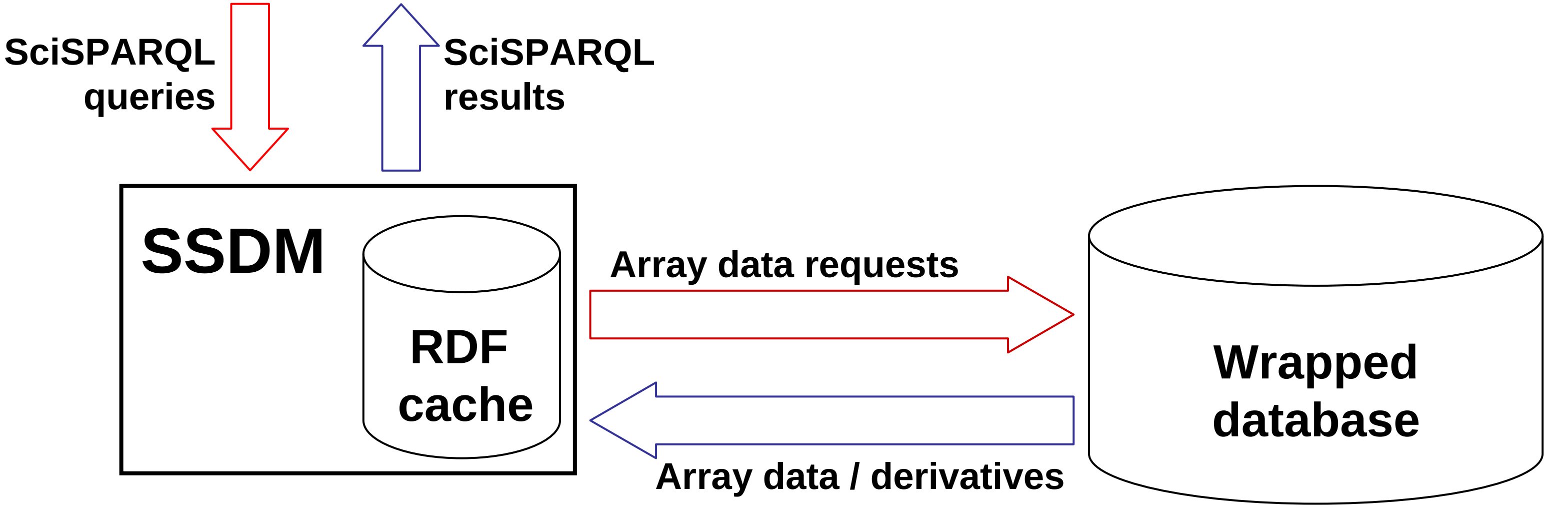


Implemented interfaces:

- Chelonia
- RasDaMan

Usage Scenarios / Configurations

- Wrapper



Implemented interfaces:

- Chelonia
- RasDaMan

- Introduction
- RDF & SPARQL
- RDF with Arrays & Scientific SPARQL
- Scientific SPARQL Database Manager
- Extensible Array Storage
  - – usage scenarios / configurations
  - – Array Storage Extensibility Interface
  - – relational back-end
- Array Query Benchmark
- Summary

Main-memory Array Descriptor

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

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

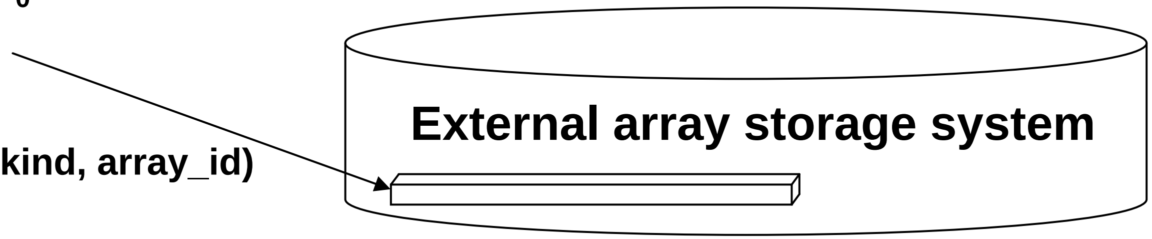
type	size	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
integer	**15**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**

Array Proxy

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

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

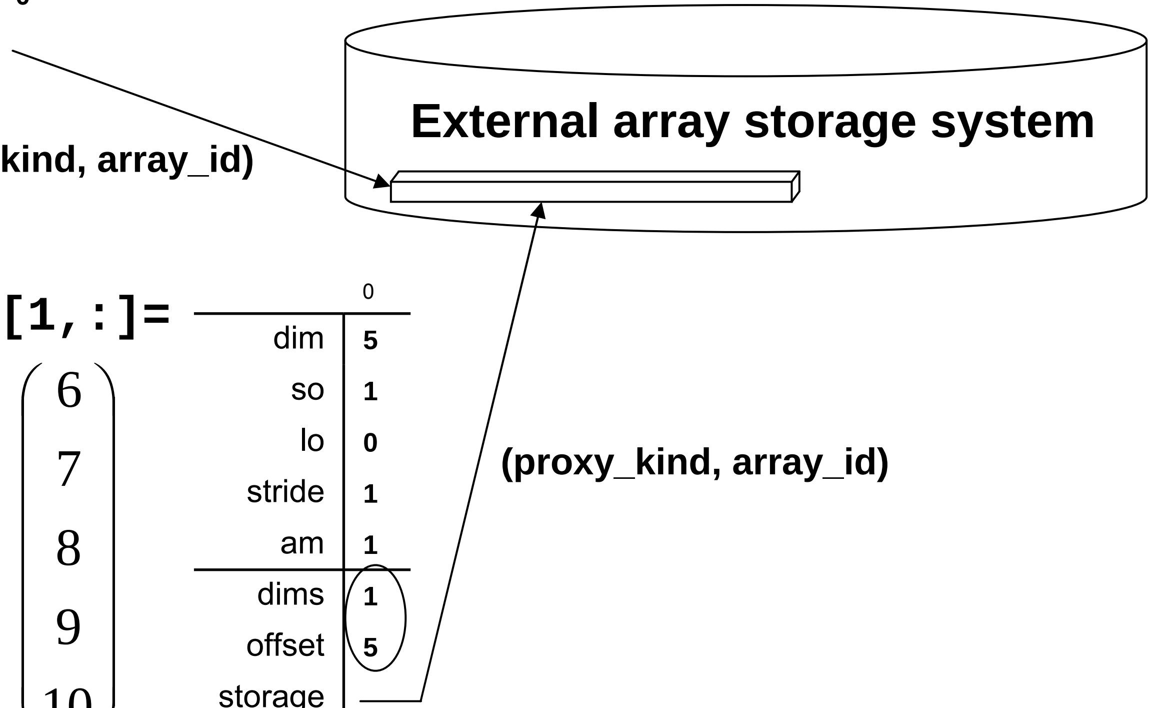
(proxy_kind, array_id)


External array storage system

Array Proxy

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

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



External array storage system

(proxy_kind, array_id)

(proxy_kind, array_id)

`?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	

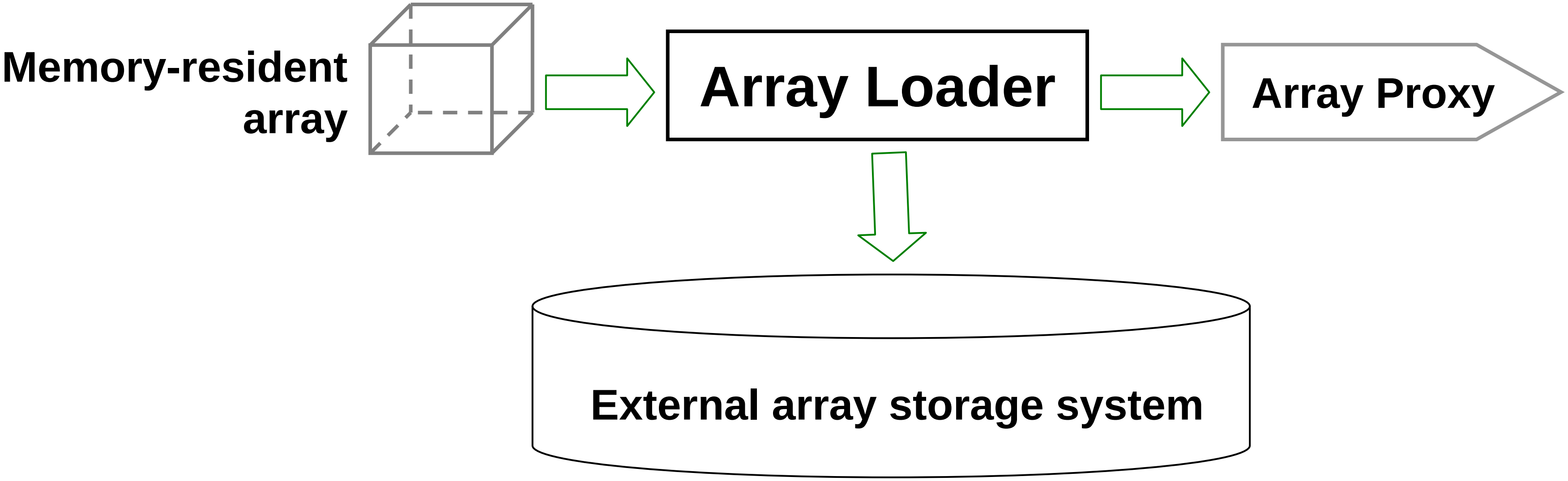
Array Storage Extensibility Interface

?

External array storage system

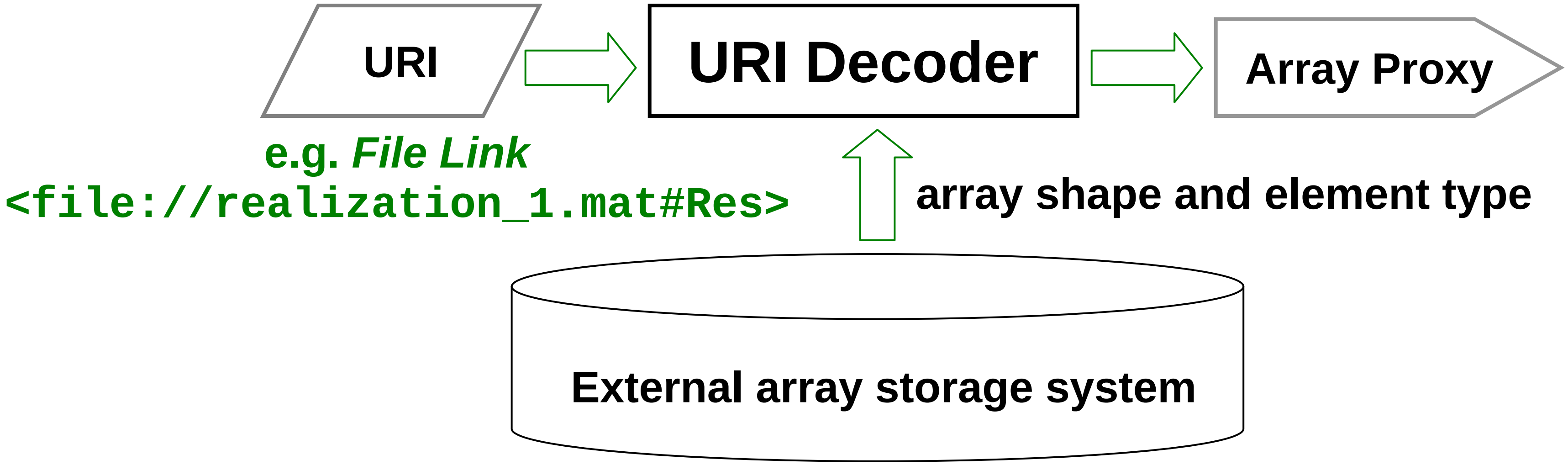
Array Storage Extensibility Interface

- **Array Loader**



Array Storage Extensibility Interface

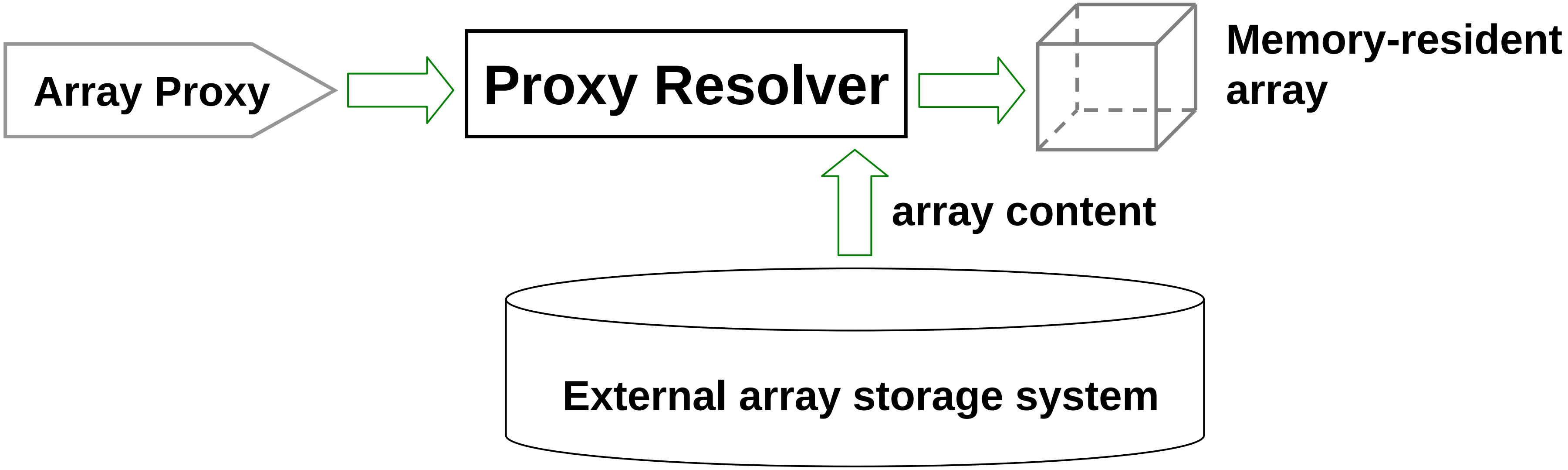
- **URI Decoder**



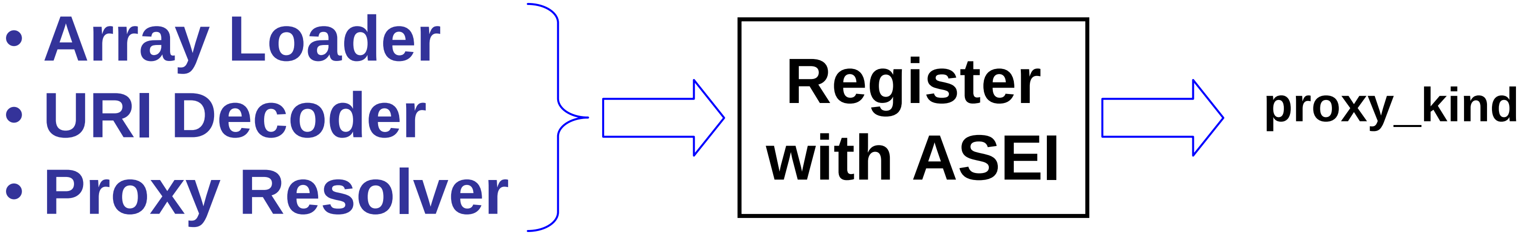
URI
URI Decoder
Array Proxy
e.g. *File Link*
<file://realization_1.mat#Res>
array shape and element type
External array storage system

Array Storage Extensibility Interface

- Proxy Resolver



Array Storage Extensibility Interface



- Array Loader
- URI Decoder
- Proxy Resolver

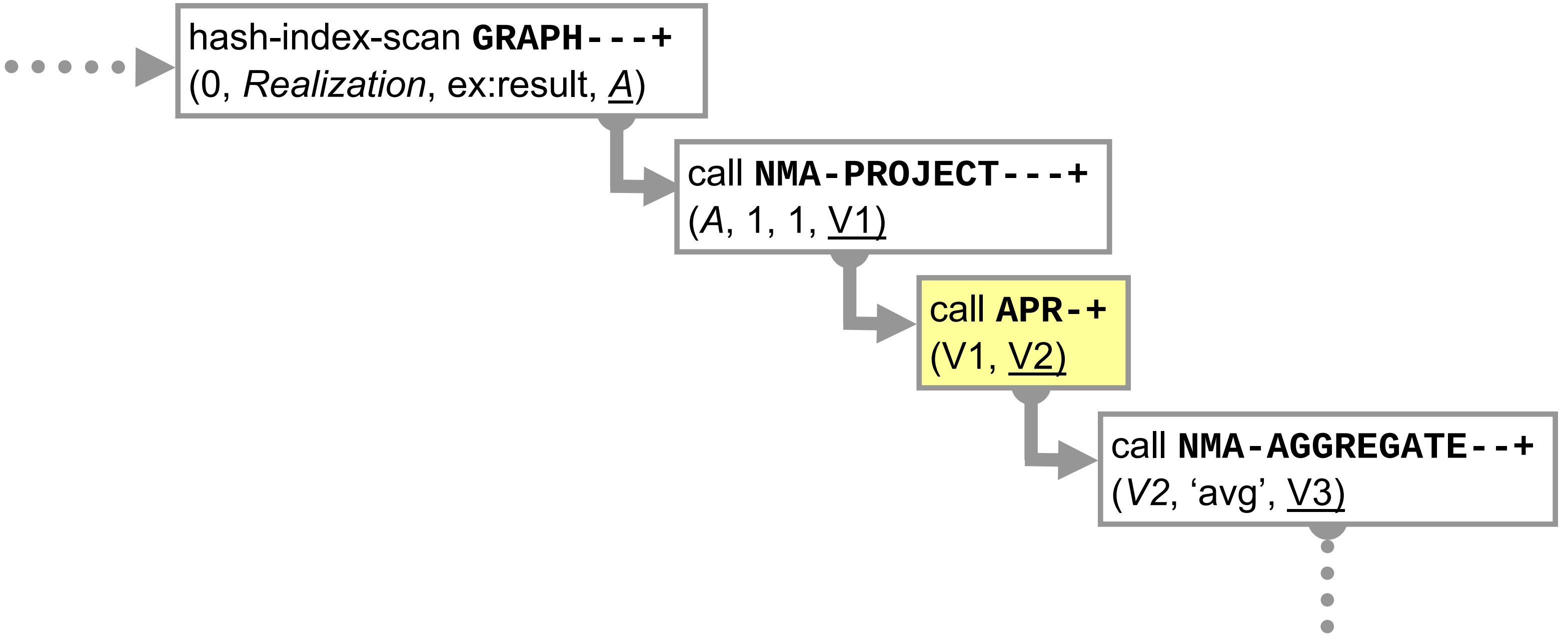
Register with ASEI

proxy_kind

Proxy Resolver: APR()

```
SELECT (array_avg(?A[:,2]) AS ?col2_avg)
 WHERE { [] ex:id 1 ;
            ex:result ?A }
```

```
select rdf:array_avg(APR(aref(a,1,1)))
  from Literal a, Literal g:0
 where (g:0, URI('http://udbl.uu.se/ex#id'), 1) in GRAPH(0)
   and (g:0, URI('http://udbl.uu.se/ex#result'), a) in GRAPH(0);
```



hash-index-scan **GRAPH---+**
(0, *Realization*, ex:result, *A*)

call **NMA-PROJECT---+**
(*A*, 1, 1, V1)

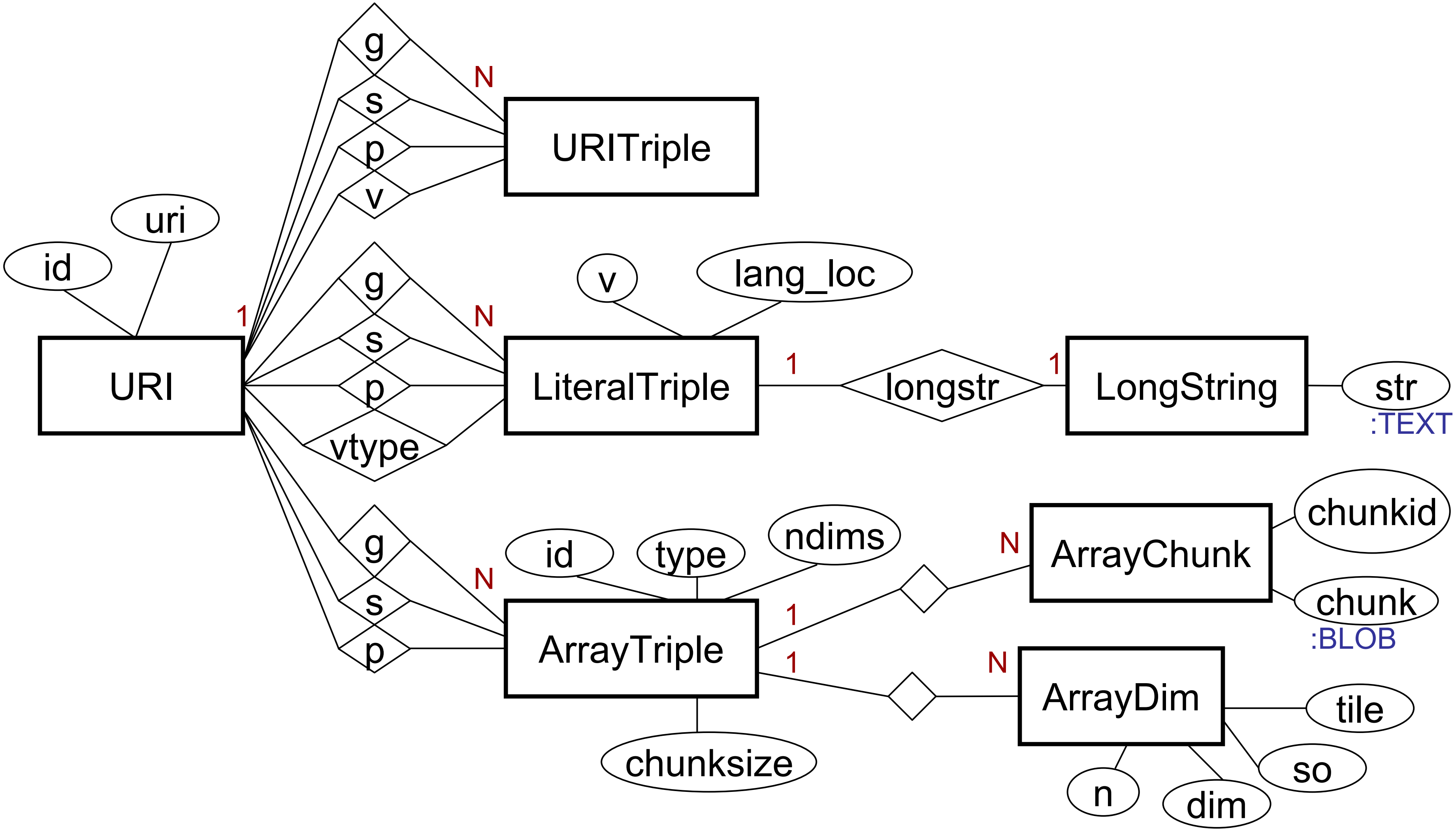
call **APR-+**
(V1, V2)

call **NMA-AGGREGATE--+**
(*V2*, 'avg', V3)

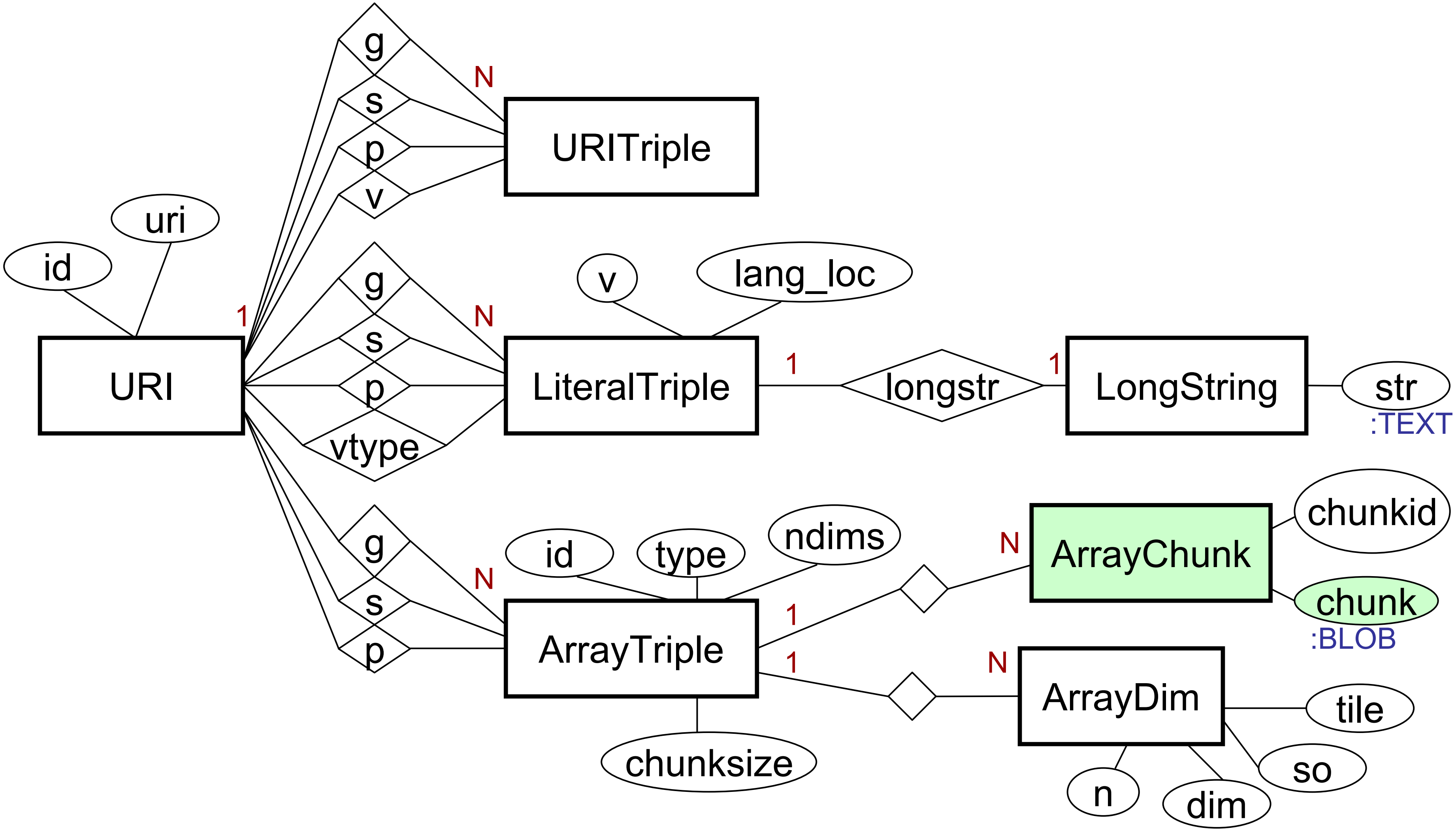
- Introduction
- RDF & SPARQL
- RDF with Arrays & Scientific SPARQL
- Scientific SPARQL Database Manager
- Extensible Array Storage
  - usage scenarios / configurations
  - Array Storage Extensibility Interface
  - relational back-end
- Array Query Benchmark
- Summary

UPPSALA UNIVERSITET

RDF with Arrays Storage Schema



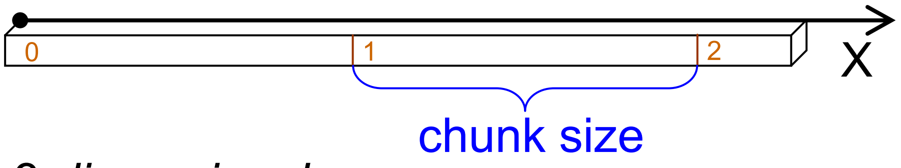
RDF with Arrays Storage Schema



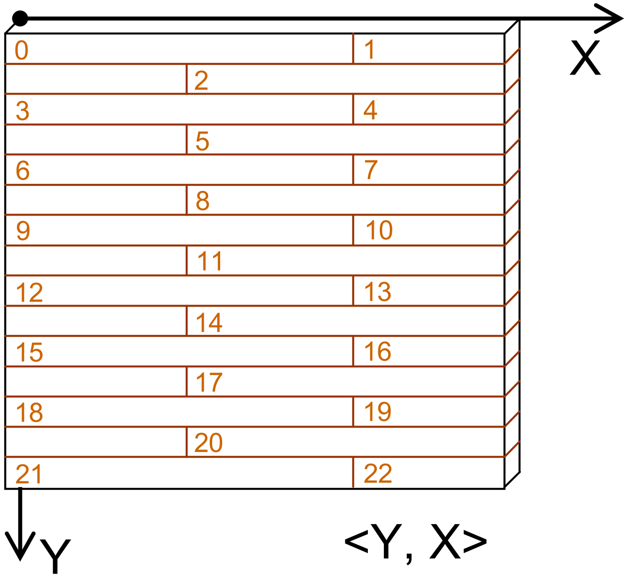
Array Partitioning

Linear Chunks

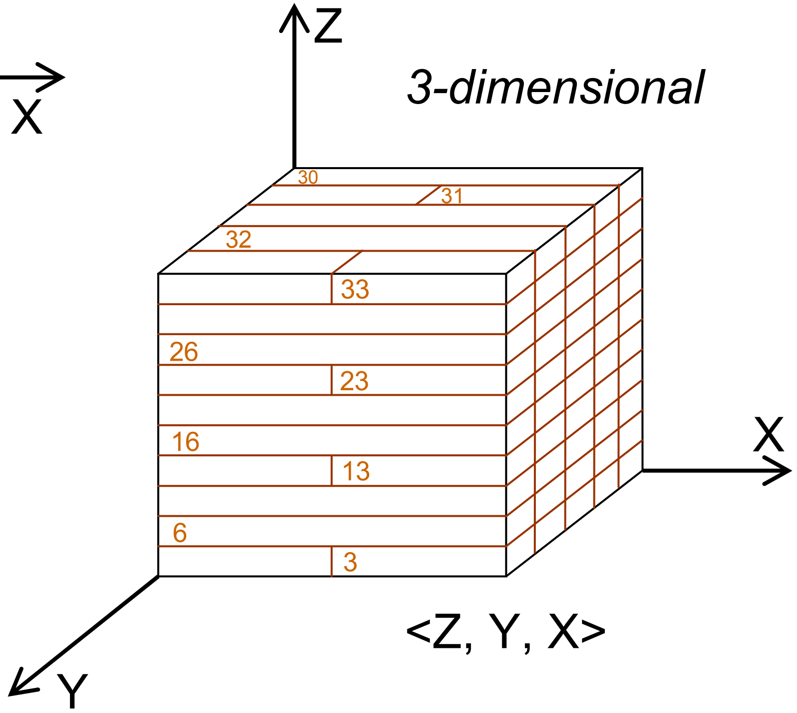
1-dimensional



2-dimensional



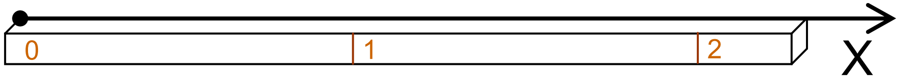
3-dimensional



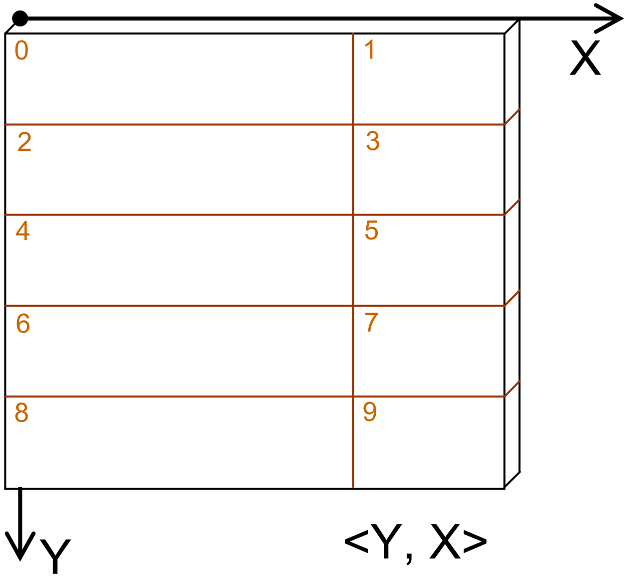
Array Partitioning

Multidimensional chunks

1-dimensional

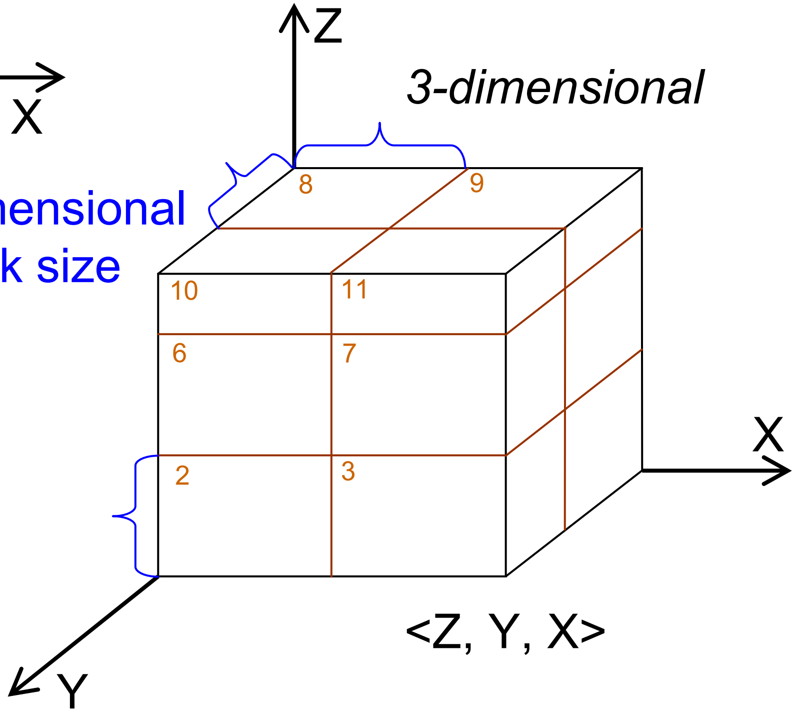


2-dimensional



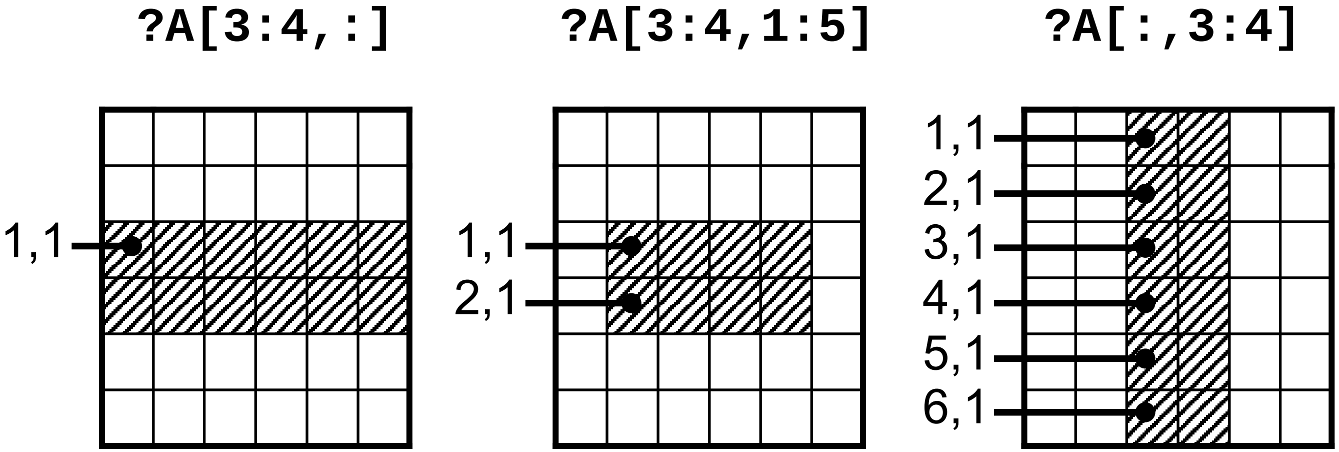
3-dimensional

multidimensional chunk size



Array Fragments

- **Defined by array proxies**



`?A[3:4,:]` `?A[3:4,1:5]` `?A[:,3:4]`

(a)
1 fragment of 12

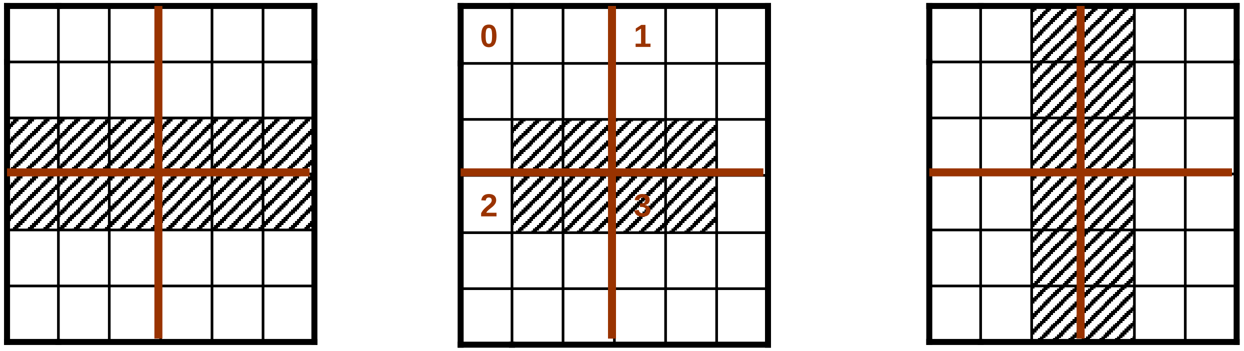
(b)
2 fragments of 4

(c)
6 fragments of 2

Data Transfer Operations

- **Intersections of fragments and chunks**

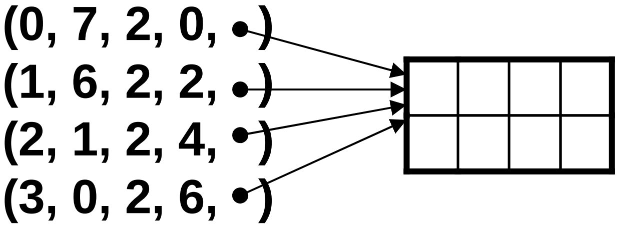
(chunkid, read_pos, length, write_pos, result)



(a)
1 fragment of 12
4 ops of 3

(b)
2 fragments of 4
4 ops of 2

(c)
6 fragments of 2
12 ops of 1

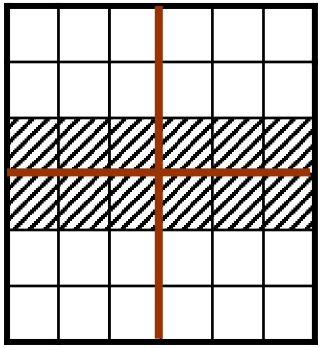


(0, 7, 2, 0, •)
(1, 6, 2, 2, •)
(2, 1, 2, 4, •)
(3, 0, 2, 6, •)

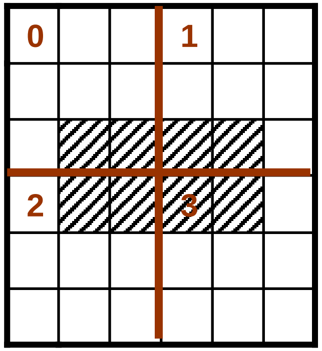
Data Transfer Operations

• Intersections of fragments and chunks

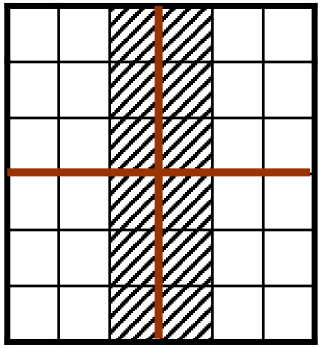
(chunkid, read_pos, length, write_pos, result)



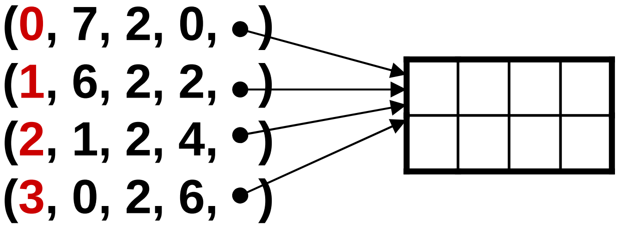
(a)
1 fragment of 12
4 ops of 3



(b)
2 fragments of 4
4 ops of 2



(c)
6 fragments of 2
12 ops of 1



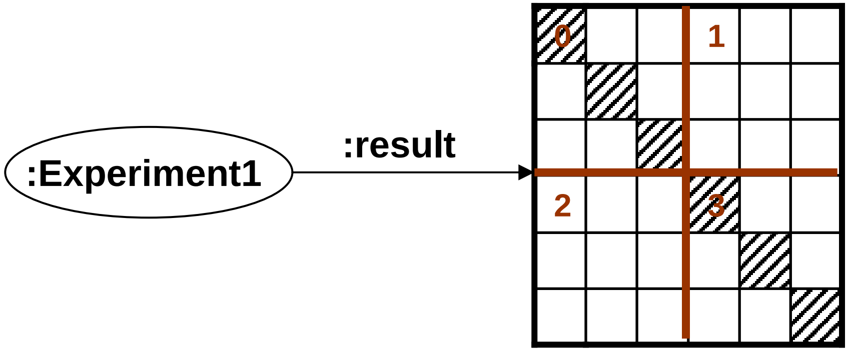
(0, 7, 2, 0, •)
(1, 6, 2, 2, •)
(2, 1, 2, 4, •)
(3, 0, 2, 6, •)

Optimize array content retrieval

- **for single proxy involving multiple chunks**
- **for a series of proxies**

Data Transfer Operations

```
SELECT ?i (?A[?i, ?i] AS ?e)
 WHERE { :Experiment1 :result ?A }
```

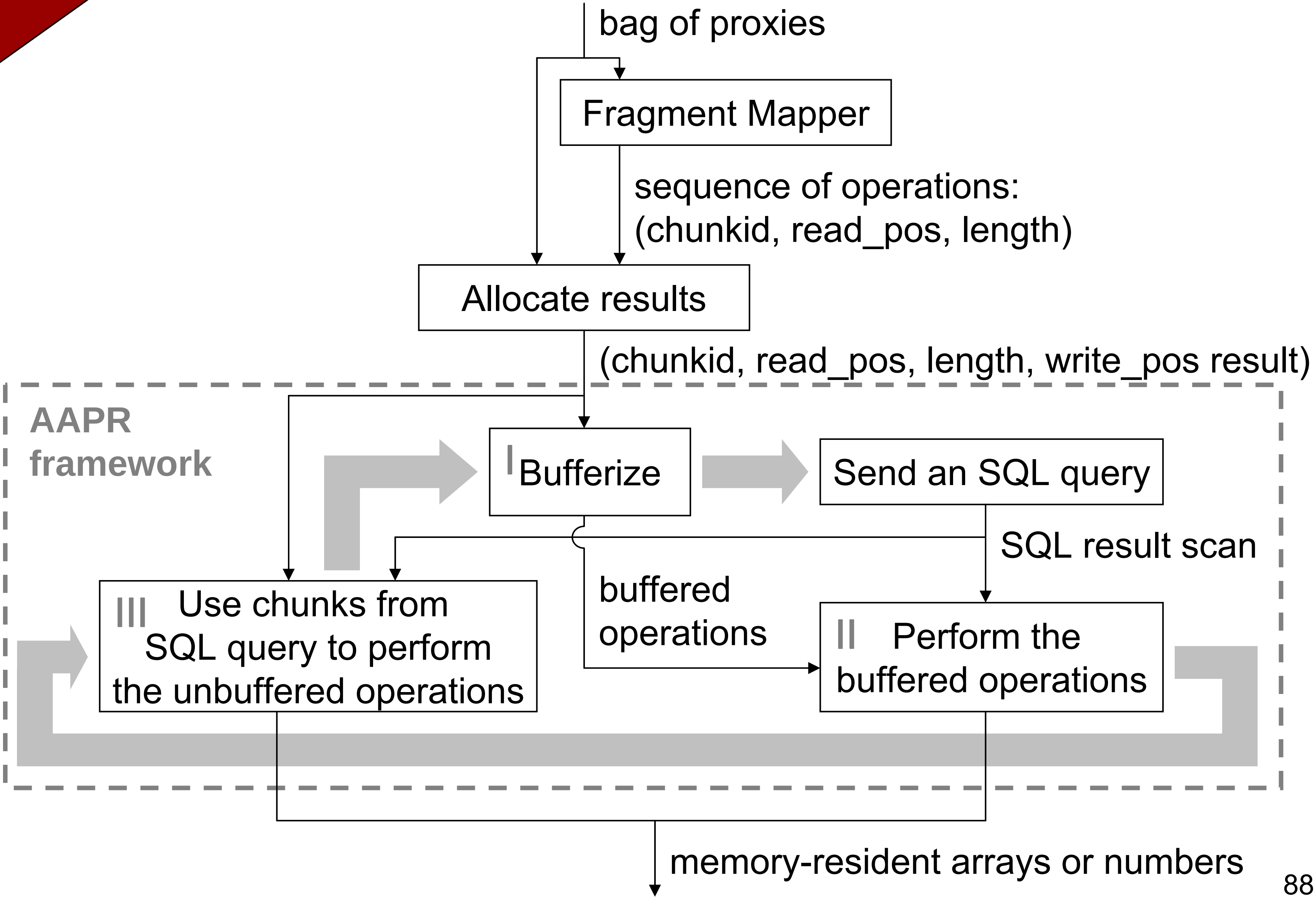


(b)
6 fragments of 1
6 ops of 1

(0, 0, 1, 0, •) → □   **(3, 0, 1, 0, •) → □**
(0, 4, 1, 0, •) → □   **(3, 4, 1, 0, •) → □**
(0, 8, 1, 0, •) → □   **(3, 8, 1, 0, •) → □**

Optimize array content retrieval
- **for single proxy involving multiple chunks**
- **for a series of proxies**

Aggregated APR Framework



bag of proxies

Fragment Mapper

sequence of operations: (chunkid, read_pos, length)

Allocate results

(chunkid, read_pos, length, write_pos result)

AAPR framework

I Bufferize

Send an SQL query

SQL result scan

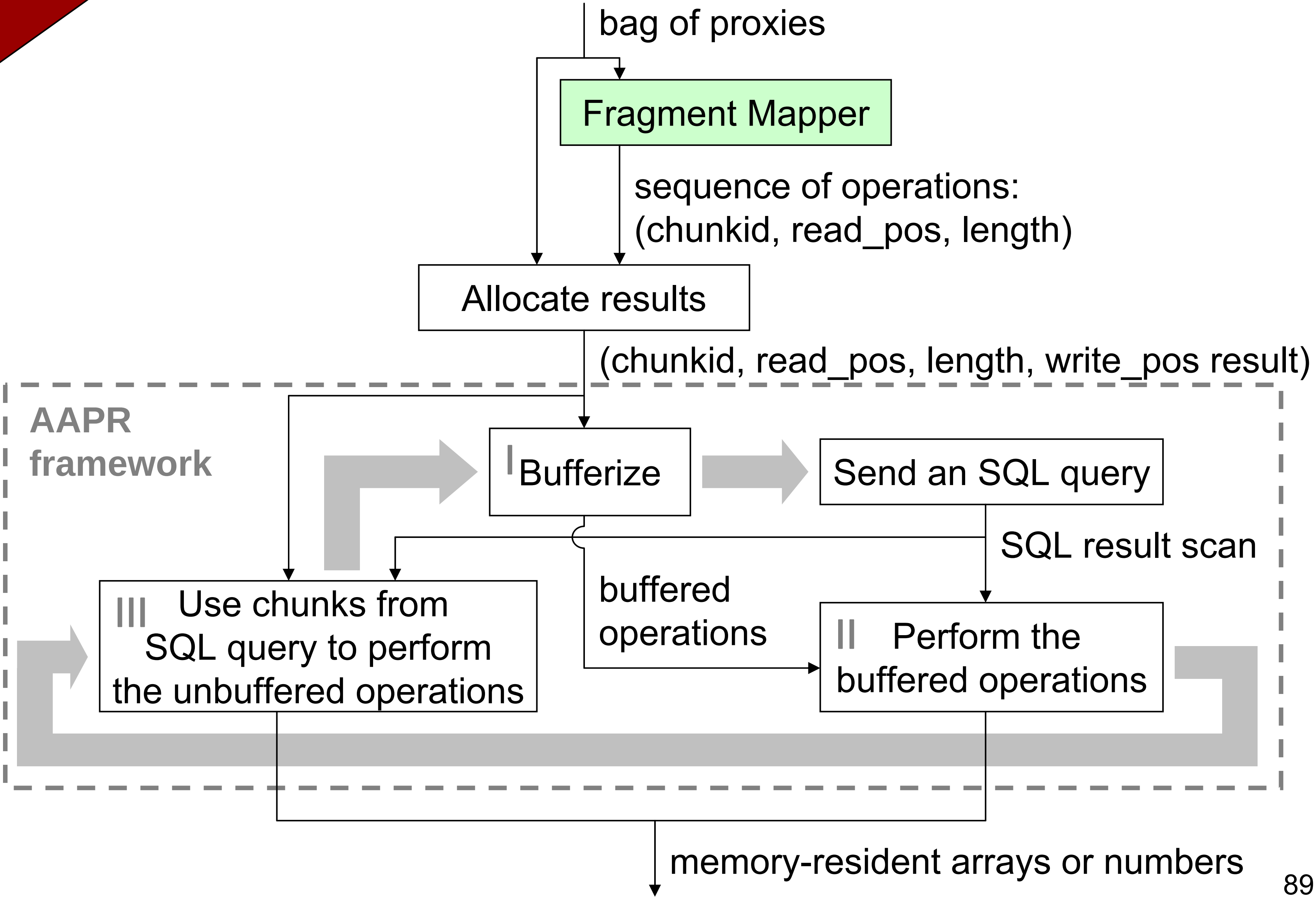
buffered operations

II Perform the buffered operations

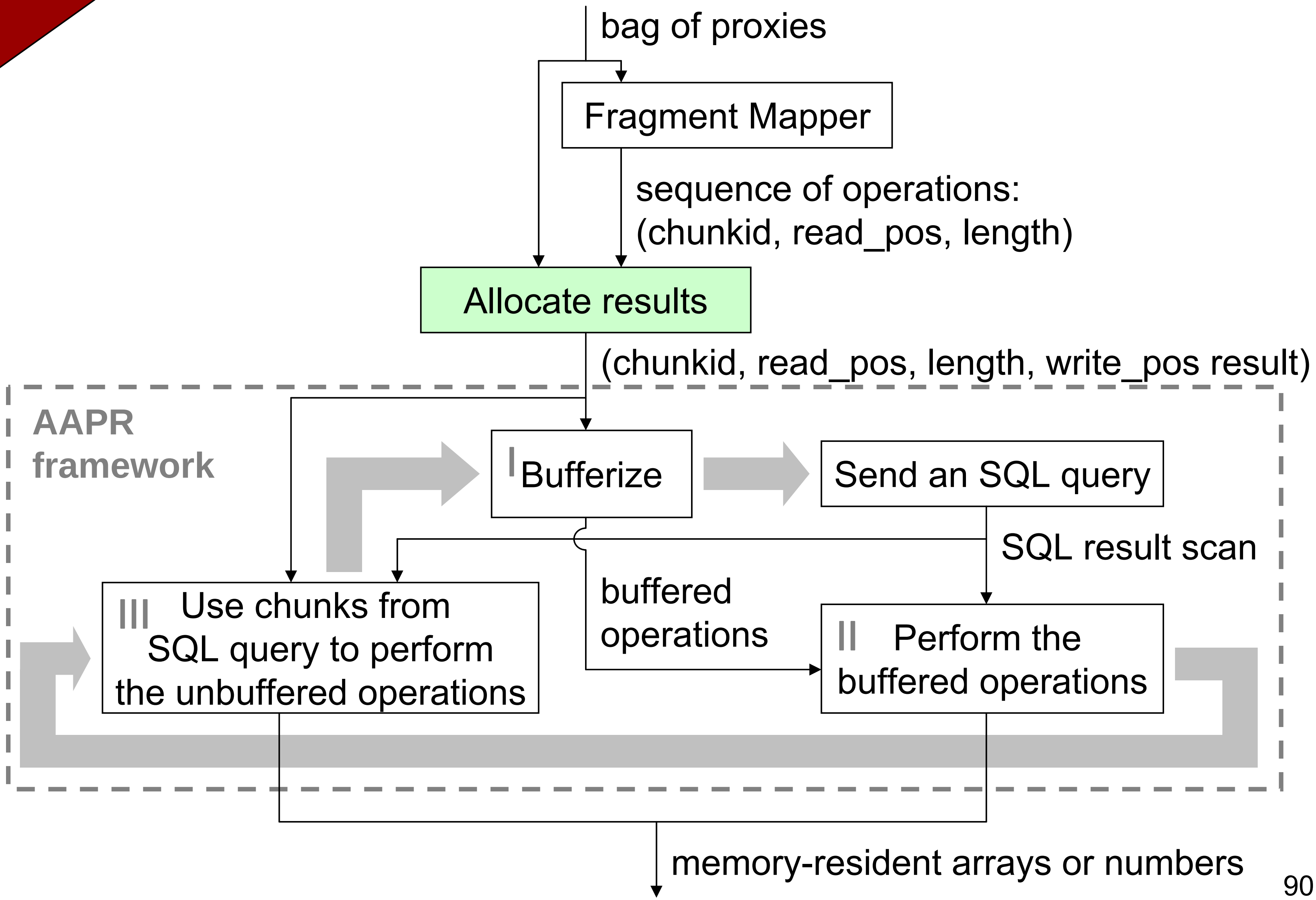
III Use chunks from SQL query to perform the unbuffered operations

memory-resident arrays or numbers

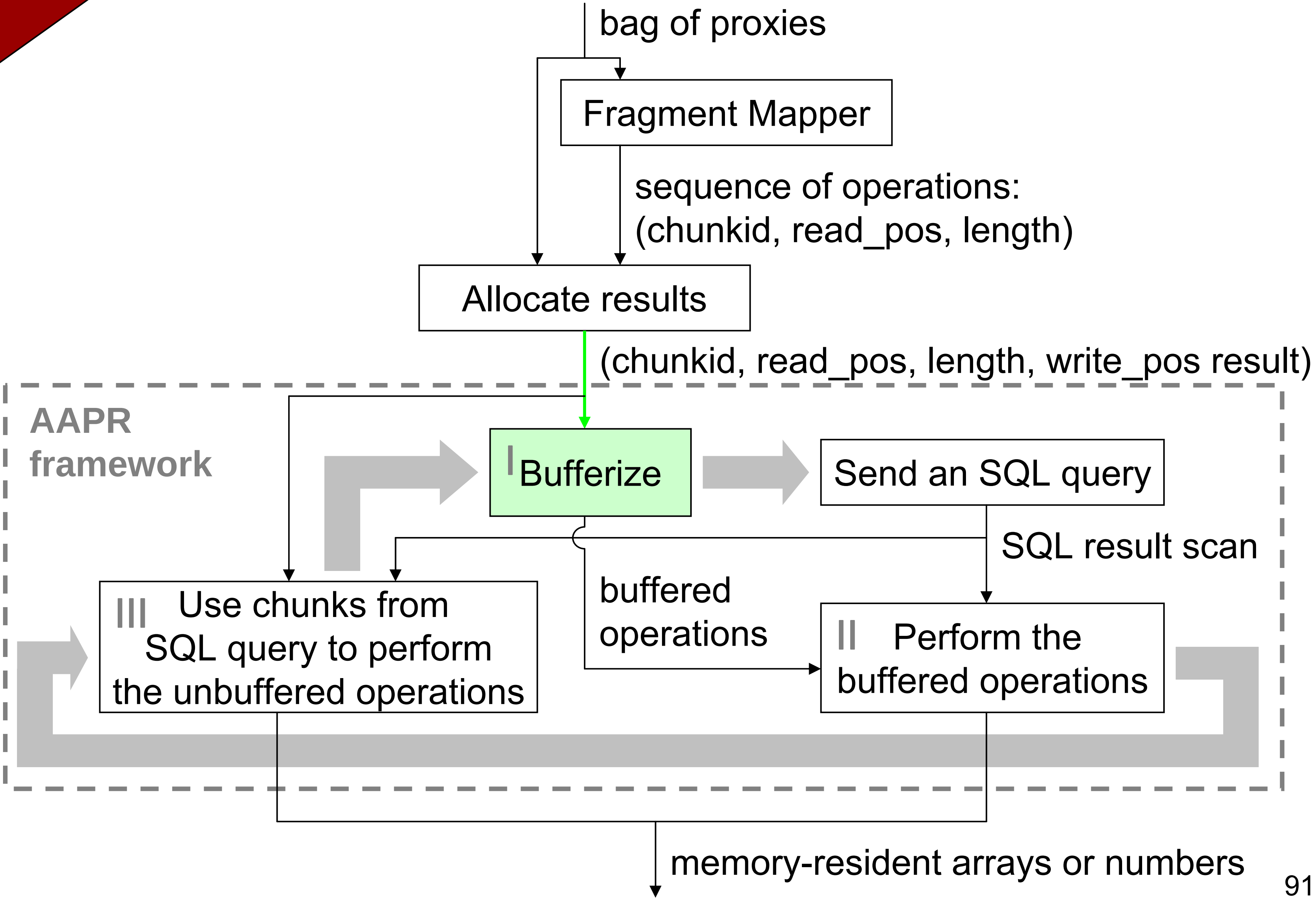
Aggregated APR Framework



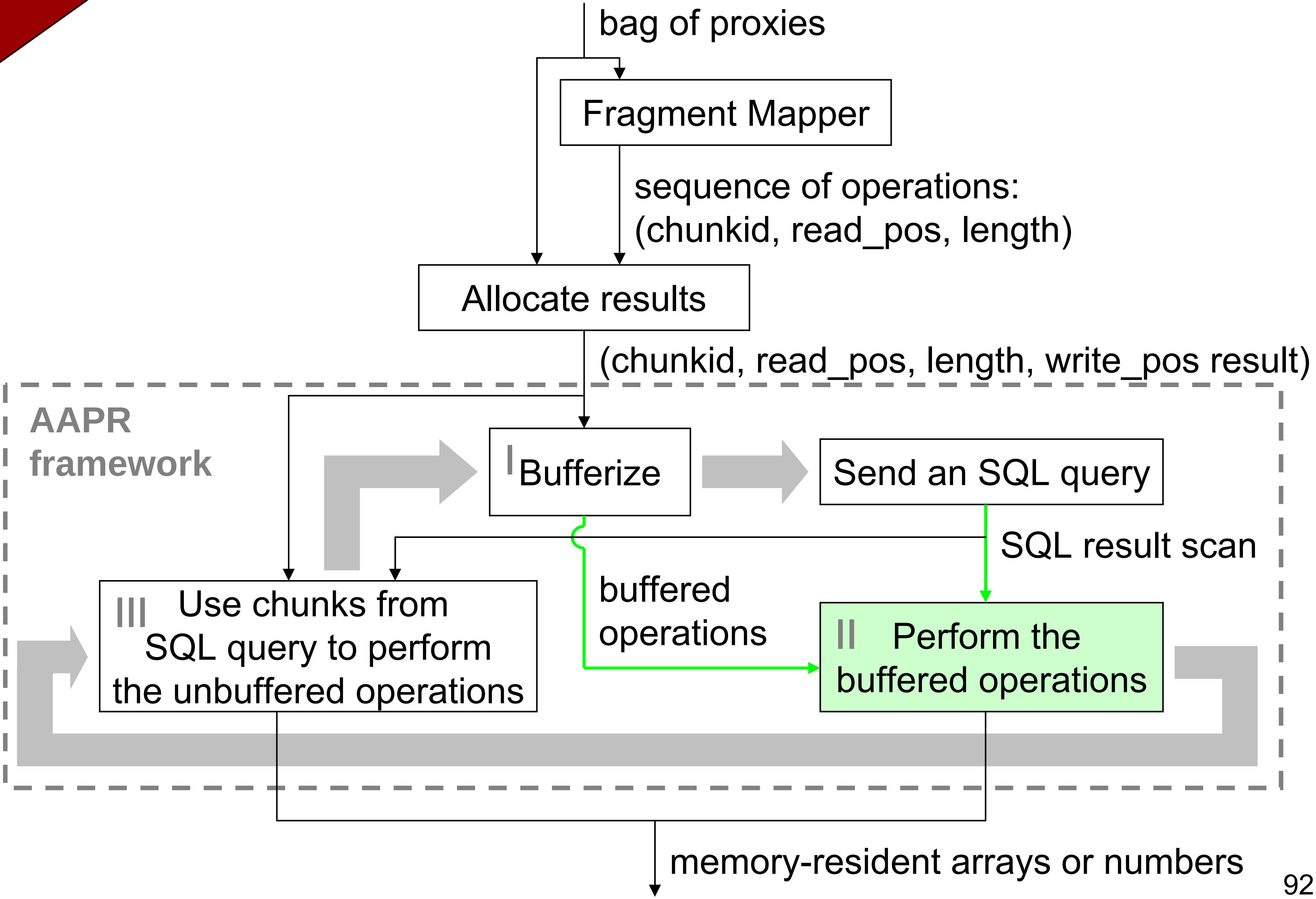
Aggregated APR Framework



Aggregated APR Framework



Aggregated APR Framework



bag of proxies

Fragment Mapper

sequence of operations:
(chunkid, read_pos, length)

Allocate results

(chunkid, read_pos, length, write_pos result)

AAPR
framework

I Bufferize

Send an SQL query

SQL result scan

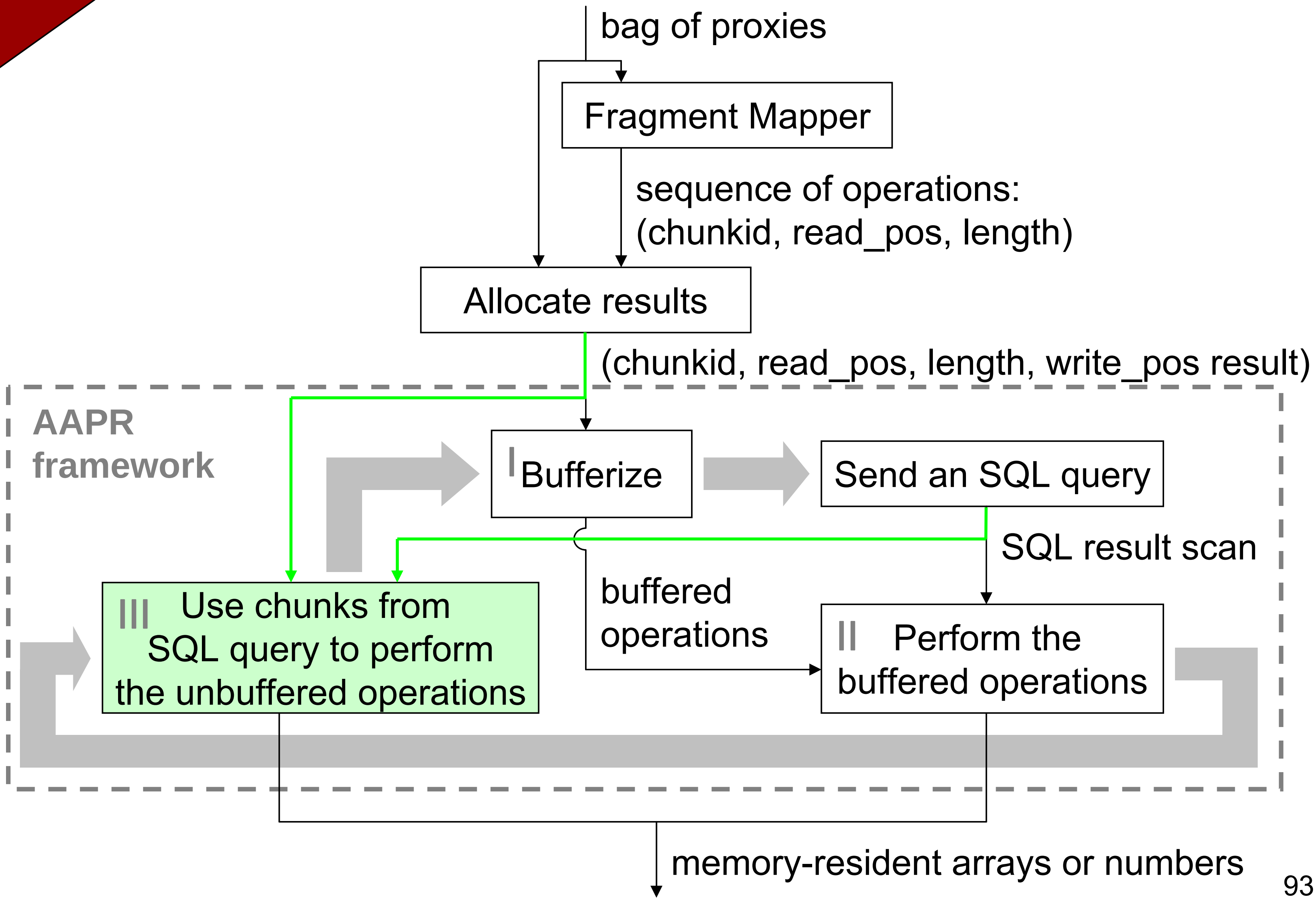
buffered
operations

II Perform the
buffered operations

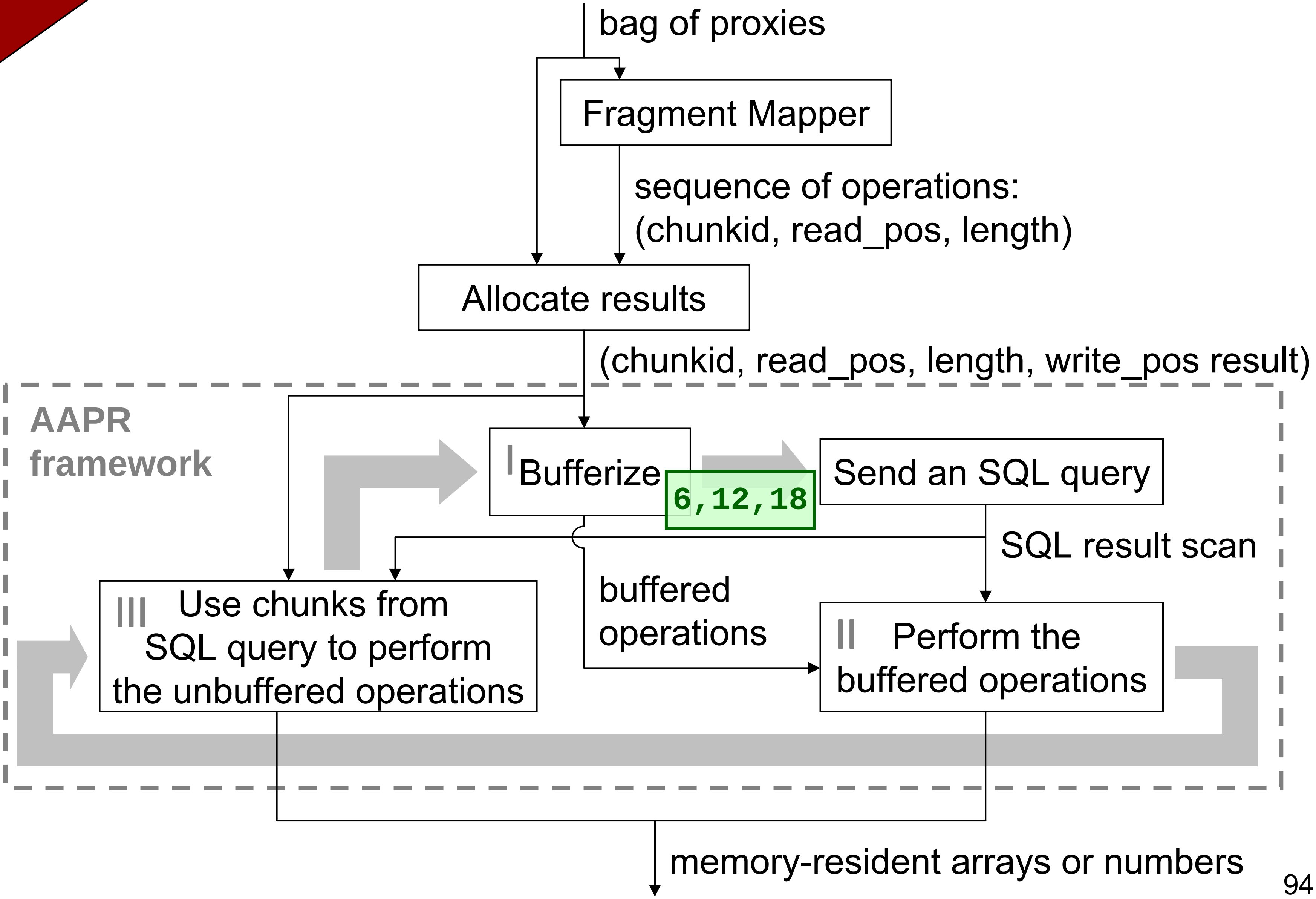
III Use chunks from
SQL query to perform
the unbuffered operations

memory-resident arrays or numbers

Aggregated APR Framework



Aggregated APR Framework



bag of proxies

Fragment Mapper

sequence of operations:
(chunkid, read_pos, length)

Allocate results

(chunkid, read_pos, length, write_pos result)

AAPR framework

I Bufferize

6,12,18

Send an SQL query

SQL result scan

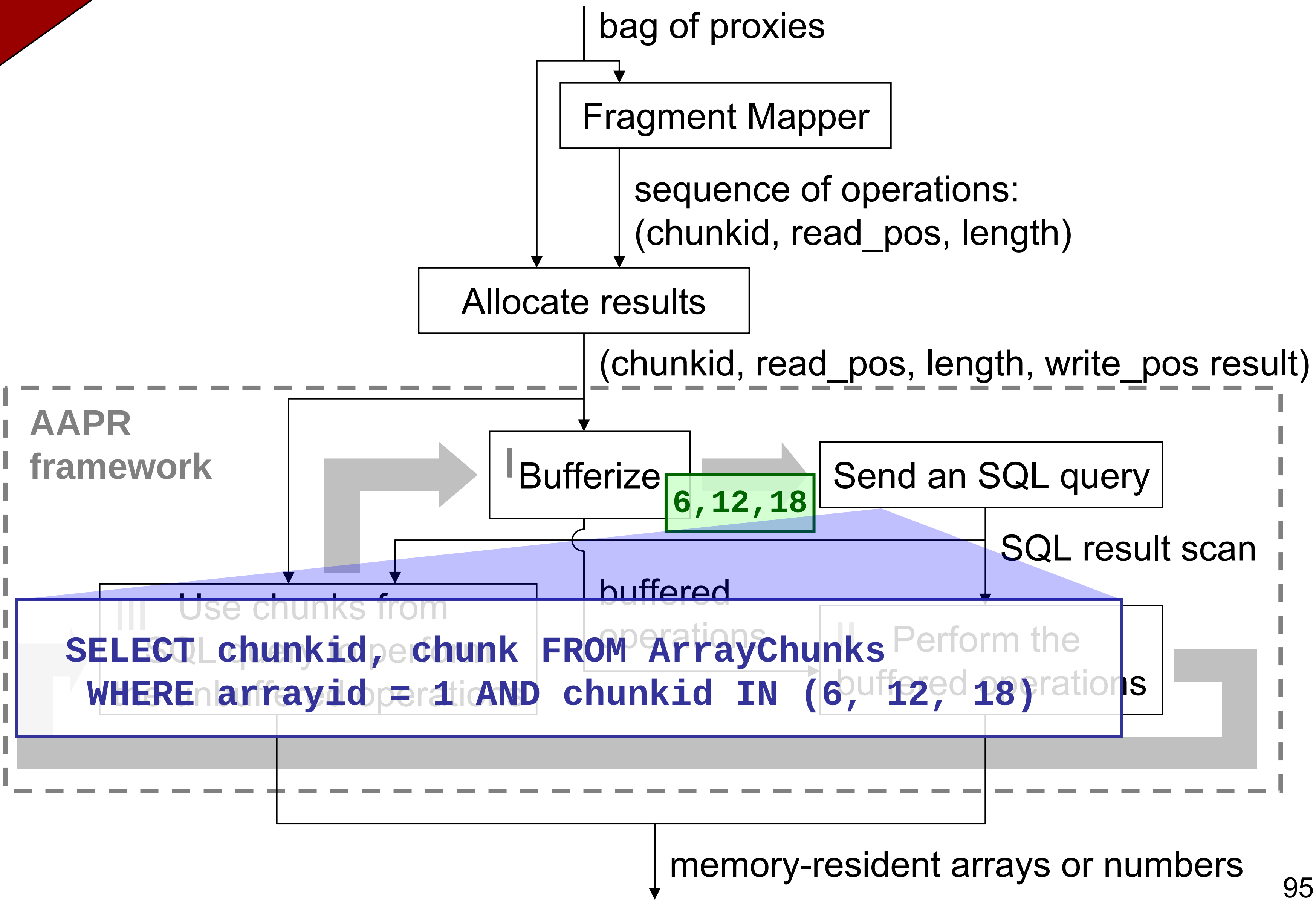
buffered operations

III Use chunks from SQL query to perform the unbuffered operations

II Perform the buffered operations

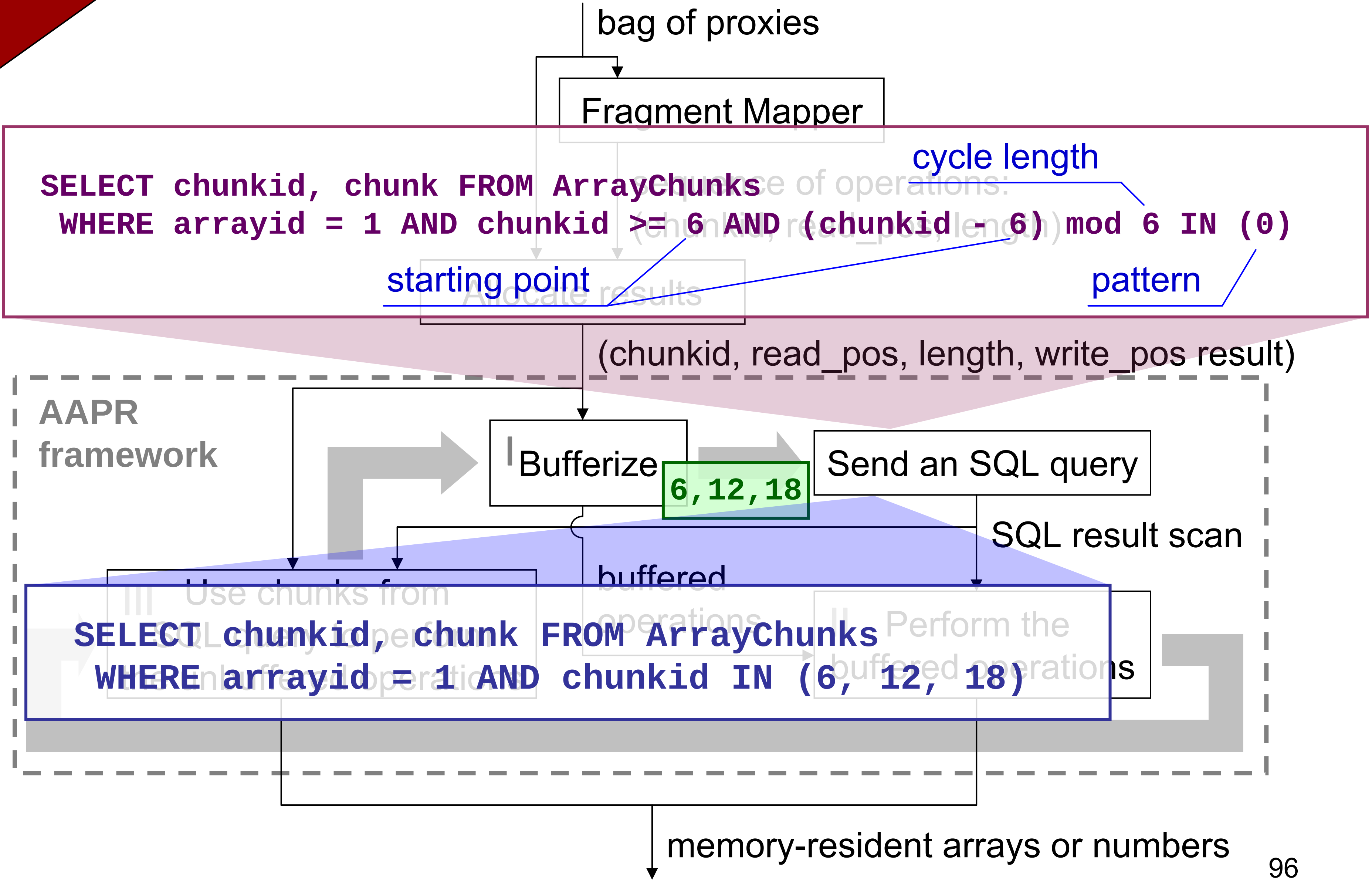
memory-resident arrays or numbers

Aggregated APR Framework



```
SELECT chunkid, chunk FROM ArrayChunks
 WHERE arrayid = 1 AND chunkid IN (6, 12, 18)
```

Aggregated APR Framework



bag of proxies
Fragment Mapper
sequence of operations:
(chunkid, read_pos, length)
Allocate results
(chunkid, read_pos, length, write_pos result)
AAPR framework
Bufferize
6,12,18
Send an SQL query
SQL result scan
buffered operations
Use chunks from SQL query to perform the buffered operations
Perform the buffered operations
memory-resident arrays or numbers
cycle length
starting point
pattern

```
SELECT chunkid, chunk FROM ArrayChunks
 WHERE arrayid = 1 AND chunkid >= 6 AND (chunkid - 6) mod 6 IN (0)
```

```
SELECT chunkid, chunk FROM ArrayChunks
 WHERE arrayid = 1 AND chunkid IN (6, 12, 18)
```

- Introduction
- RDF & SPARQL
- RDF with Arrays & Scientific SPARQL
- Scientific SPARQL Database Manager
- Extensible Array Storage
- Array Query Benchmark
  - benchmark definition
  - results
- Summary

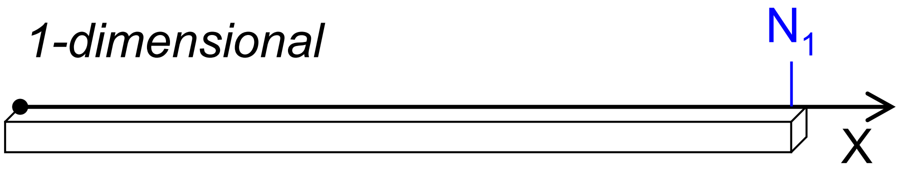
Array Model

$$A\colon \underbrace{\{1..N_1\} \times \ldots \times \{1..N_n\}}_{\text{domain}} \rightarrow \underbrace{R}_{\text{range}}$$

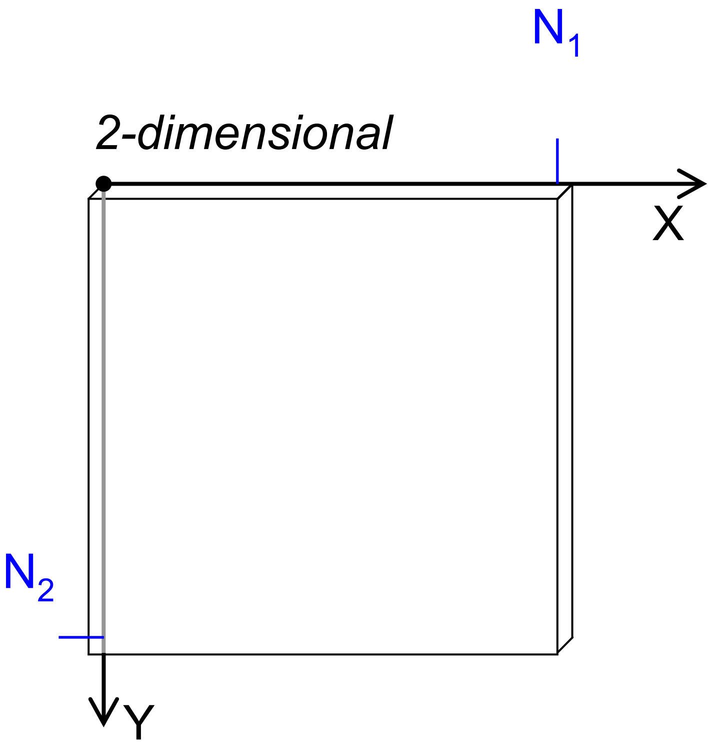
array shape:

$$<N_1 \ldots N_n>$$

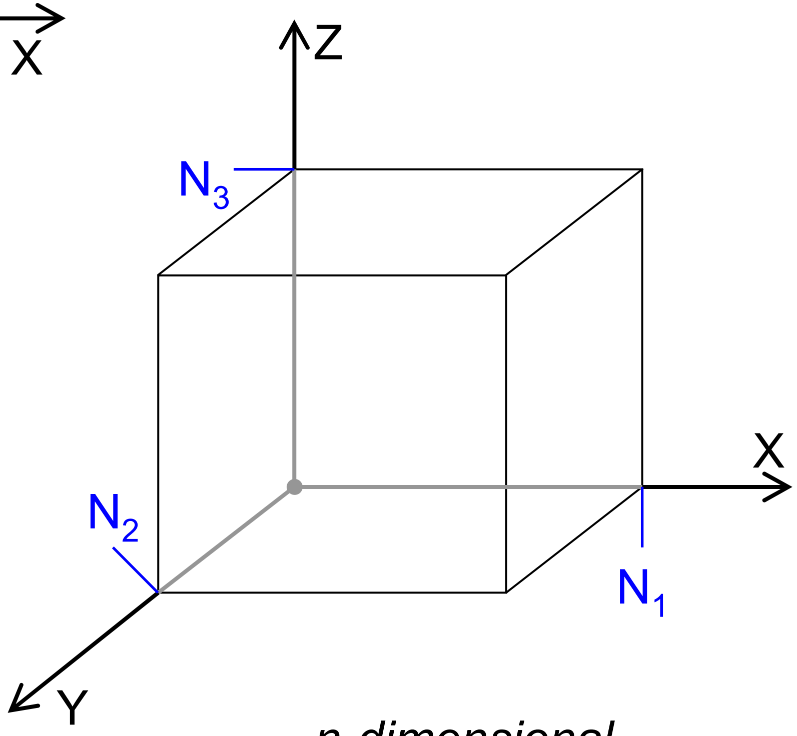
1-dimensional



2-dimensional



3-dimensional



n-dimensional

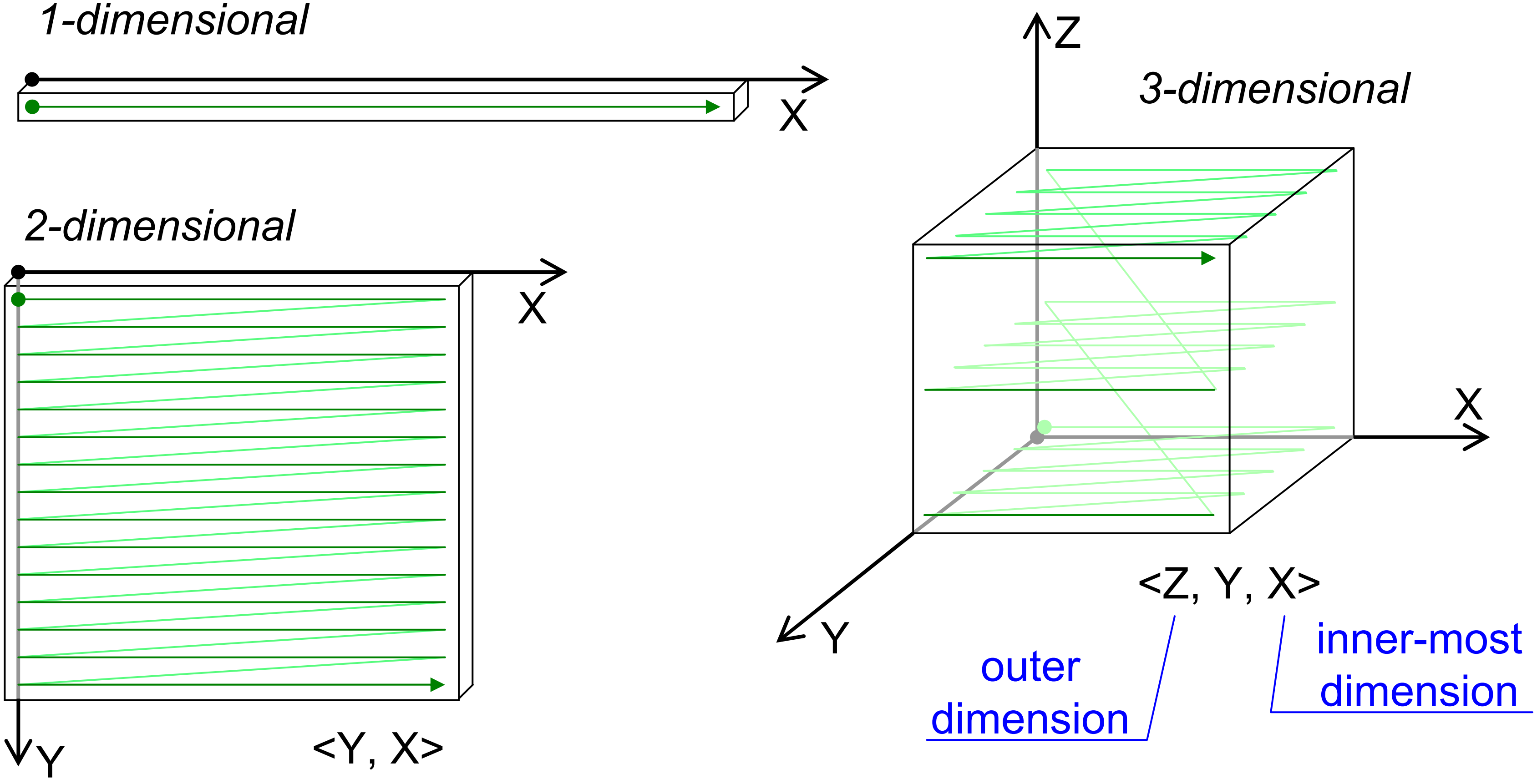
?

Benchmark Variables

Data properties

- array shape and element type

Array Storage Order



1-dimensional

X

2-dimensional

X

Y

<Y, X>

(row-by-row storage)

3-dimensional

Z

X

Y

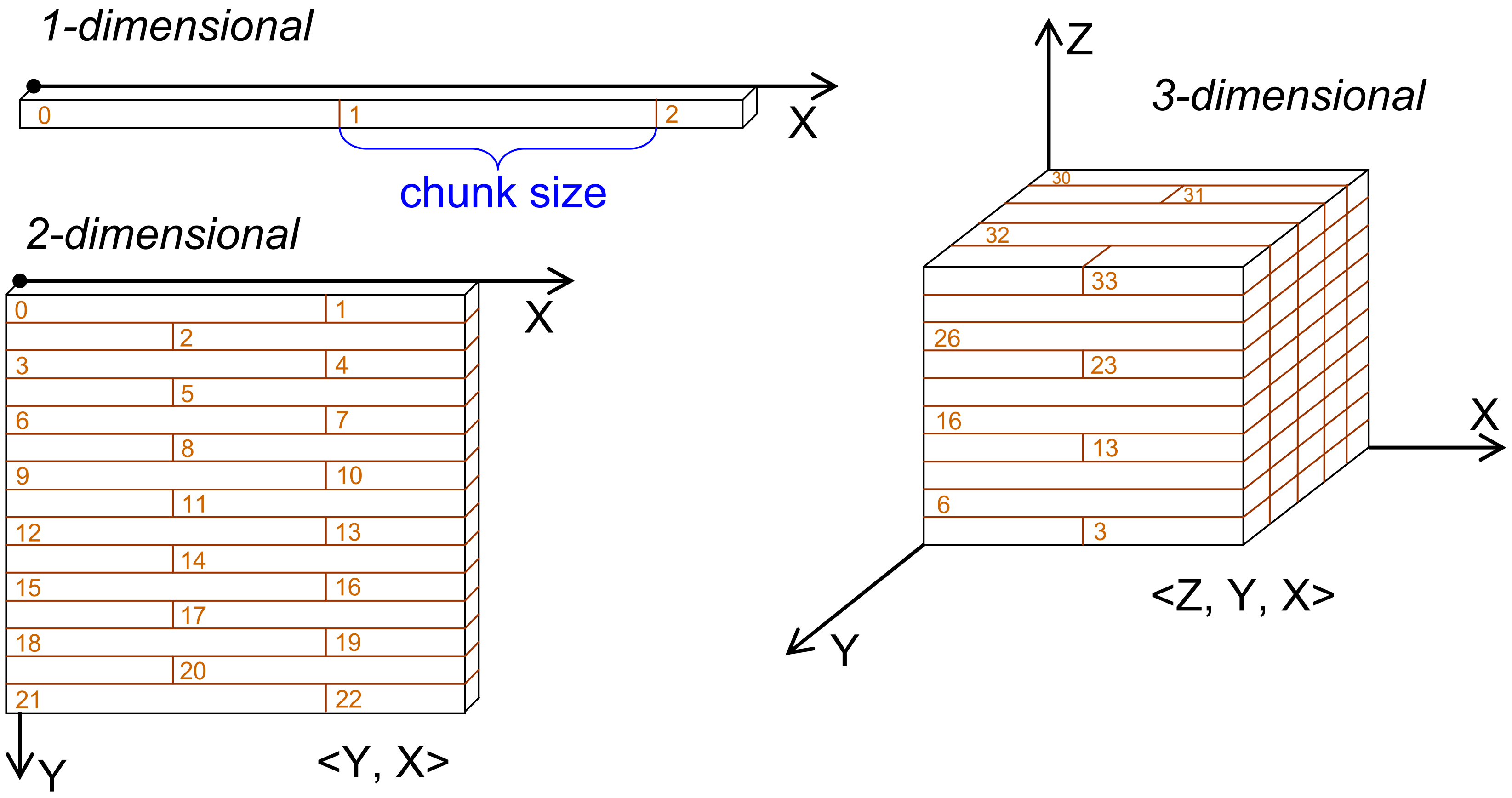
<Z, Y, X>

outer dimension

inner-most dimension

Array Partitioning

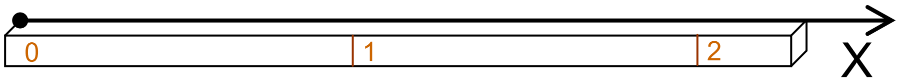
Linear Chunks



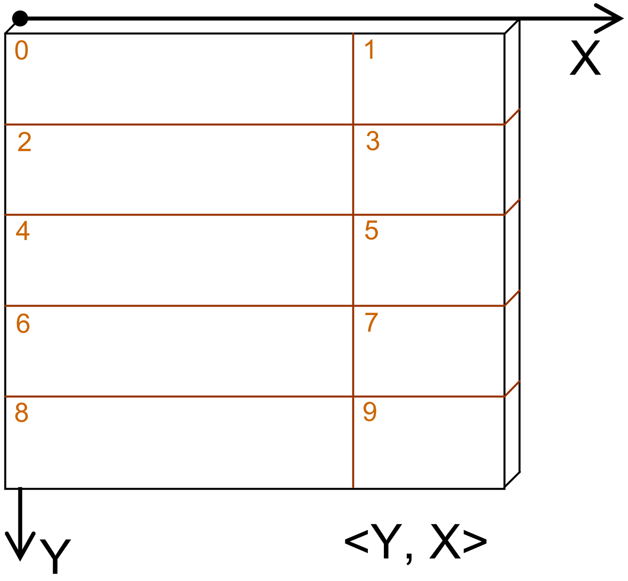
Array Partitioning

Multidimensional chunks

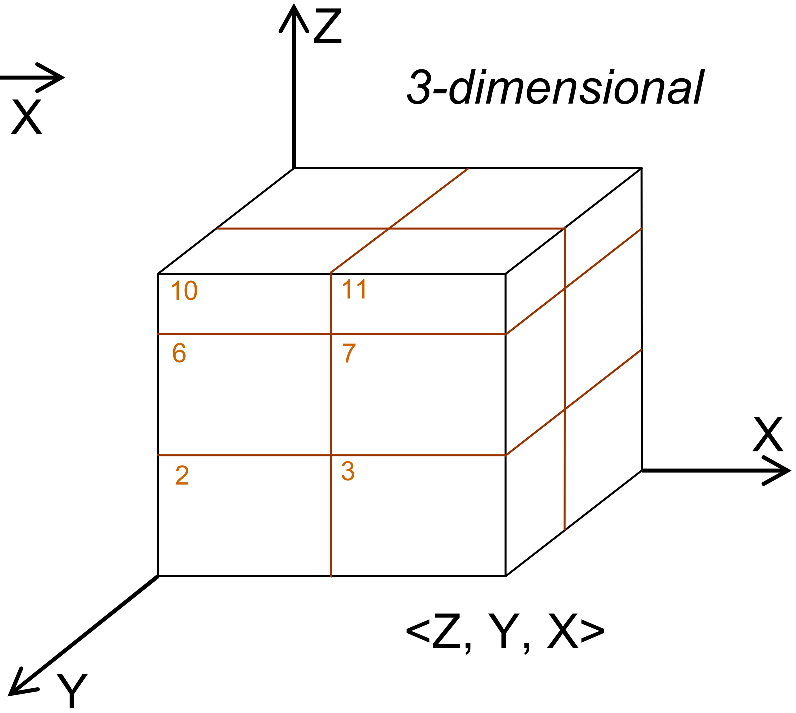
1-dimensional



2-dimensional

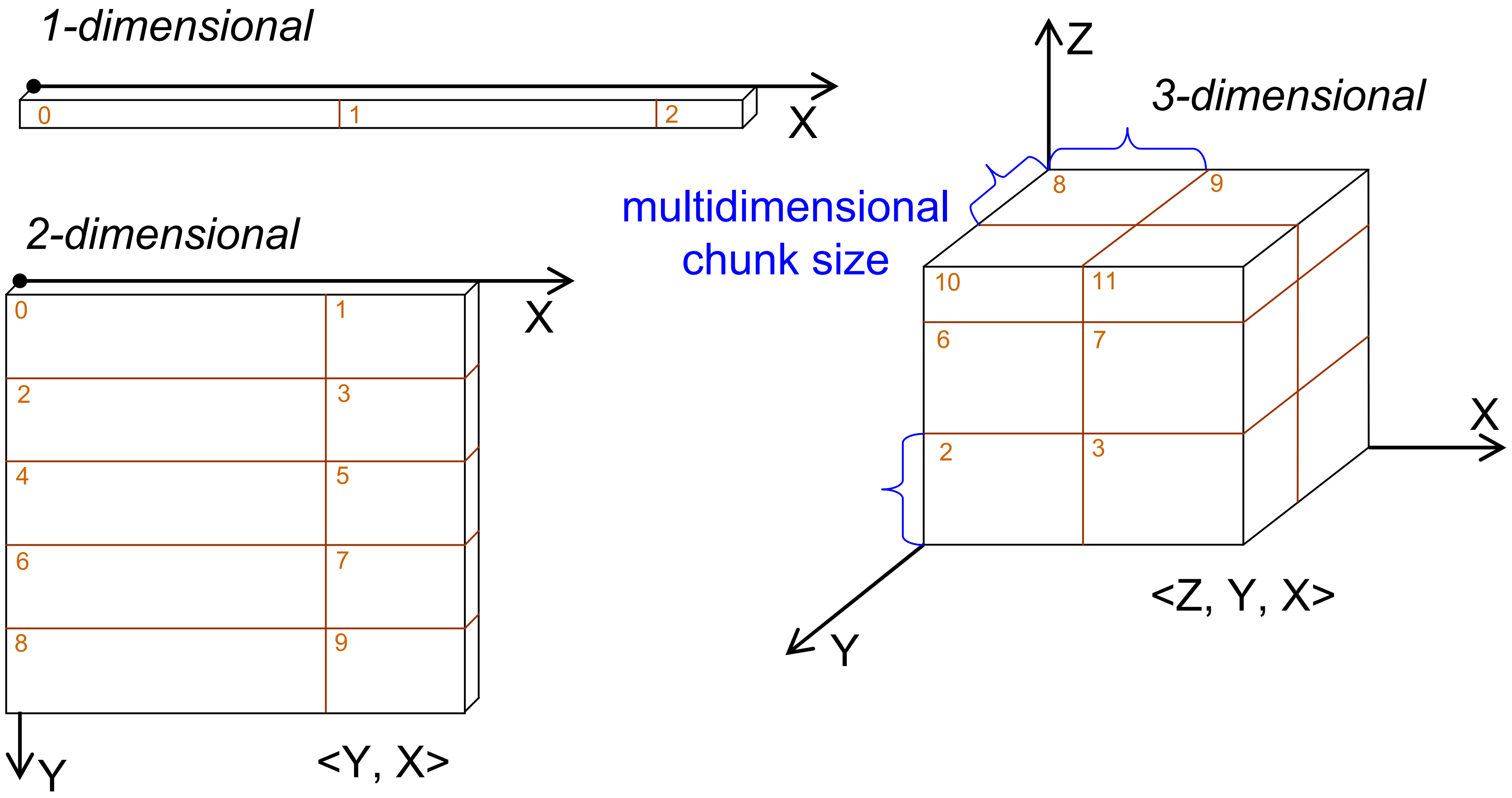


3-dimensional



Array Partitioning

Multidimensional chunks



1-dimensional

2-dimensional

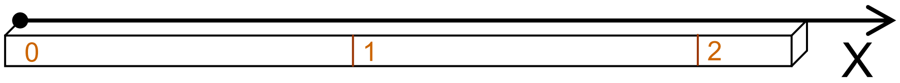
3-dimensional

multidimensional chunk size

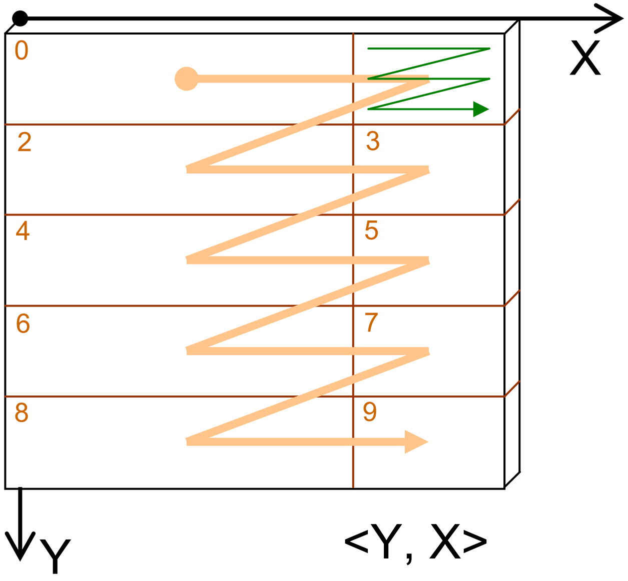
Array Partitioning

Multidimensional chunks

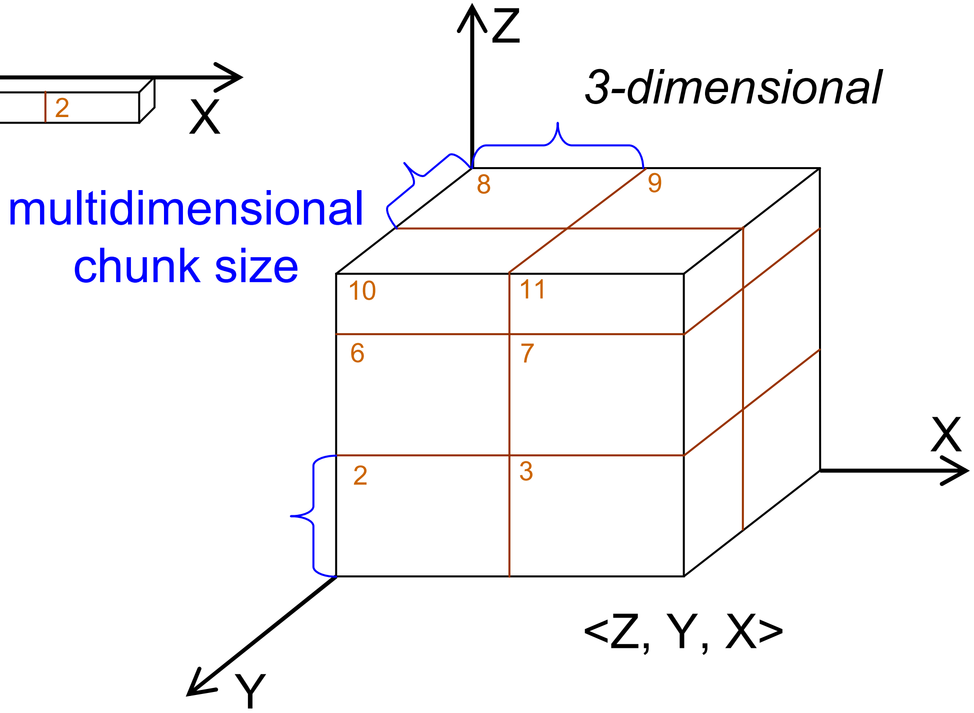
1-dimensional



2-dimensional



3-dimensional



Benchmark Variables

Data properties

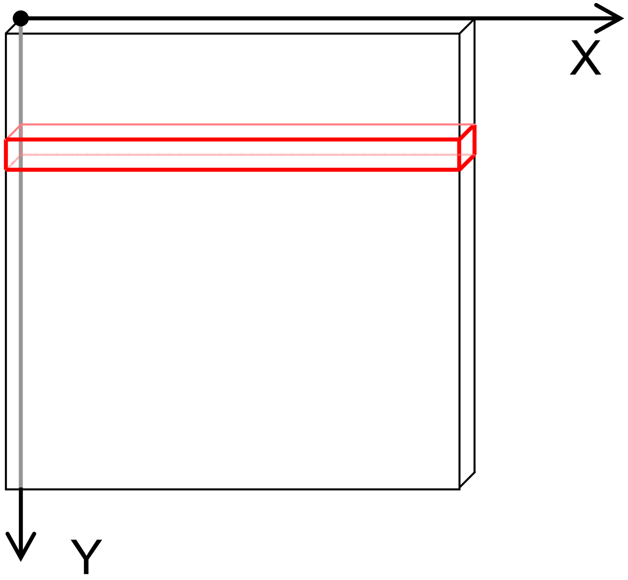
- array shape and element type

Data storage options

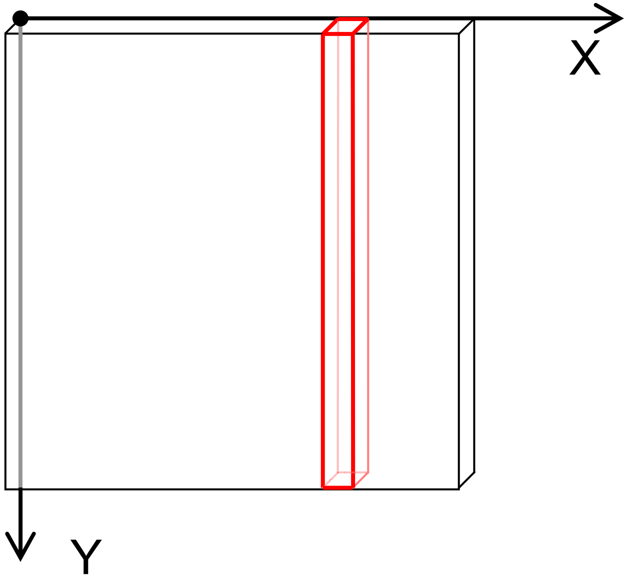
- partitioning: linear / multidimensional
- chunk size
- nesting order of dimensions

Array Access Patterns

- *single row*

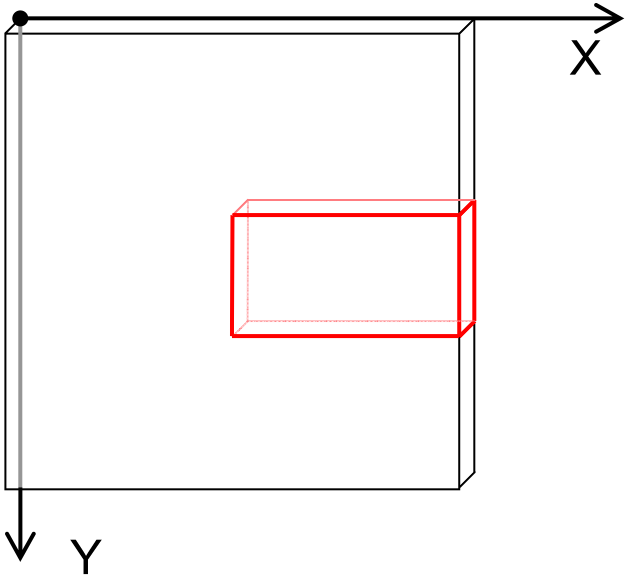


- *single column*

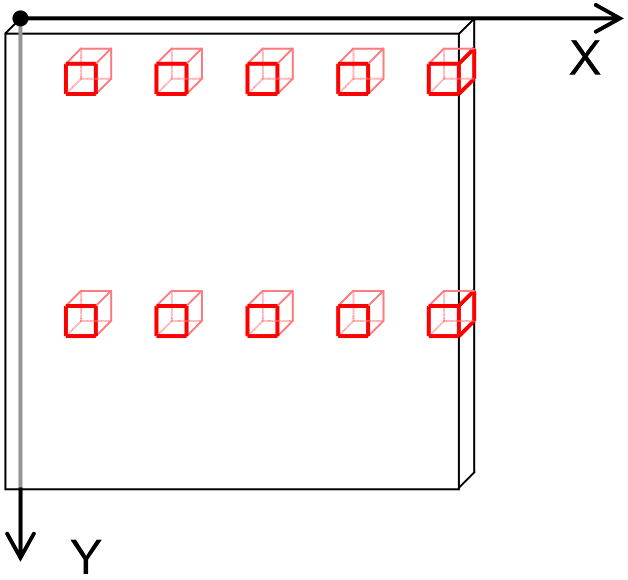


Array Access Patterns

- *rectangular region*

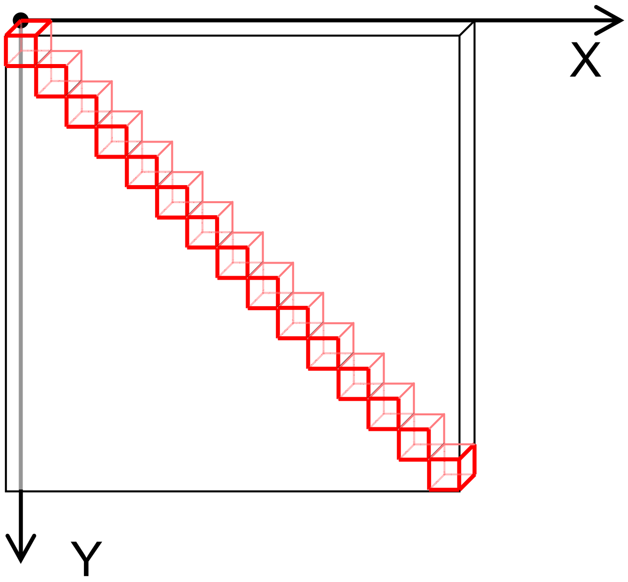


- *regular grid*

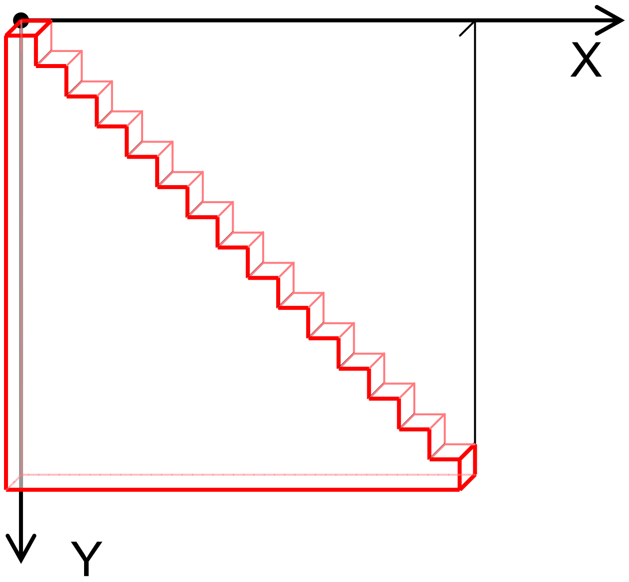


Array Access Patterns

- *diagonal*

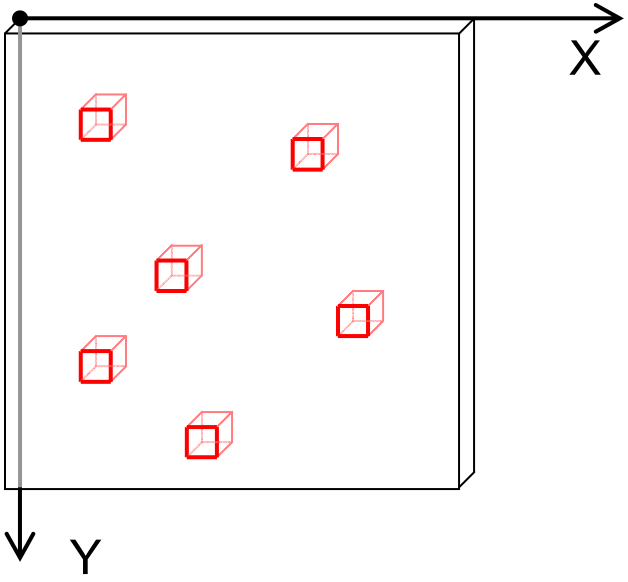


- *triangular*

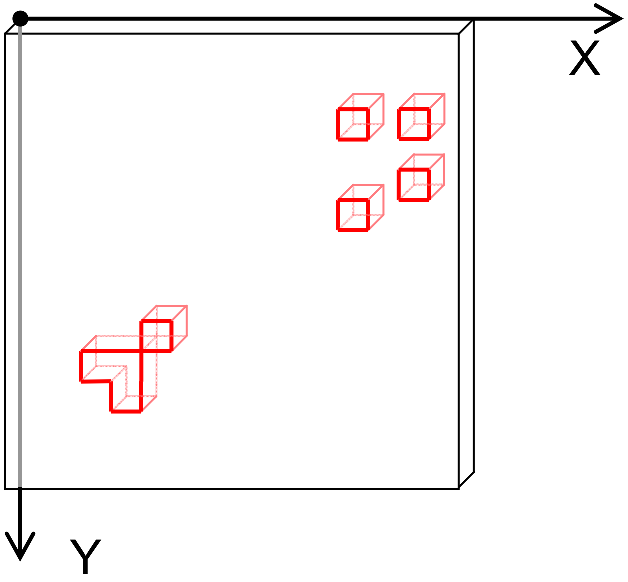


Array Access Patterns

- *random (uniform)*



- *random (clustered)*



Benchmark Variables

Data properties

- array shape and element type

Data storage options

- partitioning: linear / multidimensional
- chunk size
- nesting order of dimensions

Array query properties

- logical access pattern
- intra-array selectivity
- logical locality

Benchmark Variables

Data properties

- array shape and element type

Data storage options

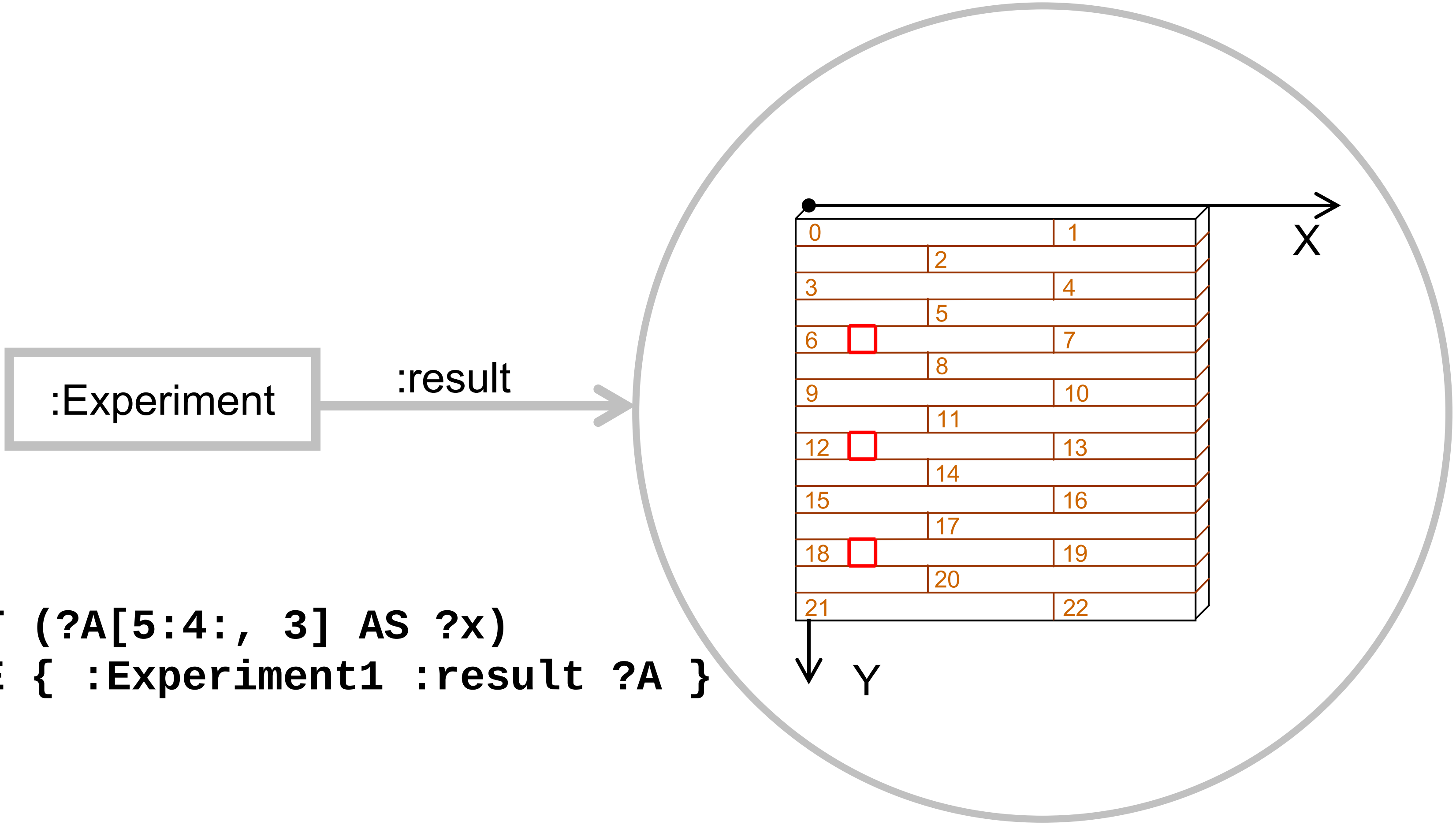
- partitioning: linear / multidimensional
- chunk size
- nesting order of dimensions

Array query properties

- logical access pattern
- intra-array selectivity
- logical locality

Query processing options

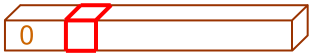
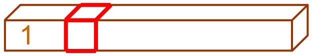
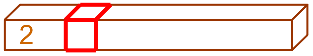
Retrieving the chunks

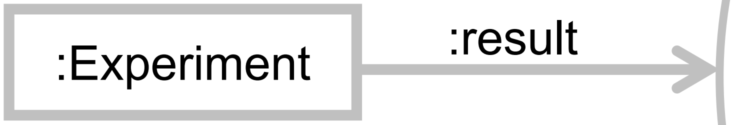


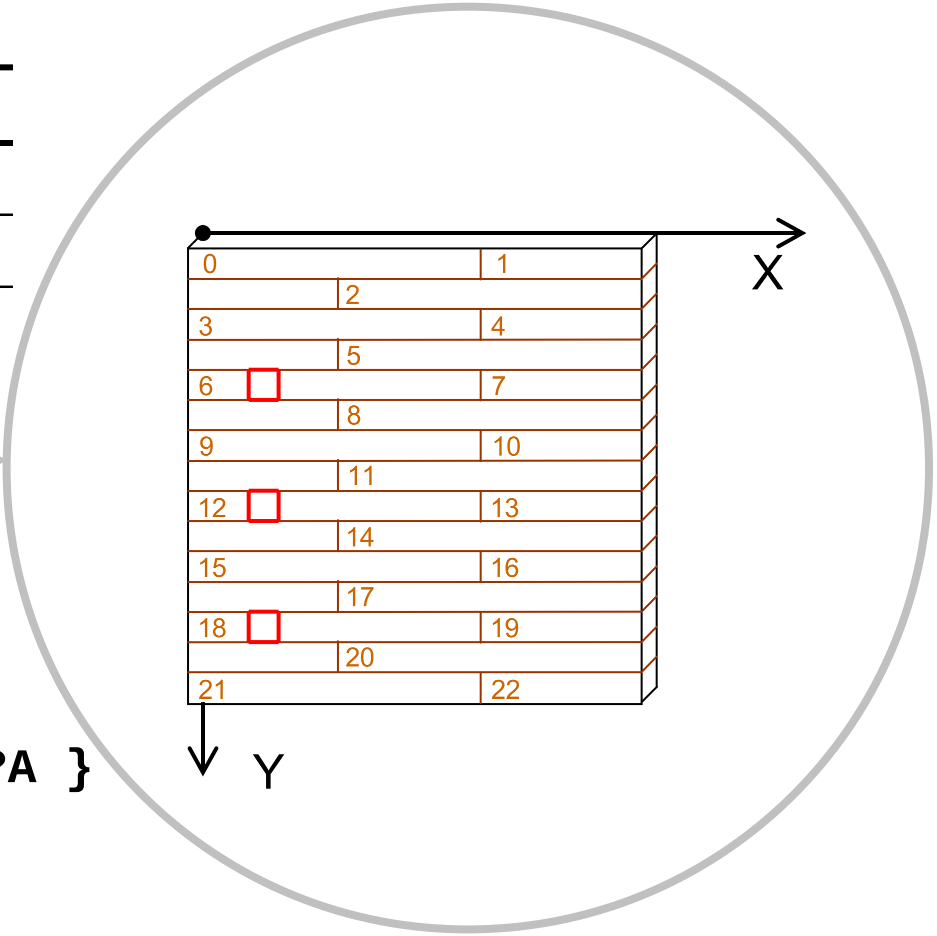
```
SELECT (?A[5:4:, 3] AS ?x)
 WHERE { :Experiment1 :result ?A }
```

Retrieving the chunks

ArrayChunks

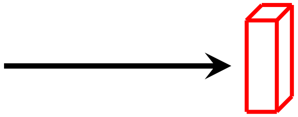
arrayid	chunkid	chunk
1	0	
1	1	
1	2	





```
SELECT (?A[5:4:, 3] AS ?x)
 WHERE { :Experiment1 :result ?A }
```

?x



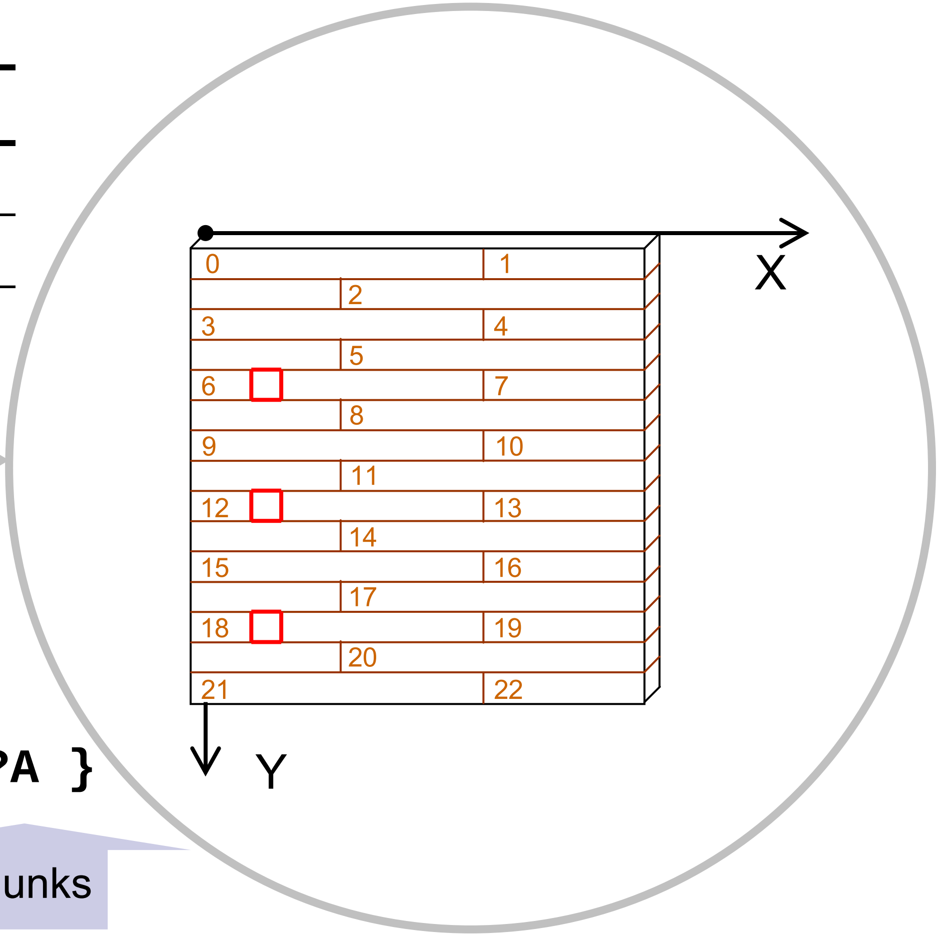
Retrieving the chunks

ArrayChunks

arrayid	chunkid	chunk
1	0	
1	1	
1	2	

:Experiment

:result



```
SELECT (?A[5:4:, 3] AS ?x)
 WHERE { :Experiment1 :result ?A }
```

chunks

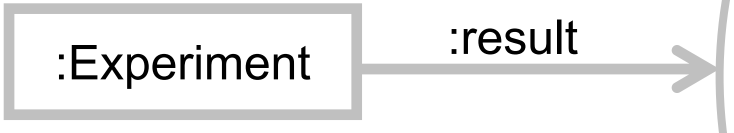
SQL

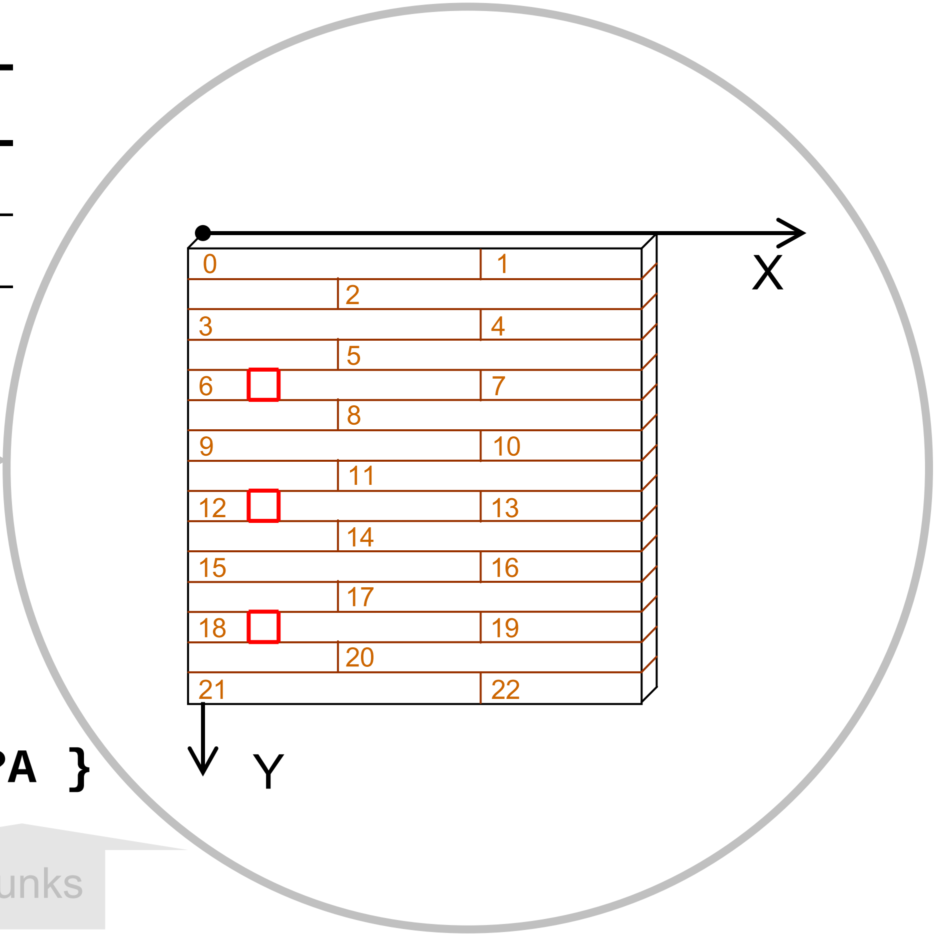
```
SELECT chunkid, chunk FROM ArrayChunks
 WHERE arrayid = 1 AND chunkid IN (6, 12, 18)
```

Retrieving the chunks

ArrayChunks

arrayid	chunkid	chunk
1	0	0
1	1	1
1	2	2





```
SELECT (?A[5:4:, 3] AS ?x)
 WHERE { :Experiment1 :result ?A }
```

chunks

SQL
```
SELECT chunkid, chunk FROM ArrayChunks
 WHERE arrayid = 1 AND chunkid IN (6, 12, 18)
```

chunks

SQL
```
SELECT chunkid, chunk FROM ArrayChunks
 WHERE arrayid = 1 AND chunkid >= 6 AND (chunkid - 6) mod 6 IN (0)
```

cycle length

starting point

pattern

Benchmark Variables

Data properties

- array shape and element type

Data storage options

- partitioning: linear / multidimensional
- chunk size
- nesting order of dimensions

Array query properties

- logical access pattern
- intra-array selectivity
- logical locality

Query processing options

- strategy: SPD / IN / hybrid
- buffer size

- Introduction
- RDF & SPARQL
- RDF with Arrays & Scientific SPARQL
- Scientific SPARQL Database Manager
- Extensible Array Storage
- Array Query Benchmark
  - benchmark definition
  - results
- Summary

Benchmark Application

Axis	Choice / Comparison
Data properties	
• array shape and element type	100 000 x 100 000 integer (40 GB)
Data storage options	
• partitioning: linear / multidimensional	(all)
• chunk size	10 000 elements / 100 x 100 elements
• nesting order of dimensions	row-by-row
Array query properties	
• logical access pattern	(all)
• intra-array selectivity	(varied)
• logical locality	(varied)
Query processing options	
• strategy: SPD / IN / hybrid	(all)
• buffer size	16, 256, 4096
Measured parameter:	query response time, s

Benchmark: Results

- **Long IN queries are answered faster than SPD queries by the RDBMS back-end**

(except for unselective queries with high physical locality)

Reason: scan vs. index lookups

Conclusion: use IN queries with a long buffer

Benchmark: Results

- **Long IN queries are answered faster than SPD queries by the RDBMS back-end**

(except for unselective queries with high physical locality)

Reason: scan vs. index lookups

Conclusion: use IN queries with a long buffer

- **For every array partitioning choice, there are best-case and worst-case workloads**

(however, multidimensional partitioning has fewer worst cases)

Reason: logical locality is better used

Conclusion: it is important to know the workload beforehand

When unknown, use isotropic multidimensional chunks

Benchmark: Results

- **Long IN queries are answered faster than SPD queries by the RDBMS back-end**

(except for unselective queries with high physical locality)

Reason: scan vs. index lookups

Conclusion: use IN queries with a long buffer

- **For every array partitioning choice, there are best-case and worst-case workloads**

(however, multidimensional partitioning has fewer worst cases)

Reason: logical locality is better used

Conclusion: it is important to know the workload beforehand

When unknown, use isotropic multidimensional chunks

- **It is possible to choose an optimal chunk size** (in terms minimizing gross data transfer), **when logical locality of the access pattern can be estimated**

UPPSALA UNIVERSITET

- Introduction
- RDF & SPARQL
- RDF with Arrays & Scientific SPARQL
- Scientific SPARQL Database Manager
- Extensible Array Storage
- Array Query Benchmark
- Summary

Summary

- RDF With Arrays
- Scientific SPARQL
  - array operations, functional views, second-order functions and closures
  - extensibility with foreign functions, flexible cost model and multidirectional computations
  - combining data and metadata search in the same query:
    - fewer round-trips to the server
    - self-contained queries
    - more freedom for the query optimizer

Summary

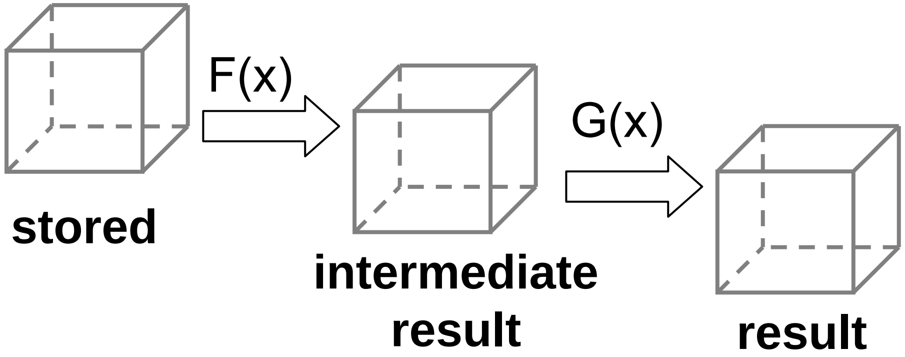
- Scientific SPARQL Implementation: SSDM
  - efficient main-memory representation of arrays and array operations
  - implementing a strict superset of SPARQL on top of an ObjectLog-based DBMS
  - extensible array storage
  - lazy retrieval of array data
  - call-in APIs, integration into Matlab

Summary

- Array Query Benchmark
  - implemented in SciSPARQL, easy to generalize for any other Array DBMS
  - used to compare different data storage and partitioning choices w.r.t. different array access patterns

Future Work

- Query Optimization
  - type inference
  - delegation of more complex tasks to the external (back-end) databases, according to their capabilities
  - automatic storage choices for the intermediate results, when computing array expressions



Future Work

- SSDM as SPARQL endpoint on the Web
  - extending SPARQL protocol to accommodate arrays
- Integration with other scientific computing environments (SciPy, R, ...)
- cloud-based deployment and parallelization
- Integration with OWL / SWRL reasoners

Related Work

- **RasDaMan** [Baumann:1994]

well-established array database, *RasQL*, *Array Algebra*, everything is arrays, integration with SciSPARQL

- **SciDB** [Cudré-Mauroux:2009, Brown:2010]

everything is arrays, some unique features (error bars, data versioning), slower than RasDaMan

- **SciHadoop** [Buck:2011]

general-purpose distributed storage and computing framework, a layer over Hadoop which is not made for arrays

- **SAGA** [Wang:2014]

lightweight (relational) database layer over a collection of binary (HDF5, NetCDF) array files, technically similar to one of SciSPARQL configurations, no published query language

THANK YOU!

The software, documentation, and examples are available at

http://user.it.uu.se/~udbl/SciSPARQL

This work is supported by

smart vortex

SSF

essence
THE e-SCIENCE COLLABORATION

UPPSALA UNIVERSITET
UMEÅ UNIVERSITET
LUND UNIVERSITY