Querying and Exchanging XML and RDF on the Web

WWW'12 Tutorial

Sherif Sakr (NICTA and UNSW)
Axel Polleres (Siemens AG)

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

Tutorial Overview

Session 1

- XQuery Overview – Sherif
- **SPARQL Overview – Axel**
- **XSPARQL: a combined language – Axel**

Session 2

- **XQuery implementations – Sherif**
- **SPARQL implementations – Sherif**
- **XSPARQL implementations – Axel**
- **(optional) Compression formats for XML+RDF: EXI+HDT – Sherif**

- **Q/A - Discussion**

XQuery

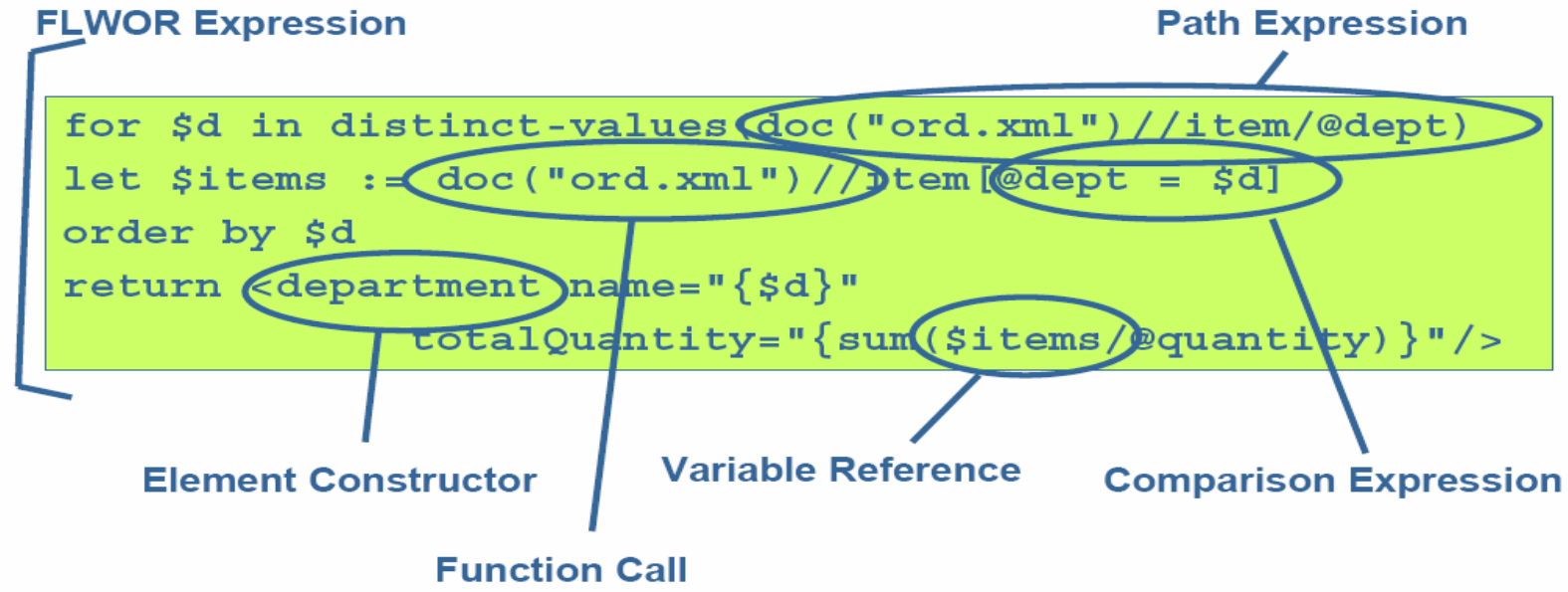
Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

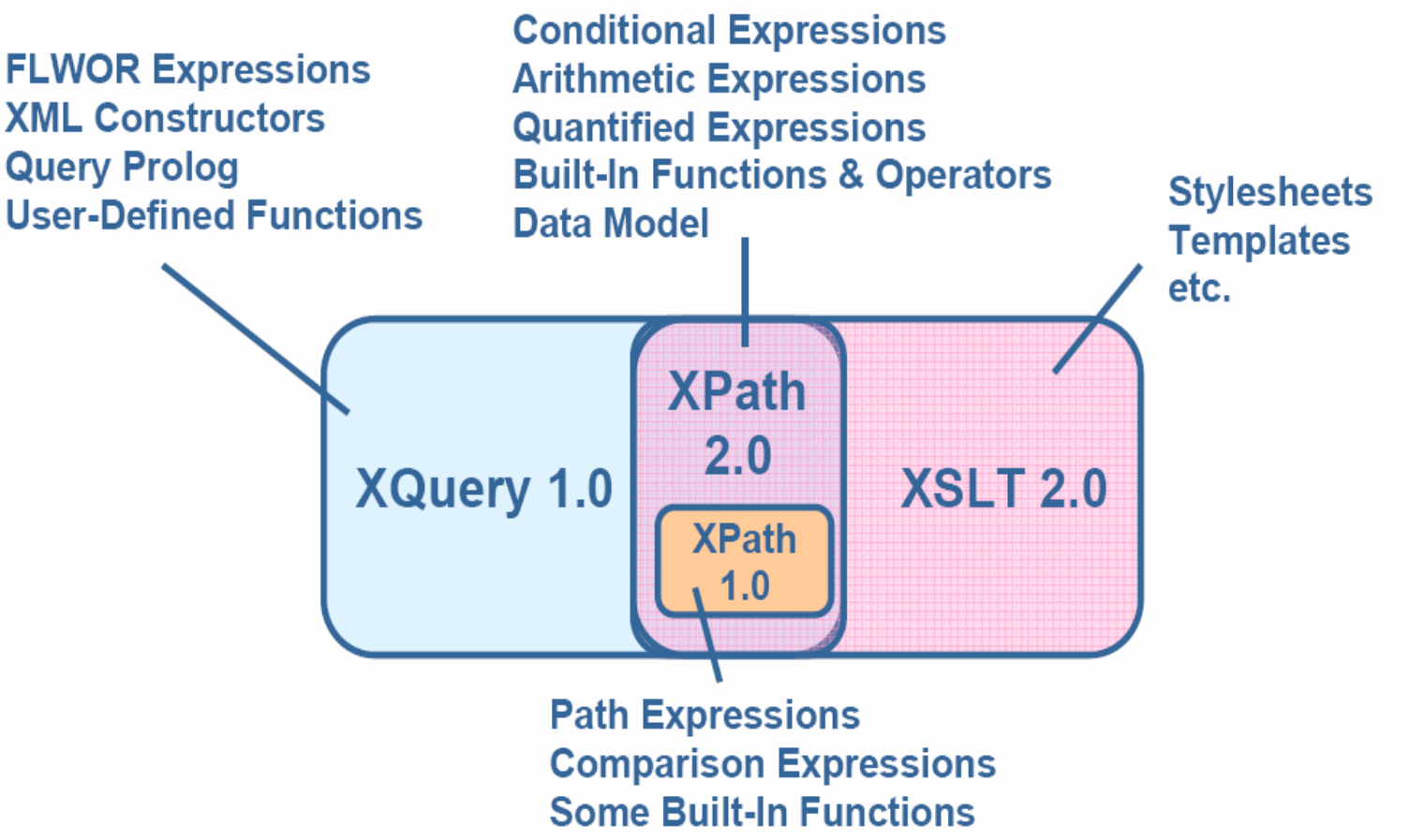
XQuery

NICTA

- XQuery is a **declarative** language in which a query is represented as an **expression**.
- XQuery expressions can be nested with full generality.



XQuery, XSLT and XPath



XQuery Data Model

- The XQuery language is designed to operate over ordered, ***finite sequences of items*** as its principal data type.
- The evaluation of any XQuery expression yields an ordered sequence of *n >= 0* items.
- These **items** can be:
  - Atomic values (integers, strings, ..., etc)
  - Unranked XML tree nodes.

Items and Ordered Sequences

- A sequence of ***n* items X_i** is written in parentheses and comma-separated form

  $(X_1,X_2,\ldots,X_n)$

- A single item ***X*** and the singleton sequence ***(X)*** are equivalent.
- Sequences ***can not*** contain other sequences (nested sequences are implicitly **flattened**)

  (0, (), (1, 2), (3)) = (0, 1, 2, 3)

  (0,1) ≠ (1,0)

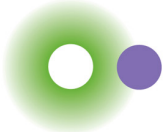
- Sequences ***can*** contain ***duplicates***

  (0, 1, 1, 2)

- Sequences may be ***heterogeneous***

  (42, "foo", 4.2, <a>)

XQuery = ½ Programming Language + ½ Query Language



- **Programming language** features:
  - Explicit iteration and variable bindings (for, let, …).
  - Recursive, user-defined functions.
  - Regular expressions, strong [static] typing.
  - Ordered sequences (much like lists or arrays).

- **Query language** features:
  - Filtering.
  - Grouping.
  - Joins.

XQuery

- **A strongly-typed, Turing-complete XML manipulation language**
  - Attempts to do static type checking against XML Schema
  - Based on an object model derived from Schema
- **Unlike SQL, fully compositional, highly orthogonal:**
  - Inputs & outputs collections (sequences or bags) of XML nodes
  - Anywhere a particular type of object may be used, may use the results of a query of the same type
  - Designed mostly by DB and functional language people
- **Attempts to satisfy the needs of data management and document management**
  - The database-style core is mostly complete (even has support for NULLs in XML!!)
  - The document keyword querying features are still in the works – shows in the order-preserving default model

Some Uses for XQuery

- Extracting information from a database for use in web service.
- Generating summary reports on data stored in XML database.
- Searching textual documents on the web for relevant information.
- Transforming XML data to XHTML format to be published on the web.
- Pulling data from different databases to be used for application integration.
- Splitting up an XML document into multiple XML documents.

XQuery Expressions

- Path expressions.
- FLWOR expressions.
- Expressions involving operators and functions .
- Conditional expressions.
- Quantified expressions.
- List constructors.
- Element constructors.
- Expressions that test or modify datatypes

Path Expression

- In a sense, the ***traversal*** or ***navigation*** of trees of XML nodes lies at the core of every XML query language.
- XQuery embeds **XPath** as its tree navigation sub-language.
- Every XPathexpression is a correct XQuery expression.
- Since navigation expressions extract (potentially huge volumes of) nodes from input XML documents, efficient XPath implementation is a prime concern to any implementation of an XQuery processors.

Path Expression

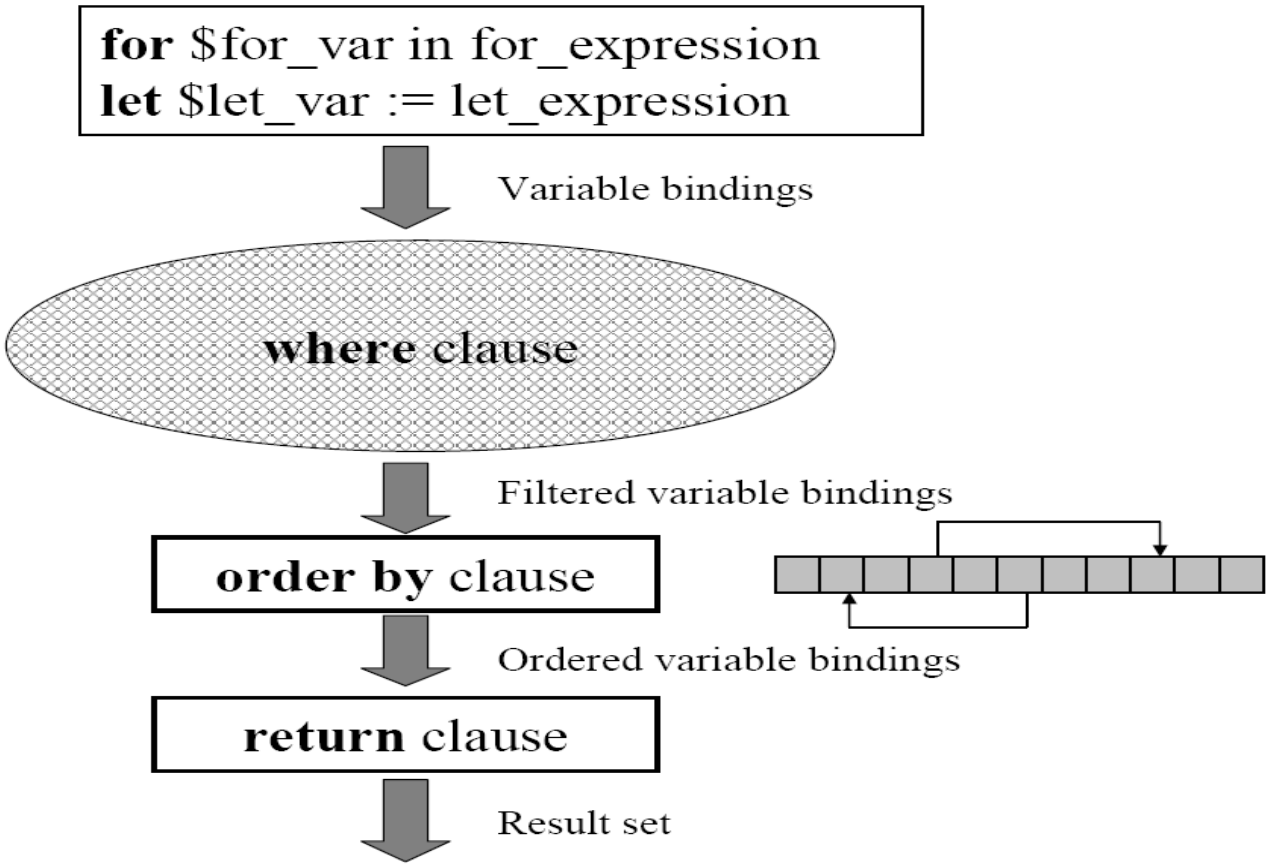
- Each path consists of one or more **steps**, syntactically separated by /

$$s_0/s_1/\ldots/s_n$$

- Each step acts like an operator that, given a **sequence of nodes** (the context set), evaluates to a **sequence of nodes**.
- XPath defines the result of each path expression to be ***duplicate free*** and ***sorted in document order***.

FLWOR Expression

- A FLWOR expression binds some expressions, applies a predicate, and constructs a new ordered result.



FLWOR Expression

- Has an analogous form to SQL's
  SELECT..FROM..WHERE..GROUPBY..ORDER BY
- The model: bind nodes (or node sets) to variables; operate over each legal combination of bindings; produce a set of nodes
- "FLWOR" statement:

for {iterators that bind variables}

let {collections}

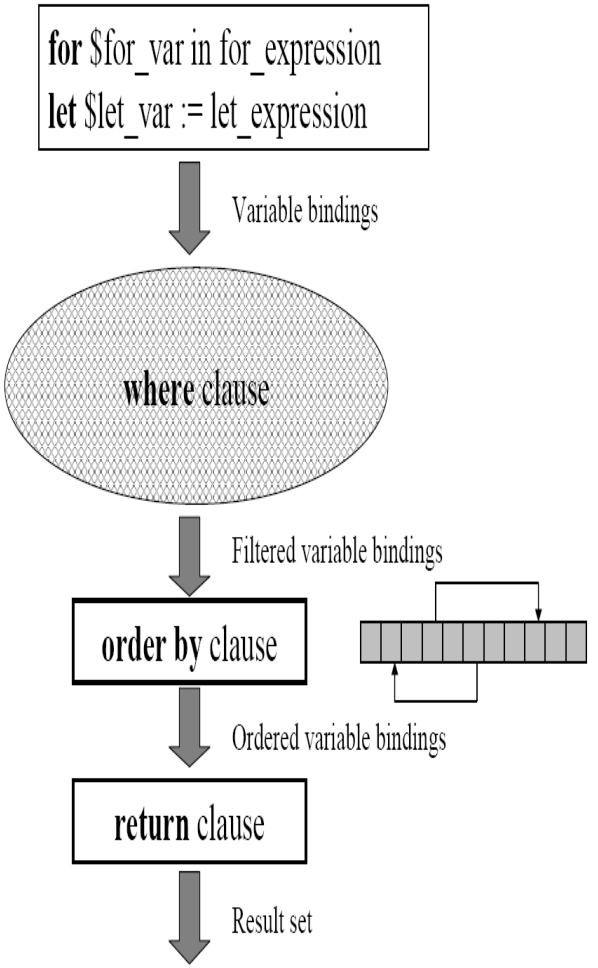
where {conditions}

order by {order-conditions}

return {output constructor}

FLWOR Expression

- The **for** construct **successively binds** each item of an expression (**expr**) to a variable (**var**), generating a so-called **tuple stream**.
- This tuple stream is then **filtered** by the **where** clause, retaining some tuples and discarding others.
- The **return** clause is evaluated **once for every tuple** still in the stream.
- The result of the expression is an **ordered** sequence containing the **concatenated** results of these evaluations.



FLWOR Expression

NICTA

Iteration

```
for $x in (3,2,1)
return ($x,"*")
```
$\Rightarrow$ `(3,"*",2,"*",1,"*")`

```
for $x in (3,2,1)
return $x,"*"
```
$\Rightarrow$ `(3,2,1,"*")`

```
for $x in (3,2,1)
return for $y in ("a","b")
       return ($x,$y)
```
$\Rightarrow$ `(3,"a",3,"b",`
`2,"a",2,"b",`
`1,"a",1,"b")`

Inner Joins


```
for $book in document("bib.xml")//book,
      $quote in document("quotes.xml")//listing
where $book/isbn = $quote/isbn
return
<book>
                { $book/title }
                { $quote/price }
</book>
```

Outer Joins


```
for $book in document("bib.xml")//book
return
<book>
        { $book/title }
        {
for $review in document("reviews.xml")//review
where $book/isbn = $review/isbn
return $review/rating
         }
</book>
```

Aggregation - Grouping

- **for** iterates on a sequence, binds a variable to each node.
- **let** binds a variable to a sequence as a whole.
- Together, they are used for representing aggregation and grouping expressions.

```
for $book in document("bib.xml")//book
let  $a := $book/author
where contains($book/publisher, "Addison-Wesley”)
return
    <book>
        {
          $book/title,
<count> Number of authors: { count($a) } </count>
        }
    </book>
```

XQuery Operators and Functions

- Infix and prefix operators (+, -, *,…).
- Parenthesized expressions.
- Arithmetic and logical operators.
- Collection operators UNION, INTERSECT and EXCEPT.
- Infix operators BEFORE and AFTER (<< , >>).
- User functions can be defined in XQuery.

Node Comparisons

NICTA

Node comparison

Node comparisons based on *identity* and *document order*:

e_1 `is` e_2	nodes $e_{1,2}$ identical?
e_1 `<<` e_2	node e_1 before e_2?
e_1 `>>` e_2	node e_1 after e_2?

Node comparison examples

```
<x>42</x> eq <x>42</x>          ⇒  true()
<x>42</x> is <x>42</x>          ⇒  false()
  root(e1) is root(e2)          ⇒  nodes e1,2 in same tree?
let $a := <x><y/></x>
return $a << $a/y               ⇒  true()
```

XQuery Comparisons

Value	comparing single values Untyped data => string	**eq, ne, lt, le, gt, ge**
General	Existential quantification Untyped data => coerced to other operand's type	**=, !=, <=, <, >, >=**
Node	for testing identity of single nodes	**is, isnot**
Order	testing relative position of one node vs. another (in document order)	**<<, >>**

Logical Expression

- Logical Operators “**and**” and “**or**”.
- The concept of **Effective Boolean Value(EBV)** is key to evaluating logical expressions.
  - **EBV** of an empty sequence is ***false***.
  - **EBV** of a non-empty sequence containing only nodes is ***true***.
  - **EBV** is the value of the expression if the expression evaluates to a value of type ***xs:boolean***.
  - **EBV** is an error in every other case.
- e.g: The expression “() and true()” evaluates to false(since () is false)

XQuery: Built-in Functions

- Over 100 functions built into XQuery.
- **String-related**
  - substring, contains, concat,…
- **Date-related**
  - current-date, month-from-date,…
- **Number-related**
  - round, avg, sum, …
- **Sequence-related**
  - index-of, distinct-values,…
- **Node-related**
  - data, empty,…
- **Document-related**
  - doc, collection, …
- **Error Handling**
  - error, exactly-one, …
- ….

XQuery: User-Defined Functions

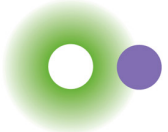
- XQuery expressions can contain **user-defined functions** which ***encapsulate*** query details.
- User-defined functions may be collected into **modules** and then **'import'ed** by a query.

Declaration of n-ary function f with body e

declare function f($\$p_1$ as t_1, ...,$\$p_n$ as t_n) as t_0 { e }

- If t_i is omitted, it defaults to item()*.
- The pair (f, n) is required to be unique (overloading).
- Atomization is applied to the i-th parameter if t_i is atomic.

User-Defined Functions Example



Reverse a sequence

Reversing a sequence does not inspect the sequence's items in any way:

```
declare function reverse($seq)
{ for $i at $p in $seq
  order by $p descending
  return $i
};

reverse((42,"a",<b/>,doc("foo.xml")))
```

Conditional Expression

- **Syntax:**

  if (expr1) **then** expr2 **else** expr3
- if **EBV** of *expr1*is true, the conditional expression evaluates to the value of *expr2*, else it evaluates to the value of *expr3.*
- Parentheses around if expression (*expr1*) are required.
- else is always required but it can be just else ().
- Useful when structure of information returned depends on a condition.
- Can be nested and used anywhere a value is expected.

```
if($book/@year <1980 )
then<old>{$x/title}</old>
else<new>{$x/title}</new>
```

Conditional Expression

- Used as an alternative way of writing the FLWOR expressions.

```
FLWOR:for $a in document("bib.xml")//article
        where $a/year < 1996
                return  $a
```

```
Conditional:for $a in document("bib.xml")//article
return
If ($a/year < 1996)
then$a
else()
```

Quantified Expressions

- **Syntax:**

[some | every]*$var* in *expr* satisfies *test_expr*

- Quantified expressions evaluate to a boolean value.
- Evaluation:
  - *$var* is bound to each of the items in the sequence resulting from *expr*.
  - For each binding, the *test_expr* is evaluated.
  - In case of
    - Existential quantification ("some"), if at least one evaluation of *test_expr* evaluates "**true**", the entire expression evaluates "**true**".
    - Universal quantification ("every"), all evaluations of *test_exp* rmust result in an **EBV** of "**true**" for the expression to return "**true**".

Quantified Expressions

- Existential Quantification
  - Give me all books where “Sailing” appear at least once in the same paragraph.

```
for $b in document("bib.xml")//book
where some $p in $b//para satisfies(contains($p,”Sailing”))
return $b/title
```

Quantified Expressions

- Universal Quantification
  - Give me all books where “Sailing” appears in every paragraph.

```
for $b in document("bib.xml")//book
where every $p in $b//para satisfies(contains($p,”Sailing”))
return $b/title
```

XQuery List Constructors

- A list may be constructed by enclosing zero or more expressions in square brackets, separated by commas.
- For example, [$x, $y, $z] denotes a list containing three members represented by variables.
- [ ] denotes an empty list.

XQuery Element Constructors

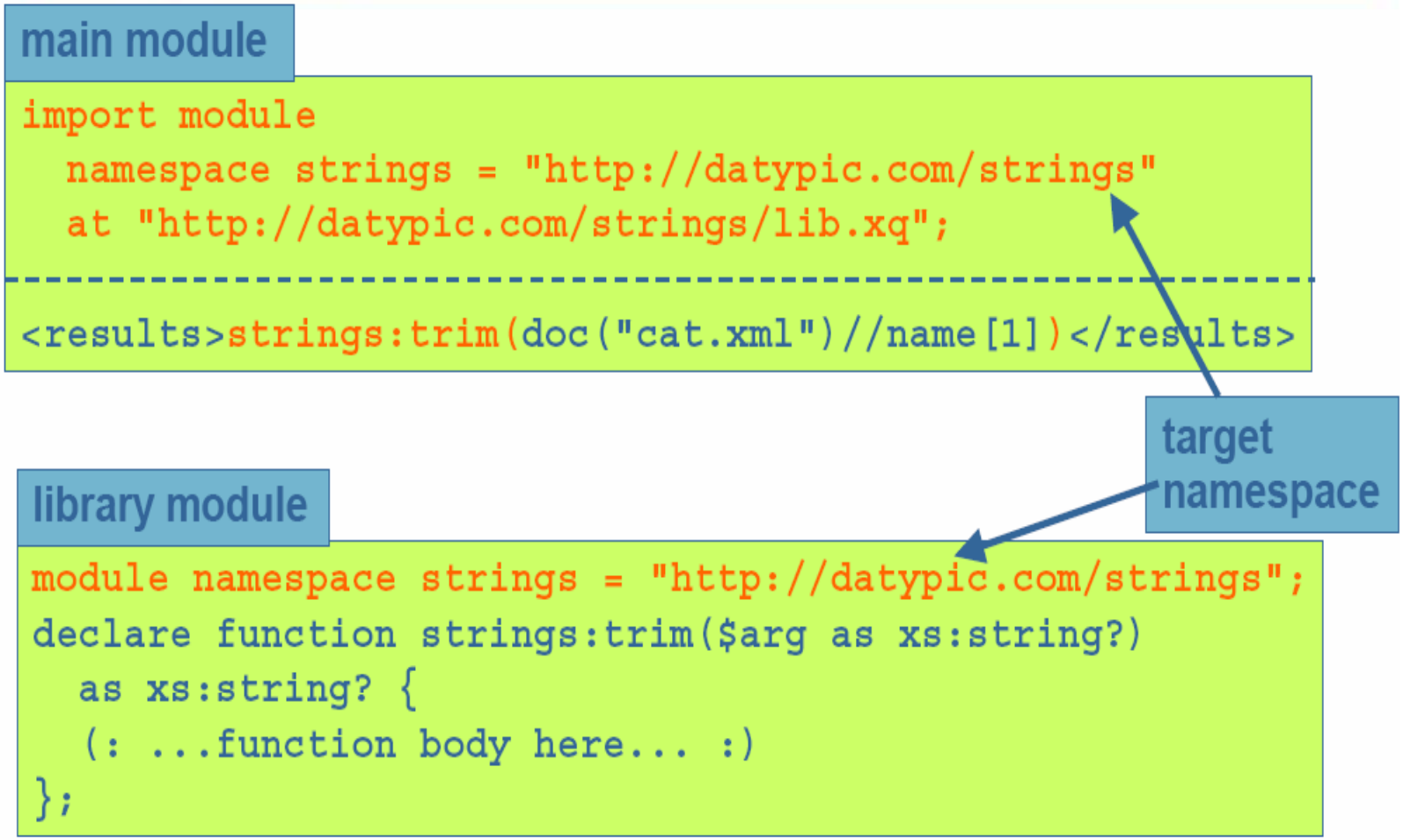
- XML **tree fragments** may be constructed "on the fly" in queries:

```
for $i in doc("auction.xml") /site/regions/australia/item
return
<item>
<name>{$i/name/text()}</name>
<description >{$i/description}</description>
</item>
```

Library Modules

- Separate XQuery documents that contain function definitions.
- Why?
  - Reusing functions among many queries.
  - Defining standard libraries that can be distributed to a variety of query users.
  - Organizing and reducing the size of query modules.

Library Modules



Global Variables

- Declared and bound in the query prolog and used through the query.
- Can be
  - Referenced in a function that is declared in that module.
  - Referenced in other modules that import the module.

```
declare variable $maxNumItems := 3;
declare variable $ordDoc := doc("ord.xml");

for $item in
  $ordDoc//item[position() <= $maxNumItems]
return $item
```

Advantages of XQuery

- It is easy to learn if knowledge of SQL and XPath is present.
- When queries are written in XQuery, they require less code than queries written in XSLT do.
- XQuery can be used as a strongly typed language when the XML data is typed, which can improve the performance of the query by avoiding implicit type casts and provide type assurances that can be used when performing query optimization.
- XQuery can be used as a weakly typed language for untyped data to provide high usability.
- Because XQuery requires less code to perform a query than does XSLT, maintenance costs are lower.
- XQuery is supported by major database vendors.

Capabilities of XQuery

- Selecting information based on specific criteria
- Filtering out unwanted information
- Searching for information within a document or set of documents
- Joining data from multiple documents or collections of documents
- Sorting, grouping, and aggregating data
- Transforming and restructuring XML data into another XML vocabulary or structure
- Performing arithmetic calculations on numbers and dates
- Manipulating strings to reformat text

Uses of XQuery

- Extracting information from a relational database for use in a web service
- Generating reports on data stored in a database for presentation on the Web as XHTML
- Searching textual documents in a native XML database and presenting the results
- Pulling data from databases or packaged software and transforming it for application integration
- Combining content from traditionally non-XML sources to implement content management and delivery
- Ad hoc querying of standalone XML documents for the purposes of testing or research

Tutorial Overview

Session 1

- XQuery Overview – Sherif
- SPARQL Overview – Axel
- XSPARQL: a combined language – Axel

Session 2

- **XQuery implementations – Sherif**
- **SPARQL implementations – Sherif**
- **XSPARQL implementations – Axel**
- **(optional) Compression formats for XML+RDF: EXI+HDT – Sherif**

- **Q/A - Discussion**

XQuery Implementations

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

XQuery Processors

- **Native XML/XQuery Processors:** The implementations of this approach make use of storage models, indexing and querying mechanisms that have been designed specifically for XML data.
- **Streaming XQuery Processors:** The implementations of this approach receive the XML data in the form of continuous streams of tokens and apply on the fly the query processing functionalities over them.
- **Relational XQuery Processors:** The implementations of this approach makes use of the relational indexing and querying mechanisms for querying the source XML data..

Timber

- A native XML database which is able to store and query XML documents.
- It uses the tree-based query algebra (TAX) which considers collections of ordered labelled trees as the basic unit of manipulation
- Each operator takes one or more sets of trees as input and produces a set of trees as output.
- The query plan tree is evaluated by pipelining one operator at a time.
- The query execution heavily depends on structural joins.

http://www.eecs.umich.edu/db/timber/

Natix

- Another native XML database which clusters subtrees of XML documents into physical records of limited size.
- The XML data tree is partitioned into small subtrees and each subtree is stored into a data page.
- The query execution engine consists of an iterator-based implementation of algebraic operators which process ordered sequences of tuples.

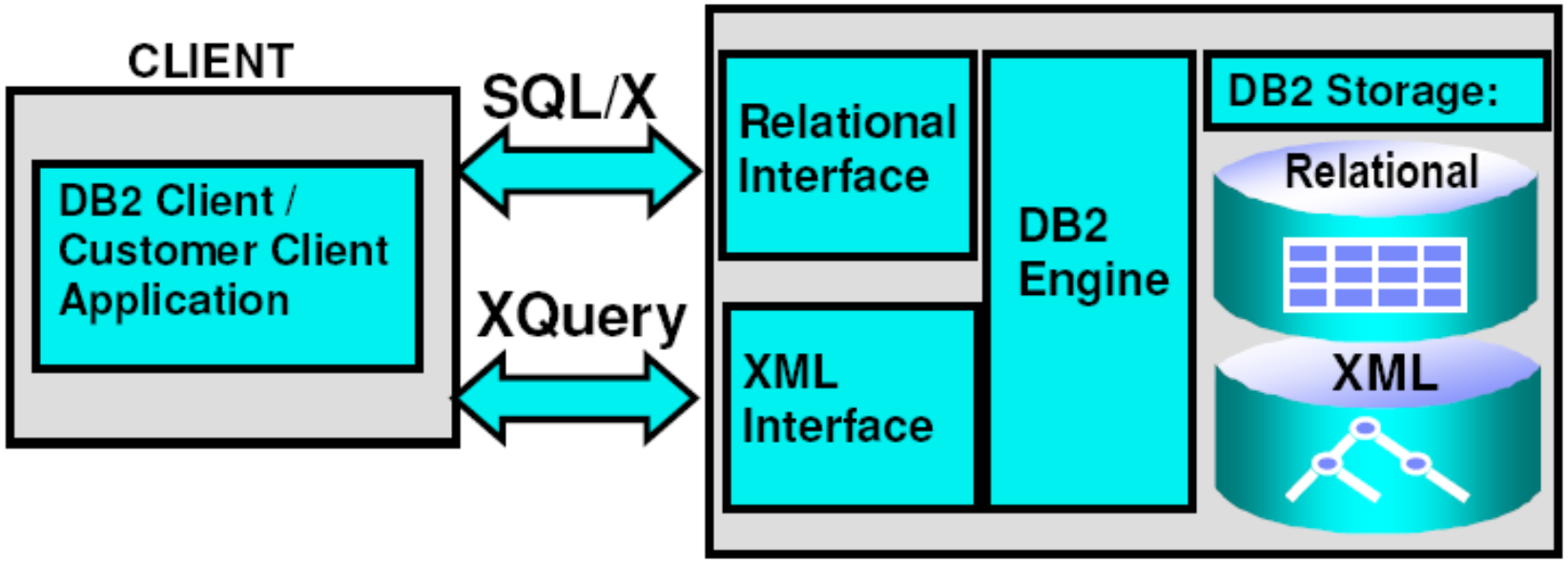
Thorsten Fiebig, Guido Moerkotte: Algebraic XML Construction and its Optimization in Natix. World Wide Web 4(3) , 2001

DB2/System RX

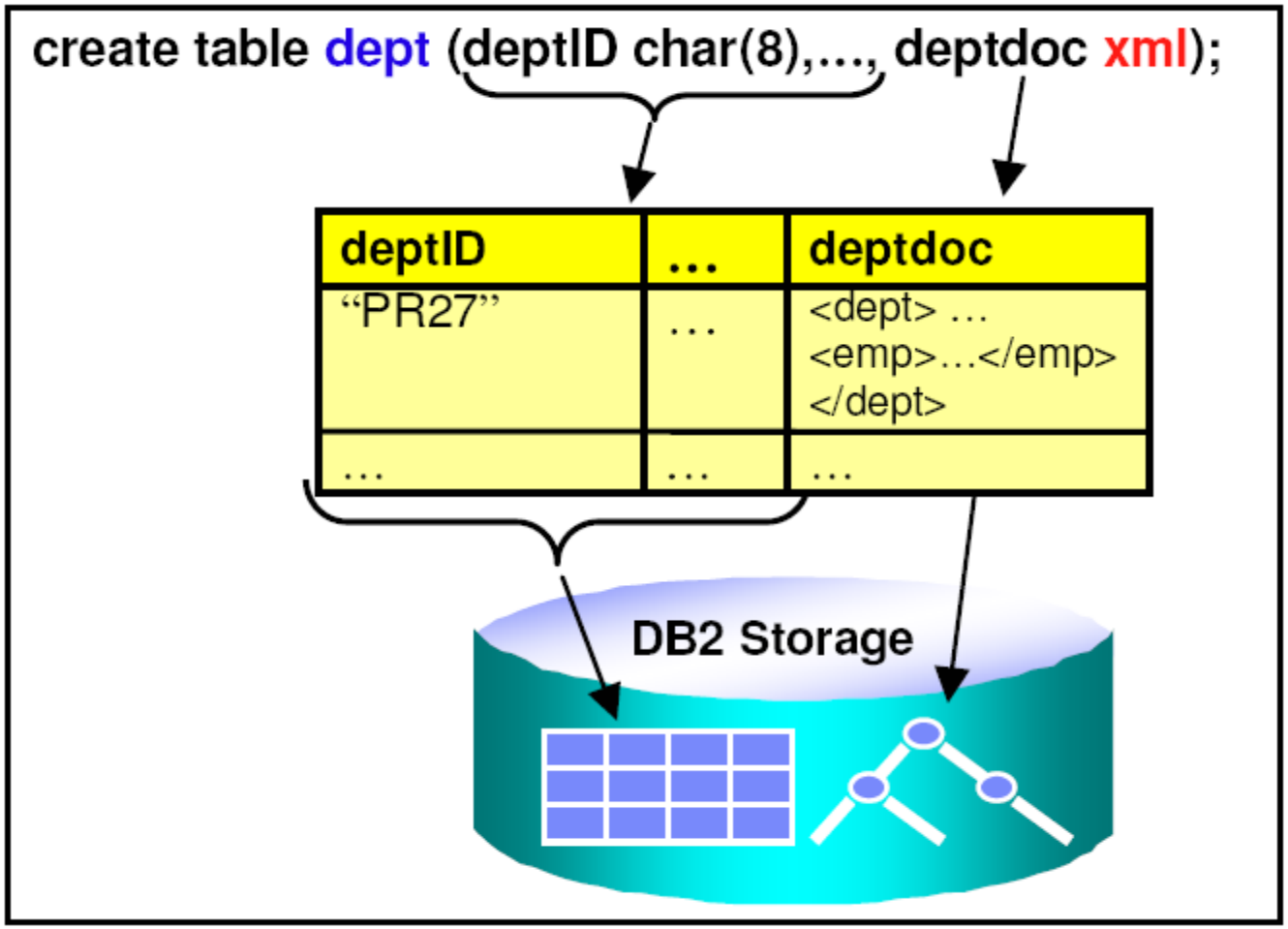
- A native XML support and XQuery implementation in IBM DB2.
- It uses a new XML data type which can be used like any other SQL type.
- A column of type XML can hold one well-formed XML document for every row of the table.
- Relational and XML data are stored differently while the relational columns are stored in traditional row structures, the XML data is stored in hierarchical structures.
- It uses Structural Indexes, Value Indexes and Full-text Indexes.
- Supports interfaces for both SQL/XML and XQuery.

Matthias Nicola and Bert Van der Linden. Native XML Support in DB2 Universal Database. VLDB, 2005.

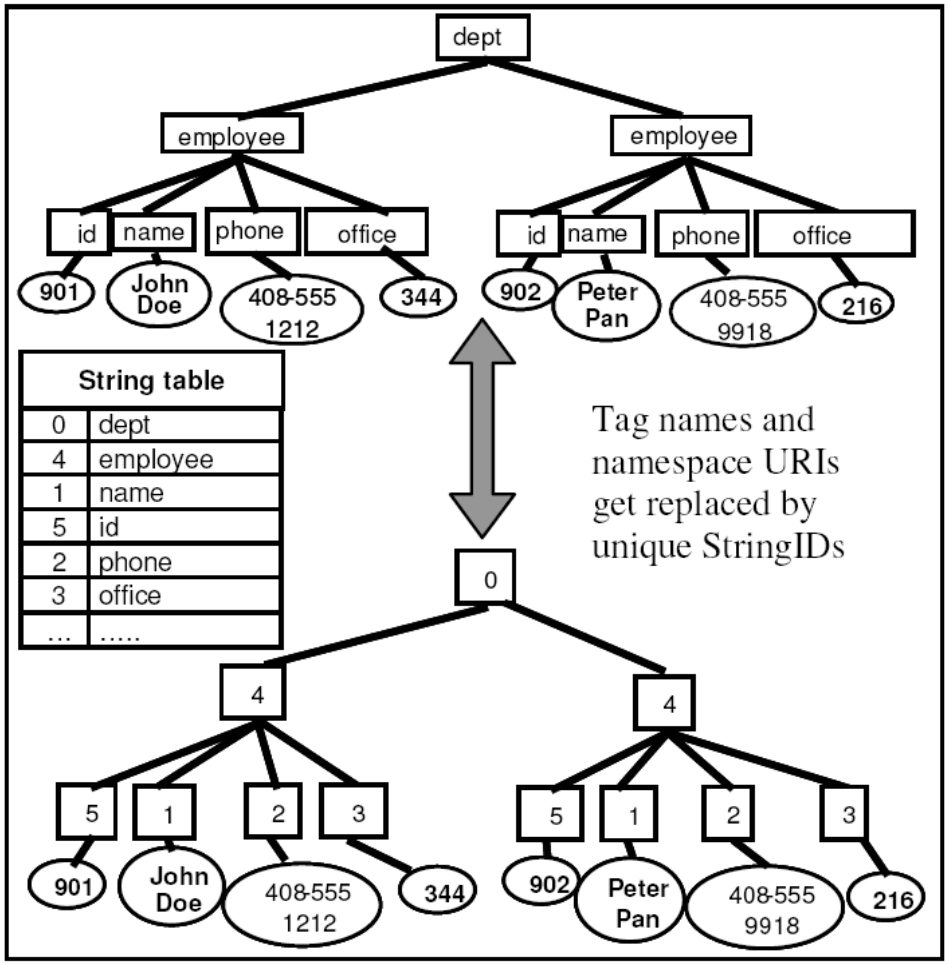
DB2/System RX



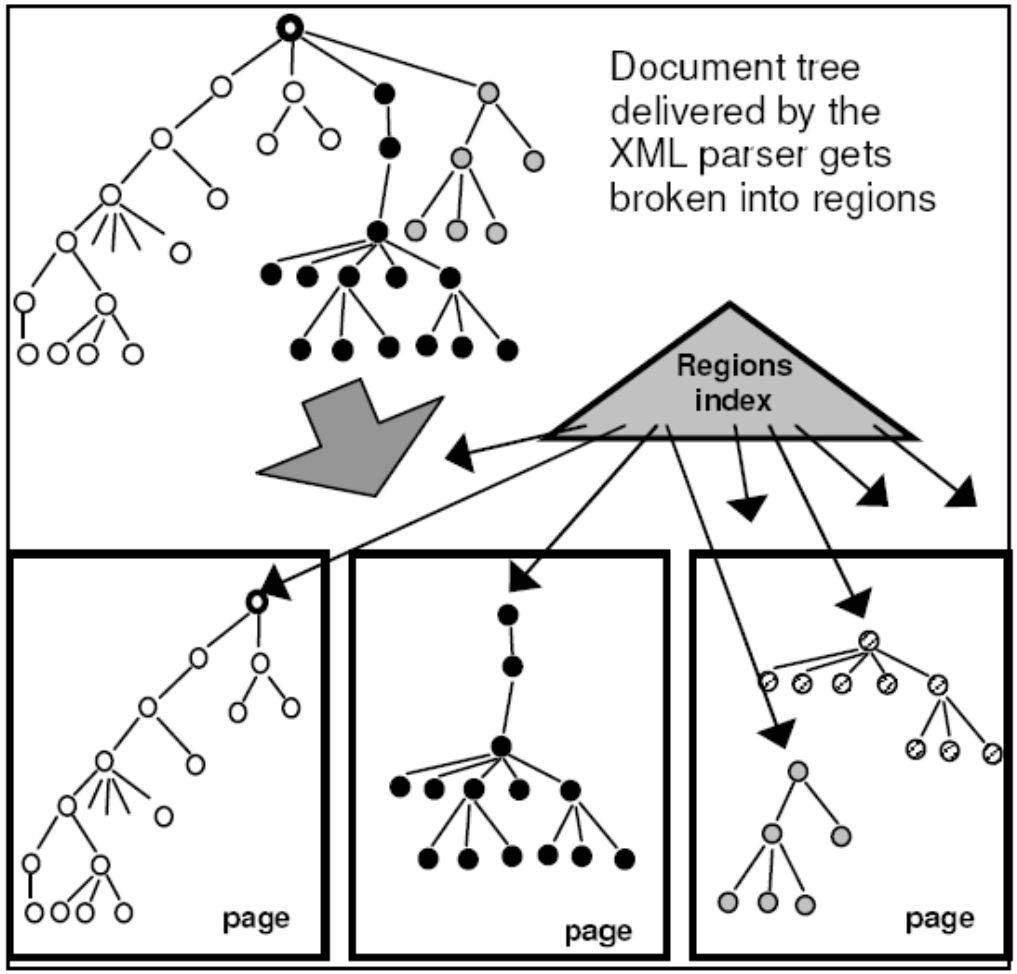
DB2/System RX



DB2/System RX



DB2/System RX

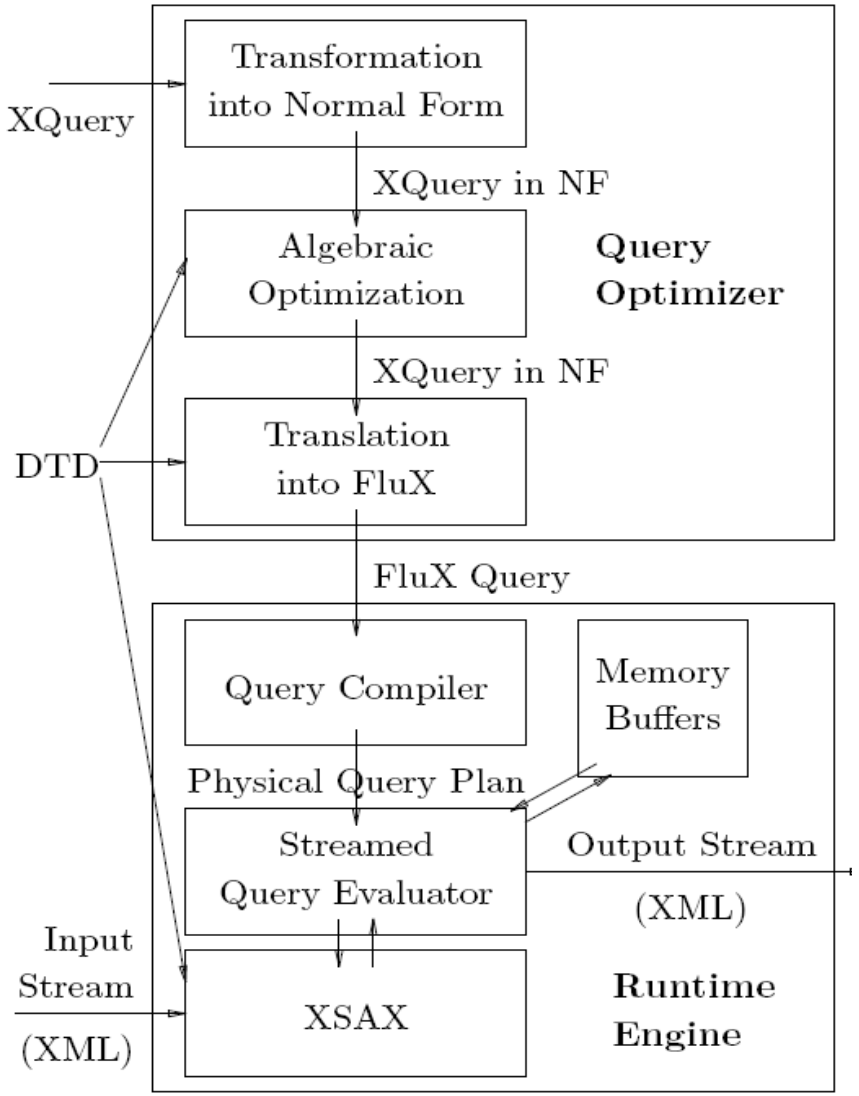


FluXQuery

- A streaming XQuery processor that is based on an internal query language called FluX which extends the main structures of XQuery by introducing a construct for event-based query processing.
- The buffer size is then optimized by analyzing the schema constraints of the XML document as well as the query syntax.
- FluX queries are transformed into physical query plans which are translated into executable JAVA code or interpreted and executed using the Streamed Query Evaluator.

Christoph Koch, Stefanie Scherzinger, Nicole Schweikardt, and Bernhard Stegmaier. FluXQuery: An Optimizing XQuery Processor for Streaming XML Data. VLDB, 2004.

FluXQuery

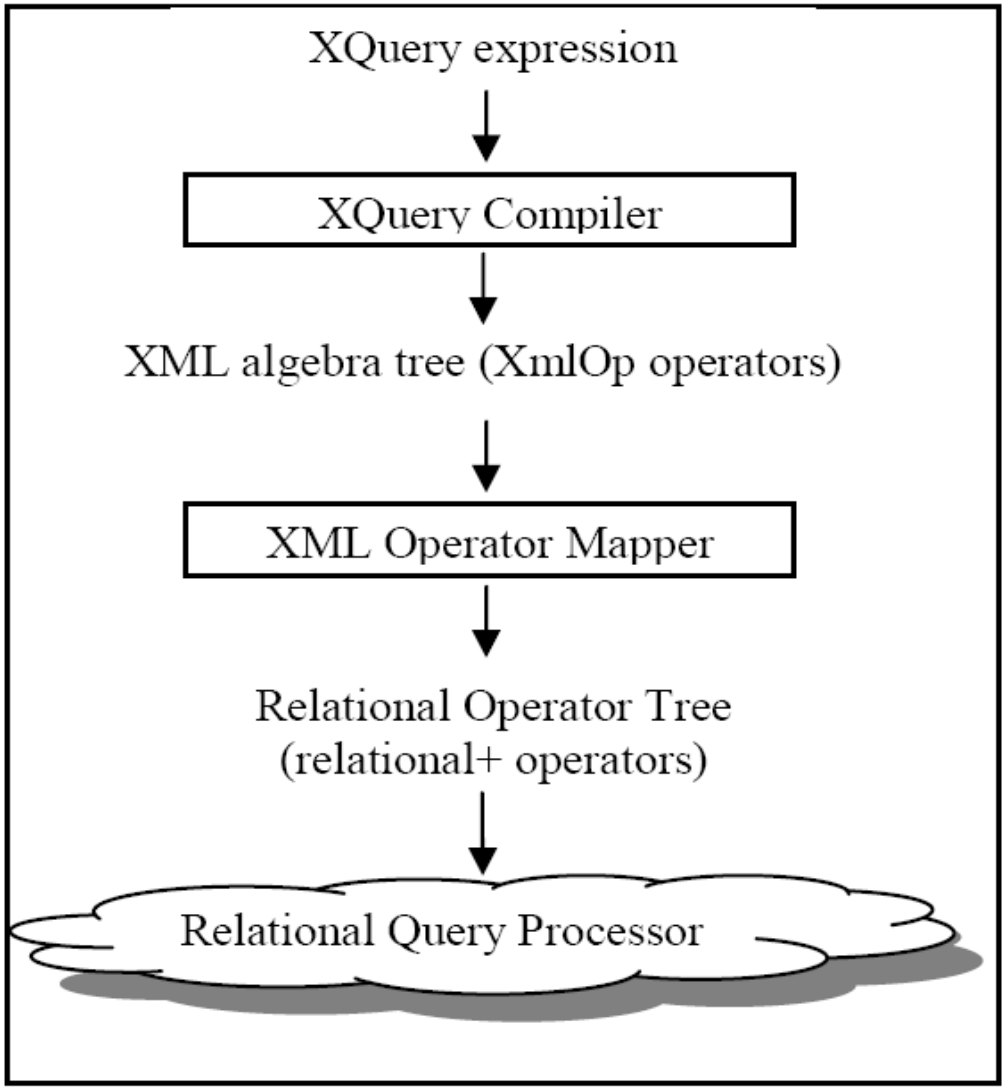


SQL Server

- Introduces native storage for XML data as a new, rich data type called XML. A table may contain one or more columns of type XML wherein both rooted XML trees and XML fragments can be stored.
- The hierarchical nature of the XML data is modelled as parent-child relationship using a node labelling scheme.
- XQuery expressions are compiled into a query trees that can be optimized and executed by the relational query processor.
- It uses a set of relational operators which is extended with additional operators for XML processing named as relational+ operators.
- The XML operator mapper recursively traverses the XML algebra tree and for each operator a relational operator sub-tree is generated and inserted into the overall relational operator tree for the XQuery expression.

Shankar Pal et al. XQuery Implementation in a Relational Database System. VLDB, 2005.

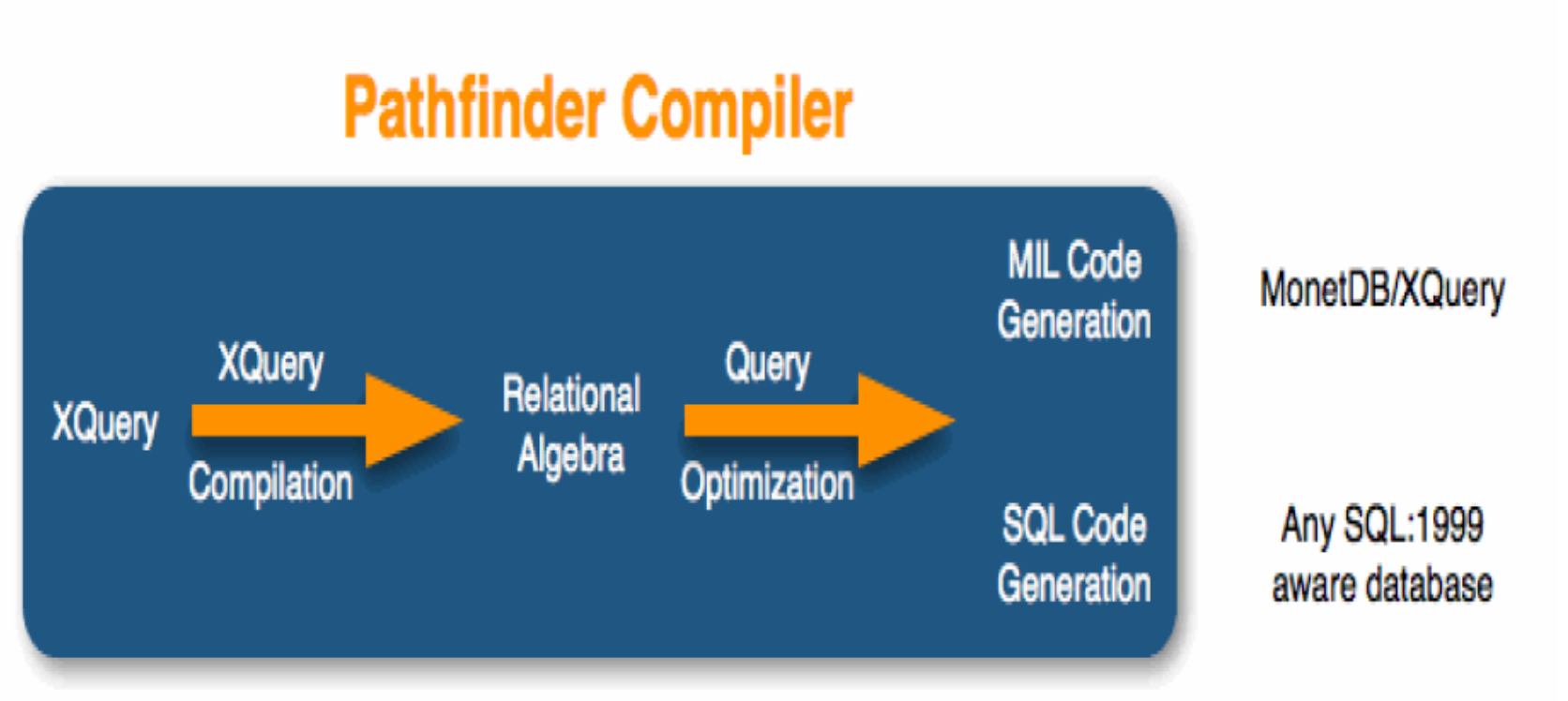
SQL Server



Pathfinder

NICTA

- A purely relational XQuery processor

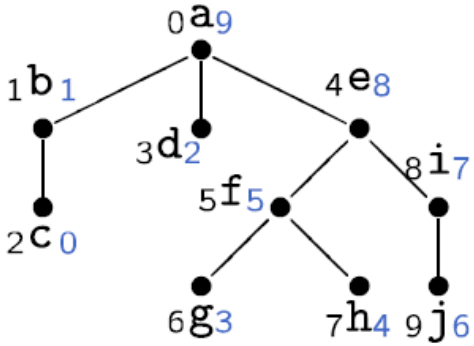


http://pathfinder-xquery.org/

Pathfinder

NICTA

Relational XML Encoding



pre	post
0	9
1	1
2	0
3	2
4	8
5	5
6	3
7	4
8	7
9	6

pre	kind
0	*elem*
1	*elem*
2	*text*
3	*text*
4	*elem*
5	*elem*
6	*elem*
7	*elem*
8	*elem*
9	*text*

pre	name
0	a
1	b
2	□
3	□
4	e
5	f
6	g
7	h
8	i
9	□

pre	type
0	t1
1	xs:string
2	□
3	□
4	t2
5	t3
6	xs:string
7	xs:string
8	xs:string
9	□

Torsten Grust: Accelerating XPath location steps. SIGMOD, 2002

Pathfinder

NICTA

Relational Algebra

π	column projection, renaming
σ	row selection
$\dot{\cup}$, $\setminus$	disjoint union, difference
δ	duplicate elimination
$\bowtie$	equi-join
$\times$	Cartesian product
ρ	row numbering
ς	staircase join
ε, τ	element/text node construction
$\circledast$	arithmetic/comparison/Boolean operator $*$

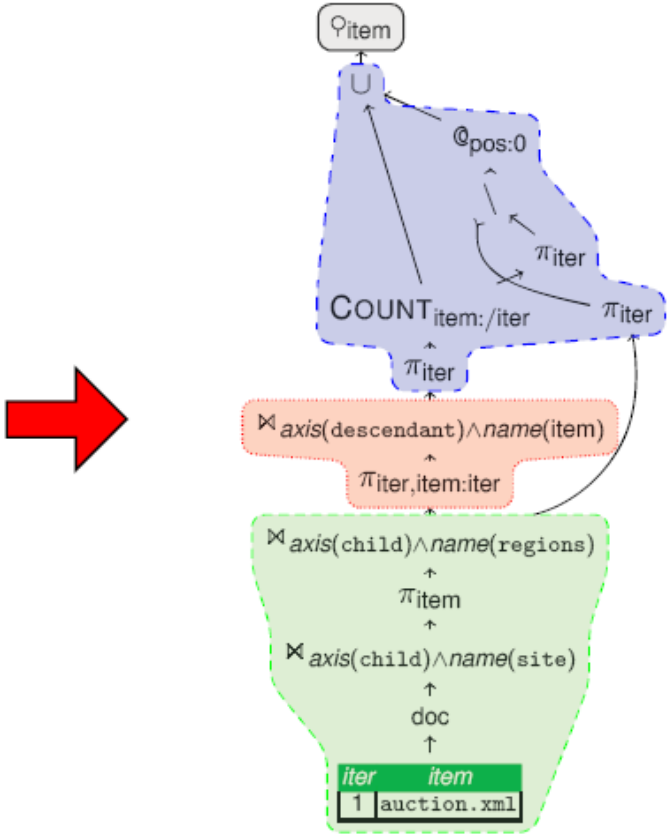
- **No tree pattern matching** or similar operators involved here
- This algebra is efficiently implementable on (top of) SQL hosts

Pathfinder

NICTA

Relational XQuery Complialtion

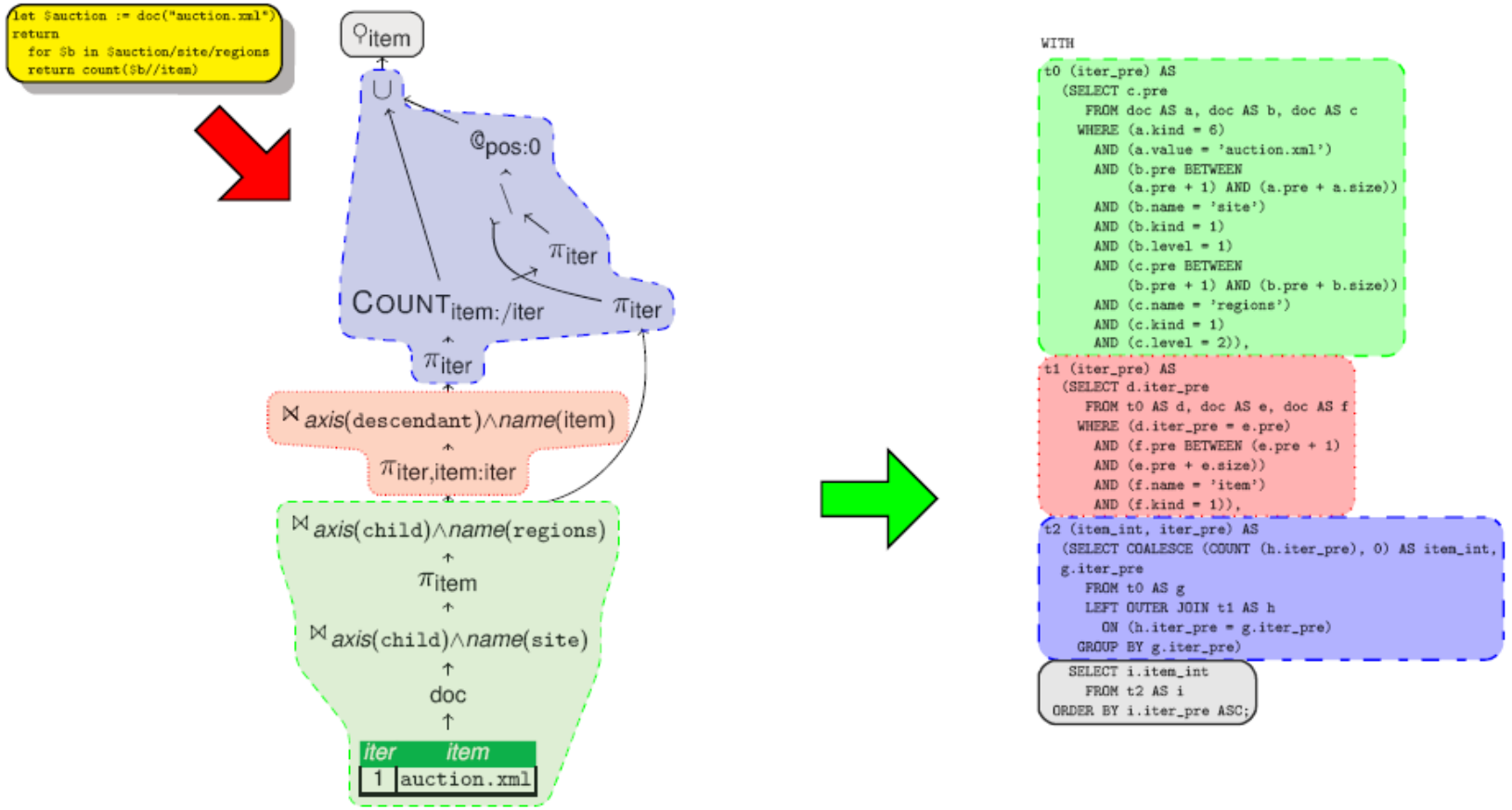
```
let $auction := doc("auction.xml")
return
  for $b in $auction/site/regions
  return count($b//item)
```



Torsten Grust, Sherif Sakr, Jens Teubner: XQuery on SQL Hosts. VLDB, 2004

Pathfinder

NICTA



Torsten Grust, Sherif Sakr, Jens Teubner: XQuery on SQL Hosts. VLDB, 2004

Other Projects

- Galax
  - http://www.galaxquery.org/
- MXQuery
  - http://mxquery.org/
- Zorba
  - http://www.zorba-xquery.com/html/index
- BaseX
  - http://basex.org/
- Xyleme
  - http://www.xyleme.com/
- eXist
  - http://exist.sourceforge.net/
- Saxon
  - http://saxon.sourceforge.net/
- MarkLogic
  - http://developer.marklogic.com/

XML Database Benchmarks

- XBench
  - http://se.uwaterloo.ca/~ddbms/projects/xbench/index.html
- XMach
  - http://dbs.uni-leipzig.de/en/projekte/XML/XmlBenchmarking.html
- XMark
  - http://www.xml-benchmark.org/
- XOO7
  - http://www.comp.nus.edu.sg/~ebh/XOO7/main.html
- TPoX
  - http://tpox.sourceforge.net/
- XPathMark
  - http://sole.dimi.uniud.it/~massimo.franceschet/xpathmark/
- MemBeR
  - http://ilps.science.uva.nl/Resources/MemBeR/member-generator.html
- XSelMark
  - http://xselmark.sourceforge.net/

Tutorial Overview

Session 1

- XQuery Overview – Sherif
- SPARQL Overview – Axel
- XSPARQL: a combined language – Axel
- Compression formats for XML+RDF: EXI+HDT – Sherif

Session 2

- XQuery implementations – Sherif
- **SPARQL implementations – Sherif**
- **XSPARQL implementations – Axel**
- **(optional) Compression formats for XML+RDF: EXI+HDT – Sherif**

- **Q/A - Discussion**

SPARQL Implementations

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

Two Alternatives to Store RDF Data

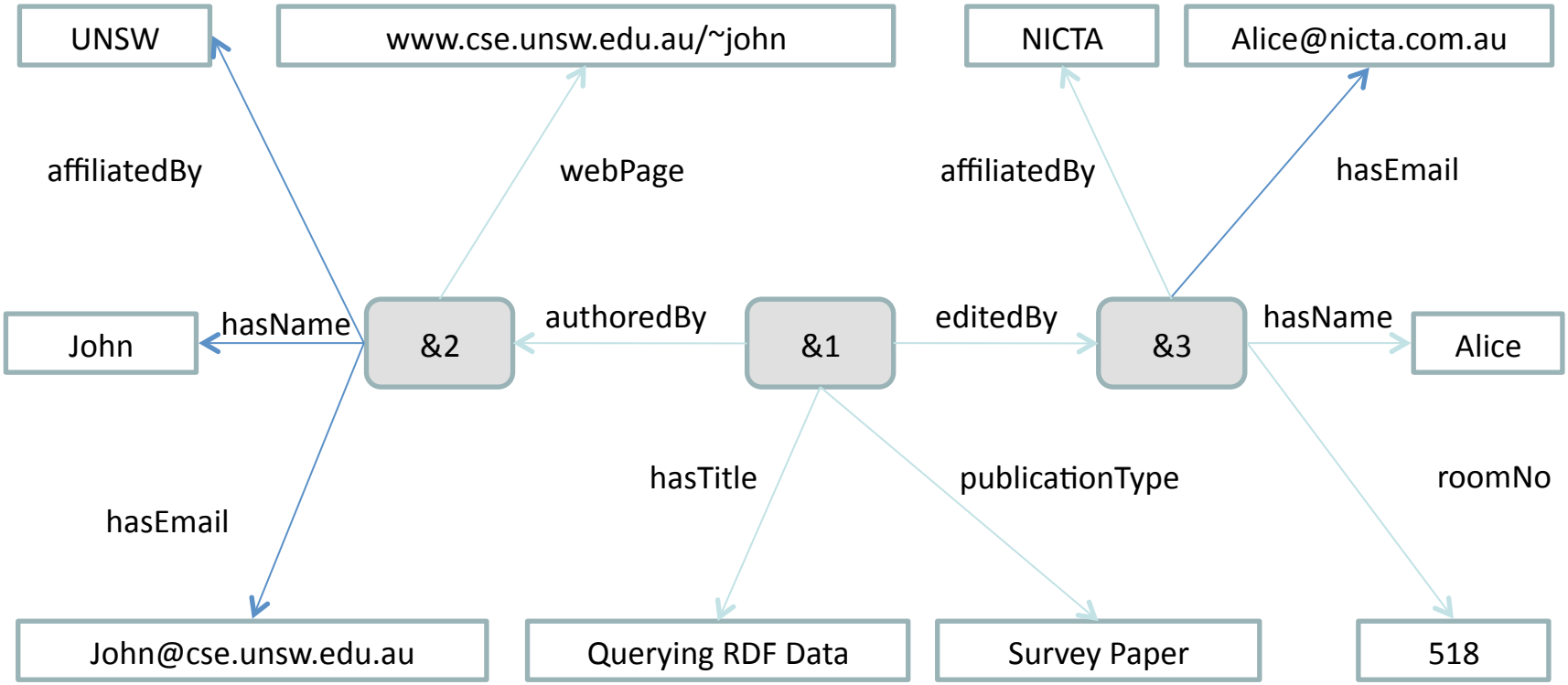
- **Relational RDF Stores**
  - Takes advantage of 30+ years of R&D
  - A schema design exercise?
  - Requires a SPARQL-to-SQL layer
  - Relative flexibility in choosing the actual back-end
- **Native RDF Stores**
  - Takes advantage of 30+ years of R&D
  - Highly tuned for RDF data

Relational RDF Stores

- **Vertical (triple) table stores:** where each RDF triple is stored directly in a *three-colum*n table (subject, predicate, object).
- **Property (n-ary) table stores:** where multiple RDF properties are modeled as ***n-ary*** table columns for the same subject.
- **Horizontal (binary) table stores:** where RDF triples are modeled as one horizontal table or into a set of vertically partitioned binary tables (one table for each RDF property).

Sherif Sakr, Ghazi Al-Naymat: *Relational processing of RDF queries: a survey*. SIGMOD Record 38(4) , 2009

Relational RDF Stores



SELECT ?Z
WHERE { ?X hasTitle “Querying RDF Data”. ?X publicationType “Survey Paper”.
?X authoredBy ?Y. ?Y webPage ?Z. }

Vertical (triple) table stores

Subject	Predicate	Object
Id1	publicationType	Survey Paper
Id1	hasTitle	Querying RDF Data
Id1	authoredBy	Id2
Id2	hasName	John
Id2	affiliatedBy	UNSW
Id2	hasEmail	John@cse.unsw.edu.au
Id2	webPage	www.cse.unsw.edu.au/~john
Id1	editedBy	Id3
Id3	hasName	Alice
Id3	affiliatedBy	NICTA
Id3	hasEmail	Alice@nicta.com.au
Id3	roomNo	518

```
Select  T3.Object
From Triples as T1, Triples as T2,
        Triples as T3, Triples as T4
Where
T1.Predicate="publicationType" and
T1.Object="Survey Paper"
and T2.predicate="hasTitle"
and T2.Object="Querying RDF Data"
and T3.Predicate="webPage"
and T1.subject=T2.subject
and T4.subject=T1.subject
and T4.Predicate="authoredBy"
and T4.Object = T3.Subject
```

No schema changes as new triples are added to the store
SPARQL queries result in equivalent SQL queries involving multiple self-joins

Property Table Stores

Publication

ID	publicationType	hasTitle	authoredBy	editedBy
Id1	Survey Paper	Querying RDF Data	Id2	id3

Person

ID	hasName	affiliatedBy	hasEmail	webPage	roomNo
Id2	John	UNSW	John@cse.unsw.edu.au	www.cse.unsw.edu.au/~john	
Id3	Alice	NICTA	Alice@nicta.com.au		518

```
Select  Person.webPage
From Person, Publication
Where Publication.publicationType = "Survey Paper"
and  Publication.hasTitle = "Querying RDF Data"
and  Publication.authoredBy = Person.ID
```

Implicit or explicit knowledge is required to build the property tables

Horizontal (Binary) Table Stores

publicationType

Id1	Survey Paper

hasTitle

Id1	Querying RDF Data

webPage

Id2	www.cse.unsw.edu.au/~john

hasName

Id2	John
Id3	Alice

affiliatedBy

Id2	UNSW
Id3	NICTA

authoredBy

Id1	Id2

editedBy

Id1	Id3

hasEmail

Id2	John@cse.unsw.edu.au
Id3	Alice@nicta.com.au

roomNo

Id3	518

```
Select  webPage.value
From  PublicationType, hasTitle,
        authoredBy, webPage
Where  publicationType.value = “Survey Paper”
and hasTitle.value = “Querying RDF Data”
and  publicationType.ID = hasTitle.ID
and  publicationType.ID  = authoredBy.ID
and   authoredBy.value = webPage.ID
```

Does this work for Dbpedia which has approx. 65K properties?

RDF-3X

- An RDF query engine which tries to overcome the criticism that triples stores incurs too many expensive self-joins by creating the exhaustive set of indexes and relying on fast processing of merge joins.
- The physical design of RDF-3x is workload-independent and eliminates the need for physical-design tuning by building indexes over all 6 permutations of the three dimensions that constitute an RDF triple.
- Additionally, indexes over count-aggregated variants for all three two-dimensional and all three one-dimensional projections are created.

Thomas Neumann, Gerhard Weikum: ***The RDF-3X engine for scalable management of RDF*** **data. VLDB J. 19 (1), 2010**

RDF-3X

NICTA

- The query processor uses the full set of indexes on the triple tables to rely mostly on *merge joins* over sorted index lists.
- The query optimizer relies upon its cost model in finding the lowest-cost execution plan and mostly focuses on join order and the generation of execution plans.
- Additionally, indexes over count-aggregated variants for all three two-dimensional and all three one-dimensional projections are created.
- It relies on two kinds of statistics:
  - *Specialized histograms* which are generic and can handle any kind of triple patterns and joins. The disadvantage of histograms is that it assumes independence between predicates.
  - *Frequent join paths* in the data which give more accurate estimation.

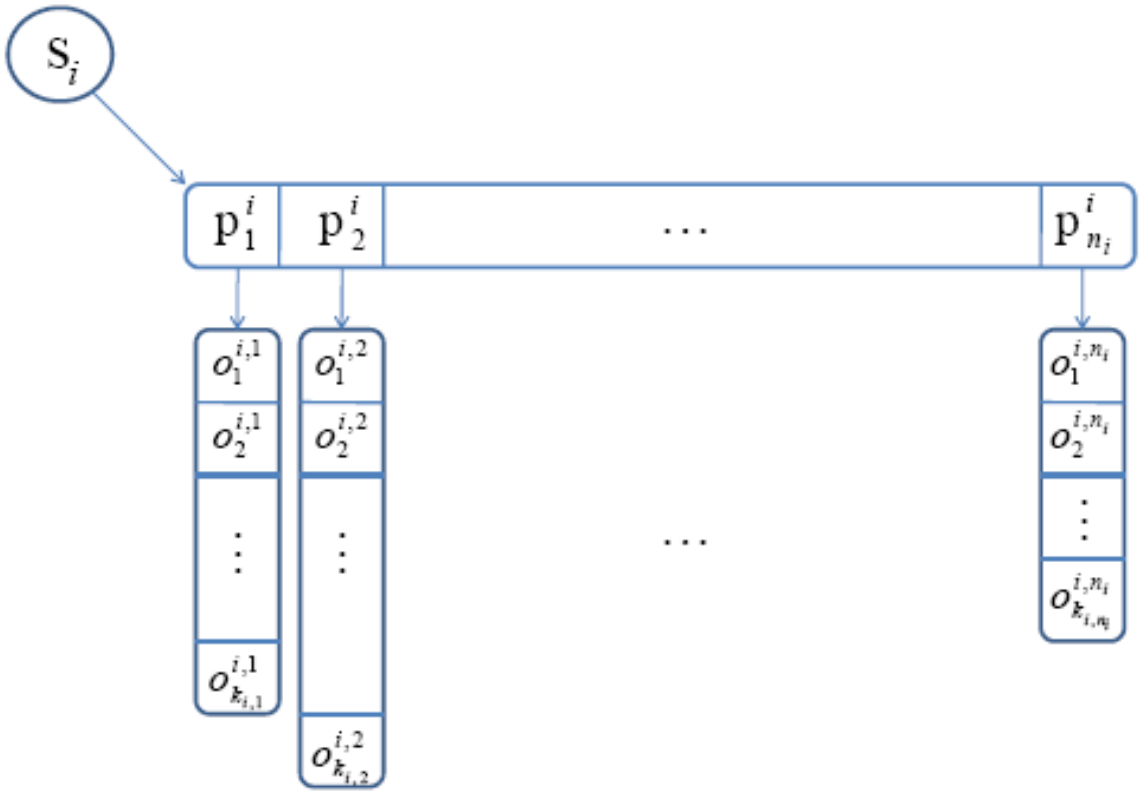
Thomas Neumann, Guido Moerkotte: *Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins*. ICDE, 2011

Hexastore

- RDF storage scheme with main focuses on scalability and generality in its data storage, processing and representation.
- Hexastore is based on the idea of indexing the RDF data in a multiple indexing scheme.
- Each RDF element type have its special index structures built around it.
- Every possible ordering of the importance or precedence of the three elements in an indexing scheme is materialized.
- In total, *six* distinct indices are used for indexing the RDF data. These indices materialize all possible orders of precedence of the three RDF elements.
- A clear disadvantage of this approach is that it features a worst-case *five-fold* storage increase in comparison to a conventional triples table.

Cathrin Weiss, Panagiotis Karras, and Abraham Bernstein: *Hexastore: sextuple indexing for semantic web data management*. PVLDB 1(1), 2008

Hexastore



An Example SPO Index of Hexastore

Oracle RDF

- Oracle introduced an Oracle-based SQL table function *RDFMATCH* to query RDF data.
- The results of *RDFMATCH* table function can be further processed by SQL's rich querying capabilities and seamlessly combined with queries on traditional relational data.
- The core implementation of *RDFMATCH* query translates to a self-join query on triple-based RDF table store.
- The resulting query is executed efficiently by making use of B-tree indexes as well as creating ***materialized join views*** for specialized *subject-property*.

Eugene Inseok Chong, Souripriya Das, George Eadon, and Jagannathan Srinivasan: *An Efficient SQL-based RDF Querying Scheme*. VLDB, 2005

Oracle RDF

- Subject-Property Matrix materialized join views are used to minimize the query processing overheads that are inherent in the canonical triple-based representation of RDF.
- The materialized join views are *incrementally* maintained based on ***user demand*** and ***query workloads***.
- A special module is provided to analyze the table of RDF triples and estimate the size of various materialized views, based on which a user can define a subset of materialized views.

Oracle RDF


```
SELECT t.r reviewer, t.c conf, t.a age
FROM TABLE(RDF_MATCH(
   '(?r  rdf:type    Student)
    (?r  ReviewerOf  ?c)
    (?r  Age         ?a)',
   RDFModels ('reviewers'),
   NULL, NULL)) t
WHERE t.a < 25;
```

Workload Independent Property Tables

- The approach provides a *tailored* schema for each RDF data based on two main parameters:
  - The *Support threshold* which represents a value to measure the strength of correlation between properties in the RDF data.
  - The *null threshold* which represents the percentage of null storage tolerated for each table in the schema.
- The approach involves two phases
  - The *clustering* phase scans the RDF data to automatically discover groups of related properties (i.e., properties that always exist together for a large number of subjects).
  - The *partitioning* phase goes over the formed clusters and balances the trade-off between storing as many RDF properties in clusters as possible while keeping null storage to a minimum based on the null threshold.

Justin J. Levandoski and Mohamed F. Mokbel: RDF Data-Centric Storage. ICWS, 2009

Path-Based RDF Stores

- The main focus of this approach is to improve the performance for *path queries* by extracting all reachable path expressions for each resource and store them.
- There is no need to perform join operations unlike the flat tripe stores or the property tables approach.
- The RDF graph is divided into subgraphs and then each subgraph is stored by applicable techniques into distinct relational tables. More precisely, all classes and properties are extracted from RDF schema data, and all resources are also extracted from RDF data.
- Each extracted item is assigned an identifier and a path expression and stored in corresponding relational table.

Akiyoshi Matono, Toshiyuki Amagasa, Masatoshi Yoshikawa, and Shunsuke Uemura: *A Path-based Relational RDF Database*. ADC, 2005

SW-Store

- The triples table is rewritten into *n* two-column tables where *n* is the number of unique properties in the data.
- In each binary table, the first column contains the subjects that define that property and the second column contains the object values for those subjects while the subjects that do not define a particular property are simply omitted from the table for that property.
- Each table is sorted by subject, so that particular subjects can be located quickly, and that fast merge joins can be used to reconstruct information about multiple properties for subsets of subjects.

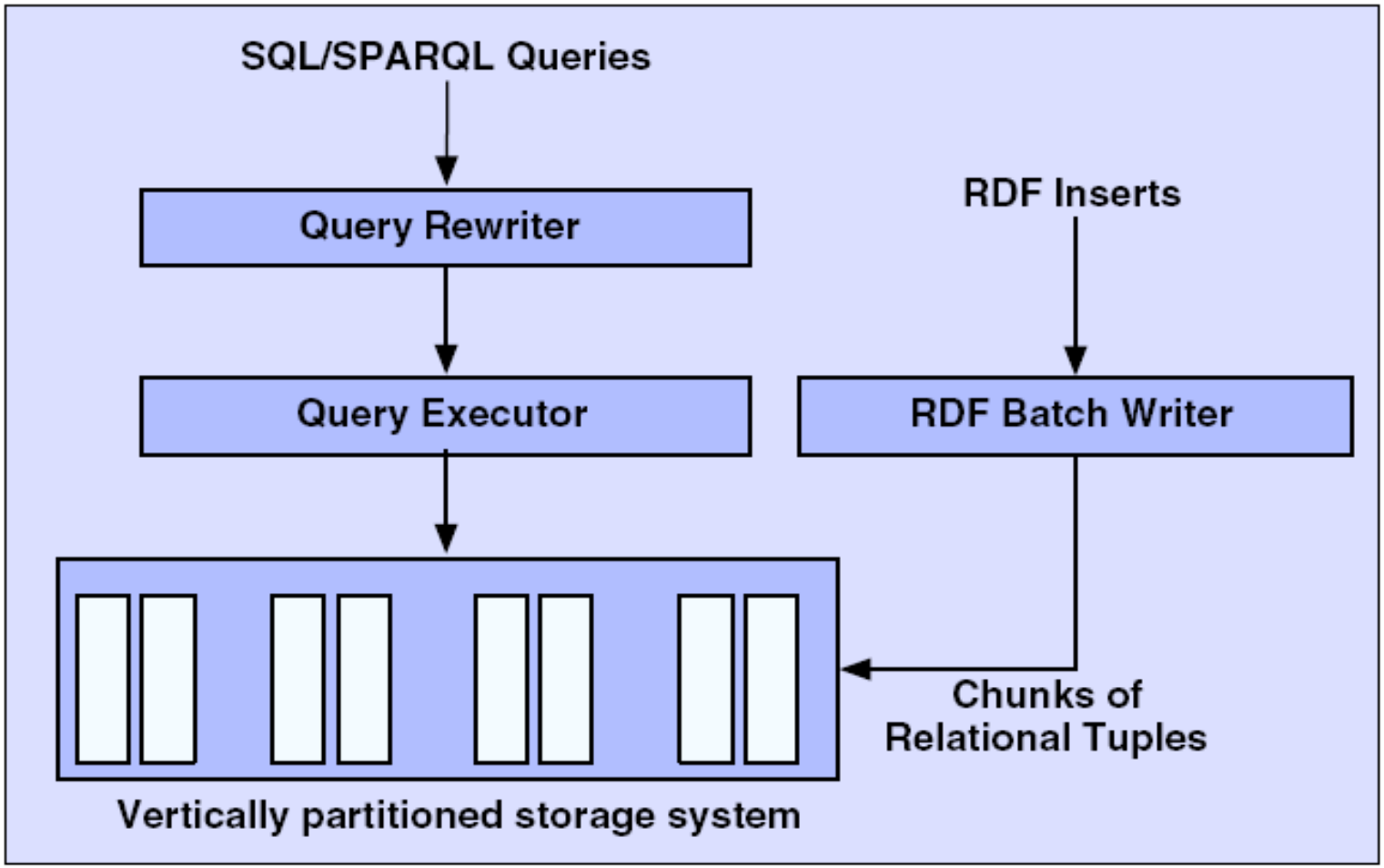
Daniel J. Abadi, Adam Marcus, Samuel Madden, and Kate Hollenbach: *SW-Store: a vertically partitioned DBMS for Semantic Web data management*. VLDB J. 18(2), 2009

SW-Store

- For a *multi-valued* attribute, each distinct value is listed in a successive row in the table for that property.
- The implementation of SW-Store relies on a column-oriented DBMS, ***C-store***, to store tables as collections of columns rather than as collections of rows.
- The advantage of this approach is that while property tables need to be carefully constructed so that they are wide enough but not too wide to independently answer queries, the algorithm for creating tables in the vertically partitioned approach is straightforward and need not change over time.
- The main disadvantages of this approach are: the increased cost of inserts new triples and the cost of tuple reconstruction

SW-Store

NICTA

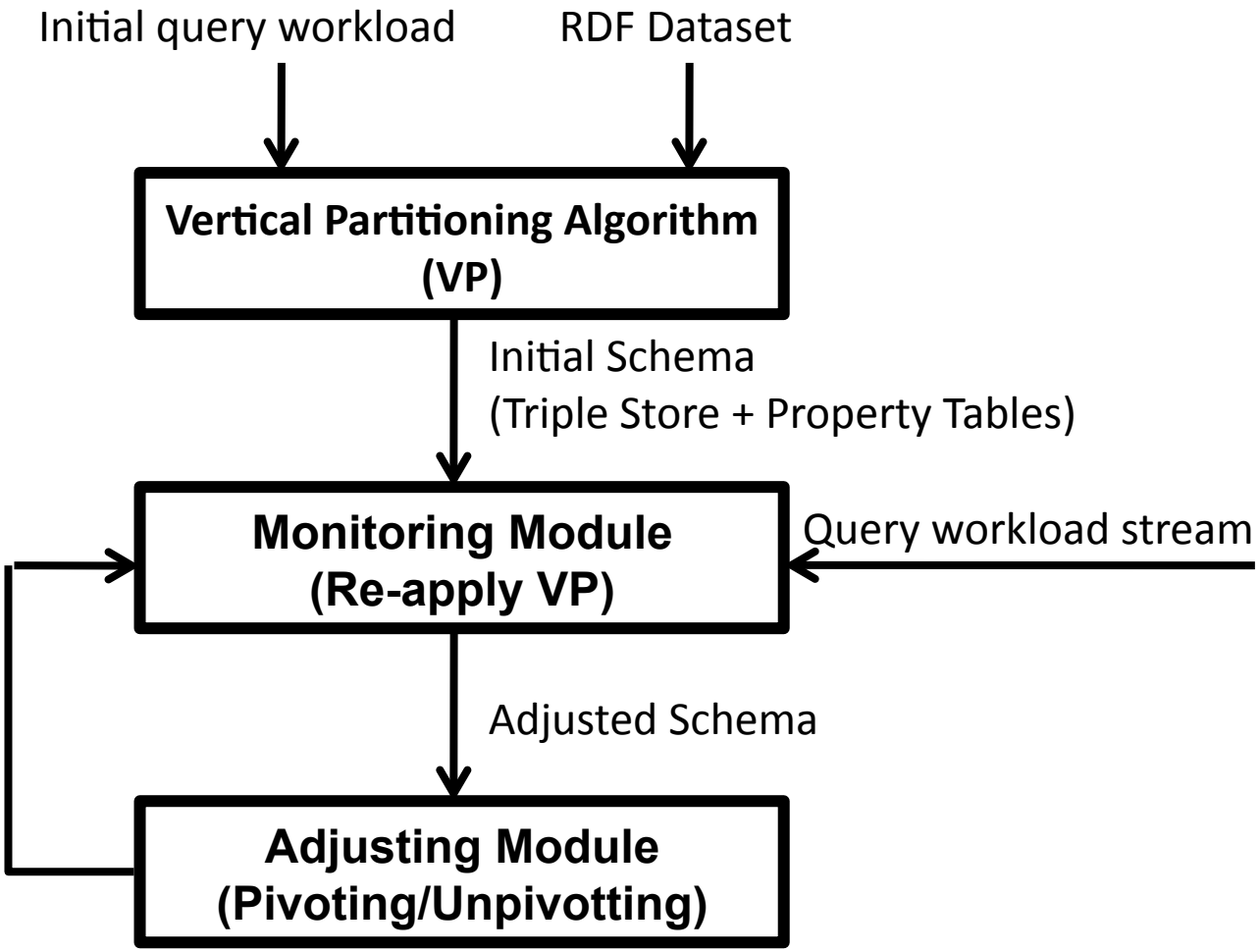


AdaptRDF

- A two-phase approach for designing efficient tailored but flexible storage solution for RDF data based on its query workload.
- A *workload-aware vertical partitioning* phase.
  - Reduces the number of join operations in the query evaluation process
- An *automated adjustment* phase
  - Maintains the efficiency of the performance of the query processing by adapting the underlying schema to cope with the dynamic nature of the query workloads
  - Rows pivoting and uninviting

Hooran MahmoudiNasab and Sherif Sakr: Efficient and Adaptable Query Workload-Aware Management for RDF Data. WISE, 2010

AdaptRDF



AdaptRDF

NICTA

	Upper							Lower		
	a_6	a_2	a_3	a_7	a_{10}	a_8	a_1	a_9	a_4	a_5
a_6	90	30	25	25	15	15	10	0	0	0
a_2	30	105	60	40	20	15	15	0	0	0
a_3	25	60	40	30	15	15	10	0	0	0
a_7	25	40	30	65	25	15	10	0	0	0
a_{10}	15	20	15	25	70	10	10	0	0	0
a_8	15	15	15	15	10	60	10	18	18	18
a_1	10	15	10	10	10	10	50	18	18	18
a_9	0	0	0	0	0	18	18	40	40	40
a_4	0	0	0	0	0	18	18	40	40	40
a_5	0	0	0	0	0	18	18	40	40	40

AdaptRDF

NICTA

Subject	Predicate	Object
Id1	publicationType	Survey Paper
Id1	hasTitle	Querying RDF Data
Id1	authoredBy	Id2
Id2	hasEmail	John@cse.unsw.edu.au
Id2	webPage	www.cse.unsw.edu.au/~john
Id1	editedBy	Id3
Id3	hasEmail	Alice@nicta.com.au

Subject	hasName	affiliatedBy	roomNo
Id2	John	UNSW	
Id3	Alice	NICTA	518

(a) Pivot (hasName, affiliatedBy, roomNo)

Subject	Predicate	Object
Id1	publicationType	Survey Paper
Id1	hasTitle	Querying RDF Data
Id1	authoredBy	Id2
Id2	hasEmail	John@cse.unsw.edu.au
Id2	webPage	www.cse.unsw.edu.au/~john
Id1	editedBy	Id3
Id3	hasEmail	Alice@nicta.com.au
Id3	roomNo	518

Subject	hasName	affiliatedBy
Id2	John	UNSW
Id3	Alice	NICTA

(b) Unpivot (roomNo)

RDF Benchmarks

- **Lehigh University Benchmark (LUBM)**
  - http://swat.cse.lehigh.edu/projects/lubm/
  - (University domain)
- **Berlin SPARQL Benchmark (BSBM)**
  - http://www4.wiwiss.fu-berlin.de/bizer/berlinsparqlbenchmark/
  - E-Commerce domain
- **The SP²Bench SPARQL Performance Benchmark**
  - http://dbis.informatik.uni-freiburg.de/index.php?project=SP2B
  - DBLP Scenario
- **The DBpedia SPARQL Benchmark**
  - http://aksw.org/Projects/DBPSB
  - Wikipedia + YAGO

Benchmarkings (1)

- The benchmark implements two relational RDF storage solutions – *triple-store* and *vertically-partitioned* – in a *MonetDB/SQL*, a fully-functional open source column-store, and a well-known – for its performance – commercial row-store DBMS.
- The vertically-partitioned approach does not outperform triple-store when both are implemented in a row-store engine.
- The vertically-partitioned approach outperforms triple-store when both are implemented in a column-store. The processing efficiency of column-stores is particularly suited for RDF data management applications.

Lefteris Sidirourgos, Romulo Goncalves, Martin L. Kersten, Niels Nes, and Stefan Manegold. *Column-store support for RDF data management: not all swans are white*. PVLDB 1(2), 2008

Benchmarkings (1)

- Once the proper clustered indices are used in a row-store engine, the triple-store performs better than the vertically-partitioned approach.
- There is a potential of a scalability problems for the vertically-partitioned approach when the number of properties in an RDF data-set is high. With a larger number of properties, the triple-store solution manages to outperform the vertically-partitioned approach also on a column-store engine.

Lefteris Sidirourgos, Romulo Goncalves, Martin L. Kersten, Niels Nes, and Stefan Manegold. *Column-store support for RDF data management: not all swans are white*. PVLDB 1(2), 2008

Benchmarkings (2)

- The benchmark evaluates the approaches of relational processing for RDF Queries: *triple stores (**TS**)*, *binary table stores (**BS**)*, *property table stores (**PS**)* and *traditional relational stores (**RS**)*.
- The experimental evaluation is done on top of the SP^{2}Bench benchmark and compares the following metrics: *loading time*, *storage costs* and *query performance*.
- The ***RS*** *scheme* is the fastest due to the less required number of insert tuple operations. Similarly, the ***TS*** requires less loading time than ***BS*** since the number of inserted tuples and updated tables are smaller for each triple.

Hooran MahmoudiNasab and Sherif Sakr: *An Experimental Evaluation of Relational RDF Storage and Querying Techniques*. BenchmarX, 2010

Benchmarkings (2)

- The ***RS*** *scheme* represents the cheapest approach because of the normalized design and the absence of any data redundancy. The ***BS*** *scheme* represents the most expensive approach due to the redundancy of the ID attributes for each binary table.
- There is no clear winner between the triple store (TS) and the binary table (BS) encoding schemes. Triple store (TS) with its simple storage and the huge number of tuples in the encoding relation is still very competitive to the binary tables encoding scheme because of the full set of B-tree physical indexes over the permutations of the three encoding fields (subject; predicate; object).

Hooran MahmoudiNasab and Sherif Sakr: ***An Experimental Evaluation of Relational RDF Storage and Querying Techniques*****. BenchmarX, 2010**

Benchmarkings (2)

- The query performance of the (BS) encoding scheme is affected badly by the increase of the number of the predicates In the input query. It is also affected by the subject-object or object-object type of joins where no index information is available for utilization. Such problem could be solved by building materialized views over the columns of the most frequently referenced pairs of attributes.
- Although their generality, there is still a clear gap between the query performance of the (TS) and (BS) encoding schemes in comparison with the tailored relational encoding scheme (RS) of the RDF data.

Hooran MahmoudiNasab and Sherif Sakr: *An Experimental Evaluation of Relational RDF Storage and Querying Techniques*. BenchmarX, 2010

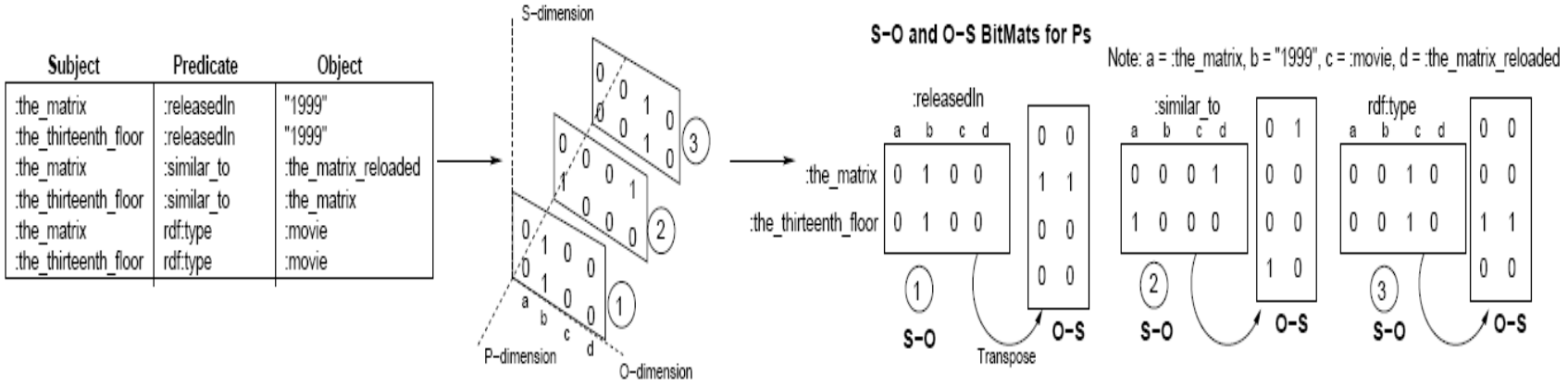
Matrix "Bit“ loaded

NICTA

- A compressed *bit-matrix* structure for storing huge RDF graphs.
- The RDF data is represented by a 3D bit-cube, where each dimension of the *bitcube* represents subjects, predicates, and objects.
- The volume of this *bitcube* is Vs × Vp × Vo.
- Each cell in the *bitcube* represents a unique RDF triple that can be formed by the combination of S, P, O positions which are the coordinates of that bit.
- For each matrix, it stores two *bitarrays* – row and column bitarray – which give a condensed representation of all the non-empty row and column values in the given BitMat.
- It employs an initial *pruning* technique, followed by a *variable-binding-matching* algorithm on *BitMats* to produce the final results.
- It does not build intermediate join tables and works directly on the compressed data.

Medha Atre, Vineet Chaoii, Mohammed J. Zaki, James A. Hendler: *Matrix "Bit" loaded: a scalable lightweight join query processor for RDF data* . WWW, 2010

Matrix "Bit“ loaded



Jena

- Developed by the Hewlett Packard Semantic Web Lab.
- Jena is a Semantic Web application framework for Java.
- It provides a programming environment for semantic technologies such as RDF(S) and OWL that supports SPARQL queries and a rule-based inference engine.
- Its engine is a combination of in-memory and native.
- A native, on-disk implementation of Jena is the *Jena TDB triple* store.

http://incubator.apache.org/jena/

Jena

- Triples are stored in the form of a *dictionary*. Every node is given a unique ID and instead of storing the whole node, only their ID is put in the triple store.
- Jena supports different RDF syntaxes: RDF/XML, N3, N-Triples, Turtle.
- Jena SDB version can use PostgreSQL, MySQL, Oracle or MS SQL Serveras its backend.

http://incubator.apache.org/jena/

AllegroGraph

- AllegroGraph is a persistent graph database developed by Franz Inc. that uses in-memory utilization in combination with native engine in order to maintain billions of quads.
- AllegroGraph supports named graphs. It has an optimised indexing system consisting of different combinations of S, P, O, G and I.
- It has a mechanism of automatic index optimisation that optimises indices based on how much they are used.
- Querying can be done through SPARQL 1.1 queries, with support for full-text, geospatial and geotemporal search.
- It is also compatible with a lot of other Franz technologies such as *RacerPro*, *Pepito*, *TopBraid Composer*, etc. to improve performance and usability.

http://www.franz.com/agraph/allegrograph/

AllegroGraph

- In June 2011, AllegroGraph announced at Semtech conference a load and query of 310 Billion triples as part of a joint project with Intel.
- In August 2011, AllegroGraph announced that with the help of Stillwater SC and Intel, they achieved the industry's first load and query of 1 Trillion RDF Triples. Total load was 1,009,690,381,946 triples in just over 338 hours for an average rate of 829,556 triples per second.
- It is in late-stage development on a clustered version that will push storage into trillions of triples.

From imagination to impact

http://www.franz.com/agraph/allegrograph/

Virtuoso

- Virtuoso is an object-relational SQL database that supports SPARQL embedded into SQL for querying RDF data in the Virtuoso database.
- Querying can be done by using SPARQL queries. There is however no SPARQL endpoint, only an SQL command line interface through which you can send SPARQL 1.1 queries to the Virtuoso repository.
- There is also a possibility to send the queries using ODBC, JDBC, ADO.NET or OLE DB.
- Full-text search and geospatial search are also supported.
- The reasoner that is implemented in the Virtuoso store is a backward chaining OWL reasoner, but reasoning can be added through the Jena framework.

http://www.openlinksw.com/dataspace/dav/wiki/Main/VOSRDF

Mulgara

- Mulgara was formerly known as Kowari.
- An Open Source, massively scalable, transaction-safe, purpose-built database for the storage, retrieval and analysis of metadata.
- It supports RDF and OWL.
- It is optimised for large amounts of triples, for metadata in particular, but it keeps low memory requirements.
- Triples in the Mulgara store are stored natively (on-disk), both locally or distributed.
- Supports SPARQL in addition to a language called *TQL* where Full-text search functionality is also available.
- Mulgara supports: RDF/XML, N-Triples, Turtle.
- Mulgara has a reasoner that supports: DAML+OIL, RDF(S) and OWL

http://www.mulgara.org/

Other Projects

- Sesame
  - http://www.openrdf.org/
- 3Store
  - http://www.aktors.org/technologies/3store/
- 4store
  - http://4store.org/
- BigOWLIM
  - http://www.ontotext.com/owlim/editions
- Bigdata
  - http://www.systap.com/bigdata.htm
- BrightstarDB
  - http://www.brightstardb.com/
- Apache Fuseki
  - http://incubator.apache.org/jena/documentation/serving_data/

Other Projects

- Parliament
  - http://parliament.semwebcentral.org/
- OpenAnzo
  - http://www.openanzo.org/
- StrixDB
  - http://www.strixdb.com/
- Redstore
  - http://www.aelius.com/njh/redstore/
- Meronymy
  - http://www.meronymy.com/
- Neo4j
  - http://neo4j.org/
- rdfstore-js
  - https://github.com/antoniogarrote/rdfstore-js

Benchmarkings (3)

- The benchmark compared between: *Jena TDB*, *AllegroGraph*, *OWLIM-SE*, *Sesame* and *Mulgara*.
- The experiments have been conducting using the LUMB, BSBM and SP²Bench benchmarks.
- The comparison metrics are: Loading time, Query time and ease of use.
- The experiments reported that the *Mulgara* was the easies to install, run and use.
- *OWLIM* has the best performance for loading time and query time followed by *Mulgara* then *Jena*.
- *AllegroGraph* is the most robust tested system.

Nicolas Vanden Bossche and Christopher Matheus: *Benchmarking RDF Stores*. Bell Labs Technical Report, 2011

Distributed Processing of RDF/SPARQL

- A horizontally scalable RDF database system.
- It installs a best-of-breed RDF-store on a cluster of machines (they use RDF-3X since they found this to be the fastest single-node RDF-store in their benchmarking).
- It partitions an RDF data set across the data stores. Instead of randomly assigning triples to partitions using hash partitioning, they take advantage of the fact that RDF uses a graph data model, they use an optimized graph partitioning algorithm.
- Triples that are close to each other in the RDF graph to be stored on the same machine and a smaller amount of network communication at query time is required.

Jiewen Huang, Daniel J. Abadi and Kun Ren:Scalable SPARQL Querying of Large RDF Graphs. PVLDB 4 (1), 2011

Distributed Processing of RDF/SPARQL

- In order to maximize the percentage of query processing that can be done in parallel, they allow some overlap of data across partitions.
- They use a method for automatic decomposition mechanism of queries into chunks that can be performed independently, with zero communication across partitions. These chunks are then reconstructed using the Hadoop MapReduce framework (***HadoopDB***).
- The performance is up to three orders of magnitude more efficient than popular multi-node RDF data management systems.

Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel J. Abadi, Alexander Rasin and Avi Silberschatz: *HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads*. PVLDB 2(1), 2009

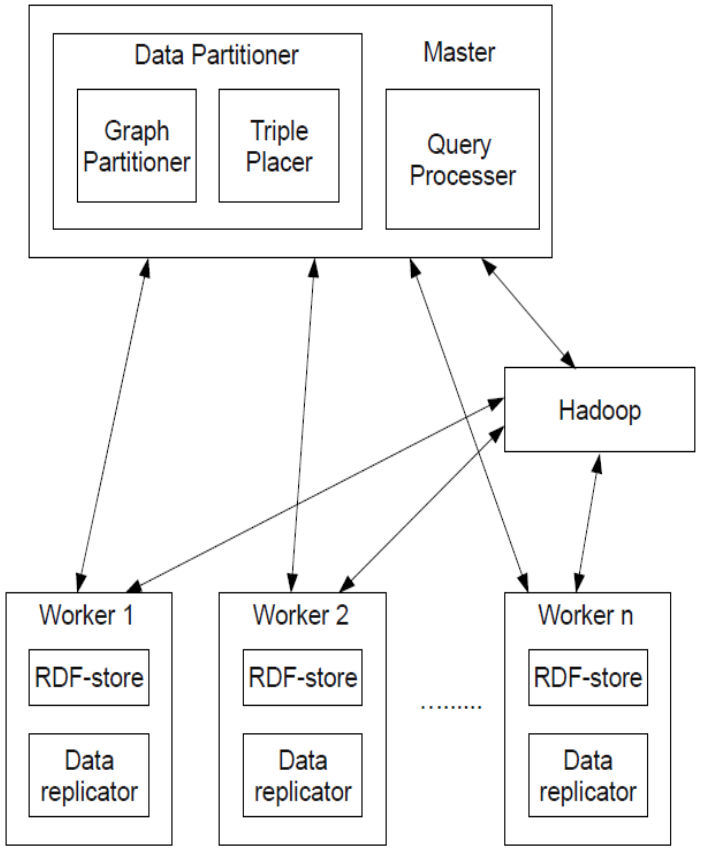
Distributed Processing of RDF/SPARQL

NICTA

•The master node also serves as the interface for SPARQL queries. It accepts queries and analyzes them closely.

• If it determines that the SPARQL pattern can be searched for completely in parallel by each worker in the cluster, then it sends the pattern to each node in the cluster for processing.

•If, it determines that the pattern requires some coordination across workers during the search, it decomposes the query into subgraphs that can be searched for in isolation, and ships these subgraphs to the worker nodes.



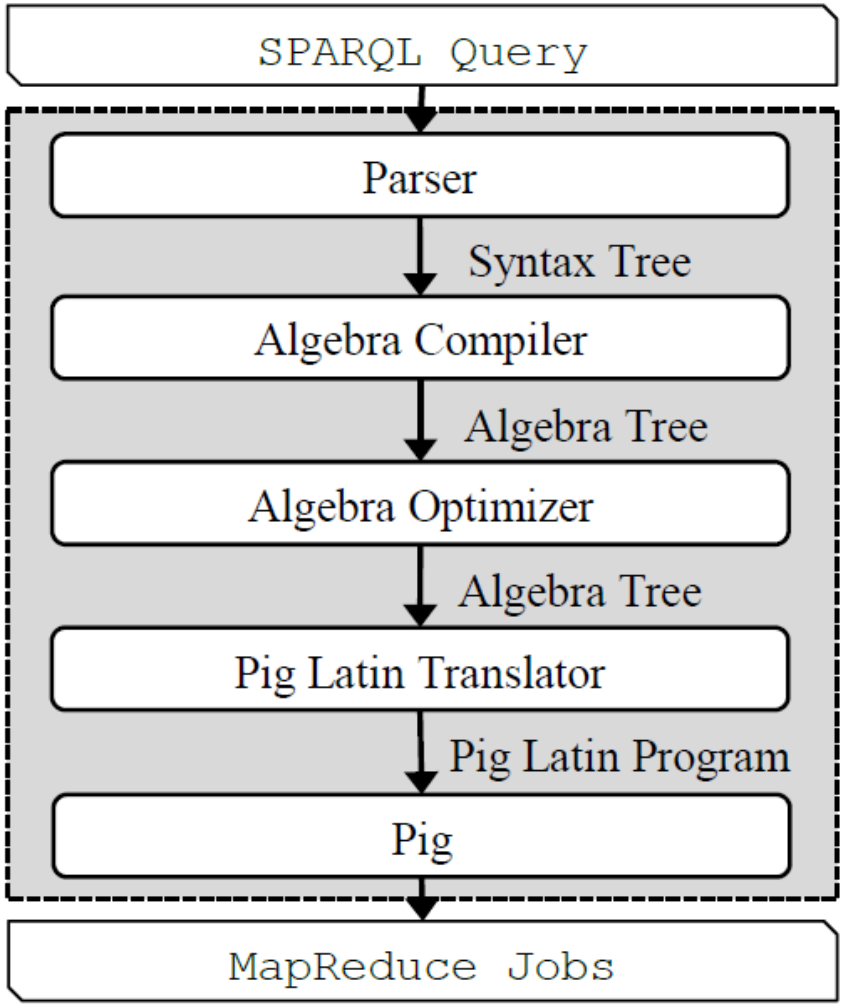
Jiewen Huang, Daniel J. Abadi and Kun Ren:Scalable SPARQL Querying of Large RDF Graphs. PVLDB 4 (1), 2011

PigSPARQL

- A system which gives us the opportunity to process complex SPARQL queries on a MapReduce cluster.
- SPARQL queries are translated into ***Pig Latin***, a data analysis language developed by Yahoo! Research.
- Pig Latin programs are executed by a series of MapReduce jobs on a Hadoop cluster.
- PigSPARQL offers not only a declarative way of specifying the transformation part, but also a scalable implementation of the whole ETL-process on a MapReduce cluster.

Alexander Schätzl, Martin Przyjaciel-Zablock and Georg Lausen: *PigSPARQL: Mapping SPARQL to Pig Latin*. SWIM, 2011

PigSPARQL



Alexander Schätzl, Martin Przyjaciel-Zablock and Georg Lausen: *PigSPARQL: Mapping SPARQL to Pig Latin*. SWIM, 2011

HBase vs MySQL Cluster

- A comparison between *HBase* and *MySQL Cluster* for SPARQL query processing.
- An experimental comparison of the two proposed approaches on a cluster of commodity machines LUMB.
- Both approaches were up to the task of efficiently storing and querying large RDF datasets.
- The the HBase solution was capable of dealing with larger RDF datasets and showed superior query performance and scalability.
- Cloud computing has a great potential for scalable Semantic Web data management.

Craig Franke, Samuel Morin, Artem Chebotko, John Abraham and Pearl Brazier: *Distributed Semantic Web Data Management in HBase and MySQL Cluster*. IEEE CLOUD, 2011

Tutorial Overview

Session 1

- XQuery Overview – Sherif
- SPARQL Overview – Axel
- XSPARQL: a combined language – Axel
- Compression formats for XML+RDF: EXI+HDT – Sherif

Session 2

- XQuery implementations – Sherif
- SPARQL implementations – Sherif
- **XSPARQL implementations – Axel**
- **(optional) Compression formats for XML+RDF: EXI+HDT – Sherif**

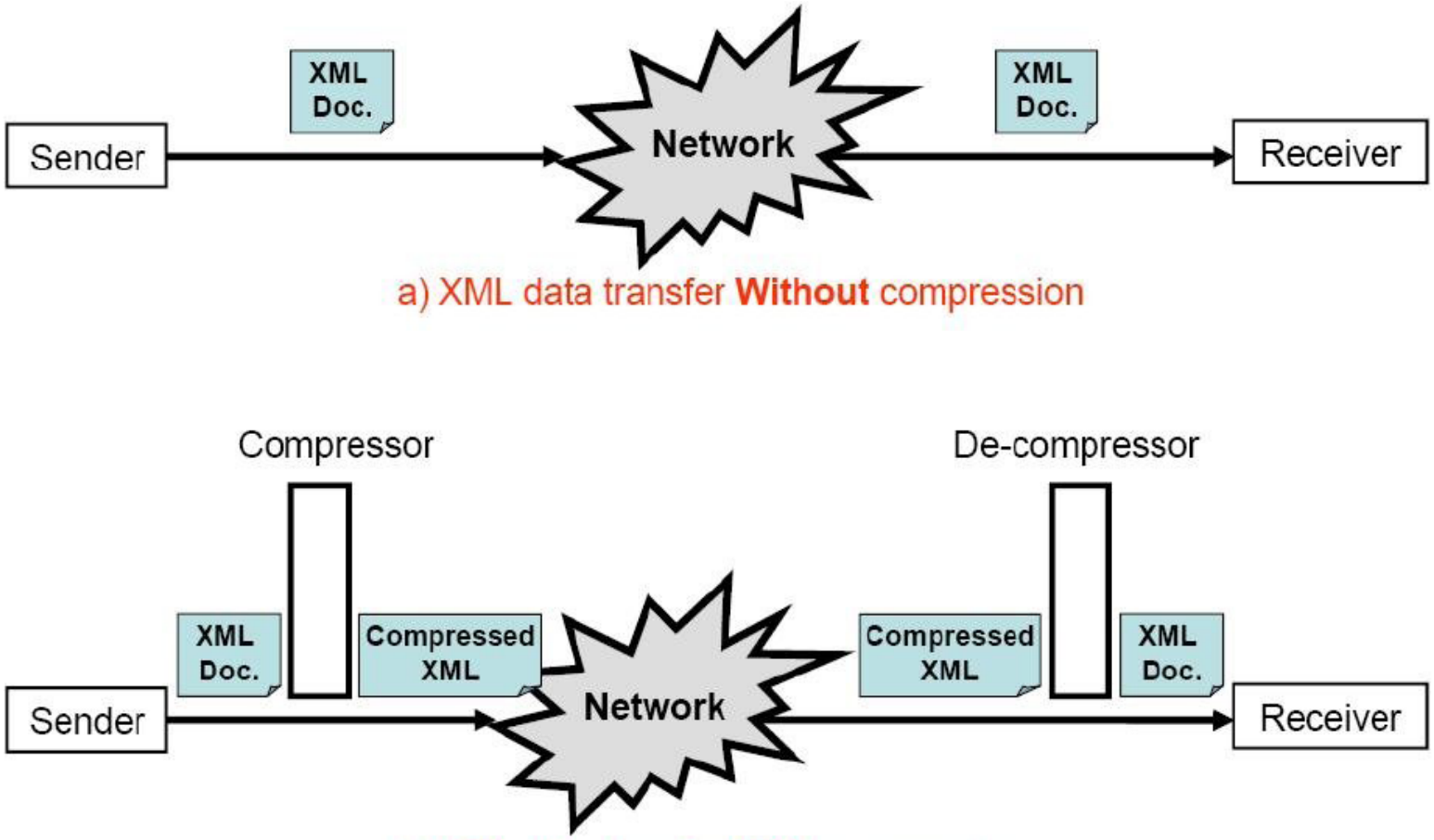
- **Q/A - Discussion**

XML Compression

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

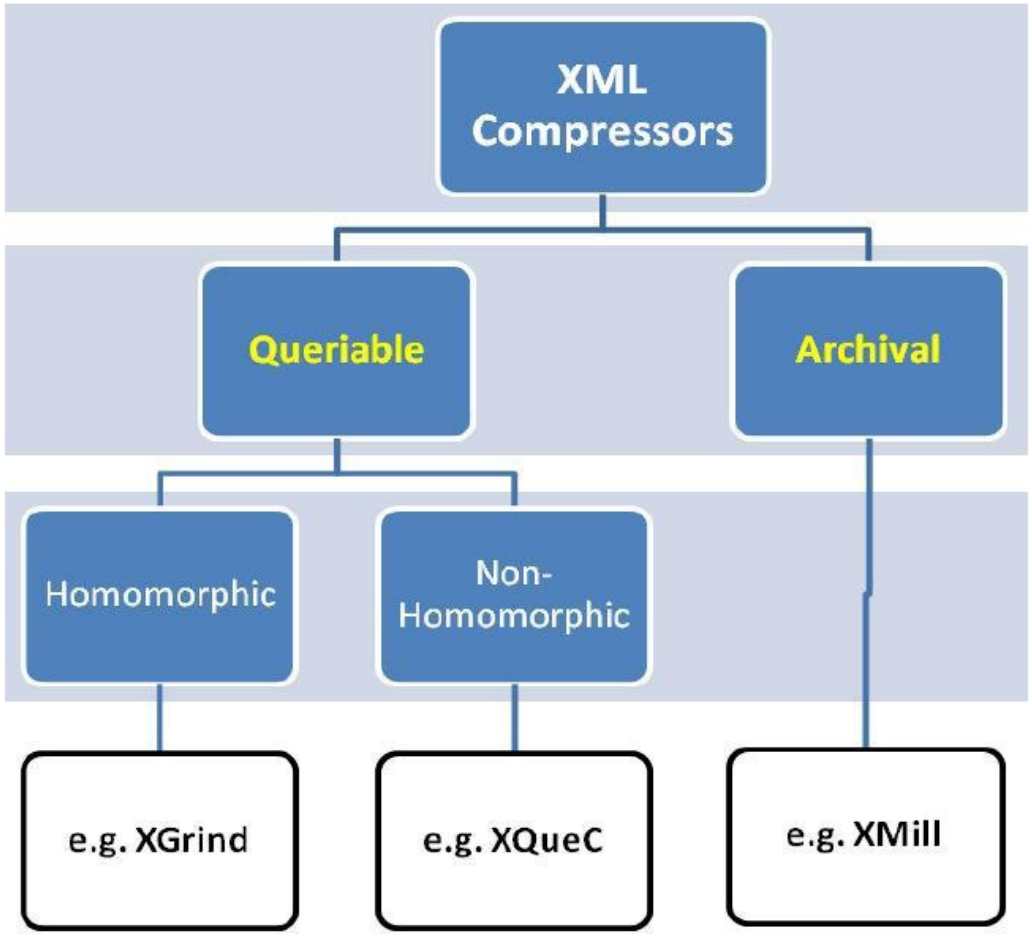
NICTA Funding and Supporting Members and Partners

XML Compression



XML Compressors

NICTA



XML Compressors

- ***General Text Compressors***: Since XML data are stored as text files, these compressors use the traditional general purpose text compression tools. They are XML-Blind, i.e. they treat XML documents as usual plain text documents and thus apply the traditional text compression techniques.
- ***XML Conscious Compressors:*** This group of compressors are designed to take the advantage of the awareness of the XML document structure in order to achieve better compression ratios over the general text compressors.
  - *Schema dependent compressors*: where both of the encoder and decoder must have access to the document schema information to achieve the compression process.
  - *Schema independent compressors*: where the availability of the schema information is not required to achieve the encoding and decoding processes.

XML Compressors

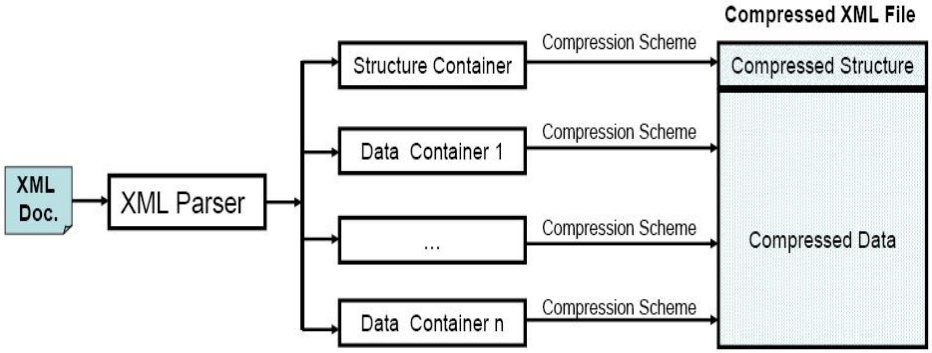
- ***Non-Queriable (Archival) XML Compressors:*** This group of the XML compressors does not allow any queries to be processed over the compressed format. The main focus of this group is to achieve the highest compression ratio. By default, general purpose text compressors belong to the non-queriable group of compressors.
- ***Queriable XML Compressors:*** This group of the XML compressors allow queries to be processed over their compressed formats. The compression ratio of this group is usually worse than that of the archival XML compressors. However, the main focus of this group is to avoid full document decompression during query execution.
  - *Homomorphic compressors:* where the original structure of the XML document is retained and the compressed format can be accessed and parsed in the same way of the original format.
  - *Non-homomorphic compressors:* where the encoding process of the XML document serrates the structural part from the data part. Therefore, the structure of the compressed format is different from the structure of the original XML document.

XMill

- Separates the structure from data and the grouping of the data values into homogenous containers based on their relative paths in the tree and their data types.
- Both of the structural and data value parts of the source XML document are collected and compressed separately.
- In the structure part, XML tags and attributes ar encoded in a dictionary-based fashion before passing it to a back-end general text compression scheme.
- In the data part, data values are grouped into homogenous and semantically related containers according to their path and data type. Each container is then compressed separately using specialized compressor that is ideal for the data type of this container.

Hartmut Liefke, Dan Suciu: XMILL: An Efficient Compressor for XML Data. SIGMOD, 2000

XMill




```
<customers>
  <customer id="c1">
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <invoice total="300">
      <items>
          <item>item1</item>
          <item>item2</item>
      <items>
    </invoice>
  </customer>

 <customer id="c2">
    <firstName>Bill</firstName>
    <lastName>Luis</lastName>
 </customer>
</customers>
```

XML File

Elements Table

1	/customers
2	/customers/customer
3	/customers/customer/firstName
4	/customers/customer/lastName
5	/customers/customer/invoice
6	/customers/customer/invoice/items
7	/customers/customer/invoice/items/item

Attributes Table

100	/customers/customer/@id
101	/customers/customer/invoice/@total

/customers/customer/firstName

Smith
Luis

/customers/customer/lastName

John
Bill

/customers/customer/invoice/items/item

Item1
item2

/customers/customer/@id

C1
C2

/customers/customer/invoice/@total

300

http://www.cs.washington.edu/homes/suciu/XMLTK/xmill/www/

XGrind

- XML-conscious compression scheme to support querying **without** the need for a full decompression of the compressed XML document.
- *XGrind* does not separate data from structure. It retains the original structure of the XML document.
- Element and attribute names are encoded using a dictionary-based encoding.
- Character data is compressed using semi-adaptive Huffman coding.
- The query processor of *XGrind* can only handle exact-match and prefix-match queries on compressed values and partial-match and range queries on decompressed values.
- Several operations are not supported by *XGrind*, for example, non-equality selections in the compressed domain. Therefore, *XGrind* cannot perform any join, aggregation, nested queries, or construct operations.

Pankaj M. Tolani, Jayant R. Haritsa, XGRIND: A query-friendly XML compressor, ICDE, 2002.

XML Compressors

- XMLPPM
  - http://xmlppm.sourceforge.net/
- XWRT
  - http://sourceforge.net/projects/xwrt/
- Exalt
  - http://exalt.sourceforge.net/
- Rngzip
  - http://contrapunctus.net/league/haques/rngzip/
- Benchmark
  - http://xmlcompbench.sf.net/

XML Fast Infoset

- http://www.w3.org/XML/Binary/
- An international standard that specifies a binary encoding format for XML documents.
- It aims to provide more efficient serialization than the text-based XML format.
- The underlying file format is with tag/length/value blocks.
- Text values of attributes and elements are therefore stored with length prefixes rather than end delimiters.
- An index table is built for most strings, which includes element and attribute names, and their values. This means that the text of repeated tags and values only appears once per document.

http://java.sun.com/developer/technicalArticles/xml/fastinfoset/

XML EXI

- W3C recommendation for a compact, high performance XML representation that is designed to work well for a broad range of applications.
- It aims of improving performance and significantly reducing bandwidth requirements without compromising efficient use of other resources such as battery life, code size, processing power, and memory.
- EXI uses a grammar-driven approach and a straightforward encoding algorithm and a small set of datatype representations. Therefore, EXI processors should be relatively simple and can be implemented on devices with limited capacity.

http://www.w3.org/XML/EXI/

XML EXI

- EXI is schema *informed*, meaning that it can utilize available schema information to improve compactness and performance, but does not depend on accurate, complete or current schemas to work.
- A program module called an **EXI processor** is used by application programs to encode their structured data into *EXI streams (***EXI stream encoder***)* and/or to decode *EXI streams* to make the structured data accessible (**EXI stream decoder)**.
- An **EXI stream** is an *EXI Header* followed by an *EXI body*. The EXI body carries the content of the document, while the EXI header communicates the options used for encoding the EXI body

XML EXI

- The building block of an *EXI body* is an *EXI event*. An EXI body consists of a sequence of EXI events representing an *EXI document* or an *EXI fragment*.
- The *EXI events* permitted at any given position in an EXI stream are determined by the *EXI grammar*. As is the case with XML, the events occur with nesting pairs of matching start element and end element events where any pair does not intersect with another except when it is fully contained in the other.
- The *EXI grammar* incorporates knowledge of the XML grammar and may be augmented and refined using schema information and fidelity options.

XML EXI

- Each EXI stream begins with an EXI header.
- The **EXI header** can identify EXI streams, distinguish EXI streams from text XML documents, identify the version of the EXI format being used, and specify the options used to process the body of the EXI stream.
- The EXI Options field within an EXI header is optional. Its presence is indicated by the value of the presence bit that follows distinguishing bits.
- When the *alignment* option is *byte-alignment* , padding bits of minimum length required to make the whole length of the header byte-aligned are added at the end of the header. On the other hand, there are no padding bits when the alignment in use is *bit-packed*.

[EXI Cookie]	Distinguishing Bits	Presence Bit for EXI Options	EXI Format Version	[EXI Options]	[Padding Bits]

XML EXI

- An EXI header MAY start with an **EXI Cookie** which is a four byte field that serves to indicate that the stream of which it is a part.
- The **Distinguishing Bits**is a two bit field of which the first bit contains the value 1 and the second bit contains the value 0. It is used to distinguish EXI streams from text XML documents in the absence of an EXI cookie.
- **EXI Format Version** identifies the version of the EXI format being used. EXI format version numbers are integers.
- **EXI Options** provides a way to specify the options used to encode the body of the EXI stream (e.g. alignment, compression, schemaid, blcoksize, valueMaxLength)

XML EXI

- The rules for encoding a series of **Events** as an EXI stream are very simple and are driven by a declarative set of grammars that describes the structure of an EXI stream.
- Every event in the stream is encoded using the same set of encoding rules, which are summarized as follows:
  1. Get the next event data to be encoded
  2. If fidelity options indicate this event type is not processed, go to step 1
  3. Use the grammars to determine the *event code* of the event
  4. Encode the *event code* followed by the event content
  5. Evaluate the grammar production matched by the event
  6. Repeat until the *End Document* (ED) event is encoded

RDF Compression

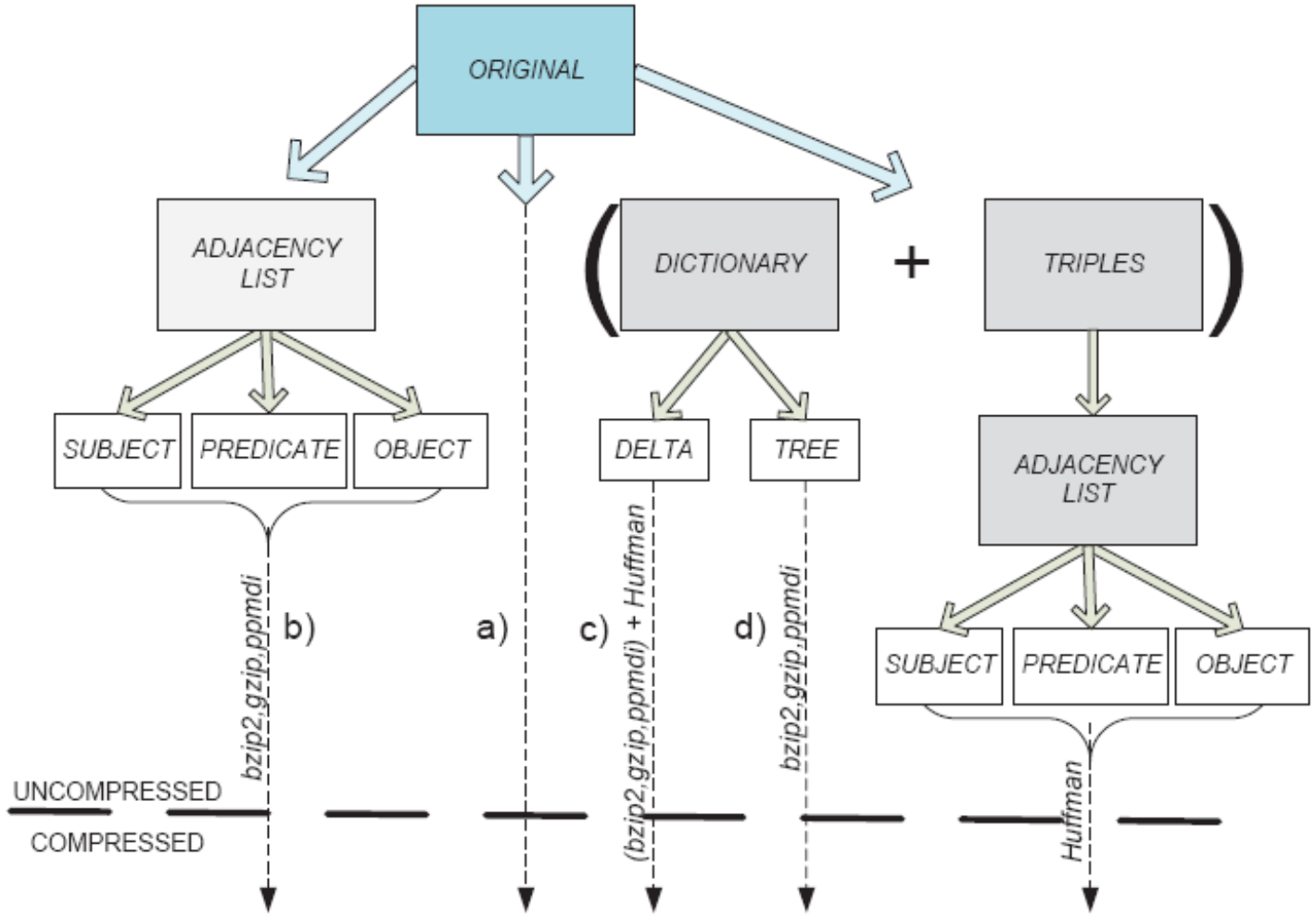
Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

Motivation of RDF Compression

- Large RDF datasets
- Verbosity and Redundancy.
- Examples
  - Billion Triple 2010 (~3200M triples, 318 gzipped chunks, ~27GB)
  - Uniprot (~845M, 12 gzipped chunks, ~23GB)

Approaches of RDF Compression



Javier D. Fernández, Claudio Gutierrez, Miguel A. Martinzez-Prieto. RDF compression: basic approaches. WWW, 2010

Approaches of RDF Compression

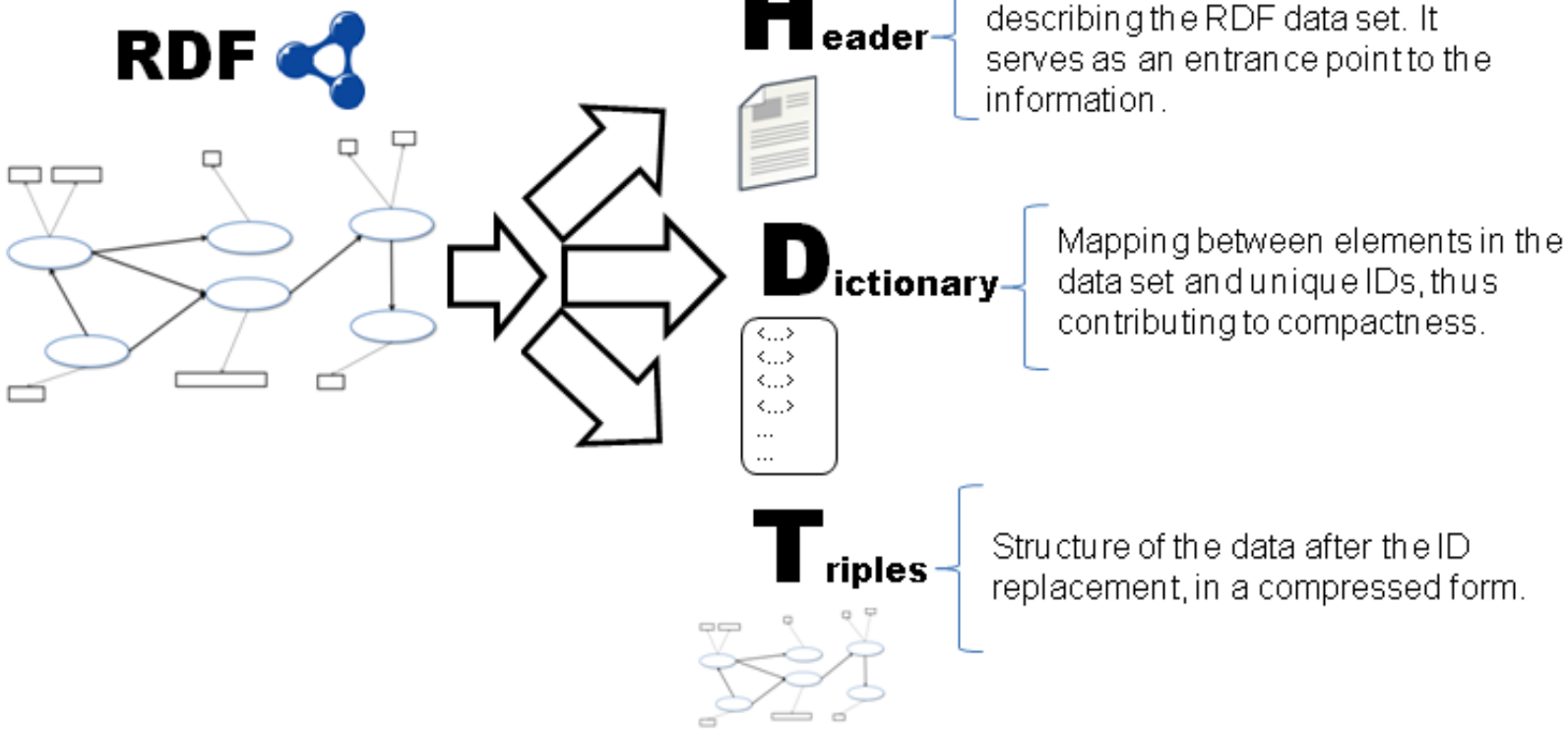
	B.Triples	Uniprot	US Census
Original Size	541.5	239.4	148.2
a) Direct Compr.	37.9 (7.0%)	7.5 (3.1%)	6.4 (4.3%)
b) Adjacency List	35.5 (6.6%)	5.3 (2.2%)	4.8 (3.3%)
c) Delta+Triples	37.9 (7.0%)	9.1 (3.8%)	9.0 (6.1%)
d) Trie+Triples	39.4 (7.3%)	9.7 (4.1%)	9.0 (6.1%)

Approaches of RDF Compression

- RDF data at big scale is highly compressible.
- Dedicated data structures, *e.g.* adjacency lists, code triples efficiently and facilitate compression (both string with ppmdi and integer with Huffman).
- RDF URIs are prone to efficient compression with standard techniques, but compression of literals deserve finer approaches.
- The structure of RDF graphs differs from XML or Web data, hence, classical approaches such as are not directly applicable.

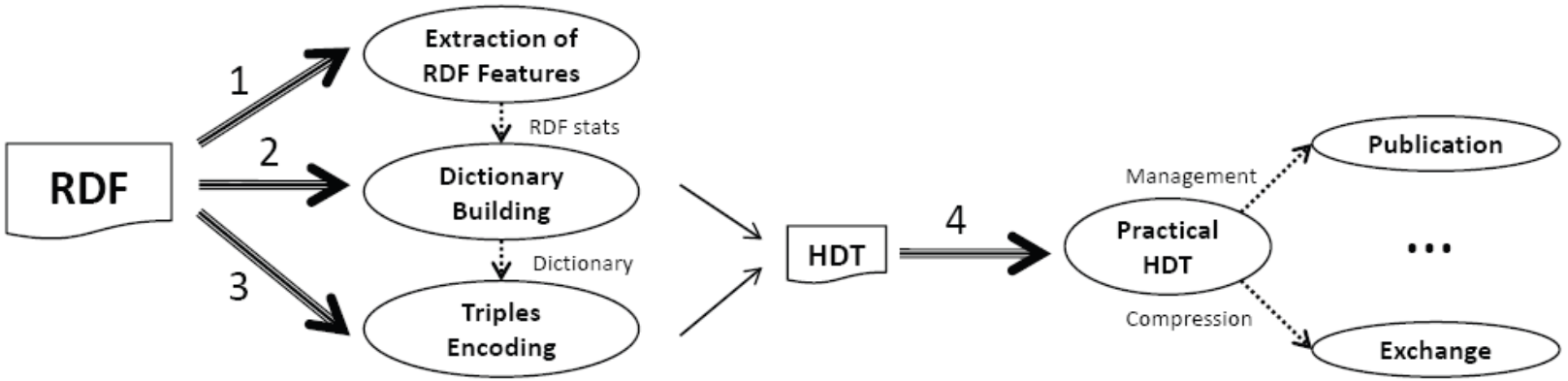
HDT Overview

NICTA

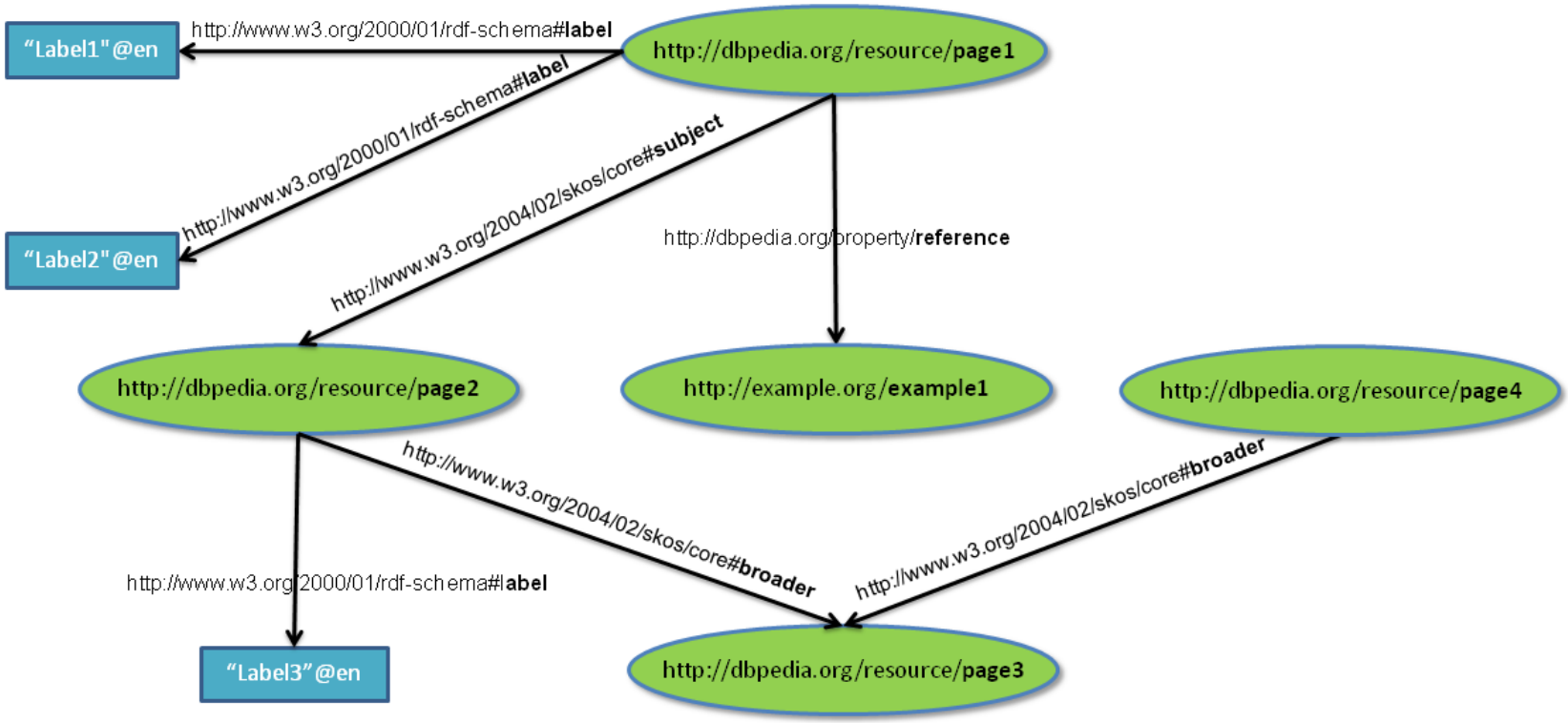


http://www.rdfhdt.org/

HDT Overview



DBPedia Example



DBPedia Header


```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

xmlns:hdt="http://purl.org/HDT/hdt#"
xmlns:dc="http://purl.org/dc/terms/"
xmlns:void="http://rdfs.org/ns/void#"
xmlns:foaf="http://xmlns.com/foaf/0.1/"
xmlns:scovo="http://purl.org/NET/scovo#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:swp="http://www.w3.org/2004/03/trix/swp-2/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

<hdt:Dataset rdf:about="http://example.org/ex/DBpediaEN">
    <hdt:publicationInformation rdf:parseType="Resource">
        <void:dataDump rdf:resource="http://example.org/ex/DBpedia.rdf"/>
        <dc:title>DBpediaEN</dc:title>
        <dc:publisher>
            <foaf:Organization>
                <foaf:homepage rdf:resource="http://example.org/theCompany"/>
            </foaf:Organization>
        </dc:publisher>
        <dc:issued>2010-10-01</dc:issued>
        <dc:source rdf:resource="http://downloads.dbpedia.org/3.5.1/en/"/>
        <dc:license rdf:resource="http://www.gnu.org/copyleft/fdl.html"/>
    </hdt:publicationInformation>

    <hdt:statisticalInformation rdf:parseType="Resource">
        <void:statItem rdf:parseType="Resource">
            <scovo:dimension rdf:resource="void:numberOfTriples"/>
            <rdf:value>7</rdf:value>
        </void:statItem>
        <void:statItem rdf:parseType="Resource">
            <scovo:dimension rdf:resource="hdt:subjectObjectRatio"/>
            <rdf:value>12.5</rdf:value>
        </void:statItem>
    </hdt:statisticalInformation>
```

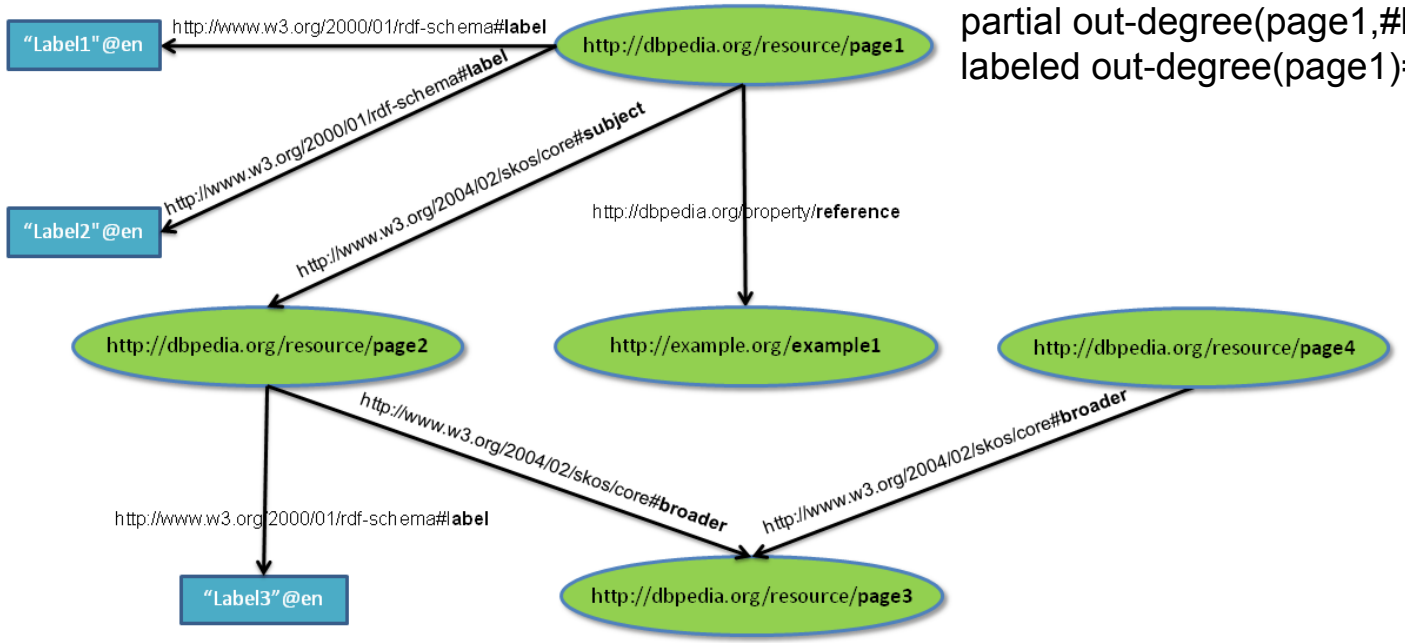
DBPedia Header


```
    <hdt:formatInformation rdf:parseType="Resource">
    <!-- ... -->
    </hdt:formatInformation>
    <hdt:additionalInformation rdf:parseType="Resource">
        <swp:signatureMethod rdf:resource="swp:JjcC14N-md5-xor-rsa"/>
        <swp:signature rdf:datatype="xsd:base64Binary">AZ8QWE...</swp:signature>
    </hdt:additionalInformation>
</hdt:Dataset>
</rdf:RDF>
```

HDT Statistics

- **out-degree**, *deg−(s)*
  - *the number of triples of G in which s occurs as subject*
  - *deg−(G), deg−(G)*
- **partial out-degree**, *deg− −(s, p)*
  - *the number of triples of G in which s occurs as subject and p as predicate*
  - *deg− −(G), deg− −(G)*
- **labeled out-degree**, *degL−(s)*
  - *the number of different predicates (labels) of G with which s is related as a subject in a triple of G*
  - *degL−(G), degL−(G)*
- **subject-object ratio**, *αs−o*
  - *the proportion of common subjects and objects in the graph G*
  - *αs−o = |SG∩OG| / |SG∪OG|*
- Symmetrically, **in-degrees**: *deg+(o), deg+(G), etc.*

DBPedia Example



out-degree(page1) = 4
partial out-degree(page1,#label) = 2
labeled out-degree(page1)=3

out-degree(page2) = 2
labeled out-degree(page2)=2

in-degree(page3) = 2
partial in-degree(page3,#broader) = 2
labeled in-degree(page3)=1

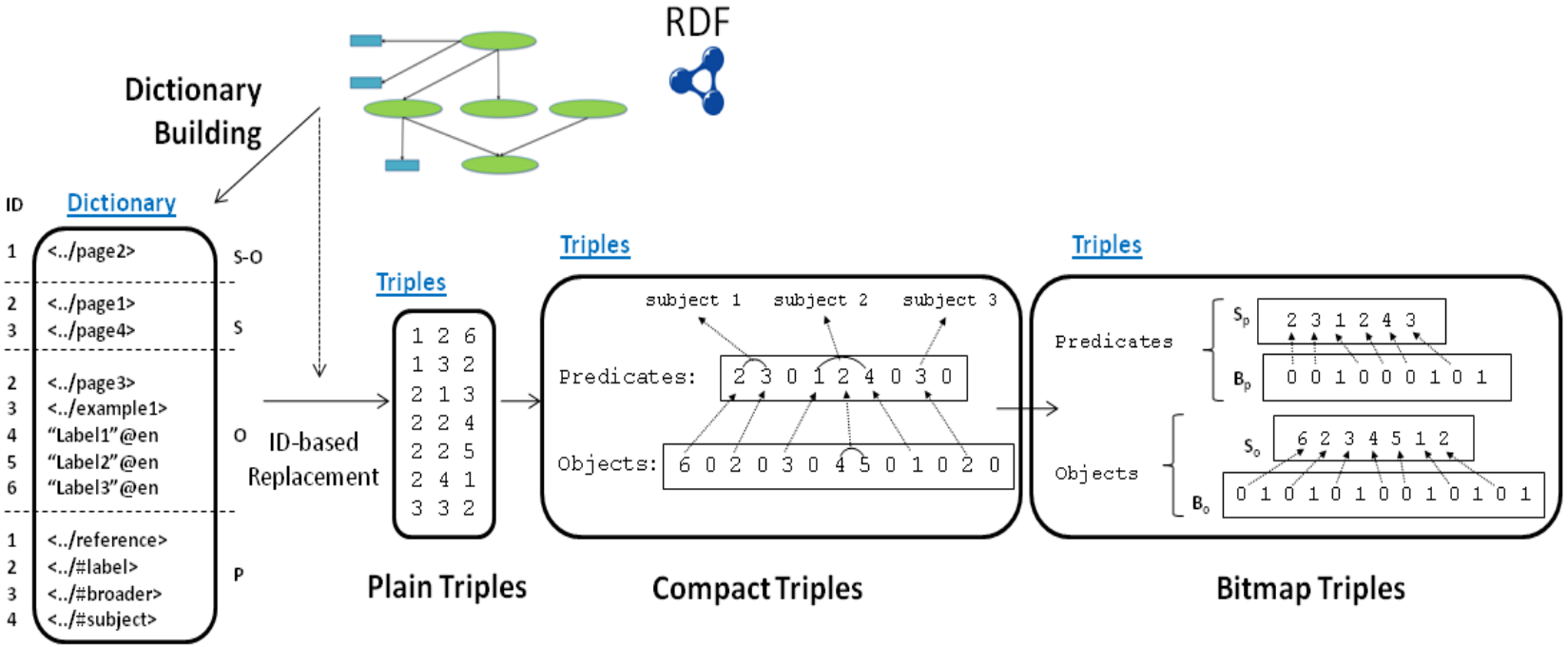
Dictionary in Practice

- Subset distinction:
  - (1) Common **subject-objects**
  - (2) The non common **subjects**
  - (3) The non common **objects**
  - (4) **Predicates**

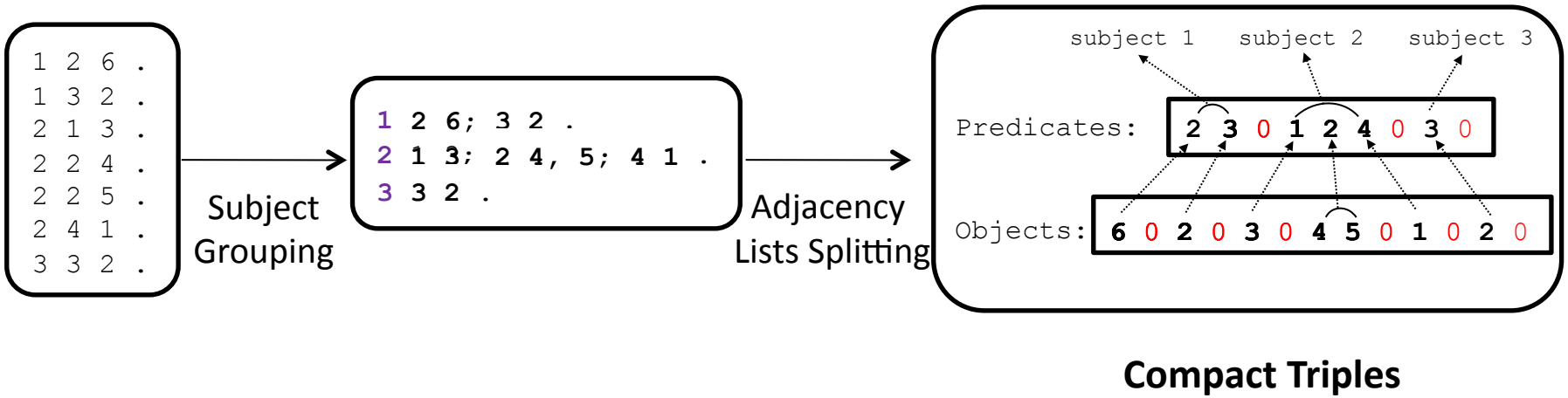
- List of strings matching the mapping of the four subsets, in order from (1) to (4).
  - A reserved character is appended to the end of each string and each vocabulary to delimit their size.

Triples

- Contains the structure of the data after the ID replacement.

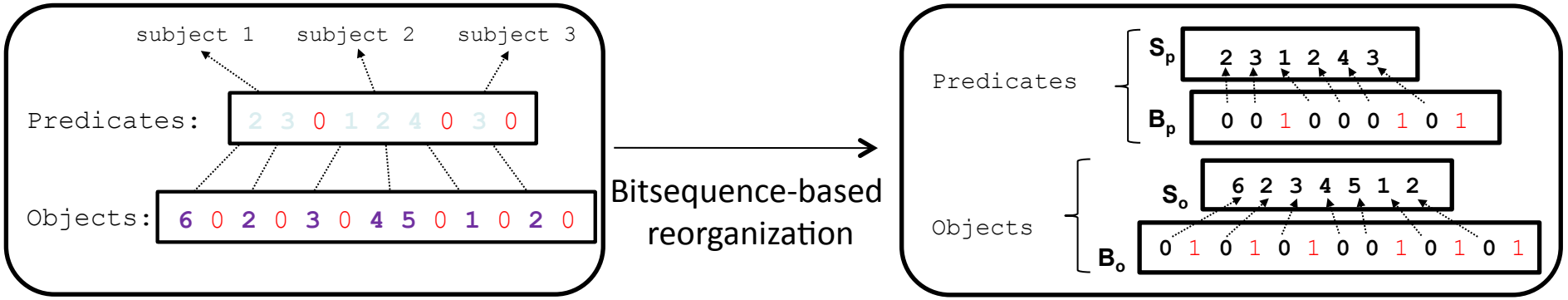


Compact Triples



Compact Triples

Binary Triples



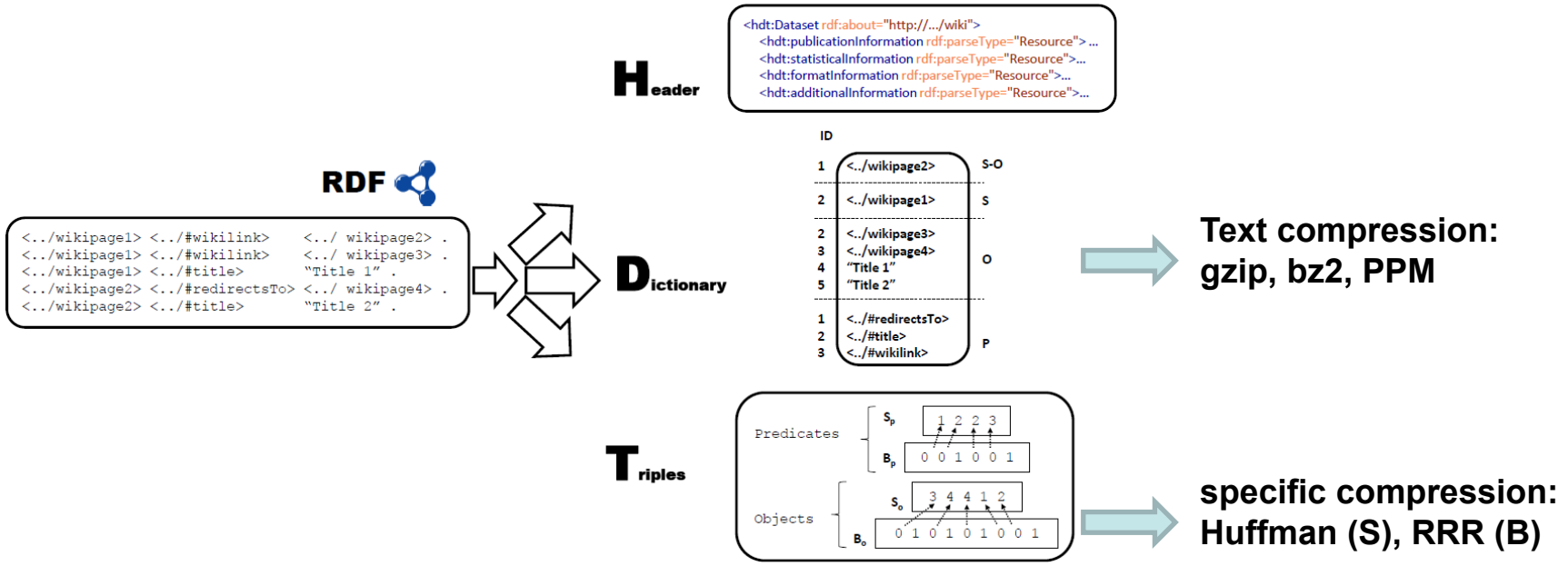
Compact Triples

Bitmap Triples

HDT Bitmap Triples Compression

NICTA

HDT-Plain / HDT-Compress



Javier D. Fernández, Claudio Gutierrez, Miguel A. Martinzez-Prieto. : Compact Representation of Large RDF Data Sets for Publishing and Exchange. ISWC, 2010

RDF Compression

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

Tutorial Overview

Session 1

- XQuery Overview – Sherif
- SPARQL Overview – Axel
- XSPARQL: a combined language – Axel

Session 2

- **XQuery implementations – Sherif**
- **SPARQL implementations – Sherif**
- **XSPARQL implementations – Axel**
- **(optional) Compression formats for XML+RDF: EXI+HDT – Sherif**

- **Q/A - Discussion**