



# XML Access Modules: Towards Physical Data Independence for XML Databases

*(ACI MD TRALALA)*

Date

INSTITUT NATIONAL  
DE RECHERCHE  
EN INFORMATIQUE  
ET EN AUTOMATIQUE



Andrei Arion  
Veronique Benzaken  
**Ioana Manolescu**  
Ravi Vijay

INRIA Futurs and Univ. Paris XI, France  
Univ. Paris XI, France  
INRIA Futurs, France  
IIT Bombay, India

# Contexte: ACI TRALALA

TRAnsformations, Logiques et LAngages

Theorie et pratique de l'interrogation de donnees XML:

automates d'arbres

langages fonctionnels, typage semantique (CQL et CDuce)

complexite de (sous-ensembles de) XPath, XQuery

compression XML

traitements XML a la volee

optimisation algebrique

optimisation de chemin d'accès aux données

# Plan

The need for physical data independence in XML databases

Our proposal: XML Access Modules (XAMs)

Algebraic language describing XML materialized views  
and indices

Answering XQueries over XAMs

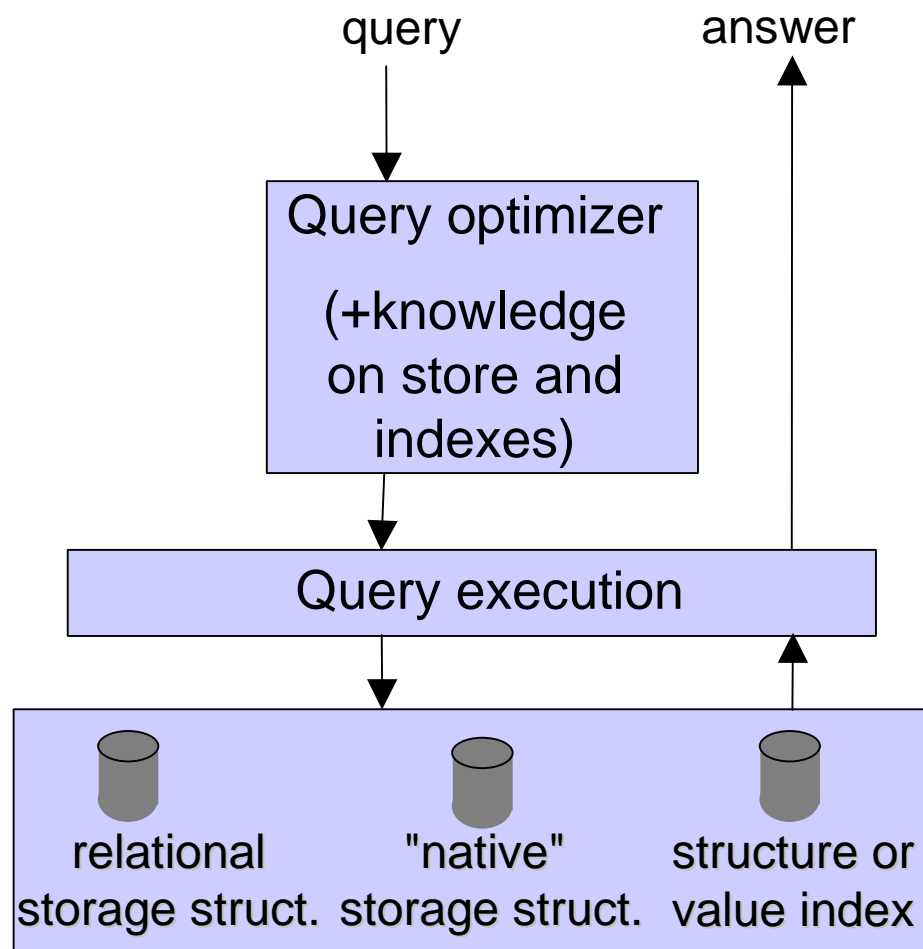
Constraint-based containment and rewriting

Outline of a XAM-based XML DBMS architecture

Conclusion

# The need for physical data independence in XML databases

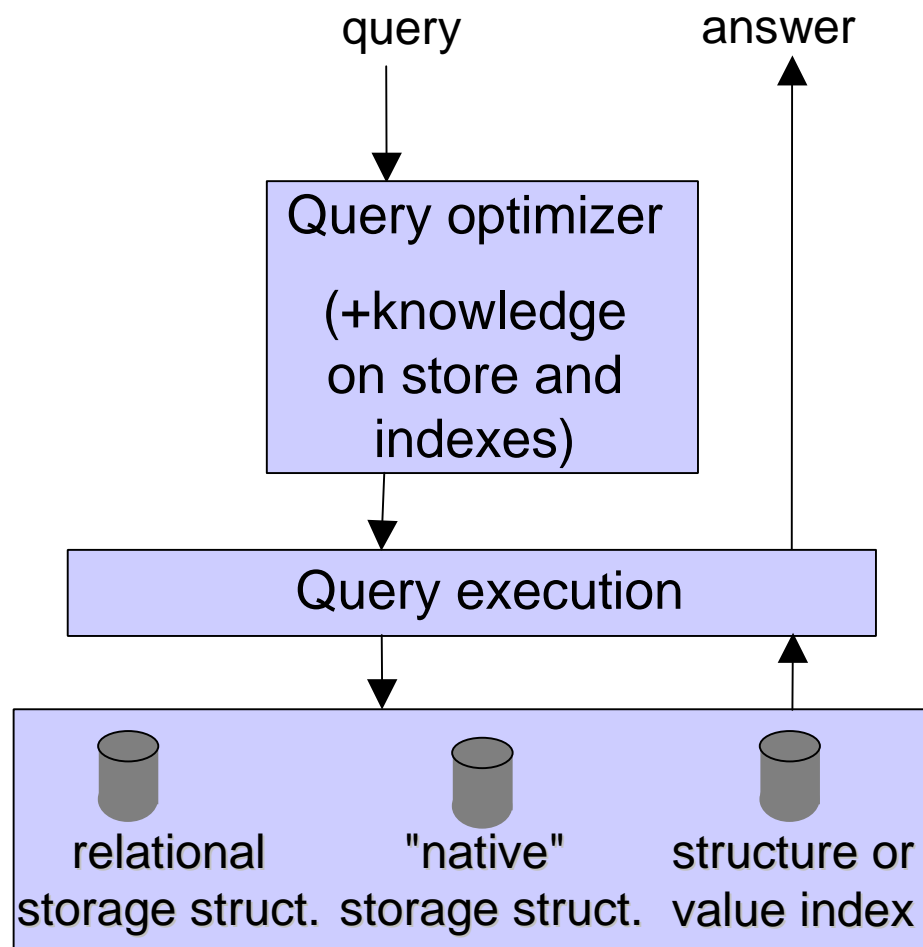
# Query processing on XML persistent stores



Many existing storage and indexing models for XML

Different applications and data sets call for different storage structures

# Query processing on XML persistent stores



Many existing storage and indexing models for XML

Different applications and data sets call for different storage structures

Rewriting the optimizer for every new storage model not an option

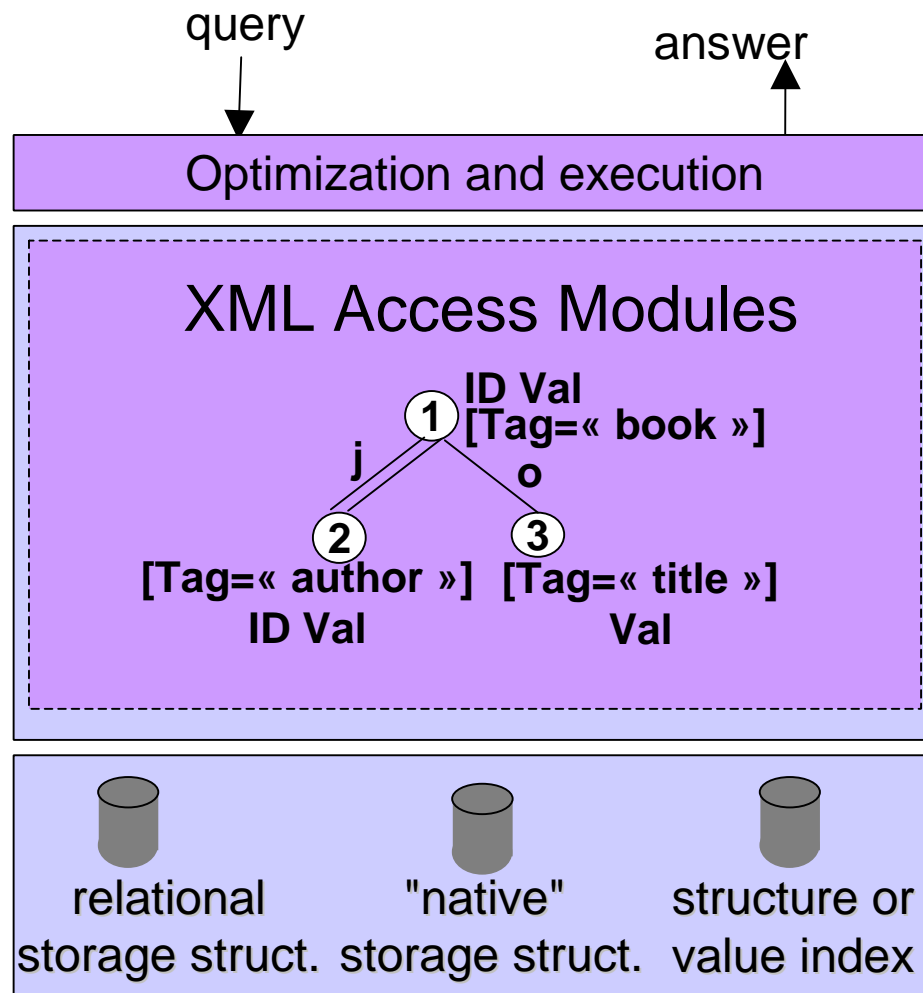
**We need:**

High-level language for describing disk-resident storage structures

Algorithms for query answering

**Physical data independence**

# Query processing on XML persistent stores



Many existing storage and indexing models for XML

Different applications and data sets call for different storage structures

Rewriting the optimizer for every new storage model not an option

**We need:**

High-level language for describing disk-resident storage structures

Algorithms for query answering

**Physical data independence**

# Related problems

## XML query cache

Several data fragments have been loaded in cache by previous queries

Problem: answer a new query based on cache fragments ?

## XML database self-tuning

Given a document and some queries, which indexes/materialized views to store to improve performance ?

## Local-as-view data integration

Several data sources, each storing part of a global dataset

Problem: answer a query over the global data by combining data sources ?



# XML Access Modules (XAMs)

# A language for XML materialized views and indexes

## Granularity

Small: values, persistent IDs

Large: full subtrees

## Organization

Trees seem natural

Nested tuples have clean algebraic foundations

## Specification

Value and structure conditions (~ selections)

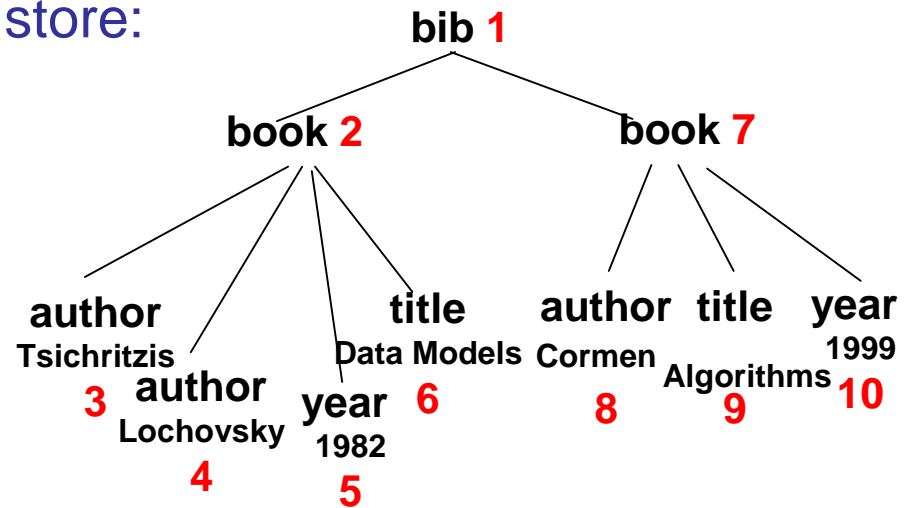
Value and structure information (~ projections)

# XML Access Modules (XAMs)

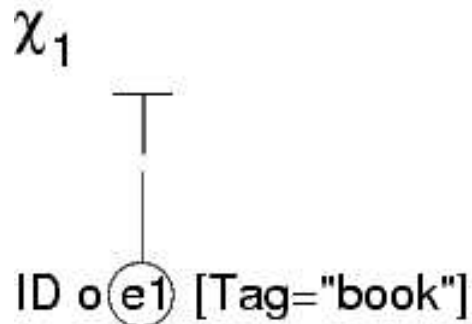
Tree-based language; nested tuple semantics

For any node, the XAM may store:

- ID
- Tag
- Val
- Cont



X1 tuples : [ (2)  
(7) ]

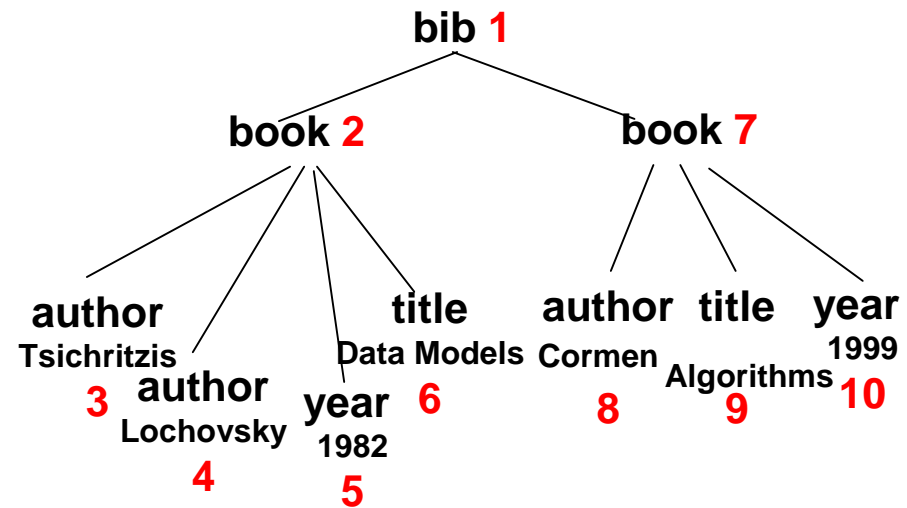


# XML Access Modules (XAMs)

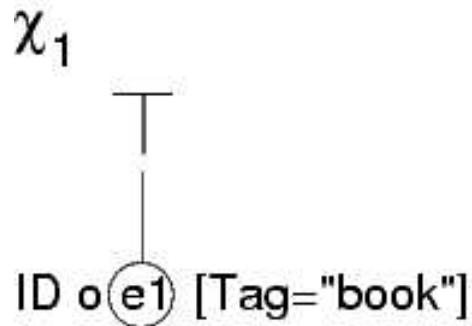
Tree-based language; nested tuple semantics

ID: i ID

- o order-preserving
- s structural
- n upward navigable
- u update resilient

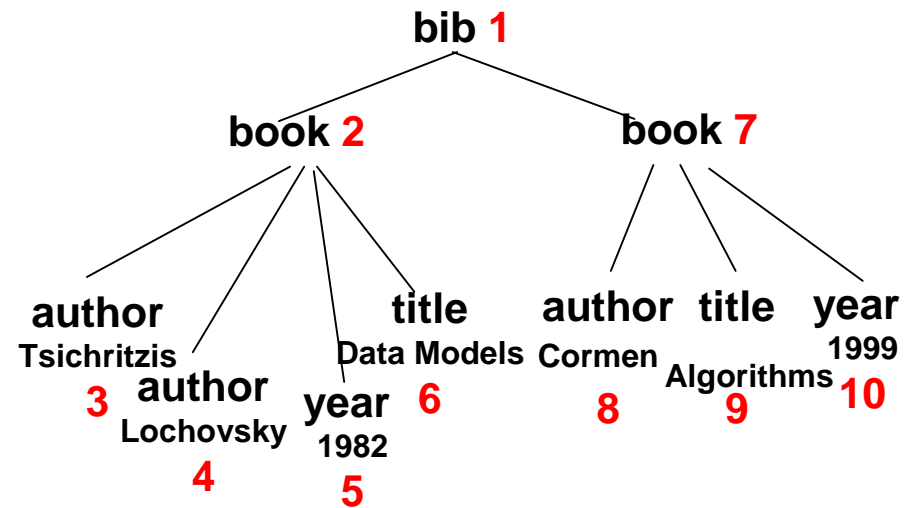
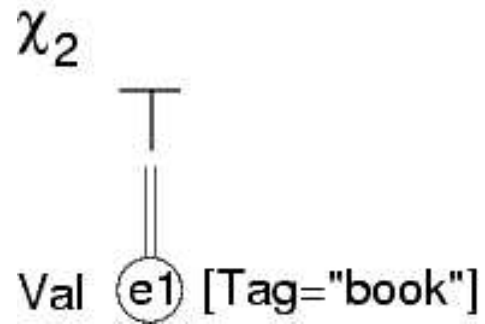


X1 tuples : [ (2)  
(7) ]



# XML Access Modules (XAMs)

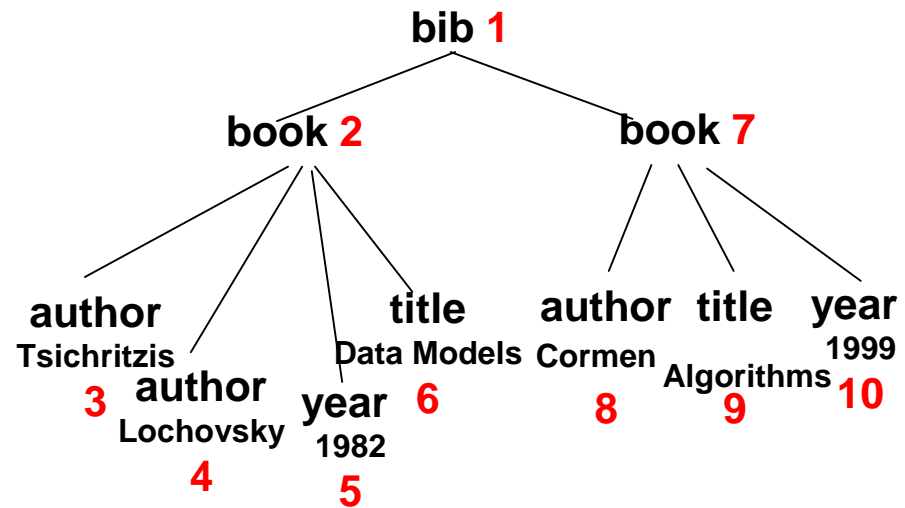
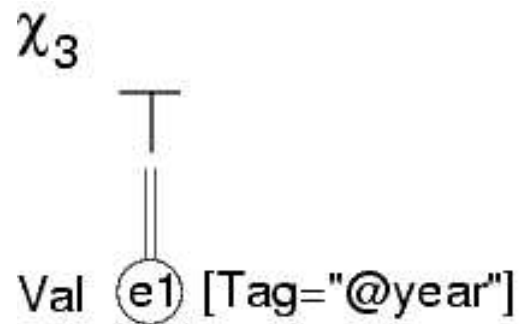
Tree-based language; nested tuple semantics



X2 tuples : [ ( $\perp$ )  
( $\perp$ ) ]

# XML Access Modules (XAMs)

Tree-based language; nested tuple semantics



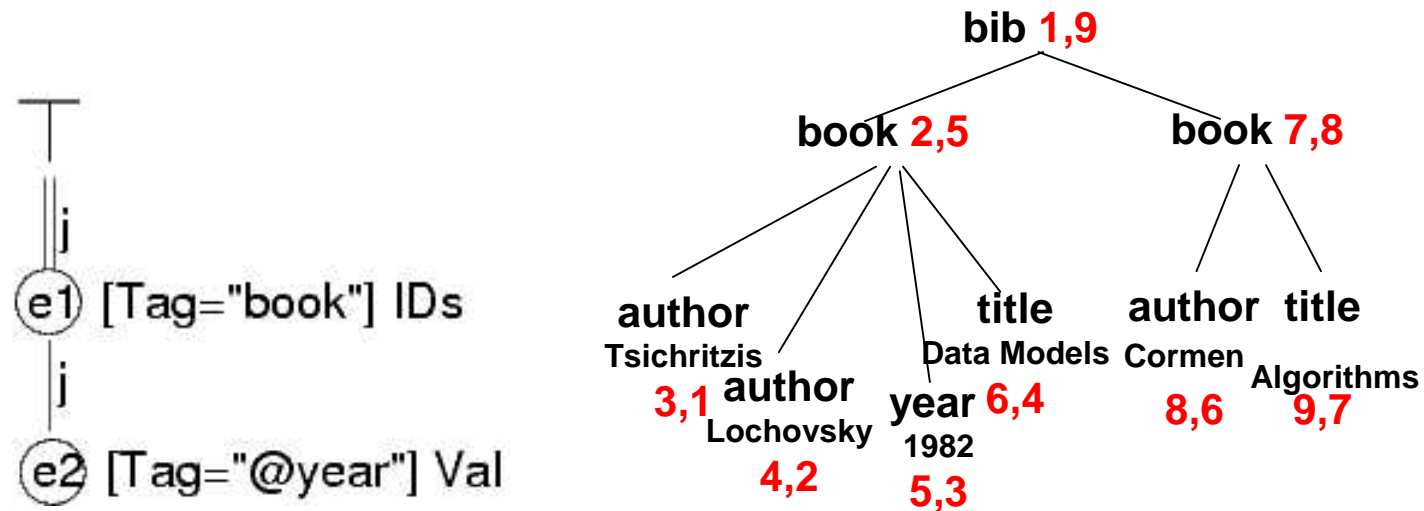
X3 tuples: [ ("1982")  
("1999") ]



# XML Access Modules (XAMs)

Tree-based language; nested tuple semantics

$\chi_5$



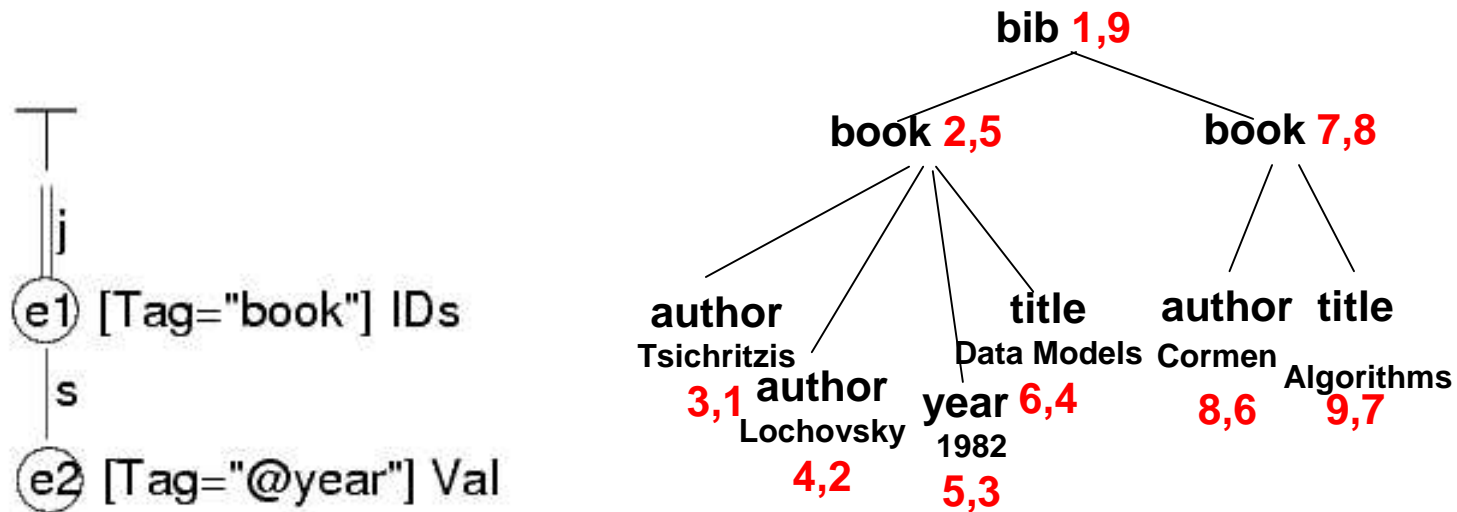
X5 tuples: [ ((2,5), "1982") ]



# XML Access Modules (XAMs)

Tree-based language; nested tuple semantics

$\chi_5$

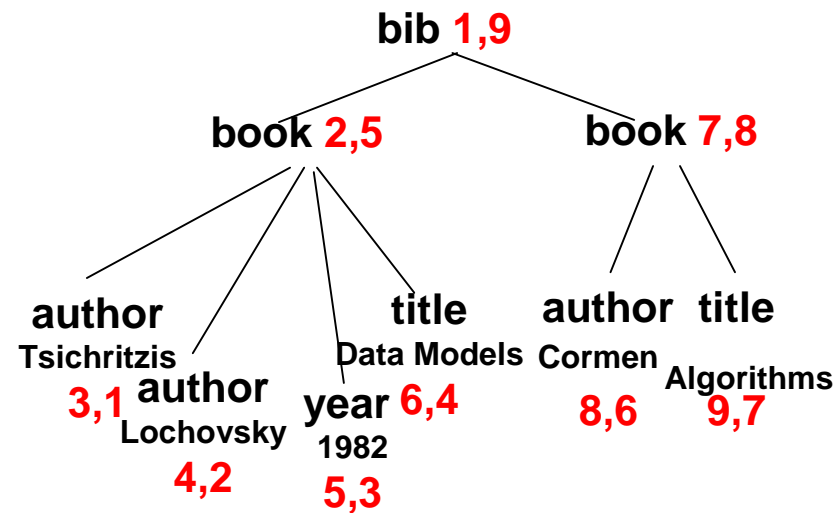
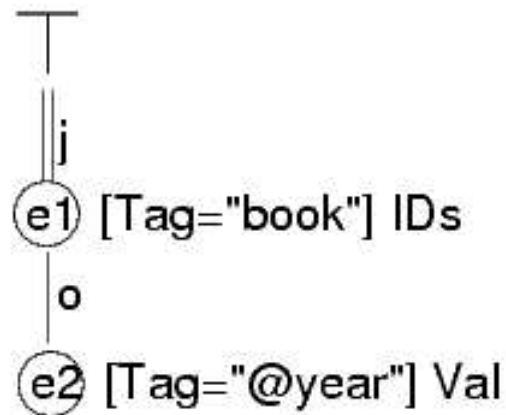


X5 tuples: [ ((2,5)) ]

# XML Access Modules (XAMs)

Tree-based language; nested tuple semantics

$\chi_5$

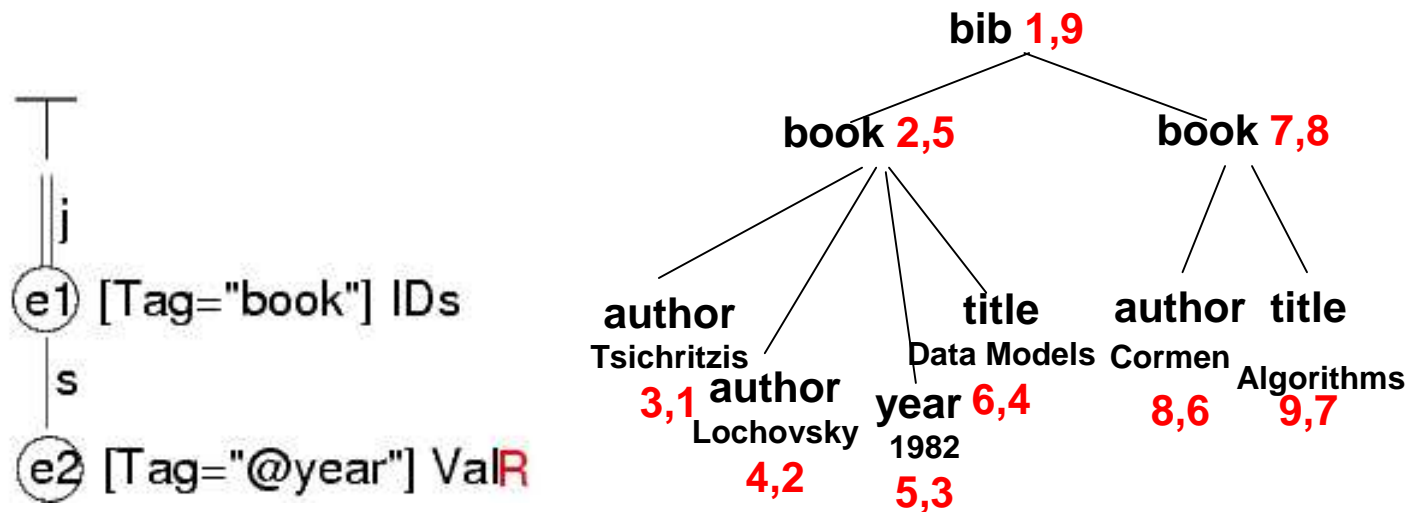


X5 tuples: [ ((2,5), "1982")  
((7,9),  $\perp$ ) ]

# XML Access Modules (XAMs)

Tree-based language; nested tuple semantics

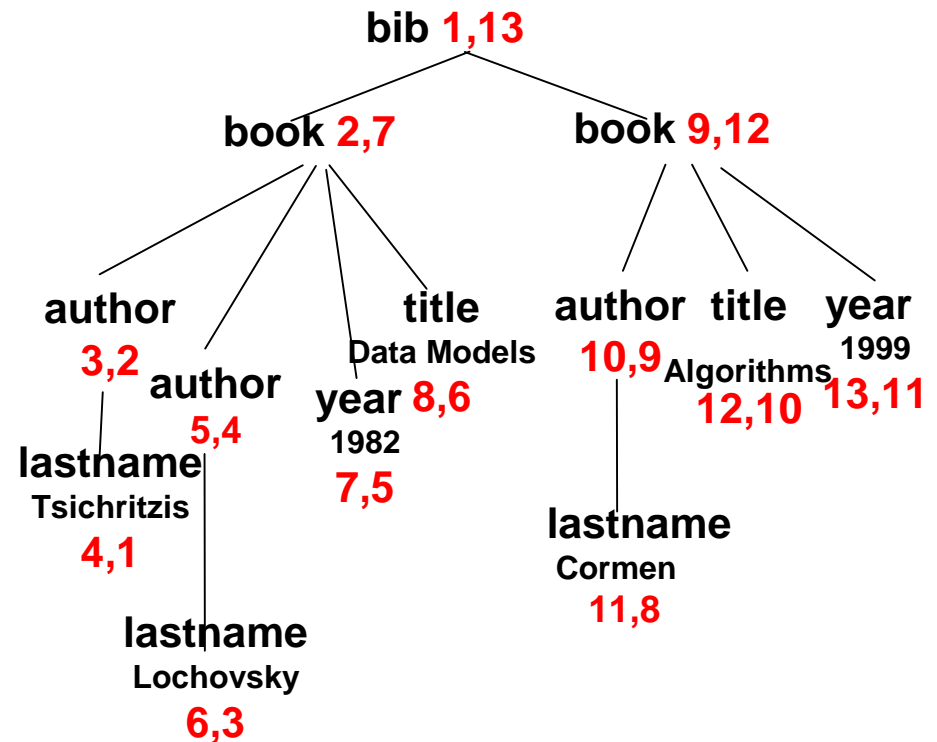
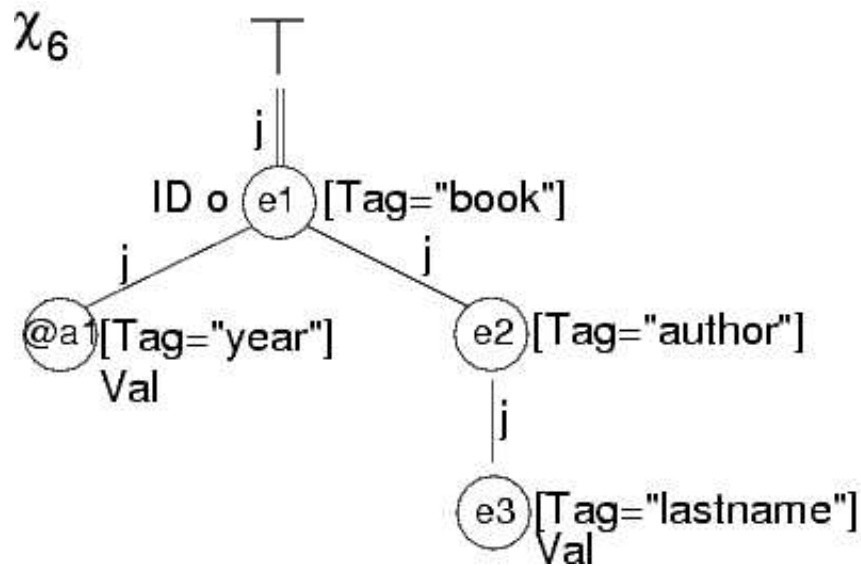
$\chi_5$



X5 contents: "1982" --> [ ((2,5), "1982") ]

# XML Access Modules (XAMs)

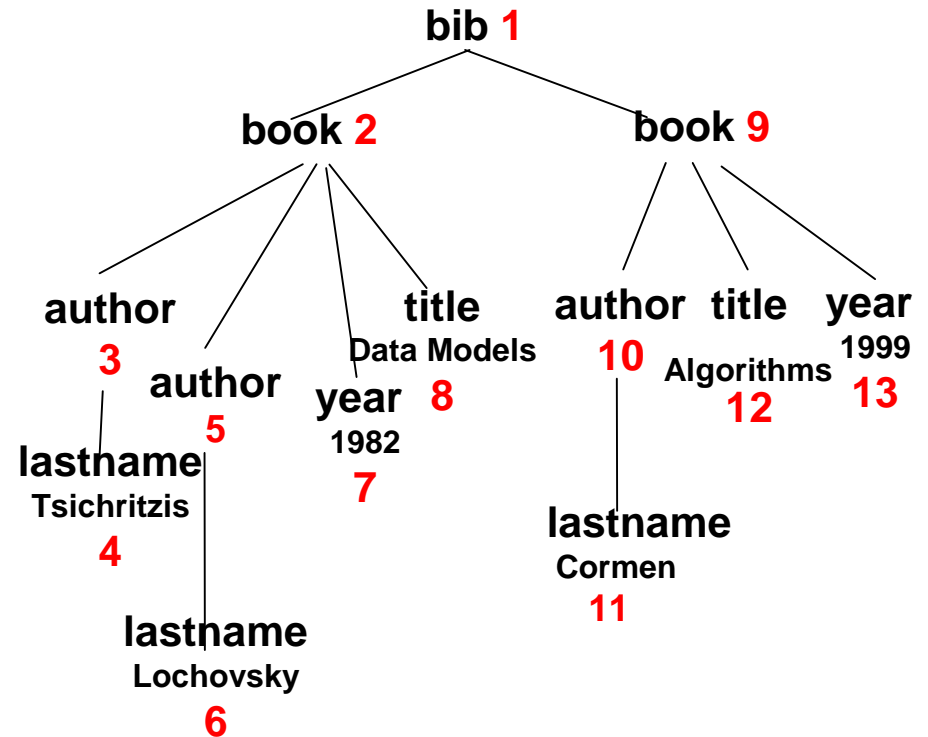
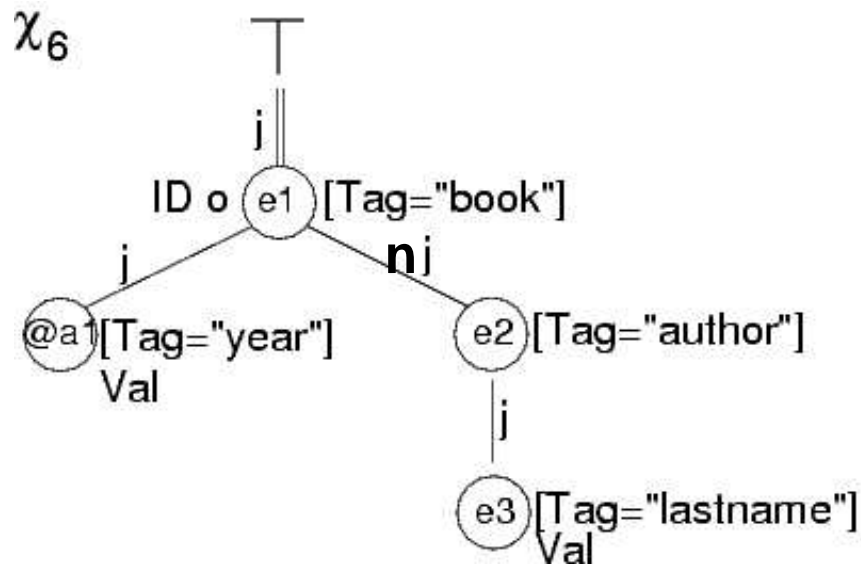
Tree-based language; nested tuple semantics



X6 tuples: [ (2, "1982", "Tsichritzis")  
 (2, "1982", "Lochovsky")  
 (9, "1999", "Cormen") ]

# XML Access Modules (XAMs)

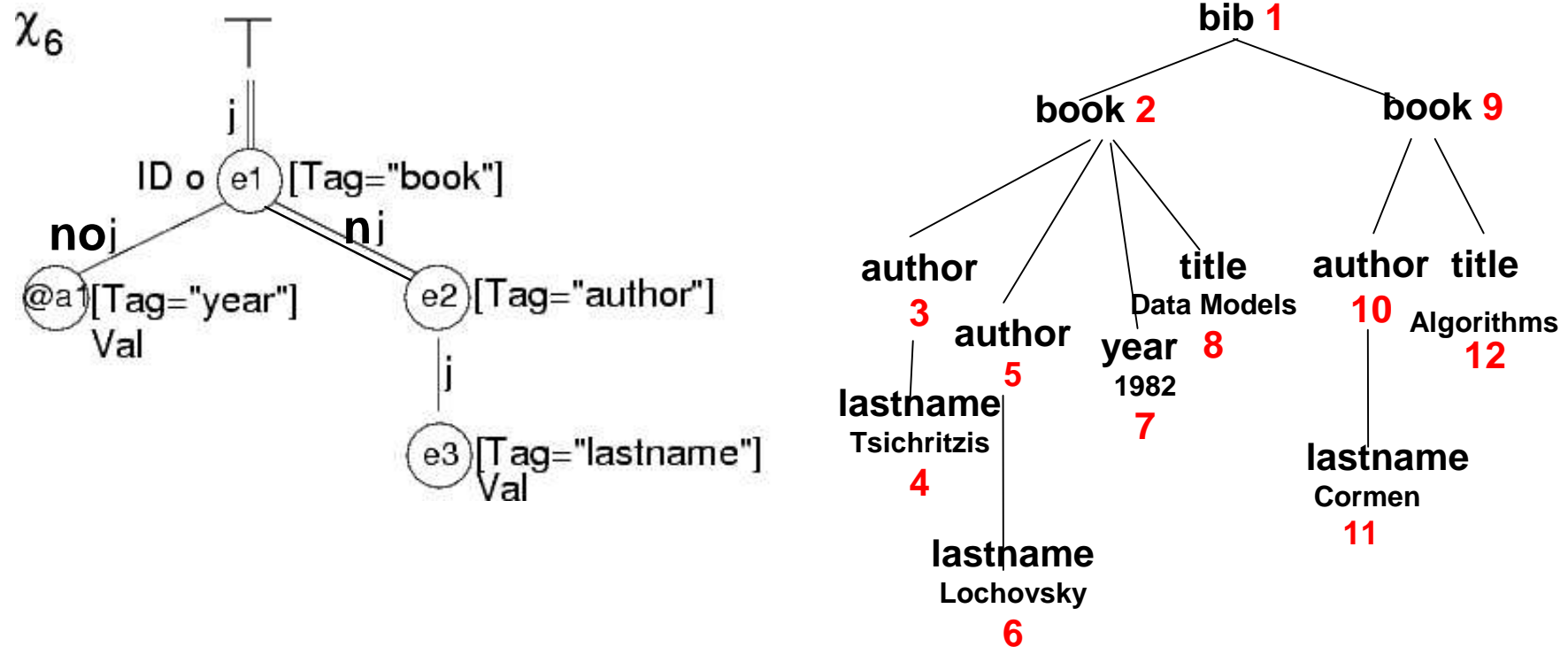
Tree-based language; nested tuple semantics



X6 tuples: [ (2, "1982", [ ("Tsichritzis"), ("Lochovsky") ] )  
 (9, "1999", [ ("Cormen") ] ) ]

# XML Access Modules (XAMs)

Tree-based language; nested tuple semantics

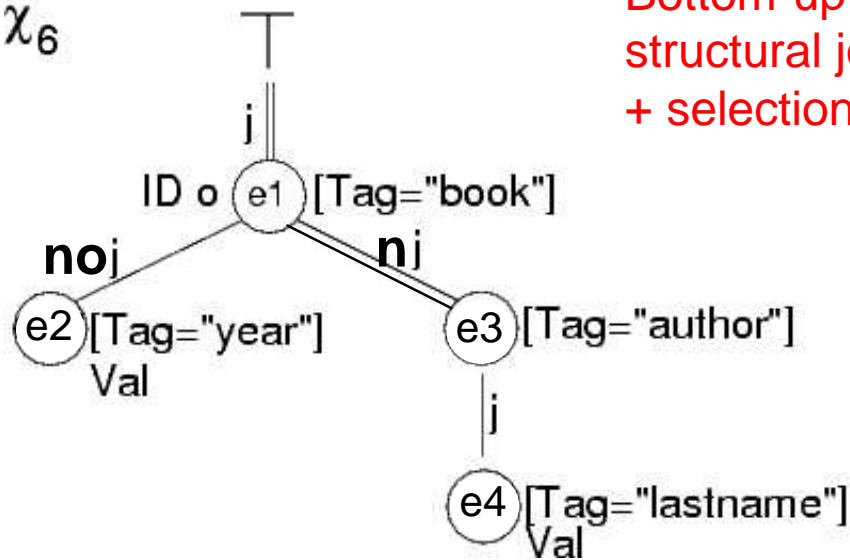


X6 tuples: [ (2, [ ("1982") ], [ ("Tsichritzis") ("Lochovsky") ] )  
 (9, [ ], [ ("Cormen") ] ) ]

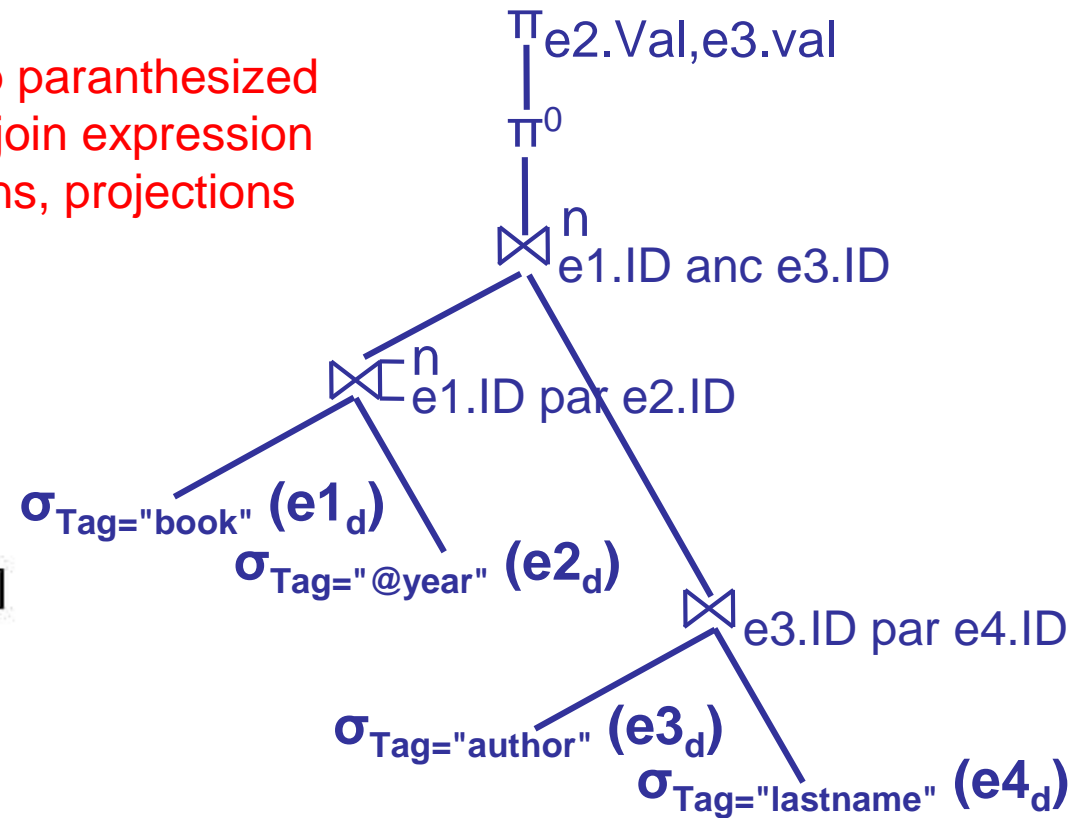
# XAM semantics

For a document  $d$ , consider basic relation  $e_d(ID, Tag, Val, Cont)$

$\chi_6$



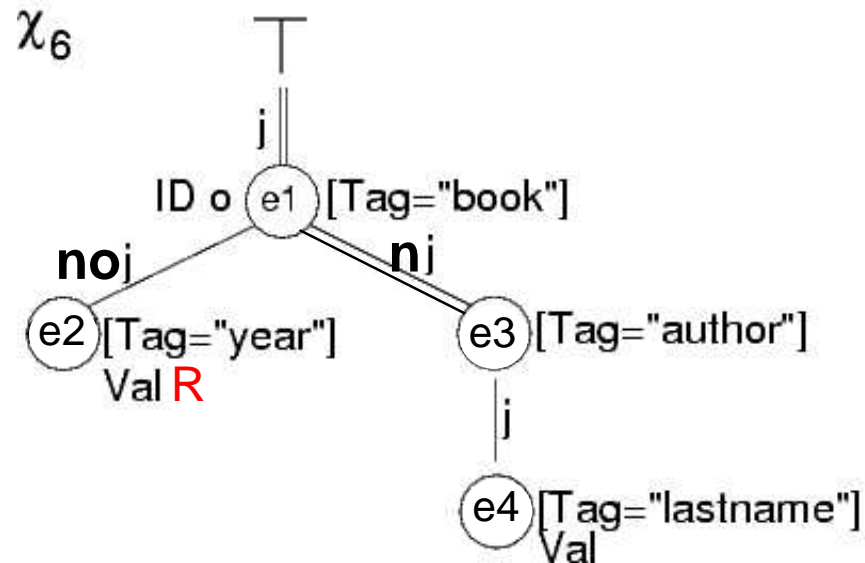
Bottom-up paranthesized structural join expression + selections, projections



# XAM semantics with access restrictions

Let  $X_0$  be the XAM obtained from  $X$  by removing all **R** annotations

Let  $t_0$  be a tuple of bindings for the **R**-annotated attributes



Content of  $X$  with bindings  $t_0 =$

$$\sigma_{R\text{attrs}=t_0}(\text{content of } X_0)$$



# XAM generality

Capture many of the XML fragmentation schemes previously proposed for storage and indexes

Tag and path partitioning, "Shared", 1-index, F-index...

Also capture original nesting !!!

Do not capture

Other navigation axes: "all a elements with their b siblings"

Negation (antijoins): "all a elements without a b child"

Value joins across unrelated elements

Restructuring

# Answering queries over XAMs

# Problem statement

Input: a query  $Q$  and a set of XAMs  $X_1, X_2, \dots, X_n$

Output: all algebraic expressions  $e(X_1, X_2, \dots, X_n)$  such that  
for any document  $d$ ,  $Q(d) = e(X_1, X_2, \dots, X_n)(d)$

Algebraic expression ingredients:

scan( $X_i$ ),  $\sigma, \pi$

$\bowtie_{\text{par/anc}}$   $\bowtie_{\text{pred}}$   $\bowtie_{\text{par/anc}}^{\sqsubset}$   $\bowtie_{\text{pred}}^{\sqsubset}$   $\bowtie_{\text{par/anc}}$   $\bowtie_{\text{pred}}$   
 $\bowtie_{\text{par/anc}}^n$   $\bowtie_{\text{pred}}^n$   $\bowtie_{\text{par/anc}}^n$   $\bowtie_{\text{pred}}^n$

navigation in XML serialized trees (Cont attributes)

# Problem statement

Input: a query  $Q$  and a set of XAMs  $X_1, X_2, \dots, X_n$

Output: all algebraic expressions  $e(X_1, X_2, \dots, X_n)$  such that  
for any document  $d$ ,  $Q(d) = e(X_1, X_2, \dots, X_n)(d)$

Remark: if  $e_1 = X_1 \bowtie X_2$  is equivalent to  $Q$ , then also  
 $e_2 = X_2 \bowtie X_1$  is equivalent to  $Q$

# Problem statement

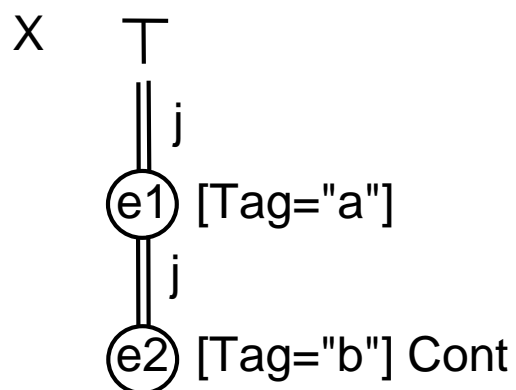
Input: a query  $Q$  and a set of XAMs  $X_1, X_2, \dots, X_n$

Output: all algebraic expressions  $e(X_1, X_2, \dots, X_n)$

(up to algebraic equivalence)

such that for any document  $d$ ,  $Q(d) = e(X_1, X_2, \dots, X_n)(d)$

Remark: consider the XAM  $X$  and queries  $Q1 = //a//b$ ,  $Q2 = //b$



$X$  can be used for  $Q1$  in general.

$X$  can also be used for  $Q2$  if all  $b$  elements have an  $a$  ancestor

# Problem statement

Input: a query  $Q$  and a set of XAMs  $X_1, X_2, \dots, X_n$

structural constraints on the document used by  $Q$

Output: all algebraic expressions  $e(X_1, X_2, \dots, X_n)$

(up to algebraic equivalence)

such that for any constr'd. doc.  $d$ ,  $Q(d) = e(X_1, X_2, \dots, X_n)(d)$

Remark: we are interested in plans that can actually be translated into executable ones

$\bowtie_{\text{par/anc}}$   $\bowtie_{\text{par/anc}}^{\square}$   $\bowtie_{\text{par/anc}}$   $\bowtie_{\text{par/anc}}^n$   $\bowtie_{\text{par/anc}}^{\square n}$  only on ID s,n

Navigation only on Cont attributes

# Problem statement

Input: a query  $Q$  and a set of XAMs  $X_1, X_2, \dots, X_n$

structural constraints on the document used by  $Q$

Output: all valid algebraic expressions  $e(X_1, X_2, \dots, X_n)$

(up to algebraic equivalence)

such that for any constr'd. doc.  $d$ ,  $Q(d) = e(X_1, X_2, \dots, X_n)(d)$

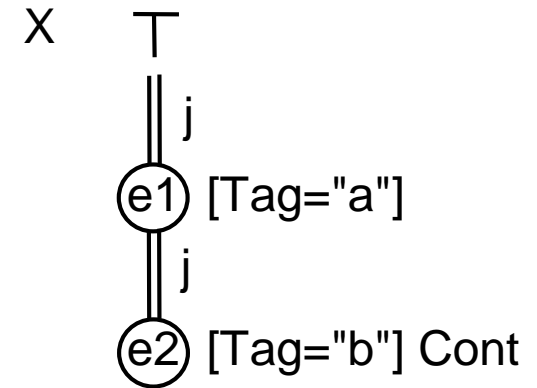
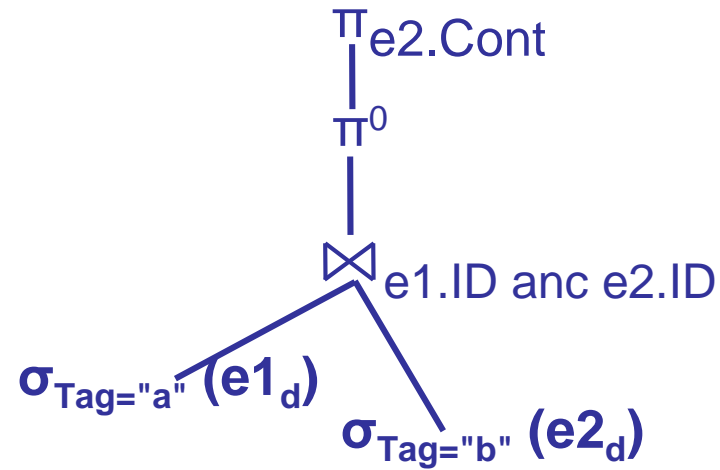
# Rewriting algorithm outline

1. Construct algebraic expressions from  $Q$ 
  - Downward XPath leads to XAM-like algebraic expressions
  - Downward XQuery leads to a join over several XAM-like algebraic expressions  $XQ_1, XQ_2, \dots, XQ_m$ .
2. Inject constraint information into  $Q$  and  $X_1, X_2, \dots, X_n$ .
3. For each  $XQ_i$ 
  - 3.1 Construct gradually larger algebraic expressions starting from  $\pi(\text{Scan}(X_j))$ , until current expr. is contained in  $XQ_i$ .
  - 3.2 Cover  $XQ_i$  with unions of contained rewritings
4. Combine all rewritings for  $XQ_1, XQ_2, \dots, XQ_m$  via joins

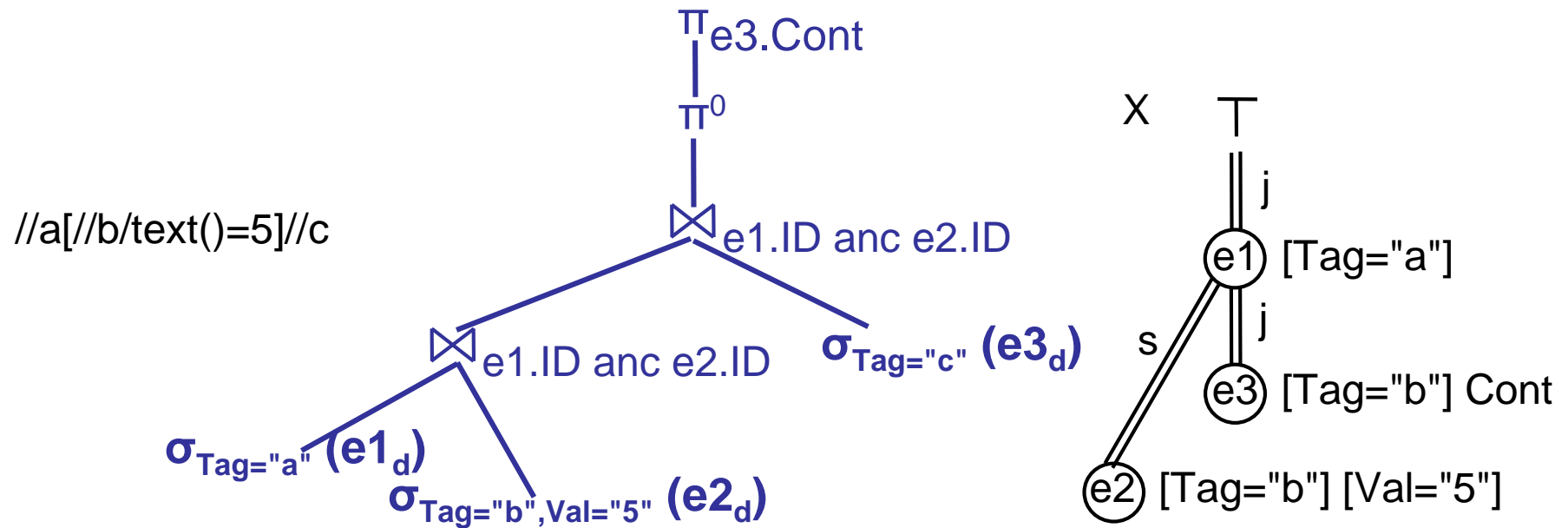


# From XPath to algebra

//a//b



# From XPath to algebra

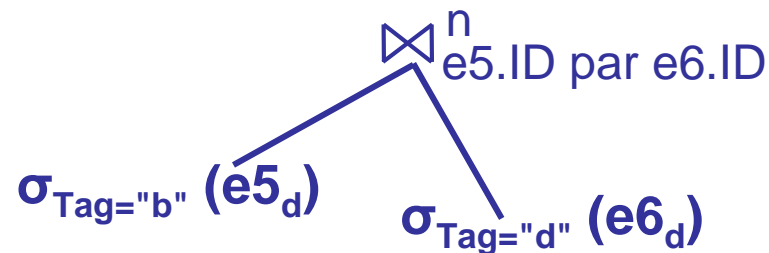
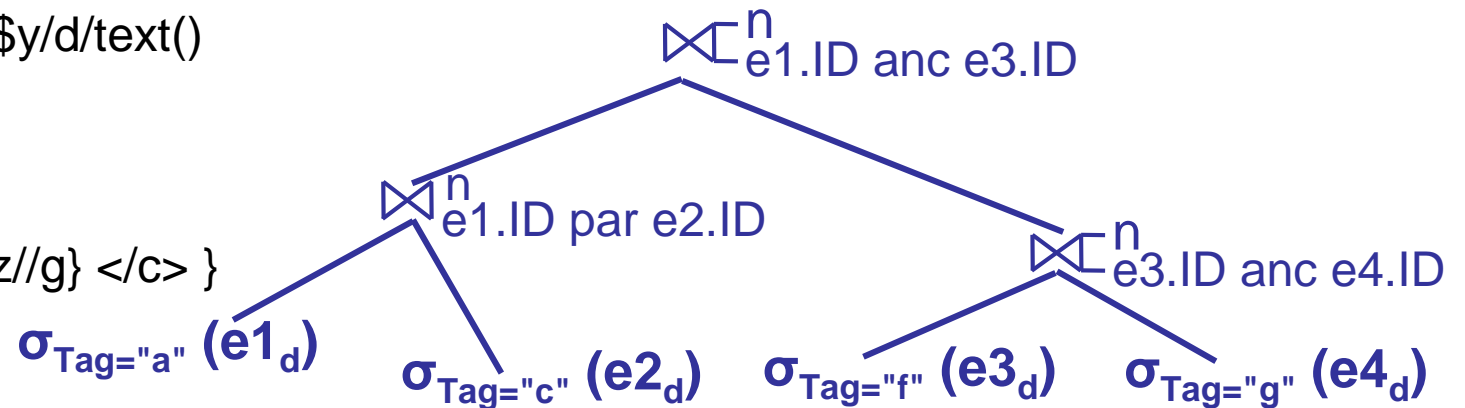


# From XQuery to algebra

```

for $x in //a, $y in //b
where $x/c/text()=$y/d/text()
return
<new>
  { for $z in $x//f
    return <c> {$z//g} </c> }
</new>

```



Join:  
e2.Val=e6.val

# Injecting constraints in views and queries

Annotate views and query with information allowing to infer which view nodes may bring information for which query node

E.g. `//*[inproceedings]` is the same as `//article`

`//*[inproceedings]` is the same as `//*[booktitle]`

We used enhanced DataGuides as constraints. Schemas also apply.

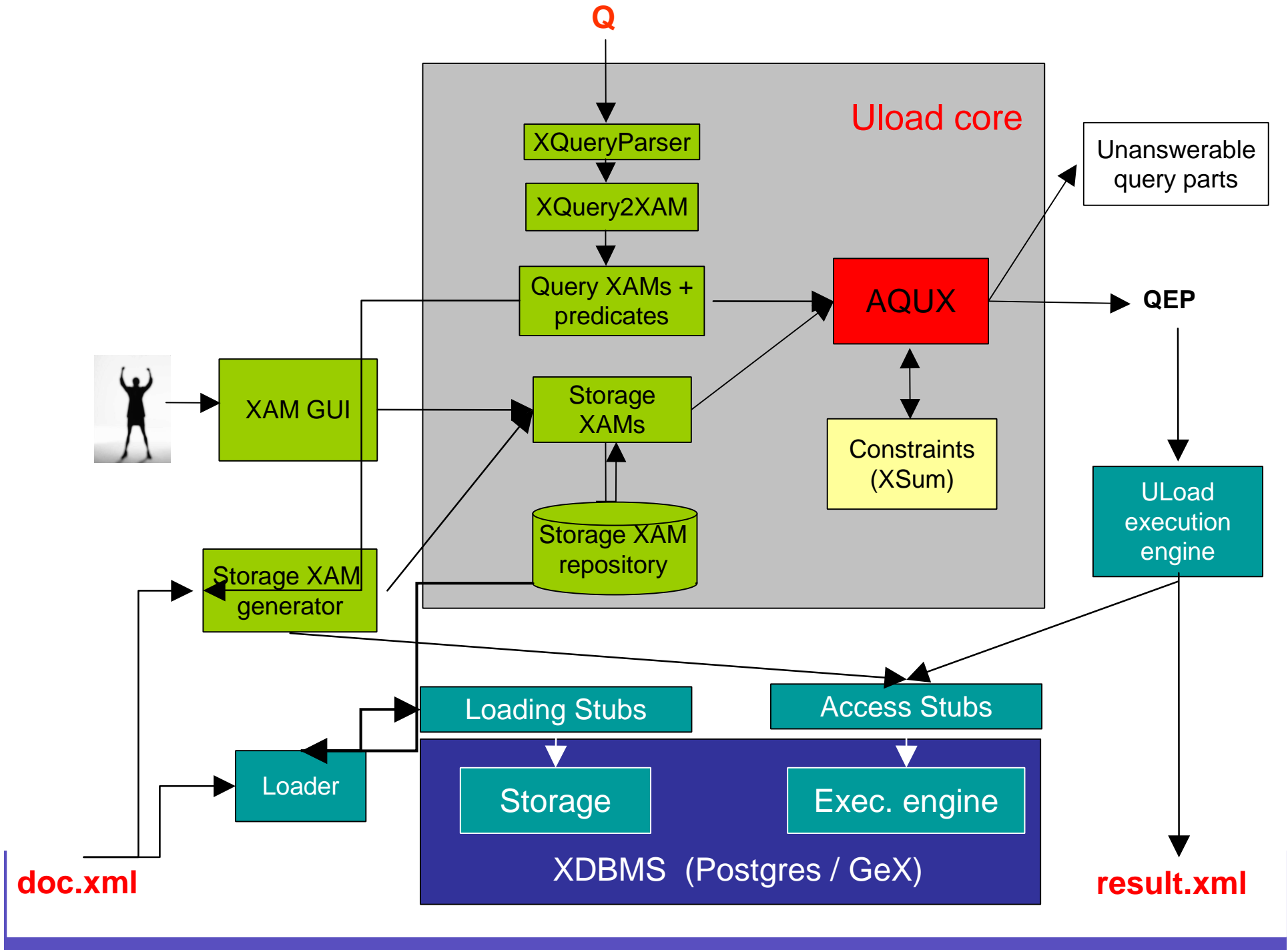
In general, constraints allow to:

- Find more rewritings

- Avoid empty-result rewritings

- Find more efficient algebraic rewritings

# ULoad: a materialized view management tool based on XAMs



# ULoad prototype demonstration

XML materialized view management for XQuery:

Materialized view creation

Data extraction & loading in native/relational repository

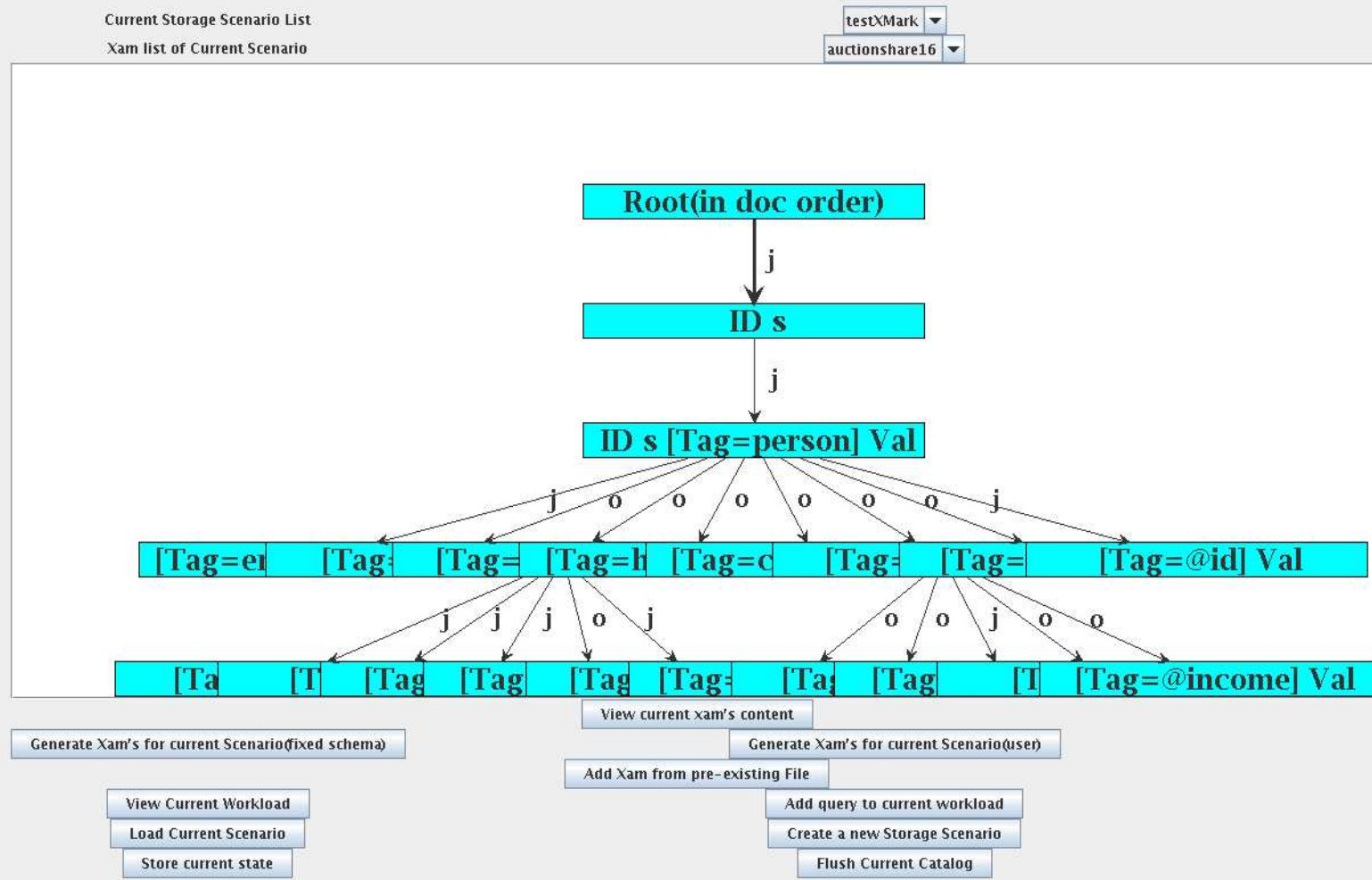
Query answering over the materialized views

Materialized view extraction from XQuery queries

Also:

Guidance in choosing views and writing queries:  
satisfiability / answerability tests

# Formalism for describing complex XML materialized views: XAMs





# Loading XAMs in a store

ioana@localhost:~

Current Storage Scenario

Table "xamtest.xam_auctionshare15"		
Column	Type	Modifiers
id1pre	integer	
id1post	integer	
id1depth	integer	
id2pre	integer	
id2post	integer	
id2depth	integer	
val2	character varying	

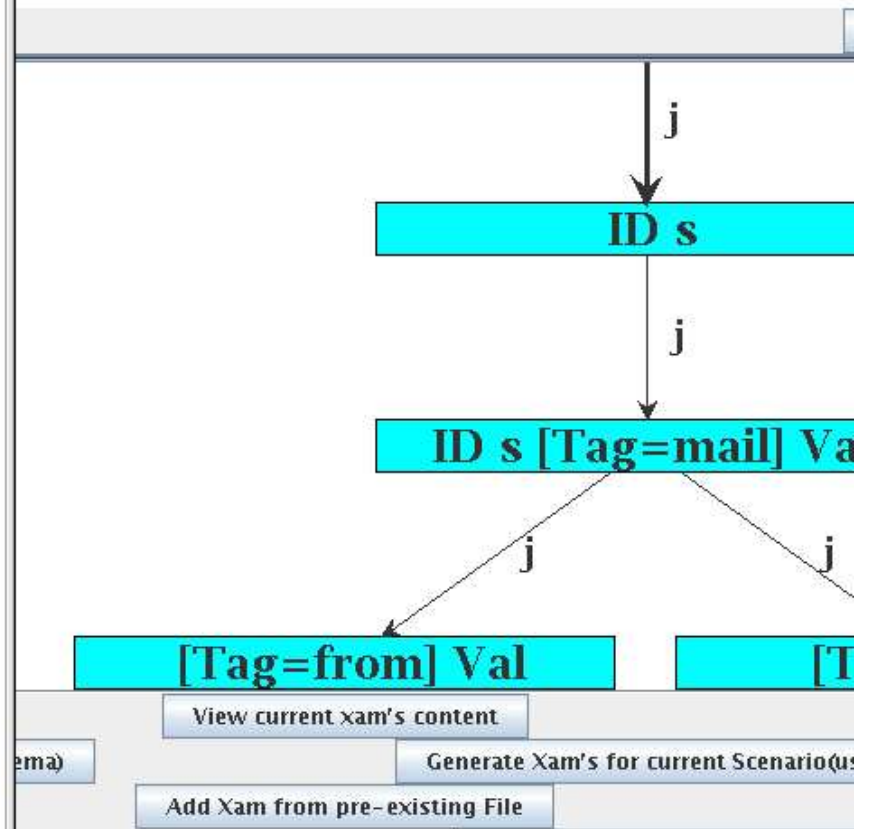
Table "xamtest.xam_auctionshare16"		
Column	Type	Modifiers
id1pre	integer	
id1post	integer	
id1depth	integer	
id2pre	integer	
id2post	integer	
id2depth	integer	
val2	character varying	
val3	character varying	
val4	character varying	
val5	character varying	
val6	character varying	
val7	character varying	
val8	character varying	
val9	character varying	
val10	character varying	
val11	character varying	

View Xam Content

FlatScan(xamtest.xam\_auctionshare1)

```

=====
(0:STRUCTURAL_ID_TYPE 1:STRUCTURAL_ID_TYPE 2:STRING_TYPE 3:STRING_TYPE 4:STRIN
(27 29 4, 28 28 5, "", "Dominic Takano mailto:Takano@yahoo.com", "Mechthild Renear m
(57 65 4, 58 58 5, "", "Fumitaka Cenzer mailto:Cenzer@savera.com", "Lanju Takano mail
(57 65 4, 64 64 5, "", "Papa Godskesen mailto:Godskesen@uwindsor.ca", "Ioana Blumber
(115 116 4, 116 115 5, "", "Aspi L'Ecuyer mailto:L'Ecuyer@intersys.com", "Lesley Jeris ma
(179 180 4, 180 179 5, "", "Honari Castan mailto:Castan@uni-muenchen.de", "Maz Lucky
dered by [0,1]{}
  
```



# Querying a database of XAMs

The screenshot shows a query tool interface with the following components:

- Queries in the workload:** A dropdown menu set to 'joinBibQuery2' containing the XQuery:

```
for $x in //book, $y in //book, $z in $x/title, $t in $y/tit
where $z=$t
return <sameTitle>{$x} <otherBook>{$y}</otherBook>
```
- Xam's in the current query:** A dropdown menu set to 'joinBibQuery21'.
- query-derived XAMs:** A blue text label with two arrows pointing to the 'Root' and '[Tag=book] Cont' nodes in the query plan.
- Query Plan Diagram:** A vertical tree structure with three cyan nodes: 'Root', '[Tag=book] Cont', and '[Tag=title]'. Arrows labeled 'j' connect 'Root' to '[Tag=book] Cont' and '[Tag=book] Cont' to '[Tag=title]'.
- XAM PREDICATES:** A text box containing: '1) Content of Node 2 in joinBibQuery21 is equal to Content of Node 2 in joinBibQuery22'.
- Navigation Buttons:** A vertical stack of five buttons: 'Check Summary based Satisfiability of current query', 'Check Storage XAM sufficiency of current Query', 'Generate Logical PLANS for Current Query', 'View Logical Plans Generated for Current Query', and 'Convert the Logical Plans to Physical Plans for Current Query'.

# Logical query plans over XAMs

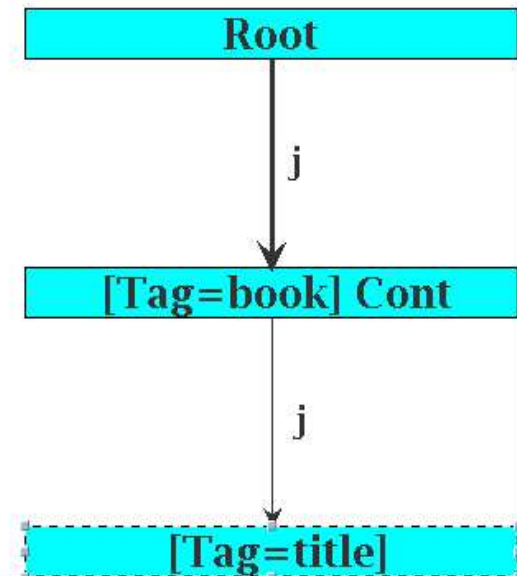
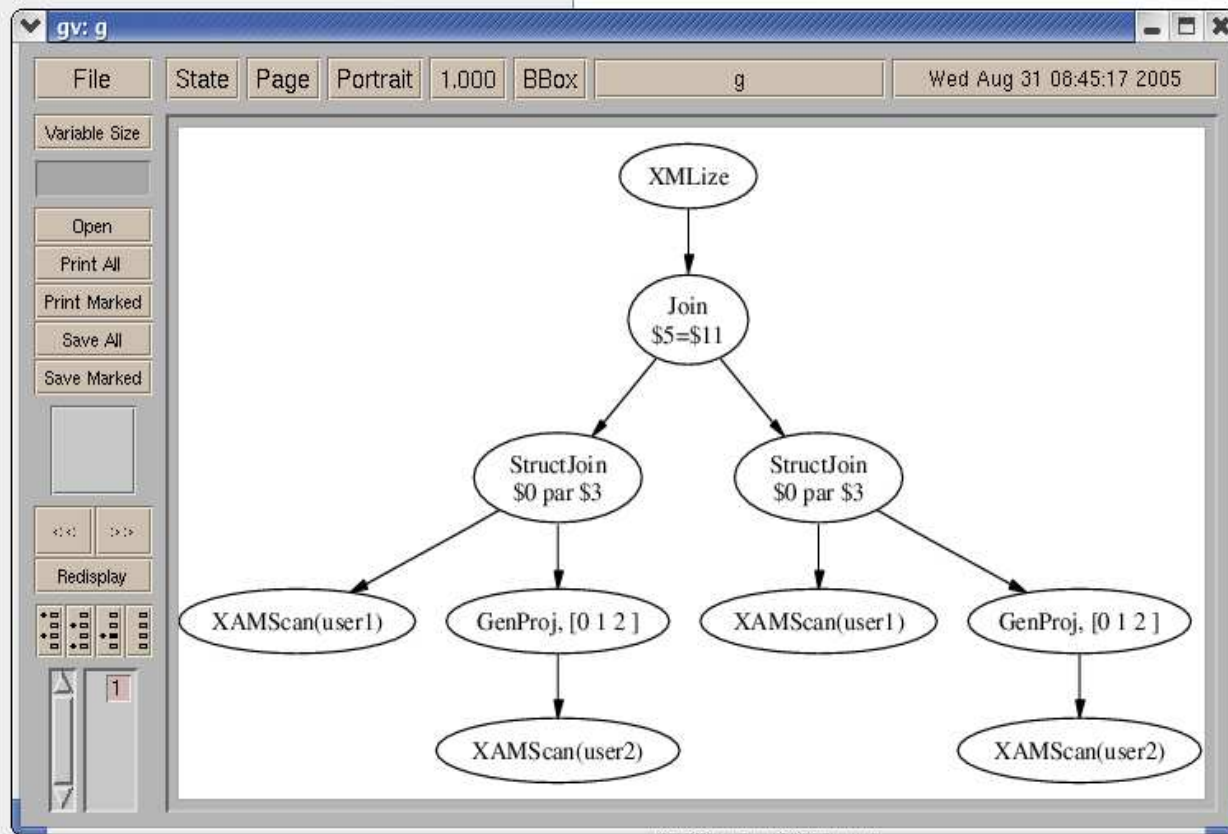
Queries in the workload

joinBibQuery2

```
for $x in //book, $y in //book, $z in $x/title, $t in $y/title
where $z=$t
return <sameTitle>{$x} <otherBook>{$y}</otherBook>
```

Xam's in the current query

joinBibQuery21



XAM PREDICATES:

# Testing query satisfiability

The screenshot shows a software interface for testing query satisfiability. On the left, a small window titled "Satisfiable" displays the message "Query is satisfiable" and an "ok" button. The main area is divided into two sections: "Queries in the workload" and "Xam's in the current query".

**Queries in the workload:** This section contains a dropdown menu labeled "joinBibQuery2" and a text area with the following query:

```
for $x in //book, $y in //book, $z in $x/title, $t in $y/tit
where $z=$t
return <sameTitle>{$z} <otherBook>{$y}</otherBook>
```

**Xam's in the current query:** This section contains a dropdown menu labeled "joinBibQuery21" and a large empty rectangular area. To the right of this area is a diagram showing a vertical sequence of three cyan rectangular nodes:

- Top node: **Root**
- Middle node: **[Tag=book] Cont**
- Bottom node: **[Tag=title]**

Arrows labeled "j" point from the "Root" node to the "[Tag=book] Cont" node, and from the "[Tag=book] Cont" node to the "[Tag=title]" node.

Below the diagram, a text box contains the following text:

XAM PREDICATES:  
1) Content of Node 2 in joinBibQuery21 is equal to Content of Node 2 in joinBibQuery22

At the bottom center, there is a button labeled "Check Summary-based Satisfiability of current query".

# Testing query coverage by the stored XAMs

UnSatisfiable

There are nodes of the query that are not covered by the storage. Please check the XAMs of the query!

ok

firstNameWrong

book[//title="Data on the Web"]//author/firstname

firstNameWrong1

Xam's in the current query

```
graph TD; Root[Root] -- j --> Book["[Tag=book]"]; Book -- s --> Title["[Tag=title] [Val=Data...]"]; Book -- j --> Author["[Tag=author]"]; Author -- j --> Firstname["[Tag=firstname] Cont"];
```

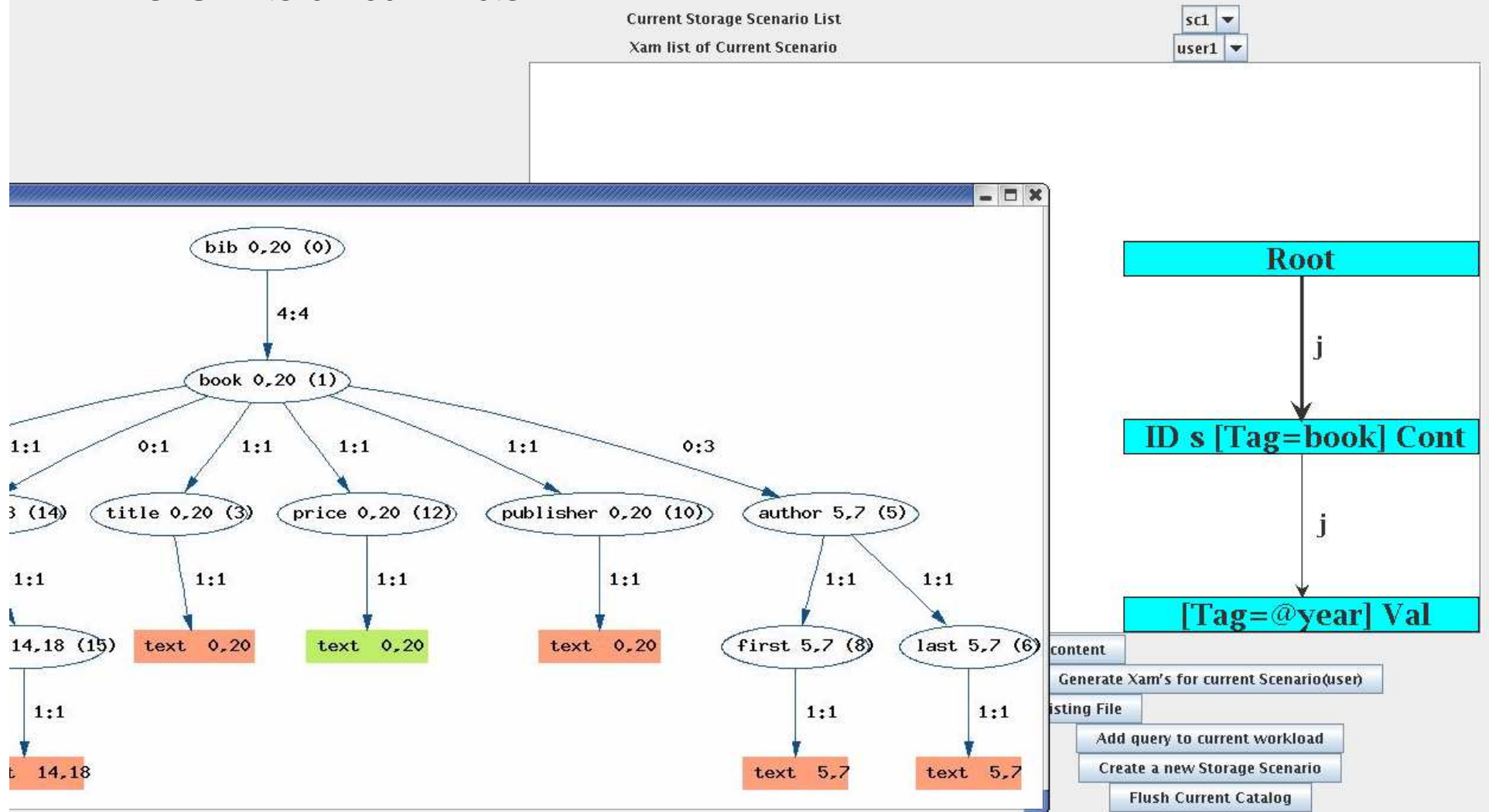
XAM PREDICATES:

Check Summary based Satisfiability of current query

Check Storage XAM sufficiency of current Query



# Behind the scene: structural constraints



# More information

<http://www-rocq.inria.fr/gemo/XAM>

A.Arion, V.Benzaken and I. Manolescu. "XML Access Modules: Towards Physical Data Independence in XML Databases", XIME-P 2005

A.Arion, V.Benzaken, I. Manolescu, R.Vijay. "ULoad: choosing the storage for your XML application", VLDB 2005 (demo)

Tech. report upcoming

# Perspectives

XAMS et ULoad:

formaliser l'inclusion de requetes et la re-écriture a base  
de XAMs (completude, complexite...)

connexion avec l'environnement de requetes CDuce



## Related works

# Related works (1)

Long history of storage and indexing schemes

Also known as "shredding" strategies, path indexes

Shanmugasundaram et al. 1999, Benedikt et al. 2001,  
Kaushik et al. 2002, ...

SQL/XML published in 2003

XPath containment and equivalence

Deutsch and Tannen, 2001 and later; based on chase

Suciu; Schwentick, Gottlob and Segoufin; Ozsoyoglu...

We provide a practical method for XPath rewriting under  
specific constraints (easier !)

# Related works (2)

## Tree pattern minimization

Amer-Yahia and Srivastava, 2001. Different containment,  
no rewriting

## XQuery containment

Halevy et al. 2004, different notion of containment, no  
constraints

## XPath rewriting with materialized views

Weak path usage. Balmin et al, VLDB 2004

## Algebraic minimization of XQuery

Deutsch and Papakonstantinou, VLDB 2004