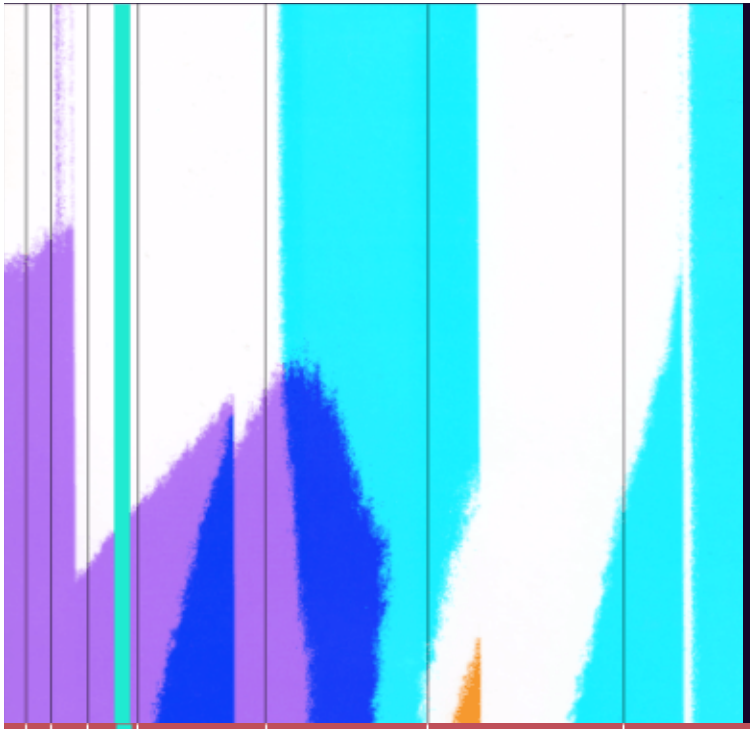


Path sequence-based XML query processing

BDA 2004, Montpellier

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

INRIA

Ioana Manolescu[1] Andrei Arion[1,2]

Angela Bonifati[3] Andrea Pugliese[4]

[1] INRIA Futurs – LRI, PCRI, France

[2] University of Paris XI

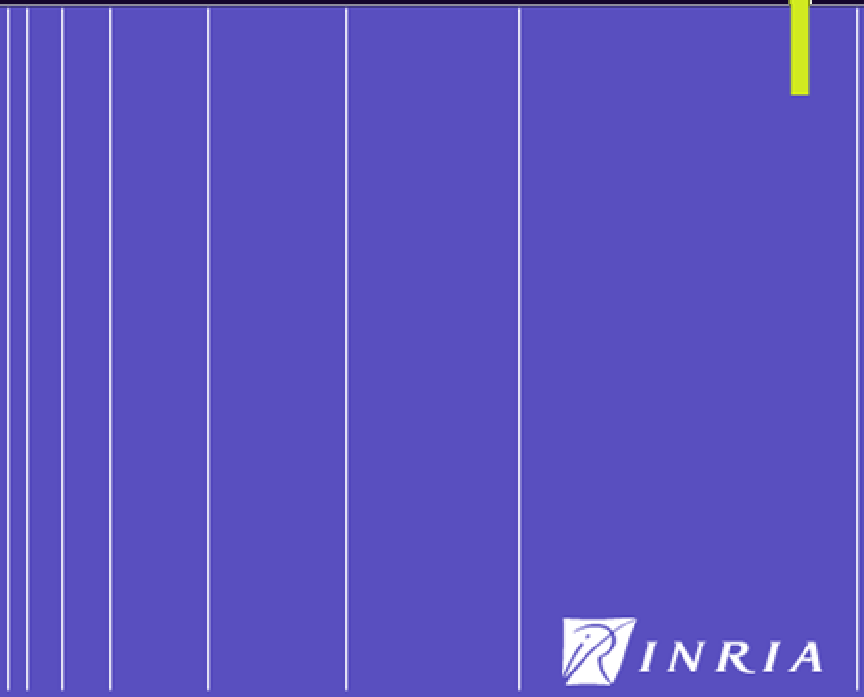
[3] ICAR CNR, Italy

[4] DEIS, University of Calabria, Italy

Plan

- Path sequence based XML storage
  - Logical path sequence model
  - Physical storage
- Path sequence based query processing
  - Path sequence-based XQuery optimization
    - FOR-WHERE
    - RETURN
  - Physical operators
  - Putting it all together
- Conclusion and future work

The path sequence model for storing XML documents

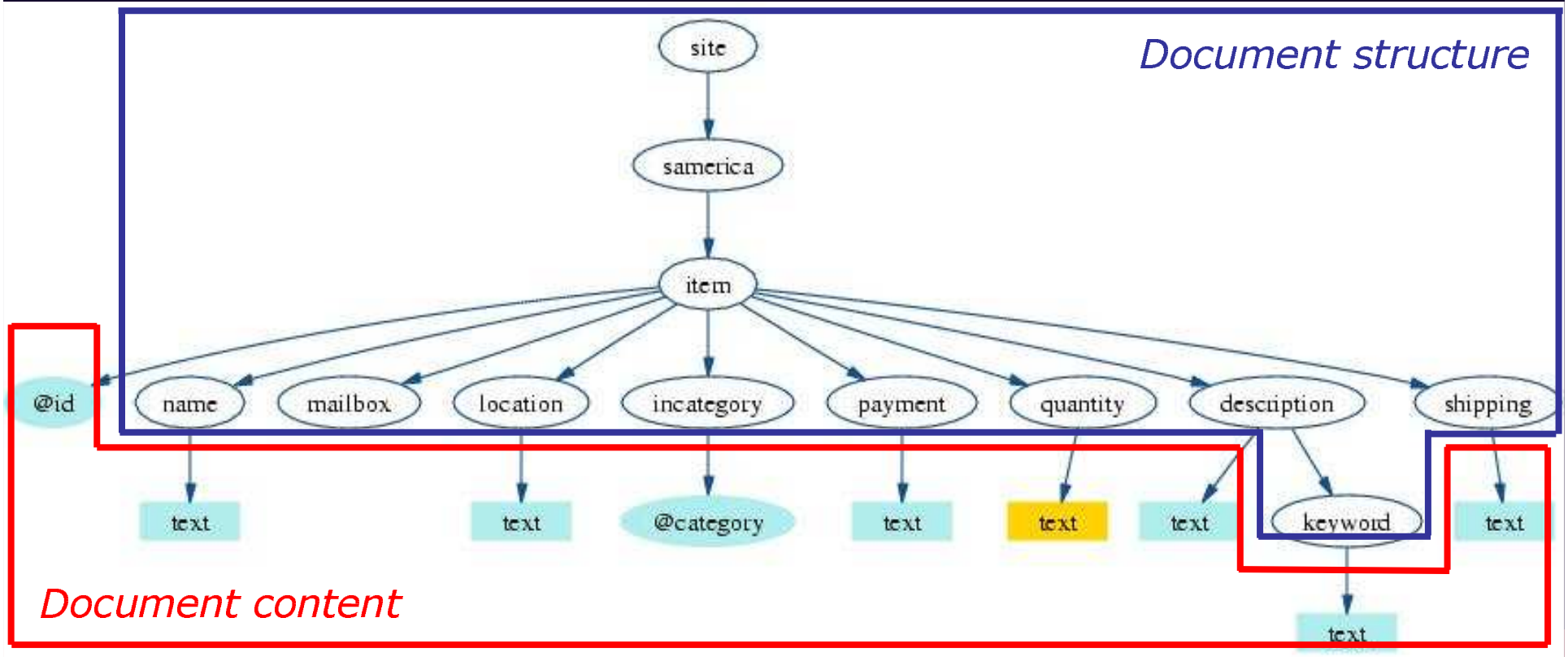


INRIA

The logical path sequences storage model

- Separate XML structure from content
  - Structure is useful
  - Content is large
- Split structure and content by the incoming path
  - Sequences of element IDs
  - Sequences of values
  - Selective data access, efficient twig pattern query evaluation
- Store structure and content in path sequences
  - Built-in document order

Separating structure from content



Storing document structure in path sequences

Assign a structural identifier to elements [pre, post, depth]

As in Timber, Natix ...

Direct establishment of structural relationships

ID1 ancestor of ID2 iff ID1.pre < ID2.pre and ID1.post > ID2.post

For parent, test also ID1.depth = ID2.depth - 1

Storing the document structure:

1 path in the doc = 1 sequence of structural element identifiers

Stored in pre (document) order

Structural identifiers

[1, 8, 1] <item @id=''item0''>
[2, 1, 2] <description> Beany Crocodile </description>
[3, 3, 2] <seller>
[4, 2, 3] <personRef>Joe</personRef>
</seller>
[5, 4, 2] <payment>Visa</payment>
[6, 5, 2] <payment> MasterCard </payment>
[7, 6, 2] <mailbox> </mailbox>
[8, 7, 2] <shipping> UPS in US only </shipping>
</item>

[1, 8, 1] ancestor of [5, 4, 2], [6, 5, 2]
[3, 3, 2] parent of [4, 2, 3]

Structural identifier sequences

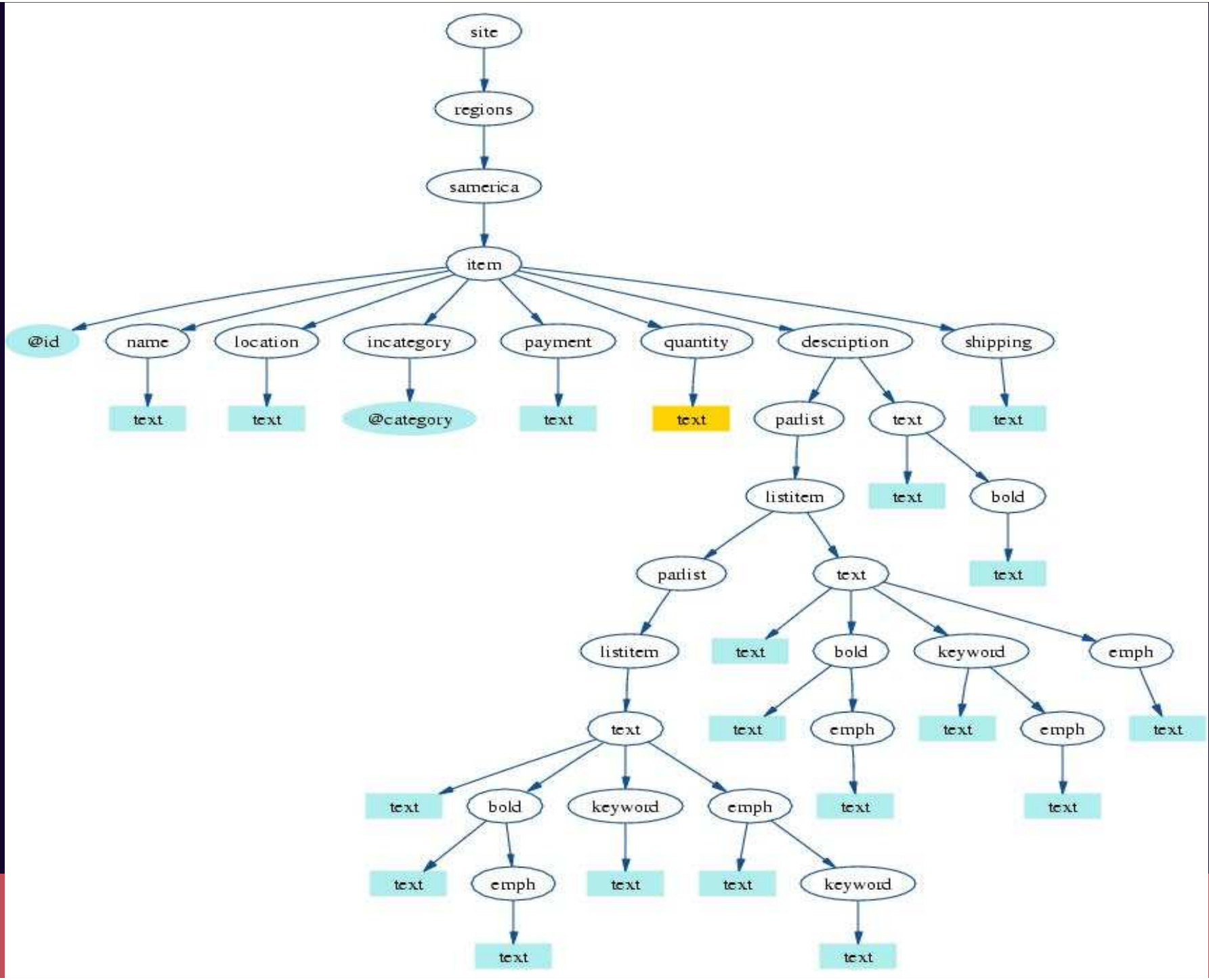
[1, 8, 1] <item @id=''item0''>
[2, 1, 2] <description> Beany Crocodile </description>
[3, 3, 2] <seller>
[4, 2, 3] <personRef>Joe</personRef>
</seller>
[5, 4, 2] <payment>Visa</payment>
[6, 5, 2] <payment> MasterCard </payment>
[7, 6, 2] <mailbox> </mailbox>
[8, 7, 2] <shipping> UPS in US only </shipping>
</item>

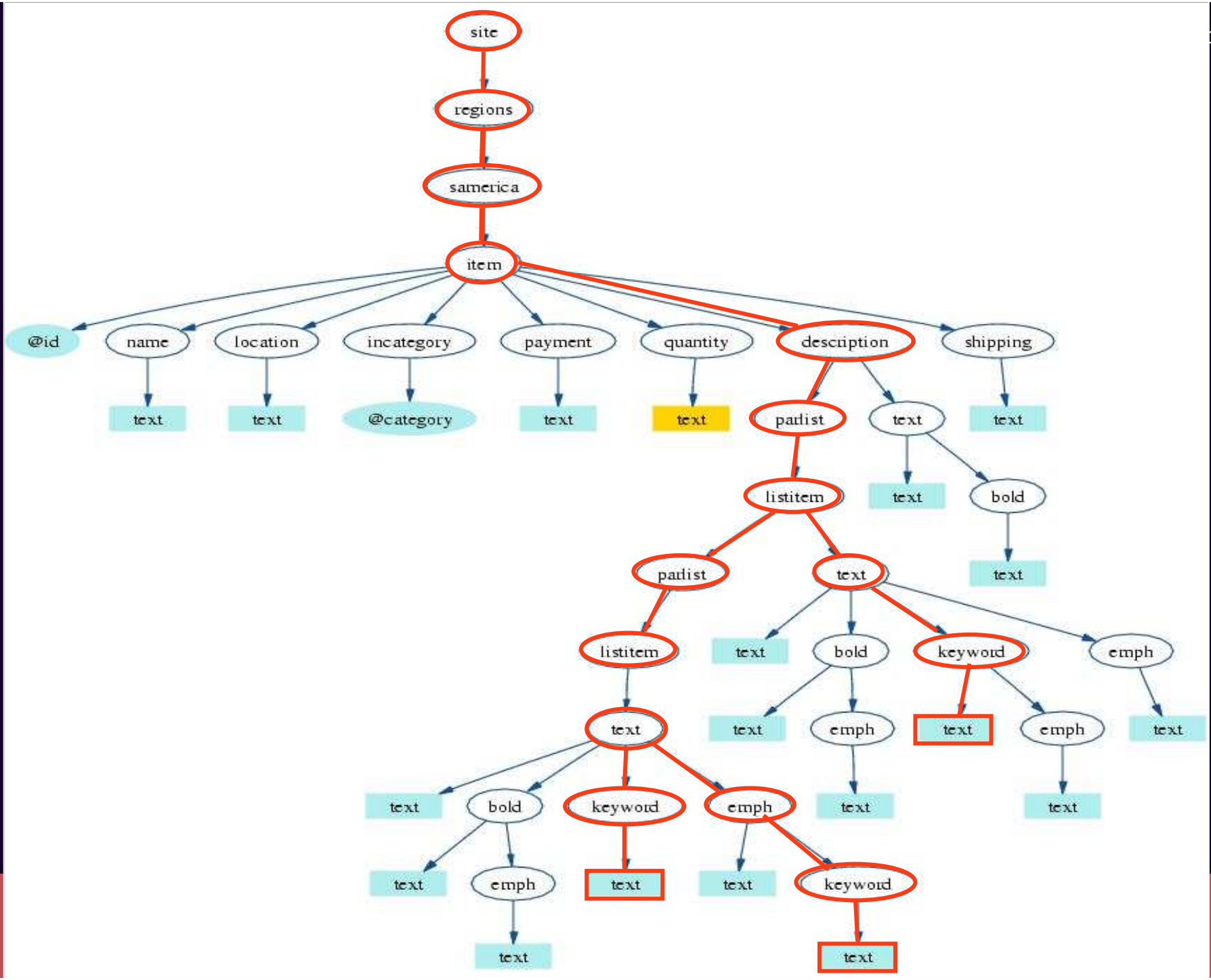
Path sequence for /site/auctions/samerica/item: { [1, 8, 1] }
Path sequence for ../item/payment: { [5, 4, 2], [6, 5, 2] }

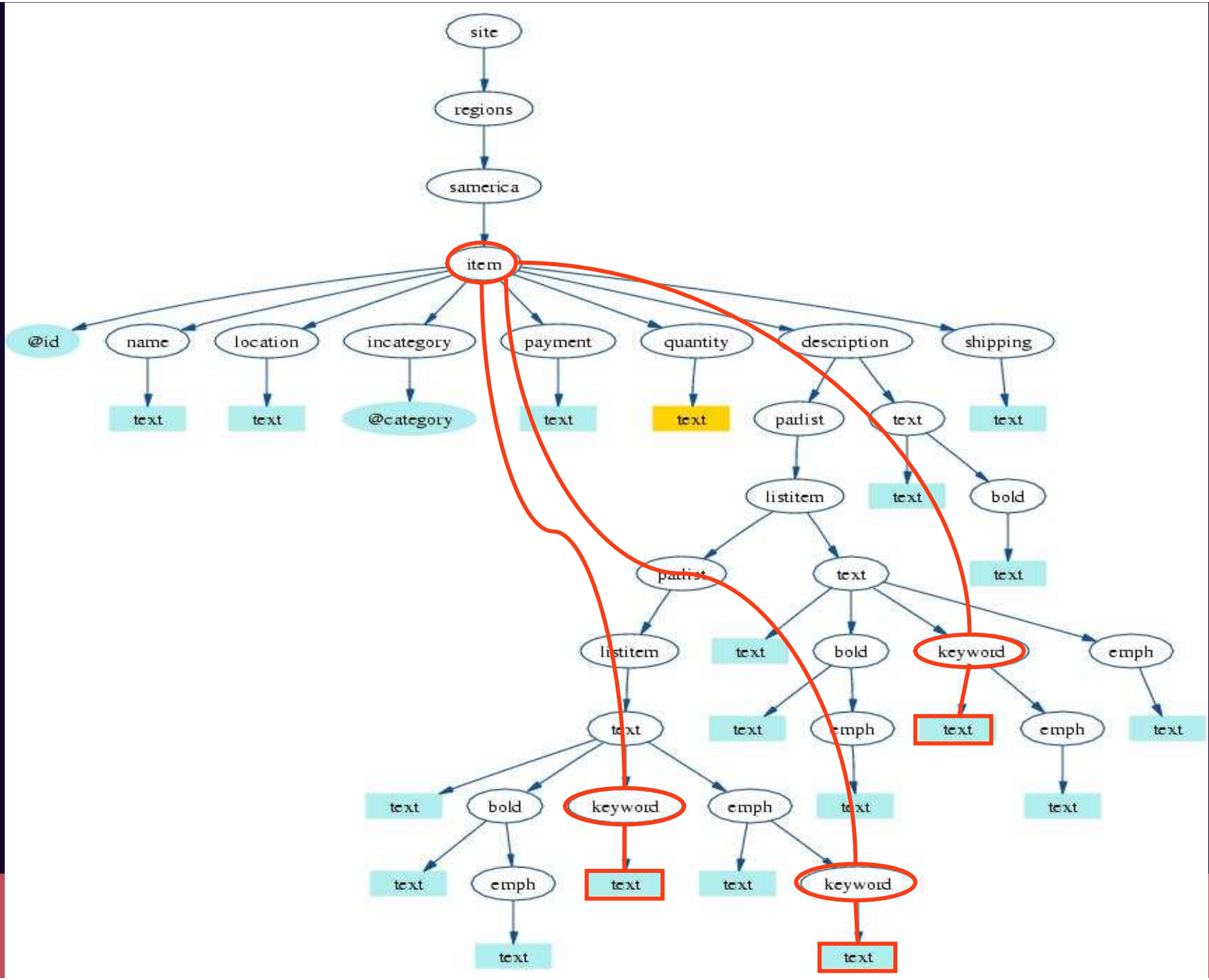
Structural identifier sequences

[1, 8, 1] <item @id=''item0''>
[2, 1, 2] <description> Beany Crocodile </description>
[3, 3, 2] <seller>
[4, 2, 3] <personRef>Joe</personRef>
</seller>
[5, 4, 2] <payment>Visa</payment>
[6, 5, 2] <payment> MasterCard </payment>
[7, 6, 2] <mailbox> </mailbox>
[8, 7, 2] <shipping> UPS in US only </shipping>
</item>

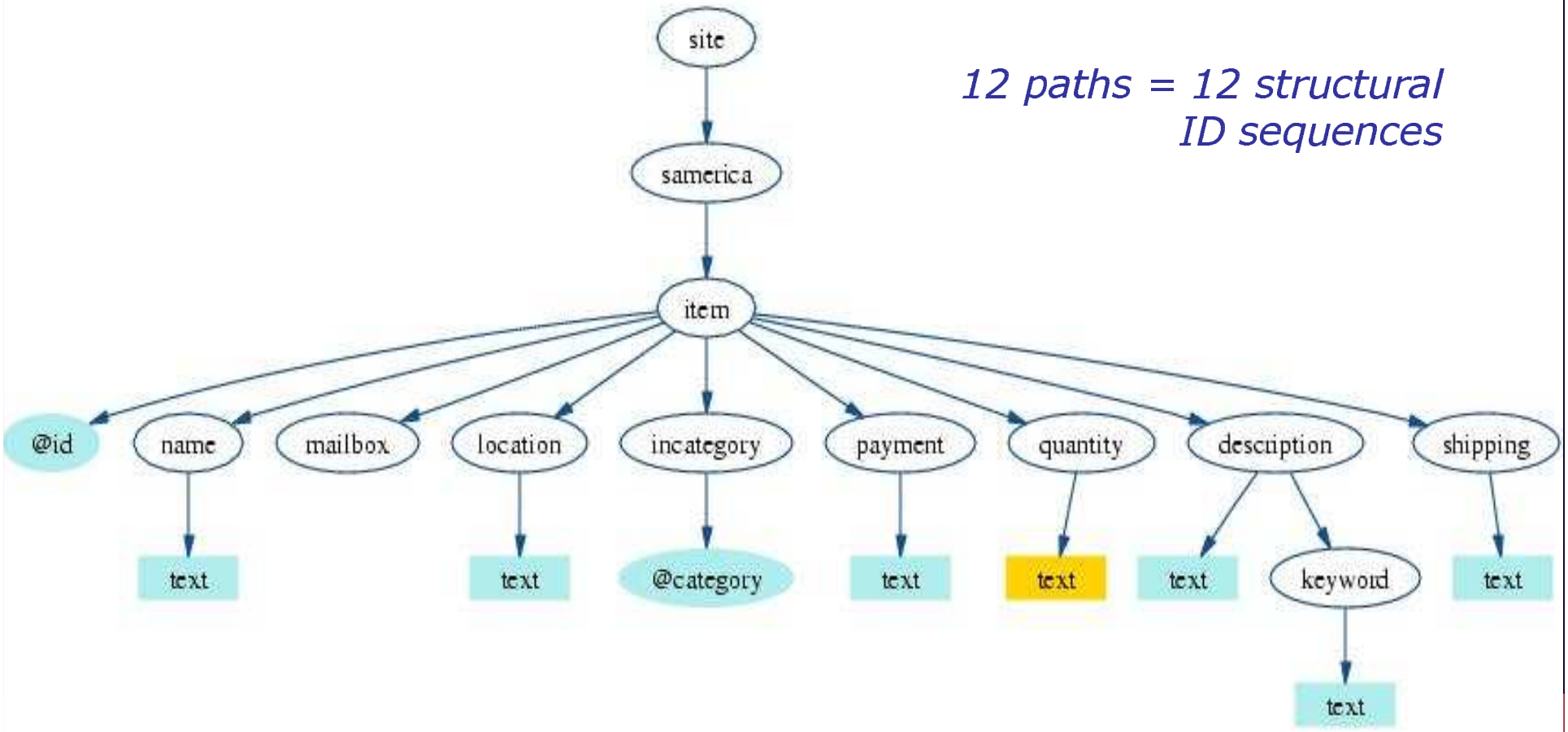
Path sequence for /site/auctions/samerica/item: { [1, 8] }
Path sequence for ../item/payment: { [5, 4], [6, 5] }


site
regions
samerica
item
@id
name
location
incategory
payment
quantity
description
shipping
text
text
@category
text
text
parlist
text
text
listitem
text
bold
text
parlist
text
text
listitem
text
bold
keyword
emph
text
text
emph
text
emph
text
text
bold
keyword
emph
text
text
text
emph
text
text
keyword
text
text


site
regions
samerica
item
@id
name
location
incategory
payment
quantity
description
shipping
text
text
@category
text
text
parlist
text
text
listitem
text
bold
text
parlist
text
text
listitem
text
bold
keyword
emph
text
text
emph
text
emph
text
text
bold
keyword
emph
text
emph
text
text
text
keyword
text
text
text


site
regions
samerica
item
@id
name
location
incategory
payment
quantity
description
shipping
text
text
@category
text
text
parlist
text
text
listitem
text
bold
text
parlist
text
listitem
text
bold
keyword
emph
text
text
emph
text
emph
text
text
bold
keyword
emph
text
text
text
text
emph
text
keyword
text
text

Structural identifier sequences



12 paths = 12 structural ID sequences

Storing document content in path sequences

For each root-to-leaf path in the XML document, store a *container: a sequence of (ID, value) pairs*

Text nodes: IDs of the parent elements

Attributes: IDs of the enclosing elements

Container for /site/samerica/item/payment:

{ ([5, 4, 2], "Visa")
  [6, 5, 2], "MasterCard") }

Other persistent storage structures

Required:

- Path sumary
  - Dataguide on tree (XML) structure
  - Easy to build, compact

Optional:

- Statistics associated to the path summary
- Value / full text indexes on containers...
- Structural indexes...

Physical storage for path sequences

ID sequences: fixed-length items

Containers: varying-length items

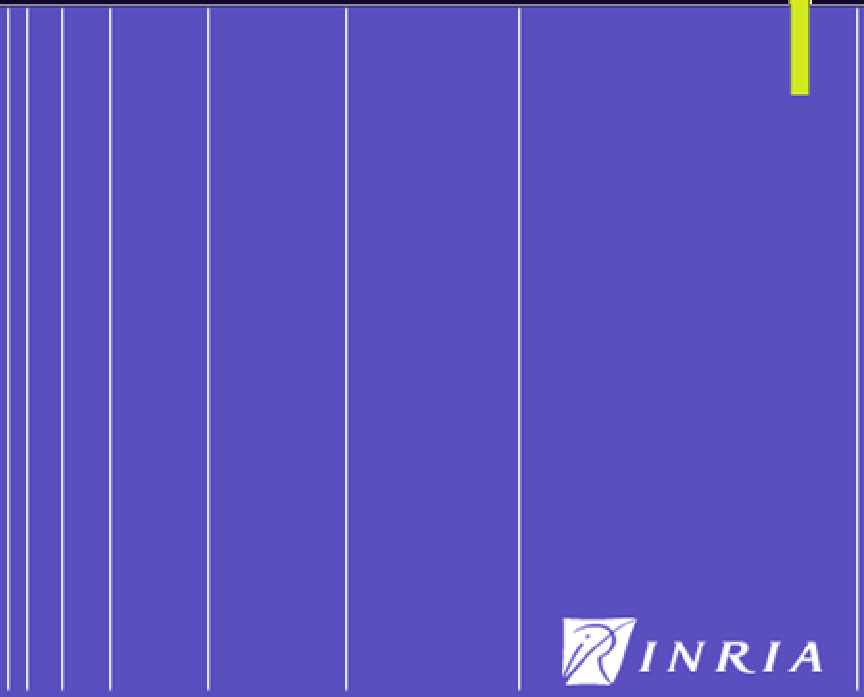
Ordered sequences (document order):

- B+ - trees
- Compact (continuous) sequences

Trade-off: compactness vs. suitability to updates

Implementation supports both, we mostly used sequences

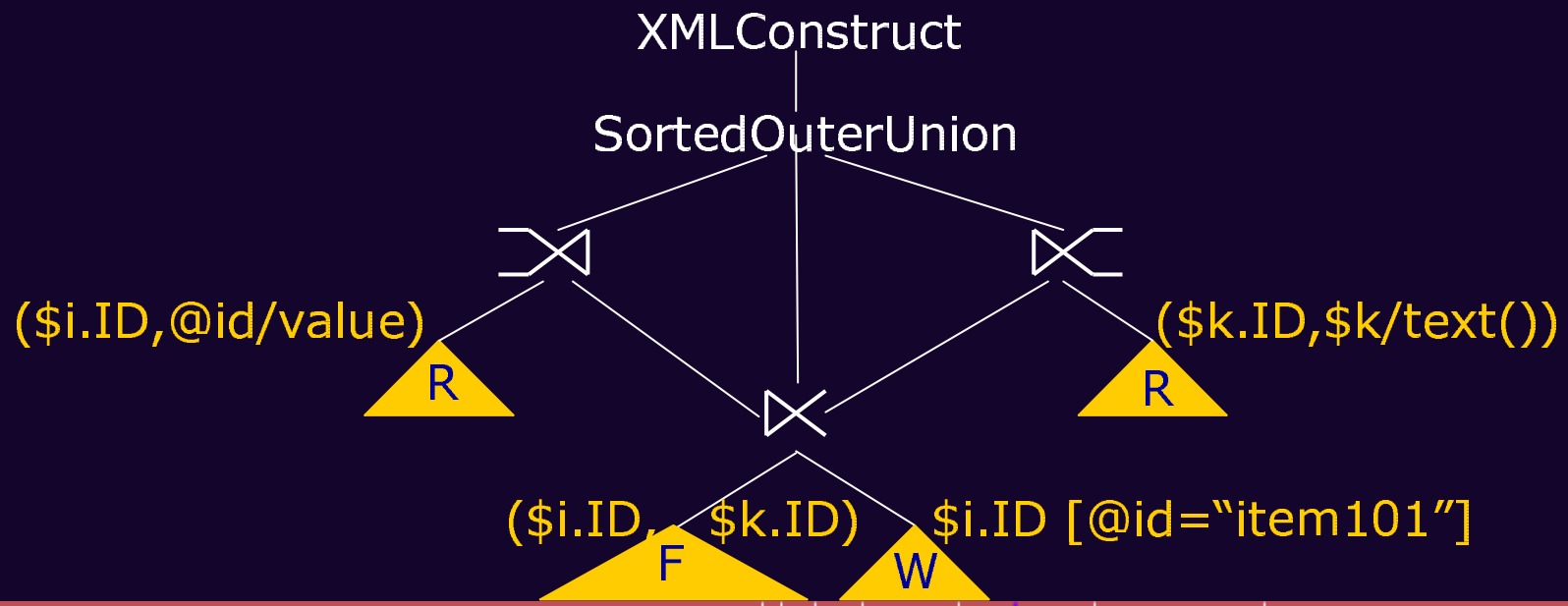
XML query processing based on path sequences



INRIA

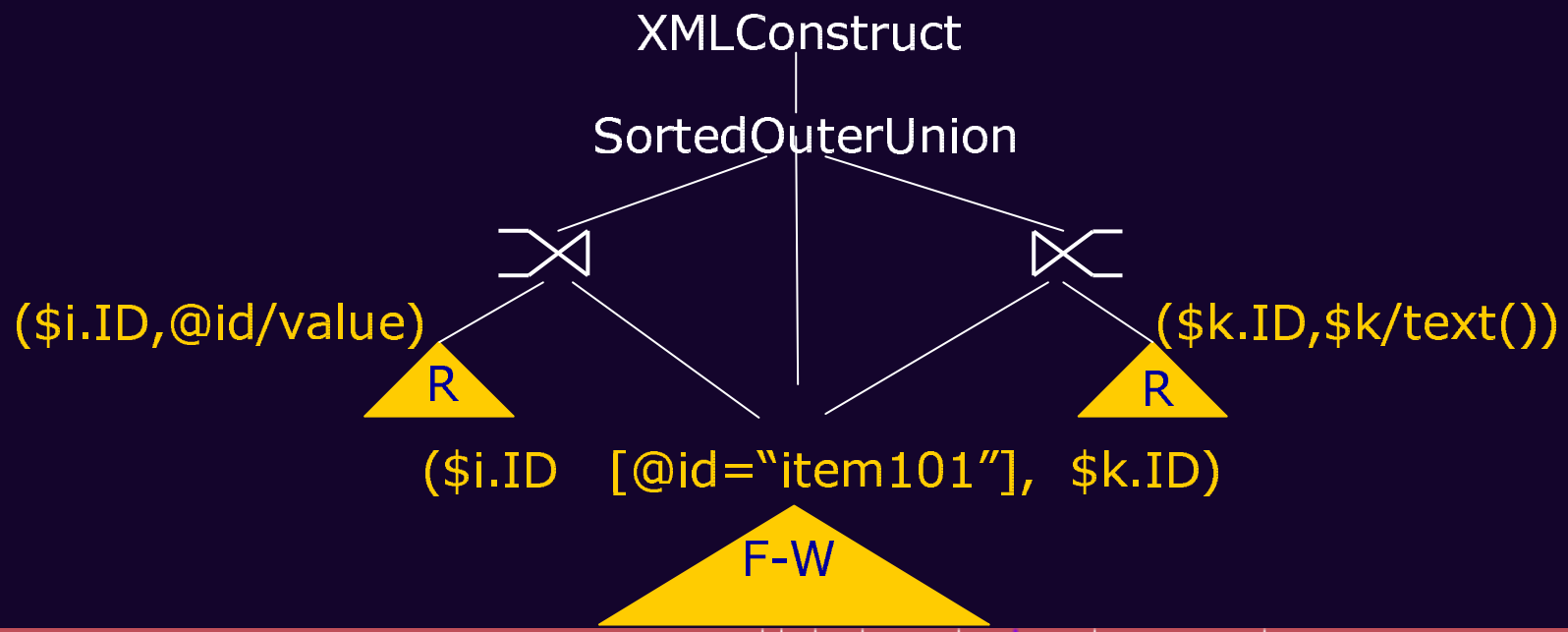
XQuery optimization approach (1)

```
for $i in //samerica/item
    $k in $i//keyword
where $i/@id="item101"
return <article>
            <numero> $i/@id/value() </numero>
            <mot-cle> $k/text() </mot-cle>
       </article>
```



XQuery optimization approach (2)

```
for $i in //samerica/item
    $k in $i//keyword where $i/@id="item101"
return <article>
            <numero> $i/@id/value() </numero>
            <mot-cle> $k/text() </mot-cle>
       </article>
```



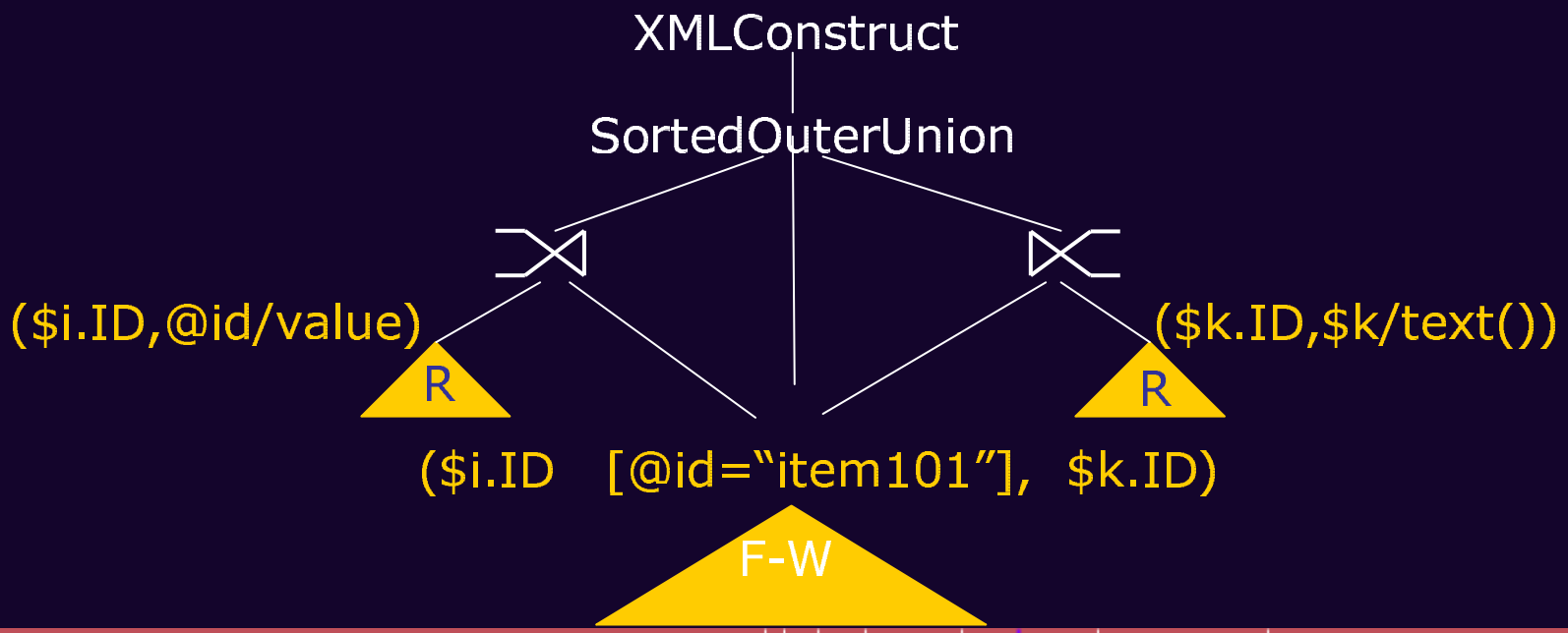
Steps in XQuery optimization

Optimize the FW plan (aka “twig pattern”):

- Variables on one or many paths; value predicates
- Output: tuples of IDs, sorted in doc order

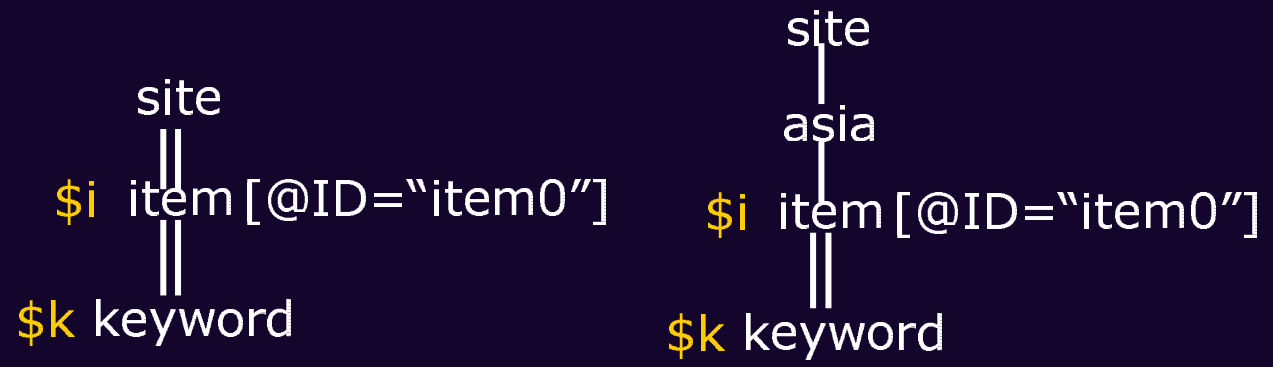
Optimize the R plan(s)

- Similar
- Based on the FW plan result



Optimizing the for-where clauses

Input: twig pattern; path summary



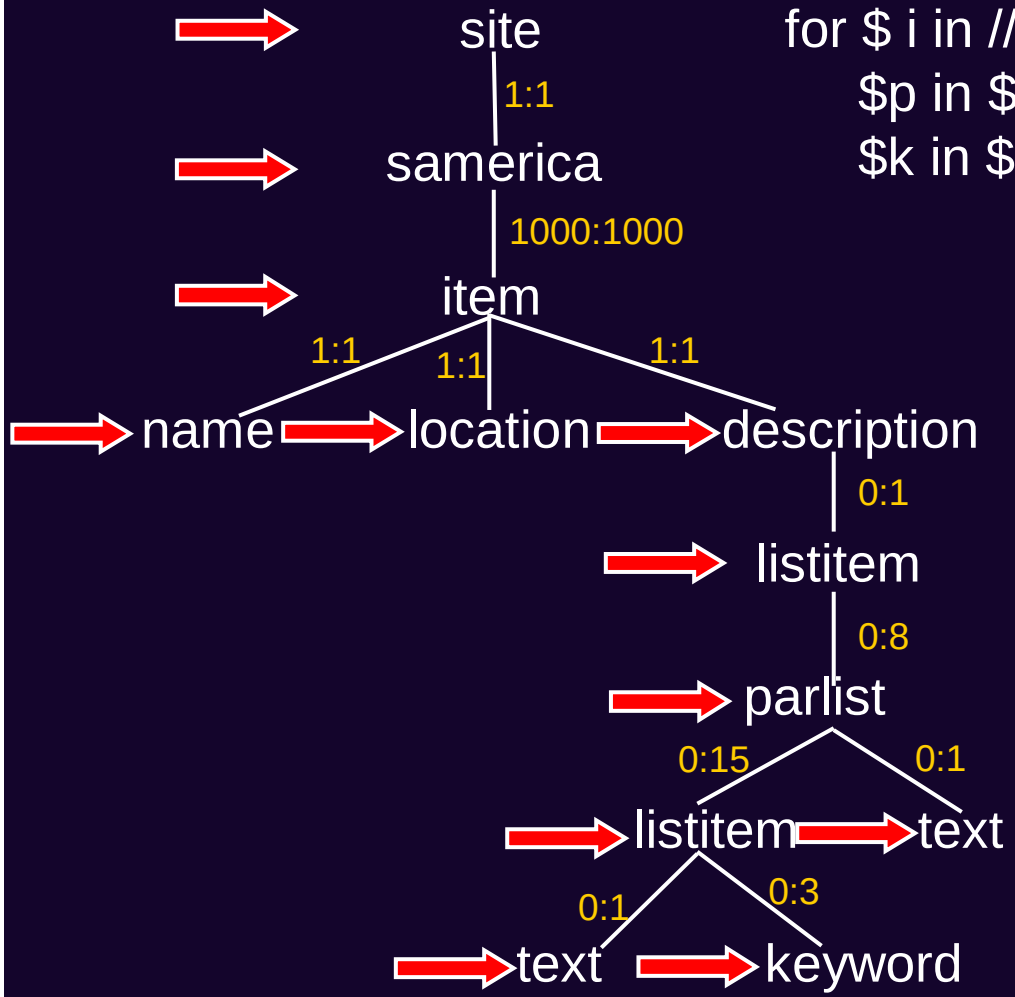
Output: *path list pList of each variable*

$i.pList = { /site/regions/africa/item, /site/regions/asia/item,
/site/regions/europe/item, /site/regions/namerica/item,
/site/regions/samerica/item }

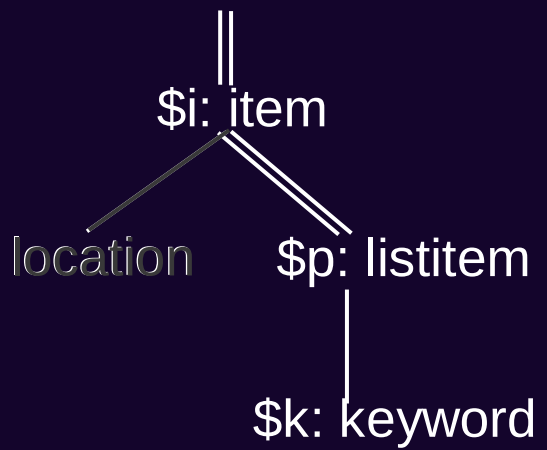
$k.pList is too large ☺

Variable path inference

Variable path inference



for $ i in //item[location],
$p in $i//listitem,
$k in $p/keyword

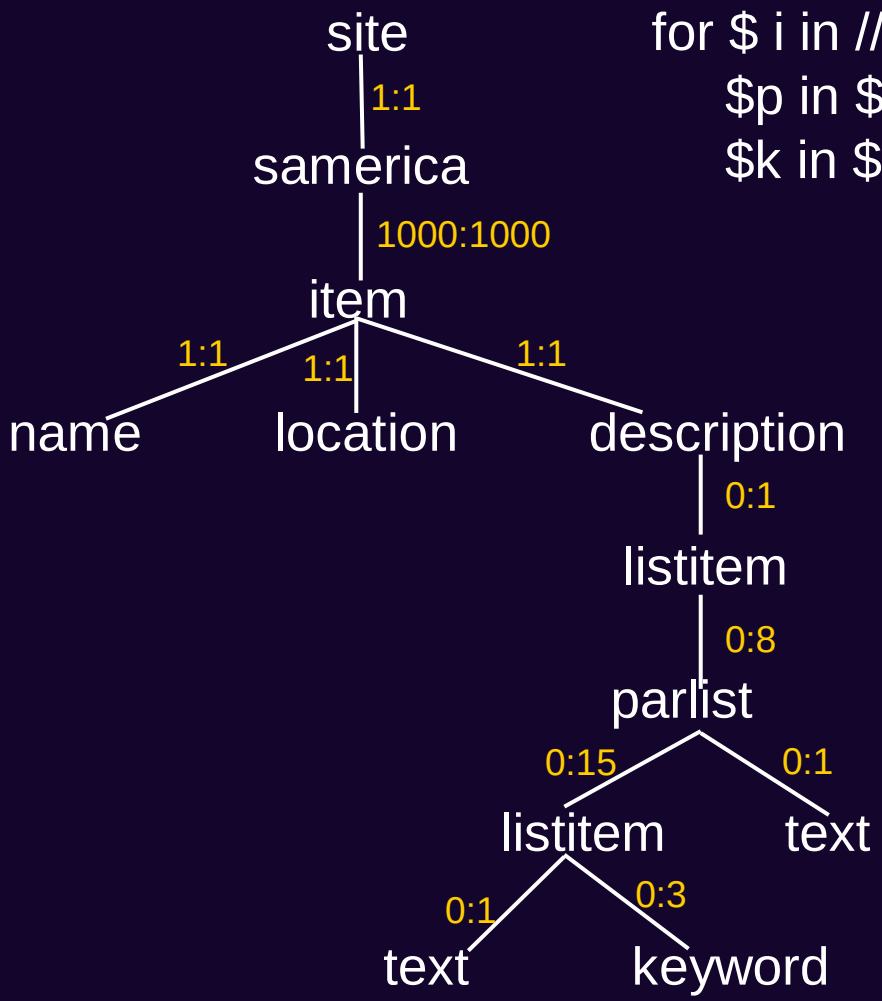


$i.pList /site/samerica/item

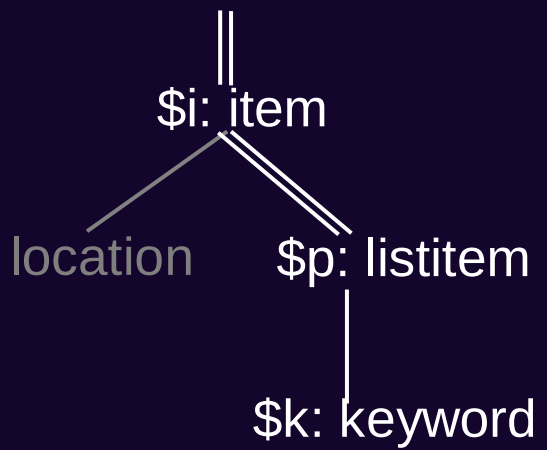
$p.pList /site/sam/item/descr/listitem
/site/sam/item/descr/litem/parlist/listitem

$k.pList
/site/sam/item/descr/litem/plist/litem/kwd

Variable path inference



for $ i in //item[location],
$p in $i//listitem,
$k in $p/keyword



$i.pList /site/samerica/item

$p.pList
/site/sam/item/descr/litem/parlist/listitem
$k.pList
/site/sam/item/descr/litem/plist/litem/kwd

Variable path inference

Traverse in parallel the path summary and the query

- Find the minimal set of paths for each variable
- Eliminate redundant branches
- Infer new structural relationships between variables

Implementation

- In memory: recursive traversals
- On disk-resident path summary: streaming query evaluation on the path summary

Physical operators

Iterator interface

Flat tuple algebra

PathScan: returns all IDs on a given path

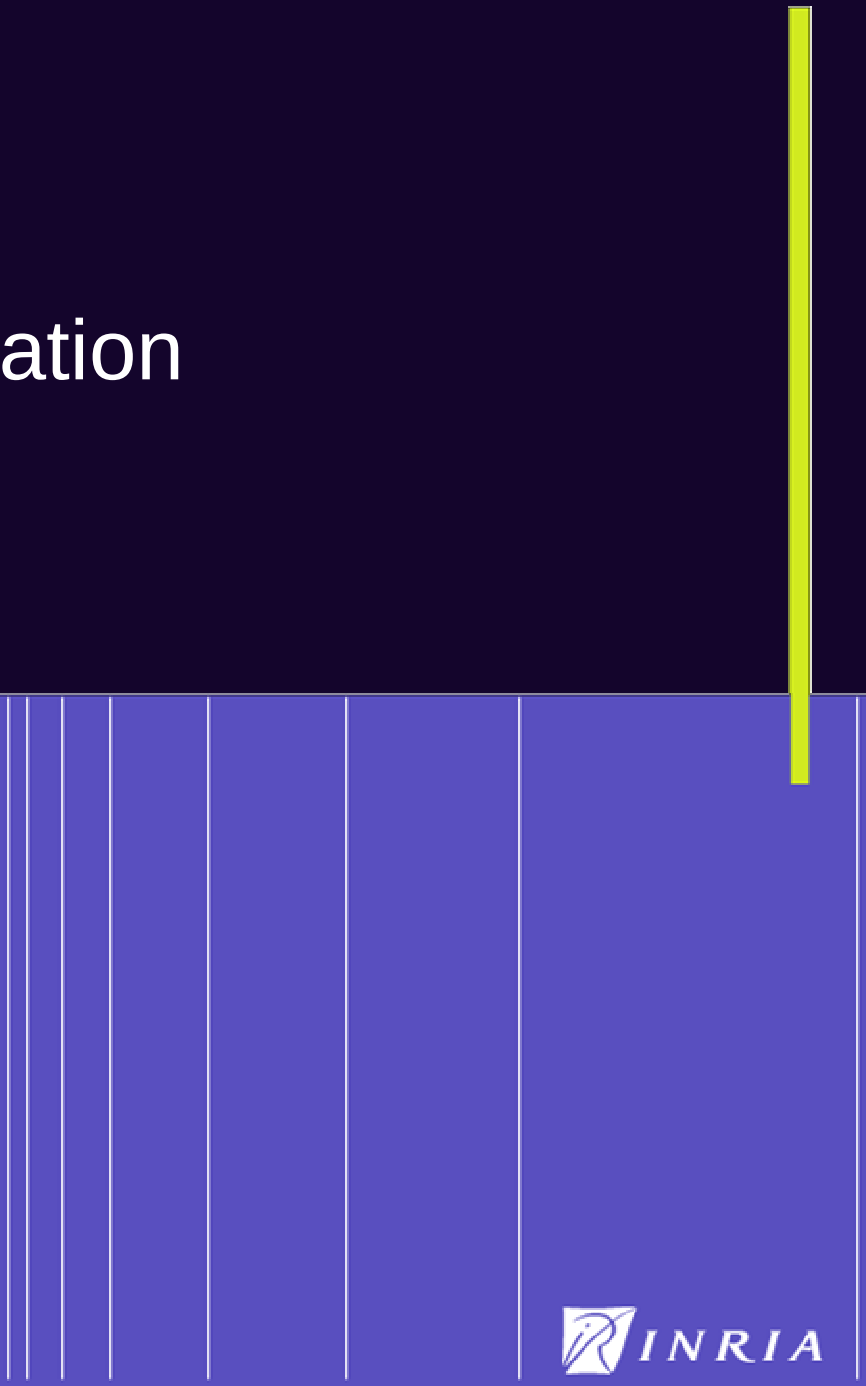
ContainerScan: returns all (ID, value) pairs on a given path

Merge: sequence fusion in document order

StackTreeDesc (structural join), outer- and semi-join variants

SortedOuterUnion for reconstruction

Performance evaluation



Physical storage

Implemented based on BerkeleyDB: free library of persistent storage structures

B+-trees, hash tables, fixed- and varying-length sequences

	B+-trees	Sequences
Path partitioning	√	√
Tag partitioning	√	√

Sequence-based storage 4 times more compact than B+-tree based

No overhead pages

Path partitioning more compact than tag-partitioning

Tags are stored only once per path

B+-trees better support updates

Query processing performance based on path sequences

Query performance depends on:

Selective data access

Path sequence access much more selective than tag-based access (e.g. Natix, Timber)

Efficient intermediary (e.g. join) algorithms

StackTreeDesc efficient (linear time); skipping joins
Path sequences enable more skipping

Query plan quality

Precise plan construction + minimization based on path summary

Path partitioning vs. tag partitioning: does it make a difference ?

Many paths lead to common tags (different meanings)

Name, title, address, comment, keyword, paragraph...

A given tag may appear in many contexts with the same meaning

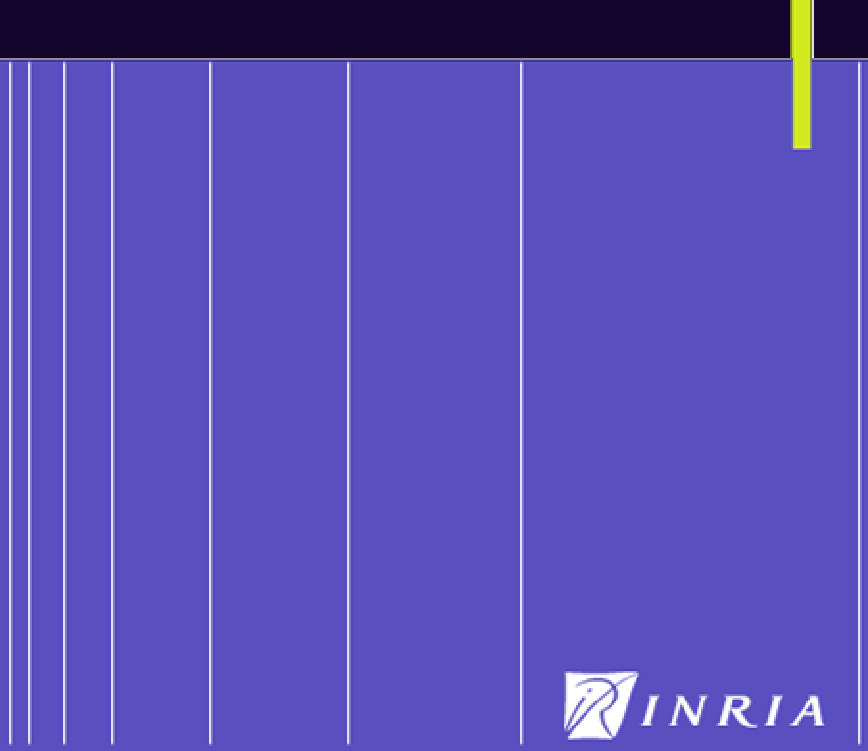
"Horizontal partitioning" e.g. auction items

Measured on a set of frequent XML documents *median fan-in*:

weighted average of #paths leading to a given tag
Values between 2.75 and 15.5

Only counter-example (median fan-in = 1): toy dataset designed by a UW CS grad student

Summary



Summary and perspectives

Path-sequence based storage model

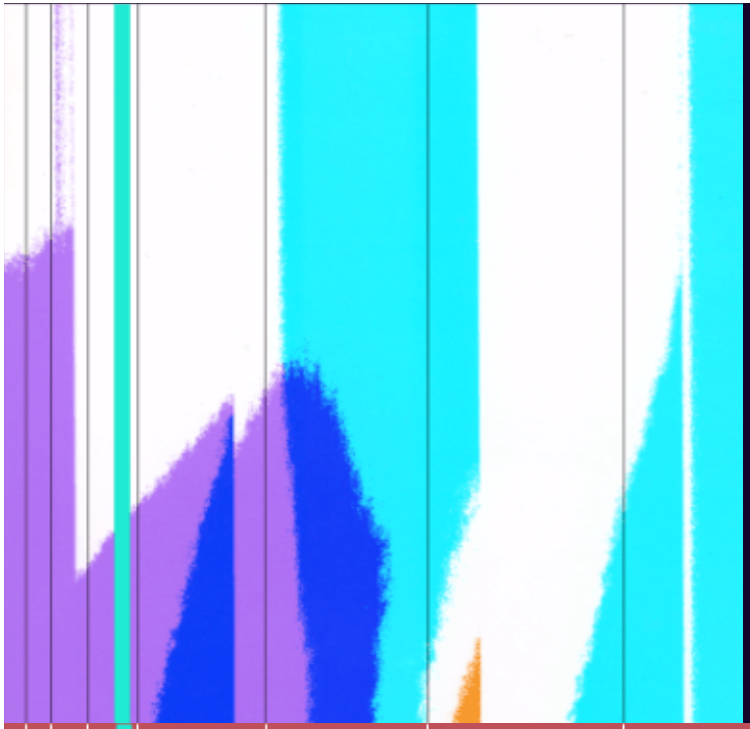
- Logical storage: Extreme "vertical" fragmentation
- Physical access: trade-offs B+-trees / sequences
- Data access: selective

Query processing on a path sequence storage

- Query optimization: minimization and pruning based on path summary
- Data reconstruction: very complex plans, good scaleup

Perspectives

- Finish up the code...
- Intermediary *physical* fragmentation degrees (path-tag)



Merci

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA