Efficient XML Storage based on DTM for Read-oriented Workloads

Graduate School of Information Science,
Nara Institute of Science and Technology

Makoto Yui
Jun Miyazaki, Shunsuke Uemura, Hirokazu Kato

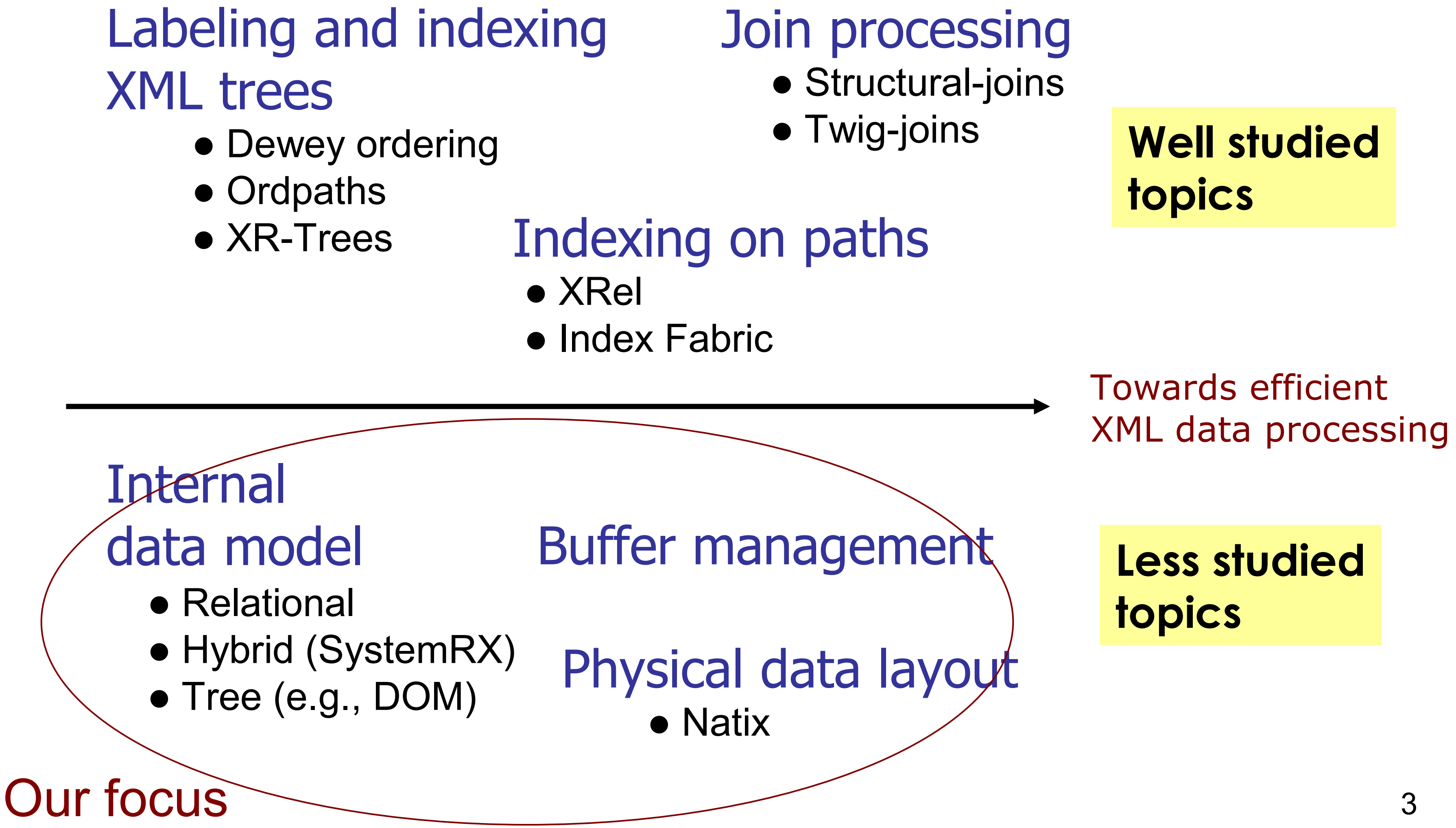
International Workshop on Advanced Storage Systems 2007 (ADSS 2007)

Outline

- Motivation
- Related work
- Document Table Model (DTM)
- XML storage based on DTM
  - System Overview
  - Physical Layout
  - Buffer Management
- Experimental Evaluation
- Conclusions

Motivation - Backgrounds

- Past research topics in XML data management



Labeling and indexing XML trees
- Dewey ordering
- Ordpaths
- XR-Trees

Join processing
- Structural-joins
- Twig-joins

Indexing on paths
- XRel
- Index Fabric

Well studied topics

Towards efficient XML data processing

Internal data model
- Relational
- Hybrid (SystemRX)
- Tree (e.g., DOM)

Buffer management

Physical data layout
- Natix

Less studied topics

Our focus

Motivation – Design goals

- Design an **XML storage scheme** optimized for read-oriented workload
  - Node-level update is not always required
  - Updating capabilit

The design of read-o ested for relational databases es yet

The reason we selected this model is set-at-a-time processing of XQuery involves lots of joins, and it causes performance deteriation.

- Focus on **iterative XQuery processing** in which an operator tree consists of iterators
  - Ideal XML storage scheme depends on the processing model (e.g., tuple-at-a-time or set-at-a-time)

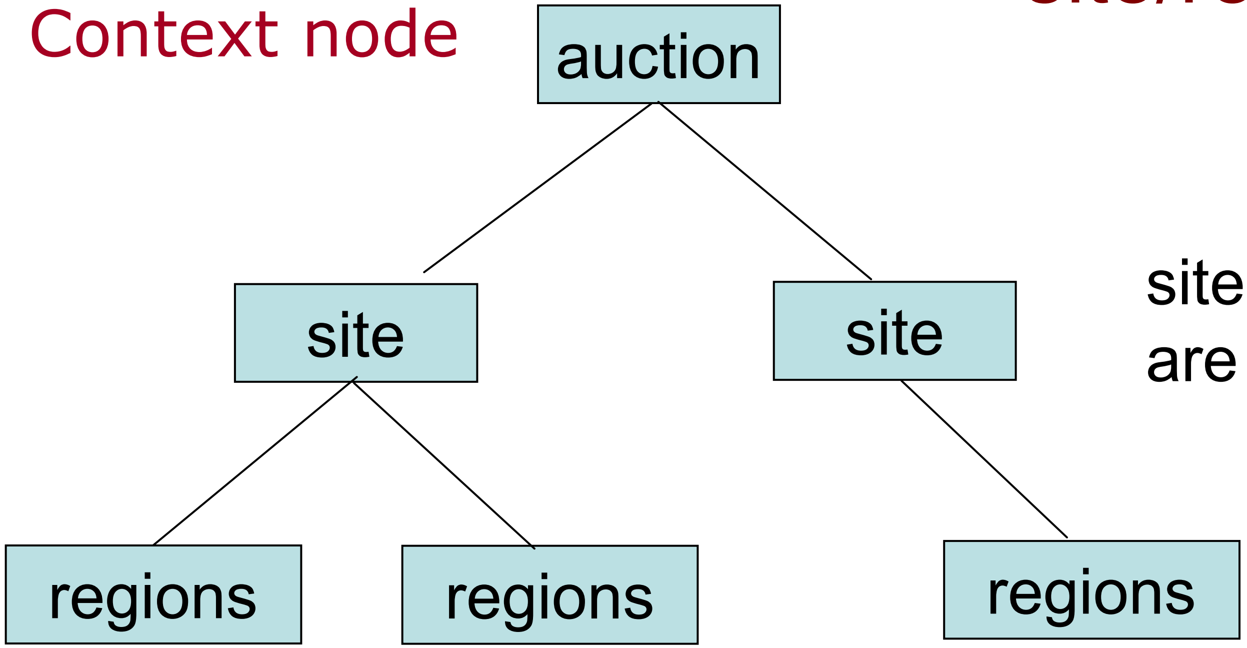
How the data access is achieved for each of the two processing model ?

Norman May, et al., Index vs. Navigation in XPath Evaluation. XSym 2006

Tree traversal of set-at-a-time processing

site/regions

Context node



sites under actions are selected

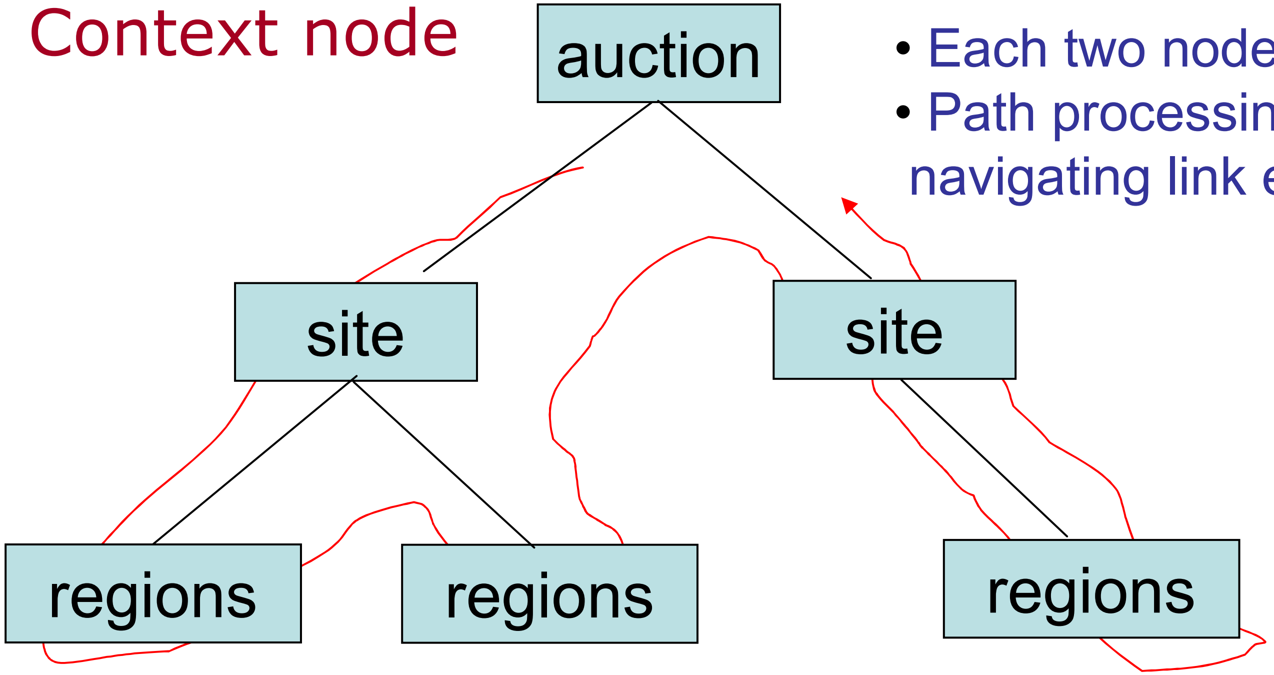
regions under sites are selected

Each edges are connected by join on nodes' identifiers

Traversed in a breadth-first search manner

Tree traversal of tuple-at-a-time processing

site/regions



- Each two nodes are connected by links
- Path processing is achieved by navigating link edges

Traversed in a depth-first search manner

It is as same manner as a document ordering

Motivation

- Design an **XML storage scheme** optimized for read-oriented workload
- Focus on **iterative XQuery processing** in which an operator tree consists of iterators

↓

We examined **actual data access patterns** when evaluating XQuery queries **in order to design the suitable data layout**

Outline

- Motivation
- Related Work
- Document Table Model (DTM)
- XML Storage based on DTM (pDTM)
  - System Overview
  - Pyshical Layout
  - Buffer Management
- Experimental Evaluation
- Conclusions

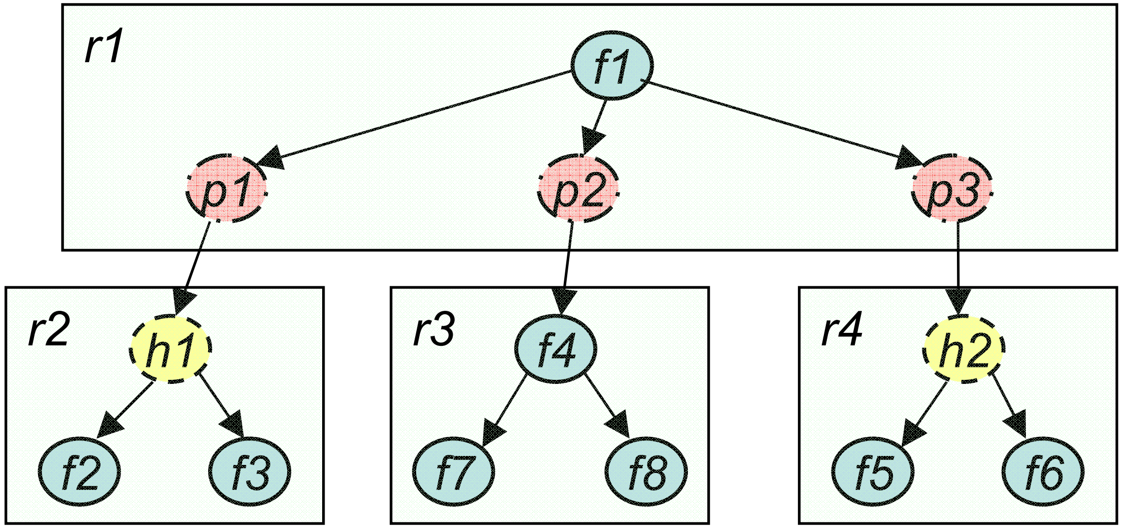
Related work (1) storing scheme based on subtrees

Natix (University of Mannheim, Germany)

T. Fiebig, et.al. Anatomy of a Native XML Base Management System.
VLDB Journal, 2002.

Allocates a page based on subtrees

- Pros — Effective for breadth-first traversals
- Cons — Not effective for depth-first traversals



P **Proxy Node**   h **Helper aggregate nodes**:

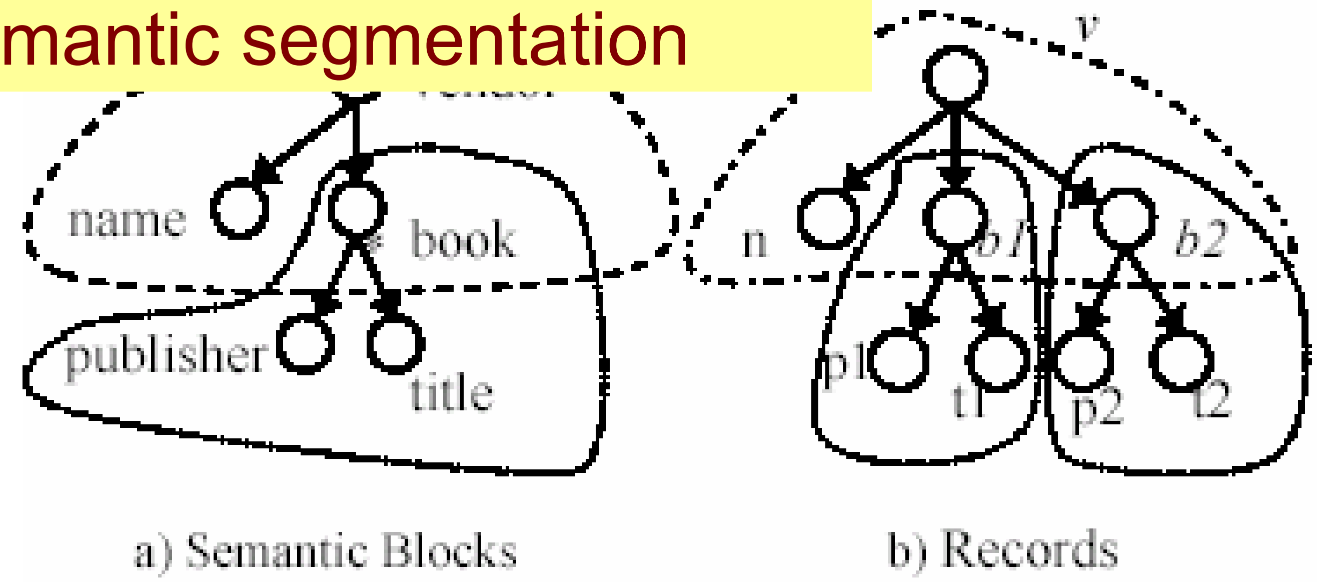
Related work (2) storing scheme based on a schema

OrientStore (Renmin University, China)

X. Meng et.al. OrientStore: A Schema Based Native XML Storage System. VLDB 2003.

- Pros
  - Effective for path processing
- Cons
  - Schema information is required
  - Not effective for serialization and
ion

Clusters records according to a semantic segmentation



a) Semantic Blocks b) Records

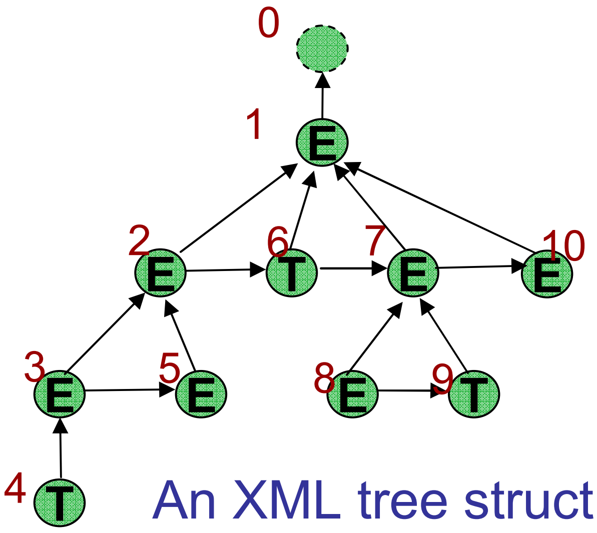
Outline

- Motivation
- Related work
- Document Table Model (DTM)
- XML Storage based on DTM (pDTM)
  - System Overview
  - Pyshical Layout
  - Buffer Management
- Experimental Evaluation
- Conclusions

Document Table Model (DTM)

- Originally used by Apache Xalan XSLT processor
- Expresses an XML document as a table form
  - DOM has object footprints
    (e.g., object instantiation and memory consumptions)
  - DTM can avoid such object footprints

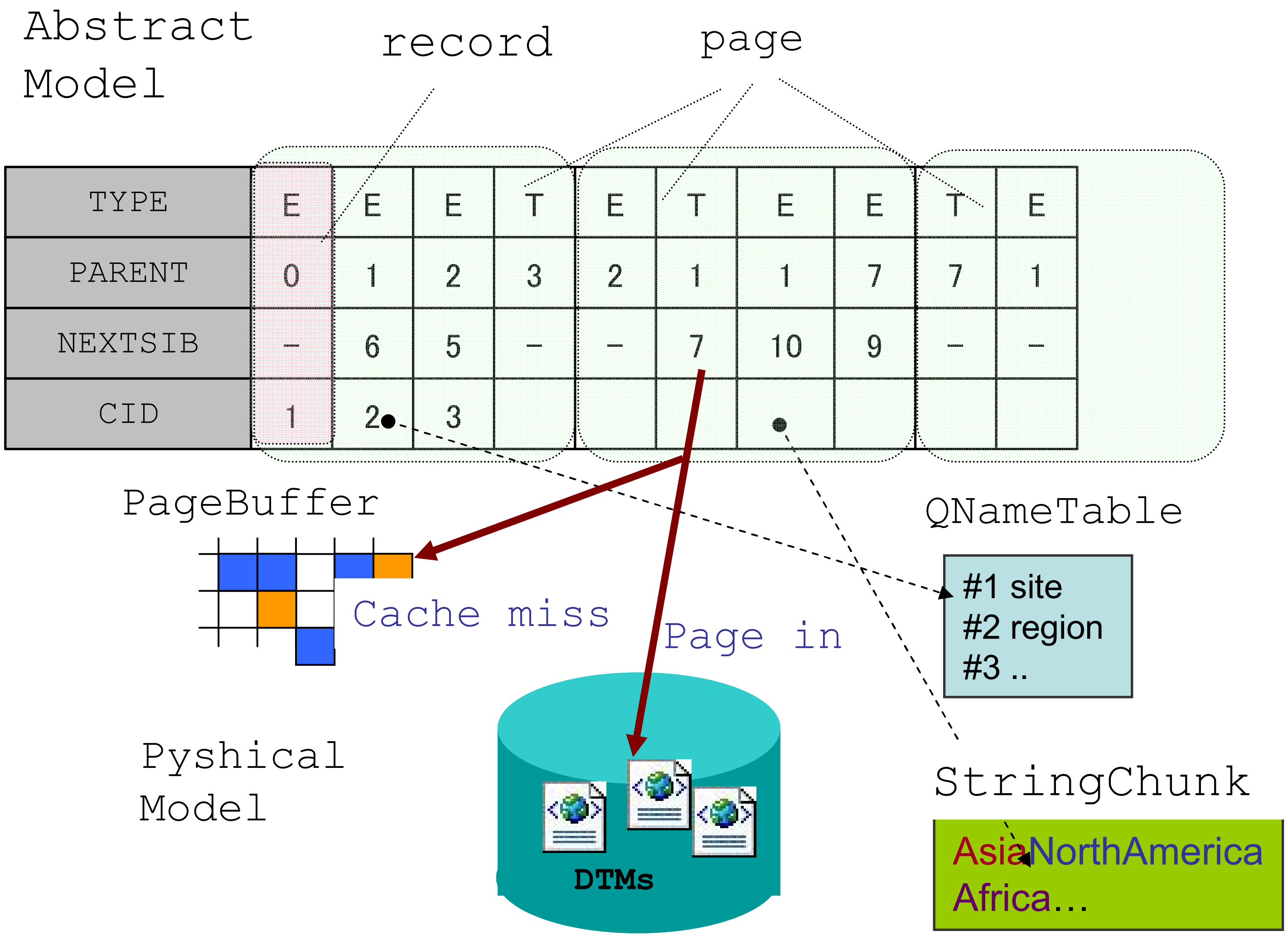
DTM table consists of primitive data types



	1	2	3	4	5	6	7	8	9	10
Event	E	E	E	T	E	T	E	E	T	E
PARENT	0	1	2	3	2	1	1	7	7	1
NEXTSIB	-	6	5	-	-	7	10	9	-	-
CID										

An XML tree structure can be represented as a table by using link values

System Overview



TYPE	E	E	E	T	E	T	E	E	T	E
PARENT	0	1	2	3	2	1	1	7	7	1
NEXTSIB	–	6	5	–	–	7	10	9	–	–
CID	1	2	3							

QNameTable

#1 site
#2 region
#3 ..

StringChunk

AsiaNorthAmerica
Africa…

Pyshical layout

Analyzing data access patterns

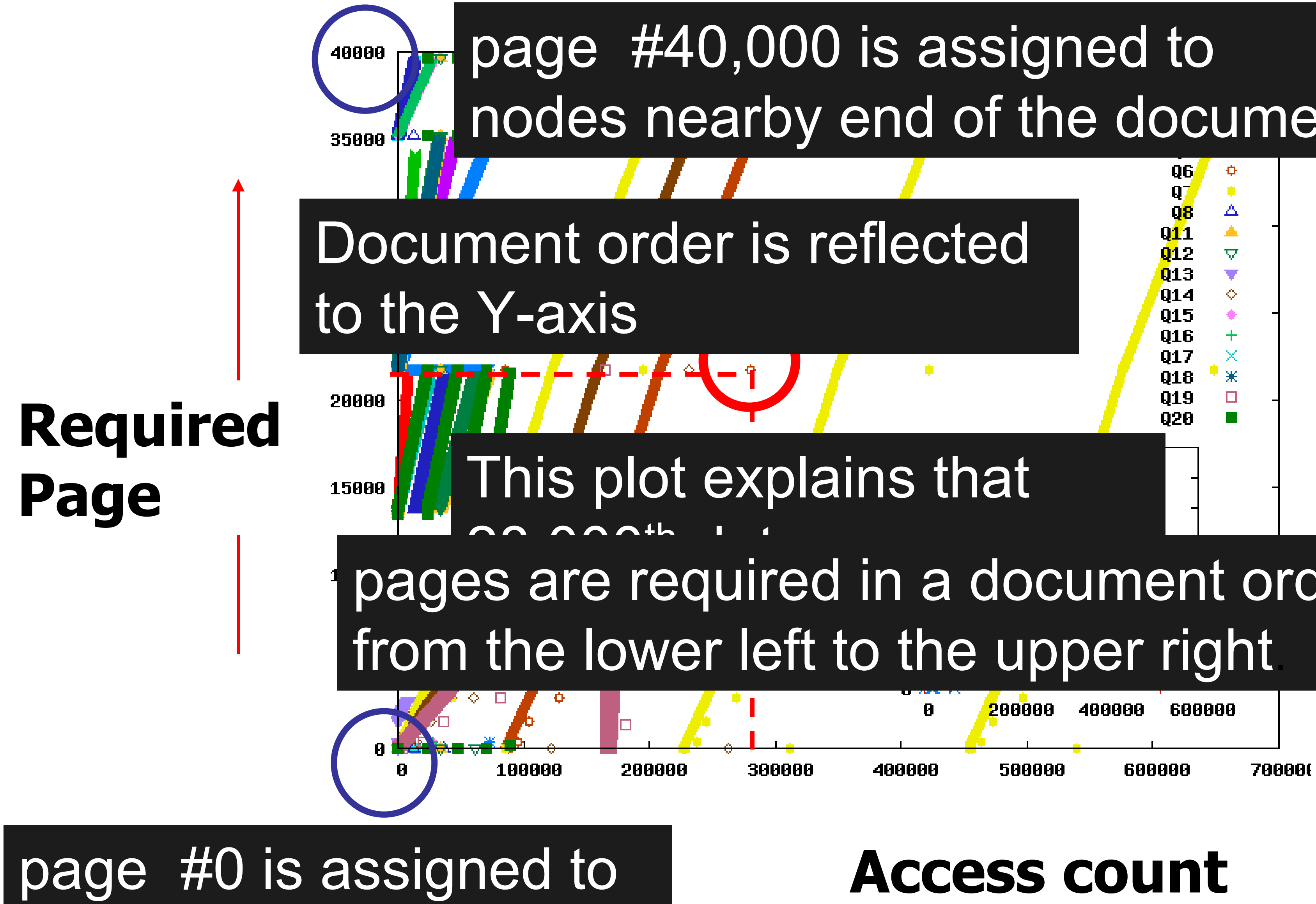
- Before designing physical layout of XML documents, we analyzed actual data access patterns of XQuery queries.

⬇

In general,

- Pages are required in the document-order
- Sequential accesses are frequently appeared

Access pattern analysis of XMark queries (1)



page #40,000 is assigned to nodes nearby end of the document

Document order is reflected to the Y-axis

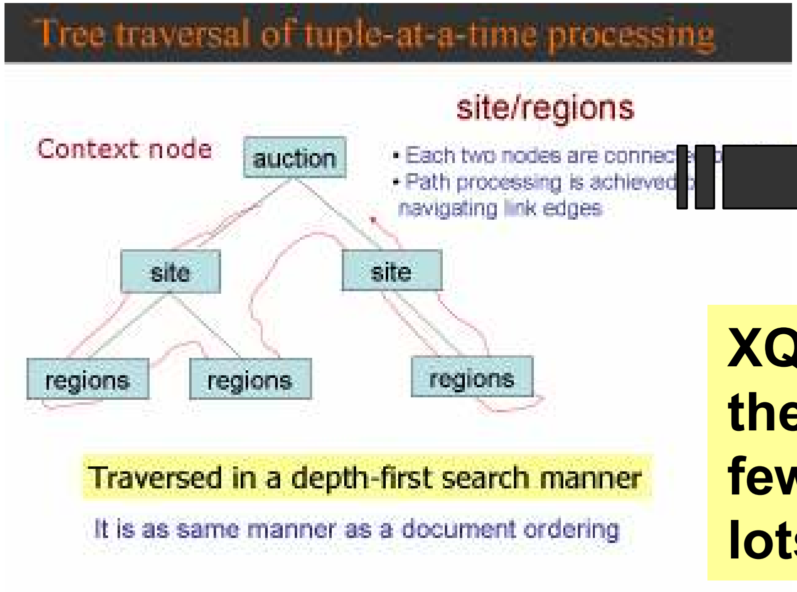
This plot explains that

pages are required in a document order from the lower left to the upper right.

page #0 is assigned to nodes nearby the root

Access pattern analysis of XMark queries (2)

Recall that we claimed that tuple-at-a-time processing of XPath queries, in general, traverses XML-tree according to the document-ordering.



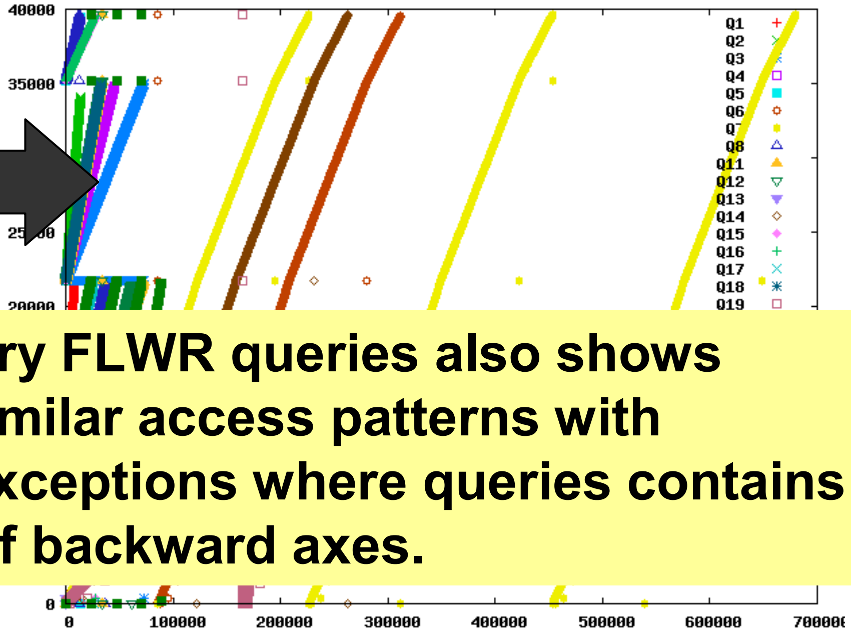
Tree traversal of tuple-at-a-time processing

site/regions

- Each two nodes are connec
- Path processing is achieved navigating link edges

Traversed in a depth-first search manner

It is as same manner as a document ordering



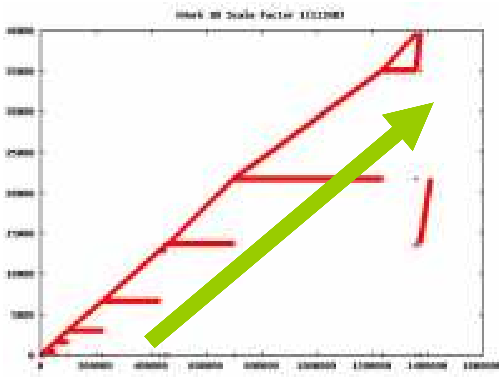
XQuery FLWR queries also shows the similar access patterns with few exceptions where queries contains lots of backward axes.

Note that the overall tendency is not restricted to XMark queries but also other benchmark queries.

Pyshical layout

- Pages are required in a document-order
- Sequential accesses are frequently appeared

- Document-ordered block allocation is suitable
- Prefetching is effective



The prefetching entries can compete for hot cache entries

We conducted informed prefetching with scan-resistant buffer management

- Scan-resistant buffer management

Is also effective to sequential scans in XML query processing

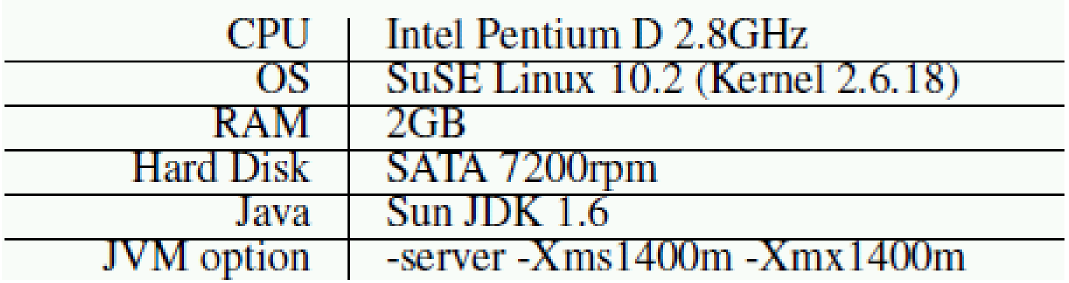
Outline

- Motivation
- Related work
- Document Table Model (DTM)
- XML Storage based on DTM (pDTM)
  - System Overview
  - Pyshical Layout
  - Buffer Management
- Experimental Evaluation
- Conclusions

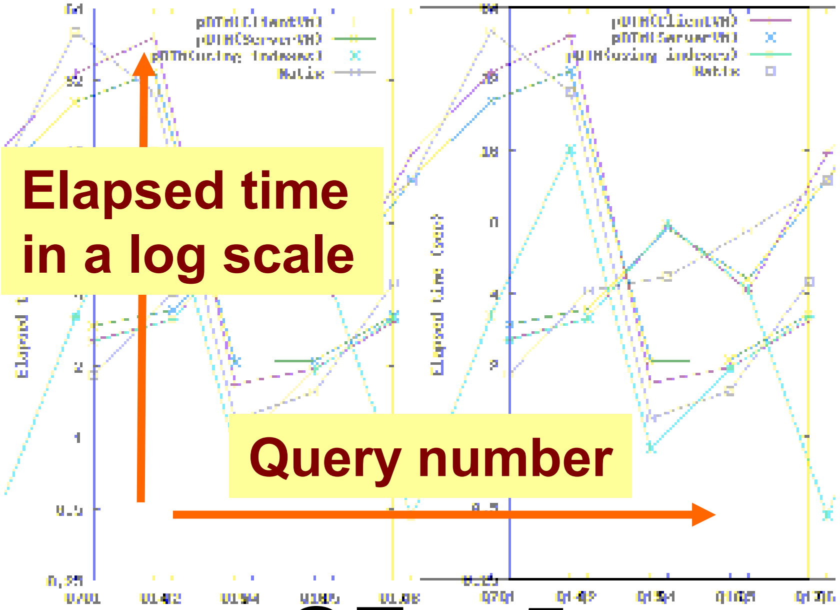
Experimental evaluation

- Compared to Natix version 2.1.1 where XMark SF = 5 and SF = 10
- Experimental settings

Today's normal PC setting



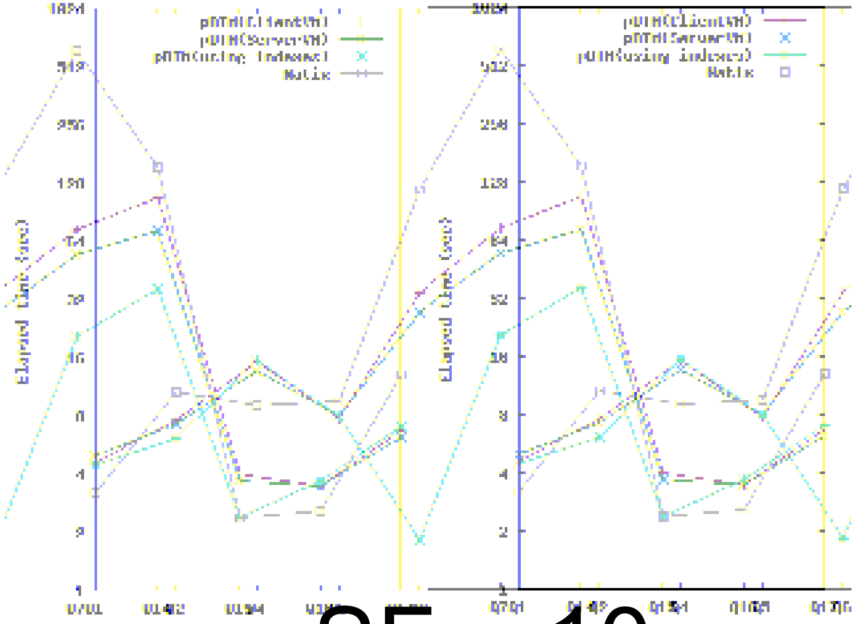
CPU	Intel Pentium D 2.8GHz
OS	SuSE Linux 10.2 (Kernel 2.6.18)
RAM	2GB
Hard Disk	SATA 7200rpm
Java	Sun JDK 1.6
JVM option	-server -Xms1400m -Xmx1400m



Elapsed time in a log scale

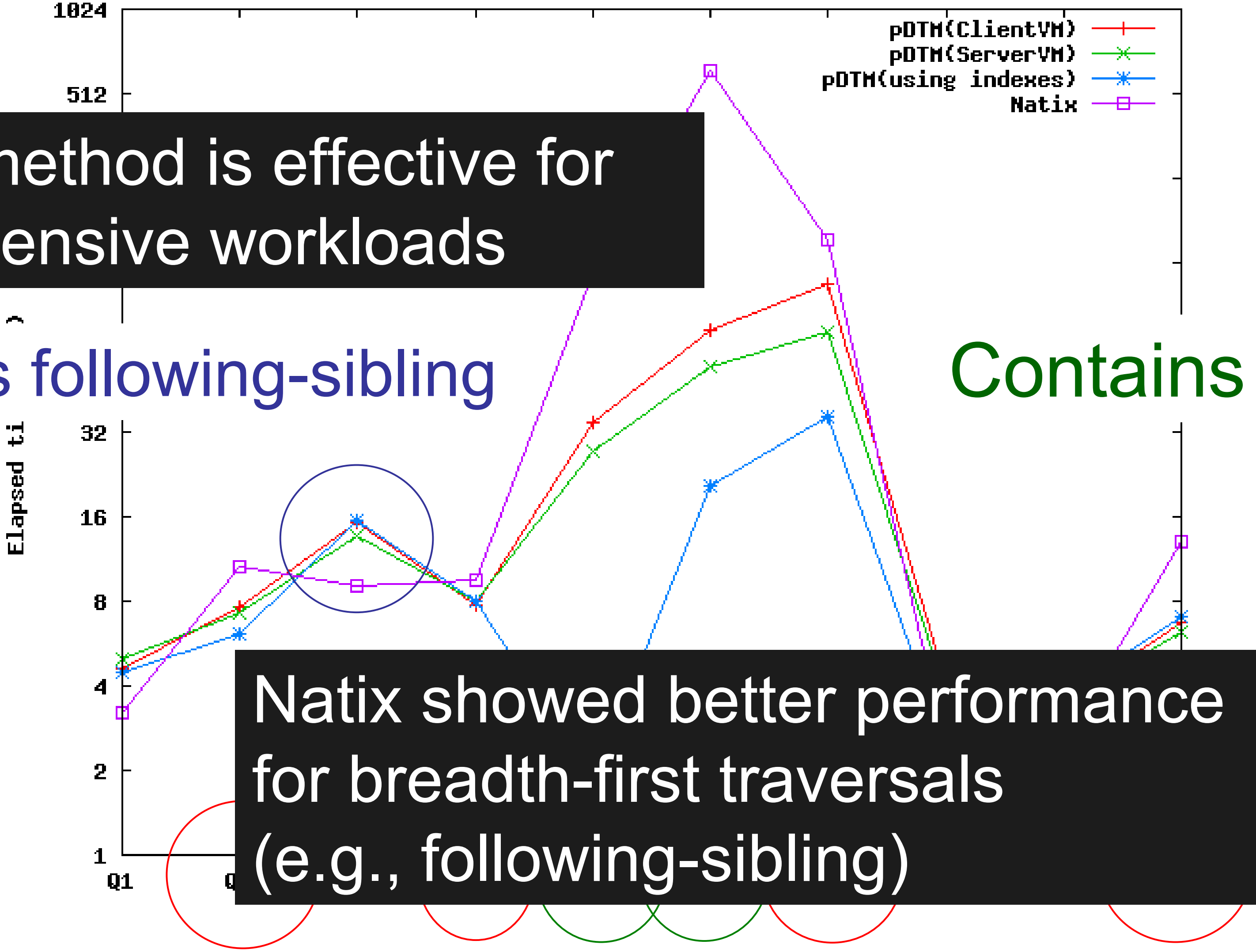
Query number

SF = 5



SF = 10

XMark SF=10



Our method is effective for IO-intensive workloads

Has following-sibling

Contains "//"

Natix showed better performance for breadth-first traversals (e.g., following-sibling)

Queries whose outputs are large

Conclusions

Summary

- Proposed an efficient XML storage scheme base on DTM for iterative XQuery processing

- Our approach is effective for IO-intensive workloads such as queries including '//'.
  - Document-ordered block allocation
  - Informed prefetching and scan-resistant caching

Future work

- Automatic database tuning based on online analysis of data access patterns (e.g., buffer replacement policy and prefetching)

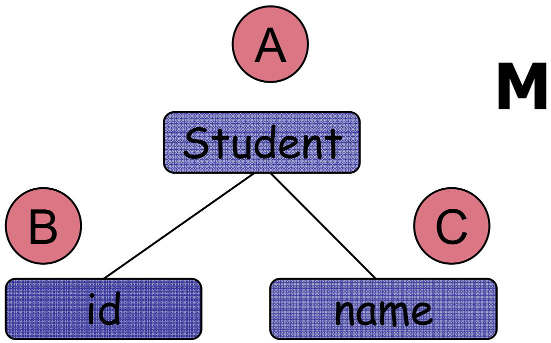
Thank you for your attention!

Questions?

Problem in XML-Relational mapping

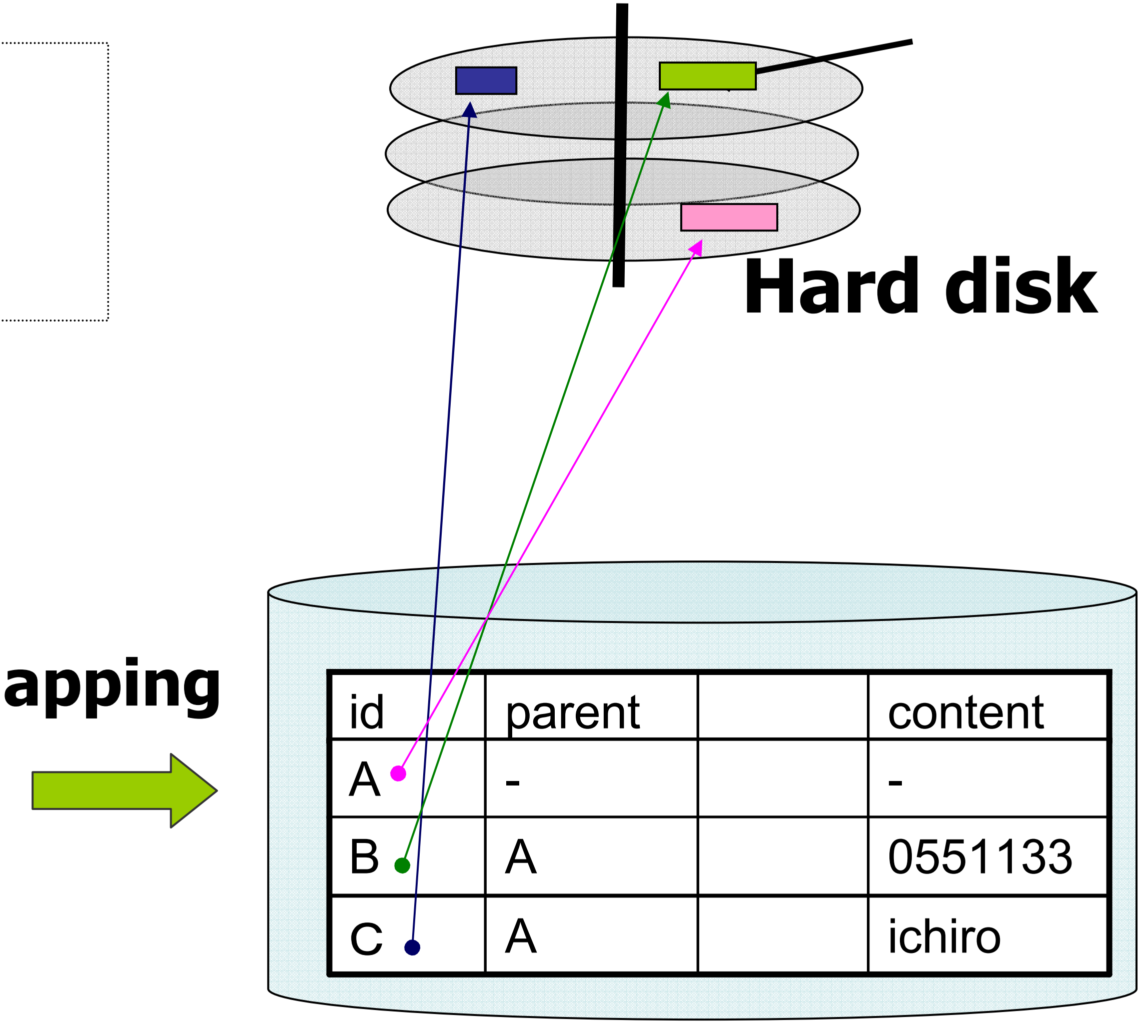
XML

```
<student>
  <id>0551133</id>
  <name>ichiro</name>
</student>
```



XML Tree

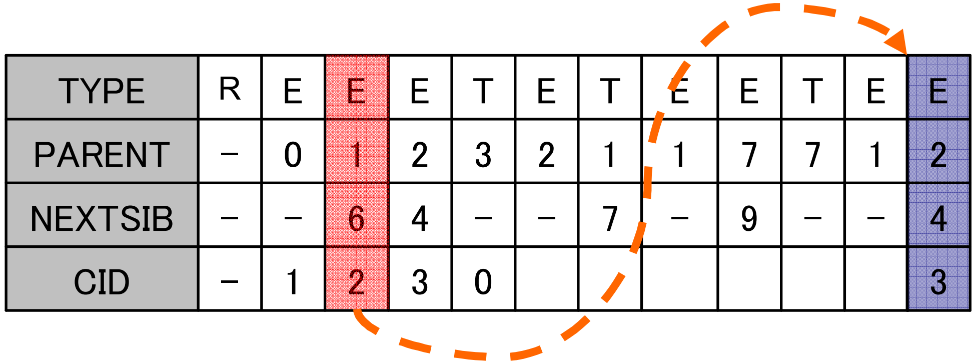
Mapping



id	parent		content
A	-		-
B	A		0551133
C	A		ichiro

Relational Table

Updating facilities and versioning

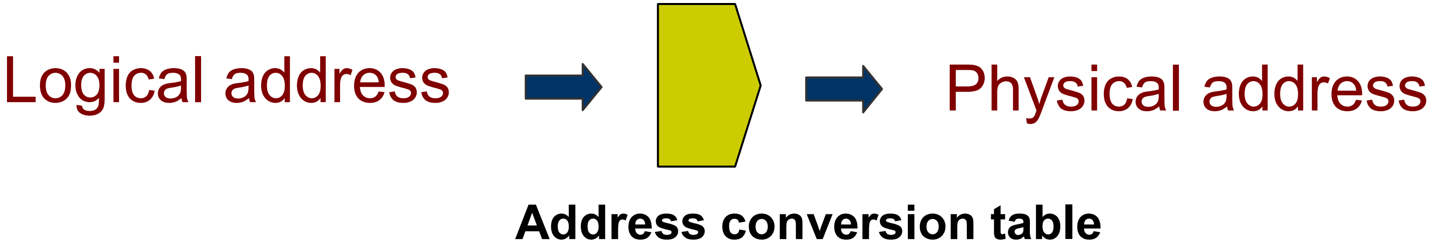


TYPE	R	E	E	E	T	E	T	E	E	T	E	E
PARENT	–	0	1	2	3	2	1	1	7	7	1	2
NEXTSIB	–	–	6	4	–	–	7	–	9	–	–	4
CID	–	1	2	3	0							3

When accessing to a record,

Logical address → Physical address

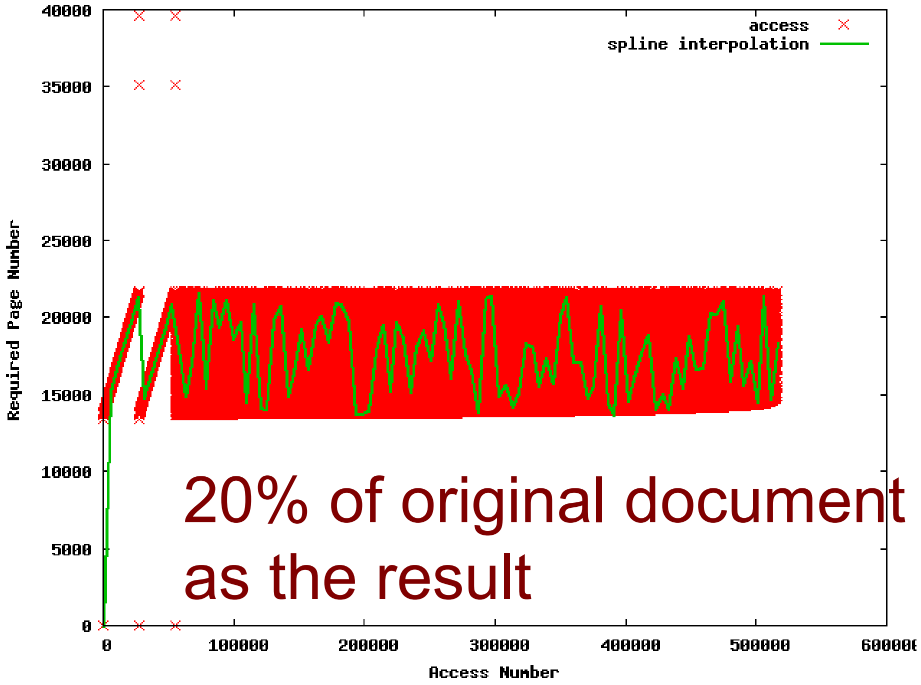
For updating facilities, we change this method as follows:

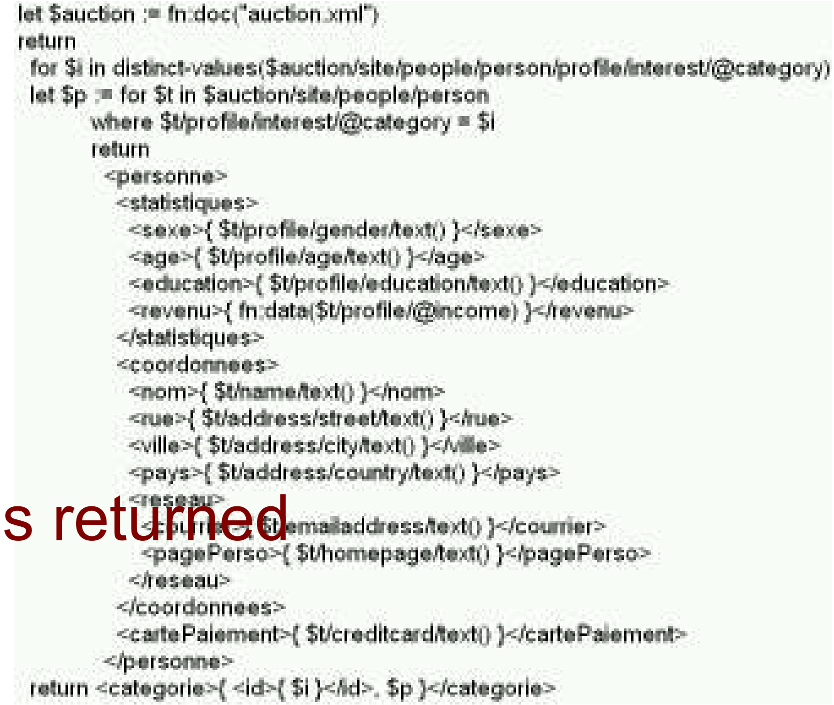


Logical address → → Physical address

Address conversion table

Buffer management (XMark Q10 as an example)



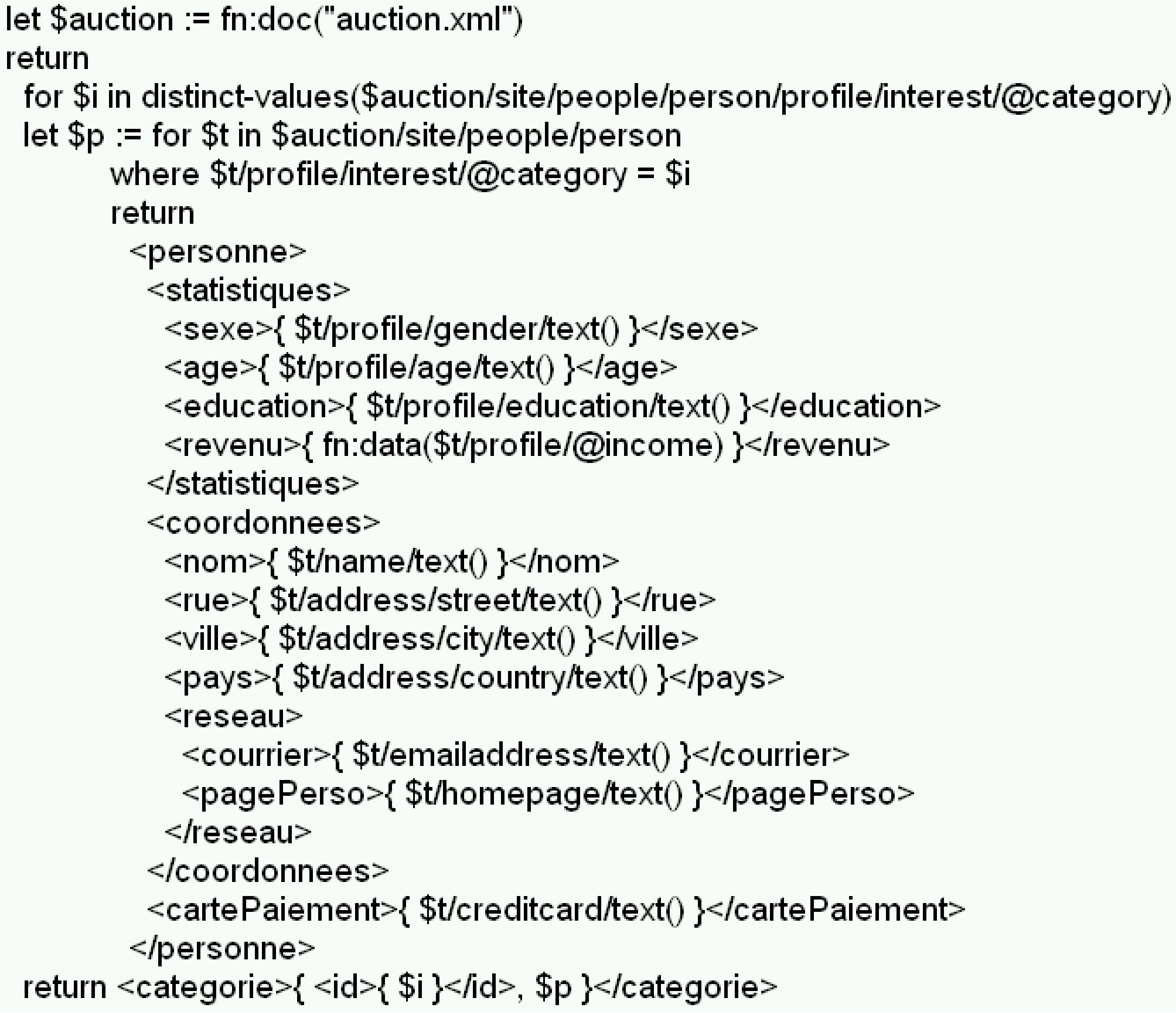


```
let $auction := fn:doc("auction.xml")
return
  for $i in distinct-values($auction/site/people/person/profile/interest/@category)
  let $p := for $t in $auction/site/people/person
            where $t/profile/interest/@category = $i
            return
              <personne>
                <statistiques>
                  <sexe>{ $t/profile/gender/text() }</sexe>
                  <age>{ $t/profile/age/text() }</age>
                  <education>{ $t/profile/education/text() }</education>
                  <revenu>{ fn:data($t/profile/@income) }</revenu>
                </statistiques>
                <coordonnees>
                  <nom>{ $t/name/text() }</nom>
                  <rue>{ $t/address/street/text() }</rue>
                  <ville>{ $t/address/city/text() }</ville>
                  <pays>{ $t/address/country/text() }</pays>
                  <reseau>
                    <courrier>{ $t/emailaddress/text() }</courrier>
                    <pagePerso>{ $t/homepage/text() }</pagePerso>
                  </reseau>
                </coordonnees>
                <cartePaiement>{ $t/creditcard/text() }</cartePaiement>
              </personne>
  return <categorie>{ <id>{ $i }</id>, $p }</categorie>
```

20% of original document is returned as the result

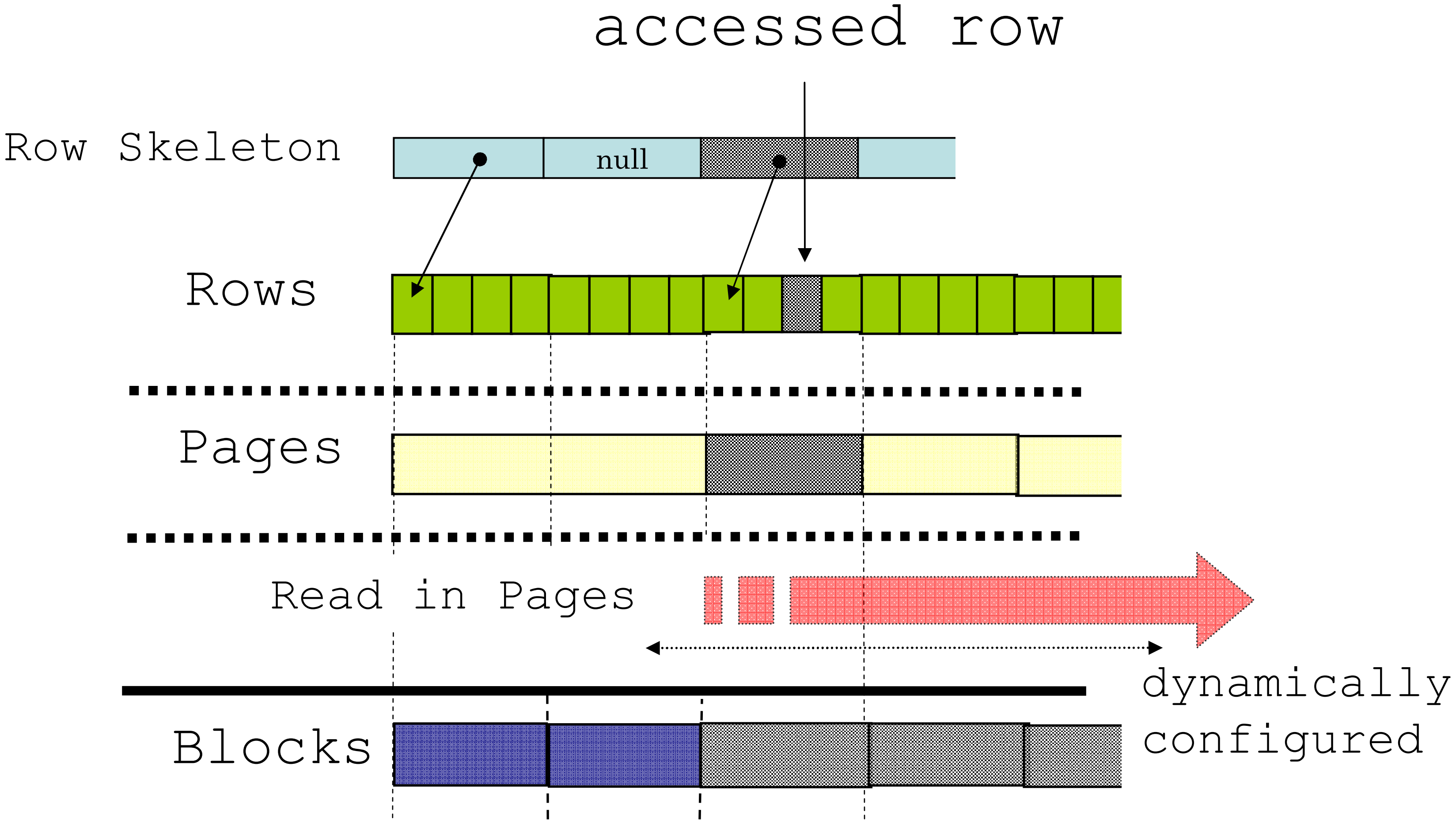
	Elapsed time (msec)	total read blocks	buffer replacement
LRU	211.83	1,919,586	567,702
2Q	185.56	80,673	0

14.2% speedups



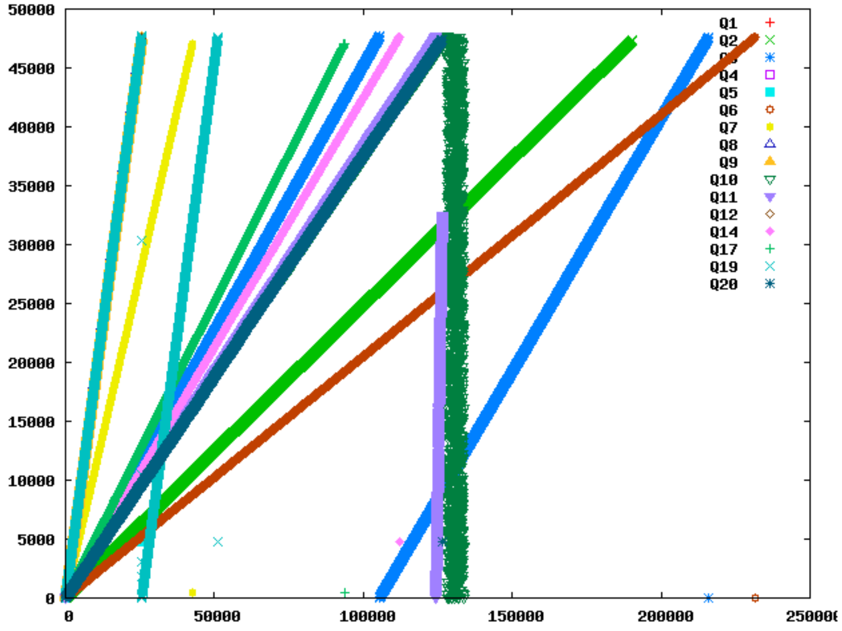
```
let $auction := fn:doc("auction.xml")
return
  for $i in distinct-values($auction/site/people/person/profile/interest/@category)
  let $p := for $t in $auction/site/people/person
            where $t/profile/interest/@category = $i
            return
              <personne>
                <statistiques>
                  <sexe>{ $t/profile/gender/text() }</sexe>
                  <age>{ $t/profile/age/text() }</age>
                  <education>{ $t/profile/education/text() }</education>
                  <revenu>{ fn:data($t/profile/@income) }</revenu>
                </statistiques>
                <coordonnees>
                  <nom>{ $t/name/text() }</nom>
                  <rue>{ $t/address/street/text() }</rue>
                  <ville>{ $t/address/city/text() }</ville>
                  <pays>{ $t/address/country/text() }</pays>
                  <reseau>
                    <courrier>{ $t/emailaddress/text() }</courrier>
                    <pagePerso>{ $t/homepage/text() }</pagePerso>
                  </reseau>
                </coordonnees>
                <cartePaiement>{ $t/creditcard/text() }</cartePaiement>
              </personne>
  return <categorie>{ <id>{ $i }</id>, $p }</categorie>
```

Pyshical layout

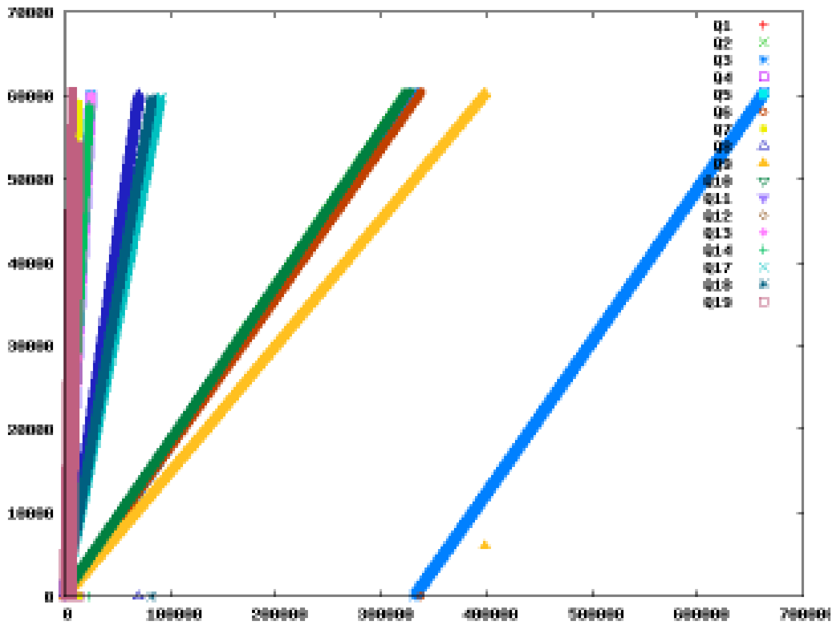


Access pattern analysis of XBench queries

DC/SD Normal



TC/SD Normal



Memory Mapped DTM

```
#include <sys/mman.h>

void *mmap(void *start, size_t length, int prot, int flags,
        int fd, off_t offset);

int munmap(void *start, sizt_t length);
```

We present a memory mapped scheme extending the DTM model, it has boost the performance significantly.

Efficient XML Storage based on DTM for Read-oriented Workloads

Graduate School of Information Science,
Nara Institute of Science and Technology

Makoto Yui

Jun Miyazaki, Shunsuke Uemura, Hirokazu Kato

International Workshop on Advanced Storage Systems 2007 (ADSS 2007)

Outline

- Motivation
- Related work
- Document Table Model (DTM)
- XML storage based on DTM
  - System Overview
  - Physical Layout
  - Buffer Management
- Experimental Evaluation
- Conclusions

Motivation - Backgrounds

- Past research topics in XML data management

Labeling and indexing XML trees

- Dewey ordering
- Ordpaths
- XR-Trees

Join processing

- Structural-joins
- Twig-joins

Indexing on paths

- XRel
- Index Fabric

Well studied topics

Towards efficient XML data processing

Internal data model

- Relational
- Hybrid (SystemRX)
- Tree (e.g., DOM)

Buffer management

Physical data layout

- Natix

Less studied topics

Our focus

Motivation – Design goals

- Design an **XML storage scheme** optimized for read-oriented workload
  - Node-level update is not always required
  - Updating capabilit

The design of read-o ested for relational databases es yet

The reason we selected this model is set-at-a-time processing of XQuery involves lots of joins, and it causes performance deteriation.

- Focus on **iterative XQuery processing** in which an operator tree consists of iterators
  - Ideal XML storage scheme depends on the processing model (e.g., tuple-at-a-time or set-at-a-time)

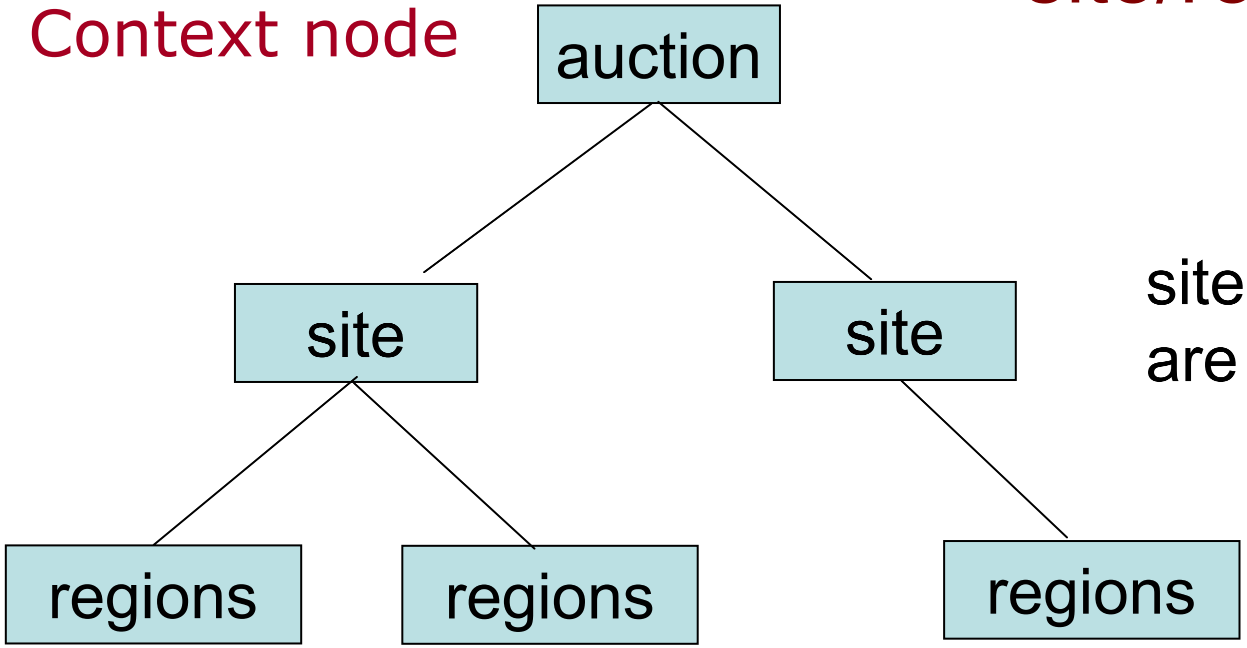
How the data access is achieved for each of the two processing model ?

Norman May, et al., Index vs. Navigation in XPath Evaluation. XSym 2006

Tree traversal of set-at-a-time processing

site/regions

Context node



sites under actions
are selected

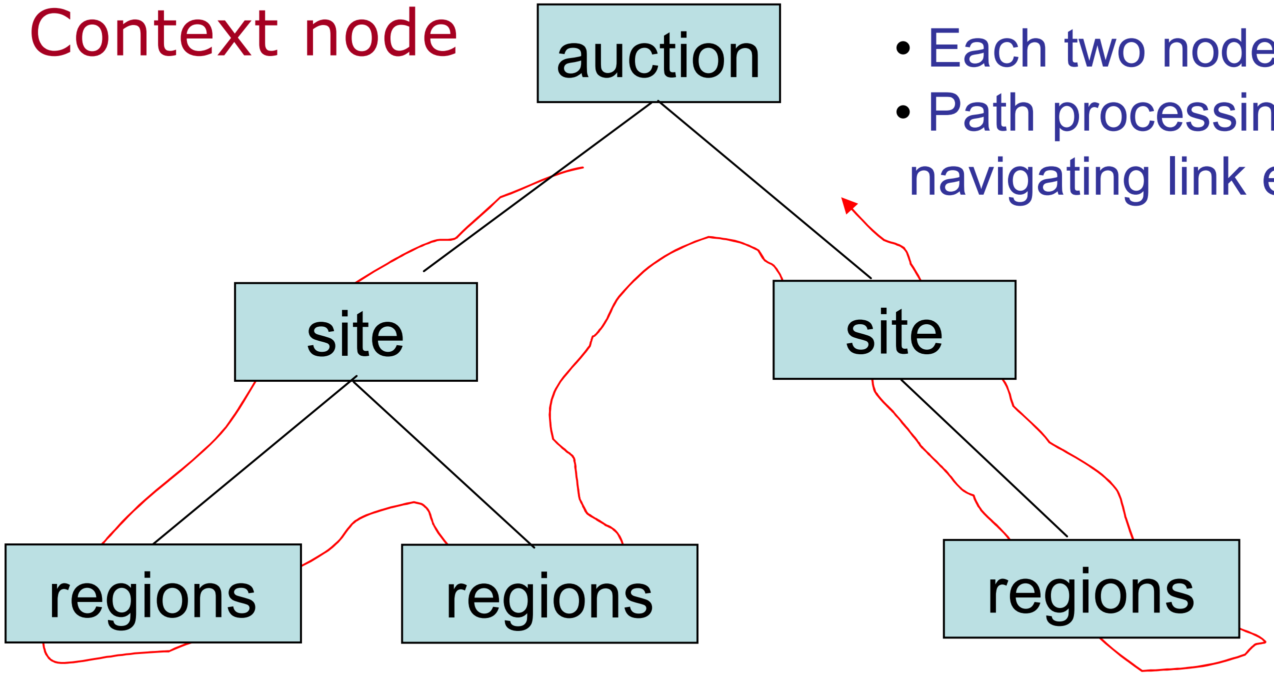
regions under sites
are selected

Each edges are connected by join
on nodes' identifiers

Traversed in a breadth-first search manner

Tree traversal of tuple-at-a-time processing

site/regions



- Each two nodes are connected by links
- Path processing is achieved by navigating link edges

Traversed in a depth-first search manner

It is as same manner as a document ordering

Motivation

- Design an **XML storage scheme** optimized for read-oriented workload

- Focus on **iterative XQuery processing** in which an operator tree consists of iterators

↓

We examined **actual data access patterns**
when evaluating XQuery queries
in order to design the suitable data layout

Outline

- Motivation
- Related Work
- Document Table Model (DTM)
- XML Storage based on DTM (pDTM)
  - System Overview
  - Pyshical Layout
  - Buffer Management
- Experimental Evaluation
- Conclusions

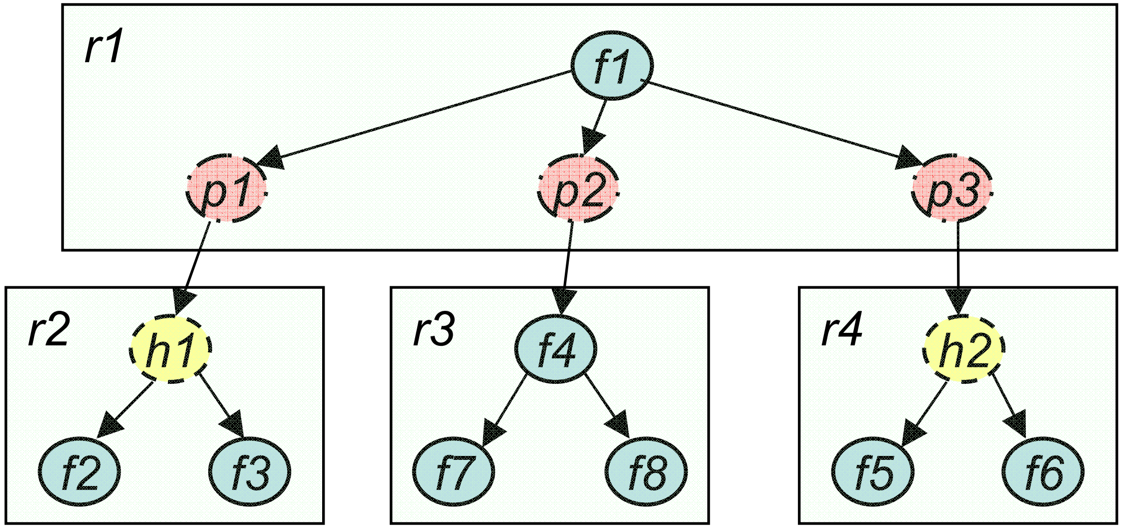
Related work (1) storing scheme based on subtrees

Natix (University of Mannheim, Germany)

T. Fiebig, et.al. Anatomy of a Native XML Base Management System. VLDB Journal, 2002.

Allocates a page based on subtrees

- Pros Effective for breadth-first traversals
- Cons Not effective for depth-first traversals



P Proxy Node h Helper aggregate nodes:

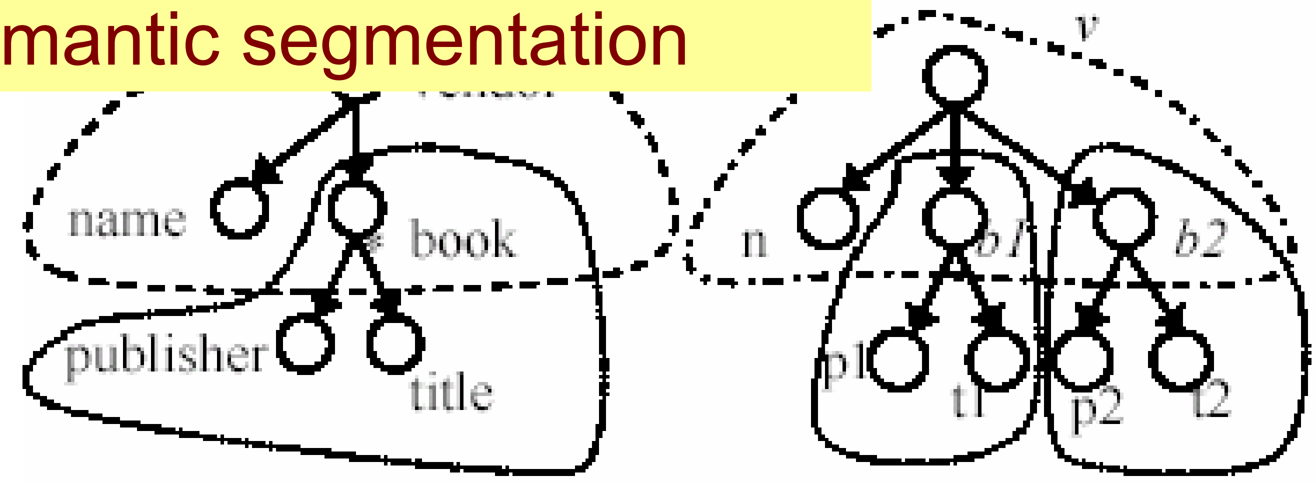
Related work (2) storing scheme based on a schema

OrientStore (Renmin University, China)

X. Meng et.al. OrientStore: A Schema Based Native XML Storage System. VLDB 2003.

- Pros
  - Effective for path processing
- Cons
  - Schema information is required
  - Not effective for serialization and
    ion

Clusters records according to a semantic segmentation



a) Semantic Blocks

b) Records

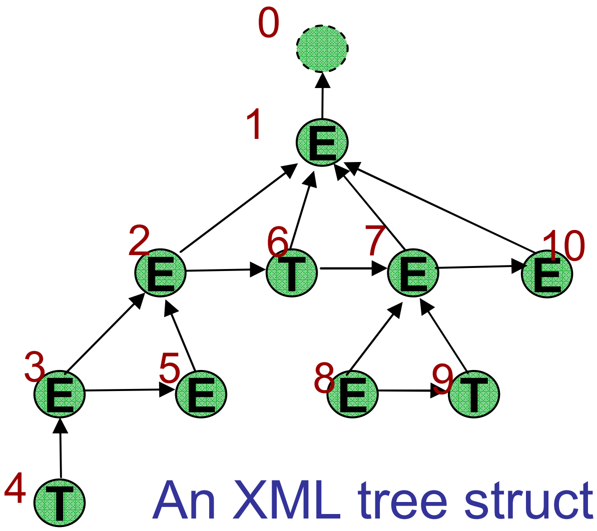
Outline

- Motivation
- Related work
- Document Table Model (DTM)
- XML Storage based on DTM (pDTM)
  - System Overview
  - Pyshical Layout
  - Buffer Management
- Experimental Evaluation
- Conclusions

Document Table Model (DTM)

- Originally used by Apache Xalan XSLT processor
- Expresses an XML document as a table form
  - DOM has object footprints (e.g., object instantiation and memory consumptions)
  - DTM can avoid such object footprints

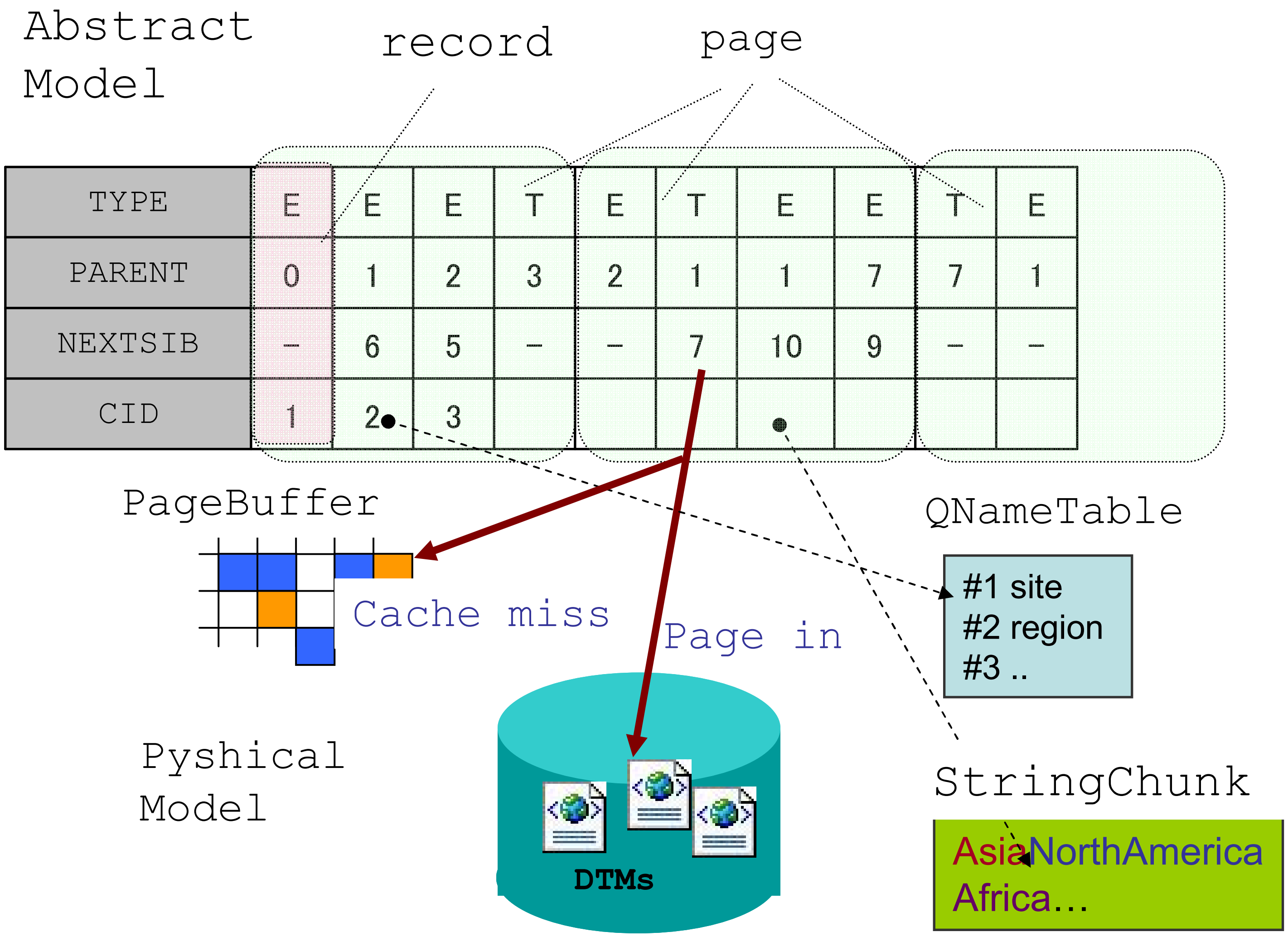
DTM table consists of primitive data types



	1	2	3	4	5	6	7	8	9	10
Event	E	E	E	T	E	T	E	E	T	E
PARENT	0	1	2	3	2	1	1	7	7	1
NEXTSIB	-	6	5	-	-	7	10	9	-	-
CID										

An XML tree structure can be represented as a table by using link values

System Overview



Abstract Model

record

page

TYPE	E	E	E	T	E	T	E	E	T	E
PARENT	0	1	2	3	2	1	1	7	7	1
NEXTSIB	–	6	5	–	–	7	10	9	–	–
CID	1	2	3							

PageBuffer

Cache miss

Page in

QNameTable

#1 site
#2 region
#3 ..

Pyshical Model

DTMs

StringChunk

AsiaNorthAmerica
Africa…

Pyshical layout

Analyzing data access patterns

- Before designing physical layout of XML documents, we analyzed actual data access patterns of XQuery queries.

In general,

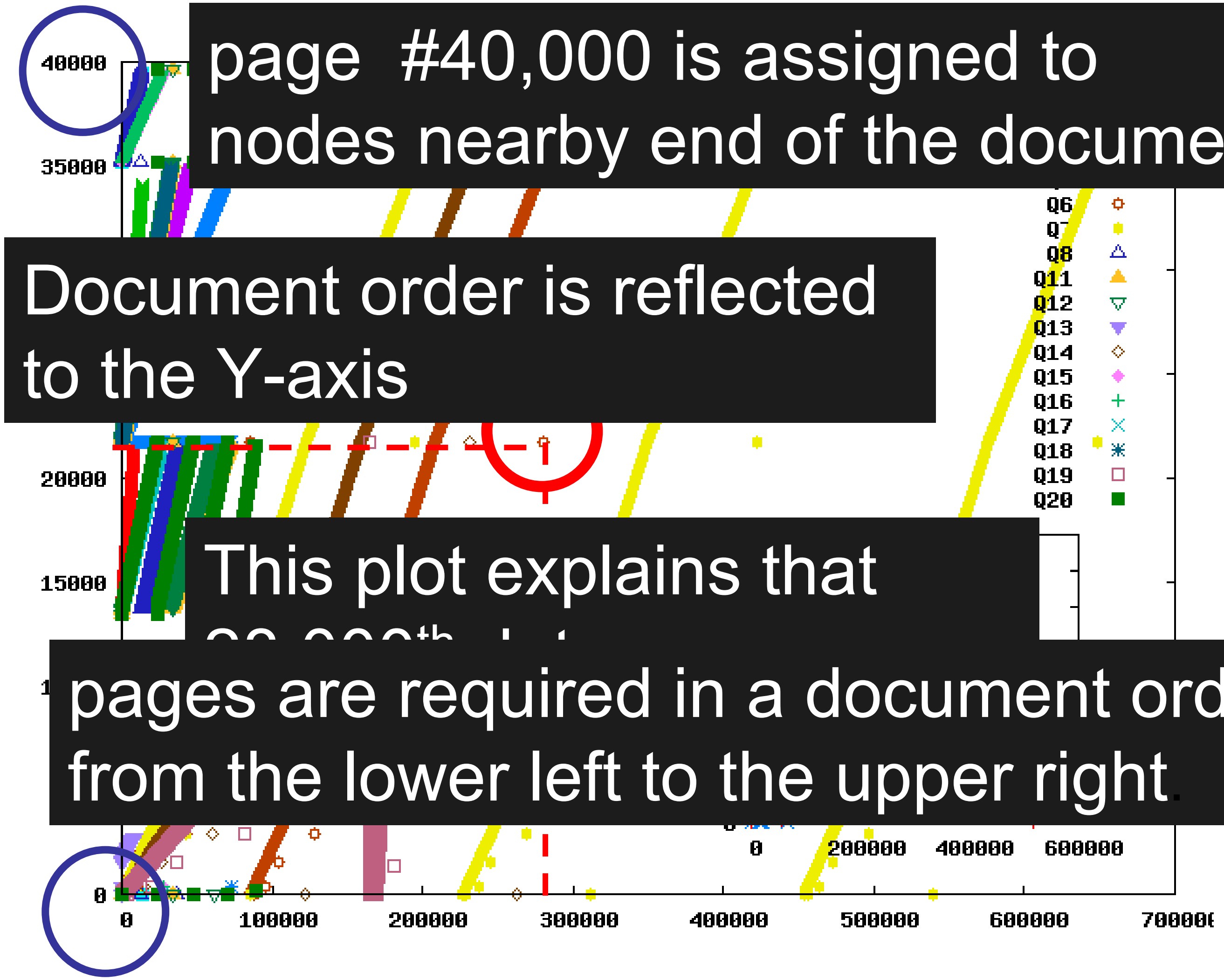
- Pages are required in the document-order
- Sequential accesses are frequently appeared

Access pattern analysis of XMark queries (1)

page #40,000 is assigned to nodes nearby end of the document

Document order is reflected to the Y-axis

Required Page



This plot explains that

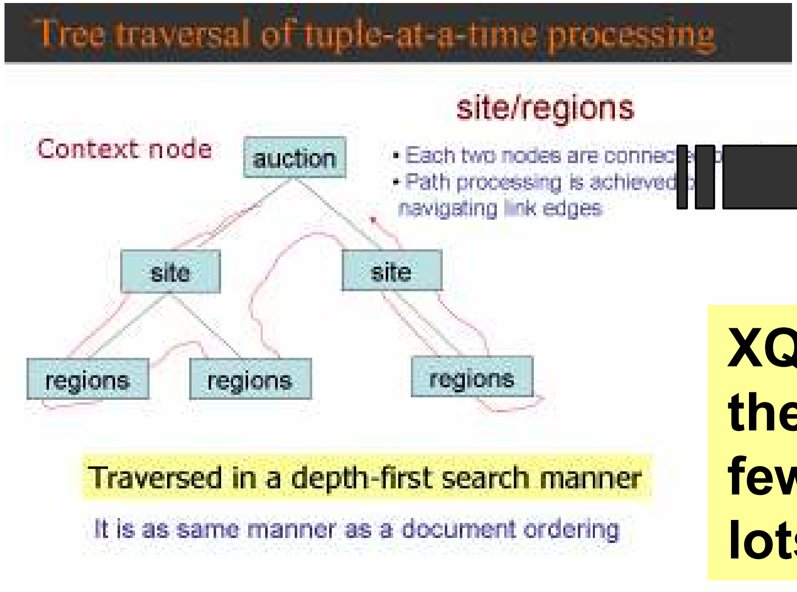
pages are required in a document order from the lower left to the upper right.

Access count

page #0 is assigned to nodes nearby the root

Access pattern analysis of XMark queries (2)

Recall that we claimed that tuple-at-a-time processing of XPath queries, in general, traverses XML-tree according to the document-ordering.



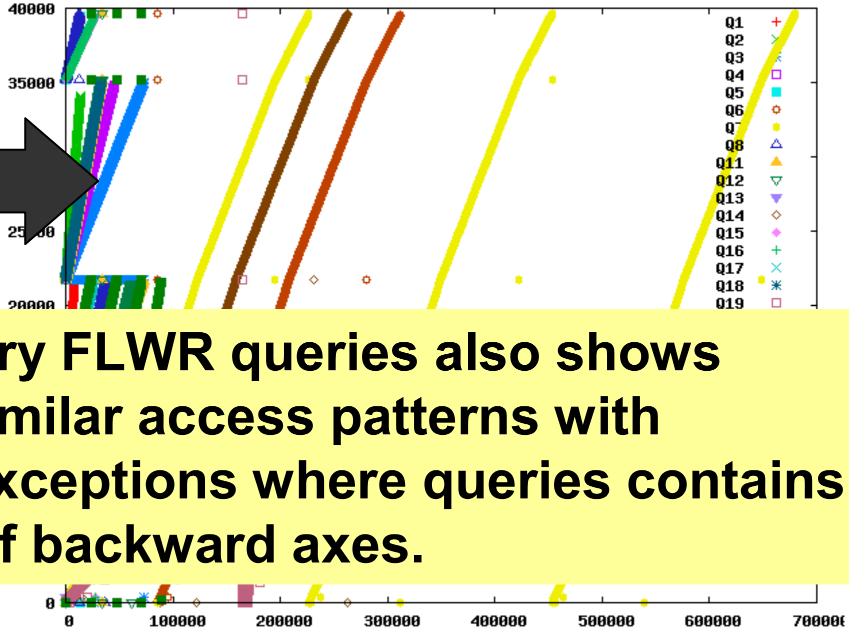
Tree traversal of tuple-at-a-time processing

site/regions

- Each two nodes are connec
- Path processing is achieved navigating link edges

Traversed in a depth-first search manner

It is as same manner as a document ordering



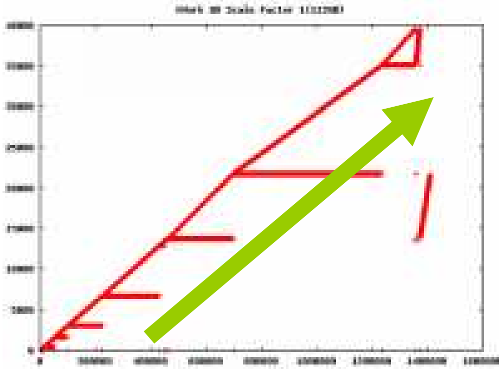
XQuery FLWR queries also shows the similar access patterns with few exceptions where queries contains lots of backward axes.

Note that the overall tendency is not restricted to XMark queries but also other benchmark queries.

Pyshical layout

- Pages are required in a document-order
- Sequential accesses are frequently appeared

- Document-ordered block allocation is suitable
- Prefetching is effective



The prefetching entries can compete for hot cache entries

We conducted informed prefetching with scan-resistant buffer management

- Scan-resistant buffer management

Is also effective to sequential scans in XML query processing

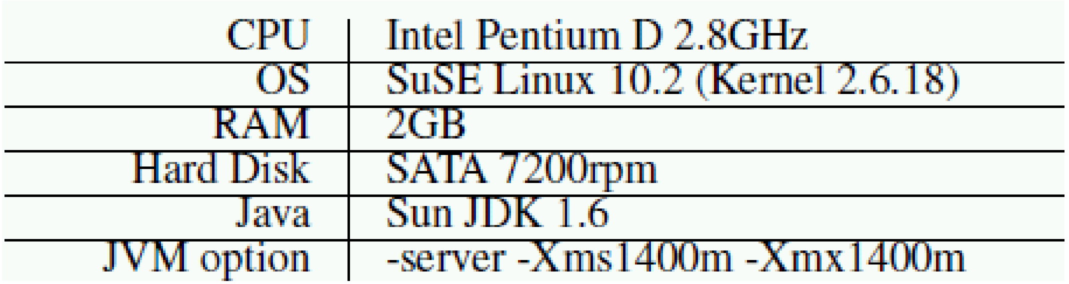
Outline

- Motivation
- Related work
- Document Table Model (DTM)
- XML Storage based on DTM (pDTM)
  - System Overview
  - Pyshical Layout
  - Buffer Management
- Experimental Evaluation
- Conclusions

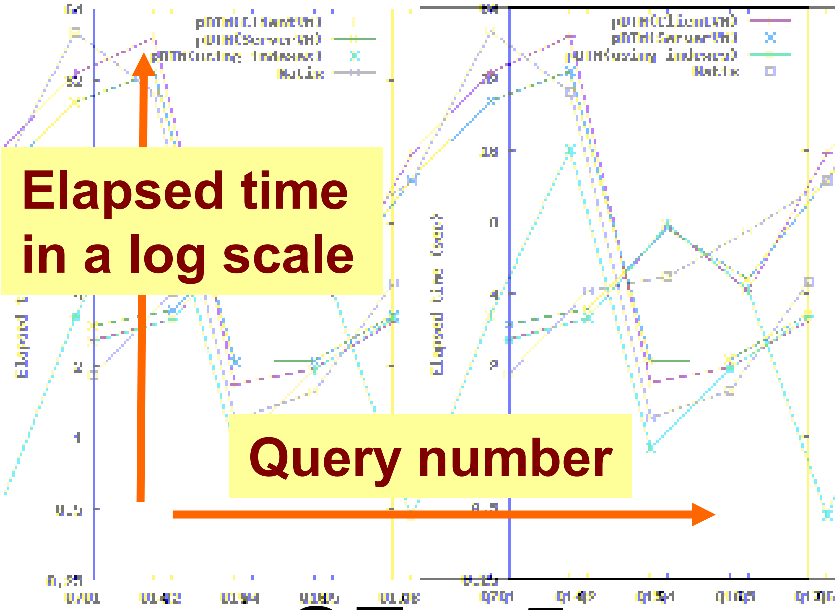
Experimental evaluation

- Compared to Natix version 2.1.1 where XMark SF = 5 and SF = 10
- Experimental settings

Today's normal PC setting



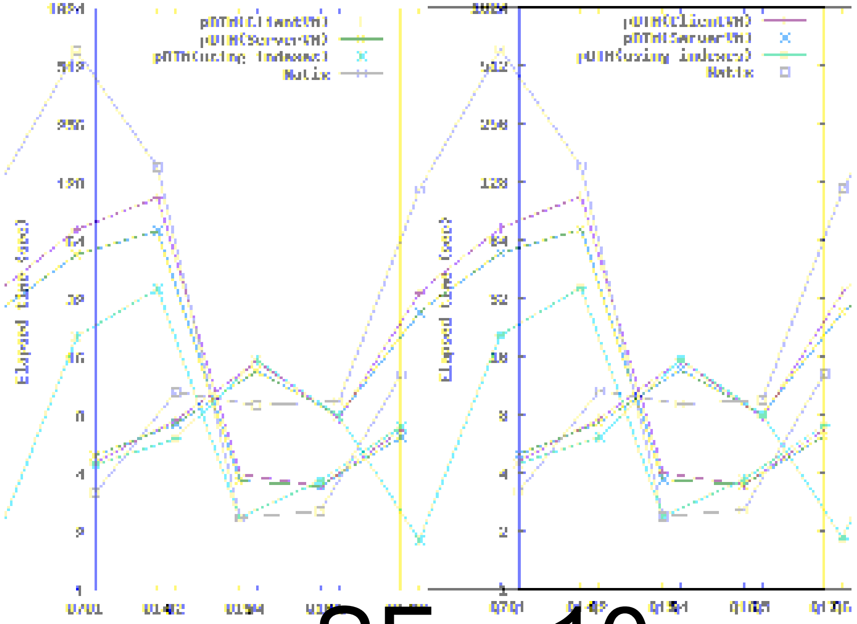
CPU	Intel Pentium D 2.8GHz
OS	SuSE Linux 10.2 (Kernel 2.6.18)
RAM	2GB
Hard Disk	SATA 7200rpm
Java	Sun JDK 1.6
JVM option	-server -Xms1400m -Xmx1400m



Elapsed time in a log scale

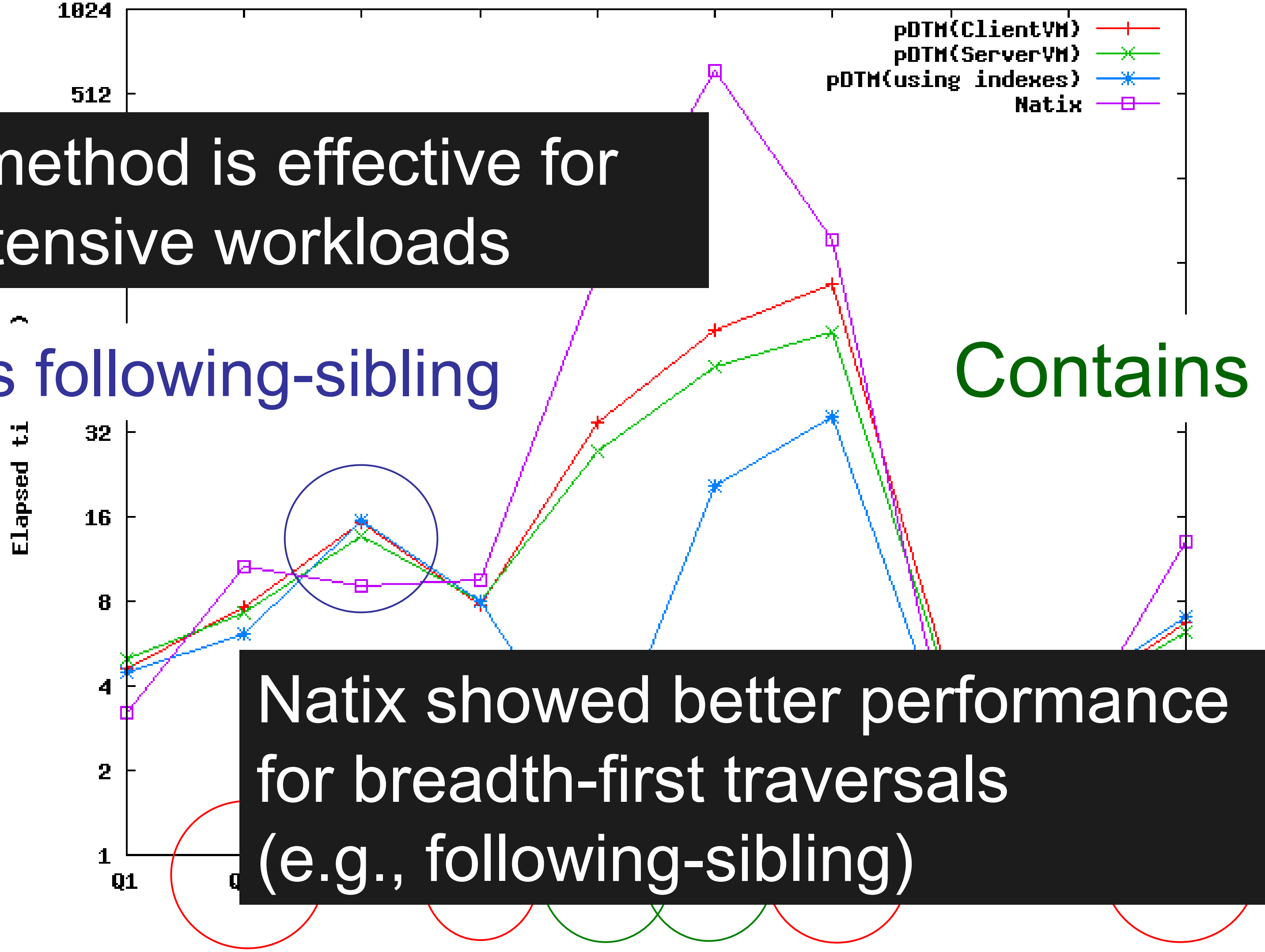
Query number

SF = 5



SF = 10

XMark SF=10



Our method is effective for IO-intensive workloads

Has following-sibling

Contains “//”

Natix showed better performance for breadth-first traversals (e.g., following-sibling)

Queries whose outputs are large

Conclusions

Summary

- Proposed an efficient XML storage scheme base on DTM for iterative XQuery processing

- Our approach is effective for IO-intensive workloads such as queries including '//'.
  - Document-ordered block allocation
  - Informed prefetching and scan-resistant caching

Future work

- Automatic database tuning based on online analysis of data access patterns (e.g., buffer replacement policy and prefetching)

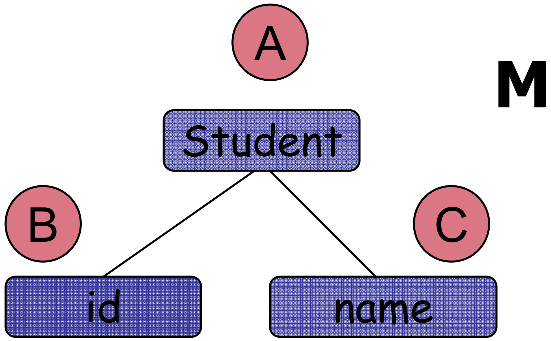
Thank you for your attention!

Questions?

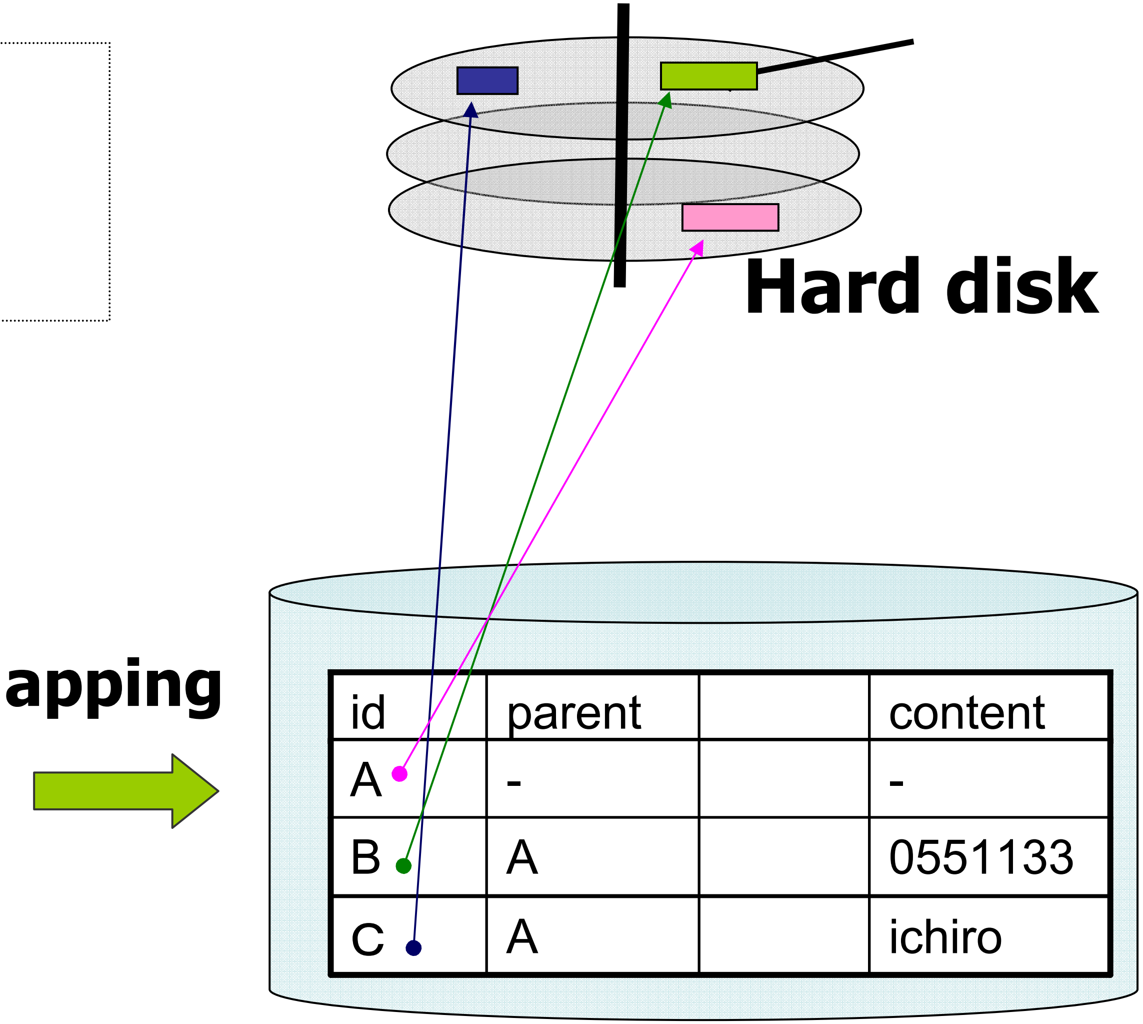
Problem in XML-Relational mapping

XML

```
<student>
  <id>0551133</id>
  <name>ichiro</name>
</student>
```



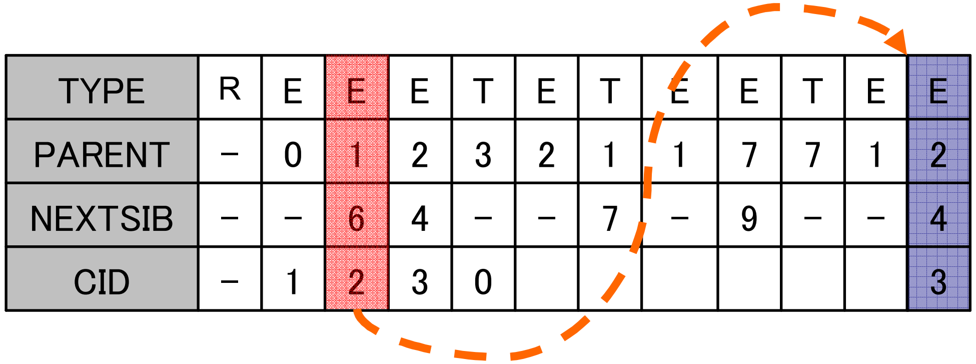
XML Tree



id	parent		content
A	-		-
B	A		0551133
C	A		ichiro

Relational Table

Updating facilities and versioning



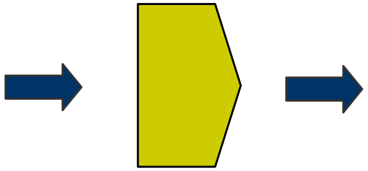
TYPE	R	E	E	E	T	E	T	E	E	T	E	E
PARENT	–	0	1	2	3	2	1	1	7	7	1	2
NEXTSIB	–	–	6	4	–	–	7	–	9	–	–	4
CID	–	1	2	3	0							3

When accessing to a record,

Logical address → Physical address

For updating facilities, we change this method as follows:

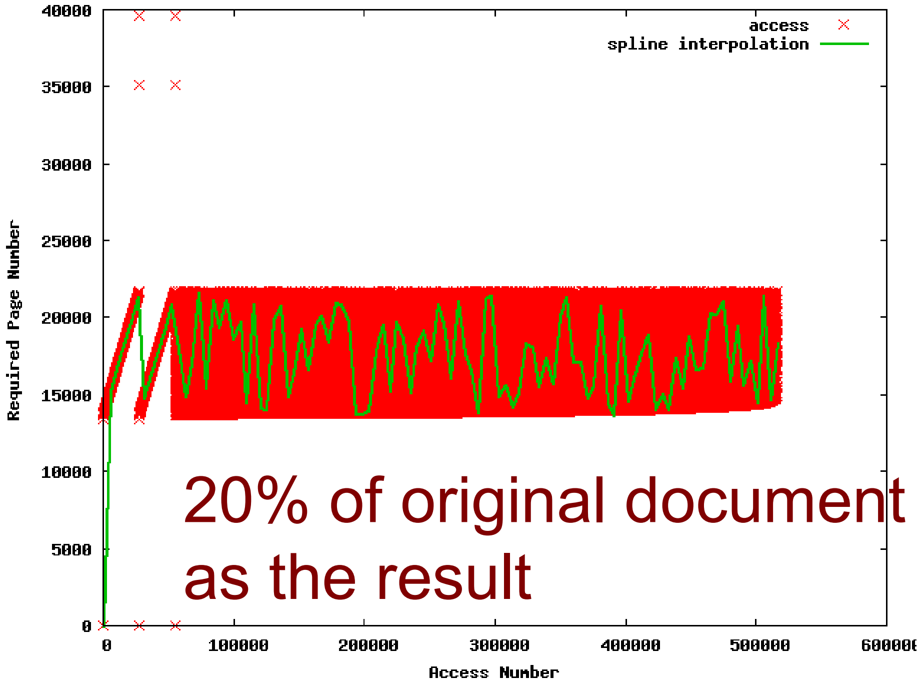
Logical address

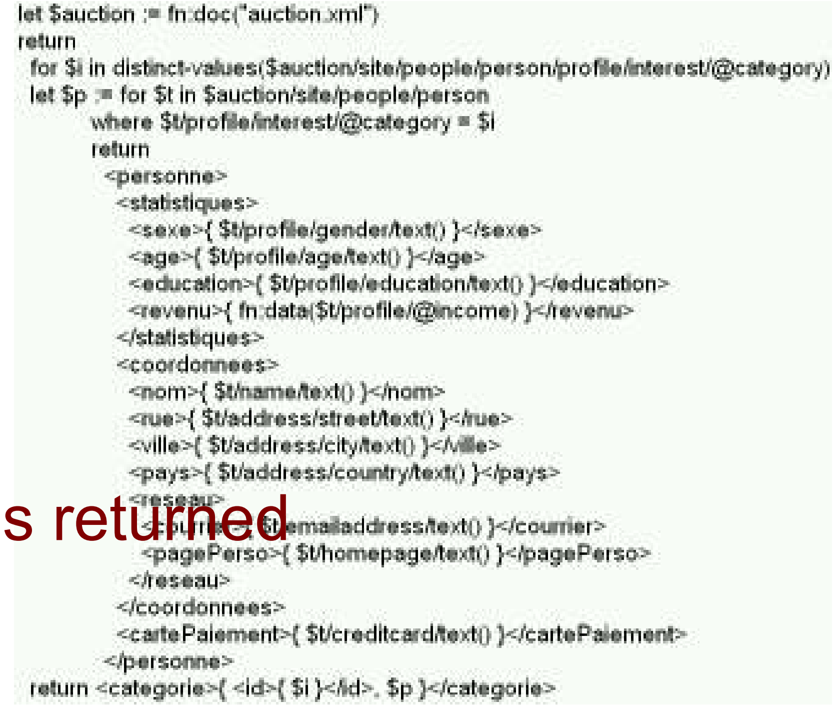


Physical address

Address conversion table

Buffer management (XMark Q10 as an example)



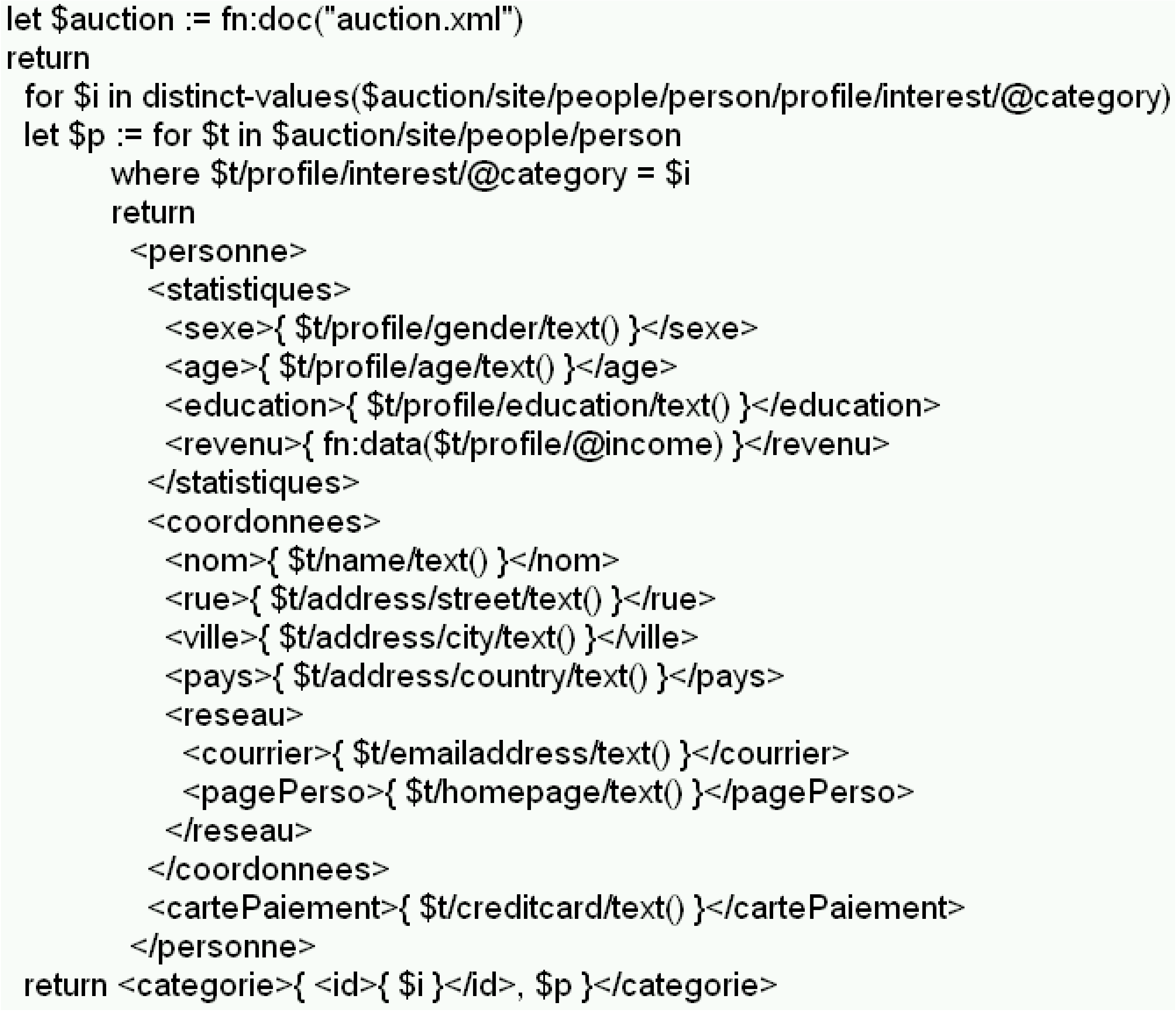


```
let $auction := fn:doc("auction.xml")
return
  for $i in distinct-values($auction/site/people/person/profile/interest/@category)
  let $p := for $t in $auction/site/people/person
          where $t/profile/interest/@category = $i
          return
            <personne>
              <statistiques>
                <sexe>{ $t/profile/gender/text() }</sexe>
                <age>{ $t/profile/age/text() }</age>
                <education>{ $t/profile/education/text() }</education>
                <revenu>{ fn:data($t/profile/@income) }</revenu>
              </statistiques>
              <coordonnees>
                <nom>{ $t/name/text() }</nom>
                <rue>{ $t/address/street/text() }</rue>
                <ville>{ $t/address/city/text() }</ville>
                <pays>{ $t/address/country/text() }</pays>
                <reseau>
                  <courrier>{ $t/emailaddress/text() }</courrier>
                  <pagePerso>{ $t/homepage/text() }</pagePerso>
                </reseau>
              </coordonnees>
              <cartePaiement>{ $t/creditcard/text() }</cartePaiement>
            </personne>
  return <categorie>{ <id>{ $i }</id>, $p }</categorie>
```

20% of original document is returned as the result

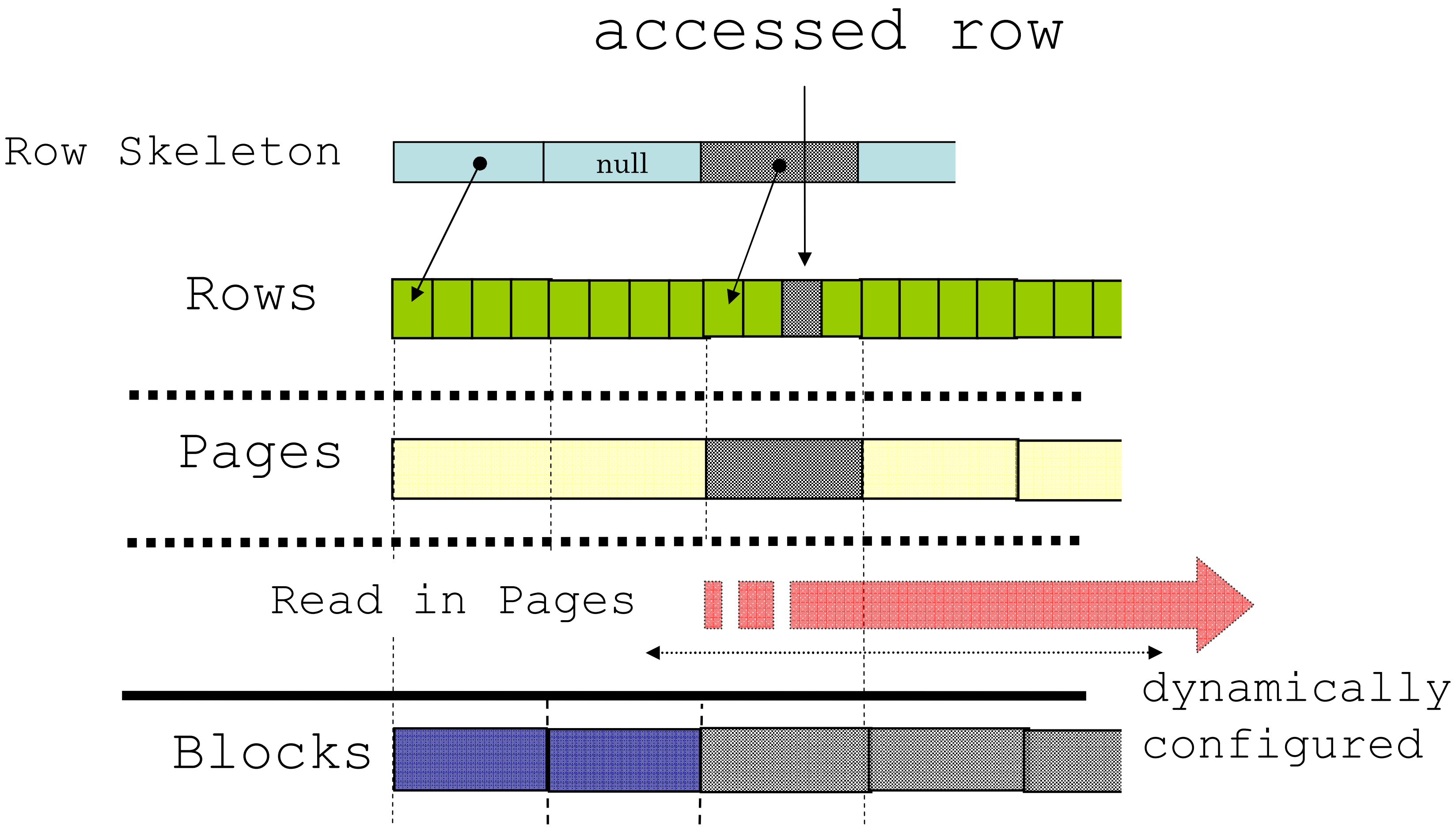
	Elapsed time (msec)	total read blocks	buffer replacement
LRU	211.83	1,919,586	567,702
2Q	185.56	80,673	0

14.2% speedups



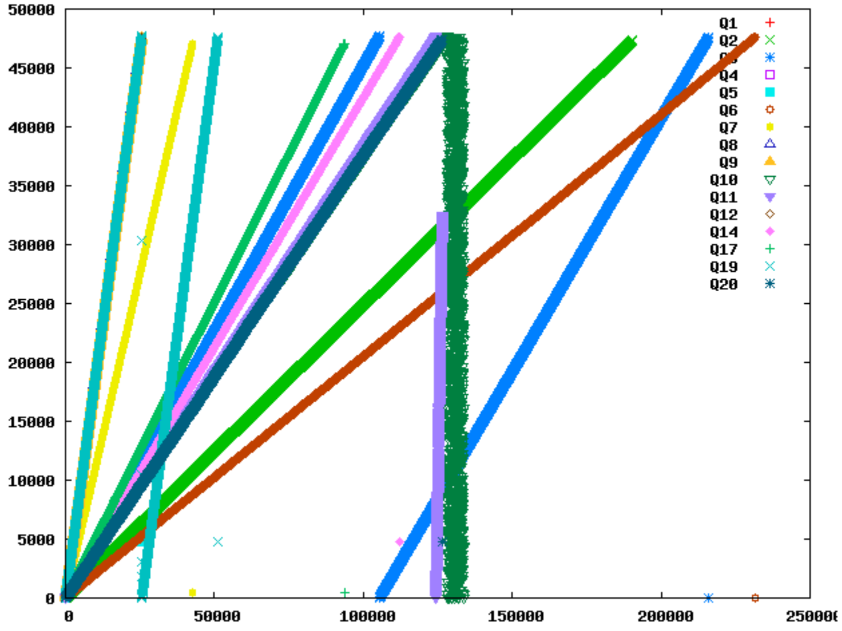
```
let $auction := fn:doc("auction.xml")
return
  for $i in distinct-values($auction/site/people/person/profile/interest/@category)
  let $p := for $t in $auction/site/people/person
            where $t/profile/interest/@category = $i
            return
              <personne>
                <statistiques>
                  <sexe>{ $t/profile/gender/text() }</sexe>
                  <age>{ $t/profile/age/text() }</age>
                  <education>{ $t/profile/education/text() }</education>
                  <revenu>{ fn:data($t/profile/@income) }</revenu>
                </statistiques>
                <coordonnees>
                  <nom>{ $t/name/text() }</nom>
                  <rue>{ $t/address/street/text() }</rue>
                  <ville>{ $t/address/city/text() }</ville>
                  <pays>{ $t/address/country/text() }</pays>
                  <reseau>
                    <courrier>{ $t/emailaddress/text() }</courrier>
                    <pagePerso>{ $t/homepage/text() }</pagePerso>
                  </reseau>
                </coordonnees>
                <cartePaiement>{ $t/creditcard/text() }</cartePaiement>
              </personne>
  return <categorie>{ <id>{ $i }</id>, $p }</categorie>
```

Pyshical layout

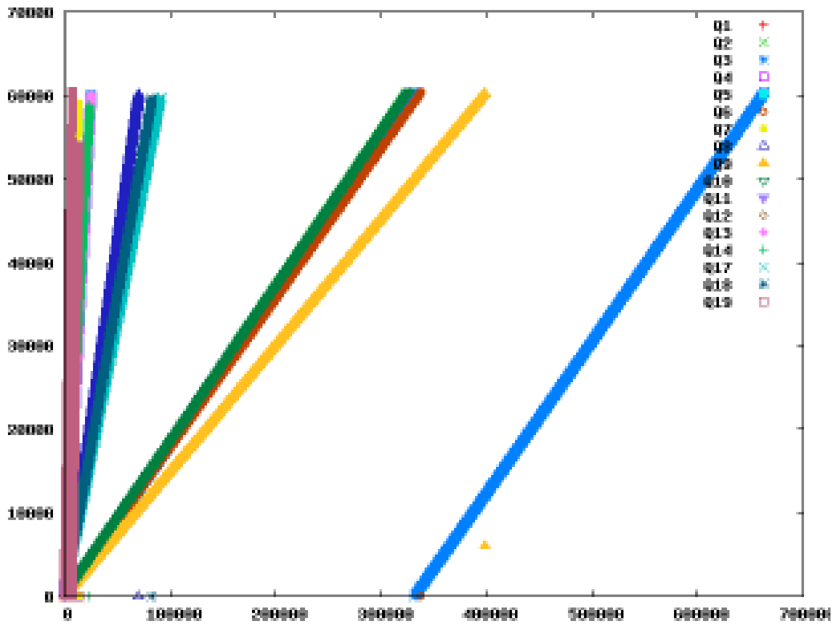


Access pattern analysis of XBench queries

DC/SD Normal



TC/SD Normal



Memory Mapped DTM

```
#include <sys/mman.h>

void *mmap(void *start, size_t length, int prot, int flags,
        int fd, off_t offset);

int munmap(void *start, sizt_t length);
```

We present a memory mapped scheme extending the DTM model, it has boost the performance significantly.