Purifying XML Structures

Ph.D. Defense

Taro L. Saito

University of Tokyo

<leo@cb.k.u-tokyo.ac.jp>

An Innovation of the XML DBMS

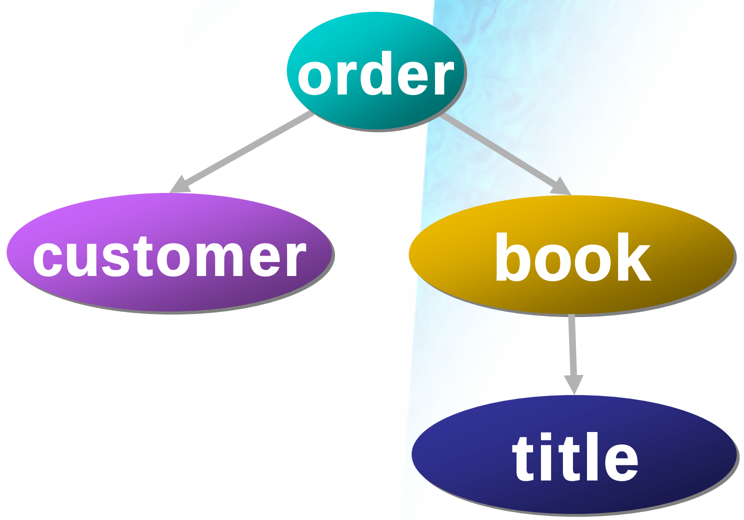
Outline

- Introduction
  - XML and Structural Fluctuation
  - Amoeba Join
- Purifying XML Structures
  - Functional Dependencies for XML
  - Amoeba Join Decomposition
  - Ubiquitous Keys
- Implementation
  - Amoeba Join Processing Algorithms
  - XML Indexing
  - Experimental Results
- Conclusions
  - Applications
  - Summary of Contributions & Future Work

Introduction

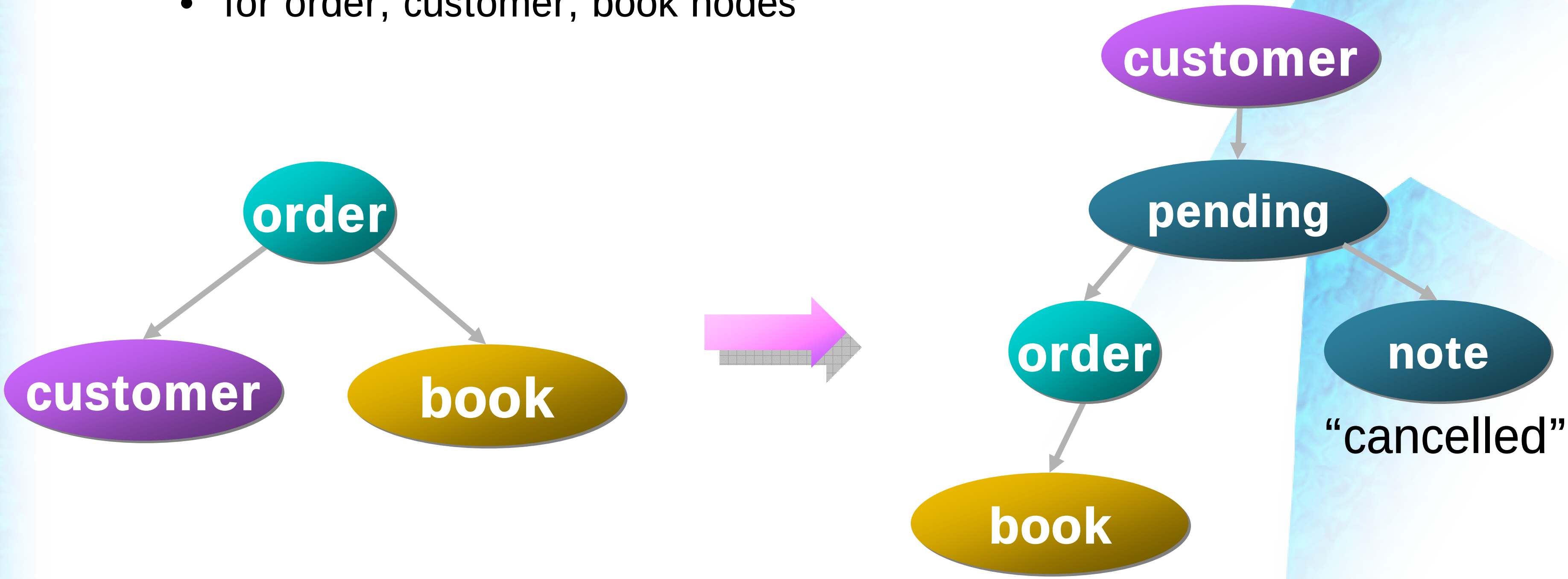
- XML (Extensible Markup Language)
  - A markup language representing a tree structure
  - Since 1996, XML has been broadly used as a data representation format
- Major drawbacks
  - Hierarchical representation of data is too complex
    - for both of human and computer programs
    - reminiscences of 1970s' discussion
      - Relational v. s. Hierarchical DB
  - There exist many alternative tree structures
    - to represent a same data model

```
<bookstore>
  <order>
     <customer>John</customer>
     <book>
        <title>Data on the Web</title>
     </book>
  </order>
</bookstore>
```



Structural Fluctuation

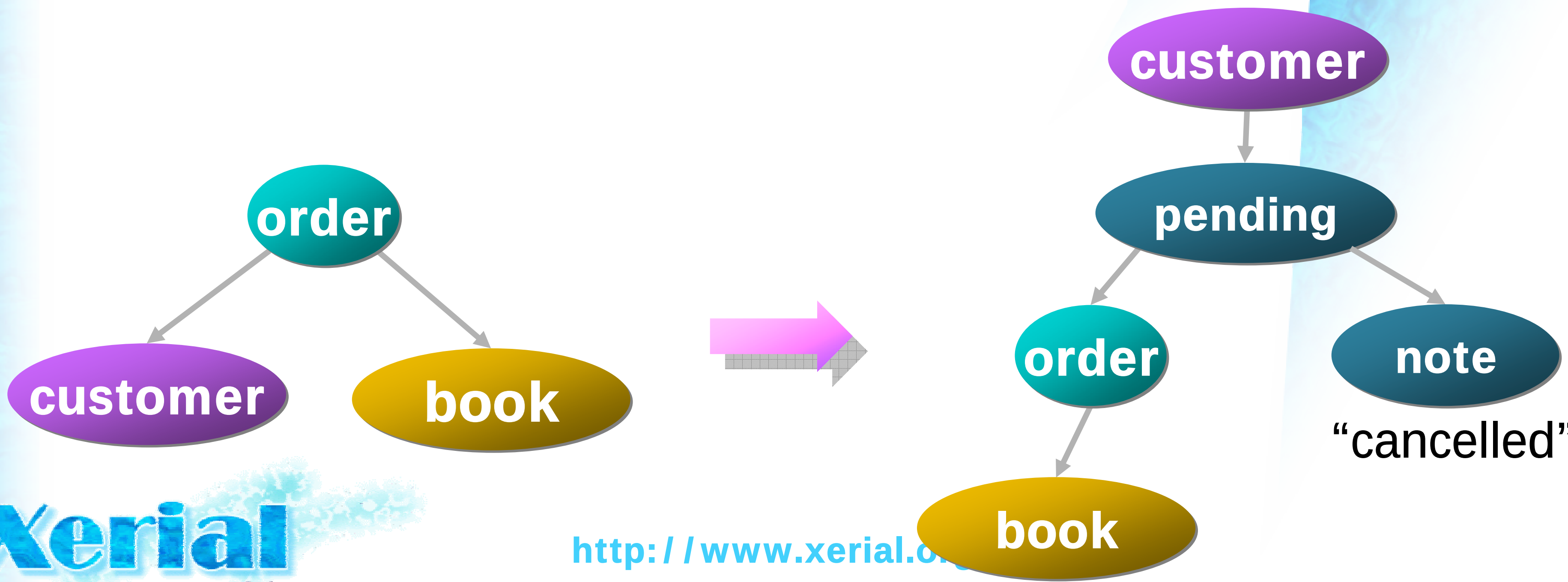
- Differently Structured XML Documents
  - representing a same data model   e.g. Amazon.com
    - for order, customer, book nodes



  - The hierarchical order of *order* and *customer* is reversed.
  - The order node is behind the pending node.

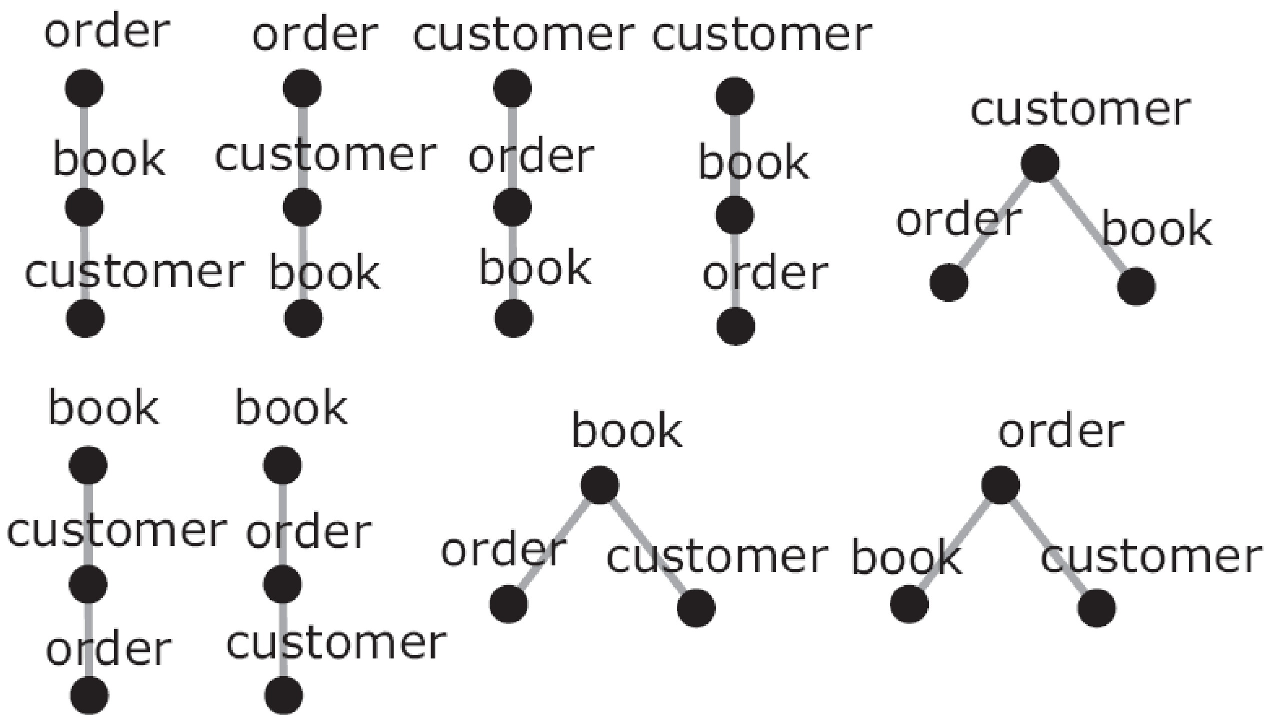
Querying Structural Fluctuation

- Standards of XML Processing: XPath, SAX, DOM, etc.
- Many parse states:
  - If we find an order, then parse customer and book
  - or if we first find an customer, then parse pending/order and book …
  - Such query processing is tedious and error-prone!
- Why we need different programs to parse the same meaning XML data?



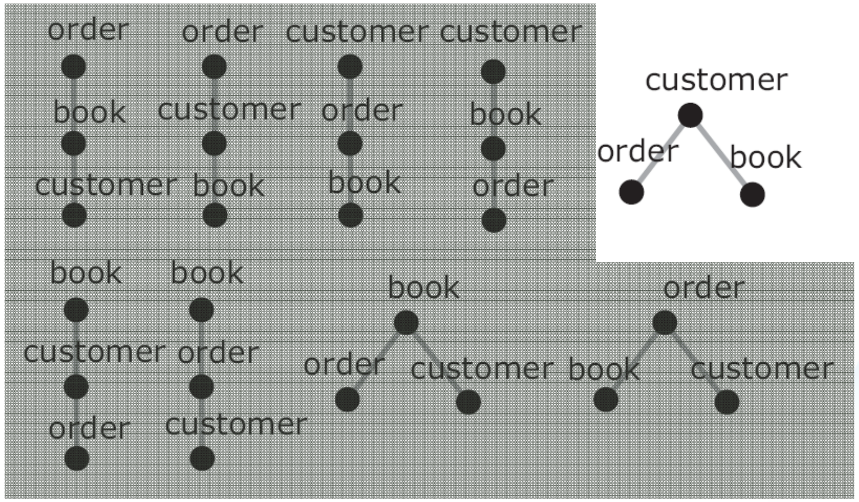
Structural Fluctuations

- In general, the number of structural fluctuations of *n* nodes is $n^{(n-1)}$
  - Enumeration of labeled trees of *n* nodes



Current Solution

- Disallow structural fluctuations by using a schema
  - XML Schema, DTD, RelaxNG, etc.

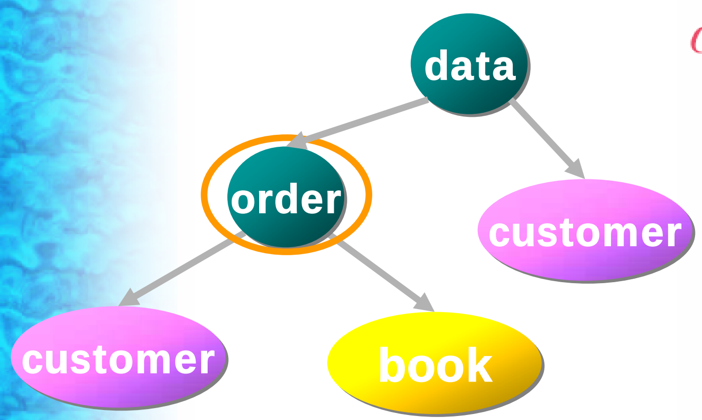


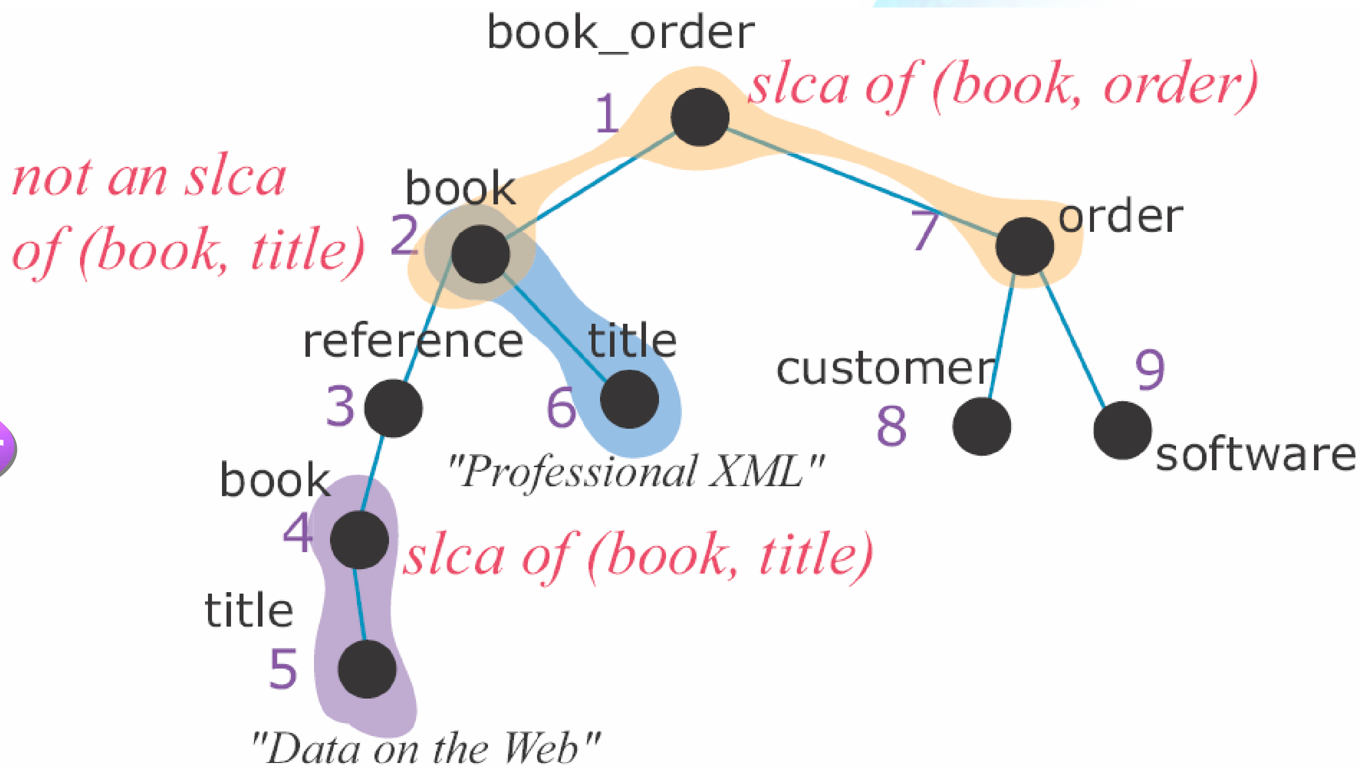
- However, fixing a tree structure involves irrelevant work in defining a data model.
  - Why we have to choose only one tree structures?

Heuristic Approach

- SLCA (Smallest Lowest Common Ancestor)
  - [Li, VLDB2004], [Xu, SIGMOD2005]
  - An *lca* node that does not contain other *lca* nodes.
  - However, it easily leads to unintended results

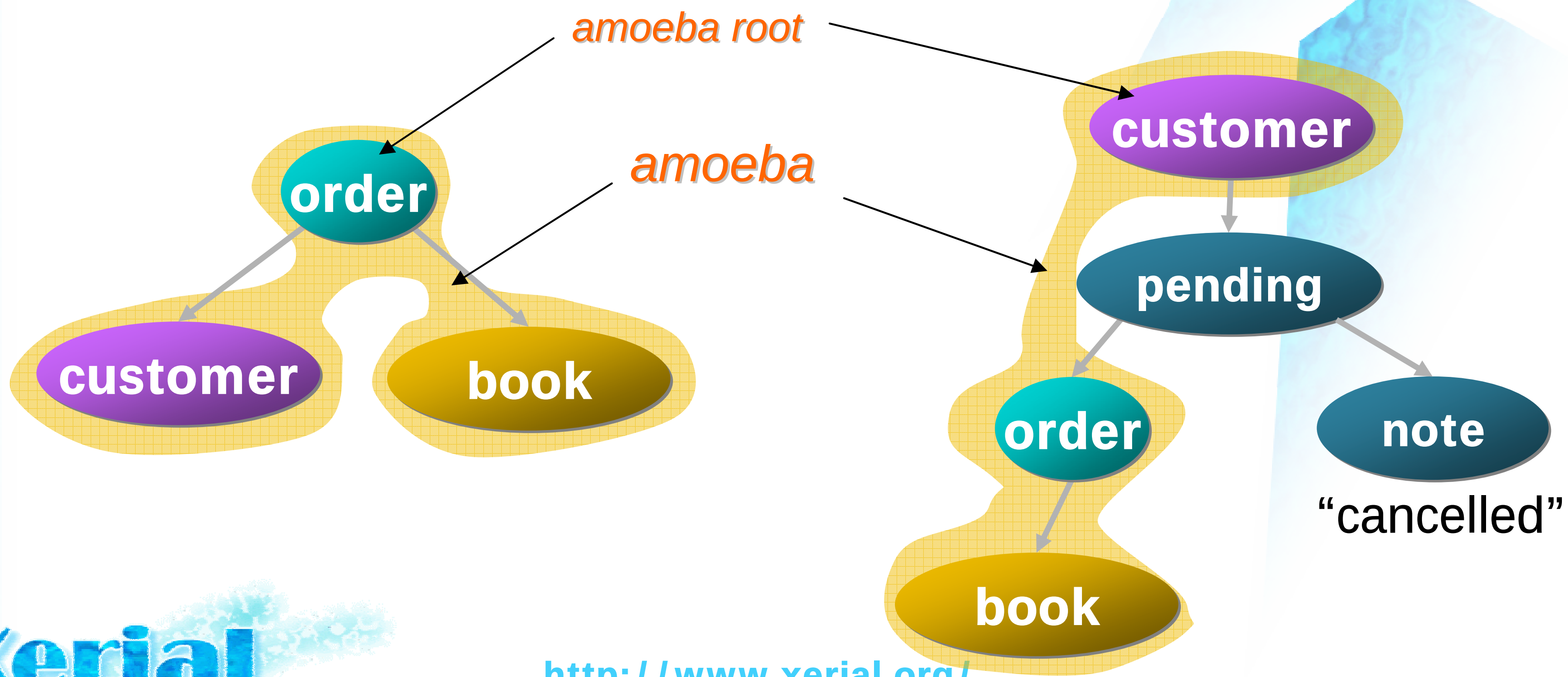
slca of
(customer, book)



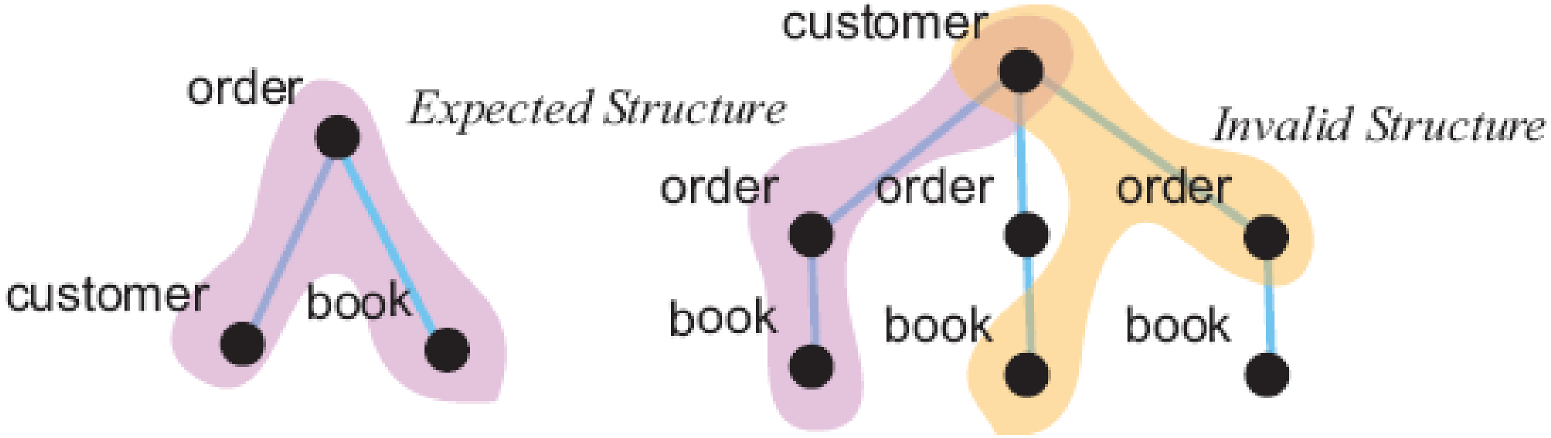


Amoeba Join

- Amoeba Join: AJ(order, customer, book)
  - [WebDB 2006]
  - retrieves node tuple such that
    - one of (order, customer, book) nodes is a common ancestor of the others.
  - Handles every structural fluctuation



Semantics of XML Structures



- Semantics implied in XML data
  - Each order node should have a single book node
    - Invalid structure might be retrieved without considering such semantics of data.
  - Instances of such invalid structures could be numerous
- To represent semantics of XML data, we introduce functional dependencies for XML

Functional Dependency (FD)

- Functional Dependency
  - $X \rightarrow Y$: if two tuples *p, q* agree with *X*, then also agree with *Y*

order	book	title
1	b1	Database Systems
2	b1	Database Systems
3	b2	Data on the Web

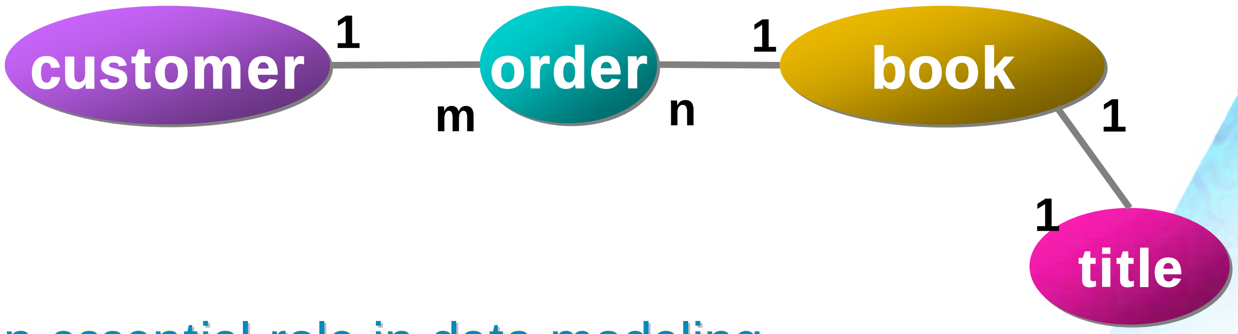
- FDs: order $\rightarrow$ book, book $\rightarrow$ title

order	book
1	b1
2	b1
3	b2

book	title
b1	Database Systems
b2	Data on the Web

- FD is generally used to avoid redundancies of data
  - Normal Form

Data Modeling & FD



- FD has an essential role in data modeling
  - describe one-to-one, one-to-many, many-to-many relationships
    - ex. ER (Entity-Relationship) diagram, UML (Unified Modeling Language)
- Example
  - order → book, order → customer
    - An order has a book. An order has a customer.
    - A book has many orders. A customer has many orders (one-to-many)
  - book → title, title → book
    - A book has a title; a title belongs to a book (one-to-one)
  - customer, book → order
    - An order connects many customers and books (many-to-many)

Functional Dependencies for XML

- Previous Work of FDs for XML
  - [Buneman et al., WWW2001], [Arenas and Libkin, TODS2004]
  - based on fixed paths
    - Because there was no counter part of relation (tables) in XML
  - e.g. /order → /order/book
    - Structural fluctuations are not allowed:

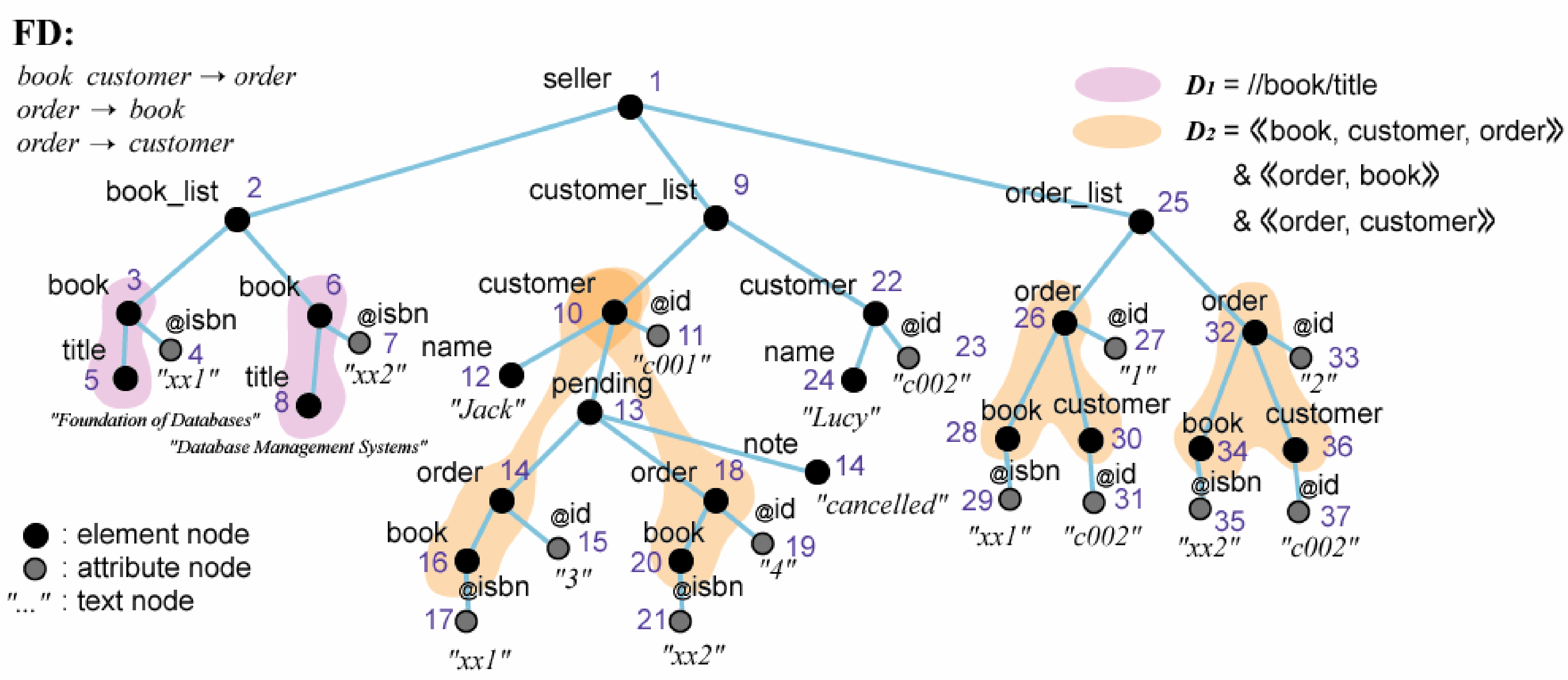
```
<order id="1">
  <book isbn="xx1"/>
  <customer id="c001"/>
</order>
```

⇔

```
<book isbn="xx1">
  <order id="1">
     <customer id="c001"/>
  </order>
</book>
```

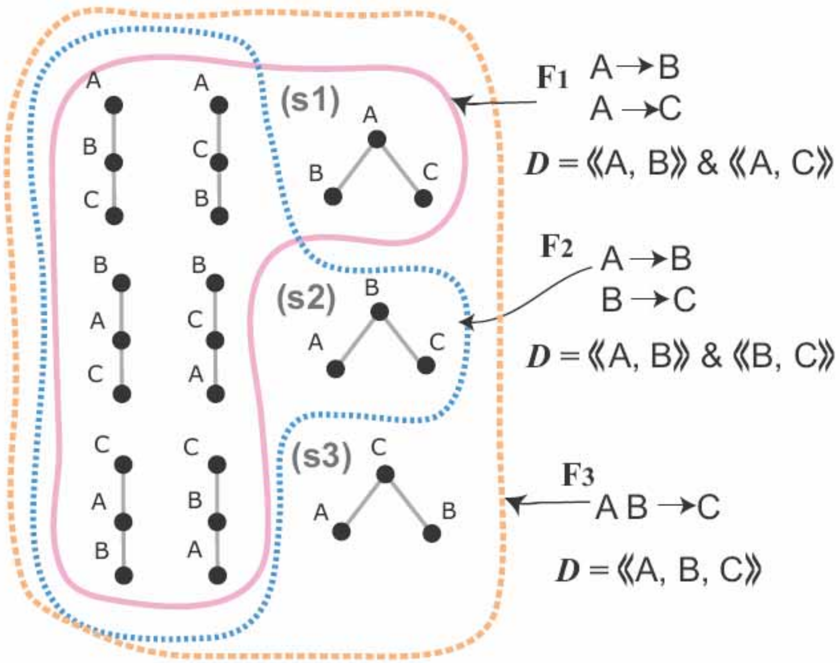
  - In reality, however, the constraint on the path, a book must be a child of an order, is too strong.
  - Their definition has no loss-less decomposition

Relation in XML



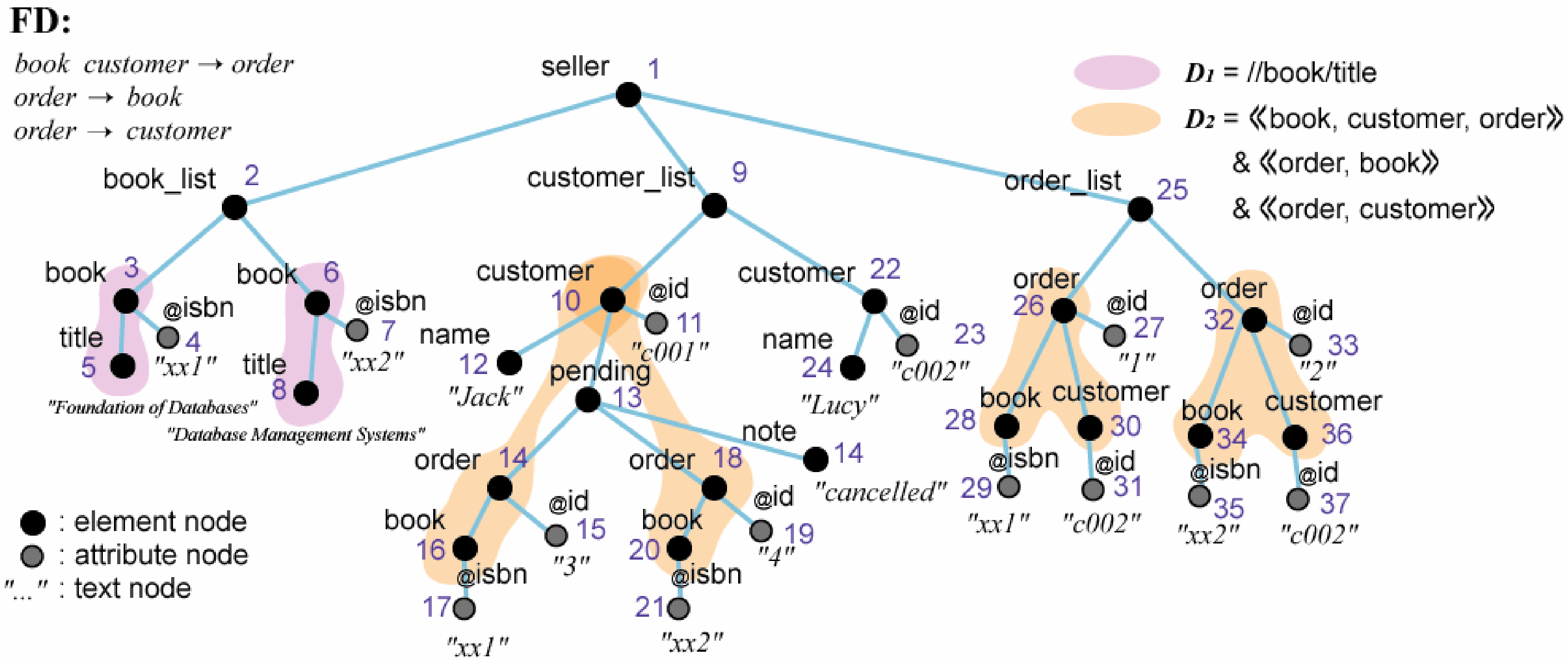
- Relation in XML allows a zigzag shape
- For an FD: *book, customer → order*
  - (book, customer, order) must be an amoeba

A set of FDs defines XML structures



- Traditional Approach:
  - XML data (Structured Data) -> Data Model
- Our approach: Data Model (FD) -> XML Structures
  - Allows various XML structures to describe a data model
  - Enhancing expressive power of XML databases

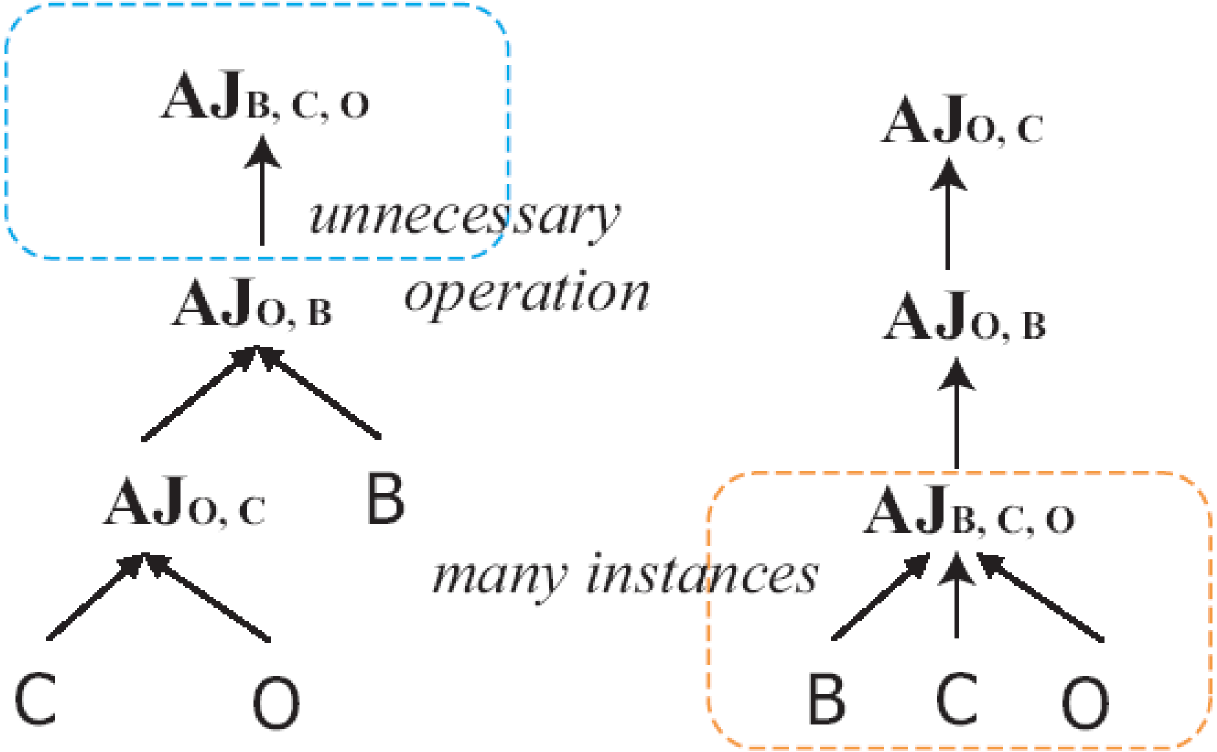
Amoeba Join Satisfying FDs



- AJ_F(order, book, customer)
  - retrieves a relation in XML satisfying a set F of FDs
- Makes easier managing multiple hierarchies of XML tree structures
  - An amoeba join AJ_F(order, book, customer) can track D_2

Amoeba Join Decomposition

$f1: B, C \rightarrow O$
$f2: O \rightarrow B$
$f3: O \rightarrow C$

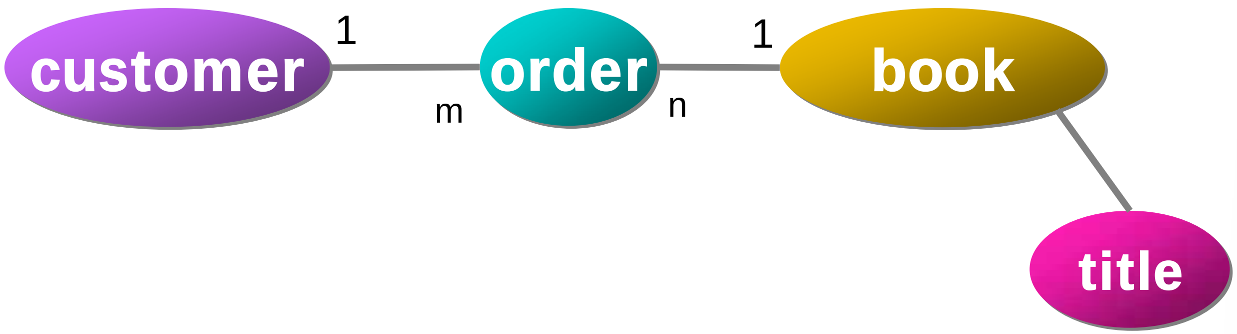


$$
\begin{aligned}
& AJ_F(B, C, O) \\
&= AJ_{B,C,O}(AJ_{F1}(B, C, O)) \\
&= AJ_{B,C,O}(AJ_{O,B}(AJ_{F2}(C, O), B)) \\
&= AJ_{B,C,O}(AJ_{O,B}(AJ_{O,C}(C, O), B))
\end{aligned}
$$

$$
\begin{aligned}
& AJ_F(B, C, O) \\
&= AJ_{O,C}(AJ_{F1}(B, C, O)) \\
&= AJ_{O,C}(AJ_{O,B}(AJ_{F2}(B, C, O))) \\
&= AJ_{O,C}(AJ_{O,B}(AJ_{B,C,O}(B, C, O)))
\end{aligned}
$$

FD Based XML Query Processing

- No explicit path structures are required
- Examples:
  - FDs
    - book, customer → order • order → book • order → customer
  - A query for book and order node: AJ_F(book, order)
    - book and order nodes compose amoebas
  - A query for book and customer nodes:
    - AJ_F(book, customer)
      - book and customer nodes might be connected through order nodes
    - Thus, AJ_F(book, customer, order) is evaluated
- Relation in XML is dynamically determined according to query targets



Functional Dependencies and Keys

- Key is a special case of a functional dependency
  - e.g. order (id) → book, customer
    - order (id) is a key

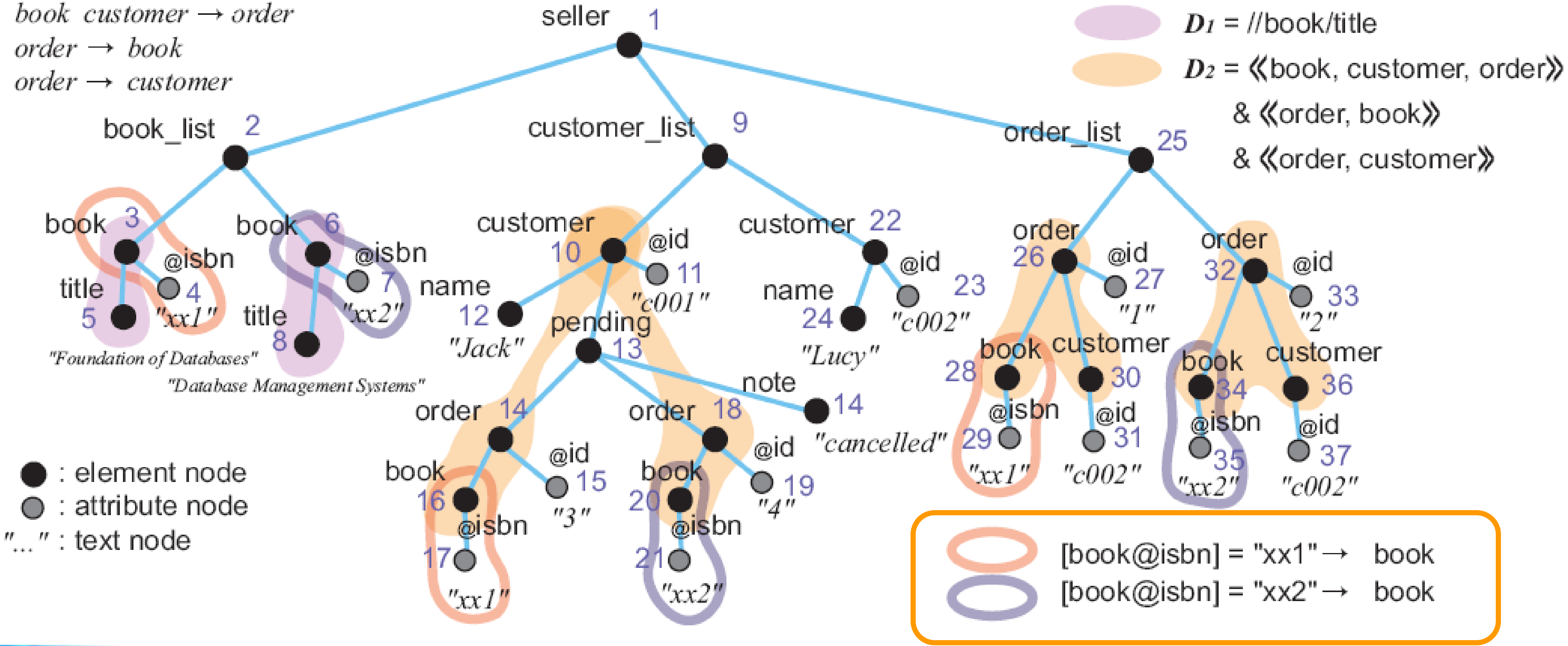
order	book	customer
1	b1	John
2	b2	Lucy

- Using a relation in XML, we can define keys for XML
  - [order@id] → book, customer
    - Given an order id, we can uniquely determine book and title nodes
    - XML structures: <<order, book>>, <<order, customer>>
- More general description of keys
  - In [Buneman, et al. WWW2001], it is not allowed to reverse the position of order and book nodes

Ubiquitous Keys

FD:

book customer → order
order → book
order → customer

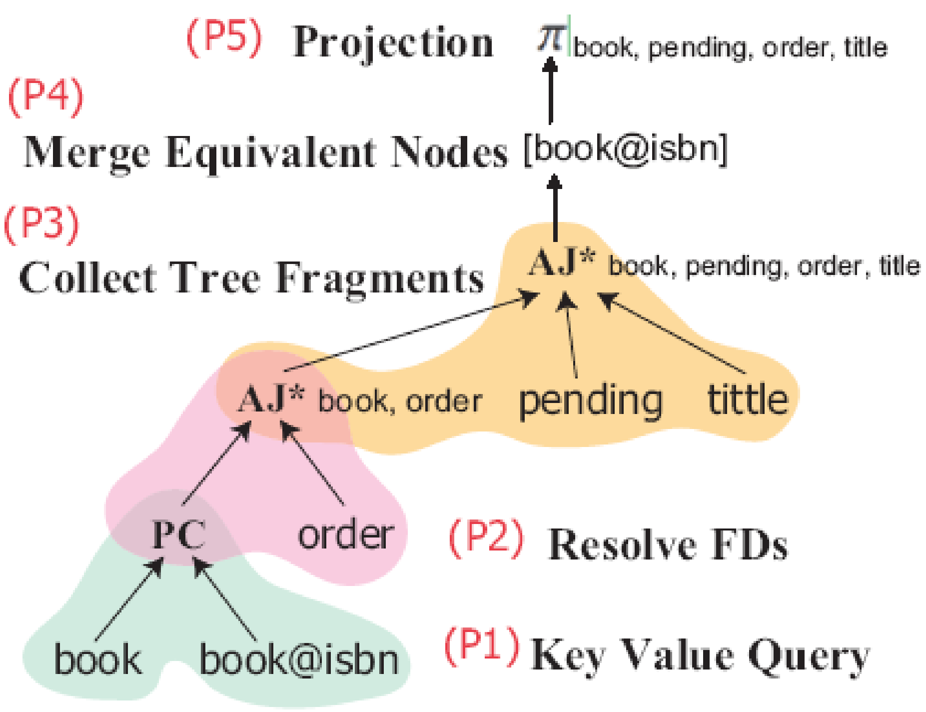



```
(Q1):  for $x in /seller/book_list/book
           $y in /seller/customer_list/customer/pending/order/book
       where $x/@isbn = $y/@isbn
       return $x/title

(A1):  AJ(book, [pending, order, title])
```

Querying without using Structures

- AJ(book, [pending, order, title])
  - book nodes are merged using ubiquitous keys



key domains

book	book@isbn	[book@isbn]	pending	order	title
3	4	"xx1"			5
6	7	"xx2"			8
16	17	"xx1"	13	14	
20	21	"xx2"	13	18	
28	29	"xx1"			
34	35	"xx2"			

key value

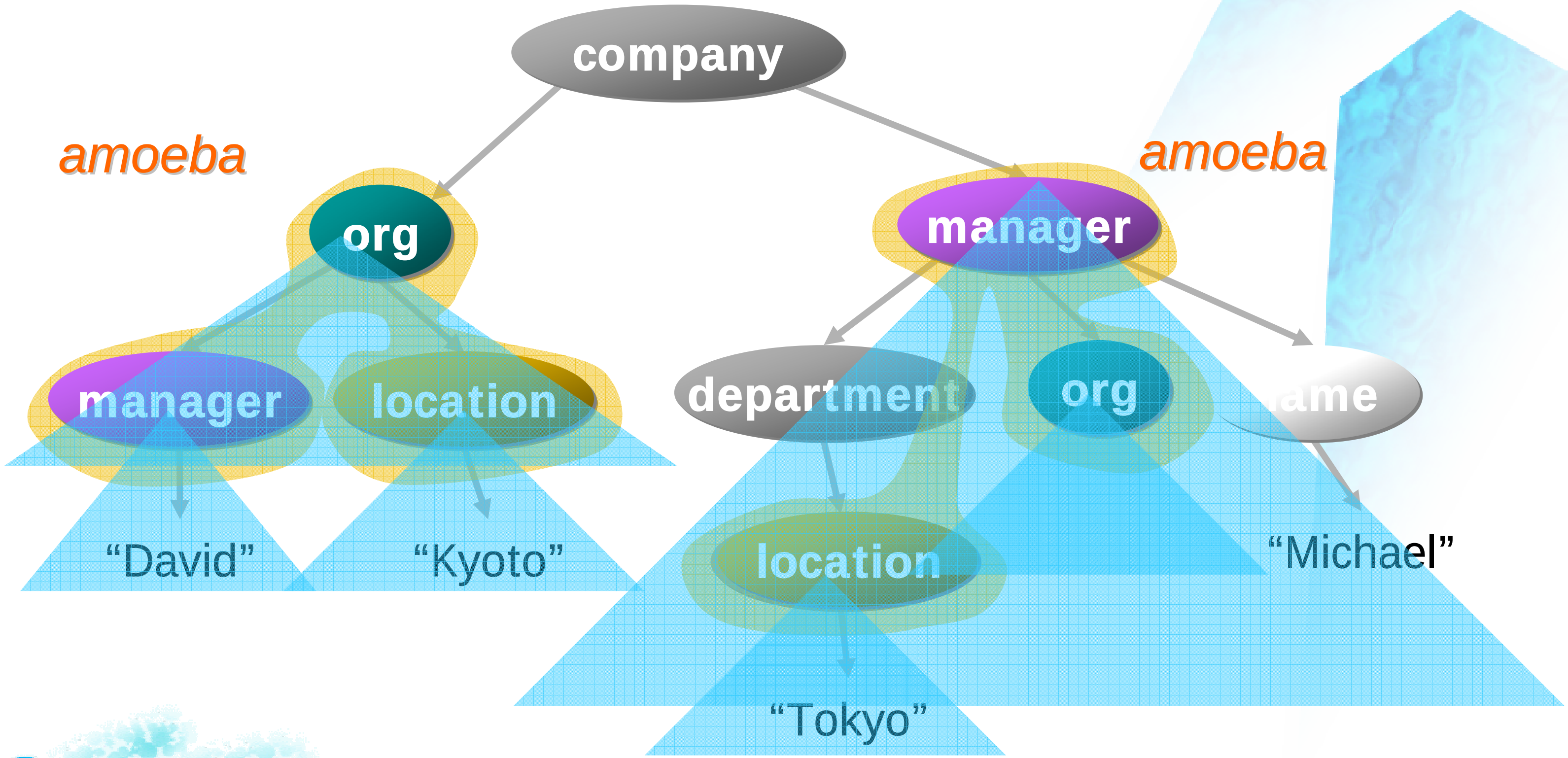
merge & projection

book	[book@isbn]	pending	order	title
{3, 16, 28}	"xx1"	13	14	5
{6, 20, 34}	"xx2"	13	18	8

Amoeba Join Processing

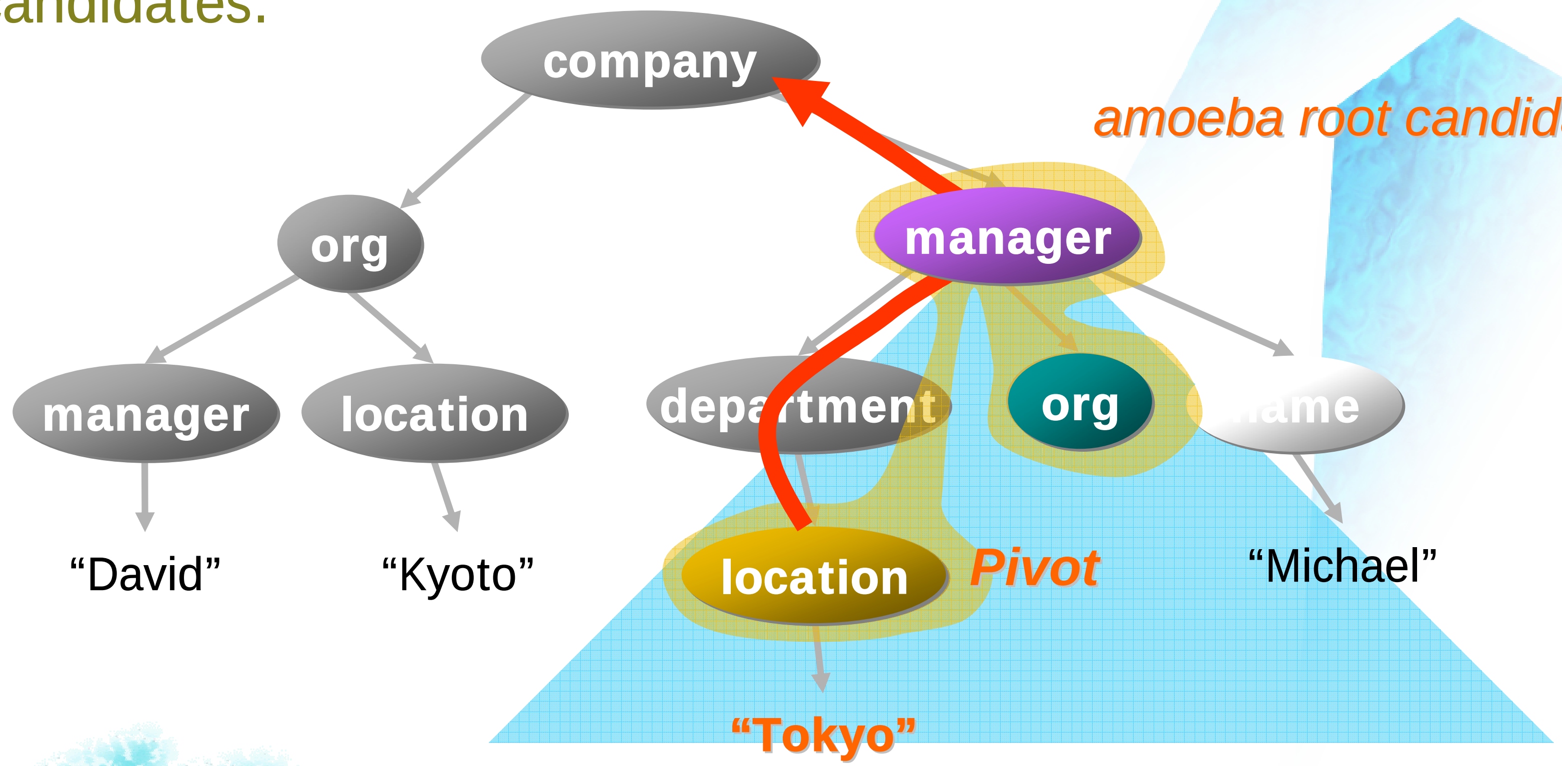
Sweep Amoeba Join Algorithm

- Fetch all input nodes
  - AJ(org, manager, location)
  - Sort input nodes in their document orders.
- Sweep sorted input nodes
  - Assume the smallest node in the input nodes as an amoeba root.
  - Search their descendant regions for components of amoebas



Disk I/O Optimization

- AJ(org, manager, location = “Tokyo”)
  - Choose pivot nodes from a small input domain
  - Traverse upward to find amoeba root candidates
  - Search space for amoeba is localized under the amoeba root candidates.

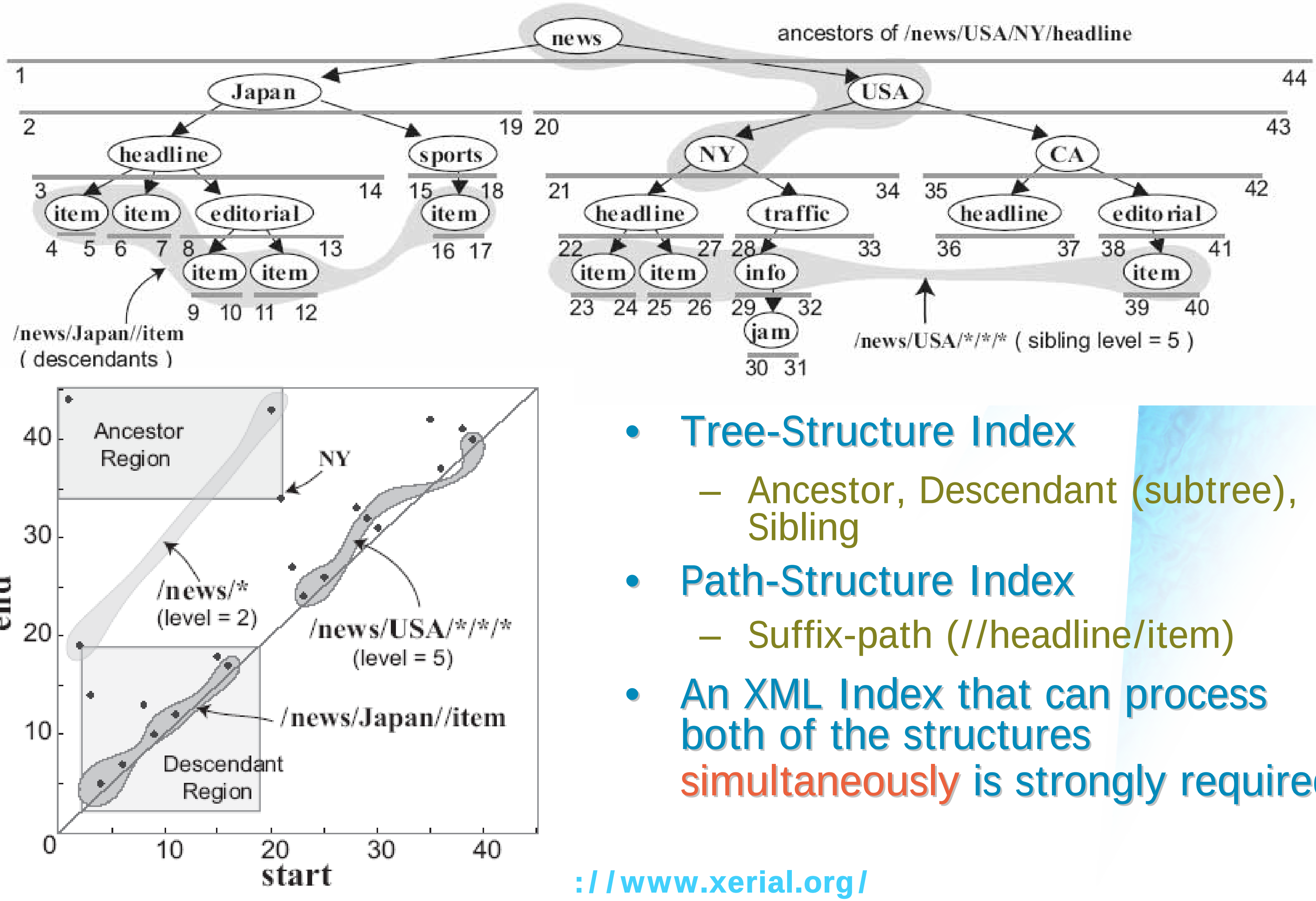


XML Indexing

History of XML Indexing

- A hundreds of XML indexing papers ....
  - tailored to specific queries
    - XPath query, structural-join (A//D), twig-queries, text search, etc.
  - from many research areas
    - Database Community
      - DataGuides (1997), 1-index (1999), XR-tree (2002), PathFinder (2006)
      - Node labeling (static or updatable)
        - » Dewey order, ORDPATH(2004), BLAS(2004), Pbi (2005)
    - Information Retrieval (IR)
      - inverted indexes for text data. SLCA (2005)
    - Compressed Index
      - XBW (Ferrangina, WWW2006)

Multidimensional Aspects of XML



- Tree-Structure Index
  - Ancestor, Descendant (subtree), Sibling
- Path-Structure Index
  - Suffix-path (//headline/item)
- An XML Index that can process both of the structures simultaneously is strongly required

Our Approach

- [DASFAA2007]
- Integrating tree-structure and path indexes
  - As a multidimensional index
    - (start, end, level, path)
  - It can be implemented on top of the B+-tree
- Why B+-trees?
  - Index structures and transaction management, recovery, logging, caching etc. are interdependent.
  - We already have many transaction management techniques on B+-trees
    - Transaction management on R-tree is not seriously supported.

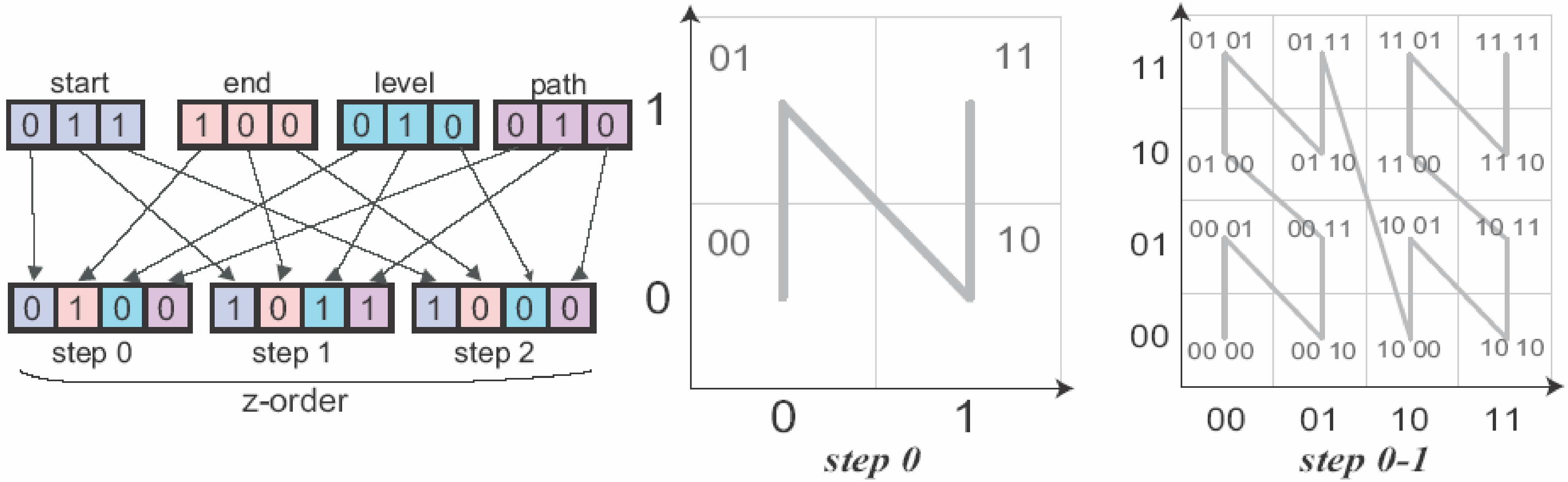
Inverted-Path Index

path ID	inverted path
1	news\
2	Japan\news\
3	headline\Japan\news\
4	headline\NY\USA\news\
5	headline\CA\USA\news\
6	item\headline\Japan\news\
7	item\headline\NY\USA\news\
8	item\editorial\Japan\news\
9	item\editorial\CA\USA\news\
10	item\sports\Japan\news\

path ID	inverted path
11	editorial\Japan\news\
12	editorial\CA\USA\news\
13	sports\Japan\news\
14	USA\news\
15	NY\USA\news\
16	traffic\NY\USA\news\
17	info\traffic\NY\USA\news\
18	jam\info\traffic\NY\USA\news\
19	CA\USA\news\

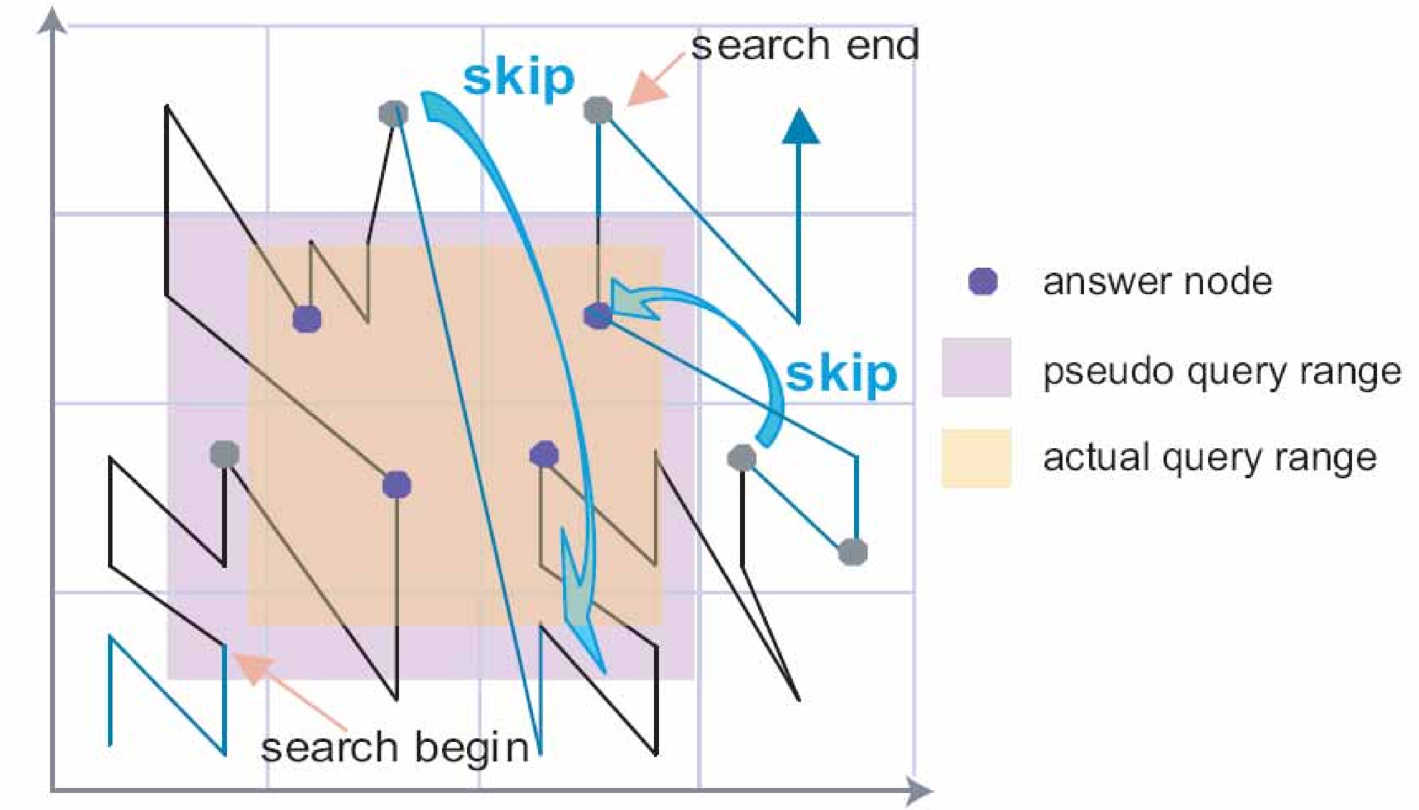
- Align inverted paths in the lexicographical order
  - facilitates suffix path queries
- Examples (suffix-path query range):
  - //item [6, 11)
  - //headline/item [6, 8)

Z-Order



- Align multidimensional points (nodes) in z-order
  - Interleave function gives z-order in the multidimensional space
- Each step in z-orders splits slices into two

Range Query



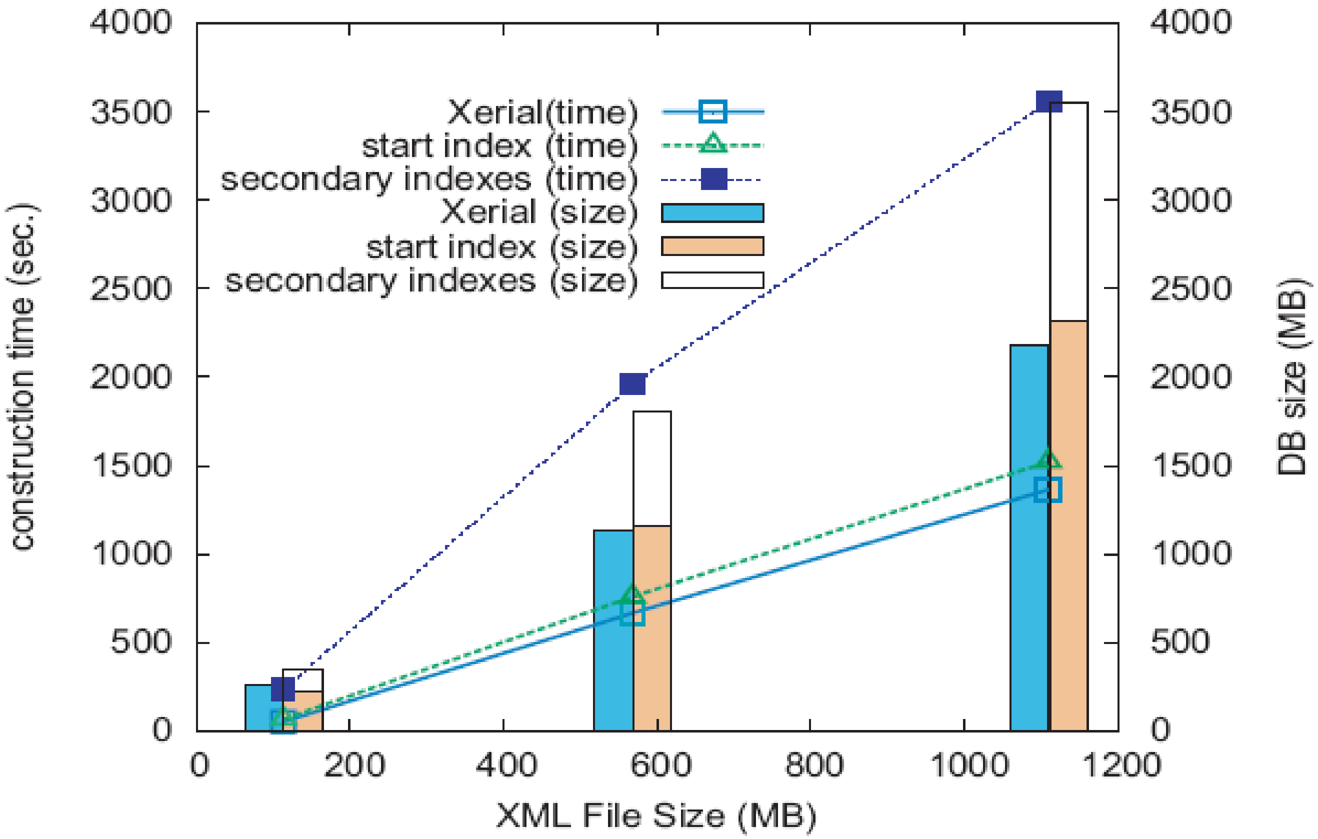
- Traverse B+-tree in the order of z-order

Experimental Results

Implementation

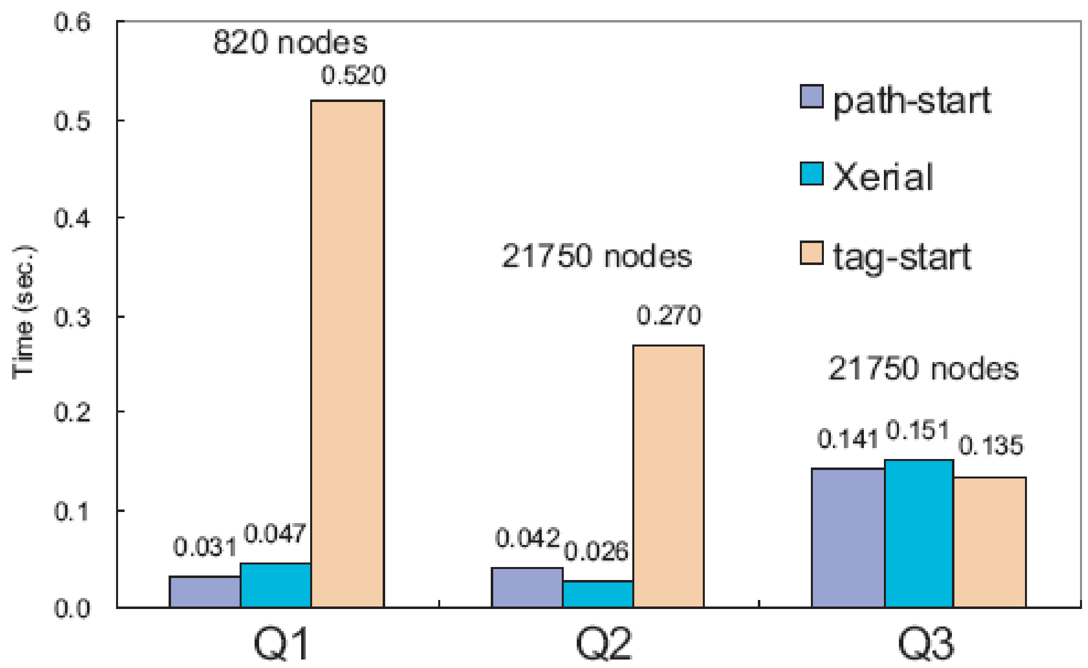
- Xerial
  - http://www.xerial.org/
  - XML Database Management System
    - XML data is multi-dimensionally indexed
    - supporting amoeba joins & XPath queries
  - Implemented in C++
    - about 150,000 lines of codes
      - Query compiler & scheduler, query processing algorithms
      - Database indexing, XML processor, etc.
- Machine environment for experiments
  - Windows XP notebook
  - Pentium M 2GHz, 1GB Main Memory
  - 5,400 rpm HDD (100GB)

Database Size



- Data set: XMark Benchmark XML Document
- Xerial is space-efficient

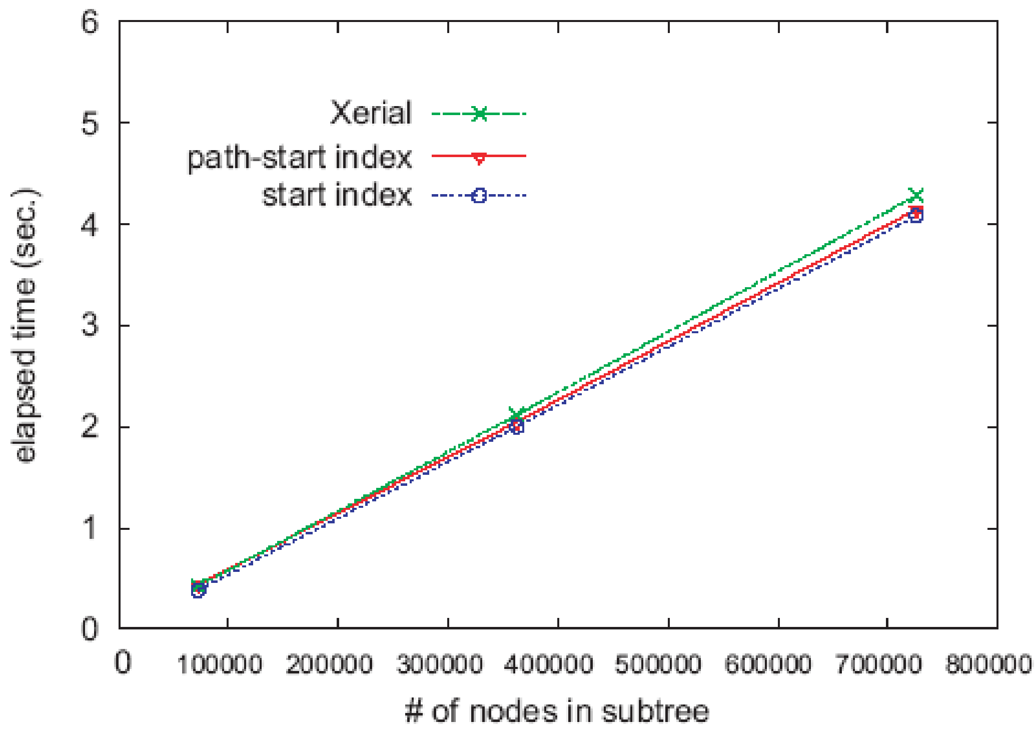
Suffix-Path Query Performance



Q1: //category/description/parlist/listitem Q3: //item
Q2: /site/regions/asia/item/description

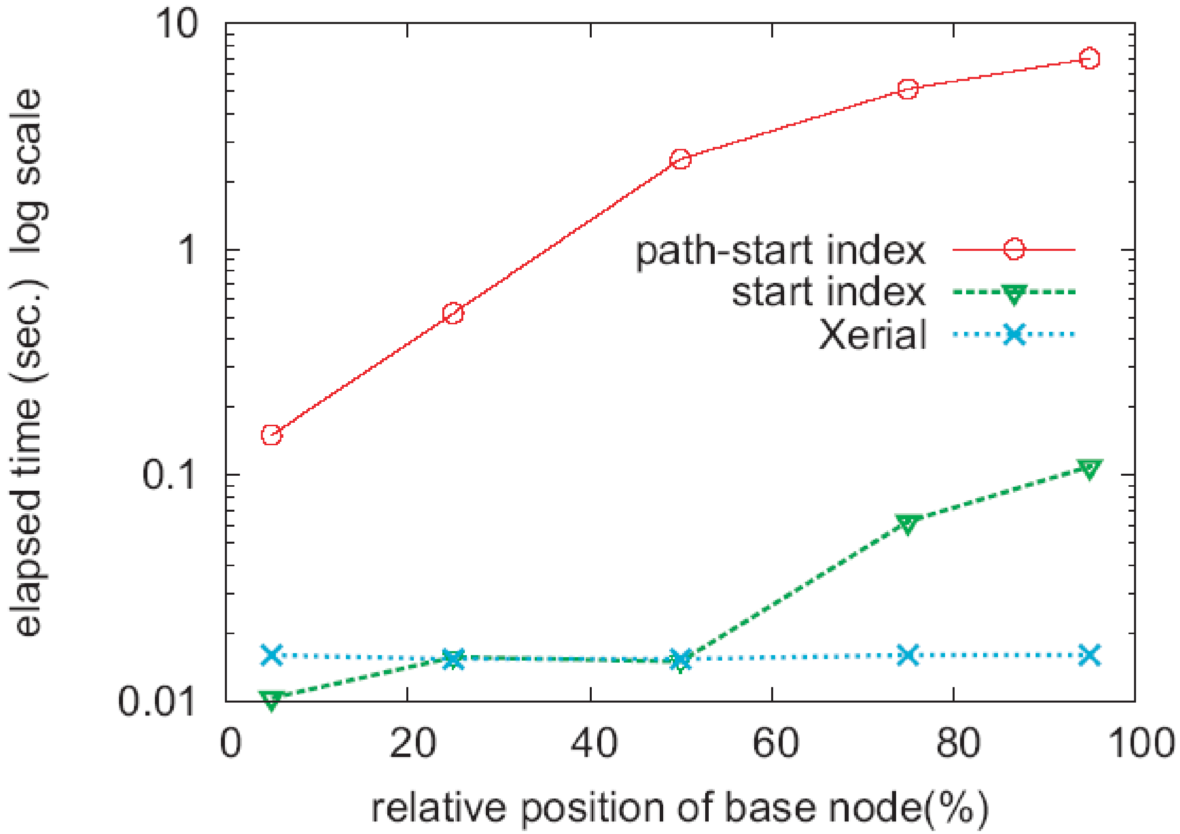
- Xerial & path-start index is fastest

Subtree Retrieval Performance



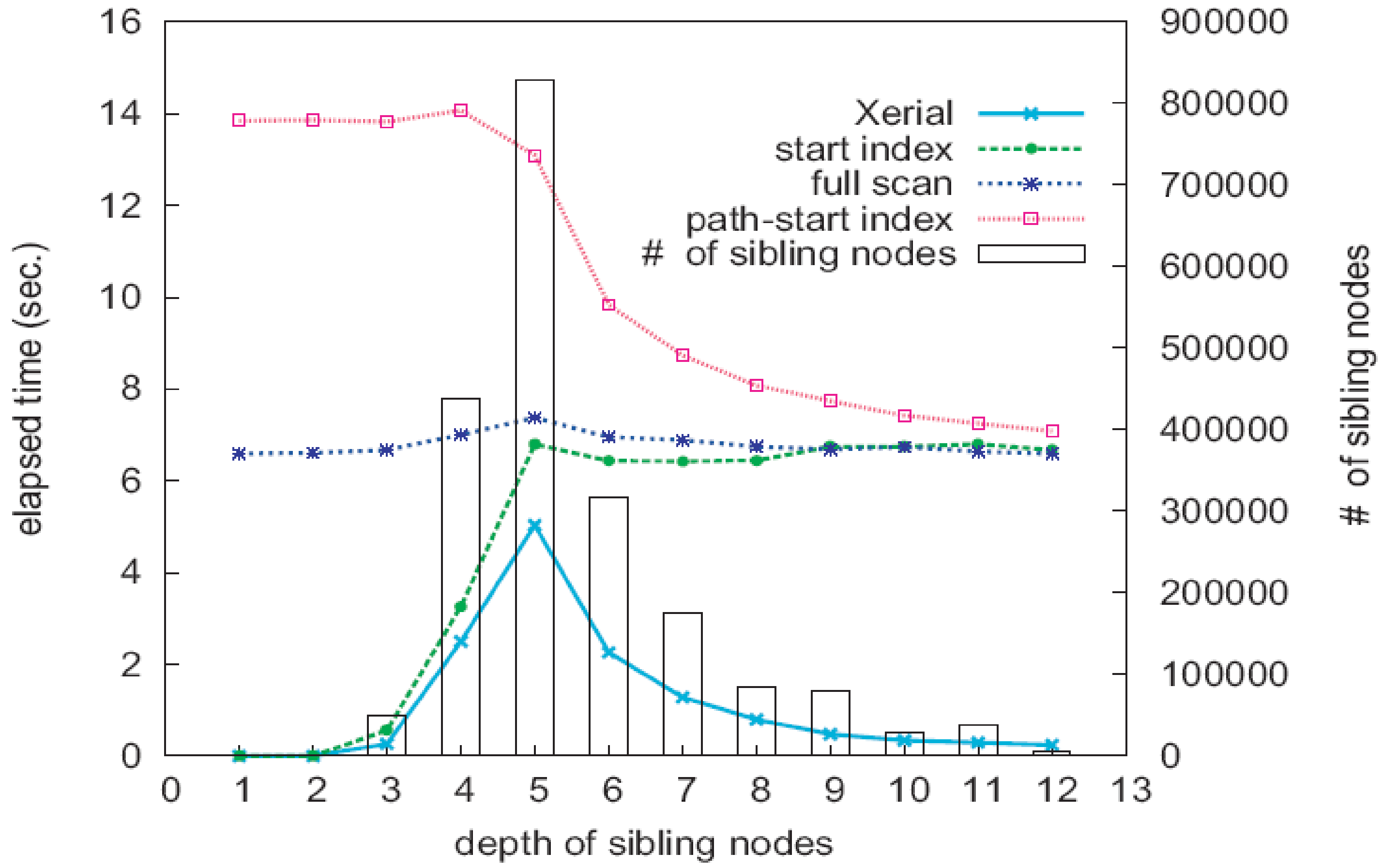
- All of the indexes shows similar performance
  - XML nodes are sorted in the order of start values

Ancestor Retrieval



- The number of the previous nodes of a context node affects the ancestor-query performance.

Sibling Query Performance



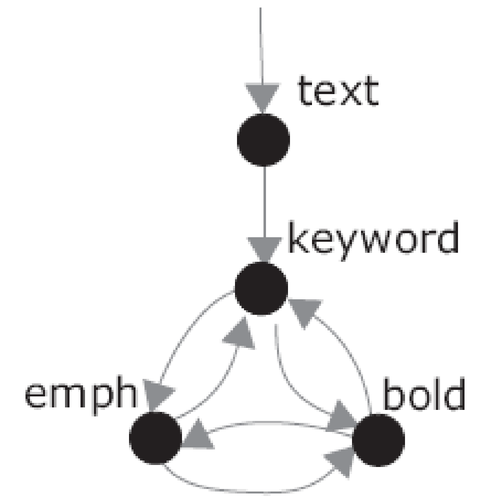
- Without indexes for level-values, retrievals of sibling nodes are inefficient

Amoeba Join Performance

	XMark (factor = 0.1, 12M)					XMark (factor = 0.5, 57M)					XMark (factor = 1.0, 114M)				
	QK	SW/I	SW/S	BF/I	BF/S	QK	SW/I	SW/S	BF/I	BF/S	QK	SW/I	SW/S	BF/I	BF/S
Q1	2.71	**0.39**	5.47	> *8d*	> *8d*	22.91	**1.97**	30.81	> *3y*	> *3y*	62.20	**4.17**	69.09	> *24y*	> *24 y*
Q2	**0.06**	0.32	5.57	106.75	115.94	**0.05**	1.20	29.34	> *0.5h*	> *0.5h*	**0.06**	2.67	67.12	> *11h*	> *11h*
Q3	**0.05**	0.11	5.43	20.02	26.42	**0.07**	3.97	29.41	> *0.1h*	> *0.1h*	**0.06**	8.95	66.02	> *0.5h*	> *0.5h*
Q4	**0.06**	0.41	7.98	> *30y*	> *30y*	**0.05**	10.96	43.41	> *162c*	> *162c*	**0.07**	22.12	90.95	> *2631c*	> *2631c*

Q1 : AJ(emph, bold, keyword)
Q2 : AJ(emph, bold, keyword=>"aboard notes")
Q3 : AJ(item, @id="item100", description)
Q4 : AJ(item, @id="item100", description, location, text)

h : hours (= 3600 sec), d : days (= 24h), y: years (= 365d), c: centuries (= 100y)



- Algorithm
  - QK: Quicker, SW: Sweep, BF: Brute Force
- Index
  - I: Index Scan, S: Sequential Scan
- Quicker algorithm is fastest when we can localize search regions

Improvement by AJ Decomposition

	query statement	1M	#result	5M	#result	25M	#result	50M	#result
Q1	*AJ(book, order)*	0.24	9403	1.14	46146	5.60	223637	11.34	447174
Q2	*AJ(customer, order)*	0.14	9403	0.64	46146	3.24	223637	6.44	447174
Q3	*AJ(order, book, customer)*	3.00	583603	14.89	2857746	Out of Memory		Out of Memory	
Q4	*AJ$_F$(order, book, customer)*	**0.51**	**9403**	**2.11**	**46146**	**10.41**	**223637**	**21.25**	**447174**
Q5	*AJ*(book, order)*	0.32	16094	1.51	79102	7.46	383420	15.17	766740
Q6	*AJ*(book, [order, pending, titile])*	1.17	66094	7.74	359102	112.08	1753420	486.66	3516740
Q7	*AJ*$_F$(book, [order, pending, titile])*	**0.91**	**16594**	**4.30**	**81902**	**21.19**	**397120**	**88.51**	**792420**

	query statement	schedule
Q1	*AJ(book, order)*	*AJ (book, order)*
Q2	*AJ(customer, order)*	*AJ (customer, order)*
Q3	*AJ(order, book, customer)*	*AJ (order, book, customer)*
Q4	*AJ$_F$(order, book, customer)*	*AJ$_{o.c}$ (AJ$_{o.b}$ (order, book), customer)*
Q5	*AJ*(book, order)*	*AJ* (book, order)*
Q6	*AJ*(book, [order, pending, titile])*	*AJ*$_{b,o,p,t}$ (PC(book, book@isbn), order, pending, titile)*
Q7	*AJ*$_F$(book, [order, pending, titile])*	*AJ*$_{b,o,p,t}$ (AJ*$_{b,o}$ (PC(book, book@isbn), order), pending, title)*

- Without decomposing amoeba joins, the number of XML structures to be retrieved explodes.

Perspectives

Applications

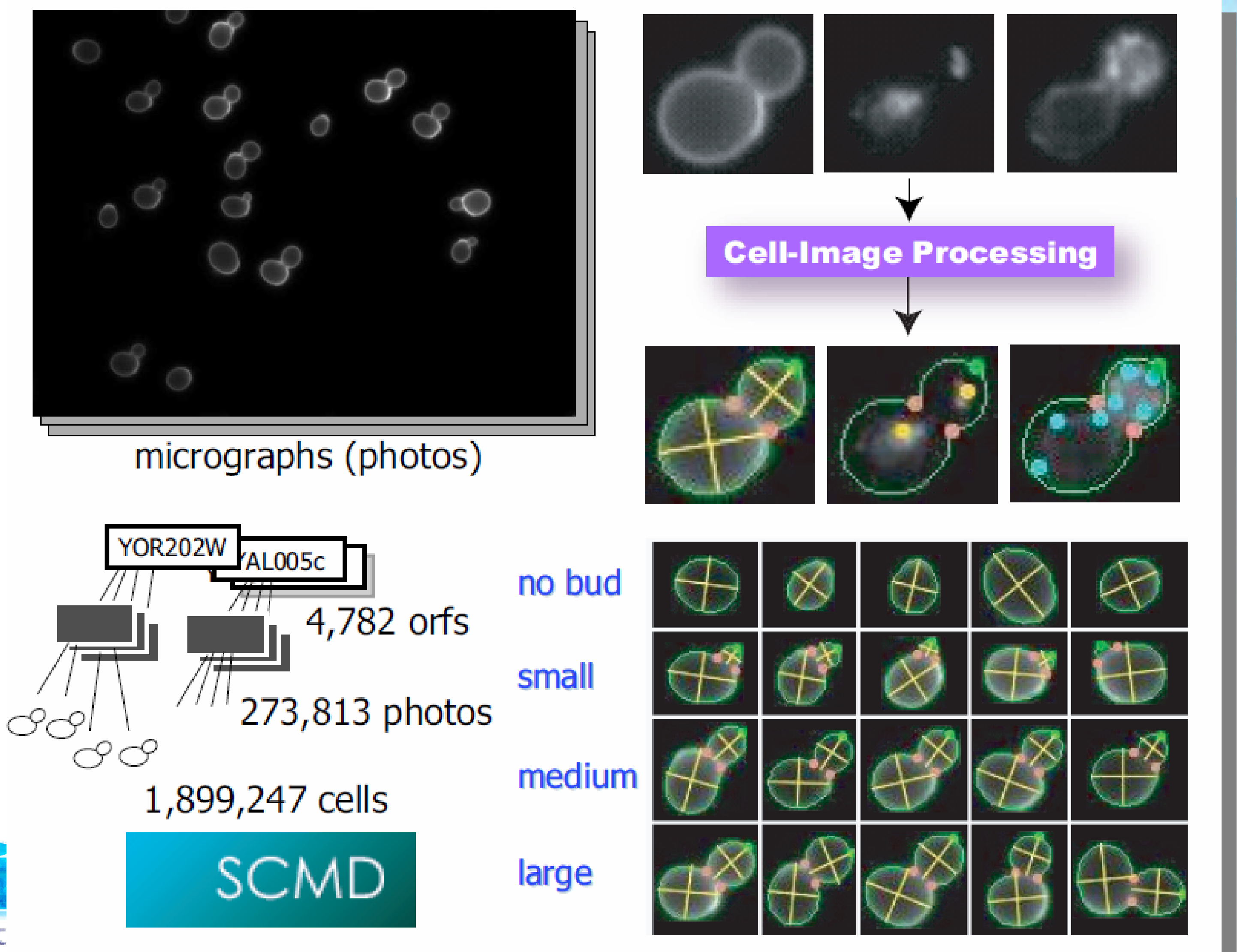
- Our methods can be applied various XML databases

- Examples of promising applications
- File Systems
  - Represent files with XML format
    - reorganization and enhancing information of files with tags

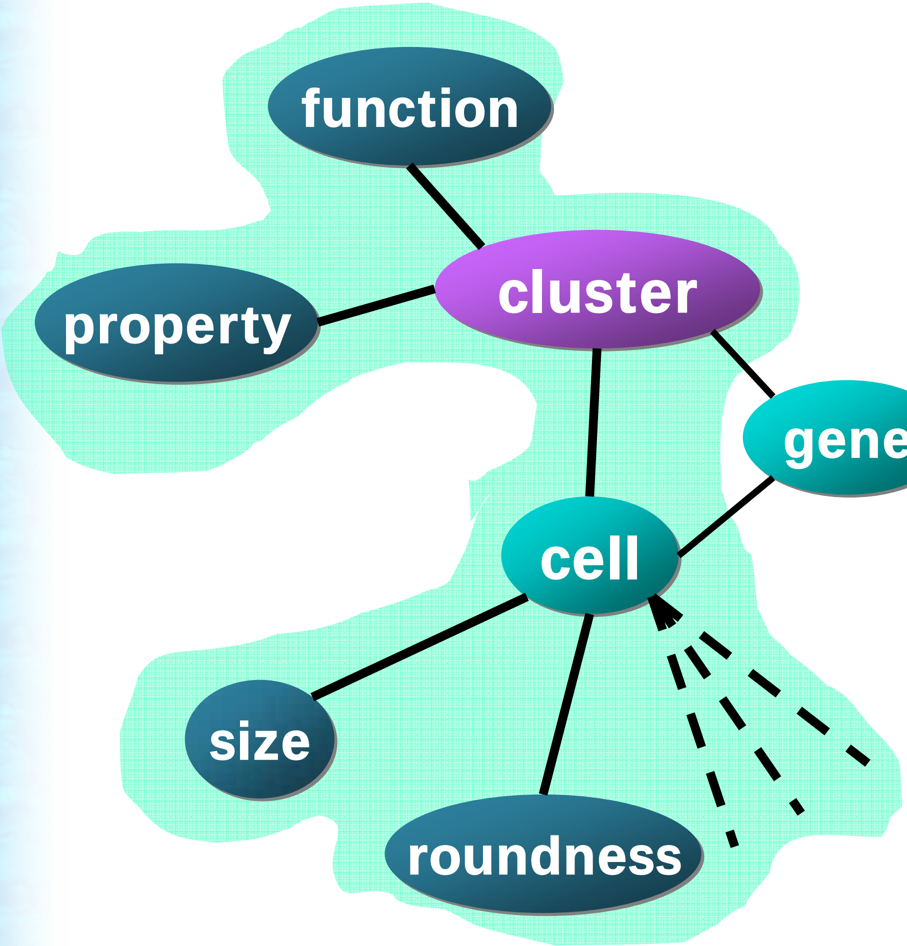
- Bioinformatics
  - Reorganization of data is frequent
    - Statistical analysis (classification, transformation, cleansing, etc.)
  - Integration of various data sources is required

SCMD

- SCMD (*Saccharomyces Cerevisiae* Morphological Database)
  - [NAR04], [NAR05], [PNAS05]



Deep Copies of XML Data

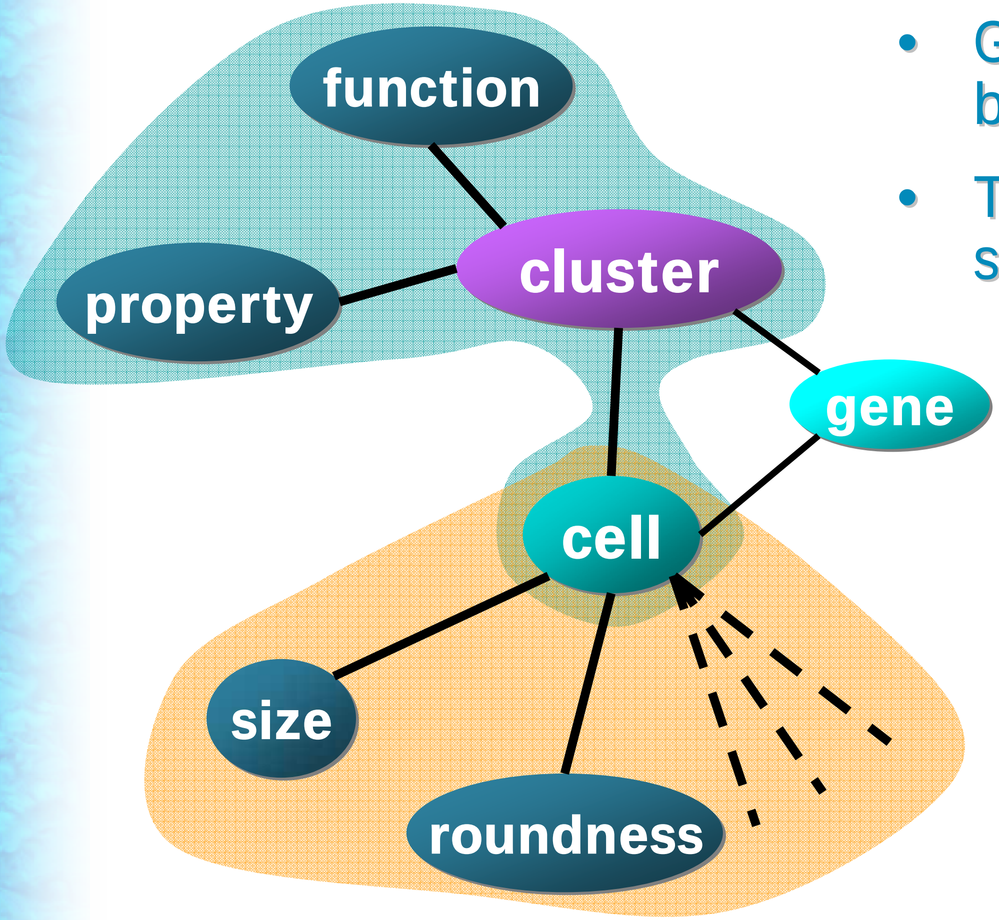



```
<cell>
  <size>...</size>
  <roundness>...</roundness>
  <cluster>
    <function>...</function>
    <property>...</property>
  </cluster>
</cell>
```

```
<cluster>
  <function>...</function>
  <property>...</property>
  <cell>
    <size> ... </size>
    <roundness>..</roundness>
  </cell>
</cluster>
```

- Many duplicates (deep copies) of data

Shallow Copies of XML Data



- Graph-structured data model can be decomposed into several trees
- To connect nodes in trees, we need shallow copies of nodes.

```
<cell id="1">
  <size>...</size>
  <roundness>...</roundness>
</cell>
```

```
<cluster>
  <function>...</function>
  <property>...</property>
  <cell id="1"/>
</cluster>
```

- With FD-based query processing
  - It becomes easier to manage shallow-copy representation of XML data

Future Work

- Query Optimization
  - Efficient amoeba join decomposition scheduling
  - Integration of index-lookups and cost-based optimization
  - Indexes for amoeba structures
- More complex semantics
  - Ownerships of nodes
  - Scope of attributes

```
<dept name="R&D">
   <manager>David</manager>
   <section name="Biology">
       <manager>Kevin</manager>
   </section>
</dept>
```

- Updates of XML Data
  - Detecting violation of FDs
  - Automatically constructs XML structures
    - From unstructured data

Our Contributions

- Amoeba Join
  - Tracks various XML structures
- Functional Dependency
  - defines XML structures of interest
  - Conceptual change: Data model (FD) defines XML structures
- Amoeba Join Decomposition
  - makes faster the FD-based query processing
- XML Indexing
  - A space-efficient XML indexing technique

Thank you!

This is the end of the presentation