ADT 2010

XQuery Updates in MonetDB/XQuery

Stefan Manegold
Stefan.Manegold@cwi.nl
http://www.cwi.nl/~manegold/

Schedule

- 09.11.2010:
  - RDBMS back-end support for XML/XQuery (1/2):
    - Document Representation (*XPath Accelerator*, *Pre/Post plane*)
- 16.11.2010:
  - XPath navigation (*Staircase Join*)
  - XQuery to Relational Algebra Compiler:
    - Item- & Sequence- Representation
    - Efficient FLWoR Evaluation (*Loop-Lifting*)
    - Optimization
- 23.11.2010:
  - ***RDBMS back-end support for XML/XQuery (2/2):***
    - ***Updateable Document Representation***
  - Other (DB-) approaches to XML/XQuery processing

XML/XQuery Updates

- XQuery Update Facility 1.0
  - W3C Candidate Recommendation
  - http://www.c3.org/TR/xquery-update-10/
- Categorize updates into
  - Value updates
  - Structural updates

(MonetDB/XQuery does not yet support the latest syntax changes made by W3C; for details see
http://monetdb.cwi.nl/XQuery/Documentation/XQuery-Updates.html)

Value Updates

```
do replace value of fn:doc("bib.xml")/books/book[1]/price
with fn:doc("bib.xml")/books/book[1]/price * 1.1

do replace value of fn:doc(“bib.xml”)/books/book[2]/@isbn
with “90-6196-517-9”

do rename fn:doc(“bib.xml”)/books/book[3]/author[1]
into “primary-author”

do rename fn:doc(“bib.xml”)/journals/journal[9]/@isbn
into “issn”
```

=> map directly to simple value updates in relational storage

Structural Updates

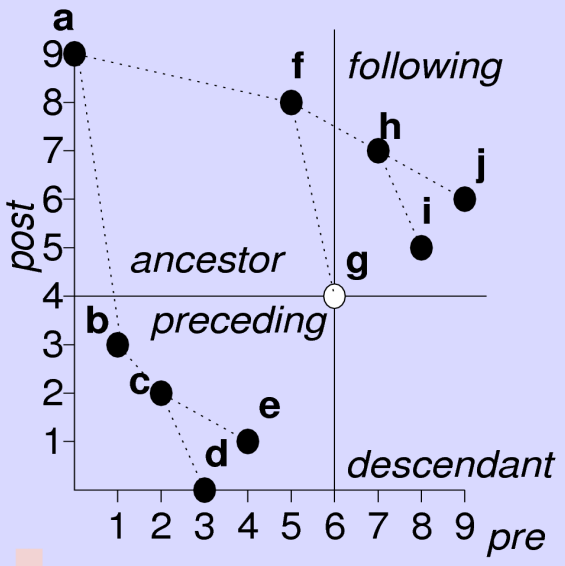
```
do insert attribute isbn {“90-6196-517”}
into fn:doc("bib.xml")/books/book[17]

do delete fn:doc(“bib.xml”)/books/book[2]/@wrong


do insert <author>Stefan Manegold</author>
after fn:doc(“bib.xml”)/books/book[33]/author[last()]

do replace fn:doc(“bib.xml”)/books/book[44]/author[1]
with fn:doc(“bib.xml”)/books/book[33]/author[last()]

do delete fn:doc(“bib.xml”)/books/book[author = “Kermit”]
```

=> How to implement on pre-/post-encoding?

XML/XQuery Updates

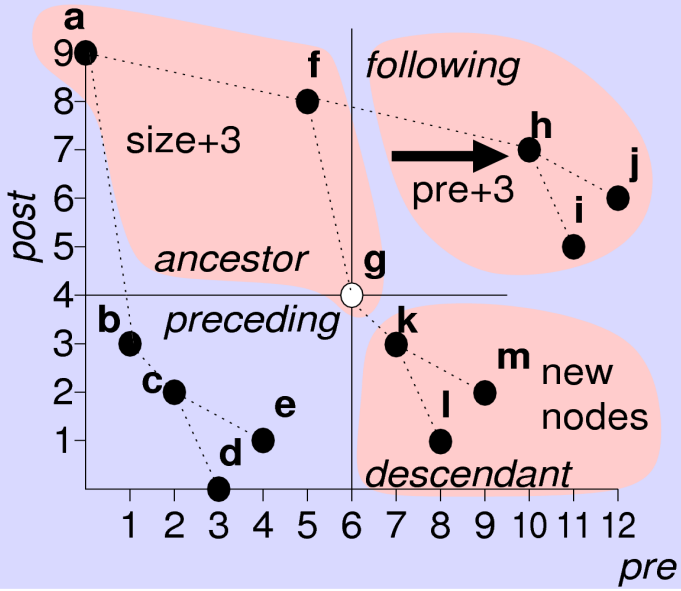
	pre	size	level	post
a	0	9	0	9
b	1	3	1	3
c	2	2	2	2
d	3	0	3	0
e	4	0	3	1
f	5	4	1	8
g	6	0	2	4
h	7	2	2	7
i	8	0	3	5
j	9	0	3	6

(pre+size−level=post)



do insert <k><l/><m/></k> as first into /a/f/g

	pre	size	level
a	0	12	0
b	1	3	1
c	2	2	2
d	3	0	3
e	4	0	3
f	5	7	1
g	6	3	2
h	10	2	2
i	11	0	3
j	12	0	3
k	7	2	3
l	8	0	4
m	9	0	4



XML/ XQuery Updates

Read–Only vs. Updatable Representation

pre	size	level	
0	9	0	a
1	3	1	b
2	2	2	c
3	0	3	d
4	0	3	e
5	4	1	f
6	0	2	g
7	2	2	h
8	0	3	i
9	0	3	j

rid	size	level	
0	10	0	a
1	3	1	b
2	2	2	c
3	0	3	d
4	0	3	e
5	5	1	f
6	1	2	g
7	0	null	
8	2	2	h
9	0	3	i
10	0	3	j
11	4	null	
12	3	null	
13	2	null	
14	1	null	
15	0	null	

here: pagesize = 8

unused space:

- **level = NULL**
- **size set to unite consecutive space**

do insert <k><l/><m/></k>
as first into /a/f/g

- **first try to handle the insert inside a page**
- **if full, append pages (NULL padded)**

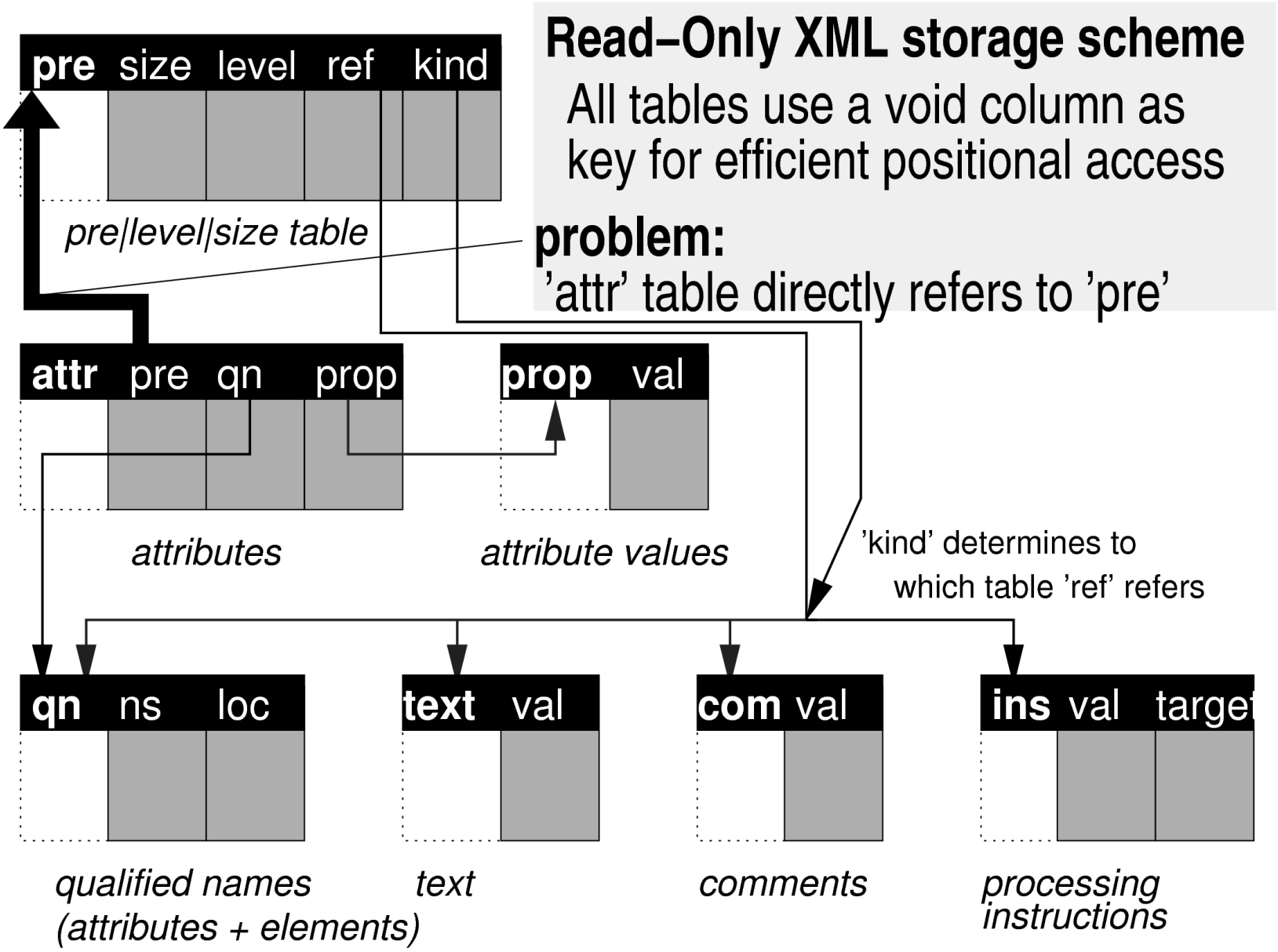
rid	size	level	
0	18	0	a
1	3	1	b
2	2	2	c
3	0	3	d
4	0	3	e
5	13	1	f
6	9	2	g
7	2	3	k
8	2	2	l
9	0	3	m
10	0	3	
11	4	null	
12	3	null	
13	2	null	
14	1	null	
15	0	null	
16	0	4	h
17	0	4	i
18	5	null	j
19	4	null	
20	3	null	
21	2	null	
22	1	null	
23	0	null	

pre	size	level	
0	18	0	a
1	3	1	b
2	2	2	c
3	0	3	d
4	0	3	e
5	13	1	f
6	9	2	g
7	2	3	k
8	0	4	l
9	0	4	m
10	5	null	
11	4	null	
12	3	null	
13	2	null	
14	1	null	
15	0	null	
16	2	2	h
17	0	3	i
18	0	3	j
19	4	null	
20	3	null	
21	2	null	
22	1	null	
23	0	null	

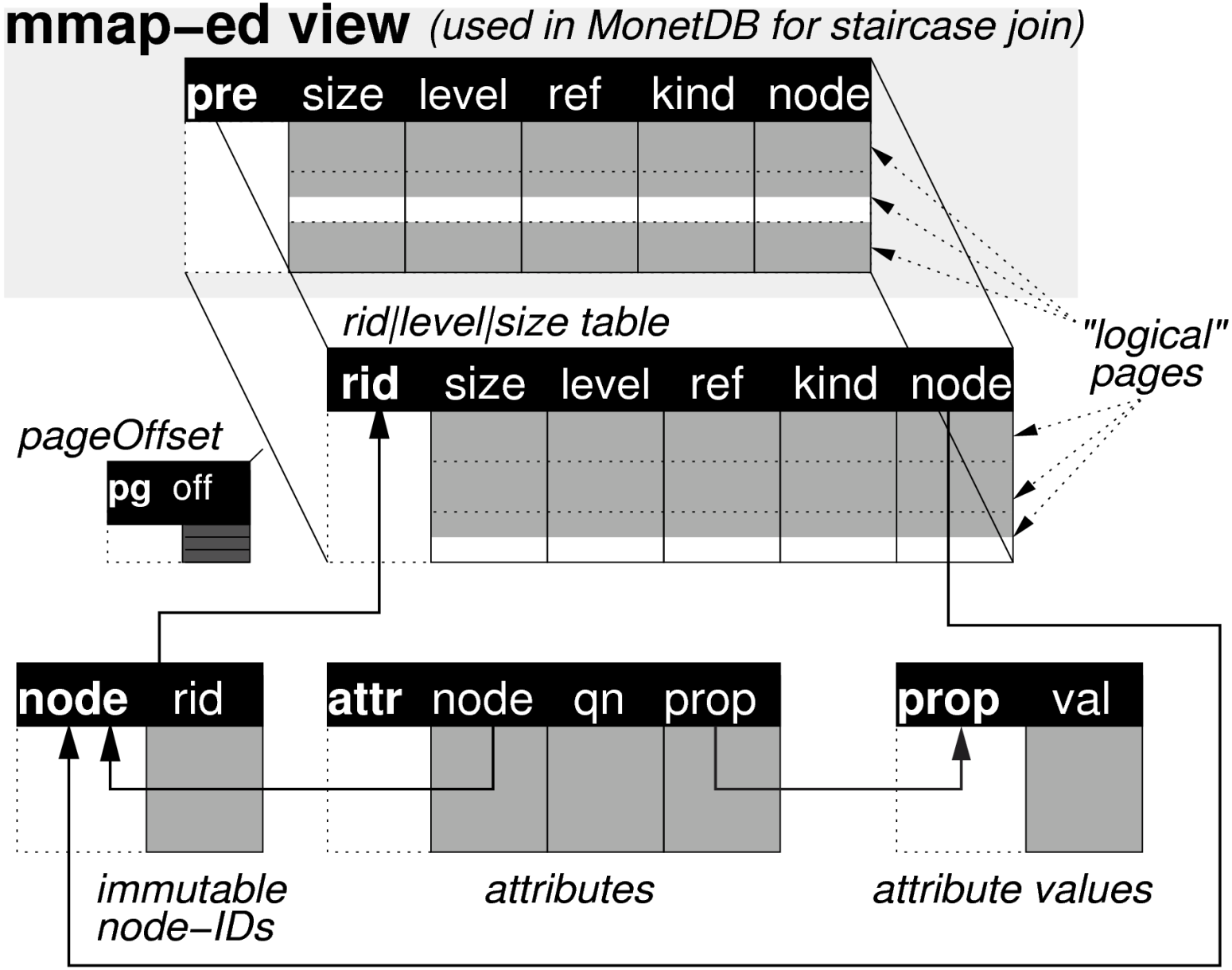
pg	off
0	0
1	2
2	1

- **pre|size|level is table VIEW with pages in logical order**
- **pre is a ROWNUM in the view, therefore it adapts automatically**

XML/XQuery Updates

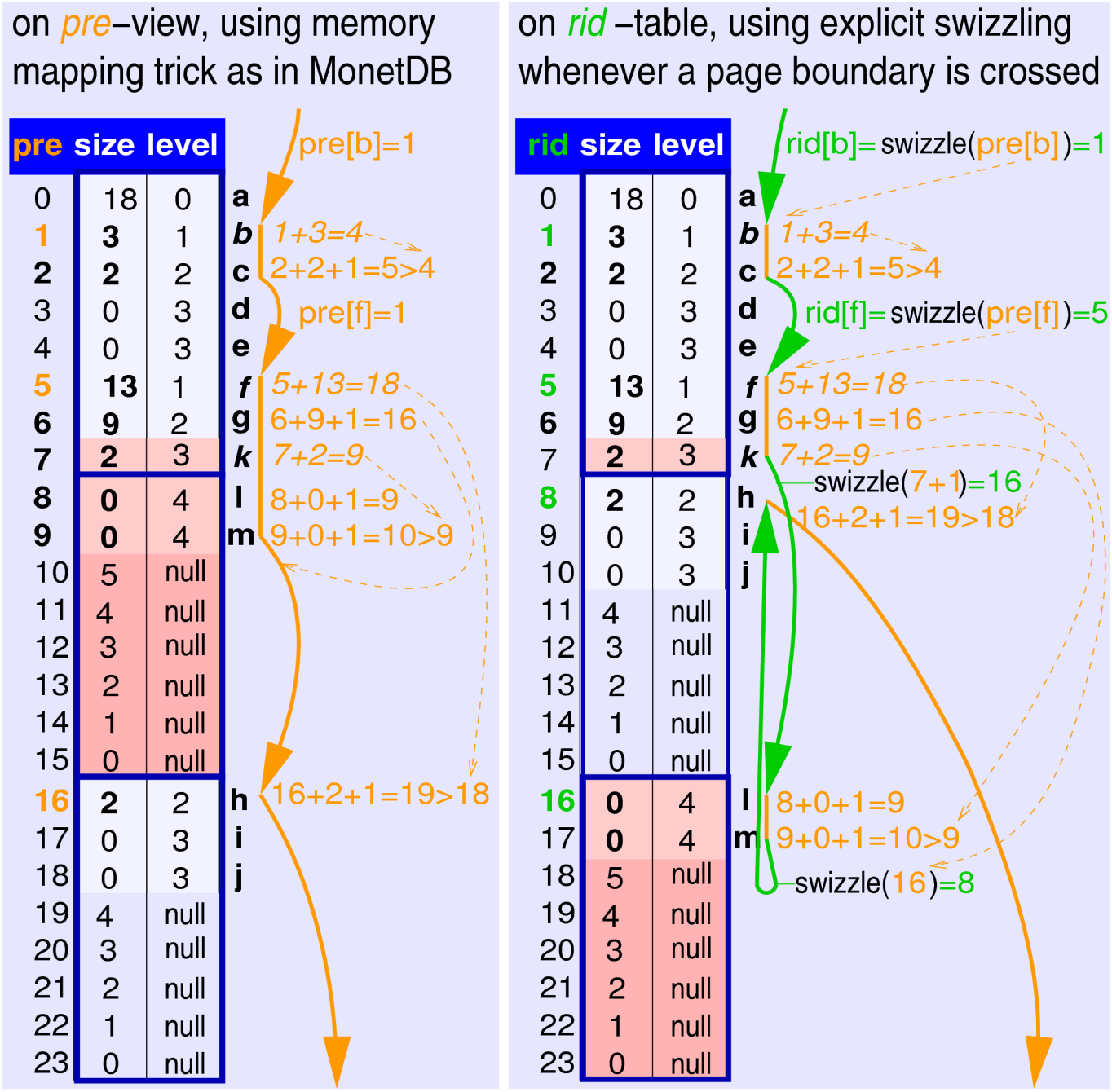


XML/XQuery Updates



XML/ XQuery Updates

Staircase Join



XML Storage Revisited

	pre	post
a	0	9
b	1	3
c	2	2
d	3	0
e	4	1
f	5	8
g	6	4
h	7	7
i	8	5
j	9	6

pre	size	level
0	9	0
1	3	1
2	2	2
3	0	3
4	0	3
5	4	1
6	0	2
7	2	2
8	0	3
9	0	3

post = pre + size - level

pre	size	level
0	11	0
1	5	1
2	-1	null
3	null	null
4	2	2
5	0	3
6	0	3
7	4	1
8	0	2
9	2	2
10	0	3
11	0	3

Allow holes

rid	size	level	nid
0	11	0	N0
1	5	1	N1
2	-1	null	null
3	0	null	null
4	2	2	N2
5	0	3	N3
6	0	3	N4
7	4	1	N5
8	0	2	N6
9	2	2	N7
10	0	3	N8
11	0	3	N9

Define logical pages

XML Storage Revisited

	pre	post
a	0	9
b	1	3
c	2	2
d	3	0
e	4	1
f	5	8
g	6	4
h	7	7
i	8	5
j	9	6

pre	size	level
0	9	0
1	3	1
2	2	2
3	0	3
4	0	3
5	4	1
6	0	2
7	2	2
8	0	3
9	0	3

post = pre + size - level

pre	size	level
0	11	0
1	5	1
2	-1	null
3	null	null
4	2	2
5	0	3
6	0	3
7	4	1
8	0	2
9	2	2
10	0	3
11	0	3

Allow holes

rid	size	level	nid
0	11	0	N0
1	5	1	N1
2	-1	null	null
3	0	null	null
4	0	2	N6
5	2	2	N7
6	0	3	N8
7	0	3	N9
8	2	2	N2
9	0	3	N3
10	0	3	N4
11	4	1	N5

Define logical pages

rid = pre.swizzle(

page	map
0	0
1	2
2	1

)

XML Storage Revisited

Update-friendly
- **rid-table is append-only**
- **rid-tuples may be unused**
- **rid = autoincrement column**

MonetDB:
- rid not stored but computed
  (virtual oid)
- allows positional lookup/join

Opportunity currently not exploited by other RDBMS

Occurs widely in our XQuery translation.

rid	size	level	nid
0	11	0	N0
1	5	1	N1
2	-1	null	null
3	0	null	null
4	0	2	N6
5	2	2	N7
6	0	3	N8
7	0	3	N9
8	2	2	N2
9	0	3	N3
10	0	3	N4
11	4	1	N5

XML Storage Revisited

Update-friendly
- **rid-table is append-only**
- **rid-tuples may be unused**
- **rid = autoincrement column**

MonetDB:
- rid not stored but computed (virtual oid)
- allows positional lookup/join

Opportunity currently not exploited by other RDBMS

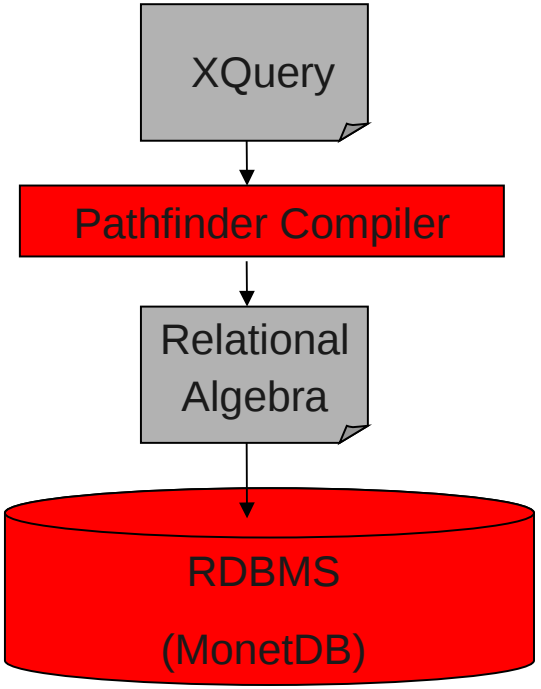
Occurs widely in our XQuery translation.

rid	size	level	nid
0	11	0	N0
1	5	1	N1
2	-1	null	null
3	0	null	null
4	0	2	N6
5	2	2	N7
6	0	3	N8
7	0	3	N9
8	2	2	N2
9	0	3	N3
10	0	3	N4
11	4	1	N5

MonetDB/XQuery

Our own XML DBMS with (almost..) full XQuery support.

- *Built purely on an RDBMS, namely MonetDB*



Pathfinder compiler & "staircase join":

Universität Tübingen (Torsten Grust, et al.)

Technical University Twente (Maurice van Keulen, et. al.)

MonetDB High-Performance DBMS

CWI Amsterdam (Peter Boncz, Stefan Manegold, ...)

Useful for:

- ***Large** XML databases!*
- *Querying XML annotations (multimedia, forensic ➔ NFI)*
- *XML information retrieval*
- *...*

Research Projects & Extensions

- *Value indeces*
- *Runtime optimization*
  - ***SIGMOD'09 [Abdel Kader, Boncz, v. Keulen Manegold]***
- *Algebraic Query Optimization*
  - *Grust, Rittinger, et al. (Universität Tübingen)*
- *Distributed XQuery ➔ P2P XQuery*
  - *SOAP group communication, XQuery RPC*
  - ***VLDB'07 [Zhang, Boncz]***
- *Benchmarking beyond XMark*
  - ***ExpDB'06 Workshop [Manegold]***
- *Support for XML Interval Annotations*
  - ***XIME-P'06 Workshop [Alink et al.]***
- *Xquery + Information Retrieval: PF/Tijah*

Conclusions

- *Relational approach can be scalable & fast*
  - *MonetDB/XQuery compares favorably with all other available systems*
- *Techniques that made it work*
  - *Property-driven peephole optimization*
    - *Order & other properties*
  - *Loop-lifted XPath steps*
    - *Evaluate Sets of context nodes in a single pass*
  - *Support for dense (autoincrement) keys*
    - *Positional lookup*
- *Background Information & Literature*

  `http://monetdb-xquery.org`

  `http://pathfinder-xquery.org`

Schedule

- 09.11.2010:
  - RDBMS back-end support for XML/XQuery (1/2):
    - Document Representation (*XPath Accelerator*, *Pre/Post plane*)
- 16.11.2010:
  - XPath navigation (*Staircase Join*)
  - XQuery to Relational Algebra Compiler:
    - Item- & Sequence- Representation
    - Efficient FLWoR Evaluation (*Loop-Lifting*)
    - Optimization
- 23.11.2010:
  - RDBMS back-end support for XML/XQuery (2/2):
    - Updateable Document Representation
  - ***Other (DB-) approaches to XML/XQuery processing***

Topics

- **Other approaches & techniques (***selection, far from complete!***)**
  - Document storage / tree encoding:
    - ORDPATH
    - DataGuides
  - XPath processing:
    - Tree patterns, holistic twig joins

Fixed-Width Tree Encodings & Updates

- **Fixed-width** tree encoding (like XPath Accelerator) are
  - Good for read(-only) processing
    - small footprint, positional lookup, staircase join
  - But inherently **static**
- **Milo et al., PODS 2002:**

  "There is a sequence of updates (subtree insertions) for any *persistent* tree encoding scheme E (*where each node keeps its initial encoding label even under updates*), such that E **needs labels of length Ω(N)** to encode the resulting tree of N nodes."

Fixed-Width Tree Encodings & Updates

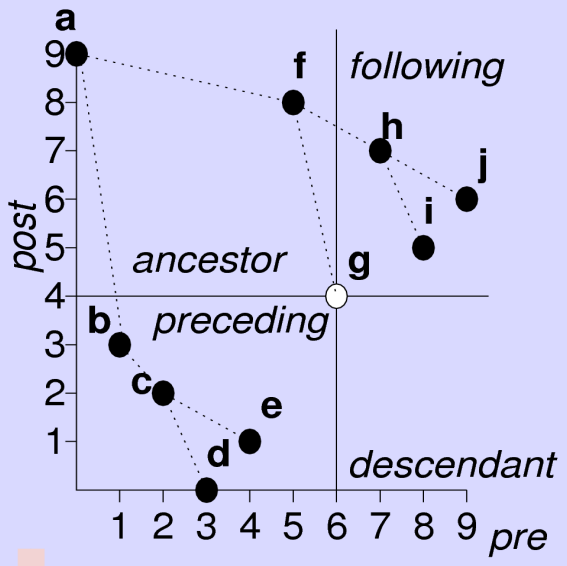
- **Fixed-width** tree encoding (like XPath Accelerator) are
  - Good for read(-only) processing
    - small footprint, positional lookup, staircase join
  - But inherently **static**
- **Milo et al., PODS 2002:**

  "There is a sequence of updates (subtree insertions) for any *persistent* tree encoding scheme E (*where each node keeps its initial encoding label even under updates*), such that E **needs labels of length Ω(N)** to encode the resulting tree of N nodes."

XML/XQuery Updates

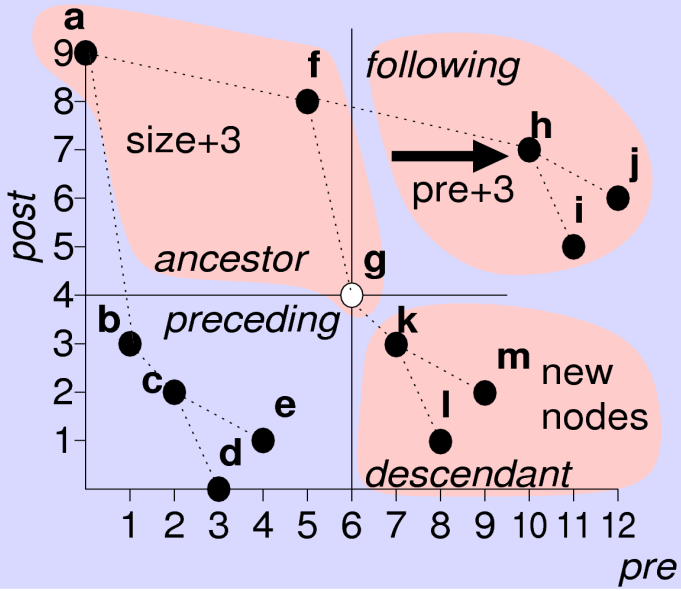
	pre	size	level	post
a	0	9	0	9
b	1	3	1	3
c	2	2	2	2
d	3	0	3	0
e	4	0	3	1
f	5	4	1	8
g	6	0	2	4
h	7	2	2	7
i	8	0	3	5
j	9	0	3	6

(pre+size−level=post)



do insert <k><l/><m/></k> as first into /a/f/g

	pre	size	level
a	0	12	0
b	1	3	1
c	2	2	2
d	3	0	3
e	4	0	3
f	5	7	1
g	6	3	2
h	10	2	2
i	11	0	3
j	12	0	3
k	7	2	3
l	8	0	4
m	9	0	4



XML/ XQuery Updates

MonetDB/XQuery *hack*:

exploit paging & mmap trick

but:

updating pg|off is still *O(N)*

Read–Only vs. Updatable Representation

pre	size	level	
0	9	0	a
1	3	1	b
2	2	2	c
3	0	3	d
4	0	3	e
5	4	1	f
6	0	2	g
7	2	2	h
8	0	3	i
9	0	3	j

rid	size	level	
0	10	0	a
1	3	1	b
2	2	2	c
3	0	3	d
4	0	3	e
5	5	1	f
6	1	2	g
7	0	null	
8	2	2	h
9	0	3	i
10	0	3	j
11	4	null	
12	3	null	
13	2	null	
14	1	null	
15	0	null	

here: pagesize = 8

unused space:

- **level = NULL**
- **size set to unite consecutive space**

do insert <k><l/><m/></k>
as first into /a/f/g

rid	size	level	
0	18	0	a
1	3	1	b
2	2	2	c
3	0	3	d
4	0	3	e
5	13	1	f
6	9	2	g
7	2	3	k
8	2	2	l
9	0	3	m
10	0	3	
11	4	null	
12	3	null	
13	2	null	
14	1	null	
15	0	null	
16	0	4	h
17	0	4	i
18	5	null	j
19	4	null	
20	3	null	
21	2	null	
22	1	null	
23	0	null	

pre	size	level	
0	18	0	a
1	3	1	b
2	2	2	c
3	0	3	d
4	0	3	e
5	13	1	f
6	9	2	g
7	2	3	k
8	0	4	l
9	0	4	m
10	5	null	
11	4	null	
12	3	null	
13	2	null	
14	1	null	
15	0	null	
16	2	2	h
17	0	3	i
18	0	3	j
19	4	null	
20	3	null	
21	2	null	
22	1	null	
23	0	null	

- **first try to handle the insert inside a page**
- **if full, append pages (NULL padded)**

pg	off
0	0
1	2
2	1

- **pre|size|level is table VIEW with pages in logical order**
- **pre is a ROWNUM in the view, therefore it adapts automatically**

Fixed-Width Tree Encodings & Updates

- **Fixed-width** tree encoding (like XPath Accelerator) are
  - Good for read(-only) processing
    - small footprint, positional lookup, staircase join
  - But inherently **static**
- **Non-solutions**:
  - **Gaps** in the encoding (*never large enough*)
  - Encoding based on **decimal fractions** (*limited precision*)
- **Possible solution**:
  - **Variable-width** tree encodings:
    - Cheaper updates
    - At the expense of more expensive read(-only) processing

A Variable-Width Tree Encoding: ORDPATH

- The **ORDPATH** encoding (used in MS SQL Server™) assigns node labels of **variable length.**

ORDPATH labels for an XML fragment

① The fragment root receives label 1.

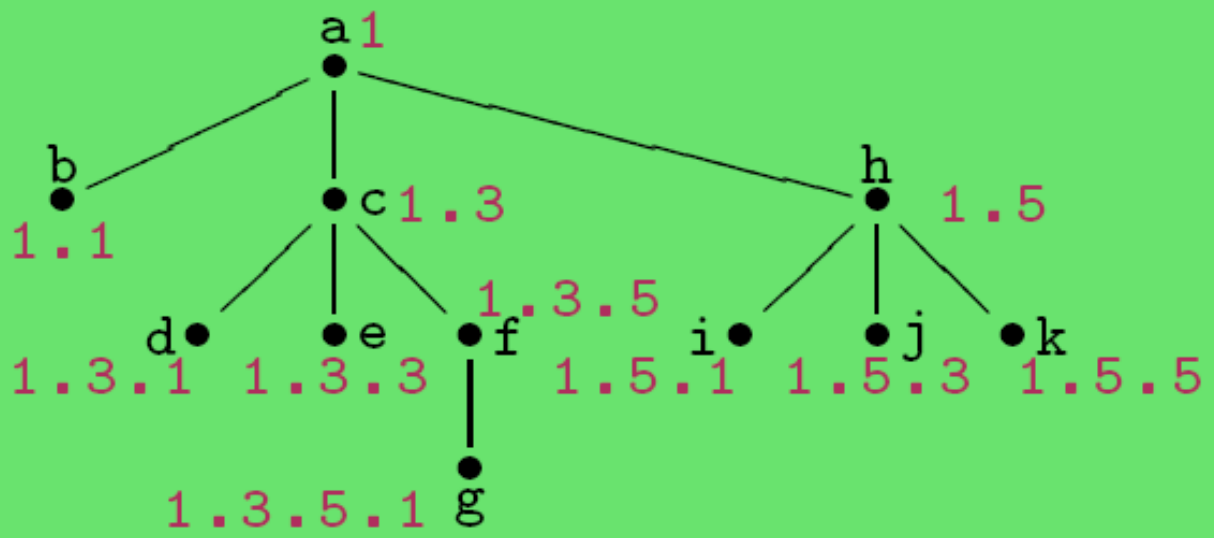
② The nth ($n = 1, 2, \ldots$) child of a parent node labelled p receives label $p\,.\,(2 \cdot n - 1)$.

- Internally, ORDPATH labels are not stored as .-separated ordinals but using a prefix-encoding (similarities with Unicode).
- O'Neil et al., SIGMOD 2004.

ORDPATH Encoding: Example

ORDPATH encoding of a sample XML fragment

```
<a>
  <b/>
  <c>
    <d/><e/>
    <f><g/></f>
  </c>
  <h>
    <i/><j/><k/>
  </h>
</a>
```



- **Note:**
  - ▷ **Lexicographic** order of ORDPATH labels ≡ document order

⇒ **Clustered index on ORDPATH labels** will be helpful.

ORDPATH: Insertion Between Siblings

- In ORDPATH, the **insertion of new nodes** between two existing sibling nodes is referred to as "*careting in*" (caret $\hat{=}$ insertion mark, λ).

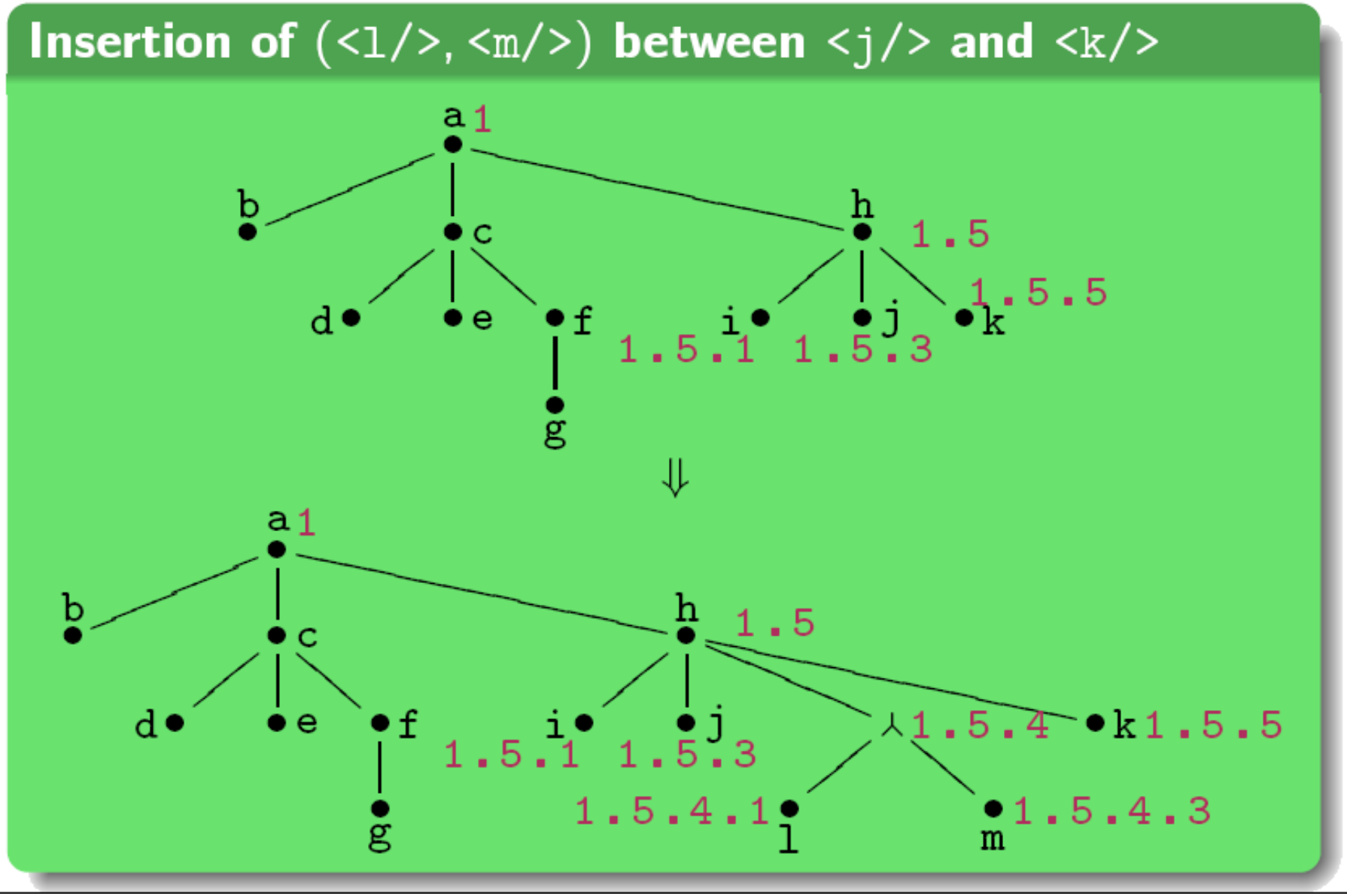
ORDPATH: node insertion

Let $(v_1, \ldots, v_n)$ denote a sequence of nodes to be inserted between two existing sibling nodes with labels $p\,.\,s$ and $p\,.\,(s+2)$, s odd. After insertion, the new label of v_i is

$$p\,.\,(s+1)\,.\,(2 \cdot i - 1) \quad .$$

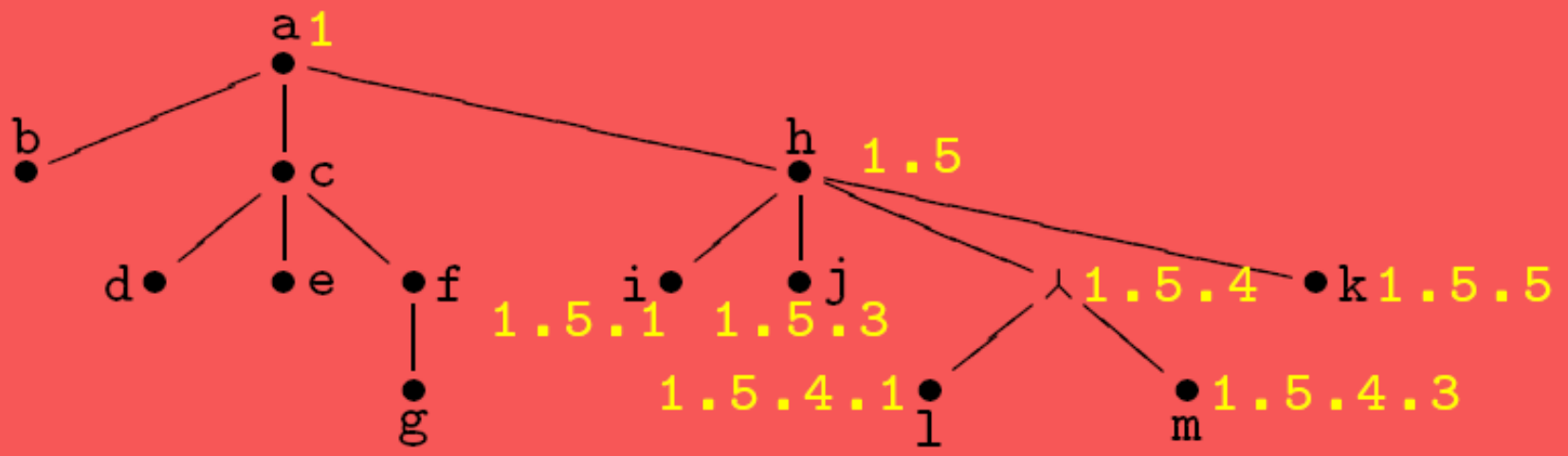
Label $p\,.\,(s+1)$ is referred to as a **caret.**

ORDPATH: Insertion Between Siblings



ORDPATH: Insertion Between Siblings

ORDPATH: Insertions at arbitrary locations?



Determine ORDPATH label of new node v inserted

① to the right of <k/>,
② to the left of <i/>,
③ between <j/> and <l/>,
④ between <l/> and <m/>.

Is ORDPATH suitable for XQuery?

Mapping core operations of the XQuery processing model to operations on ORDPATH labels:

`v/parent::node()`

① Let $p.m.n$ denote v's label (n is odd).
② If the rightmost ordinal (m) is even, remove it. Goto ②.

In other words: the carets (^) do not count for ancestry.

`v/descendant::node()`

① Let $p.n$ denote v's label (n is odd).
② Perform a lexicographic index range scan from $p.n$ to $p.(n+1)$—the *virtual following sibling* of v.

ORDPATH: Variable-Length Node Encoding

- For a 10 MB XML sample document, the authors of ORDPATH observed label lengths between 6 and 12 bytes.
- ORDPATH labels encode **root-to-node** paths => **common prefixes**.
  => Label comparisons often need to inspect encoding bits at the far right.

ORDPATH labels of <l/> and <m/>

```
1.5.4.1
1.5.4.3
```

- MS SQL Server employs further path encodings organized in **reverse** (node-to-root) order.
- **Note:** - Preorder ranks fit into CPU registers.
  - 4 byte pre's sufficient for 2^{32} = 4G nodes (11 GB XMark fits easily).
  - 8 byte pre's sufficient for 2^{64} nodes, i.e., “the universe” ...