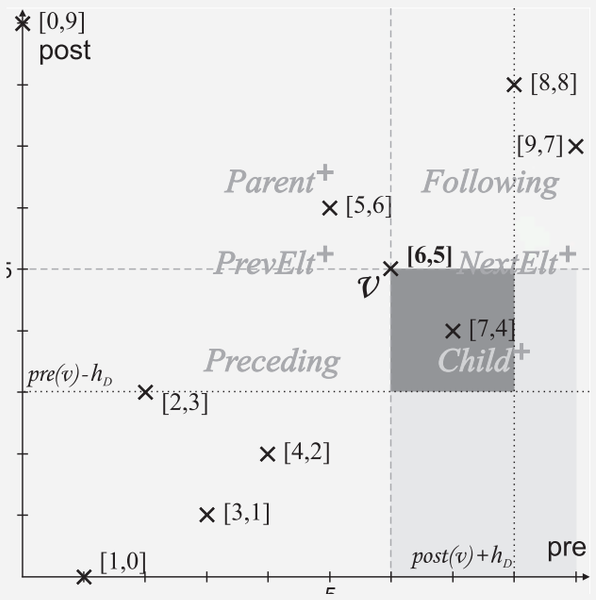
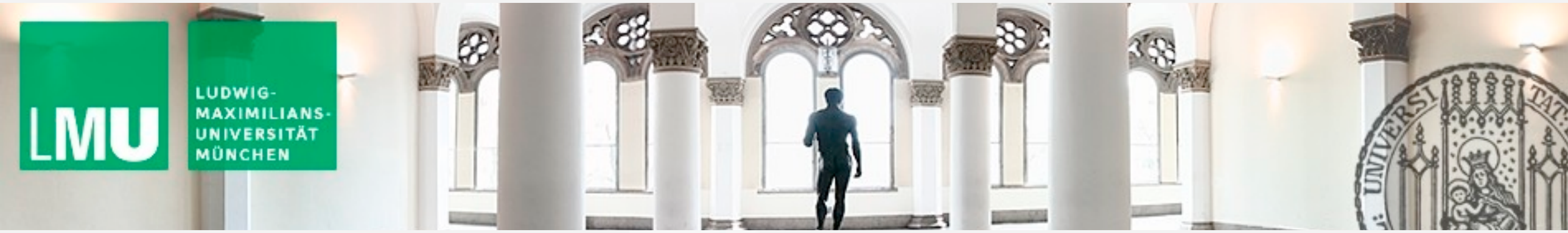
Institut
für
Informatik

Evaluating XPath Queries
with *XML Labeling Schemes*



Prof. Dr. François Bry

based on slides and articles by Georg Gottlob, Christoph Koch,
Thorsten Grust & Felix Weigel & Tim Furche

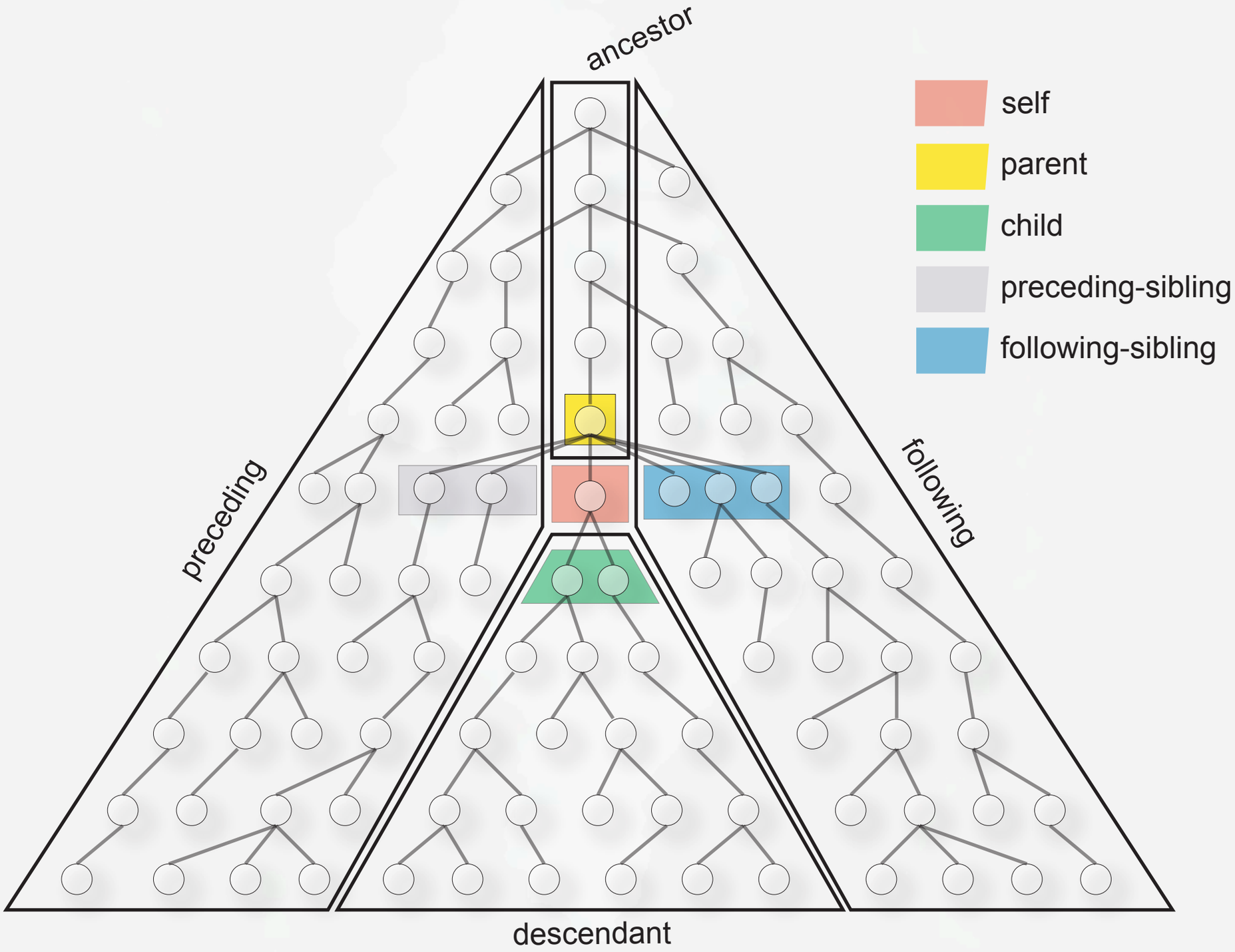


1

XPath Repetition

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

XPath Axes



Navigational XPath

Syntax

$\langle path \rangle$ ::= $\langle step \rangle \mid \langle step \rangle$ ‘/’ $\langle path \rangle \mid \langle path \rangle$ ‘∪’$\langle path \rangle \mid$ ‘/’ $\langle path \rangle$

$\langle step \rangle$ ::= $\langle axis \rangle$ ‘::’ $\langle node\text{-}test \rangle \mid \langle step \rangle$‘[’$\langle qualifier \rangle$‘]’

$\langle axis \rangle$ ::= ‘child’ | ‘descendant’ | ‘descendant-or-self’
| ‘next-sibling’ | ‘following-sibling’ | ‘following’

$\langle node\text{-}test \rangle$ ::= $\langle label \rangle \mid$ ‘node()’

$\langle qualifier \rangle$ ::= $\langle path \rangle \mid \langle path \rangle$ ‘∧’$\langle path \rangle \mid \langle path \rangle$ ‘∨’$\langle path \rangle \mid$ ‘¬’$\langle path \rangle$
| ‘lab()’ ‘=’ ‘λ’
| $\langle path \rangle$ ‘=’ $\langle path \rangle$

```
/child::conference/descendant::paper/child::author[child::node()
    =
 /child::conference/child::member/child::node()]
```

Navigational XPath

Semantics

$$[\![axis]\!]_{\text{Nodes}}(n) = \{(n' : R_{axis}(n, n')\}$$

$$[\![\lambda]\!]_{\text{Nodes}}(n) = \{(n' : \text{Lab}^{\lambda}(n')\}$$

$$[\![\texttt{node()}]\!]_{\text{Nodes}}(n) = \text{Nodes}(T)$$

$$[\![axis\texttt{::}nt\texttt{[}qual\texttt{]}]\!]_{\text{Nodes}}(n) = \{n' : n' \in [\![axis]\!]_{\text{Nodes}} \wedge n' \in [\![nt]\!]_{\text{Nodes}} \wedge [\![qual]\!]_{\text{Bool}}(n')\}$$

$$[\![step/path]\!]_{\text{Nodes}}(n) = \{n'' : n' \in [\![step]\!]_{\text{Nodes}}(n) \wedge n'' \in [\![path]\!]_{\text{Nodes}}(n')\}$$

$$[\![path_1 \cup path_2]\!]_{\text{Nodes}}(n) = [\![path_1]\!]_{\text{Nodes}}(n) \cup [\![path_2]\!]_{\text{Nodes}}(n)$$

$$[\![path]\!]_{\text{Bool}}(n) = [\![path]\!]_{\text{Nodes}}(n) \neq \emptyset$$

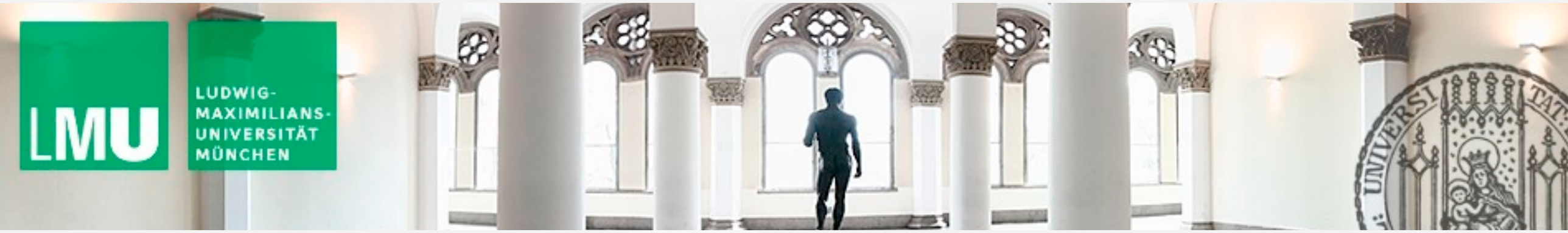
$$[\![path_1 \wedge path_2]\!]_{\text{Bool}}(n) = [\![path_1]\!]_{\text{Bool}}(n) \wedge [\![path_2]\!]_{\text{Bool}}(n)$$

$$[\![path_1 \vee path_2]\!]_{\text{Bool}}(n) = [\![path_1]\!]_{\text{Bool}}(n) \vee [\![path_2]\!]_{\text{Bool}}(n)$$

$$[\![\neg path]\!]_{\text{Bool}}(n) = \neg [\![path]\!]_{\text{Bool}}(n)$$

$$[\![\texttt{lab()} = \lambda]\!]_{\text{Bool}}(n) = \text{Lab}^{\lambda}(n)$$

$$[\![path_1 = path_2]\!]_{\text{Bool}}(n) = \exists n', n'' : n' \in [\![path_1]\!]_{\text{Nodes}}(n) \wedge n'' \in [\![path_2]\!]_{\text{Nodes}}(n) \wedge \cong (n', n'')$$



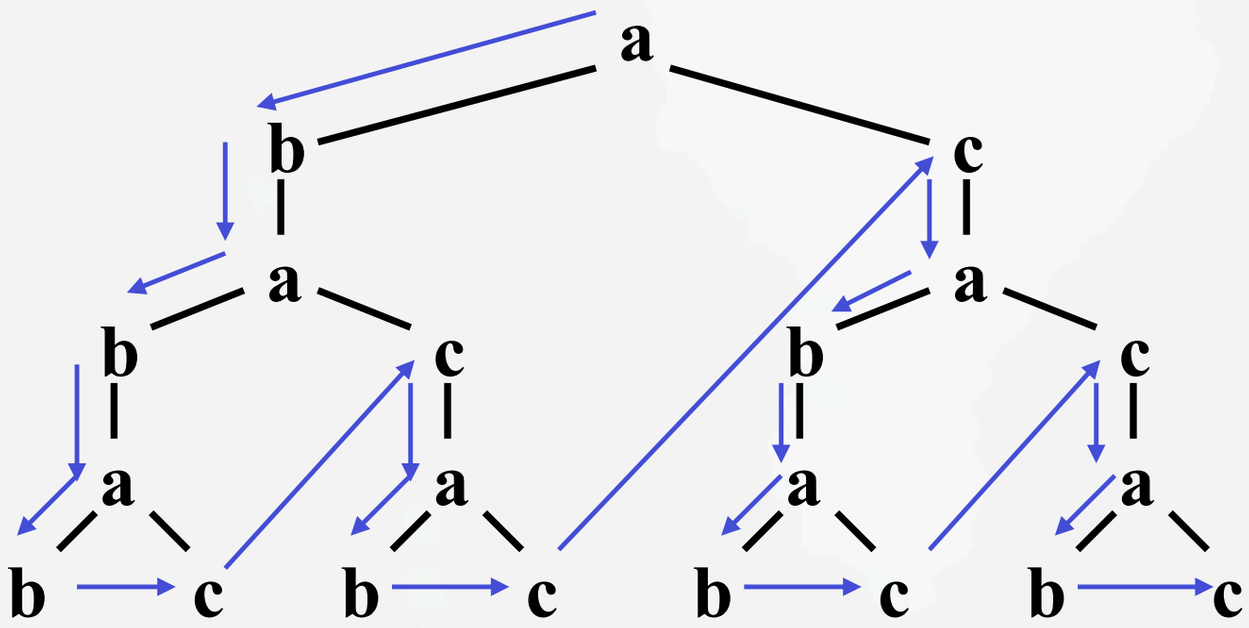
2

XPath Evaluation: Basics

Navigational XPath

Naïve Evaluation

procedure process-location-step(n_0, Q)
/* n_0 is the context node; query Q is a list of location steps */
begin
    node set S := apply Q.**head** to node n_0;
    if (Q.**tail** is not empty) **then**
        for each node $n \in S$ **do** process-location-step(n, Q.**tail**);
end

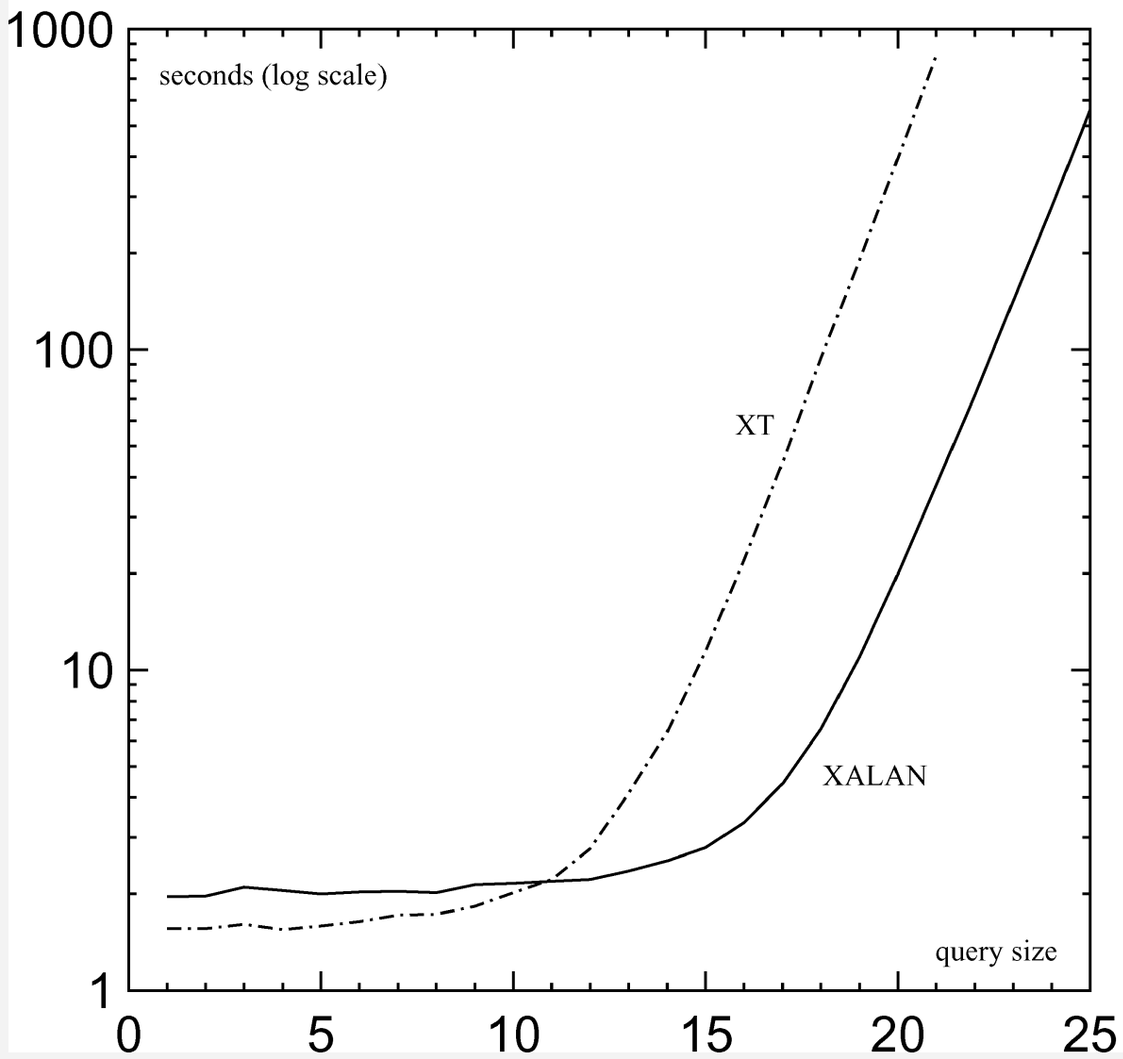


$$\text{Time}(|Q|) = |D|^{|Q|}$$

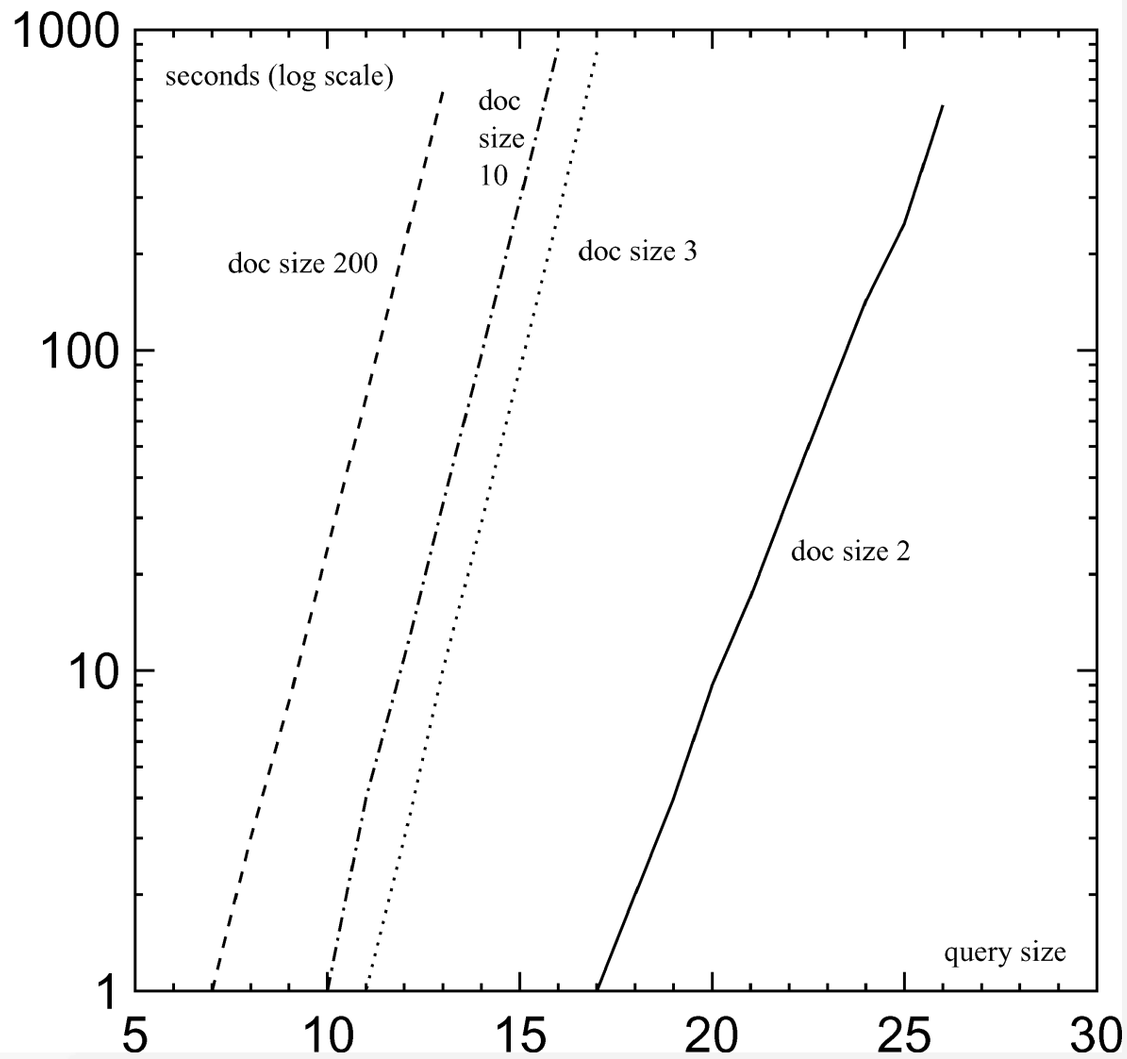
Navigational XPath

Naïve Evaluation

Experiment 1:



Experiment 2:



$\langle a \rangle \underbrace{\langle b/\rangle \ldots \langle b/\rangle}_{i \text{ times}} \langle /a \rangle$

‘//a/b/parent::a/b/parent::a/b’.

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Polynomial XPath

Bottom-up with Tabling

- Idea: **context-value tables** [Gottlob et al., 2002]
  - for each expression *e* store a context-value table
    - contains **pairs** of contexts ***c*** and values *v*
    - such that *e* evaluates to *v* in context ***c***
  - obviously each such table has at most polynomial size

- Compare: evaluation of **acyclic conjunctive queries** on RDBS
  - join tree, each intermediary result is an (at most) quadratic table
    - associating result for upper expression with result of sub-expression

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Polynomial XPath

Bottom-up with Tabling

descendant::b/following-sibling::*[position() != last()]

input context $\langle a, 1, 1\rangle$

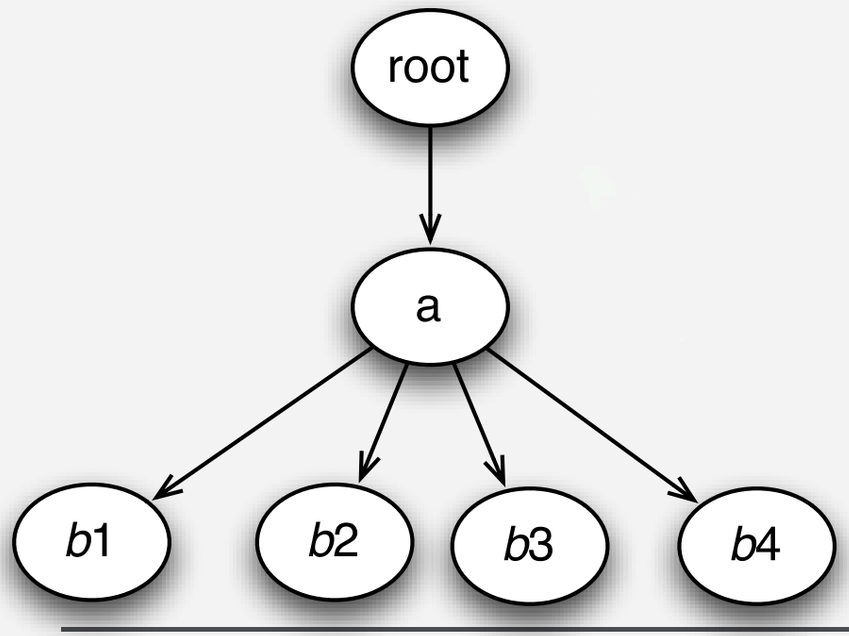
$\mathcal{E}_\uparrow[\![Q]\!]$	
x	val
r	$\{b_2, b_3\}$
a	$\{b_2, b_3\}$
b_1	$\{\}$
b_2	$\{\}$
b_3	$\{\}$
b_4	$\{\}$

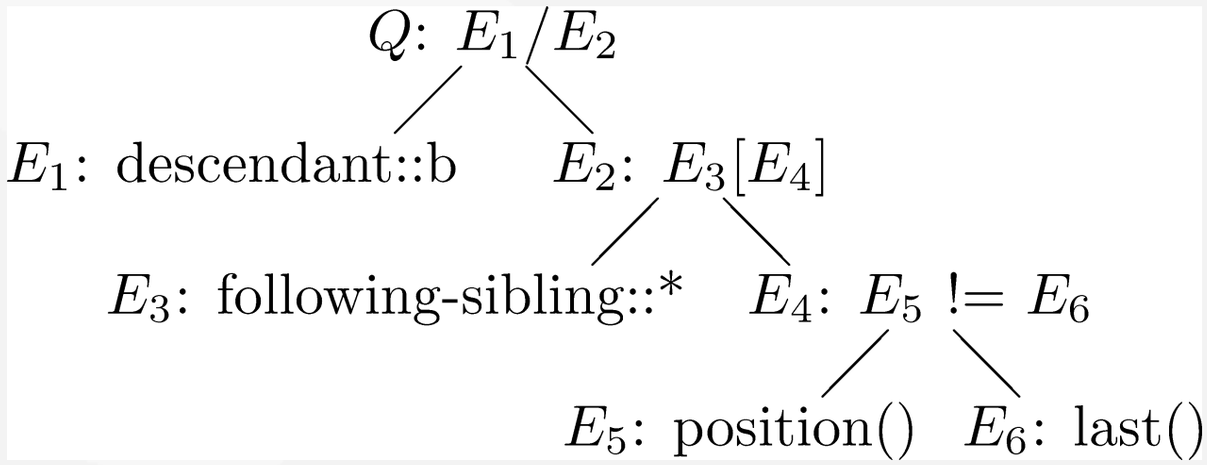
$\mathcal{E}_\uparrow[\![E_1]\!]$	
x	val
r	$\{b_1, b_2, b_3, b_4\}$
a	$\{b_1, b_2, b_3, b_4\}$
b_1	$\{\}$
b_2	$\{\}$
b_3	$\{\}$
b_4	$\{\}$

$\mathcal{E}_\uparrow[\![E_2]\!]$	
x	val
r	$\{\}$
a	$\{\}$
b_1	$\{b_2, b_3\}$
b_2	$\{b_3\}$
b_3	$\{\}$
b_4	$\{\}$

$\mathcal{E}_\uparrow[\![E_3]\!]$	
x	val
r	$\{\}$
a	$\{\}$
b_1	$\{b_2, b_3, b_4\}$
b_2	$\{b_3, b_4\}$
b_3	$\{b_4\}$
b_4	$\{\}$

Fig. 6. Context-value tables of Example 6.4.



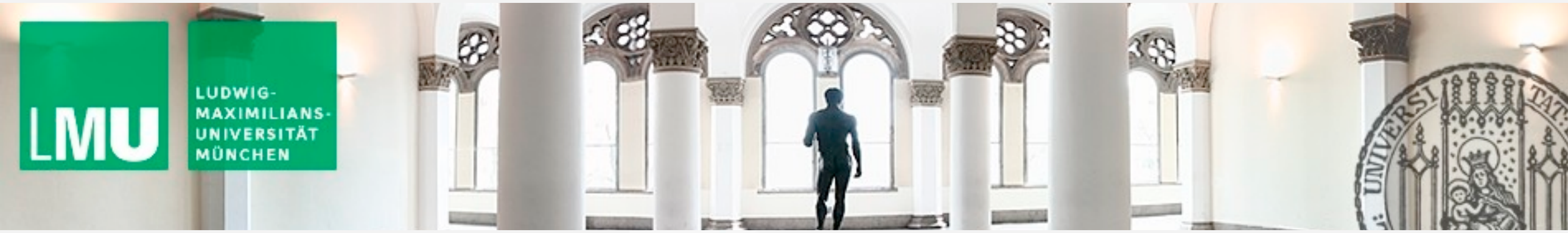


$$\mathcal{E}_\uparrow[\![E_5]\!] = \{\langle x,k,n,k\rangle \mid \langle x,k,n\rangle \in \mathbf{C}\}$$
$$\mathcal{E}_\uparrow[\![E_6]\!] = \{\langle x,k,n,n\rangle \mid \langle x,k,n\rangle \in \mathbf{C}\}$$
$$\mathcal{E}_\uparrow[\![E_4]\!] = \{\langle x,k,n,k \neq n\rangle \mid \langle x,k,n\rangle \in \mathbf{C}\}.$$

Polynomial XPath

Bottom-up with Tabling

- Result: **polynomial** (combined) complexity for XPath evaluation
  - bottom-up: $\mathbf{O(|D|^5 \cdot |Q|^2)}$ time and $\mathbf{O(|D|^4 \cdot |Q|^2)}$ space
    - why such large constants? context (^{3}) + concat, arithmetics (additional ^{2})
- Disadvantage: bottom-up processing considers too many nodes
  - no "context" in the query evaluation
    - e.g., `//a//b` considers all **b**'s in the document rather than only **b**'s in **a**'s
- Solution: top-down evaluation, but set-based
  - details: see exercises
- Navigational XPath: can be evaluated in **linear** time and space



3

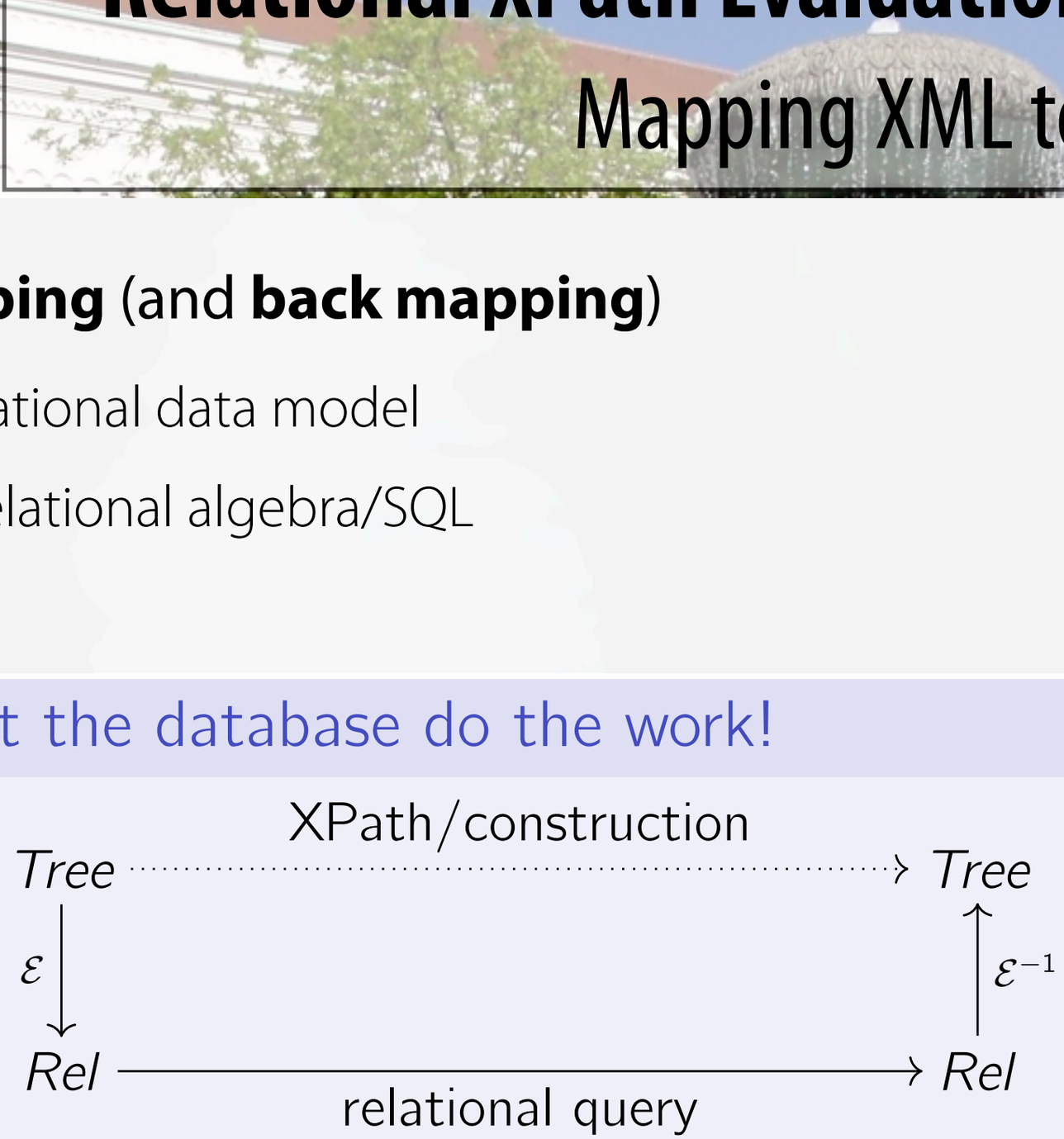
XPath Evaluation: Relational

Relational XPath Evaluation

Mapping XML to Databases

- Aim: Find **mapping** (and **back mapping**)
  - from XML to relational data model
  - from XPath to relational algebra/SQL
  - such that:

Our goal: let the database do the work!



  - nevertheless: labeling schemes can also benefit "native" XML storage

Hard requirements:

1. $\mathcal{E}$ is required to reflect **document order** and **node identity.**
   - *Otherwise*: cannot enforce XPath semantics, cannot support << and `is`, cannot support node construction.
2. $\mathcal{E}$ is required to encode the **XQuery DM node properties.**
   - *Otherwise*: cannot support XPath axes, cannot support XPath node tests, cannot support atomization, cannot support validation.
3. $\mathcal{E}$ is able to encode any well-formed **schema-less** XML fragment (*i.e.*, $\mathcal{E}$ is "**schema-oblivious**", see below).
   - *Otherwise*: cannot process non-validated XML documents, cannot support arbitrary node construction.

Soft requirements (primarily motivated by performance concerns):

4. **Data-bound operations** on trees (potentially delivering/copying lots of nodes) should map into efficient database operations.
   - *XPath location steps* (12 axes)
5. Principal, recurring **operations imposed by the XQuery semantics** should map into efficient database operations.
   - *Subtree traversal* (atomization, element construction, serialization).

Relational XPath Evaluation

Dead End: Large Object Blocks

- XML document is stored in a **CLOB** or **BLOB**
  - character or binary large object block
  - XML document is opaque to the database engine
    - a separate XML parser & XPath/XQuery evaluator are needed
  - **not** a relational encoding
- Nevertheless used in **SQL/XML** standard
  - roughly speaking "relational world" + "XML world"
  - fairly separate, originally not even the atomic types were compatible

Relational XPath Evaluation

Dead End: Schema-based Encoding

XML address database (excerpt)

```
<person>
  <name><first>John</first><last>Foo</last></name>
  <address><street>13 Main St</street>
           <zip>12345</zip><city>Miami</city>
  </address>
</person>
<person>
  <name><first>Erik</first><last>Bar</last></name>
  <address><street>42 Kings Rd</street>
           <zip>54321</zip><city>New York</city>
  </address>
</person>
```

Schema-based relational encoding: table person

id	first	last	street	zip	city
0	John	Foo	13 Main St	12345	Miami
1	Erik	Bar	42 Kings Rd	54321	New York

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Relational XPath Evaluation

Dead End: Schema-based Encoding

- observe for each element **type** what **sub**-elements it can contains
  - best done from DTD/XML Schema but can be discovered
  - **tailored to one specific schema**
- works reasonably well for **very regular XML data**
- fails for flexible schema (like HTML)
  - mixed content
  - support for horizontal axes (`following` etc.), no order support
  - updates that change the schema

Relational XPath Evaluation

Dead End: Schema-based Encoding

Irregular hierarchy

```
<a no="0">
  <b><c>X</c><c/></b>
</a>
<a no="1">
  <b><c>Y</c></b>
</a>
<a><b/></a>
<a no="3"/>
```

A relational encoding

id	@no	b
0	0	α
3	1	β
5	NULL^a	γ
6	3	NULL^b

id	b	c
1	α	X
2	α	NULL^c
4	β	Y

Issues:

- Number of encoding tables depends on nesting depth.
- Empty element `c` encoded by NULL^c, empty element `b` encoded by absence of γ (will need *outer join* on column `b`).
- NULL^a encodes absence of attribute, NULL^b encodes absence of element.
- Document order/identity of `b` elements only implicit.

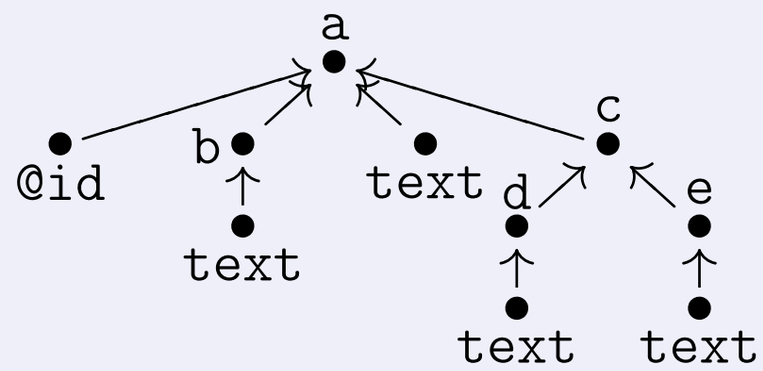
Relational XPath Evaluation

Dead End: Adjacency-based Encoding

Adjacency-based encoding of XML fragments

```
<a id="0">
   <b>fo</b>o
   <c>
     <d>b</d><e>ar</e>
   </c>
</a>
```

≡



Resulting relational encoding

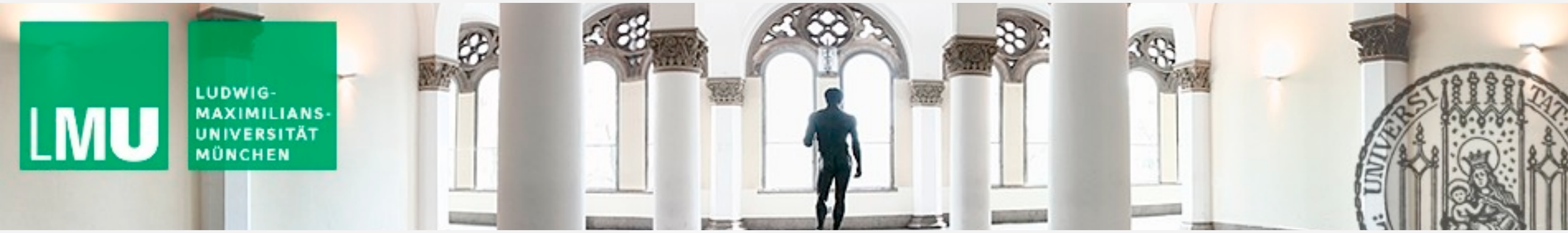
id	parent	tag	text	val
0	NULL	a	NULL	NULL
1	0	@id	NULL	"0"
2	0	b	NULL	NULL
3	2	NULL	"fo"	NULL
4	0	NULL	"o"	NULL
5	0	c	NULL	NULL
⋮				

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Relational XPath Evaluation

Dead End: Adjacency-based Encoding

- Idea: store the **parent `id`** for each node
  - looks promising for document order, node identity, reconstruction
  - but: `descendant`, `ancestor`, `following`, `preceding`
    - require **unbounded** number of joins → SQL recursion
    - horribly **inefficient**
- what we need is a more "**expressive**" label than just **parent `id`**
  - such that we can **evaluate** (all/important) **XPath** axes given only the label
    - ideally: constant or logarithmic
  - such that the **iteration** over all related nodes for each XPath axes is efficient
    - ideally: linear
  - such that the **space** needed for the labels is linear or at worst O(*n* log *n*)



4

Labeling Schemes for XML

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Labeling Schemes for XML

Overview

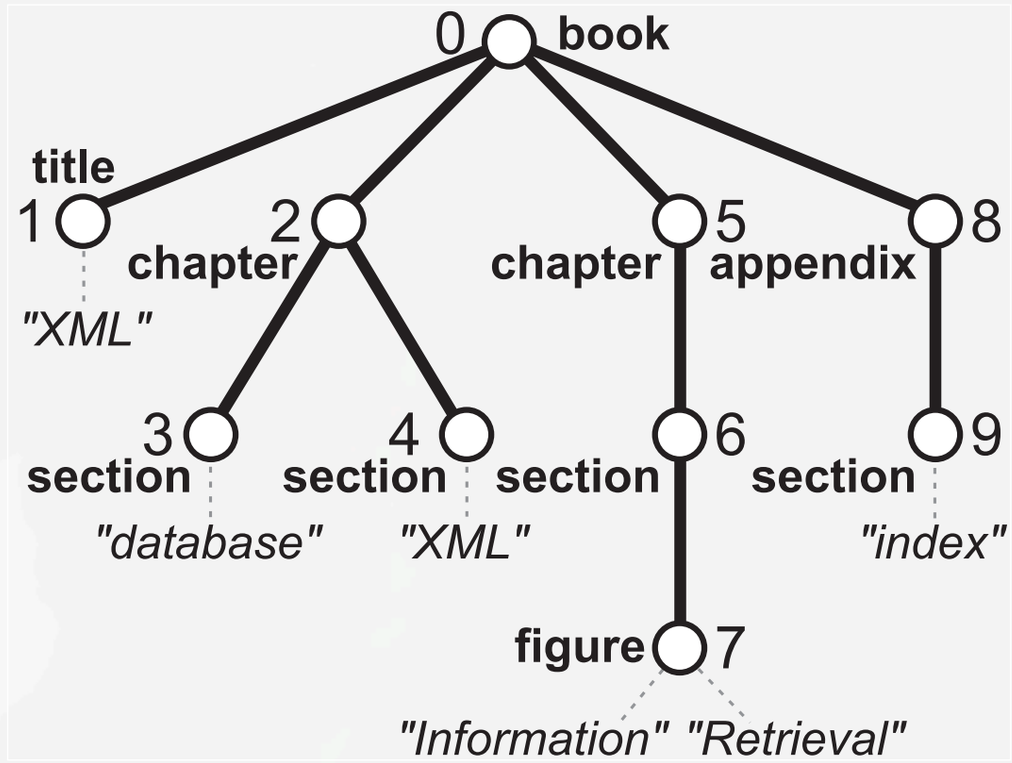
- Aim for each XPath axis (but at least `child`, `descendant`):
  - given two node labels **constant** or **logarithmic test** if related by axis
  - given a node label **linear** or **polylinear iteration** over nodes related by axis
  - size of a node label should be **constant** or **logarithmic**
- Given an XML document
  - labeling should be **computable** in **polynomial** (better: **linear**) time
- Three **main classes** of labeling schemes
  - **interval**-based labelings: numeric labels & intervals or ranges to describe the related nodes per axis
  - **prefix**- or path-based labelings: labels represent root-to-leaf paths
  - **arithmetic** or multiplicative labelings: arithmetic relations between labels

Labeling Schemes for XML

Sample Document

```
<?xml version="1.0" ?>
<book>
  <title>XML</title>
  <chapter>
    <section>database</section>
    <section>XML</section>
  </chapter>
  <chapter>
    <section>
      <figure>Information Retrieval</figure>
    </section>
  </chapter>
  <appendix>
    <section>index</section>
  </appendix>
</book>
```

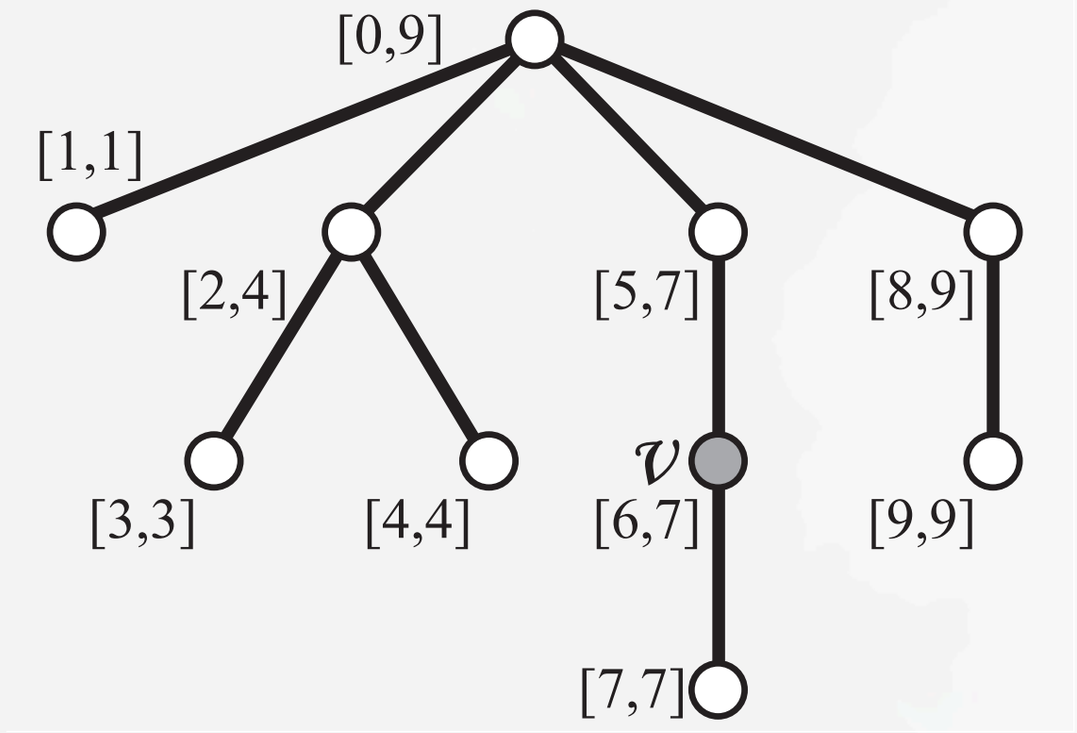
a. XML serialization



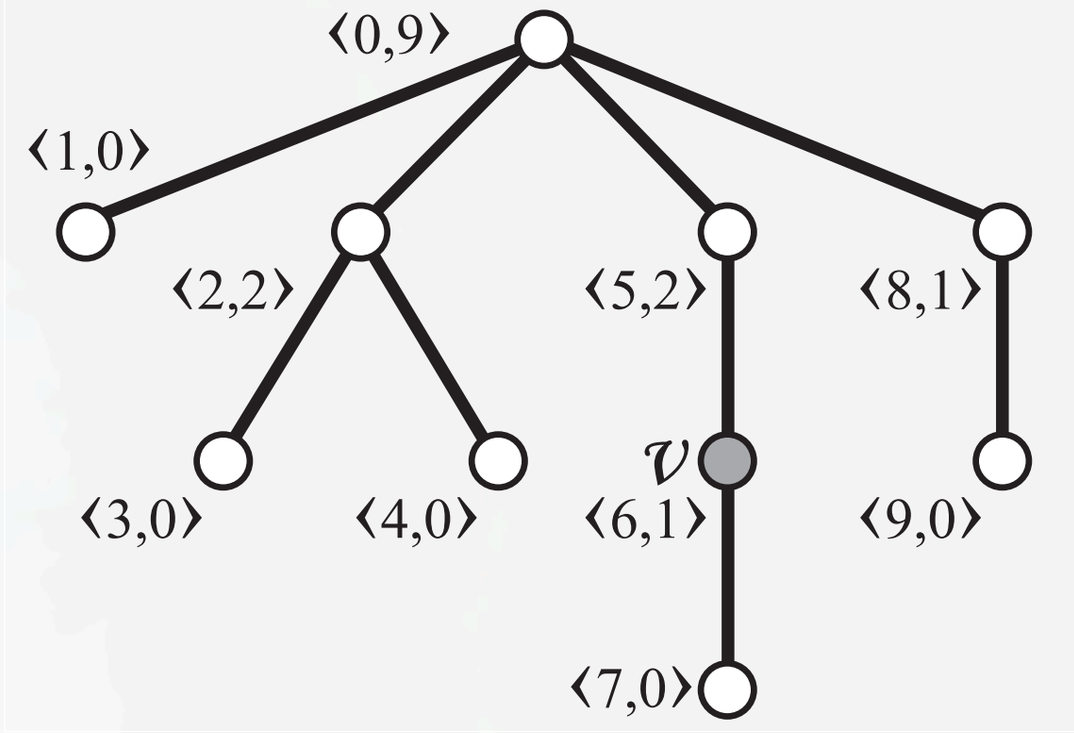
b. document tree *D* with preorder labels

Labeling Schemes for XML

Interval Labelings I



a. Pre/Max



b. Order/Size

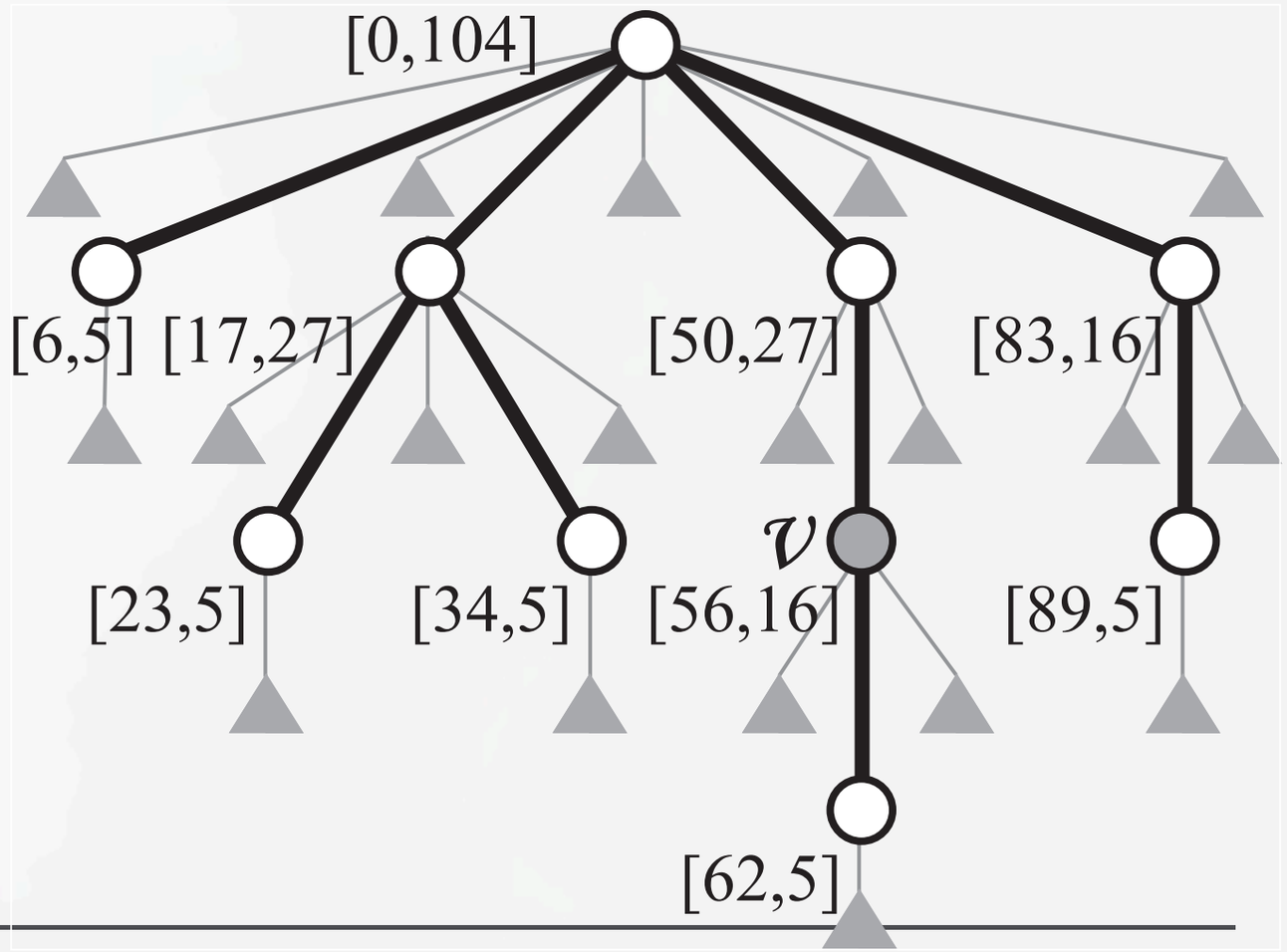
Label: [pre, max(pre of desc)]
desc(*v*,*w*) = pre(*v*) ≤ pre(*w*) ≤ max(*v*)
foll(*v*,*w*) = pre(*w*) > max(*v*)
Updates: both change

Label: [pre, number of desc)]
desc(*v*,*w*) = pre(*v*) ≤ pre(*w*) ≤ pre(*v*) + size(*v*)
foll(*v*,*w*) = pre(*w*) > pre(*v*) + size(*v*)
Updates: only pre changes

Labeling Schemes for XML

Interval Labelings II

- both pre/max and order/size need **linear** time for each update
- improvement: leave virtual "**gaps**" for inserting nodes
  - here: virtual gaps represented as gray **rectangles** of size **5**
  - label with **order/size**
    - but as in the "virtual" tree
- called: "extended preorder"



Labeling Schemes for XML

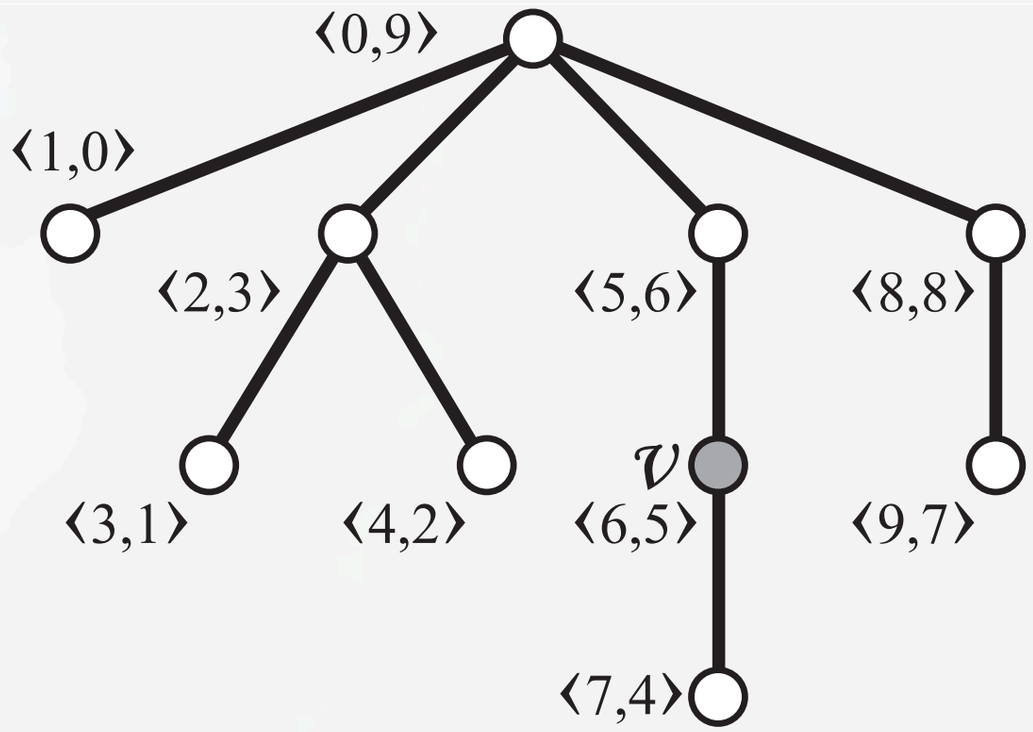
Interval Labelings III

- **Pre-/Post-Encoding**
  - very close to stream of XML start and end tags
  - allows to compute all structural XPath axes

Label: [pre, post]

desc(v,w) = pre(v) ≤ pre(w) ∧ post(w) ≤ post(v)

foll(v,w) = pre(w) > post(v)



Labeling Schemes for XML

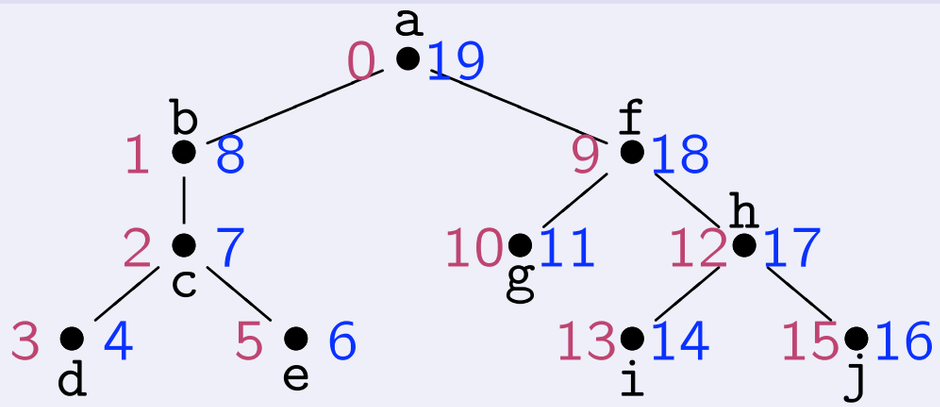
Start/End = *Stretched* Pre/Post Encoding

"Stretched" (or coupled) *preorder*/*postorder* ranks

Perform a depth-first, left-to-right traversal of the skeleton tree.
Maintain counter *rank* (initally 0).

1. Whenever a node v is visited first, assign $pre(v) \leftarrow rank$; increment *rank*.
2. When v is visited last, assign $post(v) \leftarrow rank$; increment *rank*.

Example



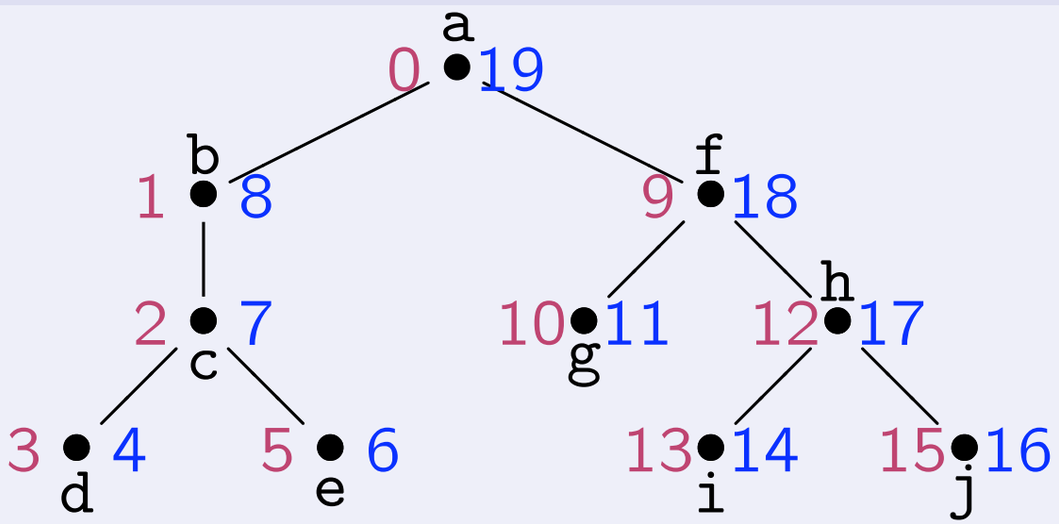
This encoding is also known as "start–end" numbering.

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

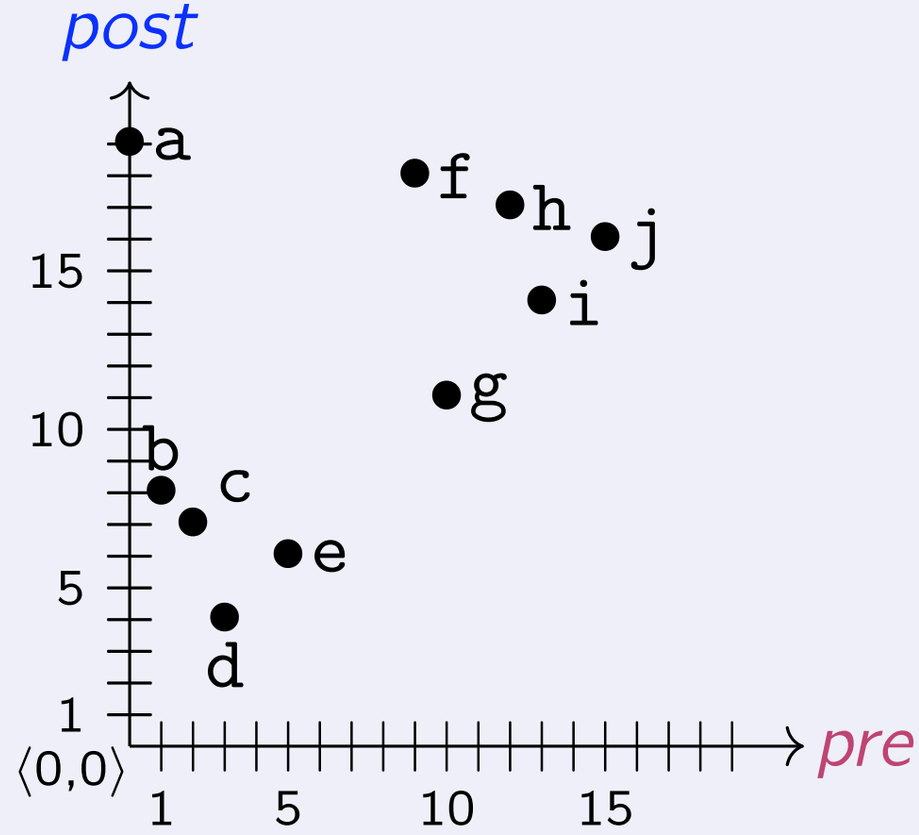
Labeling Schemes for XML

Start/End = *Stretched* Pre/Post Encoding

start-end numbering



Stretched *pre*/*post* plane



Node identifiers of bit width n encode 2^{n-1} nodes.

Labeling Schemes for XML

Start/End = *Stretched* Pre/Post Encoding

Node distribution in the stretched *pre*/*post* plane has interesting properties:

- The axes *window*(·) predicates continue to work as before.

Further:

Characterization of `descendant` axis

Node v is selected by c/`descendant::node()`, iff

$$pre(v) \in (pre(c), post(c)) \quad \textbf{or} \quad post(v) \in (pre(c), post(c))$$

Subtree size (exact, no estimation)

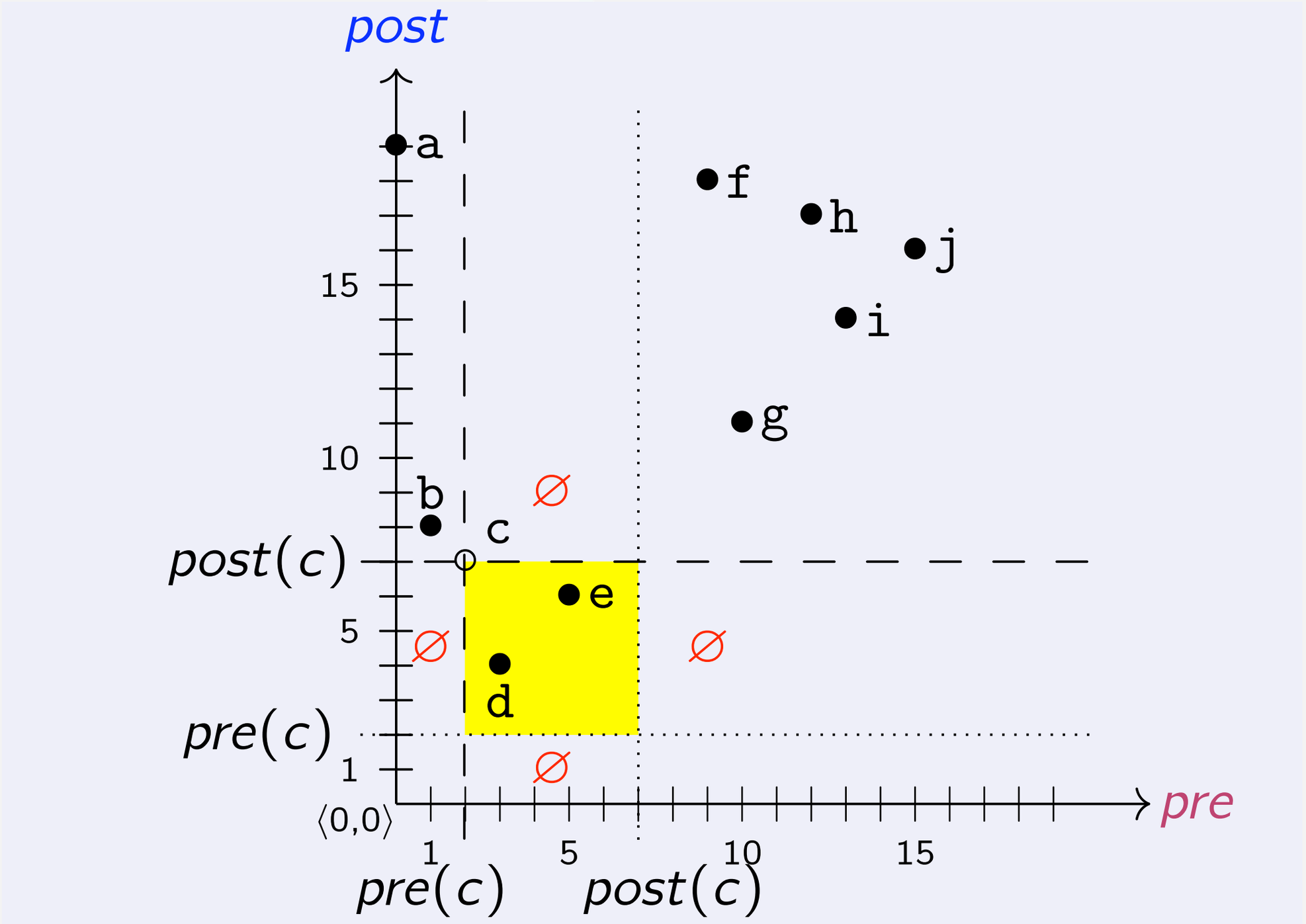
For any node v:

$$size(v) = 1/2 \cdot (post(v) - pre(v) - 1)$$

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Labeling Schemes for XML

Start/End = *Stretched* Pre/Post Encoding



LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Labeling Schemes for XML

Start/End = *Stretched* Pre/Post Encoding

- Summary: start/end encoding
  - **label** size is larger than for pre/post-encoding
  - but: **windows** for major XPath axes are much **tighter**
  - size of **subtrees** can be **precisely** computed
  - update behavior is unchanged
- Essentially we can use the **same translation** from XPath for SQL
  - for start/end as for pre/post
  - see following slides

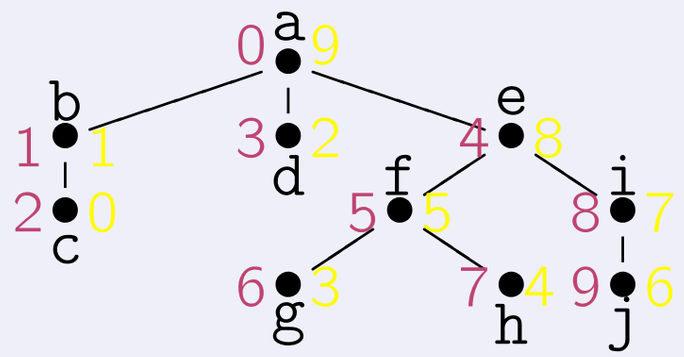
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Relational XPath Evaluation

XPath Accelerator Pre/Post-Encoding

XML fragment f and its skeleton tree

```
<a>
  <b>c</b>
  <!--d-->
  <e><f><g/><?h?></f>
     <i>j</i>
  </e>
</a>
```



Pre/*post* encoding of f: table `accel`

pre	*post*	*par*	*kind*	*tag*	*text*
0	9	NULL	elem	a	NULL
1	1	0	elem	b	NULL
2	0	1	text	NULL	c
3	2	0	com	NULL	d
4	8	0	elem	e	NULL
5	5	4	elem	f	NULL
6	3	5	elem	g	NULL
7	4	5	pi	NULL	h
8	7	4	elem	i	NULL
9	6	8	text	NULL	j

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Relational XPath Evaluation

Translating Relation Steps

Evaluate an XPath location step by means of a **window query** on the *pre*/*post* plane.

1. Table `accel` encodes an XML fragment,
2. table `context` encodes the **context node sequence** (in XPath accelerator encoding).

like **context value table**

XPath location step (axis α) $\Rightarrow$ SQL window query

$$
\begin{array}{rl}
\texttt{SELECT} & \texttt{DISTINCT } v'.* \\
\texttt{FROM} & \texttt{context } v, \texttt{accel } v' \\
\texttt{WHERE} & v' \texttt{ INSIDE } \mathit{window}(\alpha, v) \\
\texttt{ORDER BY} & v'.\mathit{pre}
\end{array}
$$

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Relational XPath Evaluation

From Axes to Pre/Post-Windows

Window def's for axis α, name test t ($* = \textit{don't care}$)

Axis α	Query window $\textit{window}(\alpha::t, v)$				
	pre	*post*	*par*	*kind*	*tag*
`child`	$\langle\ (v.pre, *)$	$(*, v.post)$	$v.pre$	*elem*	$t\ \rangle$
`descendant`	$\langle\ (v.pre, *)$	$(*, v.post)$	$*$	*elem*	$t\ \rangle$
`descendant-or-self`	$\langle\ [v.pre, *)$	$(*, v.post]$	$*$	*elem*	$t\ \rangle$
`parent`	$\langle\ v.par$	$(v.post, *)$	$*$	*elem*	$t\ \rangle$
`ancestor`	$\langle\ (*, v.pre)$	$(v.post, *)$	$*$	*elem*	$t\ \rangle$
`ancestor-or-self`	$\langle\ (*, v.pre]$	$[v.post, *)$	$*$	*elem*	$t\ \rangle$
`following`	$\langle\ (v.pre, *)$	$(v.post, *)$	$*$	*elem*	$t\ \rangle$
`preceding`	$\langle\ (*, v.pre)$	$(*, v.post)$	$*$	*elem*	$t\ \rangle$
`following-sibling`	$\langle\ (v.pre, *)$	$(v.post, *)$	$v.par$	*elem*	$t\ \rangle$
`preceding-sibling`	$\langle\ (*, v.pre)$	$(*, v.post)$	$v.par$	*elem*	$t\ \rangle$

Relational XPath Evaluation
From Pre/Post-Window to SQL Predicate

`descendant::foo`, context node v

$$v' \texttt{ INSIDE } \langle (v.pre, *), (*, v.post), *, elem, \texttt{foo} \rangle$$
$$\equiv$$
$$v'.pre > v.pre \texttt{ AND } v'.post < v.post \texttt{ AND}$$
$$v'.kind = elem \texttt{ AND } v'.tag = \texttt{foo}$$

`ancestor-or-self::*`, context node v

$$v' \texttt{ INSIDE } \langle (*, v.pre], [v.post, *), *, elem, * \rangle$$
$$\equiv$$
$$v'.pre <= v.pre \texttt{ AND } v'.post >= v.post \texttt{ AND}$$
$$v'.kind = elem$$

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Summary

- you **need** to know
  - **complexity** of XPath query evaluation
    - how to achieve **polynomial** complexity
  - how to **map XML** to relational databases
    - weaknesses of CLOB/BLOB-based, schema-based, adjacency-based storage
    - what is a **labeling scheme**, what is a **good** labeling scheme
    - classes of labeling schemes
  - details of **pre/post-encoding**
  - details of **start/end-encoding**
  - translation of XPath to SQL under pre/post- or start/end-encoding.