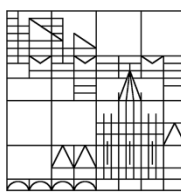


XQuery Full Text Implementation in BaseX

XSym/VLDB 2009

Christian Grün, Sebastian Gath,
Alexander Holupirek, Marc H. Scholl

Database and Information Systems Group
University of Konstanz, Germany



Motivation

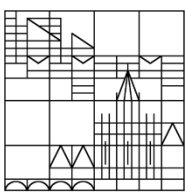
XQUERY FULL TEXT

XQuery/XPath Full Text 1.0

- upcoming W3C Recommendation for content-based XML queries
- brings DB and IR world together
- first implementations available (Qizx, MXQuery, xDB, BaseX)

Challenges

- large text corpora/XML instances
 - complete embedding in XQuery language
 - classical retrieval features: stemming, thesaurus, stop words
- all features need to be supported, yet performance is essential



Queries **XQUERY FULL TEXT**

Document-based location path with predicate

```
//book[title ftcontains 'Crime and Punishment']
```

Optional filters and options

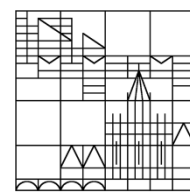
```
//book[section ftcontains ('heritage' ftand 'claim'  
    window 10 words) language 'en' with stemming]/title
```

Queries without document reference

```
'a b c' ftcontains 'b'
```

Dynamic item values

```
func:merge($a) ftcontains { func:stem($a), $b, $c }
```

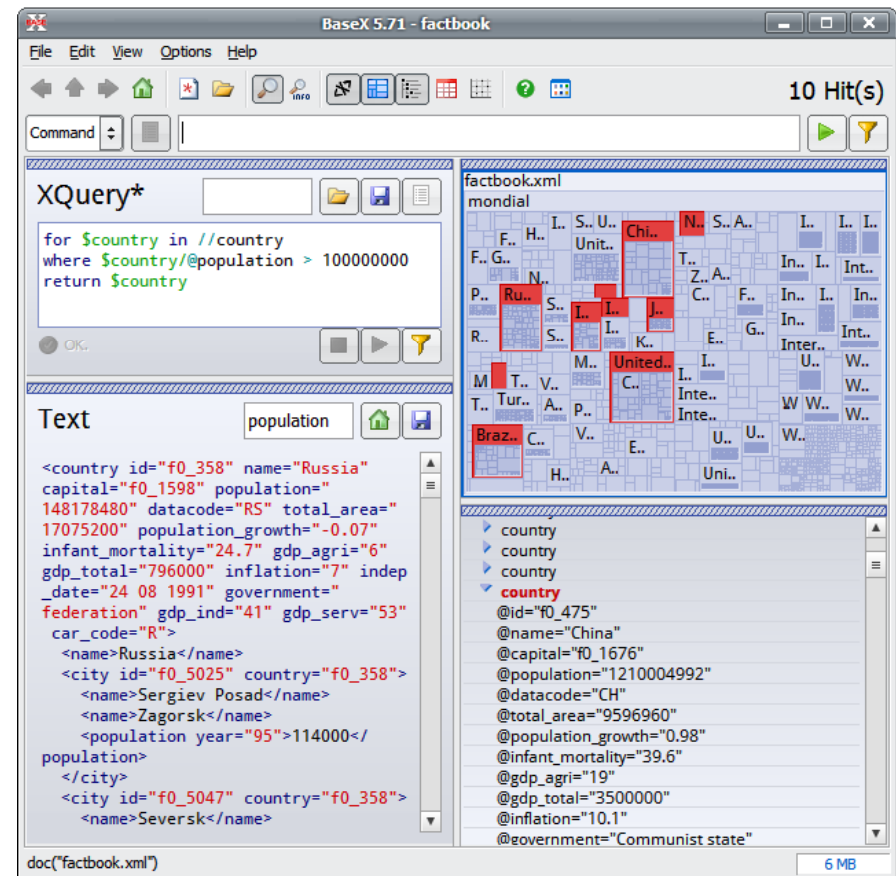


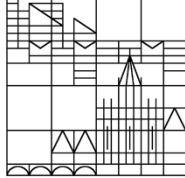
BaseX

XQUERY FULL TEXT

Native XML Database and XQuery Processor

- first complete XQuery Full Text implementation
- high XQuery conformance (99.9%)
- various index structures:
names, paths, values, full text
- tight backend/frontend coupling,
real-time querying
- open source (BSD) since 03/07



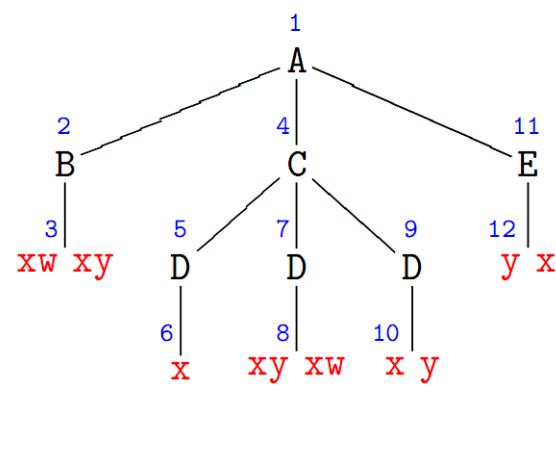


Storage XQUERY FULL TEXT

Document Storage

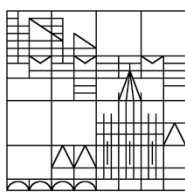
- inspired by XPath Accelerator¹ and MonetDB/XQuery
- flat, compressed table storage, using pre/dist/size encoding:

```
<A>
  <B>xw xy</B>
  <C>
    <D>x</D>
    <D>xy xw</D>
    <D>x y</D>
  </C>
  <E>y x</E>
</A>
```



pre	dist	size	data
1	1	11	A
2	1	1	B
3	1	0	xw xy
4	3	6	C
5	1	1	D
6	1	0	x
7	3	1	D
8	1	0	xy xw
9	5	1	D
10	1	0	x y
11	10	1	E
12	1	0	y x

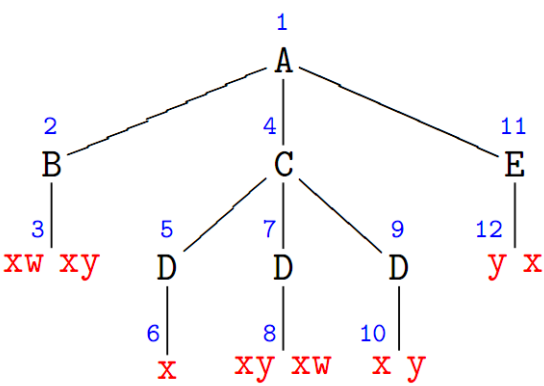
¹ Torsten Grust, Accelerating XPath Location Steps. SIGMOD 2002



Storage XQUERY FULL TEXT

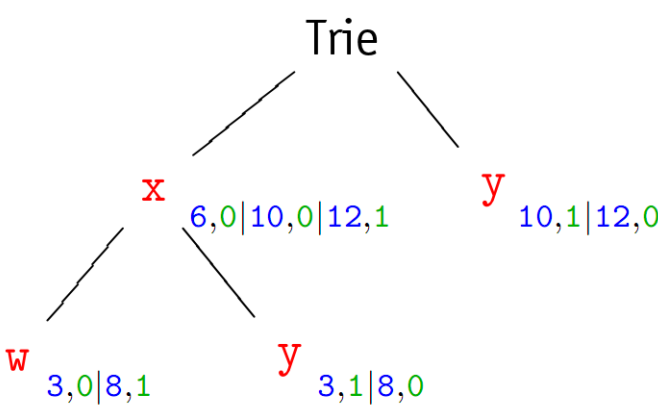
Indexes

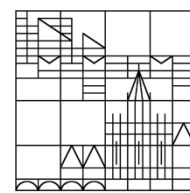
- names (tags, attribute names)
- paths (unique location paths)
- values (texts, attribute values)



Full Text Index

- Compressed Trie
- node: **characters** and **pre**, **pos** value pairs
- value pairs are sorted
- essential for pipelined evaluation





Evaluation

XQUERY FULL TEXT

Sequential Scan

- performs the predicate test for each location path
- touches all addressed nodes at least once

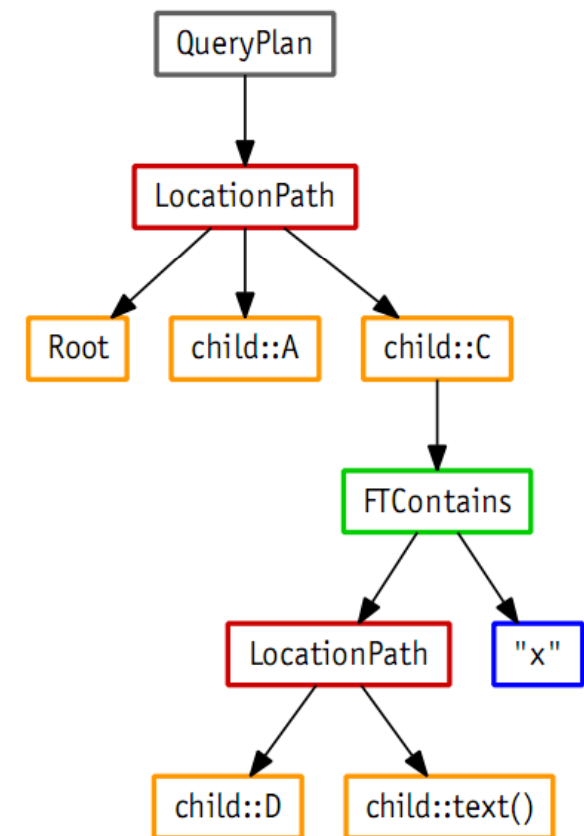
`/A/C[D/text() ftcontains "x"]`

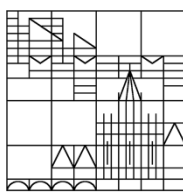
Index-based processing

- performs the predicate test first
- traverses the inverted path for all index items

Hybrid Approach

- combination of sequential and index-based processing





Evaluation: Index-Based

Indexing on document level

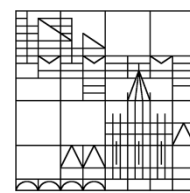
- popular approach in relational databases
- no performance boost for large documents

Indexing of location paths

- simple queries with fixed path can be easily sped up
- does not work for nested/more complex queries

XQuery Index Functions

- allows for explicit index calls
- no benefit for internal query optimization



Evaluation: Index-Based

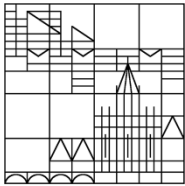
Dynamic Approach

- all text nodes are indexed
- predicates with `ftcontains` are analyzed for index access
- costs are estimated for each index access
- cheapest predicates are rewritten to index operators
- remaining location paths are inverted (utilizing the XPath Symmetries²)

Advantages

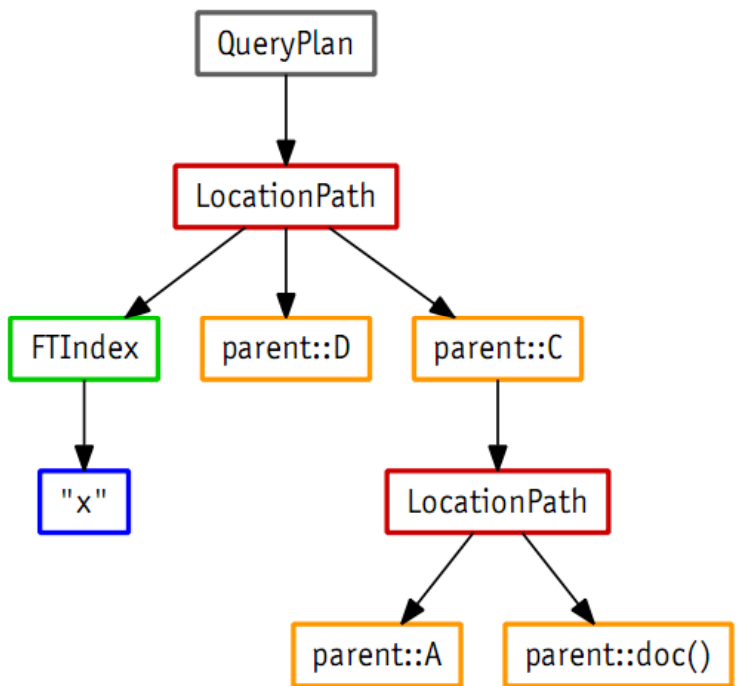
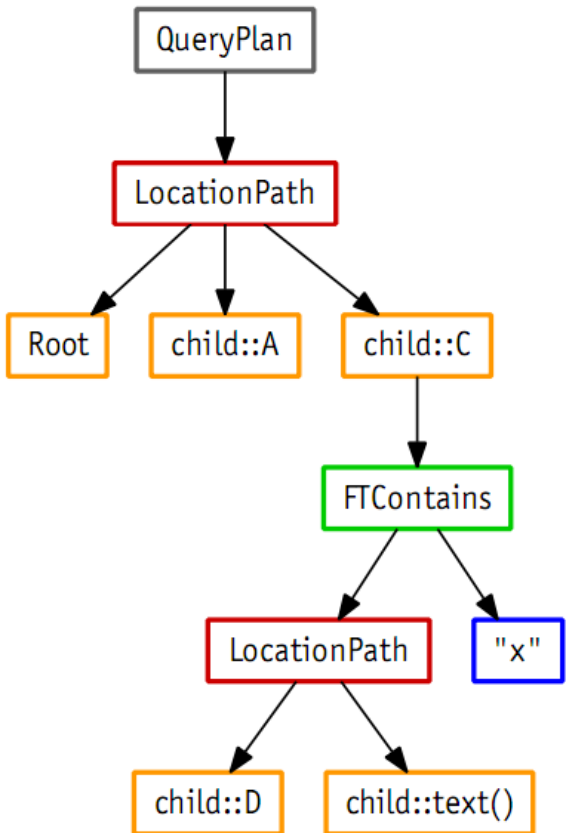
- + many queries with nested/complex location paths can be optimized
- + query writing and query optimization are uncoupled

² Dan Olteanu et al., XPath: Looking Forward. XMLDM Workshop 2002

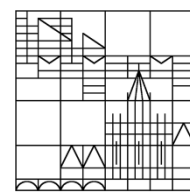


Evaluation: Index-Based

`/A/C[D/text() ftcontains "x"]`

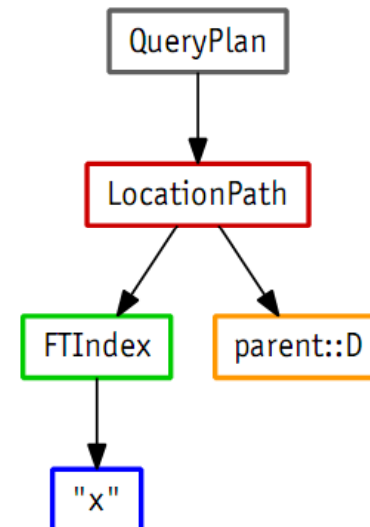
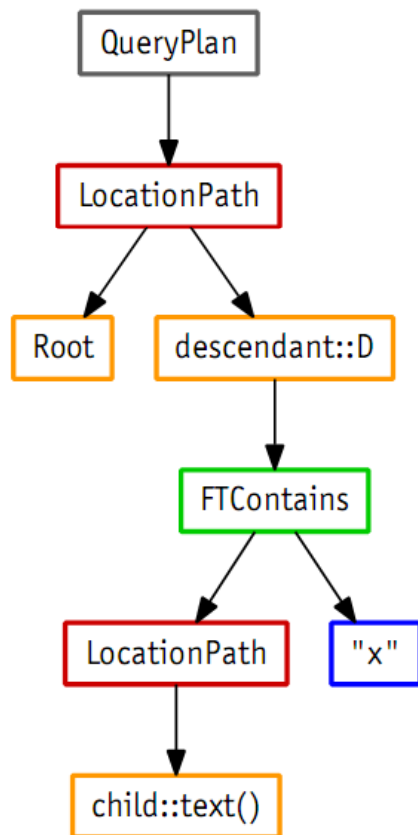


`FTIndex("x")/parent::D/parent::C`
`[parent::A/parent::doc()]`

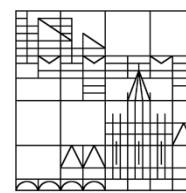


Evaluation: Index-Based

`/descendant::D[text() ftcontains "x"]`

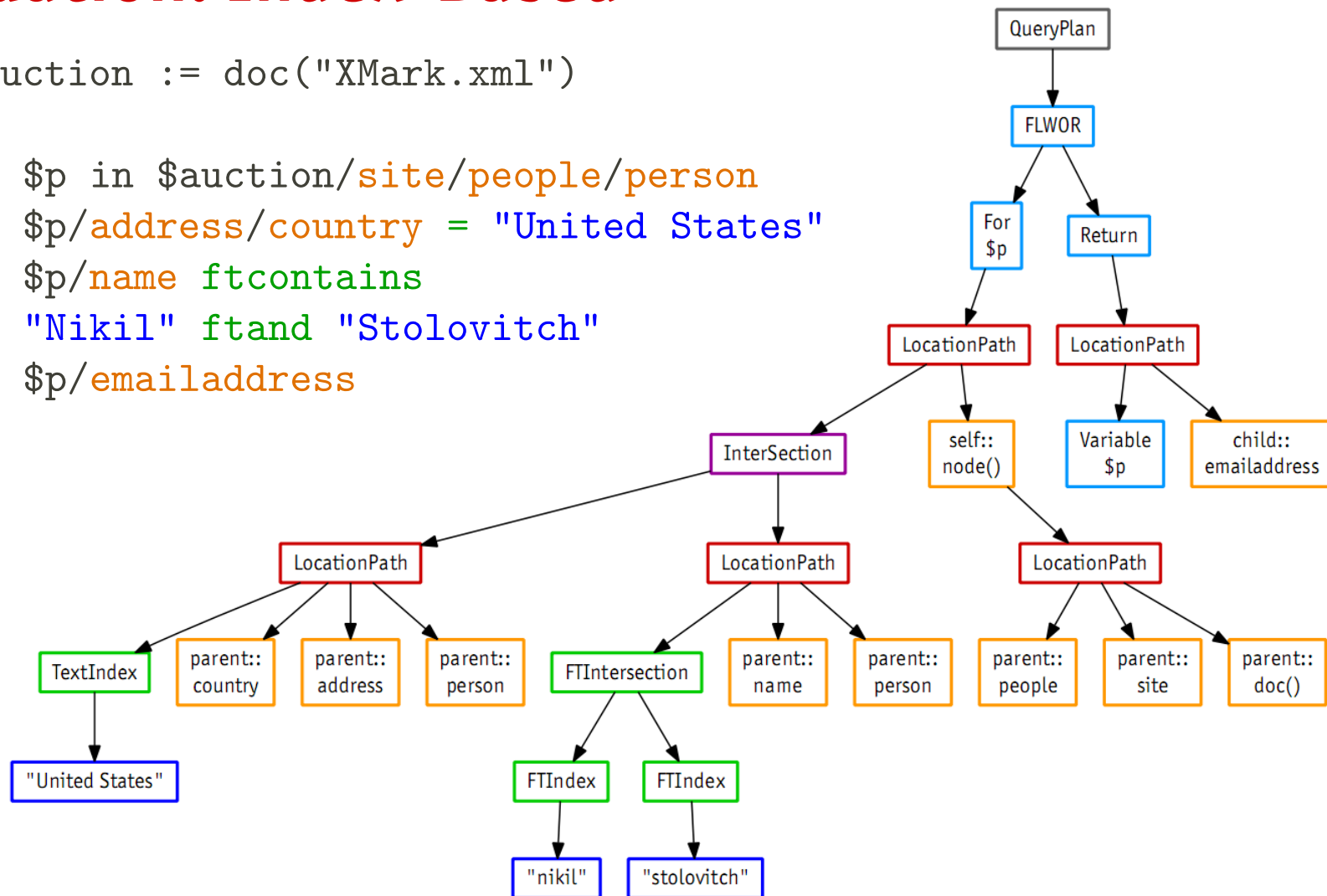


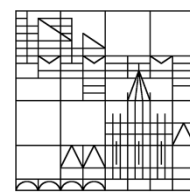
`FTIndex("x")/parent::D`



Evaluation: Index-Based

```
let $auction := doc("XMark.xml")
return
  for $p in $auction/site/people/person
  where $p/address/country = "United States"
  and $p/name ftcontains
    "Nikil" ftand "Stolovitch"
  return $p/emailaddress
```

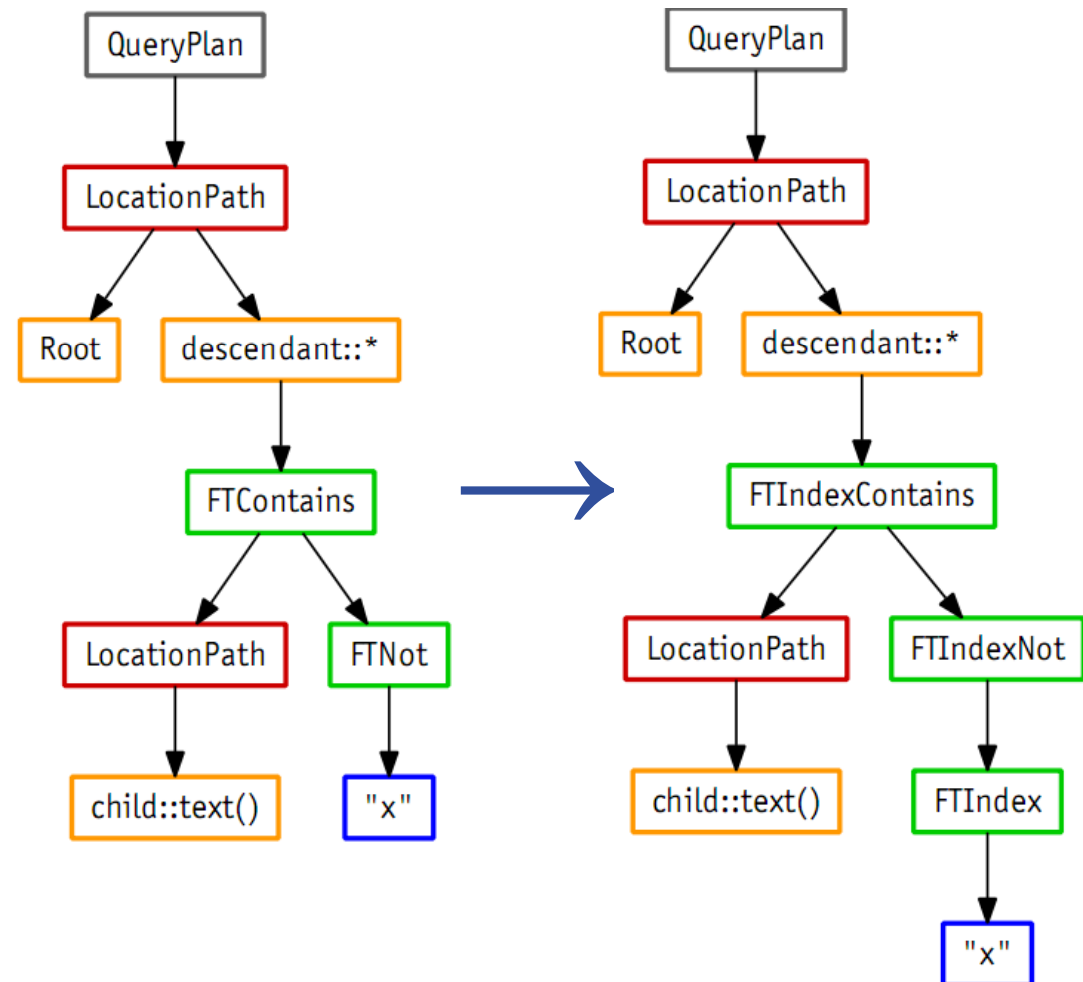


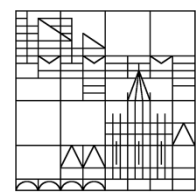


Evaluation: Hybrid

Hybrid Approach

- the `ftnot` operator cannot be processed by only using the index
- yet, index can be applied to avoid tokenization of all text nodes
- optimized plan combines seq. scan and index access
- sortedness of nodes and index results leads to linear costs





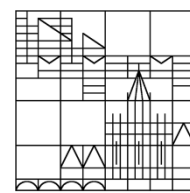
Evaluation: Pipelining

Iterative/pipelined Evaluation

- items are processed one-by-one
- constant memory consumption
- most efficient if large results are reduced to small, final result sets

Index Access

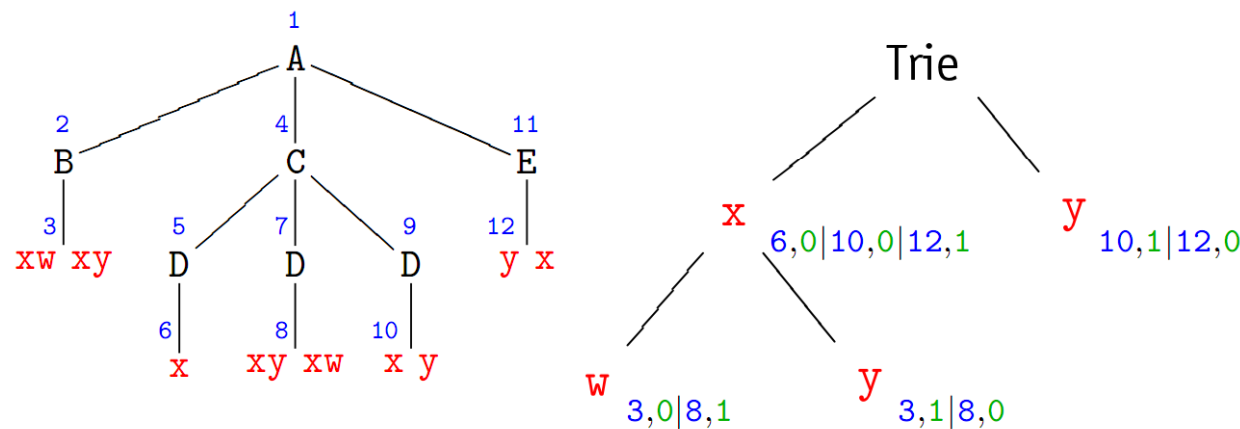
- all XQFT operators can be processed in an iterative manner
- a pipelined index operator returns single items
- this way, the same full-text operators can be applied on both sequential and index-based processing
- again, the sortedness of index results avoids pipeline blocking



Evaluation: Pipelining

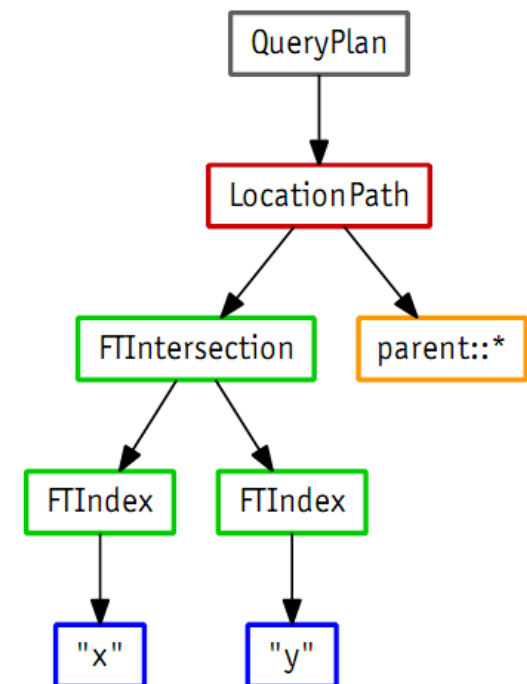
Index-based evaluation of `ftand`

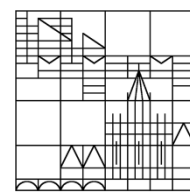
- FTIntersection operator merges index results:



- first argument call delivers value pairs `[6,0]` and `[10,1]`
- `[6,0]` is skipped, `[10,0]` and `[10,1]` are merged & returned
- finally, `[12,1]` and `[12,0]` are merged & returned

```
//*[text() ftcontains
    "x" ftand "y"]
```

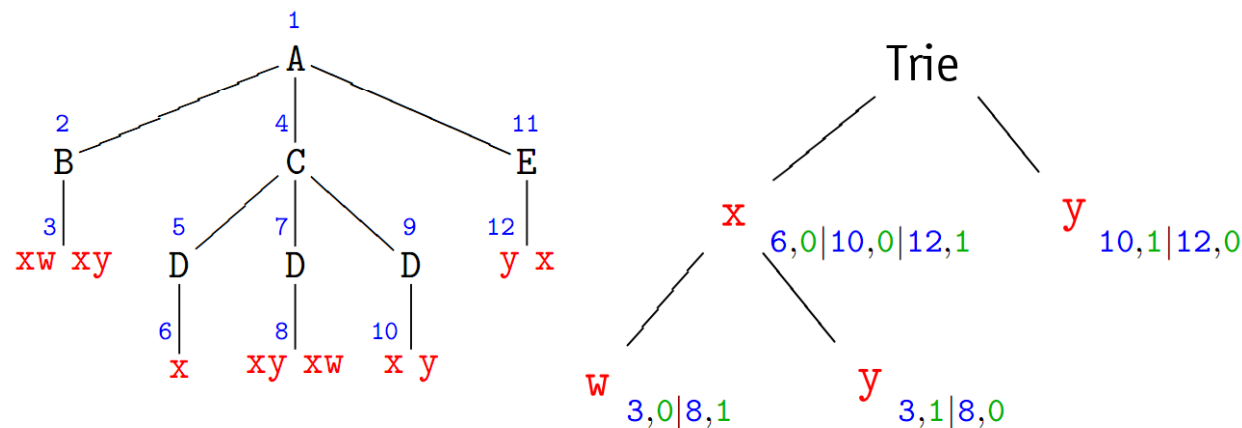




Evaluation: Pipelining

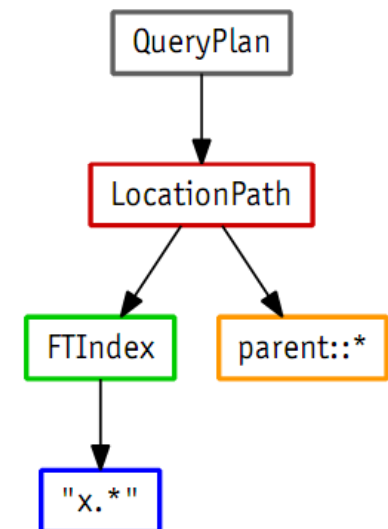
Index-based evaluation of wildcards

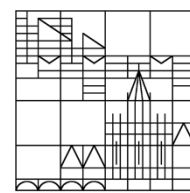
- Wildcards results are merged by FTIndex (ftor works similar):



- [3,0] and [3,1] are merged and returned
- next results are: [6,0], [8,0 | 8,1], [10,0], [12,1]

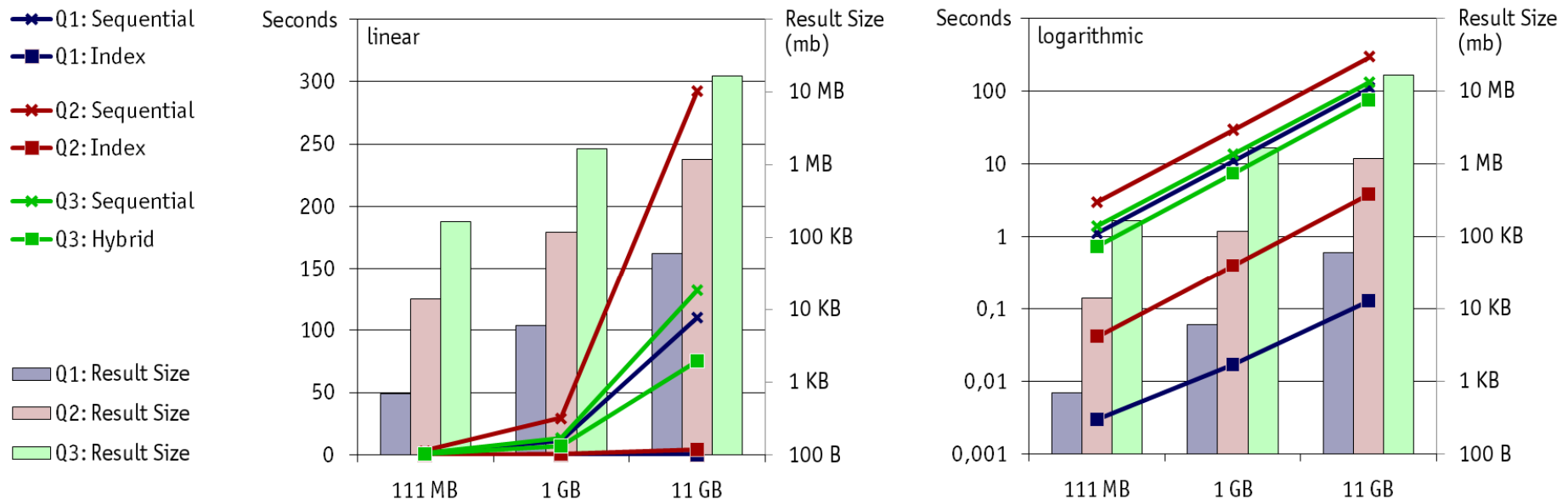
```
//*[text() ftcontains  
"x.*" with wildcards]
```





Performance

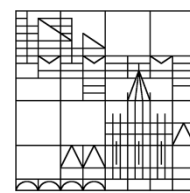
XQUERY FULL TEXT



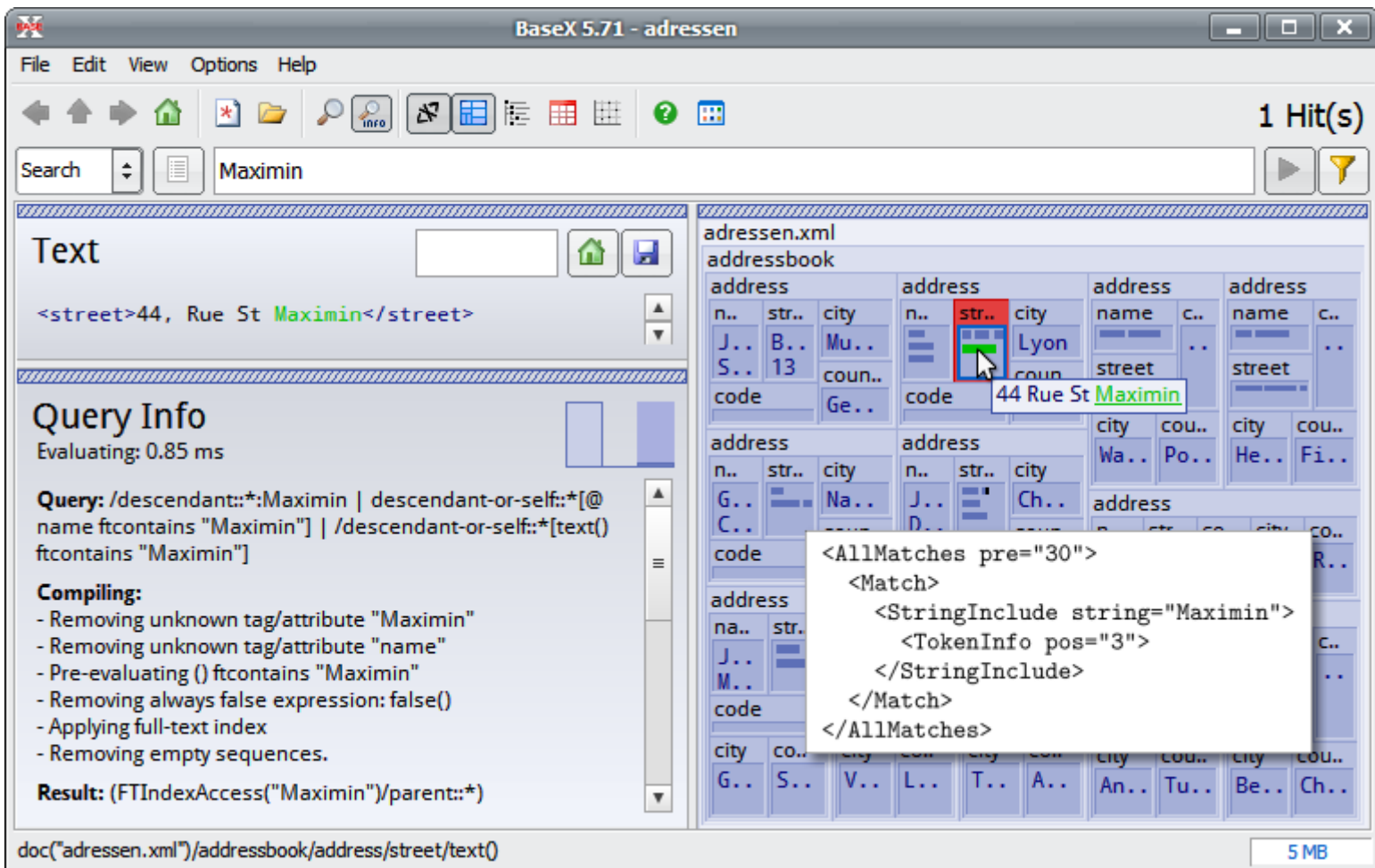
Q1: `doc("xmark")//keyword[text() ftcontains "barrel"]`

Q2: `for $mail in doc("xmark")/site/regions/*/item/mailbox/mail
where $mail//text/text() ftcontains "seeking.*" with wildcards
return $mail/from`

Q3: `for $item in doc("xmark")/site/regions/*/item
where $item//listitem/text/text() ftcontains ftnot "preventions"
return <result>{ $item/location/text() }</result>`



Frontend XQUERY FULL TEXT



The screenshot shows the BaseX 5.7.1 - adressen application window. The search bar contains 'Maximin' and shows '1 Hit(s)'. The main window displays the XML document 'adressen.xml' with a table view of address data. A tooltip is visible over the 'str..' column, showing the full text '44 Rue St Maximin'. The left sidebar contains a 'Text' editor with the XML snippet '<street>44, Rue St Maximin</street>' and a 'Query Info' section.

Text

```
<street>44, Rue St Maximin</street>
```

Query Info
Evaluating: 0.85 ms

Query: /descendant::*:Maximin | descendant-or-self::*[@name ftcontains "Maximin"] | /descendant-or-self::*[text() ftcontains "Maximin"]

Compiling:

- Removing unknown tag/attribute "Maximin"
- Removing unknown tag/attribute "name"
- Pre-evaluating () ftcontains "Maximin"
- Removing always false expression: false()
- Applying full-text index
- Removing empty sequences.

Result: (FTIndexAccess("Maximin")/parent::*)

XML Document: adressen.xml

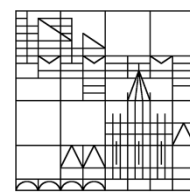
address	city	code
44 Rue St Maximin	Lyon	

Tooltip:

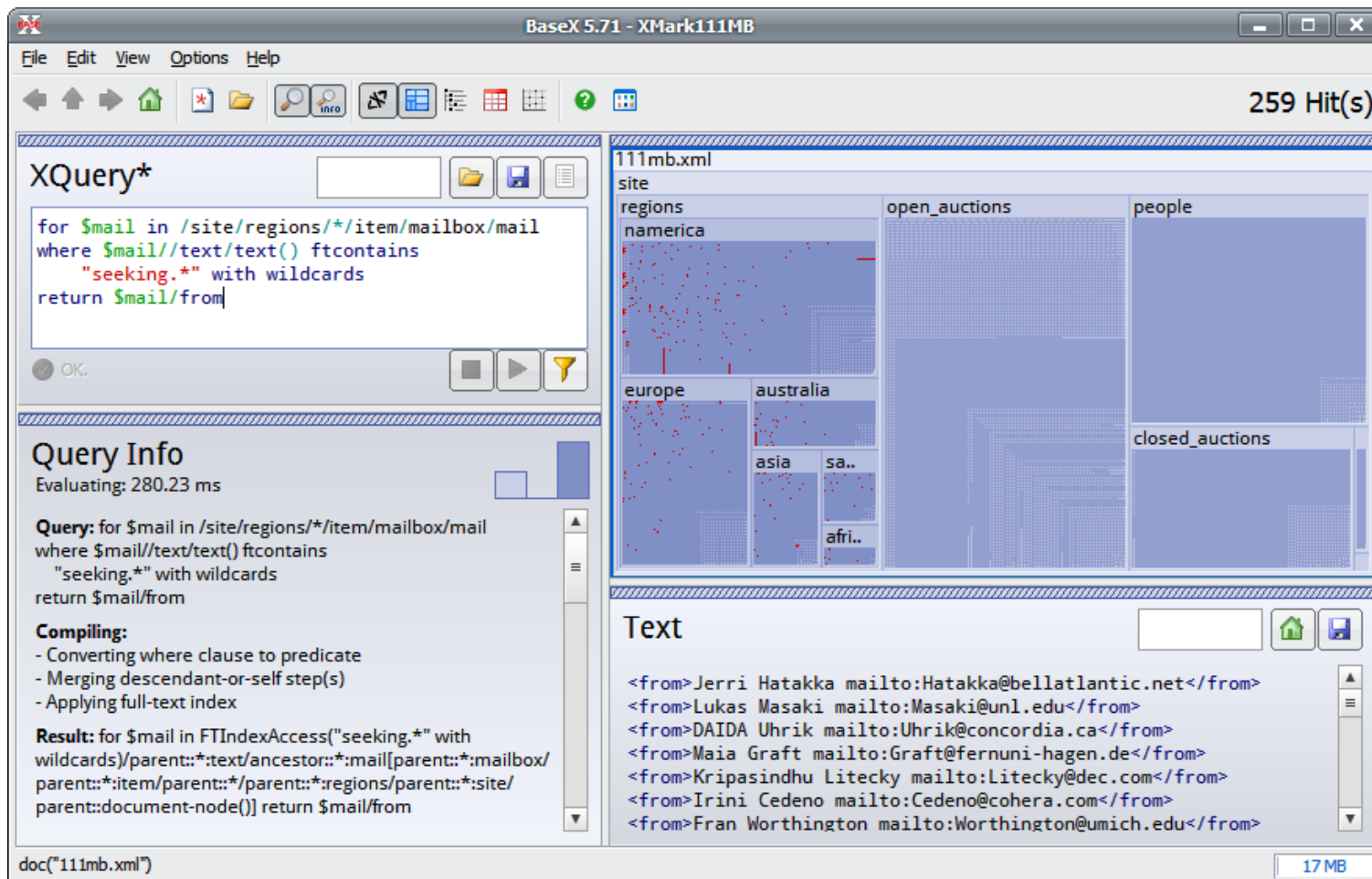
```
<AllMatches pre="30">
  <Match>
    <StringInclude string="Maximin">
      <TokenInfo pos="3">
    </StringInclude>
  </Match>
</AllMatches>
```

doc("adressen.xml")/addressbook/address/street/text()

5 MB



Frontend XQUERY FULL TEXT



The screenshot shows the BaseX 5.71 - XMark111MB application window. The interface is divided into several sections:

- Top Bar:** Displays the file name "111mb.xml" and the number of hits "259 Hit(s)".
- XQuery* Section:** Contains the XQuery code:


```
for $mail in /site/regions/*/item/mailbox/mail
where $mail//text/text() ftcontains
"seeking.*" with wildcards
return $mail/from
```
- Query Info Section:** Shows the query execution status: "Evaluating: 280.23 ms". It also displays the compiled query and the result set.

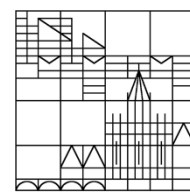
Query: for \$mail in /site/regions/*/item/mailbox/mail
where \$mail//text/text() ftcontains
"seeking.*" with wildcards
return \$mail/from

Compiling:

 - Converting where clause to predicate
 - Merging descendant-or-self step(s)
 - Applying full-text index

Result: for \$mail in FTIndexAccess("seeking.*" with wildcards)/parent::*:text/ancestor::*:mail[parent::*:mailbox/parent::*:item/parent::*:parent::*:regions/parent::*:site/parent::document-node()] return \$mail/from
- Text Section:** Displays the resulting XML data, which is a list of email addresses:


```
<from>Jerri Hatakka mailto:Hatakka@bellatlantic.net</from>
<from>Lukas Masaki mailto:Masaki@unl.edu</from>
<from>DAIDA Uhrík mailto:Uhrík@concordia.ca</from>
<from>Maia Graft mailto:Graft@fernuni-hagen.de</from>
<from>Kripasindhu Litecky mailto:Litecky@dec.com</from>
<from>Iriní Cedeno mailto:Cedeno@cohera.com</from>
<from>Fran Worthington mailto:Worthington@umich.edu</from>
```



Conclusion

XQUERY FULL TEXT

XQuery Full Text is getting popular!

- many of our users are already working with XQFT
- more and more implementations arise

Open Challenges

- suitable scoring algorithms for XML data (see INEX, SIGIR, ...)
- runtime optimizations to allow for index access of variable `ftcontains` strings

...thanks for listening!