Module 4

Implementation of XQuery

Part 3: Support for Streaming XML

Motivation

- XQuery used in very different environments:
  - XQuery implementations on XML stored in databases (with indexes).
  - **Main-memory XQuery implementations on XML in files, sent as streams, computed on the fly…**
- Example Applications:
  - Web Services (e.g., ActiveXML).
  - Telecommunication apps (XML messages).
  - XML documents.
  - Information Integration.

Challenges to Address

- Efficient Representation: Compression
- Matching Content/Message Brokering
- Discarding unneeded Data: Projection

Reducing the space overhead

- XML uses rather verbose syntax
  - High bandwidth overhead
  - Slow parsing speed
- Excludes usage in resource-constrained environments
- Compress XML to trade additional CPU time to storage/transfer cost

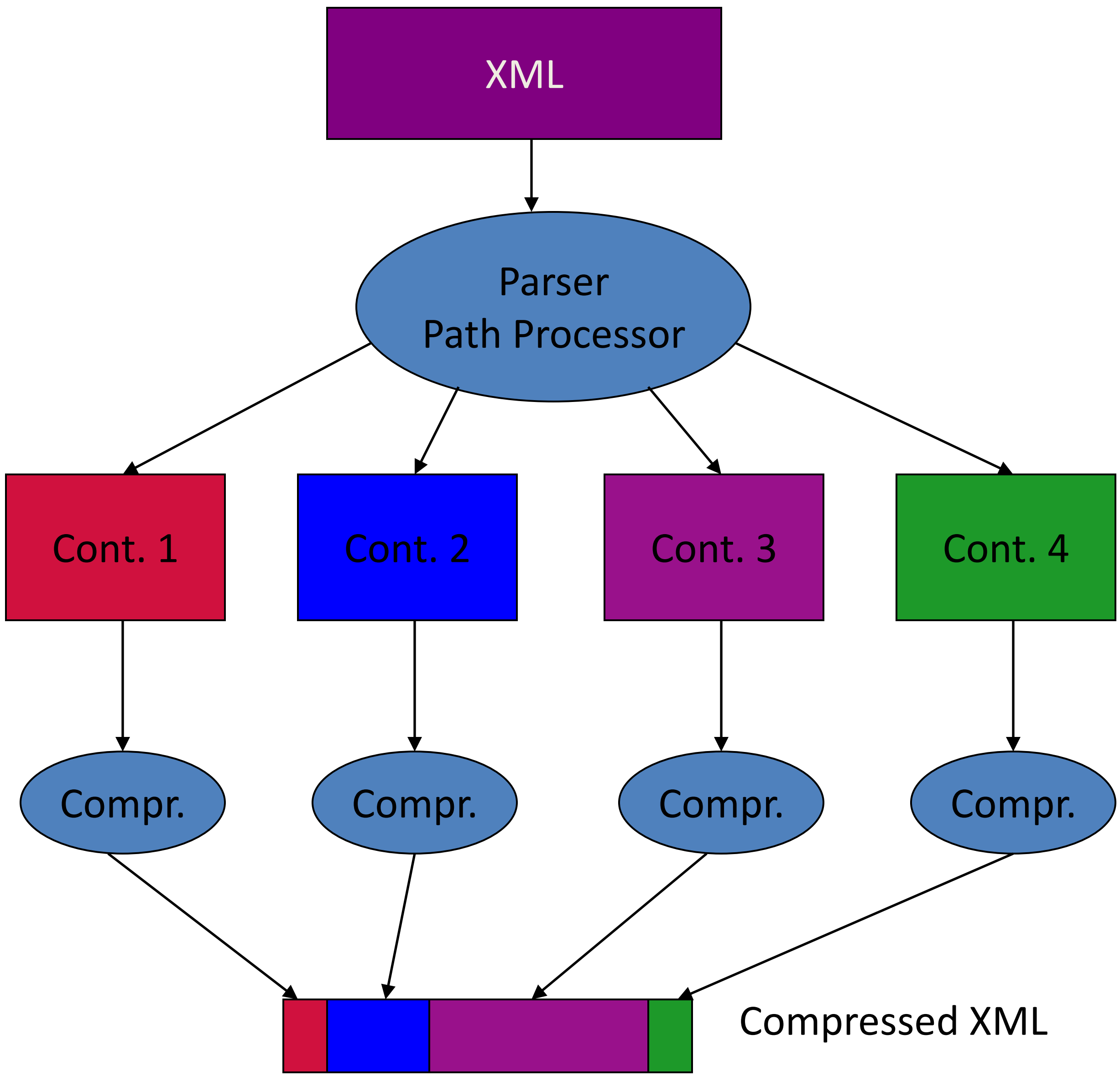
Classification of Compression

- XML knowledge
  - General Text Compression
  - Schema-dependent compression
  - Schema-independent compression
- Queryable
  - Archive-only
  - Homomorphic compression
  - Non-homomorphic compression

Compression

- Classic approaches: e.g., Lempel-Ziv, Huffman
  - decompress before queries
  - miss special opportunities to compress XML structure
  - Not Queryable at all
- XMill: Liefke & Suciu 2000
  - Idea: separate data and structure -> reduce entropy
  - separate data of different type -> reduce entropy
  - specialized compression algo for structure, data types
- Assessment
  - Very high compression rates for documents > 20 KB
  - Decompress before query processing (bad!)
  - Indexing the data not possible (or difficult)

Xmill Architecture



XMill Example

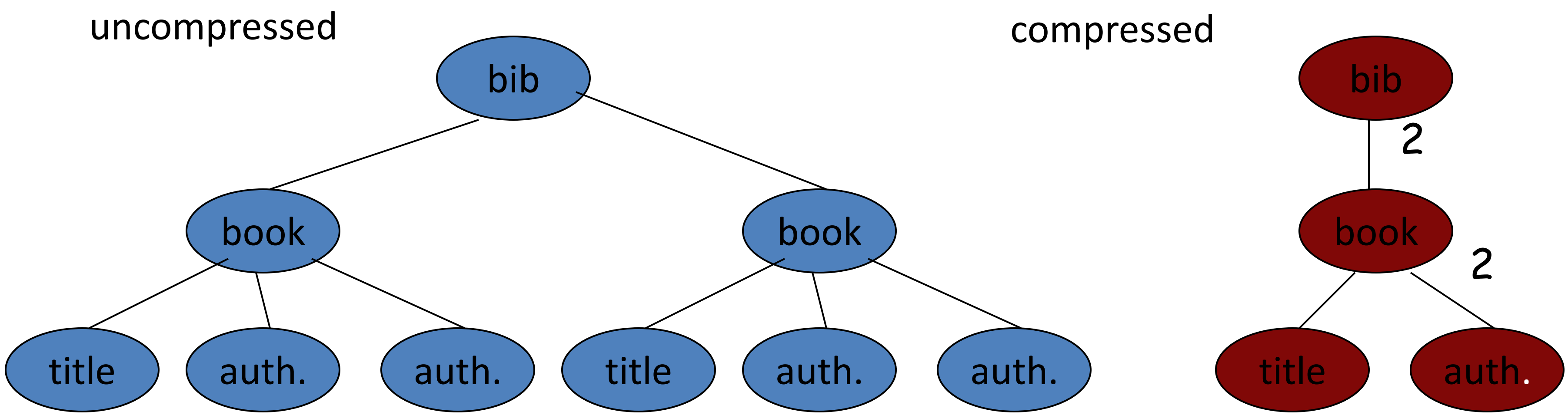
<book price=„69.95“>
<title> Die wilde Wutz </title>
<author> D.A.K. </author>
<author> N.N. </author>
</book>

- Dictionary Compression for Tags:
  book = #1, @price = #2, title = #3, author = #4
- Containers for data types:
  ints in C1, strings in C2
- Encode structure (/ for end tags) - skeleton:
  gzip(#1 #2 C1 #3 C2 / #4 C2 / #4 C2 / /)

Querying Compressed Data

(Buneman, Grohe & Koch 2003)

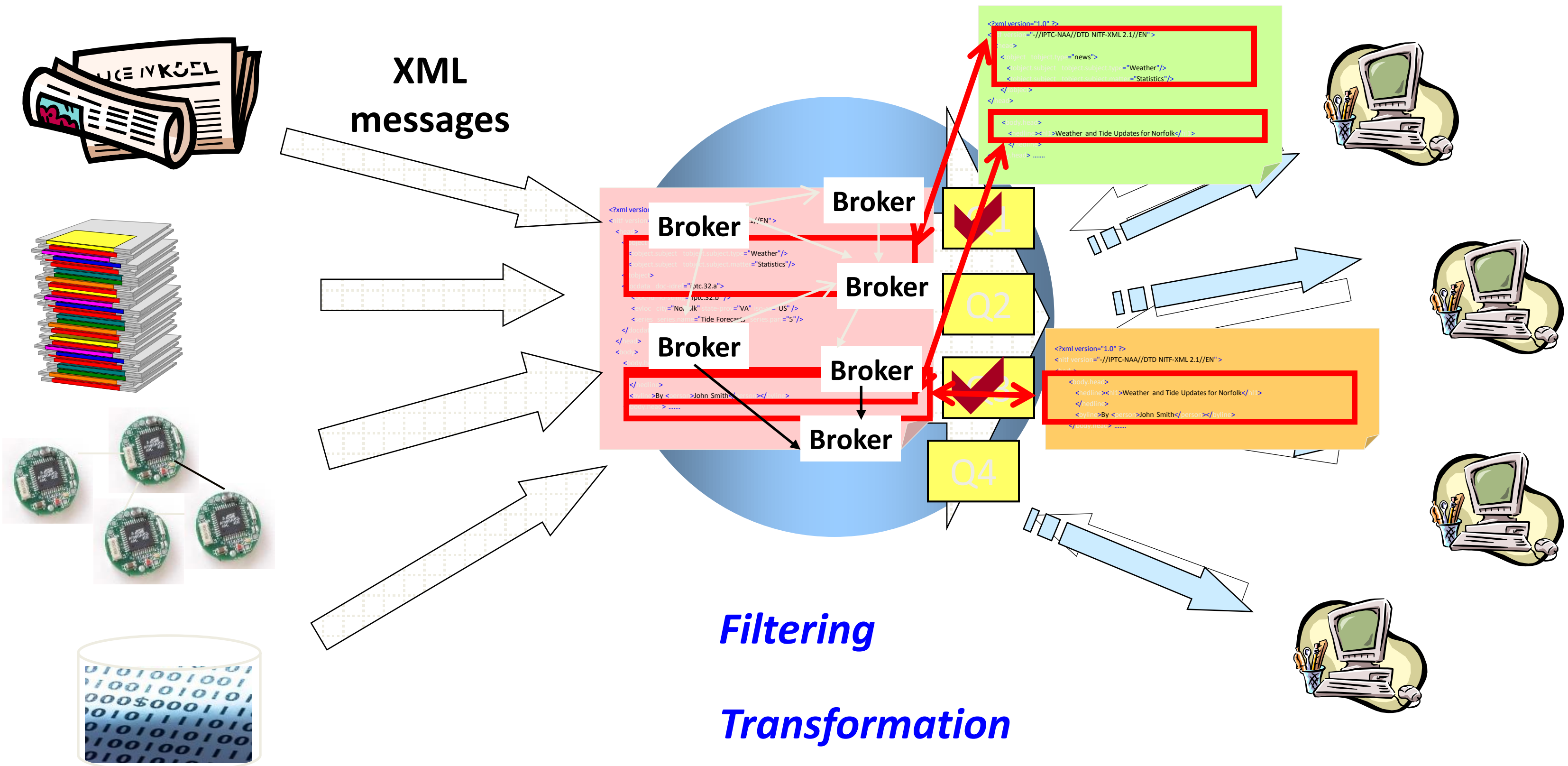
- Idea:
  - extend Xmill
  - special compression of skeleton
  - lower compression rates,
  - but no decompression for XPath expressions



Compression

- XML-aware compressors outperform text compressors
- Queryable compressors show worse compression than archival
- Not much adoption outside research
- Binary XML
  - picks up many compression ideas
  - Now a W3C standard: EXI

Content Matching: XML Message Brokering



Filtering

Transformation

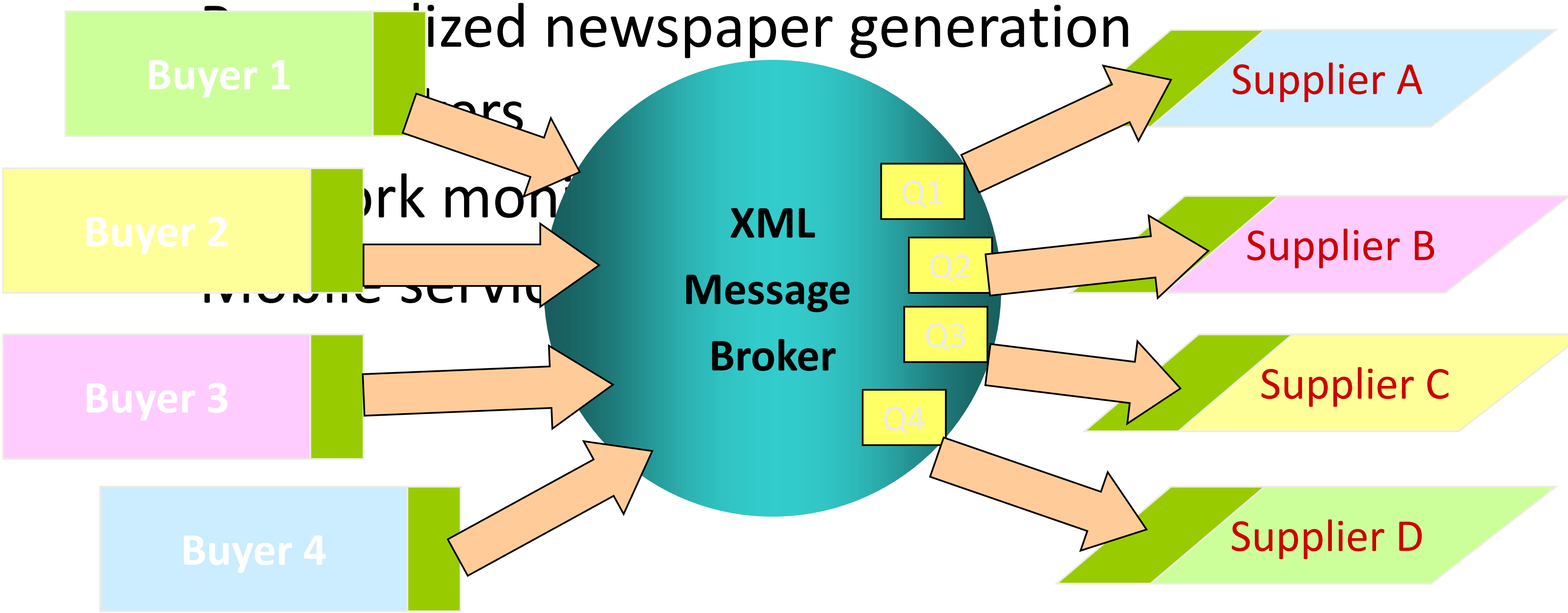
Routing

Message-based Middleware

- Publish/Subscribe
  - Subscribers express interests, later notified of relevant data from publishers.
  - Loose coupling at the communication level.
- XML, a de facto standard for online data exchange
  - Flexible, extensible, self-describing.
  - Enhanced functionality: XSLT, XQuery, …
  - Loose coupling at the content level.
- XML message brokering
  - Publish/subscribe + XML = flexibility at communication and content levels.
  - Declarative XML queries provide high functionality.

New Applications

- Message brokering supports a large number of emerging distributed applications:
  - Application integration
  - ized newspaper generation
  - ers
  - rk moni
  - obile servic



Problem Statement

Inputs:

(1) continuously arriving XML messages (usually small)

(2) a set of XQuery queries representing client interests

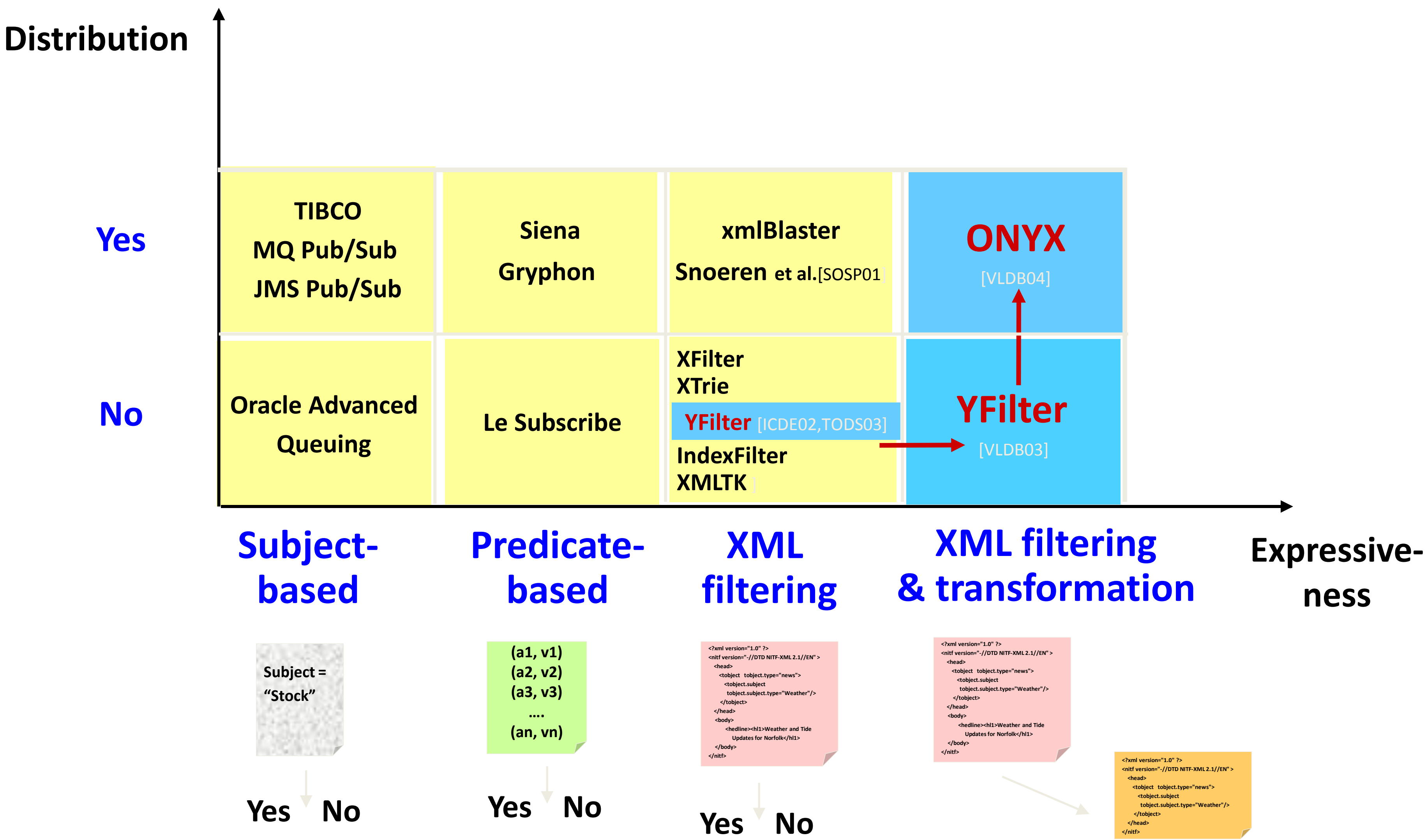
Main functions of an XML message broker:

- ***Filtering***: matches messages to query predicates.
- ***Transformation***: restructures the matching messages.
- ***Routing***: directs messages to queries over a network of brokers.

Challenges: providing this functionality for

- ***large numbers of queries*** (e.g., 10's thousands of them)
- ***high volumes of XML messages*** (e.g., tens or hundreds/sec)

Design Space



YFilter & ONYX

- **YFilter**, a system for XML filtering and transformation.
- ***Filtering*** exploiting sharing:
  - Order-of-magnitude performance benefits over previous work.
  - Scalable to 100's thousands of distinct queries.
  - YFilter 1.0 release: used in research projects and product development, being integrated into Apache Hermes for WS-Notification.
- ***Transformation*** exploiting sharing:
  - The first algorithm for transformation for a large set of queries.
  - Scalable up to 10's of thousands of distinct queries.
- ***Routing*** **(ONYX)**: an overlay network of brokers with routing abilities, providing flexible, Internet-scale XML dissemination services.

The Filtering Problem

- Full XPath/XQuery too expensive ☹
- Query language: path expression =

  (('/' | '//') (ElementName | '*') Predicate*)+

- The filtering problem:
  - Given (1) a set Q = Q, …, Qn of path queries, where each Qi has an associated query identifier, and (2) a stream of XML documents.
  - Compute, for each document D, the set of query identifiers corresponding to the XPath queries that match D.

Constructing an FSM for a Query

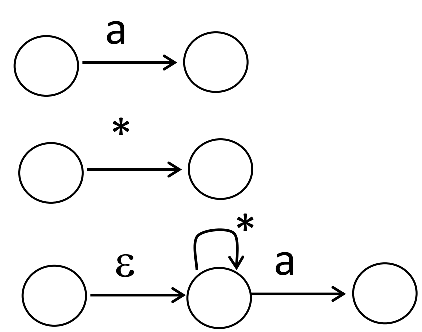
Key Idea: represent query paths as state machine that are driven by the XML parser (SAX)

- Simple paths: ((“/” | “//”) (ElementName | “*”))+
- A finite state machine (FSM) for each path: mapping steps to machine states.

Map location steps to FSM fragments.

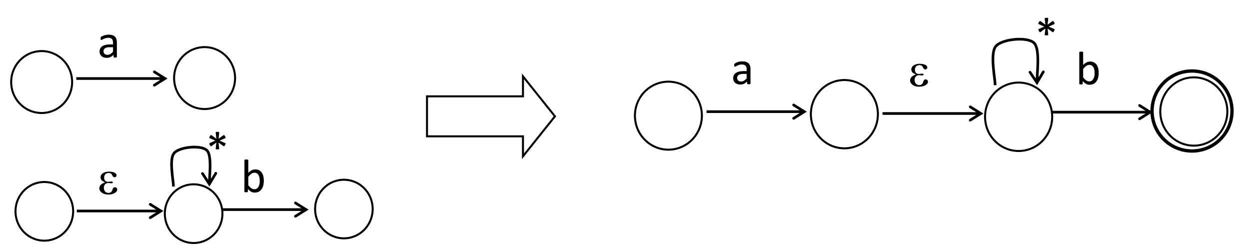
Location steps: /a, /*, //a

FSM fragments



Concatenate FSM fragments for location steps in a query.

Query “/a//b”



Constructing the Combined FSM

YFilter builds a single combined FSM for all paths!

- Complete prefix sharing among paths.
- Nondeterministic Finite Automaton (NFA)-based implementation: a small machine size, flexible, easy to maintain, etc.
- Output function (Moore machine): accepting states → partition of query ids.

Q1=/a/b

Q2=/a/c

Q3=/a/b/c

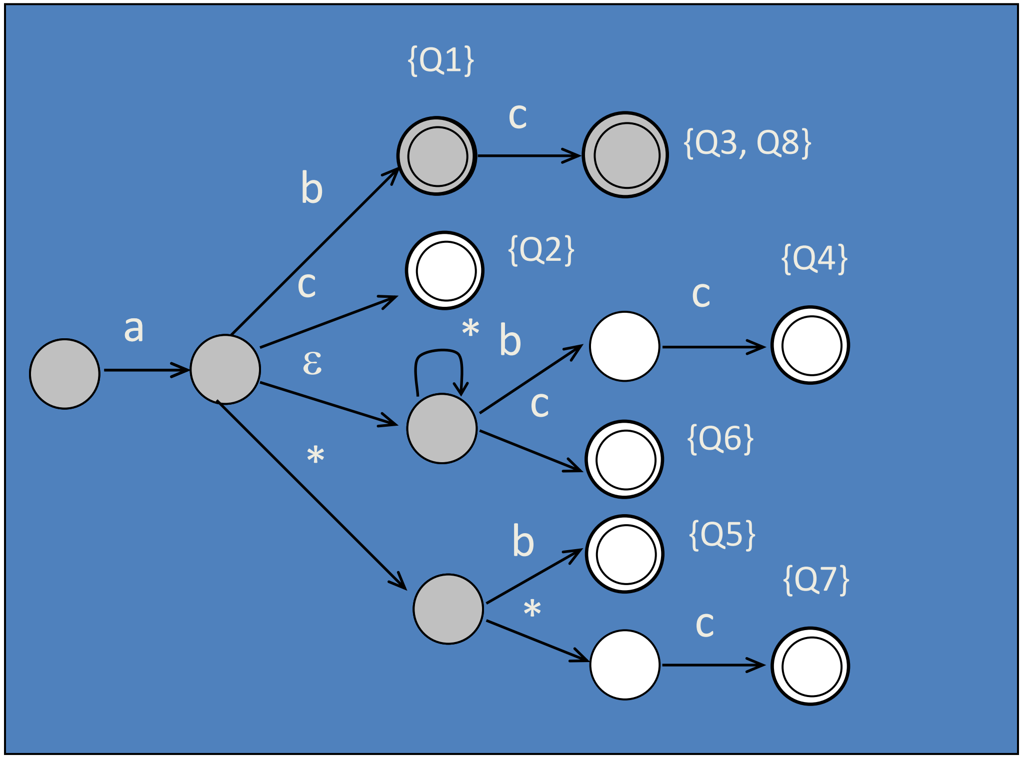
Q4=/a//b/c

Q5=/a/*/b

Q6=/a//c

Q7=/a/*/*/c

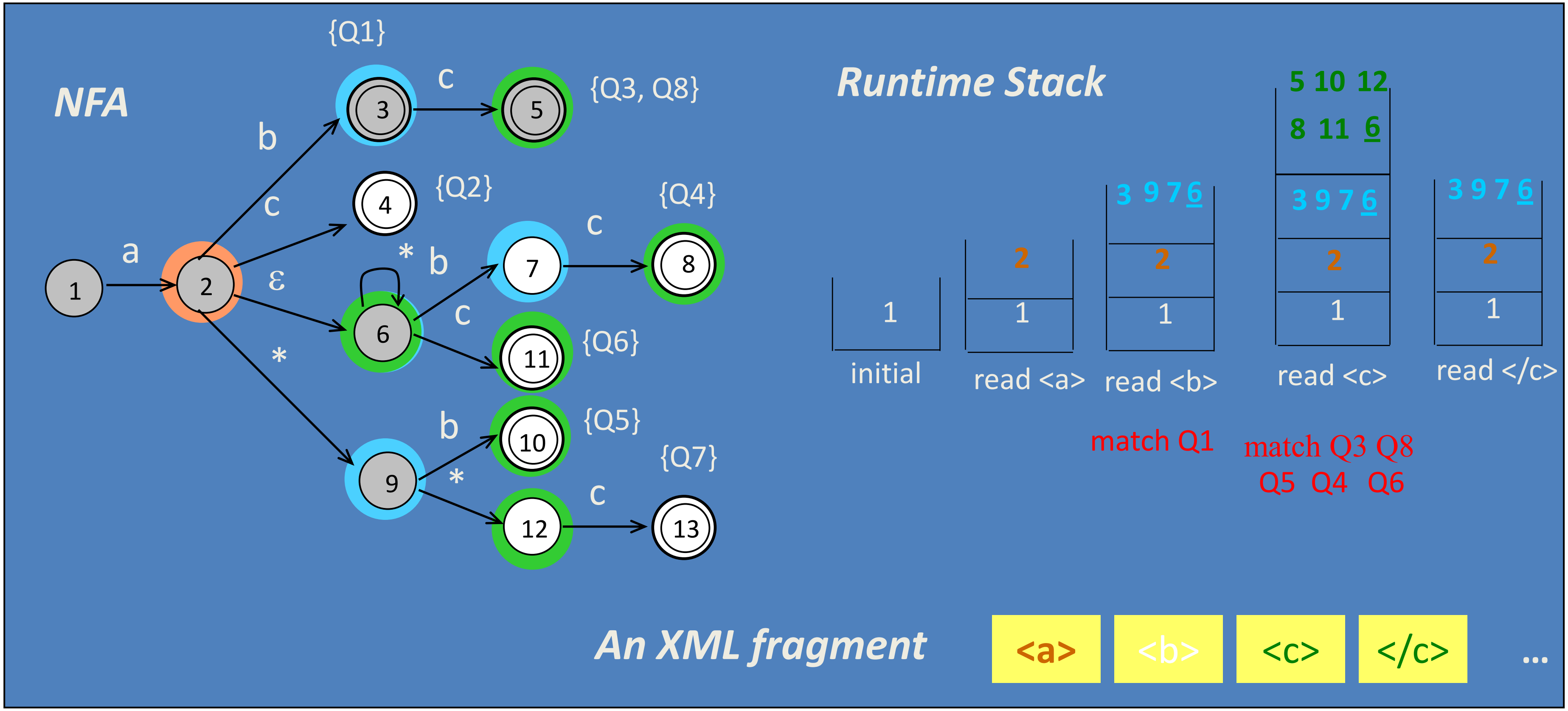
Q8=/a/b/c



Execution Algorithm

YFilter uses a stack mechanism to handle XML

- Backtracking in the NFA.
- No repeated work for the same element!



DFA vs. NFA

- DFA has exponential number of states
  - Large main-memory requirements
  - Or I/O needed in order to process messages
- DFA has high maintenance costs
  - Need to rerun Myhill/Büchi algorithm, everytime a new profile is posted or deleted
- NFA is slower than DFA
- NFA: entries in stack can grow exponentially
  - In practice, XML documents are fairly flat
- **NFA is the clear winner (current trade-offs)!**

Performance results for YFilter

- YFilter scales to 150,000 distinct path queries w/o predicates.
  - Consistently takes 30 msec or less.
  - Achieves a 25x performance improvement over previous approaches
- Deep element nesting: No exponential blow-up of active states.
- Sensitivity to ‘*’ and “//”: Little, due to effective prefix sharing.
- NFA maintenance for query updates: Tens of milliseconds for inserting 1000 queries.
- YFilter handles 100’s thousands of queries with predicates.
  - No real competition before
  - Mechanism not shown here. What are the difficulties?

XML Projection

Memory Limitations

- Main-memory XQuery implementations cannot handle large documents.
- Complex XQuery expressions require materialization (DOM).
- DOM is the bottleneck.

XQuery Processors	Maximum Document Size
Quip	7Mb
Kweelt	17Mb
Galax	33Mb
Xalan (XSLT)	75Mb

XMark Query 1 on an IBM laptop T23 (256Mb RAM)

Projection: Example

XMark Query 1
for $b in /site/people/person[@id="person0"]
return $b/name

```
<site>
    <regions>...</regions>
    <people>
    ...
      <person id="person120">
         <name>Wagar Bougaut</name>
         <emailaddress>mailto:Bougaut@wgt.edu</emailaddress>
      </person>
      <person id="person121">
         <name>Waheed Rando</name>
         <emailaddress>mailto:Rando@pitt.edu</emailaddress>
         <address>
            <street>32 Mallela St</street>
            <city>Tucson</city>
            <country>United States</country>
            <zipcode>37</zipcode>
         </address>
<creditcard>7486 5185 1962 7735</creditcard>
<profile income="59224.09">
...
```

Less than 2% of original document !

Projection: Intuition

- Given a query:

  ```
  For $b in /site/people/person[@id="person0"]
  Return $b/name
  ```

  - Most nodes in the input document(s) are not required.
  - Projection operation removes unnecessary nodes.
  - Evaluation of the query on projected document yields the same results as on the original document.
- How it works:
  - Projection defined by set of paths.
  - Static analysis infers sets of paths used within a query.

```
/site/people/person
/site/people/person/@id
/site/people/person/name
```

Projection: Challenges

- For an XQuery expression, compute all paths that allow to reach nodes required to evaluate the expression.
- XQuery is complex:
  - Variables
  - Composition
  - Syntactic Sugar
  - Complex expressions
- Have to be able to analyze all of XQuery.

XML Projection

- Similar to relational projection:
  - One key operation.
  - Prunes unnecessary part of the data.
  - Essential for memory management.
- Specific problems related to XML:
  - Projection must operate on trees.
  - Requires analysis of the query.
  - Need to address XQuery complexity.

Notation

- Projection Paths:
  - Path expressions are noted using XPath semantics (/site/people/person/@id)
  - “#” notation used when subtree should be kept (/site/people/person/name#)
- Static Analysis: inference rule notation

$$Expr \Rightarrow Paths$$

Static Analysis: Variables

- Variables can be bound to nodes coming form different paths.

```
for $b in /site/people/(teacher | student)
return $b/name
```

- Analysis must remember paths to which variable was bound

```
/site/people/teacher
/site/people/student
```

- Environment is maintained during path analysis:

$$Env \mid\!- \; Expr \Rightarrow Paths$$

Static Analysis: Example

- Literals do not require any paths:

$$\frac{}{Env \ |\text{-}\ Literal \Rightarrow \{\}}$$

```
32 => {}
```

- Paths are propagated in a sequence:

$$\frac{Env \ |\text{-}\ Expr1 \Rightarrow Paths1 \quad Env \ |\text{-}\ Expr2 \Rightarrow Paths2}{Env \ |\text{-}\ Expr1,Expr2 \Rightarrow Paths1 \ U \ Paths2}$$

```
/a/b => {/a/b}
```

```
/a/d => {/a/d}
```

```
/a/b,/a/d
=> {/a/b,/a/d}
```

Static Analysis: Composition

```
(if (count (/site/regions/*) = 3)
then /site/people/person
else /site/open_auctions/open_auction)/@id
```

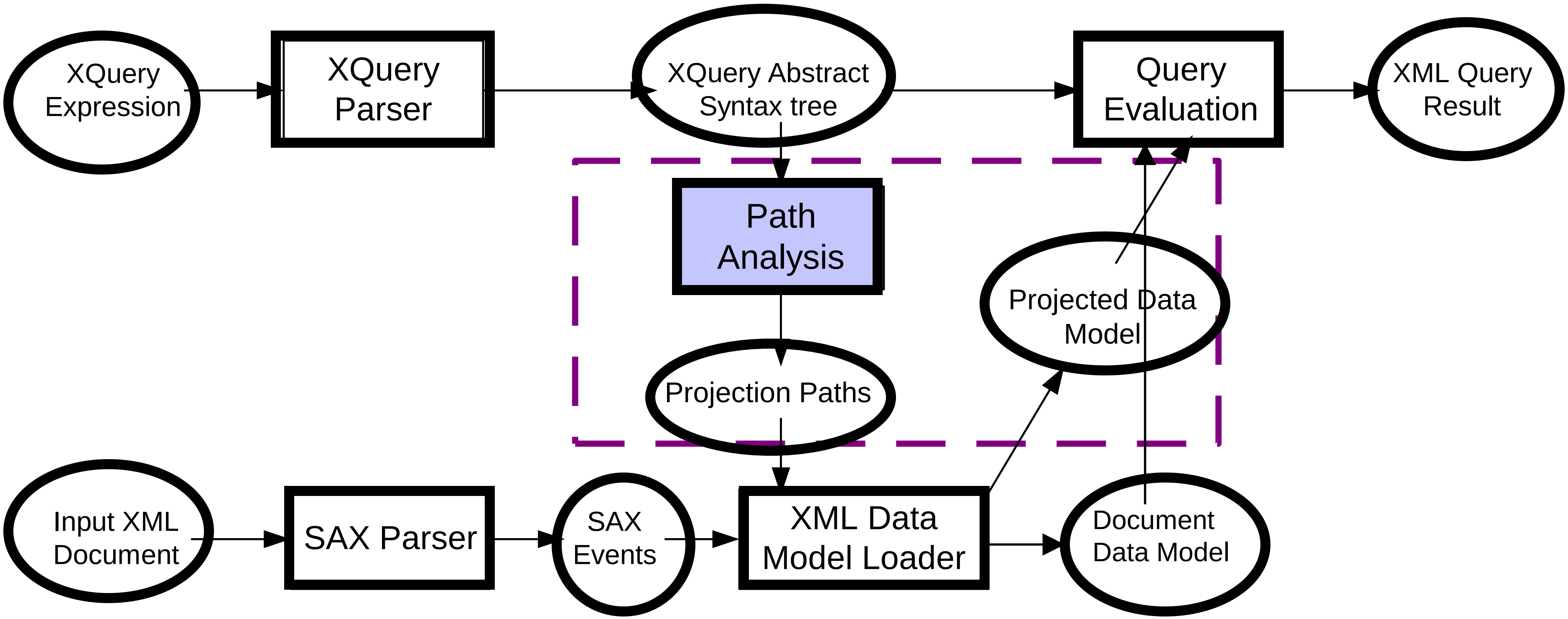
- **`/@id`** does <u>not</u> apply to **`/site/regions/*`**
- Final set of paths should be

```
/site/regions/*
/site/people/person/@id
/site/open_auctions/open_auction/@id
```

- Need to differentiate two sets of paths during analysis:
  - *Returned Paths*: returned by the expression, further path steps are applied on them.
  - *Used Paths:* used to compute the expression.

Env |- *Expr* => *Paths* using *UPaths*

XQuery Processing Architecture



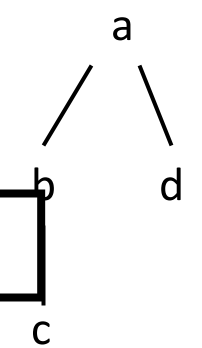
Loading Algorithm: Description

- Input:
  - Set of projection paths.
  - Document SAX events.
- Decide on action to apply on document nodes:
  - Skip: ignore node and its subtree.
  - KeepSubtree: keep node and its subtree.
  - Keep: keep node without its subtree.
  - Move: keep processing SAX events. Current node is only kept if some of its children are kept.
- Keep a set of current paths.

Loading Algorithm: Example

Projection Paths:
/a/b/c#
/a/d

Loaded Nodes:

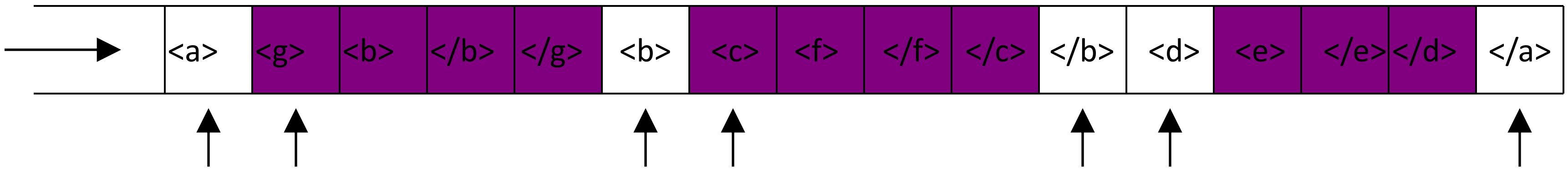


Similar to XML filtering algorithms

Current Pat

Limitations:
- Backward Axis!
- Number of current paths can be huge (descendant axis)

Document Stream

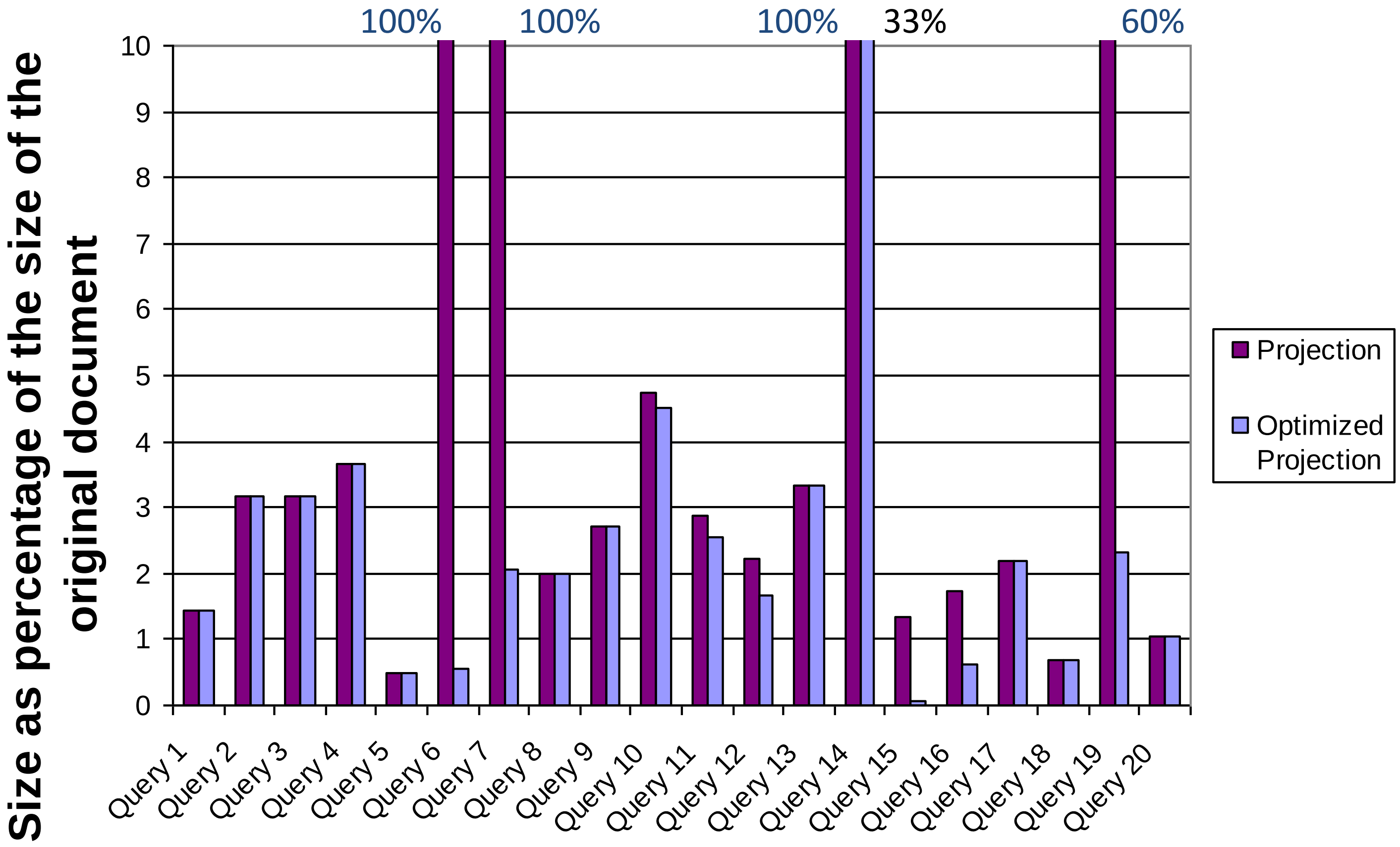


Experiments: Settings

- XML Projection Evaluation:
  - **Effectiveness**: projection impact on different queries.
  - **Maximum document size**: largest document that can be processed.
  - **Processing time**: effect on processing time.
- Experimental Setup:
  - Default XMark document size: 50Mb.

Configuration	CPU	Cache Size	RAM
A	1GHz	256Kb	256Mb
B	550MHz	512Kb	768Mb
C (default)	1.4GHz	256Kb	2Gb

Experiments: Effectiveness

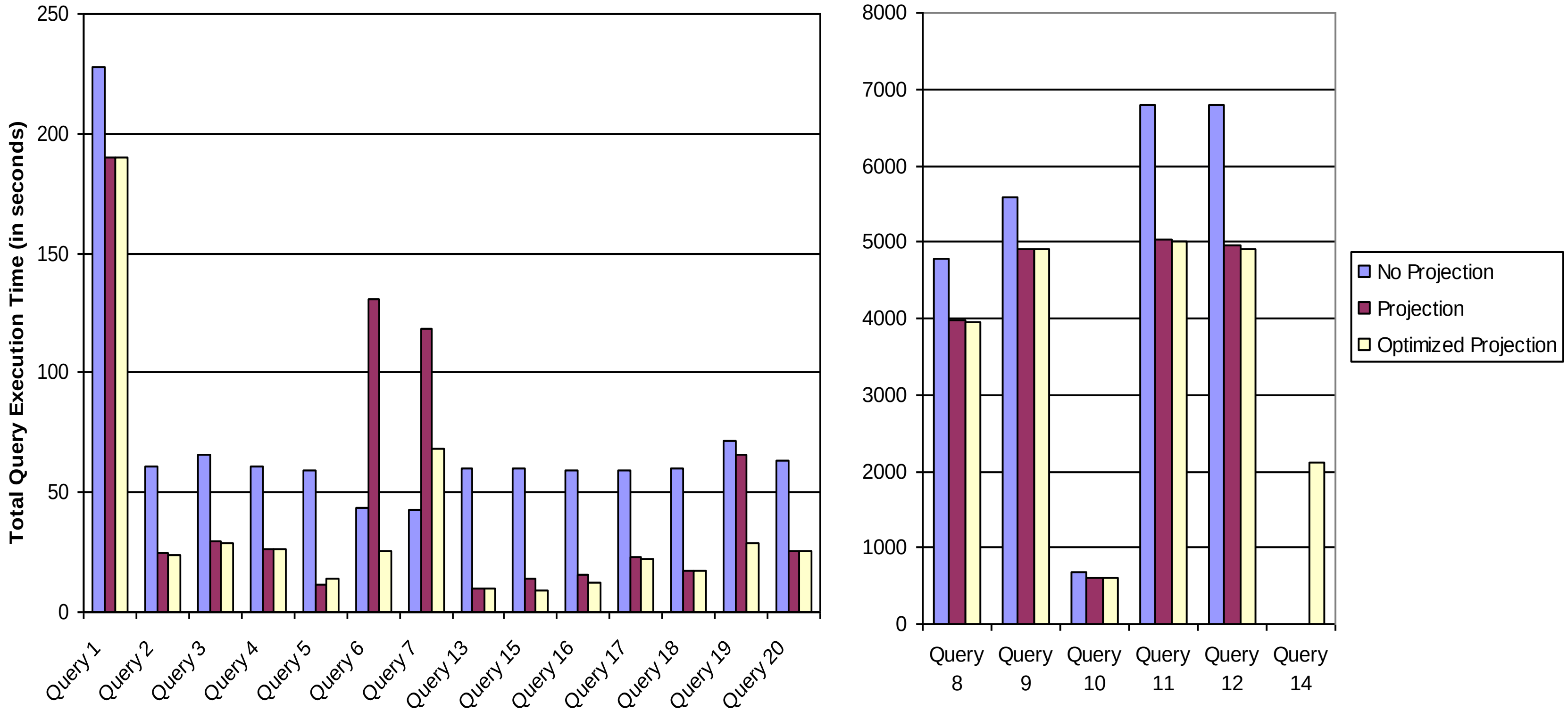


All queries but one require less than 5% of the document.

Experiments: Maximal Document Size

Configuration		A	B	C
XMark Query 3 (simple selection with predicate)	No Projection	33Mb	220Mb	520Mb
	Optimized Projection	1Gb	1.5Gb	1.5Gb
XMark Query 14 (Non-selective path query with predicates)	No Projection	20Mb	20Mb	20Mb
	Optimized Projection	100Mb	100Mb	100Mb
XMark Query 15 (Long, very selective path query)	No Projection	33Mb	220Mb	520Mb
	Optimized Projection	1Gb	2Gb	2Gb

Experiments: Query Execution Time

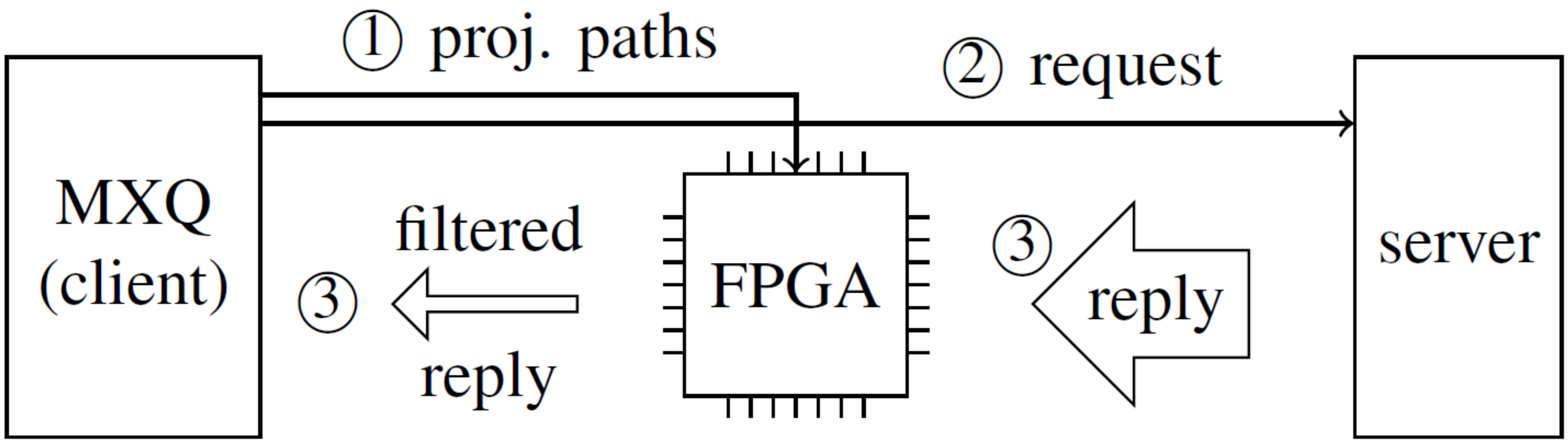


Projection significantly reduces query processing time
Next Bottleneck: Joins!

Hardware-based Projection

- Projection effective to reduce memory consumption, document processing cost
- Still bound by XML parsing speed
  - Best parsers on modern CPUs: 10-30 MB/s
- How can we do better:
  - Hardware/Software Co-Design!
  - Run Projection on an FPGA!
  - Parse and project on wire speed!

Hardware-based Projection (2)



1. Extract Projection Path, load into FPGA
2. Request XML document
3. Send (regular) XML to FPGA
   Receive filtered XML from FPGA

FPGAs

- Field-Programmable Gate Arrays
- Reconfigurable Hardware
  - Memory
  - Logic Gates
  - Wires
- Massive parallelism possible
- „Create“ custom processor
- Slow to reprogram

Projection Processing on FPGAs

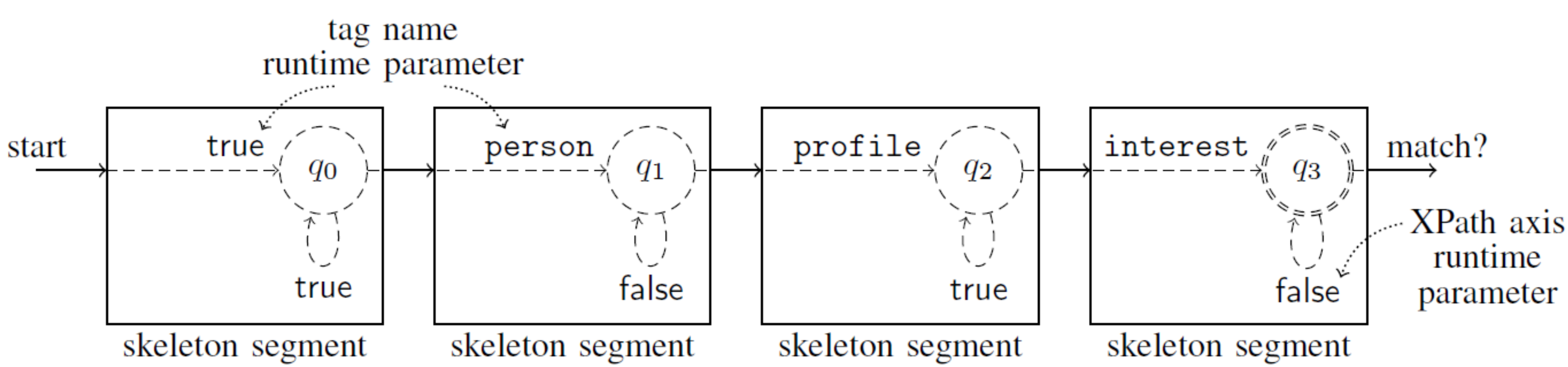
- FPGA very efficient in running automata

⇒Use automata-based path processing (see before)

- Reprogramming Slow

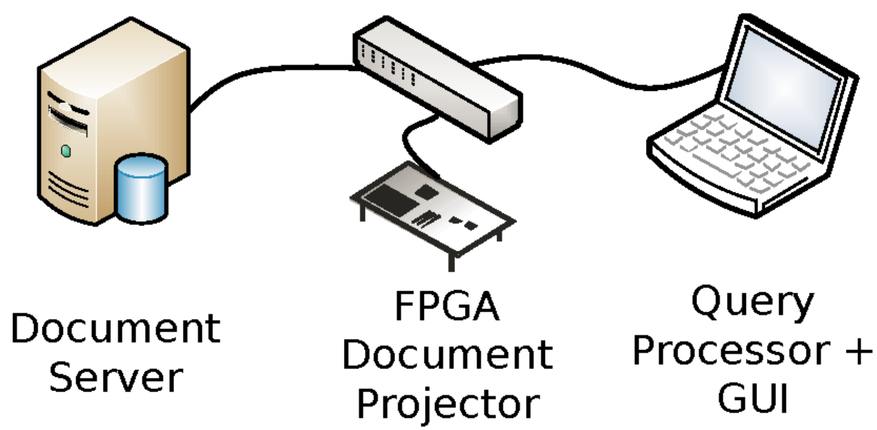
⇒Provide general „skeleton“ path processor

⇒Instantiate for specific projection paths

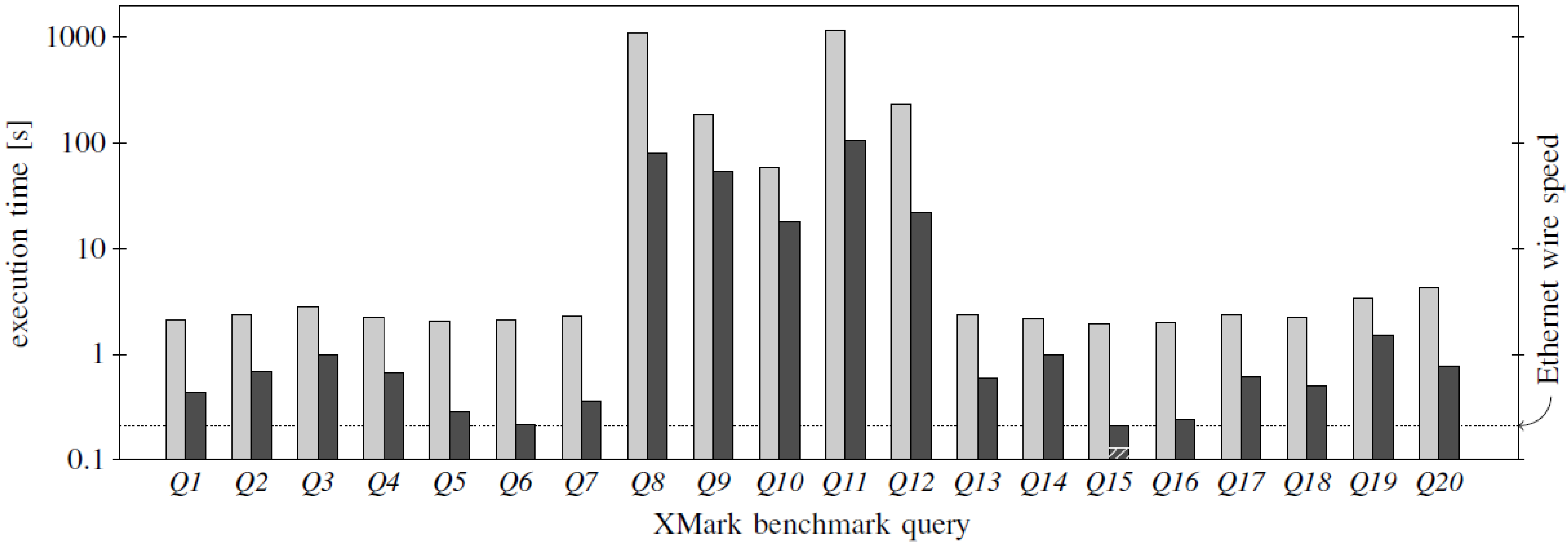


Evaluation/Demo Setup

- Use FPGA boards with 1GB Ethernet
- Send XML document over network using UDP
- Run stock MXQuery with UDP receiver



Performance Results



- Performance gains of 1-2 orders of magnitude
- Many queries close to network limit
- Q15 slowed down by Gigabit Ethernet!