Efficient Filtering of XML Documents for Selective Dissemination of Information

Mehmet Altinel and Micheal J. Franklin

CIS650 – Advanced Topics in Databases

Murat Cakir

Outline

- Introduction to SDI systems
- An XML Based SDI Structure
- XFilter Implementation
- Enhanced Filtering Algorithms
- Performance Issues

Introduction

- The development of the Internet and networking technologies made it possible to access increasing volumes of data in a convenient way.
- As a consequence of these advances, Information Dissemination applications are gaining popularity in distributing data to the end users.
- Increasing volume of data available in electronic format forces the designers of ID systems to broadcast their data in a selective manner.
- Selective dissemination of information (SDI) applications filter unnecessary data by considering user profiles.
- E.g.: timely received/collected new data such as stock quotes, traffic news, sports tickers and music

Introduction

- Key challenge in SDI is to efficiently and quickly search the huge set of user profiles to identify the relevant documents.
- Traditional SDI systems:
  - Are based on simple keyword matching and typical Information Retrieval techniques.
  - e.g. a subscriber profile containing the keyword “NBA” will match all those news containing the keyword “NBA”
  - Subscriber might also receive irrelevant information such as news with headline “Bill Gates love to watch NBA”
  - Effectiveness of profiles is more important than the efficiency of filtering to produce quality results in this context.
  - This limits scalability of the IR based systems.
  - Cannot exploit the structure of the document containing the data

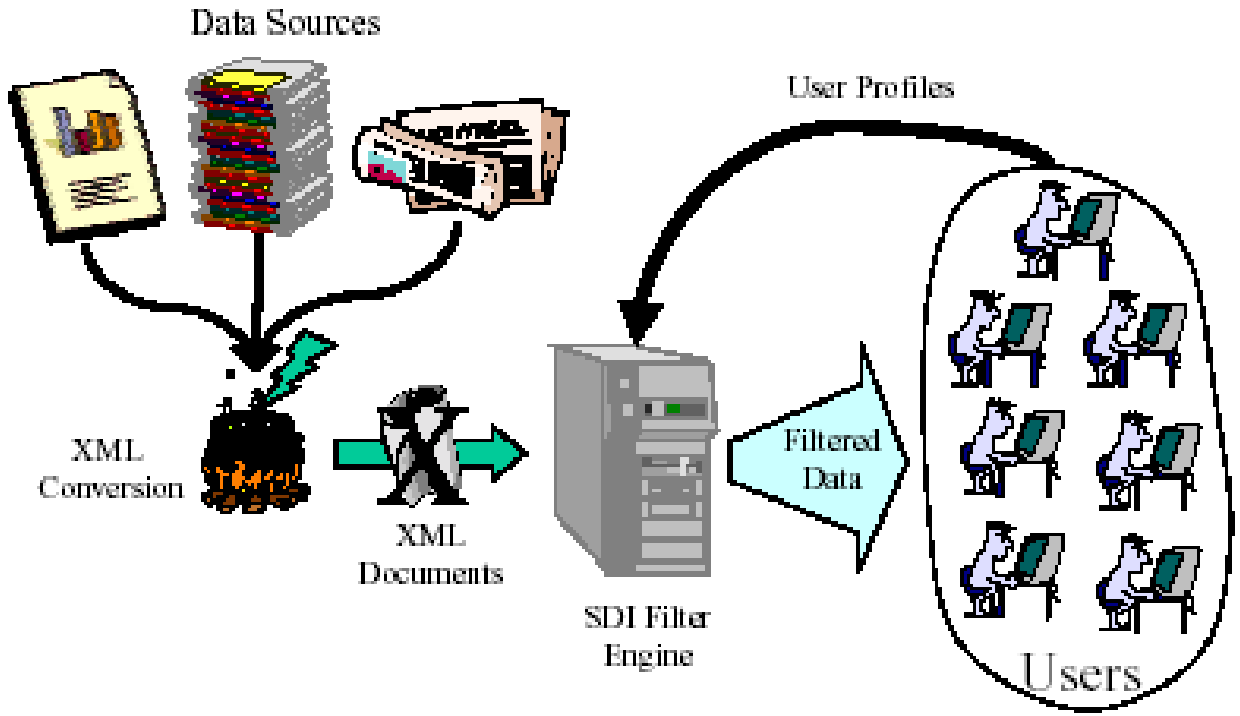
Introduction

- Important observation: The roles of queries and the data are reversed in building high-performance, scalable SDI systems.
- In a DBMS data items are indexed and stored
- In a SDI system large numbers of queries are stored , and the documents are matched against them.
- Thus, it is reasonable to index queries (user profiles) in a SDI system.

Introduction – Use of XML in SDI

- XML has emerged as a standard information exchange mechanism on the Internet.
- XML allows encoding of structural information within documents.
- This structure can be used to create more accurate user profiles.
- Matching profiles to documents will be an additional overhead in this approach.
- “XFilter” tries to exploit the structure of the XML documents to perform more efficient and more accurate filtering.

An XML-based SDI Architecture



- Incoming documents are XML-encoded.
- Profiles are converted into a format that can be efficiently stored and processed by the Filter Engine. (XPath based)

- Subscribers create their profiles via a GUI
- These preferences are treated as standing queries applied to all incoming docs.
- Profile model is based on XPath
- E.g. /sports/nba//news

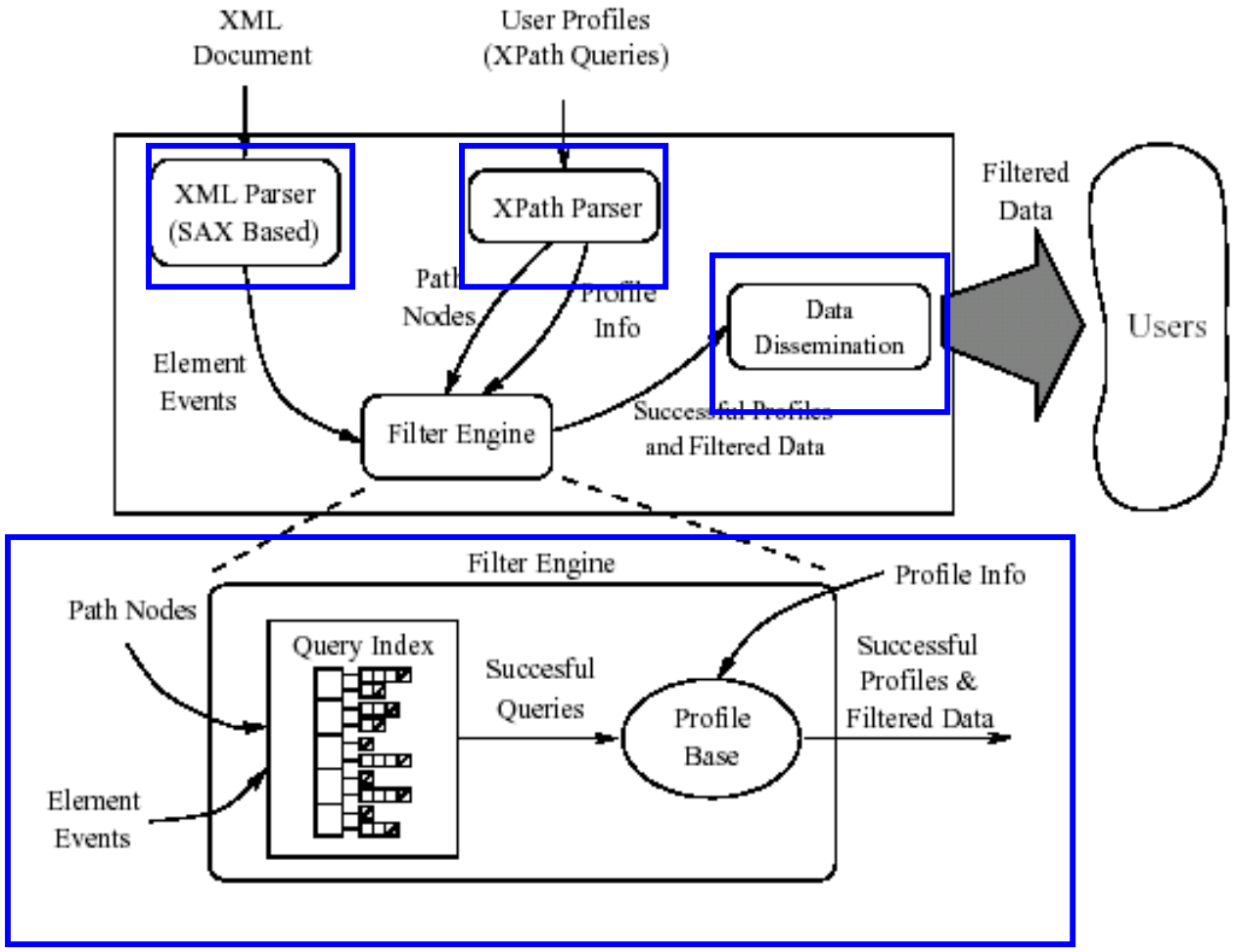
XPath

- XPath treats XML docs as a tree of nodes
- XPath expressions are patterns that can be matched to nodes in a XML tree.
- Evaluation of a Xpath expression yields either a node set, a boolean, a number or a string.
- “/” parent-child operator
- “//” ancestor-descendant operator
- “*” wildcard: matches any element name

XPath as a Profile Language

- "[…]" denote filter expressions
- E.g. //product[price/msrp<300]/name
- Selects name elements of the XML document if the msrp of the product element is <300.
- In XFilter , XPath is used to select entire documents rather than parts of documents.
- If the XPath exp. representing profile info matches at least one element of a document, then the document is passed to the user.

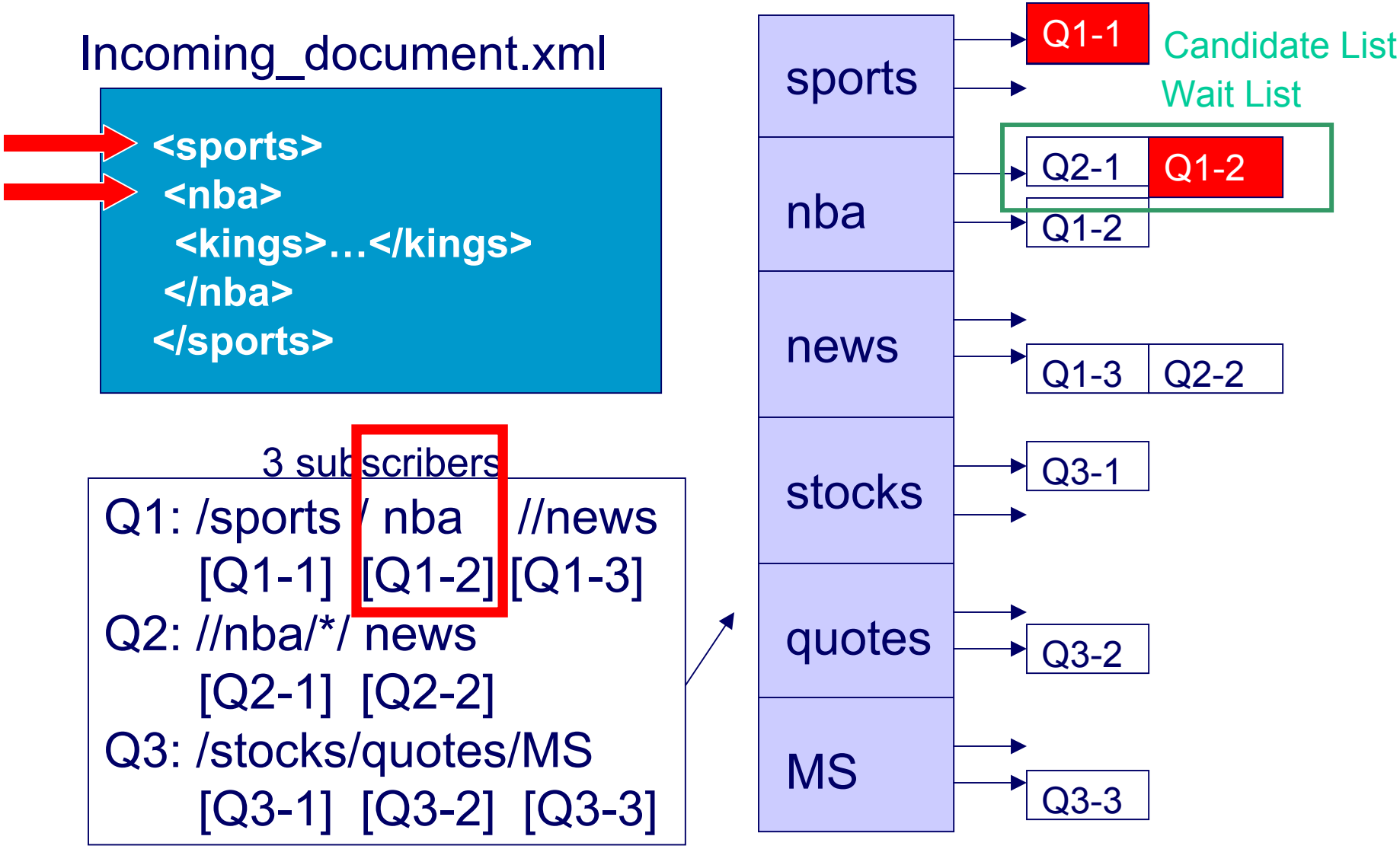
XFilter Architecture



Major components:

1. Event-base parser for XML document
2. XPath parser for user profiles
3. Filter engine, matching between profile and XML documents
4. Dissemination engine, for delivery the filtered data

An Illustrative Example

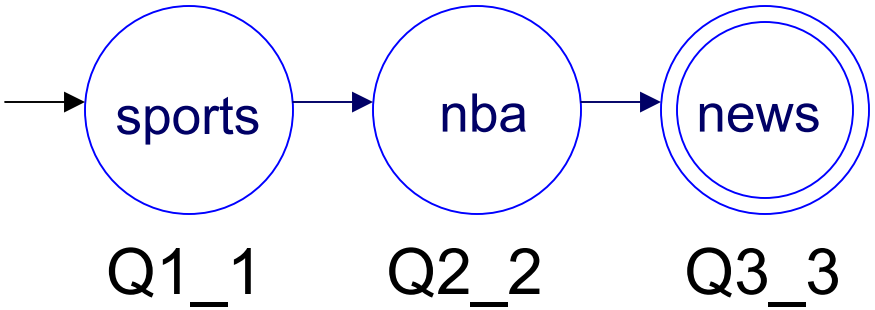


Filter Engine of XFilter

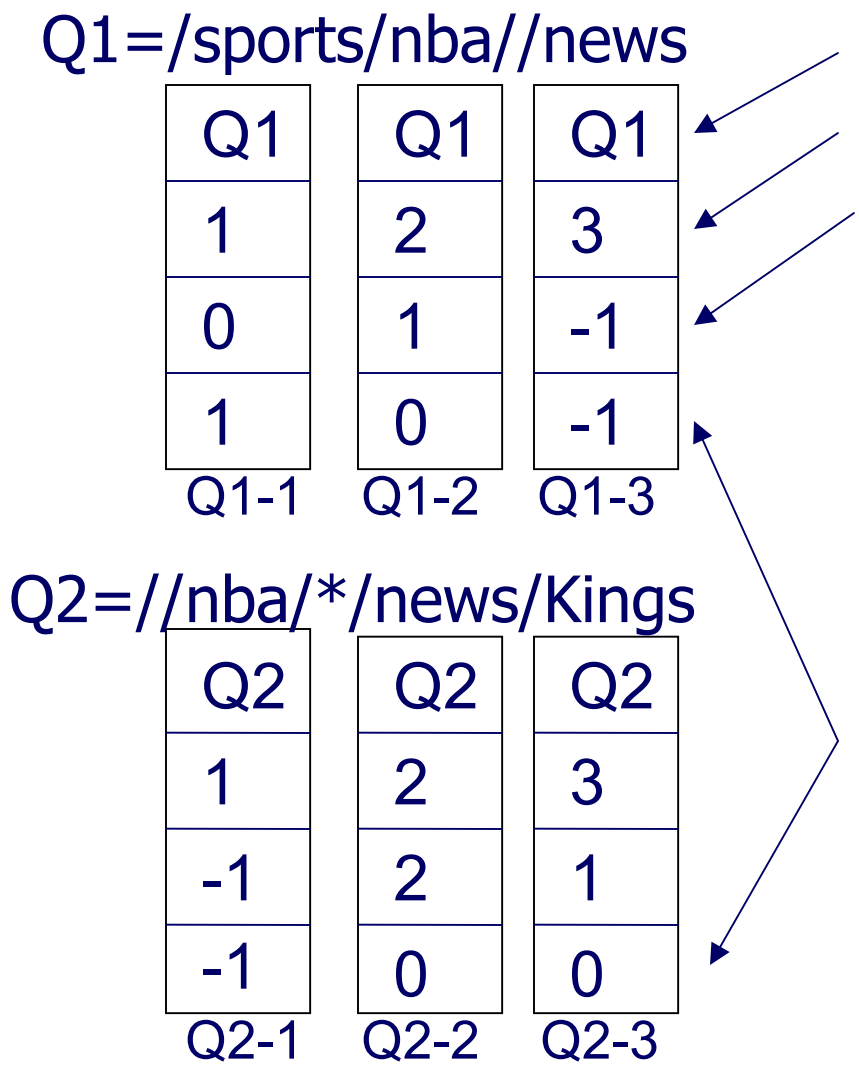
- XFilter converts each XPath expression (representing user profiles) to a Finite State Machine for efficient evaluation.
- A user query matches to the incoming XML document WHEN the FSM of the XPath query reaches its final state. In that case the document is sent to the user.
- A Query Index is built over the elements of the XPath queries.

Path Nodes and FSMs

- XPath parser decomposes XPath expressions into a set of path nodes.
- These nodes act as the states of corresponding FSM
- A node in the Candidate List denotes the current state
- The rest of the states are in corresponding Wait Lists.
- e.g. Q1 = /sports/nba//news
- Corresponding FSM:



Decomposing Into Path Nodes



Query ID

Position

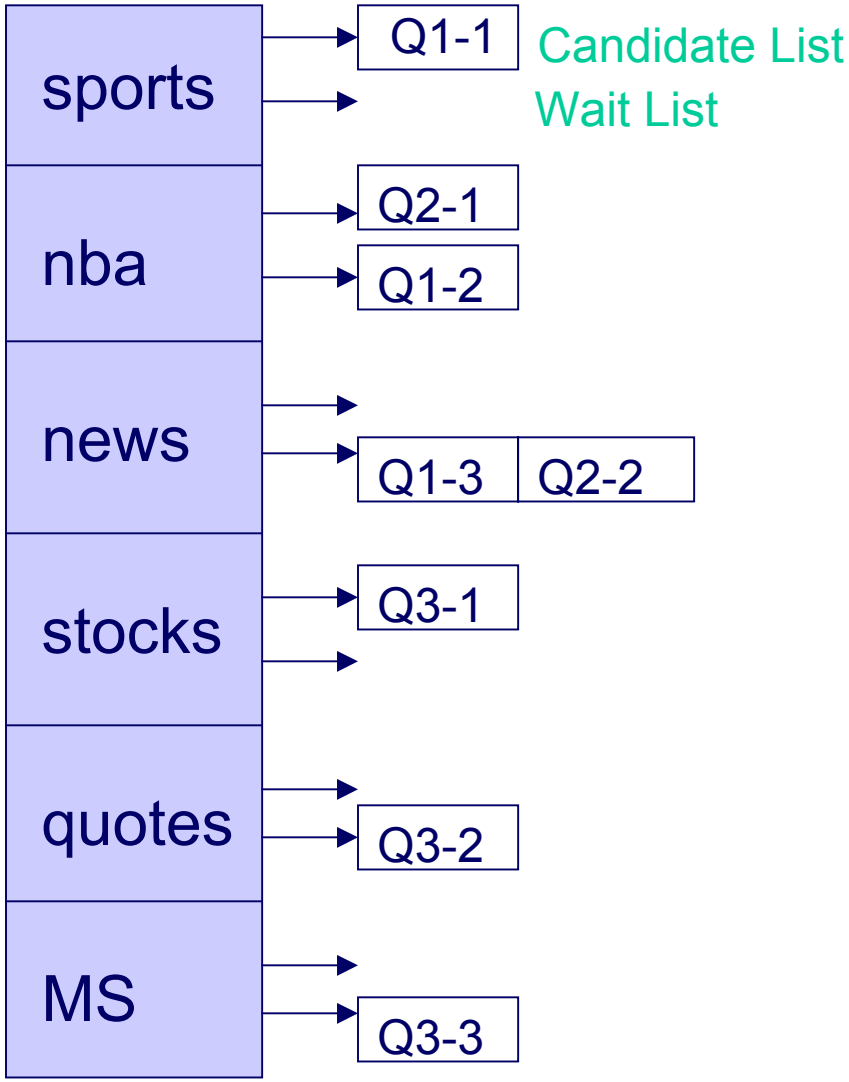
Relative Position:

- =0 for 1st node if 1st node is not preceded by "//"
- =-1 for any node preceded by "//"
- Else =1+ (no of "*" nodes between itself and predecessor node)

Level:

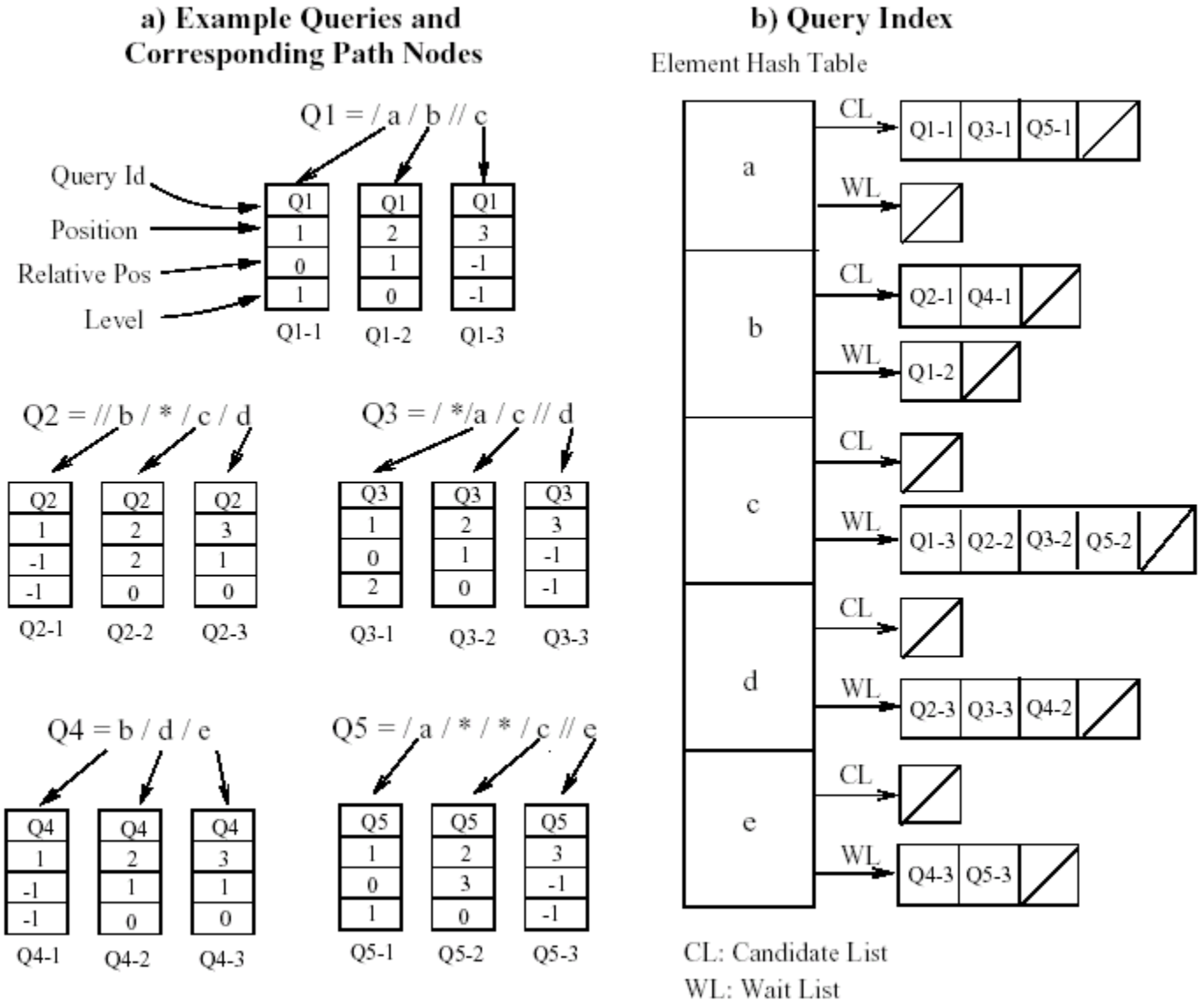
- If 1st node and have absolute distance from the root, then level = 1+ distance from root
- If Rel. Pos. is –1, it is also –1, else =0

Query Index



- All element names appearing on the user queries are added to the Query Index
- Each unique element name is linked to two lists: Candidate List and Wait List
- The current state of each query is placed in CL, others are in WL
- Events that cause state transition are generated by the doc XML parser.

How to Build an Index on Path Nodes



Event Based XML Parsing

- Events are used to drive the profile matching process.
- When a XML document arrives, it runs thru the SAX XML Parser and checks the corresponding entry in the Query Index when encountering:
  - A begin element tag
  - An end element tag
  - Data internal to an element

Input XML	SAX API
<?xml version="1.0"> <sports><news> <nba> <kings> Kings:112 – Lakers:100 </kings> </nba> </news></sports> </doc>	Start document Start element: sports Start element: news Start element: nba Start element: kings Characters: "Kings:112 – Lakers:100" End element: kings End element: nba, … , End element: sports End document

Event Based XML Parsing (cont)

- Start_Element_Handler (element_name, element level, attribute name, attribute values)

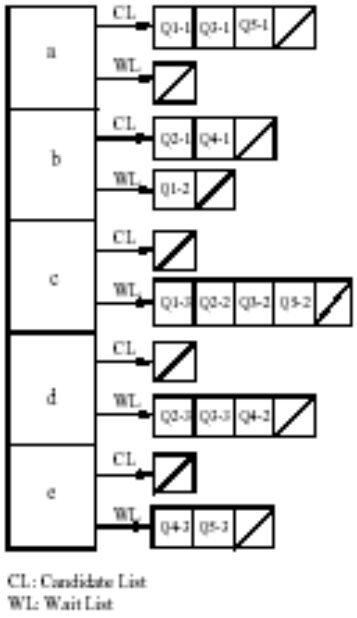
  {

  Lookup the element name in the Query Index and examine all nodes in the CL and perform LEVEL CHECK and ATTRIBUTE FILTER CHECK

  }

Q1
1
0
1

Q1-1



Level Check and Attribute Check

- Level check is made to ensure that the element appears in the document matches the expected level in the user query
- Recall:
  - If the level of a path node is –1 → relative pos is –1 i.e. a "//" is before this node → unrestricted (can be dynamically updated)
  - else the level of path node must = the level of the input element
- The attribute filter check applies any simple predicates that reference the attributes of the element

Level Check and Attribute Check

- If both level check and attribute check succeeds, we continue to process that query.
- If that node is the final path node (final state) of the query (e.g. Q1-3) then the document matches the query; else the FSM is moved to the next state.
- State transition is done by copying the next node of the query from WL to CL and update the corresponding relative position and level

End Element Handler and Element Characters Handler

- When an end element (i.e. closing tags in XML) is encountered in SAX parser, the path node of that element is deleted from CL.
- When element data is encountered in SAX parser, it works like the start element handler except it performs a content check rather than an attribute check.

List Balancing

- Recall:
  - The first path node of the XPath query is placed on the CL and remaining path nodes are placed on corresponding WLs. Problems?
  - Inefficient for many situations in an XML doc as the 1st element usually have poor selectivity.
  - Some CL will be very long as compared to others (e.g. if we are broadcasting news, the length of CL of element “news” will be very long since all subscribers will be interested in element “news”)

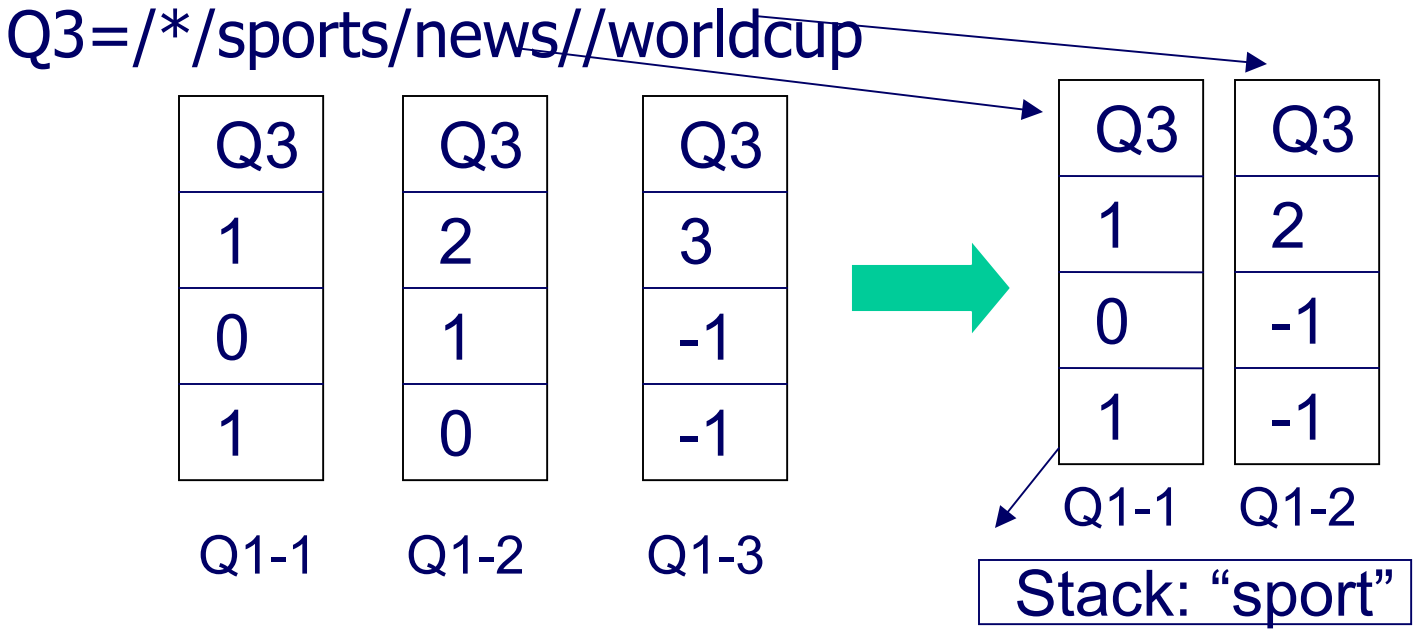
List Balancing

- For each query, we need to choose a path node to place in the Candidate List, so that the length of each list in the index will be roughly the same.
- List balancing introduce a "pivot" node

  When a new query is added to the index, the element node of the query whose entry in the index has shortest CL is chosen as pivot and placed it on the CL (instead of the 1st node)

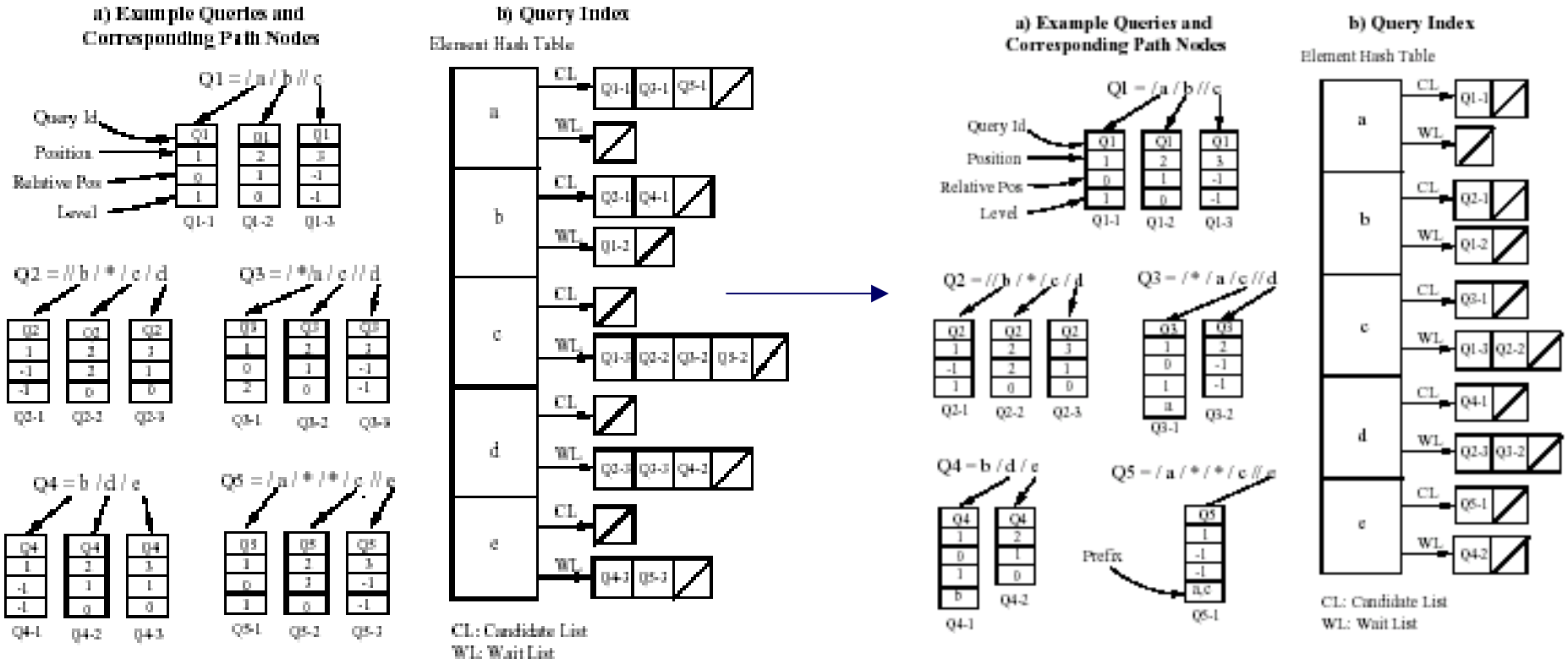
  - E.g. When a new subscriber add /sports/worldcup//news, if the length of "worldcup" element is shortest compared to "sports" and "news", "worldcup" is the pivot and added to CL. (the first node to be checked for this query)
  - The prefix "sports" will then be a precondition and use a stack to hold it, the filter will stop is the precondition for the node fails

List Balancing (cont)



- Assume the element “news” has the shortest CL among the 3 elements
- Tradeoff: Additional work needs to be done to handle the prefix “sport”

List Balancing (Cont)



Prefiltering

- Path exps are processed a level at a time.
- Unnecessary work may be done for queries that fails at later elements during evaluation.
- Prefiltering eliminates those queries containing an element name that is not present in the input document to avoid unnecessary work done
- Prefiltering is done before order and filter checking (so every incoming document is parsed twice)

Prefiltering (Cont)

- A “key” element is chosen from the element names of each query during initial parsing
- The key is chosen like List Balancing whereas a hash table(call occurrence table) containing an entry of <element name, QueryID1, …, QueryIDn> is constructed when a document arrives
- The queries referenced by the table are checked to see if all of the element names exist in the document, only the successful queries would go further

Prefiltering Example

- Assume the selected key for each query is in blue
- Q1: /sports/nba//news/scores
- Q2: /sports/NFL//news
- Q3: /sports/nba/Kings//news
- Q4: /sports//Kings/ranking

```
<sports>
<nba>
  <Kings>
    <news>Stojakovic…</news>
  </Kings>
  <Lakers>
    <news>Kings beat Lakers</news>
  </Lakers>
</nba>
</sports>
```

Sports1.xml

Element names occurring in the incoming document

sports	
nba	Q1
Lakers	
news	
Kings	Q3,Q4

Occurrence Table

All elements in Queries exists in The document? Q3

Proceed with Basic or List Balance Alg.

Performance Evaluation

- Evaluate the performance by varying:
- Number of user profiles
- Depth of subscriber queries and incoming XML documents
- Probability of wildcards
- Filter placement and selectivity
- List Balance with Prefiltering has the best performance

The End