VLDB 2007

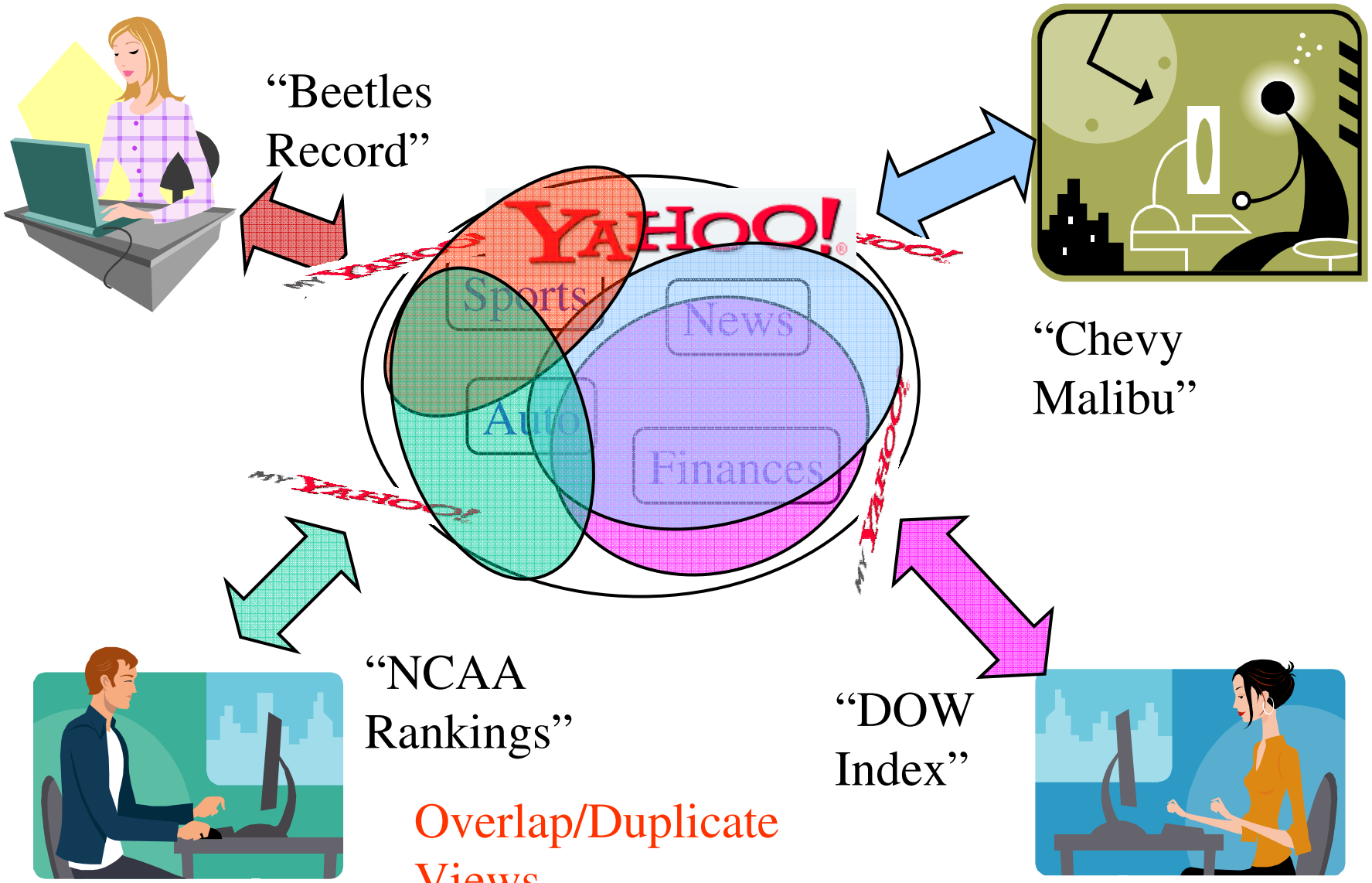
Efficient Keyword Search over Virtual XML Views

Feng Shao, Lin Guo, Chavdar Botev, Anand Bhaskar, Muthiah Chettiar, Fan Yang
Cornell University

Jayavel Shanmugasundaram

Yahoo! Research

Applications - Personal Portal



cations – Information Ir

"Vista", "budget"

"Vista" "budget"

Projects/Feedback XML View

Projects/Feedback XML View

personalized views, by privilege

project.doc (in XML format)

Email with comments on projects (in XML format)

<project>
<title>…</title>

…

<project>

<comment>

…

</comment>

<feedback>

…

</feedback>

<comment>

…

</comment>

Keyword Search over XML View

- *Materialized* XML Views?
  - Similar to keyword search over XML documents
    - ❖Many well-studied algorithms
    - ❖Materialize views when loading documents
  - Not applicable in emerging applications!
    - ❖Overlap/Duplicate/Update overhead
    - ❖View definitions not known a-priori

- →Keyword Search over *Virtual* XML Views

Related Work

- Scoring and Indexing in IR community
  - DBXplorer [Agrawal02], Banks [Bhalotia02], ObjectRank [Balmin04], XRank [Guo02], Discover [Hristidis 02]
  - Work with materialized documents
- Integrating keyword search and structural queries
  - GTP [Chen 03], TermJoin [Khalifa 03]
  - Access base data to evaluate the view
- Projecting XML documents [Marian 03]
  - Access base data; not leveraging indexes

Outline

- Motivation
- **Problem Definition**
- High-level Overview
- PDT Generation Algorithm
- Experimental Results
- Conclusion

Problem Definition

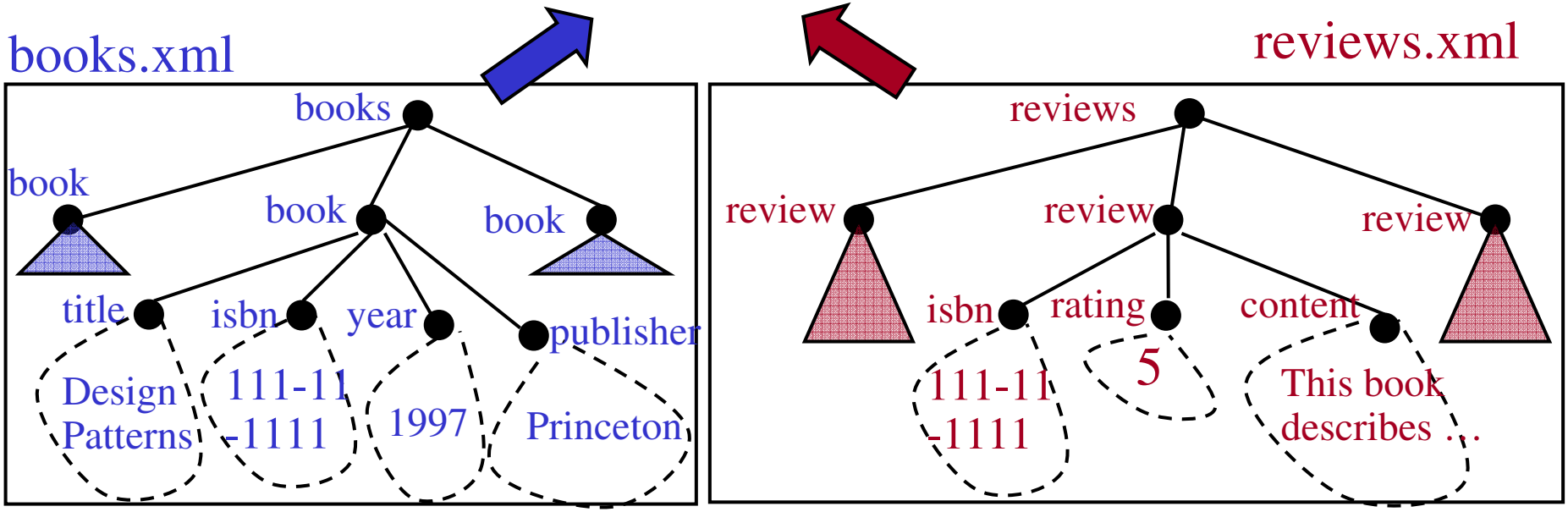
- Ranked Keyword Search over Virtual XML Views
  - Input: a set of keywords Q = {k1, k2, …, kn}, an XML view definition V over an XML database D
  - Output: k view elements with highest scores
  - TF-IDF scores
    - TF(k, e): # occurences of the keyword k in an element e
    - IDF(k): the inverse of # of elements containing k
    - Score(e, Q) = Σ_iTF(k_i,e) * IDF(k_i)
    - Score(e, Q) is further normalized by the length of the view elements

Running Example

Virtual View

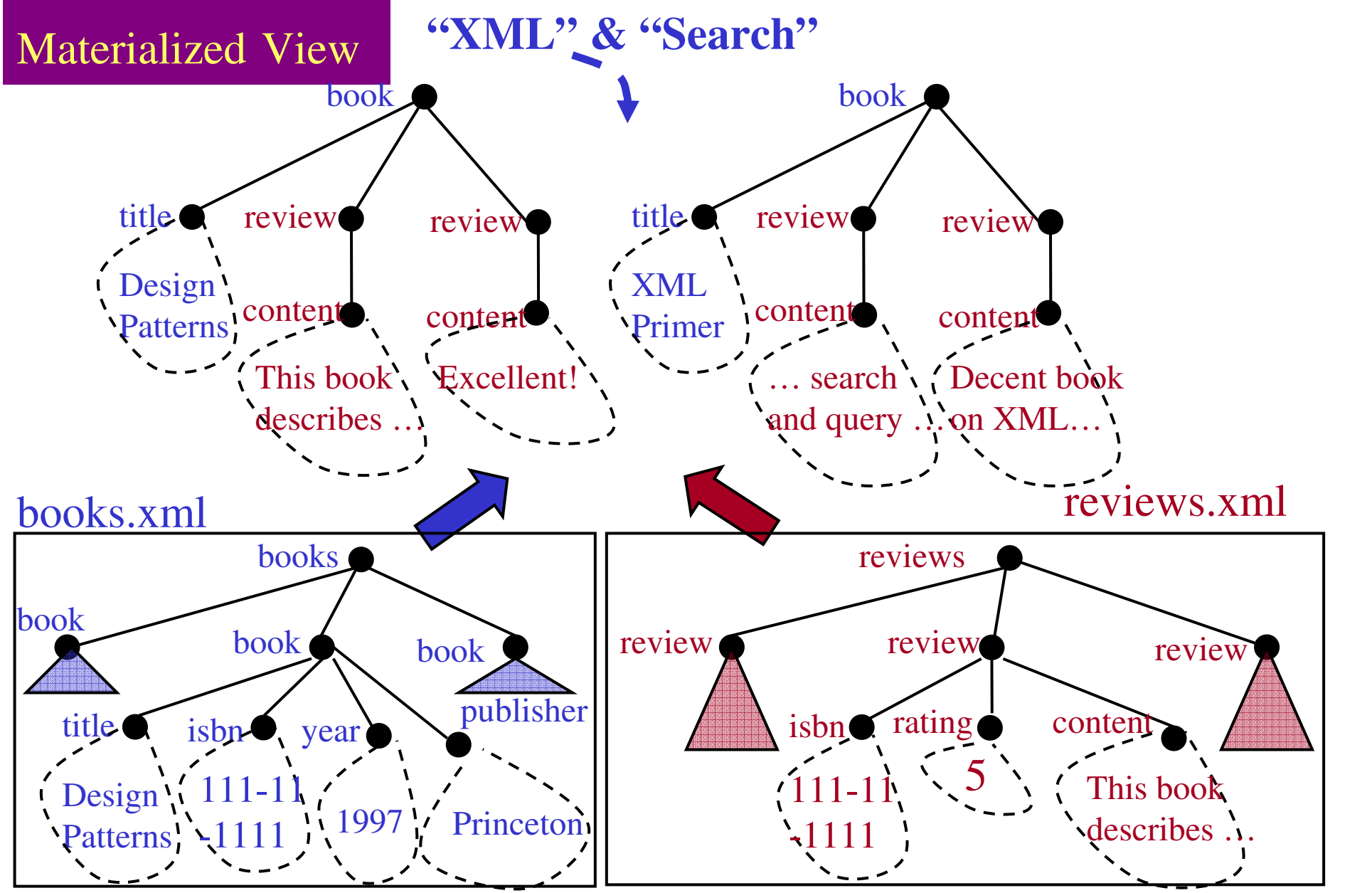
"XML" & "Search"

```
for $book in fn:doc(books.xml)/books//book
where $book/year > 1995
return <book> $book/title
            for $review in fn:doc(reviews.xml)/reviews//review
            where $review/isbn = $book/isbn
            return   <review> $review/content </review>
          </book>
```



Running Example

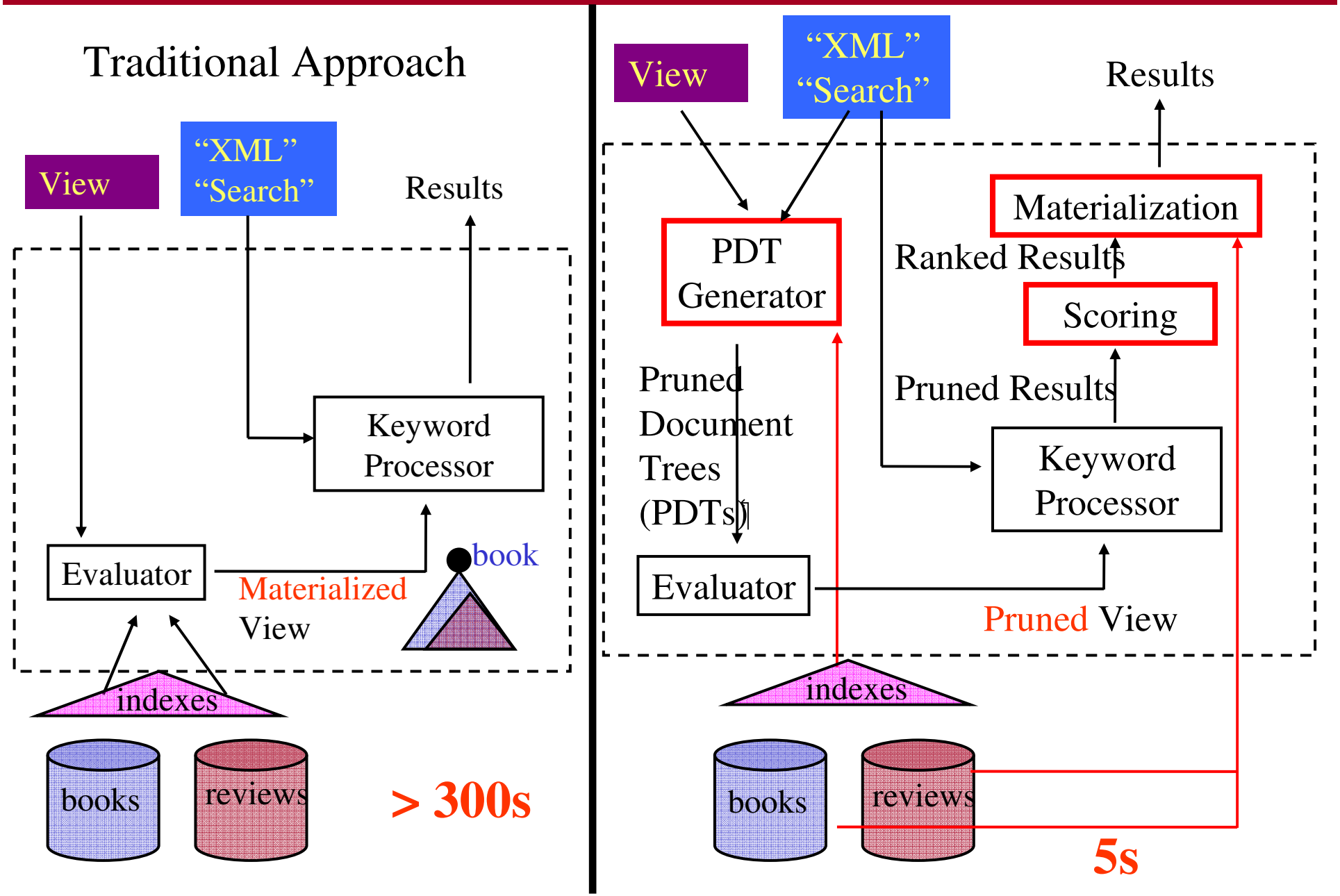
Materialized View



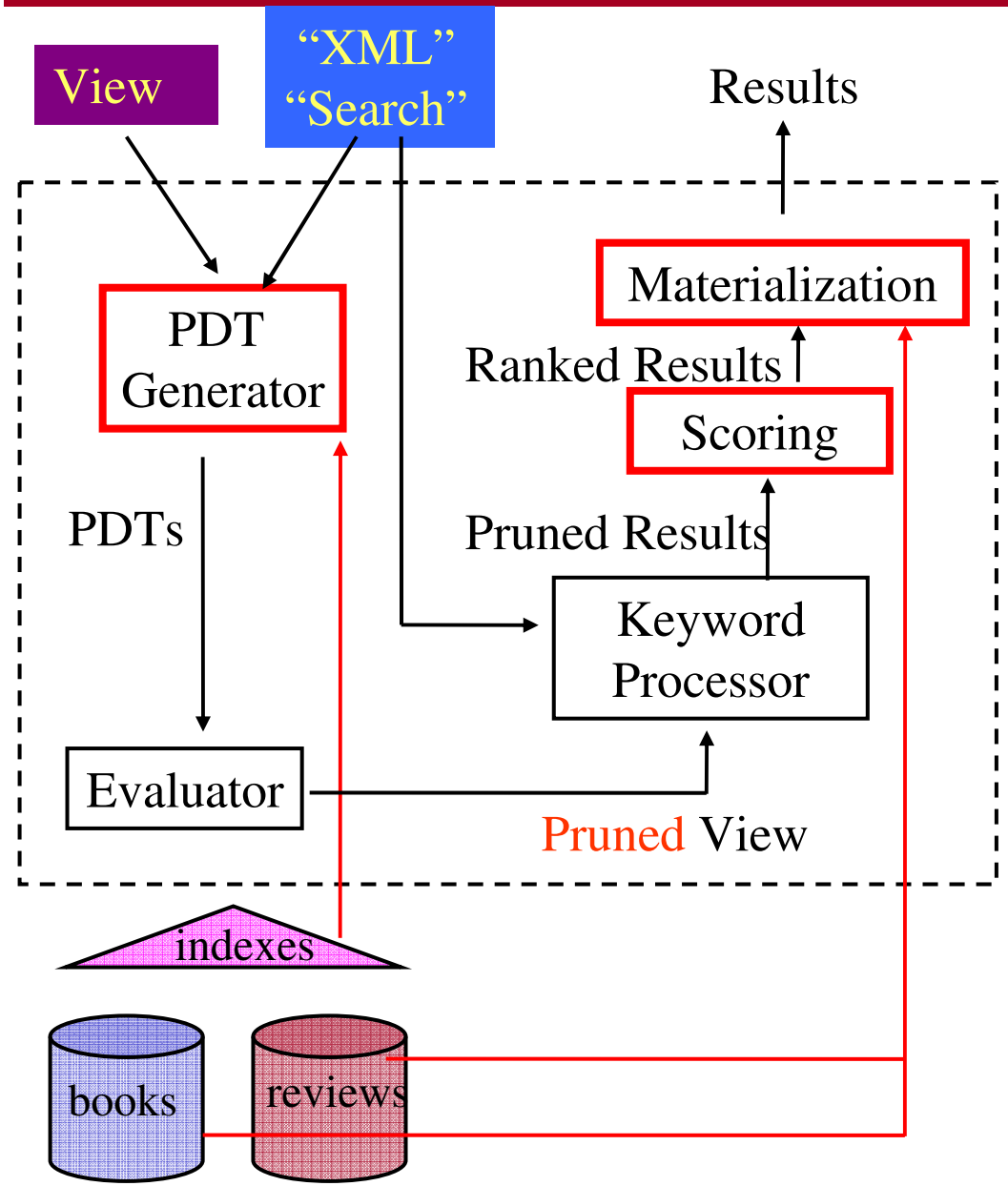
Outline

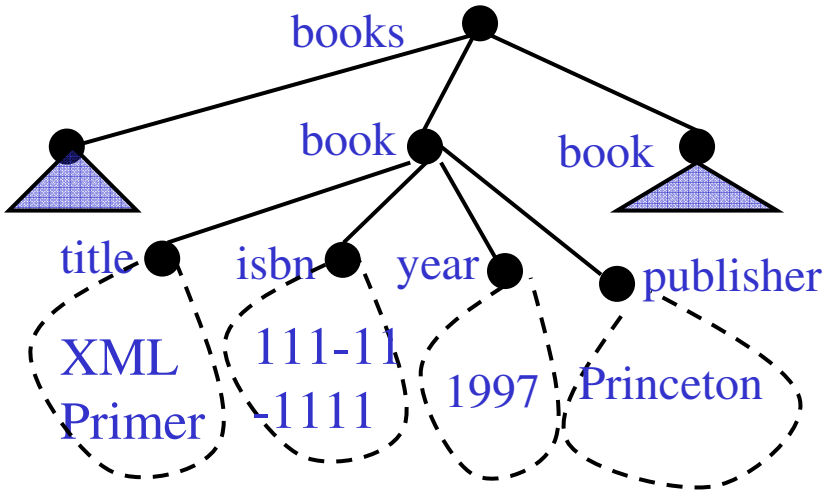
- Motivation
- **Problem Definition**
- **High-level Overview**
- PDT Generation Algorithm
- Experimental Results
- Conclusion

Our Approach

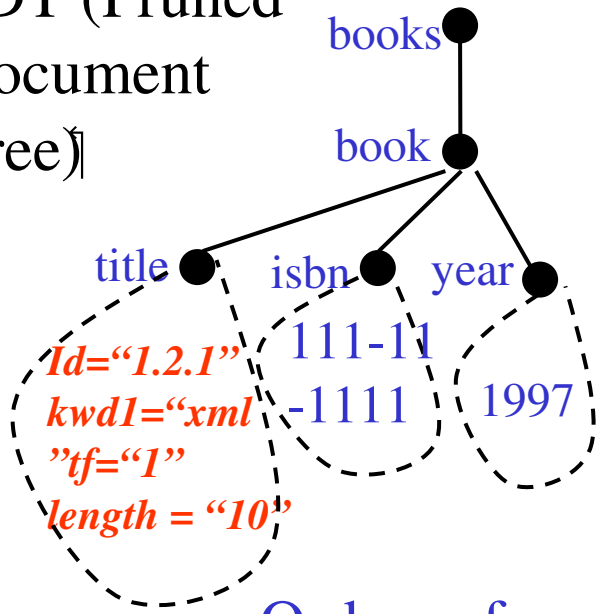


Our Approach



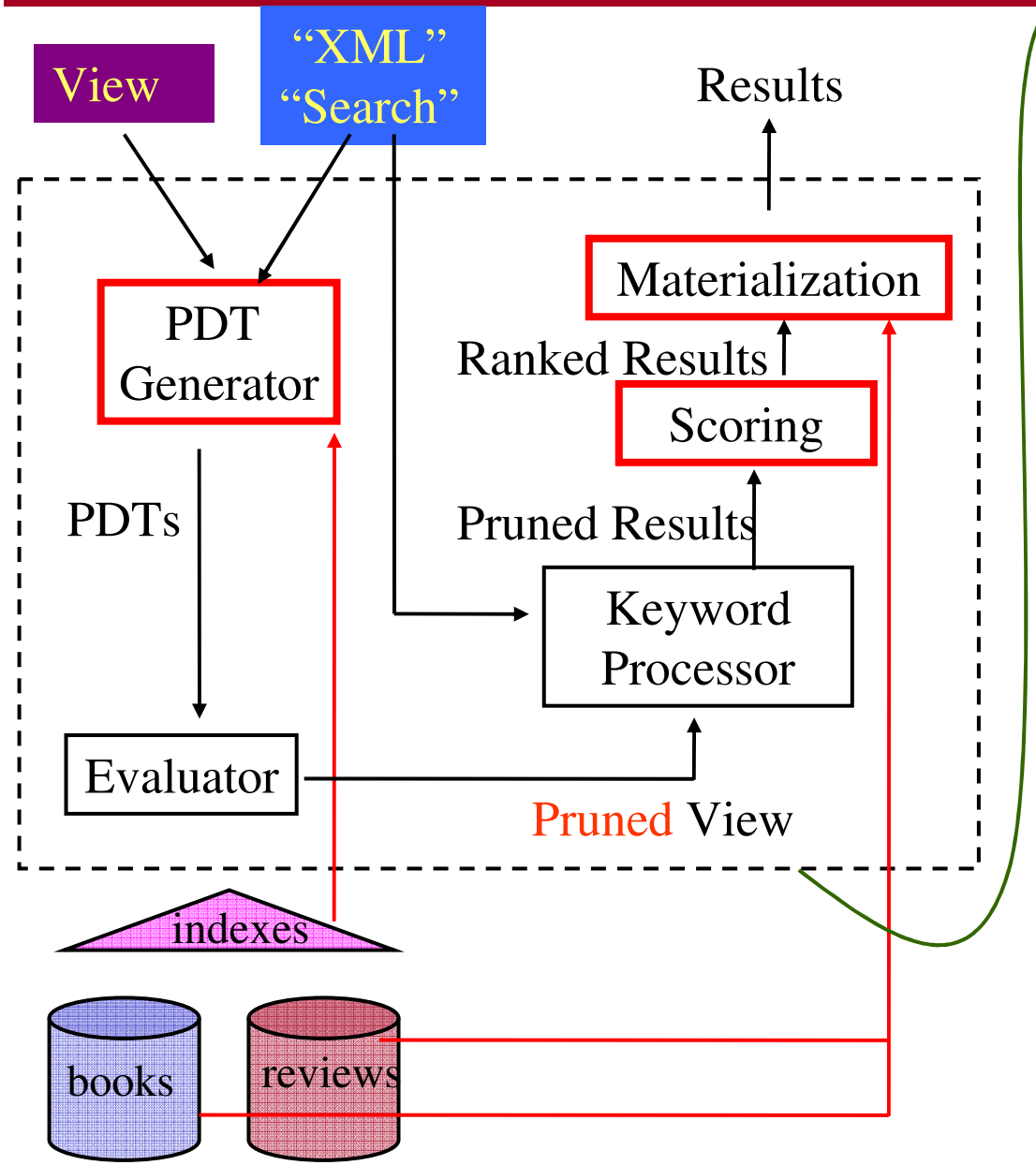


PDT (Pruned Document Tree)



Orders of magnitude smaller!

Our Approach -- Challenges



1. Joining books & reviews requires isbn (data value)

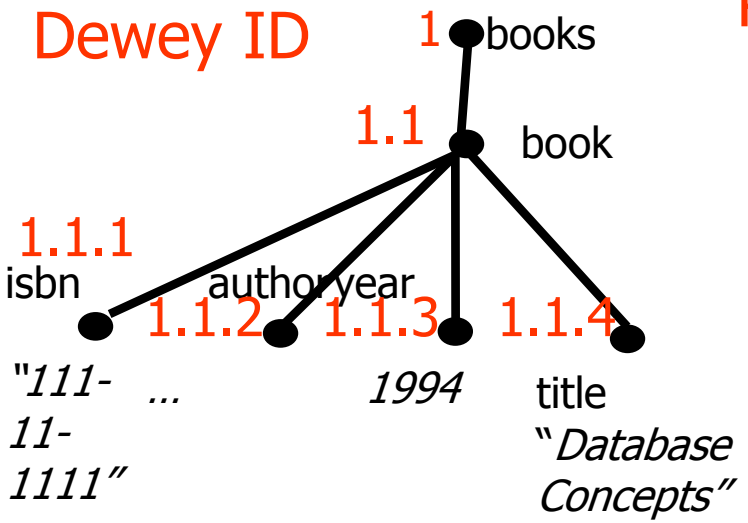
   -- how to get data values without accessing the base data?

2. Scoring view elements requires aggregate statistical data (e.g., tf from book and review)?

   -- How to collect them without materializing the view elements?

Dewey ID and Index

Dewey ID

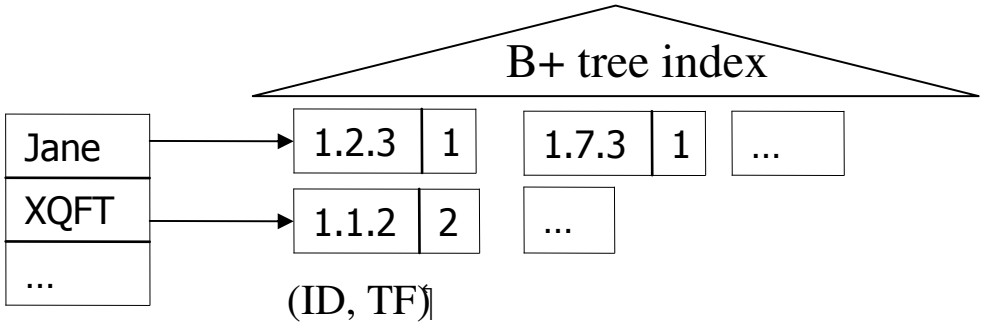


Path Index – [Chen05], [Yoshikawa01]

B+-Tree

PathID	Value	IDList
...	...	...
/books/book/isbn	"111-111-1111"	1.1.1,1.2.1
/books/book/isbn	"222-222-2222"	1.2.1
...	...	...
/books/book/author/fn	"Jane"	1.2.3, 1.7.3

Inverted Index: [Guo03],[Botev05]

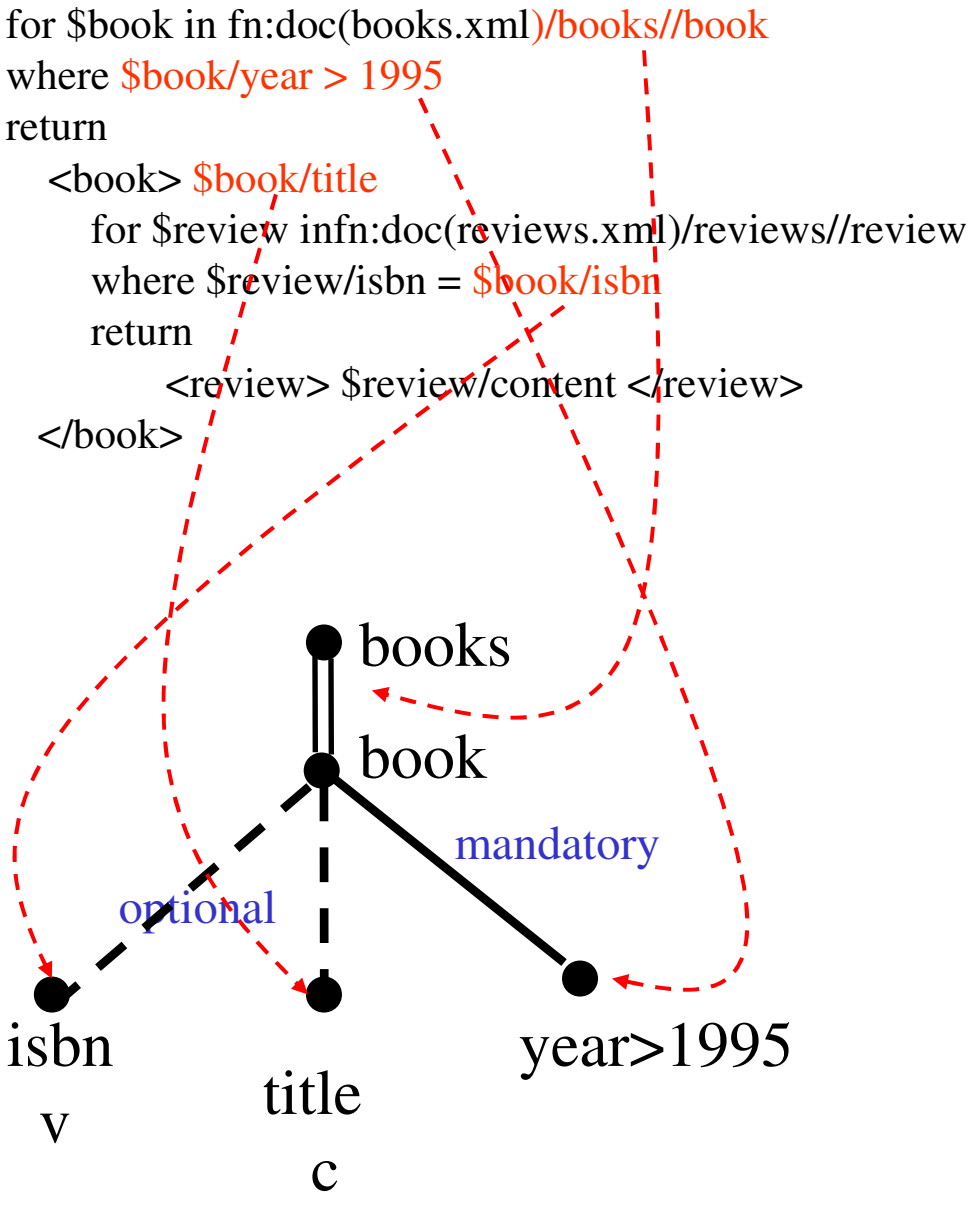


Outline

- Motivation
- Problem Definition
- High-level Overview
- **PDT Generation Algorithm**
- Experimental Results
- Conclusion

XML View → Query Pattern Tree (QPT)

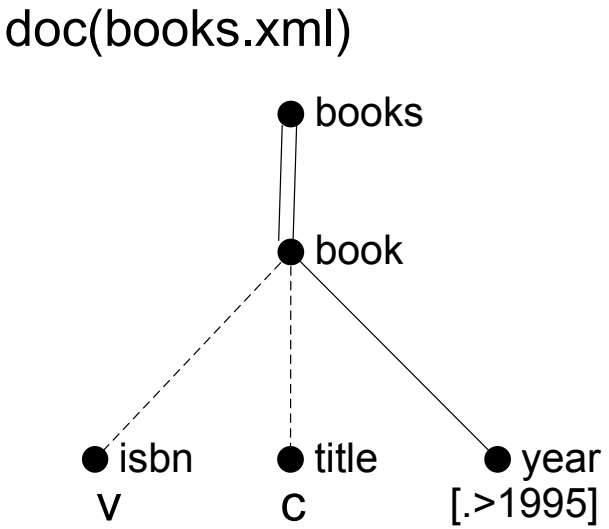
- Similar to GTP, proposed by Chen 2003 for normal query evaluation
  - Captures the structural parts required by queries
  - Mandatory/Optional edges
- New features
  - Node annotations
    - V: value required to evaluate the view
    - C: content used in the view

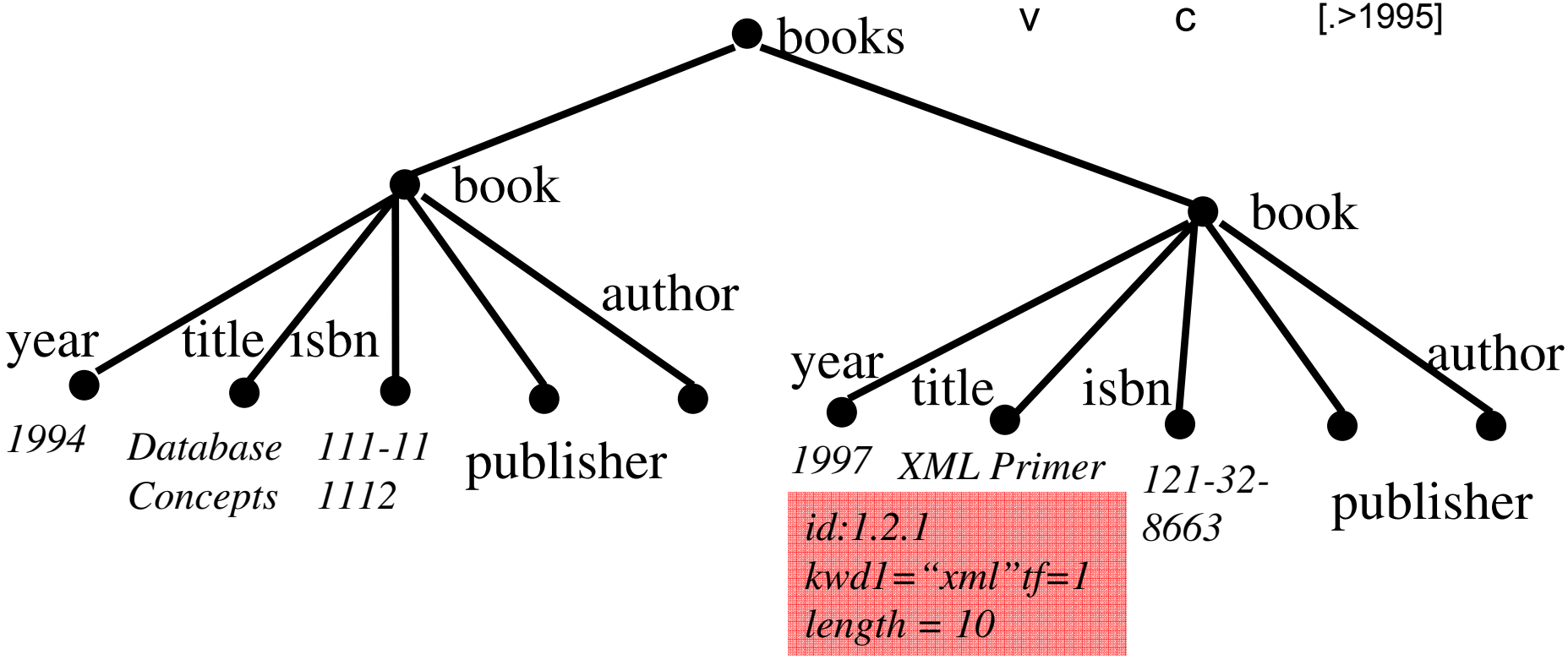


PDT Intuition

- Restrictions enforced by QPT

  Predicate Restriction
  Descendant Restriction
  Ancestor Restriction

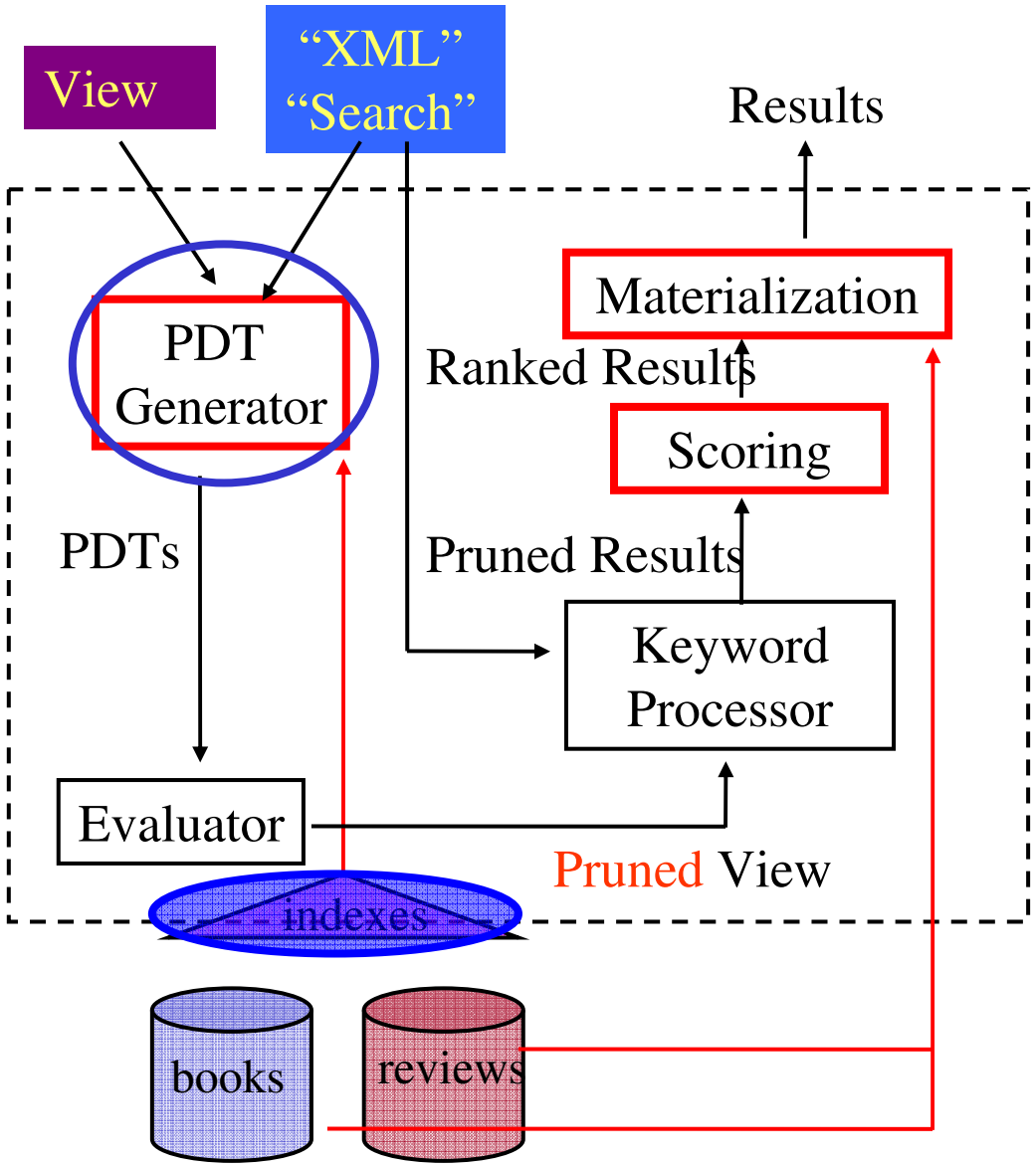




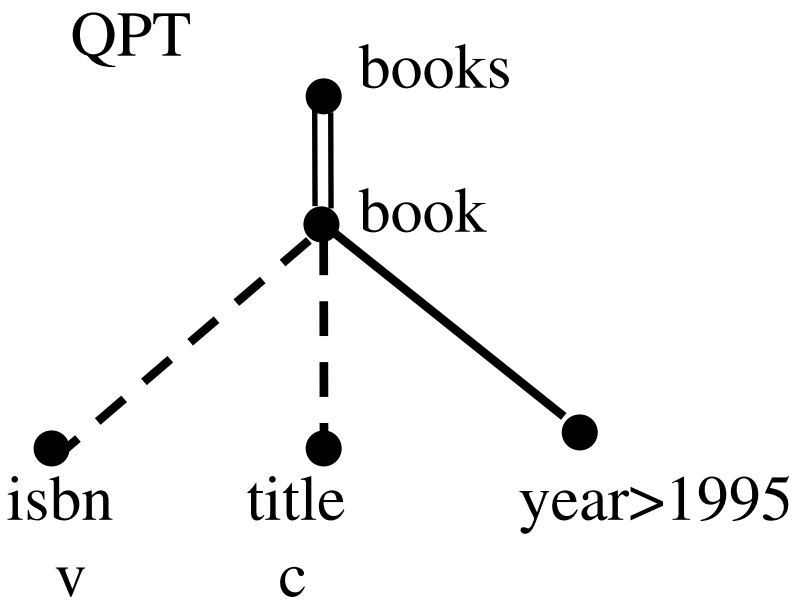
PDT Generation

1. Get ID lists for paths in the QPT

2. Merge IDs in the lists to create the PDT



Step 1: Get List of IDs



B+-Tree

PathID	Value	IDList
…	…	…
/books/book/isbn	“111-11-111”	1.1.1
/books/book/isbn	“121-23-1321”	1.2.1
…	…	…
/books/book/author/fn	“Jane”	1.2.3, 1.7.3

Key idea: for each node without mandatory child edges, obtain the corresponding list of ids

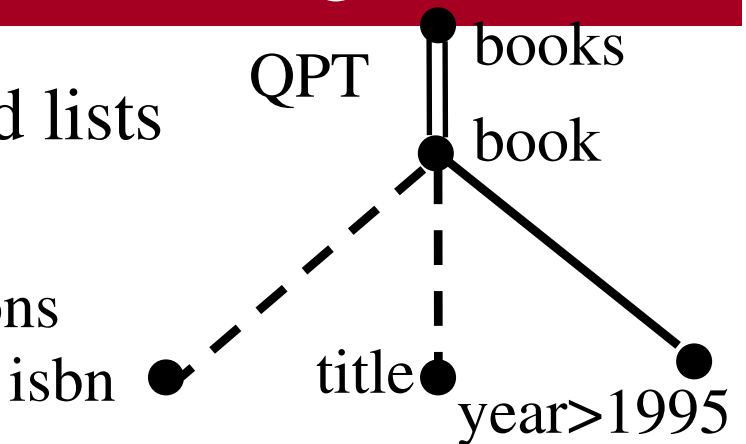
books//book/isbn: (1.1.1:”111-11-111”),(1.2.1,”121-23-1321”)

books//book/title: 1.1.4, 1.2.3, 1.9.3

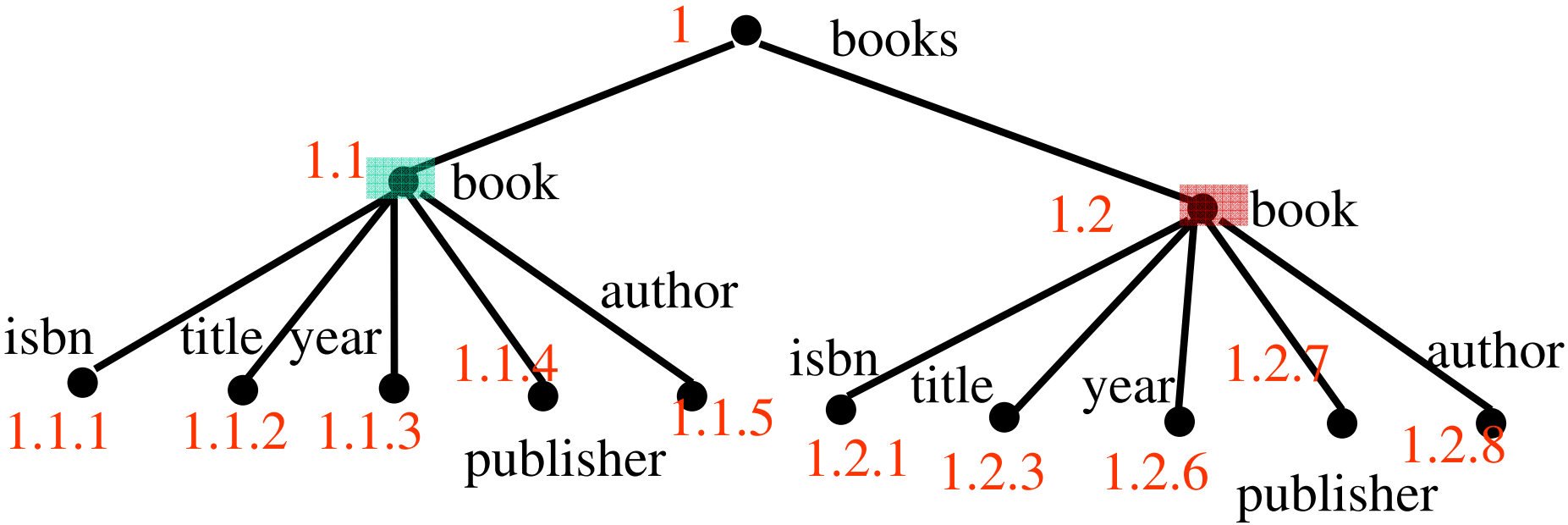
books//book/year: (1.2.6, 1.5.1:”1996”), (1.6.1:”1997”)

Step 2: Merging IDs -- Challenges

- Makes a single pass over relevant id lists
  - Flat indices → nested structure
  - Enforce ancestor/descendant restrictions

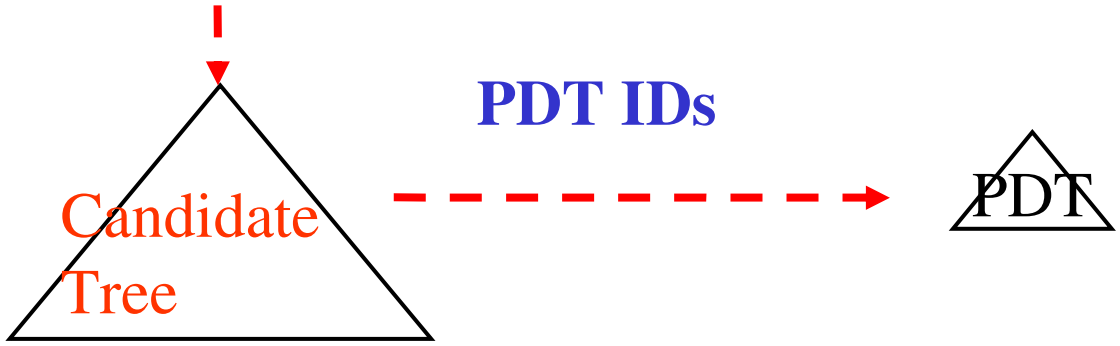


(books//book/isbn, (1.1.1: “111-11-1111”), (1.2.1: “121-23-1321”),...)
(books//book/title, 1.1.4, 1.2.3, 1.9.3, …)
(books//book/year, (1.2.6, 1.5.1: “1996”), (1.6.1:”1997"), …)



PDT Generator – Merging IDs

(books//book/isbn, (1.1.1: “111-11-1111”), (1.2.1: “121-23-1321”),...)
(books//book/title,1.1.4, 1.2.3, 1.9.3, …)
(books//book/year, (1.2.6, 1.5.1: “1996”), (1.6.1:”1997"), …)

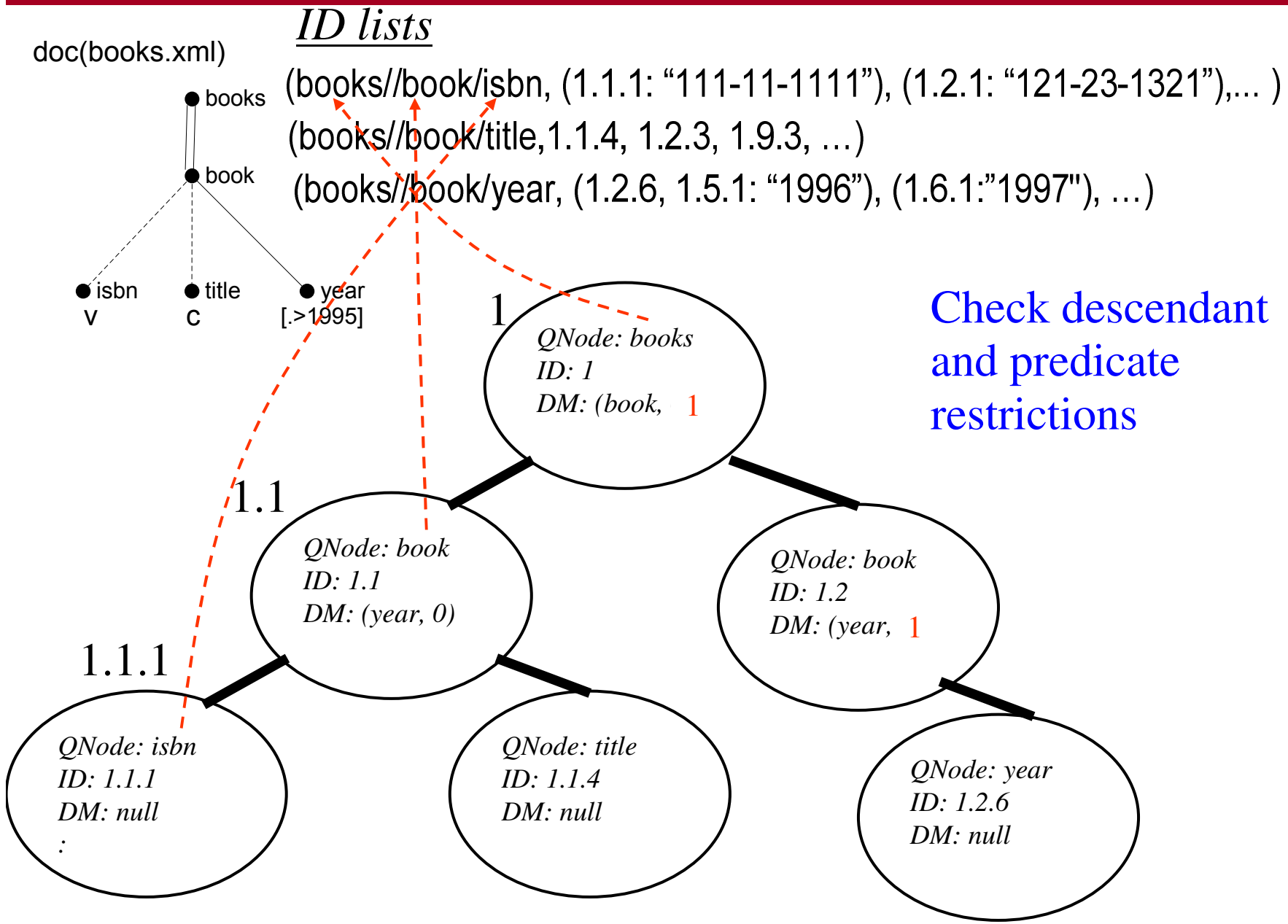


Idea: a loop that merges ids in the lists, and creates the CT nodes in dewey id order

At each step, we check the min id in the CT

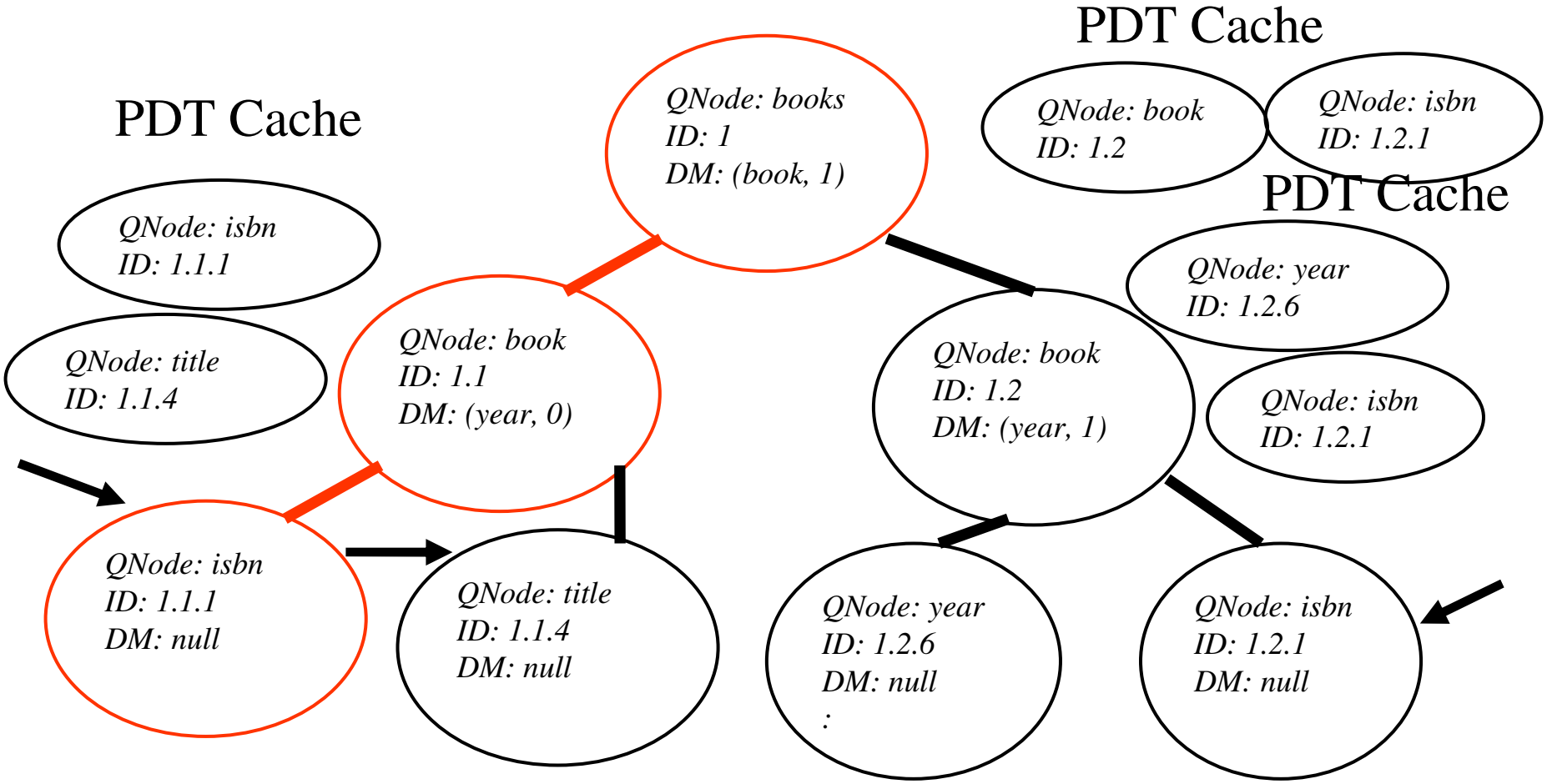
- if satisfies all restrictions → PDT
- if satisfies descendant restriction and not ancestor → PDT Cache
- if not satisfies descendant restriction and does not have child node in the CT → Discard

Adding CT Nodes from Top Down



Removing CT Nodes from Bottom Up

- Try to determine if a node should be in the PDT: check ancestor constraints
  - Remove IDs known to be non-PDT nodes
  - Nodes in the PDT cache – defer checking ancestor restrictions



Correctness and Complexity

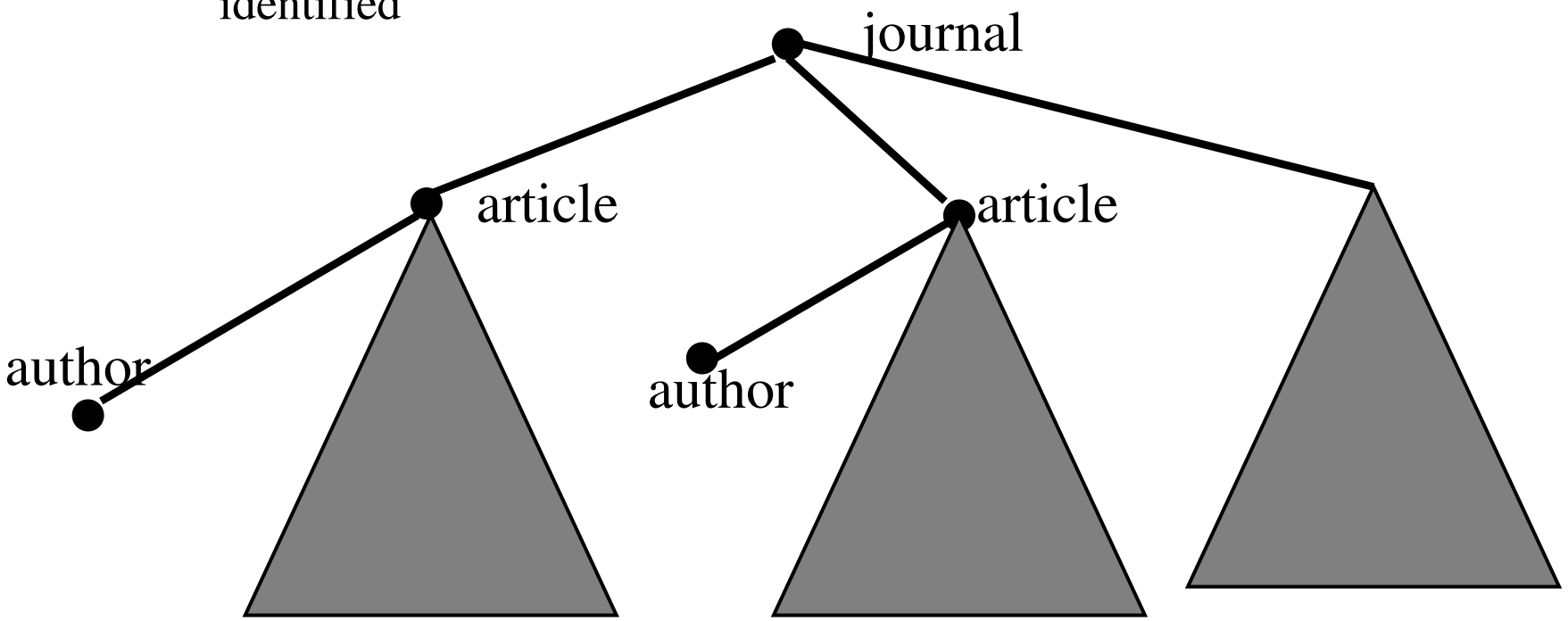
- Theorem (Informal)
  - Given a set of keywords, an XQuery view and a database,
    - The result sequence, after being materialized, are identical to as if the view was materialized
    - The byte lengths of each element are identical
    - The TFs of each keyword in each element are identical
  - Formal proof in the technical report
- Complexity: polynomial with respect to the number of IDs, the length of paths, the depth of the documents, and the number of keywords

Outline

- Motivation
- Problem Definition
- High-level Overview
- **Evaluation Algorithm**
- **Experimental Results**
- Conclusion

Experiments

- Real-world INEX data
  - 500MB
  - Publications with author information and others
  - View: nested articles under authors.
    - Only require author names when evaluating the view
    - Article content (huge) only required after the top k results are identified

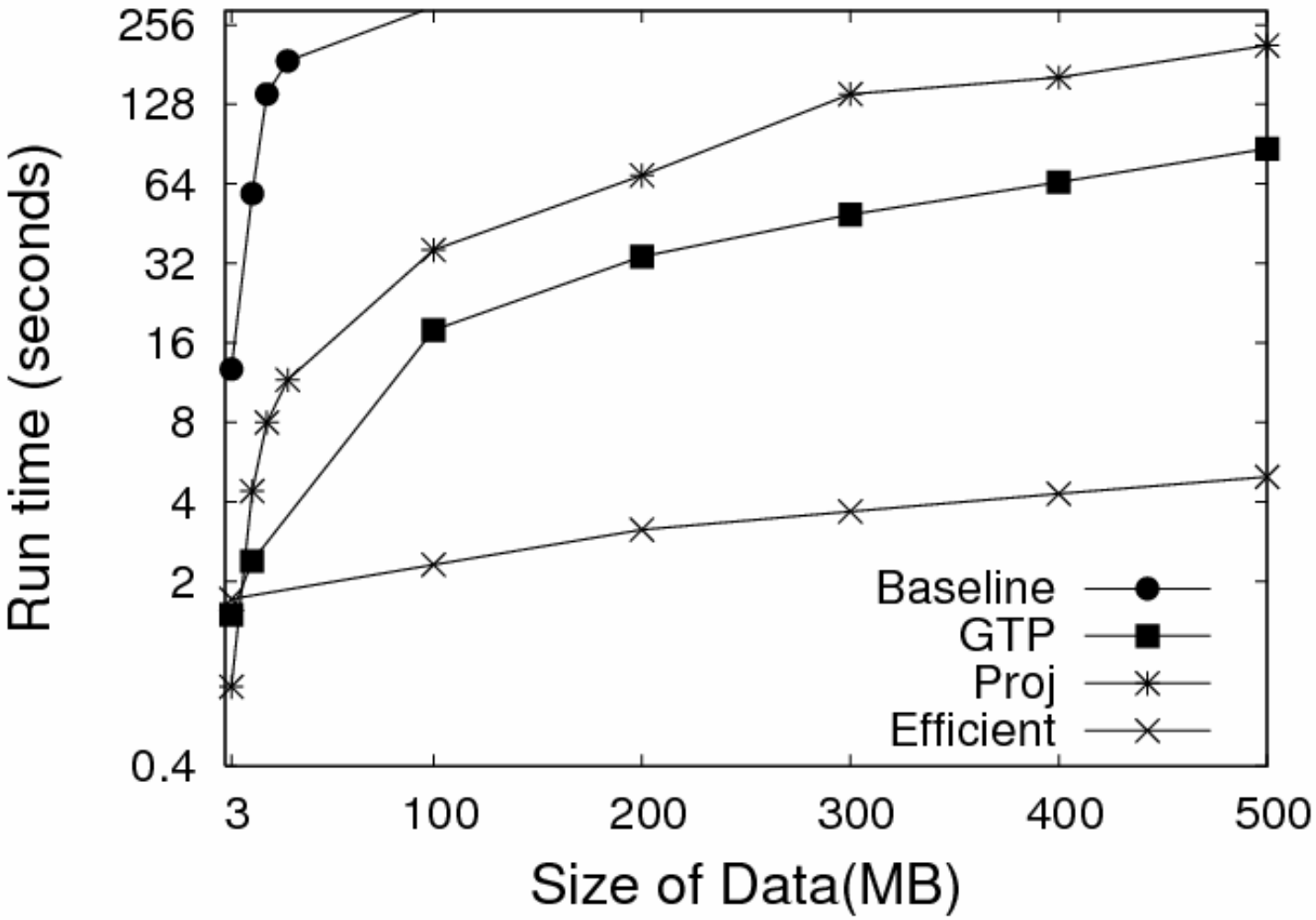


Experiments

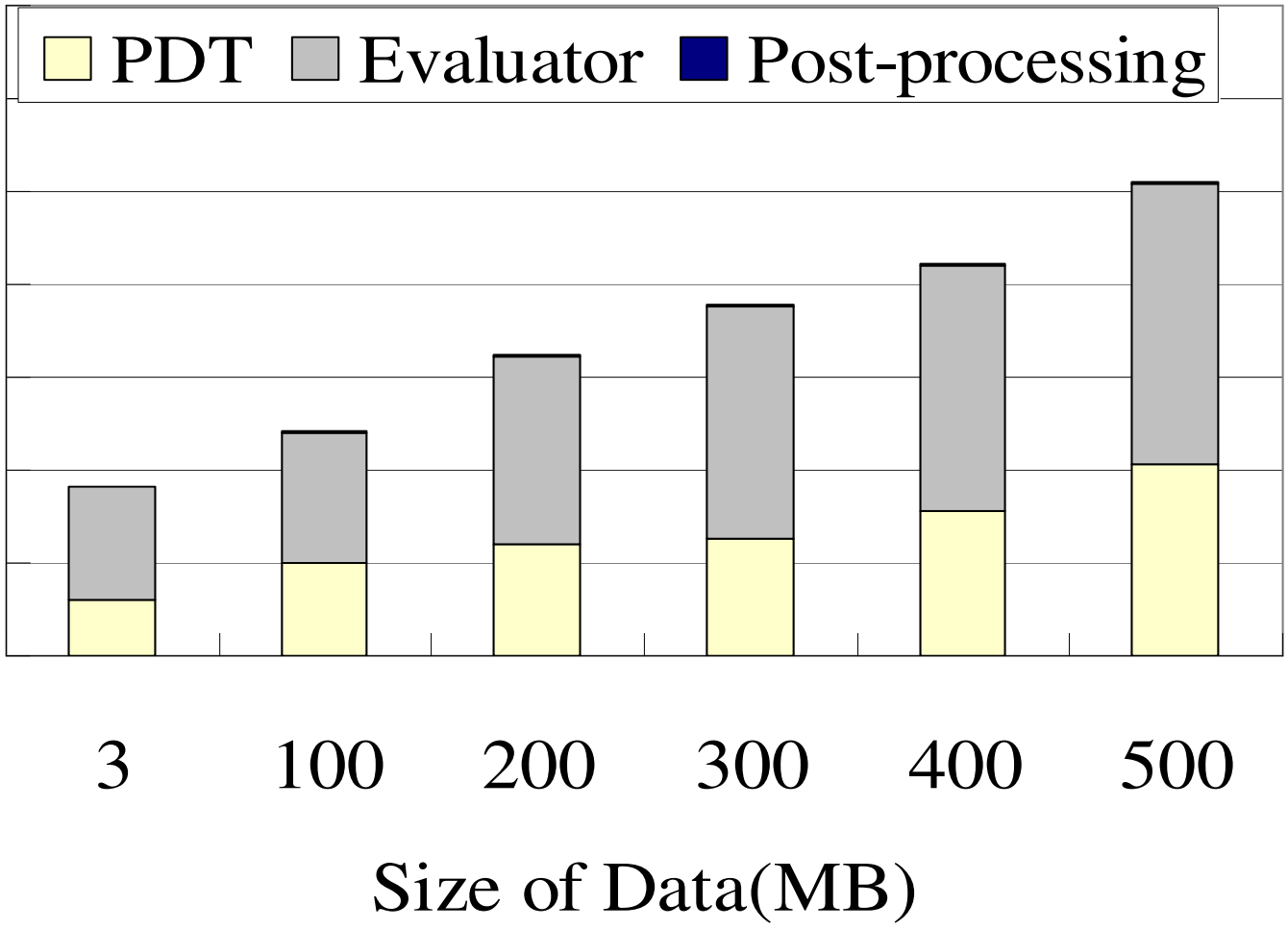
- Setup
  - 3.4Ghz CPU, 2GB Mem
  - Windows XP
  - Implemented in C++

- Alternatives
  - Baseline: materialize all view results on the fly
  - Timber (GTP [Chen 03] + TermJoin [Khalifa 03])
    - not tokenized but still access base data to evaluate the view
  - Proj [Marian 03] : access base data to produce PDT

Varying size of data



Varying size of data



Outline

- Motivation
- Problem Definition
- High-level Overview
- **Evaluation Algorithm**
- Experimental Results
- **Conclusion**

Conclusion

- A system architecture for keyword search over virtual XML views
- Novel algorithms to generate pruned data relevant to XML view
- Implemented, and experimentally evaluated
  - 10 times faster than other alternatives
- Future work
  - Top-K keyword search queries
    - Our approach returns pruned version of “all” elements, which is unnecessary
    - Returns most relevant results only
  - QPT/PDT may be adapted for normal query evaluations

Optimizations and Extensions

- Extensions
  - One ID corresponds to more than one QPT nodes
    - //a//a → /a/a/a
    - QNode → QNodeSet
- Optimizations
  - Currently lazy checking of ancestor restrictions
    - Can check in top down phase, and save memory usage of pdt cache
  - PDT nodes are output not in document order
    - Can enforce document order

Complexity

- $O(Nqdf+Nqd^2+Nd^3+Ndkc)$
  - N: # of IDs in the lists
  - q: the depth of the paths
  - d: the depth of the documents
  - k: the number of keywords
  - c: unit cost of inverted list access

  - $Nqdf+Nqd^2$: cost of top down processing
  - Nd^3: cost of bottom up processing
  - Ndkc: cost of inverted list access