# Supporting Branched Versions on XML Documents

Zografoula Vagena
MIRELLA M. MORO
Vassilis J. Tsotras

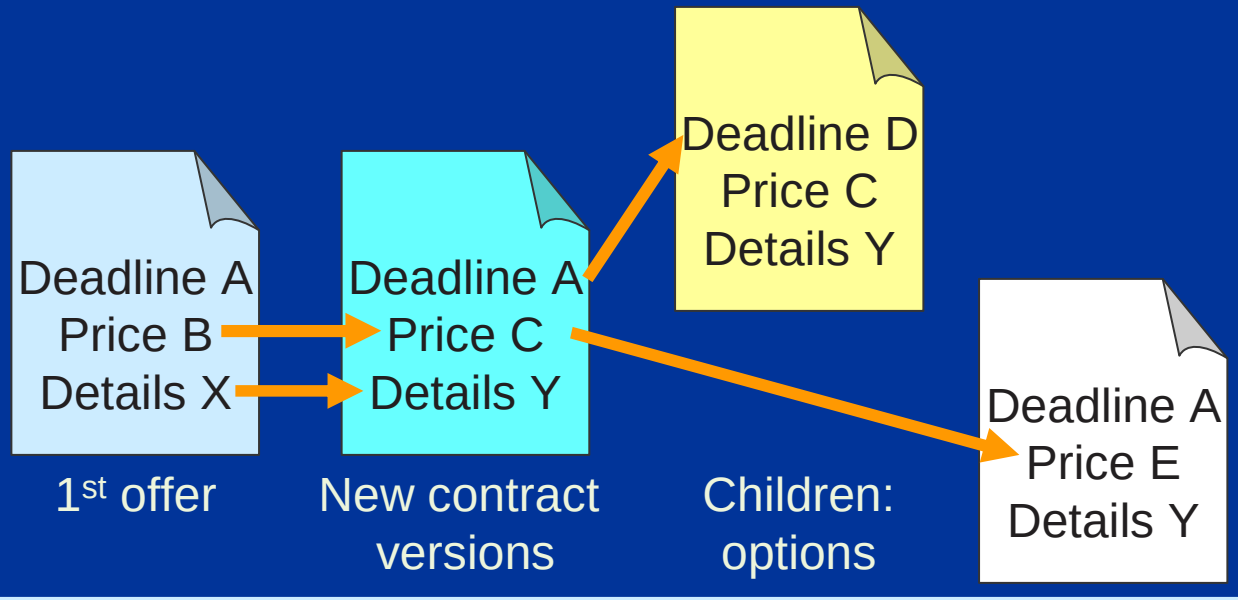
University of California, Riverside

# Outline

- Motivation
- Current approaches
- Our proposal
  - Data structures
  - Optimizations
  - Version Range PathStack
- Experimental evaluation
- Conclusions

# Motivation

- Example: Business negotiation process
- Sequence of messages between parts to agree on a business contract



# Motivation

- XML: standard for authoring, storing, exchanging and presenting electronic documents
- Document management:

  storing and querying current and past state of documents = versions
- Need to identify and return parts of document version(s) as a response to a user query

# Current Approaches

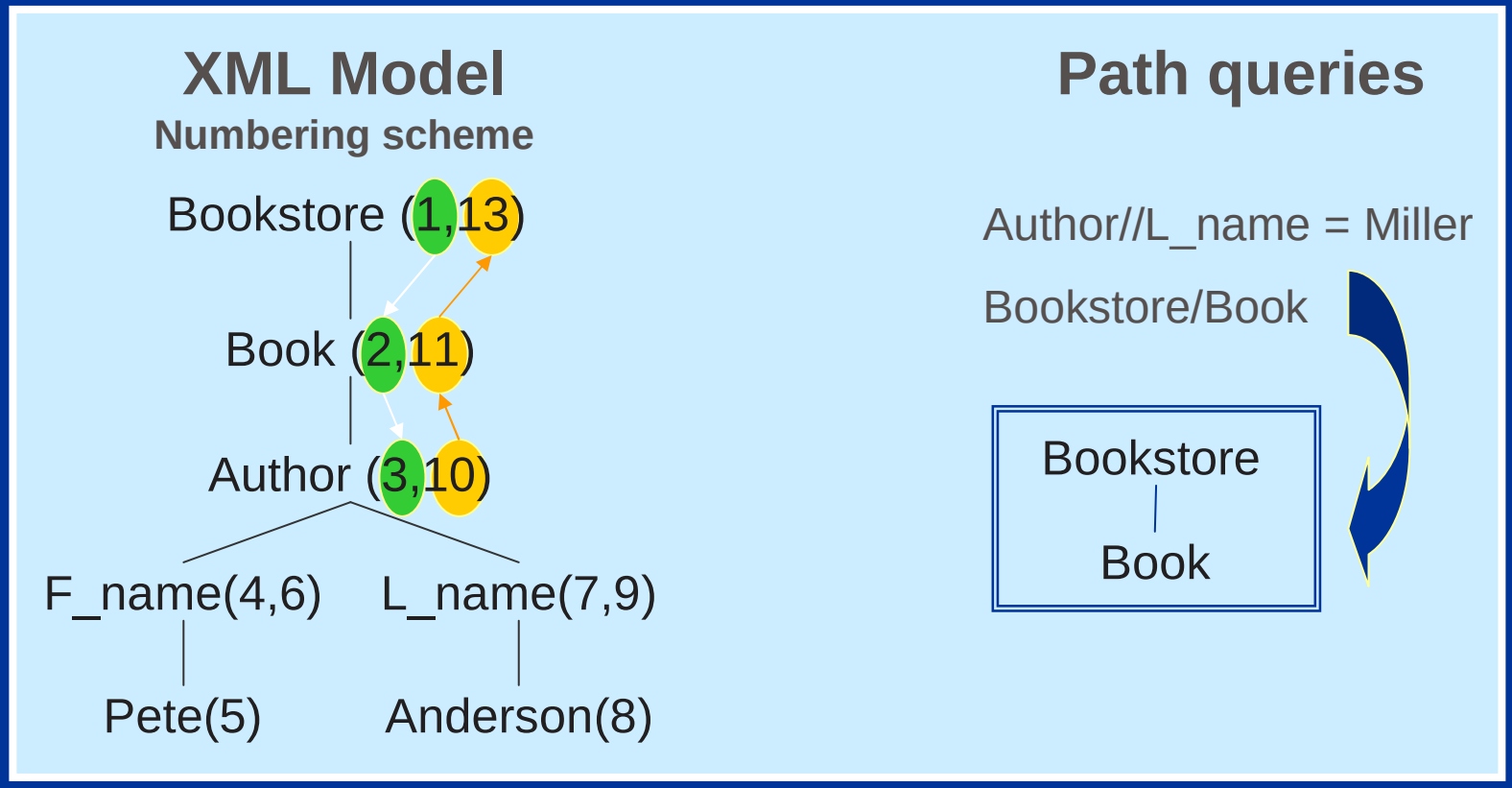
- **Problem**
  - Holistically answer path queries over (multiple) branched versions of an XML archive
  - Consider range of versions

- **Current approaches**
  - Snapshot approach

    - Fast query performance
    - High storage space
  - Log approach

    - Minimal space requirement
    - Slow query performance

# Current Approaches




- Evaluation of general path expressions has been proposed only for the static, non-versioned case and, more recently, in the linear versioning case

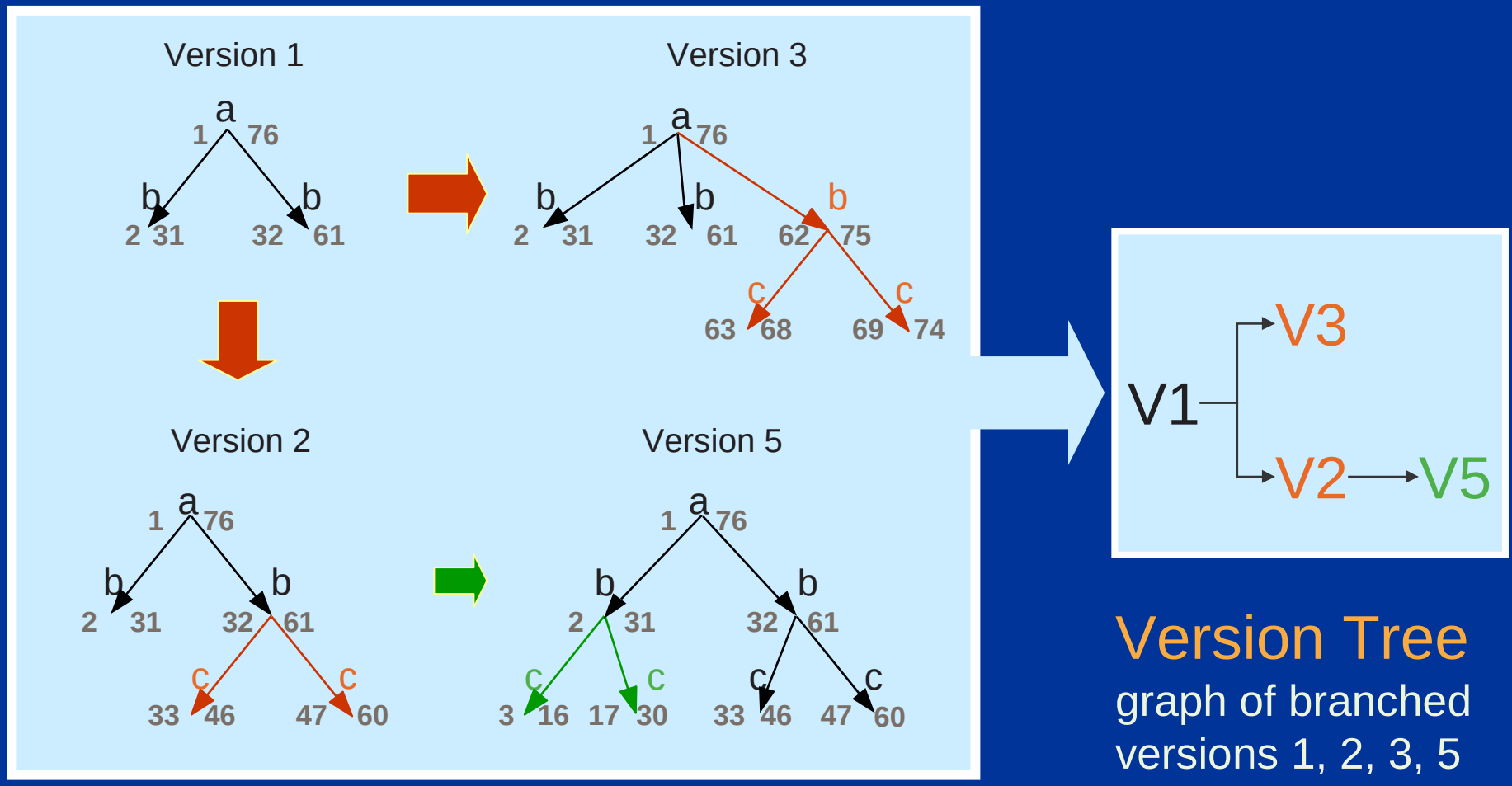
# Our Proposal

- Storage technique for versioned archive with small replication:

  Version Tree + BT-ElementList

- *Version Range Pathstack* for path expression over multiple versions
  - Versions in the same path
  - Versions that share same parent

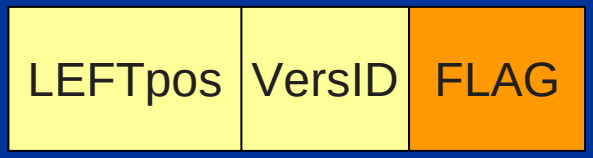
# 1. DATA STRUCTURES



# 1. DATA STRUCTURES

- Data records: document nodes

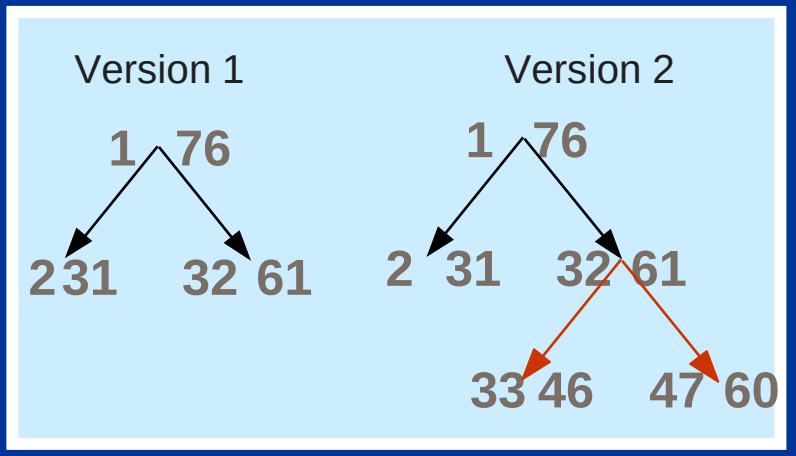

- When node is deleted
  - Tombstone record

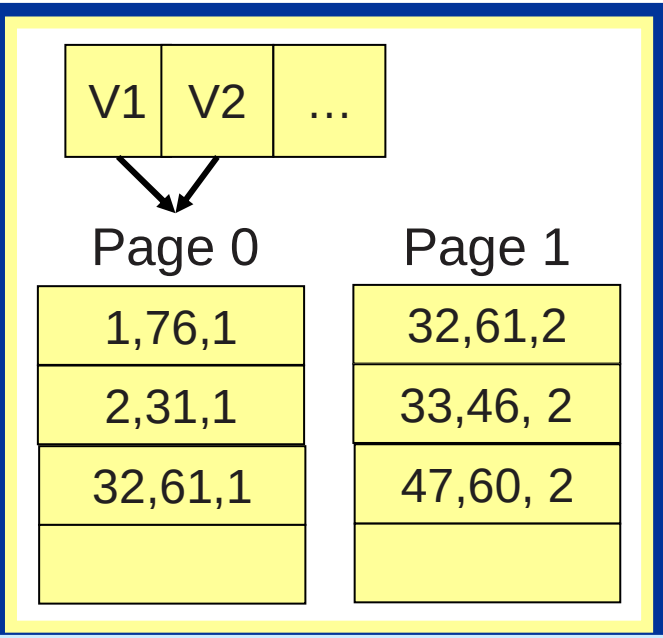


# 1. DATA STRUCTURES

- BT-ElementList: document nodes organized in element lists
- Input = element lists,

  one for each distinct type, sorted by leftPos number
- Branch-Version each list
- To reconstruct a particular document version: merge scan over the element lists as of that version



# 2. OPTIMIZATIONS

- Store in the same page as many versions from the same path as possible
- Using the version tree, allow less replication

# 2. OPTIMIZATIONS

## New version split

| Page 0 | Page 1 | Page 2 | Page 3 | Page 4 | Page 5 | Page 6 |
|---|---|---|---|---|---|---|
| 1,1,... | 1,2,... | 32,2,... | 1,3,... | 3,3,... | 1,4, ... | 32,4,... |
| 2,1,... | 2,2,... | 33,2,... | 2,3,... | 17,3,... | 2,4,... | 62,4,... |
| 32,1... | | 47,2,... | | 32,3... | | 63,4,... |
| | | | | | | 69,4,... |

- Reduces replication between ancestor-descendant versions in the version-tree

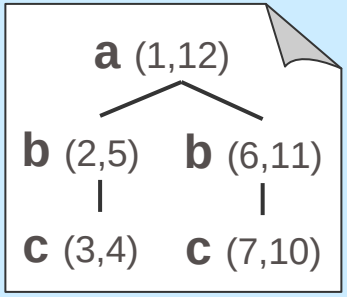
| Page 0 | Page 1 | Page 2 | Page 3 |
|---|---|---|---|
| 1,1,… | 32, 2 ,… | 3, 3, … | 32, 4, … |
| 2,1,… | 33, 2,… | 17, 3, … | 62, 4, … |
| 32,1 ,… | 47, 2,… | 32, 3, … | 63, 4, … |
| | | | 69, 4, … |

# ORIGINAL PATHSTACK

- PathStack: an optimal, holistic approach
  - Input = element lists,
    - one for each distinct type, sorted by leftPos number
  - Scan input data in document order only once,
    - merge-join fashion
  - In-memory stacks: keep total and partial query answers
    - discard elements when they are no longer needed
  - It would require a stored snapshot of each version, prohibitively large space
  - It does not deal with branched versions

# ORIGINAL PATHSTACK

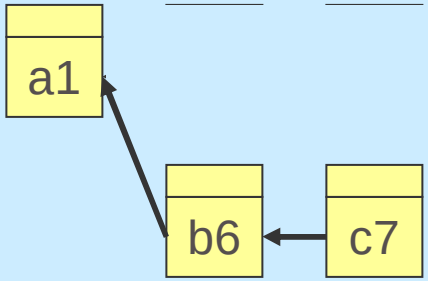
Document fragment



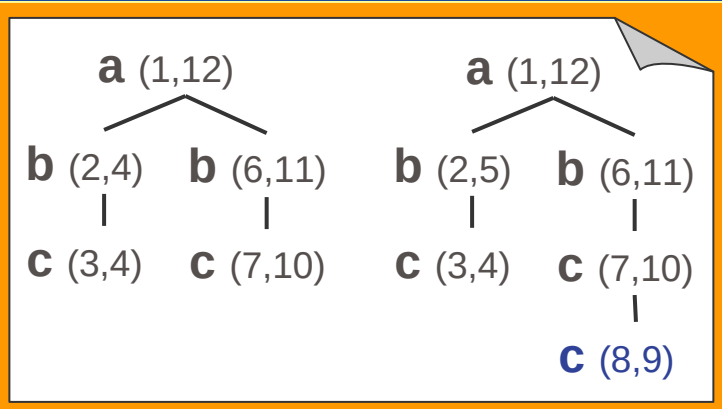

Query a /b /c

Input a(1,12)
b(2,5), b(6,11)
c(3,4), c(7,10)

Stacks



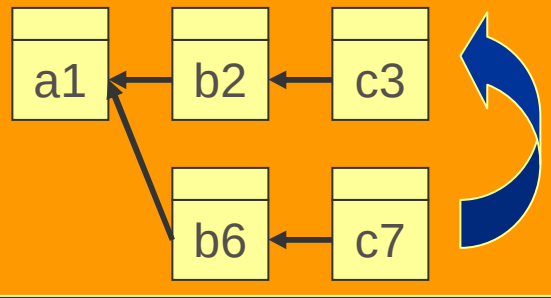

Document fragment




Input a(1,12)
b(2,4), b(6,11)
c(3,4), c(7,10)

a(1,12)
b(2,5), b(6,11)
c(3,4), c(7,10), c(8,9)

Stacks



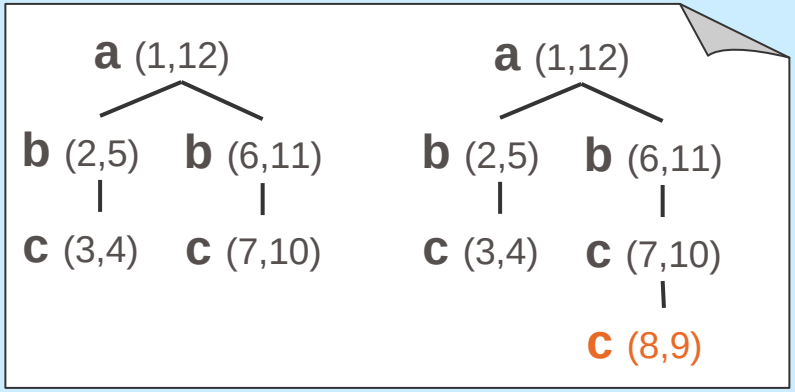
# 3. VERSION RANGE PATHSTACK

- Input = path expression + version ids to be queried
  + BT-element list
  - Each record may belong to more than 1 version
  - PathStack: but input BT-element list

- Use method multiple_versions_getMinSource()
  to identify the next element in document
  over the union of versions in the query
- One single scan on the input data
  - Results for the union of the query versions

# 3. VERSION RANGE PATHSTACK

Document fragment




**Version Tree**

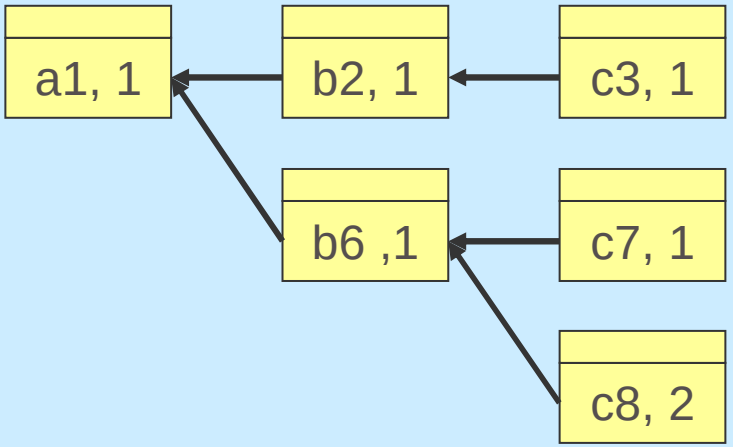
V1 → V2

**Input**

Page 0

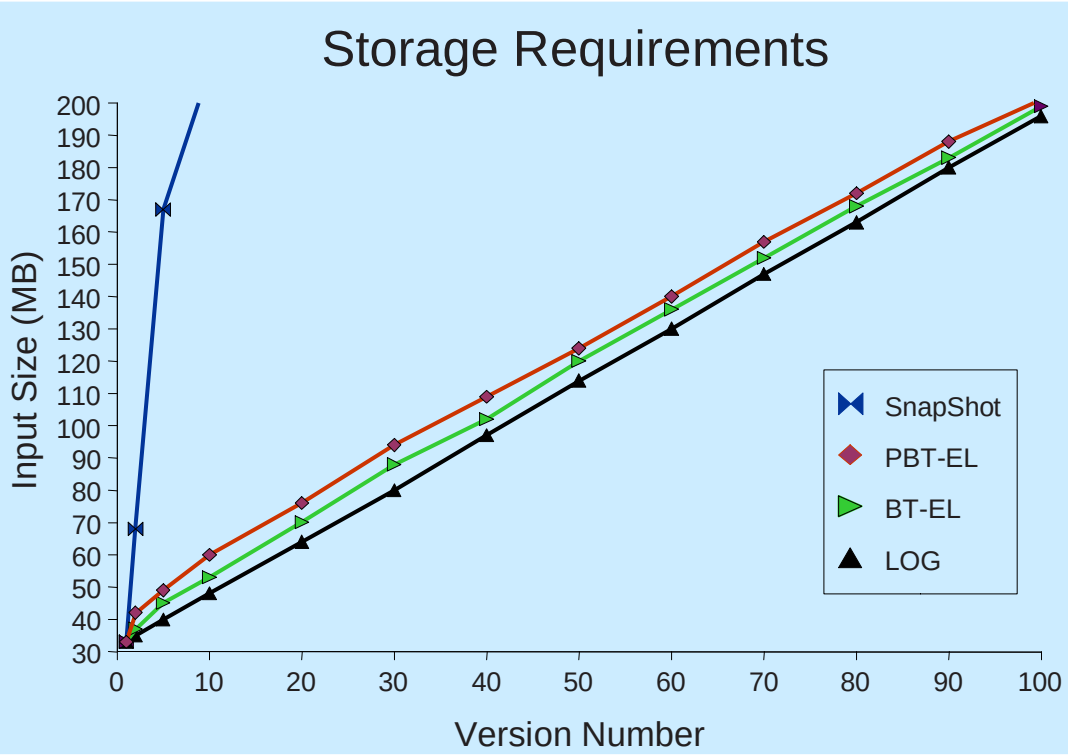
| |
|---|
| a, 1, 10, 1 |
| b, 2, 5, 1 |
| c, 3, 4, 1 |
| b, 6, 11, 1 |
| c, 7, 10, 1 |
| c, 8, 9, 2 |

Stacks



# Experiments Evaluation

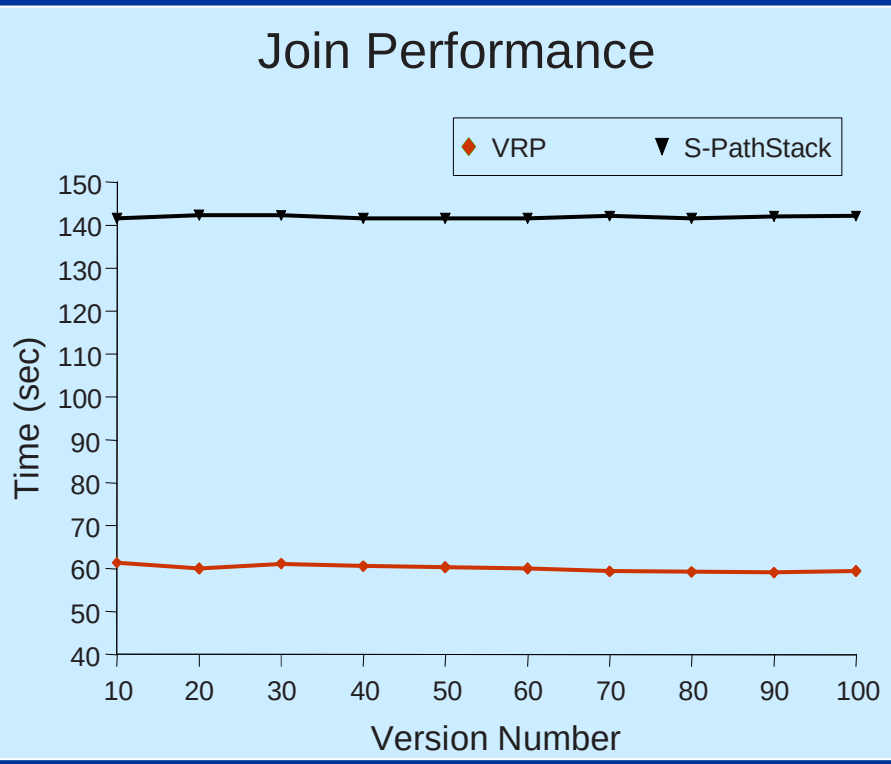
Storage requirements for BT_Element List list compared to the log and snapshot approaches




- LOG: lower extreme, but impractical for query processing
- BT-EL and PBT-EL similar to LOG but:
  - BT-EL: more efficient
  - PBT-EL: BT-EL without optimization
- SnapShot: prohibitively large space

# Experiments Evaluation

Scalability of Join Performance: *VRP* performance when varying number of versions in the document




- Original snapshot PathStack: run each of the 3 versions one at a time
- VRP: lowest performance, independent from the document size

# Conclusions

- **Context:** Collaborative applications $\Rightarrow$ Branch versions of XML documents
- **Problem:** Answer path queries over branched versions of an XML archive + queries over many versions at a time
- **Solution:**
  - Storage technique for versioned archive with small replication
  - VRP algorithm for path expression over multiple versions
- **Experiments:** Efficient and scalable performance

# Questions?