# TASM: Top-$k$ Approximate Subtree Matching


Nikolaus Augsten[1] Denilson Barbosa[2]
Michael Böhlen[3] Themis Palpanas[4]


[1]Free University of Bozen-Bolzano, Italy
augsten@inf.unibz.it

[2]University of Alberta, Canada
denilson@cs.ualberta.ca

[3]University of Zurich, Switzerland
boehlen@ifi.uzh.ch

[4]University of Trento, Italy
themis@disi.unitn.eu


UNIVERSITY OF ALBERTA



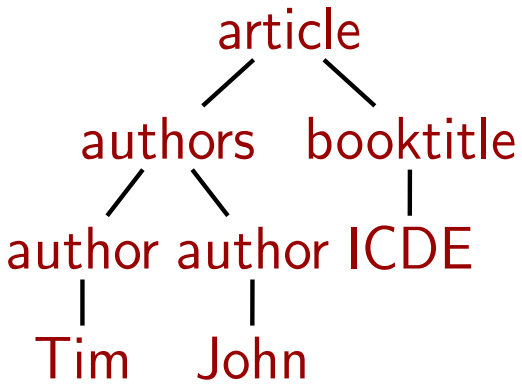

ICDE 2010, March 3
Long Beach, CA, USA

# Outline

# Outline

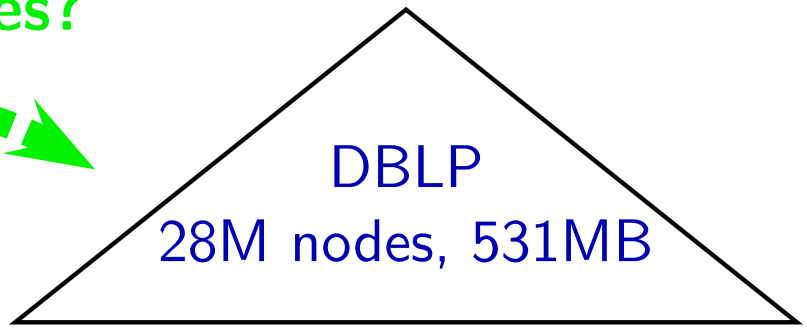
# Motivation

**Query** (XML fragment)



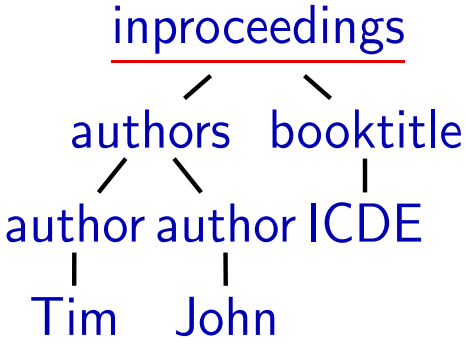

**top-$k$ matches?**

**Document** (very large XML)

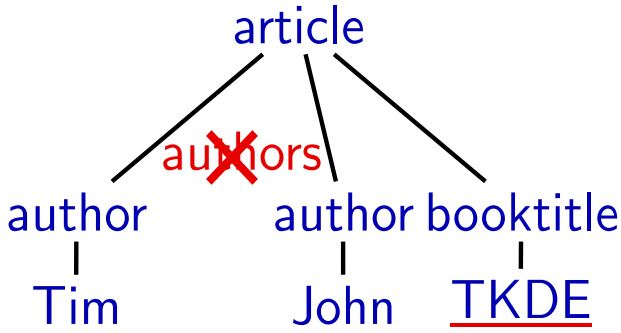



Rank the **top-$k$ matches** for the **article query** in the **DBLP document**!

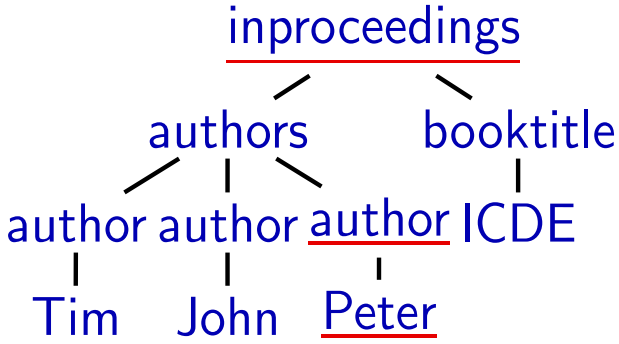
Example Answer: $k = 3$




(1 error)




(2 errors)




(3 errors)

# TASM: Top-$k$ Approximate Subtree Matching

## Definition (TASM: Top-$k$ Approximate Subtree Matching)

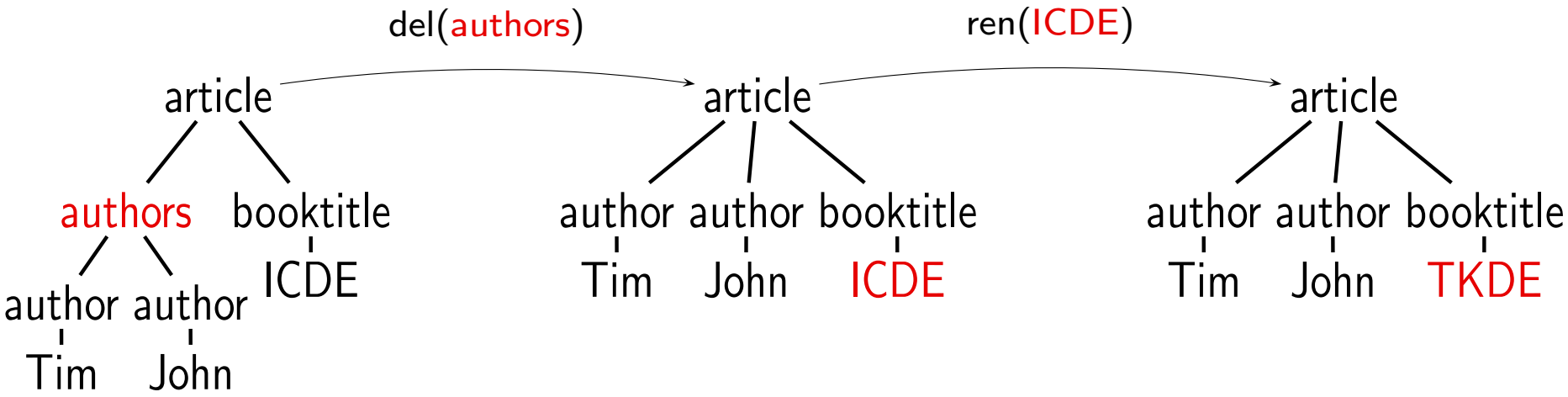
**Given:** query tree $Q$, document tree $T$, size $k$ of ranking

**Goal:** Compute a

- top-**k** ranking $R = (T_1, T_2, \ldots, T_k)$
- of all subtrees $T_i$ of document $T$
- with respect to query $Q$
- using the tree edit distance for the ranking.

- Subtree $T_i$:
  - a node and all its descendants
  - largest subtree is document itself
- top-**k** ranking $R = (T_1, T_i, \ldots, T_k)$
  - subtrees **sorted** by distance to query
  - **best k** subtrees: $T_i \notin R \Rightarrow ted(Q, T_k) \leq ted(Q, T_i)$

# Ranking Function: Tree Edit Distance (TED)




- **Tree Edit Distance**: Minimum number of node edit operations (insert, rename, delete) that transform one tree into the other.
- **TASM** computes TED between query and document subtrees
- **Size and number** of computed subtrees define TASM complexity

# State of the Art

- **TASM-Dynamic**: dynamic programming solution[1]
  - computes distance to **every subtree** of the document
  - **use smaller subtrees** to compute larger ones
  - rank subtrees by visiting **memoization table**
  - **Space complexity**: $O(mn)$, $m$: query size, $n$: document size
- **Space complexity limits** application to databases
  - in database applications $n$ is huge (database size!)
  - TASM-Dynamic maintains two $m \times n$ **matrixes in RAM**
  - $>$ **6GB RAM** for our tiny query ($m = 8$) on DBLP ($n = 28 \times 10^6$)
- For **database** size solutions **dynamic programming** is **too expensive**.

State-of-the-art algorithms do not scale!

[1] Zhang and Shasha 1989, Demaine et al. 2007

# Problem Definition

Find a solution for **TASM** (Top-k Approximate Subtree Matching) that

- scales to **very large documents**
- runs in **small memory**
- ranks subtrees **correctly** (no heuristics!)

# Outline

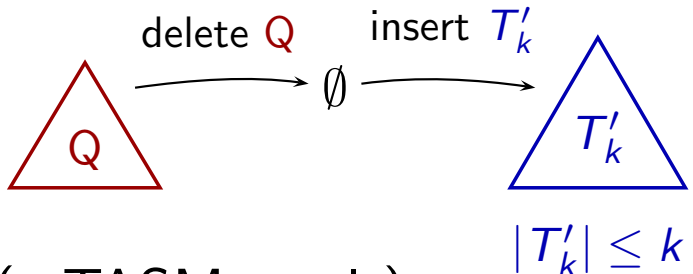
# Outline

# Subtree Size Upper Bound in Three Steps

1. **Rank first $k$ subtrees** of $T$ in postorder: $R' = (T'_1, T'_2, \ldots, T'_k)$

   worst match

   (i) $ted(Q, T'_k) \leq |Q| + |T'_k|$



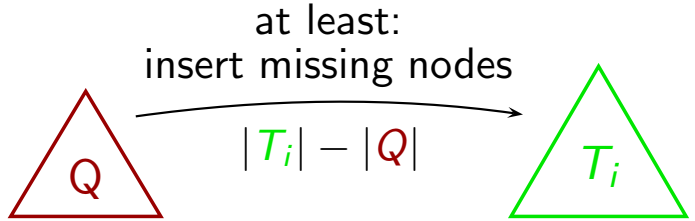

2. **Final ranking** $R = (T_1, T_2, \ldots, T_k)$ (=TASM result)

   $T_i$'s in $R$ are better than worst match $T'_k$ of $R'$

   (ii) $ted(Q, T_i) \leq ted(Q, T'_k) \leq |Q| + |T'_k|$

3. **Size upper bound** for subtree $T_i$

   $|T_i| - |Q| \leq ted(Q, T_i)$




$$|T_i| \leq ted(Q, T_i) + |Q| \leq 2|Q| + |T'_k| \leq 2|Q| + k$$

# Upper Bound on Subtree Size

**Theorem (Upper Bound on Subtree Size)**

*TASM needs to consider only small document subtrees of size $\tau$ or less:*

$$\tau = 2|Q| + k$$

Upper bound is very powerful:

- **independent of document** size and structure!
- **linear in query** size and **k**

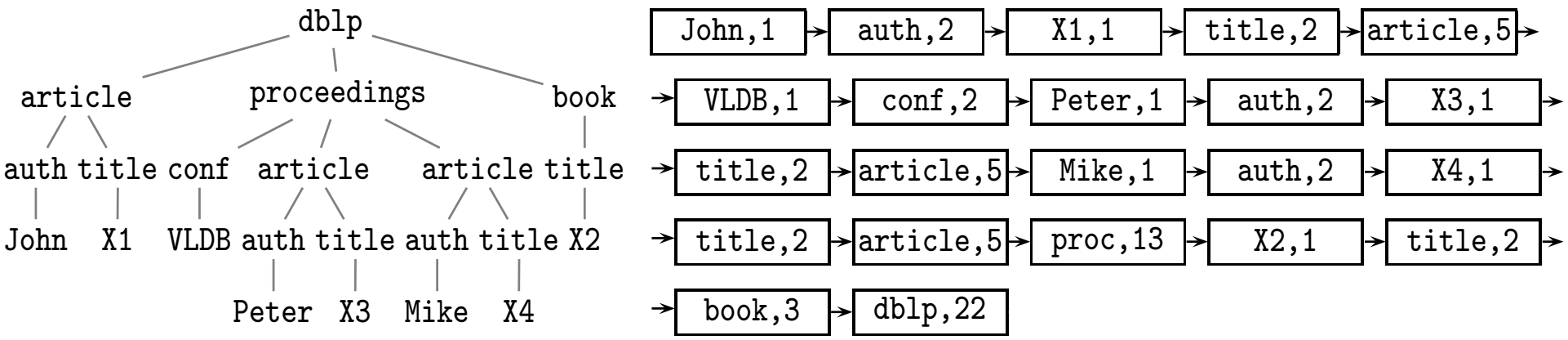
**Example**: top-10 with example query $|Q| = 8$ on DBLP (28M nodes)

- **with** bound: max subtree size $\tau = 2 * 8 + 10 = \mathbf{26}$
- **without** bound: maximum subtree size is **28M** (whole document)!

Document-independent upper bound on subtree size!

# Outline

# Document Format: Postorder Queue

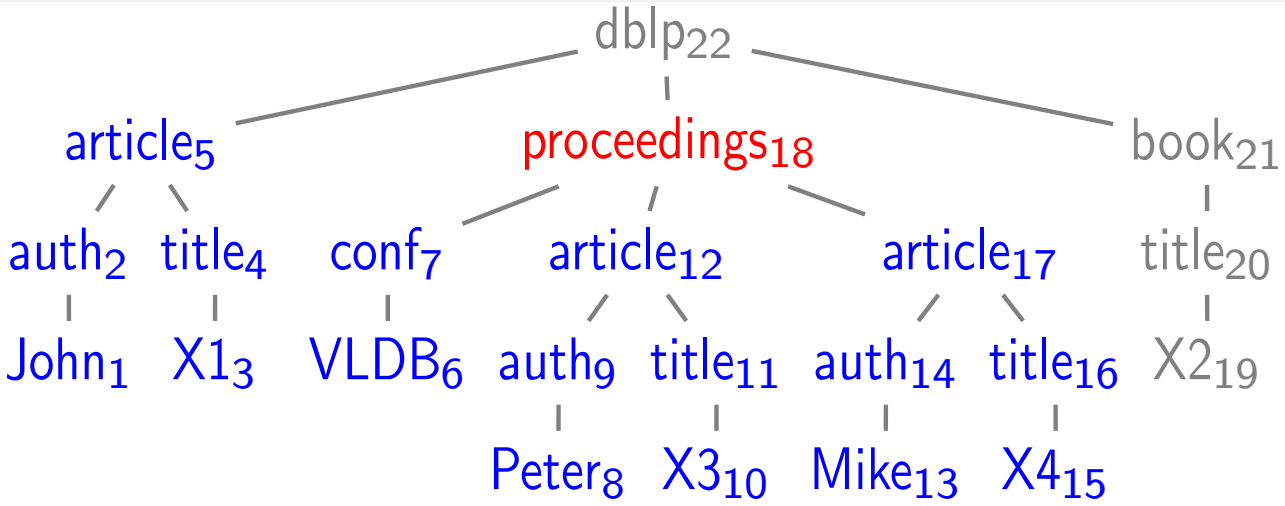



- **Postorder queue**: queue of (label,size)-pairs
  - **dequeue** removes leftmost element, e.g., $(\texttt{John}, 1)$
  - no random access!
- **Relevant and state-of-the-art for XML Parsing**
  - full subtree known only at closing tag
  - closing tags appear in postorder
- **Implementation** is efficient and heavily used for
  - XML streams
  - plain XML files (e.g., SAX)
  - XML in database (Dewey, interval encoding, ...)

# Candidate Subtrees

- **Candidate subtrees** are all subtrees $T_i$ of the document with
  - $|T_i| \leq \tau$ AND
  - $T_i$ is **not contained** in a larger subtree $|T_j| \leq \tau$
- **Pruning**: find candidate subtrees

# Simple Pruning Approach



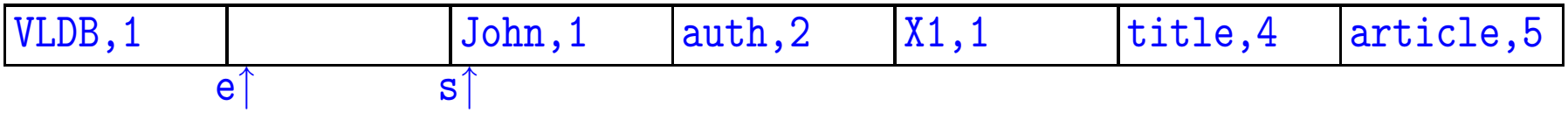

- **Simple pruning** approach: ($\tau = 6$ in example above)
  - add nodes to memory buffer until non-candidate ($|T_i| > \tau$) is added
  - subtrees of non-candidate with $|T_i| \leq \tau$ are candidate subtrees
- **Problem**: **memory buffer** can grow very **large**!
  - must keep subtrees in memory until non-candidate ancestor is read
  - worst case: memory buffer stores $O(n)$ nodes (frequent in data-centric XML!)
- **Example**: DBLP, $\tau = 50$
  - 99% of nodes are still in buffer when root node is read!
- **Simple pruning not feasible** for large documents!

# Efficient Pruning is Tricky!

- **Problem:** when can we remove a node from the buffer?
  - when we see $|T_i| \leq \tau$, we don't yet know about parent (postorder!)
  - subtree of parent might be smaller than $\tau$!
- **Our Solution** does not wait for parent
  - **prefix ring buffer**: fixed size buffer
  - **pruning rule:** prune based on following nodes

# Pruning in Small Memory

prefix ring buffer ($\tau = 6$)




**Prefix ring buffer** of size $\tau + 1$ (main memory)

- **stores prefix** ($\tau$ nodes in postorder) of the document
- **two operations**
  - append new node
  - remove leftmost subtree/node

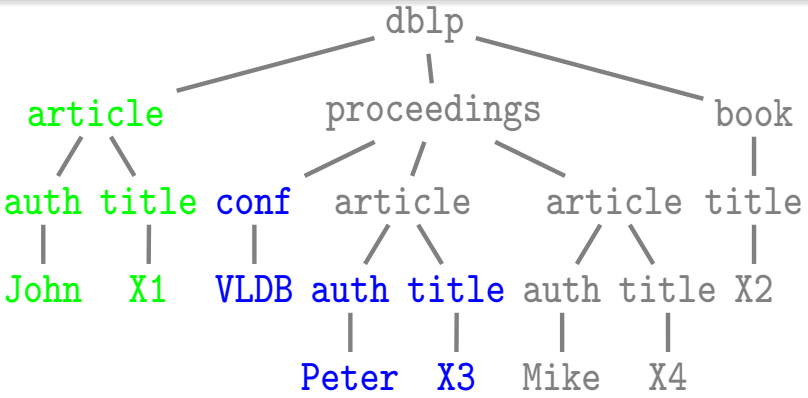
**Pruning rule:** If **leftmost node** in full ring buffer is

- **leaf**: leftmost subtree is candidate subtree
- **non-leaf**: leftmost node is non-candidate node

# Pruning Rule – Intuition

- **Candidate subtree**: leftmost node is a leaf
  - $T_i$: leftmost subtree, starts with leftmost node
  - $T_j$: smallest subtree that contains $T_i$
  - due to postorder: $T_j$ contains all nodes in buffer
  - since $|T_i| \leq \tau$ and $|T_j| > \tau$: $T_i$ is a candidate
- **Non-candidate node**: leftmost node is a non-leaf
  - leftmost non-leaf is parent of previously removed nodes
  - we remove either candidate subtrees and non-candidate nodes
  - in both cases: parent is a non-candidate

# Prefix Ring Buffer Pruning – Example



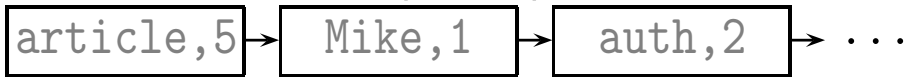

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

---

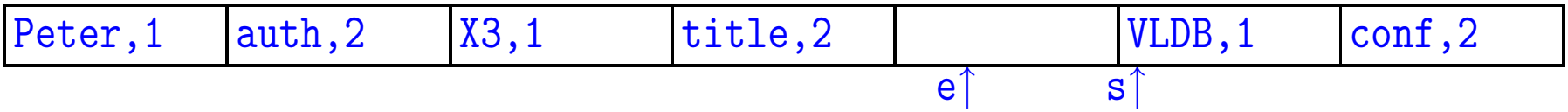
$\tau = 6$

append

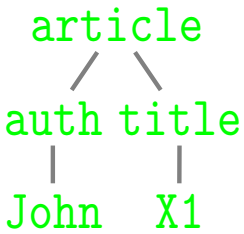
postorder queue (input)




prefix ring buffer (main memory)




candidate subtrees: (output)



# TASM-Postorder

**TASM-postorder**

1. empty ranking $R$, tightening upper bound $\tau' = \tau$
2. for each candidate subtree $T_i$
   a. if $|R| = k$: update $\tau' = \min(\tau, \max(R) + |Q|)$
   b. compute tree edit distance for all subtrees of $T_i$ within $\tau'$
   c. update ranking $R$

## Theorem (TASM-Postorder)

*The* **space** *complexity of TASM-postorder is* **independent of the document size**:

$$O(m^2 + mk)$$

*($m$: query size, $k$: result size)*

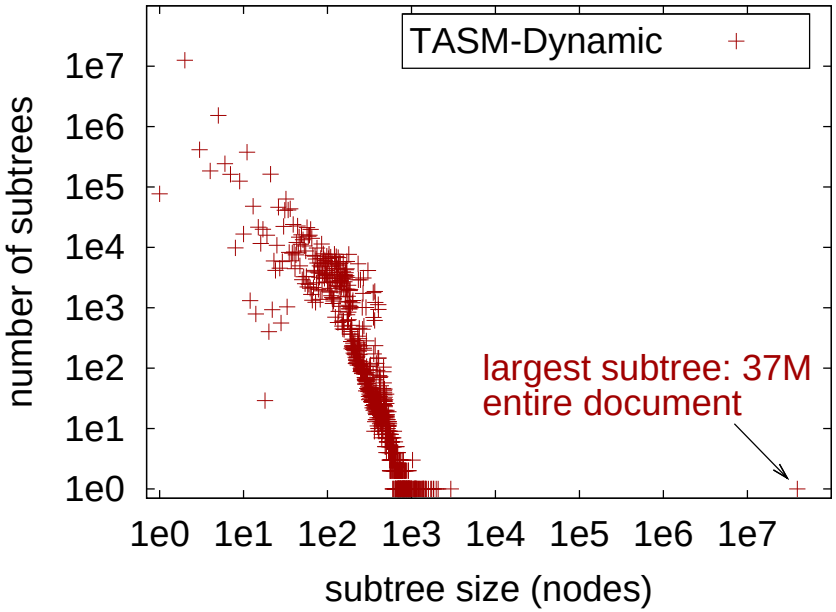
TASM-postorder scales to very large documents!

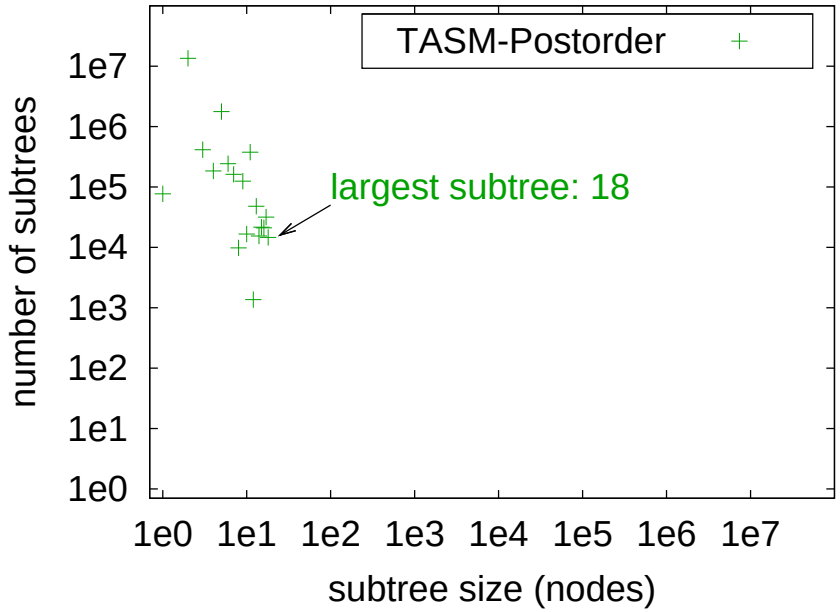
# Outline

# Pruning Effectiveness

**Prefix ring buffer pruning is very effective!**
Maximum subtree reduced from 37M to 18 nodes.

- **Dataset: PSD** protein sequences, 37M nodes, 683MB
- Compute TASM ($|Q| = 4$, $k = 1$)
  - TASM-dynamic (state of the art)
  - TASM-postorder (our solution)
- **Histogram** of computed subtrees

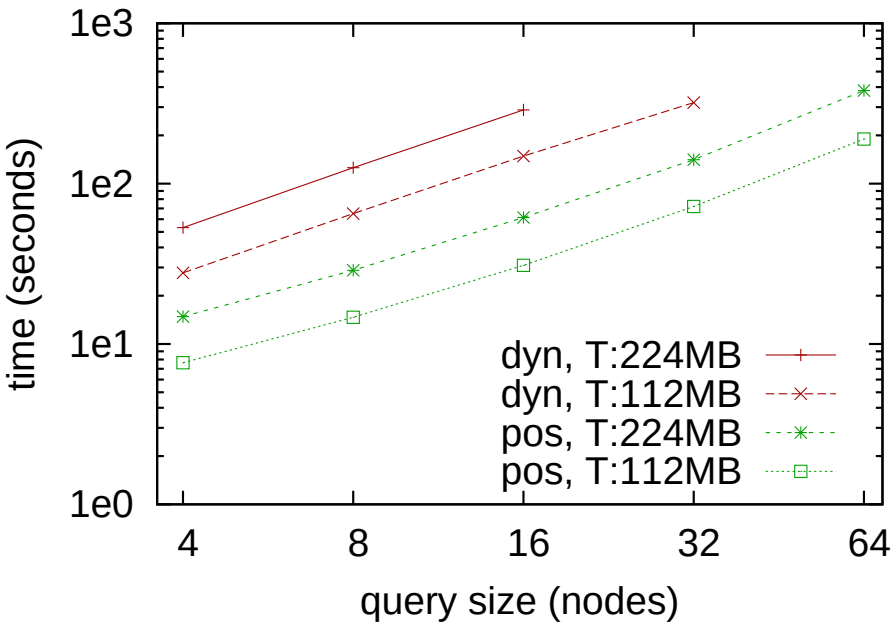


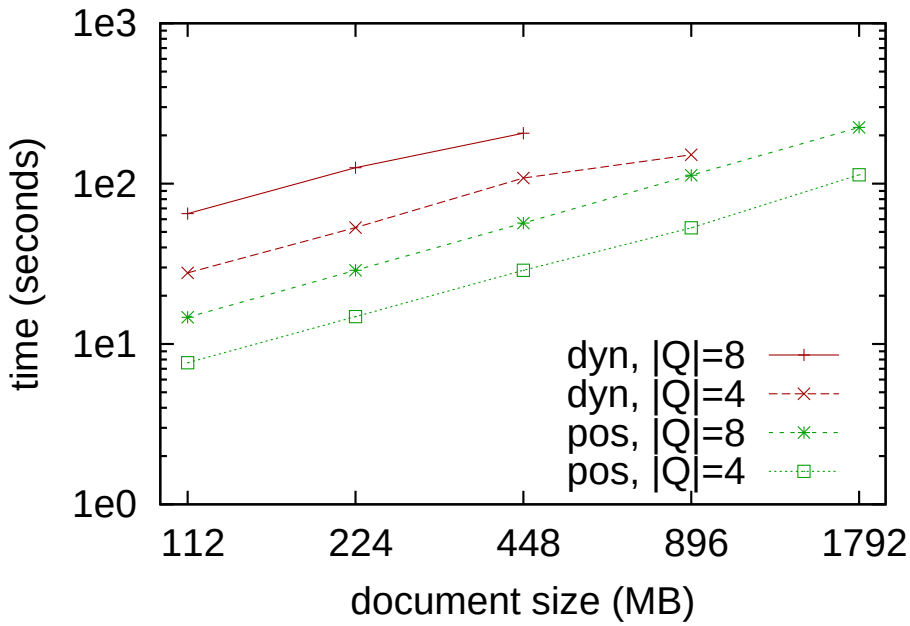



# Scalability: TASM-Postorder vs. TASM-Dynamic

TASM-postorder much faster than TASM-dynamic.

- **Dataset: XMark** (synthetic XML for benchmark)
- Vary **query size** and **document size**
- Compute TASM ($k = 5$)
  - TASM-dynamic (state of the art)
  - TASM-postorder (our solution)
- Measure wall clock **time**

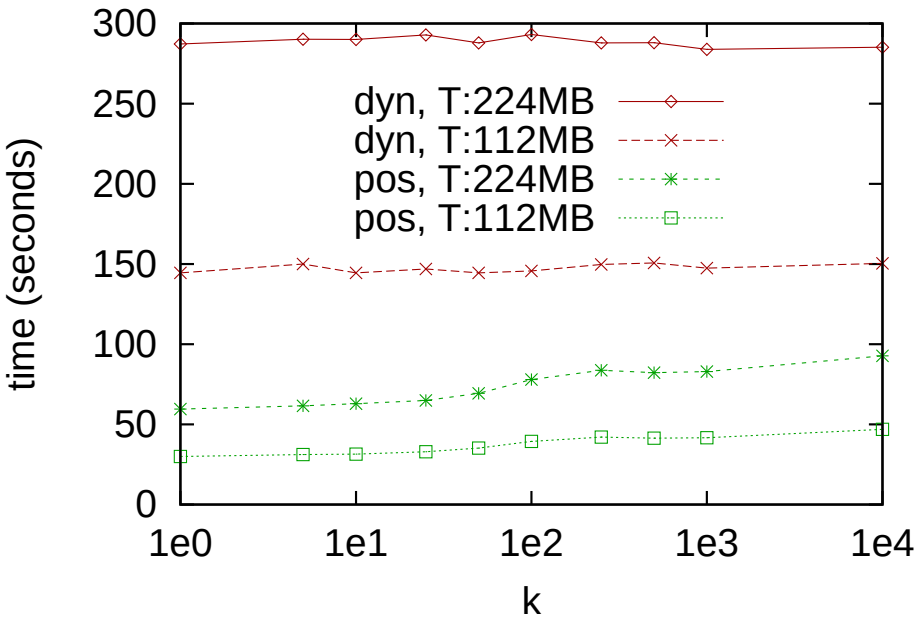


# Scalability with Result Size $k$

**TASM-postorder scales well with $k$.**

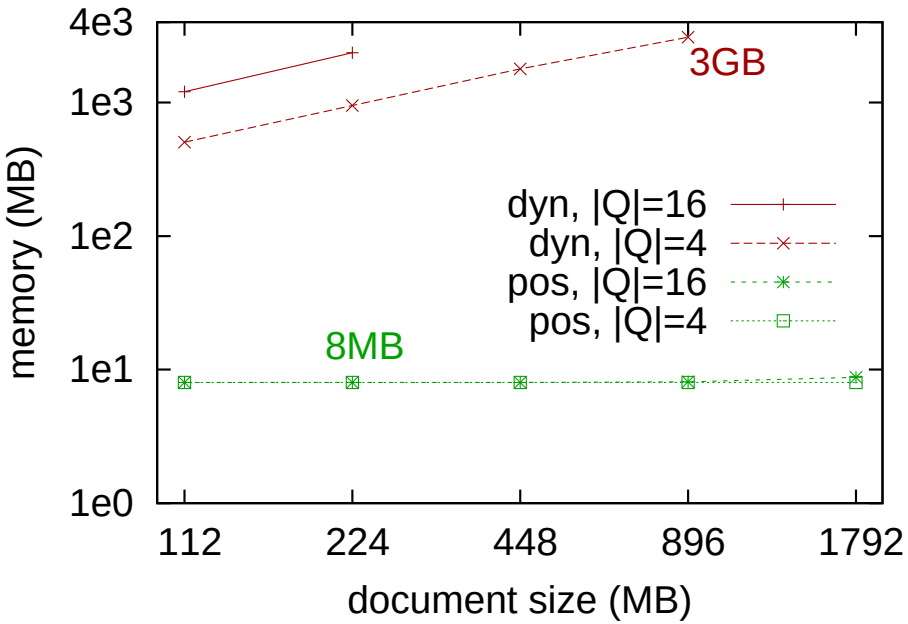
Increasing $k$ by 4 orders of magnitude only doubles runtime.




- **Dataset: XMark** (synthetic XML for benchmark)
- Vary **k (size of ranking)**
- Compute TASM ($|Q| = 16$)
  - TASM-dynamic (state of the art)
  - TASM-postorder (our solution)
- Measure wall clock **time**

# Space complexity: TASM-Postorder vs. TASM-Dynamic

TASM-postorder: space independent of document!




- **Dataset: XMark** (synthetic XML for benchmark)
- Vary **document size**
- Compute TASM ($k = 5$)
  - TASM-dynamic (state of the art)
  - TASM-postorder (our solution)
- Measure **main memory usage**

# Outline

# Conclusion

## Conclusion

- **Prefix Ring Buffer** for space efficient pruning
- **Dynamic programming does not scale** for database size solutions.
- **Upper bound $\tau$**: limit maximum subtree size for TASM
- **TASM-postorder:** highly scalable TASM algorithm

TASM-postorder makes TASM feasible.

**Future Work** – New research opportunities:

- tune tree edit distance to different applications
- index the document: can we avoid a document scan?
- parallel TASM algorithm: where to split document?


Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann.
An optimal decomposition algorithm for tree edit distance.
In *ICALP*, volume 4596 of *LNCS*, pages 146–157, Wroclaw, Poland, July 2007. Springer.


K. Zhang and D. Shasha.
Simple fast algorithms for the editing distance between trees and related problems.
*SIAM J. on Computing*, 18(6):1245–1262, 1989.