XML Advance Querying

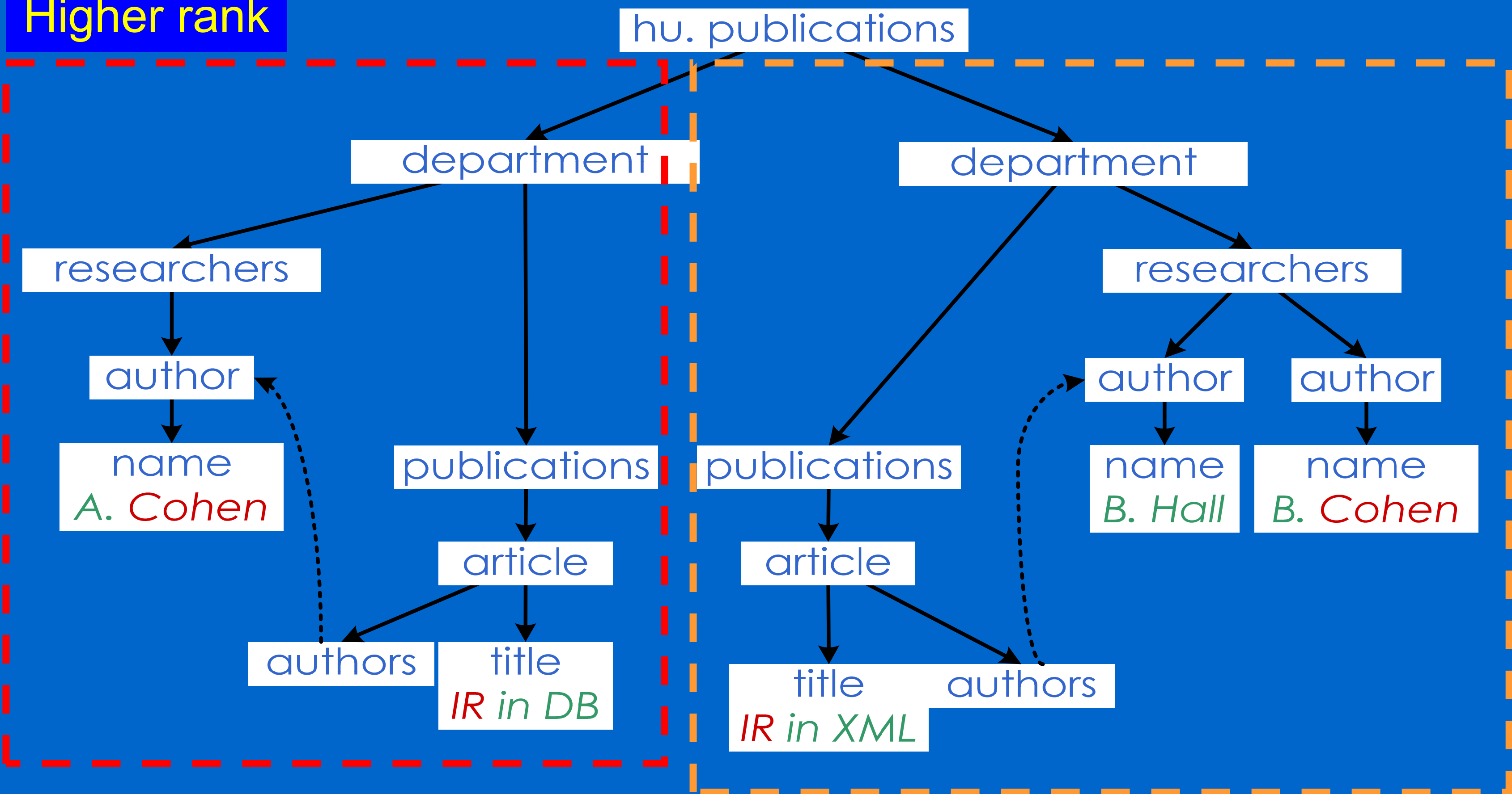
Introduction

- Queries may not always be precise and can return a large number of results, especially in large document collections.
- Rank the query results so that the most relevant results appear first
- XML Scoring and Ranking
  - Score elements with respect to their relevance to a query
  - Determine the appropriate level of component granularity to return to users

XML scoring and ranking:
score elements wrt. their relevance to a query

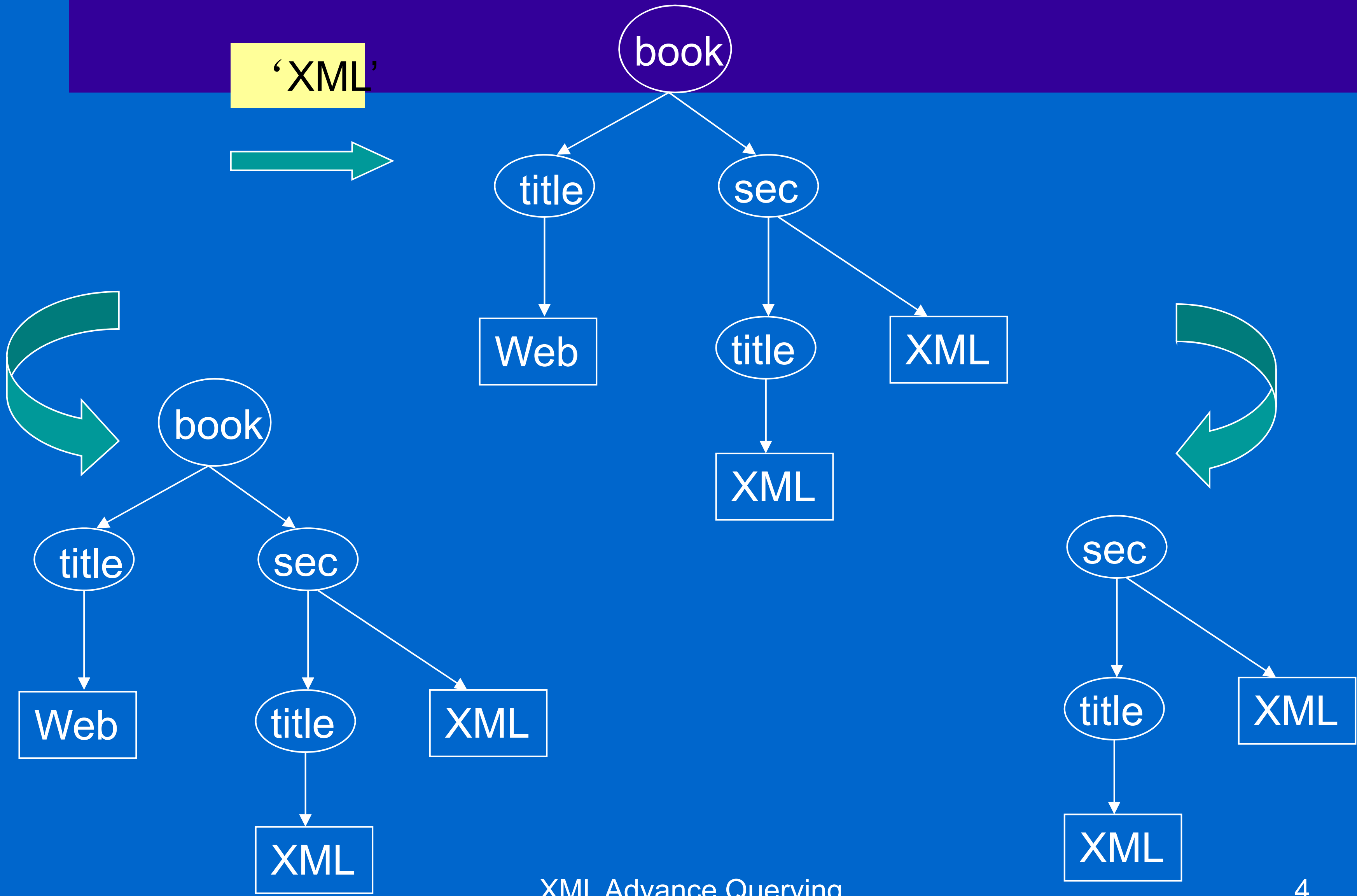
{Cohen , IR}

Higher rank



XML scoring and ranking:

Determine the appropriate level of component granularity to return to users

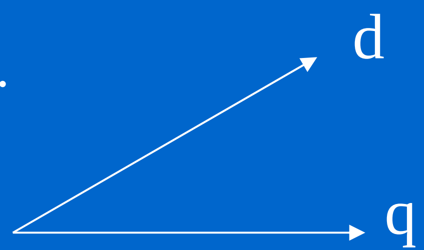


Scoring, Ranking and Querying

- Common method
  - Traditional weighting terms: TF-IDF values
- Challenges
  - Term and element statistics
    - Scoring at element level requires statistics at element level
    - XML elements are nested
  - Capture the structure of XML document
- Discussion
  - Vector-based scoring
  - XRank
  - XML Query Relaxation

Vector Space Model in Text Retrieval Task

- Given a query
  - According to the retrieval formula, compute the relevance score for each document;
  - Rank the documents according to relevance score.
- Vector Space Model
  - Represent doc/query by a vector of terms
  - Relevance between doc and query →distance between two vectors
  - Weighting terms (TF-IDF values)



$$\rho(q,d) = \frac{\sum_{t \in q \cap d} w_q(t) \times w_d(t)}{|q| \times |d|}$$

Searching XML Documents via XML Fragments

- SIGIR 2003
- Utilize vector space models
  - Both documents and queries are expressed in free text.
  - Compare unstructured data to unstructured data
- Document collection:
  - XML documents
    - Each document is a hierarchical structure of nested elements
    - Markup in the document mainly serves for exposing the logical structure of a document.
- Query
  - content + explicit references to the XML structure
  - specifies the target element need to be returned

Extending the Vector Space Model

- Indexing unit: (t_i, c_i)
  - E.g. (“Harry Potter ”, /book/title)
  - Can be matched with
    - (“Harry Potter ”,/book)
    - (“Harry Potter ”,/book/sec/title)
- Retrieval Formula

Context resemblance measure

$$\rho(q,d) = \frac{\sum_{(t,c_i)\in q} \sum_{(t,c_k)\in q} w_q(t,c_i) \times w_d(t,c_k) \times cr(c_i,c_k)}{|q| \times |d|}$$

Perfect match: $cr(c_i,c_k) = 1$,when $c_i = c_k$; 0 ,otherwise.

Partial match: $cr(c_i,c_k) = \frac{1+|c_i|}{1+|c_k|}$,when c_i subsequence of c_k; 0, otherwise

Fuzzy match: $cr(c_i,c_k) = StrSimilarity(c_i,c_k)$

Flat (ignore context): $cr(c_i,c_k) = 1, \forall c_i, c_k$

Extending the Vector Space

$$\rho(q,d) = \frac{\sum_{(t,c_i)\in q} \sum_{(t,c_k)\in q} w_q(t,c_i) \times \boxed{w_d(t,c_k) \times cr(c_i,c_k)}}{|q| \times |d|}$$

$$w_d(t,c_k) = tf_d(t,c_k) \times idf(t,c_k)$$,where $$idf(t,c) = \log\left(\frac{|N|}{|N_{(t,c)}|}\right)$$

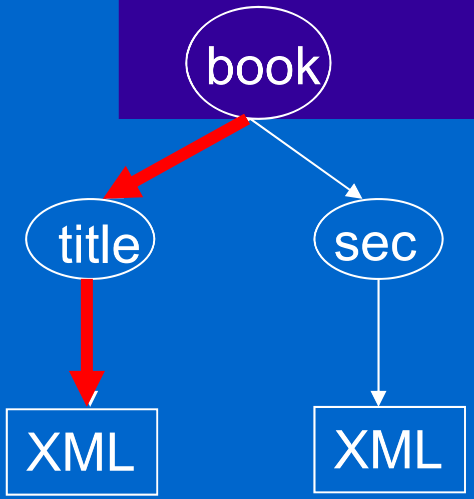
If c is rare, idf(t,c) would be high in spite of t being very common.

"Merge-idf" variant:

$$w_d(t,c_k) = tf_d(t,c_k) \times idf(t,C)$$,where $C = \bigcup_k c_k$ and $cr(c_i,c_k) > 0$

"Merge" variant:

$$tf_d(t,C) \times idf(t,C) \times cr(c_i,C)$$



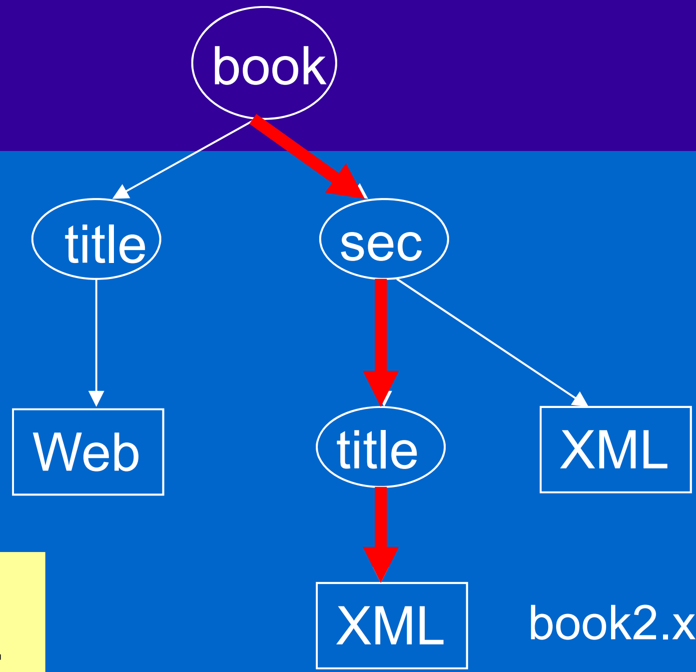
book1.xml

C_1 =/book/title

Query:
<book>
<title> XML </title>
</book>

C=/book/title

cr_1=1
cr_2=(1+2)/(1+3)=0.75



book2.xml

C_2=/book/sec/title

$$\rho(q,d)=\frac{\sum_{(t,c_i)\in q}\sum_{(t,c_k)\in d} w_q(t,c_i)\times w_d(t,c_k)\times cr(c_i,c_k)}{|q|\times|d|}$$

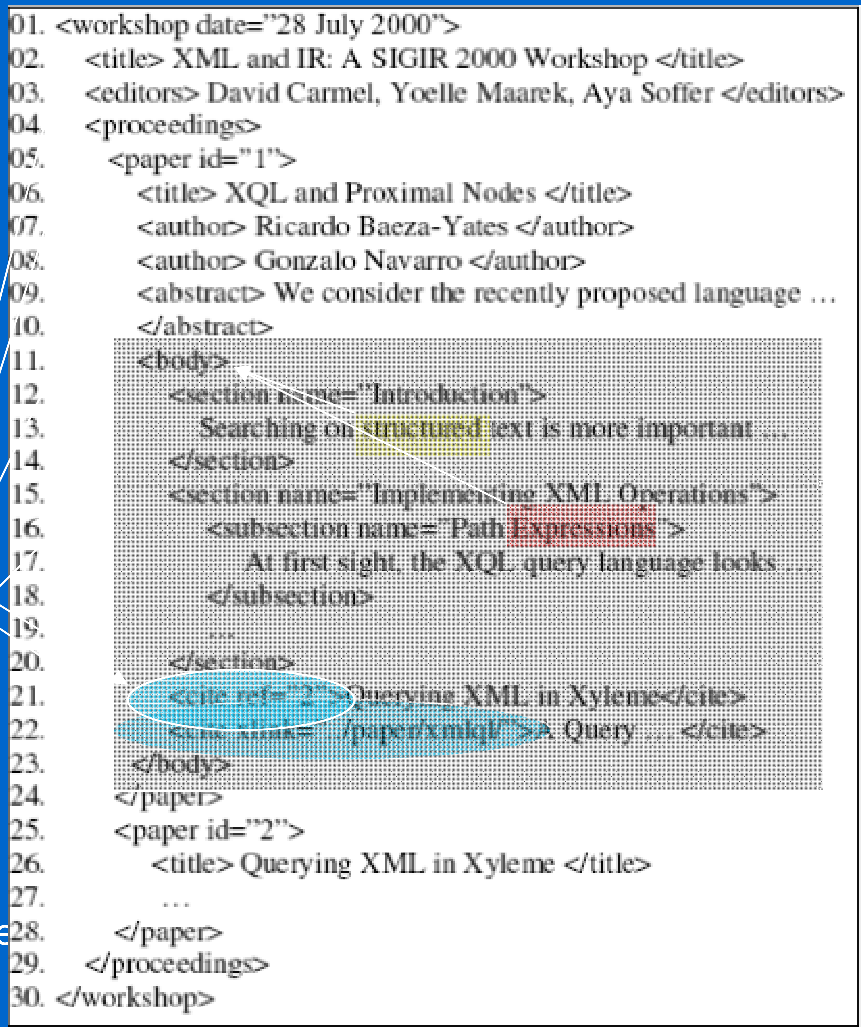
$$cr(c_i,c_k)=\frac{1+|c_i|}{1+|c_k|}$$

XRANK: Ranked Keyword Search over XML Documents

- *ElemRank* is similar to Google's *PageRank* but is computed at the granularity of an element and takes the nested structure of XML into account
- Algorithms
  - Naïve
  - Dewey Inverted List (DIL)
  - Ranked Dewey Inverted List (RDIL)
  - Hybrid Dewey Inverted List (HDIL)

Problem Overview

- Similar to Google
  - Text search
  - Hyperlink
    - May rank high
  - Keyword proximity
  - Ranking
- Different from Google
  - Hierarchical structure
  - Element as result



```
<workshop date="28 July 2000">
  <title> XML and IR: A SIGIR 2000 Workshop </title>
  <editors> David Carmel, Yoelle Maarek, Aya Soffer </editors>
  <proceedings>
    <paper id="1">
      <title> XQL and Proximal Nodes </title>
      <author> Ricardo Baeza-Yates </author>
      <author> Gonzalo Navarro </author>
      <abstract> We consider the recently proposed language ...
      </abstract>
      <body>
        <section name="Introduction">
          Searching on structured text is more important ...
        </section>
        <section name="Implementing XML Operations">
          <subsection name="Path Expressions">
            At first sight, the XQL query language looks ...
          </subsection>
          ...
        </section>
        <cite ref="2">Querying XML in Xyleme</cite>
        <cite xlink="../paper/xmlql/">A Query ... </cite>
      </body>
    </paper>
    <paper id="2">
      <title> Querying XML in Xyleme </title>
      ...
    </paper>
  </proceedings>
</workshop>
```

Definitions

- Set of nodes: $N = NE \bigcup NV$
  - *NE* is the set of elements
  - *NV* is the set of values (including element tags and attribute names)
- *CE* is the set of containment edges relating nodes:

$(u,v) \in CE$ iff *v* is a value/nested sub-element of *u* ➔ *u* is the parent of node *v*.

```
<workshop date="28 July 2000">
  <title> XML and IR: A SIGIR 2000 Workshop </title>
  <editors> David Carmel, Yoelle Maarek, Aya Soffer </editors>
  <proceedings>
    <paper id="1">
      <title> XQL and Proximal Nodes </title>
      <author> Ricardo Baeza-Yates </author>
      <author> Gonzalo Navarro </author>
      <abstract> We consider the recently proposed language ...
      </abstract>
      <body>
        <section name="Introduction">
          Searching on structured text is more important ...
        </section>
        <section name="Implementing XML Operations">
          <subsection name="Path Expressions">
            At first sight, the XQL query language looks ...
          </subsection>
          ...
        </section>
        <cite ref="2">Querying XML in Xyleme</cite>
        <cite xlink="../paper/xmlql/">A Query ... </cite>
      </body>
    </paper>
    <paper id="2">
      <title> Querying XML in Xyleme </title>
      ...
    </paper>
  </proceedings>
</workshop>
```

Definitions

- A node u is an ancestor of node v if there is a sequence of containment edges that lead from u to v.
- The predicate $contains^{*}(v,k)$ is true if the node v directly of indirectly contains the keyword k.
- HE is the set of hyperlink edges related nodes: $(u,v) \in HE$ iff u contains a hyperlink reference to v.

Keyword Query Results

- Query of *n* keywords: $Q = \{k_1, \ldots, k_n\}$
- Let R_o be the set of elements that directly or indirectly contain all the query keywords:

$$R_o = \{v \mid v \in NE \wedge \forall k \in Q(contains*(v,k))\}$$

- The result of query *Q*:

$$result(Q) = \{v \mid \forall k \in Q, \exists c \in N((v,c) \in CE \wedge c \notin R_o \wedge contains*(c,k))\}$$

  – return the sub-elements if possible for a more specific result
  – *result(Q)* is a set of solution, but it does not cover all solutions!!!

Ranking Keyword Query Results

- Ranking function depends on:
  - Result specificity
  - Keyword proximity
  - Hyperlink awareness

→ ElemRank ~ PageRank of Google

Ranking

- Search query: $Q = \{k_1, \ldots, k_n\}$
- Assume v_1 is an element solution
  → for every keyword k_i, there is a sequence of containment edges in *CE* $(v_1, v_2), (v_2, v_3), \ldots, (v_t, v_{t+1})$ such that v_{t+1} is a *value node* that directly contains the keyword k_i

$$r(v_1, k_i) = ElemRank(v_t) \times decay^{t-1}$$

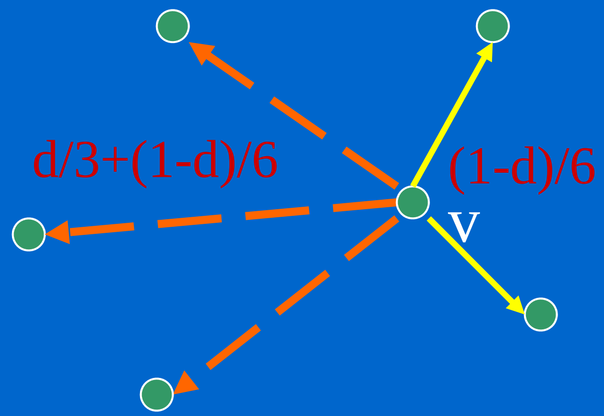
Multiple matches

$$\hat{r}(v_1, k_i) = f(r_1, r_2, \ldots, r_m)$$ f is an aggregate function, say sum

$$R(v_1, Q) = \left(\sum_{1 \le i \le n} \hat{r}(v_1, k_i) \right) \times p(v_1, k_1, k_2, \ldots, k_n)$$

p is a proximity function

PageRank [Brin & Page 1998]



: random jump

d : probability of following hyperlink

: Hyperlink edge

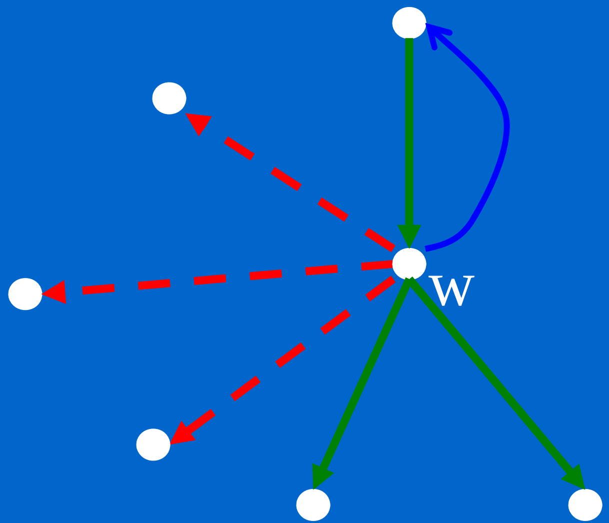
1-d: Probability of random jump

$$p(v) = d \times \sum_{(u,v)\in HE} \frac{p(u)}{N_h(u)} + \frac{1-d}{N_d}$$

$N_h(u)$: the number of out-going hyperlinks from document *u*

N_d: total number of documents

From PageRank to ElemRank



: Hyperlink edge

: **Containment edge (CE)**

: Reverse containment edge

$$p(v) = \frac{1-d}{N_d} + d \times \sum_{(u,v)\in HE} \frac{p(u)}{N_h(u)}$$

$$e(v) = \frac{1-d}{N_e} + d \times \sum_{(u,v)\in E} \frac{e(u)}{(N_h(u) + N_c(u) + 1)}$$

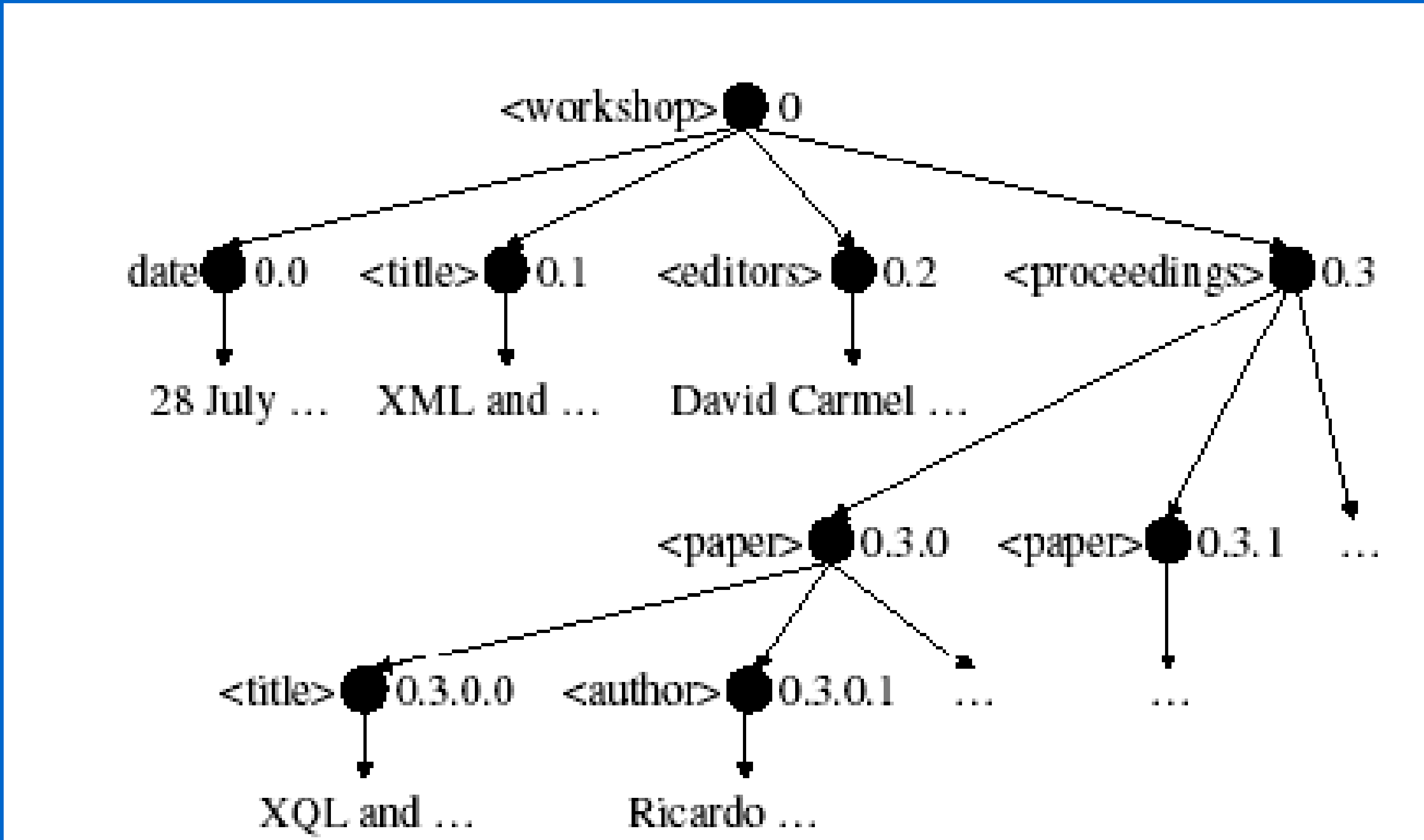
$$e(v) = \frac{1-d_1-d_2}{N_e} + d_1 \times \sum_{(u,v)\in HE} \frac{e(u)}{N_h(u)} + d_2 \times \sum_{(u,v)\in CE\cup CE^{-1}} \frac{e(u)}{N_c(u)+1}$$

$$e(v) = \frac{1-d_1-d_2-d_3}{N_e} + d_1 \times \sum_{(u,v)\in HE} \frac{e(u)}{N_h(u)} + d_2 \times \sum_{(u,v)\in CE} \frac{e(u)}{N_c(u)} + d_3 \sum_{(u,v)\in CE^{-1}} e(u)$$

Naïve algorithm

- Naïve approach
  - Treat each element as a document
  - Build regular inverted list index structures over elements
- Naïve problems
  - Space overhead
  - False query results
  - Inaccurate ranking of results

Dewey Index



<workshop> 0
date 0.0
<title> 0.1
<editors> 0.2
<proceedings> 0.3
28 July …
XML and …
David Carmel …
<paper> 0.3.0
<paper> 0.3.1
…
<title> 0.3.0.0
<author> 0.3.0.1
…
…
XQL and …
Ricardo …

Dewey Inverted List Index

- Generate Dewey elements IDs
- Build index

PL: the position of keyword k inside the element

XQL →

Dewey Id	ElemRank	Position List	
5.0.3.0.0	85	32	
6.0.3.8.3	38	89	91
…	…	…	

Sorted by Dewey Id

Document ID

Unnecessary? Always 0!

Ricardo →

Dewey Id	ElemRank	Position List
5.0.3.0.1	82	38
8.0.1.4.2	99	52
…	…	…

Sorted by Dewey Id

DIL Query Process

- Key idea: merge the query keyword inverted lists, and compute the longest prefix
- Sort all Dewey IDs → start from the smallest

nothing

5.0.3.0.0

XQL

Dewey	Rank[1]	Rank[2]	PosList[1]	PosList[2]	ContainsAll
0	85		32		0
0					0
3					0
0					0
5					0

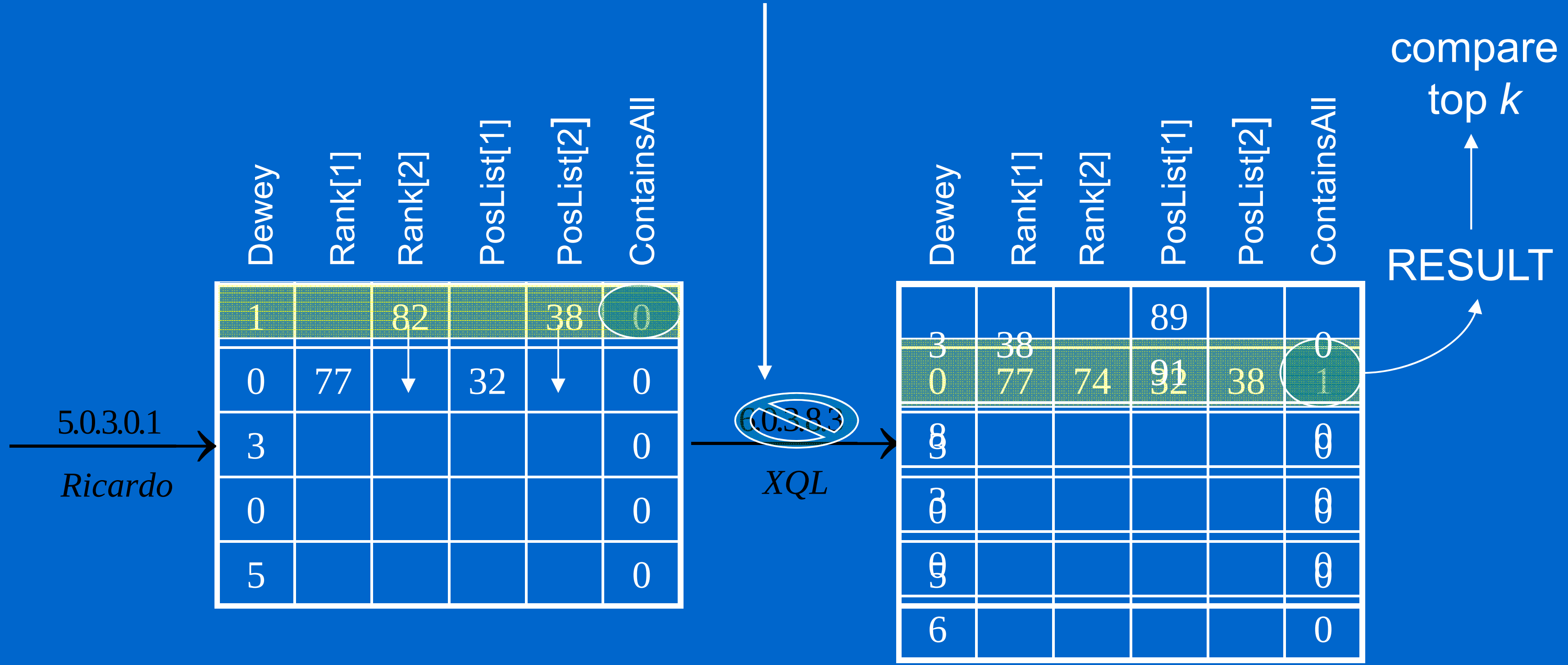
5.0.3.0.1

Ricardo

Dewey	Rank[1]	Rank[2]	PosList[1]	PosList[2]	ContainsAll
0	77		32		0
3					0
0					0
5					0

DIL Query Process

6.0.3.8.3: Prefix not the same, pop up the stack



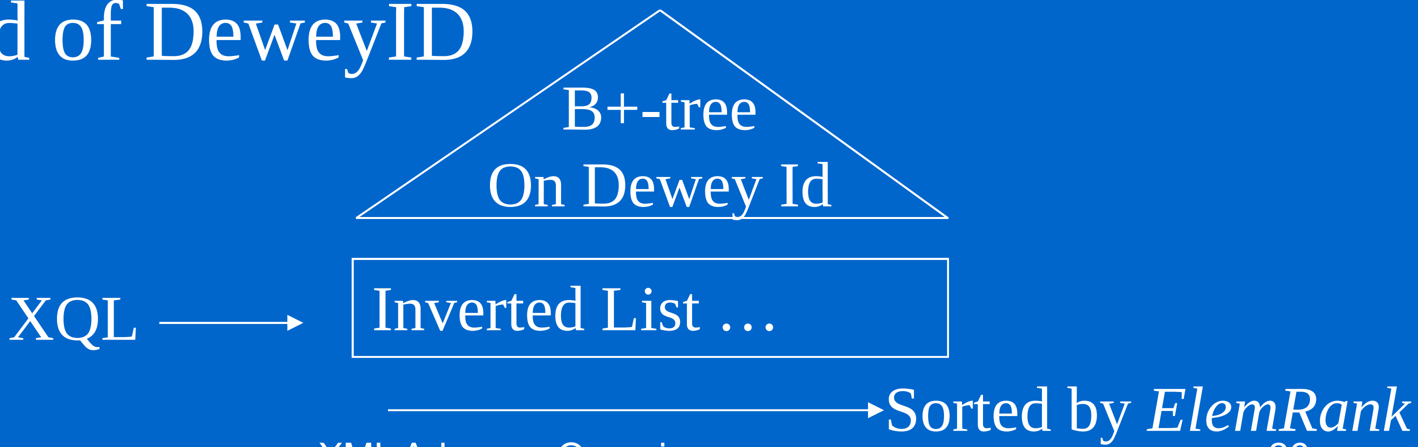
DIL Challenge

- Merging multiple inverted lists
  - Simple equality merge not sufficient
  - Need to infer “most specific” result
- Suppress spurious results
- Two-dimensional proximity

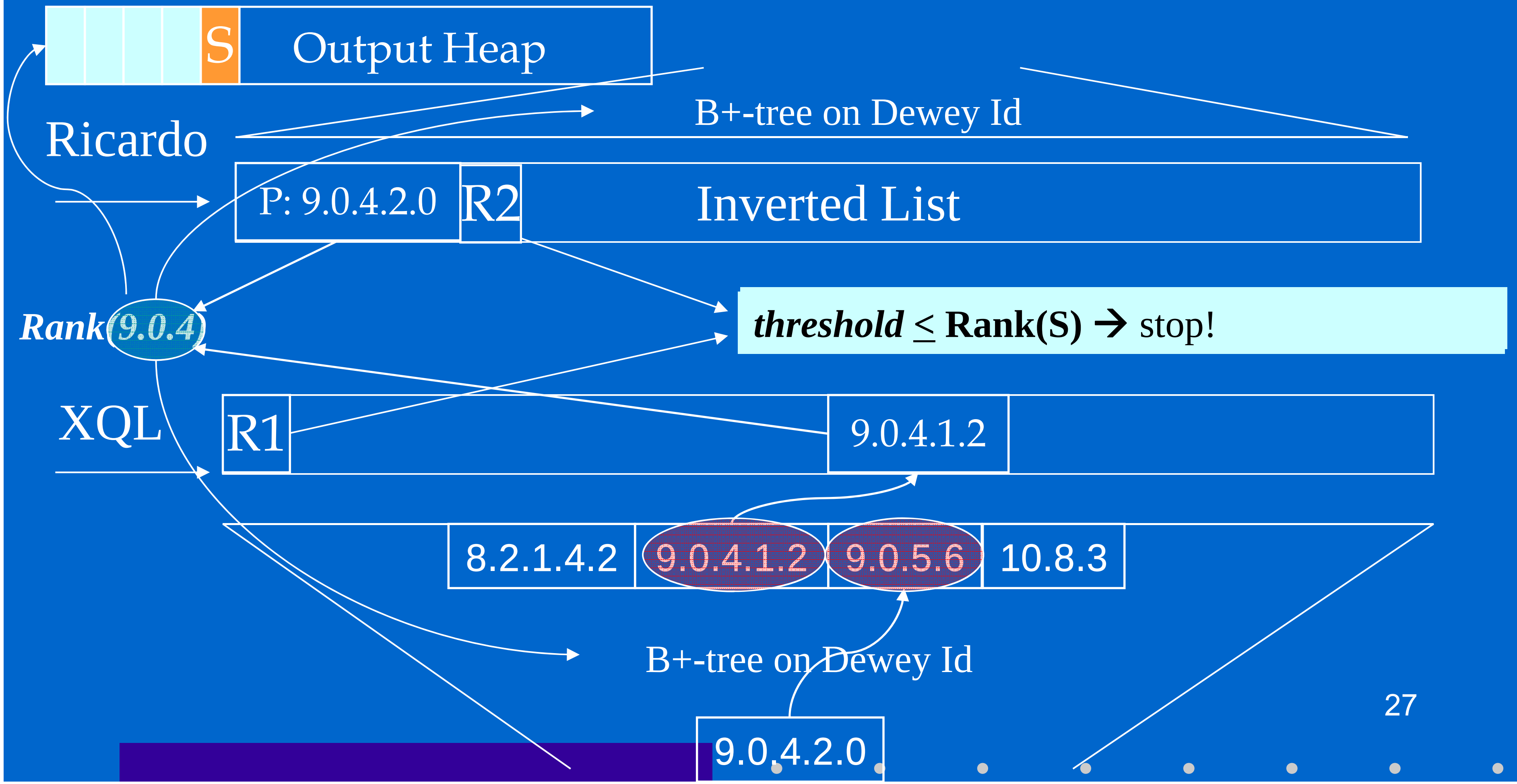
- Algorithm that addresses above issues in a single but FULL scan over inverted lists

RDIL Algorithm

- Advantages
  - Higher ranked results likely to appear first
  - Query processing can be terminated early
- Indexing: order inverted lists by ElemRank instead of DeweyID



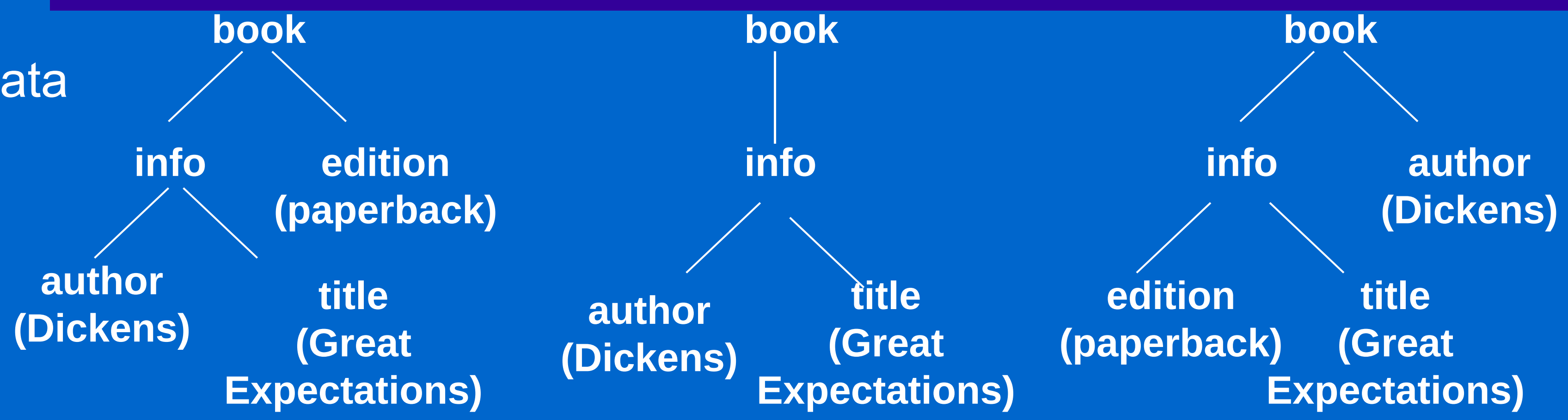
RDIL Query Process



XML Query Relaxation [VLDB 2005]

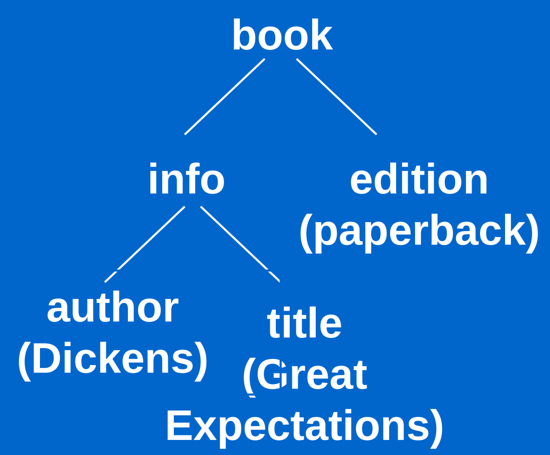
Motivations: XML Data Heterogeneity

Data



Query:

book[./info[./title="Great Expectations" and ./author="Dickens"] and ./edition="paperback"]

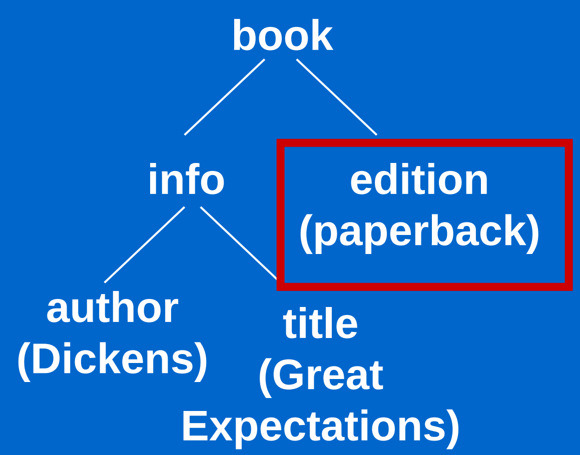


XML Query Relaxation

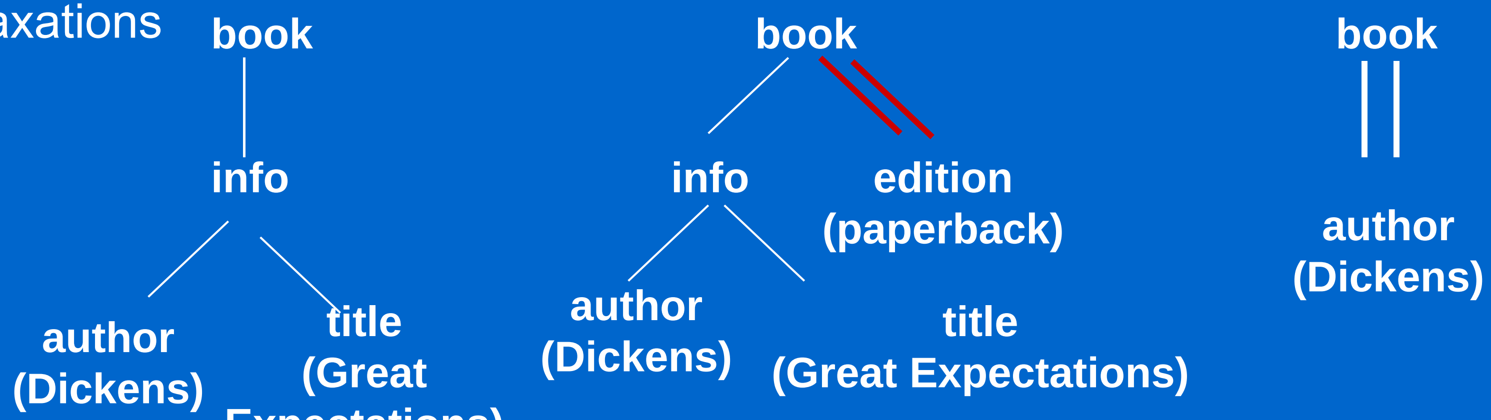
[Amer-Yahia et al. EDBT’02]

- Tree pattern relaxations:
  - Leaf node deletion
  - Edge generalization
  - Subtree promotion

Query



Relaxations



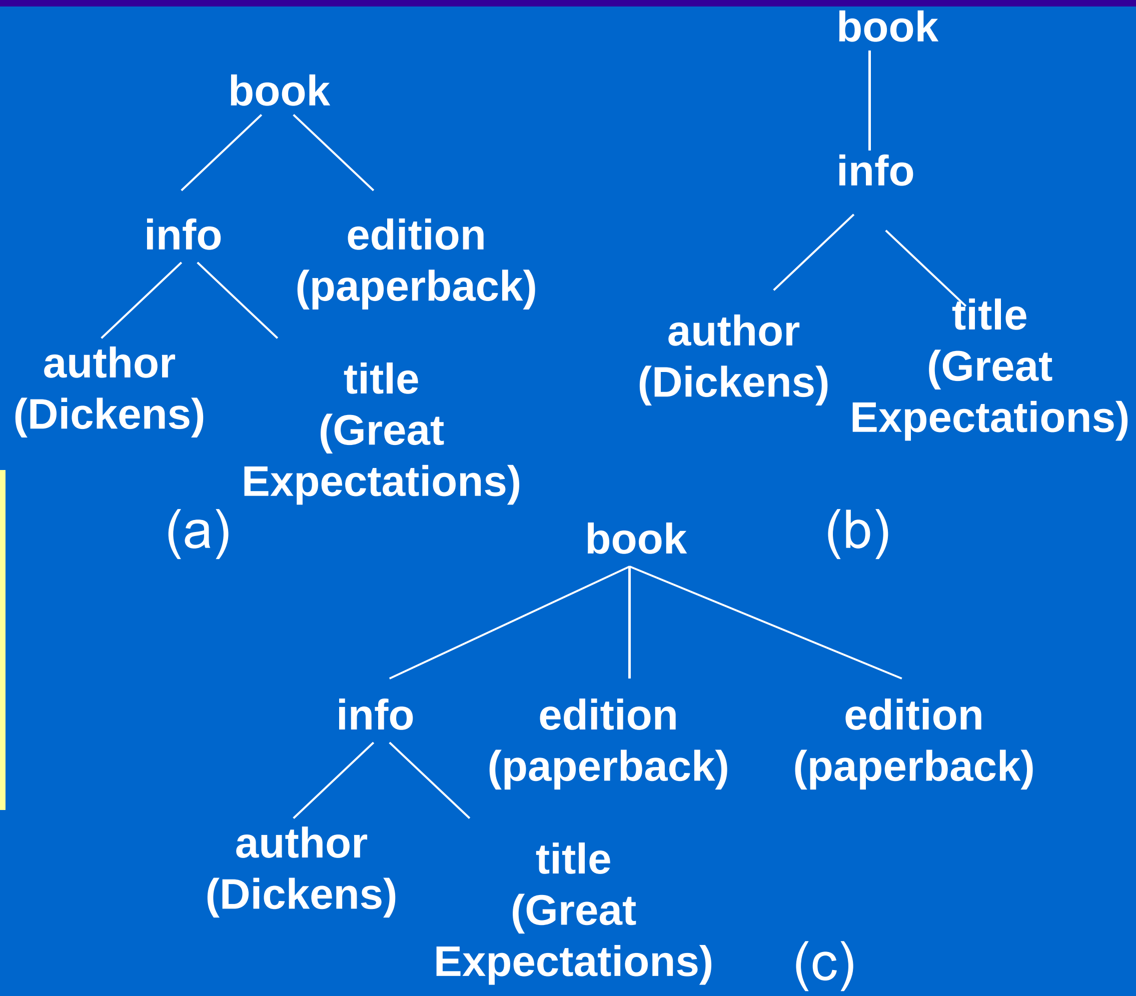
Scoring Function for XML Approximate Matches

Exact matches should be scored higher than relaxed matches (*idf*)

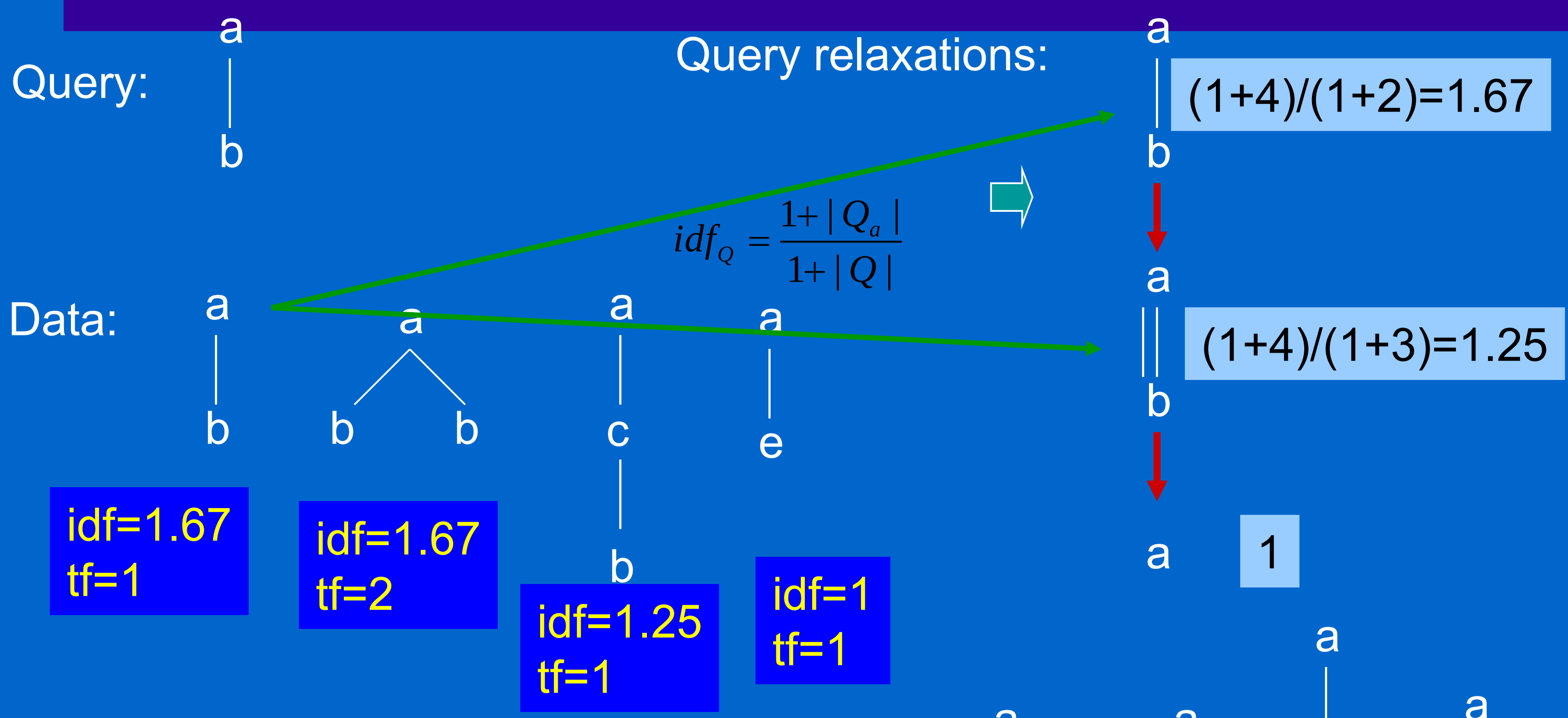
score(a)>=score(b)

Distinguished nodes with several matches should be ranked higher than those with fewer matches (*tf*)

score(a)<=score(c)

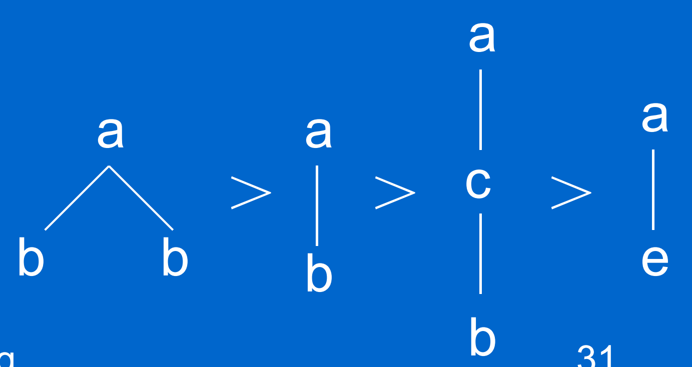


Scoring



$$idf_Q = \frac{1+|Q_a|}{1+|Q|}$$

d ≤ d'   if   *idf*(d) < *idf*(d')
or   *idf*(d)=*idf*(d') and *tf*(d) ≤*tf*(d')



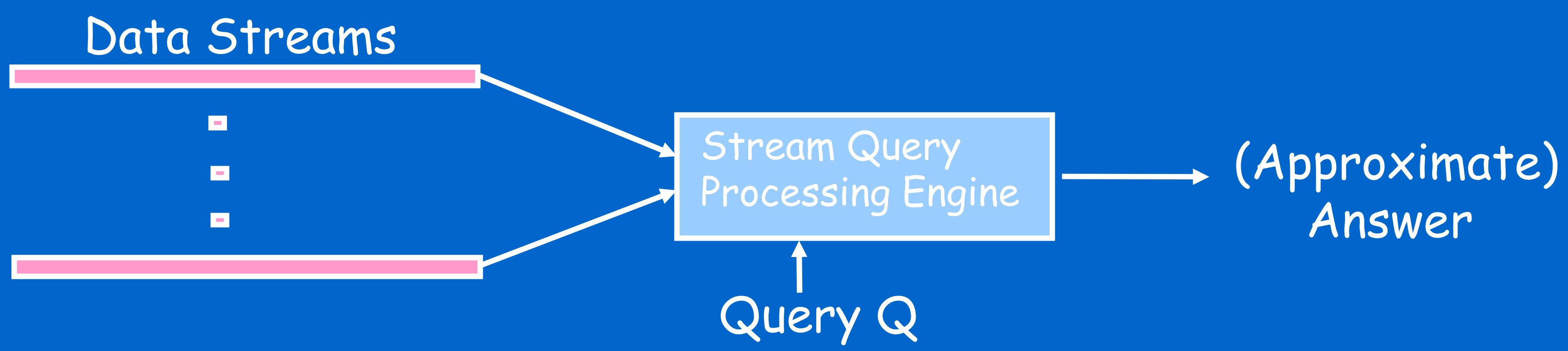
Summary

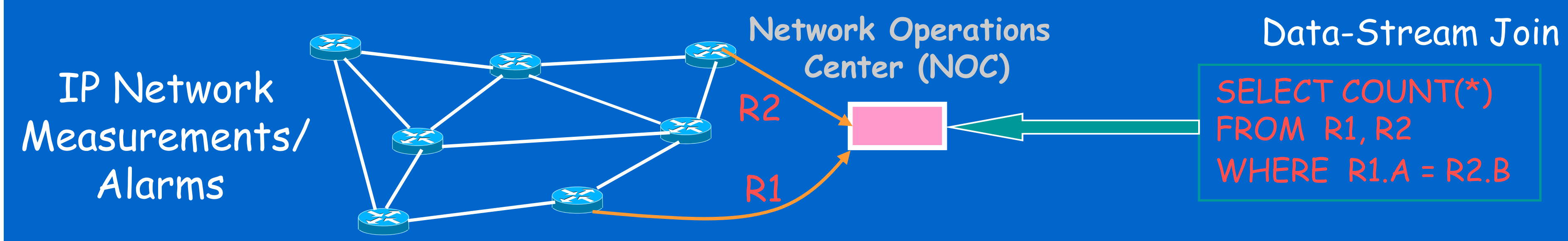
- Score value should reflect relevance of answer to user query.
  - Higher scores imply a higher degree of relevance.
- Queries return document fragments.
  - Granularity of returned results affects scoring.
- For queries containing conditions on structure
  - structural conditions may affect scoring.
- Existing proposals extend common scoring methods:
  - PageRank or vector-based similarity.
- Scoring for keyword search
  - No structure constraint in query
  - How to take document structure into account
  - How about semantic information

XML Data Streams

- A data stream is a massive, continuous sequence of data records
- Requirements for stream query processing
  - *Single Pass:* Each record is examined at most once, in fixed (arrival) order
  - *Small Space:* Log or poly-log in data stream size
  - *Real-time:* Per-record processing time must be low
- Example Application: Network Management
- Correlating XML Data Streams Using Tree-Edit Distance Embeddings
  - Minos Garofalakis & Amit Kumar, PDOS 2003

Query Processing over Data Streams

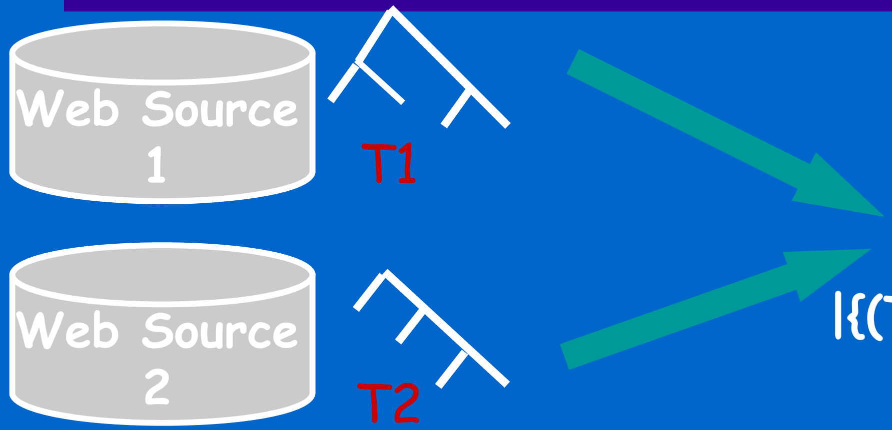




Processing XML Data Streams

- XML: Much richer, (semi)structured data model
  - Ordered, node-labeled data trees
- Bulk of work on XML streaming: *Content-based filtering of XML documents* (publish/subscribe systems)
  - Quickly match incoming documents against standing XPath subscriptions

XML stream correlation query: *Similarity-Join*

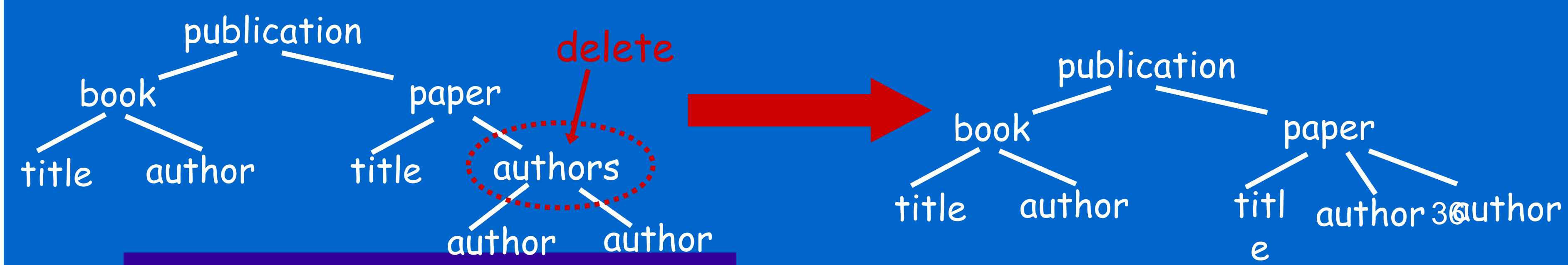


Different data representation for same information (DTDs, optional elements)

$$|\mathrm{SimJoin}(S1,\ S2)| = |\{(T1,T2) \in S1 \times S2:\ \mathrm{dist}(T1,T2) \leq \theta\}|$$

Degree of content similarity between streaming XML sources

- Correlation metric: *Tree-edit distance ed(T1,T2)*
  - Node relabels, inserts, deletes - also, allow for *subtree moves*



Approach

- Construct low-distortion embedding for tree-edit distance over *streaming* XML documents -- Requirements:
  - *Small space/time*
  - *Oblivious:* Can compute V(T) independent of other trees in the stream(s)

- An algorithm for low-distortion, oblivious embedding of the tree-edit distance metric in small space/time
  - Fully deterministic, embed into L1 vector space $\leq$
  - Bound of $O(\log^2 n \log^* n)$ on distance distortion for trees with n nodes

$$\| V(S) - V(T) \|_1 = \sum | V(S)[i] - V(T)[i] | = O(\log^2 n \log^* n) \cdot ed(S,T)$$

Approach

- Applications in XML stream query processing
  - Combine the embedding with existing pseudo-random sketching techniques
  - Build a small-space sketch synopsis for a massive, streaming XML data tree
    - Concise surrogate for tree-edit distance computations
  - Approximating tree-edit distance similarity joins over XML streams in small space/time

Embedding Algorithm

- *Key Idea:* Given an XML tree T, build a *hierarchical parsing structure* over T by intelligently grouping nodes and contracting edges in T
  - At parsing level i: T(i) is generated by grouping nodes of T(i-1) (T(0) = T)
  - Each node in the parsing structure (T(i), for all i = 0, 1, ...) corresponds to a connected subtree of T
  - Vector image V(T) is basically the *characteristic vector* of the resulting multiset of subtrees (in the entire parsing structure)

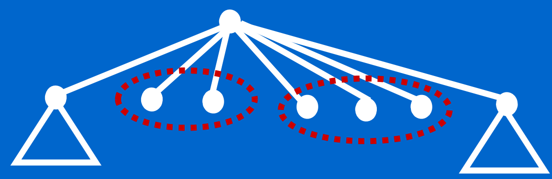
V(T)[x] = no. of times subtree x appears in the parsing structure for T

Embedding Algorithm

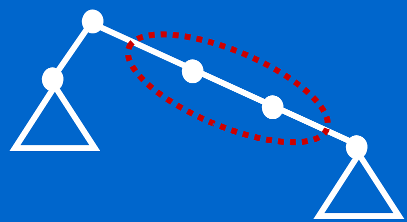
- The parsing guarantees
  - O(log|T|) parsing levels (constant-fraction reduction at each level)
  - V(T) is *very sparse:* Only O(|T|) non-zero components in V(T)
    - Even though dimensionality = $O((4|\sigma|)^n)$ (σ = label alphabet)
    - Allows for effective sketching
  - V(T) is constructed in time $O(|T|\log^*|T|)$

Embedding Algorithm

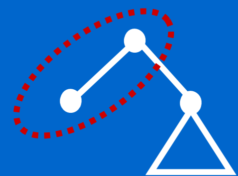
- Node grouping at a given parsing level T(i): Create groups of 2 or 3 nodes of T(i) and merge them into a single node of T(i+1)
  - 1. Group maximal sequence of contiguous leaf children of a node



  - 2. Group maximal sequence of contiguous nodes in a chain



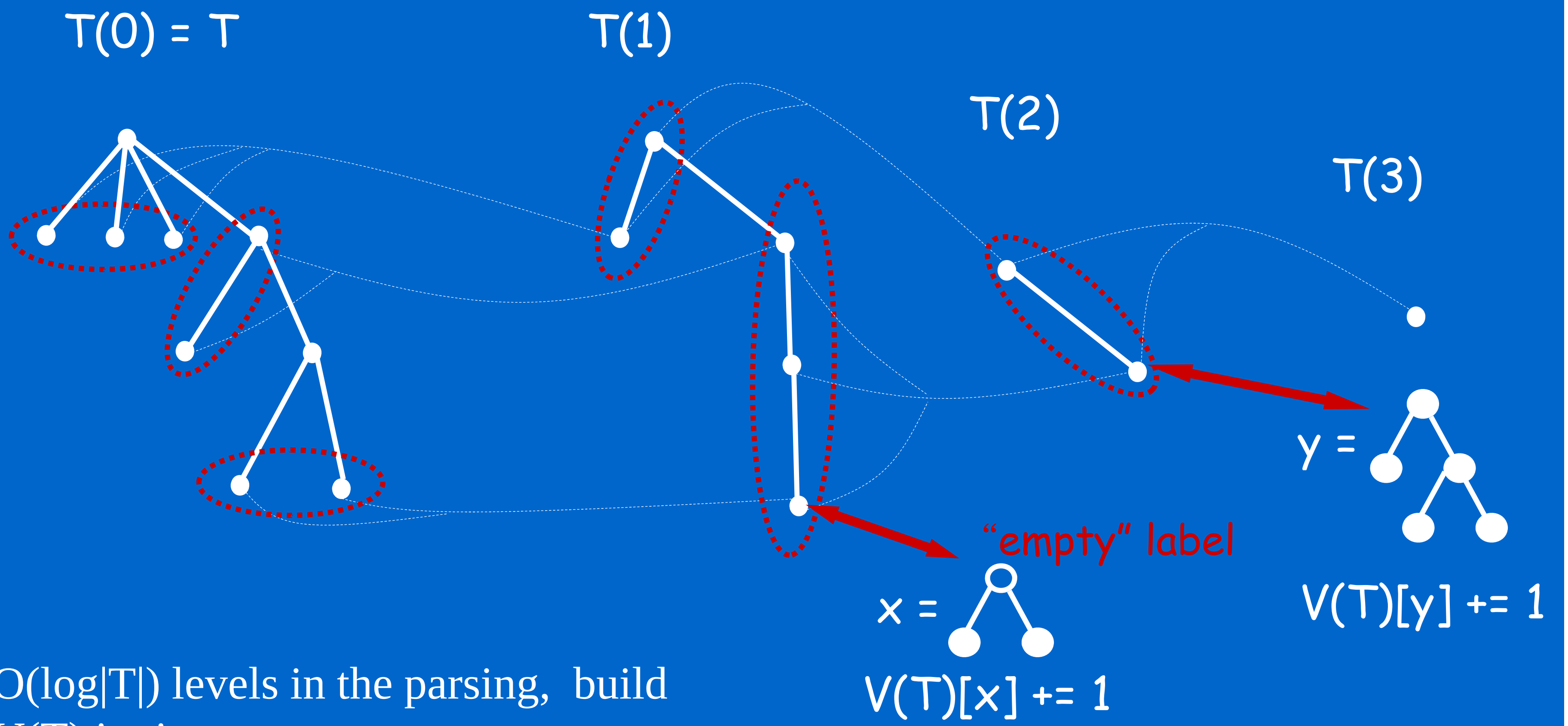
  - 3. Fold leftmost lone leaf child into parent



Embedding Algorithm

- Grouping for Cases 1,2: Deterministic coin-tossing process of Cormode and Muthukrishnan [SODA′02]
  - *Key property:* Insertion/deletion in a sequence of length k only affects the grouping of nodes in a radius of $\log^* k + 5$ from the point of change

Hierarchical Tree Parsing Example

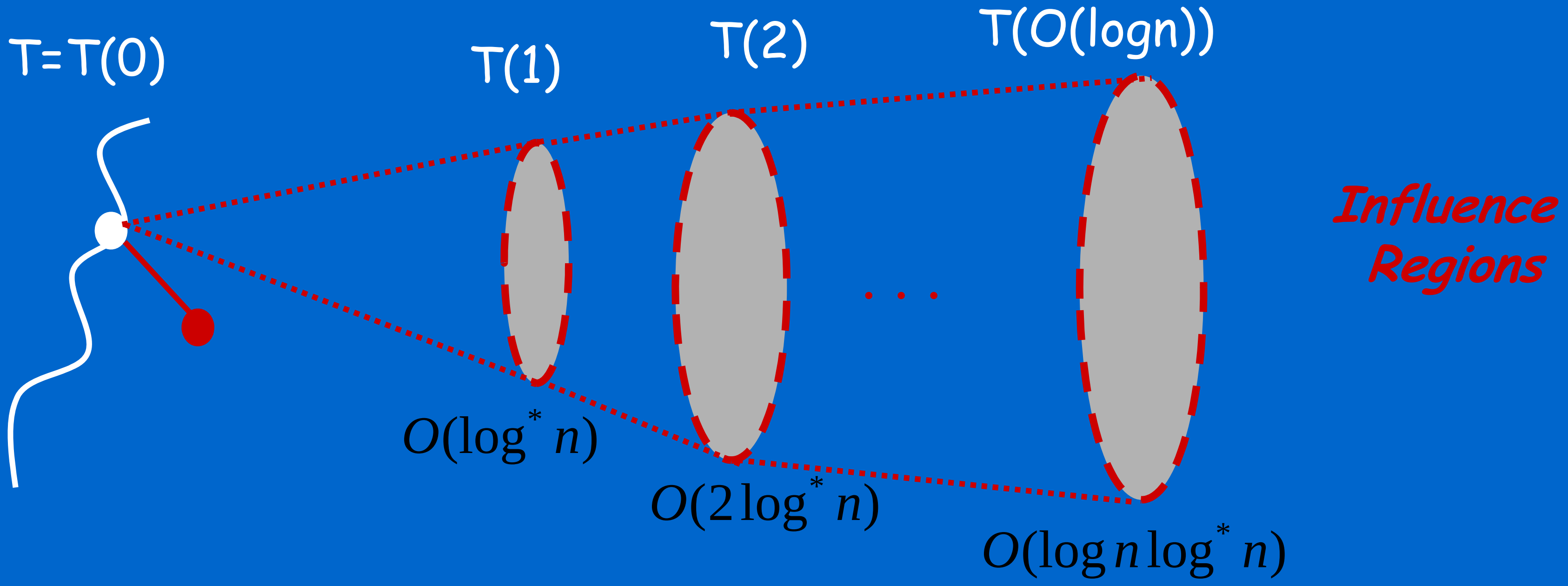


- O(log|T|) levels in the parsing, build V(T) in time $O(|T|\log^{*}|T|)$

Sketching a Massive, Streaming XML Tree

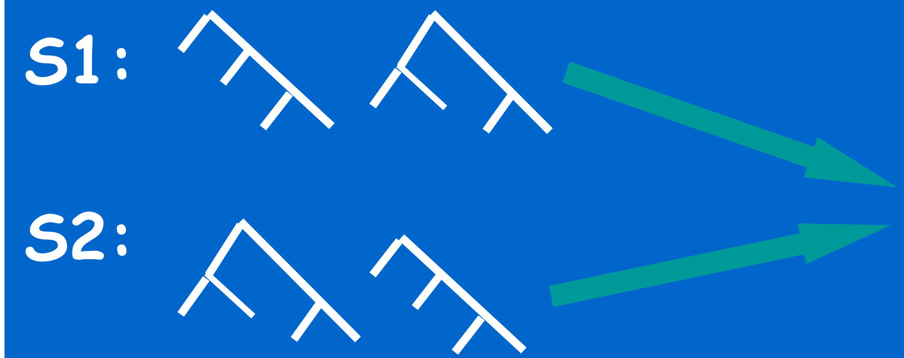
- *Input:* Massive XML data tree T (n = |T| >> available memory), seen in preorder
- *Output:* Small space surrogate (vector) for high-probability, approximate tree-edit distance computations
- *Key Ideas*
  - Incrementally parse T to produce V(T) as elements stream in
  - Just need to retain the *influence region* nodes for each parsing level and for each node in the current root-to-leaf path

Sketching a Massive, Streaming XML Tree



– While updating V(T), also produce an *L1 sketch* of the V(T) vector using the techniques of Indyk

Approximate Similarity Joins over XML Streams



$$|SimJoin(S1, S2)| = |\{(T1,T2) \in S1 \times S2: \ ed(T1,T2) \leq \theta \ \}|$$

- *Input:* Long streams S1, S2 of N (short) XML documents ($\leq$ b nodes)
- *Output:* Estimate for |SimJoin(S1, S2)|

Approximate Similarity Joins over XML Streams

- *Key Ideas*
  - The embedding of streaming document trees, plus two distinct levels of sketching
    - One to reduce L1 dimensionality,   one to capture the data distribution (for joining)
    - Finally, similarity join in lower-dimensional L1 space