Scoring & result assembly

CE-324: Modern Information Retrieval

Sharif University of Technology

M. Soleymani

Fall 2018

Most slides have been adapted from: Profs. Manning, Nayak & Raghavan (CS-276, Stanford)

Outline

- Speeding up vector space ranking
- Putting together a complete search system
  - Will require learning about a number of miscellaneous topics and heuristics

Recap: Queries as vectors

- Vector space scoring
  - We have a weight for each term in each doc
  - Represent queries as vectors in the space
  - Rank documents according to their cosine similarity to the query in this space
    - Or something more complex: BM25, proximity, …
- Vector space scoring is
  - Entirely query dependent
  - Additive on term contributions – no conditionals etc.
  - Context insensitive (no interactions between query terms)

TAAT vs DAAT techniques

- TAAT = "Term At A Time"
  - Scores for all docs computed concurrently, one query term at a time
- DAAT = "Document At A Time"
  - Total score for each doc (incl all query terms) computed, before proceeding to the next
- Each has implications for how the retrieval index is structured and stored

Computing cosine scores

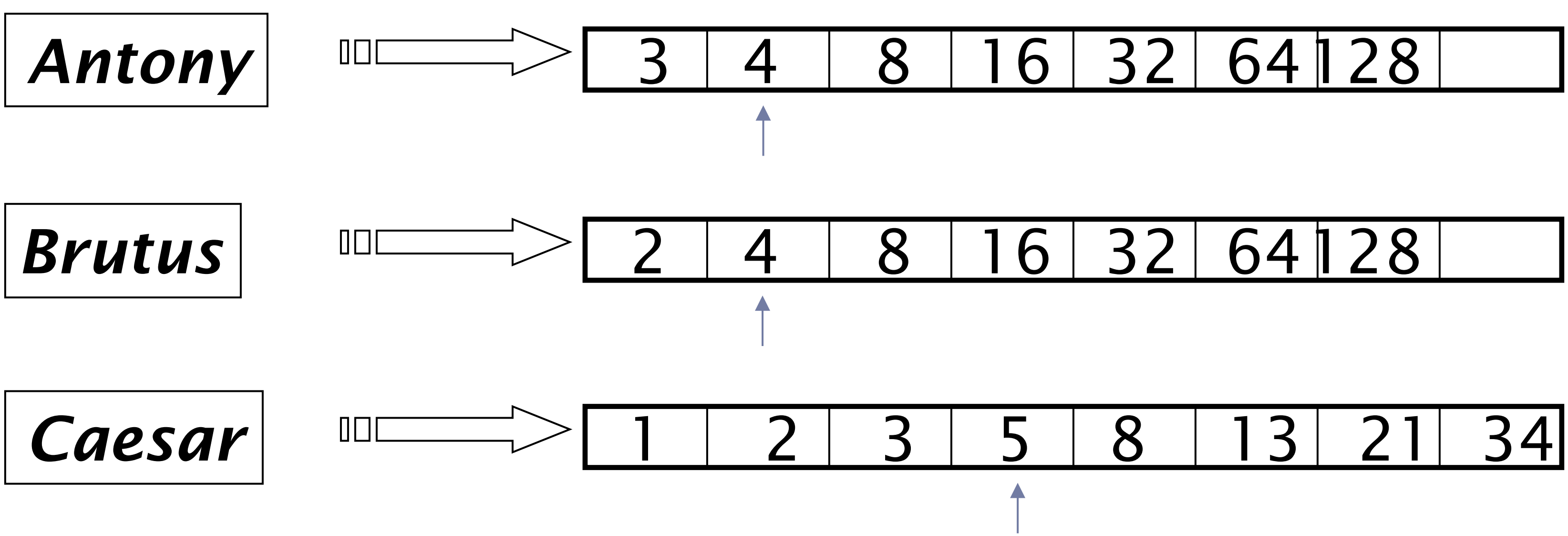
COSINESCORE(q)

float Scores$[N] = 0$
float Length$[N]$
for each query term t
do calculate $w_{t,q}$ and fetch postings list for t
    for each pair$(d, \mathrm{tf}_{t,d})$ in postings list
    do *Scores*$[d] += w_{t,d} \times w_{t,q}$
Read the array *Length*
for each d
do *Scores*$[d]$ = *Scores*$[d]$/*Length*$[d]$
return Top K components of *Scores*$[]$

Term-at-a-time vs. doc-at-a-time processing

- Completely process the postings list of the first query term, then process the postings list of the second query term and so forth

- Doc-at-time



Efficient cosine ranking

- Find the K docs in the collection "nearest" to the query $\Rightarrow$ K largest query-doc cosines.

- Efficient ranking:
  - Choosing the K largest cosine values efficiently.
    - Can we do this without computing all N cosines?

Safe vs non-safe ranking

- The terminology "safe ranking" is used for methods that guarantee that the *K* docs returned are the *K* absolute highest scoring documents
  - (Not necessarily just under cosine similarity)
- Is it ok to be non-safe?
- If it is – then how do we ensure we don't get too far from the safe solution?
  - How do we measure if we are far?

Non-safe ranking

- Non-safe ranking may be okay
  - Ranking function is only a proxy for user happiness
  - Documents close to top K may be just fine
- Index elimination
- Champion lists
- High/low lists, tiered indexes
- Order postings by *g(d)* (query-indep. quality score)

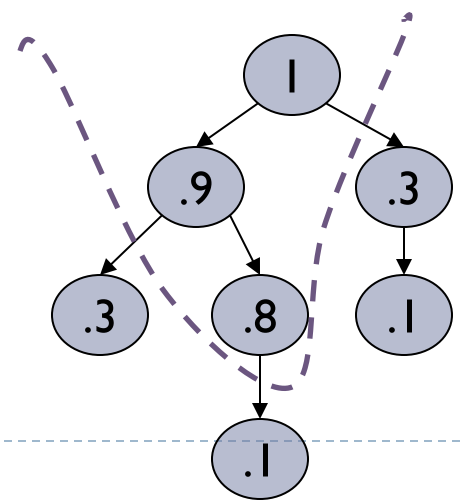
Computing the K largest cosines: selection vs. sorting

- Retrieve the top K docs
  - not to totally order all docs in the collection
- Can we pick off docs with K highest cosines?
- Let J = number of docs with nonzero cosines
  - We seek the K best of these J

Use heap for selecting top K

- Construction: $2J$ operations
- K "winners": $2K \log J$ operations

- For $J = 1M$, $K = 100$, this is about 10% of the cost of sorting.



Efficient cosine ranking

- What we're doing in effect: solving the K -nearest neighbor problem for a query vector
  - In general, we do not know how to do this efficiently for high-dimensional spaces
- But it is solvable for short queries, and standard indexes support this well

Cosine similarity is only a proxy

- Cosine similarity is just a proxy for user happiness
  - If we get a list of K docs "close" to top K by cosine measure, it should be ok

Generic idea of inexact top k search

- Find a set A of *contenders*, with $K < |A| \ll N$
  - A does not necessarily contain the top *K*
    - but has many docs from among the top *K*
- Return the top *K* docs in *A*
- Think of A as <u>pruning</u> non-contenders
- Same approach is also used for other scoring functions
  - Will look at several schemes following this approach

Ideas for more efficient computation of top k

- Index elimination
- Champion lists
- Global ordering
- Impact ordering
- Cluster pruning

Index elimination for cosine computation

- Basic algorithm: considers docs containing at least one query term

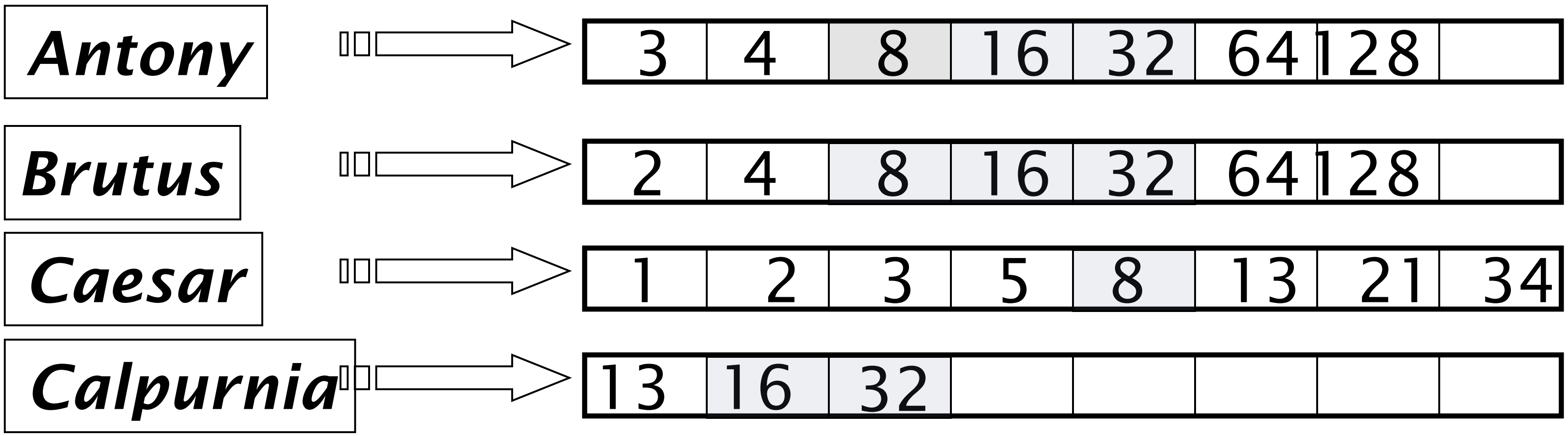
- Extend this basic algorithm to:
  - Only consider docs containing many (or all) query terms
  - Only consider high-idf query terms

Docs containing many query terms

When we have multi-term queries

- Only compute scores for docs containing several of the query terms
  - Say, at least 3 out of 4
  - Imposes a "soft conjunction" on queries seen on web search engines (early Google)
  - May find fewer than k candidates
- Easy to implement in postings traversal

3 of 4 query terms



Scores only computed for docs 8, 16 and 32.

High-idf query terms only

- Query: ***catcher in the rye***
  - Only accumulate scores from ***catcher*** and ***rye***
- Intuition: ***in*** and ***the*** contribute little to the scores and so <u>don't alter rank-ordering much</u>
- Benefit:
  - Postings of low-idf terms have many docs
    - $\rightarrow$ many docs are eliminated from set *A* of contenders

Champion lists

- r docs of highest weight in the posting list of each dictionary term
  - Call this the champion list for t
    - aka fancy list or top docs for t
- At query time, only compute scores for docs in the champion list of some (or all of) query terms
  - Pick the K top-scoring docs from amongst these
- Note that r has to be chosen at index build time
  - Thus, it's possible that the obtained list of docs contains less than K docs

High and low lists

- For each term, two postings lists <u>*high*</u> and <u>*low*</u>
  - High: like the champion list
  - Low: all other docs containing t
- Only traverse *high* lists first
  - If we get more than K docs, select top K and stop
  - Else proceed to get docs from *low* lists
- A means for segmenting index into two <u>tiers</u>

Static quality scores

- Top-ranking docs needs to be both *relevant* and *authoritative*
  - *Relevance:* modeled by cosine scores
  - *Authority:* typically a query-independent property of a doc

- Examples of authority signals
  - Wikipedia among websites
  - Articles in certain newspapers
  - A paper with many citations
  - Pagerank

Quantitative

Modeling authority

- Assign to each doc d a ***query-independent*** quality score in [0,1] (called $g(d)$)
  - A quantity like the number of citations scaled into [0,1]

Net score

- Simple total score: combining cosine relevance and authority

$$\text{NetScore}(q, d) = g(d) + cosine(q, d)$$

  - Can use some other linear combination
  - Indeed, any function of the two "signals" of user happiness

- Now we seek the top K docs by <u>net score</u>

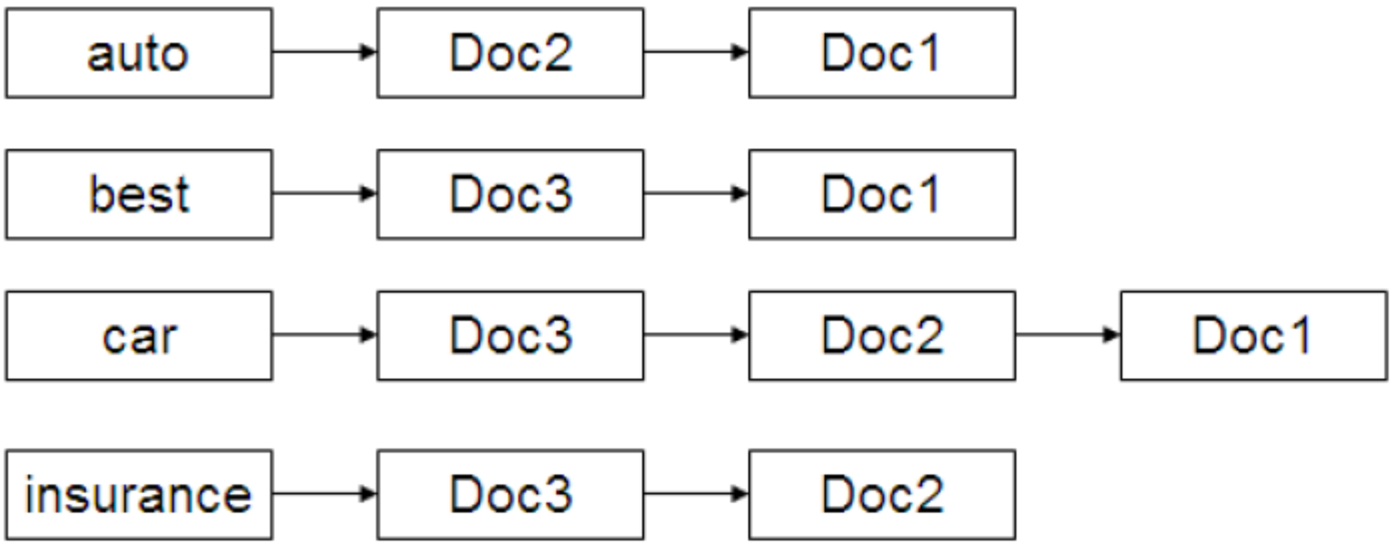
Top K by net score – fast methods

- First idea: Order all postings by $g(d)$

- Key: this is a <u>common ordering for all postings</u>
  - All postings are ordered by a single common ordering
  - and the merge is then performed by a single pass through the postings

- Can concurrently traverse query terms' postings for
  - Postings intersection
  - Cosine score computation

Static quality-ordered index



$g(1) = 0.25$
$g(2) = 0.5$
$g(3) = 1$

Why order postings by $g(d)$?

- $g(d)$-ordering: top-scoring docs likely to appear early in postings traversal

- In time-bound applications:
  - It allows us to stop postings traversal early
    - E.g., we have to return search results in 50 ms

Global champion lists

- Can combine champion lists with $g(d)$-ordering?

- Maintain for each term a champion list of r docs with highest $g(d) + \mathrm{tf.idf}_{td}$
  - Sorted by a common order $g(d)$

- Seek top-K results from only the docs in these champion lists

Impact-ordered postings

- If we have impact ordering
  - Docs in the top k are likely to occur early in the ordered lists.
- We sort each postings list according to weight $wf_{t,d}$
  - Simplest case: normalized tf-idf weight
    - ⇒ Early termination while processing postings lists is unlikely to change the top k.

Term-at-a-time processing

```
FASTCOSINESCORE(q)
 float Scores[N] = 0
for each d
do Initialize Length[d] to the length of doc d
for each query term t
do calculate w_{t,q} and fetch postings list for t
    for each pair(d, tf_{t,d}) in postings list
    do add wf_{t,d} to Scores[d]
Read the array Length[d]
for each d
do Divide Scores[d] by Length[d]
return Top K components of Scores[]
```

Impact-ordered postings

- Now: not all postings in a common order!
  - no longer a consistent ordering of docs in postings lists.
  - no longer can employ document-at-a-time processing

- Term-at-a-time processing
  - Create an accumulator for each docID you encounter

- How do we compute scores in order to pick off inexact top K?
  - Early termination
  - idf-ordered terms

1. Early termination

- When traversing t's postings, stop early after either
  - a fixed number of r docs
  - $wf_{t,d}$ drops below some threshold

2. idf-ordered terms

- When considering the postings of query terms
  - Look at them in order of decreasing idf
    - High idf terms likely to contribute most to score
  - As we update score contribution from each query term we can stop when doc scores are relatively unchanged
    - If the changes are minimal, we may omit accumulation from the remaining query terms
    - or alternatively process shorter prefixes of their postings lists.

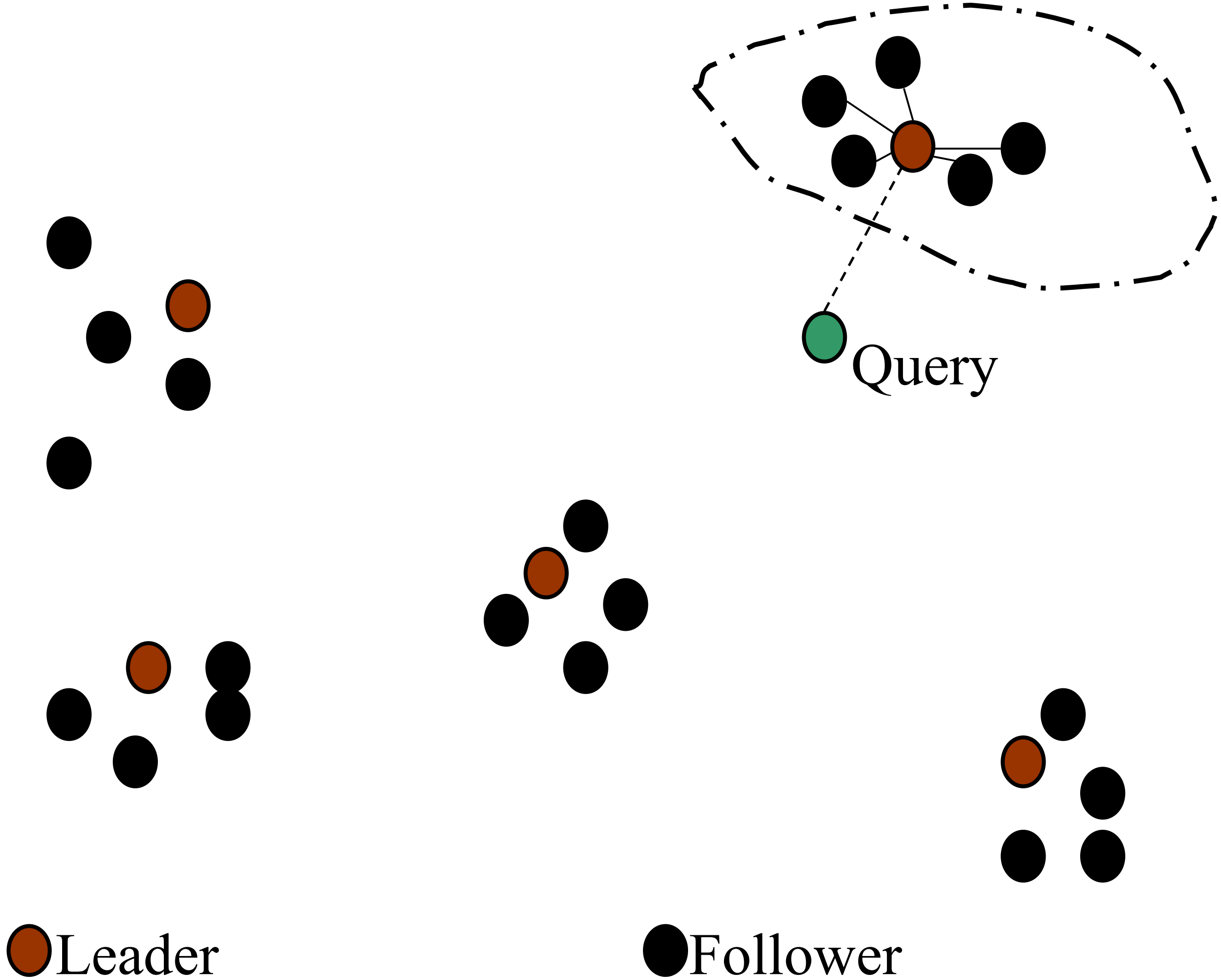
Cluster pruning: preprocessing

- *Leaders:* $\sqrt{N}$ *docs* at random
- For every other doc, pre-compute nearest leader
  - *Followers:* Docs attached to a leader
  - Likely: each leader has ~ $\sqrt{N}$ followers.
- Why random sampling for finding leaders:
  - Fast approach
  - Leaders reflect data distribution

Cluster pruning: query processing

- Given query Q, find its nearest *leader* L.
- Seek K nearest docs from among *L*'s followers.

Visualization



General variants

- Have each follower attached to b_1 nearest leaders.
- From query, find b_2 nearest leaders and their followers.
- Can recurse on leader/follower construction.

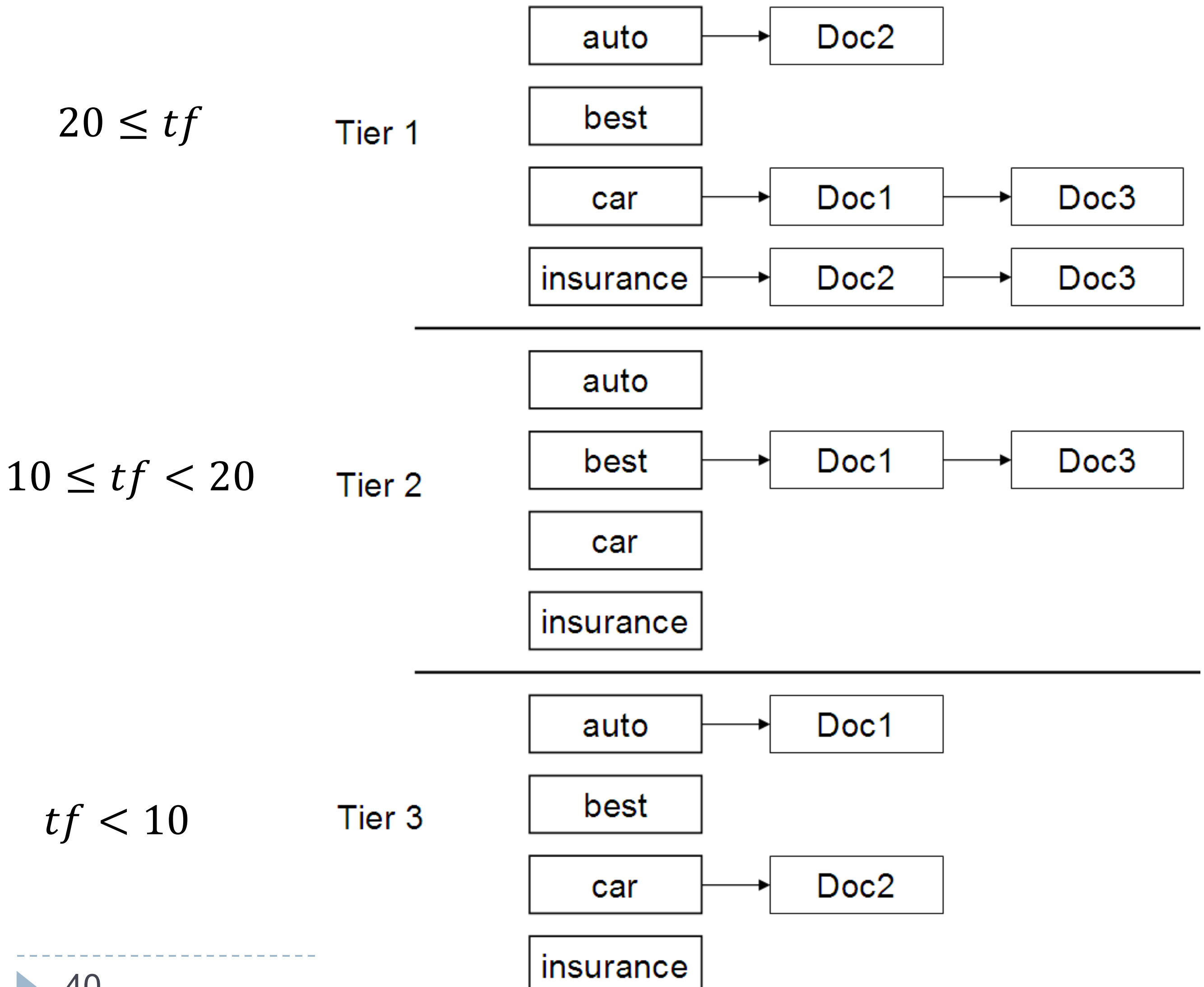
Tiered indexes

- Basic idea:
  - Create several tiers of indexes
  - During query processing, start with highest-tier index
  - If highest-tier index returns at least k (e.g., k = 100) results:
    - stop and return results to user
  - If we've only found < k hits: repeat for next index in tier cascade

Tiered indexes

- Break postings up into a hierarchy of lists
  - Most important to least important
  - Can be done by $g(d)$ or another measure
- Inverted index ⇒ tiers of decreasing importance
- At query time use top tier unless it fails to yield K docs
  - If so drop to lower tiers
- Tiered indexes as one of the reasons for the success of early Google (2000/01)
  - along with PageRank, use of anchor text and proximity constraints

Example tiered index



Safe ranking

- When we output the top *K* docs, we have a proof that these are indeed the top *K*
- Does this imply we always have to compute all *N* cosines?
  - We'll look at pruning methods
  - So we only fully score some *J* documents
- Do we have to sort the *J* cosine scores?

WAND scoring

- An instance of DAAT scoring
- Basic idea reminiscent of branch and bound
  - We maintain a running *threshold* score – e.g., the K^{th} highest score computed so far
  - We prune away all docs whose cosine scores are guaranteed to be below the threshold
  - We compute exact cosine scores for only the un-pruned docs

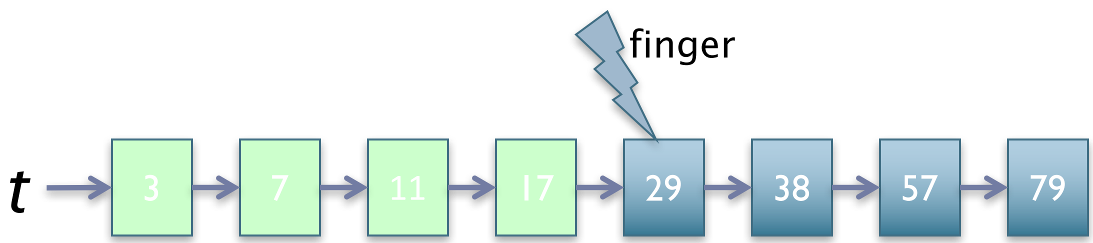
Broder et al. Efficient Query Evaluation using a Two-Level Retrieval Process.

Index structure for WAND

- Postings ordered by docID
- Assume a special iterator on the postings of the form "go to the first docID greater than or equal to *X*"
- Typical state: we have a "finger" at some docID in the postings of each query term
  - Each finger moves only to the right, to larger docIDs
- Invariant – all docIDs lower than any finger have already been *processed*, meaning
  - These docIDs are either pruned away or
  - Their cosine scores have been computed

Upper bounds

- At all times for each query term t, we maintain an *upper bound* UB_t on the score contribution of any doc to the right of the finger
  - Max (over docs remaining in t's postings) of $w_t(\text{doc})$



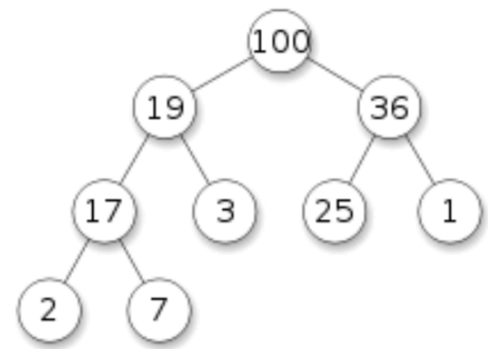
$UB_t = w_t(38)$

As finger moves right, *UB* drops

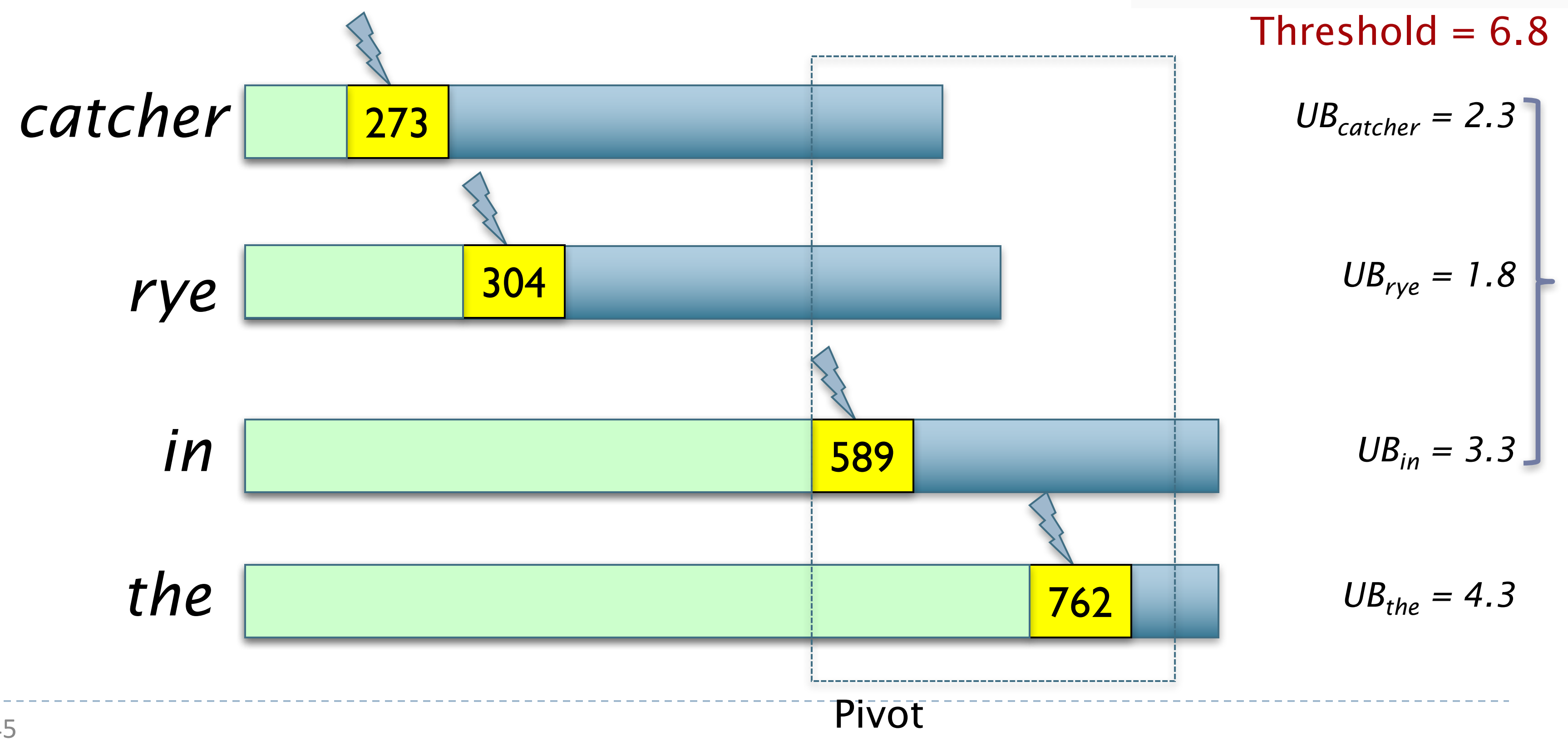
Pivoting

- Query: *catcher in the rye*
- The current finger positions are as below

Top-k

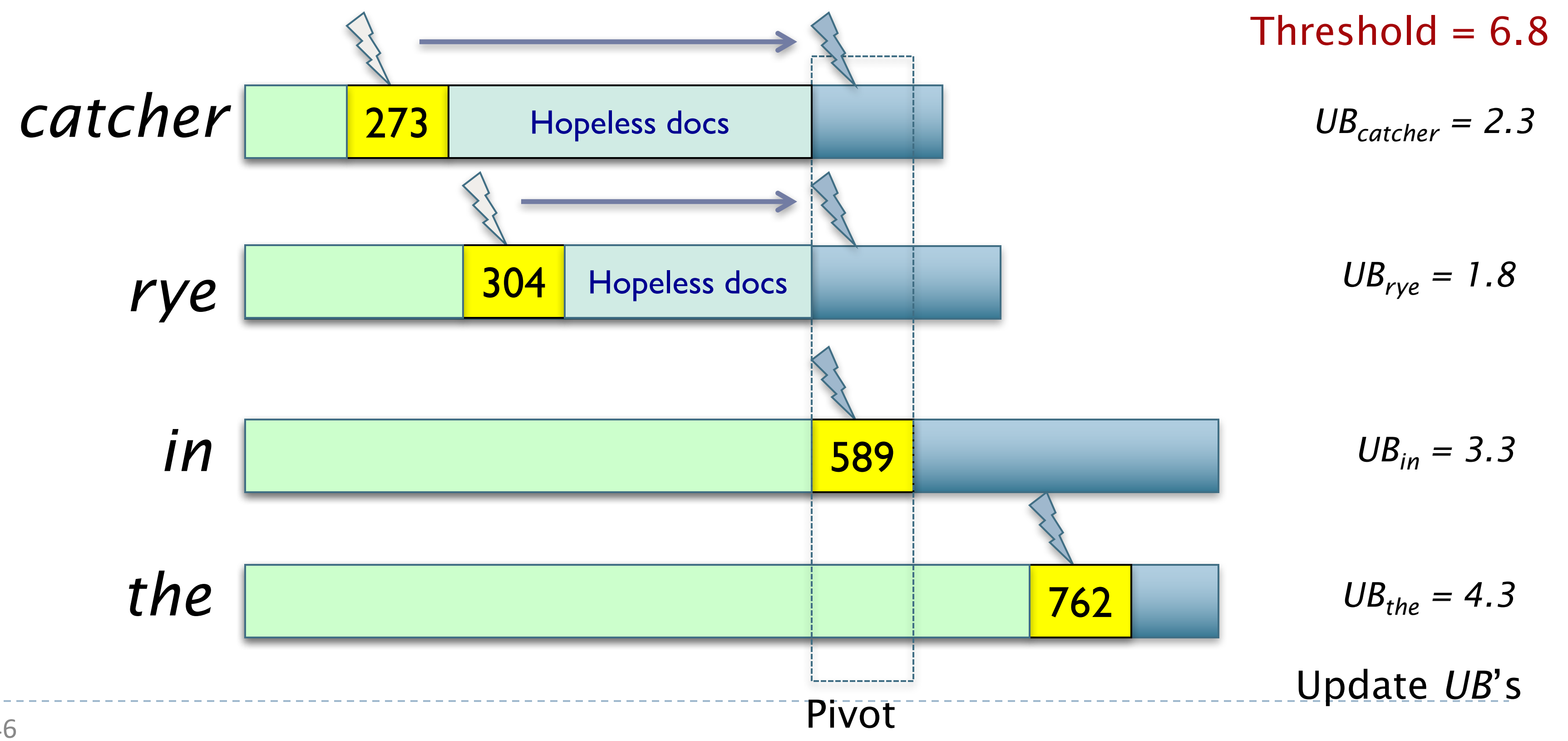


Scores>=6.8



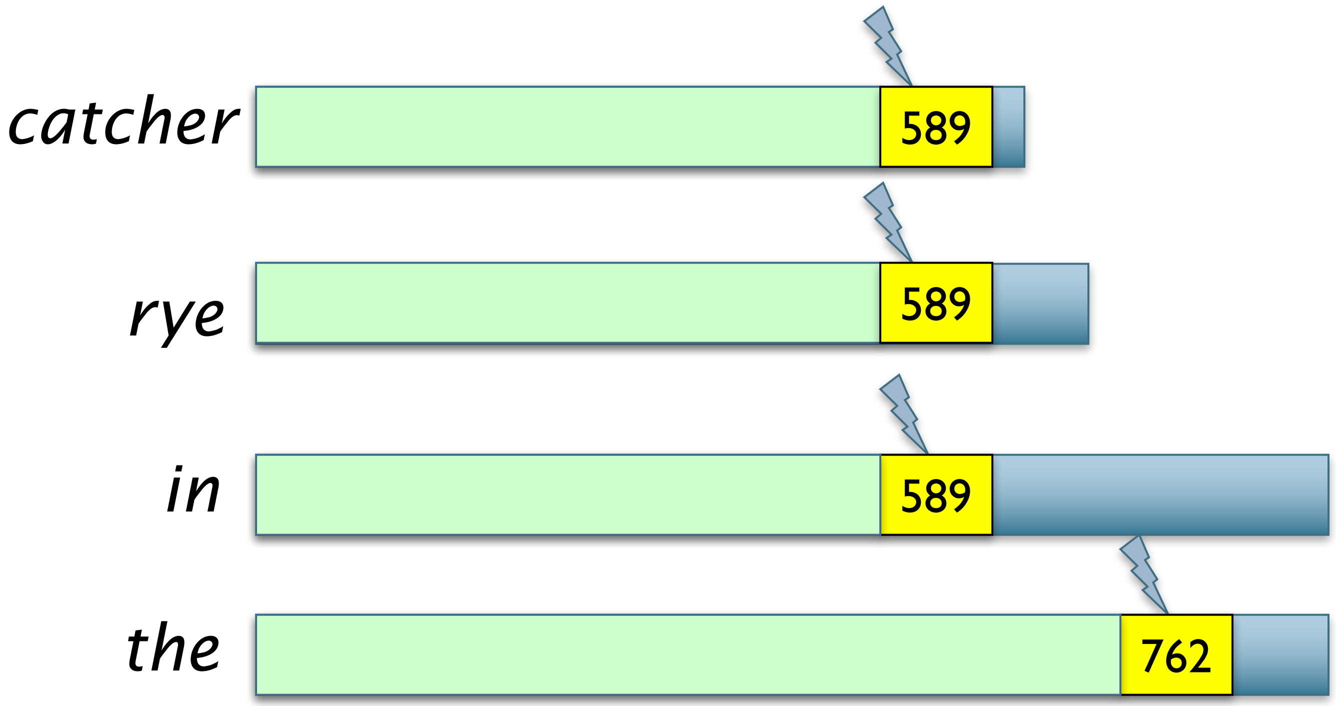
Prune docs that have no hope

- Terms sorted in order of finger positions
- Move fingers to 589 or right



Compute 589's score if need be

- If 589 is present in enough postings, compute its full cosine score – else some fingers to right of 589
- Pivot again …



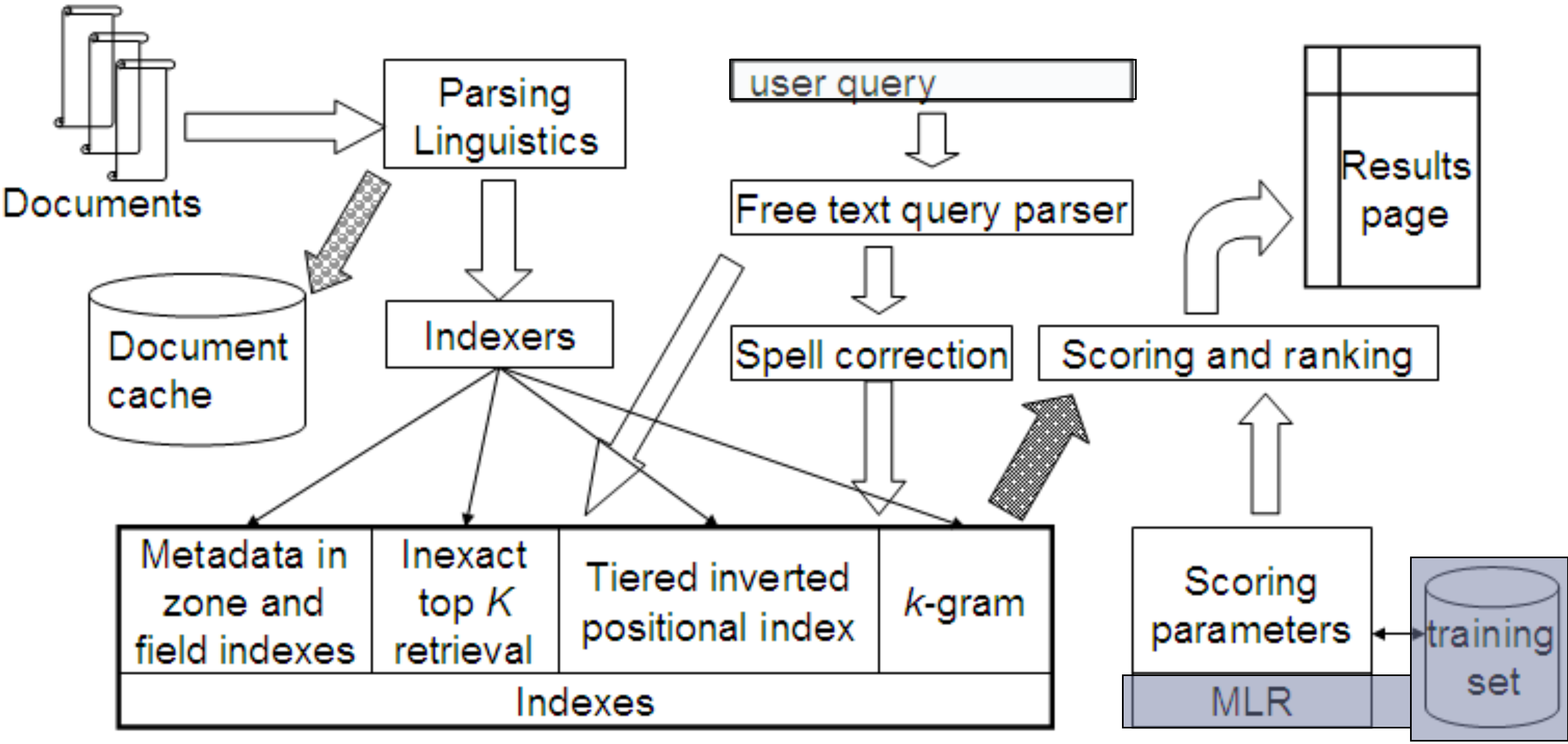
WAND summary

- In tests, WAND leads to a 90+% reduction in score computation
  - Better gains on longer queries
- Nothing we did was specific to cosine ranking
  - We need scoring to be *additive* by term
- WAND and variants give us <u>safe ranking</u>
  - Possible to devise "careless" variants that are a bit faster but not safe (see summary in Ding+Suel 2011)
  - Ideas combine some of the non-safe scoring we considered

Term frequencies in the inverted index

- In each posting, store $tf_{t,d}$ in addition to docID
  - As an integer frequency, not as a (log-)weighted real number
    - because real numbers are difficult to compress.
    - overall, additional space requirements are small: a byte per posting or less

Complete search system



Components we haven't covered yet

- Proximity ranking
  - rank docs in which the query terms occur in the same local window higher
- Query parser
- Zone indexes
- Document cache: we need this for generating snippets
- Machine-learned ranking functions

Query term proximity

- Free text queries: just a set of terms
  - Users may prefer docs in which query terms occur within close proximity of each other
  - *w*: smallest window in a doc containing all query terms
    - Query: ***strained mercy***
    - Doc: "*The quality of mercy is not strained*"
    - w: 4 (words)
  - Would like scoring function to take this into account – how?

Query parsers

- Free text query from user may spawn one or more queries to the indexes
  - Run the query as a phrase query
  - If $<K$ docs contain the phrase run smaller phrase queries
  - If we still have $<K$ docs, run the vector space query
  - Rank matching docs by vector space scoring

- This sequence is issued by a <u>query parser</u>

Query parsers

- Example:
  - Query: ***rising interest rates***
  - If $< K$ docs contain

    "***rising interest rates***"

    run queries

    "***rising interest***" and "***interest rates***"
  - If we still have $< K$ docs, run the vector space query

    rising interest rates

- We need aggregate scoring function that *accumulates evidence* of a doc's relevance from multiple sources

Aggregate scores

- Score functions can combine cosine, static quality, proximity, etc.

- How do we know the best combination?
  - Some applications – expert-tuned
  - Increasingly common: machine-learned

Vector space retrieval: Interactions

- Combining **Boolean retrieval** with vector space retrieval?
  - no easy way of combining vector space and Boolean queries
  - Postfiltering is simple, but can be very inefficient – no easy answer.
- Combining **phrase retrieval** with vector space retrieval?
  - no way of demanding a vector space score for a phrase query
  - can in some cases be combined usefully (query parser)
- Combining **wild cards** with vector space retrieval?

Parametric and zone indexes

- Thus far, a doc has been a sequence of terms
- In fact docs have multiple parts, some with special semantics:
  - Author
  - Title
  - Date of publication
  - Language
  - Format
  - etc.
- These constitute the metadata about a document

Fields

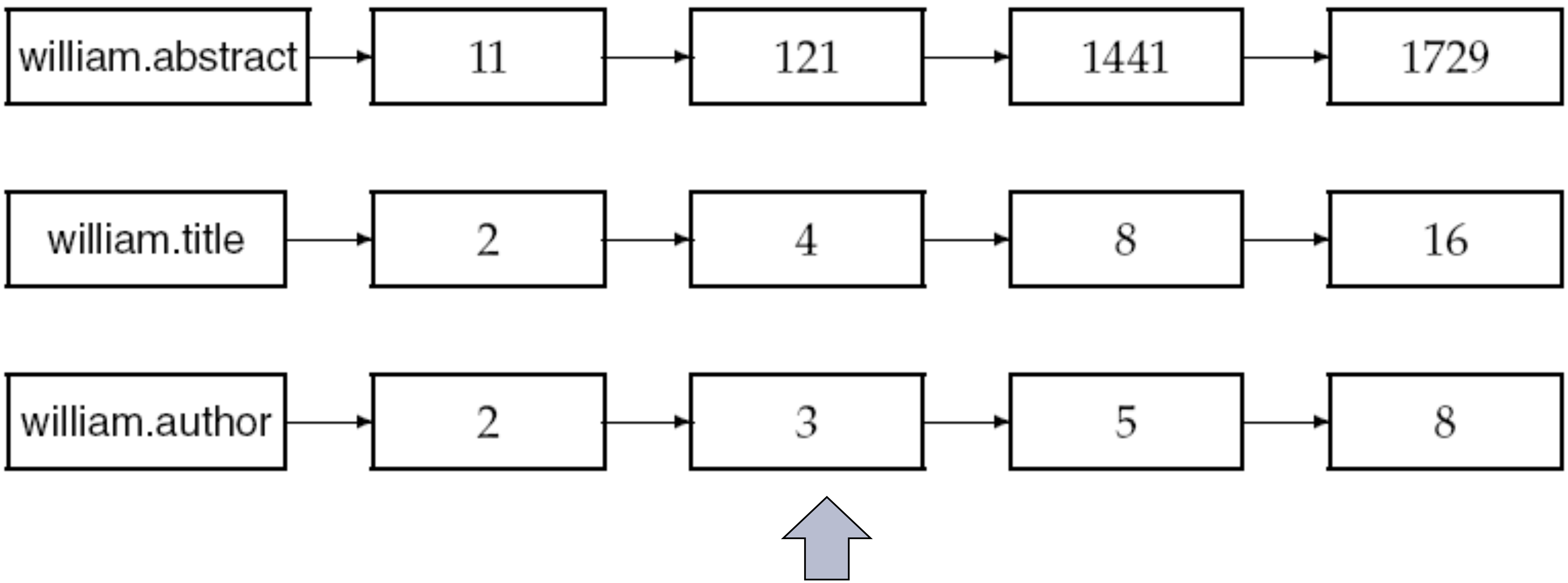
- We sometimes wish to search by these metadata
  - E.g., find docs authored by William Shakespeare in the year 1601, containing *alas poor Yorick*
    - Year = 1601 is an example of a field
    - Author last name = shakespeare
- Field or parametric index: postings for each field value
  - Sometimes build range trees (e.g., for dates)
- Field query typically treated as conjunction
  - (doc *must* be authored by shakespeare)

Zone

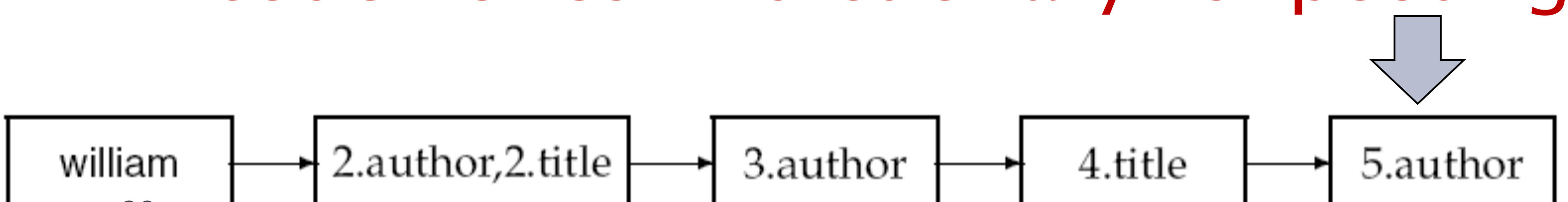
- A zone is a region of the doc that can contain an arbitrary amount of text, e.g.,
  - Title
  - Abstract
  - References …
- Example: “find docs with *merchant* in the title zone and matching the query *gentle rain*”
- Build inverted indexes on zones (to permit querying)

Zone examples: the body of the doc, all highlighted text in the doc, anchor text, text in metadata fields

Example zone indexes



Encode zones in dictionary vs. postings.



Resources

- IIR 7, 6.1
- Resources at http://ifnlp.org/ir
  - How Google tweaks its ranking function
  - Interview with Google search guru Udi Manber
  - Amit Singhal on Google ranking
  - SEO perspective: ranking factors