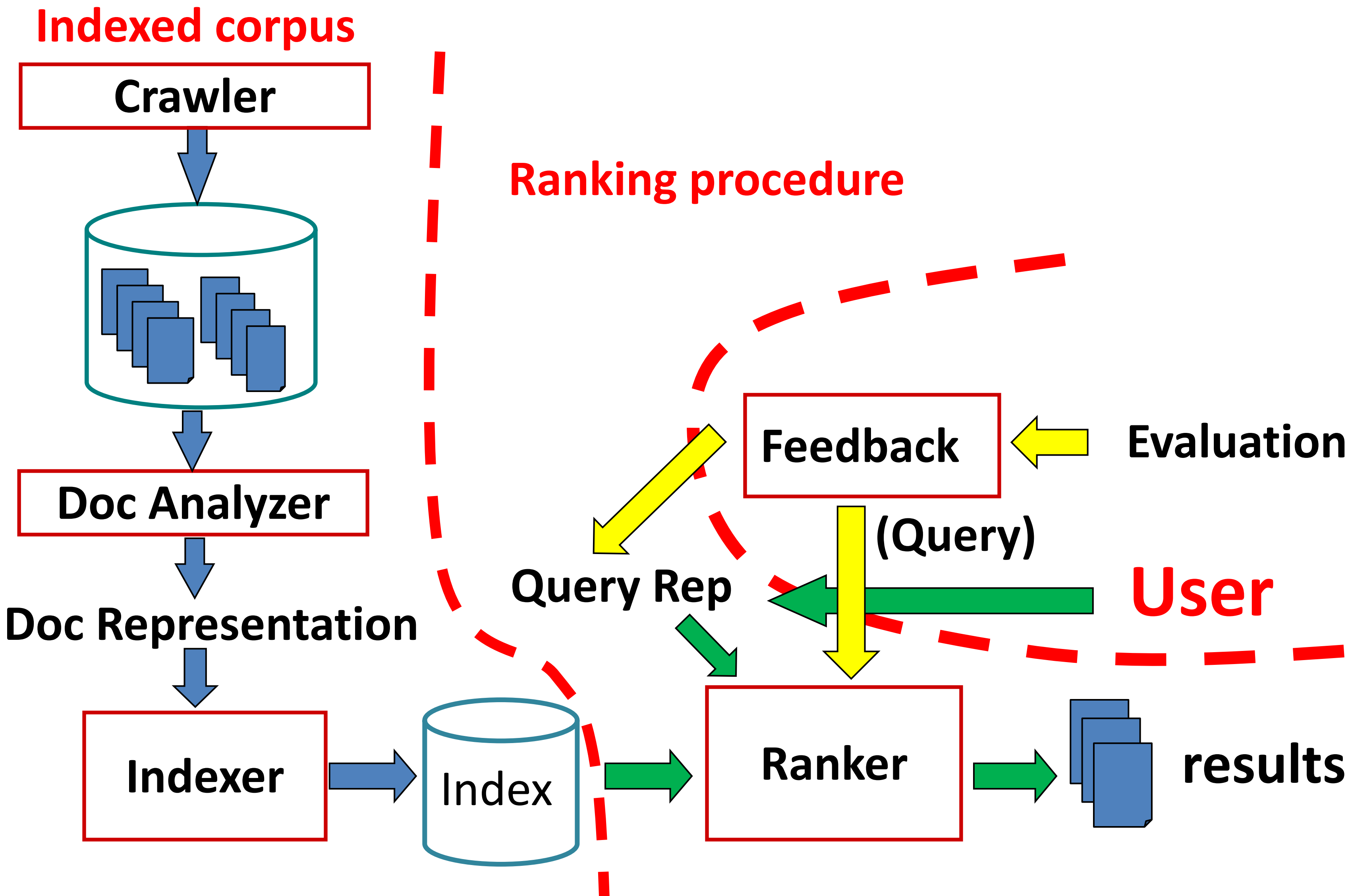
Inverted Index

Hongning Wang
CS@UVa

Abstraction of search engine architecture



What we have now

- Documents have been
  - Crawled from Web
  - Tokenized/normalized
  - Represented as Bag-of-Words
- Let's do search!
  - Query: "information retrieval"

	information	retrieval	retrieved	is	helpful	for	you	everyone
Doc1	1	1	0	1	1	1	0	1
Doc2	1	0	1	1	1	1	1	0

Complexity analysis

- Space complexity analysis
  - $O(D * V)$
    - D is total number of documents and V is vocabulary size
  - Zipf's law: each document only has about 10% of vocabulary observed in it
    - 90% of space is wasted!
  - Space efficiency can be greatly improved by only storing the occurred words

Solution: linked list for each document

Complexity analysis

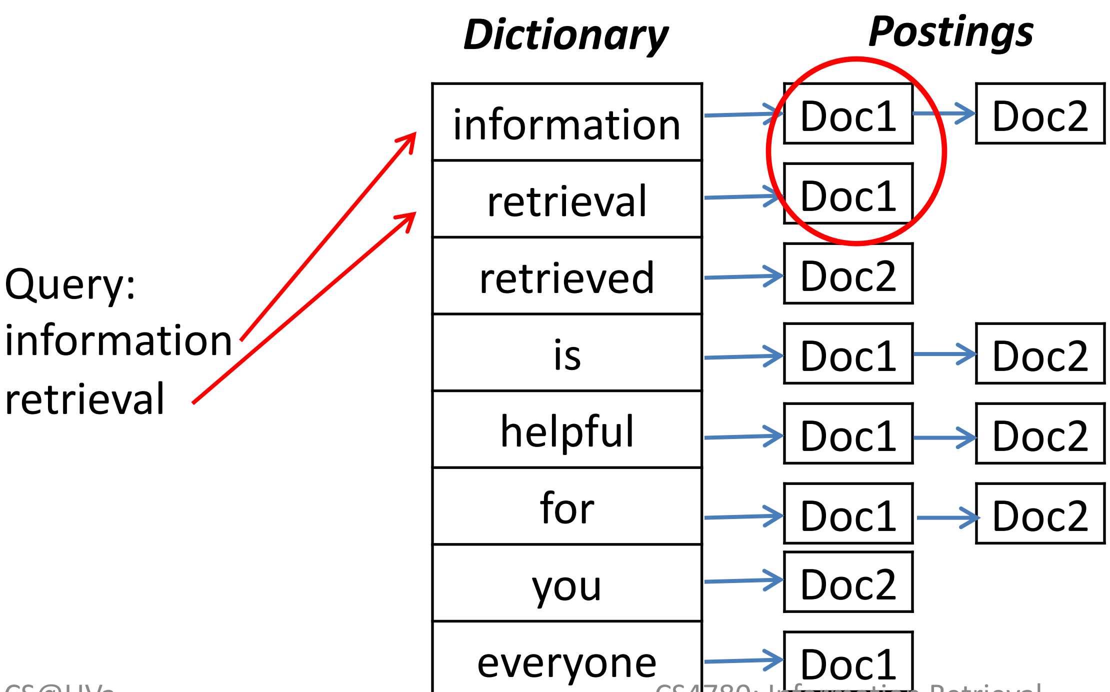
- Time complexity analysis
  - $O(|q| * D * |D|)$
    - $|q|$ is the length of query, $|D|$ is the length of a document

```
doclist = []
for (wi in q) {
    for (d in D) {
        for (wj in d) {
            if (wi == wj) {
                doclist += [d];
                break;
            }
        }
    }
}
return doclist;
```

Bottleneck, since most of them won't match!

Solution: inverted index

- Build a look-up table for each word in vocabulary
  - From word to documents!



Time complexity:

- $O(|q| * |L|)$, $|L|$ is the average length of posting list
- By Zipf's law, $|L| \ll D$

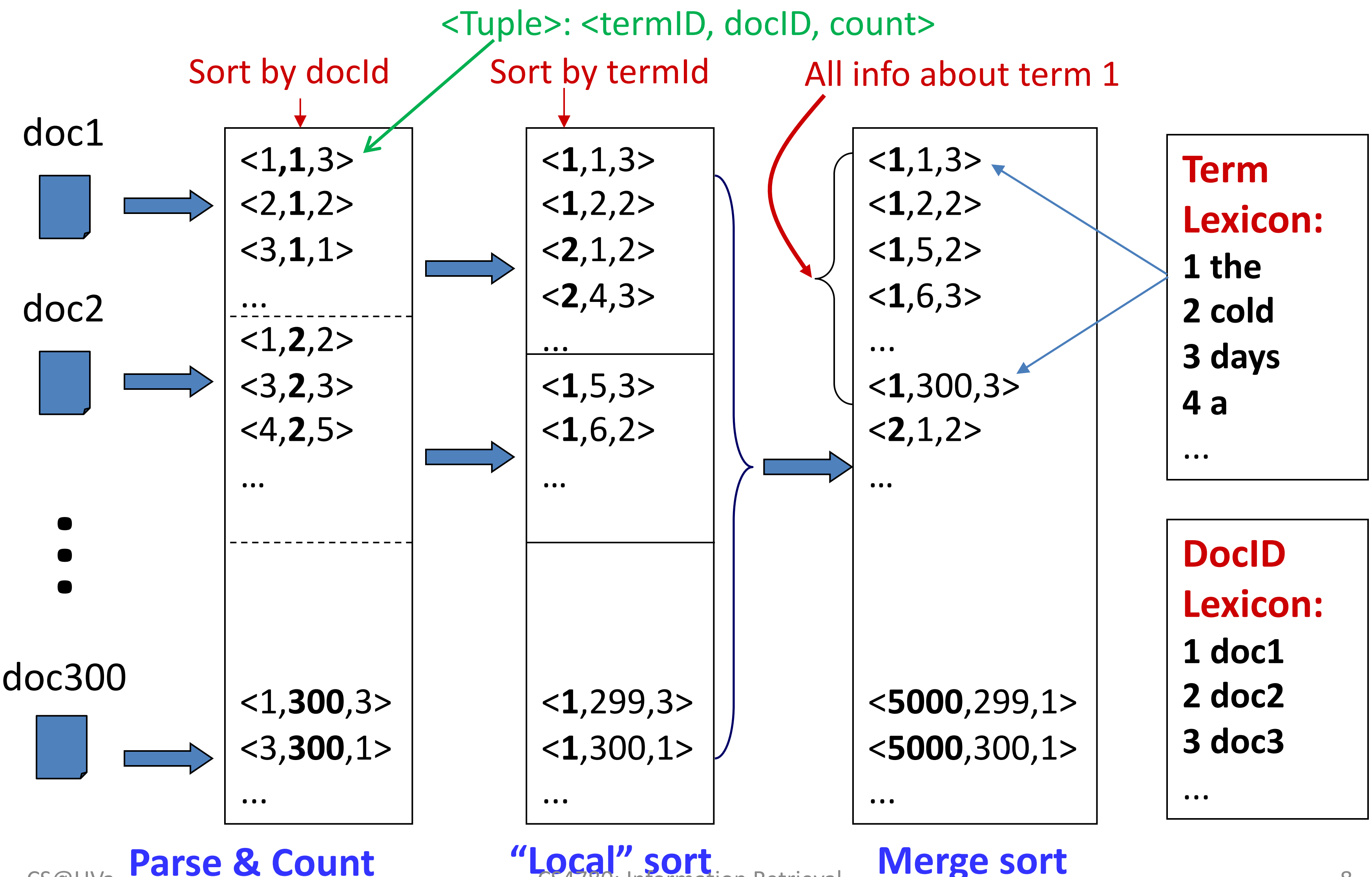
Structures for inverted index

- Dictionary: modest size
  - Needs fast random access
  - Stay in memory
    - Hash table, B-tree, trie, …
- Postings: huge
  - Stay on disk
  - Sequential access is expected
  - Contain docID, term freq, term position, …
  - Compression is needed

"Key data structure underlying modern IR"

- Christopher D. Manning

Sorting-based inverted index construction

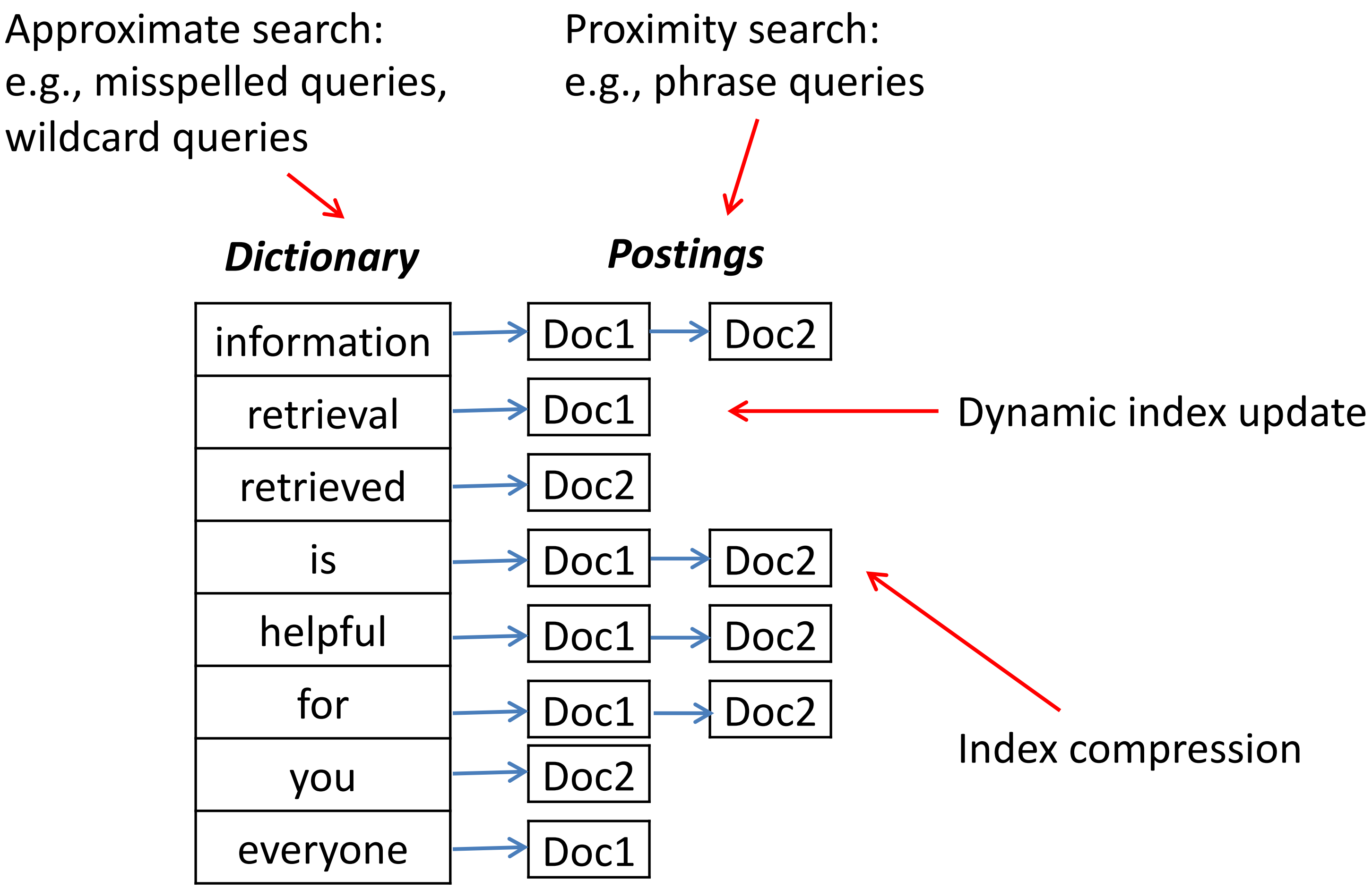


Sorting-based inverted index

- Challenges
  - Document size exceeds memory limit
- Key steps
  - Local sort: sort by termID
    - For later global merge sort
  - Global merge sort
    - Preserve docID order: for later posting list join

Can index large corpus with a single machine! Also suitable for MapReduce!

A close look at inverted index



Dynamic index update

- Periodically rebuild the index
  - Acceptable if change is small over time and penalty of missing new documents is negligible
- Auxiliary index
  - Keep index for new documents in memory
  - Merge to index when size exceeds threshold
    - Increase I/O operation
    - Solution: multiple auxiliary indices on disk, logarithmically merging

Index compression

- Benefits
  - Save storage space
  - Increase cache efficiency
  - Improve disk-memory transfer rate
- Target
  - Postings file

Basics in coding theory

- Expected code length
  - $E[L] = \sum_i p(x_i) \times l_i$

Event	P(X)	Code
a	0.75	0
b	0.10	10
c	0.10	111
d	0.05	110

$E[L] = 1.4$

Index compression

- Observation of posting files
  - Instead of storing docID in posting, we store gap between docIDs, since they are ordered
  - Zipf's law again:
    - The more frequent a word is, the smaller the gaps are
    - The less frequent a word is, the shorter the posting list is
  - Heavily biased distribution gives us great opportunity of compression!

Information theory: entropy measures compression difficulty.

Index compression

- Solution
  - Fewer bits to encode small (high frequency) integers
  - Variable-length coding
    - Unary: $x \geq 1$ is coded as x-1 bits of 1 followed by 0, e.g., 3=> 110; 5=>11110
    - γ-code: x=> unary code for $1+\lfloor \log x \rfloor$ followed by uniform code for $x-2^{\lfloor \log x \rfloor}$ in $\lfloor \log x \rfloor$ bits, e.g., 3=>101, 5=>11001
    - δ-code: same as γ-code ,but replace the unary prefix with γ-code. E.g., 3=>1001, 5=>10101

Index compression

- Example

Table 1: Index and dictionary compression for Reuters-RCV1. (Manning et al. Introduction to Information Retrieval)

Data structure	**Size (MB)**
Text collection	960.0
dictionary	11.2
Postings, uncompressed	400.0
Postings γ-coded	101.0

Compression rate: (101+11.2)/960 = 11.7%

Search within in inverted index

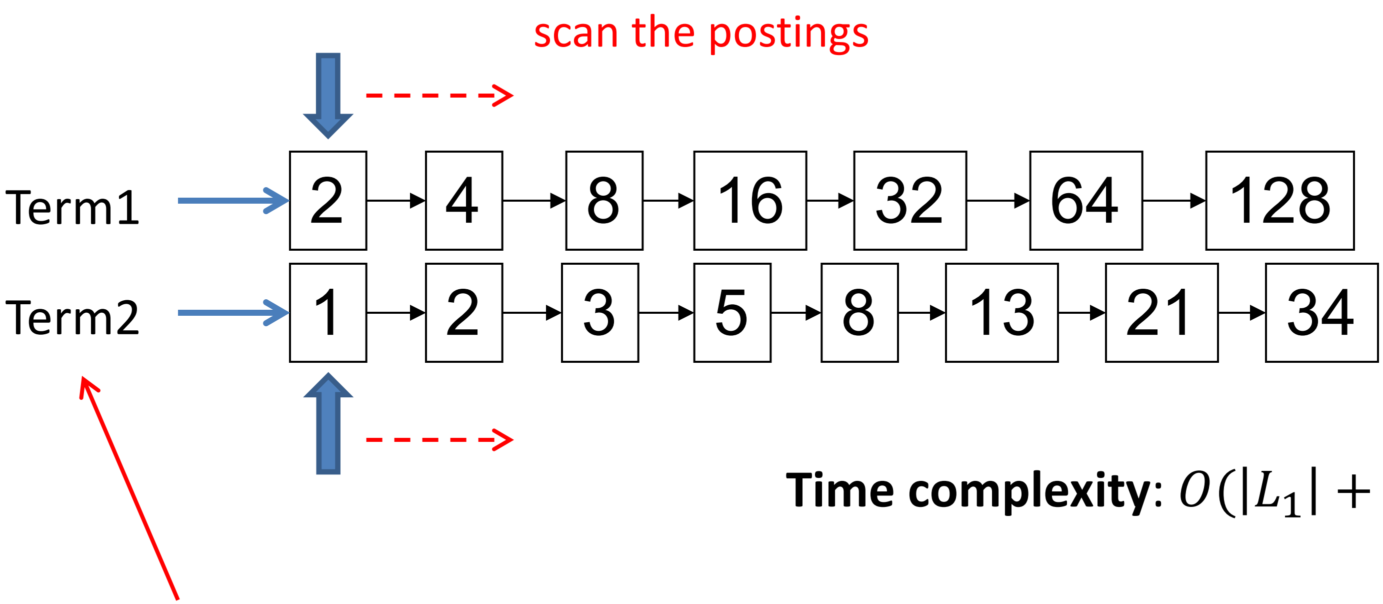
- Query processing
  - Parse query syntax
    - E.g., Barack AND Obama, orange OR apple
  - Perform the same processing procedures as on documents to the input query
    - Tokenization->normalization->stemming->stopwords removal

Search within inverted index

- Procedures
  - Lookup query terms in the dictionary
  - Retrieve the posting lists
  - Operation
    - AND: intersect the posting lists
    - OR: union the posting list
    - NOT: diff the posting list

Search within inverted index

- Example: AND operation



Time complexity: $O(|L_1| + |L_2|)$

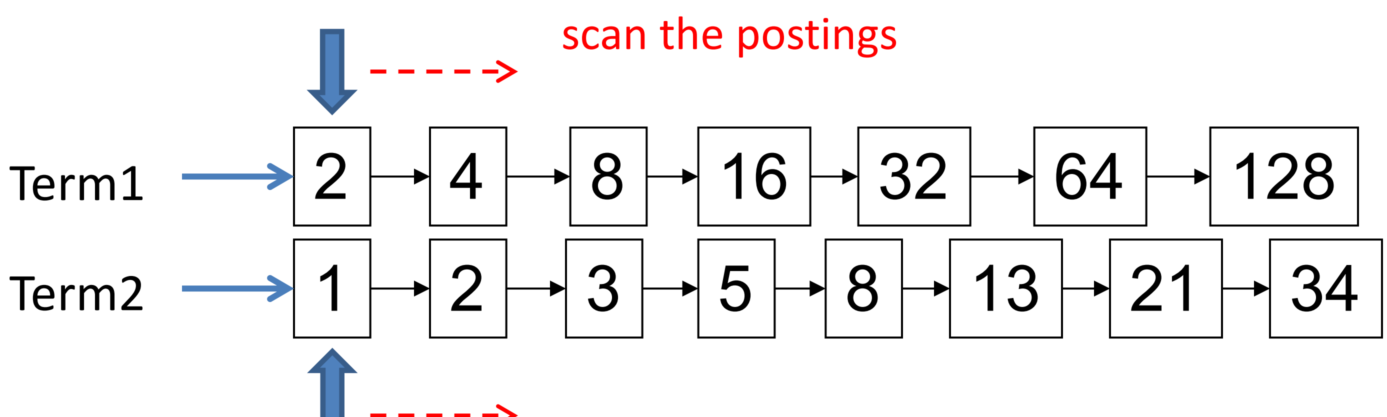
Trick for speed-up: when performing multi-way join, starts from lowest frequency term to highest frequency ones

Phrase query

- “computer science”
  - “He uses his computer to study science problems” is not a match!
  - We need the phase to be exactly matched in documents
  - N-grams generally does not work for this
    - Large dictionary size, how to break long phrase into N-grams?
  - We need term positions in documents
    - We can store them in the inverted index

Phrase query

- Generalized postings matching
  - Equality condition check with requirement of position pattern between two query terms
    - e.g., T2.pos-T1.pos = 1 (T1 must be immediately before T2 in any matched document)
  - Proximity query: |T2.pos-T1.pos| ≤ k



More and more things are put into index

- Document structure
  - Title, abstract, body, bullets, anchor
- Entity annotation
  - Being part of a person’s name, location’s name

Spelling correction

- Tolerate the misspelled queries
  - “barck obama” -> “barack obama”
- Principles
  - Of various alternative correct spellings of a misspelled query, choose the ***nearest*** one
  - Of various alternative correct spellings of a misspelled query, choose the ***most common*** one

Spelling correction

- Proximity between query terms
  - Edit distance
    - Minimum number of edit operations required to transform one string to another
    - Insert, delete, replace
    - Tricks for speed-up
      - Fix prefix length (error does not happen on the first letter)
      - Build character-level inverted index, e.g., for length 3 characters
      - Consider the layout of a keyboard
        - » E.g., ‘u’ is more likely to be typed as ‘y’ instead of ‘z’

Spelling correction

- Proximity between query terms
  - Query context
    - “flew form IAD” -> “flew from IAD”
  - Solution
    - Enumerate alternatives for all the query terms
    - Heuristics must be applied to reduce the search space

Spelling correction

- Proximity between query terms
  - Phonetic similarity
    - “herman” -> “Hermann”
  - Solution
    - Phonetic hashing – similar-sounding terms hash to the same value

What you should know

- Inverted index for modern information retrieval
  - Sorting-based index construction
  - Index compression
- Search in inverted index
  - Phrase query
  - Query spelling correction

Today’s reading

- Introduction to Information Retrieval
  - Chapter 2: The term vocabulary and postings lists
    - Section 2.3, Faster postings list intersection via skip pointers
    - Section 2.4, Positional postings and phrase queries
  - Chapter 4: Index construction
  - Chapter 5: Index compression
    - Section 5.2, Dictionary compression
    - Section 5.3, Postings file compression