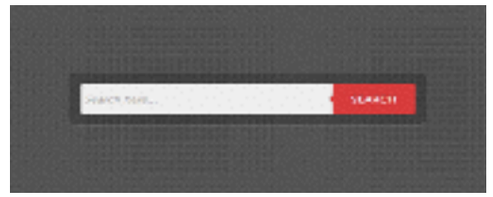
Information Retrieval

Danushka Bollegala

Anatomy of a Search Engine

Query Processing



Document Indexing

Results Ranking

Document Processing

- Format detection
  - Plain text, PDF, PPT, …
- Text extraction
  - Convert to plain text
- Language detection
- Tokenisation
- Indexing

Language Detection

- How to detect the language of a document?
  - Check for specific characters such as kanji characters for Chinese, and Hiragana/Katakana for Japanese, Hangul for Korean etc.
- Not always perfect
  - Some documents might contain multiple languages
    - e.g. Japanese web site that teaches English
  - Character-based statistical approaches are used
- If we get the language wrong, we will use a wrong tokeniser

Tokenisation

- If we do not tokenise properly, then we cannot search for those terms!
- Tokens vs. Words
  - Token refers to a single unit of text that we can search for
    - *the burger i ate was an awesome burger*
    - tokens = the, burger, i, ate, was, an, awesome, burger
- Tokens might not necessarily be words in English
- Repeating tokens are counted separately
  - note the two *burger* tokens in the previous example

Is tokenisation simple?

- Simple! Just split at spaces to get tokens
  - s.split()
- What about the following?
  - Dr. D. T. Bollegala
    - Should we consider *Dr*, . , *D*, ., *T*, ., *Bollegala* or *Dr.*, *D.*, *T.*, *Bollegala*?
- Japanese and Chinese languages do not use spaces at all!
  - 私は学校に行きました.
  - Tokens: 私/は/学校/に/行きました/.

Indexing

- Search engines create an *inverted index* from tokens to documents for efficient retrieval
- Similar to the indexes you find at the end of a text book
  - *Support Vector Machines p. 13, p. 56, p. 124*
- Index is a large table between tokens and unique document ids

Inverted Index

D_1 = I went to school
D_2 = The school was closed
D_3 = Tomorrow is a holiday

I	D_1
went	D_1
school	D_1 D_2
tomorrow	D_3
holiday	D_3
closed	D_2
to	D_1

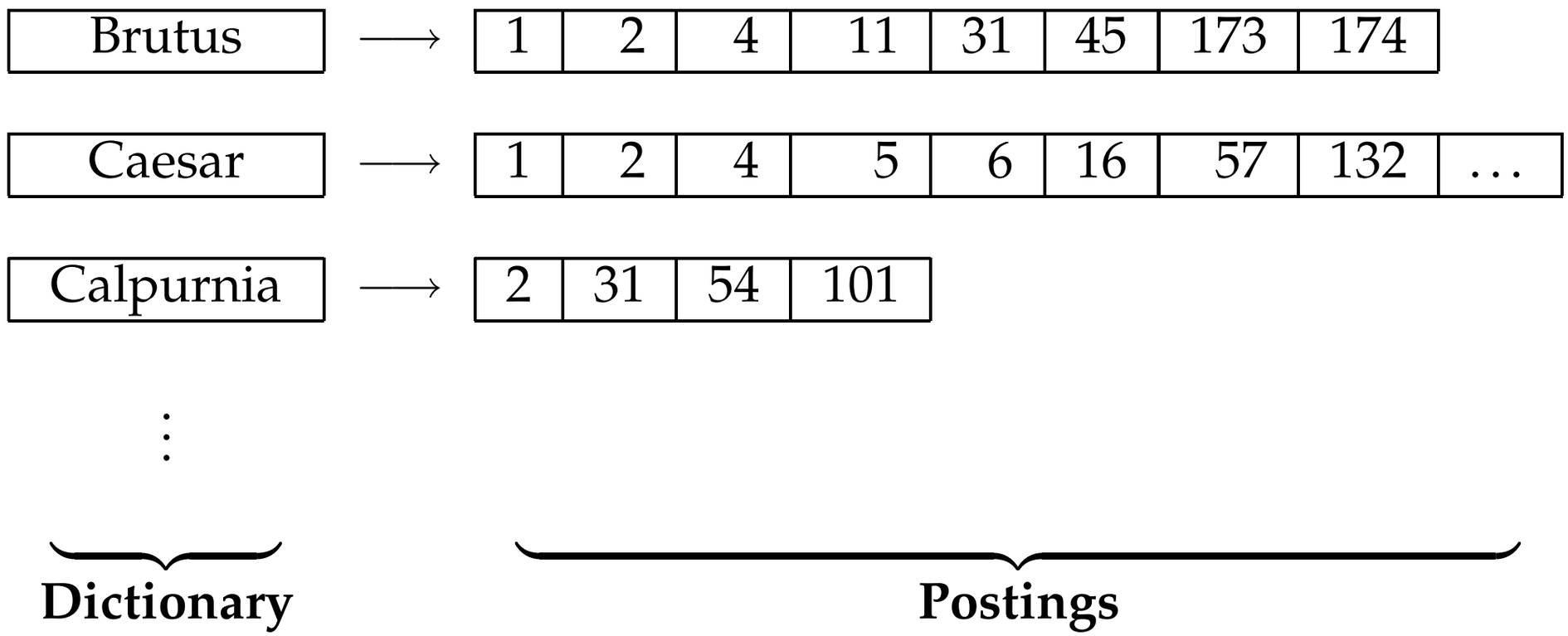
The list of document IDs for a particular token is called a *posting list*

Notes

- We can assign integer IDs to documents so that we can sort the posting lists.
- By doing so we can quickly answer AND queries.
- We can assign integer ids to terms (tokens) as well. This will save space.
- We only need to store one entry for a word in a document. (multiple entries can be ignored, *bag-of-words* model).
- We need to store the length of a posting list as meta data.
- AND queries
  - Start with the shorter posting list.
  - Find the document ids in the shorter list in the longer list.

Example

Query = *Brutus* AND *Calpurnia*
Results = 2, 31



Start with the posting list for Calpurnia (shorter list), and check for 2, 31. No point checking any further than 173. We will not find 54 and 101 in Brutus.

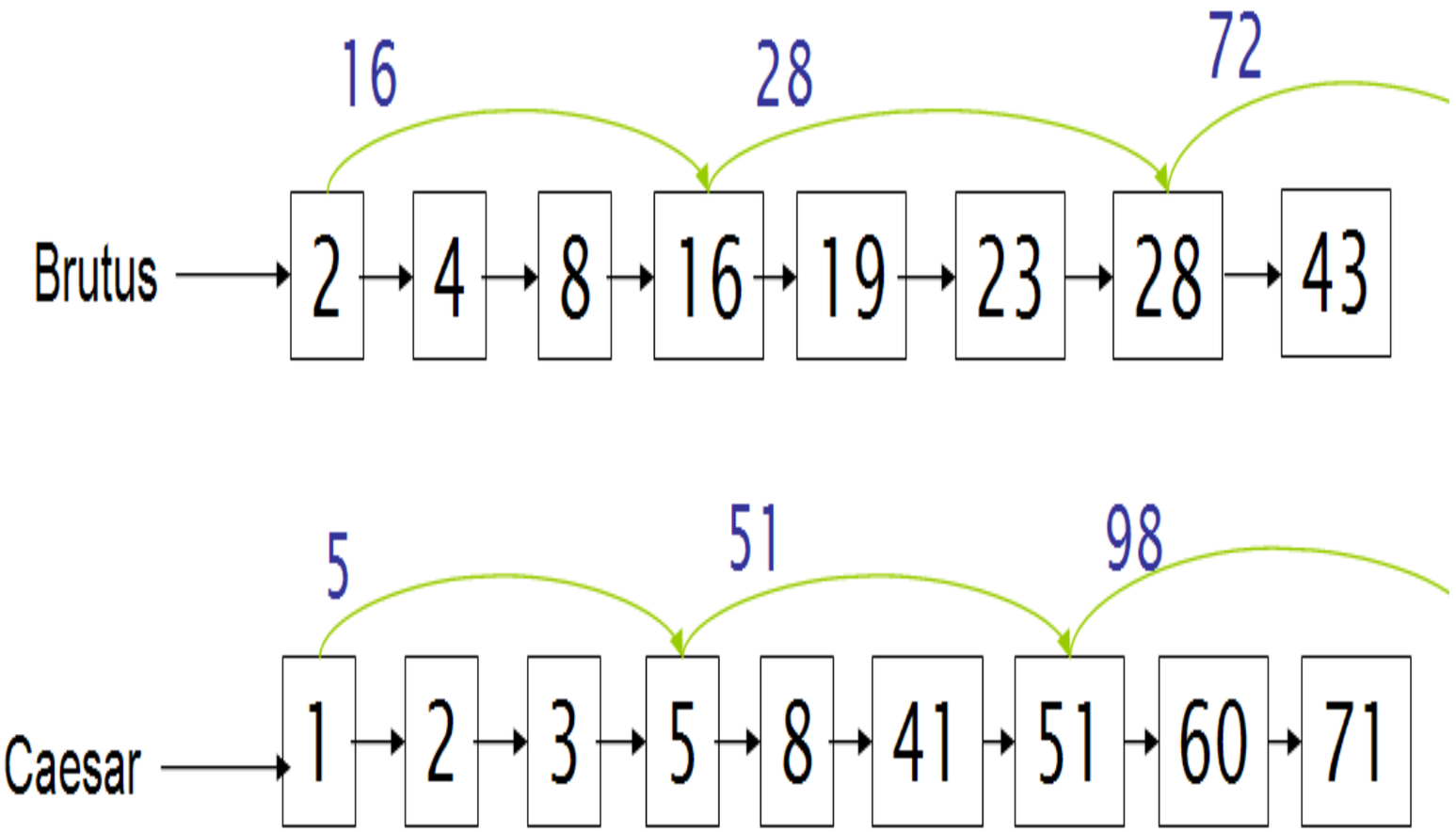
AND query processing

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7     else if docID(p1) < docID(p2)
 8             then p1 ← next(p1)
 9             else p2 ← next(p2)
10  return answer
```

Skip pointers

- The previous version of answering AND queries is inefficient.
  - O(n+m) if the length of the two posting lists are n and m.
- We can add *skip pointers* to speed up the search.
- If the value to be searched for is larger than the skip pointer then we can directly skip over all the values under the skip pointer.
- How to find skip points? (square root heuristic)
- Trade-off:
  - number of skips vs. skip range

Example



After matching up to 8, and when we want to match 41 next, we note that at 16 we have a skip of 28. This means that we will not observe 41 during this skip range. We can skip over 19 and 23, and resume the search process from 28. We cannot skip to 72 because 51 is in between 28 and 72.

Disk access vs. Memory access

- Disk seek time = 5ms
- disk read per 1b = $2x10^{-8}$s
- memory read per 1b = 10^{-9}s
- Reading from memory is faster compared to disk.
- We need to read in blocks (ca. 64kb) when we read from the disk because of the seek overhead.
- Main memory is limited (10~100GB) vs. disk space (1~10TB)

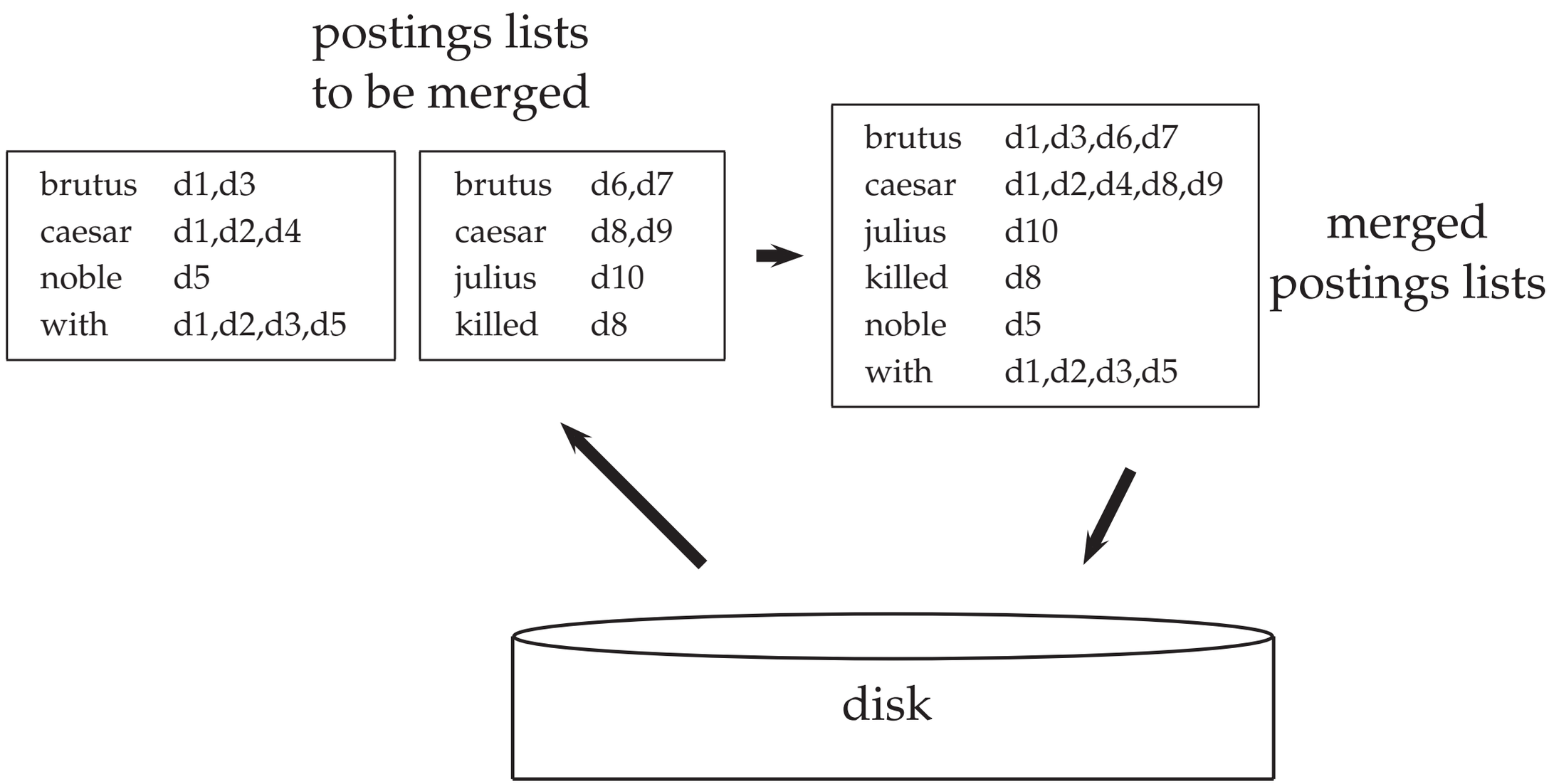
Blocked Sort-Based Indexing

- BSBI (Blocked Sort-based Indexing)
  - Segment the document collection into parts of equal size
  - Sort the termID-docID pairs for each block in memory
  - Store intermediate sorted results on disk
  - Merge all intermediate results into the final index

BSBI

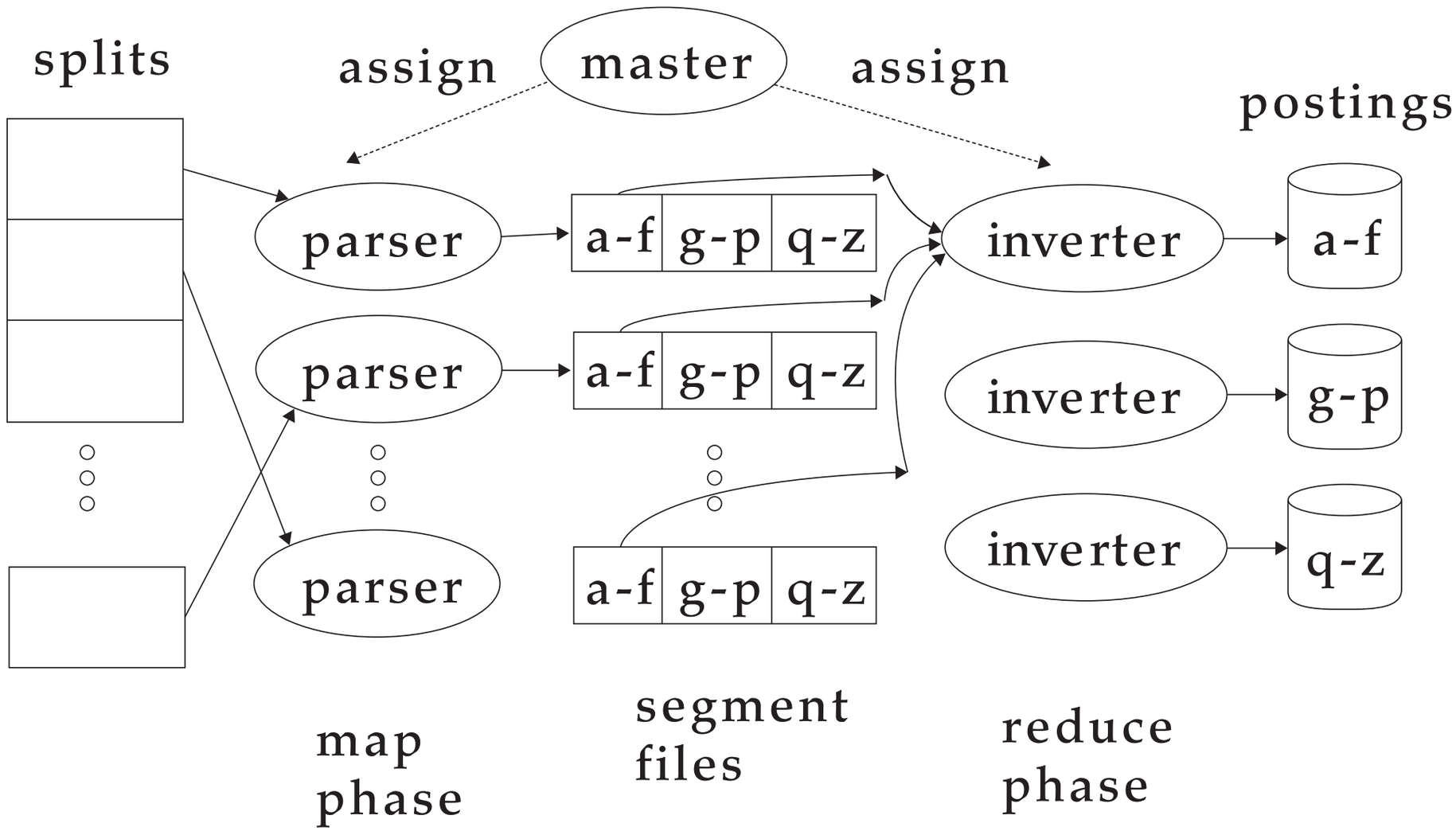
BSBIndexConstruction()
1 $n \leftarrow 0$
2 **while** (all documents have not been processed)
3 **do** $n \leftarrow n + 1$
4 $block \leftarrow$ ParseNextBlock()
5 BSBI-Invert($block$)
6 WriteBlockToDisk($block, f_n$)
7 MergeBlocks($f_1, \ldots, f_n; f_{\text{merged}}$)

Example: BSBI



Web scale indexing

- Most large documents collections (e.g. Web) result in indexes that cannot be stored in a single machine
- Distributed indexing methods are required
- Methods based on MapReduce are used.



Ranking

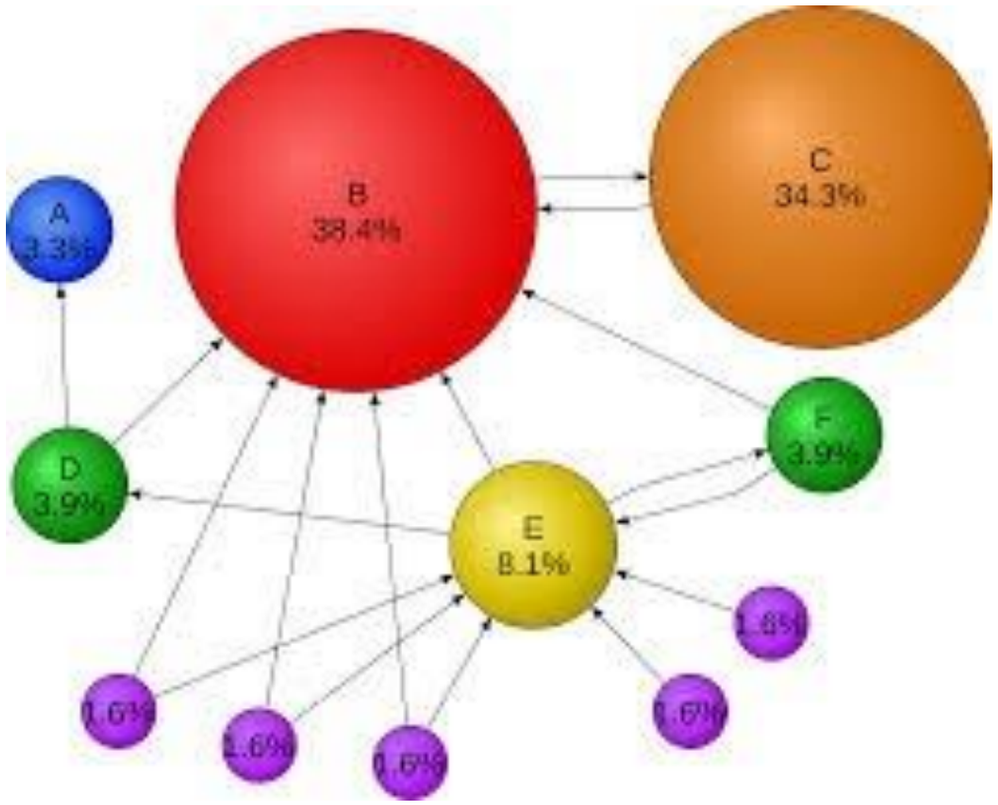
- Often there are hundreds of documents that contain a particular query
- We must rank the search results according to their *relevance* to a query
- There are numerous factors that need to be considered when computing f(**q**,**d**), the relevance of a document **d** to a query **q**.

Static Ranking

- The ranking of a document independent of the query
  - PageRank is a famous example of a static ranking algorithm

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

Discussed later in our Graph Mining lecture



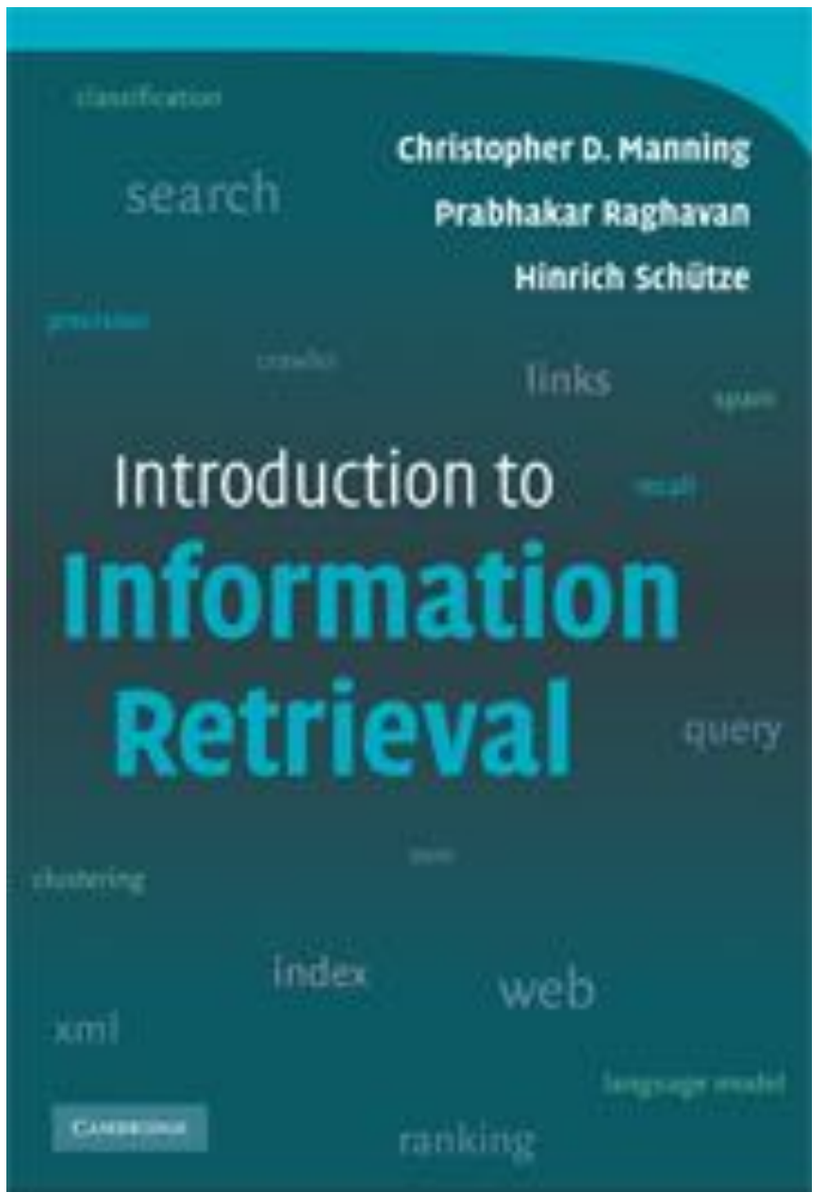
Dynamic Ranking

- The rank of a document depends on the query
- Features
  - term frequency
  - PageRank
  - novelty of the document
  - position of the query within the document
    - title, anchor text, links, etc.
- f(q,d) is computed as the linear combination of numerous features that indicate relevance
  - $f(q,\mathbf{d}) = \mathbf{w}^T \boldsymbol{\psi}(q,\mathbf{d})$

How to learn the relevance weights?

- Clickthrough data are collected by the search engines
- Assume that we entered a query *q* and obtained a ranked list of documents d_1, d_2, d_3.
- If we skip d_1 and clicked on d_2, then the search engine creates a training instance indicating that $f(q,d_2) > f(q,d_1)$
- Billions of people are using search engines and clicking on documents, giving a large and cheap training dataset to learn the relevance function f.

References



PDF available here.
http://www-nlp.stanford.edu/IR-book/