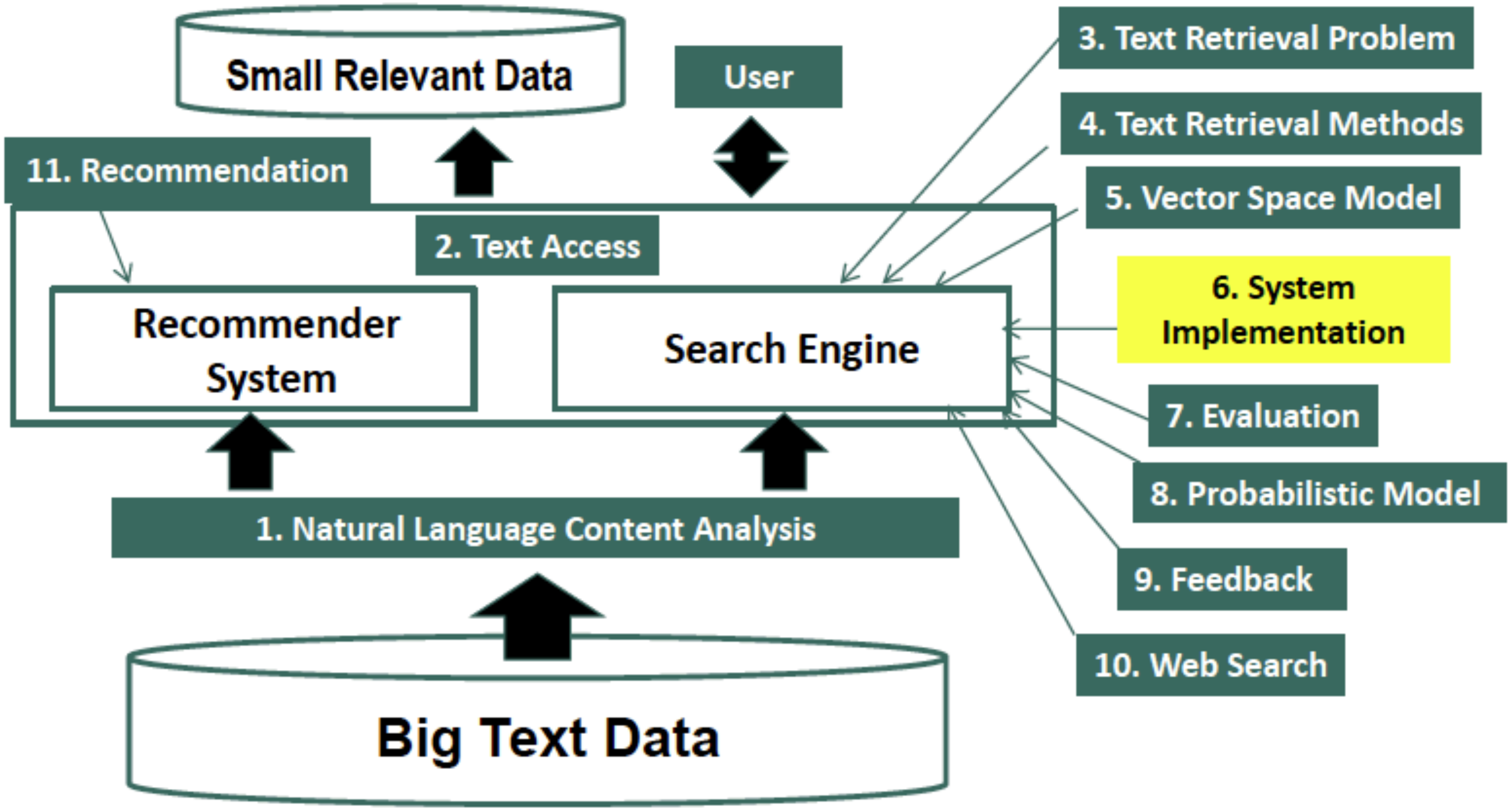
Information Retrieval

System Implementation: Inverted Index Construction

Dr. Iqra Safder

Implementation of Text Retrieval Systems



Inverted Index Example

doc 1

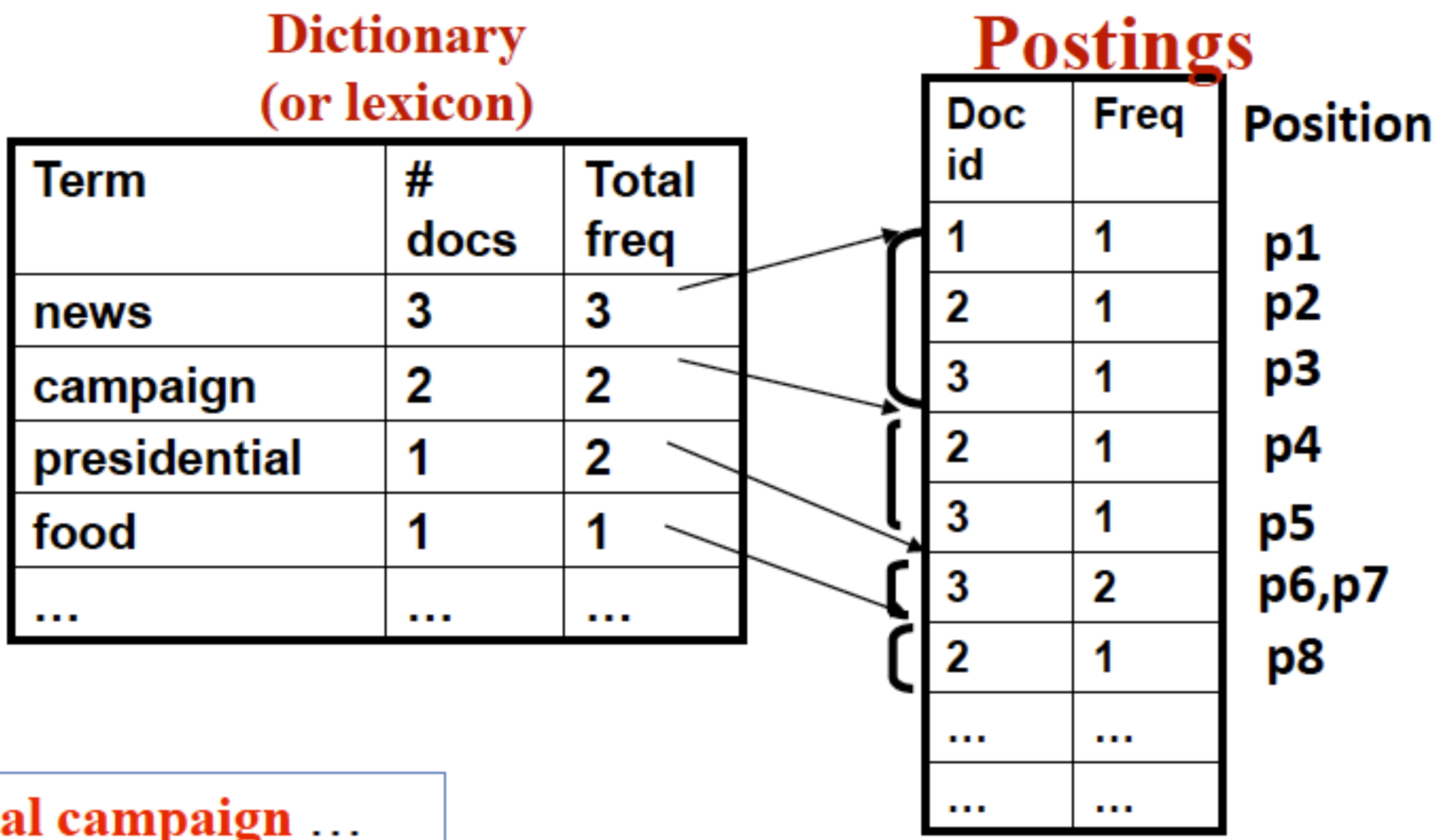
… **news about**

doc 2

… **news about** organic food **campaign**…

doc 3

… **news** of **presidential campaign** …
… **presidential** candidate …



Dictionary (or lexicon)

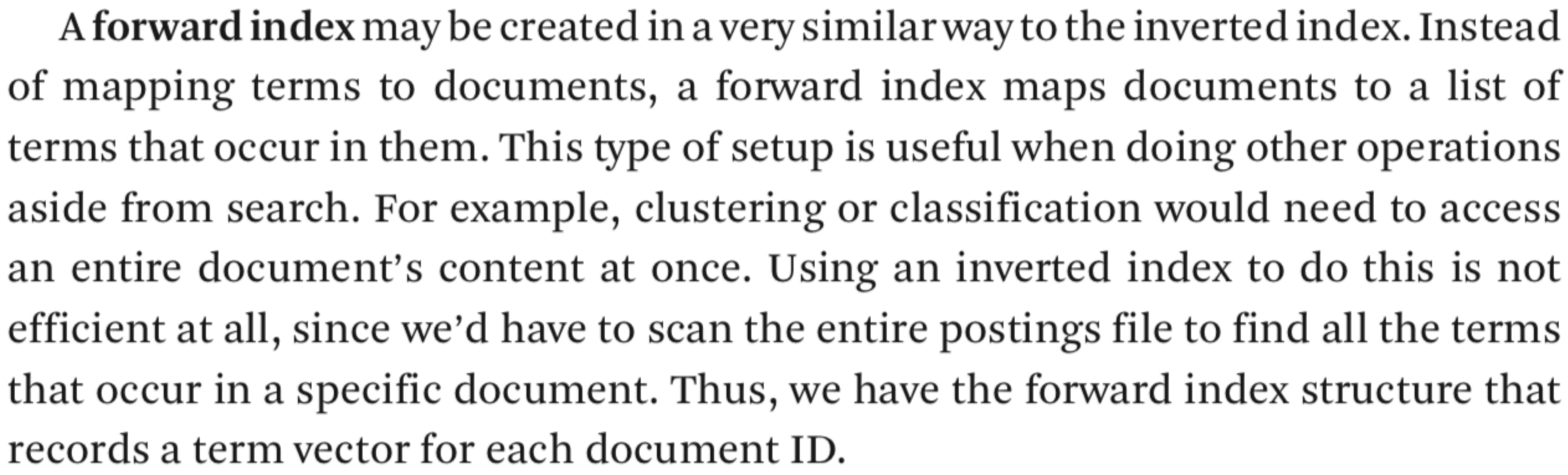
Term	# docs	Total freq
news	3	3
campaign	2	2
presidential	1	2
food	1	1
…	…	…

Postings

Doc id	Freq	Position
1	1	p1
2	1	p2
3	1	p3
2	1	p4
3	1	p5
3	2	p6,p7
2	1	p8
…	…	
…	…	

IDF is calculated using # docs

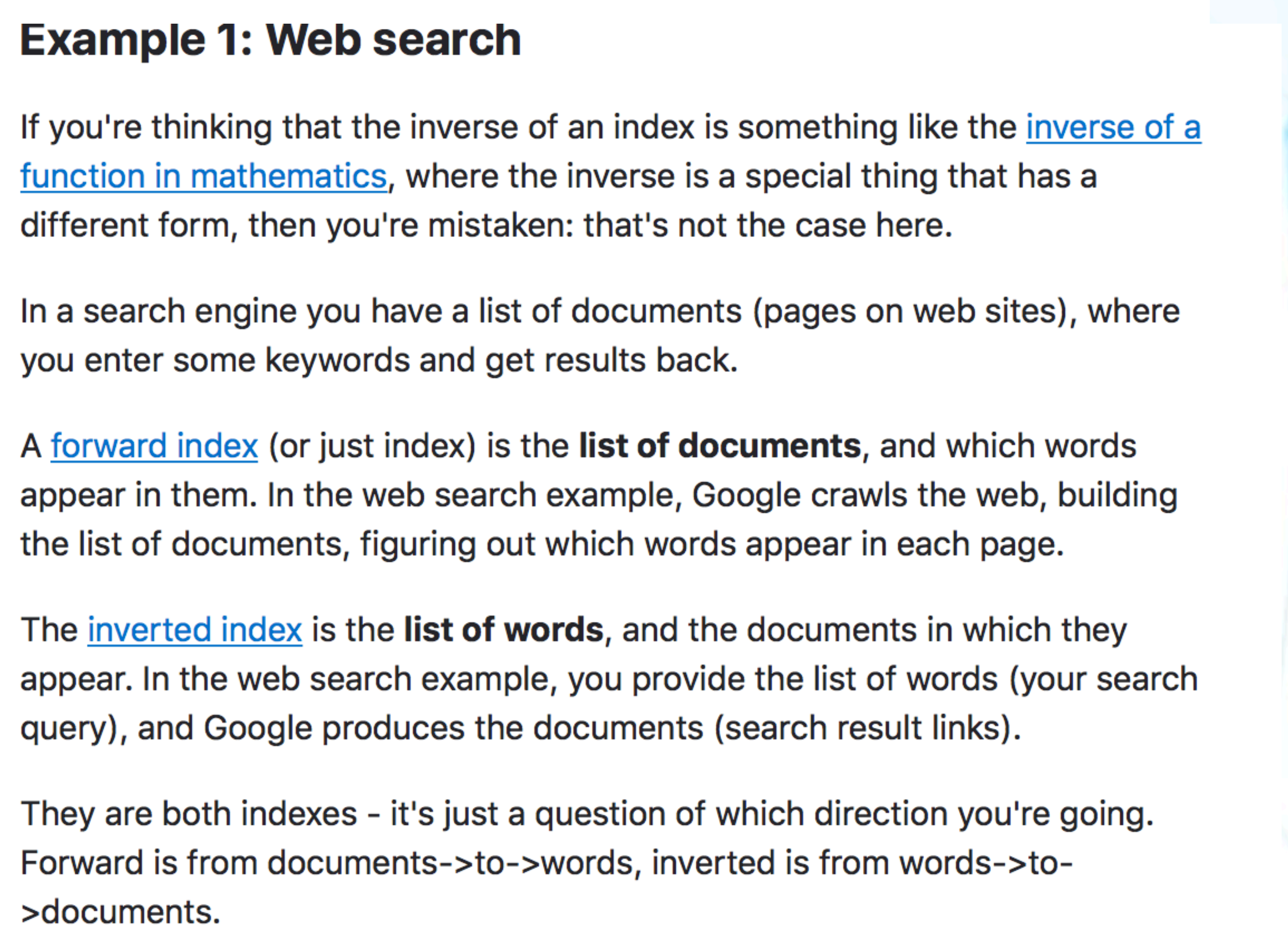
Forward Index



A **forward index** may be created in a very similar way to the inverted index. Instead of mapping terms to documents, a forward index maps documents to a list of terms that occur in them. This type of setup is useful when doing other operations aside from search. For example, clustering or classification would need to access an entire document's content at once. Using an inverted index to do this is not efficient at all, since we'd have to scan the entire postings file to find all the terms that occur in a specific document. Thus, we have the forward index structure that records a term vector for each document ID.

Indexing is the process of creating these data structures based on a set of tokenized documents

Forward Index VS Inverted Index



Example 1: Web search

If you're thinking that the inverse of an index is something like the inverse of a function in mathematics, where the inverse is a special thing that has a different form, then you're mistaken: that's not the case here.

In a search engine you have a list of documents (pages on web sites), where you enter some keywords and get results back.

A forward index (or just index) is the **list of documents**, and which words appear in them. In the web search example, Google crawls the web, building the list of documents, figuring out which words appear in each page.

The inverted index is the **list of words**, and the documents in which they appear. In the web search example, you provide the list of words (your search query), and Google produces the documents (search result links).

They are both indexes - it's just a question of which direction you're going. Forward is from documents->to->words, inverted is from words->to->documents.

http://stackoverflow.com

Document	Keywords
doc1	hello, sky, morning
doc2	tea, coffee, hi
doc3	greetings, sky

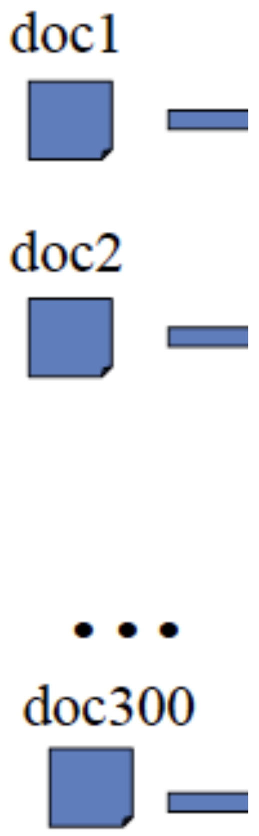
Word	Documents
hello	doc1
sky	doc1, doc3
coffee	doc2
hi	doc2
greetings	doc3

Constructing Inverted Index

- The main difficulty is to build a huge index with limited memory
- Memory-based methods: not usable for large collections
- Sort-based methods:
  - Step 1: Collect local (termID, docID, freq) tuples
  - Step 2: Sort local tuples (to make "runs")
  - Step 3: Pair-wise merge runs
  - Step 4: Output inverted file

Indexing is the process of creating these data structures based on a set of tokenized documents

Sort-based Inversion



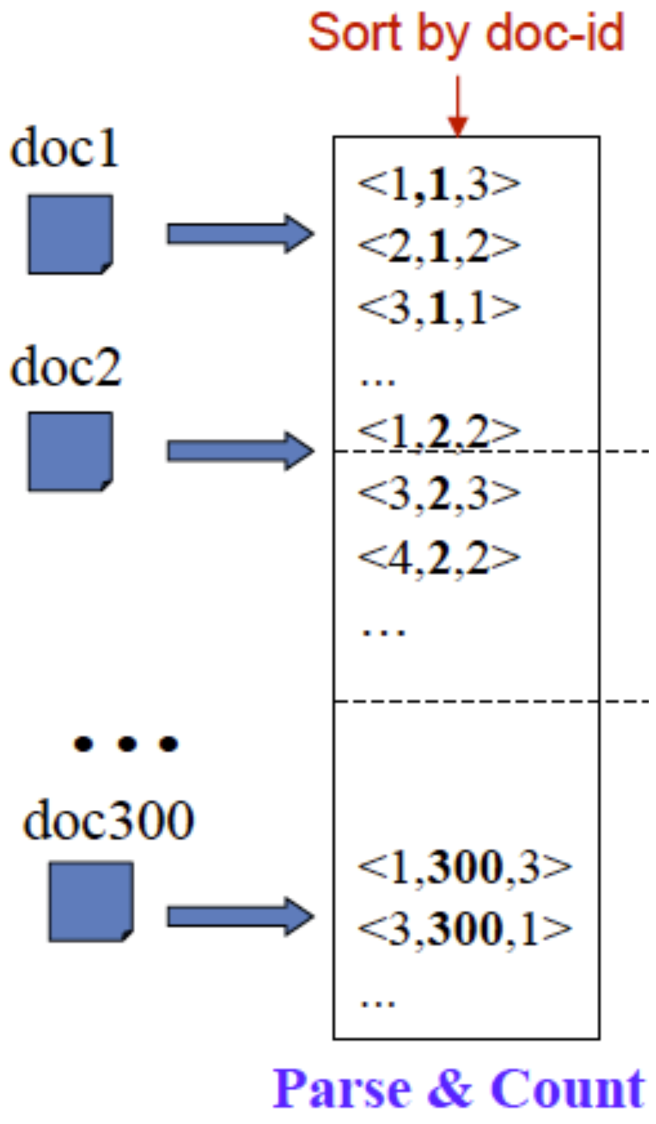
Term Lexicon:

the 1
campaign 2
news 3
a 4
…

DocID Lexicon:

doc1 1
doc2 2
doc3 3
…

Sort-based Inversion



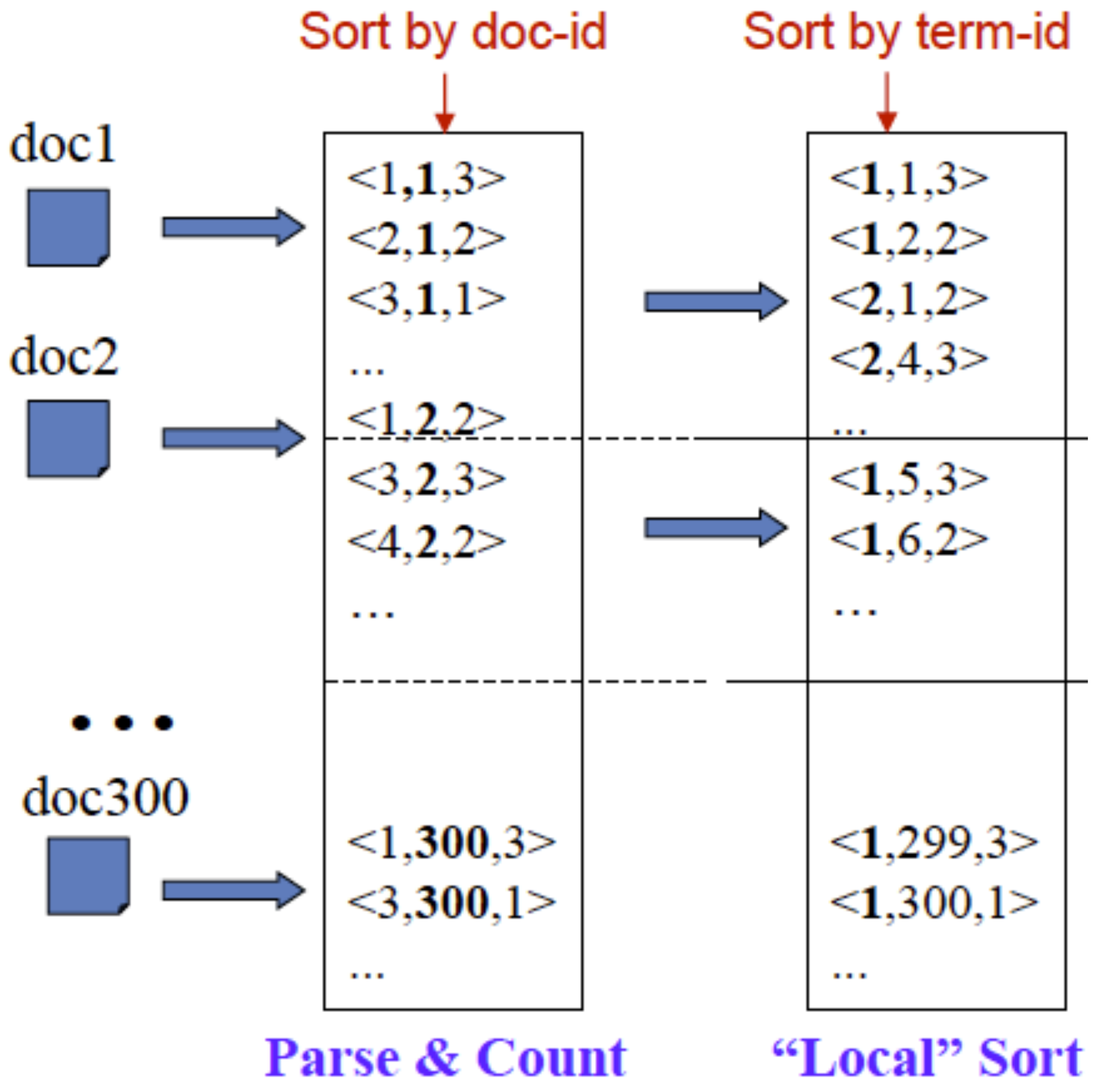
Term Lexicon:

the 1
campaign 2
news 3
a 4
…

DocID Lexicon:

doc1 1
doc2 2
doc3 3
…

Sort-based Inversion



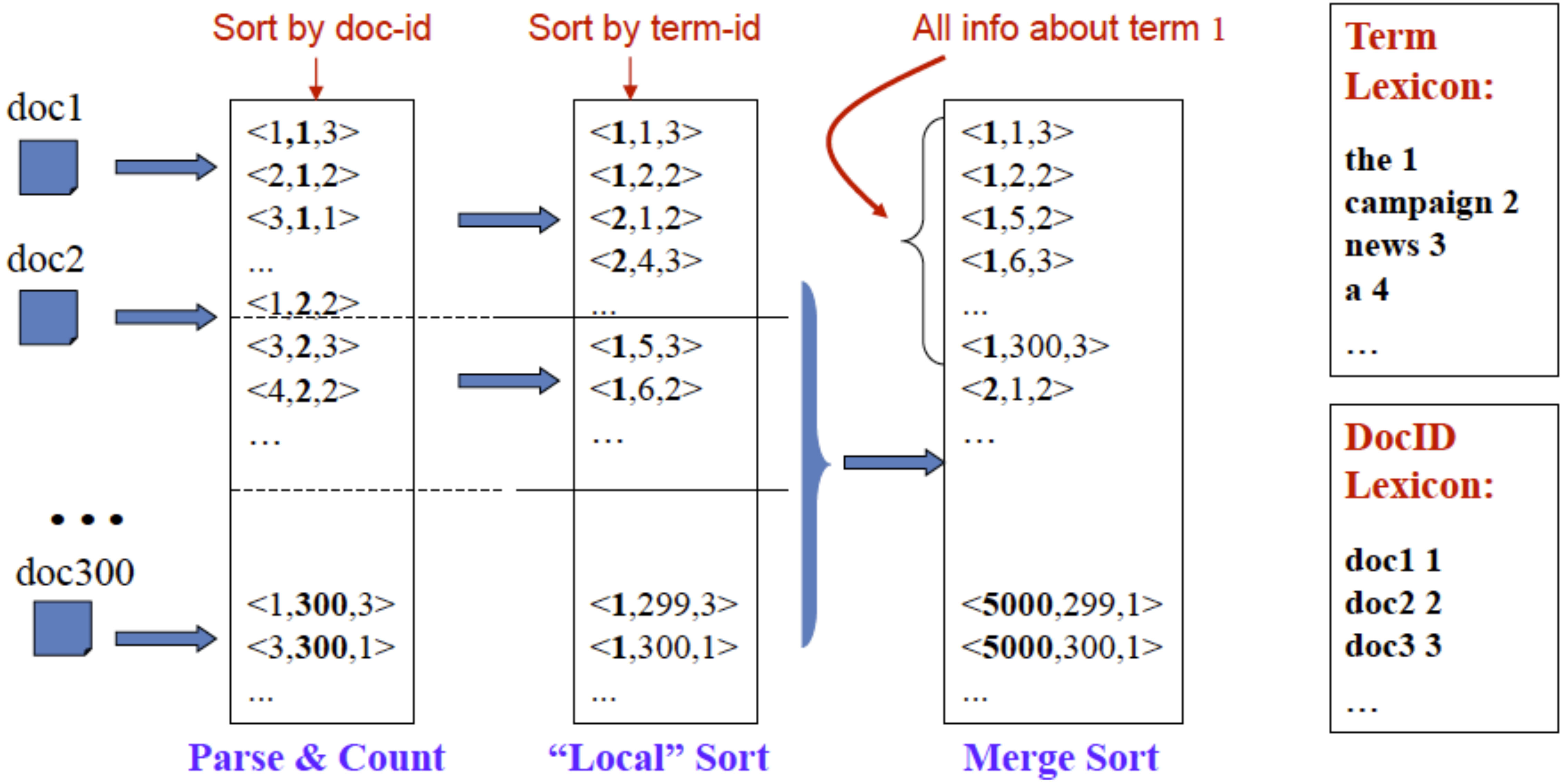
Term Lexicon:

the	1
campaign	2
news	3
a	4
…	

DocID Lexicon:

doc1	1
doc2	2
doc3	3
…	

Sort-based Inversion



Sort by doc-id
Sort by term-id
All info about term 1
doc1
doc2
• • •
doc300
<1,1,3>
<2,1,2>
<3,1,1>
...
<1,2,2>
<3,2,3>
<4,2,2>
…
<1,300,3>
<3,300,1>
...
<1,1,3>
<1,2,2>
<2,1,2>
<2,4,3>
...
<1,5,3>
<1,6,2>
…
<1,299,3>
<1,300,1>
...
<1,1,3>
<1,2,2>
<1,5,2>
<1,6,3>
...
<1,300,3>
<2,1,2>
…
<5000,299,1>
<5000,300,1>
...
Term Lexicon:
the 1
campaign 2
news 3
a 4
…
DocID Lexicon:
doc1 1
doc2 2
doc3 3
…
Parse & Count
"Local" Sort
Merge Sort