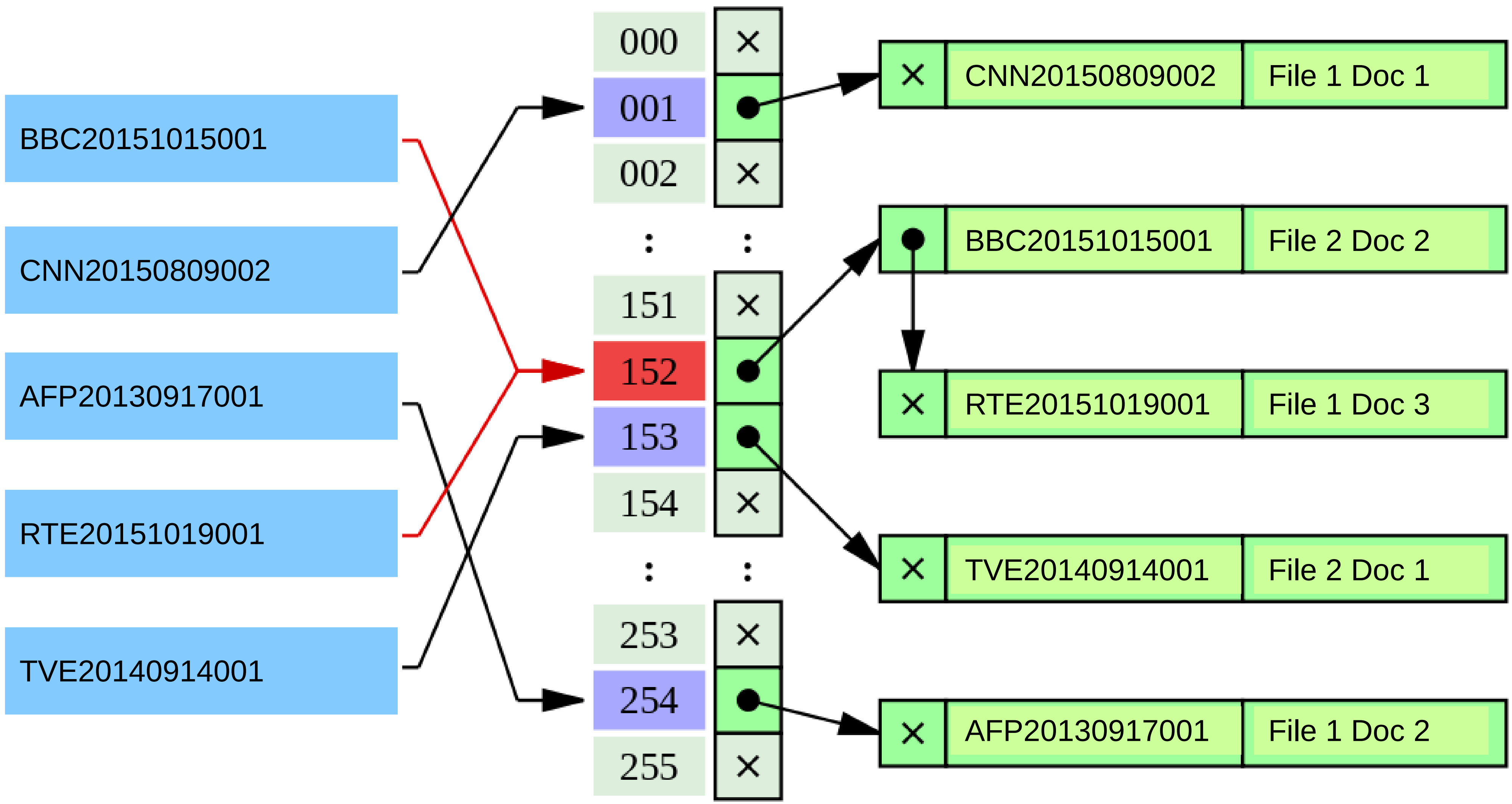
Text Indexing

Class	Algorithmic Methods of Data Mining
Program	M. Sc. Data Science
University	Sapienza University of Rome
Semester	Fall 2015
Lecturer	Carlos Castillo http://chato.cl/

Sources:

- Gonzalo Navarro: “Indexing and Searching.” Chapter 9 in *Modern Information Retrieval, 2nd Edition*. 2011. [slides]
- Christopher D. Manning, Prabhakar Raghavan & Hinrich Schütze: “Introduction to Information Retrieval”. 2008 [link]

Index by document ID



Document identifiers

Physical locations

Search by keywords

- Given a set of keywords
- Find documents containing *all* keywords
- Each keyword may be in millions of documents
- Hundreds of queries per second

Indexing the documents helps

- For an Information Retrieval system that uses an index, efficiency means:
  - Indexing time: Time needed to build the index
  - Indexing space: Space used during the generation of the index
  - Index storage: Space required to store the index
  - Query latency: Time interval between the arrival of the query and the generation of the answer
  - Query throughput: Average number of queries processed per second
- We assume a static or semi-static collection

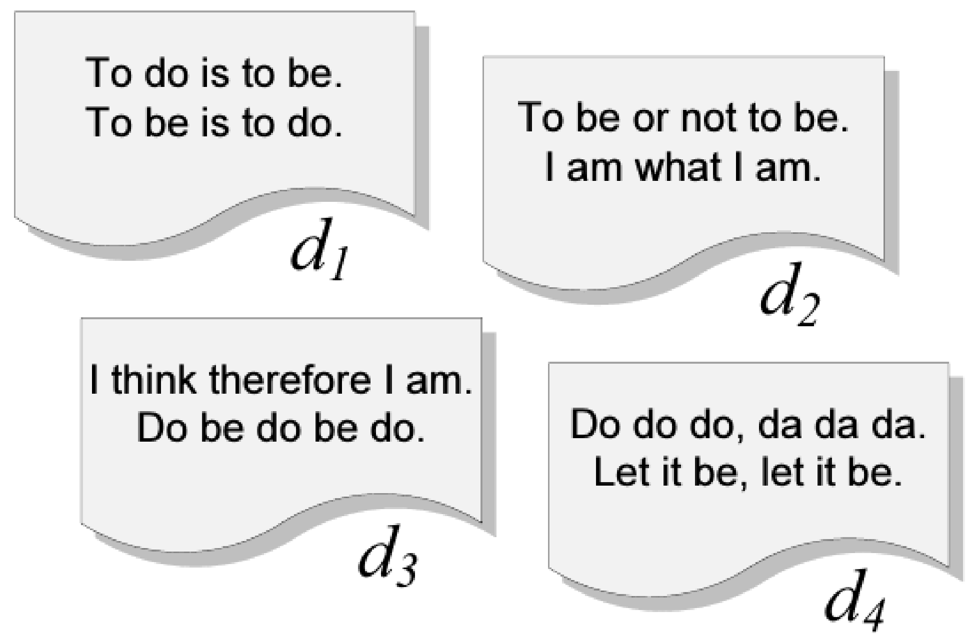
Inverted index

- The index we have so far:
  - Given a document ID
  - Return the words in the document
- The index we want:
  - Given a word
  - Return the IDs of documents containing that word

Term-document matrix

Vocabulary	n_i	d_1	d_2	d_3	d_4
to	2	4	2	-	-
do	3	2	-	3	3
is	1	2	-	-	-
be	4	2	2	2	2
or	1	-	1	-	-
not	1	-	1	-	-
I	2	-	2	2	-
am	2	-	2	1	-
what	1	-	1	-	-
think	1	-	-	1	-
therefore	1	-	-	1	-
da	1	-	-	-	3
let	1	-	-	-	2
it	1	-	-	-	2

Doc frequency   Term frequencies



Space inefficient: why?

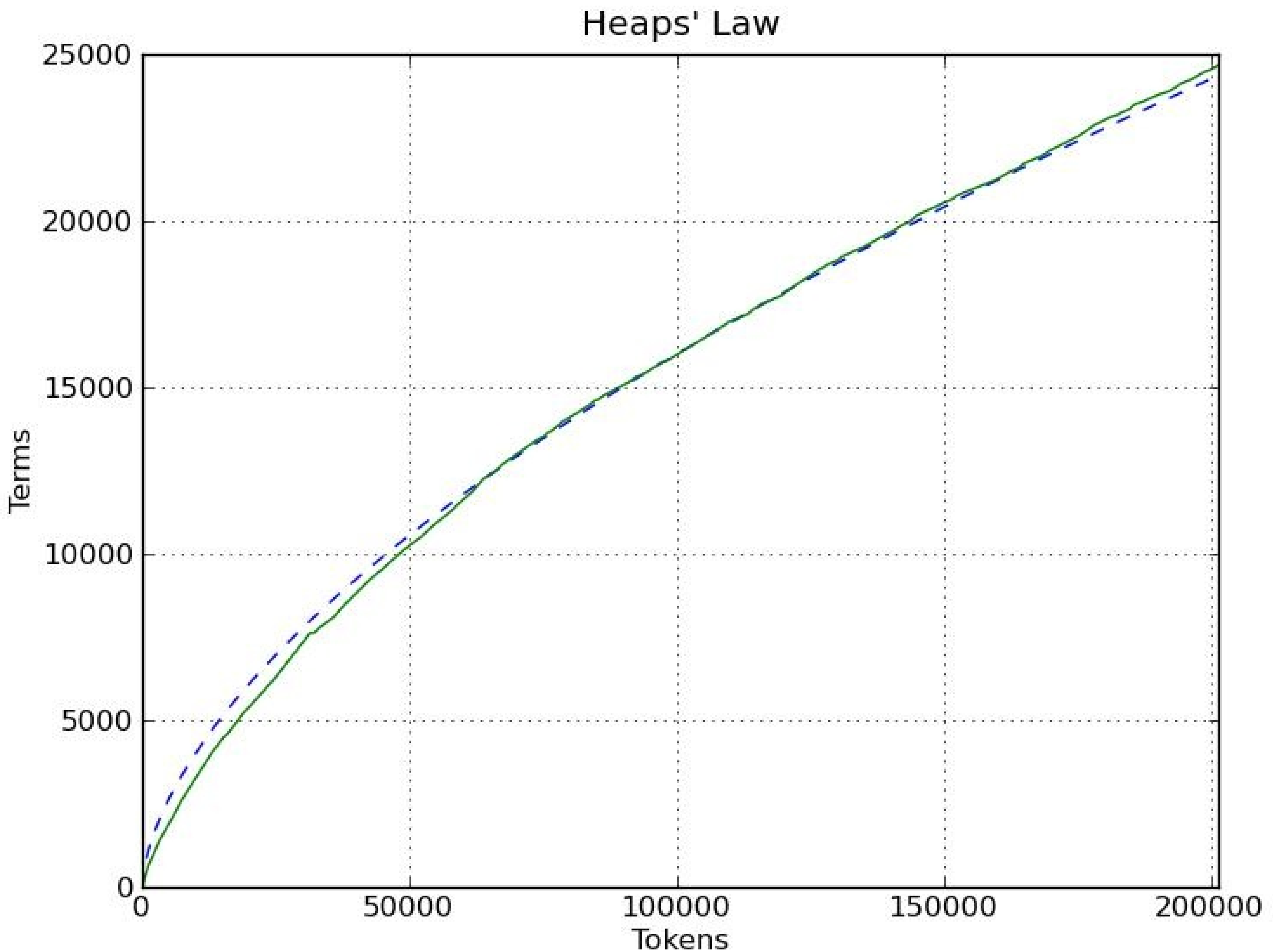
How large is the vocabulary?

$$V(n) \approx K n^{\beta}$$

In English:

$$K \in [10, 100]$$

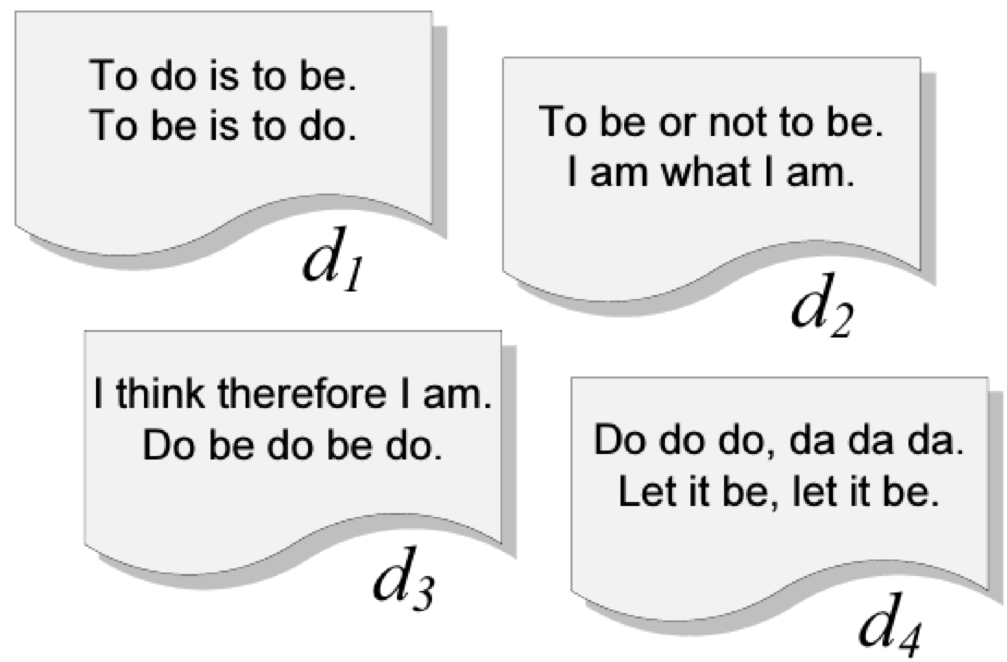
$$\beta \in [0.4, 0.6]$$



Why it is not bounded?

Inverted index

Vocabulary	n_i	Occurrences as inverted lists
to	2	[1,4],[2,2]
do	3	[1,2],[3,3],[4,3]
is	1	[1,2]
be	4	[1,2],[2,2],[3,2],[4,2]
or	1	[2,1]
not	1	[2,1]
I	2	[2,2],[3,2]
am	2	[2,2],[3,1]
what	1	[2,1]
think	1	[3,1]
therefore	1	[3,1]
da	1	[4,3]
let	1	[4,2]
it	1	[4,2]



Inverted index (vocabulary)

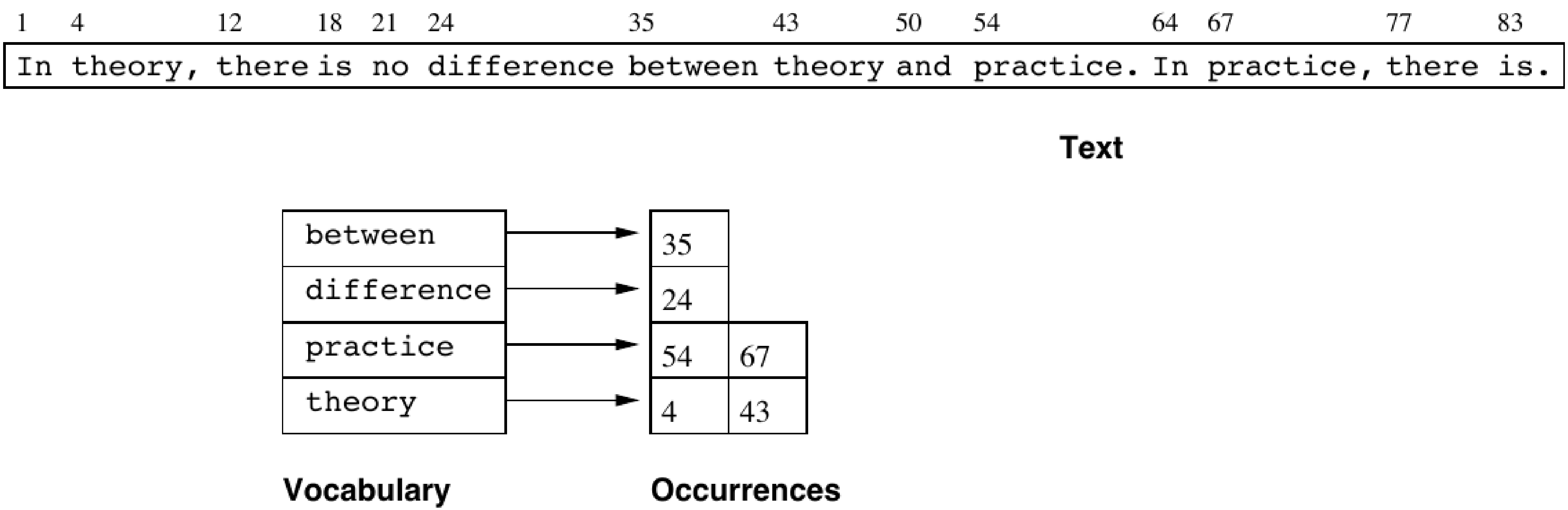
Vocabulary	n_i
to	2
do	3
is	1
be	4
or	1
not	1
I	2
am	2
what	1
think	1
therefore	1
da	1
let	1
it	1

What are the alternatives for storing the vocabulary?

What are the trade-offs involved?

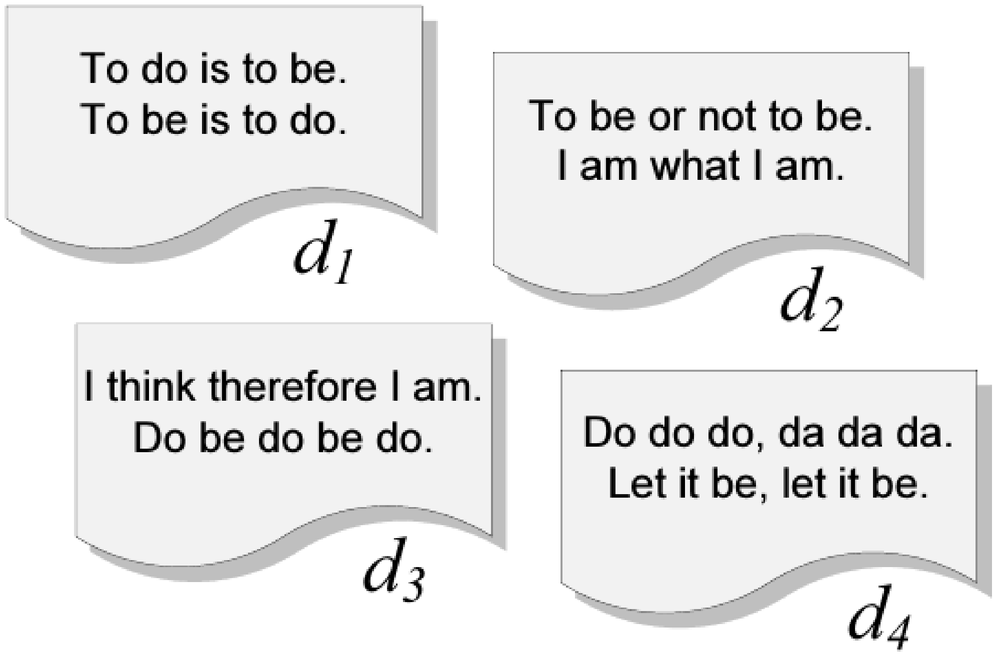
Full inverted index
(single document, character level)

- Allows us to answer phrase and proximity queries, e.g. “theory * practice” or “difference between theory and practice”



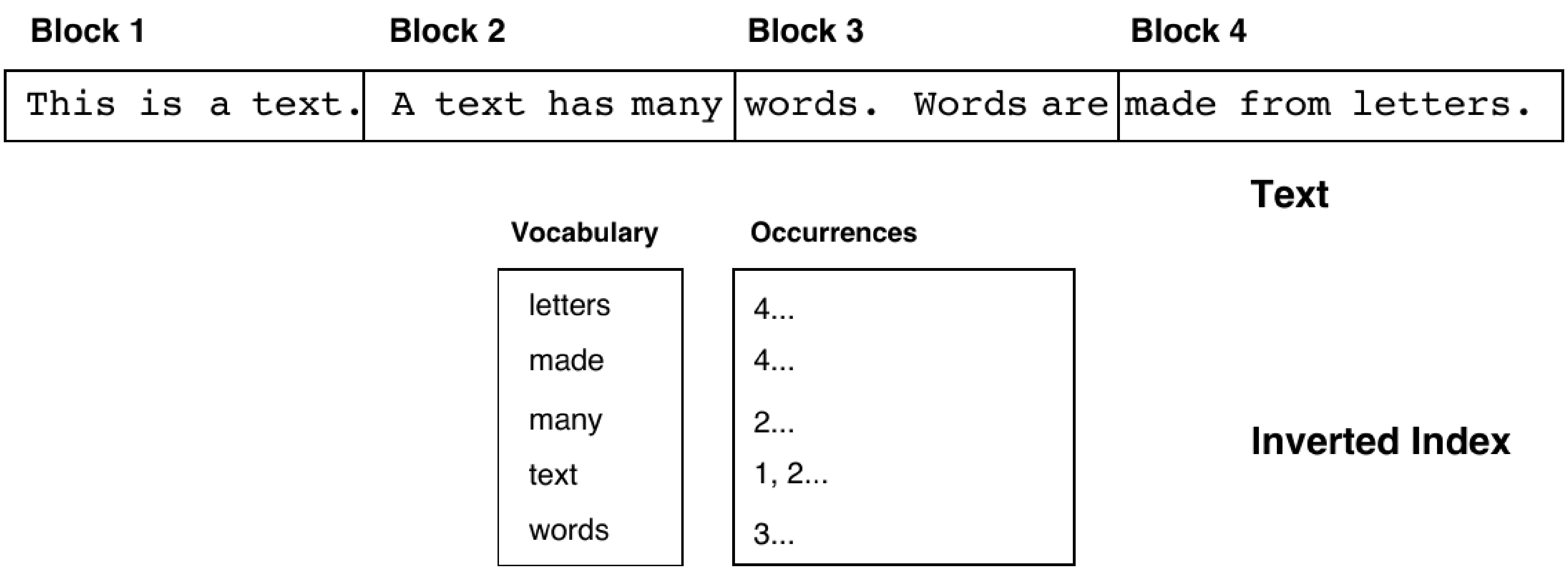
Full inverted index
(multiple documents, word-level)

Vocabulary	n_i	Occurrences as full inverted lists
to	2	[1,4,[1,4,6,9]],[2,2,[1,5]]
do	3	[1,2,[2,10]],[3,3,[6,8,10]],[4,3,[1,2,3]]
is	1	[1,2,[3,8]]
be	4	[1,2,[5,7]],[2,2,[2,6]],[3,2,[7,9]],[4,2,[9,12]]
or	1	[2,1,[3]]
not	1	[2,1,[4]]
I	2	[2,2,[7,10]],[3,2,[1,4]]
am	2	[2,2,[8,11]],[3,1,[5]]
what	1	[2,1,[9]]
think	1	[3,1,[2]]
therefore	1	[3,1,[3]]
da	1	[4,3,[4,5,6]]
let	1	[4,2,[7,10]]
it	1	[4,2,[8,11]]



Space usage of an index

- Vocabulary requires $O(n^\beta), \beta < 1$
- Occurrences require $O(n)$
- Address documents or words?
- Address blocks is an intermediary solution



Phrase search

- How do you do a phrase search with:
  - Addressing document
  - Addressing words
  - Addressing blocks

Estimated sizes of indices

Index granularity	Single document (1 MB)		Small collection (200 MB)		Medium collection (2 GB)	
Addressing words	45%	73%	36%	64%	35%	63%
Addressing documents	19%	26%	18%	32%	26%	47%
Addressing 64K blocks	27%	41%	18%	32%	5%	9%
Addressing 256 blocks	18%	25%	1.7%	2.4%	0.5%	0.7%

Try it

Build an inverted index with word addressing for these documents

Consider "warm" and "warming" as a single term "warm"

Verify: third posting list has 3 docs

d1: "global warming"

d2: "global climate"

d3: "climate change"

d4: "warm climate"

d5: "global village"

http://chato.cl/2015/data_analysis/exercise-answers/text-indexing_exercise_01_answer.txt

Searching time

- Assuming the vocabulary fits on main memory, and *m* terms in the query, this is *O(m)*
- The time is dominated by merging the lists of the words
- Merging is fast if lists are sorted
  - At most *n1 + n2* comparisons where n1 and n2 are the sizes of the posting lists

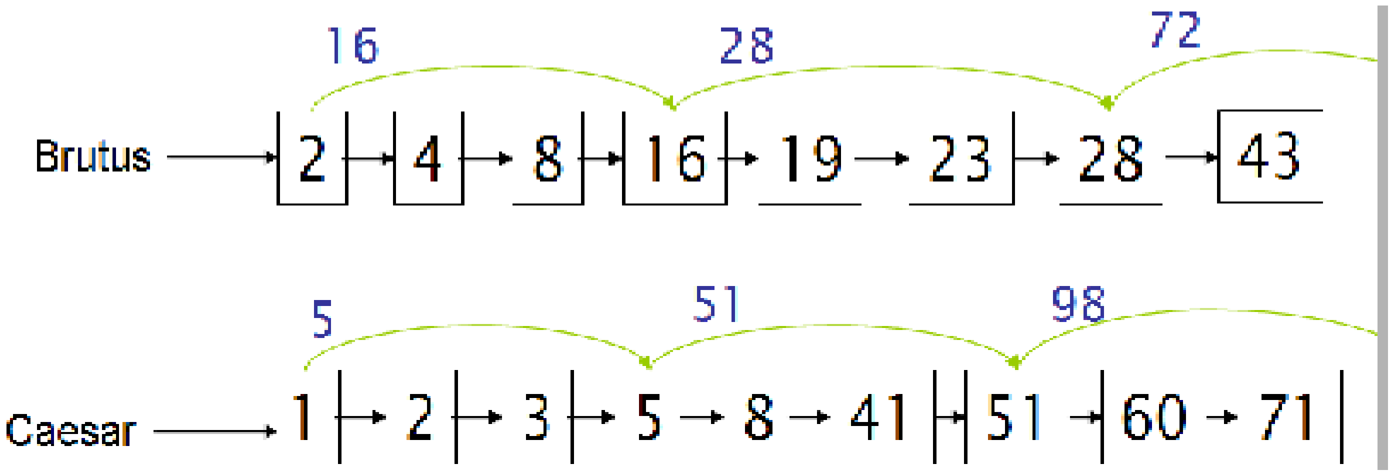
Example

- Documents containing “syria”
  - 1, 3, 12, 15, 19, 20, 34, 90, 96
- Documents containing “russia”
  - 1, 9, 10, 18, 19, 24, 35, 90, 101

What should we do if one of the posting lists is very small compared to the other?

What should we do if there are more than 2 posting lists?

Skip lists in indexing



- “Skips” are special shortcuts in the list
- Useful to avoid certain comparisons
- Good strategy is $\sqrt{p}$ skips for list of size p

Compressing inverted indexes

- Documents containing “robot”
  - 1, 3, 12, 15, 19, 20, 24
- Sorted in ascending order, could encode as (smaller) gaps
  - 1, +2, +9, +3, +4, +1, +4
- Gaps are small for frequent words and large for infrequent words
- Thus, compression can be obtained by encoding small values with shorter codes

Binary coding

Number (decimal)	Binary (16 bits)	Unary	
1	0000000000000001	0	
2	0000000000000010	10	
3	0000000000000011	110	
4	0000000000000100	1110	
5	0000000000000101	11110	
6	0000000000000110	111110	
7	0000000000000111	1111110	
8	0000000000001000	11111110	
9	0000000000001001	111111110	
10	0000000000001010	1111111110	

16 bits allows to encode gaps of 64K docids

Unary coding

Number (decimal)	Binary (16 bits)	Unary	
1	0000000000000001	0	
2	0000000000000010	10	
3	0000000000000011	110	
4	0000000000000100	1110	
5	0000000000000101	11110	
6	0000000000000110	111110	
7	0000000000000111	1111110	
8	0000000000001000	11111110	
9	0000000000001001	111111110	
10	0000000000001010	1111111110	

For small gaps this saves a lot of space

Elias-γ coding

- Unary code for $1 + \lfloor \log_2(x) \rfloor$
- Binary code of length $\lfloor \log_2(x) \rfloor$ for $x - 2^{\lfloor \log_2(x) \rfloor}$
- Example $10 = 2^3 + 2 = 1110010$

Elias-γ coding

Number (decimal)	Binary (16 bits)	Unary	Elias-γ
1	0000000000000001	0	0
2	0000000000000010	10	100
3	0000000000000011	110	101
4	0000000000000100	1110	11000
5	0000000000000101	11110	11001
6	0000000000000110	111110	11010
7	0000000000000111	1111110	11011
8	0000000000001000	11111110	1110000
9	0000000000001001	111111110	1110001
10	0000000000001010	1111111110	1110010

In practice, indexing with this coding uses about 1/5 of the space in TREC-3 (a collection of about 1GB of text)

Try it

Encode the list 1, 5, 14 using:

- *Standard binary coding (8 bits)*
- *Gap encoding in binary (8 bits)*
- *Gap encoding in unary*
- *Gap encoding in gamma coding*

Which one is shorter?

http://chato.cl/2015/data_analysis/exercise-answers/text-indexing_exercise_02_answer.txt