

Boolean retrieval & basics of indexing

CE-324: Modern Information Retrieval

Sharif University of Technology

M. Soleymani

Fall 2018

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

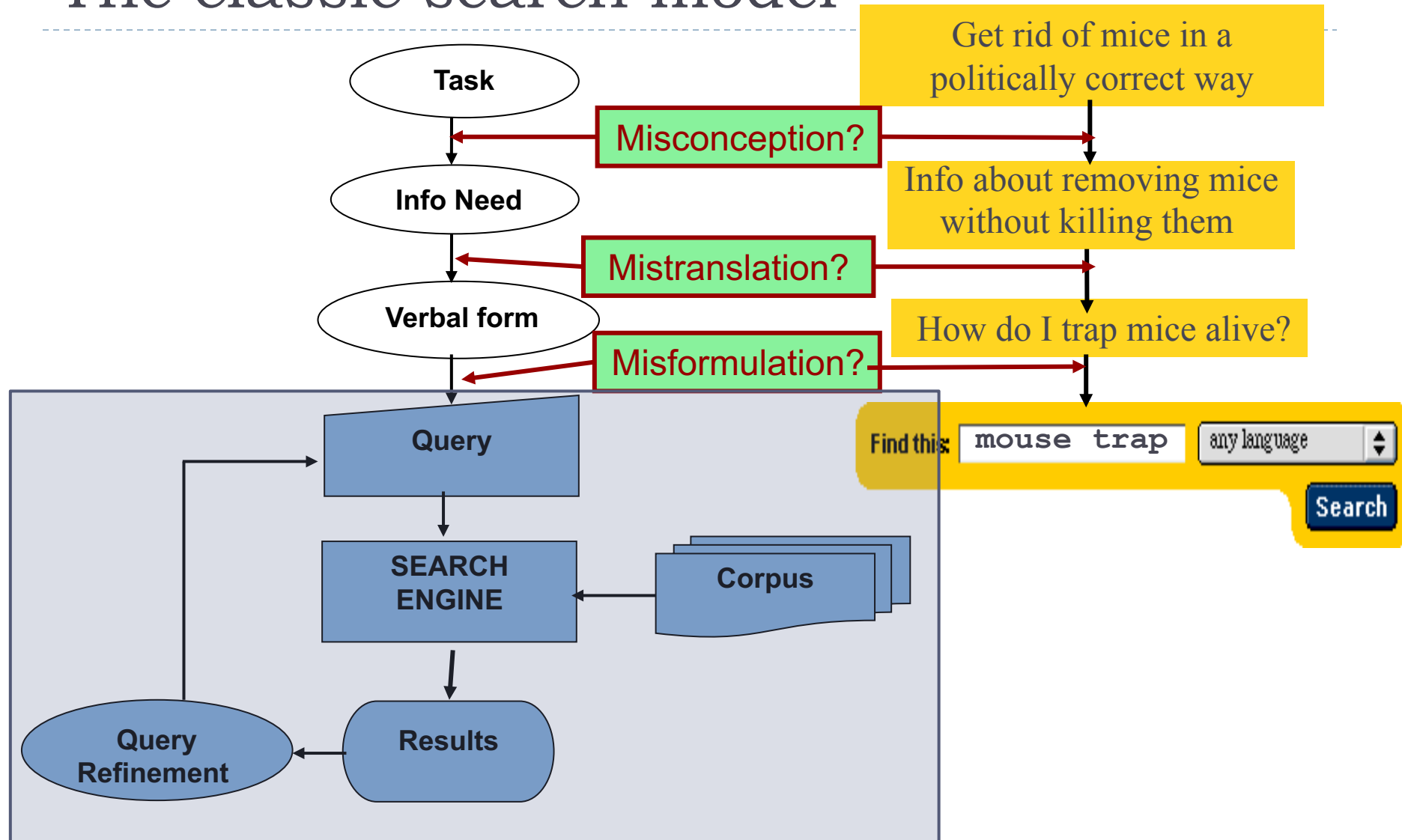
Boolean retrieval model

- ▶ Query: Boolean expressions
 - ▶ Boolean queries use AND, OR and NOT to join query terms
- ▶ Views each doc as a set of words
 - ▶ Term-incidence matrix is sufficient
 - ▶ Shows presence or absence of terms in each doc
- ▶ Perhaps the simplest model to build an IR system on

Boolean queries: Exact match

- ▶ In pure Boolean model, retrieved docs are not ranked
 - ▶ Result is a set of docs.
 - ▶ It is precise or exact match (docs match condition or not).
- ▶ Primary commercial retrieval tool for 3 decades (Until 1990's).
- ▶ Many search systems you still use are Boolean:
 - ▶ Email, library catalog, Mac OS X Spotlight

The classic search model



Example: Plays of Shakespeare

- ▶ Which plays of Shakespeare contain the words **Brutus** **AND Caesar** but **NOT Calpurnia**?
 - ▶ scanning all of Shakespeare's plays for **Brutus** and **Caesar**, then strip out those containing **Calpurnia**?
- ▶ The above solution cannot be the answer for large corpora (computationally expensive)
- ▶ Efficiency is also an important issue (along with the effectiveness)
 - ▶ **Index**: data structure built on the text to speed up the searches

Example: Plays of Shakespeare

Term-document incidence matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0



1 if play contains
word, 0 otherwise

Incidence vectors

- ▶ So we have a 0/1 vector for each term.
 - ▶ **Brutus AND Caesar** but NOT Calpurnia
- ▶ To answer query: take the vectors for **Brutus**, **Caesar** and **Calpurnia** (complemented) → bitwise AND.
 - ▶ 110100 AND 110111 AND 101111 = 100100.

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Answers to query

Brutus AND Caesar but NOT **Calpurnia**

► Antony and Cleopatra, Act III, Scene ii

Agrippa [Aside to DOMITIUS ENOBARBUS]: Why, Enobarbus,
When Antony found Julius **Caesar** dead,
He cried almost to roaring; and he wept
When at Philippi he found **Brutus** slain.

► Hamlet, Act III, Scene ii

Lord Polonius: I did enact Julius **Caesar** I was killed i' the
Capitol; **Brutus** killed me.



Bigger collections

- ▶ Number of docs: $N = 10^6$
- ▶ Average length of a doc \approx 1000 words
- ▶ No. of distinct terms: $M = 500,000$
- ▶ Average length of a word \approx 6 bytes
 - ▶ including spaces/punctuation
- ▶ 6GB of data

Sparsity of Term-document incidence matrix

▶ 500K x 1M matrix has half-a-trillion 0's and 1's.

▶ But it has no more than one billion 1's.

▶ matrix is extremely sparse.

▶ so a minimum of 99.8% of the cells are zero.

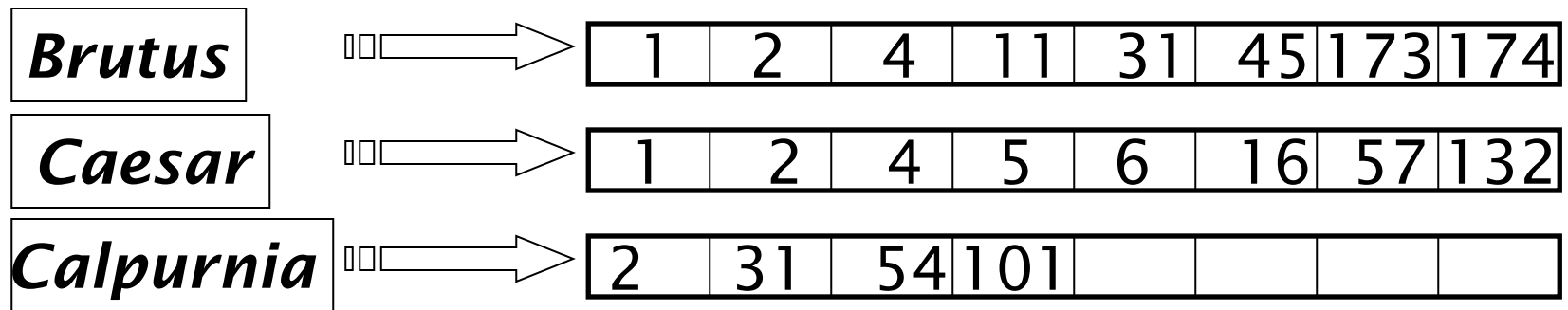


▶ What's a better representation?

▶ We only record the 1 positions.

Inverted index

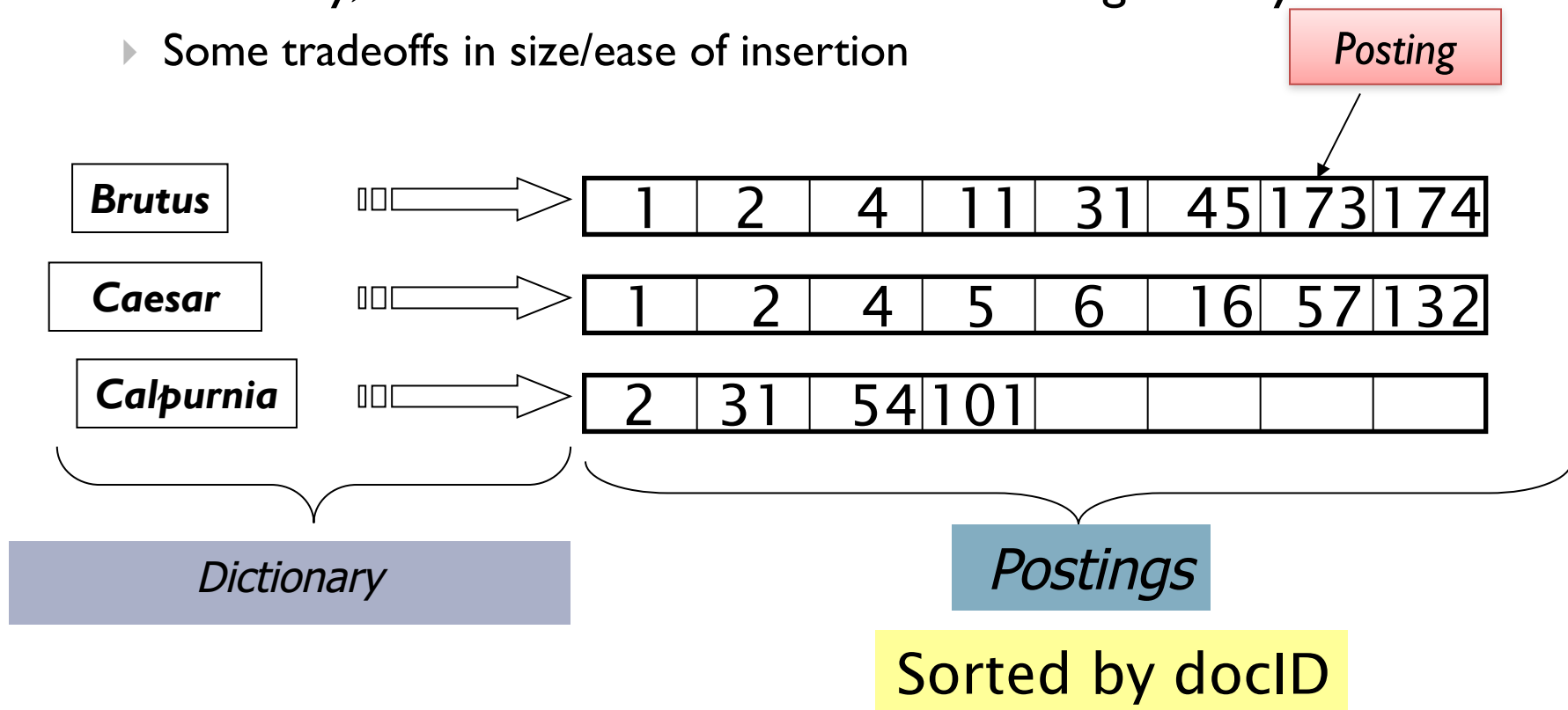
- ▶ For each term t , store a list of all docs that contain t .
 - ▶ Identify each by a **docID**, a document serial number
- ▶ Can we use fixed-size arrays for this?



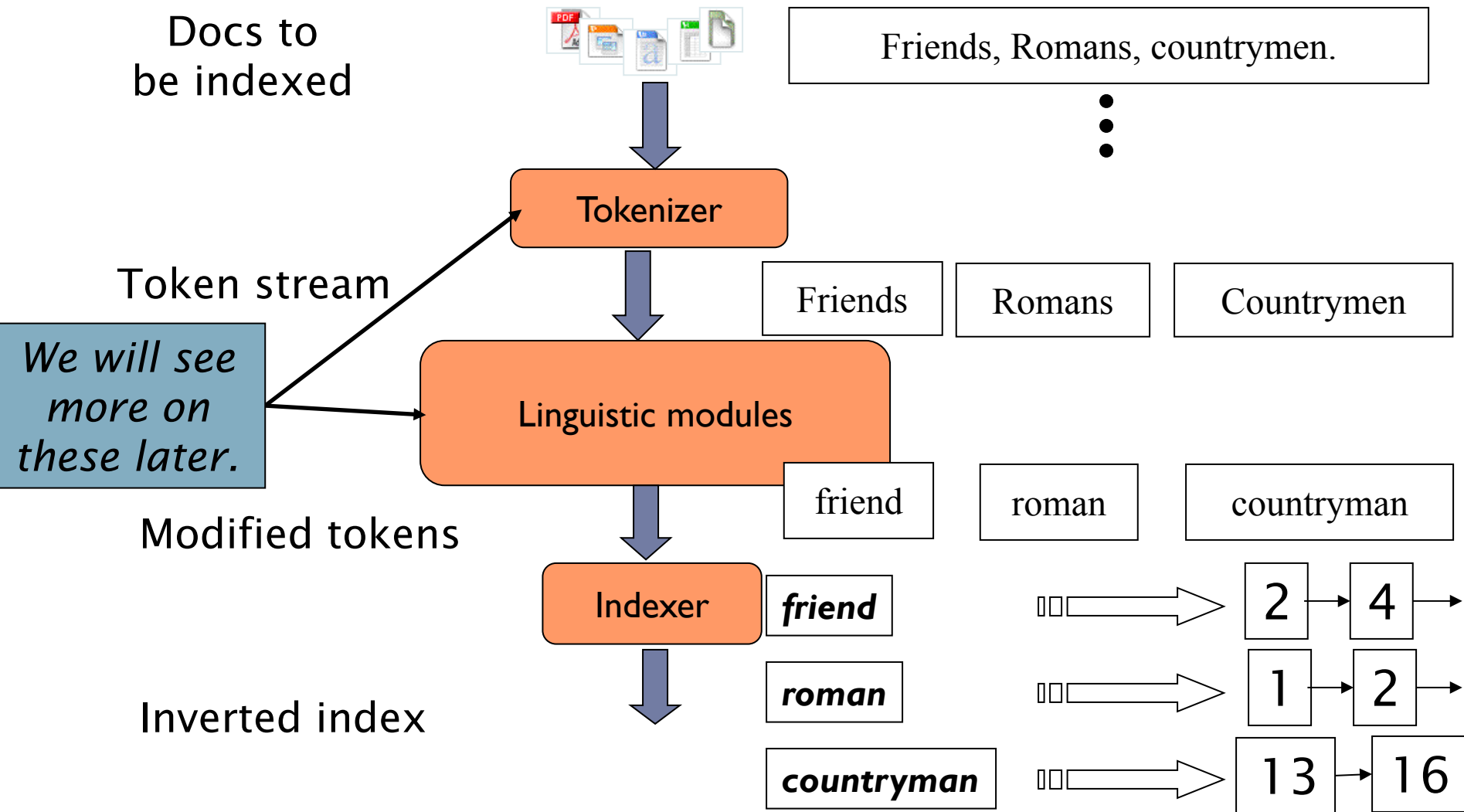
What happens if the word *Caesar* is added to doc 14?

Inverted index

- ▶ We need variable-size postings lists
 - ▶ On disk, a continuous run of postings is normal and best
 - ▶ In memory, can use linked lists or variable length arrays
 - ▶ Some tradeoffs in size/ease of insertion



Inverted index construction



Indexer steps: Token sequence

- ▶ Sequence of (Modified token, Document ID) pairs.

Doc 1

I did enact Julius
Caesar I was killed
i' the Capitol;
Brutus killed me.

Doc 2

So let it be with
Caesar. The noble
Brutus hath told you
Caesar was ambitious



Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Indexer steps: Sort

- ▶ Sort by terms
 - ▶ And then docID

Core indexing step

Term	docID	Term	docID
I	1	ambitious	2
did	1	be	2
enact	1	brutus	1
julius	1	brutus	2
caesar	1	capitol	1
I	1	caesar	1
was	1	caesar	2
killed	1	caesar	2
i'	1	did	1
the	1	enact	1
capitol	1	hath	1
brutus	1	I	1
killed	1	I	1
me	1	i'	1
so	2	it	2
let	2	julius	1
it	2	killed	1
be	2	killed	1
with	2	let	2
caesar	2	me	1
the	2	noble	2
noble	2	so	2
brutus	2	the	1
hath	2	the	2
told	2	told	2
you	2	you	2
caesar	2	was	1
was	2	was	2
ambitious	2	with	2

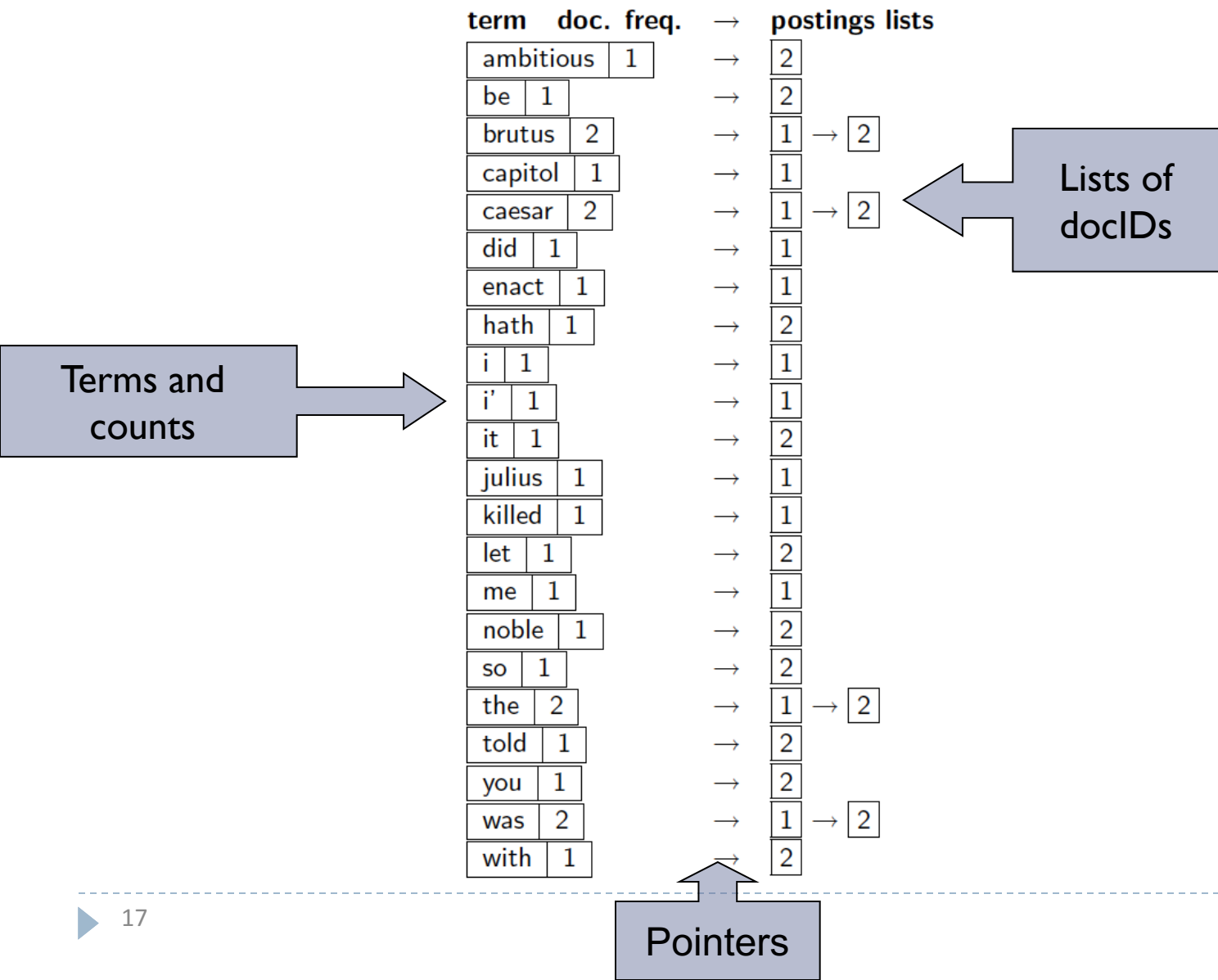
Indexer steps: Dictionary & Postings

- ▶ Multiple term entries in a single doc are merged.
- ▶ Split into Dictionary and Postings
- ▶ Document frequency information is added.

Why frequency?
Will discuss later.

Term	docID	term	doc.	freq.	→	postings lists
ambitious	2	ambitious	1		→	2
be	2	be	1		→	2
brutus	1	brutus	2		→	1 → 2
brutus	2					
capitol	1	capitol	1		→	1
caesar	1	caesar	2		→	1 → 2
caesar	2					
caesar	2					
did	1	did	1		→	1
enact	1	enact	1		→	1
hath	1	hath	1		→	2
I	1	i	1		→	1
I	1	i'	1		→	1
i'	1					
it	2	it	1		→	2
julius	1	julius	1		→	1
killed	1	killed	1		→	1
killed	1					
let	2	let	1		→	2
me	1	me	1		→	1
noble	2	noble	1		→	2
so	2	so	1		→	2
the	1					
the	2	the	2		→	1 → 2
told	2	told	1		→	2
you	2	you	1		→	2
was	1	was	2		→	1 → 2
was	2					
with	2	with	1		→	2

Where do we pay in storage?



A naïve dictionary

- ▶ An array of struct:

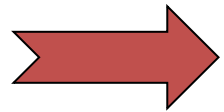
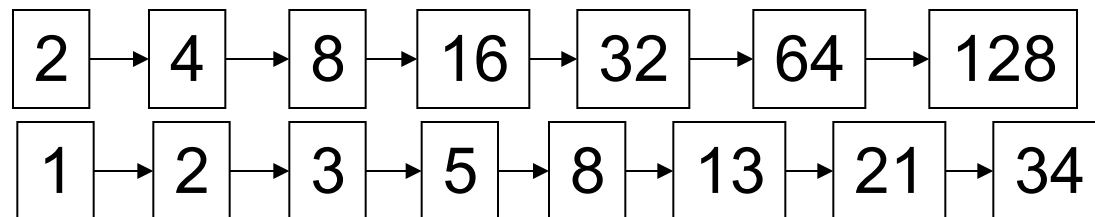
term	document frequency	pointer to postings list
a	656,265	→
aachen	65	→
...
zulu	221	→

Query processing: AND

- ▶ Consider processing the query:
Brutus AND Caesar
 - ▶ Locate ***Brutus*** in the dictionary;
 - ▶ Retrieve its postings.
 - ▶ Locate ***Caesar*** in the dictionary;
 - ▶ Retrieve its postings.
 - ▶ “Merge” (intersect) the two postings:

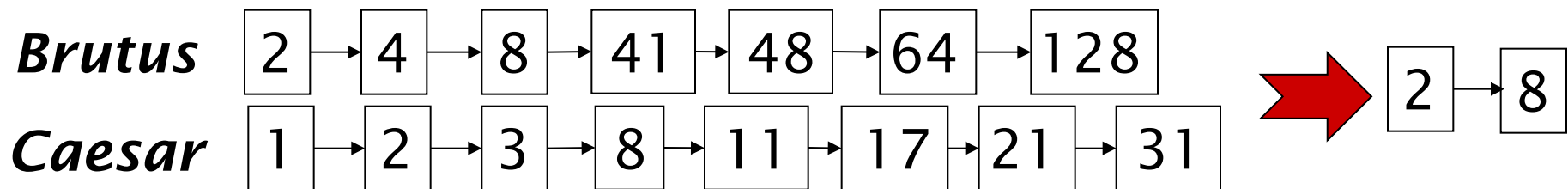
Brutus

Caesar



The merge

- ▶ Walk through the two postings simultaneously, in time linear in the total number of postings entries



If list lengths are x and y , merge takes $O(x+y)$ operations.
Crucial: postings sorted by docID.

Intersecting two postings lists (a “merge” algorithm)

```
INTERSECT( $p_1, p_2$ )
1   $answer \leftarrow \langle \rangle$ 
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3  do if  $docID(p_1) = docID(p_2)$ 
4      then  $\text{ADD}(answer, docID(p_1))$ 
5           $p_1 \leftarrow next(p_1)$ 
6           $p_2 \leftarrow next(p_2)$ 
7      else if  $docID(p_1) < docID(p_2)$ 
8          then  $p_1 \leftarrow next(p_1)$ 
9          else  $p_2 \leftarrow next(p_2)$ 
10 return  $answer$ 
```

Boolean queries: More general merges

- ▶ Exercise: Adapt the merge for the queries:

Brutus AND NOT Caesar

Brutus OR NOT Caesar

Can we still run through the merge in time $O(x + y)$?

Merging

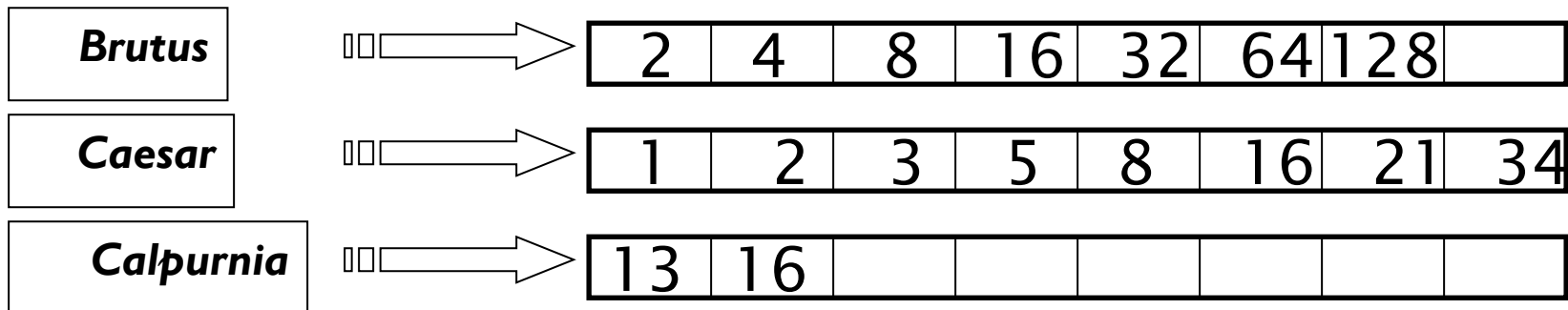
What about an arbitrary Boolean formula?

(Brutus OR Caesar) AND NOT (Antony OR Cleopatra)

- ▶ Can we merge in “linear” time for general Boolean queries?
 - ▶ Linear in what?
- ▶ Can we do better?

Query optimization

- ▶ What is the best order for query processing?
- ▶ Consider a query that is an *AND* of n terms.
- ▶ For each of the n terms, get its postings, then *AND* them together.

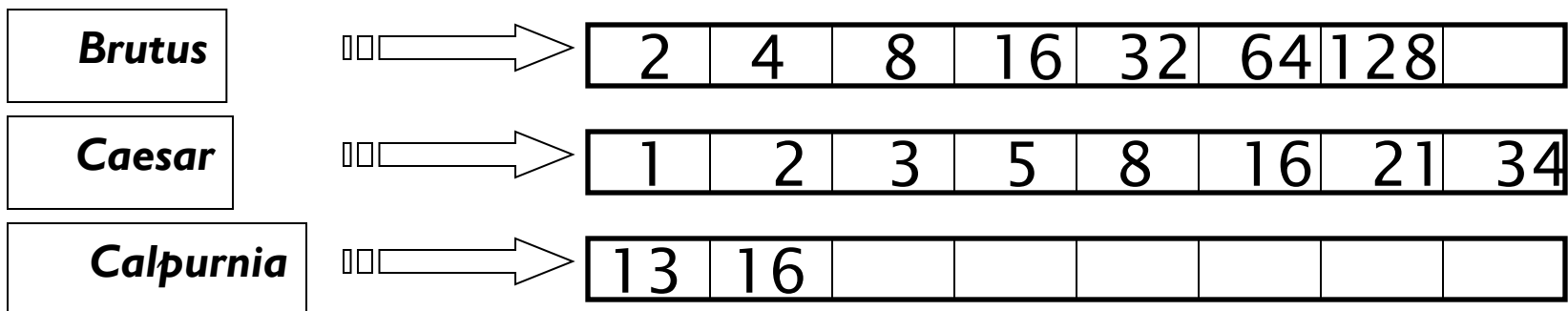


Query: Brutus AND Calpurnia AND Caesar

Query optimization example

- ▶ Process in order of increasing freq:
 - ▶ *start with smallest set, then keep cutting further.*

This is why we kept
document freq. in dictionary



Execute the query as (***Calpurnia AND Brutus***) ***AND Caesar***.

More general optimization

- ▶ Example:

(madding OR crowd) AND (ignoble OR strife)

- ▶ Get doc frequencies for all terms.
- ▶ Estimate the size of each *OR* by the sum of its doc. freq.'s (conservative).
- ▶ Process in increasing order of *OR* sizes.

Summary of Boolean IR:

Advantages of exact match

- ▶ It can be implemented very efficiently
- ▶ Predictable, easy to explain
 - ▶ precise semantics
- ▶ Structured queries for pinpointing precise docs
 - ▶ neat formalism
- ▶ Work well when you know exactly (or roughly) what the collection contains and what you're looking for

Summary of Boolean IR:

Disadvantages of the Boolean Model

- ▶ Query formulation (Boolean expression) is difficult for most users
 - ▶ Too simplistic Boolean queries by most users
 - ▶ AND, OR as opposite extremes in a precision/recall tradeoff
 - ▶ Usually either too few or too many docs in response to a user query
- ▶ Retrieval based on binary decision criteria
 - ▶ No ranking of the docs is provided
- ▶ Difficulty increases with collection size

Ranking results in advanced IR models

- ▶ Boolean queries give inclusion or exclusion of docs.
 - ▶ Results of queries in Boolean model as a set
- ▶ Modern information retrieval systems are no longer based on the Boolean model
- ▶ Often we want to rank/group results
 - ▶ Need to measure proximity from query to each doc.
 - ▶ Index term weighting can provide a substantial improvement