

CS3245

Information Retrieval

Lecture 2: Boolean retrieval

2



Live Q&A
<https://pollev.com/jin>

Last Time: Ngram Language Models

- Unigram LM: Bag of words
- Ngram LM: use $n-1$ tokens of context to predict n^{th} token
- Larger n -gram models more accurate but each increase in order requires exponentially more space

We'll return to this in probabilistic information retrieval.

Information Retrieval (IR)



Finding materials (usually documents)

of an **unstructured** nature (usually text)

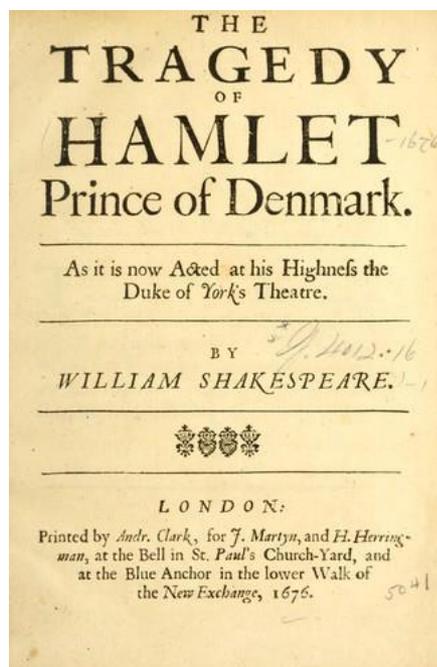
that satisfy an **information need**

from within **large collections** (usually stored on computers).

Let's start with Boolean Retrieval with Shakespeare!

Boolean Retrieval with Shakespeare

- **The collection:** ~40 plays by Shakespeare
 - <http://shakespeare.mit.edu/index.html>



Enter HAMLET

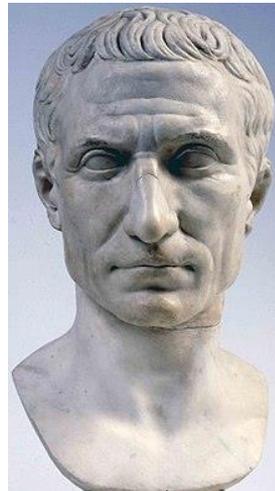
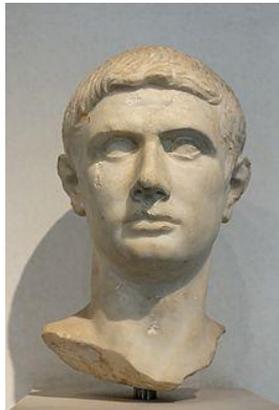
HAMLET

To be, or not to be, that is the question,
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,



Boolean Retrieval with Shakespeare

- **The information need** (in verbal form): Which plays of Shakespeare mention *Brutus* and *Caesar* but *not Calpurnia*?





Boolean Retrieval with Shakespeare

- **The information need** (in verbal form): Which plays of Shakespeare mention *Brutus* and *Caesar* but *not Calpurnia*?
- 3 conditions to be satisfied at the same time
 1. Mentions Brutus
 2. Mentions Caesar
 3. Does not mention Calpurnia
- **The query**: Brutus AND Caesar AND (NOT Calpurnia)

Boolean Retrieval with Shakespeare

- **The query:** Brutus AND Caesar AND (NOT Calpurnia)
- Naïve Approach:
 - For each play, run CTRL+F for **Brutus**, **Caesar**, and **Calpurnia**, separately
 - If there is **at least one** match for **Brutus**, **at least one** for **Caesar**, but **none** for **Calpurnia**, add this play to the result
- It's one solution, but why isn't it the only solution?
 - Too Slow! (for large corpora)

Indexing



- **The Index:** 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 the **play** contains the **word**, 0 otherwise

Blanks on slides, you may want to fill in

Query processing



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

Brutus AND Caesar AND (NOT Calpurnia)

- Take the rows for **Brutus**, **Caesar** and **Calpurnia** (complemented, why?) and bitwise **AND** them.

Results



- Antony and Cleopatra, Act III, Scene ii

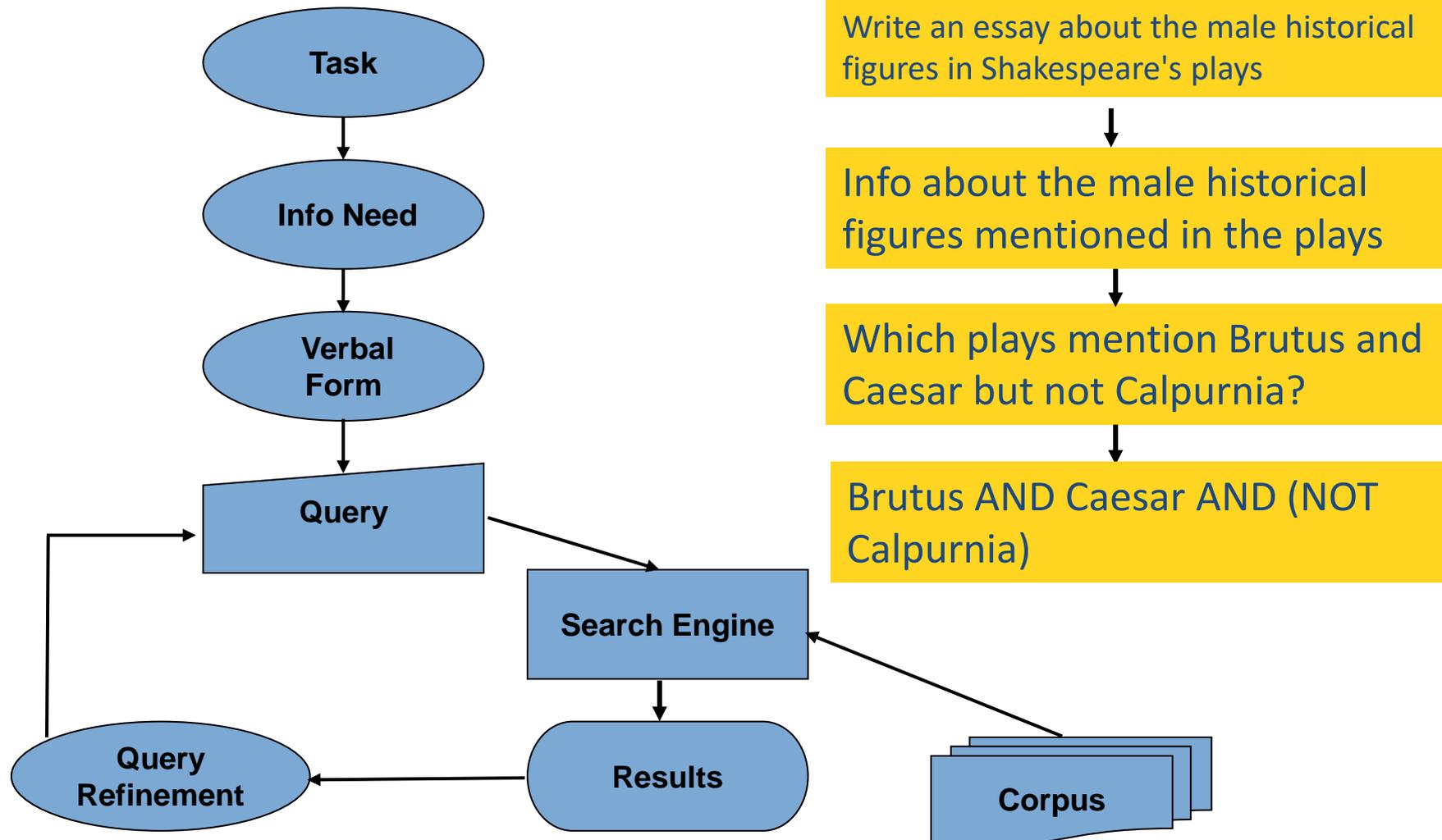
Agrippa [Aside to DOMITIUS ENOBARBUS]: Why, Enobarbus,
When Antony found Julius **Caesar** dead,
He cried almost to roaring; and he wept
When at Philippi he found **Brutus** slain.

- Hamlet, Act III, Scene ii

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



The classic search model



Relevance is the key!



- Information Retrieval (IR) is **finding material** (usually documents) ... that satisfies an **information need** ...
- **Relevance**: Whether the documents returned help to satisfy the information need.
- Evaluation metrics (to be covered in Week 9)
 - **Precision** : Fraction of retrieved docs that are **relevant** to user's information need
 - **Recall** : Fraction of **relevant** docs in collection that are retrieved

Bigger collections



- Consider $N = 1$ million documents, each with about 1000 words.
 - 1000×1 million = 1 billion words in total
- Avg 6 bytes/word including spaces/punctuation
 - 6GB of data in the documents.
- Say there are $M = 500K$ *distinct* terms among these.

Tough to build the matrix



- $500\text{K} \times 1\text{M}$ matrix has half a trillion 0s and 1s. B-I-G
- But it is extremely sparse.
 - Each document is 1000 words long \rightarrow
 - At most **1K** 1s among the **500K** cells in each **column** (i.e., document).
- Can we do better?

Inverted index



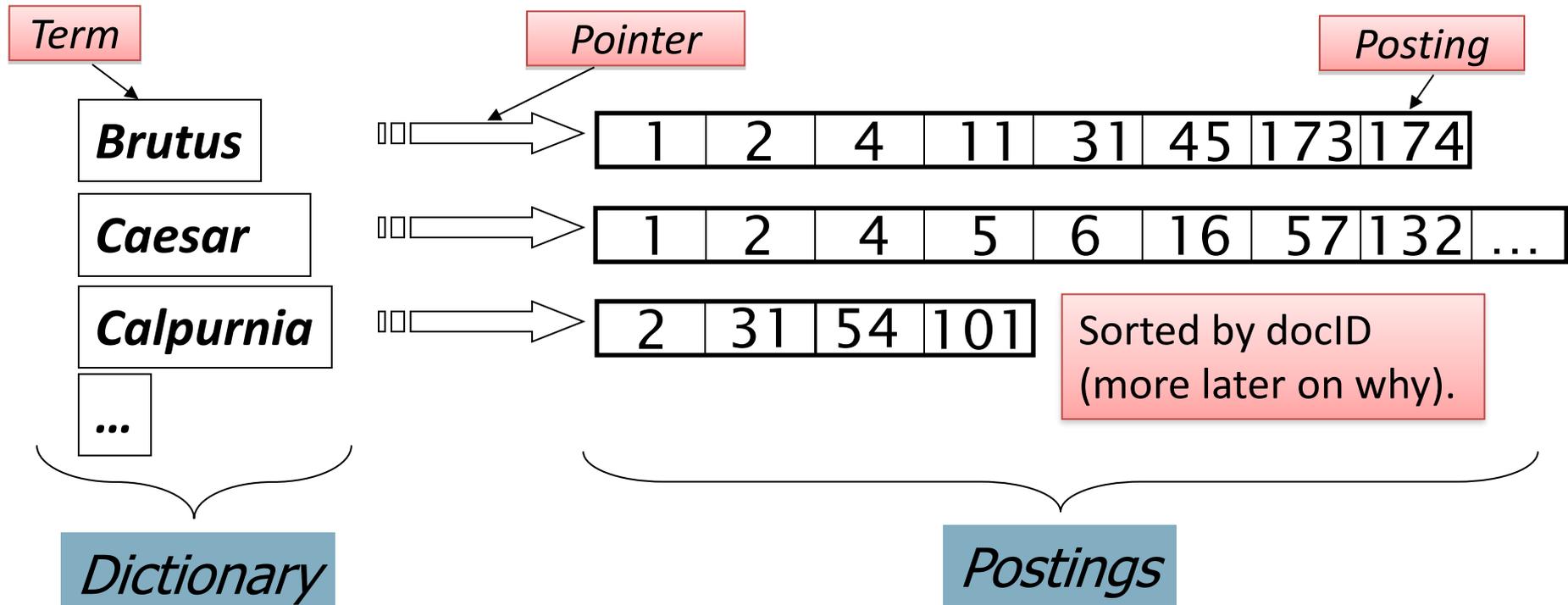
- Observation 1: It is easier to manage documents by serial numbers instead of titles.

24601

- Identify each document by a unique serial number called **docID**.
 - Antony and Cleopatra → 1
 - Julius Caesar → 2
 - ...

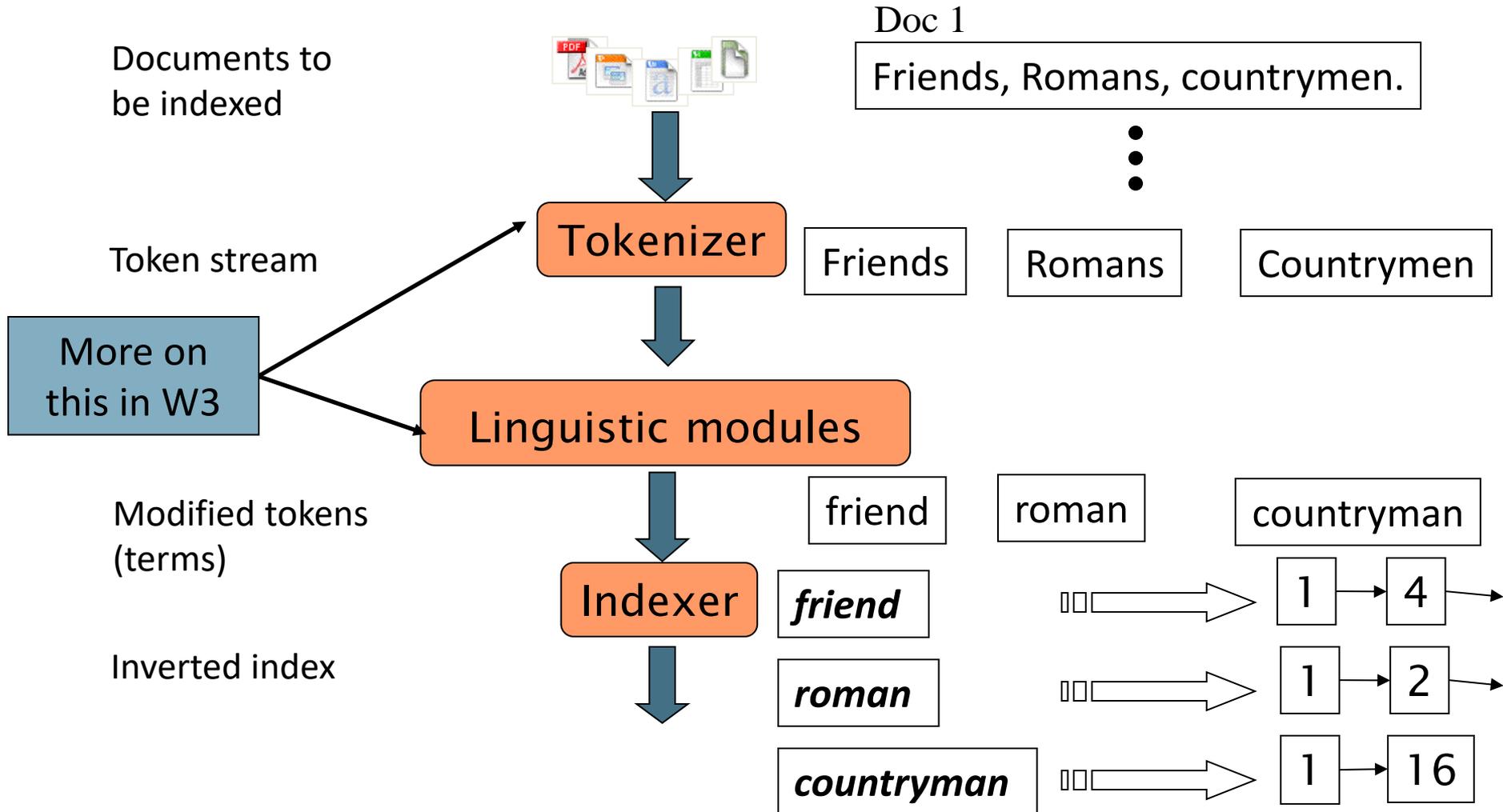
Inverted index

- Observation 2: Searches start with terms
- For each term t , store a **list** of all documents that contain t .





Inverted index construction



Indexer steps:

Generate token sequence



- Generate sequence of (Term, DocID) 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
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



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

Indexer steps: Dictionary & Postings

- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings
- Document frequency information is also stored.

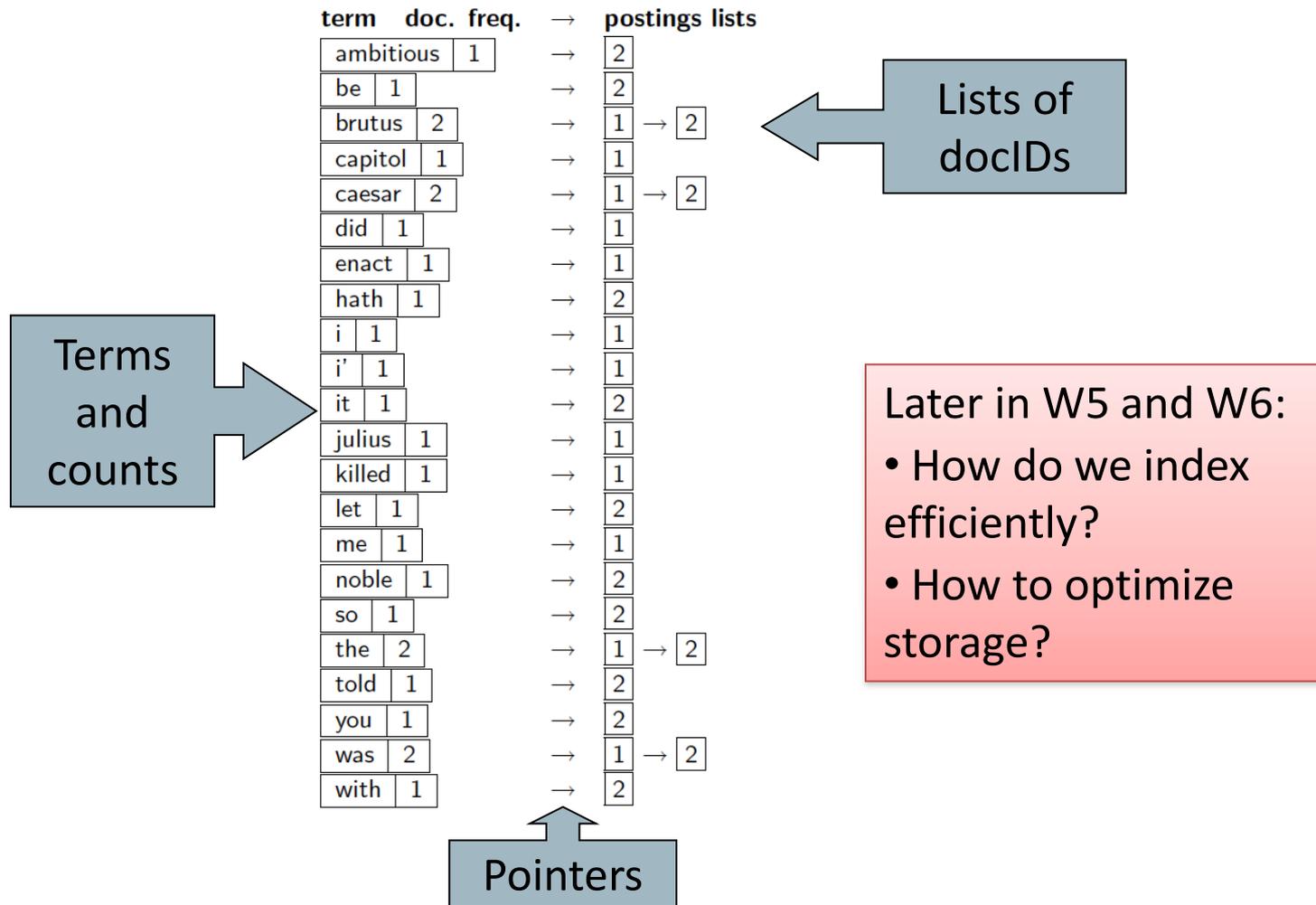
Why frequency?
Will discuss later.

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



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

What do we pay in storage?



How do we process queries?



- Boolean queries
 - AND
 - OR
 - NOT

- Basic query optimization

Later in W3 and W4 –

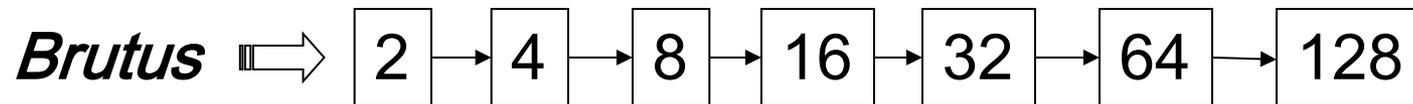
- How to further optimize query processing ?
- What other kinds of queries can we process?

Query processing: AND

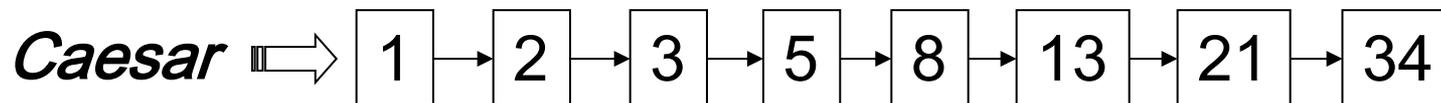
- How to process this query?

Brutus AND *Caesar*

- Locate *Brutus* in the Dictionary;
 - Retrieve its postings.



- Locate *Caesar* in the Dictionary;
 - Retrieve its postings.

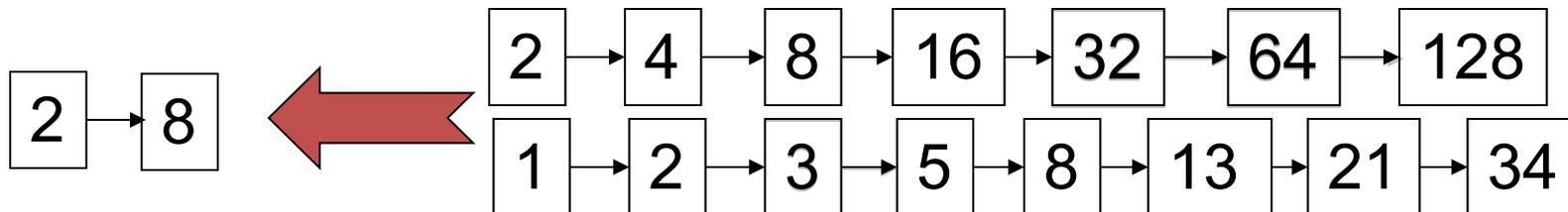


- "Merge" the two postings
 - (Intersect) Keep only the common entries.

The merge



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



If the list lengths are x and y , the merge takes $O(x+y)$ operations.
Crucial: postings must be 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  $\text{docID}(p_1) = \text{docID}(p_2)$ 
4      then  $\text{ADD}(\text{answer}, \text{docID}(p_1))$ 
5           $p_1 \leftarrow \text{next}(p_1)$ 
6           $p_2 \leftarrow \text{next}(p_2)$ 
7      else if  $\text{docID}(p_1) < \text{docID}(p_2)$ 
8          then  $p_1 \leftarrow \text{next}(p_1)$ 
9          else  $p_2 \leftarrow \text{next}(p_2)$ 
10 return answer
```

Query processing: OR



- How to process this query?

Brutus OR Caesar

- Locate ***Brutus*** in the Dictionary;
 - Retrieve its postings.
- Locate ***Caesar*** in the Dictionary;
 - Retrieve its postings.
- "Merge" the two postings
 - (Union) Keep all entries that appear in any of the two postings.

Query processing: NOT



- How to process this query?

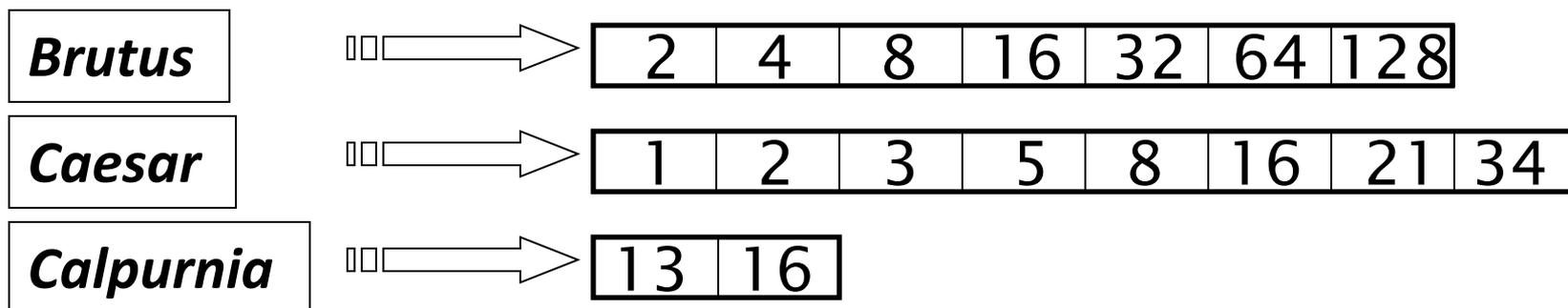
NOT Brutus

- Retrieve the full list of documents
- Locate ***Brutus*** in the Dictionary;
 - Retrieve its postings.
- "Merge" the full list and the postings
 - (Filtering) Keep all entries that appear the full list but not in the postings.

Query optimization



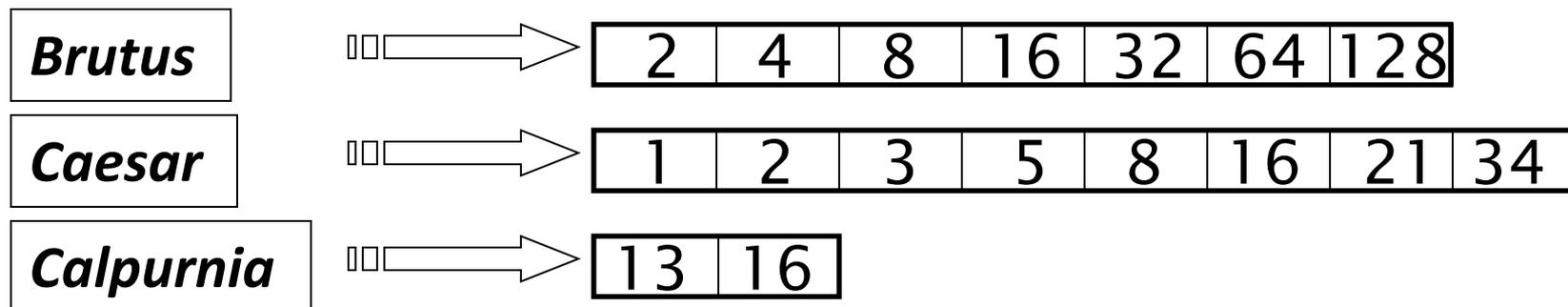
- Consider a query that is an *AND* of n terms.
Brutus AND Caesar AND Calpurnia
- What is the best order for query processing?



Query optimization example

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

This is why we kept
document frequency in the dictionary!



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

More general optimization



e.g., (*madding OR crowd*) AND (*ignoble OR strife*)
AND (*killed OR slain*)

- Get document frequencies (**dfs**), for all terms.
- Estimate the size of each *OR* by the sum of its **dfs** (conservative).
- Process in increasing order of *OR* sizes.



Check your understanding

- Recommend a query processing order for

*(tangerine OR trees) AND
(marmalade OR skies) AND
(kaleidoscope OR eyes)*

Term	Freq
eyes	213312
kaleidoscop	87009
marmalade	107913
skies	271658
tangerine	46653
trees	316812

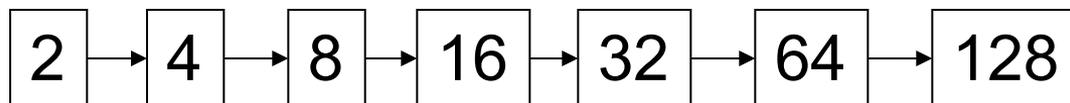
Mixing AND/OR with NOT



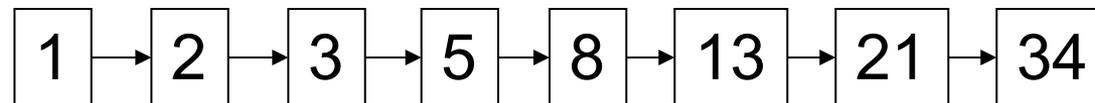
- How about these queries?

Brutus AND NOT Caesar

Brutus OR NOT Caesar



Brutus



Caesar

Question: Can we still process the query in $O(x+y)$?

What can we achieve?

Boolean Retrieval



- The **Boolean retrieval model** is able to process queries which are based on Boolean expressions:
 - Views each document as a set of words
 - Is precise: document matches condition or not.
- Perhaps the simplest model to build an IR system on
- Primary commercial retrieval tool for 3 decades.
- Many search systems you still use are Boolean:
 - E.g., Library Catalog in NUS
<https://linc.nus.edu.sg/search/Y>

Example: WestLaw

<http://www.westlaw.com/>



- Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992)
 - Tens of terabytes of data; 700,000 users
- Long, precise queries; proximity operators; incrementally developed; not like web search
 - What is the statute of limitations in cases involving the federal tort claims act?
LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM
 - /3 = within 3 words, /S = in same sentence

Example: WestLaw

<http://www.westlaw.com/>



- Many professional searchers still like Boolean search
 - You know exactly what you are getting
- But that doesn't mean it actually works better...

Summary

Covered the whole of information retrieval from 1000 feet up

- Indexing to store information efficiently for both space and query time.
- Run time builds relevant document list. Must be *f a s t*.

Resources for today's lecture

- *Introduction to Information Retrieval*, chapter 1