Information Retrieval

IR basics

Boolean search, Inverted index

Adapted from several IR courses:
C. Manning, P. Raghavan and H. Schütze (http://nlp.stanford.edu/IR-book/)
W. Croft, D. Metzler, and T. Strohman (http://www.search-engines-book.com/)

IR in 1680

- Which plays of Shakespeare contain the words ***Brutus*** AND ***Caesar*** but NOT ***Calpurnia***?
- One could grep all of Shakespeare's plays for ***Brutus*** and ***Caesar***, then strip out lines containing ***Calpurnia***?

IR in 1680

- Which plays of Shakespeare contain the words ***Brutus*** AND ***Caesar*** but NOT ***Calpurnia***?
- One could grep all of Shakespeare's plays for ***Brutus*** and ***Caesar***, then strip out lines containing ***Calpurnia***?

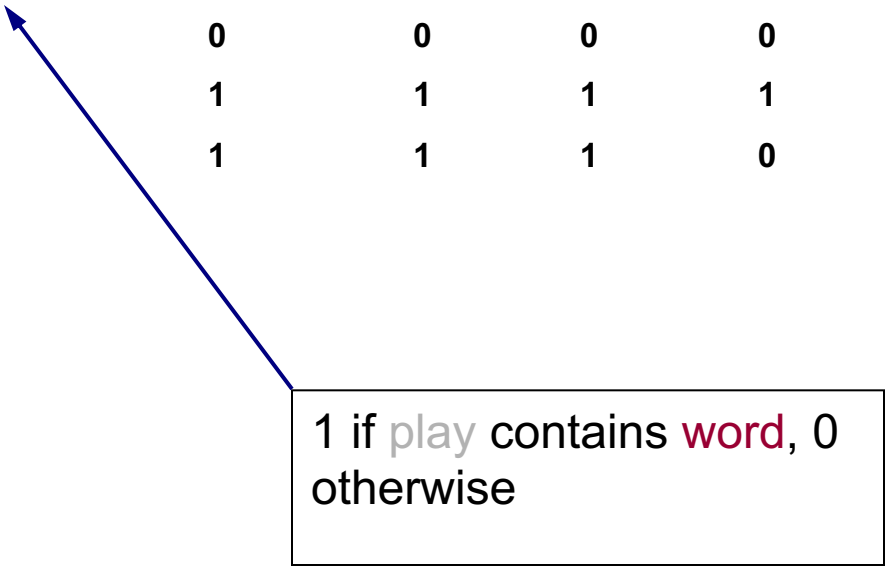
```
time grep brutus t8.shakespeare.txt
real    0m0.082
suser   0m0.069
ssys    0m0.008

wc t8.shakespeare.txt
124456  901325 5458199 t8.shakespeare.txt
```

IR in 1680

- Which plays of Shakespeare contain the words ***Brutus*** AND ***Caesar*** but NOT ***Calpurnia***?
- One could grep all of Shakespeare's plays for ***Brutus*** and ***Caesar***, then strip out lines containing ***Calpurnia***?
- Why is that not the answer?
  - Slow (for large corpora)
  - NOT ***Calpurnia*** is non-trivial
  - Other operations (e.g., find the word ***Romans*** near ***countrymen***) not feasible
  - Ranked retrieval (best documents to return)

Term-document incidence

	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



Incidence vectors

- So we have a 0/1 vector for each term.
- To answer query?

  Brutus AND ***Caesar*** but NOT ***Calpurnia***

- take the vectors for ***Brutus***, ***Caesar*** and ***Calpurnia*** (complemented) ➔ bitwise *AND*.

- 110100 *AND* 110111 *AND* 101111 = 100100.

Term-document incidence

Brutus AND ***Caesar*** AND (NOT ***Calpurnia***)

	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
NOT Calpurnia	1	0	1	1	1	1
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Term-document incidence

Brutus AND ***Caesar*** AND (NOT ***Calpurnia***)

	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
NOT Calpurnia	1	0	1	1	1	1
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Answers to query

- 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

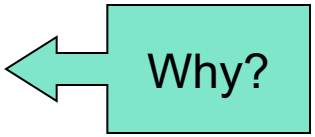
Lord Polonius: I did enact Julius ***Caesar*** I was killed i' the Capitol; ***Brutus*** killed me.

Bigger collections

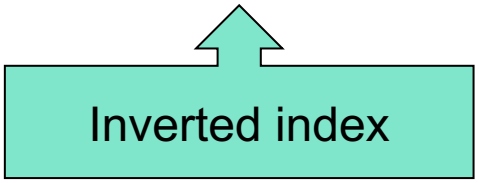
- Consider *N* = 1 million documents, each with about 1000 words.
- Avg 6 bytes/word including spaces/punctuation
  - 6GB of data in the documents.
- Say there are *M* = 500K *distinct* terms among these.

Can't build the matrix

- 500K x 1M matrix has half-a-trillion 0's and 1's.
- But it has no more than one thousand million 1's.
  - matrix is extremely sparse.

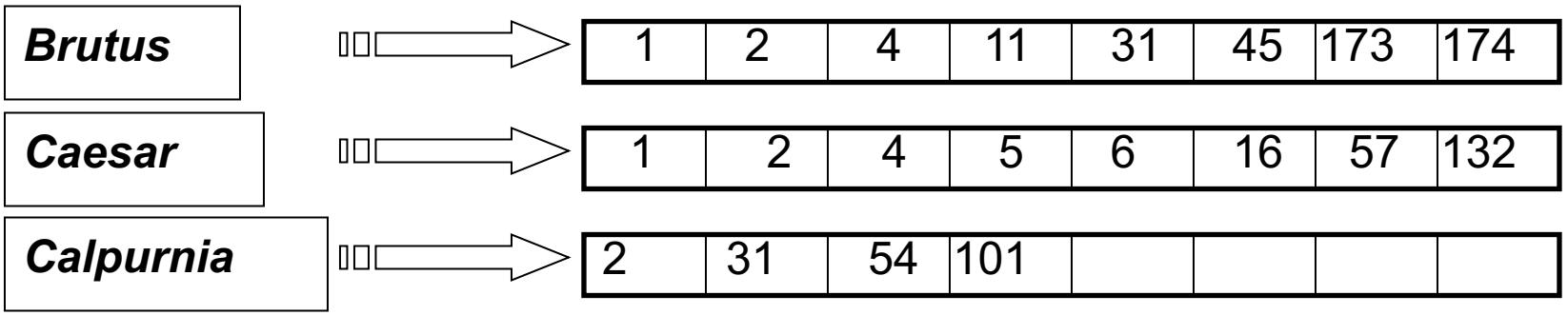


- What's a better representation?
  - Record only the "1" positions.



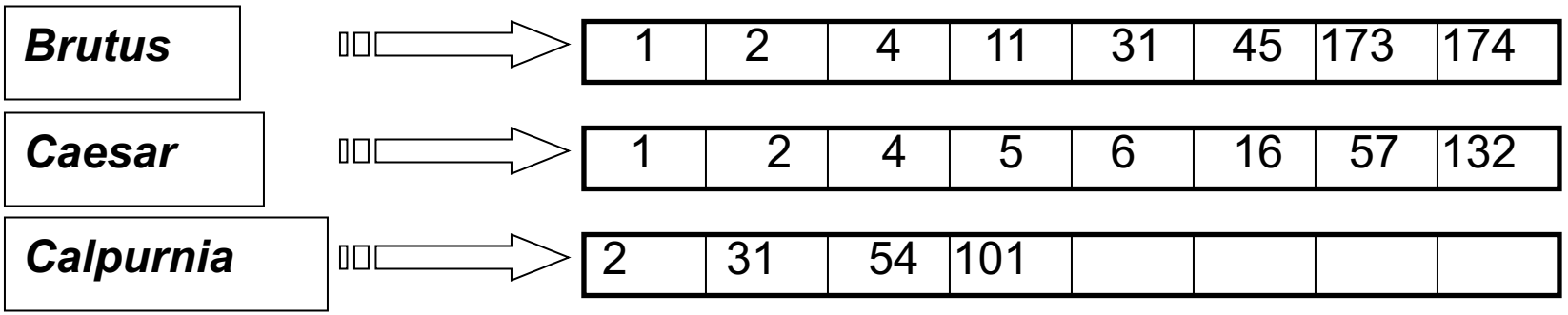
Inverted index

- For each term *t*, we must store a list of all documents that contain *t*.
  - Identify each by a **docID**, a document serial number



Inverted index

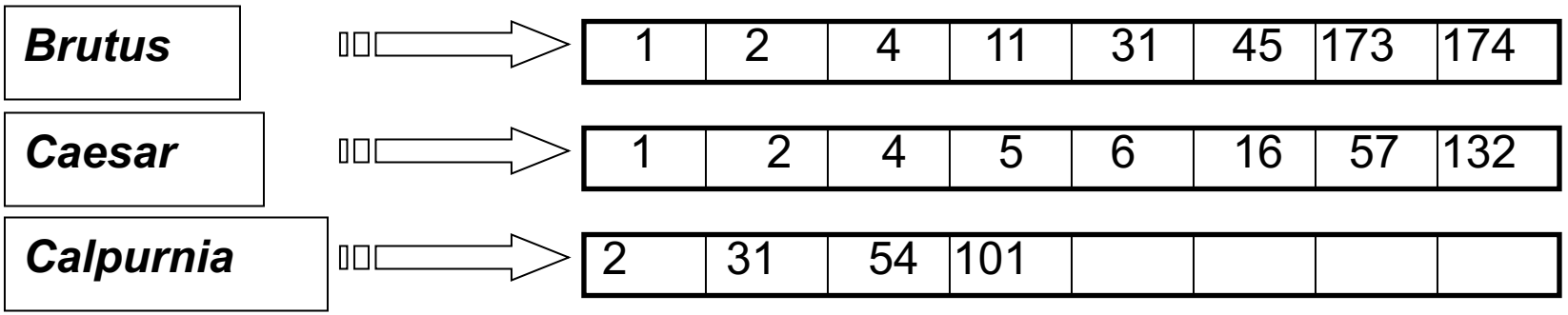
- For each term *t*, we must store a list of all documents that contain *t*.
  - Identify each by a **docID**, a document serial number



- Use fixed sized arrays?

Inverted index

- For each term *t*, we must store a list of all documents that contain *t*.
  - Identify each by a **docID**, a document serial number

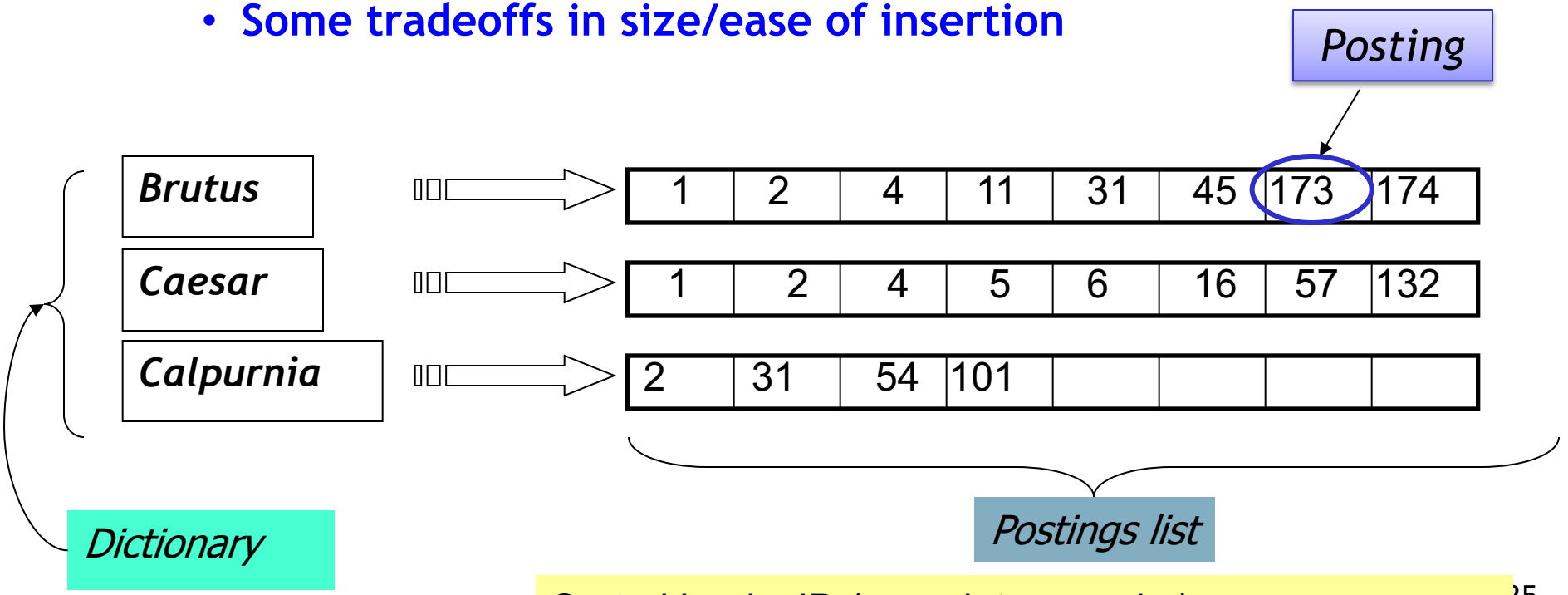


- Use fixed sized arrays?

What happens if we add a new document with the word ***Caesar***?

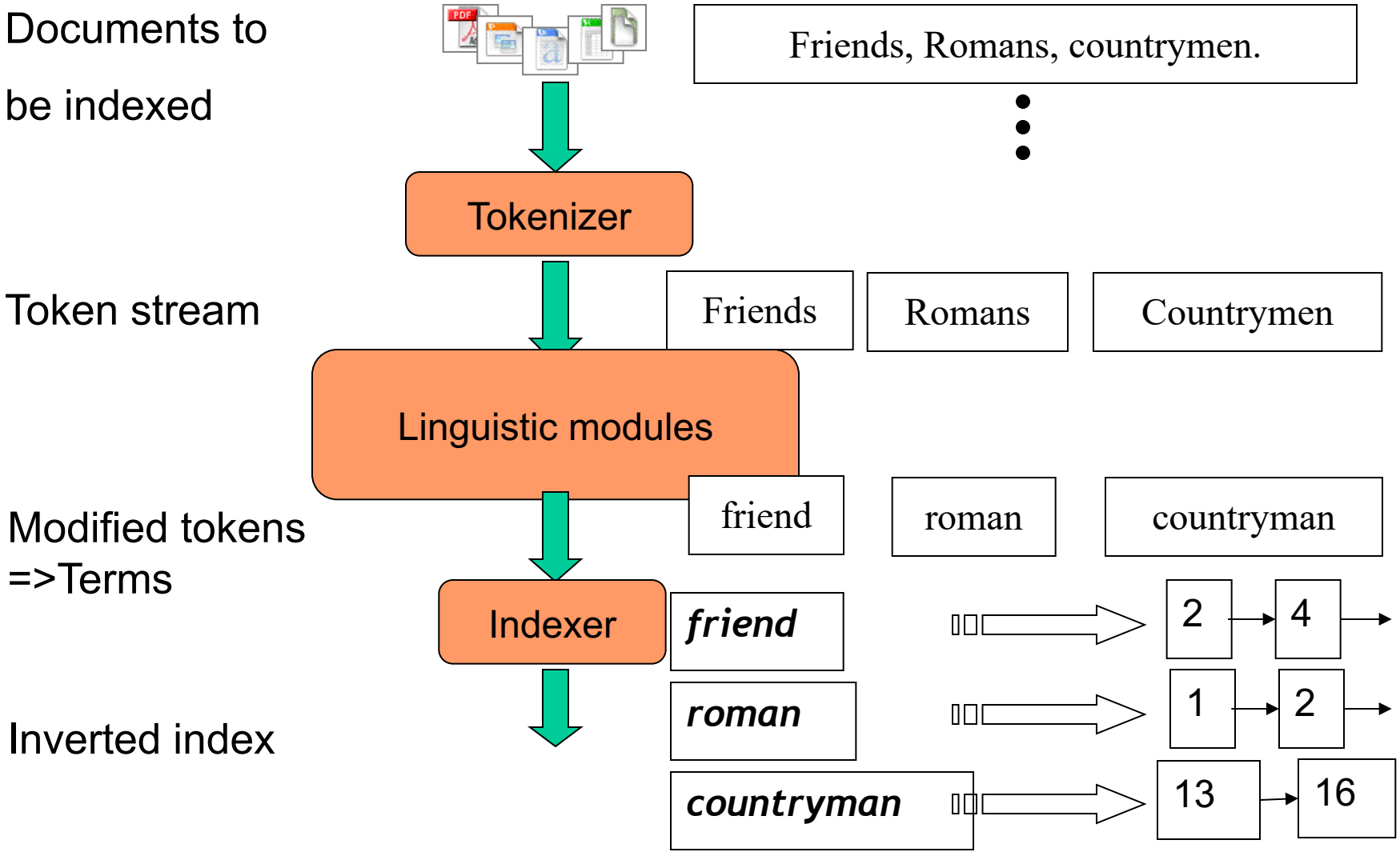
Inverted index

- We need variable-size postings lists
  - On disk, a continuous run of postings is normal and best
    - **E.g. comma-separated in a text file**
  - In memory, can use linked lists or variable length arrays
    - **Some tradeoffs in size/ease of insertion**

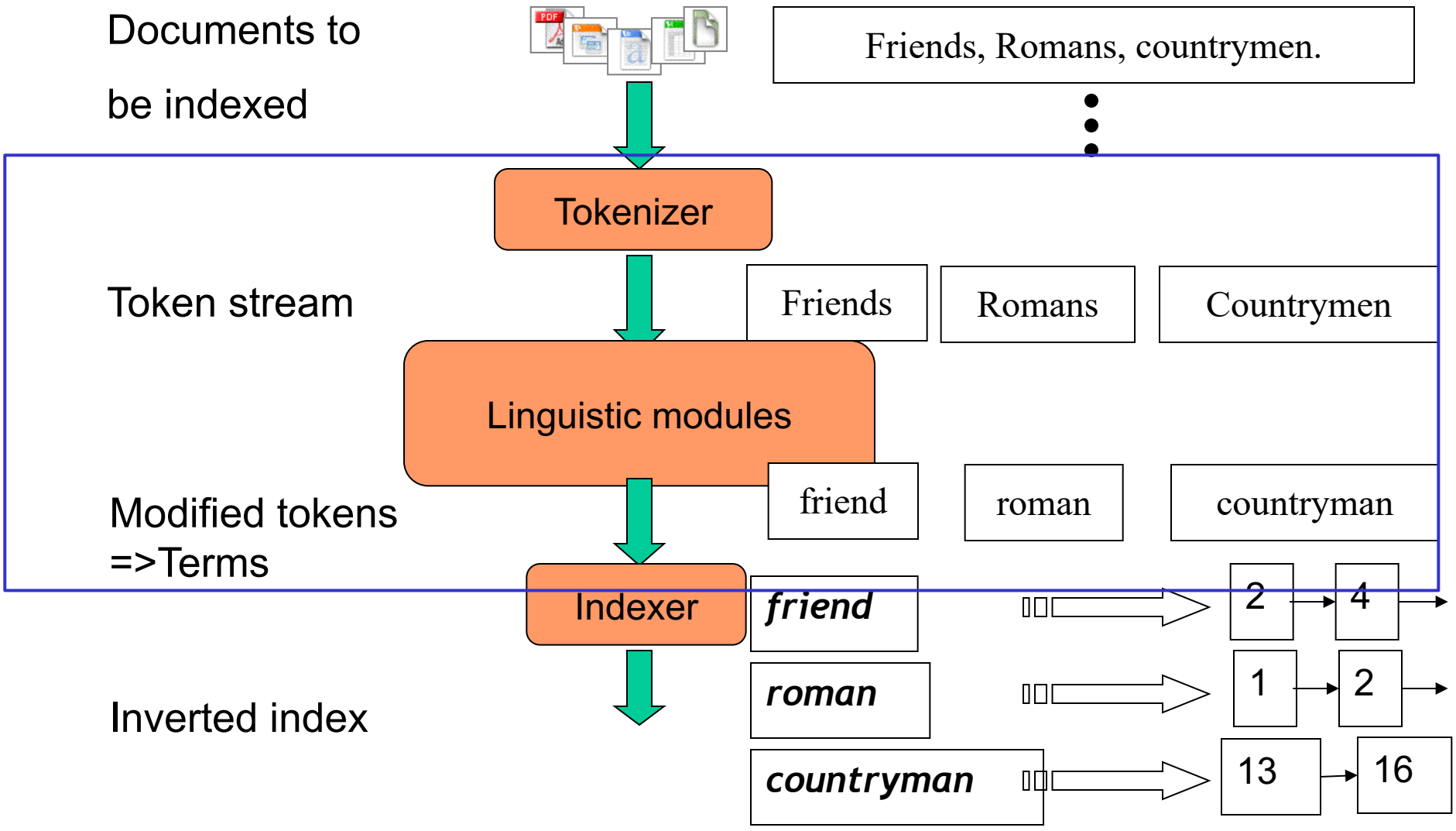


Sorted by docID (more later on why).

Inverted index construction



Inverted index construction



Initial stages of text processing

- Tokenization
  - Cut character sequence into word tokens
    - **Deal with *"John's"*, *a state-of-the-art solution***
- Normalization
  - Map text and query term to same form
    - **You want *U.S.A.* and *USA* to match**
- Stemming
  - We may wish different forms of a root to match
    - ***authorize, authorization***
- Stop words
  - We may omit very common words (or not)
    - ***the, a, to, of***

Indexer steps: Token sequence

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

Doc 1

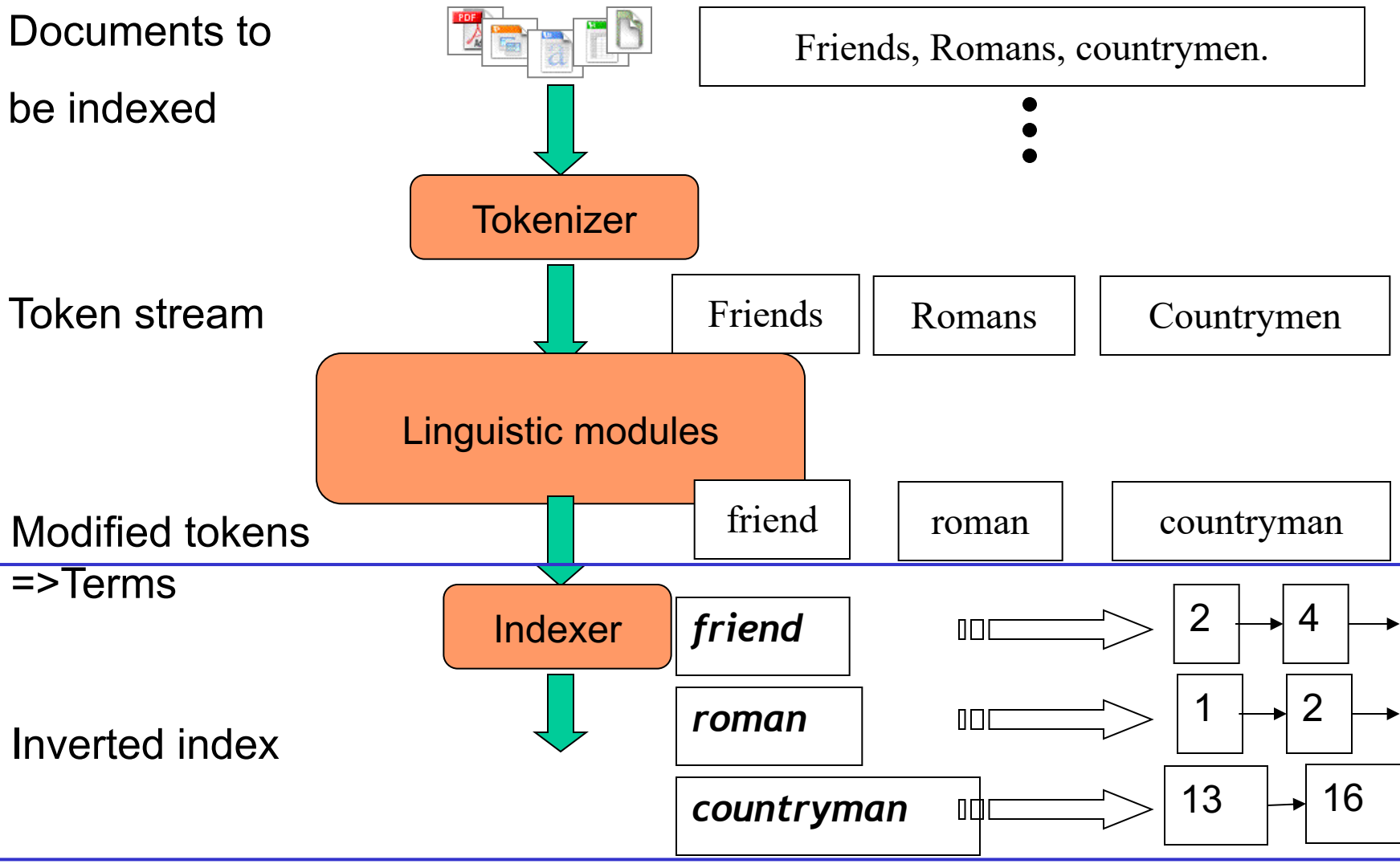
I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

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

Inverted index construction



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: 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

- Need to keep information for a given term together
- Avoid changing / rewriting index on disk
- We will see later how this is handled in practice

Indexer steps: Dictionary & Postings

- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings
- Doc. frequency information is added.

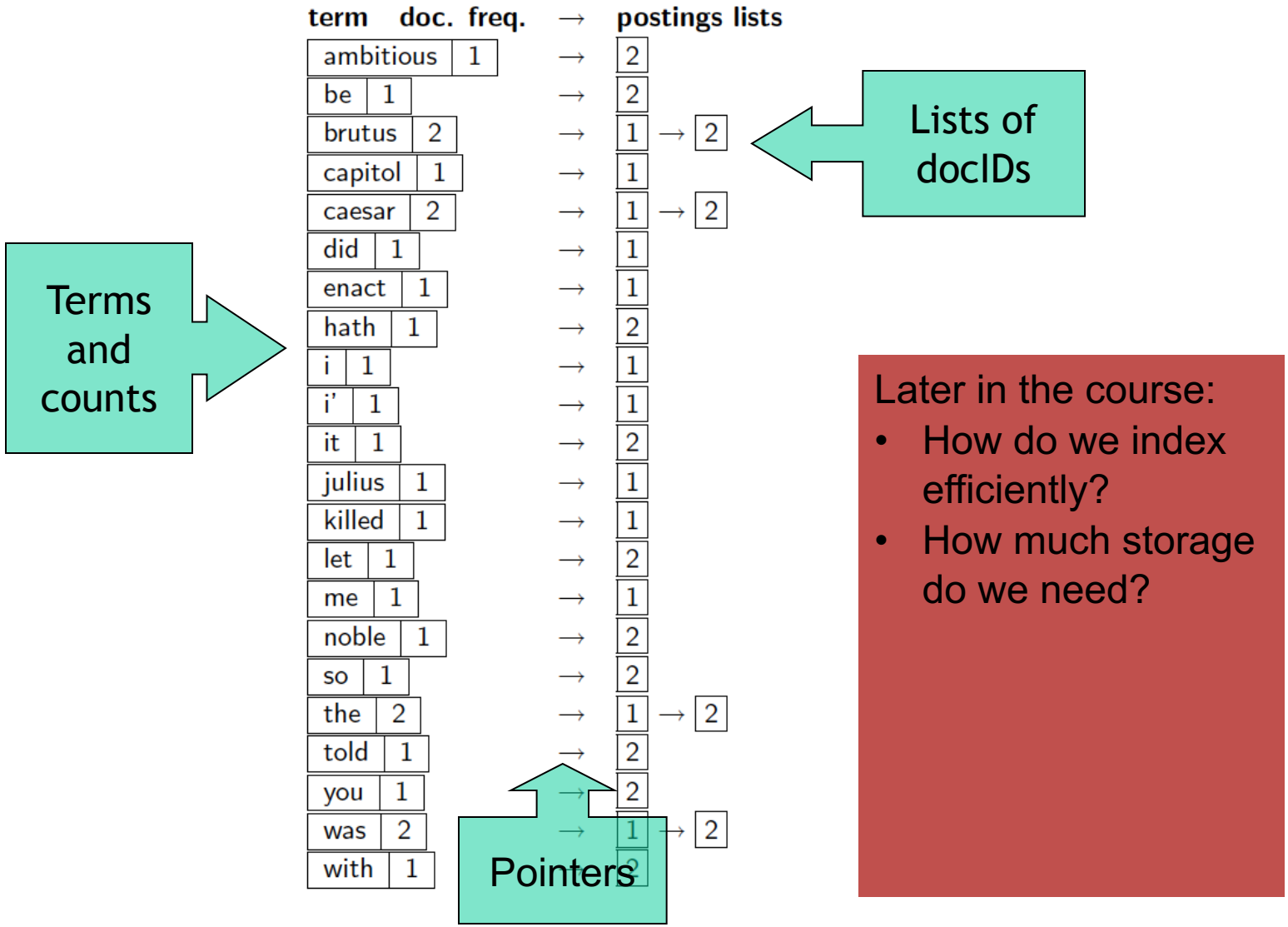
Why frequency?

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
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

Where do we pay in storage?



Later in the course:

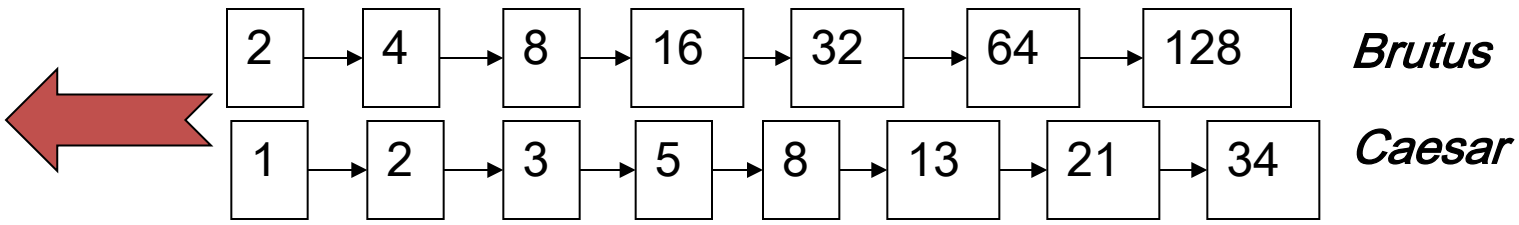
- How do we index efficiently?
- How much storage do we need?

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" the two postings:

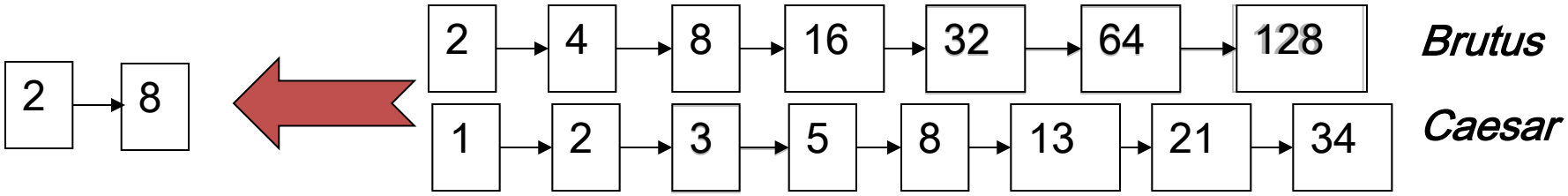


Intersecting two postings lists (a "merge" algorithm)

```
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
```

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.

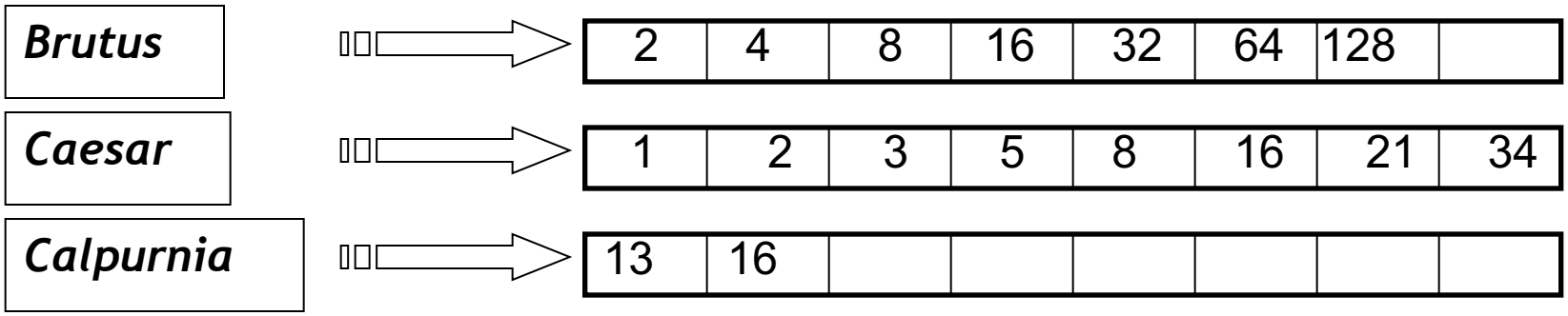
Crucial: postings sorted by docID.

Boolean queries: Exact match

- The Boolean retrieval model is being able to ask a query that is a Boolean expression:
  - Boolean Queries are queries using AND, OR and NOT to join query terms
    - **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:
  - Email, library catalog, Mac OS X Spotlight

Query optimization

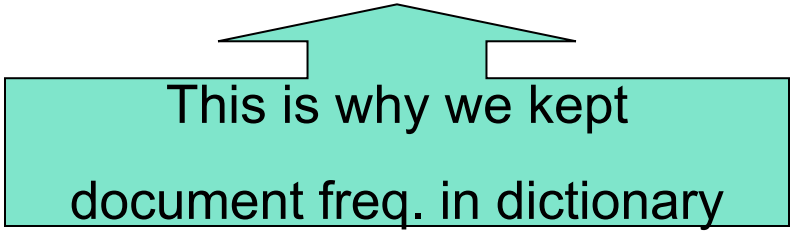
- What is the best order for query processing?
- Consider a query that is an AND of n terms.
- For each of the n terms, get its postings, then AND them together.

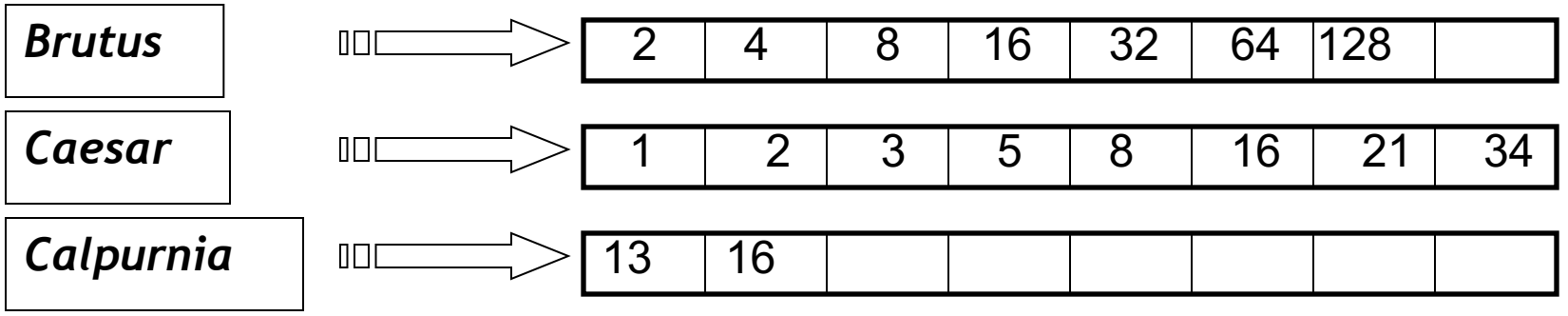


Query: ***Brutus*** *AND* ***Calpurnia*** *AND* ***Caesar***

Query optimization example

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





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

More general optimization

- e.g., *(**madding** OR **crowd**) AND (**ignoble** OR **strife**)*
- Get doc. freq.'s 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.

Exercise

- Recommend a query processing order for

*(**tangerine** OR **trees**) AND*

*(**marmalade** OR **skies**) AND*

*(**kaleidoscope** OR **eyes**)*

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

Query processing exercises

- Exercise: If the query is ***friends*** *AND* ***romans*** *AND (NOT* ***countrymen****)*, how could we use the freq of ***countrymen***?
- Exercise: Extend the merge to an arbitrary Boolean query. Can we always guarantee execution in time linear in the total postings size?
- Hint: Begin with the case of a Boolean *formula* query: in this, each query term appears only once in the query.

This lesson

- Boolean indexing and search
- Term-document matrix
- Inverted index
- Inverted index construction

Next lessons

- Handling phrases
- Index construction