Information Retrieval

IR basics

Boolean search, Inverted index

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?

  - grep Brutus shakespeare.txt 0.123s
  - grep Caesar shakespeare.txt 0.112s
  - grep -v Calpurnia shakespeare.txt 0.184s

  - wc shakespeare.txt
  - 124456 901325 5458199 shakespeare.txt

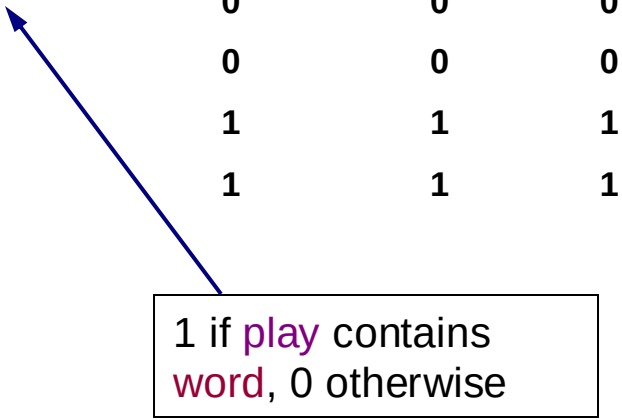
IR in 1680

- 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) are not feasible
  - Can't do 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 the query?
  - **Brutus AND Caesar but NOT Calpurnia**

- Bitwise AND OF the vectors for Brutus (110100), Caesar (110111), and Calpurnia (complemented=101111)
  - 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

110100 AND 110111 AND 101111 = 100100.

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 in 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-billion 0's and 1's.
- But it has no more than one thousand million 1's.

Can't build the matrix

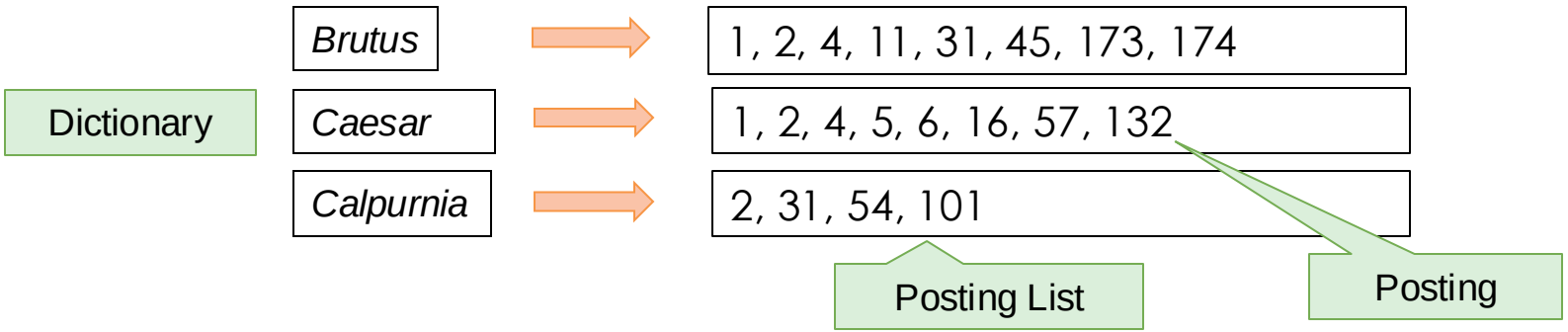
- 500K x 1M matrix has half-a-trillion 0's and 1's.
- But it has no more than one thousand million 1's.

- The matrix is extremely sparse! what's a better representation?
  - Record only the "1" positions.

Inverted index

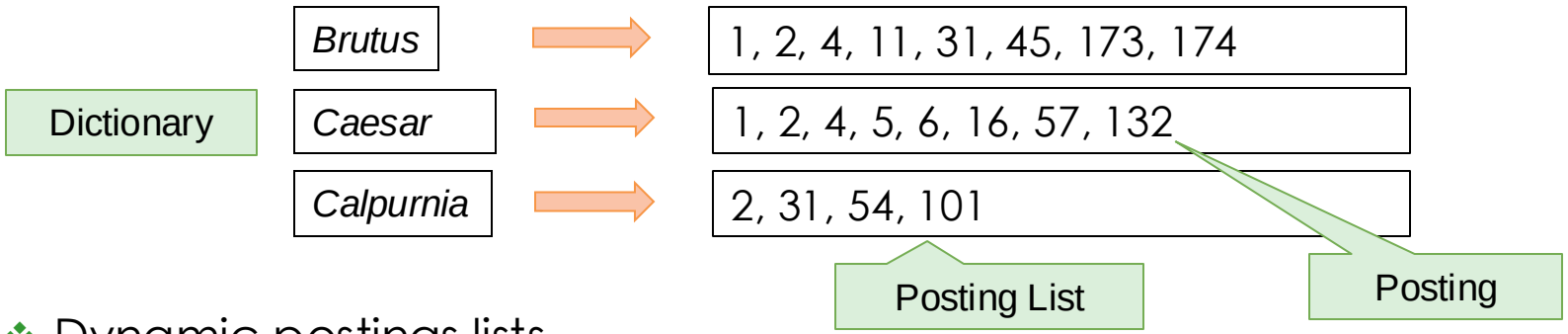
Inverted index

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



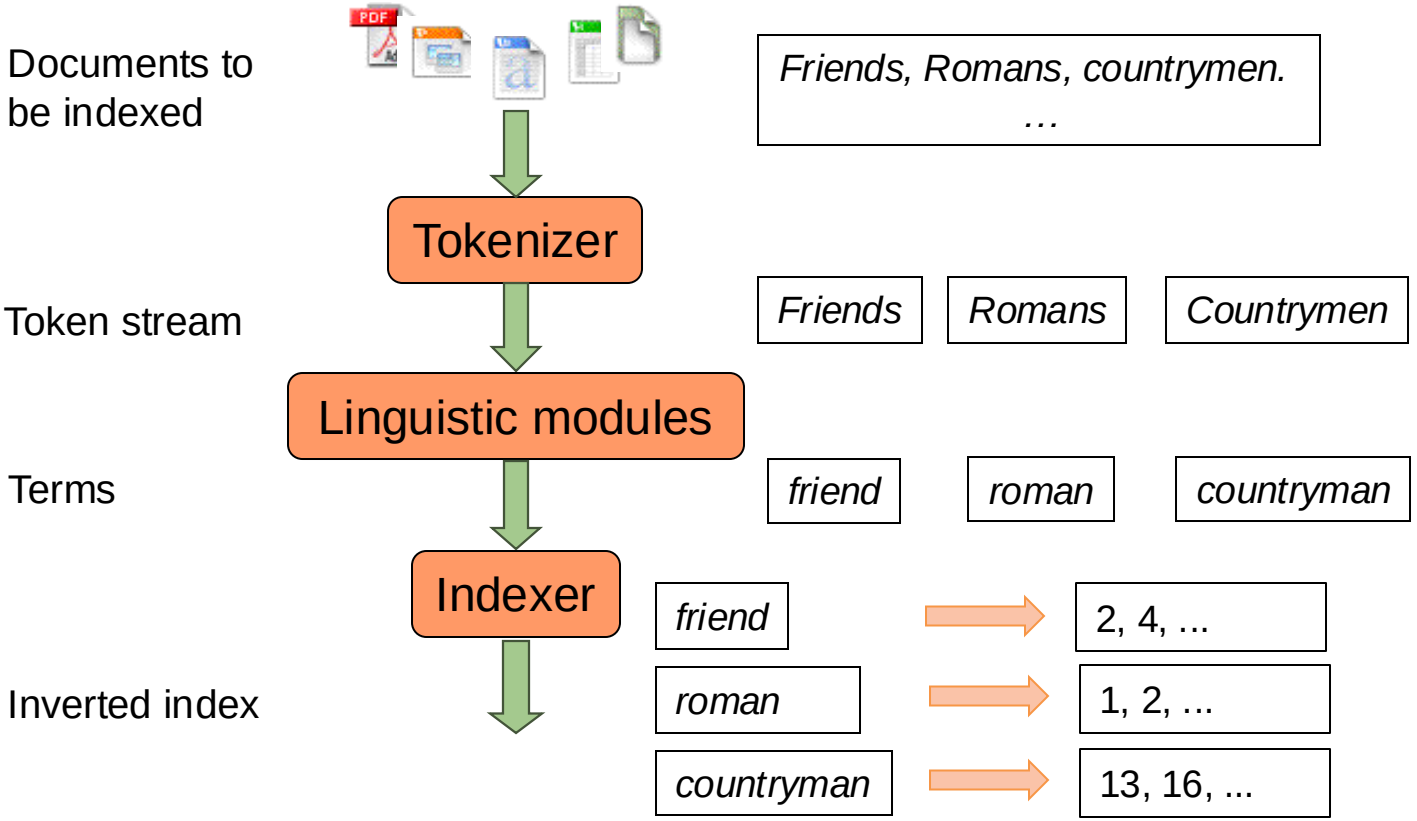
Inverted index

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

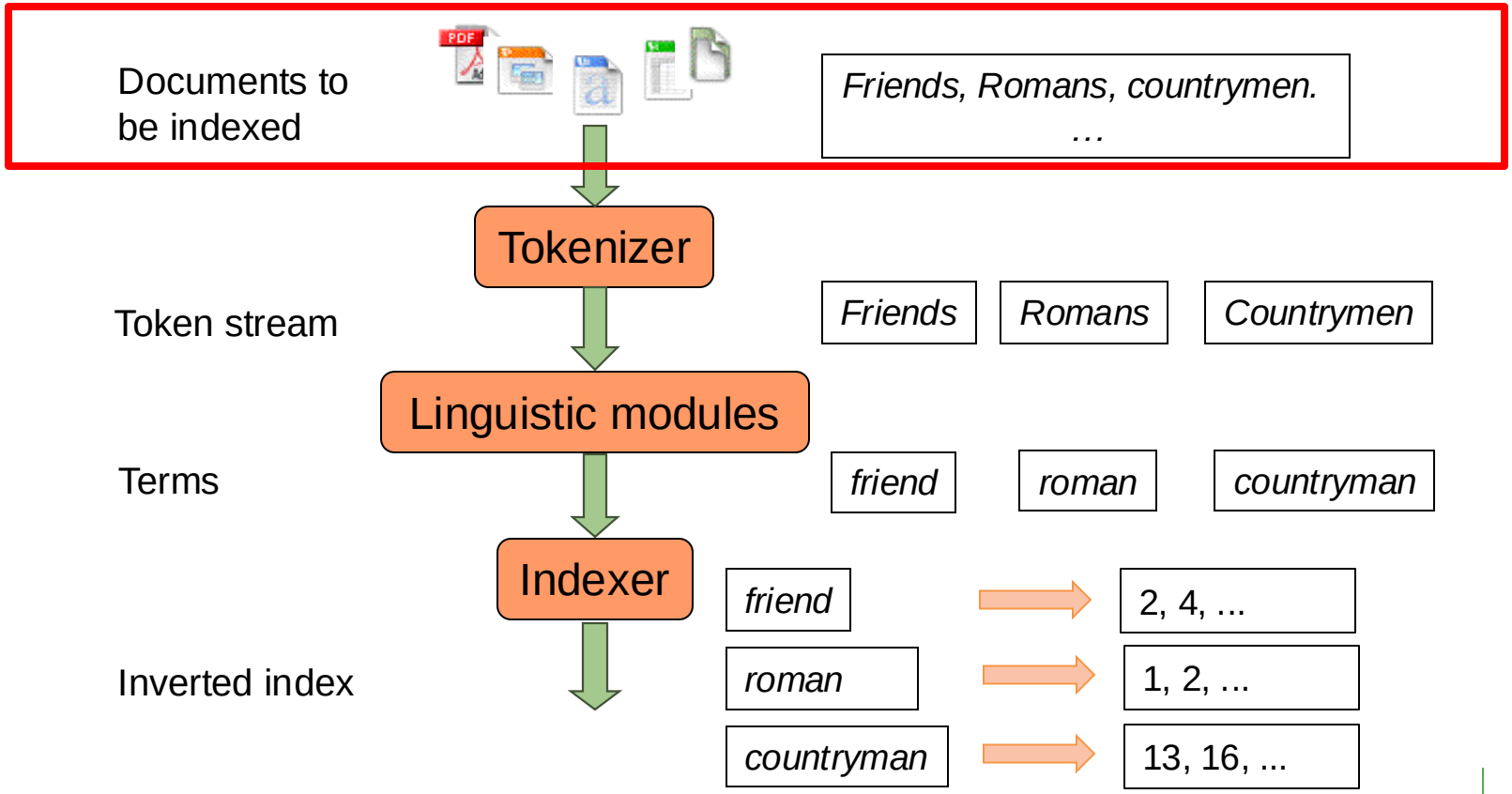


- Dynamic postings lists
  - On disk, a continuous run of postings
    - E.g., comma-separated in a text file
  - In memory, linked lists or variable-length arrays
    - Some tradeoffs in size/ease of insertion

Inverted index construction



Inverted index construction

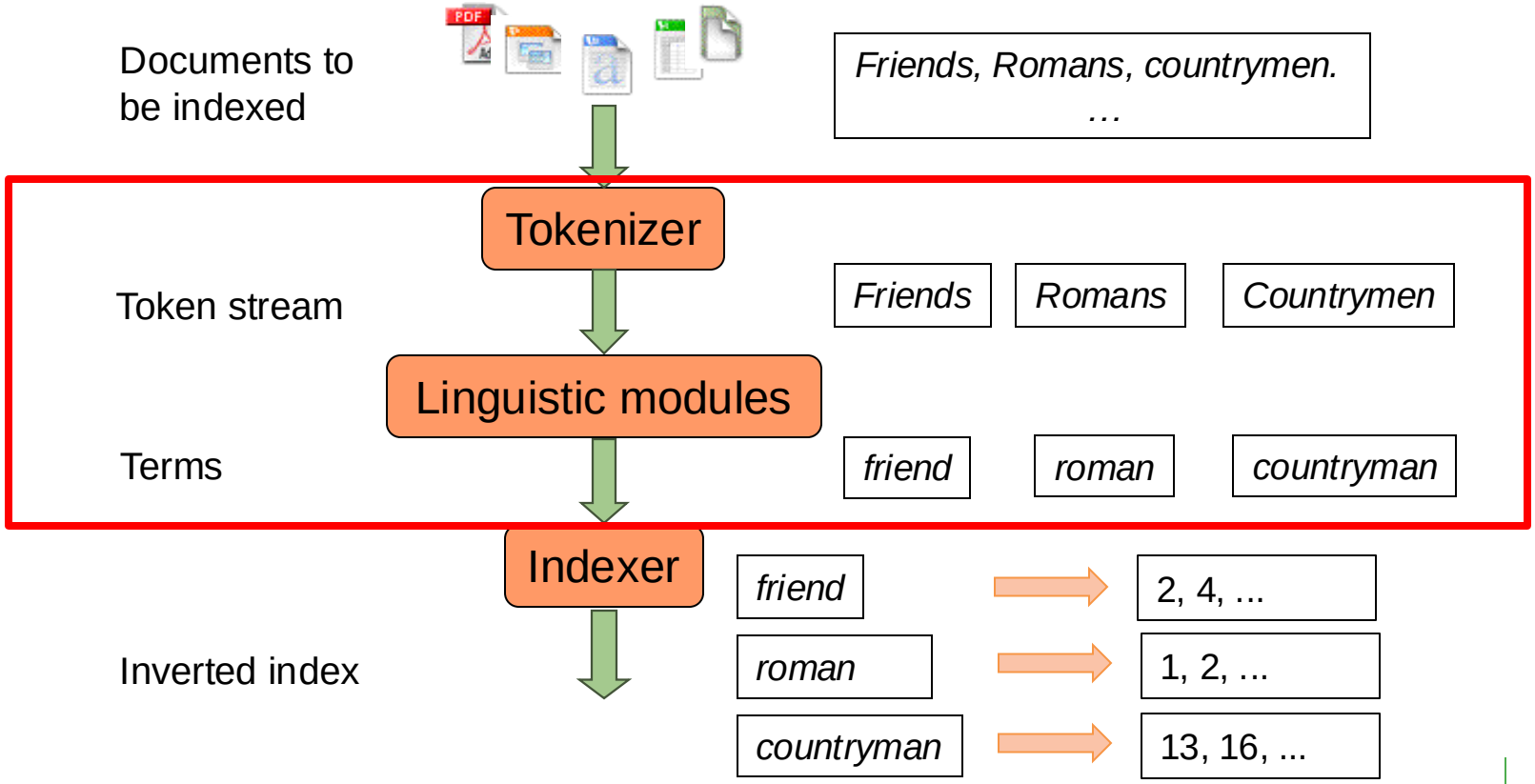


Documents

- What is a unit document?
  - A file?
  - Documents within a file? (e.g. XML, CSV)
  - An email? (Perhaps one of many in a single mbox file)
    - What about an email with 5 attachments?
  - A group of files (e.g., PPT or LaTeX split over HTML pages)

- What format is it in?
  - pdf/word/excel/html?
- What language is it in?
- What character set is in use?

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

idea

- Replace next slides by naïve algorithm
- Discuss limitations
- Present alternative

Indexer steps: Token sequence

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

Doc 1

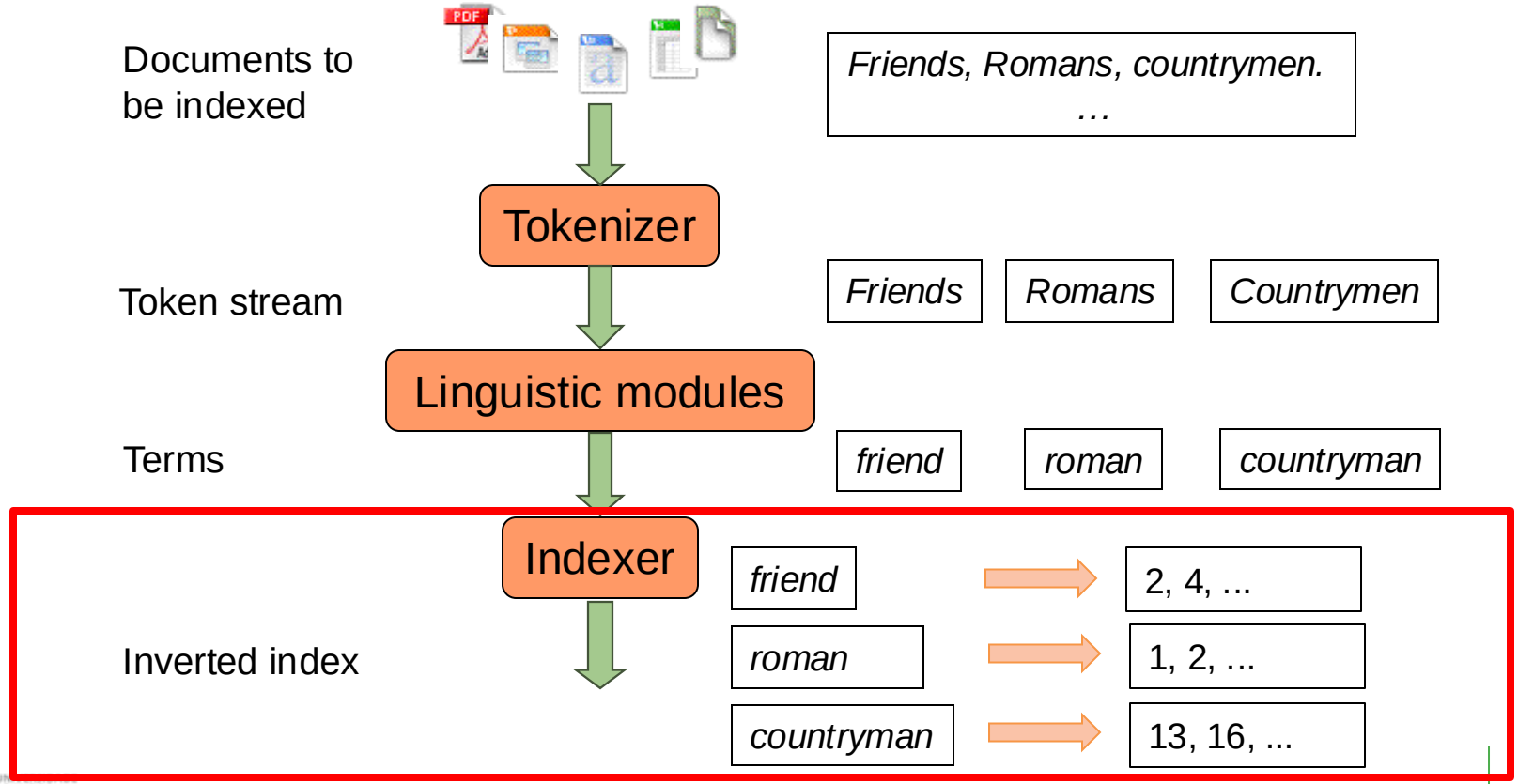
I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

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

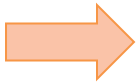


Indexer steps: Sort

- 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

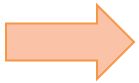
Indexer steps: Sort

- Sort by terms
  - And then docID

Core indexing step

- Need to keep information for a given term together
- Avoid changing / rewriting index on disk

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



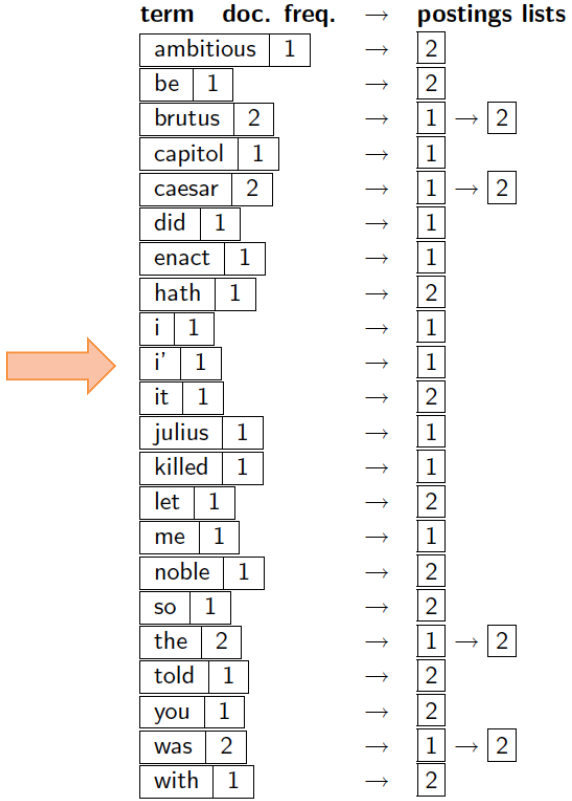
Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Indexer steps: Dictionary & Postings

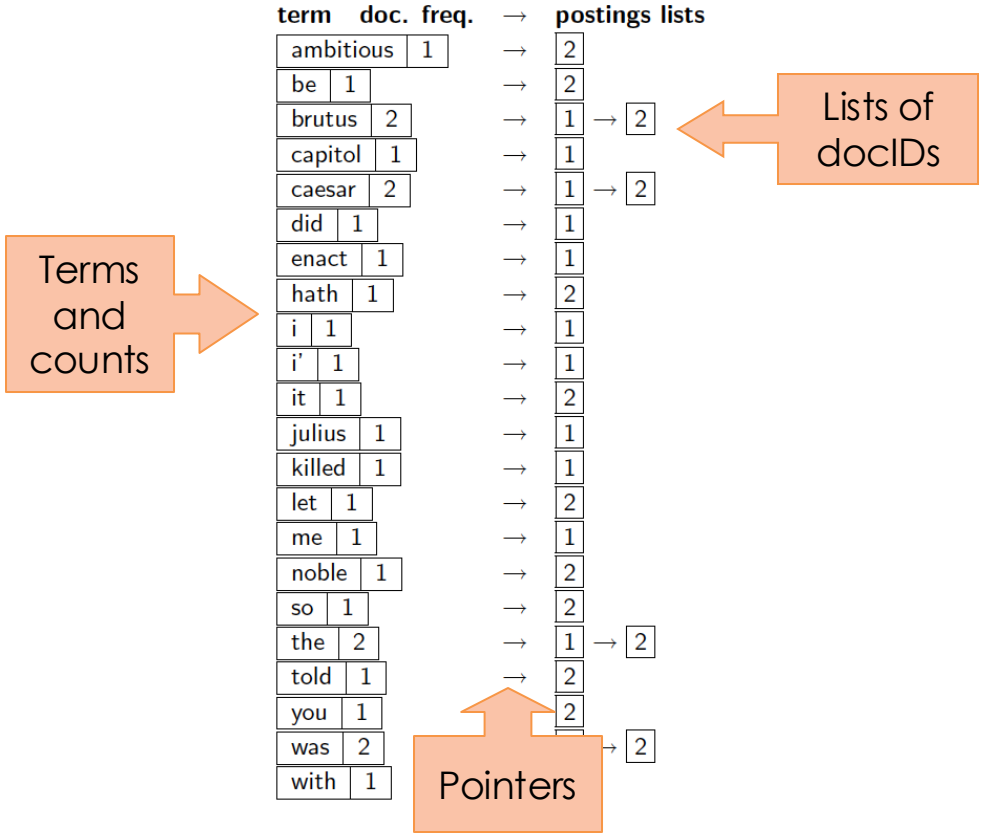
- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings
- Doc. frequency information is added.

Why frequency?

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

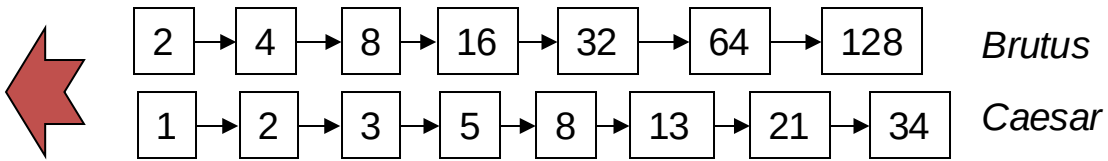


Where do we pay in storage?



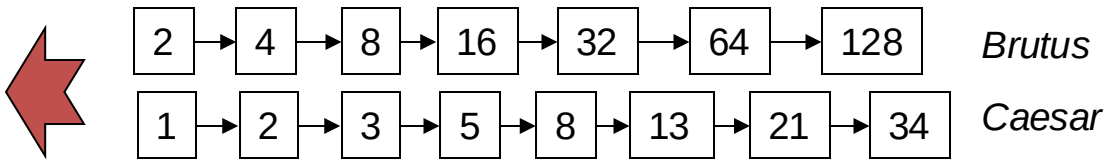
Query processing: AND

- Consider processing the query:
  - Brutus AND Caesar
- Locate Brutus in the Dictionary and retrieve its postings.
- Locate Caesar in the Dictionary and retrieve its postings.
- "Merge" the two postings:



Query processing: AND

- Consider processing the query:
  - Brutus AND Caesar
- Locate Brutus in the Dictionary and retrieve its postings.
- Locate Caesar in the Dictionary and retrieve its postings.
- "Merge" the two postings:



- Walk through the two postings simultaneously, in time linear in the total number of posting entries
- If the list lengths are x and y, the merge takes O(x+y) operations.
  - Crucial: postings sorted by docID.

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
```

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
- Primary commercial retrieval tool
  - Email, library catalog, Mac OS X Spotlight

Boolean queries: More general merges

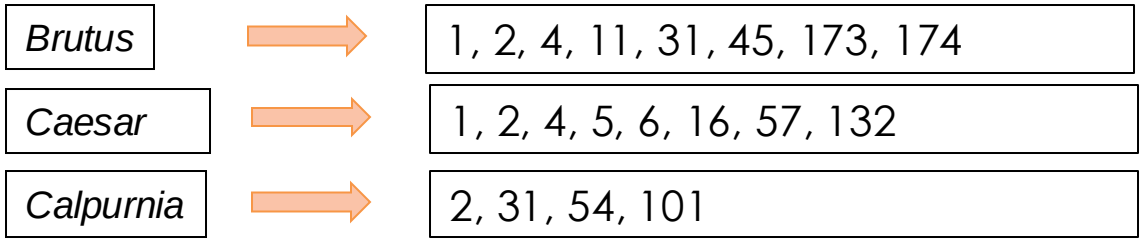
- Exercise:
  - Adapt the merge for the queries:
    - Brutus AND NOT Caesar
    - Brutus OR NOT Caesar
  - Can we still run through the merge in time O(x+y)?
  - What can we achieve?

Merging

- What about an arbitrary Boolean formula?
  - (Brutus OR Caesar) AND NOT
  - (Antony OR Cleopatra)
- Can we always merge in "linear" time?
  - Linear in what?
- Can we do better?

Query optimization

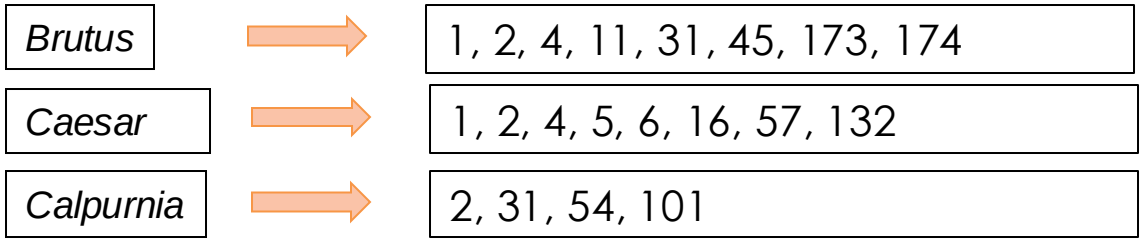
- What is the best order for query processing?
- Consider a query that is an AND of n terms.
  - ***Brutus*** *AND* ***Calpurnia*** *AND* ***Caesar***
- For each of the n terms, get its postings, then AND them together.



32

Query optimization

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



- Process in order of increasing frequency:
  - start with the smallest set, then keep cutting further.
    - This is why we kept document freq. in the dictionary

More general optimization

- e.g., (madding OR crowd) AND (ignoble OR strife)
  - Get documents' frequencies (DF) for all terms.
  - Estimate the size of each OR by the sum of its DF(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

This lesson

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