Text Mining and Information Retrieval

Laura Brown

Some slides adapted from P. Smyth; Han, Kamber, & Pei;
Tan, Steinbach, & Kumar; C. Volinsky; R. Tibshirani; D. Kauchak
and http://nlp.stanford.edu/IR-book/

Outline

- Information Retrieval
  - What is it?
  - Challenges
- Represent Text Data
  - Term-document incidence matrices
  - Inverted index
- Boolean Queries
  - Query processing
  - Query optimization

Information Retrieval

- What comes to mind when I say “information retrieval”?

- Where have you seen IR? What are some real-world examples/uses?
    - Search engines (web search)
    - File search (e.g. OS X Spotlight, Windows Instant Search, Google Desktop)
    - Databases?
    - Catalog search (e.g. library)
    - Intranet search (i.e. corporate networks)

Information Retrieval

- Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

Information Retrieval

- Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).
    - Find all documents about computer science
    - Find all course web pages at Michigan Tech
    - What is the cheapest flight from LA to NY?
    - Who was the 15th president?

Information Retrieval

- Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

- What is the difference between an *information need* and a *query*?

Information Retrieval

- Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

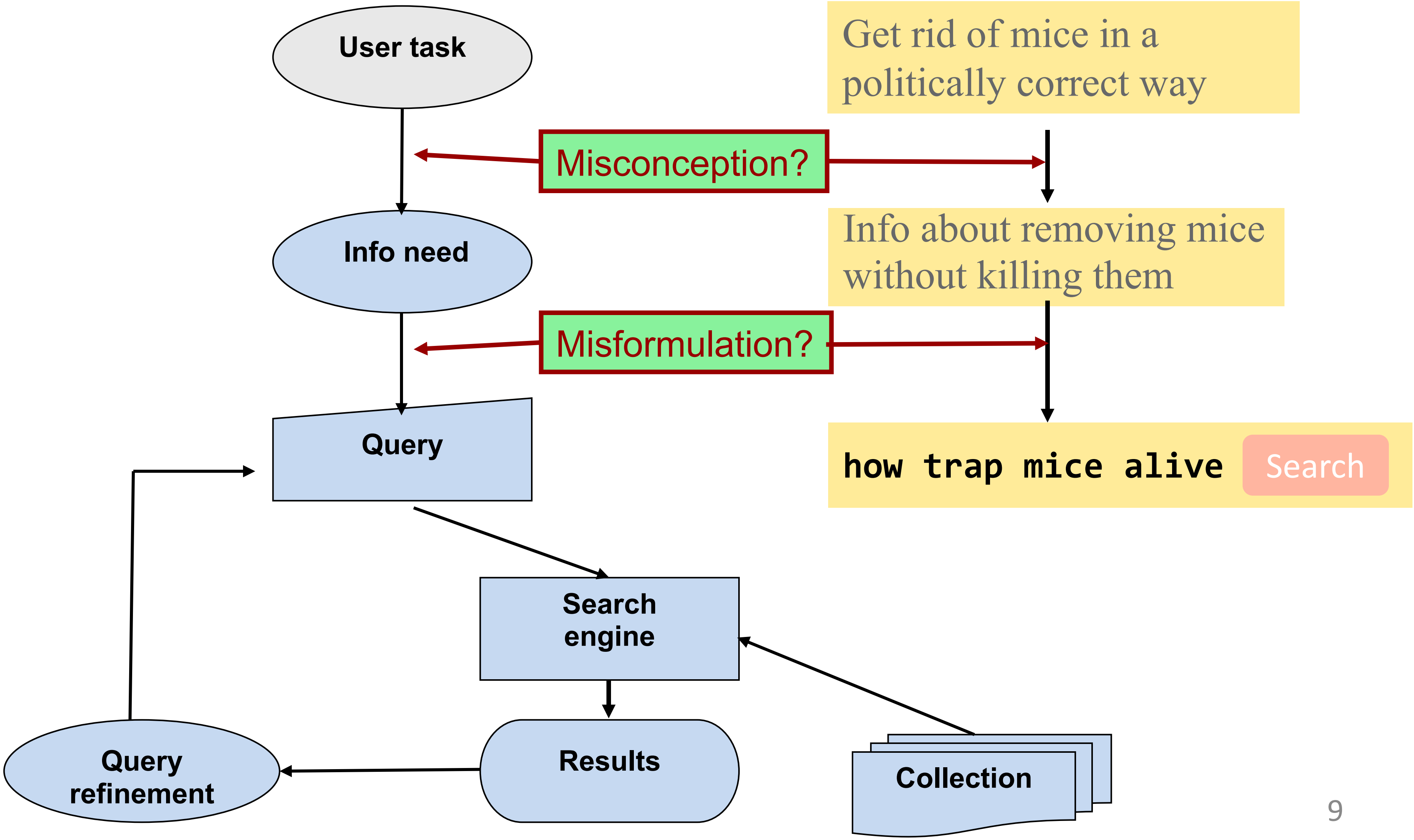
Information need	Query
• Find all documents about computer science	“computer science”
• Find all course web pages at Michigan Tech	Michigan Tech AND college AND *url-contains* class
• Who is was the 15th president?	WHO=president NUMBER=15

Assumptions of Information Retrieval

- Collection: A set of documents
  - Assume it is a static collection for the moment

- Goal: Retrieve documents with information that is relevant to the user’s information need and helps the user complete a task

Classic Search Model



User task

Info need

Query

Search engine

Results

Query refinement

Collection

Misconception?

Misformulation?

Get rid of mice in a politically correct way

Info about removing mice without killing them

how trap mice alive

Search

How good are the retrieved docs?

- *Precision* : Fraction of retrieved docs that are relevant to the user's information need
- *Recall* : Fraction of relevant docs in collection that are retrieved

  - More precise definitions and measurements to follow later

Information Retrieval Challenges

- Main problems with text retrieval:
  - What does relevant mean?
  - How do you know if you have the right documents?
  - How can user feedback be incorporated?

Term-document incidence matrices

Introduction to Information Retrieval

Unstructured data in 1620

- 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 this not the answer?

Unstructured data in 1620

- 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 this 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 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

Brutus *AND* ***Caesar*** *BUT NOT* ***Calpurnia***

1 if play contains word, 0 otherwise

Incidence vectors

- For each term, we have a 0/1 vector
  - Caesar = 110111
  - Brutus = 110100
  - Calpurnia = 010000
- To answer query?

	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

- For each term, we have a 0/1 vector
  - Caesar = 110111
  - Brutus = 110100
  - Calpurnia = 010000
- To answer query? take vectors for ***Brutus, Caesar*** and ***Calpurnia*** (complemented) → bitwise *AND*.
  - Answer = 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

- Antony and Cleopatra, Act II, Scene ii

  Agrippa [Aside to DOMITIUS ENOBARNUS]: Why, Enobarbas,
  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.

Incidence vectors

- For each term, we have a 0/1 vector
  - Caesar = 110111
  - Brutus = 110100
  - Calpurnia = 010000
- Bitwise AND the vectors together using the complemented vector for all NOT queries

Any problem with this approach?

Bigger Collections

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

Bigger Collections

- 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 billion 1’s.
  - Each of the 1 million documents has at most 1000 1’s
  - Matrix is extremely sparse!

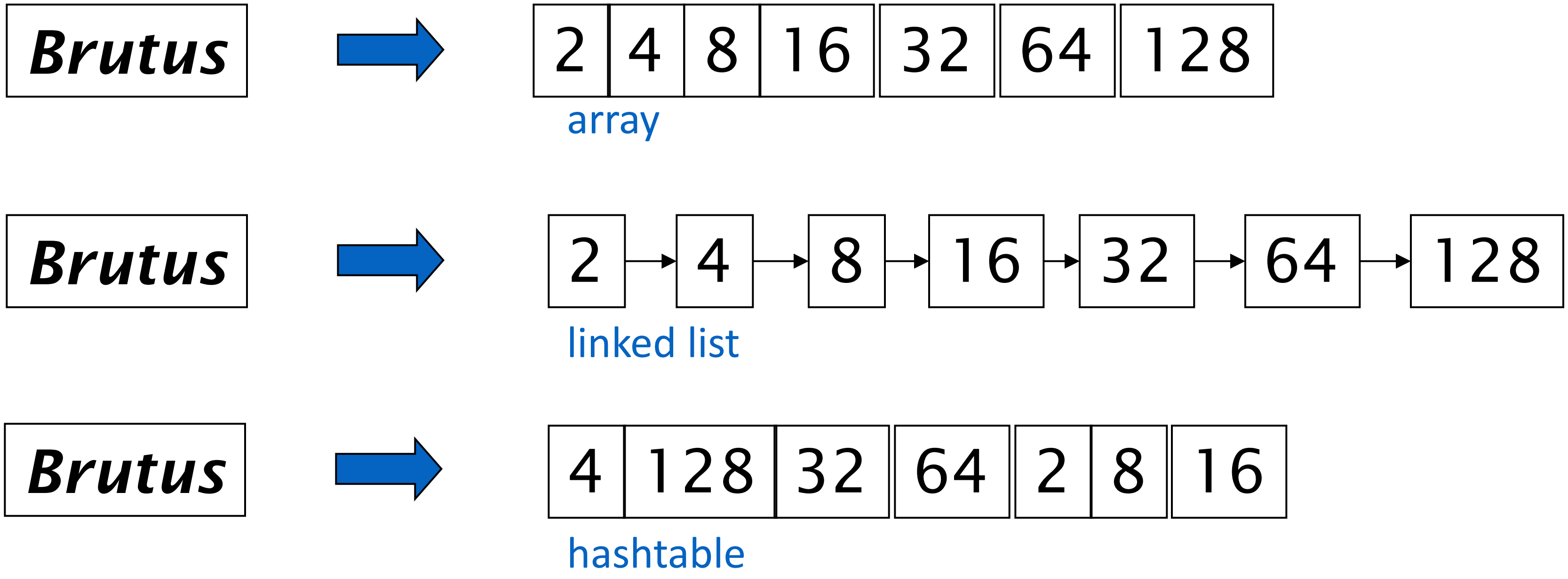
- What’s a better representation?
  - Only record the 1 positions

Inverted Index

Introduction to Information Retrieval

Inverted index

- For each term *t*, we store a list of all documents that contain *t*
  - Identify each doc by a **docID**, a document serial number
- What data structures might we use for this?



Brutus → 2 | 4 | 8 | 16 | 32 | 64 | 128

array

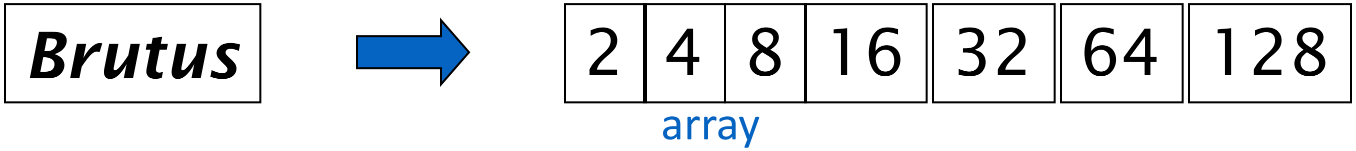
Brutus → 2 → 4 → 8 → 16 → 32 → 64 → 128

linked list

Brutus → 4 | 128 | 32 | 64 | 2 | 8 | 16

hashtable

Inverted index representation



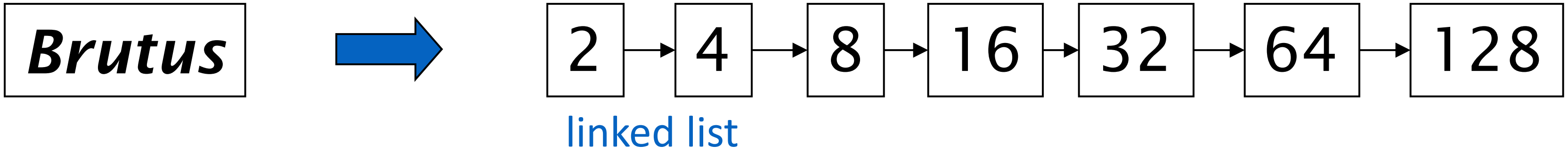
Brutus

2	4	8	16	32	64	128

array

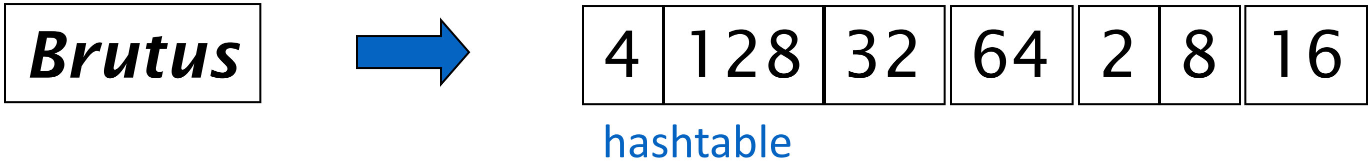
- Pros
  - Simple to implement
  - No extra pointers required for data structure
  - Contiguous memory
- Cons
  - How do we pick the size of the array?
  - What if we want to add additional documents?

Inverted index representation



- Pros
  - Dynamic space allocation
  - Insertion of new documents is straightforward
- Cons
  - Memory overhead of pointers
  - Noncontiguous memory access

Inverted index representation



Brutus

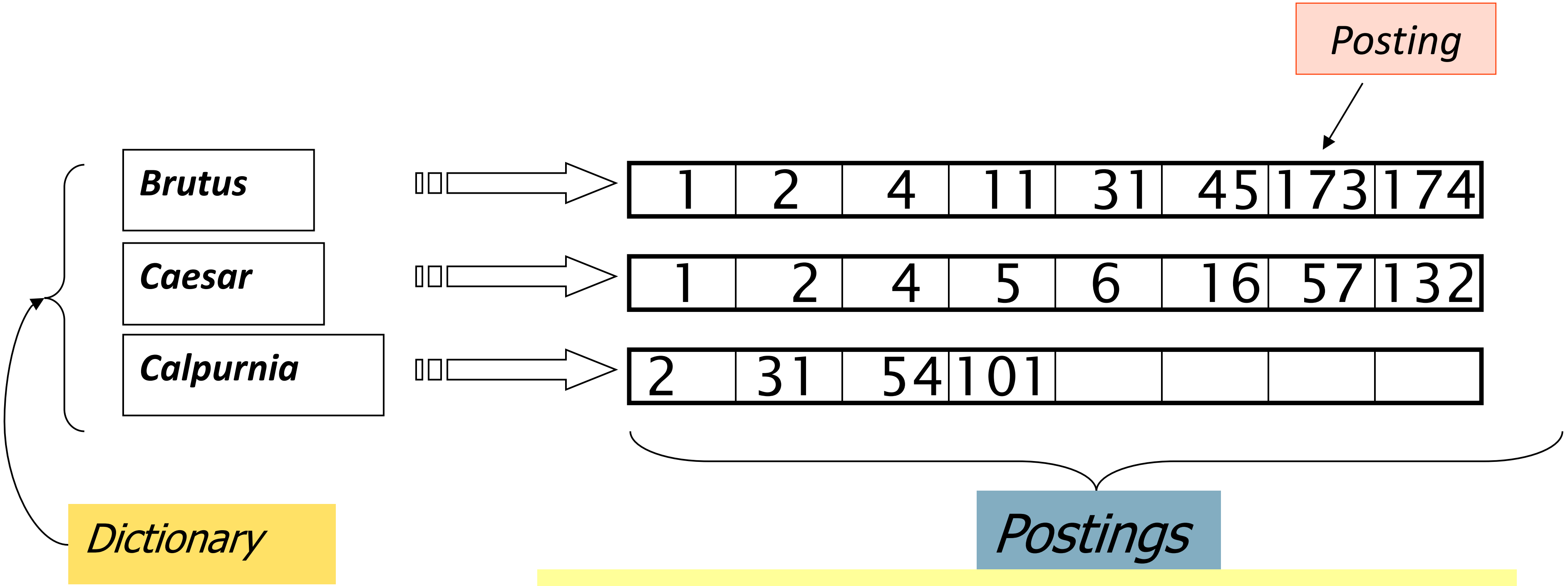
4 | 128 | 32 | 64 | 2 | 8 | 16

hashtable

- Pros
  - Search in constant time
  - Contiguous memory
- Cons
  - How do we pick the size?
  - What is we want to add additional documents?
    - May have to rehash
  - To get constant time operations, lots of unused slots/memory

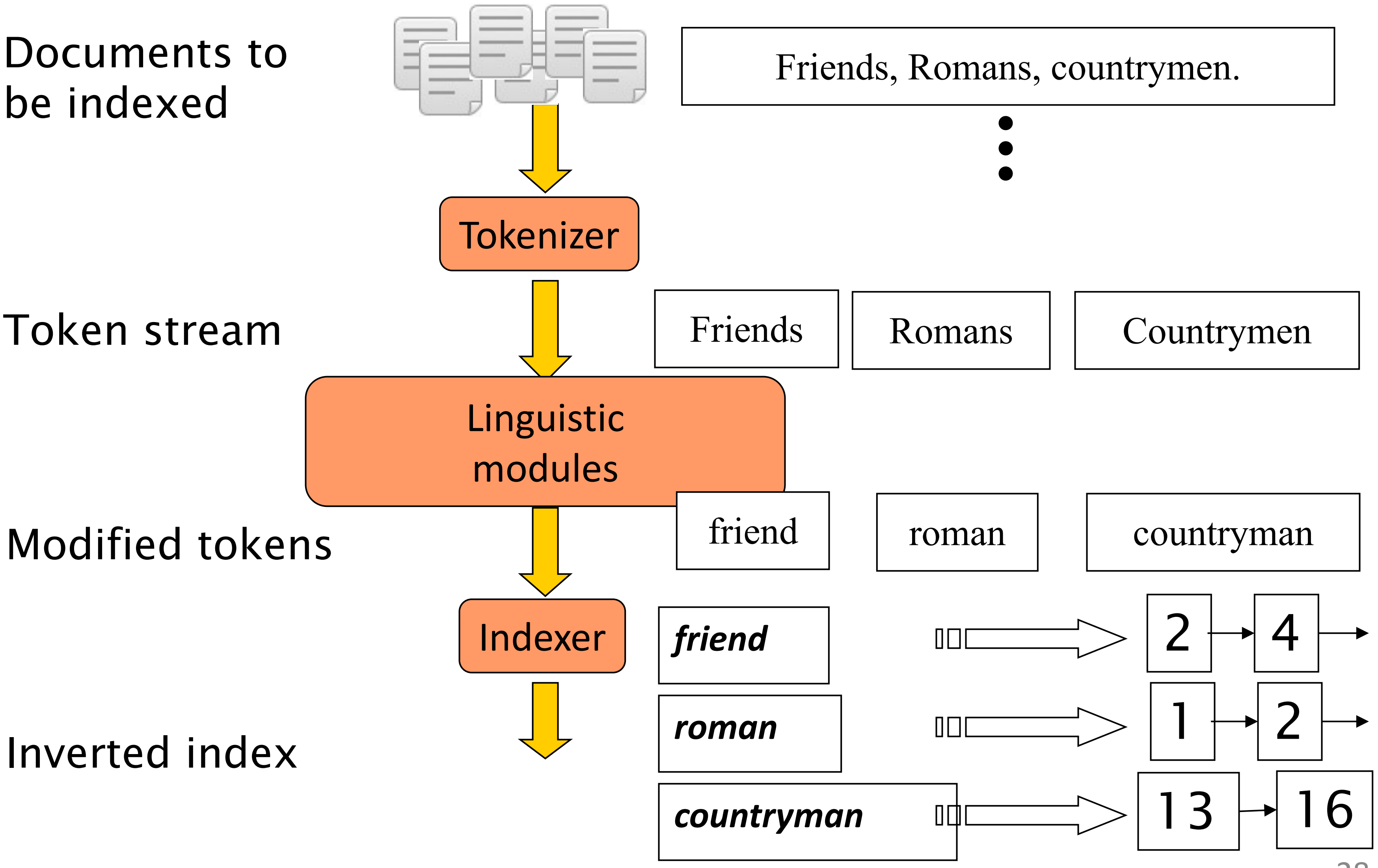
Inverted index

- We need variable-size postings lists
  - On disk, a continuous run of postings
  - In memory, can use linked lists



Sorted by docID (more later on why).

Inverted index construction



Documents to be indexed

Friends, Romans, countrymen.

⋮

Tokenizer

Token stream

Friends Romans Countrymen

Linguistic modules

Modified tokens

friend roman countryman

Indexer

Inverted index

friend → 2 → 4 →

roman → 1 → 2 →

countryman → 13 → 16

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***

We will dig into these later!

Indexer steps: Token sequence

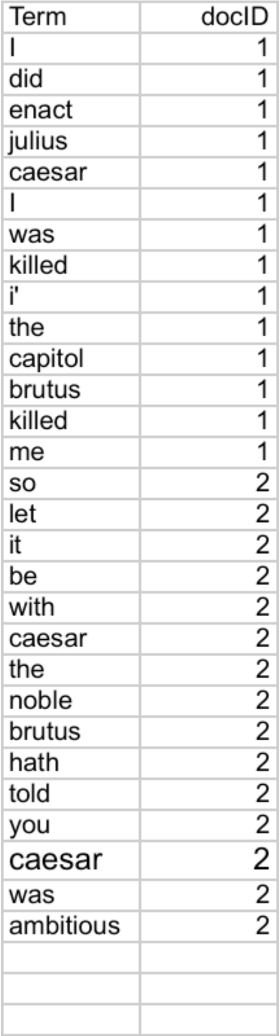
- Sequence of (Modified token, Document ID) pairs.

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

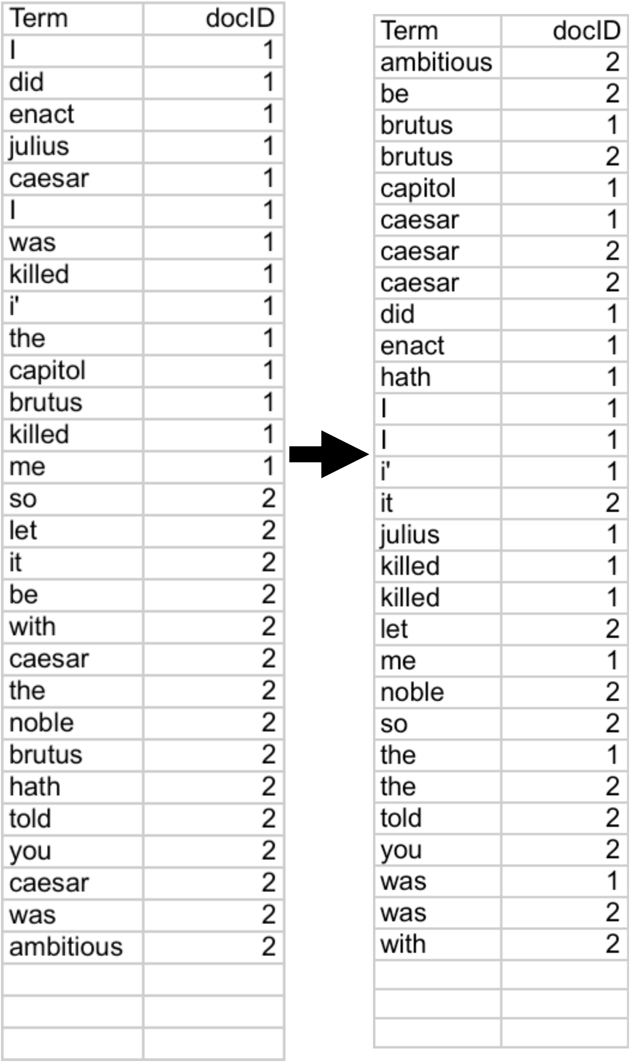


Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Indexer steps: Sort

- Sort by terms
  - At least conceptually
    - And then docID

Core indexing step

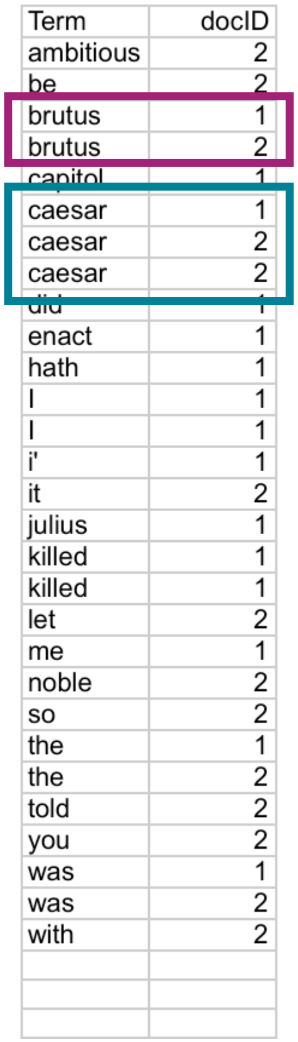


Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

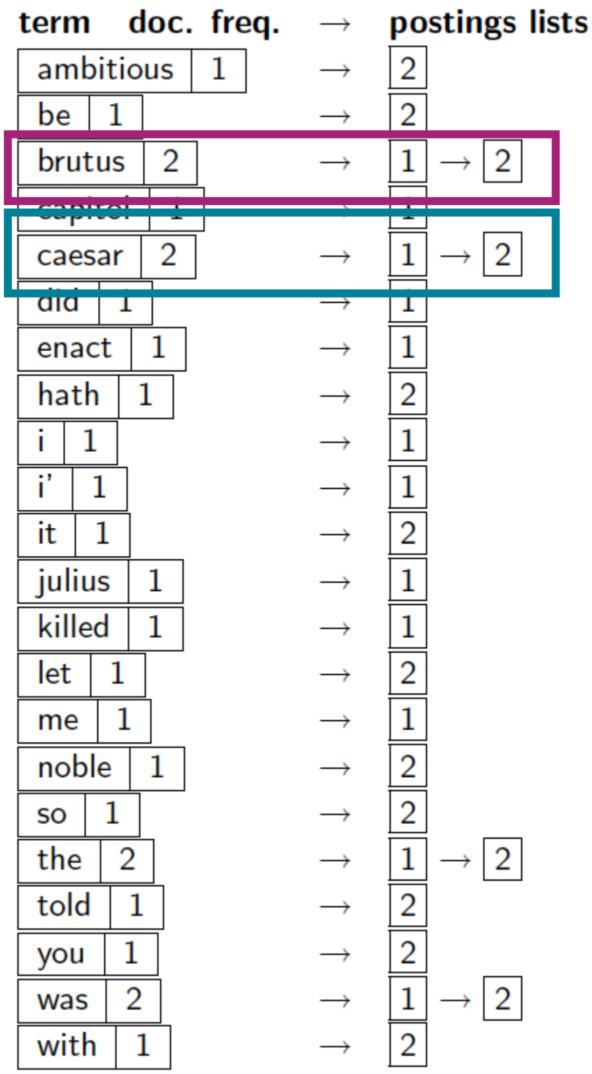
Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Indexer steps: Dictionary & Postings

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

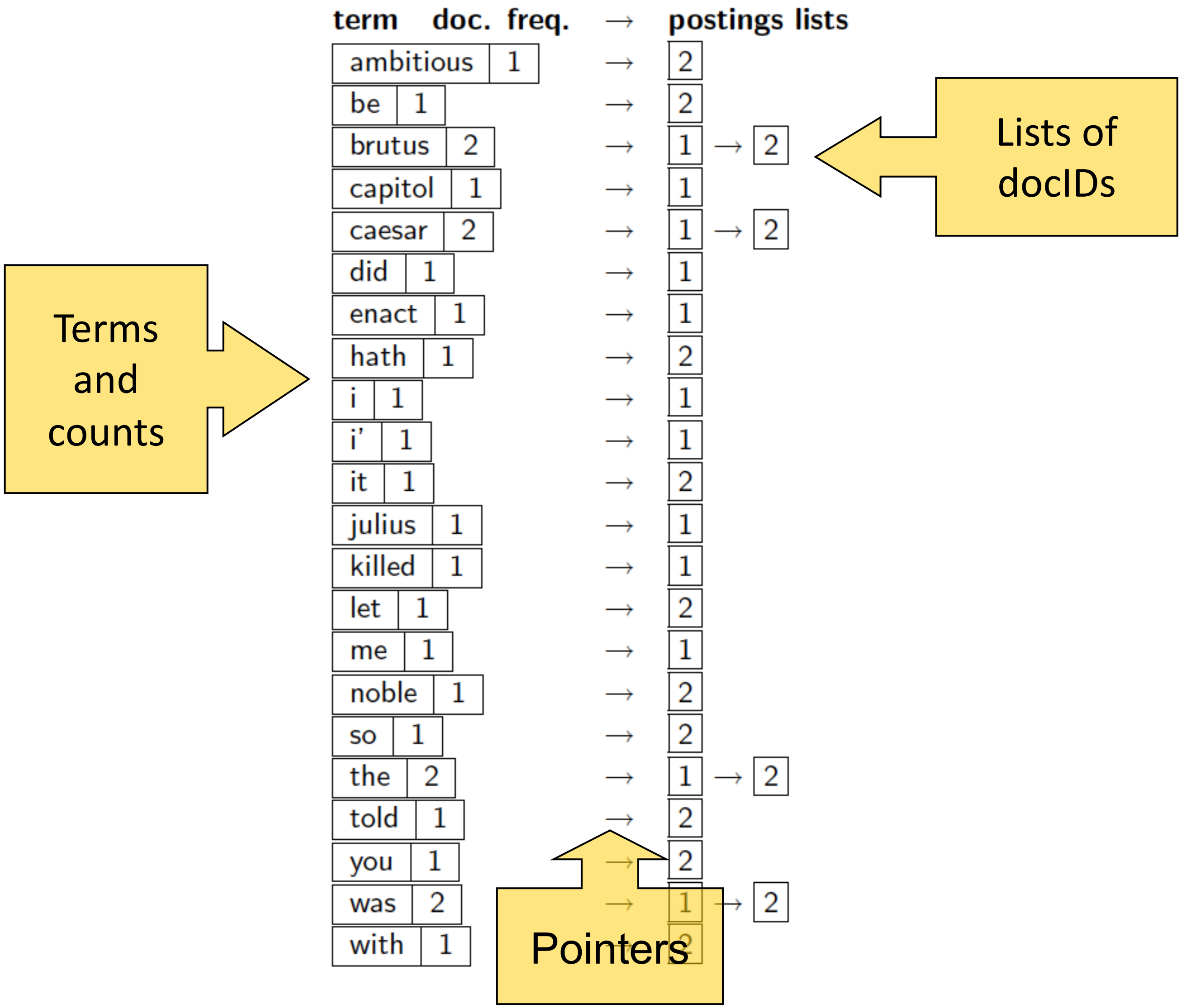


Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2



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

Indexing Storage



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

Lists of docIDs

Terms and counts

Pointers

Query processing with an inverted index

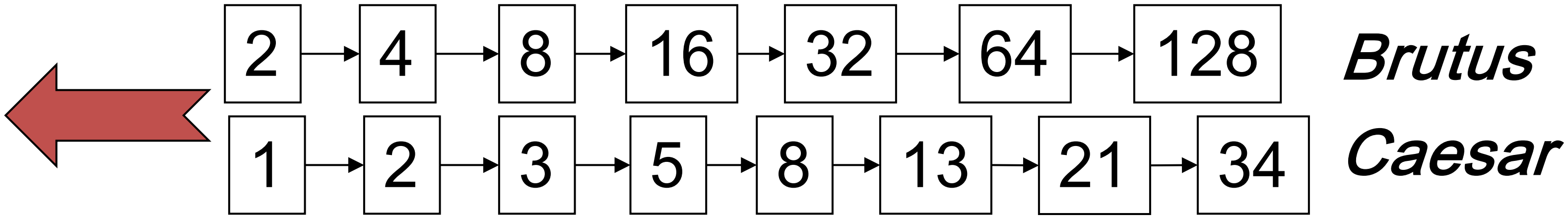
Introduction to Information Retrieval

Query processing: AND

- Consider processing the query:

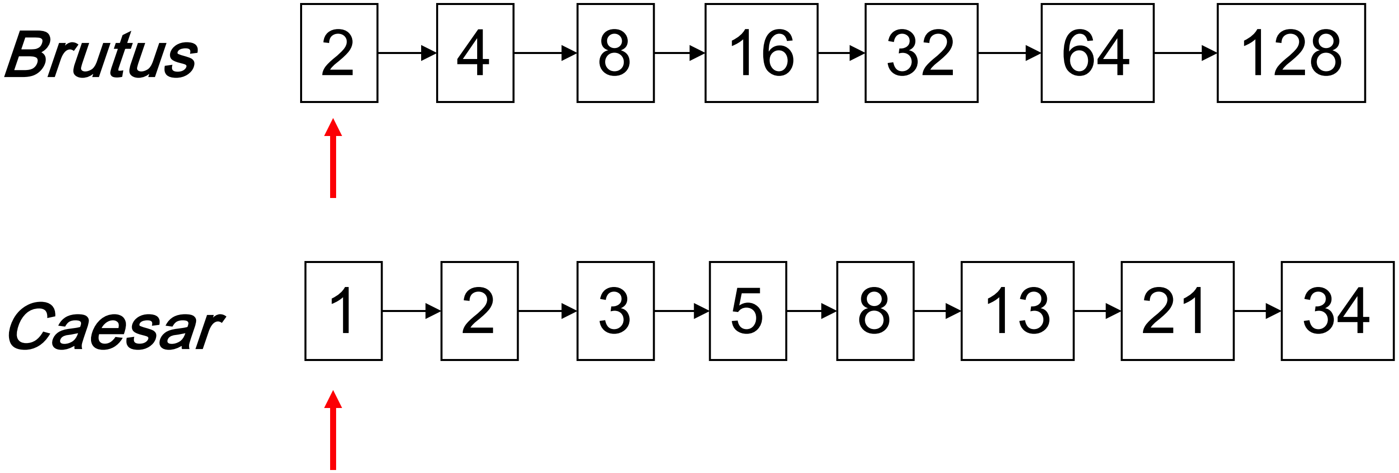
  Brutus *AND* ***Caesar***

  - Locate ***Brutus*** in the Dictionary;
    - Retrieve its postings
  - Locate ***Caesar*** in the Dictionary;
    - Retrieve its postings
  - "Merge" the two postings (intersect the document sets)



The merge

- Walk through the two postings simultaneously



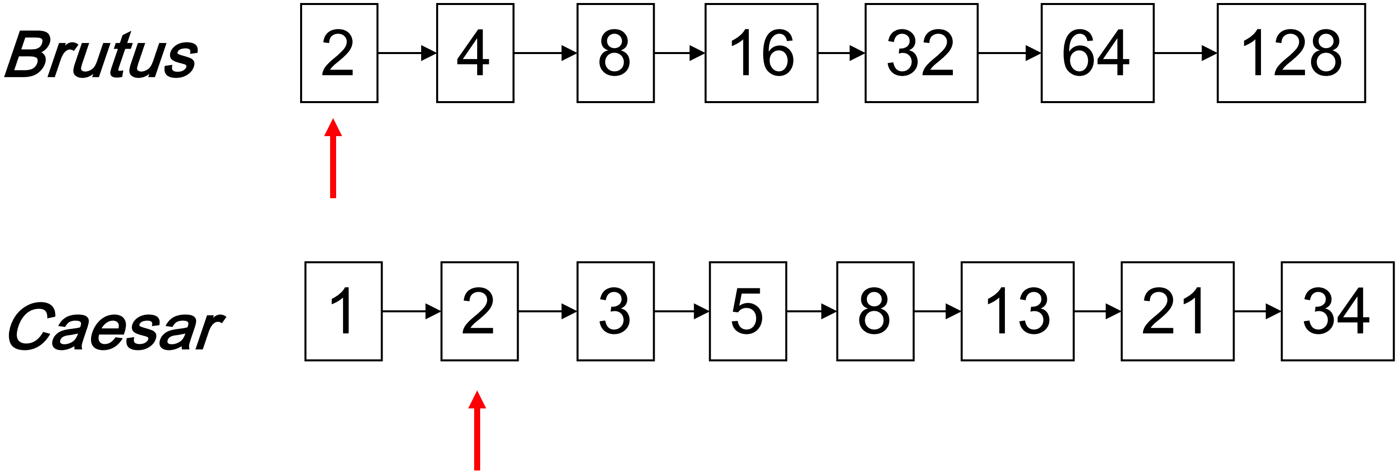
Brutus 2 → 4 → 8 → 16 → 32 → 64 → 128

Caesar 1 → 2 → 3 → 5 → 8 → 13 → 21 → 34

Brutus AND Caesar

The merge

- Walk through the two postings simultaneously



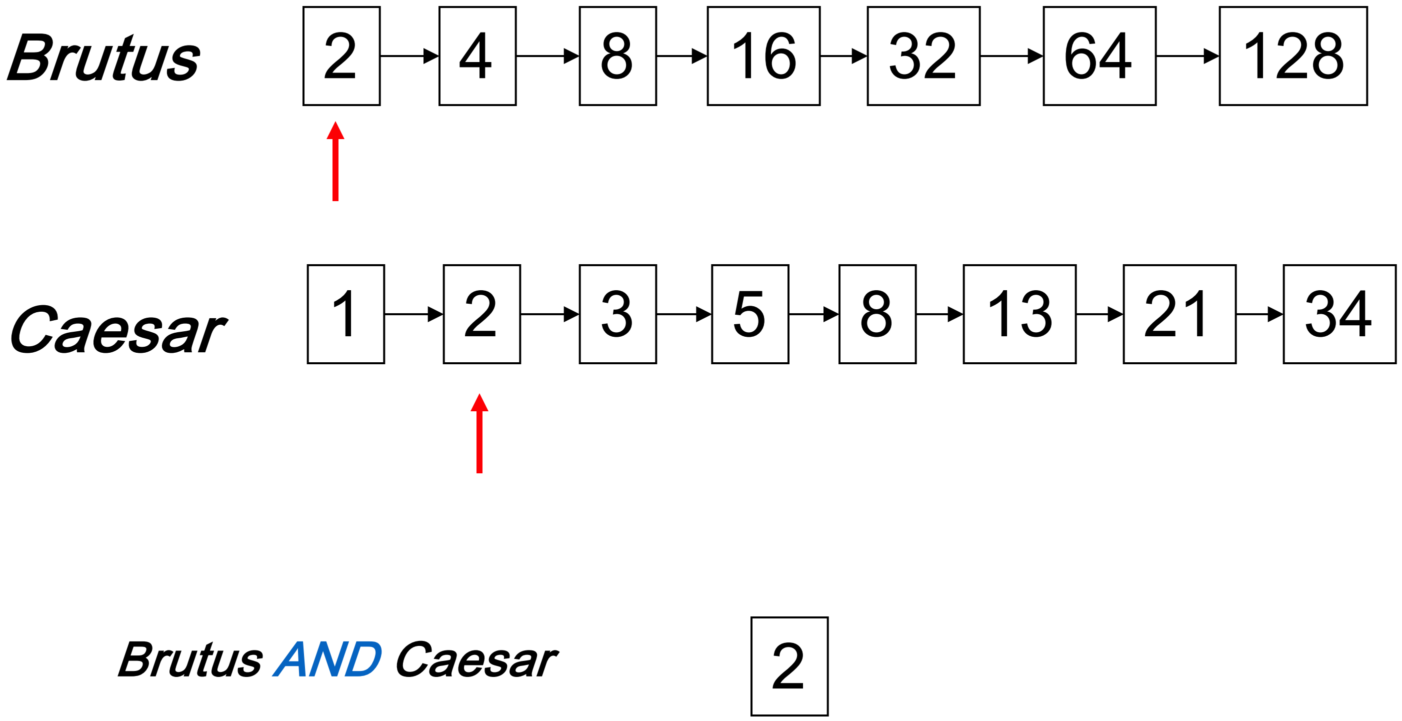
Brutus 2 → 4 → 8 → 16 → 32 → 64 → 128

Caesar 1 → 2 → 3 → 5 → 8 → 13 → 21 → 34

Brutus AND Caesar

The merge

- Walk through the two postings simultaneously



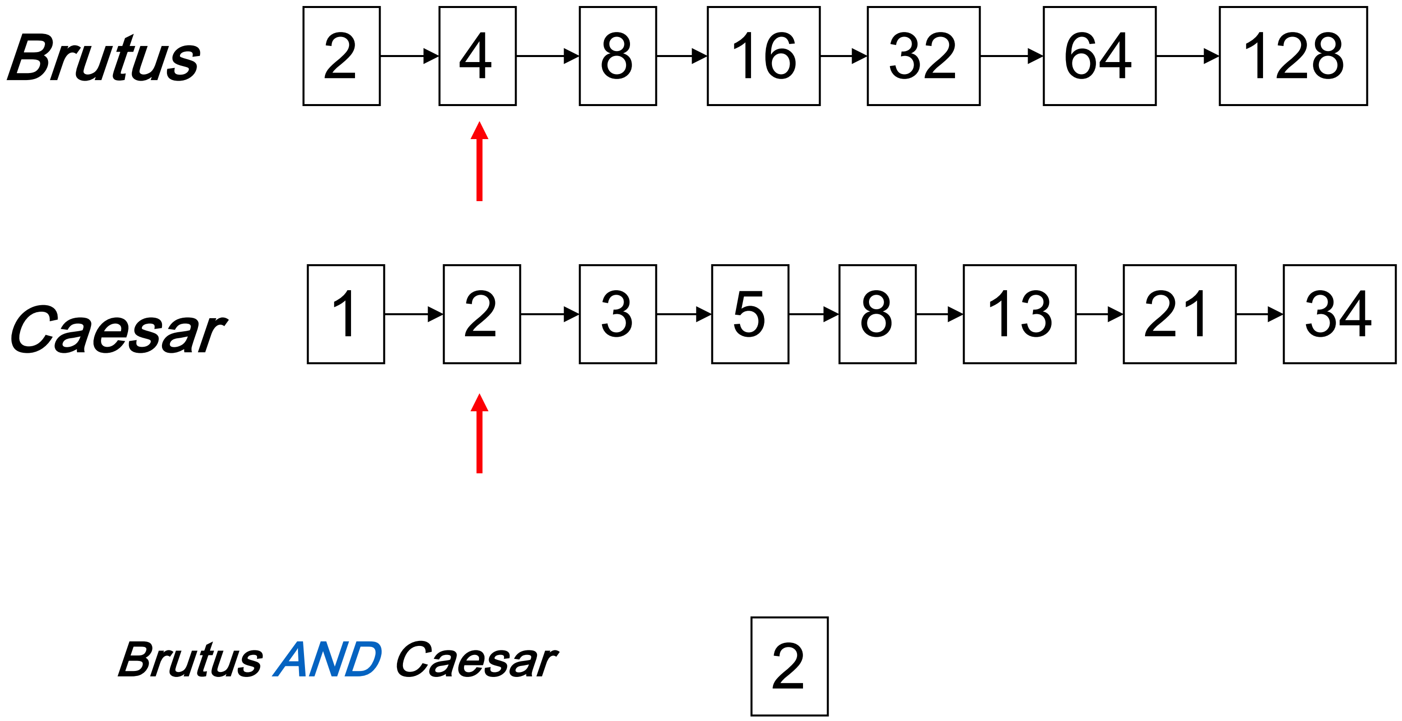
Brutus 2 → 4 → 8 → 16 → 32 → 64 → 128

Caesar 1 → 2 → 3 → 5 → 8 → 13 → 21 → 34

Brutus AND Caesar 2

The merge

- Walk through the two postings simultaneously



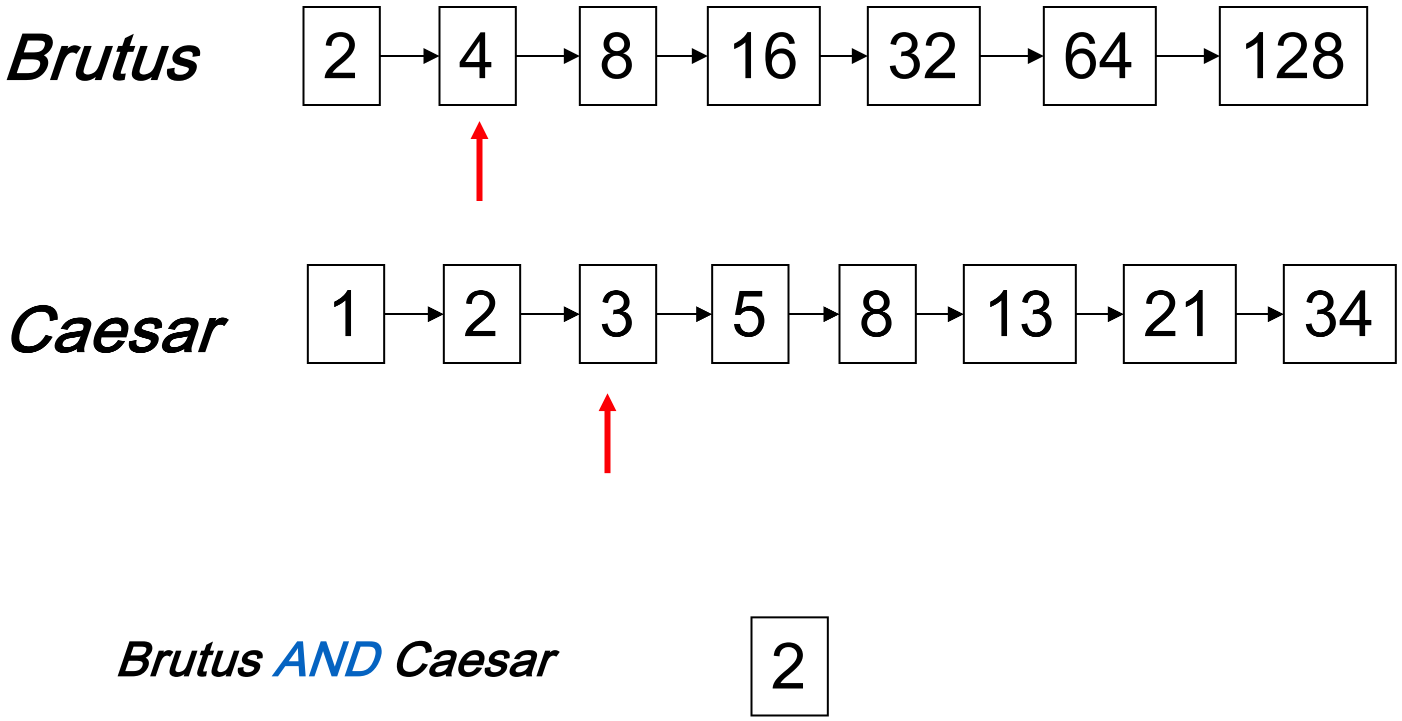
Brutus 2 → 4 → 8 → 16 → 32 → 64 → 128

Caesar 1 → 2 → 3 → 5 → 8 → 13 → 21 → 34

Brutus AND Caesar 2

The merge

- Walk through the two postings simultaneously



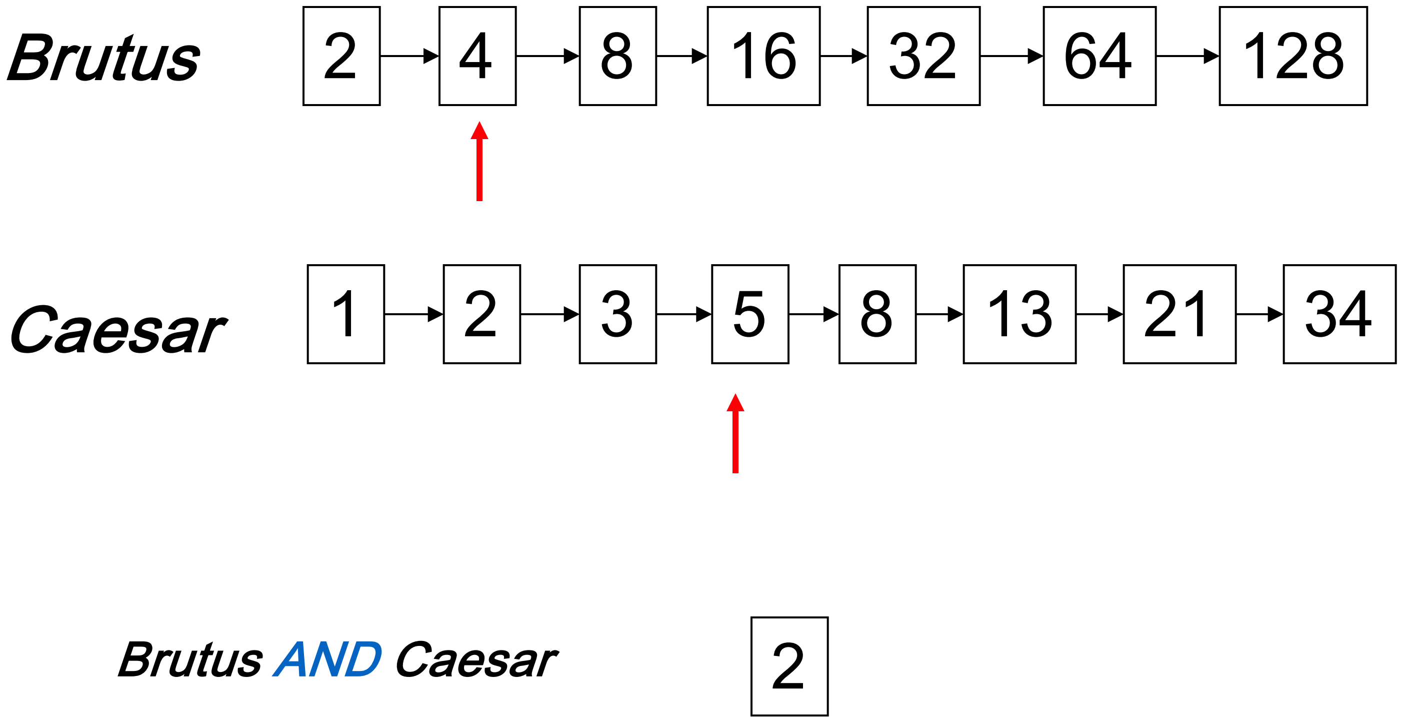
Brutus 2 → 4 → 8 → 16 → 32 → 64 → 128

Caesar 1 → 2 → 3 → 5 → 8 → 13 → 21 → 34

Brutus AND Caesar 2

The merge

- Walk through the two postings simultaneously



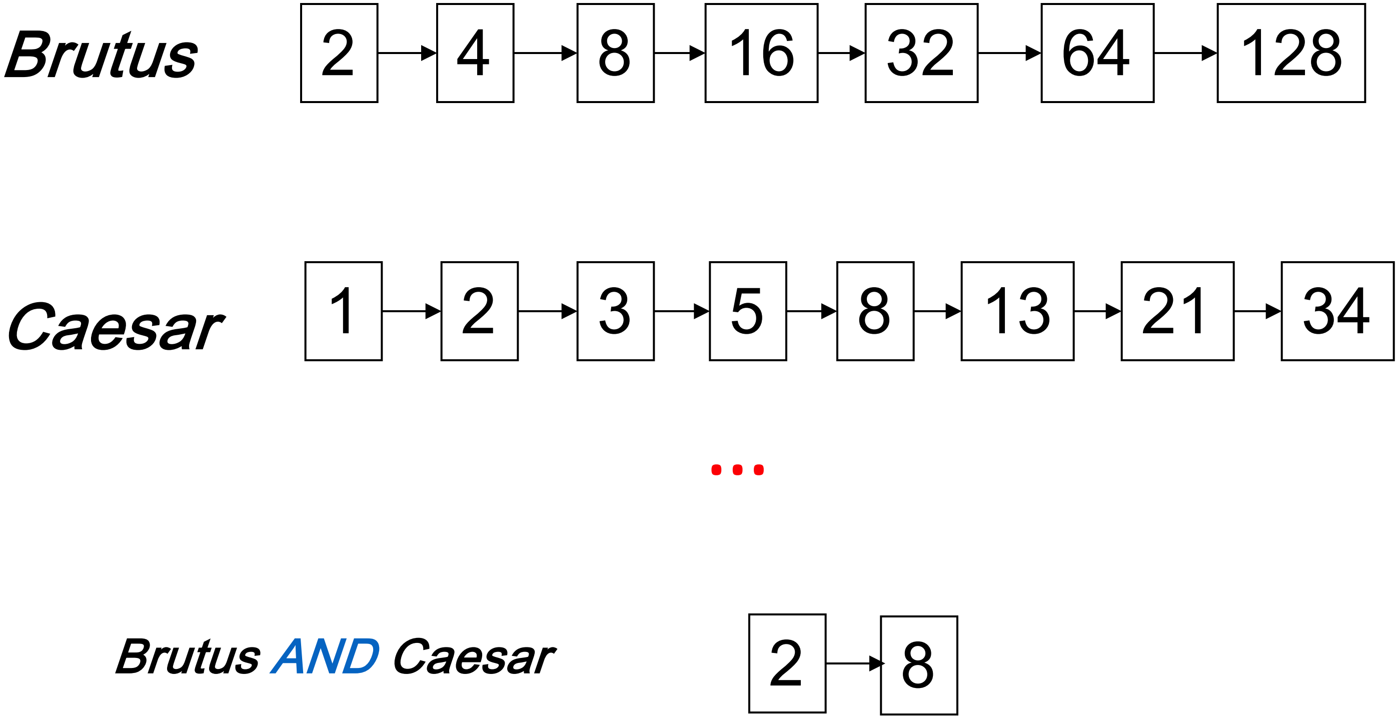
Brutus 2 → 4 → 8 → 16 → 32 → 64 → 128

Caesar 1 → 2 → 3 → 5 → 8 → 13 → 21 → 34

Brutus AND Caesar 2

The merge

- Walk through the two postings simultaneously



Brutus 2 → 4 → 8 → 16 → 32 → 64 → 128

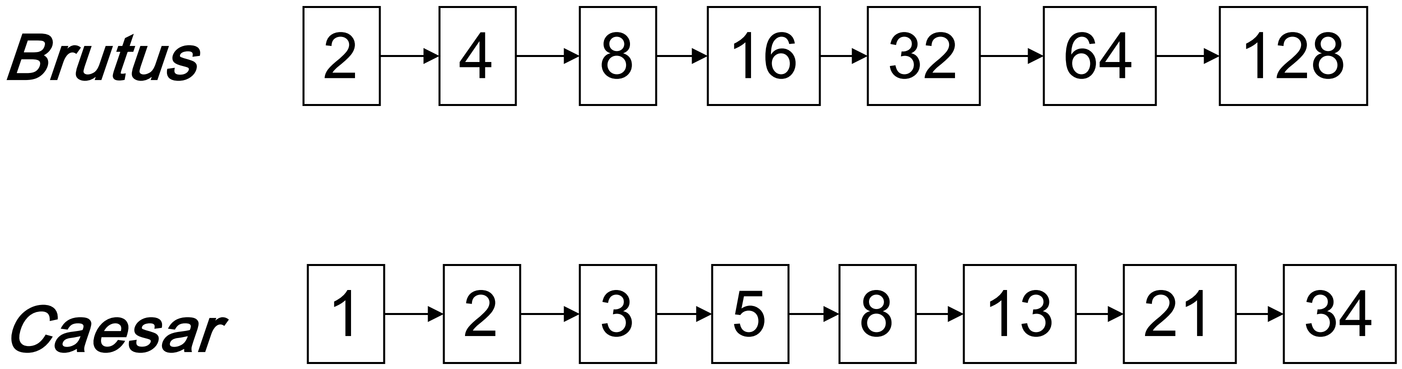
Caesar 1 → 2 → 3 → 5 → 8 → 13 → 21 → 34

…

Brutus AND Caesar 2 → 8

The merge

- Walk through the two postings simultaneously



Brutus 2 → 4 → 8 → 16 → 32 → 64 → 128

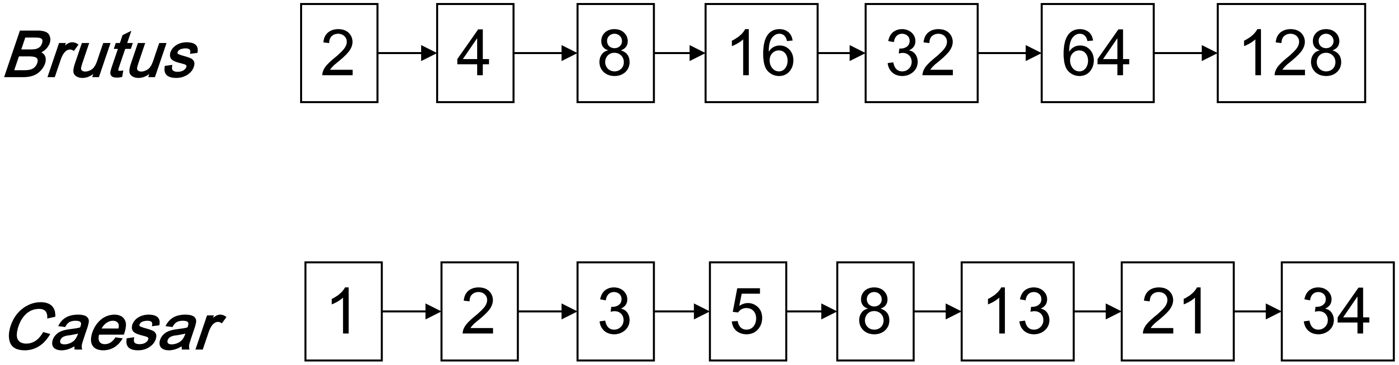
Caesar 1 → 2 → 3 → 5 → 8 → 13 → 21 → 34

What assumption are we making about the postings lists?

For efficiency, when we construct the index, we ensure that the postings lists are sorted

The merge

- Walk through the two postings simultaneously



Brutus 2 → 4 → 8 → 16 → 32 → 64 → 128

Caesar 1 → 2 → 3 → 5 → 8 → 13 → 21 → 34

What is the running time?

O(length1 + length2)

Boolean Queries

Introduction to Information Retrieval

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, macOS Spotlight

Merging

What about an arbitrary Boolean formula?

(Brutus *OR* ***Caesar)*** *AND NOT*

(Antony *OR* ***Cleopatra)***

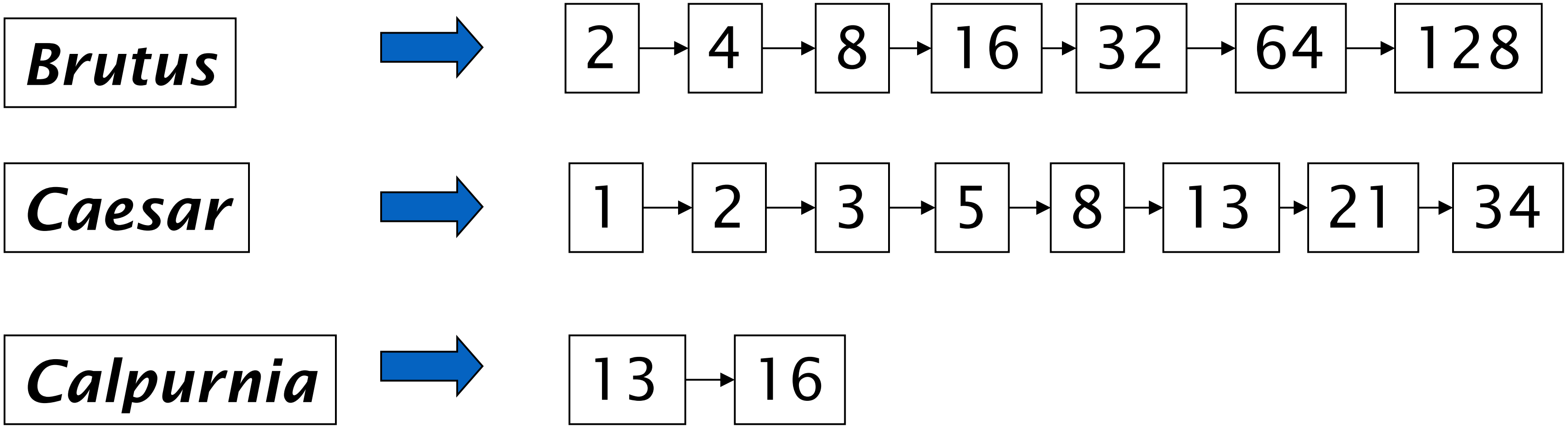
- *x* = ***(Brutus*** *OR* ***Caesar)***
- *y* = ***(Antony*** *OR* ***Cleopatra)***
- *x AND NOT y*

- Is there an upper bound on the running time?
  - O(total_terms * query_terms)
- What about ***Brutus*** *AND* ***Calpurnia*** *AND* ***Caesar***?

Query Optimization

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

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



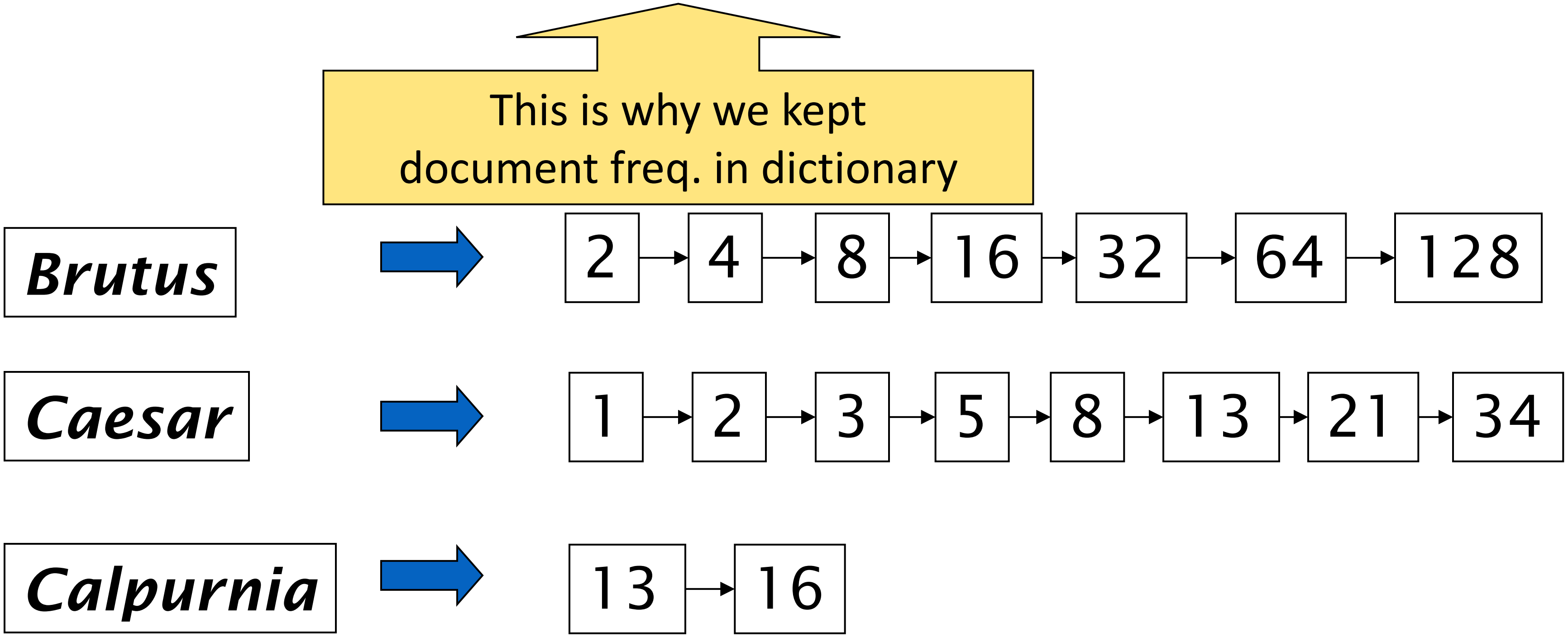
Brutus → 2 → 4 → 8 → 16 → 32 → 64 → 128

Caesar → 1 → 2 → 3 → 5 → 8 → 13 → 21 → 34

Calpurnia → 13 → 16

Query optimization example

- Heuristic: Process in order of increasing freq:
  - *merge the two terms with the shortest postings list*



This is why we kept document freq. in dictionary

Brutus → 2 → 4 → 8 → 16 → 32 → 64 → 128

Caesar → 1 → 2 → 3 → 5 → 8 → 13 → 21 → 34

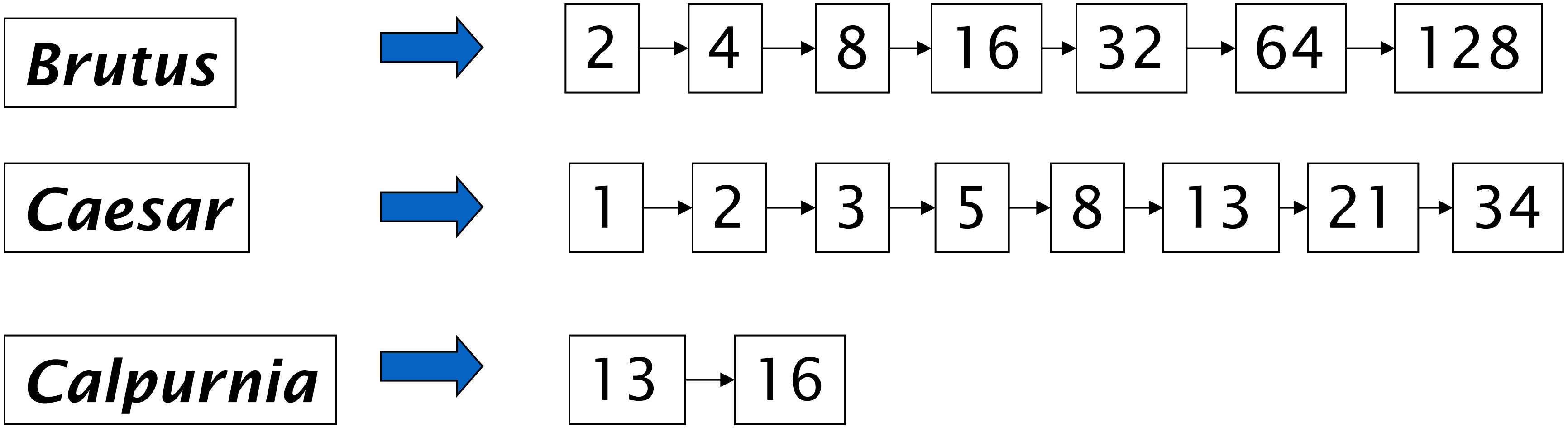
Calpurnia → 13 → 16

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

Query optimization - OR

Query: ***Brutus*** *OR* ***Calpurnia*** *OR* ***Caesar***

- Consider a query that is an *OR* of *t* terms.
- What is the best order for query processing?
- Same: still want to merge the shortest postings lists first



Brutus → 2 → 4 → 8 → 16 → 32 → 64 → 128

Caesar → 1 → 2 → 3 → 5 → 8 → 13 → 21 → 34

Calpurnia → 13 → 16

Query optimization in general

Query: (madding OR crowd) AND (ignoble OR strife)

- Need to evaluate OR statements first
- Which OR should we do first?
  - 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