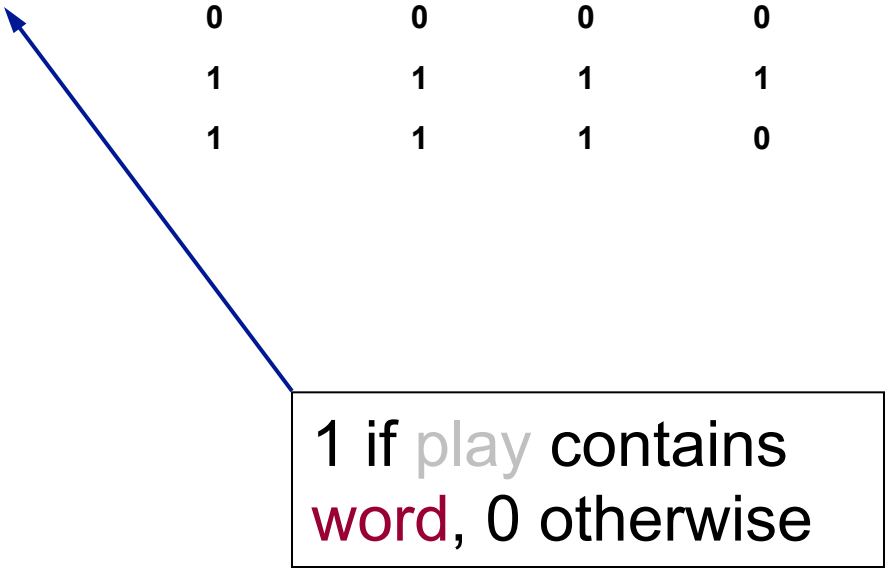
CSE 535
Information Retrieval

Lecture 2: Boolean Retrieval Model

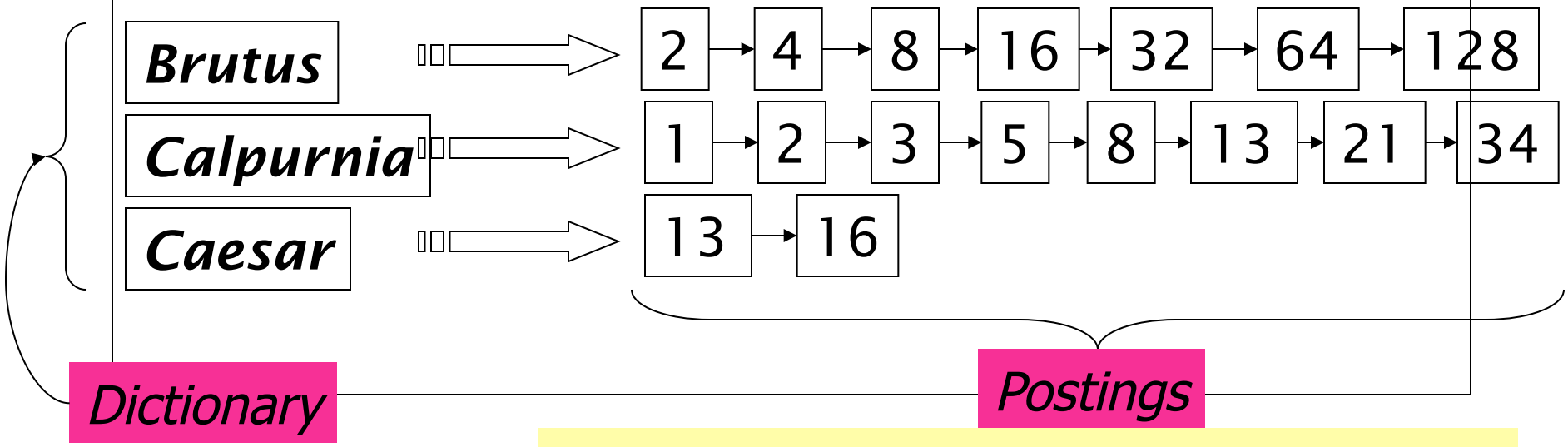
Term-document incidence

	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



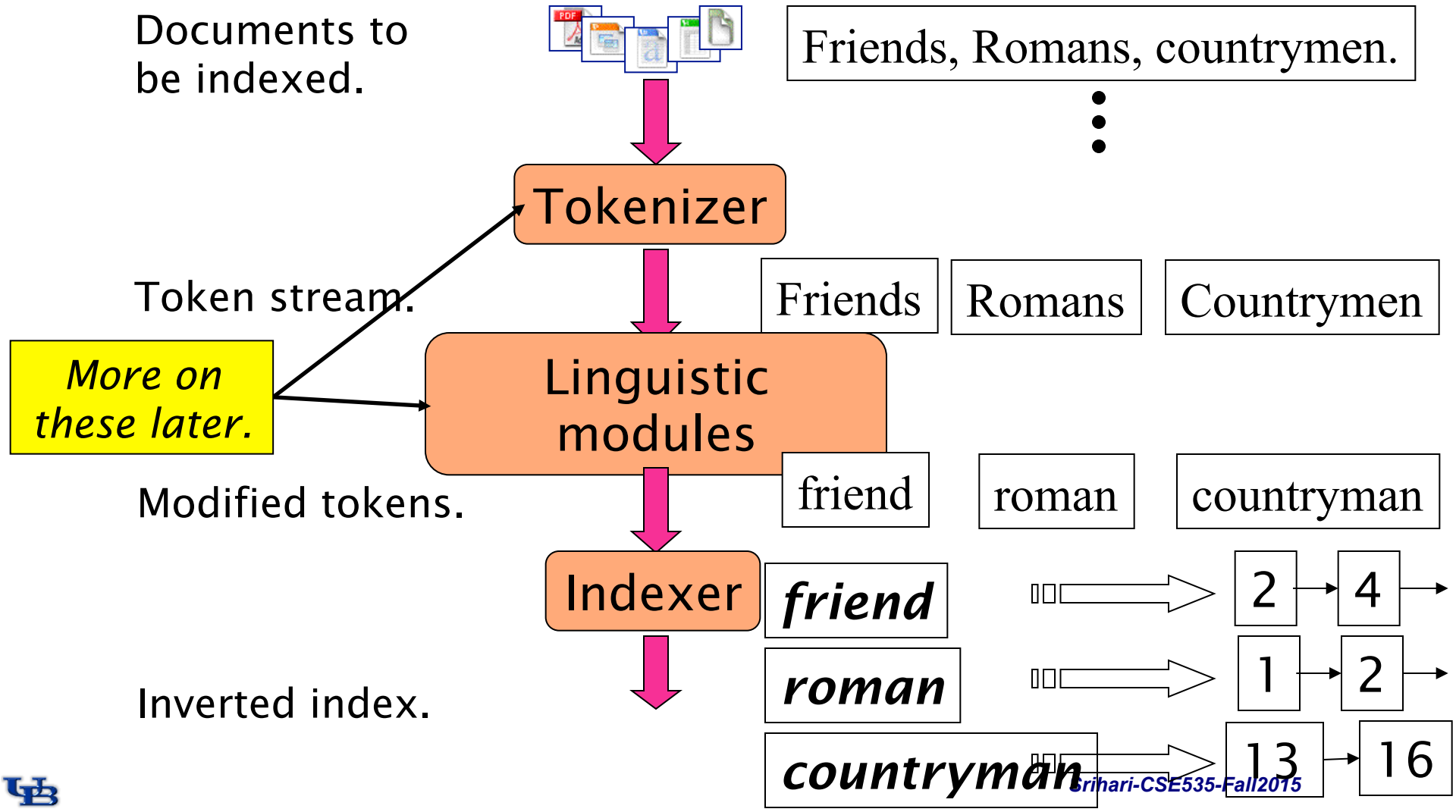
Inverted index

- **Linked lists generally preferred to arrays**
  - Dynamic space allocation
  - Insertion of terms into documents easy
  - Space overhead of pointers



Sorted by docID (more later on why).

Inverted index construction



Indexer steps

- **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	Doc #
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

- **Sort by terms.**

Core indexing step.

Term	Doc #
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	Doc #
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

- **Multiple term entries in a single document are merged.**
- **Frequency information is added.**

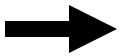
Why frequency?
Will discuss later.

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

and a *Postings* file.

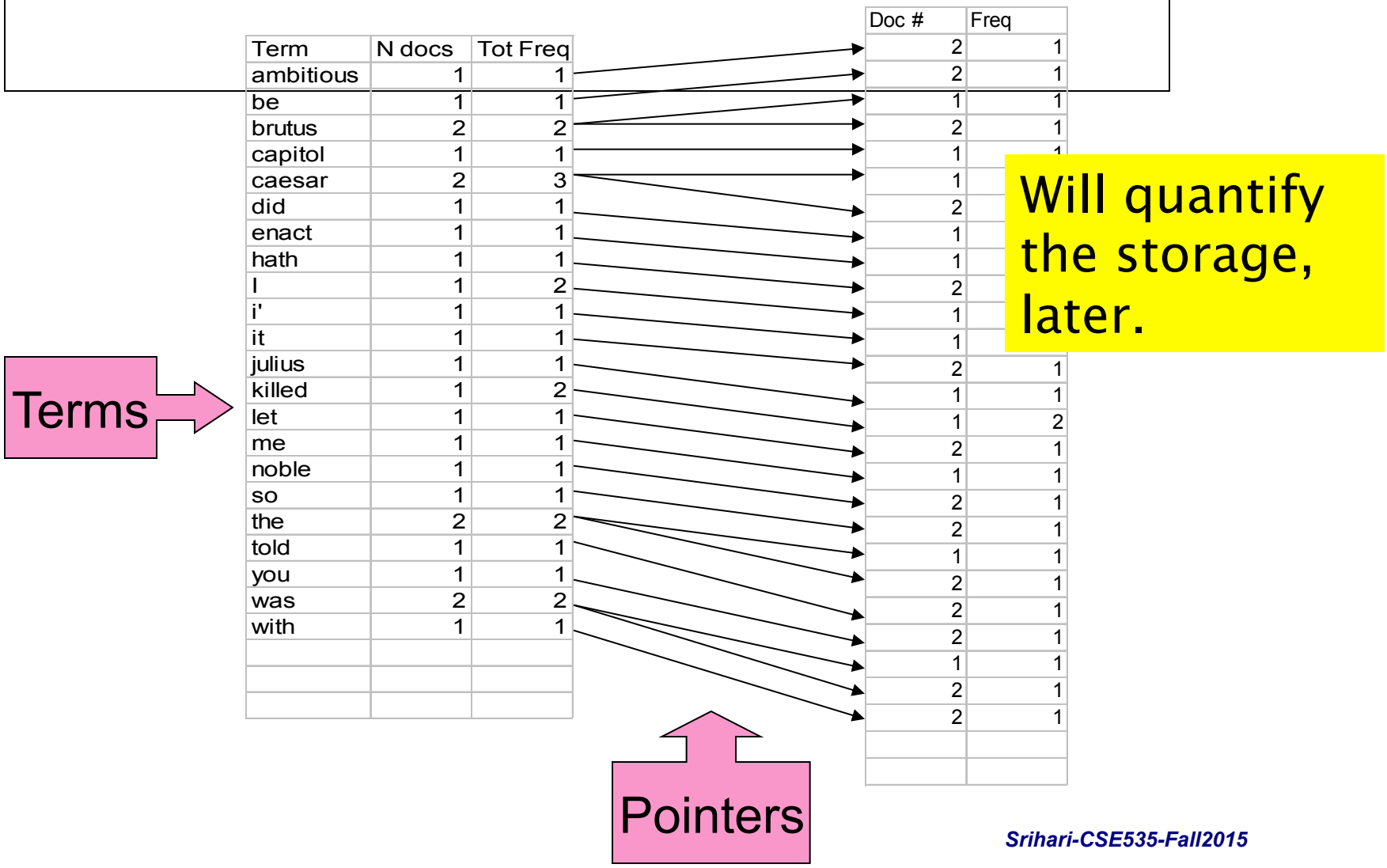
Term	Doc #	Freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1



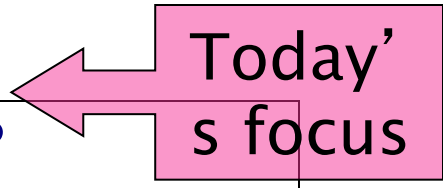
Term	N docs	Tot Freq
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
i'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	2	2
told	1	1
you	1	1
was	2	2
with	1	1

Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	1
2	2
1	1
1	1
2	1
1	2
1	1
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

Where do we pay in storage?



The index we just built



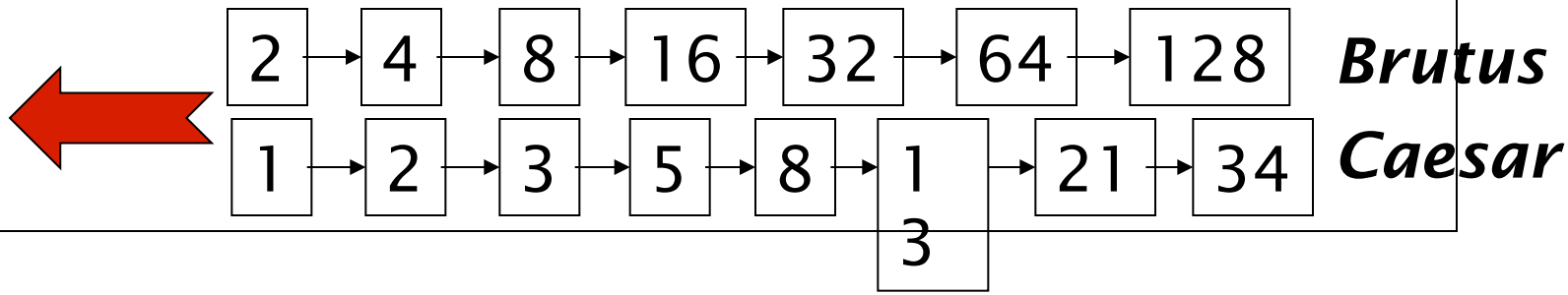
- **How do we process a Boolean query?**
  - Later - what kinds of queries can we process?

Query processing

- **Consider processing the query:**

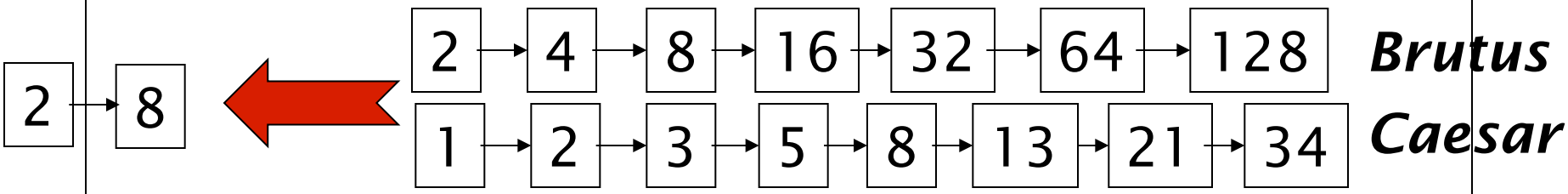
  Brutus *AND* ***Caesar***

  - Locate ***Brutus*** in the Dictionary;
    - ✓ Retrieve its postings.
  - Locate *Caesar* in the Dictionary;
    - ✓ Retrieve its postings.
  - "Merge" the two postings: actually, intersecting them



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.

Basic postings intersection

```
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
```

► **Figure 1.7** Algorithm for the intersection of two postings lists p_1 and p_2.

Boolean queries: Exact match

- **Queries using *AND*, *OR* and *NOT* together with query terms**
  - Views each document as a set of words
  - Is precise: document matches condition or not.
- **Primary commercial retrieval tool for 3 decades.**
- **Professional searchers (e.g., Lawyers) still like Boolean queries:**
  - You know exactly what you're getting.

Example: WestLaw http://www.westlaw.com/

- **Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992)**
- **About 7 terabytes of data; 700,000 users**
- **Majority of users *still* use boolean queries**
- **Example query:**
  - What is the statute of limitations in cases involving the federal tort claims act?
  - LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM
- **Long, precise queries; proximity operators; incrementally developed; not like web search**

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?

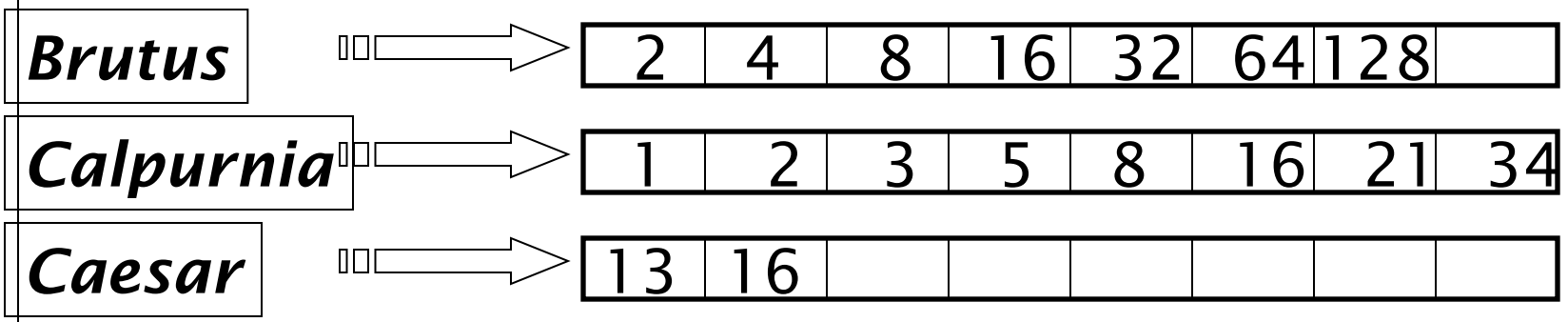
*(Brutus **OR** Caesar) **AND NOT***

*(Antony **OR** Cleopatra)*

- **Can we always merge in "linear" time?**
  - Linear in what?
- **Can we do better?**

Query optimization

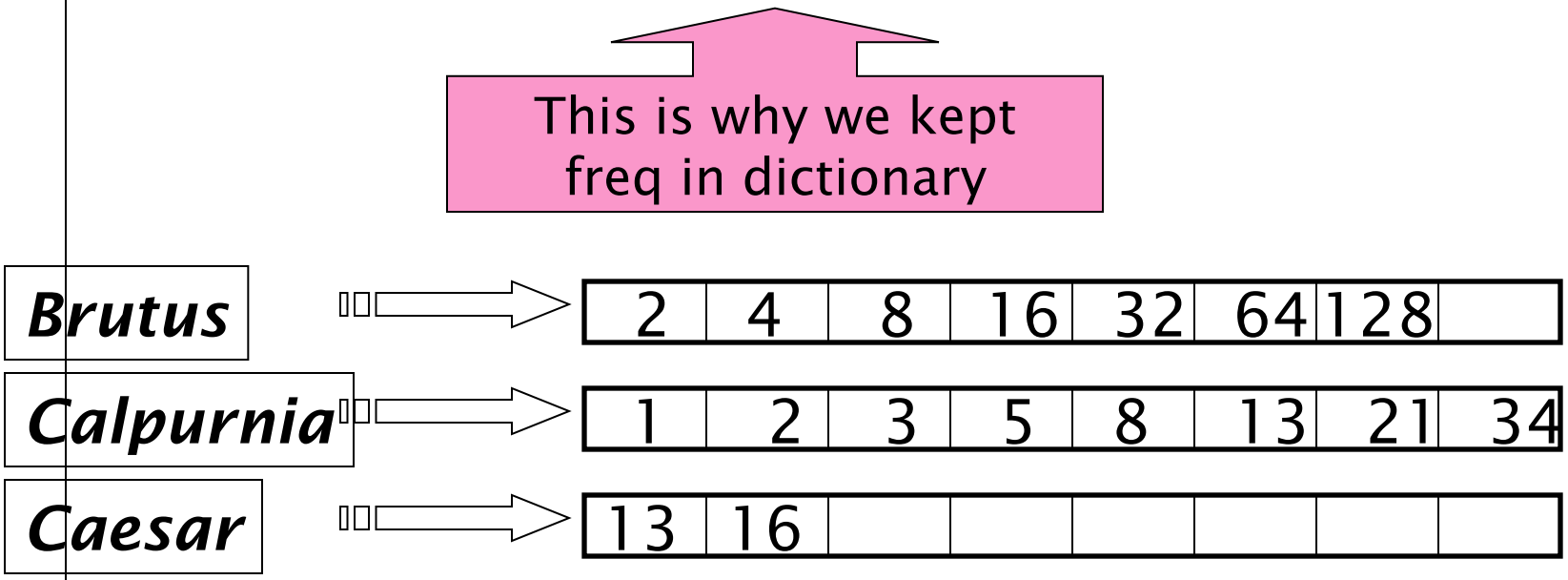
- What is the best order for query processing?
- Consider a query that is an *AND* of *t* terms.
- For each of the *t* terms, get its postings, then *AND* together.



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 (***Caesar*** *AND* ***Brutus)*** *AND* ***Calpurnia***.

Query optimization

```
INTERSECT(⟨t_1, ..., t_n⟩)
1  terms ← SORTBYINCREASINGFREQUENCY(⟨t_1, ..., t_n⟩)
2  result ← POSTINGS(FIRST(terms))
3  terms ← REST(terms)
4  while terms ≠ NIL and result ≠ NIL
5  do list ← POSTINGS(FIRST(terms))
6      result ← INTERSECT(result, POSTINGS(FIRST(terms)))
7      terms ← REST(terms)
8
9  return result
```

► **Figure 1.8** Algorithm for conjunctive queries that returns the set of documents containing each term in the input list of terms.

More general optimization

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

Beyond Boolean term search

- **What about phrases?**
- **Proximity: Find *Gates NEAR Microsoft*.**
  - Need index to capture position information in docs. More later.
- **Zones in documents: Find documents with (*author* = *Ullman*) *AND* (text contains *automata*).**

Evidence accumulation

- **1 vs. 0 occurrence of a search term**
  - 2 vs. 1 occurrence
  - 3 vs. 2 occurrences, etc.
- **Need term frequency information in docs.**
- **Used to compute a score for each document**
- **Matching documents *rank-ordered* by this score.**