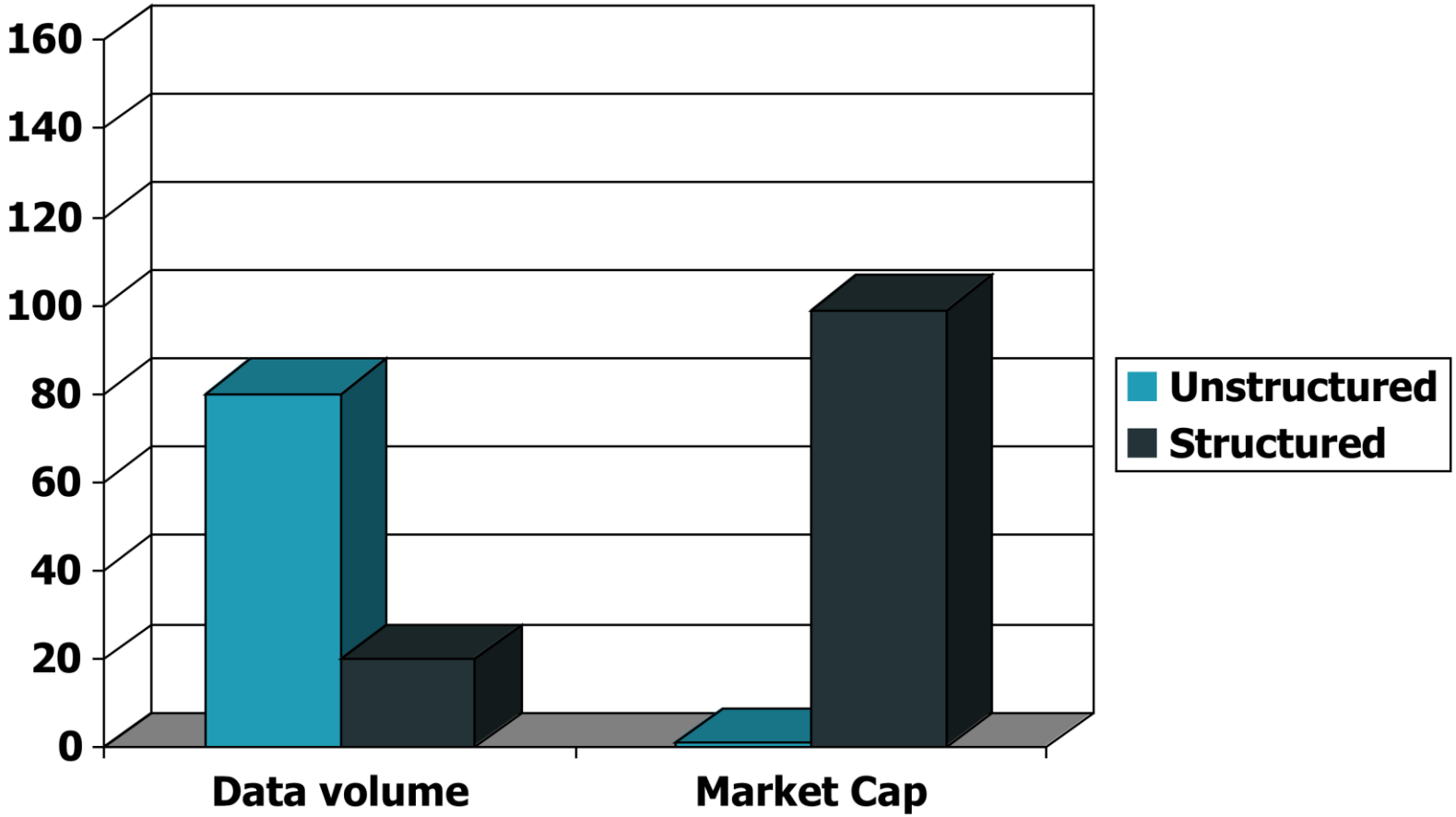
LC029 정보검색

Chapter 1: Boolean Retrieval

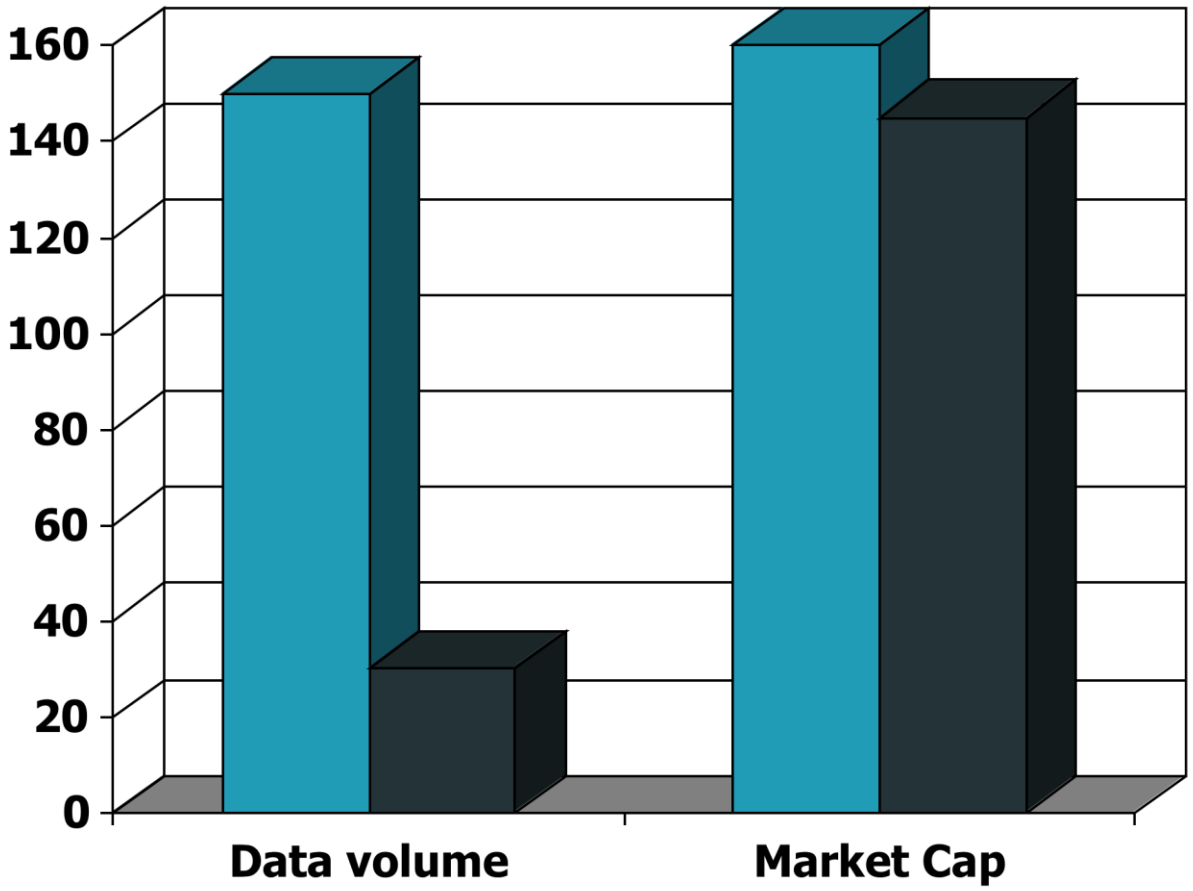
Information Retrieval

- 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).

Unstructured (text) vs. structured (database) data in 1996



Unstructured (text) vs. structured (database) data in 2006



Unstructured data 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***?
  - 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)

UNIX command : grep

```
a: Brutus Caesar Calpurnia
b: Brutus Calpurnia
c: Caesar Calpurnia
d: Brutus Caesar
e: Calpurnia
f: Brutus
```

UNIX command : grep

```
$ grep Brutus *
```

```
$ grep Brutus * | grep Caesar
```

```
$ grep Brutus * | grep Caesar | grep -v Calpurnia
```

```
$ grep -v Caesar *
```

Term-document (incidence) matrix

	Antony & 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 play contains word, 0 otherwise

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

Incidence vectors

- So we have a 0/1 incidence 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.

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

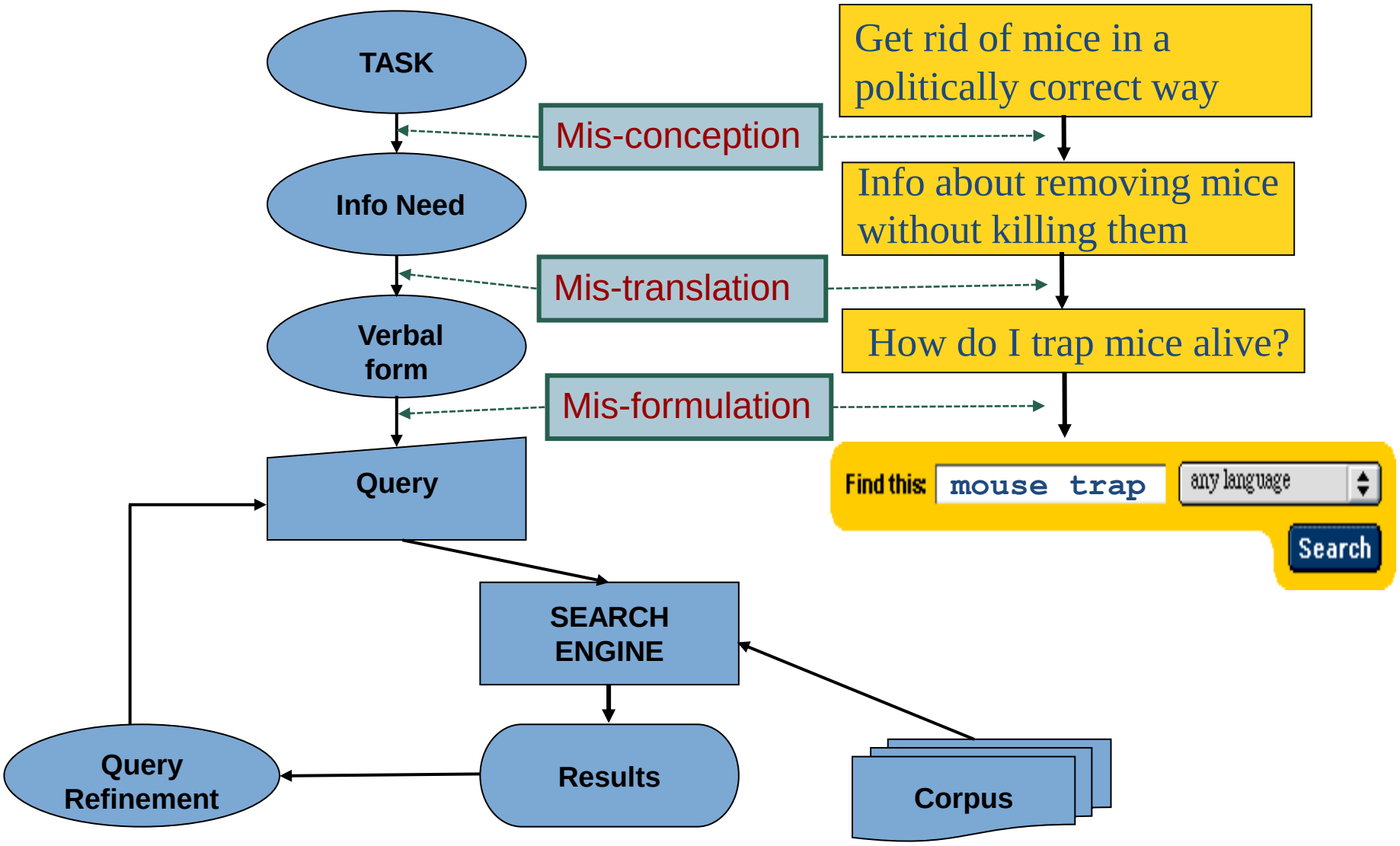
- Hamlet, Act III, Scene ii

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

Basic assumptions of Information Retrieval

- **Collection**: Fixed set of documents
- **Goal**: Retrieve documents with information that is relevant to user's information need and helps him complete a task

The classic search model



How good are the retrieved docs?

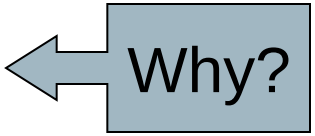
- ***Precision*** : Fraction of retrieved docs that are relevant to user's information need
- ***Recall*** : Fraction of relevant docs in collection that are retrieved
- **F-measure** : The weighted harmonic mean of precision and recall

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.

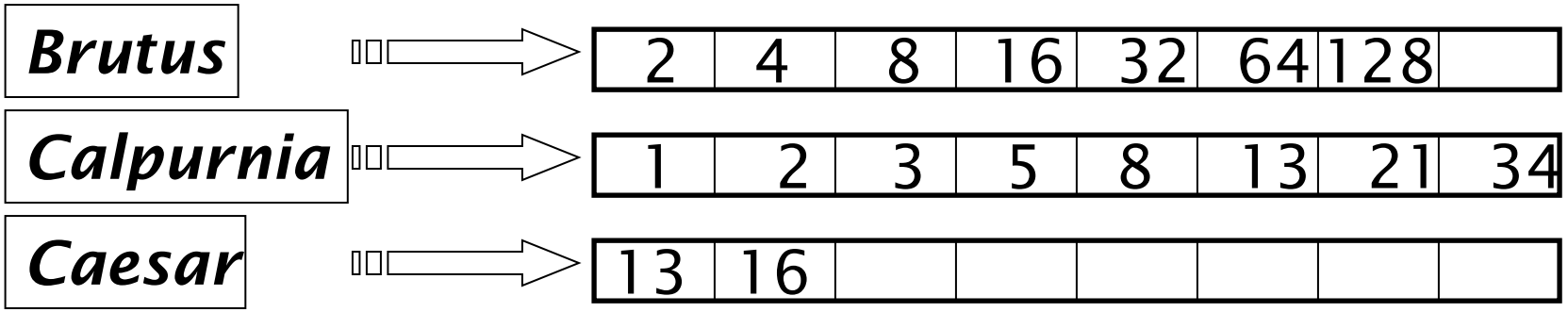
Can’t build the term-document matrix

- 500K x 1M matrix has half-a-trillion 0’s and 1’s.
- But it has no more than one billion 1’s.
  - matrix is extremely sparse
  - 99.8% of the cells are zero
- What’s a better representation?
  - We only record the 1 positions



Inverted index (inverted file)

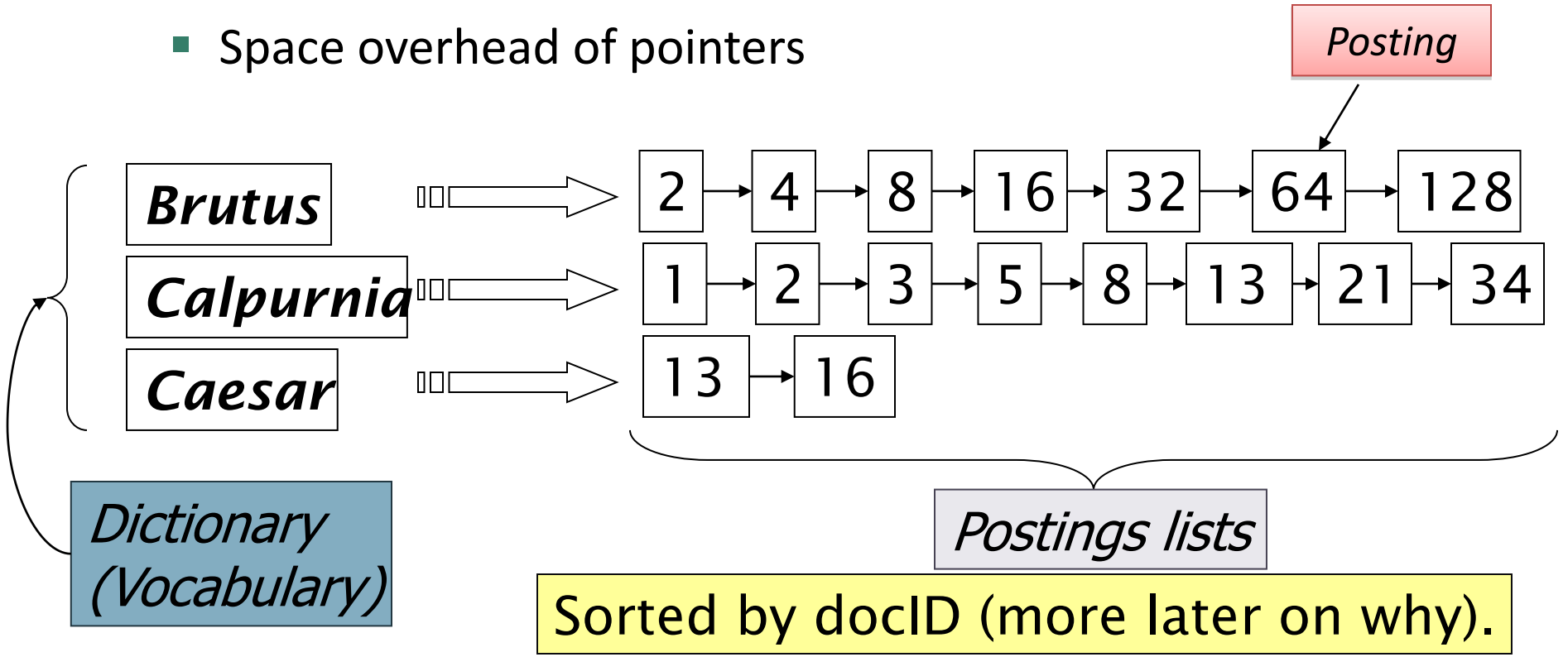
- For each term *T*, we must store a list of all documents that contain *T*.
- Do we use an array or a linked list for this?



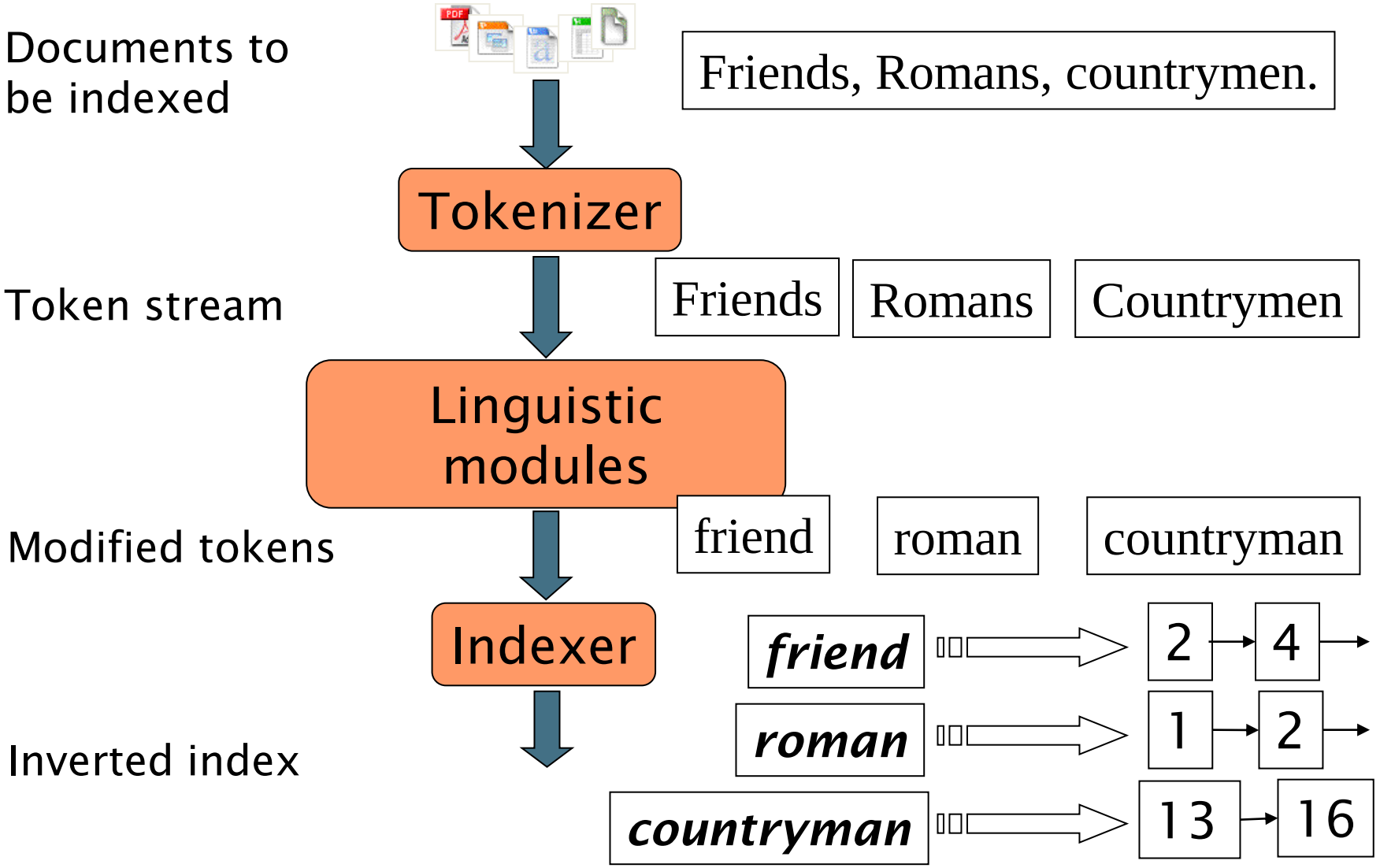
What happens if the word ***Caesar*** is added to document 14?

Inverted index

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



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

Indexer steps

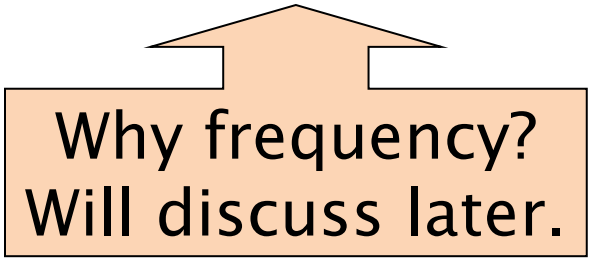
- Sort by terms

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

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

Indexer steps

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



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

Term	Doc #	Term 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

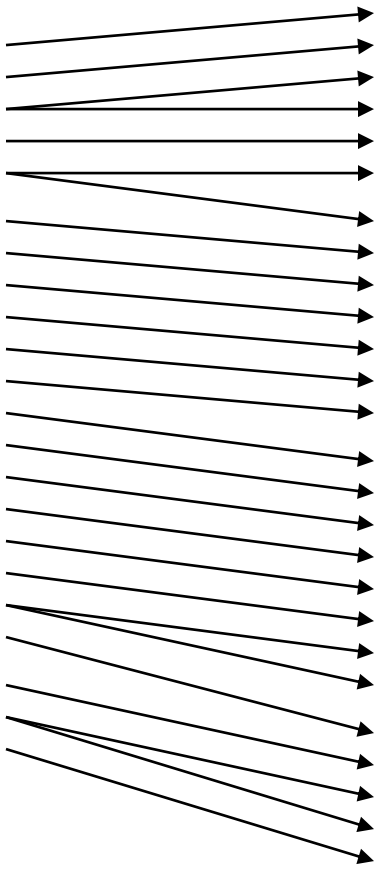
Indexer steps

Term Frequency (TF)

Document Frequency (DF)

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

Term	Doc freq	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

- The result is split into a *Dictionary* file and a *Postings* file

Where to keep Dictionary and Postings

- Dictionary is commonly kept in memory.
- Postings are normally kept on disk.
- For in-memory postings,
  (1) linked list scheme
      easy to insert new postings into the posting list
      space overhead for pointers
  (2) variable length array scheme
      no overhead for pointers
      the use of contiguous memory increases speed
  (3) hybrid scheme

The index we just built

- How do we process a query?
  - We deal with Boolean queries
  - Use the inverted index to process a query
  - Query types

    (1) ***Conjunctive*** query

    Brutus AND Calpurnia

    (2) ***Disjunctive*** query

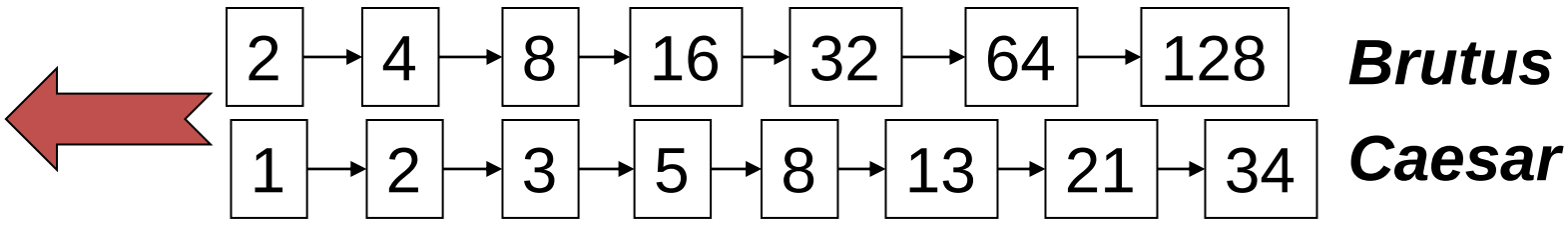
    Brutus OR Calpurnia

Query processing: AND

- Consider processing the query:

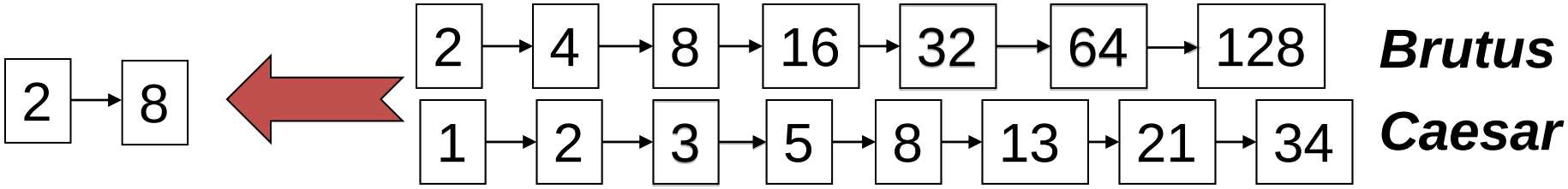
  Brutus *AND* ***Caesar***

  - Locate ***Brutus*** in the Dictionary
    - Retrieve its postings list
  - Locate ***Caesar*** in the Dictionary
    - Retrieve its postings list
  - "Merge" (intersection) the two postings list:



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 process 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 given 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)
- Tens of terabytes of data; 700,000 users
- Majority of users *still* use boolean queries

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

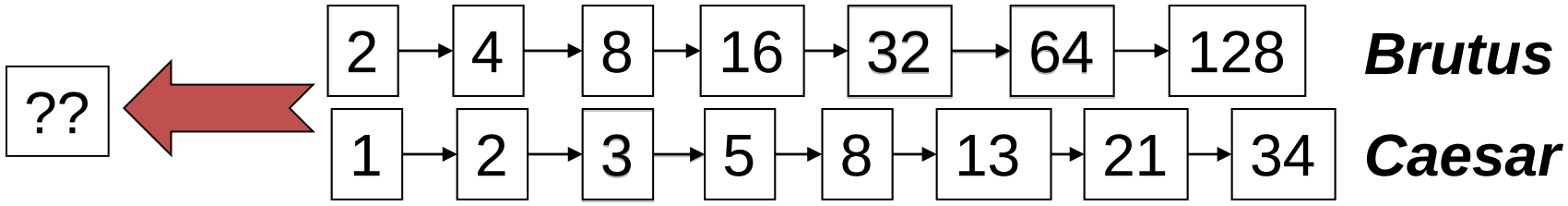
- Example query:
  - What is the statute of limitations in cases involving the Federal Tort Claims Act?
    limit! /3 statute action /s federal /2 tort /3 claim
  - Requirements for disabled people to be able to access a workplace
    disabl! /p access! /s work-site work-place (employment /3 place)
  - /3 = within 3 words
    /s = in same sentence
    /p = in the same paragraph

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

- Note that SPACE is disjunction, not conjunction!
  - Long, precise queries
  - Proximity operators
  - Incrementally developed
- Professional searchers often like Boolean search:
  - Precision, transparency and control
- But that doesn't mean they actually work better….

Boolean queries: More general merges

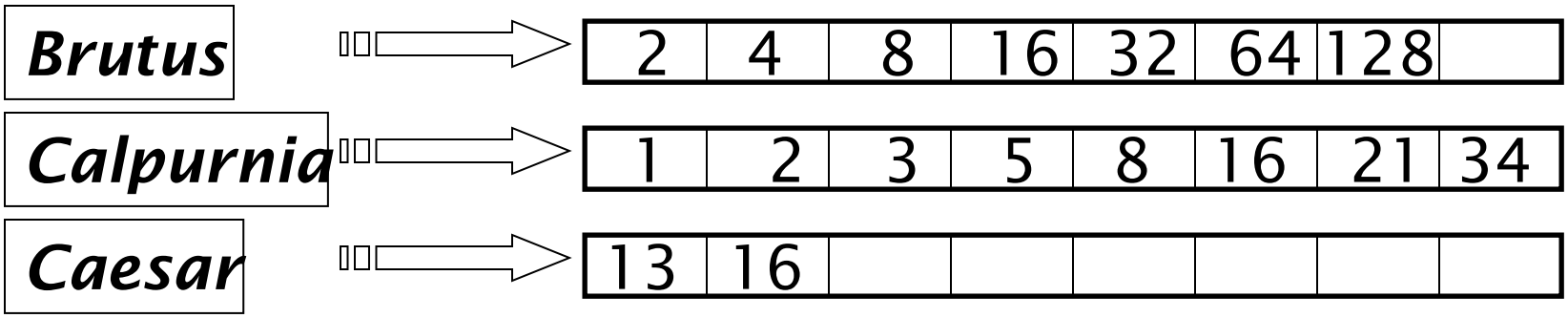
- Walk through the two postings simultaneously, in time linear in the total number of postings entries



Brutus *AND NOT* ***Caesar***
Brutus *OR NOT* ***Caesar***

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 list, then *AND* them together.

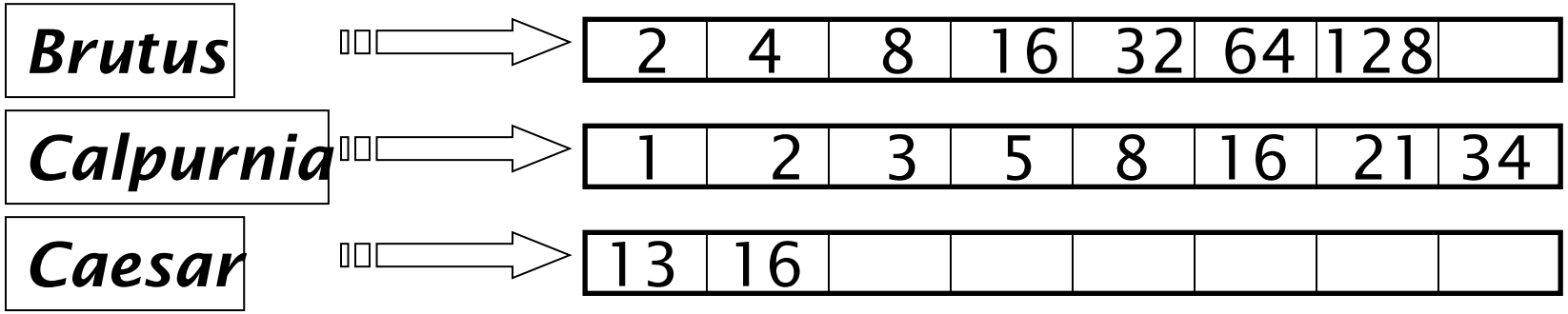


Query: ***Brutus*** *AND* ***Calpurnia*** *AND* ***Caesar***

Query optimization example

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

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



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

More general optimization

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

- Get doc 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.

More general optimization : Exercise

- Recommend a query processing order for

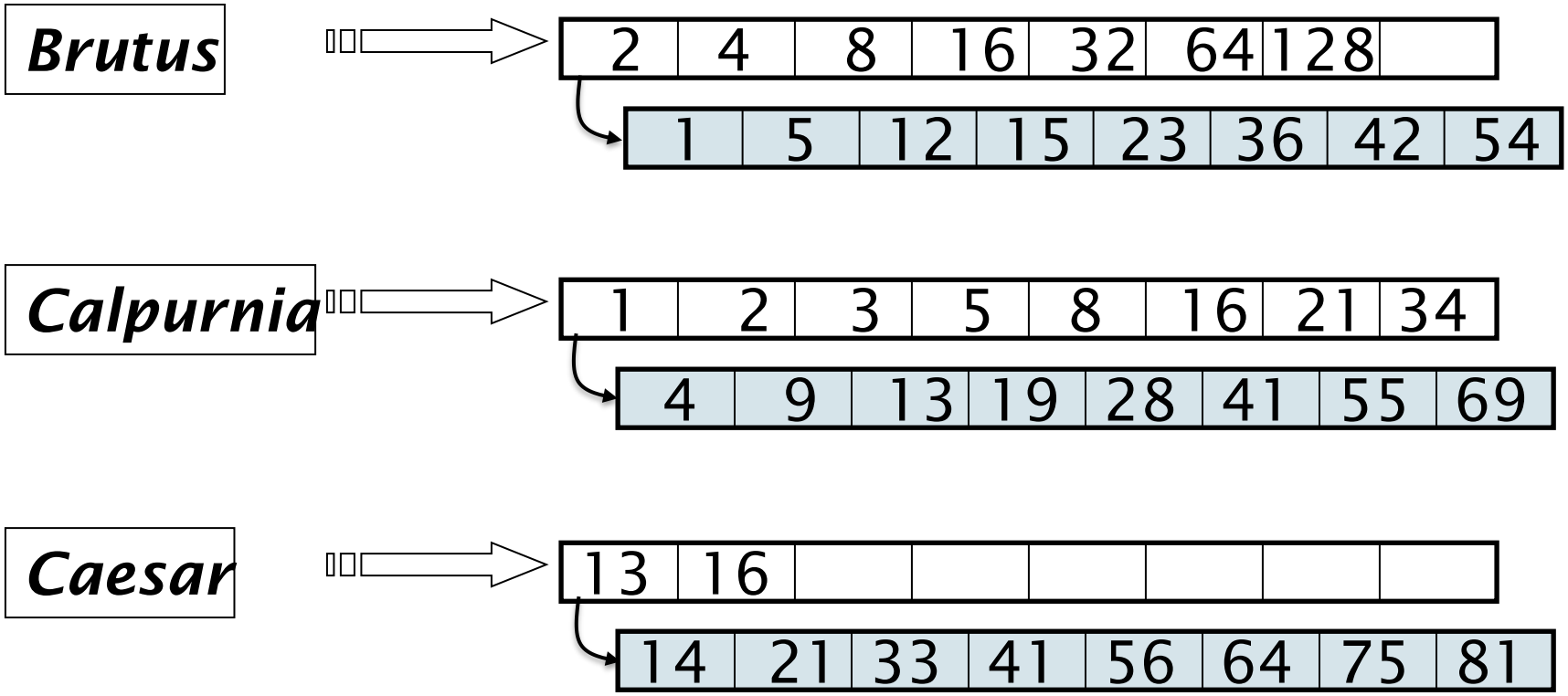
  (*tangerine* OR ***trees*****)** AND **(*marmalade*** OR ***skies*****)** AND **(*kaleidoscope*** *OR* ***eyes*****)**

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

Beyond term search, what's ahead in IR?

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

Inverted index with position information

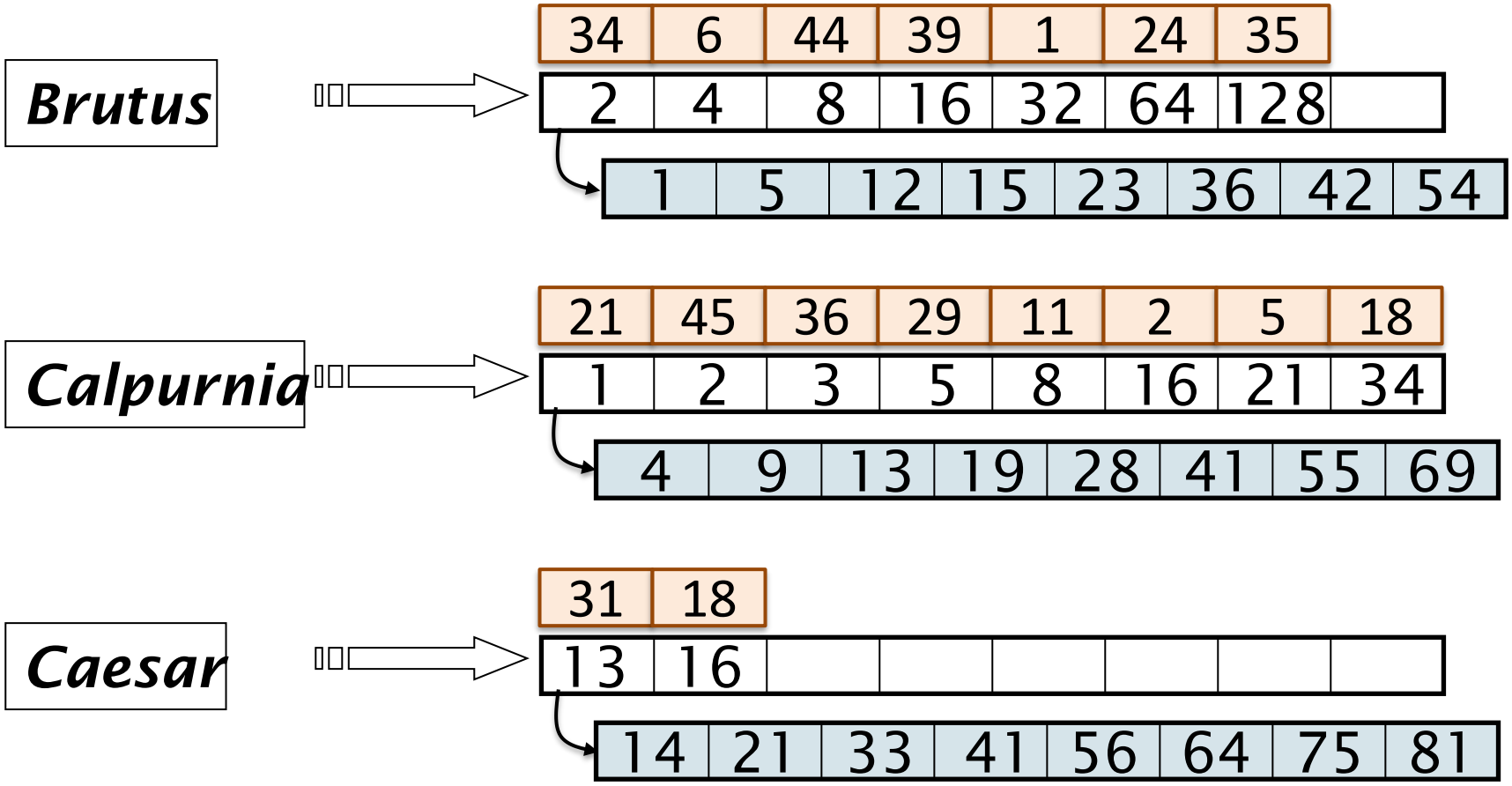


Query: ***Brutus*** */3* ***Calpurnia*** */5* ***Caesar***

Evidence accumulation

- 1 vs 0 occurrence of a search term
  - 2 vs 1 occurrence
  - 3 vs 2 occurrences, etc.
  - Usually more seems better
- Need Term Frequency information in docs

Inverted index with term frequency



Ranking search results

- Boolean queries give **inclusion** or **exclusion** of docs.
  - Ranking is not possible!!
- Often we want to rank/group results
  - Measure **proximity** from query for each document
  - Use the **term frequency** information
  - Documents in reverse **chronological order** (Westlaw)

IR vs databases : Structured vs unstructured data

- Structured data tends to refer to information in "tables"

Employee	Manager	Salary
Smith	Jones	50000
Chang	Smith	60000
Ivy	Smith	50000

Typically allows numerical range and exact match (for text) queries, e.g.,
Salary < 60000 AND Manager = Smith.

Unstructured data

- Typically refers to **free text**
- Allows
  - **Keyword queries** including operators
  - More sophisticated **“concept” queries**
    - *e.g.* Find all web pages dealing with ***drug abuse***
- Classic model for searching text documents

Semi-structured data

- In fact almost no data is “unstructured” to say nothing of linguistic structure
  - *e.g.* This slide has distinctly identified zones such as the *Title* and *Bullets*
- Facilitates “semi-structured” search such as
  - *Title* contains data AND *Bullets* contain search
- The focus of XML search

Semi-structured data

- XML documents

```
<book id="bk101">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <genre> cooking </genre>
    <publish_date>2000-10-01</publish_date>
    <price>30.00</price>
    <contents> ... </contents>
</book>
```

More sophisticated semi-structured search

- *Title* is about Object Oriented Programming AND *Author* something like stro*rup

  where * is the wild-card operator

- Issues:
  - how do you process “about”?
  - how do you rank results?

Clustering and classification

- **Clustering**
  - Given a set of documents, group them into clusters based on their contents.
- **Classification**
  - Given a set of topics, decide which topic the given documents belongs to.

The web and its challenges

- Unusual and diverse documents
- Unusual and diverse users, queries, information needs
  - *cf*. WestLaw users
- Beyond terms, exploit ideas from social networks
  - link analysis, clickstreams ...

More sophisticated *information* retrieval

- Cross-language information retrieval
- Question answering
- Summarization
- Text mining
- …