Information Retreival

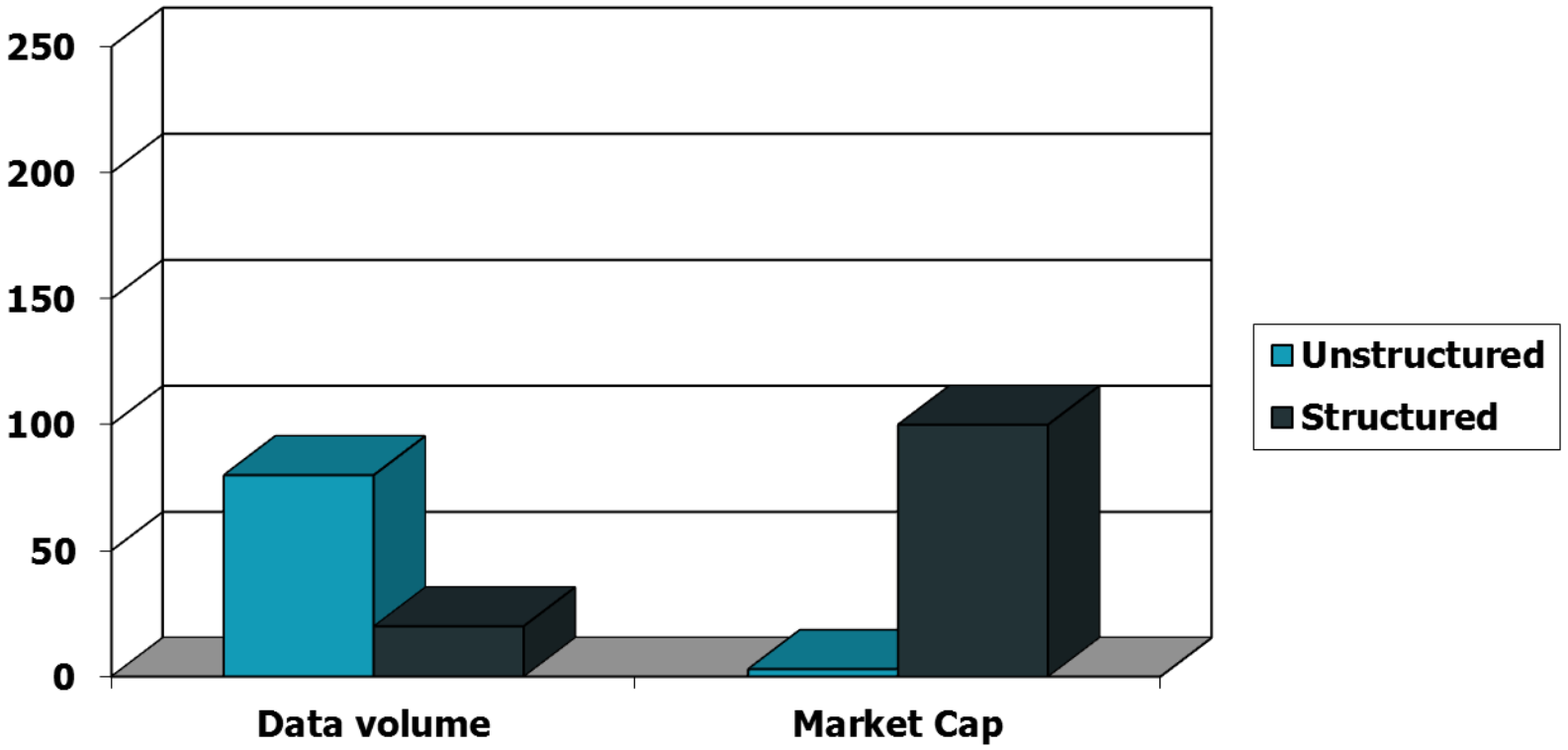
Dr. Qaiser Abbas
Department of Computer Science & IT
University of Sargodha
qaiser.abbas@uos.edu.pk

Course material adopted from Stanford NLP Group

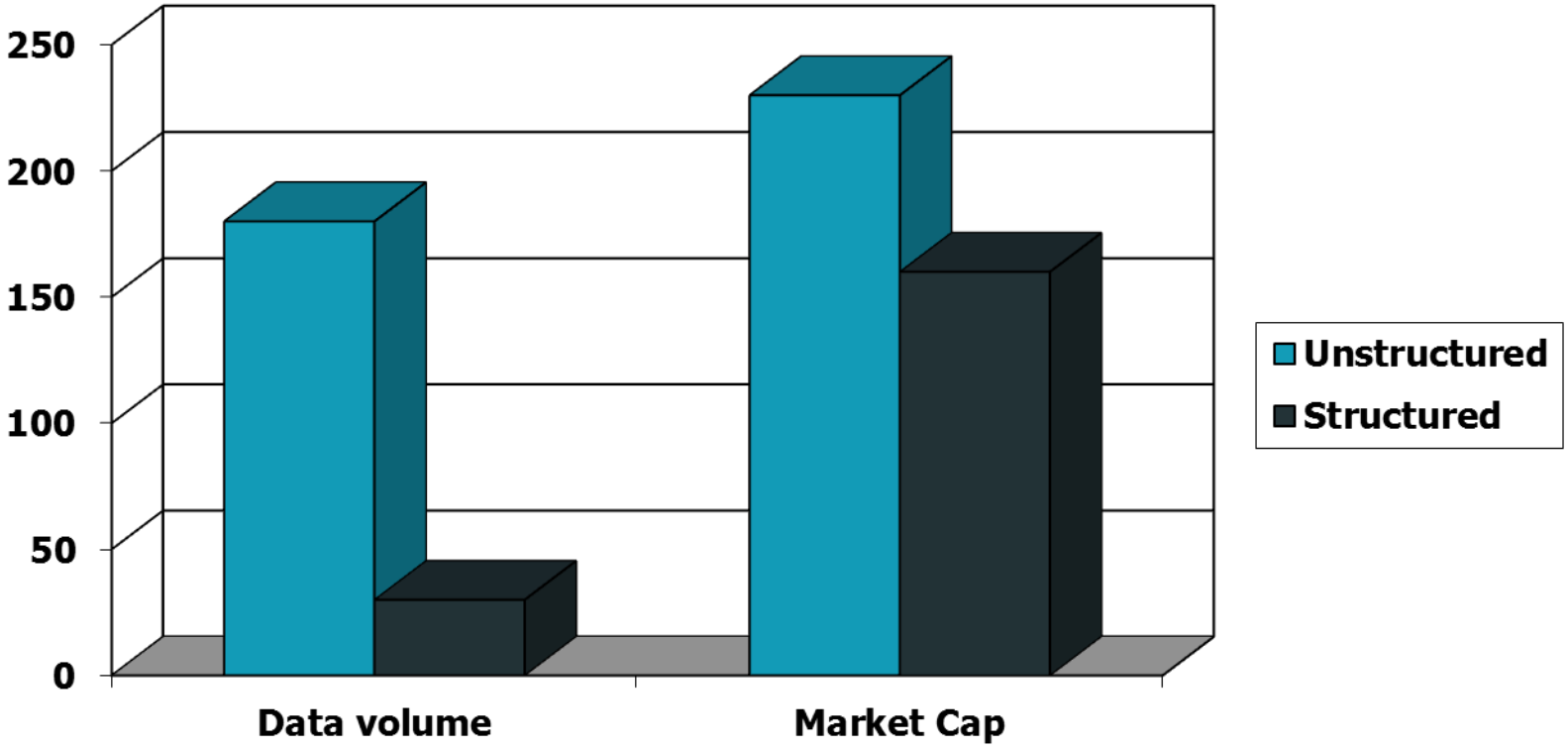
Information Retrieval

- 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).
  - These days we frequently think first of web search, but there are many other cases:
    - E-mail search
    - Searching your laptop
    - Corporate knowledge bases
    - Legal information retrieval

Unstructured (text) vs. structured (database) data in the mid-nineties



Unstructured (text) vs. structured (database) data today



Basic assumptions of IR

- **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.

How good are the retrieved docs?

- ***Precision*** : What fraction of the returned results are relevant to the information need?
- ***Recall*** : What fraction of the relevant documents in the collection were re- turned by the system?

  - More precise definitions and measurements to follow later.

Introduction to

Information Retrieval

Term-document incidence matrices

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 that not the answer?
  - Slow (for large corpora)
  - *NOT* ***Calpurnia*** is non-trivial (important)
  - Other operations (e.g., find the word ***Romans*** near ***countrymen***) not feasible due to line-basing.
  - Ranked retrieval (best documents to return)
    - Later lectures

Term-document incidence matrices

	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

1 if play contains word, 0 otherwise

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

Incidence vectors

- So, we have a 0/1 vector for each term.
- To answer query: take the vectors for ***Brutus, Caesar*** and ***Calpurnia*** (complemented) ➔ bitwise *AND*.
  - 110100 *AND*
  - 110111 *AND*
  - 101111 =
  - **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 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 term-docs matrix has half-a-trillion 0's and 1's.

- But it has no more than one billion 1's.
  - matrix is extremely sparse.

- What's a better representation?
  - We only record the 1 positions.

Introduction to

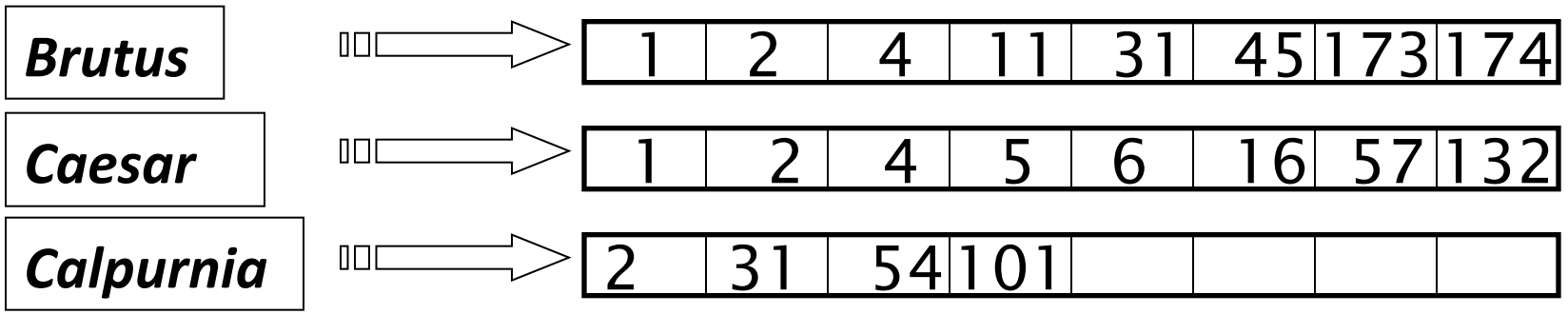
Information Retrieval

The Inverted Index

The key data structure underlying modern IR

Inverted index

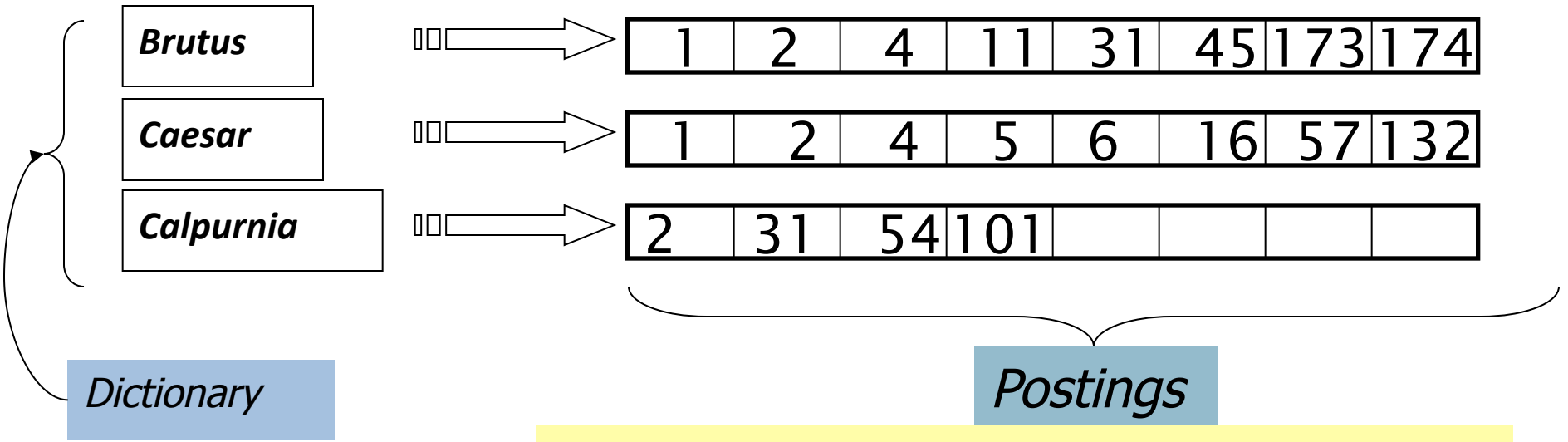
- For each term *t*, we must store a list of all documents that contain *t*.
  - Identify each doc by a **docID**, a document serial number
- Can we use the fixed-size arrays for this?



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

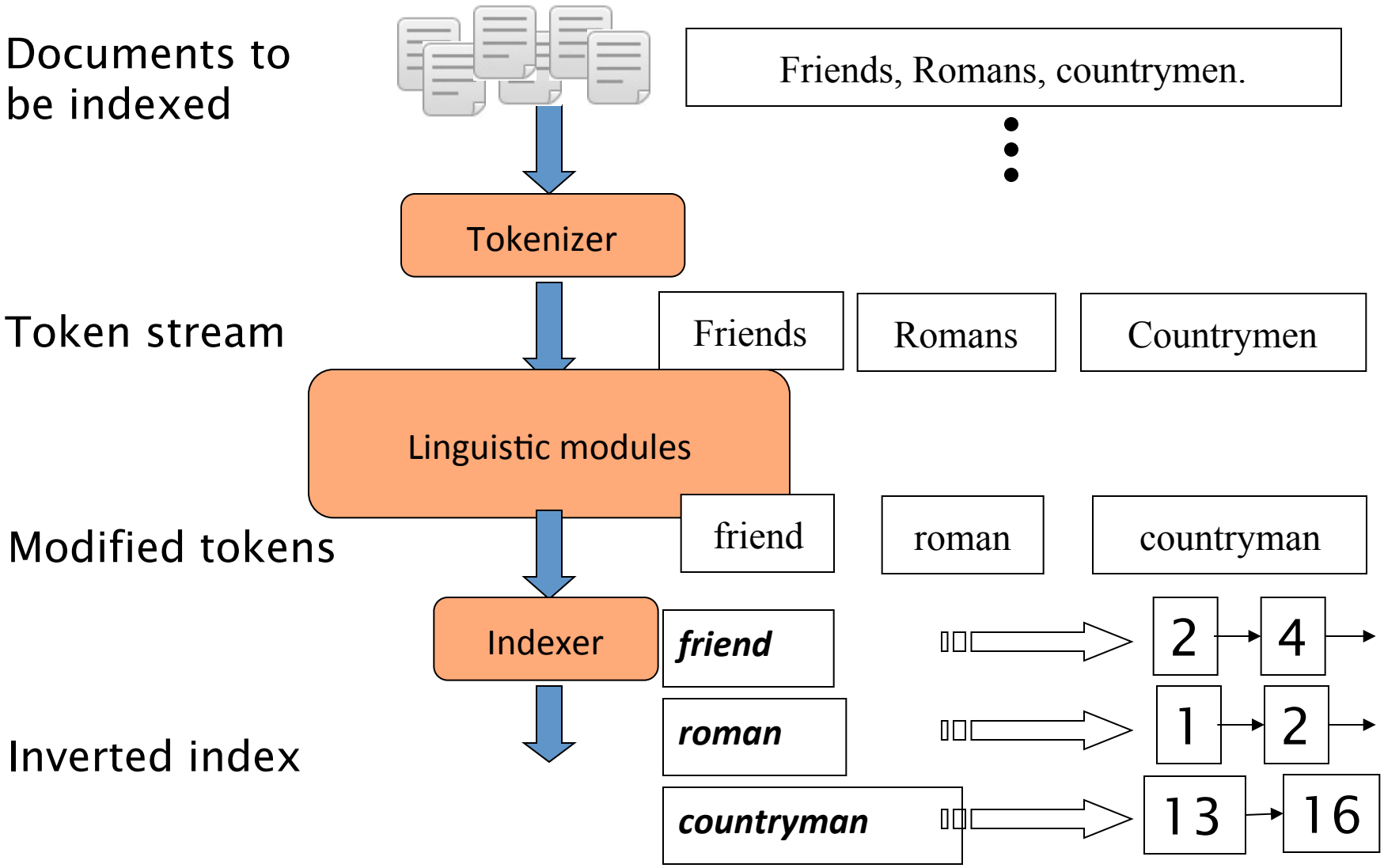
Inverted index

- We need variable-size postings lists.
  - On disk, a continuous run of postings is normal and best
  - 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



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.

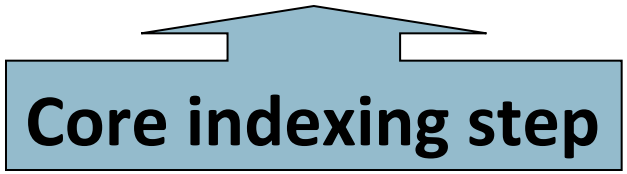
Doc 2

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

Indexer steps: Sort

- Sort by terms
  - And then docID



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

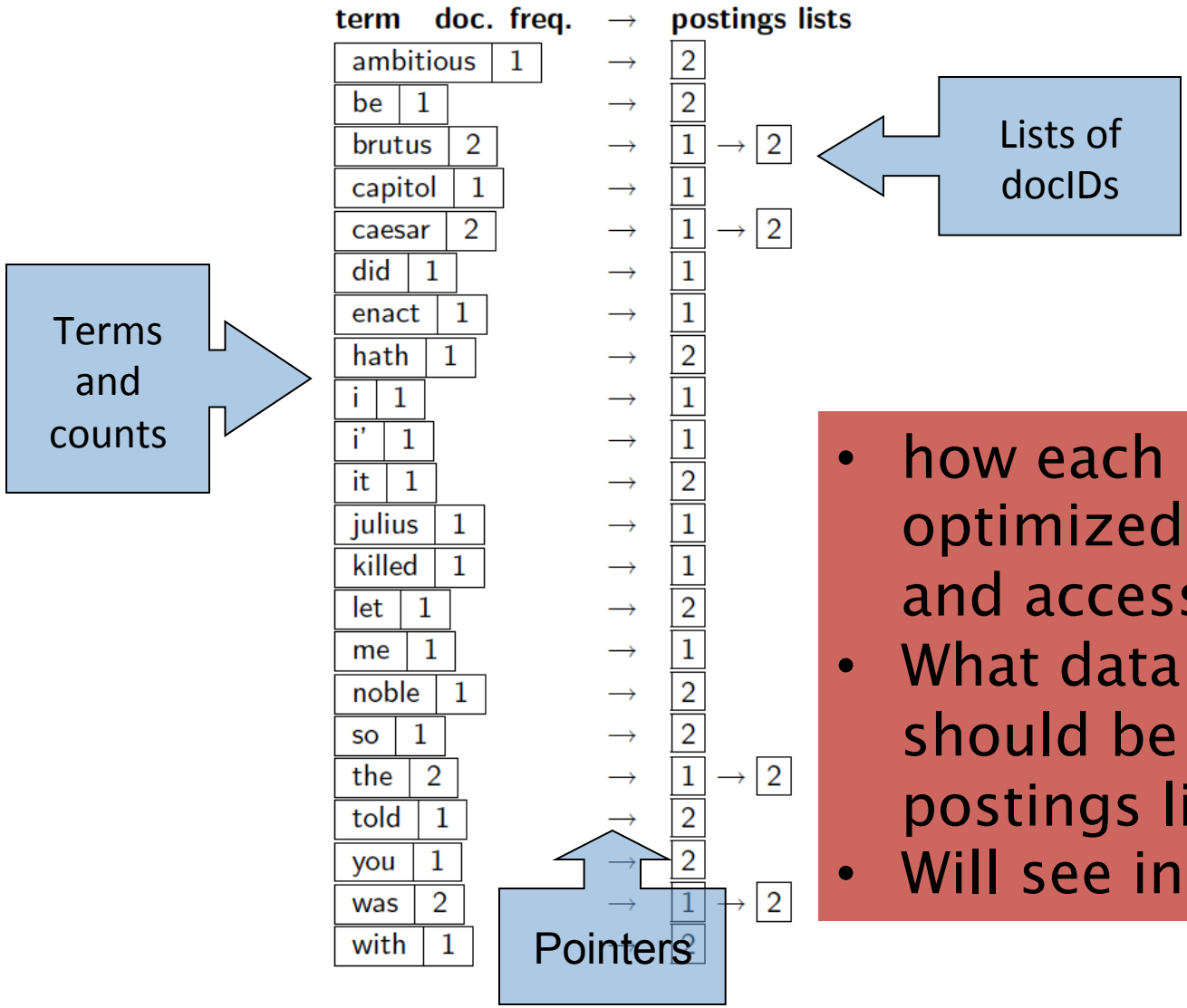
- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings.
- Doc. frequency information is added.

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

Why frequency?
Will discuss later.

Where do we pay in storage?



- how each can be optimized for storage and access efficiency?
- What data structure should be used for a postings list?
- Will see in Ch5

Homework # 1

- **Exercise 1.1** [⋆]
  - Draw the inverted index that would be built for the following document collection. (See Figure 1.3 for an example.)
  - **Doc 1** new home sales top forecasts
  - **Doc 2** home sales rise in july
  - **Doc 3** increase in home sales in july
  - **Doc 4** july new home sales rise

Homework # 1

- **Exercise 1.2** [⋆]
  - Consider these documents:
    - **Doc 1** breakthrough drug for schizophrenia
    - **Doc 2** new schizophrenia drug
    - **Doc 3** new approach for treatment of schizophrenia
    - **Doc 4** new hopes for schizophrenia patients
  - a. Draw the term document incidence matrix for this document collection.
  - b. Draw the inverted index representation for this collection, as in Figure 1.3 (page 7)

Homework # 1

Exercise 1.3 [⋆]

For the document collection shown in Exercise 1.2, what are the returned results for these queries:

a. schizophrenia AND drug

b. for AND NOT(drug OR approach)

Introduction to

Information Retrieval

Query processing with an inverted index

The index we just built

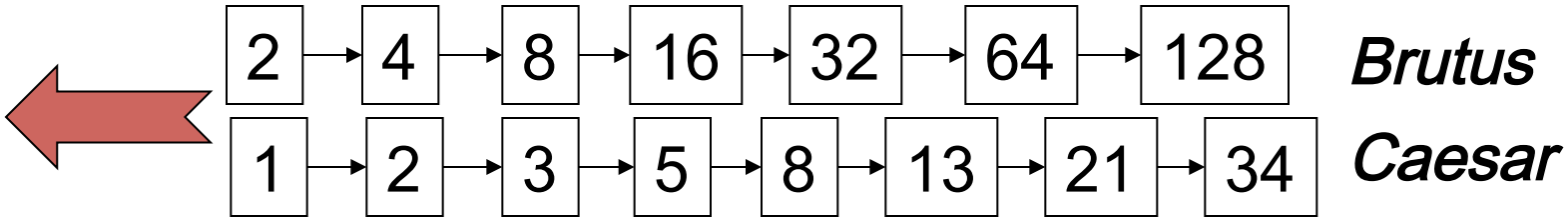
- How do we process a query?
  - Later - what kinds of queries can we process?

Query processing: AND

- Consider processing the query:

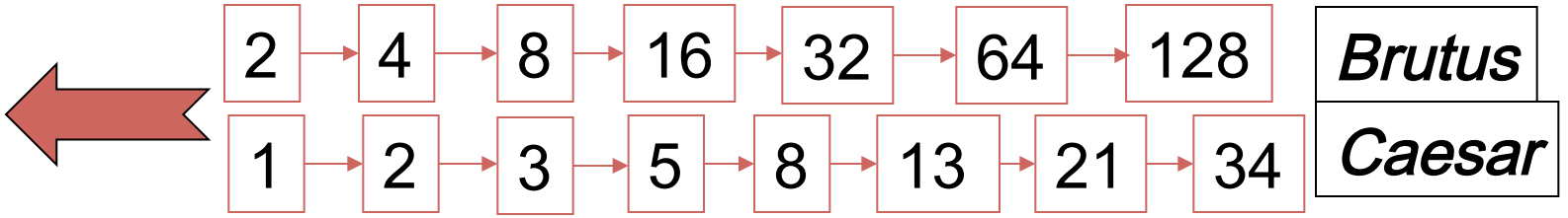
  Brutus *AND* ***Caesar***

  – 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, 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.

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
```

Class Exercise

- **Exercise 1.4 (page 12)**
  For the queries below, can we still run through the intersection in time *O(x+y)*, where *x* and *y* are the lengths of the postings lists for Brutus and Caeser? If not, what can we achieve?
  a. Brutus AND NOT Caeser
  b. Brutus OR NOT Caeser

  Solution a. Page 10 of the book defines the complexity of querying *O(N)* as *O(x +y)* where *x* and *y* are lengths of the postings lists to be intersected. For the given condition Brutus AND NOT Caeser, consider the following postings list

  Case 1 - when the postings list for Brutus has lesser number of postings than that for Caeser:
  - Brutus 1 3 10 21
  - Caesar 1 6 9 23 45 57

  We have to find the set of documents that have Brutus and do not have Caeser. We use the following logic

Class Exercise

Position pointer p1 to the first posting in the postings list for Brutus and pointer p2 to the first posting in postings list for term Caesar.

Compare the DocIDs pointed by each pointer (Compare DocID(p1) and DocID(p2)

1. If DocID(p1) = DocID(p2), then it means that the docID in that posting contains both the terms Brutus and Caeser. We do not want this.

   So move to the next posting in both the lists. Go to point 2 (compare operation).

2. If DocID(p1) < DocID(p2), then it means that DocID(p1) has the term Brutus AND NOT Caeser. This is what we want, so store the DocID(p1) in an answer array

   Move the pointer for term Brutus to the next posting in the list. Go to point 2 (Compare operation)

3. If DocID(p1) > DocID(p2),

   We move the pointer for Caeser to the next posting. Go to point 2 (compare operation).

We run the compare and increment loop till the p1 points to NULL and then we stop the operation.

We do not need to run the operation till p2 points to NULL as we require the DocIDs that have Brutus in it. e.g. we do not have to consider postings 45 and 57 in the list for Caeser.

Answer = 3, 10, 21

Class Exercise

Thus, the Complexity of Querying is *O(x+y1)*, where *x* is the length of the postings list for the term that has to be in the expression and *y1* is the length of the postings list traversed, for the term to be excluded, when x reaches null.

In this case, *O(x+y1)* <= *O(x+y),* where *y1* <= *y*

Case 2 - When postings list for Brutus is greater than that for Caeser.

Brutus 1 5 11 21 45 55
Caeser 1 11 170

Even in this case, the entire length of postings list for Brutus has to be traversed (*x*), only that length of postings list for Caeser has to be traversed (y1), till p1 reaches null, Thus the Complexity of Querying is *O(x +y1)*, where *y1* <= *y*

Class Exercise

Solution b.

For Brutus OR NOT Caeser, we need to find documents having the term Brutus, cannot have Caeser Or not having the term Caeser, can have Brutus or cannot have Brutus.

Brutus	1	5	11	21	45	55
Caeser	1	11	170			
Other	2	10	11	33	34	

Here the entire length (*x*) of postings list for Brutus has to be traversed to find DocIDs that contain Brutus, The entire length (*z*) of postings list for Other has to be traversed. Similarly, The length (*y1*) of postings list for Caeser has to be traversed till *x* and *z* both reach NULL.

Thus the Complexity of Querying is *O(x+y1+z)*, where *y1* <= *y*

Introduction to

Information Retrieval

The Boolean Retrieval Model
& Extended Boolean Models

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 for 3 decades until 1990.
- Many search systems you still use are Boolean e.g. email, library catalog, Mac OS X Spotlight
- Extended Boolean models includes more operators like proximity discussed next.

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

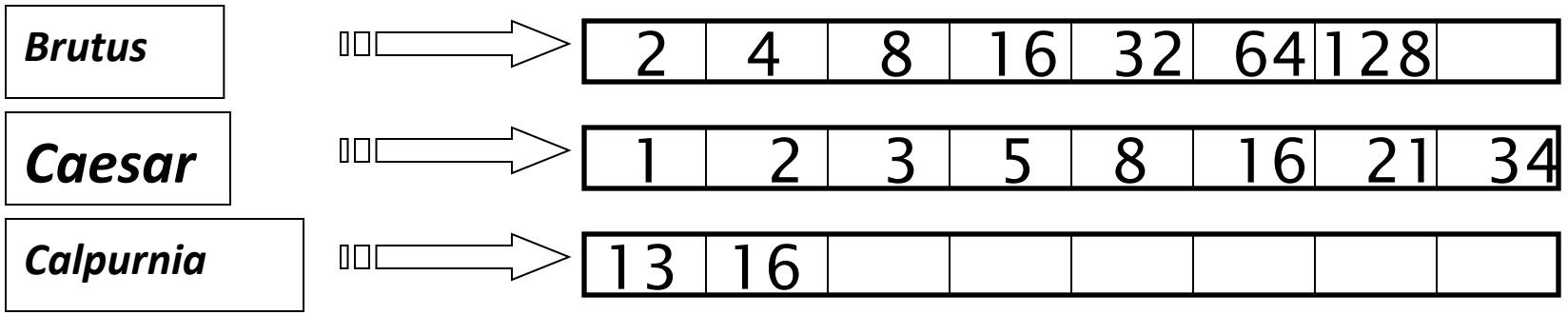
- Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992; new federated search added 2010)
- Tens of terabytes of data; ~700,000 users
- Majority of users *still* use boolean queries
- Example query:
  - What is the statute (written law) of limitations in cases involving the federal tort (wrongful act against the contract) claims act?
  - LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM
    - /3 = within 3 words, /S = in same sentence

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

- Another example query:
  - Requirements for disabled people to be able to access a workplace
  - disabl! /p access! /s work-site work-place (employment /3 place)
- Note that SPACE is disjunction, not conjunction!
- Long, precise queries; proximity operators; incrementally developed; not like web search
- Many professional searchers still like Boolean search
  - You know exactly what you are getting
- But that doesn't mean it actually works better….

Query optimization

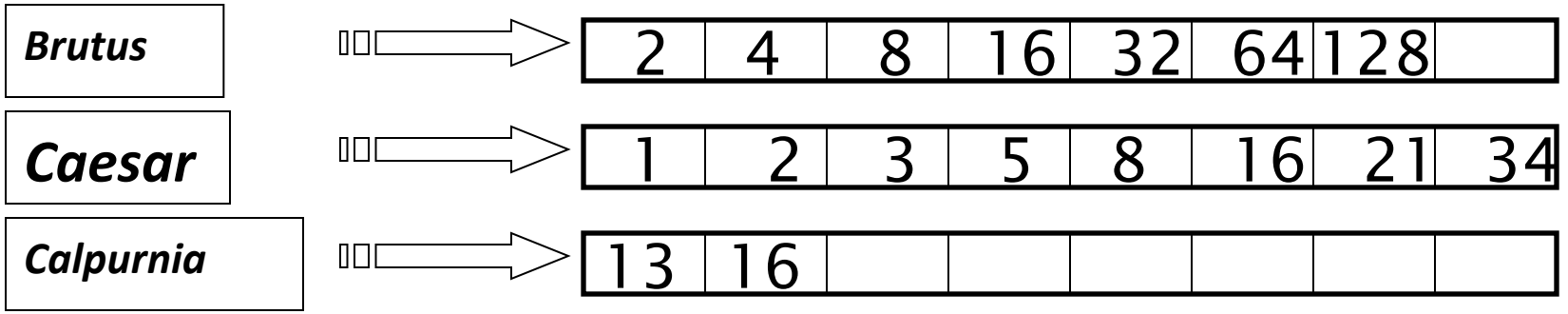
- What is the best order for query processing?
- Consider a query that is an *AND* of *n* terms.
- For each of the *n* terms, get its postings, then *AND* them together.



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 docs freq.'s (conservative).
- Process in increasing order of *OR* sizes.

Class Exercise

- Recommend a query processing order for

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

- Which two terms should we process first?

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

Homework # 1

- Try the search feature at http://www.rhymezone.com/shakespeare/
- Write down five search features you think it could do better

Homework # 1

- Visit the following link and build an IR system basing Inverted index and Boolean search.
- https://github.com/utkarshmankad/inverted-index-boolean-search

Homework # 1

- **Exercise 1.5** [⋆] Extend the postings merge algorithm to arbitrary Boolean query formulas. What is its time complexity? For instance, consider:
  - (Brutus OR Caesar) AND NOT (Antony OR Cleopatra)
- Can we always merge in linear time? Linear in what? Can we do better than this?

Homework # 1

Exercise 1.8 [⋆]

If the query is:

e. friends AND romans AND (NOT countrymen)

how could we use the frequency of countrymen in evaluating the best query evaluation order? In particular, propose a way of handling negation in determining the order of query processing.

Exercise 1.10 [⋆⋆]

Write out a postings merge algorithm, in the style of Figure 1.6 (page 11), for an *x* OR *y* query.

Course Grading

- **For MS:**
  - Mid Term (30%)
  - Final Term (50%)
  - Assignments, Quizzes (20%)
- **For PhD:**
  - Final Term (50%)
  - Assignment, Quizzes (20%)
  - Term Paper (30%)