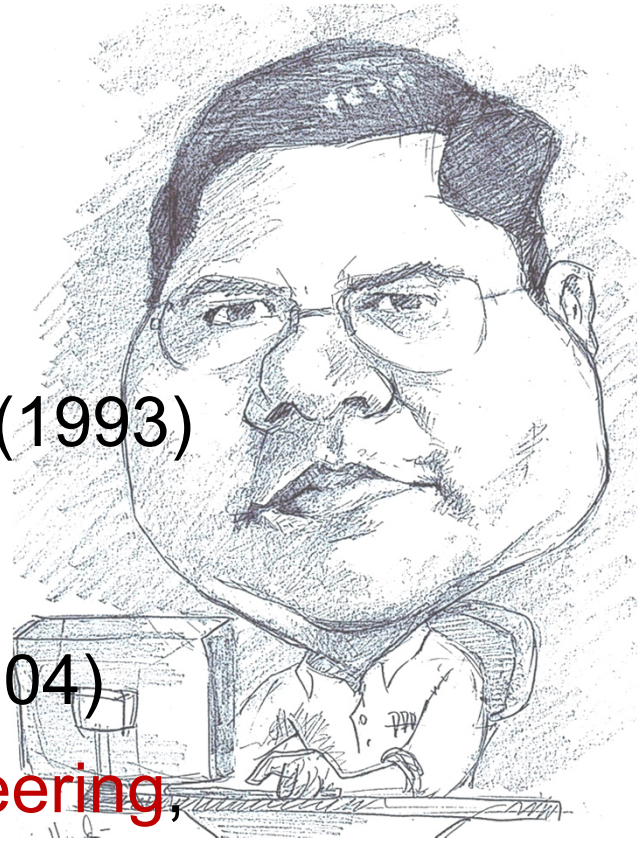
Information Retrieval: Boolean Retrieval

Romi Satria Wahono
romi@romisatriawahono.net
http://romisatriawahono.net
0878-804804-85

BRAINMATICS
IT Training & Consulting

IlmuKomputer.Com
ikatlah ilmu dengan menuliskannya

Romi Satria Wahono



- Lahir di Madiun, 2 Oktober 1974
- SD Sompok Semarang (1987)
- SMPN 8 Semarang (1990)
- SMA Taruna Nusantara, Magelang (1993)
- S1, S2 dan S3 (on-leave)
  Department of Computer Sciences
  Saitama University, Japan (1994-2004)
- Core Competence: Software Engineering, Computational Intelligence
- Founder dan Koordinator IlmuKomputer.Com
- CEO PT Brainmatics Cipta Informatika

Course Contents

1. Introduction
2. Boolean Retrieval
3. The Term Vocabulary
4. Dictionaries and Tolerant Retrieval
5. Index Construction
6. Index Compression
7. Vector Space Model
8. Computing Scores
9. Evaluation in Information Retrieval
10. Relevance Feedback and Query Expansion
11. XML Retrieval

Course Contents

12. Probabilistic Information Retrieval
13. Language Models for Information Retrieval
14. Text Classification and Naive Bayes
15. Vector Space Classification
16. Support Vector Machines and Machine Learning on Documents
17. Flat Clustering
18. Hierarchical Clustering
19. Latent Semantic Indexing
20. Web Search
21. Web Crawling and Indexes
22. Link Analysis

BOOLEAN RETRIEVAL

Shakespeare's Collected Works

- 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? We need more!
  1. Slow (for large corpora) → need to process large
  2. Other operations (e.g., find the word ***Romans*** near ***countrymen***) not feasible → need to allow more flexible matching operations
  3. Without ranked retrieval (best documents to return) → need to allow ranked retrieval

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

- 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

(Brutus) (Caesar) (Not Calpurnia)

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 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?

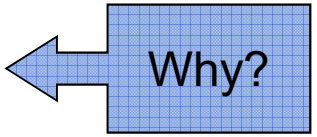
- **Recall**: What fraction of the relevant documents in the collection were returned by the system?

Bigger collections

- Consider *N* = 1 million documents, each with about 1000 words.
- Average 6 bytes/word including spaces and 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-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?

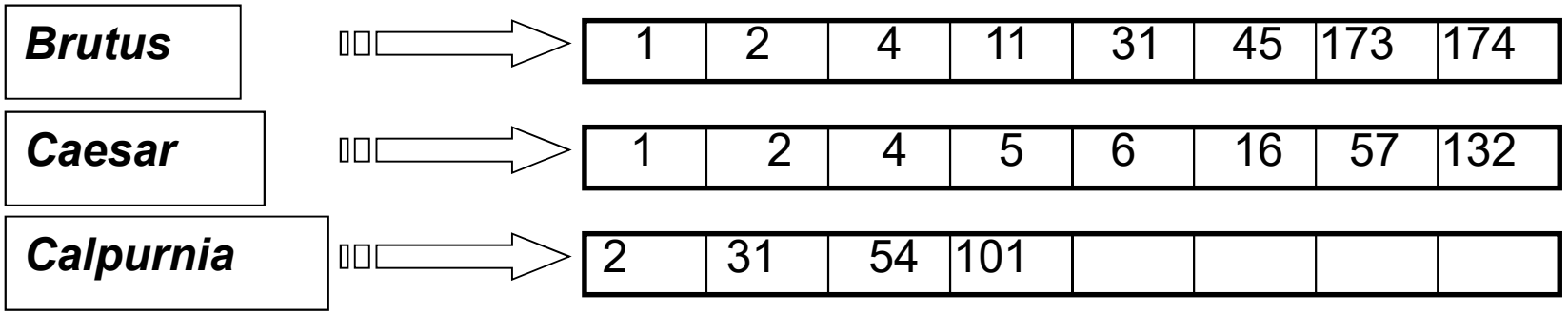


  - We only record the 1 positions

Inverted Index

Inverted Index

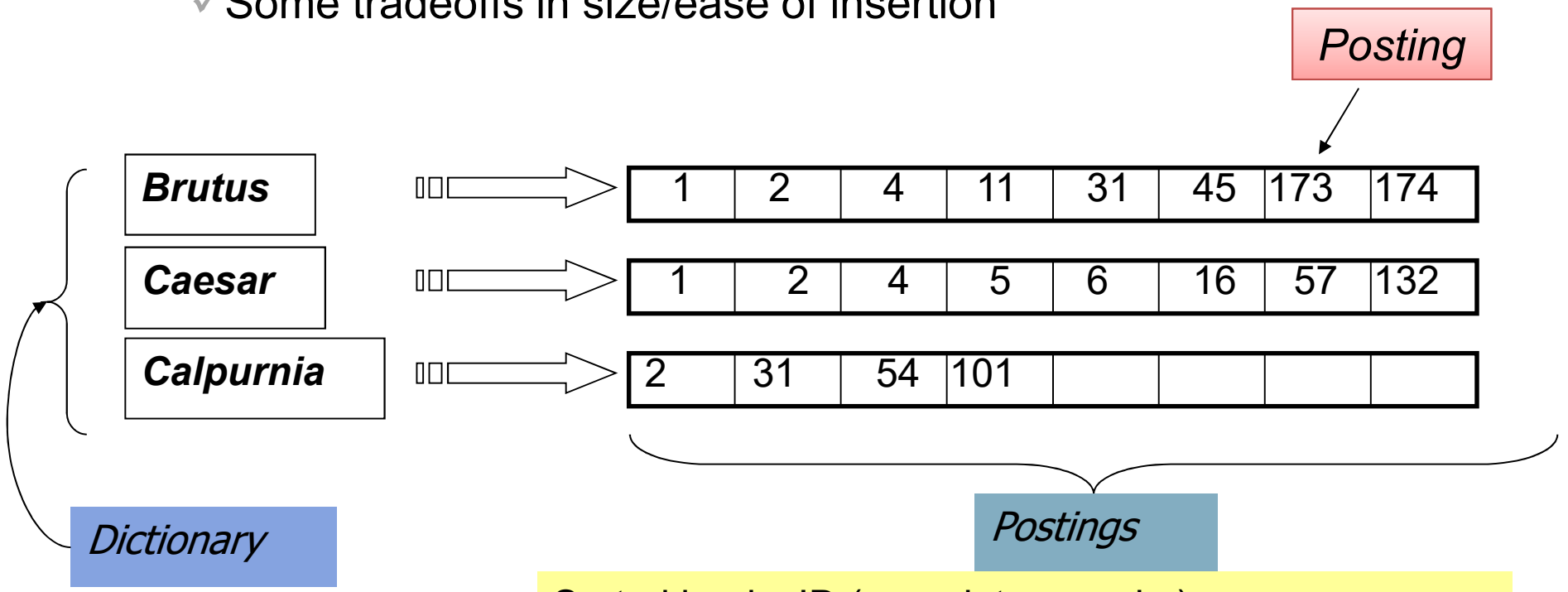
- For each term *t*, we must store a list of all documents that contain *t*
- Identify each by a **docID**, a document serial number
- Can we used fixed-size arrays for this?



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

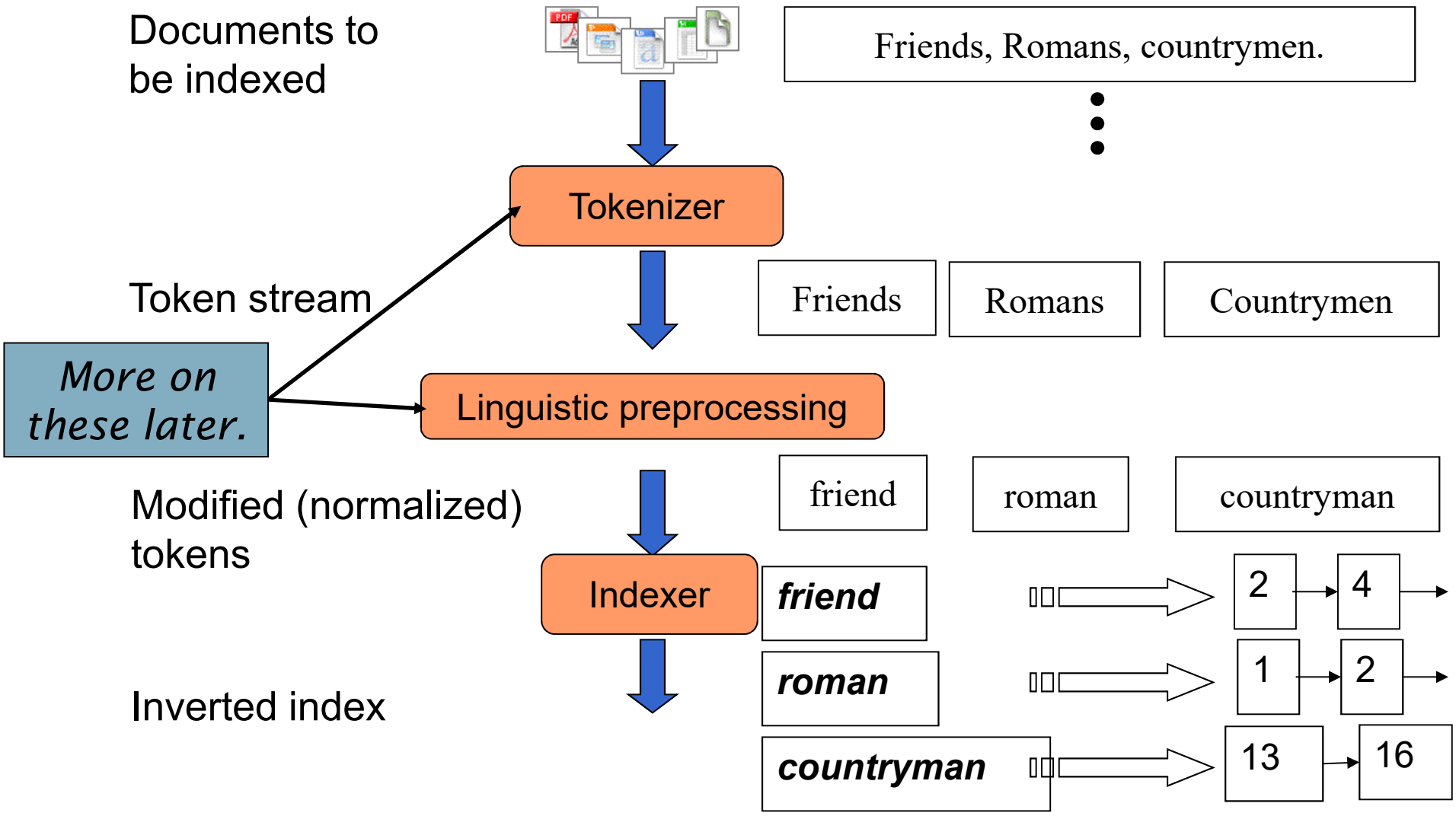
Inverted Index

- 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



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

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

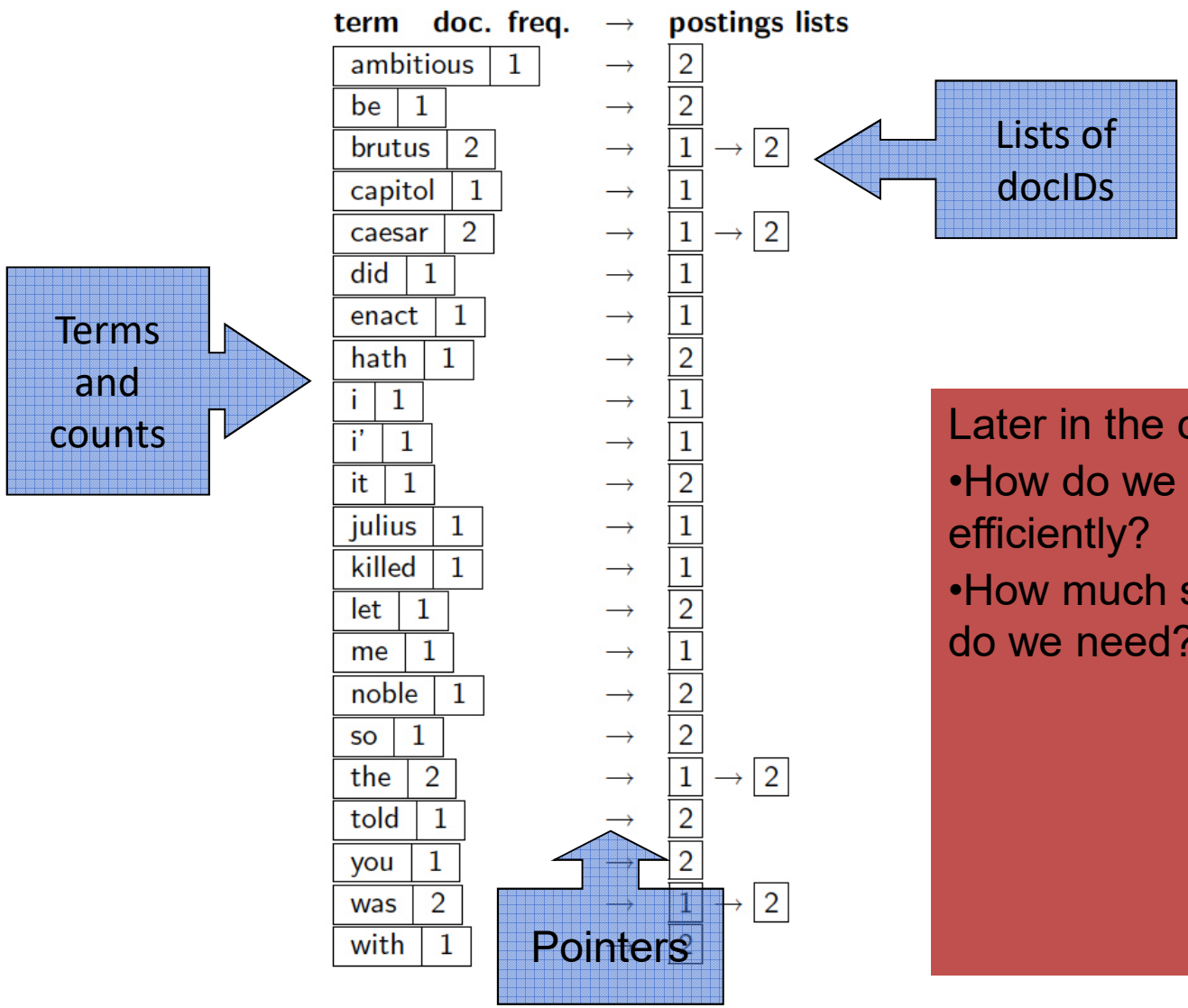
- Multiple term entries in a single document are merged
- Split into Dictionary and Postings
- Document frequency information is added

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?



Later in the course:

- How do we index efficiently?
- How much storage do we need?

The Index We Just Built

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

Exercise: term-document incidence matrix

- Draw the term-document incidence matrix for this document collection

  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

- Tunjukkan hasil retrieval untuk query boolean: Schizophrenia AND drug

Exercise: Inverted Index

- Draw the inverted index representation for this collection

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

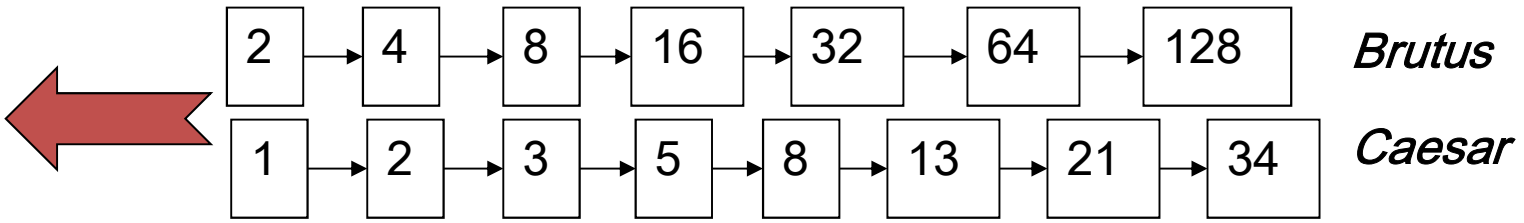
Processing Boolean Queries

Query Processing: AND

- Consider processing the query:

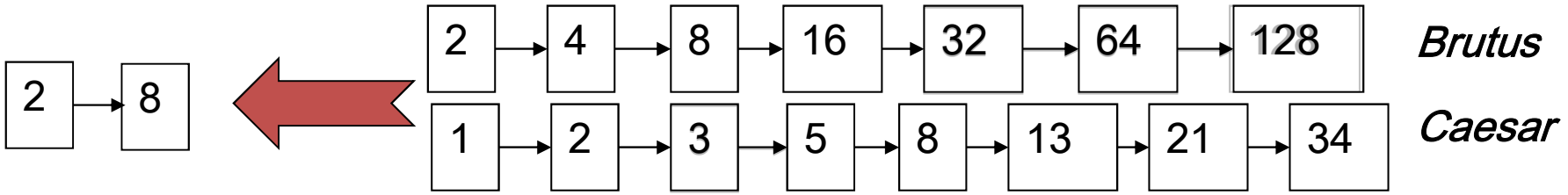
  Brutus *AND* ***Caesar***

  1. Locate ***Brutus*** in the Dictionary
     - ✓ Retrieve its postings.
  2. Locate ***Caesar*** in the Dictionary
     - ✓ Retrieve its postings.
  3. "Merge" the two postings



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
```

Exercise: Query Processing

- For the document collection shown in the following

  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

- what are the returned results for these queries:

  1. schizophrenia AND drug
  2. for AND NOT(drug OR approach)

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
- Many search systems you still use are Boolean:
  - Email, library catalog, Mac OS X Spotlight

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 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**
  - /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….

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?

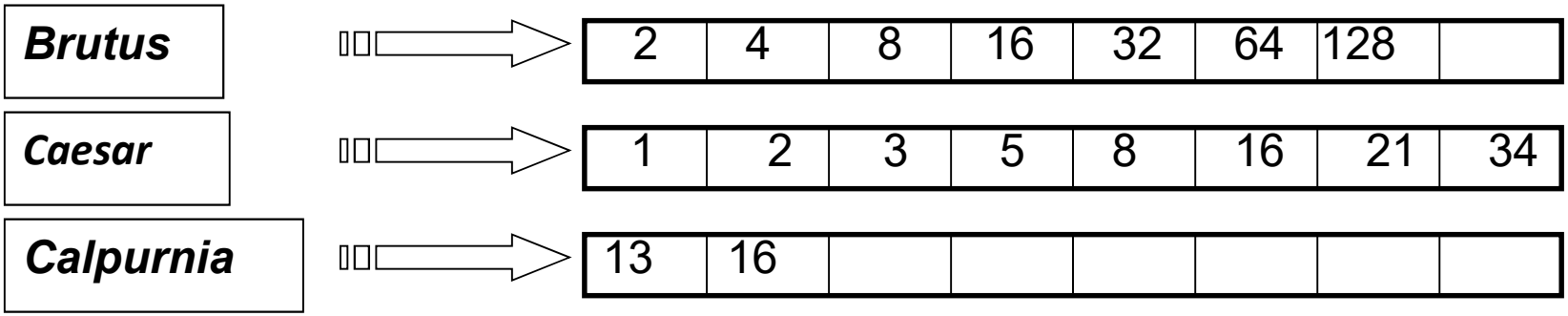
***(Brutus** OR **Caesar)** AND NOT*

(Antony** OR **Cleopatra)

- Can we always merge in “linear” time?
  - Linear in what?
- Can we do better?

Query Optimization

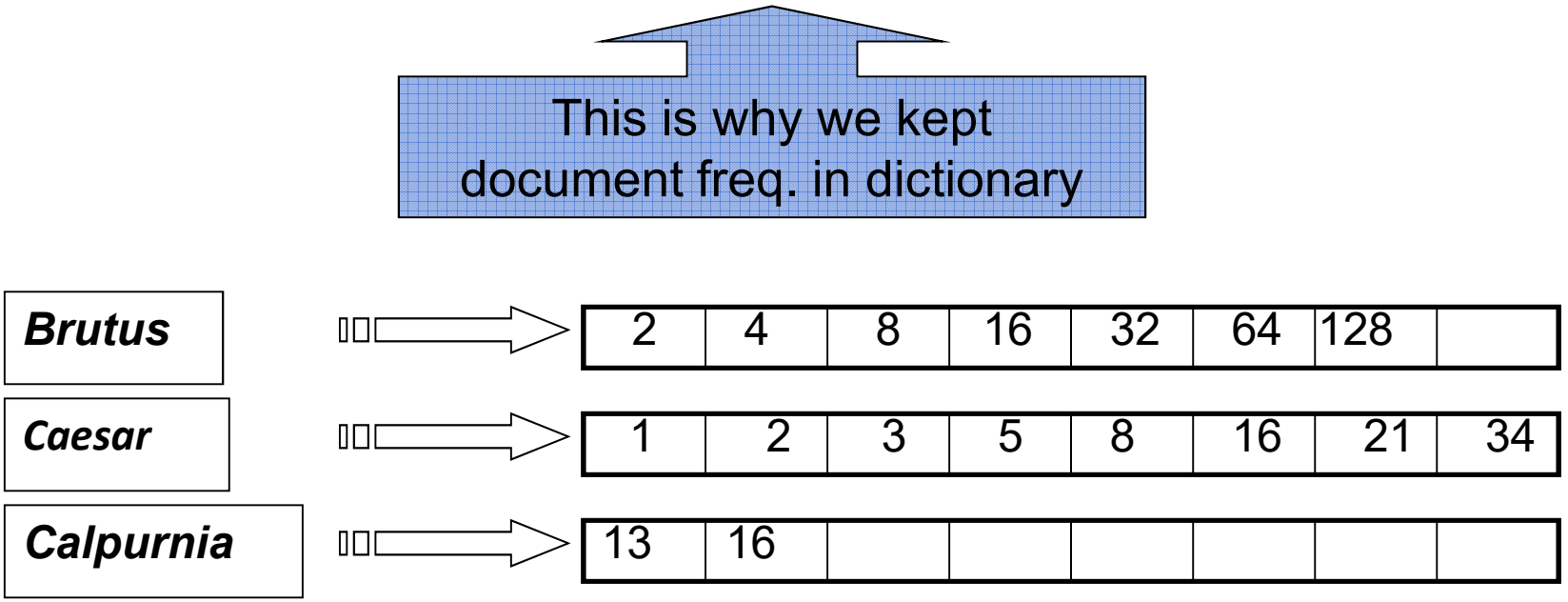
- What is the best order for query processing?
- Consider a query that is an *AND* of *n* terms.
- For each of the *n* terms, get its postings, then *AND* them together.



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

Exercise: Query Processing Order

- Recommend a query processing order for

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

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

Query processing exercises

- Exercise: If the query is ***friends*** *AND* ***romans*** *AND (NOT* ***countrymen****)*, how could we use the freq of ***countrymen***?
- Exercise: Extend the merge to an arbitrary Boolean query. Can we always guarantee execution in time linear in the total postings size?
- Hint: Begin with the case of a Boolean *formula* query: in this, each query term appears only once in the query.

Exercise

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

References

1. Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze, Introduction to Information Retrieval, *Cambridge University Press*, 2008
2. Stefan Büttcher, Charles L. A. Clarke, and Gordon V. Cormack, Information Retrieval: Implementing and Evaluating Search Engines, *The MIT Press*, 2010
3. Bruce Croft, Donald Metzler, and Trevor Strohman, Search Engines: Information Retrieval in Practice, *Addison Wesley*, 2009
4. David A. Grossman and Ophir Frieder, Information Retrieval: Algorithms and Heuristics 2nd edition, *Springer*, 2004
5. Charles T. Meadow, Bert R. Boyce, Donald H. Kraft, and Carol L Barry, Text Information Retrieval Systems Third Edition, Library and Information Science, 2007