EECS 395/495 Lecture 2: Intro to IR and Inverted Indices

Based on slides by Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze, and Dan Weld

Reminder

- Project Proposals Due next Monday
  - April 11, 11:59PM
- Sign up for project meetings April 12 and 13
  - Link to schedule will be sent via e-mail

Goals

- **How does Web search work?**
  - **Tuesdays**
- What’s the future of Web search?
  - Thursdays

How Does Web Search Work?

- **Inverted Indices**
  - **Inverted Indices enable fast IR**
  - **Tokenization & Linguistic issues**
  - **Handling phrase queries**
  - Scalable, distributed construction & deployment
- Web Crawlers
- Ranking Algorithms

Information Retrieval (IR) Example

- 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)
  - Other operations (e.g., find the word ***Romans*** near ***countrymen***) not feasible
  - Ranking? (best documents to return)
    - Later lectures

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

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.

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 <u>*distinct*</u> 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.

Why?

  – matrix is extremely sparse.
- What’s a better representation?
  – We only record the 1 positions.

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
…	
so	2
let	2
it	2
be	2
with	2
caesar	2
the	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
…	
so	2
let	2
it	2
be	2
with	2
caesar	2
the	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

- 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

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
…		

- The result is split into a *Dictionary* file 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	Coll 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?

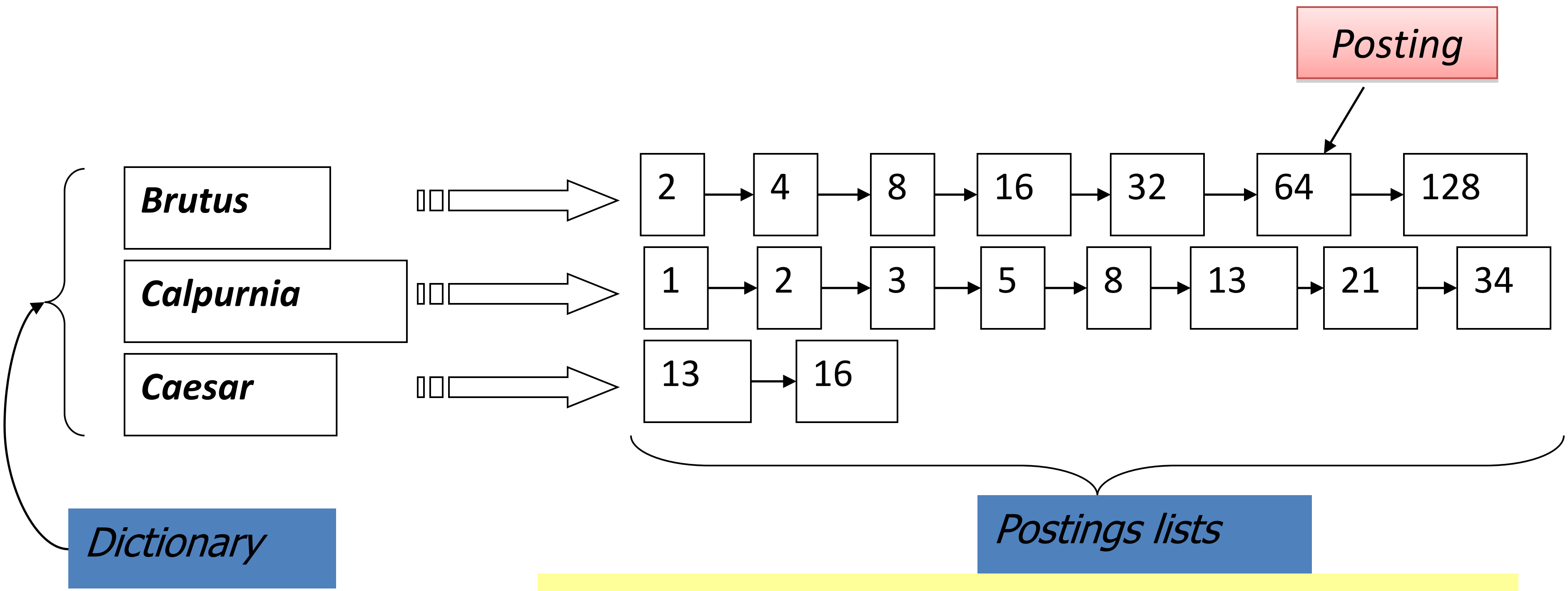
Term	N docs	Coll freq
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
i'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	2	2
told	1	1
you	1	1
was	2	2
with	1	1

Terms

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

Pointers

Inverted index



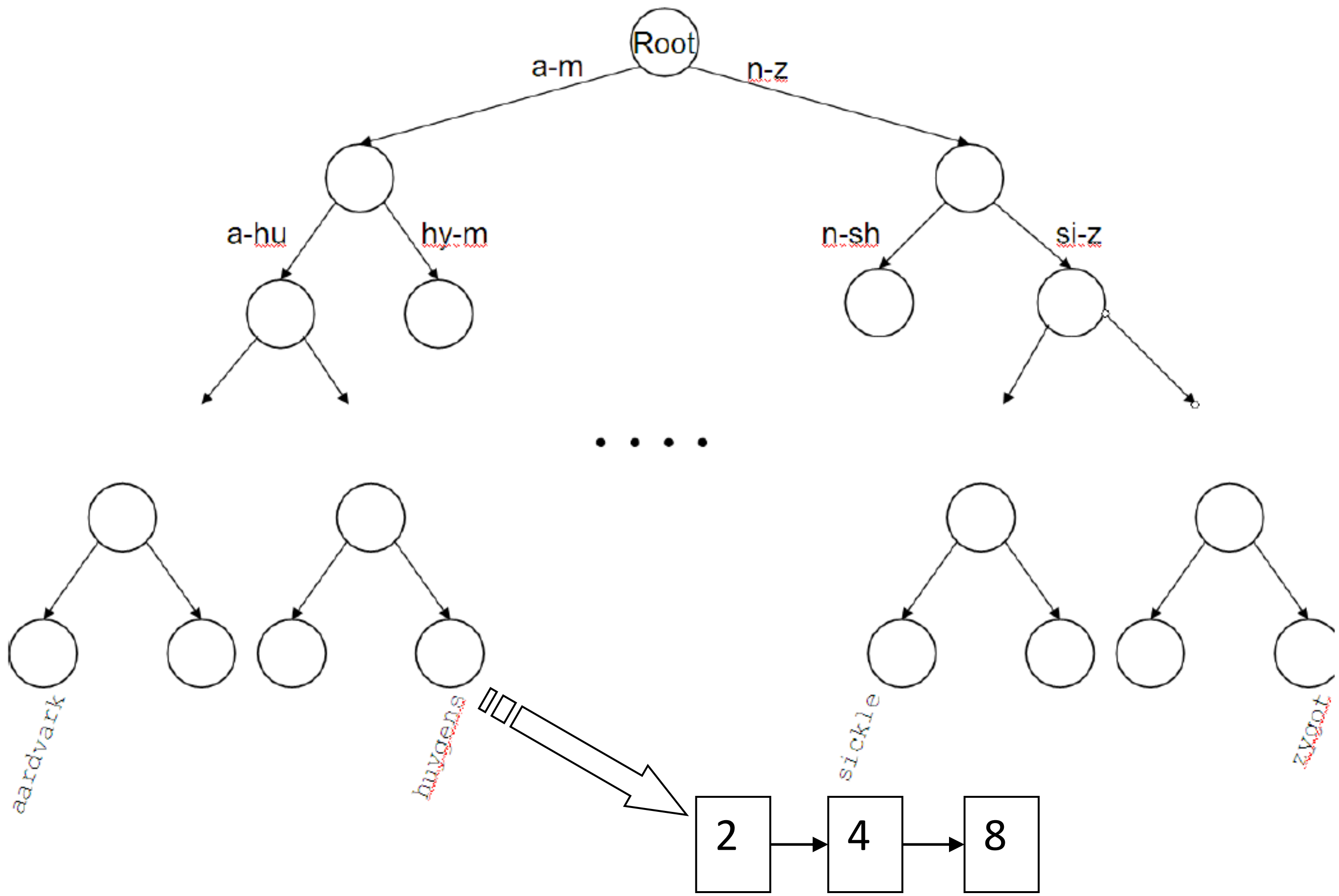
Sorted by docID (more later on why).

The index we just built

- How do we process a query?
  - Look up each query word in the dictionary
    - Retrieve postings lists
  - Merge

Looking up words in dictionary

Use an auxiliary data structure (e.g. hash table, search tree)



Query processing: AND

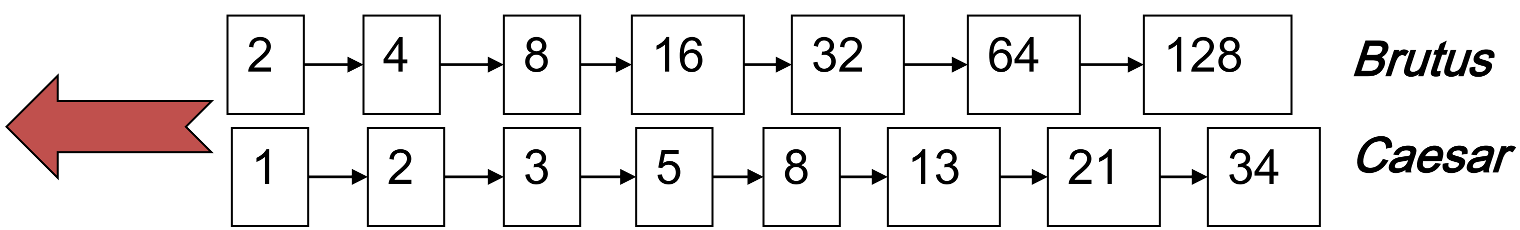
- Consider processing the query:

  Brutus *AND* ***Caesar***

  – Locate ***Brutus*** in the Dictionary;
    - Retrieve its postings.

  – Locate *Caesar* in the Dictionary;
    - Retrieve its postings.

  – "Merge" the two postings:



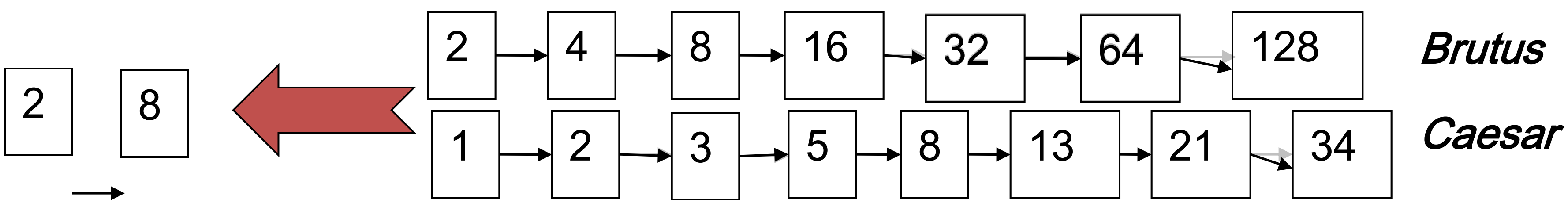
Naive Merge

- For each DocID *d1* in list 1
  - For each DocID *d2* in list 2
    - If *d1* = *d2*, output the match

- For lists of lengths *x* and *y*, this merge takes time proportional to *xy*.
  - US: 5 billion hits, China: 1 billion hits
  - US AND China: 5,000,000,000,000,000,000 operations?
    - Not promising

Better merge

- Idea: Walk through the lists simultaneously
- Loop until you're at the end of both lists:
  - If current DocIDs match, output the DocID
  - Step forward on the list w/lowest current DocID



For **sorted** lists of lengths *x* and *y*, a merge takes O(*x*+*y*) operations.

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?

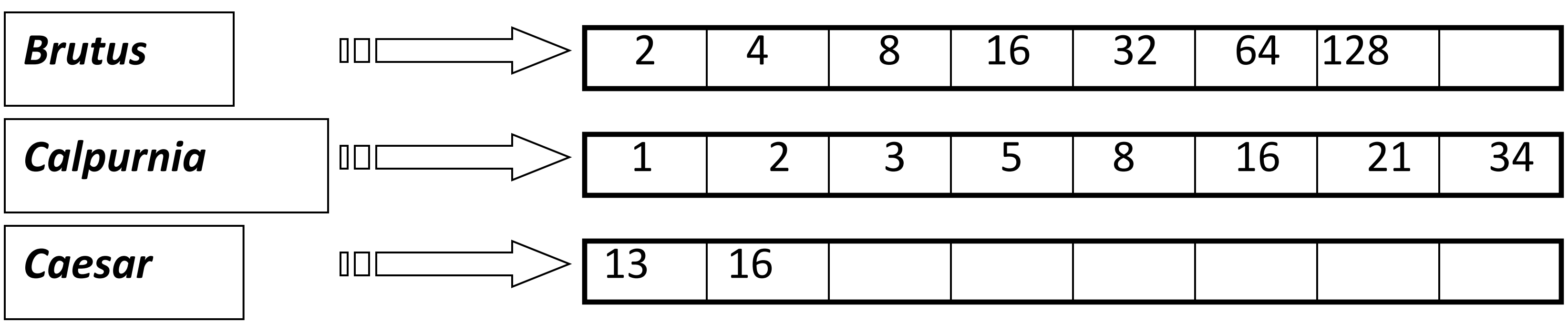
(Brutus *OR* ***Caesar)*** *AND NOT*

(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 *t* terms.
- For each of the *t* 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*.

Brutus	2	4	8	16	32	64	128	
Calpurnia	1	2	3	5	8	13	21	34
Caesar	13	16						

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

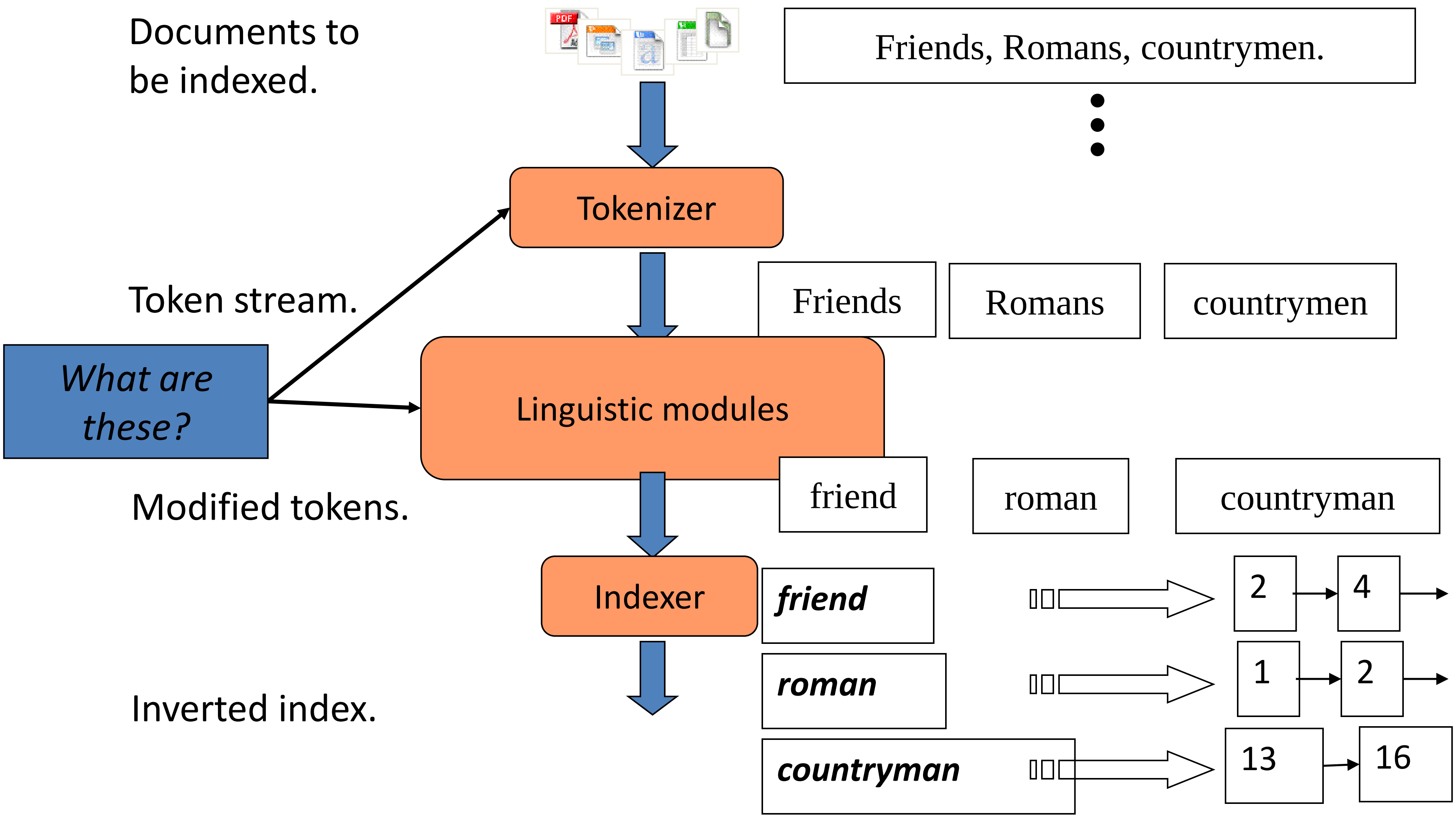
Query processing exercises

- 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.

Outline

- Inverted Indices
  - How they enable fast IR
  - **Tokenization & Linguistic issues**
  - Handling phrase queries

Inverted index construction



Tokenization

- Input: “***Friends, Romans and Countrymen***”
- Output: Tokens
  - ***Friends***
  - ***Romans***
  - ***Countrymen***
- What tokens to emit?

Issues in Tokenization

- ***Finland's capital*** $\rightarrow$

  Finland? Finlands? Finland's?
- ***Hewlett-Packard*** $\rightarrow$
  Hewlett and ***Packard*** as two tokens?
  - What about ***co-education*** ?
- ***San Francisco***: one token or two? How do you decide it is one token?

Stop words

- Exclude from dictionary entirely the commonest words? Intuition:
  - Lose little semantic content *the, a, and, to, be*
  - Gain space (~30% of postings for top 30 wds)
- Not typically done (anymore)
  - Compression & optimization techniques (later) means cost for including stop words is low
  - You need them for:
    - Phrase queries: “King of Denmark,” “Flights to Orlando”

Normalization

- Need to "normalize" terms in indexed text as well as query terms into the same form
  - We want to match ***U.S.A.*** and ***USA***
  - ***co-education*** and ***coeducation***
- Ideal: asymmetric expansion
  - Enter: ***window*** Search: ***window, windows***
  - Enter: ***windows*** Search: ***Windows, windows, window***
  - Enter: ***Windows*** Search: ***Windows***
- More powerful; but less efficient, hard to define

Case folding

- Reduce all letters to lower case
  - exception: upper case in mid-sentence?
    - e.g., ***Fed*** vs. ***fed***
      - “we Fed officials” != “we fed officials”
    - ***US*** vs ***us***
  - But…users submit lowercase regardless of ‘correct’ capitalization

Thesauri and Soundex

- Handle synonyms and homonyms
  - Hand-constructed equivalence classes
    - e.g., ***car*** = ***automobile***
    - ***your ≠ you're***

- Index such equivalences?
- Or expand query?
  - More later ...

Stemming

- Reduce terms to their “roots” before indexing
  - e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.
- Language dependent

for example compressed and compression are both accepted as equivalent to compress.

→

for exampl compress and compress ar both accept as equival to compress

Porter's algorithm

- Most common stemmer for English
  - Available: http://tartarus.org/~martin/PorterStemmer/
- *Rules* applied sequentially in phases, e.g.:
  - *sses* $\rightarrow$ *ss*
  - *ies* $\rightarrow$ *i*
  - *ational* $\rightarrow$ *ate*
  - *tional* $\rightarrow$ *tion*

Challenges

- Sandy
- Sanded

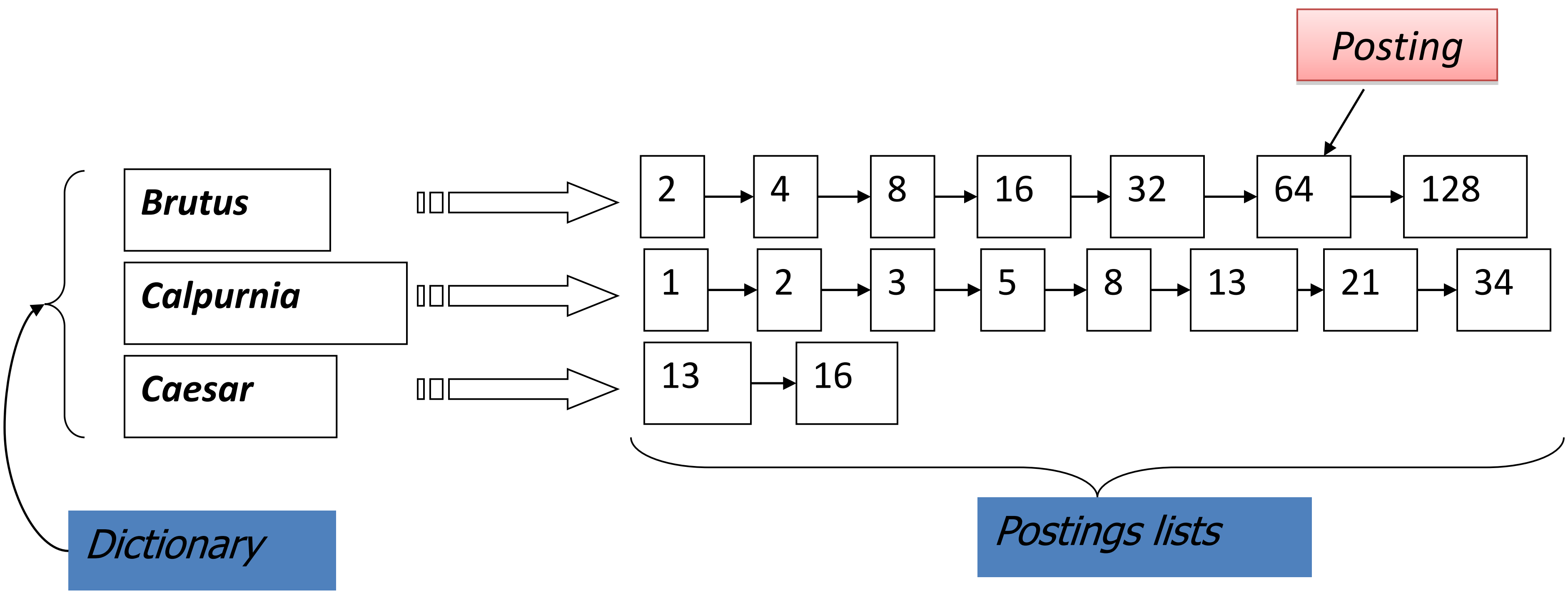
➔ Sand ???

Outline

- Inverted Indices
  - How they enable fast IR
  - Tokenization & Linguistic issues
  - **Handling phrase queries**

Phrase Queries

Our existing inverted index **can't** be used to process phrase queries…



Solution: Positional indexes

- In the postings, store, for each ***term***, entries of the form:

  <***term**, number of docs containing* ***term***;

  doc1: position1, position2 … ;

  doc2: position1, position2 … ;

  etc.>

Positional index example

<***be***: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367, …>

Which of docs 1,2,4,5 could contain "***to be or not to be***"?

- Similar merge algorithm
  – But we now need to deal with more than just equality

Processing a phrase query

- Extract inverted index entries for each distinct term: ***to, be, or, not.***
- Merge their *doc:position* lists to enumerate all positions with "***to be or not to be***".
  - ***to**:*
    - *2*:1,17,74,222,551; *4*:8,16,190,429,433; *7*:13,23,191; ...
  - ***be**:*
    - *1*:17,19; *4*:17,191,291,430,434; *5*:14,19,101; ...
- Same general method for proximity searches

Positional index size (1)

- Need an entry for each **occurrence**, not just once per **document**
- Index size depends on average document size
  - Average web page has <1000 terms
  - SEC filings, books, even some epic poems … easily 100,000 terms
- Consider a term with frequency 0.1%

Document size	Postings	Positional postings
1000	1	1
100,000	1	100

Positional Index Size (2)

- Rules of thumb (for English-like languages):
  - A positional index is 2–4 times as large as a non-positional index
  - Positional index size 35–50% of volume of original text
- Worth the space
  - Quotes are the one advanced search operator that people actually use
  - Phrases also useful *implicitly* for ranking

Reminder

- Project Proposals Due next Monday
  – April 11, 11:59PM
- Sign up for project meetings April 12 and 13
  – Link to schedule to be sent via e-mail