# Information Retrieval

CS 547/DS 547

Worcester Polytechnic Institute
Department of Computer Science
Instructor: Prof. Kyumin Lee

# Project

- 3~4 person team
- Notify names of your project team members by February 2
- Dates:
  - [7%] Project Proposal Writing: March 17 by 11:59pm
  - [5%] Project Proposal Presentation: March 21
  - [8%] Project website: April 25 by 11:59pm
  - [11%] Project Workshop: April 26 in-class
- https://canvas.wpi.edu/courses/46542/pages/project?module_item_id=888446

# Previous Year's Projects

- https://exquisite-chebakia-9a513b.netlify.app/
- https://newsinspector.github.io/
- https://sites.google.com/view/newsbaordrecommender/home
- https://kratikashetty.github.io/CS547-Information-Retrieval/
- https://wheeleddoors.github.io/index/
- https://yelp-recommendation.ue.r.appspot.com/report
- https://github.com/khordoo/disaster-watch-classifier

# Previous Class…

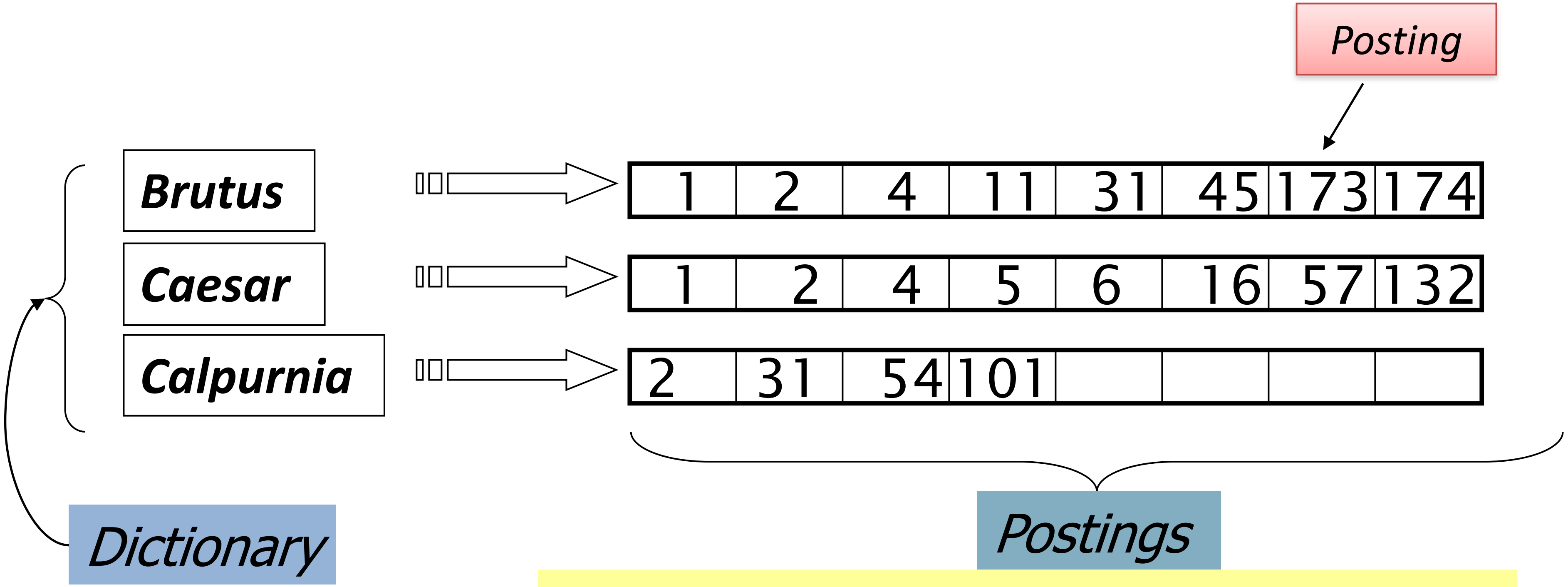
Boolean Retrieval Model

# Previous Class…

Boolean Retrieval Model

Inverted index

# Inverted index




Sorted by docID (more later on why).

# Previous Class…

Query Optimization

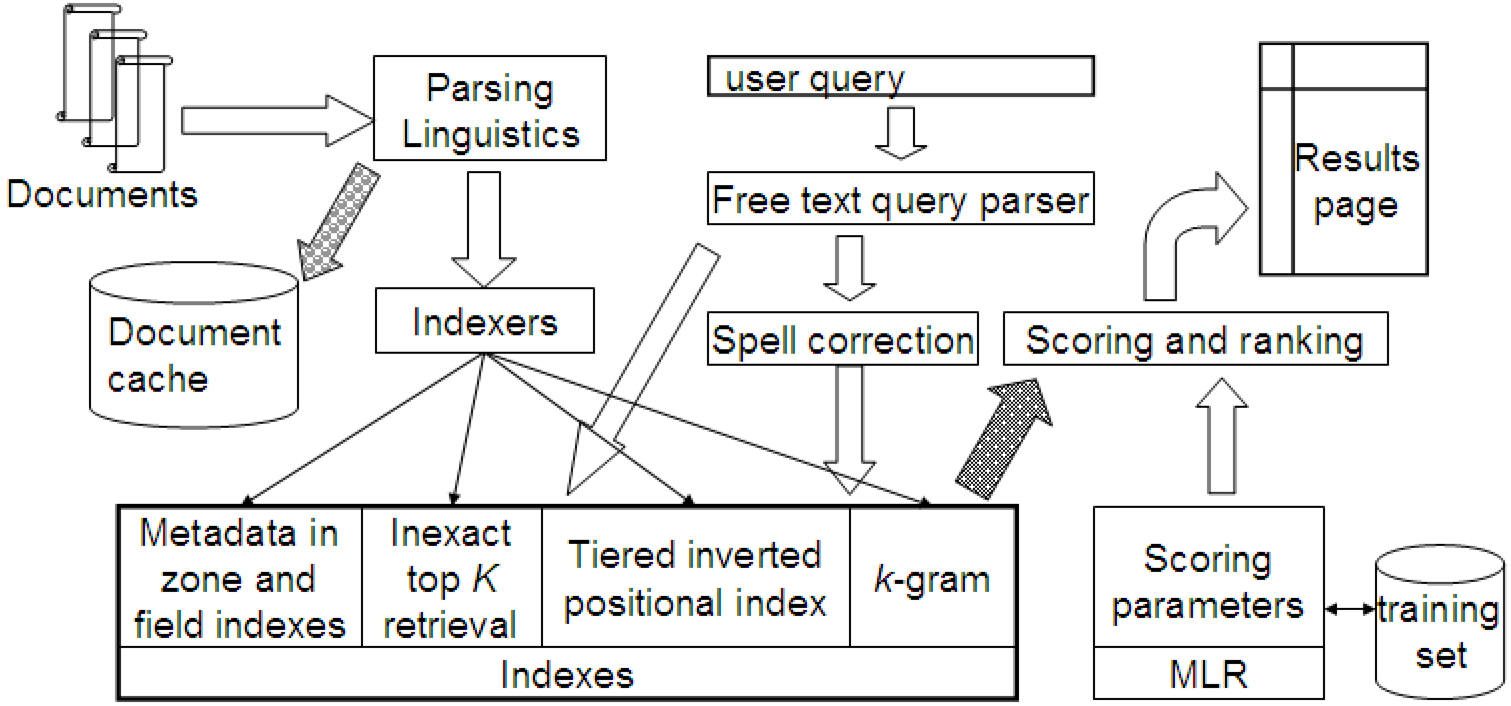
# Query optimization

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

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

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

# What’s next …



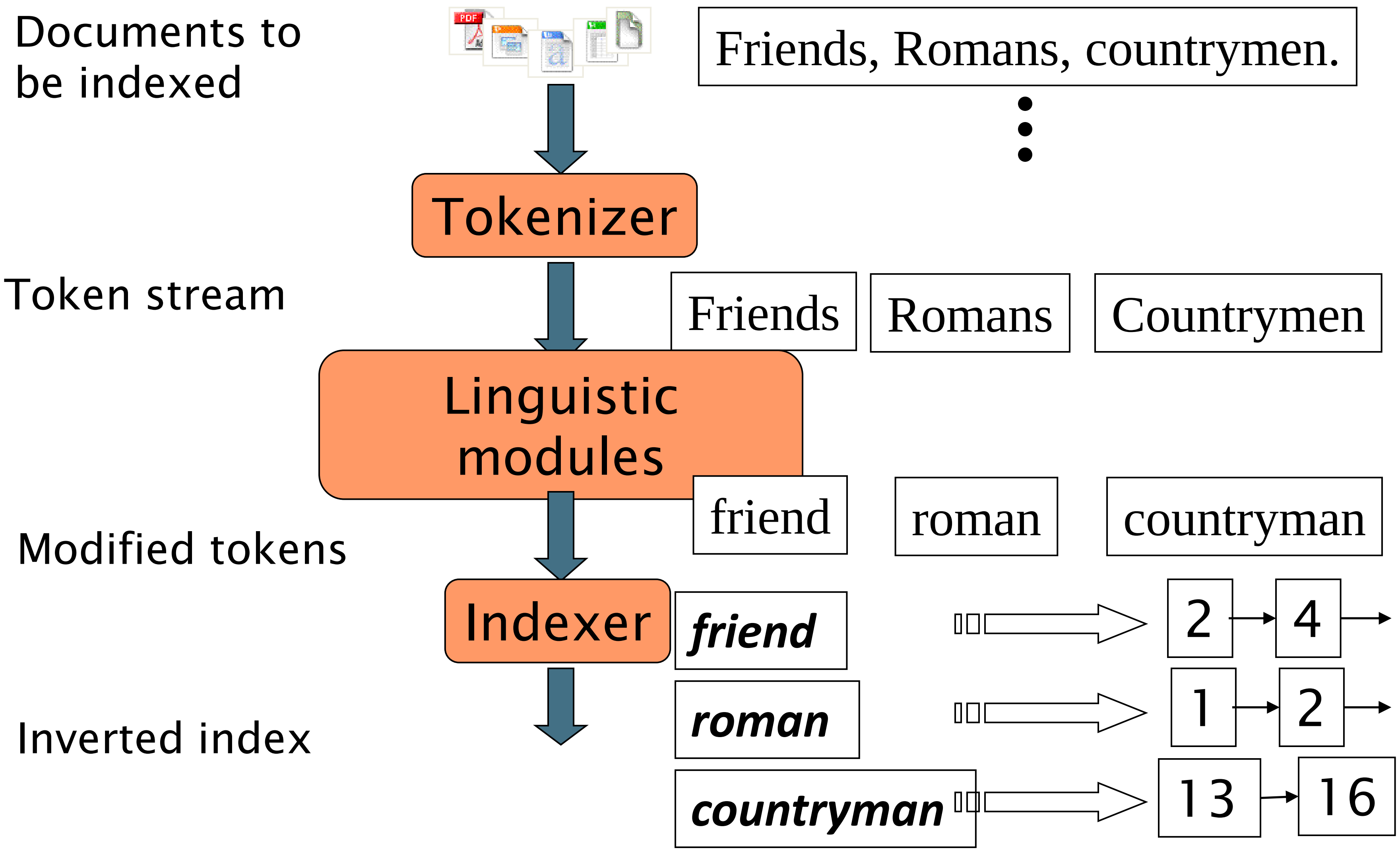
Documents
Parsing
Linguistics
user query
Free text query parser
Results
page
Document
cache
Indexers
Spell correction
Scoring and ranking
Metadata in
zone and
field indexes
Inexact
top K
retrieval
Tiered inverted
positional index
k-gram
Indexes
Scoring
parameters
MLR
training
set

# Our assumptions so far

- We know what a document is
- We know what a term is
  - In reality, it can be complex

- So… We'll look at how we define and process the vocabulary of terms in a collection

# Recall the basic indexing pipeline



# 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***

# Parsing a document

- What format is it in?
  - pdf/word/excel/html?
- What language is it in?
- What character set is in use?
  - (CP1252, UTF-8, …)

Each of these is a classification problem, which we will study later in the course.

But these tasks are often done heuristically …

# Tokenization

- Input: "***Friends, Romans and Countrymen***"
- Output: Tokens
  - ***Friends***
  - ***Romans***
  - ***Countrymen***
- A token is an instance of a sequence of characters
- Each such token is now a candidate for an index entry, after further processing
  - Described below
- But what are valid tokens to emit?

# Why tokenization is difficult
# -- even in English

- Example: *Mr. O’Neill thinks that the boys’ stories about Chile’s capital aren’t amusing.*
- **Tokenize this sentence**

# One word or two? (or several)

- Hewlett-Packard
- State-of-the-art
- co-education
- the hold-him-back-and-drag-him-away maneuver
- data base
- San Francisco
- Los Angeles-based company
- cheap San Francisco-Los Angeles fares
- York University vs. New York University

# Numbers

- 3/12/91
- 12/3/91
- Mar 12, 1991
- B-52
- 100.2.86.144
- (800) 234-2333
- 800.234.2333

# Chinese: No whitespace

莎拉波娃现在居住在美国东南部的佛罗里达。今年 4 月 9 日，莎拉波娃在美国第一大城市纽约度过了 1 8 岁生日。生日派对上，莎拉波娃露出了甜美的微笑。

# Bidirectionality in Arabic

- Arabic (or Hebrew) is basically written right to left, but with certain items like numbers written left to right
- Words are separated, but letter forms within a word form complex ligatures

استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.

- ‘Algeria achieved its independence in 1962 after 132 years of French occupation.’

# Normalization

- Need to “normalize” words in indexed text as well as query words into the same form
  - We want to match ***U.S.A.*** and ***USA***
- We most commonly implicitly define **equivalence classes** of terms
  - e.g., deleting periods to form a term
- Alternative is to do asymmetric expansion:
  - Enter: *window*  Search: *window*, *windows*
  - Enter: *windows*  Search: *Windows*, *windows*
  - Enter: *Windows*  Search: *Windows*
- Potentially more powerful, but less efficient

# Case folding

- Reduce all letters to lower case
  - exception: upper case in mid-sentence?
    - e.g., General Motors
    - Fed vs. fed
    - SAIL vs. sail
  - Often best to lower case everything, since users will use lowercase regardless of ‘correct’ capitalization…

# Stop words

- With a stop list, you exclude from the dictionary entirely the commonest words. Intuition:
  - They have little semantic content: *the, a, and, to, be*
  - There are a lot of them: ~30% of postings for top 30 words
- But the trend is away from doing this:
  - Good compression techniques means the space for including stop words in a system is very small
  - Good query optimization techniques mean you pay little at query time for including stop words.
  - You need them for:
    - Phrase queries: “King of Denmark”
    - Various song titles, etc.: “Let it be”, “To be or not to be”
    - “Relational” queries: “flights to London”

# Lemmatization

- Reduce inflectional/variant forms to base form
- Example: *am, are, is* $\rightarrow$ *be*
- Example: *car, cars, car's, cars'* $\rightarrow$ *car*
- Example: the boy's cars are different colors $\rightarrow$ the boy car be different color
- Lemmatization implies doing "proper" reduction to dictionary headword form (the lemma).

# Stemming

- Reduce terms to their “roots” before indexing
- “Stemming” suggests crude affix chopping
- language dependent
- Example: ***automate(s), automatic, automation*** all reduced to ***automat***.

# Porter Stemming Algorithm

- Most common algorithm for stemming English
- Results suggest that it is at least as good as other stemming options
- Contains 5 phases of reductions
- Phases are applied sequentially
- Each phase consists of a set of commands.
  - Sample command: Delete final ement if what remains is longer than 1 character
  - replacement → replac
  - cement → cement

# Porter stemmer: A few rules

| **Rule** | | | **Example** | | |
|---|---|---|---|---|---|
| SSES | $\rightarrow$ | SS | caresses | $\rightarrow$ | caress |
| IES | $\rightarrow$ | I | ponies | $\rightarrow$ | poni |
| SS | $\rightarrow$ | SS | caress | $\rightarrow$ | caress |
| S | $\rightarrow$ | | cats | $\rightarrow$ | cat |

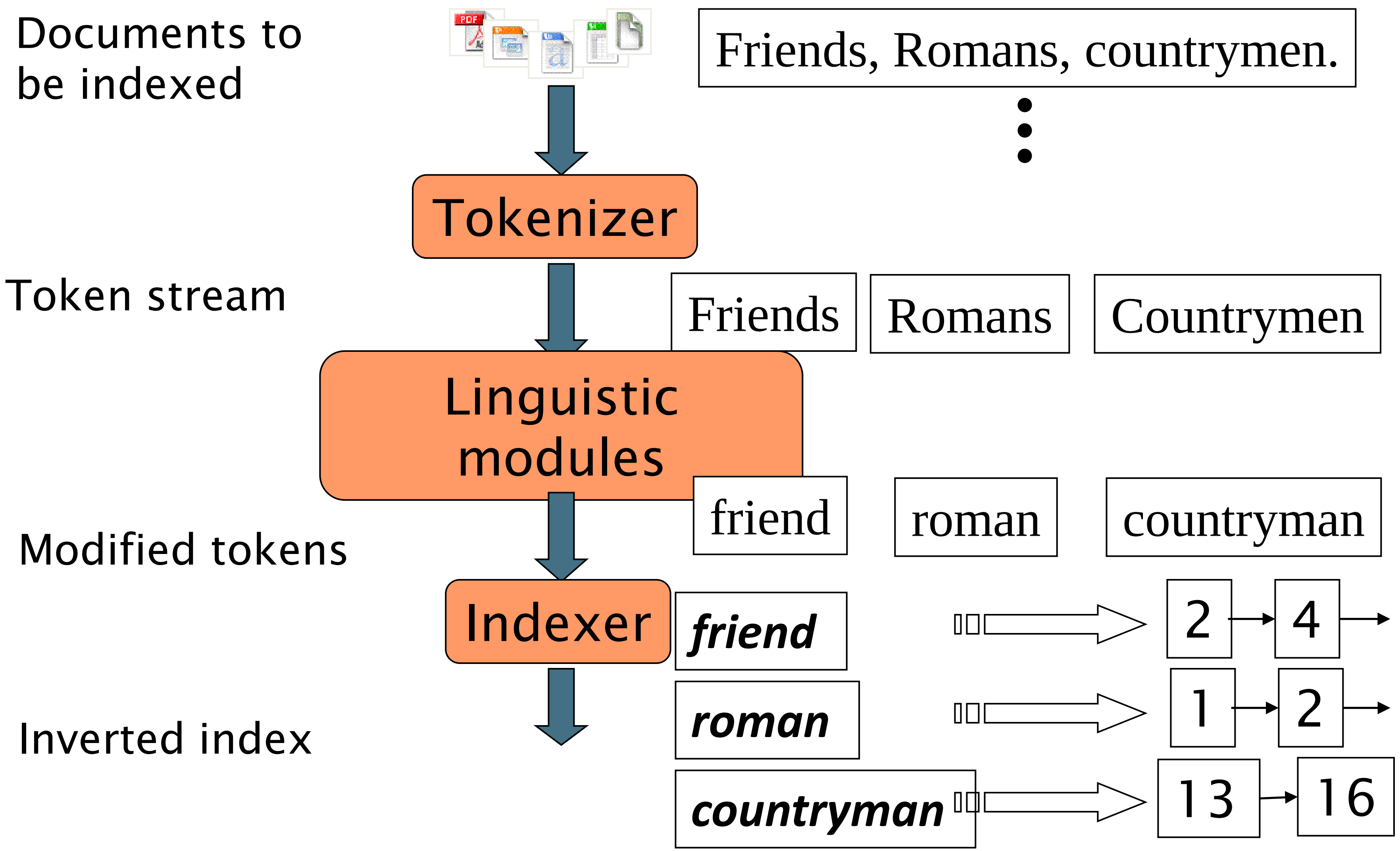
# Three stemmers: A comparison

- **Sample text**: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation
- **Porter stemmer**: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret
- **Lovins stemmer**: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres
- **Paice stemmer**: such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

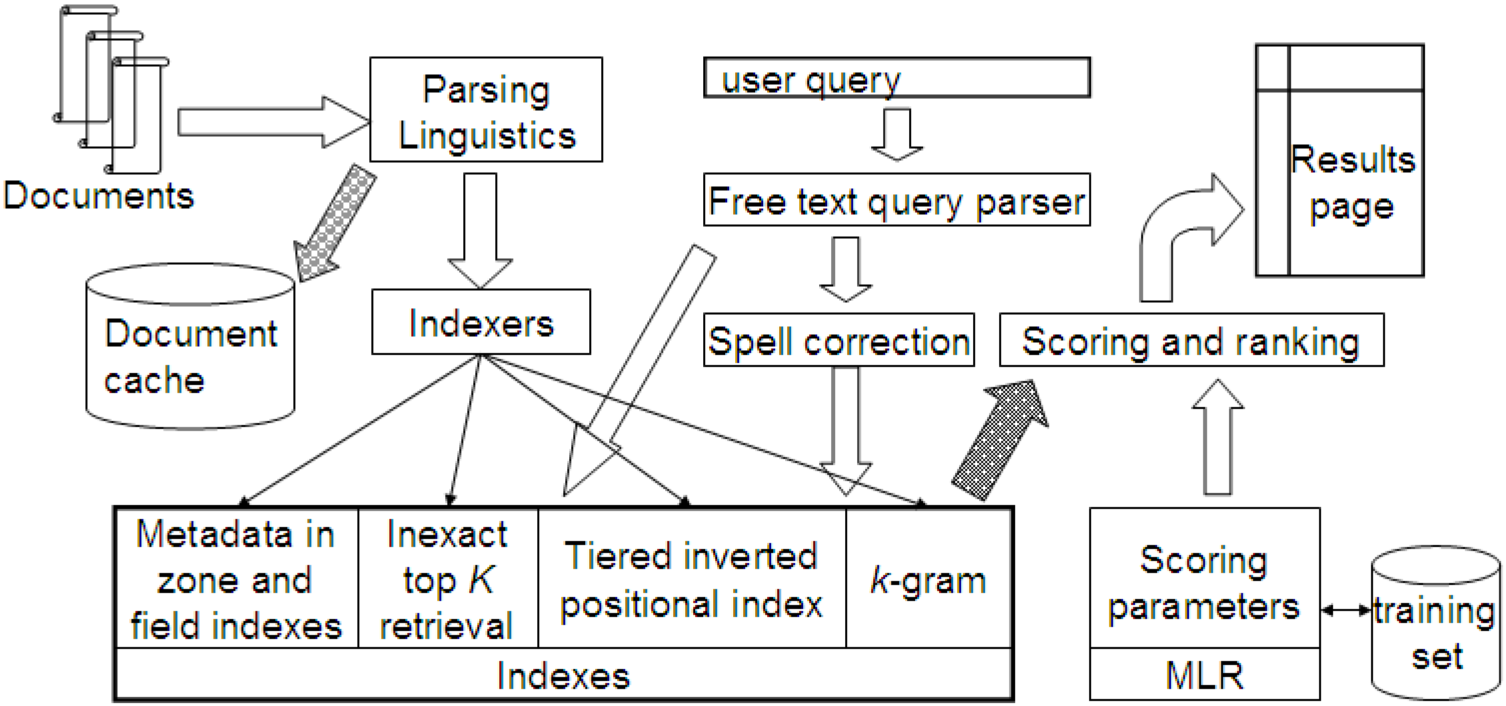
# Does stemming improve effectiveness?

- In general, stemming increases effectiveness for some queries, and decreases effectiveness for others.
- Porter Stemmer equivalence class **oper** contains all of **operate operating operates operation operative operatives operational.**
- Queries where stemming hurts: “operational AND research”, “operating AND system”, “operative AND dentistry”

# Recall the basic indexing pipeline



# Big Picture



Documents
Parsing
Linguistics
user query
Free text query parser
Results
page
Document
cache
Indexers
Spell correction
Scoring and ranking
Metadata in
zone and
field indexes
Inexact
top K
retrieval
Tiered inverted
positional index
k-gram
Indexes
Scoring
parameters
MLR
training
set

# HW1

- https://canvas.wpi.edu/courses/46542/assignments/283401?module_item_id=888447

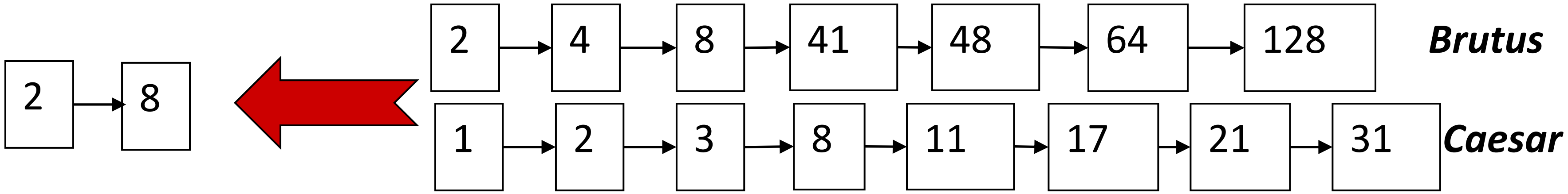
# Next…

- Need a better index than simple <term: docs>
- How can we improve on our basic index?
  - **Skip pointers**: faster postings merges
  - **Positional index**: Phrase queries and Proximity queries
  - **Permuterm index**: Wildcard queries

# Faster postings merges:
# Skip pointer

# Recall basic merge

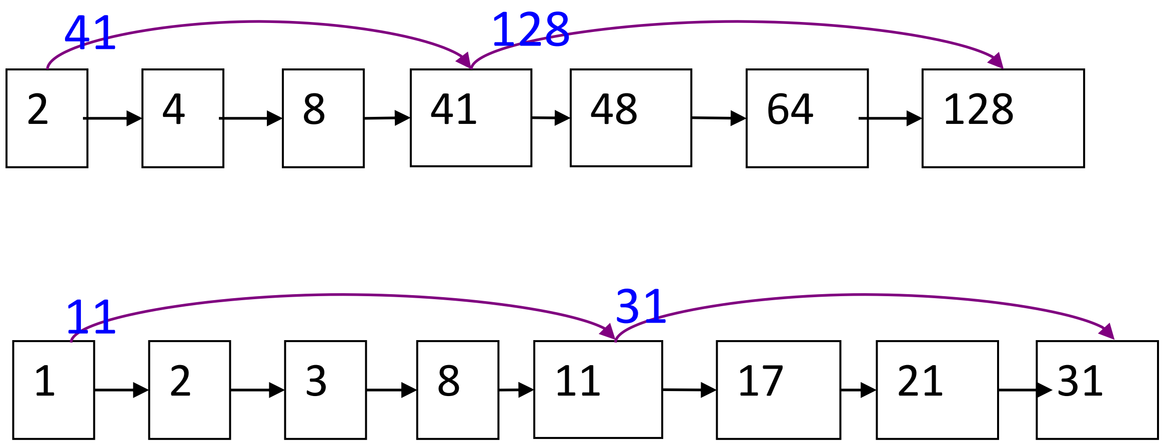
- Walk through the two postings simultaneously, in time linear in the total number of postings entries




If the list lengths are *m* and *n*, the merge takes O(*m*+*n*) operations.

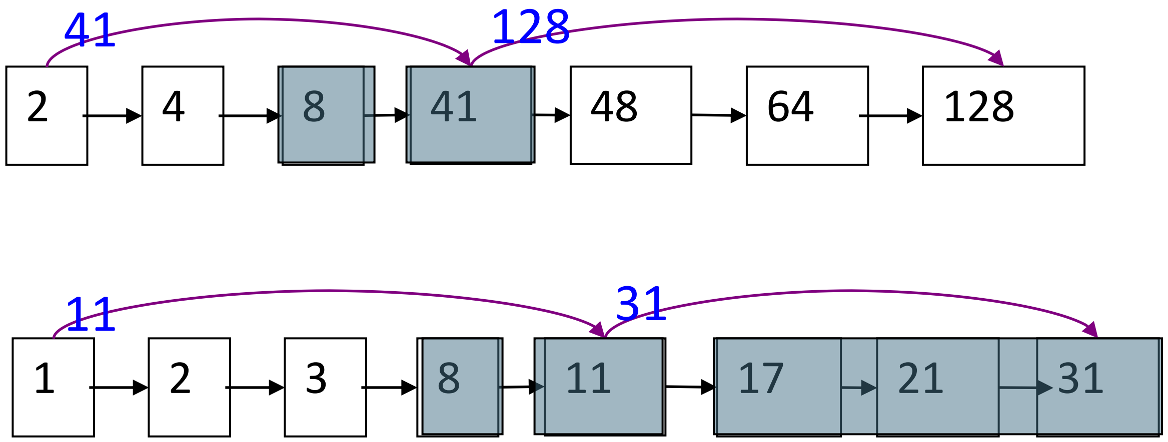
Can we do better?
Yes (if the index isn't changing too fast).

# Augment postings with skip pointers (at indexing time)




- Why?
- <u>To skip postings that will not figure in the search results.</u>
- How?
- Where do we place skip pointers?

# Query processing with skip pointers




Suppose we've stepped through the lists until we process **8** on each list. We match it and advance.

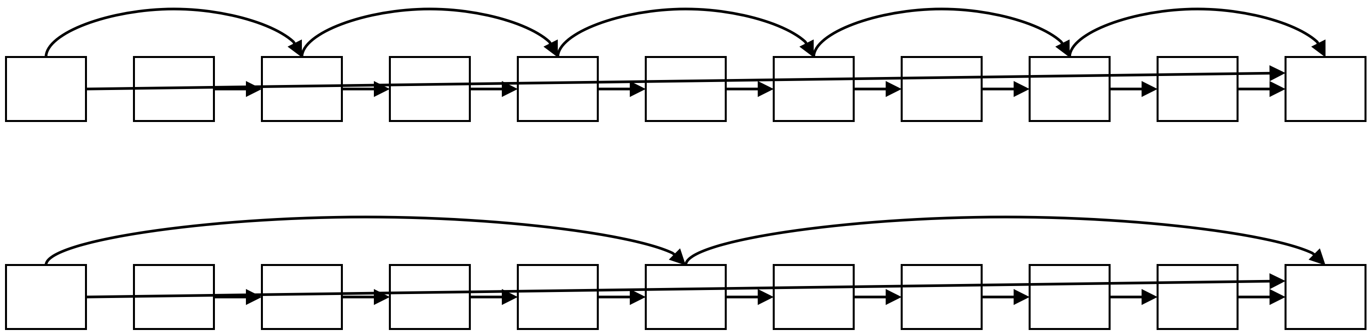
We then have **41** and **11** on the lower. **11** is smaller.

But the skip successor of **11** on the lower list is **31**, so we can skip ahead past the intervening postings.

```
INTERSECTWITHSKIPS(p1, p2)
answer ← ⟨ ⟩
while p1 ≠ NIL and p2 ≠ NIL
do if docID(p1) = docID(p2)
     then ADD(answer, docID(p1))
          p1 ← next(p1)
          p2 ← next(p2)
     else if docID(p1) < docID(p2)
            then if hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
                   then while hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
                          do p1 ← skip(p1)
                   else p1 ← next(p1)
            else if hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
                   then while hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
                          do p2 ← skip(p2)
                   else p2 ← next(p2)
return answer
```

# Where do we place skips?

- Tradeoff:
  - More skips $\rightarrow$ shorter skip spans $\Rightarrow$ more likely to skip. But lots of comparisons to skip pointers.
  - Fewer skips $\rightarrow$ few pointer comparison, but then long skip spans $\Rightarrow$ few successful skips.



# Placing skips

- So… More skips or fewer skips… Where to add skip pointers???
- Simple heuristic: for postings of length *L*, use $\sqrt{L}$ evenly-spaced skip pointers
- Easy if the index is relatively static; harder if *L* keeps changing because of updates.

# Positional Index

# Phrase queries

- Want to be able to answer queries such as ***"stanford university"*** – as a phrase
- Thus the sentence *"I went to university at Stanford"* is not a match.
  - The concept of phrase queries has proven easily understood by users; about 10% of web queries are phrase queries

- How??

# A first attempt: Biword indexes

- Index every consecutive pair of terms in the text as a phrase
- For example the text "Friends, Romans, Countrymen" would generate the biwords
  - ***friends romans***
  - ***romans countrymen***
- Each of these biwords is now a dictionary term
- Two-word phrase query-processing is now immediate.

# Longer phrase queries

- Longer phrases can be processed by breaking them down?
- ***stanford university palo alto*** can be broken into the Boolean query on biwords:

  ***stanford university*** *AND* ***university palo***

  *AND* ***palo alto***

<u>*Any problem?*</u>

Without the docs, we cannot verify that the docs matching the above Boolean query do contain the phrase.

Can have false positives!

# Solution 2: Positional indexes

- In the postings, store, for each ***term*** the position(s) in which tokens of it appear:

<***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***"?

- Can compress position values/offsets
- Nevertheless, this expands postings storage *substantially*

# 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

# Proximity queries

- Employment /3 place
  - Here, /*k* means "within *k* words of (on either side)".
- Clearly, positional indexes can be used for such queries; biword indexes cannot.

# Proximity Queries in Search Engines

- Google Search supports
  - keyword1 AROUND(n) keyword2
- Bing
  - keyword1 near:n keyword2 where n=the number of maximum separating words.
- Yahoo
  - keyword1 NEAR keyword2
- Exalead
  - keyword1 NEAR/n keyword2 where n is the number of words.

E.g., hotel around(5) terminal vs hotel around(3) terminal at Google

reference: https://en.wikipedia.org/wiki/Proximity_search_(text)

# Positional index size

- 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

- You can compress position values/offsets
- Nevertheless, a positional index expands postings storage substantially
- Nevertheless, it is now standardly used because of the power and usefulness of phrase and proximity queries … whether used explicitly or implicitly in a ranking retrieval system.

# Rules of thumb

- A positional index is 2–4 as large as a non-positional index
- Positional index size 35–50% of volume of original text

# Positional Indexes: Wrap-up

- With a positional index, we can answer
  - phrase queries
  - proximity queries

# Today…

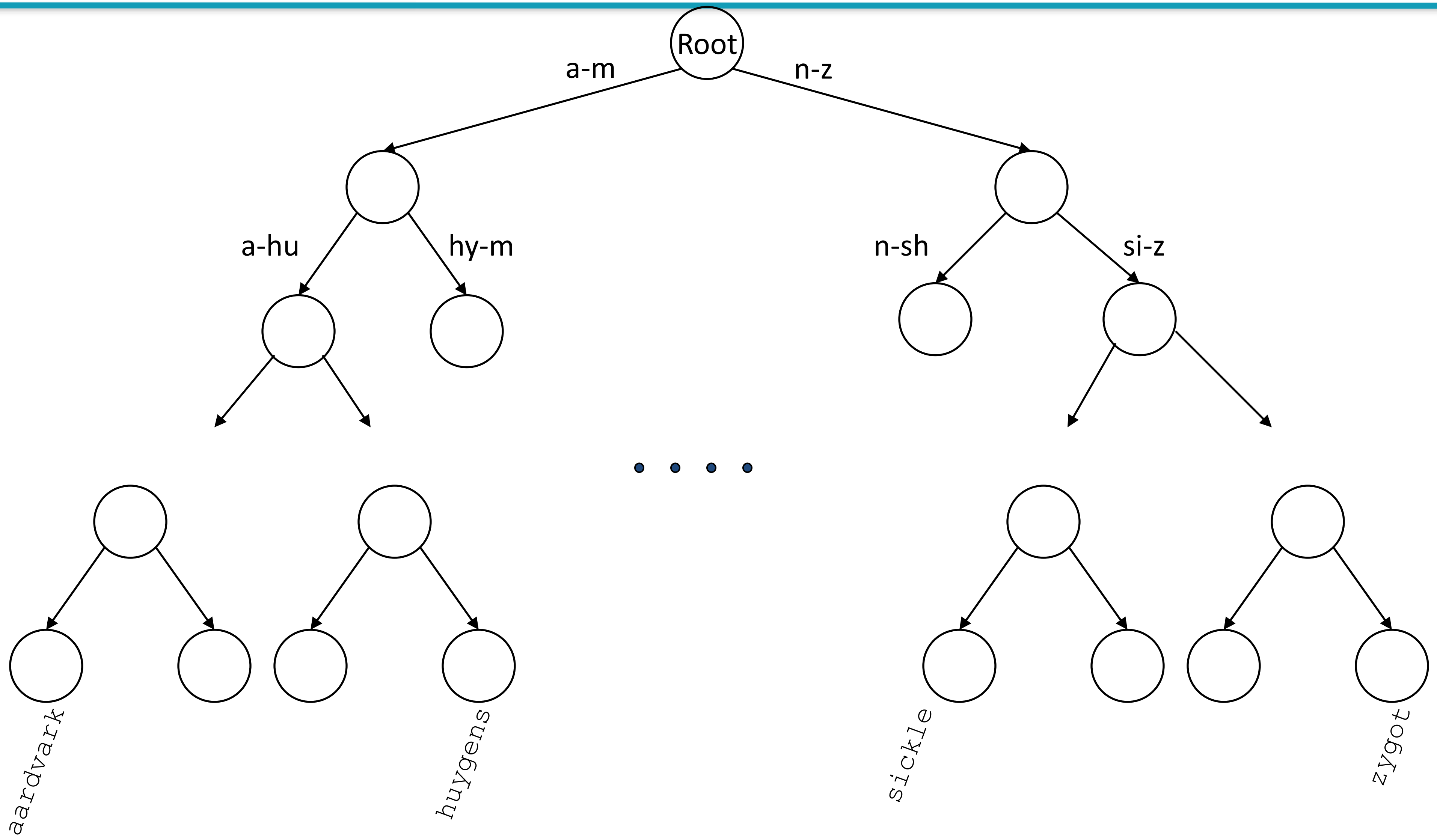
- Need a better index than simple <term: docs>
- How can we improve on our basic index?
  - **Skip pointers**: faster postings merges
  - **Positional index**: Phrase queries and Proximity queries
  - **Permuterm index**: Wildcard queries

# Wild-card queries

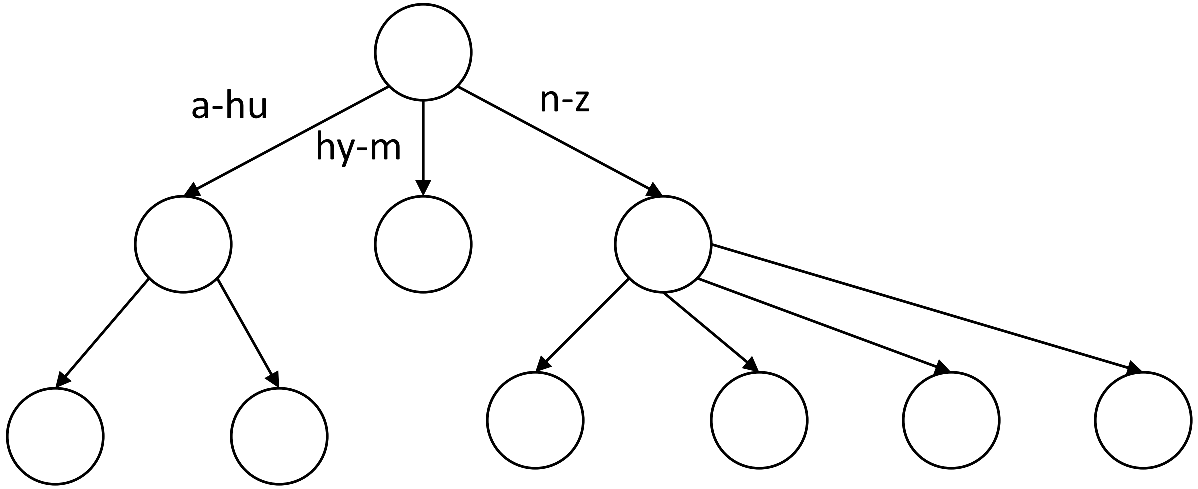
# Wild-card queries: *

- ***mon*:*** find all docs containing any word beginning with "mon".
- Easy with binary tree (or B-tree) lexicon: retrieve all words in range: ***mon ≤ w < moo***

# Tree: binary tree



# Tree: B-tree




- Definition: Every internal node has a number of children in the interval [*a*,*b*] where *a, b* are appropriate natural numbers, e.g., [2,4].

# Wild-card queries: *

- ***mon*:*** find all docs containing any word beginning with "mon".
- Easy with binary tree (or B-tree) lexicon: retrieve all words in range: ***mon ≤ w < moo***
- ****mon:*** find words ending in "mon": harder
  - Maintain an additional B-tree for terms *backwards.*

  Can retrieve all words in range: ***nom ≤ w < non****.*

Exercise: from this, how can we enumerate all terms meeting the wild-card query ***pro*cent*** ?

# Query processing

- At this point, we have an enumeration of all terms in the dictionary that match the wild-card query.
- We still have to look up the postings for each enumerated term.
- E.g., consider the query:

  ***se*ate*** *AND* ***fil*er***

  This may result in the execution of many Boolean *AND* queries.

# B-trees handle *'s at the end of a query term

- How can we handle *'s in the middle of query term?
  - ***co*tion***
- We could look up ***co**** AND ****tion*** in a B-tree and intersect the two term sets
  - Expensive
- The solution: transform wild-card queries so that the *'s occur at the end
- This gives rise to the **Permuterm** Index.

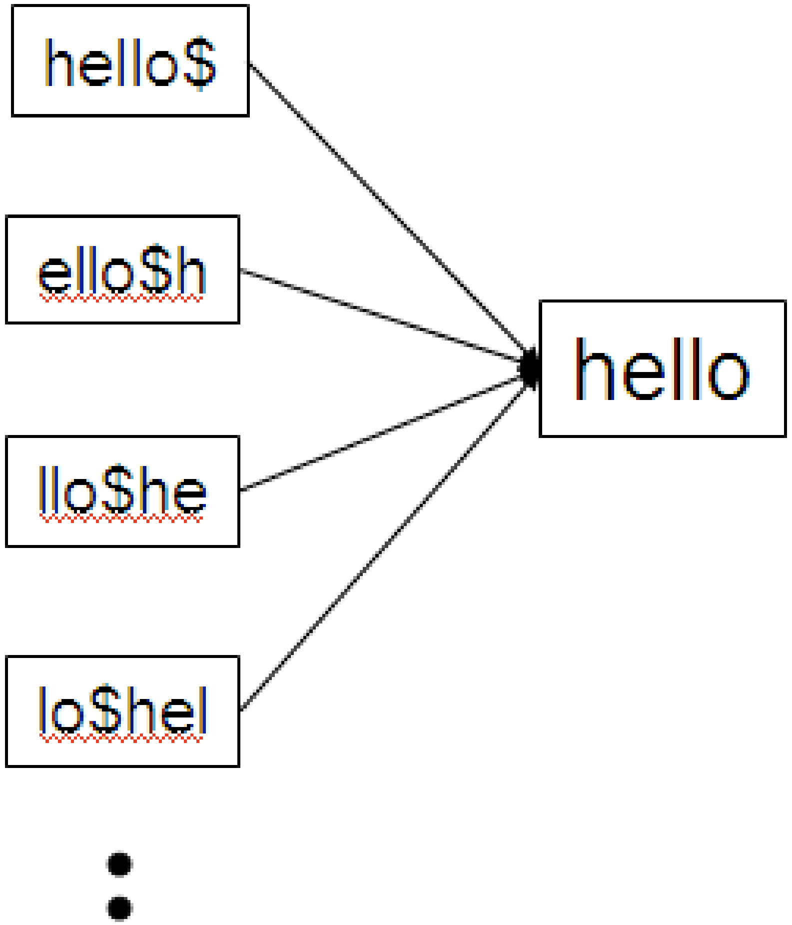
# Permuterm index

- For term ***hello***, index under:
  - ***hello$, ello$h, llo$he, lo$hel, o$hell, $hello***

  where $ is a special symbol.

Query = ***hel*o***
**X=*hel*, Y=*o***
Lookup ***o$hel****

- Queries:
  - **X** lookup on **X$**  **X*** lookup on $**X***
  - ***X** lookup on **X$***  ***X*** lookup on **X***
  - **X*Y** lookup on **Y$X***
  - **X*Y*Z** ??? Exercise!


hello$
ello$h
llo$he
lo$hel
hello

# Permuterm query processing

- Rotate query wild-card to the right
- Now use B-tree lookup as before.
- *Permuterm problem: ≈ quadruples lexicon size*

Empirical observation for English.

# Any Questions?