

# Information Retrieval

## Text Analysis and Processing

*Adapted from several IR courses:*

*C. Manning, P. Raghavan and H. Schütze (<http://nlp.stanford.edu/IR-book/>)  
W. Croft, D. Metzler, and T. Strohman (<http://www.search-engines-book.com/>)*

# Last lesson

- Boolean indexing and search
- Term-document matrix
- Inverted index
- Inverted index construction

# Plan for this lecture

- Preprocessing to form the term vocabulary
  - Documents
  - Tokenization
  - Stopping
  - Lemmatization
  - Stemming
- Handling phrases

# Recall the basic indexing pipeline

Documents to  
be indexed.



Friends, Romans, countrymen.

⋮

Tokenizer

Token stream.

Friends

Romans

Countrymen

Linguistic modules

Modified tokens.

friend

roman

countryman

Indexer

*friend*

→ 2 → 4 →

*roman*

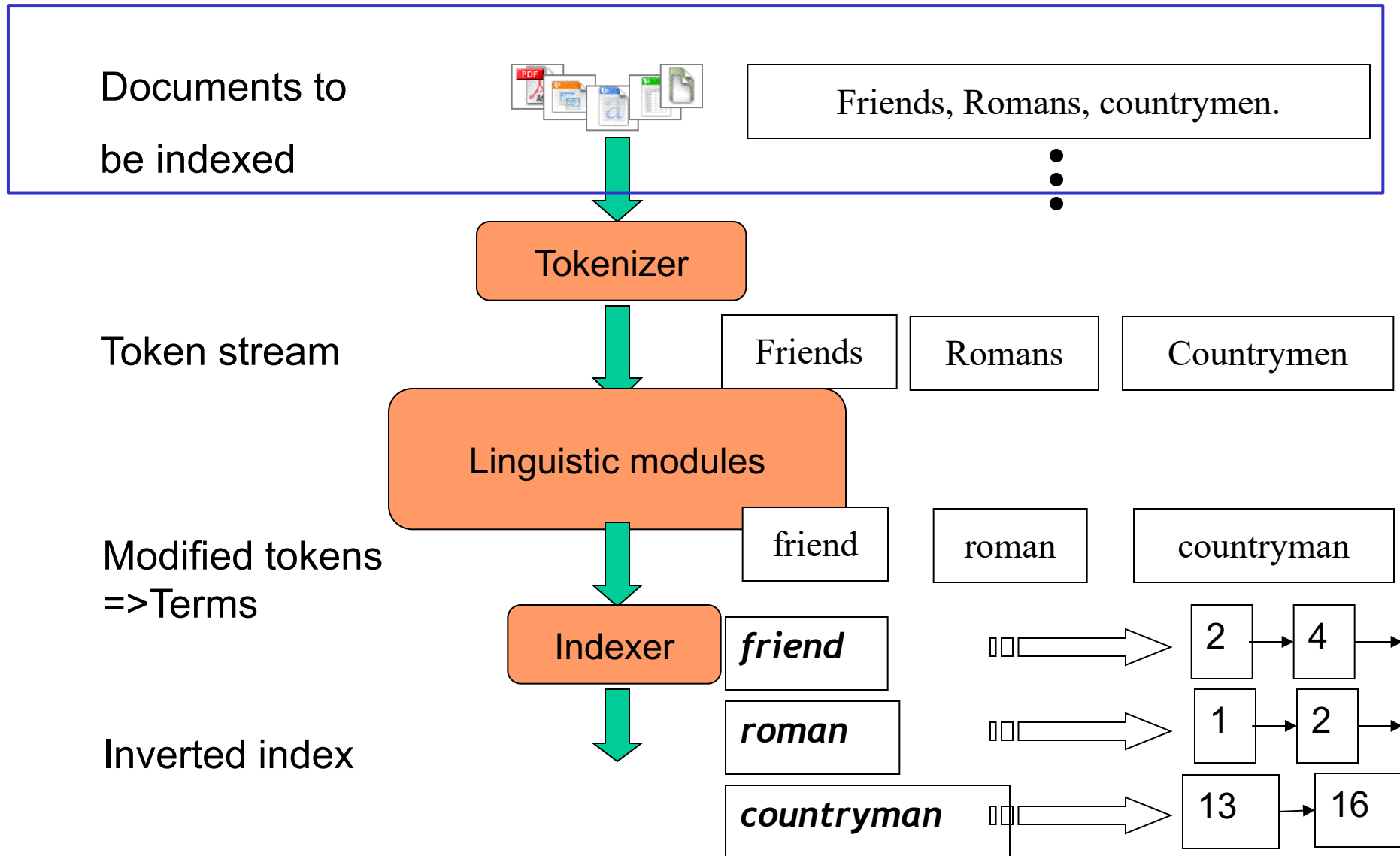
→ 1 → 2 →

*countryman*

→ 13 → 16 →

Inverted index.

# Inverted index construction



# Parsing a document

- What format is it in?
  - pdf/word/excel/html?
- What language is it in?
- What character set is in use?

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

But these tasks are often done heuristically ...

# Complications: Format/language

- Documents being indexed can include docs from many different languages
  - A single index may have to contain terms of several languages.
- Sometimes a document or its components can contain multiple languages/formats
  - French email with a German pdf attachment.
  - French email quoting clauses from contract in English
- There are commercial and open source libraries that can handle a lot of this stuff

# Complications: What is a document?

We return from our query “documents” but there are often interesting questions of grain size:

What is a unit document?

- A file?
- Documents within a file? (e.g. XML, CSV)
- An email? (Perhaps one of many in a single mbox file)
  - **What about an email with 5 attachments?**
- A group of files (e.g., PPT or LaTeX split over HTML pages)



# Inverted index construction

Page 12

Documents to  
be indexed



Friends, Romans, countrymen.

⋮

Token stream

Tokenizer

Friends

Romans

Countrymen

Linguistic modules

friend

roman

countryman

Modified tokens  
=>Terms

Indexer

*friend*

*roman*

*countryman*

2

4

1

2

13

16

Inverted index

# Tokens and Terms

# 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?

# Tokenization

- Alphanumeric sequences with 3 or more characters
- Example:
  - “Bigcorp's 2007 bi-annual report showed profits rose 10%.” becomes
  - “bigcorp 2007 annual report showed profits rose”
- Too simple for search applications or even large-scale experiments
- Why? Too much information lost
  - Small decisions in tokenizing can have major impact on effectiveness of some queries

# Tokenization

- Issues in tokenization:
  - *Finland's capital* →  
*Finland* AND *s*? *Finlands*? *Finland's*?
  - *Hewlett-Packard* → *Hewlett* and *Packard* as two tokens?
    - *state-of-the-art*: break up hyphenated sequence.
    - *co-education*
    - *lowercase, lower-case, lower case* ?
    - It can be effective to get the user to put in possible hyphens
  - *San Francisco*: one token or two?
    - How do you decide it is one token?

# Numbers

- *3/20/91*                      *Mar. 12, 1991*                      *20/3/91*
- *55 B.C.*
- *B-52*
- *My PGP key is 324a3df234cb23e*
- *(800) 234-2333*
  - Often have embedded spaces
  - Older IR systems may not index numbers
    - But often very useful: think about things like looking up error codes/stacktraces on the web
    - (One answer is using n-grams: IIR ch. 3)
  - Will often index “meta-data” separately
    - Creation date, format, etc.

# Tokenization: language issues

- French

- *L'ensemble* → one token or two?
  - *L ? L' ? Le ?*
  - Want *l'ensemble* to match with *un ensemble*  
Until at least 2003, it didn't on Google  
**Internationalization!**

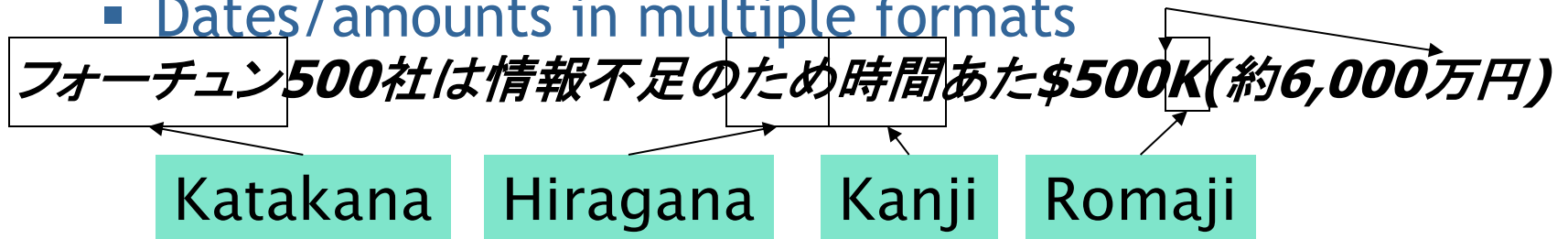
- German noun compounds are not segmented

- *Lebensversicherungsgesellschaftsangestellter*
- 'life insurance company employee'
- German retrieval systems benefit greatly from a **compound splitter** module  
Can give a 15% performance boost for German

# Tokenization: language issues

- Chinese and Japanese have no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - Not always guaranteed a unique tokenization
- Further complicated in Japanese, with multiple alphabets intermingled

- Dates/amounts in multiple formats



End-user can express query entirely in hiragana!



# Tokenization: language issues

- 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 عام من الاحتلال الفرنسي.

← → ← →

← start

- ‘Algeria achieved its independence in 1962 after 132 years of French occupation.’
- With Unicode, the surface presentation is complex, but the stored form is straightforward

# 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 (IIR 5) means the space for including stop words in a system is very small
  - Good query optimization techniques (IIR 7) 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”

# Normalization to terms

- We may 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*
- Result is terms: a **term** is a (normalized) word type, which is an entry in our IR system dictionary
- We most commonly implicitly define equivalence classes of terms by, e.g.,
  - deleting periods to form a term
    - *U.S.A., USA* → *USA*
  - deleting hyphens to form a term
    - *anti-discriminatory, antidiscriminatory* → *antidiscriminatory*

# Normalization: other languages

- Accents: e.g., French *résumé* vs. *resume*.
- Umlauts: e.g., German: *Tuebingen* vs. *Tübingen*
  - Should be equivalent
- Most important criterion:
  - How are your users like to write their queries for these words?
- Even in languages that standardly have accents, users often may not type them
  - Often best to normalize to a de-accented term
    - *Tuebingen, Tübingen, Tubingen* → *Tubingen*

# Normalization: other languages

- Normalization of things like date forms
  - *7月30日 vs. 7/30*
  - *Japanese use of kana vs. Chinese characters*
- Tokenization and normalization may depend on the language and so is intertwined with language detection

*Morgen will ich in MIT ...*

Is this  
German “mit”?
- Crucial: Need to “normalize” indexed text as well as query terms

# 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...

- Google example:
  - Query: C.A.T.

The screenshot shows a Google search for the query "C.A.T.". The search bar at the top displays the Google logo and the text "C.A.T.". Below the search bar, there are tabs for "All", "Images", "Videos", "News", "Maps", and "More". The search results are displayed below the tabs, showing approximately 1,860,000 results in 0.70 seconds.

The first result is "Cat - Wikipedia, the free encyclopedia" with the URL <https://en.wikipedia.org/wiki/Cat>. The snippet describes the domestic cat (Latin: *Felis catus*) or the feral cat (Latin: *Felis silvestris catus*) as a small, typically furry, carnivorous mammal. It also mentions African wildcat, Creme Puff, and a list of cat breeds.

The second result is "Caterpillar: Cat | global-selector" with the URL [www.cat.com/](http://www.cat.com/). The snippet describes the company as a manufacturer of construction and mining equipment, diesel and natural gas engines, industrial gas turbines, and a wide offering of related services.

The third result is "In the news" with a sub-header "Taxidermy cat handbag sells at auction in New Zealand". The snippet mentions a handbag made from a taxidermied roadkill ginger cat selling at auction in New Zealand. Below this, there are two news snippets: "Noel Edmonds phoned my cat | Peter Ormerod | Opinion" from The Guardian and "Taxidermy cat bag sold for \$545" from Stuff.co.nz.

The fourth result is "Cats the Musical - Official Website & Tickets" with the URL [www.catsmusical.com/](http://www.catsmusical.com/). The snippet describes it as the official home of Andrew Lloyd Webber's world-famous, family-favourite musical CATS.

The fifth result is "Cat phones: Rugged Phones & Accessories" with the URL [www.catphones.com/](http://www.catphones.com/). The snippet describes the company as a manufacturer of rugged phones and accessories designed to be durable and tough enough to handle any day.

On the right side of the search results, there is a "Cat" section with a grid of cat images. Below the images, there is a "Cat" section with a description of the domestic cat or feral cat as a small, typically furry, carnivorous mammal. It also includes scientific information: Scientific name: *Felis catus*, Lifespan: 15 years (Domesticated), Gestation period: 64 - 67 days, Higher classification: Felis, Length: 46 cm (Without Tail), and Mass: 3.6 - 4.5 kg (Adult). Below this, there is a "Breeds" section with a grid of cat breeds: British Shorthair, Siamese cat, Persian cat, Ragdoll, and Maine Coon.

# Normalization to terms

- An alternative to equivalence classing is to do asymmetric expansion
- An example of where this may be useful
  - Enter: *window*                      Search: *window, windows*
  - Enter: *windows*   Search: *Windows, windows, window*
  - Enter: *Windows*   Search: *Windows*
- Potentially more powerful, but less efficient

# Thesauri and soundex

- Do we handle synonyms and homonyms?
  - E.g., by hand-constructed equivalence classes
    - *car = automobile      color = colour*
  - We can rewrite to form equivalence-class terms
    - When the document contains *automobile*, index it under *car-automobile* (and vice-versa)
  - Or we can expand a query
    - When the query contains *automobile*, look under *car* as well
- What about spelling mistakes?
  - One approach is Soundex, which forms equivalence classes of words based on phonetic heuristics
- More in IIR 3 and IIR 9



# Lemmatization

- Reduce inflectional/variant forms to base form
- E.g.,
  - *am, are, is* → *be*
  - *car, cars, car's, cars'* → *car*
- *the boy's cars are different colors* → *the boy car be different color*
- Lemmatization implies doing “proper” reduction to dictionary headword form

# Stemming

- Many morphological variations of words
  - inflectional (plurals, tenses)
  - derivational (making verbs nouns etc.)
- In most cases, these have the same or very similar meanings
- Stemmers attempt to reduce morphological variations of words to a common stem
  - usually involves removing suffixes
- Can be done at indexing time or as part of query processing (like stopwords)

# Stemming

- Two basic types
  - Dictionary-based: uses lists of related words
  - Algorithmic: uses program to determine related words
- Algorithmic stemmers
  - suffix-s: remove 's' endings assuming plural
    - e.g., cats → cat, lakes → lake, wiis → wii
  - Some problems:
    - supplies → supplie (*should be supply*)
    - ups → up (*different concept*)

# Porter's algorithm

- Commonest algorithm for stemming English
  - Results suggest it's at least as good as other stemming options
- Conventions + 5 phases of reductions
  - phases applied sequentially
  - each phase consists of a set of commands
  - sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*

# Typical rules in Porter

- *ational* → *ate* (e.g., rational → rate)
- *tional* → *tion* (e.g., conventional → convention)
- *sses* → *ss* (e.g., guesses → guess)
- *ies* → *i* (e.g., dictionaries → dictionary)
- Weight of word sensitive rules
  - ( $m > 1$ ) *EMENT* →
    - *replacement* → *replac*
    - *cement* → *cement*

# Other stemmers

- Other stemmers exist:
  - Lovins stemmer
    - <http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovins.htm>
    - Single-pass, longest suffix removal (about 250 rules)
  - Paice/Husk stemmer
  - Snowball
    - <http://snowball.tartarus.org>
- Full morphological analysis (lemmatization)
  - At most modest benefits for retrieval

# Language-specificity

- The above methods embody transformations that are
  - Language-specific, and often
  - Application-specific
- These are “plug-ins” to be added to the indexing process
- Both open source and commercial plug-ins are available for handling these

## Does stemming help?

- English: very mixed results. Helps recall for some queries but harms precision on others
  - E.g., operative (dentistry)  $\Rightarrow$  oper
- Definitely useful for Spanish, German, Finnish, ...
  - 30% performance gains for Finnish!



# Phrase queries and positional indexes

# 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; one of the few “advanced search” ideas that works
  - Many more queries are *implicit phrase queries*
- For this, it no longer suffices to store only *<term : docs>* entries
  - More vocabulary’s entries, OR
  - The posting list structure must be expanded.

# Phrases

- Text processing issue - how are phrases recognized?
- Two possible approaches:
  - Use word n-grams (n-words)
  - Store word positions in indexes and use proximity operators in queries

# 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 are processed by combining shorter phrases:
- *stanford university palo alto* can be broken into the Boolean query on biwords:

*stanford university AND university palo AND palo alto*

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



Can have false positives!

# Issues for biword indexes

- False positives, as noted before
- Index blowup due to bigger dictionary
  - Infeasible for more than biwords, big even for them
- Biword indexes are not the standard solution (for all biwords) but can be part of a compound strategy

# Google N-Grams

- Web search engines index n-grams
- Google sample [2006]:

Number of tokens:	1,024,908,267,229
Number of sentences:	95,119,665,584
Number of unigrams:	13,588,391
Number of bigrams:	314,843,401
Number of trigrams:	977,069,902
Number of fourgrams:	1,313,818,354
Number of fivegrams:	1,176,470,663

- Most frequent trigram in English is “all rights reserved”
  - In Chinese, “limited liability corporation”

## Solution 2: Positional indexes

- In the postings, store, for each *term* the position(s) in it appears:

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

*doc1*: position1, position2 ... ;

*doc2*: position1, position2 ... ;

etc.>



## Solution 2: Positional indexes

- In the postings, store, for each *term* the position(s) in it appears:

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

*doc1*: position1, position2 ... ;

*doc2*: position1, position2 ... ;

etc.>

- Note: to use the correct terminology, *tokens* appear in documents; the indexed form of a token is a *term*

Ex: “**Operate** an **operating** system...”

<**oper**; doc1: 0, 2>

# Positional index example

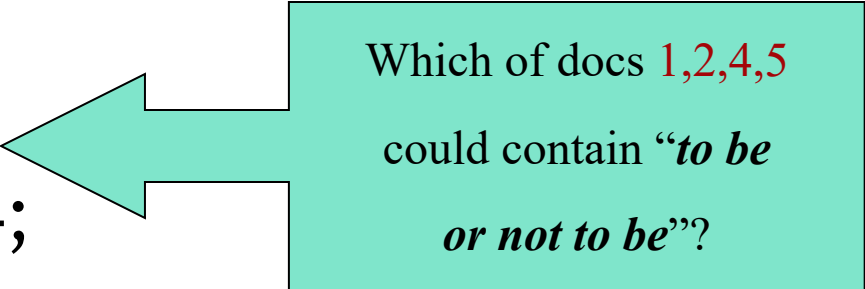
<*be*: 993427;

*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*"?

- For phrase queries, we use a merge algorithm recursively at the document level
- But we now need to deal with more than just equality

# Positional index size

- You can compress position values/offsets (more on this later; see IIR book)
- Nevertheless, a positional index expands postings storage *substantially*
- Nevertheless, a positional index 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.

# 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

- Need an entry for each occurrence, not just once per document
- Index size depends on average document size
  - Average web page has <1000 terms
  - Books, ... easily 100,000 terms
- Consider a term with frequency 0.1%

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

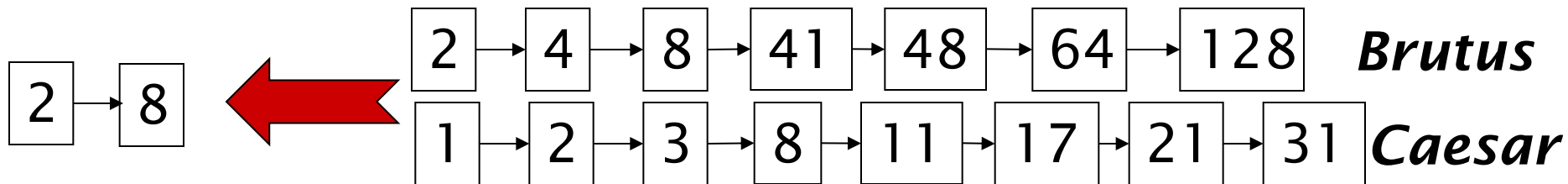
## Rules of thumb

- A positional index is 2-4 as large as a non-positional index
- Positional index size 35-50% of volume of original text
- Caveat: all of this holds for “English-like” languages

**Faster postings merges:  
Skip pointers/Skip lists**

## Recall basic merge

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



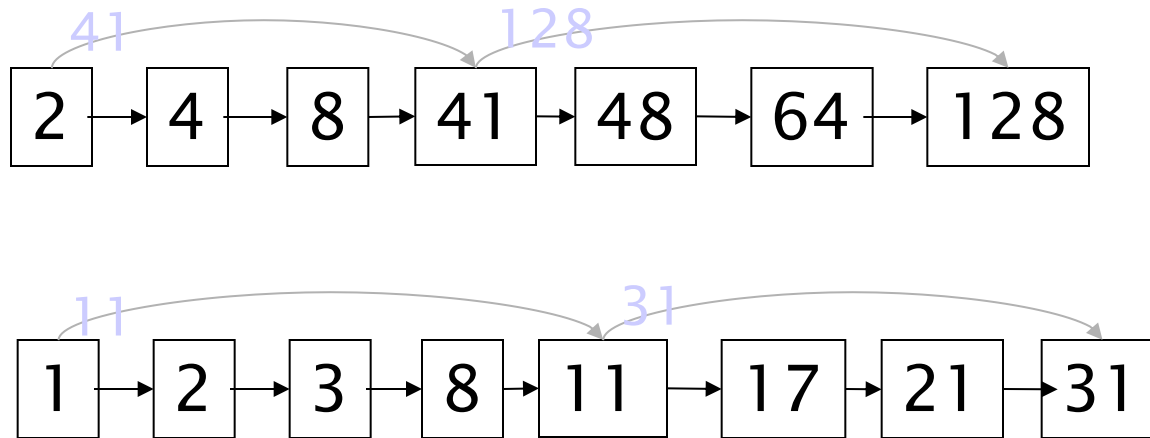
If the list lengths are  $m$  and  $n$ , the merge takes  $O(m+n)$  operations.

Can we do better?

Yes (if the index isn't changing too fast).

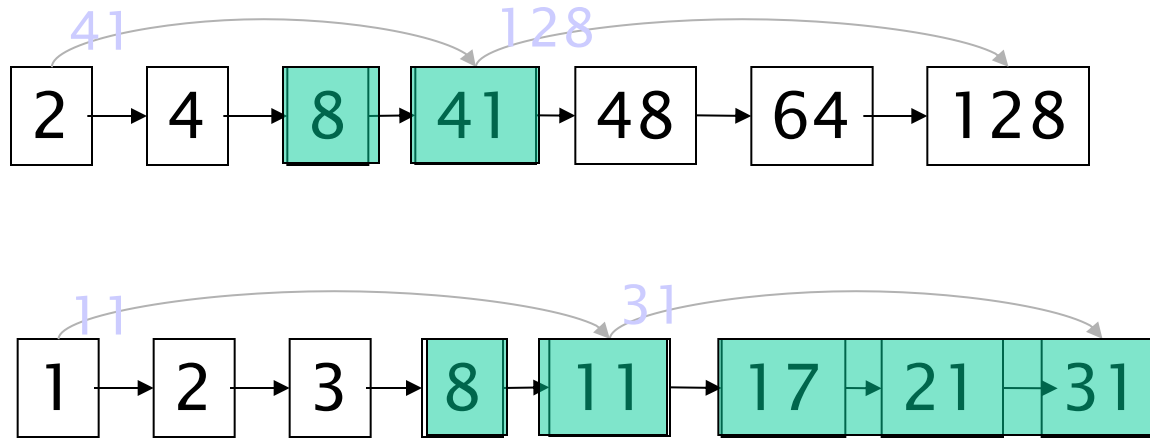


# Augment postings with skip pointers (at indexing time)



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

# Query processing with skip pointers



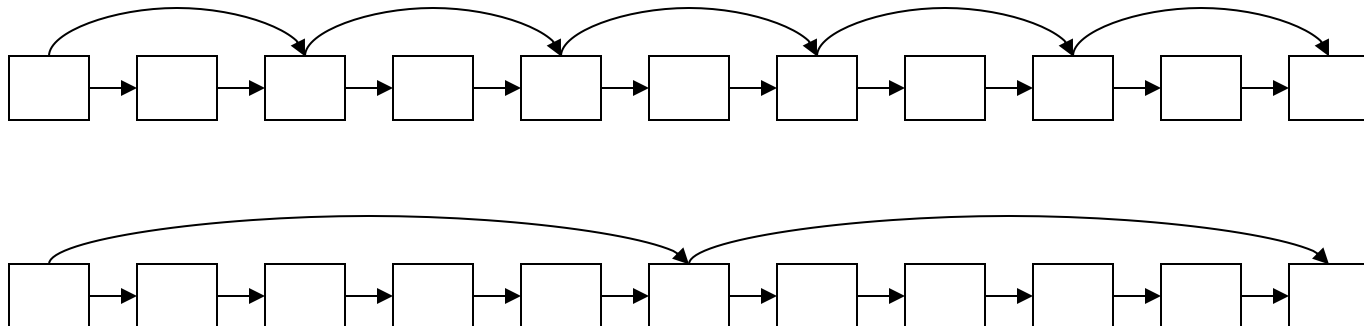
Suppose we've stepped through the lists until we process 8 on each list. We match it and advance.

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.

# 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

- Simple heuristic: for postings of length  $L$ , use  $\sqrt{L}$  evenly-spaced skip pointers [Moffat and Zobel 1996]
- This ignores the distribution of query terms.
- Easy if the index is relatively static; harder if  $L$  keeps changing because of updates.
- This definitely used to help; with modern hardware it may not unless you're memory-based [Bahle et al. 2002]
  - The I/O cost of loading a bigger postings list can outweigh the gains from quicker in memory merging!

# This lesson

- Documents
- Tokenization
- Stopping
- Lemmatization
- Stemming
- Handling phrases: bigram index, positional index
- Skip pointers

# Next lessons

- Index construction
  - Data structures
  - Indexing strategies
  - Distributed indexing
- Ranking
  - Weighting schemes
  - Vector space scoring