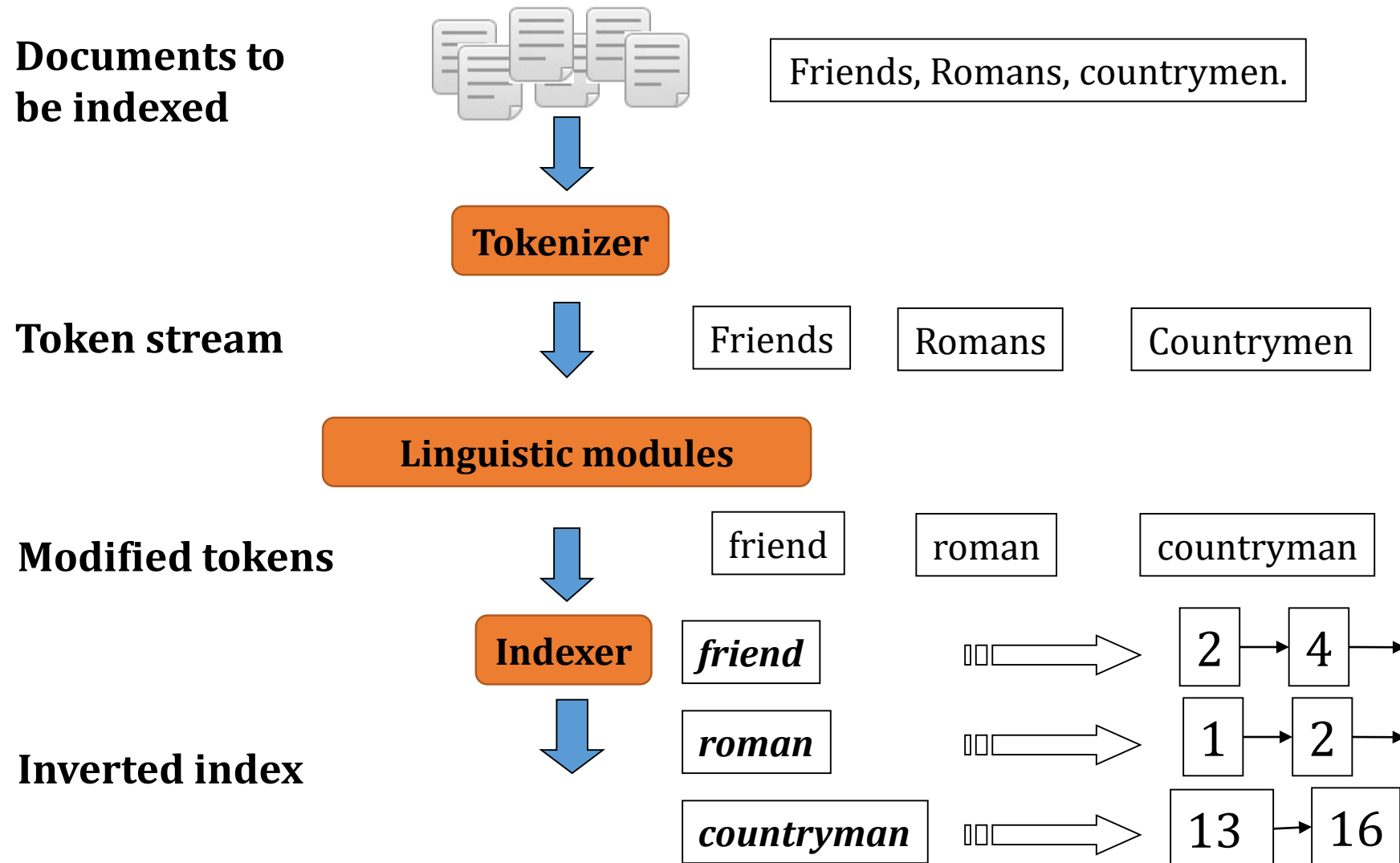


02

The Term Vocabulary & Postings Lists

Document Ingestion

Recall the basic indexing pipeline



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

Complications: Format/language

- Documents being indexed can include docs from many different languages
 - A single index may contain terms from many languages.
- Sometimes a document or its components can contain multiple languages/formats
 - French email with a German pdf attachment.
 - French email quote clauses from an English-language contract
- 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?
 - 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)

Tokens

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

- 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

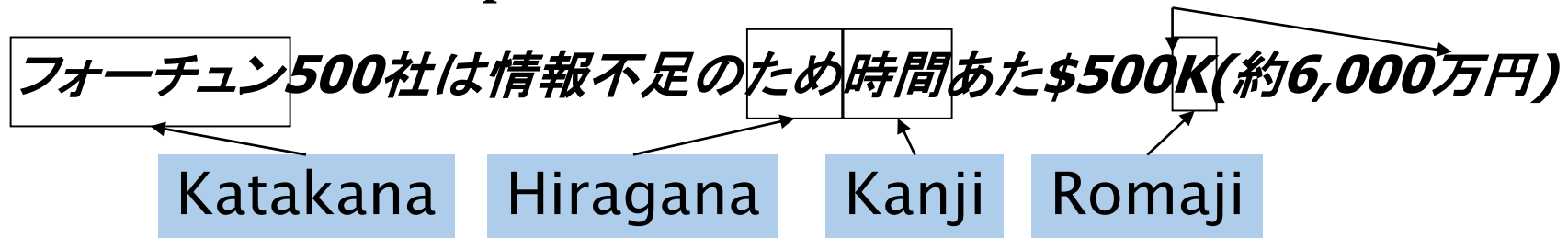
- **3/20/91**
- **55 B.C.**
- **B-52**
- **My PGP key is 324a3df234cb23e**
- **(800) 234-2333**
 - Often have embedded spaces
 - Older IR systems may not index numbers
 - But often very useful: think about things like looking up error codes/stacktraces on the web
 - (One answer is using n-grams)
 - Will 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

Terms

The things indexed in an IR system

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”

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 *identically*

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...

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

Stemming and Lemmatization

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

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

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

- 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

- *sses* → *ss*
- *ies* → *i*
- *ational* → *ate*
- *tional* → *tion*

- Weight of word sensitive rules
- $(m > 1)$ *EMENT* →
 - *replacement* → *replac*
 - *cement* → *cement*

Other stemmers

- Other stemmers exist:
 - Lovins stemmer
 - <http://snowballstem.org/algorithms/lovins/stemmer.html>
 - Single-pass, longest suffix removal (about 250 rules)
 - Paice/Husk stemmer
 - <http://paicehusk.appspot.com>
 - Snowball
 - <http://snowballstem.org/demo.html>
- 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-in” addenda to the indexing process
- Both open source and commercial plug-ins are available for handling these

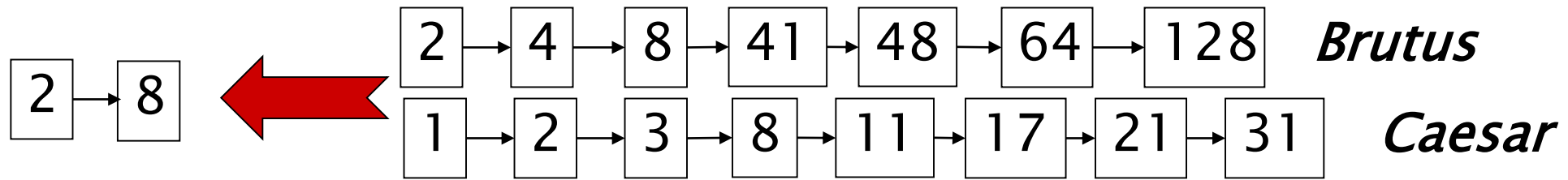
Does stemming help?

- 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!

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

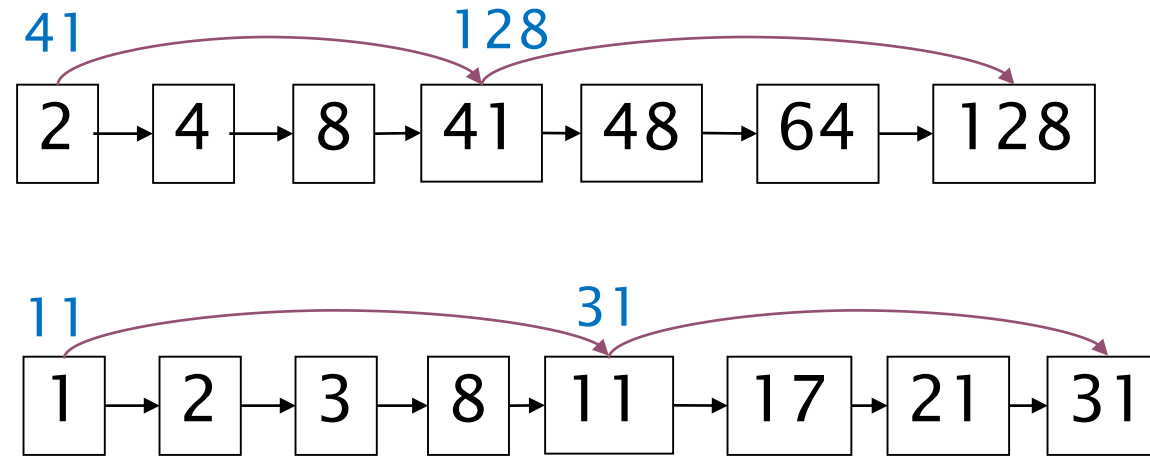


If the list lengths are m and n , the merge takes $O(m+n)$ operations.

Can we do better?

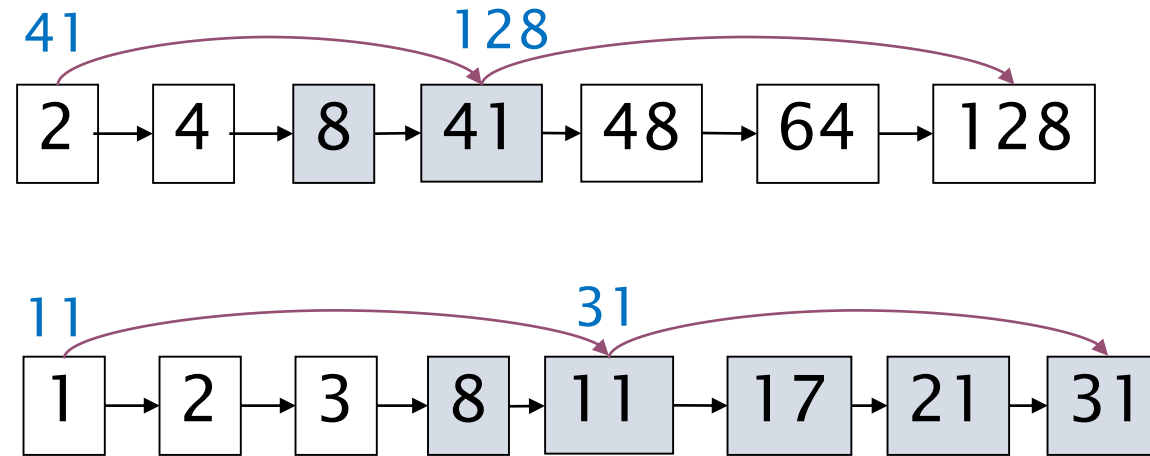
Yes (if the index isn't changing too fast).

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.

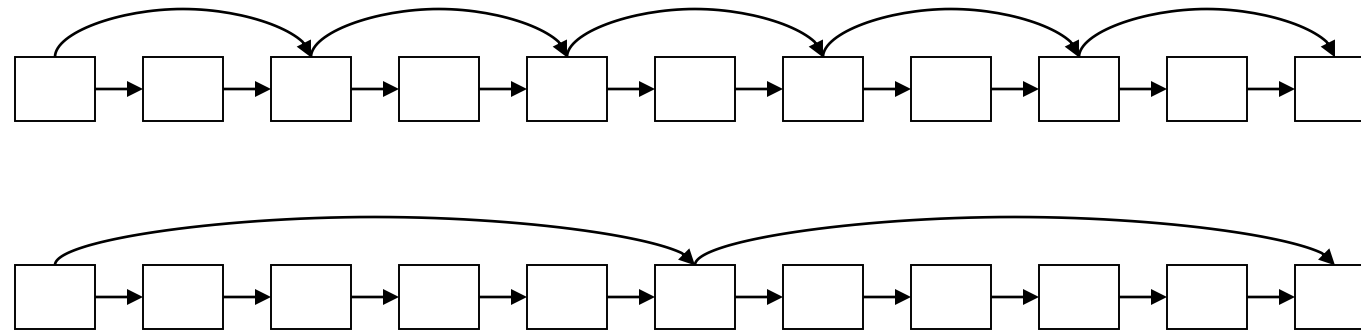
We then have 41 and 11 on the lower. 11 is smaller.

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!