Information Storage and Retrieval

CSCE 670
Texas A&M University
Department of Computer Science & Engineering
Instructor: Prof. James Caverlee

Text Retrieval Basics
19 January 2017

Today: Foundations

The Central Problem in Search

Searcher

Author

Concepts

Concepts

Query Terms

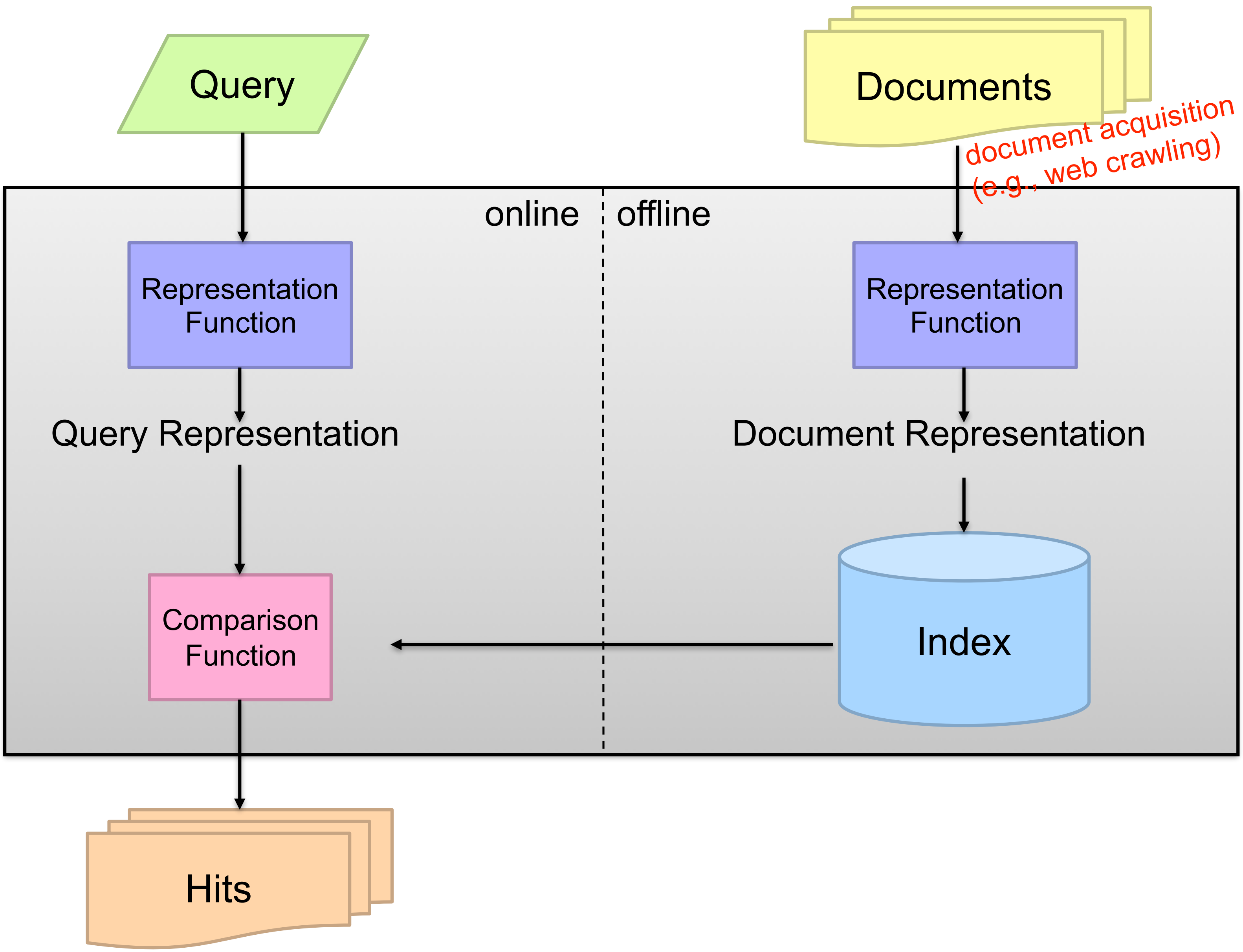
"tragic love story"

Document Terms

"fateful star-crossed romance"

Do these represent the same concepts?

Abstract IR Architecture



Simplest model: Boolean 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.

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 corpora

- Consider N = 1B documents, each with about 1K terms.
- Average 6 bytes/term including spaces/punctuation
  - 6TB of data in the documents.
- Say there are m = 50M distinct terms among these.

Can't build the matrix

- 50M x 1B matrix has 50 quadrillion 0's and 1's.
  - 50,000,000,000,000,000
- But it has no more than one trillion 1's. Why?
  - Matrix is extremely sparse.
- What's a better representation?
  - We only record the 1 positions.

Inverted index

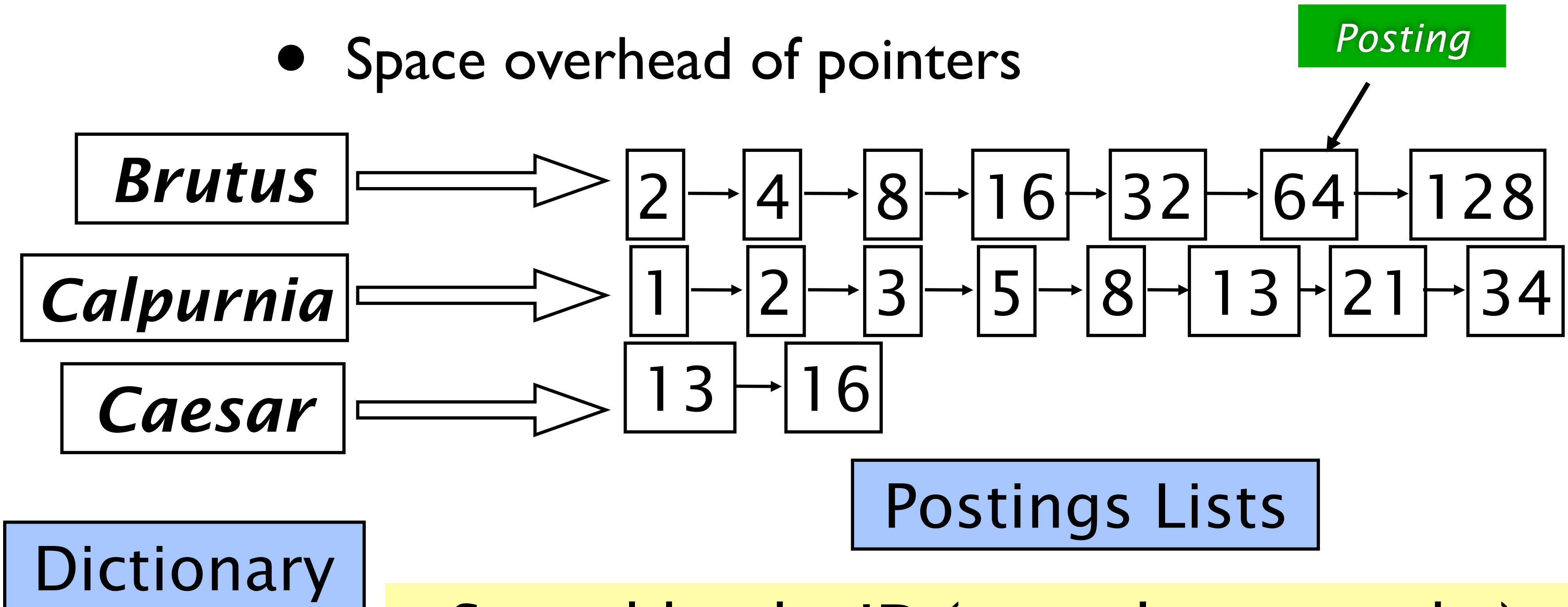
- For each term T, we must store a list of all documents that contain T.
- Do we use an array or a list for this?

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

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

Inverted index

- Linked lists generally preferred to arrays
  - Dynamic space allocation
  - Insertion of terms into documents easy
  - Space overhead of pointers



Sorted by docID (more later on why).

Query processing: AND

- Consider processing the query:

  Brutus AND Caesar

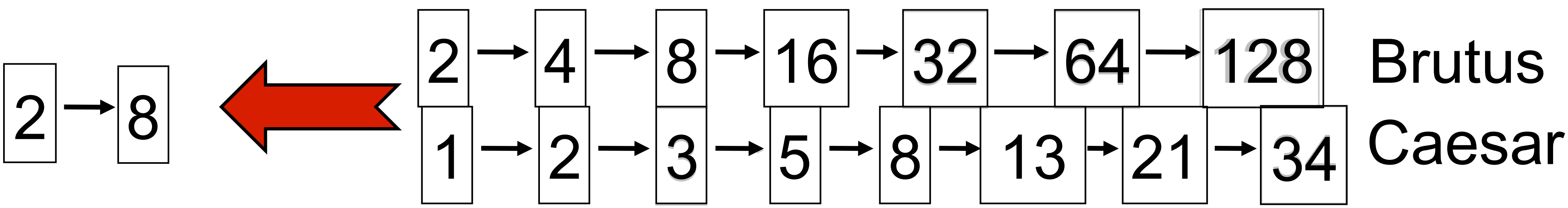
  - Locate Brutus in the Dictionary;
    - Retrieve its postings.
  - Locate Caesar in the Dictionary;
    - Retrieve its postings.
  - "Merge" the two postings:

$2 \rightarrow 4 \rightarrow 8 \rightarrow 16 \rightarrow 32 \rightarrow 64 \rightarrow 128$ Brutus

$1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 8 \rightarrow 13 \rightarrow 21 \rightarrow 34$ Caesar

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.

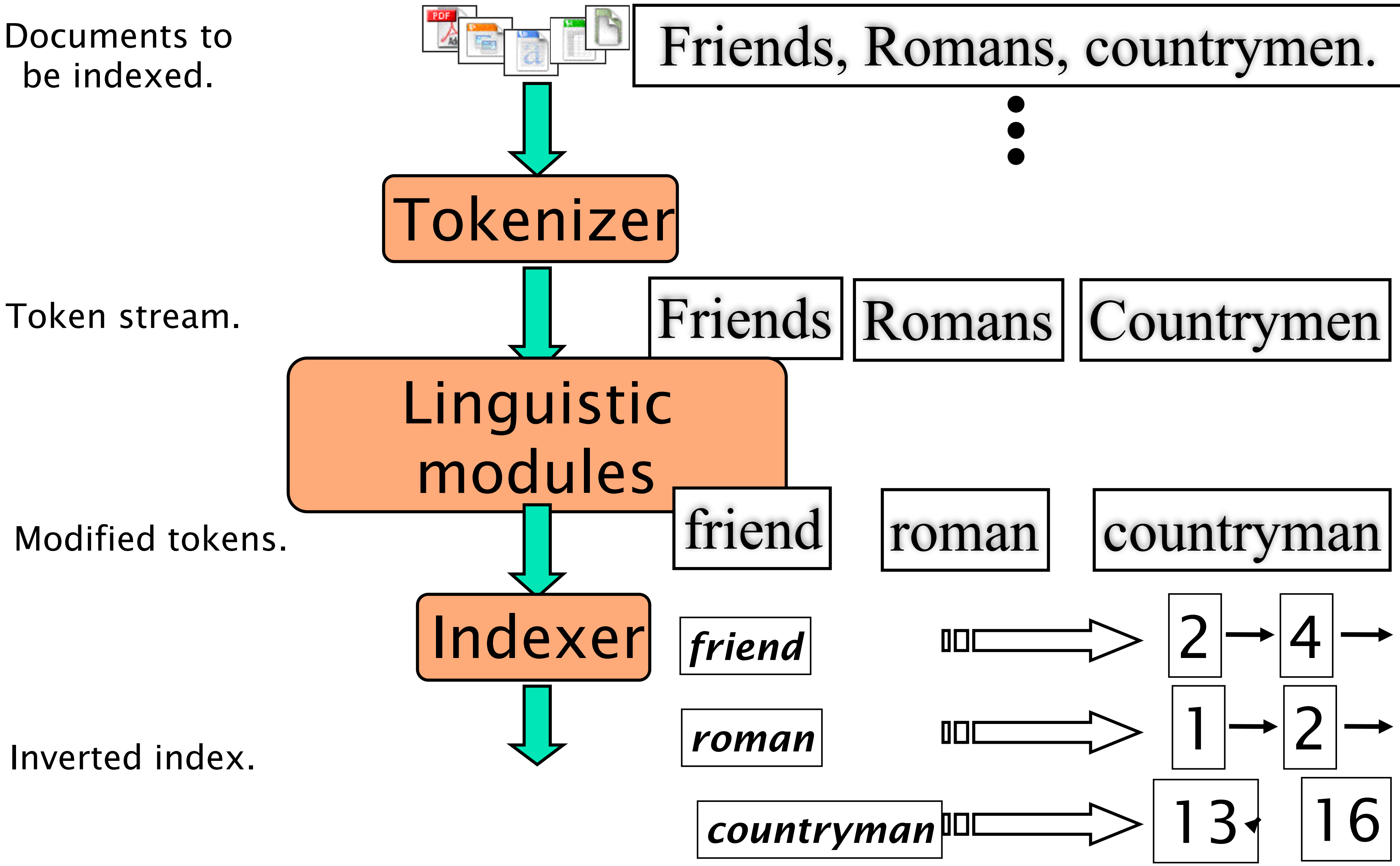
Crucial: postings sorted by docID.

Boolean Retrieval: Strengths and Weaknesses

- Strengths
  - Precise, if you know the right strategies
  - Precise, if you have an idea of what you're looking for
  - Implementations are fast and efficient
- Weaknesses
  - Users must learn Boolean logic
  - Boolean logic insufficient to capture the richness of language
  - No control over size of result set: either too many hits or none
  - When do you stop reading? All documents in the result set are considered "equally good"
  - What about partial matches? Documents that "don't quite match" the query may be useful also

Next time, we'll talk about "ranked retrieval" with the vector space model

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.
- <u>What is a unit document</u>?
  - A file?
  - An email? (Perhaps one of many in an mbox.)
  - An email with 5 attachments?
  - A group of files (PPT or LaTeX in HTML)

Tokenization

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

Pair up … and …

Create a set of rules to tokenize this paragraph:

The Texas A&M Aggies, buoyed by their victory over South Carolina, moved up 12 spots to No. 9 in the AP Top 25 after the opening weekend of college football. The top four in the rankings -- Florida State, Alabama, Oregon and Oklahoma -- are unchanged, but the No. 1 Seminoles and No. 2 Crimson Tide lost some support in the first poll of the regular season after close victories against heavy underdogs. Texas A&M began the post-Johnny Manziel era with a 52-28 victory at South Carolina. The loss dropped the Gamecocks from No. 9 to No. 21.

What are the tokens emitted by your approach?

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１８岁生日。生日派对上，莎拉波娃露出了甜美的微笑。

Ambiguous segmentation in Chinese

和尚

- Can be treated as one word meaning "monk" or as two words meaning "and" and "still"

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!

Other cases of “no whitespace”

- Compounds in Dutch and German
- Computerlinguistik → Computer + Linguistik
- Lebensversicherungsgesellschaftsangestellter
- → leben + versicherung + gesellschaft + angestellter
- Inuit: tusaatsiarunnanngittualuujunga (I can’t hear very well.)
- Swedish, Finnish, Greek, Urdu, many other languages

Language issues in French

- ***L'ensemble*** → one token or two?
  - ***L*** ? ***L'*** ? ***Le*** ?
  - Want ***l'ensemble*** to match with ***un ensemble***

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.

  ← → ← → ← start
- ‘Algeria achieved its independence in 1962 after 132 years of French occupation.’
- Bidirectionality is not a problem if text is coded in Unicode

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***
- We most commonly implicitly define **equivalence classes** of terms
  - e.g., by deleting periods in 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
- Why don’t you want to put *window, Window, windows,* and *Windows* in the same equivalence class?

Normalization: other languages

- Accents: ***résumé*** vs. ***resume***.
- Most important criterion:
  - How are your users likely to write their queries for these words?
- Even in languages that standardly have accents, users often may not type them
- German: Tuebingen vs. Tübingen
  - Should be equivalent

Normalization: other languages

- Need to "normalize" indexed text as well as query terms into the same form

  7月30日 vs. 7/30

- Character-level alphabet detection and conversion
  - Tokenization not separable from this.
  - Sometimes ambiguous:

Morgen will ich in $\boxed{\text{MIT}}$ …

Is this German "mit"?

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 dictionary entirely the commonest words. Intuition:
  - They have little semantic content: *the, a, and, to, be*
  - They take a lot of space: ~30% of postings for top 30
- But the trend is away from doing this:
  - Good compression techniques means the space for including stopwords 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"

More equivalence classing

- Soundex: Chapter 3
  - phonetic equivalence: Tchebyshev = Chebysheff
- Thesaurus: Chapter 9
  - semantic equivalence: car = automobile

Lemmatization

- Reduce inflectional/variant forms to base form
- Example: am, are, is → be
- Example: car, cars, car's, cars' → car
- Example: the boy's cars are different colors → the boy car be different color
- Lemmatization implies doing "proper" reduction to dictionary headword form (the lemma).
- Inflectional morphology (cutting → cut) vs. derivational morphology (destruction → destroy)

Stemming

- Definition of stemming: Crude heuristic process that chops off the ends of words in the hope of achieving what “principled” lemmatization attempts to do with a lot of linguistic knowledge.
- Language dependent
- Often inflectional and derivational
- Example for derivational: **automate, automatic, automation** all reduce to **automat**

Porter algorithm

- Most common algorithm for stemming English
- Results suggest that it is at least as good as other stemming options
- Conventions + 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
- Sample convention: Of the rules in a compound command, select the one that applies to the longest suffix.

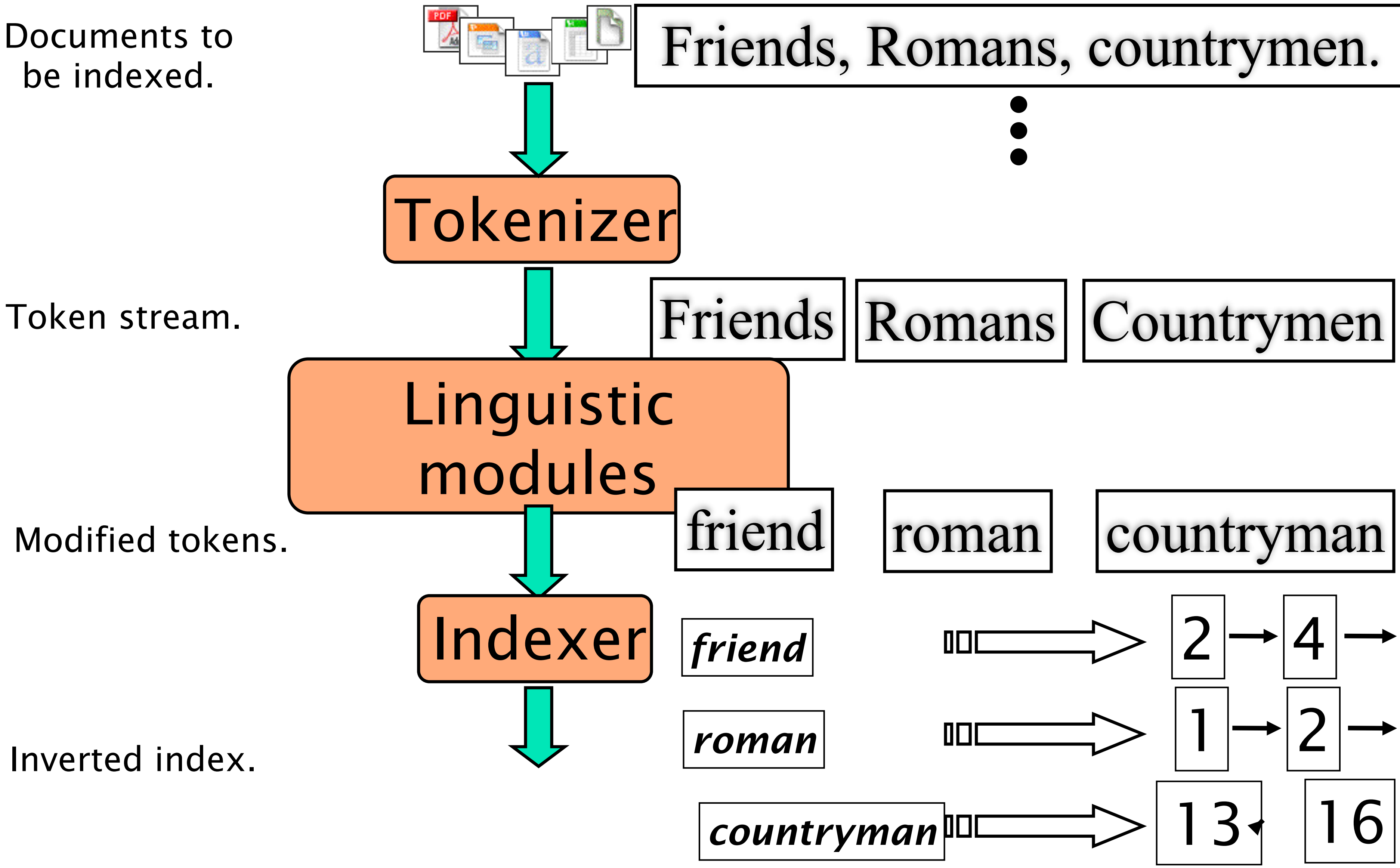
Porter stemmer: A few rules

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

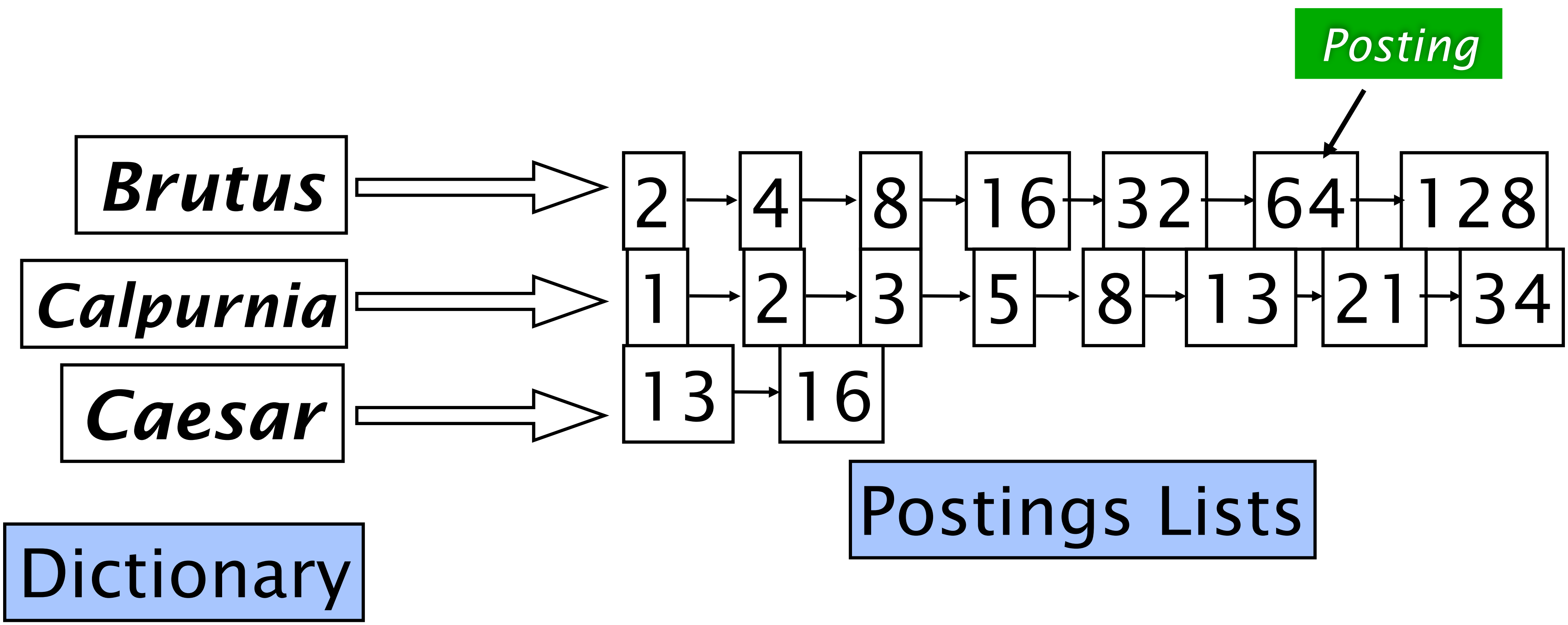
Recall Basic Indexing Pipeline



Dictionaries

Inverted index

- For each term t, we store a list of all documents that contain t



Dictionaries

- The dictionary is the data structure for storing the term vocabulary
- Term vocabulary: the data

Dictionary as array of fixed-width entries

- For each term, we need to store a couple of items
  - document frequency
  - pointer to postings list
  - ...
- Assume for the time being that we can store this information in a fixed-length array
- Assume that we store these entries in an array

Dictionary as array of fixed-width entries

term	document frequency	pointer to postings list
a	656,265	$\longrightarrow$
aachen	65	$\longrightarrow$
...	...	...
zulu	221	$\longrightarrow$

- How do we look up an element in this array at query time?

Data structures for looking up term

- Two main classes of data structure
  - hashes and trees
- Some IR systems use hashes, some use trees
- Criteria for when to use hashes vs trees
  - Is there a fixed number of terms or will it keep growing?
  - What are the relative frequencies with which various keys will be accessed?
  - How many terms are we likely to have?

Hashes

- Each vocabulary term is hashed to an integer
- Try to avoid collisions
- At query time, do the following: hash query term, resolve collisions, locate entry in fixed-width array
- Pros: hash lookup is faster than tree lookup
- Cons:
  - No way to find minor variants
  - No prefix search (all terms starting with "auto"
  - Need to rehash everything periodically if vocabulary keeps growing

Trees

- Trees solve the prefix problem
- Simplest tree: binary tree
- Search is slightly slower than in hashes: O(logM), where M = size of vocabulary
  - O(logM) holds for balanced trees only
  - Rebalancing is expensive
- One alternative: B-trees

Alternative index structures

How can we improve on the basic index?

- Need a better index than simple <term: docs>
  - **Skip pointers:** faster postings merges
  - **Positional index:** Phrase queries and Proximity queries
  - **Permuterm index:** Wildcard queries
  - **k-gram index:** Wildcard queries and spell correction

Positional Indexes

Phrase queries

- Want 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

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 as we did with wild-cards:
- ***stanford university palo alto*** can be broken into the Boolean query on biwords:

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

Longer phrase queries

- Longer phrases are processed as we did with wild-cards:
- ***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!

Solution 2: Positional indexes

- Store, for each ***term***, entries of the form:

<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

- LIMIT! /3 STATUTE /3 FEDERAL /2 TORT
  Here, /*k* means "within *k* words of".
- Clearly, positional indexes can be used for such queries; biword indexes cannot.

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

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

Combination schemes

- These two approaches can be profitably combined
  - For particular phrases (***"Lada Gaga"***, ***"Steve Jobs"***) it is inefficient to keep on merging positional postings lists
    - Even more so for phrases like ***"The Who"***
- Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme
  - A typical web query mixture was executed in ¼ of the time of using just a positional index
  - It required 26% more space than having a positional index alone

Positional Indexes: Wrap-up

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