Australian National University

COMP4650/6490 Document Analysis Autumn - 2021

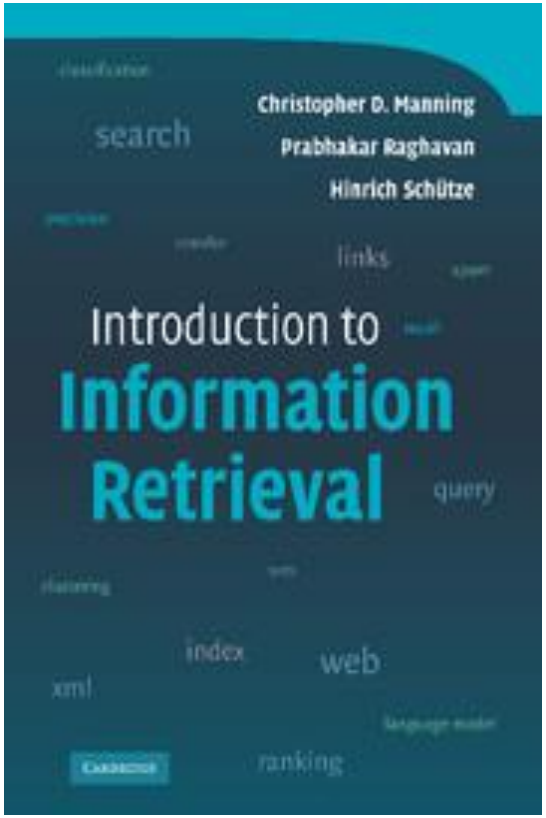
Introduction to Information Retrieval

School of Computing, ANU

IR Module Overview

- Information retrieval (IR) part consists of **four** lectures:
  1. Introduction to IR + Boolean model
  2. Ranked retrieval model
  3. Evaluation of IR systems
  4. Web search basics

Textbook



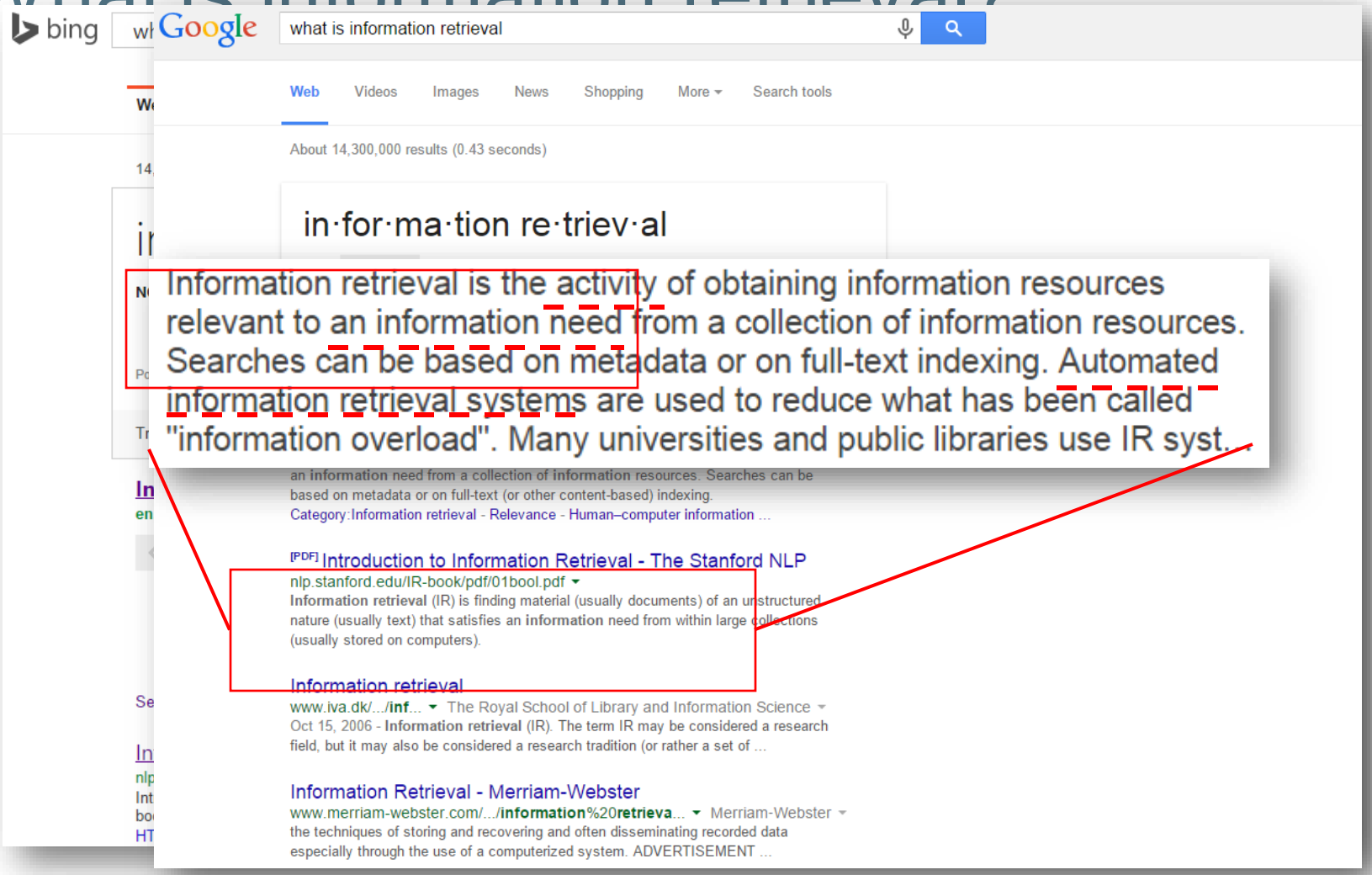
- Introduction to information retrieval
- https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf
- Chapters: 1, 2, 4, 6, 8, 19

Table of Contents

- Lecture Overview
- Introduction to Boolean Retrieval
  - Information Retrieval
  - Term-Document Matrix
  - Inverted Index
  - Boolean Retrieval with Inverted Index
  - Document Tokenization

What is Information Retrieval

What is information retrieval?



Information Retrieval

- "Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)."

  Manning et al.

- You may think of *web search* first, but there are many other cases
  - E-mail search
  - Searching your laptop
  - Corporate knowledge bases
  - Image search, video search

Why information retrieval

- Information overload
  - "*It refers to the difficulty a person can have understanding an issue and making decisions that can be caused by the presence of too much information.*" - wiki

Why information retrieval

- An essential tool to deal with information overload

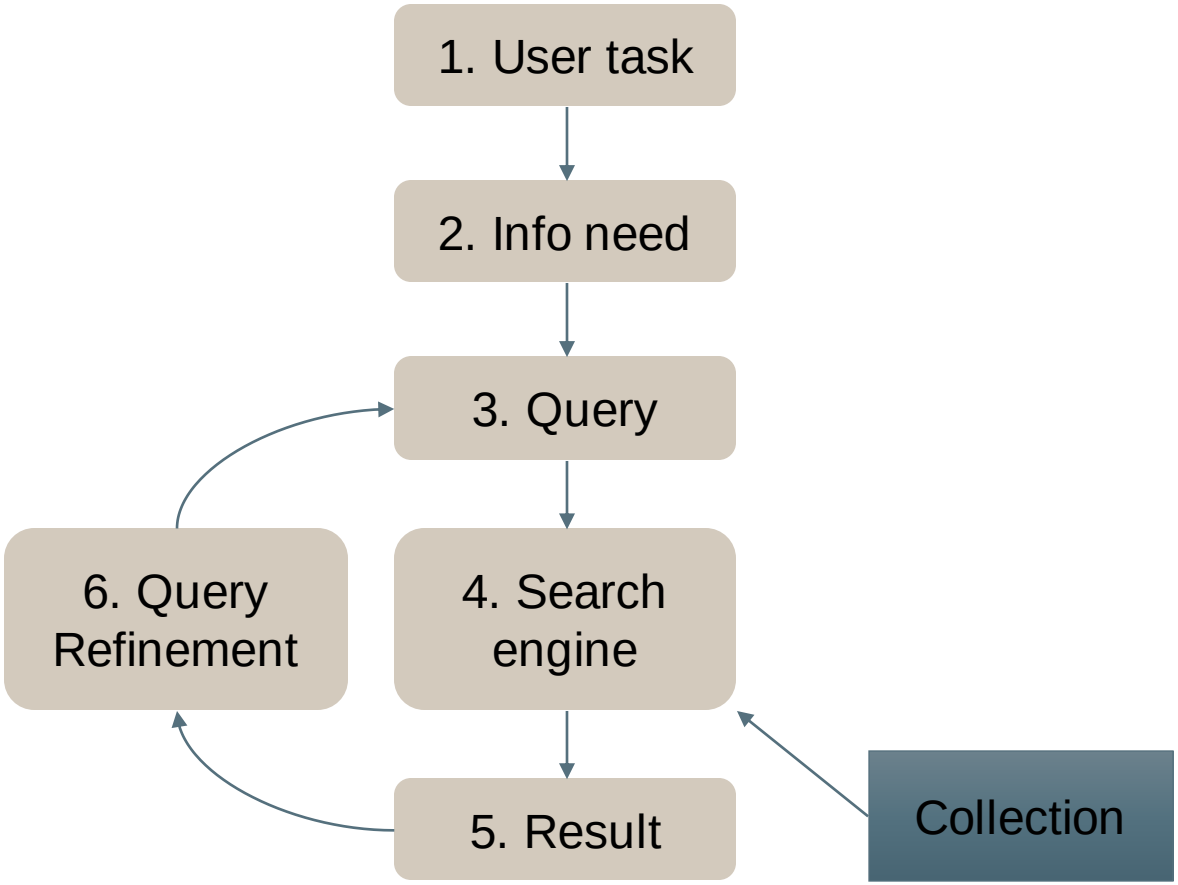
How to perform information retrieval

- Information retrieval when we did not have a computer

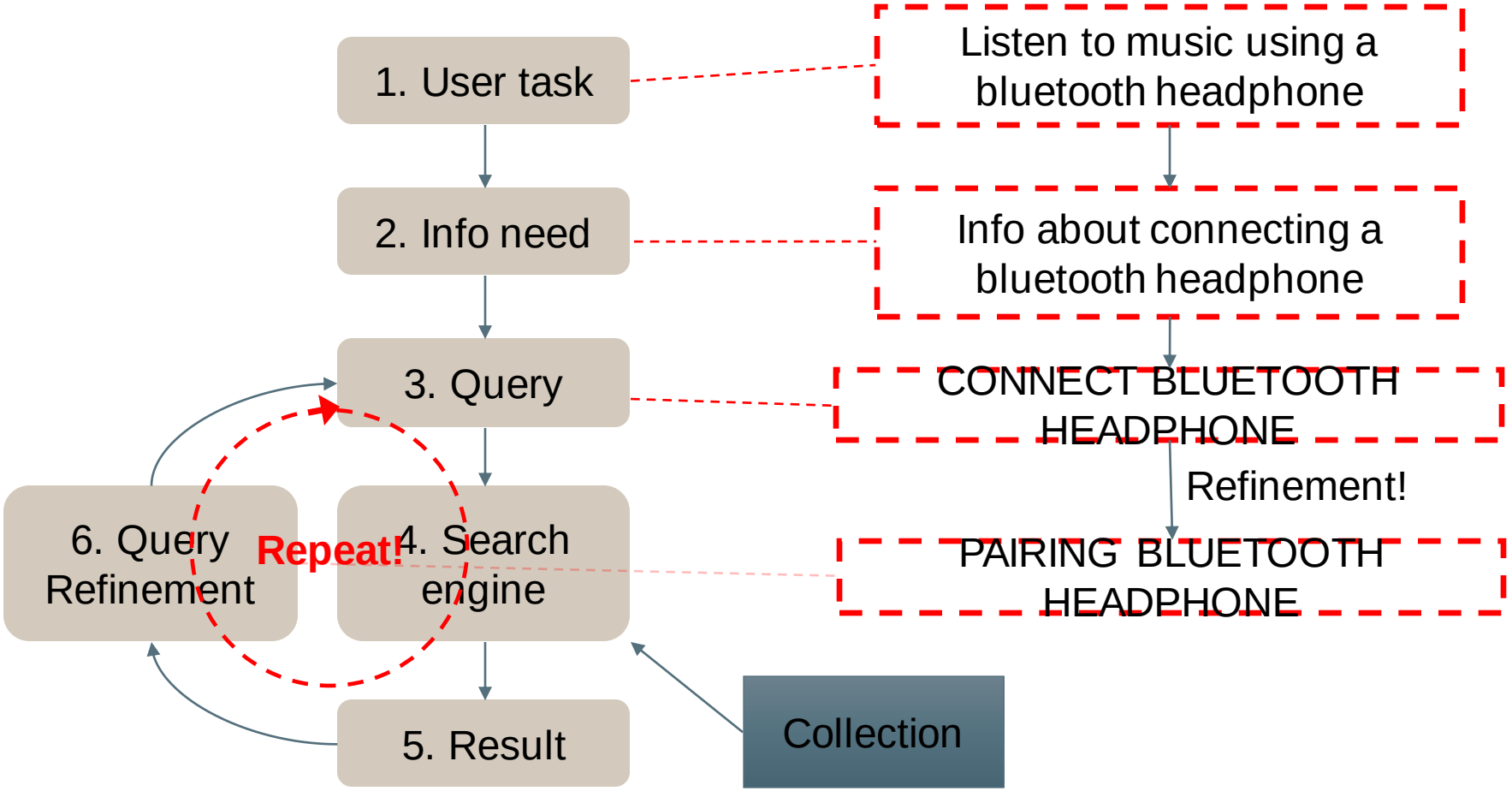
Basic Assumptions

- **Collection**: A set of *documents*
  - Assume it as a static collection for the moment
- **Goal**: Retrieve documents with information that is relevant to the user's *information need* and helps the user complete a task
  - User's information need is often underspecified

Classic Search Model



Classic Search Model



Key Objectives

- Every good IR system needs to achieve
  - **Effectiveness**
    - More than 130 trillion pages are indexed by Google
  - **Accuracy**
    - Top 10 pages from 130 trillion pages?

IR vs. NLP

- Information retrieval
  - Computational approaches
  - Statistical (shallow) understanding of language
  - Handle large scale problems
- Natural language processing
  - Cognitive, symbolic and computational approaches
  - Semantic (deep) understanding of language
  - (often times) small scale problems

IR and NLP are getting closer

- IR => NLP
  - Larger data collections
  - Scalable/robust NLP techniques, e.g., translation models
- NLP => IR
  - Deep analysis of text documents and queries
  - **Information extraction** for structured IR tasks

Search with Boolean query

- Boolean query
  - E.g., “obama” AND “healthcare” NOT “news”
- Procedures
  - Lookup query term in the dictionary
  - Retrieve the posting lists
  - Operation
    - AND: intersect the posting lists
    - OR: union the posting list
    - NOT: diff the posting list

Retrieval procedure in modern IR

- Boolean model provides **all** the ranking candidates
  - Locate documents satisfying Boolean condition
    - E.g., “obama healthcare” -> “obama” OR “healthcare”
- Rank candidates by relevance
  - Important: the notation of relevance
- Efficiency consideration
  - Top-k retrieval (Google)

Term-document incidence matrices

	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

- 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 10111 = **100100**

	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

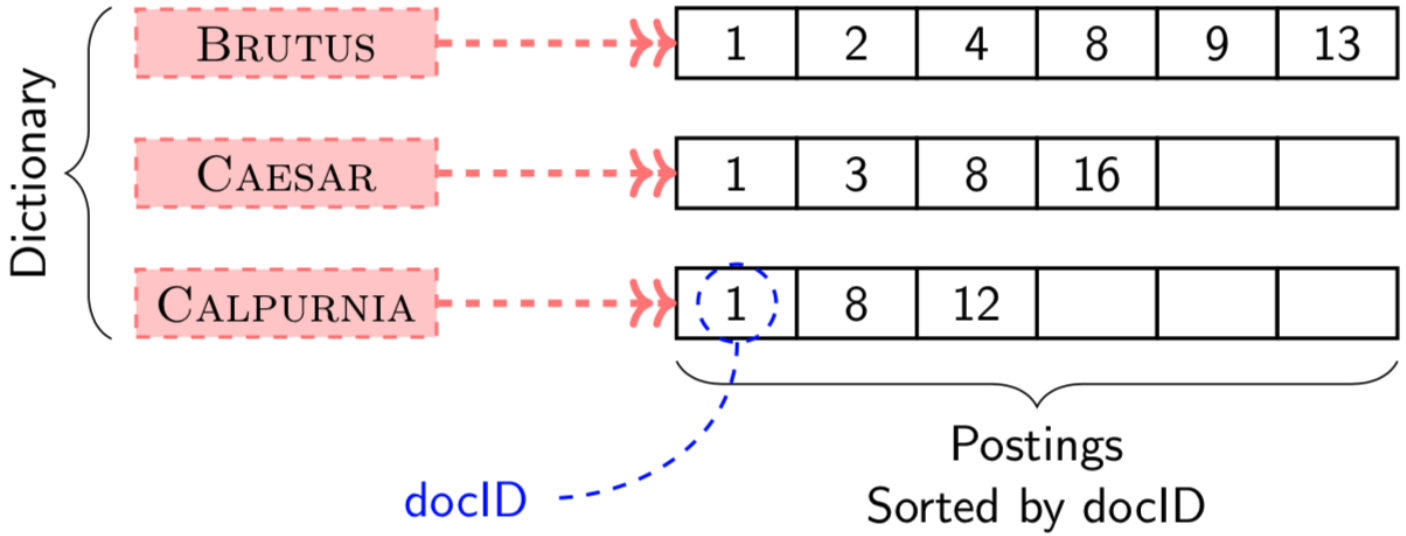
Efficiency

- Bigger Collections
  - 1 million documents
  - Each 1,000 words long
- Avg 6 bytes/word including spaces/punctuation
  - 6GB of data in the documents.
- Assume there are M = 500K distinct terms among these.
  - Corresponds to a matrix with 500 billion entries
  - But it has no more than one billion 1's
  - Extremely sparse matrix!

Efficient data structure tailored for boolean retrieval
=> Inverted index!

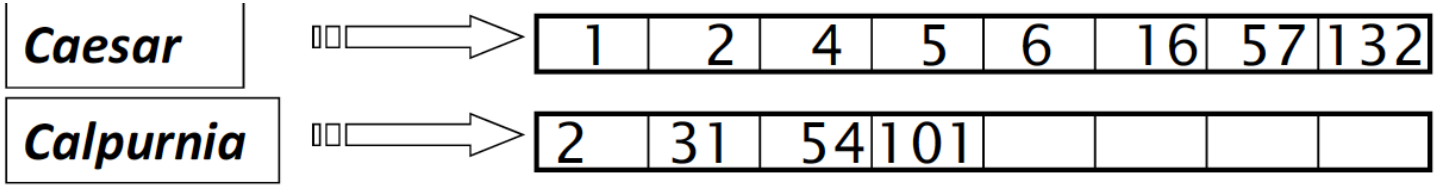
Inverted Index

- Inverted index consists of a *dictionary* and *postings*
  - **Dictionary**: a set of unique terms
  - **Posting**: variable-size array that keeps the list of documents given term
- INDEXER: Construct inverted index from raw text



Inverted Index

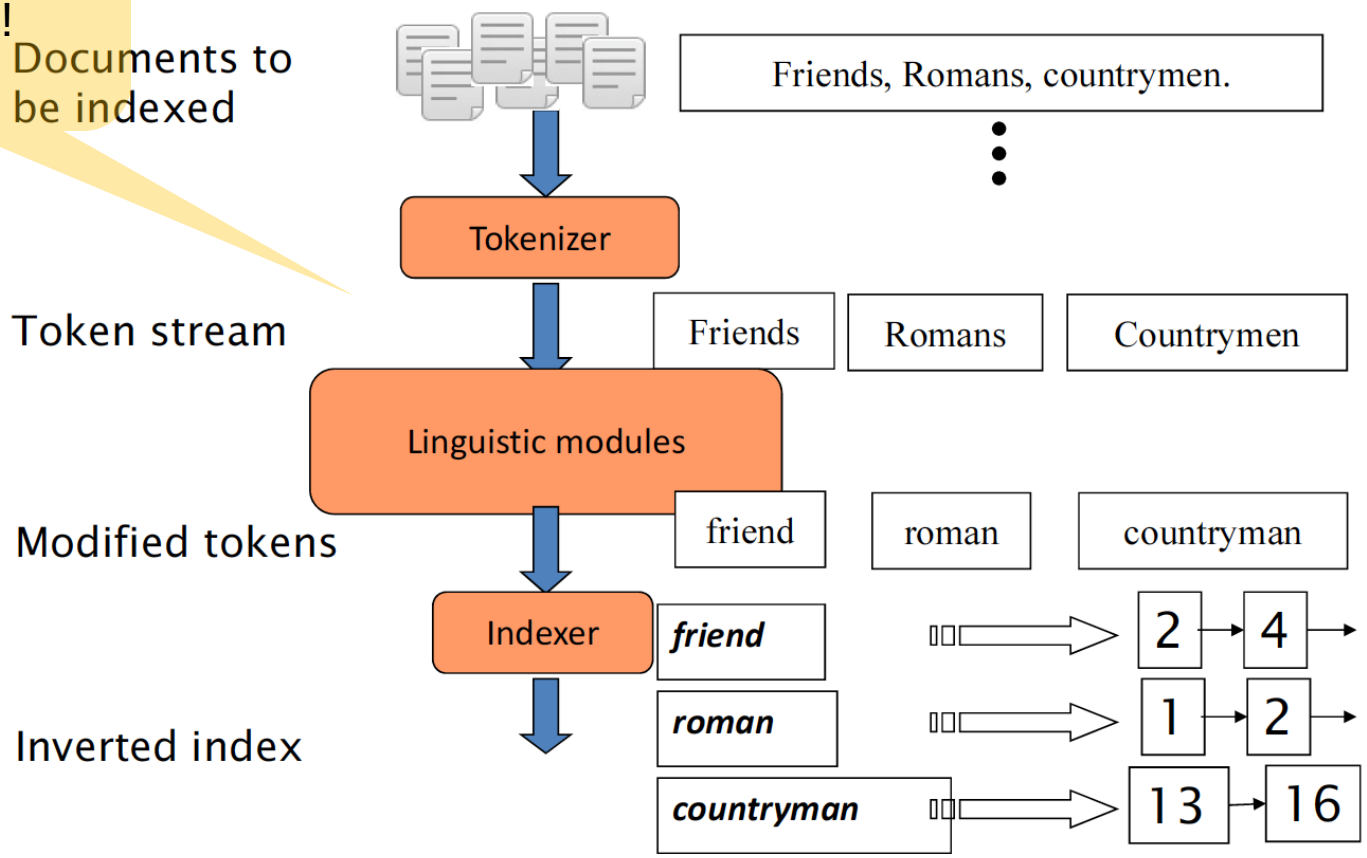
- For each term t, we must store a list of all documents that contain t.
  - Identify each doc by a **docID**, a document serial number
- We need variable-size postings lists
  - On disk, a continuous run of postings is normal and best
  - In memory, can use linked lists or variable length arrays



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

Inverted Index Construction

More in NLP module!

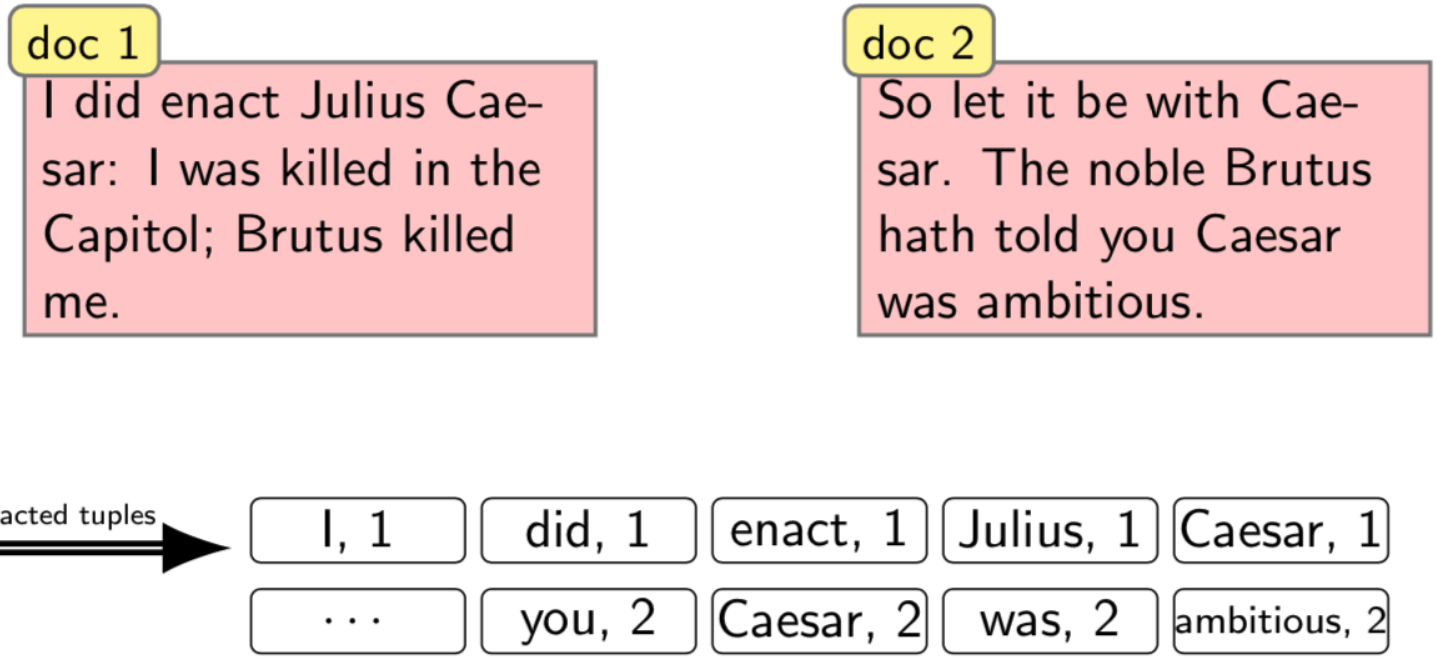


Initial Stages of Text Processing

- 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
  - E.g. authorize, authorization
- Stop words
  - We may omit very common words (or not)
  - E.g. the, a, to, of

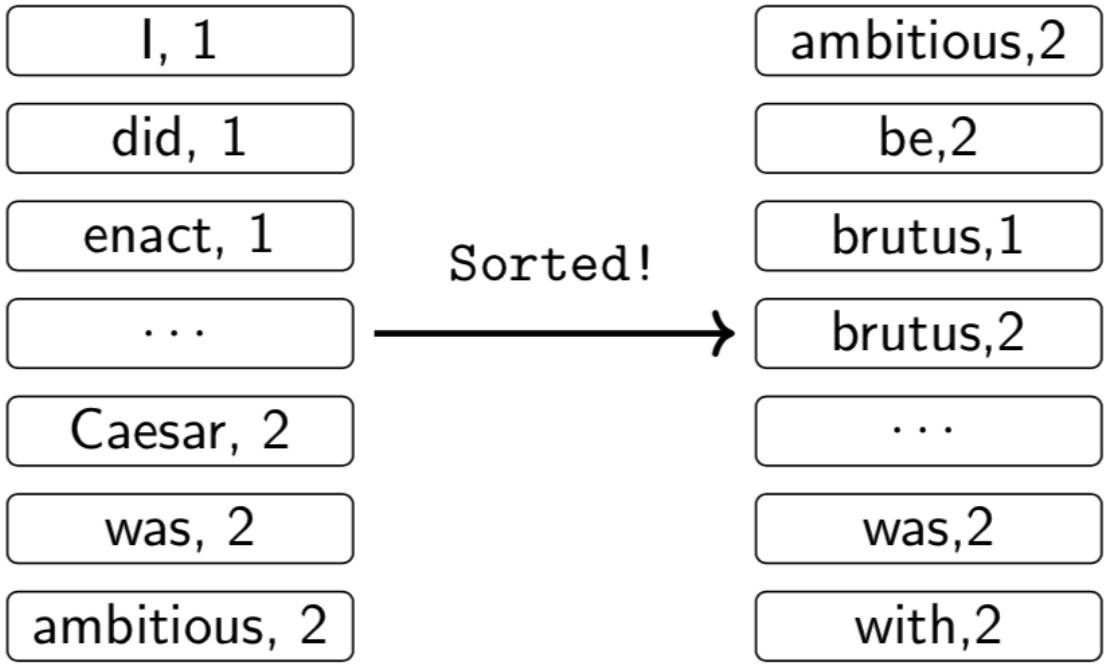
Indexer Step 1: Token sequence

- Scan documents for indexable terms
- Keep list of (token, docID) pairs.



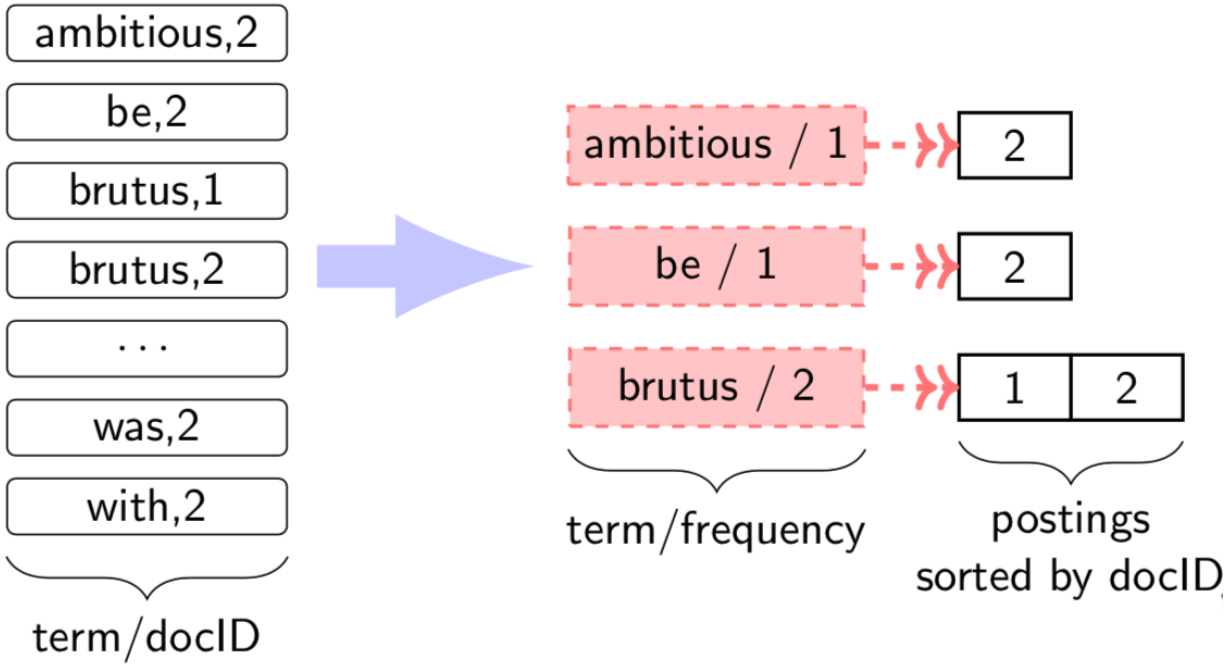
Indexer Step 2:

- Sort tuples by terms (and then docID)



Indexer Step 3:

- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings
- Doc. frequency information is added.



Boolean Retrieval with Inverted Index

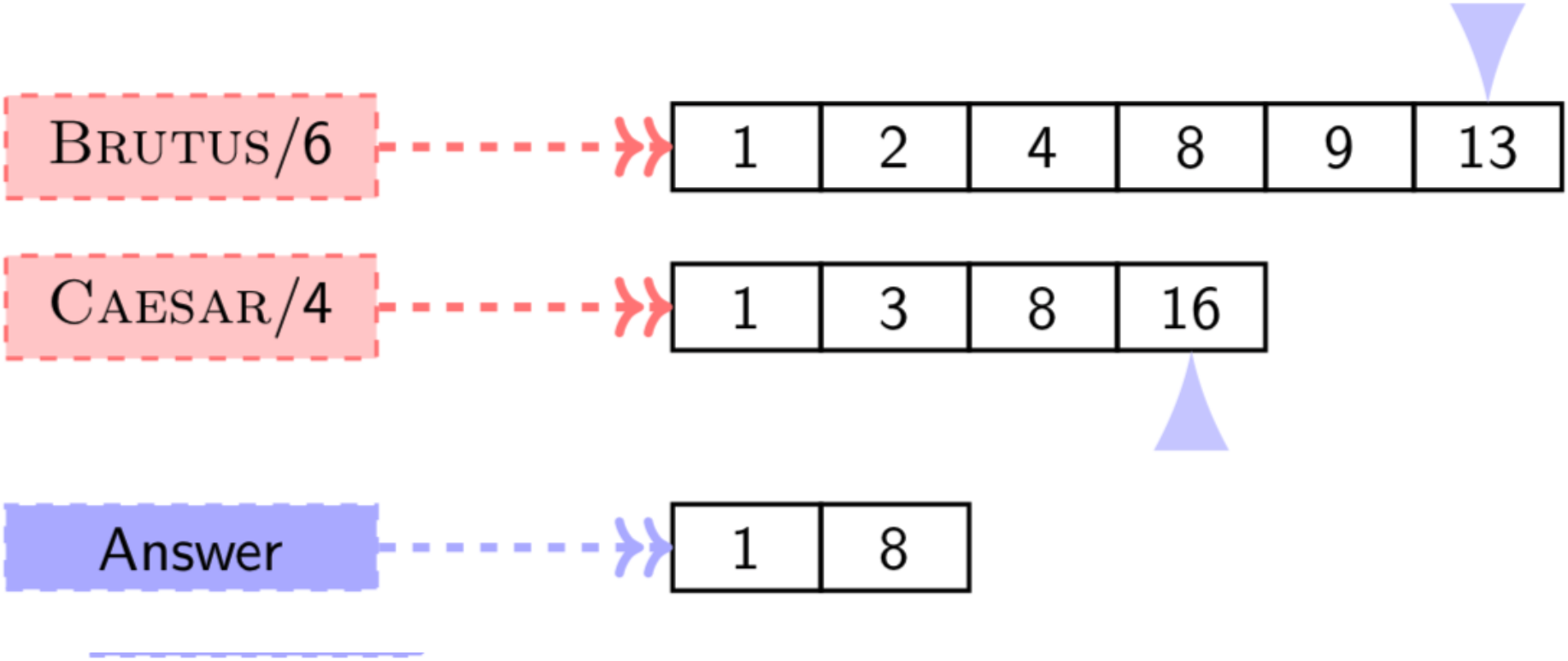
- Easy to retrieve all documents containing term *t*
- How to find documents containing *BRUTUS AND CEASAR* using postings?

  → Linear time retrieval algorithm

Algorithm 1 Intersection(p_1, p_2)

```
1:  answer ← {}
2:  while p1 ≠ NIL and p2 ≠ NIL do
3:      if docID(p1) = docID(p2) then
4:          ADD(answer, docID(p1)); p1 ← next(p1); p2 ← next(p2)
5:      else if docID(p1) < docID(p2) then
6:          p1 ← next(p1)
7:      else
8:          p2 ← next(p2)
9:      end if
10: end while
```

Intersection



Boolean Retrieval

- Can answer any query which is a Boolean expression: AND, OR, NOT
- Precise: each document matches, or not
- Extended Boolean allows more complex queries
- Primary commercial search for 30+ years, and still very popular

A Step back

- So far we assumed that we can easily scan terms from a document.
- But the scanning consists of following steps:
  - Tokenization
  - Stopwords
  - Normalization
  - Stemming and Lemmatization

Tokenization

- Task of chopping a document into *tokens*
- Chopping by *whitespace* and throwing punctuation are not enough.
- Examples:
  - New York, San Francisco
  - Phone numbers ((800) 234 2333) and dates (Mar 11, 1983)
  - LOWERCASE and LOWER CASE
  - Hyphenation
    - Co-education, Hewlett-Packard, doc-ument (line end)
- Regex tokenizer: simple and efficient
- Whatever method you use, always do **the same** tokenization of document and query text

Stopwords Removal & Normalization

- Stopwords removal
  - Stop words usually refer to the most common words in a language.
  - E.g. the, a, an, and, or, will, would, could
  - Reduce the number of postings that a system has to store
  - These words are not very useful in keyword search.
- Normalization
  - Keep **equivalence class** of terms: U.S.A = USA = united states
  - Synonym list: car = automobile
  - Capitalization: ferrari → Ferrari
  - Case-folding: Automobile → automobile

Stemming and Lemmatization

- {run, running, ran} → run
- Stemming
  - Stemming turns tokens into stems, which are the same regardless of inflection.
  - Stems need not be real words

Rule	Example
Replace ies with i	ponies → poni
Replace y with i	pony → poni
Remove ing	falling → fall
Remove s	dogs → dog
Remove es	replaces → replac
Remove e	replace → replac

Table: Porter Stemmer[1]

- Lemmatization

  Lemmatization turns words into lemmas, which are dictionary entries.

  The word “better” has “good” as its lemma.

Summary

- Lecture Overview
- Introduction to Boolean Retrieval
  - Information Retrieval
  - Term-Document Matrix
  - Inverted Index
  - Boolean Retrieval with Inverted Index
  - Document Tokenization
- Any questions?

References

- Some lecture slides are from:
  - Hongning Wang (2017) CS 4501/6501: Information retrieval, CS@Uva
  - Pandu Nayak and Prabhakar Raghavan, CS276, Information Retrieval and Web Search, Stanford University