



# **F033583 INTRODUCTION TO WEB SEARCH AND MINING**

1

**Kenny Q. Zhu (朱其立)**

**Dept. of Computer Science**

**Shanghai Jiao Tong University**

# KENNY Q. ZHU



## Research Interests:

### NLP & Knowledge Engineering

Information extraction

Knowledge discovery

Commonsense reasoning

Text generation (QA/QG/Summarization)

### Programming Languages

Concurrent and Distributed Languages

Data Processing

Probabilistic Programming

## Recent Publications:

AAAI 2019, ACL 2018, EMNLP 2018, AAAI 2018, IJCAI 2017, AAAI 2016, KR 2016, AAAI 2015, EMNLP 2015

Degrees:

*National University of Singapore*

Postdoc:

*Princeton University*

Experiences:

*Microsoft Redmond, USA*

*Microsoft Research Asia*

Faculty Member (PhD. Advisor) at *SJTU* since 2009

Director of **ADAPT Lab**



# ADMINISTRATIVE INFO (I)

- All-English Course: everything in English!
- Lecturer:
  - Kenny Zhu, SEIEE #03-541, [kzhu@cs.sjtu.edu.cn](mailto:kzhu@cs.sjtu.edu.cn)
  - Office hours: by appointment or after class
- Teaching Assistant:
  - Flora Huang, SEIEE #03-341, [florahuangss@163.com](mailto:florahuangss@163.com)
  - Kelsey Huang, SEIEE #03-341, [shengleihuang@sjtu.edu.cn](mailto:shengleihuang@sjtu.edu.cn)
  - Office hours: Thursday 16:00 - 17:00
- Course Web Page (definitive source!):  
<http://www.cs.sjtu.edu.cn/~kzhu/wsm/>

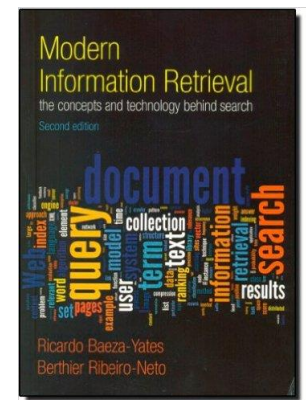
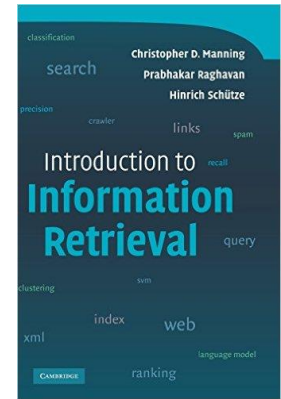
# ADMINISTRATIVE INFO (II)

## ○ Format:

- Lectures on Friday (3 periods with 1-2 breaks)
- Part of lectures may be tutorials – Led by Flora/Kelsey; Your participation is **REQUIRED!**

## ○ Reference Texts:

- Introduction to Information Retrieval, by Christopher D. Manning and Prabhakar Raghavan
- Modern Information Retrieval: The Concepts and Technology behind Search (2nd Edition), by Ricardo Baeza-Yates, Berthier Ribeiro-Neto
- Mining the Web: Discovering Knowledge from Hypertext Data Hardcover , by Soumen Chakrabarti
- Web Information Retrieval (Data-Centric Systems and Applications), by Stefano Ceri and Alessandro Bozzon
- Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data (Data-Centric Systems and Applications), by Bing Liu



# ADMINISTRATIVE INFO (III)

- 2-credit course
- Modes of Assessment:
  - In-class quizzes: 30%
  - Assignments: 30%
  - Programming Project: 40%
- Quizzes
  - Given out at random times during class
  - Submit at the end of class
  - Bring piece of paper and a pen every time!
- Tutorials (ad hoc)
  - Discuss hard assignment questions and also issues in project
  - You may be asked to present your answers
  - Volunteer to win extra scores!

# ADMINISTRATIVE INFO (IV)

## ○ Assignments

- Released (usually) on Friday
- Hard copies due next Friday before class
- Submit hard copies to me or TA's office
- Submit code/data to Flora's email
- Late submission: -30% of full score for each additional day

## ○ Programming Project

- Group project (max 3 people)
- Implement a crawler to crawl large amount of data and then develop a search engine of the pages
- Produce a report + code + data: due end of semester

# WECHAT GROUP



WSM-2019课程群



该二维码7天内(3月8日前)有效, 重新进入将更新

# DISCLAIMER

- Part of the materials in this presentation were adapted from the slides created by Manning et al. of Stanford University and Schütze et al. of University of Stuttgart.





# BOOLEAN RETRIEVAL

9

# OUTLINE

- Introduction
- Inverted index
- Processing Boolean queries
- Query optimization

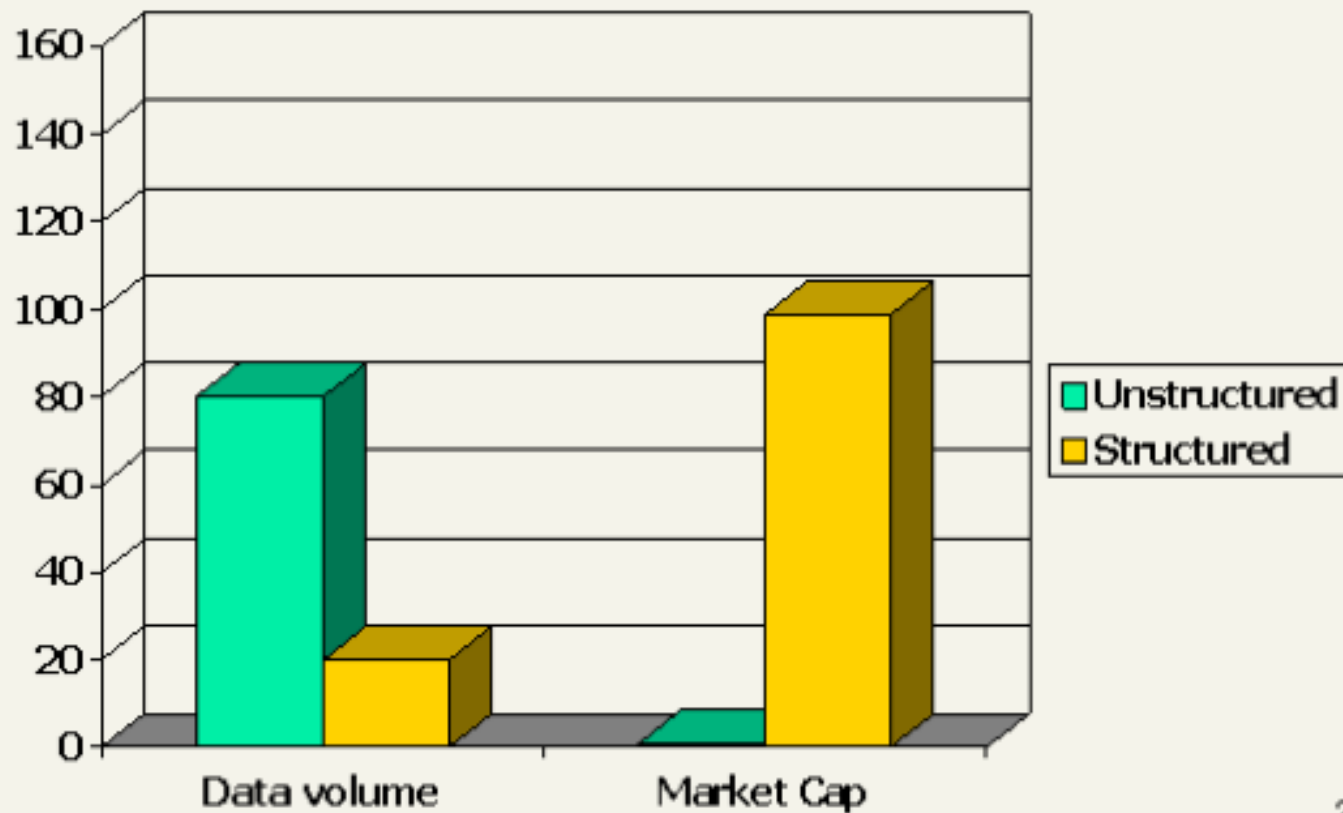
# Definition of *information retrieval*

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).

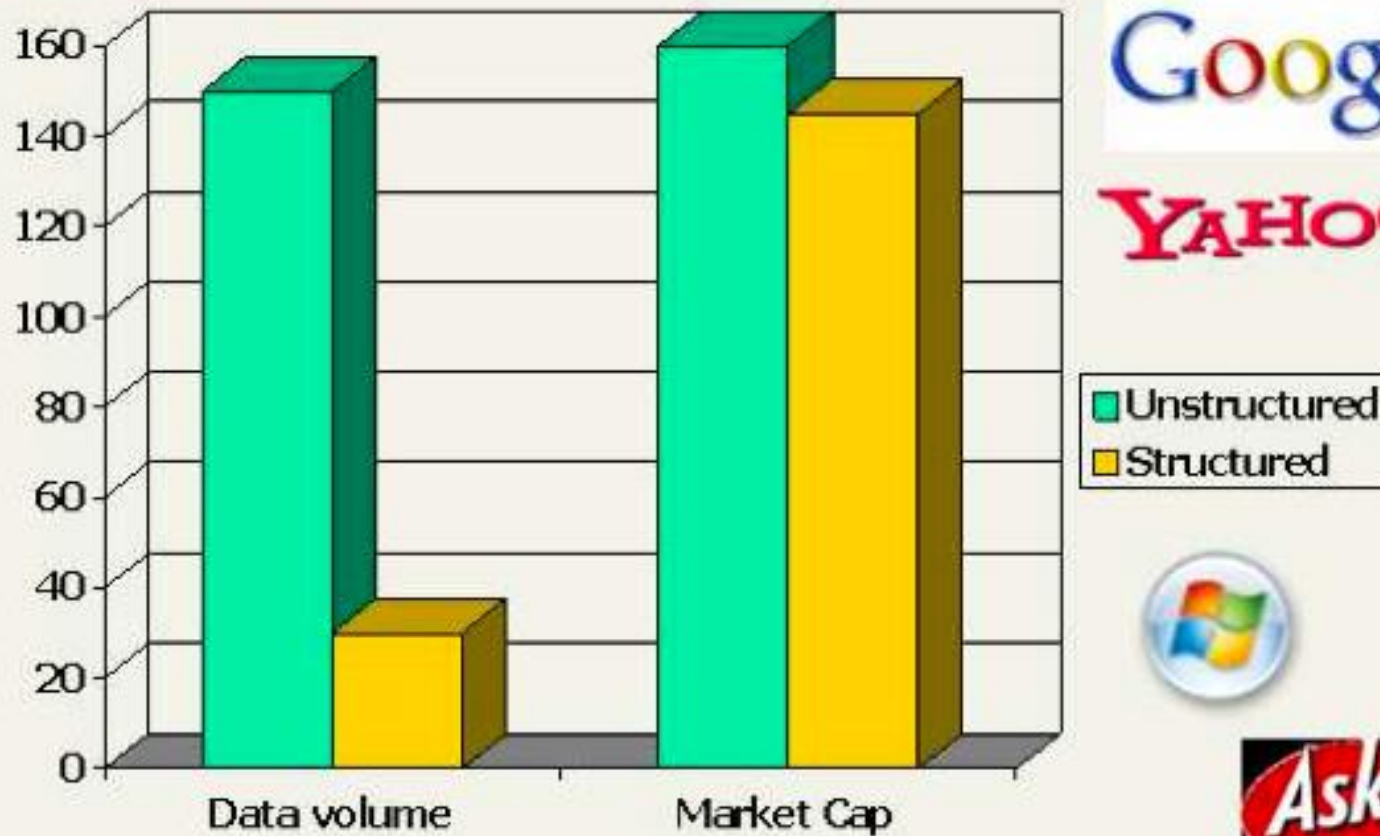
# HOW GOOD ARE THE RETRIEVED DOCS?

- *Precision* : Fraction of retrieved docs that are relevant to the user's *information need*
- *Recall* : Fraction of relevant docs in collection that are retrieved
  - More precise definitions and measurements to follow later

# Unstructured (text) vs. structured (database) data in 1996



# Unstructured (text) vs. structured (database) data in 2006



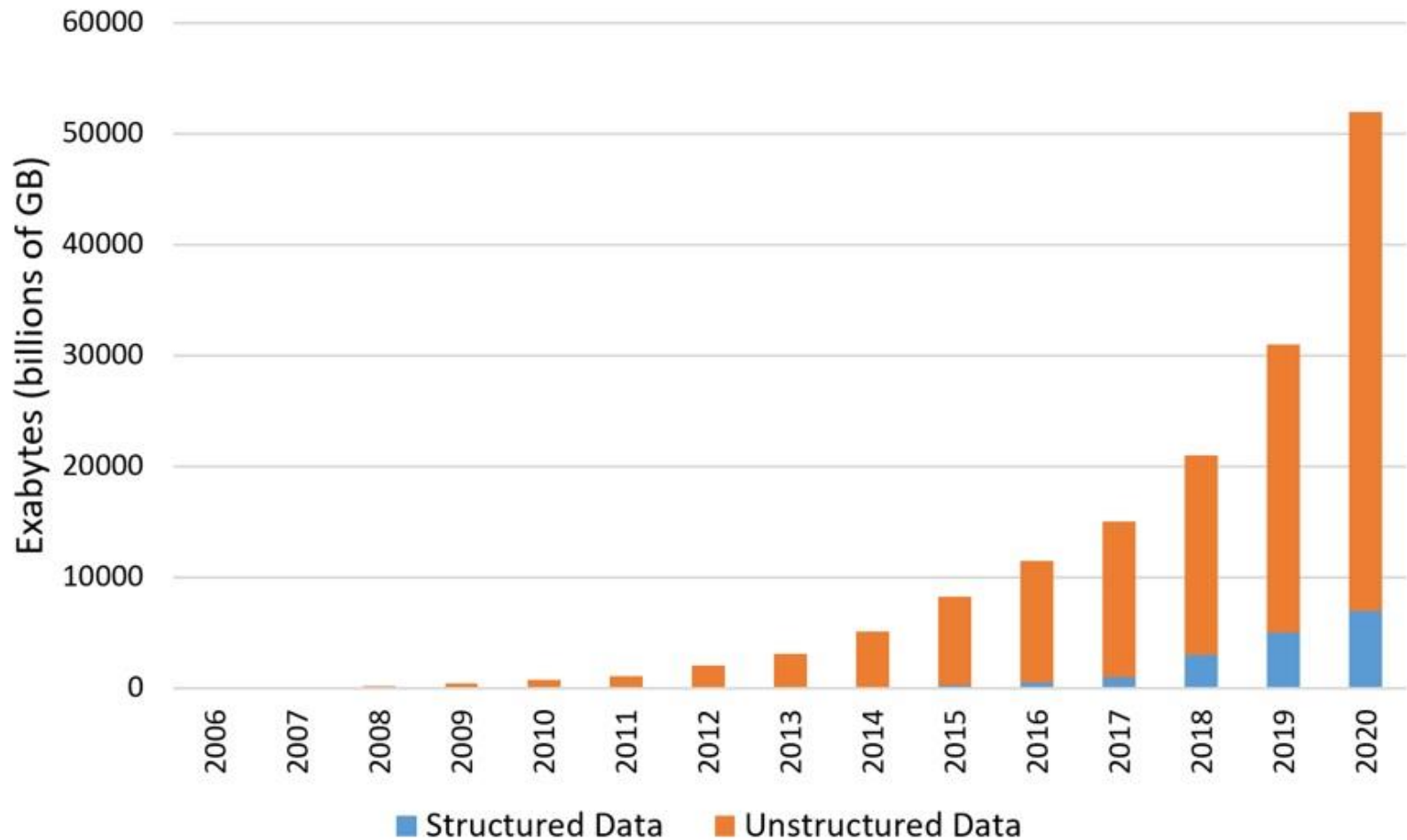
Google™

YAHOO!®



# AFTER 2006...

## The Cambrian Explosion...of Data



# Boolean retrieval

- The Boolean model is arguably the simplest model to base an information retrieval system on.
- Queries are Boolean expressions, e.g., CAESAR AND BRUTUS
- The search engine returns all documents that satisfy the Boolean expression.

Question: Does Google use the Boolean model?

Answer: Not quite...



# Does Google use the Boolean model?

- On Google, the default interpretation of a query  $[w_1 w_2 \dots w_n]$  is  $w_1 \text{ AND } w_2 \text{ AND } \dots \text{ AND } w_n$
- Cases where you get hits that do not contain one of the  $w_i$  :
  - anchor text
  - page contains variant of  $w_i$  (morphology, spelling correction, synonym)
  - long queries ( $n$  large)
  - boolean expression generates very few hits
- Simple Boolean vs. Ranking of result set
  - Simple Boolean retrieval returns matching documents in no particular order.
  - Google (and most well designed Boolean engines) rank the result set – they rank good hits (according to some estimator of relevance) higher than bad hits.

# OUTLINE

- Introduction
- **Inverted index**
- Processing Boolean queries
- Query optimization

# Unstructured data in 1650: Shakespeare



# Unstructured data in 1650

- Which plays of Shakespeare contain the words BRUTUS AND CAESAR, but not CALPURNIA?
- One could “grep” all of Shakespeare’s plays for BRUTUS and CAESAR, then strip out lines containing CALPURNIA
- Why isn’t “grep” a good solution?
  - Slow (for large collections)
  - grep is line-oriented, IR is document-oriented
  - “NOT CALPURNIA” is non-trivial
  - Other operations (e.g., find the word ROMANS near COUNTRYMAN ) not feasible

# Term-document incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
ANTHONY	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
...						

Entry is 1 if term occurs. Example: CALPURNIA occurs in *Julius Caesar*. Entry is 0 if term doesn't occur. Example: CALPURNIA doesn't occur in *The tempest*.

# Incidence vectors

- So we have a 0/1 vector for each term.
- To answer the query BRUTUS AND CAESAR AND NOT CALPURNIA:
  - Take the vectors for BRUTUS, CAESAR AND NOT CALPURNIA
  - Complement the vector of CALPURNIA
  - Do a (bitwise) **and** on the three vectors
  - $110100 \text{ AND } 110111 \text{ AND } 101111 = 100100$

# 0/1 vector for BRUTUS AND CAESAR AND NOT CALPURNIA:

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
ANTHONY	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
...						
result:	1	0	0	1	0	0

# Answers to query

*Anthony 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:

Julius Caesar:

I did enact  
I was killed i'  
the Capitol; Brutus killed me.



# Bigger collections

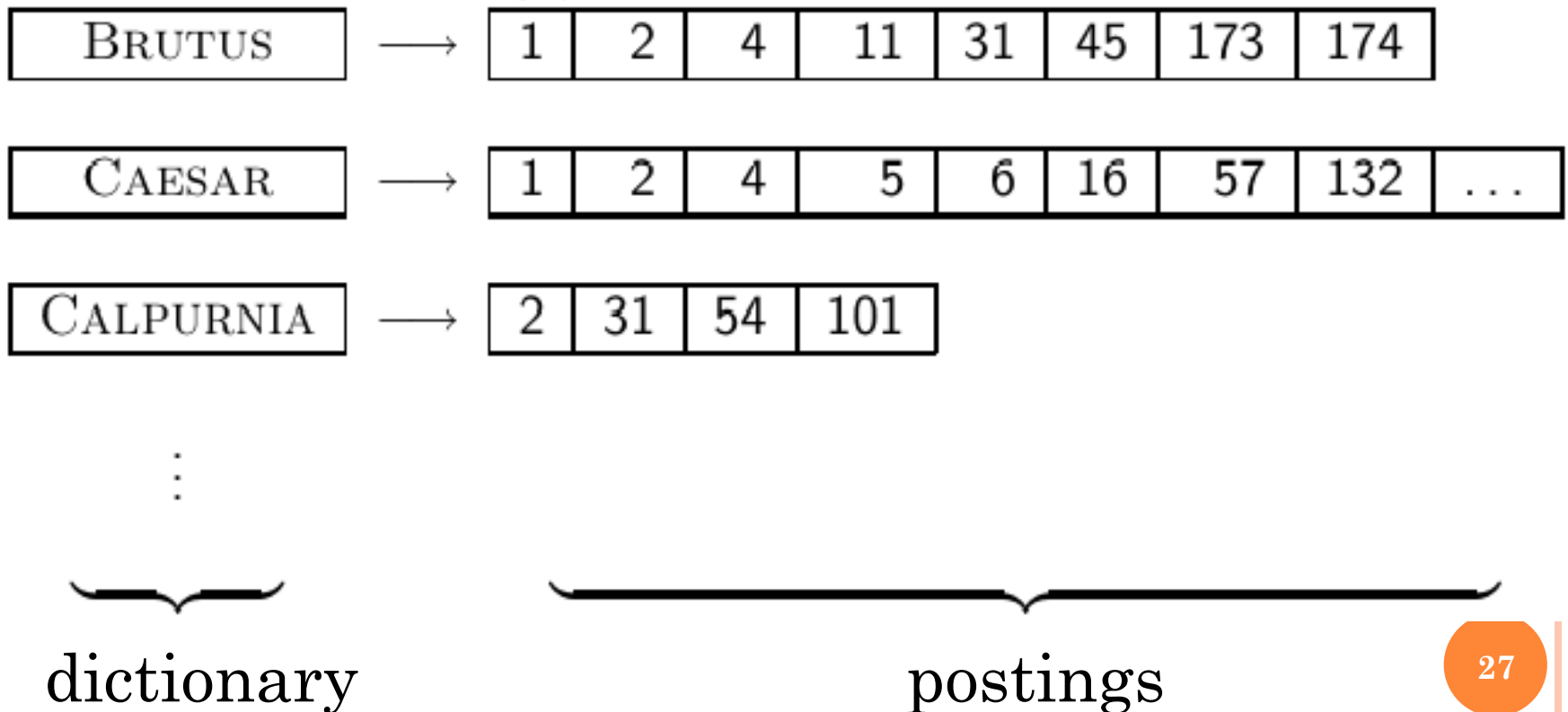
- Consider  $N = 10^6$  documents, each with about 1000 tokens
- $\Rightarrow$  total of  $10^9$  (1 billion) tokens
- On average 6 bytes per token, including spaces and punctuation
- $\Rightarrow$  size of document collection is about  $6 \cdot 10^9 = 6$  GB
- Assume there are  $M = 500,000$  distinct terms in the collection
- (Notice that we are making a term/token distinction.)

# Can't build the incidence matrix

- $M = 500,000 \times 10^6 =$  half a trillion 0s and 1s.
- But the matrix has no more than one billion 1s.
  - Matrix is extremely sparse.
- What is a better representations?
  - We only record the 1s.

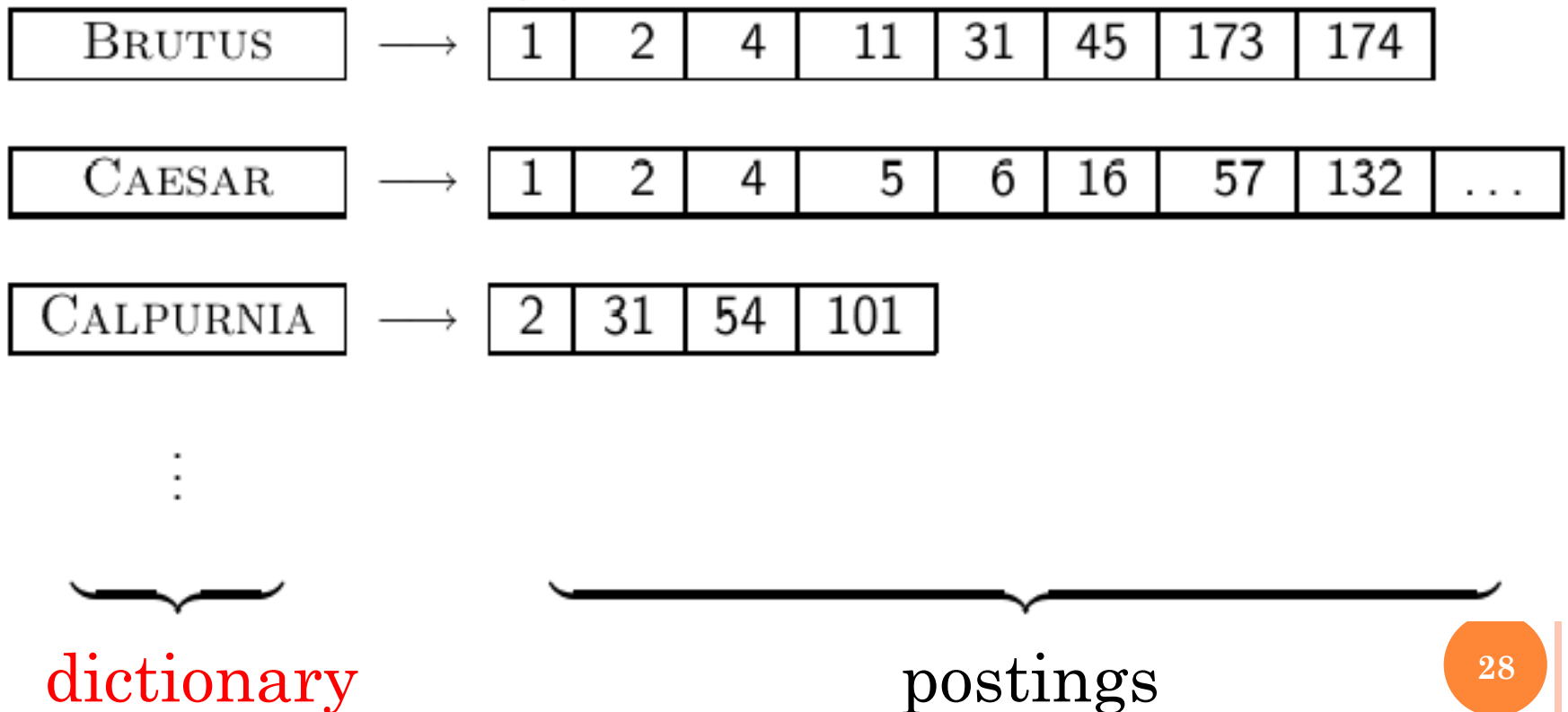
# Inverted Index

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



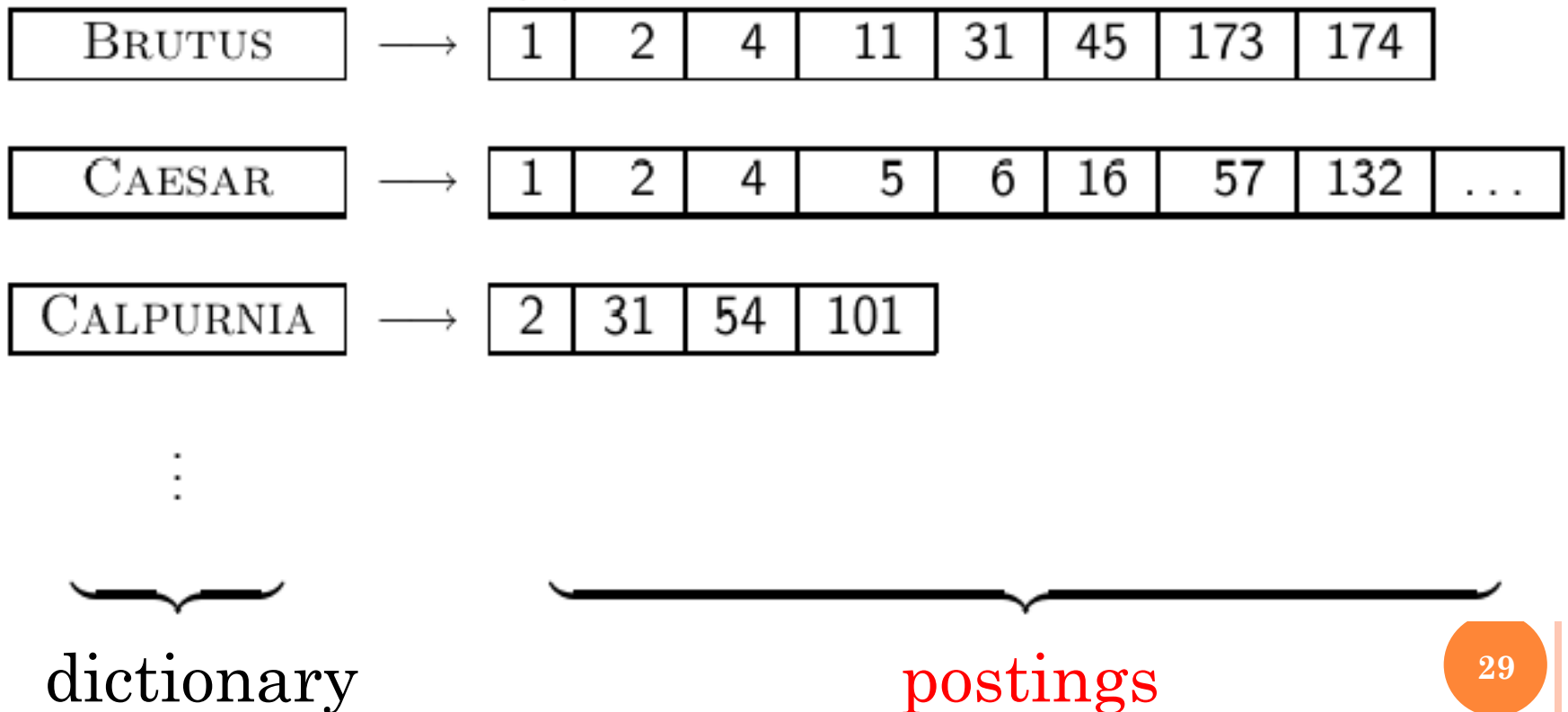
# Inverted Index

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



# Inverted Index

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



# INVERTED INDEX CONSTRUCTION

1. Collect the documents to be indexed:

Friends, Romans, countrymen. So let it be with Caesar ...

2. Tokenize the text, turning each document into a list of tokens:

Friends Romans countrymen So ...

3. Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms:

friend roman  
countryman so ...

4. Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.

# 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*

# Tokenizing and preprocessing

**Doc 1.** I did enact Julius Caesar: I was killed i' the Capitol; Brutus killed me.

**Doc 2.** So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious:



**Doc 1.** i did enact julius caesar i was killed i' the capitol brutus killed me

**Doc 2.** so let it be with caesar the noble brutus hath told you caesar was ambitious



# Generate posting

**Doc 1.** i did enact julius caesar i was  
killed i' the capitol brutus killed me  
**Doc 2.** so let it be with caesar the  
noble brutus hath told you caesar was  
ambitious



term	docID
i	1
did	1
enact	1
julius	1
caesar	1
i	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

# Sort postings

term	docID		term	docID
i	1		ambitious	2
did	1		be	2
enact	1		brutus	1
julius	1		brutus	2
caesar	1		capitol	1
i	1		caesar	1
was	1		caesar	2
killed	1		caesar	2
i'	1		did	1
the	1		enact	1
capitol	1		hath	1
brutus	1		i	1
killed	1		i	1
me	1	⇒	i'	1
so	2		it	2
let	2		julius	1
it	2		killed	1
be	2		killed	1
with	2		let	2
caesar	2		me	1
the	2		noble	2
noble	2		so	2
brutus	2		the	1
hath	2		the	2
told	2		told	2
you	2		you	2
caesar	2		was	1
was	2		was	2
ambitious	2		with	2

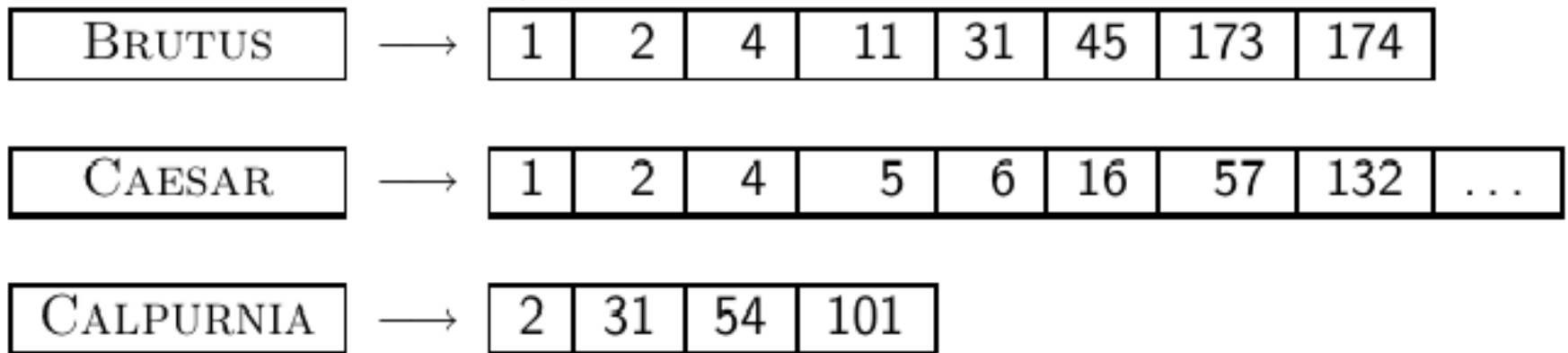
# Create postings lists, determine document frequency

term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
i	1
i	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2



term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

# Split the result into dictionary and postings file



dictionary  
postings

# Later in this course

- Index construction: how can we create inverted indexes for large collections?
- How much space do we need for dictionary and index?
- Index compression: how can we efficiently store and process indexes for large collections?
- Ranked retrieval: what does the inverted index look like when we want the “best” answer?

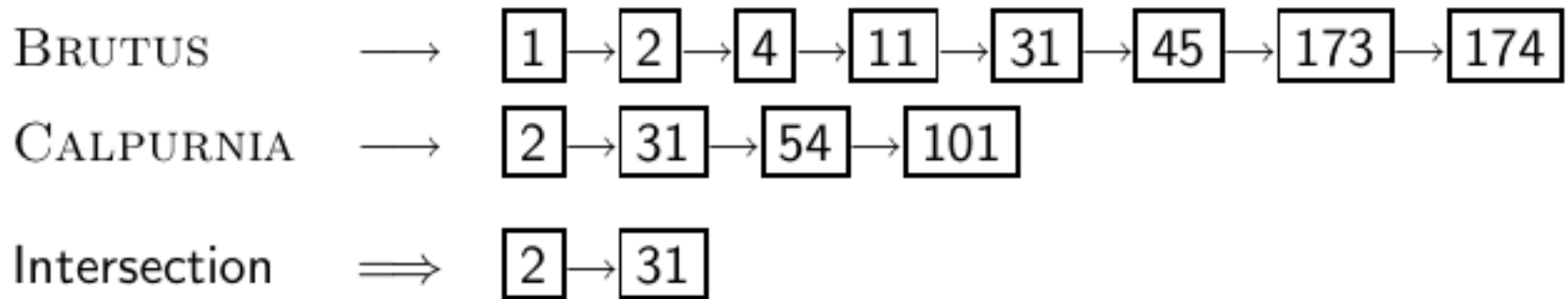
# OUTLINE

- Introduction
- Inverted index
- **Processing Boolean queries**
- Query optimization

# Simple conjunctive query (two terms)

- Consider the query: BRUTUS AND CALPURNIA
- To find all matching documents using inverted index:
  1. Locate BRUTUS in the dictionary
  2. Retrieve its postings list from the postings file
  3. Locate CALPURNIA in the dictionary
  4. Retrieve its postings list from the postings file
  5. Intersect the two postings lists
  6. Return intersection to user

# Intersecting two posting lists



- This is linear in the length of the postings lists.
- Note: This only works if postings lists are sorted.

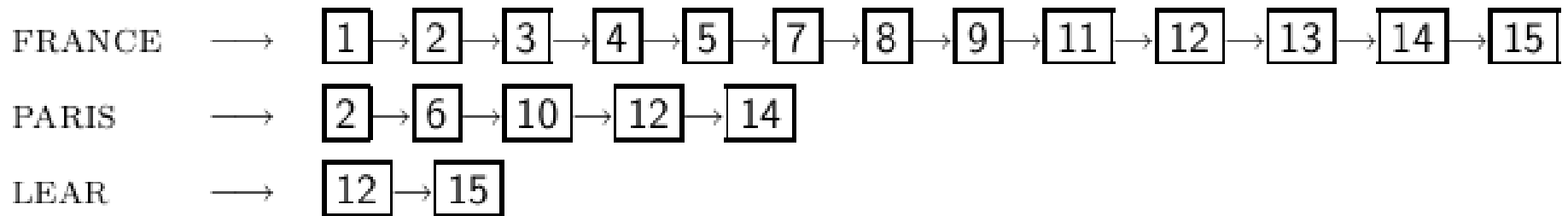


# Intersecting two posting lists

INTERSECT( $p_1, p_2$ )

```
1  answer  $\leftarrow \langle \rangle$ 
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3  do if  $\text{docID}(p_1) = \text{docID}(p_2)$ 
4      then ADD(answer,  $\text{docID}(p_1)$ )
5           $p_1 \leftarrow \text{next}(p_1)$ 
6           $p_2 \leftarrow \text{next}(p_2)$ 
7      else if  $\text{docID}(p_1) < \text{docID}(p_2)$ 
8          then  $p_1 \leftarrow \text{next}(p_1)$ 
9          else  $p_2 \leftarrow \text{next}(p_2)$ 
10 return answer
```

# Quiz: Query processing



Compute hit list for ((paris AND NOT france) OR lear)

# Boolean queries

- The Boolean retrieval model can answer any query that is a Boolean expression.
  - Boolean queries are queries that use AND, OR and NOT to join query terms.
  - Views each document as a **set** of terms.
  - Is precise: Document matches condition or not.
- Primary commercial retrieval tool for 3 decades
- Many professional searchers (e.g., lawyers) still like Boolean queries.
  - You know exactly what you are getting.
- Many search systems you use are also Boolean: spotlight, email, intranet etc.

# Commercially successful Boolean retrieval: Westlaw

- Largest commercial legal search service in terms of the number of paying subscribers
- Over half a million subscribers performing millions of searches a day over tens of terabytes of text data
- The service was started in 1975.
- In 2005, Boolean search (called “Terms and Connectors” by Westlaw) was still the default, and used by a large percentage of users . . .
- . . . although ranked retrieval has been available since 1992.

# Westlaw: Example queries

*Information need:* Information on the legal theories involved in preventing the disclosure of trade secrets by employees formerly employed by a competing company

*Query:* “trade secret” /s disclos! /s prevent /s employe!

*Information need:* Requirements for disabled people to be able to access a workplace

*Query:* disab! /p access! /s work-site work-place (employment /3 place)

*Information need:* Cases about a host’s responsibility for drunk guests

*Query:* host! /p (responsib! liab!) /p (intoxicat! drunk!) /p guest

# Westlaw: Comments

- Proximity operators: /3 = within 3 words, /s = within a sentence, /p = within a paragraph
- Space is disjunction, not conjunction! (This was the default in search pre-Google.)
- Long, precise queries: incrementally developed, not like web search
- Why professional searchers often like Boolean search: precision, transparency, control
- When are Boolean queries the best way of searching? Depends on: information need, searcher, document collection, . . .

# OUTLINE

- Introduction
- Inverted index
- Processing Boolean queries
- Query optimization

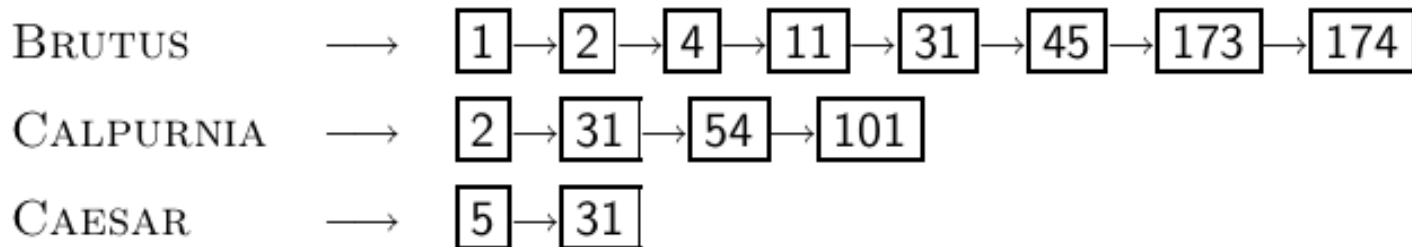
# Query optimization

- Consider a query that is an **and** of  $n$  terms,  $n > 2$
- For each of the terms, get its postings list, then **and** them together
- Example query: BRUTUS AND CALPURNIA AND CAESAR
- What is the best order for processing this query?



# Query optimization

- Example query: BRUTUS AND CALPURNIA AND CAESAR
- Simple and effective optimization: **Process in order of increasing frequency**
- Start with the shortest postings list, then keep cutting further
- In this example, first CAESAR, then CALPURNIA, then BRUTUS



# Optimized intersection algorithm for conjunctive queries

```
INTERSECT( $\langle t_1, \dots, t_n \rangle$ )  
1  terms  $\leftarrow$  SORTBYINCREASINGFREQUENCY( $\langle t_1, \dots, t_n \rangle$ )  
2  result  $\leftarrow$  postings(first(terms))  
3  terms  $\leftarrow$  rest(terms)  
4  while terms  $\neq$  NIL and result  $\neq$  NIL  
5  do result  $\leftarrow$  INTERSECT(result, postings(first(terms)))  
6    terms  $\leftarrow$  rest(terms)  
7  return result
```

# More general optimization

- Example query: (MADDING OR CROWD) and (IGNOBLE OR STRIFE)
- Get frequencies for all terms
- Estimate the size of each or by the sum of its frequencies (conservative)
- Process in increasing order of or sizes

# QUIZ: QUERY PROCESSING ORDER

- Recommend a query processing order for

*(tangerine OR trees) AND  
(marmalade OR skies) AND  
(kaleidoscope OR eyes)*

Term	Freq
eyes	213312
kaleidoscope	87009
marmalade	107913
skies	271658
tangerine	46653
trees	316812



# TERM VOCABULARY & POSTING LISTS

53

# OUTLINE

- Documents
- Terms
  - General + Non-English
  - English
- Skip pointers
- Phrase queries

# Documents

- Previously: Simple Boolean retrieval system
- Our assumptions were:
  - We know what a document is.
  - We can “machine-read” each document.
- This can be complex in reality.

# Parsing a document

- We need to deal with format and language of each document.
- What format is it in? pdf, word, excel, html etc.
- What language is it in?
- What character set is in use?
- Each of these is a classification problem, which we will study later in this course.
- Alternative: use heuristics
  - machine learning vs. heuristics:
  - Data driven vs. human experience



# Format/Language: Complications

- A single index usually contains terms of several languages.
  - Sometimes a document or its components contain multiple languages/formats.
  - French email with Spanish pdf attachment
- What is the document unit for indexing?
  - A file?
  - An email?
  - An email with 5 attachments?
  - A group of files (ppt or latex in HTML)?
- Bottomline: Answering the question “what is a document?” is not trivial and requires some design decisions.
- Also: XML

# OUTLINE

- Documents
- Terms
  - General + Non-English
  - English
- Skip pointers
- Phrase queries

# Definitions

- **Word** – A delimited string of characters as it appears in the text.
- **Term** – A “normalized” word (case, morphology, spelling etc); an equivalence class of words.
- **Token** – An instance of a word or term occurring in a document.
- **Type** – The same as a term in most cases: an equivalence class of tokens.

# Normalization

- Need to “normalize” terms in indexed text as well as query terms into the same form.
- Example: We want to match *U.S.A.* and *USA*
- We most commonly implicitly define **equivalence classes** of terms.
- Alternatively: do asymmetric expansion
  - window → window, windows
  - windows → Windows, windows
  - Windows (no expansion)
- More powerful, but less efficient
- Why don't you want to put *window*, *Window*, *windows*, and *Windows* in the same equivalence class?
- Because they have different meanings

# Normalization: Other languages

- Normalization and language detection interact.
- *PETER WILL NICHT MIT.* → MIT = mit
- *He got his PhD from MIT.* → MIT ≠ mit

# Recall: Inverted index construction

- Input:

Friends, Romans, countrymen. So let it be with Caesar . . .

- Output:

friend roman countryman so . . .

- Each token is a candidate for a postings entry.
- What are valid tokens to emit?

# Quiz: Tokens, Words and Terms

*In June, the dog likes to chase the cat in the barn.*

- How many word tokens are there?
- How many terms are there?

# Tokenization problems: 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

- 3/20/91
- 20/3/91
- Mar 20, 1991
- B-52
- 100.2.86.144
- (800) 234-2333
- 800.234.2333
- Older IR systems may not index numbers . . .
- . . . but generally it's a useful feature.

# 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

The two characters can be treated as one word meaning 'monk' or as a sequence of two words meaning 'and' and 'still'.

# Other cases of “no whitespace”

- Compounds in Dutch, German, Swedish
- Computerlinguistik → Computer + Linguistik
- Lebensversicherungsgesellschaftsangestellter  
→ leben + versicherung + gesellschaft + angestellter  
(life insurance company employee)
- Inuit: tusaatsiarunnangittualuujunga (I can't hear very well.)
- Many other languages with segmentation difficulties: Finnish, Urdu, . . .

# Japanese

ノーベル平和賞を受賞したワンガリ・マータイさんが名誉会長を務めるMOTTAINAIキャンペーンの一環として、毎日新聞社とマガジンハウスは「私の、もったいない」を募集します。皆様が日ごろ「もったいない」と感じて実践していることや、それにまつわるエピソードを800字以内の文章にまとめ、簡単な写真、イラスト、図などを添えて10月20日までにお送りください。大賞受賞者には、50万円相当の旅行券とエコ製品2点の副賞が贈られます。

4 different “alphabets”: Chinese characters, hiragana syllabary for inflectional endings and functional words, katakana syllabary for transcription of foreign words and other uses, and latin. No spaces (as in Chinese). End user can express query entirely in hiragana!

# Arabic script

ك ت ا ب ← كِتَابٌ  
un b ā t i k  
/kitābun/ ‘*a book*’

# Arabic script: Bidirectionality

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.

# Accents and diacritics

- Accents: résumé vs. resume (simple omission of accent)
- Umlauts: Universität vs. Universitaet (substitution with special letter sequence “ae”)
- Most important criterion: How are users likely to write their queries for these words?
- Even in languages that standardly have accents, users often do not type them. (Polish?)



# Case folding

- Reduce all letters to lower case
- Possible exceptions: capitalized words in mid-sentence
- MIT vs. mit
- Fed vs. fed
- It's often best to lowercase everything since users will use lowercase regardless of correct capitalization.

# Stop words

- stop words = extremely common words which would appear to be of little value in helping select documents matching a user need
- Examples: *a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with*
- Stop word elimination used to be standard in older IR systems.
- But you need stop words for phrase queries, e.g. “King of Denmark”
- Most web search engines index stop words.

# More equivalence classing

- Soundex: IIR 3 (phonetic equivalence, Muller = Mueller)
- Thesauri: IIR 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.
  - Example command: Delete final *ement* if what remains is longer than 1 character
  - replacement → replac
  - cement → cement
- Example convention: Of the rules in a compound command, select the one that applies to the longest suffix.

# Porter stemmer: A few rules

## Rule

SSES  $\rightarrow$  SS

IES  $\rightarrow$  I

SS  $\rightarrow$  SS

S  $\rightarrow$

## Example

caresses  $\rightarrow$  caress

ponies  $\rightarrow$  poni

caress  $\rightarrow$  caress

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 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.
- Queries where stemming is likely to help: [tartan sweaters], [sightseeing tour san francisco] (equivalence classes: {sweater,sweaters}, {tour,tours})
- 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]

## QUIZ: LEMMA AND STEM

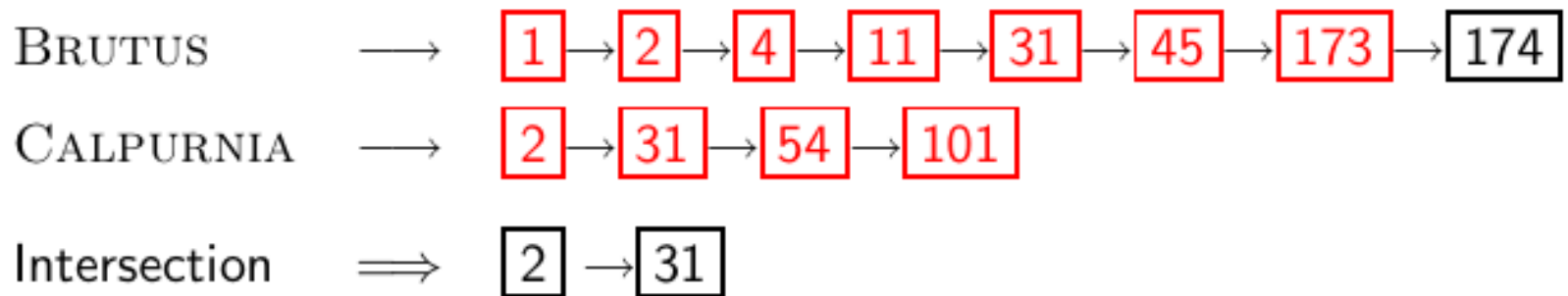
Are the following statements true or false?

1. In a Boolean retrieval system, stemming never lowers precision. (T/F)
2. In a Boolean retrieval system, stemming sometimes lowers recall. (T/F)
3. Stemming increases the size of the vocabulary. (T/F)
4. Stemming is invoked at indexing time but not while processing a query. (T/F)

# OUTLINE

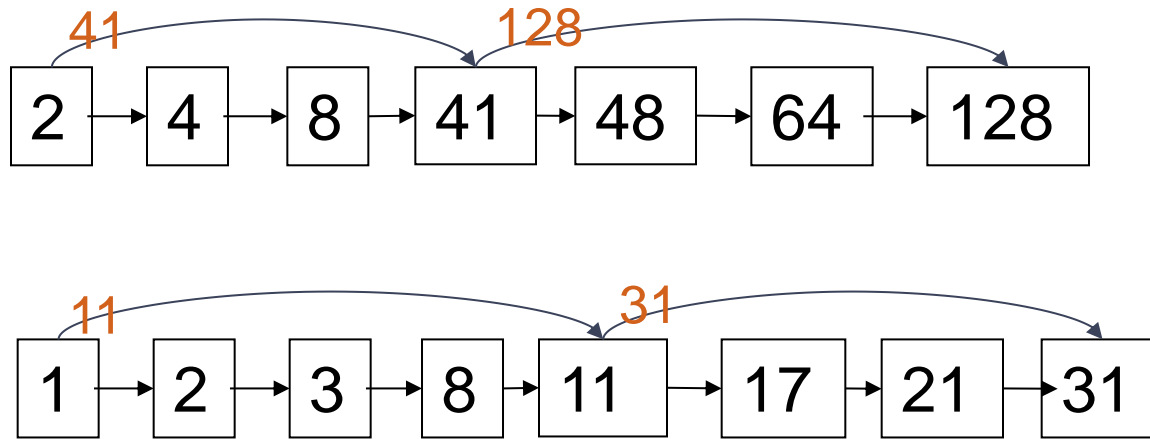
- Documents
- Terms
  - General + Non-English
  - English
- Skip pointers
- Phrase queries

# Recall basic intersection algorithm



- Linear in the length of the postings lists.
- Can we do better?

# AUGMENT POSTINGS WITH SKIP POINTERS (AT INDEXING TIME)



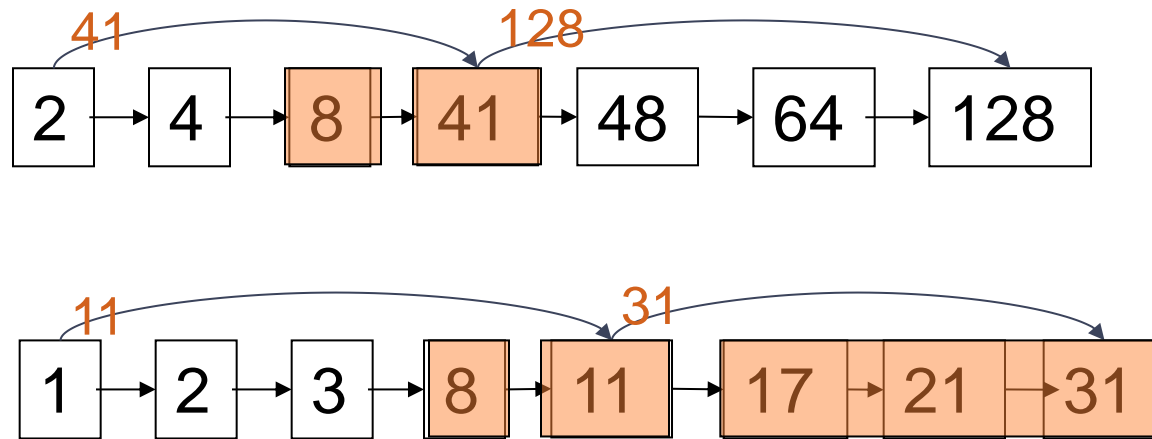
- Why?

To skip postings that will not feature 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.

# Intersection with skip pointers

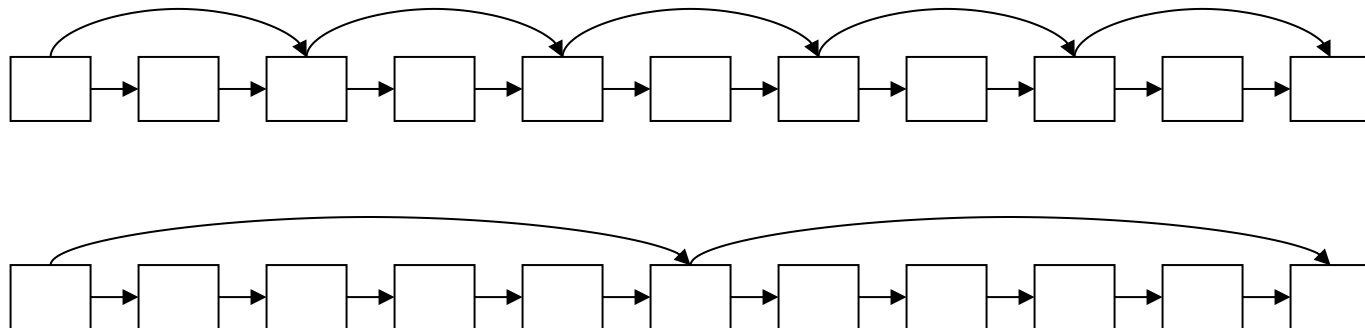
INTERSECTWITHSKIPS( $p_1, p_2$ )

```
1  answer  $\leftarrow \langle \rangle$ 
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3  do if  $\text{docID}(p_1) = \text{docID}(p_2)$ 
4      then  $\text{ADD}(\text{answer}, \text{docID}(p_1))$ 
5           $p_1 \leftarrow \text{next}(p_1)$ 
6           $p_2 \leftarrow \text{next}(p_2)$ 
7  else if  $\text{docID}(p_1) < \text{docID}(p_2)$ 
8      then if  $\text{hasSkip}(p_1)$  and  $(\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2))$ 
9          then while  $\text{hasSkip}(p_1)$  and  $(\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2))$ 
10             do  $p_1 \leftarrow \text{skip}(p_1)$ 
11             else  $p_1 \leftarrow \text{next}(p_1)$ 
12  else if  $\text{hasSkip}(p_2)$  and  $(\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1))$ 
13      then while  $\text{hasSkip}(p_2)$  and  $(\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1))$ 
14          do  $p_2 \leftarrow \text{skip}(p_2)$ 
15          else  $p_2 \leftarrow \text{next}(p_2)$ 
16  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.





# Where do we place skips? (cont)

- Simple heuristic: for postings list of length  $P$ , use  $\sqrt{P}$  evenly-spaced skip pointers.
- This ignores the distribution of query terms.
- Easy if the index is static; harder in a dynamic environment because of updates.
- How much do skip pointers help?
- They used to help a lot.
- With today's fast CPUs, they don't help that much anymore.