Term Vocubular and Posting Lists [1]

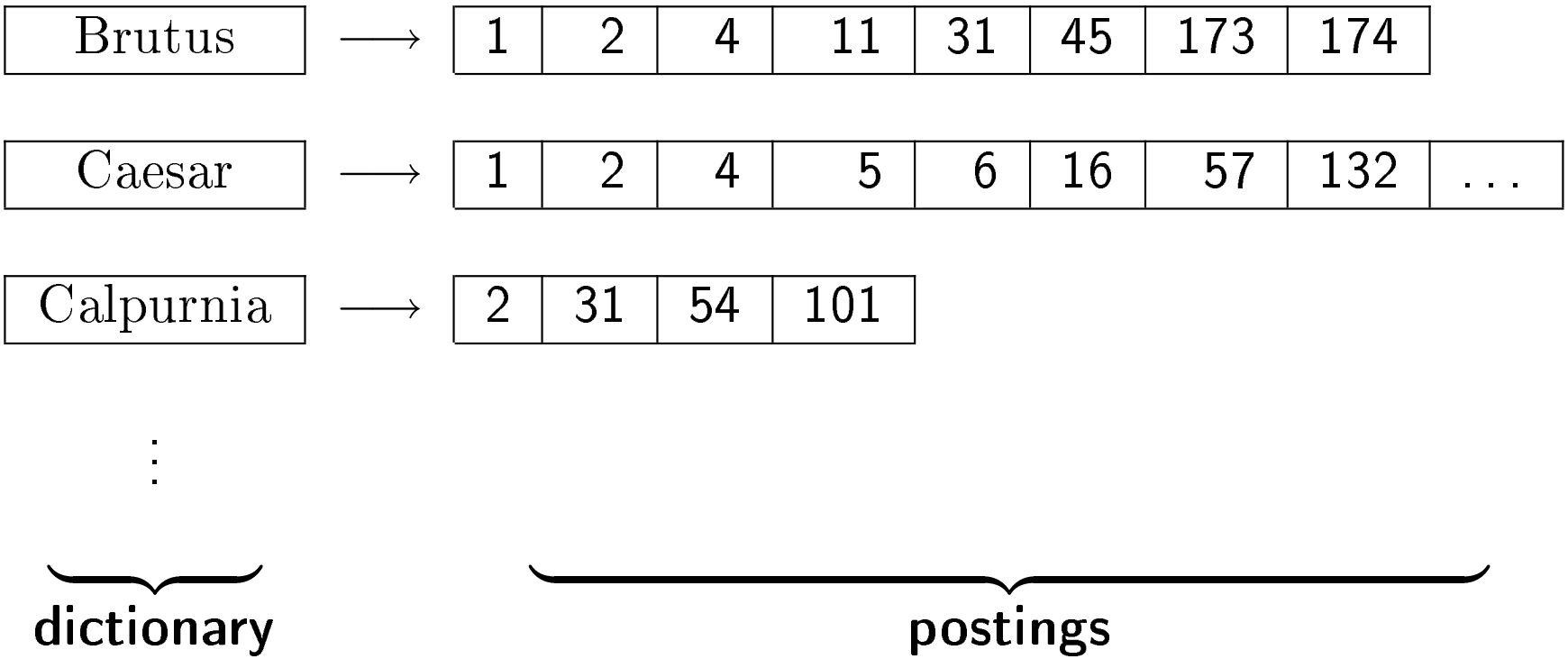
September, 2009

[1] Vorlage: Folien von M. Schütze

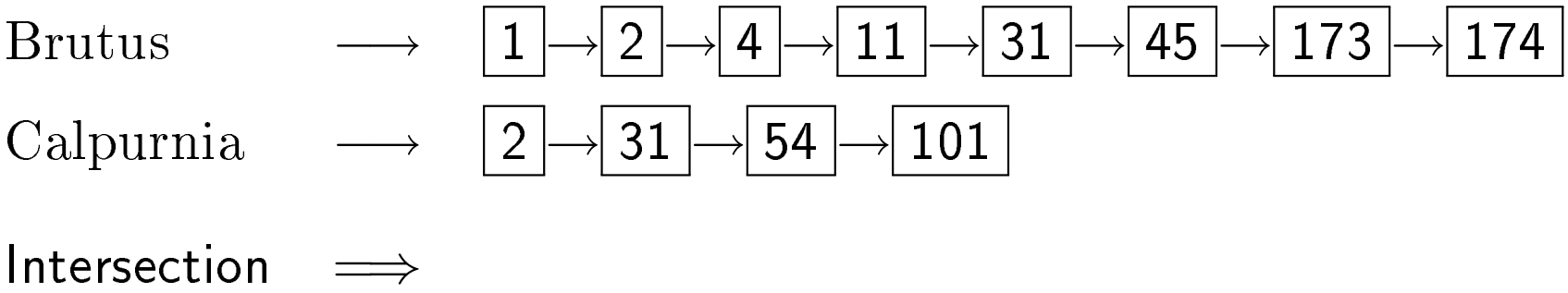
Outline

Inverted index

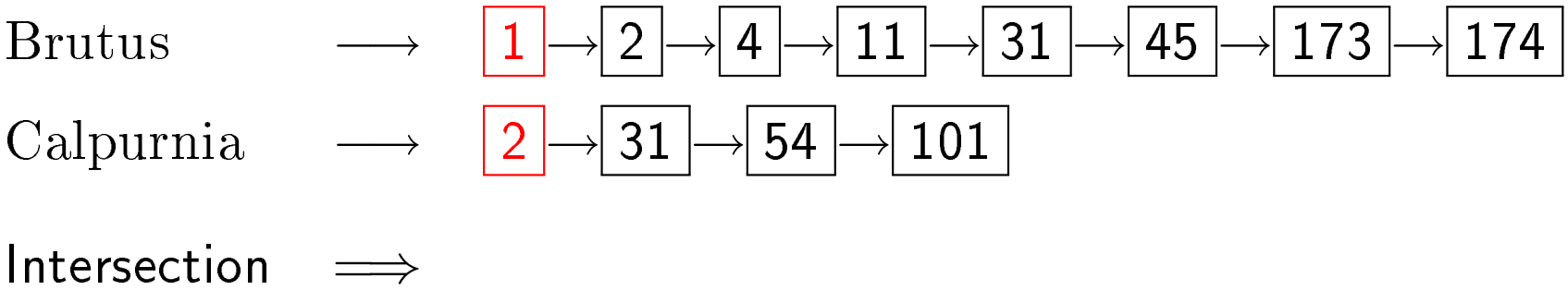
For each term t, we store a list of all documents that contain t.



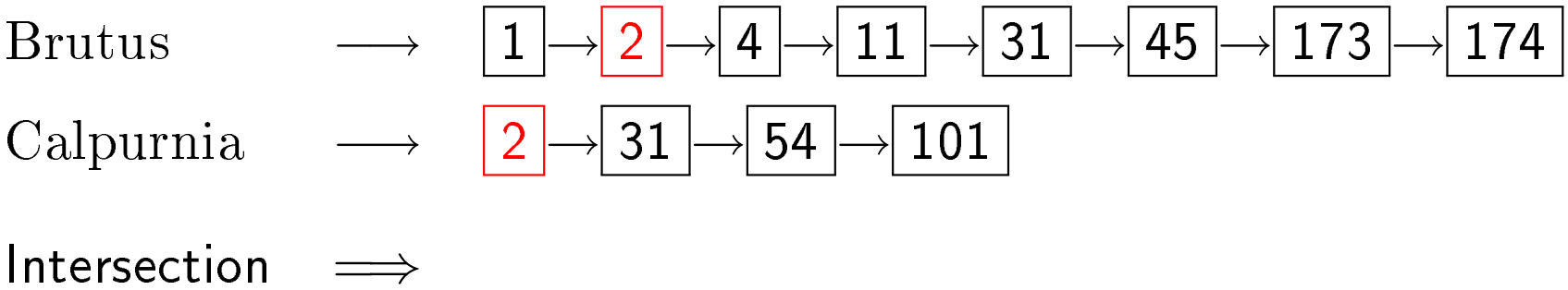
Intersecting two postings lists



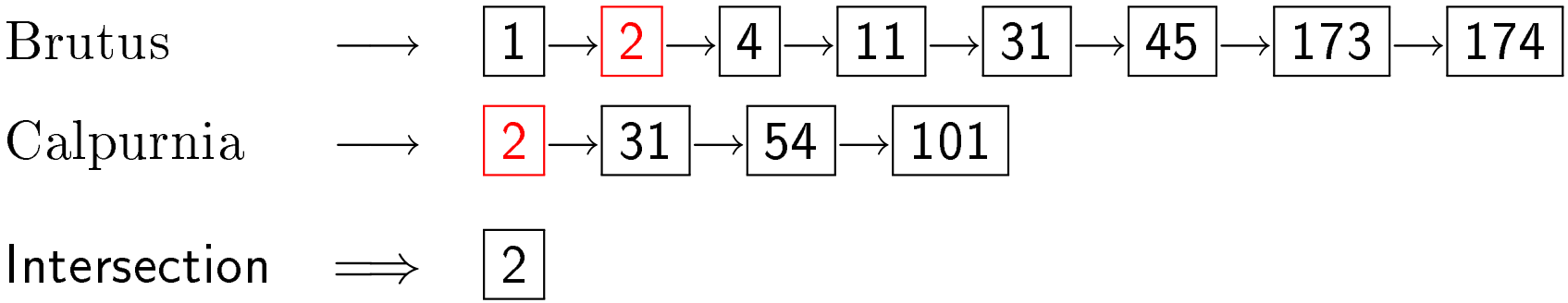
Intersecting two postings lists



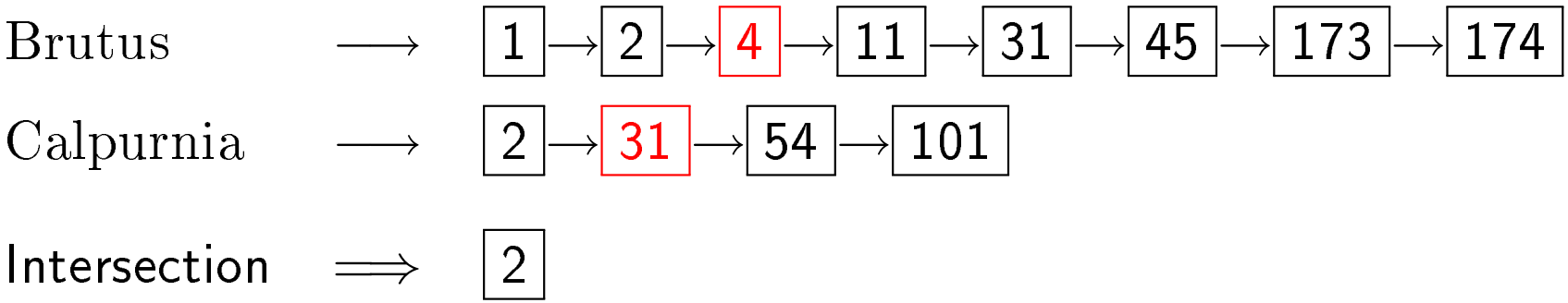
Intersecting two postings lists



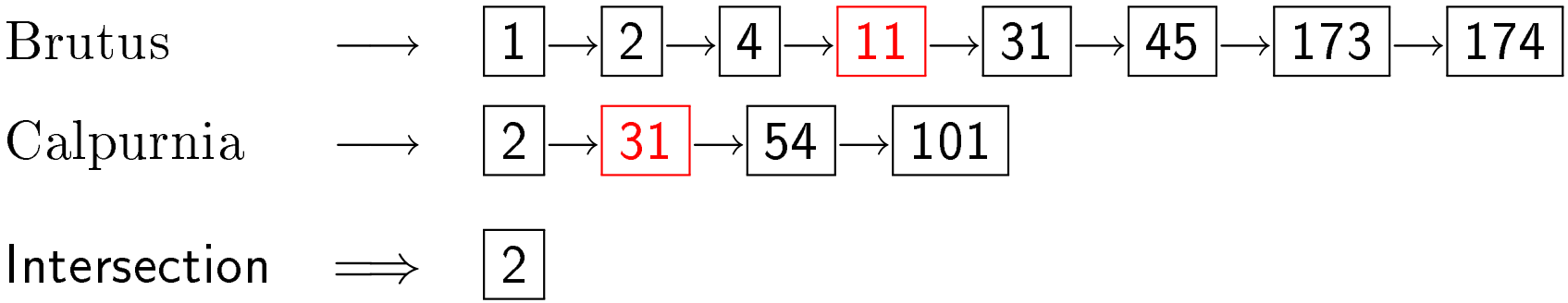
Intersecting two postings lists



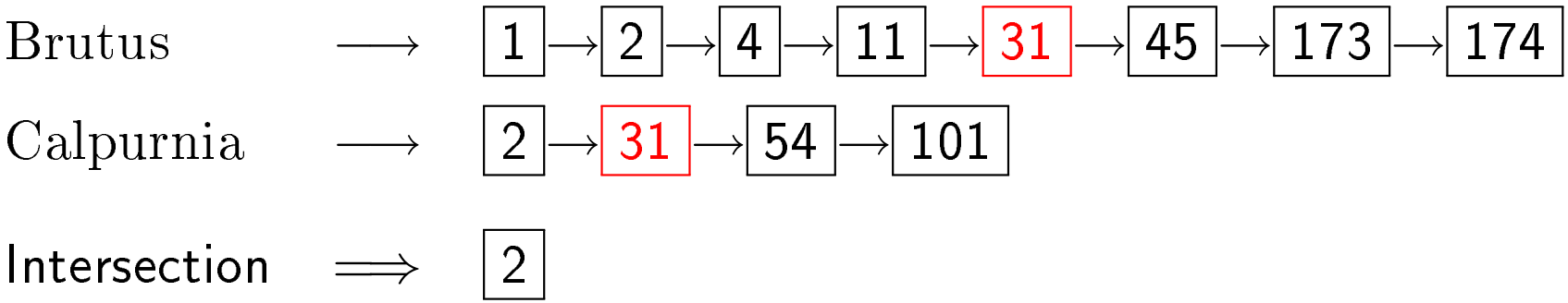
Intersecting two postings lists



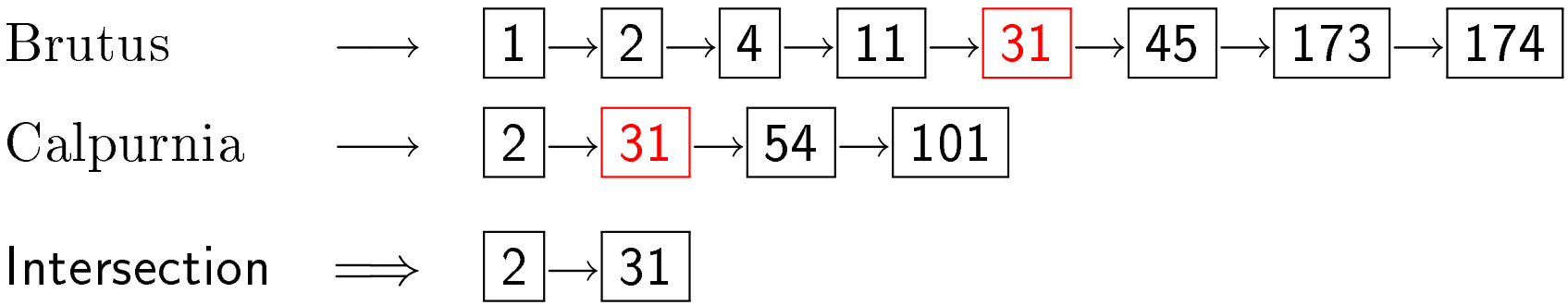
Intersecting two postings lists



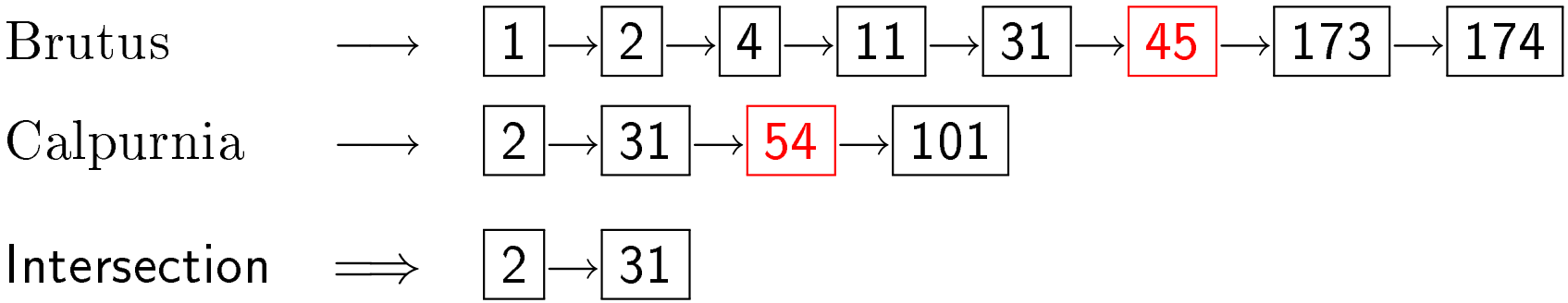
Intersecting two postings lists



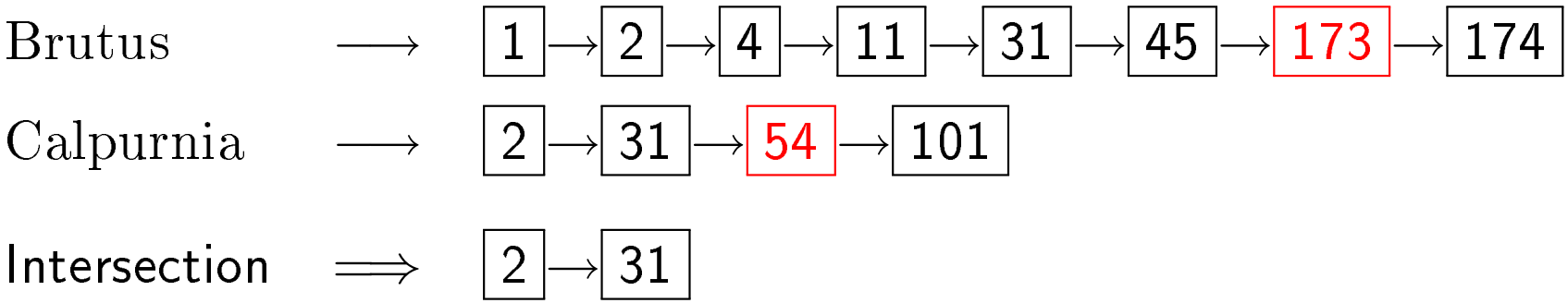
Intersecting two postings lists



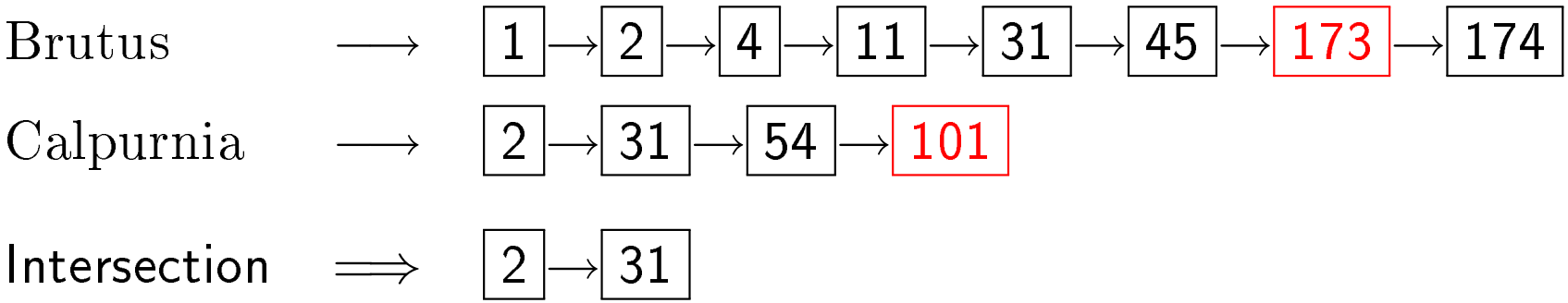
Intersecting two postings lists



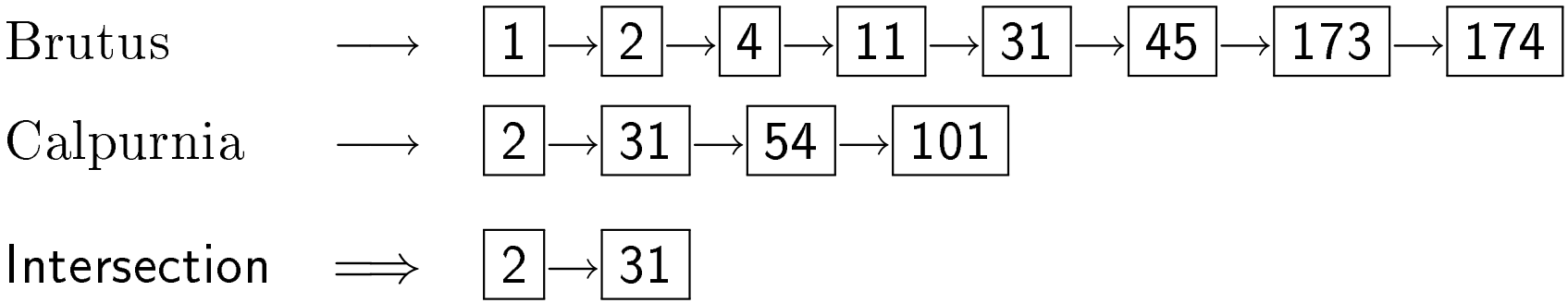
Intersecting two postings lists



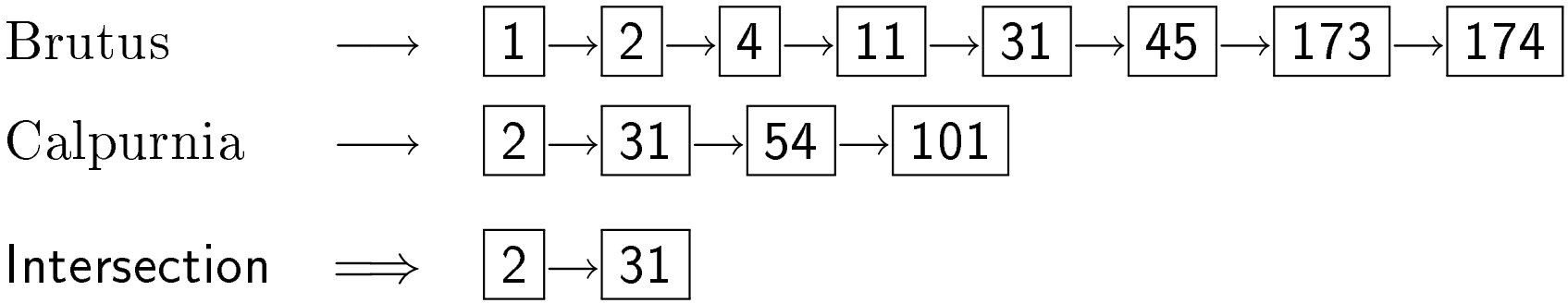
Intersecting two postings lists



Intersecting two postings lists



Intersecting two postings lists



- Linear in the length of the postings lists.

Constructing the inverted index: Sort postings

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

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

Outline

Terms and documents

- Last lecture: Simple Boolean retrieval system

Terms and documents

- Last lecture: Simple Boolean retrieval system
- Our assumptions were:

Terms and documents

- Last lecture: Simple Boolean retrieval system
- Our assumptions were:
  - We know what a document is.

Terms and documents

- Last lecture: Simple Boolean retrieval system
- Our assumptions were:
  - We know what a document is.
  - We know what a term is.

Terms and documents

- Last lecture: Simple Boolean retrieval system
- Our assumptions were:
  - We know what a document is.
  - We know what a term is.
- Both issues can be complex in reality.

Terms and documents

- Last lecture: Simple Boolean retrieval system
- Our assumptions were:
  - We know what a document is.
  - We know what a term is.
- Both issues can be complex in reality.
- We'll look a little bit at what a document is.

Terms and documents

- Last lecture: Simple Boolean retrieval system
- Our assumptions were:
  - We know what a document is.
  - We know what a term is.
- Both issues can be complex in reality.
- We'll look a little bit at what a document is.
- But mostly at terms: How do we define and process the vocabulary of terms of a collection?

Parsing a document

- Before we can even start worrying about terms ...

Parsing a document

- Before we can even start worrying about terms . . .
- . . . need to deal with format and language of each document.

Parsing a document

- Before we can even start worrying about terms . . .
- . . . need to deal with format and language of each document.
- What format is it in? pdf, word, excel, html etc.

Parsing a document

- Before we can even start worrying about terms . . .
- . . . 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?

Parsing a document

- Before we can even start worrying about terms . . .
- . . . 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?

Parsing a document

- Before we can even start worrying about terms . . .
- . . . 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 (IIR 13).

Parsing a document

- Before we can even start worrying about terms . . .
- . . . 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 (IIR 13).
- Alternative: use heuristics

Format/Language: Complications

- A single index usually contains terms of several languages.

Format/Language: Complications

- A single index usually contains terms of several languages.
- Sometimes a document or its components contain multiple languages/formats.

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

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?

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?

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?

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?

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)?

Terms

Definitions

- Word – A delimited string of characters as it appears in the text.

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.

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.

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.

Type/token distinction: Example

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

Type/token distinction: Example

- In June, the dog likes to chase the cat in the barn.
- How many tokens? How many types?

Recall: Inverted index construction

- Input:

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

  ...

Recall: Inverted index construction

- Input:

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

  ...
- Output:

  friend | roman | countryman | so ...

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.

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?

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

One word or two? (or several)

- Hewlett-Packard
- State-of-the-art

One word or two? (or several)

- Hewlett-Packard
- State-of-the-art
- co-education

One word or two? (or several)

- Hewlett-Packard
- State-of-the-art
- co-education
- the hold-him-back-and-drag-him-away maneuver

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

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

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

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

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

Numbers

- 3/12/91
- 12/3/91

Numbers

- 3/12/91
- 12/3/91
- Mar 12, 1991

Numbers

- 3/12/91
- 12/3/91
- Mar 12, 1991
- B-52

Numbers

- 3/12/91
- 12/3/91
- Mar 12, 1991
- B-52
- 100.2.86.144

Numbers

- 3/12/91
- 12/3/91
- Mar 12, 1991
- B-52
- 100.2.86.144
- (800) 234-2333

Numbers

- 3/12/91
- 12/3/91
- Mar 12, 1991
- B-52
- 100.2.86.144
- (800) 234-2333
- 800.234.2333

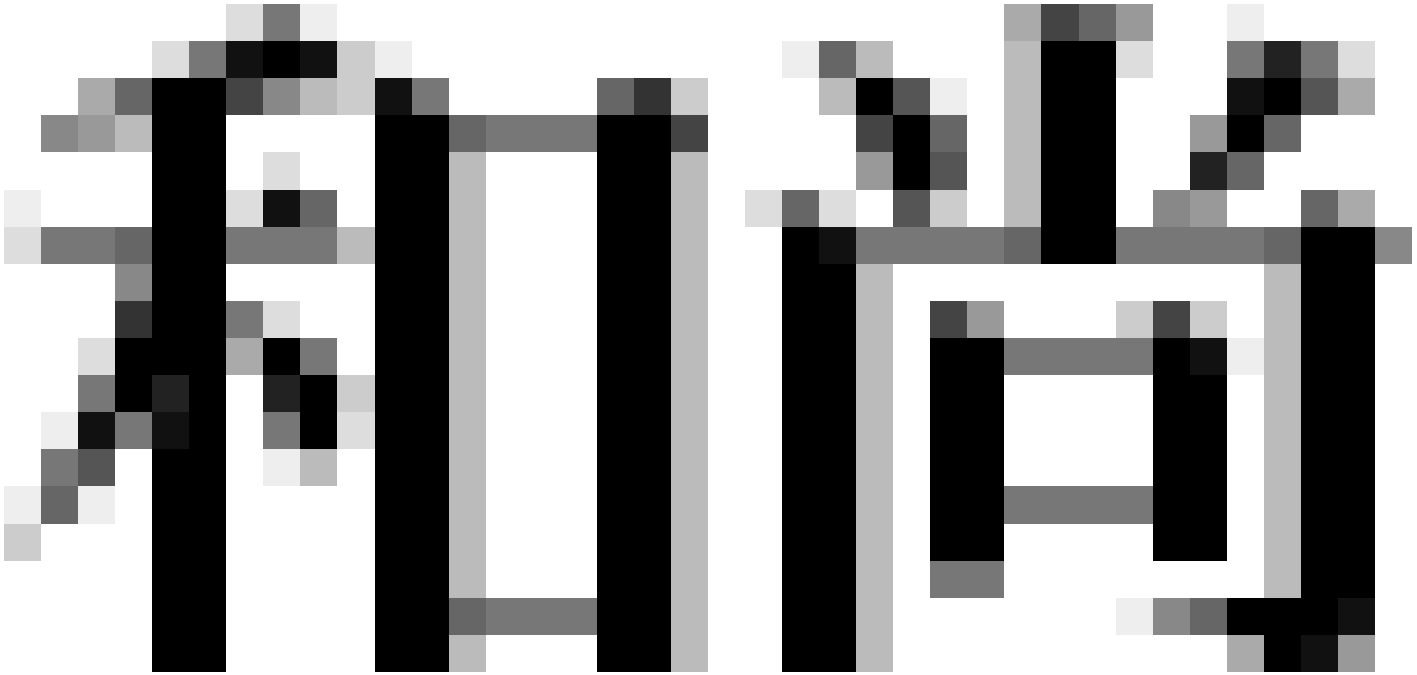
Numbers

- 3/12/91
- 12/3/91
- Mar 12, 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



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 and German

Other cases of "no whitespace"

- Compounds in Dutch and German
- Computerlinguistik → Computer + Linguistik

Other cases of "no whitespace"

- Compounds in Dutch and German
- Computerlinguistik → Computer + Linguistik
- Lebensversicherungsgesellschaftsangestellter

Other cases of "no whitespace"

- Compounds in Dutch and German
- Computerlinguistik → Computer + Linguistik
- Lebensversicherungsgesellschaftsangestellter
- → leben + versicherung + gesellschaft + angestellter

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.)

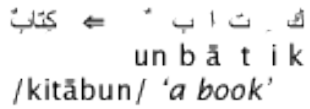
Other cases of "no whitespace"

- 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

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。

Arabic script



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.'

Back to English

Normalization

- Need to “normalize” terms in indexed text as well as query terms into the same form.

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

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.

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

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

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

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)

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

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?

Normalization: Other languages

- Accents: résumé vs. resume (simple omission of accent)

Normalization: Other languages

- Accents: résumé vs. resume (simple omission of accent)
- Umlauts: Universität vs. Universitaet (substitution with special letter sequence "ae")

Normalization: Other languages

- 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?

Normalization: Other languages

- 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?)

Normalization: Other languages

- 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?)
- Normalization and language detection interact.

Normalization: Other languages

- 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?)
- Normalization and language detection interact.
- PETER WILL NICHT MIT. $\rightarrow$ MIT $=$ mit

Normalization: Other languages

- 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?)
- Normalization and language detection interact.
- PETER WILL NICHT MIT. $\rightarrow$ MIT $=$ mit
- He got his PhD from MIT. $\rightarrow$ MIT $\neq$ mit

Case folding

- Reduce all letters to lower case

Case folding

- Reduce all letters to lower case
- Possible exceptions: capitalized words in mid-sentence

Case folding

- Reduce all letters to lower case
- Possible exceptions: capitalized words in mid-sentence
- MIT vs. mit

Case folding

- Reduce all letters to lower case
- Possible exceptions: capitalized words in mid-sentence
- MIT vs. mit
- Fed vs. fed

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

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 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.

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"

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, Tchebyshev = Chebysheff)

More equivalence classing

- Soundex: IIR 3 (phonetic equivalence, Tchebyshev = Chebysheff)
- Thesauri: IIR 9 (semantic equivalence, car = automobile)

What does Google do?

- Stop words
- Normalization
- Tokenization
- Lowercasing
- Stemming
- Non-latin alphabets
- Umlauts
- Compounds
- Numbers

Lemmatization

- Reduce inflectional/variant forms to base form

Lemmatization

- Reduce inflectional/variant forms to base form
- Example: am, are, is $\longrightarrow$ be

Lemmatization

- Reduce inflectional/variant forms to base form
- Example: am, are, is $\longrightarrow$ be
- Example: car, cars, car's, cars' $\longrightarrow$ car

Lemmatization

- Reduce inflectional/variant forms to base form
- Example: am, are, is $\longrightarrow$ be
- Example: car, cars, car's, cars' $\longrightarrow$ car
- Example: the boy's cars are different colors $\longrightarrow$ the boy car be different color

Lemmatization

- Reduce inflectional/variant forms to base form
- Example: am, are, is $\longrightarrow$ be
- Example: car, cars, car's, cars' $\longrightarrow$ car
- Example: the boy's cars are different colors $\longrightarrow$ the boy car be different color
- Lemmatization implies doing "proper" reduction to dictionary headword form (the lemma).

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.

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

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

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

Porter algorithm

- Most common algorithm for stemming English
- Results suggest that it is at least as good as other stemming options

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

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

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.

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

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

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

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

Does stemming improve effectiveness?

- In general, stemming increases effectiveness for some queries, and decreases effectiveness for others.

Does stemming improve effectiveness?

- In general, stemming increases effectiveness for some queries, and decreases effectiveness for others.
- Porter Stemmer equivalence class oper contains all of operate operating operates operation operative operatives operational.

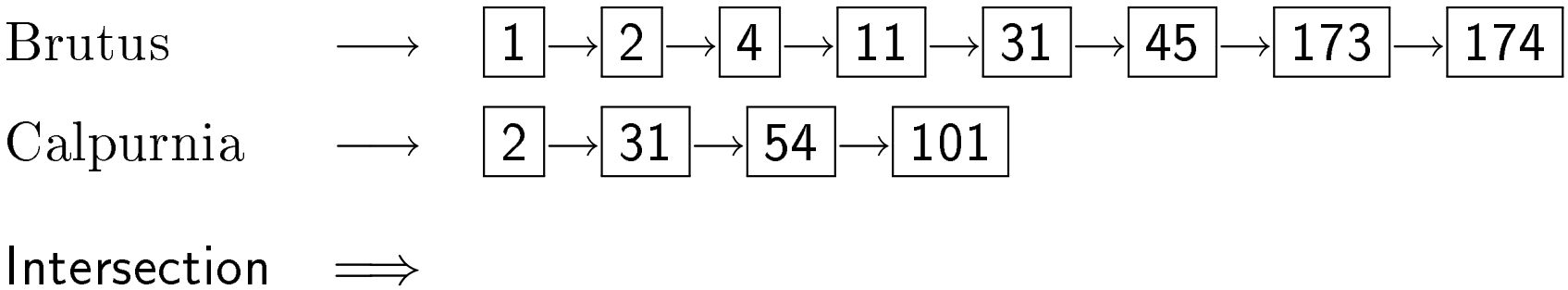
Does stemming improve effectiveness?

- In general, stemming increases effectiveness for some queries, and decreases effectiveness for others.
- 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”

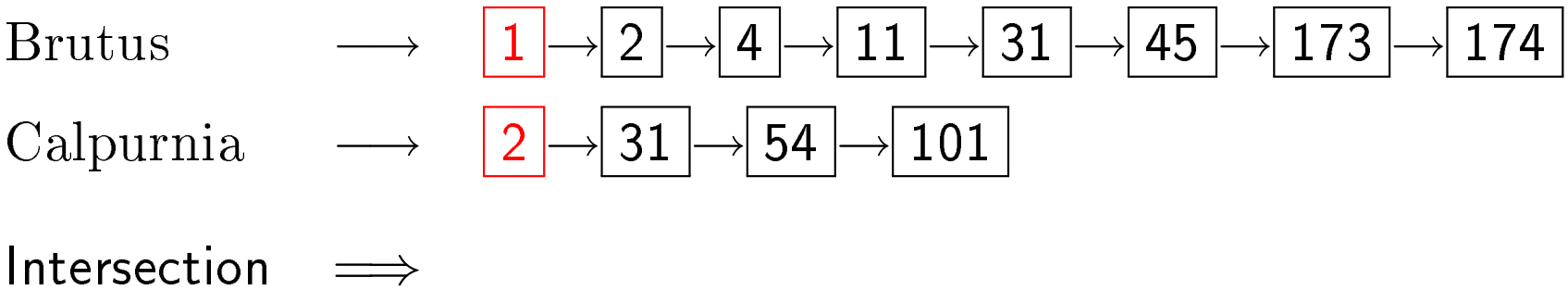
Interesting issues in your native language?

Outline

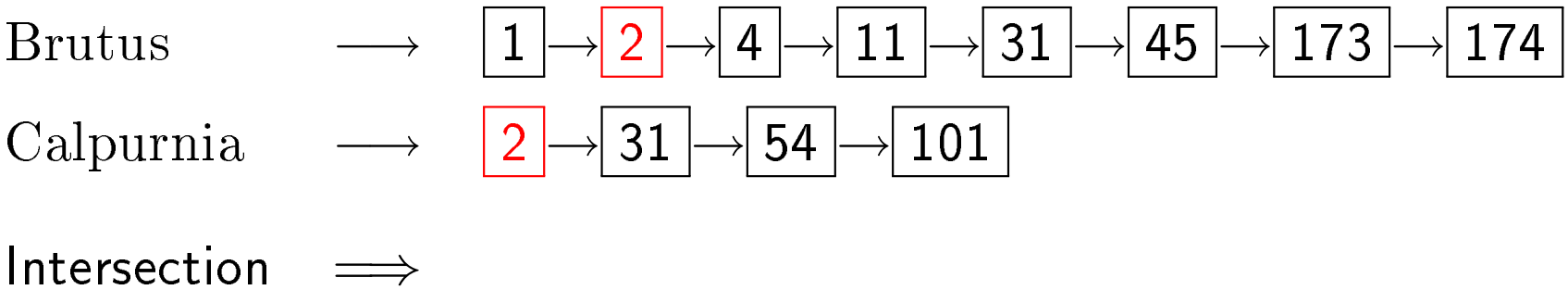
Recall basic intersection algorithm



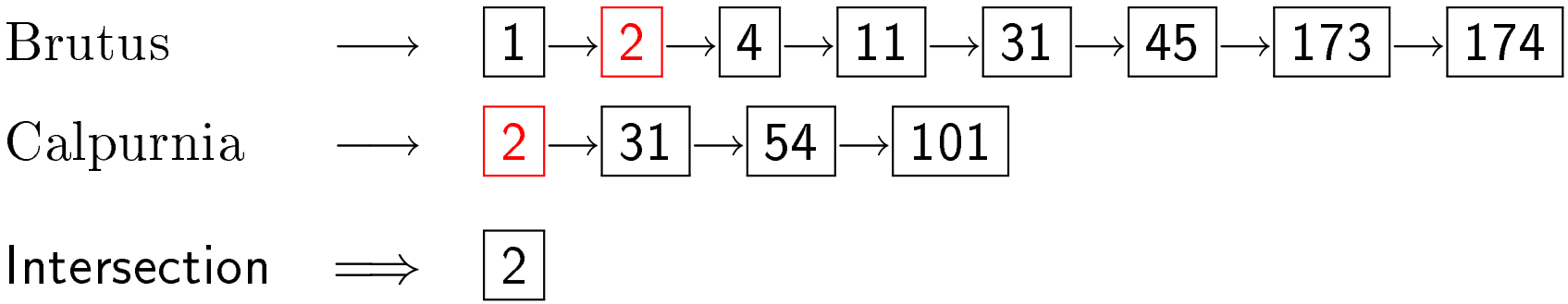
Recall basic intersection algorithm



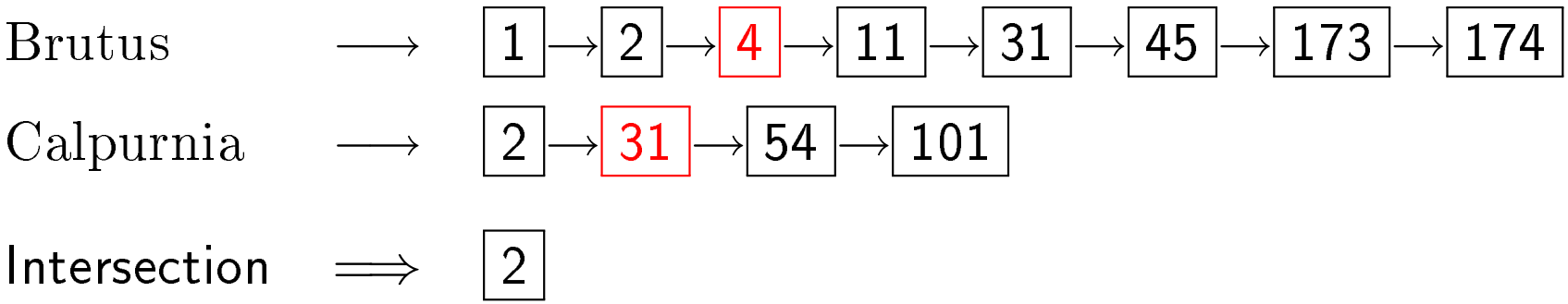
Recall basic intersection algorithm



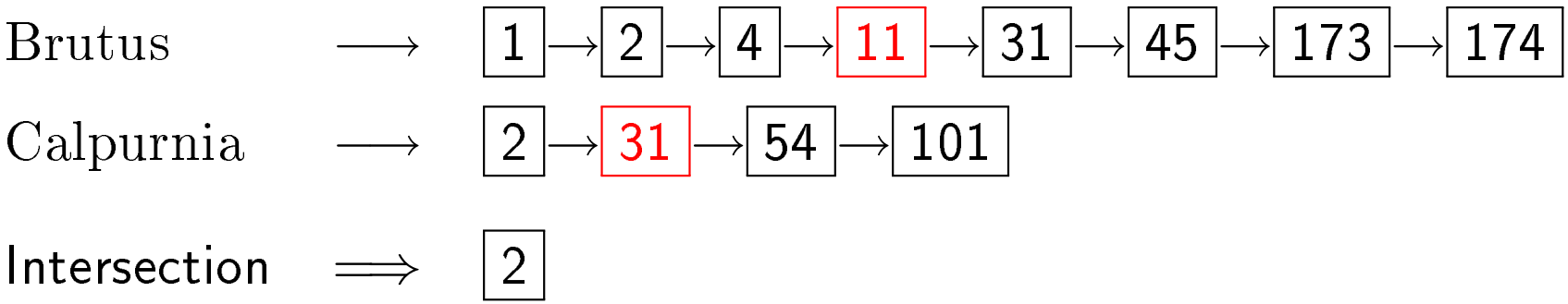
Recall basic intersection algorithm



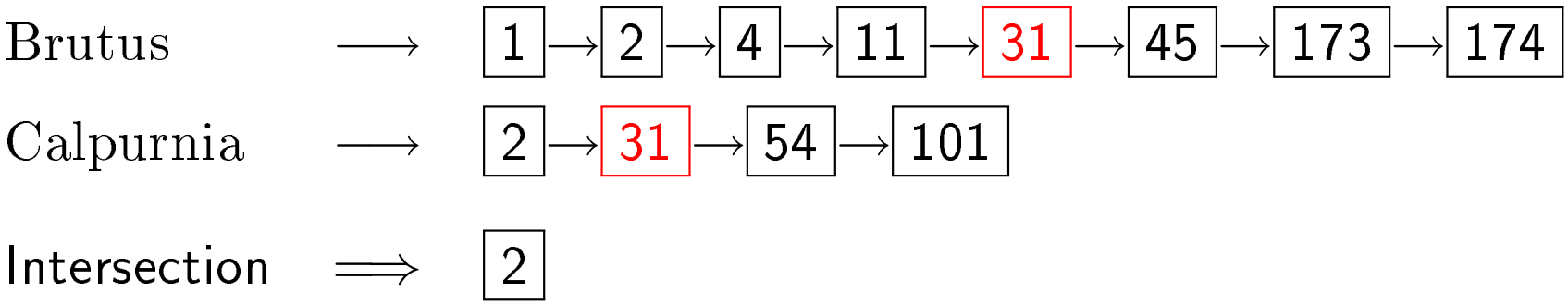
Recall basic intersection algorithm



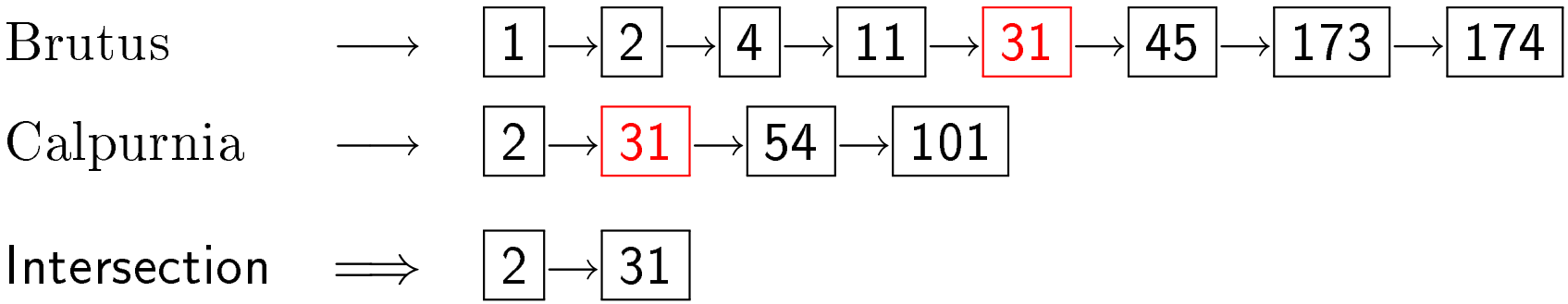
Recall basic intersection algorithm



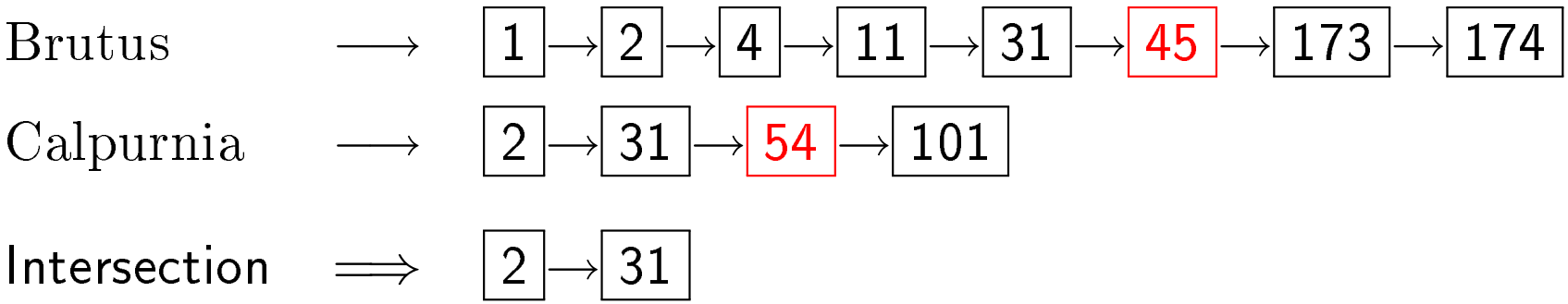
Recall basic intersection algorithm



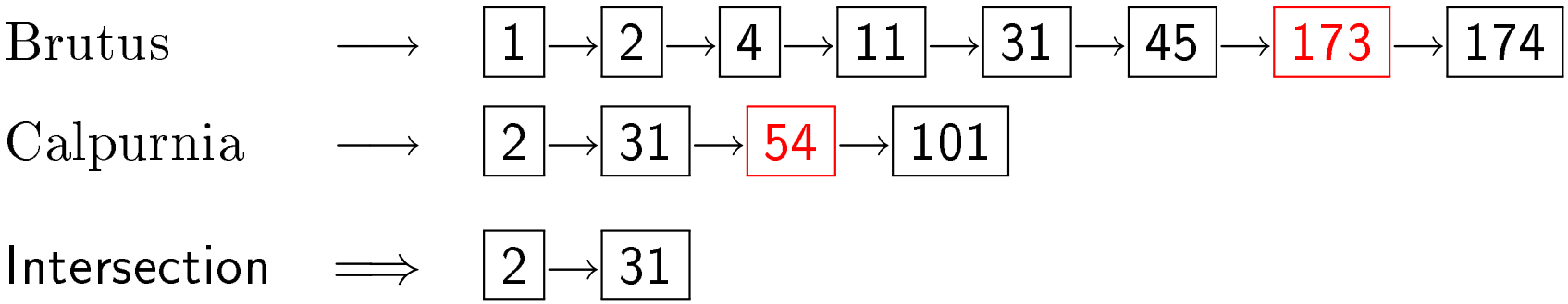
Recall basic intersection algorithm



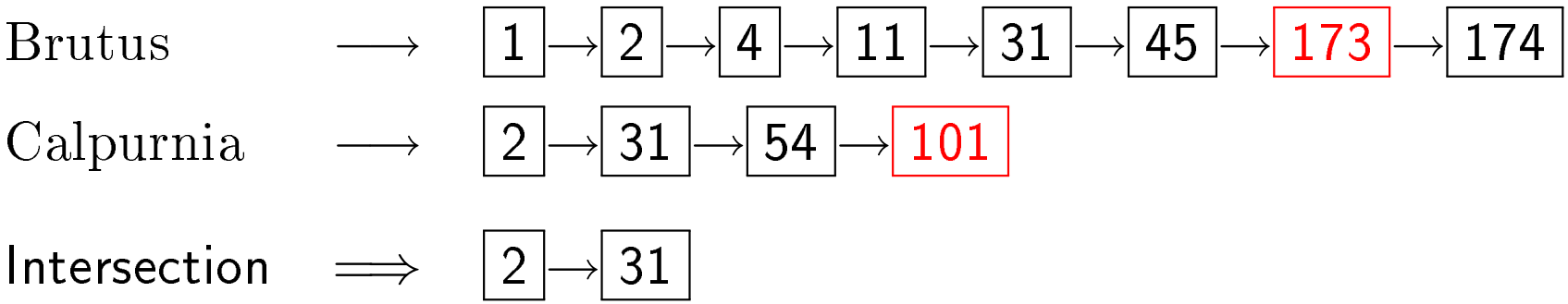
Recall basic intersection algorithm



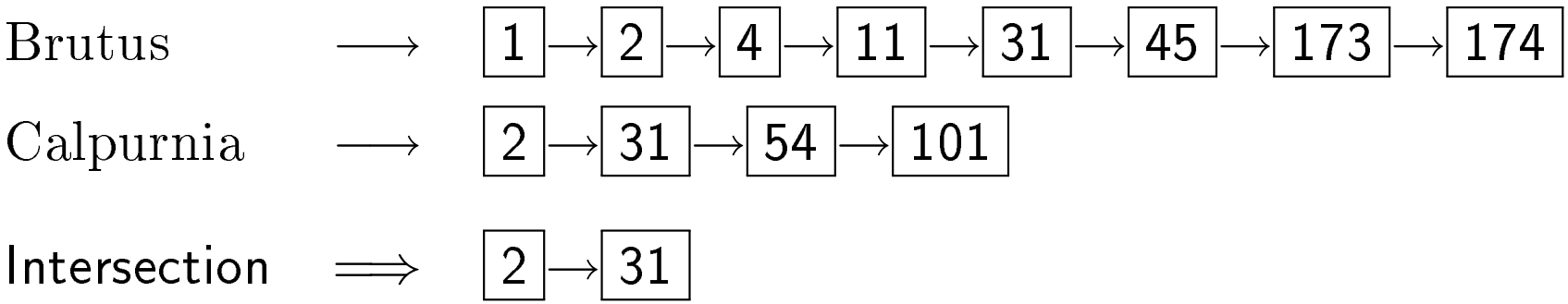
Recall basic intersection algorithm



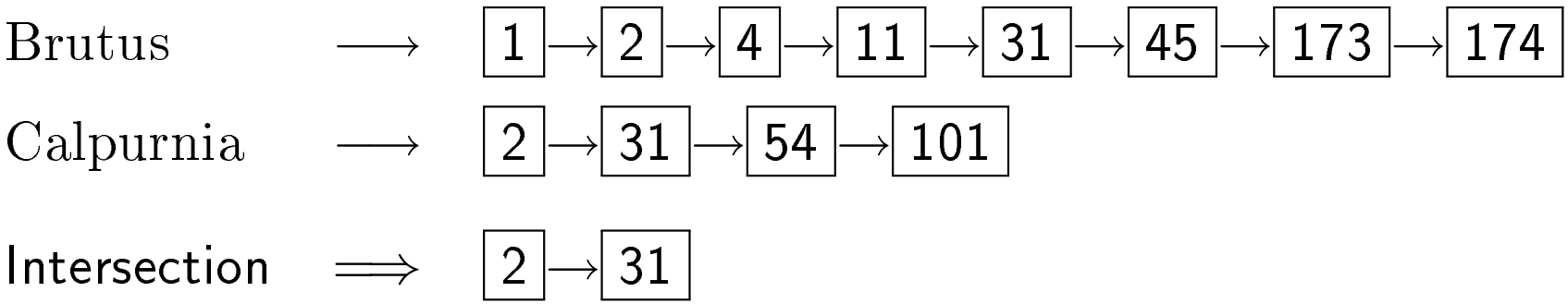
Recall basic intersection algorithm



Recall basic intersection algorithm



Recall basic intersection algorithm



- Can we do better?

Skip pointers

- Skip pointers allow us to skip postings that will not figure in the search results.

Skip pointers

- Skip pointers allow us to skip postings that will not figure in the search results.
- This makes intersecting postings lists more efficient.

Skip pointers

- Skip pointers allow us to skip postings that will not figure in the search results.
- This makes intersecting postings lists more efficient.
- Some postings lists contain several million entries – so efficiency can be an issue even if though basic intersection is linear.

Skip pointers

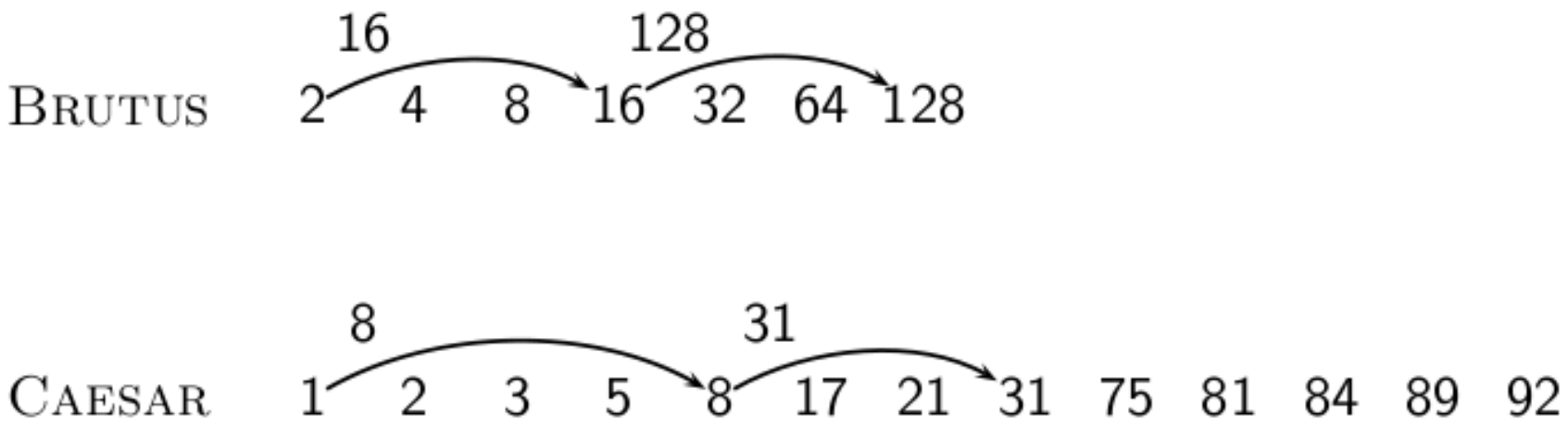
- Skip pointers allow us to skip postings that will not figure in the search results.
- This makes intersecting postings lists more efficient.
- Some postings lists contain several million entries – so efficiency can be an issue even if though basic intersection is linear.
- Where do we put skip pointers?

Skip pointers

- Skip pointers allow us to skip postings that will not figure in the search results.
- This makes intersecting postings lists more efficient.
- Some postings lists contain several million entries – so efficiency can be an issue even if though basic intersection is linear.
- Where do we put skip pointers?
- How do we make sure results don't change?

Skip lists

Basic idea


16
128
BRUTUS
2 4 8 16 32 64 128
8
31
CAESAR
1 2 3 5 8 17 21 31 75 81 84 89 92

Intersecting with skip pointers

```
IntersectWithSkips(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then Add(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then if hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
 9                     then p1 ← skip(p1)
10                     else p1 ← next(p1)
11                else if hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
12                     then p2 ← skip(p2)
13                     else p2 ← next(p2)
14  return answer
```

Where do we place skips?

- Tradeoff: number of items skipped vs. frequency skip can be taken

Where do we place skips?

- Tradeoff: number of items skipped vs. frequency skip can be taken
- More skips: Each skip pointer skips only a few items, but we can frequently use it.

Where do we place skips?

- Tradeoff: number of items skipped vs. frequency skip can be taken
- More skips: Each skip pointer skips only a few items, but we can frequently use it.
- Fewer skips: Each skip pointer skips many items, but we can not use it very often.

Where do we place skips? (cont)

- Simple heuristic: for postings list of length P, use $\sqrt{P}$ evenly-spaced skip pointers.

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.

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 relatively static; harder in a dynamic environment because of updates.

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 relatively static; harder in a dynamic environment because of updates.
- How much do skip pointers help?

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 relatively static; harder in a dynamic environment because of updates.
- How much do skip pointers help?
- They used to help lot.

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 relatively static; harder in a dynamic environment because of updates.
- How much do skip pointers help?
- They used to help lot.
- With today's fast CPUs, they don't help that much anymore.

Outline

Phrase queries

- We want to answer a query such as “stanford university” – as a phrase.

Phrase queries

- We want to answer a query such as "stanford university" – as a phrase.
- Thus The inventor Stanford Ovshinsky never went to university shouldn't be a match.

Phrase queries

- We want to answer a query such as "stanford university" – as a phrase.
- Thus The inventor Stanford Ovshinsky never went to university shouldn't be a match.
- The concept of phrase query has proven easily understood by users.

Phrase queries

- We want to answer a query such as "stanford university" – as a phrase.
- Thus The inventor Stanford Ovshinsky never went to university shouldn't be a match.
- The concept of phrase query has proven easily understood by users.
- About 10% of web queries are phrase queries.

Phrase queries

- We want to answer a query such as "stanford university" – as a phrase.
- Thus The inventor Stanford Ovshinsky never went to university shouldn't be a match.
- The concept of phrase query has proven easily understood by users.
- About 10% of web queries are phrase queries.
- Consequence for inverted index: no longer suffices to store docIDs in postings lists.

Phrase queries

- We want to answer a query such as "stanford university" – as a phrase.
- Thus The inventor Stanford Ovshinsky never went to university shouldn't be a match.
- The concept of phrase query has proven easily understood by users.
- About 10% of web queries are phrase queries.
- Consequence for inverted index: no longer suffices to store docIDs in postings lists.
- Any ideas?

Biword indexes

- Index every consecutive pair of terms in the text as a phrase.

Biword indexes

- Index every consecutive pair of terms in the text as a phrase.
- For example, Friends, Romans, Countrymen would generate two biwords: "friends romans" and "romans countrymen"

Biword indexes

- Index every consecutive pair of terms in the text as a phrase.
- For example, Friends, Romans, Countrymen would generate two biwords: "friends romans" and "romans countrymen"
- Each of these biwords is now a vocabulary term.

Biword indexes

- Index every consecutive pair of terms in the text as a phrase.
- For example, Friends, Romans, Countrymen would generate two biwords: "friends romans" and "romans countrymen"
- Each of these biwords is now a vocabulary term.
- Two-word phrases can now easily be answered.

Longer phrase queries

- A long phrase like "stanford university palo alto" can be represented as the Boolean query "stanford university" AND "university palo" AND "palo alto"

Longer phrase queries

- A long phrase like "stanford university palo alto" can be represented as the Boolean query "stanford university" AND "university palo" AND "palo alto"
- We need to do post-filtering of hits to identify subset that actually contains the 4-word phrase.

Extended biwords

- Parse each document and perform part-of-speech tagging

Extended biwords

- Parse each document and perform part-of-speech tagging
- Bucket the terms into (say) nouns (N) and articles/prepositions (X)

Extended biwords

- Parse each document and perform part-of-speech tagging
- Bucket the terms into (say) nouns (N) and articles/prepositions (X)
- Now deem any string of terms of the form NX*N to be an extended biword

Extended biwords

- Parse each document and perform part-of-speech tagging
- Bucket the terms into (say) nouns (N) and articles/prepositions (X)
- Now deem any string of terms of the form NX*N to be an extended biword
- Examples:

catcher	in	the	rye	king	of	Denmark
N	X	X	N	N	X	N

Extended biwords

- Parse each document and perform part-of-speech tagging
- Bucket the terms into (say) nouns (N) and articles/prepositions (X)
- Now deem any string of terms of the form NX*N to be an extended biword
- Examples:

catcher	in	the	rye	king	of	Denmark
N	X	X	N	N	X	N

- Include extended biwords in the term vocabulary

Extended biwords

- Parse each document and perform part-of-speech tagging
- Bucket the terms into (say) nouns (N) and articles/prepositions (X)
- Now deem any string of terms of the form NX*N to be an extended biword
- Examples:

catcher	in	the	rye	king	of	Denmark
N	X	X	N	N	X	N

- Include extended biwords in the term vocabulary
- Queries are processed accordingly

Issues with biword indexes

- Why are biword indexes rarely used?

Issues with biword indexes

- Why are biword indexes rarely used?
- False positives, as noted above

Issues with biword indexes

- Why are biword indexes rarely used?
- False positives, as noted above
- Index blowup due to very large term vocabulary

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $\underline{\text{to}_1\ \text{be}_2\ \text{or}_3\ \text{not}_4\ \text{to}_5\ \text{be}_6}$

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $\underline{\text{to}_1\ \text{be}_2\ \text{or}_3\ \text{not}_4\ \text{to}_5\ \text{be}_6}$

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $\underline{\text{to}_1\ \text{be}_2\ \text{or}_3\ \text{not}_4\ \text{to}_5\ \text{be}_6}$

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $\underline{\text{to}_1\ \text{be}_2\ \text{or}_3\ \text{not}_4\ \text{to}_5\ \text{be}_6}$

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $\underline{\text{to}_1\ \text{be}_2\ \text{or}_3\ \text{not}_4\ \text{to}_5\ \text{be}_6}$

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $\underline{\text{to}_1\ \text{be}_2\ \text{or}_3\ \text{not}_4\ \text{to}_5\ \text{be}_6}$

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $\underline{\text{to}_1\ \text{be}_2\ \text{or}_3\ \text{not}_4\ \text{to}_5\ \text{be}_6}$

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $\underline{\text{to}_1\ \text{be}_2\ \text{or}_3\ \text{not}_4\ \text{to}_5\ \text{be}_6}$

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $\underline{\text{to}_1\ \text{be}_2\ \text{or}_3\ \text{not}_4\ \text{to}_5\ \text{be}_6}$

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $\underline{\text{to}_1\ \text{be}_2\ \text{or}_3\ \text{not}_4\ \text{to}_5\ \text{be}_6}$

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $\underline{\text{to}_1\ \text{be}_2\ \text{or}_3\ \text{not}_4\ \text{to}_5\ \text{be}_6}$

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $\underline{\text{to}_1\ \text{be}_2\ \text{or}_3\ \text{not}_4\ \text{to}_5\ \text{be}_6}$

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $\underline{\text{to}_1\ \text{be}_2\ \text{or}_3\ \text{not}_4\ \text{to}_5\ \text{be}_6}$

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $\underline{\text{to}_1\ \text{be}_2\ \text{or}_3\ \text{not}_4\ \text{to}_5\ \text{be}_6}$

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $\underline{\text{to}_1\ \text{be}_2\ \text{or}_3\ \text{not}_4\ \text{to}_5\ \text{be}_6}$

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $\underline{\text{to}_1\ \text{be}_2\ \text{or}_3\ \text{not}_4\ \text{to}_5\ \text{be}_6}$

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $\underline{\text{to}_1\ \text{be}_2\ \text{or}_3\ \text{not}_4\ \text{to}_5\ \text{be}_6}$

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $\underline{\text{to}_1\ \text{be}_2\ \text{or}_3\ \text{not}_4\ \text{to}_5\ \text{be}_6}$

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $\underline{\text{to}_1\ \text{be}_2\ \text{or}_3\ \text{not}_4\ \text{to}_5\ \text{be}_6}$

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩
Document 4 is a match!

Proximity search

- We just saw how to use a positional index for phrase searches.

Proximity search

- We just saw how to use a positional index for phrase searches.
- We can also use it for proximity search.

Proximity search

- We just saw how to use a positional index for phrase searches.
- We can also use it for proximity search.
- For example: *employment /3 place*

Proximity search

- We just saw how to use a positional index for phrase searches.
- We can also use it for proximity search.
- For example: *employment /3 place*
- Find all documents that contain employment and place within 3 words of each other.

Proximity search

- We just saw how to use a positional index for phrase searches.
- We can also use it for proximity search.
- For example: *employment /3 place*
- Find all documents that contain employment and place within 3 words of each other.
- Employment agencies that place healthcare workers are seeing growth is a hit.

Proximity search

- We just saw how to use a positional index for phrase searches.
- We can also use it for proximity search.
- For example: *employment /3 place*
- Find all documents that contain employment and place within 3 words of each other.
- Employment agencies that place healthcare workers are seeing growth is a hit.
- Employment agencies that help place healthcare workers are seeing growth is not a hit.

Proximity search

- Simplest algorithm: look at cross-product of positions of (i) employment in document and (ii) place in document

Proximity search

- Simplest algorithm: look at cross-product of positions of (i) employment in document and (ii) place in document
- Very inefficient for frequent words, especially stop words

Proximity search

- Simplest algorithm: look at cross-product of positions of (i) employment in document and (ii) place in document
- Very inefficient for frequent words, especially stop words
- Note that we want to return the actual matching positions, not just a list of documents.

Proximity search

- Simplest algorithm: look at cross-product of positions of (i) employment in document and (ii) place in document
- Very inefficient for frequent words, especially stop words
- Note that we want to return the actual matching positions, not just a list of documents.
- This is important for dynamic summaries etc.

"Proximity" intersection

```
POSITIONALINTERSECT(p1, p2, k)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then l ← ⟨ ⟩
 5             pp1 ← positions(p1)
 6             pp2 ← positions(p2)
 7             while pp1 ≠ NIL
 8             do while pp2 ≠ NIL
 9                do if |pos(pp1) − pos(pp2)| ≤ k
10                      then ADD(l, pos(pp2))
11                      else if pos(pp2) > pos(pp1)
12                              then break
13                   pp2 ← next(pp2)
14                while l ≠ ⟨ ⟩ and |l[0] − pos(pp1)| > k
15                do DELETE(l[0])
16                for each ps ∈ l
17                do ADD(answer, ⟨docID(p1), pos(pp1), ps⟩)
18                pp1 ← next(pp1)
19             p1 ← next(p1)
20             p2 ← next(p2)
21        else if docID(p1) < docID(p2)
22               then p1 ← next(p1)
23               else p2 ← next(p2)
24  return answer
```

Combination scheme

- Biword indexes and positional indexes can be profitably combined.

Combination scheme

- Biword indexes and positional indexes can be profitably combined.
- Many biwords are extremely frequent: Michael Jackson, Britney Spears etc

Combination scheme

- Biword indexes and positional indexes can be profitably combined.
- Many biwords are extremely frequent: Michael Jackson, Britney Spears etc
- For these biwords, increased speed compared to positional postings intersection is substantial.

Combination scheme

- Biword indexes and positional indexes can be profitably combined.
- Many biwords are extremely frequent: Michael Jackson, Britney Spears etc
- For these biwords, increased speed compared to positional postings intersection is substantial.
- Combination scheme: Include frequent biwords as vocabulary terms in the index. Do all other phrases by positional intersection.

Combination scheme

- Biword indexes and positional indexes can be profitably combined.
- Many biwords are extremely frequent: Michael Jackson, Britney Spears etc
- For these biwords, increased speed compared to positional postings intersection is substantial.
- Combination scheme: Include frequent biwords as vocabulary terms in the index. Do all other phrases by positional intersection.
- Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme. Faster than a positional index, at a cost of 26% more space for index.

"Positional" queries on Google

- For web search engines, positional queries are much more expensive than regular Boolean queries.

"Positional" queries on Google

- For web search engines, positional queries are much more expensive than regular Boolean queries.
- Let's look at the example of phrase queries.

"Positional" queries on Google

- For web search engines, positional queries are much more expensive than regular Boolean queries.
- Let's look at the example of phrase queries.
- Why are they more expensive than regular Boolean queries?

"Positional" queries on Google

- For web search engines, positional queries are much more expensive than regular Boolean queries.
- Let's look at the example of phrase queries.
- Why are they more expensive than regular Boolean queries?
- Can you demonstrate on Google that phrase queries are more expensive than Boolean queries?

Resources

- Chapter 2 of IIR

Resources

- Chapter 2 of IIR
- Resources at http://ifnlp.org/ir

Resources

- Chapter 2 of IIR
- Resources at http://ifnlp.org/ir
- Porter stemmer