Multimedia Information Retrieval and Technology

L2. Query Processing

By : Fangyu Wu

Room: SD555

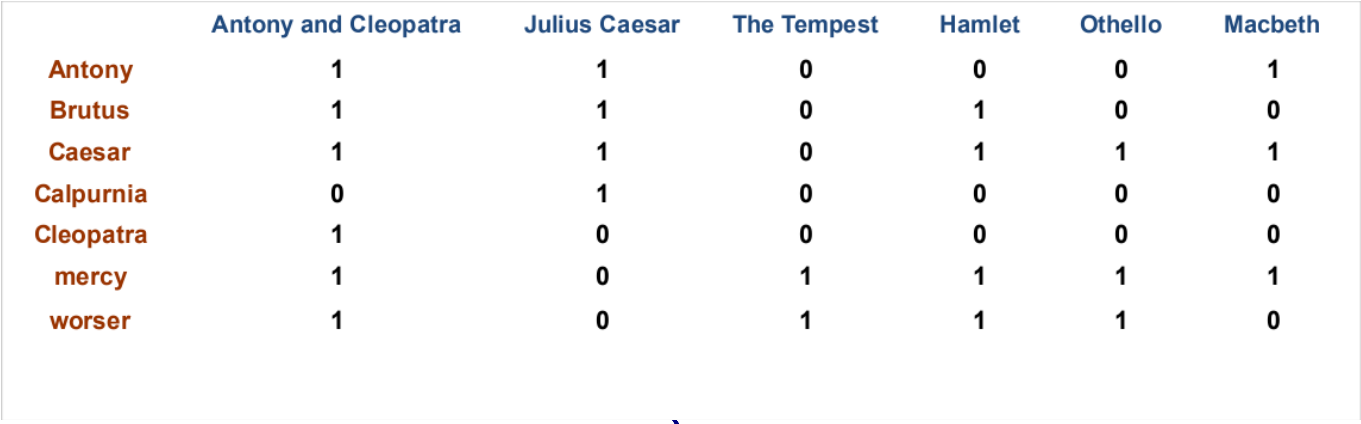
Tel. no. 88161780

Xi’an Jiaotong-Liverpool University

西交利物浦大学

Recap

Term-document incidence matrices

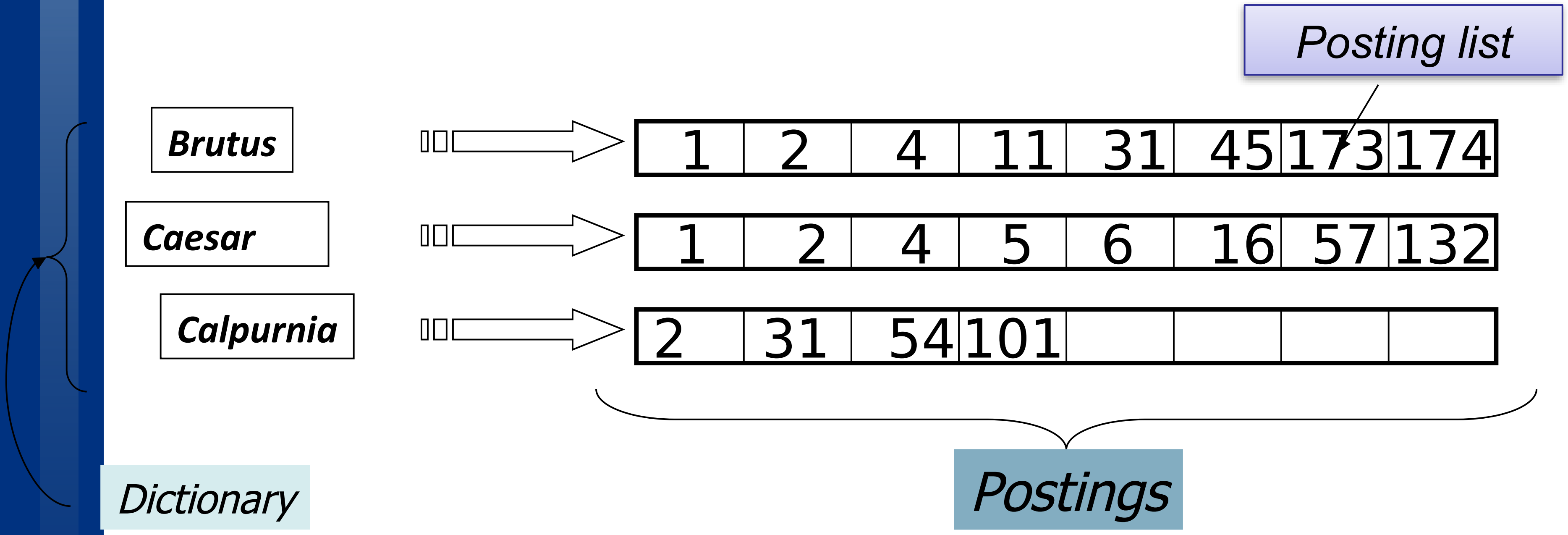


	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

1 if play contains word, 0 otherwise

Recap

Inverted index



Brutus	1	2	4	11	31	45	173	174
Caesar	1	2	4	5	6	16	57	132
Calpurnia	2	31	54	101				

Shakespeare plays

http://www.rhymezone.com/shakespeare/

Processing Boolean queries

I. Query processing with an inverted index
II. Query optimization
III. The Extended Boolean Models
IV. Faster posting list intersection

The index we just built

How do we process a query?

AND operation

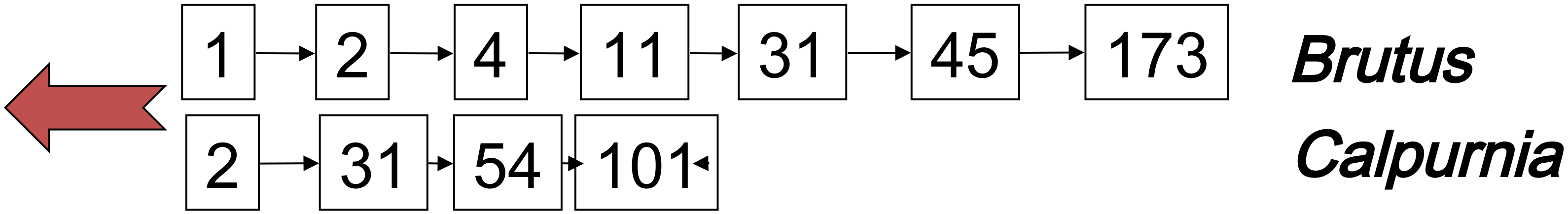
The *intersection* operation is the crucial one: we need to efficiently intersect postings lists so as to be able to quickly find documents that contain both terms.

Query processing: AND

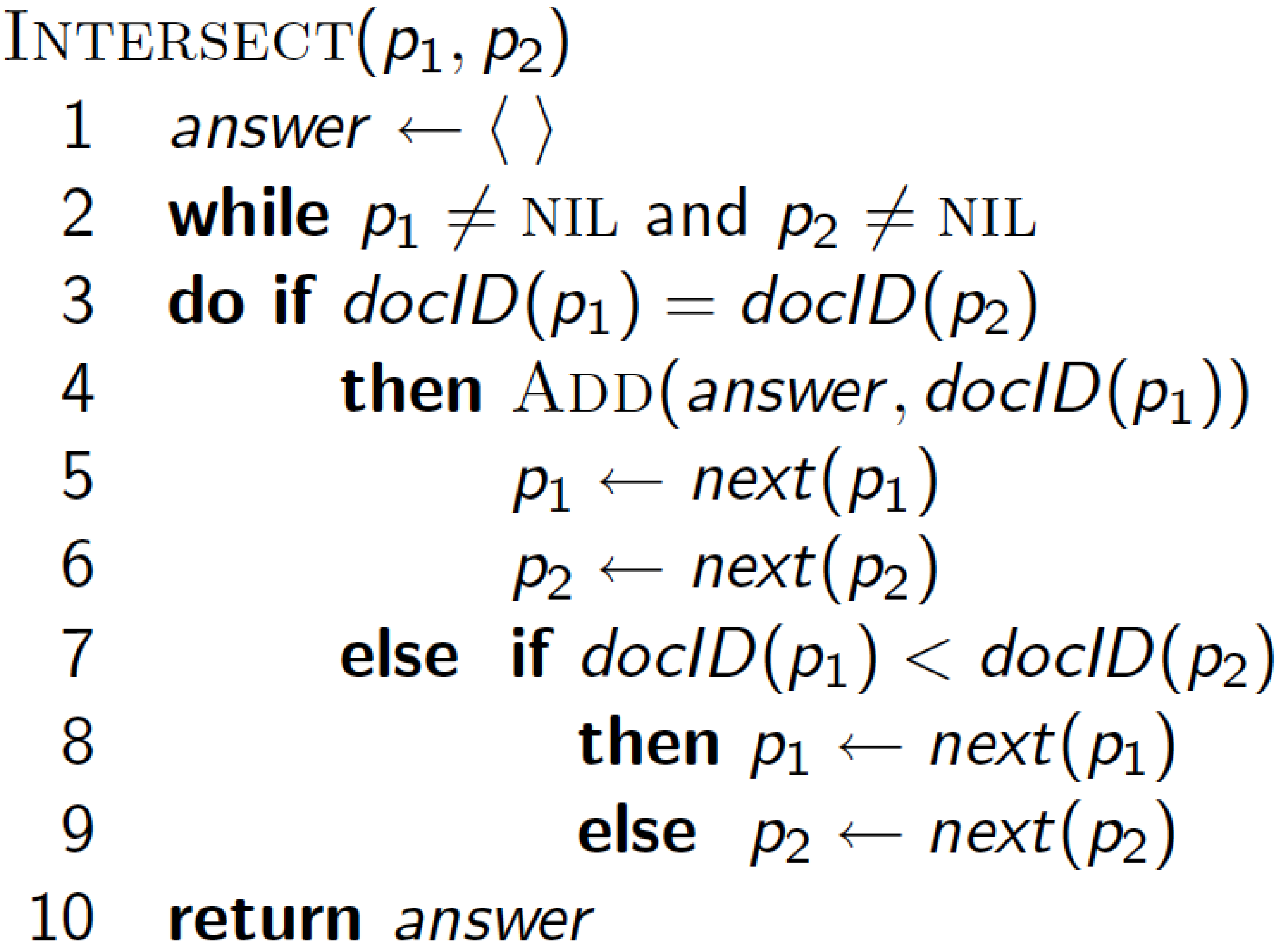
Consider processing the query:

Brutus *AND* ***Calpurnia***

1. Locate ***Brutus*** in the Dictionary; Retrieve its postings.
2. Locate ***Calpurnia*** in the Dictionary; Retrieve its postings.
3. "Merge" the two postings (intersect the document sets):



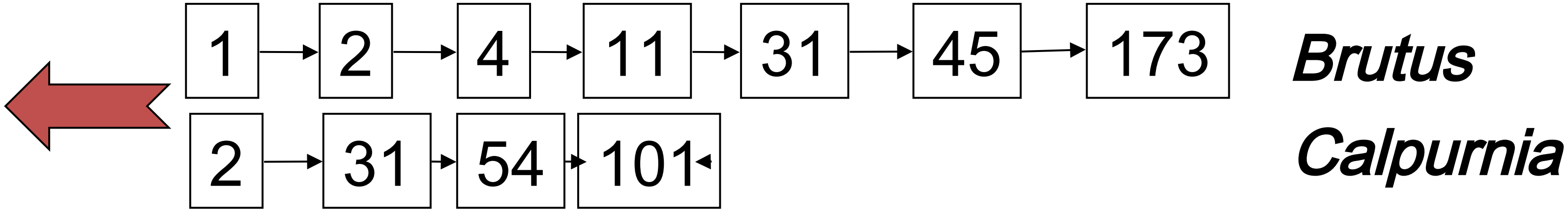
Intersecting two postings lists (a “merge” algorithm)



```
INTERSECT($p_1, p_2$)
answer ← ⟨ ⟩
while $p_1 \neq$ NIL and $p_2 \neq$ NIL
do if $docID(p_1) = docID(p_2)$
        then ADD($answer, docID(p_1)$)
             $p_1 \leftarrow next(p_1)$
             $p_2 \leftarrow next(p_2)$
        else if $docID(p_1) < docID(p_2)$
               then $p_1 \leftarrow next(p_1)$
               else $p_2 \leftarrow next(p_2)$
return answer
```

The merge

Walk through the two postings simultaneously, in time linear in the total number of postings entries



1 → 2 → 4 → 11 → 31 → 45 → 173 ***Brutus***

2 → 31 → 54 → 101 ***Calpurnia***

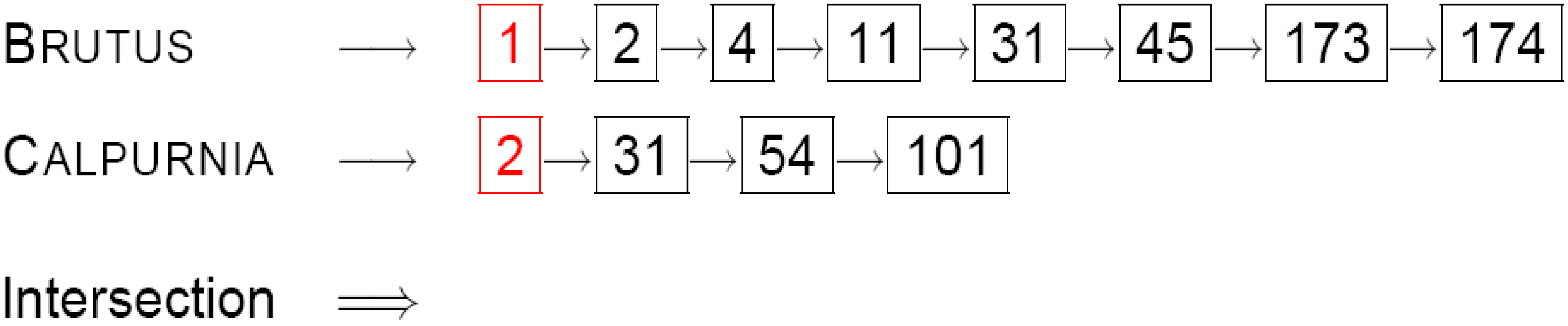
Intersecting two postings lists

$$\text{BRUTUS} \quad \longrightarrow \quad \boxed{1} \to \boxed{2} \to \boxed{4} \to \boxed{11} \to \boxed{31} \to \boxed{45} \to \boxed{173} \to \boxed{174}$$

$$\text{CALPURNIA} \quad \longrightarrow \quad \boxed{2} \to \boxed{31} \to \boxed{54} \to \boxed{101}$$

$$\text{Intersection} \quad \Longrightarrow$$

Intersecting two postings lists

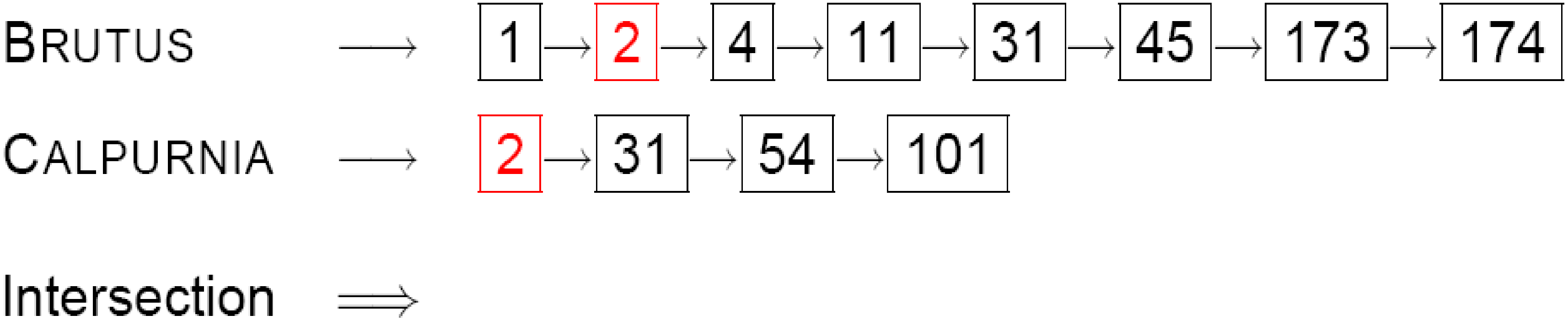


Brutus ⟶ 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

Calpurnia ⟶ 2 → 31 → 54 → 101

Intersection ⟹

Intersecting two postings lists

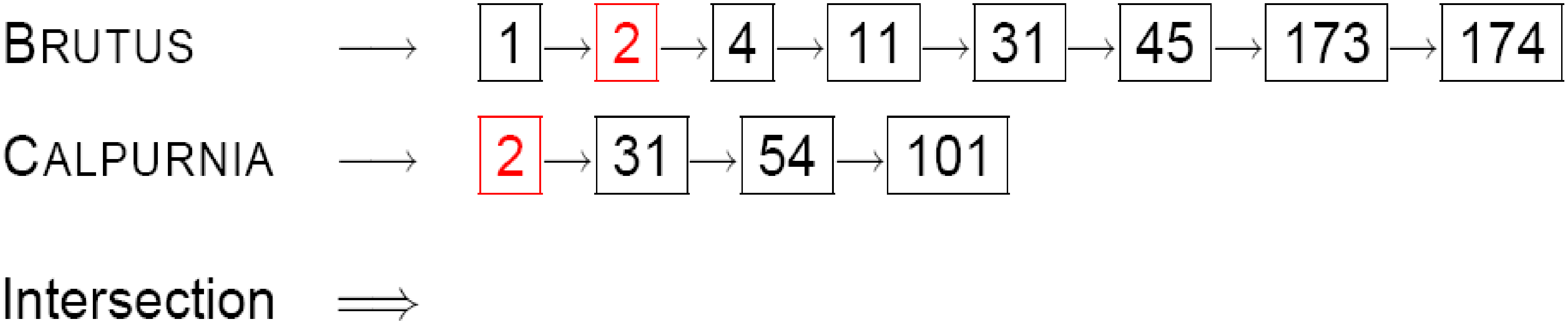


BRUTUS ⟶ 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

CALPURNIA ⟶ 2 → 31 → 54 → 101

Intersection ⟹

Intersecting two postings lists

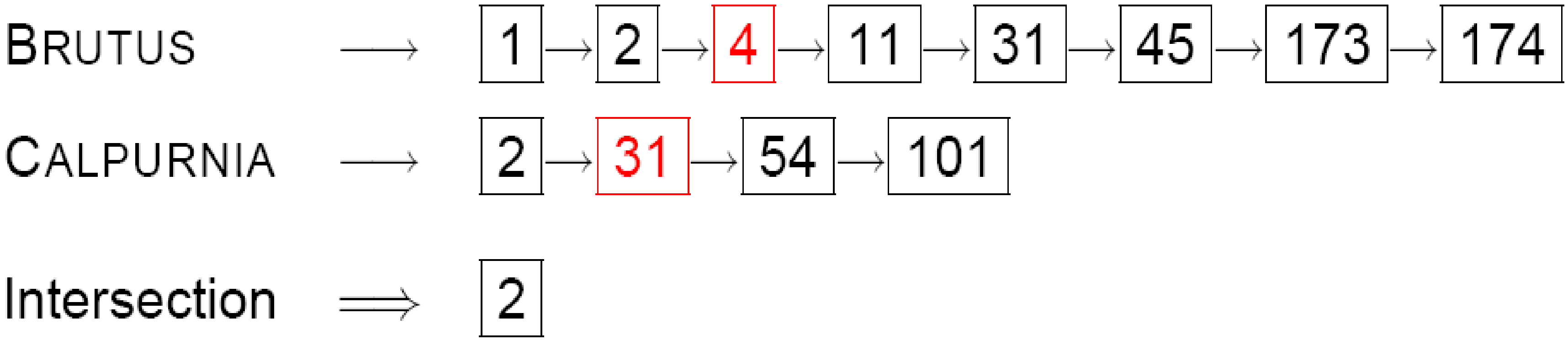


BRUTUS ⟶ 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

CALPURNIA ⟶ 2 → 31 → 54 → 101

Intersection ⟹

Intersecting two postings lists

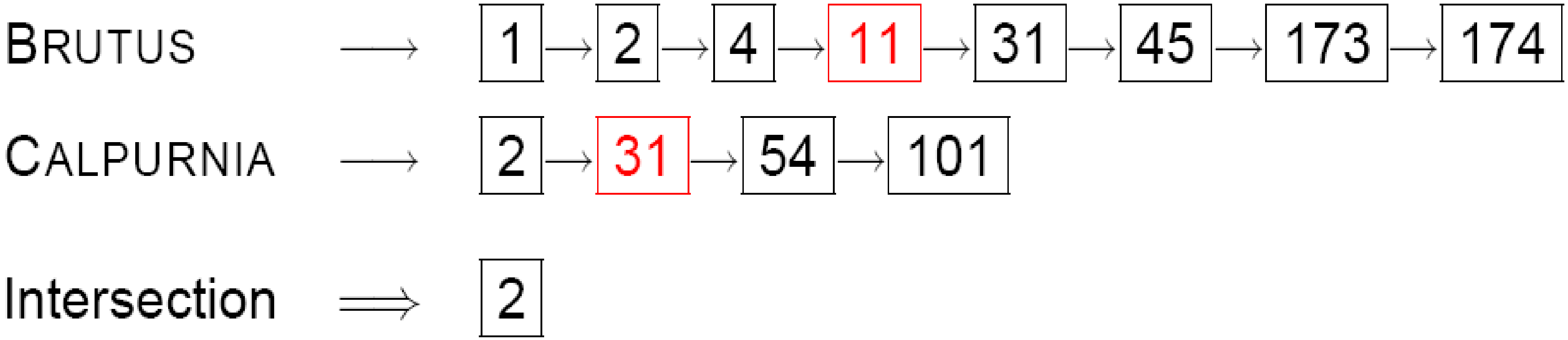


BRUTUS ⟶ 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

CALPURNIA ⟶ 2 → 31 → 54 → 101

Intersection ⟹ 2

Intersecting two postings lists

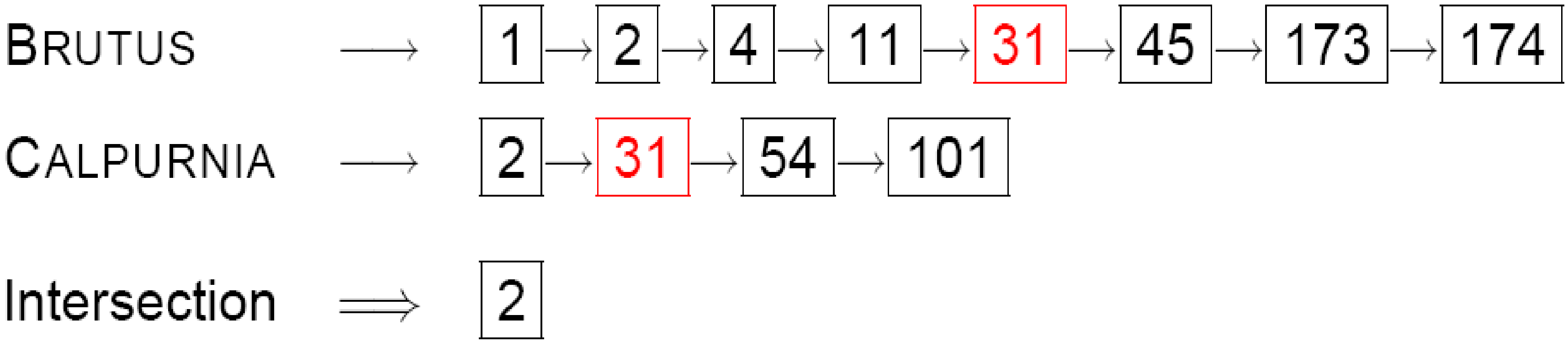


BRUTUS ⟶ 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

CALPURNIA ⟶ 2 → 31 → 54 → 101

Intersection ⟹ 2

Intersecting two postings lists

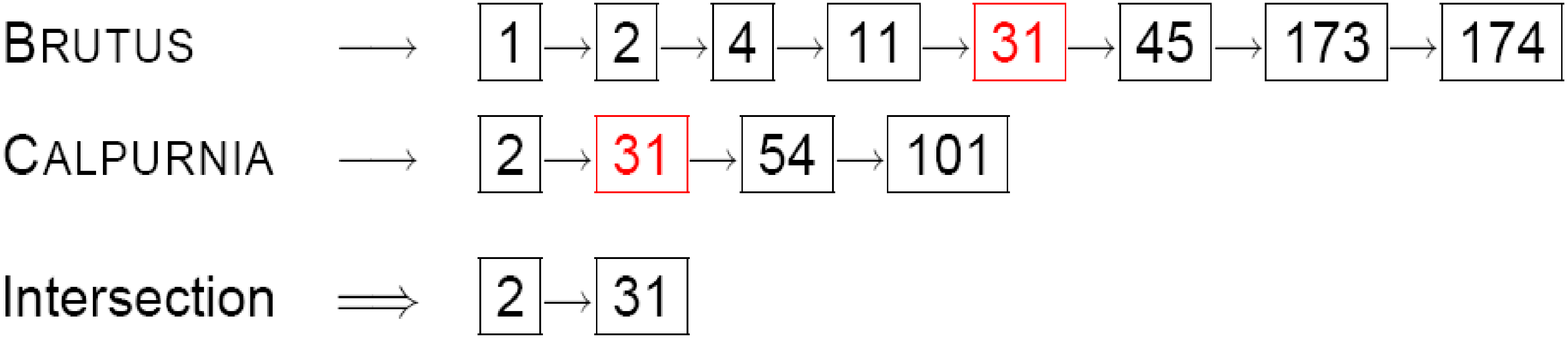


Brutus → 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

Calpurnia → 2 → 31 → 54 → 101

Intersection ⟹ 2

Intersecting two postings lists

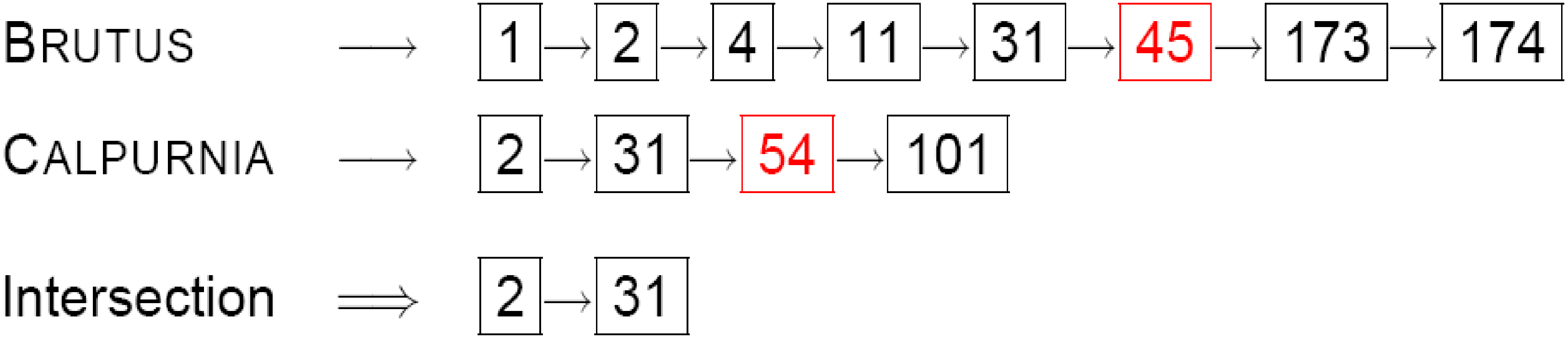


BRUTUS ⟶ 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

CALPURNIA ⟶ 2 → 31 → 54 → 101

Intersection ⟹ 2 → 31

Intersecting two postings lists

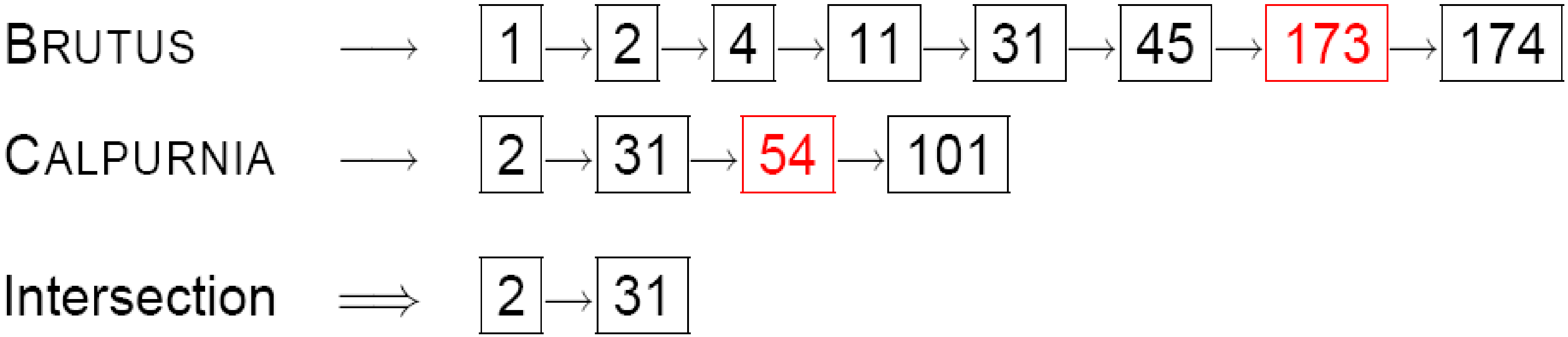


Brutus ⟶ 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

Calpurnia ⟶ 2 → 31 → 54 → 101

Intersection ⟹ 2 → 31

Intersecting two postings lists

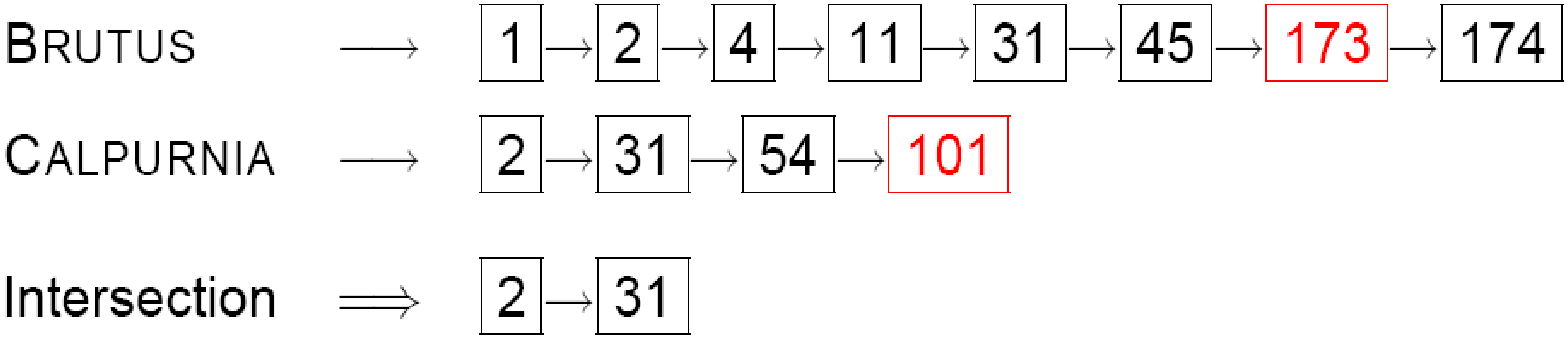


BRUTUS ⟶ 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

CALPURNIA ⟶ 2 → 31 → 54 → 101

Intersection ⟹ 2 → 31

Intersecting two postings lists

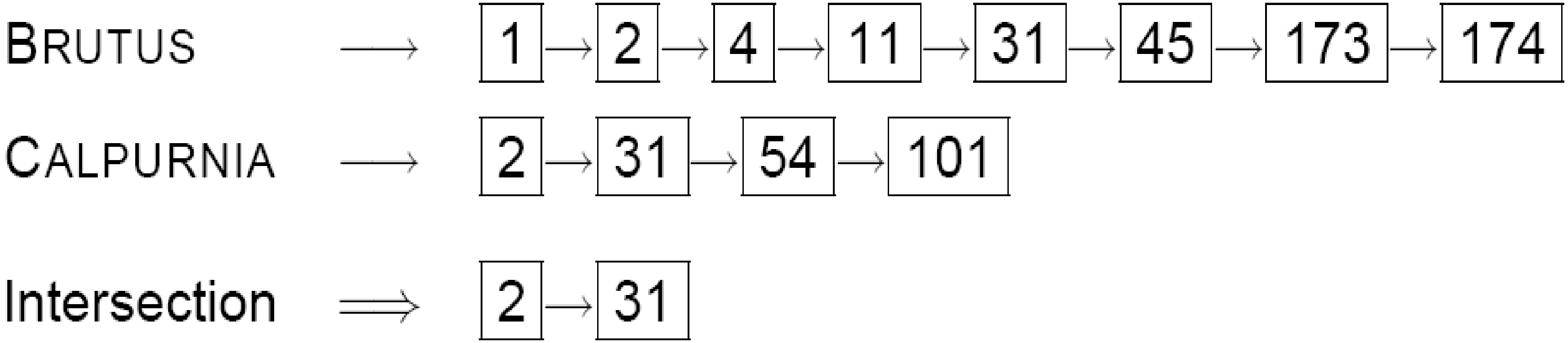


BRUTUS → 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

CALPURNIA → 2 → 31 → 54 → 101

Intersection ⟹ 2 → 31

Intersecting two postings lists

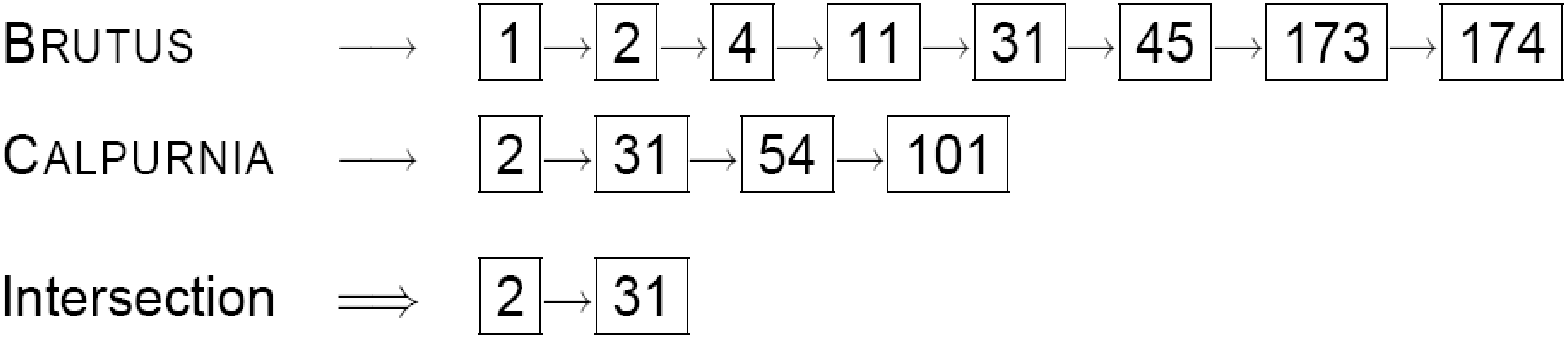


BRUTUS → 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

CALPURNIA → 2 → 31 → 54 → 101

Intersection ⟹ 2 → 31

Intersecting two postings lists



BRUTUS ⟶ 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

CALPURNIA ⟶ 2 → 31 → 54 → 101

Intersection ⟹ 2 → 31

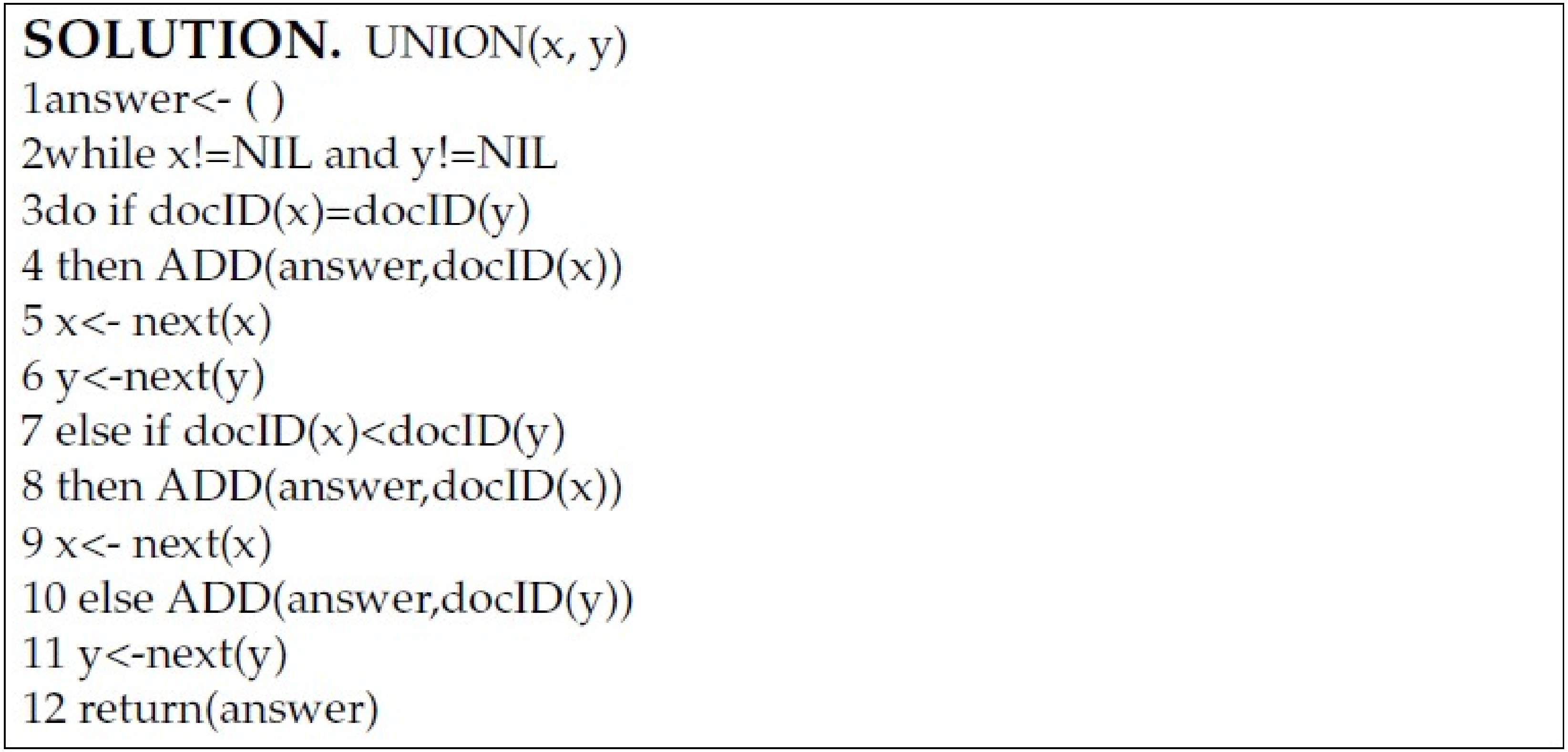
- This is linear in the length of the postings lists.
- This only works if postings lists are sorted.

If the list lengths are *x* and *y*, the merge takes O(*x*+*y*) operations.
Crucial: postings sorted by docID.

Exercise:

Write out a postings merge algorithm for an *x* or *y* query

```
Or(x,y>
answer←()
while x!null && y!null
    do if docID(x)=docID(y)
        then ADD(docID(x))
            x←next(x)
            y←next(y)
    else if docID(x)<docID(y)
        ADD(answer,x)
        x ← next(x)
    else
        ADD(answer,y)
        y←next(y)
return answer
```



SOLUTION. UNION(x, y)

```
answer<- ( )
while x!=NIL and y!=NIL
    do if docID(x)=docID(y)
        then ADD(answer,docID(x))
            x<- next(x)
            y<-next(y)
        else if docID(x)<docID(y)
            then ADD(answer,docID(x))
                x<- next(x)
            else ADD(answer,docID(y))
                y<-next(y)
return(answer)
```

I. Query processing with an inverted index

II. Query optimization

III. The Extended Boolean Models

IV. Faster posting list intersection

Query Optimization

The process of selecting how to organize the work of answering a query so that the least amount of work needs to be done by the system.

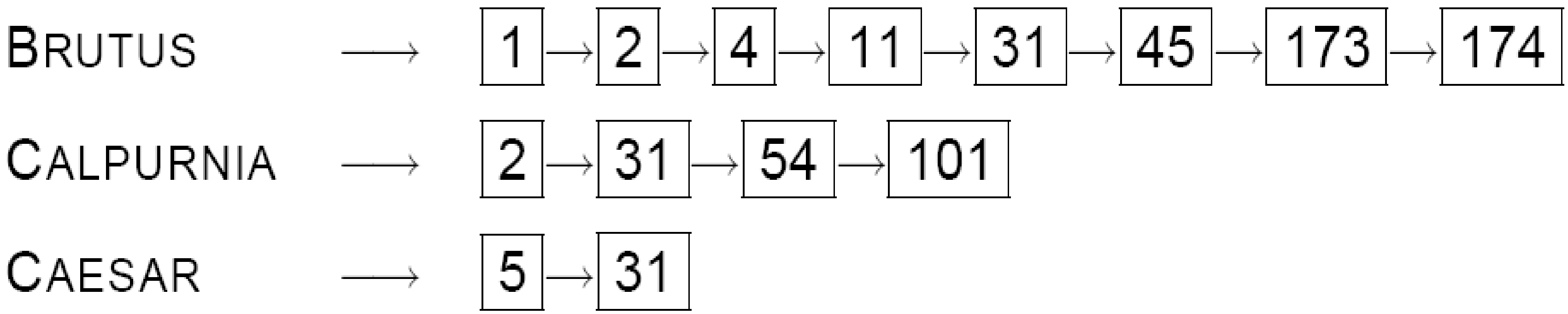
Query Optimization

- What is the best order for query processing?
- Consider a query that is an AND of n terms, n>2:

  Brutus and Calpurnia and Caesar

Query optimization

- Example query: BRUTUS AND CALPURNIA AND CAESAR



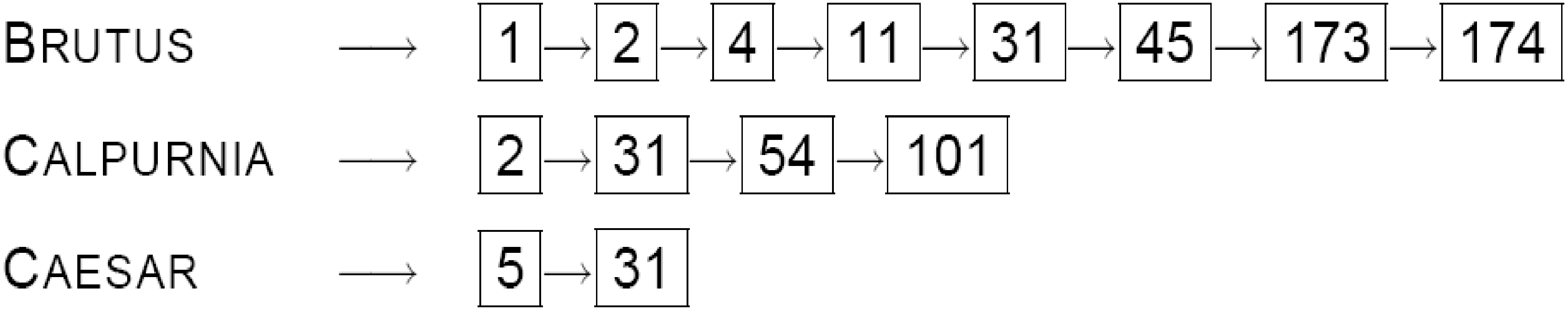
BRUTUS ⟶ 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

CALPURNIA ⟶ 2 → 31 → 54 → 101

CAESAR ⟶ 5 → 31

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



BRUTUS → 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

CALPURNIA → 2 → 31 → 54 → 101

CAESAR → 5 → 31

Optimized intersection algorithm for conjunctive queries

```
Intersect($\langle t_1, \ldots, t_n \rangle$)
    terms ← SortByIncreasingFrequency($\langle t_1, \ldots, t_n \rangle$)
    result ← postings(first(terms))
    terms ← rest(terms)
    while terms $\neq$ nil and result $\neq$ nil
    do result ← Intersect(result, postings(first(terms)))
        terms ← rest(terms)
    return result
```

Query Optimization

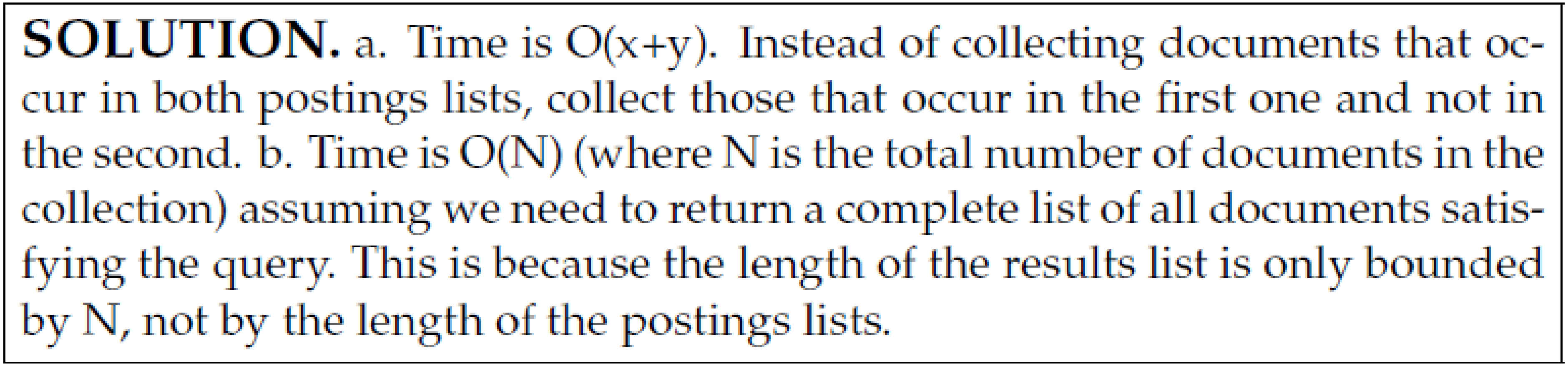
This is the first justification for keeping the frequency of terms in the dictionary, it allows us to make this ordering decision based on in-memory data before accessing any postings list.

Exercise

For the queries below, can we still run through the intersection in time $O(x + y)$ for a collection size N (number of total documents), where x and y are the lengths of the postings lists for Brutus and Caesar? If not, what can we achieve?

a. Brutus AND NOT Caesar

b. Brutus OR NOT Caesar



SOLUTION. a. Time is O(x+y). Instead of collecting documents that occur in both postings lists, collect those that occur in the first one and not in the second. b. Time is O(N) (where N is the total number of documents in the collection) assuming we need to return a complete list of all documents satisfying the query. This is because the length of the results list is only bounded by N, not by the length of the postings lists.

Exercise:

If the query is ***friends*** *AND* ***romans*** *AND (NOT* ***countrymen****)*, how could we use the frequency of ***countrymen***?

I. Query processing with an inverted index

II. Query optimization

III. The Extended Boolean Models

IV. Faster posting list intersection

Boolean queries: Exact match

- The Boolean retrieval model is being able to ask a query that is a Boolean expression:
  - Boolean Queries are queries using *AND, OR* and *NOT* to join query terms
    - Views each document as a set of words
    - Boolean retrieval is precise: document matches condition or not. (No ranking)
  - Perhaps the simplest model to build an IR system on

Boolean model VS ranked retrieval

Boolean retrieval model：

- Primary commercial retrieval tool for 3 decades.
- Many search systems still in use are Boolean:
  - Email, library catalog, Mac OS X Spotlight

In ranked retrieval models, users largely use free text queries, that is, just typing one or more words rater than using a precise language with operators for building up query expressions, and the system decides which documents best satisfy the query.

Example: WestLaw http://www.westlaw.com/

Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992; new federated search added 2010)

Tens of terabytes of data; ~700,000 users

Majority of users still use boolean queries

Example: WestLaw

Operators:

- & is AND
- /s, /p, /k ask for matches in the same sentence, same paragraph or within k words respectively.
- The exclamation mark (!) gives a trailing wildcard query.
  - disab! matches all words starting with disab.

Example: WestLaw

Information need: What is the statute of limitations in cases involving the federal tort claims act?

Query: LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM

/3 = within 3 words,
/S = in same sentence

Example: WestLaw

- Long, precise queries; proximity operators; incrementally developed; not like web search
- Many professional searchers still like Boolean search
  - You know exactly what you are getting

I. Query processing with an inverted index

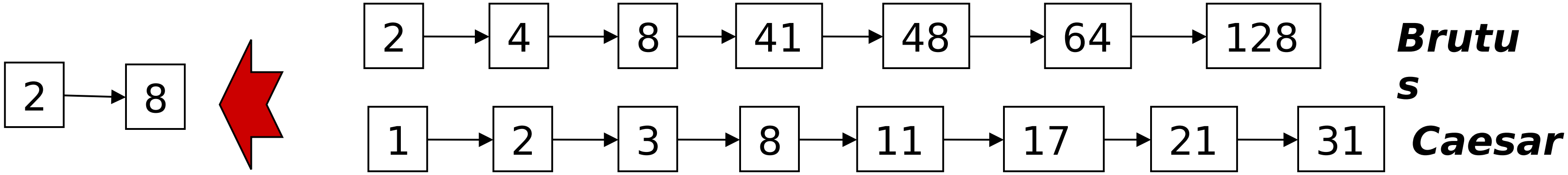
II. Query optimization

III. The Extended Boolean Models

IV. Faster posting list intersection

Recall basic merge

Walk through the two postings simultaneously, in time linear in the total number of postings entries

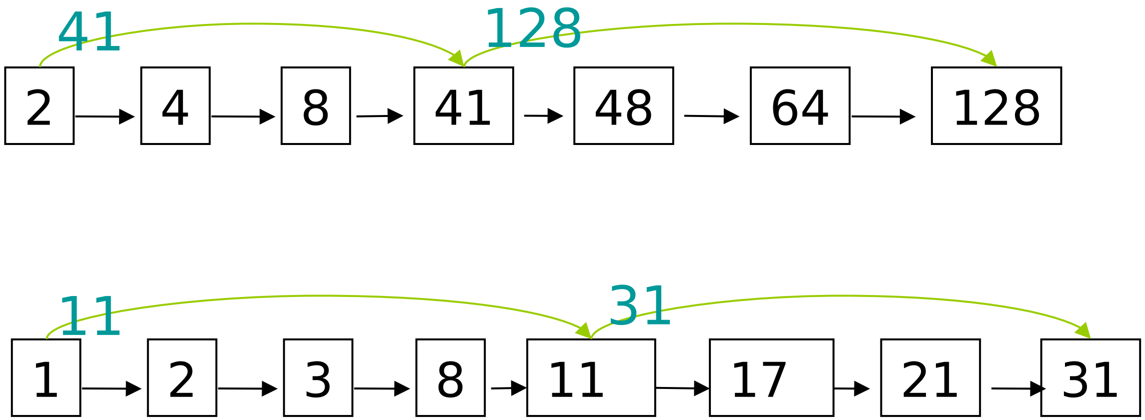


If the list lengths are *m* and *n*, the merge takes O(*m*+*n*) operations.

Can we do better?

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

Augment postings with skip pointers (at indexing time)

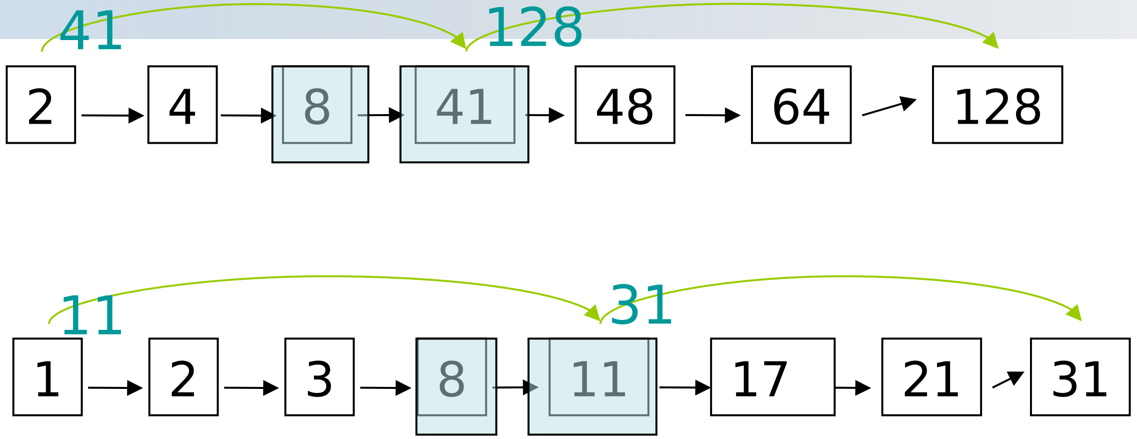


Why?

To skip postings that will not figure in the search results.

Where do we place skip pointers? How to do efficient merging using skip pointers.

Query processing with skip pointers



1. We process **8** on each list

- We match it and advance.
- We then have **41** and **11**, **11** is smaller. Rather than advancing the lower pointer, we first check the skip pointer, and note that **31** is also smaller than 41.
- we can skip ahead past the intervening postings and advance the lower pointer to **31.**

Xi'an Jiaotong-Liverpool University 西交利物浦大学

Intersecting with skip pointers

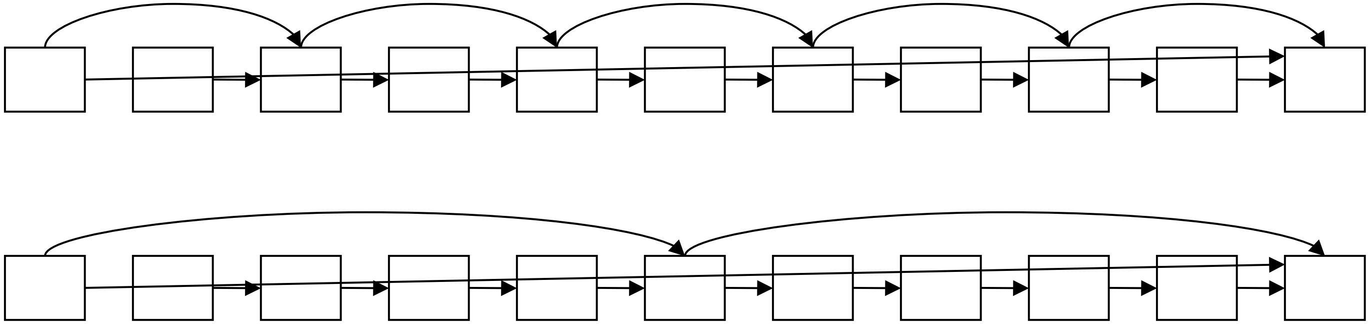
```
IntersectWithSkips($p_1, p_2$)
answer $\leftarrow \langle\ \rangle$
while $p_1 \neq$ NIL and $p_2 \neq$ NIL
do if $docID(p_1) = docID(p_2)$
      then Add($answer, docID(p_1)$)
           $p_1 \leftarrow next(p_1)$
           $p_2 \leftarrow next(p_2)$
      else if $docID(p_1) < docID(p_2)$
              then if $hasSkip(p_1)$ and $(docID(skip(p_1)) \leq docID(p_2))$
                      then $p_1 \leftarrow skip(p_1)$
                      else $p_1 \leftarrow next(p_1)$
              else if $hasSkip(p_2)$ and $(docID(skip(p_2)) \leq docID(p_1))$
                      then $p_2 \leftarrow skip(p_2)$
                      else $p_2 \leftarrow next(p_2)$
return answer
```

Query processing with skip pointers

Skip pointers will only be available for the original postings lists. (The skip pointer is put at index construction time when we do sorting.)

The presence of skip pointers only helps for AND queries, not for OR queries.

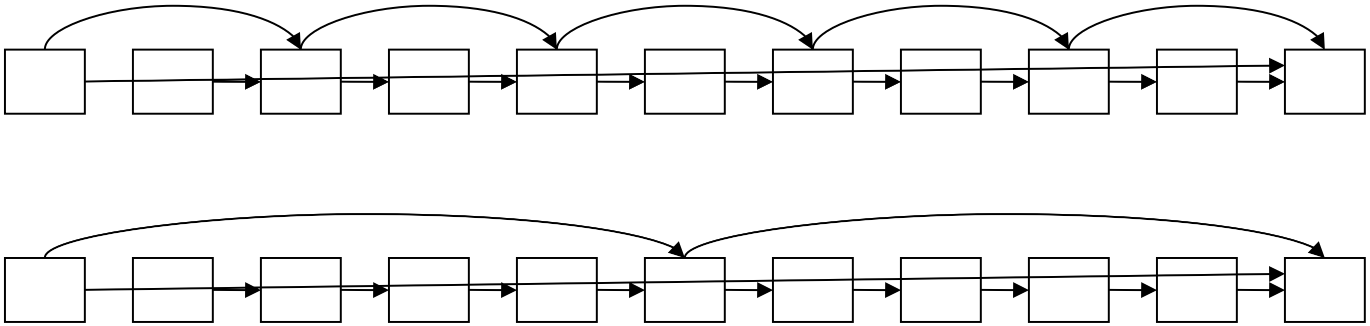
Where do we place skips?



Where do we place skips?

Tradeoff:

- More skips → shorter skip spans ⇒ more likely to skip. But lots of comparisons to skip pointers.
- Fewer skips → few pointer comparison, but then long skip spans ⇒ few successful skips.



Placing skips

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

This ignores the distribution of query terms.

Easy if the index is relatively static; harder if L keeps changing because of updates.