

## Boolean queries: More general merges

---

- ▶ **Exercise:** Adapt the merge for the queries:

***Brutus AND NOT Caesar***

***Brutus OR NOT Caesar***

Can we still run through the merge in time  $O(x + y)$ ?

# Merging

---

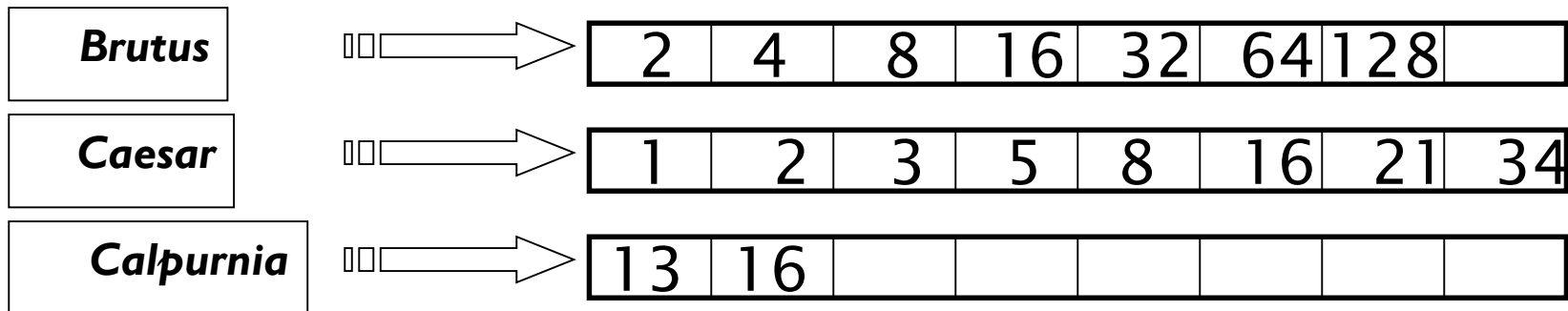
What about an arbitrary Boolean formula?

***(Brutus OR Caesar) AND NOT (Antony OR Cleopatra)***

- ▶ Can we merge in “linear” time for general Boolean queries?
  - ▶ Linear in what?
- ▶ Can we do better?

# Query optimization

- ▶ What is the best order for query processing?
- ▶ Consider a query that is an *AND* of  $n$  terms.
- ▶ For each of the  $n$  terms, get its postings, then *AND* them together.

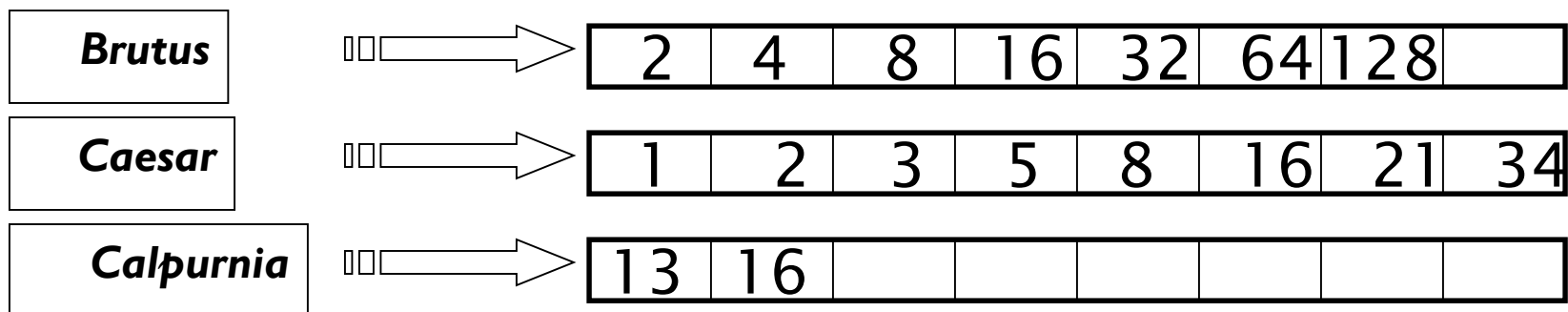


**Query: Brutus AND Calpurnia AND Caesar**

# Query optimization example

- ▶ Process in order of increasing freq:
  - ▶ *start with smallest set, then keep cutting further.*

This is why we kept document freq. in dictionary



Execute the query as ***(Calpurnia AND Brutus) AND Caesar.***

## More general optimization

---

▶ Example:

***(madding OR crowd) AND (ignoble OR strife)***

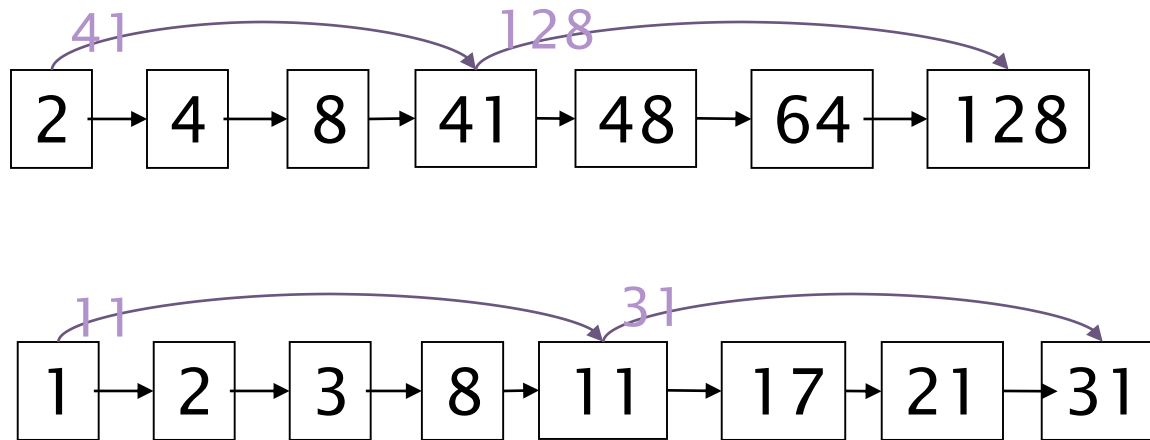
- ▶ Get doc frequencies for all terms.
- ▶ Estimate the size of each *OR* by the sum of its doc. freq.'s (conservative).
- ▶ Process in increasing order of *OR* sizes.

Faster postings merges: skip lists



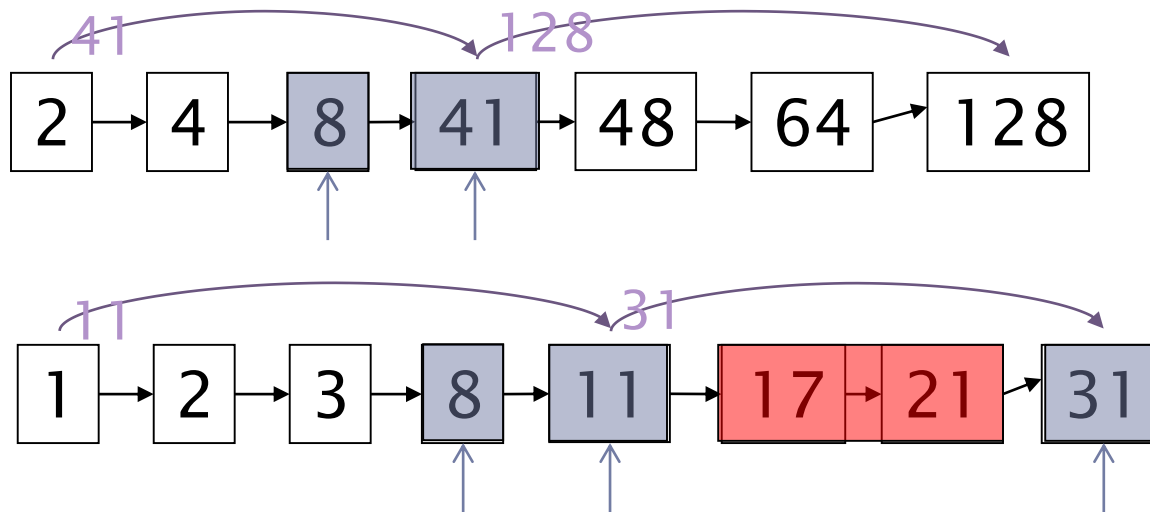
# Augment postings with skip pointers

---



- ▶ It is useful for AND queries
- ▶ To skip postings that will not figure in the results.
- ▶ Where do we place skip pointers?

# Query processing with skip pointers



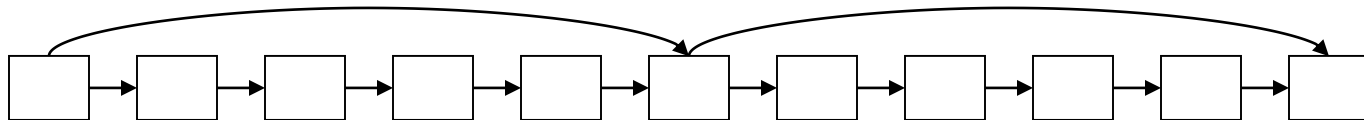
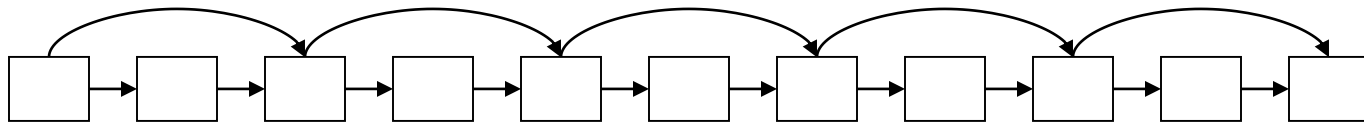
- ❖ Suppose we are processing **8** on each list. We match it and advance.
- ❖ We then have **41** and **11**.
- ❖ The skip successor of **11** is **31** ( $31 < 41$ ). So, we can skip ahead past the intervening postings.

# Where do we place skips?

---

## ▶ Tradeoff:

- ▶ More skips → shorter skip spans
  - ▶ More likely to skip but lots of comparisons to skip pointers (and also more space for them)
- ▶ Fewer skips → long skip spans
  - ▶ few successful skips but also few pointer comparison (and also less space for them)



# Placing skips

---

- ▶ Simple heuristic
  - ▶ For posting of length  $L$ , use  $\sqrt{L}$  evenly-spaced skip pointers
  - ▶ Easy if the index is relatively static
- ▶ This ignores the distribution of query terms
- ▶ This definitely used to help; with modern hardware it may not unless you're memory-based (Bahle et. al 2002)
  - ▶ The I/O cost of loading bigger postings list can outweigh the gains from in memory merging

# Summary of Boolean IR

# Advantages of exact match

---

- ▶ It can be implemented very efficiently
- ▶ Predictable, easy to explain
  - ▶ precise semantics
- ▶ Structured queries for pinpointing precise docs
  - ▶ neat formalism
- ▶ Work well when you know exactly (or roughly) what the collection contains and what you're looking for

# Disadvantages of the Boolean Model

---

- ▶ Query formulation (Boolean expression) is difficult for most users
  - ▶ Too simplistic Boolean queries by most users
  - ▶ AND, OR as opposite extremes in a precision/recall tradeoff
    - ▶ Usually either too few or too many docs in response to a user query
- ▶ Retrieval based on binary decision criteria
  - ▶ No ranking of the docs is provided
- ▶ Difficulty increases with collection size

# Ranking results in advanced IR models

---

- ▶ Boolean queries give inclusion or exclusion of docs.
  - ▶ Results of queries in Boolean model as a set
- ▶ Modern information retrieval systems are no longer based on the Boolean model
- ▶ Often we want to rank/group results
  - ▶ Need to measure proximity from query to each doc.
  - ▶ Index term weighting can provide a substantial improvement