Web Search:
Techniques, algorithms and Aplications

Basic Techniques
for Web Search

German Rigau <german.rigau@ehu.es>

[Based on slides by Eneko Agirre …
and Christopher Manning and Prabhakar Raghavan]

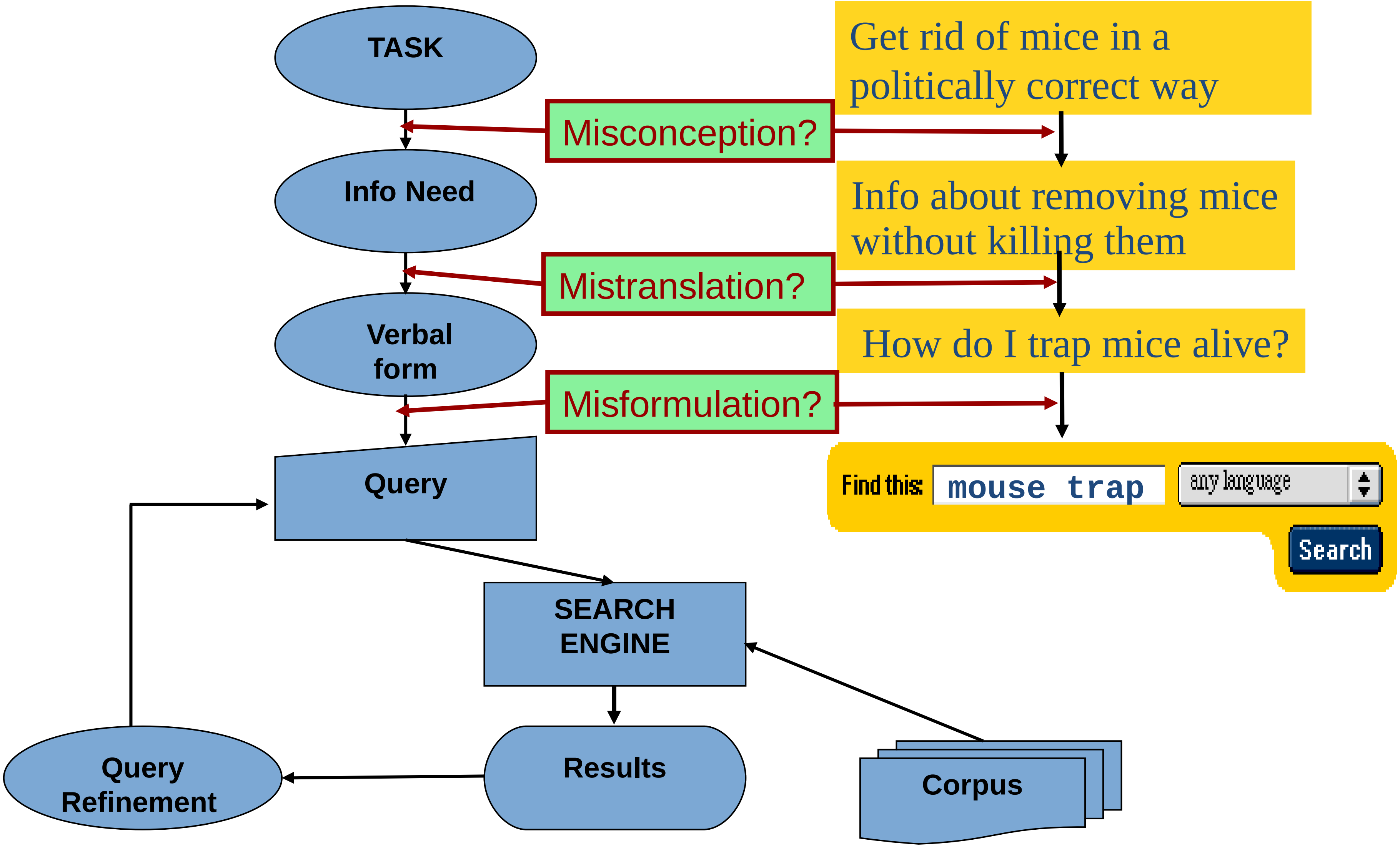
Basic Techniques for Web Search

- Review of applications
- Basic Techniques in detail:
  - **Boolean search**
  - Vocabularies, dictionaries, index
  - Scoring, complete system, evaluation
  - Web search
- Semantic search

Basic assumptions of Information Retrieval

- Collection: Fixed set of documents
- Goal: Retrieve documents with information that is relevant to the user’s information need and helps the user complete a task

The classic search model



TASK

Info Need

Verbal form

Query

SEARCH ENGINE

Results

Query Refinement

Corpus

Misconception?

Mistranslation?

Misformulation?

Get rid of mice in a politically correct way

Info about removing mice without killing them

How do I trap mice alive?

Find this: mouse trap

any language

Search

How good are the retrieved docs?

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

Information retrieval in 1680

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 is that not the answer?

- Slow (for large corpora)
- *NOT* ***Calpurnia*** is non-trivial
- Other operations (e.g., find the word ***Romans*** near ***countrymen***) not feasible
- Ranked retrieval (best documents to return)

Term-document incidence

	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

Brutus** AND **Caesar** BUT NOT **Calpurnia

1 if play contains word, 0 otherwise

Incidence vectors

So we have a 0/1 vector for each term.

To answer query: take the vectors for ***Brutus, Caesar*** and ***Calpurnia*** (complemented) → bitwise *AND*.

110100 *AND* 110111 *AND* 101111 = 100100.

Answers to query

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

Bigger collections

- Consider *N* = 1 million documents, each with about 1000 words.
- Avg 6 bytes/word including spaces/punctuation
  - 6GB of data in the documents.
- Say there are *M* = 500K *distinct* terms among these.

Can’t build the matrix

- 500K x 1M matrix has half-a-trillion 0’s and 1’s.
- But it has no more than one billion 1’s.
  - Matrix is extremely sparse.
- What’s a better representation?
  - We only record the 1 positions.

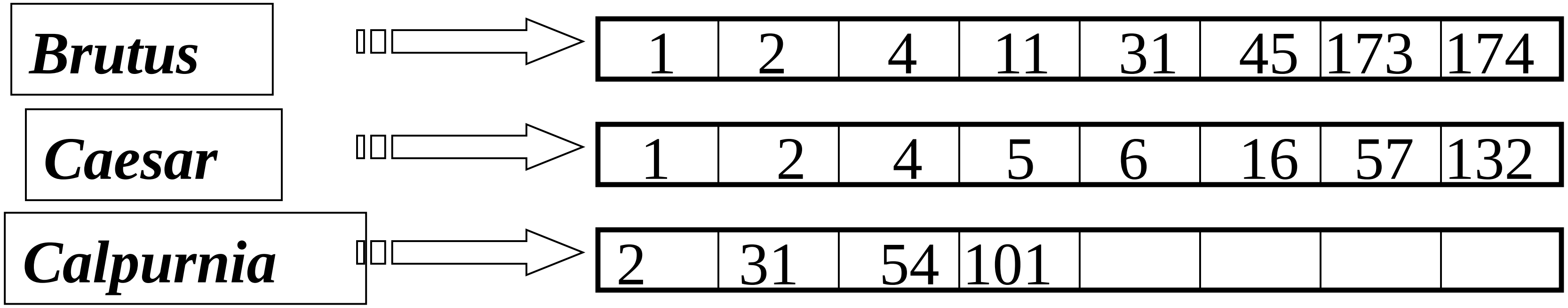
Why?

Inverted index

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

- Identify each by a **docID**, a document serial number

Can we used fixed-size arrays for this?



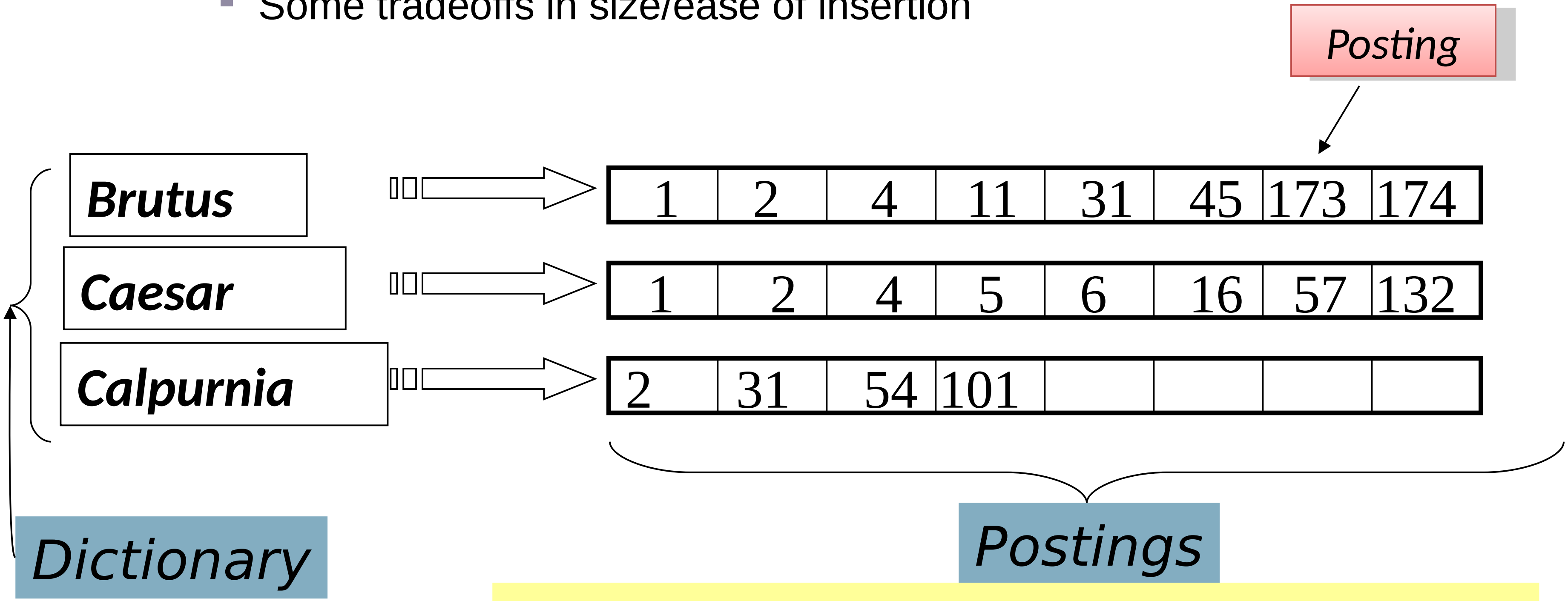
Brutus	1	2	4	11	31	45	173	174
Caesar	1	2	4	5	6	16	57	132
Calpurnia	2	31	54	101				

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

Inverted index

We need variable-size postings lists

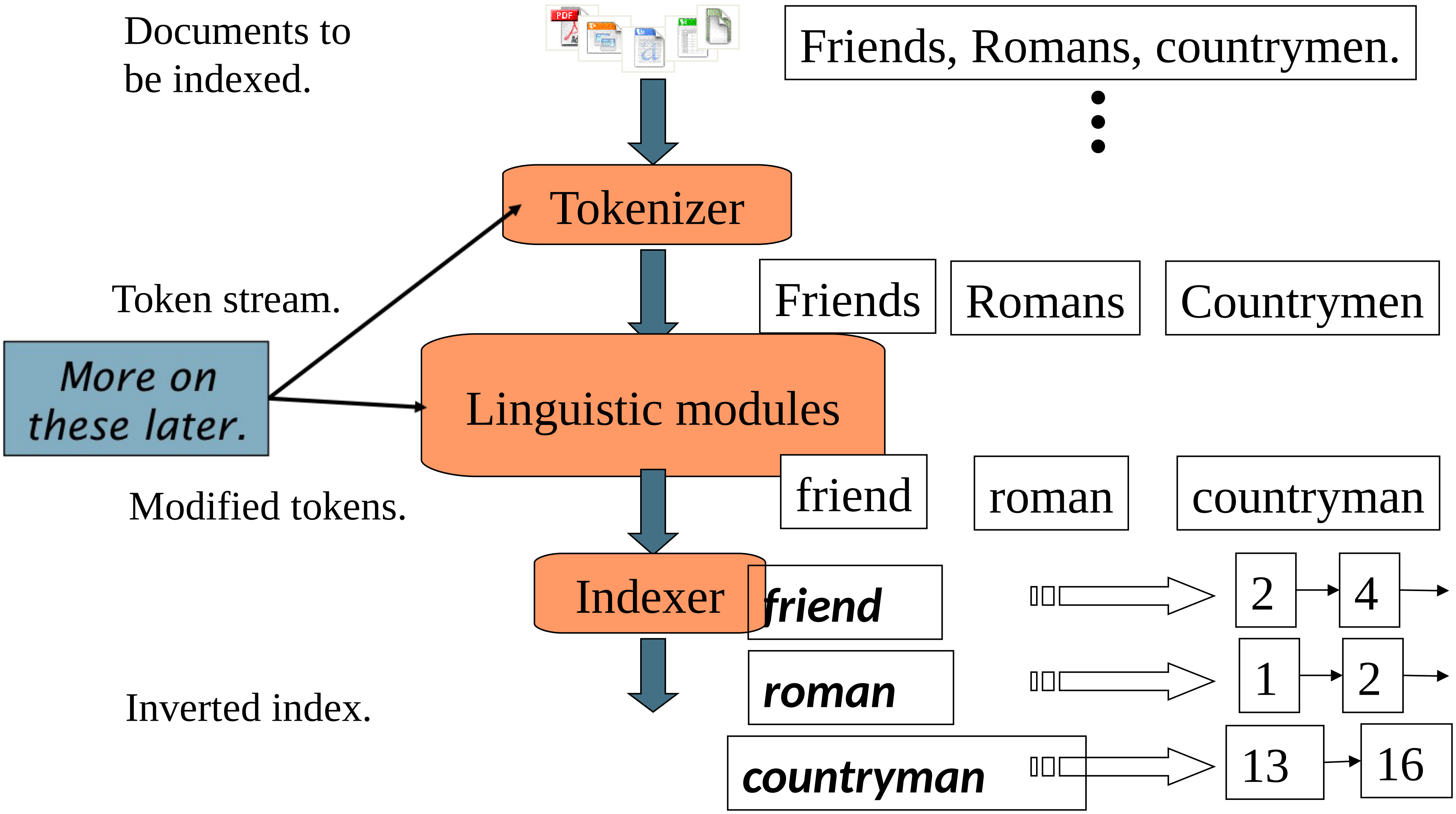
- On disk, a continuous run of postings is normal and best
- In memory, can use linked lists or variable length arrays
  - Some tradeoffs in size/ease of insertion



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

Sorted by docID (more later on why).

Inverted index construction



Documents to be indexed.

Friends, Romans, countrymen.

⋮

Tokenizer

Token stream.

Friends | Romans | Countrymen

More on these later.

Linguistic modules

Modified tokens.

friend | roman | countryman

Indexer

Inverted index.

friend → 2 → 4 →

roman → 1 → 2 →

countryman → 13 → 16

Indexer steps: Token sequence

Sequence of (Modified token, Document ID) pairs.

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

Indexer steps: Sort

Sort by terms

- And then docID

Core indexing step

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

Indexer steps: Dictionary & Postings

Multiple term entries in a single document are merged.

Split into Dictionary and Postings

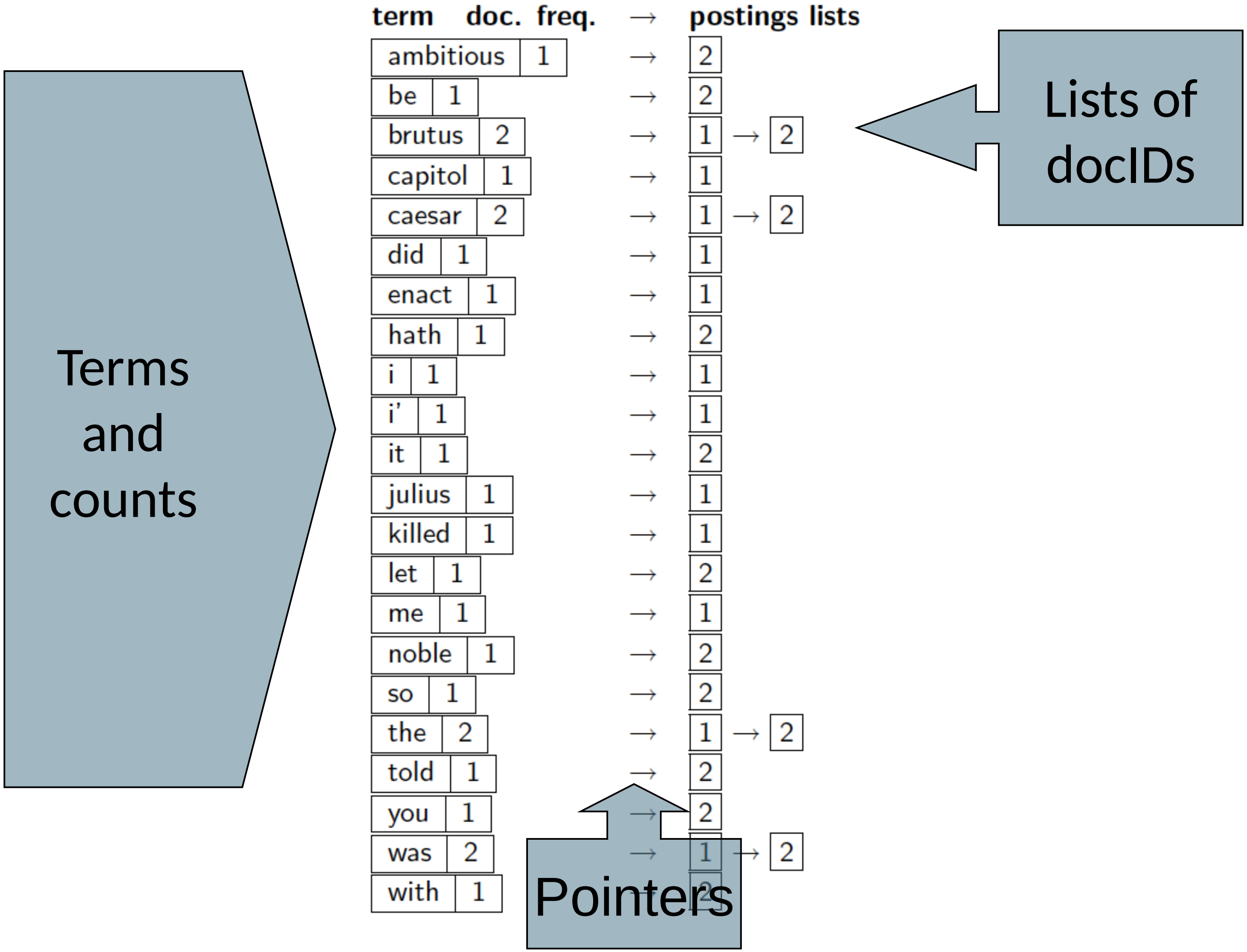
Doc. frequency information is added.

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

Why frequency?
Will discuss later.

Where do we pay in storage?



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

Terms and counts

Lists of docIDs

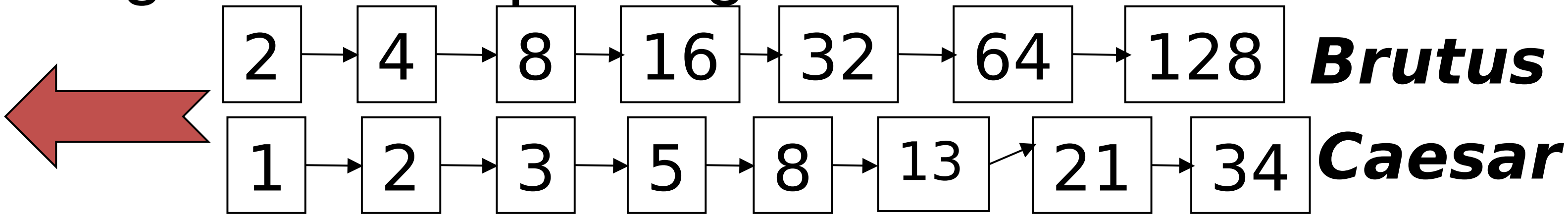
Pointers

Query processing: AND

Consider processing the query:

Brutus *AND* ***Caesar***

- Locate ***Brutus*** in the Dictionary;
  - Retrieve its postings.
- Locate ***Caesar*** in the Dictionary;
  - Retrieve its postings.
- "Merge" the two postings:

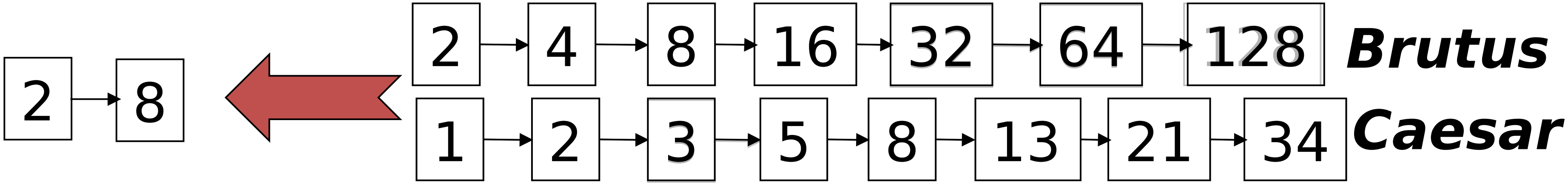


2 → 4 → 8 → 16 → 32 → 64 → 128 ***Brutus***

1 → 2 → 3 → 5 → 8 → 13 → 21 → 34 ***Caesar***

The merge

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



2 → 4 → 8 → 16 → 32 → 64 → 128 **_Brutus_**

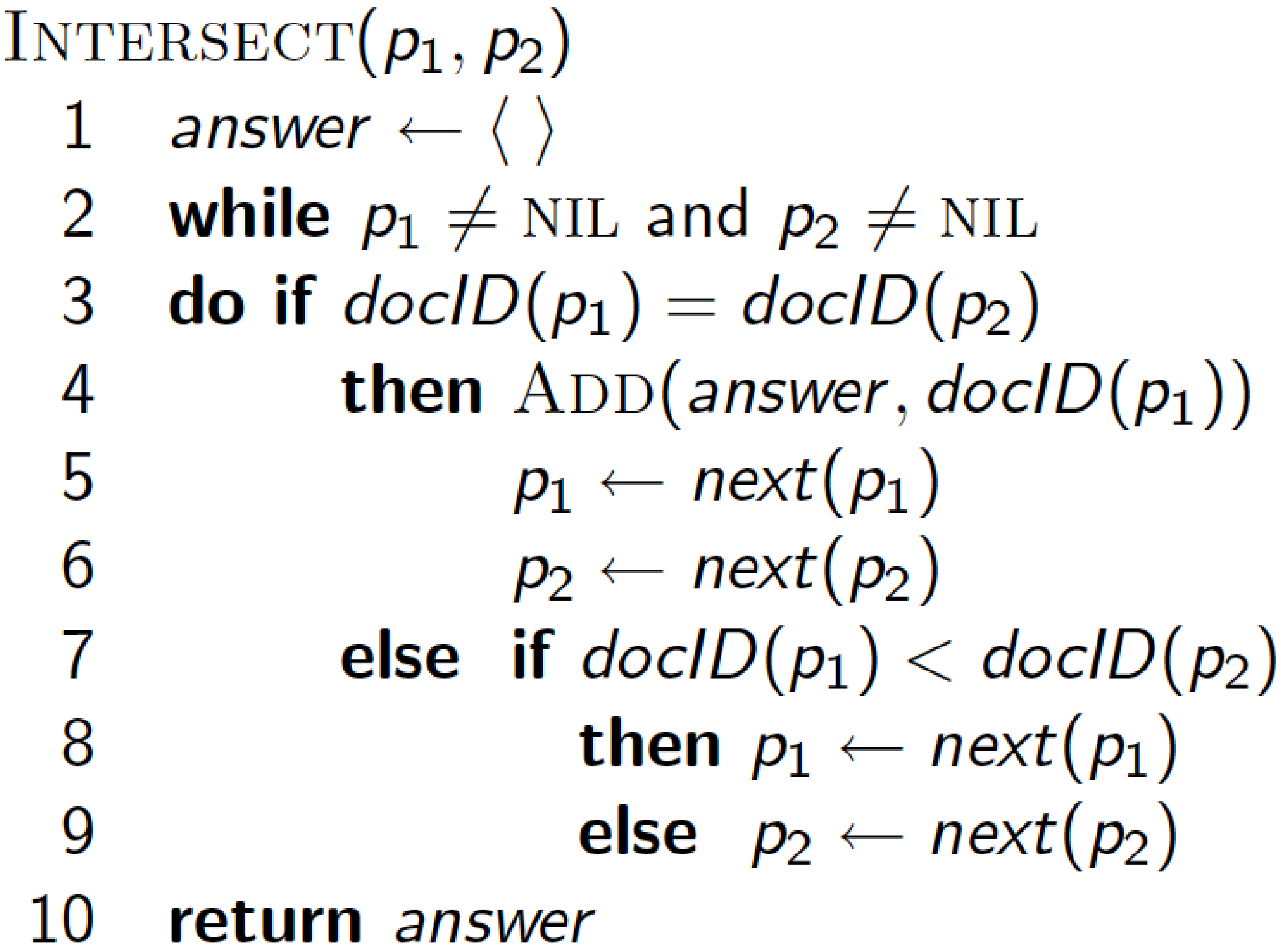
1 → 2 → 3 → 5 → 8 → 13 → 21 → 34 **_Caesar_**

2 → 8

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

Crucial: postings sorted by docID.

Intersecting two postings lists (a “merge” algorithm)



```
INTERSECT($p_1, p_2$)
  $answer \leftarrow \langle\ \rangle$
  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 $p_1 \leftarrow next(p_1)$
               else $p_2 \leftarrow next(p_2)$
  return $answer$
```

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
    - Is precise: document matches condition or not.
  - Perhaps the simplest model to build an IR system on
- Primary commercial retrieval tool for 3 decades!
- Many search systems you still use are Boolean:
  - Email, library catalog, Mac OS X Spotlight

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

- Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992)
- Tens of terabytes of data; 700,000 users
- Majority of users *still* use boolean queries
- Example query:
  - What is the statute of limitations in cases involving the federal tort claims act?
  - LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM
    - /3 = within 3 words, /S = in same sentence

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

- Another example query:
  - Requirements for disabled people to be able to access a workplace
  - disabl! /p access! /s work-site work-place (employment /3 place)
- Note that SPACE is disjunction, not conjunction!
- 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
- But that doesn’t mean it actually works better….

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)?
- What can we achieve?

Merging

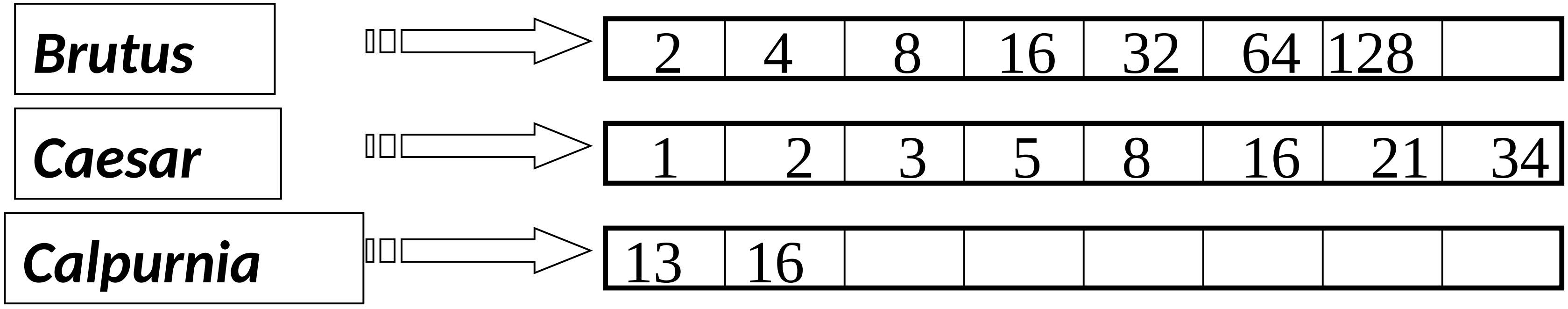
- What about an arbitrary Boolean formula?

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

  (***Antony*** *OR* ***Cleopatra***)
- Can we always merge in “linear” time?
  - 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.



Brutus	2	4	8	16	32	64	128	
Caesar	1	2	3	5	8	16	21	34
Calpurnia	13	16						

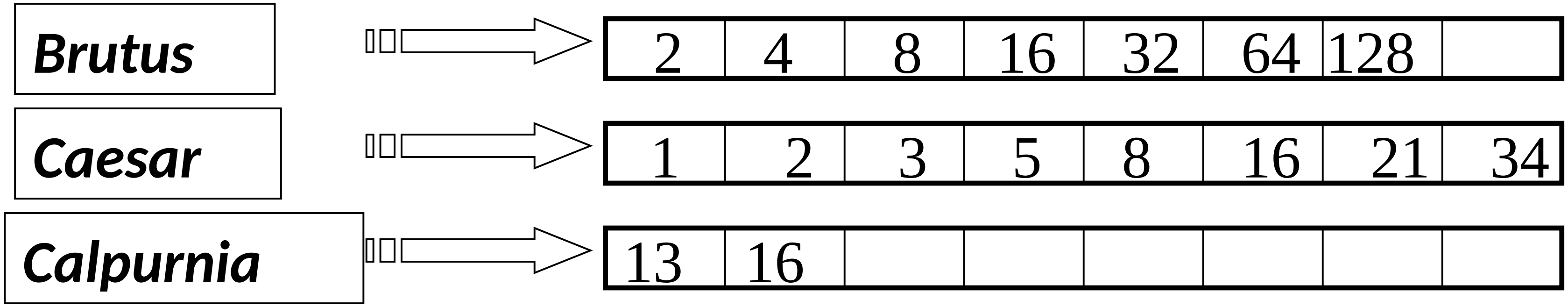
Query: ***Brutus*** *AND* ***Calpurnia*** *AND* ***Caesar***

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



Brutus → 2 4 8 16 32 64 128

Caesar → 1 2 3 5 8 16 21 34

Calpurnia → 13 16

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

More general optimization

- e.g., *(**madding** OR **crowd**) AND (**ignoble** OR **strife**)*
- Get doc. freq.’s 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.

Exercise

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**

What's ahead in IR? Beyond term search

- What about **phrases**?
  - ***Stanford University***
- Proximity: Find ***Gates*** *NEAR* ***Microsoft***.
  - Need index to capture **position** information in docs.
- **Zones** in documents: Find documents with (*author* = ***Ullman***) *AND* (text contains ***automata***).

Evidence accumulation

- 1 vs. 0 occurrence of a search term
    - 2 vs. 1 occurrence
    - 3 vs. 2 occurrences, etc.
    - Usually more seems better ...
- Need term **frequency** information in docs

Ranking search results

- Boolean queries give inclusion or exclusion of docs.
- Often we want to rank/group results
  - Need to measure **proximity** from query to each doc.
  - Need to decide whether docs presented to user are singletons, or a group of docs covering various aspects of the query.

Basic Techniques for Web Search

- Review of applications
- Basic Techniques in detail:
    - Boolean search
    - Vocabularies, dictionaries, index
    - Scoring, complete system, evaluation
    - Web search
- Semantic search

Web Search:

Techniques, algorithms and Aplications

Basic Techniques
for Web Search

German Rigau <german.rigau@ehu.es>

[Based on slides by Eneko Agirre …
and Christopher Manning and Prabhakar Raghavan]