



Inspire...Educate...Transform.

2. Text Indexing and Crawling

Dr. Manish Gupta

Sr. Mentor – Academics, INSOF

Adapted from <https://web.stanford.edu/class/cs276a/handouts/lecture1.ppt>
<http://www.cs.uoi.gr/~pitoura/courses/ap/ap13/slides/lecture11-crawl.pptx>

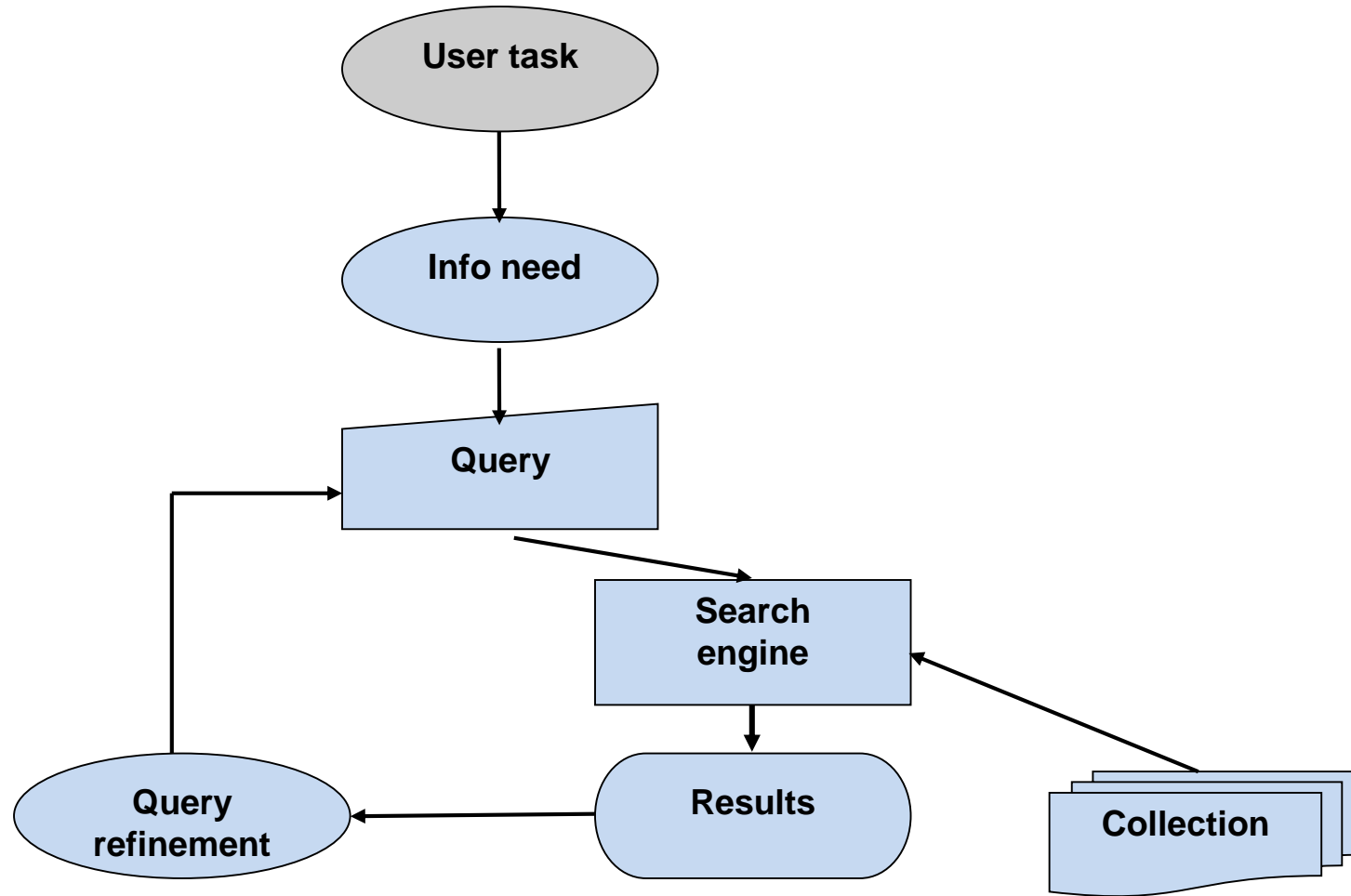
Course Content

- Collection of three main topics of high recent interest.
 - Search engines (Crawling, Indexing, Ranking)
 - Language Modeling
 - **Text Indexing and Crawling**
 - Relevance Ranking
 - Link Analysis Algorithms
 - Text Processing (NLP, NER, Sentiments)
 - Natural Language Processing
 - Named Entity Recognition
 - Sentiment Analysis
 - Summarization
 - Social networks (Properties, Influence Propagation)
 - Social Network Analysis
 - Influence Propagation in Social Networks

Today's Agenda

- Text Indexing

The Classic Search Model



Query on Unstructured Data

- 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 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

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**

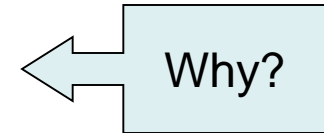
	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

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.

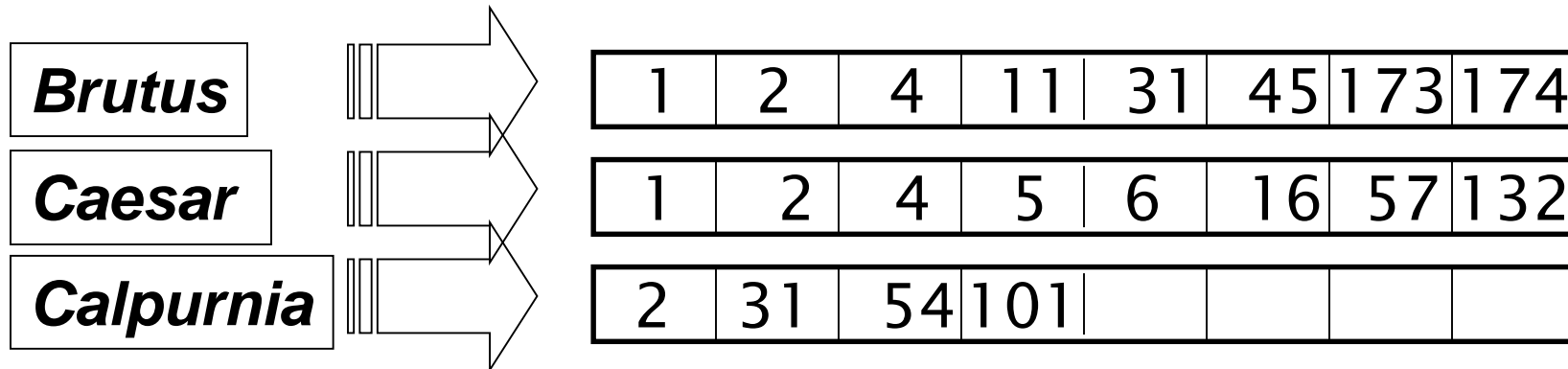
Can't Build the Matrix

- 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.



Inverted Index

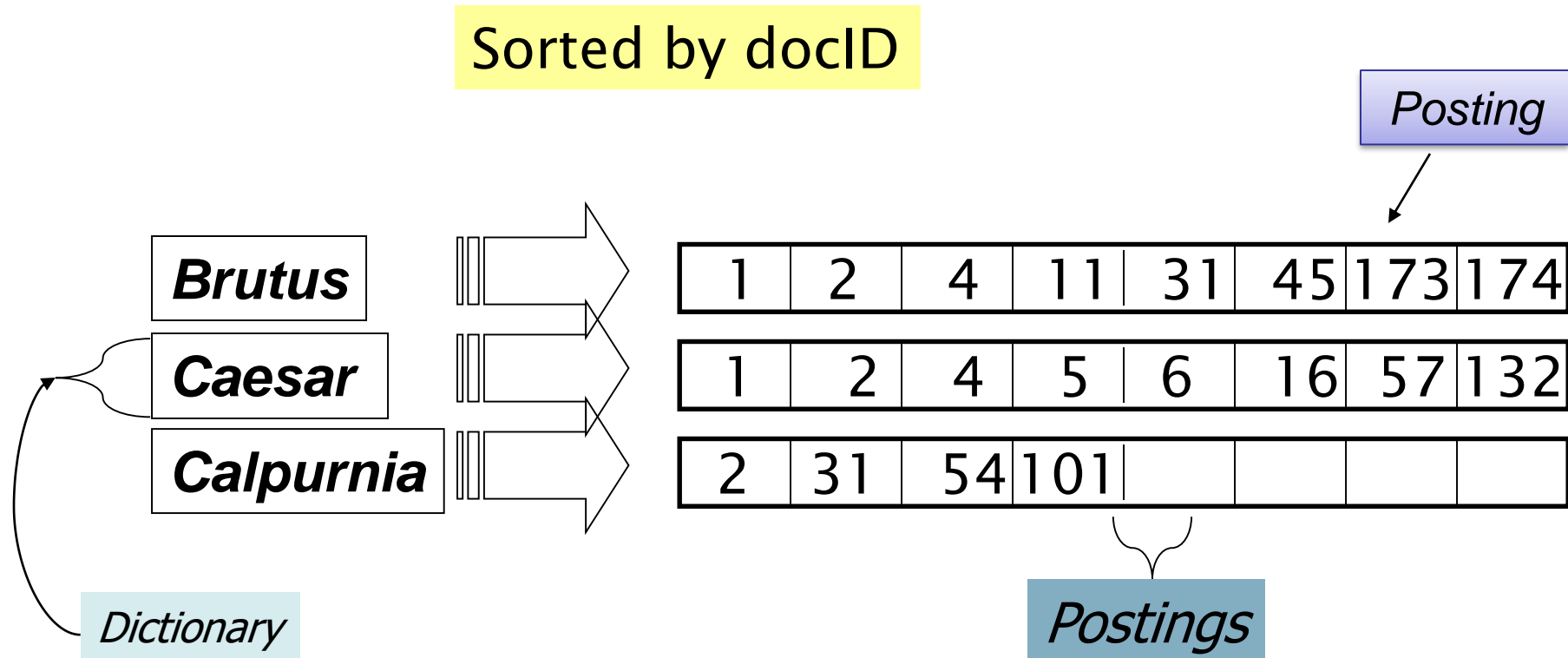
- For each term t , we must store a list of all documents that contain t .
 - Identify each doc by a **docID**, a document serial number
- Can we use fixed-size arrays for this?



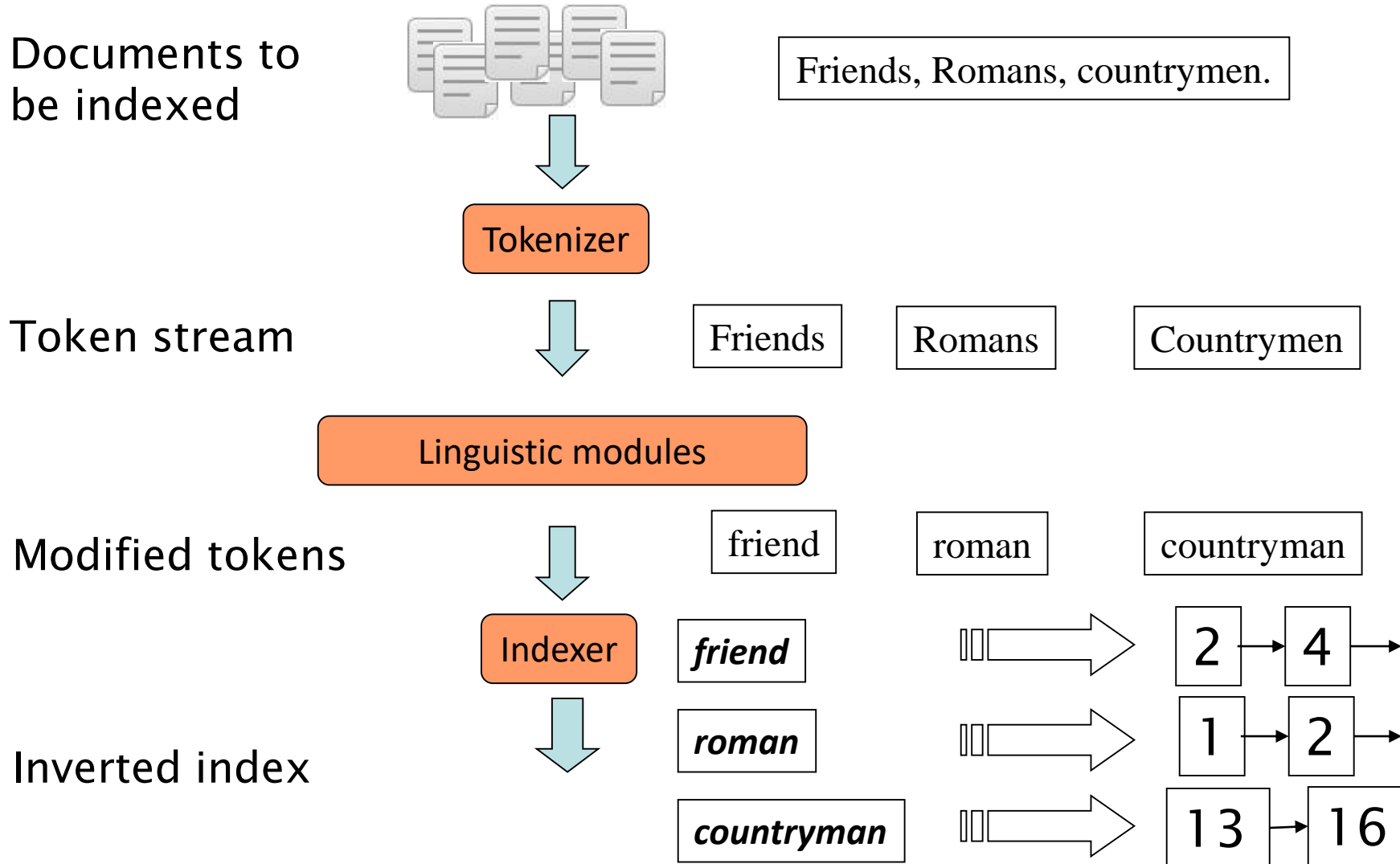
What happens if the word **Caesar** is added to document 14?

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



Inverted Index Construction



Initial Stages of Text Processing

- Tokenization
 - Cut character sequence into word tokens
 - Deal with *“John’s”, a state-of-the-art solution*
- Normalization
 - Map text and query term to same form
 - You want **U.S.A.** and **USA** to match
- Stemming
 - We may wish different forms of a root to match
 - *authorize, authorization*
- Stop words
 - We may omit very common words (or not)
 - *the, a, to, of*

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
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



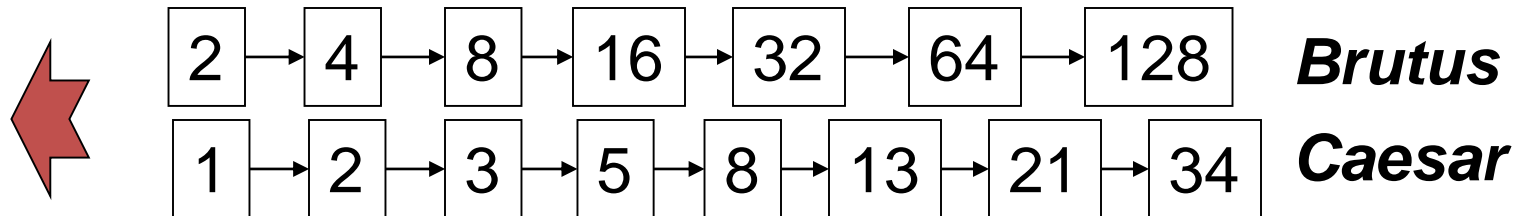
term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

Query Processing: AND

- Consider processing the query:

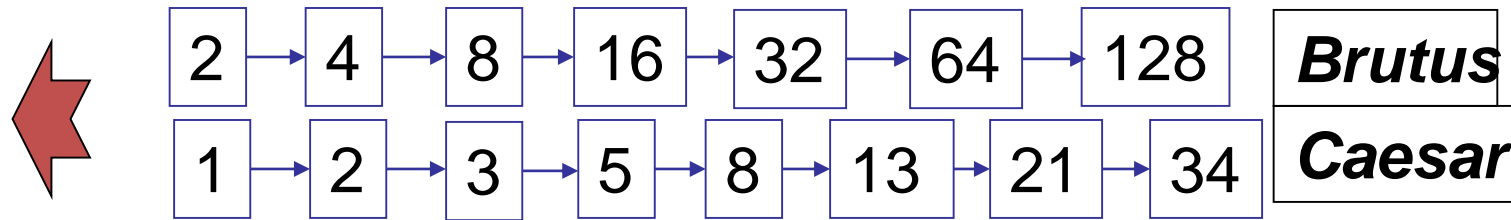
Brutus AND Caesar

- Locate ***Brutus*** in the Dictionary;
 - Retrieve its postings.
- Locate ***Caesar*** in the Dictionary;
 - Retrieve its postings.
- “Merge” the two postings (intersect the document sets):



The Merge

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



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

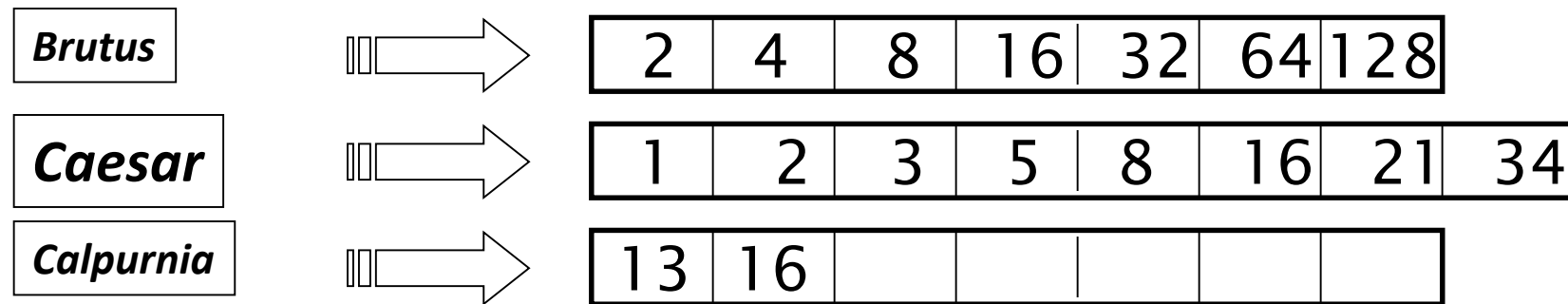
Crucial: postings sorted by docID.

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

Query Optimization

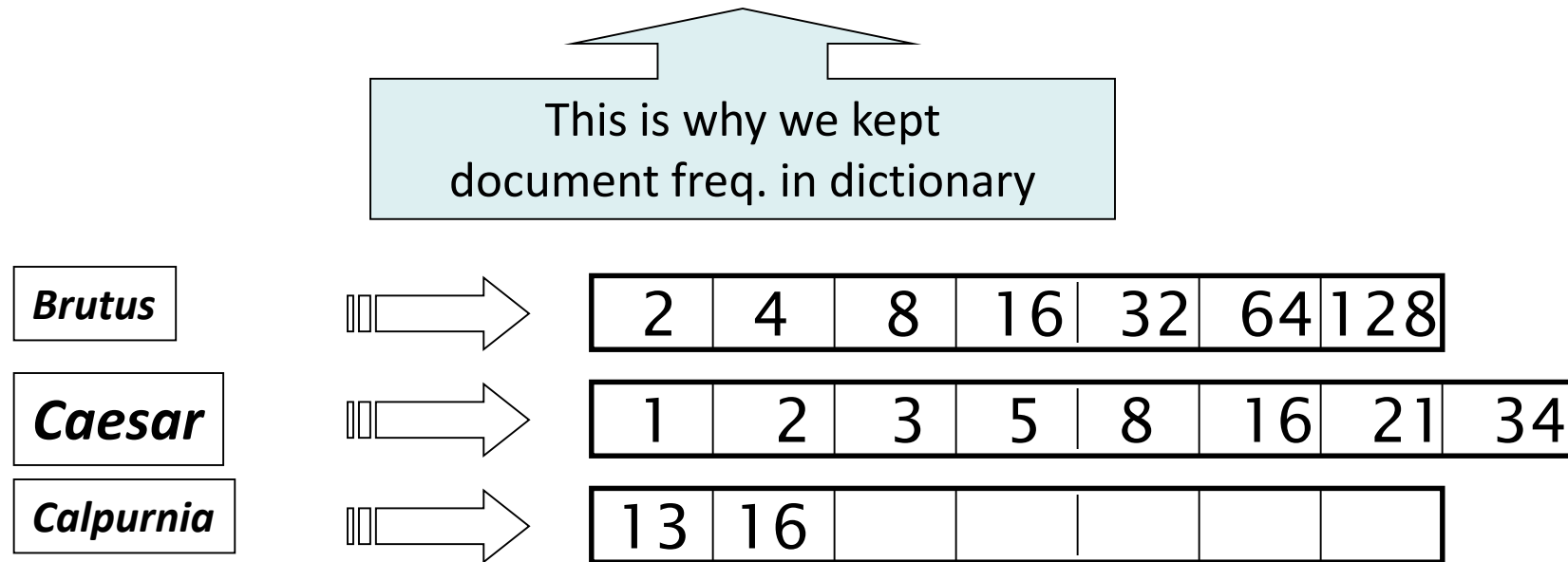
- What is the best order for query processing?
- Consider a query that is an *AND* of n terms.
- For each of the n terms, get its postings, then *AND* them together.



Query: **Brutus AND Calpurnia AND Caesar**

Query Optimization Example

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



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

Phrase Queries

- We want to be able to answer queries such as “*stanford university*” – as a phrase
- Thus the sentence “*I went to university at Stanford*” is not a match.
 - The concept of phrase queries has proven easily understood by users; one of the few “advanced search” ideas that works
 - Many more queries are *implicit phrase queries*
- For this, it no longer suffices to store only *<term : docs>* entries

A First Attempt: Bi-word Indexes


- Index every consecutive pair of terms in the text as a phrase
- For example the text “Friends, Romans, Countrymen” would generate the biwords
 - *friends romans*
 - *romans countrymen*
- Each of these bi-words is now a dictionary term
- Two-word phrase query-processing is now immediate.

Longer Phrase Queries

- Longer phrases can be processed by breaking them down
- ***stanford university palo alto*** can be broken into the Boolean query on biwords:

stanford university AND university palo AND palo alto

Without the docs, we cannot verify that the docs matching the above Boolean query do contain the phrase.



Can have false positives!

Issues for Bi-word Indexes

- False positives, as noted before
- Index blowup due to bigger dictionary
 - Infeasible for more than bi-words, big even for them
- Bi-word indexes are not the standard solution (for all bi-words) but can be part of a compound strategy

Solution 2: Positional Indexes

- In the postings, store, for each ***term*** the position(s) in which tokens of it appear

<***term***, number of docs containing ***term***;

doc1: position1, position2 ... ;

doc2: position1, position2 ... ;

etc.>

Positional Index Example

- For phrase queries, we use a merge algorithm recursively at the document level
- But we now need to deal with more than just equality

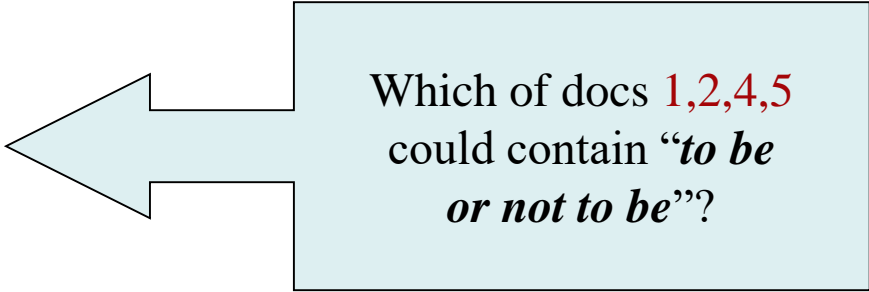
<*be*: 993427;

1: 7, 18, 33, 72, 86, 231;

2: 3, 149;

4: 17, 191, 291, 430, 434;

5: 363, 367, ...>



Which of docs *1,2,4,5*
could contain “*to be*
or not to be”?

Processing a Phrase Query

- Extract inverted index entries for each distinct term: ***to, be, or, not.***
- Merge their *doc:position* lists to enumerate all positions with “***to be or not to be***”.
 - ***to:***
 - 2:1,17,74,222,551; 4:8,16,190,429,433; 7:13,23,191; ...
 - ***be:***
 - 1:17,19; 4:17,191,291,430,434; 5:14,19,101; ...

Positional Index Size

- A positional index expands postings storage *substantially* even though indices can be compressed
- Need an entry for each occurrence, not just once per document
- Index size depends on average document size
 - Average web page has <1000 terms
 - SEC filings, books, even some epic poems ... easily 100,000 terms
- Consider a term with frequency 0.1%

Document size	Postings	Positional postings
1000	1	1
100,000	1	100

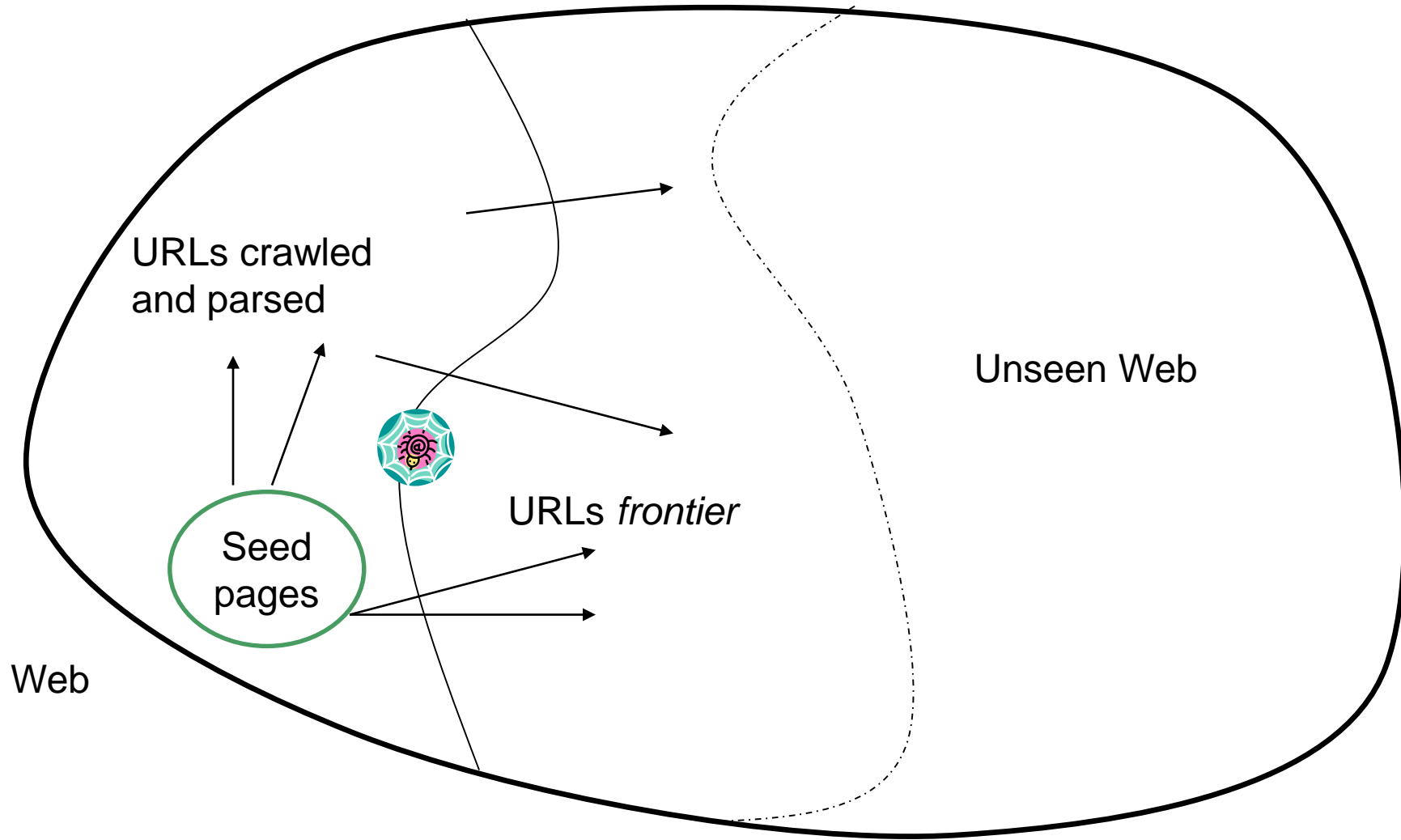
Today's Agenda

- Text Indexing
- Crawling

Basic Crawler Operation

- Begin with known “seed” pages
- Fetch and parse them
 - Extract URLs they point to
 - Place the extracted URLs on a queue
- Fetch each URL on the queue and repeat

Crawling Picture



Simple Picture – Complications

- Web crawling isn't feasible with one machine
 - All of the above steps distributed
- Even non-malicious pages pose challenges
 - Latency/bandwidth to remote servers vary
 - Webmasters' stipulations
 - How "deep" should you crawl a site's URL hierarchy?
 - Site mirrors and duplicate pages
- Malicious pages
 - Spam pages
 - Spider traps – including dynamically generated
- Politeness – don't hit a server too often

What any Crawler *must* do

- Be Polite: Respect implicit and explicit politeness considerations for a website
 - Only crawl pages you're allowed to
 - Respect *robots.txt*
 - Website announces its request on what can(not) be crawled
 - For a URL, create a file `URL/robots.txt`
- Be Robust: Be immune to spider traps and other malicious behavior from web servers

Robots.txt Example

- No robot should visit any URL starting with "/yoursite/temp/", except the robot called "searchengine":

```
User-agent: *
```

```
Disallow: /yoursite/temp/
```

```
User-agent: searchengine
```

```
Disallow:
```

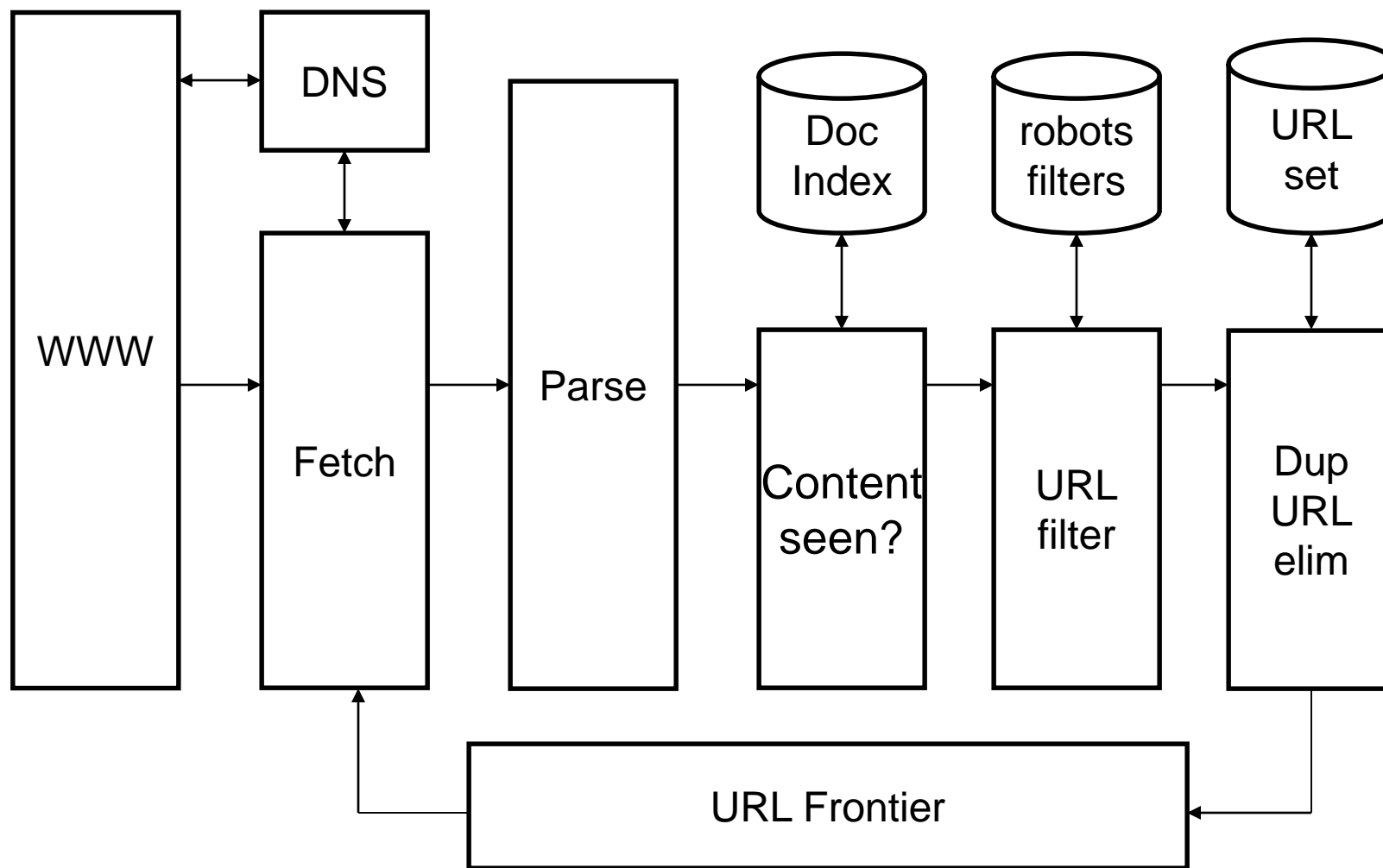
What any Crawler *should* do

- Be capable of distributed operation: designed to run on multiple distributed machines
- Be scalable: designed to increase the crawl rate by adding more machines
- Performance/efficiency: permit full use of available processing and network resources

What any Crawler *should* do

- Fetch pages of “higher quality” first
- Continuous operation: Continue fetching fresh copies of a previously fetched page
- Extensible: Adapt to new data formats, protocols

Basic Crawler Architecture



Take-away Messages

- Indexing
 - Binary Incidence Matrices
 - Inverted Index
 - Positional Inverted Index
- Crawling
 - Crawl as much as possible and as soon as possible
 - Always keep index fresh
 - Crawl high priority pages more frequently
 - Politeness

Further Reading

- Chapters 1,2,5 of [Manning-Raghavan-Schuetze book](http://nlp.stanford.edu/IR-book/)
 - <http://nlp.stanford.edu/IR-book/>
- Chapter 3 (Web Search and Information Retrieval) from [Mining the Web](http://www.cse.iitb.ac.in/soumen/mining-the-web/)
 - <http://www.cse.iitb.ac.in/soumen/mining-the-web/>
- [As We May Think -- Vannevar Bush](http://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881/)
 - <http://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881/>

International School of Engineering

Plot 63/A, 1st Floor, Road # 13, Film Nagar, Jubilee Hills, Hyderabad - 500 033

For Individuals: +91-9502334561/63 or 040-65743991

For Corporates: +91-9618483483

Web: <http://www.insofe.edu.in>

Facebook: <https://www.facebook.com/insofe>

Twitter: <https://twitter.com/Insofeedu>

YouTube: <http://www.youtube.com/InsofeVideos>

SlideShare: <http://www.slideshare.net/INSOFE>

LinkedIn: <http://www.linkedin.com/company/international-school-of-engineering>