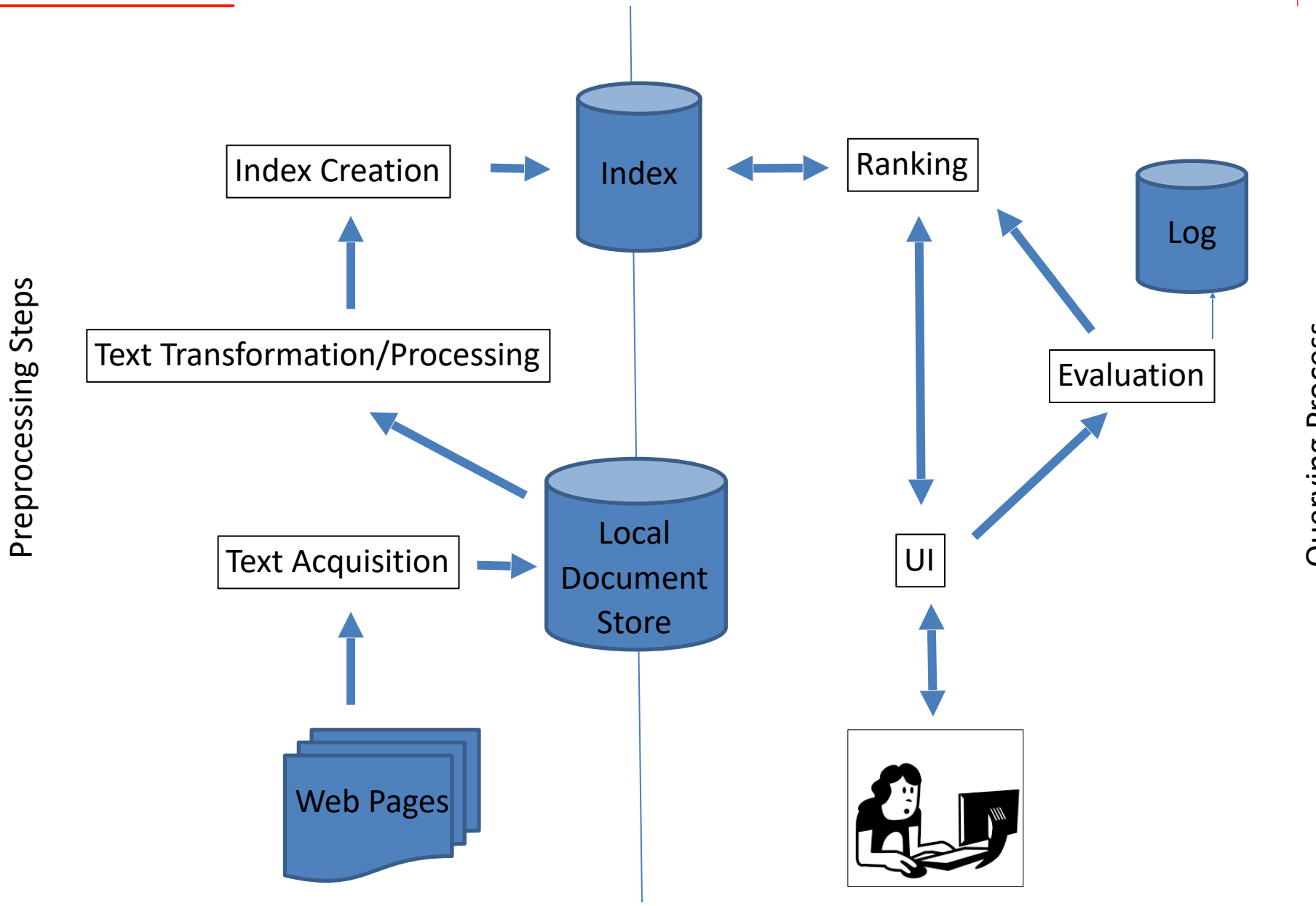
Informatics 225
Computer Science 221

Information Retrieval

Lecture 15

These course materials borrow, with permission, from those of Prof. Cristina Videira Lopes, Addison Wesley 2008, Chris Manning, Pandu Nayak, Hinrich Schütze, Heike Adel, Sascha Rothe, Jerome H. Friedman, Robert Tibshirani, and Trevor Hastie. Powerpoint theme by Prof. André van der Hoek.

Architecture



Stemming: Revisiting Error metrics

- False positive : Over-stemming
  - When words with different stems are stemmed to the same root
  - e.g. Suffix-s: **Assumption** —> if a word ends with `s` , probably plural
    - cats —> cat or lakes —> lake
    - Error: *ups* —> *up*
  - e.g. Error: *universal/university/universe* —> *universe*

- False negative: Under-stemming
  - When words that should have been but are not stemmed to same root
  - Error: *alumnus/alumni/alumnae* —> should have been 'alumni' but results in *alumnus/alumni/alumna*

Term-document incidence matrix

Information Retrieval

These course materials borrow, with permission, from those of Prof. Cristina Videira Lopes, Addison Wesley 2008, Chris Manning, Pandu Nayak, Hinrich Schütze, Heike Adel, Sascha Rothe, Jerome H. Friedman, Robert Tibshirani, and Trevor Hastie

Unstructured data in 1620

- 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*) is not feasible
  - Ranked retrieval is not possible
    - Users want ordered documents

Term-document incidence matrices

- Term-document incidence matrices : Boolean matrix indicating if a term (rows) exists in a certain document (columns).

	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

How to answer the query:

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

Incidence vectors

- Each term has a 0/1 vector (rows of the matrix)

- To answer query: take the rows for ***Brutus, Caesar*** and ***Calpurnia*** (complemented) ➔ bitwise *AND*.

$$\mathcal{R} = \mathcal{I}_{Bru} \wedge_{bitwise} \mathcal{I}_{Cae} \wedge_{bitwise} (\neg \mathcal{I}_{Cal})$$

	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

Incidence vectors

- Each term has a 0/1 vector (rows of the matrix)

- To answer query: take the rows for ***Brutus, Caesar*** and ***Calpurnia*** (complemented) ➔ bitwise *AND*.

  - (Brutus) 1 1 0 1 0 0 *AND*
  - (Caesar) 1 1 0 1 1 1 *AND*
  - (NOT Calpurnia) 1 0 1 1 1 1 =
  - **1 0 0 1 0 0**

	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

Incidence vectors

- Each term has a 0/1 vector (rows of the matrix)

- To answer query: take the rows for ***Brutus, Caesar*** and ***Calpurnia*** (complemented) ➔ bitwise *AND*.

  - (Brutus) 1 1 0 1 0 0 *AND*
  - (Caesar) 1 1 0 1 1 1 *AND*
  - (NOT Calpurnia) 1 0 1 1 1 1 =
  - **1 0 0 1 0 0**

	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

Problem

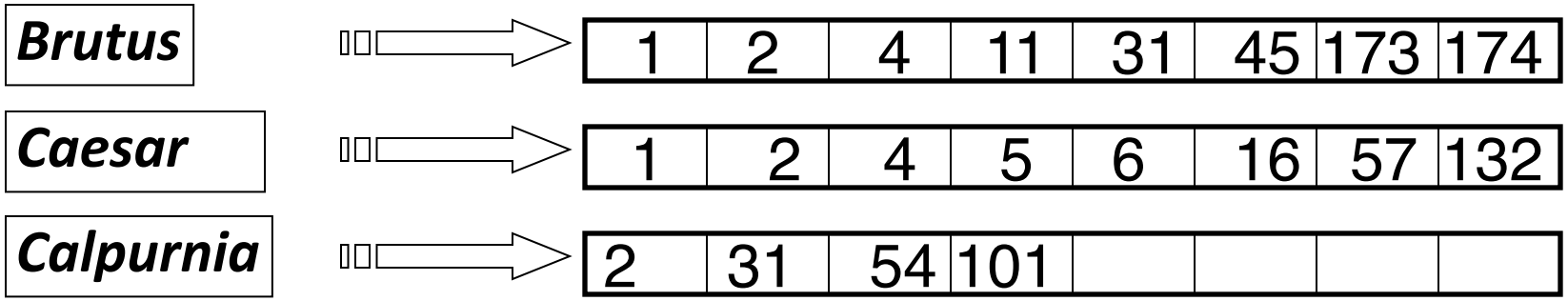
- Unfeasible for large number of documents because of the size of the resulting matrix
- But matrix is sparse
- **So….**

Inverted Index

Information Retrieval

Inverted index

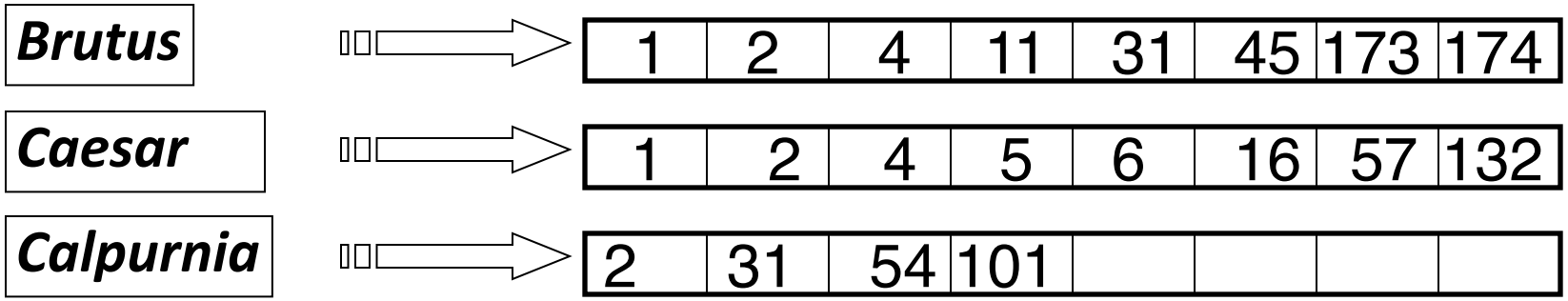
- For each term *t*, we must store a list of all documents that contain *t*.
  - Identify each doc by a **docID**, a document serial number



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

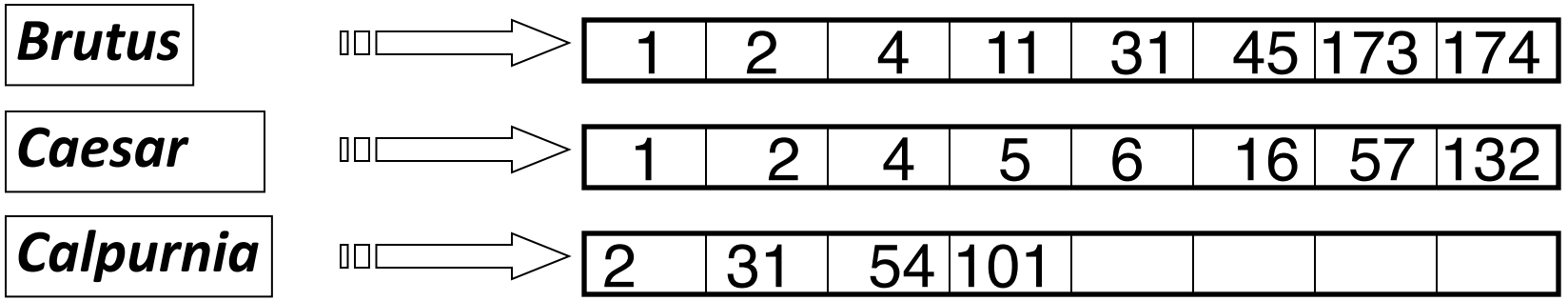
- Can we used fixed-size arrays for this?



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

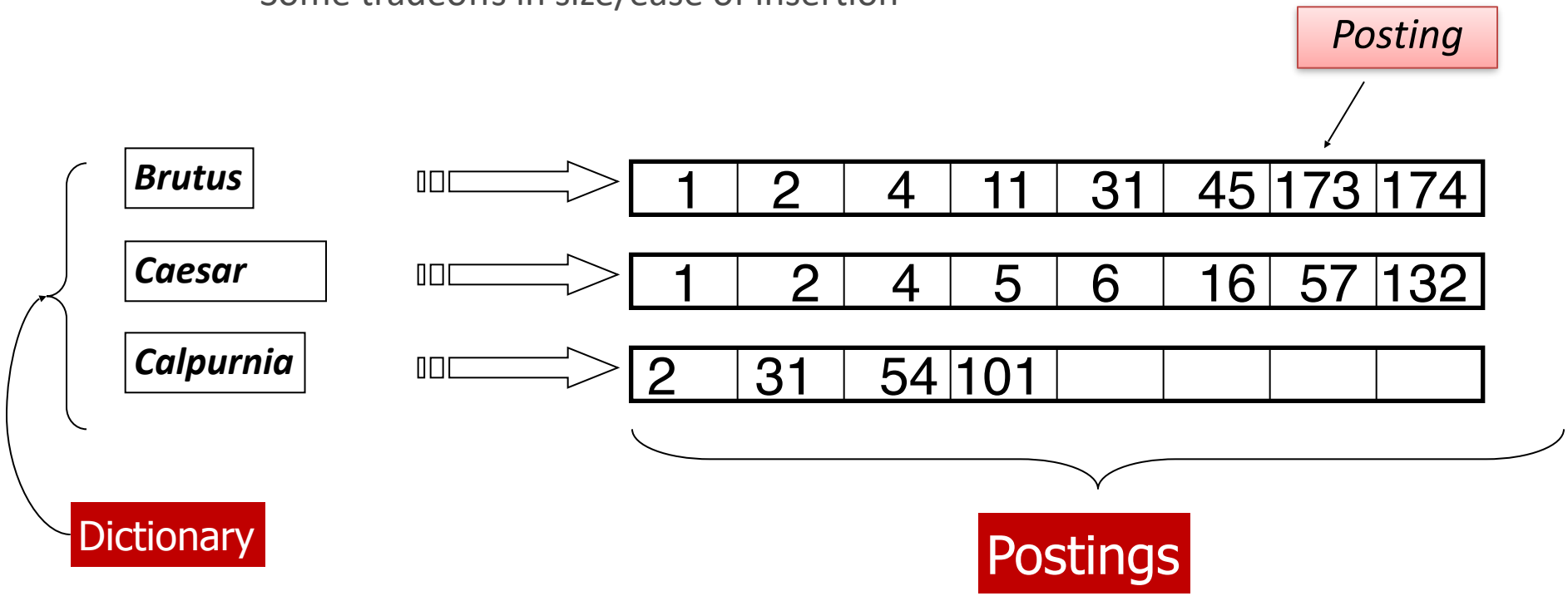
- Can we used fixed-size arrays for this?



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

Inverted index

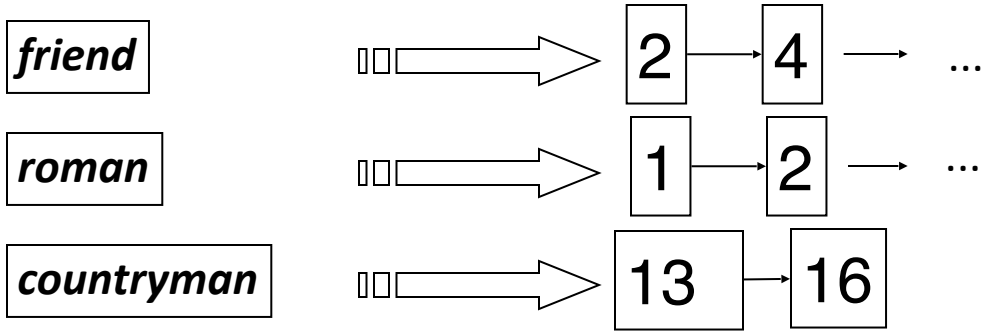
- We need variable-size postings lists
  - On disk, a continuous run of postings is the optimal solution
  - In memory, can use linked lists, variable length arrays, associative arrays
    - Some tradeoffs in size/ease of insertion



Sorted by docID (more later on why).

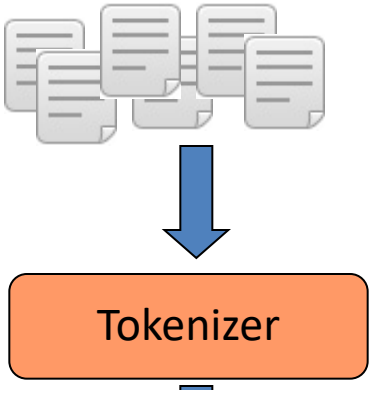
Inverted index

- Is normaly implemented as a map/dictionary with term as the key and the postings list as the value



Inverted index construction

Documents to be indexed



Friends, Romans, countrymen.

Inverted index construction

Documents to be indexed

Friends, Romans, countrymen.

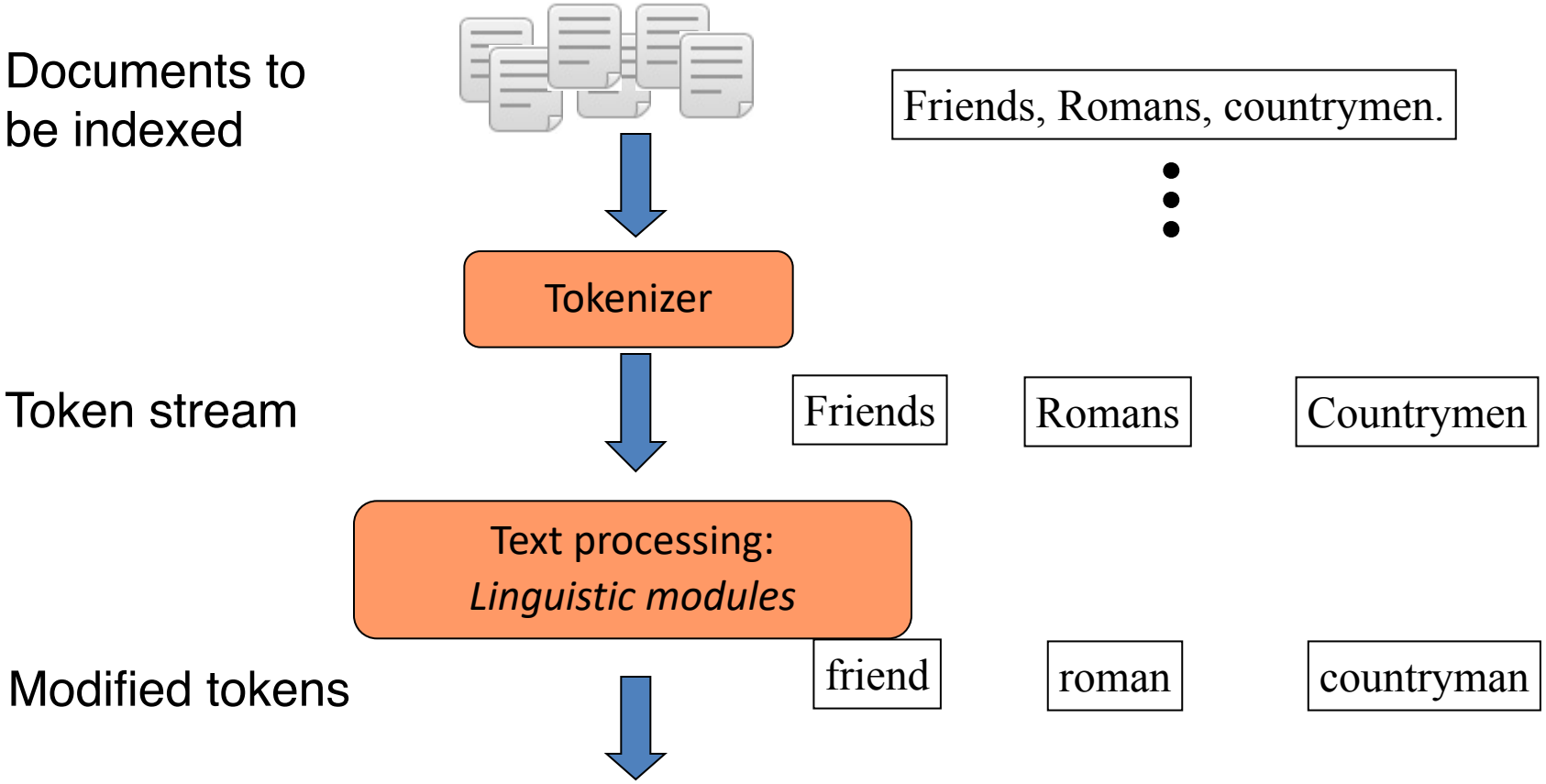
Tokenizer

Token stream

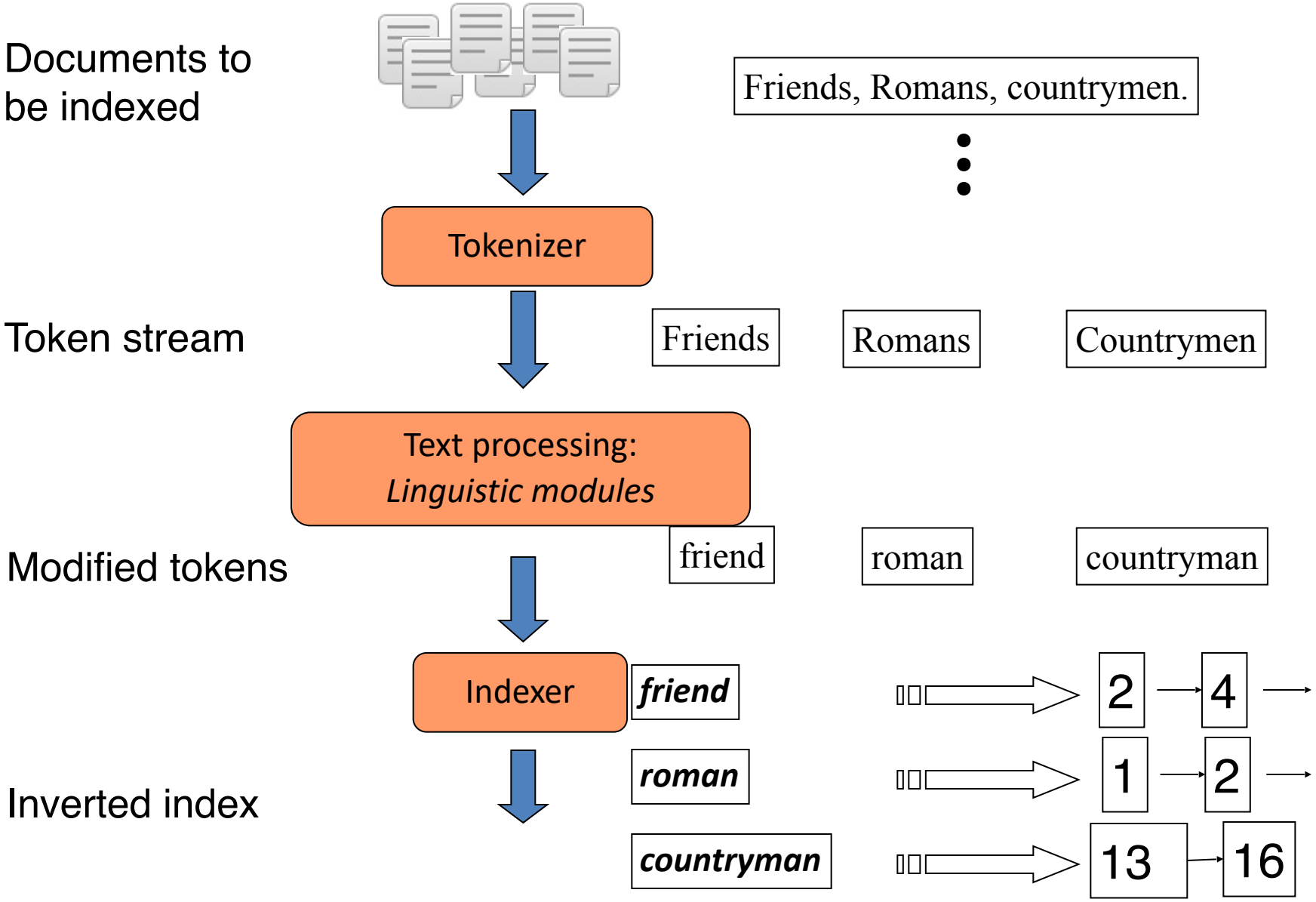
Friends | Romans | Countrymen

Text processing:
Linguistic modules

Inverted index construction



Inverted index construction



Typical minimal 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 (perhaps!)**
  - We **may** wish different forms of a root to match
    - ***authorize***, ***authorization***
- **Stop words (perhaps!)**
  - 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: Dictionary & Postings

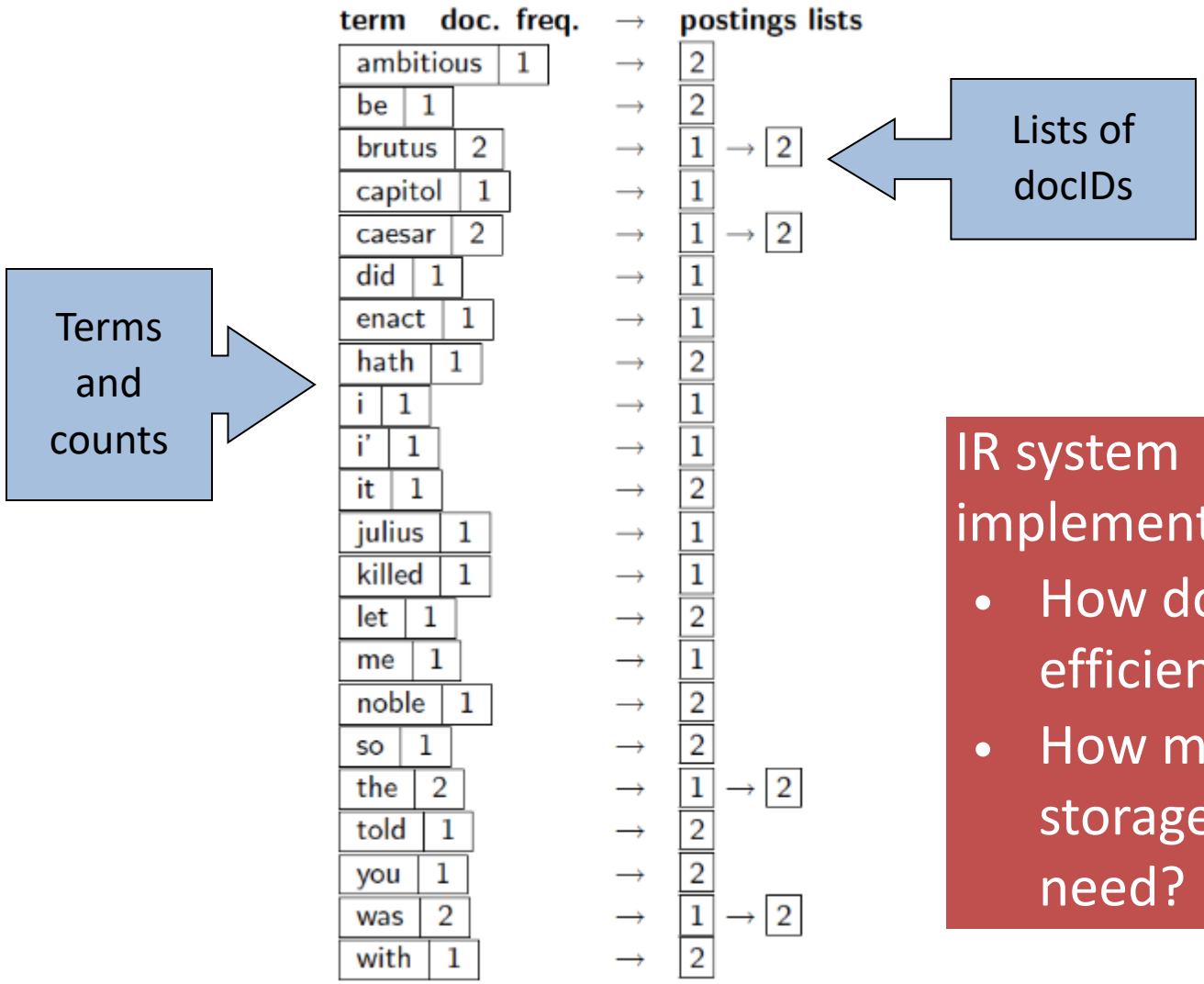
- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings
- Doc. frequency information is added.

Why frequency?
Will discuss later.

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

Where do we pay in storage?



IR system implementation

- How do we index efficiently?
- How much storage do we need?

Inverted index for small data

- If it fits in memory:
  - Python's dictionaries {}; Java's HashMap; ...
    - Fill it during indexing
    - Write it in a file
    - Load it back to dict/HashMap for search

Inverted index for small data

- If it fits in memory:
  - Python's dictionaries {}; Java's HashMap; ...
    - Fill it during indexing
    - Write it in a file
    - Load it back to dict/HashMap for search
- Very simple to implement…
  - ***But this will not work in any realistic web search engine***

Inverted index for large data

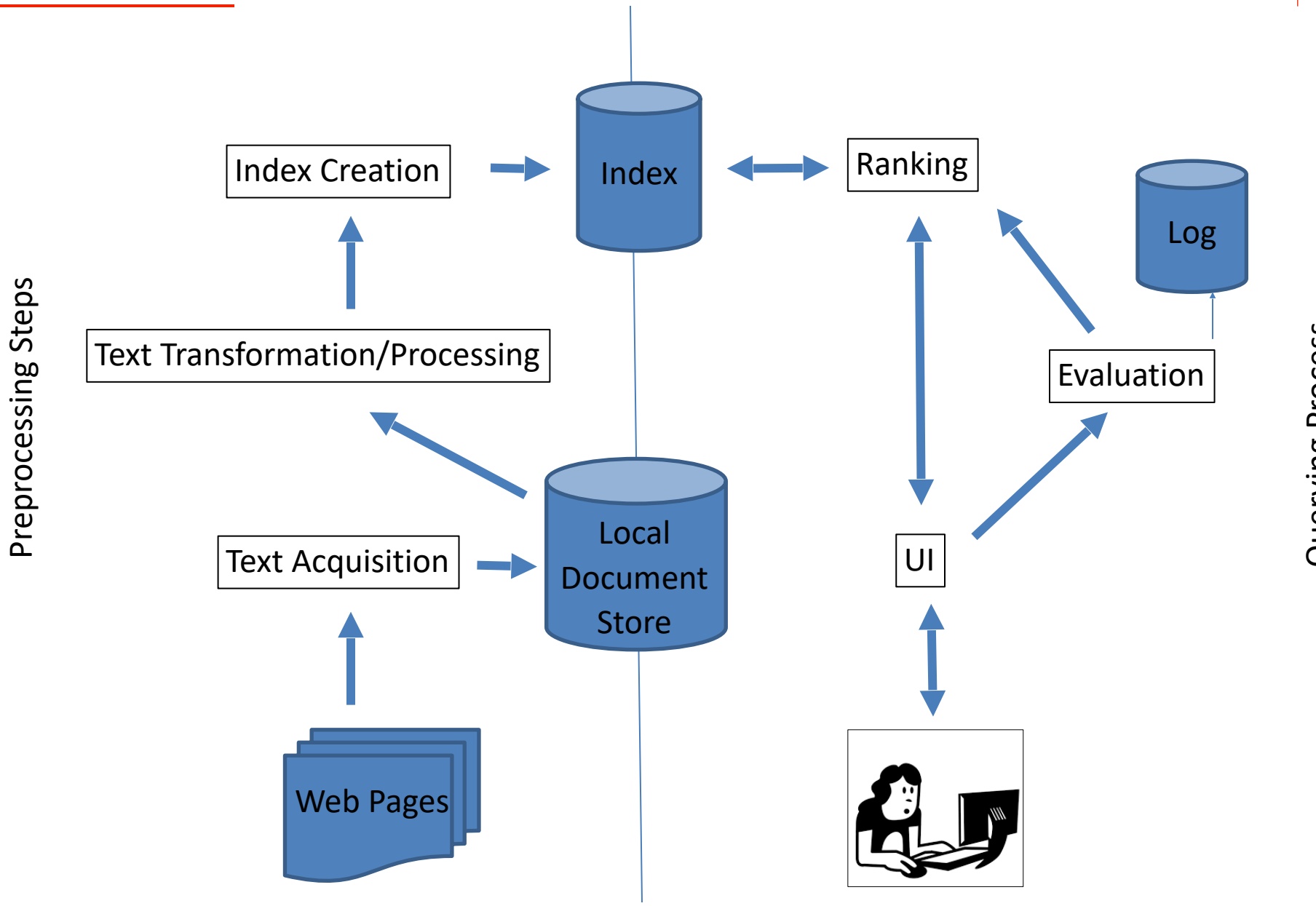
- Back-of-the-envelope calculations:
  - Vocabulary: 50,000 tokens
  - Average postings list size: 20
  - = 50,000 x 20 = 1,000,000 postings
  - Doc-id = URL, avg size: 50 characters ≈ 100 bytes (Python)
  - = 1,000,000 x 100 = 100,000,000 bytes = 100M
  - Plus payload in postings (e.g. tf-idf), 4 bytes each
  - = many more Mb
    - But the web comprises >> 10^6 documents (only hosts alone are > 10^9…)

- **Usually, index does not fit in memory all at once**
  - Out of memory errors

Query processing with inverted index

Information Retrieval

Architecture



The index we just built

- How do we process a query?
  - Later – what kinds of queries can we process?

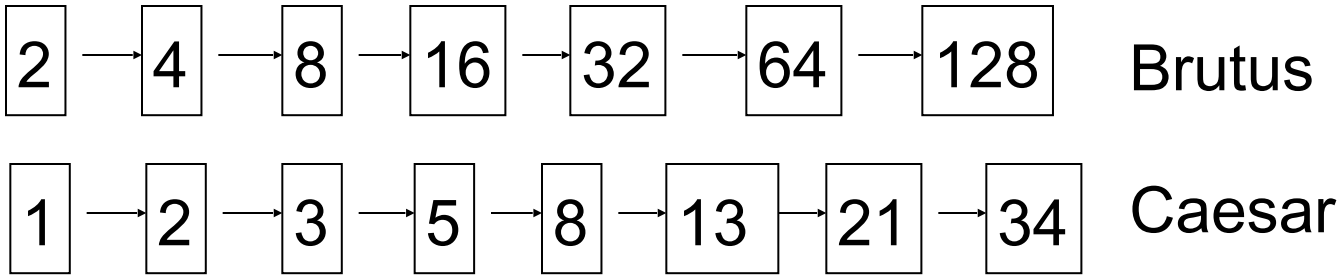
Query processing: AND

- Consider processing the query:

  Brutus** AND **Caesar

  – Locate *Brutus* in the Dictionary;
    - Retrieve its postings.

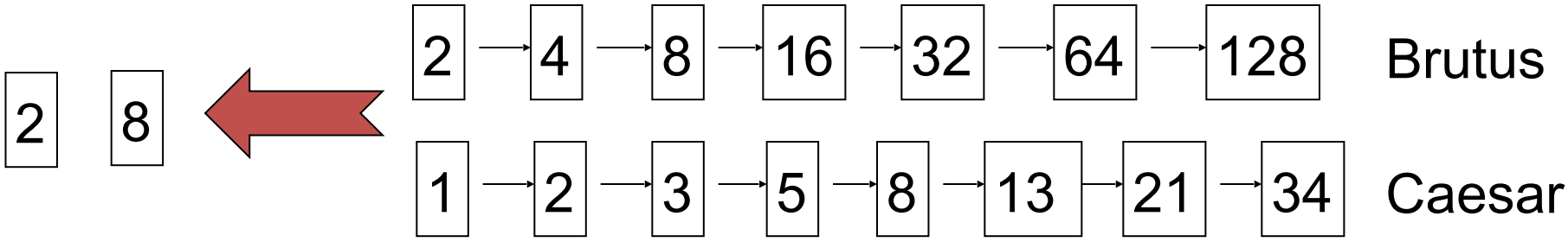
  – Locate *Caesar* in the Dictionary;
    - Retrieve its postings.



  – "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 that takes place at query time** takes O(*x*+*y*) operations.

Crucial: postings sorted by docID.

Simple algorithm to perform the "merge"

```
INTERSECT(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 p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

Boolean Retrieval Model

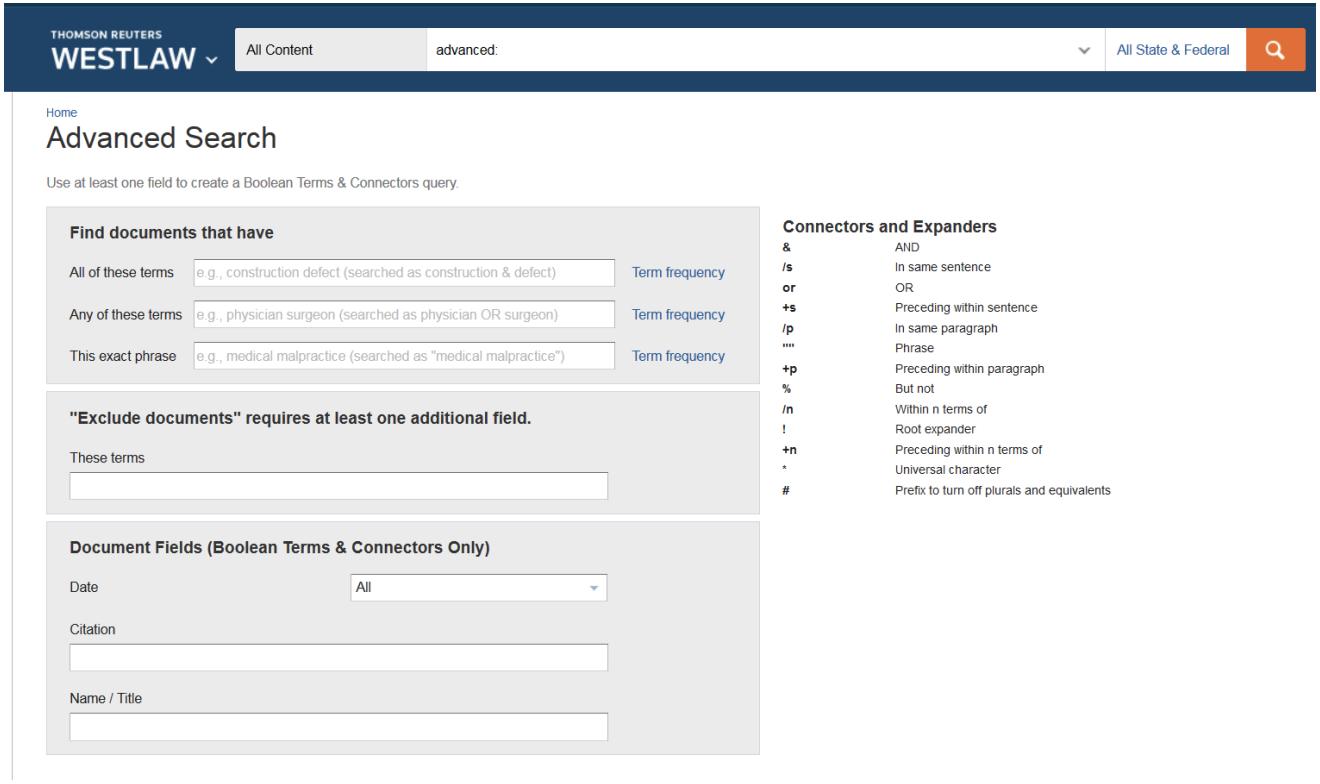
Information Retrieval

Boolean queries: Exact match

- Boolean retrieval model : queries are Boolean expressions
  - They use *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 are Boolean:
  - Email, library catalog, macOS Spotlight

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



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
  - In law, it is important to have "exact" results
- 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**

Indexes and Ranking

Information Retrieval

Indexes

- **Indexes** are **data structures** designed to **make search faster**
- Text search has unique requirements, which leads to unique data structures
- Most common data structure in text search is *inverted index*
  - general name for a class of structures
  - "inverted" because documents are associated with words, rather than words with documents
    - **similar to a *concordance***

Indexes and Ranking

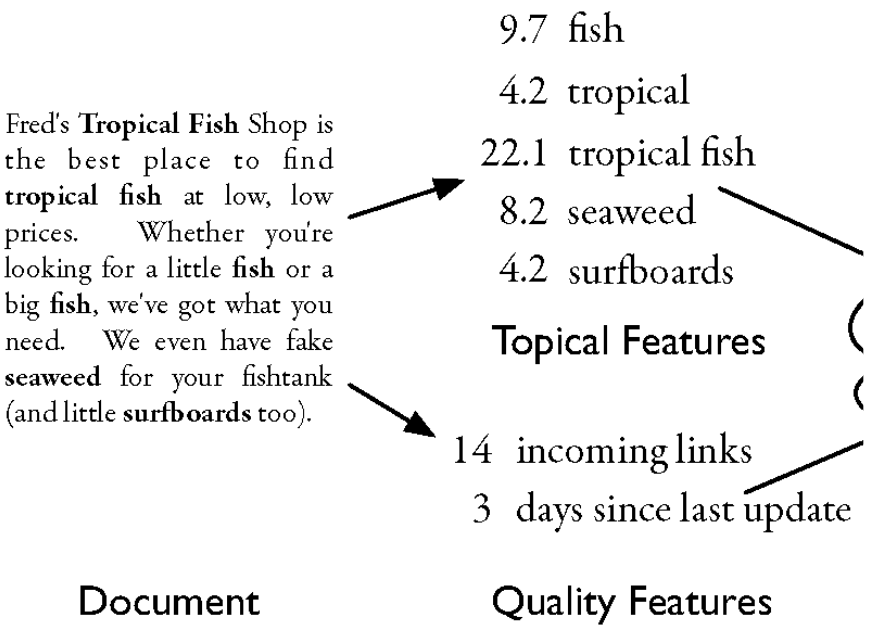
- Indexes are designed to support *search*
  - faster response time, supports updates
- Web search engines use a particular form of search: *ranking*
  - documents are retrieved in sorted order according to a score computing using the document representation, the query, and a *ranking algorithm*
- What is a reasonable abstract model for ranking?
  - enables discussion of indexes without details of retrieval model

Abstract Model of Ranking

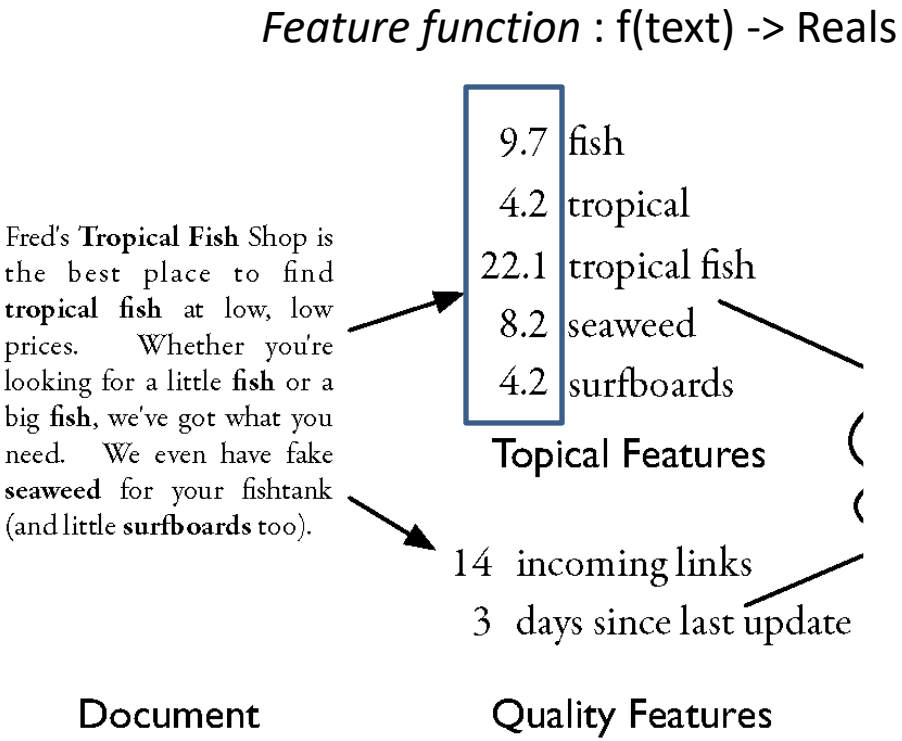
Fred's **Tropical Fish** Shop is the best place to find **tropical fish** at low, low prices. Whether you're looking for a little **fish** or a big **fish**, we've got what you need. We even have fake **seaweed** for your fishtank (and little **surfboards** too).

Document

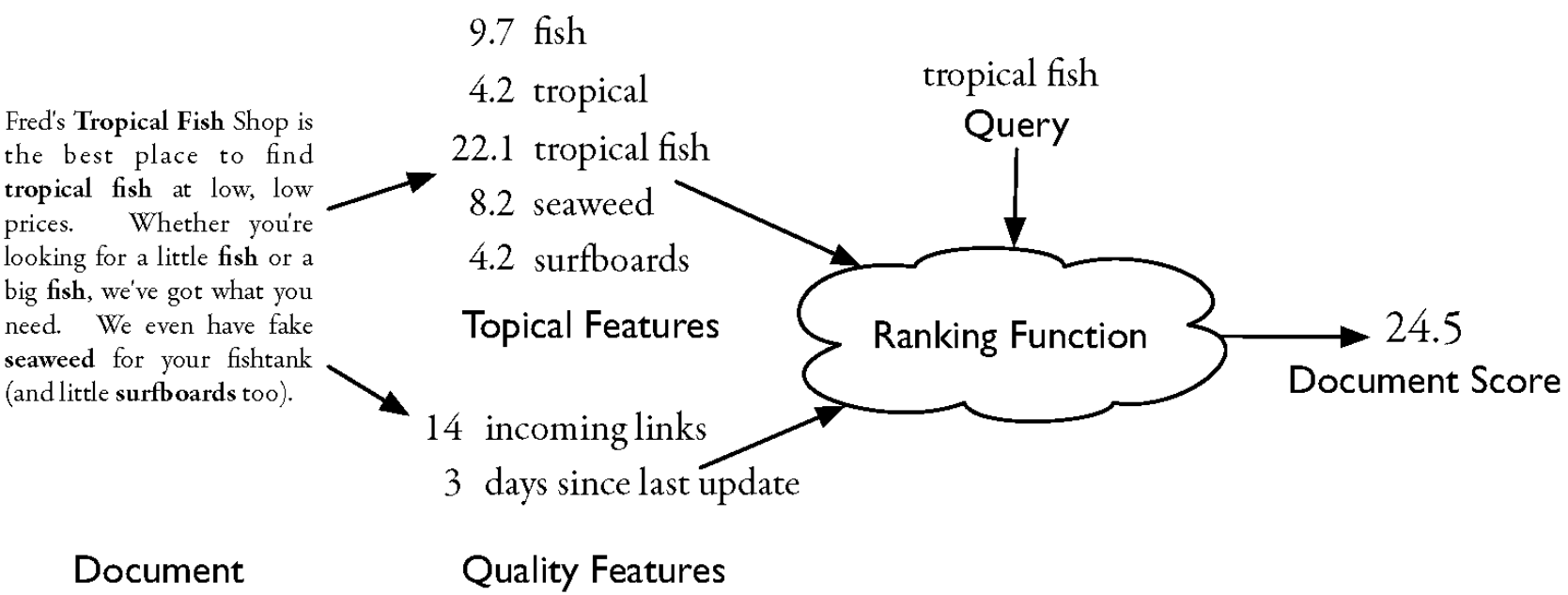
Abstract Model of Ranking



Abstract Model of Ranking



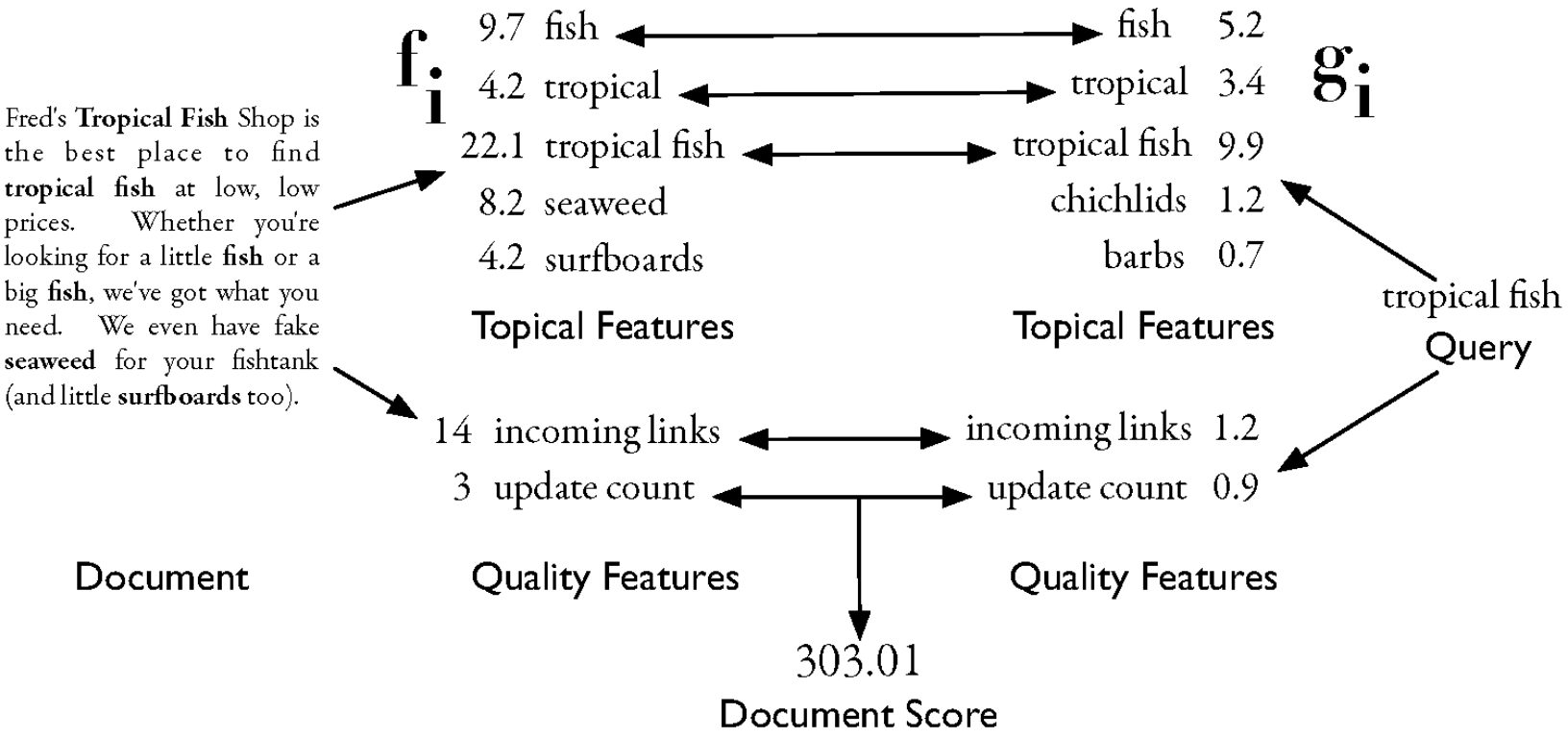
Abstract Model of Ranking



More Concrete Model

$$R(Q,D) = \sum_i g_i(Q) f_i(D)$$

f_i is a document feature function
g_i is a query feature function
i runs over the features



Inverted Index

- Each index term is associated with an *inverted list*
  - Contains lists of documents, or lists of word occurrences in documents, and other information
  - Each entry is called a *posting*
  - The part of the posting that refers to a specific document or location is called a *pointer*
  - Each document in the collection is given a unique number
  - Lists are usually *document-ordered* (sorted by document number)

Simple Inverted Index

S_1 Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

S_2 Fishkeepers often use the term tropical fish to refer only those requiring fresh water, with saltwater tropical fish referred to as marine fish.

S_3 Tropical fish are popular aquarium fish, due to their often bright coloration.

S_4 In freshwater fish, this coloration typically derives from iridescence, while salt water fish are generally pigmented.

and	1				only	2									
aquarium	3				pigmented	4									
are	3	4			popular	3									
around	1				refer	2									
as	2				referred	2									
both	1				requiring	2									
bright	3				salt	1	4								
coloration	3	4			saltwater	2									
derives	4				species	1									
due	3				term	2									
environments	1				the	1	2								
fish	1	2	3	4	their	3									
fishkeepers	2				this	4									
found	1				those	2									
fresh	2				to	2	3								
freshwater	1	4			tropical	1	2	3							
from	4				typically	4									
generally	4				use	2									
in	1	4			water	1	2	4							
include	1				while	4									
including	1				with	2									
iridescence	4				world	1									
marine	2														
often	2	3													

Crawling review

Information Retrieval

These course materials borrow, with permission, from those of Prof. Cristina Videira Lopes, Addison Wesley 2008, Chris Manning, Pandu Nayak, Hinrich Schütze, Heike Adel, Sascha Rothe, Jerome H. Friedman, Robert Tibshirani, and Trevor Hastie

Universal Resource Identifiers

- Universal Resource Identifier (URI)
  - Definition: A string of characters used to identify a resource
- Examples of URIs:
  - http://www.ics.uci.edu (URL)
  - ISBN 0-486-27777-3 (URN)
  - ftp://ftp.ics.uci.edu (URL)
- URL (locator) vs URN (name)
  - Locator: must specify ***where*** the resource is. Name: just ***what*** it is.
- We are going to focus on URLs
  - But "URI" might slip in as synonym

Anatomy of a URL

- Syntax:
  - scheme://domain:port/path?query_string#fragment_id

    mandatory: scheme://domain

    optional: :port/path?query_string#fragment_id

    authority: domain:port
  - (slightly more complicated than this)
- Full spec:
  - http://www.w3.org/Addressing/URL/url-spec.txt

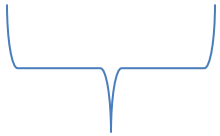
Anatomy of a URL

- http://www.ics.uci.edu/~lopes

on a web server *no port! just domain* *path* *query*

- http://calendar.ics.uci.edu/calendar.php?type=month&calendar=1&category=&month=02&year=2013

- Domains and subdomains:
  - calendar.ics.**uci.edu**



Domain name

Different Flavors of Web Data Collection

- How to acquire data?
  - Data dumps
  - URL downloads
  - Web APIs
  - Web Crawling

Data dumps

- Sites may package their data periodically and provide it as a "dump"
  - Example: Wikipedia (it suggests you to Torrent)
  - arXiv Bulk Full-Text Access: https://arxiv.org/help/bulk_data_s3

URL Downloads

- Two step process:
  1. Find out the URLs of specific resources
  2. Run a downloader that takes that list and downloads the resources
- Example: "crawling" (!) sourceforge for source code
- Some sites use regular URLs. E.g. Google Code
  - http://code.google.com/p/**python-for-android**/downloads/list
  - …
- Doesn't need to be source code; can be papers, pages, etc.
  - http://link.springer.com/chapter/10.1007/978-3-642-34213-4_1
  - http://link.springer.com/chapter/10.1007/978-3-642-34213-4_2
  - …

Web APIs

- Sites may provide (REST) interfaces for getting their data
  - Usually higher-level: avoids having to parse HTML
  - Usually restrictive: only part of the data
- Examples:
  - Facebook Graph API
    - My data in facebook api
    - More examples
  - Youtube API
  - Twitter API
  - arXiv API
  - …

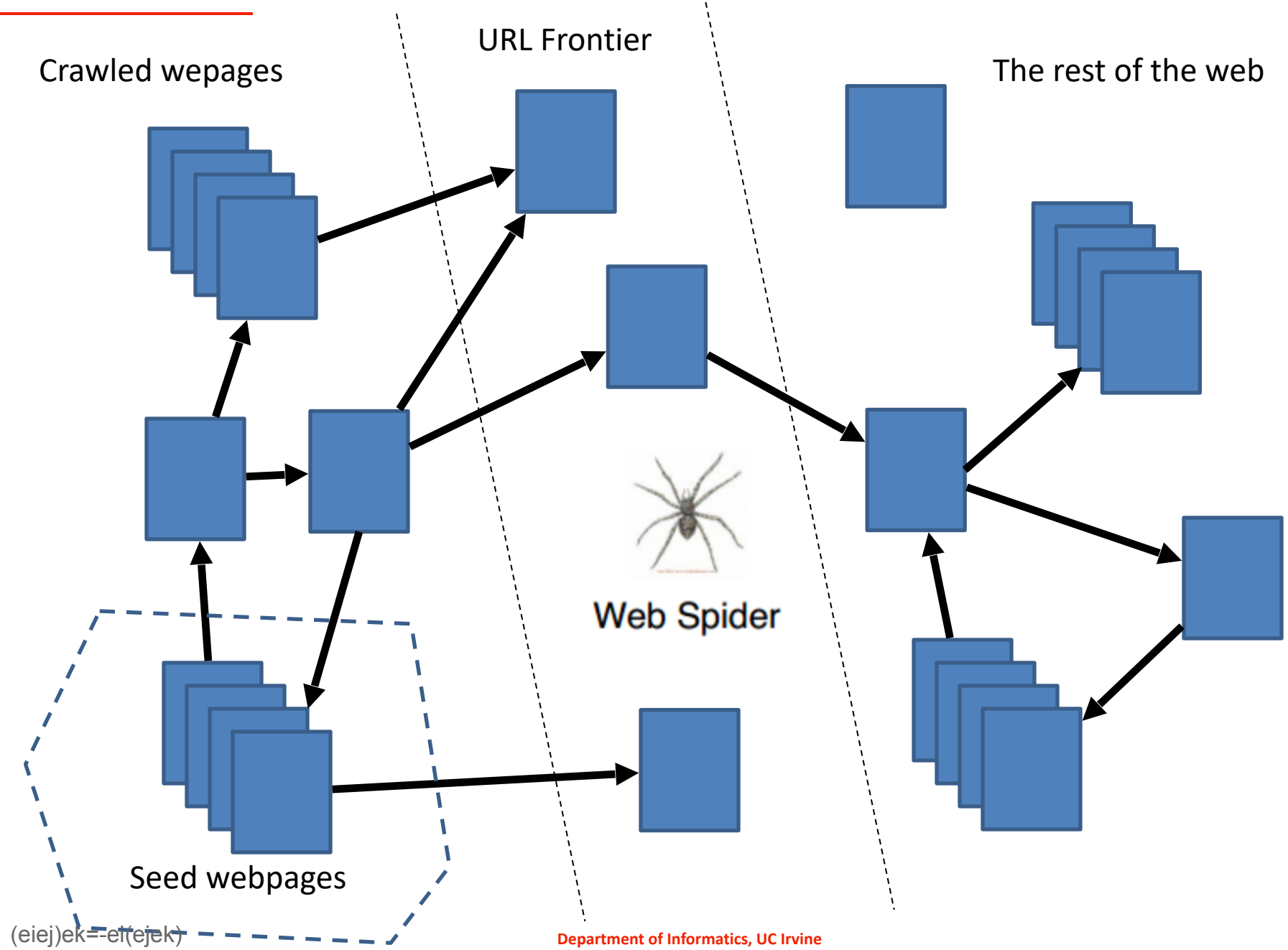
Web Crawling

- Like people, getting HTML pages and other documents and discovering new URLs as it goes
  - Good for changing collections
  - Good for unknown documents
- Web admins don't like crawlers
  - Crawlers consume resources that are meant for people

Basic Crawl Method

- Initialize a queue of URLs (seeds)
- Repeat until no more URLs in queue:
  - Get one URL from the queue
  - If the page can be crawled, fetch associated page
  - Store representation of page
  - Extract URLs from page and add them to the queue
- Queue = “frontier”

Basic Crawl Method



Pseudo Code

```
procedure CrawlerThread(frontier)
    while not frontier.done() do
        website ← frontier.nextSite()
        url ← website.nextURL()
        if website.permitsCrawl(url) then
            text ← retrieveURL(url)
            storeDocument(url, text)
            for each url in parse(text) do
                frontier.addURL(url)
            end for
        end if
        frontier.releaseSite(website)
    end while
end procedure
```

Pseudo Code

```
procedure CrawlerThread(frontier)
    while not frontier.done() do
        website ← frontier.nextSite()
        url ← website.nextURL()
        if website.permitsCrawl(url) then
            text ← retrieveURL(url)
            storeDocument(url, text)
            for each url in parse(text) do
                frontier.addURL(url)
            end for
        end if
        frontier.releaseSite(website)
    end while
end procedure
```

Permission to crawl

- Robots Exclusion Standard aka **robots.txt**
  - Sites may have that file at the root. Examples:
    - http://www.cnn.com/robots.txt
    - http://en.wikipedia.org/robots.txt
  - Very simple syntax:
    - http://www.robotstxt.org/robotstxt.html
  - **Honor basis!**
    - **It's not a security mechanism**

Permission to crawl

Exclude all

```
User-agent: *
Disallow: /
```

Allow all

```
User-agent: *
Disallow:
```

Exclude all from a part of the site

```
User-agent: *
Disallow: /cgi-bin/
Disallow: /tmp/
Disallow: /junk/
```

Exclude a single robot

```
User-agent: BadBot
Disallow: /
```

Allow a single robot

```
User-agent: Google
Disallow:

User-agent: *
Disallow: /
```

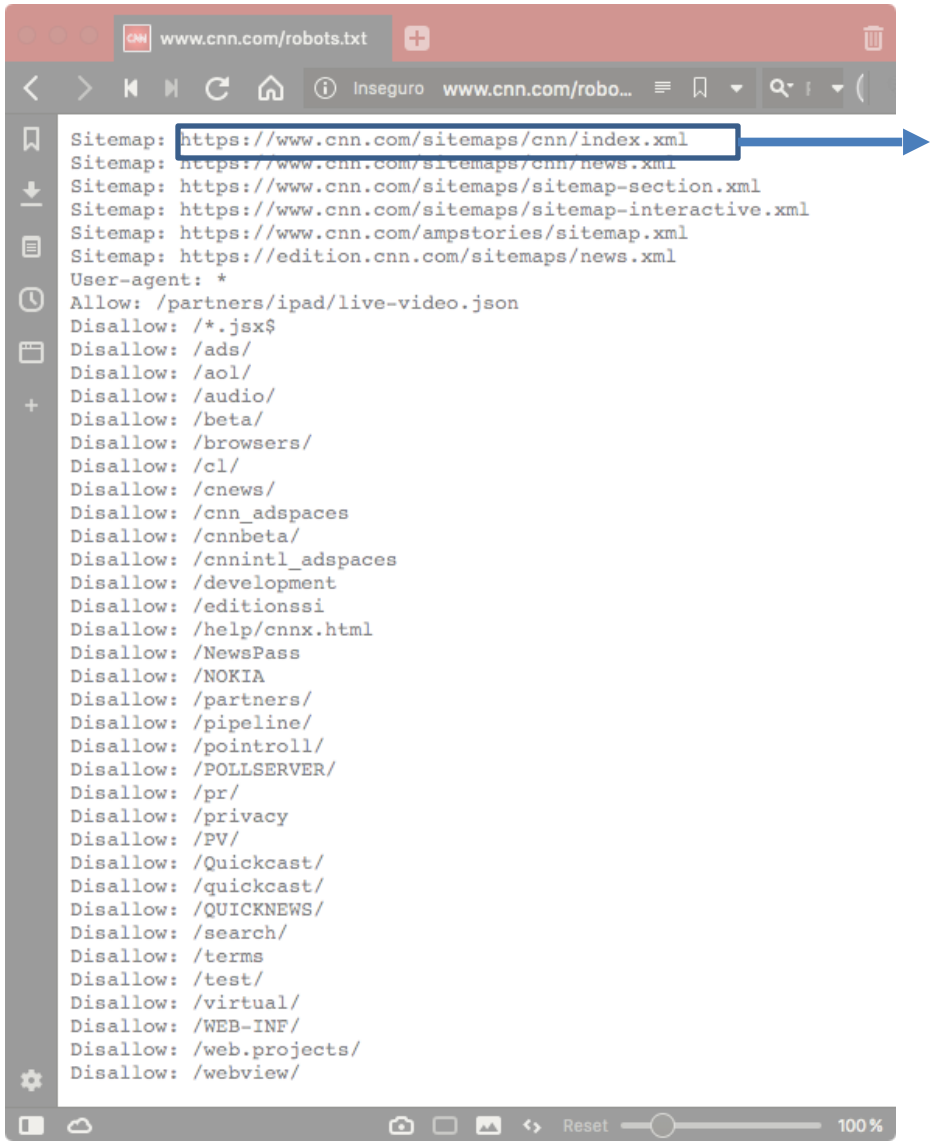
http://www.robotstxt.org/robotstxt.html

Information to crawlers

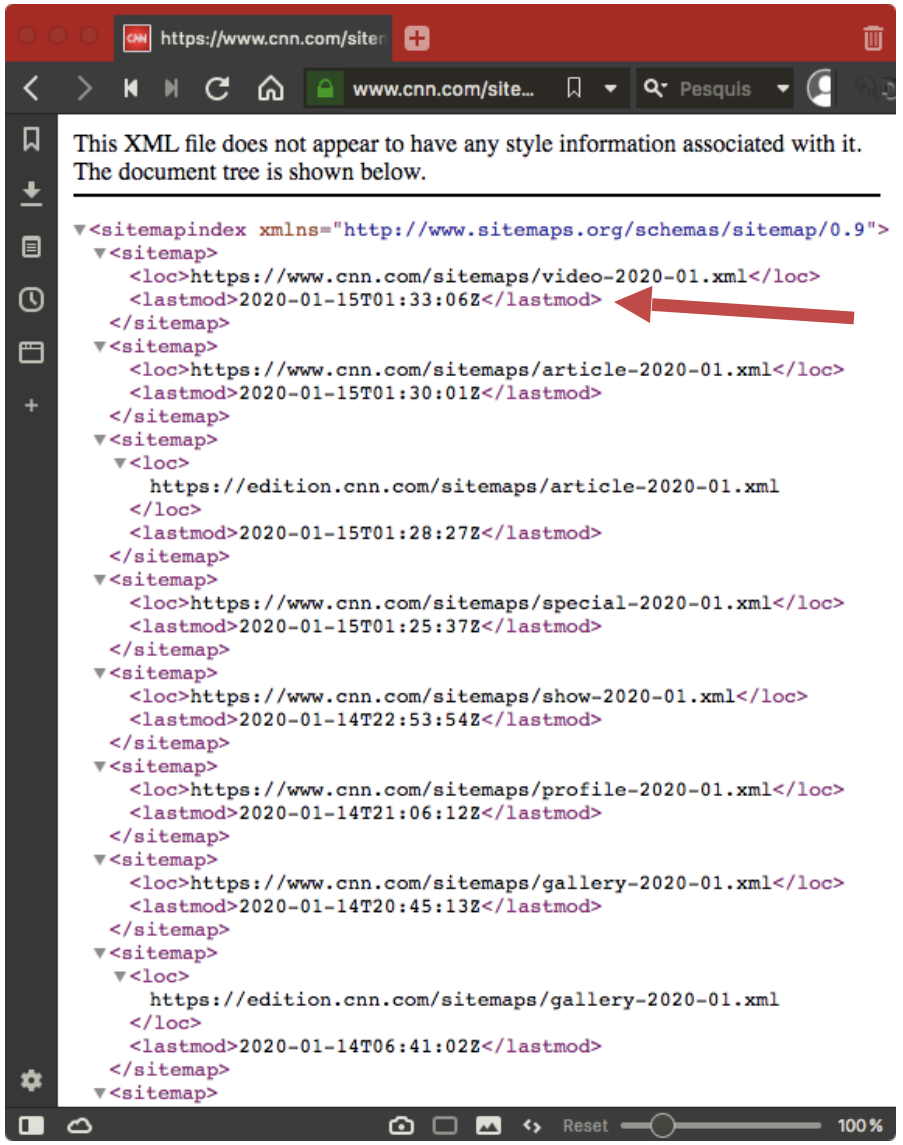
- Sitemaps (introduced by Google)
- Also listed in robots.txt
- Allow web masters to send info to crawlers
  - Location of pages that might not be linked
  - Relative importance
  - Update frequency
- Example:
  - http://www.cnn.com/robots.txt

Information to crawlers

http://www.cnn.com/robots.txt



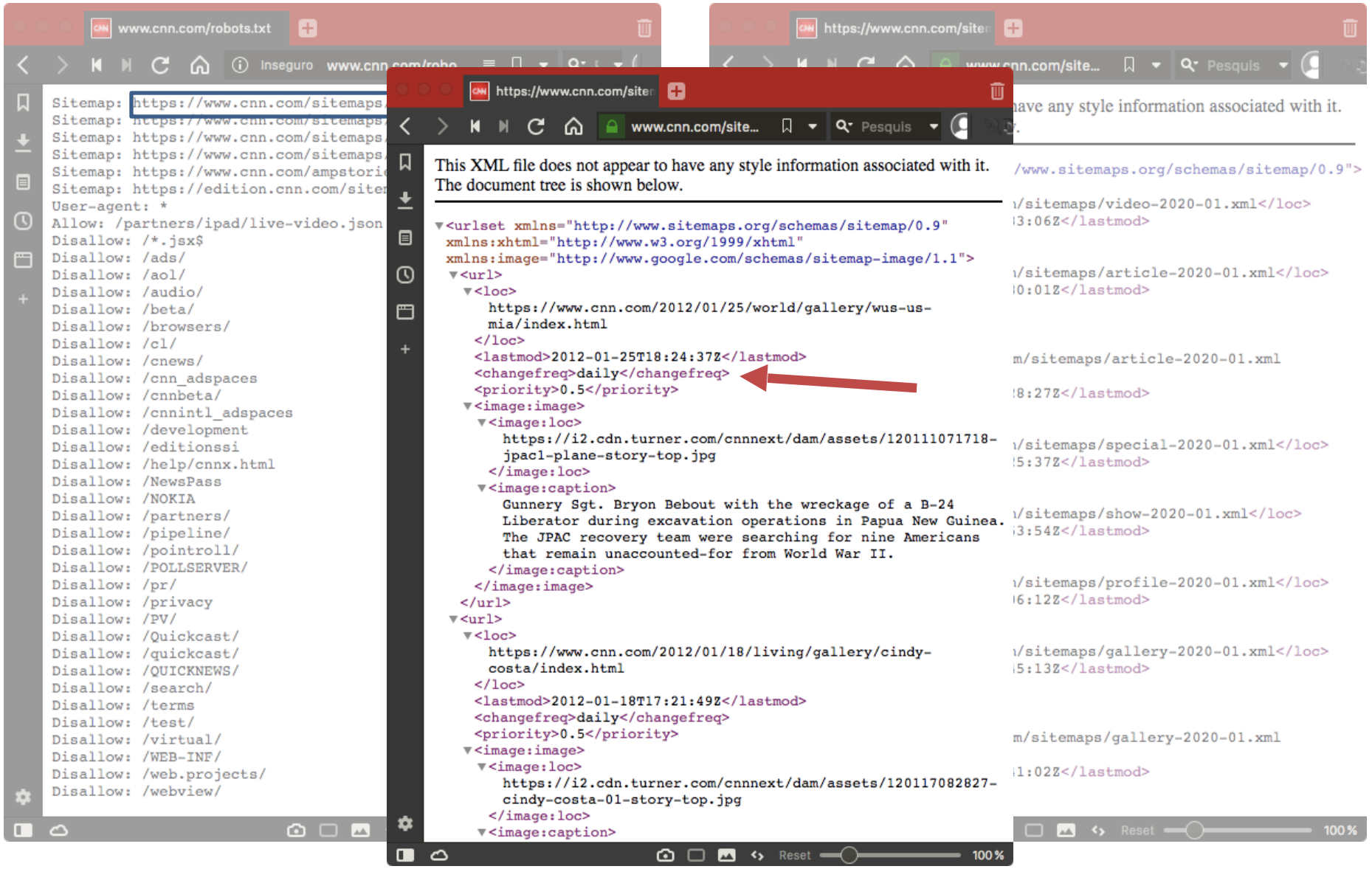
https://www.cnn.com/sitemaps/cnn/index.xml



Information to crawlers

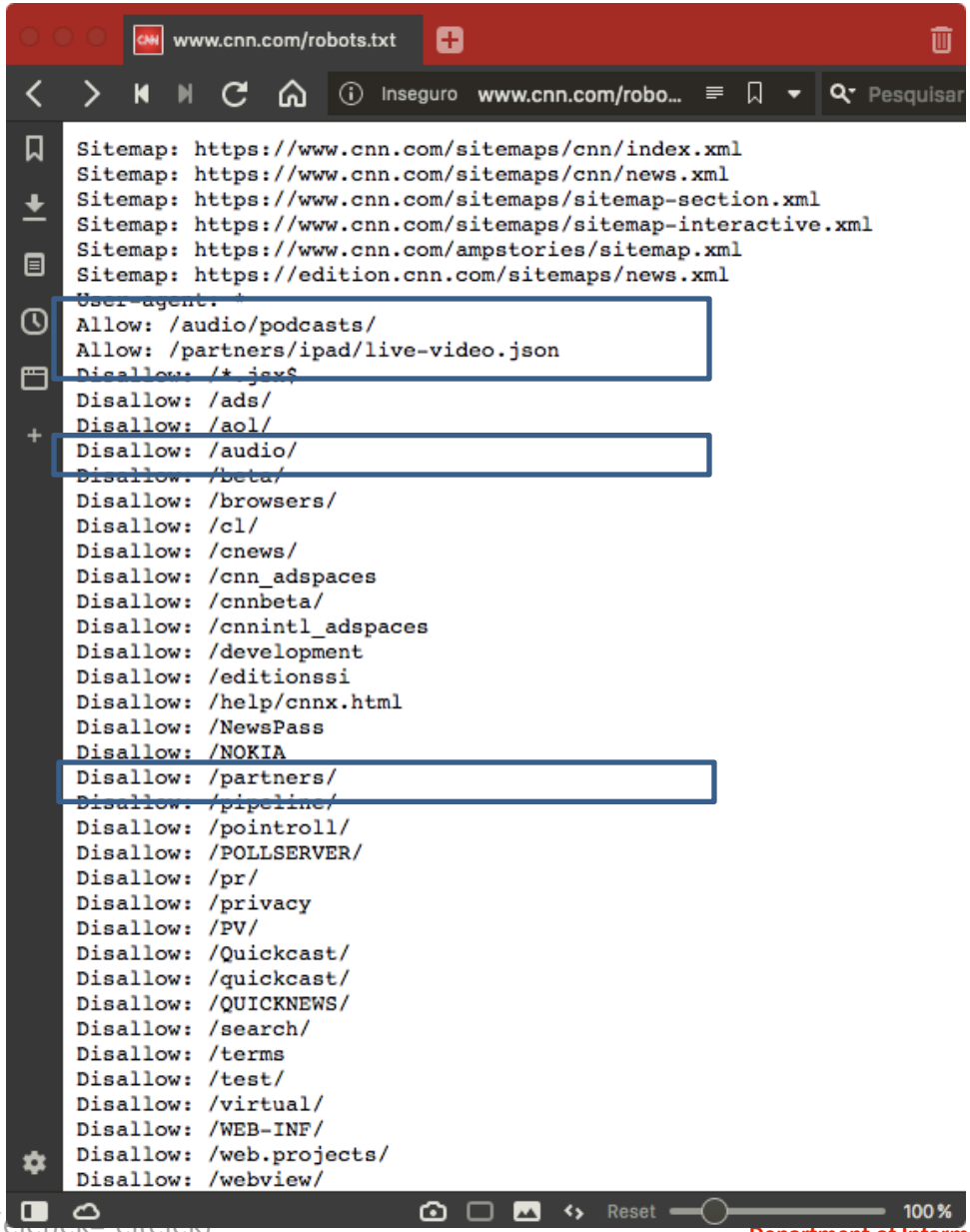
http://www.cnn.com/robots.txt

https://www.cnn.com/sitemaps/cnn/index.xml



Information to crawlers

http://www.cnn.com/robots.txt



- You can allow a specific resource from a disallowed resource
  - For instance: *you can disallow to crawl some folder, but allow the bot to crawl a resource that contains a list of links that you want to be indexed.*

“Basic method” is...

- Theoretically correct
- **Seriously lacking to use in practice**
  1. Will upset web admins (impolite)
     - It’s abusing the web servers
  2. Very slow
     - 1 page at a time
  3. Will get caught in traps and infinite sequences
  4. Will fetch duplicates without noticing
  5. Will bring in data noise
  6. Will miss content due to client-side scripting

1. Politeness

- Avoid hitting any site too often
  - Sites are for people, not for bots
- Ignore politeness ➔ Denial of service (DOS) attack
- Be polite ➔ Use artificial delays

2. Performance (I)

- Back of the envelope calculation:
  - 1 page fetch = 500ms
  - How much time to crawl 1 million pages?
    - (it's worse than that... Unresponsive servers)
- Most of the time, the crawler thread is waiting for the network data
- Solution: multi-threaded or distributed crawling
  - Politeness is harder to control (but it is possible: e.g. different servers)

2. Performance (II)

- Domain Name lookups
  - Given a domain name, retrieve its IP address
    - www.ics.uci.edu -> 128.195.1.83
- Distributed set of servers
  - Latency can be high (2 secs is not unusual)
- Common implementations are blocking
  - One request at a time
  - Result is cached
- Back of the envelope calculation:
  - 1 DNS lookup ➔ 800ms
  - How much time to lookup the entire Web?

3. Crawler traps

- **May** trap the crawler on the site forever
  - Web server responds with ever changing URLs and content
    - Dynamic pages
  - May be intentional or unintentional
    - E.g. the ICS calendar is a crawler trap
    - Webadmins can create traps to penalize impolite crawlers

- See http://www.fleiner.com/bots/
    - E.g. very large documents, disalowed in robots.txt, created to consume crawler resources or event to break poorly designed parsers of crawlers that ignore robots.txt

4. Duplicate Detection

- Duplication and near-duplication is widespread
  - Copies, mirror sites, versions, spam, plagiarism...
  - Studies: 30% of Web pages are [near-]duplicates of the other 70%
  - Little or no value, noise

- Detection
  - Detection of exact duplication is easy, but exact duplication is rare
    - Hashes, checksums
  - Detection of near-duplicates is hard
    - Page fingerprints

4. Duplicate Detection (Exact duplicate)

- *Exact* duplicate detection is relatively easy
- *Checksum* techniques
  - A checksum is a value computed from the content of the document
    - e.g., sum of the bytes in the document file

T	r	o	p	i	c	a	l		f	i	s	h	*Sum*
54	72	6F	70	69	63	61	6C	20	66	69	73	68	508

  - Possible for files with different text to have same checksum
- Functions such as a *cyclic redundancy check* (CRC), have been developed that consider the positions of the bytes
  - Still prone to collisions, but very rare
- Need to be fast

4. Duplicate Detection (Near duplicate)

- More challenging task
  - Are web pages with same text context but different advertising or format near-duplicates?

- A near-duplicate document is defined using a threshold value for some similarity measure between pairs of documents
  - e.g., document *D1* is a near-duplicate of document *D2* if more than 90% of the words in the documents are the same

(eiej)ek=-ei(ejek)

4. Duplicate Detection (Near duplicate : Fingerprint)

Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

(a) Original text

tropical fish include, fish include fish, include fish found, fish found in, found in tropical, in tropical environments, tropical environments around, environments around the, around the world, the world including, world including both, including both freshwater, both freshwater and, freshwater and salt, and salt water, salt water species

(b) 3-grams

938 664 463 822 492 798 78 969 143 236 913 908 694 553 870 779

(c) Hash values

664 492 236 908

(d) Selected hash values using *0 mod 4*

4. Duplicate Detection (Near duplicate : Simhash)

Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

(a) Original text

tropical 2 fish 2 include 1 found 1 environments 1 around 1 world 1
including 1 both 1 freshwater 1 salt 1 water 1 species 1

(b) Words with weights

tropical	01100001	fish	10101011	include	11100110
found	00011110	environments	00101101	around	10001011
world	00101010	including	11000000	both	10101110
freshwater	00111111	salt	10110101	water	00100101
species	11101110				

(c) 8 bit hash values

1 -5 9 -9 3 1 3 3

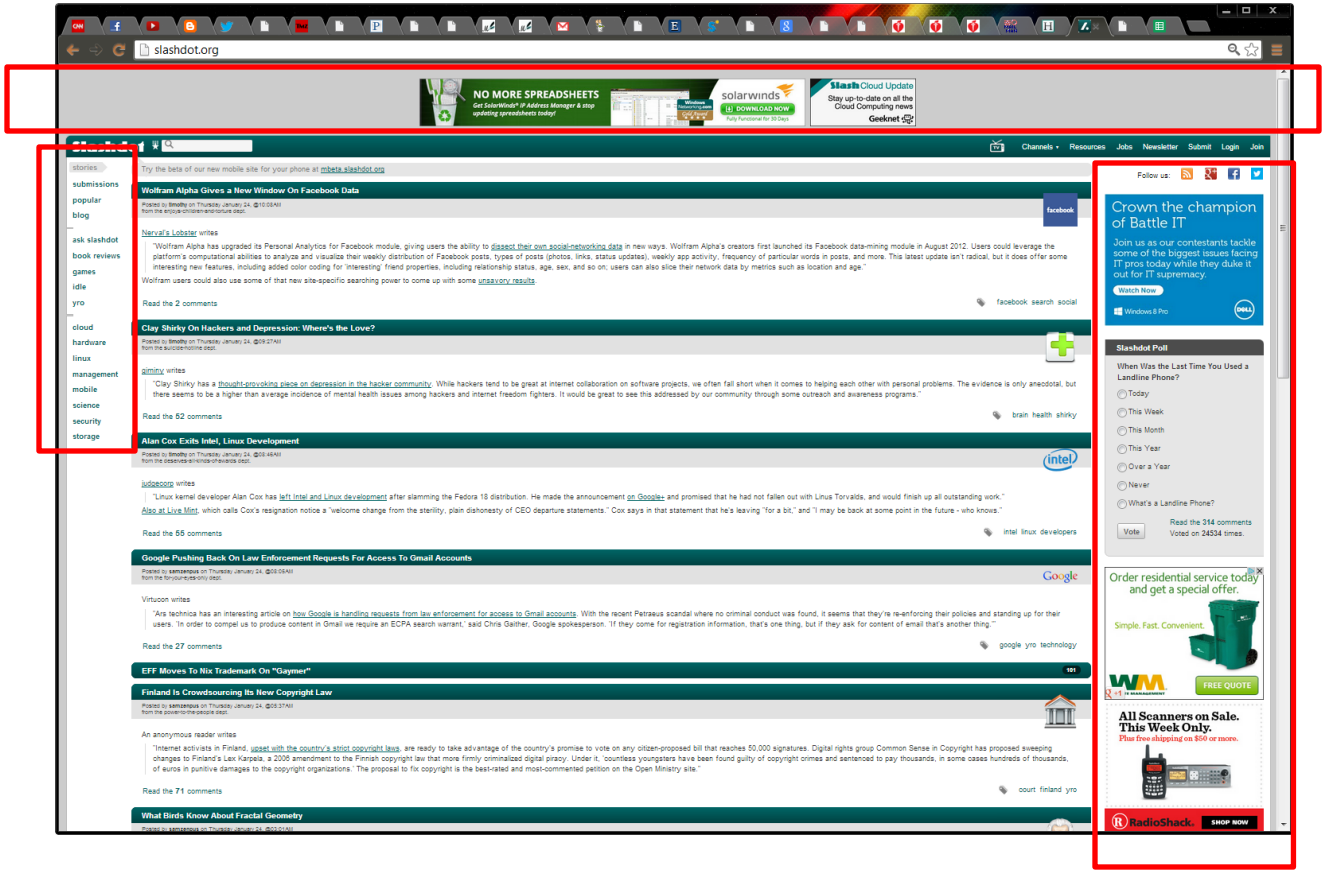
(d) Vector V formed by summing weights

1 0 1 0 1 1 1 1

(e) 8-bit fingerprint formed from V

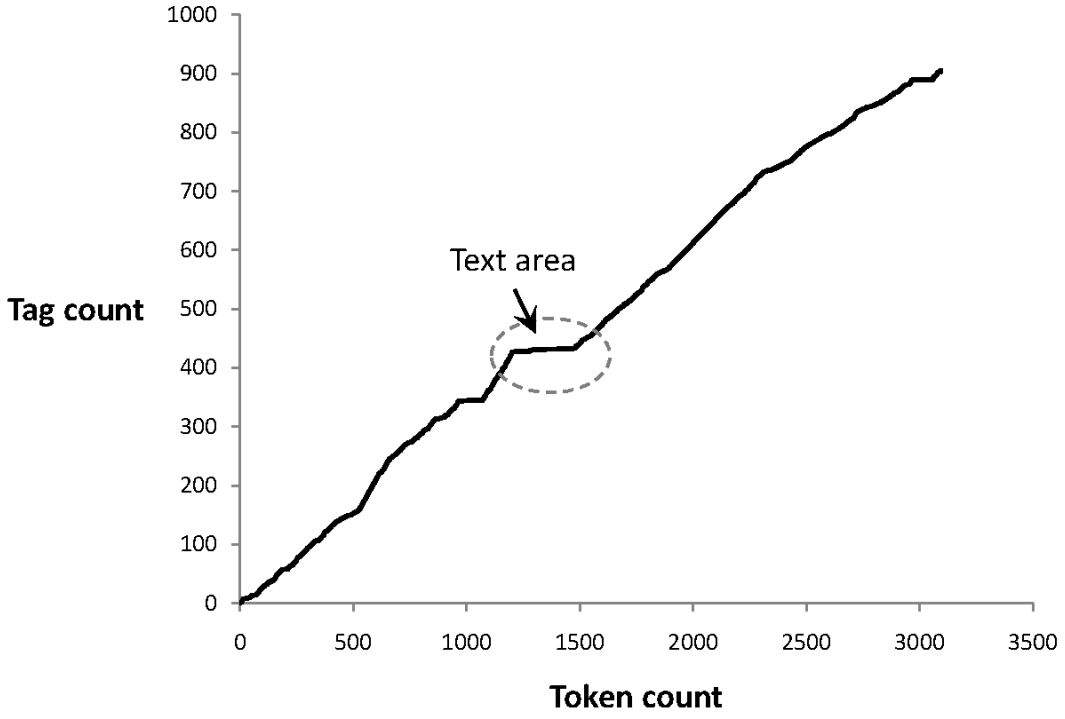
5. Data Noise

- Web pages have content not directly related to the page
  - Ads, templates, etc
  - Noise negatively impacts information retrieval



5. Data Noise : Finding Content Blocks

- Cumulative distribution of tags in the example web page
  - Document slope curve (e.g. Finn, Kushmerick & Smyth, 2001)



  - Main text content of the page corresponds to the “plateau” in the middle of the distribution

5. Data Noise : Finding Content Blocks

- Represent a **web page** as a **sequence** of **bits**, where $b_n = 1$ indicates that the ***n*th token is a tag**

- Optimization problem where we **find values of *i* and *j* to maximize** both the number of **tags below *i* and above *j* and** the number of **non-tag tokens between *i* and *j***

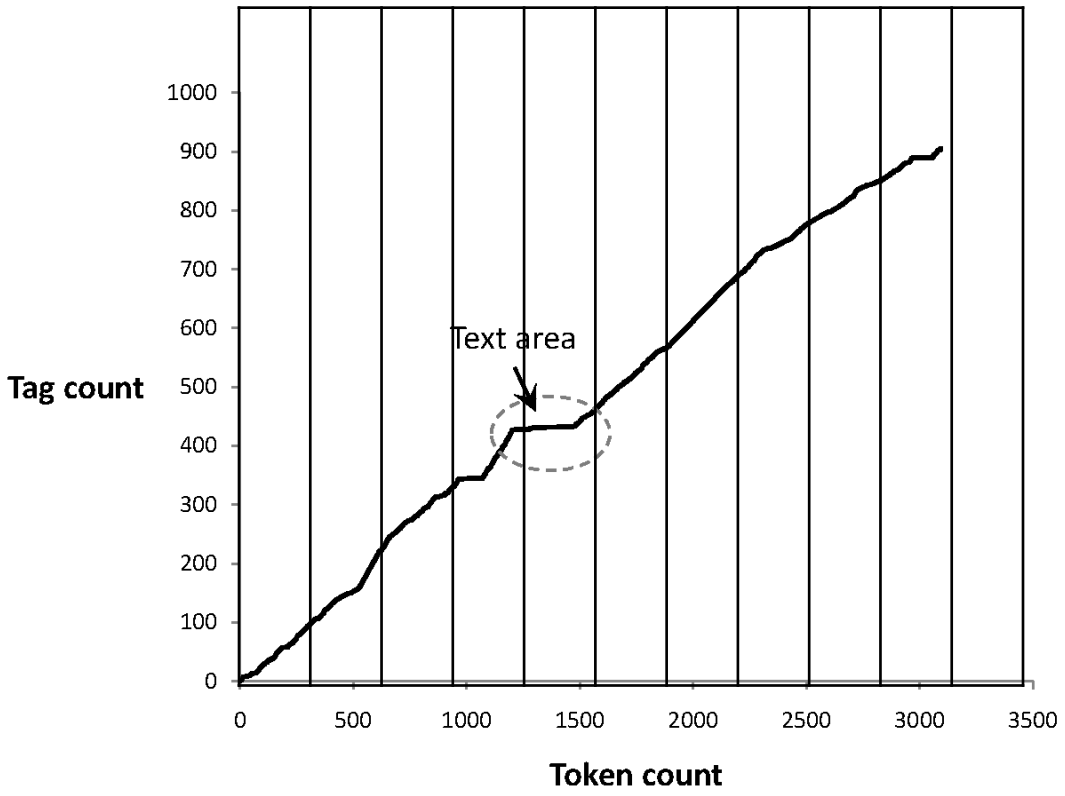
- i.e., maximize

$$\sum_{n=0}^{i-1} b_n + \sum_{n=i}^{j} (1 - b_n) + \sum_{n=j+1}^{N-1} b_n$$

(eiej)ek=-ei(ejek)

Finding Content Blocks

- Cumulative distribution of tags in the example web page

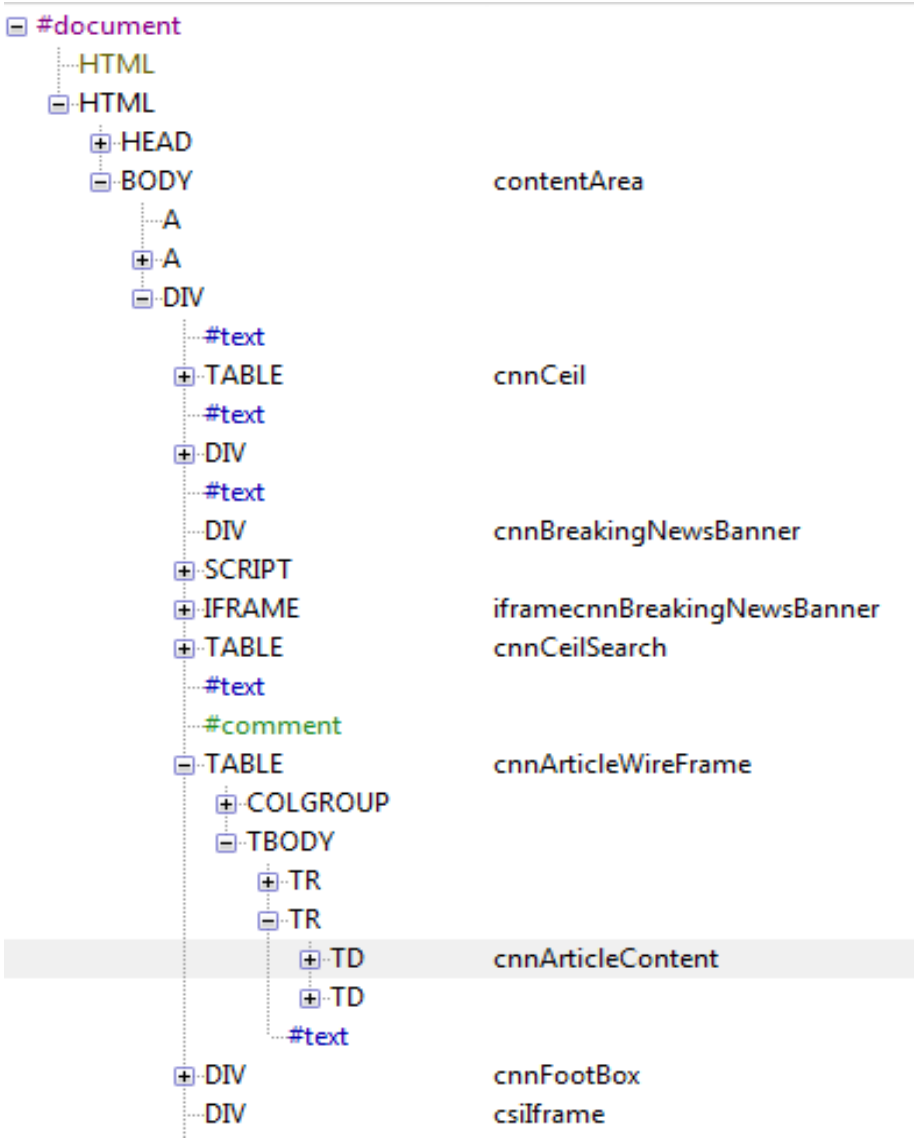


  - Determine the slopes inside slices and iterate.

Finding Content Blocks

- Other approaches use Document Object Model (DOM) structure and visual (layout) features

- HTML parser:
  - HTML -> Document Object Model representation
  - Tree like structure



6. Client-Side Scripting

- Modern web sites are heavily scripted (JavaScript)
  - Content behind XMLHttpRequests
- To get to that content crawlers must run the scripts
  - Hard thing to do: user interaction emulation (e.g. Selenium)
  - Simple crawlers will not do it

The Deep Web

- Places where crawlers rarely go...
  - Content behind login forms
  - Content behind JavaScript
  - Sites that aren't linked from anywhere

- It is estimated that the deep web is 400-500x larger than the shallow web [http://dx.doi.org/10.3998/3336451.0007.104]

Additional hint for the Quiz

- Review and make sure you understand what your crawler in Assignment 2 is doing, including behavior, HTTP status codes, etc.