Week 3 Lecture
Web Search Engine Technology

CAB431

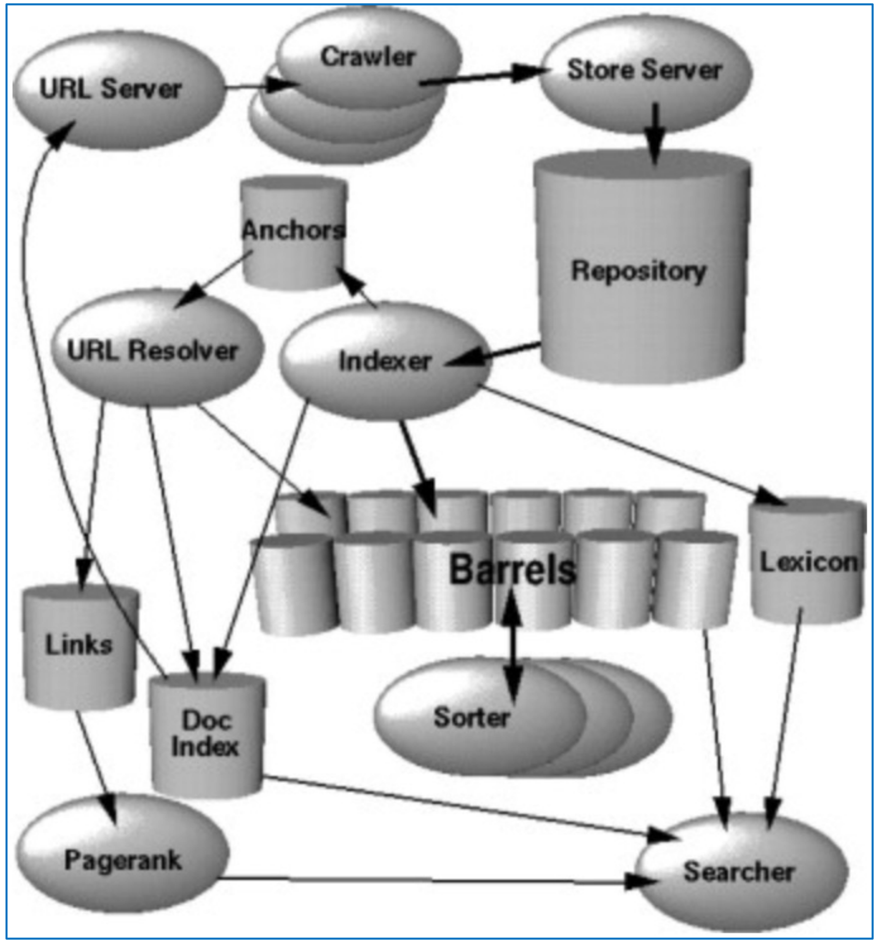
Professor Yuefeng Li
School of Computer Science
Queensland University of Technology

Learning Objectives

1. *Search Engine Architecture*
2. *Indexing Process*
    - *Text acquisition*
    - *Text transformation*
    - *Index creation*
3. Query Process
    - User interaction
    - Ranking, e.g., PageRank
    - Evaluation
4. *Web Crawler*
    - *Controlling Crawling*
    - *Freshness*
    - *Document feeds*
    - *Storing documents, e.g., BigTable*
    - *Detecting duplicates &* Removing Noise

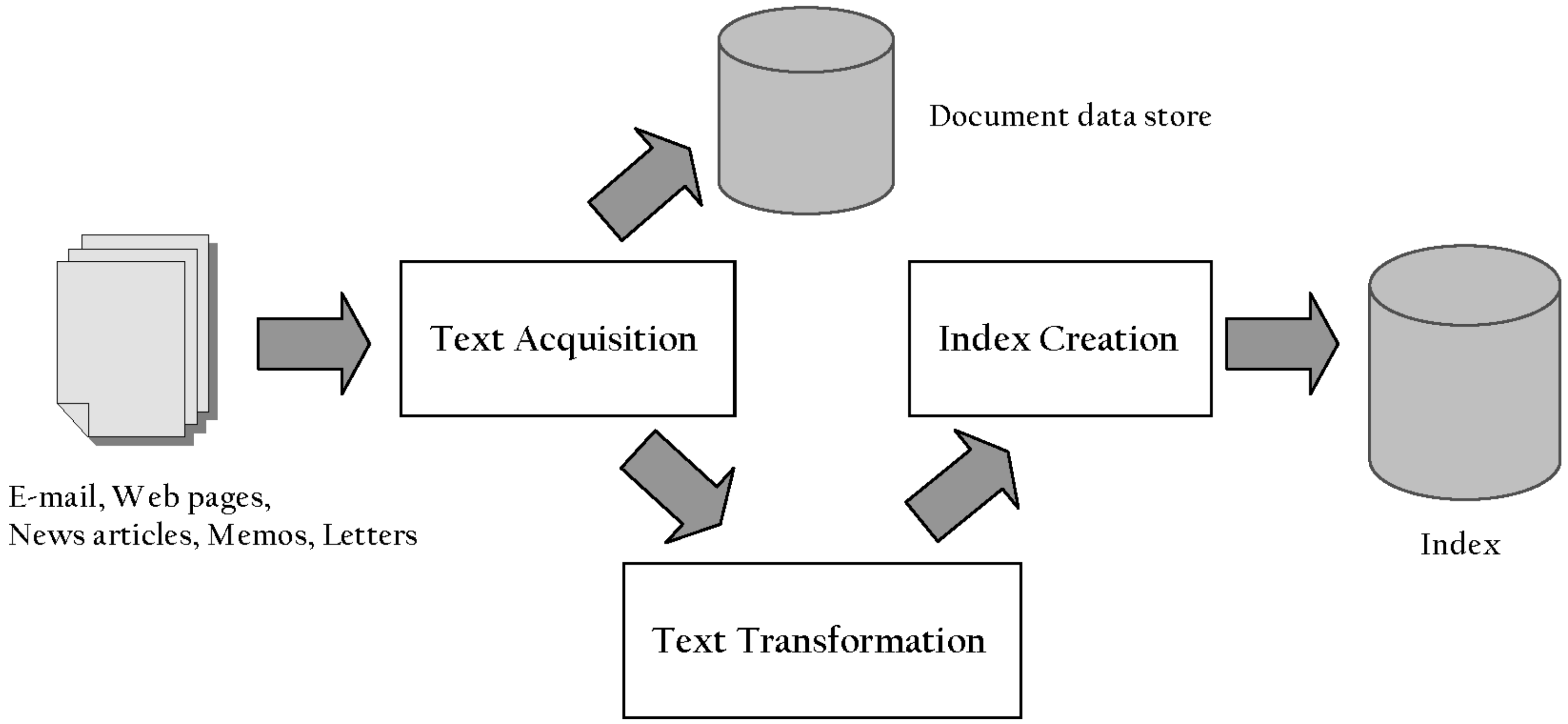
1. Search Engine Architecture

- A software architecture consists of software components, the interfaces provided by those components, and the relationships between them
  - describes a system at a particular level of abstraction
- Architecture of a search engine
  - Requirements
    - **effectiveness** (quality of results) and **efficiency** (response time and throughput)
  - Major functions
    - Indexing process and query process



High Level Google Architecture

2. Indexing Process



Indexing Process cont.

- Text acquisition
  - identifies and stores documents for indexing
- Text transformation
  - transforms documents into *index terms* or *features*
- Index creation
  - takes index terms and creates **data structures** (*indexes*) to support fast searching

Details: Text Acquisition

- Crawler
  - Identifies and acquires documents for search engine
  - Many types – web, enterprise, desktop
  - Web crawlers follow *links* to find documents
    - Must efficiently find huge numbers of web pages (*coverage*) and keep them up-to-date (*freshness*)
    - Single site crawlers for *site search*
    - *Topical* or *focused* crawlers for vertical search
  - *Document* crawlers for enterprise and desktop search
    - Follow links and scan directories

Text Acquisition cont.

- Feeds
  - Real-time streams of documents
    - e.g., web feeds for news, blogs, video, radio, tv
  - RSS is common standard
    - RSS “reader” can provide new XML documents to search engine
- Conversion
  - Convert variety of documents into a consistent text plus metadata format
    - e.g., HTML, XML, Word, PDF, etc. → XML
  - Convert text encoding for different languages
    - Using a Unicode standard like UTF-8

Text Acquisition

- Document data store
    - Stores text, metadata, and other related content for documents
        - Metadata is information about document such as type and creation date
        - Other content includes links, anchor text
    - Provides fast access to document contents for search engine components
        - e.g., result list generation
    - Could use relational database system
        - More typically, a simpler, more efficient storage system is used due to huge numbers of documents

Text Transformation

- Parser
  - Processing the sequence of text *tokens* in the document to recognize structural elements
    - e.g., titles, links, headings, etc.
  - *Tokenizer* recognizes “words” in the text
    - must consider issues like capitalization, hyphens, apostrophes, non-alpha characters, separators
  - *Markup languages* such as HTML, XML often used to specify structure
    - *Tags* used to specify document *elements*
      - E.g., <h2> Overview </h2>
    - Document parser uses *syntax* of markup language (or other formatting) to identify structure

Text Transformation cont.

- Stopping
  - Remove common words
    - e.g., “and”, “or”, “the”, “in”
  - Some impact on efficiency and effectiveness
  - Can be a problem for some queries
- Stemming
  - Group words derived from a common *stem*
    - e.g., “computer”, “computers”, “computing”, “compute”
  - Usually effective, but not for all queries
  - Benefits vary for different languages

Text Transformation

- Link Analysis
  - Makes use of *links* and *anchor text* in web pages
  - Link analysis identifies *popularity* and *community* information
    - e.g., PageRank
  - Anchor text can significantly enhance the representation of pages pointed to by links
  - Significant impact on web search
    - Less importance in other applications

Text Transformation cont.

- Information Extraction
  - Identify classes of index terms that are important for some applications
  - e.g., *named entity recognizers* identify classes such as *people*, *locations*, *companies*, *dates,* etc.
- Classifier
  - Identifies class-related metadata for documents
    - i.e., assigns labels to documents
    - e.g., topics, reading levels, sentiment, genre
  - Use depends on application

Index Creation

- Document Statistics
  - Gathers counts and positions of words and other features
  - Used in ranking algorithm
- Weighting
  - Computes weights for index terms
  - Used in ranking algorithm
  - e.g., *tf.idf* weight
    - Combination of *term frequency* in document and *inverse document frequency* in the collection

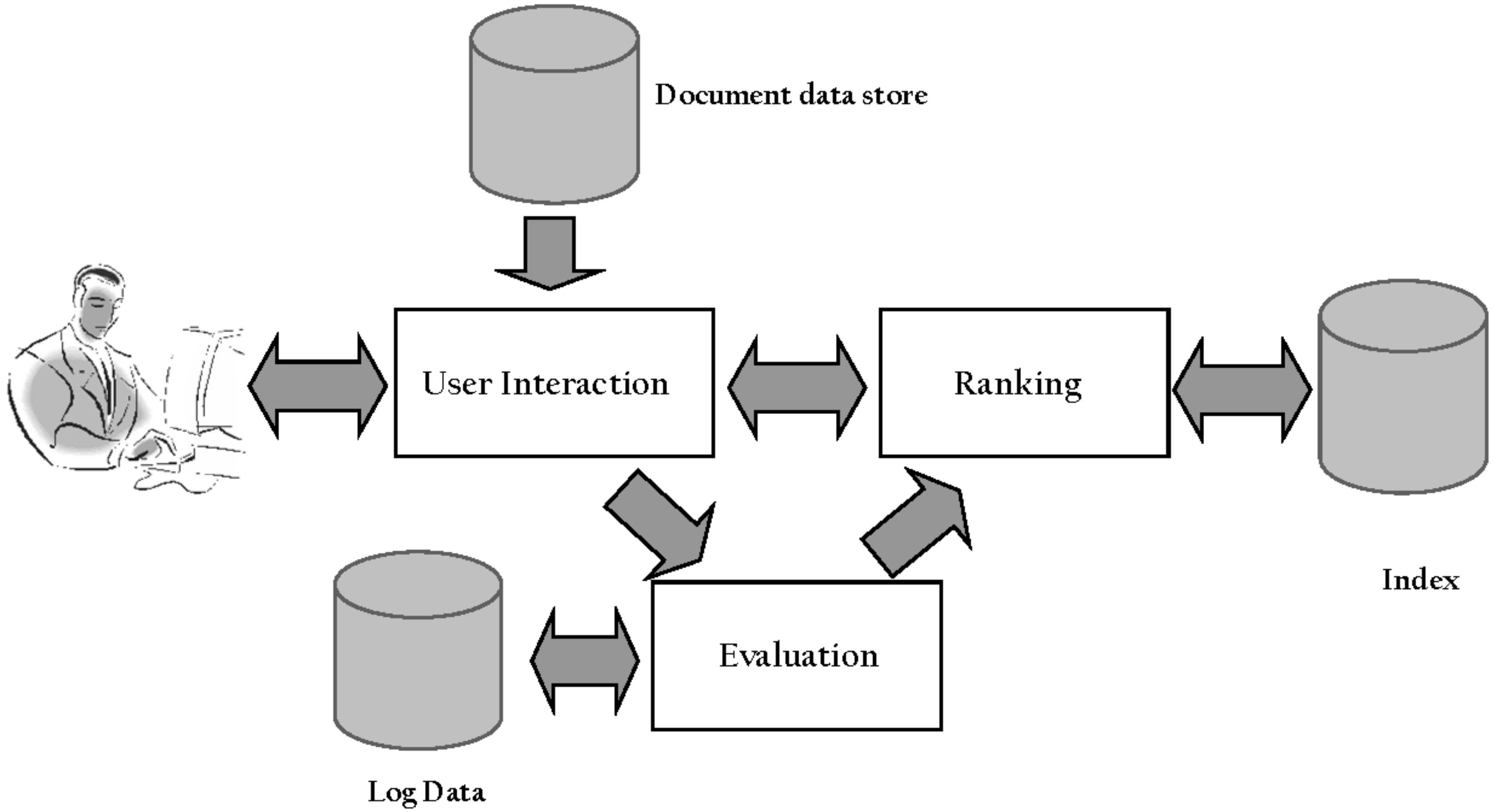
Index Creation cont.

- Inversion
  - Core of indexing process
  - Converts document-term information to term-document for indexing
    - Difficult for very large numbers of documents
  - Format of inverted file is designed for fast query processing
    - Must also handle updates
    - Compression used for efficiency

Index Creation

- Index Distribution
  - Distributes indexes across multiple computers and/or multiple sites
  - Essential for fast query processing with large numbers of documents
  - Many variations
    - Document distribution, term distribution, replication
  - *P2P* and *distributed IR* involve search across multiple sites

3. Query Process



Query Process cont.

- User interaction
  - supports creation and refinement of query, display of results
- Ranking
  - uses query and indexes to generate ranked list of documents
- Evaluation
  - monitors and measures effectiveness and efficiency (primarily offline)

User Interaction

- Query input
  - Provides interface and parser for *query language*
  - Most web queries are very simple, other applications may use forms
  - Query language used to describe more complex queries and results of query transformation
    - e.g., Boolean queries, Indri and Galago query languages
    - It is similar to the SQL language used in database applications
    - IR query languages also allow content and structure specifications, but focus on content

User Interaction cont.

- Query transformation
  - Improves initial query, both before and after initial search
  - Includes text transformation techniques used for documents
  - *Spell checking* and *query suggestion* provide alternatives to original query
  - *Query expansion* and *relevance feedback* modify the original query with additional terms

- Results output
  - Constructs the display of ranked documents for a query
  - Generates *snippets* to show how queries match documents
  - *Highlights* important words and passages
  - Retrieves appropriate *advertising* in many applications
  - May provide *clustering* and other visualization tools

Ranking

- Scoring
  - Calculates scores for documents using a ranking algorithm
  - Core component of search engine
  - Basic form of score is $\sum (q_i * d_i)$
    - q_i and d_i are query and document term weights for term i
  - Many variations of ranking algorithms and retrieval models

Link Analysis

- Billions of web pages, some more informative than others
- Links can be viewed as information about the *popularity* (*authority*?) of a web page
  - can be used by ranking algorithm
- *Inlink* (links pointing to a page) count could be used as simple measure
- Link analysis algorithms like PageRank provide more reliable ratings
  - less susceptible to link spam

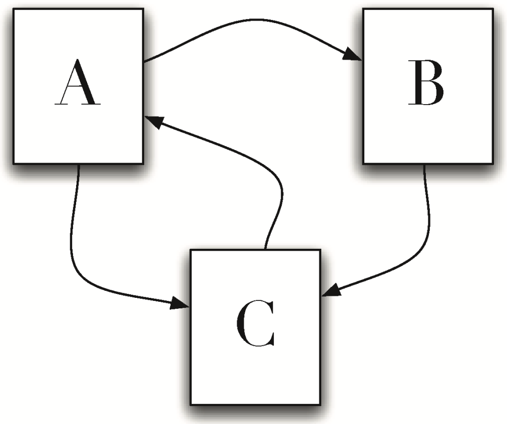
Random Surfer Model

- Browse the Web using the following algorithm:
  - Choose a random number r between 0 and 1
  - If $r < \lambda$:
    - Go to a random page
  - If $r \geq \lambda$:
    - Click a link at random on the current page
  - Start again
- PageRank of a page is the probability that the “random surfer” will be looking at that page
  - links from popular pages will increase PageRank of pages they point to

Dangling Links

- Random jump prevents getting stuck on pages that
  - do not have links
  - contains only links that no longer point to other pages
  - have links forming a loop
- Links that point to the first two types of pages are called *dangling links*
  - may also be links to pages that have not yet been crawled

PageRank



- PageRank (*PR*) of page C = *PR*(A)/2 + *PR*(B)/1
- More generally,

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L_v}$$

  - where B_u is the set of pages that point to *u*, and L_v is the number of outgoing links from page *v* (not counting duplicate links)
  - For the above example, we have $B_c = \{A, B\}$, $L_A = 2$ and $L_B = 1$.

PageRank Example

- Don't know PageRank values at start
- Assume equal values (1/3 in this case), then iterate:
  - first iteration: *PR*(C) = 0.33/2 + 0.33 = 0.5, *PR*(A) = 0.33, and *PR*(B) = 0.17
  - second: *PR*(C) = 0.33/2 + 0.17 = 0.33, *PR*(A) = 0.5, *PR*(B) = 0.17
  - third: *PR*(C) = 0.42, *PR*(A) = 0.33, *PR*(B) = 0.25
- Converges to *PR*(C) = 0.4, *PR*(A) = 0.4, and *PR*(B) = 0.2

PageRank cont.

- Taking random page jump into account, 1/3 chance of going to any page when $r < \lambda$
- $PR(C) = \lambda/3 + (1 - \lambda) \cdot (PR(A)/2 + PR(B)/1)$
- More generally,

$$PR(u) = \frac{\lambda}{N} + (1 - \lambda). \sum_{v \in B_u} \frac{PR(v)}{L_v}$$

  - where N is the number of pages, λ typically 0.15

PageRank Algorithm

```
procedure PageRank($G$)
        ▷ $G$ is the web graph, consisting of vertices (pages) and edges (links).
    $(P, L) \leftarrow G$                              ▷ Split graph into pages and links
    $I \leftarrow$ a vector of length $|P|$             ▷ The current PageRank estimate
    $R \leftarrow$ a vector of length $|P|$     ▷ The resulting better PageRank estimate
    for all entries $I_i \in I$ do
        $I_i \leftarrow 1/|P|$                    ▷ Start with each page being equally likely
    end for
    while $R$ has not converged do
        for all entries $R_i \in R$ do
            $R_i \leftarrow \lambda/|P|$ ▷ Each page has a $\lambda/|P|$ chance of random selection
        end for
        for all pages $p \in P$ do
            $Q \leftarrow$ the set of pages such that $(p, q) \in L$ and $q \in P$
            if $|Q| > 0$ then
                for all pages $q \in Q$ do
                    $R_q \leftarrow R_q + (1 - \lambda)I_p/|Q|$       ▷ Probability $I_p$ of being at page $p$
                end for
            else
                for all pages $q \in P$ do
                    $R_q \leftarrow R_q + (1 - \lambda)I_p/|P|$
                end for
            end if
            $I \leftarrow R$                         ▷ Update our current PageRank estimate
        end for
    end while
    return $R$
end procedure
```

$\boxed{|P| = N}$

$\boxed{R\text{-}I=0 \text{ or } |R\text{-}I| < \boldsymbol{\varepsilon}}$

$\boxed{Q = \{\, q \in P, (p, q) \in L \,\}}$

$\boxed{\text{for } (p, q) \in L}$

$\boxed{|P| >> |Q|}$

Ranking cont.

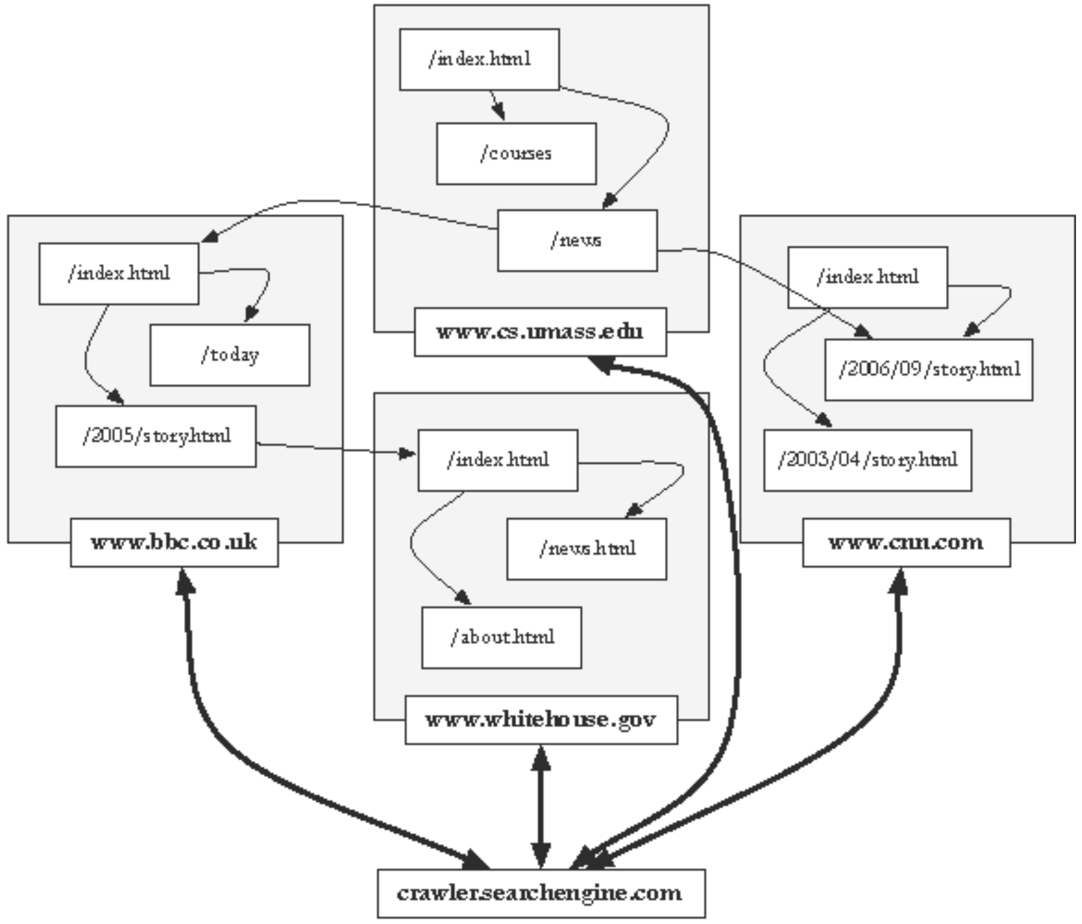
- Performance optimization
  - Designing ranking algorithms for efficient processing
    - *Term-at-a time* vs. *document-at-a-time* processing
    - *Safe* vs. *unsafe* optimizations
- Distribution
  - Processing queries in a distributed environment
  - *Query broker* distributes queries and assembles results
  - *Caching* is a form of distributed searching

Evaluation

- Logging
  - Logging user queries and interaction is crucial for improving search effectiveness and efficiency
  - *Query logs* and *clickthrough data* used for query suggestion, spell checking, query caching, ranking, advertising search, and other components
- Ranking analysis
  - Measuring and tuning ranking effectiveness
- Performance analysis
  - Measuring and tuning system efficiency

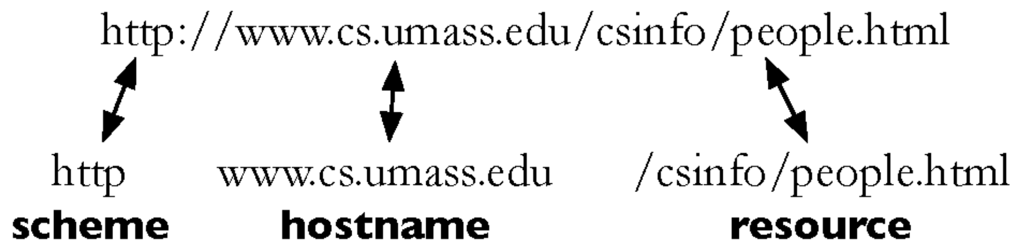
4. Web Crawler

- Finds URLs and downloads web pages automatically
  - provides the collection for searching
- **Web is huge and constantly growing**
- **Web is not under the control of search engine providers**
- Web pages are constantly changing
- Crawlers also used for other types of data
- Sites that are difficult for a crawler to find are collectively referred to as the *deep* (or *hidden*) *Web*
  - much larger than conventional Web



Retrieving Web Pages

- Every page has a unique *uniform resource locator* (URL) which uses to describe a web page in three parts: the scheme, the hostname and the resource name



- Web pages are stored on web servers that use HTTP to exchange information with client software.
- Most URLs used on the Web start with the scheme *http*, indicating that the URL represents a resource that can be retrieved using HTTP.
- The hostname follows, which is the name of the computer that is running the web server that holds this web page.
- An URL also refers to a page on that computer, e.g., /csinfo/people.html

Retrieving Web Pages cont.

- Web browsers and web crawlers are two different kinds of web clients, but both fetch web pages in the same way.
- Web crawler client program first connects to a *domain name system* (DNS) server.
- The DNS server translates the hostname into an *internet protocol* (IP) address (a number).
- Then the browser attempts to connect to a server computer with that IP address that will provide a specific port (e.g., port 80) for the new connection.
- After connection, crawler sends an HTTP request to the web server to request a page
    - usually, a GET request
    - GET /csinfo/people.html HTTP/1.0
- Once sending a short header, the Web server sends the contents of that file back to the client.

Web Crawler algorithm

- Starts with a set of *seeds*, which are a set of URLs given to it as parameters.
- Seeds are added to a URL *request queue.*
- Crawler starts fetching pages from the request queue.
- Downloaded pages are parsed to find link tags that might contain other useful URLs to fetch.
- New URLs are then added to the crawler's request queue, or *frontier which* may be a standard queue, or ordered so that important pages move to the front of the list.
- Continue until no more new URLs or disk full.

Web Crawling - Politeness Policies

- **Web crawlers spend a lot of time waiting for responses to requests**
- To reduce this inefficiency, web crawlers use threads and fetch hundreds of pages at once
- Crawlers could potentially flood sites with requests for pages
- To avoid this problem, web crawlers use *politeness policies*
  - e.g., delay between requests to same web server

Controlling Crawling

- Even crawling a site slowly will anger some web server administrators, who object to any copying of their data
- Robots.txt file can be used to control crawlers
  - The quick way to prevent robots visiting your site is put these two lines into the /robots.txt file on your server:

    User-agent: *

    Disallow: /

```
User-agent: *
Disallow: /private/
Disallow: /confidential/
Disallow: /other/
Allow: /other/public/

User-agent: FavoredCrawler
Disallow:

Sitemap: http://mysite.com/sitemap.xml.gz
```

Freshness

- Web pages are constantly being added, deleted, and modified
- Web crawler must continually revisit pages it has already crawled to see if they have changed in order to maintain the *freshness* of the document collection
    - *stale* copies no longer reflect the real contents of the web pages

Freshness

- HTTP protocol has a special request type called HEAD that makes it easy to check for page changes
  - returns information about page, not page itself

```
Client request:  HEAD /csinfo/people.html HTTP/1.1
                 Host: www.cs.umass.edu

                  HTTP/1.1 200 OK
                  Date: Thu, 03 Apr 2008 05:17:54 GMT
                  Server: Apache/2.0.52 (CentOS)
                  Last-Modified: Fri, 04 Jan 2008 15:28:39 GMT
Server response:  ETag: "239c33-2576-2a2837c0"
                  Accept-Ranges: bytes
                  Content-Length: 9590
                  Connection: close
                  Content-Type: text/html; charset=ISO-8859-1
```

Freshness

- Not possible to constantly check all pages
  - must check important pages and pages that change frequently
- *Freshness* is the **proportion** of pages that are fresh
- Optimizing for this metric can lead to bad decisions, such as not crawling popular sites
- *Age* is a better metric

Freshness vs. Age

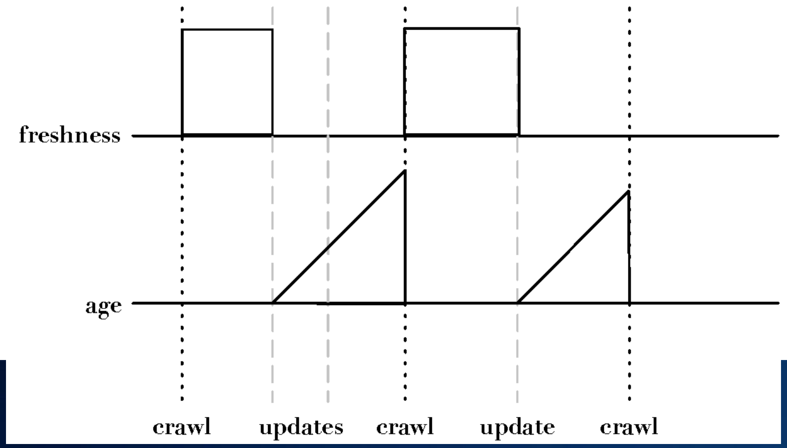
- Expected age of a page *t* days after it was last crawled (λ is the change frequency per day):

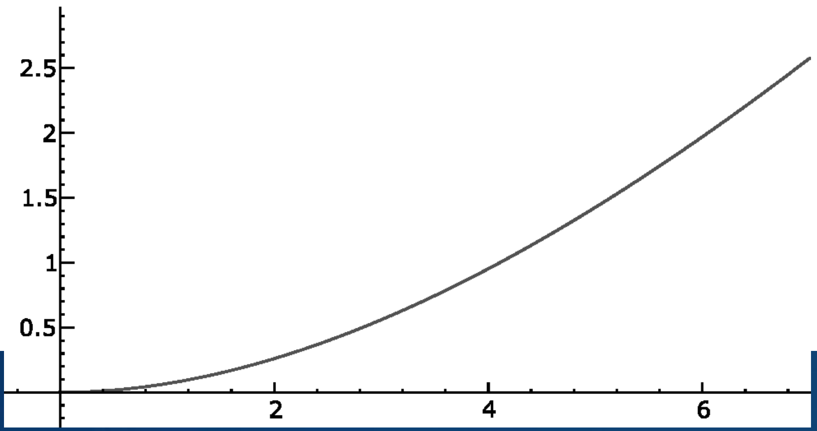
$$\text{Age}(\lambda, t) = \int_0^t P(\text{page changed at time } x)(t - x)dx$$

- Web page updates follow the Poisson distribution on average
  - time until the next update is governed by an exponential d

$$\text{Age}(\lambda, t) = \int_0^t \lambda e^{-\lambda x}(t - x)dx$$

- Older a page gets, the more it costs not to crawl it
  – e.g., expected age with mean change frequency *λ* = 1/7 (one change per week)





Focused Crawling

- Attempts to download only those pages that are about a particular topic
  - used by *vertical search* applications
- Rely on the fact that pages about a topic tend to have links to other pages on the same topic
  - popular pages for a topic are typically used as seeds
- Crawler uses *text classifier* to decide whether a page is on topic

Sitemaps

- Sitemaps contain lists of URLs and data about those URLs, such as modification time and modification frequency
- Generated by web server administrators
- Tells crawler about pages it might not otherwise find
- Gives crawler a hint about when to check a page for changes

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.company.com/</loc>
    <lastmod>2008-01-15</lastmod>
    <changefreq>monthly</changefreq>
    <priority>0.7</priority>
  </url>
  <url>
    <loc>http://www.company.com/items?item=truck</loc>
    <changefreq>weekly</changefreq>
  </url>
  <url>
    <loc>http://www.company.com/items?item=bicycle</loc>
    <changefreq>daily</changefreq>
  </url>
</urlset>
```

Document Feeds

- Many documents are *published*
  - created at a fixed time and rarely updated again
  - e.g., news articles, journal articles, blog posts, press releases, email
- Published documents from a single source can be ordered in a sequence called a *document feed*
  - new documents found by examining the end of the feed

- Two types:
  - A *push feed* alerts the subscriber to new documents
  - A *pull feed* requires the subscriber to check periodically for new documents
- Most common format for pull feeds is called *RSS*
  - Really Simple Syndication, RDF Site Summary, Rich Site Summary, or ...

Storing the Documents

- Many reasons to store converted document text
  - saves crawling time when page is not updated
  - provides efficient access to text for snippet generation, information extraction, etc.
- Database systems can provide document storage for some applications
  - web search engines use customized document storage systems

- Requirements for document storage system:
  - Random access
    - request the content of a document based on its URL
    - hash function based on URL is typical
  - Compression and large files
    - reducing storage requirements and efficient access
  - Update
    - handling large volumes of new and modified documents
    - adding new anchor text

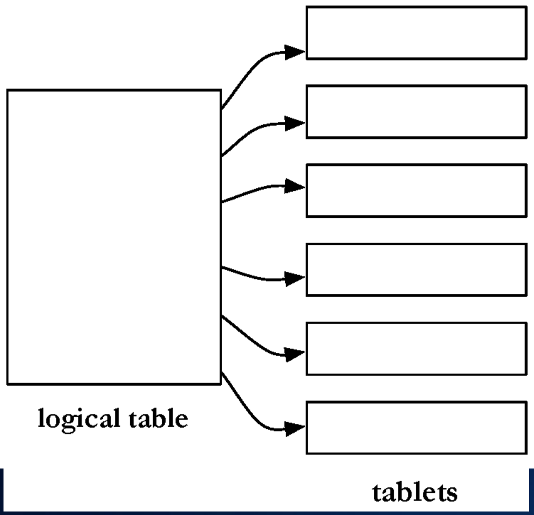
Large Files

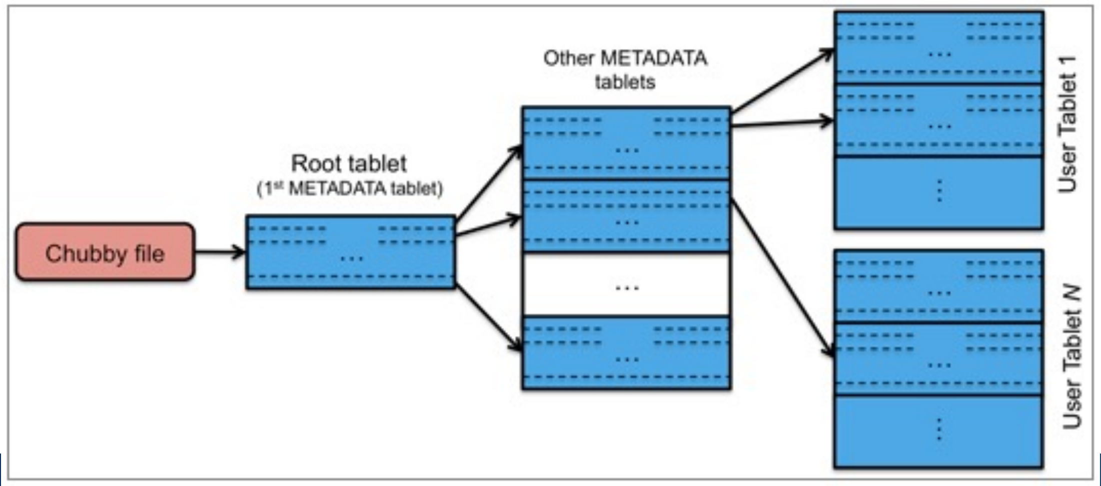
- Store many documents in large files, rather than each document in a file
  - avoids overhead in opening and closing files
  - reduces seek time relative to read time
- Compound documents formats
  - used to store multiple documents in a file
  - e.g., TREC Web

- Compression
  - Text is highly redundant (or predictable)
  - Compression techniques exploit this redundancy to make files smaller without losing any of the content
  - Compression of indexes covered later
  - Popular algorithms can compress HTML and XML text by 80%
    - e.g., DEFLATE (zip, gzip) and LZW (UNIX compress, PDF)
    - may compress large files in blocks to make access faster

BigTable

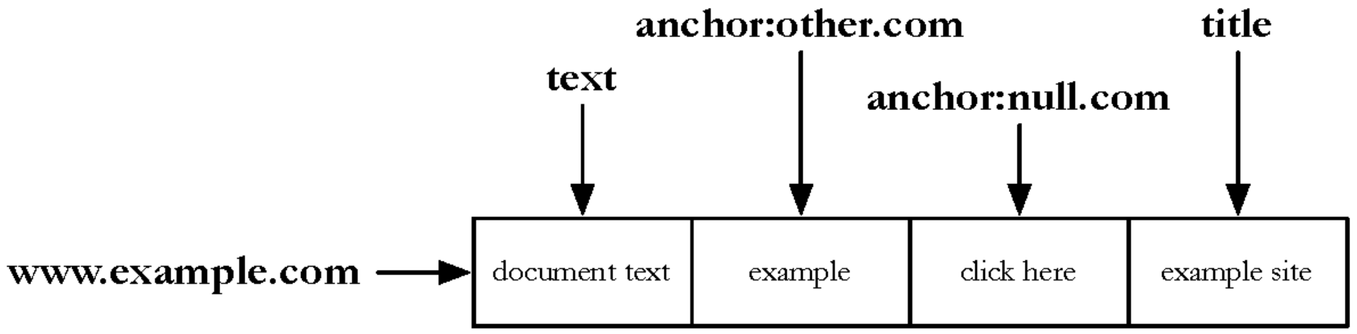
- Google’s document storage system
  - Customized for storing, finding, and updating web pages
  - Handles large collection sizes using inexpensive computers
- No query language, no complex queries to optimize
- Only row-level transactions
- Tablets are stored in a replicated file system that is accessible by all BigTable servers
- Any changes to a BigTable tablet are recorded to a transaction log, which is also stored in a shared file system
- If any tablet server crashes, another server can immediately read the tablet data and transaction log from the file system and take over





BigTable row

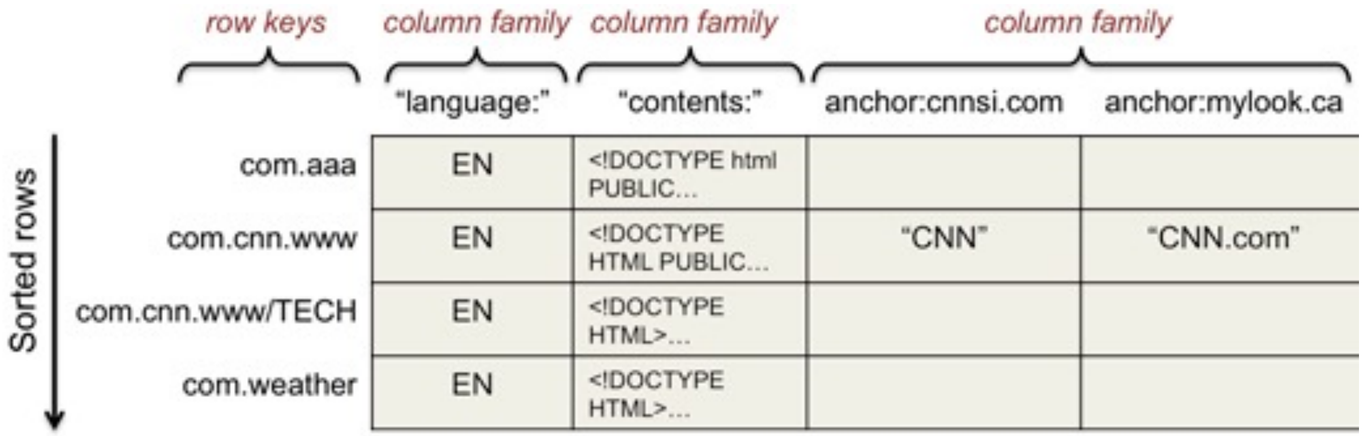
- Logically organized into rows
- A row stores data for a single web page



- Combination of a row key, a column key, and a timestamp point to a single *cell* in the row

BigTable row cont.

- BigTable can have a huge number of columns per row
  - all rows have the same column groups (families)
  - not all rows have the same columns
  - important for reducing disk reads to access document data
- Rows are partitioned into tablets based on their row keys
  - simplifies determining which server is appropriate


row keys
column family
column family
column family
Sorted rows

	"language:"	"contents:"	anchor:cnnsi.com	anchor:mylook.ca
com.aaa	EN	<!DOCTYPE html PUBLIC...		
com.cnn.www	EN	<!DOCTYPE HTML PUBLIC...	"CNN"	"CNN.com"
com.cnn.www/TECH	EN	<!DOCTYPE HTML>...		
com.weather	EN	<!DOCTYPE HTML>...		

Detecting Duplicates

- Duplicate and near-duplicate documents occur in many situations
  - Copies, versions, plagiarism, spam, mirror sites
  - 30% of the web pages in a large crawl are exact or near duplicates of pages in the other 70%
- Duplicates consume significant resources during crawling, indexing, and search
  - Little value to most users

Duplicate Detection

- *Exact* duplicate detection is relatively easy
- *Checksum* techniques
  - A checksum is a value that is computed based on 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

Near-Duplicate Detection

- 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

Near-Duplicate Detection

- *Search*:
  - find near-duplicates of a document *D*
  - *O(N)* comparisons required
- *Discovery*:
  - find all pairs of near-duplicate documents in the collection
  - $O(N^2)$ comparisons
- IR techniques are effective for search scenario
- For discovery, other techniques used to generate compact representations

Fingerprints

1. The document is parsed into words. Non-word content, such as punctuation, HTML tags, and additional whitespace, is removed.
2. The words are grouped into contiguous *n-grams* for some *n*. These are usually overlapping sequences of words, although some techniques use non-overlapping sequences.
3. Some of the n-grams are selected to represent the document.
4. The selected n-grams are hashed to improve retrieval efficiency and further reduce the size of the representation.
5. The hash values are stored, typically in an inverted index.
6. Documents are compared using overlap of fingerprints

Fingerprint Example

```
>>> otext ='Tropical fish include fish found in tropical environments around the world including both freshwater and salt
water species'
>>> otext=otext.split()
>>> trigrams = [otext[i]+' '+otext[i+1]+' '+otext[i+2] for i in range(len(otext)-2)]
>>> otext
['Tropical', 'fish', 'include', 'fish', 'found', 'in', 'tropical', 'environments', 'around', 'the', 'world', 'including', 'both', 'freshwater',
'and', 'salt', 'water', 'species']
>>> trigrams
['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']
```

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

(c) Hash values

664 492 236 908

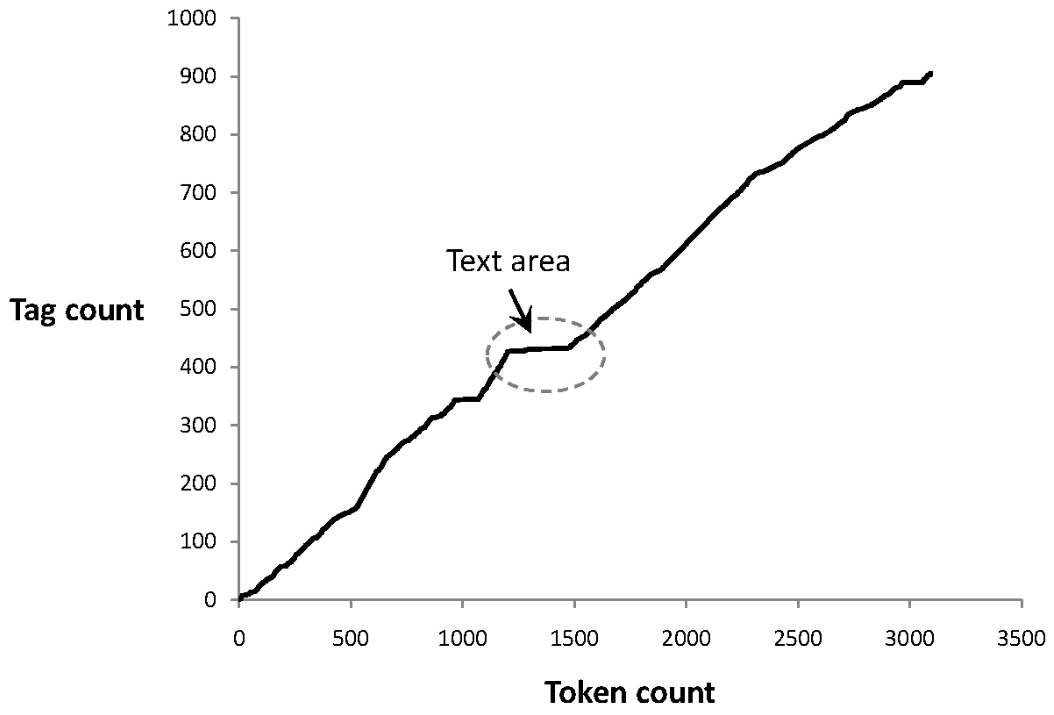
(d) Selected hash values using *0 mod 4*

Removing Noise

- Many web pages contain text, links, and pictures that are not directly related to the main content of the page
- This additional material is mostly *noise* that could negatively affect the ranking of the page
- Techniques have been developed to detect the content blocks in a web page
  - Non-content material is either ignored or reduced in importance in the indexing process

Finding Content Blocks

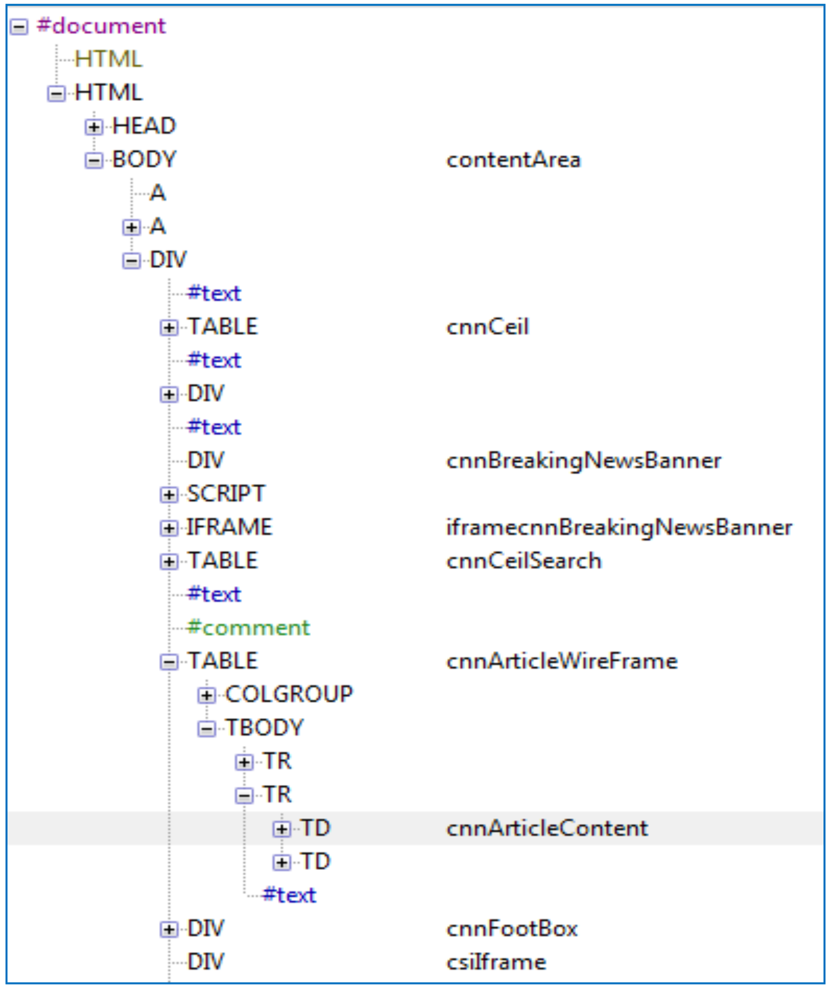
- Cumulative distribution of tags in the example web page



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

Finding Content Blocks cont.

- The structure of the web page can also be used more directly to identify the content blocks in the page
- An HTML parser can interpret the structure of the page specified using the tags, and creates a Document Object Model (DOM) representation (a tree like structure)
- Another method is to use DOM structure and visual (layout) features to identify the major components of the web page



```
⊟ #document
    HTML
  ⊟ HTML
    ⊞ HEAD
    ⊟ BODY                              contentArea
        A
      ⊞ A
      ⊟ DIV
          #text
        ⊞ TABLE                         cnnCeil
          #text
        ⊞ DIV
          #text
          DIV                           cnnBreakingNewsBanner
        ⊞ SCRIPT
        ⊞ IFRAME                        iframecnnBreakingNewsBanner
        ⊞ TABLE                         cnnCeilSearch
          #text
          #comment
        ⊟ TABLE                         cnnArticleWireFrame
          ⊞ COLGROUP
          ⊟ TBODY
            ⊞ TR
            ⊟ TR
              ⊞ TD                      cnnArticleContent
              ⊞ TD
                #text
        ⊞ DIV                           cnnFootBox
          DIV                           csiIframe
```

References

- Chapter 2, Chapter 3 and Chapter 4 (section 4.5) in text-book - W. Bruce Croft, *Search Engines - Information retrieval in Practice*; Pearson, 2010.
- Sergey Brin, Lawrence Page, “The anatomy of a large-scale hypertextual Web search engine”, Computer Networks and ISDN Systems, Volume 30, Issues 1–7, 1998, Pages 107-117.