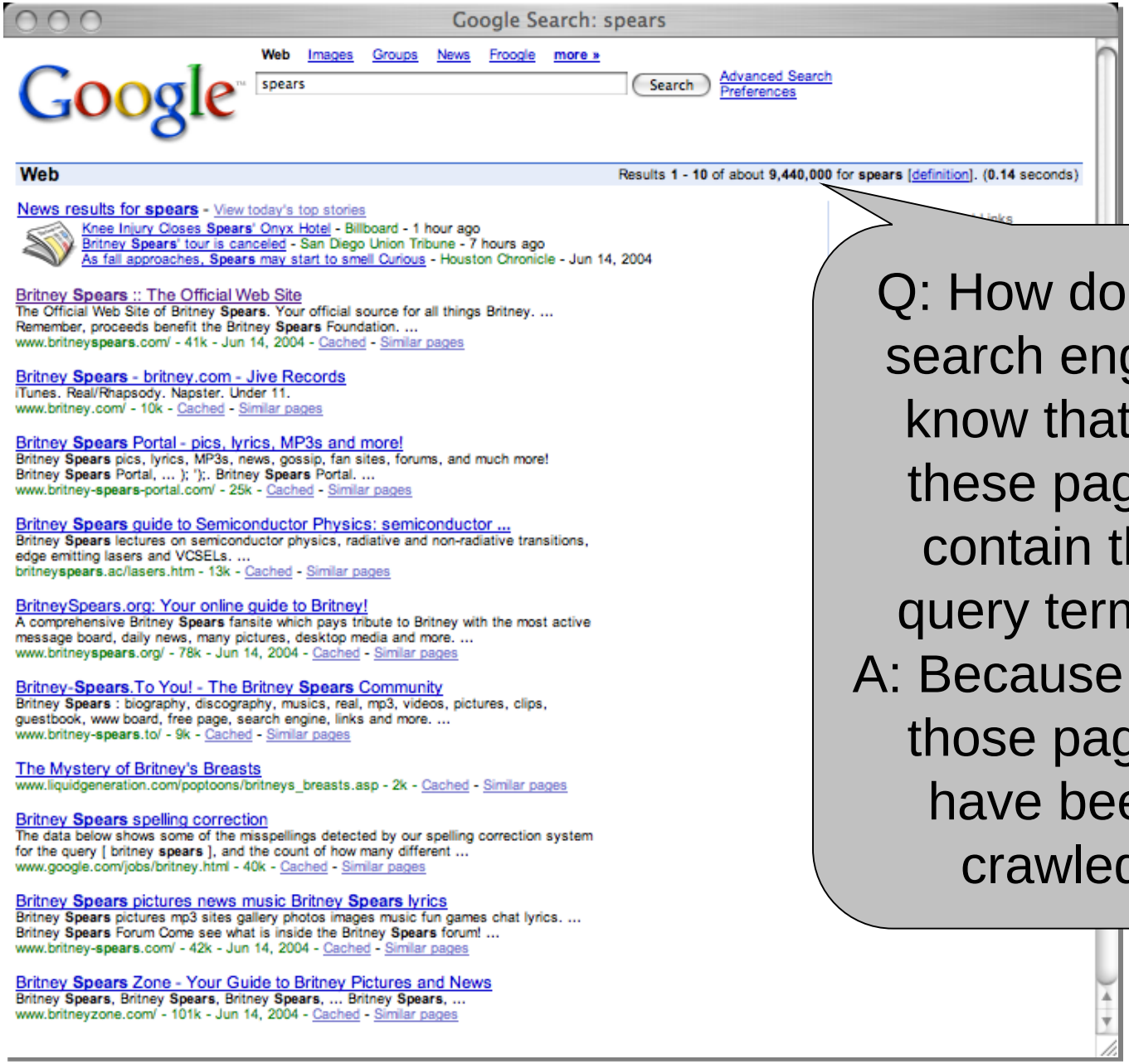
Chapter 8 - Web Crawling

Web Mining Lecture
Computer Engineering Department, Faculty of Electric and Electronics
Yıldız Technical University
Instructor: Dr. Mehmet Aktaş

Acknowledgement: Thanks to Dr. Fillippo Menczer and Dr. Bing Lui for Teaching Materials
Slides provided By Dr. Filippo Menczer
Indiana University School of Informatics

Outline

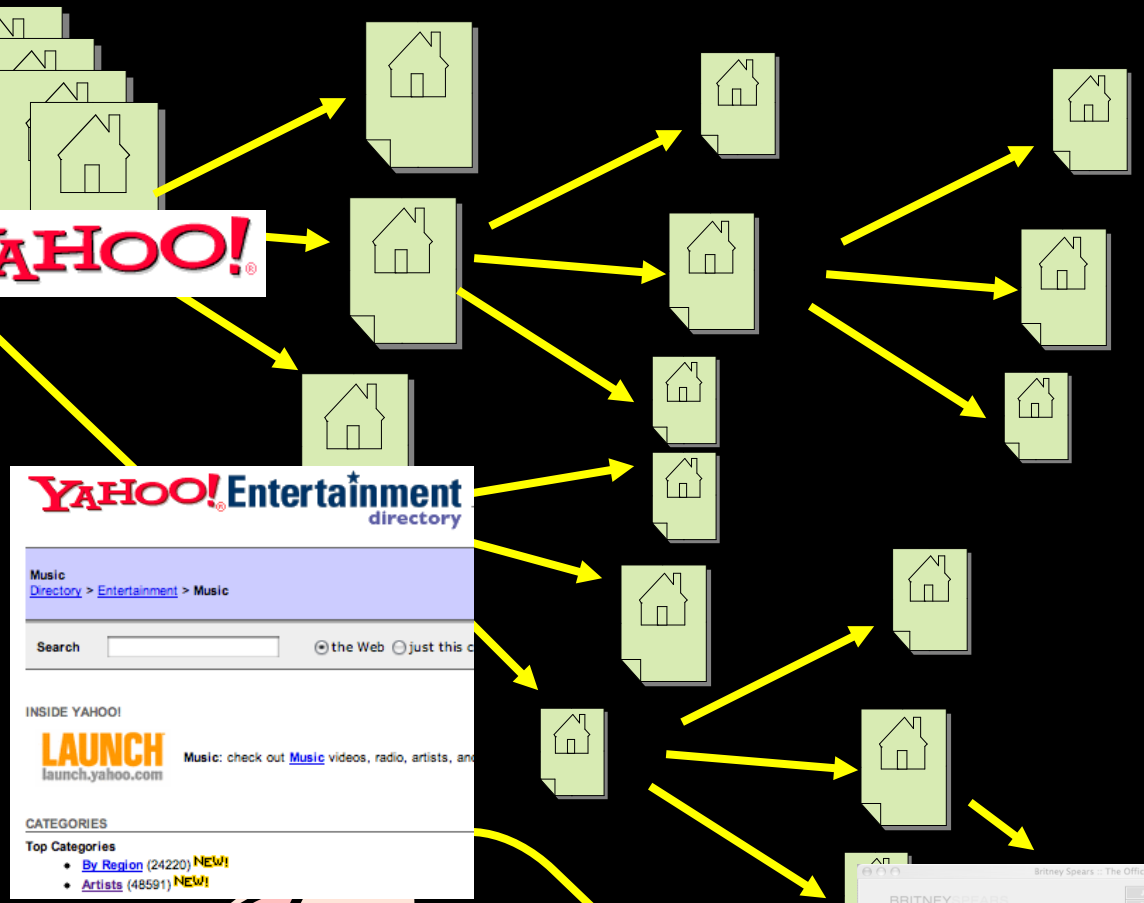
- Motivation and taxonomy of crawlers
- Basic crawlers and implementation issues
- Universal crawlers
- Preferential (focused and topical) crawlers

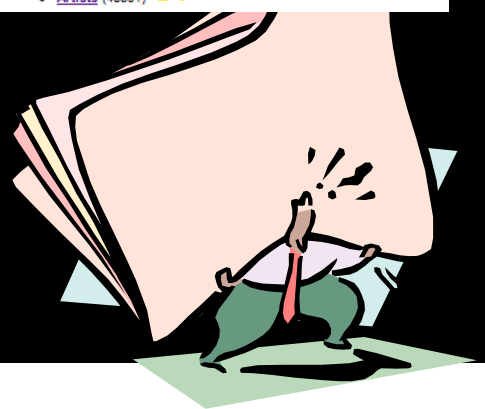


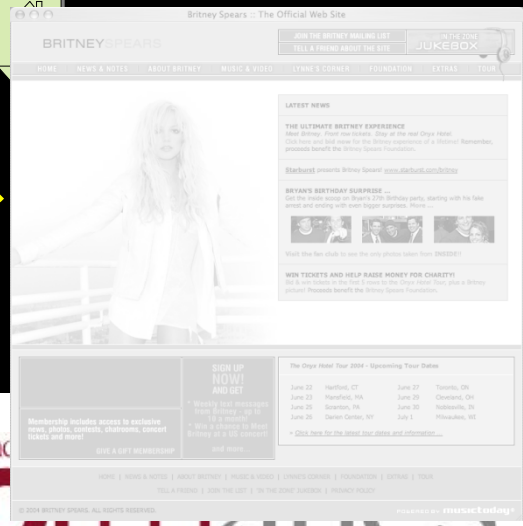
Q: How does a search engine know that all these pages contain the query terms?
A: Because all of those pages have been crawled

Bing Liu: *Web Data Mining*. Springer, 2007
Ch. 8 *Web Crawling* by Filippo Menczer

Crawler: basic idea







Bing Liu: *Web Data Mining*. Springer, 2007
Ch. 8 *Web Crawling* by Filippo Menczer

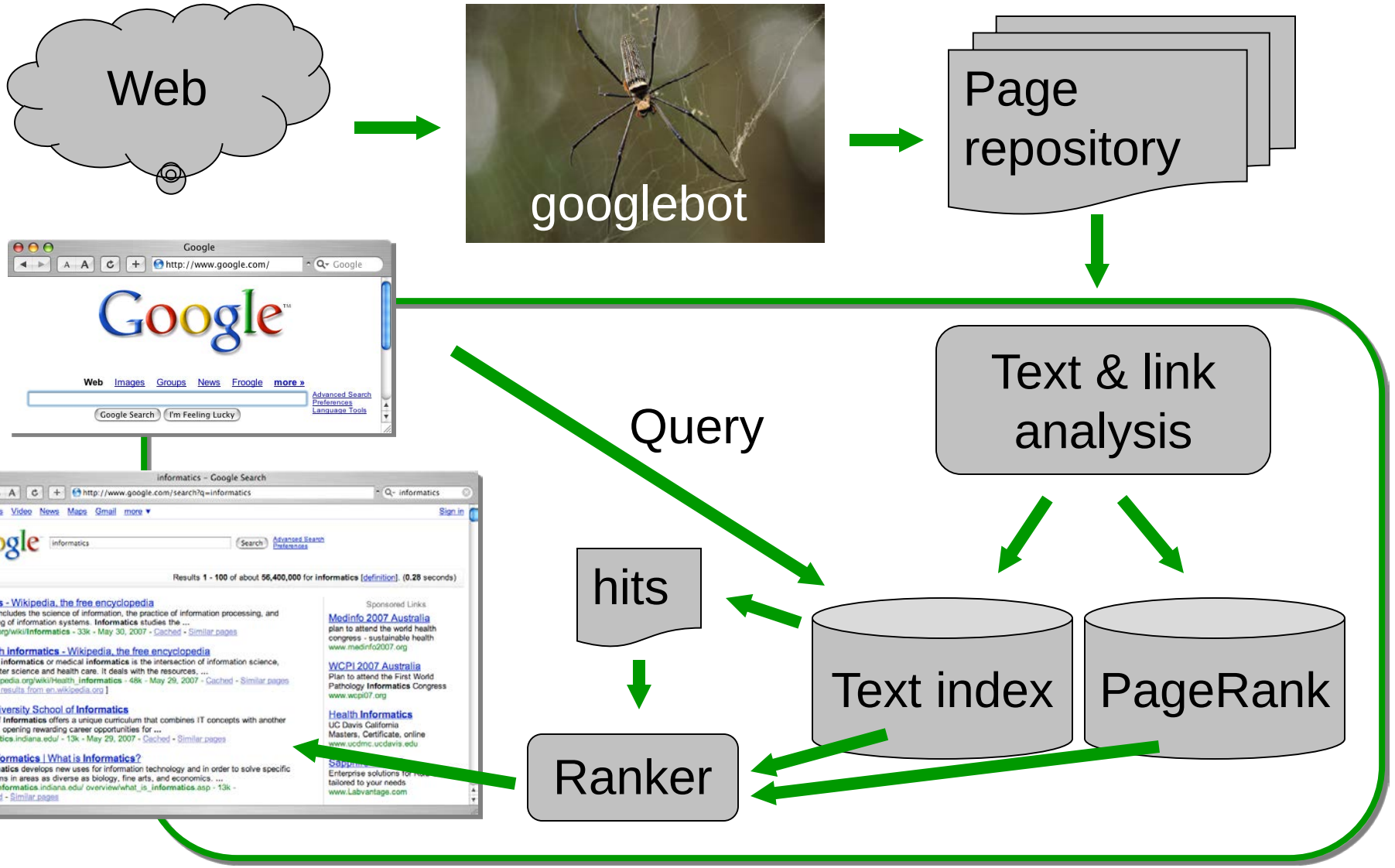
Many names

- Crawler
- Spider
- Robot (or bot)
- Web agent
- Wanderer, worm, …
- And famous instances: googlebot, scooter, slurp, msnbot, …

Motivation for crawlers

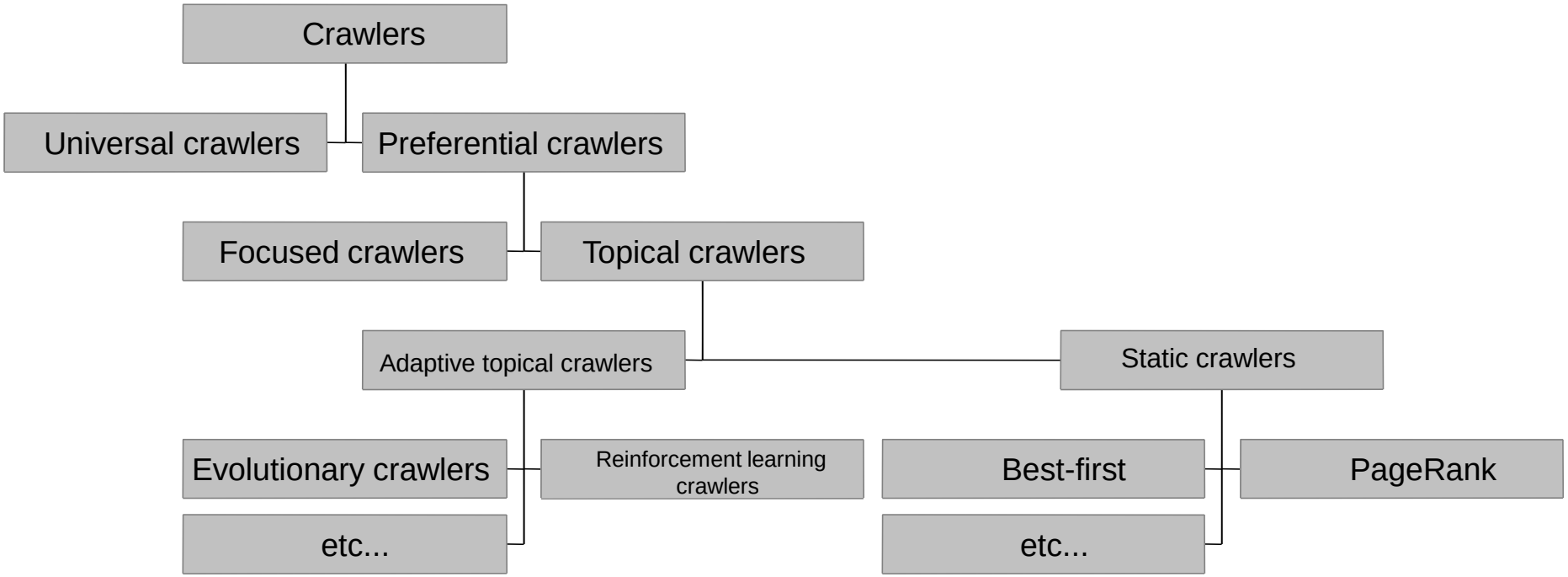
- Support universal search engines (Google, Yahoo, MSN/Windows Live, Ask, etc.)
- Vertical (specialized) search engines, e.g. news, shopping, papers, recipes, reviews, etc.
- Business intelligence: keep track of potential competitors, partners
- Monitor Web sites of interest
- Evil: harvest emails for spamming, phishing…
- … Can you think of some others?…

A crawler within a search engine



Bing Liu: *Web Data Mining*. Springer, 2007
Ch. 8 *Web Crawling* by Filippo Menczer

One taxonomy of crawlers

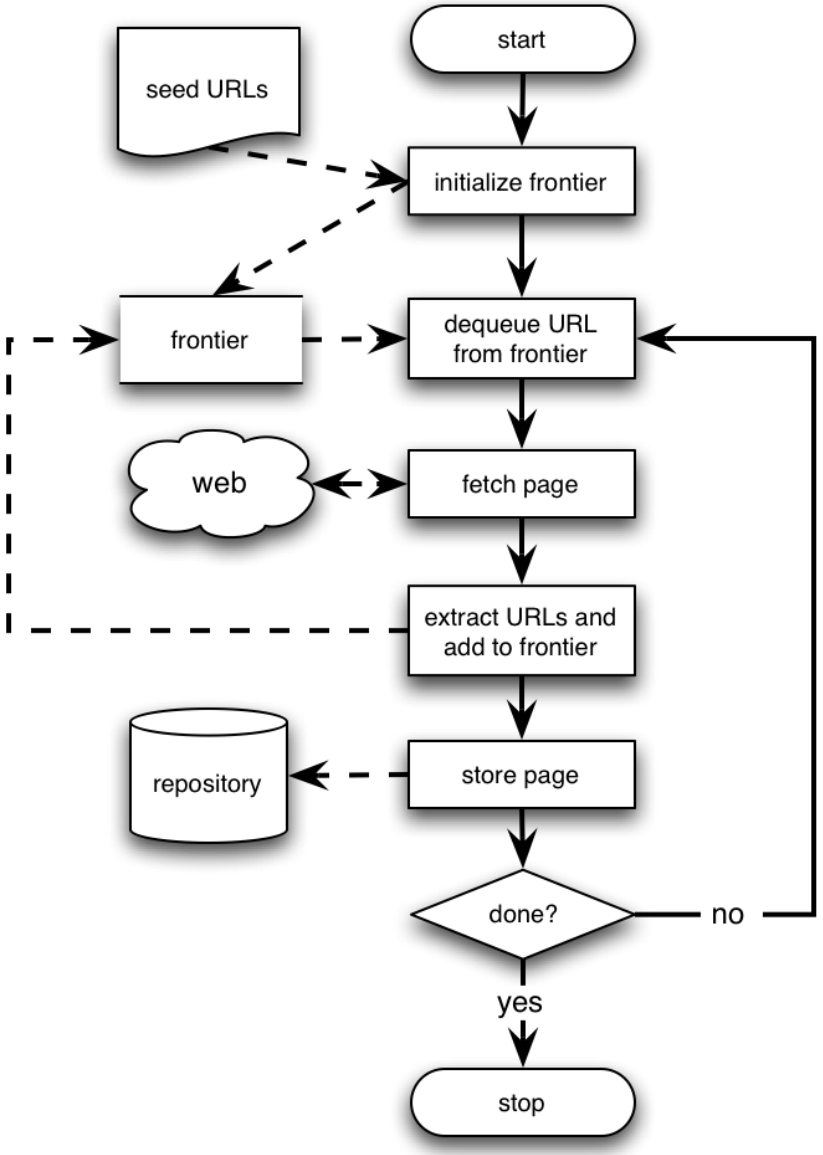


- Many other criteria could be used:
  - Incremental, Interactive, Concurrent, Etc.

Bing Liu: *Web Data Mining*. Springer, 2007
Ch. 8 *Web Crawling* by Filippo Menczer

Outline

- Motivation and taxonomy of crawlers
- Basic crawlers and implementation issues
- Universal crawlers
- Preferential (focused and topical) crawlers

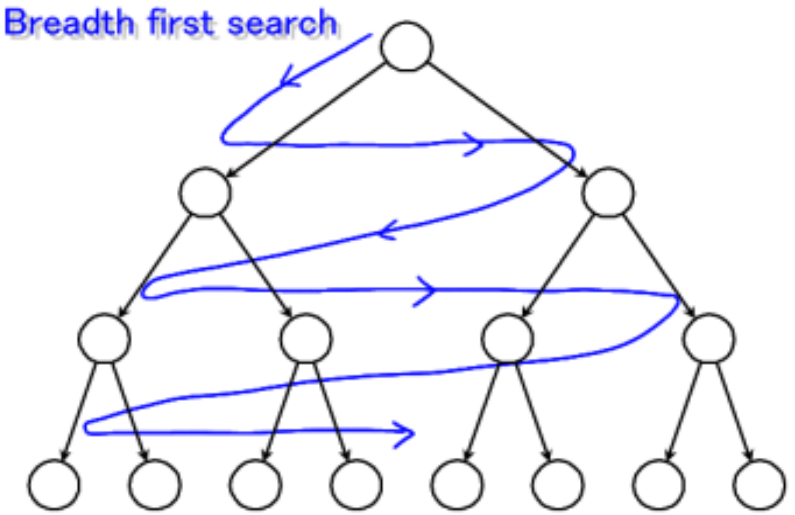


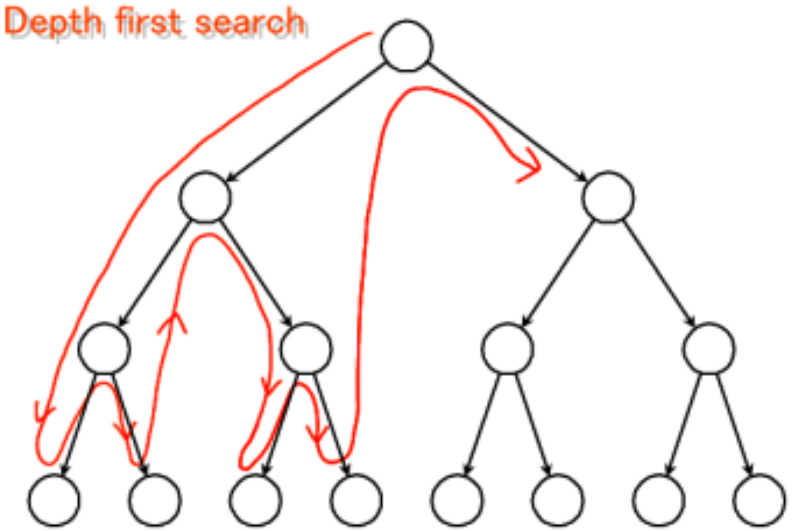
Basic crawlers

- This is a sequential crawler
- Seeds can be any list of starting URLs
- Order of page visits is determined by frontier data structure
- Stop criterion can be anything

Graph traversal (BFS or DFS?)

- Breadth First Search
  - Implemented with QUEUE (FIFO)
  - Finds pages along shortest paths
  - If we start with "good" pages, this keeps us close; maybe other good stuff…
- Depth First Search
  - Implemented with STACK (LIFO)
  - Wander away ("lost in cyberspace")





A basic crawler in Perl

- Queue: a FIFO list (shift and push)

```
my @frontier = read_seeds($file);
while (@frontier && $tot < $max) {
    my $next_link = shift @frontier;
    my $page = fetch($next_link);
    add_to_index($page);
    my @links = extract_links($page, $next_link);
    push @frontier, process(@links);
}
```

Implementation issues

- Don't want to fetch same page twice!
  - Keep lookup table (hash) of visited pages
  - What if not visited but in frontier already?
- The frontier grows very fast!
  - May need to prioritize for large crawls
- Fetcher must be robust!
  - Don't crash if download fails
  - Timeout mechanism
- Determine file type to skip unwanted files
  - Can try using extensions, but not reliable
  - Can issue 'HEAD' HTTP commands to get Content-Type (MIME) headers, but overhead of extra Internet requests

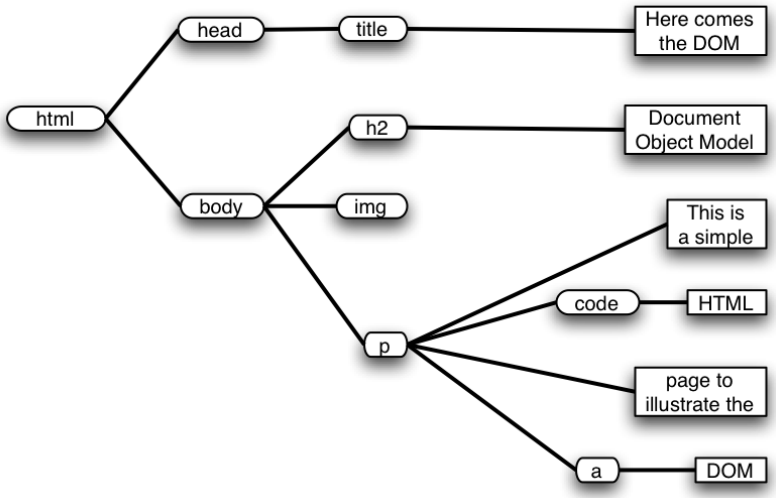
More implementation issues

- Fetching
  - Get only the first 10-100 KB per page
  - Soft fail for timeout, server not responding, file not found, and other errors

More implementation issues: Parsing

- HTML has the structure of a DOM (Document Object Model) tree
- Unfortunately actual HTML is often incorrect in a strict syntactic sense
- Crawlers, like browsers, must be robust/forgiving
- Fortunately there are tools that can help
  - E.g. tidy.sourceforge.net
- Must pay attention to HTML entities and unicode in text
- What to do with a growing number of other formats?
  - Flash, RSS, AJAX…

```
<html>
  <head>
    <title>Here comes the DOM</title>
  </head>
  <body>
    <h2>Document Object Model</h2>
    <img align="right" alt="dom pict" src="dom.png">
    <p>
      This is a simple
      <code>HTML</code>
      page to illustrate the
      <a href="http://www.w3.org/DOM/">DOM</a>
    </p>
  </body>
</html>
```



Bing Liu: *Web Data Mining*. Springer, 2007
Ch. 8 *Web Crawling* by Filippo Menczer

More implementation issues

- Stop words
  - Noise words that do not carry meaning should be eliminated ("stopped") before they are indexed
  - E.g. in English: AND, THE, A, AT, OR, ON, FOR, etc…
  - Typically syntactic markers
  - Typically the most common terms
  - Typically kept in a negative dictionary
    - 10–1,000 elements
    - E.g. http://ir.dcs.gla.ac.uk/resources/linguistic_utils/stop_words
  - Parser can detect these right away and disregard them

More implementation issues

Conflation and synonyms

- Idea: improve recall by merging words with same meaning

1. We want to ignore superficial morphological features, thus merge semantically similar tokens
   - {student, study, studying, studious} => studi
2. We can also conflate synonyms into a single form using a thesaurus
   - 30-50% smaller index
   - Doing this in both pages and queries allows to retrieve pages about 'automobile' when user asks for 'car'
   - Thesaurus can be implemented as a hash table

More implementation issues

- Stemming
  - Morphological conflation based on rewrite rules
  - Language dependent!
  - Porter stemmer very popular for English
    - http://www.tartarus.org/~martin/PorterStemmer/
    - Context-sensitive grammar rules, eg:
      - "IES" except ("EIES" or "AIES") --> "Y"
    - Versions in Perl, C, Java, Python, C#, Ruby, PHP, etc.
  - Porter has also developed Snowball, a language to create stemming algorithms in any language
    - http://snowball.tartarus.org/
    - Ex. Perl modules: Lingua::Stem and Lingua::Stem::Snowball

More implementation issues

- Static vs. dynamic pages
  - Is it worth trying to eliminate dynamic pages and only index static pages?
  - Examples:
    - http://www.census.gov/cgi-bin/gazetteer
    - http://informatics.indiana.edu/research/colloquia.asp
    - http://www.amazon.com/exec/obidos/subst/home/home.html/002-8332429-6490452
    - http://www.imdb.com/Name?Menczer,+Erico
    - http://www.imdb.com/name/nm0578801/
  - Why or why not? How can we tell if a page is dynamic? What about 'spider traps'?
  - What do Google and other search engines do?

More implementation issues

- Relative vs. Absolute URLs
  - Crawler must translate relative URLs into absolute URLs
  - Need to obtain Base URL from HTTP header, or HTML Meta tag, or else current page path by default
  - Examples
    - **Base: http://www.cnn.com/linkto/**
    - **Relative URL: intl.html**
    - **Absolute URL: http://www.cnn.com/linkto/intl.html**
    - **Relative URL: /US/**
    - **Absolute URL: http://www.cnn.com/US/**

More implementation issues

- URL canonicalization
  - All of these:
    - http://www.cnn.com/TECH
    - http://WWW.CNN.COM/TECH/
    - http://www.cnn.com:80/TECH/
    - http://www.cnn.com/bogus/../TECH/
  - Are really equivalent to this canonical form:
    - **http://www.cnn.com/TECH/**
  - In order to avoid duplication, the crawler must transform all URLs into canonical form
  - Definition of "canonical" is arbitrary, e.g.:
    - Could always include port
    - Or only include port when not default :80

More on Canonical URLs

- Some transformation are trivial, for example:
  - × http://informatics.indiana.edu
  - ✓ http://informatics.indiana.edu/
  - × http://informatics.indiana.edu/index.html#fragment
  - ✓ http://informatics.indiana.edu/index.html
  - × http://informatics.indiana.edu/dir1/./../dir2/
  - ✓ http://informatics.indiana.edu/dir2/
  - × http://informatics.indiana.edu/%7Efil/
  - ✓ http://informatics.indiana.edu/~fil/
  - × http://INFORMATICS.INDIANA.EDU/fil/
  - ✓ http://informatics.indiana.edu/fil/

More implementation issues

- Spider traps
  - Misleading sites: indefinite number of pages dynamically generated by CGI scripts
  - Paths of arbitrary depth created using soft directory links and path rewriting features in HTTP server
  - Only heuristic defensive measures:
    - Check URL length; assume spider trap above some threshold, for example 128 characters
    - Watch for sites with very large number of URLs
    - Eliminate URLs with non-textual data types
    - May disable crawling of dynamic pages, if can detect

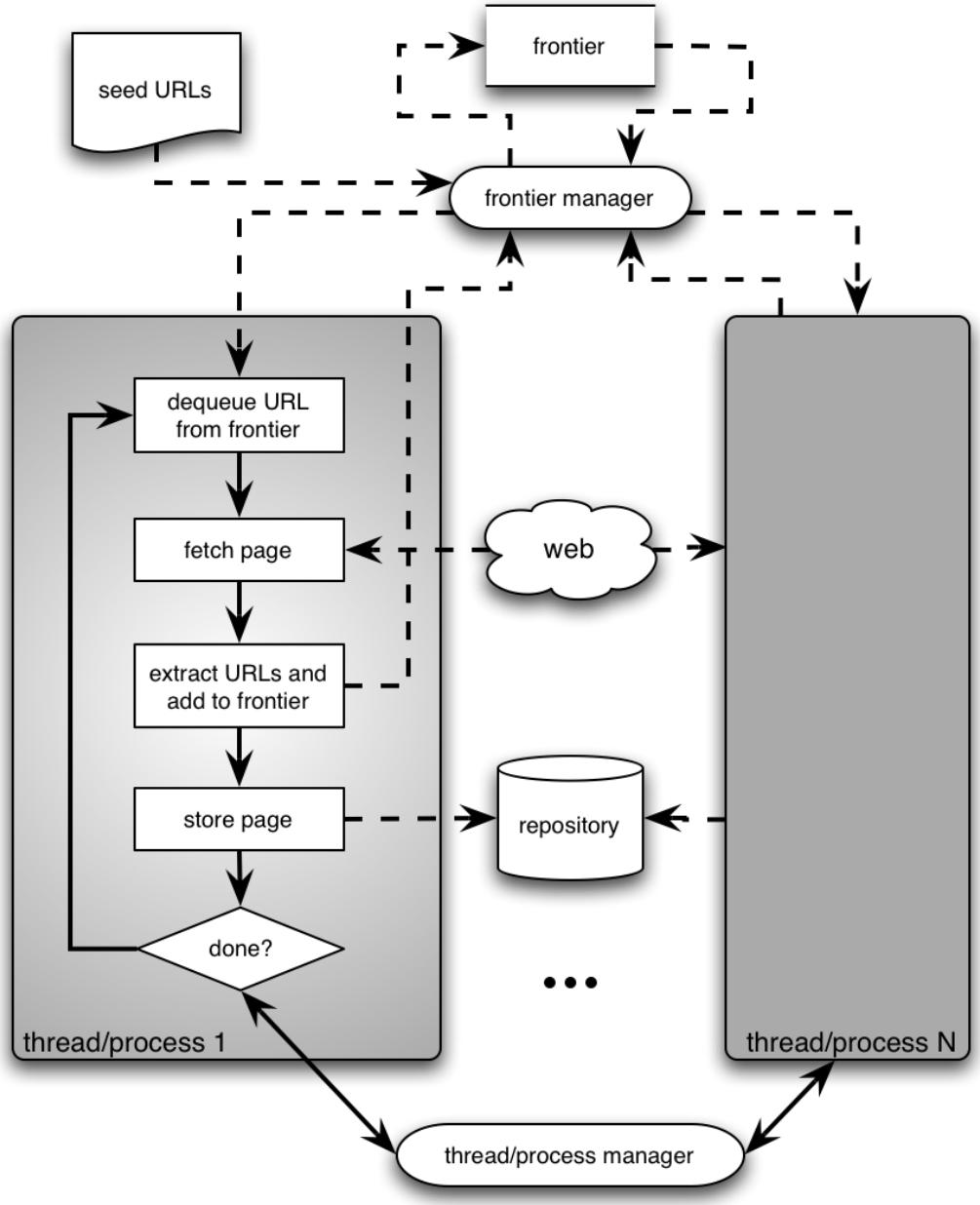
More implementation issues

- Page repository
  - Naïve: store each page as a separate file
    - Can map URL to unique filename using a hashing function, e.g. MD5
    - This generates a huge number of files, which is inefficient from the storage perspective
  - Better: combine many pages into a single large file, using some XML markup to separate and identify them
    - Must map URL to {filename, page_id}
  - Database options
    - Any RDBMS -- large overhead
    - Light-weight, embedded databases such as Berkeley DB

Concurrency

- A crawler incurs several delays:
  - Resolving the host name in the URL to an IP address using DNS
  - Connecting a socket to the server and sending the request
  - Receiving the requested page in response
- Solution: Overlap the above delays by fetching many pages concurrently

Architecture of a concurrent crawler



Bing Liu: *Web Data Mining*. Springer, 2007
Ch. 8 *Web Crawling* by Filippo Menczer

Concurrent crawlers

- Can use multi-processing or multi-threading
- Each process or thread works like a sequential crawler, except they share data structures: frontier and repository
- Shared data structures must be synchronized (locked for concurrent writes)
- Speedup of factor of 5-10 are easy this way

Outline

- Motivation and taxonomy of crawlers
- Basic crawlers and implementation issues
- Universal crawlers
- Preferential (focused and topical) crawlers

Universal crawlers

- Support universal search engines
- Large-scale
- Huge cost (network bandwidth) of crawl is amortized over many queries from users
- Incremental updates to existing index and other data repositories

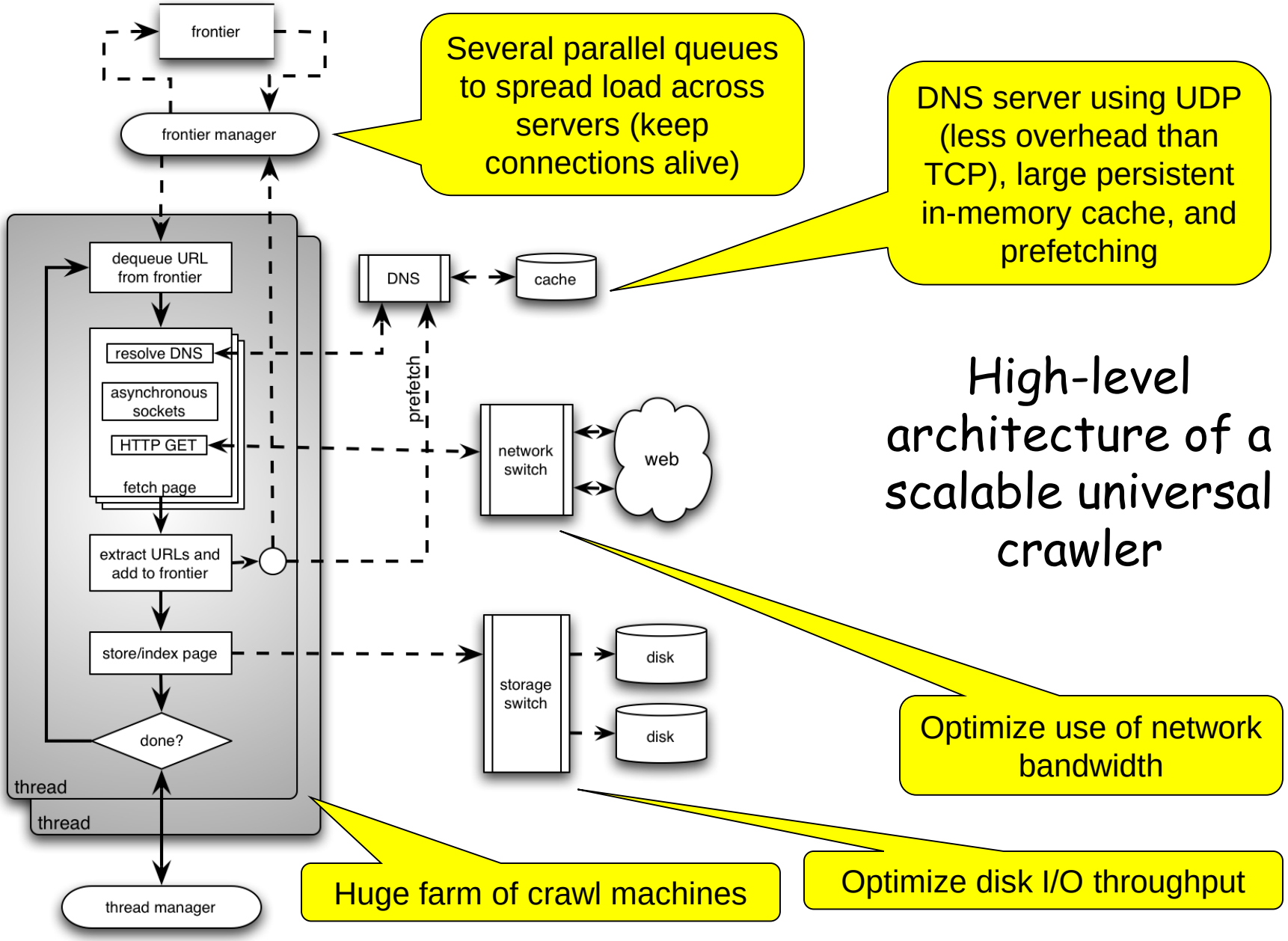
Large-scale universal crawlers

- Two major issues:

1. Performance
   - Need to scale up to billions of pages
2. Policy
   - Need to trade-off coverage, freshness, and bias (e.g. toward "important" pages)

Large-scale crawlers: scalability

- Need to minimize overhead of DNS lookups
- Need to optimize utilization of network bandwidth and disk throughput (I/O is bottleneck)
- Use asynchronous sockets
  - Multi-processing or multi-threading do not scale up to billions of pages
  - Non-blocking: hundreds of network connections open simultaneously
  - Polling socket to monitor completion of network transfers

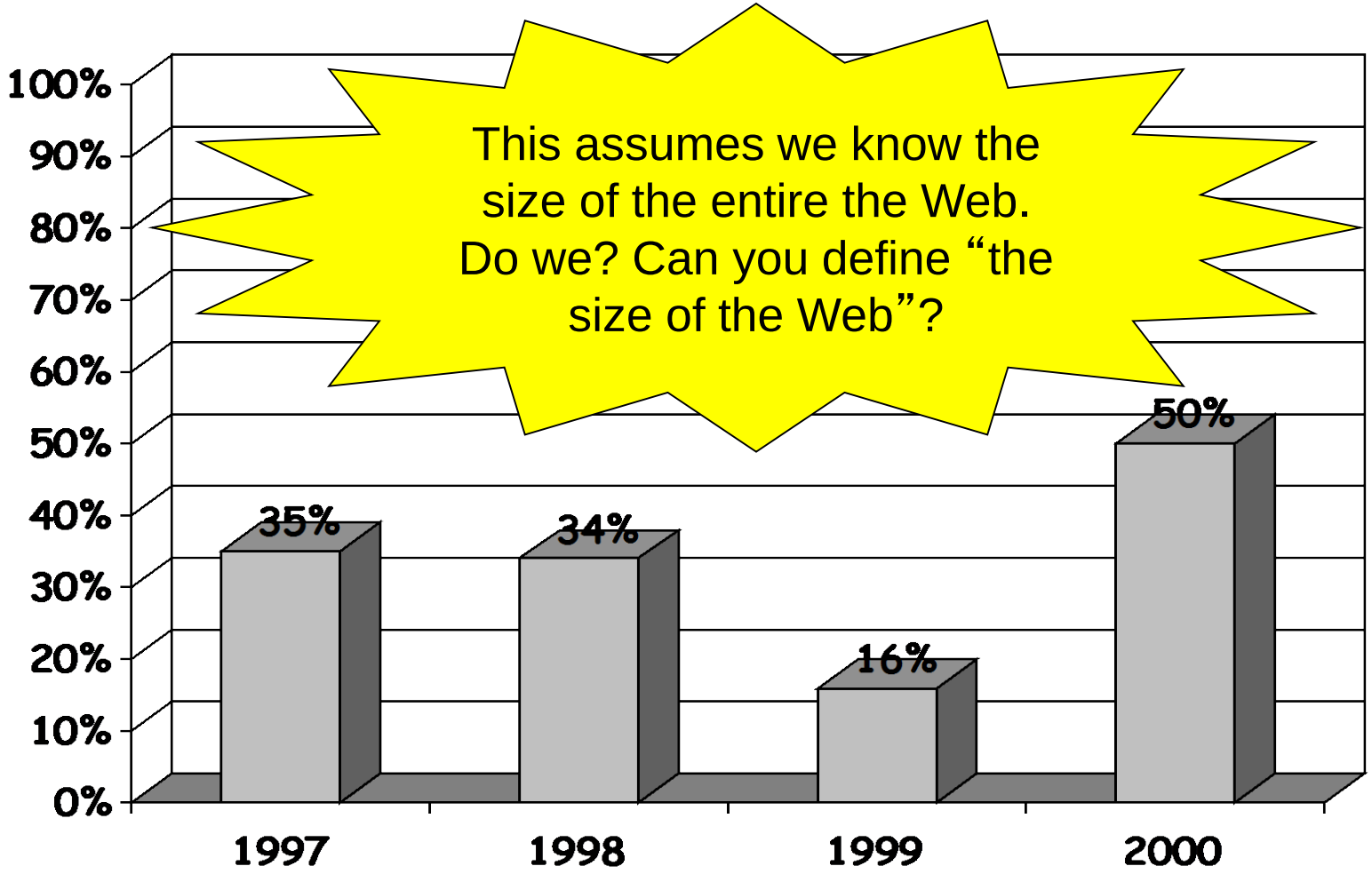


Bing Liu: *Web Data Mining*. Springer, 2007
Ch. 8 *Web Crawling* by Filippo Menczer

Universal crawlers: Policy

- Coverage
  - New pages get added all the time
  - Can the crawler find every page?
- Freshness
  - Pages change over time, get removed, etc.
  - How frequently can a crawler revisit ?
- Trade-off!
  - Focus on most "important" pages (crawler bias)?
  - "Importance" is subjective

Web coverage by search engine crawlers



Bing Liu: *Web Data Mining*. Springer, 2007
Ch. 8 *Web Crawling* by Filippo Menczer

Maintaining a "fresh" collection

- Universal crawlers are never "done"
- High variance in rate and amount of page changes
- HTTP headers are notoriously unreliable
  - Last-modified
  - Expires
- Solution
  - Estimate the probability that a previously visited page has changed in the meanwhile
  - Prioritize by this probability estimate

Estimating page change rates

- Algorithms for maintaining a crawl in which most pages are fresher than a specified epoch
  - Brewington & Cybenko; Cho, Garcia-Molina & Page
- Assumption: recent past predicts the future (Ntoulas, Cho & Olston 2004)
  - Frequency of change not a good predictor
  - Degree of change is a better predictor

Do we need to crawl the entire Web?

- If we cover too much, it will get stale
- There is an abundance of pages in the Web
- For PageRank, pages with very low prestige are largely useless
- What is the goal?
  - General search engines: pages with high prestige
  - News portals: pages that change often
  - Vertical portals: pages on some topic
- What are appropriate priority measures in these cases? Approximations?

Outline

- Motivation and taxonomy of crawlers
- Basic crawlers and implementation issues
- Universal crawlers
- Preferential (focused and topical) crawlers

Preferential crawlers

- Assume we can estimate for each page an importance measure, I(p)
- Want to visit pages in order of decreasing I(p)
- Maintain the frontier as a priority queue sorted by I(p)

Preferential crawlers

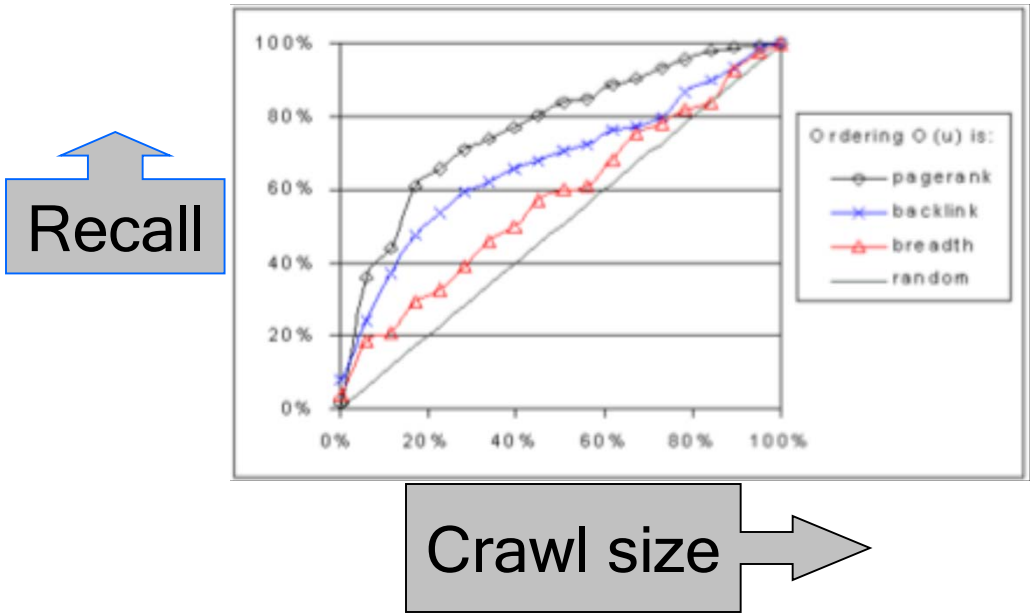
- Selective bias toward some pages, eg. most "relevant"/topical, closest to seeds, most popular/largest PageRank, unknown servers, highest rate/amount of change, etc…
- Focused crawlers
  - Supervised learning: classifier based on labeled examples
- Topical crawlers
  - Best-first search based on similarity(topic, parent)
  - Adaptive crawlers
    - Reinforcement learning
    - Evolutionary algorithms/artificial life

Preferential crawling algorithms: Examples

- Breadth-First
  - Exhaustively visit all links in order encountered
- Best-N-First
  - Priority queue sorted by similarity, explore top N at a time
- PageRank
  - Priority queue sorted by keywords, PageRank
- SharkSearch
  - Priority queue sorted by combination of similarity, anchor text, similarity of parent, etc. (powerful cousin of FishSearch)
- InfoSpiders
  - Adaptive distributed algorithm using an evolving population of learning agents

Preferential crawlers: Examples

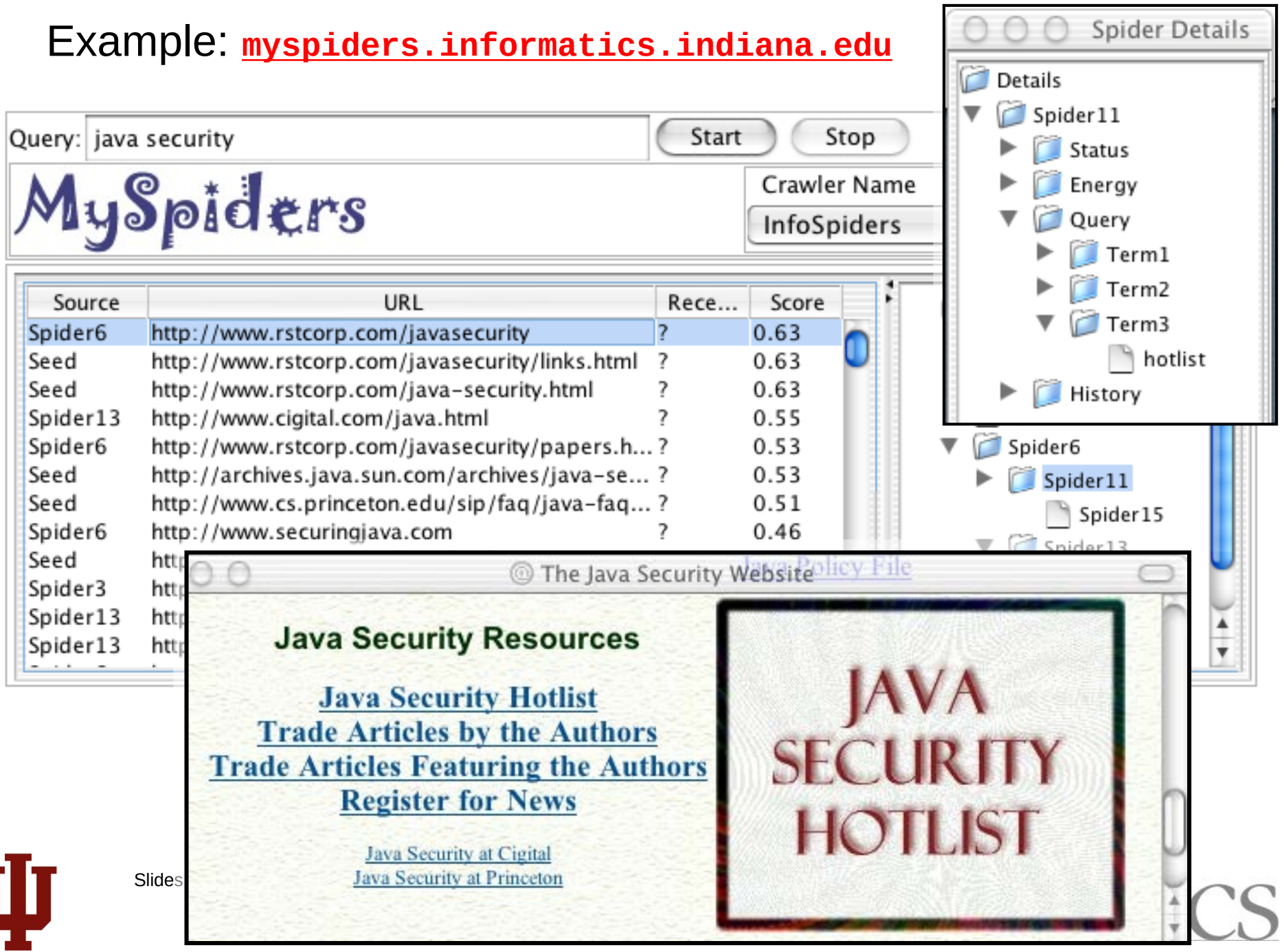
- For I(p) = PageRank (estimated based on pages crawled so far), we can find high-PR pages faster than a breadth-first crawler (Cho, Garcia-Molina & Page 1998)



Topical crawlers

- All we have is a topic (query, description, keywords) and a set of seed pages (not necessarily relevant)
- No labeled examples
- Must predict relevance of unvisited links to prioritize
- Original idea: Menczer 1997, Menczer & Belew 1998

Example: myspiders.informatics.indiana.edu

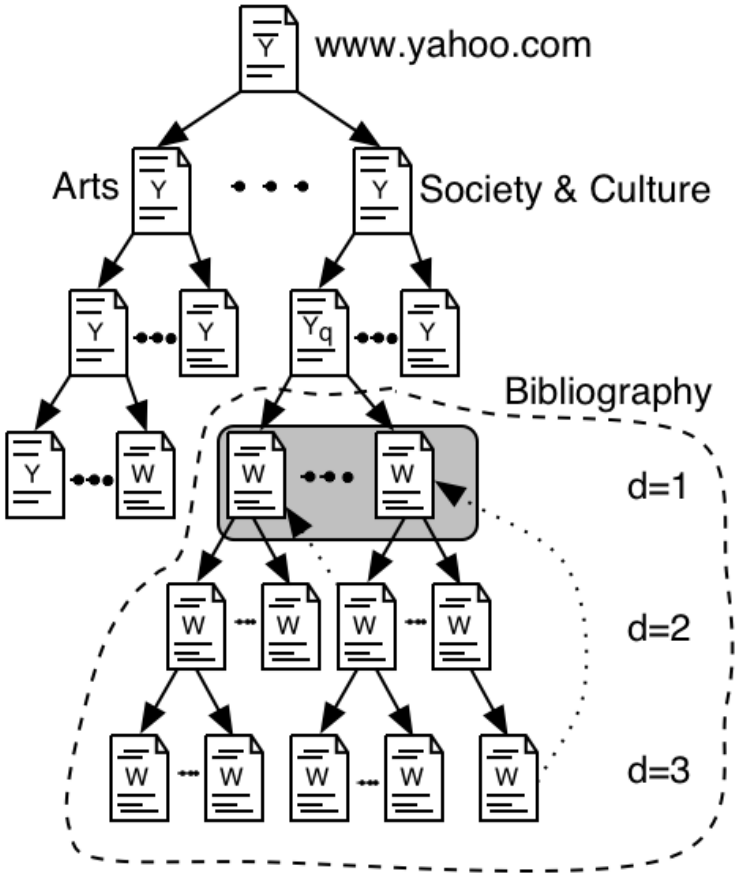


Topical locality

- Topical locality is a necessary condition for a topical crawler to work, and for surfing to be a worthwhile activity for humans
- Links must encode semantic information, i.e. say something about neighbor pages, not be random
- It is also a sufficient condition if we start from "good" seed pages
- Indeed we know that Web topical locality is strong :
  - Indirectly (crawlers work and people surf the Web)
  - From direct measurements (Davison 2000; Menczer 2004, 2005)

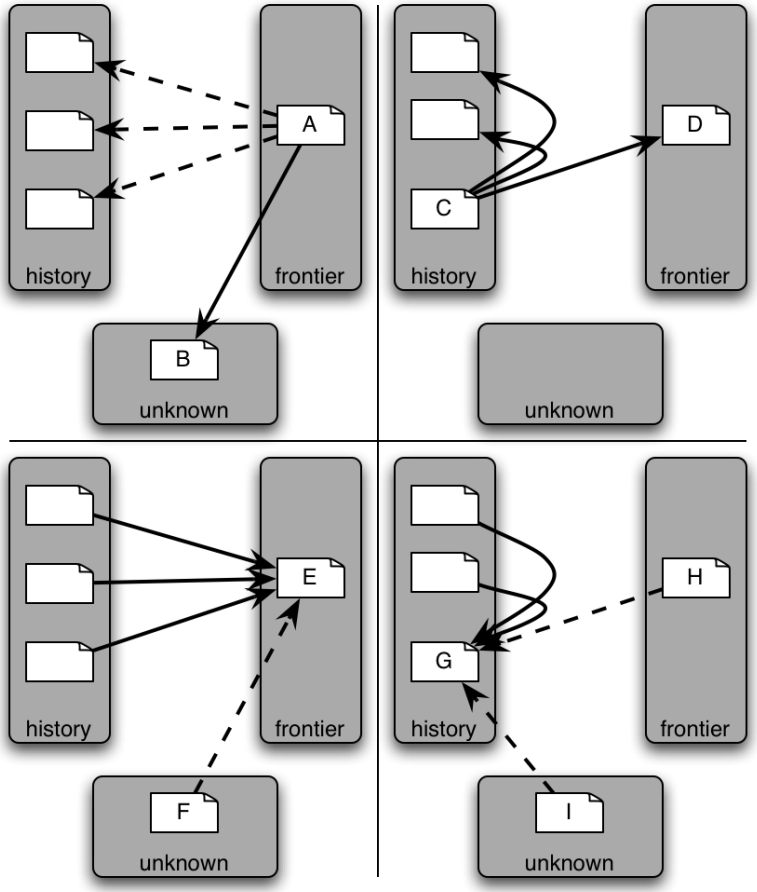
Quantifying topical locality

- Different ways to pose the question:
  - How quickly does semantic locality decay?
  - How fast is topic drift?
  - How quickly does content change as we surf away from a starting page?



Bing Liu: *Web Data Mining*. Springer, 2007
Ch. 8 *Web Crawling* by Filippo Menczer

Topical locality-inspired tricks for topical crawlers



- Co-citation (a.k.a. sibling locality): A and C are good hubs, thus A and D should be given high priority
- Co-reference (a.k.a. blbliographic coupling): E and G are good authorities, thus E and H should be given high priority

Bing Liu: *Web Data Mining*. Springer, 2007
Ch. 8 *Web Crawling* by Filippo Menczer

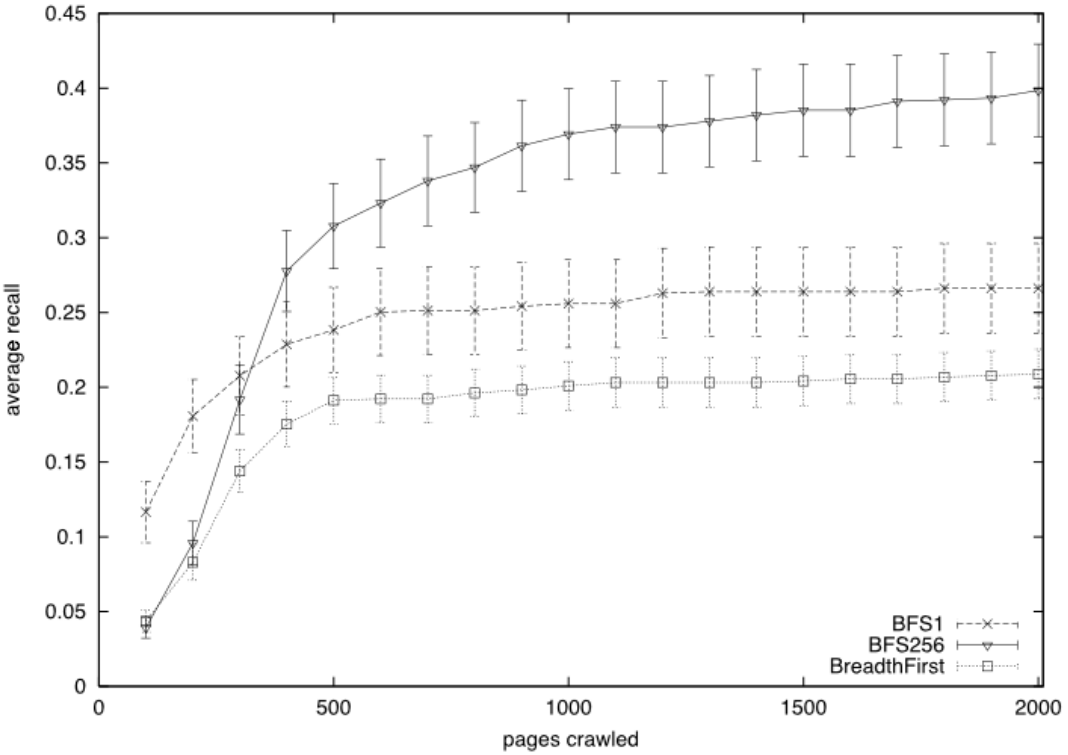
Naïve Best-First

Simplest topical crawler: Frontier is priority queue based on text similarity between topic and parent page

```
BestFirst(topic, seed_urls) {
   foreach link (seed_urls) {
      enqueue(frontier, link);
   }
   while (#frontier > 0 and visited < MAX_PAGES) {
      link := dequeue_link_with_max_score(frontier);
      doc := fetch_new_document(link);
      score :=  sim(topic, doc);
      foreach outlink (extract_links(doc)) {
         if (#frontier >= MAX_BUFFER) {
            dequeue_link_with_min_score(frontier);
         }
         enqueue(frontier, outlink, score);
      }
   }
}
```

Exploration vs Exploitation

- Best-*N*-First (or BFS*N*)
- Rather than re-sorting the frontier every time you add links, be lazy and sort only every *N* pages visited
- Empirically, being less greedy helps crawler performance significantly: escape "local topical traps" by exploring more



Pant et al. 2002

Bing Liu: *Web Data Mining*. Springer, 2007
Ch. 8 *Web Crawling* by Filippo Menczer

Outline

- Motivation and taxonomy of crawlers
- Basic crawlers and implementation issues
- Universal crawlers
- Preferential (focused and topical) crawlers

Need crawling code?

- Reference C implementation of HTTP, HTML parsing, etc
  - w3c-libwww package from World-Wide Web Consortium: www.w3c.org/Library/
- LWP (Perl)
  - http://www.oreilly.com/catalog/perllwp/
  - http://search.cpan.org/~gaas/libwww-perl-5.804/
- Open source crawlers/search engines
  - Nutch: http://www.nutch.org/ (Jakarta Lucene: jakarta.apache.org/lucene/)
  - Heretrix: http://crawler.archive.org/
  - WIRE: http://www.cwr.cl/projects/WIRE/
  - Terrier: http://ir.dcs.gla.ac.uk/terrier/
- Open source topical crawlers, Best-First-N (Java)
  - http://informatics.indiana.edu/fil/IS/JavaCrawlers/
- Evaluation framework for topical crawlers (Perl)
  - http://informatics.indiana.edu/fil/IS/Framework/