Web Mining
(網路探勘)

Web Crawling
(網路爬行)

1011WM08
TLMXM1A
Wed 8,9 (15:10-17:00) U705

Min-Yuh Day
戴敏育
Assistant Professor
專任助理教授
Dept. of Information Management, Tamkang University
淡江大學 資訊管理學系

http://mail. tku.edu.tw/myday/
2012-11-21

課程大綱 (Syllabus)

週次	日期	內容（Subject/Topics）
1	101/09/12	Introduction to Web Mining (網路探勘導論)
2	101/09/19	Association Rules and Sequential Patterns (關聯規則和序列模式)
3	101/09/26	Supervised Learning (監督式學習)
4	101/10/03	Unsupervised Learning (非監督式學習)
5	101/10/10	國慶紀念日(放假一天)
6	101/10/17	Paper Reading and Discussion (論文研讀與討論)
7	101/10/24	Partially Supervised Learning (部分監督式學習)
8	101/10/31	Information Retrieval and Web Search (資訊檢索與網路搜尋)
9	101/11/07	Social Network Analysis (社會網路分析)

課程大綱 (Syllabus)

週次	日期	內容（Subject/Topics）
10	101/11/14	Midterm Presentation (期中報告)
11	101/11/21	Web Crawling (網路爬行)
12	101/11/28	Structured Data Extraction (結構化資料擷取)
13	101/12/05	Information Integration (資訊整合)
14	101/12/12	Opinion Mining and Sentiment Analysis (意見探勘與情感分析)
15	101/12/19	Paper Reading and Discussion (論文研讀與討論)
16	101/12/26	Web Usage Mining (網路使用挖掘)
17	102/01/02	Project Presentation 1 (期末報告1)
18	102/01/09	Project Presentation 2 (期末報告2)

Outline

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

Web Crawlers

- Programs that automatically download Web pages
  - Web Crawlers
  - Web Spiders
  - Robots
  - Web Agent
  - Wanderer
  - Worm
  - Bot
  - Harvester

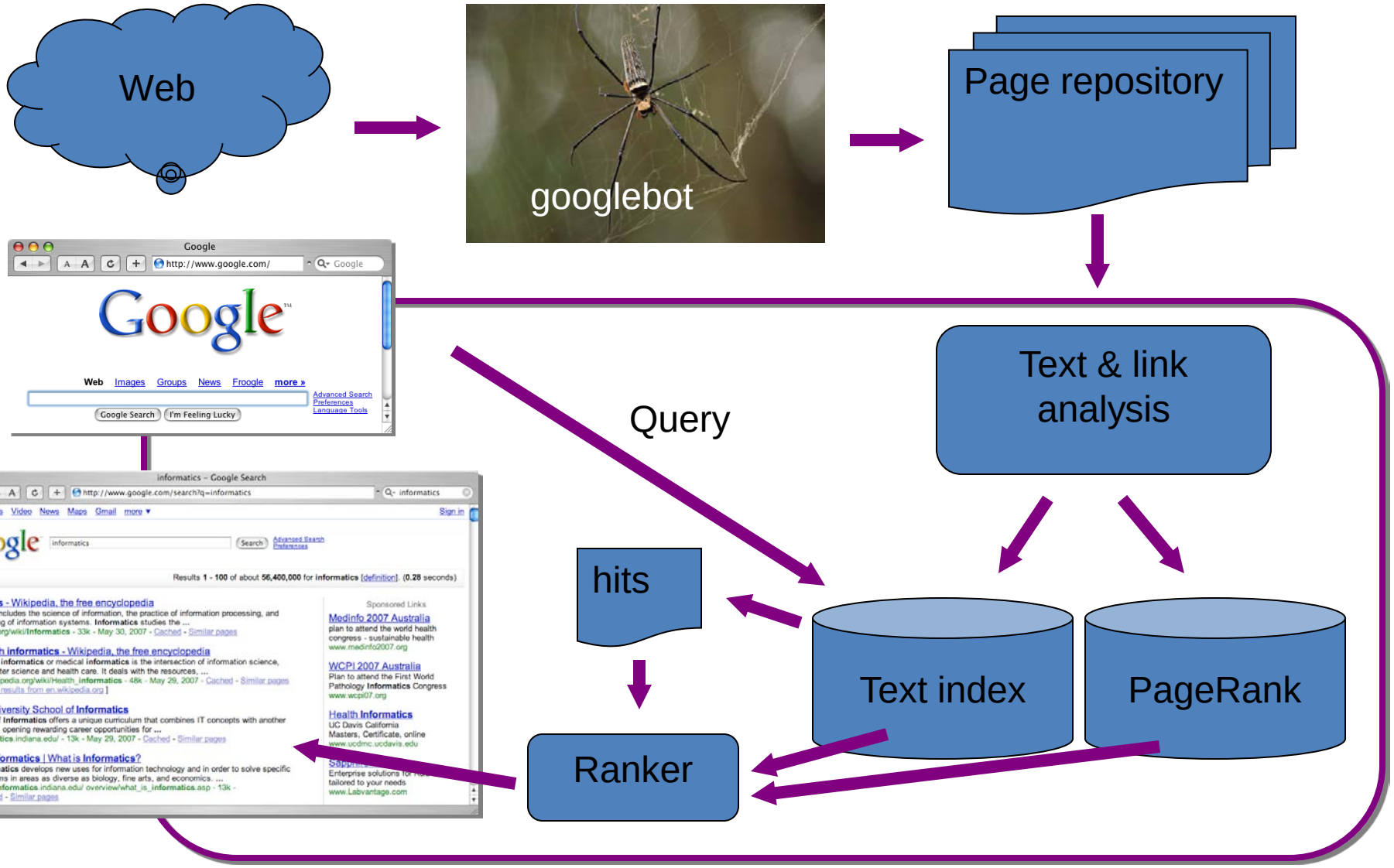
Source: Bing Liu (2011) , Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data

Motivation for crawlers

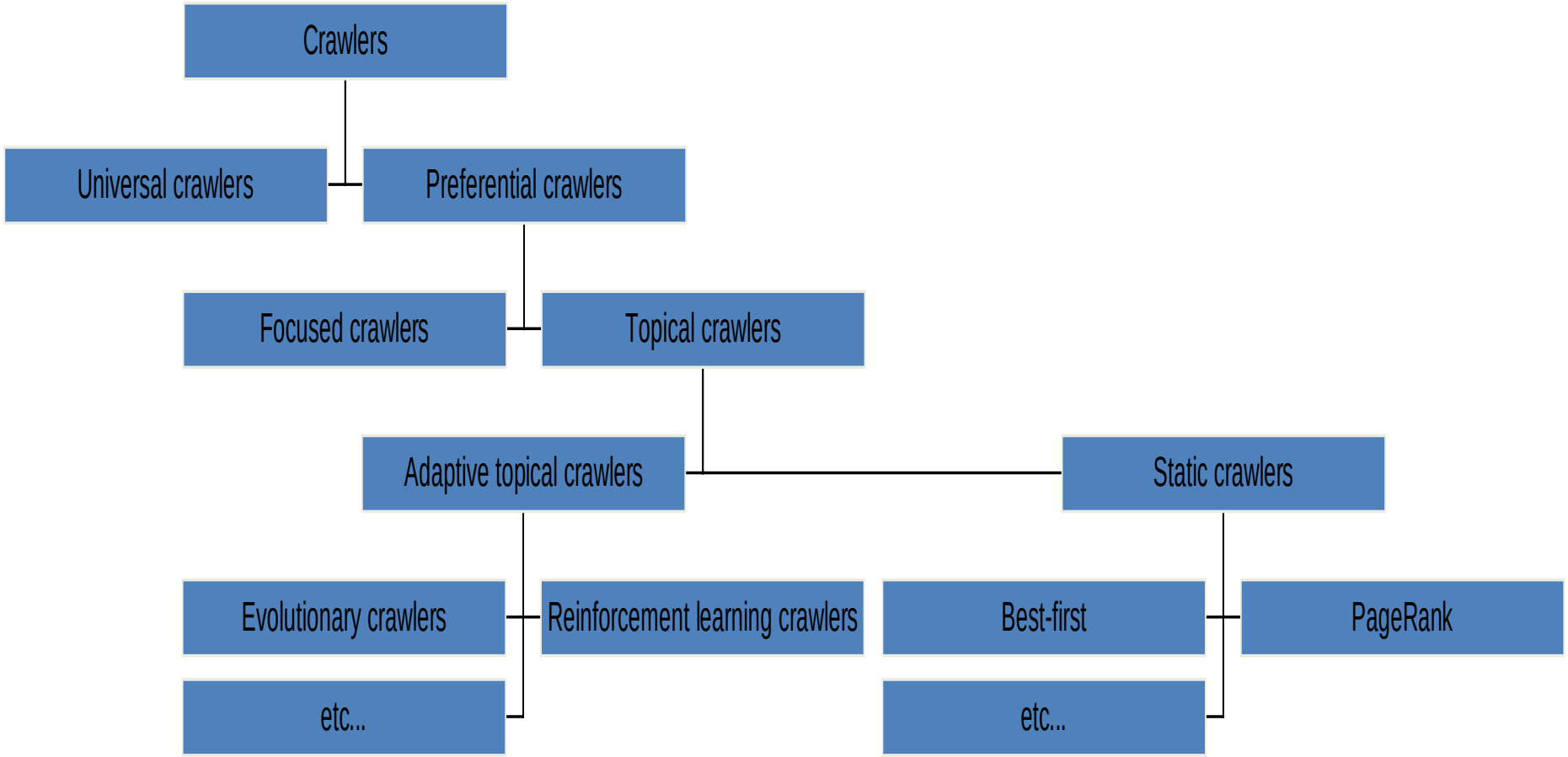
- Support universal search engines
  - Google, Yahoo, MSN/Windows Live, Ask, etc.
- Vertical (specialized) search engines
  - e.g. news, shopping, papers, recipes, reviews, etc.
- Business intelligence
  - keep track of potential competitors, partners
- Monitor Web sites of interest
- Evil
  - harvest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

Source: Bing Liu (2011) , Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data

A crawler within a search engine

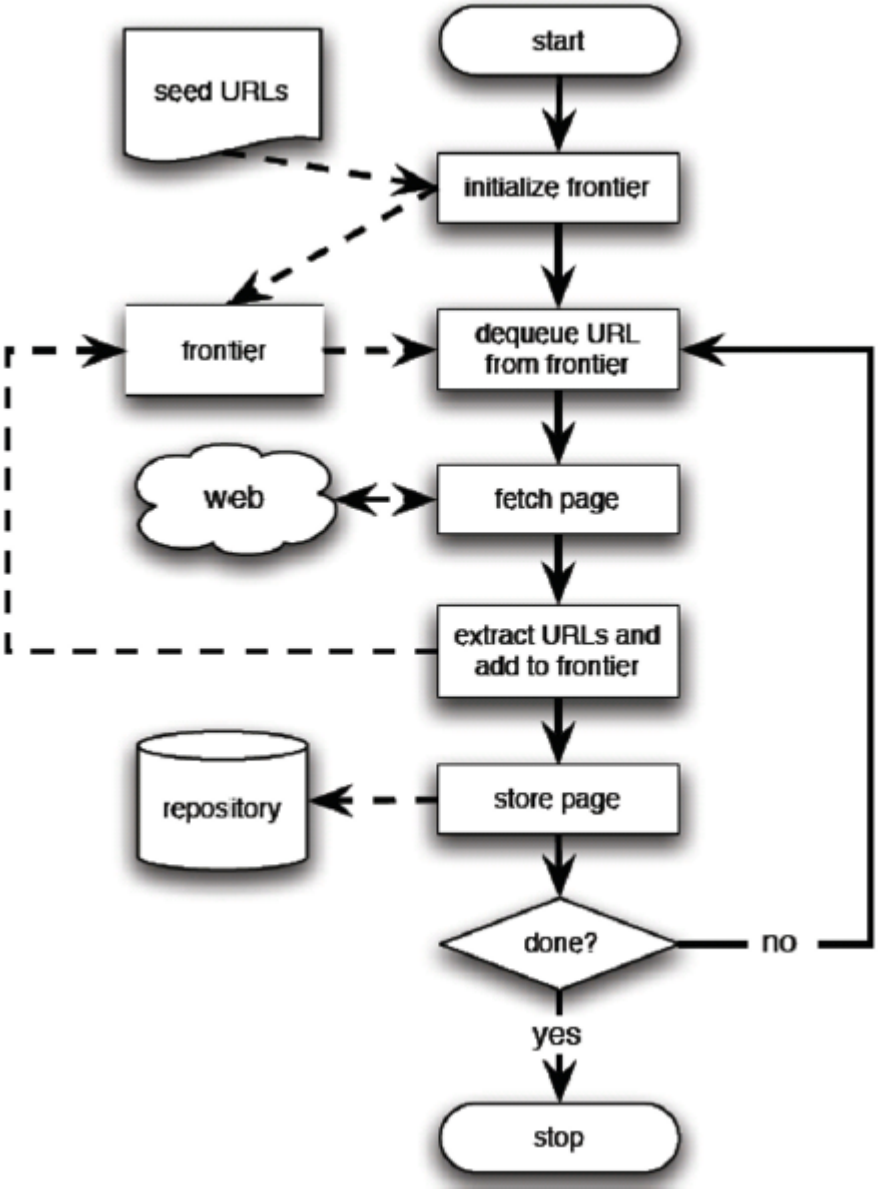


One taxonomy of crawlers



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

Basic Web Crawler



Source: Bing Liu (2011) , Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data

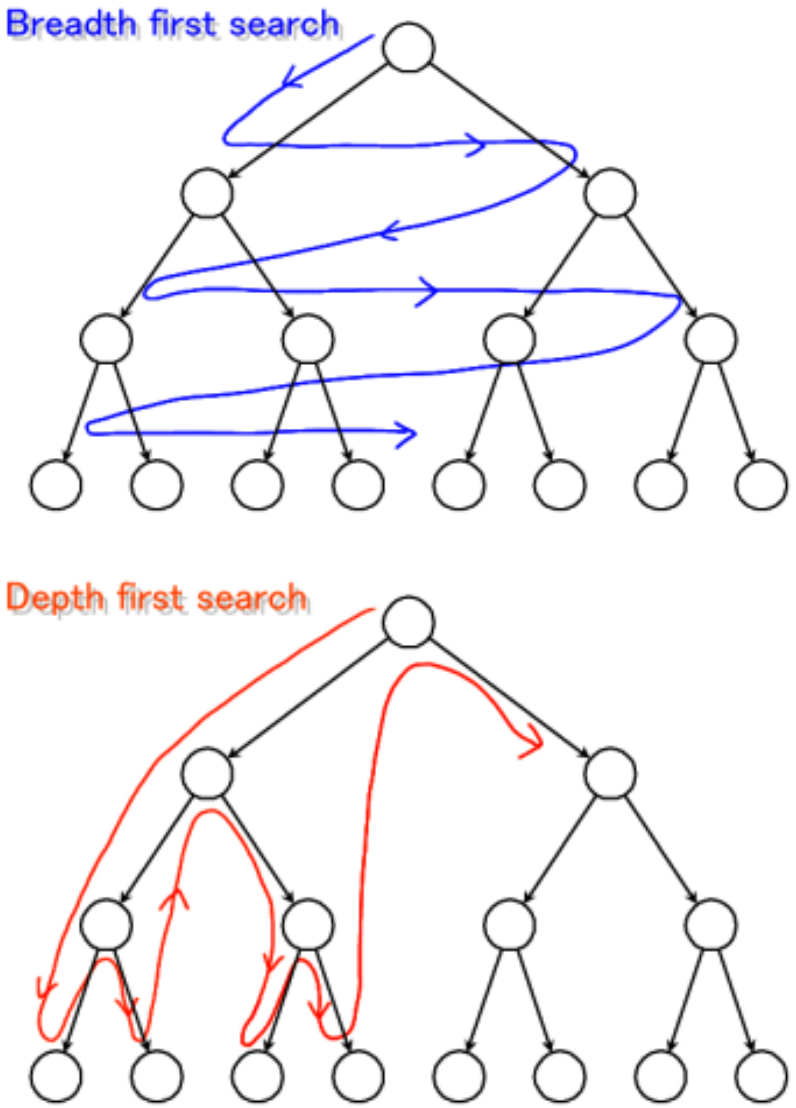
Major Steps of a Web Crawler

- **Seed URLs**
- **Frontier**
  - Crawler maintains a list of unvisited
  - URL Frontier
    - The next node to crawl
- **Fetch Page**
- **Parse and Extract URLs from Page**
  - **add URLs to frontier**
- **Store page**

Source: Bing Liu (2011) , Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data

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);
}
```

Source: Bing Liu (2011) , Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data

Open Source Crawlers

- 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/

Web Crawler Implementation issues

- Fetching
- Parsing
- Stopword Removal and Stemming
- Link Extraction and Canonicalization
- Spider Traps
- Page Repository
- Concurrency

Source: Bing Liu (2011) , Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data

Implementation issues

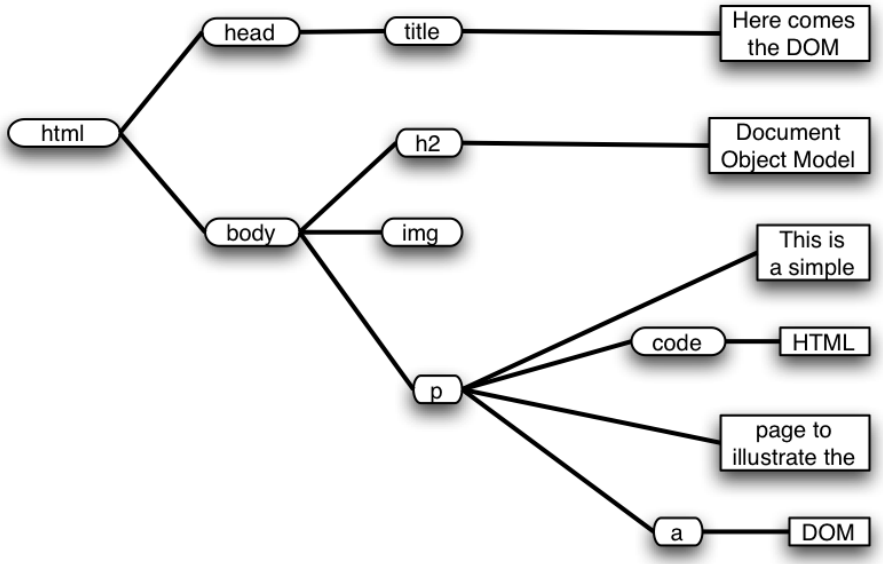
- Don’t want to fetch same page twice!
  - Keep lookup table (hash) of visited pages
  - What if not visited but in frontier already?
- The frontier grows very fast!
  - May need to prioritize for large crawls
- Fetcher must be robust!
  - Don’t crash if download fails
  - Timeout mechanism
- Determine 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

Implementation issues

- Fetching
  - Get only the first 10-100 KB per page
  - Take care to detect and break redirection loops
  - Soft fail for timeout, server not responding, file not found, and other errors

Implementation issues: Parsing

```
<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>
```



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, SVG, RSS, AJAX…

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

Implementation issues

Conflation and thesauri

- 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

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

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 on Canonical URLs

Other transformations require heuristic assumption about the intentions of the author or configuration of the Web server:

1. Removing default file name
   - ✓ **http://informatics.indiana.edu/fil/index.html**
   - × **http://informatics.indiana.edu/fil/**
   - – This is reasonable in general but would be wrong in this case because the default happens to be 'default.asp' instead of 'index.html'
2. Trailing directory
   - × **http://informatics.indiana.edu/fil**
   - ✓ **http://informatics.indiana.edu/fil/**
   - – This is correct in this case but how can we be sure in general that there isn't a file named 'fil' in the root dir?

Convert URLs to canonical forms

Description and transformation	Example and canonical form
Default port number Remove	http://cs.indiana.edu:80/ http://cs.indiana.edu/
Root directory Add trailing slash	http://cs.indiana.edu http://cs.indiana.edu/
Guessed directory* Add trailing slash	http://cs.indiana.edu/People http://cs.indiana.edu/People/
Fragment Remove	http://cs.indiana.edu/faq.html#3 http://cs.indiana.edu/faq.html
Current or parent directory Resolve path	http://cs.indiana.edu/a/./../b/ http://cs.indiana.edu/b/
Default filename* Remove	http://cs.indiana.edu/index.html http://cs.indiana.edu/
Needlessly encoded characters Decode	http://cs.indiana.edu/%7Efil/ http://cs.indiana.edu/~fil/
Disallowed characters Encode	http://cs.indiana.edu/My File.htm http://cs.indiana.edu/My%20File.htm
Mixed/upper-case host names Lower-case	http://CS.INDIANA.EDU/People/ http://cs.indiana.edu/People/

Source: Bing Liu (2011) , Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data

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

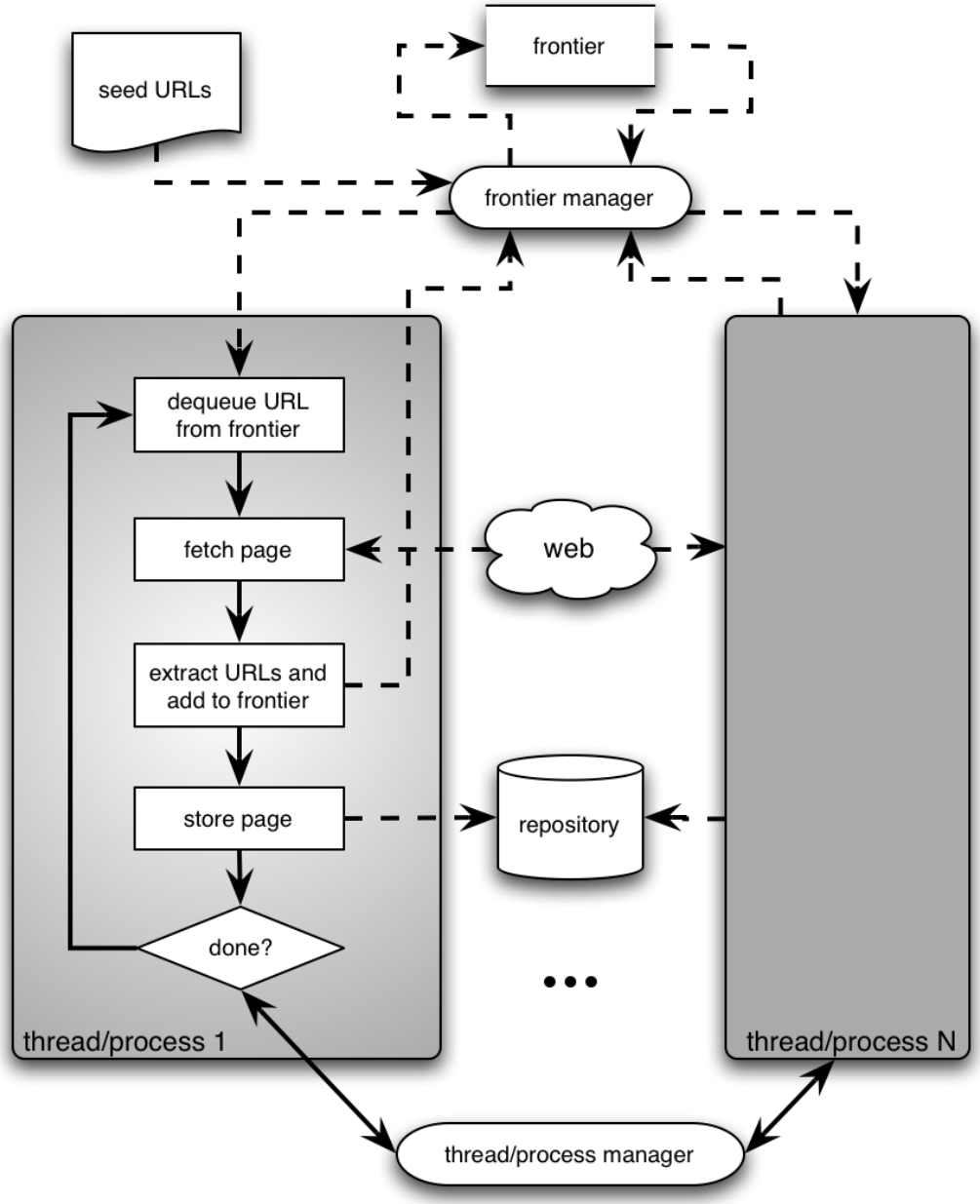
More implementation issues

- 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



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

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

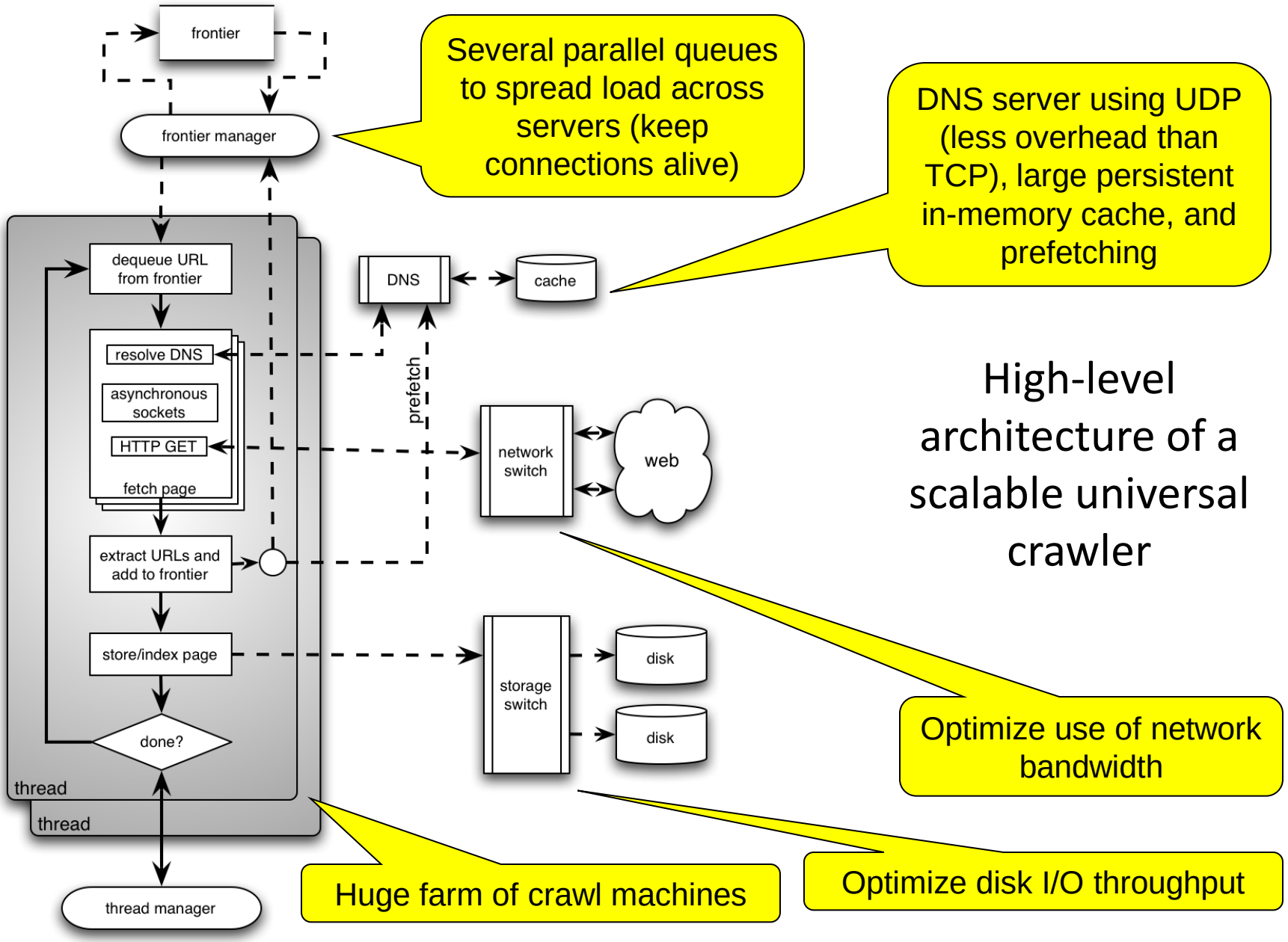
Large-scale universal crawlers

- 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


frontier
frontier manager
Several parallel queues to spread load across servers (keep connections alive)
DNS server using UDP (less overhead than TCP), large persistent in-memory cache, and prefetching
dequeue URL from frontier
DNS
cache
resolve DNS
asynchronous sockets
HTTP GET
fetch page
prefetch
network switch
web
High-level architecture of a scalable universal crawler
extract URLs and add to frontier
store/index page
storage switch
disk
disk
done?
thread
thread
Optimize use of network bandwidth
Huge farm of crawl machines
Optimize disk I/O throughput
thread manager

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

Source: Bing Liu (2011) , Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data

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

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)
- Possible figures of merit:
  - Precision ~
    | p: crawled(p) & I(p) > threshold | / | p: crawled(p) |
  - Recall ~
    | p: crawled(p) & I(p) > threshold | / | p: I(p) > threshold |

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
  - Variants: DOM context, hub scores
- 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

Focused crawlers

- Can have multiple topics with as many classifiers, with scores appropriately combined
  - (Chakrabarti et al. 1999)
- Can use a distiller to find topical hubs periodically, and add these to the frontier
- Can accelerate with the use of a critic
  - (Chakrabarti et al. 2002)
- Can use alternative classifier algorithms to naïve-Bayes, e.g. SVM and neural nets have reportedly performed better
  - (Pant & Srinivasan 2005)

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

Crawler ethics and conflicts

- Crawlers can cause trouble, even unwillingly, if not properly designed to be “polite” and “ethical”
- For example, sending too many requests in rapid succession to a single server can amount to a Denial of Service (DoS) attack!
  - Server administrator and users will be upset
  - Crawler developer/admin IP address may be blacklisted

Crawler etiquette (important!)

- Identify yourself
  - Use 'User-Agent' HTTP header to identify crawler, website with description of crawler and contact information for crawler developer
  - Use 'From' HTTP header to specify crawler developer email
  - Do not disguise crawler as a browser by using their 'User-Agent' string
- Always check that HTTP requests are successful, and in case of error, use HTTP error code to determine and immediately address problem
- Pay attention to anything that may lead to too many requests to any one server, even unwillingly, e.g.:
  - redirection loops
  - spider traps

Crawler etiquette (important!)

- Spread the load, do not overwhelm a server
  - Make sure that no more than some max. number of requests to any single server per unit time, say < 1/second
- Honor the Robot Exclusion Protocol
  - A server can specify which parts of its document tree any crawler is or is not allowed to crawl by a file named 'robots.txt' placed in the HTTP root directory, e.g. http://www.indiana.edu/robots.txt
  - Crawler should always check, parse, and obey this file before sending any requests to a server
  - More info at:
    - http://www.google.com/robots.txt
    - http://www.robotstxt.org/wc/exclusion.html

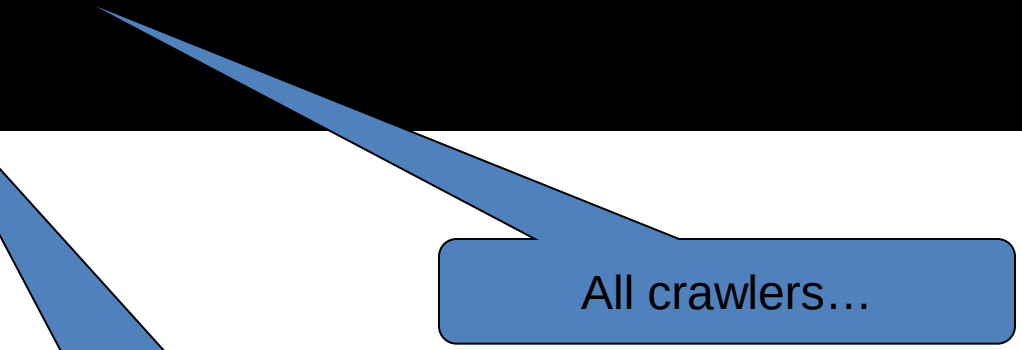
More on robot exclusion

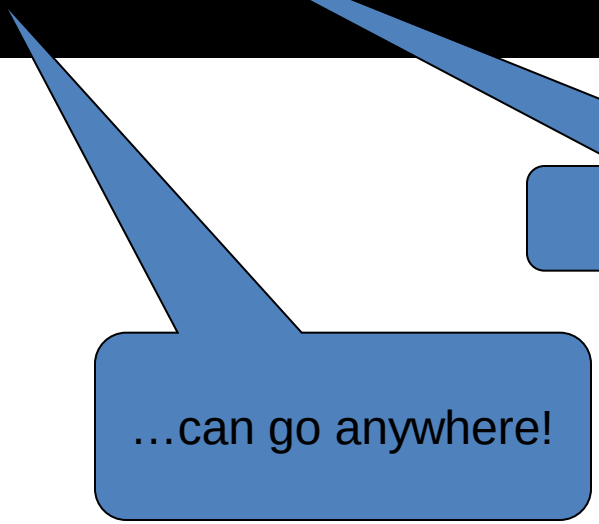
- Make sure URLs are canonical before checking against robots.txt
- Avoid fetching robots.txt for each request to a server by caching its policy as relevant to this crawler

www.apple.com/robots.txt

```
# robots.txt for http://www.apple.com/

User-agent: *
Disallow:
```





www.microsoft.com/robots.txt

```
# Robots.txt file for http://www.microsoft.com

User-agent: *
Disallow: /canada/Library/mnp/2/aspx/
Disallow: /communities/bin.aspx
Disallow: /communities/eventdetails.mspx
Disallow: /communities/blogs/PortalResults.mspx
Disallow: /communities/rss.aspx
Disallow: /downloads/Browse.aspx
Disallow: /downloads/info.aspx
Disallow: /france/formation/centres/planning.asp
Disallow: /france/mnp_utility.mspx
Disallow: /germany/library/images/mnp/
Disallow: /germany/mnp_utility.mspx
Disallow: /ie/ie40/
Disallow: /info/customerror.htm
Disallow: /info/smart404.asp
Disallow: /intlkb/
Disallow: /isapi/
#etc…
```

All crawlers…

…are not allowed in these paths…

www.springer.com/robots.txt

```
# Robots.txt for http://www.springer.com (fragment)

User-agent: Googlebot
Disallow: /chl/*
Disallow: /uk/*
Disallow: /italy/*
Disallow: /france/*

User-agent: slurp
Disallow:
Crawl-delay: 2

User-agent: MSNBot
Disallow:
Crawl-delay: 2

User-agent: scooter
Disallow:

# all others
User-agent: *
Disallow: /
```

Google crawler is allowed everywhere except these paths

Yahoo and MSN/Windows Live are allowed everywhere but should slow down

AltaVista has no limits

Everyone else keep off!

More crawler ethics issues

- Is compliance with robot exclusion a matter of law?
  - No! Compliance is voluntary, but if you do not comply, you may be blocked
  - Someone (unsuccessfully) sued Internet Archive over a robots.txt related issue
- Some crawlers disguise themselves
  - Using false User-Agent
  - Randomizing access frequency to look like a human/browser
  - Example: click fraud for ads

More crawler ethics issues

- Servers can disguise themselves, too
  - Cloaking: present different content based on User-Agent
  - E.g. stuff keywords on version of page shown to search engine crawler
  - Search engines do not look kindly on this type of "spamdexing" and remove from their index sites that perform such abuse
    - Case of bmw.de made the news

Gray areas for crawler ethics

- If you write a crawler that unwillingly follows links to ads, are you just being careless, or are you violating terms of service, or are you violating the law by defrauding advertisers?
  - Is non-compliance with Google's robots.txt in this case equivalent to click fraud?
- If you write a browser extension that performs some useful service, should you comply with robot exclusion?

Summary

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

References

- Bing Liu (2011) , “Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data,” 2nd Edition, Springer. http://www.cs.uic.edu/~liub/WebMiningBook.html