Web-Semantik-Technologien

Vorlesung WS 2019/20

thomas.kern@fh-hagenberg.at

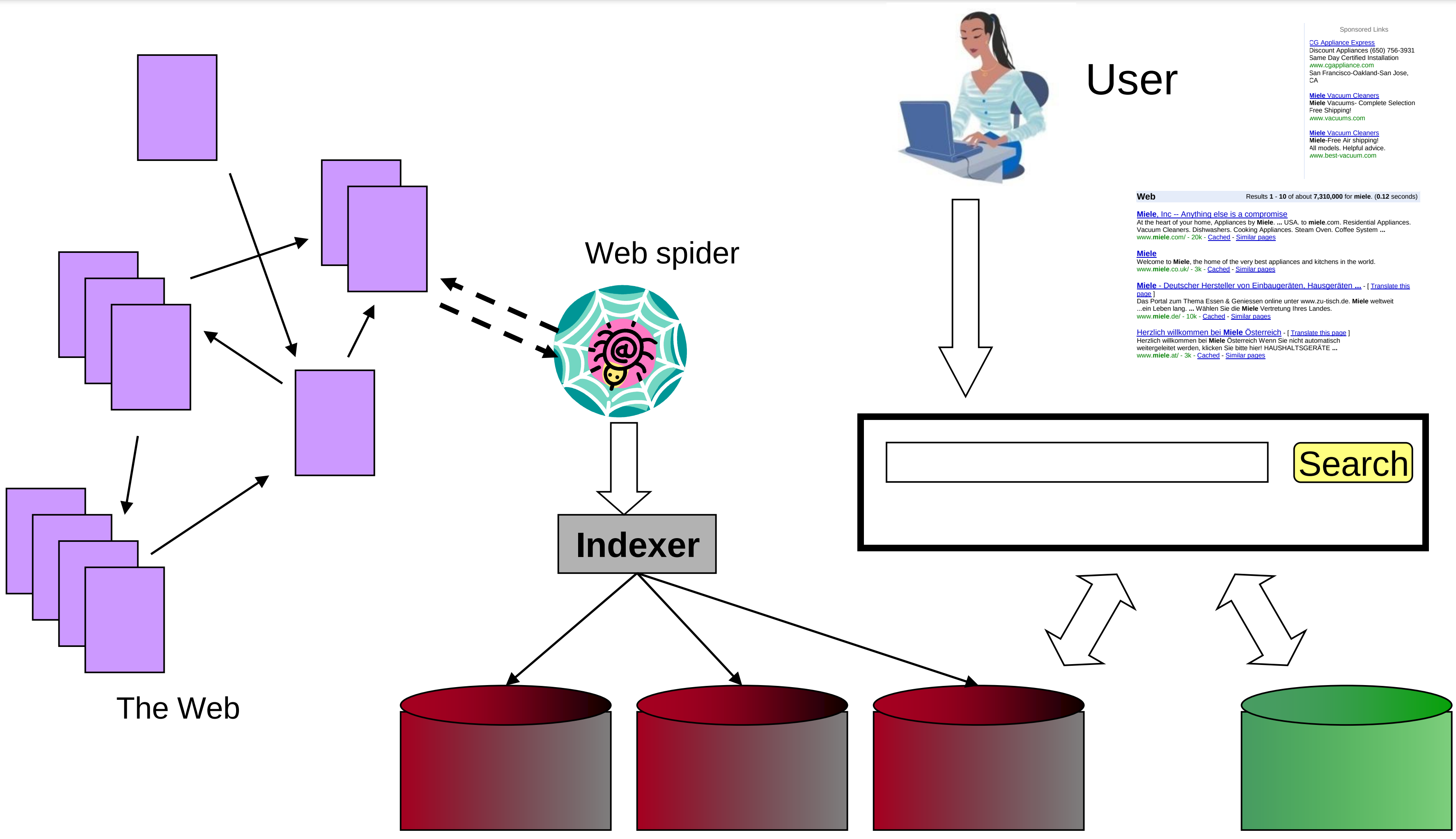
Web Search

Crawling and Web Indexes

Brief history

- **Early keyword-based engines ca. 1995-1997**
  - Altavista, Excite, Infoseek, Inktomi, Lycos
- **Paid search ranking: Goto (morphed into Overture.com → Yahoo!)**
  - Your search ranking depended on how much you paid
  - Auction for keywords
- **1998+: Link-based ranking pioneered by Google**
  - Great user experience in search of a business model
- **Result: Google added paid search "ads" to the side, independent of search results**
  - Yahoo followed suit, acquiring Overture (for paid placement)
- **2005+: Google gains search share, dominating in Europe and very strong in North America**
  - 2009: Yahoo! and Microsoft propose combined paid search offering

Web search



User Needs

- **Informational** – want to learn about something (~40% / 65%)
- **Navigational** – want to go to that page (~25% / 15%)
- **Transactional** – want to do something (web-mediated) (~35% / 20%)
  - Access a service
  - Downloads
  - Shop
- **Gray areas**
  - Find a good hub
  - Exploratory search "see what's there"

Users’ empirical evaluation of results

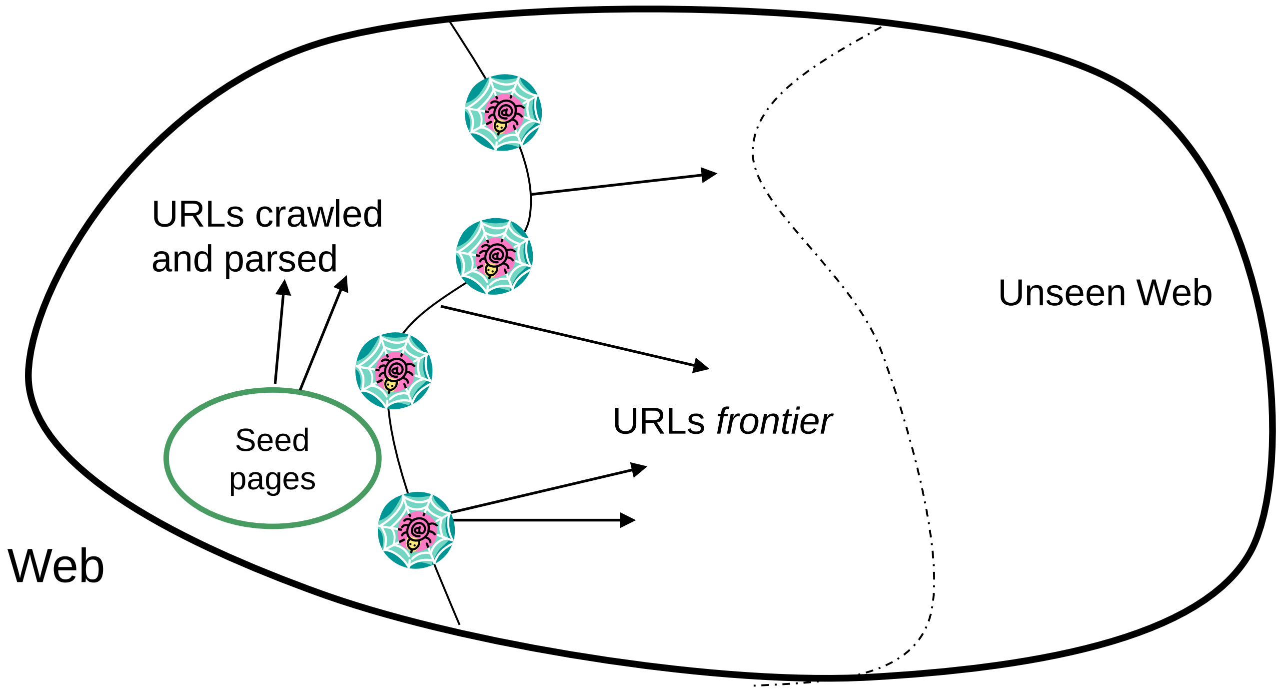
- **Quality of pages varies widely**
  - Relevance is not enough
  - Other desirable qualities
    - Content: Trustworthy, diverse, non-duplicated, well maintained
    - Web readability: display correctly & fast
    - No annoyances: pop-ups, etc

- **Precision vs. Recall**
  - On the web, recall seldom matters

User perceptions may be unscientific, but are significant over a large aggregate

Basic crawler operation

- **Begin with known "seed" URLs in a queue**
- **Fetch and parse them**
  - Extract URLs they point to
  - Place the extracted URLs on a queue
- **Fetch each URL on the queue and repeat**



Basic crawler properties

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

What crawlers *must* do…

- **Be Polite:** Only crawl allowed pages
  - Respect implicit and explicit politeness considerations
    - **Explicit politeness:** specifications from webmasters on what portions of site can be crawled (*robots.txt*)
    - **Implicit politeness:** even with no specification, avoid hitting any site too often

- **Be Robust:**
  - Be immune to spider traps and other malicious behavior from web servers

What crawlers *should* do…

- **Be capable of distributed operation**: designed to run on multiple distributed machines
- **Be scalable:** designed to increase the crawl rate by adding more machines
- **Performance/efficiency:** permit full use of available processing and network resources
- Fetch pages of "**higher quality**" first
- **Continuous operation:** Continue fetching fresh copies of a previously fetched page
- **Extensible:** Adapt to new data formats, protocols

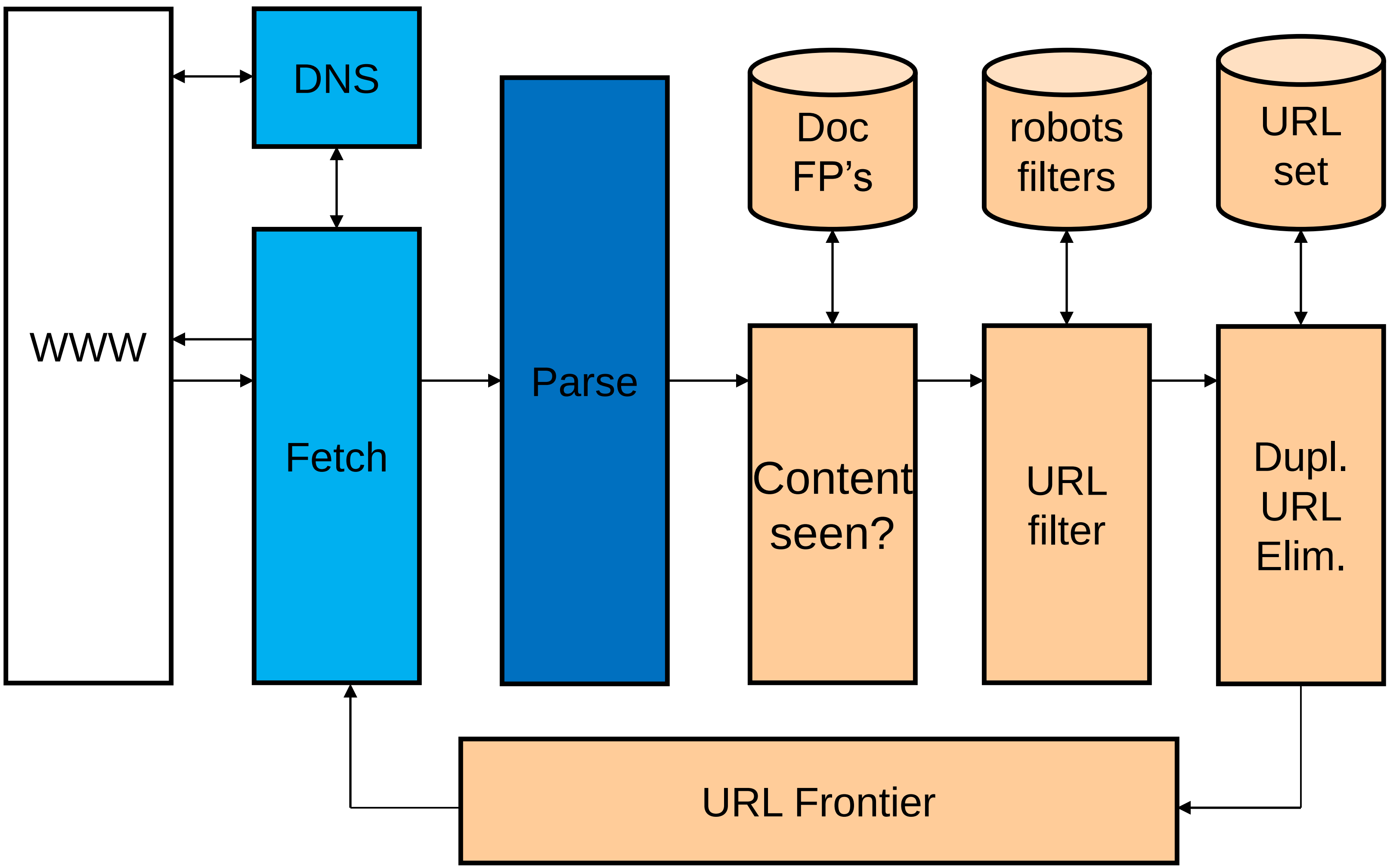
Processing steps in crawling

1. **Pick a URL from the frontier**
2. **Fetch the document at the URL**
3. **Parse the URL**
   - Extract links from it to other docs (URLs)
4. **Check if URL has content already seen**
   - If not, add to indexes
5. **For each extracted URL**
   - Ensure it passes certain URL filter tests
   - Check if it is already in the frontier (duplicate URL elimination)

Which one?

E.g., only crawl .edu, obey robots.txt, etc.

Basic crawler architecture



Basic crawler architecture

1. The ***URL frontier***, containing URLs yet to be fetched in the current crawl (in the case of continuous crawling, a URL may have been fetched previously but is back in the frontier for re-fetching).
2. A ***DNS resolution module*** that determines the web server from which to fetch the page specified by a URL.
3. A ***fetch module*** that uses the http protocol to retrieve the web page at a URL.
4. A ***parsing module*** that extracts the text and set of links from a fetched web page.
5. The ***Document Fingerprints*** determine if content has already been seen at another URL.
6. A ***URL filter*** designates URL’s to be (not) crawled.
7. A ***duplicate elimination module*** determines whether an extracted link is already in the URL frontier or has recently been fetched.

URL frontier

- Can include multiple pages from the same host
- Must avoid trying to fetch them all at the same time
- Must try to keep all crawling threads busy

DNS (Domain Name Server)

- **A lookup service on the internet**
  - Given a URL, retrieve its IP address
  - Service provided by a distributed set of servers – thus, lookup latencies can be high (even seconds)
- **Solutions**
  - DNS caching
  - Batch DNS resolver – collects requests and sends them out together

Parsing: URL normalization

- When a fetched document is parsed *(HTML, XML, PDF Parser, Plain Text)*, some of the extracted links are ***relative*** **URLs**

  (e.g., at http://en.wikipedia.org/wiki/Main_Page) we have a **relative link to /wiki/Wikipedia:General_ disclaimer** which is the same as the absolute URL http://en.wikipedia.org/wiki/Wikipedia:General_disclaimer)

- **Solution:** During parsing, must ***normalize*** **(expand) the relative URLs**

Content seen?

- Duplication is widespread on the web
- If the page just fetched is already in the index, do not further process it
- This is verified using document ***fingerprints*** or ***shingles***

Duplicate URL Elimination

- For a non-continuous (one-shot) crawl, test to see if an extracted+filtered URL has already been passed to the frontier

- **Duplication:** Exact match can be detected with ***fingerprints***
  - The simplest approach to detecting duplicates is to compute, for each web page, a fingerprint that is a succinct (say 64-bit) digest of the characters on that page.

- **Near-Duplication:** Approximate match
  - Compute syntactic similarity with an edit-distance measure
  - Use **similarity threshold** to detect near-duplicates
    - E.g., Similarity > 80% => Documents are “near duplicates”
    - Not transitive though sometimes used transitively

Computing Similarity

- **Features:**
  - Segments of a document (natural or artificial breakpoints)
  - Shingles (Word N-Grams)

a rose is a rose is a rose →

a_rose_is_a

rose_is_a_rose

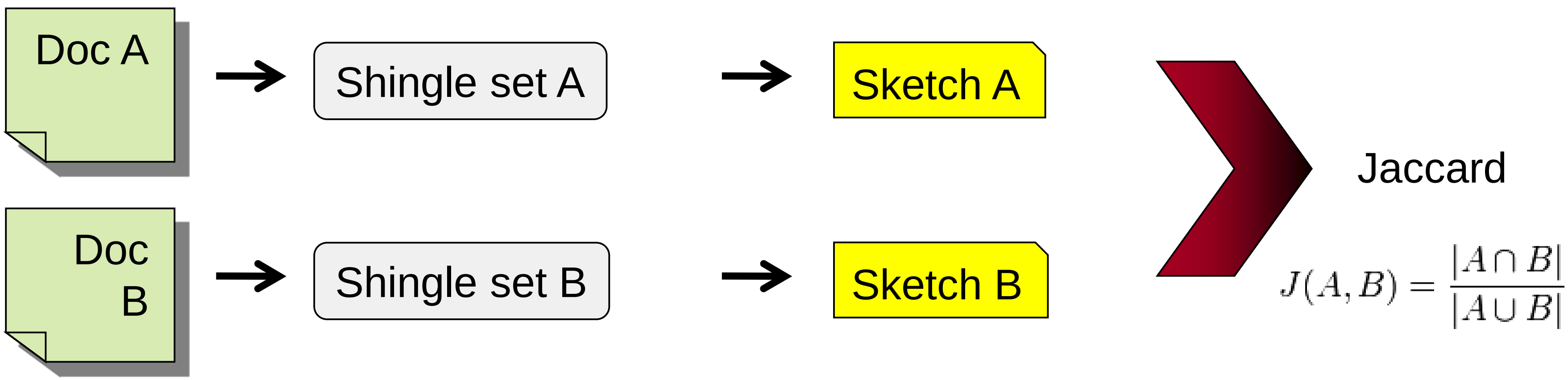
is_a_rose_is

a_rose_is_a

- Similarity measure between two docs **(= sets of shingles)**

Shingles + Set Intersection

- Computing **exact** set intersection of shingles between **all** pairs of documents is expensive/intractable
  - Approximate using a cleverly chosen subset of shingles from each (**a *sketch***)
- Estimate ***(size_of_intersection / size_of_union)*** based on a short sketch



*Documents that share ≥ τ (say 80%) corresponding vector elements are **near duplicates***

Filters and robots.txt

- **Filters:** regular expressions for URL's to be (not) crawled
- Once a *robots.txt* file is fetched from a site, need not fetch it repeatedly
  - Doing so burns bandwidth, hits web server → Cache *robots.txt* files
- Protocol for giving spiders ("**robots**") limited access to a website, originally from 1994
  http://www.robotstxt.org/robotstxt.html
- Website announces its **request on what can(not) be crawled**
  - For a URL, create a file `URL/robots.txt` with Keywords `User-agent:`, `Disallow:` to specify access restrictions.
  - Special Bots additionally use: `Allow:`, `Crawl-delay:`, `Sitemap:`

Robots.txt example: No crawler should visit any URL starting with "/yoursite/temp/", except the crawler called "searchengine":

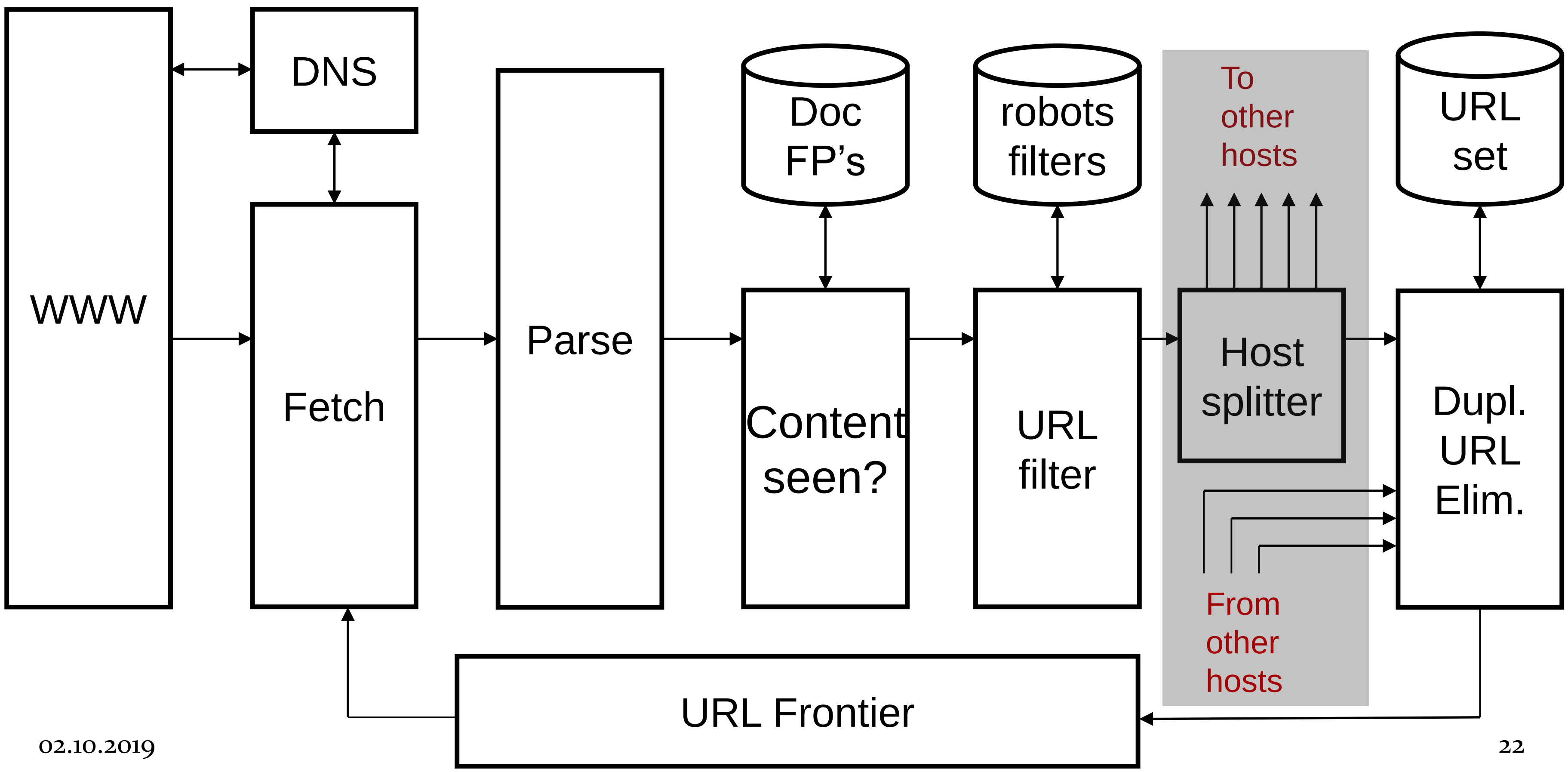
```
User-agent: searchengine
Disallow:
User-agent: *
Disallow: /yoursite/temp/
```

Distributing the crawler

- Run multiple crawl threads, under different processes – potentially at different nodes
  - Geographically distributed nodes
- Partition hosts being crawled into nodes
  - Hash used for partition
- How do these nodes communicate?

Communication between nodes

The output of the URL filter at each node is sent to the Duplicate URL Eliminator at all nodes



URL frontier: two main considerations

1. ***Politeness:*** do not hit a web server too frequently
2. ***Freshness:*** crawl some pages more often than others; e.g., pages (such as News sites) whose content changes often.

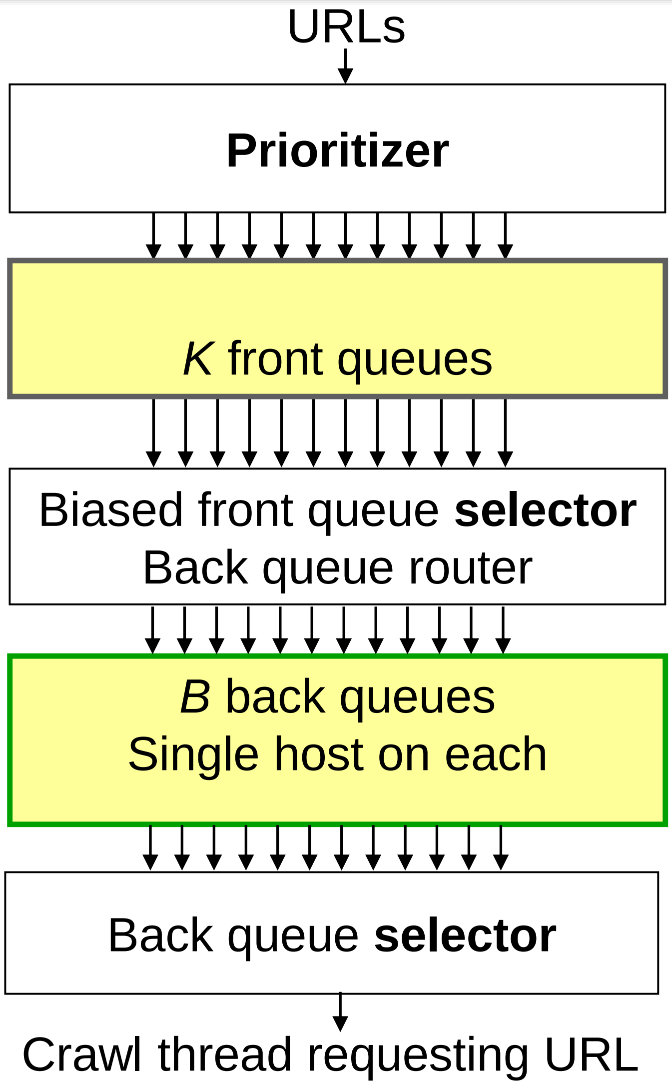
- These goals may conflict each other; e.g., simple priority queue fails – many links out of a page go to its own site, creating a burst of accesses to that site.

- **Politeness – challenges**
  - Even if we restrict only one thread to fetch from a host, can hit it repeatedly
  - **Common heuristic**: insert **time gap** between successive requests to a host that is time for most recent fetch from that host

URL frontier: Mercator scheme

Mercator URL frontier

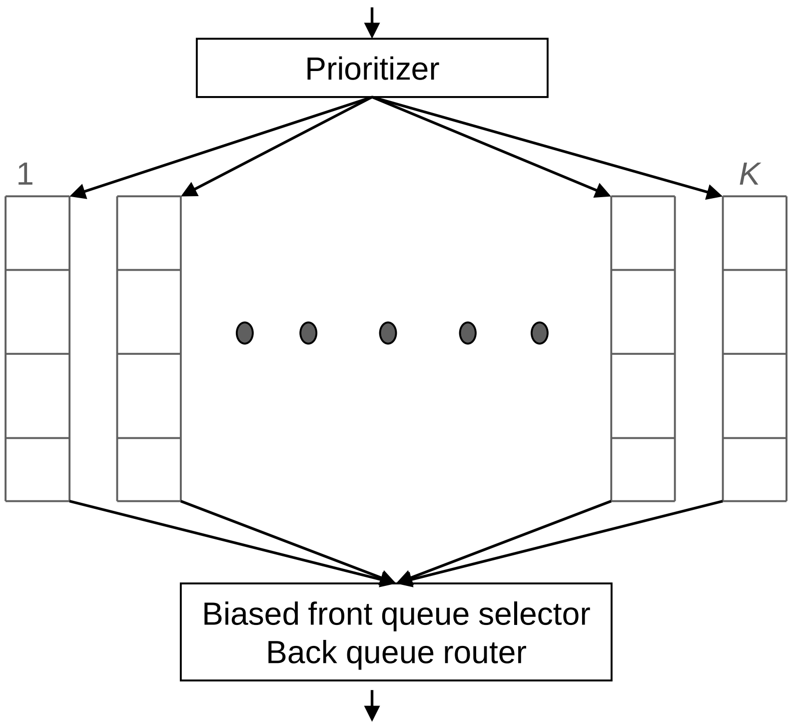
- URLs extracted from crawled pages flow in from the top into the frontier
- Front queues manage prioritization
- Back queues enforce politeness
- Each queue is FIFO
- A crawl thread requesting a URL extracts it from the bottom of the figure.
- Further information: http://www.csd.uwo.ca/faculty/solis/cs868b/2013/papers/crawler.pdf



Front queues

Front queues

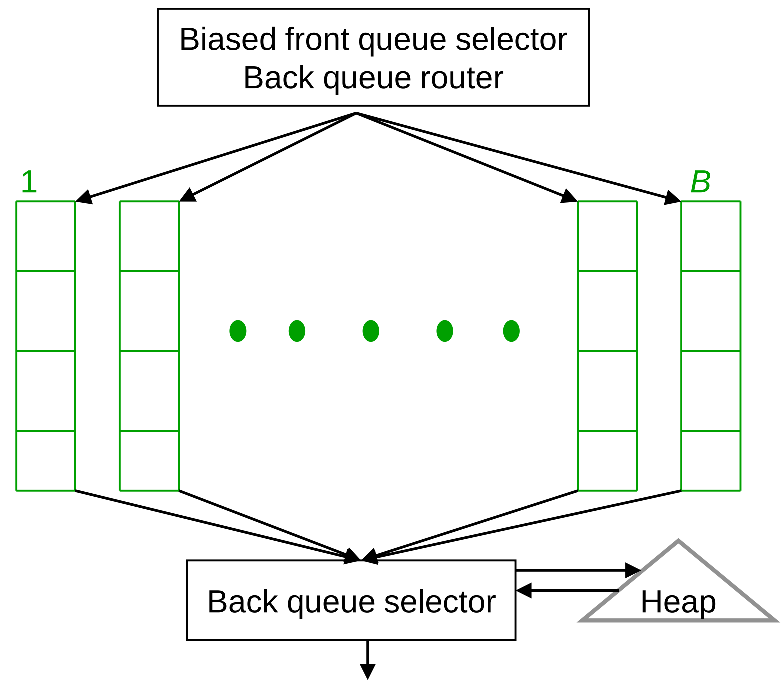
- Prioritizer assigns an integer priority to URL between 1 and K
  - Appends URL to corresponding queue
- Heuristics for assigning priority
  - Refresh rate sampled from previous crawls
  - Application-specific (e.g., "crawl news sites more often")



Back queues

Biased front queue selector

- When a *back queue* requests a URL :
  - picks a front queue from which to pull a URL
- This choice can be round robin biased to queues of higher priority, or some more sophisticated variant
  - Can be randomized
- Each back queue is kept non-empty while the crawl is in progress
- Each back queue only contains URLs from a single host
  - Maintain a table from hosts to back queues



Host name	Back queue
…	3
	1
	B

Back queue heap

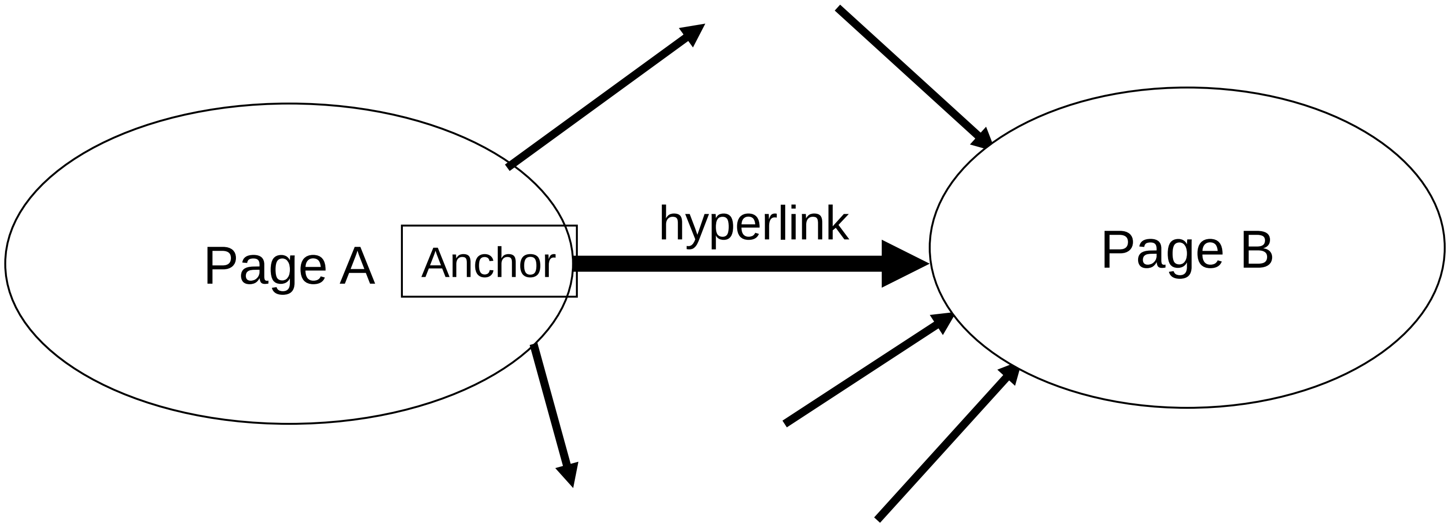
- One entry for each back queue
- The entry is the earliest time t_e at which the host corresponding to the back queue can be hit again
- This earliest time is determined from
  - Last access to that host
  - Any time buffer heuristic we choose
- **Back queue processing**
  1. A crawler thread seeking a URL to crawl:
  2. Extracts the root of the heap
  3. Fetches URL at head of corresponding back queue *q* (look up from table)
  4. Checks if queue *q* is now empty – if so, pulls a URL *v* from front queues
     - If there's already a back queue for *v*'s host, append *v* to *q* and pull another URL from front queues, repeat
     - Else add *v* to *q*
  5. When *q* is non-empty, create heap entry for it

FH OBERÖSTERREICH

University of Applied Sciences

Link Analysis

The Web as a Directed Graph



- **Assumption 1:** A hyperlink between pages denotes author perceived relevance (**quality signal**)
- **Assumption 2:** The text in the anchor of the hyperlink describes the target page (**textual context**)

Indexing anchor text

- Can sometimes have unexpected side effects - *e.g., evil empire.*
- Can score anchor text with weight depending on the authority of the anchor page's website
  - E.g., if we were to assume that content from cnn.com or yahoo.com is authoritative, then trust the anchor text from them
- **Other applications**
  - Weighting/filtering links in the graph
  - Generating page descriptions from anchor text

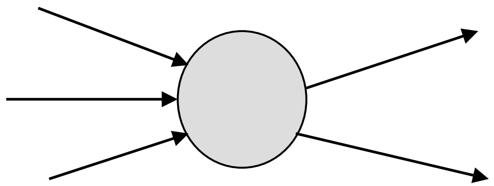
Citation Analysis → Query-independent ordering

- Citation frequency
- Co-citation coupling frequency
  - Co-citations with a given author measures “impact”
  - Co-citation analysis
- Bibliographic coupling frequency
  - Articles that co-cite the same articles are related
- **Citation indexing**
  - Who is this author cited by?
- **Pagerank** preview

Query-independent ordering

Using link counts as simple measures of popularity.

- Two basic suggestions:
  - **Undirected popularity:** Each page gets a score = the number of in-links plus the number of out-links
  - **Directed popularity:** Score of a page = number of its in-links

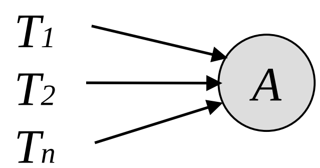


- **Query processing**
  1. First retrieve all pages meeting the text query (say *venture capital*).
  2. Order these by their link popularity
  3. More nuanced – use link counts as a measure of static goodness (PageRank) combined with text match score

Page Rank scoring

Page Rank (developed 1996 at Stanford University by Larry Page and Sergey Brin)

$$PR(A) = (1-d) + d \cdot \sum_{i=1}^{n} \frac{PR(T_i)}{C(T_i)}$$



$PR(A)$	PageRank of page A,
$PR(T_i)$	PageRank of page T_i from in-link of page A,
$C(T_i)$	number of links of page T_i
d	damping factor, between 0 and 1 (set around 0.85 according to various studies).

- Imagine a browser (a person) doing a random walk on web pages:
  - Start at a random page
  - At each step, go out of the current page along one of the links on that page, equiprobably
- In the "steady state" each page has a long-term visit rate - use this as the page's score.
- The web is full of dead-ends. Random walk can get stuck in dead-ends

Teleporting

- At a dead end, jump to a random web page.
- At any non-dead end, with probability 10%, jump to a random web page.
  - With remaining probability (90%), go out on a random link (10% - a parameter)

Page Rank

- **Page Rank** is used in google, but is hardly the full story of ranking
  - Many sophisticated features are used
  - Some address specific query classes
  - Machine learned ranking heavily used
- Page Rank still very useful for things like crawl policy
- Since August 2013: **Google Hummingbird Algorithm**
  - A **more human way** to interact with users
  - Paying more attention to each word in a query, ensuring that **the whole query** – the whole sentence or conversation or meaning – is taken into account, rather than particular words. The goal is that **pages matching the meaning do better**, rather than pages matching just a few words.

Topic Specific Pagerank

- **Goal:** pagerank values that depend on query *topic*
- Conceptually, we use a random surfer who teleports, with say 10% probability, using the following rule:
  - Select a topic (say, one of the 16 top level categories) based on a query & user -specific distribution over the categories
  - Teleport to a page uniformly at random within the chosen topic

- **Hard to implement:** can't compute PageRank at query time!

- **Offline:** Compute pagerank for individual topics
  - Query independent as before
  - Each page has multiple pagerank scores – one for each ODP category, with teleportation only to that category

Open Directory Project
http://www.dmoz.org/about.html

- **Online:** Query context classified into (distribution of weights over) topics
  - Generate a dynamic pagerank score for each page – weighted sum of topic-specific pageranks

Influencing Pagerank ("Personalization")

- **Input:**
  - Web graph $\boldsymbol{W}$
  - Influence vector $\boldsymbol{v}$ over topics
    - $\boldsymbol{v}$: (page $\rightarrow$ degree of influence)

Vector has one component for each topic

- **Output:**
  - Rank vector $\boldsymbol{r}$: (page $\rightarrow$ page importance with respect to $\boldsymbol{v}$)

  $$\boldsymbol{r} = \boldsymbol{PR(W, v)}$$

Connectivity Server

- Support for fast queries on the web graph
  - Which URLs point to a given URL?
  - Which URLs does a given URL point to?
- Stores mappings in memory from
  - URL to outlinks, URL to inlinks

Adjacency lists

- The set of neighbors of a node
- Assume each URL represented by an integer, e.g., for a 4 billion page web, need 32 bits per node
- **Adjaceny list compression (Boldi/Vigna):**
  - Similarity (between lists)
  - Locality (many links from a page go to "nearby" pages)
  - Use gap encodings in sorted lists
  - Distribution of gap values

Hyperlink-Induced Topic Search (HITS)

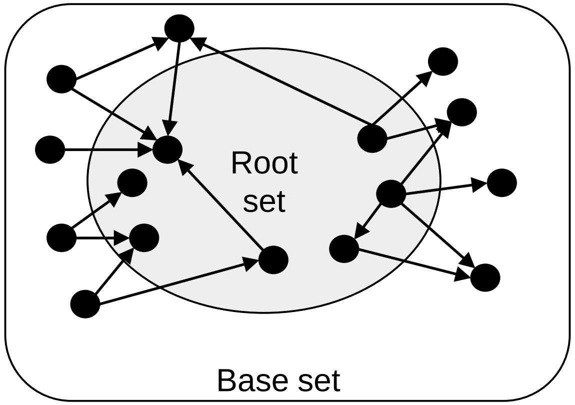
- In response to a query, instead of an ordered list of pages each meeting the query, find **two sets of inter-related pages**:
  - ***Hub pages*** are good lists of links on a subject.
  - ***Authority pages*** occur recurrently on good hubs for the subject.
- Gets at a broader slice of ***common opinion***.
- Thus, a good ***hub page*** for a topic ***points to many authoritative pages*** for that topic.
- A good ***authority page*** for a topic ***is pointed to by many good hubs*** for that topic.
- Circular definition - will turn this into an iterative computation.

High-level scheme

- Extract from the web a **base set** of pages that *could* be good hubs or authorities.
- From these, identify a small set of top hub and authority pages → iterative algorithm.

Base set

- Given text query (say *browser*), use a text index to get all pages containing *browser*.
  - Call this the **root set** of pages.
- Add in any page that either
  - points to a page in the root set, or
  - is pointed to by a page in the root set.
- Call this the **base set**.

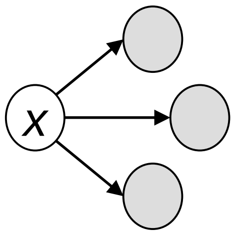


Distilling hubs and authorities

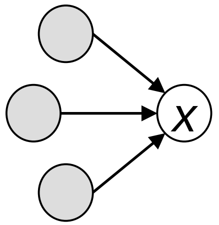
- Compute, for each page ***x*** in the base set, a **hub score *h(x)*** and an **authority score *a(x)*.**
- Initialize: for all x, $h(x) \leftarrow 1$; $a(x) \leftarrow 1$;
- Iteratively update all $h(x)$, $a(x)$;
- After iterations
  - output pages with highest $h()$ scores as top hubs
  - highest $a()$ scores as top authorities.

Iterative update

$$h(x) \leftarrow \sum_{x \mapsto y} a(y)$$



$$a(x) \leftarrow \sum_{y \mapsto x} h(y)$$



The trouble with paid search ads …

- **It costs money. What's the alternative?**
- **Search Engine Optimization:**
  - "Tuning" your web page to rank highly in the algorithmic search results for select keywords
  - Alternative to paying for placement
  - Thus, intrinsically a marketing function
- **Performed by companies, webmasters and consultants ("Search engine optimizers") for their clients**

Spam techniques

- **Cloaking**
  - Serve fake content to search engine spider
  - DNS cloaking: Switch IP address. Impersonate
- **Doorway pages**
  - Pages optimized for a single keyword that re-direct to the real target page
- **Link spamming**
  - Mutual admiration societies, hidden links, awards
  - *Domain flooding:* numerous domains that point or re-direct to a target page
- **Robots**
  - Fake query stream – rank checking programs
  - “Curve-fit” ranking programs of search engines
  - Millions of submissions via Add-Url

The war against spam

- **Quality signals - Prefer authoritative pages based on:**
  - Votes from authors (linkage signals)
  - Votes from users (usage signals)
- **Policing of URL submissions**
  - Anti robot test
- **Limits on meta-keywords**
- **Robust link analysis**
  - Ignore statistically implausible linkage (or text)
  - Use link analysis to detect spammers
  - (guilt by association)
- **Spam recognition by machine learning**
  - Training set based on known spam
- **Family friendly filters**
  - Linguistic analysis, general classification techniques, etc.
  - For images: flesh tone detectors, source text analysis, etc.
- **Editorial intervention**
  - Blacklists
  - Top queries audited
  - Complaints addressed
  - Suspect pattern detection