Lecture 5

Web Characterization and Crawling

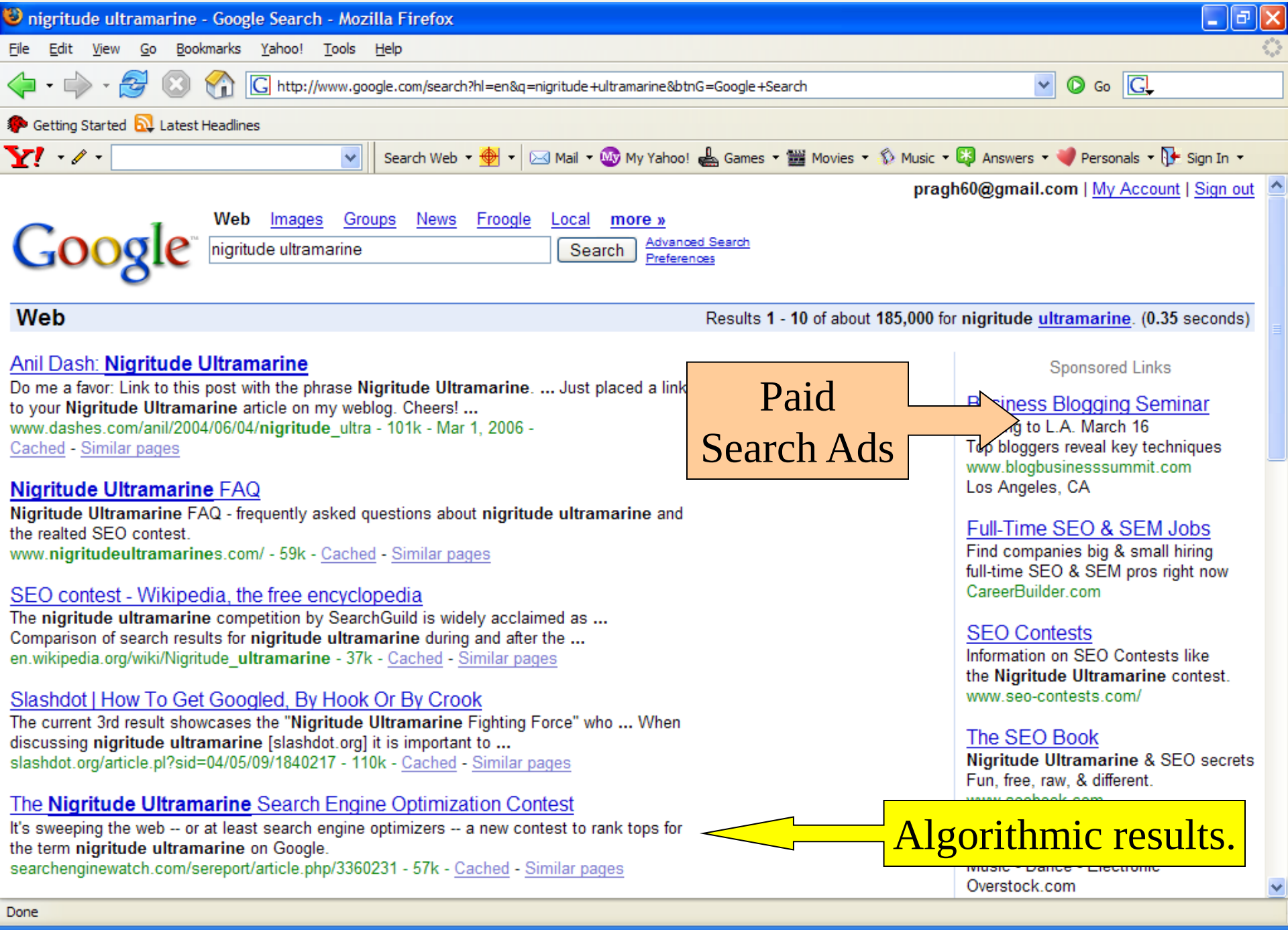
Many thanks to Prabhakar Raghavan for sharing most content from the following slides

Brief (non-technical) history

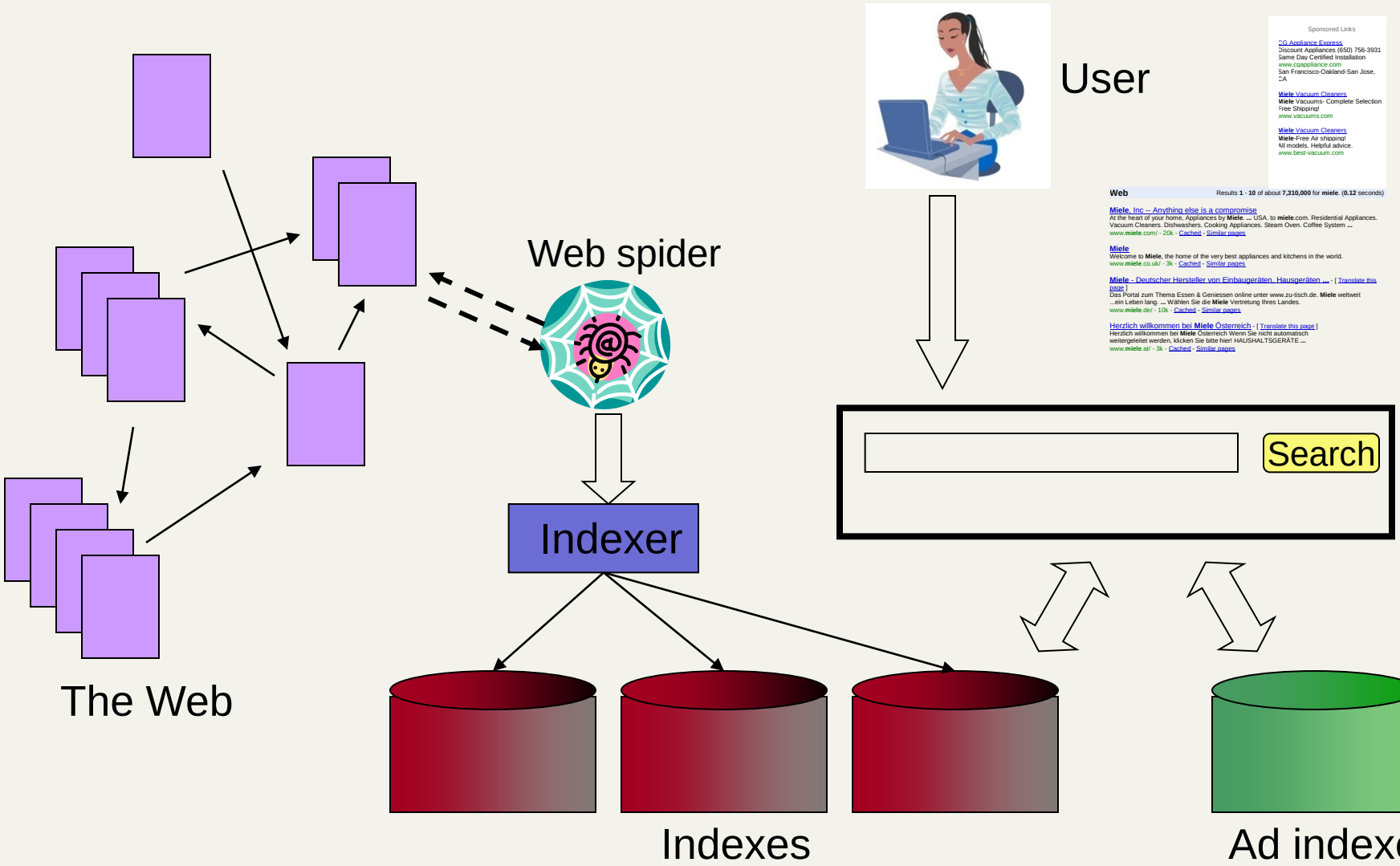
- Early keyword-based engines ca. 1995-1997
  - Altavista, Excite, Infoseek, Inktomi, Lycos
- Paid search ranking: Goto (morphed into Overture.com → Yahoo!)
  - Your search ranking depended on how much you paid
  - Auction for keywords: ***casino*** was expensive!

Brief (non-technical) history

- 1998+: Link-based ranking pioneered by Google
  - Blew away all early engines save Inktomi
  - Great user experience in search of a business model
  - Meanwhile Goto/Overture's annual revenues were nearing $1 billion
- Result: Google added paid search "ads" to the side, independent of search results
  - Yahoo followed suit, acquiring Overture (for paid placement) and Inktomi (for search)
- 2005+: Google gains search share, dominating in Europe and very strong in North America
  - 2009: Yahoo! and Microsoft propose combined paid search offering


nigritude ultramarine - Google Search - Mozilla Firefox
File Edit View Go Bookmarks Yahoo! Tools Help
http://www.google.com/search?hl=en&q=nigritude+ultramarine&btnG=Google+Search
Go
Getting Started
Latest Headlines
Search Web
Mail
My Yahoo!
Games
Movies
Music
Answers
Personals
Sign In
pragh60@gmail.com | My Account | Sign out
Google
Web Images Groups News Froogle Local more »
nigritude ultramarine
Search
Advanced Search
Preferences
Web
Results 1 - 10 of about 185,000 for nigritude ultramarine. (0.35 seconds)
Anil Dash: Nigritude Ultramarine
Do me a favor: Link to this post with the phrase Nigritude Ultramarine. ... Just placed a link to your Nigritude Ultramarine article on my weblog. Cheers! ...
www.dashes.com/anil/2004/06/04/nigritude_ultra - 101k - Mar 1, 2006 -
Cached - Similar pages
Nigritude Ultramarine FAQ
Nigritude Ultramarine FAQ - frequently asked questions about nigritude ultramarine and the realted SEO contest.
www.nigritudeultramarines.com/ - 59k - Cached - Similar pages
SEO contest - Wikipedia, the free encyclopedia
The nigritude ultramarine competition by SearchGuild is widely acclaimed as ...
Comparison of search results for nigritude ultramarine during and after the ...
en.wikipedia.org/wiki/Nigritude_ultramarine - 37k - Cached - Similar pages
Slashdot | How To Get Googled, By Hook Or By Crook
The current 3rd result showcases the "Nigritude Ultramarine Fighting Force" who ... When discussing nigritude ultramarine [slashdot.org] it is important to ...
slashdot.org/article.pl?sid=04/05/09/1840217 - 110k - Cached - Similar pages
The Nigritude Ultramarine Search Engine Optimization Contest
It's sweeping the web -- or at least search engine optimizers -- a new contest to rank tops for the term nigritude ultramarine on Google.
searchenginewatch.com/sereport/article.php/3360231 - 57k - Cached - Similar pages
Paid Search Ads
Sponsored Links
Business Blogging Seminar
g to L.A. March 16
Top bloggers reveal key techniques
www.blogbusinesssummit.com
Los Angeles, CA
Full-Time SEO & SEM Jobs
Find companies big & small hiring full-time SEO & SEM pros right now
CareerBuilder.com
SEO Contests
Information on SEO Contests like the Nigritude Ultramarine contest.
www.seo-contests.com/
The SEO Book
Nigritude Ultramarine & SEO secrets
Fun, free, raw, & different.
Overstock.com
Algorithmic results.
Done

Web search basics



User Needs

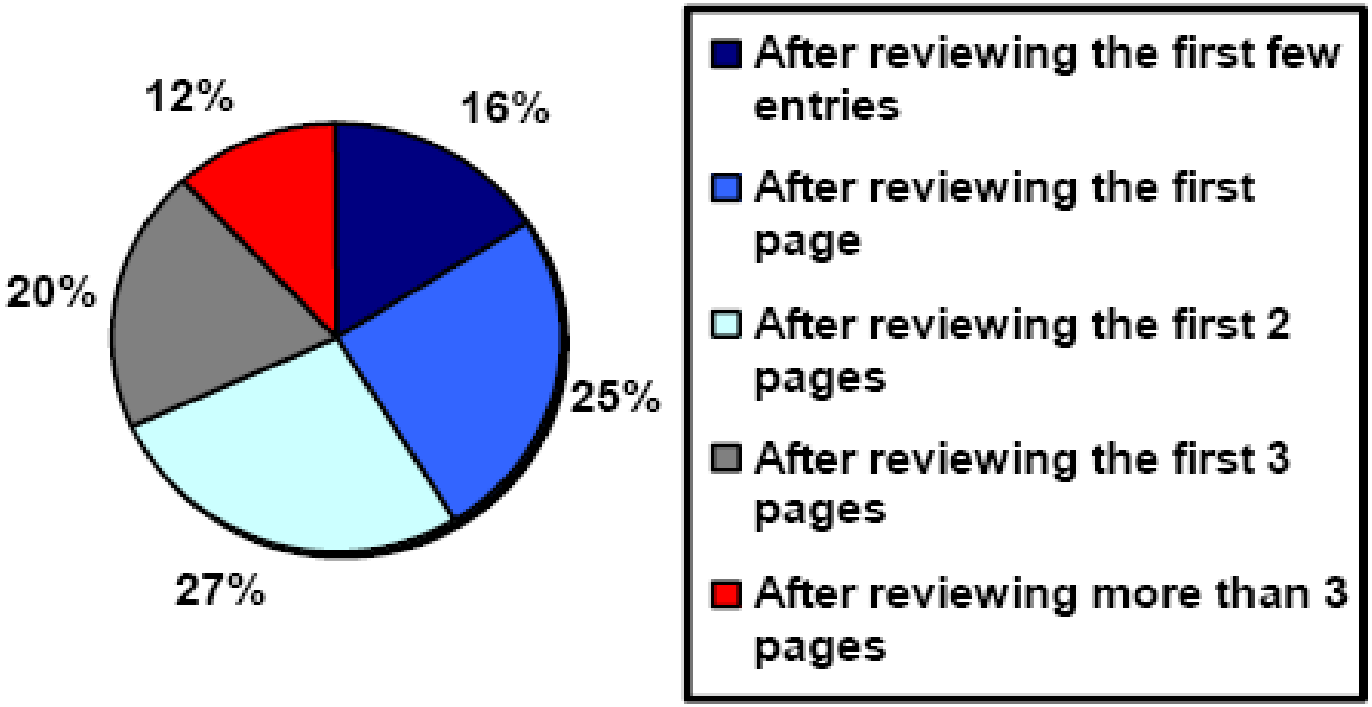
- Need [Brod02, RL04]
  - **Informational** – want to learn about something (~40% / 65%)

    Low hemoglobin
  - **Navigational** – want to go to that page (~25% / 15%)

    United Airlines
  - **Transactional** – want to do something (web-mediated) (~35% / 20%)
    - Access a service — **Seattle weather**
    - Downloads — **Mars surface images**
    - Shop — **Canon S410**
  - **Gray areas**
    - Find a good hub — **Car rental Brasil**
    - Exploratory search “see what’s there”

How far do people look for results?

"When you perform a search on a search engine and don't find what you are looking for, at what point do you typically either revise your search, or move on to another search engine? (Select one)"



(Source: iprospect.com WhitePaper_2006_SearchEngineUserBehavior.pdf)

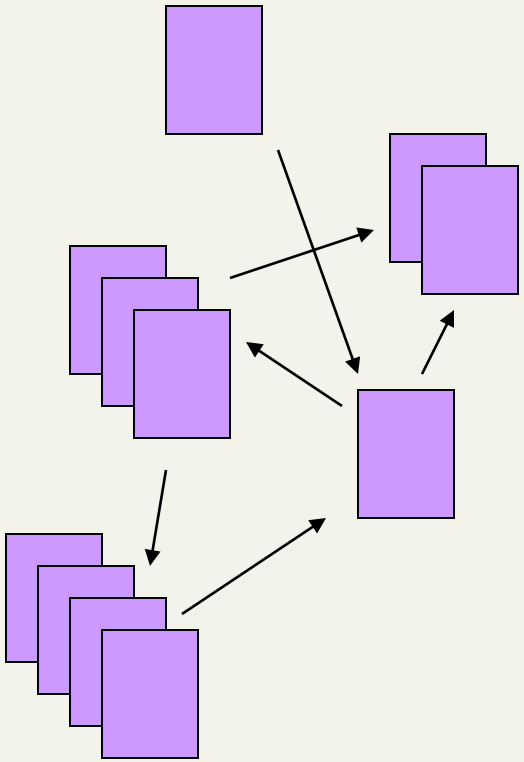
Users' empirical evaluation of results

- Quality of pages varies widely
  - Relevance is not enough
  - Other desirable qualities (non true IR, but getting in these days)
    - 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
- What matters
  - Precision at 1? Precision *above the fold*?
  - Comprehensiveness – must be able to deal with obscure queries
    - Recall matters when the number of matches is very small
- User perceptions may be unscientific, but are significant over a large aggregate

Users' empirical evaluation of engines

- Relevance and validity of results
- UI – Simple, no clutter, error tolerant
- Trust – Results are objective
- **Coverage** of topics for polysemic queries
- Pre/Post process tools provided
  - Mitigate user errors (auto spell check, search assist,…)
  - Explicit: Search within results, more like this, refine ...
  - Anticipative: related searches
- Deal with idiosyncrasies
  - Web specific vocabulary
    - Impact on stemming, spell-check, etc
  - Web addresses typed in the search box

The Web document collection



The Web

- No design/co-ordination
- Distributed content creation, linking, democratization of publishing
- Content includes truth, lies, obsolete information, contradictions …
- Unstructured (text, html, …), semi-structured (XML, annotated photos), structured (Databases)…
- Scale much larger than previous text collections … but corporate records are catching up (interesting)
- Growth – slowed down from initial “volume doubling every few months” but still expanding
- Content can be *dynamically generated*

Search Engine Optimization

The trouble with paid search ads …

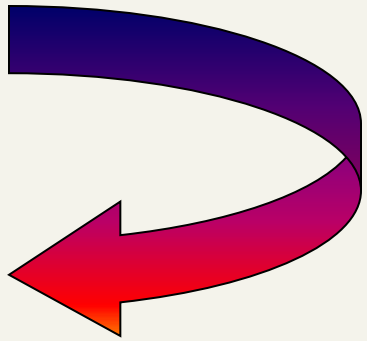
- It costs money. What's the alternative?
- *Search Engine Optimization:*
  - "Tuning" your web page to rank highly in the algorithmic search results for select keywords
  - Alternative to paying for placement
  - Thus, intrinsically a marketing function
- Performed by companies, webmasters and consultants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

Search engine optimization

- Motives
  - Commercial, political, religious, lobbies
  - Promotion funded by advertising budget
- Operators
  - Contractors (Search Engine Optimizers) for lobbies, companies
  - Web masters
  - Hosting services
- Forums
  - E.g., Web master world (www.webmasterworld.com)
    - Search engine specific tricks
    - Discussions about academic papers ☺

Simplest forms

- First generation engines relied heavily on *tf/idf*
  - The top-ranked pages for the query **`maui resort`** were the ones containing the most **`maui`**'s and **`resort`**'s
- SEOs responded with dense repetitions of chosen terms
  - e.g., **`maui resort maui resort maui resort`**
  - Often, the repetitions would be in the same color as the background of the web page
    - Repeated terms got indexed by crawlers
    - But not visible to humans on browsers



Pure word density cannot be trusted as an IR signal

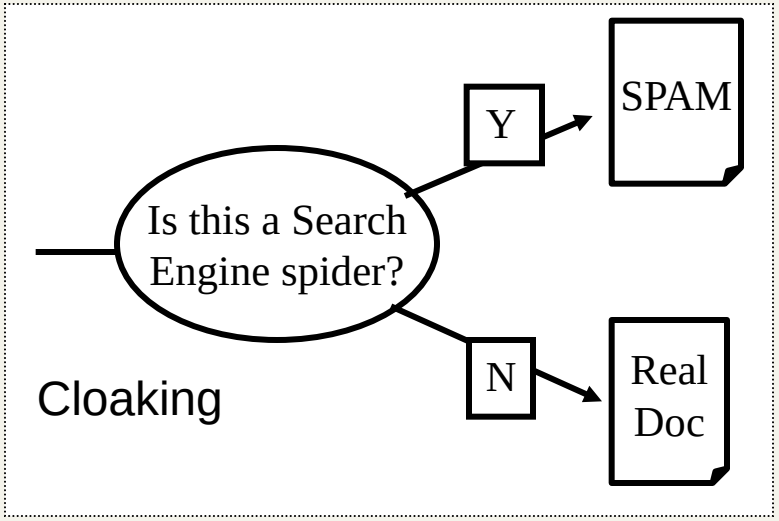
Variants of keyword stuffing

- Misleading meta-tags, excessive repetition
- Hidden text with colors, style sheet tricks, etc.

Meta-Tags =
"… London hotels, hotel, holiday inn, hilton, discount, booking, reservation, sex, mp3, britney spears, viagra, …"

Cloaking

- Serve fake content to search engine spider
- DNS cloaking: Switch IP address. Impersonate



More spam techniques

- **Doorway pages**
  - Pages optimized for a single keyword that re-direct to the real target page
- **Link spamming**
  - Mutual admiration societies, hidden links, awards – more on these later
  - *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

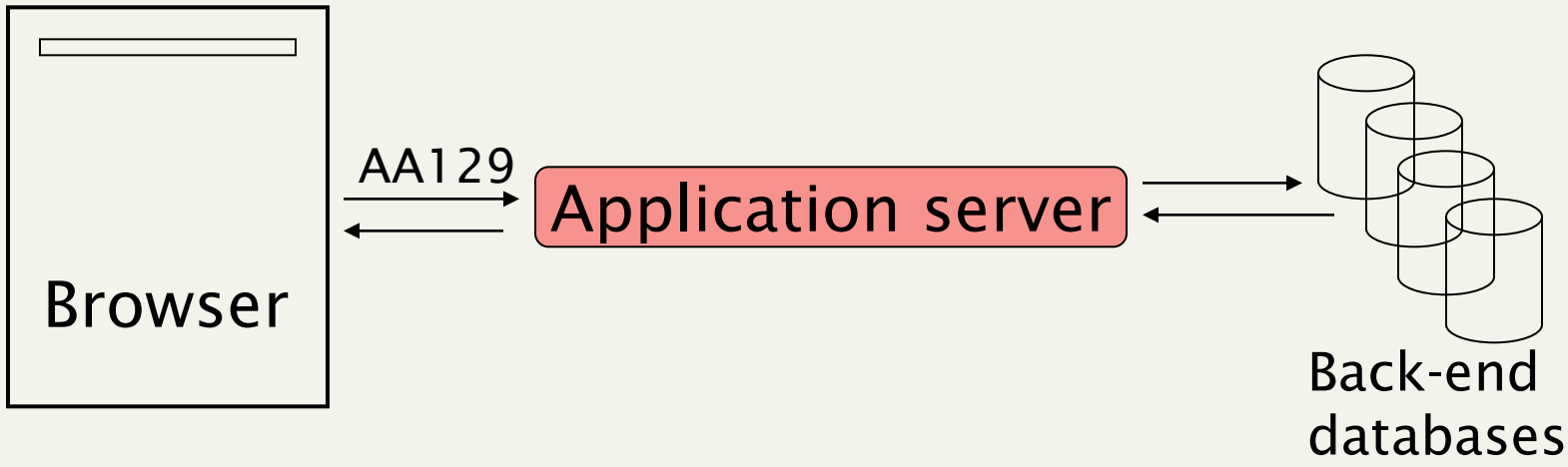
More on spam

- Web search engines have policies on SEO practices they tolerate/block
  - http://styleguide.yahoo.com/
  - http://www.google.com/intl/en/webmasters/
- Adversarial IR: the unending (technical) battle between SEO's and web search engines
- Research http://airweb.cse.lehigh.edu/

Size of the web

The Web: Dynamic content

- A page without a static html version
  - E.g., current status of flight AA129
  - Current availability of rooms at a hotel
- Usually, assembled at the time of a request from a browser
  - Typically, URL has a '?' character in it



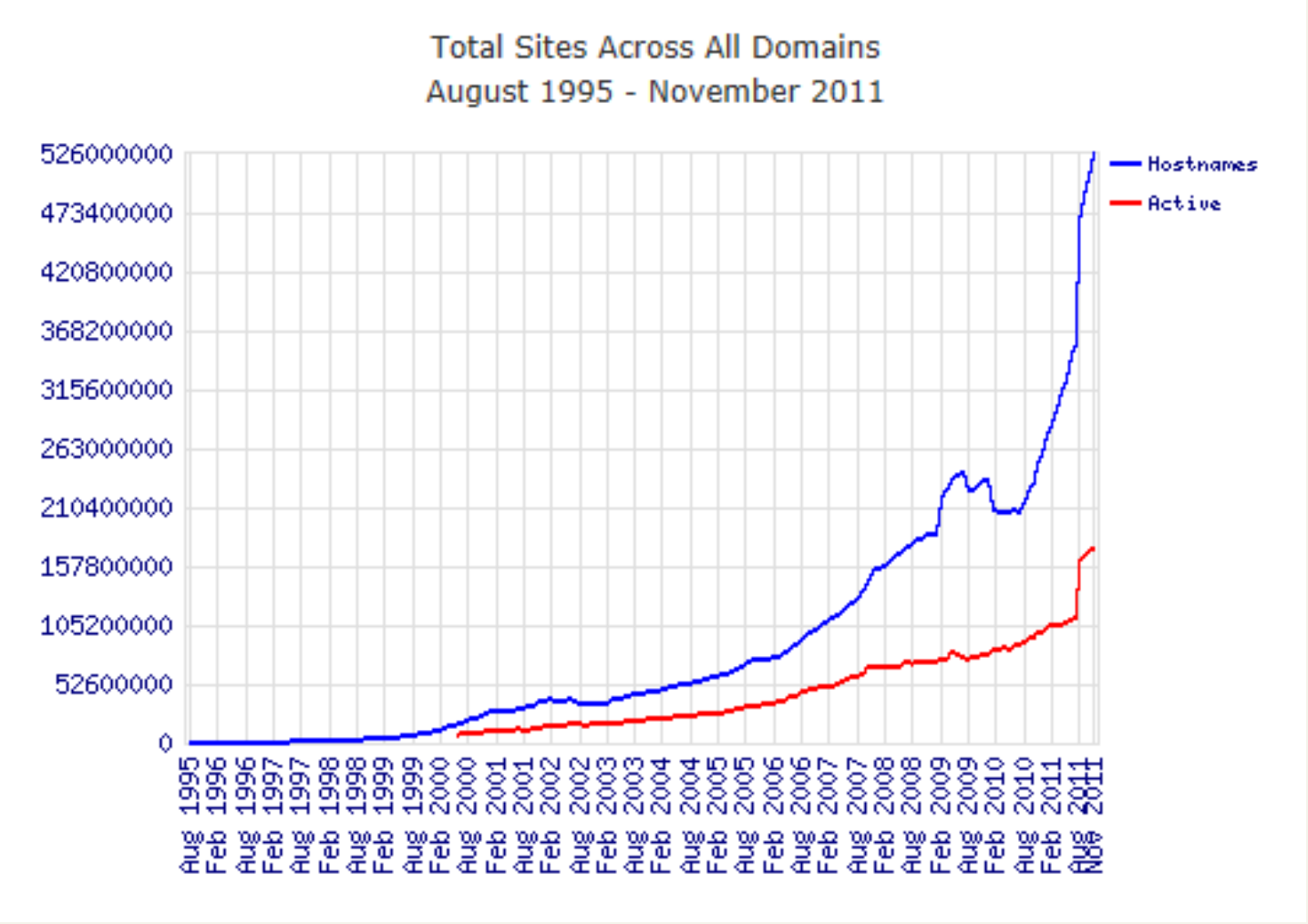
Dynamic content

- Most dynamic content is ignored by web spiders
  - Many reasons including malicious spider traps
- Some dynamic content (news stories from subscriptions) are sometimes delivered as static content
  - Application-specific spidering
- Spiders commonly view web pages just as Lynx (a text browser) would
- Note: even “static” pages are typically assembled on the fly (e.g., headers are common)

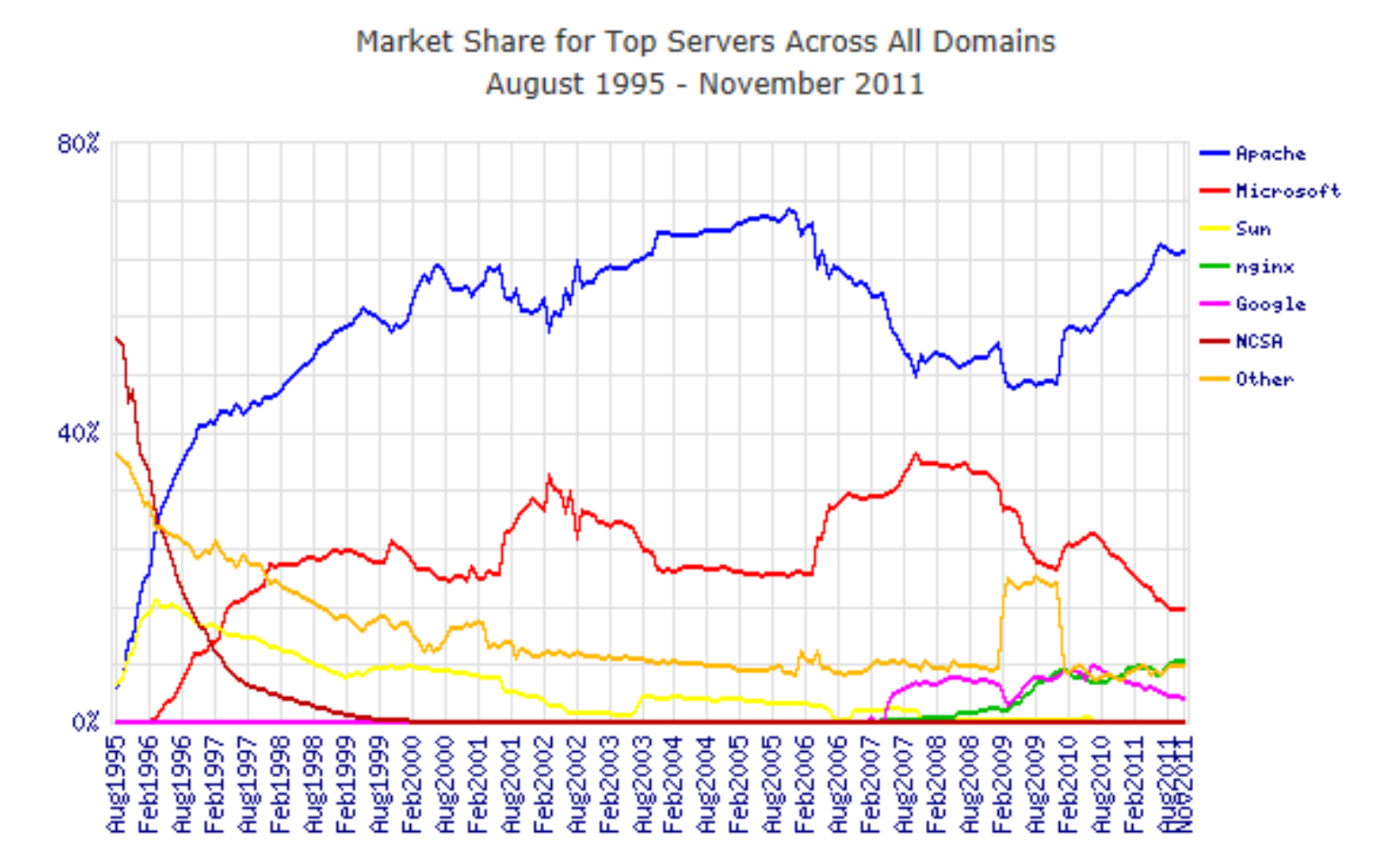
The web: size

- What is being measured?
  - Number of hosts – netcraft survey
    - http://news.netcraft.com/archives/web_server_survey.html
    - Monthly report on how many web hosts & servers are out there
  - Number of static (html) pages – numerous estimates (will discuss later)

The web: size


Total Sites Across All Domains
August 1995 - November 2011
526000000
473400000
420800000
368200000
315600000
263000000
210400000
157800000
105200000
52600000
0
Hostnames
Active
Aug 1995
Feb 1996
Aug 1996
Feb 1997
Aug 1997
Feb 1998
Aug 1998
Feb 1999
Aug 1999
Feb 2000
Aug 2000
Feb 2001
Aug 2001
Feb 2002
Aug 2002
Feb 2003
Aug 2003
Feb 2004
Aug 2004
Feb 2005
Aug 2005
Feb 2006
Aug 2006
Feb 2007
Aug 2007
Feb 2008
Aug 2008
Feb 2009
Aug 2009
Feb 2010
Aug 2010
Feb 2011
Aug 2011
Nov 2011

The web: size


Market Share for Top Servers Across All Domains
August 1995 - November 2011
80%
40%
0%
Apache
Microsoft
Sun
nginx
Google
NCSA
Other
Aug1995
Feb1996
Aug1996
Feb1997
Aug1997
Feb1998
Aug1998
Feb1999
Aug1999
Feb2000
Aug2000
Feb2001
Aug2001
Feb2002
Aug2002
Feb2003
Aug2003
Feb2004
Aug2004
Feb2005
Aug2005
Feb2006
Aug2006
Feb2007
Aug2007
Feb2008
Aug2008
Feb2009
Aug2009
Feb2010
Aug2010
Feb2011
Aug2011
Nov2011

The web: evolution

- All of these numbers keep changing
- Relatively few scientific studies of the evolution of the web [Fetterly & al, 2003]
  - http://research.microsoft.com/en-us/projects/pageturner/pageturner-spe2004.pdf
- Sometimes possible to extrapolate from small samples (fractal models) [Dill & al, 2001]
  - http://www.vldb.org/conf/2001/P069.pdf

Rate of change

- [Cho00] 720K pages from 270 popular sites sampled daily from Feb 17 – Jun 14, 1999
  - Any changes: 40% weekly, 23% daily
- [Fett02] Massive study 151M pages checked over few months
  - Significant changed -- 7% weekly
  - Small changes – 25% weekly
- [Ntul04] 154 large sites re-crawled from scratch weekly
  - 8% new pages/week
  - 8% die
  - 5% new content
  - 25% new links/week

What is the size of the web ?

- Issues
  - The web is really infinite
    - Dynamic content, e.g., calendar
    - Soft 404: www.yahoo.com/<anything> is a valid page
  - Static web contains syntactic duplication, mostly due to mirroring (~30%)
  - Some servers are seldom connected
- Who cares?
  - Media, and consequently the user
  - Engine design
  - Engine crawl policy. Impact on recall.

What can we attempt to measure?

- The relative sizes of search engines
  - The notion of a page being indexed is still *reasonably* well defined.
  - Already there are problems
    - Document extension: e.g. engines index pages not yet crawled, by indexing anchortext.
    - Document restriction: All engines restrict what is indexed (first *n* words, only relevant words, etc.)
- The coverage of a search engine relative to another particular crawling process.

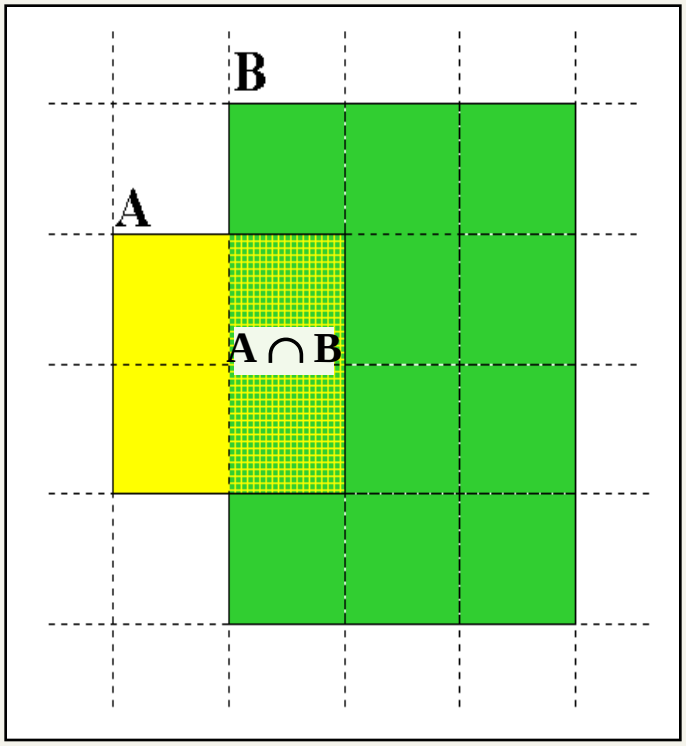
New definition?

(IQ is whatever the IQ tests measure.)

- The statically indexable web is whatever search engines index.

- Different engines have different preferences
  - max url depth, max count/host, anti-spam rules, priority rules, etc.
- Different engines index different things under the same URL:
  - frames, meta-keywords, document restrictions, document extensions, ...

Relative Size from Overlap
Given two engines A and B



Sample URLs randomly from A

Check if contained in B and vice versa

$$A \cap B = (1/2) * \text{Size } A$$

$$A \cap B = (1/6) * \text{Size } B$$

$$(1/2)*\text{Size } A = (1/6)*\text{Size } B$$

$$\therefore \text{ Size } A / \text{ Size } B = (1/6)/(1/2) = 1/3$$

Each test involves: (i) Sampling (ii) Checking

Sampling URLs

- Ideal strategy: Generate a random URL and check for containment in each index.
- Problem: Random URLs are hard to find! Enough to generate a random URL contained in a given Engine.
- Approach 1: Generate a random URL contained in a given engine
  - Suffices for the estimation of relative size
- Approach 2: Random walks / IP addresses
  - In theory: might give us a true estimate of the size of the web (as opposed to just relative sizes of indexes)

Statistical methods

- Approach 1
  - Random queries
  - Random searches
- Approach 2
  - Random IP addresses
  - Random walks

Random URLs from random queries

- Generate random query: how?
  - **Lexicon:** 400,000+ words from a web crawl
  - **Conjunctive Queries:** w_1 and w_2

    e.g., vocalists AND rsi
- Get 100 result URLs from engine A
- Choose a random URL as the candidate to check for presence in engine B
- This distribution induces a probability weight W(p) for each page.
- Conjecture: $W(SE_A) / W(SE_B) \sim |SE_A| / |SE_B|$

Query Based Checking

- *Strong Query* to check whether an engine *B* has a document *D*:
  - Download *D*. Get list of words.
  - Use 8 low frequency words as AND query to *B*
  - Check if *D* is present in result set.
- Problems:
  - Near duplicates
  - Frames
  - Redirects
  - Engine time-outs
  - Is 8-word query good enough?

Advantages & disadvantages

- Statistically sound under the induced weight.
- Biases induced by random query
  - Query Bias: Favors content-rich pages in the language(s) of the lexicon
  - Ranking Bias: *Solution:* Use conjunctive queries & fetch all
  - Checking Bias: Duplicates, impoverished pages omitted
  - Document or query restriction bias: engine might not deal properly with 8 words conjunctive query
  - Malicious Bias: Sabotage by engine
  - Operational Problems: Time-outs, failures, engine inconsistencies, index modification.

Random searches

- Choose random searches extracted from a local log [Lawrence & Giles 97] or build “random searches” [Notess]
  - Use only queries with small result sets.
  - Count normalized URLs in result sets.
  - Use ratio statistics

Advantages & disadvantages

- Advantage
  - Might be a better reflection of the human perception of coverage
- Issues
  - Samples are correlated with source of log
  - Duplicates
  - Technical statistical problems (must have non-zero results, ratio average not statistically sound)

Random searches

- 575 & 1050 queries from the NEC RI employee logs
- 6 Engines in 1998, 11 in 1999
- Implementation:
  - Restricted to queries with < 600 results in total
  - Counted URLs from each engine after verifying query match
  - Computed size ratio & overlap for individual queries
  - Estimated index size ratio & overlap by averaging over all queries

Queries from Lawrence and Giles study

- *adaptive access control*
- *neighborhood preservation topographic*
- *hamiltonian structures*
- *right linear grammar*
- *pulse width modulation neural*
- *unbalanced prior probabilities*
- *ranked assignment method*
- *internet explorer favourites importing*
- *karvel thornber*
- *zili liu*
- *softmax activation function*
- *bose multidimensional system theory*
- *gamma mlp*
- *dvi2pdf*
- *john oliensis*
- *rieke spikes exploring neural*
- *video watermarking*
- *counterpropagation network*
- *fat shattering dimension*
- *abelson amorphous computing*

Random IP addresses

- Generate random IP addresses
- Find a web server at the given address
  - If there’s one
- Collect all pages from server
  - From this, choose a page at random

Advantages & disadvantages

- Advantages
  - Clean statistics
  - Independent of crawling strategies
- Disadvantages
  - Doesn’t deal with duplication
  - Many hosts might share one IP, or not accept requests
  - No guarantee all pages are linked to root page.
    - Eg: employee pages
  - Power law for # pages/hosts generates bias towards sites with few pages.
    - But bias can be accurately quantified IF underlying distribution understood
  - Potentially influenced by spamming (multiple IP’s for same server to avoid IP block)

Random walks

- View the Web as a directed graph
- Build a random walk on this graph
  - Includes various “jump” rules back to visited sites
    - Does not get stuck in spider traps!
    - Can follow all links!
  - Converges to a stationary distribution
    - Must assume graph is finite and independent of the walk.
    - Conditions are not satisfied (cookie crumbs, flooding)
    - Time to convergence not really known
  - Sample from stationary distribution of walk
  - Use the “strong query” method to check coverage by SE

Advantages & disadvantages

- Advantages
  - “Statistically clean” method at least in theory!
  - Could work even for infinite web (assuming convergence) under certain metrics.
- Disadvantages
  - List of seeds is a problem.
  - Practical approximation might not be valid.
  - Non-uniform distribution
    - Subject to link spamming

Conclusions

- No sampling solution is perfect.
- Lots of new ideas ...
- ....but the problem is getting harder
- Quantitative studies are fascinating and a good research problem

Duplicate detection

Duplicate documents

- The web is full of duplicated content
- Strict duplicate detection = exact match
  - Not as common
- But many, many cases of near duplicates
  - E.g., Last modified date the only difference between two copies of a page

Duplicate/Near-Duplicate Detection

- *Duplication*: Exact match can be detected with fingerprints
- *Near-Duplication*: Approximate match
  - Overview
    - 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

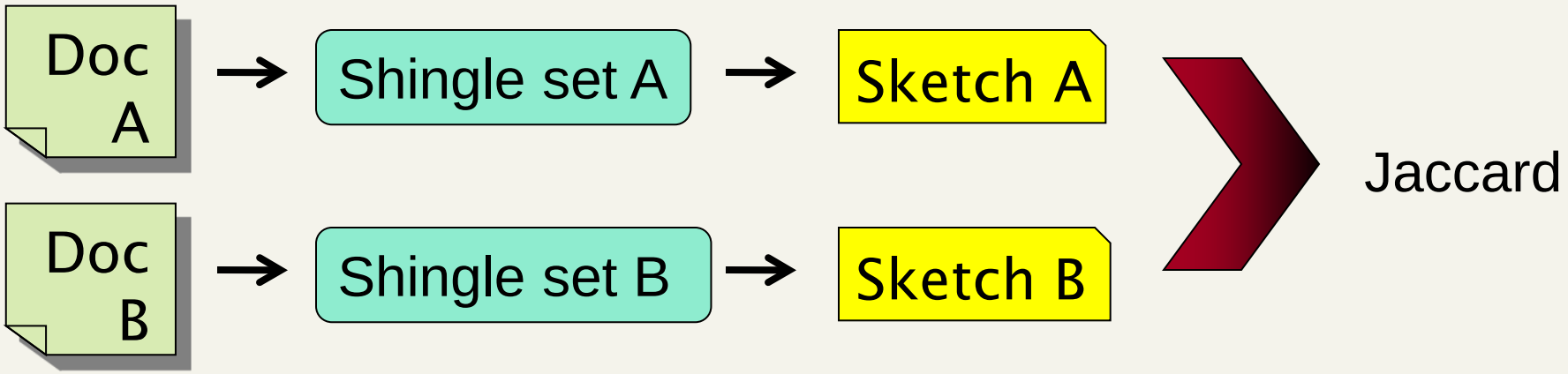
    rose_is_a_rose

    is_a_rose_is

    a_rose_is_a
- Similarity Measure between two docs (= sets of shingles)
  - Set intersection
  - Specifically (Size_of_Intersection / Size_of_Union)

Shingles + Set Intersection

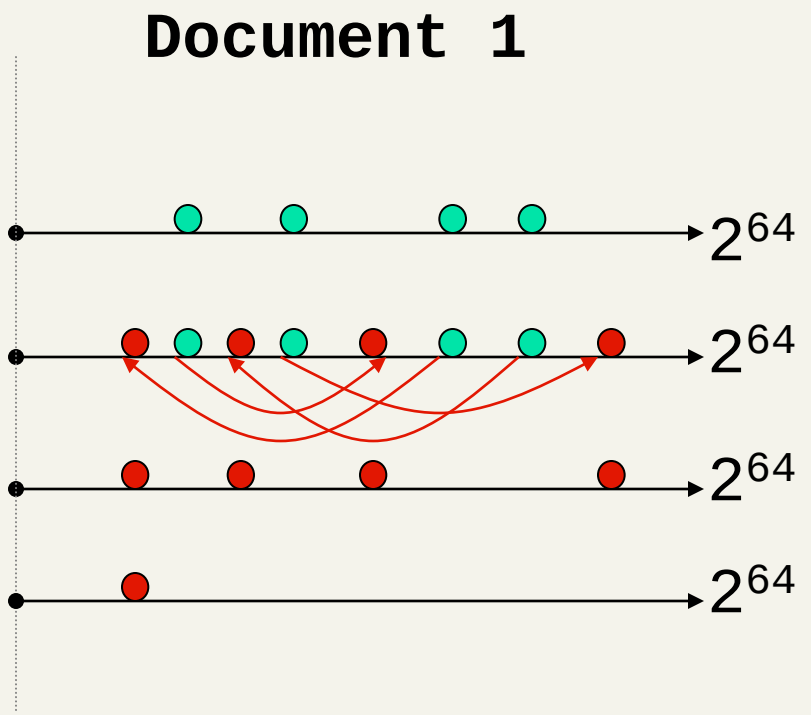
- Computing exact set intersection of shingles between all pairs of documents is expensive/intractable
  - Approximate using a cleverly chosen subset of shingles from each (a *sketch*)
- Estimate (size_of_intersection / size_of_union) based on a short sketch



Sketch of a document

- Create a “sketch vector” (of size ~200) for each document
  - Documents that share ≥ *t* (say 80%) corresponding vector elements are near duplicates
  - For doc *D*, sketch$_D$[*i*] is as follows:
    - Let f map all shingles in the universe to 0..2^m (e.g., f = fingerprinting)
    - Let π_i be a *random permutation* on 0..2^m
    - Pick MIN {π_i(f(s))} over all shingles *s* in *D*

Computing Sketch[i] for Doc1

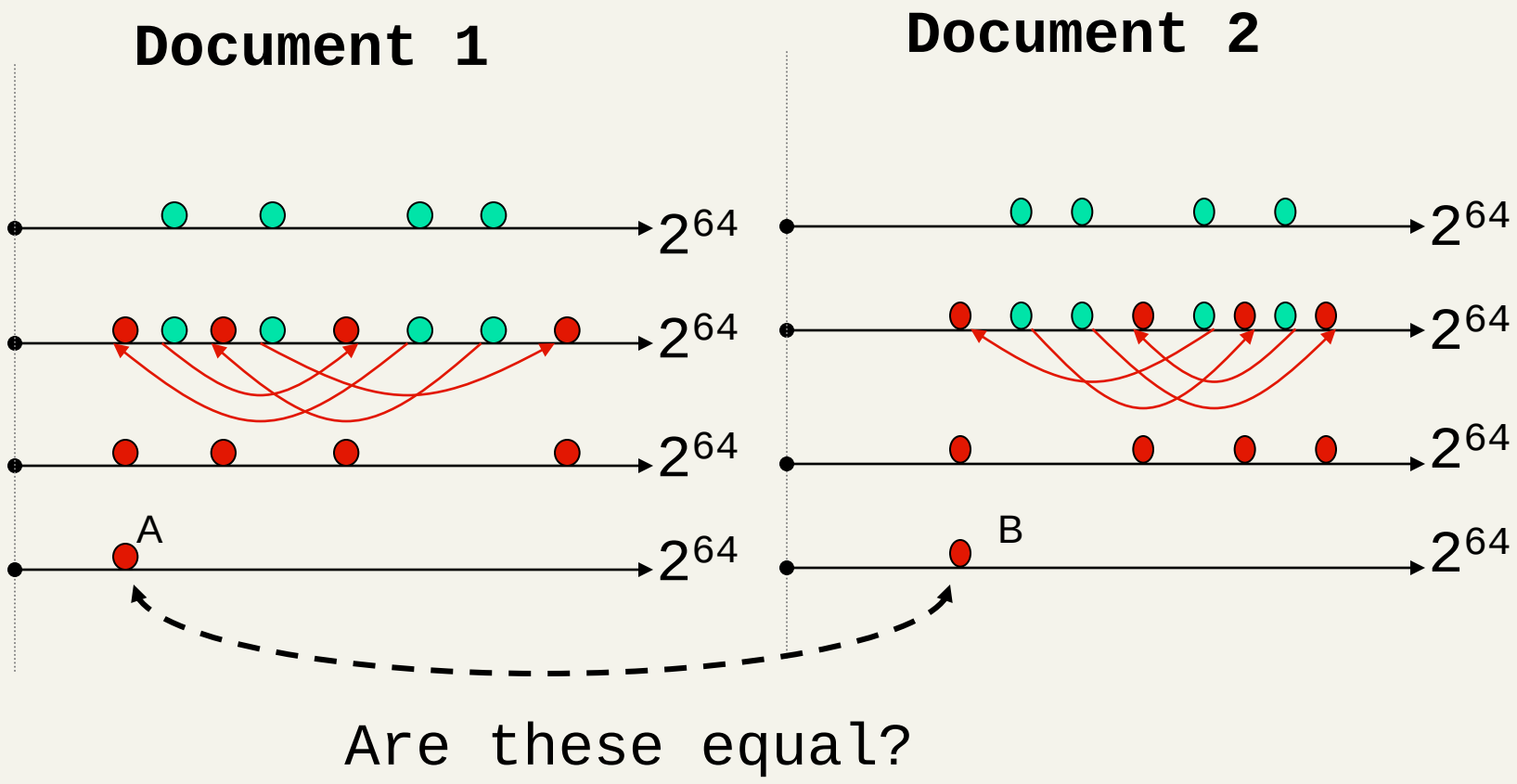


Start with 64-bit *f*(shingles)

Permute on the number line with π_i

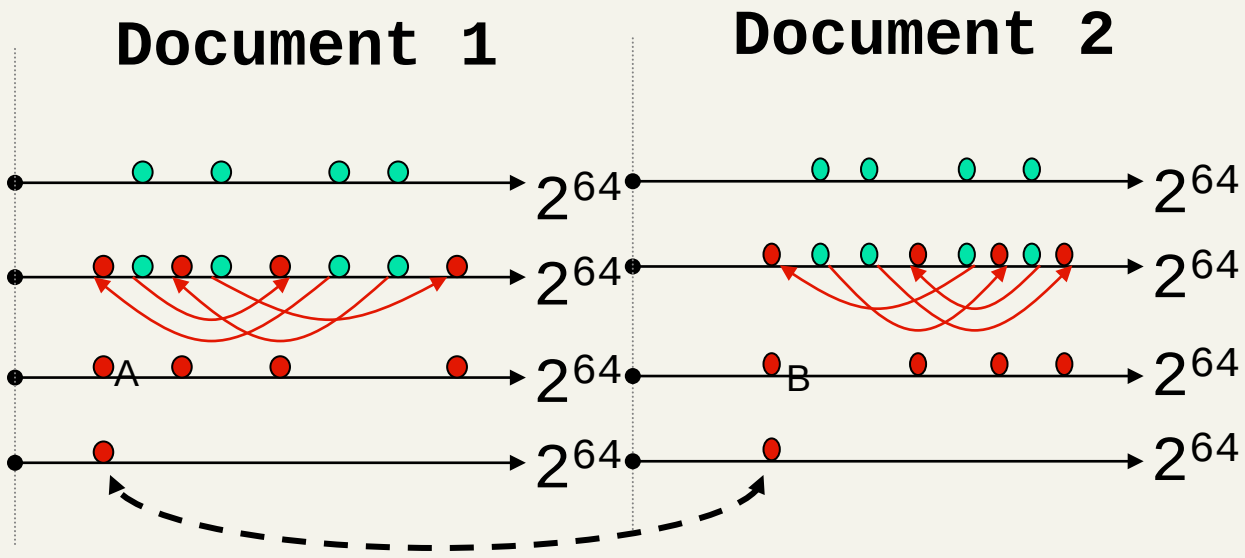
Pick the min value

Test if Doc1.Sketch[i] = Doc2.Sketch[i]



Test for 200 random permutations: $\pi_1, \pi_2, \ldots \pi_{200}$

However…



A = B iff the shingle with the MIN value in the union of Doc1 and Doc2 is common to both (i.e., lies in the intersection)

Claim: This happens with probability

`Size_of_intersection / Size_of_union`

Set Similarity of sets C_i , C_j

$$\text{Jaccard}(C_i, C_j) = \frac{|C_i \cap C_j|}{|C_i \cup C_j|}$$

- View sets as columns of a matrix A; one row for each element in the universe. a_{ij} = 1 indicates presence of item i in set j
- Example

C_1	C_2
0	1
1	0
1	1
0	0
1	1
0	1

Jaccard(C_1, C_2) = 2 / 5 = 0.4

Key Observation

- For columns C_i, C_j, four types of rows

	C_i	C_j
A	1	1
B	1	0
C	0	1
D	0	0

- Overload notation: A = # of rows of type A
- **Claim**

$$\text{Jaccard}(C_i, C_j) = \frac{A}{A + B + C}$$

“Min” Hashing

- Randomly permute rows
- Hash $h(C_i)$ = index of first row with 1 in column C_i
- Surprising Property

- Why? $P\left[h(C_i) = h(C_j)\right] = \text{Jaccard}(C_i, C_j)$
  - Both are A/(A+B+C)
  - Look down columns C_i, C_j until first non-Type-D row
  - $h(C_i) = h(C_j)$ ←→ type A row

Min-Hash sketches

- Pick *P* random row permutations
- MinHash sketch

$Sketch_D$ = list of *P* indexes of first rows with 1 in column C

- Similarity of signatures
  - Let sim[sketch(C_i),sketch(C_j)] = fraction of permutations where MinHash values agree
  - Observe E[sim(sig(C_i),sig(C_j))] = Jaccard(C_i,C_j)

Example

	C_1	C_2	C_3
R_1	1	0	1
R_2	0	1	1
R_3	1	0	0
R_4	1	0	1
R_5	0	1	0

Signatures

	S_1	S_2	S_3
Perm 1 = (12345)	1	2	1
Perm 2 = (54321)	4	5	4
Perm 3 = (34512)	3	5	4

Similarities

	1-2	1-3	2-3
Col-Col	0.00	0.50	0.25
Sig-Sig	0.00	0.67	0.00

Implementation Trick

- Permuting universe even once is prohibitive
- **Row Hashing**
  - Pick P hash functions h_k: {1,...,n}→{1,...,O(n)}
  - Ordering under h_k gives random permutation of rows
- **One-pass Implementation**
  - For each C_i and h_k, keep "slot" for min-hash value
  - Initialize all slot(C_i,h_k) to infinity
  - Scan rows in arbitrary order looking for 1's
    - Suppose row R_j has 1 in column C_i
    - For each h_k,
      - if $h_k(j) < slot(C_i,h_k)$, then $slot(C_i,h_k) \leftarrow h_k(j)$

Example

	C_1	C_2
R_1	1	0
R_2	0	1
R_3	1	1
R_4	1	0
R_5	0	1

$h(x) = x \bmod 5$

$g(x) = 2x+1 \bmod 5$

	C_1 slots	C_2 slots
$h(1) = 1$	1	-
$g(1) = 3$	3	-
$h(2) = 2$	1	2
$g(2) = 0$	3	0
$h(3) = 3$	1	2
$g(3) = 2$	2	0
$h(4) = 4$	1	2
$g(4) = 4$	2	0
$h(5) = 0$	1	0
$g(5) = 1$	2	0

Comparing Signatures

- Signature Matrix S
  - Rows = Hash Functions
  - Columns = Columns
  - Entries = Signatures
- Can compute – Pair-wise similarity of any pair of signature columns

All signature pairs

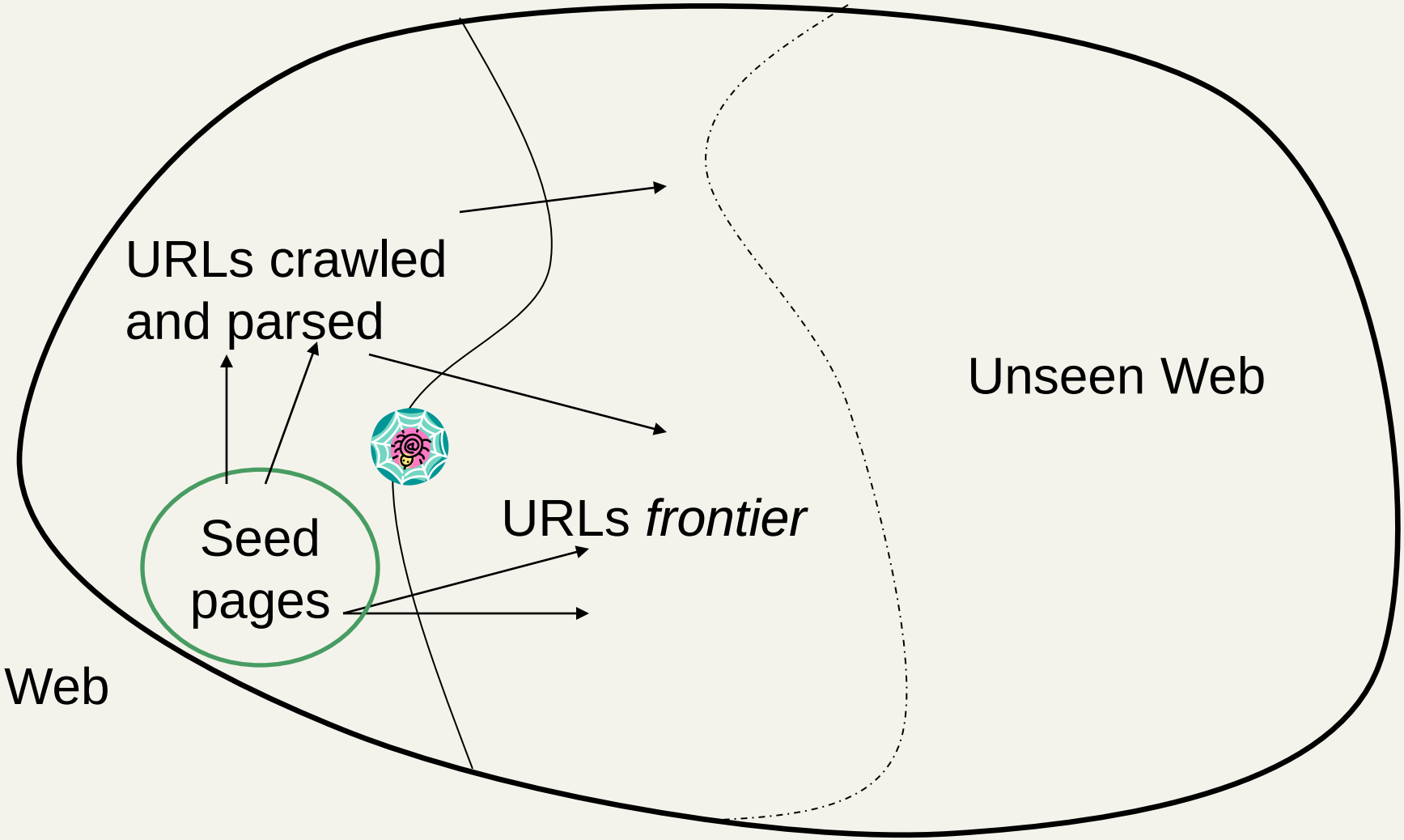
- Now we have an extremely efficient method for estimating a Jaccard coefficient for a single pair of documents.
- But we still have to estimate N^2 coefficients where N is the number of web pages.
  - Still slow
- One solution: locality sensitive hashing (LSH)
- Another solution: sorting (Henzinger 2006)

Web Crawling

Basic crawler operation

- Begin with known “seed” pages
- Fetch and parse them
  - Extract URLs they point to
  - Place the extracted URLs on a queue
- Fetch each URL on the queue and repeat

Crawling picture



Simple picture – complications

- Web crawling isn't feasible with one machine
  - All of the above steps distributed
- 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
- Malicious pages
  - Spam pages
  - Spider traps – incl dynamically generated
- Politeness – don't hit a server too often

What any crawler *must* do

- Be Polite: Respect implicit and explicit politeness considerations for a website
  - Only crawl pages you're allowed to
  - Respect *robots.txt* (more on this shortly)
- Be Robust: Be immune to spider traps and other malicious behavior from web servers

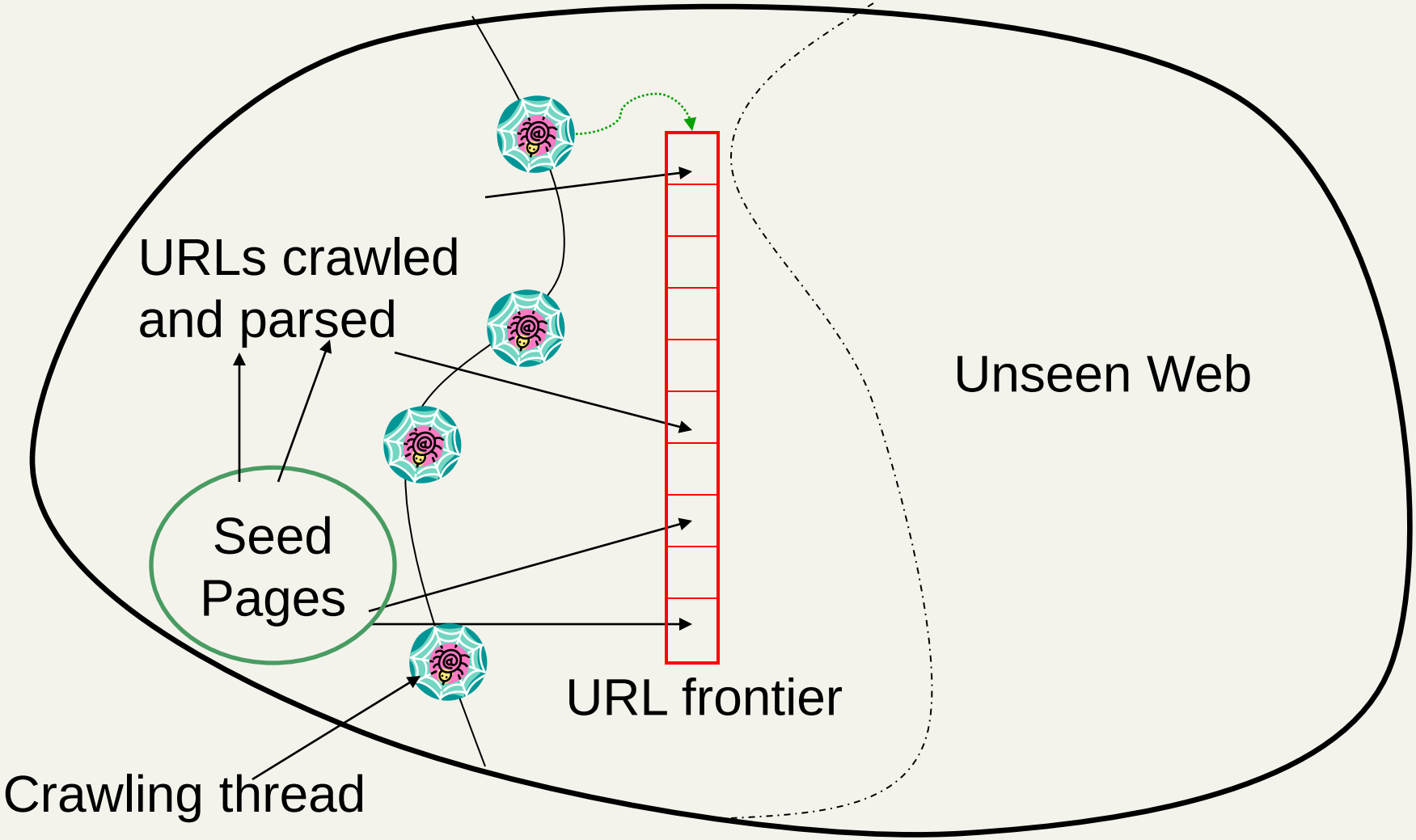
What any crawler *should* do

- 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

What any crawler *should* do

- Fetch pages of “higher quality” first
- Continuous operation: Continue fetching fresh copies of a previously fetched page
- Extensible: Adapt to new data formats, protocols

Updated crawling picture



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

Explicit and implicit politeness

- 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

Robots.txt

- Protocol for giving spiders (“robots”) limited access to a website, originally from 1994
  - www.robotstxt.org/wc/norobots.html
- Website announces its request on what can(not) be crawled
  - For a URL, create a file `URL/robots.txt`
  - This file specifies access restrictions

Robots.txt example

- No robot should visit any URL starting with "/yoursite/temp/", except the robot called “searchengine":

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

User-agent: searchengine
Disallow:
```

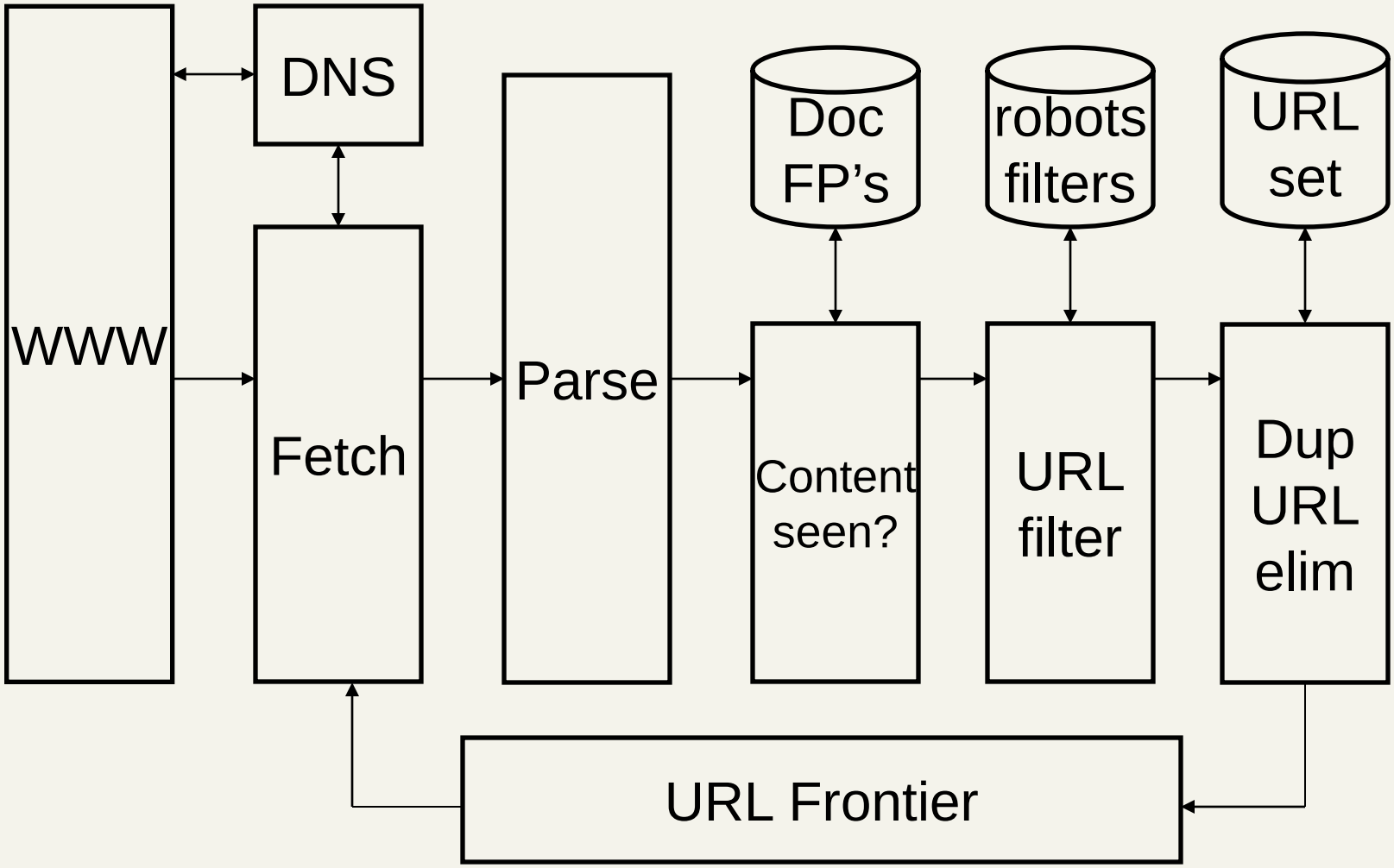
Processing steps in crawling

- Pick a URL from the frontier
- Fetch the document at the URL
- Parse the URL
  - Extract links from it to other docs (URLs)
- Check if URL has content already seen
  - If not, add to indexes
- 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 crawl architecture



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)
- Common OS implementations of DNS lookup are *blocking*: only one outstanding request at a time
- Solutions
  - DNS caching
  - Batch DNS resolver – collects requests and sends them out together

Parsing: URL normalization

- When a fetched document is parsed, 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

- During parsing, must normalize (expand) such 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

Filters and robots.txt

- Filters – regular expressions for URL's to be crawled/not
- 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

Duplicate URL elimination

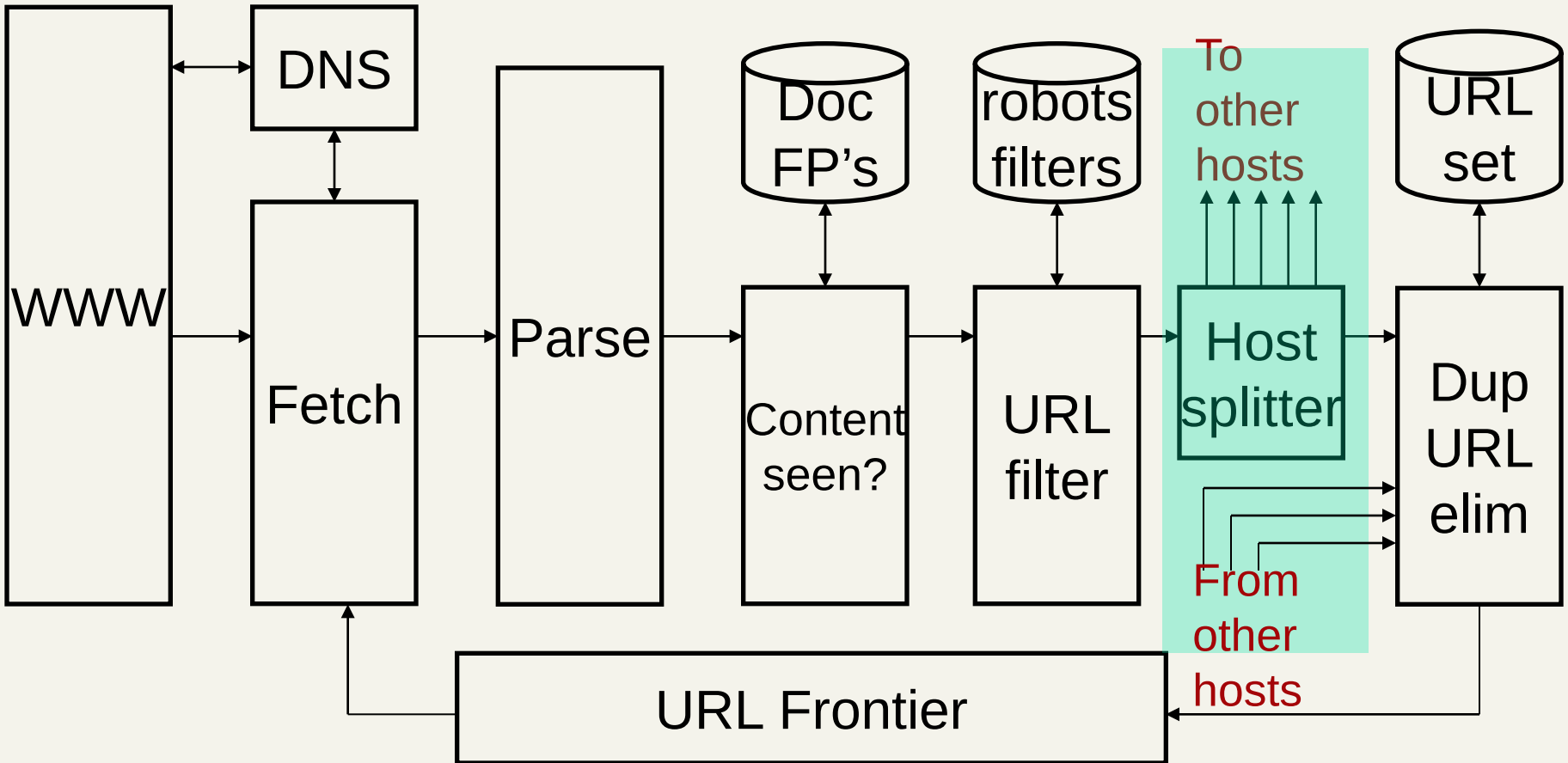
- For a non-continuous (one-shot) crawl, test to see if an extracted+filtered URL has already been passed to the frontier
- For a continuous crawl – see details of frontier implementation

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

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

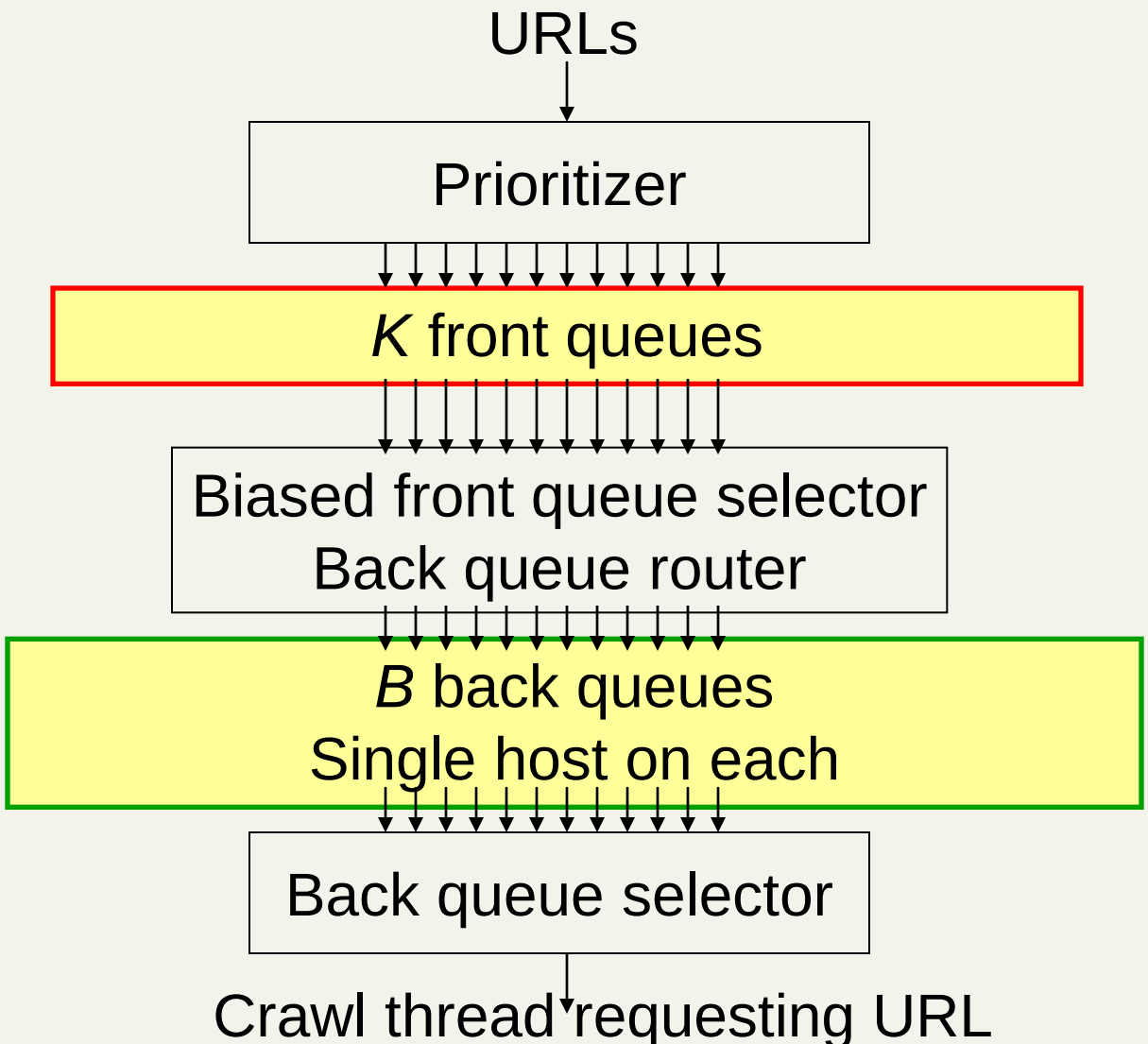
These goals may conflict each other.

(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

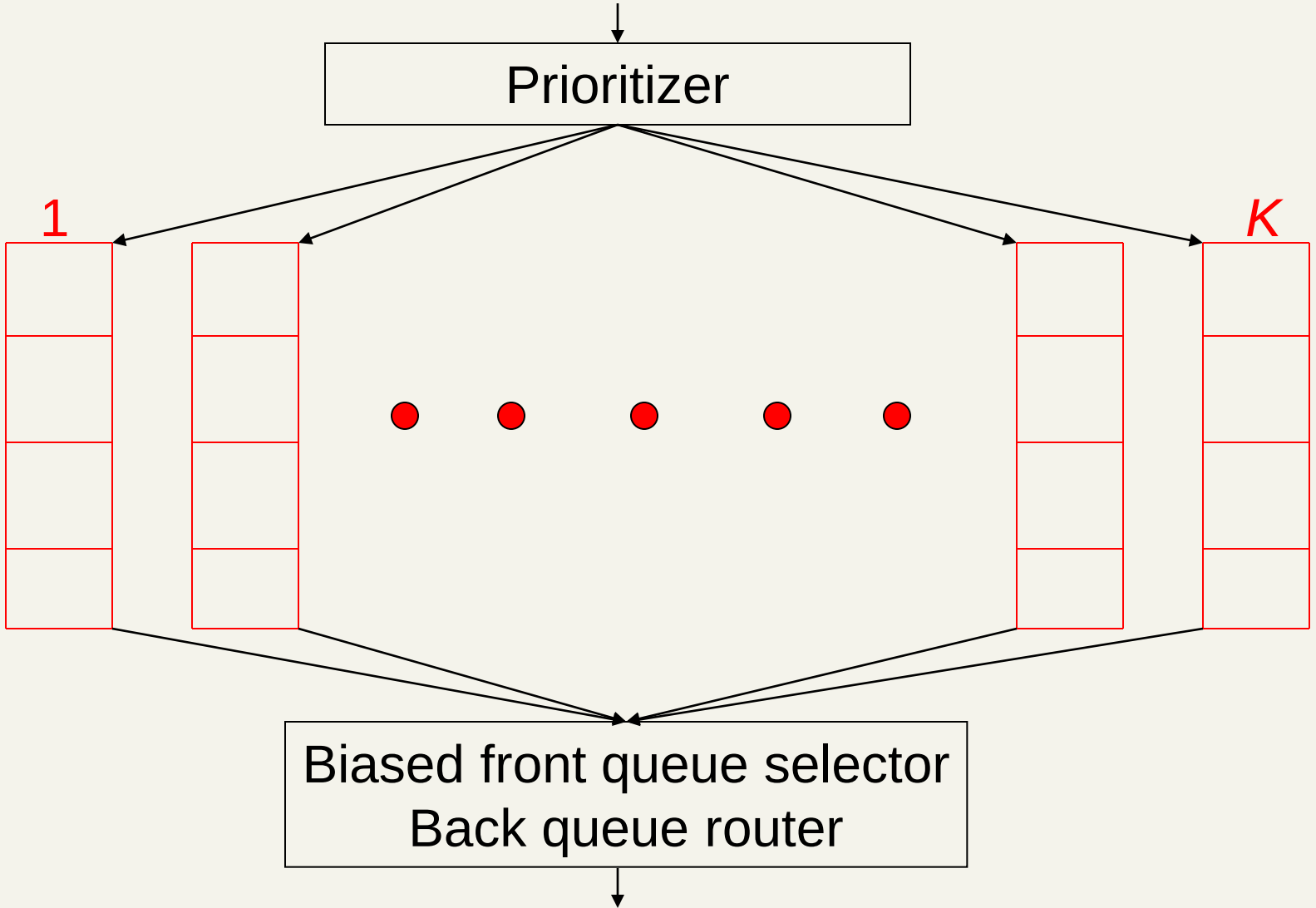
URL frontier: Mercator scheme



Mercator URL frontier

- URL’s flow in from the top into the frontier
- Front queues manage prioritization
- Back queues enforce politeness
- Each queue is FIFO

Front queues



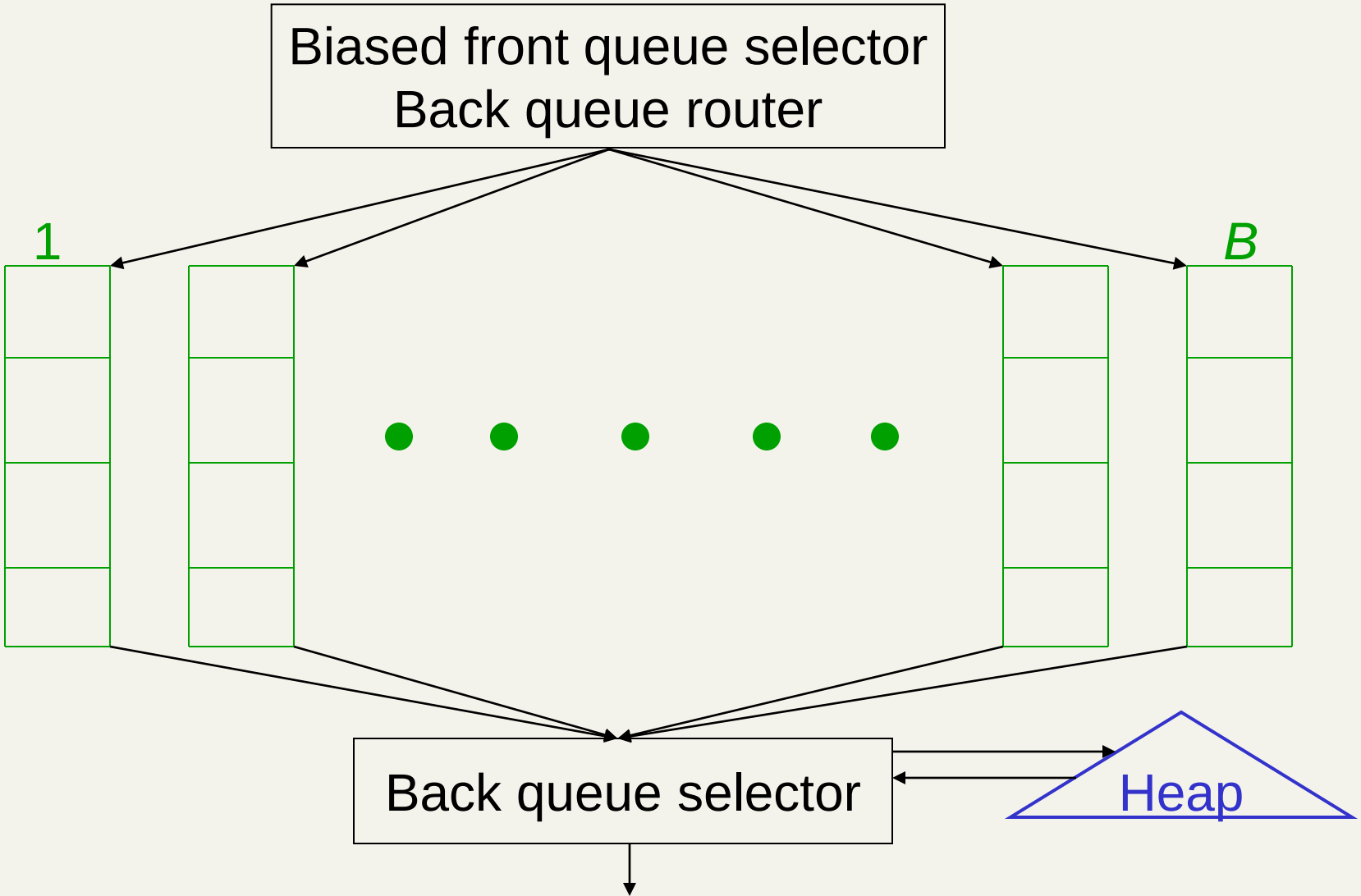
Front queues

- Prioritizer assigns to URL an integer priority 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”)

Biased front queue selector

- When a back queue requests a URL (in a sequence to be described): 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

Back queues


Biased front queue selector
Back queue router
1
B
Back queue selector
Heap

Back queue invariants

- 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

- A crawler thread seeking a URL to crawl:
- Extracts the root of the heap
- Fetches URL at head of corresponding back queue *q* (look up from table)
- 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*
- When *q* is non-empty, create heap entry for it

Number of back queues *B*

- Keep all threads busy while respecting politeness
- Mercator recommendation: three times as many back queues as crawler threads

Connectivity servers

Connectivity Server
[*CS1:* Bhar98b, *CS2 & 3:* Rand01]

- 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 out-links, URL to in-links

- Applications
  - Crawl control
  - Web graph analysis
    - Connectivity, crawl optimization
  - Link analysis

More optimized techniques

- Boldi and Vigna
  - http://www2004.org/proceedings/docs/1p595.pdf
- Webgraph – set of algorithms and a java implementation
- Fundamental goal – maintain node adjacency lists in memory
  - For this, compressing the adjacency lists is the critical component

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
- Naively, this demands 64 bits to represent each hyperlink

Adjaceny list compression

- Properties exploited in compression:
  - Similarity (between lists)
  - Locality (many links from a page go to “nearby” pages)
  - Use gap encodings in sorted lists
  - Distribution of gap values

Storage

- Boldi/Vigna get down to an average of ~3 bits/link
  - (URL to URL edge)
  - For a 118M node web graph
- How?

Main idea of Boldi/Vigna

- Consider lexicographically ordered list of all URLs, e.g.,
  - www.stanford.edu/alchemy
  - www.stanford.edu/biology
  - www.stanford.edu/biology/plant
  - www.stanford.edu/biology/plant/copyright
  - www.stanford.edu/biology/plant/people
  - www.stanford.edu/chemistry
- Main thesis: because of templates, the adjacency list of a node is similar to one of the 7 preceding URLs in the lexicographic ordering
- Express adjacency list in terms of one of these

Resources

- Image IR data sets:
  - Image CLEF: http://www.imageclef.org/2009/photo
  - MIR Flickr: http://press.liacs.nl/mirflickr/