# SPAM
## (SEARCH ENGINE OPTIMIZATION)

# The trouble with paid search ads …

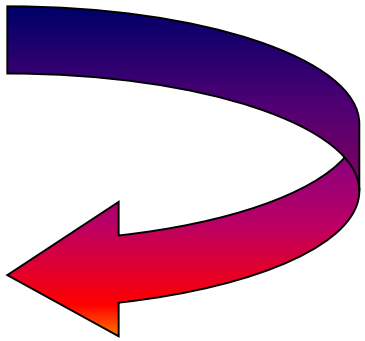
- It costs money. What's the alternative?
- *Search Engine Optimization:*
  - "Tuning" your web page to rank highly in the algorithmic search results for select keywords
  - Alternative to paying for placement
  - Thus, intrinsically a marketing function
- Performed by companies, webmasters and consultants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

# Search engine optimization (Spam)

- Motives
  - Commercial, political, religious, lobbying
  - 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

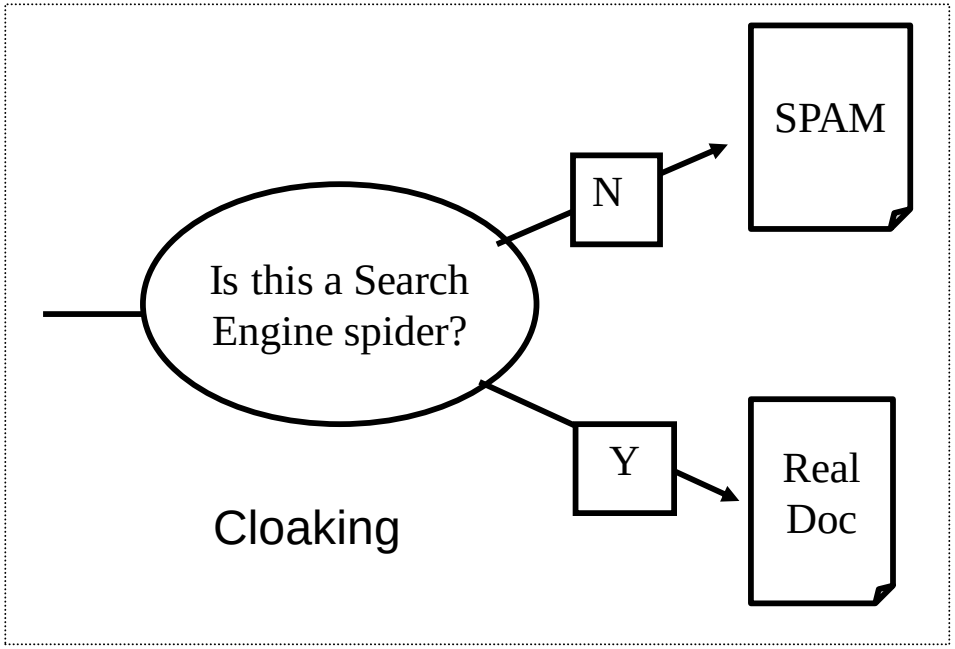
# Variants of keyword stuffing

- 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
  - https://www.bing.com/toolbox/webmaster/
  - 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

# What is the size of the web ?

- Issues
  - The web is really infinite
    - Dynamic content, e.g., calendars
    - 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.

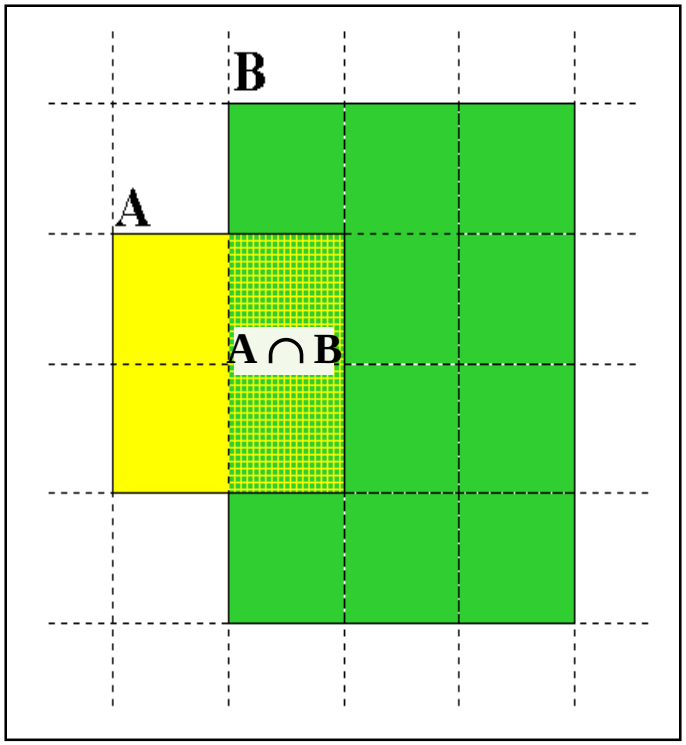
# What can we attempt to measure?

- 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 anchor text.
    - Document restriction: All engines restrict what is indexed (first *n* words, only relevant words, etc.)

# New definition?

- The statically indexable web is whatever search engines index.
  - IQ is whatever the IQ tests measure.
- 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.

Not an English dictionary

# 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 (because it covers all the human searches)
- 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

# Quiz: Queries from NEC 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*

What's the problem with these queries?

# 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

# Random IP addresses

- HTTP requests to random IP addresses
  - Ignored: empty or authorization required or excluded
  - [Lawr99] Estimated 2.8 million IP addresses running crawlable web servers (16 million total) from observing 2500 servers.
  - OCLC using IP sampling found 8.7 M hosts in 2001
    - Netcraft [Netc02] accessed 37.2 million hosts in July 2002
- [Lawr99] exhaustively crawled 2500 servers and extrapolated
  - Estimated size of the web to be 800 million pages
  - Estimated use of metadata descriptors:
    - Meta tags (keywords, description) in 34% of home pages, Dublin core metadata in 0.3%

# 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.
    - E.g.: employee home 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 search engine

# 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

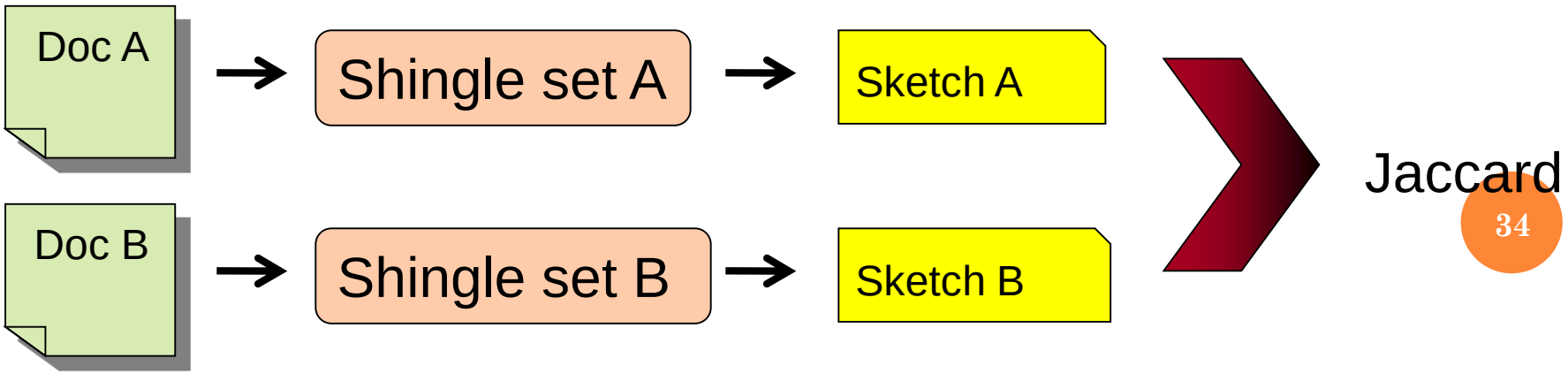
    rose_is_a_rose

    is_a_rose_is

    a_rose_is_a
- Similarity Measure between two docs (= sets of shingles)
  - Jaccard coefficient: Size_of_Intersection / Size_of_Union

# Shingles + Set Intersection

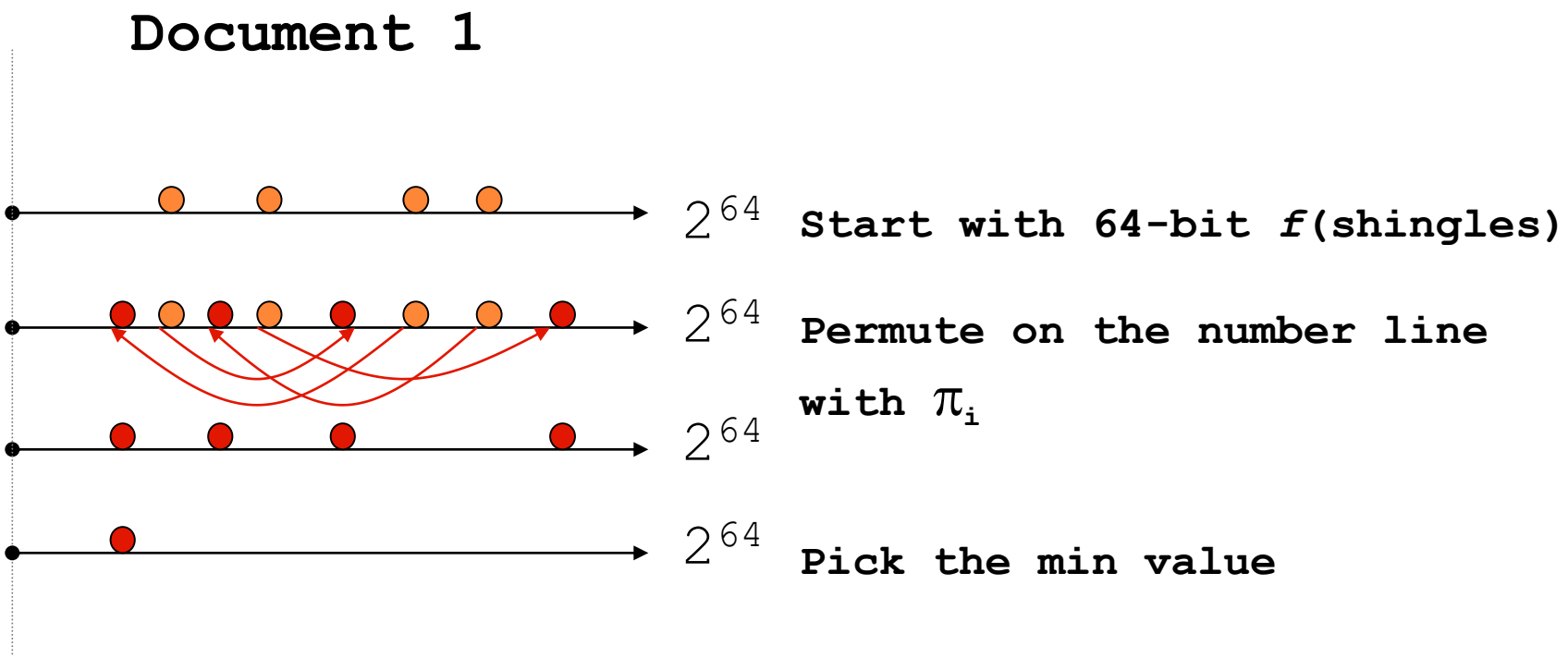
- Computing exact set intersection of shingles between all pairs of documents is expensive/intractable
  - Approximate using a cleverly chosen subset of shingles from each (a *sketch*)
- Estimate (size_of_intersection / size_of_union) based on a short sketch



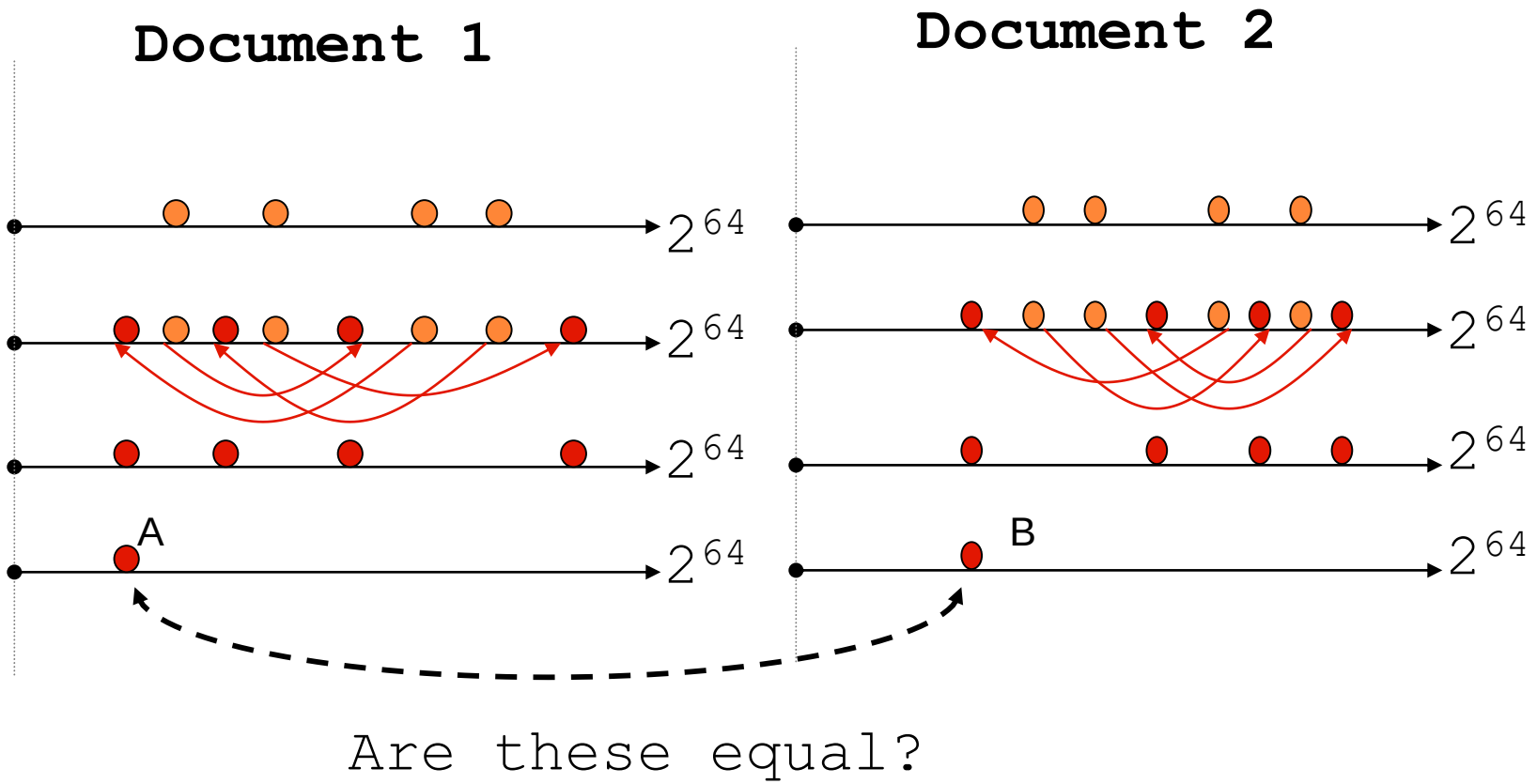
# Sketch of a document

- Create a "sketch vector" (of size ~200) for each document
  - Documents that share $\geq t$ (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 $0..2^m$-1 (e.g., f = fingerprinting)
    - Let $\pi_i$ be a *random permutation* on $0..2^m$-1
    - Pick MIN $\{\pi_i(f(s))\}$ over all shingles $s$ in $D$

# Computing Sketch[i] for Doc1

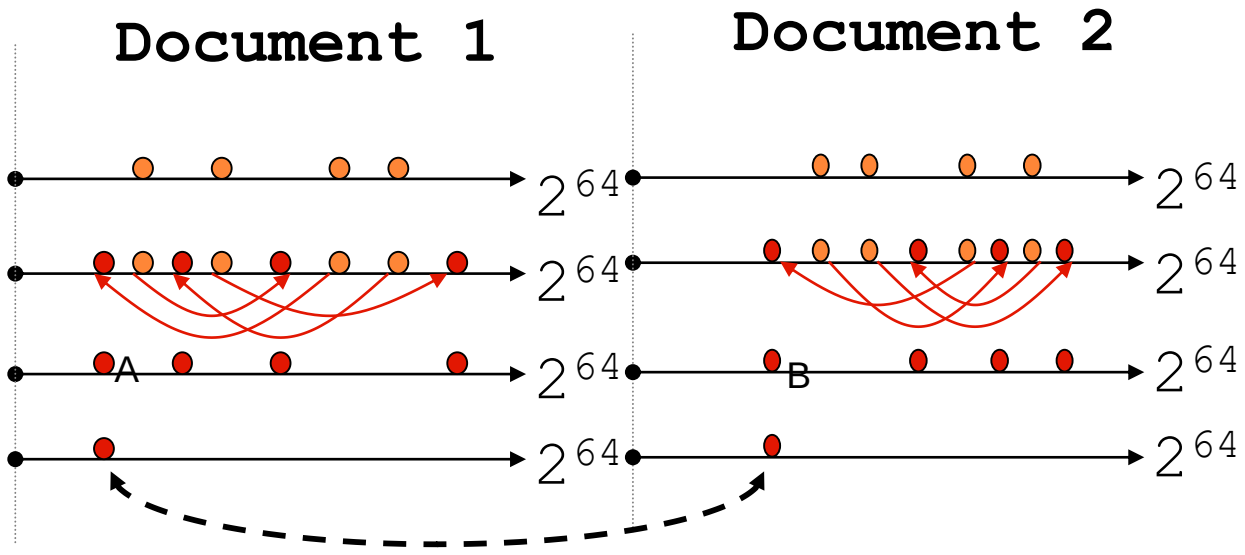


# Test if Doc1.Sketch[i] = Doc2.Sketch[i]




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

# HOWEVER…




Why?

Theorem:

Jaccard (D1, D2) = Prob(A = B)

# 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

# Quiz: Calculate Jaccard

| C1 | C2 | C3 |
|---|---|---|
| 1 | 0 | 1 |
| 1 | 0 | 0 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

- From the perspective of jaccard, which one is more similar to C1, between C2 and C3? Why?

# 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

$$P\left(h(C_i) = h(C_j)\right) = \text{Jaccard}\left(C_i, C_j\right)$$

- Why?
  - 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(sketch($C_i$),sketch($C_j$))] = 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**

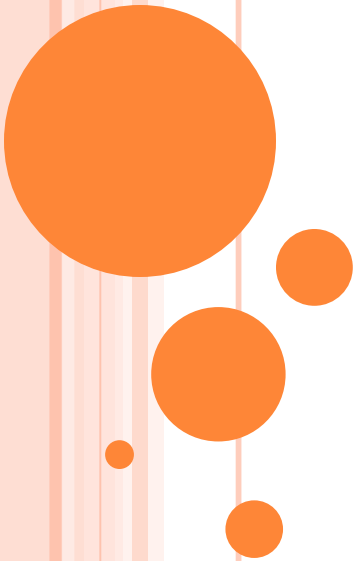
| | 1-2 | 1-3 | 2-3 |
|---|---|---|---|
| **Col-Col** | 0.00 | 0.50 | 0.25 |
| **Sig-Sig** | 0.00 | 0.67 | 0.00 |

# 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)

# More resources

- IIR Chapter 19

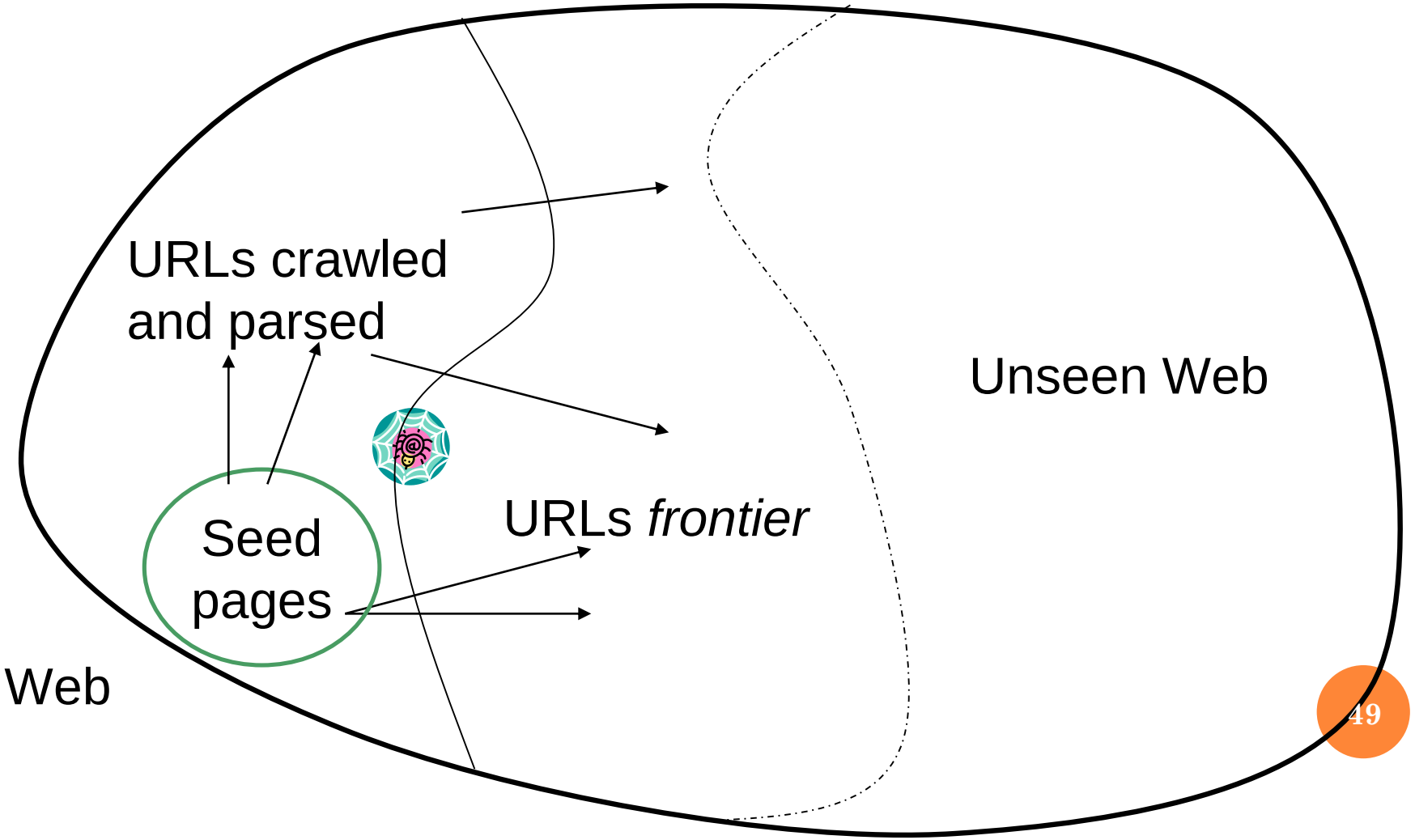


# Crawling and Web Indexes

# Basic crawler operation

- Begin with known "seed" URLs
- 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
- Malicious pages
  - Spam pages
  - Spider traps – incl 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 – don't hit a server too often

# What any crawler *must* do

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

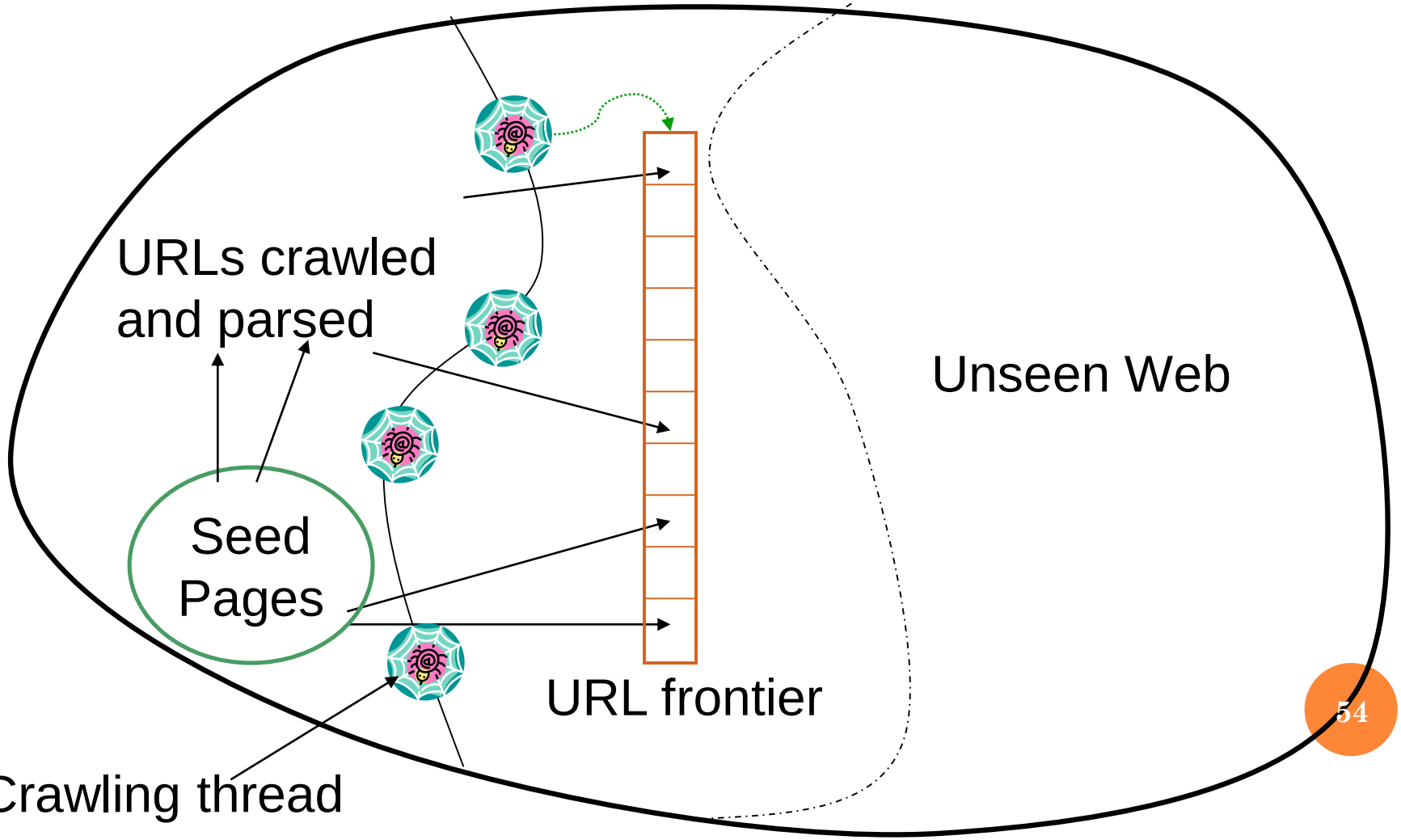
# What any crawler *should* do

- 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 server, create a file `/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:
```

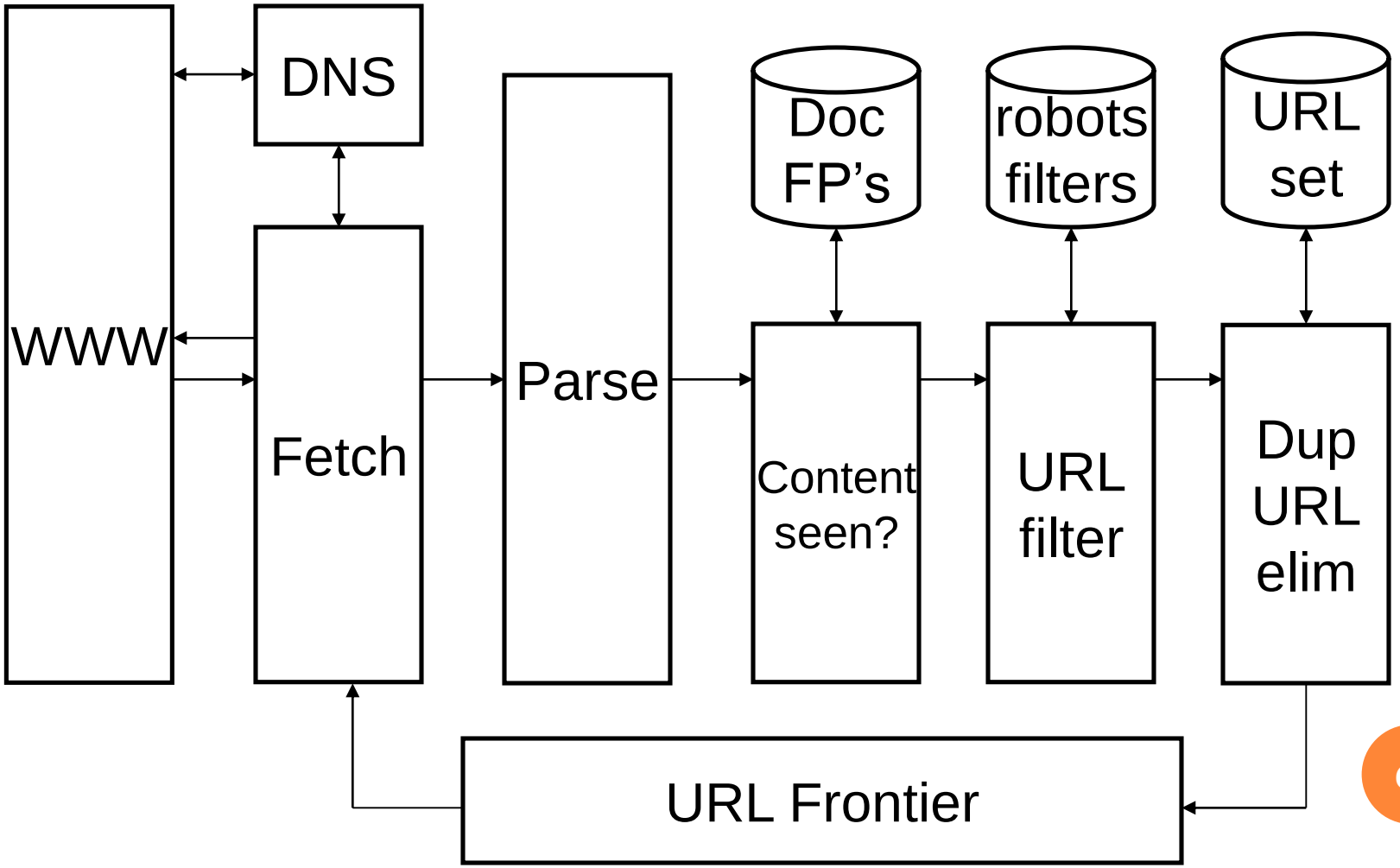
# Processing steps in crawling

- 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., http://en.wikipedia.org/wiki/Main_Page has 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

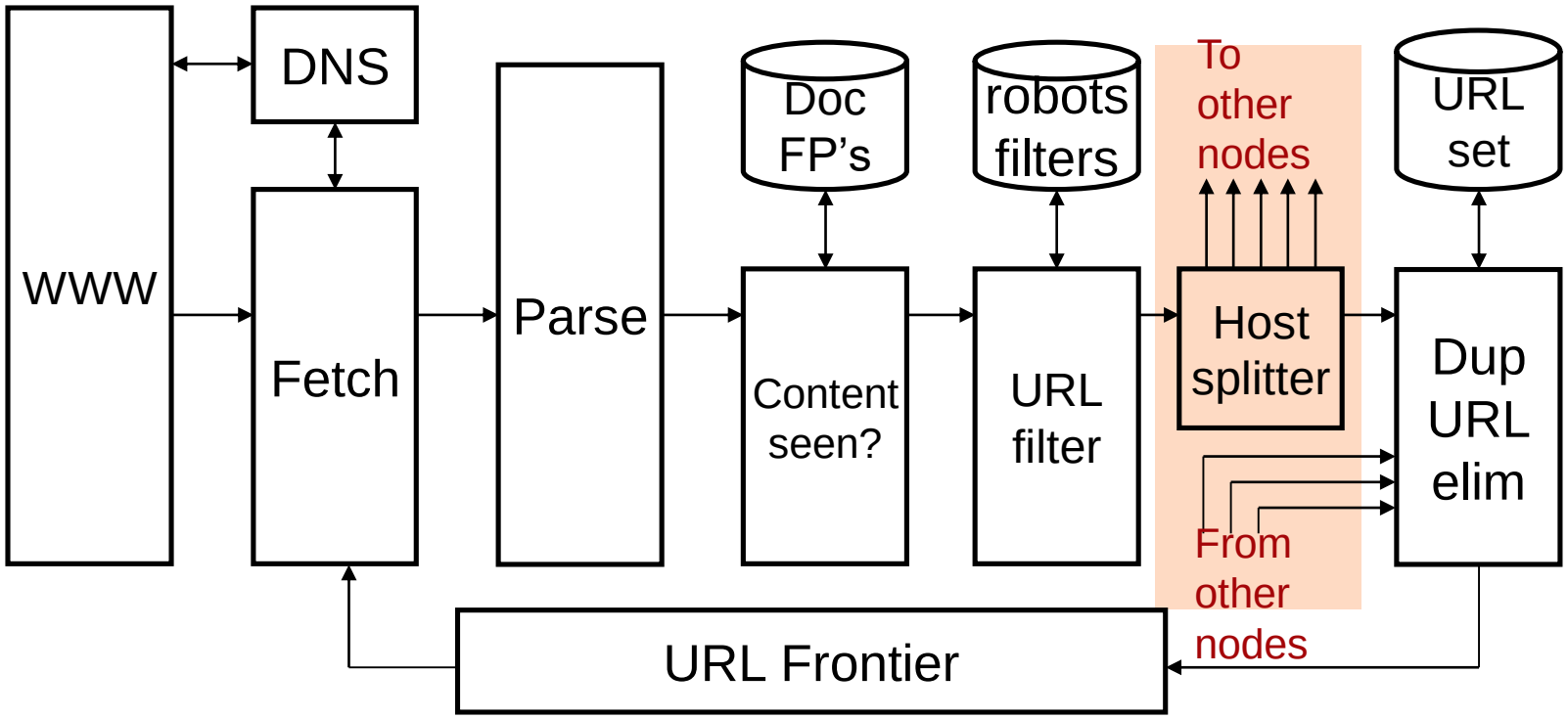
- For a continuous crawl – see details of frontier implementation

# 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 and share URLs?

# Communication between nodes

- Output of the URL filter at each node is sent to the Dup URL Eliminator of the appropriate node



# 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

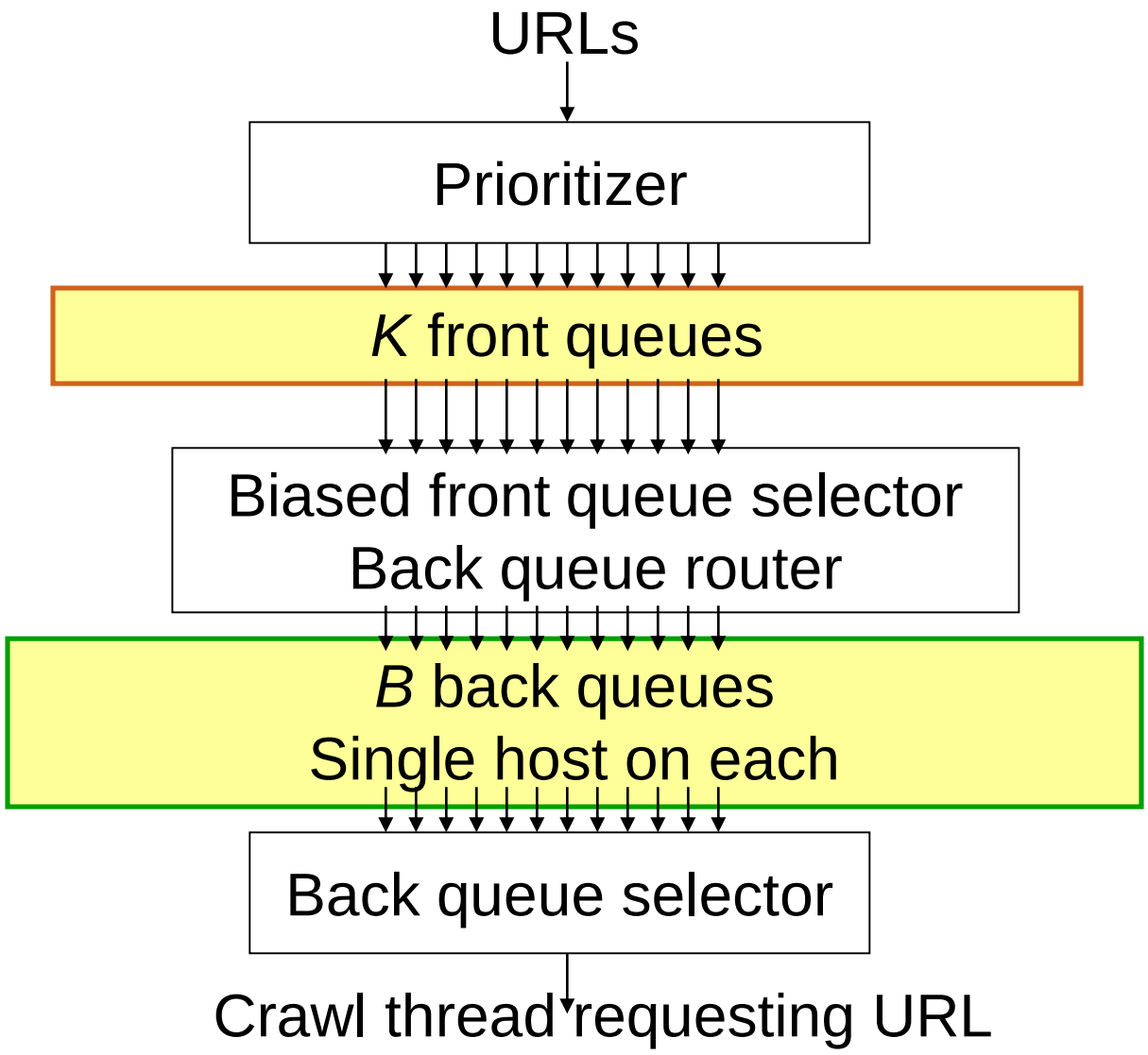
These goals may conflict each other.

(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

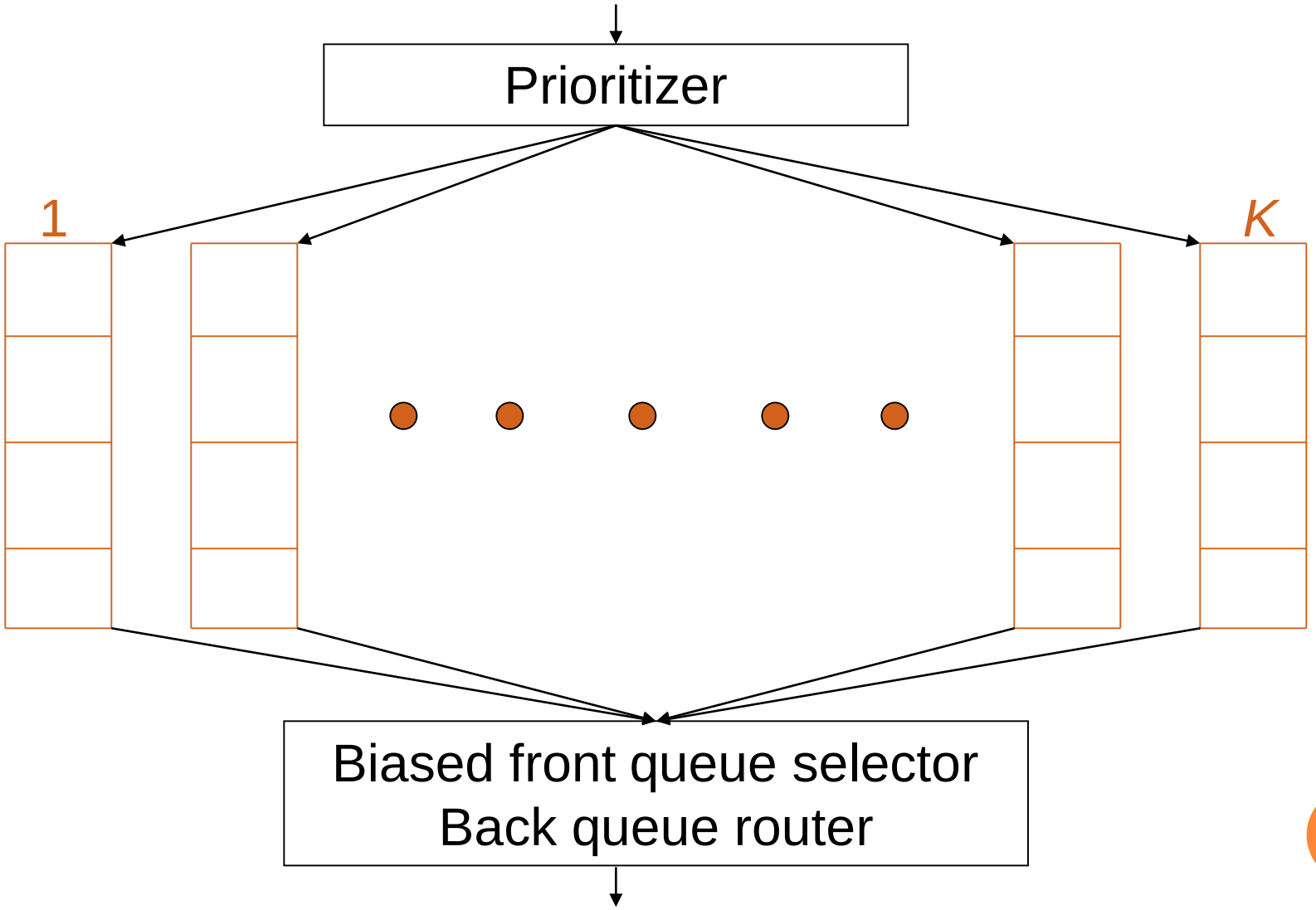
# URL FRONTIER: MERCATOR SCHEME



# Mercator URL frontier

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

# Front queues



# Front queues

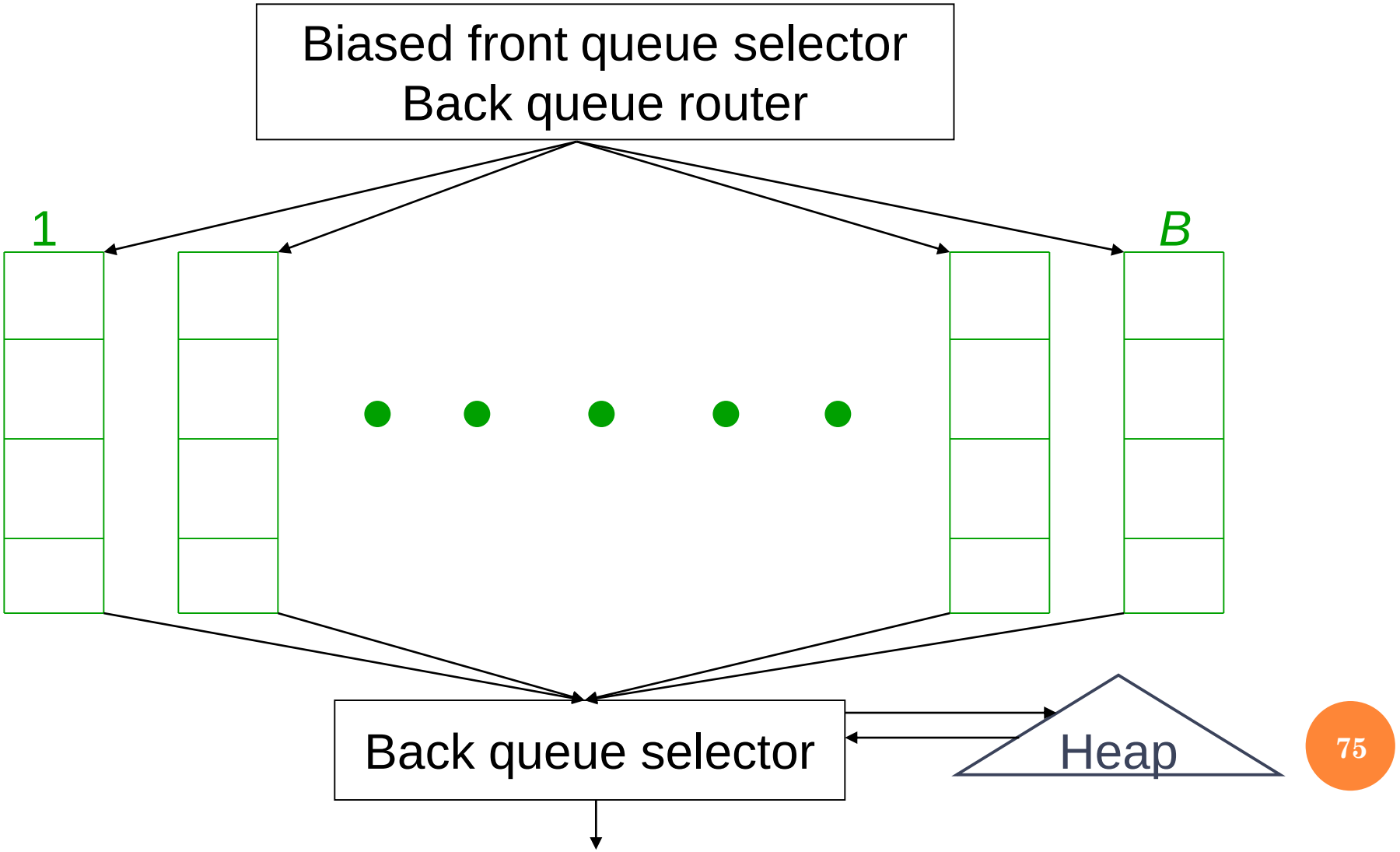
- Prioritizer assigns to URL an integer priority between 1 and *K*
  - Appends URL to corresponding queue
- Heuristics for assigning priority
  - Refresh rate sampled from previous crawls
  - Application-specific (e.g., "crawl news sites more often")

# 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 that back queue and pull another URL from front queues, repeat
  - Else add $v$ to $q$
- When $q$ is non-empty, create heap entry for it

# Quiz: Front Queues

- What's the purpose of the front queues?
  a) Parallelism
  b) Politeness
  c) Prioritization
  d) Throughput

# Number of back queues *B*

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

# Resources

- IIR Chapter 20
- Mercator: A scalable, extensible web crawler (Heydon et al. 1999)
- A standard for robot exclusion