universität koblenz
weiter:denken

Web Information Retrieval
Web Crawling
(SOSE 2023, 07.06.2023)

Frank Hopfgartner, Stefania Zourlidou
Institute for Web Science and Technologies

Credit for these slides

These slides have been adapted from

- Web IR (Zeyd Boukhers-WeST, SOSE 2020)

Recapitulation

universität koblenz
weiter:denken

- Classical IR vs. Web IR
- Web IR, Web search basics
  - Ads
  - Spams
  - Duplicates
  - User Needs
  - Web Graph (anchor text)
  - Indexing

Objectives of this lecture

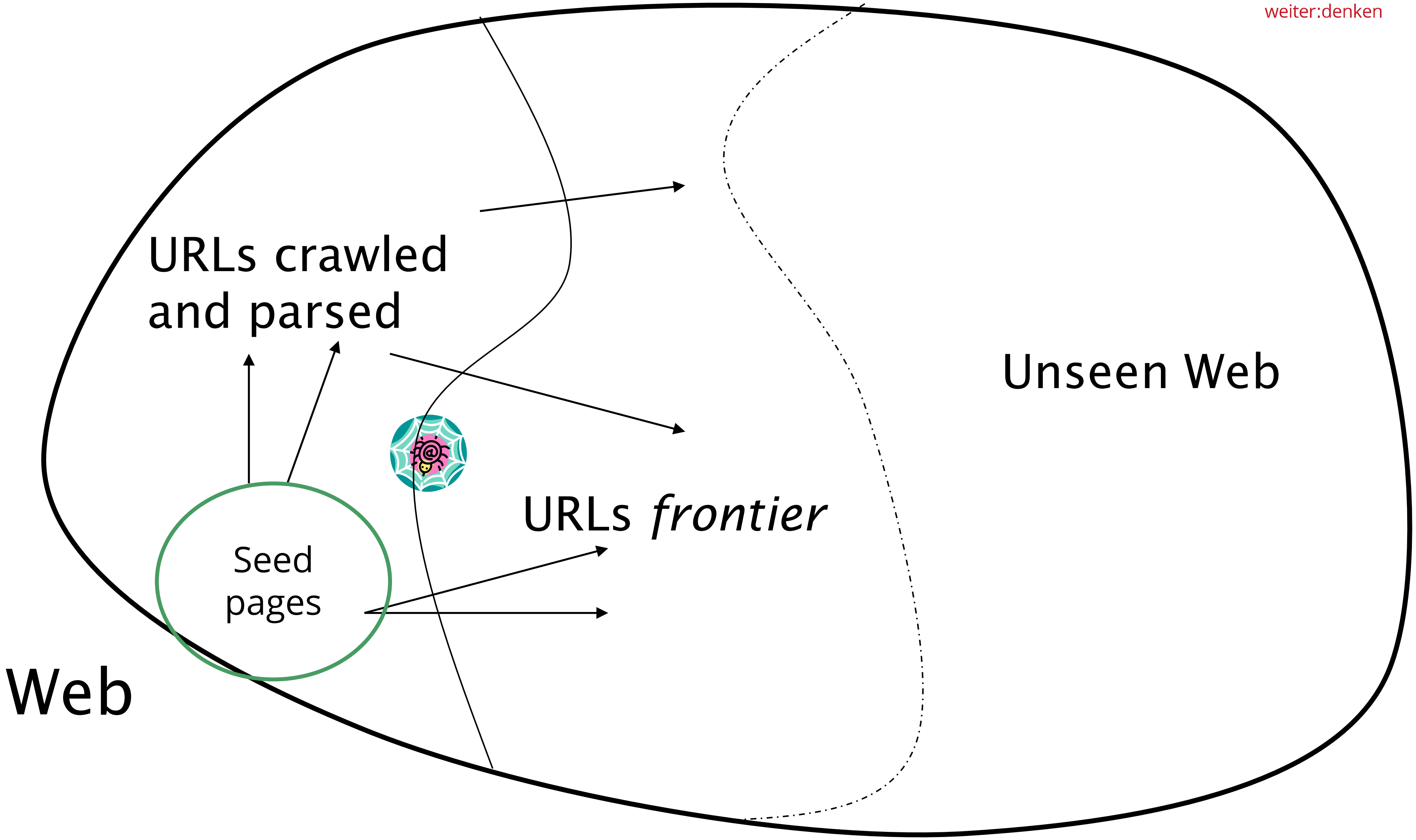
- Crawler
  - what is it?
    - features a crawler *must* provide
    - features a crawler *should* provide
  - crawler architecture
    - robots exclusion protocol
    - url normalization
  - why distributing the crawler
  - the URL frontier

1. Web crawling: what is it?

Basic crawler operation

1. Begin with a known “seed” URLs
2. Fetch and parse the URLs
   a) Extract URLs they point to
   b) Place the extracted URLs on the queue
3. Go to 2

Crawling picture



Simple picture – complications

- Web crawling is not feasible with one machine
  - All of the above steps are 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

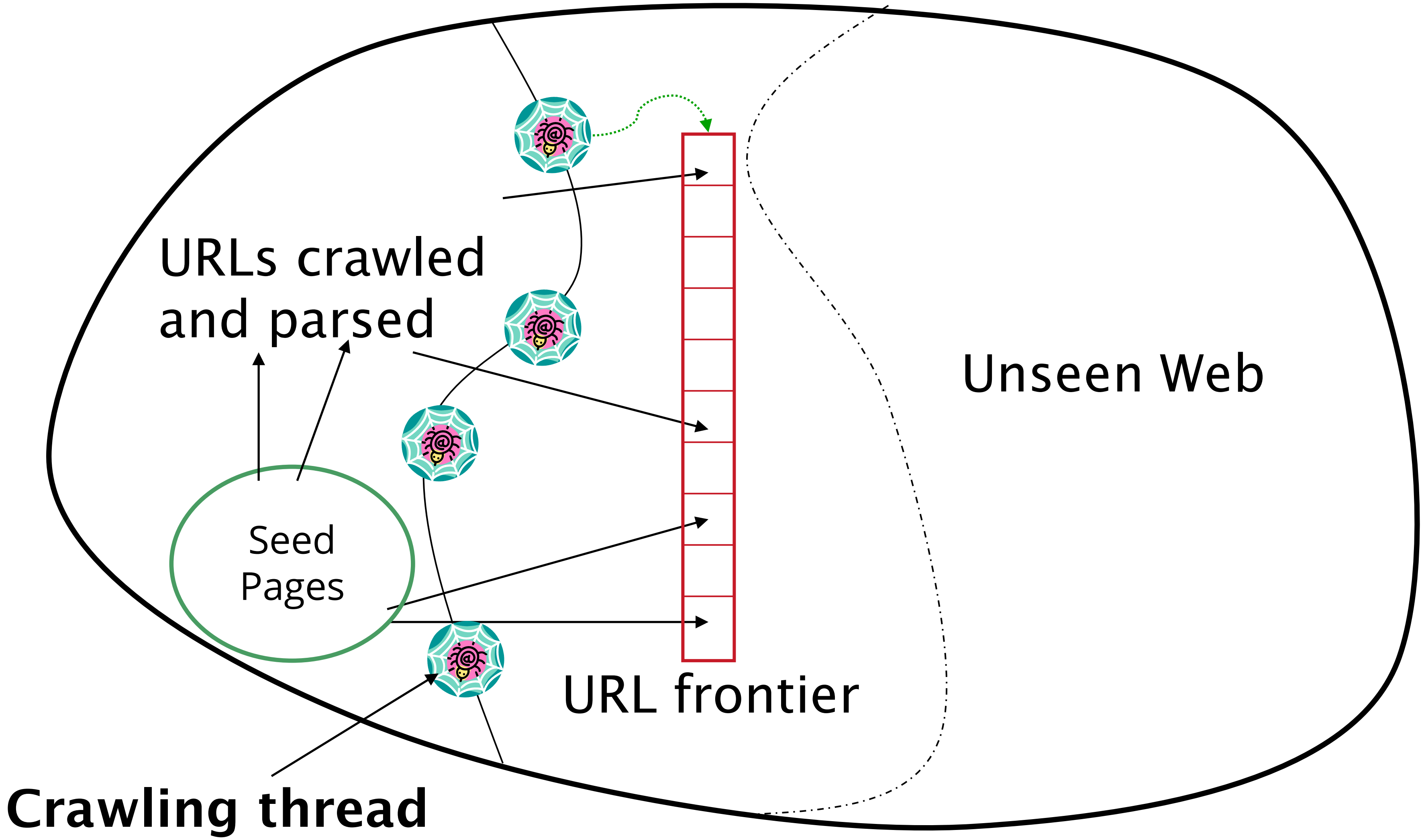
What a crawler *must* do

- Be **polite**: respect implicit and explicit politeness considerations
  - Only crawl allowed pages
  - Respect *robots.txt*
- Be **robust**: be immune to spider traps and other malicious behavior from web servers

What a 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
- 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

universität koblenz
weiter:denken

- Pick a URL from the frontier

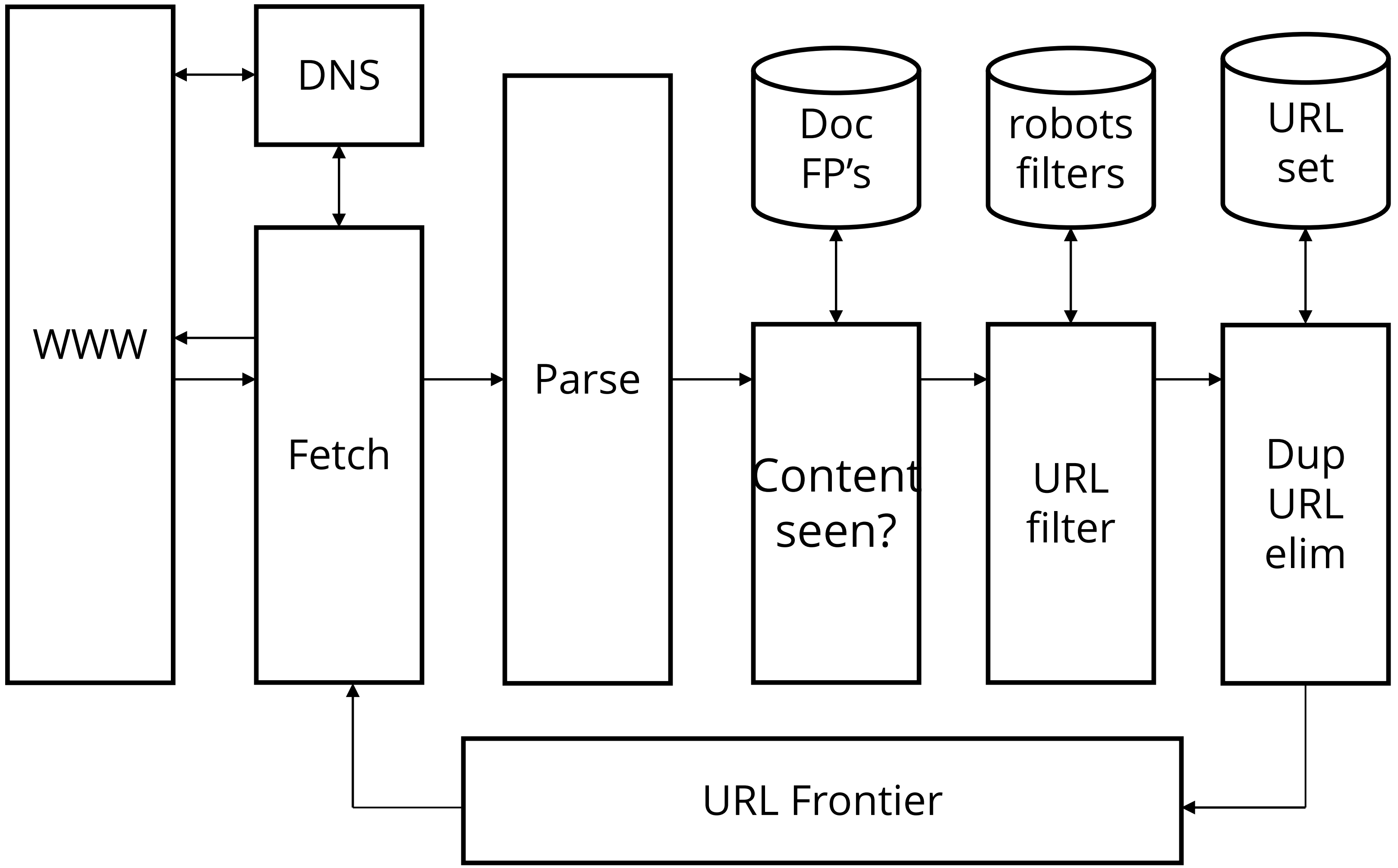
Which one?

- Fetch the document (web page) at the URL
- Parse the URL
  - Extract the text and set of links from the fetched web page
- 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)

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

2. Basic crawl architecture

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 being fetched is already in the index, do not further process it
- This is verified using document fingerprints or shingles

Filters and robots.txt

- <u>Filters</u> – 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

3. Distributing the crawler

universität koblenz
weiter:denken

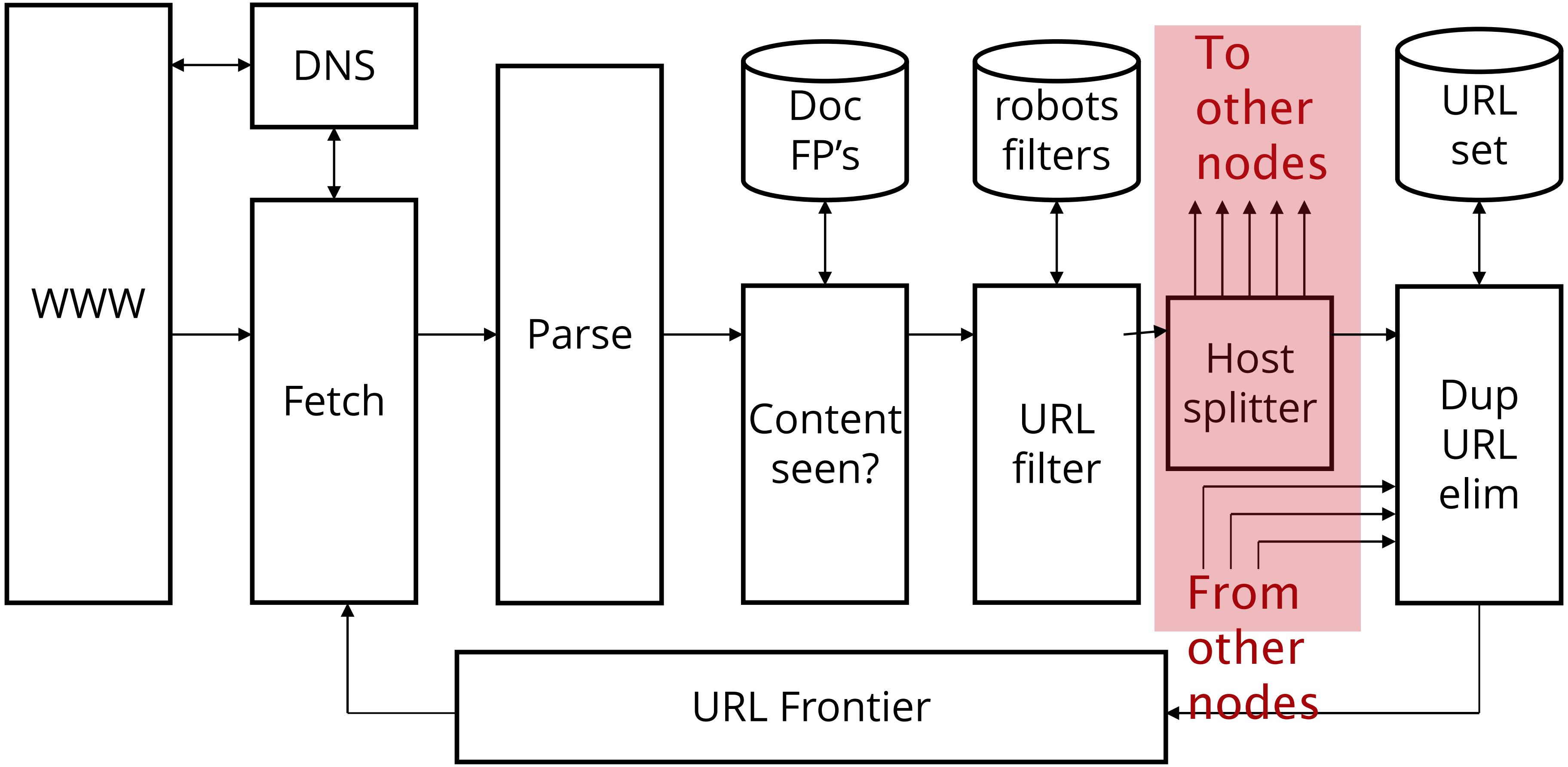
Distributing the crawler

- 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?

Google data centers (wazfaring. com)

Communication between nodes

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



4. The URL frontier

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

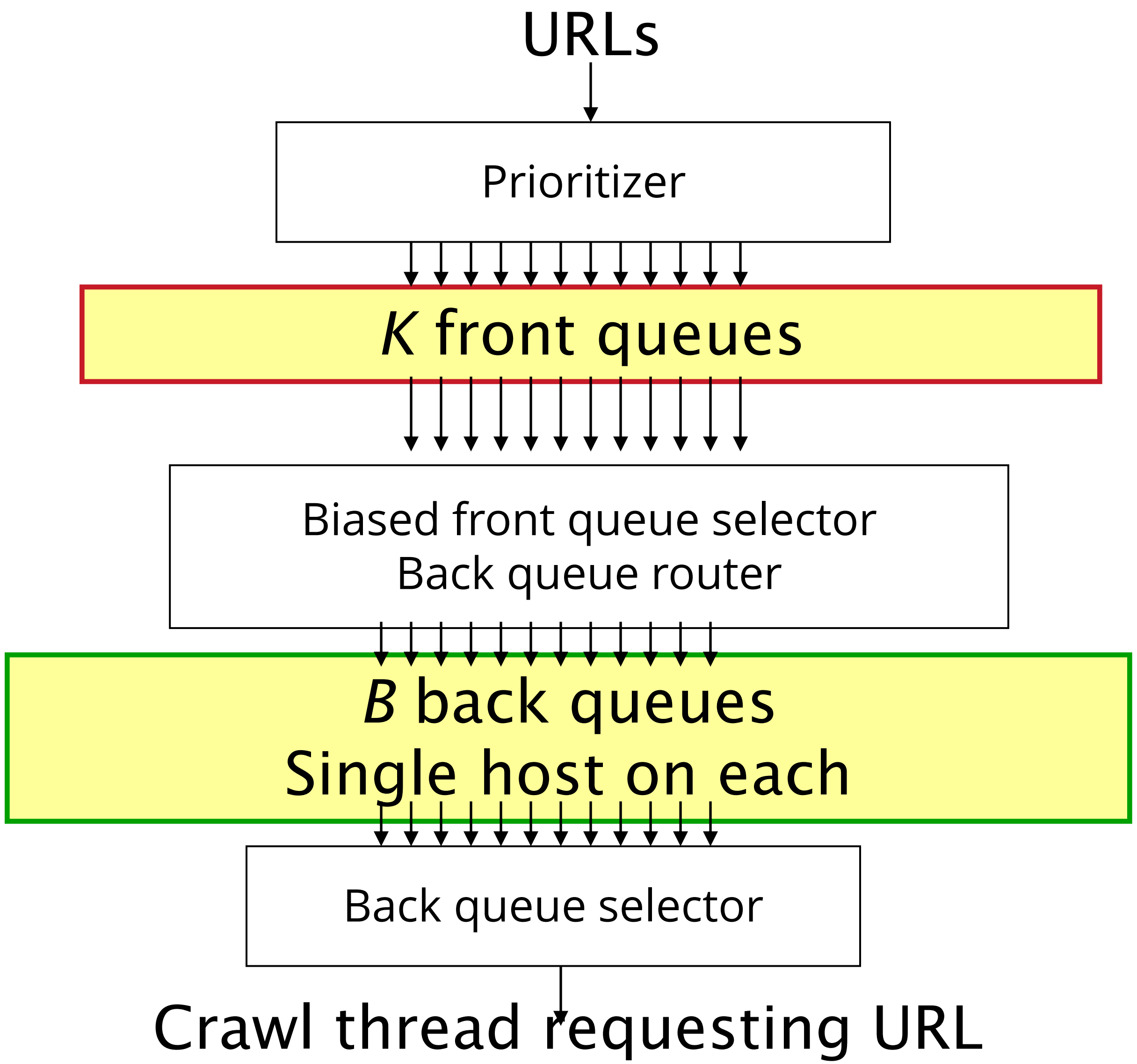
These goals may conflict each other

- 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

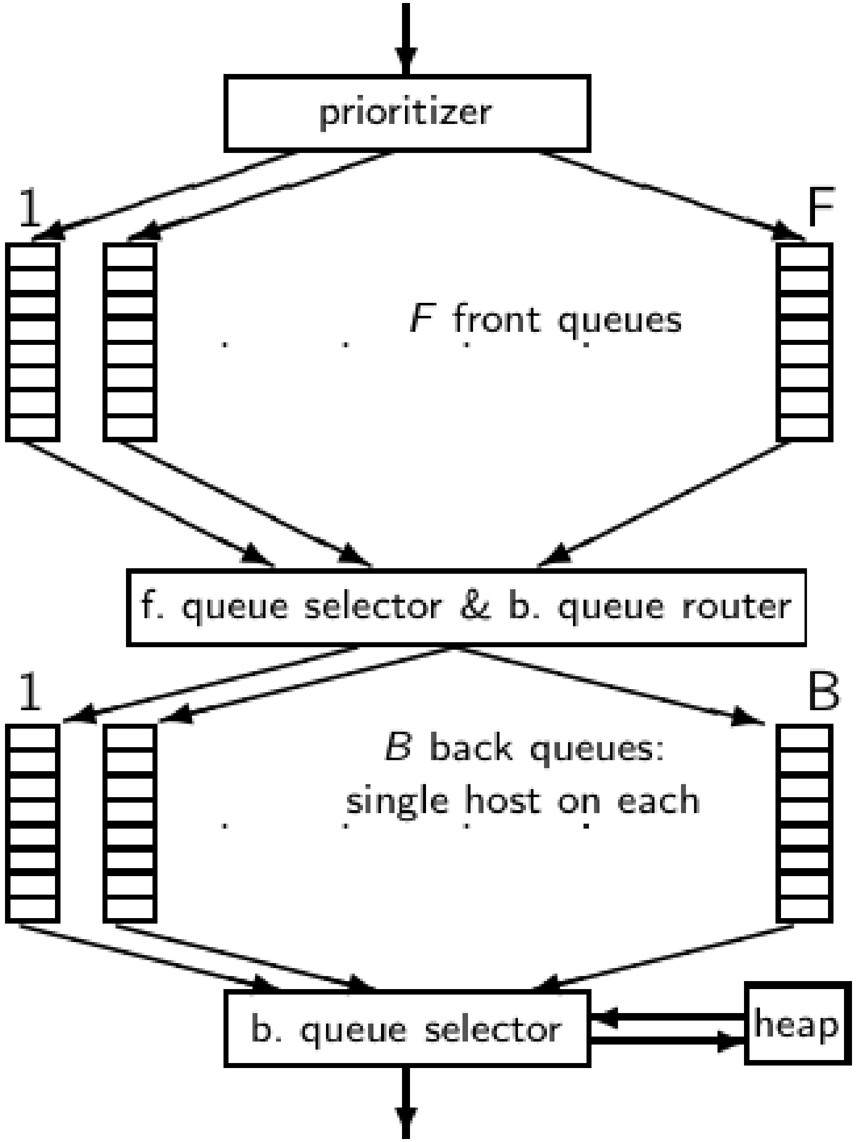
Politeness – challenges

- 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

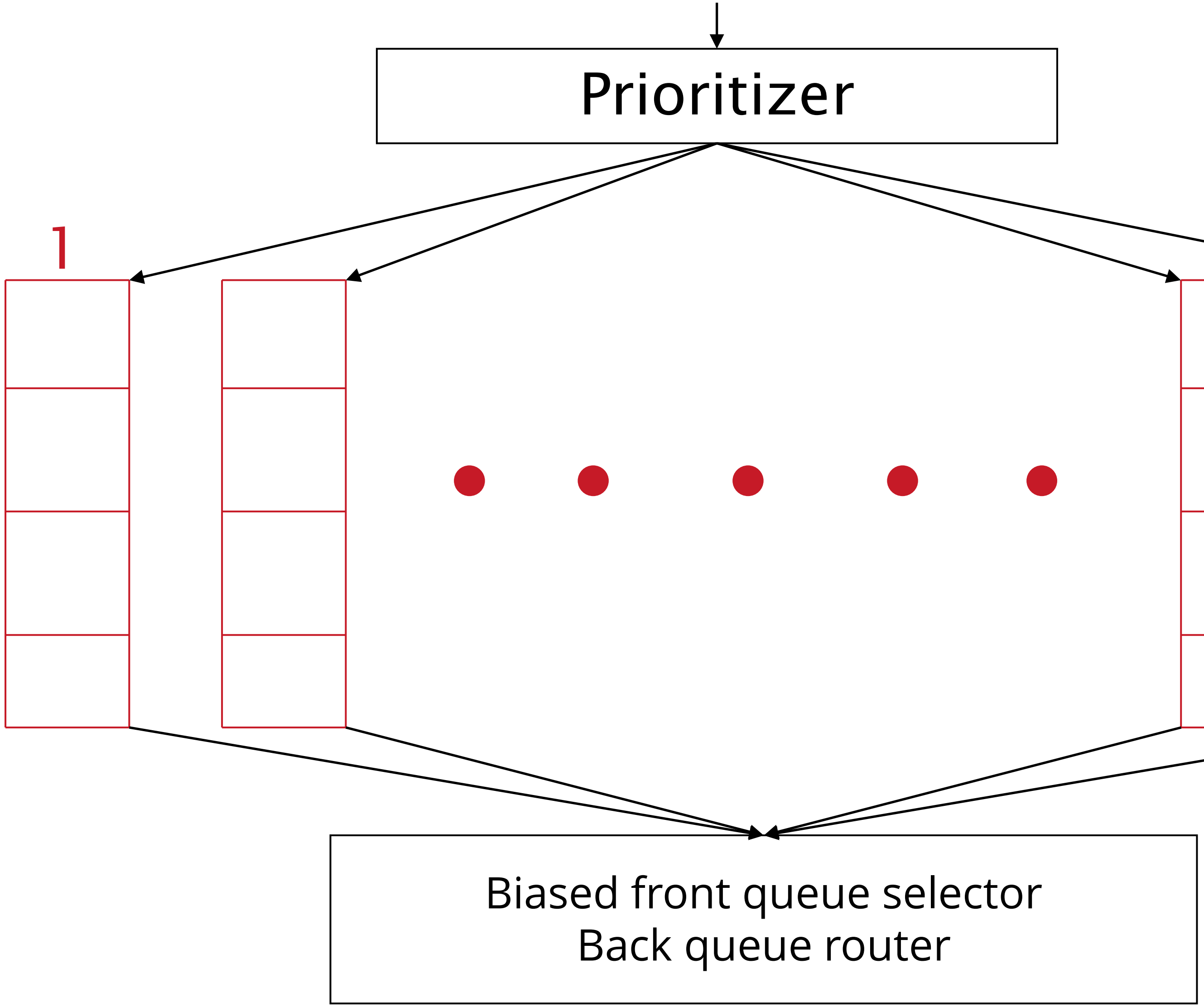


Mercator URL frontier



- 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

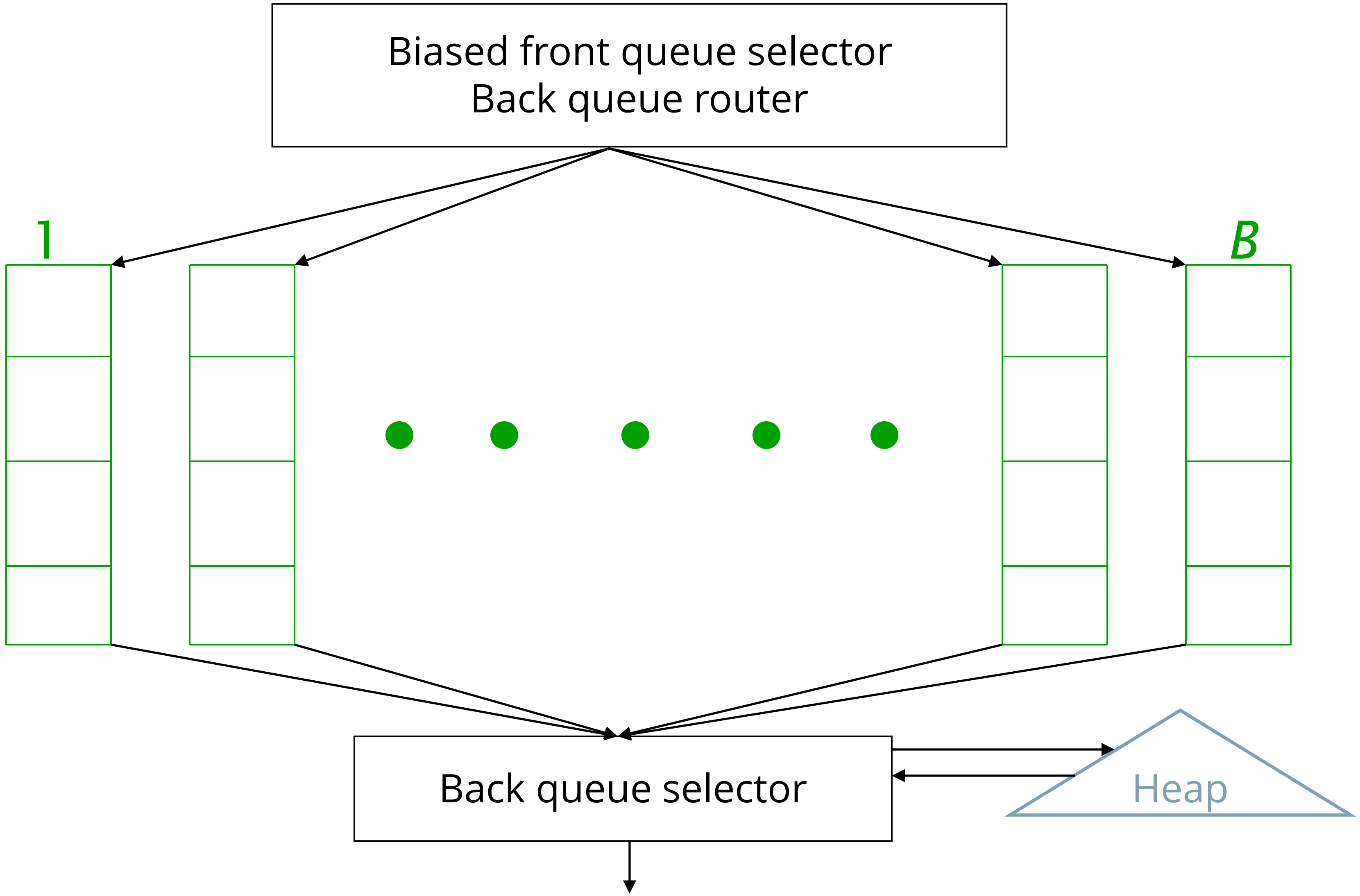
universität koblenz
weiter:denken

- 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

universität koblenz
weiter:denken

- 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 is 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

universität koblenz
weiter:denken

5. Summary

Summary

- Web crawling
  - what is it?
  - basic crawler architecture
  - distributing the basic crawler architecture
  - the URL frontier

References

[1] https://olat.vcrp.de/auth/RepositoryEntry/4071063853

[2] https://nlp.stanford.edu/IR-book/information-retrieval-book.html
Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008

- ► Chapter 20 (Web crawling and indexes)

[3] Some extra resources

- ► Mercator: A scalable, extensible Web crawler
- ► A Standard for Robot Exclusion