مبانی بازیابی اطلاعات و جستجوی وب

۱۲- Crawling

Basic crawler operation

- Initialize queue with URLs of known seed pages
- Repeat
  - Take URL from queue
  - Fetch and parse page
  - Extract URLs from page
  - Add URLs to queue
- Fundamental assumption: The web is well linked.

What's wrong with the simple crawler

- Scale: we need to distribute.
- We can't index everything: we need to subselect. How?
- Duplicates: need to integrate duplicate detection
- Spam and spider traps: need to integrate spam detection
- Politeness: we need to be "nice" and space out all requests for a site over a longer period (hours, days)
- Freshness: we need to recrawl periodically.
  - Because of the size of the web, we can do frequent recrawls only for a small subset.
  - Again, subselection problem or prioritization

Magnitude of the crawling problem

- To fetch 20,000,000,000 pages in one month . . .
- . . . we need to fetch almost 8000 pages per second!
- Actually: many more since many of the pages we attempt to crawl will be duplicates, unfetchable, spam etc.

What a crawler must do

Be polite

- Don’t hit a site too often
- Only crawl pages you are allowed to crawl: robots.txt

Be robust

- Be immune to spider traps, duplicates, very large pages, very large websites, dynamic pages etc

Robots.txt

- Protocol for giving crawlers ("robots") limited access to a website, originally from 1994
- Examples:
  - User-agent: *

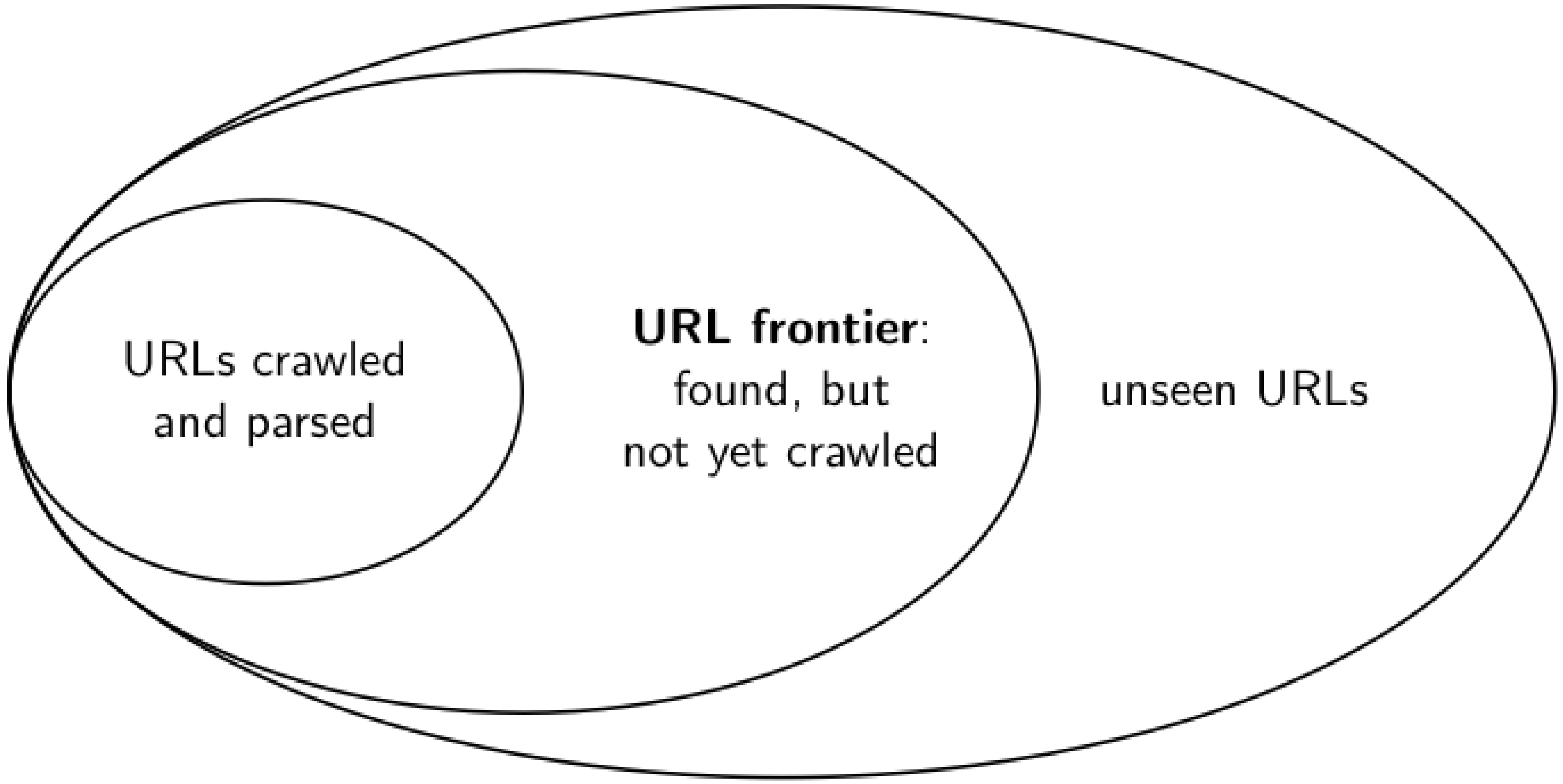
    Disallow: /yoursite/temp/
  - User-agent: searchengine

    Disallow:
- Important: cache the robots.txt file of each site we are crawling

What any crawler should do

- Be capable of distributed operation
- Be scalable: need to be able to increase crawl rate by adding more machines
- Fetch pages of higher quality first
- Continuous operation: get fresh version of already crawled pages

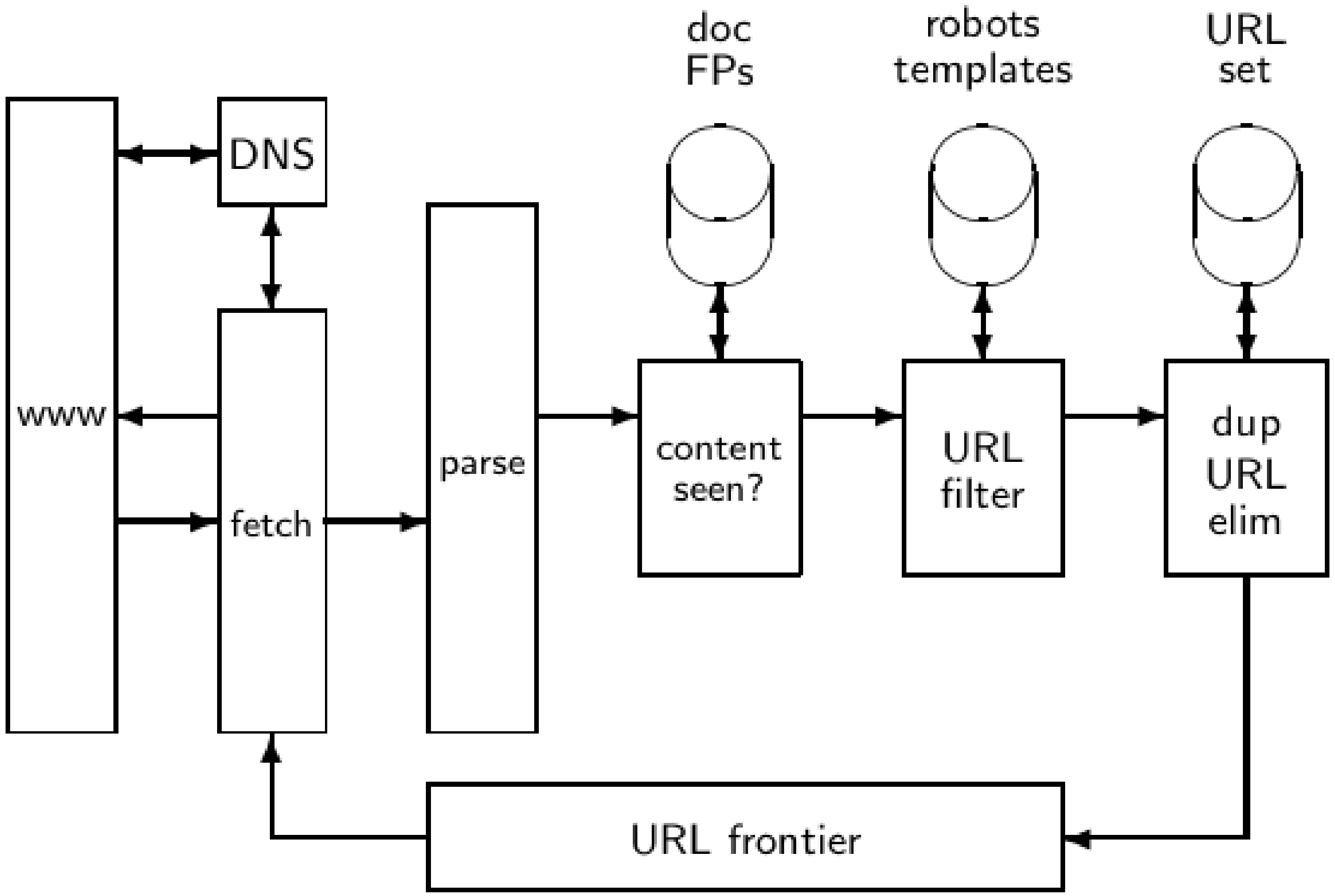
URL frontier



URL frontier

- The URL frontier is the data structure that holds and manages URLs we've seen, but that have not been crawled yet.
- Can include multiple pages from the same host
- Must avoid trying to fetch them all at the same time
- Must keep all crawling threads busy

Basic crawl architecture



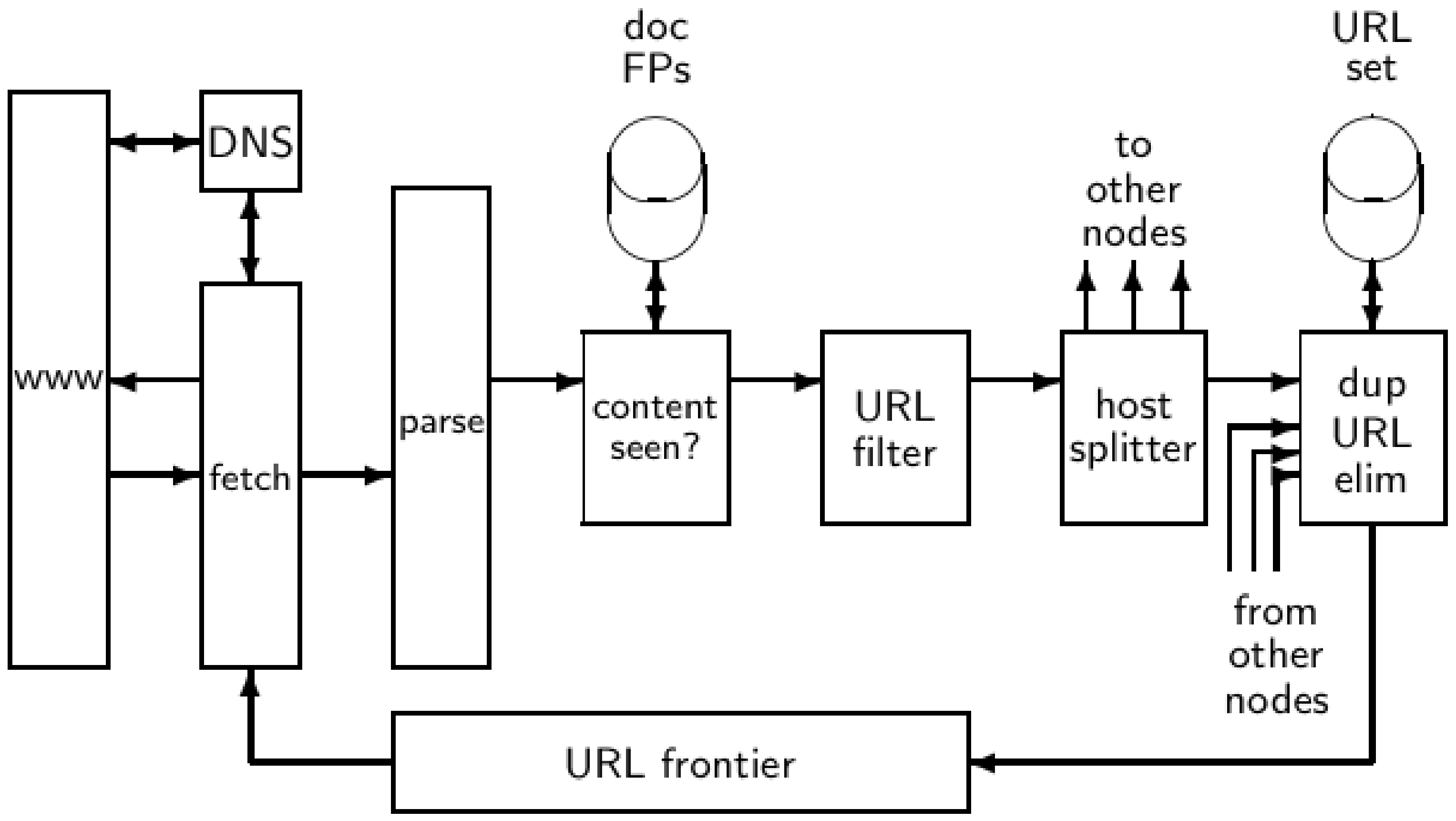
Content seen

- For each page fetched: check if the content is already in the index
- Check this using document fingerprints or shingles
- Skip documents whose content has already been indexed

Distributing the crawler

- Run multiple crawl threads, potentially at different nodes
  - Usually geographically distributed nodes
- Partition hosts being crawled into nodes

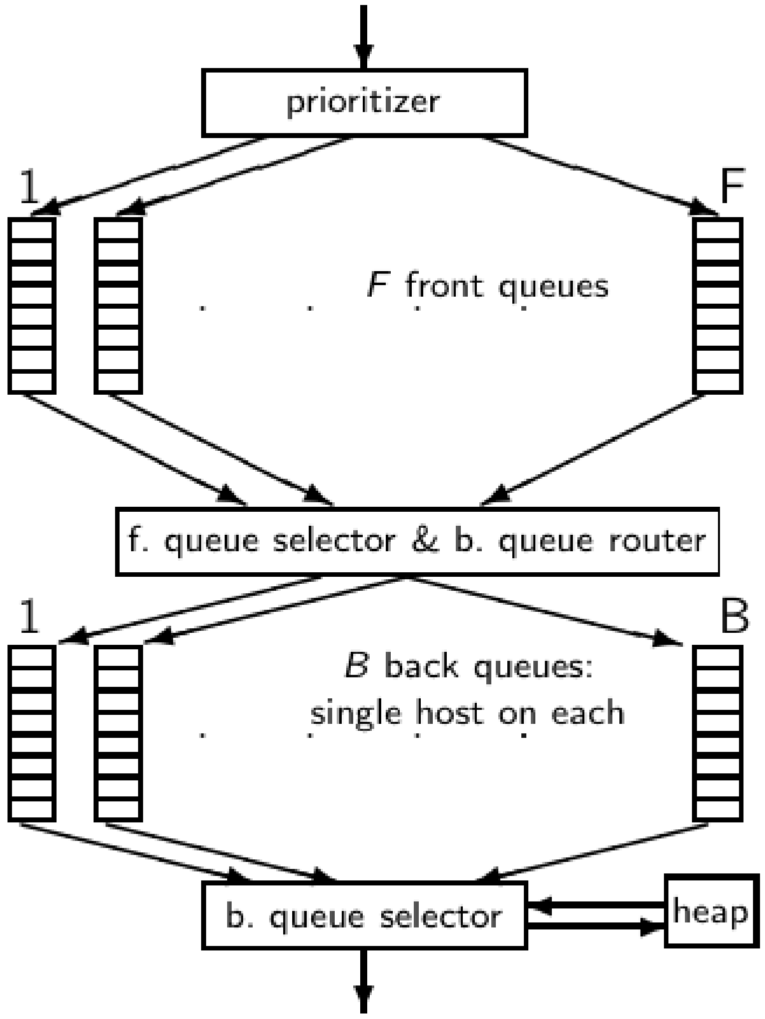
Distributed crawler



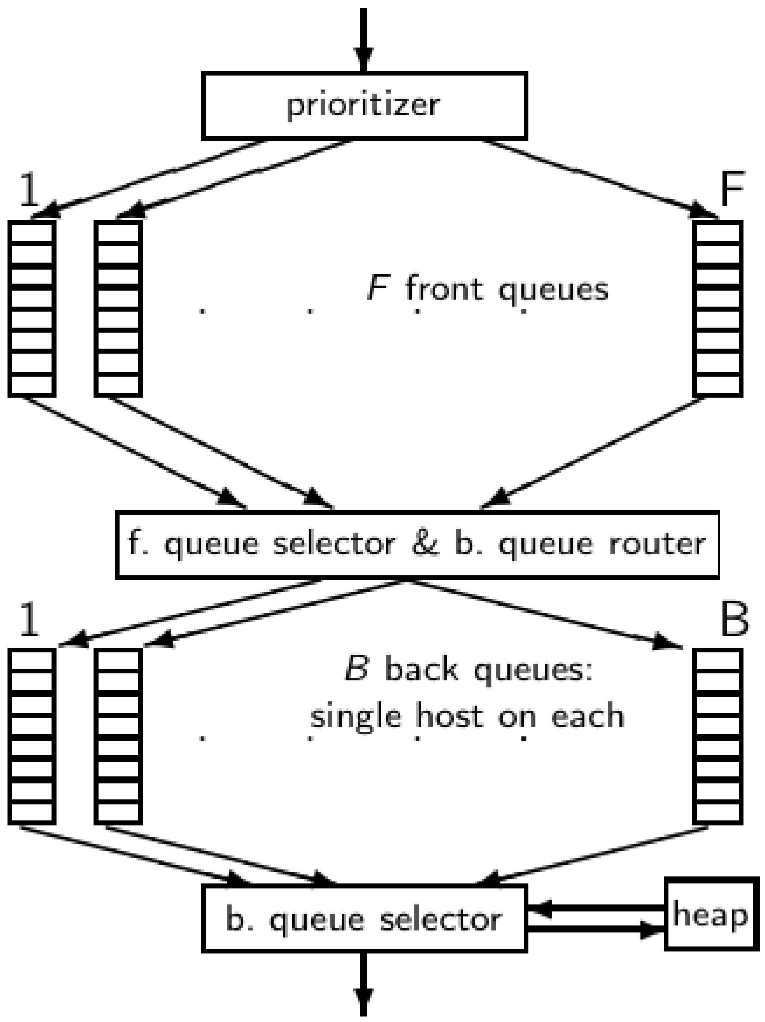
URL frontier: Two main considerations

- Politeness: Don’t hit a web server too frequently
  - E.g., insert a time gap between successive requests to the same server
- Freshness: Crawl some pages (e.g., news sites) more often than others
- Not an easy problem: simple priority queue fails.

Mercator URL frontier

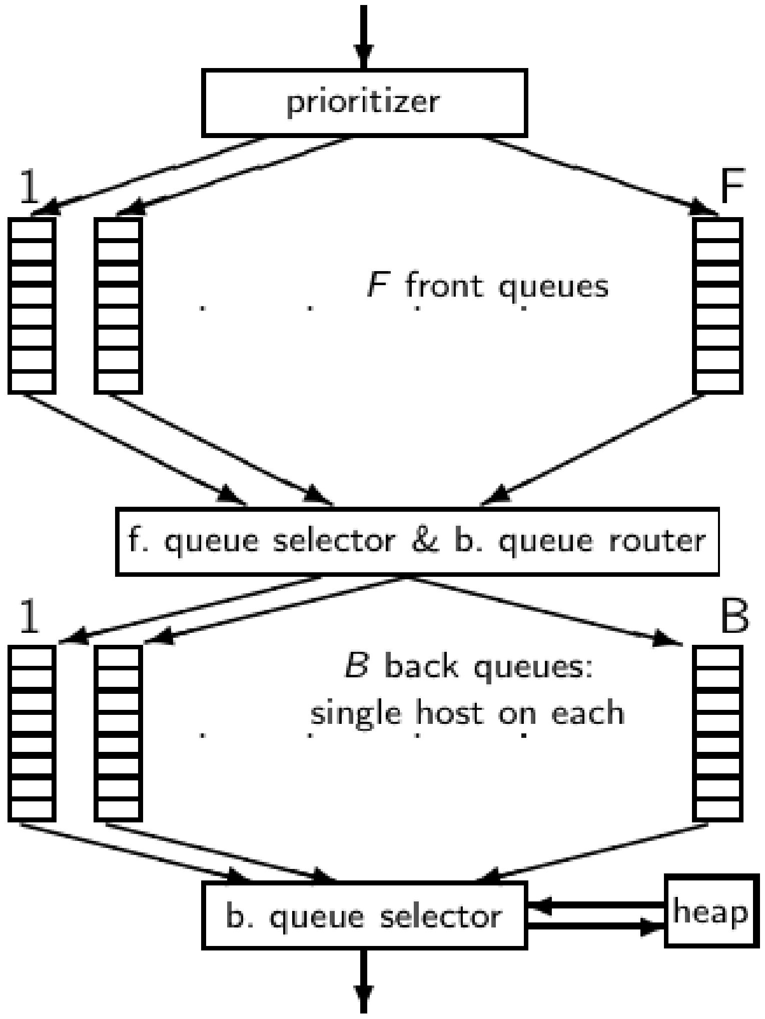


Mercator URL frontier



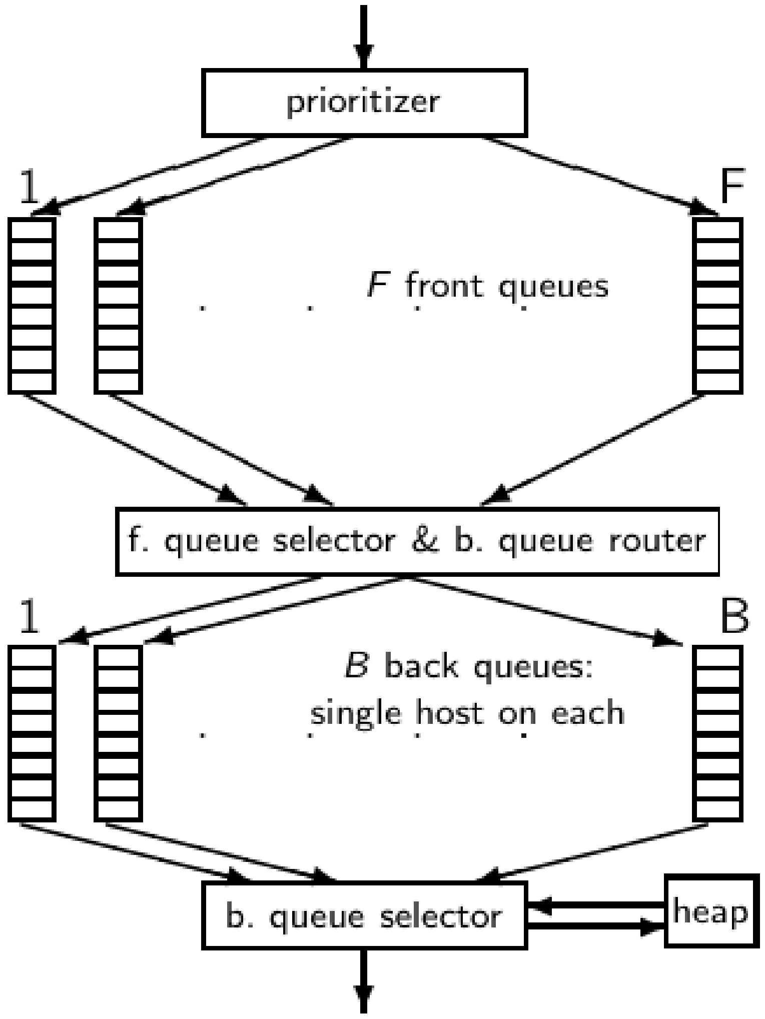
- URLs flow in from the top into the frontier.

Mercator URL frontier


prioritizer
1
F
F front queues
f. queue selector & b. queue router
1
B
B back queues: single host on each
b. queue selector
heap

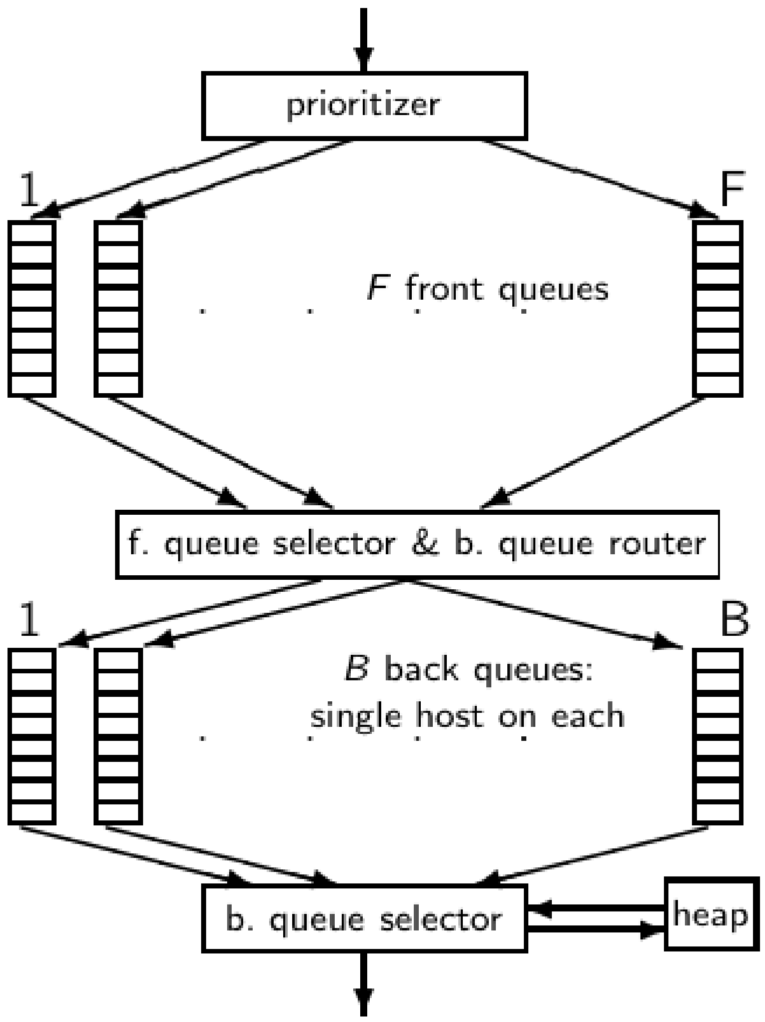
- URLs flow in from the top into the frontier.
- Front queues manage prioritization.

Mercator URL frontier



- URLs flow in from the top into the frontier.
- Front queues manage prioritization.
- Back queues enforce politeness.

Mercator URL frontier



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

Resources

- Chapter 20 of IIR