Web Crawling

Alfan F. Wicaksono
Fakultas Ilmu Komputer
Universitas Indonesia

Web Crawling

- Proses untuk mengumpulkan halaman-halaman Web, untuk nantinya akan di-index dan mendukung search engine.

- **Goal:** Bagaimana secara **cepat** dan **efisien** mengumpulkan banyak halaman Web beserta struktur link yang menghubungkan mereka semua?

Web Crawling = Graph Traversal

URL Frontier: Sebuah struktur data seperti **Queue** , **Stack**, atau yang lainnya

```
S = {seed URLs or Pages}
repeat
   remove an element s from S
   if the content of s not seen before:
      add s to index
   foreach (s -> v):
      if URL v not crawled before and
         URL v not in URL filter:
          insert v in S
```

Tarik halaman **s** dari Web dan parsing HTML-nya

Untuk setiap URL **v** yang ada di dalam halaman **s** (<a href=…>)

Fitur yang **harus** dimiliki Crawler

- Robustness
  - Kebal terhadap **spider traps**: sebuah generator halaman web yang membuat sebuah Crawler "terjebak" dan terus menerus mengambil dan parsing halaman-halaman Web yang tidak berhingga banyaknya pada sebuah domain.

- Politeness
  - Ada aturan yang mengatur seberapa sering Crawler harus akses sebuah domain; atau aturan tentang halaman-halaman Web apa saja yang tidak boleh diakses pada suatu domain.

Explicit & Implicit Politeness

- Explicit Politeness
  - Spesifikasi yang dibuat oleh Webmaster suatu domain tentang bagian pada himpunan halaman Web mereka yang boleh diakses.
  - **Robots.txt**

- Implicit Politeness
  - Sebuah etika umum bahwa Crawler yang kita buat jangan terlalu sering akses suatu domain (perlu diatur dengan jangka waktu).
  - *"only one connection is open at a time to any host"*
  - *"a waiting time of a few seconds occurs between successive requests to a host"*

Apa itu robots.txt?

Sebuah file yang disediakan suatu website tentang bagaimana bots atau orang lain bisa scrap website tersebut.

Contoh:

https://www.detik.com/robots.txt

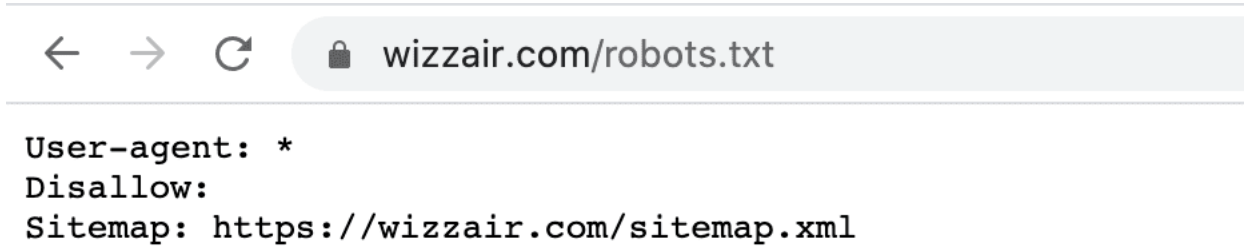
```
User-agent: Googlebot
Disallow: */komentar$
Disallow: */komentar?*
Disallow: */komentar/
Disallow: /ajax/
Disallow: /api/
Disallow: /search/
Disallow: /tag/news/
Disallow: /tag/foto/
Disallow: *?tag_from
Disallow: *?_ga
Disallow: *&sortby
Disallow: *?device=desktop
Disallow: *&device=desktop

User-agent: ChatGPT-User
Disallow: /

User-agent: OpenAI
Disallow: /

User-agent: CCBot
Disallow: /

User-agent: *
Allow: /
Sitemap: https://www.detik.com/sitemap.xml
```

Web crawler Google

Sisanya boleh melakukan scraping

Apa itu robots.txt?

Semuanya diizinkan untuk scraping:



Semuanya tidak diizinkan untuk scraping:

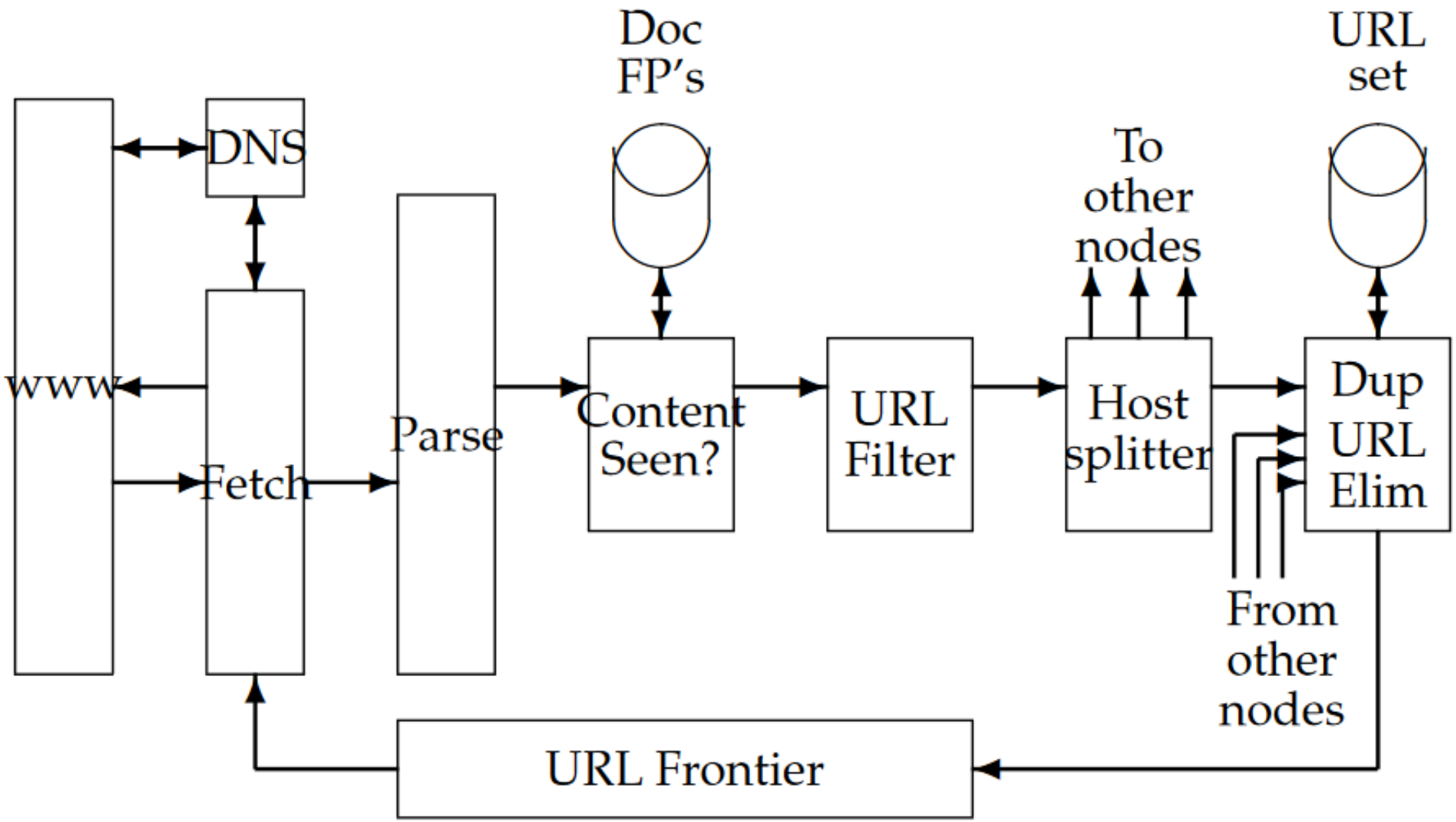
```
User-agent: *
Disallow: /
```

Fitur yang **perlu** dimiliki Crawler

- **Distributed**: bisa dijalankan di banyak thread atau bahkan di banyak mesin.
- **Scalable**: Jika mesin bertambah, ***crawl rate*** juga harus meningkat.
- **Performance & Efficiency**: perlu efisien dalam menggunakan CPU, *storage*, dan *network bandwidth*.
- **Quality**: perlu menarik halaman Web yang berkualitas terlebih dahulu

Rates reported for serious crawlers: **200-400 pages/sec**.

Arsitektur Umum (IR Book)



DNS (Domain Name Server)

- **DNS lookup**: proses konversi dari *domain name* ke *IP address* yang unik.

- Proses DNS lookup:
  - Proses lookup tidak hanya terjadi pada sebuah DNS server, tetapi bisa melibatkan banyak DNS server, melibatkan banyak requests dan banyak round-trips di Internet.
  - Well-known bottleneck in Web crawling!

DNS Cache

- Gunakan Cache untuk menyimpan mapping yang sebelumnya sudah pernah dilakukan DNS lookup; sehingga di masa mendatang menghindari akses ke DNS server di Internet.

- Implementasi local DNS cache jangan **synchronous**! karena request oleh thread lain pada mesin yang sama akan **blocked** menunggu request yang lebih awal selesai dahulu.

- Coba cek implementasi DNS yang asynchronous!
  - https://www.gnu.org/software/adns/

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Fetching & Parsing

BeautifulSoup: sebuah library pada Python untuk melakukan parsing pada HTML.

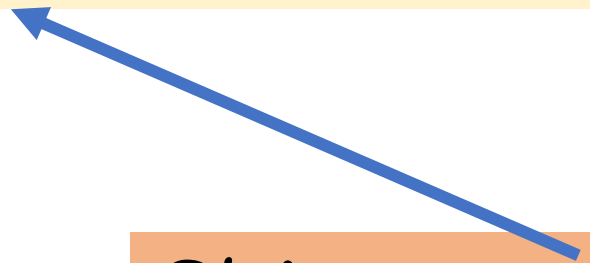
```
html_doc = """
<html><head><title>The Dormouse's story</title></head>
<body>
  <p class="title"><b>The Dormouse's story</b></p>
  <p class="story">Once upon a time there were three little sisters; and their names were
    <a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
    <a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
    <a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
    and they lived at the bottom of a well.
  </p>

  <p class="story">...</p>
</body>
</html>
"""
```

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Fetching & Parsing

Memuat semua "string" HTML ke BeautifulSoup

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(html_doc, 'html.parser')
```



Object soup (abstraksi struktur data dari parsed HTML)

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

```
soup.title
# <title>The Dormouse's story</title>

soup.title.name
# u'title'

soup.title.string
# u'The Dormouse's story'

soup.title.parent.name
# u'head'

soup.p
# <p class="title"><b>The Dormouse's story</b></p>

soup.p['class']
# u'title'

soup.a
# <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>
```

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

```
soup.find_all('a')
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
#  <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
#  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]

soup.find(id="link3")
# <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
```

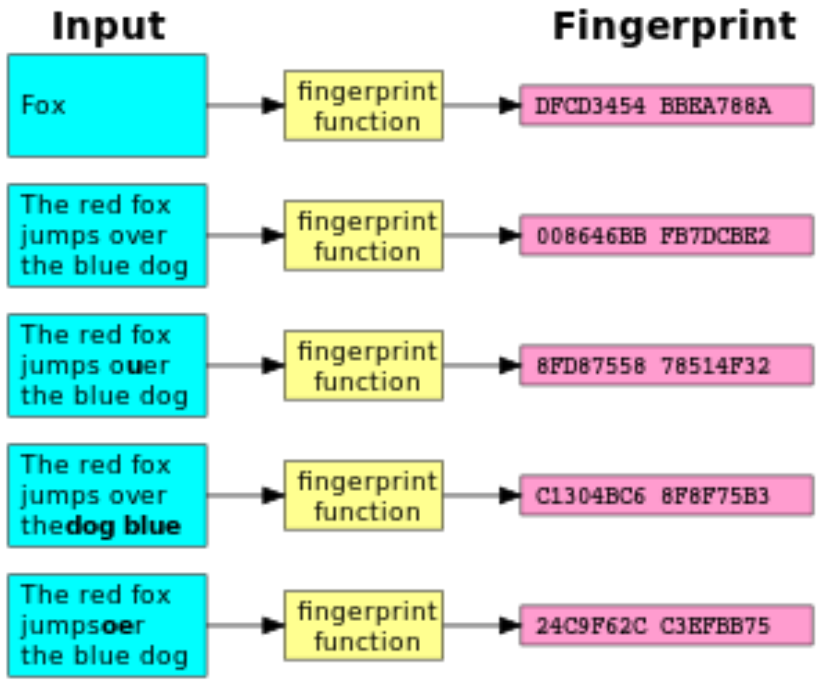
```
for link in soup.find_all('a'):
    print(link.get('href'))

# http://example.com/elsie
# http://example.com/lacie
# http://example.com/tillie
```

Parsing & Fetching

- Limit download size
- Avoid redirect loops
- URL Normalization
  - When a fetched document is parsed, some of the extracted links are relative URLs
  - www.example.com has a relative link to /andrew. So we need to normalize the link into: www.example.com/andrew

Content Seen?

- Web banyak mengandung dokumen duplikat.
- Jika sebuah halaman Web yang baru ditarik ternyata sudah ada di index, maka jangan dilanjutkan prosesnya.
- Pemeriksaan bisa diimplementasikan dengan dokumen **fingerprint**, seperti **checksum**, yang memetakan (semacam **hash function**) isi dokumen ke **bit string pendek** dan **unik**.



https://en.wikipedia.org/wiki/Fingerprint_(computing)

URL Filter & Robots.txt

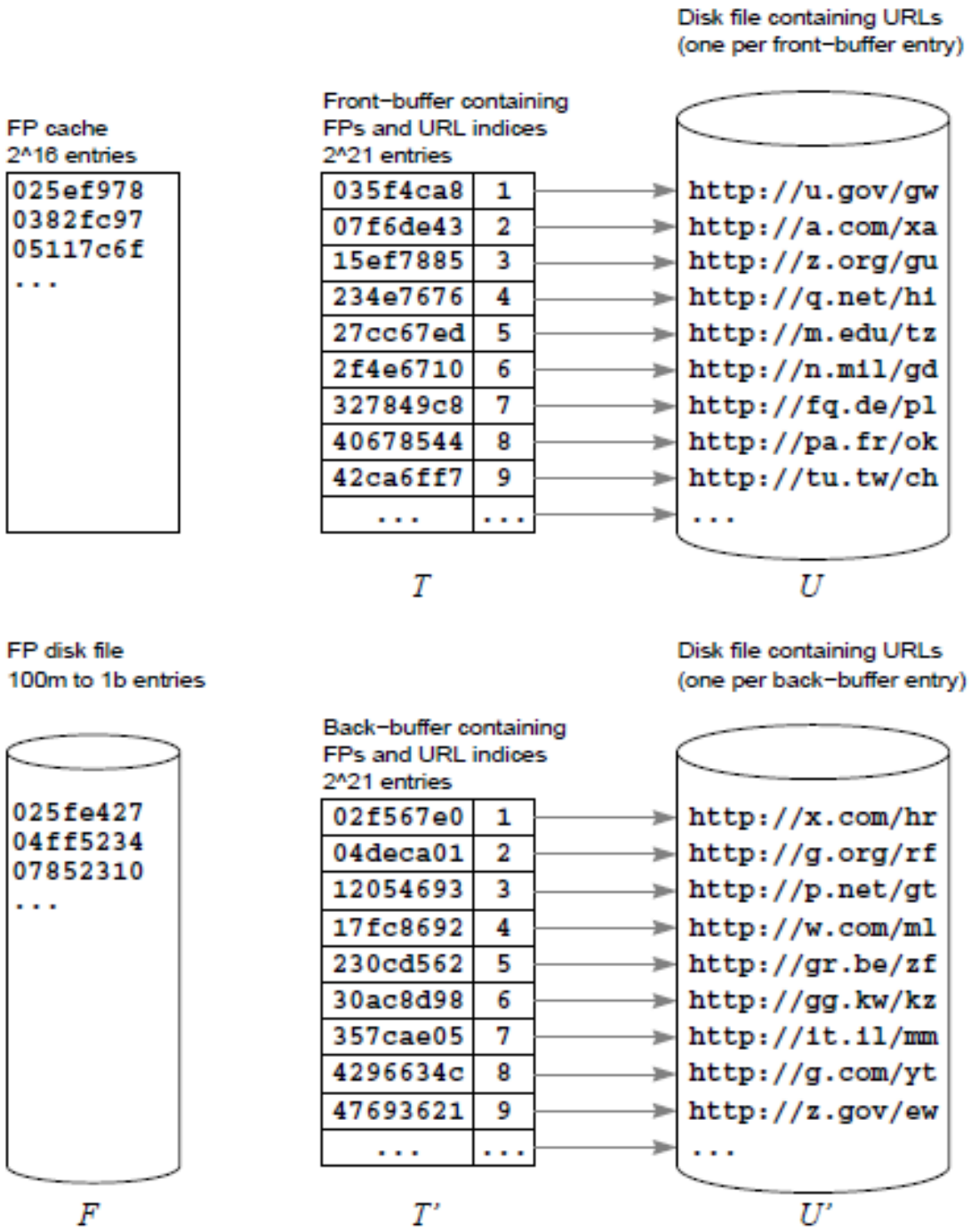
- URL Filter bisa diimplementasikan dengan **Regular Expression** untuk memeriksa apakah sebuah URL boleh di-crawl atau tidak.

- Robots.txt untuk sebuah domain jangan di-download berkali-kali. Cukup sekali saja.
  - Perlu mekanisme untuk **Robots.txt Cache**

Duplicate URL Eliminator (DUE)

Najork & Heydon, 2001 (Mercator)

- DUE maintains an **in-memory hash table** of all URLs that have been encountered before.
- To save space, the table stores **8-byte checksums** (or fingerprint, FP) of the URLs rather than the URLs themselves (Rabin's fingerprinting algorithm).
- **In-memory FP (fingerprint)** cache is needed for **popular** ones.
- Using this method, storing the checksums of 1 billion URLs in such a hash table requires slightly over 5 GB.



Distributing the Crawler

- Jalankan Crawler pada **banyak thread**, dan **banyak mesin**
- Pertimbangkan **Geographically distributed machines**

- **Host Splitter:** Daftar host yang akan di-crawl perlu dipartisi ke bebera mesin -> **satu mesin untuk menangani satu host**

URL Frontier

- "Sederhananya" hanya Priority Queue. Namun, tidak bisa dengan Priority Queue "sederhana" ☺

- Isu yang perlu diperhatikan:
  - **Scalability**: 100M of URLs = GBs of data --> tidak muat memory?
  - **Politeness**: jangan membebani Web host
  - **Priority**: beberapa dokumen punya prioritas yang lebih tinggi untuk diproses lebih dahulu.
  - **Self-loop**: beberapa halaman Web sengaja berisi link ke URL dirinya sendiri.

URL Frontier: Disk-based Queues

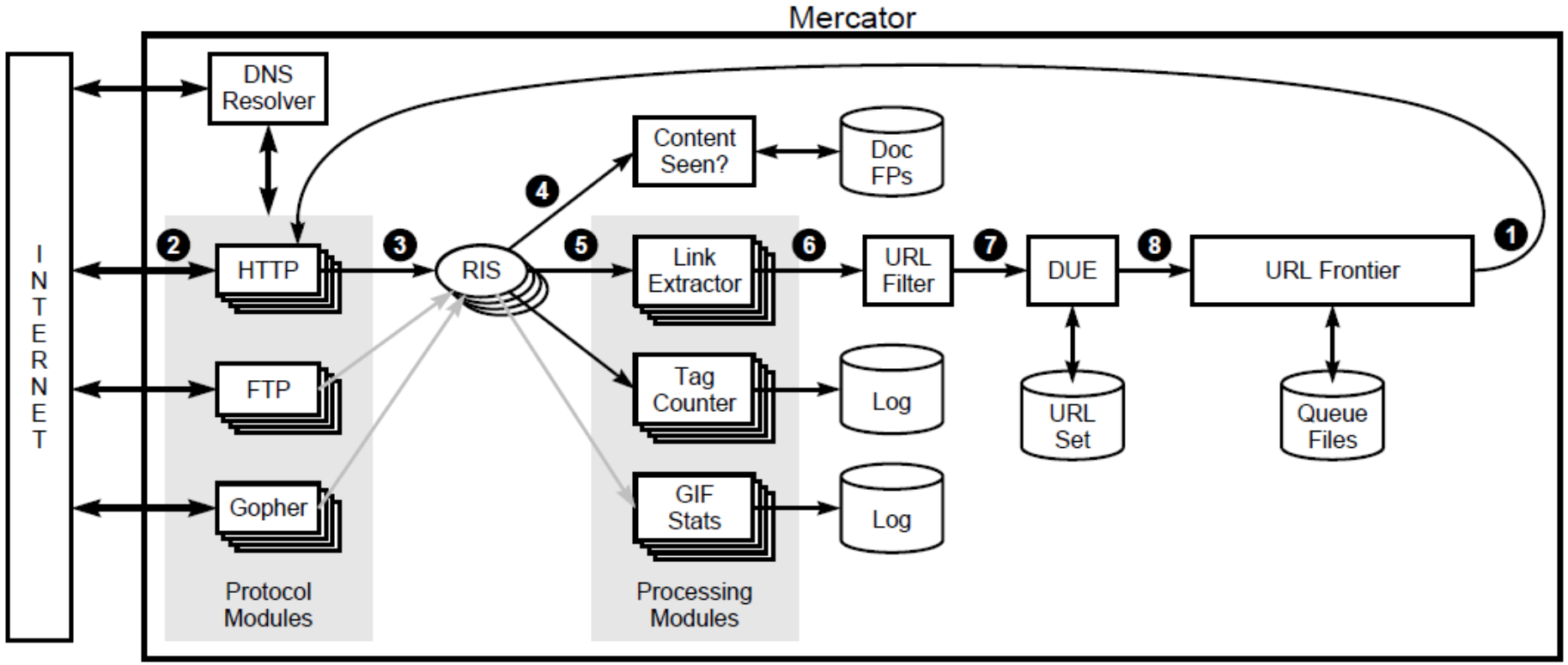
- Bagi isi queue menjadi **beberapa block**
- Block yang berisi **head** dan **tail** dari queue disimpan di memori (**cache**); sisanya disimpan di harddisk.
- Ketika **tail block** terisi penuh, tulis ke harddisk.
- Ketika **head block** kosong, muat block baru dari harddisk ke memori

- Contoh implementasi: **Persistent Queue**

https://colab.research.google.com/drive/1MFJnLEzV2gWld5dedIBy-2spbqbUSQsK?usp=sharing

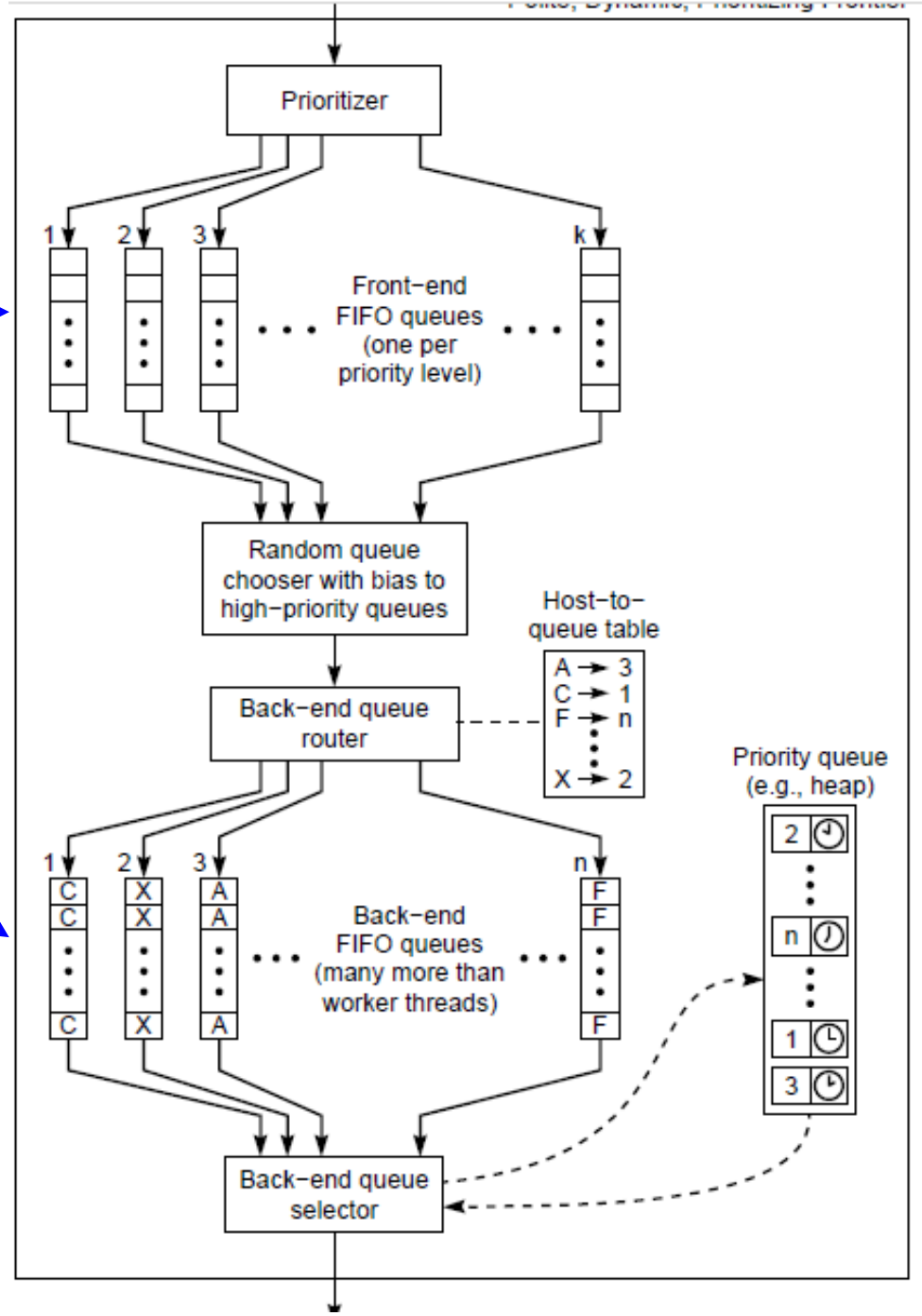
Contoh Crawler: Mercator Scheme

- Marc Najork and Allan Heydon. 2001. *High-Performance Web Crawling*. Technical Report 173, Compaq Systems Research Center.
- http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-173.pdf



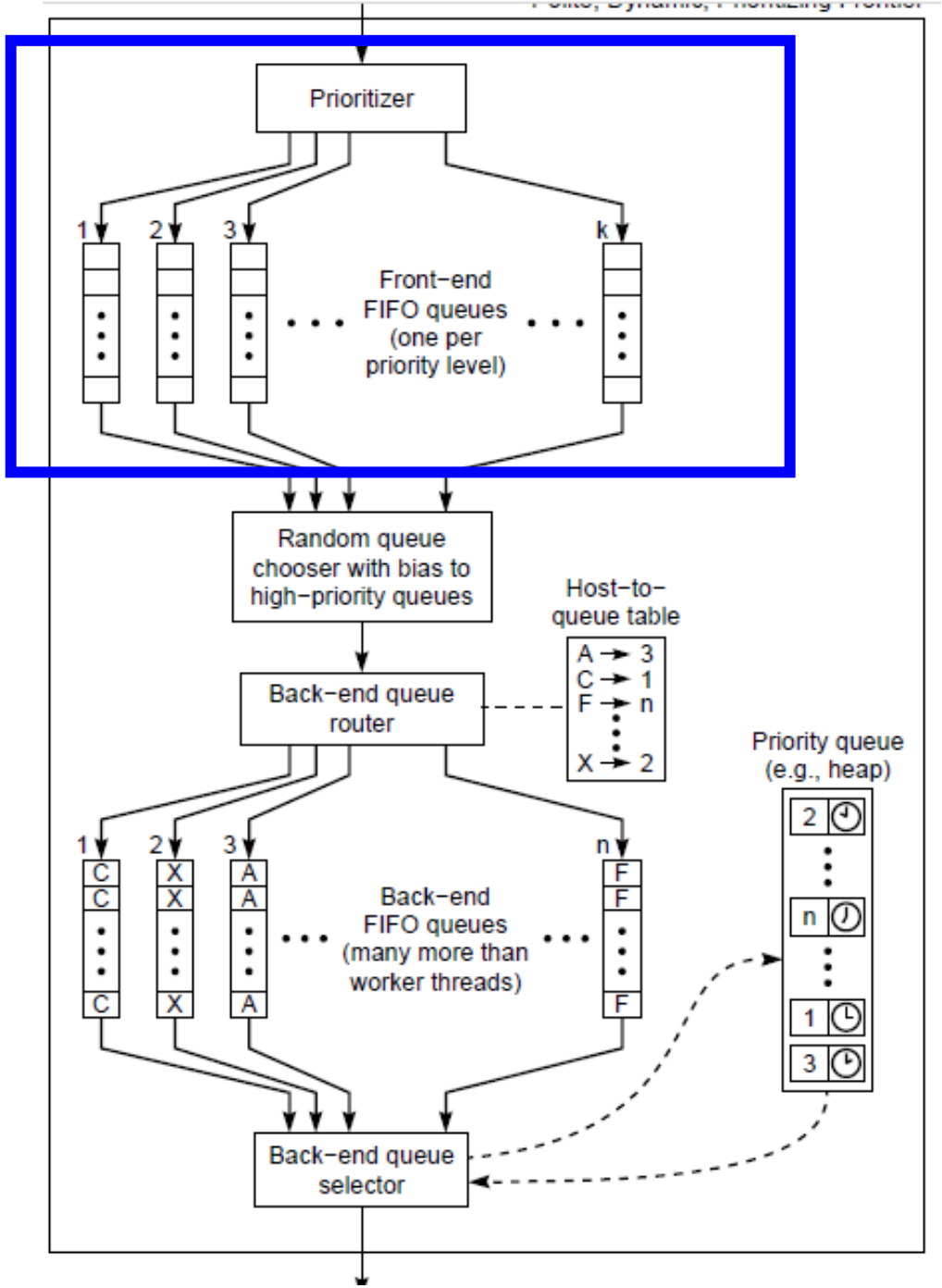
URL Frontier pada Mercator

- **Front-end Queues** manage prioritization
- **Back-end Queues** ensure politeness



Front-end Queues

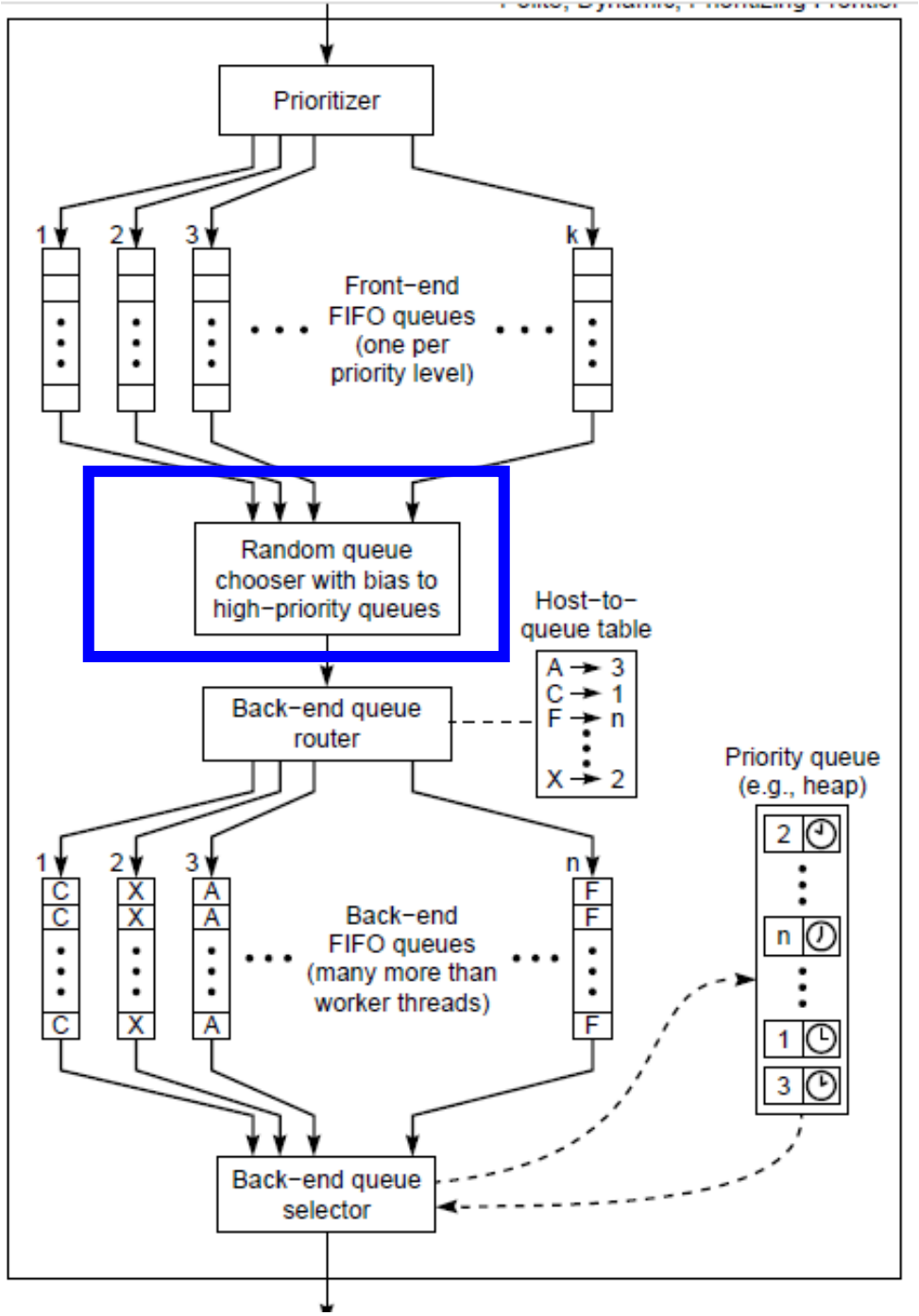
- Masing-masing queue mempunyai label nilai prioritas **1 – k**.
- Ketika ada URL yang masuk, sistem akan memberikan nilai prioritas **1 - k** ke URL tersebut dan kemudian masuk ke Queue yang bersesuaian.
- Menentukan prioritas?
  - Dokumen yang isinya sering berubah pada masa lampau akan mendapatkan prioritas tinggi.
  - URL dari situs berita biasa punya prioritas tinggi



Biased Front Queue Selector

- Ketika sebuah Back-end Queue kosong, ia meminta URL dari Front-end Queue; perlu ada mekanisme **untuk memilih URL** dari Front Queue.

- Strategi
  - Bisa dengan **Round-Robin** tetapi bias terhadap prioritas
  - Bisa juga dengan **randomized selection**, tetapi dengan distribusi random yang bias terhadap prioritas (tidak uniform)

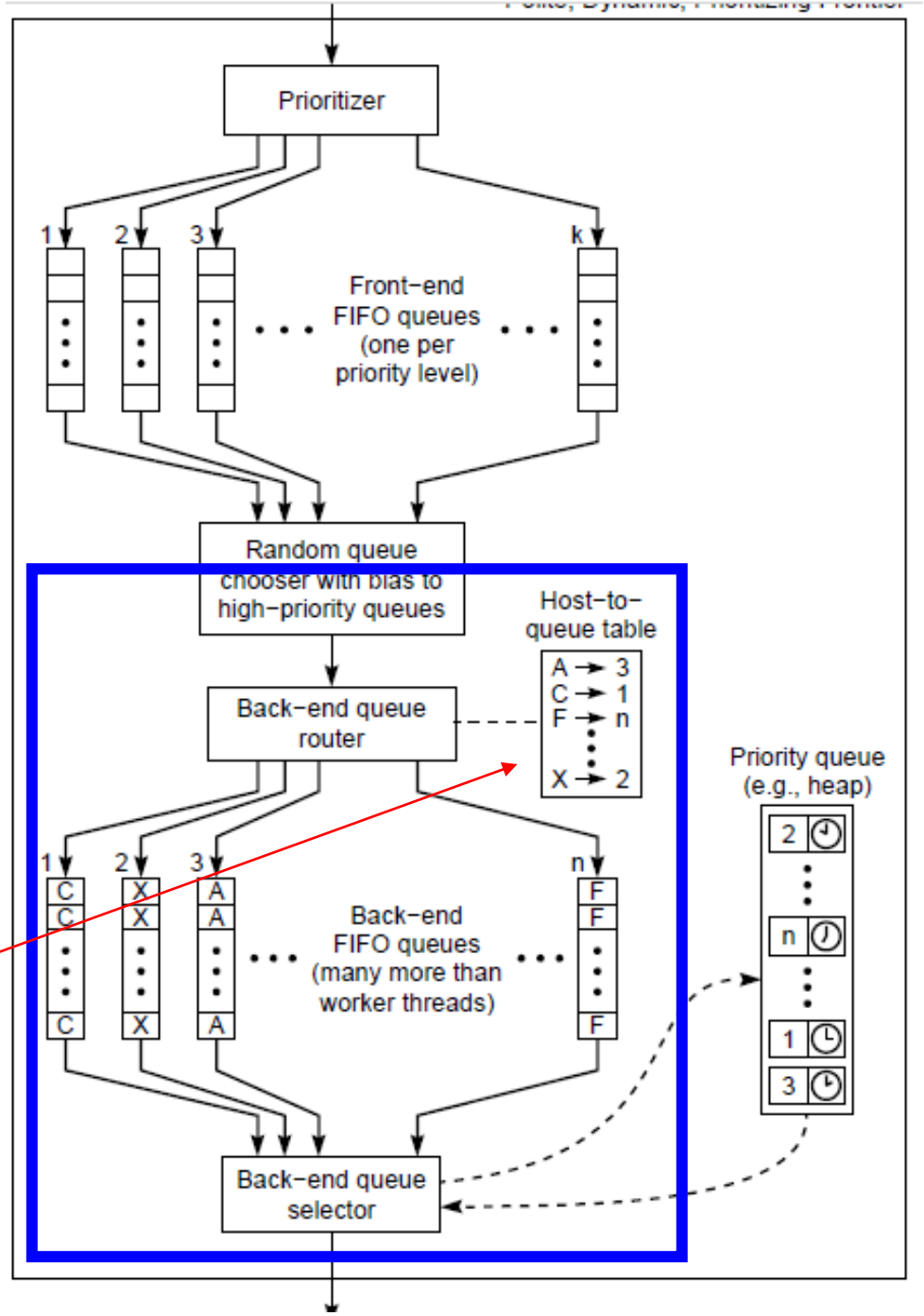


Back-end Queues

Invariants:

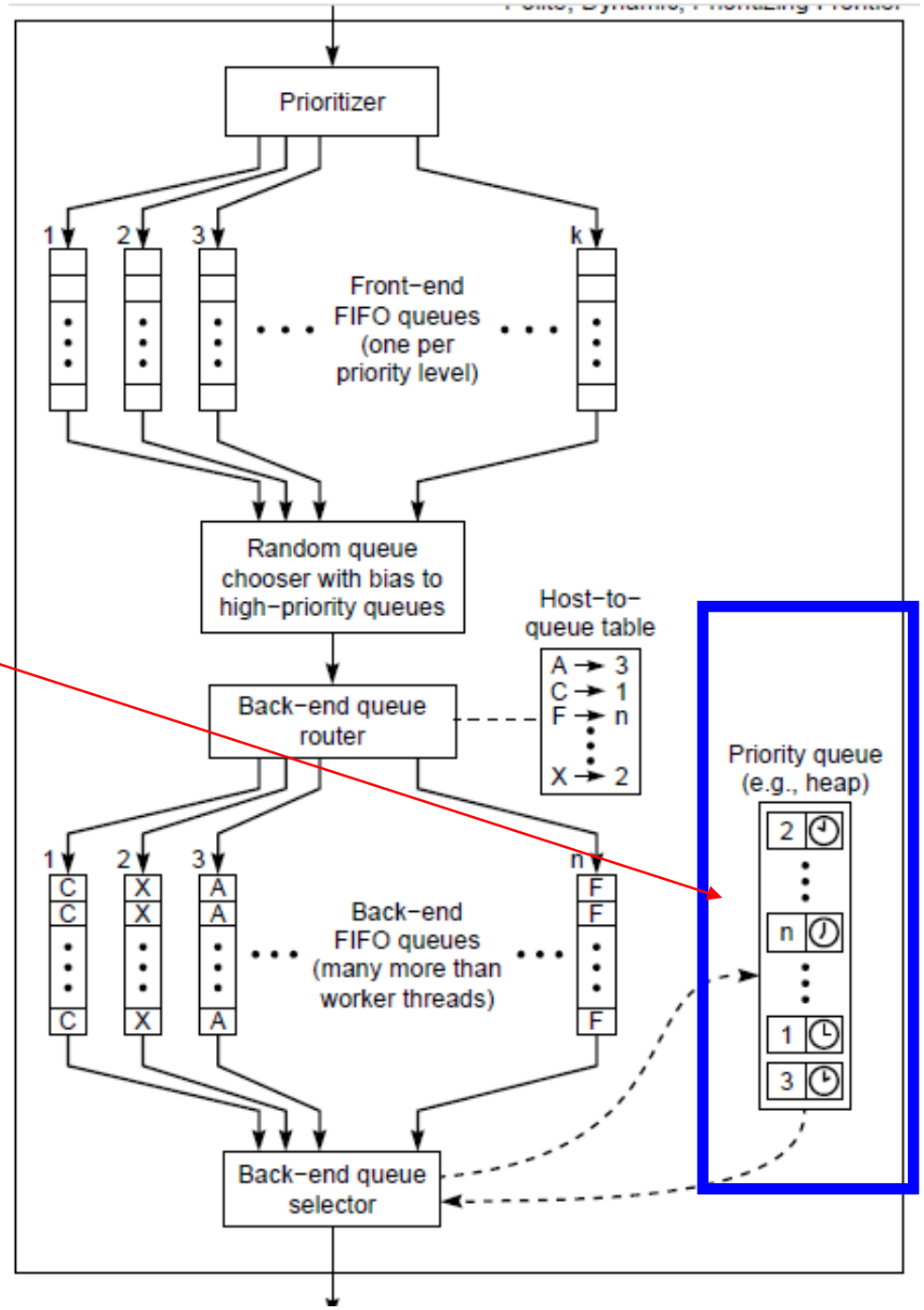
- Selama Crawler masih berjalan, setiap **back-end queue tidak boleh kosong**.

- **Setiap back-end queue hanya mengandung URL dari satu host saja**
  - Ada tabel Host-to-queue untuk menjaga mapping dari Host ke nomor Back-end queue.



Back-end Queue Heap

- Satu entry untuk satu back-end queue
- Sebuah entry berisi informasi jadwal waktu kapan Host pada back-queue yang bersesuaian akan ditarik dan diproses lagi.
- Bagaimana assign jadwal?
  - Salah satunya berdasarkan waktu terakhir kapan dokumen tersebut diambil
  - **Contoh:** jadwal baru = waktu saat ini + 10 * waktu pengambilan terakhir



Contoh Implementasi URL Frontier

Credit:
Steve Howard, Thumbtack, Inc.

```
def ready_sites(self):
  while (self._host_heapq and self._host_heapq[0][0] <= time_now):
    can_crawl_at_time, host = heapq.heappop(self._host_heapq)
    queue = self._host_queue_assignments[host]
    self._num_enqueued -= 1
    yield queue.get()

def fill_host_queue(self, queue):
  while self._front_queue:
    fetch_task = self._front_queue.get()
    host = self._get_host(fetch_task.site)
    if host in self._host_queue_assignments:
      self._host_queue_assignments[host].put(fetch_task)
    else:
      self._unassigned_queues.remove(queue)
      self._host_queue_assignments[host] = queue
      queue.put(fetch_task)
      self._add_to_heap(host)
      return
```

How to build "real" Mercator?

- https://dnsbelgium.github.io/mercator-workshop/1