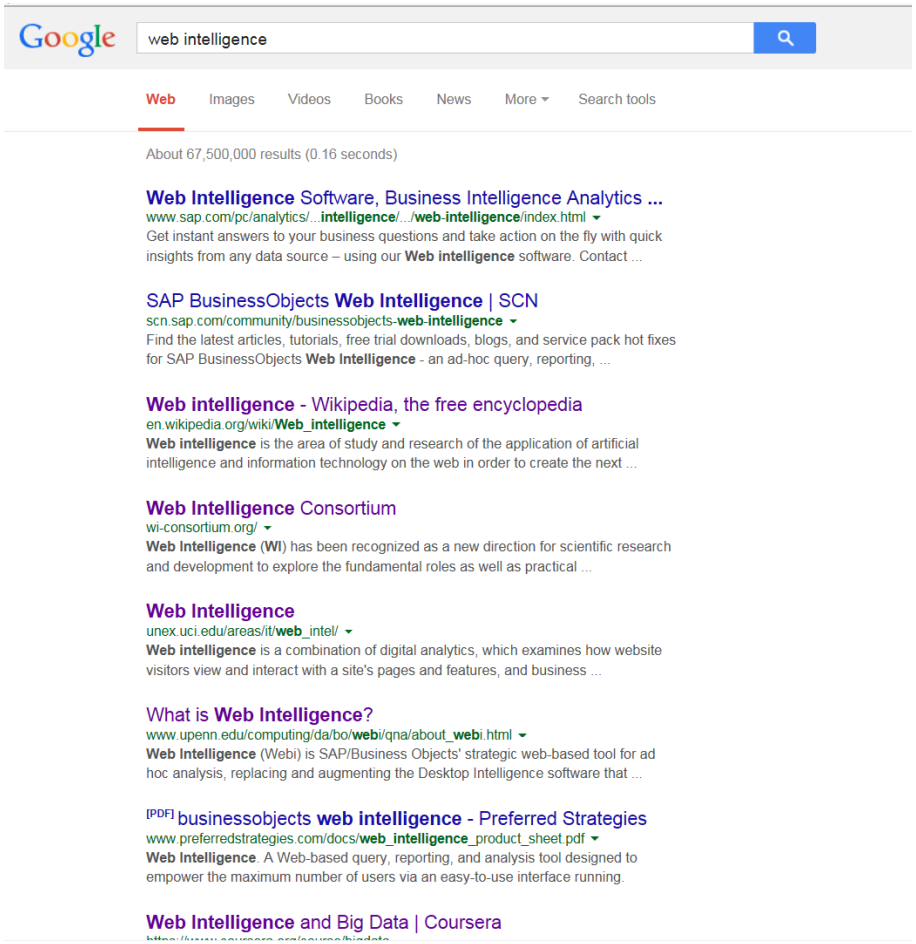
AALBORG UNIVERSITY

Web Crawlers

Peter Dolog
dolog@cs.aau.dk
http://people.cs.aau.dk/~dolog

Based (heavily) on Stanford slides by Christopher Manning & Pandu Nayak ,on the 'Introduction to Information Retrieval ' book (Chap. 20) by Christopher Manning, Prabhakar Raghavan & Hinrich Schütze, and Bo Thiesson, AAU.

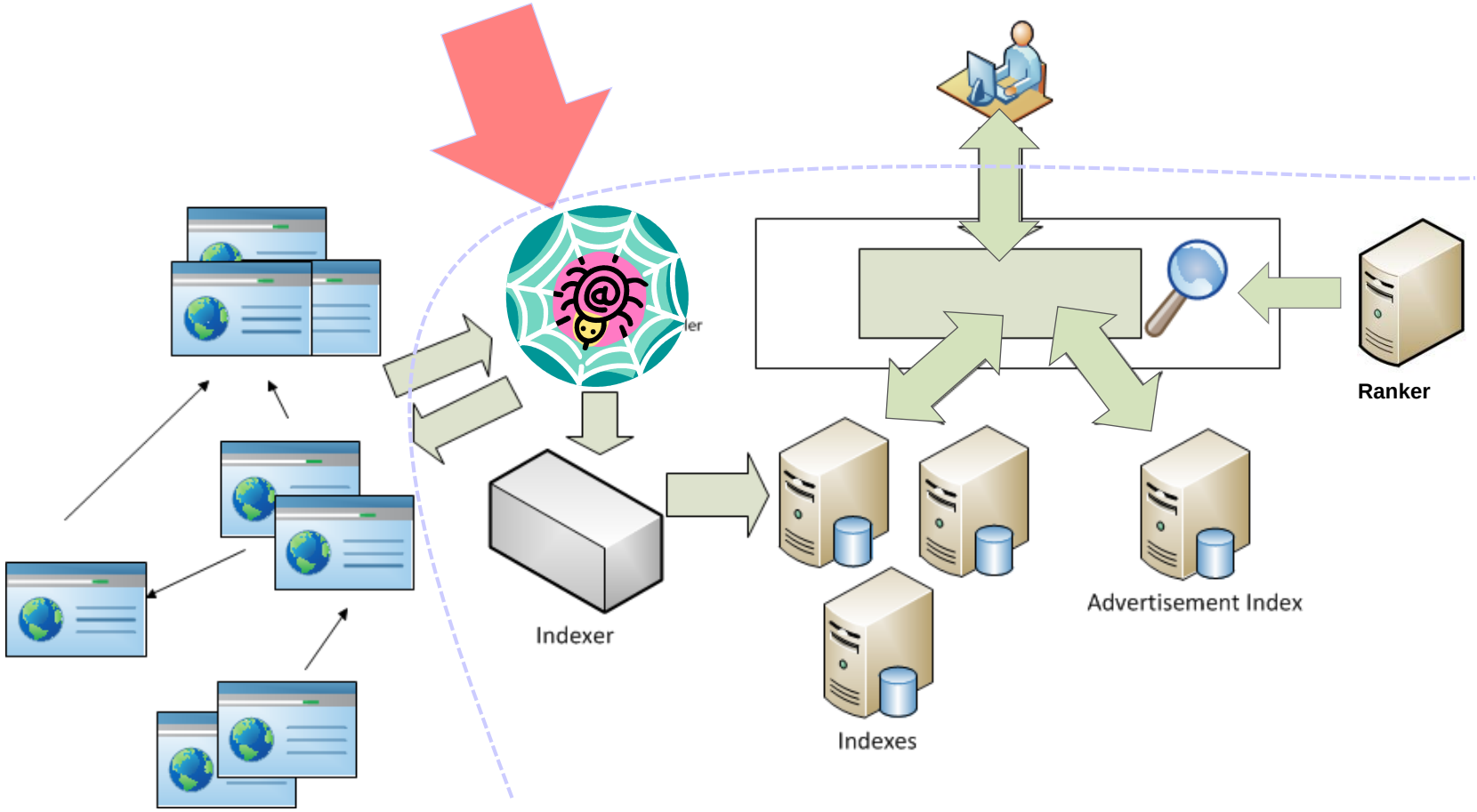
How does Google know about this!!!



Outline

- Search engine architecture (recap)
- Basic crawl architecture
  - Politeness (robots.txt)
  - Crawl priority
  - URL Frontier
  - Specific components in architecture
- Distributed crawl architecture

Search Engine Architecture



Web Crawler (aka. Spider)

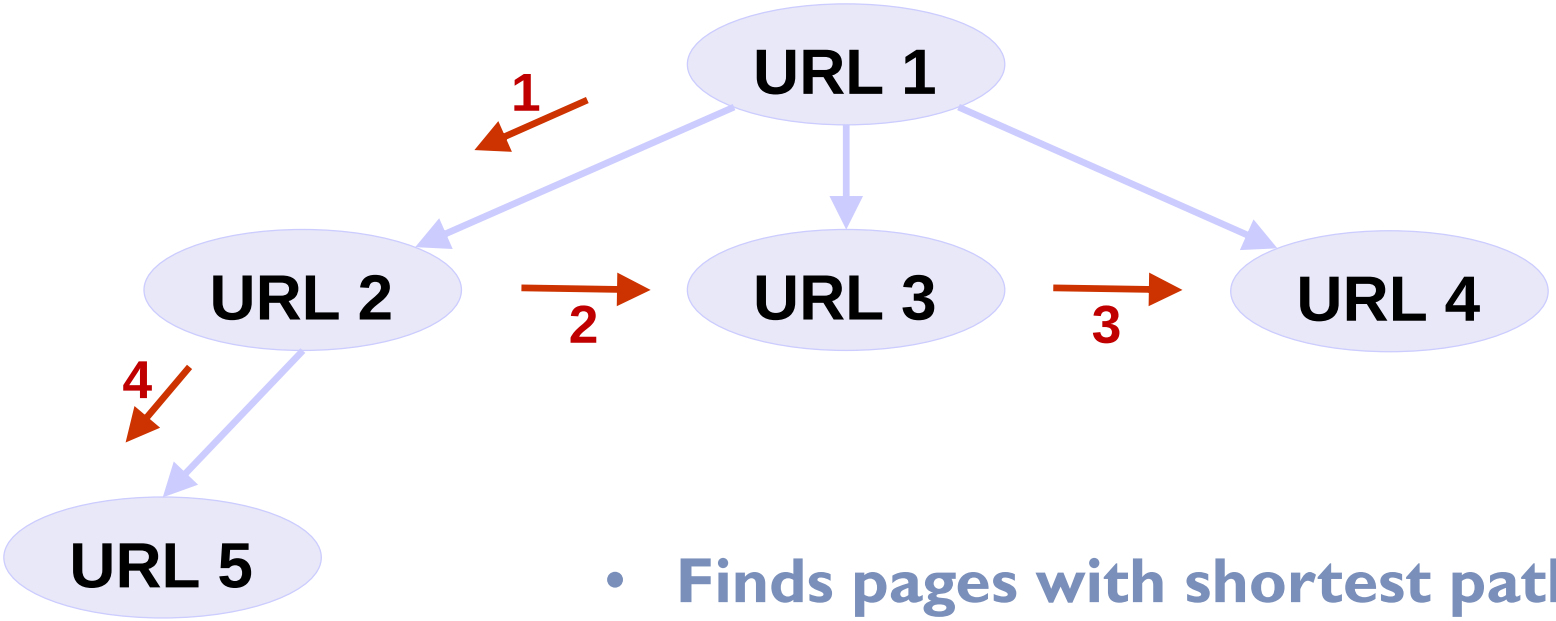
- A program that navigates the hypertext structure of the Web, gather pages for the indexer.
  - Input: Seed URLs
  - Output: Content & structural representation for (part of) the Web
  - Objective: Quickly and efficiently gather as many useful pages as possible.
  - Goal: hundreds of pages per second
- Types of crawlers
  - Periodic
  - Focused
  - Incremental

10^9 pages at 100 pages/sec
= 10^7 secs = 4 months!

Basic crawling

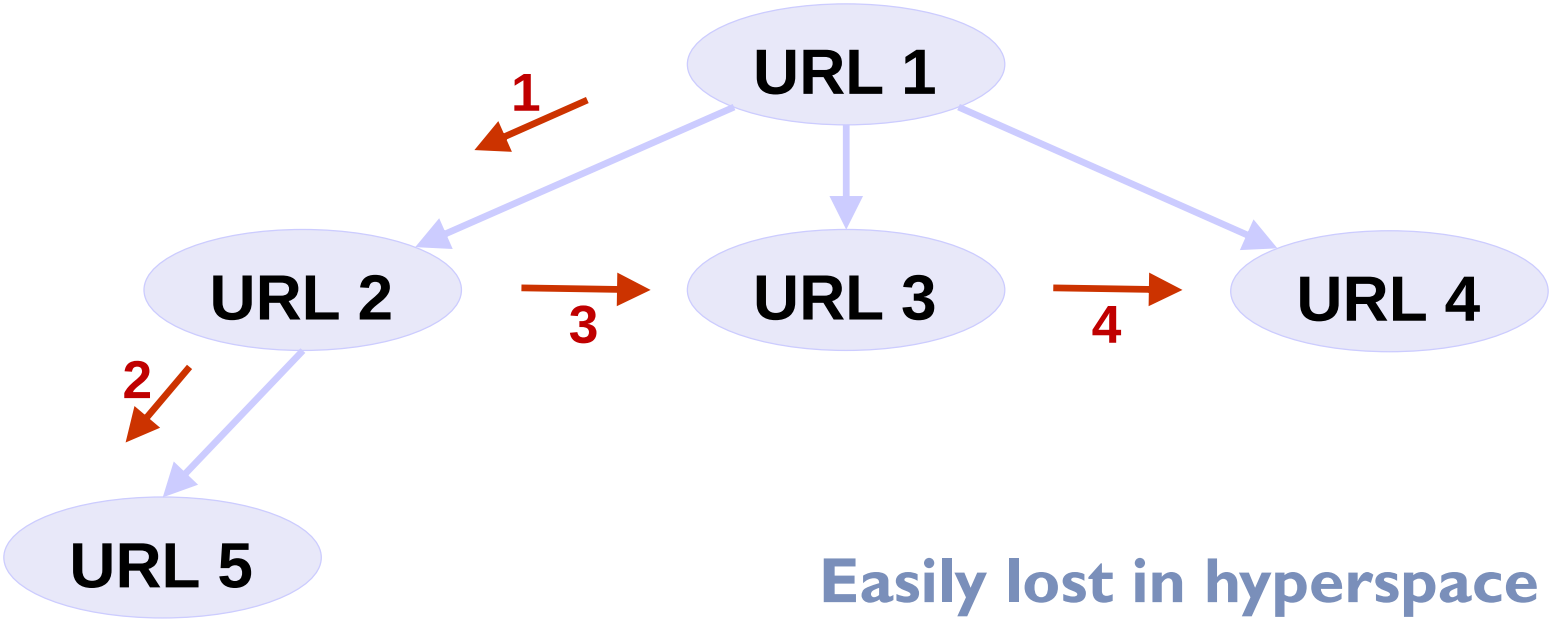
1. Begin with initial set of URLs in queue/frontier – "the seed"
2. Fetch next page from URL in queue
3. Parse page
   - Extract text and pass to indexer
   - Extract links and add to URL queue
4. Delete or re-prioritize current URL from queue
5. Repeat 2-5

(Too) Simple Crawling Strategies -- *Breadth* First Search



- Finds pages with shortest path
- Good seed set → close to topic

(Too) Simple Crawling Strategies -- *Depth* First Search



Easily lost in hyperspace

- Spam pages
- Spider traps

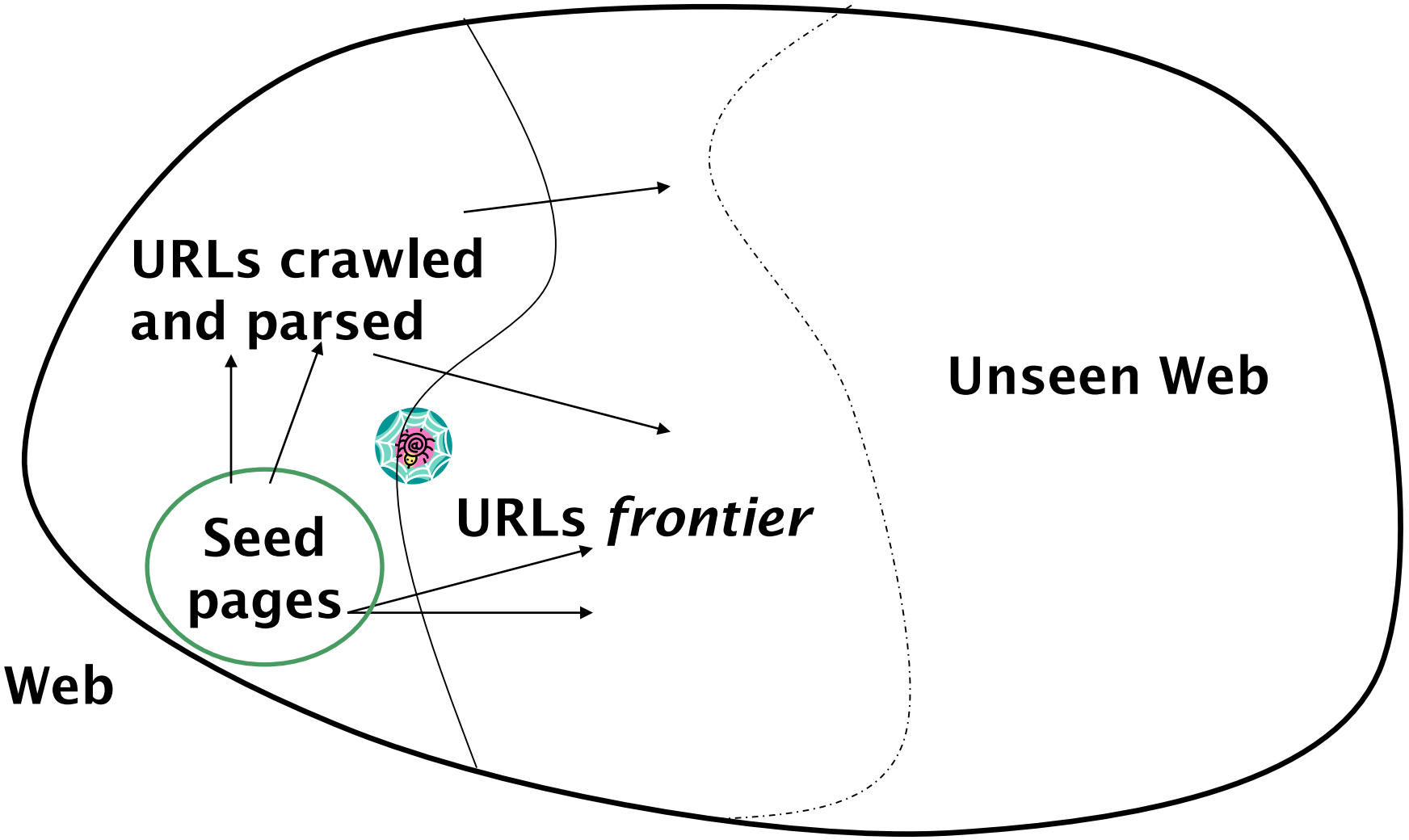
http://www.google.com/xxx

Google

404. That's an error.

The requested URL /xxx was not found on this server.
That's all we know.

Crawling picture



Simple picture – complications

Web crawling isn't feasible with one machine

- All of the above steps distributed

2.5M secs. in a month!

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

Mandatory features

Robustness

- Be immune to spider traps (infinite number of pages in particular domain) and other malicious behavior from web servers
- Be immune to faulty pages. Crawler should not fail on syntactically incorrect HTML (forgiveness)

Politeness

- Explicit politeness: specifications from webmasters on what portions of site can be crawled

`robots.txt`

- Implicit politeness: avoid hitting any site too often!

Robots.txt – Explicit Politeness

Protocol for giving crawlers ("robots") limited access to a website, originally from 1994

- http://www.robotstxt.org

Website announces its request on what can(not) be crawled

- For a server, create a file `/robots.txt`
- This file specifies access restrictions

Some robots.txt examples

To exclude all robots from part of the server

```
User-agent: *
Disallow: /cgi-bin/
Disallow: /tmp/
Disallow: /junk/
```

To exclude a single robot

```
User-agent: BadBot
Disallow: /
```

To allow a single robot

```
User-agent: Google
Disallow:

User-agent: *
Disallow: /
```

To exclude all files except one

This is currently a bit awkward, as there is no "Allow" field. The easy way is to put all files to be disallowed into a separate directory, say "stuff", and leave the one file in the level above this directory:

```
User-agent: *
Disallow: /~joe/stuff/
```

When to Fetch `robots.txt`

When committing URL to frontier ??

- Pro: Ensures high locality as the extracted links from a page refer mostly to the same host. Caching ⇒ fast
- Con: Most of the pages are of low rank/quality, thus sitting in frontiers for days ⇒ `robots.txt` might not be fresh when fetching the page

NO

When fetching page ??

- Con/Pro: Caching ⇒ not as fast, but good enough
- Pro: Ensures freshness

YES

Your HandsOn

Be polite - Be polite - Be polite:

- Only one connection open to a host at any time
- Guarantee a few secs. waiting time between successive requests to a host
- Obey `robots.txt`
- Fetch robots.txt right before page is retrieved – not when URL committed to frontier

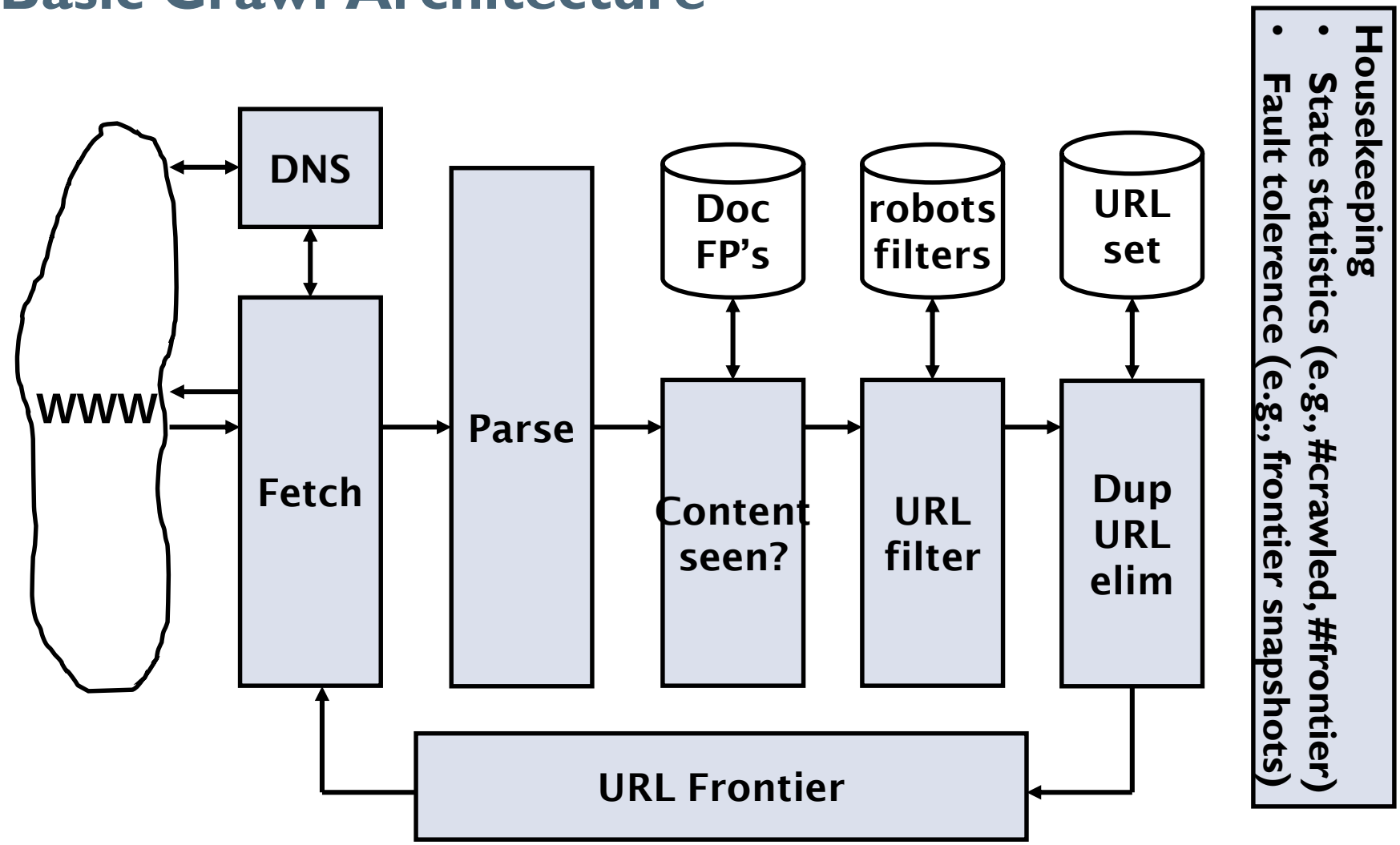
Desired Features

- Distributed – across multiple machine
- Scalable – increase crawler rate by adding machines and bandwidth
- Performance and efficiency – with respect to resources
- Quality – prioritize "better/authoritative" pages
- Freshness – observe and estimate change rates
- Extensible – new data formats, protocols, technologies

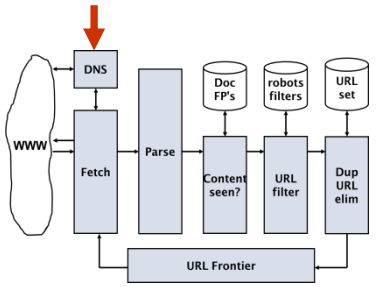
Basic Crawling (better)

1. Begin with initial set of URLs in queue/frontier – "the seed"
2. Fetch next page from URL in queue
3. Parse page
   a. Extract text ~~and pass to indexer~~
   b. Check if URL has content already seen. If not:
      - Add to index
      - Extract "link-to" URLs ~~and add to frontier queue~~
4. For each extracted URL
   a. Normalize URL
   b. Check that it passes certain URL filter tests. E.g.:
      - Focused crawl: only crawl .dk
      - Obey robots.txt (freshness caveat)
   c. Check that not already in frontier
   d. Add to frontier if passing tests
5. Delete or re-prioritize current URL from queue

Basic Crawl Architecture



DNS (Domain Name Server)



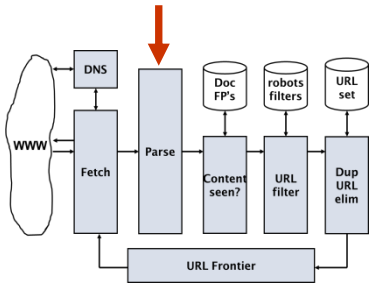
A lookup service on the internet

- Given a URL, retrieve its IP address
  - E.g. www.wikipedia.org → 198.35.26.96
- Service provided by a distributed set of servers – thus, lookup latencies can be high (even seconds)
- Standard libraries implements synchronous DNS lookups
- Bottleneck for web crawling (recall our goal of 100 pages/sec)

Solutions

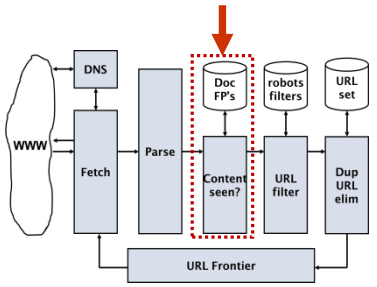
- DNS caching
- Batch DNS resolver – collects requests and sends out together
- Implement own DNS resolver

Parsing: URL normalization (canonicalization)



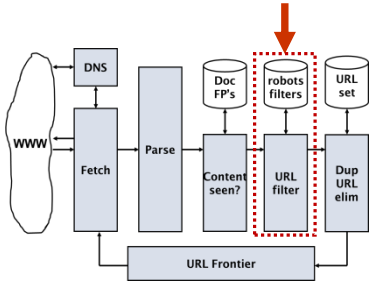
- Expand relative links into their absolute URL
  - E.g., `http://en.wikipedia.org/wiki/Main_Page` has a relative link to `/wiki/Wikipedia:General_disclaimer` which is the same as the absolute URL `http://en.wikipedia.org/wiki/Wikipedia:General_disclaimer`
- Convert the protocol and host to lower case
  - `HTTP://www.Example.com/` → `http://www.example.com/`
- Capitalize letters in escape sequences
  - `http://www.example.com/a%c2%b1b` → `http://www.example.com/a%C2%B1b`
- Decode percent-encoded octets of unreserved characters
  - `http://www.example.com/%7Eusername/` → `http://www.example.com/~username/`
- More normalization "rules" can be found on
  - `http://en.wikipedia.org/wiki/URL_normalization`

Content seen? – previous lecture!



- Widespread duplication on the web; by some estimates, 25-40% of the web is near-duplicate
- If the page just fetched is already in the index, do not further process it
- Verified using document fingerprints or shingles

Filters and robots.txt



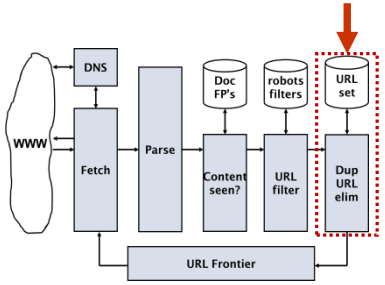
Filters

- typically, regular expressions for URLs to be crawled/not
- Lexicons

robots.txt

- 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

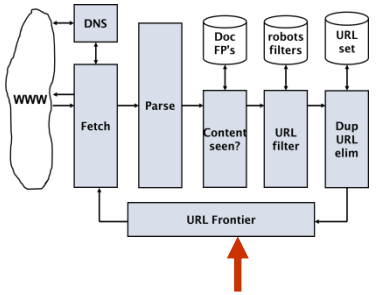
Duplicate URL elimination



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

URL frontier



Is BFS or DFS good enough? **NO!**

Two main considerations:

- Politeness: do not hit a web server too frequently
- Quality + Freshness: crawl some pages more often than others

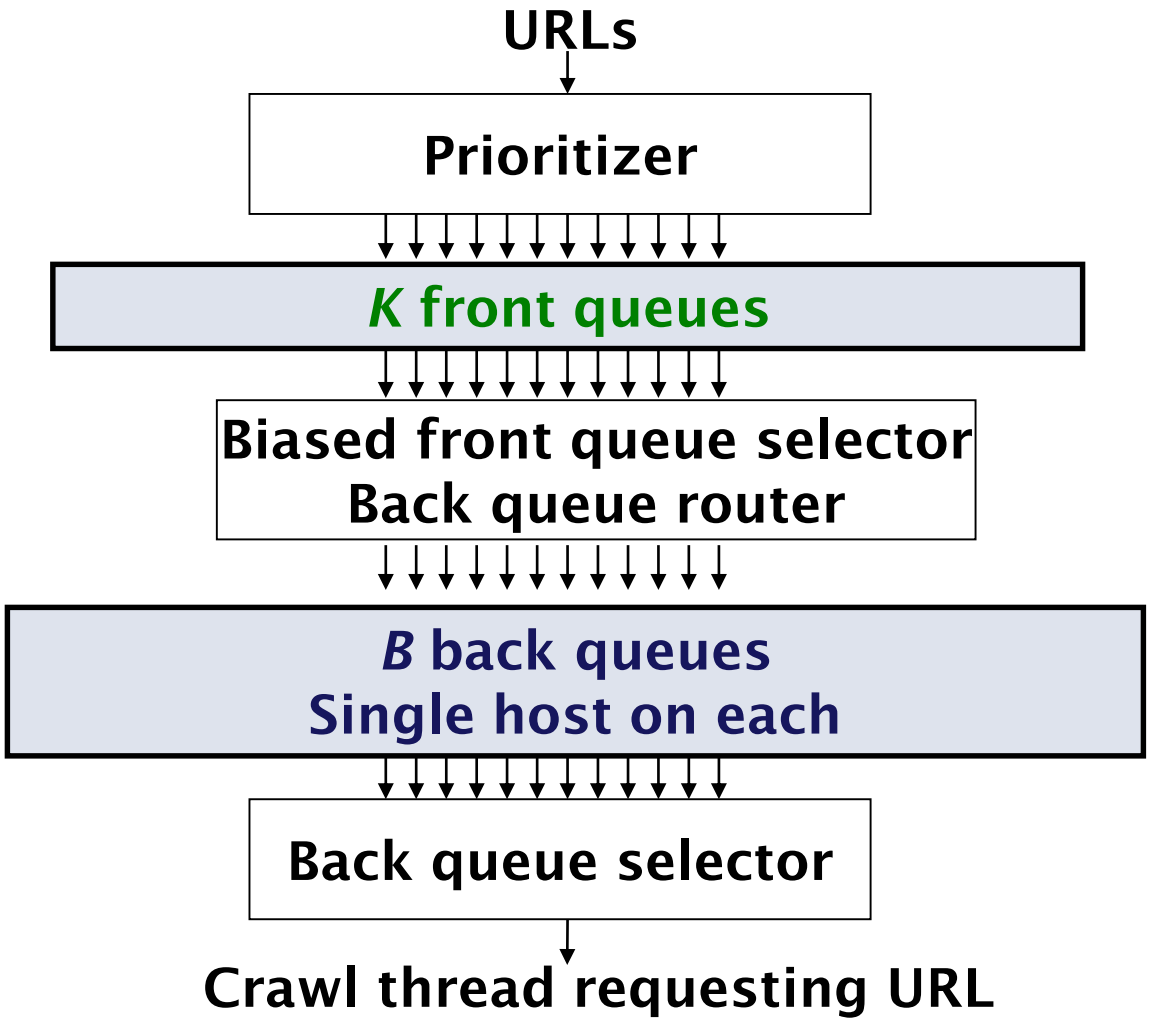
  E.g., pages (such as News sites) whose content changes often

What about simple priority queue? **NO!**

- Many links out of a page go to its own (high-quality) site, creating a burst of accesses to that site.
- Politeness and quality+freshness conflict!

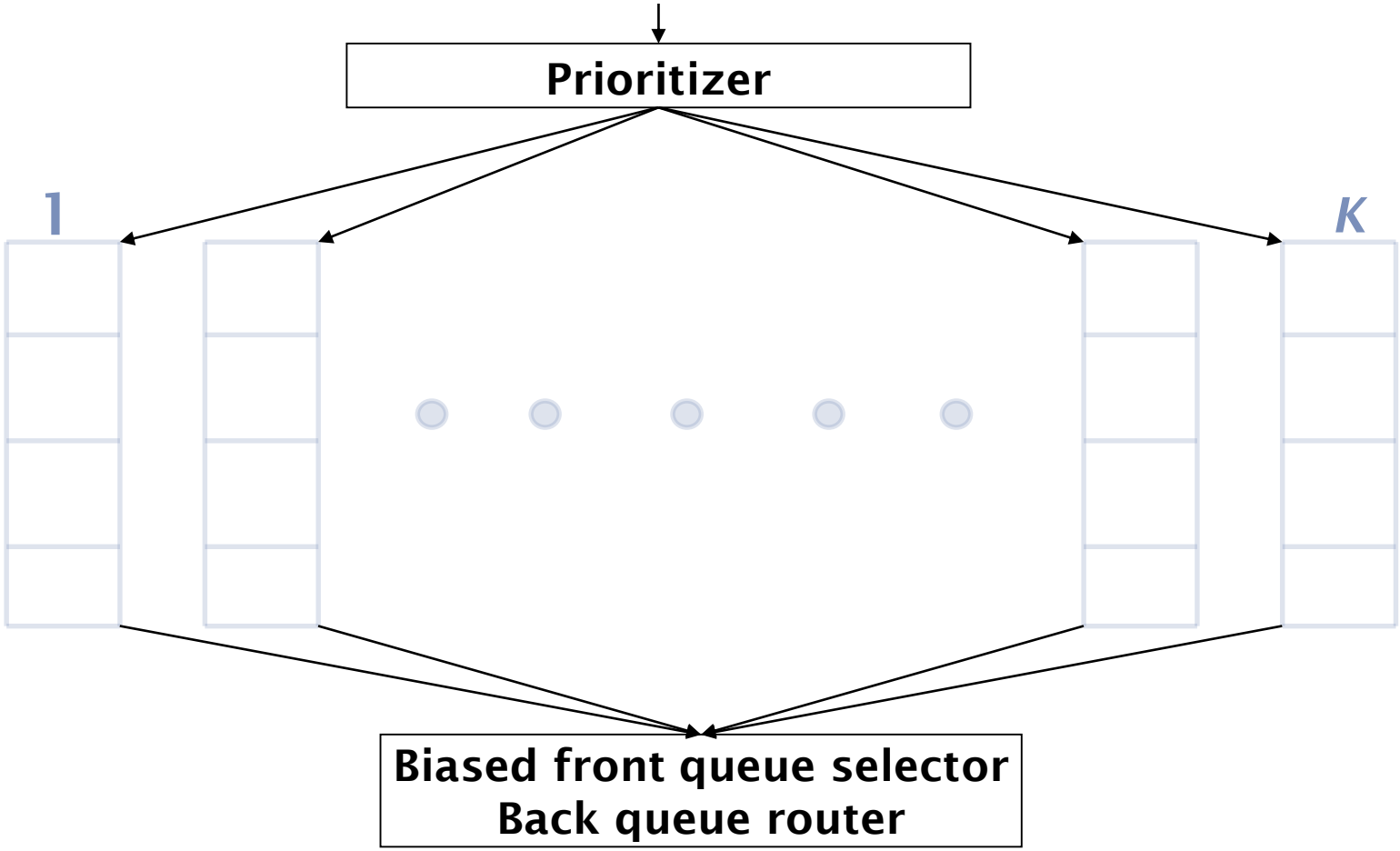
What do we do?

URL frontier: Mercator scheme



- **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")

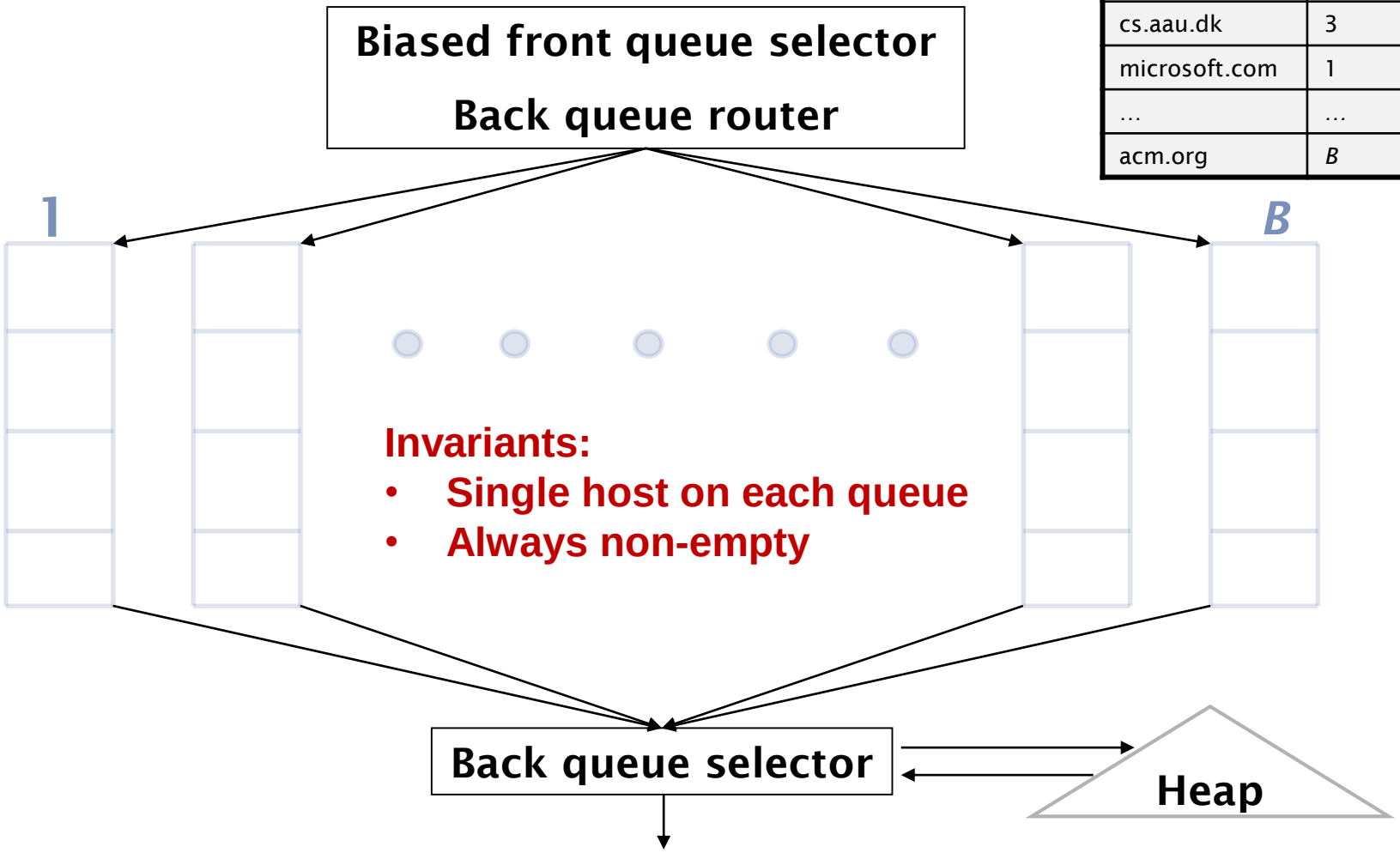
Biased front queue selector

- 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

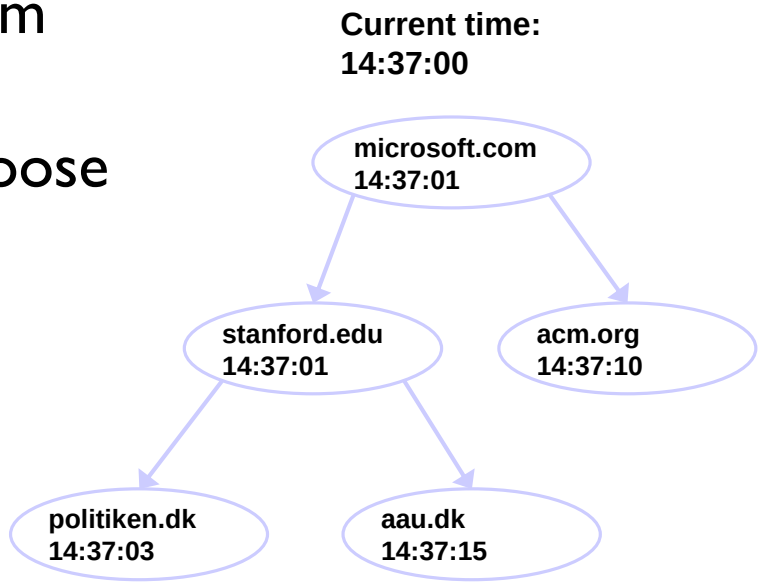
Current host to back queue mapping

Host name	Back queue
cs.aau.dk	3
microsoft.com	1
…	…
acm.org	*B*

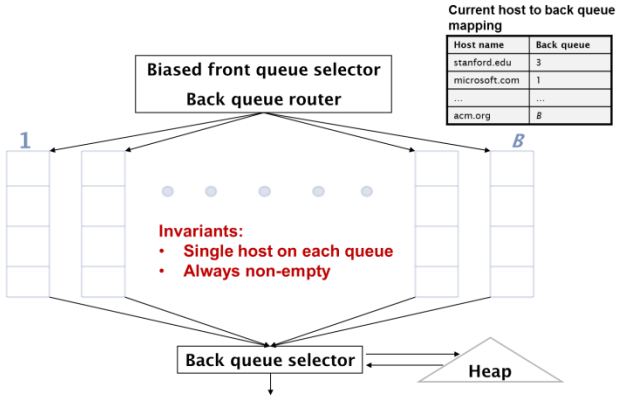


Back queue heap

- One entry for each back queue (host, earliest time)
- The entry is the earliest time 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



1. A crawler thread seeks a URL to crawl:
2. Extracts the root of the heap
3. Fetches URL at head of corresponding back queue q (look up from table)
4. Checks if queue q is now empty – if so, pulls a URL v from front queues
   a. If there's already a back queue for v's host, append v to it and pull another URL from front queues, repeat
   b. Else add v to q
5. When q is non-empty, create heap entry for it

Guidelines for queues

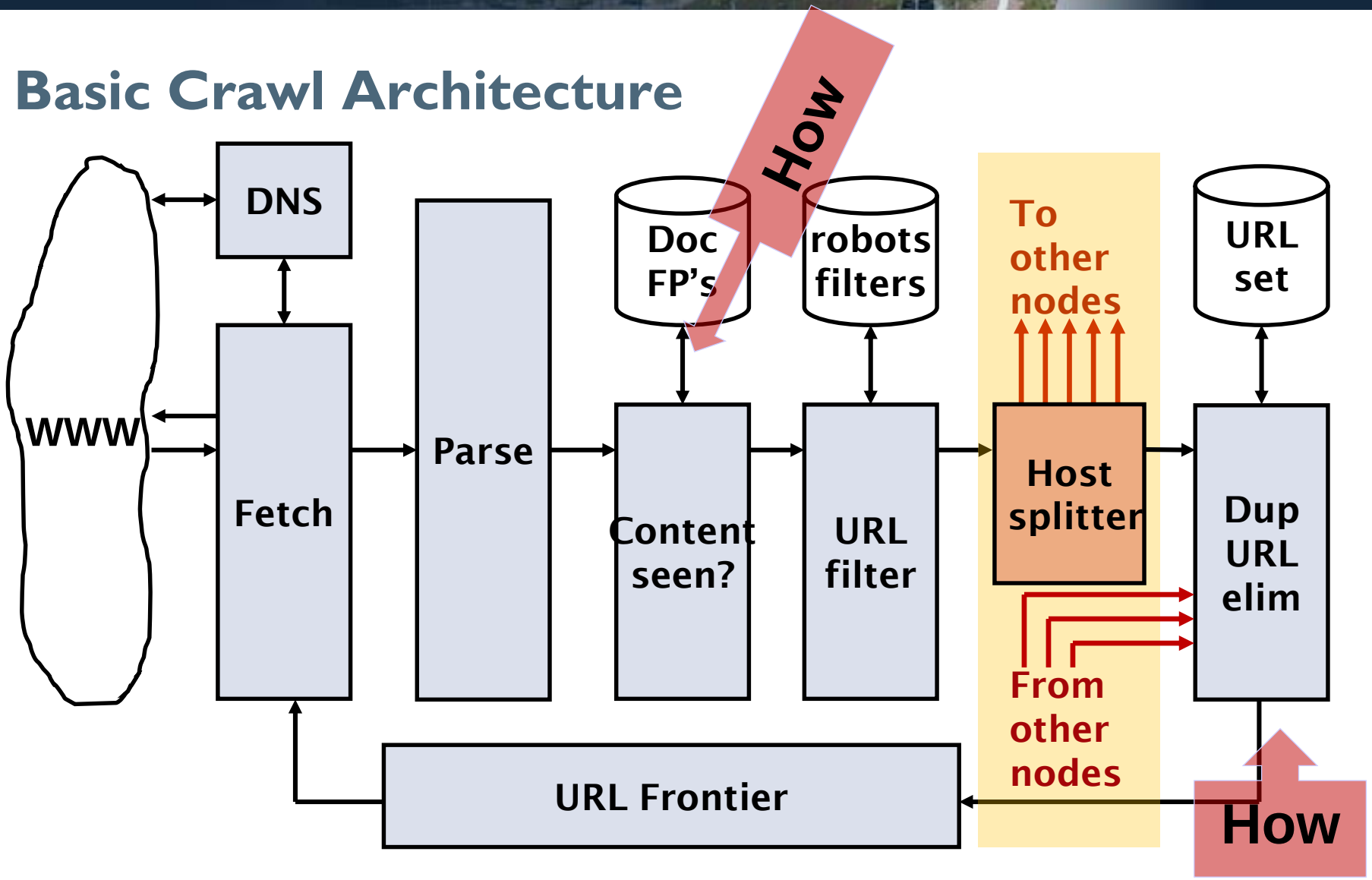
- Keep all threads busy while respecting politeness
  - Number of front queues determines the prioritization
  - Number of back queues determines to which extend can all the crawler threads be busy
- Mercator recommendation: 3x as many back queues as crawler threads
- Priorities in front queues (determined by crawl designer):
  - Refresh rates sampled from previous crawls
  - Domain and application dependent policies (e.g., news hosts first and often)
  - DFS, BFS
  - …

Distributing the crawler

- Run multiple crawl threads, under different processes – potentially at different compute nodes

- Partition hosts being crawled into nodes
  - Hash & modulo
  - Geo-location

Basic Crawl Architecture



Distributing critical tasks

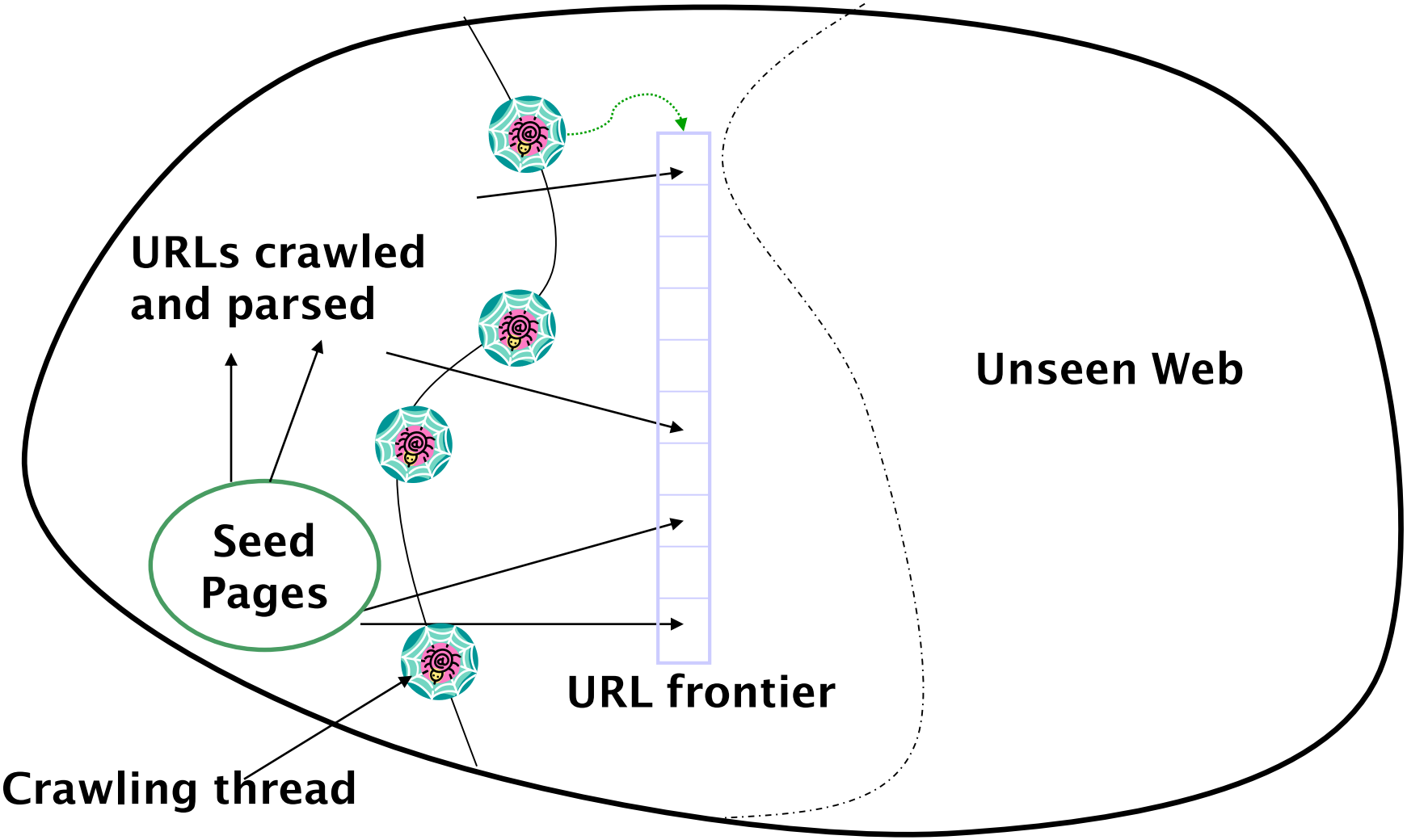
Duplicate URL eliminiation – Easy!

- Deterministic partitioning of hosts into compute and storage nodes

”Content seen?” determination – Difficult!

- No locality in Fingerprints/shingles → similar or slightly updated docs may reside on very different compute and storage nodes
- Results in many remote procedure calls (expensive)
- Mitigations:
  - batch lookup requests
  - Save FP/shingle with URL in the frontier

Updated crawling picture



Outline

- Search engine architecture (recap)
- Basic crawl architecture
  - Politeness (robots.txt)
  - Crawl priority
  - URL Frontier
  - Specific components in architecture
- Distributed crawl architecture