

CS276

Lecture 14

Crawling and web indexes

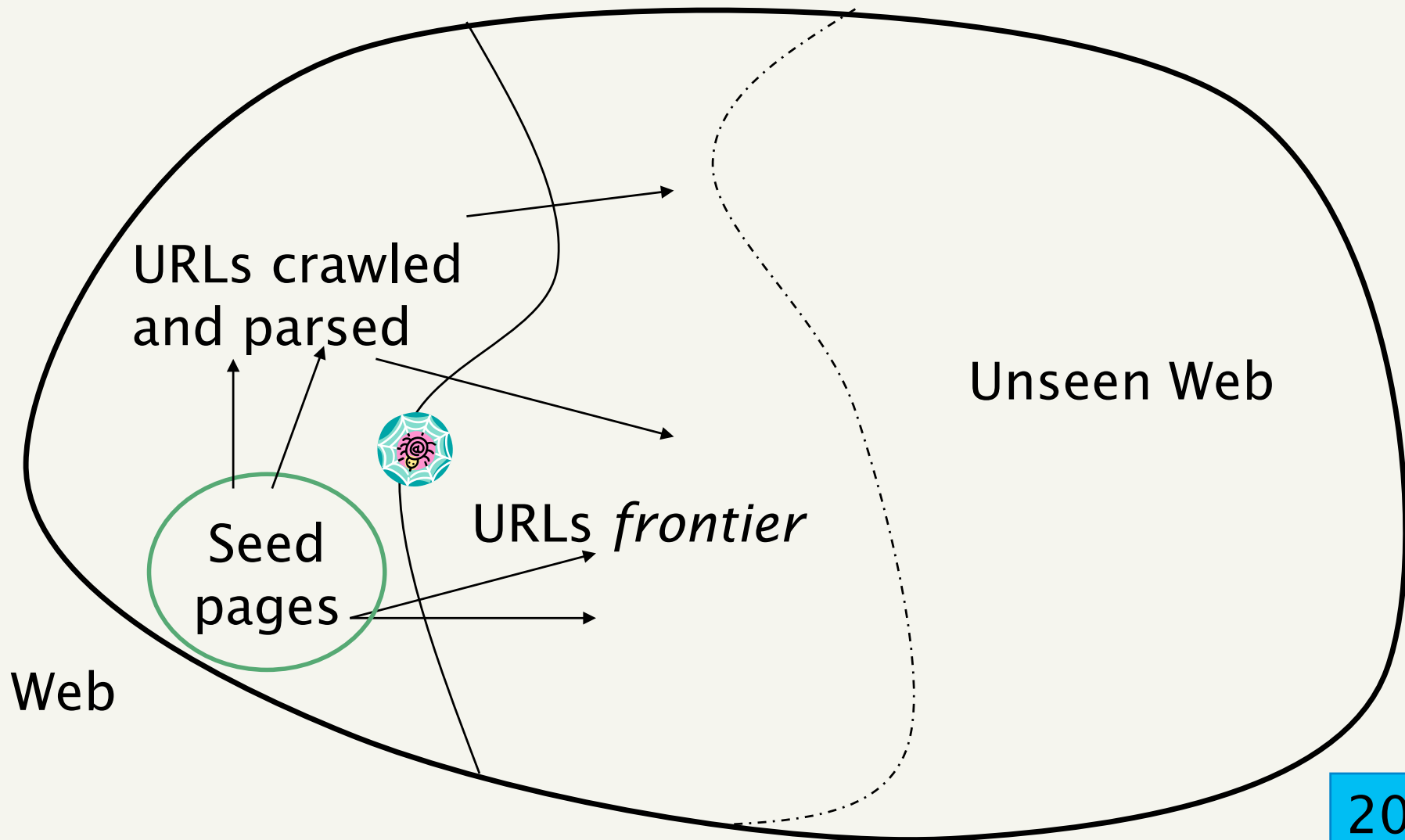
Today's lecture

- Crawling
- Connectivity servers

Basic crawler operation

- Begin with known “seed” pages
- 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
- 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
- Malicious pages
 - Spam pages
 - Spider traps – incl dynamically generated
- 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

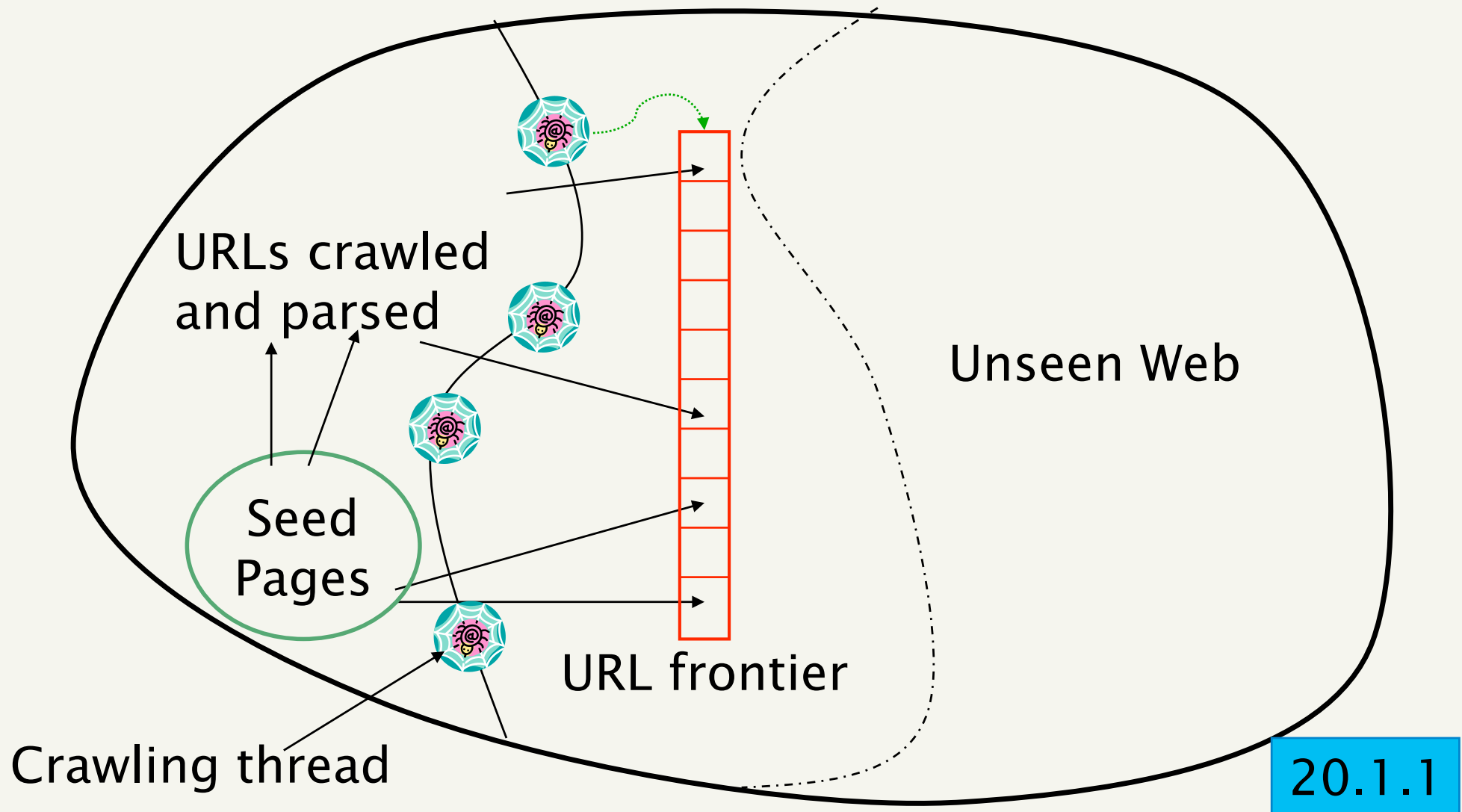
What any crawler *should* do

- 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 URL, create a file `URL/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":

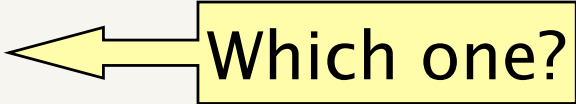
```
User-agent: *
```

```
Disallow: /yoursite/temp/
```

```
User-agent: searchengine
```

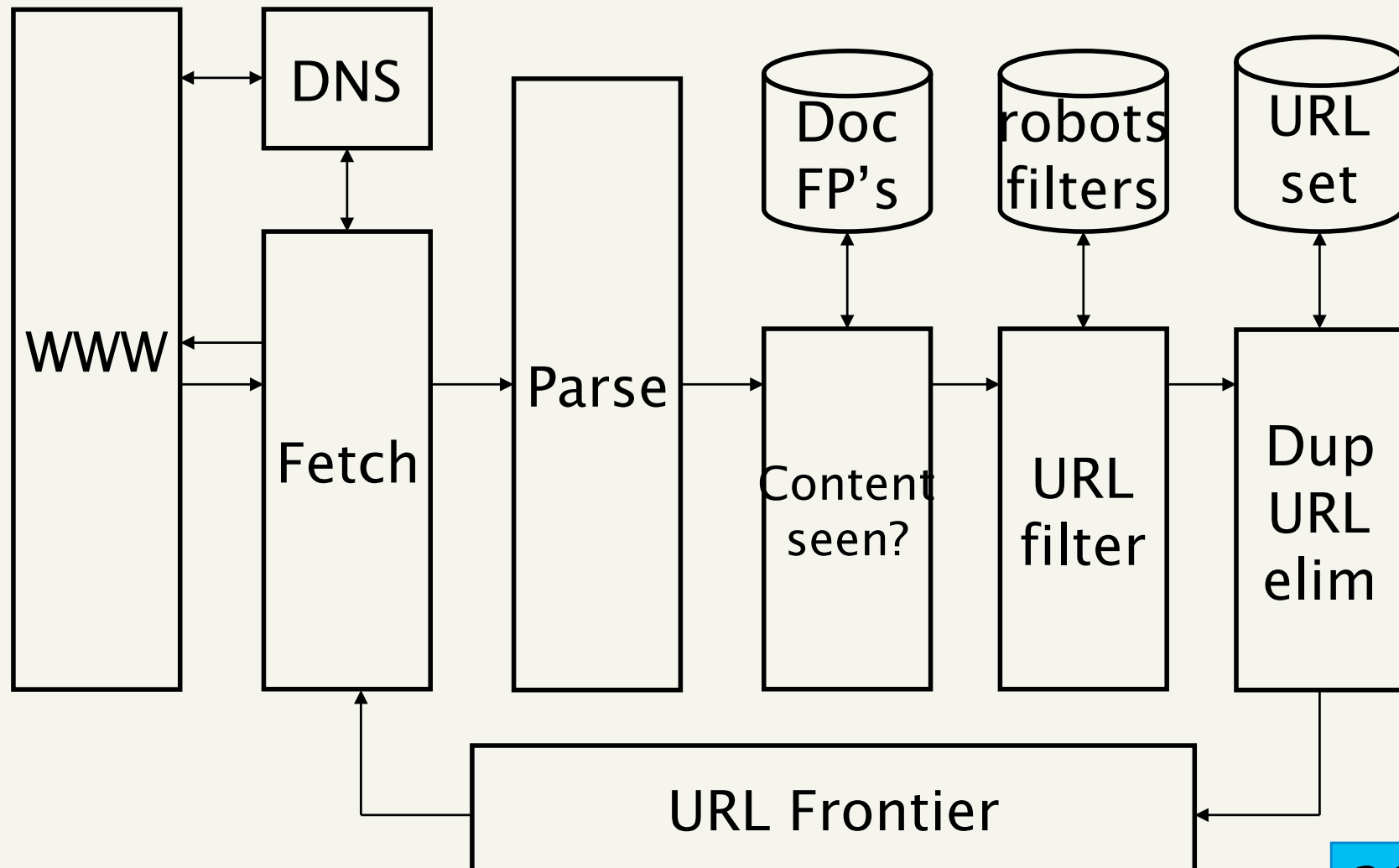
```
Disallow:
```

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)

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

Basic crawl architecture



Parsing: URL normalization

- When a fetched document is parsed, some of the extracted links are *relative* URLs

- E.g., at

http://en.wikipedia.org/wiki/Main_Page

we have 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

Duplicate URL elimination

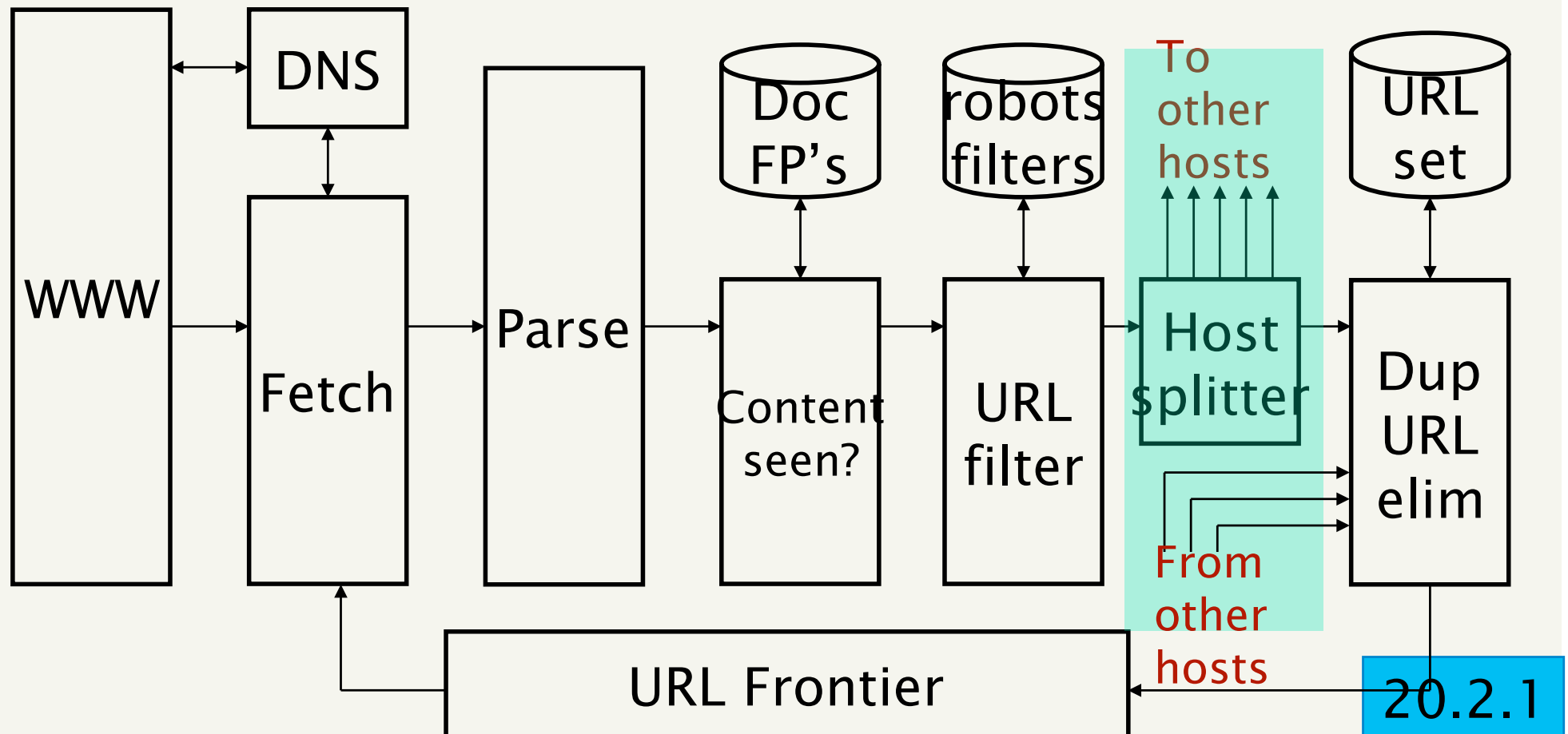
- For a non-continuous (one-shot) crawl, test to see if an extracted +filtered URL has already been passed to the frontier
- For a continuous crawl – see details of frontier implementation

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?

Communication between nodes

- The output of the URL filter at each node is sent to the Duplicate URL Eliminator at all nodes



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 \gg time for most recent fetch from that host