Introduction to Informa I

Lecture 17: Crawling ndexes

Today's lecture

- Crawling

Basic crawler operation

- Begin with known "seed" URLs
- Fetch and parse them

  - Extract URLs they point to

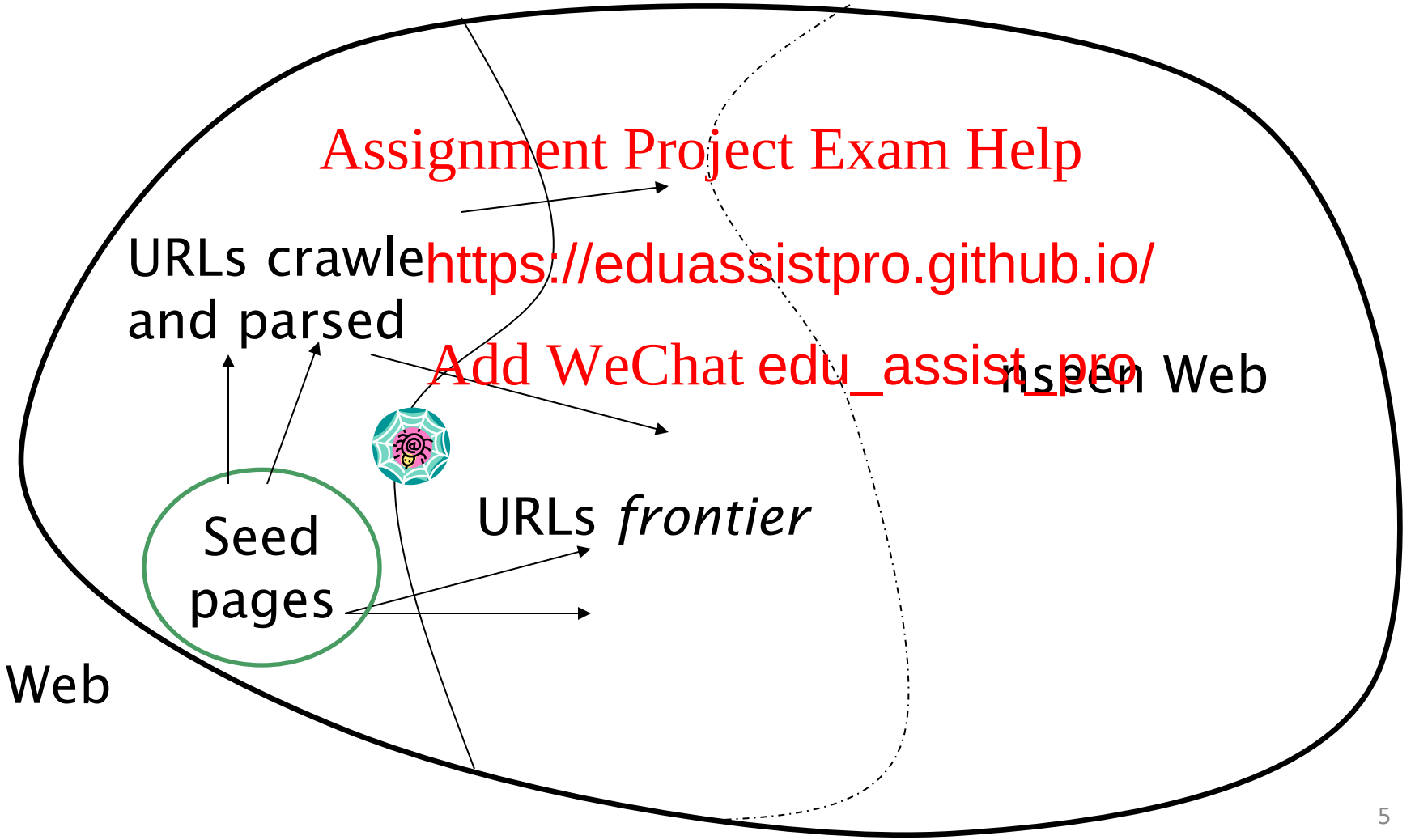
  - Place the extracted URLs on a queue

- Fetch each URL on the queue and repeat

http://www.nature.com/nature/journal/v405/n6783/pdf/405112a0.pdf

Structure of the W a 2000)

Crawling picture



Simple Crawler Thr

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

Simple picture – co ions

- Web crawling isn't feasible with one machine
  - All of the above steps distributed
- Malicious pages
  - Spam pages
  - Spider traps
- Even non-malicious pages p nges
  - 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 any crawler

- Be Polite: Respect implicit and explicit politeness considerations

  - Only cra
  - Respect *r* this shortly)

- Be Robust: Be immune r traps and other malicious behavior from web servers

What any crawler s o

- Be capable of distributed operation: designed to run on multiple distributed machines
- Be scalable: the crawl rate by adding m
- Performance/efficiency: p use of available processing and esources

What any crawler s o

- Fetch pages of “higher quality” first
- Continuous operation: Continue fetching fresh copie tched page
- Extensible: formats, protocols

Freshness

- HTTP protocol has a special request type called HEAD that makes it easy to check for page changes
  - returns information about page, not page itself

Freshness

- Web pages are constantly being added, deleted, and modified
- Web crawler must continually revisit pages it has already crawl changed in order to maintain th ument colle
  - *stale* copies no longer reflect ntents of the web pages

Freshness

- Not possible to constantly check all pages
  - must check important pages and pages that change frequently

- Freshness is t s that are fresh

- Optimizing fo o bad decisions, such as not crawling popul

- *Age* is a better metric

Freshness vs. Age

Age

- Expected age of a page *t* days after it was last crawled:

- Web page upda bution on average

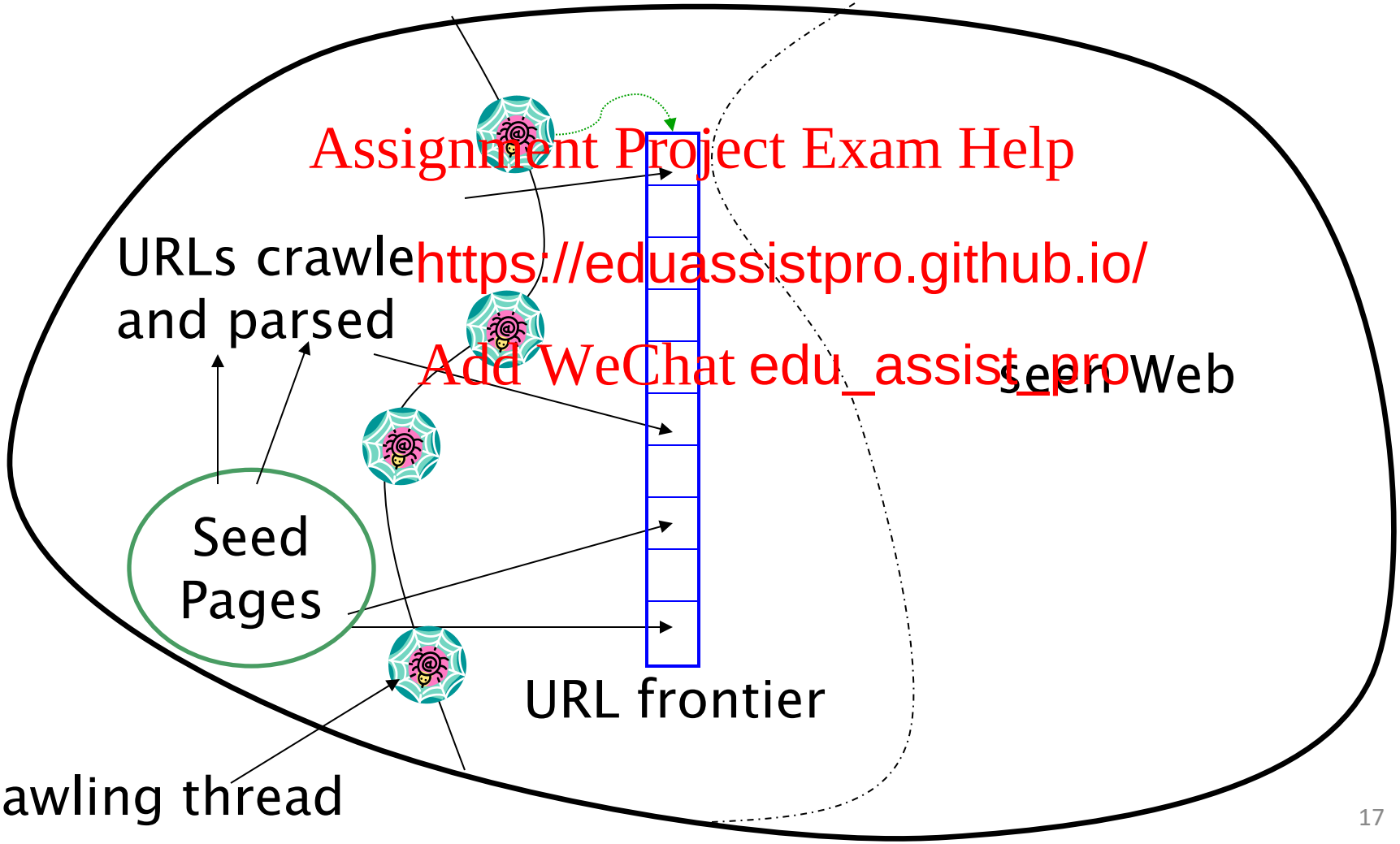
  - time until the next update is governed by an exponential distribution

$$\text{Age}(\lambda, t) = \int_0^t \lambda e^{-\lambda x}(t-x)dx$$

Age

- The older a page gets, the more it costs not to crawl it
  - e.g., expected age with mean change frequency $\lambda = 1/7$ (one change

Updated crawling



Simple Crawler Thr

```
1
2
3   Assignment Project Exam Help
4
5
6         https://eduassistpro.github.io/
7
8         Add WeChat edu_assist_pro
9
10
11
12
13
14
```

URL frontier

- Can include multiple pages from the same host

- Must avoid em all at the same time

- Must try to keep all cr reads busy

Explicit and Implici ess

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

  - robots.txt

- Implicit politeness: ev o 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 ann https://eduassistpro.github.io/ 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:
```

try http://www.taobao.com/robots.txt

Processing steps in g

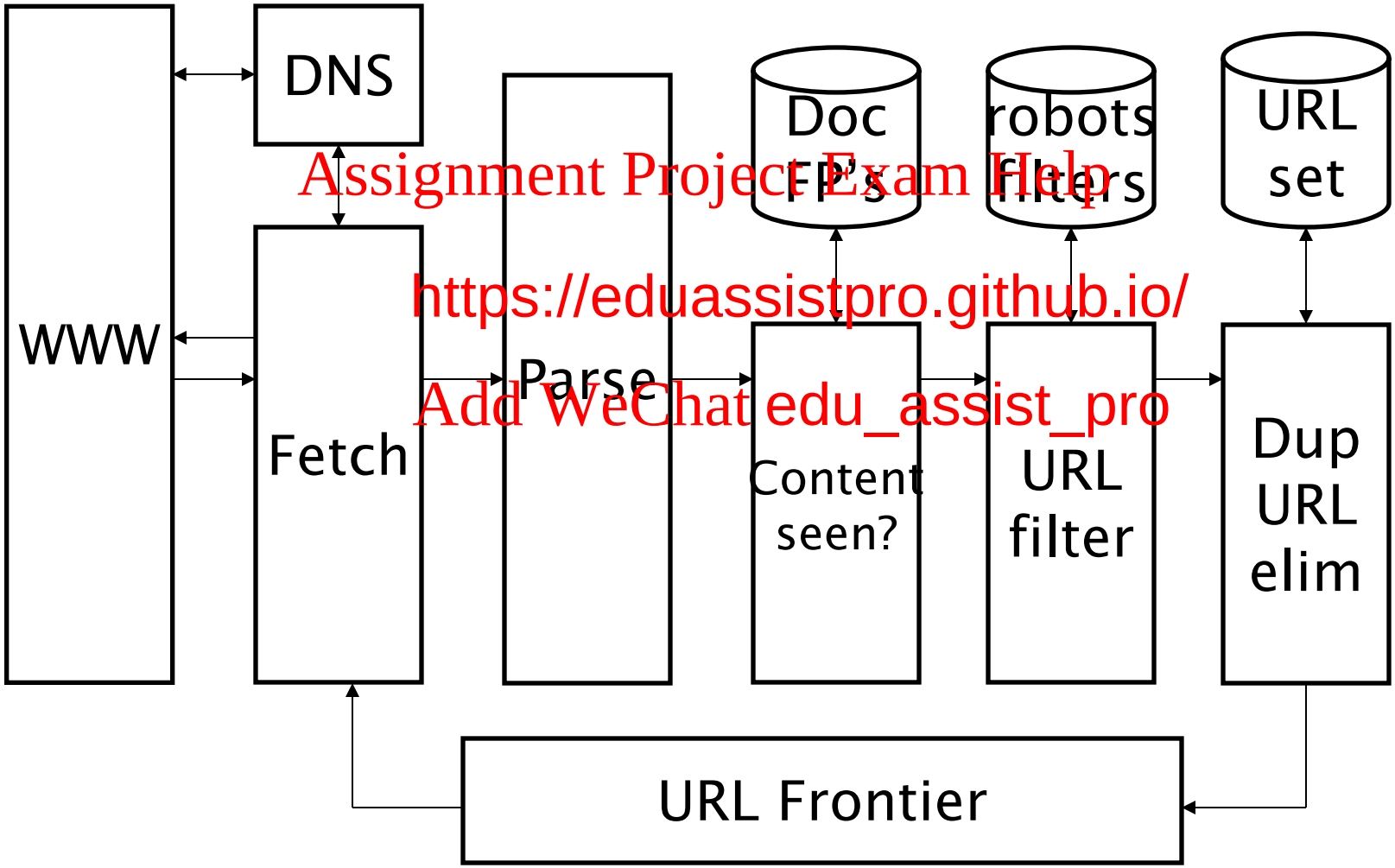
- Pick a URL from the frontier

Which one?

- Fetch the document at the URL
- Parse the URL
  - Extract links
- Check if URL has content al
  - 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 archite



DNS (Domain Name System)

- A lookup service on the internet
  - Given a URL, retrieve its IP address
  - Service provided by a distributed set of servers – thus, lookup laten onds)
- Common OS i S lookup are *blocking*: only one outstan st at a time
- Solutions
  - DNS caching
  - Batch DNS resolver – collects requests and sends them out together

Parsing: URL norm

- When a fetched document is parsed, some of the extracted links are *relative* URLs
- E.g., at http://e ain_Page

we have a relative
/wiki/Wikipedia:General_di hich 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 fetched is already in the index, process it
- This is verified using fingerprints or shingles

Removing Noise

- Many web pages contain text, links, and pictures that are not directly related to the main content of the page

- This additiona *ise* that could negatively aff page

- Techniques have been developed to detect th content blocks in a web page
  - Non-content material is either ignored or reduced in importance in the indexing process

Noise Example

Finding Content Bl

- Cumulative distribution of tags in the example web page

- Main text content of the page corresponds to the "plateau" in the middle of the distribution

Finding Content Bl

- Represent a web page as a sequence of bits, where $b_n = 1$ indicates that the nth token is a tag
- **Optimization problem** where we find values of i and j to maximize gs below i and above j and th een i and j

- i.e., maximize

$$\sum_{n=0}^{i-1} b_n + \sum_{n=i}^{j} (1 - b_n) + \sum_{n=j+1}^{N-1} b_n$$

Finding Content Bl

- Other approaches use DOM structure and visual (layout) features

Filters and robot

- Filters – regular expressions for URL's to be crawled/not

- Once a ro hed from a site, need tedly

  - Doing so burns ban its web server
- Cache robots.txt files

Duplicate URL elim

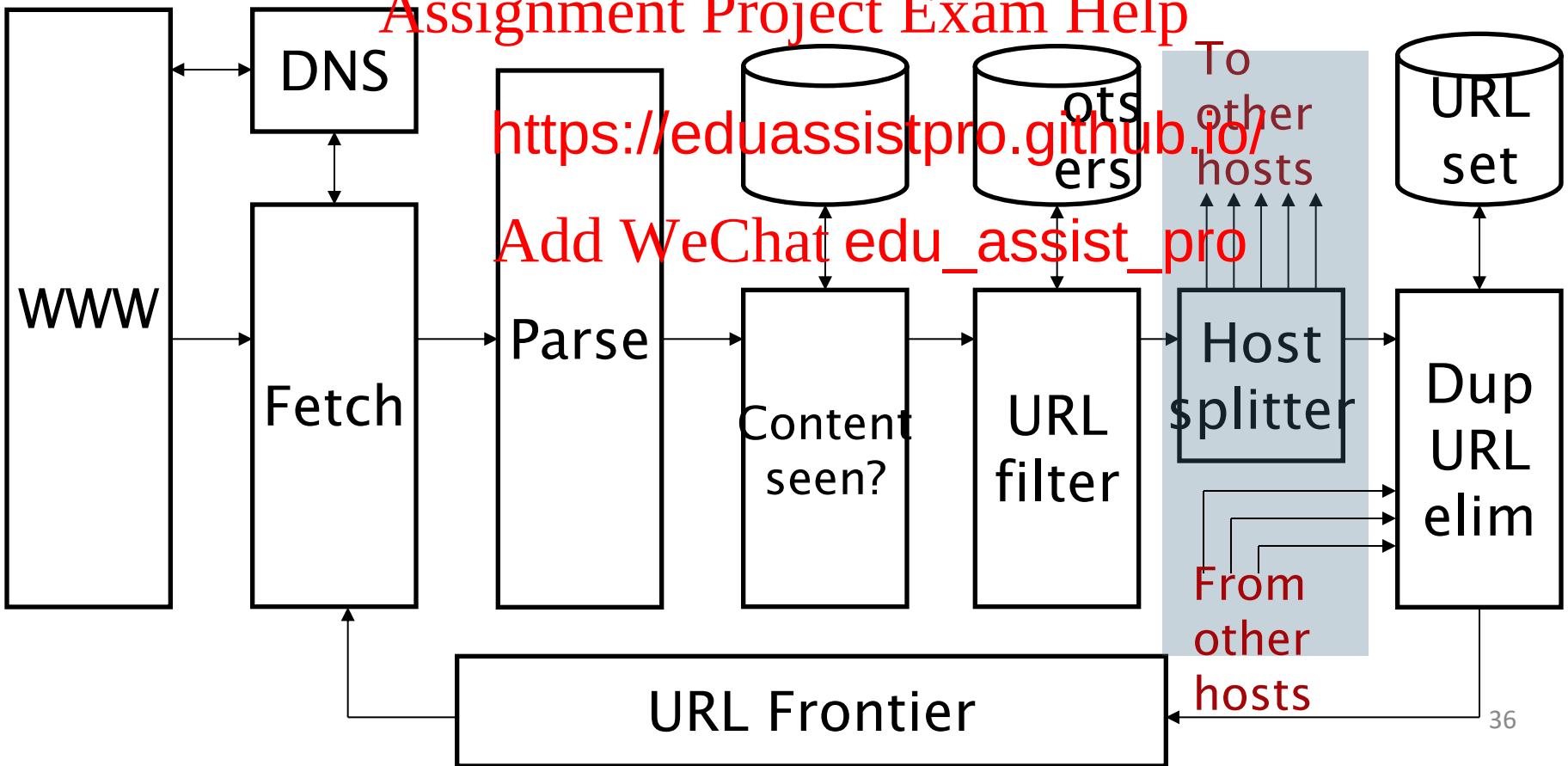
- For a non-continuous (one-shot) crawl, test to see if an extracted+filtered URL has already be ontier
- For a contin et frontier implementati

Distributing the cr

- Run multiple crawl threads, under different processes – potentially at different nodes
  - Geographically distributed nodes
- Partition hos
  - 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 ma derations

- Politeness: do not hit a web server too frequently
- Freshness: crawl some pages more often than others
  - E.g., pages nose content changes often

These goals may conflict ea

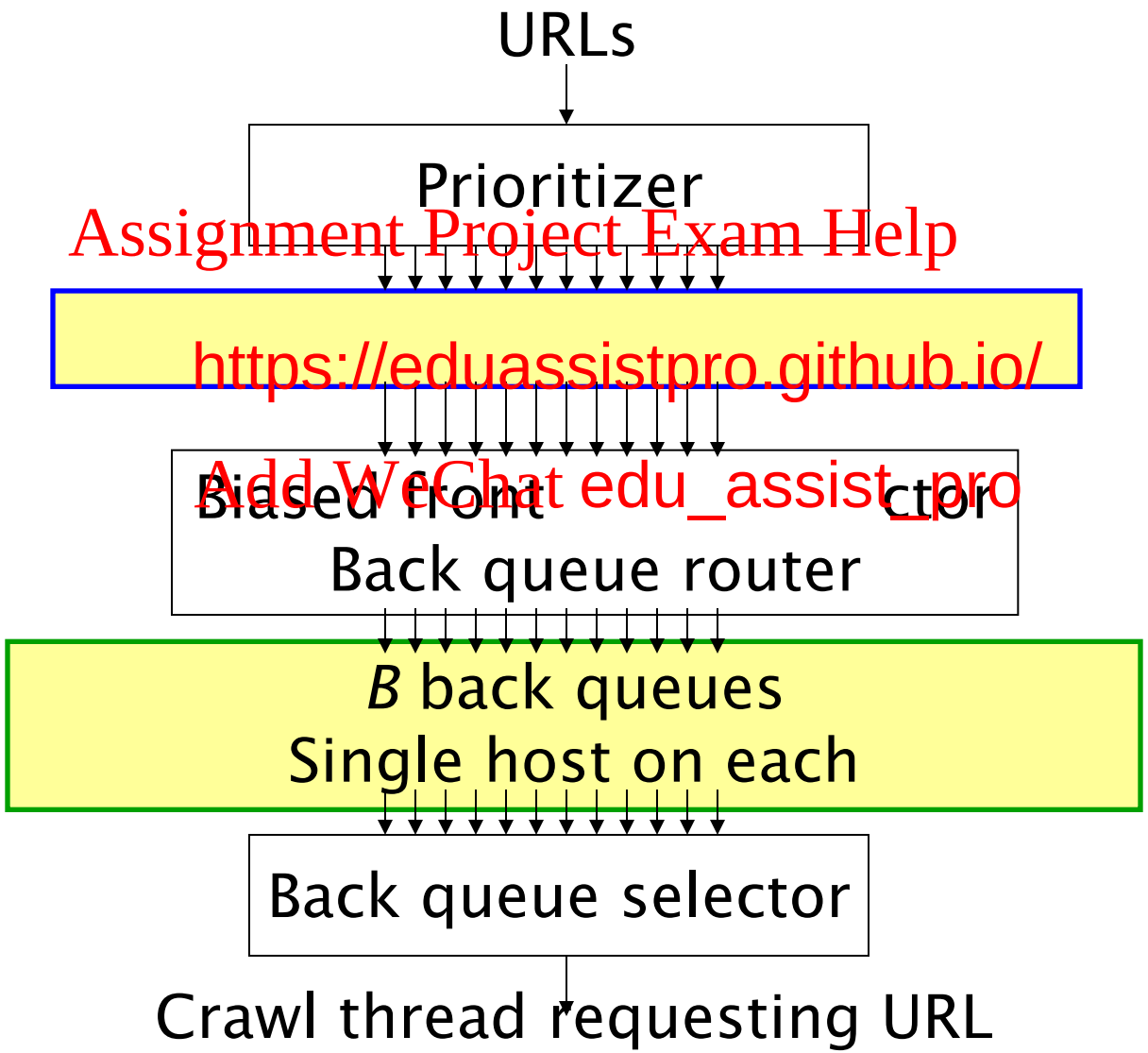
(E.g., simple priority queue fails – many links out of a page go to its own site, creating a burst of accesses to that site.)

Politeness -- challe

- Even if we restrict only one thread to fetch from a host, can hit it repeatedly

- Common h e gap between successive that is >> time for most recent fetch fro st

URL frontier: Mercator scheme

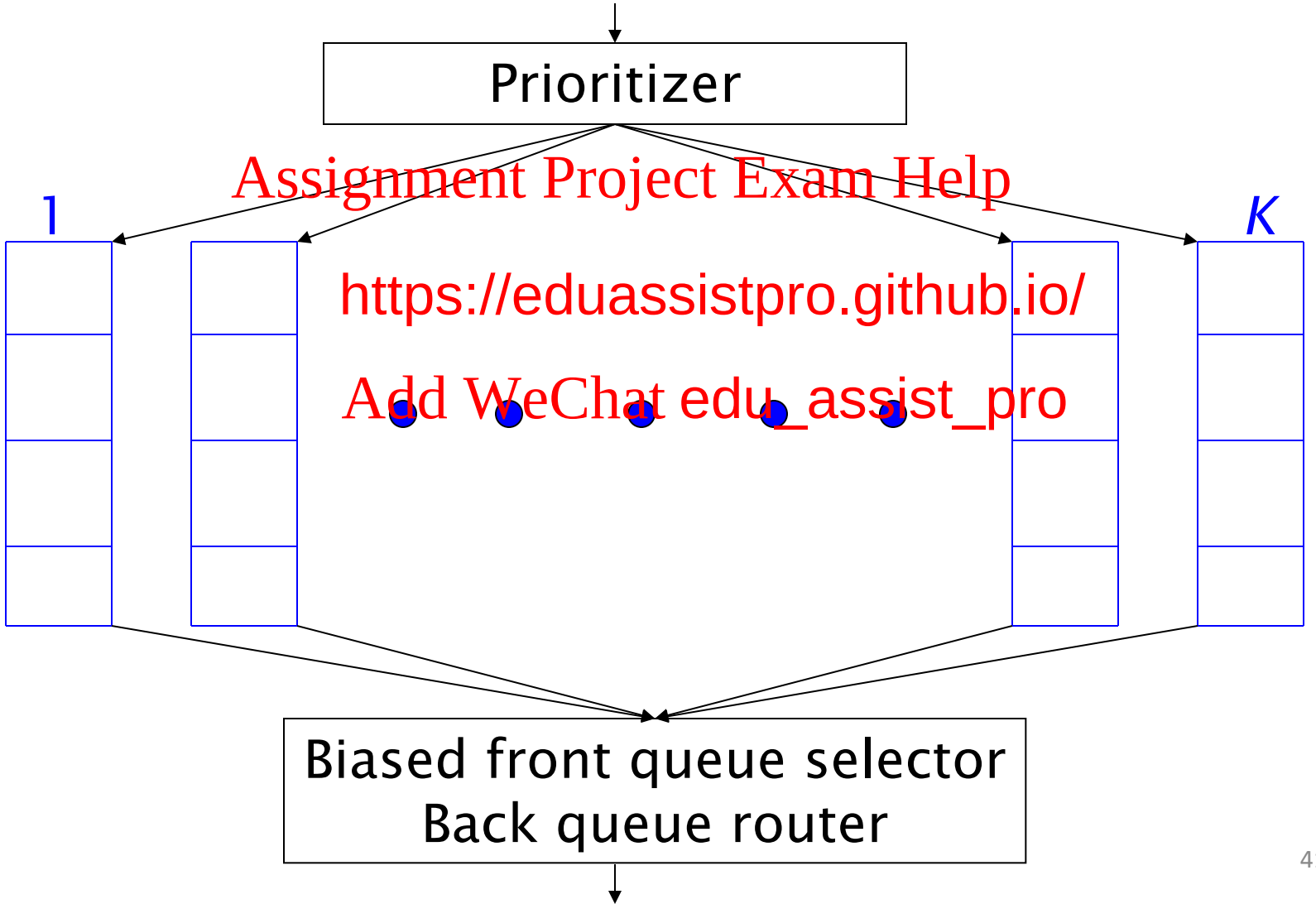


Mercator URL fron

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

- Back queues
- Each queue i

Front queues



Front queues

- Prioritizer assigns to URL an integer priority between 1 and *K*
  - Appends URL to corresponding queue
- Heuristics fo
  - Refresh rate sampled fro
  - Application-specific (e.g., “crawl news sites more often”)

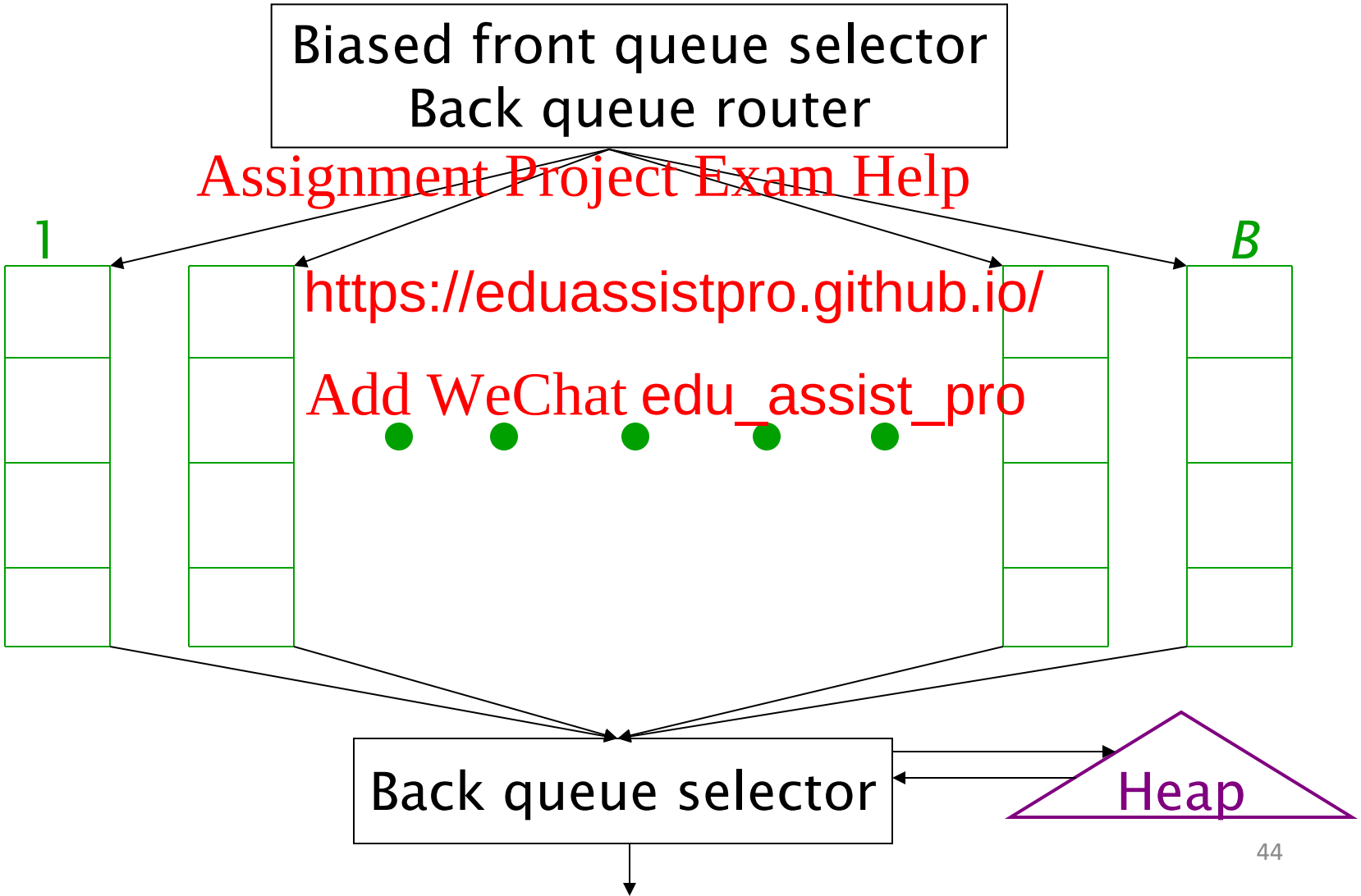
Biased front queue r

- 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 c https://eduassistpro.github.io/ ased to queues of higher priority, or som phisticated variant

  - Can be randomized

Back queues



Back queue invari

- Each back queue is kept non-empty while the crawl is in progress
- Each back RLs from a single host
  - Maintain a table from ck 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 that 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 proces

- A crawler thread seeking a URL to crawl:
- Extracts the root of the heap

- Fetches URL g back queue *q*
(look up fro
- Checks if queue *q* is now so, pulls a URL *v* from front queues
  - If there's 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 qu

- Keep all threads busy while respecting politeness
- Mercator recommendation: three times as many back queues

Resources

- IIR Chapter 20
- Mercator: A scalable, extensible web crawler (Heydon et al. 1999)
- A standard fo