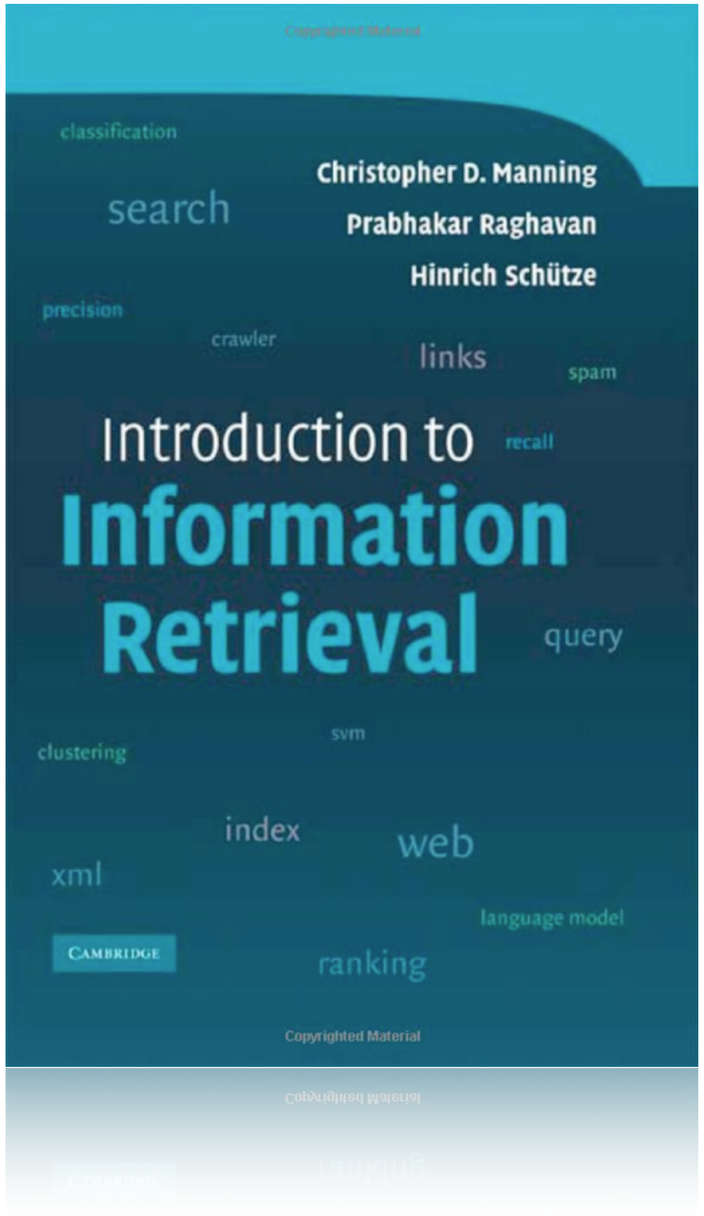
University of East Anglia

Web Crawling

Information Retrieval
CMP-5036A
Sarah Taylor
s.l.taylor@uea.ac.uk

Overview

- Basic crawler operation
- Politeness
- Architecture
- Crawling strategies - breadth and depth-first crawling
- Distributed crawlers
- Implementation Issues
- Crawling in Python



Remember to refer to the course text

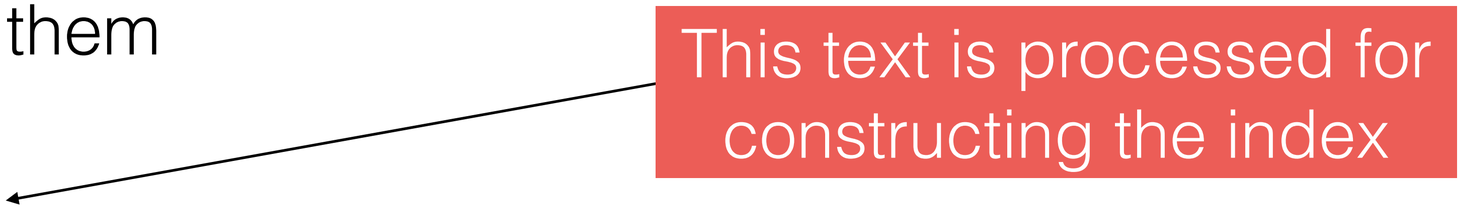
It's **freely** available for download at https://nlp.stanford.edu/IR-book

What is a Web Crawler?

- Also known as a web spider or robot
- Automated script that browses the web collecting useful webpages and the link structure that interconnects them
- Legitimate uses:
  - Search engines automatically index the internet using crawlers
  - Automatic maintenance of web pages (link checking, HTML validation etc.)
- Illegitimate uses include crawlers that harvest email address for spam

Basic Crawler Operation

- Begin with known “seed” URLs
- Fetch and parse them
  - Extract text
  - Extract URLs that the page points to
  - Place URLs in a queue (called the *URL Frontier*)
- Fetch each URL in the URL Frontier and repeat



Features that a Crawler Must Have

- **Robustness to *Spider traps***
    - *Spider traps* are web pages (not always malicious) that make the crawler get stuck fetching an infinite number of pages in a particular domain.
- **Politeness**
    - Crawlers should respect both implicit and explicit policies set out by the web server regulating the rate at which they can be visited

Politeness

- **Explicit Politeness**: specifications from webmasters on what portions of site can be crawled. The can set out instructions in:
    - robots.txt
    - HTML <META> tag
- **Implicit Politeness**: even with no specification, a crawler should avoid hitting any site too often
    - Only one connection should be open to a host at a time
    - Some time should occur between successive requests to a host (Google defaults to a rate of 5 requests per second)

Robots.txt

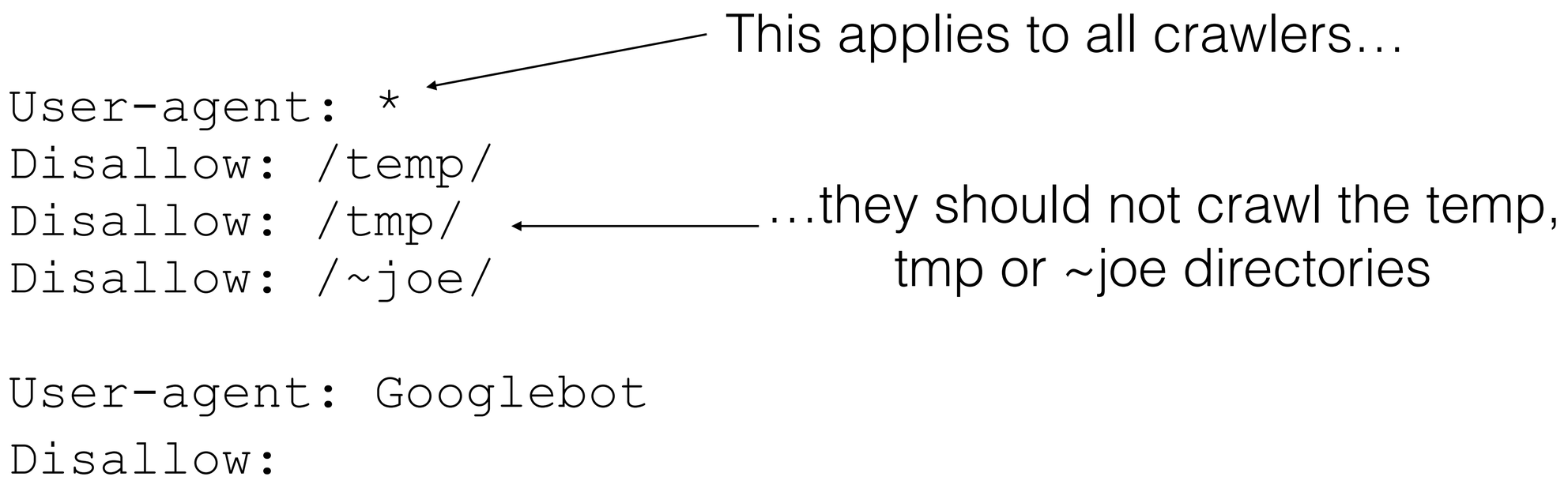
- Protocol for giving spiders (“robots”) limited access to a website, originally from 1994
    - http://www.robotstxt.org/robotstxt.html
- Placed in a website’s top-level directory
- Contains a request on what files and directories can(not) be crawled by certain spiders

Robots.txt (example)


```
User-agent: *
Disallow: /temp/
Disallow: /tmp/
Disallow: /~joe/

User-agent: Googlebot
Disallow:
```

Robots.txt (example)



Robots.txt (example)

```
User-agent: *
Disallow: /temp/
Disallow: /tmp/
Disallow: /~joe/

User-agent: Googlebot
Disallow:
```

This applies to Google robots…

Robots.txt (example)

```
User-agent: *
Disallow: /temp/
Disallow: /tmp/
Disallow: /~joe/

User-agent: Googlebot
Disallow:
```

This applies to Google robots…

… allow access to the entire site
(this overrides the previous statement)

User Agents

http://www.robotstxt.org/db.html

HTML <META> tag

```
<html>
<head>
<title>...</title>
<META NAME="ROBOTS" CONTENT="NOINDEX, NOFOLLOW">
</head>
...
```

HTML <META> tag

```
<html>
<head>
<title>...</title>
<META NAME="ROBOTS" CONTENT="NOINDEX, NOFOLLOW">
</head>
…
```

Specifies whether crawlers should index the page.
Options: INDEX (default), NOINDEX

HTML <META> tag

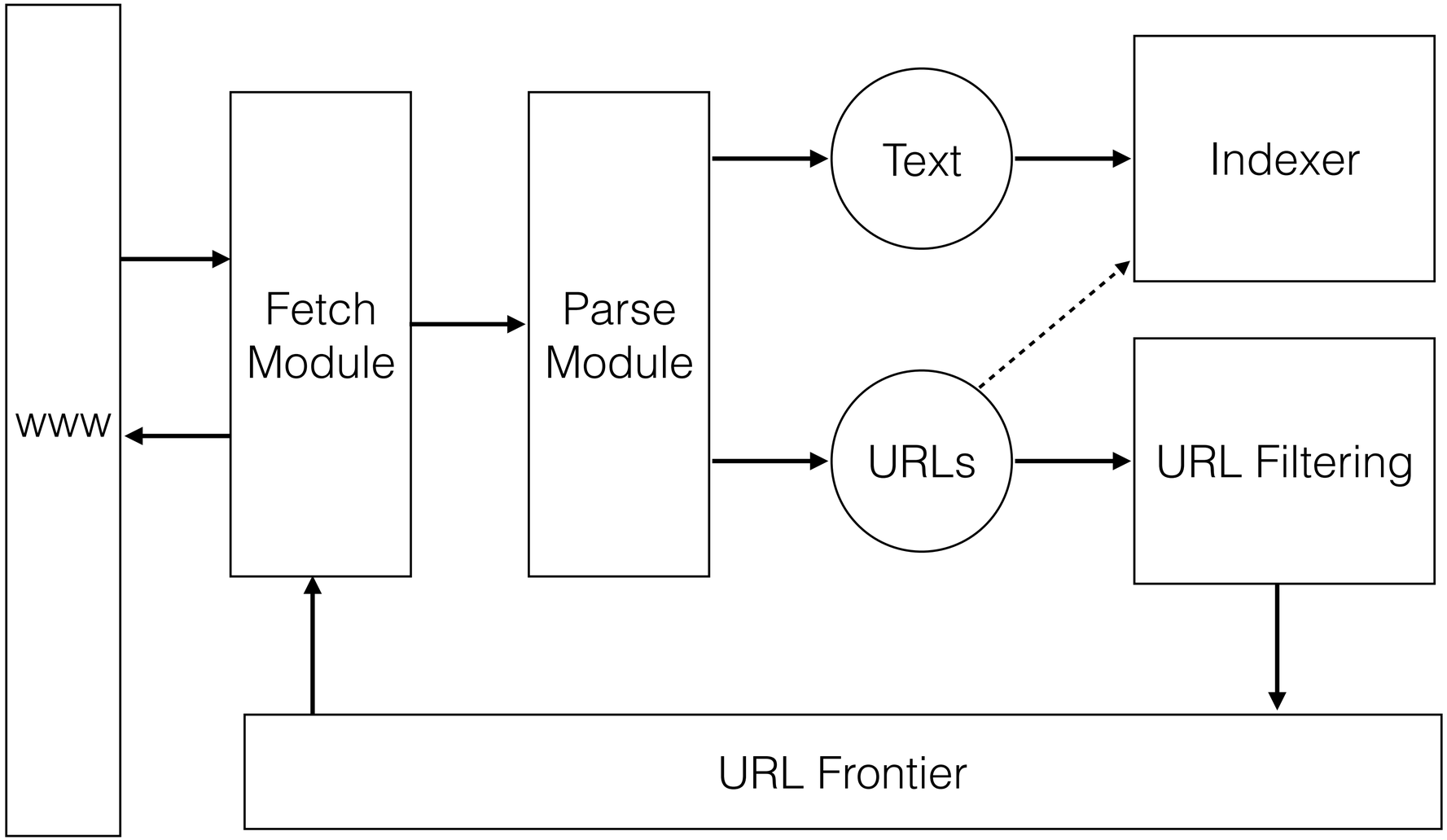
```
<html>
<head>
<title>...</title>
<META NAME="ROBOTS" CONTENT="NOINDEX, NOFOLLOW">
</head>
…
```

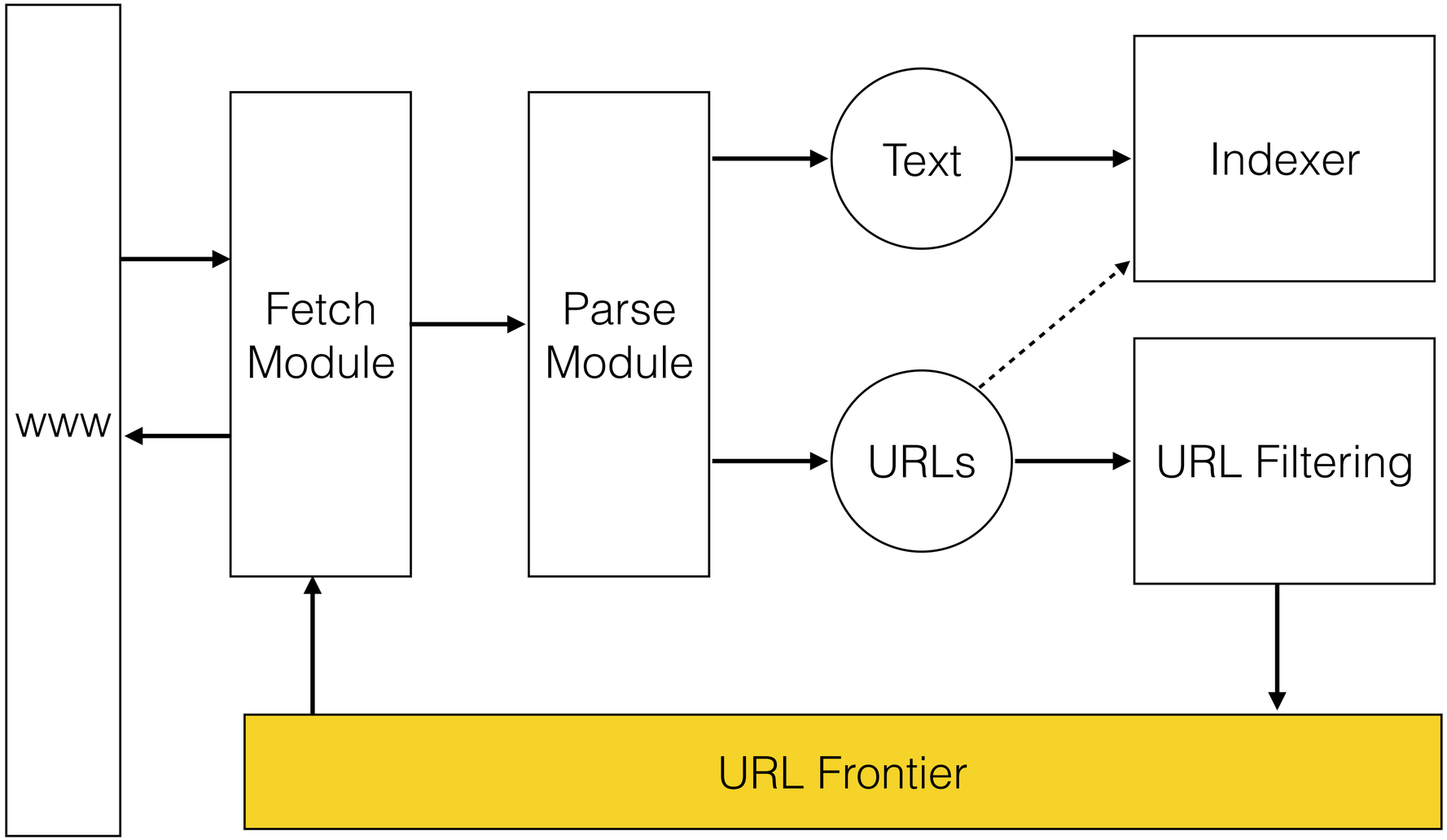
Specifies whether crawlers should crawl links from the page.
Options: FOLLOW (default), NOFOLLOW

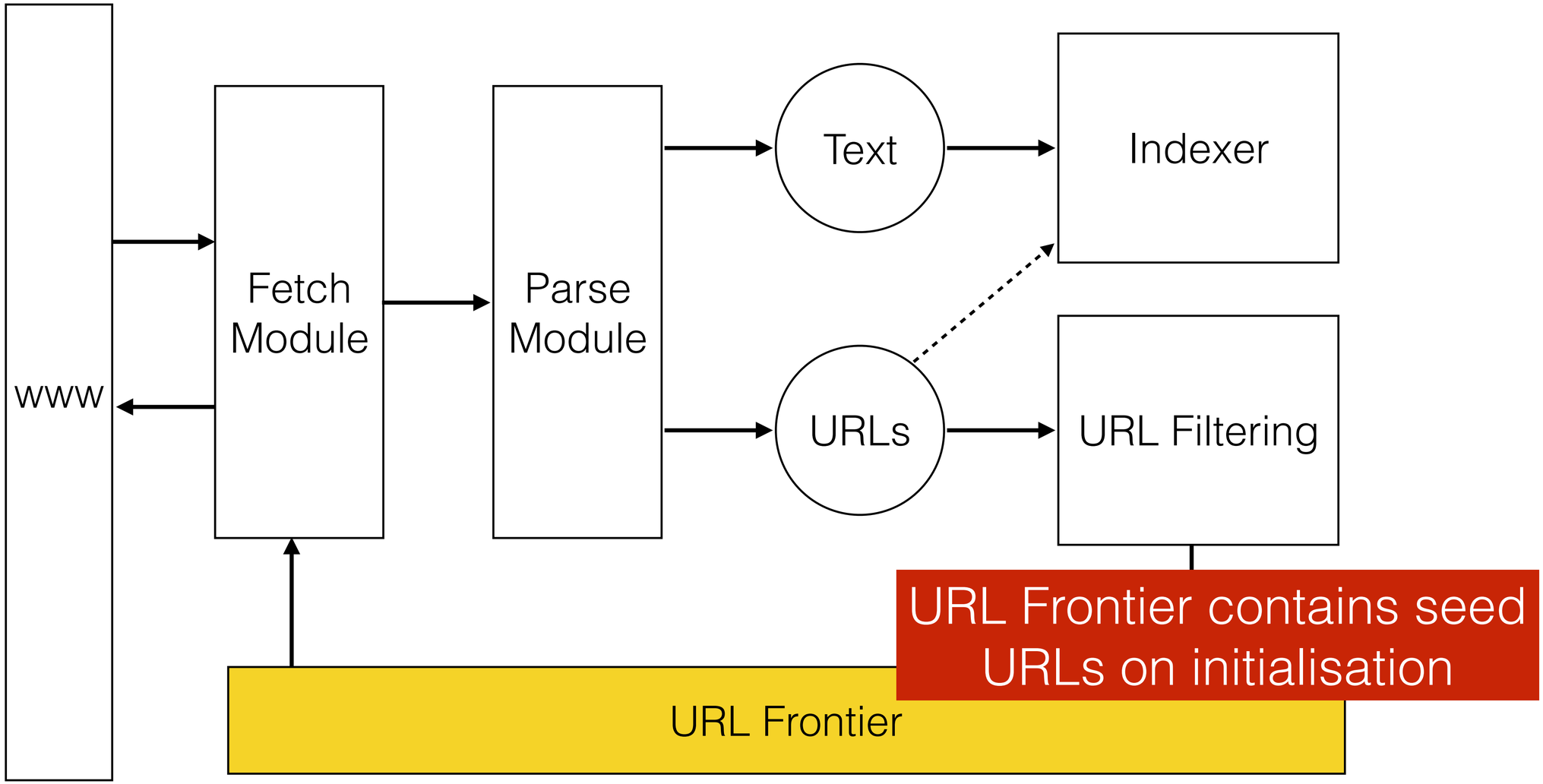
Why Block Web Crawlers?

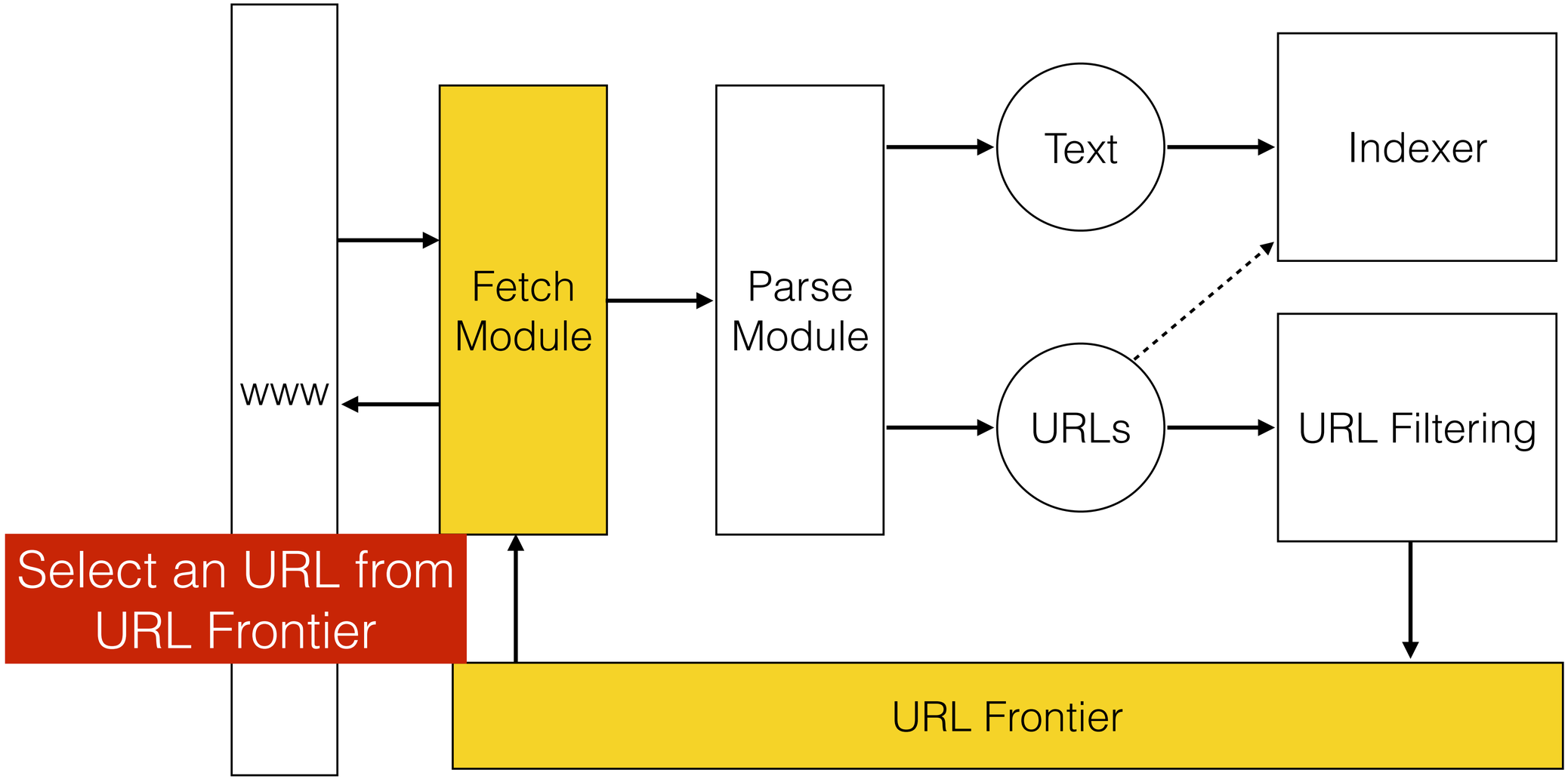
- Search results
- Test/incomplete sites
- Private content
- Automatically generated/dynamic content

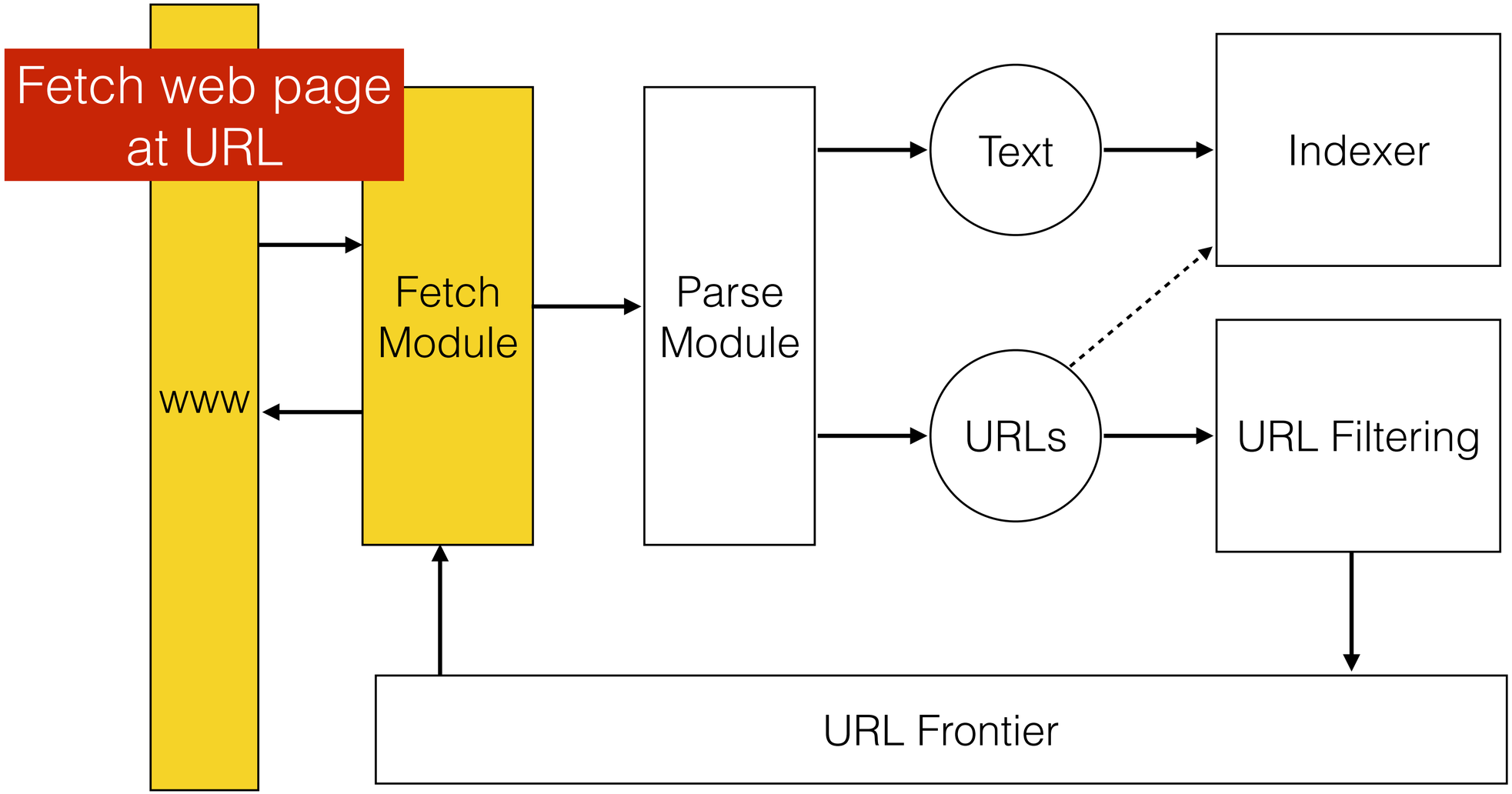
Crawler Architecture

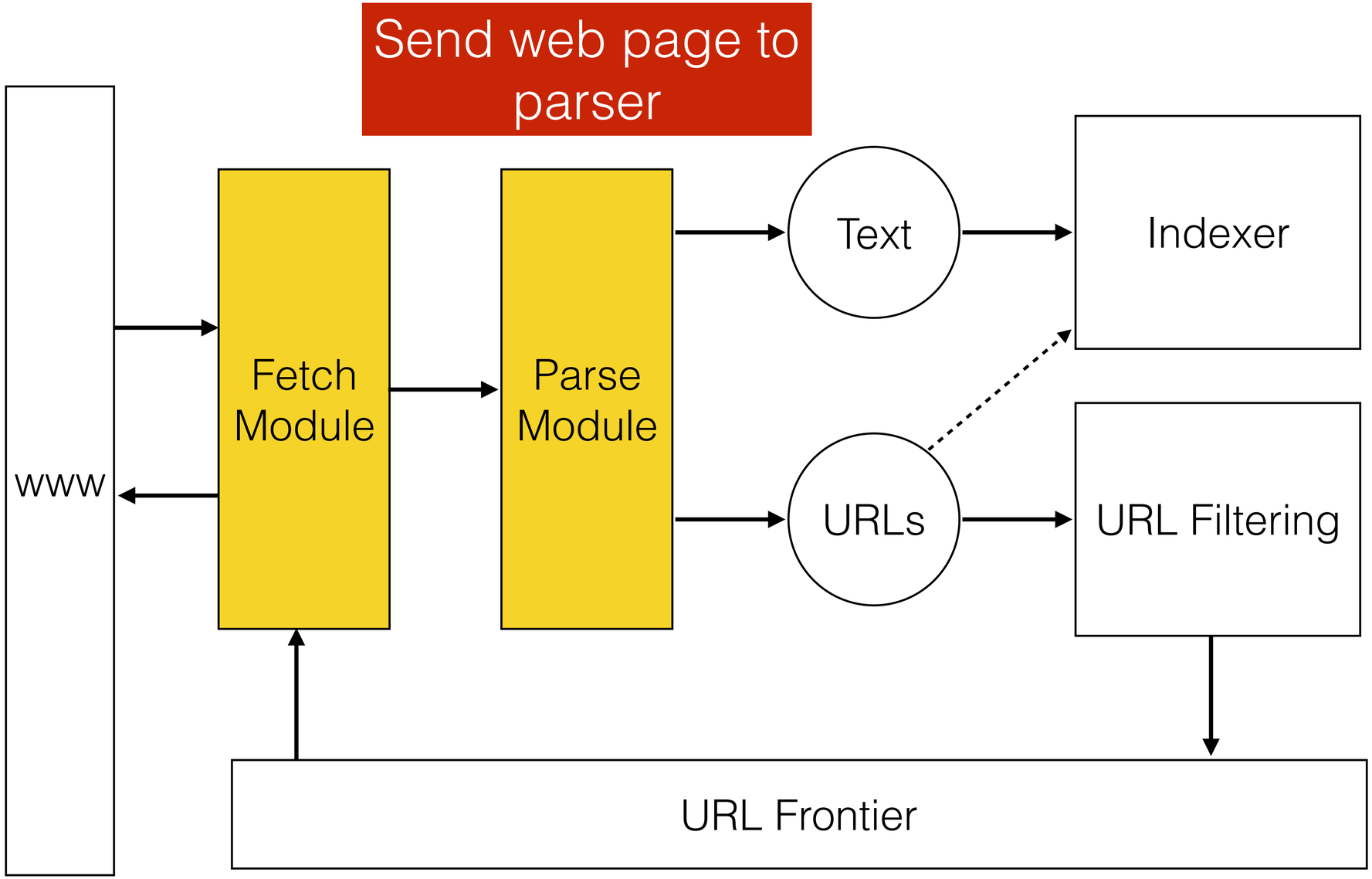

www
Fetch Module
Parse Module
Text
Indexer
URLs
URL Filtering
URL Frontier

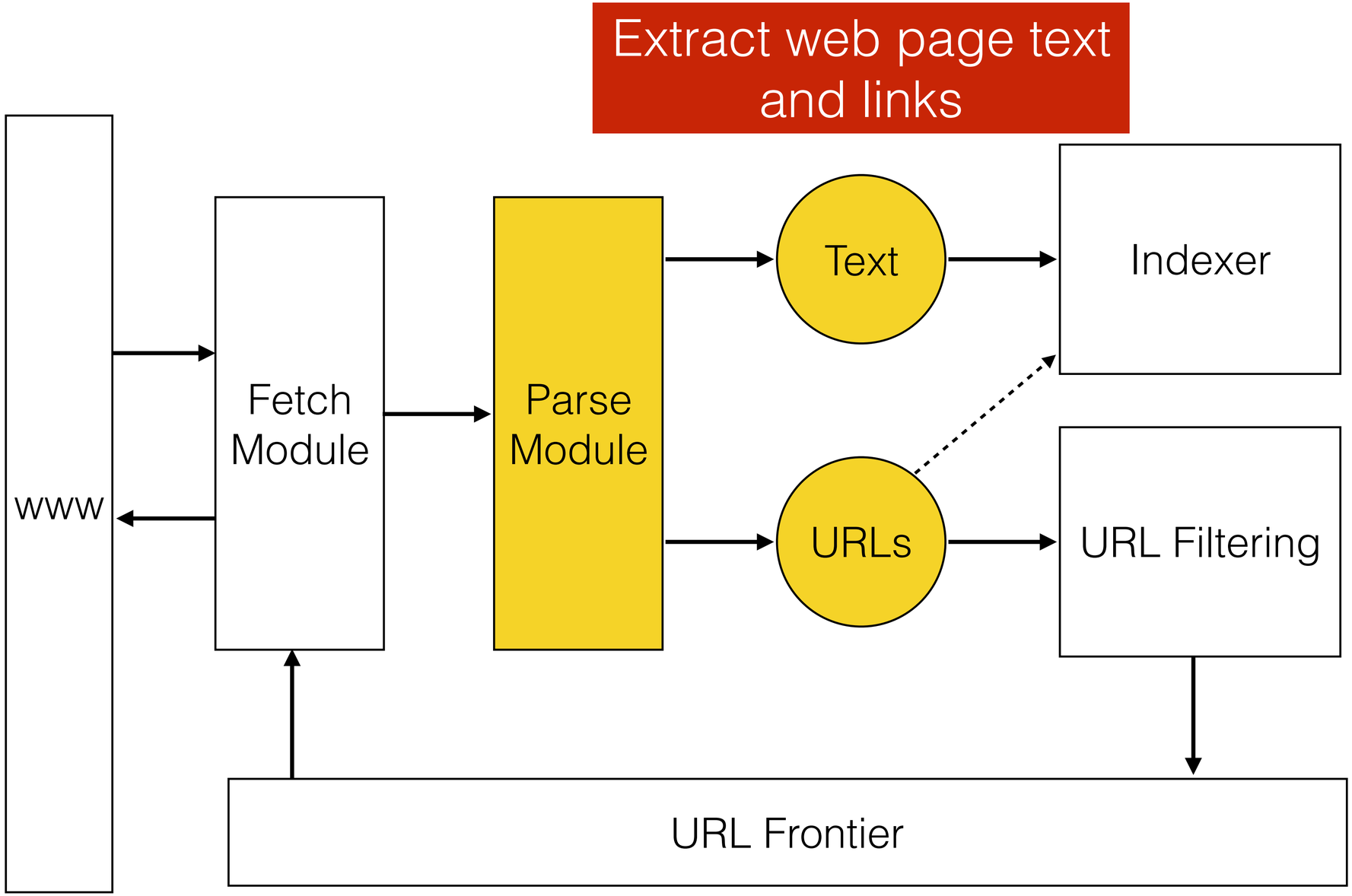

www
Fetch Module
Parse Module
Text
Indexer
URLs
URL Filtering
URL Frontier


www
Fetch Module
Parse Module
Text
URLs
Indexer
URL Filtering
URL Frontier contains seed URLs on initialisation
URL Frontier

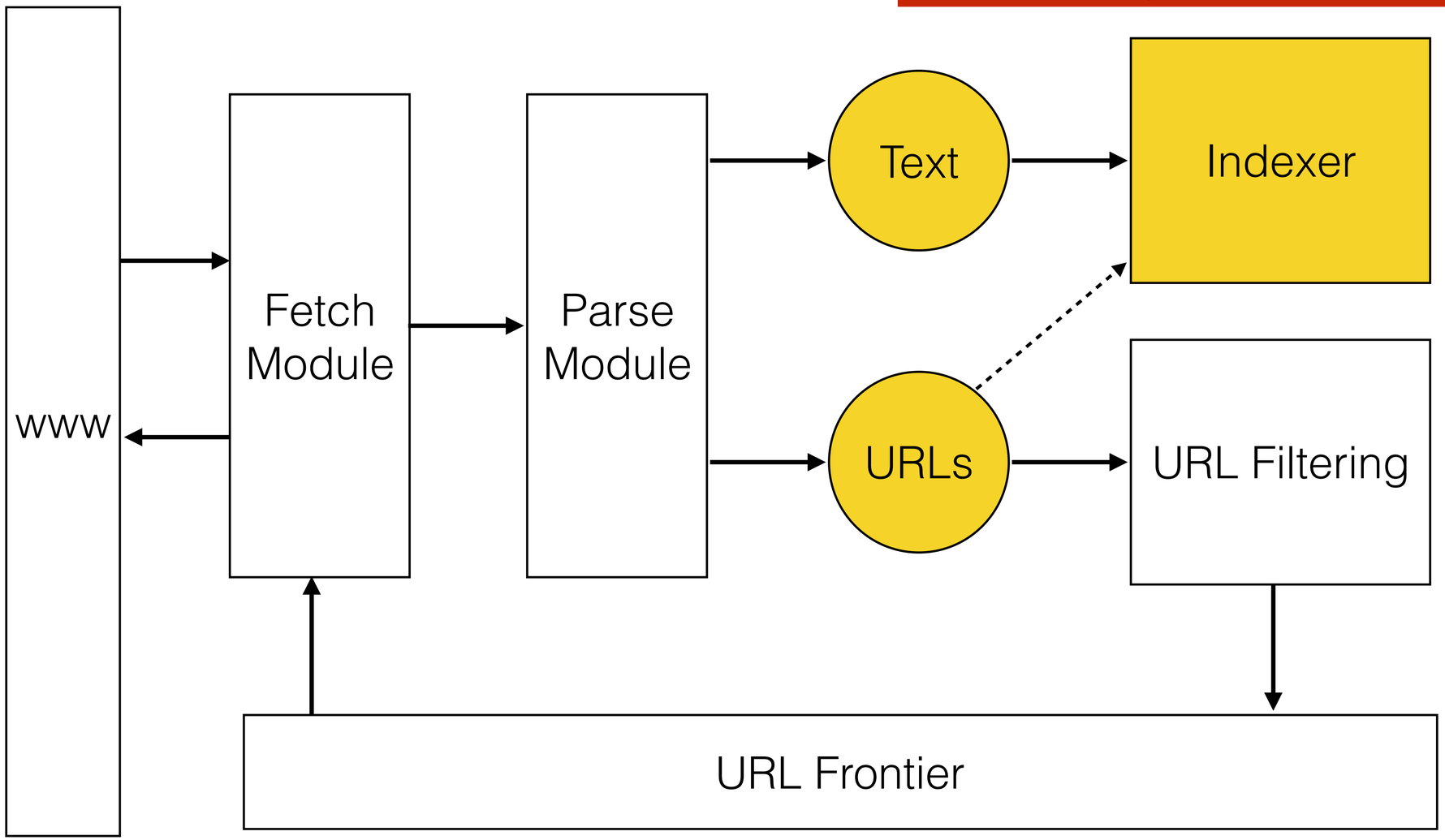

www
Fetch Module
Parse Module
Text
Indexer
URLs
URL Filtering
Select an URL from URL Frontier
URL Frontier

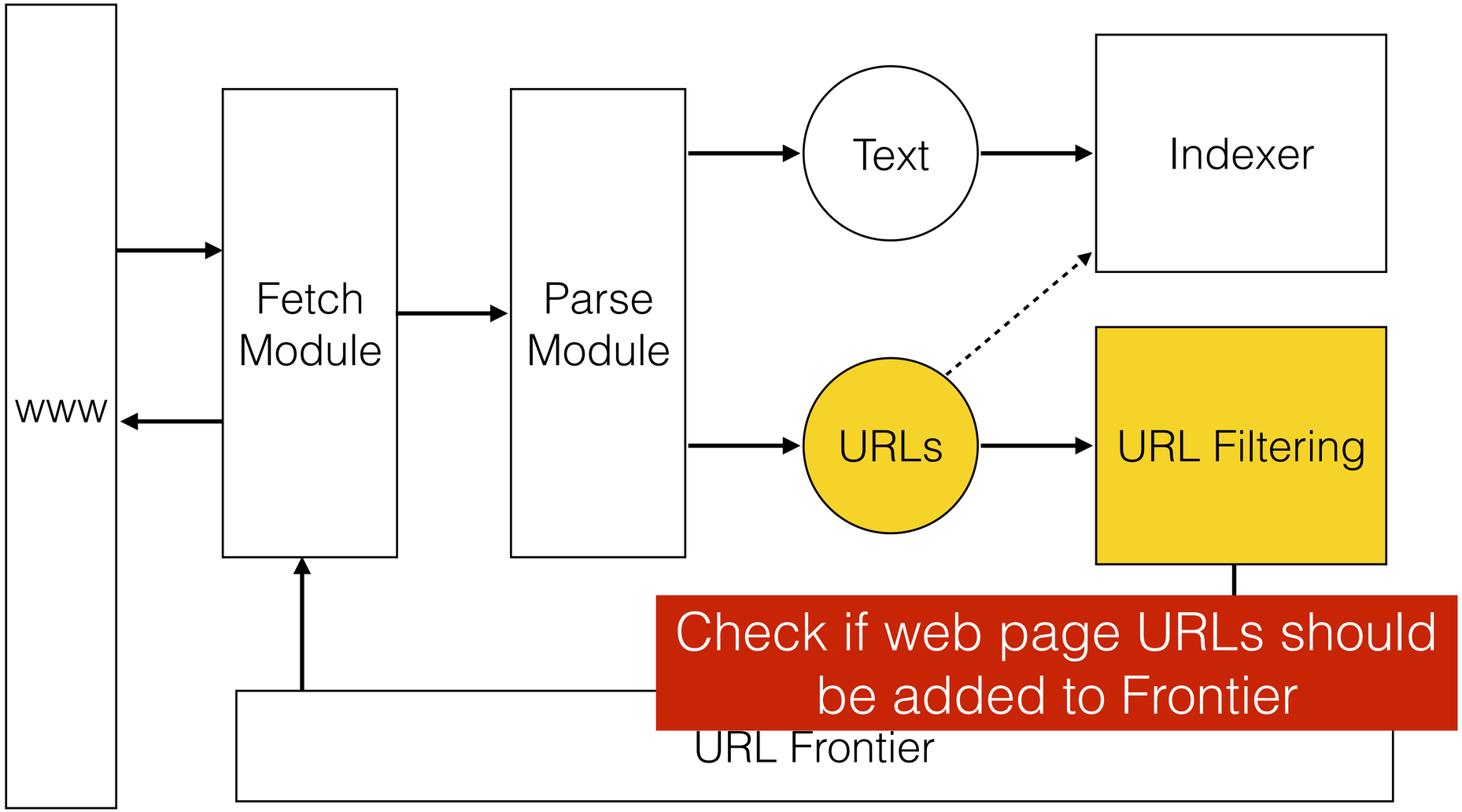

Fetch web page at URL
www
Fetch Module
Parse Module
Text
Indexer
URLs
URL Filtering
URL Frontier

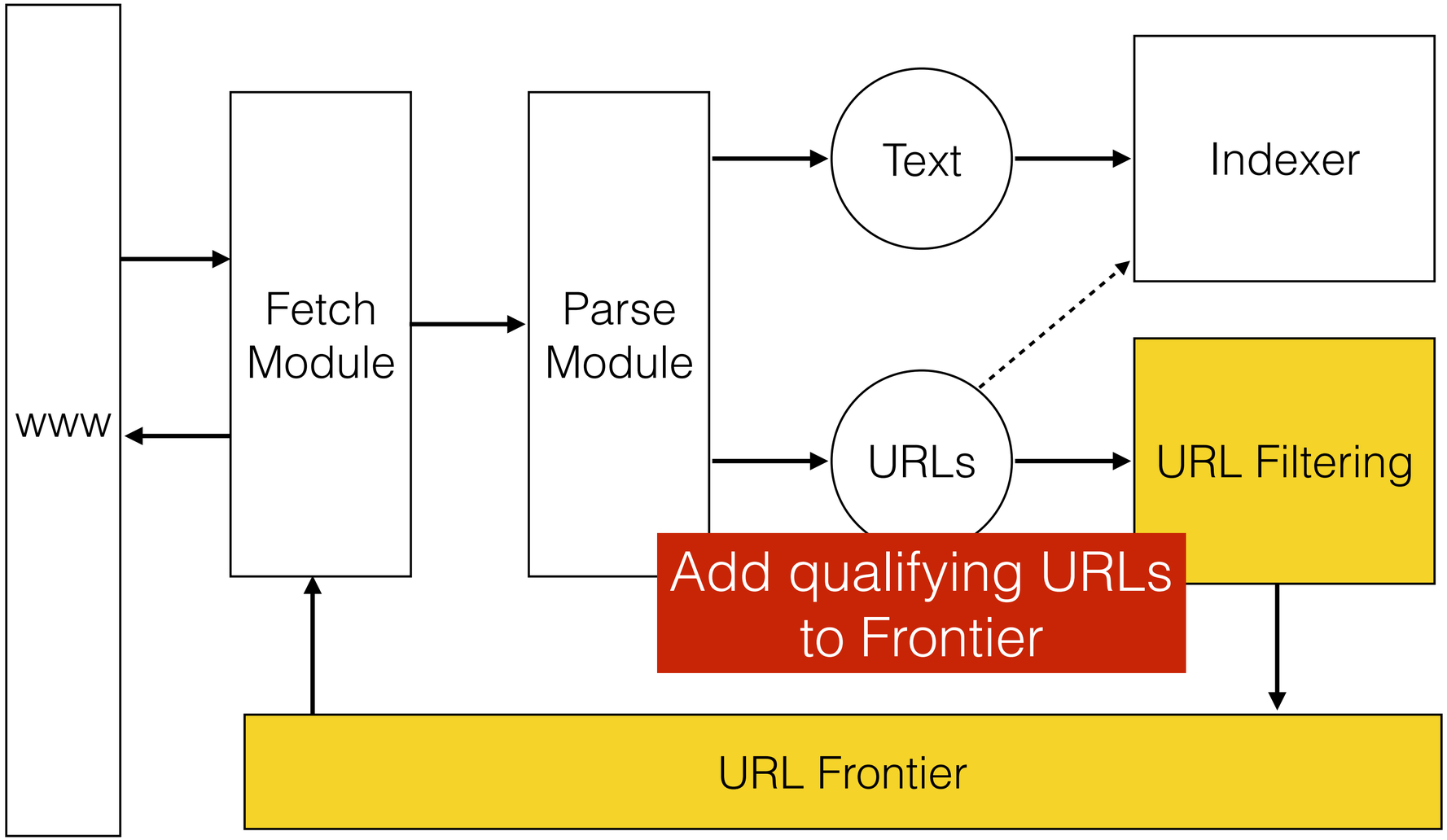

Send web page to parser
www
Fetch Module
Parse Module
Text
Indexer
URLs
URL Filtering
URL Frontier

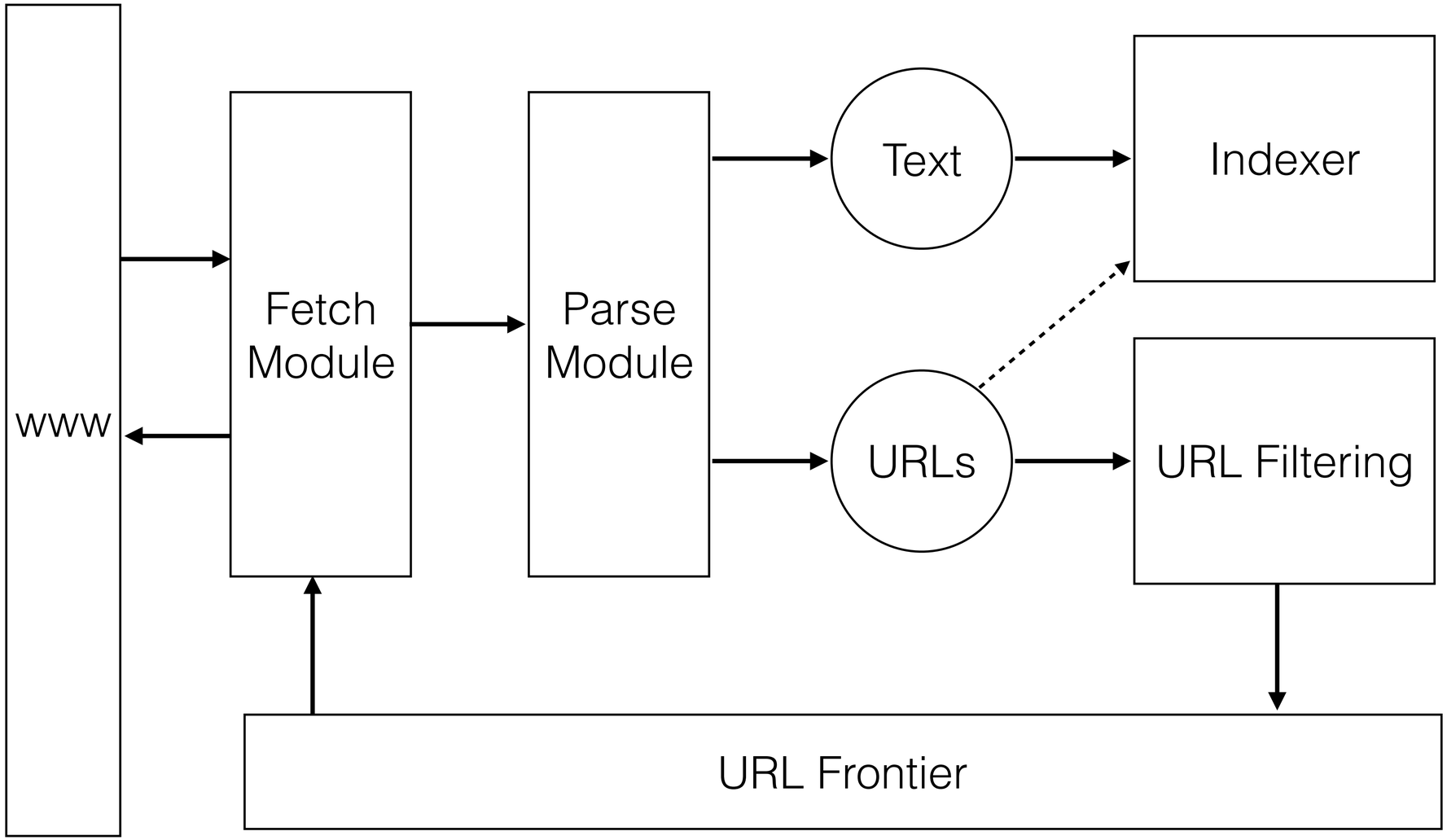

Extract web page text and links
www
Fetch Module
Parse Module
Text
URLs
Indexer
URL Filtering
URL Frontier

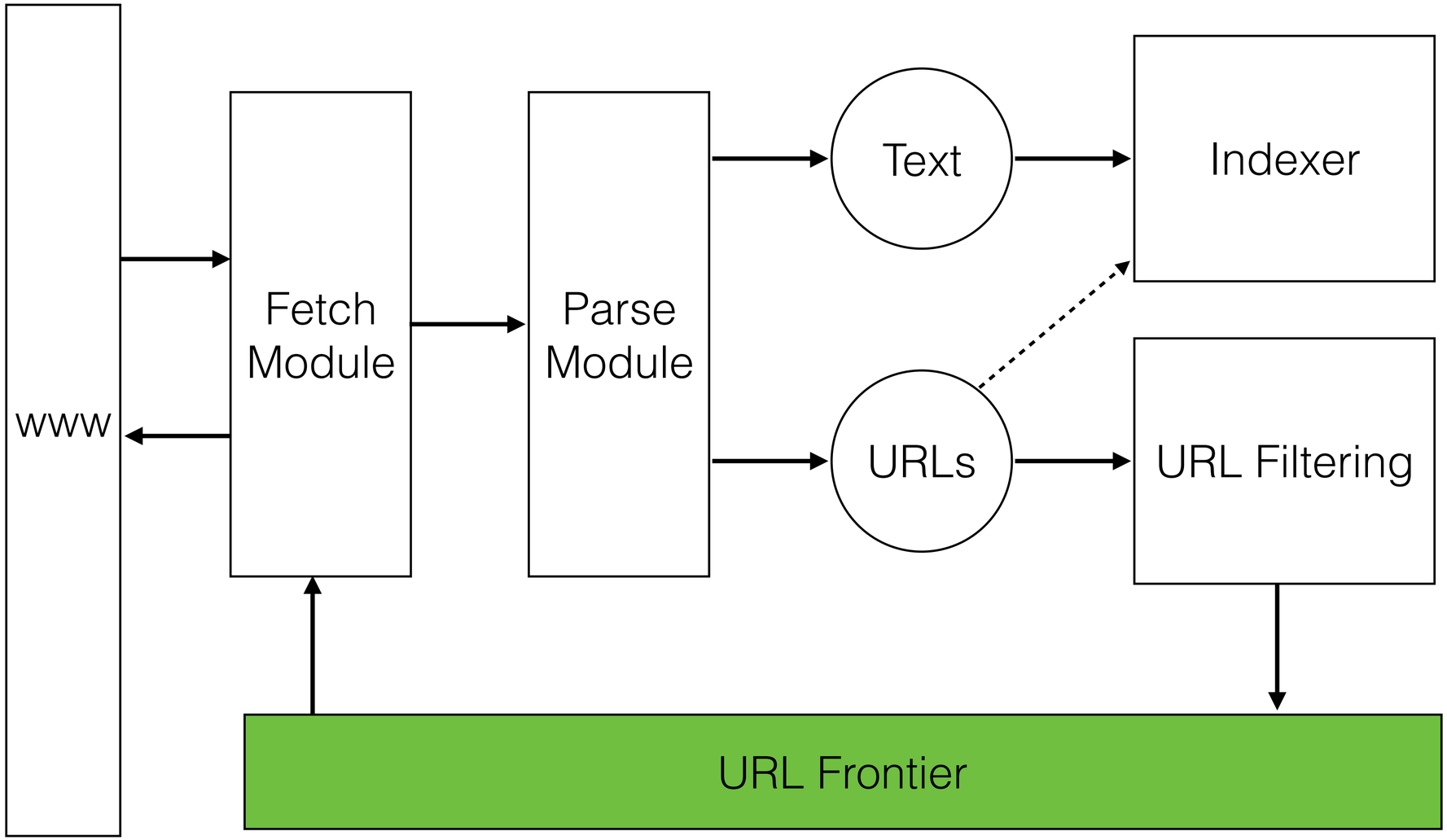
Send the text (and sometimes links) to the indexer




www
Fetch Module
Parse Module
Text
Indexer
URLs
URL Filtering
Check if web page URLs should be added to Frontier
URL Frontier


www
Fetch Module
Parse Module
Text
Indexer
URLs
URL Filtering
Add qualifying URLs to Frontier
URL Frontier


www
Fetch Module
Parse Module
Text
Indexer
URLs
URL Filtering
URL Frontier

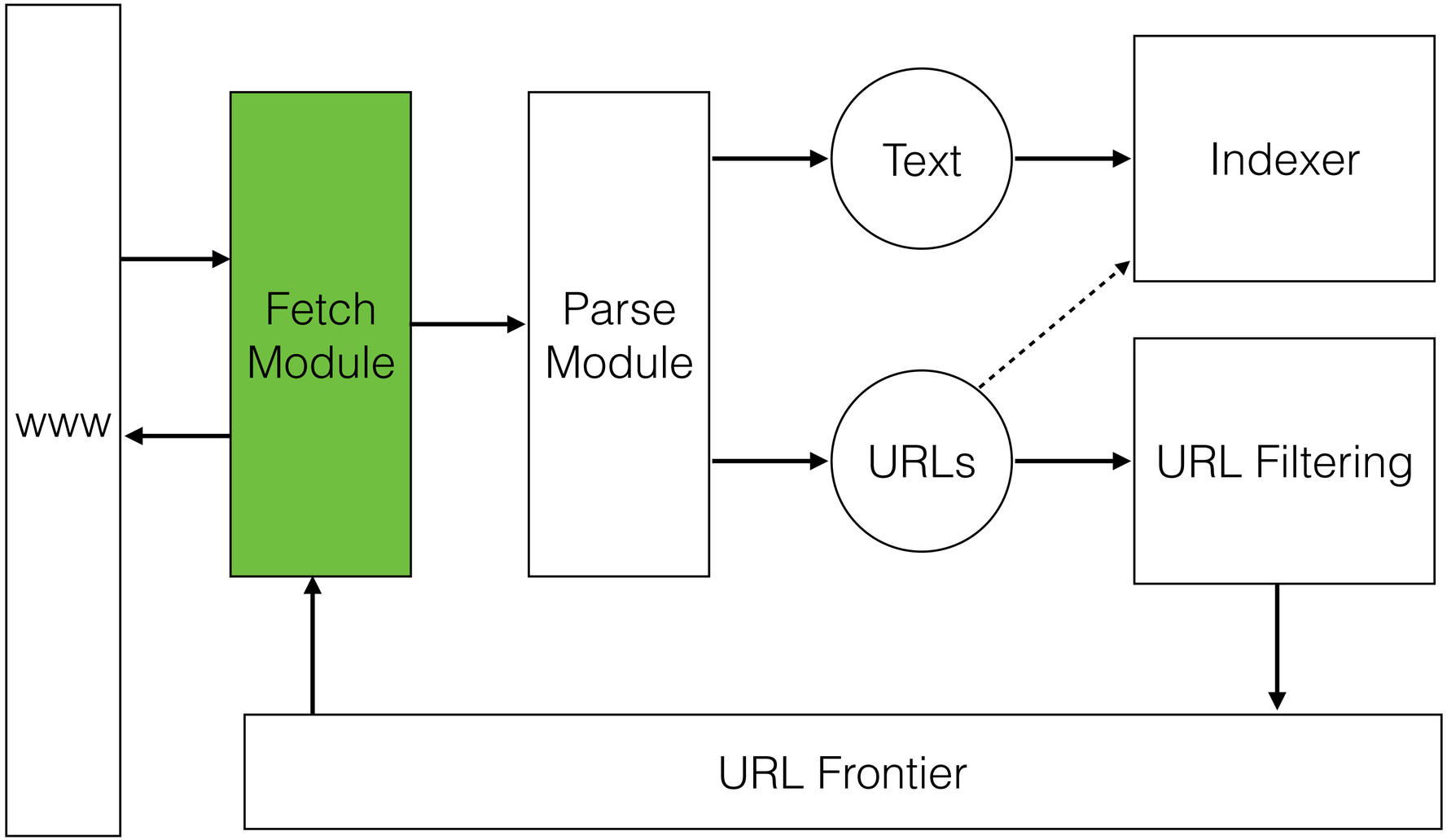

www
Fetch Module
Parse Module
Text
Indexer
URLs
URL Filtering
URL Frontier

URL Frontier

- Contains URLs to be fetched in current crawl
- Contains seed URL(s) in first loop
- Good practise to always check a URL against exclusions in robots.txt before fetching it
- The order in which the URLs are returned are governed by two considerations:
    1. *High quality* pages are prioritised
    2. Politeness constraints (avoiding requests to the same host within a short time span)

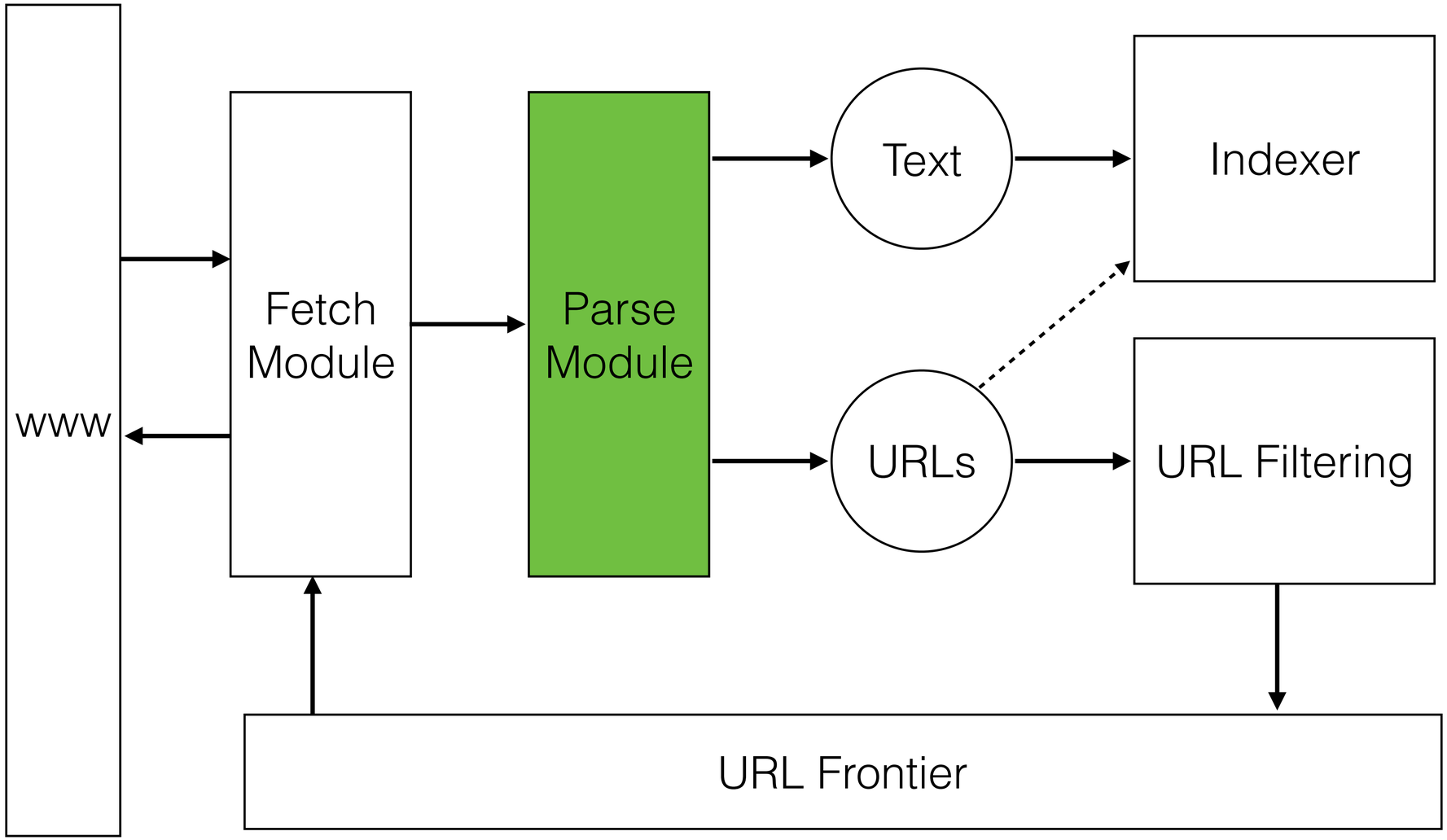
URL Frontier Prioritisation

- A concern for larger sites
- The priority of an URL can be judged based on various criteria:
    - The rate at which the web page at this URL has changed between previous crawls
        - E.g. A document that exhibited frequent change in the past would be assigned a higher priority than documents that have been updated less frequently
    - Staleness
        - Has it been too long since the previous crawl?
    - Other ad-hoc heuristics
        - E.g. Prioritise pages from news services
        - Set lower priority if site contains duplicated content
        - Set higher priority to popular URLs


www
Fetch Module
Parse Module
Text
Indexer
URLs
URL Filtering
URL Frontier

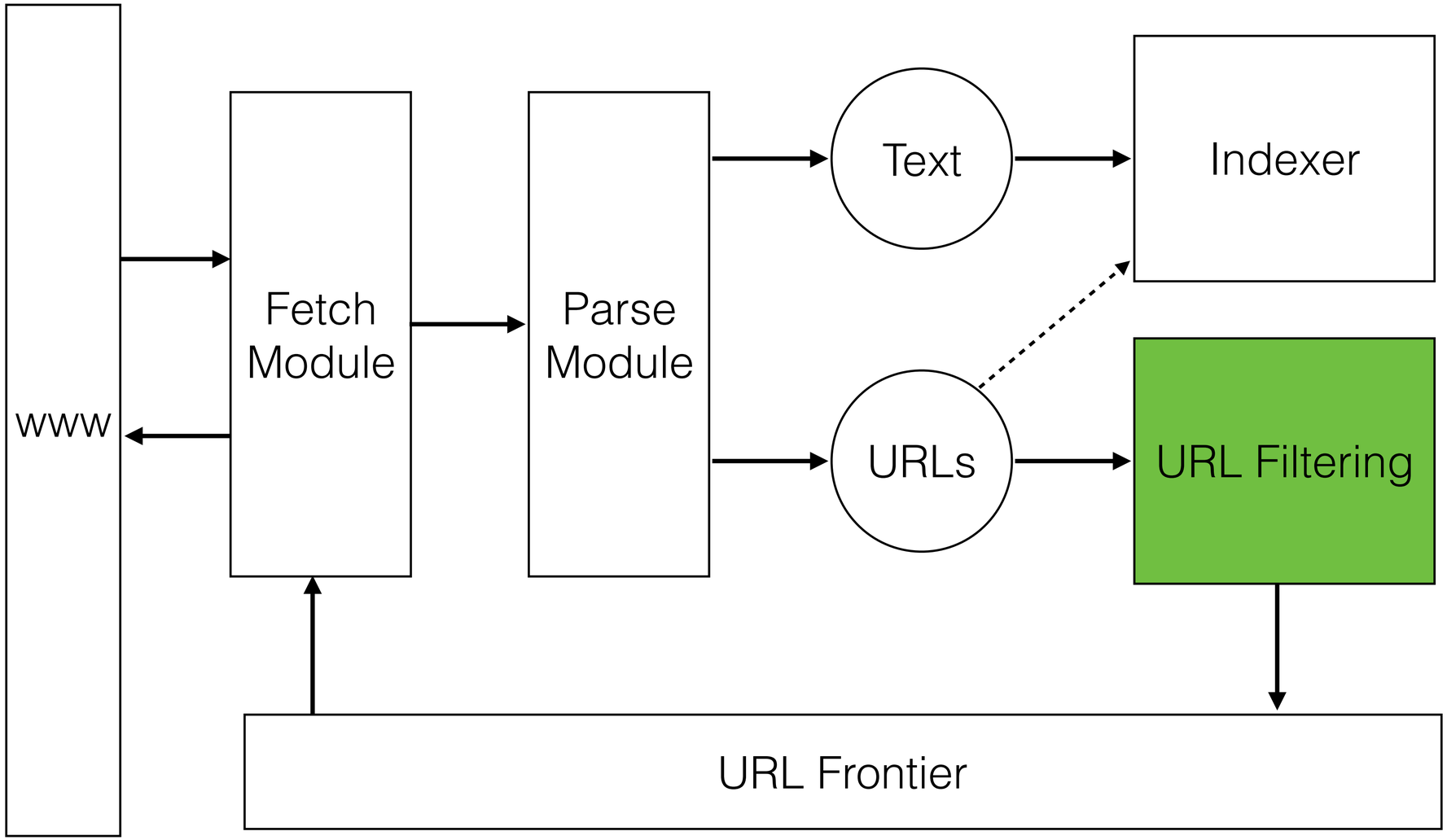
Fetch Module

- Resolves the IP address from domain name by querying DNS server
  - DNS server may contact other DNS servers
  - This stage is a bottleneck in crawling
    - Use a DNS cache containing recently used domain names
- Usually uses HTTP protocol to download webpage at the URL
- Web page is written to temporary storage while parsing is performed


www
Fetch Module
Parse Module
Text
Indexer
URLs
URL Filtering
URL Frontier

Parse Module

- Remove markup, extract content for indexing etc.
- URL normalisation
  - Convert all to lowercase and standardise
  - When a fetched document is parsed, some of the extracted links are relative URLs
    - E.g. http://en.wikipedia.org/wiki/Main_Page contains a relative link to:
      /wiki/Wikipedia:General_disclaimer
      which is the same as the absolute URL:
      http://en.wikipedia.org/wiki/Wikipedia:General_disclaimer
    - We must expand such relative URLs


www
Fetch Module
Parse Module
Text
Indexer
URLs
URL Filtering
URL Frontier

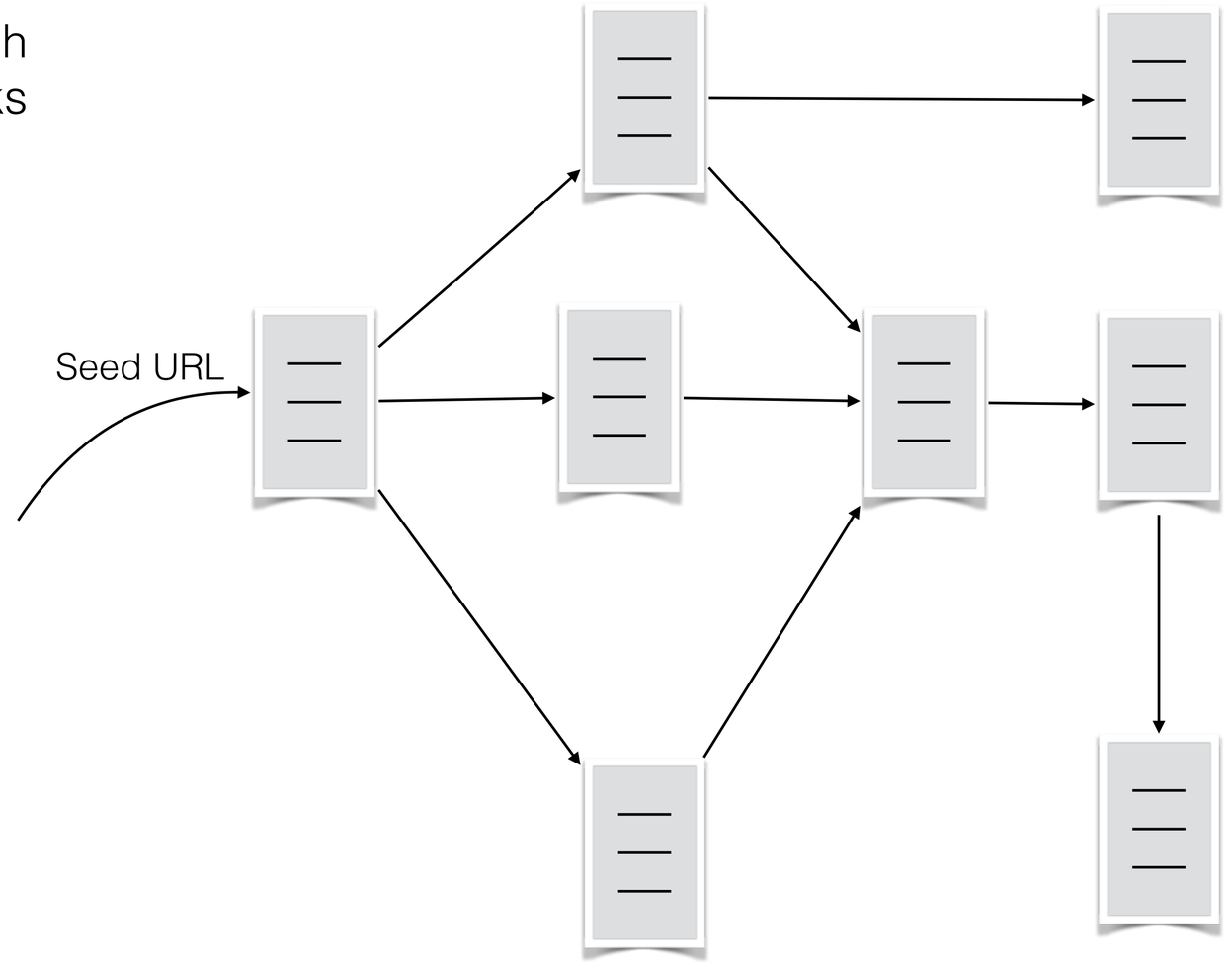
URL Filtering

- Decides which URLs to add to the URL Frontier for crawling
- Tests whether a web page with the same content has already been seen at another URL (e.g. using a checksum)
- Removes hosts that should not be crawled (specified by user)
- Checks URLs against exclusions in robots.txt
- Checks whether URL already exists in frontier or if it has already been crawled (note: only applies to non-*continuous* crawling)

Crawling Strategies

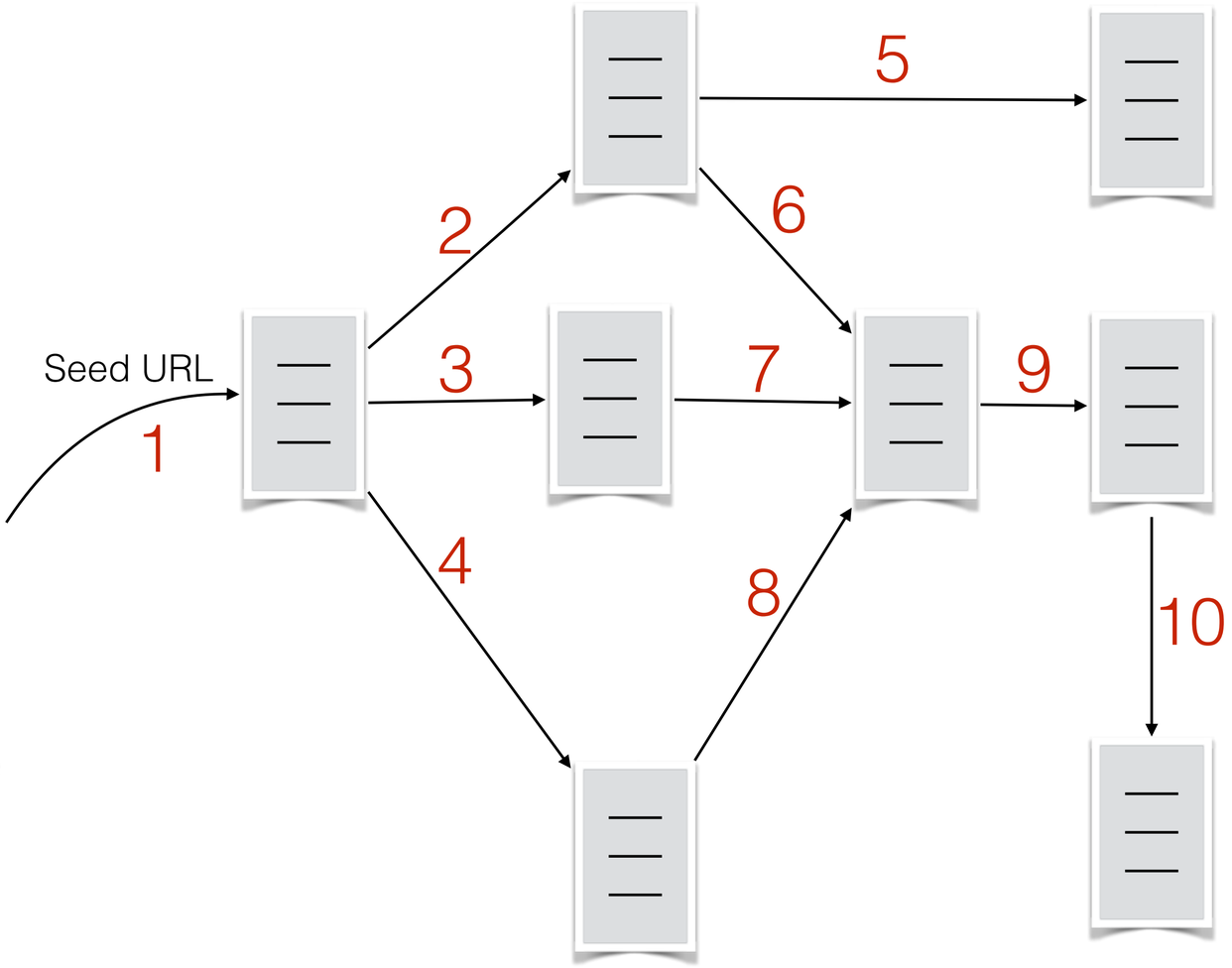
Crawling Strategies

- Web can be thought of as a huge directed graph (web) with documents as vertices and hyperlinks as edges
- Crawling traverses the graph
- The order in which the graph is traversed is usually either:
    - Breadth-first
    - Depth-first



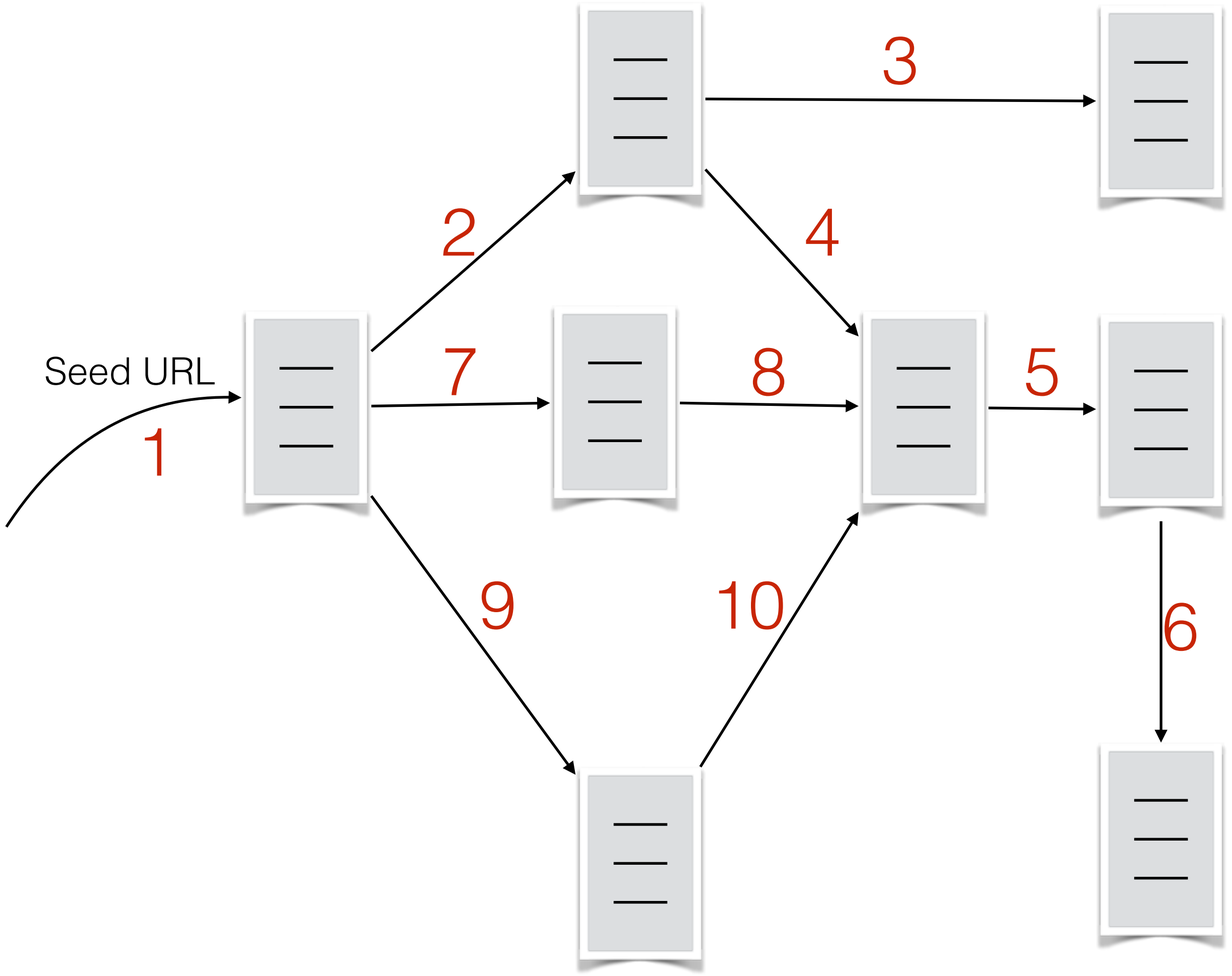
Breadth-first Traversal

1. Place all seed URLs in a queue
2. While the queue is not empty:
   A. Visit first node of queue
   B. Append linked URLs to queue if they don't already exist or have not already been visited
   C. Remove current node from queue



Depth-first Traversal

1. Get the first seed URL
2. Retrieve URL and fetch the first non-visited linked URL from that page
3. Repeat 2 until there are no more non-visited links on the page
4. Get the next non-visited link in the previous node and repeat 2



Depth-First vs Breadth-First

- For a **complete** scan, they both have the same result
- Breadth-first
    - Implemented with a queue (FIFO)
    - Finds pages along shortest paths
    - If we start with “good” pages, this keeps us close
- Depth-first
    - Implemented with a stack (LIFO)
    - Can wander away…

Continuous and Non-Continuous Crawling

- Some applications require re-indexing of previously fetched pages for *freshness*
    - E.g. search engines require a fairly current representation of a web page so continuously crawls looking for new or modified content
    - **Continuous** crawlers revisit previously fetched web pages
        - Visit rate ideally approximates the rate at which the page is updated
- Other applications require a **non-continuous** (one-shot) crawler, and the same web page should never be processed more than once
    - Eg. indexing a collection of historical documents

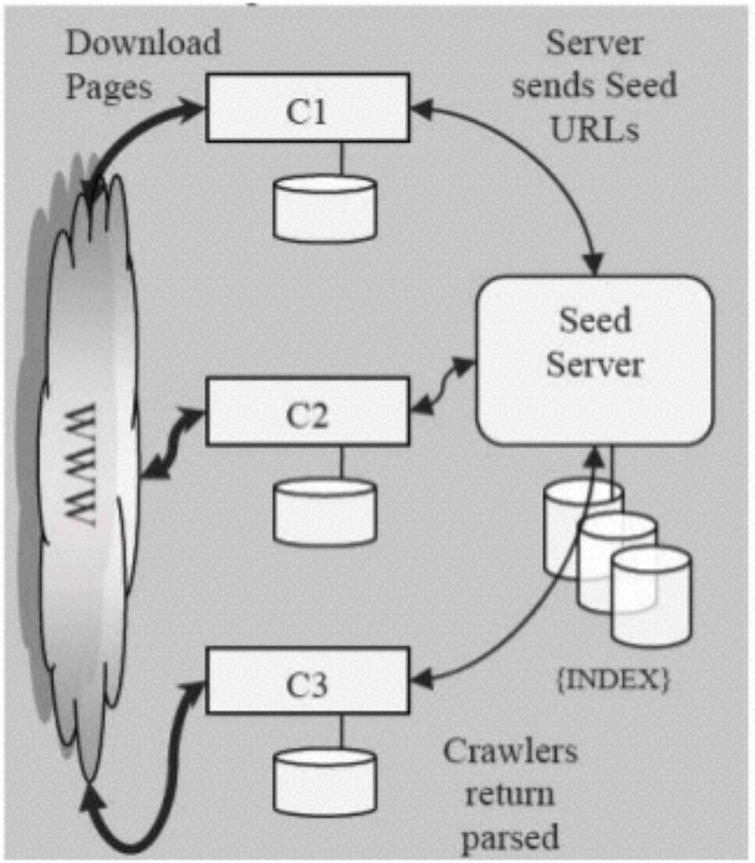
Distributing the Crawler

- Crawlers should be able to execute in a **distributed** fashion across multiple machines and use multiple threads

- It's good practise to distribute hosts across processes rather than URLs for two reasons:

    1. It makes it easier to crawl **politely**, since no more than one connection to a site will be open at any given time (two threads/ processes won't be visiting the same site simultaneously)

    2. You may geographically distribute your machines to process the domains in the nearby locations



Distributing the Crawler

- Usually crawling is performed on a large number of processes or threads
    - Each is assigned a set of URLs to process
    - Independently process and index the assigned URLS
    - When a new URL is found on a downloaded page, a *host splitter* checks that the URL's hostname is assigned to this process. If not, the URL is rerouted to the appropriate process.
- Indexes from each separate process are merged centrally
- Frontier and repository can be shared or the seed URLs can be fully partitioned across processes

Implementation Considerations

Implementation Issues

- Robots.txt should be cached so that it doesn't need to be re-read for each file on the same host (politeness)
    - (However, after a certain time has elapsed, the crawler should check for file changes and re-read it if necessary (to maintain *freshness*))
- We don't want to fetch the same page twice
    - Keep a table of visited sites
- Certain files should be skipped, e.g. Images, video etc.
    - Check the file meta data to determine its file type

More Implementation Issues

- Dynamically generated pages
    - Some websites are dynamically generated using JavaScript. When you download the page using a crawler, you might not have its full content unless you use a JavaScript engine
- Servers return all kinds of HTTP errors (500, 404, 400, etc.)
- Servers are often unreachable and cause timeouts.
    - The domain/website might not exist anymore, or there might be DNS problems, or it might be under heavy load, or the server might be incorrectly configured
- Some pages contain incorrectly formed HTML

Spider Traps

- Misleading sites with indefinite number of pages
    - dynamically generated pages
- Strategies to stop the crawler from getting stuck:
    - Check the URL length and limit the number of characters (e.g. 128 characters)
    - Not process sites that have a very large number of hyperlinks
    - Disable crawling dynamic sites — IF detectable

Crawling in Python

PCcrawler.py

- A stand-alone crawler written in Python (http://math.nist.gov/~RPozo/ngraph/webcrawler.html)

>> python3 PCcrawler.py domain-pattern seed-url max-pages

PCcrawler.py

- A stand-alone crawler written in Python (http://math.nist.gov/~RPozo/ngraph/webcrawler.html)

```
>> python3 PCcrawler.py domain-pattern seed-url max-pages
```

Use the Python 3.6 interpreter

PCcrawler.py

- A stand-alone crawler written in Python (http://math.nist.gov/~RPozo/ngraph/webcrawler.html)

>> python3 PCcrawler.py domain-pattern seed-url max-pages

Call PCcrawler.py

PCcrawler.py

- A stand-alone crawler written in Python (http://math.nist.gov/~RPozo/ngraph/webcrawler.html)

>> python3 PCcrawler.py domain-pattern seed-url max-pages

This pattern has to appear
in every URL that is crawled.
If it doesn't appear anywhere in
the URL, the page is not processed.

PCcrawler.py

- A stand-alone crawler written in Python (http://math.nist.gov/~RPozo/ngraph/webcrawler.html)

```
>> python3 PCcrawler.py domain-pattern seed-url max-pages
```


The seed URL. This is where the crawler starts the crawl.

PCcrawler.py

- A stand-alone crawler written in Python (http://math.nist.gov/~RPozo/ngraph/webcrawler.html)

>> python3 PCcrawler.py domain-pattern seed-url max-pages

The maximum number of pages to crawl.

PCcrawler.py - example

>> python3 PCcrawler.py inforet.cmp.uea.ac.uk http://inforet.cmp.uea.ac.uk/ 10

Lab

- Brings together elements from last three lectures: Term normalisation, indexing and crawling.
- Crawl the *Looking Glass* and build an index containing:
  - A vocabulary table - a list containing the distinct terms in the corpus
  - A document table - a list containing the URLs of each document
  - A postings table - a dictionary mapping from term ID to document ID
- Forms the basis of your coursework…

Resources

- An interesting read: https://benbernardblog.com/the-tale-of-creating-a-distributed-web-crawler/