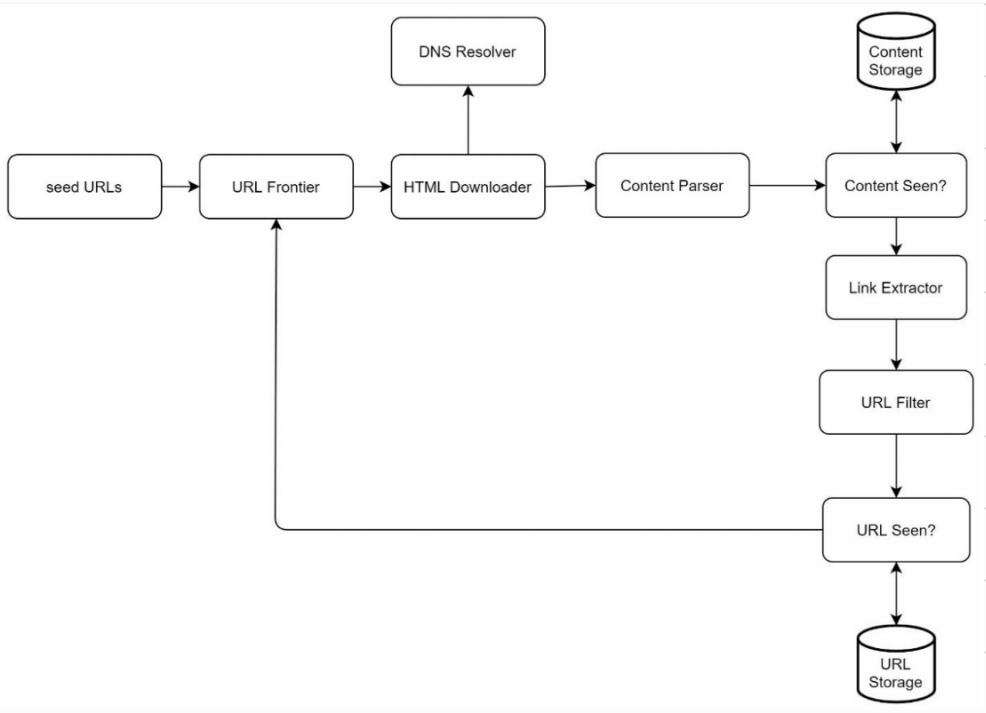
Design a Web Crawler

- Step 1 - Understand the problem and establish design scope
  - The basic algorithm of a web crawler is simple:
    1. Given a set of URLs, download all the web pages addressed by the URLs.
    2. Extract URLs from these web pages.
    3. Add new URLs to the list of URLs to be downloaded. Repeat these 3 steps.

  - Characteristics of a good web crawler:
    - Scability: The web is very large. There are billion of web pages out there. Web crawling should be extremely efficient using parallelization.
    - Robustness: The web is full of traps. Bad HTML, unresponsive servers, crashes, malicious links etc. are all common. The crawler must handle all those edge cases.
    - Politeness: The crawler should not make too many requests to a website within a short time interval.
    - Extensibility: The system is flexible so that minimal changes are needed to support new content types.

  - Back of the envelope estimation
    - Assume 1 billion web pages are downloaded every month.
    - QPS: 1,000,000,000 / 30 / 24 / 3600 = ~400 pages per second
    - Peak QPS = 2 * QPS = 800
    - Assume average web size is 500k.
    - 1 billion pages x 500k = 500TB per month.
    - Assuming data are stored for five years, 500TB x 12 x 5 = 30PB. A 30PB storage is needed to store five-year content.

- Step 2 - Propose high-level design and get buy-in



- Send URLs
  - A web crawler uses seed URLs as starting point for the crawl process.
  - A good seed URL serves as a good starting point that a crawler can utilize to traverse as many links as possible.
  - The general strategy is to divide the entire URL space into smaller ones.
  - The first proposed approach is based on locality as different countries may have different popular websites.
  - Another way is to choose seed URLs based on topics.

- URL Frontier
  - Most modern web crawlers split the crawl state into two: to be downloaded and already downloaded.
  - The component that stores URLs to be downloaded is called the URL frontier.
  - It can be referred to as a FIFO queue.

- HTML Downloader
  - The HTML downloader downloads web pages from the internet.
  - These URLs are provided by the URL frontier.

- DNS Resolver
  - To download a web page, a URL must be translated into an IP address.
  - The HTML downloader calls the DNS resolver to get the corresponding IP address for the URL.

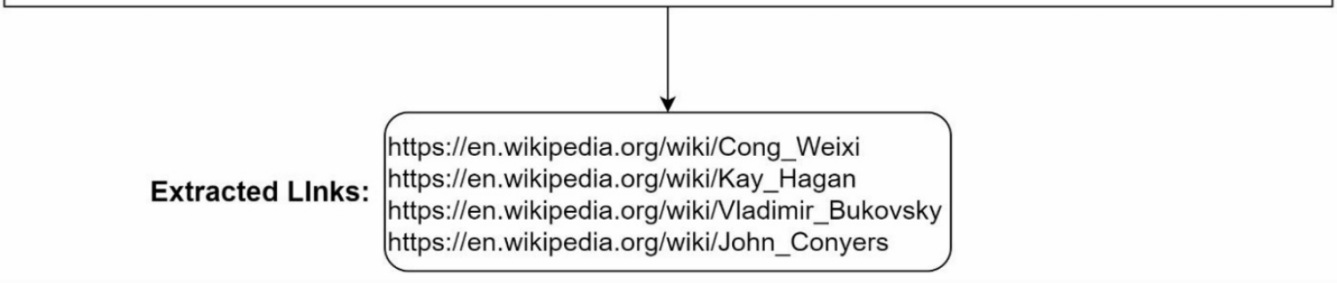
- Content Parser
  - After a web page is downloaded, it must be parsed and validated because malformed web pages could provoke problems and waste storage space.
  - Implementing a content parser in a crawl server will slow down the crawling process.

- Content Seen?
  - 'Content Seen' data structure is used to eliminate data redundancy and shorten processing time. It helps to detect new content previously stored in the system.
  - An efficient way to compare two HTML documents is to compare their hash values.

- Content Storage
  - It is a storage system for storing HTML content.
    - Most of the content is stored on disk because the data set is too big to fit in memory.
    - Popular content is kept in memory to reduce latency.

- URL Extractor
  - URL Extractor parses and extracts links from HTML pages.

```
<html class="client-nojs" lang="en" dir="ltr">
   <head>
      <meta charset="UTF-8"/>
      <title>Wikipedia, the free encyclopedia</title>
   </head>
   <body>
      <li><a href="/wiki/Cong_Weixi" title="Cong Weixi">Cong Weixi</a></li>
      <li><a href="/wiki/Kay_Hagan" title="Kay Hagan">Kay Hagan</a></li>
      <li><a href="/wiki/Vladimir_Bukovsky" title="Vladimir Bukovsky">Vladimir Bukovsky</a></li>
      <li><a href="/wiki/John_Conyers" title="John Conyers">John Conyers</a></li>
   </body>
</html>
```



Extracted LInks:

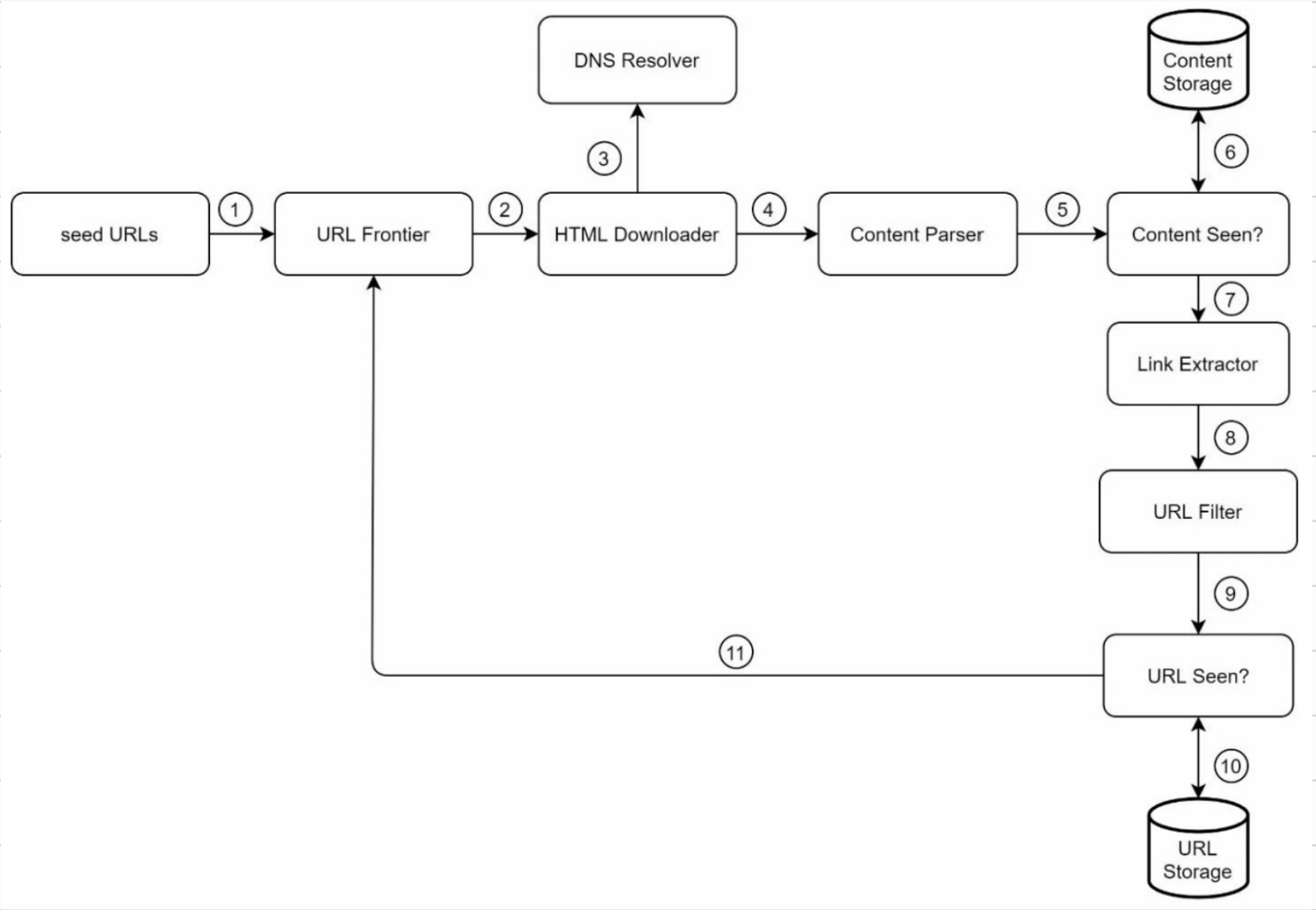
https://en.wikipedia.org/wiki/Cong_Weixi
https://en.wikipedia.org/wiki/Kay_Hagan
https://en.wikipedia.org/wiki/Vladimir_Bukovsky
https://en.wikipedia.org/wiki/John_Conyers

- URL Filter
  - The URL filter excludes certain content types, file extensions, error links and URLs in "blacklisted" sites.

- URL Seen?
  - It is a datastructure that keeps track of URLs that are visited before or already in the frontier.
  - It helps to avoid adding same URL multiple times.
  - Bloom filter and hash tables are common technique to implement the "URL Seen" component.

- URL Storage
  - URL Storage stores already visited URLs.

- Web Crawler Workflow

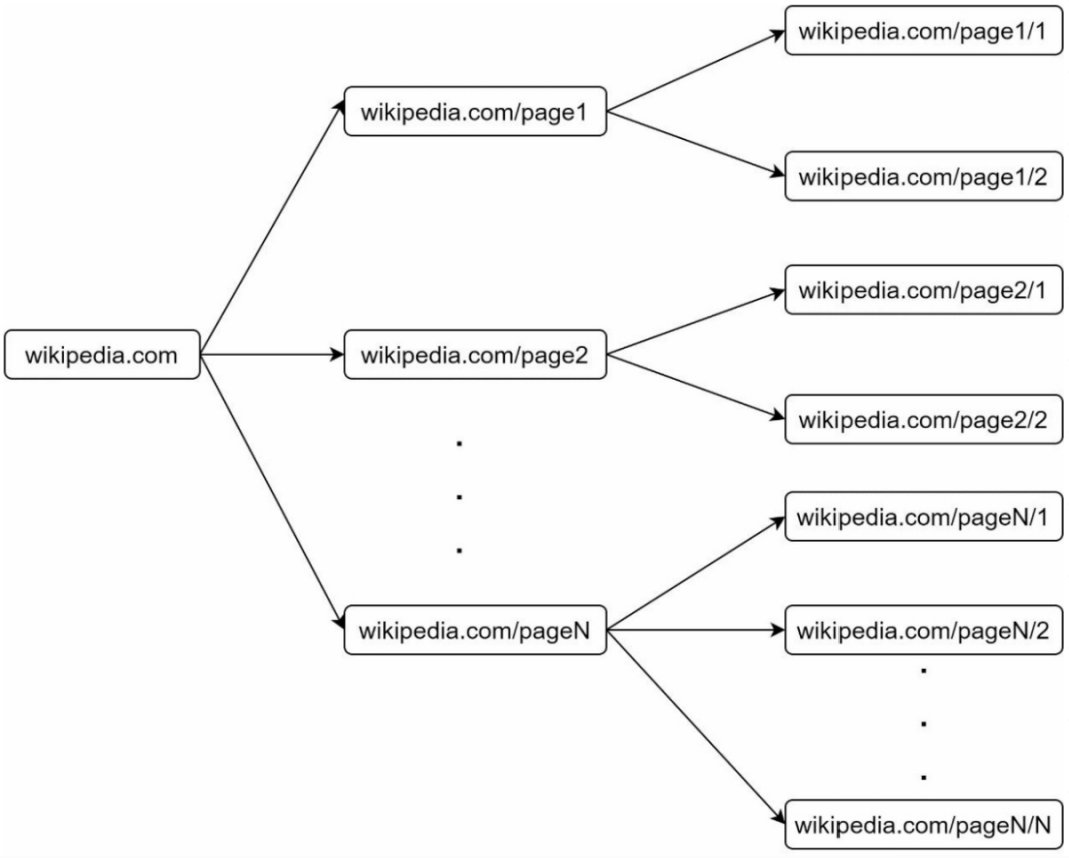


- Step 1: Add seed URLs to the URL Frontier.
- Step 2: HTML downloader fetches a list of URLs from URL Frontier.
- Step 3: HTML downloader gets IP addresses of URLs from DNS resolver and starts downloading.
- Step 4: Content parser parses HTML pages and checks if pages are malformed.
- Step 5: After content is parsed and validated, it is passed to the content seen component.
- Step 6: Conten Seen component checks if HTML page is already in the storage.
- Step 7: Link extractor extract links from HTML pages.
- Step 8: Extracted links are passed to the URL filter.
- Step 9: After links are filtered, they are passed to URL Seen.

- Step 10: URL Seen component checks if a URL is already in the storage.
- Step 11: If a URL has not been processed before, it is added to the URL Frontier.

- Step 3 - Design Deep Dive
  - Building components and techniques:
    - DFS vs BFS
    - URL Frontier
    - HTML Downloader
    - Robustness
    - Extensibility
    - Detect and avoid problematic content.

- DFS vs BFS
  - The crawl process can be seen as traversing a directed graph from one web page to others.
  - DFS is not a good choice because the depth of DFS can be very deep.
  - BFS is commonly used by web crawlers and is implemented by a FIFO queue. However, this implementation has 2 problems:
    - Most links from the same web page are linked back to the same host. When the crawler tries to download web pages in parallel, host servers will be flooded with requests. This is considered as "impolite".

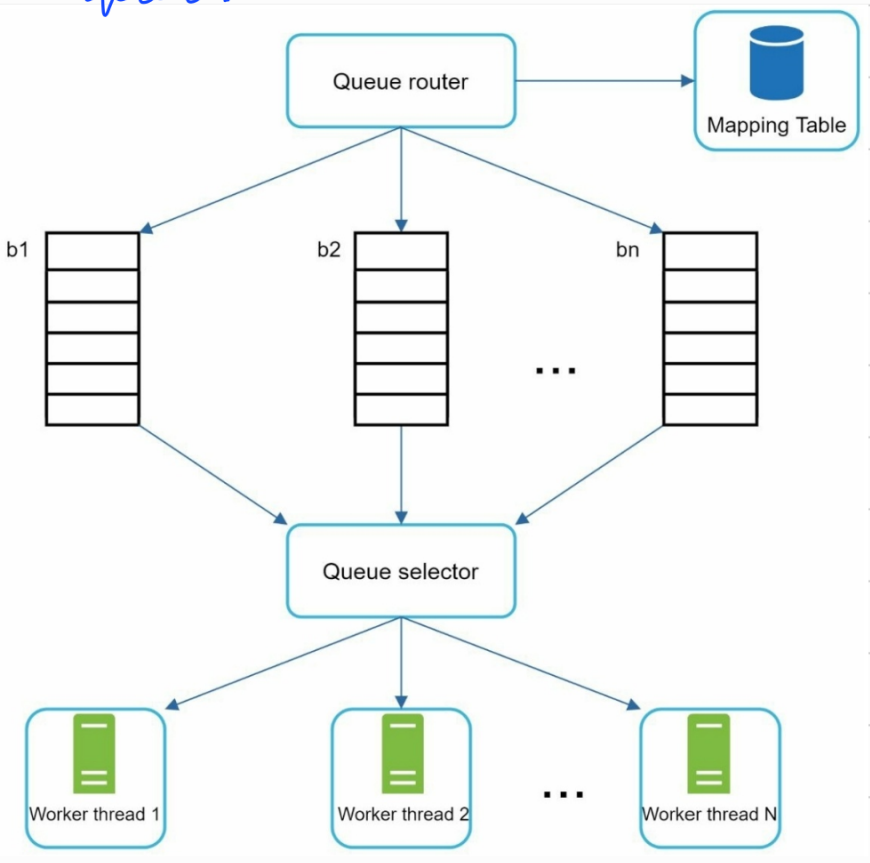


- Standard BFS does not take the priority of a URL into consideration. The web is large and not every page has the same level of quality and importance. Therefore, we may want to prioritize URLs according to their page ranks, web traffic, update frequency etc.

- URL Frontier
  - A URL Frontier is a data structure that stores URLs to be downloaded.
  - The URL frontier is an important component to ensure politeness, URL prioritization and freshness.

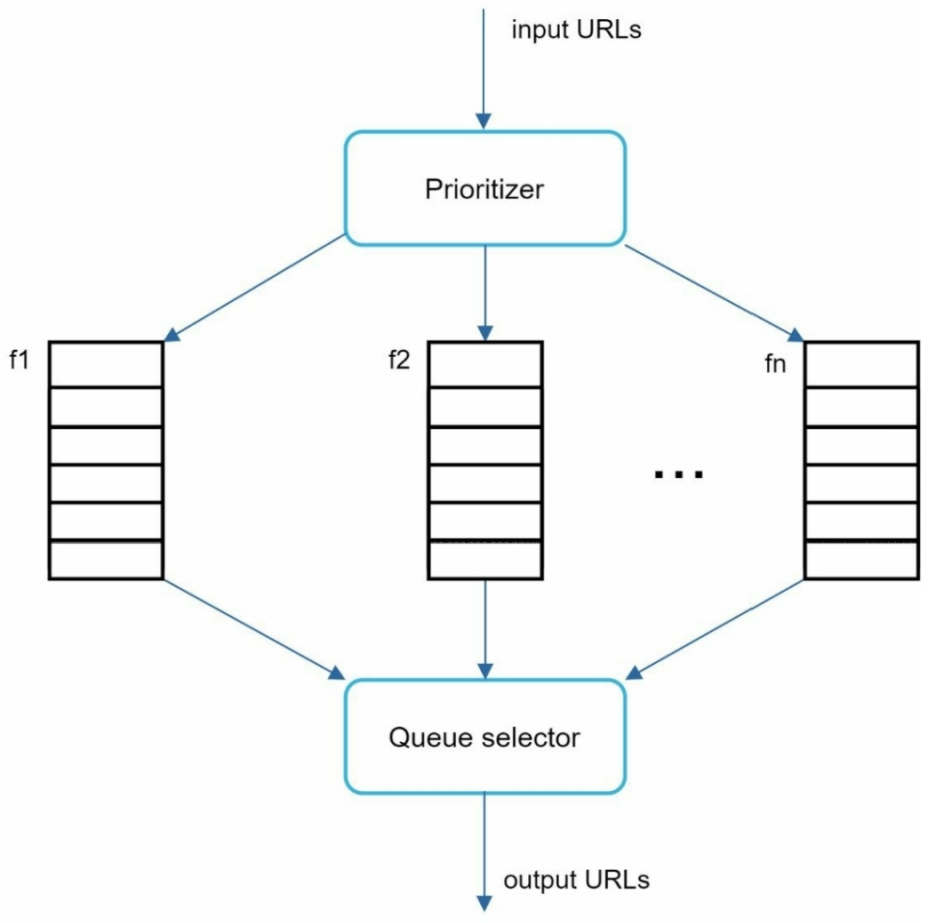
- Politeness
  - Generally a web crawler should avoid sending too many requests to the same hosting server within a short period.
  - Sending too many requests is considered as "impolite" or even treated as Denial of Service attack.

- The general idea of enforcing politeness is to download one page at a time from the same host. A delay can be added between two download tasks. The politeness constraint is implemented by maintaining a mapping from website hostnames to download threads. Each download thread has a separate FIFO queue and only downloads URLs obtained from that queue.



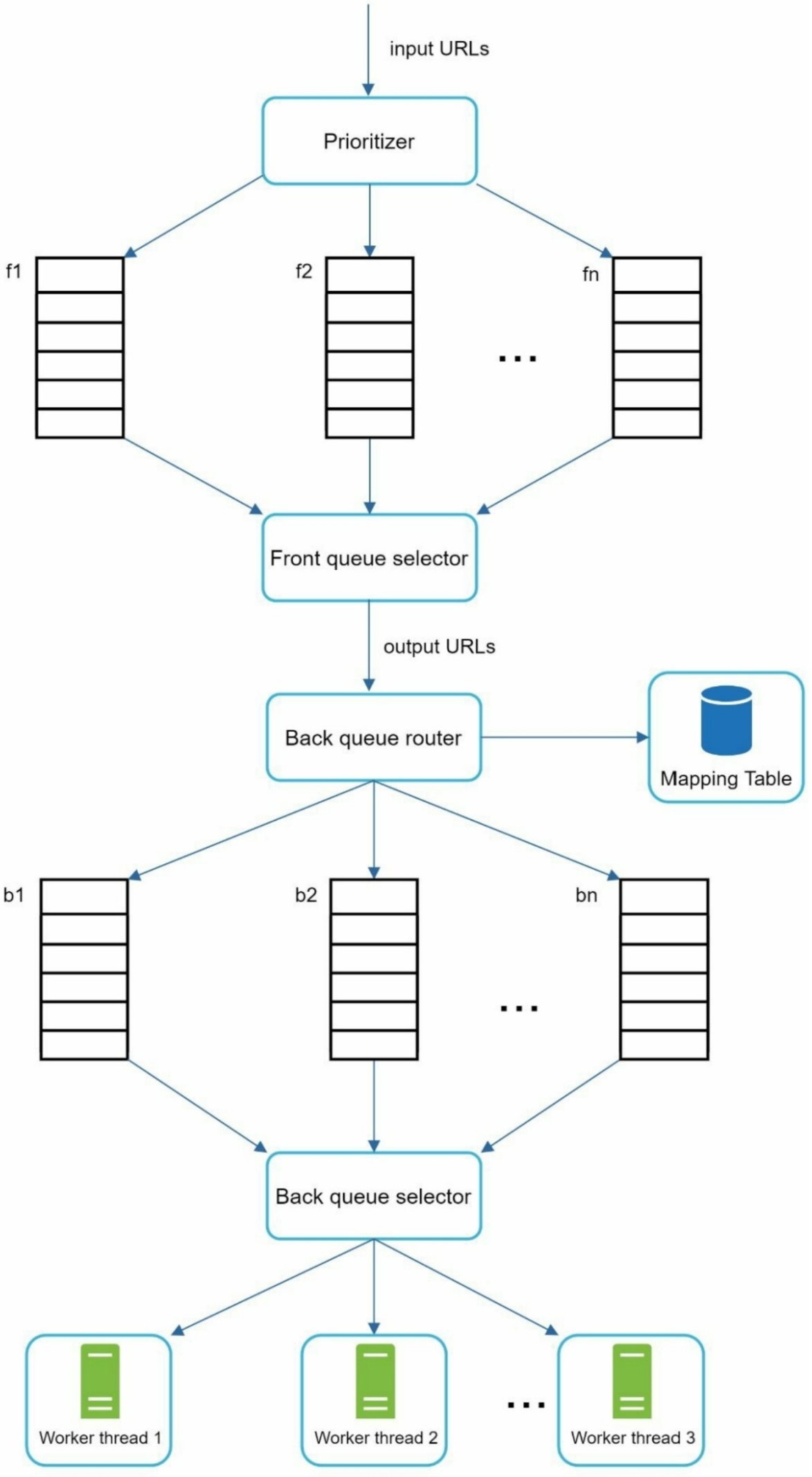
1. Queue Router: It ensures that each queue only contains URLs from the same host.
2. Mapping Table: It maps each host to a queue.
3. FIFO queues: Each queue contains URLs from same host.
4. Queue Selector: Each worker thread is mapped to a FIFO queue, and it only downloads URLs from that queue. The queue selection logic is done by the queue selector.
5. Worker Thread: downloads web pages from same host. A delay can be added between two download tasks.

- Priority
  - URLs are prioritized based on usefulness, which can be measured by PageRank, website traffic, update frequency etc. "Prioritizer" is the component that handles URL prioritization.



1. Prioritizer: It takes URLs as input and computes the priorities.
2. Queue: Each queue has an assigned priority. Queues with higher priority are selected with higher probability.
3. Queue Selector: Randomly choose a queue with a bias towards queues with higher priority.

- URL Frontier contains 2 modules:
  1. Front Queues: Manage Prioritization
  2. Back Queues: Manage Politeness


input URLs
Prioritizer
f1
f2
fn
Front queue selector
output URLs
Back queue router
Mapping Table
b1
b2
bn
Back queue selector
Worker thread 1
Worker thread 2
Worker thread 3

- Freshness
  - Web pages are constantly being added, deleted and edited. A web crawler must periodically recrawl downloaded pages to keep the dataset fresh.
  - Recrawl all the URLs is time-consuming and resource intensive.
    1. Recrawl based on web pages update history.
    2. Prioritize URLs and recrawl important pages first and more frequently.

- Storage for URL Frontier
  - In real world crawl for search engines, the number of URLs in the frontier could be hundreds of millions. Putting everything in memory is neither durable or scalable.
  - Keeping everything in the disk is undesirable neither because the disk is slow; and it can easily become a bottleneck.
  - Hybrid approach is adopted, the majority of URLs are stored on disk, so the storage space is not a problem.
  - To reduce the cost of reading from the disk and writing to the disk, we maintain buffers in memory for enque/deque operations.
  - Data in the buffer is periodically written to the disk.

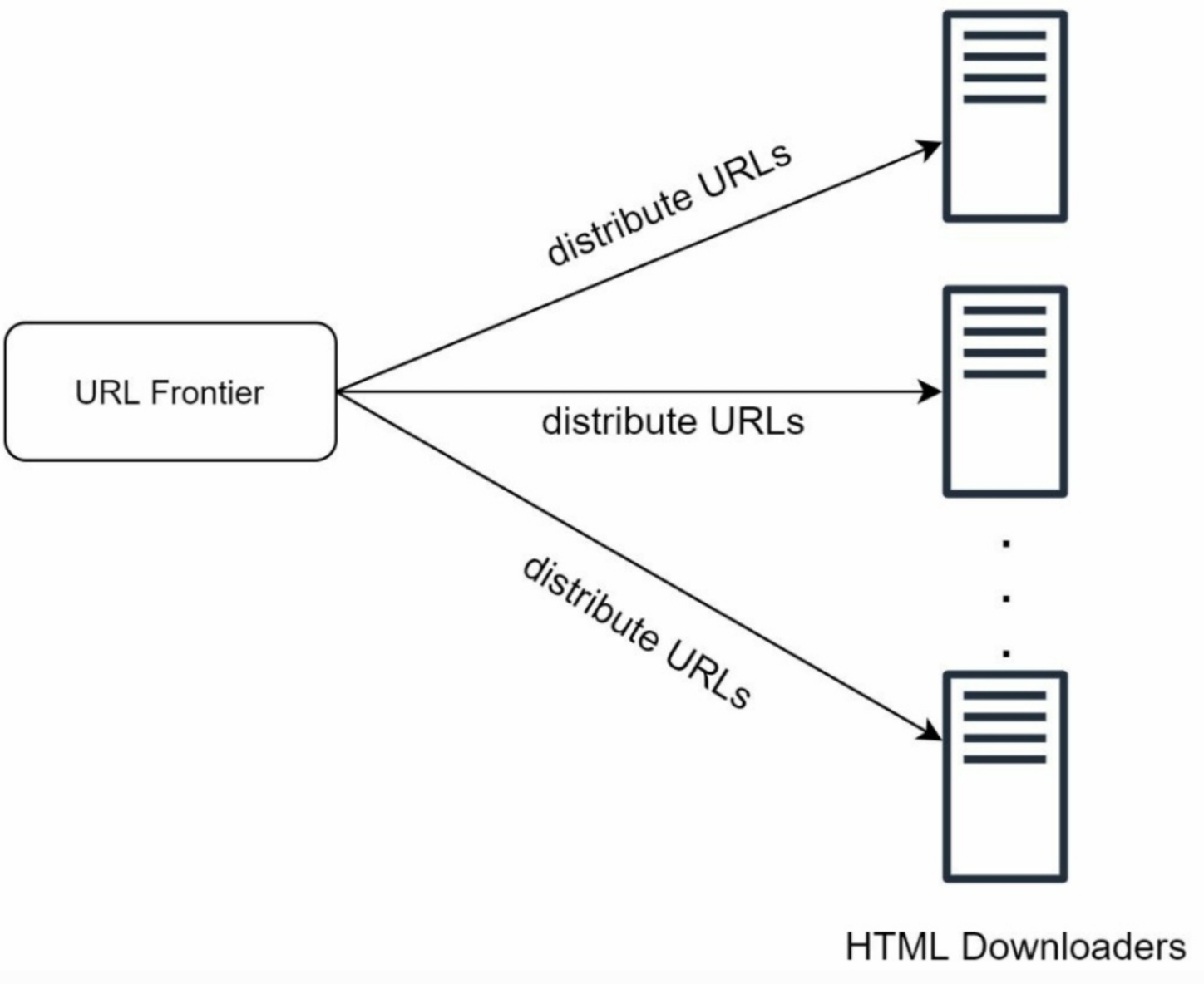
- HTML Downloader
  - HTML Downloader downloads web pages from the internet using the HTTP protocol.

- Robots.txt
  - Robots.txt called Robots exclusion protocol, is a standard used by websites to communicate with crawlers. It specifies what pages crawlers are allowed to download.

- Before attempting to crawl a web site, a crawler should check its corresponding robots.txt first and follow its rules.
- To avoid repeat downloads of robots.txt file, we cache the results of the file.

- Performance Optimization

1. Distributed Crawl
   - To achieve high performance, crawl jobs are distributed into multiple servers, and each server runs multiple threads.
   - The URL space is partitioned into smaller pieces; so, each downloader is responsible for a subset of URLs.



2. Cache DNS Resolver

- DNS resolver is a bottleneck for crawlers because DNS requests might take some time due to the synchronous nature of many DNS interfaces.
- DNS response time ranges from 10 ms to 200 ms.
- Once a request to DNS is carried out by a crawler thread, other threads are blocked until the first request is completed.
- Maintaining our DNS cache to avoid calling DNS frequently is an effective technique speed optimization.

3. Locality

- Distribute crawl servers geographically. When crawl servers are close to website hosts, crawlers experience faster download time.

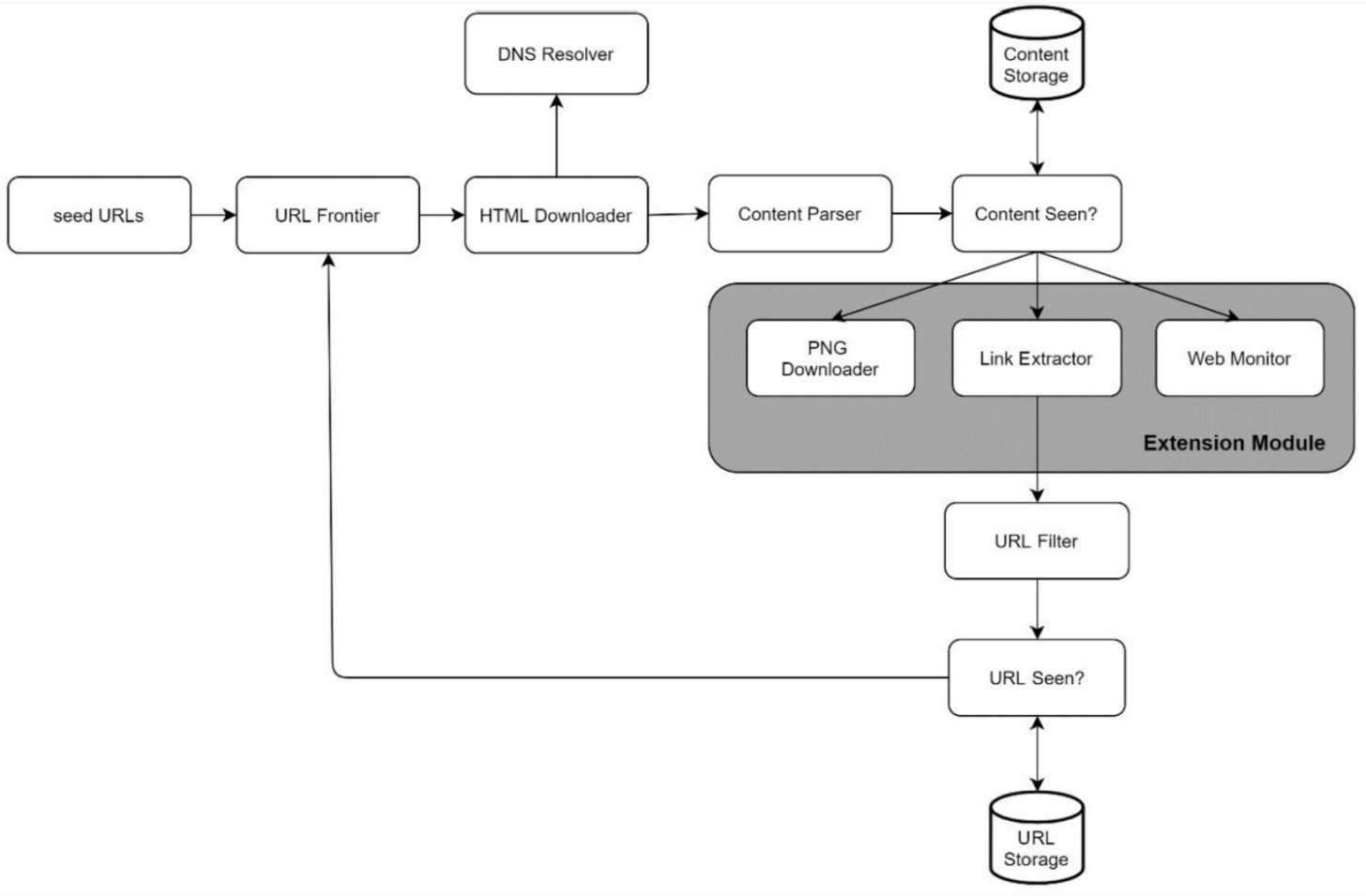
4. Short Timeout

- Some web servers respond slowly or may not respond at all.
- To avoid long wait time, a maximal wait time is specified. If a host does not respond within a predefined time, the crawler will stop the job and crawl some other pages.

- Robustness
  - Consistent hashing: This helps to distribute loads among downloaders. A new downloader server can be added or removed using consistent hashing.
  - Save crawl states and data: To guard against failures, crawl states and data are written to a storage system. A disrupted crawl can be restarted easily by loading saved states and data.

- Exception handling: Errors are inevitable and common in a large-scale system. The crawler must handle exceptions gracefully without crashing the system.
- Data Validation: This is an important measure to prevent system errors.

- Extensibility



1. PNG downloader module is plugged-in to download PNG files.
2. Web monitor module is added to monitor the web and prevent copyright and trademark infringements.

- Detect and avoid problematic content

1. Redundant Content
   - 30% of the web pages are duplicates. Hashes or checksums help to detect duplication.

2. Spider Traps
   - A spider trap is a web page that causes a crawler in an infinite loop.
   - Such spider traps can be avoided by setting a maximum length for URLs.
   - However, no one-size-fits-all solution exists to detect spider traps. Websites containing spider traps are easy to identify due to an unusually large number of web pages discovered on such websites.
   - It is easy to manually verify and identify a spider trap and exclude them from crawler.

3. Data Noise
   - Some of the contents have little to no value, such as advertisements, code snipper, spam URLs etc.
   - Those contents are not useful for crawlers and should be excluded if possible.

- Step 4 - Wrap Up
  - Server-side rendering: Numerous websites use scripts like JS, AJAX etc to generate links on the fly. If we download and parse web pages directly, we will not be able to retrieve dynamically generated links. To solve this problem, we perform server-side rendering first before parsing the page.
  - Filter out unwanted pages: With finite storage capacity and crawl resources, an anti-spam component is beneficial in filtering out low quality and spam pages.
  - Database Replication and sharding: Techniques like replication and sharding are used to improve the data layer availability, scalability and reliability.
  - Horizontal Scaling: For large scale crawl, hundreds or even thousands of servers are needed to perform download tasks. The key is to keep the server stateless.
  - Availability, consistency and reliability: These concepts are at the large of any large system success.
  - Analytics: Collecting and analyzing data are important parts of any system because data is key ingredient for fine-tuning.