LC029 정보검색

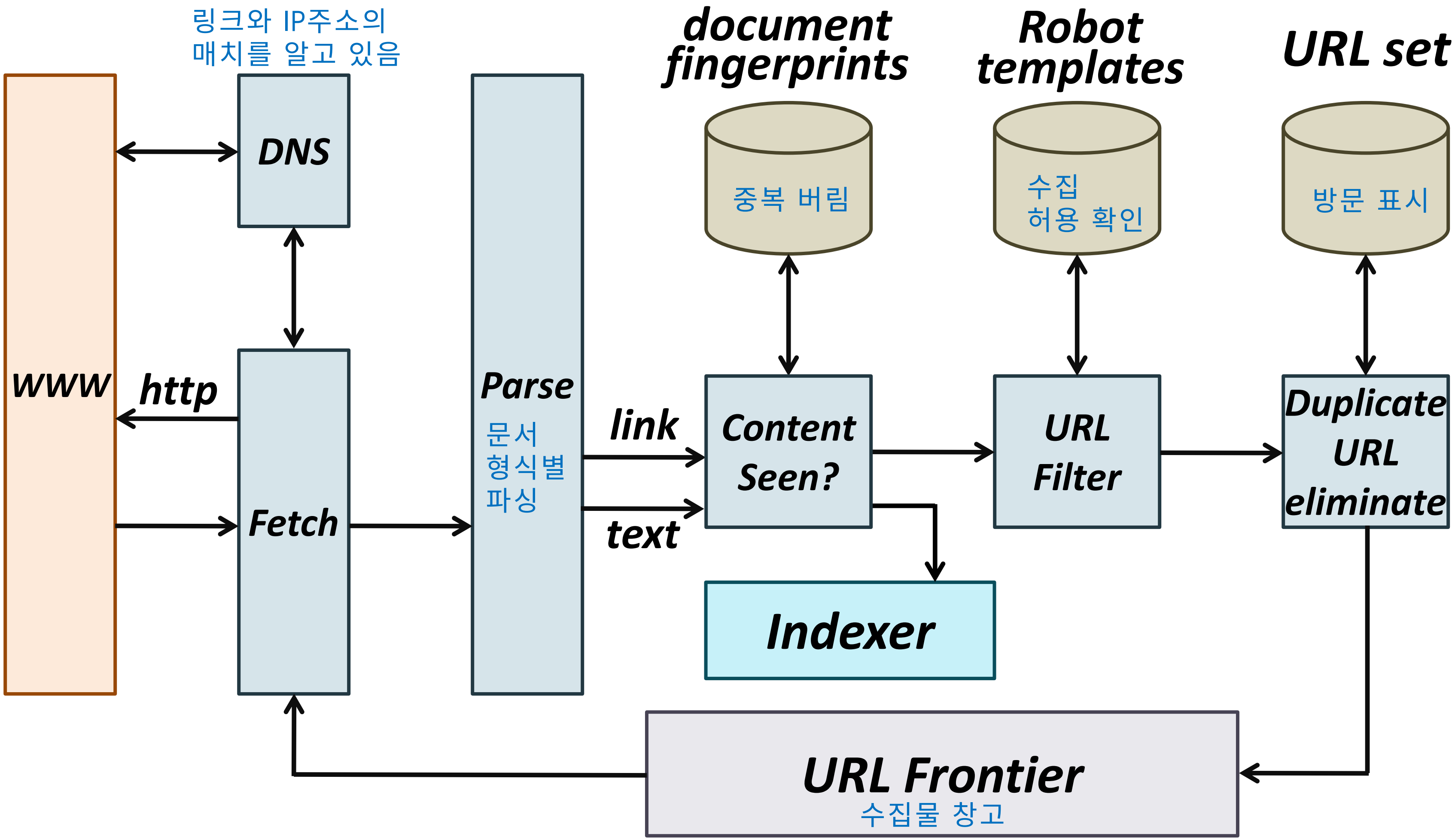
2022년 11월 24일

Chapter 20 : Web crawling and indexes

1. A Crawler
2. Features of a Crawler
3. Distributing the crawler
4. URL Frontier
5. Connectivity Servers
6. Distributing Indexes

A Crawler

Basic Crawler Architecture



Basic crawler operation

- Pick a URL from the frontier.
- Fetch the document at that URL.
- Parse the fetched document.
  - Extract links to other docs (URLs)
- Check if the document 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).

Begin with a seed set.

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

DNS (Domain Name Server)

- Provide a lookup service on the internet.
  - Given a URL, retrieve its IP address. URL 주면 IP 알려줌
- Service provided by a distributed set of servers. Therefore, lookup latencies can be high (even seconds). DNS 서버는 분산되어 있어서 응답까지 지연이 좀 있다
  - DNS caching is used. 캐시 쓴다

한번에 하나씩만 물어볼 수 있다. 뒤에 들어온 요청은 기다린다(block)

- Common DNS implementations are *blocking*.
  - Once a request is made to DNS, other requests are blocked until the first request is completed.
  - Most web crawlers implement their own DNS resolver as a component of the crawler.

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.htm

<A href="other/page.htm">Other Page</A>

- Extracted link is "other/page.htm" which is the same as the absolute URL
  http://en.wikipedia.org/wiki/other/page.htm.
- During parsing, we must normalize such relative URLs.

Content seen?

- Duplication is widespread on the web.
- If the page just fetched is already in the index, do not further process it.
- This is verified using document fingerprints and shingles.

Filters and robots.txt

- **Filters** 특정 패턴을 갖는 URL을 크롤링에서 제외할 수 있음
  - Determine URLs to be excluded from the frontier.

    정규식 사용
  - Regular expressions are used for URLs to be crawled/not.
- **robots.txt**
  - **Robot Exclusion Protocol** for giving spiders ("robots") limited access to a website, originally from 1994.
    For more, see www.robotstxt.org/robotstxt.html.
  - Once a robots.txt file is fetched from a site, do not fetch it repeatedly. (doing so burns bandwidth, hits web server)
  - Therefore, we do cache robots.txt files.

    같은 사이트를 다음에 또 갈 때는 캐시로 저장해둔 로봇텍스트 씀

robots.txt : example

- For a URL, create a file **`URL/robots.txt`**.

```
# robots.txt for http://www.example.com/

User-agent: *
Disallow: /cgi-bin/

User-agent: searchengine
Disallow:
```

- No robot should visit any pages in **`URL/cgi-bin/`**, except a web robot called “searchengine”.

Housekeeping Tasks

- Typically performed by a dedicated thread.
- It wakes up once every few seconds to log crawl 주기적으로 일어나 일하는 스레드 progress statistics (URLs crawled, frontier size, etc)
- Periodically, it takes a snapshot of the crawler's state (say, URL frontier) and stores it on a disk. 백업한다
- On a catastrophic failure, the crawling is restarted from the most recent snapshot (checkpoint).

Features of a Crawler

Features a crawler should provide

- Be **<u>Polite</u>** 공손해져라
  Respect explicit and implicit politeness considerations.
  - **Explicit politeness** 명시적 공손함
    Specifications from webmasters on what portions of site can be crawled. 로봇텍스트를 지키는 건 제작자 양심
    - For this specification, **robots.txt** is used.
  - **Implicit politeness** 묵시적 공손함
    Even with no specification, avoid hitting any site too often. 특정 사이트 너무 많이 방문하지 않기 → DDOS 됨
    - Only one connection should be open to any given host at a time.
    - A waiting time of a few seconds should occur between successive requests to a host.

Features a crawler should provide

- Be **<u>Robust</u>** 강건하기
  Be immune to **spider traps** and other malicious behavior from web servers. 함정에 걸리지 말기
  예 : 서로 링크된 문서만 무한반복하지 않기
- Be **<u>Scalable</u>** 컴퓨터를 한번에 여러 대 돌려도 정상작동해야 함
  Designed to increase the crawl rate by adding more machines.
- 분산 처리 가능
  Be capable of **<u>Distributed operation</u>**
  Designed to run on multiple distributed machines.
- **<u>Performance/Efficiency</u>** 가성비
  Permit full and efficient use of available processing and network resources.

Features a crawler should provide

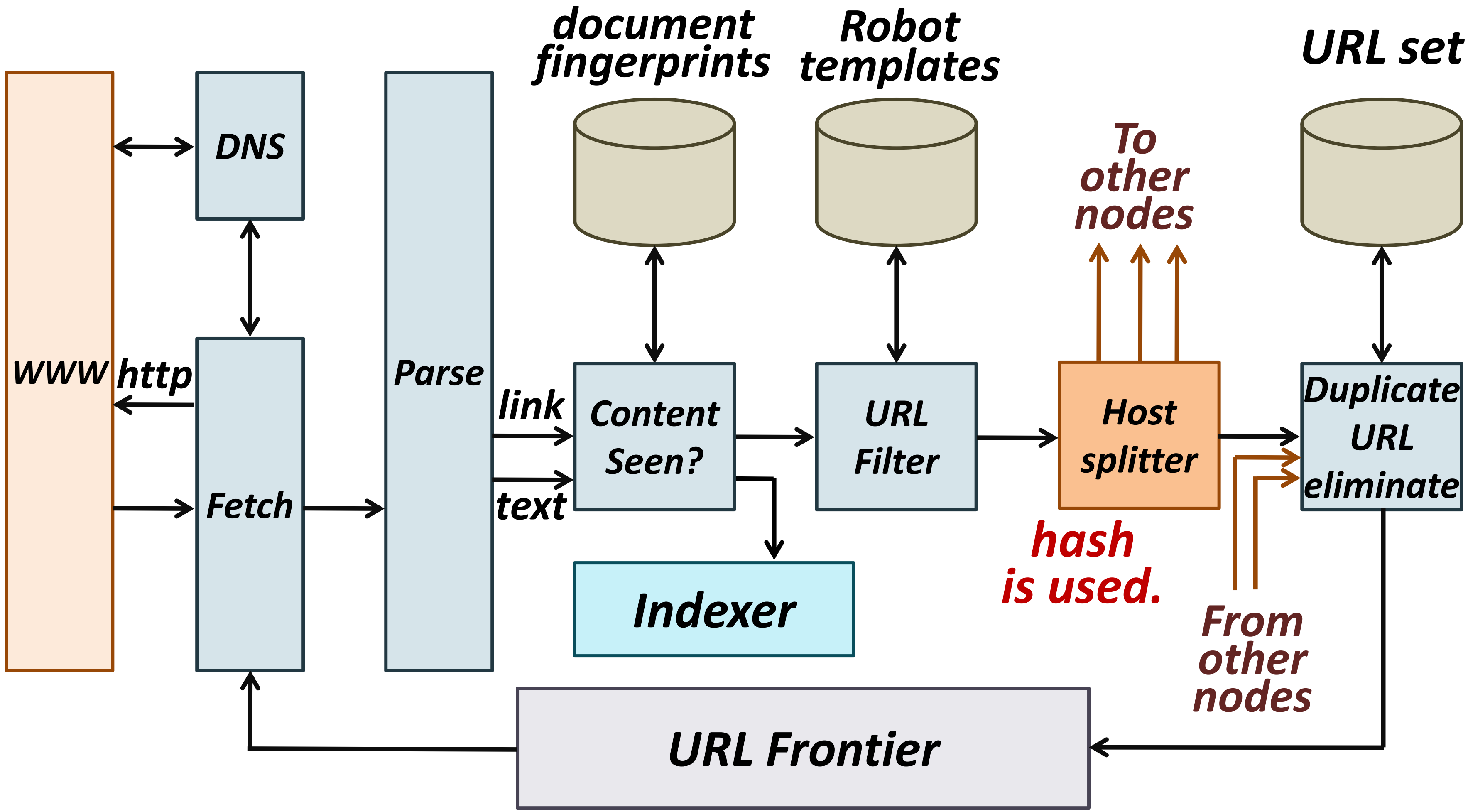
- Fetch pages of **higher quality first**. 고급 문서 먼저 fetch하기
- **Continuous operation** 내용이 바뀐 문서 업데이트하기
  Continue fetching fresh copies of a previously fetched page, so that the search engine's index is fairly current.
- **Extensible** 다양한 문서를 다룰 수 있어야 한다
  Adapt to new data formats, protocols, etc.

Distributing the crawler

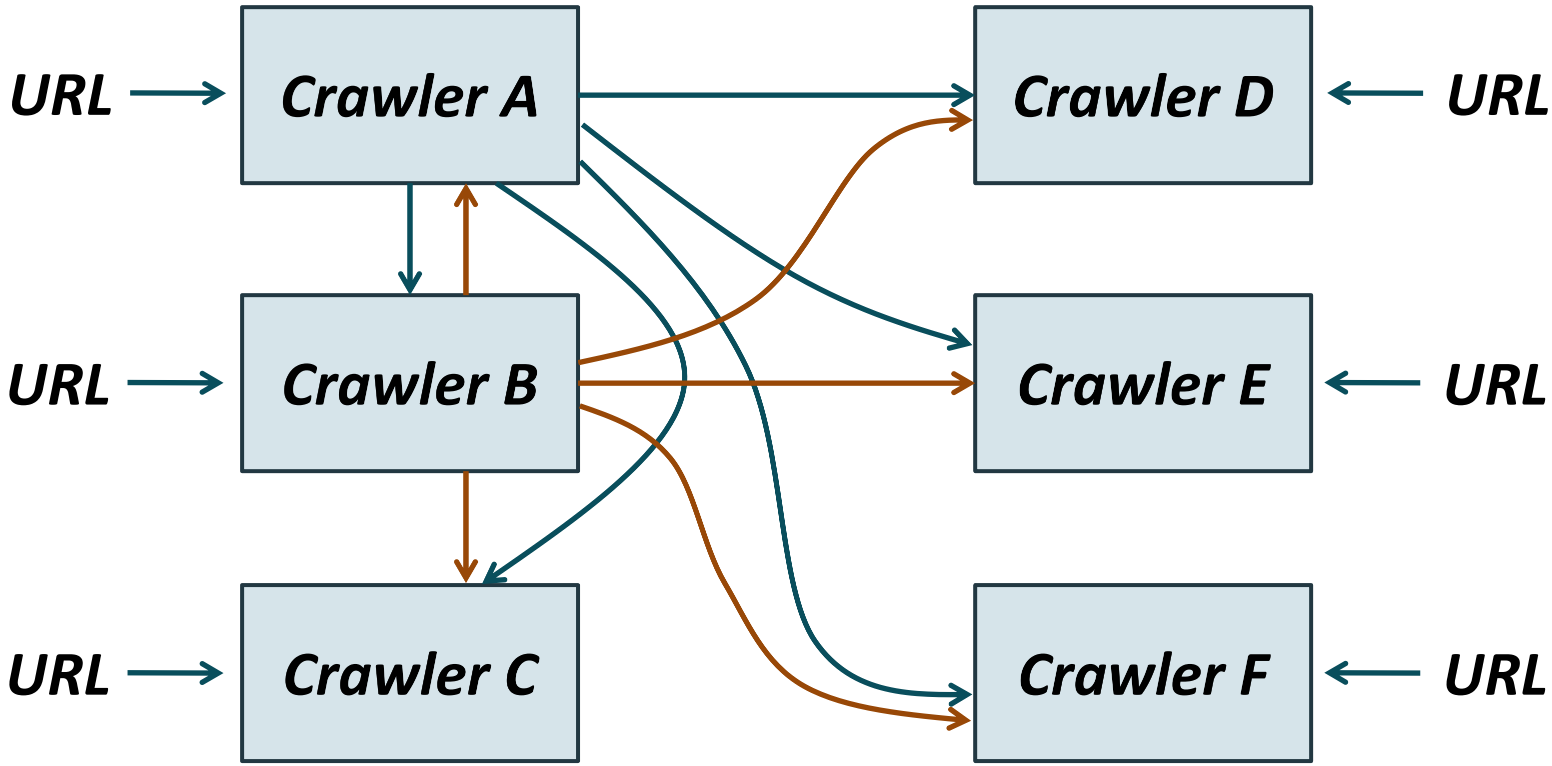
Distributing the crawler

- Run multiple crawl threads, under different processes, potentially at different nodes. 컴퓨터
  - Geographically distributed nodes
- **Host Splitter** does partition hosts being crawled into nodes. 지금 크롤링되고 있는 웹사이트를 여러 대의 컴퓨터에서 처리할 수 있도록 URL을 분산시킴
  - Hash used for partition. 해시 사용
- Distributing the crawler is essential for **scaling**.

Distributing the crawler



Distributing the crawler



URL Frontier

URL Frontier

- Contain URLs to be fetched in the current crawl.
  - Can include multiple pages from the same host.
- Provide a URL in some order when a crawler thread seeks one.
  - Must avoid trying to fetch them all at the same time.
  - Must try to keep all crawling threads busy.

URL Frontier: two main considerations

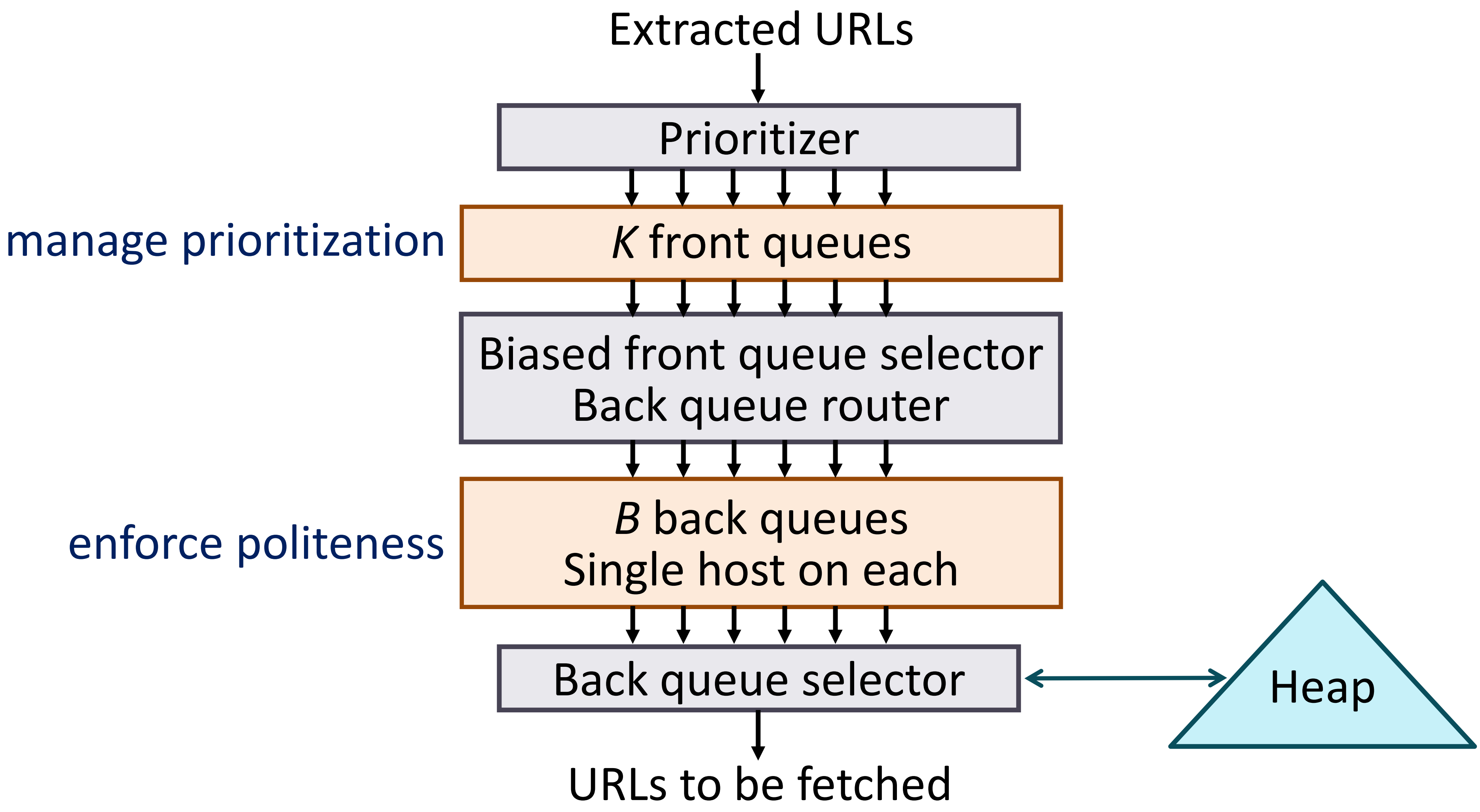
- **Freshness**
  Crawl high-quality pages more often than others.
  - Pages (such as news sites) whose content changes often get high priority.
- **Politeness** 같은 사이트 너무 자주 방문하지 않기
  Do not hit a web server too frequently.
  - Even if we restrict only one thread to fetch from a host, the crawler can hit it repeatedly.
  - So, a waiting time of a few seconds is inserted between successive requests to a host.

Implementation of URL Frontier

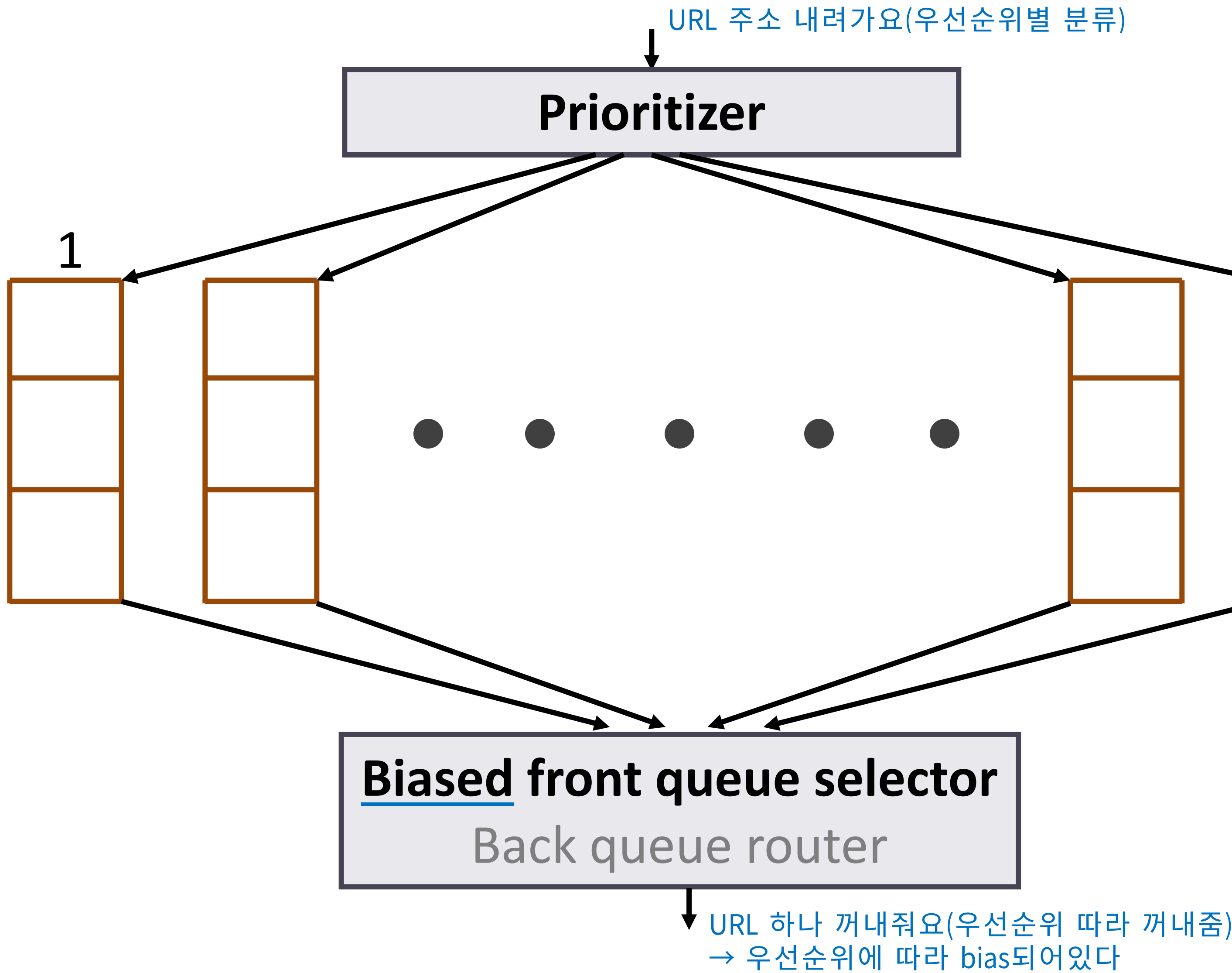
- Simple priority queue fails because these goals may conflict each other.
  - Crawl high-quality pages more often than others.
  - Do not hit a web server too frequently.

  Note that many links out of a page go to its own site, creating a burst of accesses to that site.

Implementation of URL Frontier



Front Queues



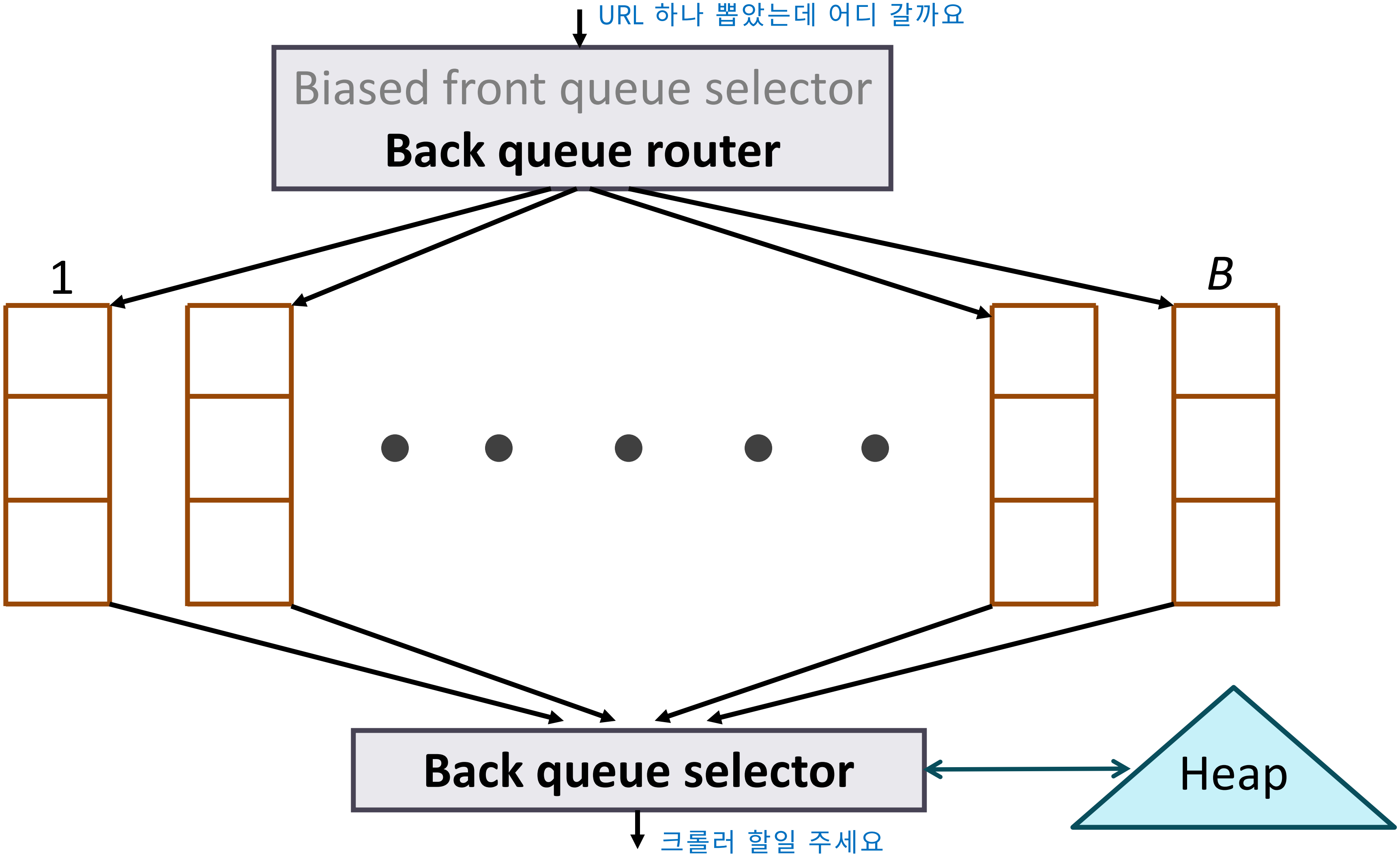
Front Queues

- Prioritizer assigns to URL an integer priority between 1 and *K*.
  - Appends a new URL to the corresponding queue.
- Heuristics for assigning priority 우선순위 부여 휴리스틱
  - Refresh rate sampled from previous crawls.
    과거 크롤링 데이터 기반
  - Application-specific 용도에 따라
    e.g. crawl news sites more often.

Biased Front Queue Selector

- When a back queue requests a URL,
  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



URL 하나 뽑았는데 어디 갈까요
Biased front queue selector
Back queue router
1
B
Back queue selector
Heap
크롤러 할일 주세요

Back Queues

- Each back queue is kept non-empty while the crawl is in progress.
- Each back queue only contains URLs from a single host.
  - Maintain a table from hosts to back queues.

Host name	Back queue
stanford.edu	23
microsoft.com	47
sungshin.ac.kr	*12*

The Heap

- One entry for each back queue. 백큐 하나당 노드 하나씩 갖는다
- The entry is the earliest time t_e at which the host corresponding to the back queue can be hit again.
- This earliest time is determined from 방문 쿨타임이 적게 남은 것부터 먼저 나온다
  - Last access to that host.
  - Any time buffer heuristic we choose.

매번 쿨타임이 가장 적게 남은 것부터 꺼내줘도 크롤러가 기다리는 시간이 필요할 수도 있다
: 크롤러가 일 다 하고 왔는데 다음 방문지 쿨타임이 남은 경우

Back Queue Processing

- When a crawler thread seeking a URL to crawl,
  - Extracts the root of the heap.
  - Wait until the corresponding time entry t_e.
- Fetches URL at head of corresponding back queue *q*.
- Create a new heap entry for the URL.
- Checks if queue *q* is now empty.
  - if so, pulls a URL *v* from front queues.
  - if there is a back queue ***q'*** for *v*'s host, append ***v*** to ***q'*** and pull another URL from front queues, repeat until ***v* for *q*** is found.
  - add *v* to *q*

크롤러가 URL을 하나 가져가면 큐에 빈칸이 생기니까 다시 큐를 채우도록 요청해야 한다
URL을 하나 달라고 요청할 수는 있지만, 어느 URL을 달라고는 요청할 수 없다
될 때까지 요청한다

Number of Back Queues *B*

- Keep all threads busy while respecting politeness.
- Mercator recommendation
  Three times as many back queues as crawler threads.

백큐 개수는 크롤러가 사이트 방문 쿨타임을 지키면서 바쁘게 일할 수 있게 경험적으로 정한다

Size of URL Frontier

- On a web-scale crawl, the URL frontier may grow too big to reside in memory. 웹 스케일 크롤링하다보면 프론티어가 너무 커질 수 있다
- Solution
  프론티어의 대부분 내용은 디스크에 둔다
  - Let most of the URL frontier reside on disk.
  - A portion of each queue is kept in memory, with more brought in from disk as it is drained in memory.

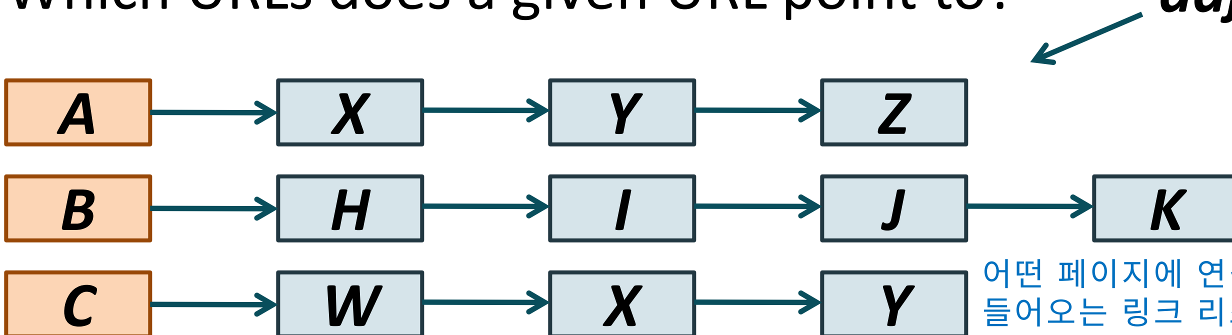
  프론티어의 일부는 메모리로 가져와서(일정량 유지) 큐로 준다

Connectivity Servers

Connectivity Server

- Support for fast queries on the web graph
  - Which URLs point to a given URL?
  - Which URLs does a given URL point to?

adjacency table



어떤 페이지에 연결된 다른 페이지 리스트
들어오는 링크 리스트 따로
나가는 링크 리스트 따로 있음

- Applications
  - Web graph analysis for sophisticated crawl optimization
  - Link analysis

Connectivity Server

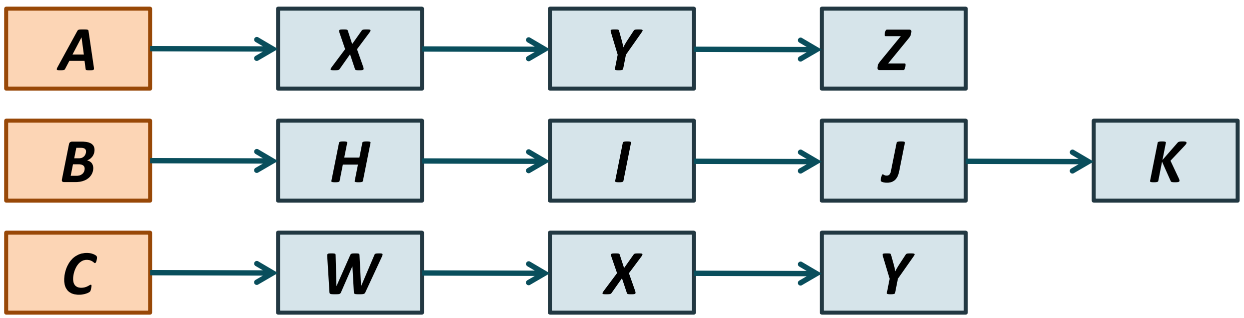
- Store the adjacency table in memory
  - From URL to outlinks
  - From URL to inlinks
- Memory requirement 문서가 하도 많아서 리스트도 용량이 많이 필요함
  - Assume that the Web had 4 billion pages, each with ten links to other pages.
  - Assuming each URL represented by an integer, we need 32 bit integer per node.

  $4 \times 10^9 \times 10 \times 4 \times 2 = 3.2 \times 10^{11} = 3.2 \times 10^2$ TB
- Thus, compressing the adjacency table is critical.

압축을 잘 해야 한다

Compression of Adjacency Table

- Properties exploited in compression:
  - **Similarity** (between lists) 비슷한 링크 서로 참조하기



From URL to outlinks

  - **Locality** (many links from a page go to “nearby” pages)
    - Use gap encodings in sorted lists

지역성 : 성신포탈, 교육시스템 등 가까이 붙어다니는 링크도 있지

Compression of Adjacency Table

nate.com/a.htm	nate.com/w.htm, naver.com/x.htm
nate.com/b.htm	nate.com/cafe/boss.htm, nate.com/y.htm
nate.com/c.htm	nate.com/cafe/boss.htm, naver.com/x.htm
nate.com/d.htm	nate.com/w.htm, nate.com/y.htm, ever.com/z.htm

Adjacency Table
(From URL to outlinks)

Compression of Adjacency Table

- Consider lexicographically ordered list of all URLs.

1	www.stanford.edu/alchemy
2	www.stanford.edu/biology
3	www.stanford.edu/biology/plant
4	www.stanford.edu/biology/plant/copyright
5	www.stanford.edu/biology/plant/people
6	www.stanford.edu/chemistry

- To each URL, a unique integer identifier is assigned.

Compression of Adjacency Table

nate.com/a.htm	nate.com/w.htm, naver.com/x.htm
nate.com/b.htm	nate.com/cafe/boss.htm, nate.com/y.htm
nate.com/c.htm	nate.com/cafe/boss.htm, naver.com/x.htm
nate.com/d.htm	nate.com/w.htm, nate.com/y.htm, ever.com/z.htm

1	nate.com/a.htm
2	nate.com/b.htm
3	nate.com/c.htm
4	nate.com/d.htm
5	nate.com/cafe/boss.htm
6	nate.com/w.htm
7	naver.com/x.htm
8	nate.com/y.htm
9	ever.com/z.htm

⇩

1	6, 7
2	5, 8
3	5, 7
4	6, 8, 9

Compression of Adjacency Table

- Main idea : **Similarity** 서로 비슷하게 구성된 리스트는 참조하자
  The adjacency list of a node is <u>similar</u> to one of the <u>7 preceding URLs</u> in the lexicographic ordering.

- Consider the following adjacency table.

1	1, 2, 4, 8, 16, 32, 64
2	1, 4, 9, 16, 25, 36, 49, 64
3	1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144
4	1, 4, 8, 16, 25, 36, 49, 64

Compression of Adjacency Table

- Express adjacency list in terms of one of the 7 preceding URLs.

1	1, 2, 4, 8, 16, 32, 64
2	**1, 4, 9, 16, 25, 36, 49, 64**
3	1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144
4	**1, 4, 8, 16, 25, 36, 49, 64**

offset -2, remove 9, add 8

2칸 앞 리스트에서 9 빼고 8 넣어라

Compression of Adjacency Table

- Main Idea : **Locality**
  Use gap encodings in sorted lists.

1	1, 2, 4, 8, 16, 32, 64
2	**1, 4, 9, 16, 25, 36, 49, 64**
3	1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144
4	**1, 4, 8, 16, 25, 36, 49, 64**

1	1, +1, +2, +4, +8, +16, +32
2	1, +3, +5, +7, +9, +11, +13, +15
3	1, +1, +1, +2, +3, +5, +8, +13, +21, +34, +55
4	offset -2, remove 9, add 8

Compression of Adjacency Table

- How to process connectivity queries?

1	1, +1, +2, +4, +8, +16, +32
2	1, +3, +5, +7, +9, +11, +13, +15
3	1, +1, +1, +2, +3, +5, +8, +13, +21, +34, +55
4	offset -2, remove 9, add 8

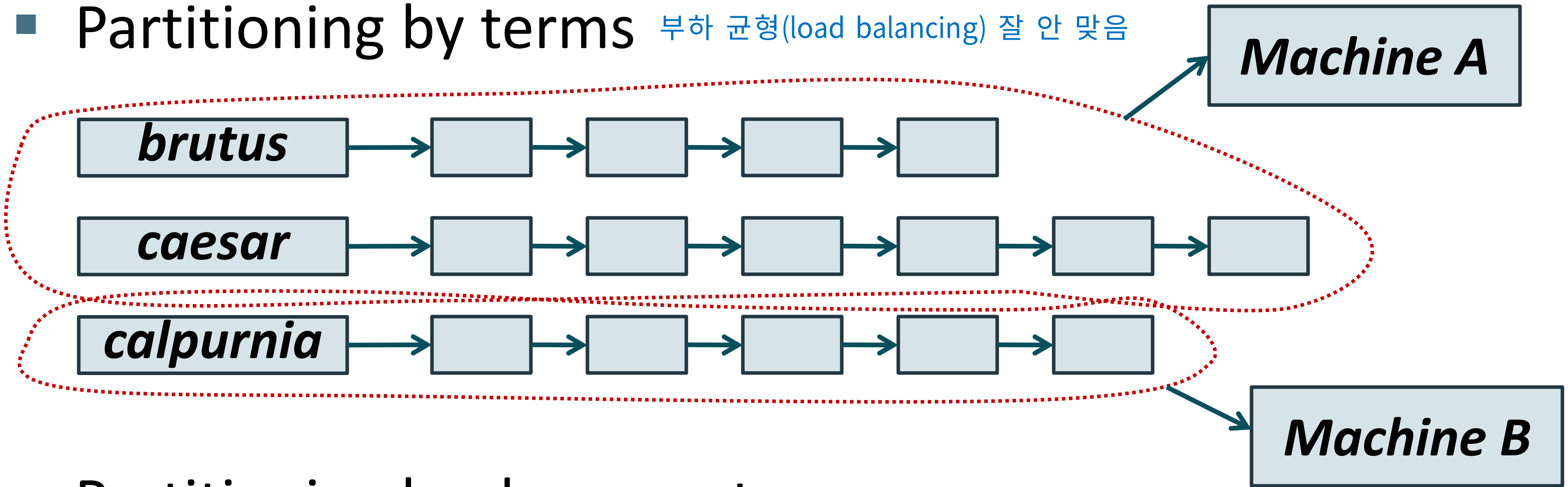
- We need to reconstruct the entries of the adjacency table.

리스트 항목에 쓸 바이트 수를 줄일 수 있지만
나중에 참고할 때 다시 계산해서 리스트를 복구해야 함

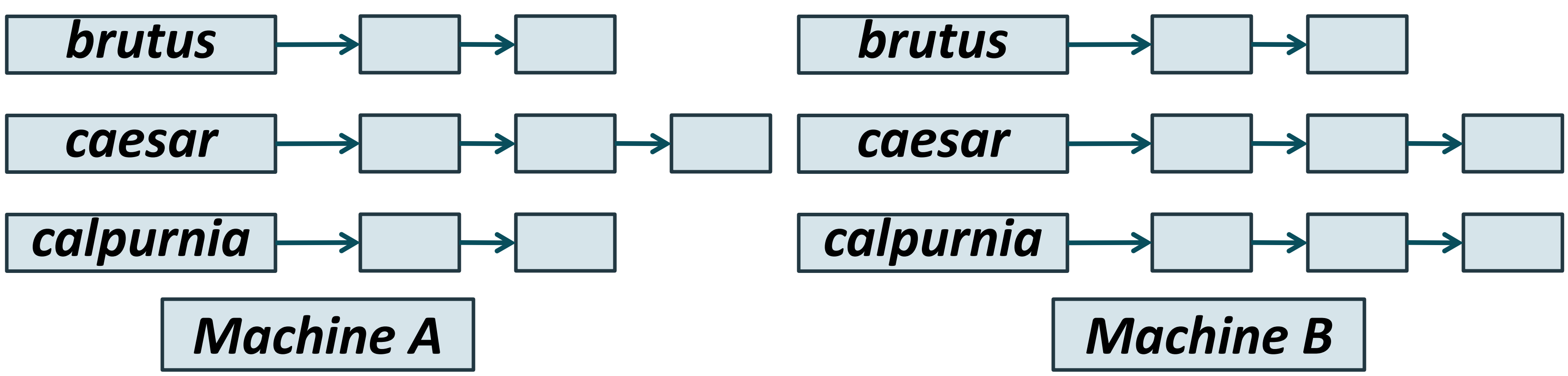
Distributing Indexes

Distributing Indexes

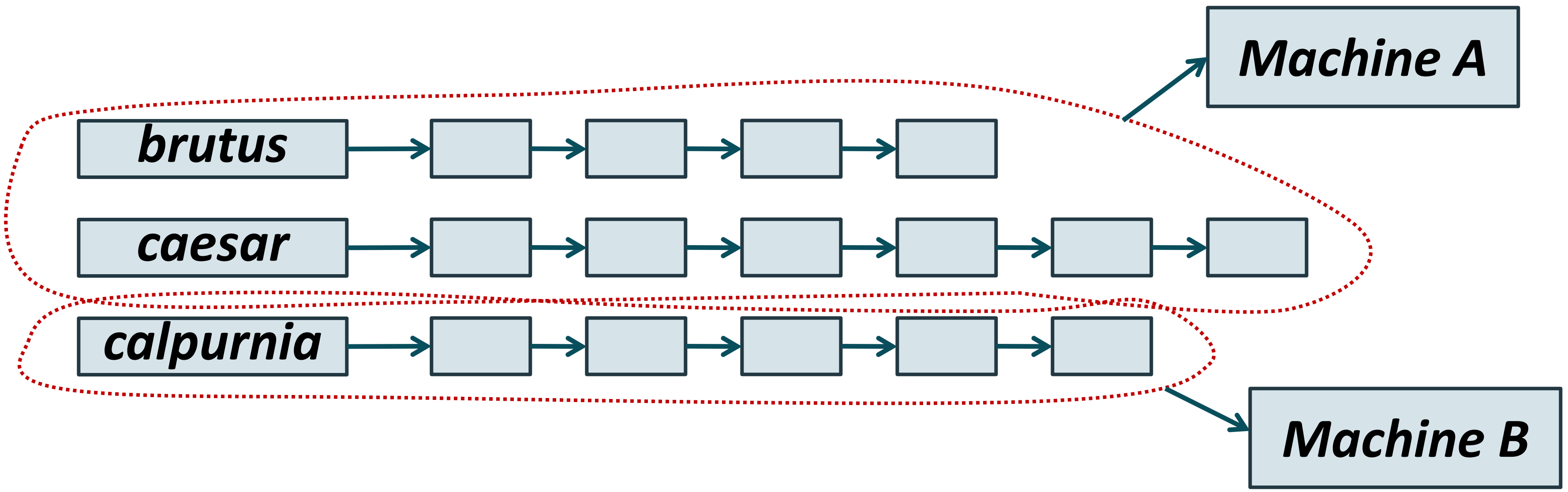
- Partitioning by terms 부하 균형(load balancing) 잘 안 맞음



- Partitioning by documents



Partitioning by terms



Q1 **brutus**

Q2 **calpurnia**

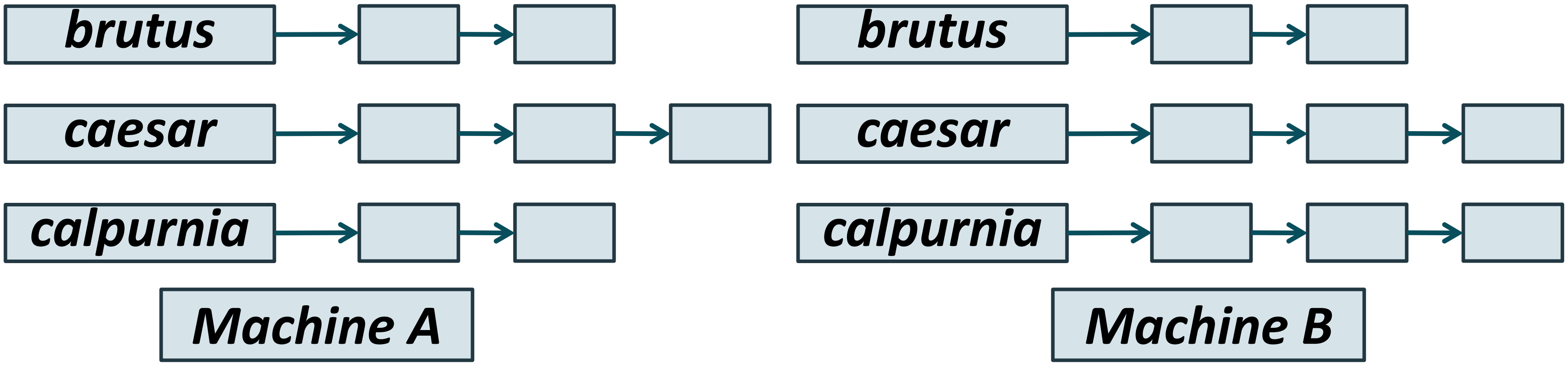
Q3 **brutus AND calpurnia**

Partitioning by terms

- The dictionary of index terms is partitioned into subsets.
- A query is routed to the nodes corresponding to its query terms.
- **Pros** 서로 다른 질의가 들어왔을 때 각 기기에서 동시에 처리 가능
  - Concurrent query processing is possible since different query terms would hit different set of machines.
- **Cons**
  - Multi-word query processing is inefficient due to the overhead of sending long postings list between machines for merging. 복합 질의 비효율적
  - Load balancing is difficult since it depends on the distribution of query terms not on term frequency.

load 관리 어려움

Partitioning by documents



Q1 brutus

Q2 calpurnia

Q3 brutus AND calpurnia

Partitioning by documents

- Each node contains the index for a subset of all documents.
  - Each query is distributed to all nodes.
  - The results from various nodes are merged.
- Multi-word query processing is rather efficient since the merged lists are moving between machines for final merging.

장점 : 복합 질의 연산할 때 각자 계산 후 보내기 때문에 기기당 계산량 적음
단점 : 리스트를 자꾸 주고받아야 함

Partitioning by documents

- How to partition documents to nodes?
  - According to crawling architecture, one simple approach would be to assign *all pages from a* ***host*** to a single **node**.
    - At query time, the top *k* results from each **node** being merged to find the top *k* document for the query.
    - The problem is that search result would come from a small number of **hosts**.
  - A hash of each URL into the space of nodes results in a more **uniform distribution** of query-time computation across nodes. 호스트 스플리터가 하는 일

Distributing the crawler