Lecture 2:
Web Crawling

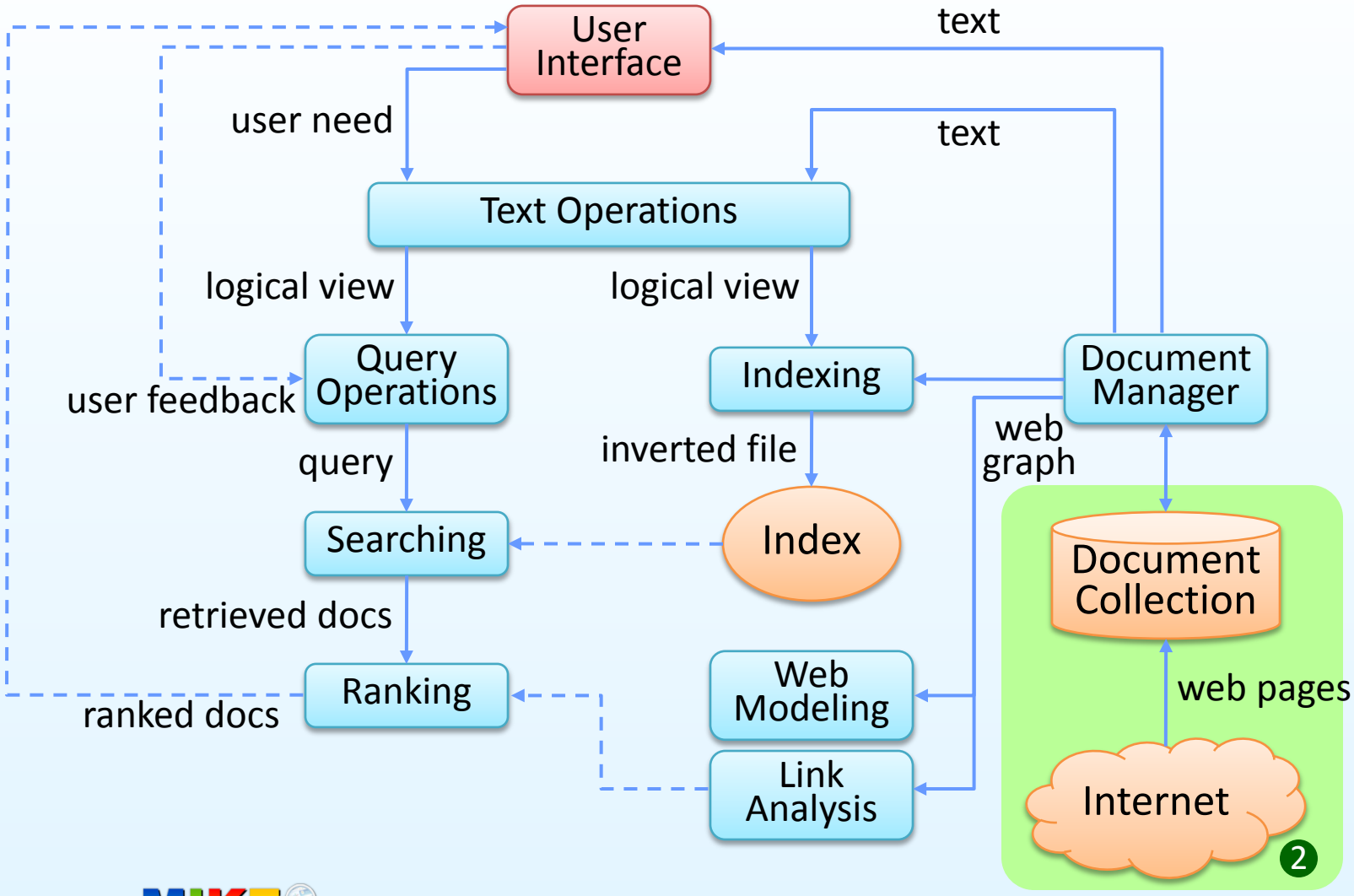
01204453 Web Information Retrieval and Mining

Department of Computer Engineering
Faculty of Engineering, Kasetsart University
Bangkok, Thailand.

Department of
Computer Engineering
Kasetsart University

MIKE LABORATORY

Review: Search Engine Architecture



How can we obtain a web page?

- Windows
  - Save as a web page from a web browser
- Text-based Linux
  - Retrieve a web page using the lynx or wget commands

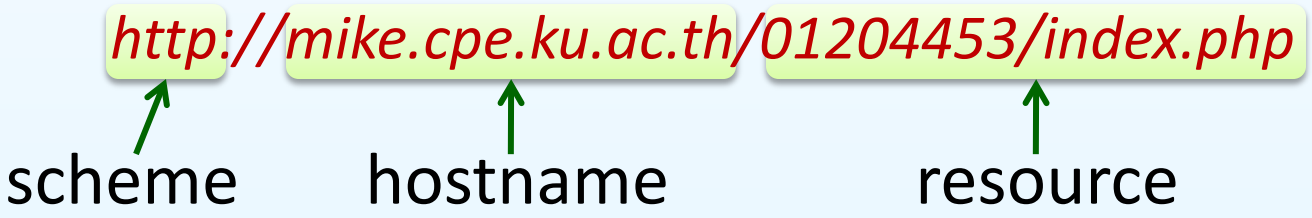
Web Crawler

- A software for automatic downloading web pages from the Internet.
- Also known as web spider or web robot

- Cycle of a crawling process
  - Start downloading with a set of seed URLs
  - Extract links and add new ones to a central queue
  - Select a new page from queue and download it
  - Repeat the process until a stop criterion is met

Retrieving Web Pages

- Every page has a unique uniform resource locator (URL).
- Web pages are stored on web servers that use HTTP to exchange information with client software.
- e.g.,



Retrieving Web Pages (Cont.)

1. A client program connects to a domain name system (DNS) server.
2. DNS server translates the hostname into an internet protocol (IP) address.
3. Client then attempts to connect to server host using specific port.
4. After connection, client sends an HTTP request to the web server to request a page.
   - usually a GET request

The lynx Command

- Non-graphical web browser
- Can dump the retrieved content in a clean text format
- Also, provide a list of out-going links

- Simple usage:

```
lynx [option] [URL]
Option: -dump       dump content in text format
        -timeout=N  set maximum timeout for waiting
                    for a page
```

The lynx Command (Cont.)

- Benefits
  - Automatically solve DNS resolution
  - Provide already cleaned content
  - Complete relative addresses
- Drawbacks
  - No access to metadata
  - No access to anchor text
  - Do not handle pages with frameset
- One line to get the links from a page:

```
lynx -dump [URL] | grep -e "[0-9]*\.http://.*"
```

The wget Command

- A Linux application designed to
  - Retrieve a web page, or
  - Follow links up to a certain depth

- Simple usage:

```
wget [option] [URL]
Option: -O file    specify the target output file
        -r         recursive downloading
        -l depth   follow links only up to the
                   depth of depth (default 5)
        -T seconds set timeout
```

The wget Command (Cont.)

- Benefits
  - Provide unmodified content (original format)
  - Recursively download pages
  - Keep the stored file structure

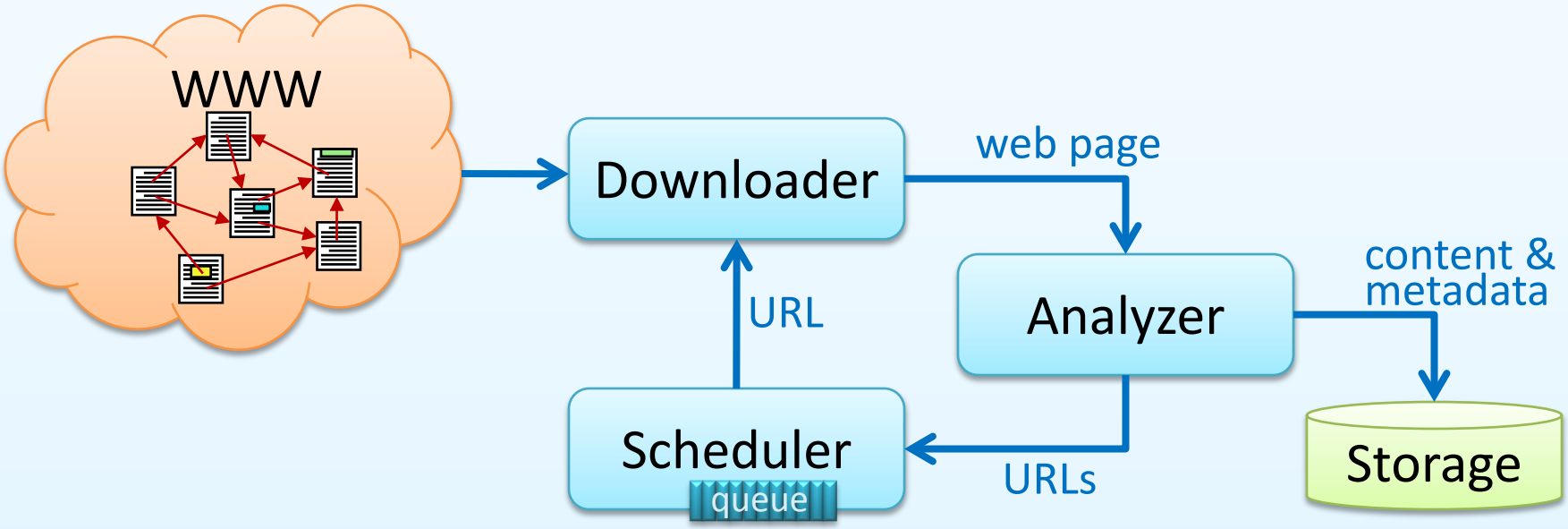
Architecture and Implementation

A Basic Crawling Algorithm

- Initialize a queue of URLs ("seed" URLs)
- Repeat
  - Remove a URL from the queue
  - Fetch associated page
  - Parse and analyze page
  - Store representation of page
  - Extract URLs from page and add to queue
  - Continue until no more new URL or meeting other criterion

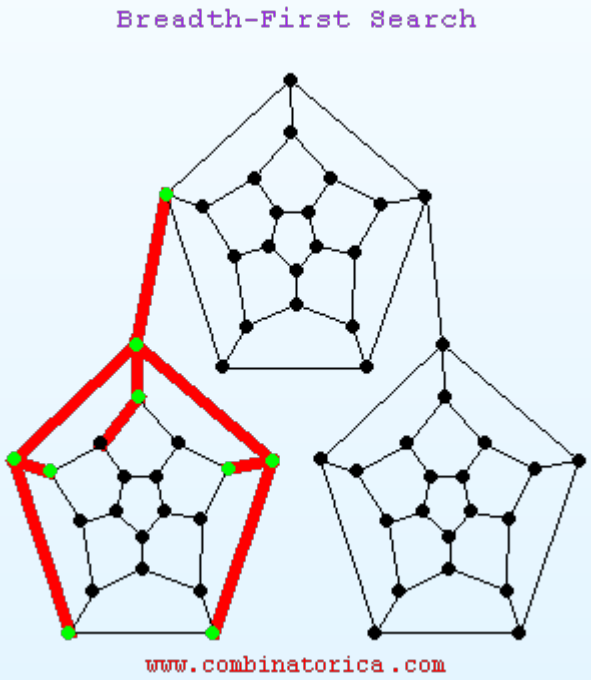
Architecture of a Crawler

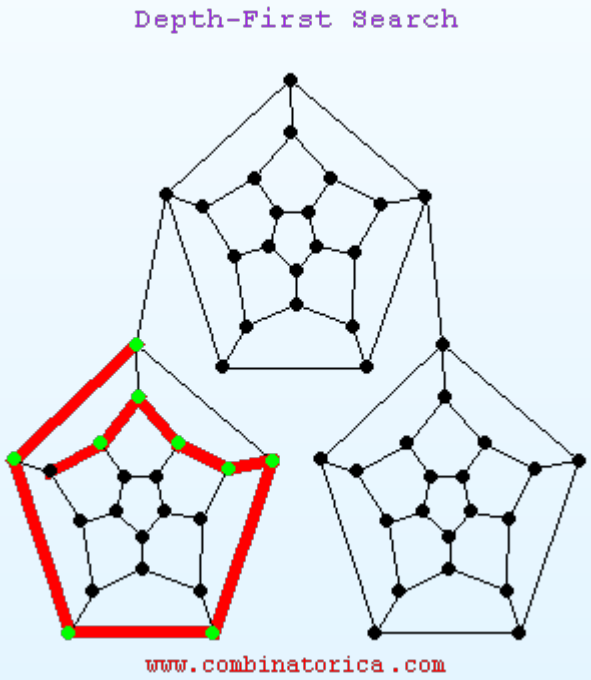
- Compose of 4 main modules
  - Scheduler: maintain a queue of URLs to visit
  - Downloader: download web pages
  - Analyzer: analyze content and links
  - Storage: store content and metadata



Queuing Strategies

- 2 basic scheduling based on graph traversal algorithm:
  - FIFO—append to end of Q
    - i.e., breadth first search (BFS)
  - LIFO—add to front of Q
    - i.e., depth first search (DFS)





BFS & DFS: Trade-off

- Breadth first search
  - Pros: uniformly explore outward from the root page
  - Cons: require much memory, i.e., exponential in depth
  - Standard crawling method

- Depth first search
  - Pros: require memory of only depth times, i.e., linear in depth
  - Cons: get lost pursuing a single thread

What any crawlers must do?

- Politeness
  - Implicit: avoid several requests to the same web server
  - Explicit: retrieve only allowed pages, i.e., the robots.txt file
- Robustness
  - Handle crawler traps

Example of Crawler Traps

- Creation of indefinitely deep directory structures
  - e.g., *http://foo.com/bar/foo/bar/foo/bar/foo/bar/.....*

```
.
|-- bar
|   |-- foo -> ../foo
|   `-- index.html
`-- foo
    |-- bar -> ../bar
    `-- index.html
```

link-to-foo

link-to-bar

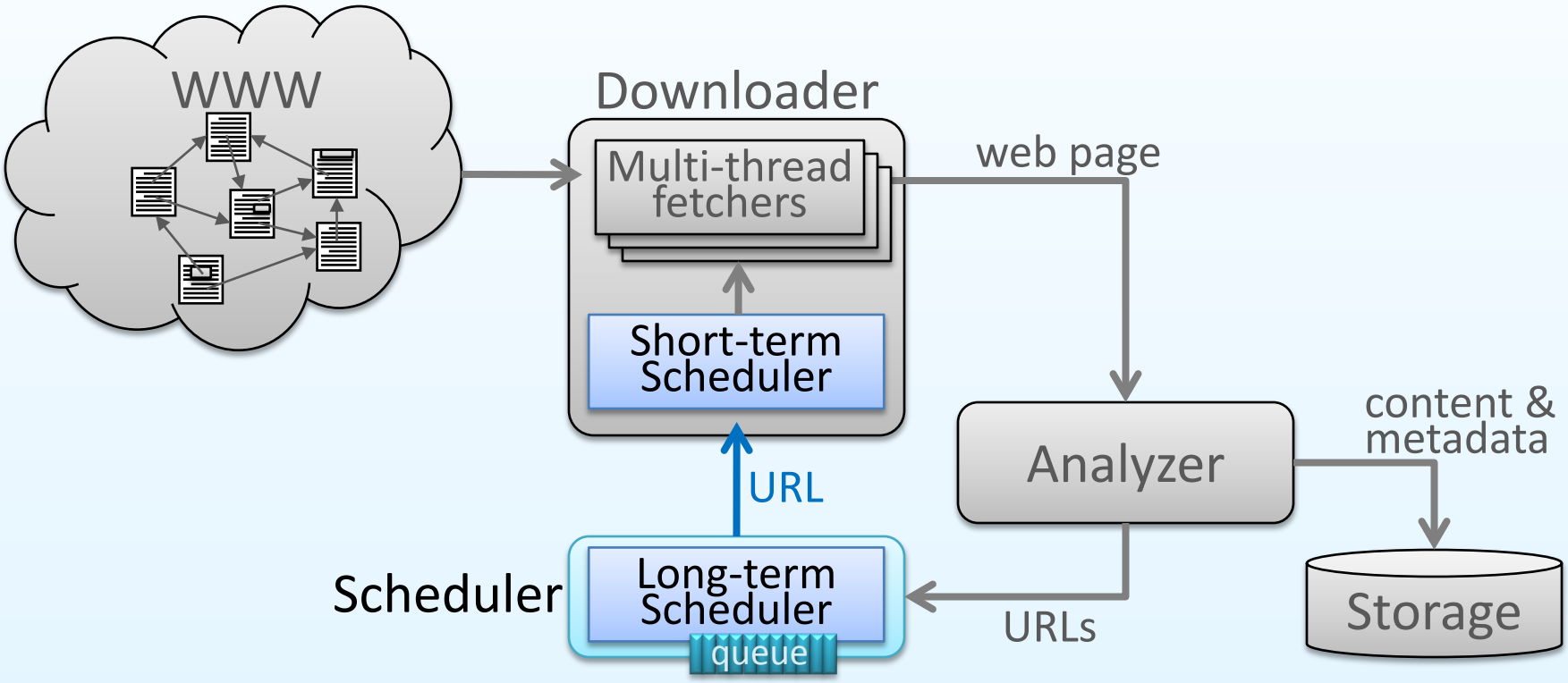
- Dynamic pages that produce an infinite number of pages
  - e.g., a calendar page, http://www.timeanddate.com/calendar/?country=68

What any crawlers should do?

- Scalability
  - Design to run on multiple distributed machines
  - Can increase the crawl rate by adding more machines
- Efficiency
  - Permit full utilization of available processing and network resources
  - First crawl good information pages, and avoid spam ones
- Extensibility
  - Adapt to new data format, protocols, future technology, etc.

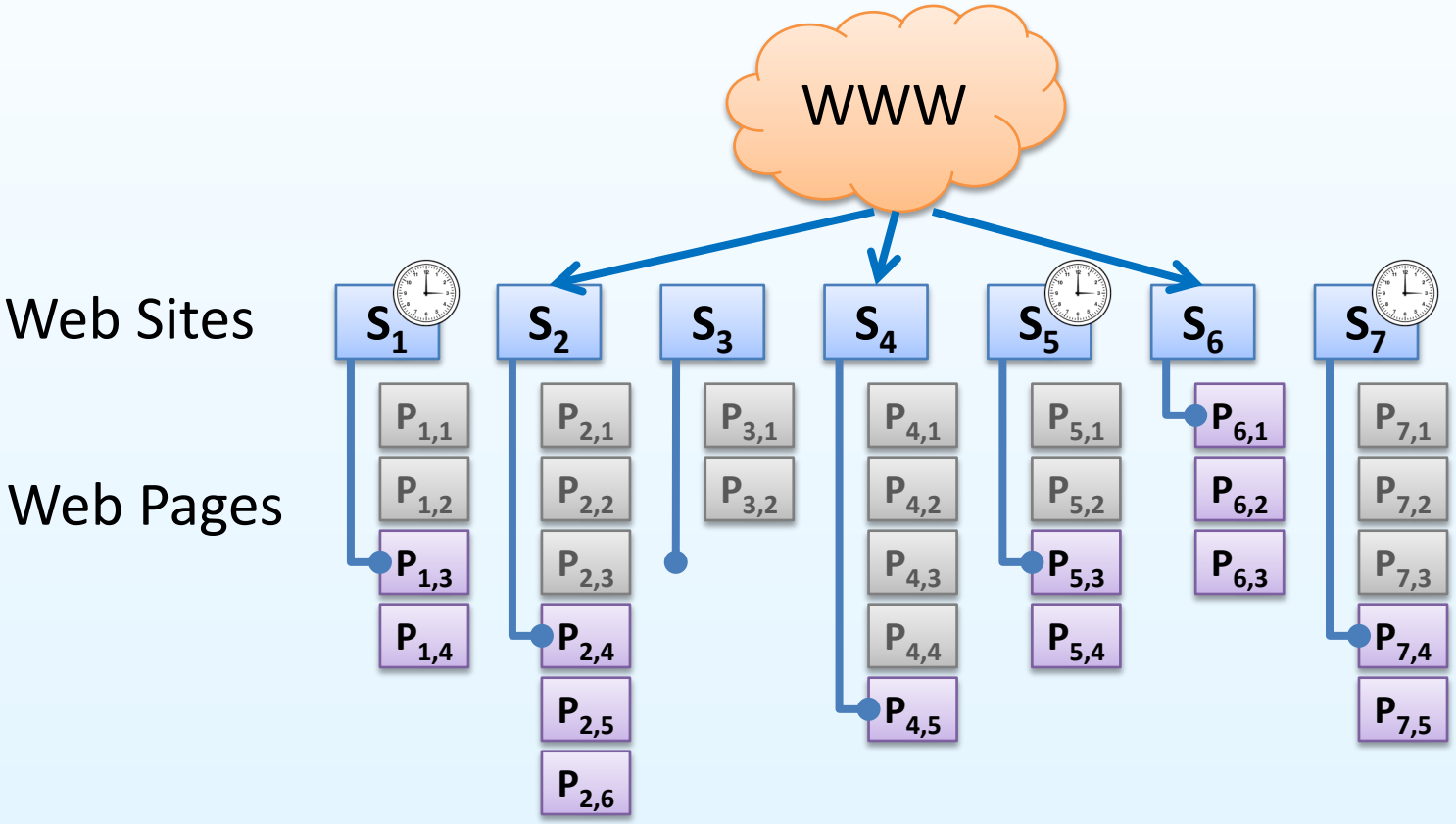
Designing Scheduler

- The scheduling can be divided into two parts:
  - Long-term scheduling: decide which pages to visit next
  - Short-term scheduling: re-arrange pages to fulfill politeness



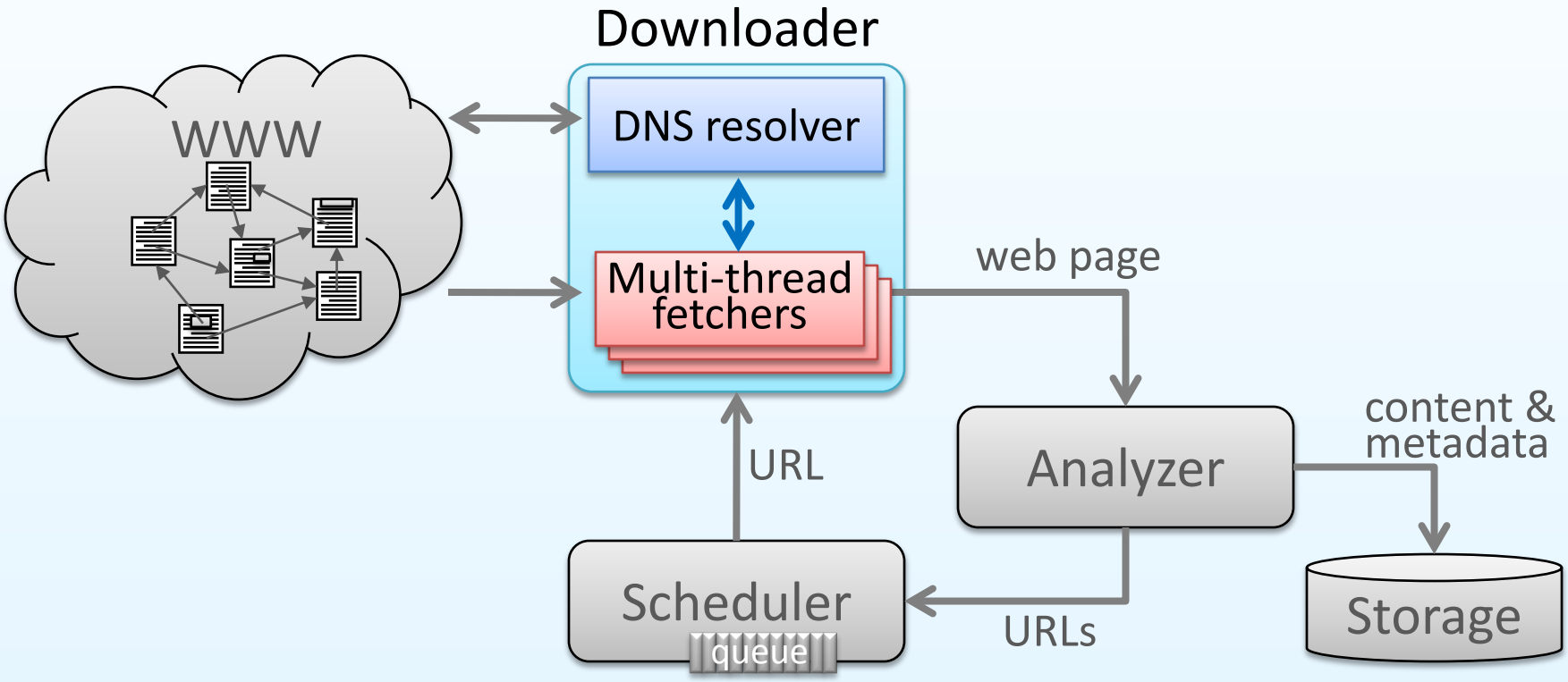
Designing Scheduler (Cont.)

- The short-term schedule
  - Require maintaining several queues, i.e., one for each site
  - List pages to be downloaded in each queue

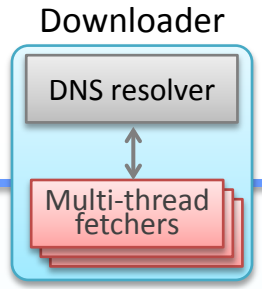


Designing Downloader

- Multi-thread fetchers are needed to increase throughput.
- DNS is needed resolved a IP address.

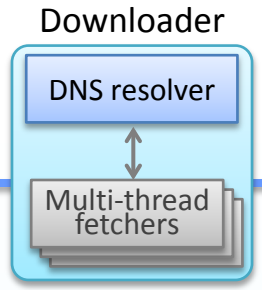


Multi-Thread Fetchers



- Avoid network bottleneck
  - Delay in downloading individual page
- Request each page from a different server in parallel
  - Maximize throughput and avoid overloading any single server

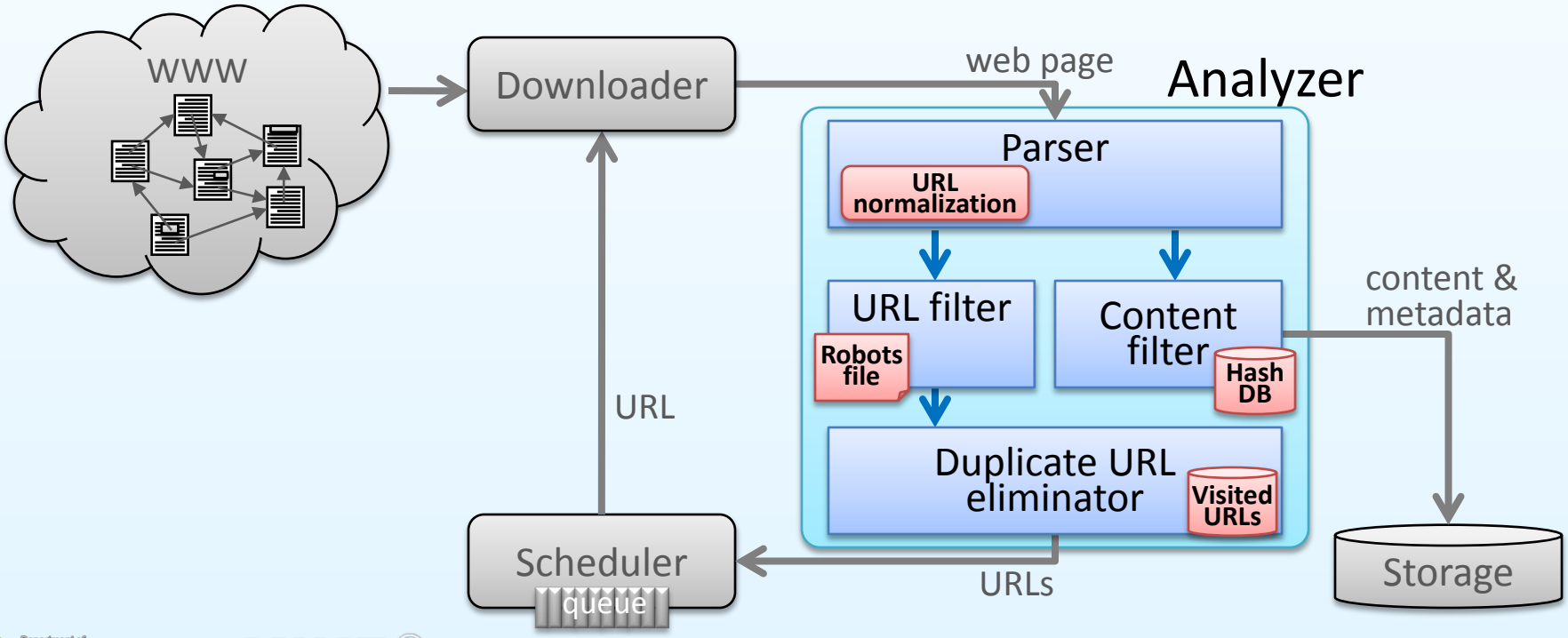
DNS Resolver



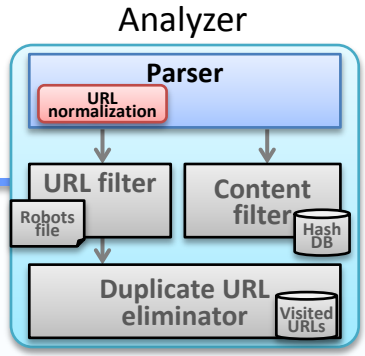
- A lookup service on the Internet
  - Retrieve the IP address corresponding to a given hostname
  - Spend highly variable latencies due to distributed sets of servers
- Common OS implementations of DNS looking are blocking
  - Only one outstanding request at a time
- Solutions:
  - DNS caching
  - Batch DNS resolver—collect requests and send them out together

Designing Analyzer

- In practice, analyzer composes of several sub-modules:
  - Parser: analyze page content and links
  - Content filter: avoid storing redundant content
  - URL filter: explicitly respect politeness, i.e., robots.txt
  - Duplicate URL eliminator: avoid enqueuing redundant URLs



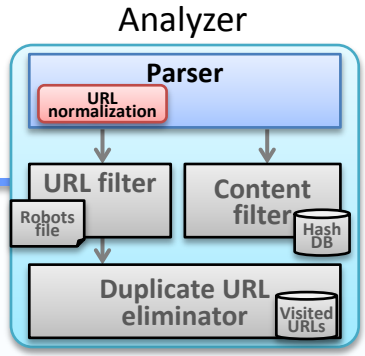
Parser



- Parse content into a specify format
- Extract links contained in that page

- Several kinds of embedded links
  - Visible link
    - e.g., *anchor text*
  - Frame
    - e.g., *<frame src="**srcOfFrame**">*
  - Image
    - e.g., *<img src="**srcOfImage**">*
  - etc.

Parser: URL Normalization



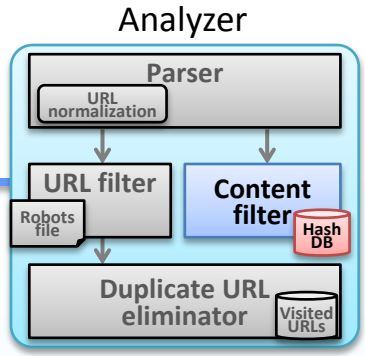
- Many of extracted links are relative URLs.

http://mike.cpe.ku.ac.th/01204453/index.html

- URLs can end with "/".
- URLs can contain a self-reference "#".
- URL normalization (canonicalization) is a module for completing a relative URL to the absolute one.

../index.html	→ *http://mike.cpe.ku.ac.th*
home.html	→ *http://mike.cpe.ku.ac.th/01204453/home.html*
download/data.html	→ *http://mike.cpe.ku.ac.th/01204453/download/data.html*
/	→ *http://mike.cpe.ku.ac.th*
index.html#sub	→ *http://mike.cpe.ku.ac.th/01204453*

Content Filter

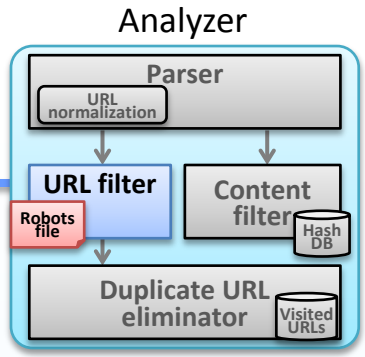


- Duplication is widespread on the Web.

- Types of duplicates
  - Intentional duplicates: mirroring of other pages
  - Unintentional duplicates: the result of the way that many web, e.g.,
    - Identifier embedded in the URLs to track user's behavior
      *http://foo.com/bar/page.php**&sessid=09A89732***
    - Internal page fragments (self-references)
      *http://foo.com/bar/page.html**#subsection***

- Done by using document fingerprints or shingles
  - A type of hashing scheme

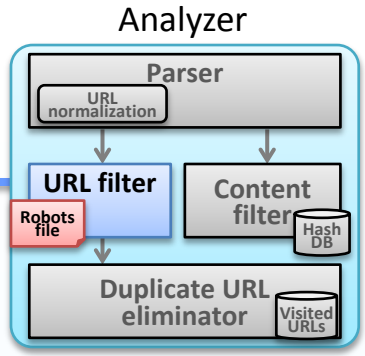
URL Filter



- Explicitly maintain politeness policies

- 2 kinds of controlling policies
  - robots.txt file: exclusion
  - sitemaps: inclusion

Robots Exclusion



- Introduced by Martijn Koster in 1994, when working for the WebCrawler
- Known as the robots.txt protocol
- Specify some restricted access to pages
- Based on regular expression matching
- The file is located in the root of host's web directory
  - e.g., *http://foo.com/robots.txt*
- More detail: *http://www.robotstxt.org*

Example of Robots Exclusion

- Exclude all robots from the entire site:

```
User-agent: *
Disallow: /
```

- Exclude specific directories:

```
User-agent: *
Disallow: /tmp/
Disallow: /cgi-bin/
Disallow: /users/paranoid/
```

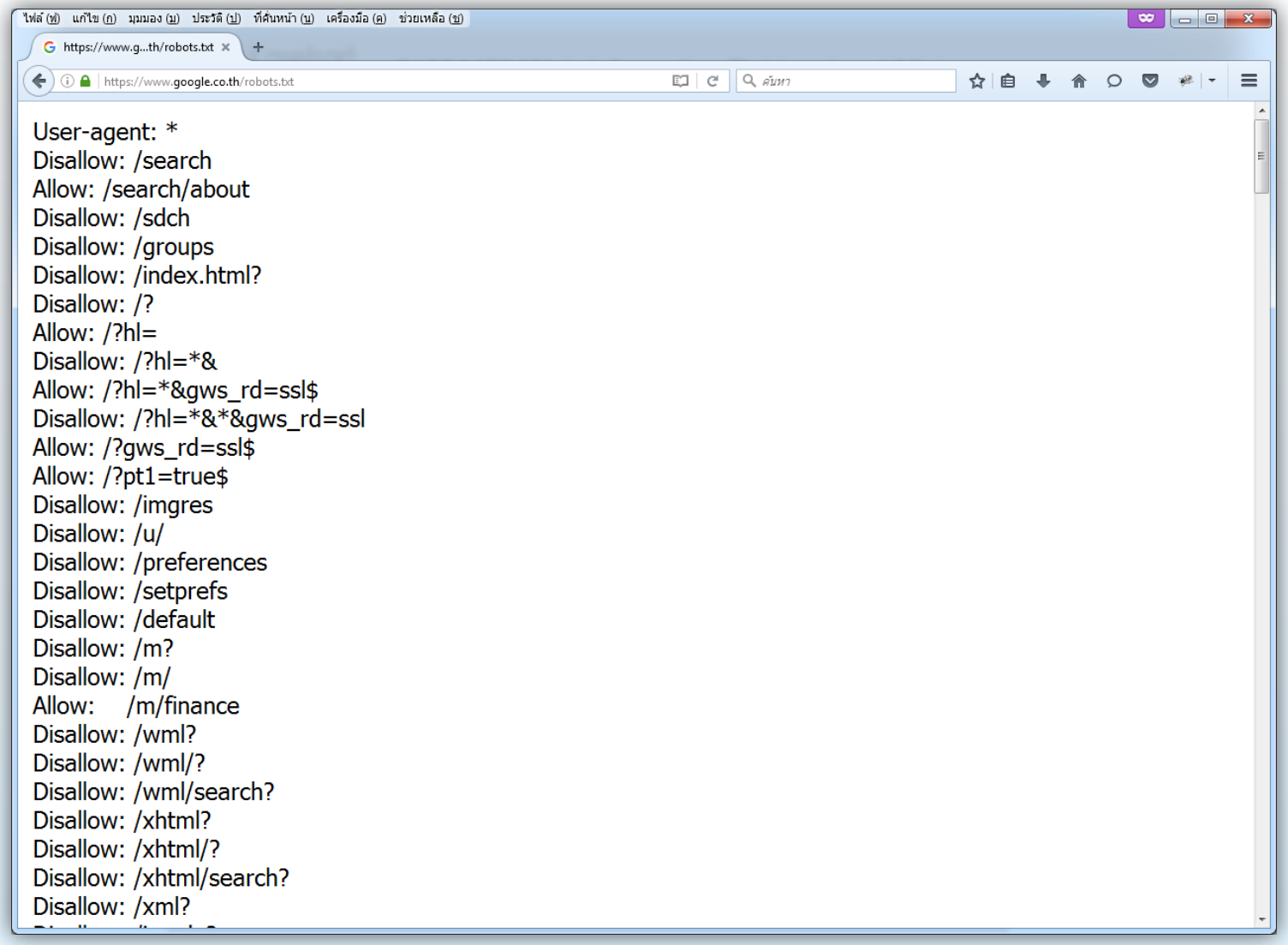
- Exclude a specific robot:

```
User-agent: GoogleBot
Disallow: /
```

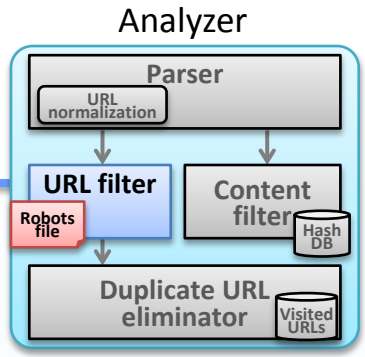
- Allow a specific robot:

```
User-agent: GoogleBot
Allow: /
User-agent: *
Disallow: /
```

Google's robots.txt



Sitemaps Inclusion



- Introduced by Google, but open standard
- XML based
- Allow webmasters to give hints to web crawlers
  - Location of pages (including URL islands)
  - Relative importance of pages
  - Update frequency of pages
- Location of sitemaps can be included in the robots.txt, by adding the following line:

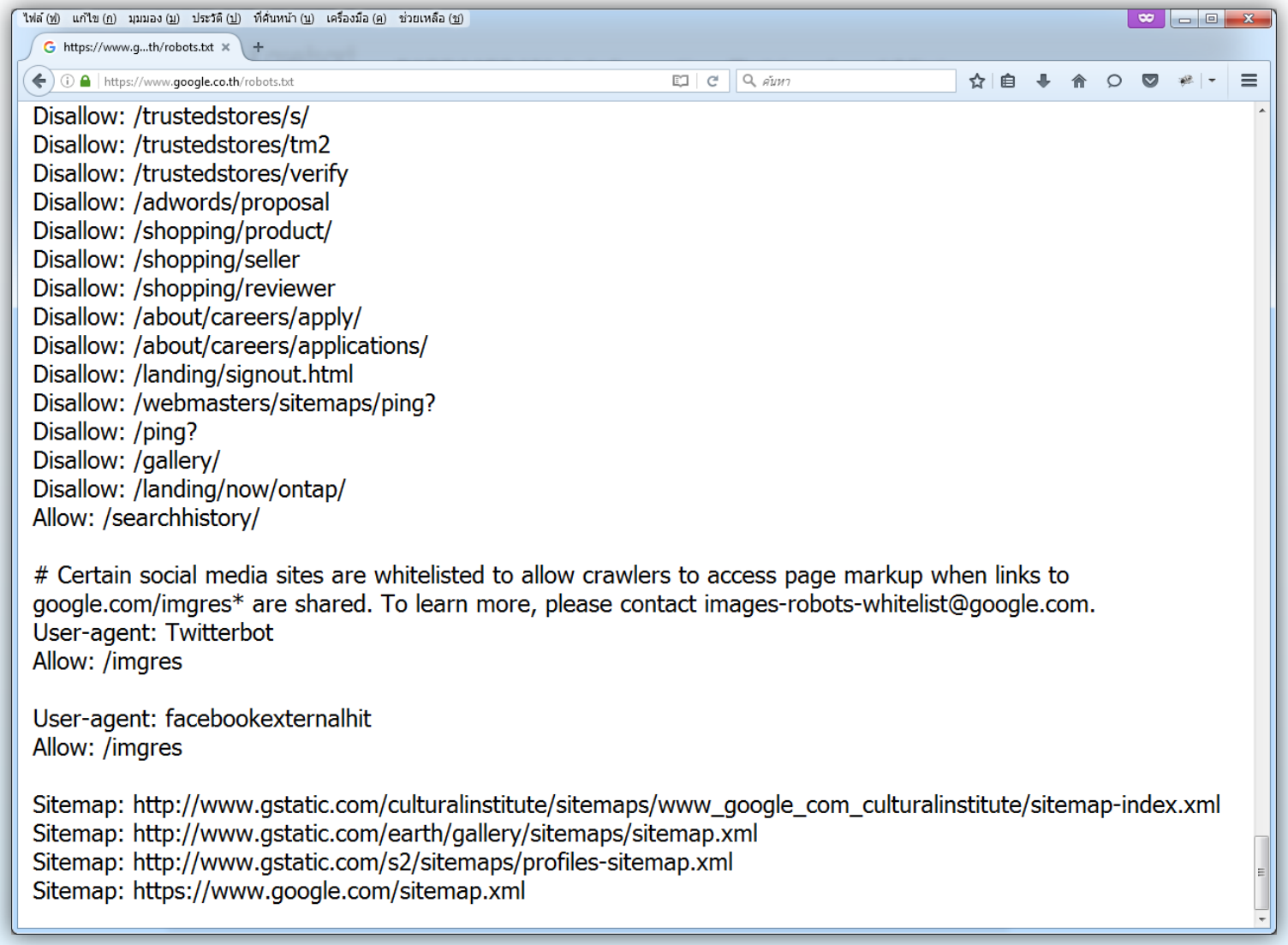
```
Sitemap: <sitemap_location>
```

- More detail: *http://www.sitemaps.org*

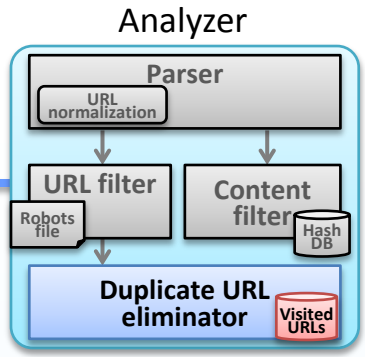
Example of Sitemaps

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.company.com/</loc>
    <lastmod>2008-01-15</lastmod>
    <changefreq>monthly</changefreq>
    <priority>0.7</priority>
  </url>
  <url>
    <loc>http://www.company.com/items?item=truck</loc>
    <changefreq>weekly</changefreq>
  </url>
  <url>
    <loc>http://www.company.com/items?item=bicycle</loc>
    <changefreq>daily</changefreq>
  </url>
</urlset>
```

Google's Sitemaps



Duplicate URL Eliminator



- For a one-time crawl
  - Ignore already visited URLs
  - Group all redundant URLs to a unique one
- For a continuous crawl
  - Eliminate stale and low quality pages

Distributed Crawling

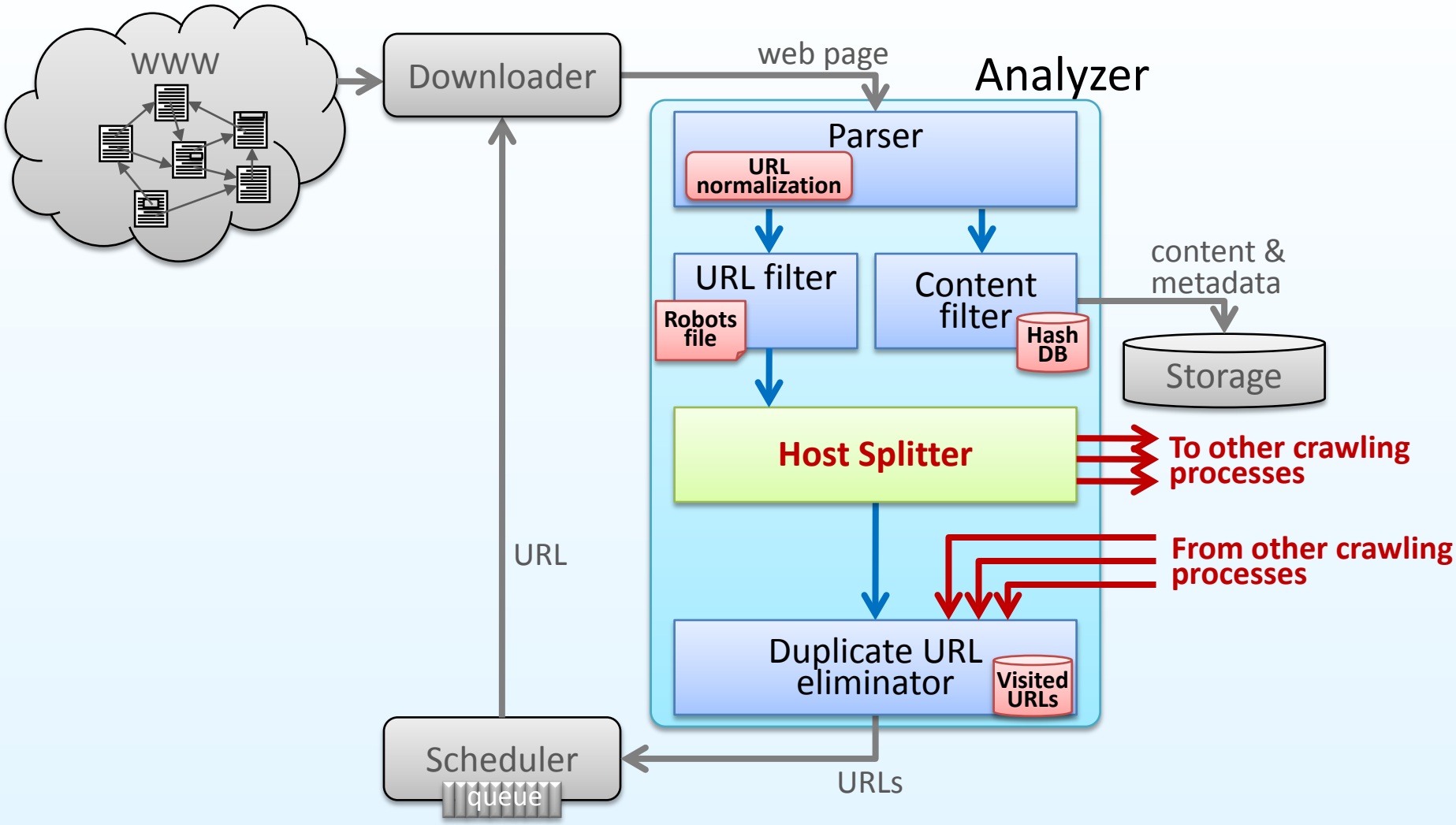
- To achieve better scalability and be more tolerant to failure
- The most important issue
  - Avoid downloading the same page more than once
  - Avoid overloading the same web servers
- The coordination among processes is done by exchanging URLs
- The goal of the crawler designer is to minimize the communication overhead

Distributed Crawling (Cont.)

- A fully distributed crawling system requires a policy for assigning the new URLs discovered.

- An effective assignment function must have
  - Balancing property: each crawling process should get approximately the same number of hosts
  - Contra-variance property: if the number of crawling processes grows, the number of hosts assigned to each process must shrink
  - Dynamic property: the number of crawling processes can be changed by adding or removing

Distributed Crawling (Cont.)



Focused Crawling

- A crawler created for some specific purpose
- Schedule or sort queue to explore more "interesting" pages first
- 2 basic styles of focused crawling
  - Topic-focused
  - Link-focused

Topic-Focused Crawling

- Assume desired topic description or sample pages of interest are given

- Sort queue of links by the similarity (e.g. cosine metric) of their source pages and/or anchor text to this topic description
  - Related to topic tracking and detection

Link-Focused Crawling

- Monitor links and keep track of in-degree and out-degree of each page encountered
  - Sort queue to prefer popular pages with many in-coming links (*authorities*)
  - Sort queue to prefer portal pages with many out-going links (*hubs*)

Up-to-date Crawling Policy

- The web is very dynamic
  - Pages and links are always created, updated, deleted
- Maintain for the fresh web collection
  - Periodically check crawled pages for updates and deletions
  - Simple method is to check pages' header information
    - e.g., the last-modified META tag
  - Only reload entire pages if needed.
- Track how often each page is updated
  - Preferentially update pages that are often changed to optimize freshness

Any Question?