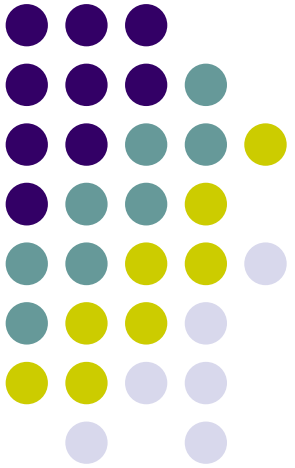
Web Search (1)



References:

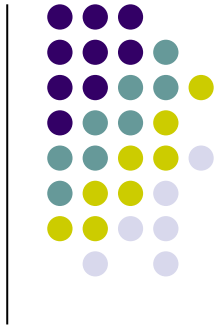
Gerald Benoit, Simmons College, Web Search

Dustin Boswell, Distributed Web Crawling

Pandu Nayak and Prabhakar Raghavan，Stanford University

Edt. By: Qingcai Chen, HITSZ

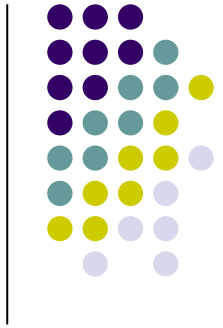
Content



- Overview of web search
- Web crawler
  - ***Politeness***
  - ***Performance***
- *Indexing/ranking*
- *Scalability*

} (later lessons)

Web Search



Goal

Provide information discovery for large amounts of open access material on the web

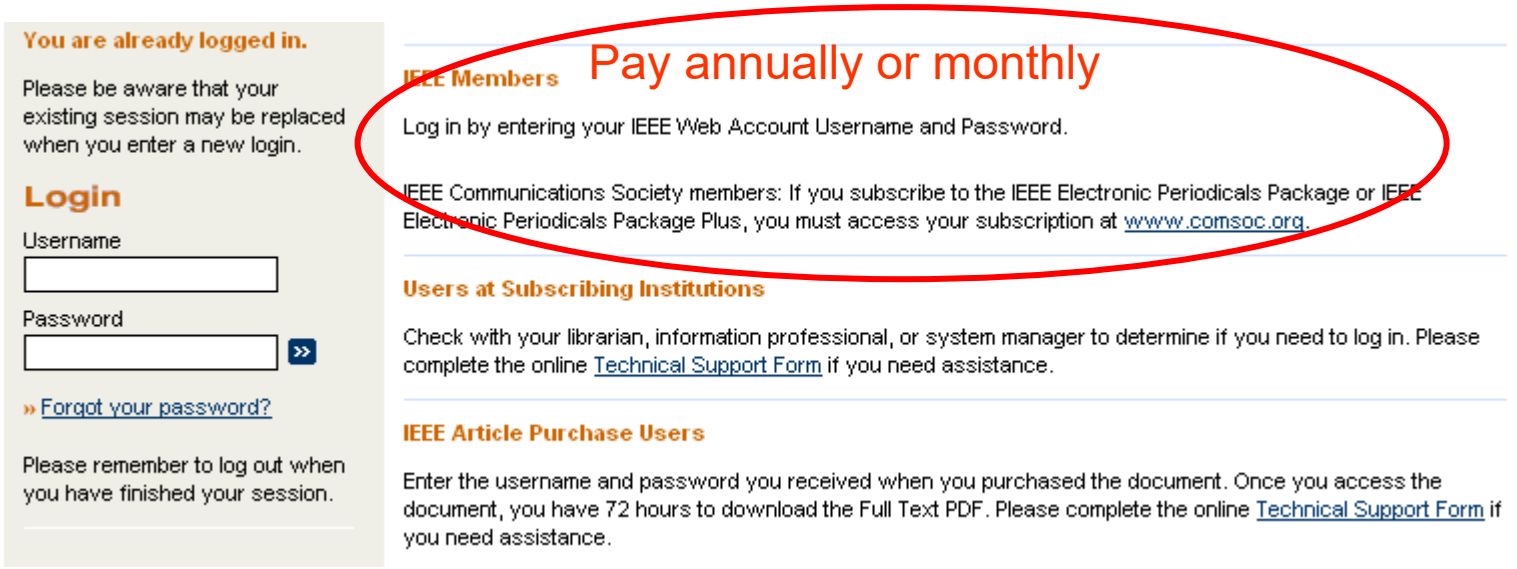
Challenges

- Volume of material -- several billion items, growing steadily
- Items created dynamically or in databases (deep web, about 150 times of web pages of surface web)
- Great variety -- length, formats, quality control, purpose, etc.
- Inexperience of users -- range of needs
- Economic models to pay for the service

Economic Models

Subscription

Monthly fee with logon provides unlimited access (introduced by InfoSeek)



Economic Models

Advertising

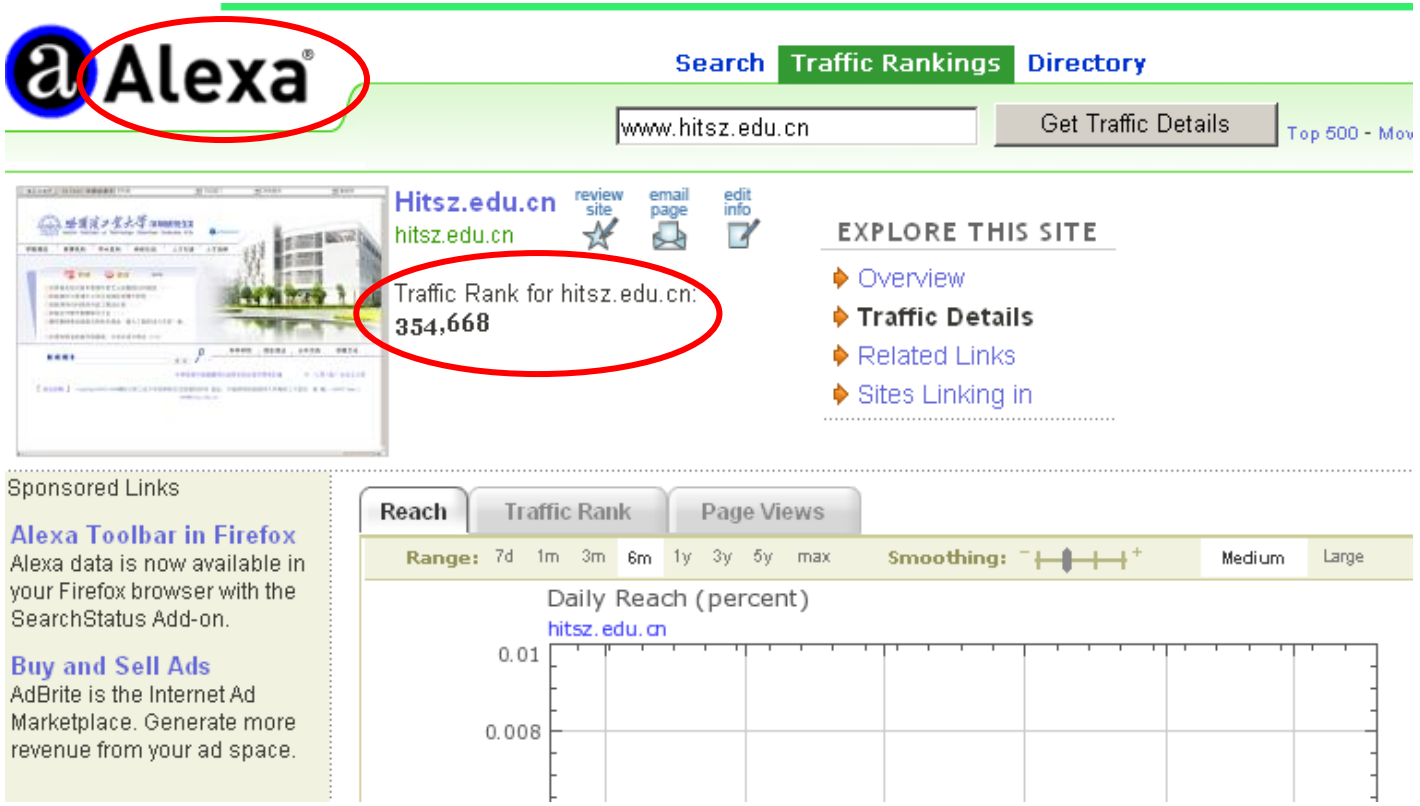
Access is free, with display advertisements (introduced by Lycos, but boosted by Google®)

Can lead to distortion of results to suit advertisers (Don't be evil?)

Economic Models

Advertising

Traffic Ranking, the “life” of a commercial search engine!



Economic Models

Licensing

Cost of company are covered by fees, licensing of software and specialized services

Strategies

Subject hierarchies — 1st Generation

- Yahoo! -- use of human indexing

Web crawling + automatic indexing – 2nd Generation

- General -- Google, Baidu, Infoseek, Lycos, AltaVista,...

Mixed models

- Human directed web crawling and automatic indexing

Strategies - Examples

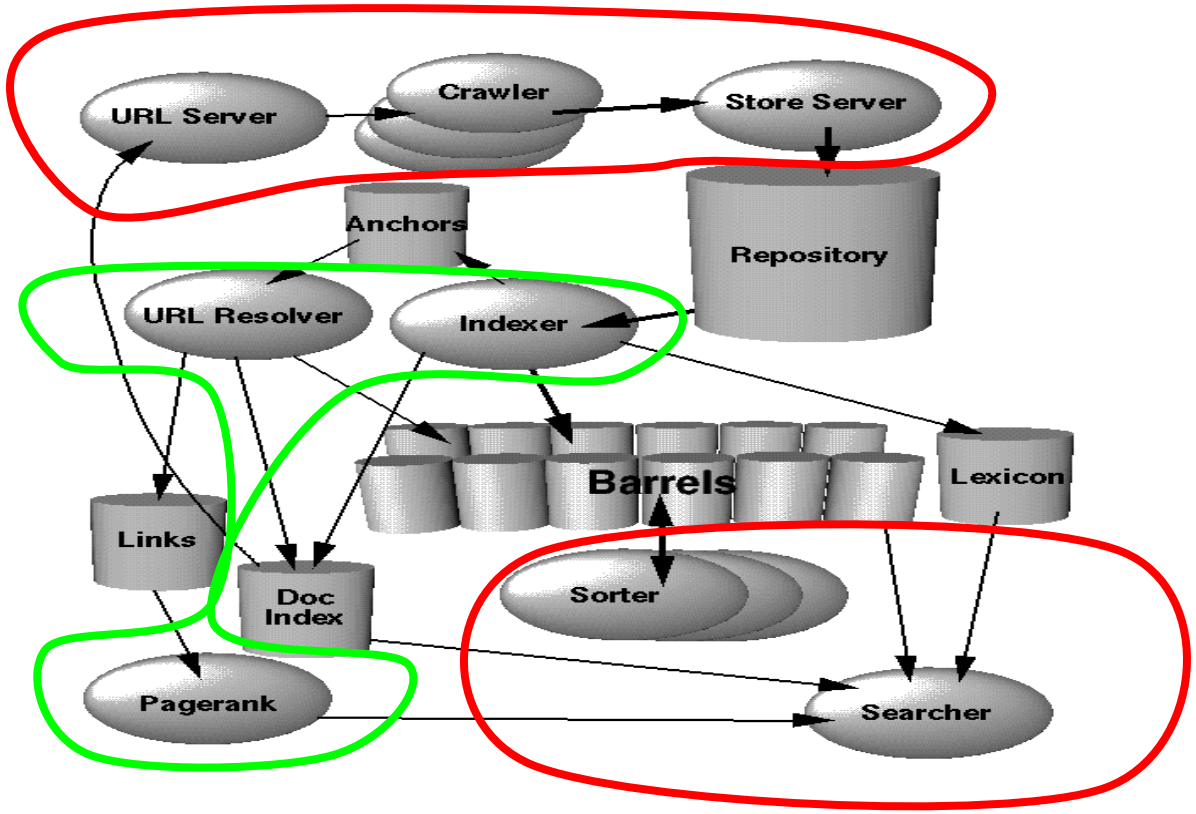
Components of Web Search Service

Components

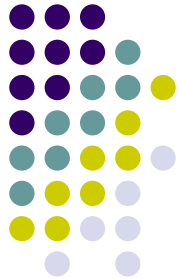
- Web crawler
- Indexing system
- Search system

Considerations

- Economics
- Scalability
- Legal issues



Content



- *Overview of web search*
- Web crawler
  - ***Politeness***
  - ***Performance***

What is a Web Crawler?

Web Crawler? What? How?

- A program for downloading web pages.
- Given an initial set of seed URLs, it recursively downloads every page that is linked from pages in the set.
- Variations:

  A focused web (专业爬虫) crawler downloads only those pages whose content satisfies some criterion. (Needs the support of categorization approaches)

  A deep web crawler downloads dynamic web content (trends to be hot!)

Also known as a web spider

Simple Web Crawler Algorithm

Basic Algorithm

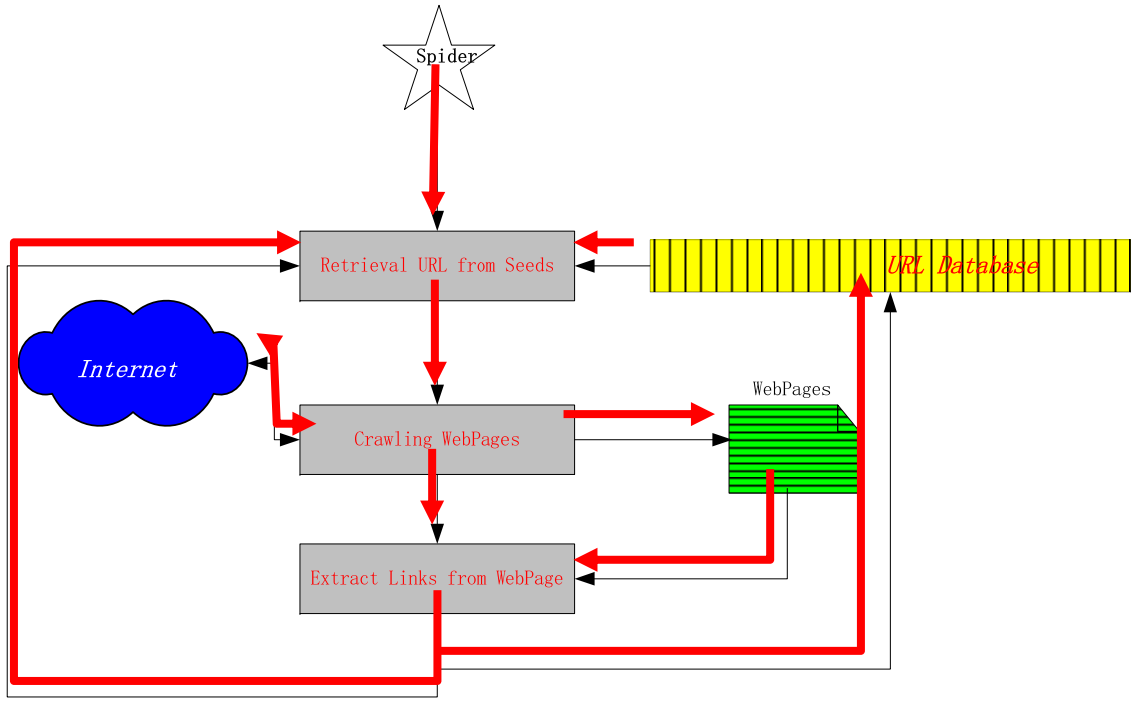
Let *S* be set of URLs to pages waiting to be indexed. Initially *S* is the singleton, *s*, known as the seed.

Take an element *u* of *S* and **retrieve** the page, *p*, that it references.

Parse the page *p* and **extract** the set of URLs *L* it has links to.

Update $S = S + L - u$

Repeat as many times as necessary.



Simple Web Crawler Algorithm (cont'd)

UrlsTodo = { ‘ ‘yahoo.com/index.html’ ’ }

Repeat:

url = UrlsTodo.getNext()

html = Download(url)

UrlsDone.insert(url)

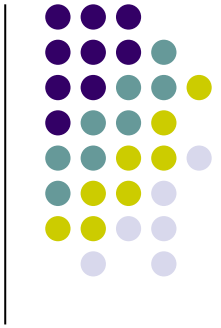
newUrls = parseForLinks(html)

For each newUrl not in UrlsDone:

UrlsTodo.insert(newUrl)

http://sports.yahoo.com/mlb/news?slug=ti-mlb_07_dimaggio031907&prov=yhoo&type=lgns

Not so simple…



Performance -- How do you crawl 1,000,000,000 pages?

Politeness -- How do you avoid overloading servers?

Failures -- Broken links, time outs, spider traps.

Strategies -- How deep do we go? *Depth first* or *breadth first*?

Implementations -- How do we store and update *S* and the other data structures needed? (Parallel file system?)

What to retrieve

No web crawler retrieves everything

- Most crawlers retrieve only
  - HTML (leaves and nodes in the tree)
  - ASCII clear text (only as leaves in the tree)
- Some retrieve
  - PDF
  - PostScript,…
- Indexing after crawl
  - Some index only the first part of long files
  - Do you keep the files (e.g., Google cache)?

I 百度诉360违反Robots协议案开庭 百度索赔1亿元 crawler”

腾讯科技 [微博] 2013年10月16日18:34

[导读]百度认为，奇虎360违背了
容。

腾讯科技 启言 10月16日报道

百度诉奇虎360违反"Robots协议
京市第一中级人民法院开庭审理。

百度方面认为，360搜索在未获得百度公司允许的情况下，违反业内公认的Robots协议，抓取百度旗下百度知道、百度百科、百度贴吧等网站的内容，已经构成了不正当竞争，并向奇虎索赔1亿元。

2012年8月，360搜索悄然上线后不久即违反Robots协议，强行抓取百度旗下网站百度知道、百度百科、百度贴吧、百度旅游等内容。360搜索在百度Robots文本中还未将360爬虫写入的情况下，违反Robots协议内容，强制对"百度知道"、"百度百科"等百度网站内容进行了抓取。

百度公司认为，奇虎360的行为违背了国际通行的行业规则、不顾百度的权利声明和技术措施，非法抓取、复制百度网站内容，直接以快照形式向网民提供，严重侵害了百度的合法权益，构成了不正当竞争。随后，百度公司将奇虎360诉至北京市第一中级人民法院，该案于今年2月23日正式立案。

百度公关部郭彪向媒体表示，Robots协议是网站信息和网民隐私保护的国际通行规范之一，理应得到全球互联网公司的共同遵守。而360公司回应称，Robots协议的本质是网站和搜索引擎爬虫的沟通方式，用来指导搜索引擎更好地抓取网站内容，robots协议的创始人Martijn Koster从一开始即预测到了，有的商家可能用其作为不正当的市场竞争工具。因此，他在1994年创制伊始便告诫人们，"如果该协议被当成市场竞争工具，爬虫不需要采纳"。

360公司认为，谷歌(微博)、雅虎、微软等的robots协议都是旨在防止搜索爬虫抓取到涉及用户登录信息，同时对所有爬虫一视同仁。百度Robots协议允许谷歌、必应、搜狗、搜搜、即刻、盘古等其他搜索引擎抓取百度知道、贴吧等内容，唯独禁止搜索市场份额排名第二的360搜索抓取，这是滥用Robots协议维持其搜索市场垄断地位的行为。

Robots Exclusion

Example file: /robots.txt

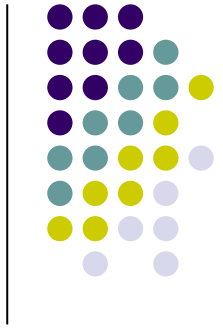
```
# Disallow allow all robots
User-agent: *
Disallow: /cyberworld/map/
Disallow: /tmp/          # these will soon disappear
Disallow: /foo.html

# To allow Cybermapper
User-agent: cybermapper
Disallow:
```

Extracts from: http://www.google.com/robots.txt

User-agent: *
Allow: /searchhistory/
Disallow: /search
Disallow: /groups
Disallow: /images
Disallow: /catalogs
Disallow: /catalogues
Disallow: /news
Disallow: /nwshp
Disallow: /?
Disallow: /addurl/image?
Disallow: /pagead/
Disallow: /relpage/
Disallow: /relcontent
……

The Robots META tag



The **Robots META tag** allows HTML authors to indicate to visiting robots if a document may be indexed, or used to harvest more links. No server administrator action is required.

Note that currently only a few robots implement this.

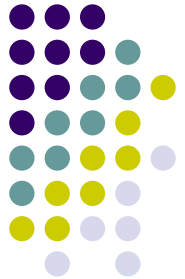
In this simple example:

<meta name="robots" content="noindex, nofollow">

a robot should neither index this document, nor analyze it for links.

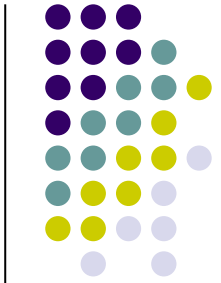
http://www.robotstxt.org/wc/exclusion.html#meta

Content



- *Overview of web search*
- Web crawler
  - ***Politeness***
  - ***Performance***

Statistics to Keep in Mind

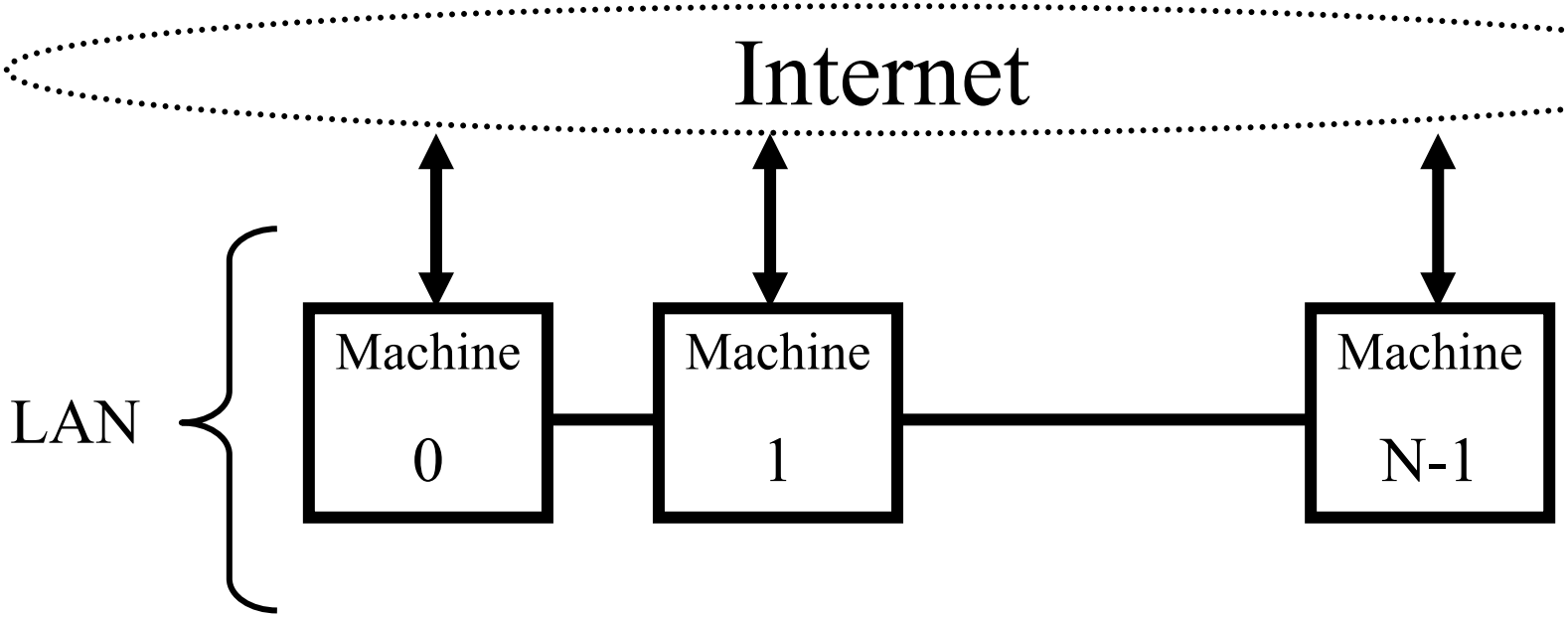


Documents on the web:	8 Billion + (by Google's count, early than 2005, >10 Billions now)
Avg. HTML size:	15KB
Avg. URL length:	50+ characters
Links per page:	10
External Links per page:	2

Download the entire web in a year: **2026 urls / second !**

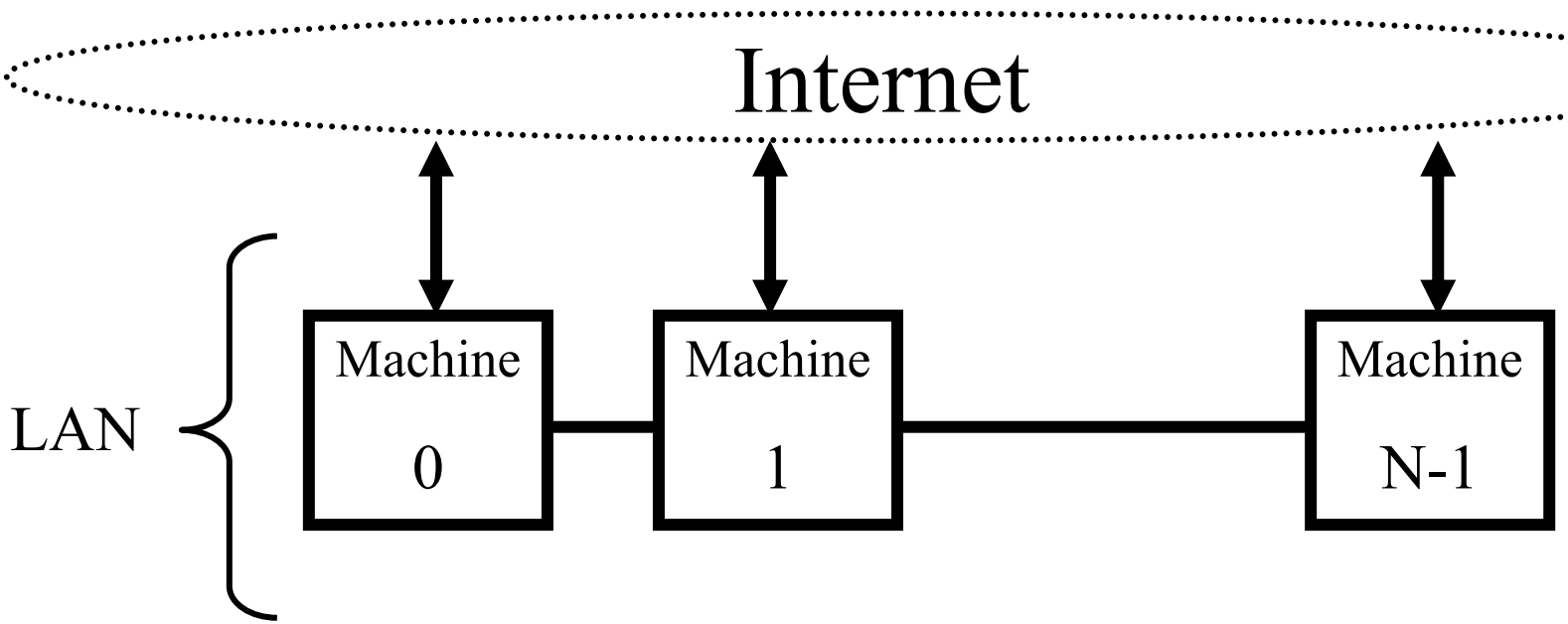
8 Billion * 15KB	=	120	TeraBytes of HTML
8 Billion * 50 chars	=	400	GigaBytes of URL's !!
		→	multiple machines required

Distributing the Workload



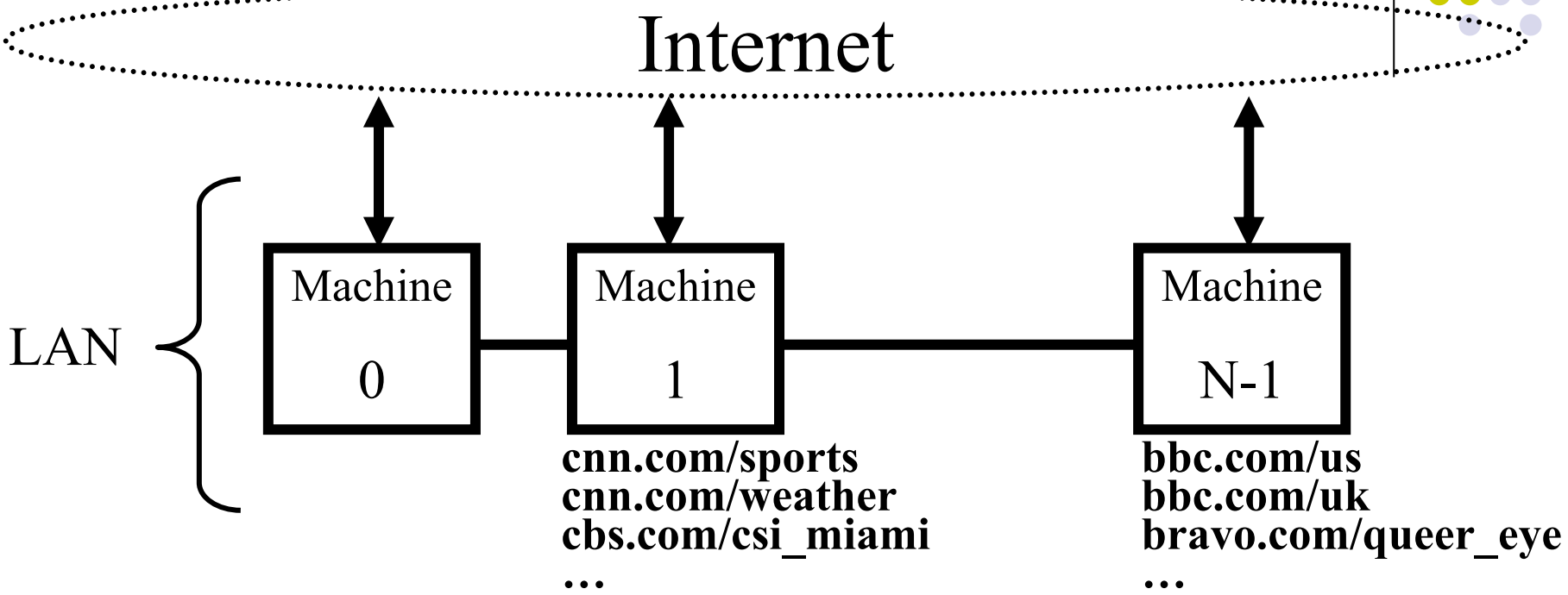
- Each machine is assigned a fixed subset of the url-space
- Then how to assign the tasks?

Distributing the Workload



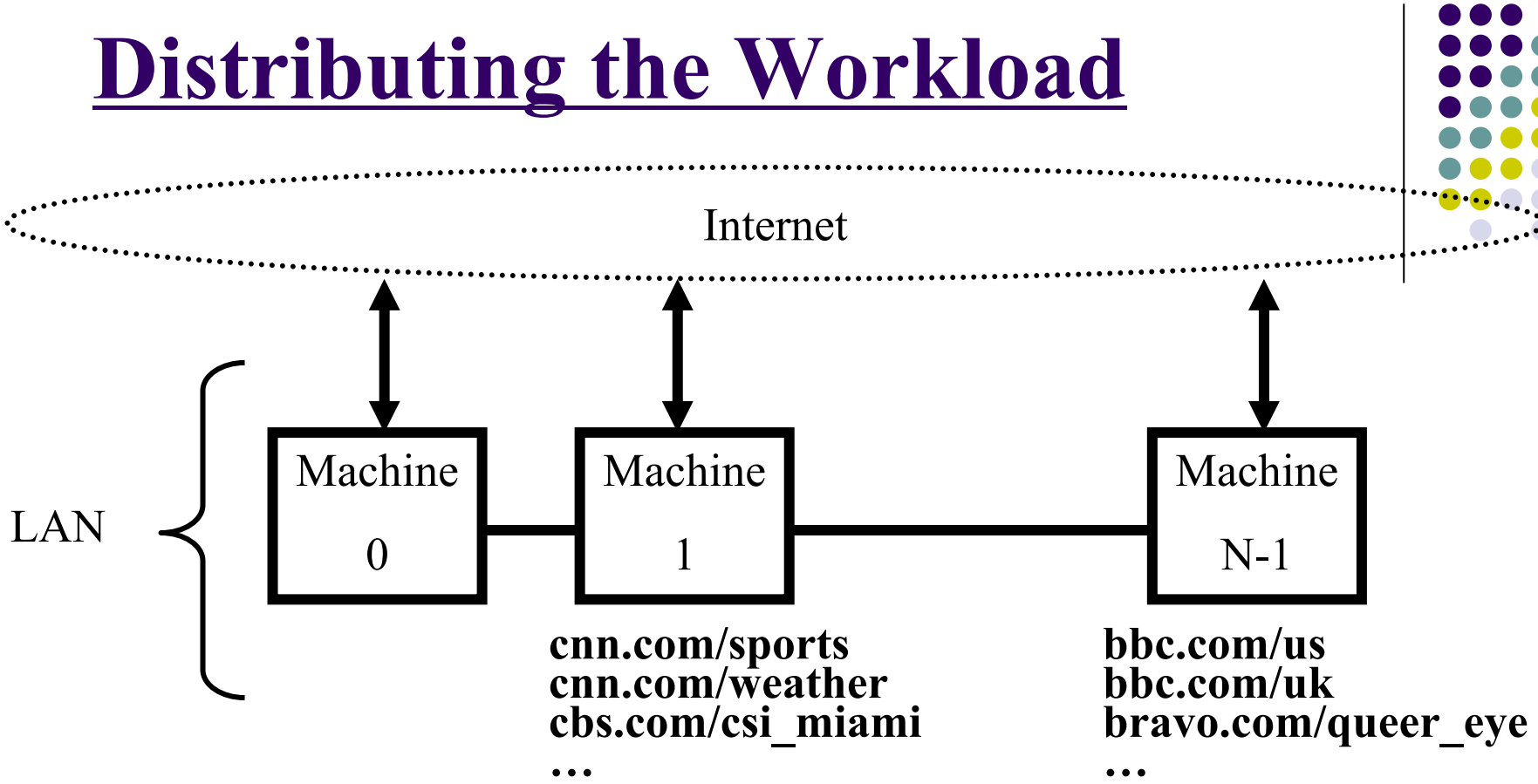
- Each machine is assigned a fixed subset of the url-space
- **`machine = hash( url's domain name )% N`**

Distributing the Workload



- Each machine is assigned a fixed subset of the url-space
- **`machine = hash( url's domain name )% N`**

Distributing the Workload



- Each machine is assigned a fixed subset of the url-space
- **machine = hash(url's domain name)% N**
  - Communication: a couple urls per page (very small)
  - DNS cache per machine
  - Maintain politeness : don't want to DOS (Denial of Service) attack someone!

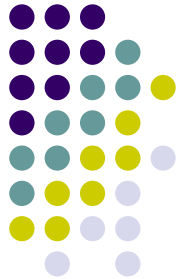
Software Hazards

- Slow/Unresponsive DNS Servers
- Slow/Unresponsive HTTP Servers

} parallel / async interface desired

- Large or Infinite-sized pages
- Infinite Links (“domain.com/time=100”, “…101”, “…102”, …)
- Broken HTML

Building a Web Crawler: Links are not Easy to Extract



Relative/Absolute

CGI

- Parameters
- Dynamic generation of pages

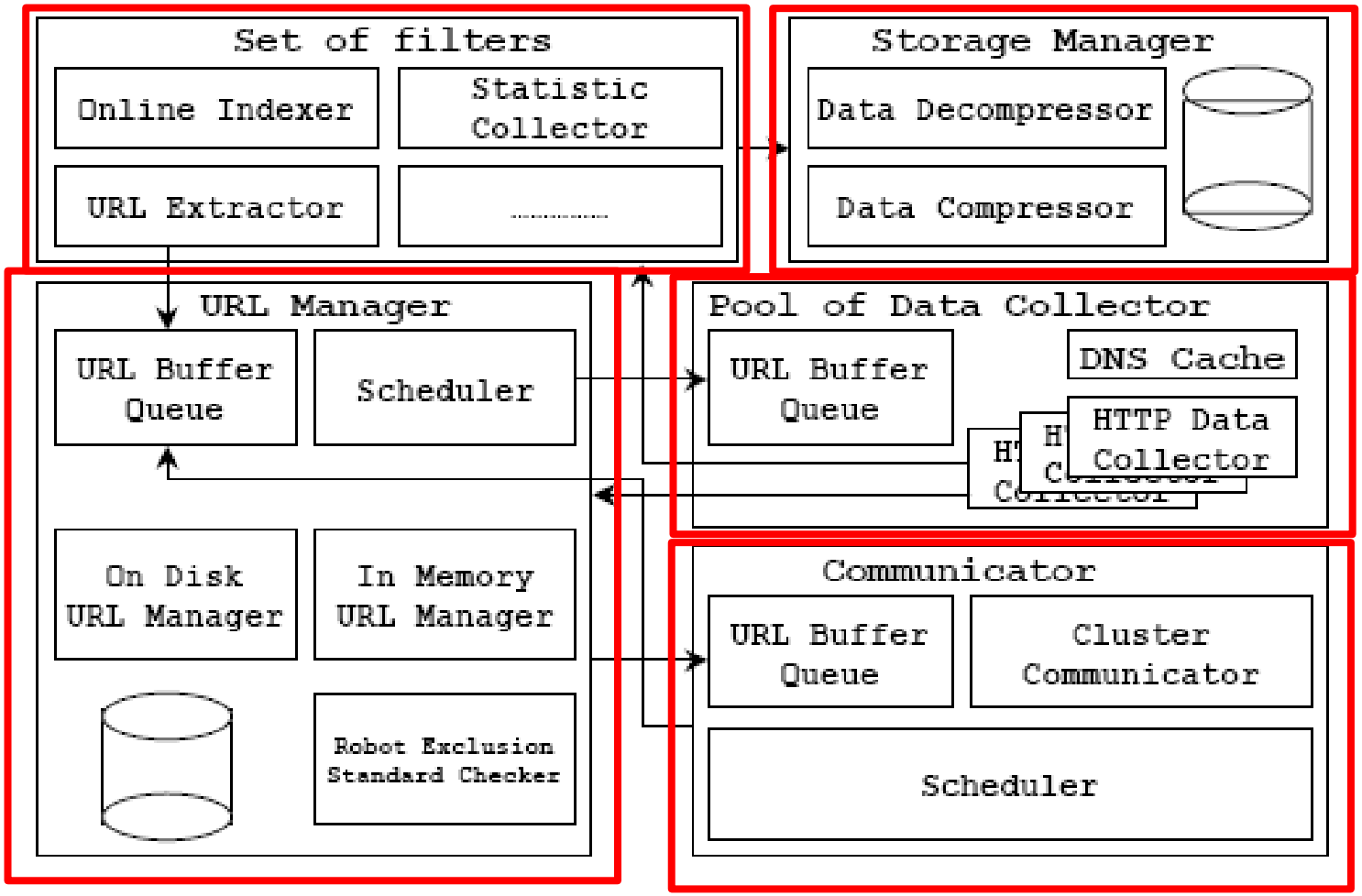
Server-side scripting

Server-side image maps

Links buried in scripting code (e.g. in Java Script or JS)

An example crawler - architecture

Koht-arsa, Sanguanpong, High performance large scale web spider architecture. 2002

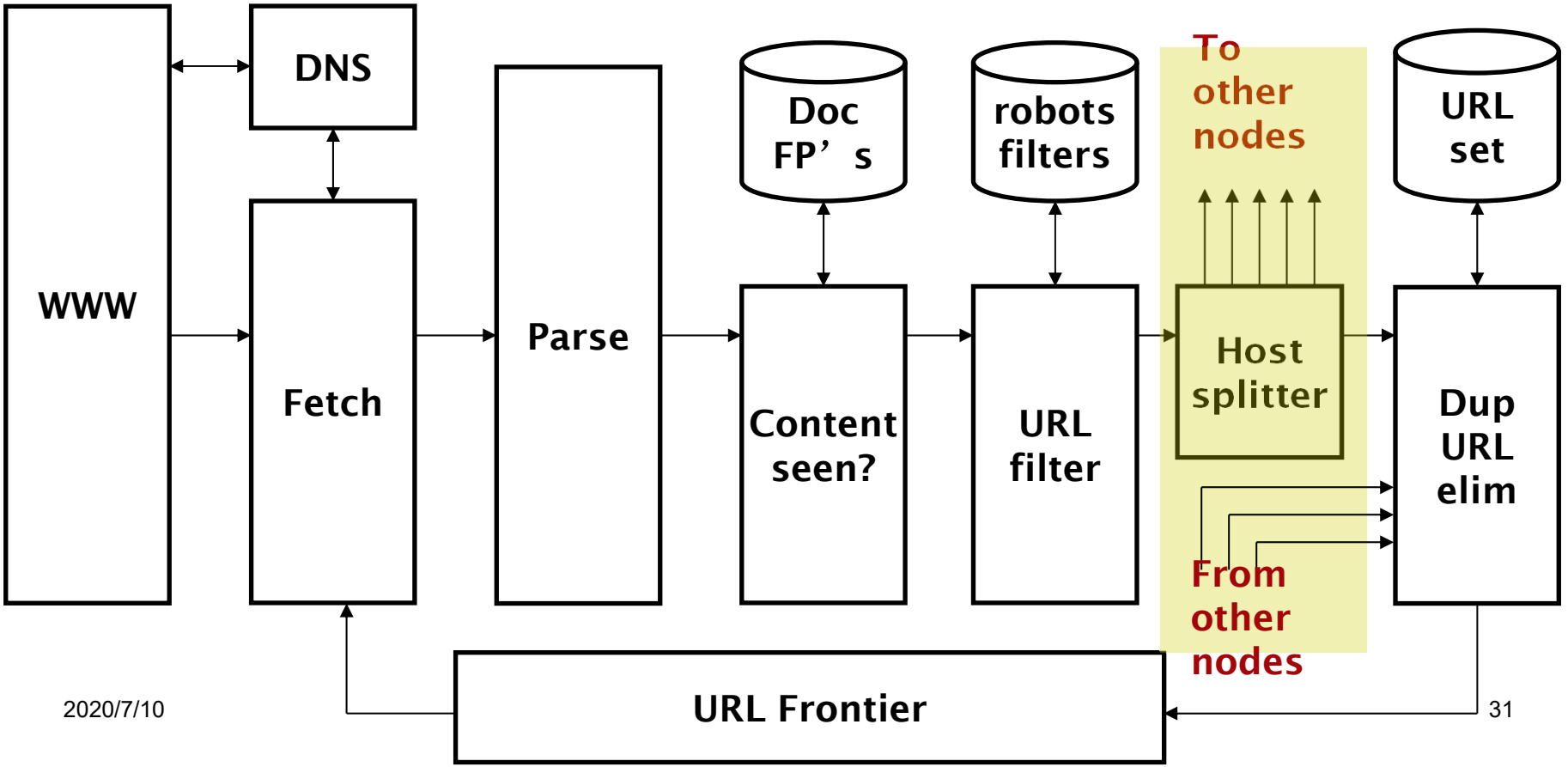


Example crawler architecture (Cont’d)

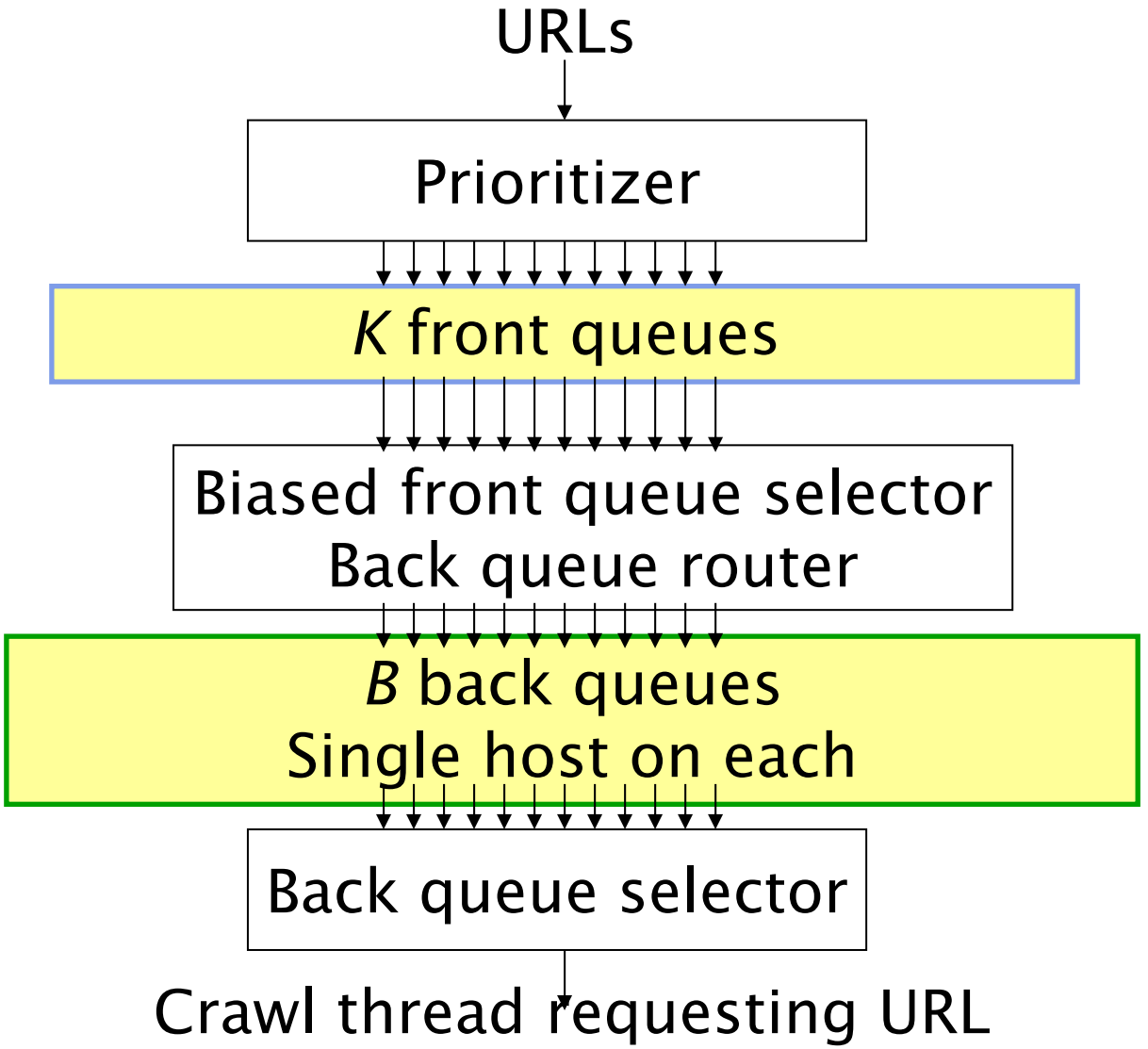
- **URL Manager**
  - The URL Manager keeps track of all the URL set of that node.
- **Pool of data collector**
  - The pool of data collector queue the URL list sent from the URL Manager. It has many collector threads to fetch the data from the web servers.
- **Set of filters**
  - link extract
  - collection statistic
  - online indexer
- **Storage manager**
  - The storage manager’s role is compression and decompressions, storing and retrieving the data.
  - Data compressing: zlib, lzo, etc.
- **Communicator**
  - The communicator send/receive new URLs found from the node that found it to the node that responsible to manage it.

Communication between nodes

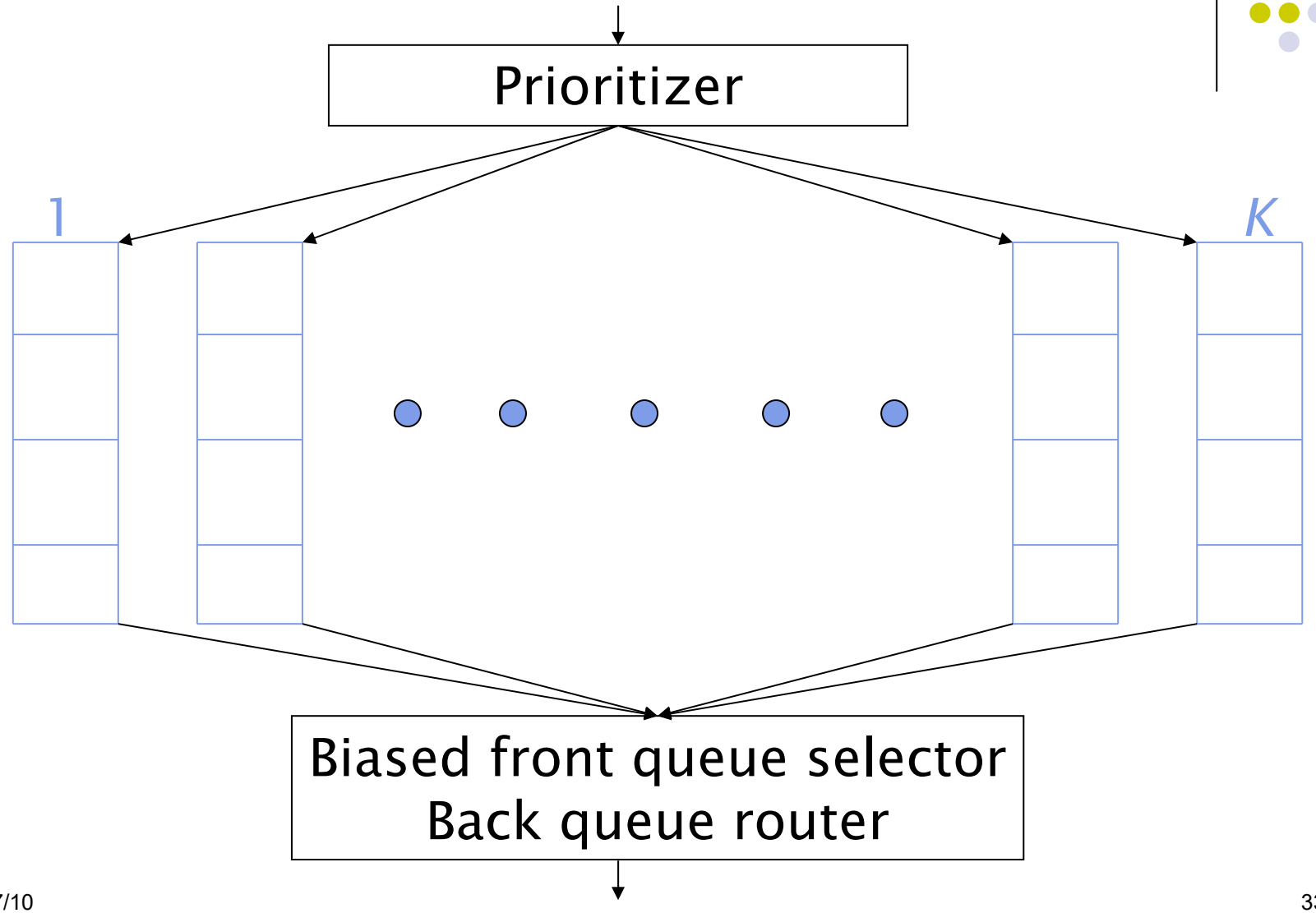
- Output of the URL filter at each node is sent to the Dup URL Eliminator of the appropriate node



URL frontier: Mercator scheme



Front queues



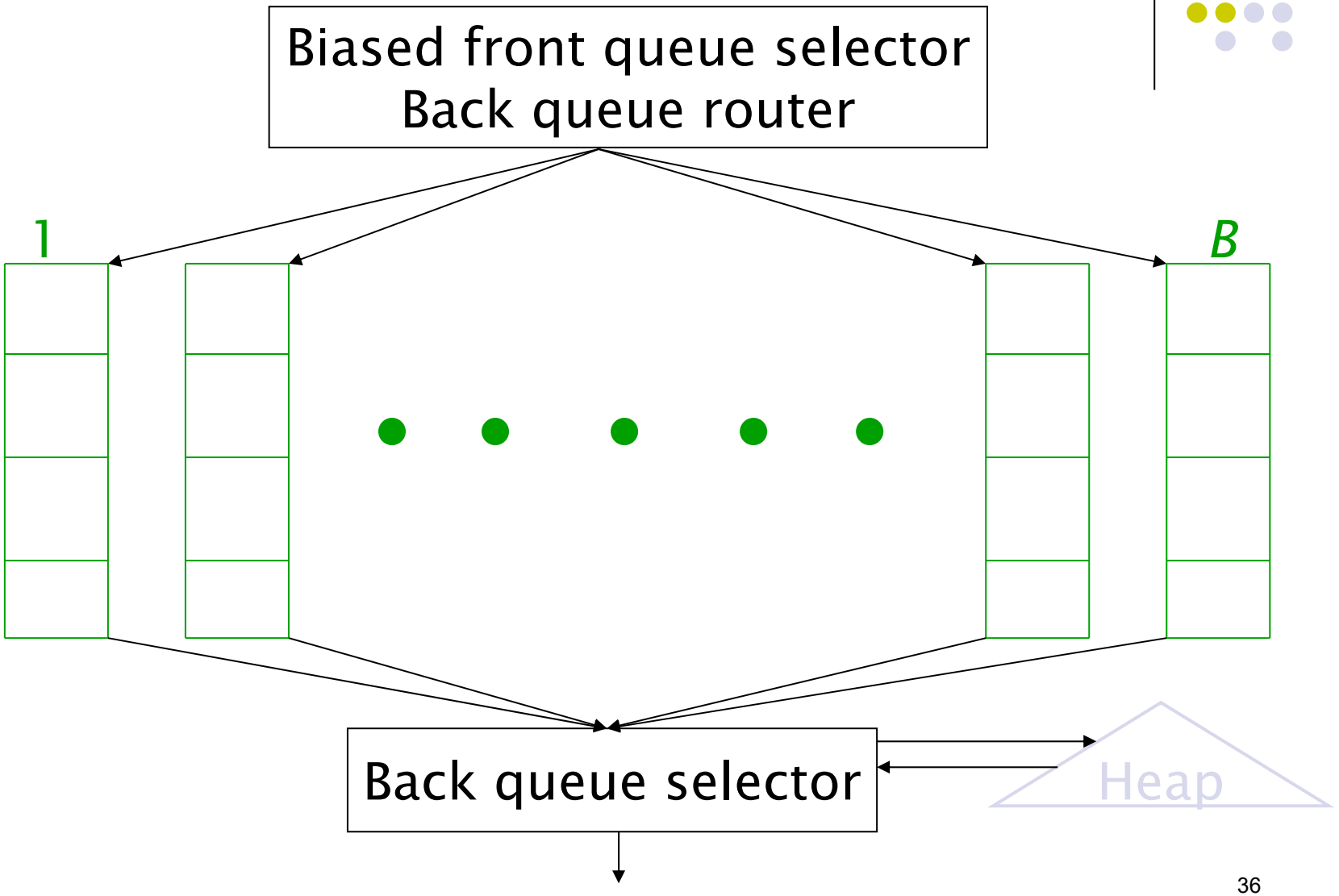
Front queues

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

Biased front queue selector

- 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 can be round robin biased to queues of higher priority, or some more sophisticated variant
  - Can be randomized

Back queues

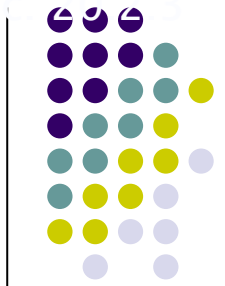

Biased front queue selector
Back queue router
1
B
Back queue selector
Heap

Back queue invariants

- 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

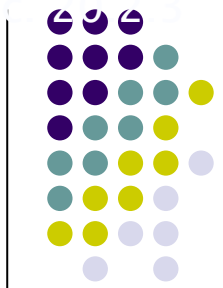
Host name	Back queue
…	3
	1
	B

Back queue heap



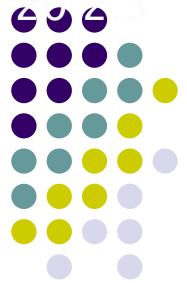
- One entry for each back queue
- The entry is the earliest time t_e at which the host corresponding to the back queue can be hit again
- This earliest time is determined from
  - Last access to that host
  - Any time buffer heuristic we choose

Back queue processing



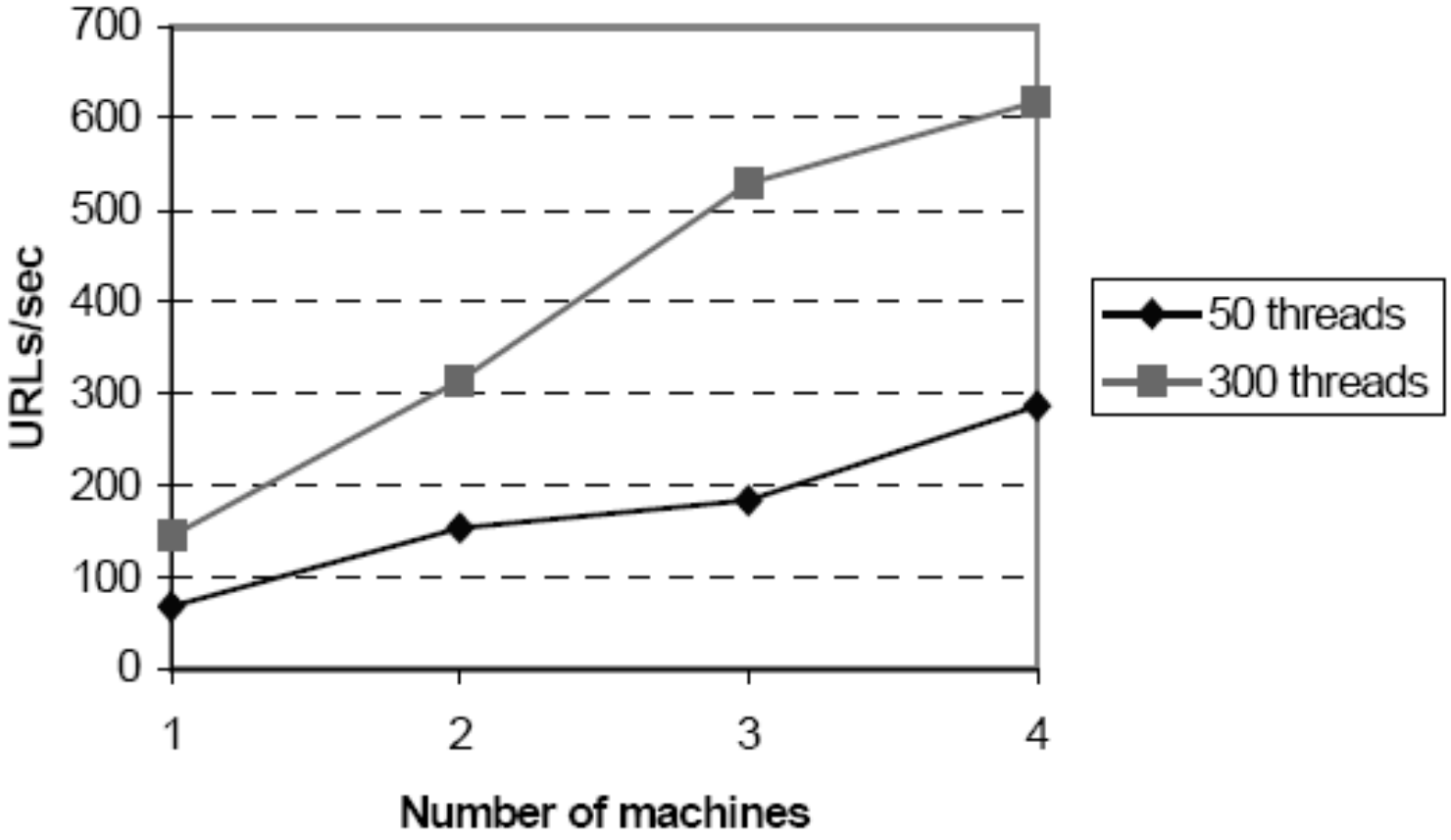
- A crawler thread seeking a URL to crawl:
- Extracts the root of the heap
- Fetches URL at head of corresponding back queue *q* (look up from table)
- Checks if queue *q* is now empty – if so, pulls a URL *v* from front queues
  - If there’s already a back queue for *v*’s host, append *v* to that queue 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 queues *B*



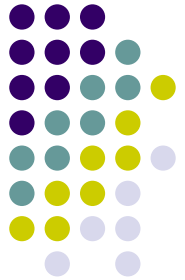
- Keep all threads busy while respecting politeness
- Mercator recommendation: three times as many back queues as crawler threads

Example crawler – gathering speed



Discussion

- How frequently to crawl and what strategies to use?
- Duplicate (重复) detection?
- How to avoid spam?
- Strategies for crawling based on the content of web pages (focused and selective crawling).

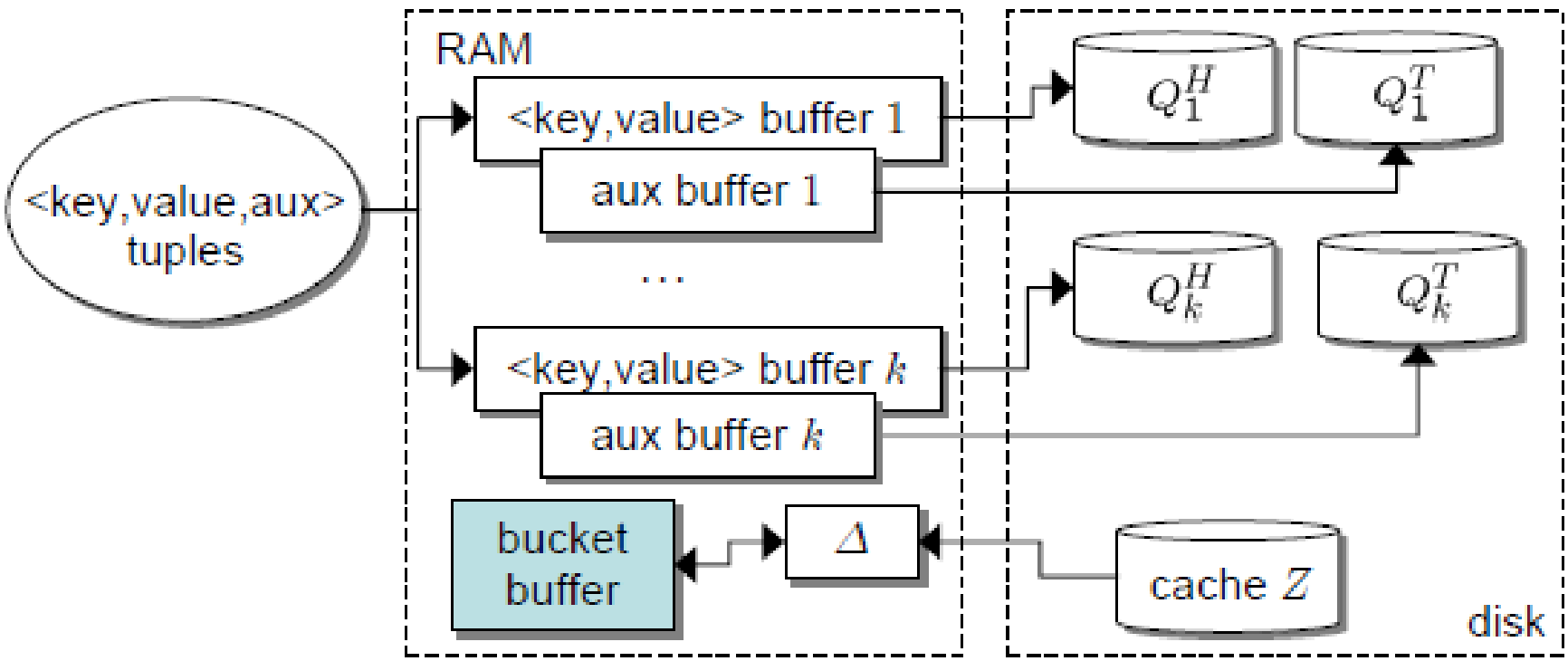


Question?

Reading Report:
IRLbot: Collecting 6 Billion Pages within 41 Days?

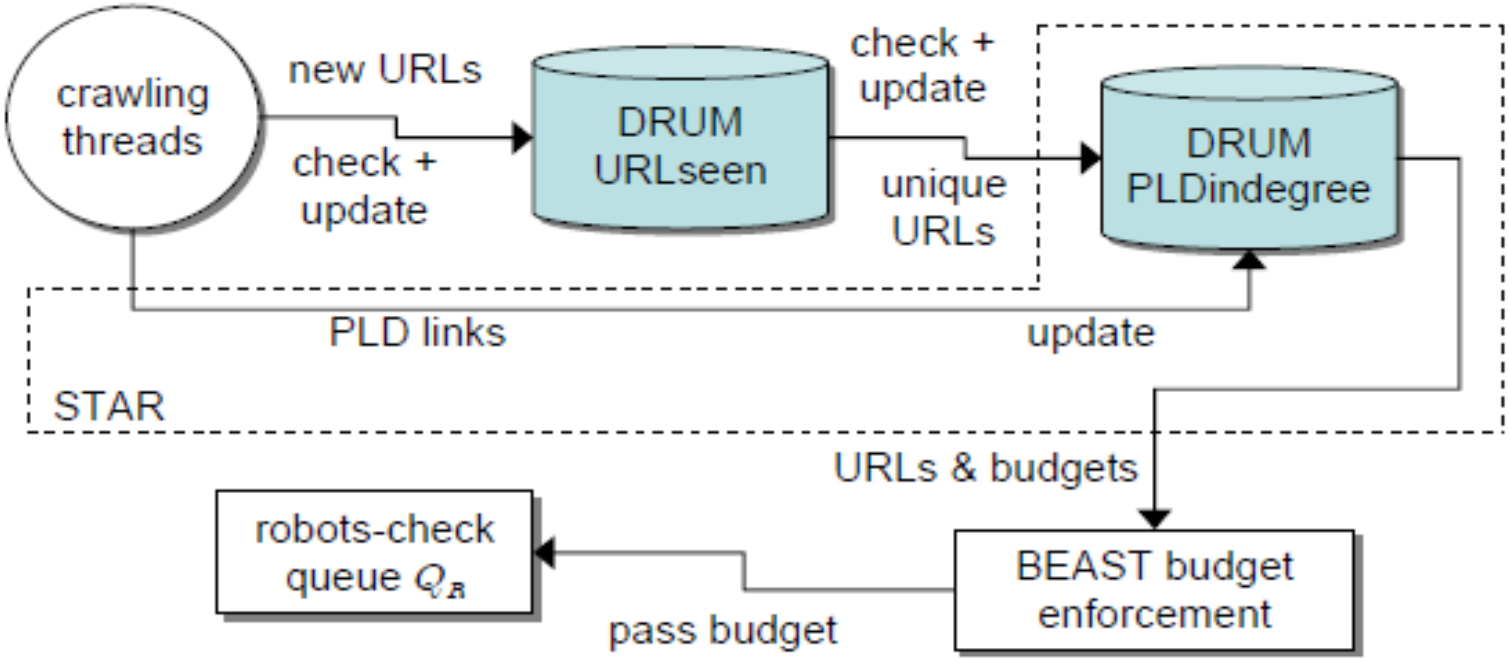
- WWW’08 best paper
  - Hsin-Tsang Lee, Derek Leonard, Xiaoming Wang, and Dmitri Loguinov, **IRLbot: Scaling to 6 Billion Pages and Beyond. WWW’08, Beijing, China**
- They are trying to address the key problems of a crawler, i.e.
  - Scalability: DRUM (Disk Repository with Update Management)
  - Spam avoidance: STAR
  - Politeness: BEAST
- Details……

DRUM (Disk Repository with Update Management)



STAR (Spam Tracking and Avoidance through Reputation)

Set URL budget of for each pay-level domain (PLD) according to its PLDIndegree (PLD入度)

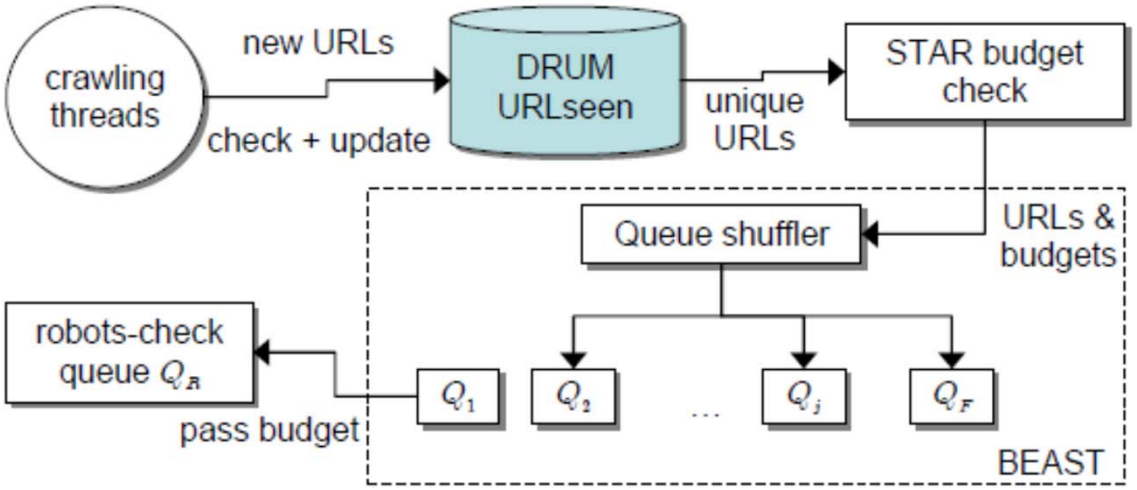


BEAST (Budget Enforcement with Anti-Spam Tactics)

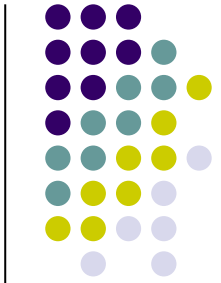
- keep t_0 = 40 seconds for accessing all low-ranked PLDs
- for high-ranked PLDs scale it down proportional to their budget B_x

$$t_1 \propto (1/B_x)^* t_0$$

-



Discussion



- Though high performance they reached, some issues are still left:
  - Downloading of large files
  - Webpage refreshing
  - Intellectual property protection
  - Web servers’ bandwidth wasting
  - etc.

Further Reading

“IRLbot: Collecting 6 Billion Pages within 41 Days?”, WWW’08

Heritrix
http://crawler.archive.org/

Larbin
http://larbin.sourceforge.net/index-eng.html

Allan Heydon and Marc Najork, *Mercator: A Scalable, Extensible Web Crawler*. Compaq Systems Research Center, June 26, 1999.
http://www.research.compaq.com/SRC/mercator/papers/www/paper.html

Koht-arsa, Sanguanpong, High performance large scale web spider architecture. The 2002 Internataional Symposium on Communications and Information Technology, Thailand, October 2002

Dustin Boswell dboswell et al, Distributed High-performance Web Crawlers: A Survey of the State of the Art .

Backup

- More examples of web crawler
  - Google
  - Mercator (for AltaVista)
  - Heritrix
  - Labin

Previous Web Crawlers

	Google Prototype – 1998	Mercator – 2001 (used at AltaVista)
Downloading (per machine):	300 asynch connections	100’s of synchronous threads
Crawling Results:	4 machines 24 million pages 48 pages/ second	4 machines 891 million 600 pages/second

Example: Heritrix Crawler

A high-performance, open source crawler for production and research

Developed by the Internet Archive and others

Before Heritrix, Cornell computer science used the Mercator web crawler for experiments in selective web crawling (automated collection development). Mercator was developed by Allan Heydon, Marc Njork and colleagues at Compaq Systems Research Center. This was continuation of work of Digital's AltaVista group.

Heritrix: Design Goals

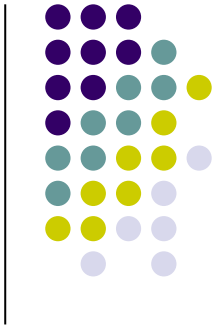
Broad crawling: Large, high-bandwidth crawls to sample as much of the web as possible given the time, bandwidth, and storage resources available.

Focused crawling: Small- to medium-sized crawls (usually less than 10 million unique documents) in which the quality criterion is complete coverage of selected sites or topics.

Continuous crawling: Crawls that revisit previously fetched pages, looking for changes and new pages, even adapting its crawl rate based on parameters and estimated change frequencies.

Experimental crawling: Experiment with crawling techniques, such as choice of what to crawl, order of crawled, crawling using diverse protocols, and analysis and archiving of crawl results.

Heritrix



Design parameters

- Extensible. Many components are plugins that can be rewritten for different tasks.
- Distributed. A crawl can be distributed in a symmetric fashion across many machines.
- Scalable. Size of within memory data structures is bounded.
- High performance. Performance is limited by speed of Internet connection (e.g., with 160 Mbit/sec connection, downloads 50 million documents per day).
- Polite. Options of weak or strong politeness.

- Continuous. Will support continuous crawling.

Heritrix: Main Components

Scope: Determines what URIs are ruled into or out of a certain crawl. Includes the **seed URIs** used to start a crawl, plus the rules to determine which discovered URIs are also to be scheduled for download.

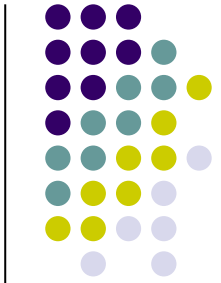
Frontier: Tracks which URIs are scheduled to be collected, and those that have already been collected. It is responsible for selecting the next URI to be tried, and prevents the redundant rescheduling of already-scheduled URIs.

Processor Chains: Modular Processors that perform specific, ordered actions on each URI in turn. These include fetching the URI, analyzing the returned results, and passing discovered URIs back to the Frontier.

Mercator: Main Components

- Crawling is carried out by multiple **worker threads**, e.g., 500 threads for a big crawl.
- The **URL frontier** stores the list of absolute URLs to download.
- The **DNS resolver** resolves domain names into IP addresses.
- **Protocol modules** download documents using appropriate protocol (e.g., HTML).
- **Link extractor** extracts URLs from pages and converts to absolute URLs.
- **URL filter** and **duplicate URL eliminator** determine which URLs to add to frontier.

Mercator: The URL Frontier



A repository with two pluggable methods: add a URL, get a URL.

Most web crawlers use variations of breadth-first traversal, but ...

- Most URLs on a web page are relative (about 80%).
- A single FIFO queue, serving many threads, would send many simultaneous requests to a single server.

Weak politeness guarantee: Only one thread allowed to contact a particular web server.

Stronger politeness guarantee: Maintain *n* FIFO queues, each for a single host, which feed the queues for the crawling threads by rules based on priority and politeness factors.

Mercator: Duplicate URL Elimination

Duplicate URLs are not added to the URL Frontier

Requires efficient data structure to store all URLs that have been seen and to check a new URL.

In memory:

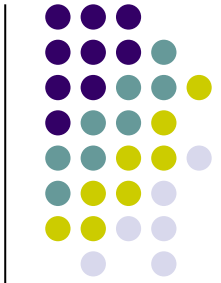
Represent URL by 8-byte checksum. Maintain in-memory hash table of URLs.

Requires 5 Gigabytes for 1 billion URLs.

Disk based:

Combination of disk file and in-memory cache with batch updating to minimize disk head movement.

Mercator: Domain Name Lookup



Resolving domain names to IP addresses is a major bottleneck of web crawlers.

Approach:

- Separate DNS resolver and cache on each crawling computer.
- Create multi-threaded version of DNS code (BIND).

These changes reduced DNS loop-up from 70% to 14% of each thread's elapsed time.