Web Crawling

Introduction to Information Retrieval

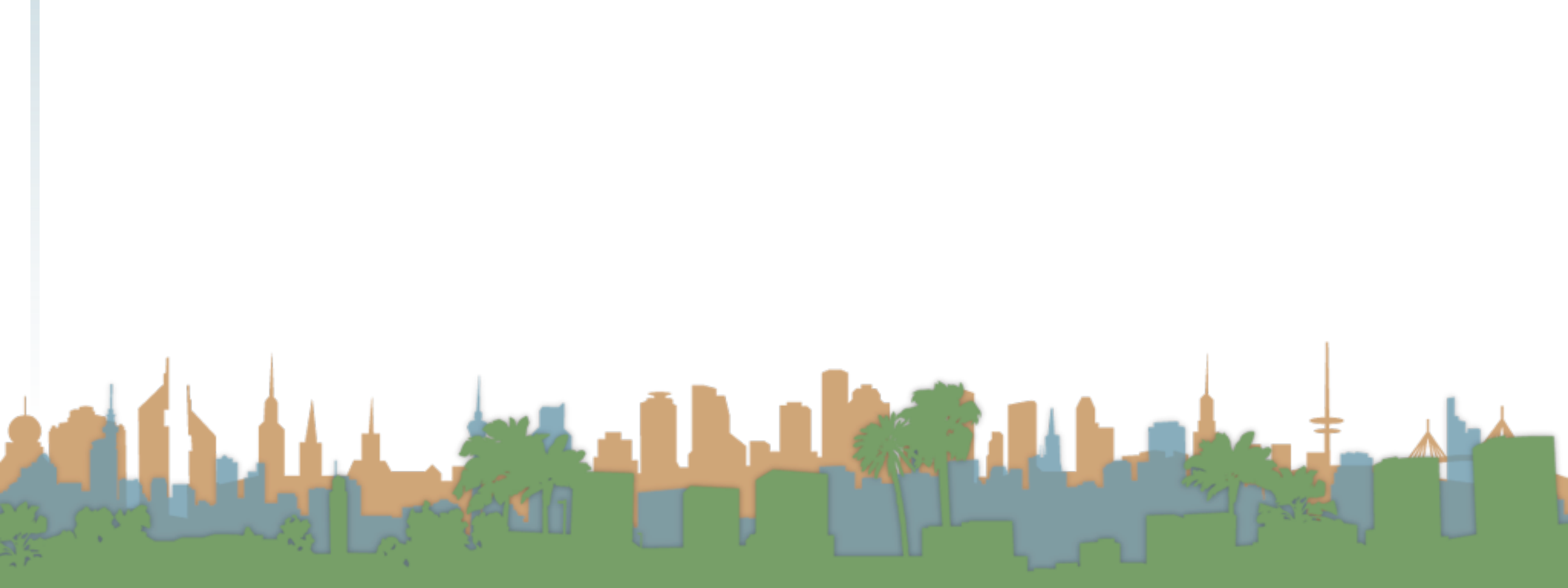
Informatics 141 / CS 121
Donald J. Patterson

Content adapted from Hinrich Schütze
http://www.informationretrieval.org

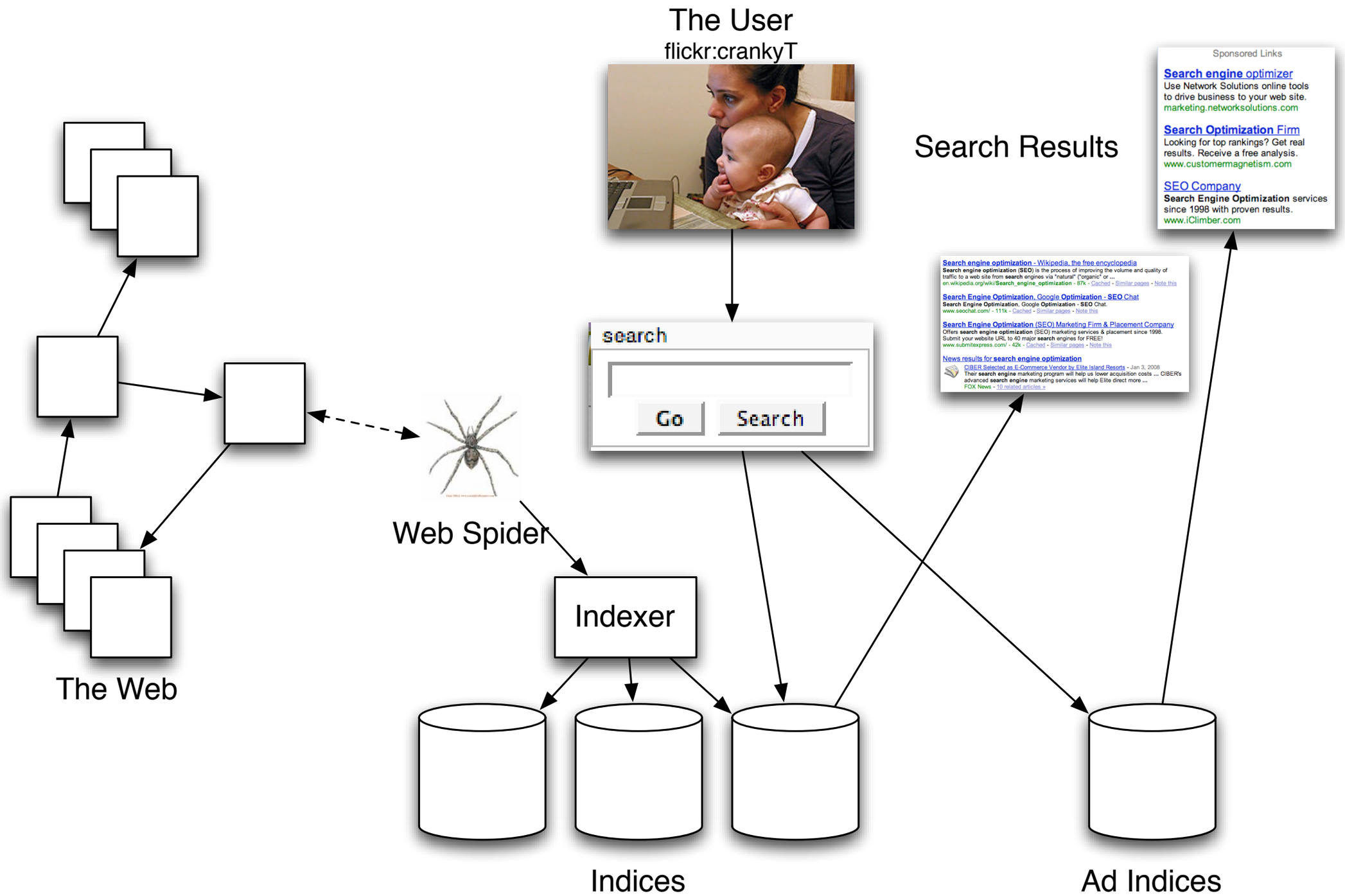
Web Crawling Outline

Overview

- Introduction
- URL Frontier
- Robust Crawling
  - DNS



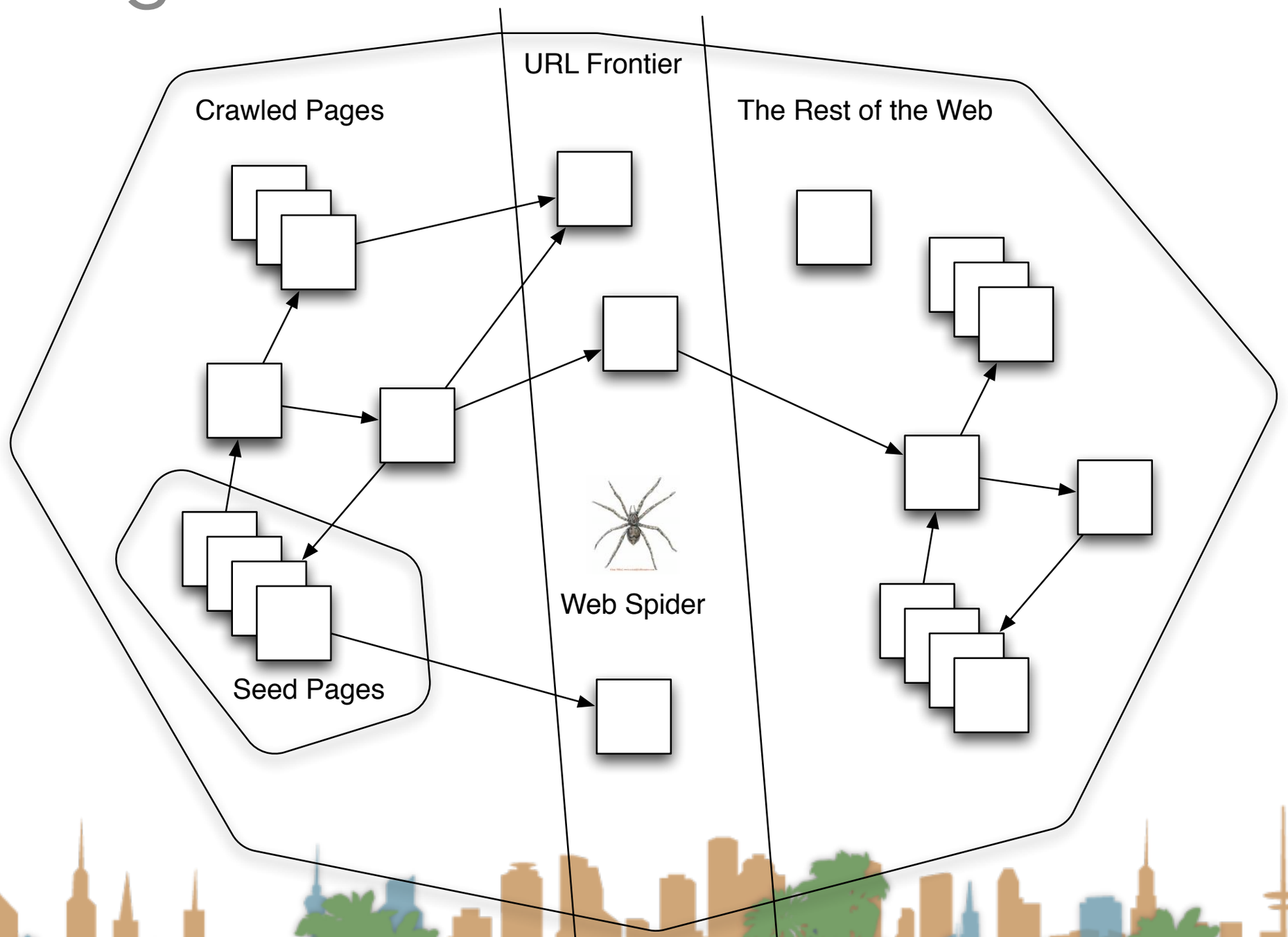
Introduction



The basic crawl algorithm

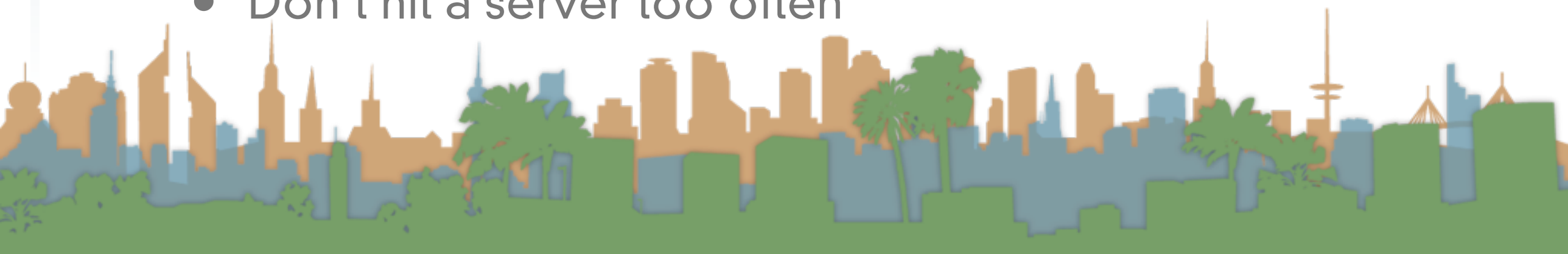
- Initialize a queue of URLs ("seed" URLs)
- Repeat
  - Remove a URL from the queue
  - Fetch associated page
  - Parse and analyze page
  - Store representation of page
  - Extract URLs from page and add to queue

Crawling the web



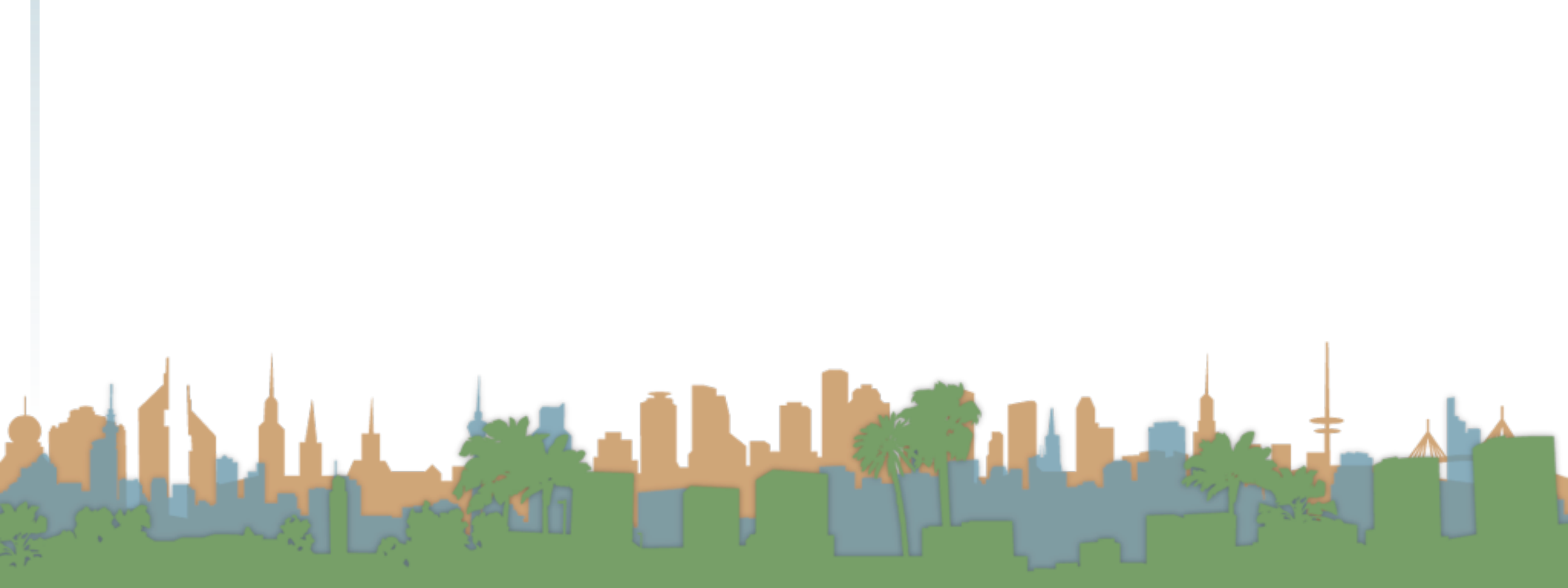
Basic Algorithm is not reality...

- Real web crawling requires multiple machines
  - All steps distributed on different computers
- Even Non-Adversarial pages pose problems
  - Latency and bandwidth to remote servers vary
  - Webmasters have opinions about crawling their turf
    - How “deep” in a URL should you go?
  - Site mirrors and duplicate pages
- Politeness
  - Don’t hit a server too often



Basic Algorithm is not reality...

- Adversarial Web Pages
  - Spam Pages
  - Spider Traps



Minimum Characteristics for a Web Crawler

- Be Polite:
  - Respect implicit and explicit terms on website
  - Crawl pages you're allowed to
  - Respect "robots.txt" (more on this coming up)
- Be Robust
  - Handle traps and spam gracefully

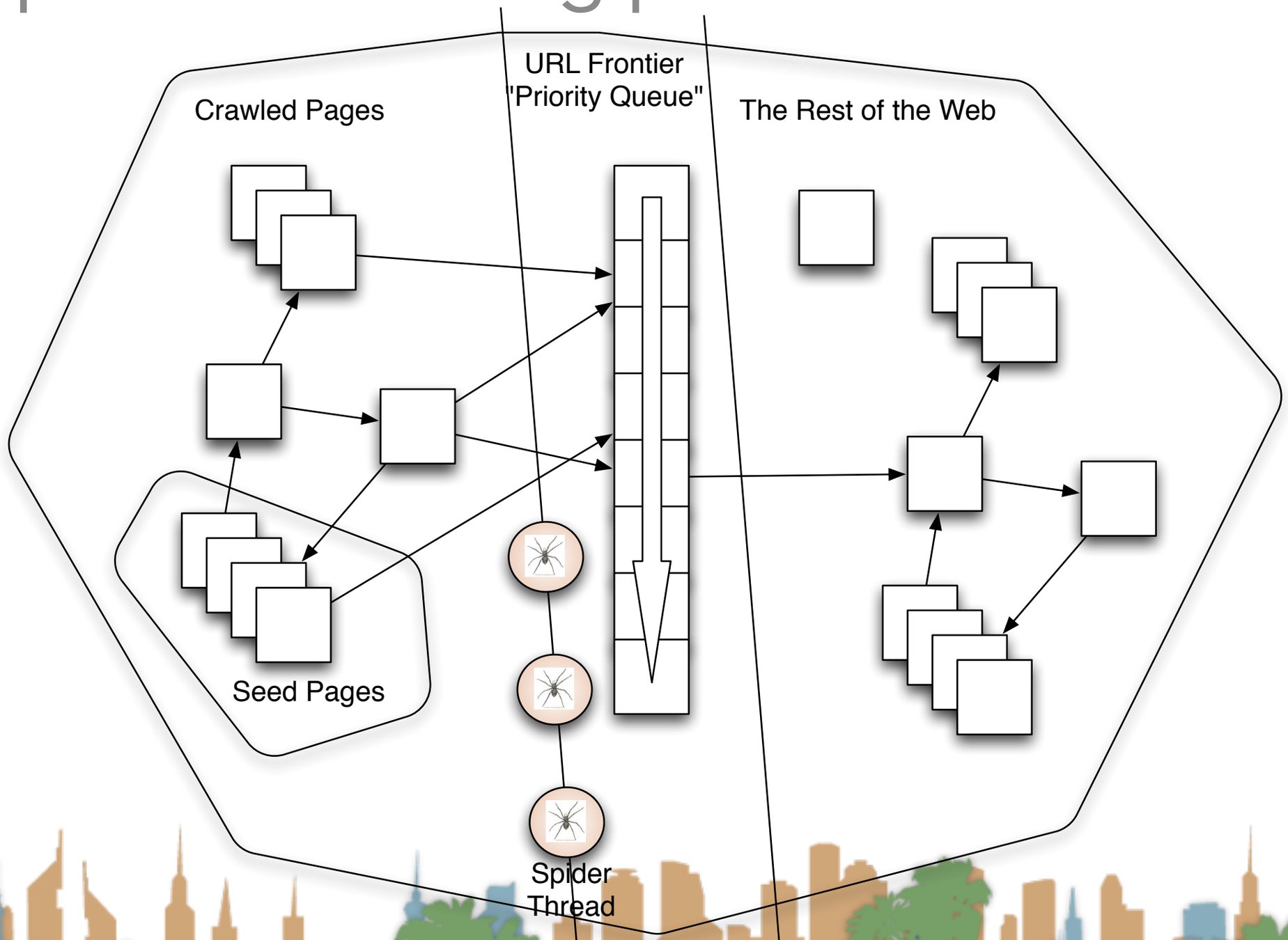
Desired Characteristics for a Web Crawler

- Be a distributed systems
  - Run on multiple machines
- Be scalable
  - Adding more machines allows you to crawl faster
- Be Efficient
  - Fully utilize available processing and bandwidth
- Focus on “Quality” Pages
  - Crawl good information first

Desired Characteristics for a Web Crawler

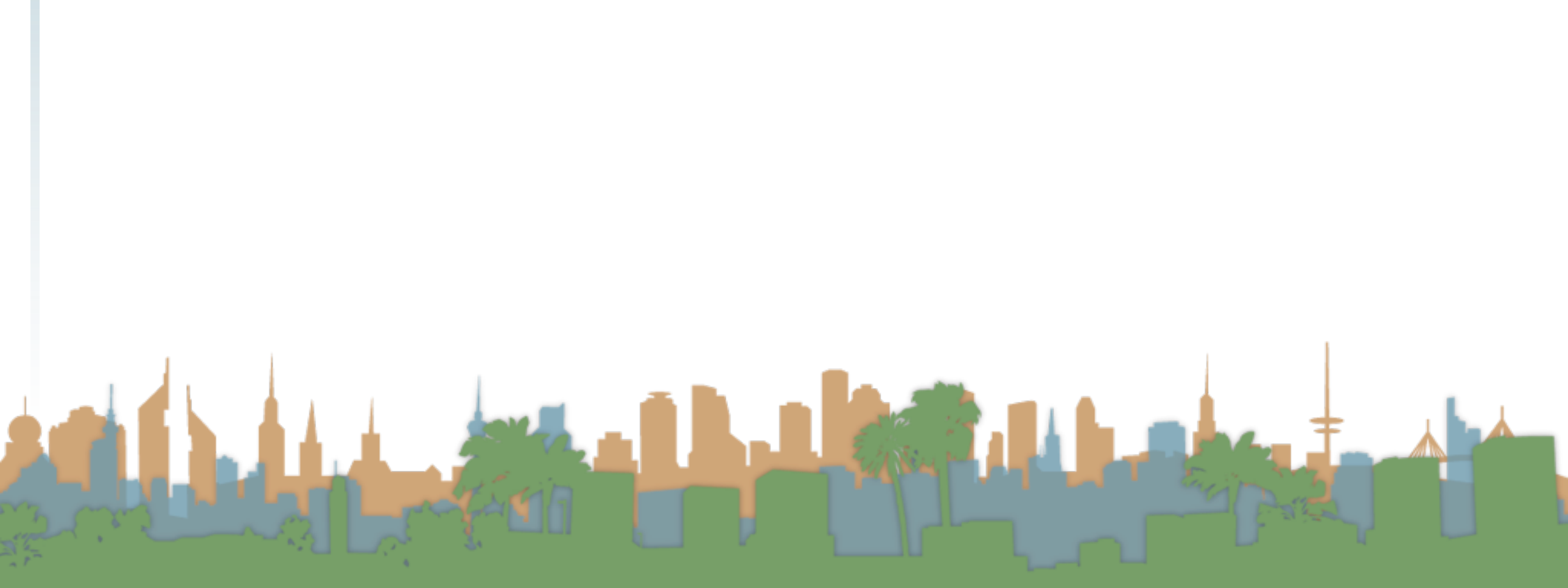
- Support Continuous Operation
  - Fetch fresh copies of previously crawled pages
- Be Extensible
  - Be able to adapt to new data formats, protocols, etc.
    - Today it's AJAX, tomorrow it's SilverLight, then....

Updated Crawling picture



URL Frontier

- Frontier Queue might have multiple pages from the same host
  - These need to be load balanced (“politeness”)
- All crawl threads should be kept busy

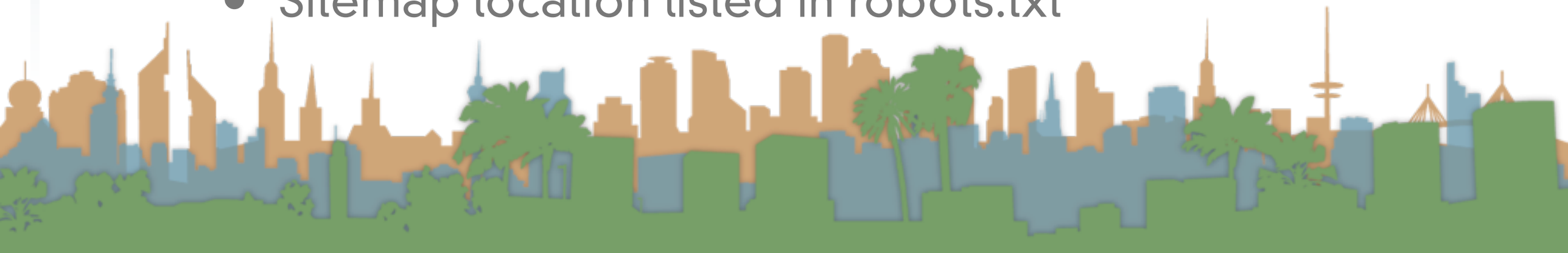


Politeness?

- It is easy enough for a website to block a crawler
- Explicit Politeness
  - "Robots Exclusion Standard"
    - Defined by a "robots.txt" file maintained by a webmaster
      - What portions of the site can be crawled.
        - Irrelevant, private or other data excluded.
      - Voluntary compliance by crawlers.
      - Based on regular expression matching

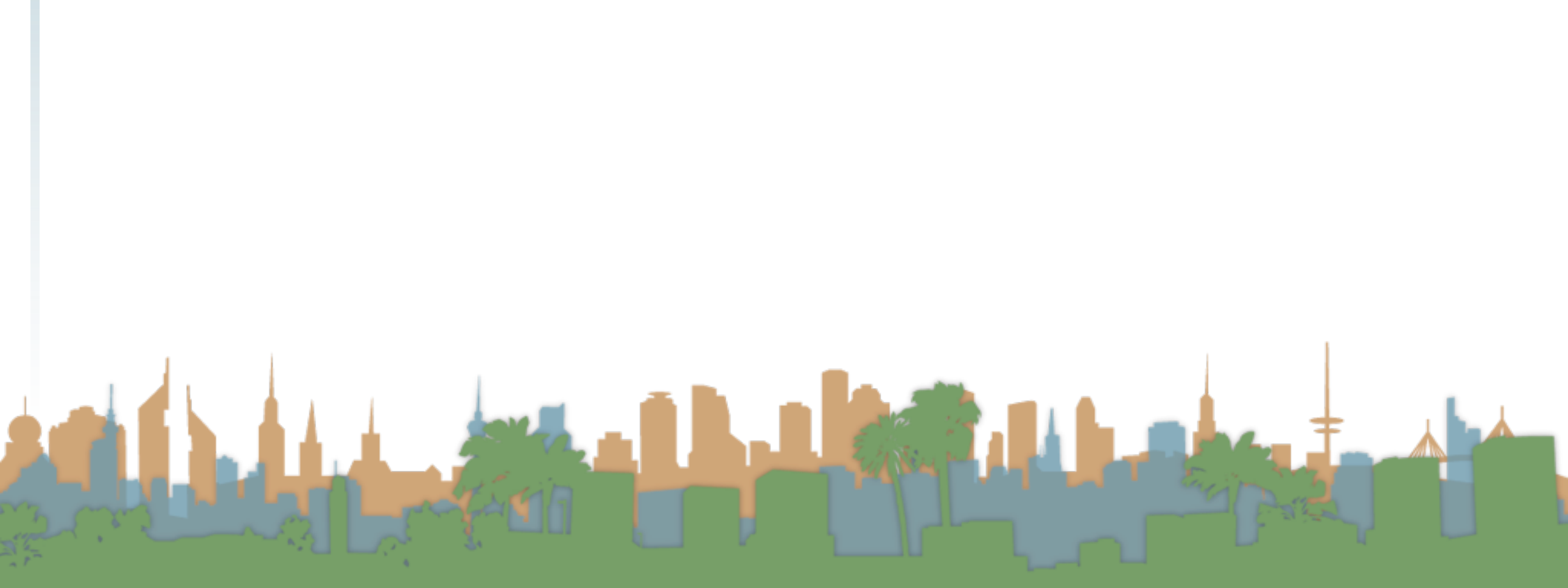
Politeness?

- Explicit Politeness
  - “Sitemaps”
    - Introduced by Google, but open standard
    - XML based
    - Allows webmasters to specify:
      - Location of pages (URL islands)
      - Importance of pages
      - Update frequency of pages
- Sitemap location listed in robots.txt



Politeness?

- Implicit Politeness
  - Even without specification avoid hitting any site too often
  - It costs bandwidth and computing resources for host.



Politeness?

Statistics for:
djp3.net

Summary
When:
Monthly history
Days of month
Days of week
Hours
Who:
Countries
- Full list

Hosts
- Full list
- Last visit
- Unresolved IP Address

Robots/Spiders visitors
- Full list
- Last visit

Navigation:
Visits duration
File type
Viewed
- Full list
- Entry
- Exit

Operating Systems
- Versions
- Unknown

Browsers
- Versions
- Unknown

Referers:
Origin
- Refering search engines
- Refering sites

Search
- Search Keyphrases
- Search Keywords

Last Update: 14 Jan 2008 - 02:59
Reported period: - Year - 2007 OK

speakeasy

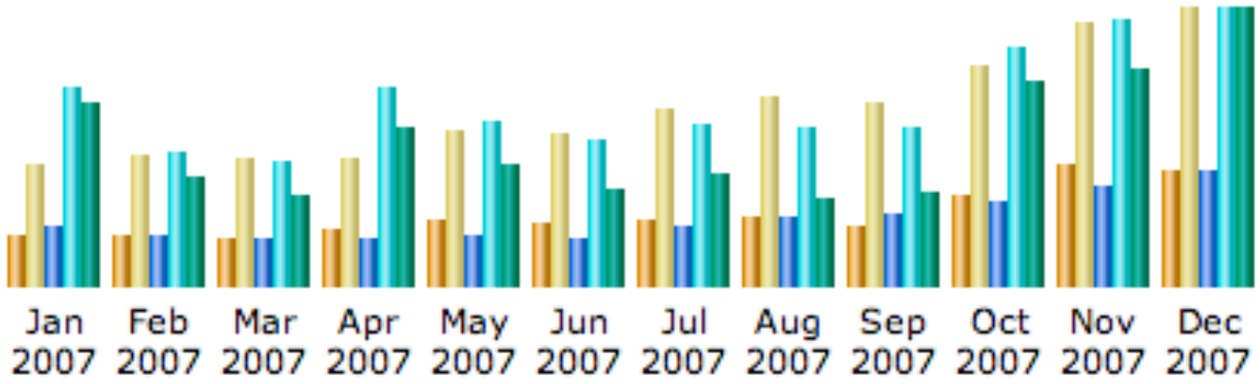
Back to main page

Robots/Spiders visitors

30 different robots	Hits	Bandwidth	Last visit
Googlebot	1393868+104	5.11 GB	31 Dec 2007 - 23:50
Inktomi Slurp	36668+221	554.25 MB	31 Dec 2007 - 23:55
MSNBot	19522+2	699.90 MB	28 Dec 2007 - 08:01
Unknown robot (identified by 'crawl')	15949+13	89.34 MB	31 Dec 2007 - 22:24
AskJeeves	7016+1	106.29 MB	31 Dec 2007 - 23:49
Google AdSense	2701	100.26 MB	31 Dec 2007 - 22:10
psbot	2268+1	80.48 MB	31 Dec 2007 - 09:59
Unknown robot (identified by 'robot')	930+1	19.10 MB	31 Dec 2007 - 09:34
Turn It In	350+1	6.32 MB	03 Sep 2007 - 15:44
BaiDuSpider	300	10.22 MB	26 Nov 2007 - 07:32
GigaBot	243	5.27 MB	30 Dec 2007 - 05:06
Scooter	90+3	288.75 KB	27 Nov 2007 - 14:30
PhpDig	91	2.28 MB	21 Oct 2007 - 09:51
WISENutbot	76	1.94 MB	13 Jan 2007 - 14:04
Magpie	25	43.48 KB	24 Dec 2007 - 00:51
Unknown robot (identified by hit on 'robots.txt')	0+16	4.38 KB	14 Nov 2007 - 03:43
EchO!	14	287.09 KB	27 Dec 2007 - 13:56
Internet Shinchakubin	13	385.03 KB	27 Nov 2007 - 15:23
BBot	10	146.35 KB	13 Jun 2007 - 15:17
arks	8	142.24 KB	27 Nov 2007 - 12:25
MSIECrawler	8	263.02 KB	26 Dec 2007 - 11:16

Politeness?

Monthly history



Month	Unique visitors	Number of visits	Pages	Hits	Bandwidth
Jan 2007	1221	2946	8938	30536	699.28 MB
Feb 2007	1179	3099	7852	20475	415.75 MB
Mar 2007	1120	3063	7099	18978	350.88 MB
Apr 2007	1362	3067	7175	30320	599.91 MB
May 2007	1612	3746	7584	25114	469.32 MB
Jun 2007	1474	3662	7138	22292	370.11 MB
Jul 2007	1592	4210	9165	24766	430.61 MB
Aug 2007	1658	4567	10600	24142	336.08 MB
Sep 2007	1458	4403	11149	24414	356.60 MB
Oct 2007	2148	5299	12877	36427	783.78 MB
Nov 2007	2890	6317	15300	40487	833.75 MB
Dec 2007	2748	6631	17553	42281	1.03 GB
Total	20462	51010	122430	340232	6.55 GB

Robots.txt - Exclusion

- Protocol for giving spiders ("robots") limited access to a website
  - Source: http://www.robotstxt.org/wc/norobots.html
- Website announces what is okay and not okay to crawl:
- Located at http://www.myurl.com/robots.txt
- This file holds the restrictions

Robots.txt Example

- http://www.ics.uci.edu/robots.txt

```
User-agent: MOMspider              # The Multi-Owner Maintenance Spider
Disallow: /cgi-bin/                #     Script files
Disallow: /Admin/MOM/              #     Local MOMspider output
Disallow: /~fielding/MOM/          #     Local MOMspider output
Disallow: /TR/                     #     Dienst Technical Report Server
Disallow: /Server/                 #     Dienst Technical Report Server
Disallow: /Document/               #     Dienst Technical Report Server
Disallow: /MetaServer/             #     Dienst Technical Report Server
Disallow: /~eppstein/pubs/cites/           #     Eppstein Database
Disallow: /~fiorello/pvt/          #     Private pages

User-agent: *                      # All other spiders should avoid
Disallow: /cgi-bin/                #     Script files
Disallow: /Test/                   #     The test area for web experimentation
Disallow: /Admin/                  #     Huge server statistic logs
Disallow: /TR/                     #     Dienst Technical Report Server
Disallow: /Server/                 #     Dienst Technical Report Server
Disallow: /Document/               #     Dienst Technical Report Server
Disallow: /MetaServer/             #     Dienst Technical Report Server
Disallow: /~fielding/MOM/          #     Local MOMspider output
Disallow: /~kanderso/hidden        #     Ken Anderson's stuff
Disallow: /~eppstein/pubs/cites/           #     Eppstein Database
Disallow: /~fiorello/pvt/          #     Private pages
Disallow: /~dean/
Disallow: /~wwwoffic/
Disallow: /~ucounsel/
Disallow: /~sao/
Disallow: /~support/
Disallow: /~icsdb/
Disallow: /bin/
```

Sitemaps - Inclusion

- https://www.google.com/webmasters/tools/docs/en/protocol.html#sitemapXMLExample

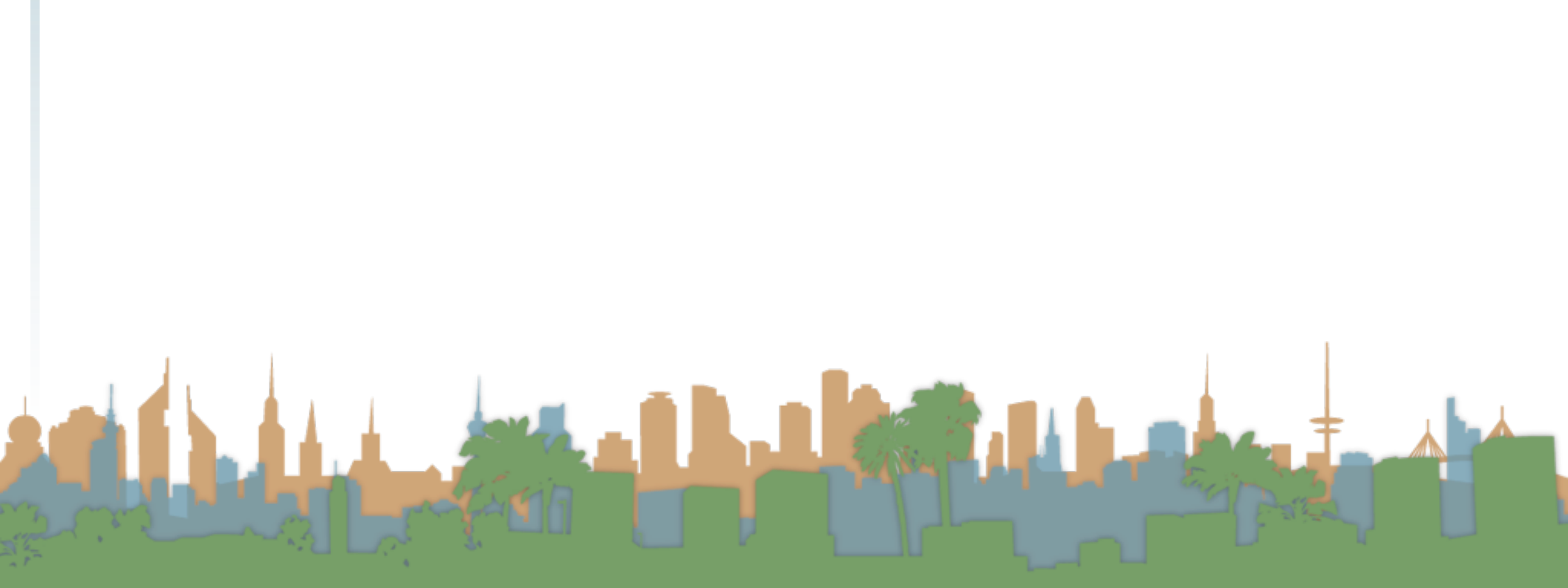
```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">

   <url>
      <loc>http://www.example.com/</loc>
      <lastmod>2005-01-01</lastmod>
      <changefreq>monthly</changefreq>
      <priority>0.8</priority>
   </url>
   <url>
      <loc>http://www.example.com/catalog?item=12&amp;desc=vacation_hawaii</loc>
      <changefreq>weekly</changefreq>
   </url>
   <url>
      <loc>http://www.example.com/catalog?item=73&amp;desc=vacation_new_zealand</loc>
      <lastmod>2004-12-23</lastmod>
      <changefreq>weekly</changefreq>
   </url>
   <url>
      <loc>http://www.example.com/catalog?item=74&amp;desc=vacation_newfoundland</loc>
      <lastmod>2004-12-23T18:00:15+00:00</lastmod>
      <priority>0.3</priority>
   </url>
   <url>
      <loc>http://www.example.com/catalog?item=83&amp;desc=vacation_usa</loc>
      <lastmod>2004-11-23</lastmod>
   </url>
</urlset>
```

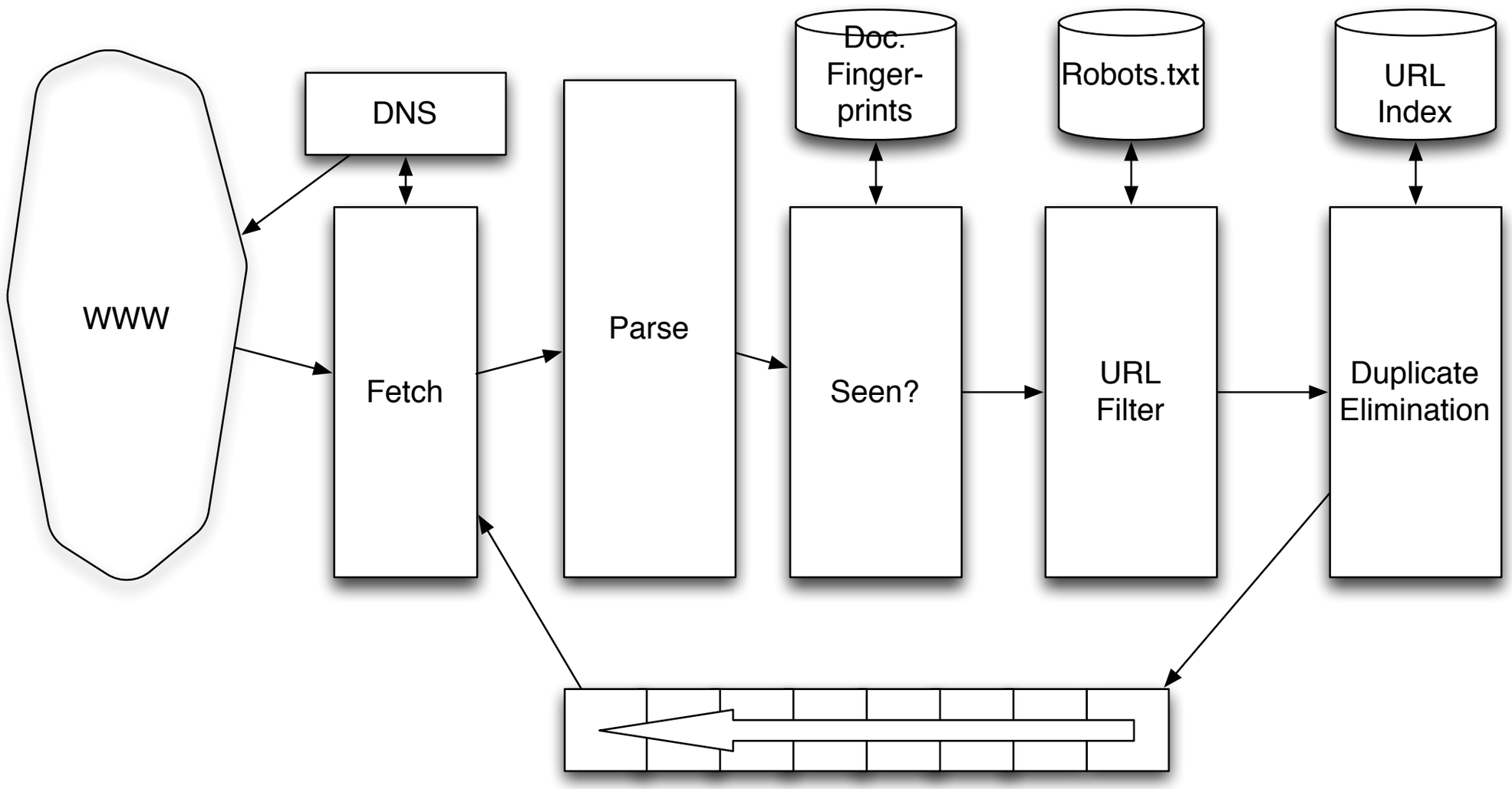
Web Crawling Outline

Overview

- Introduction
- URL Frontier
- Robust Crawling
  - DNS



A Robust Crawl Architecture



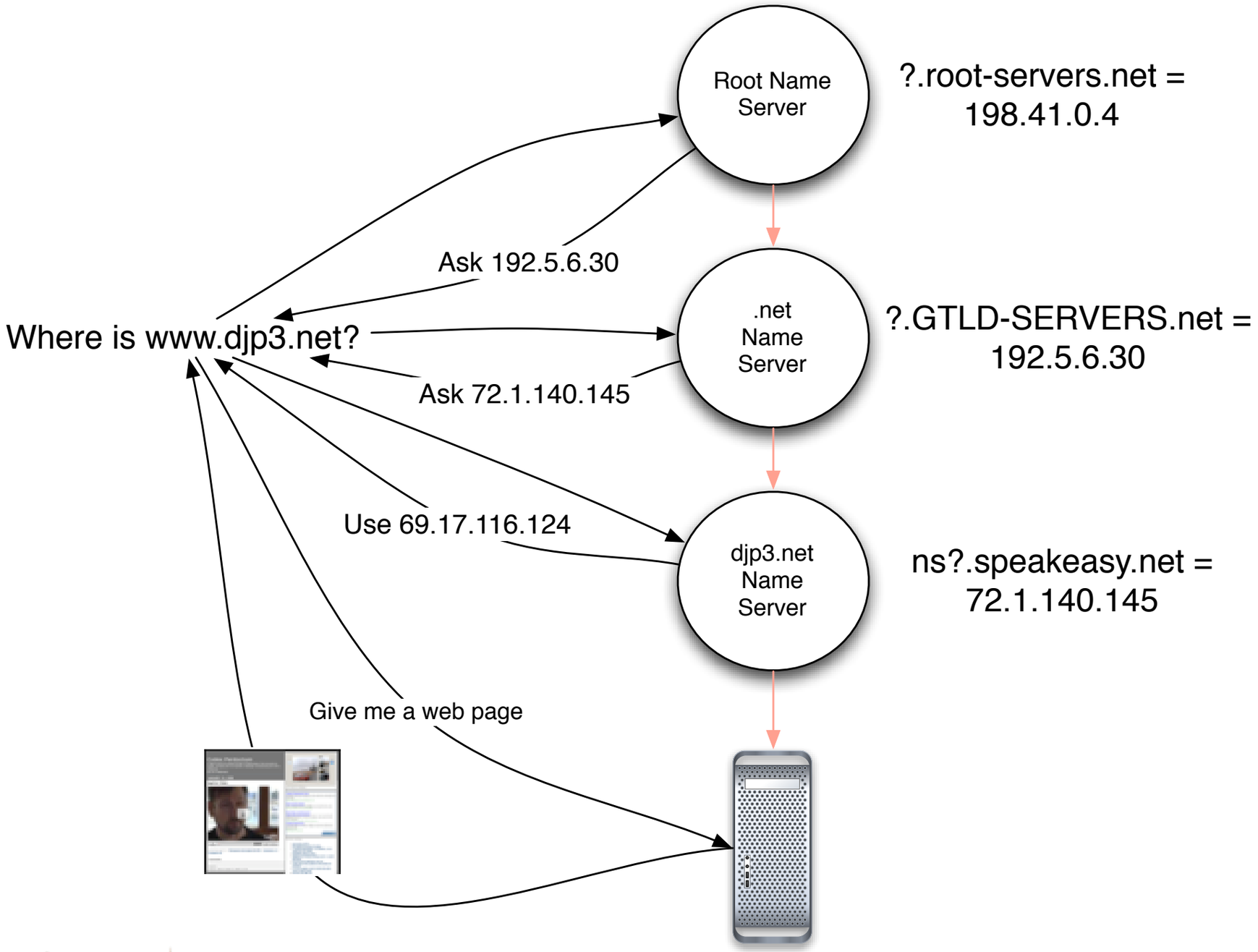
Processing Steps in Crawling

- Pick a URL from the frontier (how to prioritize?)
- Fetch the document (DNS lookup)
- Parse the URL
  - Extract Links
  - Check for duplicate content
    - If not add to index
- For each extracted link
  - Make sure it passes filter (robots.txt)
  - Make sure it isn’t in the URL frontier

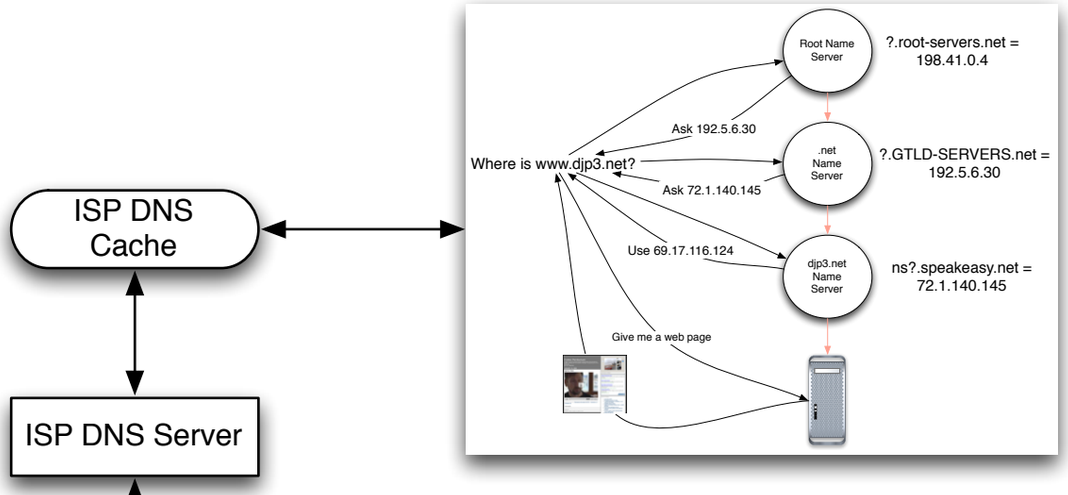
Domain Name Server

- A lookup service on the internet
  - Given a URL, retrieve its IP address
    - www.djp3.net -> 69.17.116.124
- This service is provided by a distributed set of servers
  - Latency can be high
    - Even seconds
- Common OS implementations of DNS lookup are blocking
  - One request at a time
- Solution:
  - Caching
  - Batch requests

dig +trace www.djp3.net

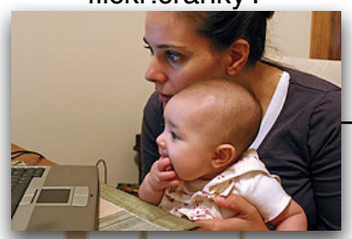


What really happens



The User

flickr:crankyT



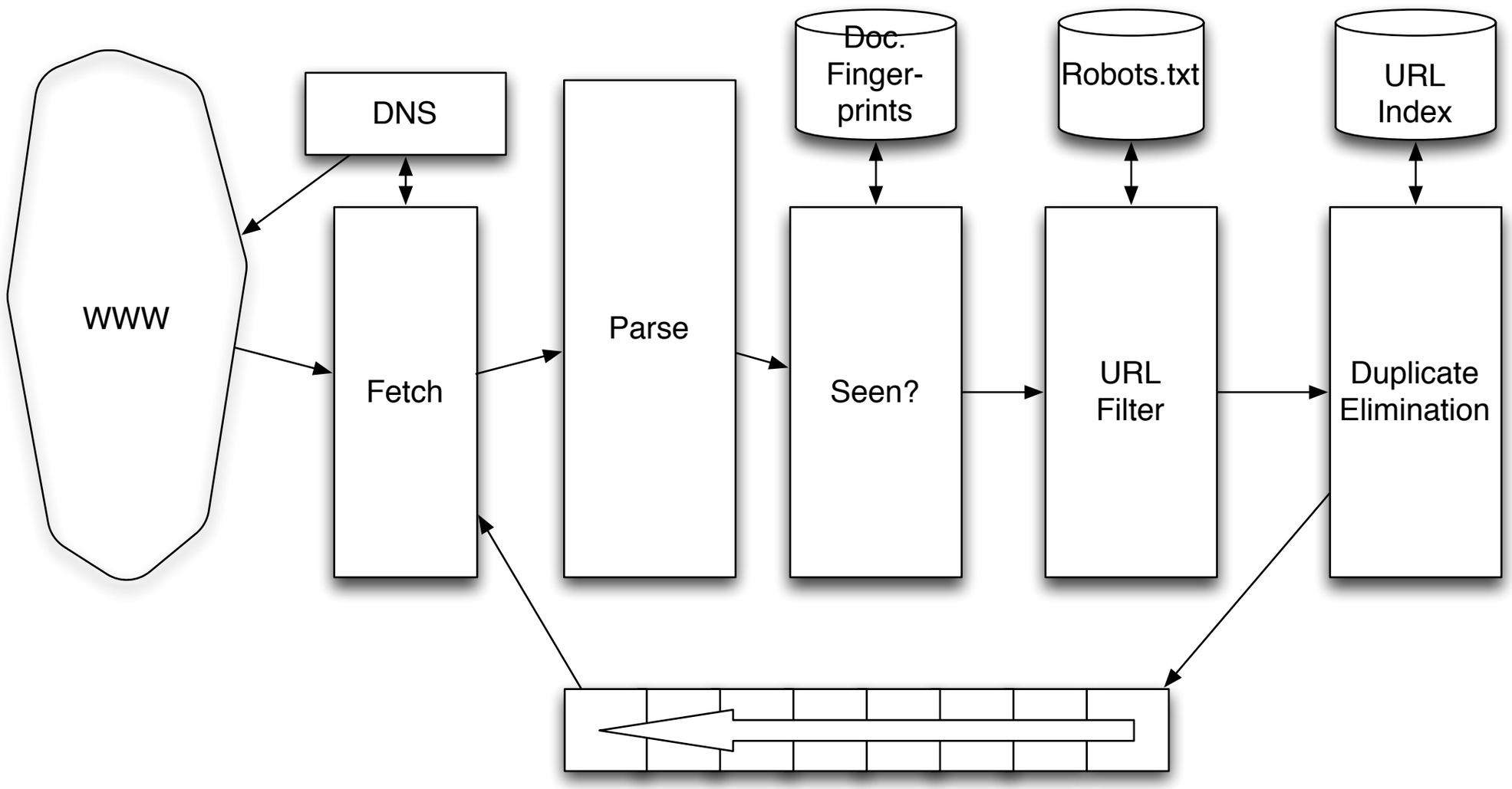
Give me a web page

Class Exercise

- Calculate how long it would take to completely fill a DNS cache.
  - How many active hosts are there?
  - What is an average lookup time?
  - Do the math.

http://www.flickr.com/photos/lurie/298967218/

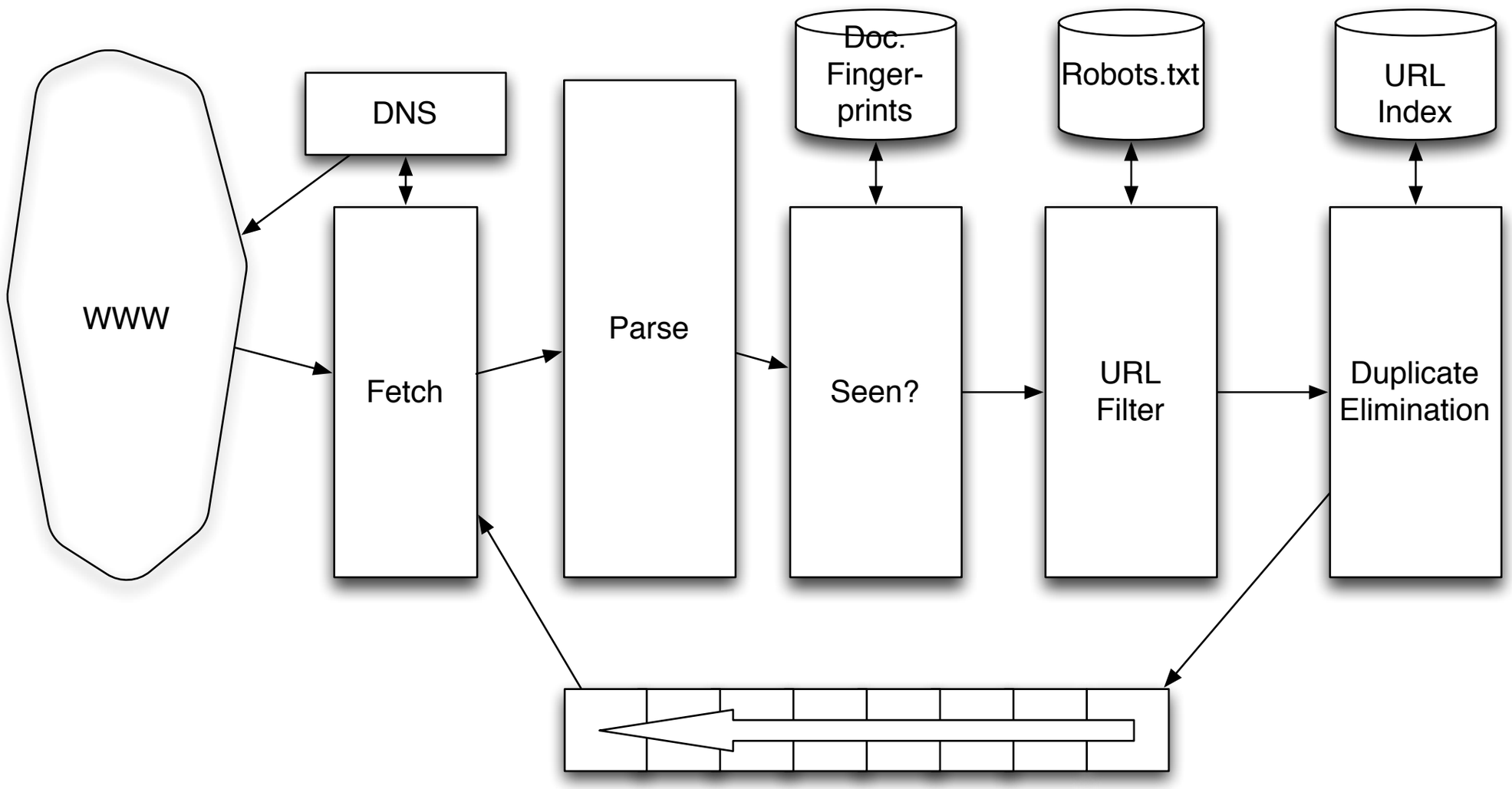
A Robust Crawl Architecture



Parsing: URL normalization

- When a fetched document is parsed
  - some outlink URLs are relative
    - For example:
      - http://en.wikipedia.org/wiki/Main_Page
      - has a link to "/wiki/Special:Statistics"
      - which is the same as
      - http://en.wikipedia.org/wiki/Special:Statistics
- Parsing involves normalizing (expanding) relative URLs

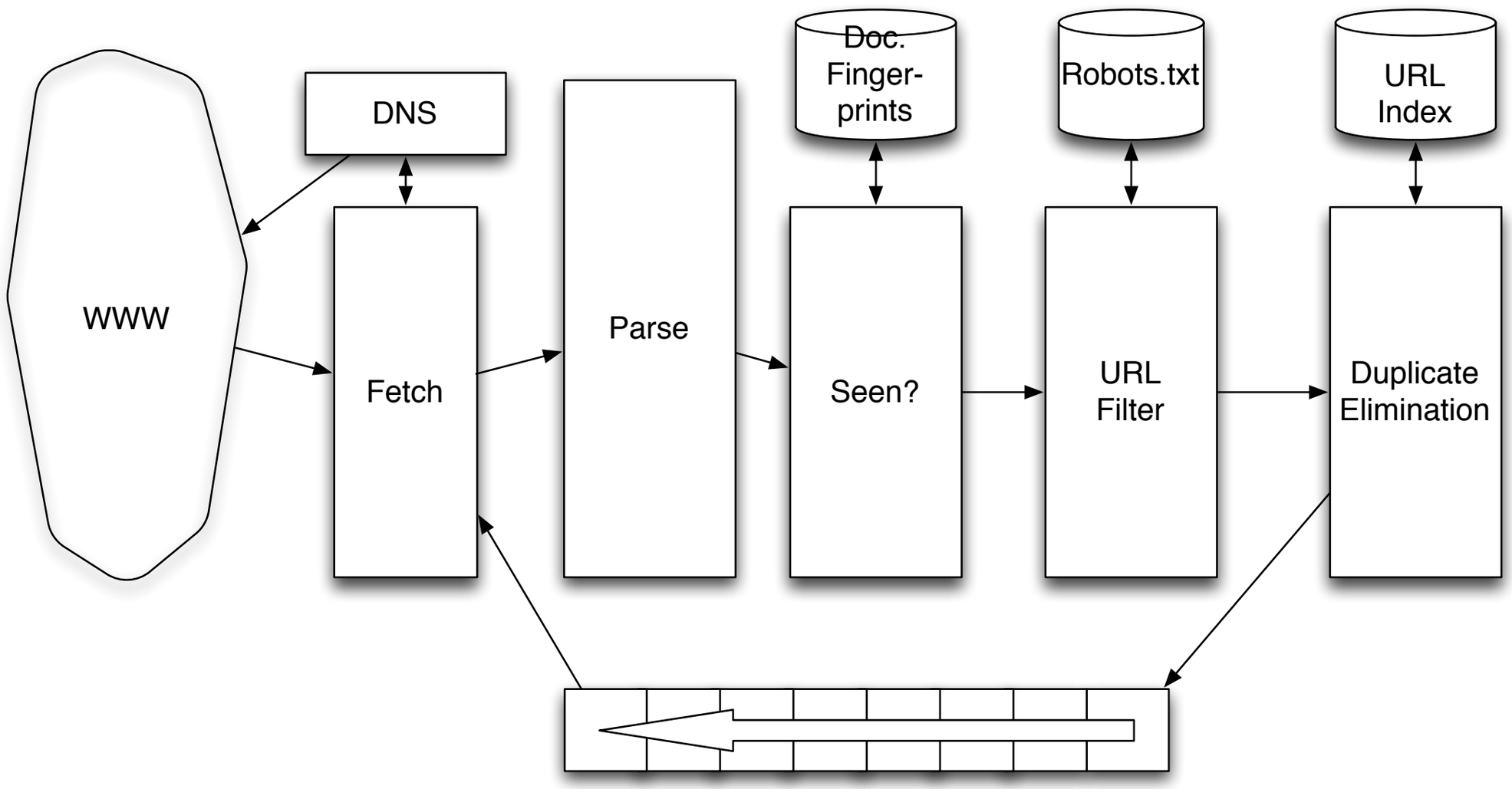
A Robust Crawl Architecture



Content Seen?

- Duplication is widespread on the web
- If a page just fetched is already in the index, don't process it any further
- This can be done by using document fingerprints/shingles
  - A type of hashing scheme

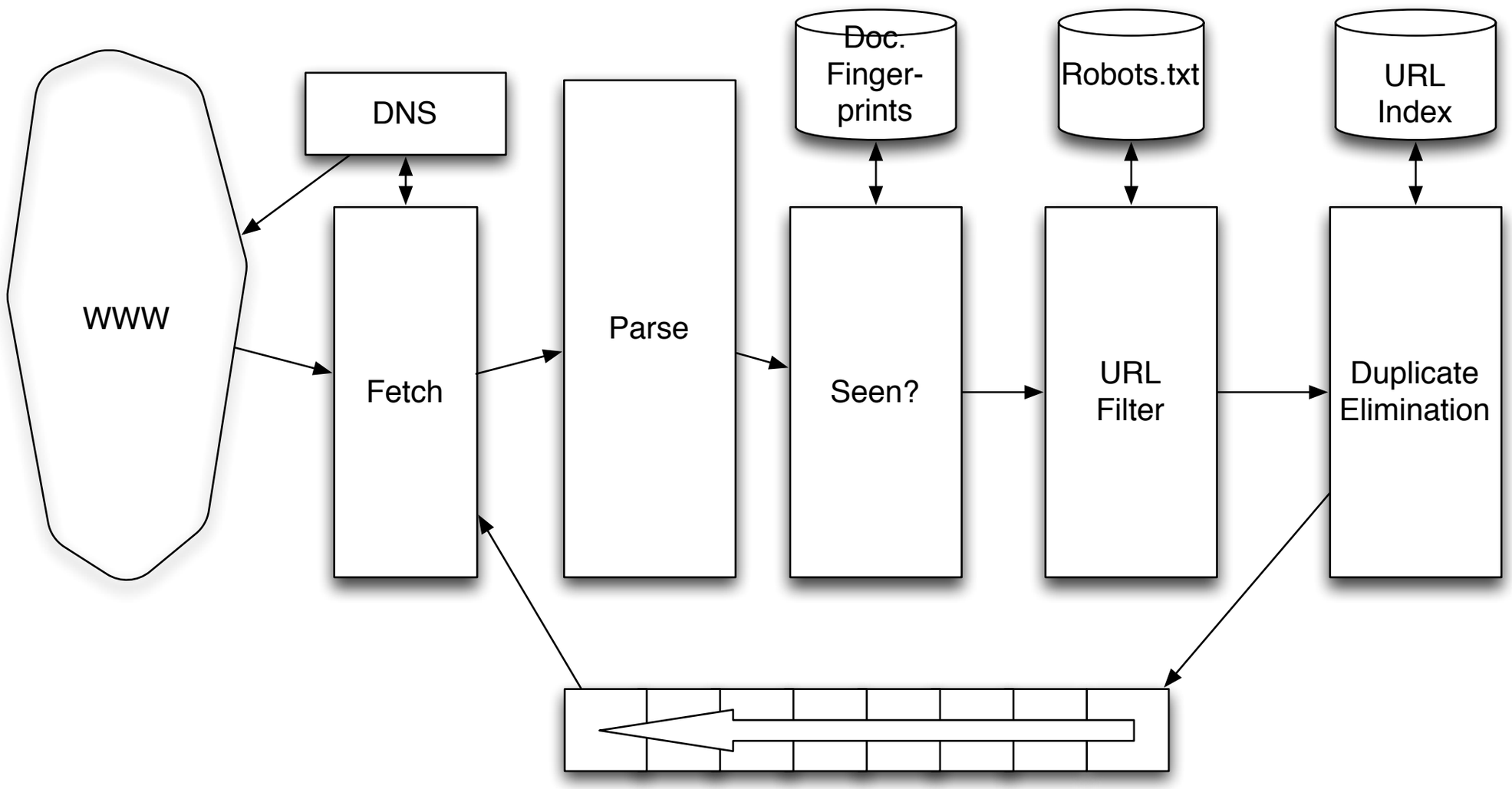
A Robust Crawl Architecture



Compliance with webmasters wishes...

- Robots.txt
  - Filters is a regular expression for a URL to be excluded
  - How often do you check robots.txt?
    - Cache to avoid using bandwidth and loading web server
- Sitemaps
  - A mechanism to better manage the URL frontier

A Robust Crawl Architecture



Duplicate Elimination

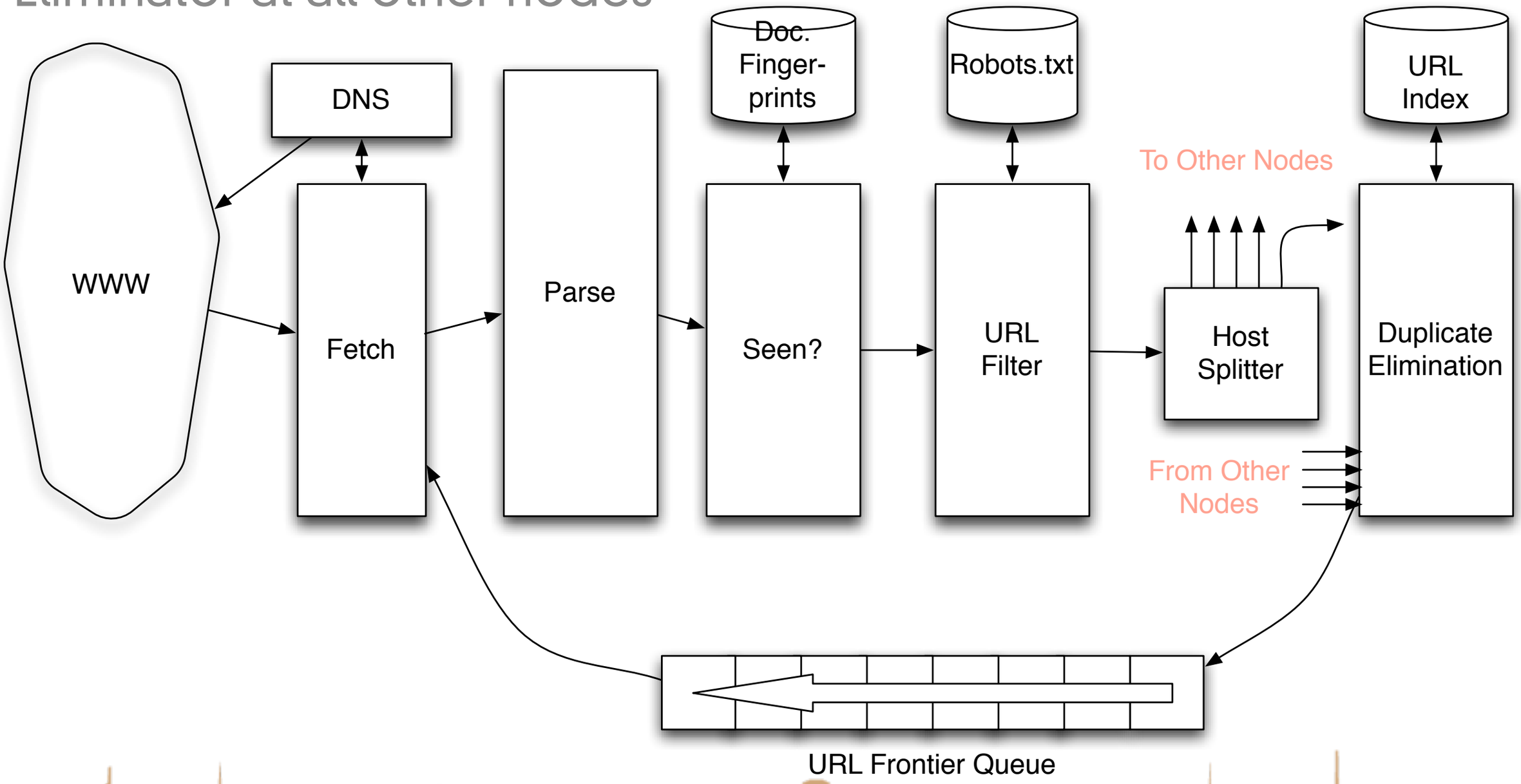
- For a one-time crawl
  - Test to see if an extracted,parsed, filtered URL
    - has already been sent to the frontier.
    - has already been indexed.
- For a continuous crawl
  - See full frontier implementation:
    - Update the URL's priority
      - Based on staleness
      - Based on quality
      - Based on politeness

Distributing the crawl

- The key goal for the architecture of a distributed crawl is cache locality
- We want multiple crawl threads in multiple processes at multiple nodes for robustness
  - Geographically distributed for speed
  - Partition the hosts being crawled across nodes
    - Hash typically used for partition
- How do the nodes communicate?

Robust Crawling

The output of the URL Filter at each node is sent to the Duplicate Eliminator at all other nodes

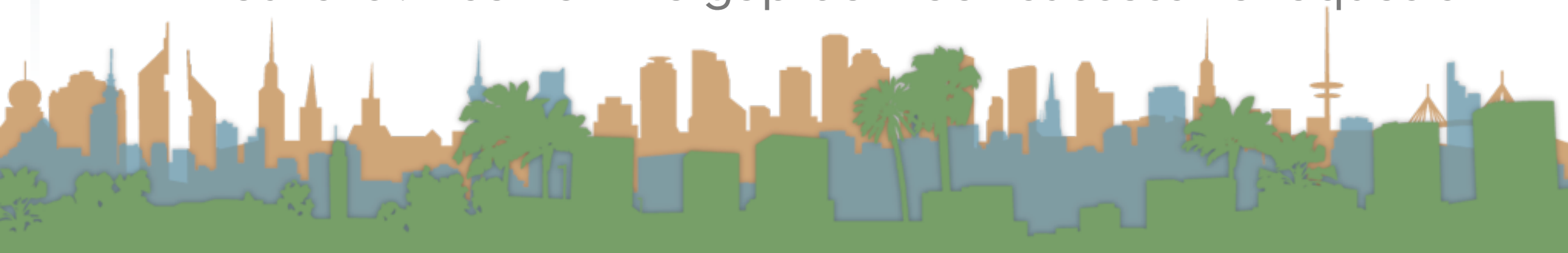


URL Frontier

- Freshness
  - Crawl some pages more often than others
    - Keep track of change rate of sites
    - Incorporate sitemap info
- Quality
  - High quality pages should be prioritized
    - Based on link-analysis, popularity, heuristics on content
- Politeness
  - When was the last time you hit a server?

URL Frontier

- Freshness, Quality and Politeness
  - These goals will conflict with eachother
  - A simple priority queue will fail because links are bursty
    - Many sites have lots of links pointing to themselves creating bursty references
    - Time influences the priority
- Politeness Challenges
  - Even if only one thread is assigned to hit a particular host it can hit it repeatedly
  - Heuristic : insert a time gap between successive requests



Magnitude of the crawl

- To fetch 1,000,000,000 pages in one month...
  - a small fraction of the web
- we need to fetch 400 pages per second !
- Since many fetches will be duplicates, unfetchable, filtered, etc. 400 pages per second isn't fast enough

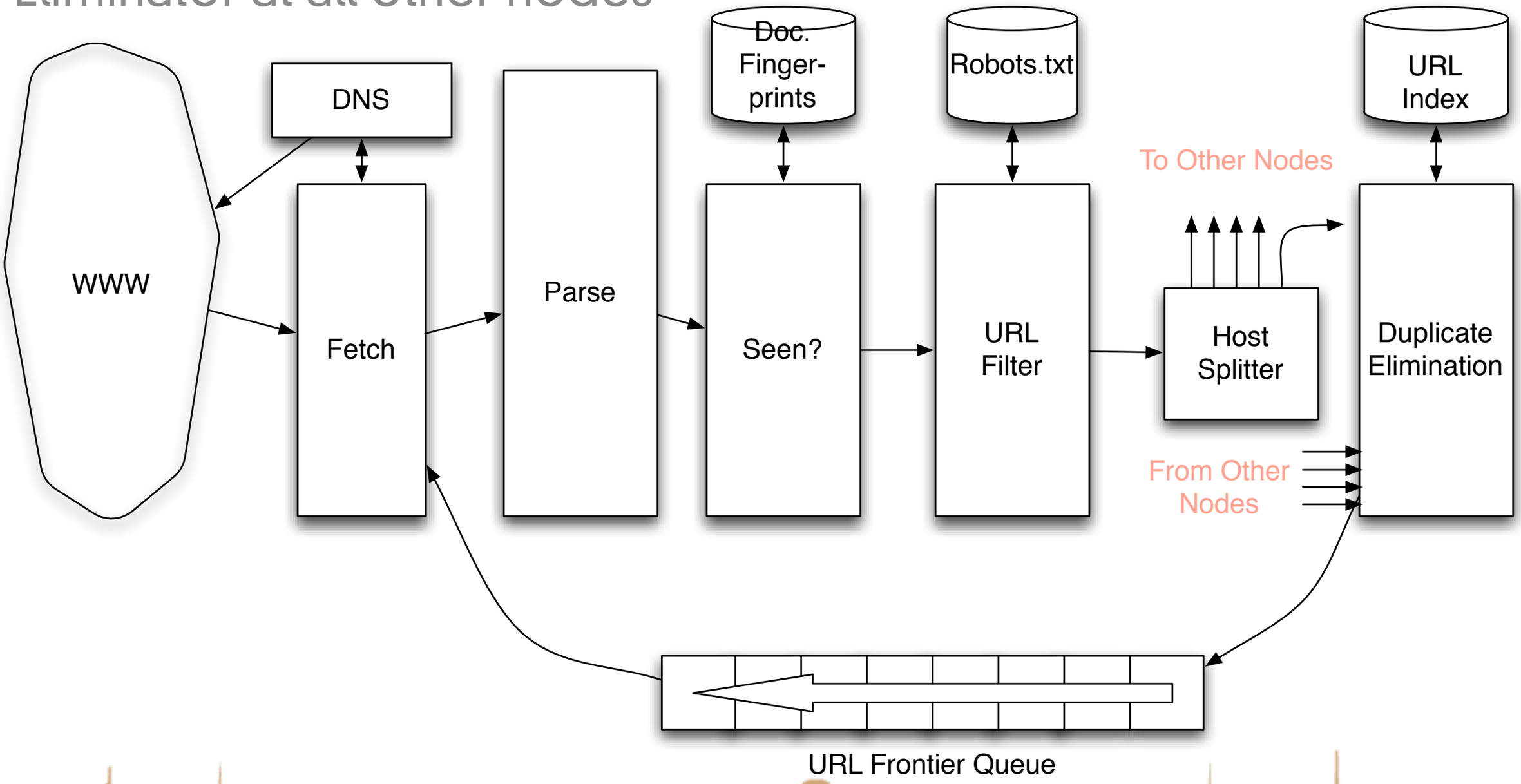
Web Crawling Outline

Overview

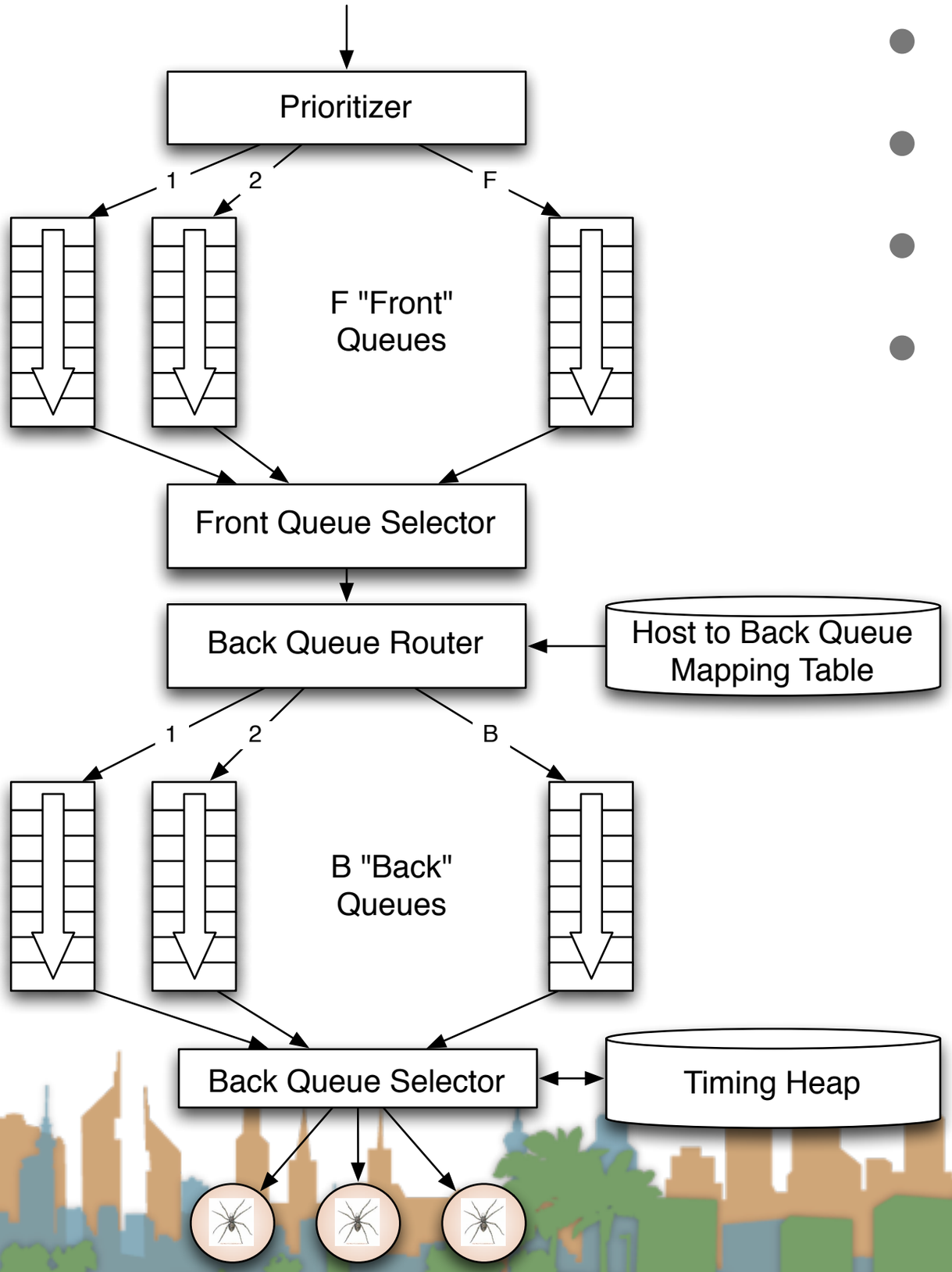
- Introduction
- URL Frontier
- Robust Crawling
  - DNS
  - Various parts of architecture
  - URL Frontier
- Index
  - Distributed Indices
  - Connectivity Servers

Robust Crawling

The output of the URL Filter at each node is sent to the Duplicate Eliminator at all other nodes



URL Frontier Implementation - Mercator

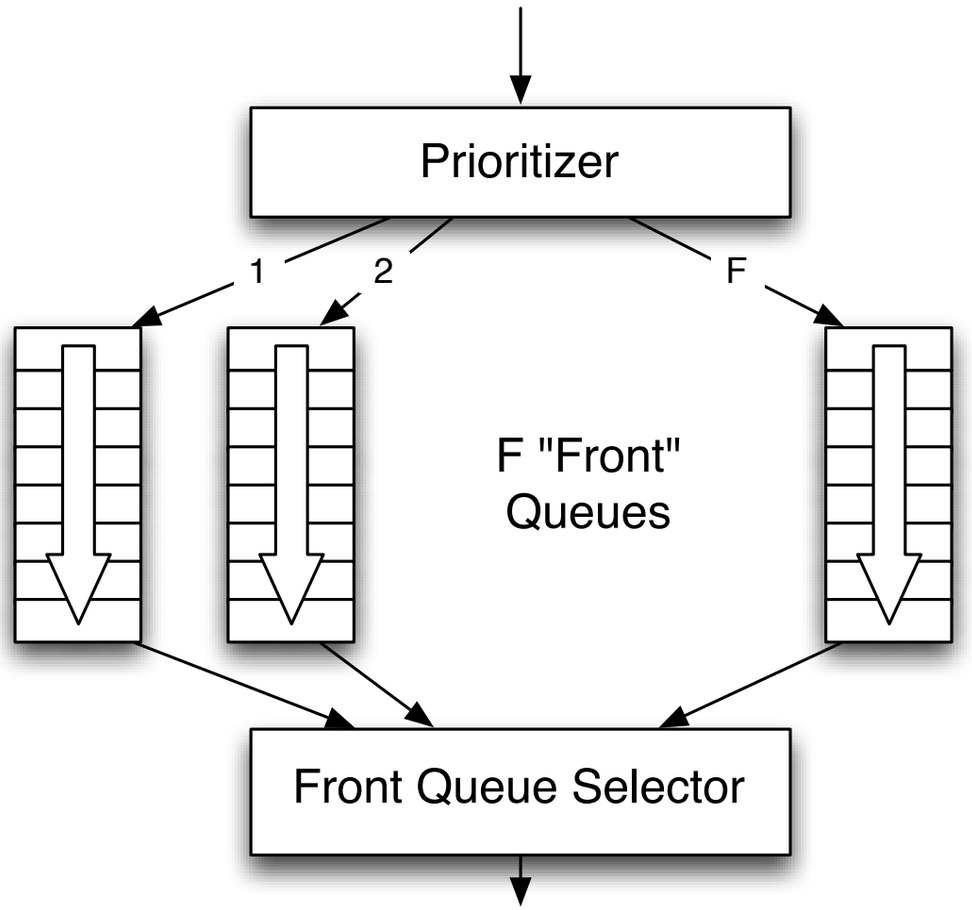


- URLs flow from top to bottom
- Front queues manage priority
- Back queue manage politeness
- Each queue is FIFO

http://research.microsoft.com/~najork/mercator.pdf

URL Frontier Implementation - Mercator

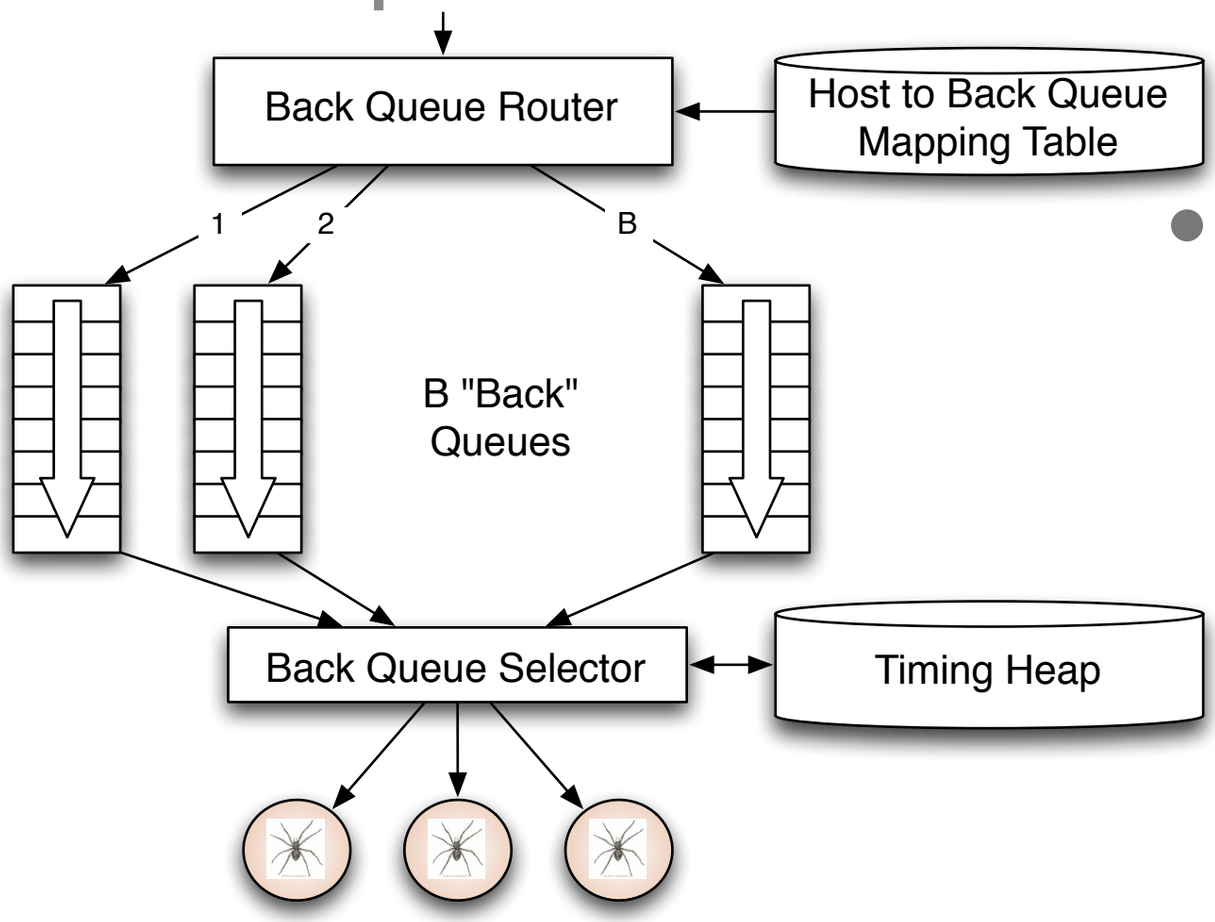
Front queues



- Prioritizer takes URLS and assigns a priority
  - Integer between 1 and F
  - Appends URL to appropriate queue
- Priority
  - Based on rate of change
  - Based on quality (spam)
  - Based on application

URL Frontier Implementation - Mercator

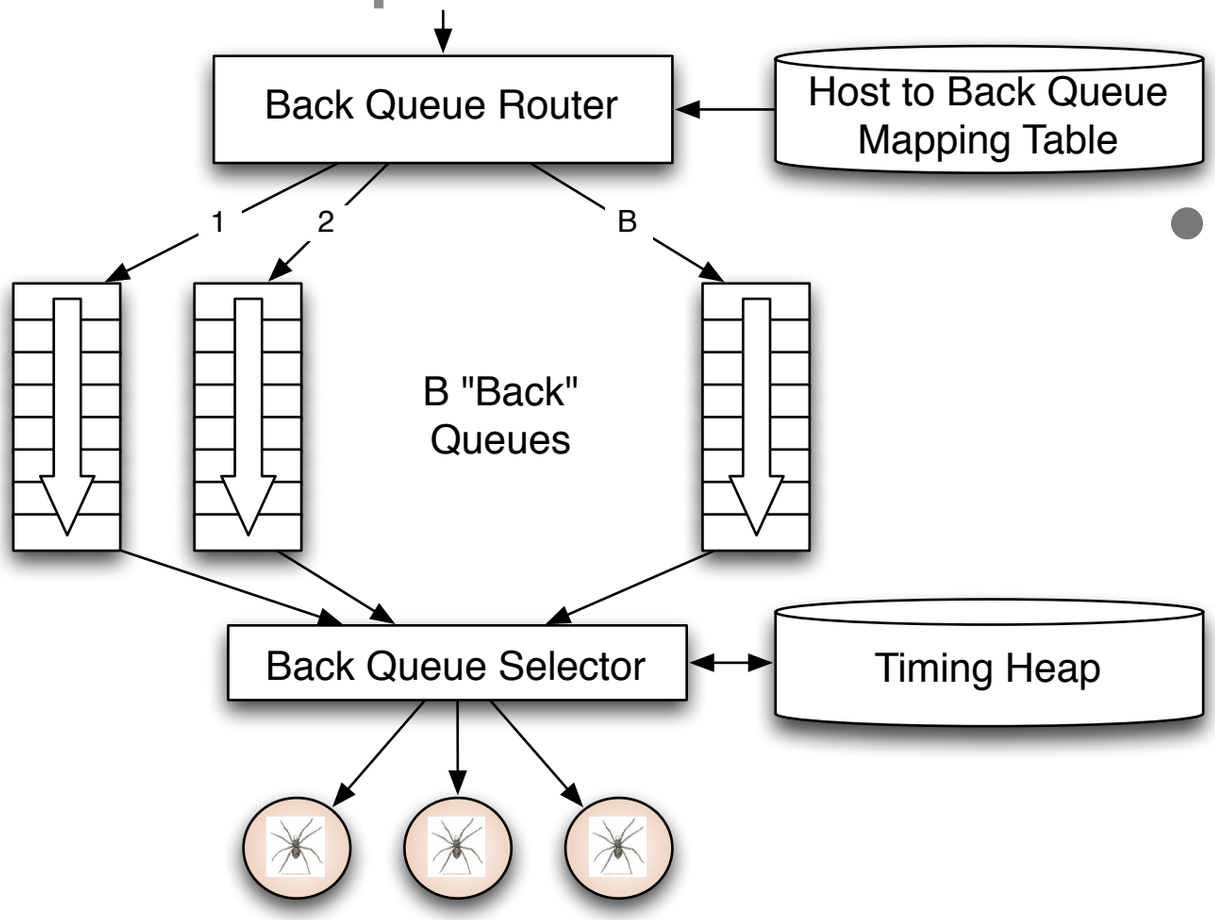
Back queues



- Selection from front queues is initiated from back queues
- Pick a front queue, how?
  - Round robin
  - Randomly
  - Monte Carlo
  - Biased toward high priority

URL Frontier Implementation - Mercator

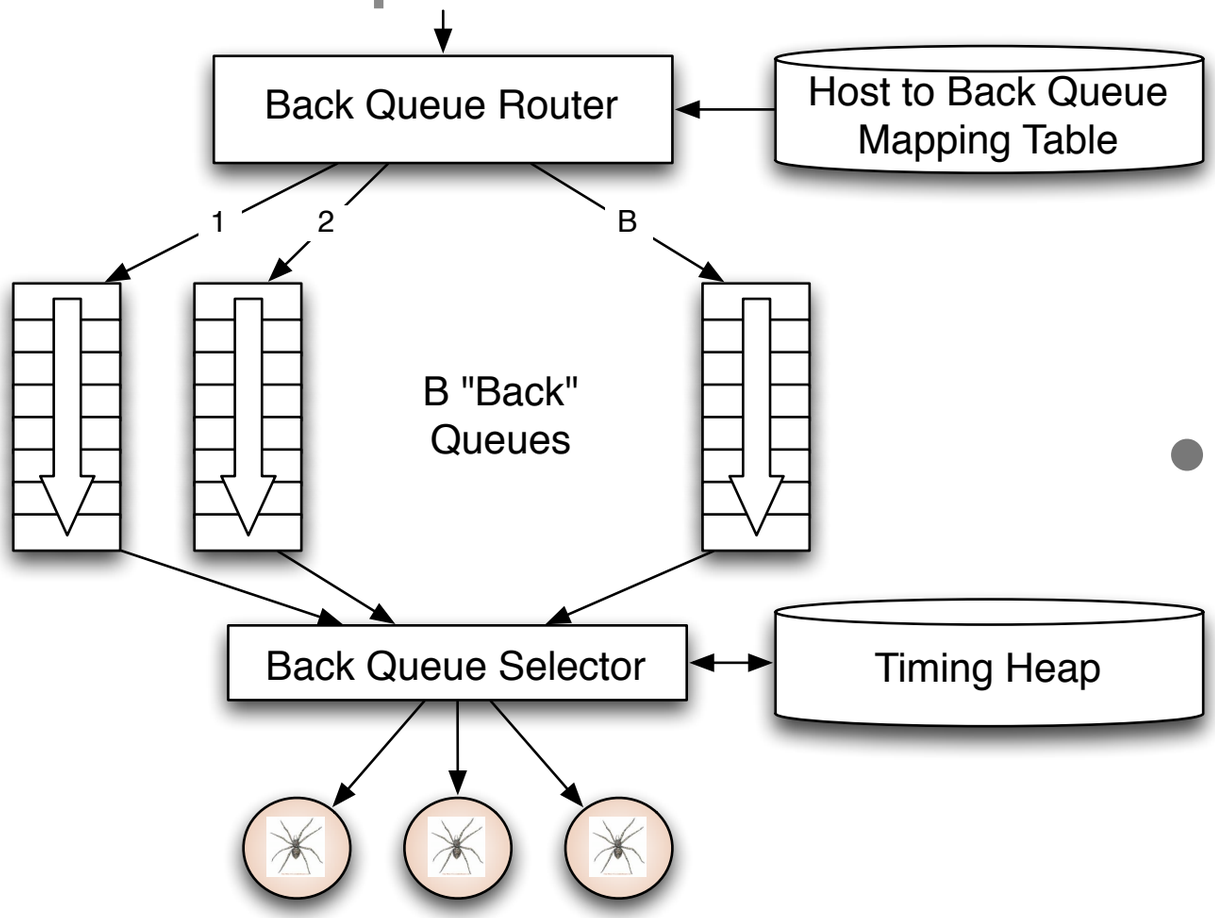
Back queues



- Each back queue is non-empty while crawling
- Each back queue has URLs from one host only
  - Maintain a table of URL to back queues (mapping) to help

URL Frontier Implementation - Mercator

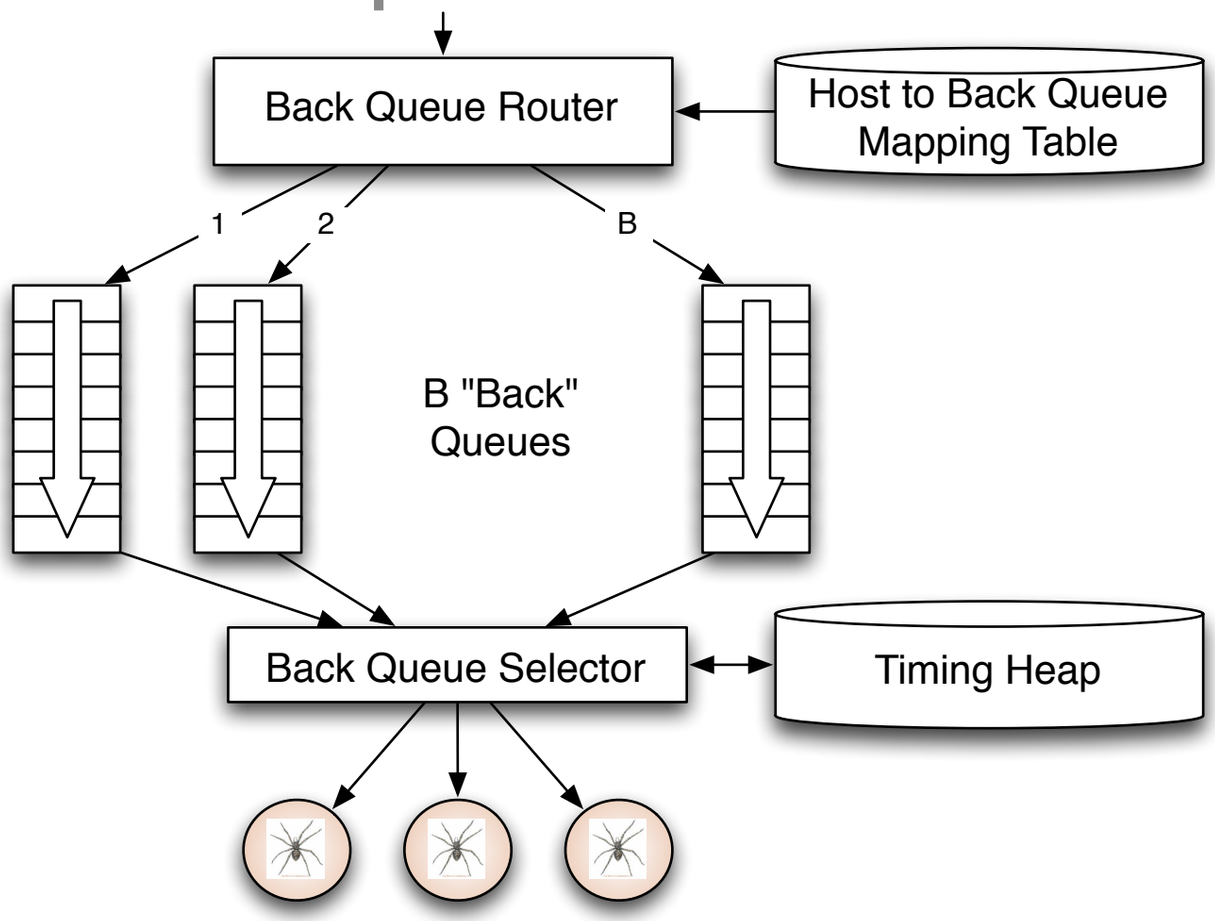
Back queues



- Timing Heap
  - One entry per queue
  - Has earliest time that a host can be hit again
- Earliest time based on
  - Last access to that host
- Plus any appropriate heuristic

URL Frontier Implementation - Mercator

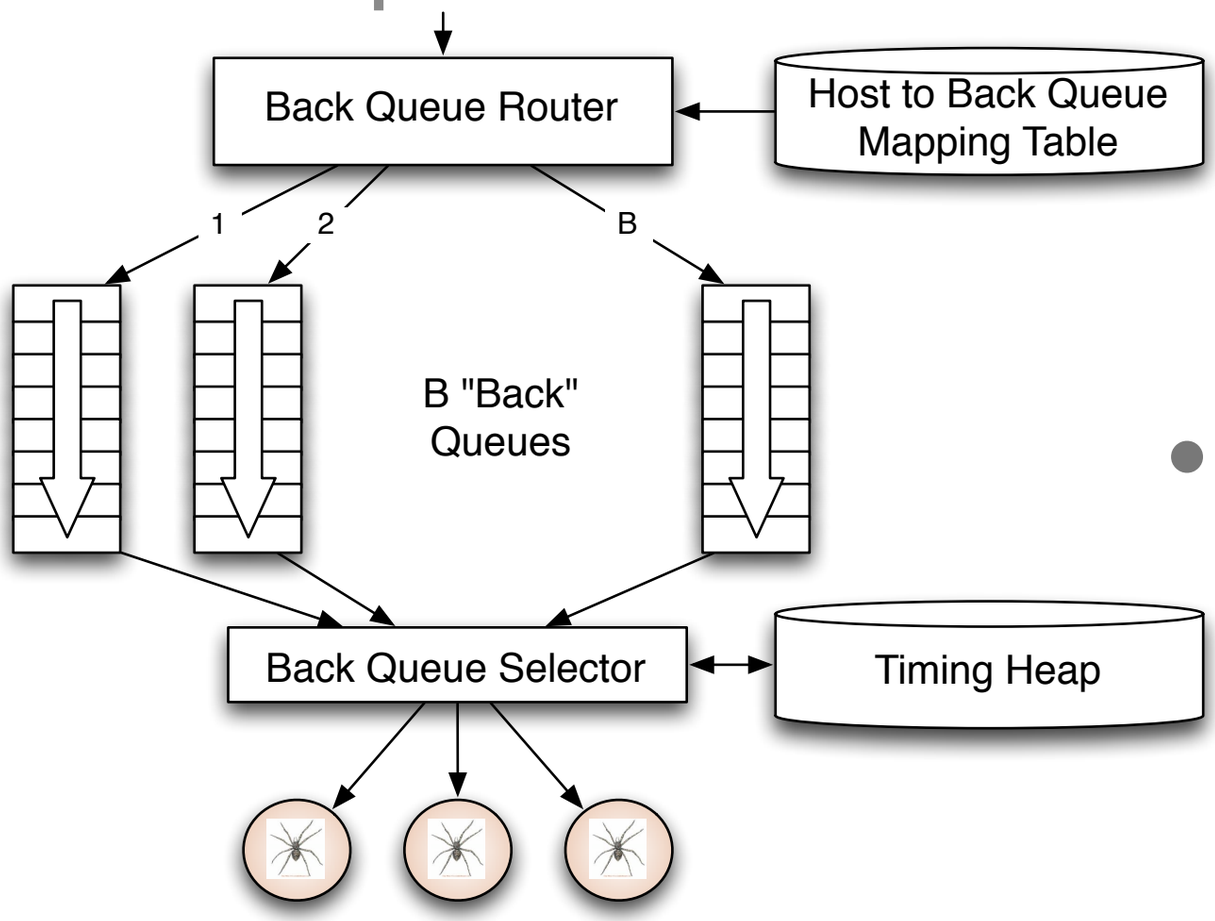
Back queues



- A crawler thread needs a URL
  - It gets the timing heap root
  - It gets the next eligible queue based on time, b.
  - It gets a URL from b
  - If b is empty
  - Pull a URL v from front queue
  - If back queue for v exists place it in that queue, repeat.
  - Else add v to b - update heap.

URL Frontier Implementation - Mercator

Back queues



- How many queues?
  - Keep all threads busy
  - ~3 times as many back queues as crawler threads
- Web-scale issues
  - This won't fit in memory
- Solution
  - Keep queues on disk and keep a portion in memory.

Web Crawling Outline

Overview

- Introduction
- URL Frontier
- Robust Crawling
  - DNS
  - Various parts of architecture
  - URL Frontier
- Index
  - Distributed Indices
  - Connectivity Servers

The index

- Why does the crawling architecture exists?
  - To gather information from web pages (aka documents).
- What information are we collecting?
  - Keywords
    - Mapping documents to a “bags of words” (aka vector space model)
  - Links
    - Where does a document link to?
    - Who links to a document?

The index has a list of vector space models

1 1998
1 Every
1 Her
1 I
1 I'm
1 Jensen's
2 Julie
1 Letter
1 Most
1 all
1 allegedly
1 back
1 before
1 brings
2 brothers
1 could
1 days
1 dead
1 death
1 everything
1 for
1 from
1 full
1 happens
1 haunts
1 have
1 hear
3 her
1 husband
1 if
1 it
1 killing
1 letter
1 nothing
1 now
1 of
1 pray
1 read,
1 saved
1 sister
1 stands
1 story
1 the
2 they
1 time
1 trial
1 wonder
1 wrong
1 wrote

1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1

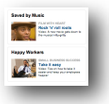
Indices

Our index is a 2-D array or Matrix

A Row For Each Web Page (or "Document")

A Column for Each Word (or "Term")

1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1



1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 2

⋮

0 0 0 1 1 4 1 1 1 1 0 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 0 1 1 1 2

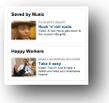
Indices

"Term-Document Matrix" Capture Keywords

A Column for Each Word (or "Term")

A Row For Each Web Page (or "Document")

1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1



1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 2

0 0 0 1 1 4 1 1 1 1 0 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 0 0 1 1 1 2

The Term-Document Matrix

- Is really big at a web scale
- It must be split up into pieces
- An effect way to split it up is to split up the same way as the crawling
  - Equivalent to taking horizontal slices of the T-D Matrix
  - Helps with cache hits during crawl
- Later we will see that it needs to be rejoined for calculations across all documents

Connectivity Server

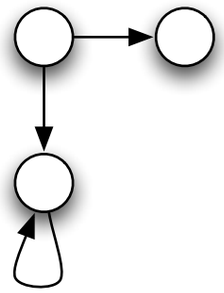
- Other part of reason for crawling
- Supports fast queries on the web graph
  - Which URLS point to a given URL (in-links)?
  - Which URLS does a given URL point to (out-links)?
- Applications
  - Crawl control
  - Web Graph Analysis (see Assignment #03)
  - Link Analysis (aka PageRank)
    - Provides input to “quality” for URL frontier

Adjacency Matrix - Conceptual Idea

A

B

C



	A	B	C
A	0	1	1
B	0	0	0
C	0	0	1

Connectivity Server in practice

- What about Adjacency Lists instead?
  - Set of neighbors of a node
  - Assume each URL represented by an integer
  - i.e. 4 billion web pages need 32 bits per URL
  - Naive implementation requires 64 bits per link
    - 32 bits to 32 bits

Connectivity Server in practice

- What about Adjacency Lists instead?
  - Non-naive approach is to exploit compression
    - Similarity between lists of links
    - Locality (many links go to “nearby” links)
    - Use gap encodings in sorted lists
    - Leverage the distribution of gap values

Connectivity Server in practice

- Current state of the art is Boldi and Vigna
  - http://www2004.org/proceedings/docs/1p595.pdf
  - They are able to reduce a URL to URL edge
    - From 64 bits to an average of 3 bits
    - For a 118 million node web graph
- How?

Connectivity Server in practice

- Consider a lexicographically ordered list of all URLS, e.g:
  - http://www.ics.uci.edu/computerscience/index.php
  - http://www.ics.uci.edu/dept/index.php
  - http://www.ics.uci.edu/index.php
  - http://www.ics.uci.edu/informatics/index.php
  - http://www.ics.uci.edu/statistics/index.php

Connectivity Server in practice

- Each of these URLs has an adjacency list
- Main idea: because of templates, the adjacency list of a node is similar to one of the 7 preceding URLs in the lexicographic ordering.
- So, express adjacency list in terms of a template

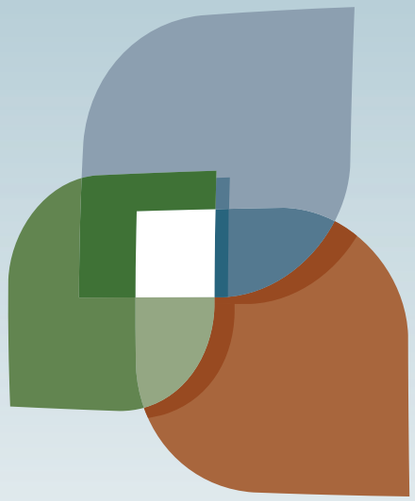
Connectivity Server in practice

- Consider these adjacency lists
  - 1, 2, 4, 8, 16, 32, 64
  - 1, 4, 9, 16, 25, 36, 49, 64
  - 1, 2, 3, 5, 6, 13, 21, 34, 55, 89, 144
  - 1, 4, 8, 16, 25, 36, 49, 64
    - Encode this as row(-2), -URL(9), +URL(8)
- Very similar to tricks done in assembly code

Connectivity Server in practice summary

- The web is enormous
- A naive adjacency matrix would be several billion URLS on a side
- Overall goal is to keep the adjacency matrix in memory
- Webgraph is a set of algorithms and a java implementation for examining the web graph
- It exploits the power law distribution to compress the adjacency matrix very tightly
- http://webgraph.dsi.unimi.it/

End of Chapter 20



LUCI