

Web Crawling

Introduction to Information Retrieval

INF 141/ CS 121

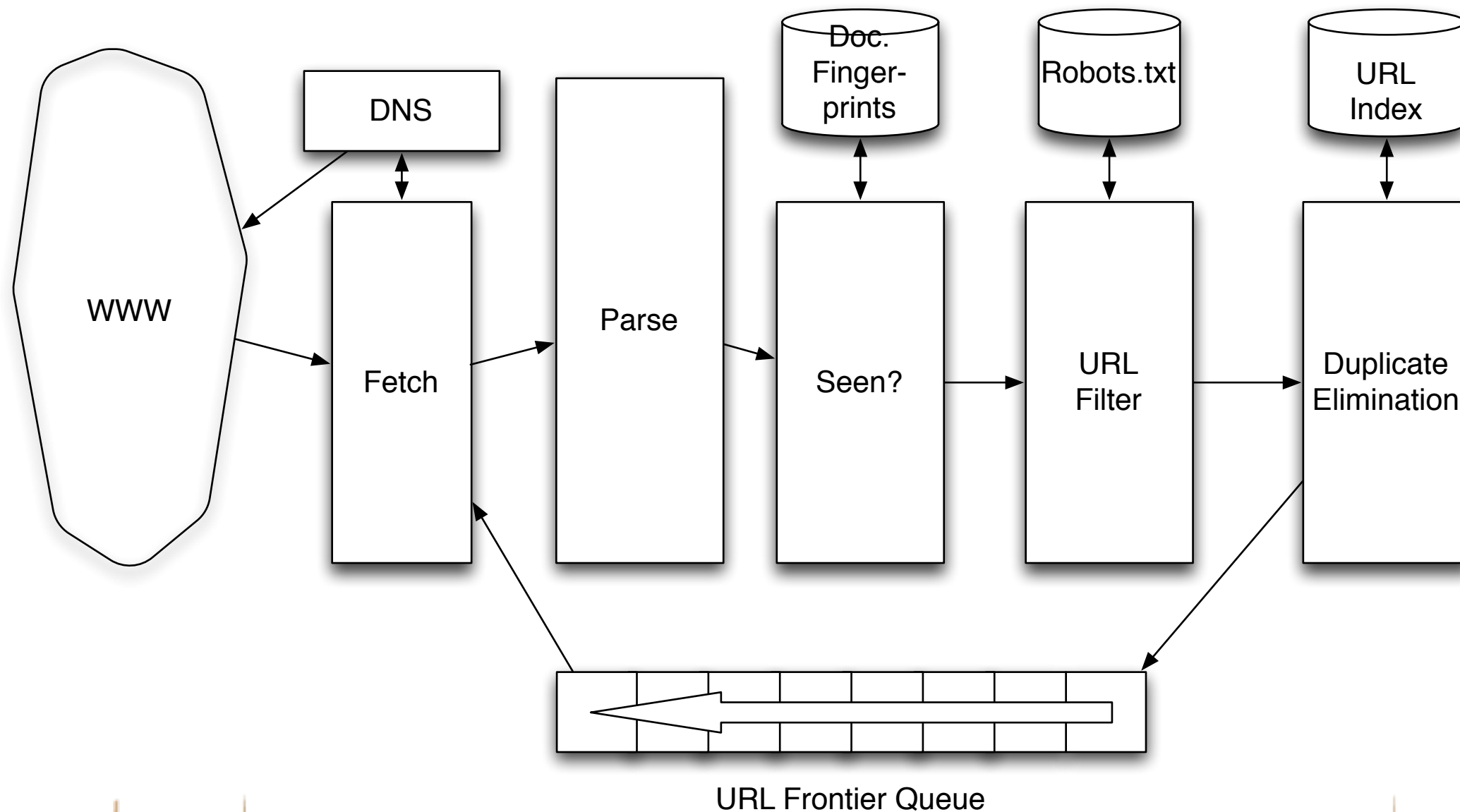
Donald J. Patterson

Content adapted from Hinrich Schütze

<http://www.informationretrieval.org>



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



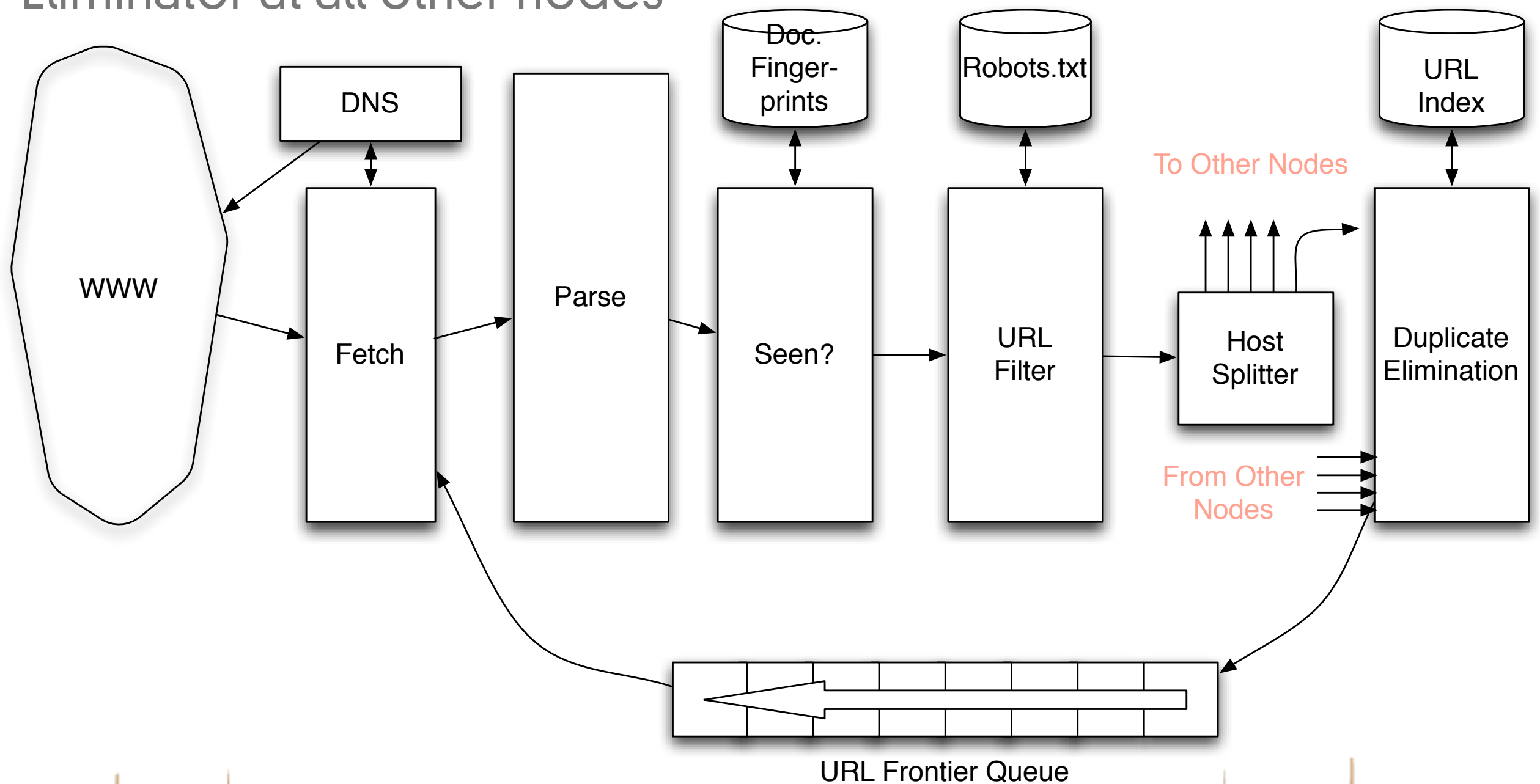
Distributing the crawl

- 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 each other
 - 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



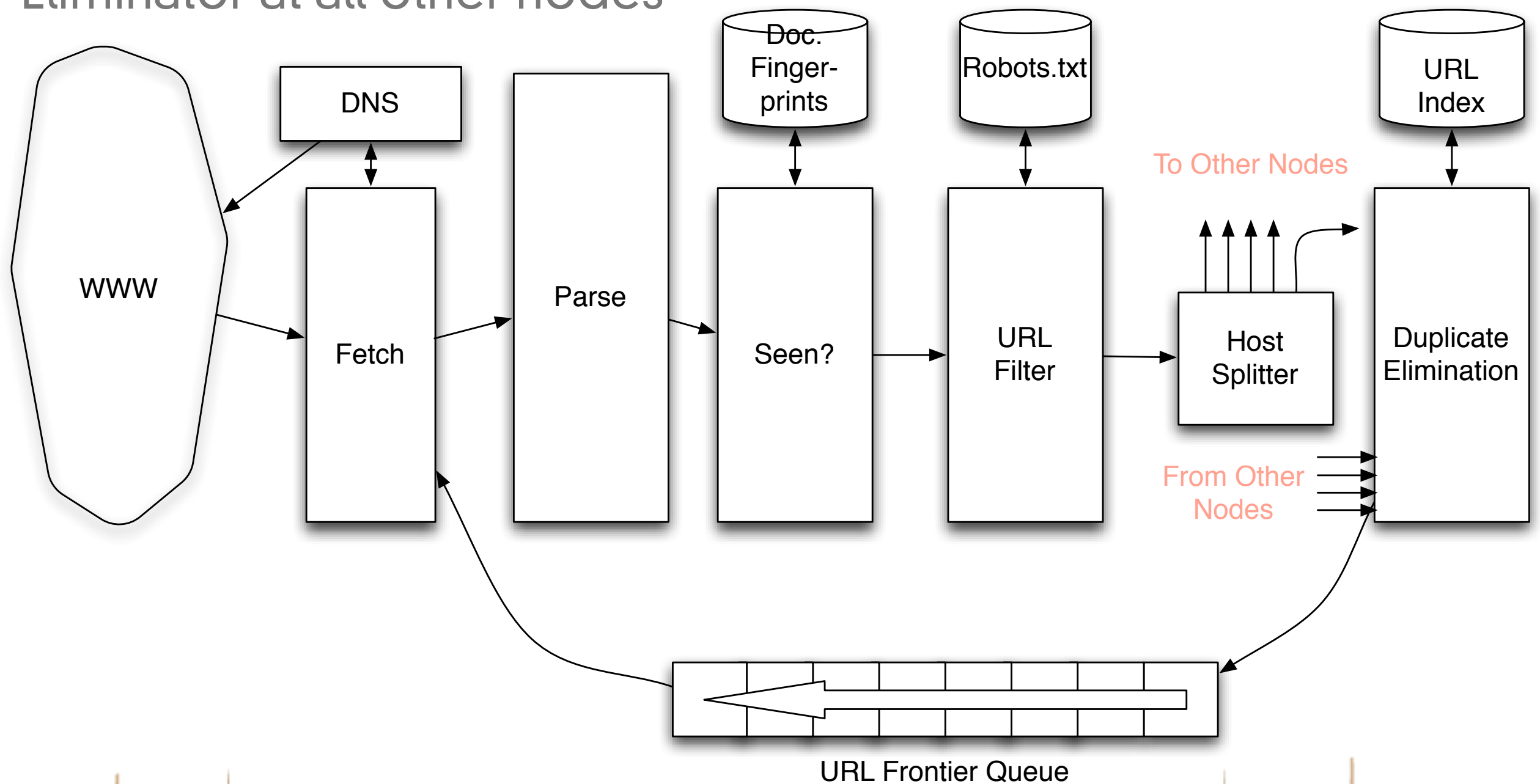
Overview

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



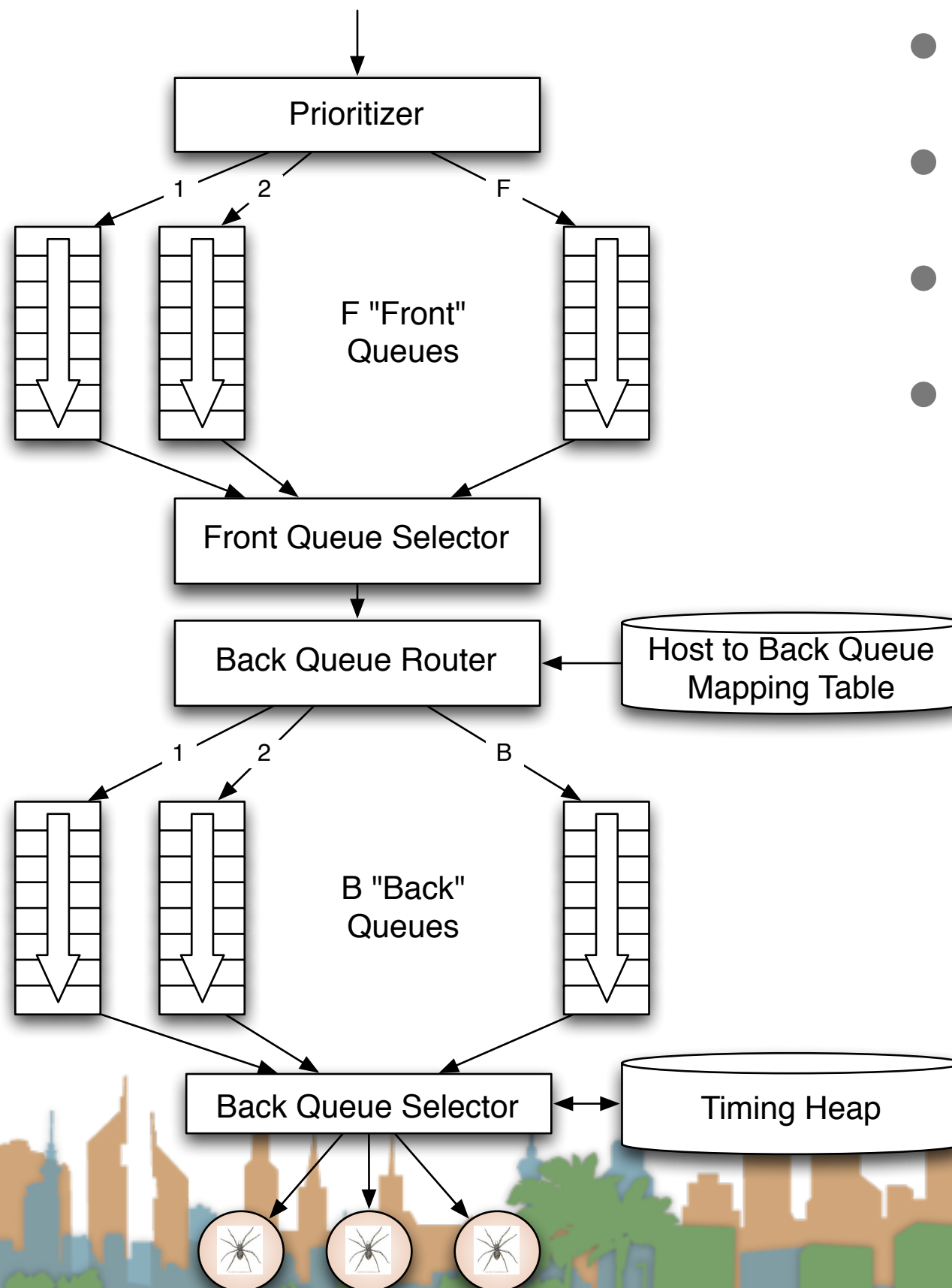
Robust Crawling

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

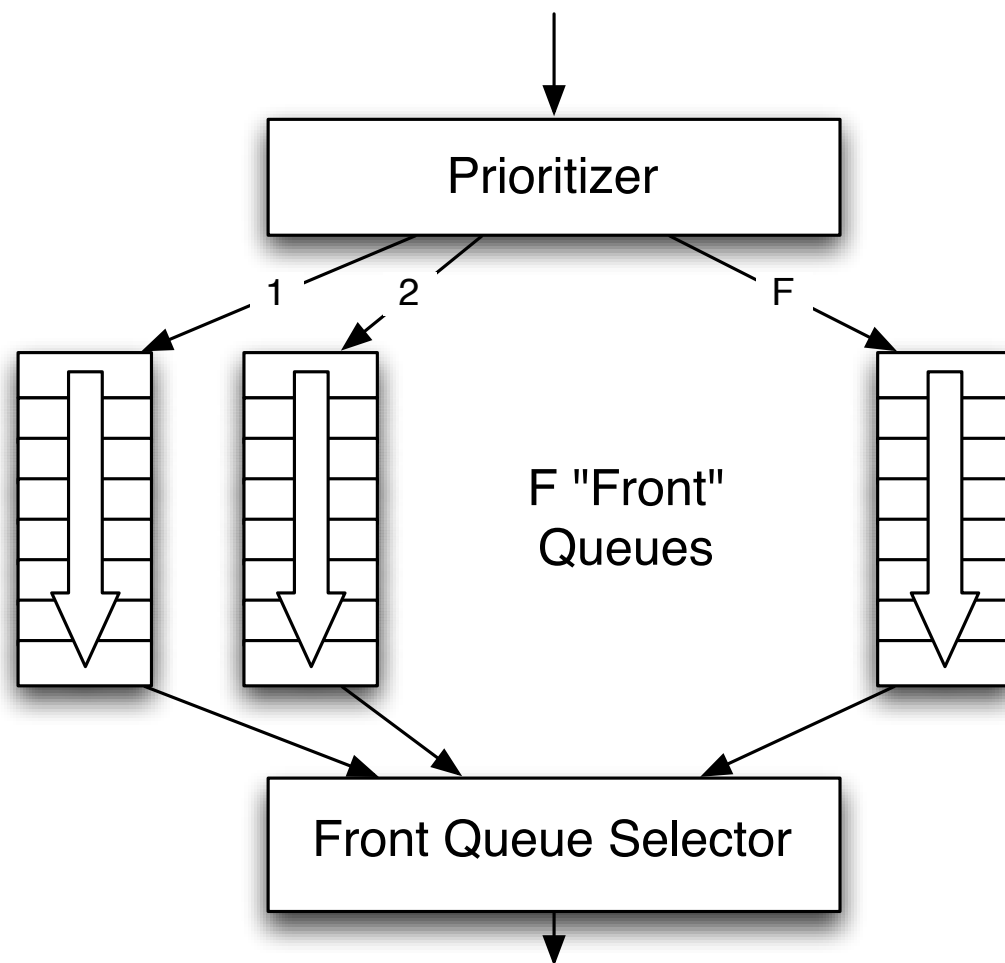


URL Frontier Implementation - Mercator

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

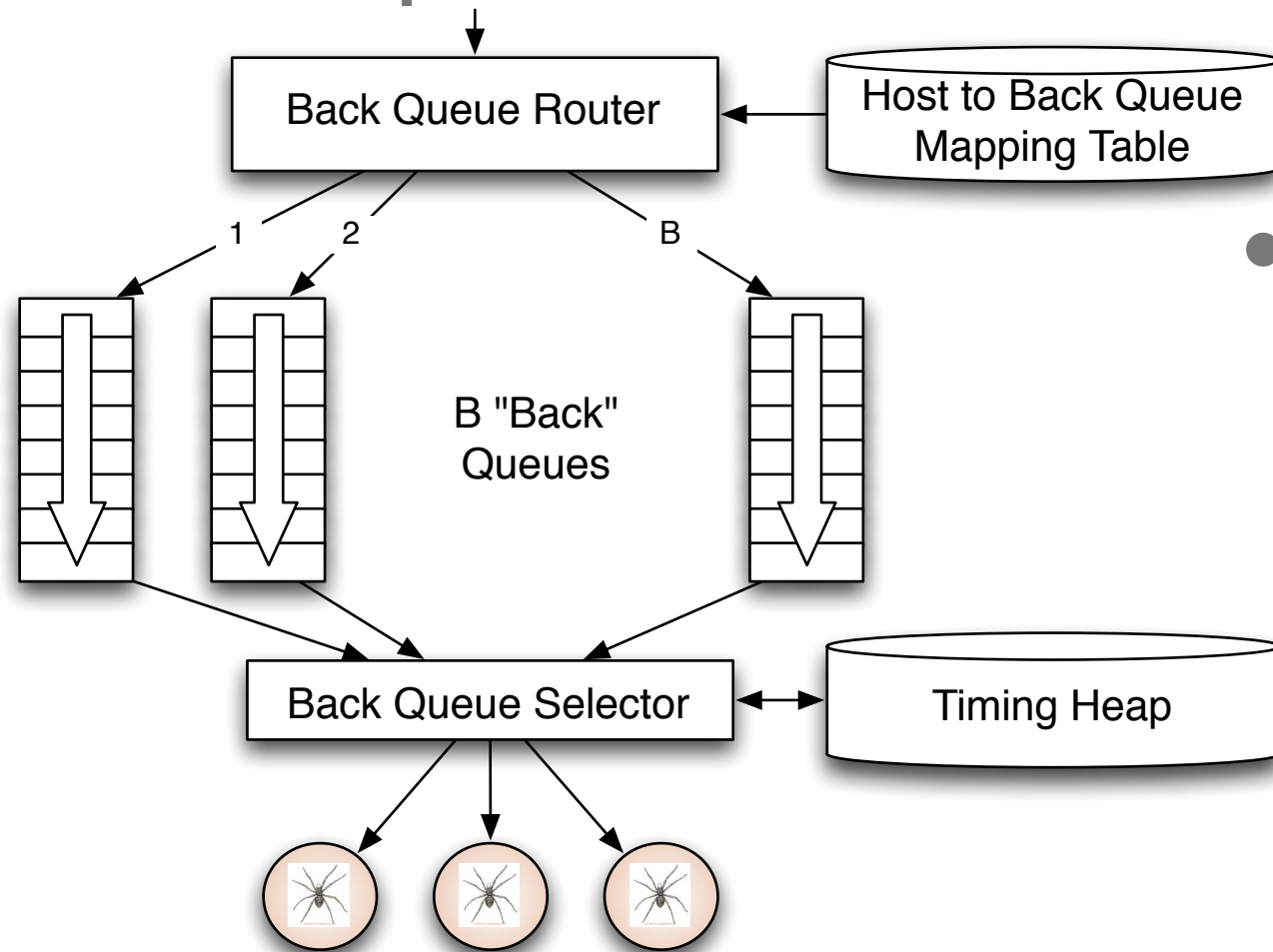


Front queues



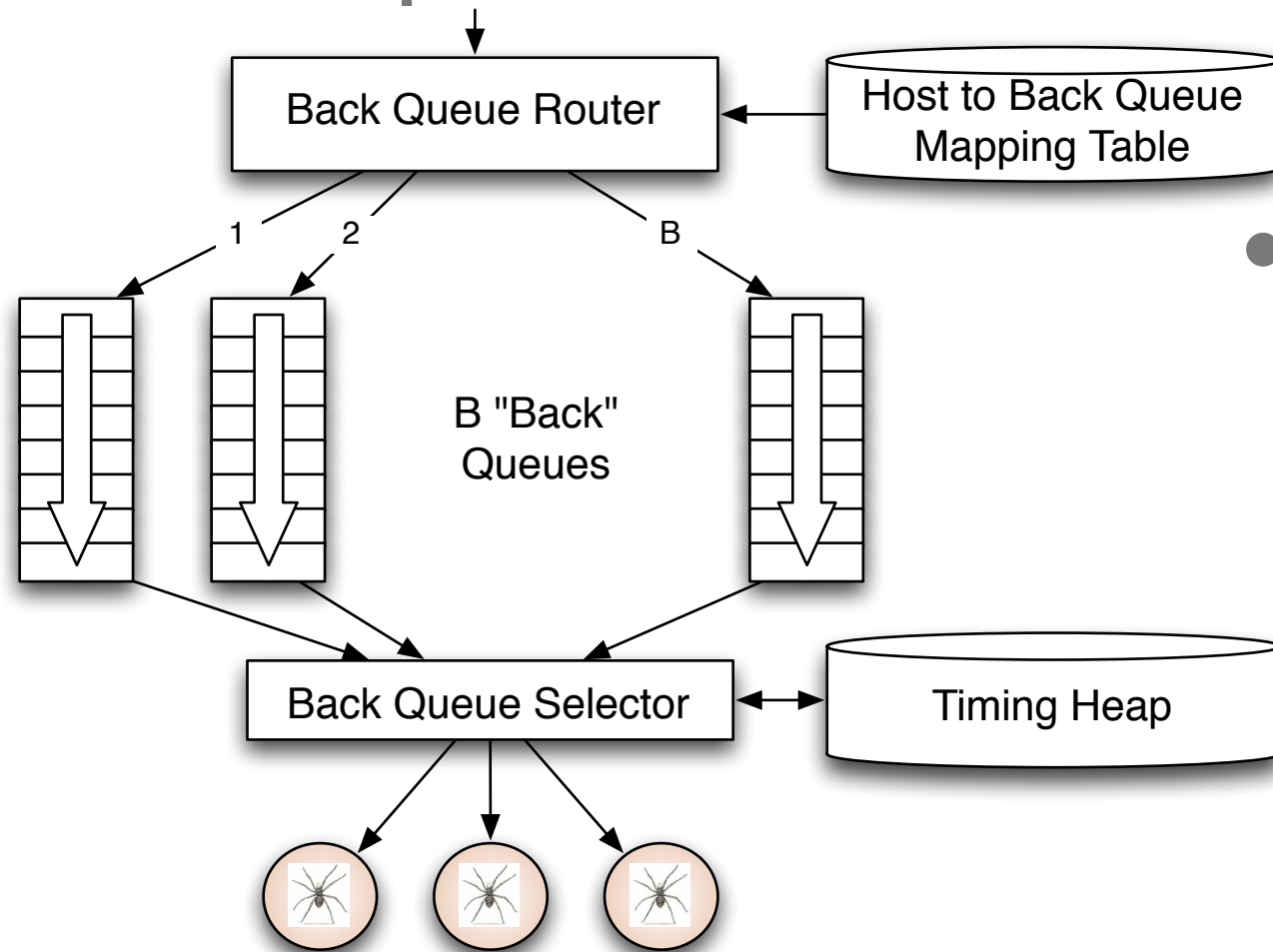
- Prioritizer takes URLs and assigns a priority
- Integer between 1 and F
- Appends URL to appropriate queue
- Priority
 - Based on rate of change
 - Based on quality (spam)
 - Based on application

Back queues



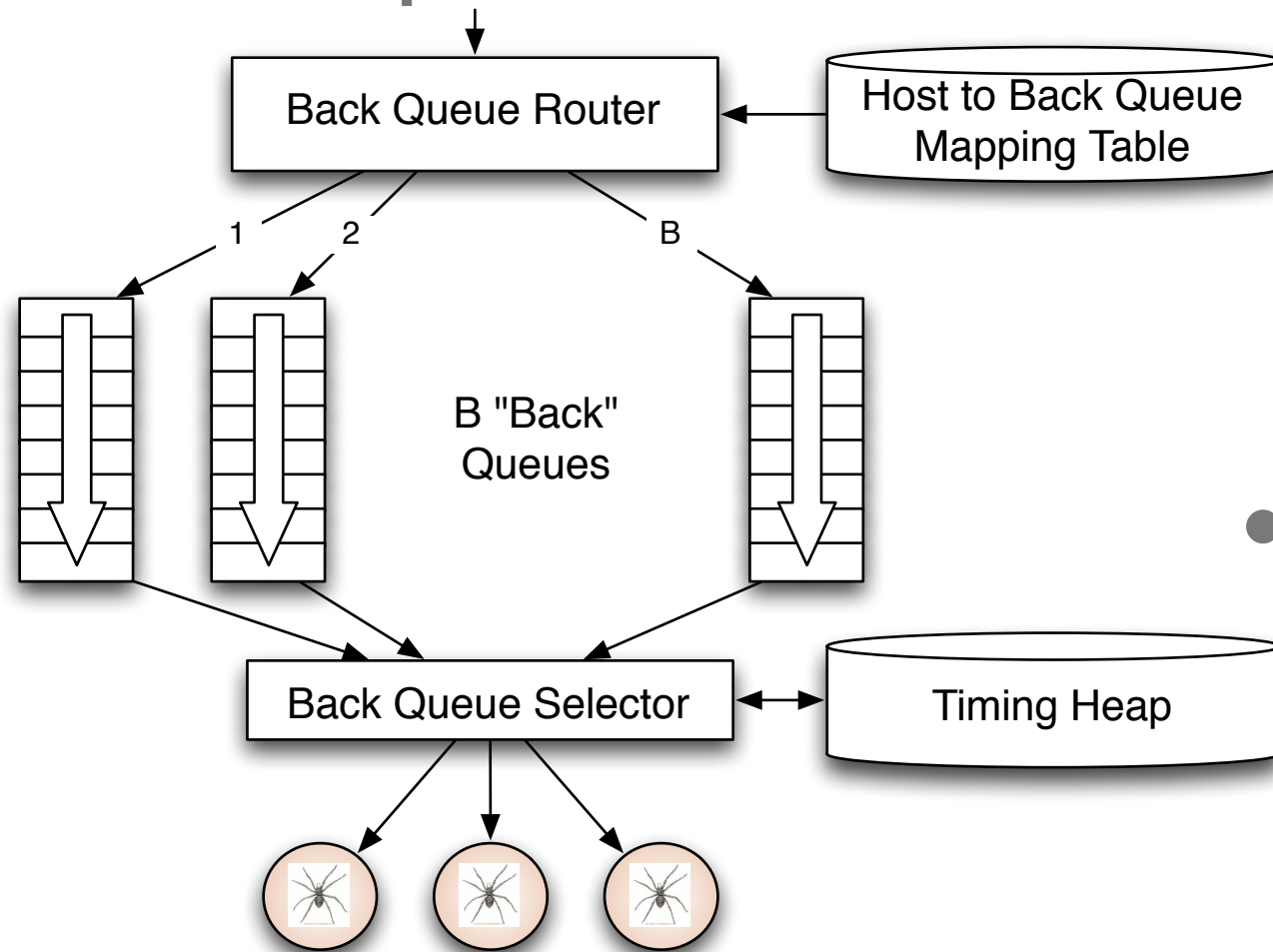
- Selection from front queues is initiated from back queues
- Pick a front queue, how?
 - Round robin
 - Randomly
 - Monte Carlo
 - Biased toward high priority

Back queues



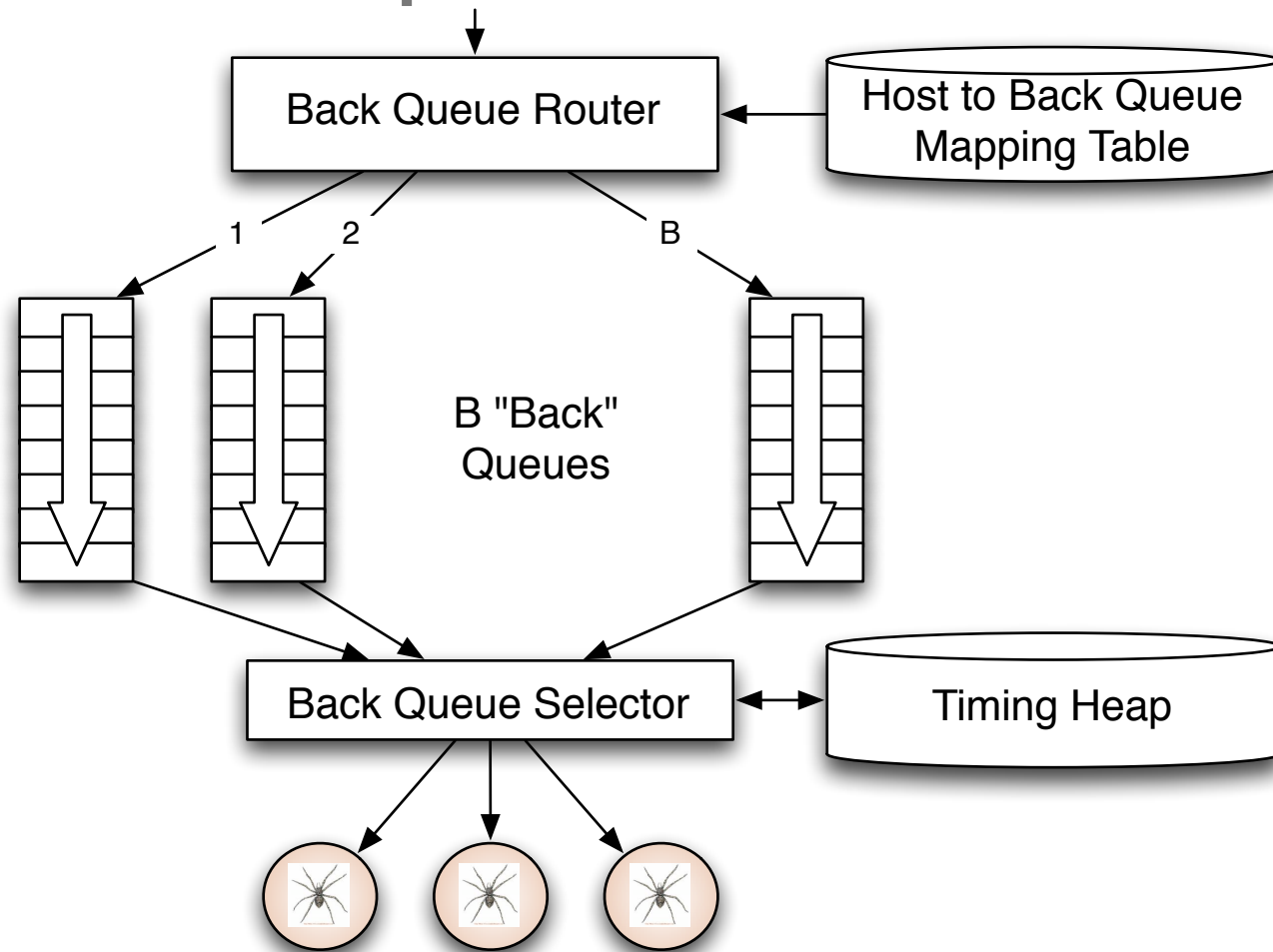
- Each back queue is non-empty while crawling
- Each back queue has URLs from **one host only**
- Maintain a table of URL to back queues (mapping) to help

Back queues



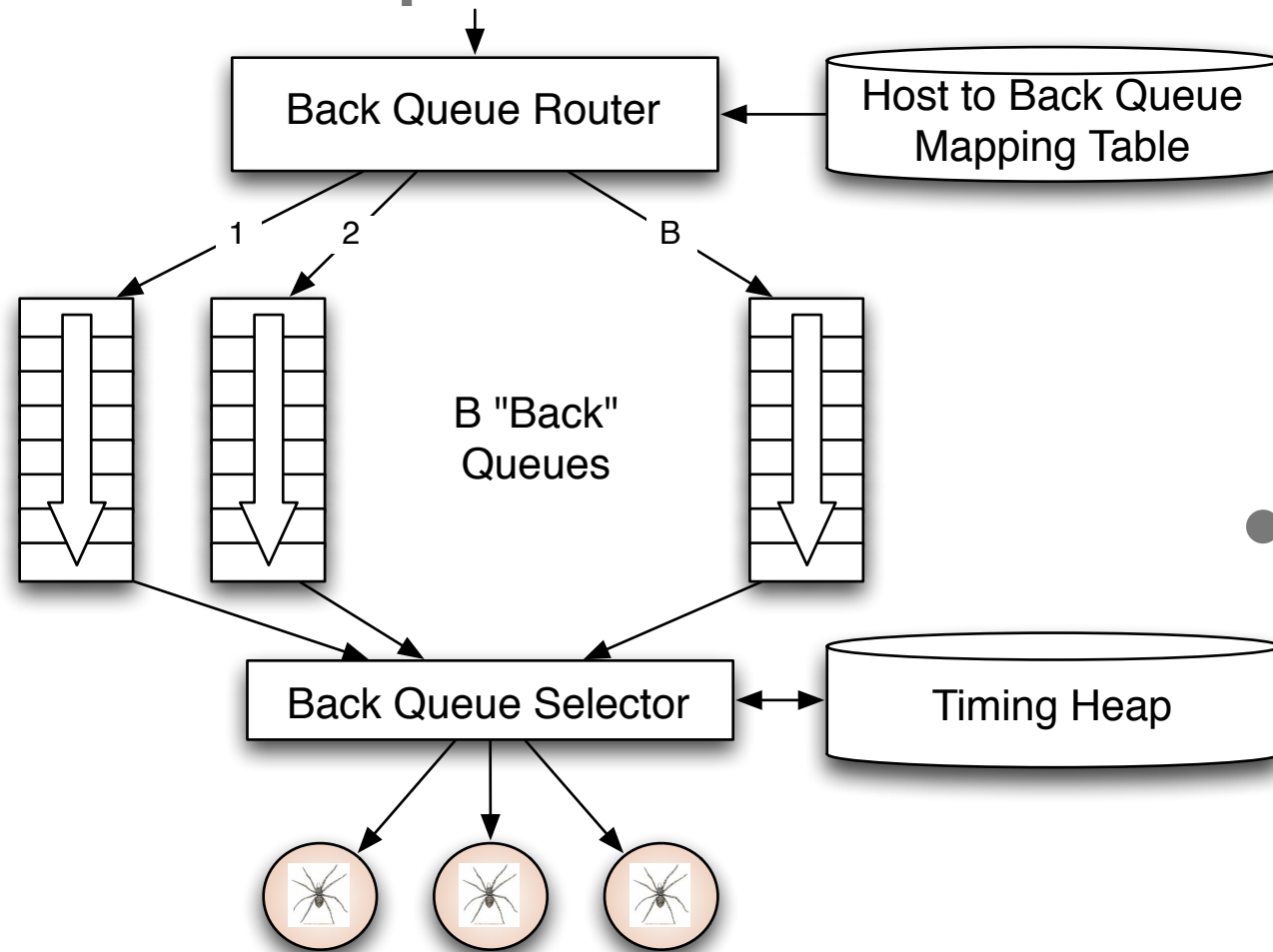
- Timing Heap
- One entry per queue
- Has earliest time that a host can be hit again
- Earliest time based on
 - Last access to that host
 - Plus any appropriate heuristic
 - robots.txt "crawl-delay"
 - sitemaps instruction

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.

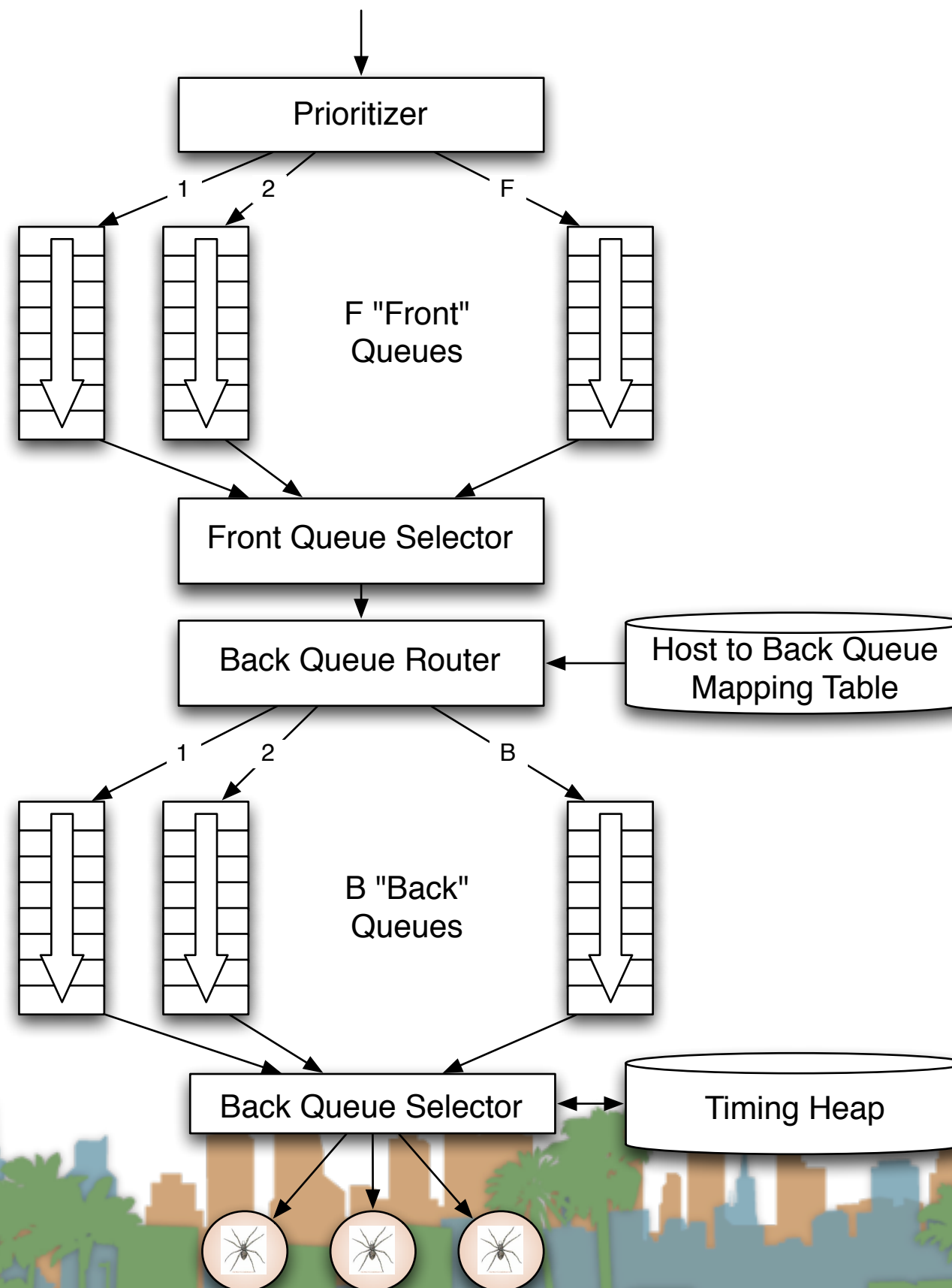
Back queues



- 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.

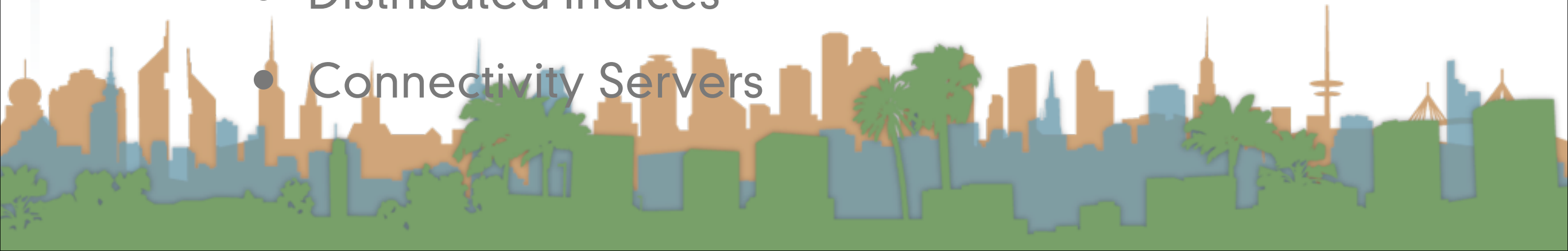


URL Frontier Implementation - Mercator - walk through the process



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 exist?
- 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 2500	2 justin
1 l	1 lamborghini
1 a	1 miami
1 after	1 mugshots
1 and	1 news
1 arrest	1 other
1 at	2 photos
1 beach	1 pills
1 beer	1 police
6 bieber	1 pot
1 bond	1 racing
1 breaking	1 report
1 celeb	1 say.
1 charges	1 see
1 cnn	1 set
1 did	1 singer
1 do	1 story
1 drag	1 the
1 dui	1 tv
1 f	1 was
1 face	1 watch
1 facing	1 what
1 full	1 yellow
1 having	
1 in	
1 judge	

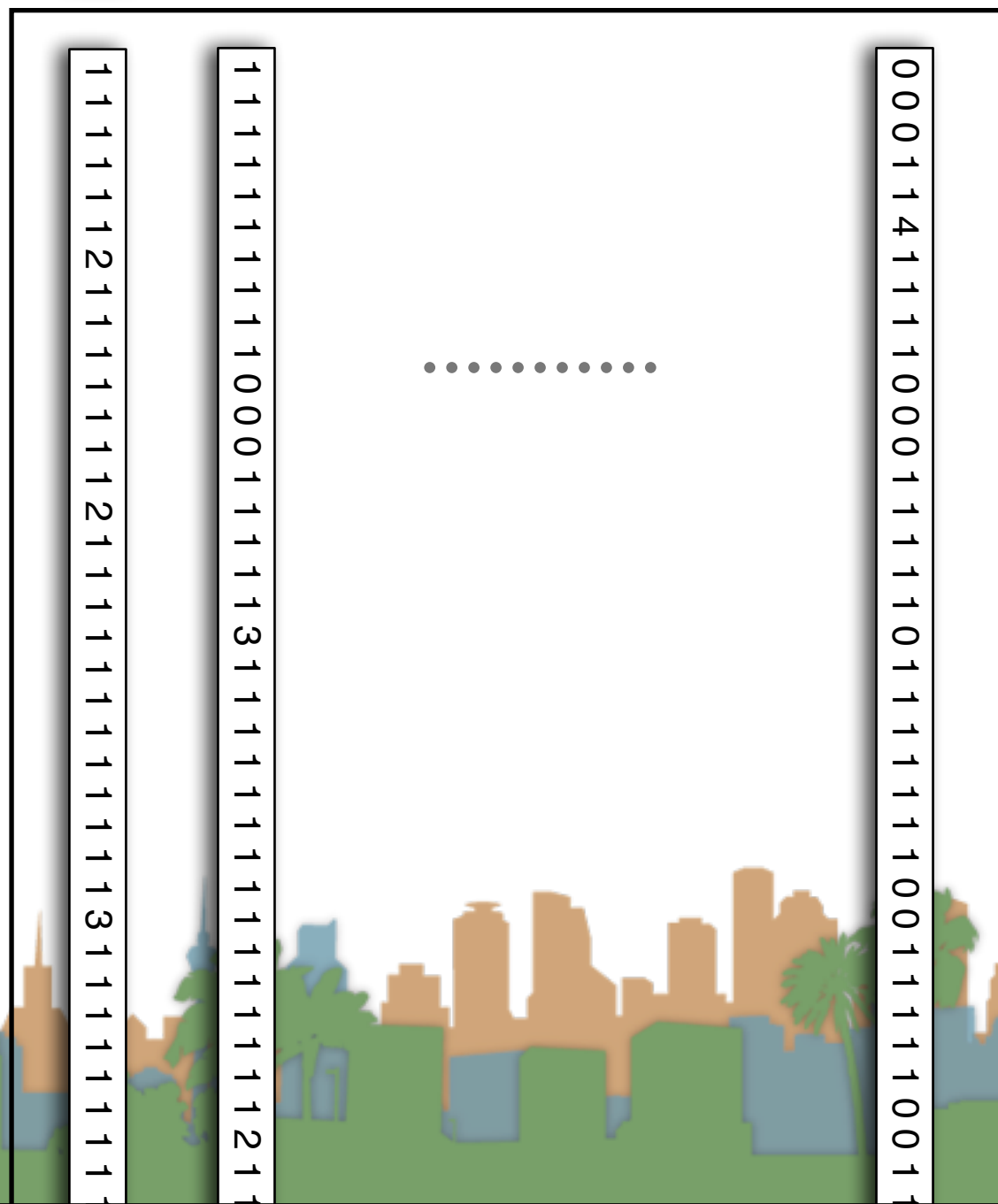
1 1 1 1 1 1 1 1 1 6 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Our index is a 2-D array or Matrix

A Column for Each Web Page (or “Document”)



A Row For Each Word (or "Term")



"Term-Document Matrix" Capture Keywords

A Column for Each Web Page (or “Document”)



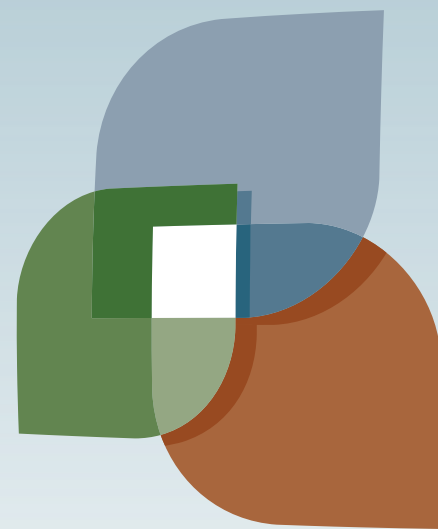
A Row For Each Word (or "Term")

[illegible]

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 vertical 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





L U C I

