Crawling the web

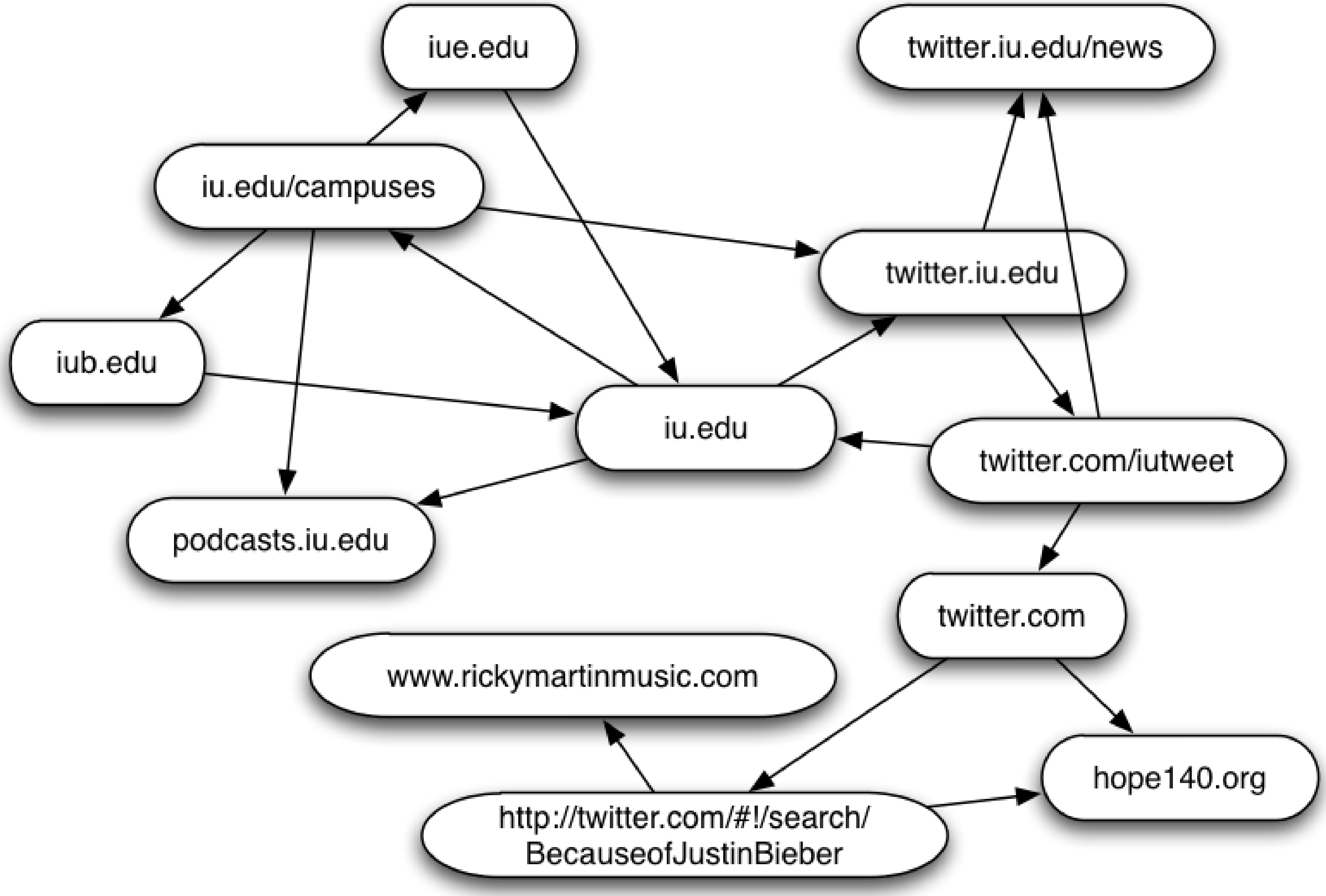
Info 427

Announcements

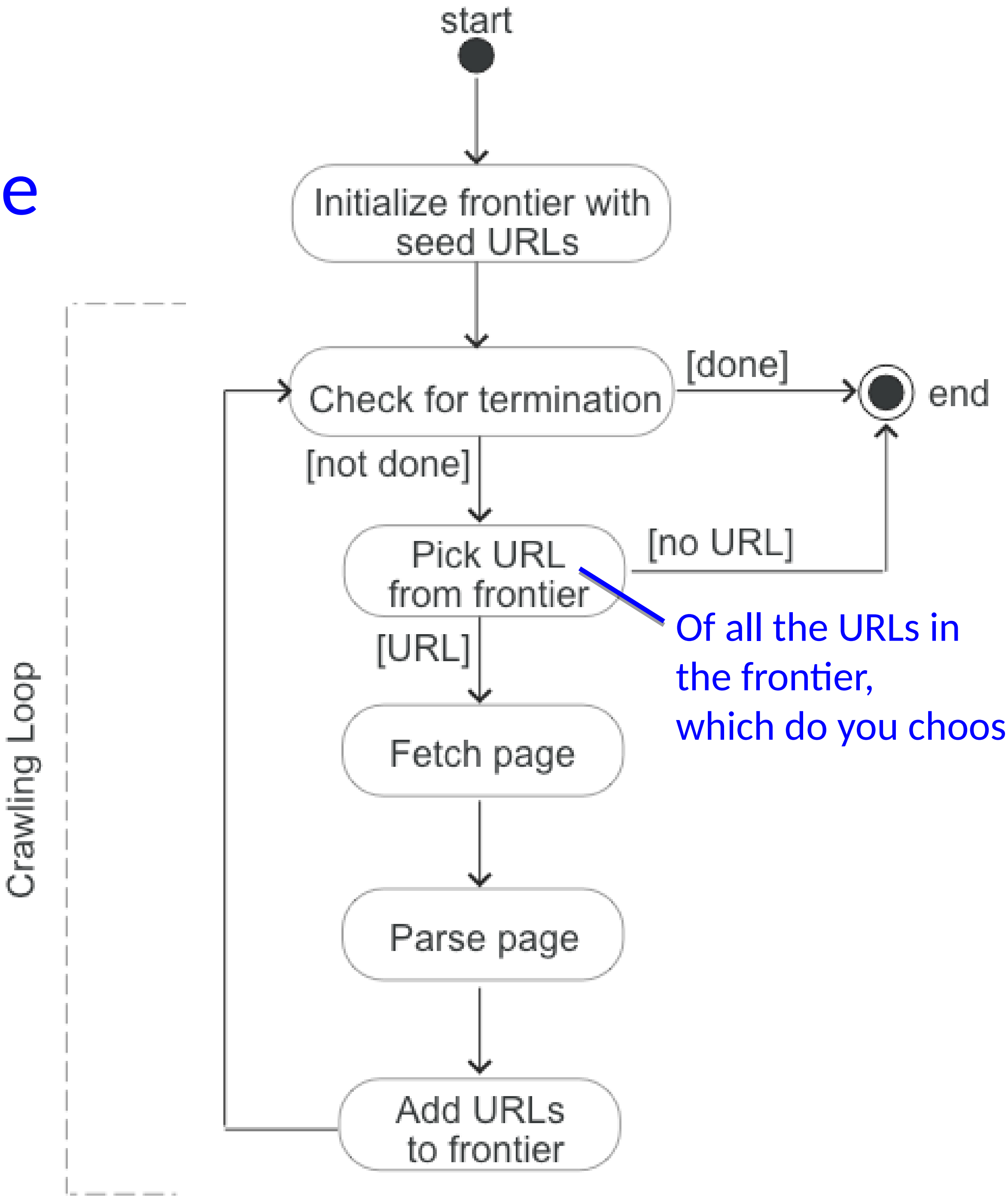
- Assignment 2 due this week!

The web as a graph

- Each vertex of the graph is a webpage
- Edges represent links
  - An edge between A and B means that A links to B
- Crawling the web == traversing this graph
  - Except that we don’t know the structure of the graph ahead of time
- And the graph is changing, even as we traverse it!


iue.edu
twitter.iu.edu/news
iu.edu/campuses
twitter.iu.edu
iub.edu
iu.edu
twitter.com/iutweet
podcasts.iu.edu
twitter.com
www.rickymartinmusic.com
hope140.org
http://twitter.com/#!/search/
BecauseofJustinBieber

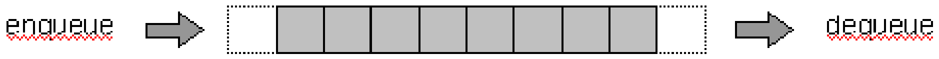
Crawler Architecture



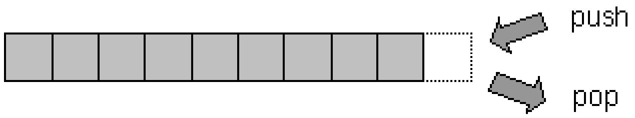
Of all the URLs in the frontier, which do you choose?

Two useful data structures

- Queue (First-in-First-out)
  - Add new elements to end
  - Remove elements from the front



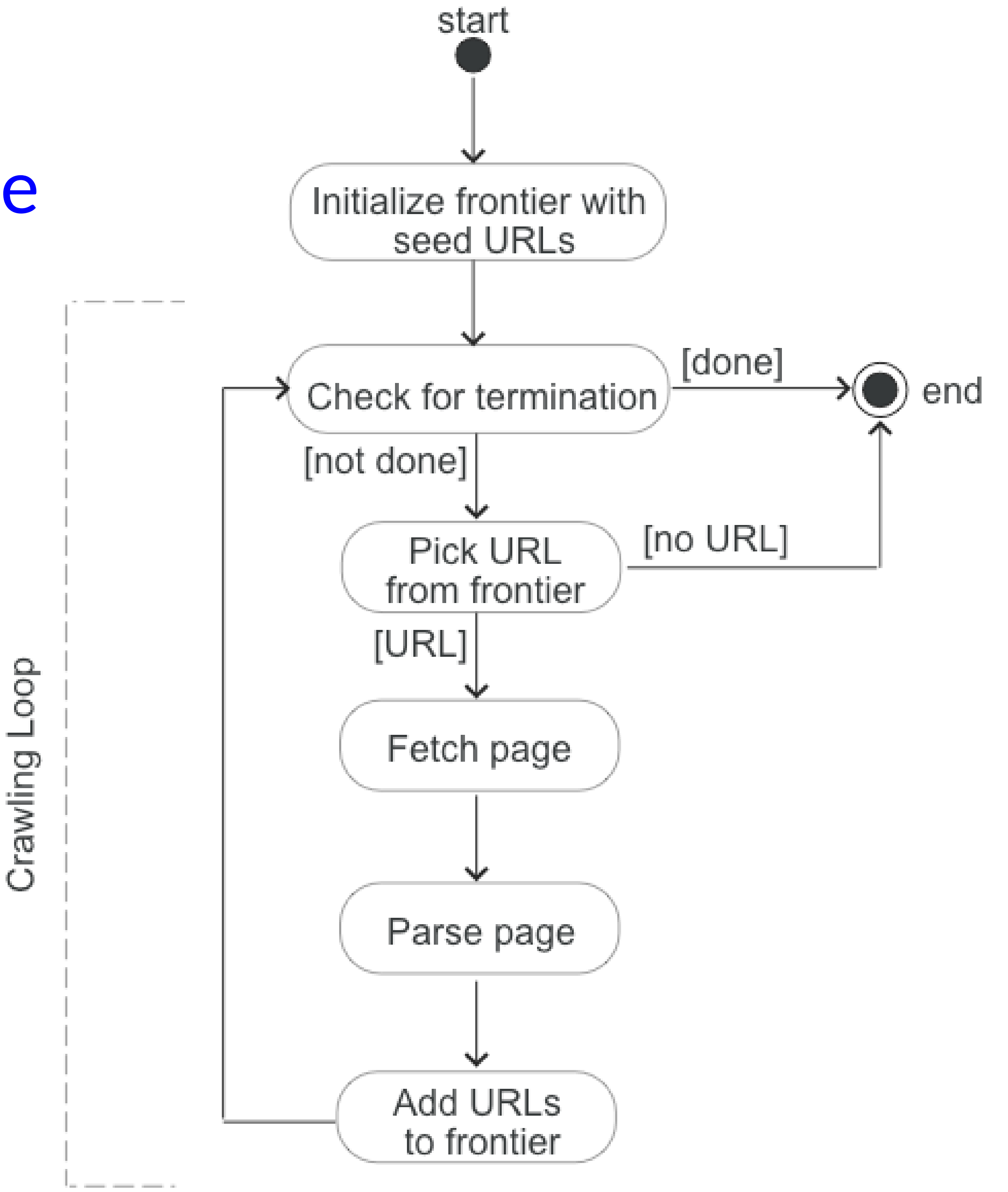
- Stack (Last-in-First-out)
  - Add new elements to the end (or top)
  - Also remove elements from the top



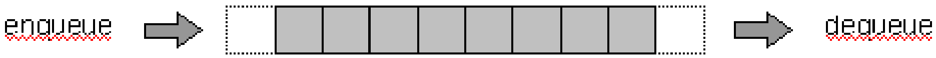
Crawler Architecture

If the frontier is a **queue**, the graph is traversed in **breadth-first search (BFS)** order.

If the frontier is a **stack,** the graph is traversed in **depth-first search (DFS)** order.

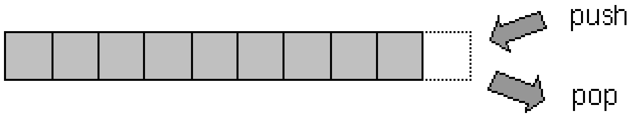


BFS pseudocode:

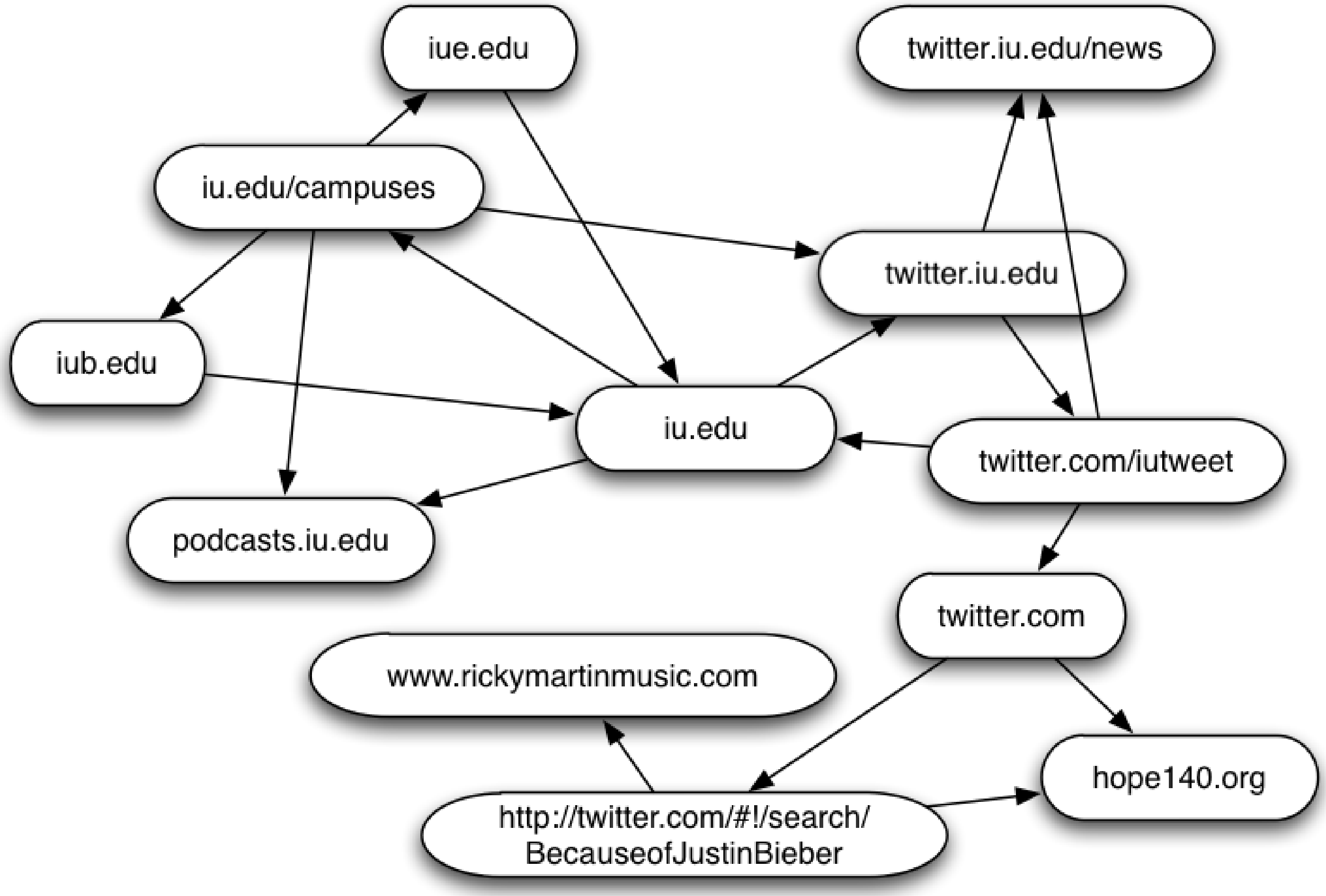


- Queue Q;
- Add seed nodes (URLs) to end of Q;
- While Q is not empty
  - Remove node n from front of Q
  - If n has not been visited, add n’s children to the back of Q

DFS pseudocode:



- Stack S;
- Add seed nodes (URLs) to front of S;
- While S is not empty
  - Remove node n from front of S
  - If n has not been visited, add n’s children to the front of S



BFS pseudocode:

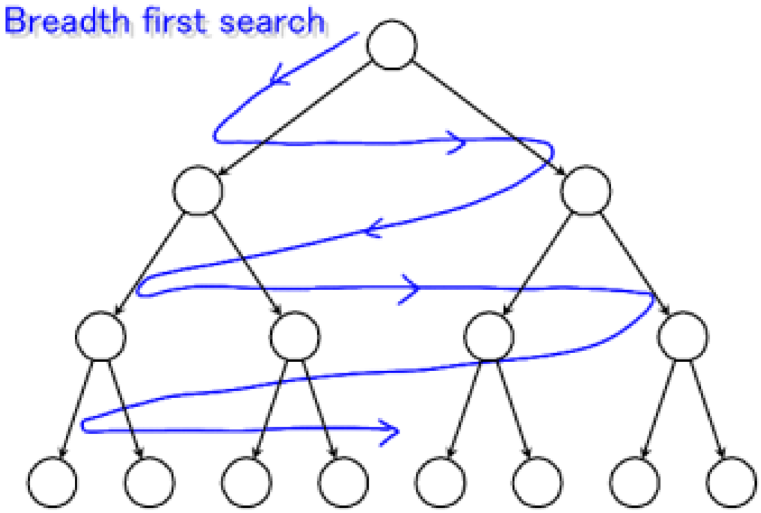
```
• Add seed nodes (URLs) to end of Q;
• While Q is not empty
    – Remove node n from front of Q
    – If n has not been visited, add n's
      children to the back of Q
```

DFS pseudocode:

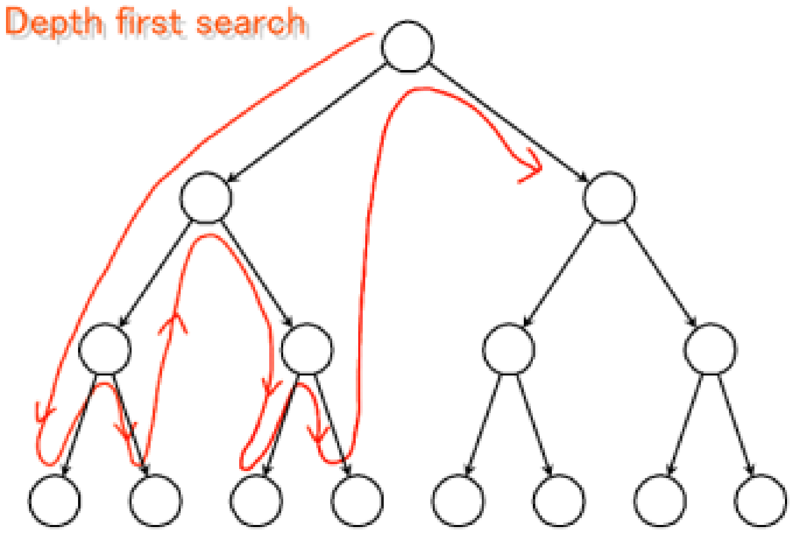
```
• Add seed nodes (URLs) to front of S;
• While S is not empty
    – Remove node n from front of S
    – If n has not been visited, add n's
      children to the front of S
```

Graph traversal

- Breadth First Search
  - Visits all children of the root, then all children of the children, etc.
  - Finds pages along shortest paths from the seed page
  - Implemented with a Queue (First-in-First-out)



- Depth First Search
  - Visits the root's first child, then the first child of that child, etc.
  - Implemented with a Stack (Last-in-First-out)



Finding and following links

- Crawler needs to parse HTML code to find links to follow
  - look for tags like `<a href="http://site.com/page.html">`
- Also needs to resolve relative URLs to absolute URLs
  - E.g. in the page http://www.cnn.com/linkto/:

    <a href=intl.html> refers to

    http://www.cnn.com/linkto/intl.html

    <a href=/US/> refers to

    http://www.cnn.com/US/

Canonical URLs

- Crawler converts URLs to a canonical form:
  - e.g. convert:

    http://www.cnn.com/TECH

    http://WWW.CNN.COM/TECH/

    http://www.cnn.com/bogus/../TECH/

    to:

    http://www.cnn.com/TECH/

Document Conversion

- Text is stored in hundreds of incompatible file formats
  - e.g., raw text, RTF, HTML, XML, Microsoft Word, PDF
- Non-text files also important
  - e.g., PowerPoint, Excel
- Crawlers use a conversion tool
  - converts the document content into a tagged text format such as HTML or XML
  - retains some of the important formatting information

Static vs. dynamic pages

- *Static pages* are just HTML files sent over the internet
- *Dynamic pages* are ones whose content is computed in response to your request
  - http://www.census.gov/cgi-bin/gazetteer
  - http://informatics.indiana.edu/research/colloquia.asp
  - http://www.amazon.com/exec/obidos/subst/home/home.html/002-8332429-6490452
  - http://www.imdb.com/Name?Menczer,+Erico
  - http://www.imdb.com/name/nm0578801/

- What do Google and other search engines do?

Web Crawling- implementation issues

- Fetching
- Parsing
- Link extraction and canonicalization
- Spider trap
- Page repository
- concurency

Implementation Issues : concurrency

- A crawler consumes three main resources:
  - Network,
  - CPU,
  - and disk.
- Each is a bottleneck with limits imposed by bandwidth, CPU speed, and disk seek/transfer times.
- The simple sequential crawler makes a very inefficient use of these resources because at any given time two of them are idle while the crawler attends to the third.

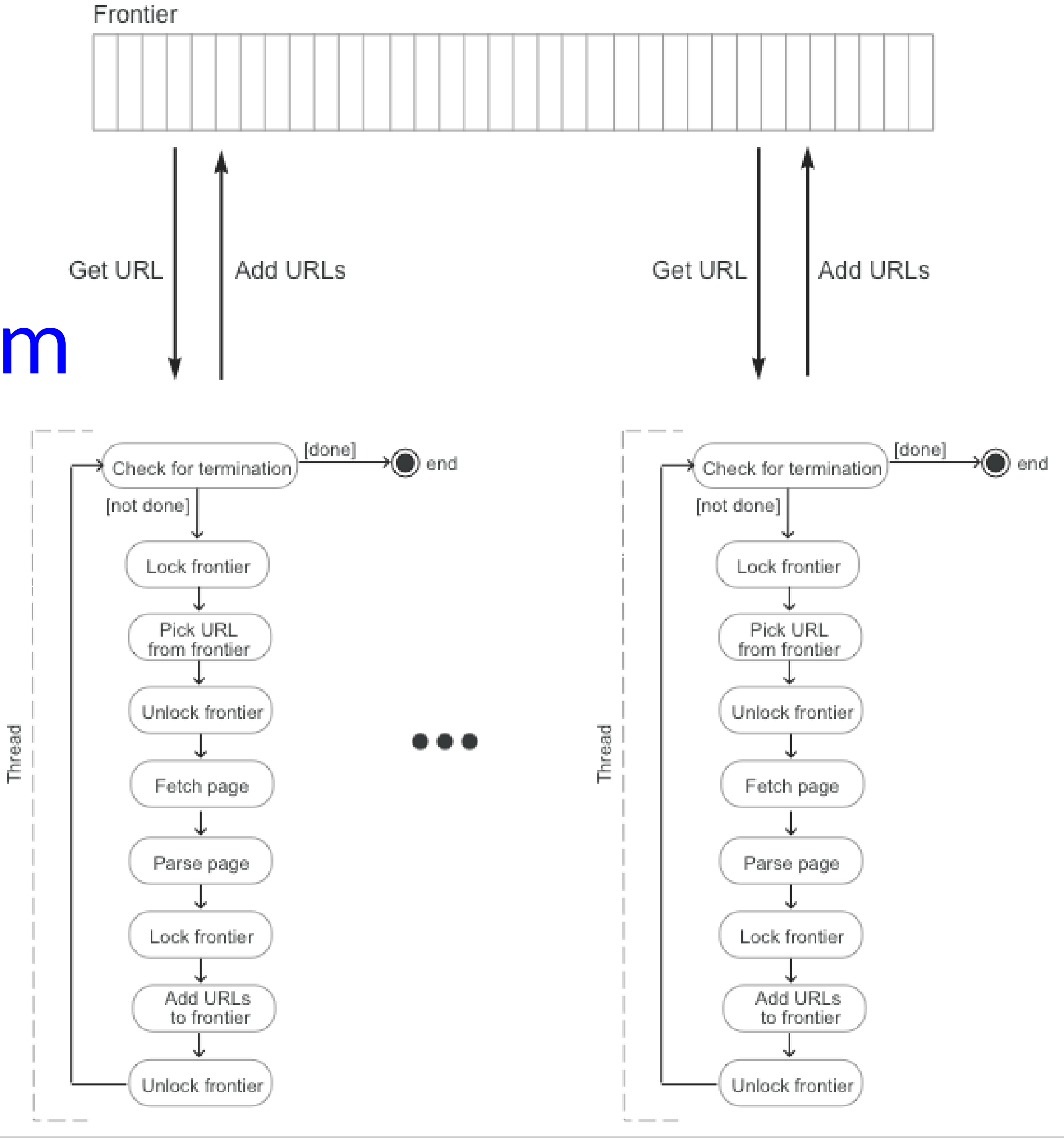
Web Crawling- concurrency

- Web crawlers waste a lot of time waiting for responses to requests
  - What’s a solution?

Web Crawling- concurrency

- The most straightforward way to speed-up a crawler is through concurrent processes or threads.
- Multiprocessing may be somewhat easier to implement than multithreading depending on the programming language and platform,
- but it may also incur a higher overhead due to the involvement of the operating system in the management (creation and destruction) of child processes.

Parallelism



Each thread or process works as an independent crawler **except for the fact that access to the shared data structures**

- mainly the frontier, and possibly the page repository must be synchronized.
- In particular a frontier manager is responsible for locking and unlocking the frontier data structures so that only one process or thread can write to them at one time.
- Note that both enqueueing and dequeuing are write operations.
- Additionally, the frontier manager would maintain and synchronize access to other shared data structures such as the crawl history for fast look-up of visited URLs.

Distributed Crawling

- Advantages to using multiple computers for crawling
  - Helps to put the crawler closer to the sites it crawls
  - Reduces the number of sites the crawler has to remember
  - Reduces computing resources required

- Distributed crawler uses a hash function to assign URLs to crawling computers
  - hash function should be computed on the host part of each URL

- Disadvantages of distributed crawling?

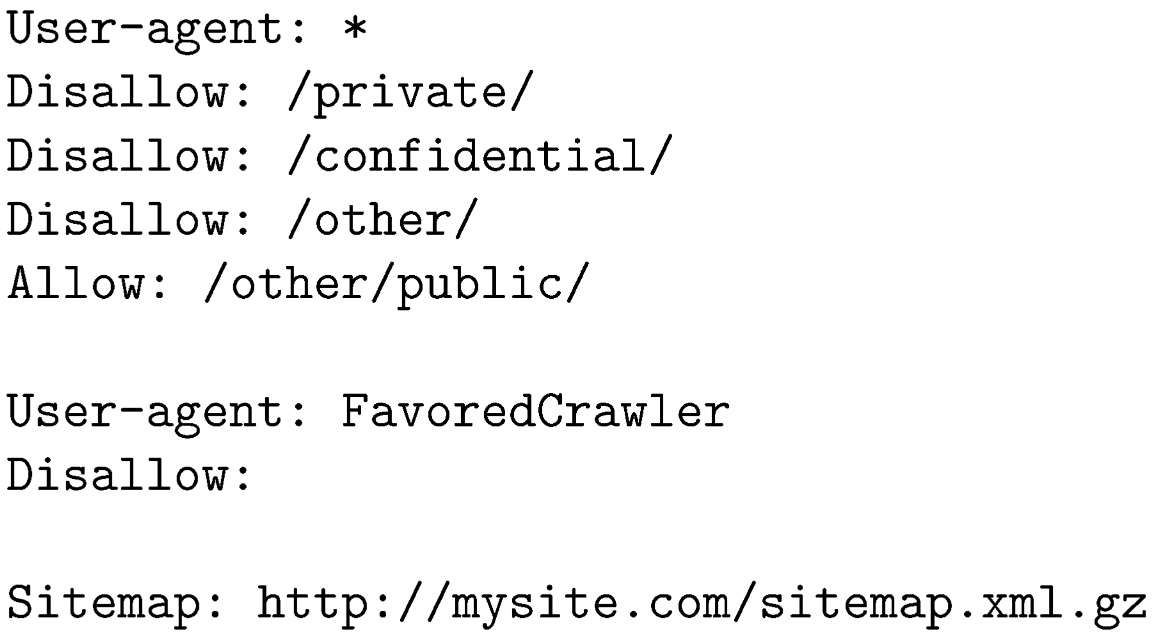
Politeness

- Modern crawlers use multiple machines to fetch hundreds of pages at once
  - But this could flood sites with requests for pages

- To avoid this, web crawlers use *politeness policies*
  - e.g., delay between requests to same web server

Controlling Crawling

- Even slow crawling will anger some web hosts, who object to any copying of their data
- Robots.txt file can be used to control crawlers
  - Websites can include this file in the main directory of their site; (nice) crawlers look for it and follow its directions



```
User-agent: *
Disallow: /private/
Disallow: /confidential/
Disallow: /other/
Allow: /other/public/

User-agent: FavoredCrawler
Disallow:

Sitemap: http://mysite.com/sitemap.xml.gz
```

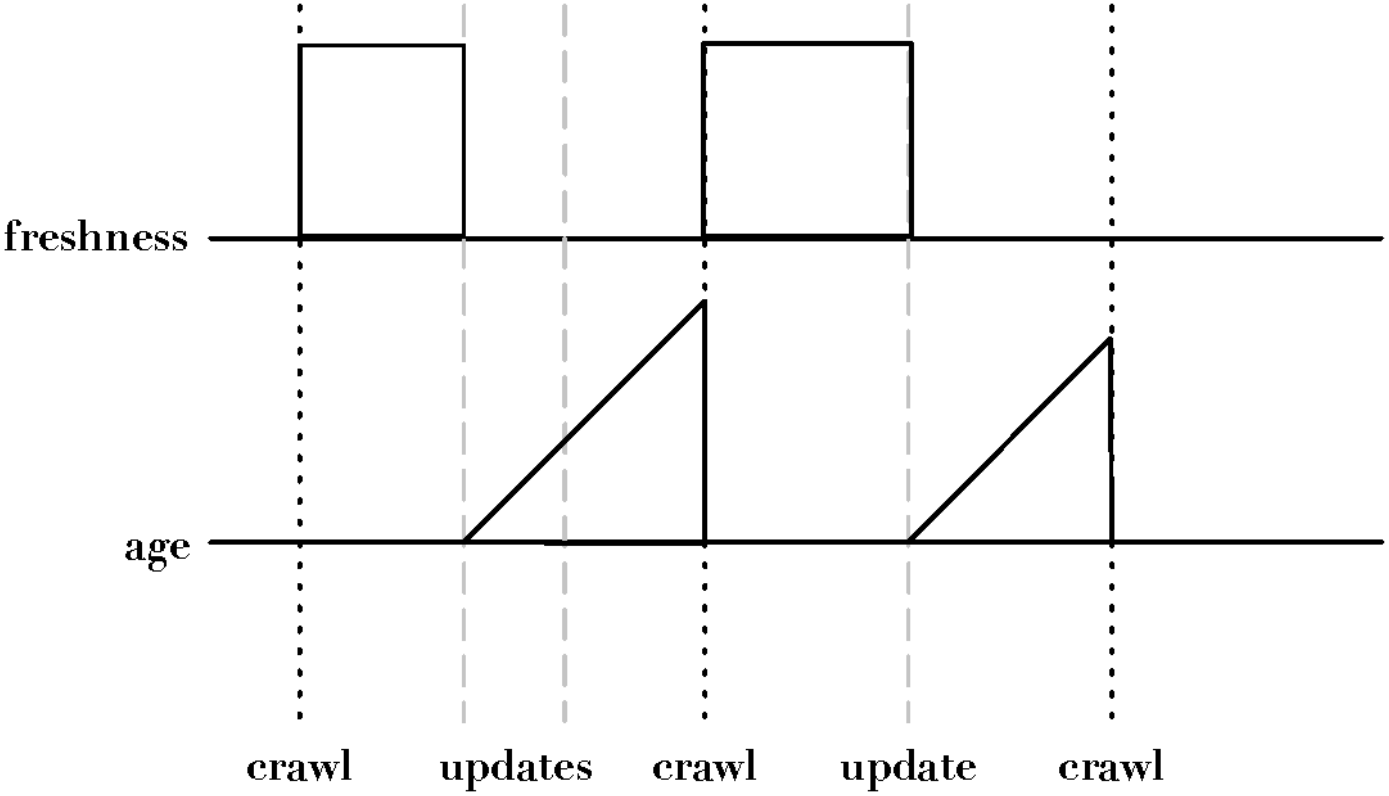
Crawler performance

- Coverage
  - Can the crawler find every page?
- Freshness
  - How frequently can a crawler revisit ?
- Trade-off!
  - Crawlers need to prioritize which pages to visit

Freshness

- Pages are constantly added, deleted, and modified
- *Freshness* is the percentage of pages for which the search engine has a current copy
  - *Staleness* is the % for which we have an outdated copy
  - *Age* is the # of days that an average page is out-of-date
  - Search engines try to maximize freshness
  - Crawlers revisit pages they have already crawled to see if they have changed

Freshness vs. Age



Age

- Expected age of a page *t* days after it was last crawled:

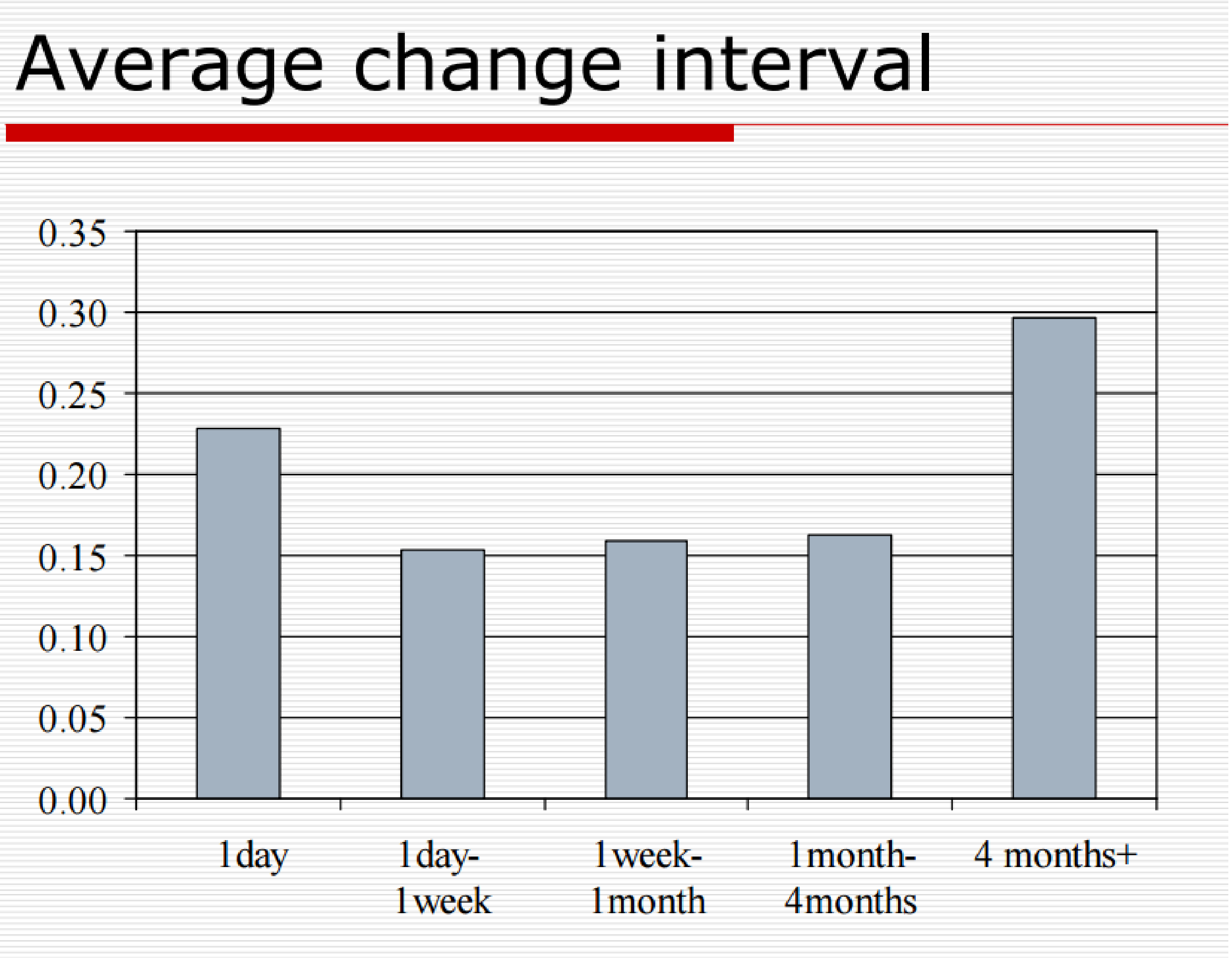
$$\text{Age}(\lambda, t) = \int_0^t P(\text{page changed at time } x)(t - x)dx$$

- Page updates generally follow a Poisson distribution
  - – time until the next update is governed by an exponential distribution

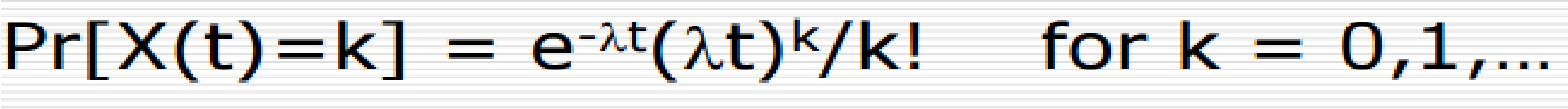
$$\text{Age}(\lambda, t) = \int_0^t \lambda e^{-\lambda x}(t - x)dx$$

How often do pages change?

- Cho et al (2000) experiment



How often do pages change?

- Assuming changes to a web page are a sequence of random events that happen independently at a fixed average rate
- Poisson process with parameter lambda
- Let X(t) be a random variable denoting the number of changes in any time interval t
- 

$\Pr[X(t)=k] = e^{-\lambda t}(\lambda t)^{k}/k! \quad \text{for } k = 0,1,\ldots$

Poisson processes

- Let us compute the expected number of changes in unit time

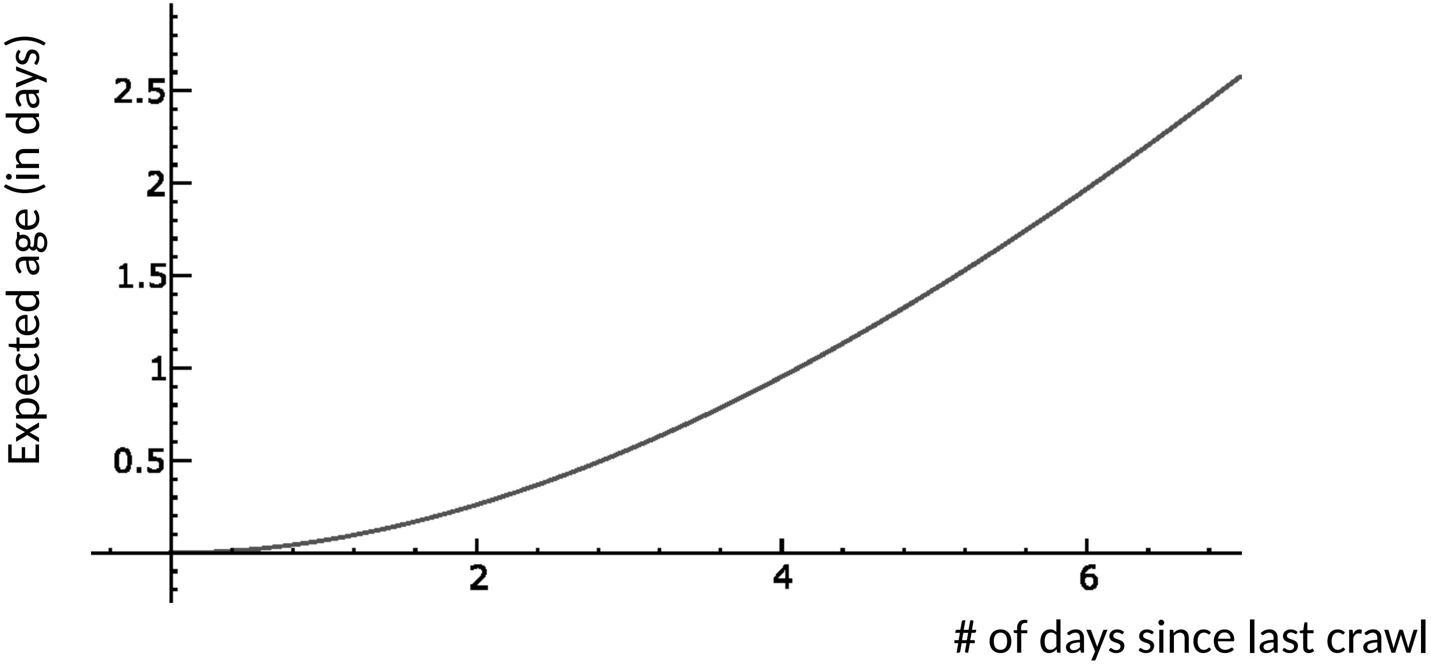
$$E[X(1)] = \Sigma_k k e^{\lambda} \lambda^k / k! = \lambda$$

-
- Lambda is therefore the average number of changes in unit time
- Called the rate parameter

Estimated age

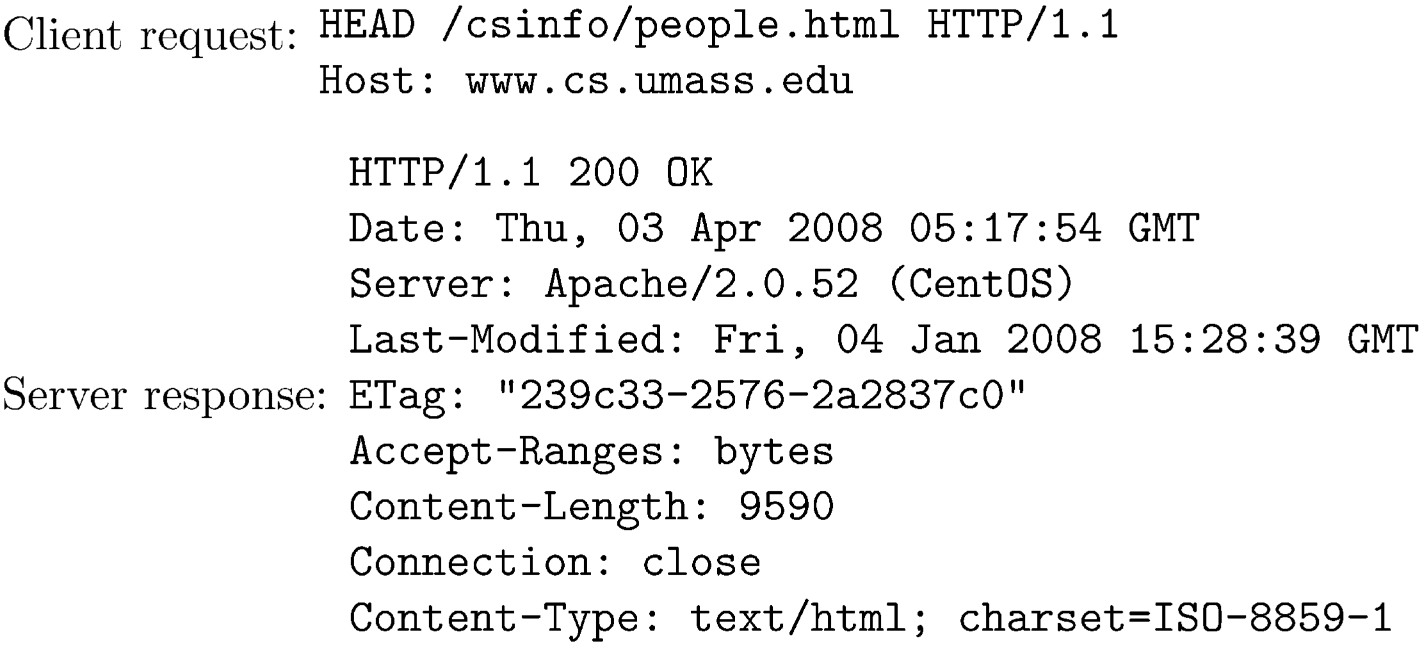
- Given an estimate of how often a page changes (λ), we can estimate the current age of a page

e.g. Expected age with λ = 1/7 (one change per week):



Checking freshness

- HTTP protocol has a special request type called HEAD that makes it easy to check for page changes
  - returns information about page, not page itself



```
Client request: HEAD /csinfo/people.html HTTP/1.1
                Host: www.cs.umass.edu

                 HTTP/1.1 200 OK
                 Date: Thu, 03 Apr 2008 05:17:54 GMT
                 Server: Apache/2.0.52 (CentOS)
                 Last-Modified: Fri, 04 Jan 2008 15:28:39 GMT
Server response: ETag: "239c33-2576-2a2837c0"
                 Accept-Ranges: bytes
                 Content-Length: 9590
                 Connection: close
                 Content-Type: text/html; charset=ISO-8859-1
```

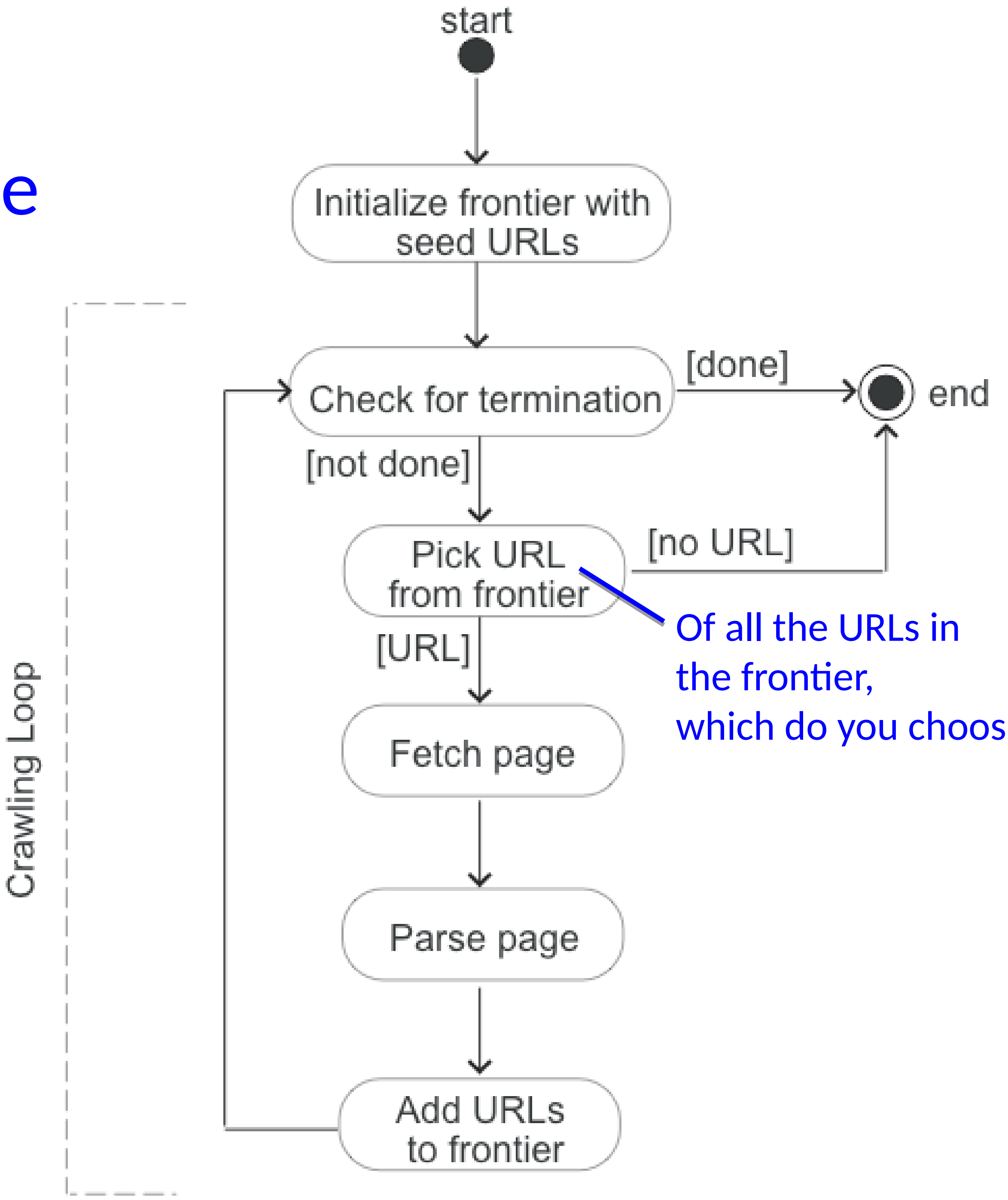
Other crawling strategies

- We’ve seen two crawling strategies so far
  - which differ in the order that the web is crawled
  - but neither one looks at the content of pages
- Modern “smart” crawlers prioritize links based on a variety of factors
  - e.g. anchor text, text surrounding link, age of server, estimates of page change rate, etc.
  - The exact techniques are closely guarded secrets

Smart crawlers

- *Best*-first search
  - Explore pages that seem “most promising” first
  - How to define most promising?
- Selective crawler
  - Bias towards most “relevant”, closest to seeds, largest pagerank, unknown servers, highest rate of change, etc…
- Topical crawlers
  - Best first search based on similarity to a topic of interest
  - How do we infer the topic of a page?

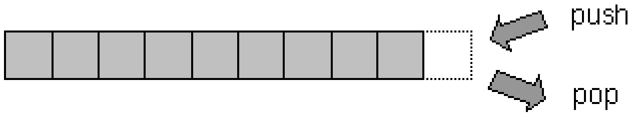
Crawler Architecture



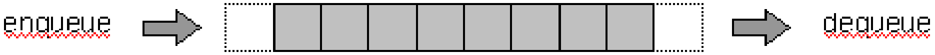
Of all the URLs in the frontier, which do you choose?

Stacks vs. Queues

- Stack



- Queue



- Priority Queue
  - You put (item, priority) pairs into queue
  - You remove the highest-priority item from the queue

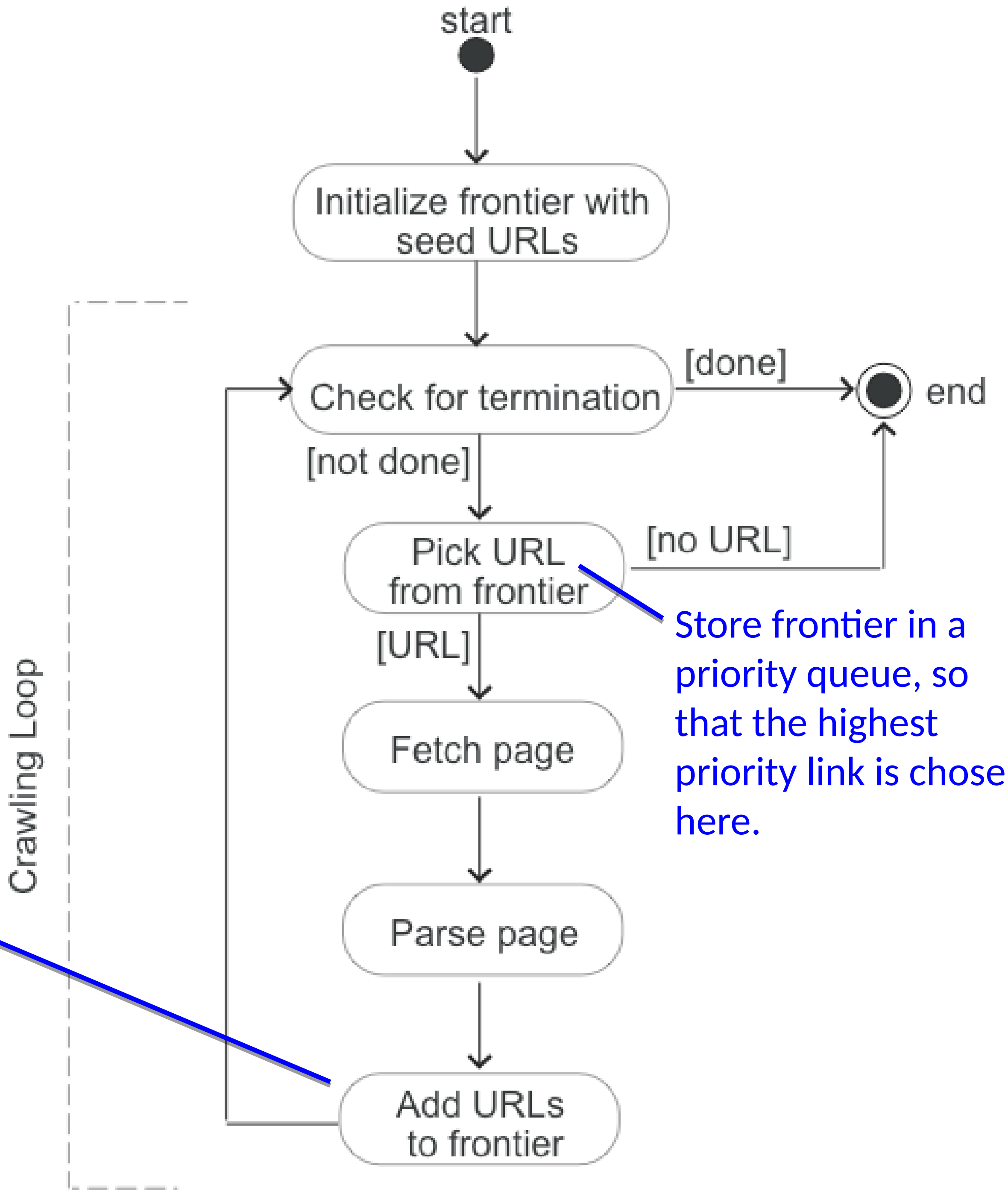
Priority queue

- Priority queue is a best-first data structure
  - When you add something to a PQ, you give an importance (priority number)
  - When you remove something, the PQ gives you the *highest priority* item in the queue
  - If multiple items have the same priority, it returns the one that was added *first*

Priority queue examples

- Hospital emergency room
  - Incoming patients see a triage nurse who assigns a priority to each patient. Highest need patients are seen first.
- Airplane boarding
  - First class, Business class, Coach
  - FIFO within each class
- Operating system scheduling
  - Important system jobs (memory management, etc) are given priority over user tasks

Smart Crawler Architecture



Store frontier in a priority queue, so that the highest priority link is chosen here.

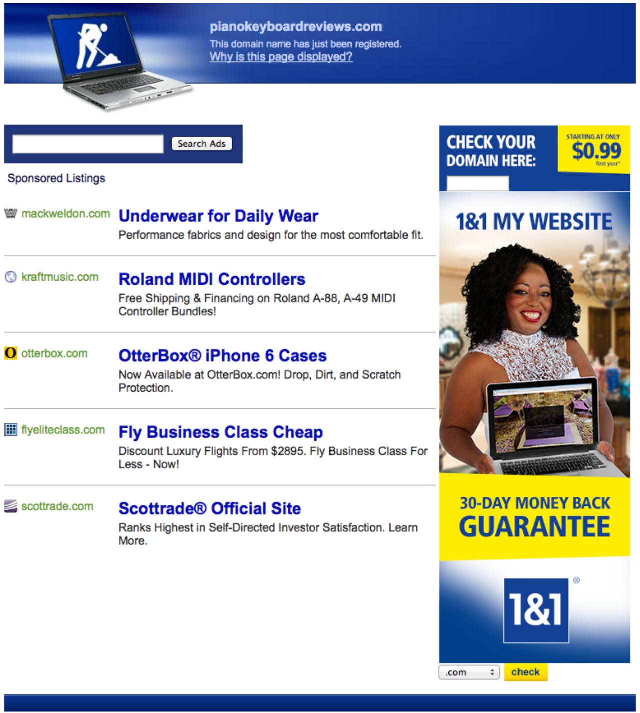
Estimate importance of each URL. E.g., use surrounding text to identify “promising” links, or punish "spamy" pages, or bias towards servers that are popular or not yet known.

PQ implementation

- Priority queues are typically implemented using a *heap*
  - A binary tree with a special property: The highest-priority element of any subtree is always at the root of the subtree.
  - Learn more in a data structures class…

One application: avoiding spam

- Spam is a huge problem on the web
  - i.e. Useless pages set up to trick people into visiting them, e.g. for ad revenue
  - Crawlers want to avoid these pages



pianokeyboardreviews.com
This domain name has just been registered.
Why is this page displayed?

Sponsored Listings

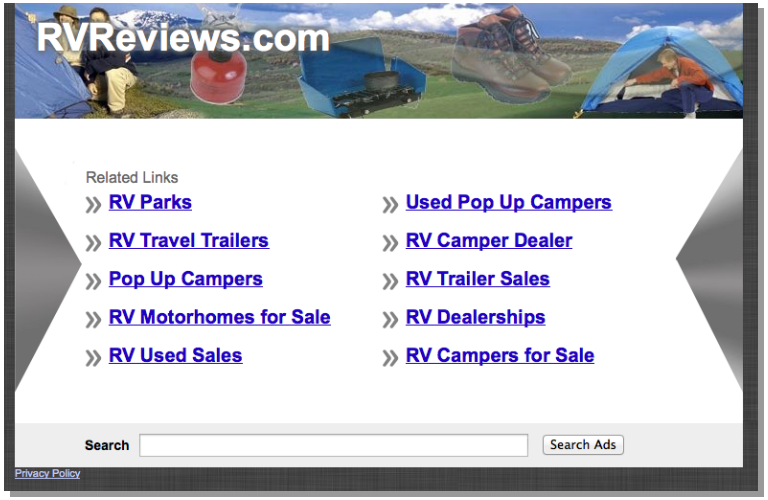
mackweldon.com **Underwear for Daily Wear**
Performance fabrics and design for the most comfortable fit.

kraftmusic.com **Roland MIDI Controllers**
Free Shipping & Financing on Roland A-88, A-49 MIDI Controller Bundles!

otterbox.com **OtterBox® iPhone 6 Cases**
Now Available at OtterBox.com! Drop, Dirt, and Scratch Protection.

flyeliteclass.com **Fly Business Class Cheap**
Discount Luxury Flights From $2895. Fly Business Class For Less - Now!

scottrade.com **Scottrade® Official Site**
Ranks Highest in Self-Directed Investor Satisfaction. Learn More.



Related Links

- RV Parks
- RV Travel Trailers
- Pop Up Campers
- RV Motorhomes for Sale
- RV Used Sales
- Used Pop Up Campers
- RV Camper Dealer
- RV Trailer Sales
- RV Dealerships
- RV Campers for Sale

Spam

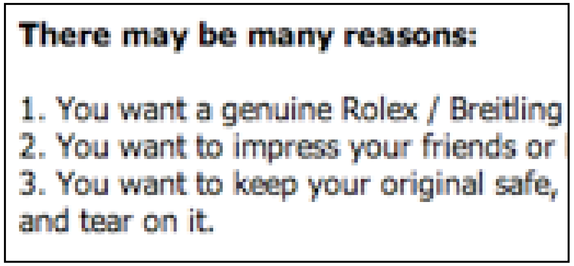
- Spam = junk e-mail and web pages

- A big problem! [Commtouch07]
  - ~96% of all email traffic on the Internet
  - ~150 billion junk emails per day
  - Spreads malware, worms, phishing schemes, etc.

- We need *classifiers* that can automatically detect spam web pages and emails
  - But, it's hard to define what spam is exactly
  - So we want to use *machine learning*, so that the computer learns what spam looks like over time!

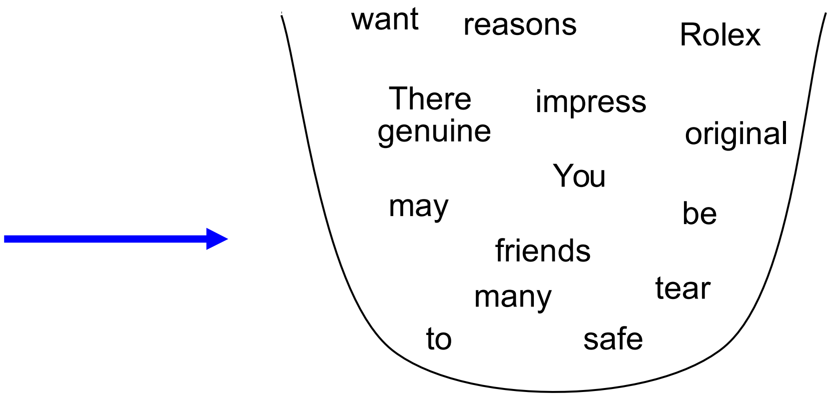
Modeling a document

- Use natural language processing techniques?
  - Parse the web page, understand the meaning, decide if email is spam
  - Too difficult for now
- Simpler alternative
  - Represent a document as an unordered collection of words (a *bag of words* model)



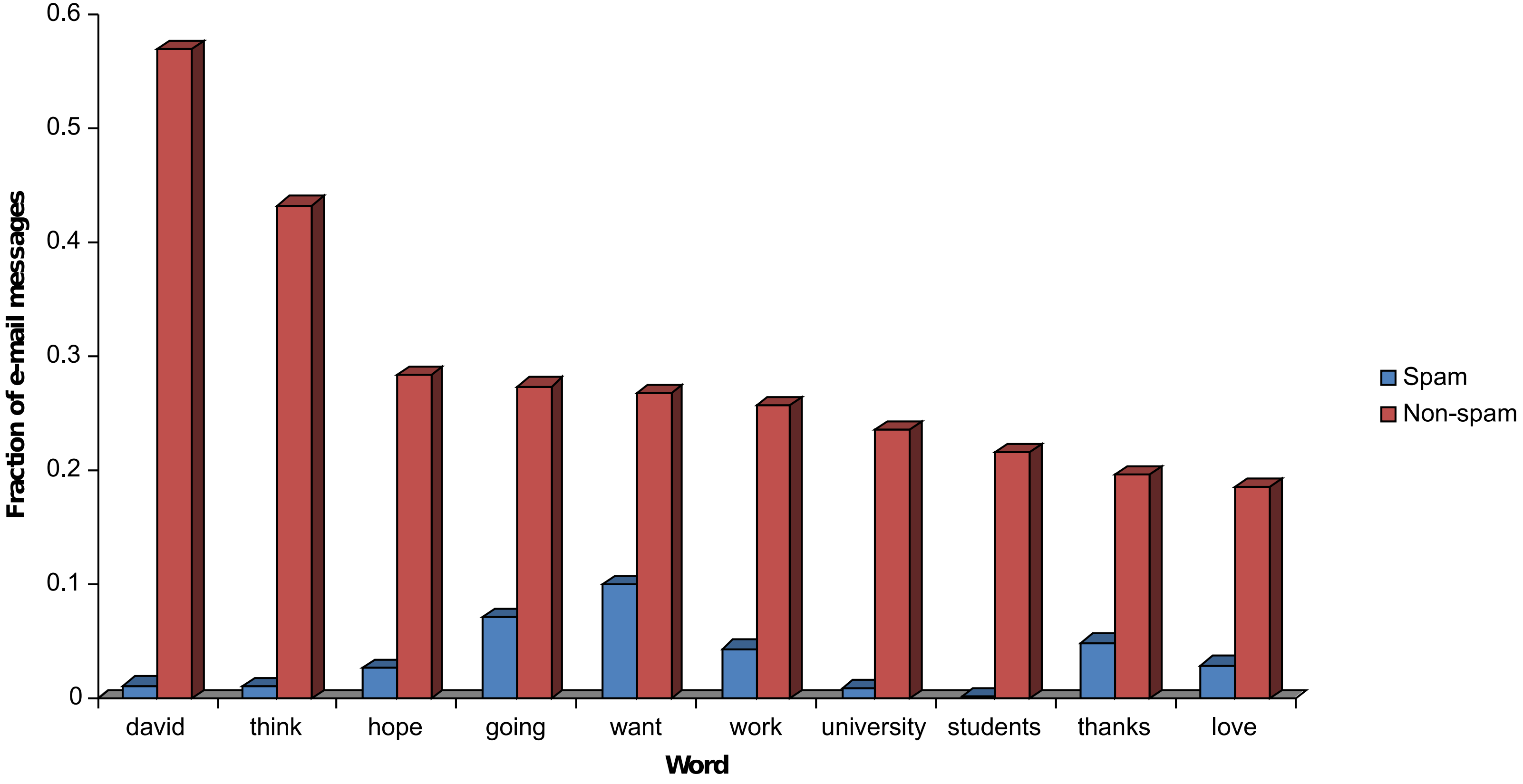
There may be many reasons:

1. You want a genuine Rolex / Breitling
2. You want to impress your friends or
3. You want to keep your original safe,
and tear on it.



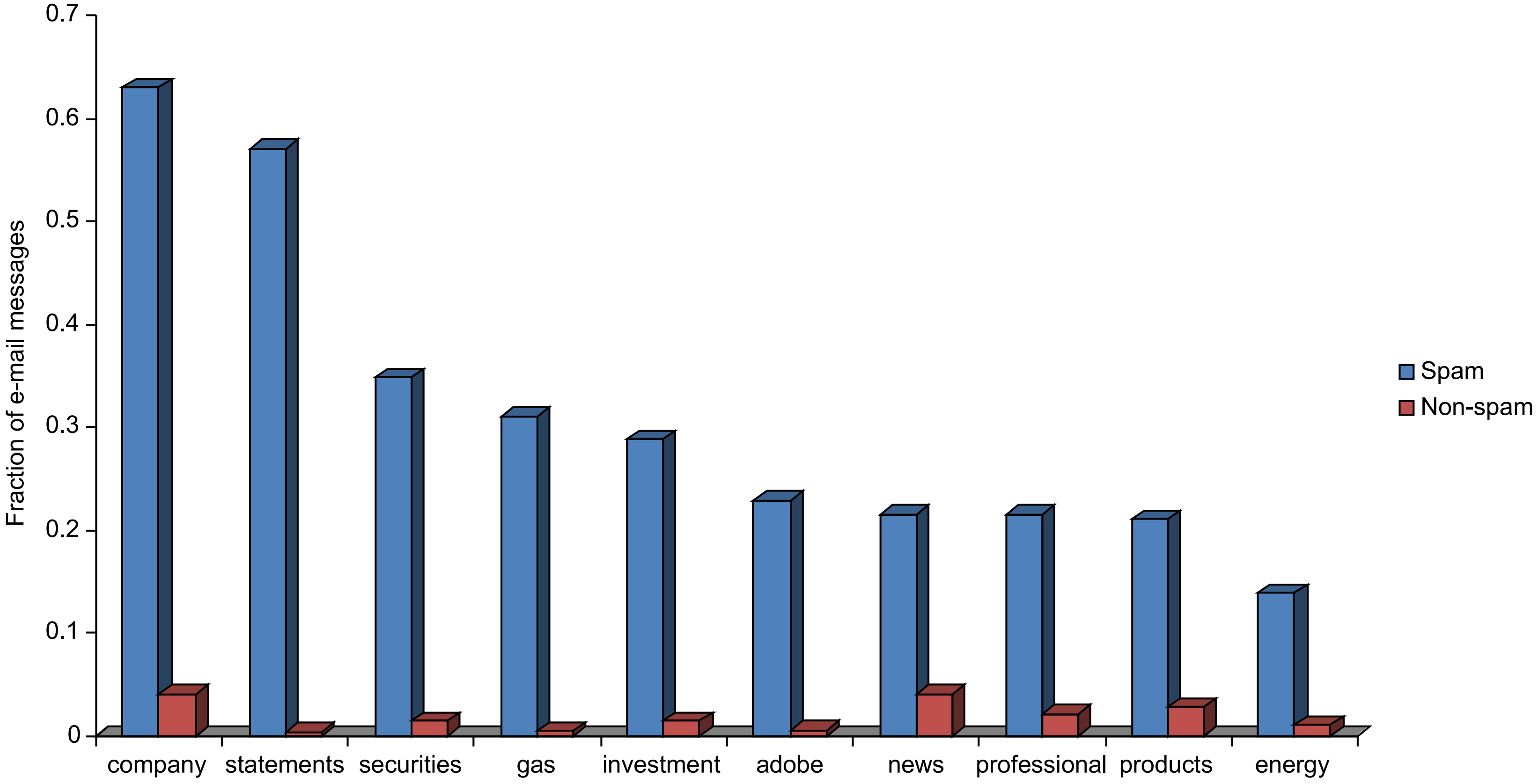
Statistical motivation

- Spam and (my) non-spam are statistically very different



Statistical motivation

- Spam and (my) non-spam are statistically very different



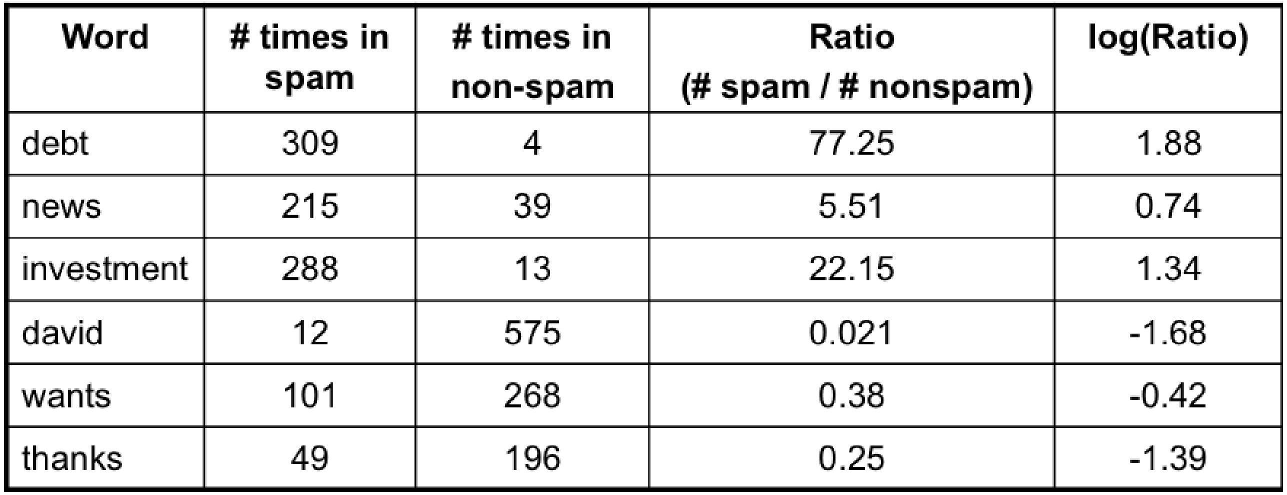
An example

- We can take some documents known to be spam and known to be legitimate, to estimate relative importance of words

Word	# times in spam	# times in non-spam	Ratio (# spam / # nonspam)	log(Ratio)
debt	309	4	77.25	1.88
news	215	39	5.51	0.74
investment	288	13	22.15	1.34
david	12	575	0.021	-1.68
wants	101	268	0.38	-0.42
thanks	49	196	0.25	-1.39

Classifying spam

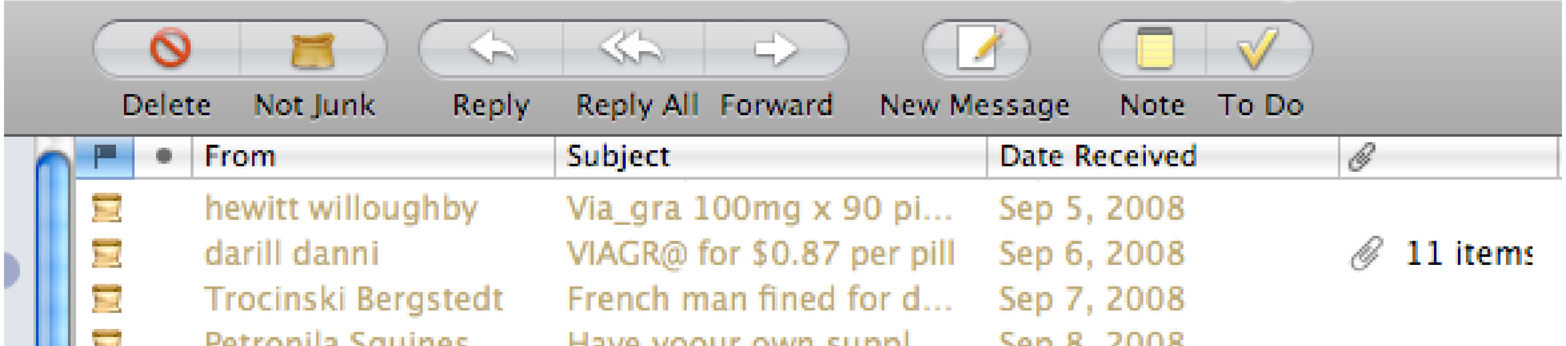
- Once we’ve constructed this table, we can use a *Bayesian classifier* to decide if a new document is spam
  – You can learn more about classifiers in a Machine Learning class
  – Multiply together the ratios for each word in the document; if greater than 1, it’s spam, and otherwise not spam
  – Or, equivalently, add up the log(ratios) for each word; if greater than 0, it’s spam, and otherwise not spam



Word	# times in spam	# times in non-spam	Ratio (# spam / # nonspam)	log(Ratio)
debt	309	4	77.25	1.88
news	215	39	5.51	0.74
investment	288	13	22.15	1.34
david	12	575	0.021	-1.68
wants	101	268	0.38	-0.42
thanks	49	196	0.25	-1.39

Learning

- An advantage of a Bayesian classifier is that it "learns" what spam looks like automatically
  - Just by counting #'s of words in spam and non-spam.
  - No need for hand-crafted rules.
  - But a good set of training data is critical.
- The classifier can continue to learn with time
  - User corrects the classifier's errors, classifier adjusts its word counts accordingly



From	Subject	Date Received	📎
hewitt willoughby	Via_gra 100mg x 90 pi...	Sep 5, 2008	
darill danni	VIAGR@ for $0.87 per pill	Sep 6, 2008	📎 11 items
Trocinski Bergstedt	French man fined for d...	Sep 7, 2008	
Petronila Squines	Have yoour own suppl	Sep 8, 2008	

Bayesian poisoning

- Spammers try to confuse the Bayesian filters
- Passive attacks
  - Add many non-spam words to web pages
  - Disguise spam words by misspelling (e.g. viagra -> vi@gra)
- Active attacks
  - Assume that it’s possible for spammer to see if an email (or webpage) is filtered out by the classifier
  - Send many email variants, observing the filter’s decision
  - Tune the emails to stay “one step ahead” of the filter