

Informatics 225

Computer Science 221

Information Retrieval

Lecture 9

Duplication of course material for any commercial purpose without the explicit written permission of the professor is prohibited.

These course materials borrow, with permission, from those of Prof. Cristina Videira Lopes, Addison Wesley 2008, Chris Manning, Pandu Nayak, Hinrich Schütze, Heike Adel, Sascha Rothe, Jerome H. Friedman, Robert Tibshirani, and Trevor Hastie. Powerpoint theme by Prof. André van der Hoek.

Back to crawling

Information Retrieval

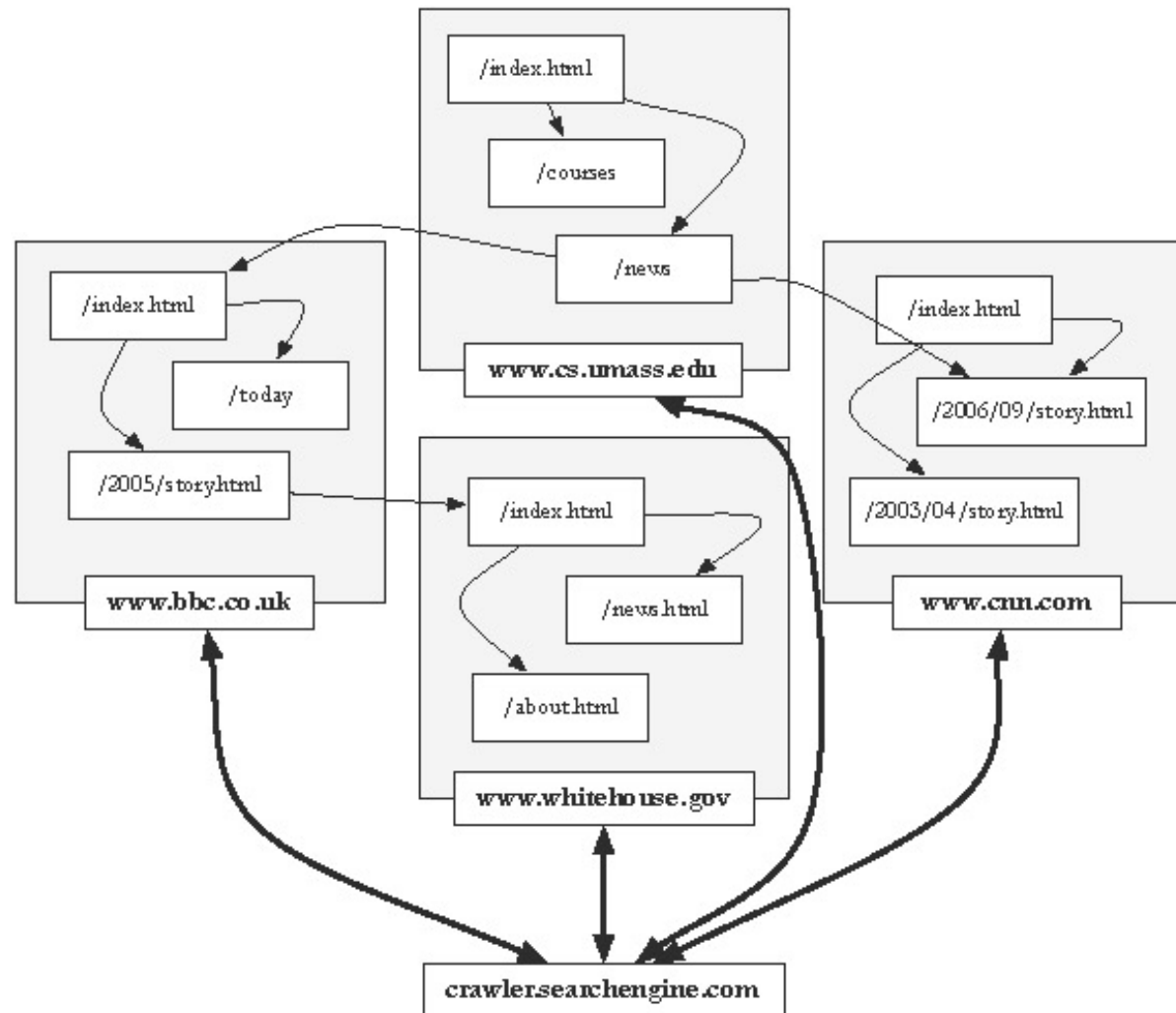
Information adapted from Addison Wesley, 2008

Pseudo Code

```
procedure CRAWLERTHREAD(frontier)
  while not frontier.done() do
    website ← frontier.nextSite()
    url ← website.nextURL()
    if website.permitsCrawl(url) then
      text ← retrieveURL(url)
      storeDocument(url, text)
      for each url in parse(text) do
        frontier.addURL(url)
      end for
    end if
    frontier.releaseSite(website)
  end while
end procedure
```

Two major jobs : download webpages if authorized, and discover new URLs.

How the crawler sees the web



```
procedure CRAWLERTHREAD(frontier)
  while not frontier.done() do
    website ← frontier.nextSite()
    url ← website.nextURL()
    if website.permitsCrawl(url) then
      text ← retrieveURL(url)
      storeDocument(url, text)
      for each url in parse(text) do
        frontier.addURL(url)
      end for
    end if
    frontier.releaseSite(website)
  end while
end procedure
```

Retrieving Web Pages

- Web crawler client program connects to a *domain name system* (DNS) server
- DNS server translates the hostname into an *internet protocol* (IP) address
- Crawler then attempts to connect to server host using specific *port*
- After connection, crawler sends an HTTP request to the web server to request a page
 - usually a GET request

Web Crawling

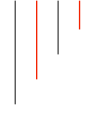
- **Problem:** Web crawlers spend a lot of time waiting for responses to requests

Web Crawling

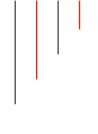


- **Problem:** Web crawlers **spend a lot of time waiting** for responses to requests
- **Solution:** To reduce this inefficiency, web crawlers **use threads** and fetch hundreds of pages at once

Web Crawling



- **Problem:** Web crawlers **spend a lot of time waiting** for responses to requests
- **Solution:** To reduce this inefficiency, web crawlers **use threads** and fetch hundreds of pages at once
- **Drawback:** Crawlers could potentially **flood sites** with requests for pages



- **Problem:** Web crawlers **spend a lot of time waiting** for responses to requests
- **Solution:** To reduce this inefficiency, web crawlers **use threads** and fetch hundreds of pages at once
- **Drawback:** Crawlers could potentially **flood sites** with requests for pages
- **Mitigation:** To avoid this problem, web crawlers use *politeness policies*
 - e.g., distribute requests between different servers, delay between requests to same web server, ...

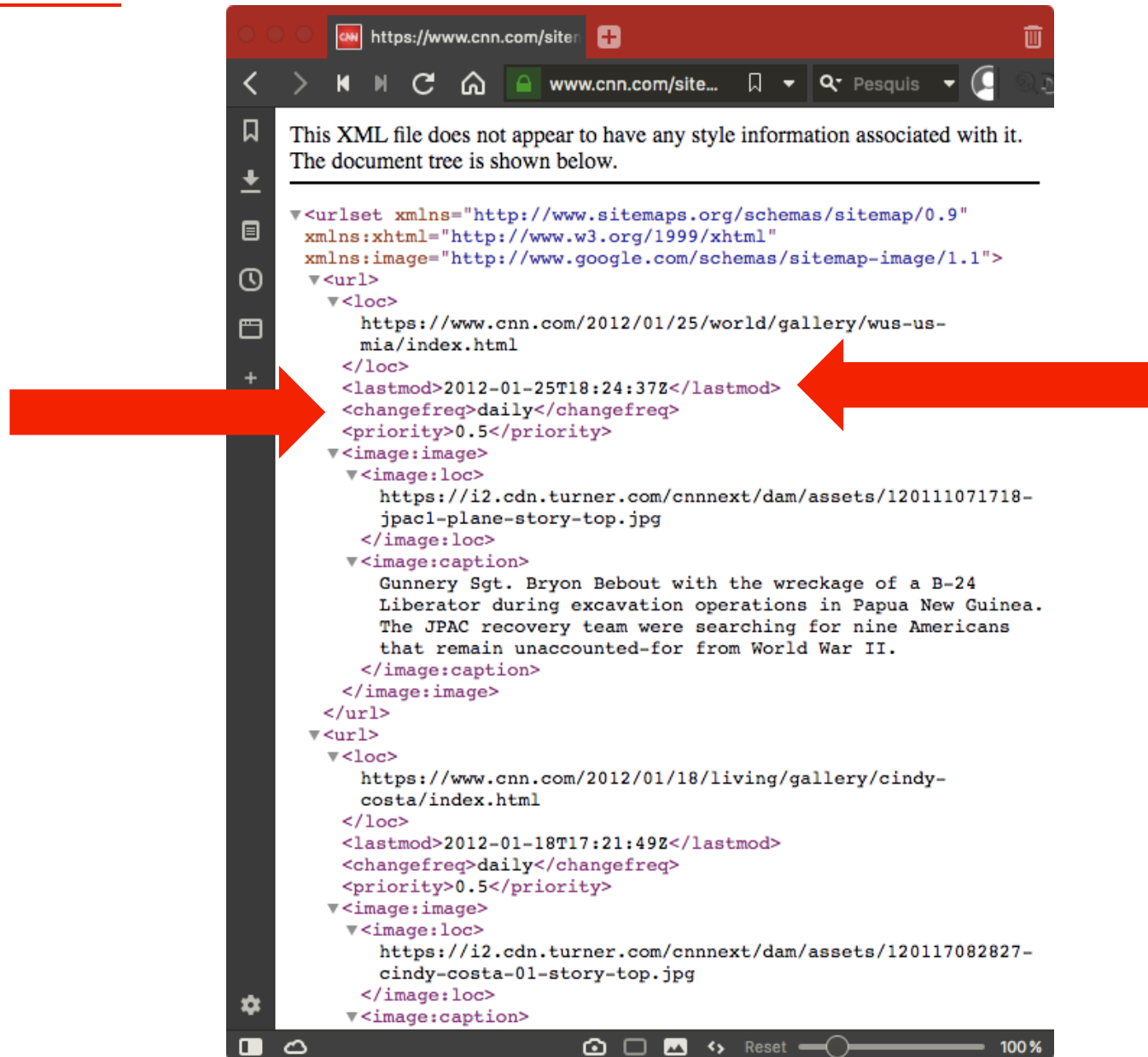
Pseudo Code : where to enforce politeness?

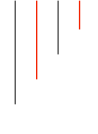
```
procedure CRAWLERTHREAD(frontier)
  while not frontier.done() do
    website ← frontier.nextSite() ← Can be here
    url ← website.nextURL() ← And here
    if website.permitsCrawl(url) then
      text ← retrieveURL(url)
      storeDocument(url, text)
      for each url in parse(text) do
        frontier.addURL(url)
      end for
    end if
    frontier.releaseSite(website)
  end while
end procedure
```

Sitemaps

- Sitemaps contain lists of URLs and data about those URLs, such as modification time and modification frequency
- Generated by web server administrators (usually by scripts)
- Can inform the crawler about pages it might not find
- Gives crawler **a hint about when** to check a page for changes

Sitemap Example





- In a website, webpages are constantly:

added, deleted, and modified
- *Stale copies (or their representation in our index) no longer reflect the real contents of the web pages*
- Web crawler must continually revisit pages
 - Check for modifications and maintain the **freshness** of the collection

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



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

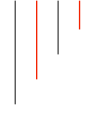
Content-Length: 9590

Connection: close

Content-Type: text/html; charset=ISO-8859-1

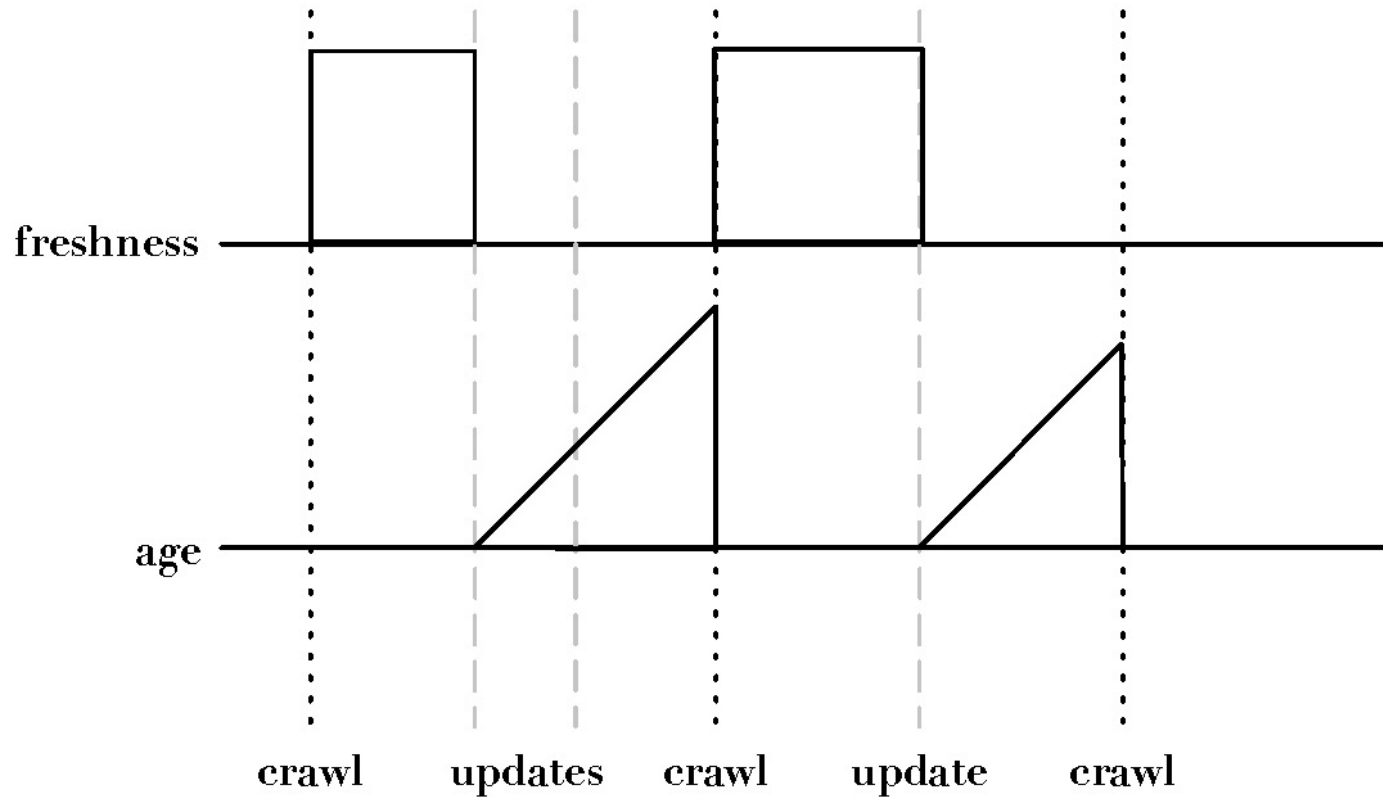


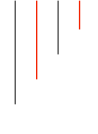
Try, for instance using “curl --head https://www.ics.uci.edu/~algol/index.html”



- Not possible to constantly check all pages
 - must check important pages and pages that change frequently
- Freshness is the proportion of pages that are fresh in the collection
- Optimizing for this metric is dangerous:
 - E.g. If you have a website that is always updated, optimizing your search engine for freshness would lead you to not crawl that site.
- Another metric: *Age*

Freshness vs. Age





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

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

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

$$\text{Age metric } \text{Age}(\lambda, t) = \int_0^t \underbrace{P(\text{page changed at time } x)}_{\text{The probability for the page to change at time } x} \underbrace{(t - x)}_{\text{some time difference between the crawling and the page change}} dx$$

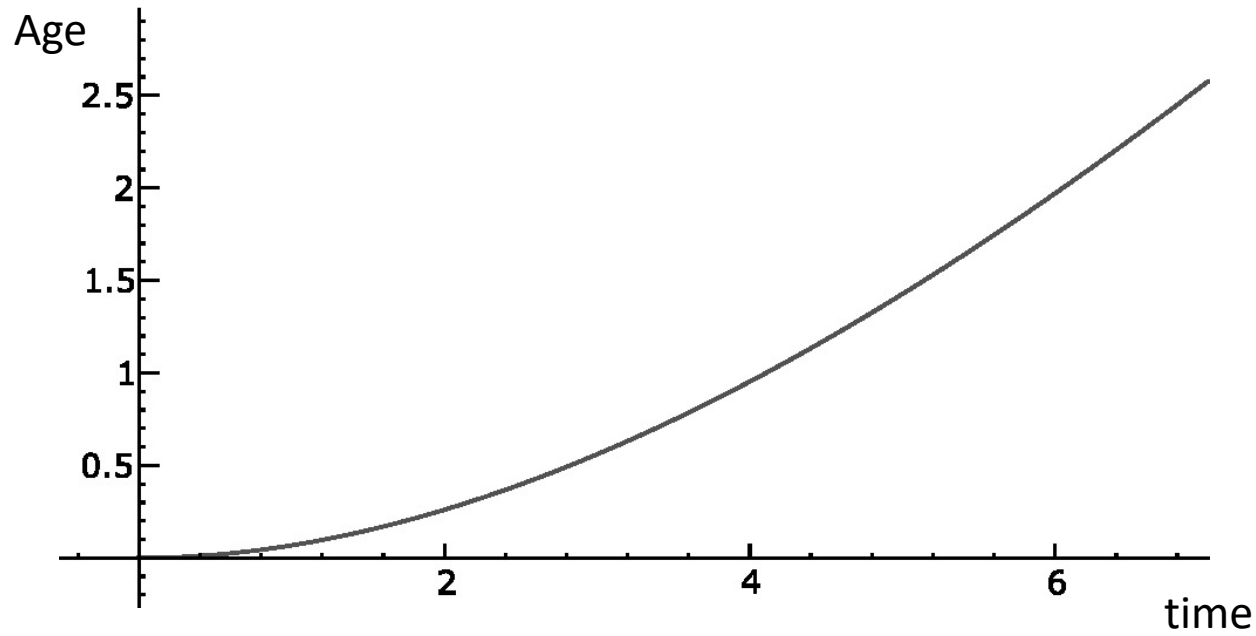
- Expected age of a page t days after it was last crawled:

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

- Web page updates follow the Poisson distribution on average
(Cho & Garcia-Molina, 2003)
 - time until the next update is governed by an exponential distribution

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

- Older a page gets, the more it costs not to crawl it
 - e.g., expected age with mean change frequency
 $\lambda = 1/7$ (one change per week)





- Three reasons to use multiple computers for crawling
 - Helps to put the crawler closer to the sites it crawls
 - Lower requirements in the network
 - Reduces the number of sites each crawler has to remember
 - Lower requirements on local memory
 - Reduces the locally required resources
- Distributed crawlers may use a hash function to assign URLs to crawling computers/threads
 - Some hash function should be computed on the host part of each URL

```
procedure CRAWLERTHREAD(frontier)
  while not frontier.done() do
    website ← frontier.nextSite()
    url ← website.nextURL()
    if website.permitsCrawl(url) then
      text ← retrieveURL(url)
      storeDocument(url, text)
      for each url in parse(text) do
        frontier.addURL(url)
      end for
    end if
    frontier.releaseSite(website)
  end while
end procedure
```

Pseudo Code

The frontier information should be shared

```
procedure CRAWLERTHREAD(frontier)
  while not frontier.done() do
    website ← frontier.nextSite()
    url ← website.nextURL()
    if website.permitsCrawl(url) then
      text ← retrieveURL(url)
      storeDocument(url, text)
      for each url in parse(text) do
        frontier.addURL(url)
      end for
    end if
    frontier.releaseSite(website)
  end while
end procedure
```

```
procedure CRAWLERTHREAD(frontier)
  while not frontier.done() do
    website ← frontier.nextSite()
    url ← website.nextURL()
    if website.permitsCrawl(url) then
      text ← retrieveURL(url)
      storeDocument(url, text)
      for each url in parse(text) do
        frontier.addURL(url)
      end for
    end if
    frontier.releaseSite(website)
  end while
end procedure
```

```
procedure CRAWLERTHREAD(frontier)
  while not frontier.done() do
    website ← frontier.nextSite()
    url ← website.nextURL()
    if website.permitsCrawl(url) then
      text ← retrieveURL(url)
      storeDocument(url, text)
      for each url in parse(text) do
        frontier.addURL(url)
      end for
    end if
    frontier.releaseSite(website)
  end while
end procedure
```

```
procedure CRAWLERTHREAD(frontier)
  while not frontier.done() do
    website ← frontier.nextSite()
    url ← website.nextURL()
    if website.permitsCrawl(url) then
      text ← retrieveURL(url)
      storeDocument(url, text)
      for each url in parse(text) do
        frontier.addURL(url)
      end for
    end if
    frontier.releaseSite(website)
  end while
end procedure
```

```
procedure CRAWLERTHREAD(frontier)
  while not frontier.done() do
    website ← frontier.nextSite()
    url ← website.nextURL()
    if website.permitsCrawl(url) then
      text ← retrieveURL(url)
      storeDocument(url, text)
      for each url in parse(text) do
        frontier.addURL(url)
      end for
    end if
    frontier.releaseSite(website)
  end while
end procedure ...
```


Other types of crawling

Information Retrieval

Information adapted from Addison Wesley, 2008

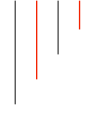
Focused Crawling

- Attempts to download only those pages that are about a particular topic
 - used by **vertical search** applications
 - can also be used to **improve a search engine in a specific topic**
 - Or around specific events in time
- Rely on the fact that pages about a topic tend to have links to other pages on the same topic
 - popular pages for a topic are typically used as seeds
 - E.g. pages that many users are clicking after they search a keyword
- Crawler uses some *text classifier* to decide whether a page is on topic

Desktop Crawls



- Used for desktop search and enterprise search
- Differences to web crawling with respect to the web
 - Much easier to find the data (file system ~ sitemap)
 - Responding quickly to updates is more important
 - Must be conservative in terms of disk, memory and CPU usage
 - *It should be transparent to the user.*
 - Need to cope with many different document formats
 - Data privacy much more important than in the web context



- In general on the web: documents are created and updated
- But many documents are also just *published*
 - created at a fixed time and rarely updated again
 - e.g., news articles, blog posts, press releases, email
- **Document feed:** Published documents from a single source can be ordered in a sequence
 - Crawlers can find new documents by examining the end of the feed

Document Feeds

- Two types of document feeds:
 - A **push feed** alerts the subscriber to new documents
 - E.g. news agencies
 - A **pull feed** requires the subscriber to check periodically for new documents
- A *push* mechanism involves a subscription:
 - E.g. WebSub (<https://www.w3.org/TR/websub/>)
- Most common format for (*pull*) feeds is called *RSS*
 - Really Simple Syndication, Resource Description Framework Site Summary, Rich Site Summary

RSS Example

```
<?xml version="1.0"?>
<rss version="2.0">
  <channel>
    <title>Search Engine News</title>
    <link>http://www.search-engine-news.org/</link>
    <description>News about search engines.</description>
    <language>en-us</language>
    <pubDate>Tue, 19 Jun 2008 05:17:00 GMT</pubDate>
    <ttl>60</ttl>

    <item>
      <title>Upcoming SIGIR Conference</title>
      <link>http://www.sigir.org/conference</link>
      <description>The annual SIGIR conference is coming!
        Mark your calendars and check for cheap
        flights.</description>
      <pubDate>Tue, 05 Jun 2008 09:50:11 GMT</pubDate>
      <guid>http://search-engine-news.org#500</guid>
    </item>
```

RSS Example

FEED CHANNEL INFO

```
<?xml version="1.0"?>
<rss version="2.0">
  <channel>
```

```
    <title>Search Engine News</title>
    <link>http://www.search-engine-news.org/</link>
    <description>News about search engines.</description>
    <language>en-us</language>
    <pubDate>Tue, 19 Jun 2008 05:17:00 GMT</pubDate>
    <ttl>60</ttl>
```


ITEM INFO

```
  <item>
    <title>Upcoming SIGIR Conference</title>
    <link>http://www.sigir.org/conference</link>
    <description>The annual SIGIR conference is coming!
      Mark your calendars and check for cheap
      flights.</description>
    <pubDate>Tue, 05 Jun 2008 09:50:11 GMT</pubDate>
    <guid>http://search-engine-news.org#500</guid>
  </item>
```


...

RSS Example

```
<?xml version="1.0"?>
<rss version="2.0">
  <channel>
```



```
    <title>Search Engine News</title>
    <link>http://www.search-engine-news.org/</link>
    <description>News about search engines.</description>
    <language>en-us</language>
    <pubDate>Tue, 19 Jun 2008 05:17:00 GMT</pubDate>
    <ttl>60</ttl>
```

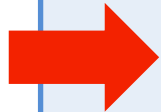


```
  <item>
    <title>Upcoming SIGIR Conference</title>
    <link>http://www.sigir.org/conference</link>
    <description>The annual SIGIR conference is coming!
      Mark your calendars and check for cheap
      flights.</description>
    <pubDate>Tue, 05 Jun 2008 09:50:11 GMT</pubDate>
    <guid>http://search-engine-news.org#500</guid>
  </item>
```


RSS Example

...

```
<item>
  <title>New Search Engine Textbook</title>
  <link>http://www.cs.umass.edu/search-book</link>
  <description>A new textbook about search engines
    will be published soon.</description>
  <pubDate>Tue, 05 Jun 2008 09:33:01 GMT</pubDate>
  <guid>http://search-engine-news.org#499</guid>
</item>
```



```
</channel>
</rss>
```

Major characteristics of RSS to crawlers

- `ttl` tag (time to live)
 - amount of time (in minutes) contents should be cached
- RSS feeds are accessed like web pages
 - using HTTP GET requests to web servers that host them
 - You can use HTTP HEAD requests to check if the feed has changed
- Easy for crawlers to parse (it is just XML)
- Easy to find new information:
 - just GET, parse and if any, grab the new `<item> ... </item>` s.