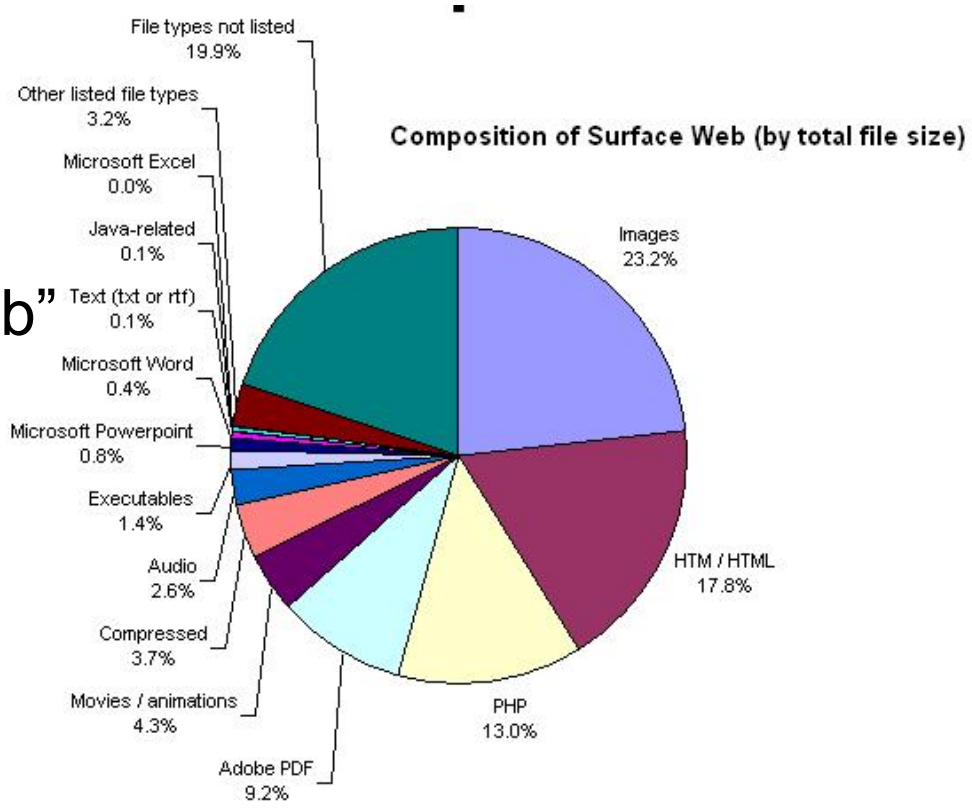
Web Challenges for IR

- Distributed Data: Documents spread over millions of different web servers.
- Volatile Data: Many documents change or disappear rapidly (e.g. dead links).
- Large Volume: Billions of separate documents.
- Unstructured and Redundant Data: No uniform structure, HTML errors, up to 30% (near) duplicate documents.
- Quality of Data: No editorial control, false information, poor quality writing, typos, etc.
- Heterogeneous Data: Multiple media types (images, video, VRML), languages, character sets, etc.

The Web (Corpus) by the Numbers (1)

- 43 million web servers
- 167 Terabytes of data
  - About 20% text/html
- 100 Terabytes in “deep Web”
- 440 Terabytes in emails
  - Original content



- [Lyman & Varian: How much Information? 2003]
  - http://www.sims.berkeley.edu/research/projects/how-much-info-2003/

The Web (Corpus) by the Numbers (2)

35 Terabytes of text on surface Web?

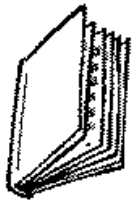
⇩

35 academic research libraries (with some 20,000 meters of shelved books each!)

1 Kilobyte = a very short story

"Jack and Jill went up the hill to fetch a pail of water. Jack fell down and broke his crown and Jill came tumbling after."

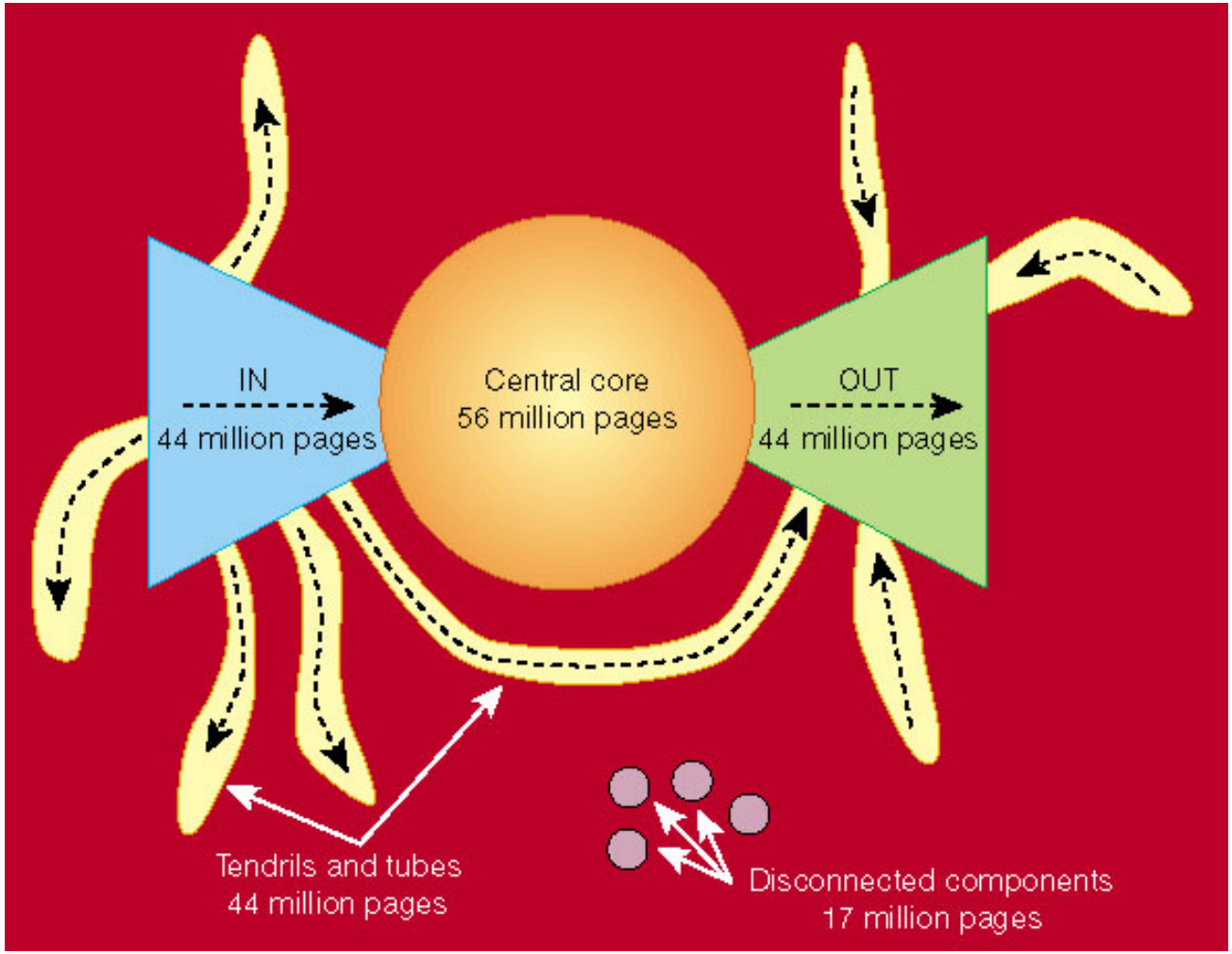
1 Megabyte = a short book



1 Gigabyte = 20 meters of shelved books

1 Terabyte = an academic research library

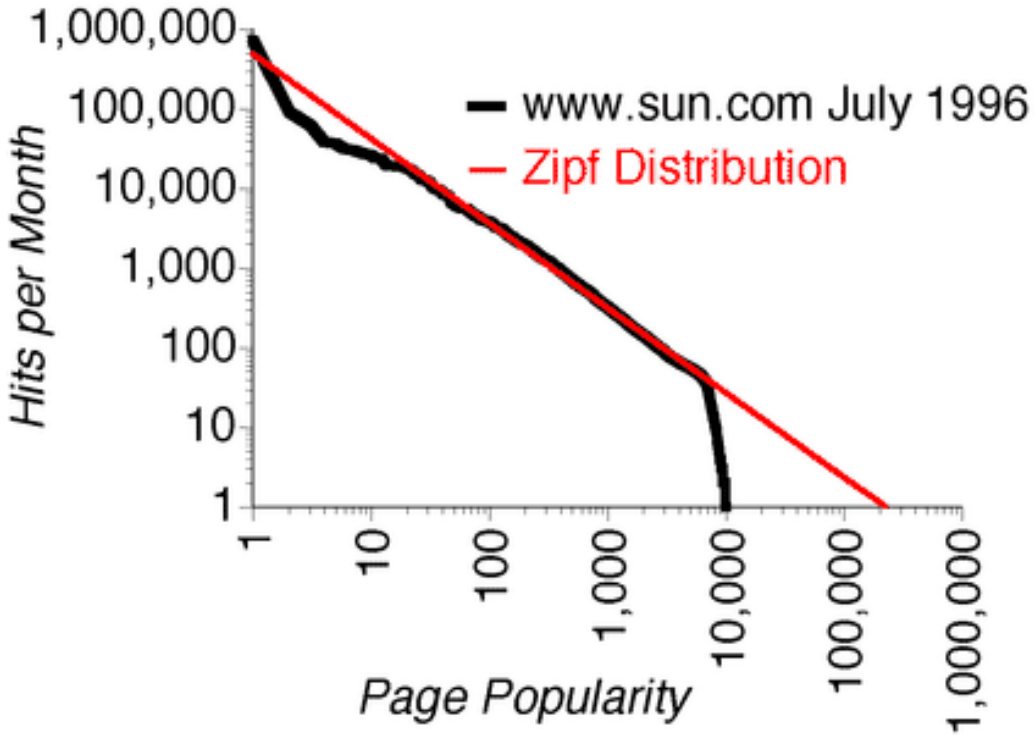
Graph Structure of the Web



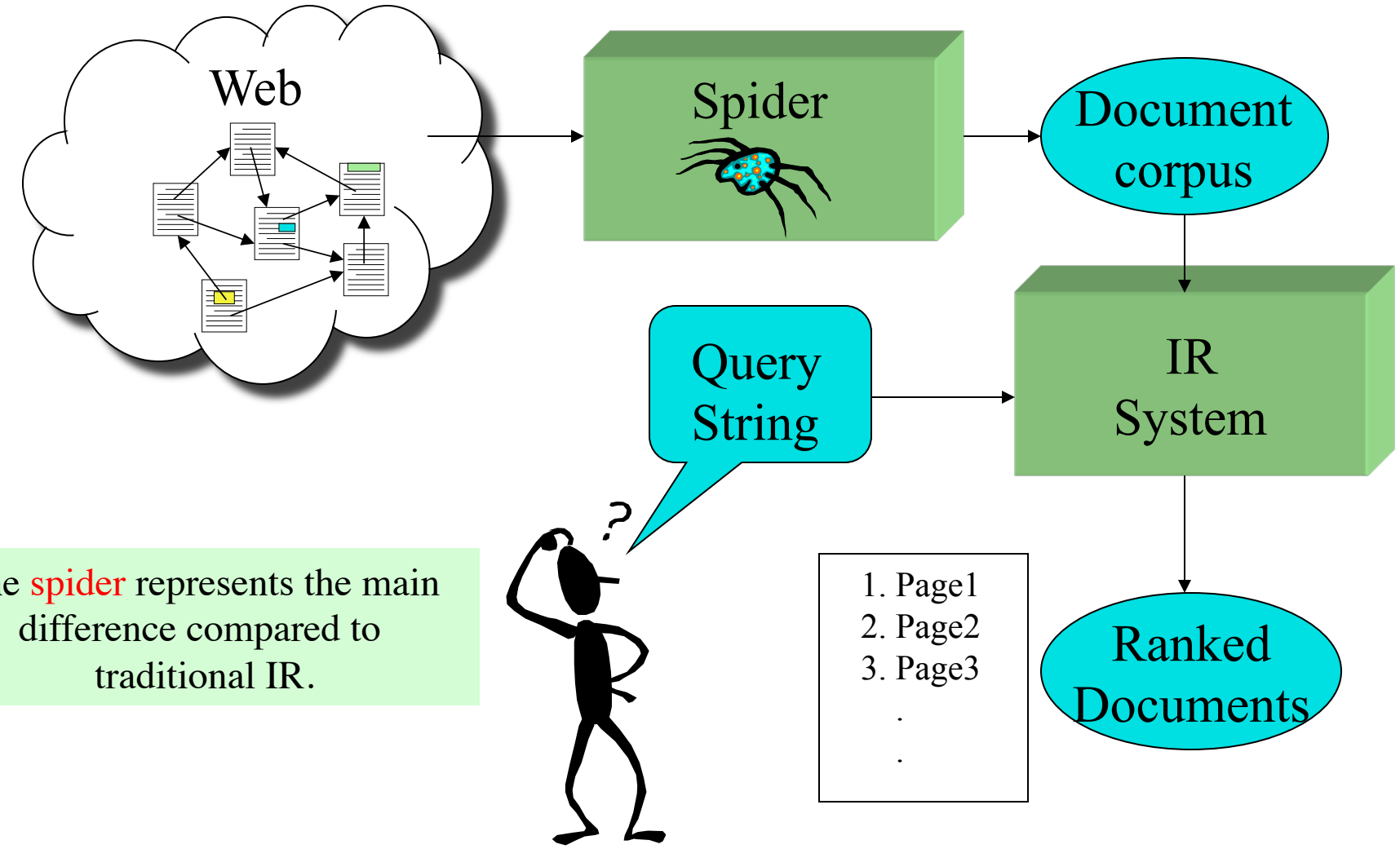
http://www9.org/w9cdrom/160/160.html

Zipf's Law on the Web

- Number of in-links/out-links to/from a page has a Zipfian distribution.
- Length of web pages has a Zipfian distribution.
- Number of hits to a web page has a Zipfian distribution.



Web Search Using IR



The spider represents the main difference compared to traditional IR.

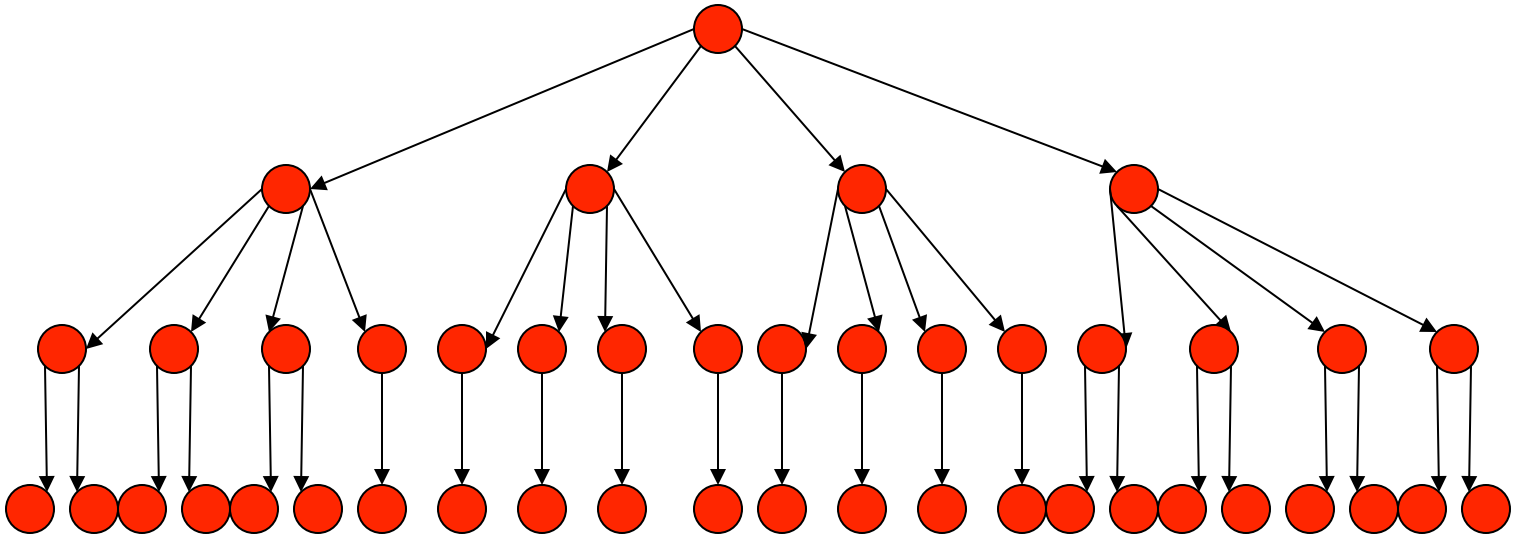
Spiders (Robots/Bots/Crawlers)

- Start with a comprehensive set of root URL's from which to start the search.
- Follow all links on these pages recursively to find additional pages.
- Index/Process all **novel** found pages in an inverted index as they are encountered.
- May allow users to directly submit pages to be indexed (and crawled from).

  – You'll need to build a simple spider for Assignment 2 to traverse the OU webpages.

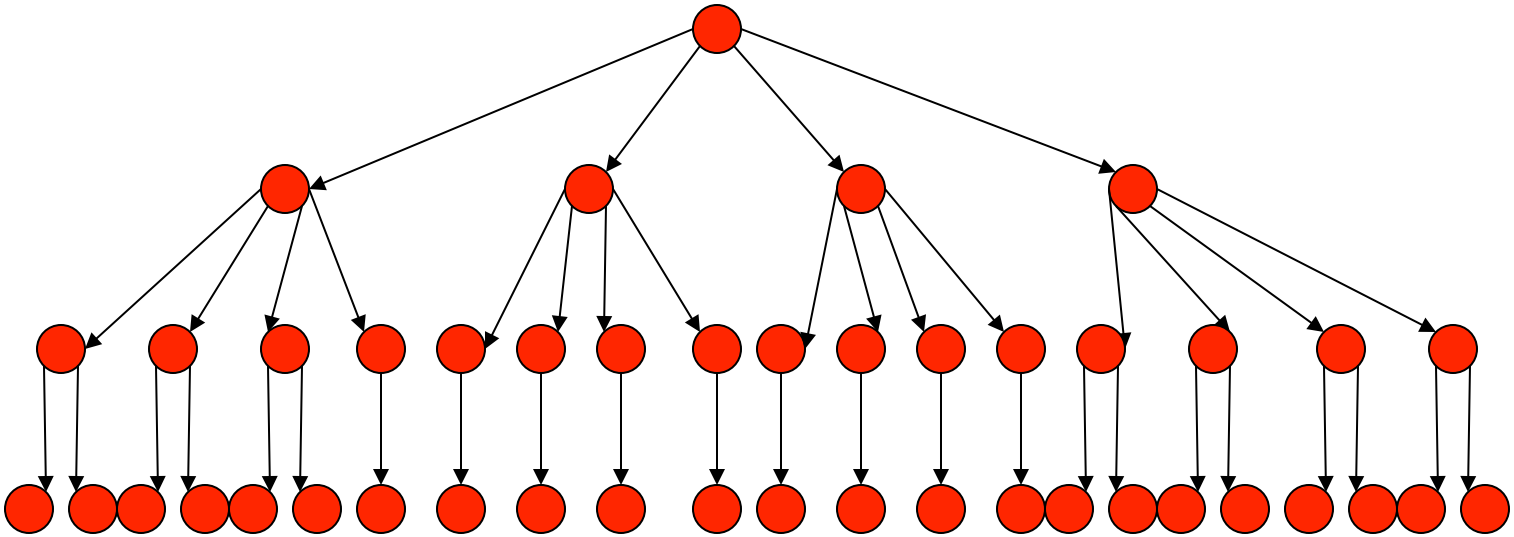
Search Strategies

Breadth-first Search



Search Strategies (cont)

Depth-first Search



Search Strategy Trade-Off's

- Breadth-first explores uniformly outward from the root page but requires memory of all nodes on the previous level (exponential in depth). Standard spidering method.
- Depth-first requires memory of only depth times branching-factor (linear in depth) but gets "lost" pursuing a single thread.
- Both strategies can be easily implemented using a queue of links (URL's).

Avoiding Page Duplication

- Must detect when revisiting a page that has already been spidered (web is a graph not a tree).
- Must efficiently index visited pages to allow rapid recognition test.
  - Tree indexing (e.g. trie)
  - Hashtable
- Index page using URL as a key.
  - Must canonicalize URL's (e.g. delete ending "/")
  - Cannot detect duplicated or mirrored pages.
- Index page using textual content as a key.
  - Requires first downloading page.

Duplicate & Near-Duplicate Detection

- The web is full of duplicated content.
- Strict duplicates are not that common:
  - exact match can be detected using *fingerprinting*.
- Near duplicates are much more common:
  - Example: last modified date the only difference between two copies of a page.
  - Efficient detection using a randomized algorithm called *shingling*:
    - Shingles are word n-grams:
      - ***a rose is a rose is a rose*** → 4-grams are
        - a_rose_is_a, rose_is_a_rose, is_a_rose_is, a_rose_is_a
    - Use Jaccard similarity between 2 docs as sets of shingles:
      - Size_of_Intersection / Size_of_Union.
    - Efficient approximation using a *sketch* of shingles from each document:
      - More details on randomized algorithm in IIR 19.6.

Spidering Algorithm

Initialize queue (Q) with initial set of known URL's.
Until Q empty or page or time limit exhausted:
    Pop URL, L, from front of Q.
    If L is not to an HTML page (.gif, .jpeg, .ps, .pdf, .ppt…)
        continue loop.
    If already visited L, continue loop.
    Download page, P, for L.
    If cannot download P (e.g. 404 error, robot excluded)
        continue loop.
    Index P (e.g. add to inverted index or store cached copy).
    Parse P to obtain list of new links N.
    Append N to the end of Q.

Queueing Strategy

- How new links are added to the queue determines search strategy.
- FIFO (append to end of Q) gives breadth-first search.
- LIFO (add to front of Q) gives depth-first search.
- Heuristically ordering the Q gives a “focused crawler” that directs its search towards “interesting” pages.

Restricting Spidering

- You can restrict spider to a particular site.
  - Remove links to other sites from Q.
- You can restrict spider to a particular directory.
  - Remove links not in the specified directory.
- Explicit politeness:
  - Obey page-owner restrictions (robot exclusion).
- Implicit politeness:
  - Avoid hitting same site too often.

Implicit Politeness

- The bandwidth available for a crawler is usually much higher than the bandwidth of the Web sites it visits.
- Using multiple threads, a Web crawler might easily overload a Web server, specially a smaller one.
- To avoid this, it is customary:
  - to open only one connection to a given Web server at a time.
  - to take a delay between two consecutive accesses:
    - Common heuristic: insert time gap between successive requests to a host that is >> time for most recent fetch from that host.
    - [Cho et al.] suggested adopting 10 seconds as the interval between consecutive accesses

Link Extraction

- Must find all links in a page and extract URLs.
  - ...
  - ...
- Must complete relative URL's using current page URL:
  - to http://ace.cs.ohio.edu/~razvan/courses/ir6900/hw02.pdf
  - to http://ace.cs.ohio.edu/~razvan/courses/cs3200/index.html

URL Syntax

- A URL has the following syntax:
  - <scheme>://<authority><path>?<query>#<fragment>
- A *query* passes variable values from an HTML form and has the syntax:
  - <variable>=<value>&<variable>=<value>…
- A *fragment* is also called a *reference* or a *ref* and is a pointer within the document to a point specified by an anchor tag of the form:
  - <A NAME="<fragment>">

Link Canonicalization

- Equivalent variations of ending directory normalized by removing ending slash.
  - http://ace.cs.ohio.edu/~razvan/
  - http://ace.cs.ohio.edu/~razvan
- Internal page fragments (ref' s) removed:
  - http://nltk.org/book/ch03.html#chap-words
  - http://nltk.org/book/ch03.html

Anchor Text Indexing

- Extract anchor text (between <a> and) of each link followed.
- Anchor text is usually descriptive of the document to which it points.
- Add anchor text to the content of the destination page to provide additional relevant keyword indices.
- Used by Google:
  - Evil Empire
  - IBM

Anchor Text Indexing (cont' d)

- Helps when descriptive text in destination page is embedded in image logos rather than in accessible text.
- Many times anchor text is not useful:
  - "click here"
- Increases content more for popular pages with many in-coming links, increasing recall of these pages.
- May even give higher weights to tokens from anchor text.

Robot Exclusion

- Web sites and pages can specify that robots should not crawl/index certain areas.
- Two components:
  - Robots Exclusion Protocol: Site wide specification of excluded directories.
  - Robots META Tag: Individual document tag to exclude indexing or following links.
- http://www.robotstxt.org/orig.html

Robots Exclusion Protocol

- Site administrator puts a "robots.txt" file at the root of the host's web directory.
  - http://www.ebay.com/robots.txt
  - http://www.cnn.com/robots.txt
- File is a list of excluded directories for a given robot (user-agent).
  - Exclude all robots from the entire site:

```
User-agent: *
Disallow: /
```

Robot Exclusion Protocol Examples

- Exclude specific directories:

```
User-agent: *
Disallow: /tmp/
Disallow: /cgi-bin/
Disallow: /users/paranoid/
```

- Exclude a specific robot:

```
User-agent: GoogleBot
Disallow: /
```

- Allow a specific robot:

```
User-agent: GoogleBot
Disallow:
```

Robot Exclusion Protocol Details

- Only use blank lines to separate different User-agent disallowed directories.
- One directory per “Disallow” line.
- No regex patterns in directories.

Robots META Tag

- Include META tag in HEAD section of a specific HTML document.
  - <meta name="robots" content="none">
- Content value is a pair of values for two aspects:
  - index | noindex: Allow/disallow indexing of this page.
  - follow | nofollow: Allow/disallow following links on this page.

Robots META Tag (cont)

- Special values:
  - all = index,follow
  - none = noindex,nofollow
- Examples:

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

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

<meta name="robots" content="none">

Robot Exclusion Issues

- META tag is newer and less well-adopted than "robots.txt".
- Standards are conventions to be followed by "good robots."
- Companies have been prosecuted for "disobeying" these conventions and "trespassing" on private cyberspace.

Multi-Threaded Spidering

- Bottleneck is network delay in downloading individual pages.
- Best to have multiple threads running in parallel each requesting a page from a different host.
- Distribute URL's to threads to guarantee equitable distribution of requests across different hosts to maximize through-put and avoid overloading any single server.
- Early Google spider had multiple co-ordinated crawlers with about 300 threads each, together able to download over 100 pages per second.

Directed/Focused Spidering

- Sort queue to explore more “interesting” pages first.
- Two styles of focus:
  - Topic-Directed
  - Link-Directed

Topic-Directed Spidering

- Assume desired topic description or sample pages of interest are given.
- Sort queue of links by the similarity (e.g. cosine metric) of their source pages and/or anchor text to this topic description.
  - Related to Topic Tracking and Detection

Link-Directed Spidering

- Monitor links and keep track of in-degree and out-degree of each page encountered.
- Sort queue to prefer popular pages with many in-coming links (*authorities*).
- Sort queue to prefer summary pages with many out-going links (*hubs*).
  - Google's PageRank algorithm.

Keeping Spidered Pages Up to Date

- Web is very dynamic: many new pages, updated pages, deleted pages, etc.
- Periodically check spidered pages for updates and deletions:
  - Just look at header info (e.g. META tags on last update) to determine if page has changed, only reload entire page if needed.
- Track how often each page is updated and preferentially return to pages which are historically more dynamic.
- Preferentially update pages that are accessed more often to optimize freshness of more popular pages.

Web Crawling in Python

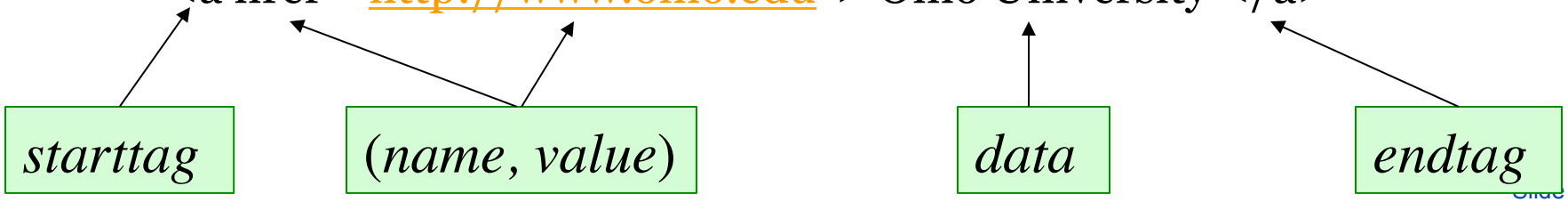
Extracting links from HTML documents:

1) Via regular expressions.
2) Via the HTMLParse class from the HTMLParse module:
   - Event based parser:
     - Scans through the document, and whenever finds an html tag, it generates an event and calls a predefined handler function.
   - Flexible, customizable:
     - We can overwrite handler functions, by subclassing.
   - We can extract both links and text content in one sweep:
     - For text content, can also use nltk.clean_html.

http://docs.python.org/2.7/library/htmlparser.html

HTMLParser: Event Handlers

- HTMLParser.handle_starttag(self, tag, attrs):
  - This method is called to handle the start of a tag.
    - The attrs argument is a list of (name, value) pairs.
- HTMLParser.handle_endtag(tag):
  - This method is called to handle the start of a tag.
- HTMLParser.handle_data(data)
  - This method is called to process arbitrary data (e.g. text nodes and the content of <script>...</script> and <style>...</style>)
- Example:
  - Ohio University



Extracting links from HTML in Python

```
from HTMLParser import HTMLParser

class MyHTMLParser(HTMLParser):
    def __init__(self):
        HTMLParser.__init__(self)
        self.links = []
    def handle_starttag(self, tag, attrs):
        if tag == 'a':
            for (name, value) in attrs:
                if name == 'href':
                    self.links.append(value)
                break
```

Add code to this class to also extract anchor text for each link.

Normalizing HTML links in Python

```
from urllib import urlopen
from MyHTMLParser import MyHTMLParser
from urlparse import urljoin

parser = MyHTMLParser()
url = "http://nltk.org/book/ch01.html"
parser.feed(urlopen(url).read())

absolutes = [urljoin(url, link) for link in parser.links]
print absolutes
```

http://docs.python.org/2/library/urlparse.html

RobotFileParser: parser for robots.txt

- The module robotparser provides a single class, RobotFileParser, which answers questions about whether or not a particular user agent can fetch a URL on the Web site that published the robots.txt file.

```
>>> import robotparser
>>> rp = robotparser.RobotFileParser()
>>> rp.set_url("http://www.musi-cal.com/robots.txt")
>>> rp.read()
>>> rp.can_fetch(”*", "www.ohio.edu")
True
```

Open Source Web Crawlers

- NUTCH is an open-source crawler written in Java that is part of the Lucene search engine:
  - It is sponsored by the Apache Foundation.
  - It includes a simple interface for intranet Web crawling as well as a more powerful set of commands for large-scale crawl.
- WIRE is an open-source web crawler written in C++:
  - Includes several policies for scheduling the page downloads.
  - Also includes a module for generating reports and statistics on the downloaded pages.
  - It has been used for Web characterization.
- Other crawlers described in the literature include:
  - ht://Dig (in C++), WebBase (in C), CobWeb (in Perl), PolyBot (in C++ and Python), and WebRace (in Java).