Spiders, crawlers, harvesters, bots

Thanks to

B. Arms

R. Mooney

P. Baldi

P. Frasconi

P. Smyth

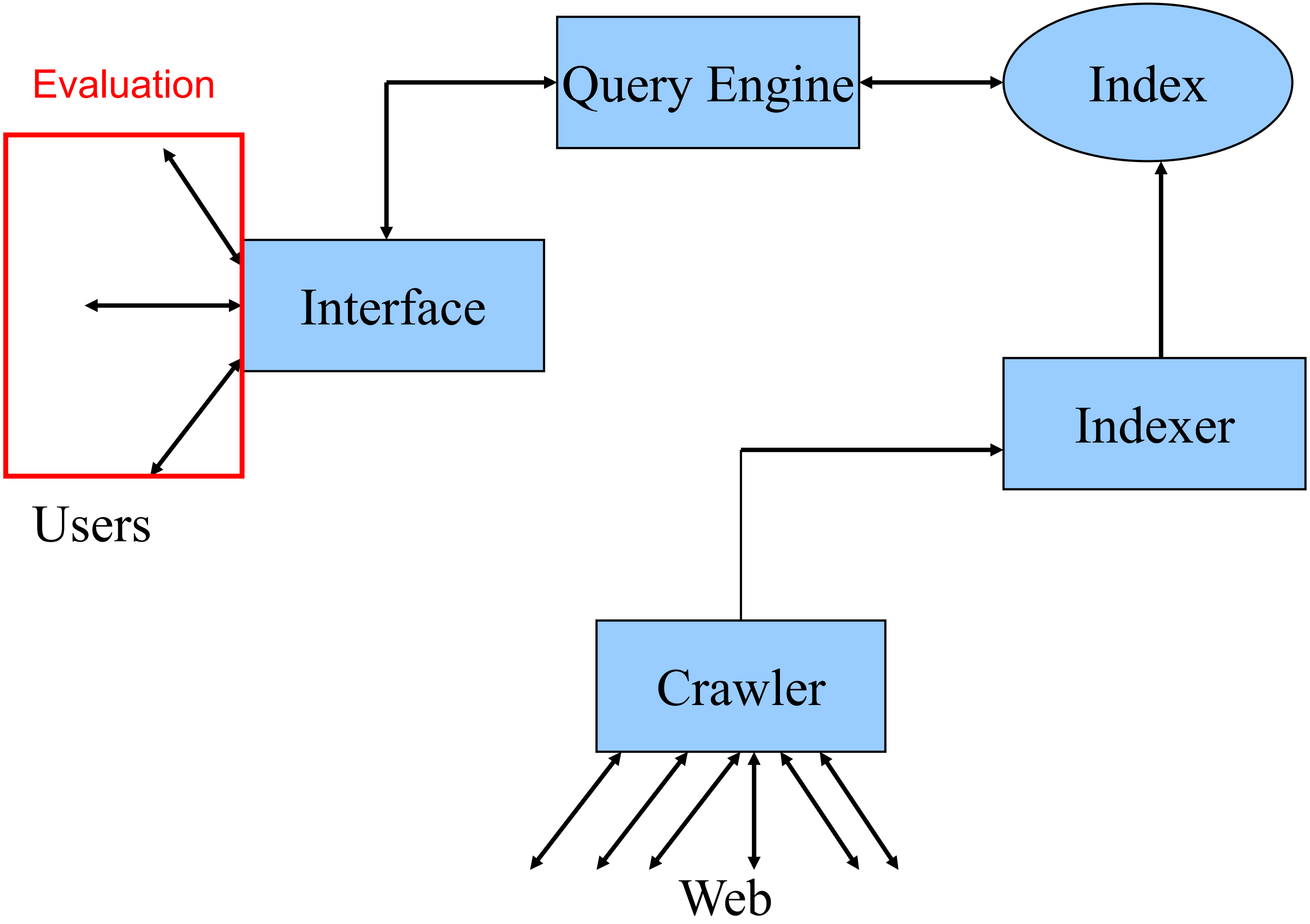
C. Manning

Last time

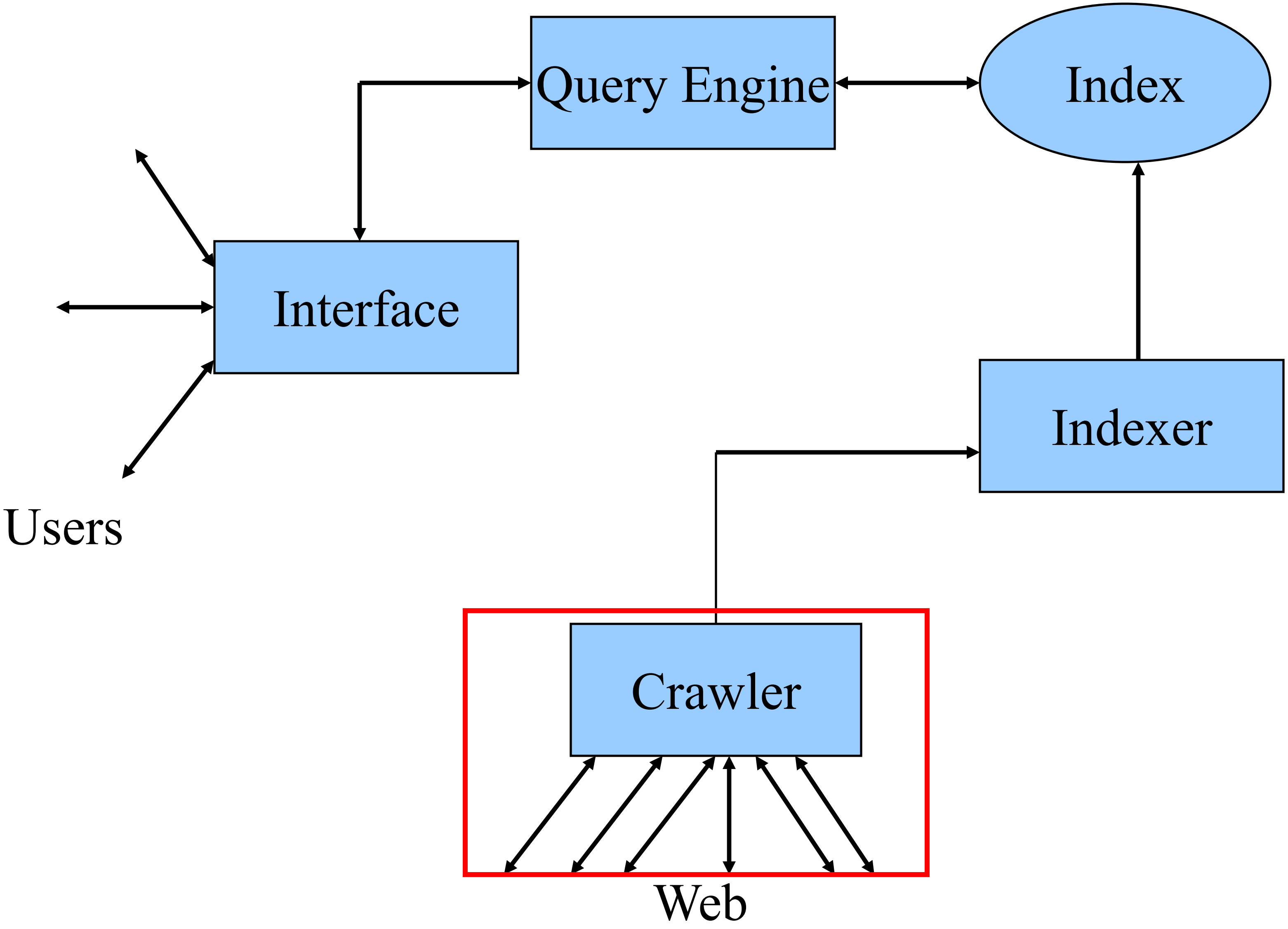
- Evaluation of IR/Search systems
  - Quality of evaluation – Relevance
  - Evaluation is empirical
  - Measurements of Evaluation
    - Precision vs recall
    - F measure
  - Test Collections/TREC

This time

- Web crawlers
- Crawler policy
- Robots.txt
- Scrapy



A Typical Web Search Engine



A Typical Web Search Engine

What is a Web Crawler?

- **The Web crawler is a foundational species!**
- Without crawlers, search engines would not exist.
  - But they get little credit!
- Outline:

  What is a crawler

  How they work

  How they are controlled

  Robots.txt

  Issues of performance

  Research

What a web crawler does

- Gets data!!!
- Can get fresh data.
- Gets data for search engines:
  - Creates and repopulates search engines data by navigating the web, downloading documents and files
  - Follows hyperlinks from a crawl list and hyperlinks in the list
  - Without a crawler, there would be nothing to search

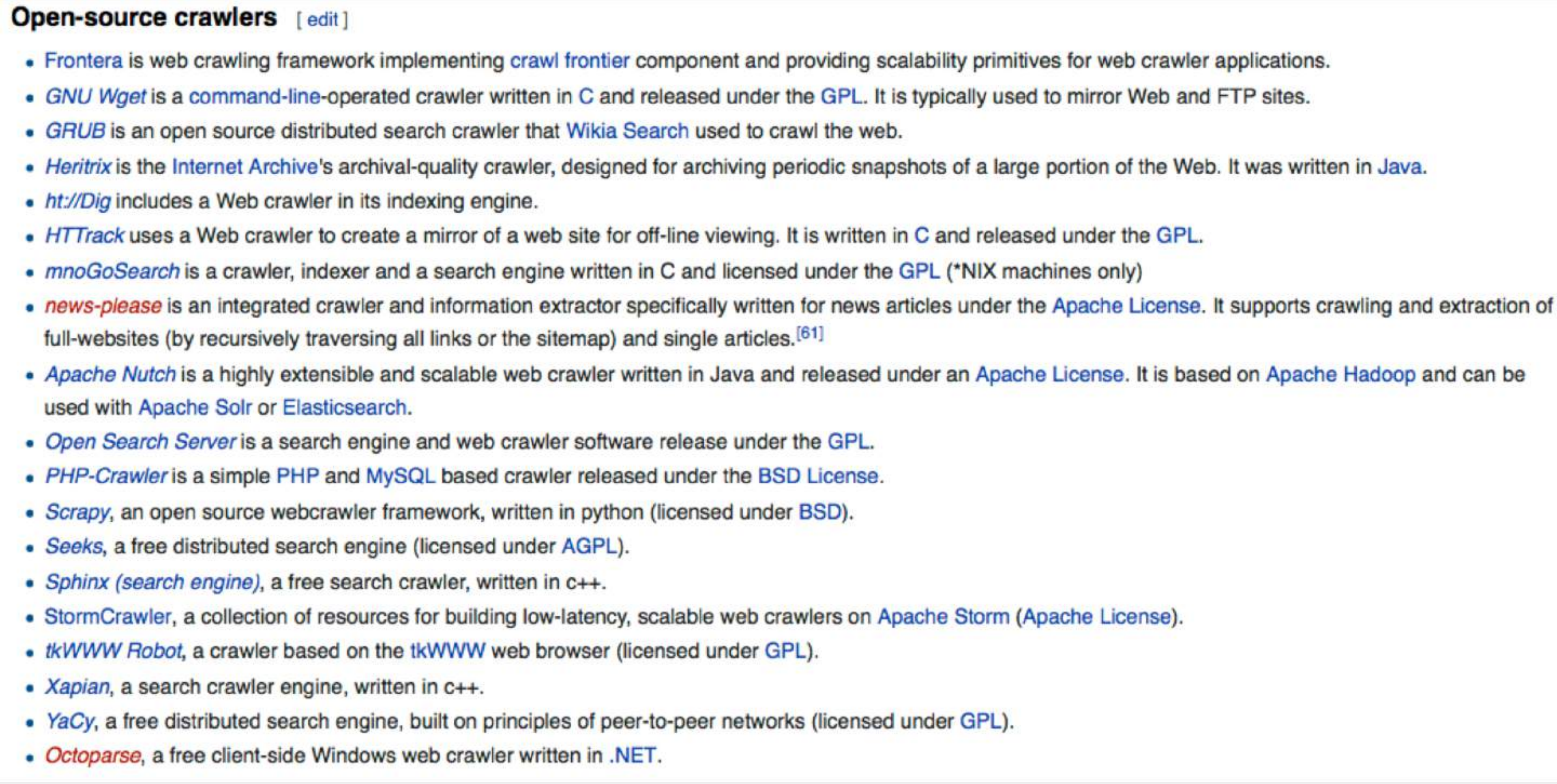
Web crawler policies

- The behavior of a Web crawler is the outcome of a combination of policies:
    - a selection policy that states which pages to download,
    - a re-visit policy that states when to check for changes to the pages,
    - a duplication policy
    - a politeness policy that states how to avoid overloading Web sites, and
    - a parallelization policy that states how to coordinate distributed Web crawlers.

Crawlers vs Browsers vs Scrapers

- Crawlers automatically harvest all files on the web
- Browsers are manual crawlers
- Web Scrapers automatically harvest the visual files for a web site, are manually directed, and are limited crawlers (sometimes called “screen scrapers”)

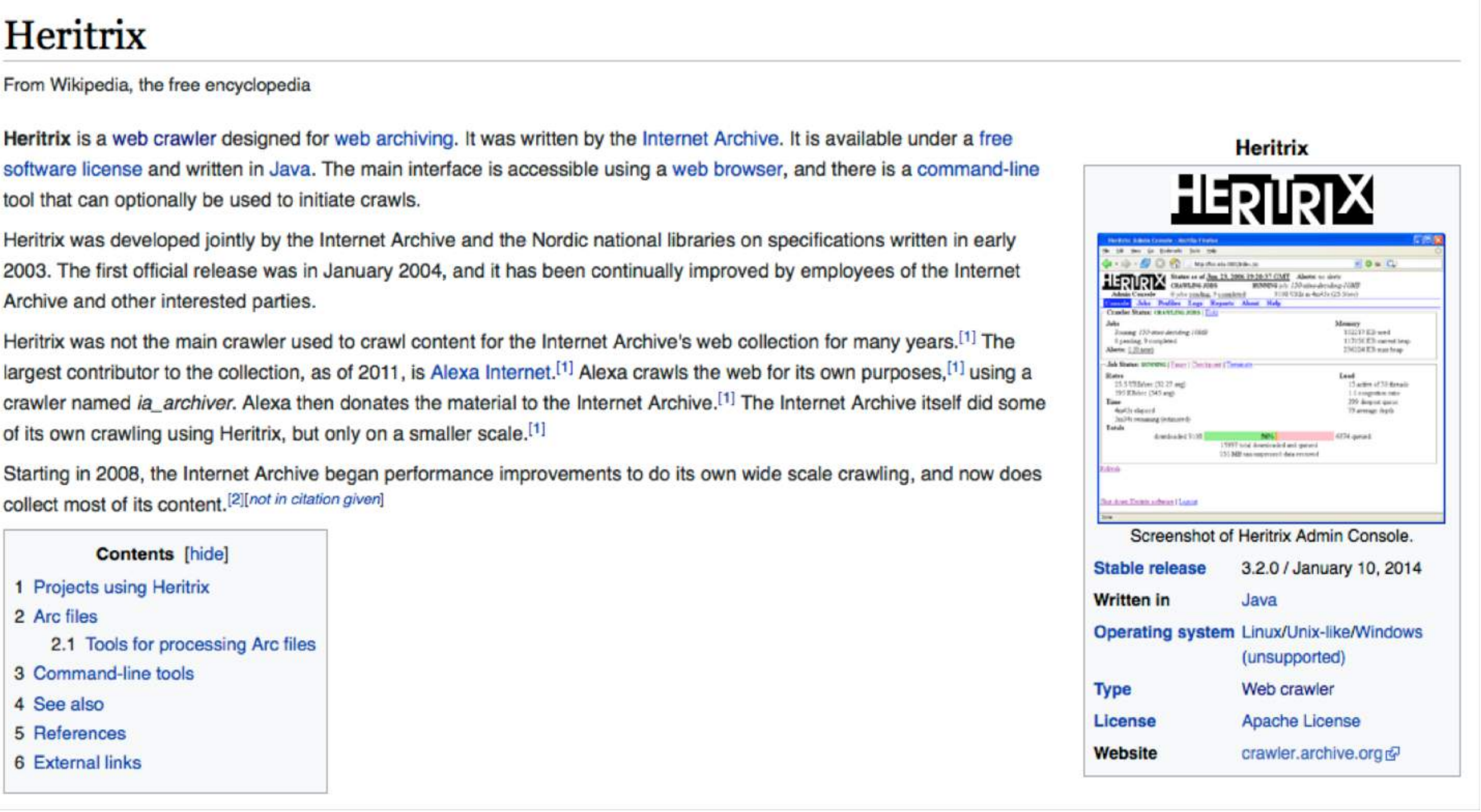
Open source crawlers



Open-source crawlers [edit]

- Frontera is web crawling framework implementing crawl frontier component and providing scalability primitives for web crawler applications.
- *GNU Wget* is a command-line-operated crawler written in C and released under the GPL. It is typically used to mirror Web and FTP sites.
- *GRUB* is an open source distributed search crawler that Wikia Search used to crawl the web.
- *Heritrix* is the Internet Archive's archival-quality crawler, designed for archiving periodic snapshots of a large portion of the Web. It was written in Java.
- *ht://Dig* includes a Web crawler in its indexing engine.
- *HTTrack* uses a Web crawler to create a mirror of a web site for off-line viewing. It is written in C and released under the GPL.
- *mnoGoSearch* is a crawler, indexer and a search engine written in C and licensed under the GPL (*NIX machines only)
- *news-please* is an integrated crawler and information extractor specifically written for news articles under the Apache License. It supports crawling and extraction of full-websites (by recursively traversing all links or the sitemap) and single articles.[61]
- *Apache Nutch* is a highly extensible and scalable web crawler written in Java and released under an Apache License. It is based on Apache Hadoop and can be used with Apache Solr or Elasticsearch.
- *Open Search Server* is a search engine and web crawler software release under the GPL.
- *PHP-Crawler* is a simple PHP and MySQL based crawler released under the BSD License.
- *Scrapy*, an open source webcrawler framework, written in python (licensed under BSD).
- *Seeks*, a free distributed search engine (licensed under AGPL).
- *Sphinx (search engine)*, a free search crawler, written in c++.
- StormCrawler, a collection of resources for building low-latency, scalable web crawlers on Apache Storm (Apache License).
- *tkWWW Robot*, a crawler based on the tkWWW web browser (licensed under GPL).
- *Xapian*, a search crawler engine, written in c++.
- *YaCy*, a free distributed search engine, built on principles of peer-to-peer networks (licensed under GPL).
- *Octoparse*, a free client-side Windows web crawler written in .NET.

Heritrix



Heritrix

From Wikipedia, the free encyclopedia

Heritrix is a web crawler designed for web archiving. It was written by the Internet Archive. It is available under a free software license and written in Java. The main interface is accessible using a web browser, and there is a command-line tool that can optionally be used to initiate crawls.

Heritrix was developed jointly by the Internet Archive and the Nordic national libraries on specifications written in early 2003. The first official release was in January 2004, and it has been continually improved by employees of the Internet Archive and other interested parties.

Heritrix was not the main crawler used to crawl content for the Internet Archive's web collection for many years.[1] The largest contributor to the collection, as of 2011, is Alexa Internet.[1] Alexa crawls the web for its own purposes,[1] using a crawler named *ia_archiver*. Alexa then donates the material to the Internet Archive.[1] The Internet Archive itself did some of its own crawling using Heritrix, but only on a smaller scale.[1]

Starting in 2008, the Internet Archive began performance improvements to do its own wide scale crawling, and now does collect most of its content.[2][*not in citation given*]

Contents [hide]

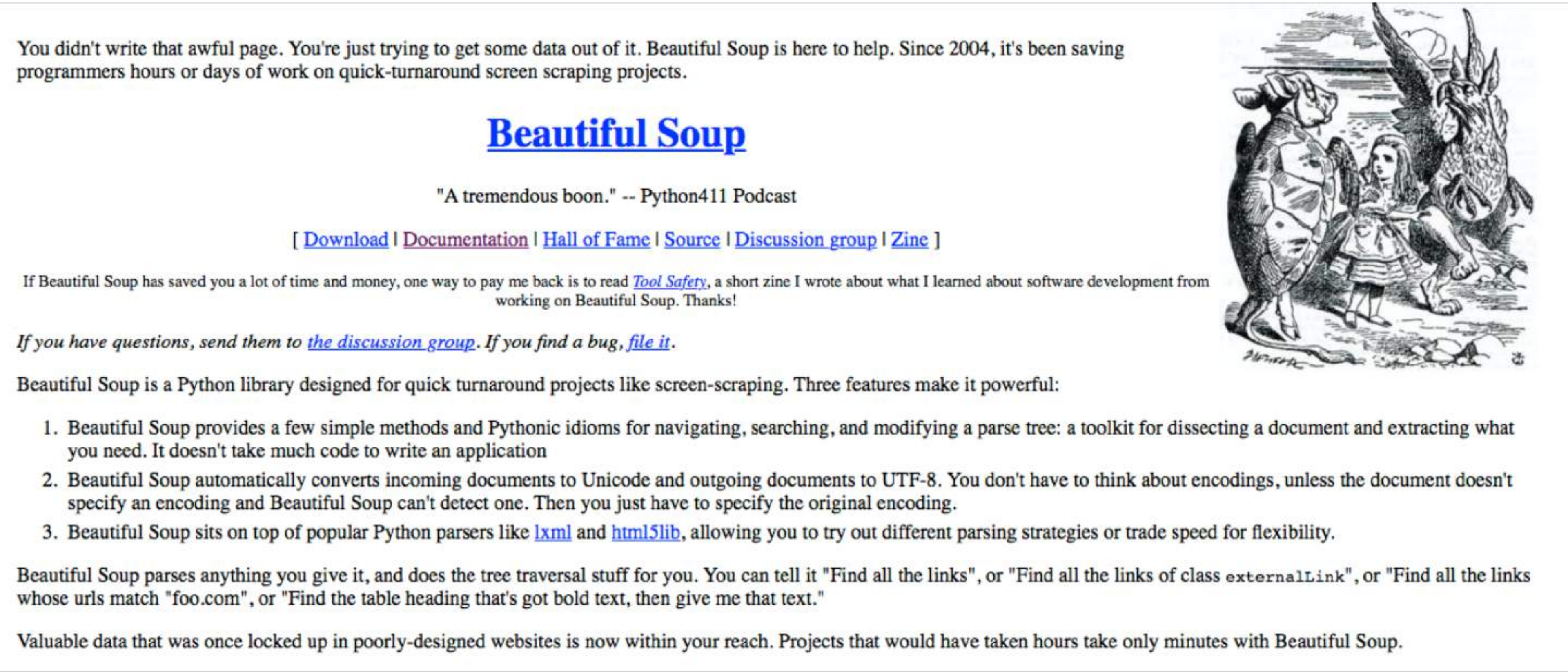
1 Projects using Heritrix
2 Arc files
2.1 Tools for processing Arc files
3 Command-line tools
4 See also
5 References
6 External links

Heritrix

Screenshot of Heritrix Admin Console.

Stable release	3.2.0 / January 10, 2014
Written in	Java
Operating system	Linux/Unix-like/Windows (unsupported)
Type	Web crawler
License	Apache License
Website	crawler.archive.org

Beautiful Soup – scraper



You didn't write that awful page. You're just trying to get some data out of it. Beautiful Soup is here to help. Since 2004, it's been saving programmers hours or days of work on quick-turnaround screen scraping projects.

Beautiful Soup

"A tremendous boon." -- Python411 Podcast

[Download | Documentation | Hall of Fame | Source | Discussion group | Zine]

If Beautiful Soup has saved you a lot of time and money, one way to pay me back is to read *Tool Safety*, a short zine I wrote about what I learned about software development from working on Beautiful Soup. Thanks!

If you have questions, send them to the discussion group. If you find a bug, file it.

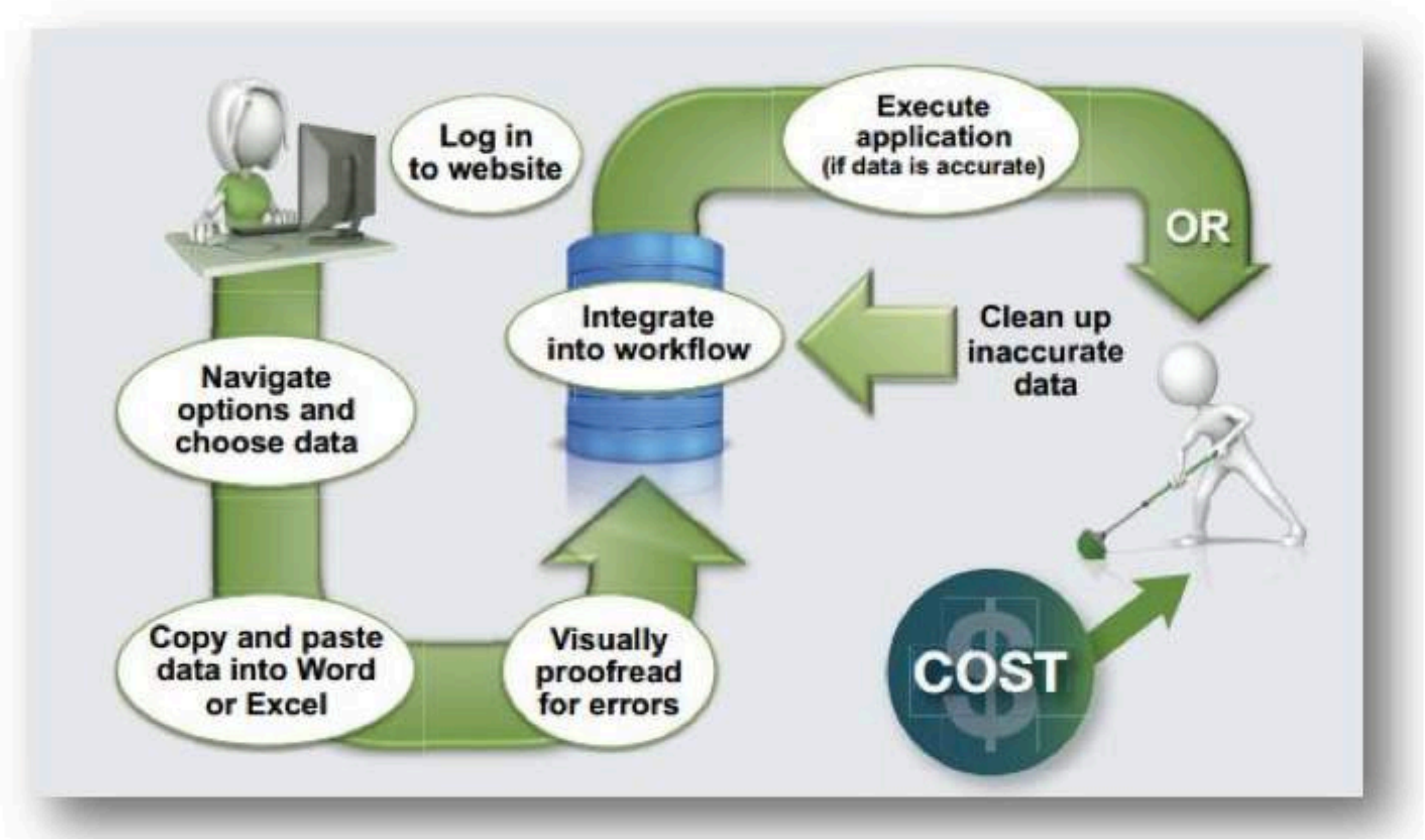
Beautiful Soup is a Python library designed for quick turnaround projects like screen-scraping. Three features make it powerful:

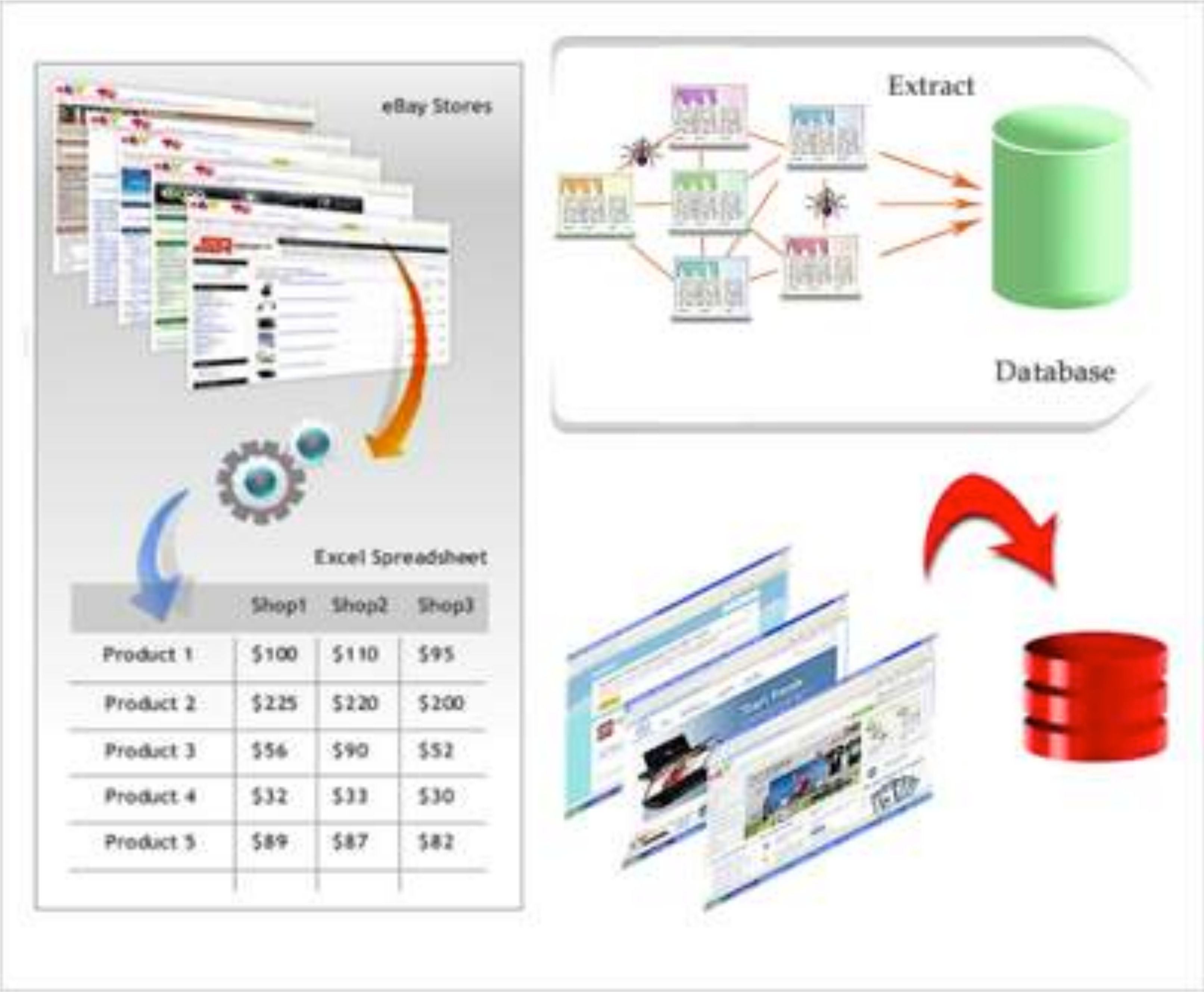
1. Beautiful Soup provides a few simple methods and Pythonic idioms for navigating, searching, and modifying a parse tree: a toolkit for dissecting a document and extracting what you need. It doesn't take much code to write an application
2. Beautiful Soup automatically converts incoming documents to Unicode and outgoing documents to UTF-8. You don't have to think about encodings, unless the document doesn't specify an encoding and Beautiful Soup can't detect one. Then you just have to specify the original encoding.
3. Beautiful Soup sits on top of popular Python parsers like lxml and html5lib, allowing you to try out different parsing strategies or trade speed for flexibility.

Beautiful Soup parses anything you give it, and does the tree traversal stuff for you. You can tell it "Find all the links", or "Find all the links of class `externalLink`", or "Find all the links whose urls match "foo.com", or "Find the table heading that's got bold text, then give me that text."

Valuable data that was once locked up in poorly-designed websites is now within your reach. Projects that would have taken hours take only minutes with Beautiful Soup.

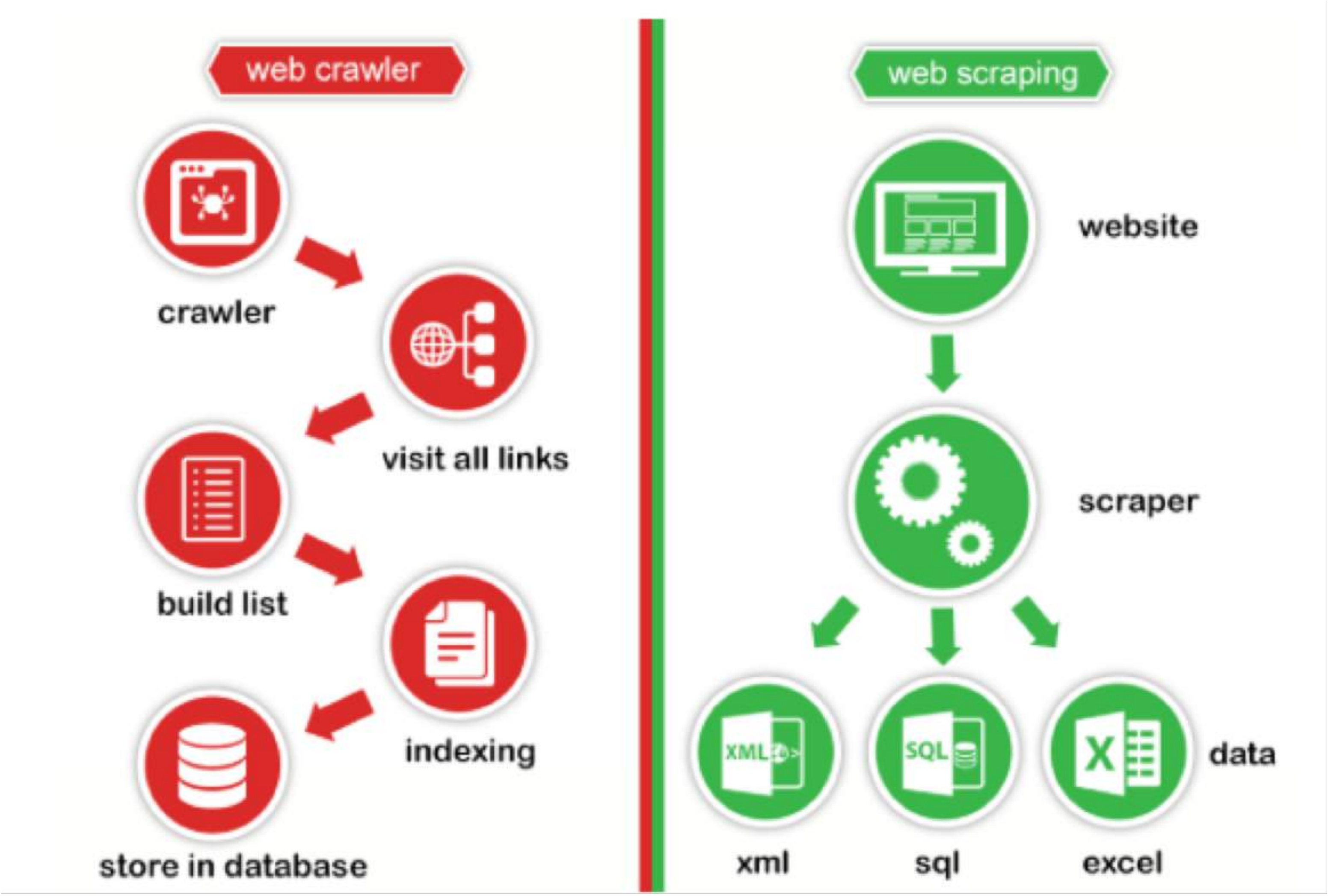
Why use a scrapper


Log in to website
Navigate options and choose data
Copy and paste data into Word or Excel
Visually proofread for errors
Integrate into workflow
Execute application (if data is accurate)
OR
Clean up inaccurate data
COST

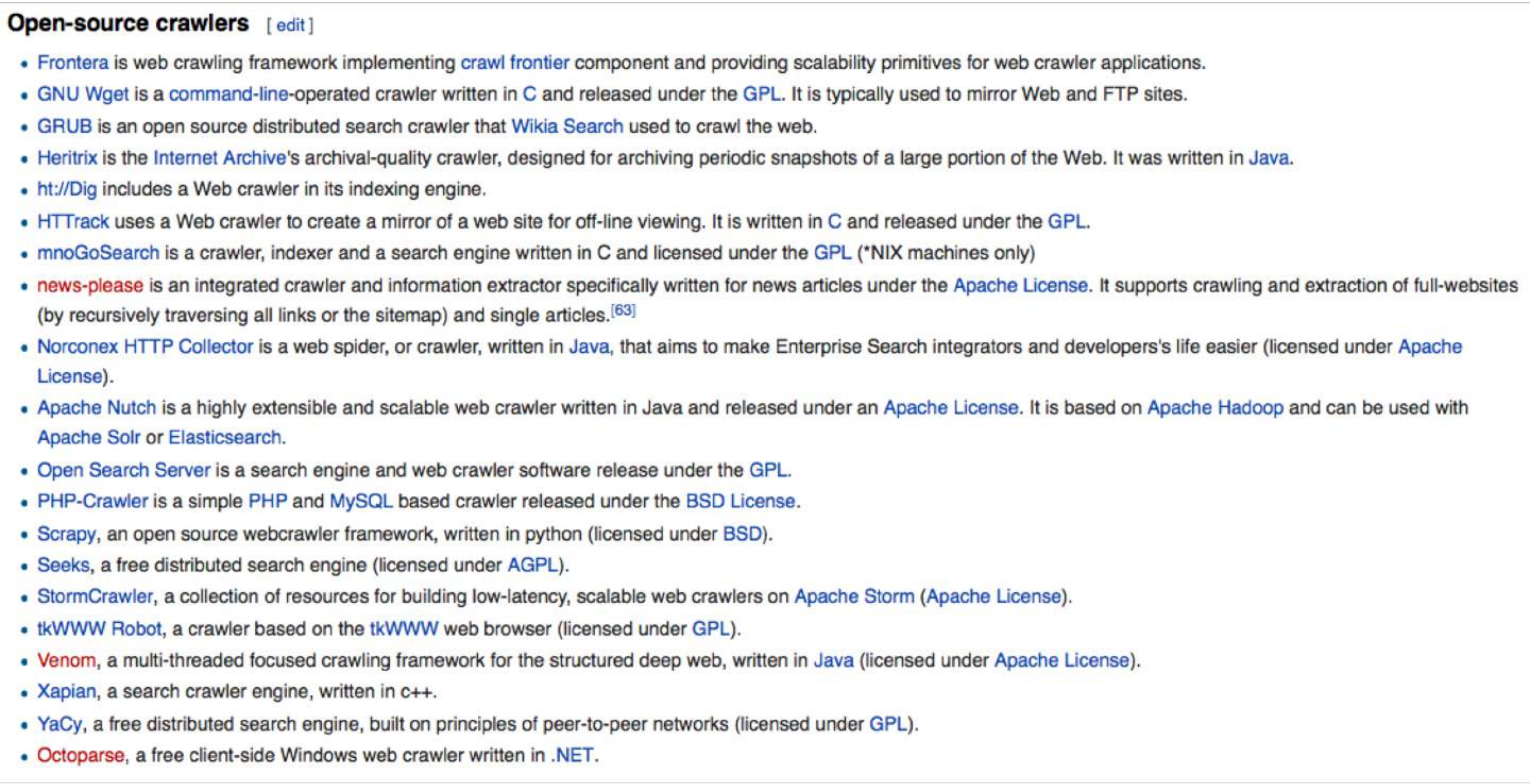


	Shop1	Shop2	Shop3
Product 1	$100	$110	$95
Product 2	$225	$220	$200
Product 3	$56	$90	$52
Product 4	$32	$33	$30
Product 5	$89	$87	$82

Web Crawler vs web scraper



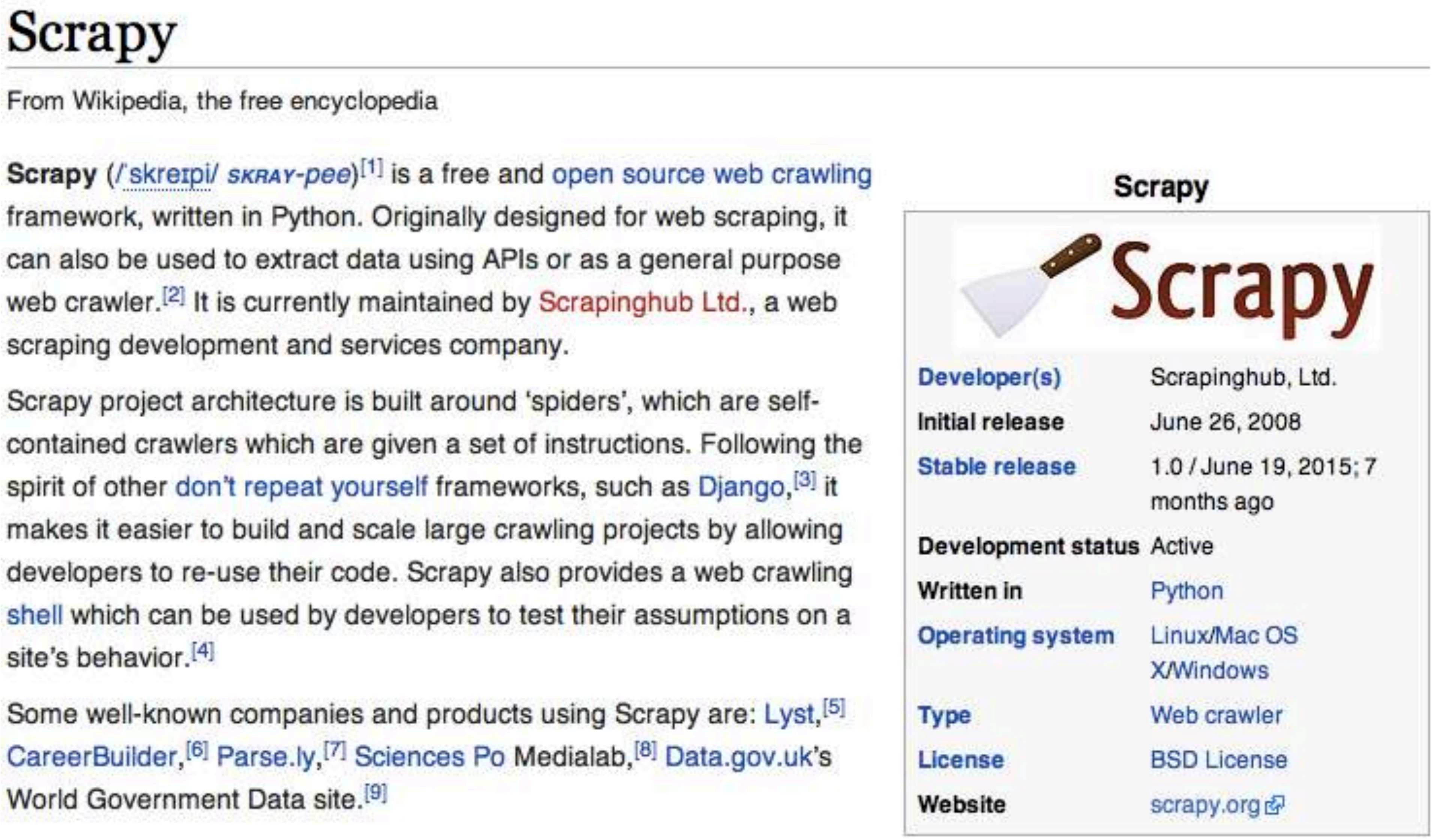
Open source crawlers



Open-source crawlers [edit]

- Frontera is web crawling framework implementing crawl frontier component and providing scalability primitives for web crawler applications.
- GNU Wget is a command-line-operated crawler written in C and released under the GPL. It is typically used to mirror Web and FTP sites.
- GRUB is an open source distributed search crawler that Wikia Search used to crawl the web.
- Heritrix is the Internet Archive's archival-quality crawler, designed for archiving periodic snapshots of a large portion of the Web. It was written in Java.
- ht://Dig includes a Web crawler in its indexing engine.
- HTTrack uses a Web crawler to create a mirror of a web site for off-line viewing. It is written in C and released under the GPL.
- mnoGoSearch is a crawler, indexer and a search engine written in C and licensed under the GPL (*NIX machines only)
- news-please is an integrated crawler and information extractor specifically written for news articles under the Apache License. It supports crawling and extraction of full-websites (by recursively traversing all links or the sitemap) and single articles.[63]
- Norconex HTTP Collector is a web spider, or crawler, written in Java, that aims to make Enterprise Search integrators and developers's life easier (licensed under Apache License).
- Apache Nutch is a highly extensible and scalable web crawler written in Java and released under an Apache License. It is based on Apache Hadoop and can be used with Apache Solr or Elasticsearch.
- Open Search Server is a search engine and web crawler software release under the GPL.
- PHP-Crawler is a simple PHP and MySQL based crawler released under the BSD License.
- Scrapy, an open source webcrawler framework, written in python (licensed under BSD).
- Seeks, a free distributed search engine (licensed under AGPL).
- StormCrawler, a collection of resources for building low-latency, scalable web crawlers on Apache Storm (Apache License).
- tkWWW Robot, a crawler based on the tkWWW web browser (licensed under GPL).
- Venom, a multi-threaded focused crawling framework for the structured deep web, written in Java (licensed under Apache License).
- Xapian, a search crawler engine, written in c++.
- YaCy, a free distributed search engine, built on principles of peer-to-peer networks (licensed under GPL).
- Octoparse, a free client-side Windows web crawler written in .NET.

Open source crawlers



Scrapy

From Wikipedia, the free encyclopedia

Scrapy (/ˈskreɪpi/ *SKRAY-pee*)[1] is a free and open source web crawling framework, written in Python. Originally designed for web scraping, it can also be used to extract data using APIs or as a general purpose web crawler.[2] It is currently maintained by Scrapinghub Ltd., a web scraping development and services company.

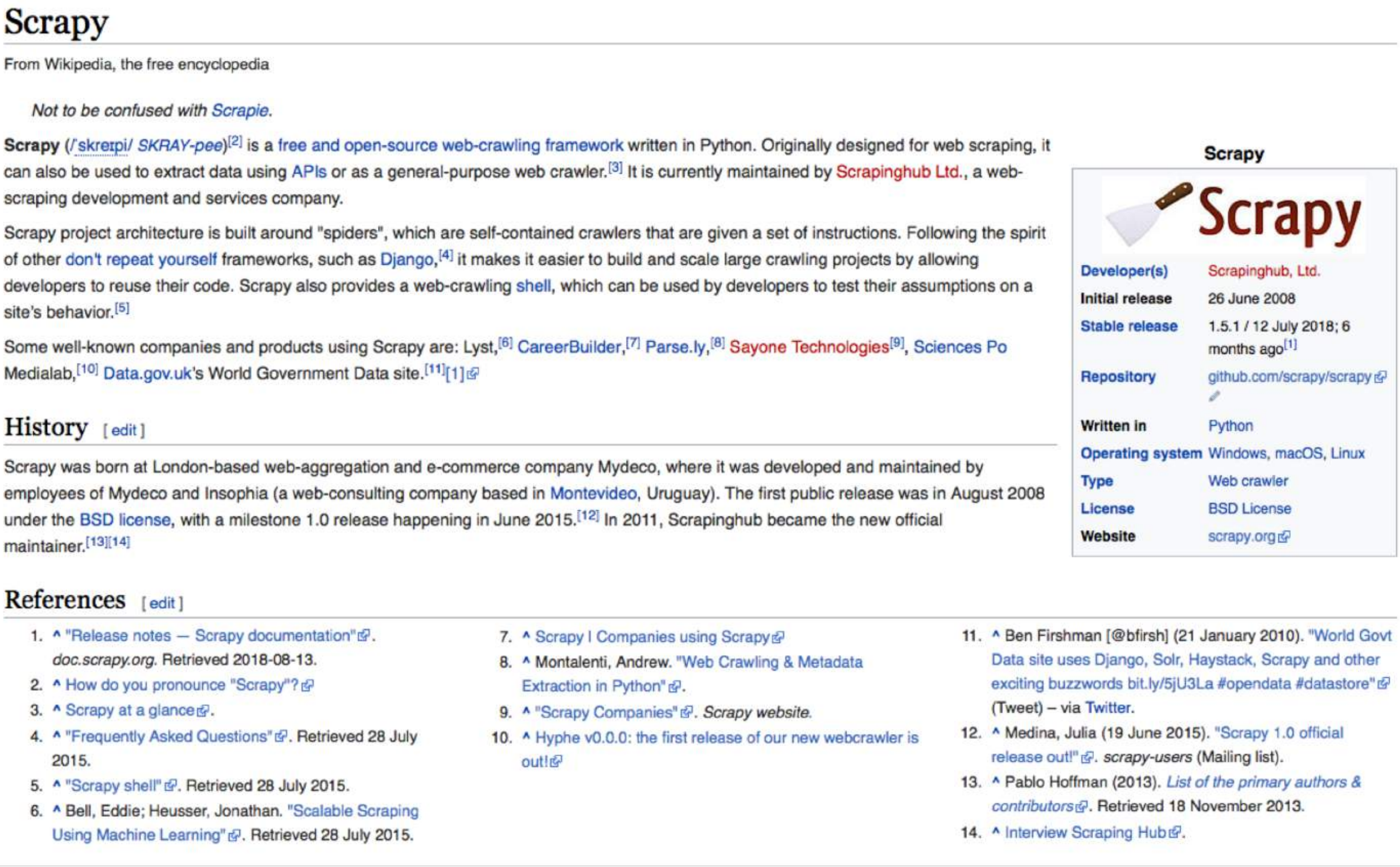
Scrapy project architecture is built around 'spiders', which are self-contained crawlers which are given a set of instructions. Following the spirit of other don't repeat yourself frameworks, such as Django,[3] it makes it easier to build and scale large crawling projects by allowing developers to re-use their code. Scrapy also provides a web crawling shell which can be used by developers to test their assumptions on a site's behavior.[4]

Some well-known companies and products using Scrapy are: Lyst,[5] CareerBuilder,[6] Parse.ly,[7] Sciences Po Medialab,[8] Data.gov.uk's World Government Data site.[9]

Scrapy

Developer(s)	Scrapinghub, Ltd.
Initial release	June 26, 2008
Stable release	1.0 / June 19, 2015; 7 months ago
Development status	Active
Written in	Python
Operating system	Linux/Mac OS X/Windows
Type	Web crawler
License	BSD License
Website	scrapy.org

Open source crawlers



Scrapy

From Wikipedia, the free encyclopedia

Not to be confused with Scrapie.

Scrapy (/ˈskreɪpi/ *SKRAY-pee*)[2] is a free and open-source web-crawling framework written in Python. Originally designed for web scraping, it can also be used to extract data using APIs or as a general-purpose web crawler.[3] It is currently maintained by Scrapinghub Ltd., a web-scraping development and services company.

Scrapy project architecture is built around "spiders", which are self-contained crawlers that are given a set of instructions. Following the spirit of other don't repeat yourself frameworks, such as Django,[4] it makes it easier to build and scale large crawling projects by allowing developers to reuse their code. Scrapy also provides a web-crawling shell, which can be used by developers to test their assumptions on a site's behavior.[5]

Some well-known companies and products using Scrapy are: Lyst,[6] CareerBuilder,[7] Parse.ly,[8] Sayone Technologies[9], Sciences Po Medialab,[10] Data.gov.uk's World Government Data site.[11][1]

Scrapy

Developer(s)	Scrapinghub, Ltd.
Initial release	26 June 2008
Stable release	1.5.1 / 12 July 2018; 6 months ago[1]
Repository	github.com/scrapy/scrapy
Written in	Python
Operating system	Windows, macOS, Linux
Type	Web crawler
License	BSD License
Website	scrapy.org

History [edit]

Scrapy was born at London-based web-aggregation and e-commerce company Mydeco, where it was developed and maintained by employees of Mydeco and Insophia (a web-consulting company based in Montevideo, Uruguay). The first public release was in August 2008 under the BSD license, with a milestone 1.0 release happening in June 2015.[12] In 2011, Scrapinghub became the new official maintainer.[13][14]

References [edit]

1. ^ "Release notes — Scrapy documentation". *doc.scrapy.org*. Retrieved 2018-08-13.
2. ^ How do you pronounce "Scrapy"?
3. ^ Scrapy at a glance.
4. ^ "Frequently Asked Questions". Retrieved 28 July 2015.
5. ^ "Scrapy shell". Retrieved 28 July 2015.
6. ^ Bell, Eddie; Heusser, Jonathan. "Scalable Scraping Using Machine Learning". Retrieved 28 July 2015.
7. ^ Scrapy | Companies using Scrapy
8. ^ Montalenti, Andrew. "Web Crawling & Metadata Extraction in Python".
9. ^ "Scrapy Companies". *Scrapy website.*
10. ^ Hyphe v0.0.0: the first release of our new webcrawler is out!
11. ^ Ben Firshman [@bfirsh] (21 January 2010). "World Govt Data site uses Django, Solr, Haystack, Scrapy and other exciting buzzwords bit.ly/5jU3La #opendata #datastore" (Tweet) – via Twitter.
12. ^ Medina, Julia (19 June 2015). "Scrapy 1.0 official release out!". *scrapy-users* (Mailing list).
13. ^ Pablo Hoffman (2013). *List of the primary authors & contributors*. Retrieved 18 November 2013.
14. ^ Interview Scraping Hub.

Web Scrapers

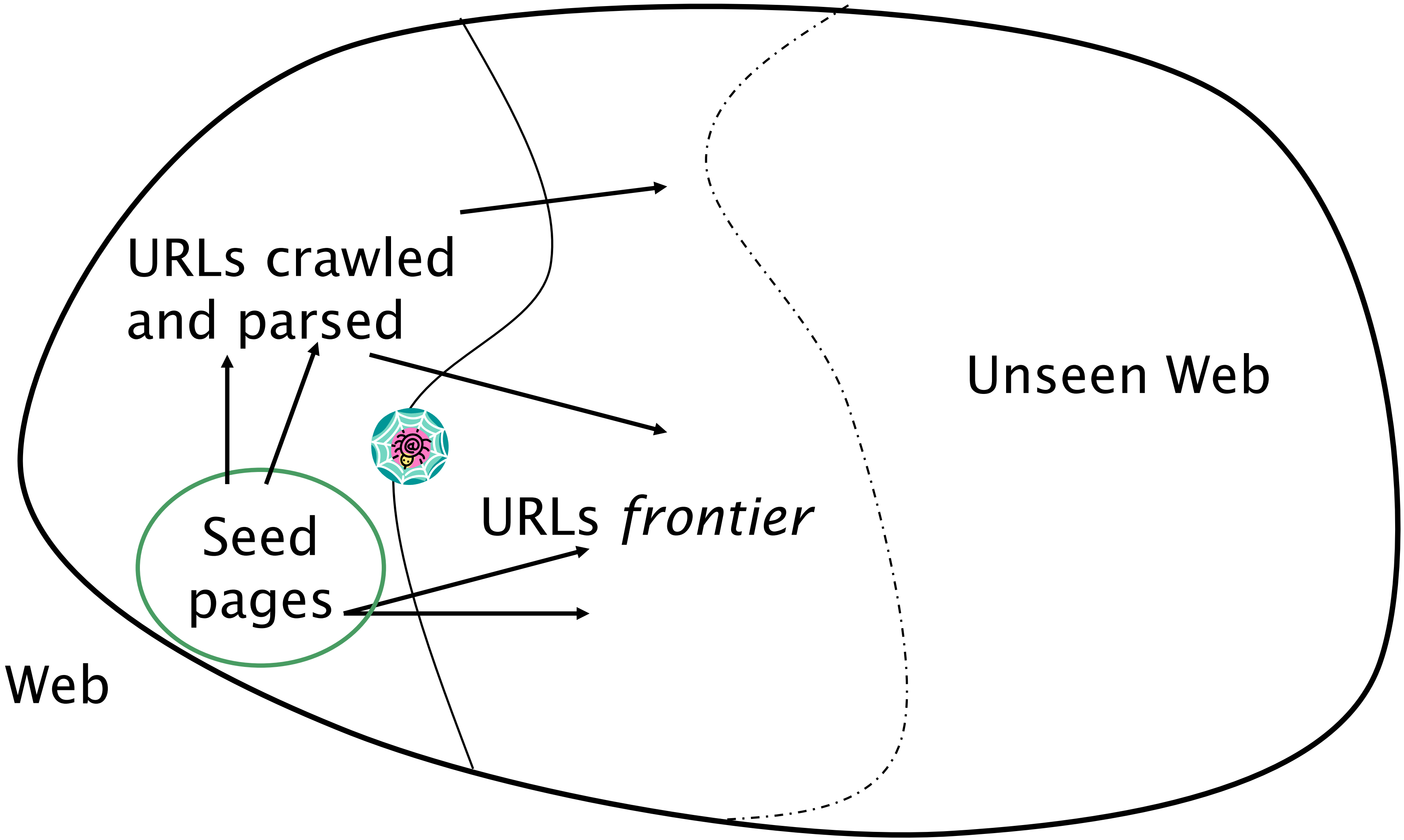
- Web scraping deals with the gathering of unstructured data on the web, typically in HTML format, putting it into structured data that can be stored and analyzed in a central local database or spreadsheet.
- Usually a manual process
- Usually does not go down into the url links

Web Crawler Specifics

- A program for downloading web pages.
- Given an initial set of **seed URLs**, it recursively downloads every page that is linked from pages in the set.
- A focused web crawler downloads only those pages whose content satisfies some criterion.

Also known as a web spider, bot, harvester.

Crawling the web



Simple picture – complications

- Web crawling difficult with one machine
  - All of the above steps can be distributed
- Malicious pages
  - Spam pages
  - Spider traps – incl dynamically generated
- Even non-malicious pages pose challenges
  - Latency/bandwidth to remote servers vary
  - Webmasters’ stipulations
    - How “deep” should you crawl a site’s URL hierarchy?
  - Site mirrors and duplicate pages
- Politeness – don’t hit a server too often

What any crawler *must* do

- Be Polite: Respect implicit and explicit politeness considerations
  - Only crawl allowed pages
  - Respect *robots.txt* (more on this shortly)
- Be Robust: Be immune to spider traps and other malicious behavior from web servers

What any crawler *should* do

Be capable of distributed operation: designed to run on multiple distributed machines

Be scalable: designed to increase the crawl rate by adding more machines

Performance/efficiency: permit full use of available processing and network resources

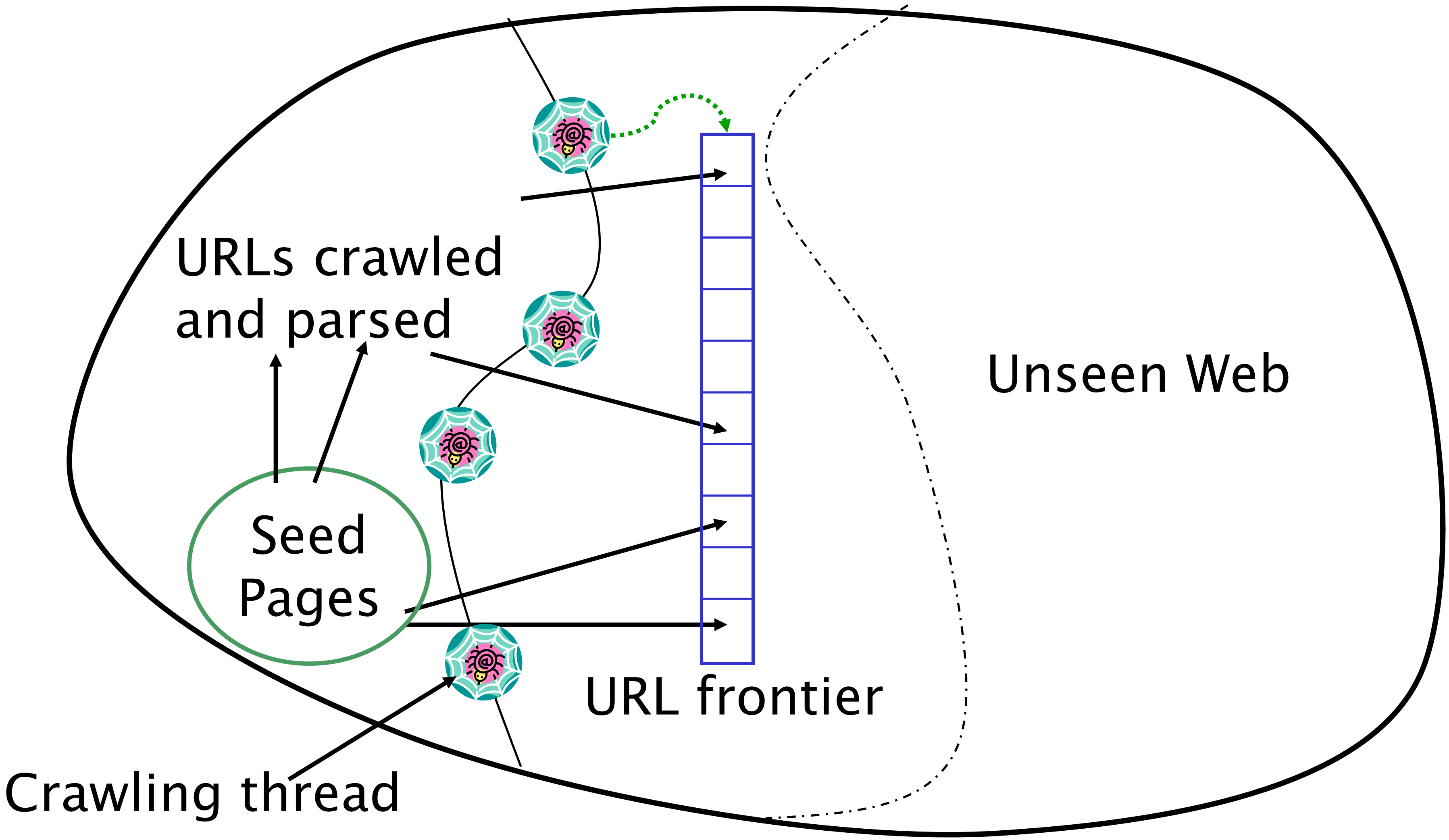
What any crawler *should* do

Fetch pages of “higher quality” first

Continuous operation: Continue fetching fresh copies of a previously fetched page

Extensible: Adapt to new data formats, protocols

More detail


URLs crawled and parsed
Unseen Web
Seed Pages
URL frontier
Crawling thread

URL frontier

The next node to crawl

- Can include multiple pages from the same host
- Must avoid trying to fetch them all at the same time
- Must try to keep all crawling threads busy

Explicit and implicit politeness

Explicit politeness: specifications from webmasters on what portions of site can be crawled

robots.txt

Implicit politeness: even with no specification, avoid hitting any site too often

Robots.txt

- Protocol for giving spiders (“robots”) limited access to a website, originally from 1994
  - www.robotstxt.org/wc/norobots.html
- Website announces its request on what can(not) be crawled
  - For a server, create a file `/robots.txt`
  - This file specifies access restrictions

Robots.txt example

No robot should visit any URL starting with "/yoursite/temp/", except the robot called “searchengine":

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

User-agent: searchengine
Disallow:
```

Processing steps in crawling

- Pick a URL from the frontier

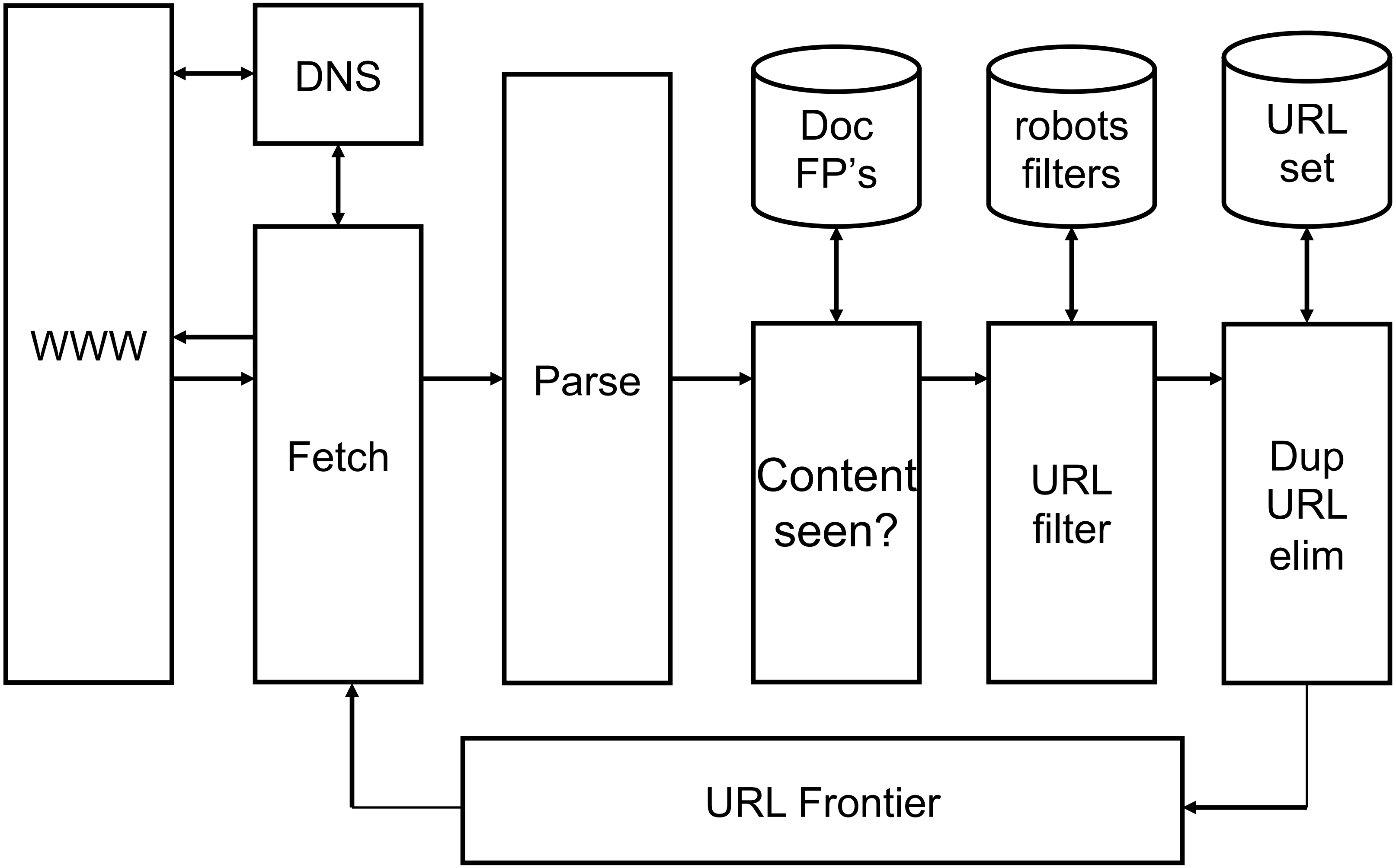
Which one?

- Fetch the document at the URL
- Parse the URL
  - Extract links from it to other docs (URLs)
- Check if URL has content already seen
  - If not, add to indexes

E.g., only crawl .edu, obey robots.txt, etc.

- For each extracted URL
  - Ensure it passes certain URL filter tests
  - Check if it is already in the frontier (duplicate URL elimination)

Basic crawl architecture



Crawling 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.
```

Web Crawler

- A crawler is a program that picks up a page and follows all the links on that page
- Crawler = Spider = Bot = Harvester
- Usual types of crawler:
  - Breadth First
  - Depth First
  - Combinations of the above

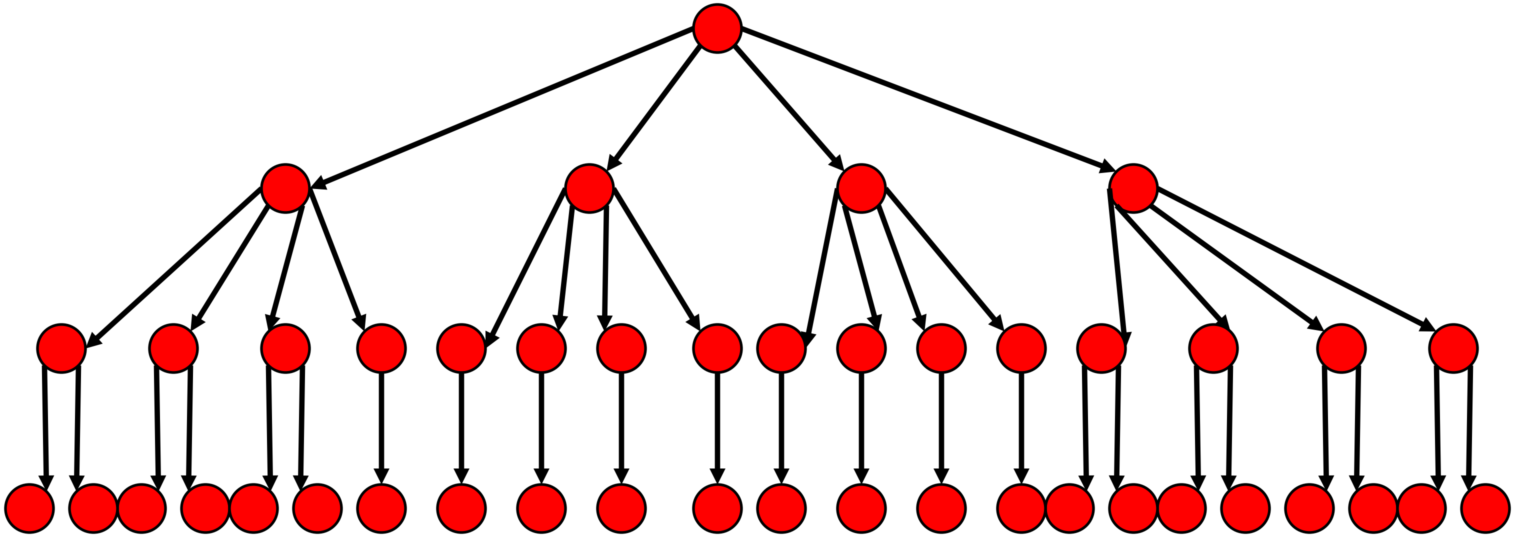
Breadth First Crawlers

Use breadth-first search (BFS) algorithm

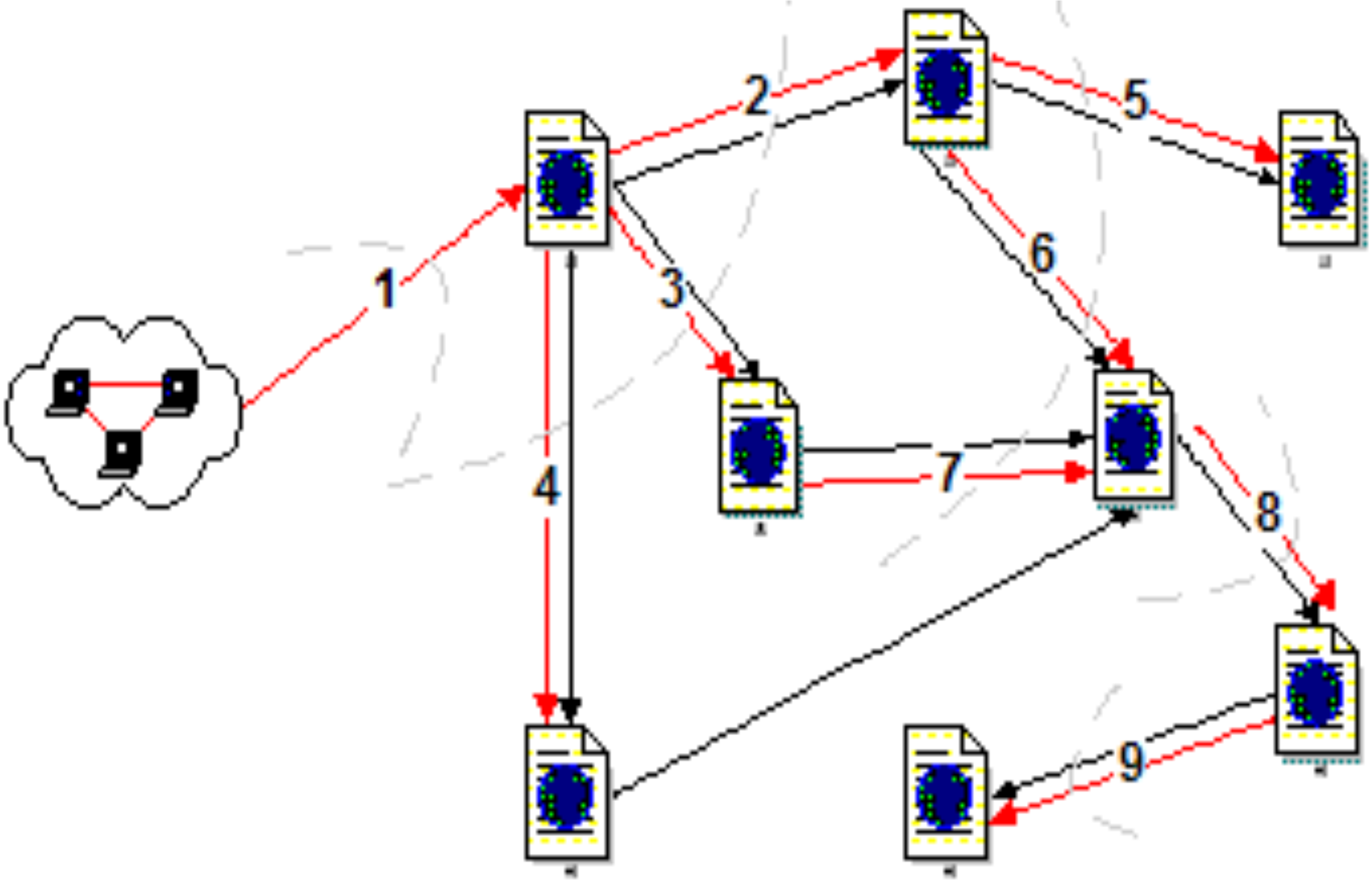
- Get all links from the starting page, and add them to a queue
- Pick the 1st link from the queue, get all links on the page and add to the queue
- Repeat above step till queue is empty

Search Strategies BF

Breadth-first Search



Breadth First Crawlers



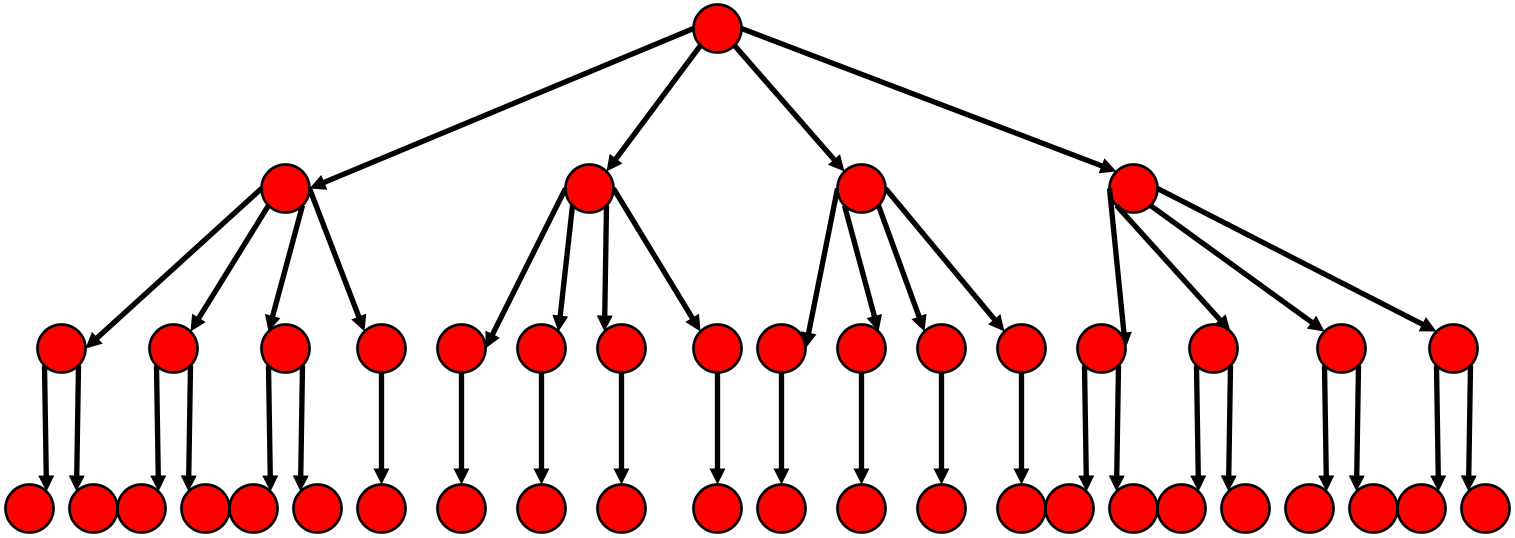
Depth First Crawlers

Use depth first search (DFS) algorithm

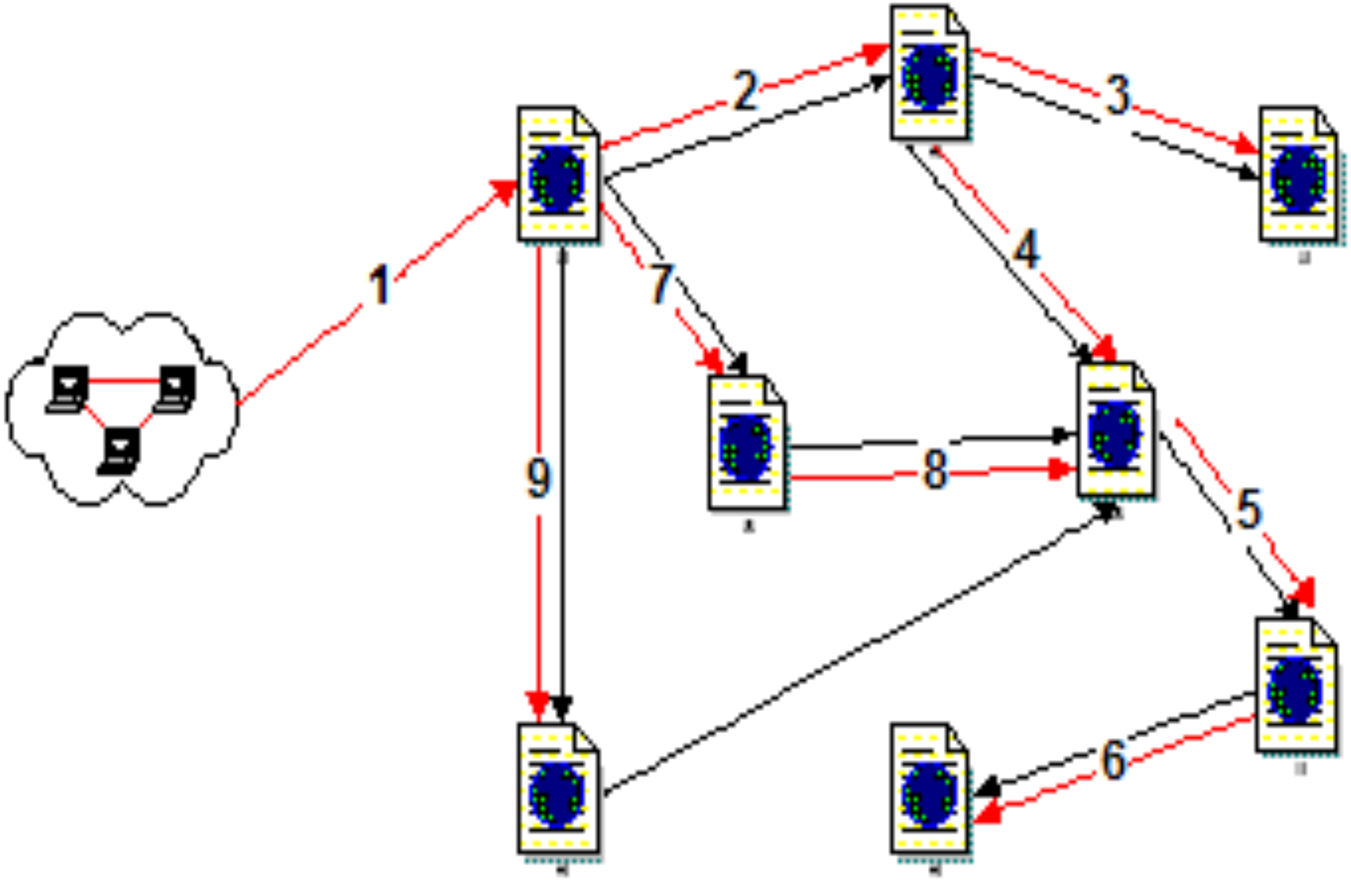
- Get the 1st link not visited from the start page
- Visit link and get 1st non-visited link
- Repeat above step till no no-visited links
- Go to next non-visited link in the previous level and repeat 2nd step

Search Strategies DF

Depth-first Search



Depth First Crawlers



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 implementable 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 “/”)
  - Not detect duplicated or mirrored pages.
- Index page using textual content as a key.
  - Requires first downloading page.
- Solr/Lucene Deduplication

Queueing Strategy

How new links 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

- Restrict spider to a particular site.
  - Remove links to other sites from Q.
- Restrict spider to a particular directory.
  - Remove links not in the specified directory.
- Obey page-owner restrictions (robot exclusion).

Link Extraction

Must find all links in a page and extract URLs.

<a href=“http://clgiles.ist.psu.edu/courses”>

Must complete relative URL’s using current page URL:

<a href=“projects”> to http://clgiles.ist.psu.edu/courses/ist441/projects

<a href=“../ist441/syllabus.html”> to http://clgiles.ist.psu.edu/courses/ist441/syllabus.html

URL Syntax

A URL has the following syntax:

<scheme>://<authority><path>?<query>#<fragment>

An *authority* has the syntax:

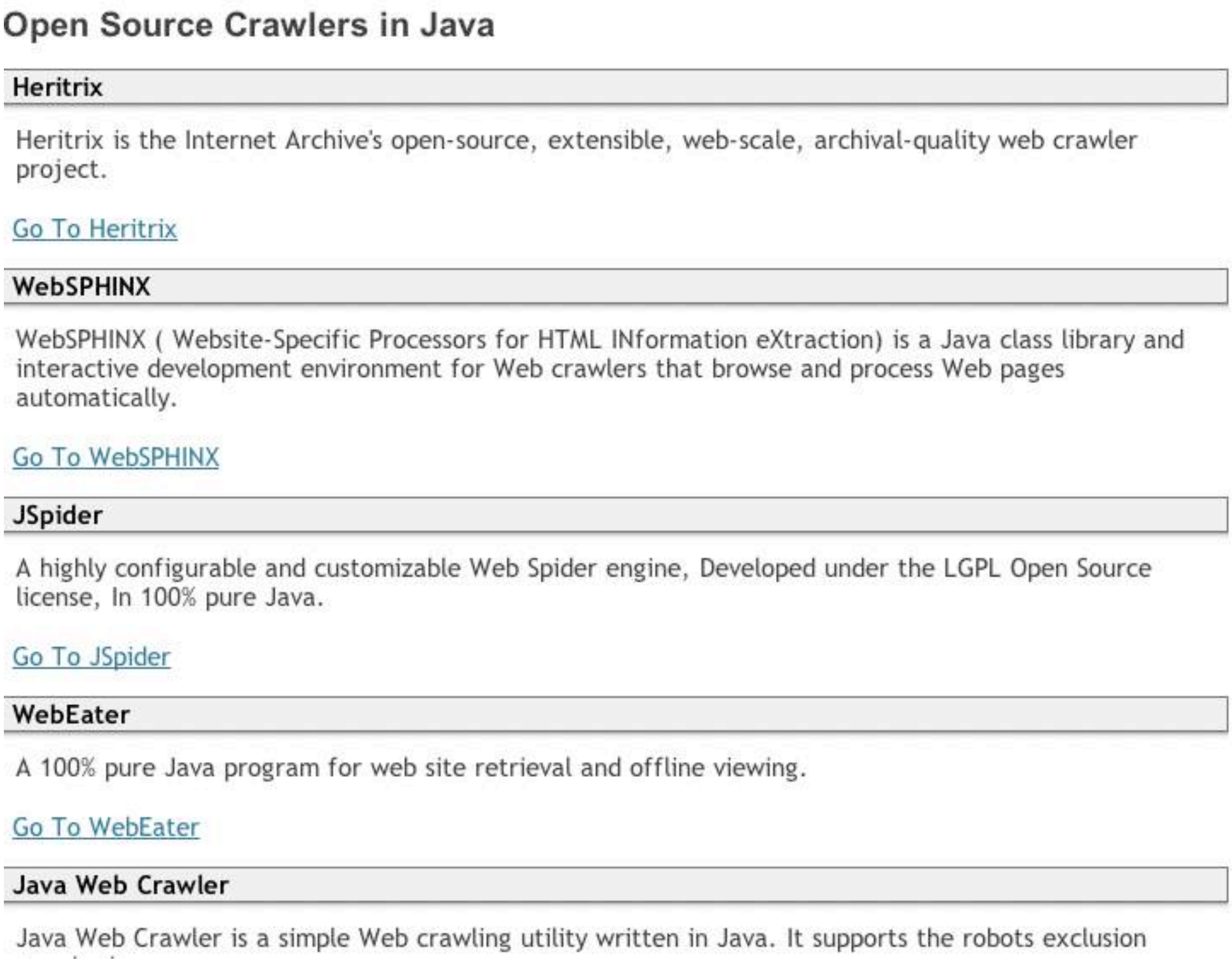
<host>:<port-number>

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>”>



Open Source Crawlers in Java

Heritrix

Heritrix is the Internet Archive's open-source, extensible, web-scale, archival-quality web crawler project.

Go To Heritrix

WebSPHINX

WebSPHINX (Website-Specific Processors for HTML INformation eXtraction) is a Java class library and interactive development environment for Web crawlers that browse and process Web pages automatically.

Go To WebSPHINX

JSpider

A highly configurable and customizable Web Spider engine, Developed under the LGPL Open Source license, In 100% pure Java.

Go To JSpider

WebEater

A 100% pure Java program for web site retrieval and offline viewing.

Go To WebEater

Java Web Crawler

Java Web Crawler is a simple Web crawling utility written in Java. It supports the robots exclusion

Robot Exclusion

How to control those robots!

Web sites and pages can specify that robots should not crawl/index certain areas.

Two components:

- Robots Exclusion Protocol (robots.txt): Site wide specification of excluded directories.
- Robots META Tag: Individual document tag to exclude indexing or following links inside a page that would otherwise be indexed

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
- http://clgiles.ist.psu.edu/robots.txt
- http://en.wikipedia.org/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: /
```

New `Allow:`

`Find some interesting robots.txt`

Robot Exclusion Protocol Examples

Exclude specific directories:

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

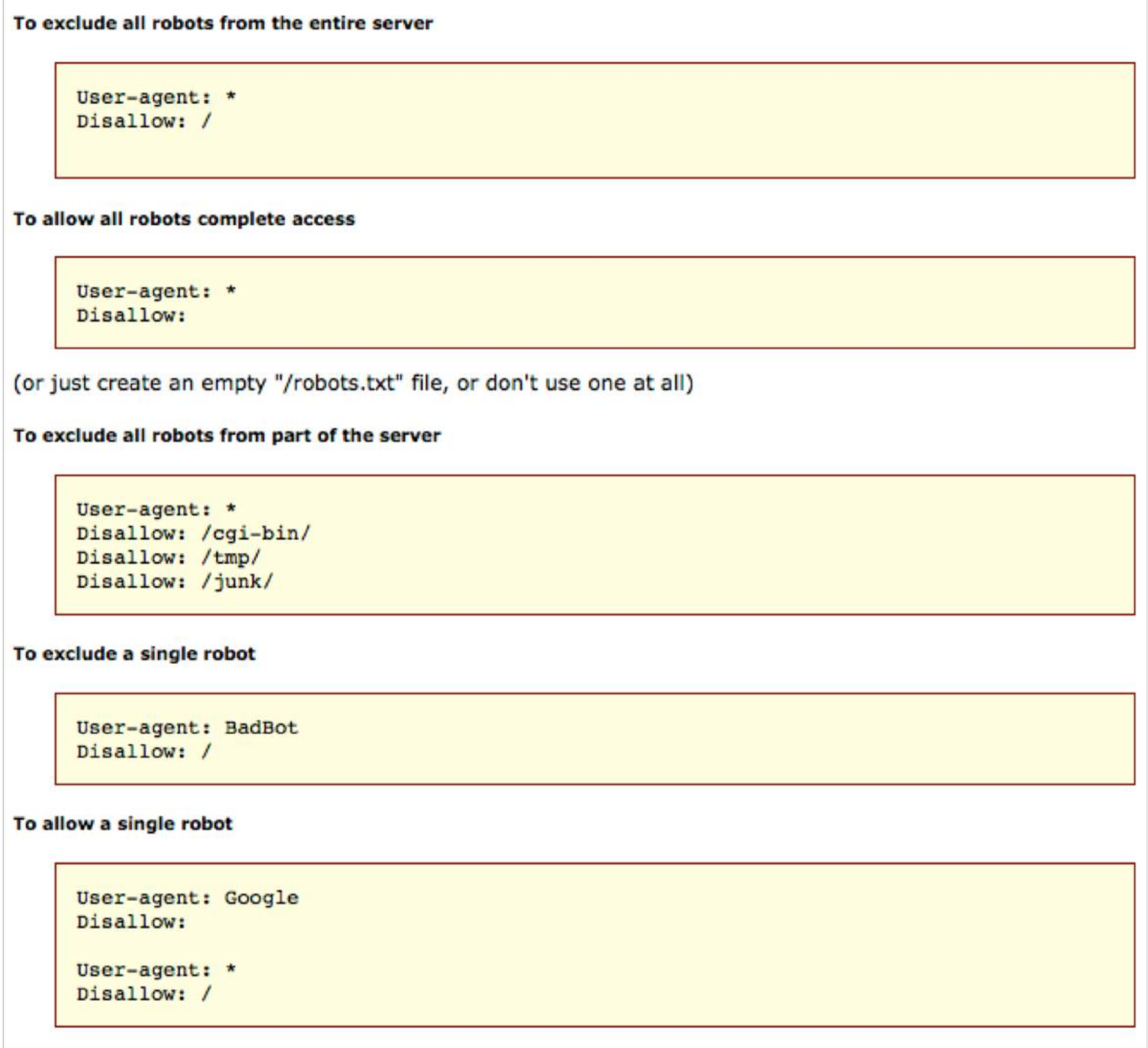
Exclude a specific robot:

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

Allow a specific robot:

```
User-agent: GoogleBot
Disallow:

User-agent: *
Disallow: /
```



To exclude all robots from the entire server

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

To allow all robots complete access

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

(or just create an empty "/robots.txt" file, or don't use one at all)

To exclude all robots from part of the server

```
User-agent: *
Disallow: /cgi-bin/
Disallow: /tmp/
Disallow: /junk/
```

To exclude a single robot

```
User-agent: BadBot
Disallow: /
```

To allow a single robot

```
User-agent: Google
Disallow:

User-agent: *
Disallow: /
```

Robot Exclusion Protocol Has Not Well Defined Details

Only use blank lines to separate different User-agent disallowed directories.

One directory per “Disallow” line.

No regex (regular expression) patterns in directories.

- What about “robot.txt”?
- Ethical robots obey “robots.txt” as best as they can interpret them

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:

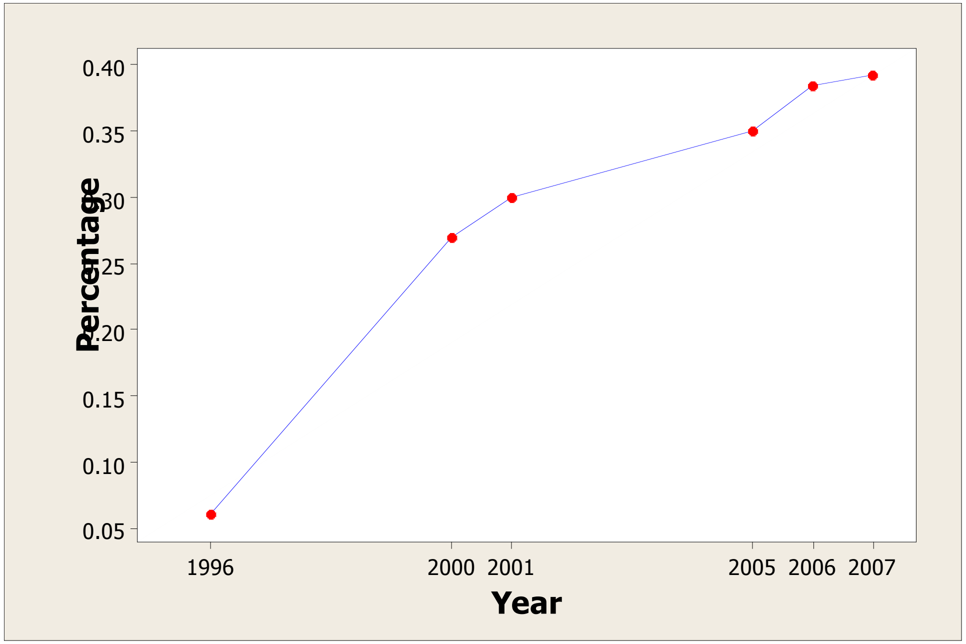
index | noindex: Allow/disallow indexing of this page.

follow | nofollow: Allow/disallow following links on this page.

History of the Robots Exclusion Protocol

A consensus June 30, 1994 on the robots mailing list

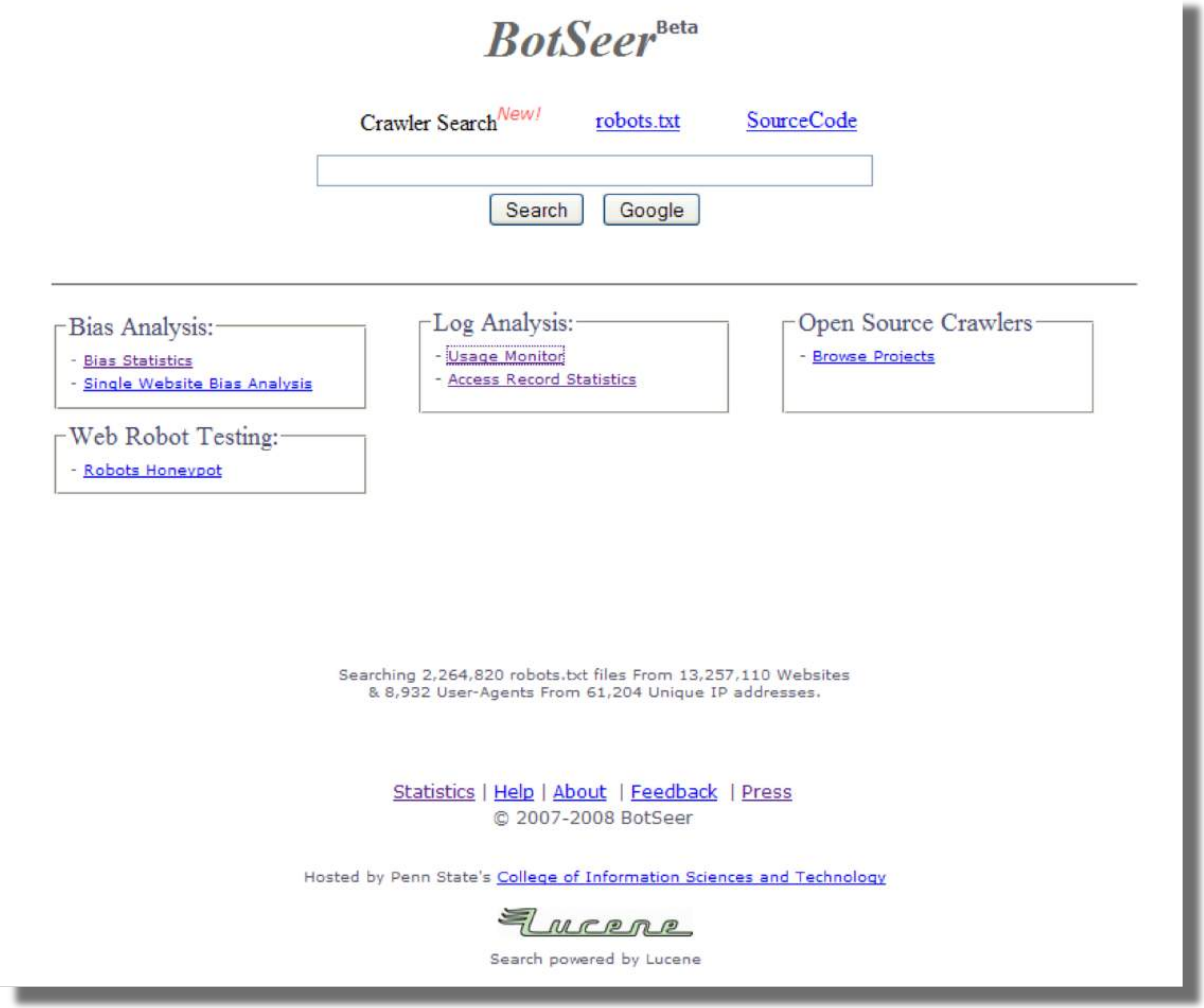
Revised and Proposed to IETF in 1996 by M. Koster[14]



Never accepted as an official standard

Continues to be used and growing

BotSeer - Robots.txt search engine



BotSeer Beta

Crawler Search New! robots.txt SourceCode

Bias Analysis:

- Bias Statistics
- Single Website Bias Analysis

Web Robot Testing:

- Robots Honeypot

Log Analysis:

- Usage Monitor
- Access Record Statistics

Open Source Crawlers

- Browse Projects

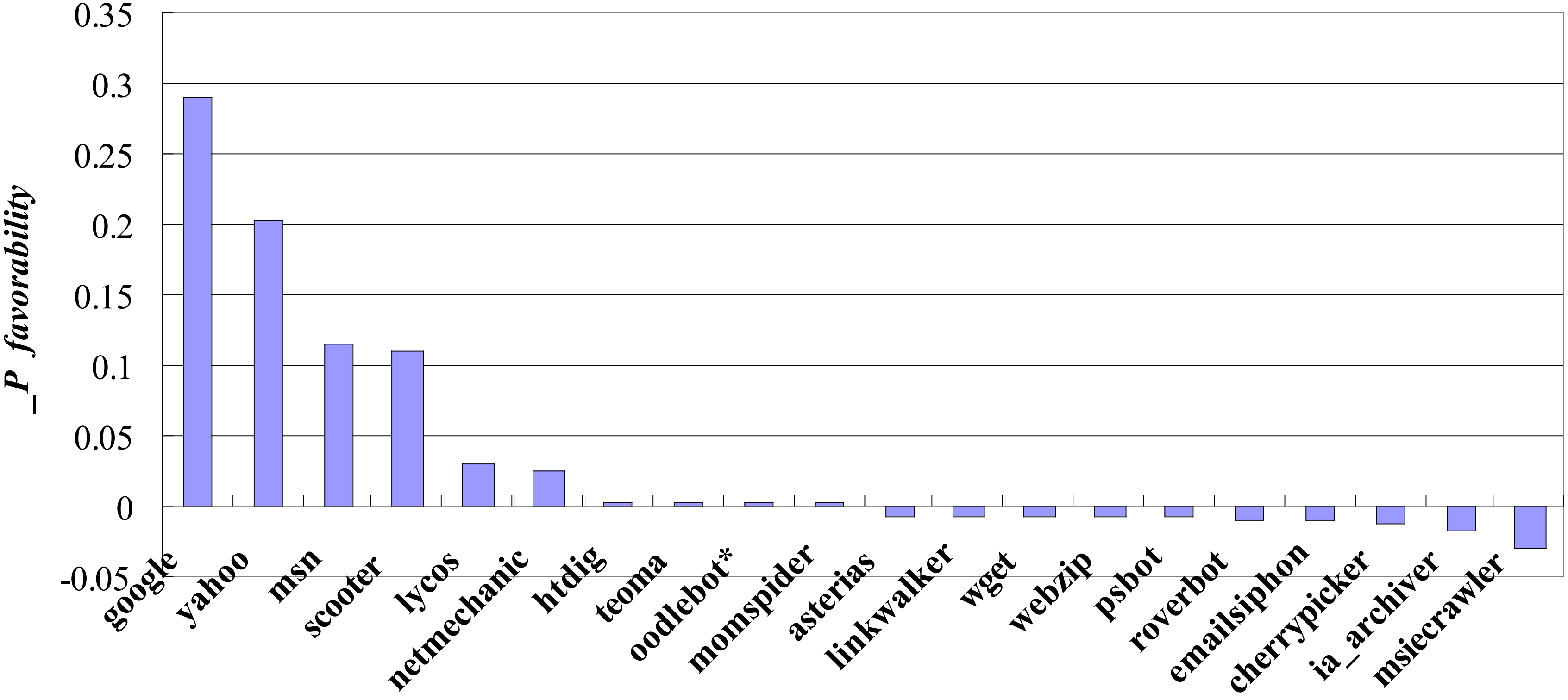
Searching 2,264,820 robots.txt files From 13,257,110 Websites
& 8,932 User-Agents From 61,204 Unique IP addresses.

Statistics | Help | About | Feedback | Press
© 2007-2008 BotSeer

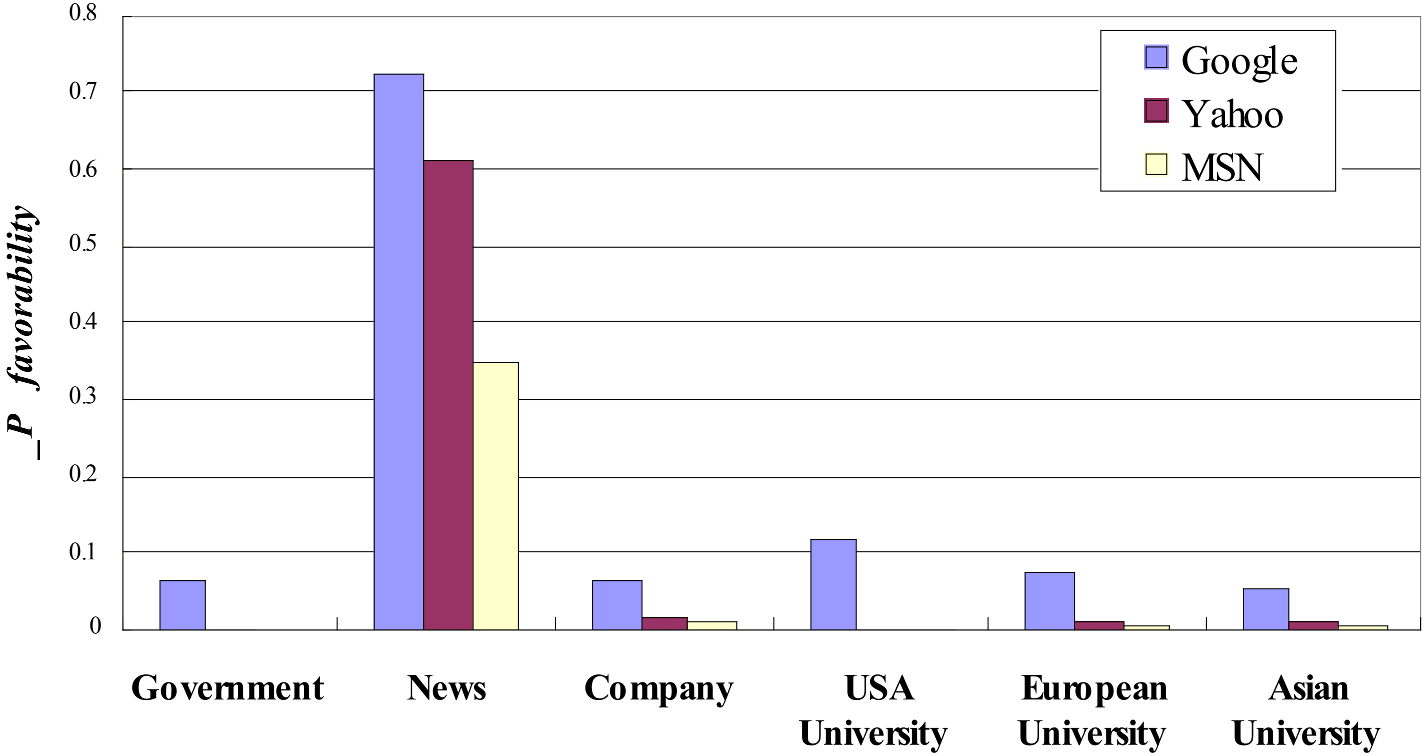
Hosted by Penn State's College of Information Sciences and Technology

Search powered by Lucene

Top 10 favored and disfavored robots – Ranked by Δ*P favorability*.

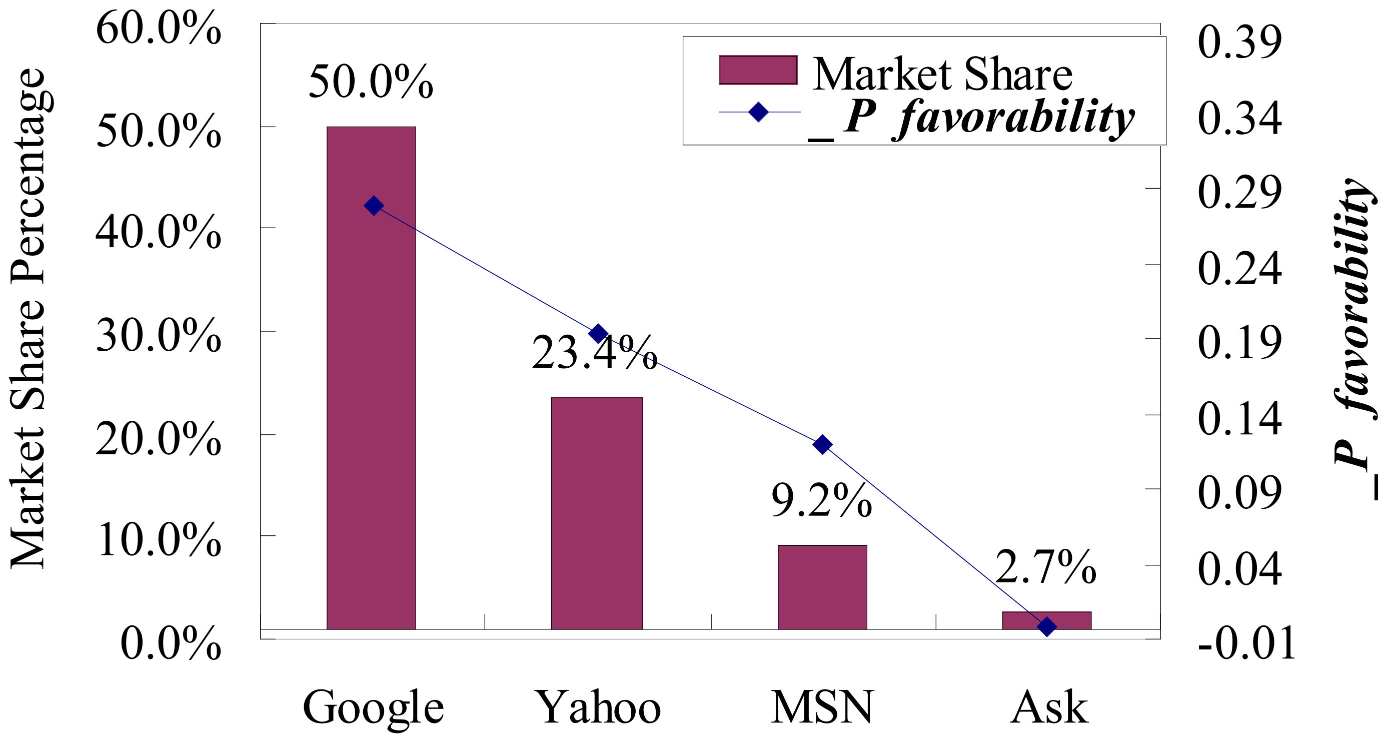


Comparison of Google, Yahoo and MSN



Search Engine Market Share vs. Robot Bias

- Pearson product-moment correlation coefficient: 0.930, P-value < 0.001

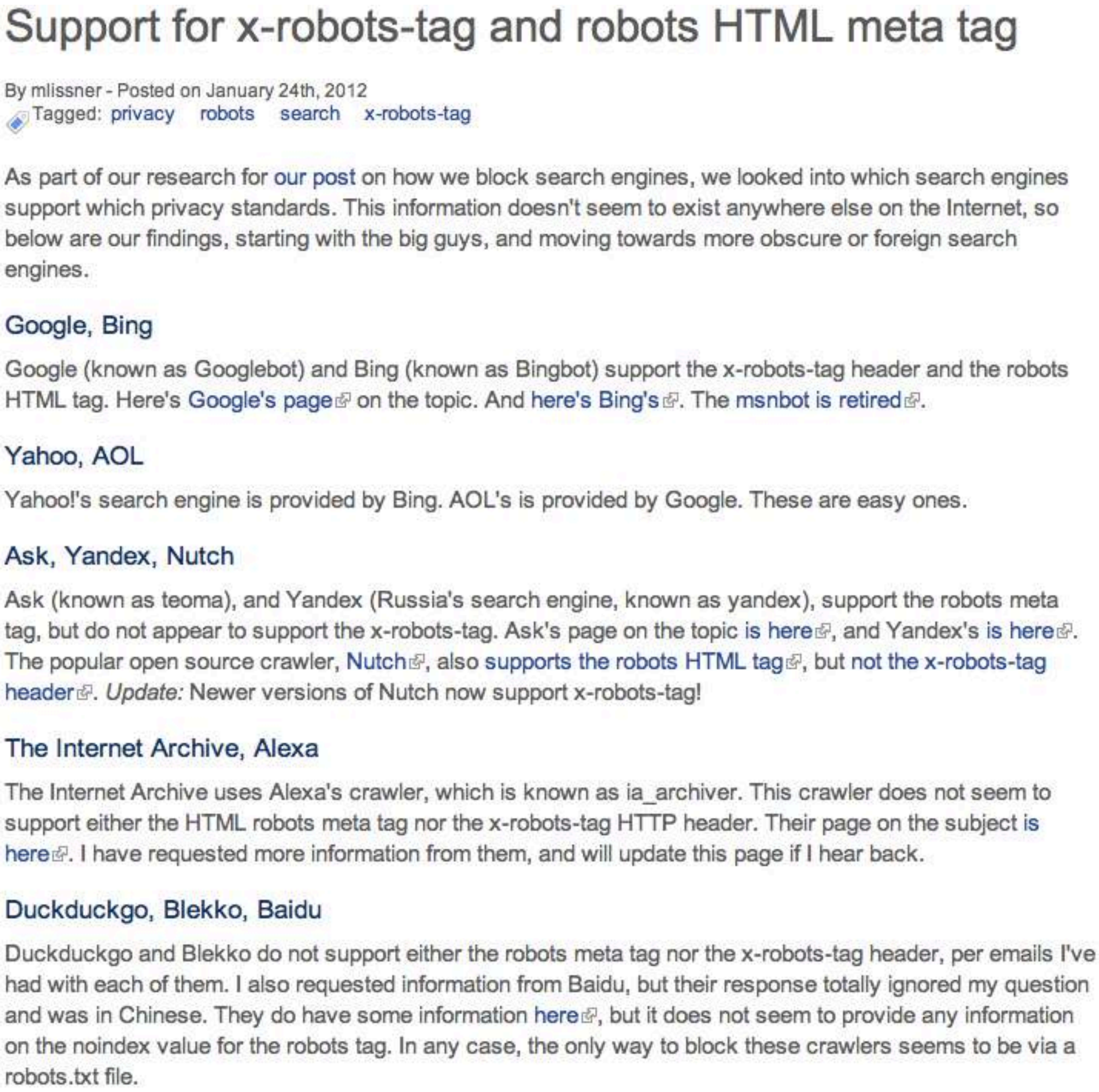


* *Search engine market share data is obtained from NielsenNetratings[16]*

Robot Exclusion Issues

- META tag is newer and less well-adopted than “robots.txt”. (growing in use – xml sitemaps)
- Standards are conventions to be followed by “good robots.”
  - Companies have been prosecuted for “disobeying” these conventions and “trespassing” on private cyberspace.
- “Good robots” also try not to “hammer” individual sites with lots of rapid requests.
  - “Denial of service” attack.

T OR F: robots.txt file increases your pagerank?



Support for x-robots-tag and robots HTML meta tag

By mlissner - Posted on January 24th, 2012
Tagged: privacy robots search x-robots-tag

As part of our research for our post on how we block search engines, we looked into which search engines support which privacy standards. This information doesn't seem to exist anywhere else on the Internet, so below are our findings, starting with the big guys, and moving towards more obscure or foreign search engines.

Google, Bing

Google (known as Googlebot) and Bing (known as Bingbot) support the x-robots-tag header and the robots HTML tag. Here's Google's page on the topic. And here's Bing's. The msnbot is retired.

Yahoo, AOL

Yahoo!'s search engine is provided by Bing. AOL's is provided by Google. These are easy ones.

Ask, Yandex, Nutch

Ask (known as teoma), and Yandex (Russia's search engine, known as yandex), support the robots meta tag, but do not appear to support the x-robots-tag. Ask's page on the topic is here, and Yandex's is here. The popular open source crawler, Nutch, also supports the robots HTML tag, but not the x-robots-tag header. *Update:* Newer versions of Nutch now support x-robots-tag!

The Internet Archive, Alexa

The Internet Archive uses Alexa's crawler, which is known as ia_archiver. This crawler does not seem to support either the HTML robots meta tag nor the x-robots-tag HTTP header. Their page on the subject is here. I have requested more information from them, and will update this page if I hear back.

Duckduckgo, Blekko, Baidu

Duckduckgo and Blekko do not support either the robots meta tag nor the x-robots-tag header, per emails I've had with each of them. I also requested information from Baidu, but their response totally ignored my question and was in Chinese. They do have some information here, but it does not seem to provide any information on the noindex value for the robots tag. In any case, the only way to block these crawlers seems to be via a robots.txt file.

Web bots

- Not all crawlers are ethical (obey robots.txt)
- Not all webmasters know how to write correct robots.txt files
  - Many have inconsistent Robots.txt
- Bots interpret these inconsistent robots.txt in many ways.
- Many bots out there!
  - It’s the wild, wild west

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

Simple Web Crawler Algorithm

Basic Algorithm

Let S be set of URLs to pages waiting to be indexed. Initially S is the singleton, s, known as the seed.

Take an element u of S and **retrieve** the page, p, that it references.

Parse the page p and **extract** the set of URLs L it has links to.

Update $S = S + L - u$

Repeat as many times as necessary.

Not so Simple…

Performance -- How do you crawl 1,000,000,000 pages?

Politeness -- How do you avoid overloading servers?

Failures -- Broken links, time outs, spider traps.

Strategies -- How deep do we go? Depth first or breadth first?

Implementations -- How do we store and update *S* and the other data structures needed?

What to Retrieve

No web crawler retrieves everything

Most crawlers retrieve only

- HTML (leaves and nodes in the tree)
- ASCII clear text (only as leaves in the tree)

Some retrieve

- PDF
- PostScript,…

Indexing after crawl

- Some index only the first part of long files
- Do you keep the files (e.g., Google cache)?

Building a Web Crawler: Links are not Easy to Extract

- Relative/Absolute
- CGI
  - Parameters
  - Dynamic generation of pages
- Server-side scripting
- Server-side image maps
- Links buried in scripting code

Crawling to build an historical archive

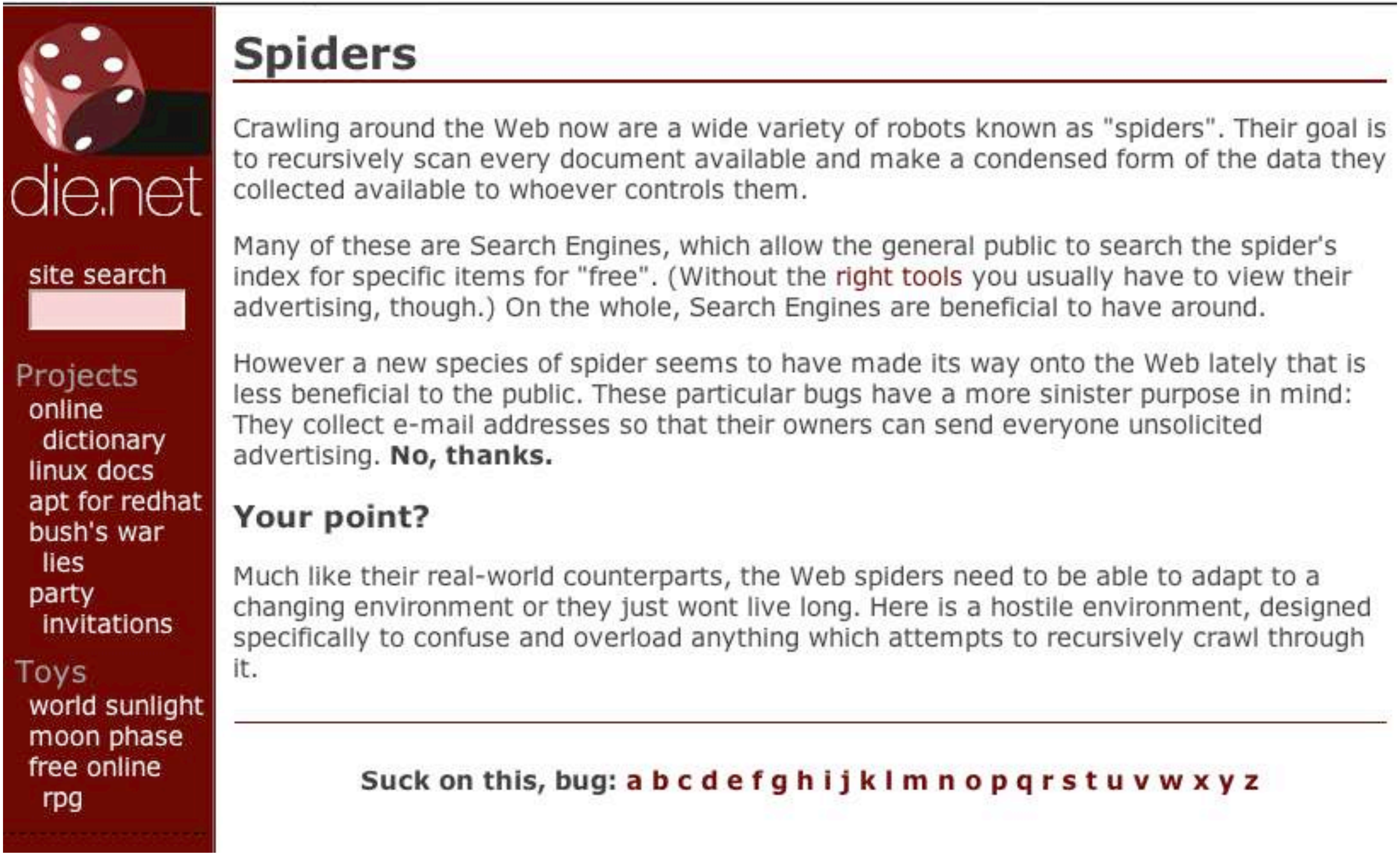
Internet Archive:

http://www.archive.org

A non-for profit organization in San Francisco, created by Brewster Kahle, to collect and retain digital materials for future historians.

Services include the ***Wayback Machine***.

http://spiders.must.die.net



Spiders

Crawling around the Web now are a wide variety of robots known as "spiders". Their goal is to recursively scan every document available and make a condensed form of the data they collected available to whoever controls them.

Many of these are Search Engines, which allow the general public to search the spider's index for specific items for "free". (Without the right tools you usually have to view their advertising, though.) On the whole, Search Engines are beneficial to have around.

However a new species of spider seems to have made its way onto the Web lately that is less beneficial to the public. These particular bugs have a more sinister purpose in mind: They collect e-mail addresses so that their owners can send everyone unsolicited advertising. **No, thanks.**

Your point?

Much like their real-world counterparts, the Web spiders need to be able to adapt to a changing environment or they just wont live long. Here is a hostile environment, designed specifically to confuse and overload anything which attempts to recursively crawl through it.

Suck on this, bug: a b c d e f g h i j k l m n o p q r s t u v w x y z

Spider Traps

- A spider trap (or crawler trap) is a set of web pages that may intentionally or unintentionally be used to cause a web crawler or search bot to make an infinite number of requests or cause a poorly constructed crawler to crash.
- Spider traps may be created to "catch" spambots or other crawlers that waste a website's bandwidth. Common techniques used are:
  - creation of indefinitely deep directory structures like
    - http://foo.com/bar/foo/bar/foo/bar/foo/bar/.....
  - dynamic pages like calendars that produce an infinite number of pages for a web crawler to follow.
  - pages filled with a large number of characters, crashing the lexical analyzer parsing the page.
  - pages with session-id's based on required cookies
  - Others?
- There is no algorithm to detect all spider traps. Some classes of traps can be detected automatically, but new, unrecognized traps arise quickly.

Research Topics in Web Crawling

Intelligent crawling - focused crawling

  How frequently to crawl

  What to crawl

  What strategies to use.

- Identification of anomalies and crawling traps.
- Strategies for crawling based on the content of web pages (focused and selective crawling).
- Duplicate detection.

Detecting Bots

It's the wild, wild west out there!

Inspect Server Logs:

•**User Agent Name** - user agent name.

•**Frequency of Access** - A very large volume of accesses from the same IP address is usually a tale-tell sign of a bot or spider.

•**Access Method** - Web browsers being used by human users will almost always download all of the images too. A bot typically only goes after the text.

•**Access Pattern** - Not erratic

Simple picture – complications

- Search engine grade web crawling isn't feasible with one machine
  - All of the above steps distributed
- Even non-malicious pages pose challenges
  - Latency/bandwidth to remote servers vary
  - Webmasters' stipulations
    - How "deep" should you crawl a site's URL hierarchy?
  - Site mirrors and duplicate pages
- Malicious pages
  - Spam pages
  - Spider traps – incl dynamically generated
- Politeness – don't hit a server too often

What any crawler *must* do

- Be Polite: Respect implicit and explicit politeness considerations
  - Only crawl allowed pages
  - Respect *robots.txt* (more on this shortly)
- Be Robust: Be immune to spider traps and other malicious behavior from web servers

What any commercial grade crawler *should* do

Be capable of distributed operation: designed to run on multiple distributed machines

Be scalable: designed to increase the crawl rate by adding more machines

Performance/efficiency: permit full use of available processing and network resources

What any crawler *should* do

Fetch pages of "higher quality" first

Continuous operation: Continue fetching fresh copies of a previously fetched page

Extensible: Adapt to new data formats, protocols

Crawling research issues

- Open research question
  - Not easy
  - Domain specific?
    - No crawler works for all problems
  - Evaluation
    - Complexity
  - Crucial for specialty search

Search Engine Web Crawling Policies

- Their policies determine what gets indexed
- Freshness
  - How often the SE crawls
- What gets ranked and how
  - SERP (search engine results page)
- Experimental SEO
  - Make changes; see what happens

Web Crawling

- Web crawlers are foundational species
  - No web search engines without them
  - Scrapers subclass of crawlers
- Crawl policy
  - Breath first
  - Depth first
- Crawlers should be optimized for area of interest
  - Focused crawlers
- robots.txt – gateway to web content
  - Crawlers obey robots.txt