# Crawlers and Crawling

# There are Many Crawlers

- **A web crawler is a computer program that visits web pages in an organized way**
  - Sometimes called a spider or robot
- **A list of web crawlers can be found at**

  http://en.wikipedia.org/wiki/Web_crawler

  Google's crawler is called googlebot, see

http://support.google.com/webmasters/bin/answer.py?hl=en&answer=182072

- **Yahoo's web crawler is/was called Yahoo! Slurp, see**

http://en.wikipedia.org/wiki/Yahoo!_Search

- **Bing uses five crawlers**
  - Bingbot, standard crawler
  - Adidxbot, used by Bing Ads
  - MSNbot, remnant from MSN, but still in use
  - MSNBotMedia, crawls images and video
  - BingPreview, generates page snapshots
- **For details see**: http://www.bing.com/webmaster/help/which-crawlers-does-bing-use-8c184ec0

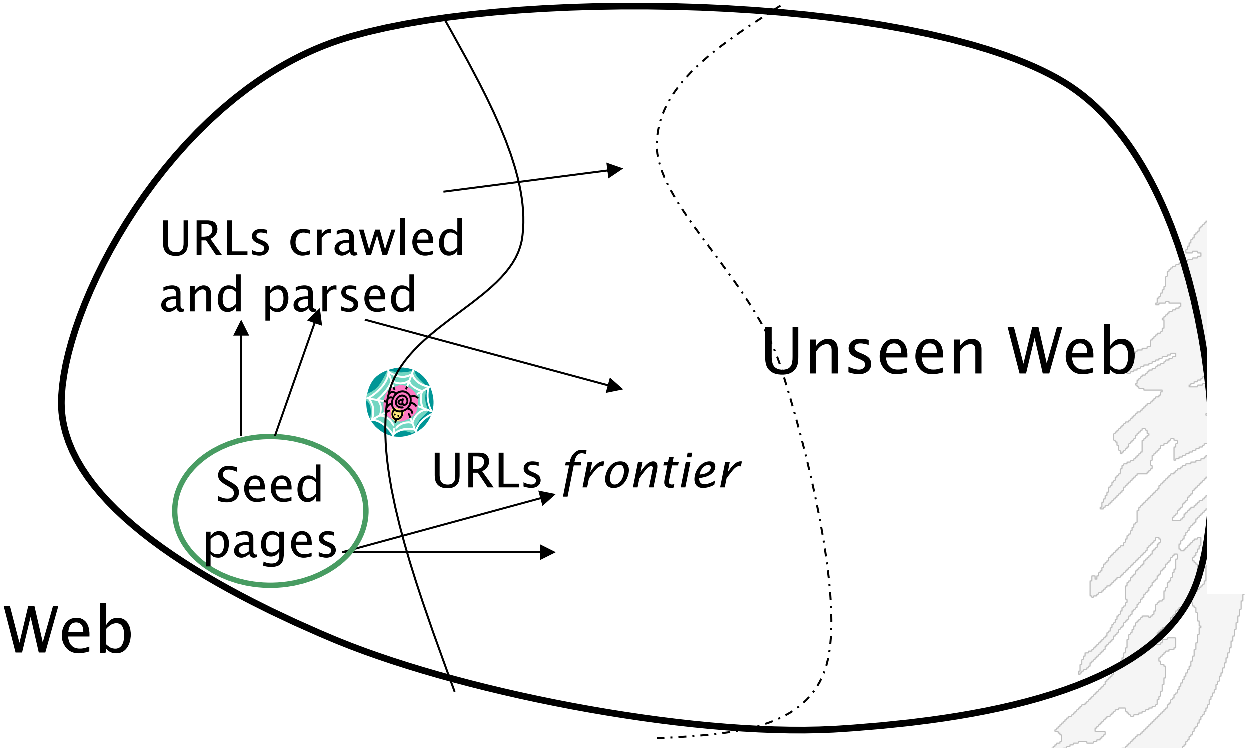
# Web Crawling Issues

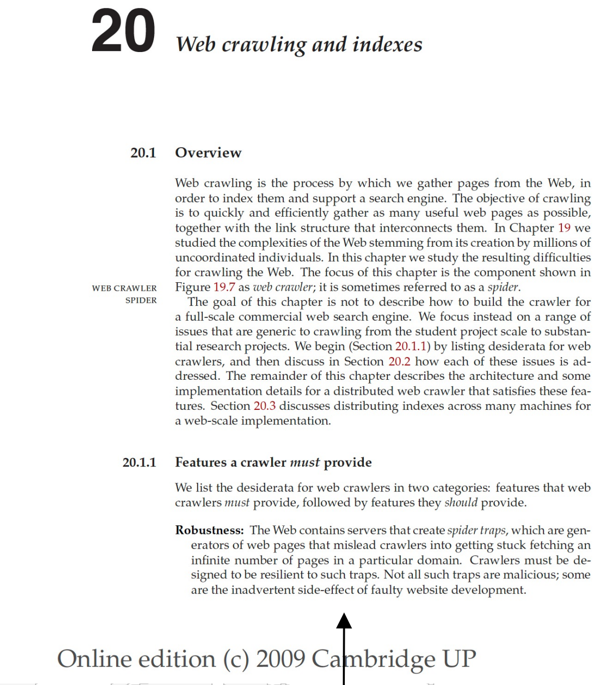
- **How to crawl?**
  - *Quality*: how to find the "Best" pages first
  - *Efficiency*: how to avoid duplication (or near duplication)
  - *Etiquette*: behave politely by not disturbing a website's performance
- **How much to crawl? How much to index?**
  - *Coverage*: What percentage of the web should be covered?
  - *Relative Coverage:* How much do competitors have?
- **How often to crawl?**
  - *Freshness*: How much has changed?
  - How much has <u>really</u> changed?

# Simplest Crawler Operation

- **Initialize (begin with known "seed" pages)**
- **Loop: Fetch and parse a page**
  - Place the page in a database
  - Extract the URLs within the page
  - Place the extracted URLs on a queue
  - Fetch a URL on the queue and repeat

# Crawling Picture





## 20 *Web crawling and indexes*

### 20.1 Overview

Web crawling is the process by which we gather pages from the Web, in order to index them and support a search engine. The objective of crawling is to quickly and efficiently gather as many useful web pages as possible, together with the link structure that interconnects them. In Chapter 19 we studied the complexities of the Web stemming from its creation by millions of uncoordinated individuals. In this chapter we study the resulting difficulties for crawling the Web. The focus of this chapter is the component shown in Figure 19.7 as *web crawler*; it is sometimes referred to as a *spider*.

WEB CRAWLER
SPIDER

The goal of this chapter is not to describe how to build the crawler for a full-scale commercial web search engine. We focus instead on a range of issues that are generic to crawling from the student project scale to substantial research projects. We begin (Section 20.1.1) by listing desiderata for web crawlers, and then discuss in Section 20.2 how each of these issues is addressed. The remainder of this chapter describes the architecture and some implementation details for a distributed web crawler that satisfies these features. Section 20.3 discusses distributing indexes across many machines for a web-scale implementation.

#### 20.1.1 Features a crawler *must* provide

We list the desiderata for web crawlers in two categories: features that web crawlers *must* provide, followed by features they *should* provide.

**Robustness:** The Web contains servers that create *spider traps*, which are generators of web pages that mislead crawlers into getting stuck fetching an infinite number of pages in a particular domain. Crawlers must be designed to be resilient to such traps. Not all such traps are malicious; some are the inadvertent side-effect of faulty website development.

Online edition (c) 2009 Cambridge UP

Our textbook

# Simple Picture – Complications

- **Crawling the entire web isn't feasible with one machine**
  - But all of the above steps can be distributed
- **Challenges**
  - **Handling/Avoiding malicious pages**
    - Some pages contain spam
    - Some pages contain spider traps – especially dynamically generated pages
  - **Even non-malicious pages pose challenges**
    - Latency/bandwidth to remote servers can vary widely
    - Robots.txt stipulations can prevent web pages from being visited
    - How can one avoid mirrored sites and duplicate pages
  - **Maintain politeness – don't hit a server too often**

# Robots.txt

- **There is a protocol that defines the limitations for a web crawler as it visits a website; its definition is here**
  - http://www.robotstxt.org/orig.html
- **The website announces its request on what can(not) be crawled by placing a robots.txt file in the root directory**
  - e.g. see

http://www.ticketmaster.com/robots.txt

# Robots.txt Example

- No robot visiting this domain should visit any URL starting with "/yoursite/temp/":

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

- Directives are case sensitive
- Additional symbols allowed in the robots.txt directives include:
  - '*‘ - matches a sequence of characters
  - '$' - anchors at the end of the URL string
- Example of '*':

```
User-agent: Slurp
Allow: /public*/
Disallow: /*_print*.html
Disallow: /*?sessionid
```

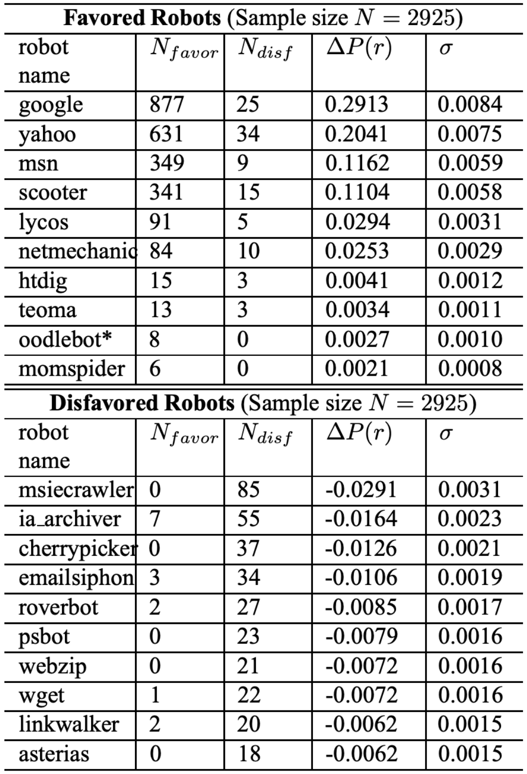
# Websites Give Preference to Googlebot

- In researching how websites treat crawlers, Zack Maril looked at 17 million robots.txt files and discovered many places where Google is given an advantage; e.g. see
- https://www.sciencedirect.com/robots.txt
- E.g.
- `# go away ? tell all others not in the list below to stay out!`
- `User-agent: *`
- `Disallow: /`
- `# As of 02/09/2021, there are 10 crawlers welcomed by SD`
- `User-agent: Googlebot`
- `Disallow: /cache/MiamiImageURL/`
- `Disallow: /science?_ob=MiamiCaptionURL*`

USC Viterbi School of Engineering

# A Study of Robots.txt Files

- ***Determining bias to search engines from robots.txt*, Giles, Sun, Zhuang, Penn State**



**Favored Robots** (Sample size $N = 2925$)

| robot name | $N_{favor}$ | $N_{disf}$ | $\Delta P(r)$ | $\sigma$ |
|---|---|---|---|---|
| google | 877 | 25 | 0.2913 | 0.0084 |
| yahoo | 631 | 34 | 0.2041 | 0.0075 |
| msn | 349 | 9 | 0.1162 | 0.0059 |
| scooter | 341 | 15 | 0.1104 | 0.0058 |
| lycos | 91 | 5 | 0.0294 | 0.0031 |
| netmechanic | 84 | 10 | 0.0253 | 0.0029 |
| htdig | 15 | 3 | 0.0041 | 0.0012 |
| teoma | 13 | 3 | 0.0034 | 0.0011 |
| oodlebot* | 8 | 0 | 0.0027 | 0.0010 |
| momspider | 6 | 0 | 0.0021 | 0.0008 |

**Disfavored Robots** (Sample size $N = 2925$)

| robot name | $N_{favor}$ | $N_{disf}$ | $\Delta P(r)$ | $\sigma$ |
|---|---|---|---|---|
| msiecrawler | 0 | 85 | -0.0291 | 0.0031 |
| ia_archiver | 7 | 55 | -0.0164 | 0.0023 |
| cherrypicker | 0 | 37 | -0.0126 | 0.0021 |
| emailsiphon | 3 | 34 | -0.0106 | 0.0019 |
| roverbot | 2 | 27 | -0.0085 | 0.0017 |
| psbot | 0 | 23 | -0.0079 | 0.0016 |
| webzip | 0 | 21 | -0.0072 | 0.0016 |
| wget | 1 | 22 | -0.0072 | 0.0016 |
| linkwalker | 2 | 20 | -0.0062 | 0.0015 |
| asterias | 0 | 18 | -0.0062 | 0.0015 |

**Table 2. Top 10 favored and disfavored robots. $\sigma$ is the standard deviation of $\Delta P(r)$.**

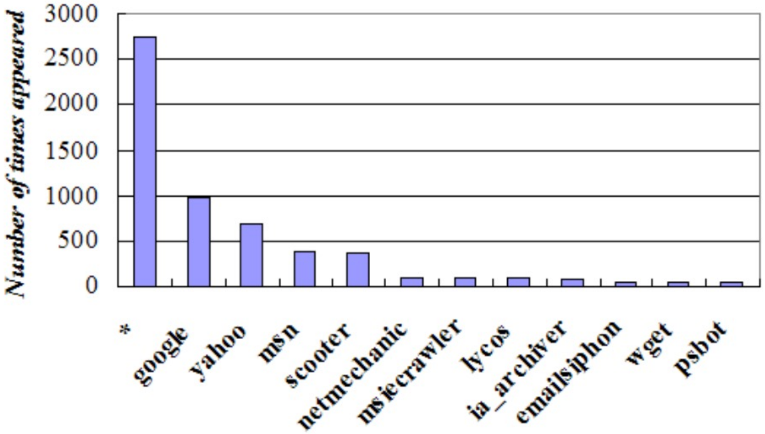
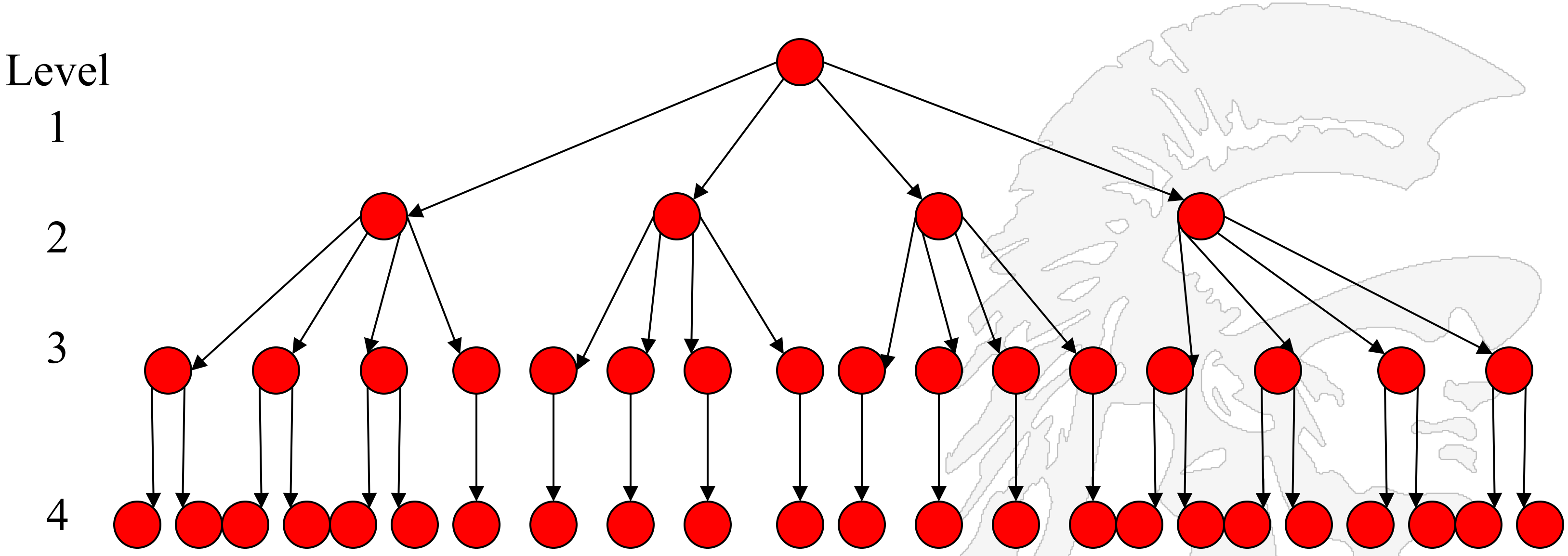


**Figure 2. Most frequently used robot names in robots.txt files. The height of the bar represents the number of times a robot appeared in our dataset.**

# Basic Search Strategies

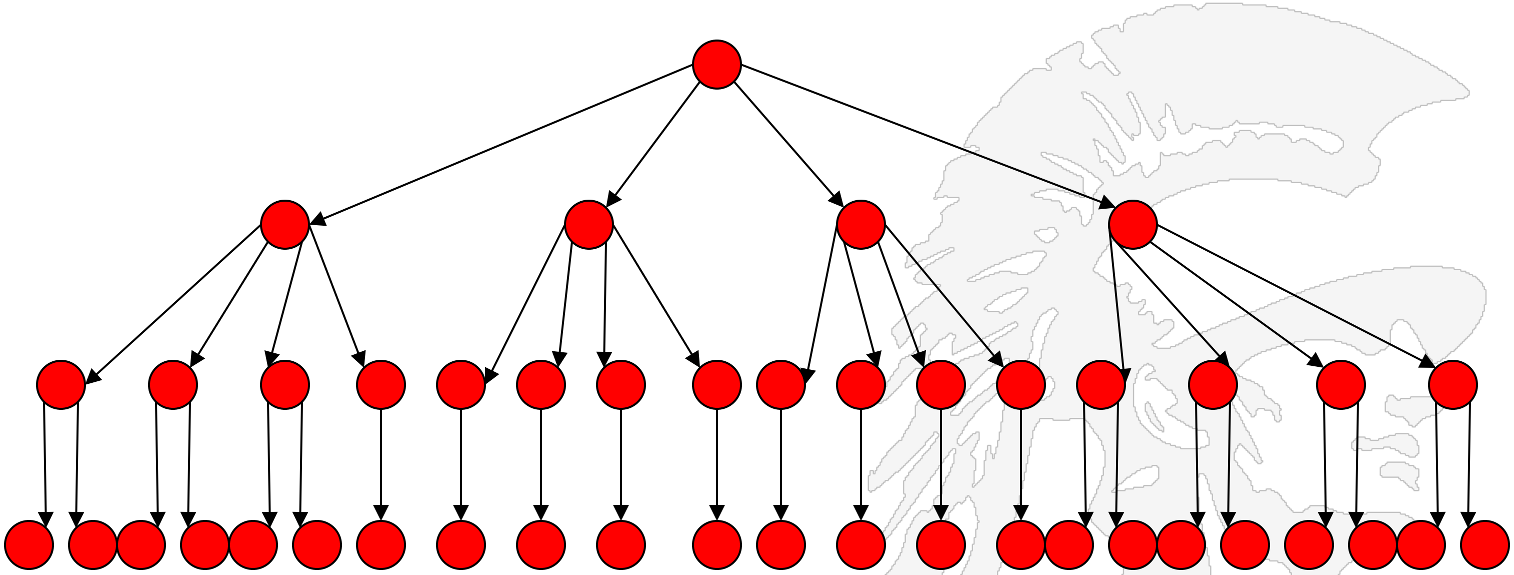
## Breadth-first Search



Examine all pages at level i before examining pages at level i+1

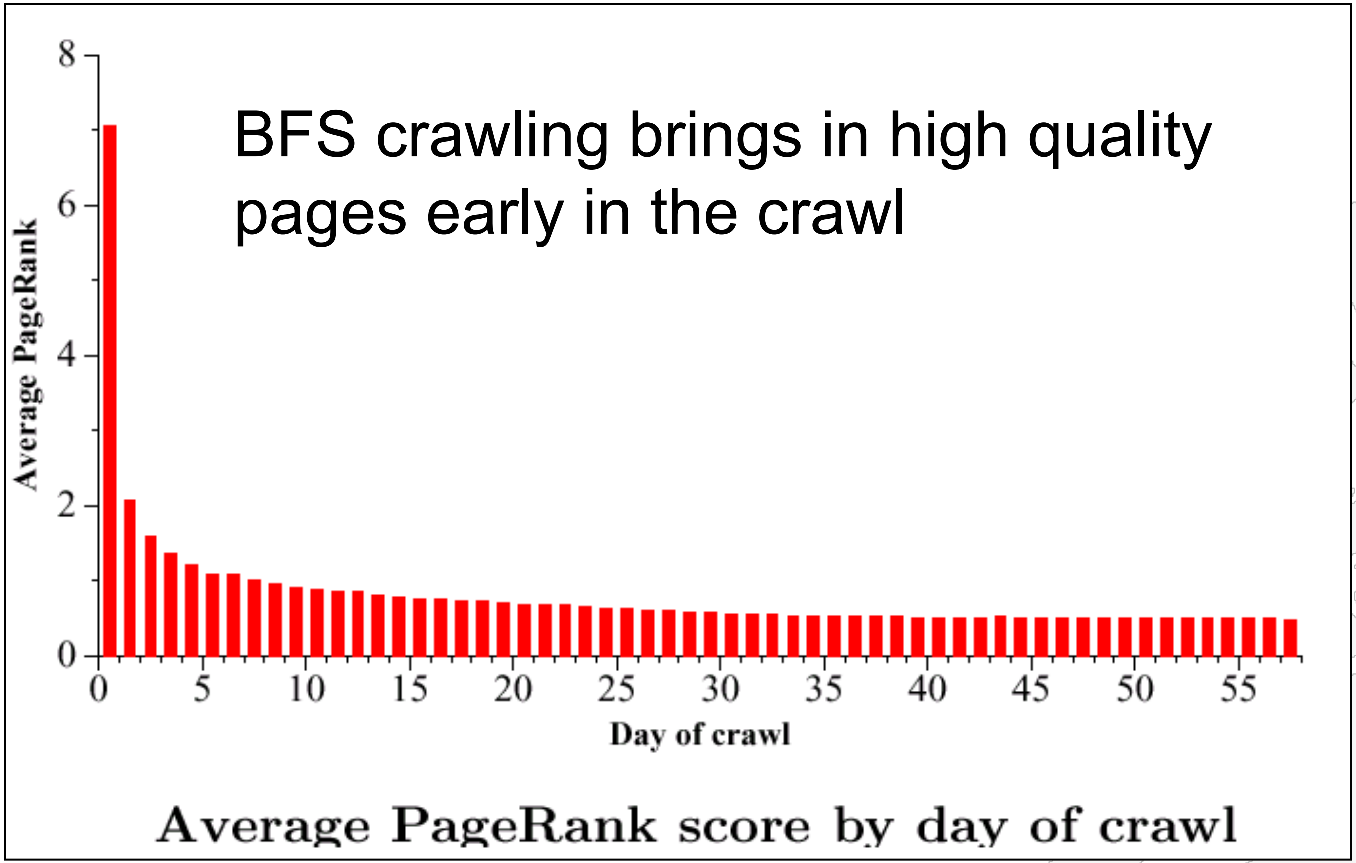
# Basic Search Strategies (cont)

## Depth-first Search



At each step move to a page down the tree

# Web Wide Crawl (328M pages) [Najo01]



BFS crawling brings in high quality pages early in the crawl

Average PageRank score by day of crawl

Page Rank is an algorithm developed by Google for determining the value of a page

# Crawling Algorithm – Version 2

```
Initialize queue (Q) with initial set of known URL’s.
Loop until Q empty or page or time limit exhausted:
    Pop a URL, call it L, from the front of Q.
    If L is not an HTML page (e.g. .gif, .jpeg, ….)
        continue the loop
    If L has already been visited, continue the 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 and store cached copy)
    Parse P to obtain list of new links N.
    Append N to the end of Q
End loop
```

# Queueing Strategy

- **How new links are added to the queue determines the 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; e.g.**
  - A document that changes frequently could be moved forward
  - A document whose content appears relevant to some topic can be moved forward
  - e.g. see Focused Crawling: A New Approach by S. Chakrabarti et al
  - https://www.sciencedirect.com/science/article/pii/S1389128699000523
- **One way to re-order the URLs on the queue is to:**
  - Move forward URLs whose In-degree is large
  - Move forward URLs whose PageRank is large
    - We will discuss the PageRank algorithm later

# Avoiding Page Duplication

- **A crawler must detect when revisiting a page that has already been crawled (Remember: the web is a graph not a tree).**
- **Therefore, a crawler must efficiently index URLs as well as already visited pages**
- **To determine if a URL has already been seen,**
  - Must store URLs in a standard format (discussed ahead)
  - Must develop a fast way to check if a URL has already been seen
- **To determine if a new page has already been seen,**
  - Must develop a fast way to determine if an *identical* page was already indexed
  - Must develop a fast way to determine if a *near-identical* page was already indexed

# Link Extraction

- **Must find all links in a page and extract URLs;**

```
var links = document.querySelectorAll("a");
for (var i = 0; i < links.length; i++) {
    var link = links[i].getAttribute("href");
    console.log(link); }
```

  – But URLs occur in tags other than <a>, e.g.

  – <frame src="site-index.html">, <area, href="…">, <meta>, <link>, <script>

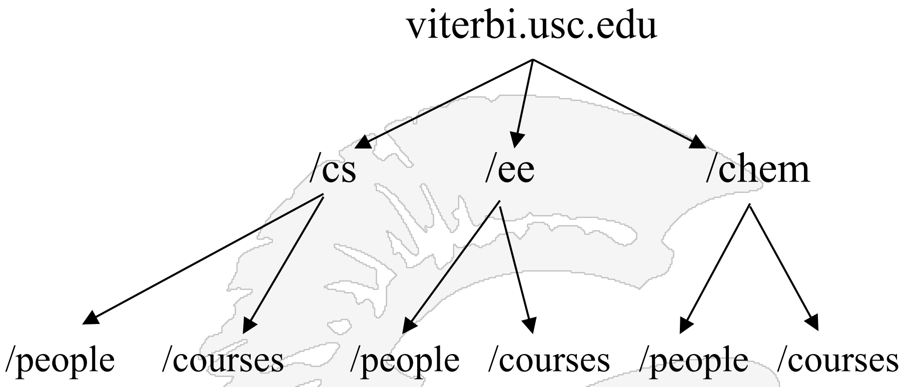
- **Relative URL's must be completed, e.g. using current page URL or <base> tag**
  – <a href="proj.html"> to http://www.myco.com/special/tools/proj.html
  – <a href="../outline/syllabus.html"> to http://www.myco.com/special/outline/syllabus.html
- **Two Anomalies**
  1. Some anchors don't have links, e.g. <a name="banner">
  2. Some anchors produce dynamic pages which can lead to looping
     <a href=http://www.mysite.com/search?x=arg1&y=arg2>

# Representing URLs

- **URLs are rather long, 80 bytes on the average, implying 1 trillion URLs will require 80 Terabytes**
  - Recently Google reported finding 30 trillion unique URLs, which by the above would require 2400 terabytes (or 2.4 petabytes) to store
1. **One Proposed Method: To determine if a new URL has already been seen**
   - First hash on host/domain name, then
   - Use a trie data structure to determine if the path/resource is the same as one in the URL database
2. **Another Proposed Method: URLs are sorted lexicographically and then stored as a delta-encoded text file**
   - Each entry is stored as the difference (delta) between the current and previous URL; this substantially reduces storage
   - However, restoring the actual URL is slower, requiring all deltas to be applied to the initial URL
   - To improve speed, checkpointing (storing the full URL) is done periodically

# Trie for URL Exact Matching

- Simplest (and worst) algorithm to determine if a new URL is in your set
  - `grep -i <search_url> <url_file>`
  - For *N* URLs and maximum length *K*, time is ***O(NK)***
- Characteristics of tries
  - They share the same prefix among multiple "words"
  - Each path from the root to a leaf corresponds to one "word"
  - *Endmarker* symbol, $, at the ends of all words
    - To avoid confusion between words with almost identical elements
      - Assume all words are $ terminated



If we store *N* URLs, each of maximum length *K*, in a binary search tree, then the search time is *O(K*log N); h*owever, using a trie, the search time is *O(K)*, at the expense of more storage

# Why Normalizing URLs is Important

- **For example, all the following URLs have the same meaning (return the same web page), but different hashes:**
  - **http://www.google.com**
  - **http://www.google.com/**
  - **https://www.google.com**
  - **www.google.com**
  - **google.com**
  - **google.com/**
  - **google.com.**

# Normalizing URLs (4 rules)

1. **Convert the scheme and host to lower case. The scheme and host components of the URL are case-insensitive.**
   - HTTP://www.Example.com/ → http://www.example.com/
2. **Capitalize letters in escape sequences. All letters within a percent-encoding triplet (e.g., "%3A") are case-insensitive, and should be capitalized. Example:**
   - http://www.example.com/a%c2%b1b → http://www.example.com/a%C2%B1b
3. **Decode percent-encoded octets of unreserved characters.** http://www.example.com/%7Eusername/ → http://www.example.com/~username/
4. **Remove the default port. The default port (port 80 for the "http" scheme) may be removed from (or added to) a URL. Example:**
   - http://www.example.com:80/bar.html →

   http://www.example.com/bar.html

- **See https://en.wikipedia.org/wiki/URL_normalization**

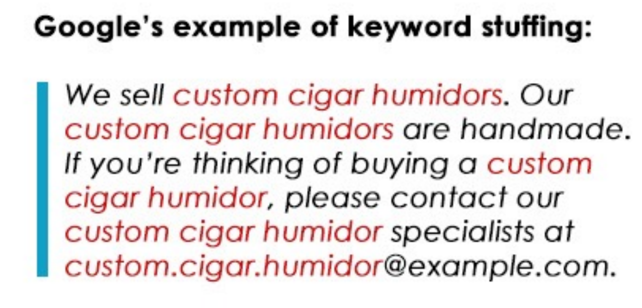
# Avoiding Spider Traps

- **A spider trap is when a crawler re-visits the same page over and over again**
- **The most well-known spider trap is the one created by the use of Session ID's**
  - J2EE, ASP, .NET, and PHP all provide session ID management
- **A Session ID is often used to keep track of visitors, and some sites puts a unique ID in the URL:**
  - An example is www.webmasterworld.com/page.php?id=264684413484654 (**Note** this URL doesn't exist).
    Each user gets a unique ID and it's often requested from each page.
    The problem here is when Googlebot comes to the page, it spiders the page and then leaves, it goes to another page and it finds a link to the previous page, but since it has been given a different session id now, the link shows up as another URL.

- One way to avoid such traps is for the crawler to be careful when the querystring "ID=" is present in the URL
- Another technique is to monitor the length of the URL, and stop if the length gets "too long"

# Handling Spam Web Pages

- The **first generation** of spam web pages consisted of pages with a high number of repeated terms, so as to score high on search engines that ranked by word frequency
  - Words were typically rendered in the same color as the background, so as to not be visible, but still count
- The **second generation** of spam used a technique called *cloaking*;
  - When the web server detects a request from a crawler, it returns a different page than the page it returns from a user request
  - The page is mistakenly indexed



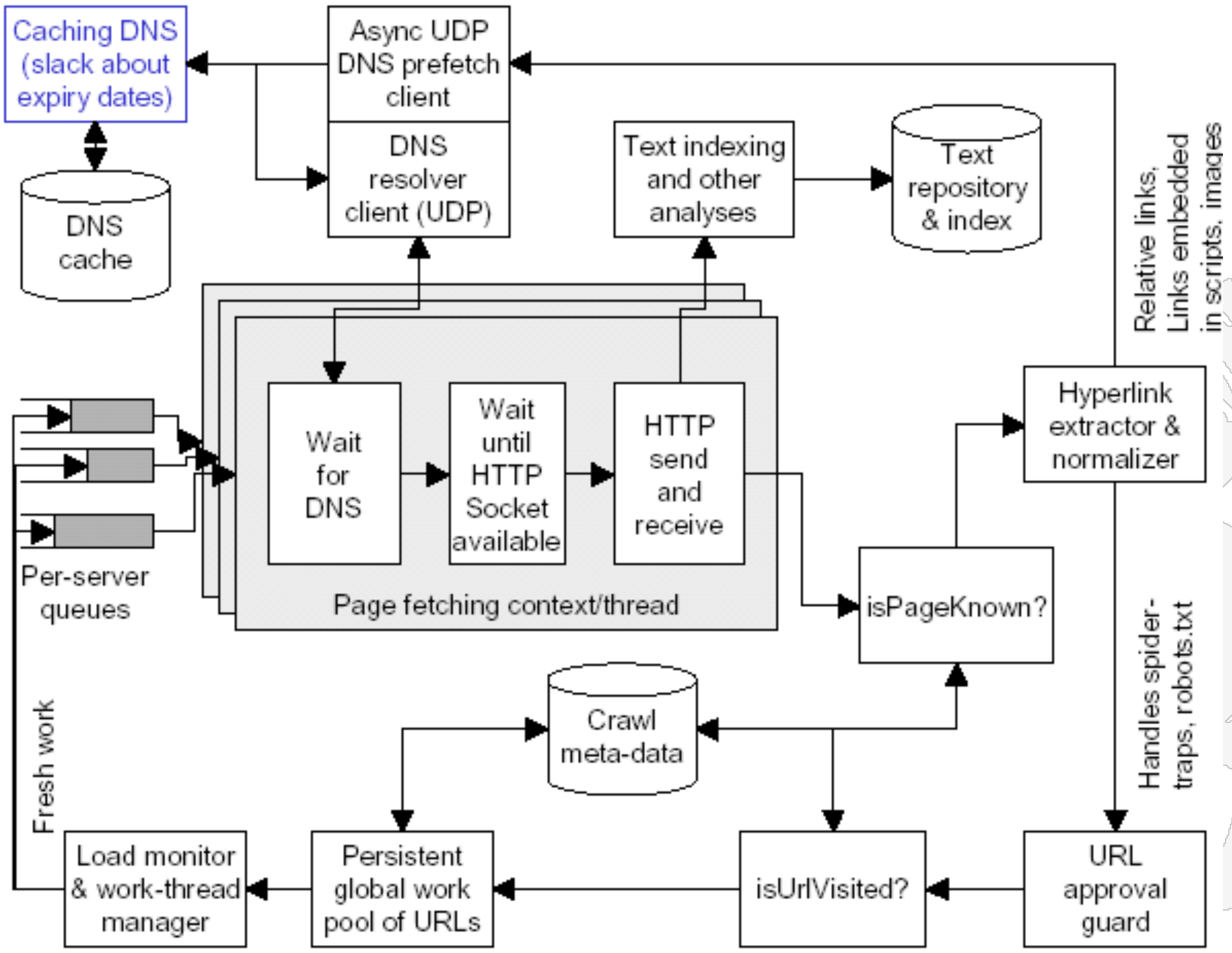
**Google's example of keyword stuffing:**

> *We sell custom cigar humidors. Our custom cigar humidors are handmade. If you're thinking of buying a custom cigar humidor, please contact our custom cigar humidor specialists at custom.cigar.humidor@example.com.*

- A third generation, called a doorway page, contains text and metadata chosen to rank highly on certain search keywords, but when a browser requests the doorway page it instead gets a more "commercially oriented" (more ads) page
- *Cloaking* and *doorway pages* are not permitted according to Google's webmaster suggestions

  See http://support.google.com/webmasters/bin/answer.py?hl=en&answer=66355

# The Mercator Web Crawler


Caching DNS (slack about expiry dates)
DNS cache
Async UDP DNS prefetch client
DNS resolver client (UDP)
Text indexing and other analyses
Text repository & index
Relative links, Links embedded in scripts, images
Wait for DNS
Wait until HTTP Socket available
HTTP send and receive
Page fetching context/thread
Hyperlink extractor & normalizer
Per-server queues
isPageKnown?
Handles spider-traps, robots.txt
Crawl meta-data
Fresh work
Load monitor & work-thread manager
Persistent global work pool of URLs
isUrlVisited?
URL approval guard

**The diagram points out all of the key elements of a crawler;**

**Notice**

1. The DNS caching server
2. Use of UDP for DNS
3. Load and thread monitor
4. Parallel threads waiting for a page to download

# Measuring and Tuning a Crawler

- **Measuring and tuning a crawler for peak performance eventually reduces to**
  - Improving URL parsing speed
  - Improving network bandwidth speed
  - Improving fault tolerance
- **More Issues (some of which are discussed ahead)**
  - Refresh Strategies: how often is the process re-started
  - Detecting duplicate pages
  - Detecting mirror sites
  - Speeding up DNS lookup (see previous slide)
  - URL normalization (discussed earlier)
  - Handling malformed HTML

# DNS caching, pre-fetching and resolution

– *A common operating system's implementation of DNS lookup is blocking: only one outstanding request at a time*; so

1. **DNS caching: build a caching server that retains IP-domain name mappings previously discovered**
2. **Pre-fetching client**
   - once a page is parsed,
     – immediately make DNS resolution requests to the caching server; and
     – if unresolved, use UDP (User Datagram Protocol) to resolve from the DNS server
3. **Customize the crawler so it allows issuing of many resolution requests simultaneously; there should be many DNS resolvers**

# Multi-Threaded Crawling

- **One bottleneck is network delay in downloading individual pages.**
- **It is best to have multiple threads running in parallel each requesting a page from a different host.**
  - a **thread** of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler.
  - In most cases, a thread is a component of a process.
  - Multiple threads can exist within the same process and share resources
- **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 coordinated crawlers with about 300 threads each,**
  - together they were able to download over 100 pages per second back in 2010
  - It is estimated that in 2021 Google downloads ~50,000 pages/seconds or 4billion+ in a day, see https://www.quora.com/How-many-pages-are-Google-bots-crawling-every-second

# Distributed Crawling Approaches

- **Once the crawler program itself has been optimized, the next issue to decide is how many crawlers will be running at any time**
- **Scenario 1:** A *centralized crawler* controling a set of parallel crawlers all running on a LAN
  - A *parallel crawler* consists of multiple crawling processes communicating via local network (sometimes called an intra-site parallel crawler)
- **Scenario 2:** A *distributed set of crawlers* running on widely distributed machines, with or without cross communication

# Distributed Model

- **If crawlers are running in diverse geographic locations, how do we organize them**
  - – By country, by region, by available bandwidth
  - – Distributed crawlers must periodically update a master index
  - – But incremental update is generally “cheap”
    - Why? Because
      - – a. you can compress the update, and
      - – b. you need only send a differential update

      both of which will limit the required communication

# Issues and Benefits of Distributed Crawling

- **Benefits:**
  - scalability: for large-scale web-crawls
  - costs: use of cheaper machines
  - network-load dispersion and reduction: by dividing the web into regions and crawling only the nearest pages
- **Issues:**
  - overlap: minimization of multiple downloaded pages
  - quality: depends on the crawl strategy
  - communication bandwidth: minimization

# Coordination of Distributed Crawling

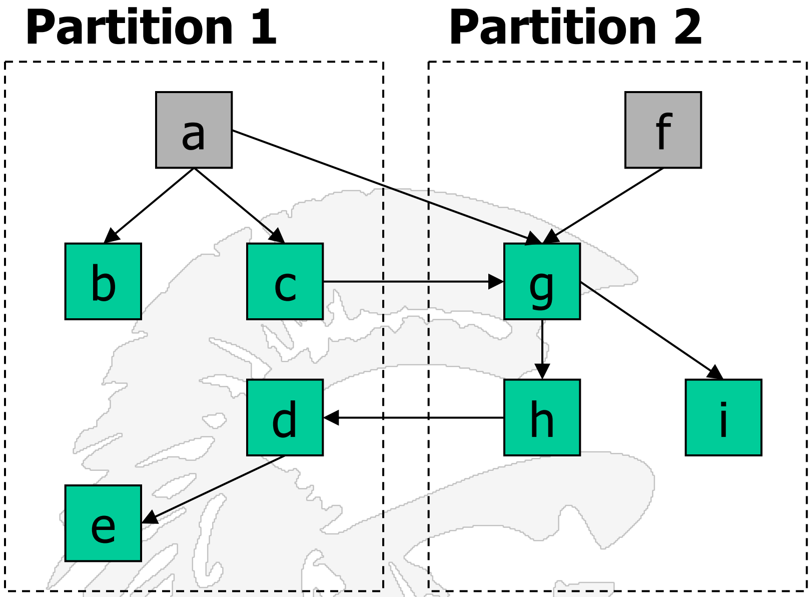
– **Three strategies**

1. **Independent:**
   - ► no coordination, every process follows its extracted links
2. **Dynamic assignment:**
   - ► a central coordinator dynamically divides the web into small partitions and assigns each partition to a process
3. **Static assignment:**
   - ► Web is partitioned and assigned without a central coordinator before the crawl starts

# Static Assignment

Links from one partition to another (inter-partition links) can be handled in one of three ways:

1. *Firewall mode*: a process does not follow any inter-partition link
2. *Cross-over mode*: a process also follows inter-partition links and possibly discovers also more pages in its partition
3. *Exchange mode*: processes exchange inter-partition URLs; this mode requires communication



# Classification of Parallel Crawlers

- **If exchange mode is used, communication can be limited by:**
  - Batch communication: every process collects some URLs and sends them in a batch
  - Replication: the *k* most popular URLs are replicated at each process and are not exchanged (previous crawl or on the fly)
- **Some ways to partition the Web:**
  - URL-hash based: this yields many inter-partition links
  - Site-hash based: reduces the inter partition links
  - Hierarchical: by TLD, e.g. .com domain, .net domain …
- **General Conclusions of Cho and Garcia-Molina**
  - Firewall crawlers attain good, general coverage with low cost
  - Cross-over ensures 100% quality, but suffer from overlap
  - Replicating URLs and batch communication can reduce overhead

# Freshness

- **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 *politeness policy* that states how to avoid overloading websites.
  - A *parallelization policy* that states how to coordinate distributed web crawlers.

# Keeping Spidered Pages Up to Date

- **Web is very dynamic: many new pages, updated pages, deleted pages, etc.**
- **Periodically check crawled pages for updates and deletions:**
  - Just look at LastModified indicator 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.**

# Implications for a Web Crawler

- **A *steady crawler* runs continuously without pause**
  - Typically search engines use multiple crawlers
- **When a crawler replaces an old version by a new page, does it do it "in-place" or "shadowing"**
  - Shadowing implies a new set of pages are collected and stored separately and all are updated at the same time
  - The above implies that queries need to check two databases, the current database and the database of new pages
  - Shadowing either slows down query processing or decreases freshness
- **Conclusions:**
  - running multiple types of crawlers is best
  - Updating in-place keeps the index current

# Cho and Garcia-Molina, 2000

- **Two simple re-visiting policies**
  - **Uniform policy:** This involves re-visiting all pages in the collection with the same frequency, regardless of their rates of change.
  - **Proportional policy:** This involves re-visiting more often the pages that change more frequently. The visiting frequency is directly proportional to the (estimated) change frequency.
- **Cho and Garcia-Molina proved the surprising result that, in terms of average freshness, the uniform policy outperforms the proportional policy in both a simulated Web and a real Web crawl.**
- **The explanation for this result comes from the fact that, when a page changes too often, the crawler will waste time by trying to re-crawl it too fast and still will not be able to keep its copy of the page fresh.**
- **To improve freshness, we should penalize the elements that change too often**

# Help the Search Engine Crawler Creating a SiteMap

- A sitemap is a list of pages of a web site accessible to crawlers
- This helps search engine crawlers find pages on the site
- XML is used as the standard for representing sitemaps
- Here is an example of an XML sitemap for a three page website

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
<url>
   <loc>http://www.example.com/?id=who</loc>
   <lastmod>2009-09-22</lastmod>
   <changefreq>monthly</changefreq>
   <priority>0.8</priority> </url>
<url>
   <loc>http://www.example.com/?id=what</loc>
   <lastmod>2009-09-22</lastmod>
   <changefreq>monthly</changefreq>
   <priority>0.5</priority> </url>
<url>
<loc>http://www.example.com/?id=how</loc>
<lastmod>2009-09-22</lastmod>
<changefreq>monthly</changefreq>
<priority>0.5</priority> </url>
</urlset>
```

Back in 2006 Google introduced the sitemap format; now Bing, Yahoo, and Ask also support sitemaps

See the Google, Bing, Yahoo, Ask announcement:
http://www.google.com/press/pressrel/sitemapsorg.html

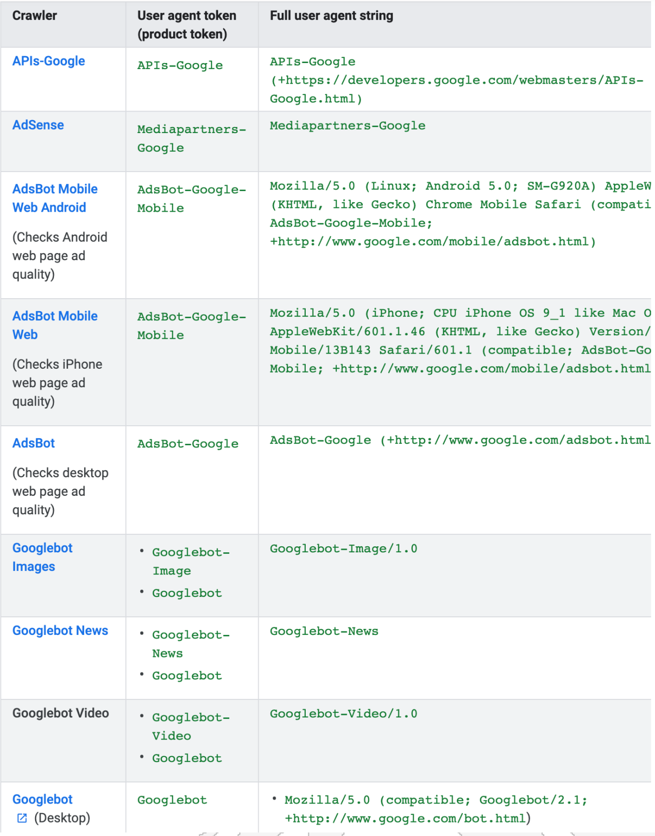
# General Sitemap Guidelines

- Use consistent, fully-qualified URLs. Google will crawl your URLs exactly as listed
- A sitemap can be posted anywhere on your site, but a sitemap affects only descendants of the parent directory
- Don't include session IDs from URLs in your sitemap.
- Sitemap files must be UTF-8 encoded, and URLs escaped appropriately.
- If you have two versions of a page, list in the sitemap only the one you prefer to appear in search results
- If you have alternate pages for different languages or regions, you can use hreflang to indicate the alternate URLs.
- 
```
<head>
 <title>Widgets, Inc</title>
  <link rel="alternate" hreflang="en-gb"
       href="https://en-gb.example.com/page.html" />
  <link rel="alternate" hreflang="en-us"
       href="https://en-us.example.com/page.html" />
  <link rel="alternate" hreflang="en”
      href="https://en.example.com/page.html" />
```

- There are many sitemap generator tools, e.g. https://slickplan.com/sitemap

# Google Crawlers

- Google now uses multiple crawlers
  - APIs-Google
  - AdSense
  - AdsBot Mobile Web Android
  - AdsBot Mobile Web
  - AdsBot
  - Googlebot Images
  - Googlebot News
  - Googlebot Video
  - Googlebot (desktop)
  - Googlebot (smartphone)
  - Mobile AdSense
  - Mobile Apps Android
  - Feedfetcher
  - Google Read Aloud



| Crawler | User agent token (product token) | Full user agent string |
|---|---|---|
| APIs-Google | APIs-Google | APIs-Google (+https://developers.google.com/webmasters/APIs-Google.html) |
| AdSense | Mediapartners-Google | Mediapartners-Google |
| AdsBot Mobile Web Android<br>(Checks Android web page ad quality) | AdsBot-Google-Mobile | Mozilla/5.0 (Linux; Android 5.0; SM-G920A) AppleW (KHTML, like Gecko) Chrome Mobile Safari (compati AdsBot-Google-Mobile; +http://www.google.com/mobile/adsbot.html) |
| AdsBot Mobile Web<br>(Checks iPhone web page ad quality) | AdsBot-Google-Mobile | Mozilla/5.0 (iPhone; CPU iPhone OS 9_1 like Mac O AppleWebKit/601.1.46 (KHTML, like Gecko) Version/ Mobile/13B143 Safari/601.1 (compatible; AdsBot-Go Mobile; +http://www.google.com/mobile/adsbot.html |
| AdsBot<br>(Checks desktop web page ad quality) | AdsBot-Google | AdsBot-Google (+http://www.google.com/adsbot.html |
| Googlebot Images | • Googlebot-Image<br>• Googlebot | Googlebot-Image/1.0 |
| Googlebot News | • Googlebot-News<br>• Googlebot | Googlebot-News |
| Googlebot Video | • Googlebot-Video<br>• Googlebot | Googlebot-Video/1.0 |
| Googlebot (Desktop) | Googlebot | • Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html) |

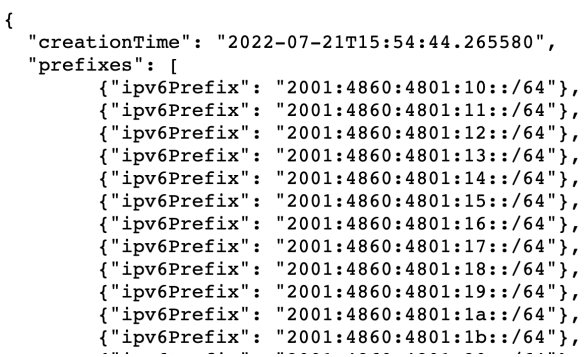
**For details see**
https://support.google.com/webmasters/answer/1061943?hl=en
**see also** Google's tool for checking how Googlebot sees your website
https://support.google.com/webmasters/answer/6066468?rd=2

# Google's Googlebot

- **Begins with a list of webpage URLs generated from previous crawls**
- **Uses Sitemap data provided by webmasters**
- **Many versions of Googlebot are run on multiple machines located near the site they are indexing**
- **Googlebot cannot see within Flash files, audio/video tracks, and content within programs**
- **Advice**
  - To prevent "File not found" in a website's error log, create an empty robots.txt file
  - To prevent Googlebot from following any links on a page, use "nofollow" meta tag
  - To prevent Googlebot from following an individual link, add "rel='nofollow'" attribute to the link

# To verify That The Visitor is Googlebot

- Method 1
  1. Run a reverse DNS lookup on the accessing IP address from your logs, using the host command.
  2. Verify that the domain name is either googlebot.com or google.com.
  3. Run a forward DNS lookup on the domain name retrieved in step 1 using the host command on the retrieved domain name.
  4. Verify that it's the same as the original accessing IP address from your logs.
- Method 2
  – match the crawler's IP address to the list of Googlebot IP addresses



```
{
  "creationTime": "2022-07-21T15:54:44.265580",
  "prefixes": [
        {"ipv6Prefix": "2001:4860:4801:10::/64"},
        {"ipv6Prefix": "2001:4860:4801:11::/64"},
        {"ipv6Prefix": "2001:4860:4801:12::/64"},
        {"ipv6Prefix": "2001:4860:4801:13::/64"},
        {"ipv6Prefix": "2001:4860:4801:14::/64"},
        {"ipv6Prefix": "2001:4860:4801:15::/64"},
        {"ipv6Prefix": "2001:4860:4801:16::/64"},
        {"ipv6Prefix": "2001:4860:4801:17::/64"},
        {"ipv6Prefix": "2001:4860:4801:18::/64"},
        {"ipv6Prefix": "2001:4860:4801:19::/64"},
        {"ipv6Prefix": "2001:4860:4801:1a::/64"},
        {"ipv6Prefix": "2001:4860:4801:1b::/64"},
```

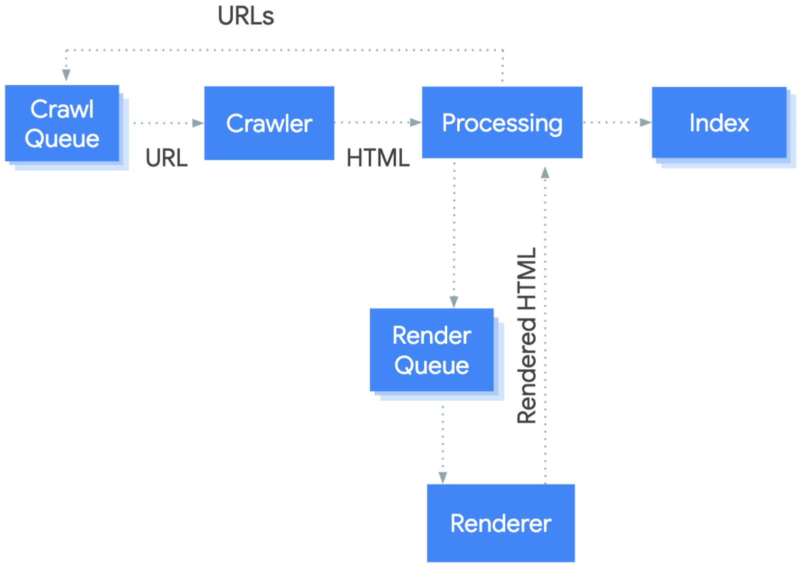
# Controlling How Often Googlebot Visits Your Site

- The term *crawl rate* means how many requests per second Googlebot makes to your site when it is crawling it: for example, 5 requests per second.
- You cannot change how often Google crawls your site, but if you want Google to crawl new or updated content on your site, you can request a recrawl; for details see

  https://developers.google.com/search/docs/advanced/crawling/ask-google-to-recrawl
- You can reduce the crawl rate by
  - returning pages with 500, 503 or 429 http status codes
  - Setting a new rate in the Search Console
  - A video discussing Google’s Crawl Status Report can be found here: https://support.google.com/webmasters/answer/9679690

# Googlebot is Really Chrome Re-packaged

- **Why:** browsers don't just render the DOM hierarchy of HTML, they include transformations via CSS and JavaScript, and for Googlebot to extract the most meaningful features from a web page it would be necessary to have access to these transformations
  - Therefore Googlebot must understand and execute JavaScript code
- **Conclusion:** Googlebot and Chrome share a great deal of code
- **Googlebot processes web pages with JavaScript in 3 phases**
  1. Crawling – processing all links
  2. Rendering – executing JS and then looping back to 1
  3. Indexing
- As of 2019 Googlebot runs the latest Chromium rendering engine
- **Note: Server-side rendering saves Googlebot from rendering the page**


URLs
Crawl Queue
URL
Crawler
HTML
Processing
Index
Render Queue
Rendered HTML
Renderer