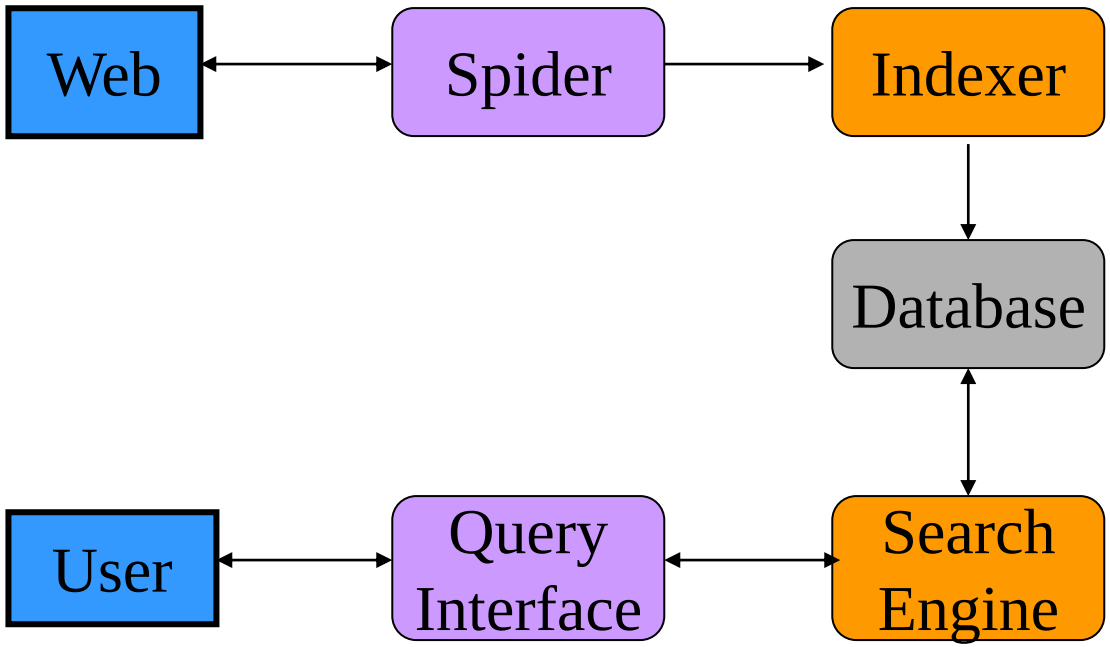
Distributed Systems

Web Searching Technologies

Some helpful web sites

- A history of search engines:
  - http://www.wiley.com/legacy/compbooks/sonnenreich/webdev/history.html
- Open source search engines written in Java:
  - http://java-source.net/open-source/search-engines
- Robots:
  - http://www.robotstxt.org/wc/robots.html

Search Engines Architecture



Spiders (Web Crawler)

- Automatically Retrieve web pages
  - Start with an URL, retrieving the web page
  - Find all URLs on the web page and recursively retrieve not-yet searched URLs
- Algorithmic Issues
  - Efficiency and Quality: how to quickly gather as many useful web pages as possible?
  - How to choose the next URL?
  - How to avoid overloaded sub-networks?

Indexer

- Select terms (keywords) to index a document
  - need co-operation from authors of webpage through Meta tags (next to Title tag in html files) to indicate specific terms to index

    <META name="keywords" content="retrieval, sensor net, cloudcomp">
- Algorithmic issues:
  - How to choose terms/phrases to index documents, such that user queries can be returned accurately and efficiently
  - How to index documents encoded in different types (multimedia data retrieval)

Database

- Structured data (see Bigtable for Google)
  - Bigtable: A Distributed Storage System for Structured Data, CACM, Jun 2008
- Parallel search and distributed storage
- Algorithmic issues:
  - Data partitioning and distributed storage: store petabytes of data across thousands of servers
  - Real-time data retrieval and update (short latency)
  - Fault-tolerance (data redundancy)

Search Engine

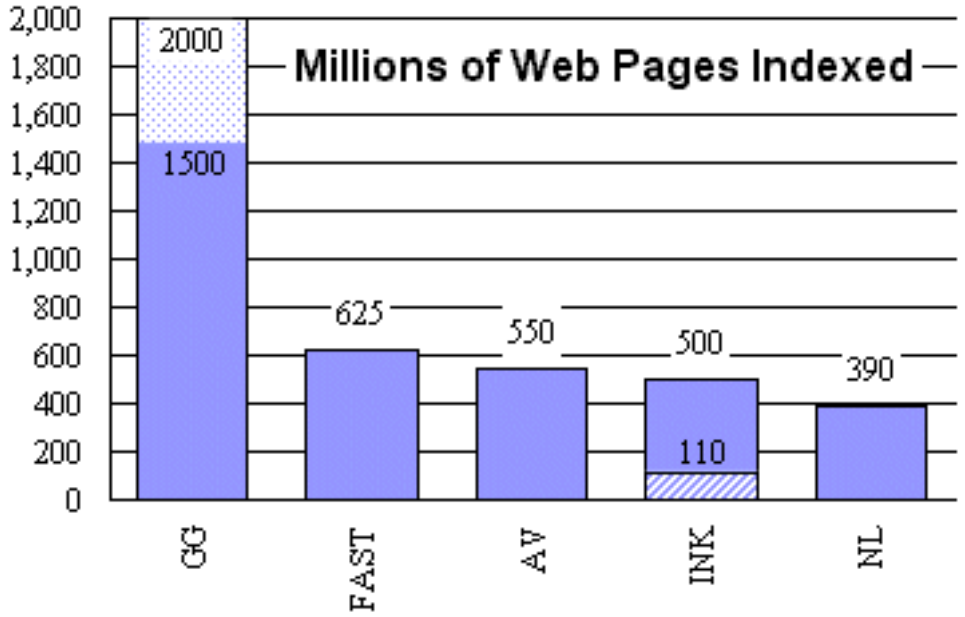
- Return the most relevant documents for queries
- Algorithmic Issues:
  - Parallel search (e.g., map-reduce techniques)
  - Relevance analysis

Major Products of Search Engines

- Alta vista
- Google
- Goto
- Lycos
- Yahoo
- …

Search Engine Sizes (Dec 11, 2001)

Estimated total web pages ~ 2 billion



Shaded area for GG and INKTOMI are pages indexed but not visited

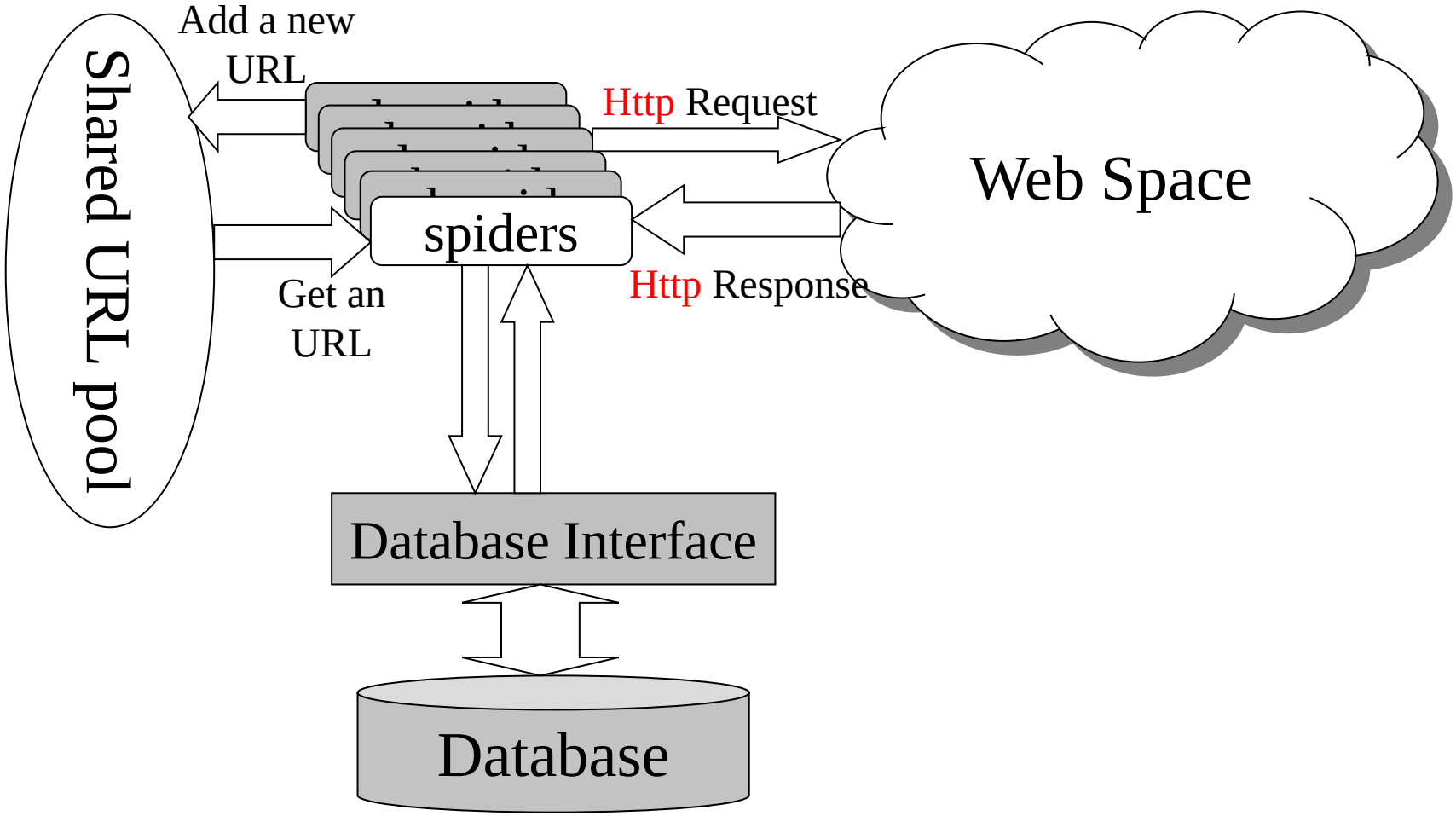
AV	**Altavista**
FAST	**FAST**
GG	**Google**
INK	**Inktomi**
NL	**Northern Light**

SOURCE: SEARCHENGINEWATCH.COM

Difficulties for Internet Information Retrieval

- Diversified Users (from layman to computer nerds).
  - Can we develop an evolving system that adapts to user?
- Ambiguity of language expression
  - This is an important issue due to varieties of data on the Internet
  - How do we collect and apply user profiling techniques to resolve it?
- False Information
  - Phising, Spam, Misleading Advertisement

Spider Architecture



Spider Programming: start with HTTP…

- Spiders use HTTP protocol to retrieve data…

```
public class HttpClientGet {
    public static void main(String[] args) throws Exception {
        String serverUrl = "www.cs.cityu.edu.hk";
        Socket s = new Socket(serverUrl, 80);
        BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));
        PrintStream out = new PrintStream(s.getOutputStream());
        out.println("GET /~jia/index.html HTTP/1.0");
        out.println(); // “GET” must be followed by a blank line.
        String line;
        while ((line = in.readLine()) != null) {
            System.out.println("> " + line);
        }
        s.close();
    }
}
```

A Simple HTTP Server

```
public class HttpServer {
   public static void main(String argv[]) throws Exception {
      ServerSocket serverSock = new ServerSocket(8632); // open server socket at port
      while (true) {
               Socket new_s = serverSock.accept();
               serveRequest(new_s);
               client.close(); }
   }
   public static void serveRequest(Socket s) throws Exception {
      PrintStream out = new PrintStream(s.getOutputStream());
      BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));
      String httpCmd = in.readLine(); // The HTTP Request is httpCmd;
      String[] httpArgs = httpCmd.split(" ");
      if (httpArgs.length < 2 || !httpArgs[0].equalsIgnoreCase("GET")) {
         out.println("400 Bad Request.");  return; }
      String fContent = readFileContent(httpArgs[1]);
      if (fContent == null) out.println("404 File Not Found");
               else out.println(fContent);
   }
```

Spider in Action!

- Initialize the queue with URLs of known pages
- Repeat doing:
  a) Take an URL from queue
  b) Fetch the web page and store/index the relevant information in the page
  c) Extract URLs from the page and add them to the queue
- Assumption: The web is well linked.

A Simple Spider

```
public class SimpleSpider {
    private Queue linkQueue=new LinkedList();
    private LinkExtractor extractor=new LinkExtractor();
    protected boolean processLink(String link) throws IOException{
        System.out.println(link);
        extractor.parse((new URL(link)).openStream()); // parse the page content…..
        Iterator it=extractor.getExtLinkIterator();  // extract all URL links in this page
        while (it.hasNext())
            linkQueue.add(it.next()); // add next link to the queue
    }
    public void work(String iLink){
        linkQueue.add(iLink); // add this link to the queue
        for (int i = 0; i <20; ++i){  // limit max # of webpages to visit
            String link=(String) linkQueue.poll(); // get a link from the queue
            if (link==null) break; // no more links
            processLink(link); }
    }
    public static void main(String[] args){
        SimpleSpider spider=new SimpleSpider();
        spider.work("http://www.cs.cityu.edu.hk"); // start from this link
    }
```

A Simple Spider (cont'd)

```
public class LinkExtractor {
   private HtmlHrefParser refParser = new HtmlHrefParser(); // HtmlHrefParser is system defined
   public void parse(InputStream stream) throws IOException {
      if (stream==null) throw new IllegalArgumentException("Illegal Argument");
      refParser.reset();
      ParserDelegator pd = new ParserDelegator();
      pd.parse(new InputStreamReader(stream), refParser, true);
   }

   public Iterator getExtLinkIterator() {
      return refParser.extLinks.iterator();  // return list of ext URL links
   }

   public Iterator getLocalLinkIterator() {
      return refParser.localLinks.iterator();
   }
}
```

Cautions about using spiders

- It may add unexpected amount of traffic if poorly designed
- Be responsible for your actions
- Test it locally before running it over the Internet
- Follow the standard guidelines
  - www.robotstxt.org/wc/guidelines.html

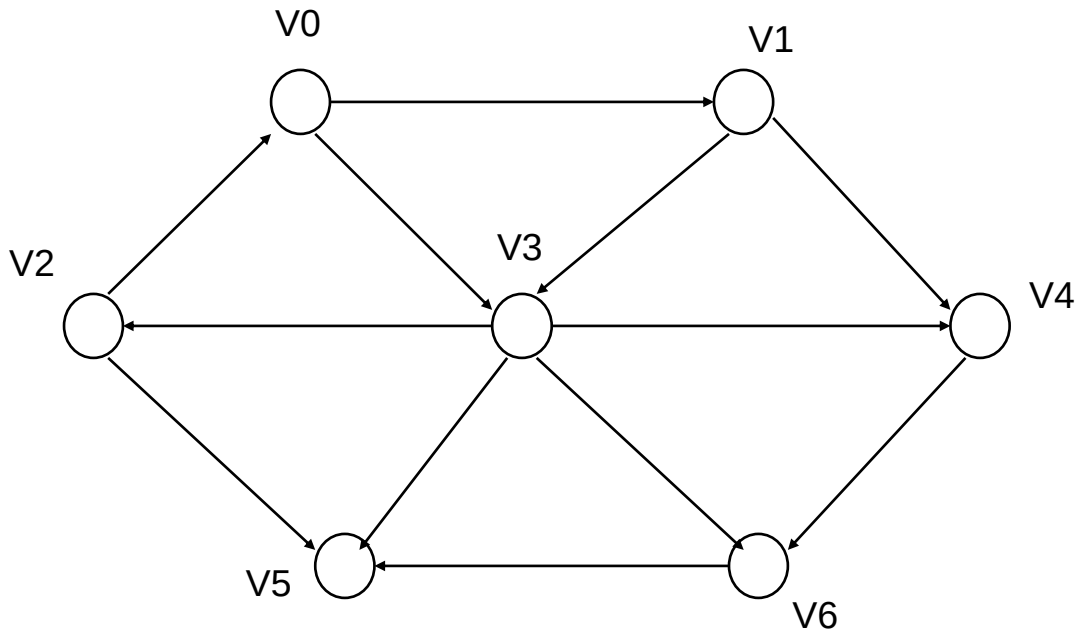
Web Data Mining: Digraph

- Nodes: web pages (its address is URL)
- Directed Edges: hyperlinks from one web page to another
- Content of a node: the content of the corresponding web page
- Dynamic nature of the digraph:
  - For some nodes, there are outgoing edges we don't know yet.
    - Nodes not yet processed (don't know its content)
    - New edges (hyperlinks) may have added to some nodes
  - For all the nodes, there are some incoming edges we don't yet know.

Construct Web Digraph

- To construct the web digraph, one needs
  - a spider to automatically collect URLs
  - a graph class to store information for nodes (URLs) and edges (hyperlinks)
- The whole digraph (URLs, HyperLinks) is huge:
  - 162,128,493 hosts (2002 Jul data)
  - 1,173,109,925 users
    - http://www.internetworldstats.com/stats.htm
  - One may need graph algorithms with secondary memories
  - Google uses thousands of workstations.

An Example Digraph

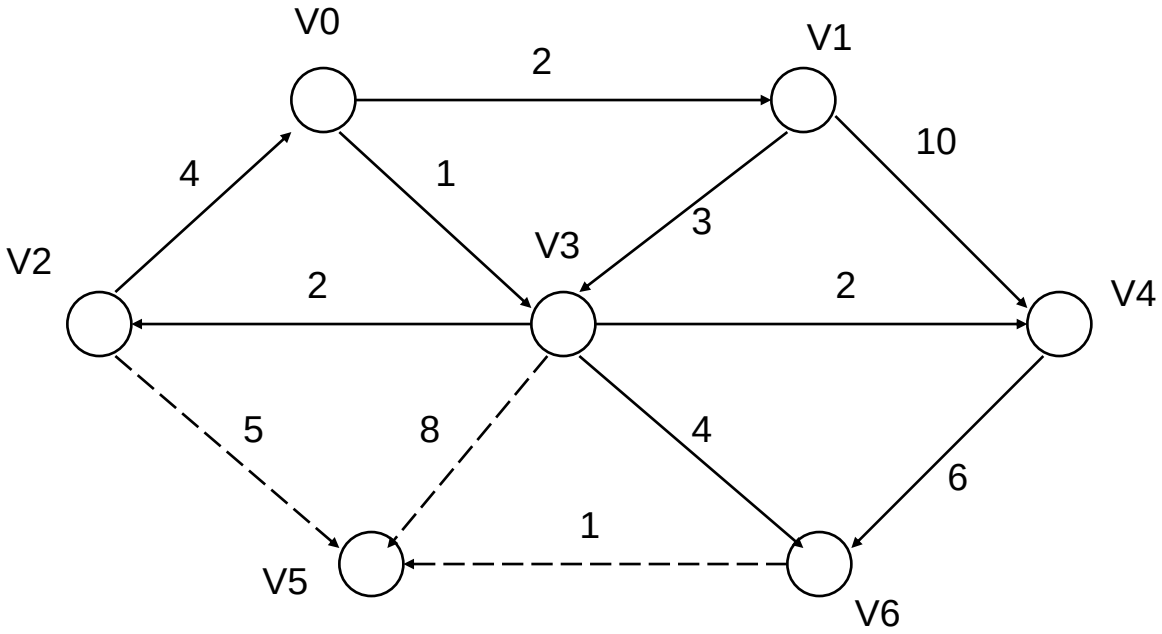


An ordinary digraph H with 7 vertices and 12 edges

Partial Map of Web Digraph

- Partial map may be sufficient for some purposes
  - e.g., we are often interested in a small portion of the whole Internet
- A partial map may be constructed within the memory space for an ordinary PC, which allows fast performance.
- With partial map, we may not be able to have all necessary information for our purpose
  - e.g., back links to a URL

An Example of a Partial Digraph



A partial digraph H: node v5 is not yet explored. We don't know that it has outgoing edges from v5 though we know its existence (by its URL).

Unknown and Dynamic Factors of Web Structure

- There is no central control of the hyperlinks and each search engine can only map a fraction of the whole web space
- Hyperlinks are dynamically added and deleted by individual web page authors
- No web page knows its incoming hyperlinks
- Some web pages are not documented by any search engine

Some Useful Functions on Digraph

- Back_Link (*the_url*):
  - find out all the urls that point to *the_url*
- Shortest_Path (*url1*, *url2*):
  - return the shortest path from *url1* to *url2*
- Maximal_Clique (*url*):
  - return a maximal clique that contains *url*
- In-Degree(*url*)
  - return the number of links that point to the *url*.
  - represents, to some extent, its popularity. The more a web page is pointed to by web pages, the more web authors are interested in

- Related data structures, algorithms and websites
  - http://webla.sourceforge.net/javadocs/pt/tumba/links/WebGraph.html
  - http://www.cs.princeton.edu/introcs/45graph/Digraph.java.html
  - http://www.quadcap.com/products/qed/docs/source/_di_graph_8java-source.html
  - http://www.cs.ucsb.edu/~kris/Research/agl/doc/agl2/Digraph.html

Back Links of Digraph

- Hyperlinks on the web are forward-pointing. Web pages do not know the hyperlinks pointing back to them
  - authors of web pages can freely link to other documents in the cyberspace without consent from their authors
- Back link information is important
  - in scientific literature, SCI (Scientific Citation Index) is an important index for judgment of academic value of one academic article
- It is not easy to find all the back links

Collect Back Link Information Via HTTP

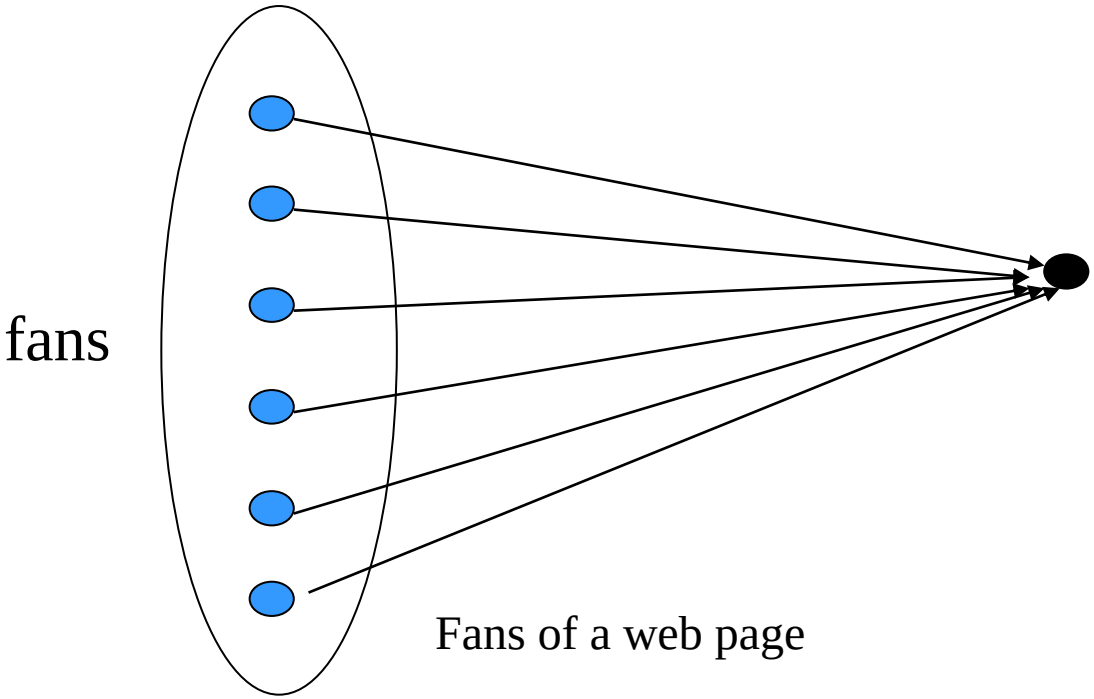
Section 10. Header field definition of Http1.0

subsection 10.13 Referer:

- Referer field (in request-header) allows the client (browser) to specify the source address (URI) from which the Request-URI was obtained.
- This allows a server to generate lists of back-links to resources for interest, logging, optimized caching, etc.
- The Referer field is empty if the request is NOT from a webpage, e.g., from the URI field in browser (i.e., the source does not have an URI).
- For details, see
  - http://www.ietf.org/rfc/rfc1945.txt

An Application of BackLink: FAN

- Fans of a web page has a link pointing to the web page.
- A user would add a link to a webpage after he accessed and viewed the content of the page.



FAN: An Indicator of Popularity

- The more fans a web page has, the more popular it is.
- SCI (Scientific Citation Index), for example, is a well known method to rate the importance of a research paper.
- However, some very popular web pages are so well known that people don't need to put them in their web pages, such as google.com, yahoo.com, etc.
- Another controversial argument about backlink popularity:
  - a page pointed by some important web pages, compared with another pointed by many un-important pages?

WebPage Ranking

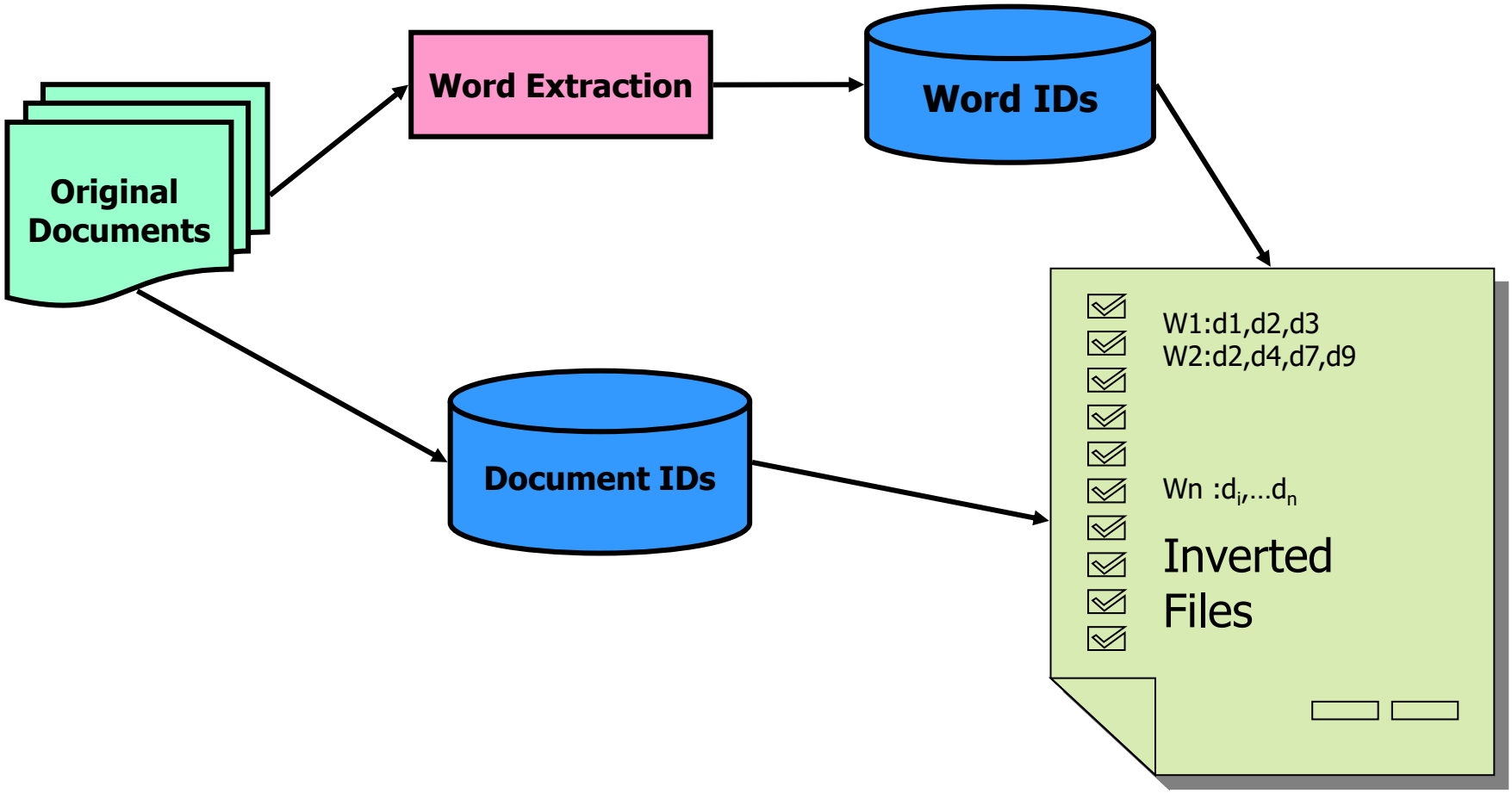
- Two factors for ranking a web page:
  - The rank of web pages pointing to it
    - The high the better
  - The number of outgoing links in the web pages pointing to it
    - The less the better
- Page Rank of page *A*, $PR_{(A)}$, is calculated as follows:
  - Suppose page *A* has pages $T_1 ... T_n$ that point to it (i.e., back links).
  - *d* is a factor between 0 and 1 (usually set to 0.85).
  - *C*(*T*) is the number of outgoing links of page *T*.

$$PR_{(A)} = (1-d) + d \times \left(\frac{PR_{(T_1)}}{C_{(T_1)}} + \ldots + \frac{PR_{(T_n)}}{C_{(T_n)}} \right)$$

Page Rank Calculation

- The definition of $PR_{(A)}$ is cyclic, i.e., the rank of a page depends on the ranks of other pages. However, page ranks can be computed by a simple iterative algorithm.
- Page ranks can be efficiently calculated by using web digraph. Page ranks for 26 million pages can be computed in a few hours on a medium size workstation.
- Page ranks help search engine to rank the search results according to their popularity or importance (no consideration of relevance of search results here).

Inverted File: indexing for search



Term-Document indexing

1. Map the file names (documents) to file IDs

Consider the following Original Documents

d1	The Department of Computer Science was established in 1984.
d2	The Department launched its first BSc(Hons) in Computer Studies in 1987.
d3	followed by the MSc in Computer Science which was started in 1991.
d4	The Department also produced its first PhD graduate in 1994.
d5	Our staff have contributed intellectually and professionally to the advancements in these fields.

Remove Stop Words

2. Remove stop words (“in”, “the”, “in”, ….) that are not meaningful for search

d1	***The*** Department ***of*** Computer Science ***was*** established ***in*** 1984.
d2	***The*** Department launched ***its first*** BSc(Hons) ***in*** Computer Studies ***in*** 1987.
d3	followed ***by the*** MSc ***in*** Computer Science ***which was*** started ***in*** 1991.
d4	***The*** Department ***also*** produced ***its first*** PhD graduate ***in*** 1994.
d5	***Our*** staff ***have*** contributed intellectually ***and*** professionally ***to the*** advancements ***in these*** fields.

Terms and Documents

3. Make lowercase (option), delete numbers (option) , insert both singular and plural forms of nouns, different tenses of verbs, adjective & adverb of the same word, for search.

d1	department computer science established
d2	department launch bsc hons computer studies
d3	follow msc computer science start
d4	department produce phd graduates
d5	staff contribute intellectually professionally advancement fields

Build Inverted File

4. Build Inverted File

1)Build the inverted file mapping from words to document IDs. You may insert information about the number of times a word appears in a document or the position it appears for more accurate search.

2)Sort the keywords for quick search.

Words	Documents	Words	Documents
department	d1,d2,d4	produce	d4
computer	d1,d2,d3	phd	d4
science	d1,d3	graduate	d4
establish	d1	staff	d5
launch	d2	contribute	d5
bsc	d2	intellectual	d5
hons	d2	professional	d5
studies	d2	advance	d5
follow	d3	field	d5
msc	d3		
start	d3		

Searching Inverted File

- Binary Search
  - Using in the small scale
- Create thesaurus and combining techniques such as:
  - Hashing
  - B^+tree
  - Pointer to the address in the indexed file

Boolean Model for Search

- Document representation: full text or a set of key-words (contained in the text or not)
- Query representation: logic operators, query terms, query expressions, e.g.,
  - A && B && (C || D)
- Searching: using inverted file and set operations to construct the result set